Resource-Constrained Learning and Inference for Visual Perception

Mengtian Li

Ph.D. Thesis

CMU-RI-TR-22-20
May 10, 2022

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Deva Ramanan, *Chair*
Martial Hebert
Mahadev Satyanarayanan
Raquel Urtasun, *Waabi & University of Toronto*
Ross Girshick, *Meta AI Research*

Thesis proposal submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics

Abstract

We have witnessed rapid advancement across major computer vision benchmarks over the past years. However, the top solutions' hidden computation cost prevents them from being practically deployable. For example, training large models until convergence may be prohibitively expensive in practice, and autonomous driving or augmented reality may require a reaction time that rivals that of humans, typically 200 milliseconds for visual stimuli. Clearly, vision algorithms need to be adjusted or redesigned when meeting resource constraints. This thesis argues that we should embrace resource constraints into the first principles of algorithm designs. We support this thesis with principled evaluation frameworks and novel constraint-aware solutions for both the cases of training and inference of computer vision tasks.

For evaluation frameworks, we first introduce a formal setting for studying training under the non-asymptotic, resource-constrained regime, i.e., budgeted training. Next, we propose streaming accuracy to evaluate latency and accuracy coherently with a single metric for real-time online perception. More broadly, building upon this metric, we introduce a meta-benchmark that systematically converts any single-frame task into a streaming perception task.

For constraint-aware solutions, we propose a budget-aware learning rate schedule for budgeted training, and dynamic scheduling and asynchronous forecasting for streaming perception. We also propose task-specific solutions, including foveated image magnification and progressive knowledge distillation for 2D object detection, multi-range pyramids for 3D object detection, and future object detection with backcasting for end-to-end detection, tracking and forecasting.

We conclude the thesis with discussions on future work. We plan to extend streaming perception to include long-term forecasting, generalize our foveated image magnification to arbitrary spatial image understanding tasks, and explore multi-sensor fusion for long-range 3D detection.

Acknowledgement

First and foremost, I am deeply grateful to my Ph.D. advisor, Deva Ramanan. I think working with Deva was the most fortunate thing during my Ph.D. I would like to thank him for being the nicest advisor possible. I still remember his gentle knock on the door, asking, ”what can I do for you” before paper deadlines. I would like to thank Deva for his radiating warmth for research. We often chat for hours about research, and it’s such an enjoyable experience to bounce ideas and passion between each other. I would also like to thank Deva for his immense wisdom. After all my Ph.D. years, there is still much left that I can learn from him, not only about how to do better research, but also about how to be a better person. I will forever cherish the lessons I learned from him and be forever grateful.

I wish to show appreciation to the incredible minds I have collaborated with during my Ph.D., without whom I could not accomplish such a large volume of work and maintain high qualities. First, I would like to thank Yu-Xiong Wang and James Hays, who I have been continuously working with over the past years. Next, I would like to thank all the first authors in my projects for their hard work and dedication: Chittesh Thavamani, Kartikeya Sharma, Neehar Peri, Ziqi Pang, Shengcao Cao, Shubham Gupta, Jeet Kanjani and Nithin Kumar. Additionally, I would like to thank my other collaborators for their valuable ideas, insightful discussions, and excellent writings: Benjamin Wilson, Aljosa Osep, Jonathon Luiten, Liangyan Gui, Ersin Yumer, Zhe Lin, Radomír Měch, Shu Kong, Kris M Kitani, Laura Leal-Taixé, Nicolas Cebron, and Francesco Ferroni.

I would like thank my Ph.D. thesis committee members, Martial Hebert, Mahadev (Satya) Satyanarayanan, Raquel Urtasun, and Ross Girshick, for their critical questions and insightful comments. I am incredibly grateful for Martial’s advice and help on my research and my Ph.D. application. The committee members have all produced milestone work in their respective fields of expertise. Their work inspired my research, and it is my true honor to have them judge my work. I would also like to thank the ECCV 2020 award committee for seeing the values and the potential of our *Towards Streaming Perception* work.

I would like to thank our administrative assistants, Suzanne Lyons Muth, Hadley Pratt, Barbara Jean (B.J.) Fecich and Stephanie Matvey, for making the Ph.D. life easier for us. I would also like to thank our Trinity GPU cluster for making large-scale experiments possible in an academic environment.

My appreciation goes to my talented labmates at CMU (not relisting those already listed above): Peiyun Hu, Aayush Bansal, Achal Dave, Rohit Girdhar, Ravi Teja Mullapudi, Gengshan Yang, Jason Zhang, William Qi, Olga Russakovsky, Pavel Tokmakov, Ishan Nigam, Tarasha Khurana, Aaron Huang, Cindy Wu, Chonghyuk (Andrew) Song, Zhiqiu Lin, Haithem Turki, Haochen Wang, Jessica Lee, and Kangle Deng. Working side-by-side and discussing research or career questions have been enjoyable experiences, and I have learned much and received much help from them. I will miss the heated discussion during lab meetings, closed-door chats, and gossip in EDSH 224 and 101.

I would like to extend my sincere thanks to my great friends at CMU, who made my Ph.D. years a pleasant journey (not relisting those already listed above): Allan Wang, Chia Dai, Anqi Yang, Chen-Hsuan Lin, Leo Keselman, Xingyu Lin, Yufei (Judy) Ye, Donglai Xiang, Lin Ma, Ye Yuan, Yan (Ryan) Xu, Yubo Zhang, Senthil Purushwalkam, Dinesh Reddy, Shumian Xin, Minyoung (Jacob) Huh, Hao Zhang, Luona (Luna) Yang, Yu Zhao, Maria Jahja, Ming-Fang (Allie) Chang, Nadine Chang, Helen Jiang, Chao Cao, Xianyi Cheng, Shuyan Zhou, Yunze Man, Erica Weng, and many others. I will miss those chats, those trips, and those games. Half of my Ph.D. years were overshadowed by the COVID-19 pandemic, and I am extremely grateful for their company and help in these challenging times.

Finally, I would like to thank my parents, Jinsong Li and Wen Cao, for their unconditional love and support behind every critical decision in my life, without which this thesis would not be possible.

This thesis was supported by the CMU Argo AI Center for Autonomous Vehicle Research.

Contents

Chapter 1

Introduction

Computer vision often draws inspiration from biological vision. Through billions of years of evolution, the animal kingdom witnesses a plethora of biological vision systems. From an array of nondirectional photoreceptors to eyes with lenses, corneas and irises that enable high-resolution vision, the biological vision system is vastly diverse. Yet none of them have an omnipotent vision system, as such a system *must* meet resource constraints necessary for *survival*, *e.g.*, maintaining a low energy consumption. Studies have shown that the temporal resolution of biological vision varies greatly among species, and has close ties to the metabolic rate and the living environment [85]. This suggests that biological vision adapts to resource constraints.

This thesis argues that the machine vision should also adapt to resource constraints. First and foremost, meeting resource constraints is the necessary condition for designing a practical visual perception system. A deep learning model will not run if it exceeds the hardware memory capacity, a robot will crash if its perception stack is too slow, and a cell phone's battery will quickly drain if the camera enhancement algorithm is not energy efficient. One may consider resource constraints as a pure engineering concern, however, when we embrace resource constraints into our first principles, we might arrive at novel solutions and eventually reaching a higher level of artificial intelligence. In neural architecture search, one successful strategy is to first search the most energy-efficient cell structure and then repeat it multiple times to form the whole architecture [204]. Although energy was the search criterion, the final model usually ends up with higher overall accuracy for various tasks. In fact, "bounded rationality" is recognized as a fundamental issue in computer science, economics and philosophy in 1955 by the Turing Award and Nobel Prize winner Herbert A. Simon [197]. In his seminal work, he proposed a model for studying organisms making decisions with limited computational ability. This thesis restricts the scope to computer vision and presents a series of projects on resource-constrained learning and inference to demonstrate the importance of considering resource constraints and how to design budget-aware and efficient vision algorithms.

1.1 Overview

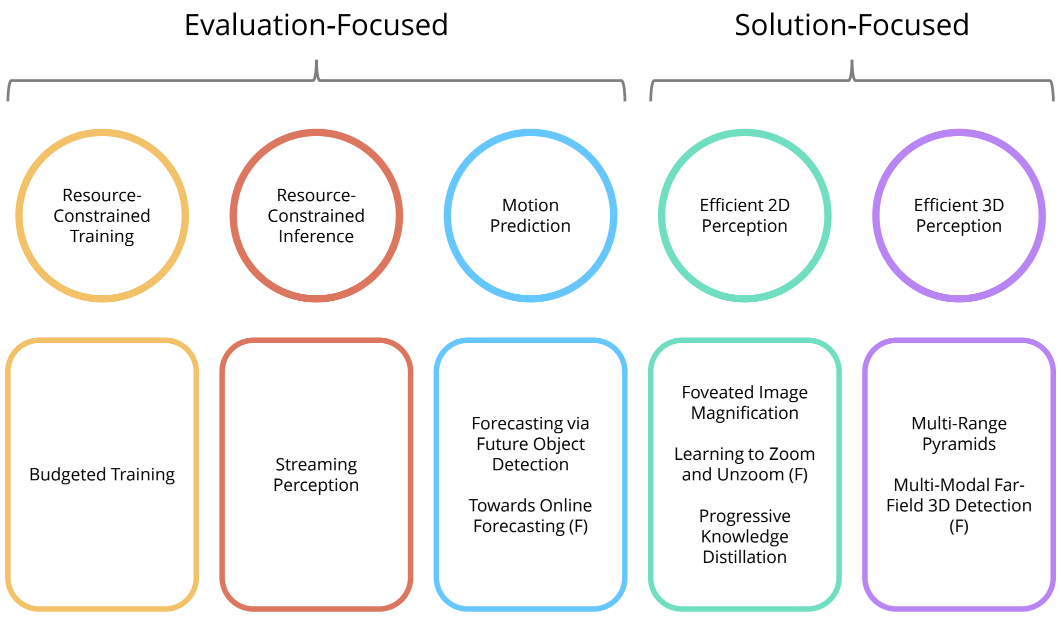

Evaluation-Focused
Solution-Focused
Resource-Constrained Training
Resource-Constrained Inference
Motion Prediction
Efficient 2D Perception
Efficient 3D Perception
Budgeted Training
Streaming Perception
Forecasting via Future Object Detection
Towards Online Forecasting (F)
Foveated Image Magnification
Learning to Zoom and Unzoom (F)
Progressive Knowledge Distillation
Multi-Range Pyramids
Multi-Modal Far-Field 3D Detection (F)

Figure 1.1: Thesis projects grouped by topics. (F) indicates future work.

As shown in Fig 1.1, my work in the space of resource-constrained learning and inference can be grouped into two parts, in which one is evaluation-focused and the other is solution-focused. Within each part, I have explored several different topics, including resource-constrained training, resource-constrained inference, motion prediction, efficient 2D perception, and efficient 3D perception. Under each topic, I have accomplished (or plan to work on) one or multiple projects. In total, this thesis covers the work of six projects and three future directions. Each project corresponds to a separate chapter. I list below the publications associated with each project. First, this list gives credits to all my collaborators. Second, the project websites listed provide additional multimedia information for supplementary illustrations.

- **Chapter 2**: Mengtian Li, Ersin Yumer and Deva Ramanan. *Budgeted Training: Rethinking Deep Neural Network Training Under Resource Constraints*. In ICLR, 2020. Project website: https://www.cs.cmu.edu/~mengtial/proj/budgetnn/.

- **Chapter 3**: Mengtian Li, Yu-Xiong Wang and Deva Ramanan. *Towards Streaming Perception*. In ECCV, 2020. Project website: https://www.cs.cmu.edu/~mengtial/proj/streaming/.

- **Chapter 4**: Chittesh Thavamani*, Mengtian Li*, Nicolas Cebron and Deva Ramanan. *FOVEA: Foveated Image Magnification for Autonomous Navigation*. In ICCV, 2021. * denotes equal contribution. Project website: https://www.cs.cmu.edu/~mengtial/proj/fovea/.

- **Chapter 5**: Shengcao Cao, Mengtian Li, James Hays, Deva Ramanan and Liangyan Gui. *Learning Lightweight Object Detectors via Progressive Knowledge Distillation*. Under review.

- **Chapter 6**: Mengtian Li, Benjamin Wilson, Yu-Xiong Wang, James Hays and Deva Ramanan. *Multi-Range Pyramids for 3D Object Detection*. Under review. Project website: https://www.cs.cmu.edu/~mengtial/proj/multirange/.

- **Chapter 7**: Neehar Peri, Jonathon Luiten, Mengtian Li, Aljosa Osep, Laura Leal-Taixé and Deva Ramanan. *Forecasting from LiDAR via Future Object Detection*. In CVPR, Jun 2022. Project website: https://github.com/neeharperi/FutureDet.

In **Chapter 8**, I will discuss several extensions to the above projects, either to cover additional tasks or to improve the efficiency and generality of proposed solutions in the above projects.

The following subsections contain summary for each project.

1.1.1 Budgeted Training

In most practical settings and theoretical analyses, one assumes that a model can be trained until convergence. However, the growing complexity of machine learning datasets and models may violate such assumptions. Indeed, current approaches for hyper-parameter tuning and neural architecture search tend to be limited by practical resource constraints. Therefore, we introduce a formal setting for studying training under the non-asymptotic, resource-constrained regime, i.e., budgeted training. We analyze the following problem: "given a dataset, algorithm, and *fixed* resource budget, what is the best achievable performance?" We focus on the number of optimization iterations as the representative resource. Under such a setting, we show that it is critical to adjust the learning rate schedule according to the given budget. Among budget-aware learning schedules, we find simple linear decay to be both robust and high-performing. We support our claim through extensive experiments with state-of-the-art models on ImageNet (image classification), Kinetics (video classification), MS COCO (object detection and instance segmentation), and Cityscapes (semantic segmentation). We also analyze our results and find that the key to a good schedule is budgeted convergence, a phenomenon whereby the gradient vanishes at the end of each allowed budget. We also revisit existing approaches for fast convergence and show that budget-aware learning schedules readily outperform such approaches under (the practical but under-explored) budgeted training setting.

1.1.2 Streaming Perception

Embodied perception refers to the ability of an autonomous agent to perceive its environment so that it can (re)act. The responsiveness of the agent is largely governed by latency of its processing pipeline. While past work has studied the algorithmic trade-off between latency and accuracy, there has not been a clear metric to compare different methods along the Pareto optimal latency-accuracy curve. We point out a discrepancy between standard offline evaluation and real-time applications: by the time an algorithm finishes processing a particular frame, the surrounding world has changed. To these ends, we present an approach that coherently integrates latency and accuracy into a single metric for real-time online perception, which we refer to as "streaming accuracy". The key insight behind this metric is to jointly evaluate the output of the entire perception stack at every time instant, forcing the stack to consider the amount of streaming data that should be ignored while computation is occurring. More broadly, building upon this metric, we introduce a meta-benchmark that systematically converts any single-frame task into a streaming perception task. We focus on the illustrative tasks of object detection and instance segmentation in urban video streams, and contribute a novel dataset with high-quality and temporally-dense annotations. Our proposed solutions and their empirical analysis demonstrate a number of surprising conclusions: (1) there exists an optimal "sweet spot" that maximizes streaming accuracy along the Pareto optimal latency-accuracy curve, (2) asynchronous tracking and future forecasting naturally emerge as internal representations that enable streaming perception, and (3) dynamic scheduling can be used to overcome temporal aliasing, yielding the paradoxical result that latency is sometimes minimized by sitting idle and "doing nothing".

1.1.3 Foveated Image Magnification

Efficient processing of high-res video streams is safety-critical for many robotics applications such as autonomous driving. To maintain real-time performance, many practical systems downsample the video stream. But this can hurt downstream tasks such as (small) object detection. Instead, we take inspiration from biological vision systems that allocate more foveal "pixels" to salient parts of the scene. We introduce FOVEA, an approach for intelligent downsampling that ensures salient image regions remain "magnified" in the downsampled output. Given a high-res image, FOVEA applies a differentiable resampling layer that outputs a small fixed-size image canvas, which is then processed with a differentiable vision module (e.g., object detection network), whose output is then differentiably backward mapped onto the original image size. The key idea is to resample such that background pixels can make room for salient pixels of interest. In order to ensure the overall pipeline remains efficient, FOVEA makes use of cheap and readily available cues for saliency, including dataset-specific spatial priors or temporal priors computed from object predictions in the recent past. On the autonomous driving datasets Argoverse-HD and BDD100K, our proposed method boosts the detection AP over standard Faster R-CNN, both with and without finetuning. Without any noticeable increase in compute, we improve accuracy on small objects by over 2x without degrading performance on large objects. Finally, FOVEA sets a new record for streaming AP (from 17.8 to 23.0 on a GTX 1080 Ti GPU), a metric designed to capture both accuracy and latency.

1.1.4 Progressive Knowledge Distillation

Resource-constrained perception systems such as edge computing and vision-for-robotics require vision models to be both accurate and lightweight in computation and memory usage. Knowledge distillation is one effective strategy to improve the performance of lightweight classification models, but it is less well-explored for structured outputs such as object detection and instance segmentation, where the variable number of outputs and complex internal network modules complicate the distillation. In this paper, we propose a simple yet surprisingly effective sequential approach to knowledge distillation that progressively transfers the knowledge of a set of teachers to a given lightweight student. Our approach is inspired by curriculum learning: To distill knowledge from a highly accurate but complex teacher model, we construct a sequence of teachers to help the student gradually adapt. We propose a heuristic algorithm to find the near-optimal order of teachers, and exploit the backbone-neck-head modularity of detection networks to distill at the neck level. Extensive experiments show significant gains brought by our approach. On the MS COCO benchmark, we improve ResNet-50 based Mask R-CNN's detection performance by 3.2 AP, and we improve ResNet-50 based RetinaNet by 3.4 AP.

1.1.5 Multi-Range Pyramids

LiDAR-based 3D detection plays a vital role in autonomous navigation. Contemporary solutions make use of 3D voxel representations, often encoded with a bird's-eye view (BEV) feature map. While quite intuitive, such representations scale quadratically with the spatial range of the map, making them ill-suited for far-field perception. In this paper, we present a multi-range representation that retains the benefits of BEV while remaining efficient by exploiting the following insight: near-field lidar measurements are dense and optimally encoded by small voxels, while far-field measurements are sparse and better encoded with large voxels. We exploit this observation to build a collection of range experts tuned for near-vs-far field detection, and show that they can share information with each other via a single multi-range feature pyramid. We show how standard convolutions need to be adjusted for this novel representation and provide local and global across-range feature sharing mechanisms to work around this problem. We evaluate our method on the long-range detection dataset Argoverse (up to $\pm$200m), and find that our method achieves significantly higher accuracy than competitive baselines while being faster in terms of wall-clock runtime.

1.1.6 Future Object Detection

Object detection and forecasting are fundamental components of embodied perception. These two problems, however, are largely studied in isolation by the community. In this paper, we propose an end-to-end approach for detection and motion forecasting based on raw sensor measurement as opposed to ground truth tracks. Instead of predicting the current frame locations and forecasting forward in time, we directly predict future object locations and backcast to determine where each trajectory began. Our approach not only improves overall accuracy compared to other modular or end-to-end baselines, it also prompts us to rethink the role of explicit tracking for embodied perception. Additionally, by linking future and current locations in a many-to-one manner, our approach is able to reason about multiple futures, a capability that was previously considered difficult for end-to-end approaches. We conduct extensive experiments on the popular nuScenes dataset and demonstrate the empirical effectiveness of our approach. In addition, we investigate the appropriateness of reusing standard forecasting metrics for an end-to-end setup, and find a number of limitations which allow us to build simple baselines to game these metrics. We address this issue with a novel set of joint forecasting and detection metrics that extend the commonly used AP metrics from the detection community to measuring forecasting accuracy.

Chapter 2

Budgeted Training

2.1 Introduction

Deep neural networks have made an undeniable impact in advancing the state-of-the-art for many machine learning tasks. Improvements have been particularly transformative in computer vision [82, 97]. Much of these performance improvements were enabled by an ever-increasing amount of labeled visual data [119, 185] and innovations in training architectures [84, 118].

However, as training datasets continue to grow in size, we argue that an additional limiting factor is that of resource constraints for training. Conservative prognostications of dataset sizes – particularly for practical endeavors such as self-driving cars [16], assistive medical robots [208], and medical analysis [62] – suggest one will train on datasets orders of magnitude larger than those that are publicly available today. Such planning efforts will become more and more crucial, because *in the limit, it might not even be practical to visit every training example before running out of resources* [18, 173].

We note that resource-constrained training already is *implicitly* widespread, as the vast majority of practitioners have access to limited compute. This is particularly true for those pursuing research directions that require a massive number of training runs, such as hyper-parameter tuning [128] and neural architecture search [25, 140, 262].

Instead of asking "what is the best performance one can achieve given this data and algorithm?", which has been the primary focus in the field so far, we decorate this question with *budgeted training* constraints as follows: "what is the best performance one can achieve given this data and algorithm within the allowed budget?". Here, the *allowed budget* refers to a limitation

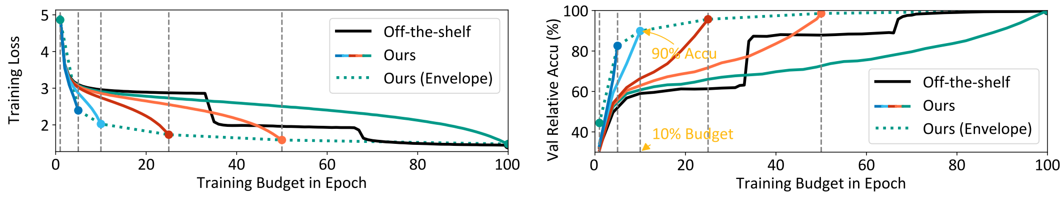


Figure 2.1: We formalize the problem of *budgeted training*, in which one maximizes performance subject to a fixed training budget. We find that a simple and effective solution is to adjust the learning rate schedule accordingly and anneal it to 0 at the end of the training budget. This significantly outperforms off-the-shelf schedules, particularly for small budgets. This plot shows several training schemes (solid curves) for ResNet-18 on ImageNet. The vertical axis in the right plot is normalized by the validation accuracy achieved by the full budget training. The dotted green curve indicates an efficient way of trading off computation with performance.

on the total time, compute, or cost spent on training. More specifically, we focus on limiting the number of iterations. This allows us to abstract out the specific constraint without loss of generality since any one of the aforementioned constraints could be converted to a finite iteration limit. We make the underlying assumption that the network architecture is constant throughout training, though it may be interesting to entertain changes in architecture during training [187, 224].

Much of the theoretical analysis of optimization algorithms focuses on asymptotic convergence and optimality [19, 160, 182], which implicitly makes use of an *infinite* compute budget. That said, there exists a wide body of work [108, 148, 177, 261] that provide performance bounds which depend on the iteration number, which apply even in the non-asymptotic regime. Our work differs in its exploration of maximizing performance for a fixed number of iterations. Importantly, the globally optimal solution may not even be achievable in our budgeted setting.

Given a limited budget, one obvious strategy might be data subsampling [6, 191]. However, we discover that a much more effective, simpler, and under-explored strategy is adopting budget-aware learning rate schedules — if we know that we are limited to a single epoch, one should tune the learning schedule accordingly. Such budget-aware schedules have been proposed in previous work [64, 133], but often for a fixed learning rate that depends on dataset statistics. In this paper, we specifically point out *linearly* decaying the learning rate to 0 at the end of the budget, may be more robust than more complicated strategies suggested in prior work. Though we are motivated by budget-aware training, we find that a linear schedule is quite competitive for general learning settings as well. We verify our findings with state-of-the-art models on ImageNet (image classification), Kinetics (video classification), MS COCO (object detection and instance segmentation), and Cityscapes (semantic segmentation).

We conduct several diagnostic experiments that analyze learning rate decays under the budgeted setting. We first observe a statistical correlation between the learning rate and the *full* gradient magnitude (over the entire dataset). Decreasing the learning rate empirically results in a decrease in the full gradient magnitude. Eventually, as the former goes to zero, the latter vanishes as well, suggesting that the optimization has reached a critical point, if not a local minimum[1]. We call this phenomenon *budgeted convergence* and we find it generalizes across budgets. On one hand, it implies that one should decay the learning rate to zero at the end of the training, even given a small budget. On the other hand, it implies one should not aggressively decay the learning rate early in the optimization (such as the case with an exponential schedule) since this may slow down later progress. Finally, we show that linear budget-aware schedules outperform recently-proposed fast-converging methods that make use of adaptive learning rates and restarts.

Our main contributions are as follows:

- We introduce a formal setting for budgeted training based on training iterations and provide an alternative perspective for existing learning rate schedules.
- We discover that budget-aware schedules are handy solutions to budgeted training. Specifically, our proposed linear schedule is more simple, robust, and effective than prior approaches, for both budgeted and general training.
- We provide an empirical justification of the effectiveness of learning rate decay based on the correlation between the learning rate and the full gradient norm.

2.2 Related Work

Learning rates. Stochastic gradient descent dates back to [182]. The core is its update step: $w_t = w_{t-1} - \alpha_t g_t$, where t (from 1 to T) is the iteration, w

[1]Whether such a solution is exactly a local minimum or not is debatable (see Sec 2.2).

are the parameters to be learned, g is the gradient estimator for the objective function[2] F, and α_t is the *learning rate*, also known as *step size*. Given base learning rate α_0, we can define the ratio $\beta_t = \alpha_t/\alpha_0$. Then the set of $\{\beta_t\}_{t=1}^{T}$ is called the *learning rate schedule*, which specifies how the learning rate should vary over the course of training. *Our definition differs slighter from prior art as it separates the base learning rate and learning rate schedule*. Learning rates are well studied for (strongly) convex cost surfaces.

Learning rate schedule for deep learning. In deep learning, there is no consensus on the exact role of the learning rate. Most theoretical analysis makes the assumption of a small and constant learning rate [57,58,81]. For variable rates, one hypothesis is that large rates help move the optimization over large energy barriers while small rates help converge to a local minimum [96, 112, 144]. Such hypothesis is questioned by recent analysis on mode connectivity, which has revealed that there does exist a descent path between solutions that were previously thought to be isolated local minima [55, 69, 75]. Despite a lack of theoretical explanation, the community has adopted a variety of heuristic schedules for practical purposes, two of which are particularly common:

- **step decay**: drop the learning rate by a multiplicative factor γ after every d epochs. The default for γ is 0.1, but d varies significantly across tasks.

- **exponential**: $\beta_t = \gamma^t$. There is no default parameter for γ and it requires manual tuning.

State-of-the-art codebases for standard vision benchmarks tend to employ step decay [26,82,97,150,222,234,242], whereas exponential decay has been successfully used to train Inception networks [201–203]. In spite of their prevalence, these heuristics have not been well studied. Recent work proposes several new schedules [92,144,198], but much of this past work limits their evaluation to CIFAR and ImageNet. For example, SGDR [144] advocates for learning-rate restarts based on the results on CIFAR, however, we find the unexplained form of cosine decay in SGDR is more effective than the restart technique. Notably, [158] demonstrate the effectiveness of linear rate decay with CaffeNet on downsized ImageNet. In our work, we rigorously evaluate on 5 standard vision benchmarks with state-of-the-art networks and under various budgets. [75] also analyze learning rate restarts

[2]Note that g can be based on a single example, a mini-batch, the full training set, or the true data distribution. In most practical settings, momentum SGD is used, but we omit the momentum here for simplicity.

and in addition, the warm-up technique, but do not analyze the specific form of learning rate decay.

Adaptive learning rates. Adaptive learning rate methods [108, 148, 177, 211] adjust the learning rate according to the local statistics of the cost surface. Despite having better theoretical bounds under certain conditions, they do not generalize as well as momentum SGD for benchmark tasks that are much larger than CIFAR [230]. We offer new insights by evaluating them under the budgeted setting. We show fast descent can be trivially achieved through budget-aware schedules and aggressive early descent is not desirable for achieving good performance in the end.

2.3 Learning Rates and Budgets

2.3.1 Budget-Aware Schedules

Learning rate schedules are often defined assuming unlimited resources. As we argue, resource constraints are an undeniable practical aspect of learning. One simple approach for modifying an existing learning rate schedule to a budgeted setting is early-stopping. Fig 2.1 shows that one can dramatically improve results of early stopping by more than 60% by *tuning* the learning rate for the appropriate budget. To do so, we simply reparameterize the learning rate sequence with a quantity not only dependent on the absolute iteration t, but also the training budget T:

Definition (Budget-Aware Schedule). Let T be the training budget, t be the current step, then a training progress p is t/T. A *budget-aware learning rate schedule* is

$$\beta_p : p \mapsto f(p), \tag{2.1}$$

where $f(p)$ is the ratio of learning rate at step t to the base learning rate α_0.

At first glance, it might be counter-intuitive for a schedule to *not* depend on T. For example, for a task that is usually trained with 200 epochs, training 2 epochs will end up at a solution very distant from the global optimal no matter the schedule. In such cases, conventional wisdom from convex optimization suggests that one should employ a large learning rate (constant schedule) that efficiently descends towards the global optimal. However, in the non-convex case, we observe empirically that a better strategy is to systematically decay the learning rate in proportion to the total iteration budget.

Budge-Aware Conversion (BAC). Given a particular rate schedule $\beta_t = f(t)$, one simple method for making it budget-aware is to rescale it, *i.e.*, $\beta_p =$

$f(pT_0)$, where T_0 is the budget used for the original schedule. For instance, a step decay for 90 epochs with two drops at epoch 30 and epoch 60 will convert to a schedule that drops at 1/3 and 2/3 training progress. Analogously, an exponential schedule 0.99^t for 200 epochs will be converted into $(0.99^{200})^p$.

It is worth noting that such an adaptation strategy already exists in well-known codebases [82] for training with limited schedules. Our experiments confirm the effectiveness of BAC as a general strategy for converting many standard schedules to be budget-aware (Tab 2.1). *For our remaining experiments, we regard BAC as a known technique and apply it to our baselines by default.*

Budget	1%	5%	10%	25%	50%	100%
exp .99	.5848	.8030	.8352	.8888	.9072	.9320
BAC	**.6086**	**.8560**	**.8996**	**.9228**	**.9272**	N/A
step-d1	.5710	.8058	.8422	.8702	.8746	.9434
BAC	**.5880**	**.8662**	**.9066**	**.9312**	**.9392**	N/A

Table 2.1: Effectiveness of budget-aware conversion (BAC) on CIFAR-10 for image classification with ResNet-18 [84]. The numbers are classification accuracy on the validation set. The 100% budget refers to training for 200 epochs. "step-d1" denotes step decay dropping once at training progress 50%. Please refer to Sec 2.4.1 for the complete setup.

Recent schedules: Interestingly, several recent learning rate schedules are implicitly defined as a function of progress $p = \frac{t}{T}$, and so are budget-aware by our definition:

- **poly** [103]: $\beta_p = (1-p)^\gamma$. No parameter other than $\gamma = 0.9$ is used in published work.
- **cosine** [144]: $\beta_p = \eta + \frac{1}{2}(1-\eta)(1+\cos(\pi p))$. η specify a lower bound for the learning rate, which defaults to zero.
- **htd** [92]: $\beta_p = \eta + \frac{1}{2}(1-\eta)(1-\tanh(L+(U-L)p))$. Here η has the same representation as in cosine. It is reported that $L = -6$ and $U = 3$ performs the best.

The poly schedule is a feature in Caffe [103] and adopted by the semantic segmentation community [34, 251]. The cosine schedule is a byproduct in work that promotes learning rate restarts [144]. The htd schedule is recently proposed [92], which however, contains only limited empirical evaluation.

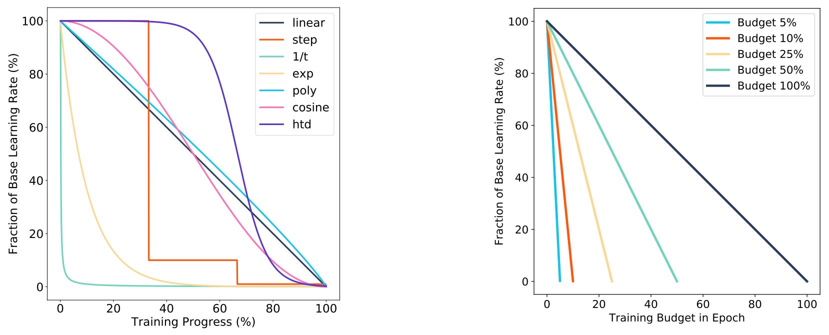


Figure 2.2: We normalize various learning rate schedules by training progress (left). Our solution to budgeted training is simple and universal — we decrease the learning rate linearly across the entire given budget (right).

None of these analyze their budget-aware property or provides intuition for such forms of decay. These schedules were treated as "yet another schedule". However, our definition of budget-aware makes these schedules stand out as a general family.

2.3.2 Linear Schedule

Inspired by existing budget-aware schedules, we borrow an even simpler schedule from the simulated annealing literature [111, 153, 161][3]:

$$\textbf{linear}: \beta_p = 1 - p. \tag{2.2}$$

In Fig 2.2, we compare linear schedule with various existing schedules under the budget-aware setting. Note that this linear schedule is completely parameter-free. This property is particularly desirable in budgeted training, where little budget exists for tuning such a parameter. The excellent generalization of linear schedule across budgets (shown in the next section) might imply that the cost surface of deep learning is to some degree self-similar. Note that a linear schedule, together with other recent budget-aware schedules, produces a constant learning rate in the asymptotic limit *i.e.*, $\lim_{T\to\infty}(1 - t/T) = 1$. Consequently, such practically high-performing schedules tend to be ignored in theoretical convergence analysis [19, 182].

[3]A link between SGD and simulated annealing has been recognized decades ago, where learning rate plays the role of temperature control [17]. Therefore, cooling schedules in simulated annealing can be transferred into learning rate schedules for SGD.

Budget	1%	5%	10%	25%	50%	100%
const	.5748 $\pm$ .0337	.7989 $\pm$ .0093	.8350 $\pm$ .0122	.8658 $\pm$ .0007	.8723 $\pm$ .0044	.8767 $\pm$ .0066
exp .95	.4834 $\pm$ .0125	.7575 $\pm$ .0053	.8567 $\pm$ .0027	.9147 $\pm$ .0030	.9295 $\pm$ .0006	.9468 $\pm$ .0021
exp .97	.5467 $\pm$ .0202	.8348 $\pm$ .0016	.8936 $\pm$ .0030	.9294 $\pm$ .0024	.9413 $\pm$ .0015	.9551 $\pm$ .0004
exp .99	.6069 $\pm$ .0219	.8557 $\pm$ .0037	.9013 $\pm$ .0036	.9227 $\pm$ .0033	.9268 $\pm$ .0026	.9310 $\pm$ .0023
step-d1	.5853 $\pm$ .0134	.8643 $\pm$ .0027	.9063 $\pm$ .0023	.9307 $\pm$ .0020	.9423 $\pm$ .0027	.9426 $\pm$ .0031
step-d2	.5487 $\pm$ .0156	.8342 $\pm$ .0052	.9043 $\pm$ .0034	.9319 $\pm$ .0037	.9461 $\pm$ .0019	.9529 $\pm$ .0009
step-d3	.4879 $\pm$ .0036	.7929 $\pm$ .0061	.8864 $\pm$ .0027	.9259 $\pm$ .0006	.9437 $\pm$ .0001	.9527 $\pm$ .0019
htd	.6450 $\pm$ .0070	.8899 $\pm$ .0043	.9219 $\pm$ .0014	.9449 $\pm$ .0031	.9520 $\pm$ .0023	.9554 $\pm$ .0013
cosine	.6343 $\pm$ .0080	.8851 $\pm$ .0024	.9223 $\pm$ .0024	.9432 $\pm$ .0024	.9520 $\pm$ .0026	.9552 $\pm$ .0021
poly	.6595 $\pm$ .0086	.8905 $\pm$ .0017	.9247 $\pm$ .0008	.9421 $\pm$ .0019	.9494 $\pm$ .0034	.9540 $\pm$ .0012
linear	.6617 $\pm$ .0079	.8915 $\pm$ .0011	.9217 $\pm$ .0028	.9412 $\pm$ .0018	.9537 $\pm$ .0020	.9563 $\pm$ .0009

Table 2.2: Comparison of learning rate schedules on CIFAR-10. The 1st, 2nd and the 3rd place under each budget are color coded. The number here is the classification accuracy and each one is the average of 3 independent runs. “step-dx” denotes decay x times at even intervals with $\gamma = 0.1$. For “exp” and “step” schedules, BAC (Sec 2.3.1) is applied in place of early stopping. We can see linear schedule surpasses other schedules under almost all budgets.

2.4 Experiments

In this section, we first compare linear schedule against other existing schedules on the small CIFAR-10 dataset and then on a broad suite of vision benchmarks. The CIFAR-10 experiment is designed to extensively evaluate each learning schedule while the vision benchmarks are used to verify the observation on CIFAR-10. We provide important implementation settings in the main text while leaving the rest of the details to the Appendix 2.A.12. In addition, we provide in Appendix 2.A.1 the evaluation with a large number of random architectures in the setting of neural architecture search.

2.4.1 CIFAR

CIFAR-10 [117] is a dataset that contains 60,000 tiny images (32×32). Given its small size, it is widely used for validating novel architectures. We follow the standard setup for dataset split [97], which is randomly holding out 5,000 from the 50,000 training images to form the validation set. For each budget, we report the best validation accuracy among epochs up till the end of the budget. We use ResNet-18 [84] as the backbone architecture and utilize SGD with base learning rate 0.1, momentum 0.9, weight decay 0.0005 and a batch size 128.

We study learning schedules in several groups: (a) constant (equivalent to not using any schedule). (b) & (c) exponential and step decay, both of which are commonly adopted schedules. (d) htd [92], a quite recent addition and not yet adopted in practice . We take the parameters with the best-reported performance $(-6, 3)$. Note that this schedule decays much slower initially than the linear schedule (Fig 2.2). (e) the smooth-decaying schedules (small curvature), which consists of cosine [144], poly [103] and the linear schedule.

As shown in Tab 2.2, the group of schedules that are budget-aware by our definition, outperform other schedules under all budgets. The linear schedule in particular, performs best most of the time including the typical full budget case. Noticeably, when exponential schedule is well-tuned for this task ($\gamma = 0.97$), it fails to generalize across budgets. In comparison, the budget-aware group does not require tuning but generalizes much better.

Within the budget-aware schedules, cosine, poly and linear achieve very similar results. This is expected due to the fact that their numerical similarity at each step (Fig 2.2). These results might indicate that *the key for a robust budgeted-schedule is to decay smoothly to zero.* Based on these observations and results, we suggest linear schedule should be the “go-to” budget-aware schedule.

2.4.2 Vision Benchmarks

In the previous section, we showed that linear schedule achieves excellent performance on CIFAR-10, in a relatively toy setting. In this section, we study the comparison and its generalization to practical large scale datasets with various state-of-the-art architectures. In particular, we set up experiments to validate the performance of linear schedule across tasks and budgets.

Ideally, one would like to see the performance of all schedules in Fig 2.2 on vision benchmarks. Due to resource constraints, we include only the off-the-shelf step decay and the linear schedule. Note our CIFAR-10 experiment suggests that using cosine and poly will achieve similar performance as linear, which are already budget-aware schedules given our definition, so we focus on linear schedule in this section.

Budget	1%	5%	10%	25%	50%	100%
			Image classification on ImageNet with ResNet			
step	.2039 ± .0029	.5194 ± .0048	.5951 ± .0021	.6558 ± .0018	.6796 ± .0008	**.6934** ± .0018
linear	**.3063** ± .0036	**.5726** ± .0024	**.6232** ± .0004	**.6634** ± .0020	**.6818** ± .0013	.6933 ± .0012
			Object detection on COCO with Mask-RCNN			
step	.0486 ± .0024	.2003 ± .0008	.2541 ± .0005	.3149 ± .0015	.3530 ± .0005	.3767 ± .0009
linear	**.0513** ± .0042	**.2090** ± .0016	**.2626** ± .0008	**.3222** ± .0014	**.3572** ± .0003	**.3795** ± .0012
			Instance segmentation on COCO with Mask-RCNN			
step	.0487 ± .0029	.1925 ± .0004	.2388 ± .0007	.2907 ± .0003	.3202 ± .0009	.3395 ± .0009
linear	**.0507** ± .0040	**.1986** ± .0012	**.2457** ± .0007	**.2942** ± .0002	**.3242** ± .0005	**.3396** ± .0009
			Semantic segmentation on Cityscapes with PSPNet			
step	.4941 ± .0011	.6358 ± .0052	.6800 ± .0010	.7250 ± .0019	.7423 ± .0094	**.7651** ± .0032
linear	**.5424** ± .0034	**.6654** ± .0014	**.7076** ± .0047	**.7399** ± .0005	**.7575** ± .0041	.7633 ± .0008
			Video classification on Kinetics with I3D			
step	.2941 ± .0028	.4981 ± .0029	.5674 ± .0013	.6459 ± .0023	.6870 ± .0025	.7134 ± .0021
linear	**.3286** ± .0042	**.5297** ± .0014	**.5967** ± .0030	**.6634** ± .0020	**.6995** ± .0011	**.7223** ± .0031

Table 2.3: Robustness of linear schedule across budgets, tasks and architectures. Linear schedule significantly outperforms step decay given limited budgets. Note that the off-the-shelf decay for each dataset has different parameters optimized for the specific dataset. For all step decay schedules, BAC (Sec 2.3.1) is applied to boost their budgeted performance. To reduce stochastic noise, we report the average and the standard deviation of 3 independent runs. See Sec 2.4.2 for the metrics of each task (the higher the better for all tasks).

We consider the following suite of benchmarks spanning many flagship vision challenges:

Image classification on ImageNet. ImageNet [185] is a widely adopted standard for image classification task. We use ResNet-18 [84] and report the top-1 accuracy on the validation set with the best epoch. We follow the step decay schedule used in [97,165], which drops twice at uniform interval ($\gamma = 0.1$ at $p \in \{\frac{1}{3}, \frac{2}{3}\}$). We set the full budget to 100 epochs (10 epochs longer than typical) for easier computation of the budget.

Object detection and instance segmentation on MS COCO. MS COCO [139] is a widely recognized benchmark for object detection and instance segmentation. We use the standard COCO AP (averaged over IoU thresholds) metric for evaluating bounding box output and instance mask output. The AP of the final model on the validation set is reported in our experiment. We use the challenge winner Mask R-CNN [82] with a ResNet-50 backbone and follow its setup. For training, we adopt the 1x schedule (90k iterations), and the off-the-shelf [82] step decay that drops 2 times with $\gamma = 0.1$ at $p \in \{\frac{2}{3}, \frac{8}{9}\}$.

Semantic segmentation on Cityscapes. Cityscapes [42] is a dataset commonly used for evaluating semantic segmentation algorithms. It contains high quality pixel-level annotations of 5k images in urban scenarios. The default evaluation metric is the mIoU (averaged across class) of the output segmentation map. We use state-of-the-art model PSPNet [251] with a ResNet-50 backbone and the full budget is 400 epochs as in standard set up. The mIoU of the best epoch is reported. Interestingly, unlike other tasks in this series, this model by default uses the poly schedule. For complete evaluation, we add step decay that is the same in our ImageNet experiment in Tab 2.3.

Video classification on Kinetics with I3D. Kinetics [105] is a large-scale dataset of YouTube videos focusing on human actions. We use the 400-category version of the dataset and a variant of I3D [26] with training and data processing code publicly available [222]. The top-1 accuracy of the final model is used for evaluating the performance. We follow the 4-GPU 300k iteration schedule [222], which features a step decay that drops 2 times with $\gamma = 0.1$ at $p \in \{\frac{1}{2}, \frac{5}{6}\}$.

If we factor in the dimension of budgets, Tab 2.3 shows a clear advantage of linear schedule over step decay. For example, on ImageNet, linear achieves 51.5% improvement at 1% of the budget. Next, we consider the full budget setting, where we simply swap out the off-the-shelf schedule with linear schedule. We observe better (video classification) or comparable (other tasks) performance after the swap. This is surprising given the fact that linear schedule is parameter-free and thus not optimized for the particular task or network.

In summary, *the smoothly decaying linear schedule is a simple and effective strategy for budgeted training*. It significantly outperforms traditional step decay given limited budgets, while achieving comparable performance with the normal full budget setting.

2.5 Discussion

In this section, we summarize our empirical analysis with a *desiderata* of properties for effective budget-aware learning schedules. We highlight those are inconsistent with conventional wisdom and follow the experimental setup in Sec 2.4.1 unless otherwise stated.

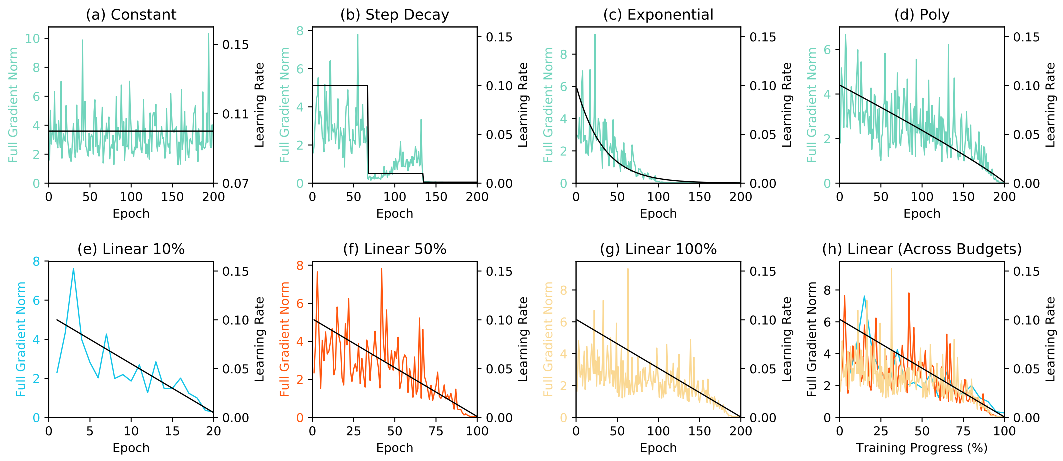


Figure 2.3: Budgeted convergence: full gradient norm $||g_t^*||$ vanishes over time (color curves) as learning rate α_t (black curves) decays. The first row shows that the dynamics of full gradient norm correlate with the corresponding learning rate schedule while the second row shows that such phenomena generalize across budgets for budget-aware schedules. Such generalization is most obvious in plot (h), which overlays the full gradient norm across different budgets. If a schedule does not decay to 0, the gradient norm does not vanish. For example, if we train a budget-unaware exponential schedule for 50 epochs (c), the full gradient norm at that time is around 1.5, suggesting this is a schedule with insufficient final decay of learning rate.

Desideratum: budgeted convergence. Convergence of SGD under non-convex objectives is measured by $\lim_{t\to\infty} \mathbb{E}[||\nabla F||^2] = 0$ [19]. Intuitively, one should terminate the optimization when no further local improvement can be made. What is the natural counterpart for "convergence" within a budget? For a dataset of N examples $\{(x_i, y_i)\}_{i=1}^N$, let us write the full gradient as $g_t^* = \frac{1}{N}\sum_{i=1}^N \nabla F(x_i, y_i)$. We empirically find that the dynamics of $||g_t^*||$ over time highly correlates with the learning rate α_t (Fig 2.3). As the learning rate vanishes for budget-aware schedules, so does the gradient magnitude. We call this "vanishing gradient" phenomenon *budgeted convergence*. This correlation suggests that decaying schedules to near-zero rates (and using BAC) may be more effective than early stopping. As a side note, budgeted convergence resonates with classic literature that argues that SGD behaves similar to simulated annealing [17]. Given that α_t and $||g_t^*||$ decrease, the

Schedule	Best Progress	Schedule	Best Progress
const	81.2% ± 16.1%	step-d2	90.5% ± 9.0%
linear	98.6% ± 1.6%	poly	99.1% ± 1.3%

Table 2.4: Where does one expect to find the model with the highest validation accuracy within the training progress? Here we show the best checkpoint location measured in training progress p and averaged for each schedule across budgets greater or equal than 10% and 3 different runs.

overall update $||-\alpha_t g_t||$ also decreases[4]. In other words, large moves are more likely given large learning rates in the beginning, while small moves are more likely given small learning rates in the end. However, the exact mechanism by which the learning rate influences the gradient magnitude remains unclear.

Desideratum: don't waste the budget. Common machine learning practise often produces multiple checkpointed models during a training run, where a validation set is used to select the best one. Such additional optimization is wasteful in our budgeted setting. Tab 2.4 summarizes the progress point at which the best model tends to be found. Step decay produces an optimal model somewhat towards the end of the training, while linear and poly are almost always optimal at the precise end of the training. This is especially helpful for state-of-the-art models where evaluation can be expensive. For example, validation for Kinetics video classification takes several hours. Budget-aware schedules require validation on only the last few epochs, saving additional compute.

Aggressive early descent. Guided by asymptotic convergence analysis, faster descent of the objective might be an apparent desideratum of an optimizer. Many prior optimization methods explicitly call for faster decrease of the objective [40, 108, 177]. In contrast, we find that one should not employ aggressive early descent because large learning rates can prevent budgeted convergence. Consider AMSGrad [177], an adaptive learning rate that addresses a convergence issue with the widely-used Adam optimizer [108]. Fig 4 shows that while AMSGrad does quickly descend over the training objective, it still underperforms budget-aware linear schedules over *any* given training budget. To examine why, we derive the equivalent rate $\widetilde{\beta}_t$ for AMSGrad and show that it is dramatically larger than our defaults, suggesting the optimizer is too aggressive.

[4]Note that the momentum in SGD is used, but we assume vanilla SGD to simplify the discussion, without losing generality.

Warm restarts. SGDR [144] explores periodic schedules, in which each period is a cosine scaling. The schedule is intended to escape local minima, but its effectiveness has been questioned [75]. Fig 5 shows that SDGR has faster descent but is inferior to budget-aware schedules for *any* budget (similar to the adaptive optimizers above). Additional comparisons can be found in Appendix 2.A.7. Whether there exists a method that achieves promising anytime performance and budgeted performance at the same time remains an open question.

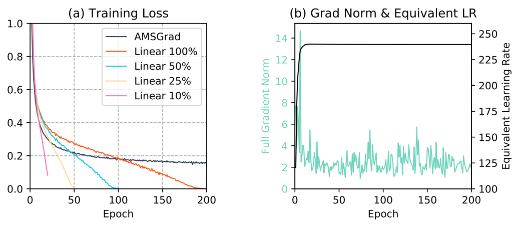


Figure 2.4: Comparing AMSGrad [177] with linear schedule. (a) while AMSGrad makes fast initial descent of the training loss, it is surpassed at each given budget by the linear schedule. (b) budgeted convergence is not observed for AMSGrad — the full gradient norm $||g_t^*||$ does not vanish (color curves). Comparing to a momentum SGD, AMSGrad recommends magnitudes larger learning rate $\widetilde{\beta}_t$ (black curve).

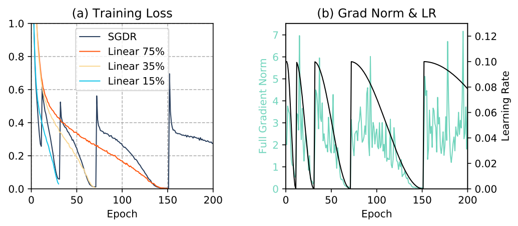


Figure 2.5: Comparing SGDR [144] with linear schedules. (a) SGDR makes slightly faster initial descent of the training loss, but is surpassed at each given budget by the linear schedule. (b) for SGDR, the correlation between full gradient norm $||g_t^*||$ and learning rate α_t is also observed. Warm restart does not help to achieve better budgeted performance.

2.6 Conclusion

This paper introduces a formal setting for budgeted training. Under this setup, we observe that a simple linear schedule, or any other smooth-decaying schedules can achieve much better performance. Moreover, the linear schedule even offers comparable performance on existing visual recognition tasks for the typical full budget case. In addition, we analyze the intriguing properties of learning rate schedules under budgeted training. We find that the learning rate schedule controls the gradient magnitude regardless of training stage. This further suggests that SGD behaves like simulated annealing and the purpose of a learning rate schedule is to control the stage of optimization.

2.A Appendix

2.A.1 Rank Prediction

In the introduction of this chapter, we list neural architecture search as an application of budgeted training. Due to resource constraint, these methods usually train models with a small budget (10-25 epochs) to evaluate their relative performance [23, 25, 175]. Under this setting, *the goal is to rank the performance of different architectures* instead of obtaining the best possible accuracy as in the regular case of budgeted training. Then one could ask the question that whether budgeted training techniques help in better predicting the relative rank. Unfortunately, budgeted training has not been studied or discussed in the neural architecture search literature, it is unknown how well models only trained with 10 epochs can tell the relative performance of the same ones that are trained with 200 epochs. Here we conduct a controlled experiment and show that proper adjustment of learning schedule, specifically the linear schedule, indeed improves the accuracy of rank prediction.

We adapt the code in [25] to generate 100 random architectures, which are obtained by random modifications (adding skip connection, removing layer, changing filter numbers) on top of ResNet-18 [82]. First, we train these architectures on CIFAR-10 given full budget (200 epochs), following the setting described in Sec 2.4.1. This produces a relative rank between all pairs of random architectures based on the validation accuracy and this rank is considered as the target to predict given limited budget. Next, every random architecture is trained with various learning schedules under various small budgets. For each schedule and each budget, this generates a complete rank. We treat this rank as the prediction and compare it with the target full-budget rank. The metric we adopt is Kendall's rank correlation coefficient (τ), a standard statistics metric for measuring rank similarity. It is based on counting the inversion pairs in the two ranks and $(\tau + 1)/2$ is approximately the probability of estimating the rank correctly for a pair.

We consider the following schedules: (1) constant, it might be possible that no learning rate schedule is required if only the relative performance is considered. (2) step decay ($\gamma = 0.1$, decay at $p \in \{\frac{1}{3}, \frac{2}{3}\}$), a schedule commonly used both in regular training and neural architecture search [164, 262]. (3) cosine, a schedule often used in neural architecture search [23, 175]. (4) linear, our proposed schedule. The results of their rank prediction capability can be seen in Tab 2.5.

The results suggest that with more budget, we can better estimate the full-budget rank between architectures. And even if only relative performance is considered, learning rate decay should be applied. Specifically, smooth-decaying schedule, such as linear or cosine, are preferred over step decay.

We list some additional details about the experiment. To reduce stochastic noise, each configuration under both the small and full budget is repeated 3 times and the median accuracy is taken. The full-budget model is trained with linear schedule, similar results are expected with other schedules as evidenced by the CIFAR-10 results in the main text (Tab 2.2). Among the 100 random architectures, 21 cannot be trained, the rest of 79 models have validation accuracy spanning from 0.37 to 0.94, with the distribution mass centered at 0.91. Such skewed and widespread distribution is the typical case in neural architecture search. We remove the 21 models that cannot be trained for our experiments. We take the epoch with the best validation accuracy for each configuration, so the drawback of constant or step decay not having the best model at the very end does not affect this experiment (see Sec 2.5).

2.A.2 Budgeted Performance Across Architectures

To reinforce our claim that linear schedule generalizes across different settings, we compare budgeted performance of various schedules on random architectures generated in the previous section. We present two versions of the results. The first is to directly average the validation accuracy of different architecture with each schedule and under each budget (Tab 2.6). The second is to normalize by dividing the budgeted accuracy by the full-budget accuracy of the same architecture and then average across different architectures (Tab 2.7). The second version assumes all architectures enjoy equal

Epoch (Budget)	1 (0.5%)	2 (1%)	10 (5%)	20 (10%)
const	0.3451	0.4595	0.6720	0.6926
step-d2	0.2746	0.3847	0.6651	0.7279
cosine	0.3211	**0.4847**	0.7023	**0.7563**
linear	**0.3409**	0.4348	**0.7398**	0.7351

Table 2.5: Small-budget and full-budget model rank correlation measured in Kendall's tau. Smooth-decaying schedules like linear and cosine can more accurately predict the true rank of different architectures given limited budget.

Epoch (Budget)	1 (0.5%)	2 (1%)	10 (5%)	20 (10%)
const	0.3892	0.4699	0.6689	0.7061
step-d2	0.4014	0.4780	0.6980	0.7754
cosine	0.4616	0.5498	0.7530	0.8029
linear	**0.4759**	**0.5745**	**0.7652**	**0.8192**

Table 2.6: Small-budget validation accuracy averaged across random architectures. Linear schedule is the most robust under small budgets.

Epoch (Budget)	1 (0.5%)	2 (1%)	10 (5%)	20 (10%)
const	0.4419	0.5343	0.7550	0.8015
step-d2	0.4590	0.5455	0.7894	0.8848
cosine	0.5326	0.6265	0.8615	0.9087
linear	**0.5431**	**0.6626**	**0.8644**	**0.9305**

Table 2.7: Tab 2.6 normalized by the full-budget accuracy and then averaged across architectures. Linear schedule achieves solutions closer to their full-budget performance than the rest of schedules under small budgets.

weighting. Under both cases, linear schedule is the most robust schedule across architectures under various budgets.

2.A.3 Equivalent Learning Rate For AMSGrad

In the discussion section, we use equivalent learning rate to compare AMSGrad [177] with momentum SGD. Here we present the derivation for the equivalent learning rate $\widetilde{\beta}_t$.

Let η_1, η_2 and ϵ be hyper-parameters, then the momentum SGD update rule is:

$$m_t = \eta_1 m_{t-1} + (1-\eta_1) g_t, \tag{2.3}$$

$$w_t = w_{t-1} - \alpha_0^{(1)} \beta_t m_t, \tag{2.4}$$

Budget	1%	5%	10%	25%	50%	100%
Subset	.3834	.6446	.7848	.8586	.9234	N/A
Full	**.5544**	**.8328**	**.9042**	**.9338**	**.9464**	**.9534**

Table 2.8: Comparison with offline data subsampling. "Subset" meets the budget constraint by randomly subsample the dataset prior to training, while "full" uses all the data, but restricting the number of iterations. Note that budget-aware schedule is used for "full".

while the AMSGrad update rule is:

$$m_t = \eta_1 m_{t-1} + (1-\eta_1) g_t, \tag{2.5}$$
$$v_t = \eta_2 v_{t-1} + (1-\eta_2) g_t^2, \tag{2.6}$$
$$\hat{m}_t = \frac{m_t}{1-\eta_1^t}, \tag{2.7}$$
$$\hat{v}_t = \frac{v_t}{1-\eta_2^t}, \tag{2.8}$$
$$\hat{v}_t^{\max} = \max(\hat{v}_t^{\max}, \hat{v}_t) \tag{2.9}$$
$$w_t = w_{t-1} - \alpha_0^{(2)} \frac{\hat{m}_t}{\sqrt{\hat{v}_t^{\max}} + \epsilon}. \tag{2.10}$$

Comparing equation 2.4 with 2.10, we obtain the equivalent learning rate:

$$\widetilde{\beta}_t = \frac{\alpha_0^{(2)}}{\alpha_0^{(1)}} \frac{1}{(1-\eta_1^t)(\sqrt{\hat{v}_t^{\max}} + \epsilon)}, \tag{2.11}$$

Note that the above equation holds per each weight. For Fig 2.4a, we take the median across all dimensions as a scalar summary since it is a skewed distribution. The mean appears to be even larger and shares the same trend as the median. In our experiments, we use the default hyper-parameters (which also turn out to have the best validation accuracy): $\alpha_0^{(1)} = 0.1$, $\alpha_0^{(2)} = 0.001$, $\eta_1 = 0.9$, $\eta_2 = 0.99$ and $\epsilon = 10^{-8}$.

2.A.4 Data Subsampling

Data subsampling is a straight-forward strategy for budgeted training and can be realized in several different ways. In our work, we limit the number of iterations to meet the budget constraint and this effectively limits the number of data points seen during the training process. An alternative is to construct a subsampled dataset offline, but keep the same number of training iterations. Such construction can be done by random sampling, which might be the most effective strategy for *i.i.d* (independent and identically distributed) dataset. We show in Tab 2.8 that even our baseline budge-aware step decay, together with a limitation on the iterations, can significantly outperform this offline strategy. For the subset setting, we use the off-the-shelf step decay (step-d2) while for the full set setting, we use the same step decay but with BAC applied (Sec 2.3.1). For detailed setup, we follow Sec 2.4.1.

Of course, more complicated subset construction methods exist, such as *core-set* construction [6]. However, such methods usually requires a feature summary of each data point and the computation of pairwise distance, making such methods unsuitable for extremely large dataset. In addition, note that our subsampling experiment is conducted on CIFAR-10, a well-constructed and balanced dataset, making smarter subsampling methods less advantageous. Consequently, the result in Tab 2.8 can as well provides a reasonable estimate for other complicated subsampling methods.

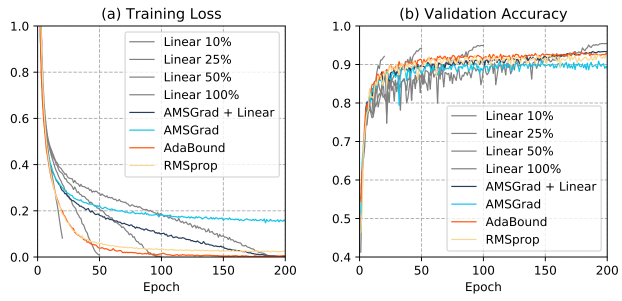
2.A.5 Additional Experiments on Cityscapes (Semantic Segmentation)

In the main text, we compare linear schedule against step decay for various tasks. However, the off-the-shelf schedule for PSPNet [251] is poly instead of step decay. Therefore, we include the evaluation of poly schedule on Cityscapes [42] in Tab 2.9. Given the similarity of poly and linear (Fig 2.2), and the opposite results on CIFAR-10 and Cityscapes, it is inconclusive that one is strictly better than the other within the smooth-decaying family. However, these smooth-decaying methods both outperform step decay given limited budgets.

Budget	1%	5%	10%	25%	50%	100%
poly	**.5476** $\pm$ .0023	**.6755** $\pm$ .0012	**.7093** $\pm$ .0058	**.7416** $\pm$ .0028	.7562 $\pm$ .0045	.7593 $\pm$ .0043
linear	.5424 $\pm$ .0034	.6654 $\pm$ .0014	.7076 $\pm$ .0047	.7399 $\pm$ .0005	**.7575** $\pm$ .0041	**.7633** $\pm$ .0008

Table 2.9: Comparison with off-the-shelf poly schedule on Cityscapes [42] using PSPNet [251]. Poly and linear are similar smooth-decaying schedules (Fig 2.2) and thus have similar performance. The exact rank differs from task to task.

2.A.6 Additional Comparison with Adaptive Learning Rates



Method	Val Accu
RMSprop	.9258
AdaBound	.9306
AMSGrad	.9113
AMSGrad + Linear	.9340
SGD + Linear 10%	.9218
SGD + Linear 25%	.9412
SGD + Linear 50%	.9546
SGD + Linear 100%	**.9562**

Figure 2.6: Comparison between budget-aware linear schedule and adaptive learning rate methods on CIFAR-10. We see while adaptive learning rate methods appear to descent faster than full budget linear schedule, at each given budget, they are surpassed by the corresponding linear schedule.

In the main text we compare linear schedule with AMSGrad [177] (the improved version over Adam [108]), we further include the classical method RMSprop [211] and the more recent AdaBound [148]. We tune these adaptive methods for CIFAR-10 and summarize the results in Fig 2.6. We observe the similar conclusion that budget-aware linear schedule outperforms adaptive methods for all given budgets.

Like SGD, those adaptive learning rate methods also takes input a parameter of base learning rate, which can also be annealed using an existing schedule. Although it is unclear why one needs to anneal an adaptive methods, we find that it in facts boosts the performance ("AMSGrad + Linear" in Fig 2.6).

2.A.7 Additional Comparison with SGDR

This section provides additional evaluation to show that learning rate restart produces worse results than our proposed budgeted training techniques under budgeted setting. In [144], both a new form of decay (cosine) and the technique of learning rate restart are proposed. To avoid confusion, we use "cosine schedule", or just "cosine", to refer to the form of decay and SGDR to a schedule of periodical cosine decays. The comparison with cosine schedule is already included in the main text. Here we focus on evaluating the periodical schedule. SGDR requires two parameters to specify the periods: T_0, the length of the first period; T_{mult}, where i-th period has length $T_i = T_0 T_{\text{mult}}^{i-1}$. In Fig 2.7, we plot the off-the-shelf SGDR schedule with $T_0 = 10$ (epoch),

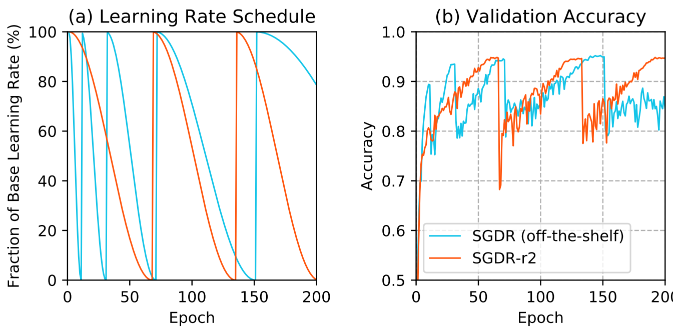


Figure 2.7: One issue with off-the-shelf SGDR ($T_0 = 10$, $T_{\text{mult}} = 2$) is that it is not budget-aware and might end at a poor solution. We convert it to a budget aware schedule by setting it to restart n times at even intervals across the budget and $n = 2$ is shown here (SGDR-r2).

Epoch	30	50	150
SGDR	.9320	.9458	.9510
linear	**.9350**	**.9506**	**.9532**

Table 2.10: Comparison with off-the-shelf SGDR at the end of each period after the first restart.

$T_{\text{mult}} = 2$. The validation accuracy plot (on the right) shows that it might end at a very poor solution (0.8460) since it is not budget-aware. Therefore, we consider two settings to compare linear schedule with SGDR. The first is to compare only at the end of each period of SGDR, where budgeted convergence is observed. The second is to convert SGDR into a budget-aware schedule by setting the schedule to restart n times at even intervals across the budget. The results under the first and second setting is shown in Tab 2.10 and Tab 2.11 respectively. Under both budget-aware and budget-unaware setting, linear schedule outperforms SGDR. For detailed setup, we follow Sec 2.4.1, of the main text and take the median of 3 runs.

2.A.8 Additional Illustrations

In the discussion section (Sec 2.5), we refer to validation accuracy curve for training on CIFAR-10, which we provide here in Fig 2.8.

Budget	1%	5%	10%	25%	50%	100%
SGDR-r1	.5002	.7908	.8794	.9250	.9380	.9488
SGDR-r2	.4710	.7888	.8738	.9216	.9412	.9502
linear	**.6654**	**.8920**	**.9218**	**.9412**	**.9546**	**.9562**

Table 2.11: Comparison with SGDR under budget-aware setting. "SGDR-r1" refers to restarting learning rate once at midpoint of the training progress, and "SGDR-r2" refers to restarting twice at even interval.

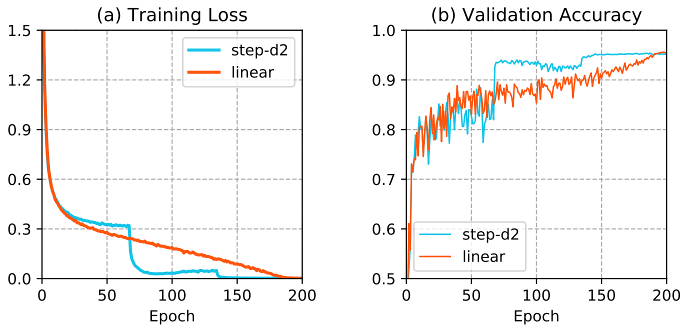


Figure 2.8: Training loss and validation accuracy for training ResNet-18 on CIFAR-10 using step decay and linear schedule. No generalization gap is observed when we only modify learning rate schedule. This figure provides details for the discussion of "don't waste budget".

2.A.9 Learning Rates in Convex Optimization

For convex cost surfaces, constant learning rates are guaranteed to converge when less or equal than $1/L$, where L is the Lipschitz constant for the gradient of the cost function ∇F [19]. Another well-known result ensures convergence for sequences that decay neither too fast nor too slow [182]: $\sum_{t=1}^{\infty} \alpha_t = \infty, \sum_{t=1}^{\infty} \alpha_t^2 < \infty$. One common such instance in convex optimization is $\alpha_t = \alpha_0/t$. For non-convex problems, similar results hold for convergence to a local minimum [19]. Unfortunately, there does not exist a theory for learning rate schedules in the context of general non-convex optimization.

2.A.10 Full Gradient Norm and the Weight Norm

In Sec 2.5, we plot the full gradient norm of the cross-entropy loss, excluding the regularization part. In fact, we use an L2-regularization (weight decay) of 0.0004 for these experiments. For completeness, we plot the weight norm

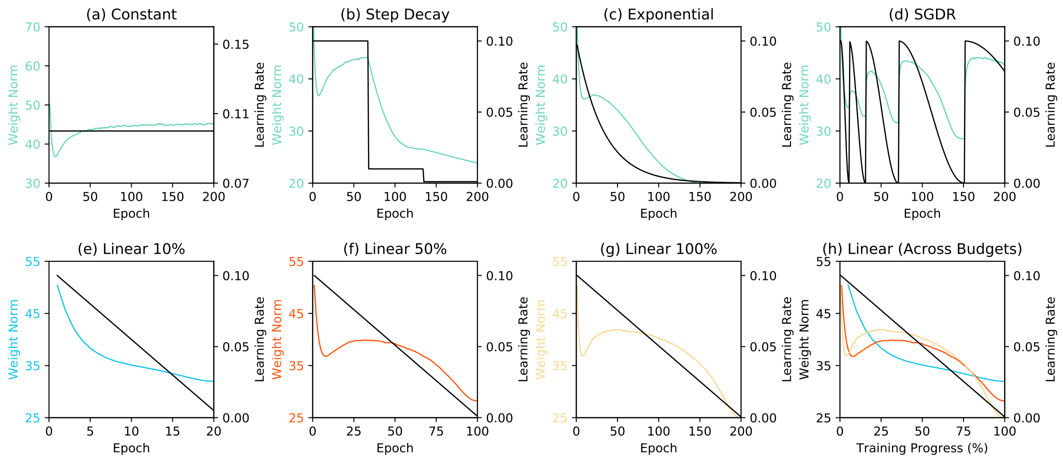


Figure 2.9: The corresponding weight norm plots for Fig 2.3 and Fig 5. We find that the weight norm exhibits a similar trend as the gradient norm.

Batch Size	Schedule	20%	50%	100%
64	step-d2	.9436 ± .0037	.9505 ± .0009	.9519 ± .0009
64	linear	**.9473** ± .0021	**.9511** ± .0008	**.9526** ± .0020
256	step-d2	.8939 ± .0027	.9291 ± .0021	.9431 ± .0008
256	linear	**.9143** ± .0018	**.9415** ± .0038	**.9484** ± .0013
1024	step-d2	.5851 ± .0460	.7703 ± .0121	.8805 ± .0007
1024	linear	**.7415** ± .0141	**.8553** ± .0023	**.8992** ± .0042

Table 2.12: Comparison between linear and step decay with different batch sizes. We can see that even when we vary the batch size, linear schedule outperforms step decay.

in Fig 2.9.

2.A.11 Additional ablation studies

Here we explore variations of batch size (Tab 2.12) and initial learning rate (Tab 2.13). Our definition of budget is the number of examples seen during training. So when the batch size increases, the number of iterations decreases. For example, on CIFAR-10, the full budget is training with batch size 128 for 200 epochs. If we train with batch size 1024 for 20% of the budget, that means training for 5 epochs.

Initial LR	0.001	0.1	1	10
step-d2	.9152 ± .0024	.9529 ± .0009	.8869 ± .0065	N/A
linear	**.9167** ± .0023	**.9563** ± .0009	**.8967** ± .0034	N/A

Table 2.13: Comparison between linear and step decay with different initial learning rate under full budget setting. On one hand, we see that linear schedule outperforms step decay under various initial learning rate. On the other hand, we see that initial learning rate is still a very important hyper-parameter that needs to be tuned even with budget-aware, smooth-decaying schedules.

2.A.12 Additional Implementation Details

Image classification on ImageNet. We adapt both the network architecture (ResNet-18) and the data loader from the open source PyTorch ImageNet example[5]. The base learning rate used is 0.1 and weight decay 5×10^{-4}. We train using 4 GPUs with asynchronous batch normalization and batch size 128.

Video classification on Kinetics with I3D. The 400-category version of the dataset is used in the evaluation. We use an open source codebase[6] that has training and data processing code publicly available. Note that the codebase implements a variant of standard I3D [26] that has ResNet as the backbone. We follow the configuration of `run_i3d_baseline_300k_4gpu.sh`, which specifies a base learning rate 0.005 and a weight decay 10^{-4}. Only learning rate schedule is modified in our experiments. We train using 4 GPUs with asynchronous batch normalization and batch size 32.

Object detection and instance segmentation on MS COCO. We use the open source implementation of Mask R-CNN[7], which is a PyTorch re-implementation of the official codebase Detectron in the Caffe 2 framework. We only modify the part of the code for learning rate schedule. The codebase sets base learning rate to 0.02 and weight decay 10^{-4}. We train with 8 GPUs (batch size 16) and keep the built-in learning rate warm up mechanism, which is an implementation technique that increases learning rate for 0.5k iterations and is intended for stabilizing the initial phase of multi-GPU training [76]. The 0.5k iterations are kept fixed for all budgets and learning

[5]https://github.com/pytorch/examples/tree/master/imagenet. PyTorch version 0.4.1.

[6]https://github.com/facebookresearch/video-nonlocal-net. Caffe 2 version 0.8.1.

[7]https://github.com/roytseng-tw/Detectron.pytorch. PyTorch version 0.4.1.

rate decay is applied to the rest of the training progress.

Semantic segmentation on Cityscapes. We adapt a PyTorch codebase obtained from correspondence with the authors of PSPNet. The base learning rate is set to 0.01 with weight decay 10^{-4}. The training time augmentation includes random resize, crop, rotation, horizontal flip and Gaussian blur. We use patch-based testing time augmentation, which cuts the input image to patches of 713×713 and processes each patch independently and then tiles the patches to form a single output. For overlapped regions, the average logits of two patches are taken. We train using 4 GPUs with *synchronous* batch normalization and batch size 12.

Chapter 3

Streaming Perception

3.1 Introduction

Embodied perception refers to the ability of an autonomous agent to perceive its environment so that it can (re)act. A crucial quantity governing the responsiveness of the agent is its reaction time. Practical applications, such as self-driving vehicles or augmented reality and virtual reality (AR/VR), may require reaction time that rivals that of humans, which is typically 200 milliseconds (ms) for visual stimuli [115]. In such settings, low-latency algorithms are imperative to ensure safe operation or enable a truly immersive experience.

Historically, the computer vision community has not particularly focused on algorithmic latency. This is one reason why a disparate set of techniques (and conference venues) have been developed for robotic vision. Interestingly, latency has been well studied recently (*e.g.*, fast but not necessarily state-of-the-art accurate detectors such as [138, 142, 178]). But it has still been primarily explored in an *offline* setting. Vision-for-online-perception imposes quite different latency demands as shown in Fig. 3.1, because by the time an algorithm finishes processing a particular frame — say, after 200ms — the surrounding world has changed! This forces perception to be ultimately predictive of the future. In fact, such predictive forecasting is a fundamental property of human vision (*e.g.*, as required whenever a baseball player strikes a fast ball [154]). So we argue that streaming perception should be of interest to general computer vision researchers.

Contribution (meta-benchmark) To help explore embodied vision in a truly online streaming context, we introduce a general meta-benchmark that

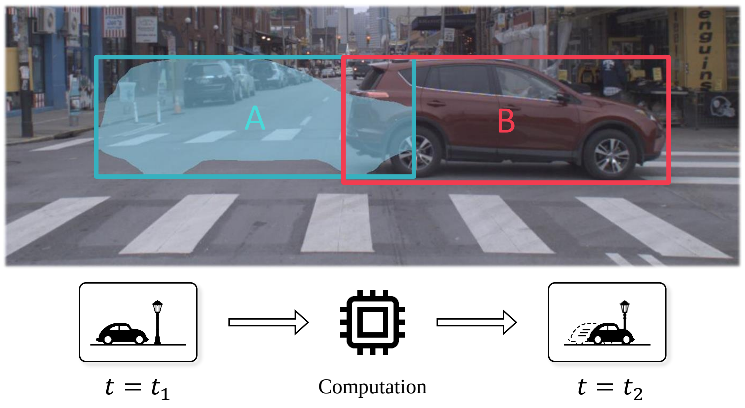


Figure 3.1: Latency is inevitable in a real-world perception system. The system takes a snapshot of the world at t_1 (the car is at location A), and when the algorithm finishes processing this observation, the surrounding world has already changed at t_2 (the car is now at location B, and thus there is a mismatch between prediction A and ground truth B). If we define streaming perception as a task of continuously reporting back the current state of the world, then how should one evaluate vision algorithms under such a setting? We invite the readers to watch a video on the project website that compares a standard frame-aligned visualization with our latency-aware visualization [Link].

systematically converts *any* single-frame task into a streaming perception task. Our key insight is that streaming perception requires understanding the state of the world at all time instants — *when a new frame arrives, streaming algorithms must report the state of the world even if they have not done processing the previous frame.* Within this meta-benchmark, we introduce an approach to measure the real-time performance of perception systems. The approach is as simple as querying the state of the world at all time instants, and the quality of the response is measured by the *original* task metric. Such an approach naturally merges latency and accuracy into a single metric. Therefore, the trade-off between accuracy versus latency can now be measured quantitatively. Interestingly, our meta-benchmark naturally evaluates the perception stack *as a whole*. For example, a stack may include detection, tracking, and forecasting modules. Our meta-benchmark can be used to directly compare such modular stacks to end-to-end black-box algorithms [145]. In addition, our approach addresses the issue that overall latency of concurrent systems is hard to evaluate (*e.g.*, latency cannot be simply characterized by the runtime of a single module).

Contribution (analysis) Motivated by perception for autonomous vehicles, we instantiate our meta-benchmark on the illustrative tasks of object detection and instance segmentation in urban video streams. Accompanied with our streaming evaluation is a novel dataset with high-quality, high-frame-rate, and temporally-dense annotations of urban videos. Our evaluation on these tasks demonstrates a number of surprising conclusions. (1) Streaming perception is significantly more challenging than offline perception. Standard metrics like object-detection average precision (AP) dramatically drop (from 38.0 to 6.2), indicating the need for the community to focus on such problems. (2) Decision-theoretic scheduling, asynchronous tracking, and future forecasting naturally *emerge* as internal representations that enable accurate streaming perception, recovering much of the performance drop (boosting performance to 17.8). With simulation, we can verify that infinite compute resources modestly improves performance to 20.3, implying that our conclusions are fundamental to streaming processing, no matter the hardware. (3) It is well known that perception algorithms can be tuned to trade off accuracy versus latency. Our analysis shows that there exists an optimal "sweet spot" that uniquely maximizes streaming accuracy. This provides a different perspective on such well-explored trade-offs. (4) Finally, we demonstrate the effectiveness of decision-theoretic reasoning that dynamically schedules which frame to process at what time. Our analysis reveals the paradox that latency is minimized by sometimes sitting idle and "doing nothing"! Intuitively, it is sometimes better to wait for a fresh frame rather than to begin processing one that will soon become "stale".

3.2 Related Work

Latency evaluation Latency is a well-studied subject in computer vision. One school of research focuses on reducing the FLOPS of backbone networks [91, 249], while another school focuses on reducing the runtime of testing time algorithms [138, 142, 178]. We follow suit and create a latency-accuracy plot under our experiment setting (Fig. 3.2). While such a plot is suggestive of the trade-off for offline data processing (*e.g.*, archived video footage), it fails to capture the fact that *when the algorithm finishes processing, the surrounding world has already changed.* Therefore, we believe that existing plots do not reveal the streaming performance of these algorithms. Aside from computational latency, prior work has also investigated algorithmic latency [151], evaluated by running algorithms on a video in the *offline* fashion and measuring how many frames are required to detect an object after it appears. In comparison, our evaluation is done in the more realistic online real-time setting, and applies to any single-frame task, instead of just object detection.

Real-time evaluation There has not been much prior effort to evaluate vision algorithms in the real-time fashion in the research community. Notable exceptions include work on real-time tracking and real-time simultaneous localization and mapping (SLAM). First, the VOT2017 tracking benchmark specifically included a real-time challenge [116]. Its benchmark toolkit sends out frames at 20 FPS to participants' trackers and asks them to report back results before the next frame arrives. If the tracker fails to respond in time, the last reported result is used. This is equivalent to applying zero-order hold to trackers' outputs. In our benchmarks, we adopt a similar zero-order hold strategy, but extend it to a broader context of arbitrary single-frame tasks and allow for a more delicate interplay between detection, tracking, and forecasting. Second, the literature on real-time SLAM also considers benchmark evaluation under a "hard-enforced" real-time requirement [21, 59]. Our analysis suggests that hard-enforcement is too stringent of a formulation; algorithms should be allowed to run longer than the frame rate, but should still be scored on their ability to report the state of the world (*e.g.*, localized map) at frame rate.

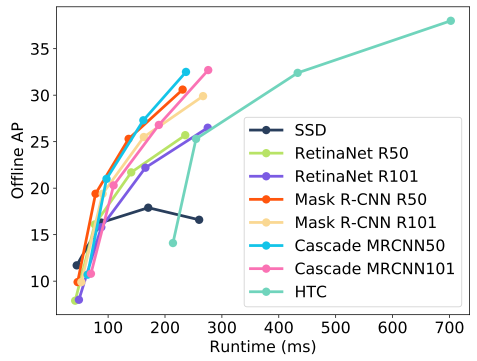


Figure 3.2: Prior art routinely explores the trade-off between detection accuracy versus runtime. We generate the above plot by varying the input resolution of each detection network. We argue that such plots are exclusive to offline processing and fail to capture latency-accuracy trade-offs in streaming perception. AP stands for average precision, and is a standard metric for object detection [139].

Progressive and anytime algorithms There exists a body of work on progressive and anytime algorithms that can generate outputs with lower latency. Such work can be traced back to classic research on intelligent planning under resource constraints [15] and flexible computation [90], studied in the context of AI with bounded rationality [186]. Progressive process-

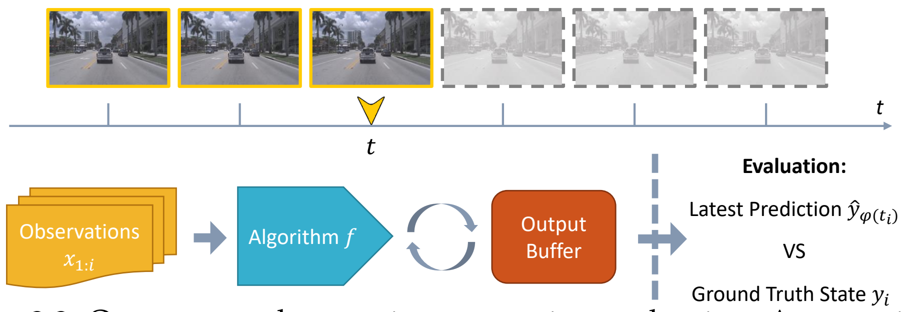


Figure 3.3: Our proposed streaming perception evaluation. A streaming algorithm f is provided with (timestamped) observations up until the current time t and refreshes an output buffer with its latest prediction of the current state of the world. At the same time, the benchmark constantly queries the output buffer for estimates of world states. Crucially, f must consider the amount of streaming observations that should be ignored while computation is occurring.

ing [260] is a paradigm that splits up an algorithm into sequential modules that can be dynamically scheduled. Often, scheduling is formulated as a decision-theoretic problem under resource constraints, which can be solved in some cases with Markov decision processes (MDPs) [259, 260]. Anytime algorithms are capable of returning a solution at any point in time [259]. Our work revisits these classic computation paradigms in the context of streaming perception, specifically demonstrating that classic visual tasks (like tracking and forecasting) naturally emerge in such bounded resource settings.

3.3 Proposed Evaluation

In the previous section, we have shown that existing latency evaluation fails to capture the streaming performance. To address this issue, here we propose a new method of evaluation. Intuitively, a streaming benchmark no longer evaluates a function, but a piece of executable code over a continuous time frame. The code has access to a sensor input buffer that stores the most recent image frame. The code is responsible for maintaining an output buffer that represents the up-to-date estimate of the state of the world (*e.g.*, a list of bounding boxes of objects in the scene). The benchmark examines this output buffer, comparing it with a ground truth stream of the actual world state (Fig. 3.3).

3.3.1 Formal definition

We model a data stream as a set of sensor observations, ground-truth world states, and timestamps, denoted respectively as $\{(x_i, y_i, t_i)\}_{i=1}^{T}$. Let f be a streaming algorithm to be evaluated. At any *continuous* time t, the algorithm f is provided with observations (and timestamps) that have appeared so far:

$$\{(x_i, t_i) | t_i \leq t\} \qquad \text{[accessible input at time } t\text{]} \quad (3.1)$$

We allow the algorithm f to generate an output prediction at *any time*. Let s_j be the timestamp that indicates when a particular prediction $\hat{y}_j$ is produced. The subscript j indexes over the N outputs generated by f over the entire stream:

$$\{(\hat{y}_j, s_j)\}_{j=1}^{N} \qquad \text{[all outputs by } f\text{]} \quad (3.2)$$

Note that this output stream is *not* synchronized with the input stream, and N has no direct relationship with T. Generally speaking, we expect algorithms to run slower than the frame rate ($N < T$).

We benchmark the algorithm f by comparing its most recent output at time t_i to the ground-truth y_i. We first compute the index of the most recent output:

$$\varphi(t) = \arg\max_j s_j < t \qquad \text{[real-time constraint]} \quad (3.3)$$

This is equivalent to the benchmark applying a *zero-order hold* for the algorithm's outputs to produce continuous estimation of the world states. Given an arbitrary single-frame loss L, the benchmark formally evaluates:

$$L_{\text{streaming}} = L(\{(y_i, \hat{y}_{\varphi(t_i)})\}_{i=1}^{T}) \qquad \text{[evaluation]} \quad (3.4)$$

By construction, the streaming loss above can be applied to *any* single-frame task that computes a loss over a set of ground truth and prediction pairs.

3.3.2 Emergent tracking and forecasting

At first glance, "instant" evaluation may seem unreasonable: the benchmark at time t queries the state at time t. Although x_t is made available to the algorithm, any finite-time algorithm cannot make use of it to generate its prediction. For example, if the algorithm takes time Δt to perform its computation, then to make a prediction at time t, it can only use data before time $t - \Delta t$. We argue that this is the *realistic* setting for streaming perception, both in biological and robotic systems. Humans and autonomous vehicles must react to the instantaneous state of the world when interacting with dynamic scenes. Such requirements strongly suggest that perception should be inherently predictive of the future. Our benchmark similarly "forces" algorithms to reason and forecast into the future, to compensate for the mismatch between the last processed observation and the present.

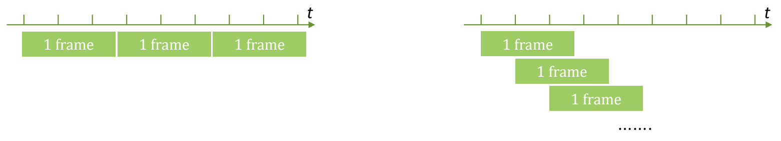
One may also wish to take into account the inference time of downstream actuation modules (that say, need to optimize a motion plan that will be executed given the perceived state of the world). It is straightforward to extend our benchmark to require algorithms to generate a forecast of the world state when the downstream module finishes its processing. For example, at time t the benchmark queries the state of the world at time $t + \eta$, where $\eta > 0$ represents the inference time of the downstream actuation module.

In order to forecast, the algorithms need to reason temporally through tracking (in the case of object detection). For example, constant velocity forecasting requires the tracks of each object over time in order to compute the velocity. Generally, there are two categories of trackers — post-hoc association [12] and template-based visual tracking [146]. In this paper, we refer them in short as "association" and "tracking", respectively. Association of previously computed detections can be made extremely lightweight with simple linking of bounding boxes (*e.g.*, based on the overlap). However, association does not make use of the image itself as done in (visual) tracking. We posit that trackers may produce better streaming accuracy for scenes with highly unpredictable motion. As part of emergent solutions to our streaming perception problem, we include both association and tracking in our experiments in the next section.

Finally, it is natural to seek out an end-to-end system that directly optimizes streaming perception accuracy. We include one such method in Appendix 3.C.2 to show that tracking and forecasting-based representations may also emerge from gradient-based learning.

3.3.3 Computational Constraints

Because our metric is runtime dependent, we need to specify the computational constraints to enable a fair comparison between algorithms. We first investigate a single GPU model (Fig. 3.4a), which is used for existing latency analysis in prior art. In the single GPU model, only a single GPU job (*e.g.*, detection or visual tracking) can run at a time. Such a restriction avoids multi-job interference and memory capacity issues. Note that a reasonable number of CPU jobs are allowed to run concurrently with the GPU job. For



(a) Single GPU model (b) Infinite GPU model

Figure 3.4: Two computation models considered in our evaluation. Each block represents an algorithm running on a device and its length indicates its runtime.

example, we allow bounding box association and forecasting modules to run on the CPU in Fig. 3.7.

Nowadays, it is common to have multiple GPUs in a single system. We investigate an *infinite* GPU model (Fig. 3.4b), with no restriction on the number of GPU jobs that can run concurrently. We implement this infinite computation model with *simulation*, described in the next subsection.

3.3.4 Challenges for practical implementation

While our benchmark is conceptually simple, there are several practical hurdles. First, we require high-frame-rate ground truth annotations. However, due to high annotation cost, most existing video datasets are annotated at rather sparse frame rates. For example, YouTube-VIS is annotated at 6 FPS, while the video data rate is 30 FPS [238]. Second, our evaluation is hardware dependent — the same algorithm on different hardware may yield different streaming performance. Such hardware-in-the-loop testing is commonplace in control systems [7] and arguably vital for embodied perception (which should by definition, depend on the agent's body!). Third, stochasticity in actual runtimes yields stochasticity in the streaming performance. Note that the last two issues are also prevalent in *existing* offline runtime analyses. Here we present high-level ideas for the solutions and leave additional details to Appendix 3.A.2 & 3.A.3.

Pseudo ground truth We explore the use of pseudo ground truth labels as a surrogate to manual high-frame-rate annotations. The pseudo labels are obtained by running state-of-the-art, *arbitrarily expensive* offline algorithms on each frame of a benchmark video. While the absolute performance numbers (when benchmarked on ground truth and pseudo ground truth labels) differ, we find that the rankings of algorithms are remarkably stable. The Pearson correlation coefficient of the scores of the two ground truth sets is 0.9925, suggesting that the real score is literally a linear function of the pseudo score. Moreover, we find that offline pseudo ground truth could also be used to self-supervise the training of streaming algorithms.

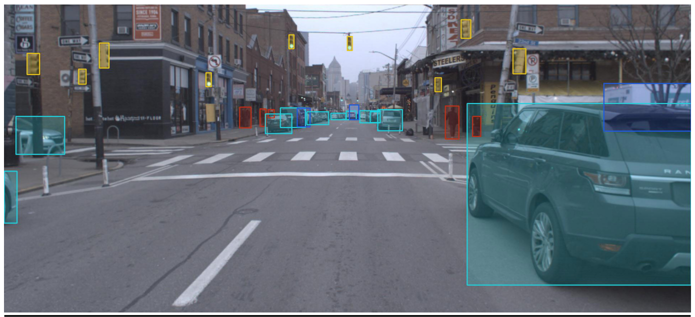
Simulation While streaming performance is hardware dependent, we now demonstrate that the benchmark can be evaluated on simulated hardware. In simulation, the benchmark assigns a runtime to each module of the algorithm, instead of measuring the wall-clock time. Then based on the assigned runtime, the simulator generates the corresponding output timestamps. The assigned runtime to each module provides a layer of abstraction on the hardware.

The benefit of simulation is to allow us to assess the algorithm performance on non-existent hardware, *e.g.*, a future GPU that is 20% faster or infinite GPUs in a single system. Simulation also allows our benchmark to inform practitioners about the *design* of future hardware platforms, *e.g.*, one can verify with simulation that 4 GPUs may be "optimal" (producing the same streaming accuracy as infinite GPUs).

Runtime-induced variance Due to algorithmic choice and system scheduling, different runs of the same algorithm may end up with different runtimes. This variation across runs also affects the overall streaming performance. Fortunately, we empirically find that such variance causes a standard deviation of up to 0.5% under our experiment setting. Therefore, we omit variance report in our experiments.

3.4 Solutions and Analysis

In this section, we instantiate our meta-benchmark on the illustrative task of object detection. While we show results on streaming detection, several key ideas also generalize to other tasks. An instantiation on instance segmentation can be found in Appendix 3.A.6. We first explain the setup and present the solutions and analysis. For the solutions, we first consider single-frame detectors, and then add forecasting and tracking one by one into the discussion. We focus on the most effective combination of detectors, trackers, and forecasters which we have evaluated, but include additional methods in Appendix 3.C.



Dataset	AP	AP$_L$	AP$_M$	AP$_S$	AP$_{50}$	AP$_{75}$
MS COCO	37.6	50.3	41.4	20.7	59.8	40.5
Argoverse-HD (Ours)	30.6	52.4	33.1	12.2	52.3	31.2

Figure 3.5: Comparison between our dataset and MS COCO [139]. Top shows an example image from Argoverse 1.1 [29], overlaid with our dense 2D annotation (at 30 FPS). Bottom presents results of Mask R-CNN [82] (ResNet 50) evaluated on the two datasets. AP$_L$, AP$_M$ and AP$_S$ denote AP for large, medium and small objects respectively. AP$_{50}$, AP$_{75}$ denote AP with IoU (Intersection over Union) thresholds at 0.5 and 0.75 respectively. We first observe that the APs are roughly comparable, showing that our annotation is reasonable in evaluating object detection performance. Second, we see a significant drop in AP$_S$ from COCO to ours, suggesting that the detection of small objects is more challenging in our setting. For self-driving vehicle applications, those small objects are important to identify when the ego-vehicle is traveling at a high speed or making unprotected turns.

3.4.1 Setup

We extend the publicly available video dataset Argoverse 1.1 [29] with our own annotations for streaming evaluation, which we name Argoverse-HD (High-frame-rate Detection). It contains diverse urban outdoor scenes from two US cities. We select Argoverse for its embodied setting (autonomous driving) and its high-frame-rate sensor data (30 FPS). We focus on the task of 2D object detection for our streaming evaluation. Under this setting, the state of the world y_t is a list of bounding boxes of the objects of interest. While Argoverse has multiple sensors, we only use the center RGB camera for simplicity. We collect our own annotations since the dataset does not provide dense 2D annotations[1]. For the annotations, we follow MS COCO [139] class definitions and format. For example, we include the "iscrowd" attribute for

[1]It is possible to derive 2D annotations from the provided 3D annotations, but we find that such derived annotations are highly imprecise.

ambiguous cases where each instance cannot be identified, and therefore the algorithms will not be wrongfully penalized. We use only a subset of 8 classes (from 80 MS COCO classes) that are directly relevant to autonomous driving: person, bicycle, car, motorcycle, bus, truck, traffic light, and stop sign. This definition allows us to evaluate off-the-shelf models trained on MS COCO. No training is involved in the following experiments unless otherwise specified. All numbers are computed on the validation set, which contains 24 videos ranging from 15–30 seconds each (the total number of frames is 15k). Figure 3.5 shows a comparison of our annotation with that of MS COCO. Additional comparison with other related datasets can be found in Appendix 3.A.4. All output timing is measured on a single Geforce GTX 1080 Ti GPU (a Tesla V100 counterpart is provided in Appendix 3.A.7).

3.4.2 Detection-Only

Table 3.1 includes the main results of using just detectors for streaming perception. We first examine the case of running a state-of-the-art detector — Hybrid Task Cascade (HTC) [32], both in the offline and the streaming settings. The AP drops significantly in the streaming setting. Such a result is not entirely surprising due to its high runtime (700ms). A commonly adopted strategy for real-time applications is to run a detector that is within the frame rate. We point out that this strategy may be problematic, since such a hard-constrained time budget results in poor accuracy for challenging tasks (Table 3.1 row 3–4). In addition, we find that many existing network implementations are optimized for throughput rather than latency, reflecting the bias of the community for offline versus online processing! For example, image pre-processing (*e.g.*, resizing and normalizing) is often done on CPU, where it can be pipelined with data pre-fetching. By moving it to GPU, we save 21ms in latency (for an input of size 960×600).

Our benchmarks allow streaming algorithms to *choose* which frames to process/ignore. Figure 3.6 compares a straight-forward schedule with our dynamic schedule (Alg. 1), which attempts to address temporal aliasing of the former. While spatial aliasing and quantization has been studied in computer vision [82], temporal quantization in the streaming setting has not been well explored. Noteably, it is difficult to pre-compute the optimal schedule because of the stochasticity of actual runtimes. Our proposed scheduling policy (Alg. 1) tries to minimize the expected temporal mismatch of the output stream and the data stream, thus increasing the overall streaming performance. Empirically, we find that it raises the AP for the detector (Table 3.1 row 7). We provide a *theoretical analysis* of the algorithm

ID	Method	Detector	AP	AP_L	AP_M	AP_S	AP_{50}	AP_{75}	Runtime
1	Accurate (Offline)	HTC @ s1.0	38.0	64.3	40.4	17.0	60.5	38.5	700.5
2	Accurate	HTC @ s1.0	6.2	9.3	3.6	0.9	11.1	5.9	700.5
3	Fast	RetinaNet R50 @ s0.2	5.5	14.9	0.4	0.0	9.9	5.6	36.4
4	Fast*	RetinaNet R50 @ s0.2	6.0	18.1	0.5	0.0	10.3	6.3	**31.2**
5	Optimized	Mask R-CNN R50 @ s0.5	10.6	21.2	6.3	0.9	22.5	8.8	77.9
6	Optimized*	Mask R-CNN R50 @ s0.5	**12.0**	**24.3**	**7.9**	**1.0**	**25.1**	**10.1**	56.7
7	+ Scheduling (Alg. 1)	Mask R-CNN R50 @ s0.5	13.0	26.6	9.2	1.1	26.8	11.1	56.7
8	+ Infinite GPUs	Mask R-CNN R50 @ s0.75	14.4	24.3	11.3	2.8	30.6	12.1	92.7

Table 3.1: Performance of existing detectors for streaming perception. The number after @ is the input scale (the full resolution is 1920×1200). * means using GPU for image pre-processing as opposed to using CPU in the off-the-shelf setting. The last column is the mean runtime of the detector for a single frame in milliseconds (mask branch disabled if applicable). The first baseline is to run an accurate detector (row 1), and we observe a significant drop of AP in the online real-time setting (row 2). Another commonly adopted baseline for embodied perception is to run a fast detector (row 3–4), whose runtime is smaller than the frame interval (33ms for 30 FPS streams). Neither of these baselines achieves good performance. Searching over a wide suite of detectors and input scales, we find that the optimal solution is Mask R-CNN (ResNet 50) operating at 0.5 input scale (row 5–6). In addition, our scheduling algorithm (Alg. 1) boosts the performance by 1.0/2.3 for AP/AP_L (row 7). In the hypothetical infinite GPU setting, a more expensive detector yields better trade-off (input scale switching from 0.5 to 0.75, almost doubling the runtime), and it further boosts the performance to 14.4 (row 8), which is the optimal solution achieved by just running the detector. Simulation suggests that 4 GPUs suffice to maximize streaming accuracy for this solution.

Algorithm 1: Shrinking-tail policy

```
1: Given finishing time $s$ and algorithm runtime $r$ in the unit of frames
   (assuming $r > 1$), this policy returns whether the algorithm should
   wait for the next frame
2: Define tail function $\tau(t) = t - \lfloor t \rfloor$
3: **return** $[\tau(s + r) < \tau(s)]$ (Iverson bracket)
```

and additional empirical results for a wide suite of detectors in Appendix 3.B.1. Note that Alg. 1 is by construction task agnostic (not specific to object detection).

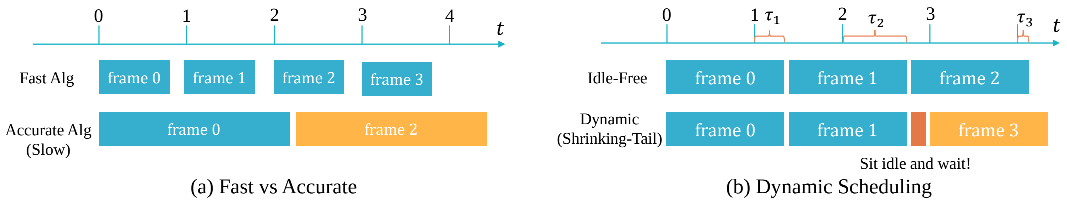


Figure 3.6: Algorithm scheduling for streaming perception with a single GPU. (a) A fast detector finishes processing the current frame before the next frame arrives. An accurate (but slow) detector cannot process every frame due to high latency. In this example, frame 1 is skipped. Note that the goal of streaming perception is not to process every frame but to produce accurate state estimates in a timely manner. (b) One straight-forward schedule is to simply process the latest available frame upon the completion of the previous processing (idle-free). However, if latest available frame will soon become stale, it might be better to idle and wait for a fresh frame (our dynamic schedule, Alg. 1). In this illustration, Alg. 1 determines that frame 2 will soon become stale and decides to wait (visualized in red) for frame 3 by comparing the tails τ_2 and τ_3.

3.4.3 Forecasting

Now we expand our solution space to include forecasting methods. We experimented with both constant velocity models and first-order Kalman filters. We find good performance with the latter, given a small modification to handle asynchronous sensor measurements (Fig. 3.7). The classic Kalman filter [104] operates on uniform time steps, coupling prediction and correction updates at each step. In our case, we perform correction updates only when a sensor measurement is available, but predict at every step. Second, due to frame-skipping, the Kalman filter should be time-varying (the transition and the process noise depend on the length of the time interval, details can be found in Appendix 3.B.2). Association for bounding boxes across frames is required to update the Kalman filter, and we apply IoU-based greedy matching. For association and forecasting, the computation involves only bounding box coordinates and therefore is very lightweight ($<$ 2ms on CPU). We find that such overhead has little influence on the overall AP. The results are summarized in Table 3.2.

Streamer (meta-detector) Note that our dynamic scheduler (Alg. 1) and asynchronous Kalman forecaster can be applied to *any* off-the-shelf detector, regardless of its underlying latency (or accuracy). This means that we

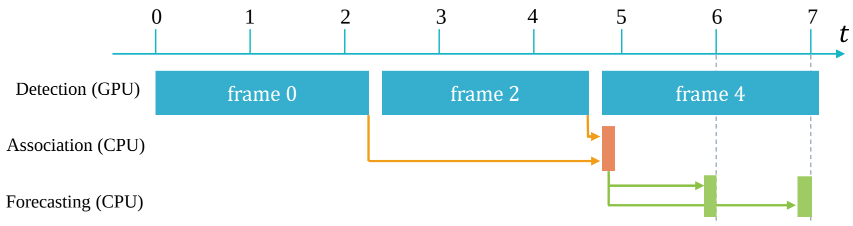


Figure 3.7: Scheduling for association and forecasting. Association takes place immediately after a new detection result becomes available, and it links the bounding boxes in two consecutive detection results. Forecasting takes place right before the next time step and it uses an asynchronous Kalman filter to produce an output as the estimation of the current world state. By default, the prediction step also updates internal states in the Kalman filter and is always called before the update step. In our case, we perform multiple update-free predictions (green blocks) until we receive a frame result.

ID	Method	AP	AP_L	AP_M	AP_S	AP_{50}	AP_{75}
1	Detection + Scheduling + Association + Forecasting	16.7	39.9	14.9	1.2	31.2	16.0
2	+ Re-optimize Detection (s0.5 $\rightarrow$ s0.75)	17.8	33.3	16.3	3.2	35.2	16.5
3	+ Infinite GPUs	20.3	38.5	19.9	4.0	39.1	18.9

Table 3.2: Streaming perception with joint detection, association, and forecasting. Association is done by IoU-based greedy matching, while forecasting is done by an asynchronous Kalman filter. First, we observe that forecasting greatly boosts the performance (from Table 3.1 row 7's 13.0 to row 1's 16.7). Also, with forecasting compensating for algorithm latency, it is now desirable to run a more expensive detector (row 2). Searching again over a large suite of detectors after adding forecasting, we find that the optimal detector is still Mask R-CNN (ResNet 50), but at input scale 0.75 instead of 0.5 (runtime 93ms and 57ms)

can assemble these modules into a *meta-detector* – which we call Streamer – that converts any detector into a streaming detection system that reports real-time detections at an arbitrary framerate. Appendix 3.B.4 evaluates the improvement in streaming AP across 80 different settings (8 detectors × 5 image scales × 2 compute models), which vary from 4% to 80% with an average improvement of 33%.

ID	Method	AP	AP$_L$	AP$_M$	AP$_S$	AP$_{50}$	AP$_{75}$
1	Detection + Visual Tracking	12.0	29.7	11.2	0.5	23.3	11.3
2	+ Forecasting	13.7	38.2	14.2	0.5	24.6	13.6
3	+ Re-optimize Detection (s0.5 $\rightarrow$ s0.75)	16.5	31.0	14.5	2.8	33.4	14.8
4	+ Infinite GPUs w/o Forecasting	14.4	24.2	11.2	2.8	30.6	12.0
5	+ Forecasting	20.1	38.3	19.7	3.9	38.9	18.7
6	Detection + Simulated Fast Tracker (2x) + Forecasting + Single GPU	19.8	39.2	20.2	3.4	38.6	18.1

Table 3.3: Streaming perception with joint detection, visual tracking, and forecasting. We see that initially visual trackers do not outperform simple association (Table 3.2) with the corresponding setting in the single GPU case. But that is reversed if the tracker can be optimized to run faster (2x) while maintaining the same accuracy (row 6). Such an assumption is not unreasonable given the fact that the tracker's job is as simple as updating locations of previously detected objects.

3.4.4 Visual tracking

Visual tracking is an alternative for low-latency inference, due to its faster speed than a detector. For our experiments, we adopt the state-of-the-art multi-object tracker [11] (which is second place in the MOT'19 challenge [50] and is open sourced), and modify it to only track previously identified objects to make it faster than the base detector (see Appendix 3.B.3). This tracker is built upon a two-stage detector and for our experiment, we try out the configurations of Mask R-CNN with different backbones and with different input scales. Also, we need a scheduling scheme for this detection plus tracking setting. For simplicity, we only explored running detection at fixed strides of 2, 5, 15, and 30. For example, stride 30 means that we run the detector once and then run the tracker 29 times, with the tracker getting reset after each new detection. Table 3.3 row 1 contains the best configuration over backbone, input scale, and detection stride.

3.5 Discussion

Streaming perception remains a challenge Our analysis suggests that streaming perception involves careful integration of detection, tracking, forecasting, and dynamic scheduling. While we present several strong solutions for streaming perception, the gap between the streaming performance and the offline performance remains significant (20.3 versus 38.0 in AP). This suggests that there is considerable room for improvement by building a better detector, tracker, forecaster, or even an end-to-end model that blurs boundary of these modules.

Formulations of real-time computation Common folk wisdom for real-time applications like online detection requires that detectors run within the sensor frame rate. Indeed, classic formulations of anytime processing require algorithms to satisfy a "contract" that they will finish under a compute budget [259]. Our analysis suggests that this view of computation might be too myopic as evidenced by contemporary robotic systems [170]. Instead, we argue that the sensor rate and compute budget should be seen as design choices that can be tuned to optimize a downstream task. Our streaming benchmark allows for such a global perspective.

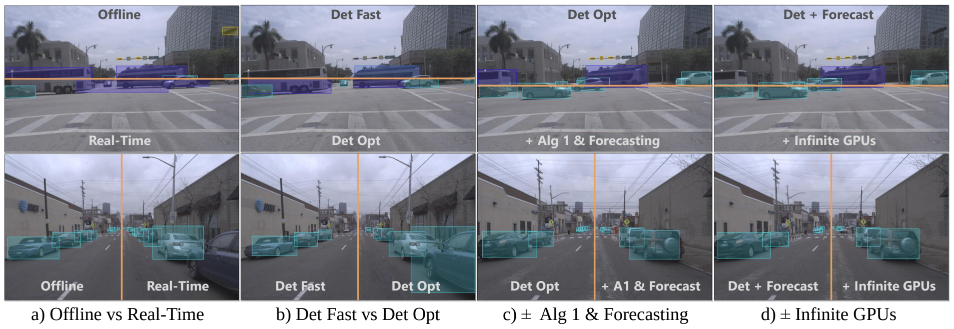


Figure 3.8: Qualitative results. Video results can be found on the project website [Link].

Generalization to other tasks By construction, our meta-benchmark and dynamic scheduler (Alg. 1) are not restricted to object detection. We illustrate such generalization with an additional task of instance segmentation (Fig. 3.9). However, there are several practical concerns that need to be addressed. Densely annotating video frames for instance segmentation is almost prohibitively expensive. Therefore, we adopt offline pseudo ground truth (Section 3.3.4) to evaluate streaming performance. Another concern is that the forecasting module is task-specific. In the case of instance segmentation, we implement it as forecasting the bounding boxes and then warping the masks accordingly. Please refer to Appendix 3.A.6 for the complete streaming instance segmentation benchmark.

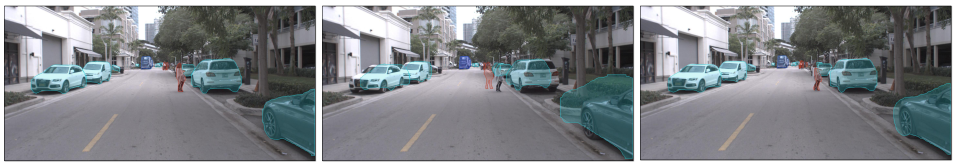


Figure 3.9: Generalization to instance segmentation. (a) The offline pseudo ground truth we adopt for evaluation is of high quality. (b) A similar latency pattern can be observed for instance segmentation as in object detection. (c) Forecasting for instance segmentation can be implemented as forecasting the bounding boxes and then warping the masks accordingly.

3.6 Conclusion and Future Work

We introduce a meta-benchmark for systematically converting any single-frame task into a streaming perception task that naturally trades off computation between multiple modules (*e.g.*, detection versus tracking). We instantiate this meta-benchmark on tasks of object detection and instance segmentation. In general, we find online perception to be dramatically more challenging than its offline counterpart, though significant performance can be recovered by incorporating forecasting. We use our analysis to develop a simple meta-detector that converts any detector (with any internal latency) into a streaming perception system that can operate at any frame rate dictated by a downstream task (such as a motion planner). We hope that our analysis will lead to future endeavor in this under-explored but crucial aspect of real-time embodied perception. For example, streaming benchmarks can be used to motivate attentional processing; by spending more compute only on spatially [68] or temporally [159] challenging regions, one may achieve even better efficiency-accuracy tradeoffs.

3.A Appendix A: Benchmark Details

3.A.1 Additional Discussion on the Benchmark Definition

In Section 3.3.1, we defined our benchmark as evaluation over a discrete set of frames. One might point out that a continuous definition is more consistent with the notion of estimating the state of the world at all time instants for streaming perception. First, we note that it is possible to define a continuous-time counterpart, where the ground truth can be obtained via polynomial interpolation and the algorithm prediction can be represented as a function of time (e.g., simply derived from extrapolating the discrete output). Also in Eq 3.4, the aggregation function (implicit in L) could be integration. However, our choice of a discrete definition is mainly for two reasons: (1) we believe a high-frame-rate data stream is able to approximate the continuous evaluation; (2) most existing single-frame metrics (L, e.g., average-precision) is defined with a discrete set of input and we prefer that our streaming metric is compatible with these existing metrics.

3.A.2 Pseudo Ground Truth

We use manually obtained ground-truth for bounding-box-based object detection. As we point out in the main text, one could make use of pseudo ground truth by simply running an (expensive but accurate) off-line detector to generate detections that could be used to evaluate on-line streaming detectors.

Here, we analyze the effectiveness of pseudo ground truth detection as a proxy for ground-truth. We adopt the state-of-the-art detector — Hybrid Task Cascade (HTC) [32] for computing the offline pseudo ground truth. As shown in Table 3.1, this offline detector dramatically outperforms all real-time streaming methods by a large margin. As shown in the main text, pseudo-streaming AP correlates extraordinarily well with ground-truth-streaming AP, with a normalized correlation coefficient of 0.9925. This suggests that pseudo ground truth can be used to rank streaming perception algorithms.

We emphasize that since we have constructed Argoverse-HD by deliberately annotating high frame rate bounding boxes, *we use real ground truth for evaluating detection performance*. However, obtaining such high-frame-rate annotations for instance segmentation is expensive. Hence we make use of pseudo ground-truth instance masks (provided by HTC) to benchmark streaming instance segmentation (Section 3.A.6).

3.A.3 Simulation

In true hardware-in-the-loop benchmarking, the output timestamp s_j is simply the wall-clock time at which an algorithm produces an output. While we hold this as the gold-standard, one can dramatically simplify benchmarking by making use of simulation, where s_j is computed using runtimes of different modules. For example, s_j for a single-frame detector on a single GPU can be simulated by adding its runtime to the time when it starts processing a frame. Complicated perception stacks require considering runtimes of all modules (we model those that contribute > 1 ms) in order to accurately simulate timestamps.

Modeling runtime distribution Existing latency analysis [138, 142, 178] usually reports only the mean runtime of an algorithm. However, empirical runtimes are in fact *stochastic* (Fig. 3.10), due to the underlying operating system scheduling and even due to the algorithm itself (e.g., proposal-based detectors often take longer when processing a scene with many objects). Because scene-complexity is often correlated across time, runtimes will also be correlated (a long runtime for a given frame may also hold for the next frame).

We performed a statistical analysis of runtimes, and found that a *marginal* empirical distribution to work well. We first run the algorithm over the entire dataset to get the empirical distribution of runtimes. At test time, we randomly sample a runtime when needed from the empirical distribution, without considering the correlation across time. Empirically, we found that the results (streaming AP) from a simulated run is within the variance of a real run.

Simulation for non-existent hardware/algorithm Through simulation, our evaluation protocol does not directly depend on hardware, but on a collection of runtime distributions for different modules (known as a *runtime profile*). One thus has the freedom to alter the distributions. For example, we can simulate a faster algorithm simply by scaling down the runtime profile. Table 3.3, uses simulation to evaluate the streaming performance of a non-existent tracker that runs twice as fast as the actual implementation on-hand. The reduced runtime could have arisen from better hardware; one can run the benchmark on a Geforce GTX 1080 Ti GPU and simulate the performance on a Tesla V100 GPU. We find that Tesla V100 makes our detectors run 16% faster, implying we can scale runtime profiles accordingly. For example, Mask R-CNN R50 @ s0.5 produces a simulated-streaming AP

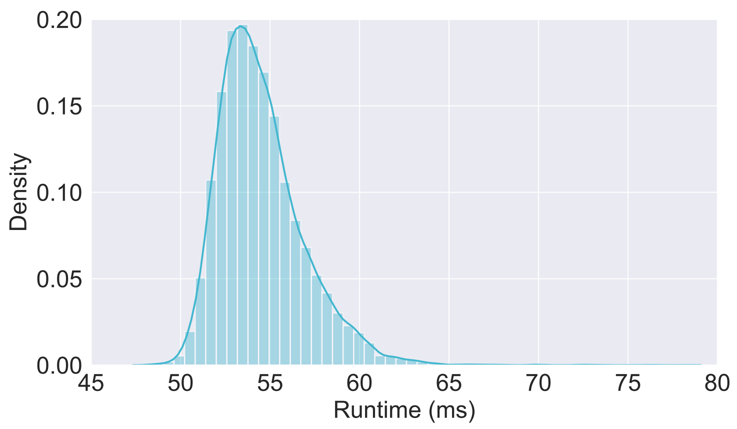


Figure 3.10: Runtime distribution for an object detector. Note that runtime is not constant, and this variance needs to be modeled in a simulation. This plot is obtained by running RetinaNet (ResNet 50) [138] on Argoverse 1.1 [29] with input scale 0.5.

of 12.652 while the real-streaming AP (on a V100) is 12.645, suggesting that effectivness of simulated benchmarking.

Infinite GPUs In simulation, we are not restricted by the number of physical GPUs present in a system. Therefore, we are able to perform analysis in the infinite GPU setting. In this setting, each detector or visual tracker runs on a different device without any interference with each other. Equivalently, we run a new GPU job on an existing device as long as it is idle. As a result, the simulation also provides information on how many GPUs are required for a particular infinite GPU experiment in practice (*i.e.*, the maximum number of concurrent jobs). We summarize the number of GPUs required for the experiments in the main text in Table 3.4. This implies that our streaming benchmark can be used to inform hardware design of *future* robotic platforms.

Runtime-induced variance As mentioned in the previous section, runtime is stochastic and has a variance up to 11.1% (standard deviation normalized by mean). Fortunately, such a variance does not transfer to the variance of our streaming metric. Empirically, we found that the variance of streaming AP of different runs (by varying the random seed) is around 0.5% for the same algorithm. In comparison, independent training runs of Mask R-CNN [82] on MS COCO [139] with the *same random seed* yield a variance of 0.3%

Table 3.4: Summary of the experiments in the infinite GPU settings (in the main text) and the number of GPUs needed in practice to achieve this performance (*i.e.*, the maximum number of concurrent jobs). This suggest that our simulation can also identify the optimal hardware configuration

Method	# of GPUs
Det (Table 3.1, row 8)	4
Det + Associate + Forecast (Table 3.2, row 3)	4
Det + Visual Track (Table 3.3, row 4)	9
Det + Visual Track + Forecast (Table 3.3, row 5)	9

on the AP (cudnn back-propagation is stochastic by default) [130]. Since the stochastic noise of streaming evaluation is at the same scale as CNN training, we ignore runtime-induced variance for our evaluation.

3.A.4 Dataset Annotation and Comparison

Based on the publicly available video dataset Argoverse 1.1 [29], we build our dataset with high-frame-rate annotations for streaming evaluation — Argoverse-HD (High-frame-rate Detection). One key feature is that the annotation follows MS COCO [139] standards, thus allowing direct evaluation of COCO pre-trained models on this self-driving vehicle dataset. The annotation is done at 30 FPS without any interpolation used. Unlike some self-driving vehicle datasets where only cars on the road are annotated [215], we also annotate background objects since they can potentially enter the drivable area. Of course, objects that are too small are omitted and our minimum size is 5×15 or 15×5 (based on the aspect ratio of the object). We outsourced the annotation job to Scale AI. In Table 3.5, we compare our annotation with existing datasets: DETRAC [225], KITTI-MOTS [215], MOTS [215], UAVDT [56], Waymo [199], and Youtube-VIS [238].

3.A.5 Experiment Settings

Platforms The CPU used in our experiments is Xeon Gold 5120, and the GPU is Geforce GTX 1080 Ti. The software environment is PyTorch 1.1 with CUDA 10.0.

Timing The setup which we time single-frame algorithms mimics the scenario in real-world applications. The offline pipeline involves several steps:

Table 3.5: Comparison of 2D video object detection datasets. For surveillance camera setups, the cameras are either stationary or have limited motion. For ego-vehicle setups, the scene dynamics evolve quickly, as (1) the ego-vehicle is traveling fast, and (2) other objects are much closer to the camera and thus have a higher speed in the image space. Our contributed dataset (annotation) is a high-frame-rate and high-resolution multi-class one compared to existing datasets

Name	Camera Setup	Image Res	Image FPS	Annot FPS	Classes	Boxes
DETRAC	Survelliance	960×540	30	6	4	1.21M
KITTI-MOTS	Ego-Vehicle	1242×375	10	10	2	46K
MOTS	Generic	1920×1080	30	30	2	30K
UAVDT	UAV Survelliance	1080×540	30	30	1	842K
Waymo	Ego-Vehicle	1920×1280	10	10	4	11.8M
Youtube-VIS	Generic	1280×720	30	6	40	131K
Argoverse-HD (Ours)	Ego-Vehicle	1920×1200	30	30	8	1.26M

loading data from the disk, image pre-processing, neural network forward pass, and result post-processing. Our timing excludes the first step of loading data from the disk. This step is mainly for dataset-based evaluation. In actual embodied applications, data come from sensors instead of disks. This is implemented by loading the entire video to the main memory before the evaluation starts. In summary, our timing (*e.g.*, the last column of Table 3.1) starts at the time when the algorithm receives the image in the main memory, and ends at the time when the results are available in the main memory (instead of in the GPU memory).

3.A.6 Alternate Task: Instance Segmentation

In the main text, we propose a meta-benchmark and mention that it can be instantiated with different tasks. In this section, we include *full benchmark evaluation* for streaming instance segmentation.

Instance segmentation is a more fine-grained task than object detection. This creates challenges for streaming evaluation as annotation becomes more expensive and forecasting is not straight-forward. We address these two issues by leveraging pseudo ground truth and warping masks according to the forecasted bounding boxes.

Another issue which we observed is that off-the-shelf pipelines are usually designed for benchmark evaluation or visualization. First, similar to object detection, we adopt GPU image pre-processing by default. Second, we found that more than 90% of the time within the mask head of Mask R-

Table 3.6: Instance segmentation overhead compared with object detection. This table lists runtimes of several methods with and without the mask head, and their differences are the extra cost which one has to pay for instance segmentation. All numbers are milliseconds except the scale column and the last column. The average overhead is 17ms or 13%

Method	Scale	w/o Mask	w/ Mask	Overhead	Overhead
	0.2	34.3	41.4	7.1	21%
	0.25	36.1	44.3	8.2	23%
Mask R-CNN ResNet 50	0.5	56.7	65.6	8.8	16%
	0.75	92.7	101.0	8.3	9%
	1.0	139.6	147.7	8.1	6%
	0.2	38.4	46.4	7.9	21%
	0.25	40.9	48.7	7.8	19%
Mask R-CNN ResNet 101	0.5	68.8	76.4	7.6	11%
	0.75	119.7	127.1	7.5	6%
	1.0	183.8	190.8	7.0	4%
	0.2	60.9	66.0	5.1	8%
	0.25	59.2	69.1	9.9	17%
Cascade MRCNN ResNet 50	0.5	80.0	95.4	15.3	19%
	0.75	118.1	133.8	15.7	13%
	1.0	164.6	181.9	17.3	10%
	0.2	66.4	71.0	4.6	7%
	0.25	65.4	75.2	9.7	15%
Cascade MRCNN ResNet 101	0.5	92.2	106.6	14.4	16%
	0.75	143.4	159.2	15.8	11%
	1.0	208.2	225.1	16.9	8%

CNN is spent on transforming masks from the RoI space to the image space and compressing them in a format to be recognized by the COCO evaluation toolkit. Clearly, compression can be disabled for streaming perception. We point out that mask transformation can also be disabled. In practice, masks are used to tell if a specific point or region contains the object. Instead of transforming the mask (which involves object-specific image resizing operations), we can transform the query points or regions, which is simply a linear transformation over points or control points. Therefore, our timing does not include RoI-to-image transformation or mask compression. Furthermore, this also implies that we do not pay an additional cost for masks in forecasting, since only the box coordinates are updated but the masks remain in the RoI space.

For the instance segmentation benchmark, we use the same dataset and the same method HTC [32] for the pseudo ground truth as for detection, and

Table 3.7: Streaming evaluation for instance segmentation. We find that *many of our observations for object detection still hold for instance segmentation*: (1) AP drops significantly when moving from offline to real time, (2) the optimal "sweet spot" is not the fastest algorithm but the algorithm with runtime more than the unit frame interval, and (3) both our dynamic scheduling and infinite GPUs further boost the performance. Note that the absolute numbers might appear higher than the tables in the main text since we use pseudo ground truth here

ID	Method	Detector	AP	AP_L	AP_M	AP_S	AP_{50}	AP_{75}	Runtime
1	Accurate (Offline)	Cascade MRCNN R50 @ s1.0	63.1	63.0	60.9	47.9	81.6	69.4	225.1
2	Accurate	Cascade MRCNN R50 @ s1.0	11.8	11.5	8.1	5.4	20.4	11.1	225.1
3	Fast	Mask R-CNN R50 @ s0.2	8.3	16.5	2.1	0.0	13.6	8.3	**41.4**
4	Optimized	Mask R-CNN R50 @ s0.5	**17.2**	**19.9**	**13.8**	**5.2**	**31.8**	**15.1**	65.6
5	+ Scheduling (Alg. 1)	Mask R-CNN R50 @ s0.5	18.3	21.4	14.9	5.8	33.5	16.4	65.5
6	+ Infinite GPUs	Mask R-CNN R50 @ s0.75	20.6	20.0	19.0	9.1	38.4	18.2	100.8

Table 3.8: Streaming evaluation for instance segmentation with forecasting. Despite that we only forecast boxes and warp masks accordingly, we still observe significant improvement from forecasting for mask AP. The optimized algorithm for row 1 is Mask R-CNN ResNet 50 @ s0.5, and for row 2 is Mask R-CNN ResNet 50 @ s0.75

ID	Method	AP	AP_L	AP_M	AP_S	AP_{50}	AP_{75}
1	Detection + Scheduling + Association + Forecasting	24.1	32.4	23.0	6.0	43.7	22.0
2	+ Infinite GPUs	29.2	30.7	30.2	11.4	53.0	26.7

we include 4 methods: Mask R-CNN [82] and Cascade Mask R-CNN [24] with ResNet 50 and ResNet 101 backbones. Since these are hybrid methods that produce both instance boxes and masks, we can measure the overhead of including masks as the difference between runtime with and without the mask head in Table 3.6. We find that the average overhead is around 13%. We include the streaming evaluation in Tables 3.7 and 3.8 (with forecasting).

3.A.7 Alternate Hardware: Tesla V100

In the main text, we propose a meta-benchmark and mention that it can be instantiated with different hardware platforms. In this section, we include *full benchmark evaluation* for streaming detection with Tesla V100 (a faster GPU than GTX 1080 Ti used in the main text).

While our benchmark is hardware dependent, the method of evaluation

Table 3.9: Performance of detectors for streaming perception on Tesla V100 (a faster GPU than the Geforce GTX 1080 Ti used in the main text). By comparing with Table 3.1 in the main text, we see that runtime is shortened and the AP is increased due to the boost of hardware performance. Different from Table 3.1, we only consider GPU image pre-processing here for simplicity. Interestingly, with additional computation power, Tesla V100 enables more expensive models like input scale 0.75 (row 4) and Cascade Mask R-CNN (row 5) to be the optimal configurations (detector and scale) under their corresponding settings. Note that the improvement from our dynamic scheduler is orthogonal to the boost from hardware performance

ID	Method	Detector	AP	AP_L	AP_M	AP_S	AP_{50}	AP_{75}	Runtime
1	Accurate (Offline)	HTC @ s1.0	38.0	64.3	40.4	17.0	60.5	38.5	338.0
2	Accurate	HTC @ s1.0	8.2	12.3	5.1	1.6	15.3	7.6	338.0
3	Fast	RetinaNet R50 @ s0.25	6.4	17.3	0.6	0.0	11.9	6.0	**43.3**
4	Optimized	Mask R-CNN R50 @ s0.75	13.0	22.2	9.5	**2.3**	**27.6**	10.9	72.1
5	+ Scheduling (Alg. 1)	Cascade MRCNN R50 @ s0.5	**14.0**	**28.8**	**9.9**	1.0	26.8	**12.2**	60.2
6	+ Infinite GPUs	Mask R-CNN R50 @ s1.0	15.9	24.1	13.2	4.9	34.2	13.3	98.8

Table 3.10: Streaming perception with joint detection, association, and forecasting on Tesla V100 (corresponding to Table 3.2 in the main text). We observe similar boost as in the detection only setting (Table 3.9). The "re-optimize detection" step finds that Mask R-CNN R50 @ s1.0 outperforms Cascade Mask R-CNN R50 @ s0.5 with forecasting (row2), and it also happens to be the optimal detector with infinite GPUs (row 3)

ID	Method	AP	AP_L	AP_M	AP_S	AP_{50}	AP_{75}
1	Detection + Scheduling + Association + Forecasting	18.2	42.7	16.1	1.1	30.9	17.7
2	+ Re-optimize Detection	**19.6**	**33.0**	**19.2**	**5.3**	**38.5**	**17.9**
3	+ Infinite GPUs	22.9	38.7	23.1	6.9	43.8	21.2

generalizes across hardware platforms, and our conclusions largely hold when the hardware environment changes. We follow the same setup as in the experiments in the main text, except that we use Tesla V100 from Amazon Web Services (EC2 instance of type `p3.2xlarge`). We provide the results for detection, forecasting, and tracking in Tables 3.9, 3.10, and 3.11, respectively. We see that *the improvement due to better hardware is largely orthogonal to the algorithmic improvement* proposed in the main text.

Table 3.11: Streaming perception with joint detection, visual tracking, and forecasting on Tesla V100 (corresponding to Table 3.3 in the main text). We find the similar conclusions that visual tracking with forecasting does not outperform association with forecasting in the single GPU case and achieves comparable performance in the infinite GPU case

ID	Method	AP	AP_L	AP_M	AP_S	AP_{50}	AP_{75}
1	Detection + Visual Tracking	12.6	21.5	9.0	2.2	27.1	10.5
2	+ Forecasting	**18.0**	**34.7**	**16.8**	**3.2**	**36.0**	**16.4**
3	+ Infinite GPUs w/o Forecasting	14.4	24.2	11.2	2.8	30.6	12.0
4	+ Forecasting	**22.8**	**38.6**	**23.0**	**6.9**	**43.7**	**21.0**

3.B Appendix B: Solution Details

3.B.1 Dynamic Scheduling

In the main text, we propose the dynamic scheduling algorithm (Alg. 1) to reduce temporal aliasing. Such an algorithm is counter-intuitive in that it minimizes latency by sometimes sitting idle. In this subsection, we provide additional theoretical analysis and empirical results for algorithm scheduling. We first introduce the framework to study algorithm scheduling for streaming perception. Next, we show theoretically that our dynamic scheduling outperforms naive idle-free scheduling for any constant runtime larger than the frame interval and any long-enough sequence length. Lastly, we verify empirically the superiority of our dynamic scheduling.

To study algorithm scheduling, we assume no concurrency (*i.e.*, a single job at a time) and that jobs are not interruptible. For notational simplicity, we assume a fixed input frame rate where frame x_i is the frame available at time $i \in \{0, \ldots, T-1\}$ (*i.e.*, zero-based indexing), and therefore i can be used to denote both frame index and time. *We assume that time (time axis, runtime, and latency) is represented in the units of the number of frames*. We also assume g to be a *single-frame* algorithm, and the streaming algorithm f is thus composed of g and a scheduling policy. No tracking or forecasting is used in the discussion below. Let k_j be the input frame index that was processed to generate output $o_j = (\hat{y}_j, s_j)$: if $\hat{y}_j = g(x_i)$, then $k_j = i$. We denote the runtime of g as r.

Definition (Temporal Mismatch) When the benchmark queries for the state of the world at frame i, the temporal mismatch is $\delta_i := i - k_j$, where $j = \arg\max_{j'} s_{j'} < i$. If there is no output available, $\delta_i := 0$. We denote the average temporal mismatch over the entire sequence as $\bar{\delta}$.

Intuitively, the temporal mismatch measures the latency of a streaming algorithm f in the unit of the number of frames (Fig. 3.11). This latency is typically higher than the runtime of the single-frame algorithm g itself due to the blocking effect of consecutive execution blocks. For example, in Figure 3.11, although runtime $r < 2$, the average mismatch $\bar{\delta} = 15/7 > 2$ for $T = 7$. Note that we define $\delta_i := 0$ if there is no output available. To avoid the degenerate case where an algorithm processes nothing and yields a zero cumulative temporal mismatch, we assume that all schedules start processing the first frame immediately at $t = 0$.

MDP Naive idle-free scheduling processes the next available frame immediately after the previous execution is finished. However, a scheduler can *choose* when and which frames to process. Selection among such choices over the data sequence can be modeled as a decision policy under a *Markov decision process* (*MDP*). An MDP formulation allows one to compute the expected future cumulative mismatch for a given policy under stochastic runtimes r. In theory, one may also be able to compute the optimal schedule (that minimizes expected cumulative mismatches) through policy search algorithms. However, Figure 3.10 shows that practical runtime profiles have low variance and are unimodal. If one assumes that runtimes are deterministic and fixed at a constant value, we will now show that our shrinking-tail policy outperforms idle-free over a range of runtimes r and sequence lengths T. We believe that constant runtime is a reasonable assumption for our setting, and empirically verify so after our theoretical analysis.

Pattern analysis Crucially, constant runtimes ensure that all transitions are deterministic, allowing for a global view of the sequence. Our key observation is that the *global sequence will contain repeating mismatch patterns*. Analysis of one such pattern sheds light on the cumulative mismatch of the entire sequence. For example, $r = 1.5$ under idle-free repeats itself every 2 processing blocks. However, different patterns emerge for different values of r and for different policies. We assume that $r > 1$ to avoid the trivial schedule where an algorithm consistently finishes before the next frame arrives. We

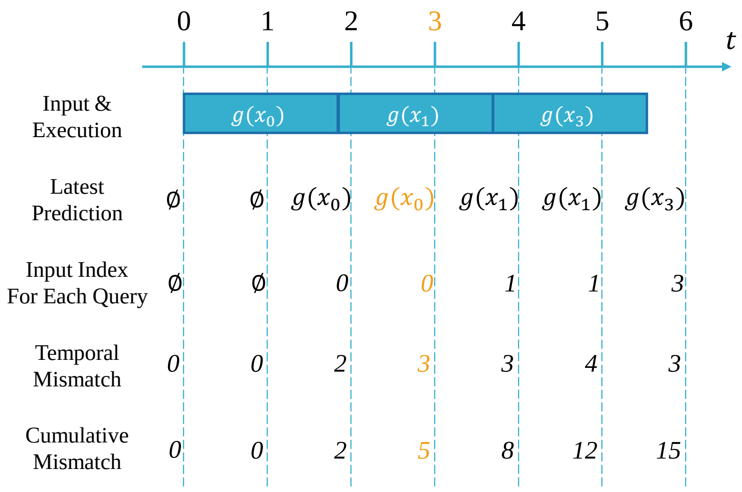


Figure 3.11: Temporal mismatch for single-frame algorithms. Take $t = 3$ (query index $i = 3$) as an example (highlighted in orange): when the benchmark queries for y_3, the latest prediction is $g(x_0)$, whose input index is 0, thus leading to a temporal mismatch of 3 (frames).

write $\bar{\delta}_{\text{if}}$ and $\bar{\delta}_{\text{st}}$ for the average temporal mismatch $\bar{\delta}$ for the idle-free and shrinking-tail policies, respectively. Our analysis is based on the concept of *tail*: $\tau(t) := t - \lfloor t \rfloor$. We denote $\tau(r)$ as τ_r for short. Note that the integral part of runtime does not contribute to the temporal quantization effect, and we thus focus on the discussion of $1 < r \leq 2$ for simplicity. We split our analysis into 3 different cases: $r = 2$, $1.5 \leq r < 2$, and $1 < r < 1.5$.

Case 1 The first is a special case where $\tau_r = 0$. It can be easily verified that idle-free is equivalent to shrinking-tail, and thus $\bar{\delta}_{\text{st}} = \bar{\delta}_{\text{if}}$.

Case 2 Now we inspect the case with $1.5 \leq r < 2$. Since $\tau(2r) < 0.5 \leq \tau(r)$, the shrinking-tail policy will output true (waiting) after processing the first frame. The waiting aligns the execution again with the integral time step, and thus for the subsequent processing blocks, it also outputs true (waiting). In summary, shrinking-tail always outputs true in this case, and its pattern in mismatch is agnostic to the specific runtime r (Fig. 3.12). Let $\bar{\delta}^r$ denote $\bar{\delta}$ with runtime r, then we can draw the conclusion that $\bar{\delta}^{r_1}_{\text{st}} = \bar{\delta}^{r_2}_{\text{st}}$ for $\lfloor r_1 \rfloor = \lfloor r_2 \rfloor$, $\tau(r_1) \geq 0.5$, and $\tau(r_2) \geq 0.5$.

We then focus on a particular case of $r = 1.5$. As shown in Figure 3.13, idle-free repeats itself in a period of 3 frames, and shrinking-tail repeats itself in a period of 2 frames. Together, they form a joint pattern that repeats itself in a period of 6 frames (their least common multiple). The diagram shows that within each common period, the difference of cumulative mismatch between idle-free and shrinking-tail is increased by 1. And it is the same for all common periods. Therefore, if $T = 6n + 1$ for some positive integer n (intuitively, the entire sequence is a multiple of several common

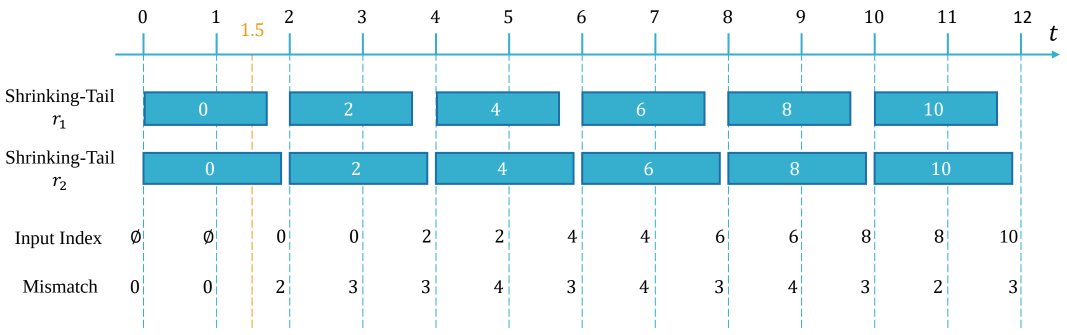


Figure 3.12: Mismatch is the same for the shrinking-tail policy with different runtime r_1 and r_2 as long as $\lfloor r_1 \rfloor = \lfloor r_2 \rfloor$, $\tau(r_1) \geq 0.5$, and $\tau(r_2) \geq 0.5$.

periods), $\bar{\delta}_{\text{st}}^{1.5} < \bar{\delta}_{\text{if}}^{1.5}$. Additionally, Figure 3.13 enumerates all possible cases, where the sequence ends before a common period is over or in the middle of a common period. All these cases have $\bar{\delta}_{\text{st}}^{1.5} \leq \bar{\delta}_{\text{if}}^{1.5}$.

Next, it is straightforward to see thatthe cumulative mismatch will not decrease if one increases the runtime r of g: $\bar{\delta}^{r_1} \leq \bar{\delta}^{r_2}$ if $r_1 \leq r_2$. Therefore, for $1.5 \leq r < 2$, we have

$$\bar{\delta}_{\text{st}}^{r} = \bar{\delta}_{\text{st}}^{1.5} \leq \bar{\delta}_{\text{if}}^{1.5} \leq \bar{\delta}_{\text{if}}^{r} \tag{3.5}$$

Case 3 The last case with $1 < r < 1.5$ (*i.e.*, $\tau_r < 0.5$) is more complicated than previous cases because the underlying repeating pattern never exactly repeats itself. To address this issue, we must introduce several new concepts to characterize such near-repeating patterns. We first observe a special type of execution block:

Definition (Shrinking-Tail Block) Denoting the start and the end time of an execution block as t_1 and t_2, a *shrinking-tail block* is an execution block such that $\tau(t_1) > \tau(t_2)$. As shown in Figure 3.14, a shrinking-tail block increases temporal mismatch.

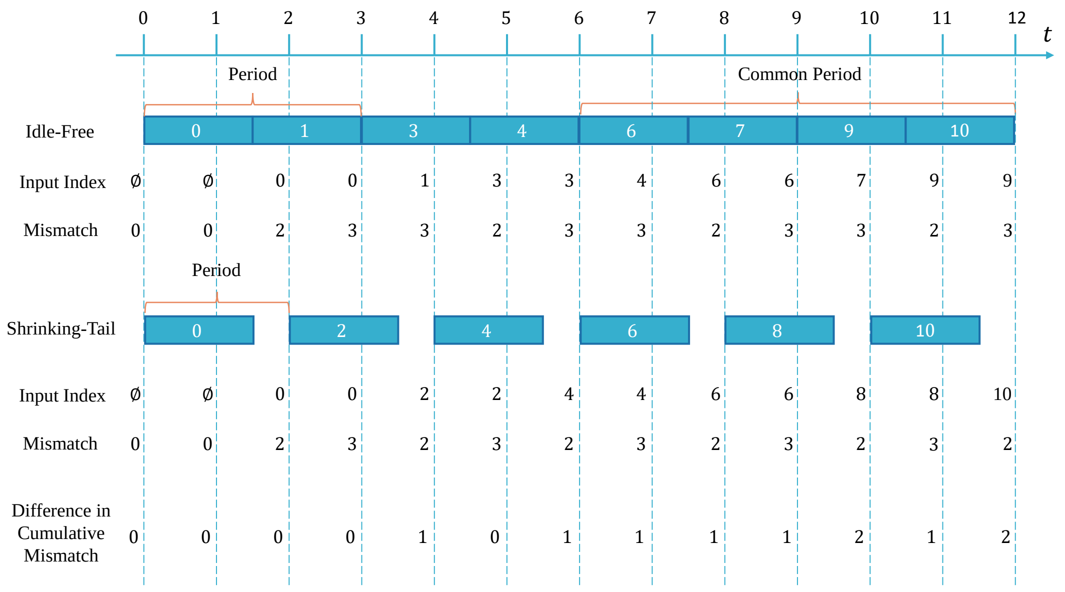


Figure 3.13: For $r = 1.5$, shrinking-tail achieves less cumulative mismatch than idle-free. Note that each policy has its own repeating period and shrinking-tail always achieves 1 less cumulative mismatch within each common period.

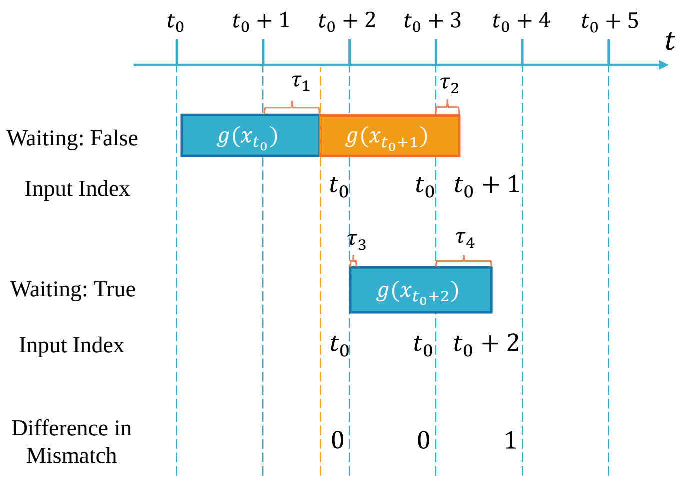


Figure 3.14: A shrinking-tail execution block (orange) increases temporal mismatch.

Definition (**Shrinking-Tail Cycle**) A sequence of execution blocks can be divided into segments by a shrinking-tail block or an idle gap. A *shrinking-tail cycle* is a set of queries covered by the segment between these dividers. Specifically, the cycle starts from the 0-th query, the last query of a shrinking-tail block, or the query at the end of an idle gap. The cycle ends either when the sequence ends or the next cycle starts. The length of a cycle is the number of queries it covers.

As shown in Figure 3.15, shrinking-tail cycles are small segments of the entire sequence and they may have different lengths. Note that the definitions of both shrinking-tail block and cycle are agnostic to r, but we only refer to them during our discussion for $1 < r < 1.5$. Instead of comparing $\bar{\delta}$ for idle-free and shrinking-tail directly, we compare them for each cycle (denoted as $\bar{\delta}_{\text{if}}^{(c)}$ and $\bar{\delta}_{\text{st}}^{(c)}$ respectively). First, we observe that a shrinking-tail cycle starts with either a shrinking-tail block or an idle gap and ends with consecutive tail-increasing blocks. Second, we observe that most queries have a mismatch of 2 for both policies (*e.g.*, Cycle 2's queries 20 to 21 and Cycle 4's queries 18 to 19 in Fig. 3.15), and that the second query in a cycle is always 3 due to having a shrinking-tail block or an idle-gap before it. This is the rounding effect when adding multiple fractional numbers. The difference between the two policies is thus the mismatch of the first query. For $1 < r < 1.5$, the first query of idle-free has a mismatch of 3, while shrinking-tail has a mismatch of 2. Intuitively, given that the majority of queries are with mismatch 2, the number of queries with mismatch 3 determines the

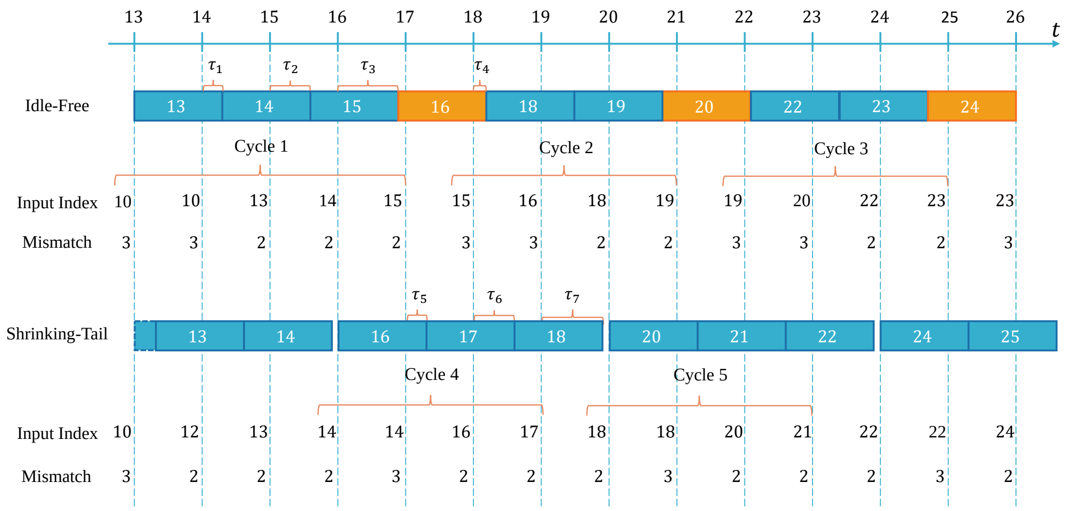


Figure 3.15: Shrinking-tail cycle. Intuitively, blocks within each shrinking-tail cycle has tails increasing ($\tau_1 < \tau_2 < \tau_3$ and $\tau_5 < \tau_6 < \tau_7$). It ends when the tail decreases or there is an idle gap, and thus the tail "shrinks".

relationship between $\bar{\delta}^{(c)}$: $\bar{\delta}_{\text{st}}^{(c)} < \bar{\delta}_{\text{if}}^{(c)}$. Therefore, when the sequence length is long enough, the policy with a smaller $\bar{\delta}^{(c)}$ leads to a smaller overall cumulative mismatch.

Now, we present a more formal analysis on the above statement. To quantify the cycle patterns, we first quantify the number of consecutive tail-increasing blocks. Let the number of consecutive tail-increasing blocks be a and the tail of the first block covered by the cycle be b (in the case where the first block starts after an idle gap, we define b to be 0). We first observe that $a = \max\{a' | a'\tau_r + b \leq 1, a' \in \mathcal{N}\} = \lfloor \frac{1-b}{\tau_r} \rfloor$. Also, b has its own range for each policy. For idle-free, $0 \leq b < \tau_r$, and for shrinking-tail, $b = 0$. Taking Figure 3.15 for example ($\tau_r = 0.3$), Cycle 1 has $a = 3$ and $b = 0$, Cycle 2 has $a = 2$ and $b = \tau_4$, and Cycle 4 has $a = 3$ and $b = 0$.

Since a might vary from cycle to cycle, we introduce a reference quantity that is constant and can be used to measure the length of cycles. Let a_0 be the a when $b = 0$, *i.e.*, $a_0 = \lfloor \frac{1}{\tau_r} \rfloor$, and c be the length of a cycle. For idle-free policy, $c = a_0 + 2$ or $a_0 + 1$. The variable length in cycles is due to variable b between cycles. When $b \leq 1 - a_0\tau_r$, we have the first type of cycle with length $a_0 + 2$ (denoted as c_1); when $b > 1 - a_0\tau_r$, we have the second type of cycle of length $a_0 + 1$ (denoted as c_2). The starting cycle in a sequence is always the first type, while the ensuing cycles can be either the first or second type. Note that it is possible that all cycles are the first type. For example, when $r = 1.25$, each cycle resets itself and $b = 0$ for all cycles. For shrinking-tail policy, each cycle resets itself (whose length denoted as c_3). Note that c_1, c_2, c_3 denotes the length of the 3 *types* of cycles and Cycle 1, 2, 3, ... in the figures denote specific cycle *instances*. From the above analysis, we can see

$$c_1 = a_0 + 2, \qquad c_2 = a_0 + 1, \qquad c_3 = a_0 + 1. \tag{3.6}$$

$$\bar{\delta}_{\text{if}}^{(c_1)} = \frac{2}{a_0+2} + 2, \qquad \bar{\delta}_{\text{if}}^{(c_2)} = \frac{2}{a_0+1} + 2, \qquad \bar{\delta}_{\text{st}}^{(c_3)} = \frac{1}{a_0+1} + 2. \tag{3.7}$$

Therefore,

$$\bar{\delta}_{\text{st}}^{(c_3)} < \bar{\delta}_{\text{if}}^{(c_1)} < \bar{\delta}_{\text{if}}^{(c_2)} \tag{3.8}$$

Next, we explain how to infer the relationship between $\bar{\delta}$ from that between $\bar{\delta}^{(c)}$. To analyze the mismatch of the whole sequence, we need to inspect the boundary cases at the start and the end of the sequence, where the cycle-based analysis might not hold. As shown in Figure 3.16, the first cycles for both policies have different mismatch patterns due to empty detection at the first two queries. Compared to regular cycles in Figure 3.15, the first cycle has 6 and 5 less total mismatch for idle-free and shrinking-tail policy respectively. Let m_1, m_2, and m_3 be the number of complete cycles of type c_1, c_2, and c_3 in a sequence, respectively, d be the number of residual

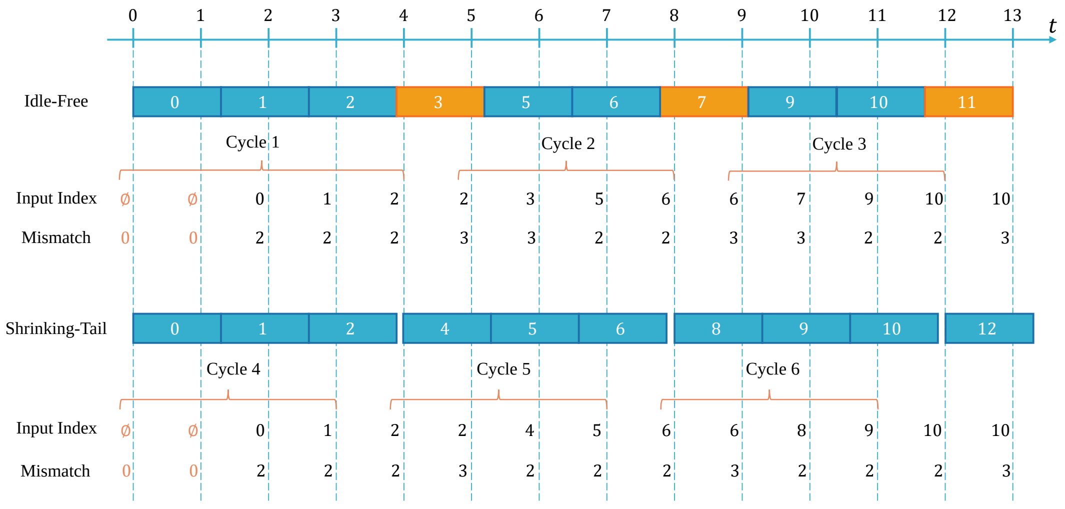


Figure 3.16: The first cycles for both policies have different mismatch patterns.

queries at the end of the sequence that do not complete a cycle, and e be the total mismatch of these d queries, then we have

$$T = m_1c_1 + m_2c_2 + d_{\text{if}} \tag{3.9}$$
$$T = m_3c_3 + d_{\text{st}} \tag{3.10}$$
$$\bar{\delta}_{\text{if}} = (m_1c_1\bar{\delta}_{\text{if}}^{(c_1)} + m_2c_2\bar{\delta}_{\text{if}}^{(c_2)} - 6 + e_{\text{if}})/T \tag{3.11}$$
$$\bar{\delta}_{\text{st}} = (m_3c_3\bar{\delta}_{\text{st}}^{(c_3)} - 5 + e_{\text{st}})/T \tag{3.12}$$

Note that the above holds only when $m_1 \geq 1$ and $m_3 \geq 1$ (the sequence is at least one cycle long for both policies). If T is smaller or equal to one cycle, the two policies are equivalent and $\delta_{\text{if}} = \delta_{\text{st}}$. When T is large enough (*e.g.*, $T \to \infty$), the $\bar{\delta}^{(c)}$ terms dominate Eq 3.11 and Eq 3.12, and due to Eq 3.7, we have $\bar{\delta}_{\text{st}} < \bar{\delta}_{\text{if}}$, which shows that the shrinking-tail policy is superior. Formally, when $T > C(r)$, where $C(r)$ is some constant depending on r, $\bar{\delta}_{\text{st}} < \bar{\delta}_{\text{if}}$.

Summary of the theoretical analysis Considering all 3 cases, we can draw the conclusion that $\bar{\delta}_{\text{st}} \leq \bar{\delta}_{\text{if}}$ when T is large enough (greater than $C(r)$ if $\tau_r < 0.5$, and no requirement otherwise). By achieving less average mismatch, shrinking-tail outperforms idle-free.

Practical Performance of Dynamic Scheduling

We apply our dynamic schedule (Alg. 1) to a wide suite of detectors under the same settings as our main experiments and summarize the results in Table 3.12. In practice, runtime is stochastic due to complicated software and hardware scheduling or running an input adaptive model, but we find the theoretical results obtained under constant runtime assumption generalizes to most of the practical cases under our experiment setting.

3.B.2 Additional Details for Forecasting

We use an asynchronous Kalman filter for our forecasting module. The state representation which we choose is $[x, y, w, h, \dot{x}, \dot{y}, \dot{w}, \dot{h}]$, where $[x, y, w, h]$ are the top-left coordinates, and width and height of the bounding box, and the remaining four are their derivatives. The state transition is assumed to be linear. We also test with the representation used in SORT [12], which assumes that the area (the product of the width and the height) varies linearly instead of that each of the width and the height varies linearly. We find that such a representation produces lower numbers in AP.

Table 3.12: Empirical performance comparison before and after using Alg. 1. We see that our shrinking-tail policy consistently boosts the streaming performance for different detectors and for different input scales. We also observe some failure cases (last two rows), where runtime is close to one frame duration. This is because our theoretical analysis assumes constant runtime, while it is dynamic in practice. Hence, the variance in runtime when it is a boundary value can make a noticeable difference on the performance

Method	AP (Before)	AP (After)	Runtime (ms)	Runtime (frames)
SSD @ s0.5	9.7	9.7	66.7	2.0
RetinaNet R50 @ s0.5	10.9	11.6	54.5	1.6
RetinaNet R101 @ s0.5	9.9	9.9	66.8	2.0
Mask R-CNN R101 @ s0.5	11.0	11.1	68.8	2.1
Cascade MRCNN R50 @ s0.5	11.3	11.7	80.0	2.4
Cascade MRCNN R101 @ s0.5	10.3	11.1	92.2	2.8
HTC @ s0.5	7.9	8.0	240.8	7.2
Mask R-CNN R50 @ s0.25	7.7	7.8	36.1	1.1
Mask R-CNN R50 @ s0.5	12.0	13.0	56.7	1.7
Mask R-CNN R50 @ s0.75	11.5	12.6	92.7	2.8
Mask R-CNN R50 @ s1.0	10.6	10.7	139.6	4.2
RetinaNet R50 @ s0.25	6.9	6.8	33.4	1.0
Mask R-CNN R50 @ s0.2	6.5	6.3	34.3	1.0

As explained in the main text, Kalman filter needs to be asynchronous and time-varying for streaming perception. Let Δt_k be the time-varying intervals between updates or prediction steps, we pick the transition matrix to be:

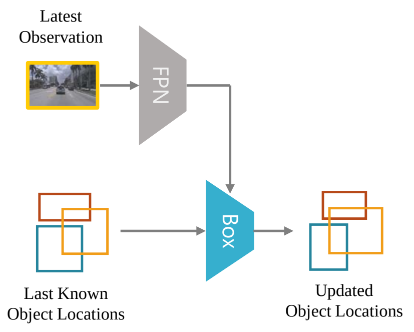
$$\mathbf{F}_k = \begin{bmatrix} \mathbf{I}_{4\times 4} & \Delta t_k \mathbf{I}_{4\times 4} \\ & \mathbf{I}_{4\times 4} \end{bmatrix} \tag{3.13}$$

and the process noise to be

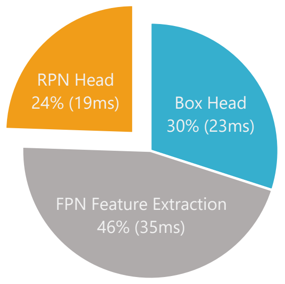
$$\mathbf{Q}_k = \Delta t_k^2 \mathbf{I}_{8\times 8} \tag{3.14}$$

Intuitively, the process noise is larger the longer between the updates.

All forecasting modules are implemented on the CPU and thus can be parallelized while the detector runs on the GPU. Our batched (over multiple objects) implementation of the asynchronous Kalman filter takes 0.98 ± 0.39ms for the update step and 0.22 ± 0.07ms for the prediction step, which are relatively very small overheads compared to detector runtimes. For scalable evaluation, we assume zero runtime for the association and forecasting



(a) Multi-Object Tracker



(b) Computation Saving

Figure 3.17: Multi-object visual tracker. The advantage of a visual tracker is that it runs faster than a detector and thus yields lower latency for streaming perception. The multi-object tracker used here is modified from [11]. It is mostly the same as a two-stage detector, except that its box head uses the last known object location as input in place of region proposals. Therefore, we get a computation saving by not running the RPN head. Runtime is measured for Mask R-CNN (ResNet 50) with input scale 0.5.

module, and implement forecasting as post-processing of the detection outputs. One might wonder that a simulated post-processing run and an actual real-time parallel execution might have different final APs. We have also implemented the latter for verification purposes. For most settings the differences are within 1%. Although for some settings the difference can reach 3%, we find such fluctuation does not affect the relative rankings.

3.B.3 Additional Details for Visual Tracking

For our tracking experiments (Section 3.4.4), we adapt and modify the state-of-the-art multi-object tracker [11]. A component breakdown in Fig. 3.17 explains how this tracker works and why it has the potential to achieve better performance under the streaming setting.

3.B.4 Evaluation of Our Meta-Detector *Streamer*

Streamer is introduced in Section 3.4.3 in the main text. Given a detector and an input scale, Streamer automatically schedules the detector and employs forecasting to compensate for some of its latency. In the single GPU case, our dynamic schedule (Alg. 1) is used and in the infinite GPU case, idle-free scheduling (Fig. 3.4c) is used. Proper scheduling requires the knowledge of runtime of the algorithm, which is known in the case of benchmark

Table 3.13: Performance boost after applying Streamer. "(B)" standards for "Before", and "(A)" standards for "After". The evaluation setting is the same as Table 3.1 in the main text. This table assumes a *single* GPU, and an infinite GPU counterpart can be found in Table 3.14. Under this setting, we observe significant improvement in AP, ranging from 5% to 78%, and averaging at 34%

Method	Scale	AP(B)	AP(A)	Boost	AP_L(B)	AP_L(A)	Boost
SSD	0.2	9.5	10.4	9%	23.5	28.6	21%
	0.25	9.3	10.6	14%	23.9	31.5	32%
	0.5	9.7	13.5	40%	20.0	32.4	62%
	0.75	6.0	10.7	78%	11.5	19.8	72%
	1.0	4.2	7.3	76%	7.3	12.5	72%
RetinaNet R50	0.2	6.0	6.3	5%	18.1	21.3	17%
	0.25	6.9	7.5	9%	19.8	26.2	33%
	0.5	10.9	14.2	30%	24.1	38.3	59%
	0.75	10.8	16.1	50%	20.2	32.9	63%
	1.0	9.9	14.1	42%	16.7	24.7	48%
RetinaNet R101	0.2	5.4	5.9	9%	14.7	19.8	35%
	0.25	6.5	7.4	14%	18.2	25.8	42%
	0.5	9.9	13.0	31%	21.5	33.6	56%
	0.75	9.9	14.5	47%	18.1	27.7	53%
	1.0	8.9	12.7	42%	14.7	22.0	50%
Mask R-CNN R50	0.2	6.5	7.2	11%	18.0	25.1	40%
	0.25	7.7	9.1	19%	20.1	29.9	49%
	0.5	12.0	16.7	39%	24.3	39.9	64%
	0.75	11.5	17.8	54%	19.5	33.3	71%
	1.0	10.6	15.0	42%	16.6	25.0	50%
Mask R-CNN R101	0.2	6.3	7.2	14%	16.7	24.1	45%
	0.25	7.6	9.0	17%	19.3	28.5	48%
	0.5	11.0	15.2	39%	21.6	35.4	64%
	0.75	10.0	15.3	52%	16.8	28.0	67%
	1.0	8.8	12.4	42%	13.7	21.2	55%
Cascade MRCNN R50	0.2	6.2	7.8	25%	15.4	25.5	66%
	0.25	7.5	9.6	28%	18.4	30.1	63%
	0.5	11.3	16.4	45%	22.6	37.5	66%
	0.75	10.9	16.7	54%	18.6	29.8	60%
	1.0	10.1	15.7	55%	15.4	25.3	64%
Cascade MRCNN R101	0.2	6.1	7.3	20%	15.2	23.1	52%
	0.25	7.4	9.5	28%	17.6	29.6	69%
	0.5	10.3	15.4	49%	20.5	34.1	66%
	0.75	9.5	14.7	54%	16.1	26.1	62%
	1.0	8.8	12.9	46%	13.7	21.8	59%
HTC	0.2	5.6	6.8	22%	12.0	17.0	42%
	0.25	6.3	8.3	31%	13.0	19.8	53%
	0.5	7.9	10.8	38%	13.3	19.9	49%
	0.75	7.1	8.6	22%	11.4	14.8	30%
	1.0	6.4	7.2	12%	9.6	11.4	18%

Table 3.14: Performance boost after applying Streamer. "(B)" standards for "Before", and "(A)" standards for "After". The evaluation setting is the same as Table 3.1 in the main text. This table assumes *infinite* GPUs, and a single GPU counterpart can be found in Table 3.13. Under this setting, we observe significant improvement in AP, ranging from 4% to 80%, and averaging at 32%

Method	Scale	AP(B)	AP(A)	Boost	AP_L(B)	AP_L(A)	Boost
SSD	0.2	9.9	10.6	7%	25.5	29.4	15%
	0.25	9.6	10.7	12%	24.9	31.7	27%
	0.5	11.3	14.7	30%	24.1	35.4	47%
	0.75	8.0	13.3	66%	14.6	25.6	76%
	1.0	5.5	9.8	80%	10.0	16.5	65%
RetinaNet R50	0.2	6.1	6.3	4%	18.6	21.3	15%
	0.25	7.1	7.6	8%	21.4	27.1	26%
	0.5	12.3	14.7	20%	28.1	40.1	42%
	0.75	13.1	18.0	37%	24.3	37.8	56%
	1.0	11.7	17.3	48%	19.5	31.3	60%
RetinaNet R101	0.2	5.5	6.0	9%	15.3	20.1	32%
	0.25	6.7	7.5	12%	18.8	26.1	38%
	0.5	11.3	14.0	24%	25.3	38.1	50%
	0.75	11.8	17.0	44%	21.3	34.3	61%
	1.0	10.8	16.3	51%	18.2	28.2	55%
Mask R-CNN R50	0.2	6.7	7.4	10%	20.0	26.2	31%
	0.25	7.8	9.2	17%	20.8	30.1	45%
	0.5	13.9	17.4	26%	29.0	42.6	47%
	0.75	14.4	20.3	40%	24.3	38.5	59%
	1.0	12.4	18.7	51%	19.4	31.4	62%
Mask R-CNN R101	0.2	6.5	7.3	13%	17.4	24.3	40%
	0.25	7.9	9.1	15%	20.5	28.9	41%
	0.5	11.9	16.2	36%	23.7	38.4	62%
	0.75	12.4	18.5	49%	20.3	35.3	74%
	1.0	10.6	16.2	53%	16.9	27.7	64%
Cascade MRCNN R50	0.2	7.0	7.9	13%	18.9	26.5	40%
	0.25	8.5	9.9	16%	22.3	31.7	42%
	0.5	12.9	17.6	37%	26.0	41.2	58%
	0.75	13.2	19.9	51%	22.1	36.5	65%
	1.0	12.6	19.8	57%	19.0	31.8	67%
Cascade MRCNN R101	0.2	6.8	7.9	17%	17.8	26.6	49%
	0.25	8.3	9.8	18%	21.0	31.7	50%
	0.5	12.6	17.0	35%	25.0	38.5	54%
	0.75	11.4	17.7	56%	19.0	32.7	72%
	1.0	10.5	16.6	59%	16.7	27.6	65%
HTC	0.2	6.3	8.0	27%	14.0	21.8	55%
	0.25	7.3	9.8	34%	15.7	25.5	62%
	0.5	9.2	13.7	50%	16.3	26.9	65%
	0.75	8.2	11.4	39%	13.2	20.5	55%
	1.0	7.4	9.3	25%	11.1	15.8	43%

evaluation. When applied in the wild, we can optionally track runtime of the algorithm on unseen data and adjust the scheduling accordingly. The forecasting module is implemented with asynchronous Kalman filter (Section 3.B.2).

Streamer has several key features. First, it enables synchronous processing for an asynchronous problem. Under the commonly studied settings (both offline and online), computation is synchronous in that the outputs and the inputs have a natural one-to-one correspondence. Therefore, many existing temporal reasoning models assume that the inputs are at a uniform rate and each input corresponds to an output [54, 63, 72]. In the real-time setting, however, such a relationship does not exist due to the latency of the algorithm, *i.e.*, the number of outputs can be arbitrary. Streamer converts detectors with arbitrary runtimes into systems that output at a designated fixed rate. In short, it abstracts away the asynchronous nature of the problem and therefore allows downstream synchronous processing. Second, by adopting forecasting, Streamer significantly boosts the performance of streaming perception. In Tables 3.13 and 3.14, we evaluate the detection AP before and after applying our meta-detector. We observe relative improvement from 4% to 80% with an average of 33% in detection AP under 80 different settings (8 detectors $\times$ 5 image scales $\times$ 2 compute models). Note that the difference of this evaluation and benchmark evaluation in the main text is that we fix the detector and input scale here, while benchmark evaluation searches over the best configuration of detectors and input scales. For the infinite GPU settings, we discount the boost from additional compute itself.

3.B.5 Implementation Details

Detectors We experiment with a large suite of object detectors: SSD [142], RetinaNet [138], Mask R-CNN [82], Cascade Mask R-CNN [24], and HTC [32]. The "optimized" and "re-optimized" rows in all tables represent the optimal configuration over all detectors and all input scales of 0.2, 0.25, 0.5, 0.75, and 1.0. We adopt mmdetection codebase [33] (one of the fastest open-source implementation for Mask R-CNN) for object detectors. Note that for all detectors, the implementation has reproduced both the accuracy and runtime reported in the original papers.

Potentially better implementation We acknowledge that there are additional bells and whistles to reduce runtime of object detectors, which might further improve the results on our benchmark. We focus on general techniques instead of device- or application-specific ones. For example, we have explored GPU image pre-processing, which is applicable to all GPUs. Another implementation technique is to use half-precision floating-point numbers (FP16), which we have not explored, since it will only pay off for certain GPUs that have been optimized for FP16 computation (it is reported that FP16 yields only marginal testing time speed boost on 1080 Ti [37]).

3.C Appendix C: Additional Baselines

3.C.1 Forecasting Baselines

We have also tested linear extrapolation (*i.e.*, constant velocity) and quadratic extrapolation for forecasting detection results. We include an illustration of linear forecasting in Fig. 3.18, and the quadratic counterpart is a straightforward extension that involves three latest detection results. Though they produce inferior results than Kalman filter, we include the results in Table 3.15 for completeness.

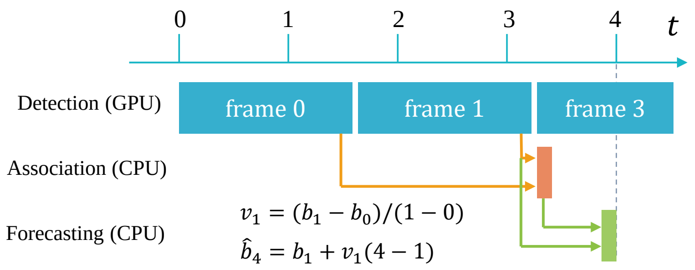


Figure 3.18: Scheduling for linear forecasting. The scheduling is similar as with the Kalman filter case in that both are asynchronous. The difference is that linear forecasting does not explicitly maintain a state representation but only stores two latest detection results. Association takes place immediately after a new detection result becomes available, and it links the bounding boxes in two consecutive detection results and computes a velocity estimate. Forecasting takes place right before the next time step, and it uses linear extrapolation to produce an output as the estimation of the current world state. The equations represent the computation for reporting to benchmark query at $t = 4$. b is a simplified representation for object location. At this time, only detection results for frame 0 and 1 are available, but through association and forecasting, the algorithm can make a better prediction for the current world state.

Table 3.15: Comparison of different forecasting methods for streaming perception. We see that both linear and Kalman filter forecasting methods significantly improve the streaming performance. Kalman filter further outperforms the linear forecasting. The quadratic forecasting decreases the AP, suggesting that high-order extrapolation is not suitable for this task. The detection used here is Mask R-CNN ResNet 50 @ s0.5 with dynamic scheduling (Alg. 1)

ID	Method	AP	AP_L	AP_M	AP_S	AP_{50}	AP_{75}
1	No Forecasting	13.0	26.6	9.2	1.1	26.8	11.1
2	Linear (constant velocity)	15.7	38.1	13.8	1.1	30.2	14.8
3	Quadratic	9.7	23.8	6.6	0.4	21.4	7.9
4	Kalman filter	**16.7**	**39.8**	**14.9**	**1.2**	**31.2**	**16.0**

Table 3.16: Standard offline forecasting evaluation for the end-to-end method F2F [145]. The goal is to forecast 3 frames into the future. Surprisingly, the more expensive F2F method performs worse than the simpler Kalman filter in terms of the overall AP

ID	Method	AP	AP_L	AP_M	AP_S	AP_{50}	AP_{75}
1	None (copy last)	13.4	24.3	10.9	1.9	27.9	11.3
2	Linear	16.3	34.8	16.8	1.8	32.9	14.3
3	Kalman filter	**19.1**	40.3	19.8	**2.6**	**35.8**	**17.7**
4	F2F	18.3	**41.0**	**20.0**	2.5	33.9	17.1

3.C.2 An End-to-End Baseline

In the main text, we break down the streaming detection task into detection, tracking, and forecasting for modular analysis. Alternatively, it is also possible to train a model that directly outputs detection results in the future. F2F [145] is one such model. Building upon Mask R-CNN, it does temporal reasoning and forecasting at the level of FPN feature maps. Note that no explicit tracking is performed. In this section, we compare against this end-to-end baseline in both offline and streaming settings.

In the offline setting, the algorithm is given s frames as input history, and outputs detection results for t frames ahead. This is the same as the evaluation in [145]. We set both s and t to be 3, as the optimal detector in our forecasting experiments (Table 3.2) has runtime of 2.78 frames. Since F2F forecasts at the FPN feature level, it is agnostic to second stage tasks. In our

Table 3.17: Streaming evaluation for the end-to-end method F2F [145]. The setting is the same as the experiments in the main text. Rows 1 and 2 are the optimized detector and the Kalman filter forecasting solution from the main text. The underlying detectors used are Mask R-CNN ResNet 50 at scale 0.5 and scale 0.75 respectively. Row 3 suggests that F2F has a low streaming AP, due to its forecasting module being very expensive (last column, runtime in milliseconds). For diagnostics purpose, we assume F2F to run as fast as our optimized detector (row 4), and arm it with our scheduling algorithm (row 5). But even so, F2F still under-performs the simple Kalman filter solution

ID	Method	AP	AP_L	AP_M	AP_S	AP_{50}	AP_{75}	Runtime
1	Detection	12.0	24.3	7.9	1.0	25.1	10.1	**56.7**
2	+ Scheduling (Alg. 1) + KF	**17.8**	**33.3**	**16.3**	**3.2**	**35.2**	**16.5**	92.7
3	F2F	6.2	11.1	3.4	0.8	13.1	5.2	321.6
4	F2F (Simulated Fast)	14.1	29.1	12.7	1.9	28.9	12.0	92.7
5	+ Scheduling (Alg. 1)	15.6	33.0	15.2	2.1	30.7	13.9	92.7

evaluation, we focus on the bounding box detection task instead of instance segmentation. Also, we conduct experiments on Argoverse-HD, consistent with the setting in our other experiments. Due to a lack of annotation, we adopt pseudo ground truth (Section 3.A.2) for training (data from the original training set of Argoverse 1.1 [29]). We implement our own version of F2F based on mmdetection (instead of Detectron as done in [145]). We train the model for 12 epochs end-to-end (a 50% longer schedule than combined stages in [145]). For a fair comparison, we also finetuned the detectors on Argoverse with the same pseudo ground truth. For Mask R-CNN ResNet 50 at scale 0.5, it boosts the offline box AP from 19.4 to 22.9. We use this finetuned detector in our method to compare against F2F. The results are summarized in Table 3.16. We see that an end-to-end solution does not immediately boost the performance. We believe that it is still an open problem on how to effectively replace tracking and forecasting with an end-to-end solution.

In the streaming setting, F2F can be viewed as a detector that compensates its own latency. The results are summarized in Table 3.17. We see that F2F is too expensive compared with other streaming solutions, showing that forecasting can help only if it is fast under our evaluation. Note that the detectors (row 1–2) are not finetuned as in the offline case, which means that they can be further improved.

Chapter 4

Foveated Image Magnification

4.1 Introduction

Safety-critical robotic agents such as self-driving cars make use of an enormous suite of high-resolution perceptual sensors, with the goal of minimizing blind spots, maximizing perception range, and ensuring redundancy [22, 29, 199]. We argue that "over-sensed" perception platforms provide unique challenges for vision algorithms since those visual sensors must rapidly consume sensor streams while continuously reporting back the state of the world. While numerous techniques exist to make a particular model run fast, such as quantization [213], model compression [38], and inference optimization [171], at the end of the day, simple approaches that subsample sensor data (both spatially by frame downsampling and temporally by frame dropping) are still most effective for meeting latency constraints [129]. However, subsampling clearly throws away information, negating the goals of high-resolution sensing in the first place! This status quo calls for novel vision algorithms.

To address this challenge, we take inspiration from the human visual system; biological vision makes fundamental use of *attentional* processing. While current sensing stacks make use of regular grid sampling, the human vision system in the periphery has a much lower resolution than in the center (fovea), due to the pooling of information from retinal receptors by retinal ganglion cells. Such variable resolution is commonly known as foveal vision [122].

In this paper, we propose FOVEAted image magnification (FOVEA) for object detection, which retains high resolution for objects of interest while maintaining a small canvas size. We exploit the sparsity of detection datasets

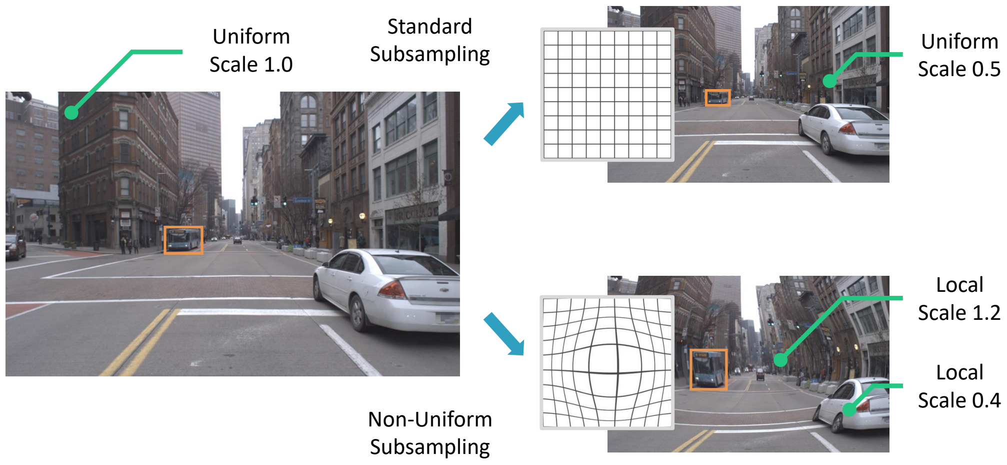


Figure 4.1: Standard image downsampling (top right) limits the capability of the object detector to find small objects. In this paper, we propose an attentional warping method (bottom right) that enlarges salient objects in the image while maintaining a small input resolution. Challenges arise when warping also alters the output labels (*e.g.*, bounding boxes).

– objects of interest usually only cover a portion of the image. *The key idea is to resample such that background pixels can make room for salient pixels of interest.* The input images are downsampled and warped such that salient areas in the warped image have higher resolutions. While image warping has been explored for image classification [102, 176] and regression [176], major challenges remain when applying such methods to detailed spatial prediction tasks such as object detection. First, processing warped images will produce warped spatial predictions (bounding box coordinates). We make use of differentiable backward maps to unwarp spatial predictions back to the original space. Second, it is hard to efficiently identify salient regions; in the worst case, a saliency network tuned for object detection may be as expensive as the downstream detection network itself, thereby eliminating any win from downsampling. In our case, we make use of cheap and readily available saliency cues, either in the form of dataset-specific spatial priors (i.e., small objects tend to exist near a fixed horizon) or temporal priors (small objects tend to lie nearby small object predictions from previous frames). Third, previous image warps (tuned for image classification tasks) can produce cropped image outputs. Since objects can appear near the image boundary, we introduce anti-cropping constraints on the warping.

We validate our approach on two self-driving datasets for 2D object detection: Argoverse-HD [129] and BDD100K [244]. First, we show that FOVEA can improve the performance of off-the-shelf detectors (Faster R-CNN [180]). Next, we finetune detectors with differentiable image warping and backward label mapping, further boosting performance. In both cases, small objects improve by more than 2x in average precision (AP). Finally, we evaluate FOVEA under streaming perception metrics designed to capture both accuracy and latency [129], producing state-of-the-art results.

4.2 Related Work

Object detection Object detection is one of the most fundamental problems in computer vision. Many methods have pushed the state-of-the-art in detection accuracy [32, 73, 137, 169, 180], and many others aim for improving the efficiency of the detectors [14, 142, 179, 205]. The introduction of fully convolution processing [192] and spatial pyramid pooling [83] have allowed us to process the input image in its original size and shape. However, it is still a common practice to downsample the input image for efficiency purposes. Efficiency becomes a more prominent issue when people move to the video domain. In video object detection, the focus has been on how to make use of temporal information to reduce the number of detectors invoked [147, 255, 257]. These methods work well on simple datasets like ImageNet VID [185], but might be unsuitable for the self-driving car senarios, where multiple new objects appear at almost every frame. Furthermore, those methods are usually designed to work in the offline fashion, *i.e.*, allowing access to future frames. Detection methods are the building blocks of our framework, and our proposed approach is largely agnostic to any particular detector.

Online/streaming perception In the online setting, the algorithm must work without future knowledge. [136] proposes the Temporal Shift Module that enables video understanding through channel shifting and in the online setting, the shifting is restricted to be uni-directional. [11] proposes a multi-object tracking method that takes input previous frame detection as addition proposals for the current frame. Our method also takes previous frame detection as input, but we use that to guide image warping. Streaming accuracy [129] is a recently proposed metric that evaluates the output of a perception algorithm at all time instants, forcing the algorithm to consider the amount of streaming data that must be ignored while computation is occuring. [129] demonstrates that streaming object detection accuracy can be significantly improved by tuning the input frame resolution and framer-

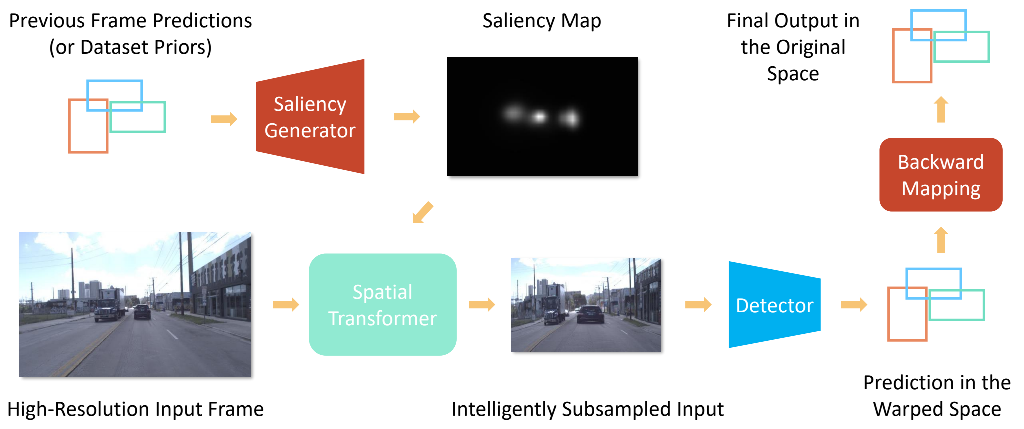

Previous Frame Predictions (or Dataset Priors)
Saliency Generator
Saliency Map
Final Output in the Original Space
Backward Mapping
High-Resolution Input Frame
Spatial Transformer
Intelligently Subsampled Input
Detector
Prediction in the Warped Space

Figure 4.2: Our proposed method for object detection. Given bounding box predictions from the previous frame (if the input are videos) or a collection of all the ground truth bounding boxes in the training set, the saliency generator creates a saliency map and that is fed into the spatial transformer (adapted from [102, 176]) to downsample the high-resolution input frame while magnifying salient regions. Then we feed the downsampled input into a regular object detector, and it produces bounding box output in the warped space, which is then converted back to the original image space as the final output.

ate. In this work, we demonstrate that adaptive attentional processing is an orthogonal dimension for improving streaming performance.

Adaptive visual attention Attentional processing has been well studied in the vision community, and it appears in different forms [45, 95, 110, 132, 141, 214]. Specially in this paper, we focus on dynamic resolutions. For image classification, [212] designs an algorithm to select high-resolution patches, assuming each patch is associated with a data acquisition cost. [152] applies non-uniform downsampling to semantic segmentation and relies on the network to learn both the forward and backward mapping, whose consistency is not guaranteed. For object detection, a dynamic zoom-in algorithm is proposed that processes high-resolution patches sequentially [68]. However, sequential execution might not meet latency requirements for real-time applications. Most similar to our work, [176] proposes an adaptive image sampling strategy that allocates more pixels for salient areas, allowing a better downstream task performance. But the method only works for image classification and regression, where the output is agnostic to the input transformation.

4.3 Approach

Assume we are given a training set of image-label pairs (I, L). We wish to learn a nonlinear deep predictor f that produces a low loss $\mathcal{L}(f(I), L)$. Inspired by past work [102,176], we observe that certain labeling tasks can be performed more effectively by warping/resampling the input image. However, when the label L itself is spatially defined (e.g., bounding box coordinates or semantic pixel labels), the label itself may need to be warped, or alternatively, the output of the deep predictor may need to be inverse-warped.

In this section, we first introduce the saliency-guided spatial transform from related work as the foundation of our method. Next, we introduce our solutions to address the challenges in image warping for object detection. An overview of FOVEA, our method, is shown in Fig 4.2.

4.3.1 Background: Saliency-Guided Spatial Transform

The seminal work of spatial transformer networks (STN) introduces a differentiable warping layer for input images and feature maps [102]. It was later extended to incorporate a saliency map to guide the warping [176]. Here we provide implementation details that are crucial to our method. Please refer to the original papers [102,176] for more details.

A 2D transformation can be written as:

$$\mathcal{T} : (x, y) \to (x', y'), \tag{4.1}$$

where (x, y) and (x', y') are the input and output coordinates. Since image pixels are usually discrete, interpolation is required to sample values at non-integral coordinates. An image warp $\mathcal{W}_\mathcal{T}$ takes input an image I, samples the pixel values according to the given transformation $\mathcal{T}$, and outputs the warped image I':

$$I'(\mathcal{T}(x, y)) = I(x, y) \tag{4.2}$$

Naive forward warping of discrete pixel locations from input I can result in non-integral target pixel positions that need to be "splatted" onto the pixel grid of I, which can produce artifacts such as holes. Instead, image warps are routinely implemented via a *backward map* [9]: iterate over each output pixel grid location, compute its *inverse mapping* $\mathcal{T}^{-1}$ to find its corresponding input coordinates (which may be non-integral), and bilinearly interpolate its color from neighboring input pixel grid points:

$$I'(x, y) = I(\mathcal{T}^{-1}(x, y)) \tag{4.3}$$

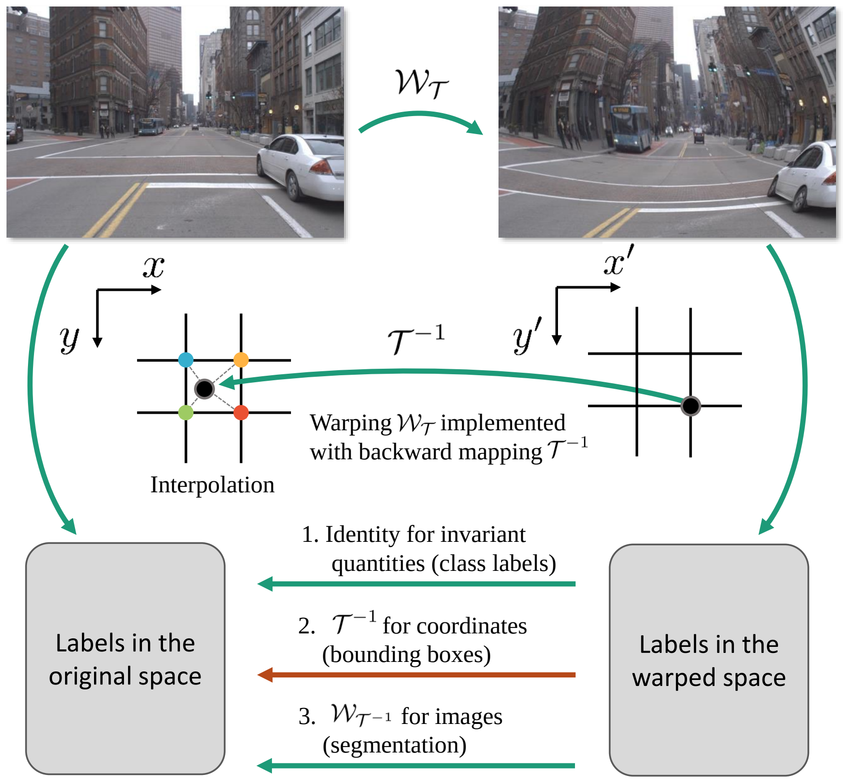

$\mathcal{W}_\mathcal{T}$
x
y
x'
y'
$\mathcal{T}^{-1}$
Warping $\mathcal{W}_\mathcal{T}$ implemented with backward mapping $\mathcal{T}^{-1}$
Interpolation
1. Identity for invariant quantities (class labels)
2. $\mathcal{T}^{-1}$ for coordinates (bounding boxes)
3. $\mathcal{W}_{\mathcal{T}^{-1}}$ for images (segmentation)
Labels in the original space
Labels in the warped space

Figure 4.3: Image warps $\mathcal{W}_\mathcal{T}$ are commonly implemented via a backward map $\mathcal{T}^{-1}$ followed by (bilinear) interpolation of nearby source pixel grid values, since forward mapping $\mathcal{T}$ can result in target pixel positions that do not lie on the pixel grid (not shown). Though image warping is an extensively studied topic (notably by [102, 176] in the context of differentiable neural warps), its effect on labels is less explored because much prior art focuses on global labels invariant to warps (e.g. an image class label). We explore warping for spatial prediction tasks whose output must be transformed back into the original image space to generate consistent output. Interestingly, transforming pixel-level labels with warp $\mathcal{W}_{\mathcal{T}^{-1}}$ requires inverting $\mathcal{T}^{-1}$, which can be difficult depending on its parameterization [9]. In this paper, we focus on transforming pixel *coordinates* of bounding boxes, which requires only the already-computed backward map $\mathcal{T}^{-1}$ (the red arrow).

In other words, the implementation of $\mathcal{W}_{\mathcal{T}}$ only requires the knowledge of the inverse transformation $\mathcal{T}^{-1}$. The pixel iteration can be replaced with a batch operation by using a grid generator and apply the transformation $\mathcal{T}^{-1}$ over the entire grid.

STN uses a differentiable formulation of $\mathcal{T}_{\theta}^{-1}$ (parameterized by θ) and an ensuing bilinear grid sampler, which is differentiable and parameter-free. [176] proposes a special form of $\mathcal{T}^{-1}$ parameterized by a saliency map S: $\mathcal{T}_{\theta}^{-1} = \mathcal{T}_{S}^{-1}$. This transform has a convolution form and is therefore fast, using the intuition that each input pixel (x, y) attracts samples from the original image with a force $S(x, y)$, leading to more sampling at salient regions. *We point out that both [102] and [176] ignore the effect of warping on the output label space and skip the modeling of the forward transform $\mathcal{T}$, which (we will show) is required for unwarping certain label types.*

4.3.2 Image Warping for Object Detection

In this section, we first explain our high-level inference formulation, then our specific form of the warping, and in the end some adjustments for training the task network.

Inference formulation We visually lay out the space of image and label warps in Fig 4.3. Recent methods for differentiable image warping assume labels are invariant under the warping (the first pathway in Fig 4.3). For object detection, however, image warping clearly warps bounding box outputs. To produce consistent outputs (e.g., for computing bounding box losses during learning), these warped outputs need to transformed back into the original space (the second pathway in Fig 4.3). Quite conveniently, because standard image warping is implemented via the backward map $\mathcal{T}^{-1}$, the backward map is already computed in-network and so can be directly applied to the pixel coordinates of the predicted bounding box. The complete procedure for our approach $\hat{f}$ can be written as $\hat{f}(I, \mathcal{T}) = \mathcal{T}^{-1}(f(\mathcal{W}_{\mathcal{T}}(I)))$. where $f(\cdot)$ is the nonlinear function that returns bounding box coordinates of predicted detections. Importantly, this convenience doesn't exist when warping pixel-level *values*; *e.g.*, when warping a segmentation mask back to the original image input space (the third pathway in Fig 4.3). Here, one needs to invert $\mathcal{T}^{-1}$ to explicitly compute the forward warp $\mathcal{T}$.

Warping formulation We adopt the saliency-guided warping formulation from [176]:

$$\mathcal{T}_x^{-1}(x,y) = \frac{\int_{x',y'} S(x',y')k((x,y),(x',y'))x'}{\int_{x',y'} S(x',y')k((x,y),(x',y'))}, \tag{4.4}$$

$$\mathcal{T}_y^{-1}(x,y) = \frac{\int_{x',y'} S(x',y')k((x,y),(x',y'))y'}{\int_{x',y'} S(x',y')k((x,y),(x',y'))}, \tag{4.5}$$

where k is a distance kernel (we use a Gaussian kernel in our experiments). However, in this general form, axis-aligned bounding boxes might have different connotations in the original and warped space. To ensure axis alignment is preserved during the mapping, we restrict the warping to be separable along the two dimensions, *i.e.*, $\mathcal{T}^{-1}(x,y) = (\mathcal{T}_x^{-1}(x), \mathcal{T}_y^{-1}(y))$. For each dimension, we adapt the previous formulation to 1D:

$$\mathcal{T}_x^{-1}(x) = \frac{\int_{x'} S_x(x')k(x',x)x'}{\int_{x'} S_x(x')k(x,x')}, \tag{4.6}$$

$$\mathcal{T}_y^{-1}(y) = \frac{\int_{y'} S_y(y')k(y',y)y'}{\int_{y'} S_y(y')k(y,y')}. \tag{4.7}$$

We call this formulation *separable* and the general form *nonseparable*. Note that the nonseparable formulation has a 2D saliency map parameter, whereas the separable formulation has two 1D saliency maps, one for each axis. Fig 4.4 shows an example of each type of warp.

One nice property of $\mathcal{T}^{-1}$ is that it is differentiable and thus can be trained with backpropagation. One limitation though is that its inverse $\mathcal{T}$ doesn't have a closed-form solution, nor does its derivative. The absence of $\mathcal{T}$ is not ideal, and we propose some workaround as shown in the following subsection.

Anti-Cropping Constraint We find the convolution form of saliency-guided spatial transform tends to crop the images, which might be acceptable for image classification where a large margin exists around the border. However, any cropping in object detection creates a chance to miss objects. We solve this by using reflect padding on the saliency map while applying the attraction kernel in Eq 4.6. This introduces symmetries about each of the edges of the saliency map, eliminating all horizontal offsets along vertical image edges and vice versa. Thus cropping is impossible under this formulation. A 1D illustration is shown in Fig 4.5 to explain the problem and the solution.

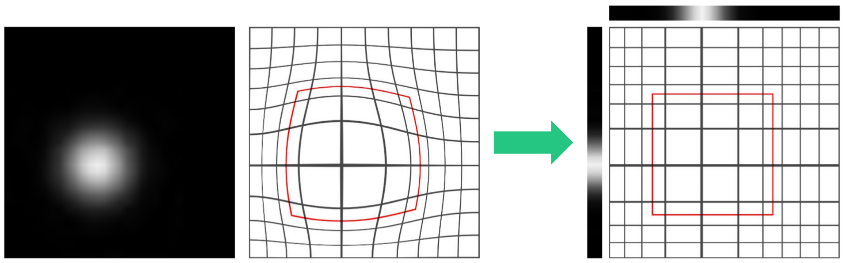


Figure 4.4: By restricting the general class of warps (**left**) to be separable (**right**), we ensure that bounding boxes in the warped image (examples outlined in red) remain axis-aligned. We demonstrate that such regularization (surprisingly) improves performance, even though doing so theoretically restricts the range of expressible warps (details in Sec 4.4.1).

Training formulation Once we have the inference formulation, training is also straightforward as we require the loss $\mathcal{L}$ to be computed in the original space: $\mathcal{L}(\mathcal{Q}(f(\mathcal{W}_{\mathcal{T}}(I)), L)$, where $\mathcal{Q}$ is the label-type-specific backward mapping as shown in Fig 4.3, and in our case, $\mathcal{Q} = \mathcal{T}^{-1}$. Note that $\mathcal{W}_{\mathcal{T}}$, f and $\mathcal{T}^{-1}$ are all differentiable. While inference itself does not require the knowledge of $\mathcal{T}$, it is not the case for training detectors with region proposal networks (RPN) [180]. When training RPNs [180], the regression targets are the deltas between the anchors and the ground truth, and the deltas are later used in RoI Pooling/Align [82,83]. The former should be computed in the original space (the ground truth is in the original space), while the latter is in the warped space (RoI Pooling/Align is over the warped image). This implies that the deltas need first to be learned in the original space, applied to the bounding box, and then mapped to the warped space using $\mathcal{T}$ for RoI Pooling/Align. But as discussed before, $\mathcal{T}$ cannot be easily computed. As a workaround, we omit the delta encoding and adopt Generalized IoU (GIoU) loss [181] to account for the lost stability. The main idea of GIoU is to better reflect the similarity of predicted and ground truth bounding boxes in cases of zero intersection; this has been shown to improve results.

4.3.3 KDE Saliency Generator

Prior work [102, 176] trains a saliency network to generate saliency maps, which we explore as a baseline in Sec 4.4.1. Because saliency maps for object detection appear hard to learn, we explore cheap alternatives for saliency map construction: dataset-level priors over object locations or temporal priors extracted from previous frame's predictions. Both priors can be opera-

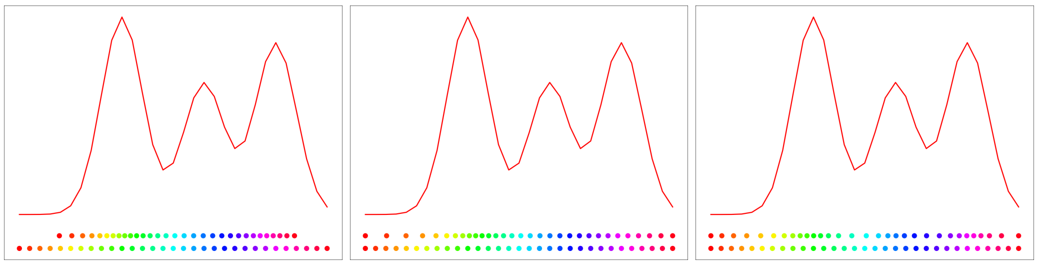

(a) Default, $\sigma \approx 5.5$ **(b)** Anti-crop, $\sigma \approx 5.5$ **(c)** Anti-crop, $\sigma \approx 1.7$

Figure 4.5: Saliency-guided transform illustrated in 1D. The red curve is a saliency map S. The bottom row of dots are the output points (at uniform intervals), and the top row of dots are the locations where we've sampled each output point from the original "image", as computed by applying $\mathcal{T}_S^{-1}$ to the output points. (a) The default transform can be understood as a weighted average over the output points and thus ignores points with near zero weights such as those at the boundaries. (b) Note the effects of introducing anti-crop reflect padding, and (c) how decreasing the std dev σ of the attraction kernel k results in more local warping around each peak (better for multimodal saliency distributions).

tionalized with an approach that converts bounding boxes to a saliency map.

Intuitively, we build a saliency map by "overlaying" boxes on top of one another via non-parametric kernel density estimation (KDE). More precisely, given a set of bounding boxes B with centers c_i, heights h_i and widths w_i, we model the saliency map S_B as a sum of normal distributions:

$$S_B^{a,b} = \frac{1}{K^2} + a \sum_{(c_i,w_i,h_i)\in B} \mathcal{N}\left(c_i, b \begin{bmatrix} w_i & 0 \\ 0 & h_i \end{bmatrix}\right) \tag{4.8}$$

where a and b are hyperparameters for amplitude and bandwidth, respectively, and K is the size of the attraction kernel k in Eq 4.6. Adding the small constant is done to prevent extreme warps. We then normalize the 2D saliency map such that it sums to 1 and marginalize along the two axes if using the separable formulation[1]. As laid out in the previous section, this is then used to generate the image transformation $\mathcal{T}_S^{-1}$ according to Eq 4.6. Ensuring that each kernel is locally normalized produces our desired behavior; we'll have high saliency for pixels covered by objects, and even higher

[1]When using the separable formulation, we could instead skip the intermediate 2D saliency map representation. However, we opt not to, because the intermediate 2D saliency map produces more interpretable visualizations, and the difference in runtime is negligible.

saliency for pixels covered by small objects (that have their Gaussian mass focused on a smaller object size).

We can apply S_B to the set of all bounding boxes in the training set to obtain a dataset-wide prior (denoted as S_D), or apply it to the previous frame's predictions to obtain a image-specific temporal prior (denoted as S_I). The former encodes dataset-level spatial priors such as small objects appearing near the horizon (Fig 4.7). The latter encodes a form of temporal contextual priming, allocating pixel samples to previously seen objects (with a default of uniform saliency for the first frame). We also experiment with a weighted combination of both: $S_C = \alpha \cdot S_I + (1 - \alpha) \cdot S_D$. All of the above saliency generators are differentiable, so the final task loss can be used to learn hyperparameters a, b, α.

4.4 Experiments

We first compare FOVEA to naive downsampling on autonomous driving datasets such as Argoverse-HD. Next, we use streaming perception metrics to show that the accuracy gain is worth the additional cost in latency. Finally, we present results on BDD100K, showing the generalization of our method. We include additional results, diagnostic experiments, and implementation details in the appendix.

4.4.1 Object Detection for Autonomous Navigation

Argoverse-HD [129] is an object detection dataset for autonomous vehicles. Noteably, it contains high framerate (30 FPS) data and annotations. As is standard practice, we adopt AP for evaluation. We also report *end-to-end* latency (including image preprocessing, network inference, and bounding box postprocessing) measured on a single GTX 1080 Ti GPU. The image resolution for this dataset is 1920×1200, much larger than COCO's, which is capped at 640. Since all models used in this paper are fully convolutional, we run them with different input scales, denoted by ratios to the native resolution, *e.g.*, 0.5x means an input resolution of 960×600.

Baseline and Setup

The baseline we compare to throughout our experiments is Faster RCNN [180] with a ResNet-50 backbone [84] plus FPN [137]. The default input scale for both the baseline and our method is 0.5x. For the baseline, however, we additionally train and test at 0.75x and 1x scales, to derive a sense of the latency-accuracy tradeoff using this model. Our contribution is orthogonal to the choice of the baseline detector and we obtain similar results with other detectors including RetinaNet [138] and YOLOF [35] (shown in Appendix 4.A.2). Additionally, we compare against other zoom-based approaches [68, 176] in Appendix 4.A.3.

Notably, Argoverse-HD's training set only contains *pseudo ground truth* (at the time of paper submission) generated by running high-performing detector HTC [32] in the offline setting. For all experiments, unless otherwise stated, we train on the train split with pseudo ground truth annotations, and evaluate on the val split with real annotations. Additional measures are taken to prevent overfitting to biased annotations. We finetune COCO pretrained models on Argoverse-HD for only 3 epochs (*i.e.*, early stopping). We use momentum SGD with a batch size of 8, a learning rate of 0.02, 0.9 momentum, 10^{-4} weight decay, and a step-wise linear learning rate decay for this short schedule [130]. Also, when training detectors with warped input, we apply our modifications to RPN and the loss function as discussed in Sec 4.3.2.

Learned Saliency

Our first control experiment does not make use of bounding box KDE priors, but rather directly learns a *global, dataset-wide* saliency map $S(x, y)$ via backprop. We directly learn both separable and nonseparable saliency maps in Tab 4.1. Training configuration and implementation details are given in Appendix 4.A.6.

We find that both separable and nonseparable warps significantly improve overall AP over the baseline, owing to the boosted performance on small objects. However, there is also a small decrease in AP on large objects. Interestingly, even though nonseparable warps are more flexible, the learned solutions look nearly separable (Fig 4.6) but perform worse, indicating overfitting. Therefore, going forward, we focus on separable warps in our experiments.

Following [176], we also learn a "saliency network" that maps each input image to its saliency map via a ResNet-18 backbone [84]. In this sense, the learned saliency map would adapt to each image. However, we find that this approach very unstable for object detection. From our experiments, even with a small learning rate of 10^{-5} on the saliency network, the model learns a degeneracy in which an extreme warp leads to no proposals being matched

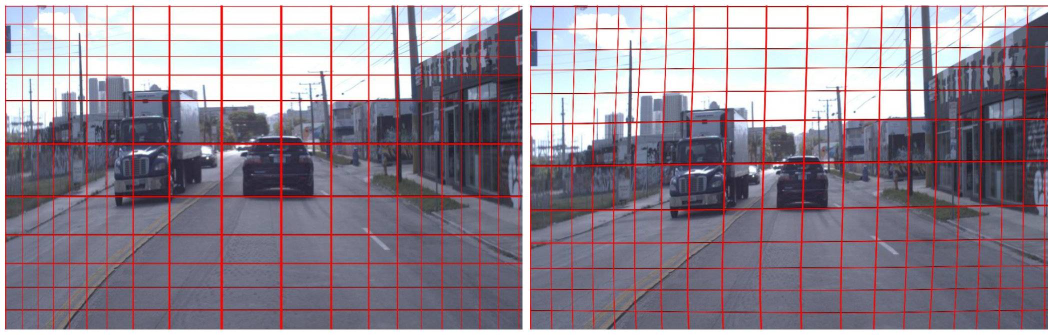


Figure 4.6: The learned direct separable (**left**) and nonseparable (**right**) dataset-wide warps. Despite the vastly greater flexibility of nonseparable warps, the learned warp is almost separable anyway.

with ground truth bounding boxes in the RoI bounding box head, leading to a regression loss of 0.

KDE Saliency Generator

This section makes use of the KDE construction in Sec 4.3.3 to generate saliency maps. We first manually tune the amplitude a and bandwidth b to obtain desired magnifications. We find that an amplitude $a = 1$ and a bandwidth $b = 64$ works the best, paired with an attraction kernel of std. dev. of about 17.8% the image height, which allows for more local warps as illustrated in Fig 4.5. We finetune our models using the same configuration as the baseline, the only difference being the added bounding box and saliency-guided spatial transformation layer. We learn S_D using all bounding boxes from the training set and for simplicity, learn S_I with jittered ground-truth boxes from the current frame (though at test-time it always uses predictions from the previous frame). We set $\alpha = 0.5$ for S_C.

We then learn hyperparameters a and b through backpropagation, since our KDE formulation is differentiable. We initialize parameters a' and b' to 0, under the construction that $a = |1 + a'| + 0.1, b = 64 \cdot |1 + b'| + 0.1$. The learning rate of a' and b' is set to 10^{-4} with zero weight decay. Other than this, we train the learned KDE (LKDE) model with the same configuration as the baseline. We implement the S_I formulation.

All results are shown in Table 4.1. Even without finetuning our detector, using a simple fixed dataset-wide warp S_D, we find significant improvements in AP. As we migrate to temporal priors with finetuning, we see even more improvement. As in the learned saliency case, these improvements in overall AP are due to large boosts in AP$_S$, outweighing the small decreases in AP$_L$. Combining our saliency signals (S_C) doesn't help, because in our case, it seems that the temporal signal is strictly stronger than the dataset-wide signal. Perhaps if we had an alternate source of saliency like a map overlay, combining saliencies could help. Our best method overall is LKDE, which learned optimal values $a = 1.07, b = 71.6$. Learning a nonseparable saliency performs better than our hand-constructed dataset-wide warp S_D; however, they're both outperformed by S_I. Importantly, our LKDE not only significantly improves AP$_S$, but also improves *all* other accuracy measures, suggesting that our method does not need to tradeoff accuracy of large objects for that of small objects. Finally, we note that our increased performance comes at the cost of only about 2 ms in latency.

Argoverse-HD before finetuning

Method	AP	AP$_{50}$	AP$_{75}$	AP$_S$	AP$_M$	AP$_L$	person	mbike	tffclight	bike	bus	stop	car	truck	Latency (ms)
Baseline	21.5	35.8	22.3	2.8	22.4	**50.6**	20.8	9.1	13.9	7.1	48.0	16.1	37.2	20.2	49.4 ± 1.0
KDE (S_D)	23.3	40.0	22.9	5.4	25.5	48.9	20.9	13.7	12.2	9.3	**50.6**	20.1	40.0	19.5	52.0 ± 1.0
KDE (S_I)	**24.1**	**40.7**	**24.3**	**8.5**	24.5	48.3	**23.0**	**17.7**	**15.1**	**10.0**	49.5	17.5	**41.0**	19.4	51.2 ± 0.7
KDE (S_C)	24.0	40.5	**24.3**	7.4	**26.0**	48.2	22.5	14.9	14.0	9.5	49.7	**20.6**	**41.0**	**19.9**	52.0 ± 1.2
Upp. Bound (0.75x)	27.6	45.1	28.2	7.9	30.8	51.9	29.7	14.3	21.5	6.6	54.4	25.6	44.7	23.7	86.9 ± 1.6
Upp. Bound (1.0x)	32.7	51.9	34.3	14.4	35.6	51.8	33.7	21.1	33.1	5.7	57.2	36.7	49.5	24.6	133.9 ± 2.2

Argoverse-HD after finetuning

Method	AP	AP$_{50}$	AP$_{75}$	AP$_S$	AP$_M$	AP$_L$	person	mbike	tffclight	bike	bus	stop	car	truck	Latency (ms)
Baseline	24.2	38.9	26.1	4.9	29.0	50.9	22.8	7.5	23.3	5.9	44.6	19.3	43.7	26.6	50.9 ± 0.9
Learned Sep.	27.2	44.8	28.3	**12.2**	29.1	46.6	24.2	14.0	22.6	7.7	39.5	**31.8**	50.0	27.8	51.5 ± 1.0
Learned Nonsep.	25.9	42.9	26.5	10.0	28.4	48.5	25.2	11.9	20.9	7.1	39.5	25.1	49.4	28.1	50.0 ± 0.8
KDE (S_D)	26.7	43.3	27.8	8.2	29.7	54.1	25.4	13.5	22.0	8.0	**45.9**	21.3	48.1	29.3	50.8 ± 1.2
KDE (S_I)	28.0	45.5	**29.2**	10.4	**31.0**	**54.5**	27.3	16.9	**24.3**	**9.0**	44.5	23.2	**50.5**	28.4	52.2 ± 0.9
KDE (S_C)	27.2	44.7	28.4	9.1	30.9	53.6	27.4	14.5	23.0	7.0	44.8	21.9	49.9	**29.5**	52.1 ± 0.9
LKDE (S_I)	**28.1**	**45.9**	28.9	10.3	30.9	54.1	**27.5**	**17.9**	23.6	8.1	45.4	23.1	50.2	28.7	50.5 ± 0.8
Upp. Bound (0.75x)	29.2	47.6	31.1	11.6	32.1	53.3	29.6	12.7	30.8	7.9	44.1	29.8	48.8	30.1	87.0 ± 1.4
Upper Bound (1.0x)	33.3	53.9	35.0	16.8	34.8	53.6	33.1	20.9	38.7	6.7	44.7	36.7	52.7	32.7	135.0 ± 1.6

Table 4.1: Results before and after finetuning on Argoverse-HD. Without retraining, processing warped images (KDE S_I, top table) improves overall AP by 2.6 points and triples AP$_S$. Even larger gains can be observed after finetuning, making our final solution (LKDE S_I) performing close to the 0.75x upper bound. Please refer to the text for a more detailed discussion.

4.4.2 Streaming Accuracy for Cost-Performance Evaluation

Streaming accuracy is a metric that coherently integrates latency into standard accuracy evaluation and therefore is able to *quantitatively measure the accuracy-latency tradeoff* for embodied perception [129]. Such a setup is achieved by having the benchmark stream the data to the algorithm in real-time and

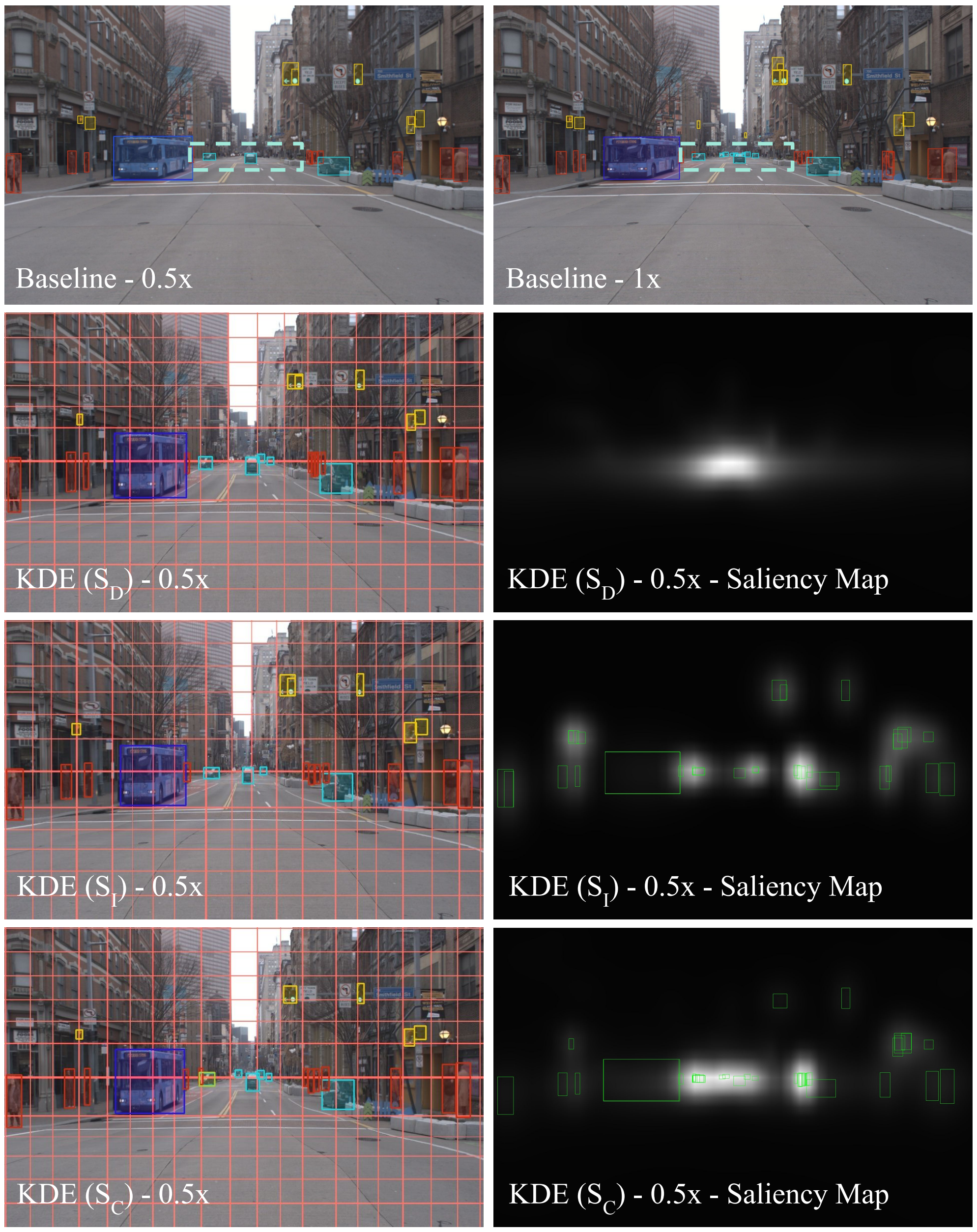

Baseline - 0.5x
Baseline - 1x
KDE (S_D) - 0.5x
KDE (S_D) - 0.5x - Saliency Map
KDE (S_I) - 0.5x
KDE (S_I) - 0.5x - Saliency Map
KDE (S_C) - 0.5x
KDE (S_C) - 0.5x - Saliency Map

Figure 4.7: Qualitative results for our methods after finetuning on Argoverse-HD. The cars in the distance (in the dotted boxes), undetected at 0.5x scale, are detected at 1x scale, and partially detected by our methods. Different rows show the variations within our method based on the source of attention.

ID	Method	AP	AP_S	AP_M	AP_L
1	Prior art [129]	17.8	3.2	16.3	33.3
2	+ Better implementation	19.3	4.1	18.3	34.9
3	+ Train with pseudo GT	21.2	3.7	**23.9**	43.8
4	2 + Ours (S_I)	19.3	5.2	18.5	39.0
5	3 + Ours (S_I)	**23.0**	**7.0**	23.7	**44.9**

Table 4.2: Streaming evaluation in the full-stack (with forecasting) setting on Argoverse-HD. We show that our proposed method significantly improves previous state-of-the-art by 5.2, in which 1.5 is from better implementation, 1.9 is from making use of pseudo ground truth and 1.8 is from our proposed KDE warping.

query for the state of the world at all time instants. One of their key observations is that by the algorithm finishes processing, the world has around changed and therefore proper temporal scheduling and forecasting methods should be used to compensate for this latency. Here we adopt their evaluation protocol for our cost-performance analysis. In our case of streaming object detection, the streaming accuracy refers to *streaming AP*. We use the same GPU (GTX 1080 Ti) and their public available codebase for a fair comparison with their proposed solution. Their proposed solution includes a scale-tuned detector (Faster R-CNN), dynamic scheduler (shrinking-tail) and Kalman Filter forecastor. Our experiments focus on improving the detector and we keep the scheduler and forecastor fixed.

Tab 4.2 presents our evaluation under the full-stack setting (a table for the detection-only setting is included in Appendix 4.A.5. We see that FOVEA greatly improves the previous state-of-the-art. The improvement first comes from a faster and slightly more accurate implementation of the baseline (please refer to Appendix 4.A.6 for the implementation details). Note that under streaming perception, a faster algorithm while maintaining the same offline accuracy translates to an algorithm with higher streaming accuracy. The second improvement is due to training on pseudo ground truth (discussed in Sec 4.4.1). Importantly, our KDE image warping further boosts the streaming accuracy significantly *on top of* these improvements. Overall, these results suggest that *image warping is a cost-efficient way to improve accuracy*.

ID	Method	AP	AP_S	AP_M	AP_L
1	Baseline (0.5x)	15.1	1.0	10.6	**39.0**
2	Ours S_D (0.5x)	13.7	1.3	10.0	34.7
3	Ours S_I (0.5x)	**16.4**	**2.1**	**12.8**	38.6
4	Baseline (0.75x)	19.7	3.0	16.1	**44.2**
5	Ours S_D (0.75x)	18.2	3.4	15.4	40.0
6	Ours S_I (0.75x)	**20.1**	**5.2**	**17.0**	42.5
7	Upper bound (1.0x)	22.6	5.7	20.1	45.7

Table 4.3: Cross-dataset generalization to BDD100K [244]. Rows 2 & 5 are saliency computed on the Argoverse-HD training set, as expected, they fail to generalize to a novel dataset. Despite operating at a larger temporal stride (5 FPS vs 30 FPS), our proposed image-adaptive KDE warping generalizes to a novel dataset (row 3 & 6). Note that here the image native resolution is smaller at 1280×720.

4.4.3 Cross-Dataset Generalization

Our experiments so far are all conducted on the Argoverse-HD dataset. In this section, we cross-validate our proposed method on another autonomous driving dataset BDD100K [244]. Note that BDD100K and Argoverse-HD are collected in different cities. For simplicity, we only test out *off-the-shelf* generalization without any finetuning. We experiment on the validation split of the MOT2020 subset, which contains 200 videos with 2D bounding boxes annotated at 5 FPS (40K frames in total). Also, we only evaluate on common classes between BDD100K and Argoverse-HD: person, bicycle, car, motorcycle, bus, and truck. The results are summarized in Tab 4.3, which demonstrate the generalization capability of our proposed method.

4.5 Conclusion

We propose FOVEA, a highly efficient attentional model for object detection. Our model magnifies regions likely to contain objects, making use of top-down saliency priors learned from a dataset or from temporal context. To do so, we make use of differentiable image warping that ensures bounding box predictions can be mapped back to the original image space. The proposed approach significantly improves over the baselines on Argoverse-HD and BDD100K. For future work, it would be natural to make use of trajectory forecasting models to provide even more accurate saliency maps for online processing.

4.A Appendix

4.A.1 Additional Diagnostic Experiments

The Role of Explicit Backward Label Mapping

Related work either focus on tasks with labels invariant to warping like image classification or gaze estimation [102, 176] (discussed in Sec 4.3.1), or expect an implicit backward mapping to be learned through black-box end-to-end training [152] (discussed in Sec 4.2). In this section, we suggest that the implicit backward label mapping approach is not feasible for object detection. To this end, we train and test our KDE methods minus any bounding box unwarping. Specifically, we no longer unwarp bounding boxes when computing loss during training and when outputting final detections during testing. Instead, we expect the model to output detections in the original image space.

Due to instability, additional measures are taken to make it end-to-end trainable. First, we train with a decreased learning rate of 1e-4. Second, we train with and without adding ground truth bounding boxes to RoI proposals. The main KDE experiments do not add ground truth to RoI proposals, because there is no way of warping bounding boxes into the warped image space (the implementation of $\mathcal{T}$ does not exist). We additionally try setting this option here, because it would help the RoI head converge quicker, under the expectation that the RPN should output proposals in the original space. All other training settings are identical to the baseline setup (Sec 4.4.1).

Results are shown in Tab 4.4. The overall AP is single-digit under all of these configurations, demonstrating the difficulty of implicitly learning the backward label mapping. This is likely due to the fact that our model is pretrained on COCO [139], so it has learned to localize objects based on their exact locations in the image, and finetuning on Argoverse-HD is not enough to “unlearn” this behavior and learn the backward label mapping. Another factor is that in the S_I and S_C cases, each image is warped differently, making the task of learning the backwards label mapping even more challenging. We suspect that training from scratch with a larger dataset like COCO and using the warp parameters (e.g. the saliency map) as input may produce better results. However, this only reinforces the appeal of our method due to ease of implementation and cross-warp generalizability (we can avoid having to train a new model for each warping mechanism).

Sensitivity to Quality of Previous-Frame Detections

Two of our methods, S_I and S_C are dependent on the accuracy of the previous-frame detections. In this section, we analyze the sensitivity of such a dependency through a soft upper bound on S_I and S_C, which is generated using the current frame's ground truth annotations in place of detections from the previous frame. This soft upper bound is a perfect saliency map, up to the amplitude and bandwidth hyperparameters. Note that this is only a change in the testing configuration.

We report results in Tab 4.4. We see a significant boost in accuracy in all cases. Notably, the finetuned KDE S_I model at 0.5x scale achieves an AP of 29.6, outperforming the baseline's accuracy of 29.2 at 0.75x scale.

Sensitivity to Inter-Frame Motion

Having noted that the S_I and S_C formulations are sensitive to the accuracy of the previous-frame detections, in this section, we further test its robustness to motion between frames. We use ground truth bounding boxes (rather than detections) from the previous frame in order to isolate the effect of motion on accuracy. We introduce a jitter parameter j and translate each of the ground truth bounding boxes in the x and y directions by values sampled from $\mathcal{U}(-j, j)$. The translation values are in pixels in reference to the original image size of 1920×1200. As in Sec 4.A.1, this is a purely testing-time change. Also note that the upper bound experiments in Sec 4.A.1 follows by setting $j = 0$. We test only on S_I and report the full results in Tab 4.4. We also plot summarized results and discuss observations in 4.8.

4.A.2 FOVEA Beyond Faster R-CNN

In the main text and other sections of the appendix, we conduct our experiment based on Faster R-CNN. However, our proposed warping-for-detection framework is agnostic to specific detectors. To show this, we test our methods on RetinaNet [138], a popular single-stage object detector, and on YOLOF [35], a recent YOLO variant that avoids bells and whistles and long training schedules (up to 8x for ImageNet and 11x for COCO compared to standard schedules for YOLOv4 [14]).

For both these detectors, we test baselines at 0.5x and 0.75x scales both before and after finetuning. We then compare these results against our KDE S_I method at 0.5x scale. We use a learning rate of 0.01 for the RetinaNet KDE S_I model and 0.005 for the RetinaNet baselines. All other training settings

Argoverse-HD before finetuning

Method	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	person	mbike	tffclight	bike	bus	stop	car	truck
Main Results (copied from the main text for comparison)														
Baseline	21.5	35.8	22.3	2.8	22.4	**50.6**	20.8	9.1	13.9	7.1	48.0	16.1	37.2	20.2
KDE (S_D)	23.3	40.0	22.9	5.4	25.5	48.9	20.9	13.7	12.2	9.3	**50.6**	20.1	40.0	19.5
KDE (S_I)	**24.1**	**40.7**	**24.3**	**8.5**	24.5	48.3	**23.0**	**17.7**	**15.1**	**10.0**	49.5	17.5	**41.0**	19.4
KDE (S_C)	24.0	40.5	**24.3**	7.4	**26.0**	48.2	22.5	14.9	14.0	9.5	49.7	**20.6**	**41.0**	**19.9**
Upp. Bound (0.75x)	27.6	45.1	28.2	7.9	30.8	51.9	29.7	14.3	21.5	6.6	54.4	25.6	44.7	23.7
Upp. Bound (1x)	32.7	51.9	34.3	14.4	35.6	51.8	33.7	21.1	33.1	5.7	57.2	36.7	49.5	24.6
Without an Explicit Backward Label Mapping (Sec 4.A.1)														
KDE (S_D)	5.4	14.2	3.7	0.0	0.9	20.7	3.2	0.4	1.2	0.8	27.9	0.0	5.3	4.2
KDE (S_I)	6.1	15.6	4.0	0.2	0.8	20.3	2.3	0.6	0.7	1.8	30.8	0.0	7.0	5.4
KDE (S_C)	6.0	15.9	3.8	0.1	0.9	21.9	3.0	0.6	0.9	1.5	30.2	0.0	6.7	5.2
Upper Bound with Ground Truth Saliency (Sec 4.A.1)														
KDE (S_I)	25.4	42.6	25.6	9.1	26.2	49.5	25.3	17.4	16.8	10.1	49.4	23.4	41.7	19.4
KDE (S_C)	24.5	41.7	24.6	7.5	26.8	48.8	23.6	14.5	15.2	9.7	49.7	22.6	41.3	19.8
Sensitivity to Inter-Frame Motion (Sec 4.A.1)														
KDE (S_I), $j = 10$	25.3	42.9	25.3	8.4	26.7	49.1	25.0	16.4	16.2	10.1	48.8	25.0	41.8	19.5
KDE (S_I), $j = 25$	24.1	41.0	24.5	6.4	26.1	49.0	24.0	12.6	15.2	9.0	48.5	22.9	41.1	19.6
KDE (S_I), $j = 50$	22.5	38.3	22.9	4.2	24.1	49.1	21.9	9.9	14.4	8.2	48.4	18.5	39.0	19.7
KDE (S_I), $j = 100$	20.9	35.1	21.6	2.8	21.9	48.0	20.1	7.1	14.0	6.8	47.8	15.3	36.7	19.1
KDE (S_I), $j = 200$	20.0	33.5	20.6	2.5	20.5	46.7	19.2	6.0	13.4	6.2	46.7	14.3	35.5	18.5

Argoverse-HD after finetuning

Method	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	person	mbike	tffclight	bike	bus	stop	car	truck
Main Results (copied from the main text for comparison)														
Baseline	24.2	38.9	26.1	4.9	29.0	50.9	22.8	7.5	23.3	5.9	44.6	19.3	43.7	26.6
Learned Sep.	27.2	44.8	28.3	**12.2**	29.1	46.6	24.2	14.0	22.6	7.7	39.5	**31.8**	50.0	27.8
Learned Nonsep.	25.9	42.9	26.5	10.0	28.4	48.5	25.2	11.9	20.9	7.1	39.5	25.1	49.4	28.1
KDE (S_D)	26.7	43.3	27.8	8.2	29.7	54.1	25.4	13.5	22.0	8.0	**45.9**	21.3	48.1	29.3
KDE (S_I)	28.0	45.5	**29.2**	10.4	**31.0**	**54.5**	27.3	16.9	**24.3**	**9.0**	44.5	23.2	**50.5**	28.4
KDE (S_C)	27.2	44.7	28.4	9.1	30.9	53.6	27.4	14.5	23.0	7.0	44.8	21.9	49.9	**29.5**
LKDE (S_I)	**28.1**	**45.9**	28.9	10.3	30.9	54.1	**27.5**	**17.9**	23.6	8.1	45.4	23.1	50.2	28.7
Upp. Bound (0.75x)	29.2	47.6	31.1	11.6	32.1	53.3	29.6	12.7	30.8	7.9	44.1	29.8	48.8	30.1
Upp. Bound (1x)	33.3	53.9	35.0	16.8	34.8	53.6	33.1	20.9	38.7	6.7	44.7	36.7	52.7	32.7
Without an Explicit Backward Label Mapping (Sec 4.A.1)														
KDE (S_D), no RoI GT	2.1	2.6	2.5	0.0	0.0	4.0	0.6	0.0	0.0	0.6	14.8	0.0	0.0	0.9
KDE (S_D)	1.8	2.7	1.9	0.0	0.0	3.2	0.6	0.0	0.0	0.0	13.3	0.0	0.1	0.6
KDE (S_I), no RoI GT	2.5	3.0	2.9	0.0	0.1	4.3	0.7	0.0	0.0	0.6	17.0	0.9	0.0	0.9
KDE (S_I)	2.0	2.8	2.4	0.0	0.0	3.7	0.6	0.0	0.0	0.0	14.8	0.0	0.3	0.5
Upper Bound with Ground Truth Saliency (Sec 4.A.1)														
KDE (S_I)	29.6	48.7	30.7	12.0	32.8	54.4	28.3	16.3	27.7	9.9	43.9	30.6	50.9	28.8
KDE (S_C)	27.8	45.5	28.8	9.6	31.7	53.4	27.5	13.9	24.7	6.5	44.5	25.1	50.2	29.6
Sensitivity to Inter-Frame Motion (Sec 4.A.1)														
KDE (S_I), $j = 10$	29.4	48.3	30.7	11.5	32.8	54.6	27.9	15.9	27.2	9.7	43.7	31.1	50.6	28.7
KDE (S_I), $j = 25$	28.0	46.1	29.2	9.2	32.1	55.3	26.4	13.9	25.9	9.3	43.9	26.8	49.2	28.7
KDE (S_I), $j = 50$	26.2	42.9	27.7	6.6	30.5	54.9	24.1	12.1	24.9	8.6	44.1	21.8	46.2	27.9
KDE (S_I), $j = 100$	24.5	39.9	25.8	4.8	28.6	53.5	22.3	10.2	23.5	7.6	43.5	17.7	43.9	27.1
KDE (S_I), $j = 200$	23.6	38.3	25.2	4.2	27.8	53.0	21.4	8.6	22.8	7.4	42.9	16.6	42.7	26.6

Table 4.4: Additional diagnostics experiments on Argoverse-HD. Please refer to Sec 4.A.1 for a detailed discussion.

for RetinaNet are identical to the Faster-RCNN baseline. For YOLOF, we use a learning rate of 0.012 and keep all other settings true to the original paper. Results are presented in Tab 4.5.

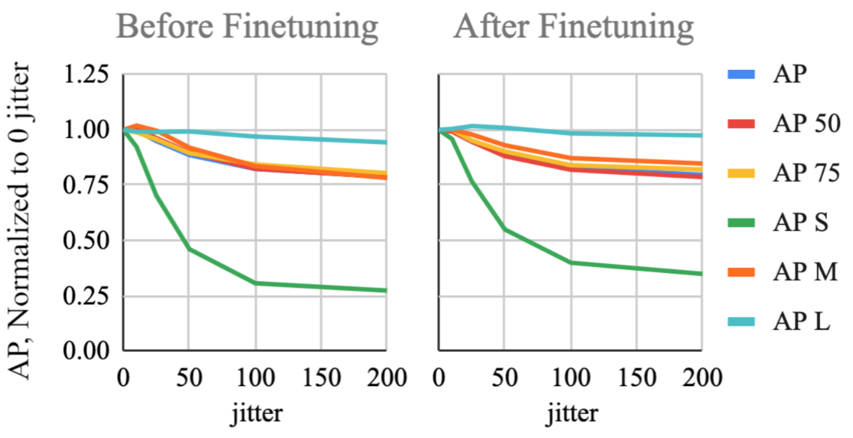


Figure 4.8: Plots showing the effect of motion (jitter) on AP using the KDE S_I formulation. Results have been normalized according to the AP at 0 jitter. As is intuitive, motion affects AP$_S$ the most and AP$_L$ the least. After finetuning (with an artificial jitter of 50), we see that the model reacts less adversely to jitter, indicating that our regularization has helped.

4.A.3 Comparison Against Additional Baselines

There are other approaches that make use of image warping or patch-wise zoom for visual understanding. The first noticeable work [176], explained extensively in the main text, warps the input image for tasks that have labels invariant to warping. The second noticeable work [68] employs reinforcement learning (RL) to decide which patches to zoom in for high-resolution processing. In this section, we attempt to compare our FOVEA with these two approaches.

Our method builds upon spatial transformer networks [102,176] and we have already compared against [176] sporadically in the main text. Here provides a summary of all the differences (see Tab 4.6). A naive approach might directly penalize the discrepancy between the output of the (warped) network and the unwarped ground-truth in an attempt to implicitly learn the inverse mapping, but this results in abysmal performance (dropping 28.1 to 2.5 AP, discussed in Sec 4.A.1). To solve this issue, in Sec 4.3.1, we note that [102,176] actually learn a backward map $\mathcal{T}^{-1}$ instead of a forward one $\mathcal{T}$. This allows us to add a backward-map layer that transforms bounding box coordinates back to the original space via $\mathcal{T}^{-1}$, dramatically improving accuracy. A second significant difference with [102,176] is our focus on attention-for-efficiency. If the effort required to determine where to attend is more than the effort to run the raw detector, attentional processing can be

Method	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
RetinaNet, Before Finetuning on Argoverse-HD						
Baseline (0.5x)	18.5	29.7	18.6	1.3	17.2	48.8
KDE (S_I)	18.5	31.2	17.9	4.5	16.8	44.9
Upp. Bound (0.75x)	24.8	38.8	25.5	4.5	28.7	52.0
RetinaNet, After Finetuning on Argoverse-HD						
Baseline (0.5x)	22.6	38.9	21.4	4.0	22.0	53.1
KDE (S_I)	24.9	40.3	25.3	7.1	27.7	50.6
Upp. Bound (0.75x)	29.9	48.6	30.1	9.7	32.5	54.2
YOLOF, Before Finetuning on Argoverse-HD						
Baseline (0.5x)	15.0	25.4	14.3	0.6	11.0	46.0
KDE (S_I)	16.8	29.0	16.0	0.9	14.0	46.4
Upp. Bound (0.75x)	21.6	35.5	22.3	2.3	22.2	52.7
YOLOF, After Finetuning on Argoverse-HD						
Baseline (0.5x)	18.4	30.5	18.3	1.4	16.5	47.9
KDE (S_I)	21.3	36.7	20.2	3.5	21.8	49.7
Upp. Bound (0.75x)	25.1	41.3	25.3	4.7	27.6	54.1

Table 4.5: Experiments with RetinaNet [138] and YOLOF [35]. We follow the same setup as the experiment with Faster R-CNN. The top quarter suggests that unlike Faster R-CNN, RetinaNet does not work off-the-shelf with our KDE warping. However, the second quarter suggests similar performance boosts as with Faster R-CNN can be gained after finetuning on Argoverse-HD. Interestingly, for YOLOF, our method boosts AP in all categories – small, medium, and large – even with off-the-shelf weights.

inefficient (see the next paragraph). [176] introduces a lightweight saliency network to produce a heatmap for where to attend; however, this model does not extend to object detection, perhaps because it requires the larger capacity of a detection network (see Sec 4.4.1). Instead, we replace this feedforward network with an essentially *zero*-cost saliency map constructed via a simple but effective global spatial prior (computed offline) or temporal prior (computed from previous frame's detections). Next, we propose a technique to prevent cropping during warping (via reflection padding, as shown in Fig 4.5), which also boosts performance by a noticeable amount. Finally, as stated in the training formulation in Sec 4.3.2, it *doesn't even make sense* to train a standard RPN-based detector with warped input due to choice of delta encoding (which normally helps stabilize training). We must remove this standard encoding and use GIoU to compensate for the lost stability during training.

Method	AP
FOVEA (Ours full)	28.1
w/o Explicit backward mapping	2.5
w/o KDE saliency (using saliency net as in [176])	Doesn't train
w/o Anti-crop regularization	26.9
w/o direct RPN box encoding	N/A

Table 4.6: Summary of key modifications in FOVEA.

Next, we attempt to compare against this RL-based zoom method [68] using our baseline detector (public implementation from mmdetection [33]) on their Caltech Pedestrian Dataset [53]. However, while their full-scale 800×600 Faster R-CNN detector reportedly takes 304ms, our implementation is *dramatically* faster (44ms), consistent with the literature for modern implementations and GPUs. This changes the conclusions of that work because full-scale processing is now faster than coarse plus zoomed-in processing (taking 28ms and 25ms respectively), even assuming a zero-runtime RL module (44ms < 28ms + 25ms).

4.A.4 Additional Visualizations

Please refer to Fig 4.9 and 4.10 for additional qualitative results of our method.

4.A.5 Detection-Only Streaming Evaluation

In Sec 4.4.2 of the main text, we provide the full-stack evaluation for streaming detection. Here we provide the detection-only evaluation for completeness in Tab 4.7. This setting only allows detection and scheduling, and thus isolating the contribution of tracking and forecasting. We observe similar trend as in the full-stack setting in Tab 4.2.

4.A.6 Additional Implementation Details

In this section, we provide additional details necessary to reproduce the results in the main text.

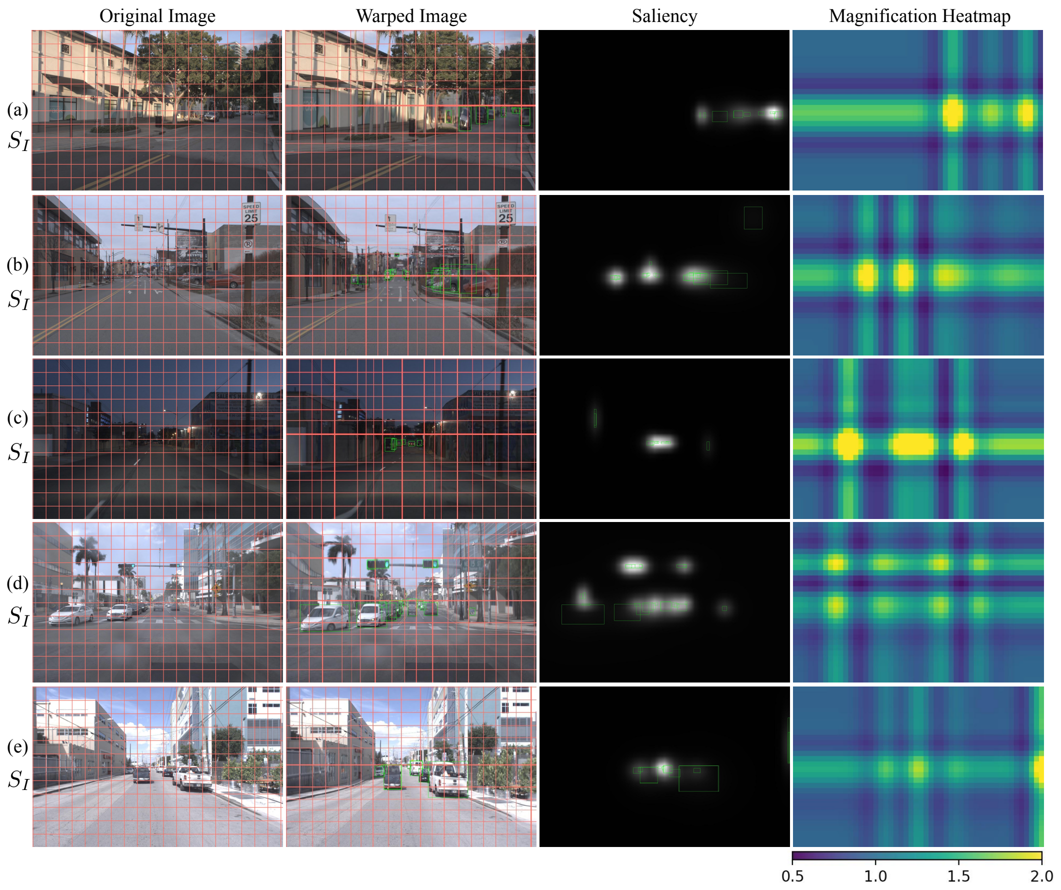


Figure 4.9: Additional examples of the S_I KDE warping method. Bounding boxes on the saliency map denote previous frame detections, and bounding boxes on the warped image denote current frame detections. The magnification heatmap depicts the amount of magnification at different regions of the warped image. (a) is an example of S_I correctly adapting to an off-center horizon. (b) shows a multimodal saliency distribution, leading to a multimodal magnification in the x direction. (c) is another example of S_I correctly magnifying small objects in the horizon. (d) is a failure case in which duplicate detections of the traffic lights in the previous frame leads to more magnification than desired along that horizontal strip. One solution to this could be to weight our KDE kernels by the confidence of the detection. (e) is another failure case of S_I, in which a small clipped detection along the right edge leads to extreme magnification in that region. One general issue we observe is that the regions immediately adjacent to magnified regions are often contracted. This is visible in the magnification heatmaps as the blue shadows around magnified regions. This is a byproduct of the dropoff in attraction effect of the local attraction kernel. Perhaps using non-Gaussian kernels can mitigate this issue.

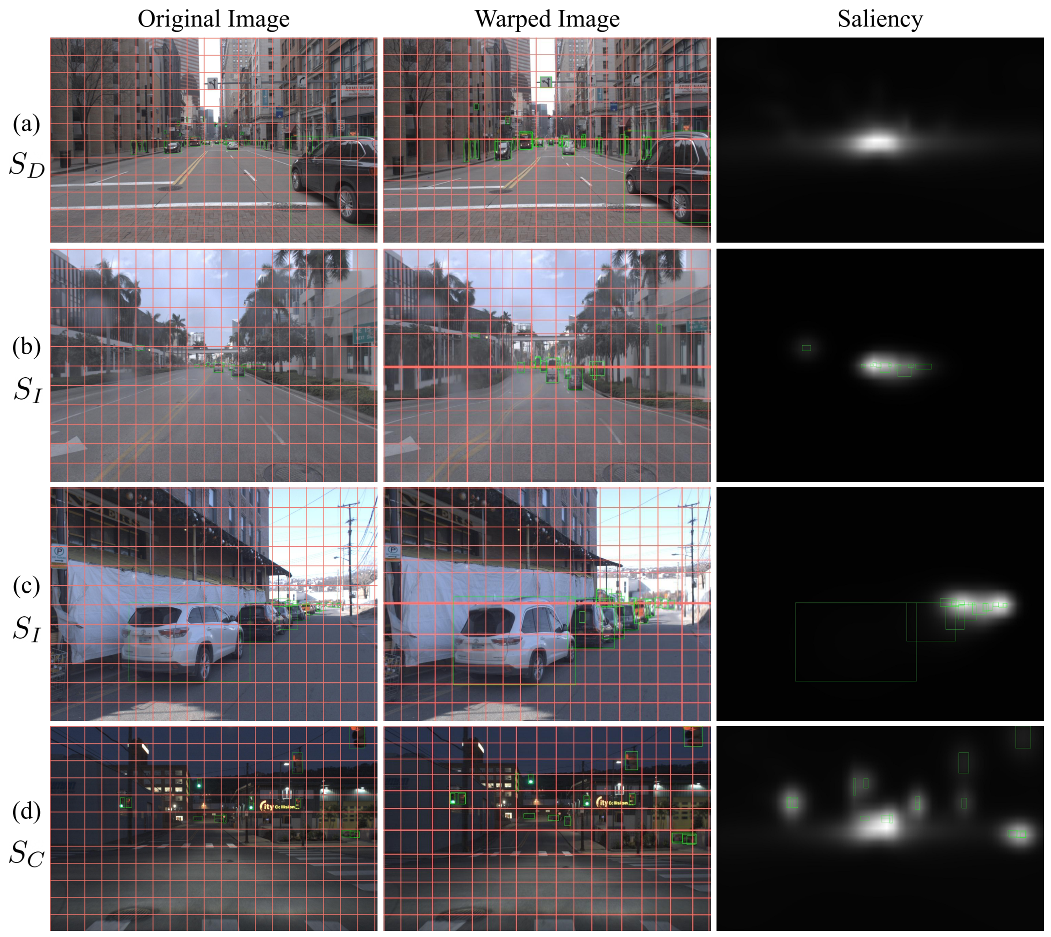


Figure 4.10: Examples of KDE warp computed from bounding boxes, extracted from a training dataset (S_D) or the previous frame's detections (S_I, S_C). We visualize predicted bounding boxes in the warped image. Recall that large objects won't be visible in the saliency due to their large variance from Eq 4.8. (a) S_D magnifies the horizon (b) S_I magnifies the center of the image, similar to S_D (c) S_I adapts to magnify the mid-right region (d) S_C's saliency combines the temporal and spatial biases.

ID	Method	AP	AP_S	AP_M	AP_L
1	Prior art [129]	13.0	1.1	9.2	26.6
2	+ Better implementation	14.4	1.9	11.5	**27.9**
3	+ Train with pseudo GT	15.7	3.0	14.8	27.1
4	2 + Ours (S_I)	15.7	4.7	12.8	26.8
5	3 + Ours (S_I)	**17.1**	**5.5**	**15.1**	27.6

Table 4.7: Streaming evaluation in the detection-only setting. First, we are able to improve over previous state-of-the-art through better implementation (row 2) and training with pseudo ground truth (row 3). Second, our proposed KDE warping further boosts the streaming accuracy (row 4-5).

For the learned separable model from Sec 4.4.1, we use two arrays of length 31 to model saliency along the x and y dimensions, and during training, we blur the image with a 47×47 Gaussian filter in the first epoch, a trick introduced in [176] to force the model to zoom. For the learned nonseparable model, we use an 11×11 saliency grid, and we blur the image with a 31×31 filter in the first epoch. We use an attraction kernel k with a standard deviation of 5.5 for both versions. Additionally, we multiply the learning rate and weight decay of saliency parameters by 0.5 in the first epoch and 0.2 in the last two epochs, for stability. We find that we don't need anti-crop regularization here, because learning a fixed warp tends to behave nicely.

For each of our KDE methods, we use arrays of length 31 and 51 to model saliency in the vertical and horizontal directions, respectively. This is chosen to match the aspect ratio of the original input image and thereby preserve the vertical and horizontal "forces" exerted by the attraction kernel.

For the baseline detector, we adopt the Faster R-CNN implementation of mmdetection 2.7 [33]. All our experiments are conducted in an environment with PyTorch 1.6, CUDA 10.2 and cuDNN 7.6.5. For streaming evaluation, we mention a performance boost due to better implementation in Tab 4.7 & Tab 4.2, and the changes are mainly adopting newer versions of mmdetection and cuDNN compared to the solution in [129] (switching from a smooth L1 loss to L1 loss for the regression part and code optimization).

Chapter 5

Progressive Knowledge Distillation

5.1 Introduction

The success of recent deep neural network models generally depends on an elaborate design of architectures with tens or hundreds of millions of model parameters. However, their huge computational complexity and massive memory/storage requirements make them challenging to be deployed in safety-critical real-time applications, especially on devices with limited resources, such as self-driving cars or virtual/augmented reality models. Such concerns have spawned a wide body of literature on compression and acceleration techniques. Many approaches focus on reducing computation demands by sparsifying/pruning networks [80, 124], quantization [174, 232], or neural architecture search [140, 262], but reduced computation does not always translate to lower latency because of subtle issues with memory access and caching on GPUs [52, 204].

Distillation: Rather than searching over new architectures, we seek to better train *existing* lightweight architectures that have already been carefully engineered for efficient memory access. Instead of relying on additional data or human supervision, we follow the large body of work on knowledge distillation [88,246], first proposed by Buciluă *et al*. for compressing the information from a large ensemble of models into a small model [20]. While most recent efforts in knowledge distillation focus on image classification, relatively less work exists for distilling object detectors. It is nontrivial to extend classification distillation methods to object detection and instance segmentation due to the complicated outputs of the tasks. Most detectors

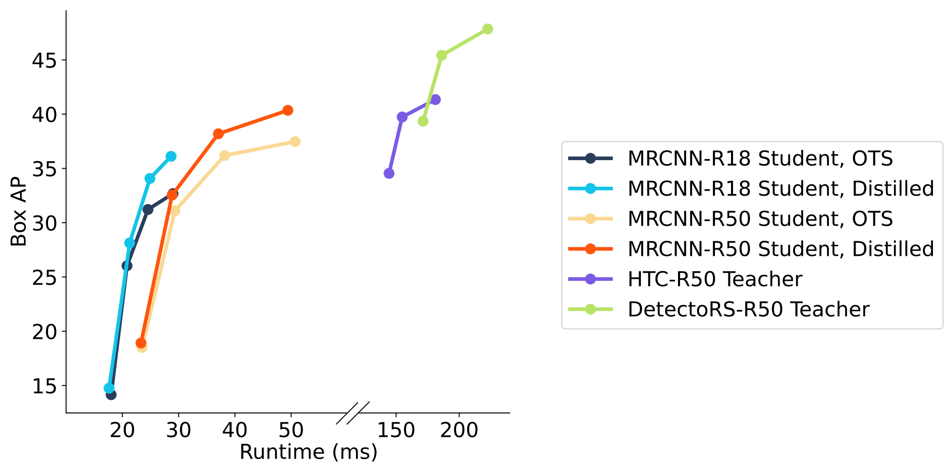


Figure 5.1: Progressive knowledge distillation on MS COCO [139]. We show the accuracy-efficiency trade-off for state-of-the-art lightweight detectors obtained by varying input image resolutions, focusing on improving off-the-shelf ('OTS') Mask R-CNN with ResNet-18 (black) and ResNet-50 (beige) backbones. Sequential distillation with more accurate but slower teachers (*first* the purple and *then* the green) strictly improves student performance without any increase in inference time. Distillation is particularly effective for lightweight networks, improving accuracy of ResNet-18 based Mask R-CNN by **3.7%** overall AP and **6.3%** large-AP (the latter of which is particularly relevant for finding nearby/large objects in an autonomous navigation context).

operate with multi-task heads (for region proposal generation, bounding box regression, and classification) that can generate variable-length outputs. Many distillation methods make use of internal features for distillation, but detection networks typically have complex modules that are hard to align across different architectures. Moreover, teacher architectures that are significantly larger than a student may serve as poor targets for distillation because of the *capacity gap* between the two models [39, 157].

Our approach: To address these challenges, we propose a method to learn lightweight detectors through *progressive modular knowledge distillation*. Specifically, we focus on two problems: First, what knowledge should be transferred from the teacher to the student? Second, how can we resolve the capacity gap between a large teacher and a small student? Typical knowledge distillation uses the logits of a teacher network as targets for learning by the student network [88]. Activations of intermediate layers can also be used as targets [1, 99, 184, 246]. We follow this line of work, and use intermediate activations of the teacher to supervise the student, making use of *modular backbone-neck-head structures* to align student and teacher features. This enables us to distill a student using teachers with different backbone architectures or even different input resolutions. For the second question, although networks with more advanced architectures tend to have better performance, empirical results show that a larger model may not serve as a better teacher because large capacity gaps between the teacher and student can degrade knowledge transfer [39]. On the other hand, the architectural similarity between the teacher and student can significantly influence the effectiveness of distillation [157]. Because different teachers may provide complementary knowledge to a student, several multi-teacher distillation methods have recently been proposed [190, 216, 245]. One rather straightforward approach might be using the average response (of logits or features) across all teachers as the supervision signal [88]. We find that *sequential distillation of multiple teachers arranged into a curriculum* significantly improves progressive knowledge transfer. Given a student, we design a heuristic algorithm to determine the order of teachers to use. Furthermore, by analyzing the training loss dynamics of the student model, we find the improvement is *not* due to minimizing the training loss better. Rather, the knowledge transferred from multiple teachers can lead the student to a flat minimum, and thus help the student *generalize* better.

To summarize, we use the feature-based knowledge from multiple teachers to progressively distill a student. Our main contributions include:

- We propose a framework for learning lightweight detectors through progressive knowledge distillation. Our approach is simple, straightforward, yet effective.
- We develop a principled way to automatically design a sequence of teachers appropriate for a specific student and progressively distill the student.
- We perform comprehensive empirical evaluation on the challenging MS COCO dataset [139]. We have observed consistent gains ($> 3\%$ AP), summarized in Figure 5.1 and Table 5.3&5.4.
- We show that the performance gain comes from better generalization rather than better optimization.

5.2 Related Work

Knowledge Distillation: Knowledge distillation or transfer, an idea of training a shallow student network with supervision from a deep teacher, was originally proposed in [20], and later formally popularized by [88]. Different categories of knowledge can be used, such as response-based knowledge [88], feature-based knowledge [87, 184], and relation-based knowledge [240]. Several multi-teacher knowledge distillation methods have recently been proposed [190, 216], which usually use the average of logits and feature representations as the knowledge [243], or randomly select one teacher from the pool of teacher networks at each iteration [66]. Mirzadeh *et al*. [157] find that an intermediate teacher assistant (which is decided based on architectural similarities) can bridge the gap between the student and the teacher. We find it more effective to use a sequence of teachers instead of their ensemble, and extend [157] to a more general case where teacher models have diverse architectures and their relative ordering is unknown.

Object Detection and Instance Segmentation: The past several years have seen remarkable progress in object detection and instance segmentation. A variety of convolutional neural network (CNN) based object detection frameworks have been proposed and could be generally divided into two categories: single-stage methods and two-stage methods. Typical single-stage methods include YOLO [178, 179] and RetinaNet [138], and typical two-stage methods include Faster R-CNN [180], R-FCN [44], and Mask R-CNN [82]. Recently, several multi-stage frameworks are proposed and achieve the state-of-the art performance, such as HTC [32] and DetectoRS [169]. These detection frameworks achieve better detection accuracy with better backbone networks as feature extractors and with more complicated heads, which are more computationally expensive.

Knowledge Distillation for Detection and Segmentation: To reduce the computational cost, knowledge distillation has been used to develop efficient detectors [30,43,51,100,143,196,221]. Li *et al*. [131] mimic ROI-pooled feature responses between a student and teacher to learn an efficient detector. Shmelkov *et al*. [196] mimic the logit responses from ROI-pooled features between a student and teacher to combat catastrophic forgetting during incremental learning. Chen *et al*. [31] use mimic learning between the CNN backbones of a teacher and student Faster R-CNN. Mehta and Ozturk [155] propose objectness scaled distillation which weighs the loss incurred by each teacher predicted object using its confidence score. Wang *et al*. [219] apply mimic learning to imitate the responses of a teacher on regions near the ground-truth boxes. Zhang *et al*. [248] use attention and non-local modules to guide distillation, enabling the student to focus on foreground objects and learn relation between objects. Guo *et al*. [78] decouple features from object and background regions and assign different importance for the student to learn. Dai *et al*. [46] make use of general instance patches, and design feature-based and relation-based distillation losses. Different from these methods that distill a single teacher, we study distillation from multiple teachers where a proper sequence of teachers is required. Also, we find a very simple feature matching loss is adequate to significantly boost student performance, and thus our training pipeline is more efficient than previous methods.

5.3 Approach

We propose to progressively distill a student model S with a pool of N teachers $\mathcal{P} = \{T_i\}_{i=1}^{N}$. Both the student and teacher models are composed of four modules: (1) backbone architecture, which is used for feature extraction, such as ResNet [84] and ResNeXt [233]; (2) neck, which is used to extract multi-level feature maps from different stages of the backbone, such as FPN [137] and Bi-FPN [205]; (3) optional region proposal network (RPN), which is used in two-stage detectors for sparse prediction; and (4) head, which generates final predictions for object detection and segmentation. We denote the output feature maps of the *neck* as F^{Net}, where Net can be either the student model S or one of the teachers $T_i \in \mathcal{P}$. With neck modules like FPN, the feature maps can be multiple-level. We first introduce how to distill with a single teacher T_i in Section 5.3.2 and then introduce how to distill with multiple teachers in Section 5.3.3.

5.3.1 Models

To test the versatility of our progressive knowledge distillation strategy, we consider a diverse set of object detectors, namely, RetinaNet [138], Mask R-CNN [82], FCOS [210], HTC [32], and DetectoRS [169]. These networks have a wide range of runtime and detection performance.

RetinaNet and FCOS are single-stage detectors, consisting of ResNet backbone, FPN neck, and a detection head. RetinaNet produces dense predictions based on anchors, while FCOS is an anchor-free method producing multi-level, per-pixel predictions.

Mask R-CNN, HTC, and DetectoRS are two/multi-stage detectors, consisting of ResNet backbone, FPN neck, region proposal networks (RPN), and prediction heads. In Mask R-CNN, the prediction heads predict categories, refine the bounding box, and generate a pixel mask of the object based on the first stage proposal. HTC is a cascading framework interweaving detection and segmentation for a joint multi-stage processing. DetectoRS

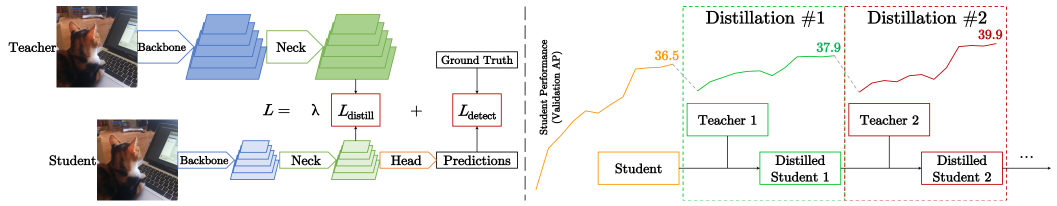


Figure 5.2: Progressive knowledge distillation for object detectors. **Left**: For each teacher-student pair, the training target is composed of two parts: L_{distill} minimizes the discrepancy between the neck feature maps of the student and the current teacher, and L_{detect} is the original detection loss based on the ground truth. **Right**: We use a *sequence* of teacher models to distill the lightweight student detector. The sequence of teachers forms a curriculum. Using a proper sequence of teachers can significantly boost the student model's performance. The example performance curve illustrates our method improves the COCO validation AP of ResNet-50 backboned RetinaNet student first from 36.5% to 37.9% using HTC (Teacher 1), and then from 37.9% to 39.9% using DetectoRS (Teacher 2).

extends HTC with the switchable atrous convolution (SAC) and the recursive feature pyramid (RFP).

We select RetinaNet and Mask R-CNN as the student models, due to their low latency, simple structure, and wide application, for single-stage and two-stage object detection respectively. More advanced models such as DetectoRS have better detection performance, but require much more training/inference time, so we use them as teachers. We aim to achieve a higher performance without changing the student model's architecture, with the help of a sequence of teacher models from the pool.

5.3.2 Single Teacher Distillation

In order to learn a fast yet accurate student detector S through distillation, we encourage the feature representation of a student network to be similar to that of the teacher network [31, 237]. To this end, we minimize the discrepancy between the feature representations of the teacher and the student. Without bells and whistles, we simply minimize the L2 distance between F^{T_i} and F^S:

$$L_{\text{distill}} = \left\| F^{T_i} - r(F^S) \right\|_2^2, \tag{5.1}$$

where $r(\cdot)$ is a function used to match the feature map dimensions of the teacher and the student.

We define $r(\cdot)$ as follows:

- (Homogeneous) If the numbers of channels C and the spatial resolutions H, W are both the same between T_i and S, *i.e.*, $C^{T_i} = C^S$, $H^{T_i} = H^S$, $W^{T_i} = W^S$, $r(\cdot)$ is an identity function.
- (Heterogeneous, different channels) If the numbers of channels C are different and the spatial resolutions H, W are the same, *i.e.*, $C^{T_i} \neq C^S$, $H^{T_i} = H^S$, $W^{T_i} = W^S$, we use 1×1 convolutional filters as $r(\cdot)$.
- (Heterogeneous, different resolutions) If the spatial resolutions H, W are different and the numbers of channels C are the same, *i.e.*, $C^{T_i} = C^S$, $H^{T_i} \neq H^S$, $W^{T_i} \neq W^S$, we use an upsampling layer as $r(\cdot)$.

Note that the mapping $r(\cdot)$ is only required at training time and thus *not adding any overhead* to the inference.

Our overall loss function can be written as:

$$L = \lambda L_{\text{distill}} + L_{\text{detect}}, \tag{5.2}$$

where λ is a balancing hyper-parameter and L_{detect} is the detection loss based on the ground truth labels. Compared to state-of-the-art detection distillation approaches [31,78,219,248], which introduce more complex designs of the distillation loss, our method is simpler and does not require running the heads of the teacher model. Our distillation loss is illustrated in Figure 5.2-**Left**.

5.3.3 Progressive Distillation with Multiple Teachers

The overall aim of knowledge distillation is to make a student mimic a teacher's output, so that the student is able to obtain similar performance to that of the teacher. However, the capacity of the student model is limited, making it hard for the student to learn from a highly complex teacher [39]. To address this issue, multiple teacher networks are used to provide more supervision to a student [190, 243]. Unlike previous methods [120, 162, 207, 231, 247] which distill knowledge from the ensemble of logits or feature information simultaneously, we propose to distill feature-based knowledge from multiple teachers *sequentially*. Our key insight is that instead of mimicking the ensemble of all feature information together, the student can be distilled more effectively by the knowledge provided by one proper teacher each time. This progressive knowledge distillation approach can be considered as designing a curriculum [10] offered by a sequence of teachers, as illustrated in Figure 5.2-**Right**.

The crucial question is: *What is the proper order $\mathcal{O}$ of the teachers when distilling the student?* A brute-force approach might search over all orders and pick the best (that produces a distilled student with the highest validation accuracy). However, the space of permutation orders grows exponentially with the number of teachers, making this impractical to scale. Therefore, we propose a principled and efficient approach based on a correlation analysis of each model's learned feature representation.

First, we quantify the dissimilarity between each pair of models' representations, as a proxy for the capacity gap between the two models. Representation (dis)similarity [113, 172, 218] has been studied to understand the learning capacity of neural models. In our setting, we find a linear regression model is adequate for measuring the representation dissimilarity. Given two pre-trained detector models A and B, we can freeze the two models' parameters, thus fixing the feature representations. Then we can learn a linear mapping $r(\cdot)$, implemented by a 1×1 convolutional layer at each feature level, as specified in the heterogeneous case in Section 5.3.2. $r(\cdot)$ is trained to minimize L_{distill}, so it can transform A's features to approximate B's features. After training $r(\cdot)$, we evaluate it by L_{distill} on the validation set, and denote the validation loss as the adaptation cost $\mathcal{C}(A, B)$. This metric can be a proxy of the capacity gap between a pair of models: When $\mathcal{C}(A, B)$ is zero, a linear mapping can transform A's features to B's, and there is no additional knowledge from B. When $\mathcal{C}(A, B)$ is large, it is more difficult to adapt A's representation to B's. Note that the adaptation cost is non-symmetric – it is relatively easier to adapt a high-capacity model's representations to a low-capacity model's representations, than the other way around.

We design a heuristic algorithm to acquire a proper distillation order $\mathcal{O}$ automatically, as shown in Algorithm 2. Suppose the maximum number of teachers to be selected is limited by k (which can be arbitrarily decided according to desired training time), and we aim to find a teacher index sequence α no longer than k. We construct the teacher order backwards: The best performing teacher is set as the final target T_{α_k}; before the final teacher, we use another teacher, which has the smallest adaptation cost $\mathcal{C}(\cdot, T_{\alpha_k})$ to that final teacher, as the penultimate teacher $T_{\alpha_{k-1}}$. We repeat this procedure to find preceding teachers, until: (1) when trying to select T_{α_j}, we find the transfer costs from remaining teachers to the next teacher $\mathcal{C}(\cdot, T_{\alpha_{j+1}})$ are all larger than the transfer cost from the student to the next teacher $\mathcal{C}(S, T_{\alpha_{j+1}})$; or (2) we reach the given maximum step limit k. Intuitively, the resulting sequence of teachers bridges the gap between the student model and the teacher, with an increasingly difficult curriculum.

Algorithm 2: Determining the Teacher Order

```
Input: Student model $S$
       Pool of teacher models $\mathcal{P} = \{T_i\}_{i=1}^{N}$
       Teacher models' performance $\{Q(T_i)\}_{i=1}^{N}$
       Maximum number of selected teachers $k$
Output: Sequence of teachers $\mathcal{O}$, len$(\mathcal{O}) \leq k$
Pick the best performing teacher: $T_{\alpha_k} \leftarrow \arg\max_{T_u \in \mathcal{P}} Q(T_u)$,
 $\mathcal{O} \leftarrow [T_{\alpha_k}]$
Exclude from pool: $\mathcal{P} \leftarrow \mathcal{P} \setminus \{T_{\alpha_k}\}$
for $j \leftarrow k-1$ to $1$ do
    Get candidate sub-pool:
     $\mathcal{P}_j = \{T_u \mid T_u \in \mathcal{P}, \mathcal{C}(T_u, T_{\alpha_{j+1}}) < \mathcal{C}(S, T_{\alpha_{j+1}})\}$
    if $\mathcal{P}_j \neq \emptyset$ then
        Pick the teacher closest to $T_{\alpha_{j+1}}$:
         $T_{\alpha_j} \leftarrow \arg\min_{T_u \in \mathcal{P}_j} \mathcal{C}(T_u, T_{\alpha_{j+1}})$
        Prepend $T_{\alpha_j}$ to $\mathcal{O}$
        Exclude from pool: $\mathcal{P} \leftarrow \mathcal{P} \setminus \{T_{\alpha_j}\}$
    else
        Break
    end
end
return $\mathcal{O}$
```

Our algorithm for designing teacher orders is lightweight. In fact, the main computation overhead of our algorithm is to train a set of tiny linear mappings ($\mathbb{R}^{256} \mapsto \mathbb{R}^{256}$ for FPN-based [137] detectors). It takes about 3 GPU hours for each student model, which is negligible compared to the distillation process that takes hundreds of GPU hours.

5.4 Experiments

We experiment with a variety of student-teacher pairs on object detection and instance segmentation tasks. We mainly focus on distilling the vanilla RetinaNet [138] and Mask R-CNN [82] models to approach the performance of slower, more sophisticated architectures. We investigate the impacts of different teachers and their orders. We show the generality and robustness of our framework on different datasets. We evaluate the improved accuracy-efficiency trade-off not only in the standard offline setting, but also in the recently proposed streaming perception scenario [129].

Model	Input Res.	Backbone	Neck	Head	AP Box	AP Mask	Runtime (ms)
Teachers							
I	$1\times$	R50	FPN	Mask R-CNN	38.2	34.7	51
II	$1\times$	R50	FPN	FCOS	38.7	-	36
III	$1\times$	R50	FPN	HTC	42.3	37.4	181
IV	$1\times$	R50+SAC	RFP	HTC (DetectoRS)	49.1	42.6	223
V	$1\times$	R50+SAC	RFP	Mask R-CNN	45.1	40.1	142
Students							
I	$1\times$	R50	FPN	RetinaNet	36.5	-	43
II	$1\times$	R50	FPN	Mask R-CNN	38.2	34.7	51
III	$1\times$	R18	FPN	Mask R-CNN	33.3	30.5	29
IV	$0.25\times$	R50	FPN	Mask R-CNN	25.8	23.0	17

Table 5.1: Configurations of the student and teacher detectors, and their performance on the COCO benchmark. We investigate a variety of models with heterogeneous input resolutions, backbones, necks, and head structures. '1×' input resolution refers to the standard 1333×800 resolution, and '$0.25\times$' means 333×200 resolution. 'R-' backbones are ResNets with different number of layers.

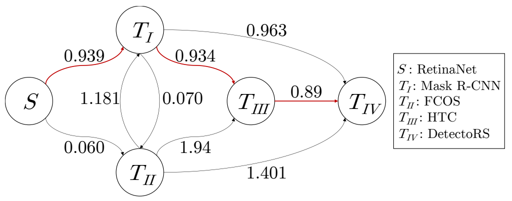


Figure 5.3: Adaptation costs among models. The number on each directed edge is the adaptation cost metric described in Section 5.3.3. Some edges are not shown for visual clarity. The red path is suggested by our proposed Algorithm 2 when $k = 3$ teachers are selected: (1) use the best performing Teacher IV as the final teacher in the sequence, (2) use the teacher closest to Teacher IV, which is Teacher III, as the second teacher, and (3) use the teacher closest to Teacher III, which is Teacher I, as the first teacher.

Datasets: We evaluate on two challenging object detection datasets: MS COCO 2017 [139] (licensed under CC BY 4.0) and Argoverse-HD [129] (licensed under CC BY-NC-SA 4.0). The gold-standard COCO dataset contains bounding boxes and instance segmentations for 80 common object categories. We train our models on the split of `train2017` (118k images) and report results on `val2017` (5k images). Due to space limit, we discuss details and experiment results of Argoverse-HD in the supplementary materials.
Evaluation Metrics: We report the standard COCO-style Average Precision (AP) metric which averages AP across Intersection over Union (IoU) thresholds from 0.5 to 0.95 with an interval of 0.05. We report AP for large, medium, and small objects, and with IoU thresholds at 0.5 and 0.75. We report end-to-end latency (from an unprocessed image to the final bounding boxes and masks) as the runtime, and measure it on a single NVIDIA Tesla V100 Graphics Card. We report streaming AP for accuracy-latency trade-off evaluation, which is explained in the supplementary materials.
Student and Teacher Models: To investigate the impact of different teacher models and their combinations, as shown in Table 5.1, we construct a variety of student-teacher pairs from a set of state-of-the-art object detection and in-

stance segmentation networks, including Mask R-CNN [82], RetinaNet [138], FCOS [210], HTC [32], and DetectoRS [169].

Baselines: We compare with three most recent state-of-the-art approaches to detector distillation [46,78,248]. We directly use the *best* reported results from their paper for comparison, even though each previous work uses a different teacher model that best fits the method. Our main contribution is *orthogonal* to previous methods: We leverage a *sequence* of teachers to distill the student, instead of designing a sophisticated distillation loss to better transfer knowledge from one single teacher. Since we are studying a new setting where multiple teachers are available, which is missing in previous literature, we mainly focus on the *absolute improvements* – the performance of our distilled student models compared with the original student models and with the performance upper-bound of the teacher models. Meanwhile, the combination of these two orthogonal directions can lead to further improvement. We demonstrate this by combining the attention-guided and non-local distillation from [248] (which includes public code and results for both of our student models) with our sequential distillation. We find using a sequence of teachers, instead of their ensemble, is more effective. Due to space limit we leave this comparison in the supplementary material.

Implementation Details: We implement detectors using the MMDetection codebase [33]. We train on 8 GPUs for 12 epochs for each distillation. For MS COCO, we use the standard input resolution of $1,333 \times 800$, with each GPU hosting 2 images. We use an initial learning rate of 0.01 (for RetinaNet students) or 0.02 (for Mask R-CNN students). We use stochastic gradient descent and a momentum of 0.9. We perform a grid search over the hyper-parameter λ. While the optimal values are dependent on the architectures of the teacher and student models, we find the performance is not very sensitive to λ between 0.3 and 0.8. We set $\lambda = 0.5$ for RetinaNet students and $\lambda = 0.8$ for Mask R-CNN students.

5.4.1 Searching for the Near-Optimal Teacher Order

As we have discussed in Section 5.3.3, finding the optimal order of teachers for the progressive knowledge distillation takes factorial time complexity. To acquire a near-optimal teacher order, we propose the heuristic Algorithm 2. In this section, we will validate that this algorithm can provide highly competitive teacher orders.

To achieve this comprehensive comparison, we distill Student I with *all orders* of teachers from the pool Teacher I-IV. We use a reduced training budget: For each teacher, we only train the student for 3 epochs on MS COCO.

k	Suggested teacher order	Student AP	All student AP range	Ranking in all orders
1	IV	36.7	[36.2, 36.8]	2 / 4
2	III→IV	37.6	[36.2, 37.6]	1 / 16
3	I→III→IV	37.9	[36.2, 38.0]	2 / 40
4	I→III→IV	37.9	[36.2, 38.2]	7 / 64

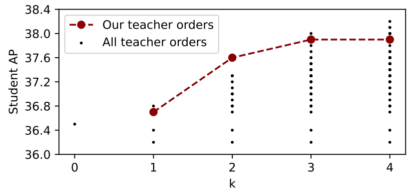


Table 5.2: Comparison of teacher order suggested by Algorithm 2 with all other orders under limited training budgets [130]. k denotes the maximum number of used teachers. **Left**: We show some statistics of possible student AP performance and the ranking of the student using our distillation order. **Right**: We visualize the comparative advantage of our teacher orders (red dots) over all other orders (black dots). Some black scatter points overlap due to the same student AP. Our proposed Algorithm 2 can consistently produce highly competitive distillation orders of teachers.

We use the linear learning rate schedule, which has been shown comparably effective in a limited budget setting by [130].

We first measure the adaptation costs among the student and teacher models. A visualization of the cost graph is shown in Figure 5.3. Following Algorithm 2, we can construct a sequence of teachers. We compare the teacher orders given by our proposed algorithm against *all other* orders, via the performance of the distilled student's performance. As shown in Table 5.2, teacher orders suggested by Algorithm 2 are consistently near-optimal in this setting. In the following sections, we will use order provided by Algorithm 2, without brute-force iterating over all possible orders. One may question that the greedy path selection shown in Figure 5.3 is be inferior to a global optimization algorithm. However, we find the later teachers impact the student performance more profoundly, so we need to greedily select teachers from the sequence tail. More details and comparison with other heuristics are provided in the supplementary material.

5.4.2 Distillation with Homogeneous Teachers

We start by distilling RetinaNet and Mask R-CNN with a ResNet-50 backbone (Student I & II). Here we consider *homogeneous* teachers where the numbers of channels and the spatial resolutions of feature maps are *consistent* between the student and teacher. For the RetinaNet student, we still consider the pool of Teacher I-IV, the same as Section 5.4.1. To control the total training time, we limit the number of teachers to be 2. Thus, we initialize from an off-the-shelf ('OTS') student, and sequentially distill it with 2 teachers, in total 24 epochs (equivalent to a $2\times$ training schedule). We also compare with student models trained with a longer $3\times$ training schedule. For the Mask R-CNN student, we should no longer use Teacher I (the student model itself) or Teacher II (the single-stage teacher does not outperform the student by a large margin). To compensate for that, we include Teacher V, which can be considered as a hybrid model of DetectoRS backbone/neck and Mask R-CNN head. Thus, the teacher pool for Mask R-CNN includes Teacher III-V. More architectural details are listed in Table 5.1.

Following Section 5.4.1, we use Algorithm 2 to determine the sequence of teachers to use for each student. For the RetinaNet student, our algorithm suggests teacher sequence III$\rightarrow$IV. For the Mask R-CNN student, our algorithm suggests teacher sequence V$\rightarrow$IV. Table 5.3 shows the results on COCO. Additional results, analysis, and ablation studies of Mask R-CNN distillation can be found in the supplementary materials.

Overall performance: Our distilled student models (row 3&10) significantly improves over the 'OTS' students (row 1&8). The box AP of RetinaNet is improved from 36.5% to 39.9% ($+3.4\%$). The box AP of Mask R-CNN is improved from 38.2% to 41.4% ($+3.2\%$) and the mask AP of Mask R-CNN is improved from 34.7% to 37.3% ($+2.6\%$). After progressive distillation, our resulting Mask R-CNN detector has *comparable performance with HTC teacher, but much less runtime* (51ms vs. 181ms).

Comparison with baselines: First, the performance gain is not merely from a longer training schedule. Our distilled student models (row 3&10) consistently outperform original students trained with a $3\times$ schedule (row 2&9). Moreover, our approach itself has outperformed previous methods in row 4, 5, 6, 11. This comparison shows that a very simple distillation loss can already outperform all previous complicated designs, if a sequence of teachers is properly used. Meanwhile, our training pipeline is more efficient than other distillation-based methods. For example, with the same student and teacher pair and the same number of epochs, our method requires about 20% less training time than [248] because we directly distill feature maps without computing attention and non-local modules. It is worth noting that our detection performance for large objects receives the most gain (about 6% AP_L improvement for both models). The reason why we emphasize AP_L is that, in an efficiency-centric real-world application (e.g. autonomous driving, robot navigation), detecting nearby larger objects is more critical than others. From a realistic perspective, better AP_L shows better applicability of our approach. When further augmenting an advanced distillation mechanism [248] with sequential distillation (row 7&12), we achieve the best stu-

ID	Model	Method	Box						Mask					
			AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
1	RetinaNet (Student I)	OTS	36.5	55.4	39.1	20.4	40.3	48.1	-	-	-	-	-	-
2		Longer $3\times$ training schedule	39.5	58.8	42.2	**23.8**	43.2	50.3	-	-	-	-	-	-
3		Ours, distilled by Teachers III→IV	**39.9**	**59.2**	**42.7**	21.7	**43.3**	**54.1**	-	-	-	-	-	-
4		Guo *et al.*, CVPR 2021 [78]	39.7	**59.6**	42.9	**23.4**	**43.6**	52.9	-	-	-	-	-	-
5		Dai *et al.*, CVPR 2021 [46]	39.1	59.0	42.3	22.8	43.1	52.3	-	-	-	-	-	-
6		Zhang *et al.*, ICLR 2021 [248]	39.6	58.8	42.1	22.7	43.3	52.5	-	-	-	-	-	-
7		Our sequential distillation + [248]'s loss	**40.2**	59.4	**43.1**	21.7	43.5	**55.6**	-	-	-	-	-	-
8	Mask R-CNN (Student II)	OTS	38.2	58.8	41.4	21.9	40.9	49.5	34.7	55.7	37.2	18.3	37.4	47.2
9		Longer $3\times$ training schedule	40.9	61.3	44.8	**24.4**	44.6	52.3	37.1	58.3	**39.9**	18.4	39.8	51.9
10		Ours, distilled by Teachers V→IV	**41.4**	**61.9**	**45.1**	23.3	**45.0**	**55.4**	**37.3**	**58.8**	39.8	**19.4**	**40.4**	**52.1**
11		Zhang *et al.*, ICLR 2021 [248] †	41.3	61.9	45.1	**23.3**	**44.8**	55.3	37.2	58.6	40.1	17.3	40.0	55.1
12		Our sequential distillation + [248]'s loss	**41.6**	**62.0**	**45.4**	**23.3**	44.7	**55.9**	**37.4**	**59.0**	**40.4**	**17.5**	**40.0**	**56.2**

† Reproduced using the code by the authors to acquire complete results including AP_{50}, AP_{75}.

Table 5.3: Homogeneous distillation of COCO detectors, where students with ResNet-50 backbones are distilled with teachers with ResNet-50 backbones. We report the detection ('Box') and segmentation ('Mask') APs, and we compare our distilled student with off-the-shelf ('OTS') student, longer trained student, and the state-of-the-art distillation baselines. Our distilled student significantly improves the detection AP over the 'OTS' student by $\mathbf{3}.4\%$ for RetinaNet and $\mathbf{3}.2\%$ for Mask R-CNN, and outperforms the baselines. A combination of our sequential distillation with an advanced distillation loss design can lead to even further improvement.

dent performance. This fact reveals more potential in detector distillation: In parallel to developing better distillation mechanisms, a teacher curriculum sequence may further boost the student performance *for free*.

5.4.3 Distillation with Heterogeneous Teachers

We now consider a more challenging heterogeneous scenario, where students and teachers have different backbones or input resolutions. Specifically, Student III, a ResNet-18 Mask R-CNN, is distilled with ResNet-50 teachers; Student IV, a model with reduced input resolution, is distilled with teachers trained with larger input resolutions. The results are summarized in Table 5.4, and additional results and ablation studies are included in the supplementary material.

Heterogeneous backbones: Student III has a ResNet-18 backbone and about half runtime as its ResNet-50 counterpart (Teacher I). We find the proper distillation scheme for Student III is to use the sequence of Teacher I→V→IV, which significantly improves Student III over the 'OTS' model. The box AP of Student III is improved from 33.3% to 37.0% $(+3.7\%)$, and especially for large objects, AP_L is improved from 43.6% to 50.0% $(+6.4\%)$.

ID	Model	Backbone	Resolution	AP		Runtime
				Box	Mask	(ms)
1	Teacher I	R50	$1\times$	38.2	34.7	51
2	Teacher IV	R50	$1\times$	49.1	42.6	223
3	Teacher V	R50	$1\times$	45.1	40.1	142
4	Student III, OTS	R18	$1\times$	33.3	30.5	29
5	Student III, Our distilled	R18	$1\times$	**37.0**	**33.7**	29
6	Student IV, OTS	R50	$0.25\times$	25.8	23.0	17
7	Student IV, Our distilled	R50	$0.25\times$	**31.5**	**28.2**	17

Table 5.4: Heterogeneous distillation of COCO detectors, where students with smaller backbones (ResNet-18 vs. ResNet-50) or input resolutions (333×200 vs. 1333×800) are distilled with heterogeneous teachers, requiring additional transfer logic (Sec. 5.3.2). We report the detection ('Box'), segmentation ('Mask') APs and runtime, and compare our distilled student with its teachers and off-the-shelf ('OTS') student. Our distilled students significantly improves the APs over the 'OTS' students by over 3%.

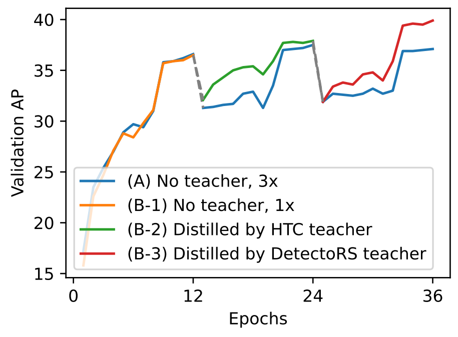
Heterogeneous input resolutions: Although inputs with varying resolutions can be fed into most object detectors without changing the architecture, the performance often degenerates when there is a resolution mismatch between training and evaluation [129, 205]. If ultimately we want to apply a detector to low-resolution inputs for fast inference, it is better to use low-resolution inputs during training. On the other hand, we conjecture that teachers with high-resolution inputs may provide finer details that can assist the student. With our progressive distillation approach, we investigate the improvement of a low-resolution student distilled by a sequence of teachers with high-resolution inputs. We denote the standard input resolution 1333×800 as $1\times$, and a reduced resolution 333×200 as $0.25\times$. We distill Student IV (with $0.25\times$ resolution) by a sequence of Teacher I variants ($0.5\times \rightarrow 0.75\times \rightarrow 1\times$). From Table 5.4, we can see substantial improvement brought by progressive knowledge distillation: the box AP is improved from 25.8% to 31.5% ($+5.7\%$) and the mask AP is improved from 23.0% to 28.2% ($+5.2\%$).

5.4.4 Unpacking the Performance Gain: Generalization or Optimization?

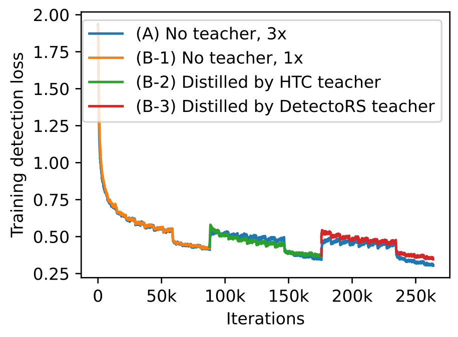
We have shown that our distilled student significantly improves the final accuracy on the *validation* data over the off-the-shelf student. As further demonstrated in Figure 5.4a, the detection validation accuracy of the distilled student is gradually increasing during the distillation process, and achieving a higher value compared with the student trained without teachers. A natural question then arises – why is distillation helping? There are two possible hypotheses: (1) *improved optimization:* distillation facilitates the optimization procedure, leading to a local minimum with a lower loss, and (2) *improved generalization:* the distillation process helps the student generalize to unseen data.

Improved optimization is typically manifested through a better model, a lower training loss and a higher validation accuracy, which is exactly the case for Mask R-CNN, HTC and DetectoRS. As a consequence, one might think that distillation works in the same way. However, our investigation suggests the opposite — our progressive distillation increases both the validation accuracy and the training loss, and therefore effectively reduces the generalization gap. In Figure 5.4, we compare the original RetinaNet model and the distilled student, which have the same architecture, the same latency and are trained on the same data, but with different supervision (only ground-truth labels vs. additional knowledge distillation). To eliminate the influence of learning rate changes, we train the original student with a $3\times$ schedule and restart the learning rate at the same time with the distilled student. Interestingly, although distillation can improve the student's validation performance, the *training* detection loss of the distilled student is higher than the original student. This suggests that distillation does *not* help the optimization process to find a local minimum with a lower training loss, but rather strengthen the generalizability of the student model.

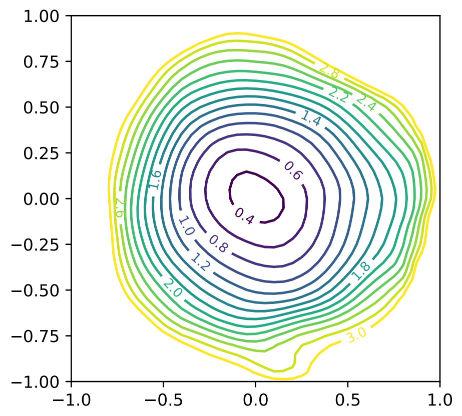
To further support this observation, we also visualize the local loss landscape, following the technique proposed by Li *et al.* [127]. The distilled student has a flatter loss landscape (Figure 5.4d) compared to the original one (Figure 5.4c). As widely believed in the machine learning literature, flat minima lead to better generalization [89, 106]. The observation shown in Figure 5.4 is illustrated for RetinaNet, but we also have similar observation in other student/teacher pairs. As a conclusion, knowledge distillation, which enforces the student to mimic the teachers' features, can be considered as an implicit regularization, and helps the student combat overfitting and achieve better generalization.



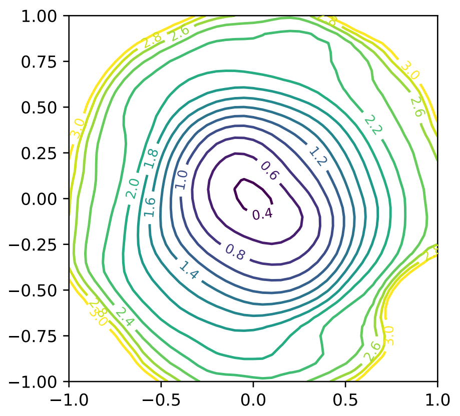
(a)



(b)



(c)



(d)

Figure 5.4: Comparisons of student models trained with and without teachers. We train a ResNet-50 backboned RetinaNet (Student I) with: (A) a prolonged $3\times$ training schedule (curves in blue); (B) progressive knowledge distillation from HTC (Teacher III) and then DetectoRS (Teacher IV) (curves in orange-green-red). We compare the validation AP (Figure 5.4a) and the training detection loss L_{detect} (Figure 5.4b) of the two students during the training process. Despite its worse training loss, the distilled student can generalize better on the validation set. We also compare the loss landscapes [127] of the original student (Figure 5.4c) and the distilled student (Figure 5.4d). Distillation can guide the student to converge to a flatter local minimum. These observations suggest distillation helps generalization rather than optimization.

5.5 Conclusion

We present a simple sequential approach to knowledge distillation, which progressively transfers the knowledge of a sequence of high-capacity teachers to learn a lightweight object detection and instance segmentation model. Our approach leverages modular structures to align student and teacher features and arranges multiple teachers into a curriculum, thus effectively mitigating the representation gap between the teacher and student. Extensive experiments demonstrate our state-of-the-art accuracy-latency trade-off on the challenging COCO dataset. We also conduct analysis to examine why distillation helps given the same model and dataset and find that distillation, via the implicit regularization imposed by teachers' supervision, improves generalization rather than optimization.

5.A Appendix

5.A.1 More Results on Searching for the Near-Optimal Teacher Order

In this section, we show more detailed results about searching a proper teacher order for progressive knowledge distillation, and validate the approach we propose in the main paper. As described in Section 3.3, we first quantify the adaptation cost $\mathcal{C}(\cdot,\cdot)$ between every pair of models in our pool, and then use a heuristic method (Algorithm 1) to construct a sequence of teachers. We have shown that the teacher order suggested by our algorithm is highly competitive in Table 2. One might think there should be better choices than a greedy algorithm on a directed graph, such as a shortest-path algorithm. To validate our algorithm design, we compare our Algorithm 1 against several other algorithms.

To begin with, we include the detailed adaptation costs $\mathcal{C}(\cdot,\cdot)$ among RetinaNet (Student I) and its teachers (Teacher I-IV) in Table 5.5. As described in Section 4.1, we have distilled Student I with *all* possible teacher orders in the pool, using a reduced training budget of 3 epochs for each teacher. The results of these mini-budget distillation are summarized in Table 5.6.

Table 5.5: Adaptation costs among Student I (RetinaNet) and Teacher I-IV (Mask R-CNN, FCOS, HTC, DetectoRS). The adaptation cost is computed pair-wise as described in Section 3.3 of the main paper. Using this metric we can construct a directed graph, as illustrated in Figure 3.

From \ To	Student I	Teacher I	Teacher II	Teacher III	Teacher IV
Student I	-	0.939	0.060	1.568	1.254
Teacher I	0.183	-	0.070	0.934	0.963
Teacher II	0.339	1.181	-	1.940	1.401
Teacher III	0.191	0.484	0.082	-	0.890
Teacher IV	0.232	0.767	0.077	1.248	-

Given the adaptation costs in Table 5.5, we can construct a directed graph, part of which has been illustrated in Figure 3. On the directed graph, we can run several algorithms to select a path. Besides our Algorithm 1, one may also propose these algorithms:

- **Shortest-path (sum)**: Set the student as the source node, and set the best

Table 5.6: Performance of Student I (RetinaNet) distilled with different teacher sequences, under reduced training budgets. For each teacher in the sequence, the student is trained for 3 epochs on COCO. After progressive knowledge distillation, the student is evaluated on the COCO validation set. The teacher orders suggested by Algorithm 1 are marked **bold**.

Length	Teacher Sequence	Student AP	Length	Teacher Sequence	Student AP	Length	Teacher Sequence	Student AP
1	III	36.8	3	III→II→IV	38.0	4	III→II→I→IV	38.2
	IV	**36.7**		**I→III→IV**	**37.9**		III→I→II→IV	38.1
	I	36.4		III→IV→II	37.9		I→III→II→IV	38.1
	II	36.2		II→III→IV	37.9		II→III→I→IV	38.0
				III→I→IV	37.8		I→III→IV→II	38.0
2	**III→IV**	**37.6**		I→II→IV	37.7		II→I→III→IV	37.9
	IV→II	37.3		I→IV→II	37.6		III→I→IV→II	37.9
	III→II	37.3		IV→II→III	37.5		I→II→III→IV	37.9
	I→IV	37.3		IV→III→II	37.5		IV→I→III→II	37.7
	IV→III	37.2		II→I→IV	37.5		II→I→IV→III	37.7
	I→III	37.1		I→III→II	37.5		I→II→IV→III	37.7
	IV→I	37.0		IV→I→III	37.4		IV→III→I→II	37.6
	II→IV	37.0		II→IV→III	37.4		IV→I→II→III	37.6
	III→I	37.0		III→IV→I	37.4		III→IV→II→I	37.6
	II→I	36.9		III→I→II	37.4		III→IV→I→II	37.6
	II→III	36.8		I→IV→III	37.4		III→II→IV→I	37.6
	I→II	36.8		IV→II→I	37.3		I→IV→III→II	37.6
				IV→III→I	37.3		IV→II→I→III	37.5
				IV→I→II	37.3		IV→III→II→I	37.5
				I→II→III	37.3		II→IV→I→III	37.5
				II→IV→I	37.2		II→III→IV→I	37.5
				II→I→III	37.2		I→IV→II→III	37.5
				III→II→I	37.2		II→IV→III→I	37.4
				II→III→I	37.1		IV→II→III→I	37.3

Table 5.7: Comparison of four algorithms for teacher order selection, in the mini-budget distillation setting. Our Algorithm 1 can consistently produce a better teacher order than other algorithms.

k	Algorithm	Suggested teacher order	Student AP	Ranking in all orders	k	Algorithm	Suggested teacher order	Student AP	Ranking in all orders
1	Shortest-path (sum)	IV	36.7	2 / 4	3	Shortest-path (sum)	II→I→IV	37.5	9 / 40
	Shortest-path (max)	IV	36.7	2 / 4		Shortest-path (max)	I→III→IV	37.9	2 / 40
	Forward construction	II	36.2	4 / 4		Forward construction	II→I→III	37.2	25 / 40
	Our Algorithm 1	IV	36.7	2 / 4		Our Algorithm 1	I→III→IV	37.9	2 / 40
2	Shortest-path (sum)	II→IV	37.0	7 / 16	4	Shortest-path (sum)	II→I→III→IV	37.9	7 / 64
	Shortest-path (max)	I→IV	37.3	2 / 16		Shortest-path (max)	II→I→III→IV	37.9	7 / 64
	Forward construction	II→I	36.9	10 / 16		Forward construction	II→I→III→IV	37.9	7 / 64
	Our Algorithm 1	III→IV	37.6	1 / 16		Our Algorithm 1	I→III→IV	37.9	7 / 64

performing teacher as the target node T_{λ_k}. Find a path $S \to T_{\lambda_1} \to \cdots \to$

T_{λ_k} that minimizes the *sum* of adaptation costs along the path: $\min_{T_{\lambda_1},\dots,T_{\lambda_{k-1}}} \mathcal{C}(S, T_{\lambda_1}) + \sum_{j=1}^{k-1} \mathcal{C}(T_{\lambda_j}, T_{\lambda_{j+1}})$.

- **Shortest-path (max)**: Set the student as the source node, and set the best performing teacher as the target node T_{λ_k}. Find a path $S \to T_{\lambda_1} \to \cdots \to T_{\lambda_k}$ that minimizes the *maximum* of adaptation costs along the path: $\min_{T_{\lambda_1},\dots,T_{\lambda_{k-1}}} \max\{\mathcal{C}(S, T_{\lambda_1}), \mathcal{C}(T_{\lambda_1}, T_{\lambda_2}), \dots, \mathcal{C}(T_{\lambda_{k-1}}, T_{\lambda_k})\}$.
- **Forward construction**: Contrary to Algorithm 1, we may start from the student and choose the nearest teacher from the current one, to construct the sequence: $T_{\lambda_1} \leftarrow \arg\min_{T_u \in \mathcal{P}} \mathcal{C}(S, T_u), T_{\lambda_{j+1}} \leftarrow \arg\min_{T_u \in \mathcal{P}} \mathcal{C}(T_{\lambda_j}, T_u)$.

The output teacher sequences and corresponding student performance of these three algorithms are summarized in Table 5.7. In this setting, our Algorithm 1 can consistently produce a competitive teacher order that leads to a good performance of the distilled student. Compared to our Algorithm 1, shortest-path (max) can achieve a similar performance, and it is only worse than ours when $k = 2$. Forward construction performs worst among the four algorithms.

In summary, a greedy backward construction like Algorithm 1 works the best in our setting, rather than globally optimized shortest-path algorithms. The final target teacher has the most profound impact on the distilled student's performance. In order to fully assist the final teacher, we need to use another teacher with the minimal adaptation cost to the final teacher before it, which is exactly the behavior of Algorithm 1.

5.A.2 Ablation Study on Distillation with Homogeneous Teachers

In this section, we provide more details about distillation with homogeneous teachers (Section 4.2). We investigate (1) the impact of each individual teacher; and (2) distillation with teachers simultaneously vs. sequentially.

Impact of individual teachers: We first distill Student II with each of the three teachers individually: Teacher III has the same backbone and neck but a more advanced head; Teacher IV has more advanced backbone, neck, and head; Teacher V has the same head but more advanced backbone and neck. Table 5.8 provides the performance of the three teachers, where Teacher IV achieves the best performance (row 1-3). From Table 5.9, we can see that our distilled students (row 2-7) *significantly and consistently* outperform the off-the-shelf student (row 1), demonstrating the effectiveness of our distillation strategy *irrespective of the types of teachers*. Moreover, the improvement of

Table 5.8: Homogeneous distillation of COCO detectors, where students with ResNet-50 backbones are distilled with teachers with ResNet-50 backbones. We report the detection ('Box') and segmentation ('Mask') APs and runtime, and we compare our distilled student with its teachers, off-the-shelf ('OTS') student. Our distilled student significantly improves the APs over the 'OTS' student by around 3%.

ID	Model	Box						Mask						Runtime
		AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	(ms)
1	Teacher III	42.3	61.1	45.8	23.7	45.6	56.3	37.4	58.4	40.2	19.6	40.4	51.7	181
2	Teacher IV	49.1	67.7	53.4	29.9	53.0	65.2	42.6	65.1	46.0	24.1	46.4	58.6	223
3	Teacher V	45.1	66.3	49.3	27.8	49.0	59.3	40.1	63.1	42.8	22.9	43.8	54.8	142
4	Student II (OTS)	38.2	58.8	41.4	21.9	40.9	49.5	34.7	55.7	37.2	18.3	37.4	47.2	51
5	Student II (distilled)	**41.4**	**61.9**	**45.1**	**23.3**	**45.0**	**55.4**	**37.3**	**58.8**	**39.8**	**19.4**	**40.4**	**52.1**	49

Table 5.9: Ablation study of homogeneous distillation of COCO detectors (models in Table 5.8). Our distillation strategy is *consistently effective irrespective of teacher type*. Moreover, sequential distillation with two teachers outperforms both distillation with a single teacher and simultaneous distillation with two teachers. Our best distilled student is obtained by *progressive* distillation, where Student II is first distilled with Teacher V (a weaker, more similar teacher with the same head as Student II) and then distilled with Teacher IV (a stronger teacher whose architecture is completely different from Student II).

ID	Student II	Box						Mask					
		AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
1	OTS	38.2	58.8	41.4	21.9	40.9	49.5	34.7	55.7	37.2	18.3	37.4	47.2
2	Distilled by Teacher III	40.2	60.7	43.8	22.5	43.8	53.4	36.3	57.3	38.7	18.9	39.3	50.3
3	Distilled by Teacher IV	40.8	61.5	44.6	23.0	44.3	54.2	36.8	58.3	39.4	19.2	39.9	51.0
4	Distilled by Teacher V	40.8	61.4	44.5	22.9	44.3	54.2	36.6	58.1	39.1	19.2	39.6	51.0
5	Distilled by Teachers IV+V	39.8	60.3	43.4	22.1	43.3	52.9	35.9	57.1	38.1	18.3	39.0	49.8
6	Distilled by Teachers IV$\rightarrow$V	41.0	61.7	44. 8	23.0	44.3	54.9	36.8	58.3	39.2	19.5	39.9	51.3
7	Distilled by Teachers V$\rightarrow$IV	**41.4**	**61.9**	**45.1**	**23.3**	**45.0**	**55.4**	**37.3**	**58.8**	**39.8**	**19.4**	**40.4**	**52.1**

the student distilled with Teacher V (row 2) over that with Teacher III (row 3) shows that a more powerful teacher generally leads to a better distilled student. Interestingly, although Teacher IV is more powerful than Teacher V, Table 5.9 shows that their distilled students achieve quite similar AP (row 2 vs. row 4). This indicates that an even more powerful teacher does not necessarily further improve the distilled student; too large a capacity and structure gap between the teacher and student will limit the effectiveness of distillation. Also, it is easier to distill from teachers with the same head.

Simultaneous vs. progressive distillation: We now distill Student II with the combined teachers, and we choose the top-performing Teacher IV and Teacher V. We investigate two types of combination – simultaneous distillation with a feature matching loss between each teacher and the student (row 5), and sequential distillation with teachers one by one (row 6-7). First, we find that using both teachers simultaneously (row 5) is *even worse* than our method using a single teacher (row 2-4). This shows that integrating different types of knowledge from multiple teachers is not a trivial task – simultaneously using the features from multiple teachers might provide *conflicting supervisions* to the student model and thus hinder its distillation process. By contrast, our sequential distillation overcomes this issue and improves the performance *irrespective of the order of the teachers* (row 6-7 vs. row 1-4). Second, the sequential order of the teachers makes a difference. A *curriculum-like progression* (row 7), where the teacher with a smaller adaptation cost is used first and that with a larger adaptation cost & a higher performance is used later, leads to the best performance.
Overall performance: Our best distillation performance is achieved when we first distill Student II with a curriculum of teachers (Teacher V$\rightarrow$IV). Overall, the box AP is improved from 38.2% to 41.4% and the mask AP is improved from 34.7% to 37.3%. Our resulting Mask R-CNN detector has *comparable performance with HTC, but much smaller runtime.*

5.A.3 Ablation Study on Distillation with Heterogeneous Teachers

In this section, we provide more details about distillation with heterogeneous teachers (Section 4.3). We investigate the heterogeneous cases where the backbones or input resolutions are different between the teachers and student.
Overall performance: Again, Tables 5.10 and 5.11 show that our distillation strategy is consistently effective with respect to all the teachers and their combinations, *e.g.*, the box AP improves from 33.3% to 37.0% and the mask AP improves from 30.5% to 33.7%.
Two signature findings in heterogeneous distillation: Compared to the homogeneous case, we find the capacity gap between models is a more important factor, and to bridge this gap a proper teacher order plays a more critical role. Details are explained as follows.
The student-teacher capacity gap is more pronounced in heterogeneous distillation. Among the four teachers, Teacher I shares exactly the same neck and head

Table 5.10: Heterogenous distillation of COCO detectors, where students with ResNet-18 backbones are distilled with teachers with ResNet-50 backbone, requiring additional transfer logic (Section 3.2). We report the detection ('Box') and segmentation ('Mask') APs and runtime, and we compare our distilled student with its teachers, and off-the-shelf ('OTS') student. Our distilled student significantly improves the APs over the 'OTS' student by over 3%.

ID	Model	Box						Mask						Runtime
		AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	(ms)
1	Teacher I	38.2	58.8	41.4	21.9	40.9	49.5	34.7	55.7	37.2	18.3	37.4	47.2	51
2	Teacher III	42.3	61.1	45.8	23.7	45.6	56.3	37.4	58.4	40.2	19.6	40.4	51.7	181
3	Teacher IV	49.1	67.7	53.4	29.9	53.0	65.2	42.6	65.1	46.0	24.1	46.4	58.6	223
4	Teacher V	45.1	66.3	49.3	27.8	49.0	59.3	40.1	63.1	42.8	22.9	43.8	54.8	142
5	Student III (OTS)	33.3	52.9	35.9	18.2	35.9	43.6	30.5	50.0	32.1	15.5	32.9	41.8	29
6	Student III (Distilled)	**37.0**	**56.8**	**39.9**	**20.2**	**39.8**	**50.0**	**33.7**	**53.6**	**36.0**	**17.2**	**36.0**	**47.3**	29

structure with the student, and has a similar but larger backbone; Teacher V has the same head with the student as well, but has a different backbone and neck; Teacher III has similar backbone and neck, but has a different head; and Teacher IV is the most powerful one with completely different architecture. Table 5.11 (rows 3-6) summarizes the distillation results with single teachers. First, directly distilling from the strongest teacher (Teacher IV) does not yield the largest improvement. Second, a relatively less powerful but more similar teacher (Teacher I) leads to the best distillation performance, improving the APs by 2%, although teachers V, III, and IV are all stronger than Teacher I. One possible reason is that Teacher I has the same neck and head as Student III as well as similar but deeper backbone, so the capacity gap between Student III and Teacher I is the smallest. Finally, we find that Teacher III is a strong but not particularly helpful teacher, achieving the worst distillation results. One possible reason is that Teacher III has a very different head from Student III, while not as stand-alone accurate as Teacher IV, making it unable to provide enough guidance to Student III. These observations suggest that a smaller capacity gap between the student and the teacher may facilities knowledge transfer.

The sequential order of the teachers plays a more critical role in the heterogeneous setting. Table 5.11 (row 7-12) presents representative results with different orders or combinations of the teachers. Again, a proper progressive distillation (row 12) outperforms simultaneous distillation (row 7-9). Notably, it is necessary to start with Teacher I, since the capacity gap between Student III and Teacher I is minimal, with difference only on the depth of their ResNet backbones. These results confirm the importance of our curriculum-

Table 5.11: Ablation study for heterogeneous COCO detector distillation (models in Table 5.10). Student III (Mask R-CNN with a ResNet-18 backbone) is distilled with teachers with *different* and larger ResNet-50 backbones. Training Student III for more epochs improves its performance, but not as much as progressive distillation with teachers. Note that for each distillation we train 12 epochs. Our distillation strategy is *consistently effective irrespective of the types of teachers*. Moreover, our sequential distillation with multiple teachers outperforms simultaneous distillation with multiple teachers. Our best distilled student is obtained by *progressive* distillation, where Student III is first distilled with Teacher I (a most similar teacher with the same head and neck as Student III and a deeper backbone), then distilled with Teacher V (a stronger teacher with the same head as Student III), and finally distilled with Teacher IV (a strongest teacher whose architecture is completely different from Student III).

ID	Model	Box						Mask					
		AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
1	Student III (OTS)	33.3	52.9	35.9	18.2	35.9	43.6	30.5	50.0	32.1	15.5	32.9	41.8
2	+12 epochs	34.6	54.5	37.2	18.8	36.9	46.1	31.6	51.5	33.6	15.8	33.7	44.0
3	+24 epochs	34.5	54.2	37.2	18.8	36.5	45.8	31.5	51.2	33.8	16.0	33.4	43.7
4	+36 epochs	34.6	54.2	37.4	18.6	36.9	46.7	31.6	51.1	33.8	15.7	33.6	44.3
3	Distilled by Teacher I	35.8	55.8	38.8	19.3	38.8	47.9	32.6	52.7	34.8	16.0	35.3	45.5
4	Distilled by Teacher III	35.2	55.2	37.8	19.1	37.8	47.4	32.1	52.0	34.0	16.1	34.5	45.2
5	Distilled by Teacher IV	35.5	55.2	38.2	19.0	37.9	48.0	32.4	51.9	34.5	15.9	34.8	45.6
6	Distilled by Teacher V	35.4	55.2	38.3	19.4	37.9	48.4	32.2	52.2	34.3	15.4	34.4	45.8
7	Distilled by Teachers IV+V	34.8	54.9	37.2	19.0	37.2	47.0	31.6	51.7	33.9	15.7	33.8	44.2
8	Distilled by Teachers I+IV+V	36.0	55.4	39.1	18.2	38.1	48.3	32.1	53.0	34.7	15.8	34.7	46.1
9	Distilled by Teachers I+III+IV+*V*	36.1	55.2	39.0	18.4	38.2	48.0	31.7	52.9	34.3	15.1	34.2	46.3
10	Distilled by Teachers I→V	36.5	56.3	39.3	19.5	38.8	49.4	33.2	53.2	35.3	16.4	35.4	46.8
11	Distilled by Teachers V→IV	35.2	55.2	37.8	19.1	37.8	47.4	32.1	52.0	34.0	16.1	34.5	45.2
12	Distilled by Teachers I→V→IV	**37.0**	**56.8**	**39.9**	**20.2**	**39.8**	**50.0**	**33.7**	**53.6**	**36.0**	**17.2**	**36.0**	**47.3**

like progression to best benefit from multiple teachers.

Training a student longer vs. distilling a student: As another sanity check, Table 5.11 includes results of training Student III with more epochs without distillation (row 2-4). We can see that the first 12 additional epochs improve APs by 1%, but there are no significant improvements even if we train for a longer period. This shows the effectiveness of detector distillation.

Distillation with different model resolutions: In Table 5.11, we have performed distillation where the student and teacher models operated on the same input image resolution (*e.g.*, the standard resolution $1,333 \times 800$ on MS COCO). In practice, one way to further reduce the latency/runtime of the student is to operate on lower-resolution images. However, this poses additional challenges – with a teacher of high input resolution and a student of low input resolution, they become even more heterogeneous. Moreover, image resolution substantially affects object detection performance [4]. Here, we are interested in performing distillation with models trained with images of different resolutions to further investigate the generalizability of our approach. More specifically, we use high-resolution models as teachers and low-resolution models as students, as shown in Table 5.12 (row 1-4).

Table 5.12: Detectors trained with different input resolutions on the COCO dataset. We use a series of Teacher I variants: Teacher I-1 is trained with the standard input resolution of $1,333 \times 800$; Teacher I-2 is trained with $1,000 \times 600$ input; Teacher I-3 is trained with 666×400 input; and the student is trained with 333×200 input. We report the detection ('Box') and segmentation ('Mask') APs and runtime. We compare our distilled student with its teachers, and off-the-shelf ('OTS') student. Our approach is *effective with even more heterogeneous teacher and student models of different input resolutions*.

ID	Model	Input	Box						Mask						Runtime
		Resolution	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	(ms)
1	Teacher I-1	1333×800	38.2	58.8	41.4	21.9	40.9	49.5	34.7	55.7	37.2	18.3	37.4	47.2	31.5
2	Teacher I-2	1000×600	37.2	57.7	40.5	19.1	40.9	50.4	33.6	54.3	35.9	15.6	37.0	47.7	24.9
3	Teacher I-3	666×400	34.7	54.0	37.2	15.6	38.1	50.4	31.2	50.5	33.2	12.2	34.4	47.0	19.7
4	Student (OTS)	333×200	25.8	41.9	27.1	7.0	27.8	44.3	23.0	38.7	23.7	5.0	23.7	41.3	16.9
5	Student (distilled)	333×200	**31.5**	**49.8**	**33.3**	**12.3**	**34.3**	**48.9**	**28.2**	**46.5**	**29.0**	**9.3**	**30.3**	**45.4**	16.9

In these experiments, the teacher and student feature maps have different *spatial resolution*. To tackle this, we simply upsample the spatial maps of the student and supervise the student with the teachers' features. Again, Table 5.12 shows that our approach is effective in this more challenging scenario. Our best performance is achieved by progressively distilling the student with its Teacher I-3, I-2, and I-1.

5.A.4 Generalizability: Combination with Other Distillation Methods

Our main contribution is *a general and flexible distillation framework* – progressive knowledge distillation via a sequence of teachers. In principle, this framework is *agnostic* to specific distillation methods and loss designs when distilling the student with a specific teacher during the intermediate stage. In the main paper, we showed that under this progressive distillation framework, a simple distillation method that matches the feature maps from the neck module without any modification or augmentation has already achieved the state-of-the-art performance. A natural question then arises – if using a more sophisticated distillation method, will the student performance further improve? Our investigation in this section shows that this is indeed the case.

Table 5.13: Our progressive knowledge distillation via a sequence of teachers is a general and flexible distillation framework: a recent, powerful detector distillation method by Zhang *et al.* [248] can be incorporated into our framework to further improve the distilled student performance. By replacing our original distillation method (which simply matches feature maps) with the more sophisticated distillation method from [248] and by using our proposed progressive teacher sequence (V→IV), the student can achieve the best performance. From a different perspective, our progressive framework enables the use of a simple feature matching distillation method, which to some extent diminishes the benefit of designing more sophisticated distillation methods. For fair comparison, the total distillation training epochs are set to 24. If two teachers are used, the student is distilled with each teacher for 12 epochs.

ID	Model	Distillation Loss	Teacher(s)	Box AP	Box AP_{50}	Box AP_{75}	Box AP_S	Box AP_M	Box AP_L	Mask AP	Mask AP_{50}	Mask AP_{75}	Mask AP_S	Mask AP_M	Mask AP_L
1	Teacher IV	-	-	49.1	67.7	53.4	29.9	53.0	65.2	42.6	65.1	46.0	24.1	46.4	58.6
2	Teacher V	-	-	45.1	66.3	49.3	27.8	49.0	59.3	40.1	63.1	42.8	22.9	43.8	54.8
3	Teacher VI	-	-	47.3	66.3	51.7	28.2	51.7	62.7	41.1	63.5	44.4	22.9	44.9	56.3
4	Student II (OTS)	-	-	38.2	58.8	41.4	21.9	40.9	49.5	34.7	55.7	37.2	18.3	37.4	47.2
5		[248]	VI	41.3	61.9	45.1	**23.3**	44.8	55.3	37.2	58.6	40.1	17.3	40.0	55.1
6	Student II	Ours	Ours (V→IV)	41.4	61.9	45.1	**23.3**	**45.0**	55.4	37.3	58.8	39.8	**19.4**	**40.4**	52.1
7		[248]	Ours (V→IV)	**41.6**	**62.0**	**45.4**	**23.3**	44.7	**55.9**	**37.4**	**59.0**	**40.4**	17.5	40.0	**56.2**

Table 5.14: Generalizability on Argoverse-HD. On the **left**, we report standard detection accuracy. 'OTS' and distilled students are trained on COCO. We observe 2% AP gains through distillation, even on novel testsets. On the **right**, we report streaming detection accuracy as defined in [129], in the detection-only setting on a Tesla V100 GPU. The second column denotes the optimal input resolution (that maximizes streaming accuracy). First, we discover that a lighter model and full-resolution input is much more helpful than having an accurate but complex model that needs to downsize input resolution. Second, our proposed distillation approach further improves over the lightweight model.

Model		box AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Stud. II	OTS	32.7	52	34.5	14.7	35.8	52.8
	Distilled	**34.4**	**54.2**	**35.9**	**15.0**	**36.8**	**57.7**
Stud. III	OTS	28.9	48.8	30.0	12.8	31.3	49.2
	Distilled	**30.6**	**49.7**	**31.8**	**12.9**	**32.6**	**51.9**

Detector	Input	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Cas. MRCNN50 [129]	0.5×	14.0	26.8	12.2	1.0	9.9	28.8
MRCNN18 (Ours)	1.0×	23.7	44.8	22.6	10.4	23.1	37.8
MRCNN18 (+ Distill)	1.0×	**25.0**	**45.8**	**24.2**	**10.5**	**24.1**	**39.3**

Specifically, we incorporate a recent, powerful detector distillation method by Zhang *et al*. [248] into our progressive distillation framework. This method uses attention and non-local modules to transform the feature maps and guide distillation, and thus the student can focus on foreground objects and learn relation between objects. They only use one teacher model for all two-stage detector students. The teacher is Cascade Mask R-CNN [24] with ResNeXt-101 [233] backbone and deformable convolutions [256], and we denote it as Teacher VI. After adding it to the teacher pool for Student II (Mask R-CNN), the teacher sequence suggested by our heuristic is still Teacher V$\rightarrow$IV. We then use this sequence of teachers to distill the student, and we replace our original feature map matching with that in their method.

We summarize the performance of the teachers and distilled Student II in Table 5.13. We have several interesting observations. First, as expected, when incorporating an elaborate distillation method/loss into our progressive knowledge distillation framework, we can further improve [248] by 0.3% overall AP and 1.1% mask AP Large. Second, our progressive framework enables the use of a simple feature matching distillation method, which to some extent diminishes the benefit of designing more sophisticated distillation methods. Third, while they both improve the student performance, our original approach and the approach combined with [248] seem to "teach" the student slightly different knowledge – for example, our original approach significantly improves the mask AP of small objects from 18.3% to 19.4%, while our approach combined with [248] significantly improves the mask AP of large objects from 47.2% to 56.2%. Also, we point out that [248] may decrease the mask performance for small objects after distillation from 18.3% to 17.3% (contrasting row 4 & 5) and there is a trade-off between mask AP Large and mask AP Small of their approach.

5.A.5 Generalizability to Other Datasets and Evaluation Protocols

In this section, we study the generalizability of our approach. As an extension from the gold-standard COCO benchmark, we evaluate our distilled student (trained on COCO) on another dataset, Argoverse-HD, and with another metric, streaming accuracy, and perform distillation on Argoverse-HD directly.

Argoverse-HD is a more challenging dataset than COCO due to higher resolution images and significantly more small objects. Constructed from the autonomous driving dataset Argoverse 1.1 [29], Argoverse-HD contains RGB

video sequences and dense 2D bounding box annotations (1,260k boxes in total). It consists of 8 object categories, which are a subset of 80 COCO classes and are directly relevant to autonomous driving: person, bicycle, car, motorcycle, bus, truck, traffic light, and stop sign. There are 38k training images and 15k validation images. We report results on the validation images. We test the distilled models trained on COCO on Argoverse-HD *without retraining*. Table 5.14-left shows the generalizability of our approach.
Streaming accuracy is a recently proposed metric that simultaneously evaluates both the accuracy and latency of algorithms in an online real-time setting [129]. The evaluator queries the state of the world at all time instants, forcing algorithms to consider the amount of streaming data that must be ignored while processing the last frame. Following the setup proposed in [129], we evaluate streaming AP in the context of real-time object detection for autonomous vehicles. Table 5.14-right shows our approach outperforming the prior results from [129] by a dramatic margin. We find significant wins by using an exceedingly lightweight network (ResNet-18 based Mask R-CNN) that can process full-resolution images without sacrificing latency. Due to much higher quantities of small objects, high-reslution processing is more effective than deeper network structures. In addition, progressive distillation further improves performance.

Table 5.15: Heterogenous distillation of Argoverse-HD detectors, where a student with ResNet-18 backbone is distilled with teachers with ResNet-50 backbones. We report the detection ('Box') APs and runtime. We compare our distilled student with its teachers, and off-the-shelf ('OTS') student. Our distilled student significantly improves the APs over the 'OTS' student by over 2%. Notably, our distilled student achieves detection accuracy that is *comparable with Teacher A but with only around third of the runtime*.

ID	Model	Backbone	Neck	Method (Head)	Box						Runtime
					AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	(ms)
1	Teacher A	ResNet-50	FPN	Faster R-CNN	29.6	48.2	30.5	16.4	33.1	45.1	79.2
2	Teacher B	ResNet-50	FPN	Cascade	32.3	50.4	35.0	16.4	37.1	47.7	89.0
3	Teacher C	ResNet-50 + SAC	RFP	Faster R-CNN	32.9	51.0	35.5	17.6	33.7	52.9	230.8
4	Teacher D	ResNet-50 + SAC	RFP	Cascade	34.5	52.0	37.7	17.9	37.0	52.8	241.2
5	Student (OTS)	ResNet-18	FPN	Faster R-CNN	27.1	48.1	27.5	14.4	31.2	40.0	29.3
6	Student (distilled)	ResNet-18	FPN	Faster R-CNN	**29.2**	**49**	**30.9**	**15**	**31.7**	**45.6**	29.5

Direct distillation on Argoverse-HD: After testing the distilled model which is trained on COCO, on the Argoverse-HD dataset [129] without re-training, we have shown the generalizability of the already-distilled models. Here we *directly distill* the student model on Argoverse-HD, using Faster R-CNN with a ResNet-18 backbone as the student model. As shown in Table 5.15, we use four teachers with ResNet-50 backbones (row 1-4), including Faster R-CNN [180] (Teacher A), Cascade R-CNN [24] (Teacher B), and DetectoRS [169] (Teachers C & D).

The results are summarized in Table 5.15. Our best distillation performance is achieved when we first distill the student with a similar teacher (Teacher A), and then progressively distill with more powerful teachers (Teachers B, then C, and finally D). Overall, the bbox mAP is improved from 27.1% to 29.2%.

In addition, comparing with Table 5.14-left, the *absolute* performance of the teachers and students in Table 5.15 is lower. This is because here we use weaker teachers and student models (Faster R-CNN for fast experiments) than the models used in Table 5.14-left (Mask R-CNN). However, the *relative* improvement (between the distilled and OTS students) of box AP (2.1%) is larger than that in Table 5.14-left (1.7%), indicating that learning distillation directly on Argoverse-HD further improves the performance.

Implementation Details: Consistent with the previous implementation details, here we implement the detectors using the MMDetection codebase. We train on 8 NVIDIA Tesla V100 Graphics Card for 12 epochs for each distillation. We use the input resolution of $1,920 \times 1,200$, with each GPU hosting 1 image. We use an initial learning rate of 0.02 and a linear learning rate decay. We use SGD and a momentum of 0.9. We use 0.8 as the hyper-parameter λ.

Chapter 6

Multi-Range Pyramids

6.1 Introduction

3D object detection plays a vital role in autonomous navigation. Despite the maturity of methods in the existing literature, most treat detection range as a constant instead of an adjustable hyperparameter [121,236,241]. However, we argue that detection range is application *dependent*. For example, small indoor robots may detect only up to a few meters while autonomous delivery trucks may require hundreds of meters of detection range to meet the safety requirement due to higher traveling speed and longer braking distance.

In this paper, we first study the effect of tuning detection range for the dominant paradigm of bird's-eye view (BEV) based 3D detection. BEV-based detectors operate on a dense 2D BEV feature map whose spatial dimensions are directly determined by the processing range and the voxel size (grid density). Interestingly, the existing literature on *2D* detection has converged on image resolution (and backbone depth) as the handy "knobs" for trading off accuracy-vs-latency [129, 205, 217]. We revisit these questions in the context of 3D, and find somewhat surprisingly that *range* is an even more effective parameter for trading off these quantities. For example, we show that even if the sensor (dataset) includes objects up to 200m, optimal accuracy-vs-latency tradeoffs maybe achieved by artificially limiting the range of the model to 100m, essentially "giving up" on far-field detections and re-allocating the additional compute to smaller (higher resolution) voxels in the near-field. More importantly, this analysis reveals that models can be tuned for particular ranges by adjusting other hyperparameters such as voxel resolution. We denote models optimized for specific ranges as *range experts*.

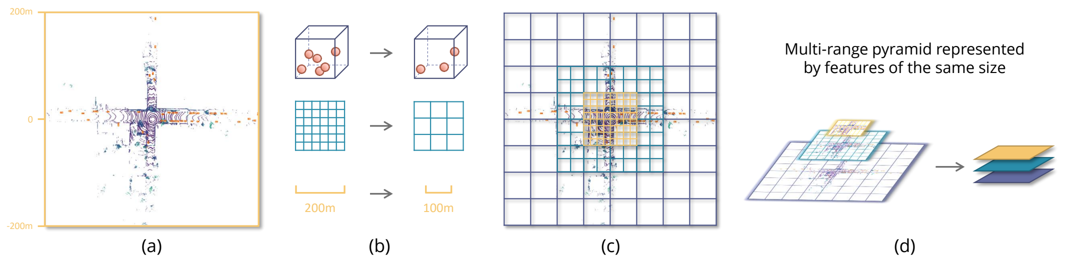

200m
0
-200m
(a)
200m
100m
(b)
(c)
Multi-range pyramid represented by features of the same size
(d)

Figure 6.1: (a) Given a far-range LiDAR input (*e.g.*, 400m×400m), most existing 3D detectors will either run into memory issues or spend hundreds of milleseconds to process the input. (b) We explore several ways to reduce compute: downsample the point cloud, adopt a coarser grid, and limit the processing range. We find that different tradeoffs are optimal for different ranges; near-range voxels can exploit finer cell sizes, while far-range voxels benefit from larger voxels that reduce sparsity while remaining efficient. (c) We propose a novel BEV feature pyramid that allocates more voxel resolution to nearby measurements. (d) Note that each range is voxelized using the same grid density and therefore resulting in feature maps of the same size. This allows us to stack the feature maps together for efficient batch processing and multi-range feature sharing within a single network.

Given a collection of experts tuned for various ranges, one approach for combining their output is simply ensembling their detections; e.g., combine 0-100m detections from the 100m expert with 100-200m detections from the 200m expert. We find such an ensemble greatly boosts detection accuracy. While perhaps unsurprising, such an architecture is performant because it exploits the well-known but under-emphasized property of LiDAR: *farther range implies more sparsity*.

However, naive ensembling is prohibitively expensive, since compute scales linearly with the number of range experts. This leads to the natural question: can we achieve multi-range performance without paying the compute cost? In this paper, we show that one can perform *even better* while remaining efficient by sharing features across range experts via a multi-range feature pyramid (Fig. 6.1). We begin by noting that the natural internal representations of properly-aligned range experts (that differ by ranges in powers of 2) themselves naturally align across adjacent layers of a neural backbone. This allows feature maps across the range experts (with varying grid resolutions and ranges) to be stack-able into a single model for efficient batch processing and multi-range feature sharing. We call this representation a *multi-range pyramid* (MRP), analogous to image feature pyramids common to 2D detection [137].

One might wonder that if long range is the bottleneck, why not adopt other alternative representations, such as range view? 3D BEV models are attractive because they are orthographic and translation-equivariant; one need not worry about perspective distortions arising from far-away objects (that would have a smaller footprint in a range image). As a result, there exist more mature methods for data augmentation [61] and temporal fusion, either at the sensor level [121, 236, 241] or at the feature level [98, 149]. Because of the strong empirical performance of 3D BEV detectors [241], we argue that making them range-efficient will be increasingly important as LiDAR sensors themselves increase in range and density.

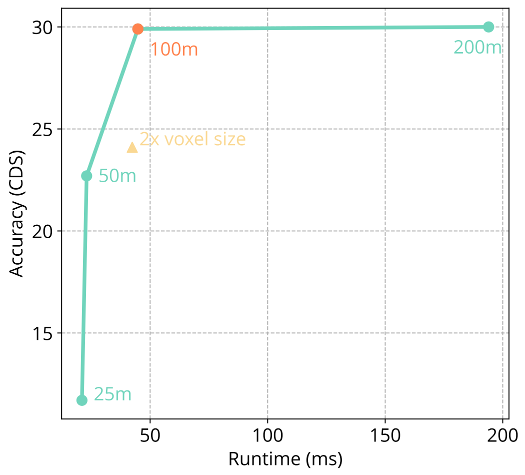


Figure 6.2: To reduce latency, should one increase voxel size or limit range? One can rerun a 100m-trained model (the orange dot) on different ranges (the green curve) by exploiting fully-convolutional processing. Retraining the model with 2X-larger voxels (yellow triangle) does reduce compute, but dramatically reduces accuracy. Additionally, not shown above, we have explored point cloud subsampling, but find this negligibly reduces latency.

To work with MRP in 3D detectors, one needs to address several technical challenges. The first challenge is how to efficiently create such a pyramid. Most BEV feature extractors begin by sparesly computing point features and then scattering them onto a dense BEV grid. We share the initial sparse point cloud features across all ranges by introducing a novel multi-range scattering operation. The next challenge is how to share features across ranges that may or may not be spatially aligned. We address this with two approaches; assuming unaligned features, we first disentangle range processing through group convolution and then share features globally with a squeeze & excitation (SE) layer [93]. If features are aligned, we replace the standard convolution operation in the BEV backbone with a multi-range convolution.

One difficulty with evaluation is the lack of long-range 3D detection datasets. Arguably the most popular 3D detection benchmark is NuScenes [22], but it annotates objects only up to 50m. As such, we validate our approach on

Argoverse [29], which includes annotations for objects up to 200m. We plan to evaluate on other long range datasets as they are released. Our evaluations compare to strong baselines for efficient 3D detection, such as PointPillars [121].

We summarize our contributions as follows:

1. We study the impact of range as a tunable parameter for 3D object detection. We draw analogy to image resolution and find the surprising conclusion that sometimes, the best solution is to limit the detection range.

2. We propose a novel multiple-range representation for BEV feature maps. We show how to build a full detector based on this novel representation.

3. We conduct extensive experiments on Argoverse dataset, showing the superior efficiency of our proposed method.

6.2 Related Work

3D detection models can be roughly categorized as: bird's-eye view, voxel-grid, graph, and range-view representation models. Unlike 2D images, point clouds are amenable to a number of different representations — each with distinct advantages and disadvantages.

Bird's-eye view Representations. 3D perception using *2D* convolution enables fast, efficient feature aggregation due to mature, highly optimized kernels available in open-source libraries; however, these methods must be carefully designed to encode geometric information in the gravity-aligned axis. [121] encode a point cloud as a "pseudo-image" — applying a PointNet [167] encoding to a set of sparse pillars in the BEV. [36] explore a multi-sensor fusion model which consists of a bird's-eye and range view of lidar sensor data, and RGB imagery. However, ego-centric point clouds are *not* dense in the BEV which consequently wastes both memory and computation. Specialized sparse operators *may* address the issues of density, but are often not as tuned for gpu-based computation.

Voxel-grid Representations. 3D convolution provides rich, expressive geometric features at the cost of a cubic run-time w.r.t. the quantized grid dimensions — leading to considerable compute challenges. [253] introduced the first end-to-end learning approach for 3D object detection by augmenting point features with positional encodings within a voxel-grid. [236] exploited the sparsity of a point cloud through 3D *sparse* convolution — greatly improving run-time to speeds suitable for exploring real-time applications. [206] combine voxel and point level processing to exploit the *efficiency* of a regular grid and the *geometric richness* of fine-grained points. Similar to our work, [250, 258] emphasize that point clouds become *sparse* at range — leading to an *imbalanced* spatial distribution of points. The authors address this observation by representing point clouds with polar and cylindrical representations, respectively; however, this can lead to spatial distortions and break the translation equivariance assumed by convolutional filters.

Graph Representations. Graph representations of point clouds encode *dynamic* neighborhoods between points while also permitting a sparse representation in the form of a sparse matrix or adjacency list [168, 223]. [194] operate directly on point clouds, without voxelization, using point-wise feature vectors for bottom up proposal generation. [94] propose using random sampling to process large point clouds to circumvent costly processing from farthest point sampling and inverse density sampling. Graph representations oftentimes require *costly* neighborhood computation using methods such as: k-nearest or fixed-radius nearest neighbors. Despite their flexible representation, 3D detection models on state-of-the-art leaderboards are still dominated by voxel-grid and BEV based methods [22, 199].

Range-view Representations. The range-view refers to the projection of an unordered set of three-dimensional coordinates onto a two-dimensional grid which contains the distance from a *visible* point to the sensor. Unlike the voxel-grid or graph representations, the range-view is not information-preserving for 3D data, i.e., each sensor return must have a clear line-of-sight between itself and the vantage point. [156] combine a range-view representation with probabilistic cuboid encoding for 3D detection. [28] explore applying different kernels to the range-view image to counteract perspective distortions and large depth gradients w.r.t. to inclination and azimuth. [200] construct a two-stage approach: first performing foreground segmentation in the range-view, then applying sparse convolutions on the remaining points.

6.3 Approach

In this section, we first introduce our proposed MRP representation, and then show how we build a detector using this representation. Lastly, we discuss options for sharing features across different range representations.

6.3.1 Multi-Range Pyramid

Without loss of generality, we assume a square processing range with an equal distance along the x and y axes in this paper. Let R be the intended processing range in the BEV, and V to be the base voxel size[1]. Given sparse point cloud features (C channels) as input, we construct a P-level multi-range pyramid (MRP) through voxelization and scattering with different range and voxel size settings. Specifically, at level i of the pyramid, it covers a range of $r_i = \frac{R}{2^{i-1}}$ and uses a voxel size of $v_i = \frac{V}{2^{i-1}}$ for $i \in 1, ..., P$. Note that the spatial dimension of the feature map at each level remains a constant throughout the pyramid: $w_i = r_i/v_i = R/V := W$. Also, voxelization and scatter operations do not alter the number of channels, and therefore, the feature map at level i contains C channels. Finally, the entire MRP is created by stacking each level-i feature map along the channel, resulting in a single feature map of size $PC \times W \times W$.

To facilitate a fair and simple comparison with the baseline model, we restrict the MRP size to match exactly with the baseline: if the baseline backbone takes $C_\text{b} \times W_\text{b} \times W_\text{b}$ as input, we construct an MRP with the size of each level to be $\frac{C_\text{b}}{P} \times W_\text{b} \times W_\text{b}$. The reduction in channels is achieved via a linear layer placed at the end of the point processing pipeline. When the model architecture is fixed, the dominating factors for the size of model are the range and the voxel size. Therefore, we introduce a notation r/s to represent the complexity of the model (or the base size of the feature map), where r is the range and s is the *reciprocal* of the voxel size, since voxel sizes are usually smaller than 1. For example, $100/4$ represents a model with detection range 100m and a voxel size 0.25m. An equal size MRP with 2 levels can be represented by $100/4 + 50/8$.

Next, we explain how we construct MRP in practise. To share computation, we can first perform voxelization and scattering once to create a huge

[1]More precisely, we use the term "voxel size" to denote the voxel dimension in the x dimension. In general, a voxel size is a 3D dimension. In this paper, we do not change the voxel size along the z (height) axis, and we assume a symmetric voxel is used along the x and y axes.

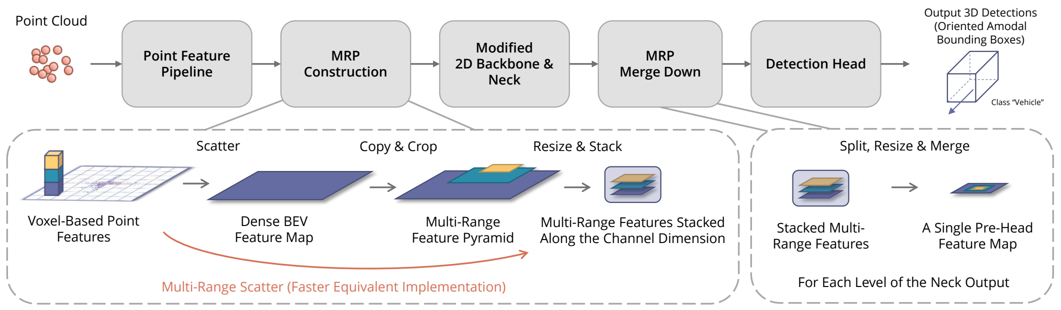

Point Cloud
Point Feature Pipeline
MRP Construction
Modified 2D Backbone & Neck
MRP Merge Down
Detection Head
Output 3D Detections (Oriented Amodal Bounding Boxes)
Class "Vehicle"
Scatter
Copy & Crop
Resize & Stack
Voxel-Based Point Features
Dense BEV Feature Map
Multi-Range Feature Pyramid
Multi-Range Features Stacked Along the Channel Dimension
Multi-Range Scatter (Faster Equivalent Implementation)
Split, Resize & Merge
Stacked Multi-Range Features
A Single Pre-Head Feature Map
For Each Level of the Neck Output

Figure 6.3: Our proposed multi-range pyramid (MRP) for 3D object detection. Building upon existing BEV frameworks (*e.g.*, PointPillars [121]), we replace multiple range-specific 2D BEV backbones with a single MRP that enables across-range feature sharing. MRPs are constructed by first computing sparse voxel-based point features at the finest resolution of interest and the largest range of interest. These sparse features are then multi-range scattered to multiple feature maps, tuned for different ranges and voxel grid sizes. MRPs then share features across range-specific feature maps, both through global pooling (Fig. 6.4) and local multi-range convolutions (Fig. 6.5). Finally, the output from range-specific neck layers are resampled into a single high-resolution, long-range feature map that is processed by a detection head.

dense feature map of size $R/V \times 2^{P-1}$, then we duplicate the crop the feature maps to form the pyramid. Finally, we resize and stack them together. This procedure is depicted in the bottom left of Fig. 6.3. The benefit is that the point processing pipeline gets shared and it can be implemented trivially with existing set of operations. However, the explicit creation of such a large feature map is prohibitively expensive in GPU memory for large ranges (*e.g.*, 6400×6400). To workaround this issue, we introduce a new operator that we call *multi-range scatter*. It takes fine-grained voxels (at size $\frac{R}{2^{P-1}}$) as input and directly construct a MRP with all levels stacked together. Compared with standard scattering operation, the multi-range version handles clashes of voxels mapped to the same cell due to the conceptual resizing operation. The implementation is fast and introduces overhead of only 2ms.

6.3.2 MRP-Based 3D Detectors

Once we have MRP defined, we show how to incorporate it into 3D detectors. First, we assume the detector make use of BEV representation at some stages, which usually correspond to the backbone and the neck. As show

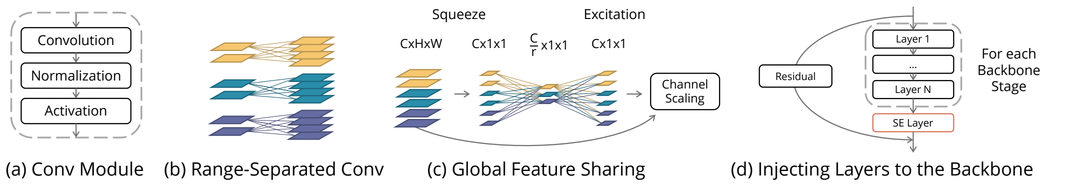


Figure 6.4: Global feature sharing in MRP. (a) If we apply the standard convolution module on unaligned multirange feature maps, range information may be mixed in suboptimal ways. (b) Therefore, we use group convolutions to restrict the information flow across backbone layers to be range-specific. (c) To allow some feature sharing across ranges, we introduce a squeeze & excitation (SE) that computes global features that do not require spatial alignment. (d) We inject SE layers at the end of each stage in the backbone.

in Fig. 6.3, we inject MRP construction and inversion layers to before and after the BEV processing stages within the base 3D detector. We modify the BEV processing part itself to make it compatible with MRP (discussed in the next subsection), but keep everything else (*e.g.*, point feature extraction, and detection head) the same as our base model. The construction process is based on the multi-range scatter introduced in the previous section. The inversion process is a simple chain of split, resize and merge operations. Specially, given an MRP of size $PC \times W \times W$, we first split along the channel dimension to get P levels of feature maps and then resize level i by a factor of 2^{1-i}. Intuitively, this is aligning the spatial representation according to the bottom level of the pyramid. Then we take the finest information for each range intervals from the respective level (*e.g.*, the center region feature is always from the top level) to form a single feature map of size $C \times W \times W$. To be exactly consistent with the base model pipeline, one could upsample the channels to PC, but we find such an operation makes little difference in practise.

6.3.3 Multi-Range Feature Sharing

After adopting the MRP representation, if we simply use the original backbone and neck, the model may produce suboptimal predictions due spatial alignment. As shown in Fig. 6.1, each level of the stacked MRP feature map correspond to a different range of the input point cloud. If we apply standard convolution that mixes the channels together, we are forcing the network to resolve the spatial alignment by itself, which may not be the most efficient way to make use of channels. The next immediate solution might be processing different level separately with P different backbone and necks so that the spatial alignment can be avoided. However, running multiple models introduces unnecessary overhead in function calls and GPU kernel calls. Taking both factors into consideration, we replace each standard convolution with group convolution with group equal to the number of levels P (Fig. 6.4ab). Note that the normalization and activation layers in a convolution module do not mix channel together by default and there is no need of group normalization. With such a replacement, we are able to "run" multiple range models within a single model. Note that this design is agnostic to specific choice of the backbone and the neck as long as they are convolutional. Since we are using group convolutions, the theoretical FLOPs of an MRP-based model is smaller than the baseline model with the same feature size. More precisely, the FLOPs of MRP is $1/P$ of the original model for each convolution module.

While the range representations are now disentangled during convolution, processing each range map separately may not be efficient usage of available channels. We propose two ways of feature sharing across different levels of the pyramid.

Global feature sharing. The spatial alignment is not an issue if the spatial resolution is 1×1. This suggests that we can first pool globally and then convolve the features across range groups. Fortunately, the community has already found a way to effectively incorporate the globally pooled features into downstream layers, and that is through a squeeze & excitation (SE) layer [93]. We illustrate an SE layer in Fig. 6.4c and where we place them in the network in Fig. 6.4d. Once we insert SE layers into the network, sharing takes place automatically, at a global level. We add an SE layer at the end of each backbone stage, and add an residual connection if it is not present. No SE layer is added inside the neck. Note that SE layers are cheap to compute and adds only negligible overhead to the overall runtime.

Local feature sharing. We offer an alternative fine-grained sharing strategy at a larger computation cost (Fig. 6.5). The key idea is we first make use of slicing and resizing operations to align the features maps and perform convolutions over aligned features in a cascade fashion from the top to the bottom of the pyramid. We observe that level i in an MRP corresponds to the center region of level $(i-1)$, but with a different spatial resolution. We define a new operation multi-range convolution (MRConv) in this way:

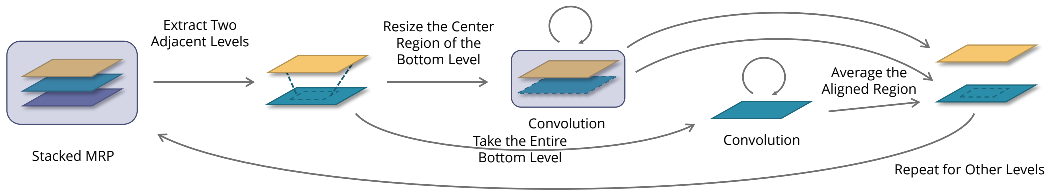

Extract Two Adjacent Levels
Resize the Center Region of the Bottom Level
Stacked MRP
Convolution
Take the Entire Bottom Level
Convolution
Average the Aligned Region
Repeat for Other Levels

Figure 6.5: Local feature sharing in MRP. We explore an alternative feature sharing strategy that replaces each standard convolution in the backbone and neck with a multi-range convolution (MRConv). The key idea is to align feature maps spatially and then perform convolution on the aligned portion. A detailed description can be found in Section 6.3.3.

we first take the center slice and of level i, upsample it by 2x and then concatenate with level $(i-1)$ for convolution. Then we split the results by half along the channel dimension. One half becomes the new level i feature map, and another half is downsampled and added back to the center region of the original level $(i-1)$ feature map. We apply another convolution just for level $(i-1)$ to propagate the fine-grained information from the center to the outer region. We repeat this process for all levels of the pyramid $(i \in P, ..., 2)$.

6.4 Experiments

In this section, we first show how detection range affects the accuracy-latency tradeoff for 3D detectors. Then we evaluate MRP on an autonomous driving dataset Argoverse [29]. We include additional ablation and generalization experiments in Appendix 6.A.1.

6.4.1 Dataset and Setup

We conduct our experiments on the publicly available Argoverse dataset. It is an autonomous driving dataset with data collected in two US cities. It considers 3 semantic classes for the 3D detection task: vehicle, pedestrian and bus. The visual sensor suite include two top mounted LiDARs, 360 degree ring cameras, and high-resolution stereo camera pairs. Notably, their setup produces long-range LiDAR point clouds and object annotations (up to ±200m). In comparison, the detection ranges for other datasets are much smaller: KITTI [70] ($+70$m, only the front-facing portion), nuScenes [22] (±50m), and Waymo [199] (±75m). The LiDAR operates at 10 Hz, and thus we adopt a 5-frame aggregation (for densification) so that the inputs are captured within the time window of 0.5 second, which matches the setup on nuScenes.

We evaluate our model with the official 3D detection metrics for Argoverse 1.1: average precision (AP), average translation error (ATE), average scale error (ASE), average orientation error (AOE), and the composite detection score (CDS). AP is computed as an average of four different true-positive thresholds: 0.5, 1.0, 2.0, and 4.0 meters. CDS, the overall summary metric, is the product of AP and the sum of the complement of the normalized true positive errors:

$$\mathrm{ATM} = 1 - \mathrm{ATE}_{\mathrm{norm}}, \tag{6.1}$$

$$\mathrm{ASM} = 1 - \mathrm{ASE}_{\mathrm{norm}}, \tag{6.2}$$

$$\mathrm{AOM} = 1 - \mathrm{AOE}_{\mathrm{norm}}, \tag{6.3}$$

$$\mathrm{CDS} = \mathrm{AP} \times (\mathrm{ATM} + \mathrm{ASM} + \mathrm{AOM}). \tag{6.4}$$

6.4.2 Baseline and Implementation Details

We adopt PointPillars [121] as our baseline detector since it is one of the state-of-the-art among efficient 3D detectors. The model first distributes points into a pillar-based representation and refines it with PointNet. Then it scatters the sparse representation to form a dense feature map. Next the dense map is processed by the SECOND backbone, a simple feed-forward network without branches. Outputs from three stages of the backbone are extracted and fed into the neck (SECONDFPN), which unifies the dimensions of the three stage features and simply concatenate them together. The concatenated feature is processed by a minimal SSD-like detection head, which contains the classification branch for object labels and a regression branch for bounding box location, size, and orientation. We mostly adopt the standard configuration for the baseline except for tuning the range, voxel size and the base learning rate and adapting the anchor configuration to Argoverse. Detailed description of the architectures can be found in the Appendix 6.A.2.

For our MRP adaption, we use a pyramid of 2 levels unless otherwise stated. We use the same set of hyperparameters and data augmentations as our baseline. Regarding the SE layers we insert to both the baseline and our models, we use the default reduction ratio of 16 during the bottleneck convolution. Furthermore, when combining MRP with SECONDFPN, the default order is to first concatenate the features and then apply MRP merge down since MRP merge down is designed to be performed outside of the

Table 6.1: Detection range is a tunable parameter. This table shows what can be achieved by simply tuning the range parameter and sometimes together with the voxel size. Please see Section 6.4.3 for a detailed discussion. We present the results of our proposed approaches in the next table (Table 6.2).

ID	Method	0-25m	25-50m	50-100m	100-200m	0-50m	0-100m	0-200m	Runtime (ms)
1	100/4 → 25	41.5				20.5	13.0	11.7	21.0 ± 3.0
2	100/4 → 50	42.2	35.8			38.7	25.1	22.7	23.0 ± 2.3
3	100/4 → 100	41.8	36.0	**22.8**		38.8	33.1	29.9	44.8 ± 2.3
4	100/4 → 200				**6.4**			30.0	194.0 ± 3.0
5	25/16 → 25	**48.4**				23.6	15.3	13.9	43.5 ± 2.6
6	50/8 → 50	46.6	**36.5**			41.1	26.9	24.4	48.2 ± 3.0
8	200/2 → 200	34.7	27.7	18.6	5.7	31.3	26.6	24.1	42.4 ± 2.1
9	Range ensemble (4,5,6)	**48.4**	**36.5**	**22.8**	**6.4**	**41.7**	**35.3**	**32.5**	285.7 ± 8.6

neck. However, the concatenation step causes the spatial alignment issue. Therefore, we simply swap the order of these two operations, *i.e.*, we first perform MRP merge down, and then concatenate the features to feed into the detection head.

We use the open source implementation from mmdetection3d [41]. We adopt a basic set of data augmentations, which include global 3D tranformations, flip in BEV, and point shuffling. We follow the standard 2x schedule to train the model, which implies training the model for 24 epochs with AdamW optimizer and step decay schedule (decay the learning rate by one magnitude at epoch 20 and 23). We use a base learning rate of 0.005 and weight decay 0.01. We train our model with 8 RTX 3090 GPUs and a batch size of 2 per GPU. Under this setting, the max range/voxel size configuration we can fit is 100/4, or equivalently 200/2. In other words, we have to use a very coarse voxel size of 0.5m if we want to train a model with ±200m input range (200/4 does not fit into the GPU memory even with a batch size of 1). The training noise (from random seed and system scheduling) is around 1% of the accuracy (standard deviation normalized by the mean), which is at an acceptable low rate, and also we report the medium of 3 runs for key experiments. For all the runtime reported in this paper, we evaluate the batch size 1 inference time on a Tesla V100 GPU, as this is the common setting for efficiency benchmarking [129,205,209].

6.4.3 Range is a Tunable Parameter

As we have mentioned earlier, detection range is largely considered as a constant in the literature. However, we point out that many can be achieved if

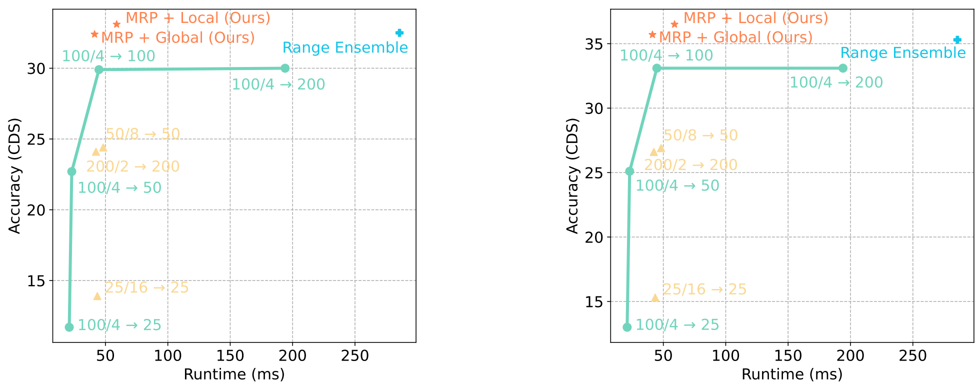


Figure 6.6: Efficiency comparison between the baselines and our methods. On the left is 0-200m evaluation while on the right it is 0-100m evaluation. In both cases, we notice that our methods achieve similar or better accuracy as the expensive range ensemble baseline with much less computation cost.

we properly tune the detection range together with the voxel size. The results are summarized in Table 6.1.

From a given baseline model (PointPillars [121]), we derive families of models based on the range parameter. We evaluate the accuracy across different range intervals and the latency of these models. Since the baseline detector is fully-convolutional (as in many other detectors), we can run inference at a different range from training. Taking this into consideration, we introduce a new notation of $r_1/s \to r_2$, where r_1 represents the range the model is trained at, v represents the reciprocal of the voxel size and s_2 represents the inference range. Note the voxel size needs to remain the same during training and testing.

First from row 1-4, we show that by limiting the range at inference time, we can derive a family of models with increasing accuracy (0-200m) and latency. However, these derived models may be sub-optimal at their respective range. When the range is limited, one can afford to train the model at a finer voxel resolution (with a fixed model size at training time, the product of r and v remains a constant). Therefore, we further optimize the voxel size for different ranges at rows 5-8 and we can these models *range experts*. Interestingly, as shown in row 8, the range expert for the 200m performs worse than row 4, where we run a 100m-trained model at 200m range. Lastly, we create a multi-range ensemble (row 9) by taking the outputs from the best model at a specific range (shown in bold). Unsurprisingly, this ensemble outperforms all other methods at the cost of a huge latency.

Table 6.2: Main results on Argoverse. All measures except the last column are accuracy measures, which are the higher the better. "Veh" and "Ped" are short for "Vehicle" and "Pedestrian" respectively. A corresponding range-based breakdown is provided in Table 6.3. The first three rows are the baseline while the latter four are our methods. We construct our MRPs on top of the well-tuned baseline (row 1). We start with an MRP adoption without the modified backbone and neck (row 4), this yields worse performance than the baseline due to spatial alignment issues. The alignment issue can be resolved by using group convolution as described in Section 6.3.3 (row 5). We can further add either global (row 6) or local (row 7) featuring sharing to our MRP-based methods, both improving the accuracy. Notably, the global sharing (row 6) improves the accuracy while running faster than the baseline (row 1 & 2). However, we do acknowledge that our method performs worse than the baseline in terms of orientation estimation.

ID	Method	CDS	AP	ATM	ASM	AOM	Veh	Ped	Bus	Runtime (ms)
1	Tuned baseline (100/4 → 100)	29.9	38.8	70.4	76.1	**77.4**	54.1	29.3	6.3	44.8 ± 2.3
2	+ Squeeze & excitation	31.0	40.4	71.8	75.9	77.3	54.7	32.6	5.6	45.8 ± 2.2
3	Range ensemble	32.5	43.1	73.9	74.5	65.5	**54.8**	36.7	5.9	285.7 ± 8.6
4	MRP - modified bkbn & neck	29.7	40.0	72.2	74.8	68.4	48.4	33.0	7.6	48.3 ± 3.1
5	MRP	30.7	40.5	70.0	**76.4**	73.5	52.4	33.3	6.5	**40.1** ± 2.9
6	MRP + global sharing	32.4	42.5	74.5	74.3	70.5	54.2	34.4	8.5	41.3 ± 3.1
7	MRP + local sharing	**33.1**	**43.4**	**75.6**	76.1	74.4	53.4	**37.3**	**8.6**	59.0 ± 3.2

The results in this table are plotted in Fig. 6.2 as well. From both the table and the figure, it's clear that the configuration 100/4 → 100 is the sweet spot for accuracy and latency tradeoff.

6.4.4 Evaluation for Multi-Range Pyramid

We first plot the performance of our MRP models in accuracy vs runtime plots in Fig. 6.6. The plots show how the baseline model is trading off accuracy and latency under different range settings and how our final methods perform compared to the baselines. We find that our proposed MRP models are above the pareto-optimal curve of the baseline model, suggesting that they have achieved a better computation tradeoff.

Next, we provide detailed evaluation of these methods in Table 6.2 & 6.3, with the first showing metric breakdown under typical setup (breaking down according to translation, scale and orientation, and also according to semantic classes), and the second showing the breakdown under each range interval. For the baselines, we first include PointPillars with optimal tuned range (row 1). Then we include a variant with squeeze and excitation layers (row 2). This aims at providing a fair comparison with our global sharing variant (row 6). To match the feature size of the $100/4$ baseline, we use an MRP of size $100/4 + 50/8$. Comparing row 4 against row 1, we see a drop with naive MRP adaption. We argue that the culprit lies in the spatial feature alignment in the backbone and the neck. The accuracy improves after the backbone and the neck is properly modified (row 5). Next, we find adding feature sharing significantly increases the accuracy. One might argue that the boost may come from the mechanism of SE layer itself, since they are considered plug-and-play modules that usually boost models' accuracy. By comparing row 2 to row 1, we see the boost of SE layer itself is around 1 point in CDS. By comparing row 6 and row 5, we see a much larger boost of 1.7 in CDS, and this demonstrates that the global feature sharing is boosting the accuracy of the model. At the same time, we notice that this sharing strategy introduces negligible overhead (around 1ms). If one can afford to have slightly more latency in their system, the local sharing variant becomes a good candidate as it further boosts the performance upon the global sharing variant, Surprisingly, it achieves better CDS compared to the expensive range ensemble baseline. Table 6.3 shows a sizeable improvement in close by CDS of our methods compared to the baselines, under the context of long-range processing. This can be safety-critical to autonomous driving since the closer the object is, the shorter *time-to-contact* is. Note that despite we show zero accuracy for the 100-200m range, our models are still capable of running at 200m at test time since it is also fully-convolutional. We choose not to since it will produce a suboptimal point in the accuracy-latency tradeoff.

Finally, we provided a detailed component-wise runtime analysis for these models in Table 6.4. All these models benchmarked share the same sizes of feature maps at the corresponding layers. We perform CUDA synchronization every time before we record the values from the timers, this ensures accurate timings for each component. We first note that the backbone is the most time-consuming component within a backbone. Our MRP models are able to bring down the latency of this part by using group convolutions. MRP also introduces additional overhead to the point processing pipeline and the BEV construction step. The former is due to the need to process at a finer voxel level, and the latter is due to the fact that there are bin-collisions within the multi-range scattering operation. The last row shows that the multi-range convolution used for local feature sharing introduces much latency for the backbone. Note that for all these models, post-processing is taking up a large chunk of time. This is a result of us using research-level code, and it can be further optimized, but it is beyond the scope of this work.

Table 6.3: Range interval evaluation for the main results. This table contains the same set of methods as in Table 6.2, but offers accuracy breaking down according to range intervals. The summary metric CDS is used here. We notice that due to we process a finer-grained voxel grid for close range, our methods (row 4-7) achieve much better short range accuracy compared to the baselines (row 1-2). Armed with feature sharing, our local sharing model (row 7) even outperforms the expensive ensemble baseline (row 3) for short ranges.

ID	Method	0-25m	25-50m	50-100m	100-200m	0-50m	0-100m	0-200m
1	Tuned baseline (100/4 → 100)	41.8	36.0	22.8		38.8	33.1	29.9
2	+ Squeeze & excitation	43.4	36.8	**24.7**		39.8	34.3	31.0
3	Range ensemble	48.4	36.5	22.8	**6.4**	41.7	35.3	32.5
4	MRP - modified bkbn & neck	45.7	34.9	19.1		40.1	32.7	29.7
5	MRP	45.1	37.2	20.8		41.0	33.9	30.7
6	MRP + global sharing	48.1	39.0	21.3		43.2	35.7	32.4
7	MRP + local sharing	**49.3**	**39.1**	22.0		**44.1**	**36.5**	**33.1**

Table 6.4: Runtime breakdown for the baseline and our MRP models. The units for the runtime is milliseconds (ms). "Point Proc" is short for the "point processing pipeline", which include voxelization and PointNet feature extraction. This table shows that our MRP models (row 2-3) are overall faster due to the backbone stage is much faster with MRP.

ID	Method	Point Proc	MRP/BEV Constr	Backbone	Neck	MRP Merge	Head	Post Proc
1	Baseline (100/4 → 100)	7.1	1.7	16.1	4.8	N/A	1.9	13.2
2	MRP	10.7	3.9	8.0	2.6	1.1	1.1	13.8
3	MRP + global sharing	10.8	3.7	9.4	2.5	1.1	1.1	13.0
4	MRP + local sharing	10.7	3.8	26.5	2.6	1.1	1.1	13.6

6.4.5 Ablation Study

In this subsection, we provide ablation study for the location of SE layers in Table 6.5. We have explored inserting it at various stages of the backbone. However, we find these design choices have little effect on accuracy and latency. Therefore, we adopt the first setting where we insert SE to every stage of the backbone, since we find this setting the simplest conceptually. We include additional ablation study in the Appendix 6.A.1.

Table 6.5: Ablation study for global feature sharing. This table explores the different impact of where to inject SE layers into the backbone (as shown in Fig. 6.3).

ID	SE Stages	CDS	AP	ATM	ASM	AOM	Runtime (ms)
1	[1, 2, 3]	**32.4**	42.5	**74.5**	74.3	70.5	41.2 $\pm$ 3.2
2	[2, 3]	**32.4**	**42.7**	72.5	**75.5**	**73.1**	40.6 $\pm$ 3.1
3	[3]	32.3	42.3	70.8	75.1	72.7	40.3 $\pm$ 2.9

6.5 Conclusion

We provide analysis on the effect of detection range for 3D object detectors, showing that range can be used to tradeoff accuracy and latency. We use our analysis to build a simple ensemble of range experts that exploits a fundamental property of lidar; namely that measurements become sparser at range, allowing for coarser voxel binning. While performant, an ensemble of range experts can be slow. We then propose to share features across range experts with a multi-range pyramid. To do so, we introduce multi-range scatter operations that enable a single set of sparse point features to populate multiple range- and resolution-specific feature maps. These feature maps are arranged into a multi-range backbone that supports both global and local convolutional feature sharing across ranges. In future work, it might be useful to process different ranges *asynchronously* at different timestamps. For example, close range feature maps might need to be visited more frequently than far range ones for autonomous navigation, suggestive of a near-far network analogous to asynchronous slow-fast networks for video processing [63].

6.A Appendix

6.A.1 Additional Generalization and Ablation Experiments

In the main text, we mentioned additional generalization and ablation experiments in Section 4. In this section, we first present an analysis on across-dataset generalization, showing our conclusions generalizes to nuScenes in the long-range setting. Next, we show our methods are generalizable to architectural variations.

Analysis on Across-Dataset Generalization

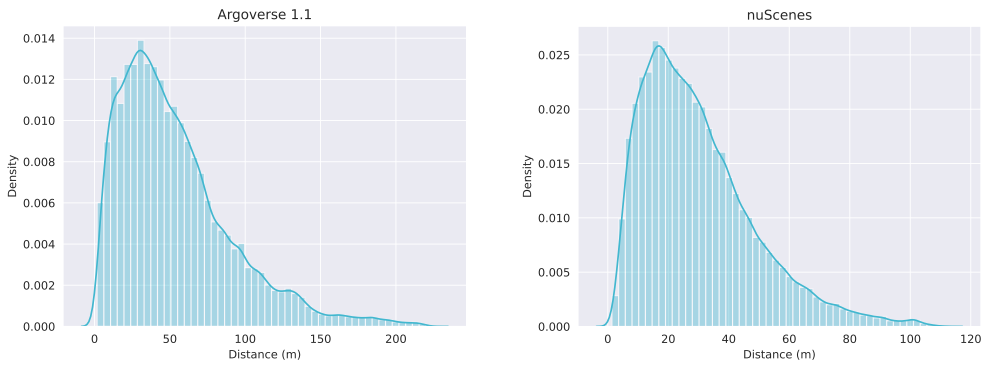


Figure 6.7: Comparison of Argoverse 1.1 and nuScenes based on the ground truth object distribution over the distance. The horizontal axis is the distance of the object to the ego vehicle and the vertical axis is the probability density. This plot shows that nuScenes is a relatively short-ranged dataset compared to Argoverse 1.1.

In the introduction section of the main text, we mentioned the need a of long-range dataset for evaluation and nuScenes is not fit for this reason. We further clarify this statement in subsection. We compare the object distribution according to distance for both Argoverse and nuScenes in Fig. 6.7. The plots show significantly different distributions among two datasets in terms of object-ego-vehicle distance. We see Argoverse 1.1 contain much more far away objects relatively. We point out that *the object distribution in nuScenes may be an artifact of the sensor setup and may not reflect the true object distribution*. Note that for the plotting the nuScenes distribution, we filter out zero-LiDAR-point annotations (consistent with the standard training and evaluation protocol), which are mostly interpolated boxes for occluded objects.

Table 6.6: Long-range evaluation (100m) on nuScenes. In this table, we compare our methods against the baselines with ground truths and predictions in the 0-100m range. The columns are standard nuScenes metrics and the per class breakdown is regarding to the mAP. Some classes have the spelling names abbreviated to fit the space, and the full class names are (in order from left to right) car, truck, trailer, bus, construction vehicle, bicycle, motorcycle, pedestrian, traffic cone, and barrier. We note that our proposed MRP with or without the global sharing outperform their baseline counterparts. Also, the much larger runtime (compared to that on Argoverse) is due to longer post-processing time.

Method	NDS	mAP	car	trck	trail	bus	cstr	bic	motc	ped	cone	barr	Runtime (ms)
Baseline (100/4 → 100)	41.5	23.4	67.9	**27.1**	12.4	33.5	1.3	0.2	11.9	48.5	9.0	22.2	72.8 $\pm$ 3.2
+ Squeeze & excitation	42.4	24.6	**68.4**	27.0	**16.8**	**36.2**	1.3	0.5	12.2	49.5	8.1	26.2	73.0 $\pm$ 2.9
MRP	42.0	23.0	66.0	25.0	6.5	27.6	**1.4**	**0.7**	14.5	54.3	7.4	26.5	**67.7** $\pm$ 3.8
MRP + global sharing	**43.0**	**24.8**	67.5	**27.1**	9.8	32.4	**1.4**	0.3	**14.6**	**56.3**	**10.5**	**28.4**	68.5 $\pm$ 3.8

In addition, we notice that despite the official nuScenes evaluation goes to 50m, there are 13.9% objects being farther away than 50m. Based on this observation, we construct a new experiment setting where the training and testing goes to 100m for all classes on nuScenes, and then we run the same baseline and our methods. The results for this long-range setting is summarized in Table 6.6. We observe similar conclusions as the case with Argoverse 1.1 where our proposed methods outperform the baselines. Therefore, the improvement of our methods is not just limited to a single dataset. We plan to evaluate on more long-range datasets in the future as they become available, to reinforce the claim that our method generalizes across datasets.

Generalization to Architectural Variations

The neck used in our baseline is SECONDFPN [236], which simply concatenates multi-stage backbone outputs and produces only a single-level feature map for the detection head. In contrast, in 2D detection literature, a feature pyramid network (FPN) neck [137] is widely adopted for its efficient top-down reasoning pipeline and its multi-level outputs [205]. In this section, we conduct experiments with FPN and present the results in Table 6.7. Note that when using MRP with a neck that produces multiple feature maps, we perform individual MRP merge down operations on each feature map separately. The results suggest the same set of conclusions where our MRP, without or without sharing, outperforms the corresponding baseline.

Table 6.7: Generalization to another architecture setting. Our experiments default to SECONDFPN as the neck. In this table, we adopt FPN for both the baseline and our models. We observe that the previous conclusions still hold under this new architectural setting.

ID	Method	CDS	AP	ATM	ASM	AOM	Runtime (ms)
1	Baseline (100/4 → 100)	30.6	40.1	70.3	76.4	74.7	65.6 $\pm$ 2.6
2	+ Squeeze & excitation	31.6	41.1	**72.5**	75.7	72.8	66.5 $\pm$ 2.9
3	MRP	31.4	41.6	70.7	**76.5**	71.4	**59.7** $\pm$ 3.4
4	MRP + global sharing	**33.5**	**43.2**	69.7	75.0	**78.4**	61.3 $\pm$ 3.3

Range-Averaged Evaluation

Our final methods in the main text do not process the point clouds in the range from 100m to 200m. Such a conclusion is a combined results of the skewed data distribution and the all-in-one-pool evaluation metrics. Since the data distribution cannot be easily altered, we adopt an alternative evaluation protocol in this section, with an additional focus on long-range detection. Specifically, we first compute the per range interval accuracy and then compute an average of these accuracies. Such a metric is also referred to as macro average performance in the classification literature [239]. By averaging across range intervals, the weight of each range interval is the same regardless of the data distribution (as long as it has non-zero density). Therefore, the 100-200m range accuracy is valued more under this alternative metric. Accordingly, we also run the baseline and our methods on 200m input. The results are shown in Table 6.8 where the added methods are at the bottom of this table. We observe that in terms of macro average accuracy (CDS), range ensemble (row 3) is the highest performing method, but it has a huge runtime cost. The second is our local sharing method (row 8) running with 200m input. It reduces the runtime cost of range ensemble by 38% while achieves similar accuracy. At the same time, it is faster and more accurate than simply running the baseline at 200m (row 8). Our methods (row 6-7) running at 100m still remain the most competitive for balancing runtime and accuracy.

Ablation Study on MRP Merge Down

When we perform the MRP merge down operation that converts the MRP into a single feature map (see Fig. 2), we use *overwrite* as the merging op-

Table 6.8: Range-averaged (macro-accuracy) evaluation. In this table we average the CDS computed from 0-100m and 100-200m in the second column as an alternative way of evaluation, which places higher weight to long-range detection given skewed data distribution. Please refer to Section 6.A.1 for a detailed discussion.

ID	Method	0-100m	100-200m	Average	Runtime (ms)
1	Tuned baseline (100/4 → 100)	33.1		16.6	44.8 $\pm$ 2.3
2	+ Squeeze & excitation	34.3		17.2	45.8 $\pm$ 2.2
3	Range ensemble	35.3	**6.4**	**20.9**	285.7 $\pm$ 8.6
4	MRP - modified bkbn & neck	32.7		16.4	48.3 $\pm$ 3.1
5	MRP	33.9		17.0	**40.1** $\pm$ 2.9
6	MRP + global sharing	35.7		17.9	41.3 $\pm$ 3.1
7	MRP + local sharing	**36.5**		18.3	59.0 $\pm$ 3.2
8	1 + (→ 200m)	33.0	**6.4**	19.7	198.7 $\pm$ 3.5
9	7 + (→ 200m)	35.2	6.0	20.6	175.9 $\pm$ 4.0

eration. In other words, during the merging of each two consecutive layers, we directly take the resized top layer and pasted it to the center region of the bottom layer. In this section, we explore other merging operations. The first one is *sum*, where the new center region of the bottom layer is the sum of the old center region and the resized top layer. The second one is *convolution*, where we first concatenate the old center region and the top layer along the channel dimension, and then perform a standard convolution with channel reduced by half in the output. Note that for all three operations, including the original overwrite, the size of the feature map remains the same and thus they are compatible with the overall MRP merge down procedure. All operations are tested using the MRP plus global sharing as the base model. We summarize the results of this ablation study in Table 6.9. We observe that a simple *overwrite* operation outperform other more complicated operations and thus we adopt overwrite as the default merge down operation.

6.A.2 Additional Model and Implementation Details

In this subsection, we include additional architectural and implementation details on both the baseline and our method.

Table 6.9: Ablation study on MRP merge down operations. We observe that overwrite performs the best among all tested operations. For detailed definition of each operation, please refer to Section 6.A.1.

ID	MRP Merge Operation	CDS	AP	ATM	ASM	AOM	Runtime (ms)
1	Overwrite	**32.4**	**42.5**	**74.5**	74.3	70.5	**41.2** $\pm$ 3.2
2	Sum	31.8	41.7	69.7	**75.2**	**77.0**	41.5 $\pm$ 2.9
3	Convolution	31.7	41.4	72.7	74.8	76.2	44.4 $\pm$ 3.1

6.A.3 Additional Details about the Baseline

As mentioned in the main text, we adopt PointPillars [121] as our baseline, which includes a pillar-based point processing pipeline, 2D backbone, concatenation-based neck and a standard anchor-based detection head. The model uses a voxel (pillar) size of 0.25m $\times$ 0.25m $\times$ 8m, with a input point cloud range of $[-100, 100]$m $\times$ $[-100, 100]$m $\times$ $[-5, 3]$m (taking the 100/4 profile here as an example). For the voxelization step, we set points per voxel to be 20, and max voxel to 60,000. *Importantly, we use non-deterministic voxelization, which drastically speeds up training by* 8x. After voxelization, we adopt a one-layer PointNet as point feature encoder, which split out features of 64 dimensions. The features are then passed through a pillar scatter layer to form a dense feature map. We adopt standard SECOND backbone and SECONDFPN neck. For the detection head, the anchors sizes and ranges are adjusted for Argoverse while other settings are kept the unchanged. We train all our models from scratch without pretraining. We use the public mmdetection3d [41] on Github for our implementation and the version we use is `e5a87f3` (commit id). For the environment, we use CUDA 11.1, cuDNN 8.0.5, and PyTorch 1.9.0.

6.A.4 Additional Details about Our Methods

Most of proposed modifications are straightforward to implement given the descriptions in the main text, with two exceptions: the first is the local sharing MRConv module and the second is an equivalent fast implementation for group convolutions. To further clarify how local sharing is done. We include a simplified PyTorch code for a 2-level pyramid in `mr_conv.py`. To enable global sharing, we simply replace each regular convolution in the backbone with a MRConv described in the file.

Next, we explain how we implement group convolutions in a faster and yet equivalent way. Both our MRP and its global sharing variant make use of group convolutions to prevent unaligned feature fusion within the pyramid. Group convolutions in theory have much lower FLOPs than regular convolutions ($1/P$ in our case). However, when we implement the group convolution by setting the groups argument in PyTorch's Conv2d, we find it runs much slower (around 4x) than a regular convolution for certain input and convolution configurations. We posit that modern deep learning primitives are not optimized for group convolutions due to their less frequent usage. We present a way here to optimize this operation at the PyTorch level. Running a group convolution with group g is theoretically equivalent to running g standard convolutions in parallel. Therefore, we create another convolution layer called ConvFork that implements group convolution using multiple parallel convolutions, and we only replace group convolutions that are slower than standard convolution with ConvFork module. *This implementation results in around 10ms (20%) reduction in overall runtime*, which allows us to show off the efficiency of our approach in wall-clock time.

Chapter 7

Future Object Detection

7.1 Introduction

Object detection and forecasting are fundamental components of embodied perception that are often studied independently. In this paper we rethink the methods and metrics for trajectory forecasting from LiDAR sensor data. Trajectory forecasting is a critical perception task for autonomous robot navigation, thus building meaningful evaluation metrics and robust methods is of utmost importance.

Traditional trajectory forecasting methods [22, 29, 199] detect [193–195] and track [107,226,228] objects in 3D LiDAR scans to obtain past trajectories (Fig. 7.1*a*). These can be used in conjunction with auto-regressive forecasting methods [2, 79, 101, 189] to estimate the future actions of surrounding agents. Recent efforts [135, 149, 229] streamline such multi-stage perception stacks and train multi-task neural networks to jointly detect, track and forecast object positions directly from raw sensor data (Fig. 7.1*b*). However, such end-to-end approaches tend to predict only a single future trajectory for each object, not accounting for future uncertainty. This is not surprising as estimating multiple futures is a significant challenge in forecasting, requiring machinery such as multiple-choice-loss [13] or generative models [3,47,48,79,101,114,125,188].

We rethink the forecasting task and propose *FutureDet*, an approach that reframes forecasting as the task of *future object detection* (Fig. 7.1*c*). Importantly, existing detectors [121, 241, 253] already learn to predict heatmaps that capture distributions over possible object locations. We re-purpose this machinery to represent possible *future* object states. To this end, we encode an accumulated sequence of past raw LiDAR scans using standard

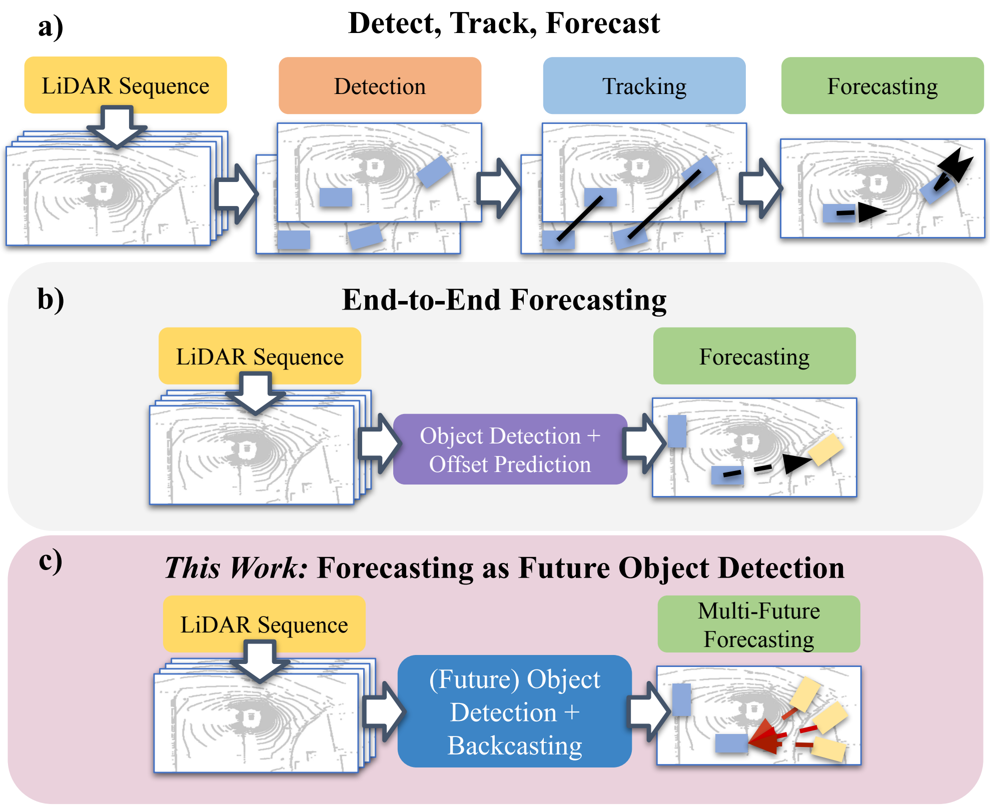


Figure 7.1: (*a*) Current stage-wise methods independently address the problems of detection, tracking, and forecasting, allowing for compounding errors in the full pipeline. Each sub-module incorrectly assumes that its input will be perfect, leading to further integration errors. In contrast to current forecasting methods that use object tracks as input, end-to-end forecasting *directly* from LiDAR sensory data (*b*) streamlines forecasting pipelines. To this end, we propose *FutureDet* (*c*), an end-to-end model capable of forecasting multiple-future trajectories directly from LiDAR via future object detection. We show that our end-to-end pipeline improves upon state-of-the-art three-stage and end-to-end methods.

backbones for 3D LiDAR-based object detection and train our network to (i) detect objects multiple timesteps into the future and (ii) estimate trajectories for these future detections back in time (*i.e.*, *back-cast*) to the current frame. By matching back-casted future detections to current detections in a many-to-one manner, our approach can represent a distribution over multiple plausible future states. Our extensive evaluation on the large-scale nuScenes [22] dataset for trajectory forecasting reveals that our proposed *FutureDet* outperforms state-of-the-art methods, *without requiring* object tracks or HD-maps as inputs to the model. We posit that tracking may *emerge* from our network (since tracking objects from accumulated past LiDAR scans may make them easier to forecast), similar to the emergence of tracking and forecasting in streaming perception [129].

Furthermore, we investigate the utility of current metrics [135, 227] for evaluating forecasting directly from raw LiDAR data. We find that existing metrics are not well suited for the task of joint detection and forecasting, allowing them to be gamed by trivial forecasters. Current metrics for end-to-end LiDAR forecasting adapt trajectory-based forecasting metrics, such as average/final displacement error (ADE/FDE). These metrics were designed for evaluating forecasting in a setting where perfect tracks are given as input, and objects don't have to be detected. However, these metrics don't adapt well to the end-to-end setting. We demonstrate that such metrics can be gamed by baselines that simply rank all stationary objects (which are trivially easy to forecast) with high confidence, dramatically outperforming all prior art. Moreover, these evaluation metrics detach two inherently interconnected tasks of detection and forecasting. Consequentially, they do not penalize *false forecasts*, *i.e.*, forecasts that do not actually belong to any objects. In this sense, the end-to-end setup and evaluation is more realistic.

To address these short-comings, we rethink the evaluation procedure for joint object detection and forecasting directly from sensor data. Our key insight is that the versatile average precision (AP) metric, a gold standard for assessing object detection performance, can be generalized to the task of joint detection and forecasting. The key feature of our novel *forecasting mAP* is that a forecast is correct *only* if the object is both correctly *detected* and *forecasted*. Our *forecasting mAP* is then calculated by simply using the machinery of AP, but using this joint detection and forecasting definition of a true positive. Furthermore, our *forecasting mAP* can be extended to evaluating multiple-future forecasts for each object by simply evaluating w.r.t. the top-K most confident forecasts per-detection. Our metric appropriately adapts forecasting metrics for end-to-end evaluation: *forecasting mAP* jointly assesses forecasting and detection, penalizing both *missed forecasts* as well as *false forecasts*. It assesses forecasting performance on the full set of object detections and embraces the inherent multi-future nature of forecasting.

Contributions: We (i) repurpose object detectors for the task of end-to-end trajectory forecasting and propose a model that can predict multiple forecasts for each current detection, (ii) rethink trajectory forecasting evaluation and show that detection and forecasting can be jointly evaluated using a generalization of well-accepted object detection metrics, and (iii) thoroughly analyze the performance of our model on the challenging nuScenes dataset [22], showing that it outperforms both previous end-to-end trainable methods and more traditional multi-stage approaches.

7.2 Related Work

Object Detection and Tracking. Due to recent advances in supervised deep learning [118] and community efforts in dataset collection and benchmarking [8,22,49,70,199], the research community has witnessed rapid improvement in LiDAR-based 3D object detection [121, 166, 193, 194], tracking [226,241], and segmentation [5,8,250]. Several methods [193,194] follow a well-established two-stage object detection pipeline using point-cloud encoder backbones and a 3D variant of a region proposal network [180], or detect objects as points, followed by classification and bounding box regression [241]. Due to the sparsity of LiDAR point clouds, recent methods accumulate multiple scans over time to improve object detection [121,241] and LiDAR panoptic segmentation performance [5]. To understand how trajectories of detected objects evolve over time, multi-object tracking methods associate detections using Kalman filters [226], learned object descriptors [65,228] or regress frame-to-frame offsets [241,252], typically followed by greedy or combinatorial optimization to resolve ambiguous data association. The latter approach can be interpreted as a single frame forecast for track association. However, autonomous vehicles must account for likely future positions of surrounding agents at longer temporal horizons to safely navigate and avoid collisions in dynamic environments.

Trajectory Forecasting. Vision-based trajectory forecasting has been posed as the task of predicting future trajectories of agents, given perfect past trajectories and a top-down image (recorded *e.g.* using drones) as input [126, 163,183]. Early physics-based models [86] explicitly model agent-agent and agent-environment interactions and have been successfully used to enhance object trackers [123, 235]. Recent methods use auto-regressive data-driven models that leverage recurrent neural networks (RNNs) and encoder-decoder-based architectures to encode the past trajectory and estimate its evolution in future frames [2]. To deal with the inherent multi-modal nature of the problem, several methods leverage generative models [74, 109] to learn a distribution over future trajectories [3, 47, 48, 79, 101, 114, 125, 188]. However, these methods and benchmarks tackle forecasting in idealized scenarios, in which the entire visual scene is directly observed, and perfect input trajectories are provided. Both are unrealistic assumptions in automotive and robotics applications. Due to the importance of forecasting for automotive path planning, recent large-scale automotive benchmarks [22, 29, 199] explicitly focus on the trajectory forecasting task. Similar to vision-based methods, these benchmarks pose the forecasting problem as trajectory estimation given past trajectories and a high-definition map of the environment. These benchmarks have facillitated a wide suite of HD-map-based forecasting methods, which either represent the HD-map as rasterized images [27, 71], graphs or vectors [67, 77, 134]. However, both algorithms and the evaluation protocol make the unrealistic assumption that detection and tracking outputs are perfect.

Forecasting from Sensor Data. Prior methods [135, 149, 229] tackle object detection, tracking, and forecasting jointly by training a single convolutional neural network in a multi-task fashion from accumulated stacks of LiDAR sweeps. Alternatively, [227] directly forecasts future LiDAR scans and leverages off-the-shelf LiDAR object detectors to detect objects in these forecasted scans. We believe that this approach of end-to-end joint detection and forecasting is a step in the right direction. However, none of the aforementioned end-to-end methods are able to reason about multiple future trajectories. To embrace the inherently uncertain future, we present an end-to-end forecasting model (Sec. 7.4) that simultaneously detects objects in the current and future timesteps given a history of LiDAR scans, anchoring multiple possible future detections to current scan detections. This approach not only outperforms the aforementioned methods w.r.t. forecasting accuracy but also allows for multiple future interpretations. Finally, we observe that ad-hoc adaptations of forecasting metrics [135, 149, 229] do not appropriately characterize certain types of forecasting errors. As a remedy, we propose a generalization of the average precision (AP) [60] metric for joint detection and forecasting in Sec. 7.3. Note that our adoption of AP is also inspired by the work of streaming perception [129], where AP is used to measure the joint performance of 2D object detection, tracking, and short-term forecasting without considering multiple futures.

7.3 Rethinking Forecasting Evaluation

Since we are tackling the task of forecasting future positions of cars directly from LiDAR scans, we assume an observed sequence of past LiDAR sensor data, up to the most recent observation $\mathcal{S}_{t_{obs}}$ at time t_{obs}, as input. We pose joint object detection and forecasting as the task of estimating a set of object locations (parametrized as 3D cuboids) in the current scan $\mathcal{S}_{t_{obs}}$ as well as their future trajectory continuations in the *future*, unobserved scans *i.e.*, $\{\mathcal{S}_t, t \in [t_{obs}+1, \ldots, T]\}$.

7.3.1 Preliminaries

Prior work [2, 79, 101, 126, 163, 183, 189] presents forecasting as the task of estimating the "correct" continuation of a given ground-truth track. In particular, given past trajectory observations $X_i = \{(x_i^t, y_i^t) \in \mathbb{R}^2, t = 1, \ldots, t_{obs}\}$, the task is to estimate future positions $Y_i = \{(x_i^t, y_i^t) \in \mathbb{R}^2, t = t_{obs}+1, \ldots, t_T\}$ for all agents present in the scene. This formalization is also adopted by recent automotive forecasting benchmarks [22, 29, 199]. However, as the ego-vehicle is moving, methods are given high-definition (HD) maps of the surrounding environment and ego-vehicle positions to account for the geometry of the surroundings. First, we discuss existing metrics for forecasting evaluation.

ADE and FDE. Average displacement error (ADE) and final displacement error (FDE) are commonly used evaluation metrics for assessing trajectory prediction. Both are measured as the L2 distance between model predictions $\{Y_i\}$ and ground truth trajectories $\{G_i\}$. To account for the inherent uncertainty in trajectory continuation, methods are evaluated over the set of top-K model predictions (w.r.t. the confidence score of each predicted forecast). These metrics assume that the set of true positives are the same for all methods. This assumption does not hold when comparing end-to-end methods, which can produce different sets of true positives, making comparison unreliable.

Miss Rate. If the final displacement error between a ground-truth trajectory and a prediction is above a center distance threshold, we count the forecast as a miss (similarly evaluated w.r.t. the set of top-K predictions). This metric evaluates the proportion of misses over all forecasts in a scene.

ADE/FDE w.r.t. Recall. The standard forecasting setup allows us to build models in isolation from other factors and has sparked rapid progress in this field of research [2, 47, 48, 79, 101, 189]. However, the standard assumption of having perfect input trajectories is not feasible in practice as it critically depends on *perfect* object tracks as inputs, which are nearly impossible to obtain in practice. To this end, [135, 149, 229] study end-to-end trajectory forecasting directly from raw sensor data, and propose an evaluation setup for end-to-end forecasting models using the aforementioned ADE and FDE at fixed recall thresholds, *i.e.*, ADE/FDE at 60% or 90%. This evaluation setting has two major short-comings:

Evaluated only on a subset of matched detections. A large number of agents are not moving and prediction of their future positions is trivial. Models that rank stationary objects higher than moving objects can obtain higher forecasting performance by specializing on trivial predictions. We provide empirical evidence for this in Sec. 7.5.1 and show that such recall-based metrics can be "gamed" using a simple constant position prediction model.
No penalty for false positives. The current evaluation protocol detaches the inter-linked tasks of detection and forecasting. As a consequence, models can predict an arbitrary number of forecasts that are not anchored to any detections. In other words, this approach does not penalize *false forecasts*, *i.e.*, forecasts not anchored to any detection, and *missed forecasts* as commonly characterized by the miss rate.

7.3.2 Average Precision is All You Need

Average Precision (AP). AP is defined as the area under the precision-recall curve [60], commonly averaged over multiple spatial overlap thresholds [139]. To compute AP we first determine the set of true positives (TP) and false positives (FP) to evaluate precision and recall. In standard object detection, TPs are considered to be successful matches between model predictions and ground-truth, typically determined based on 2D/3D intersection-over-union (IoU) [60] or distance from the object center [22] in the reference image or LiDAR point cloud, respectively. We can extend AP for joint detection and forecasting by (a) evaluating detection accuracy on the current frame or (b) evaluating detection accuracy T seconds into the future. However, (a) completely ignores forecasts and (b) doesn't ensure that future trajectories are correctly associated to the right current detection.

Forecasting Average Precision (AP_f). For the task of joint detection and forecasting, all future forecasts need to be anchored to objects, present (and detected) in $\mathcal{S}_{t_{obs}}$. A robust metric must correctly penalize trajectories with correct first frame detections and incorrect forecasts (false forecasts), and trajectories with incorrect first frame detections (missed forecasts).

To characterize both types of errors, we define a true positive with reference to the current frame t_{obs} if there is a positive match in both the current timestamp (t_{obs}) *and* the future (final) timestep $t_{obs} + T$. Otherwise, a forecast is considered to be a false positive. A successful match in the current frame is determined based on the distance from the center, averaged over distance thresholds of $\{0.5, 1, 2, 4\}$m [22]. Similarly, a successful match in the final timestep is determined based on the distance from object center, averaged over distance thresholds of $\{1, 2, 4, 8\}$m respectively. In contrast to ADE @ Recall % and FDE @ Recall %, this evaluation setting (i) takes all detections (not just true positives) into account, and (ii) penalizes missed forecasts (typically characterized by the miss-rate).

Forecasting Mean Average Precision (mAP_f). *Forecasting AP* is evaluated on the full set of detections and cannot be "gamed" by a simple constant position model. However, we note that the data itself is imbalanced: over 60% of cars in the nuScenes dataset are parked, and are therefore stationary. To this end, we define three sub-classes: *static car*, *linearly moving car* and *non-linearly moving car*. Computing sub-class AP_f can be difficult; we do not require forecasts to output sub-class labels, but assume all ground-truth objects have sub-class labels. We follow the protocol for large-vs-small object subclass evaluation from COCO [139], described further in the appendix. We then evaluate mAP_f as $\frac{1}{3}(AP_f^{\text{stat.}} + AP_f^{\text{lin.}} + AP_f^{\text{non-lin.}})$ to ensure our metric cannot be "gamed" by trivial forecasters, as well as discuss fine-grained analysis on the three cases separately. Similarly, mAP_{det} is evaluated as the average AP_{det} over the three sub-classes.

Metrics: Embracing Multiple Futures. As described, mAP_f would be suitable for evaluating forecasting for scenes with multiple future ground truth trajectories. However, this is not feasible in the practice when forecasting directly from historical sensor data. To this end, we adopt a top-K based forecasting evaluation [126, 163, 183], that does not penalize models for hypothesizing possible future trajectories anchored from a single detection. In particular, we first match predictions to ground-truth detections in t_{obs} and take the top-K highest ranked forecasts for each detection. Based on this set, we determine the best-matching forecast in terms of FDE and evaluate AP_f as explained above.

7.4 Forecasting as Future Object Detection

FutureDet addresses the forecasting problem by predicting the future locations of objects observed at t_{obs}. We can repurpose existing LiDAR detectors to predict object locations for T future (unobserved) LiDAR scans, for which ground truth supervision is given. We first describe our method and discuss our implementation based on the recently proposed CenterPoint LiDAR detector. [241].

Future object detection and forecasting are related tasks. Forecasting requires predicting consistent trajectories in every frame between the current

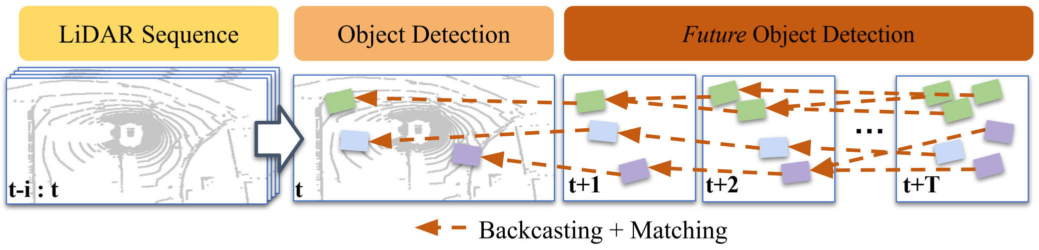


Figure 7.2: ***FutureDet.*** Based on an accumulated LiDAR sequence, *FutureDet* detects objects in the *current* frame t and in a *future* frames up to $t+T$. We then cast these future detections back-in-time (*i.e.*, *back-cast*) to the current-frame where they are matched to current frame object detections. Such matching of multiple future detections to current-frame detections is a a natural mechanism for a multi-future interpretation of the observed evidence.

frame and T future frames. To estimate forecasts from future detections we train our network to additionally estimate velocity offset vectors for every future detection. We do so for all frames between the current timestep and the final future timestep where the future detection occurs.

Backcasting vs. Forecasting. Fast and Furious [149] proposes a similar architecture that *forecasts* position offsets into the future directly from current-frame detections. Our method considers the inverse setup where we detect in both the current and future frames and predict offsets back in time. We posit that future object detection requires the network to learn forecasted feature representations [145], directly optimizing for future object positions. Our experimental evaluation and visual inspection confirm this intuition (Sec. 7.3).

Method: Embracing Multiple Futures. The task of forecasting is inherently ambiguous: while there are many plausible outcomes given an input trajectory, only one future is realized for training supervision and evaluation. Traditional forecasting methods and benchmarks facilitate multiple future predictions via top-K based evaluation, leveraging multiple-choice-loss [13] and generative models [3, 47, 48, 79, 101, 114, 125, 188] to learn a (possibly multi-modal [47, 48]) distribution over future trajectories. *FutureDet* allows for natural multi-future forecasting to emerge. We first point out that detection networks can be easily repurposed for future detection by giving target bounding boxes T seconds into the future. Since future objects are detected independently from current-frame detections, we posit that the network will produce multiple future detections for every object in the scene, effectively placing "multiple bets" where the objects may end up in the future. As all future detections are modeled by Gaussian heat maps, we implicitly obtain a multi-modal distribution over possible future locations (see Fig. 7.2).

Matching Multiple Forecasts. The task of forecasting requires all trajectories to be anchored to the set of object detections in the current (observed) LiDAR scan. For every future detection i, we backcast and compute the distance to each detection j from the previous timestep. For each i, we pick the best j (allowing for many-to-one matching). This framework naturally allows potentially multiple future forecasts to belong to each current timestep detection.

Ranking Multiple Forecasts. For all forecasts anchored at a single detection, we rank trajectories according to their forecasting score, derived using the detection confidence score of the last detection in a predicted trajectory. As shown in Table 7.2, we find there is a slight increase in performance between $K = 1$ and $K = 5$, indicating that better ranking strategies can further improve *FutureDet*.

Implementation. We train CenterPoint to detect objects in future scans. The underlying detection network simply thinks it's finding T times as many object classes (e.g., `cars` and `future-cars`) with additional regression offsets (analogous to existing velocity regressors). In addition, we repurpose the ground truth sampling (*a.k.a.* copy-paste) augmentation [254] to increase the diversity of training trajectories. This provides considerable improvement in linear and non-linear forecasting performance. We use the PyTorch toolbox to train all models for 20 epochs with the Adam optimizer and a one-cycle learning rate scheduler.

CenterPoint is already a one-frame forecaster. It detects objects and predicts one-frame future velocity vectors that are used as cues for tracking. It does this by accumulating T previous LiDAR scans and encodes the accumulated point cloud sequences using a VoxelNet [253] backbone. Such a tracker could be used as input to auto-regressive forecasting methods, *e.g.*, [189], however, we argue that we can use such a spatiotemporal representation to directly forecast.

CenterPoint models object locations as Gaussians. It does so by producing a 2D bird's-eye-view (BEV) heatmap, which models the continuous likelihood of detections at each point in the BEV space. Detections are obtained by finding local maxima in these heatmaps via non-maximum suppression (NMS). By reusing this representation for future detection, our detection heatmaps are effectively a forecast of a continuous likelihood field for the locations of objects. This continuous field naturally encodes the uncertainty of future detections, accounts for multi-modality, and provides a continuous representation for forecasting.

7.5 Experimental Evaluation

We conduct our experimental analysis on the nuScenes [22] dataset. As we are tackling end-to-end forecasting from sensor data, we do not follow the established evaluation protocol that provides ground-truth trajectories and HD maps as input (as explained in Sec. 7.3.1). First, we perform breakdown analysis of evaluation metrics proposed in [135] and our *forecasting mAP* by analyzing the performance of a simple constant position model (Sec. 7.5.1). After verifying that our proposed evaluation setting is not easily "gamable", as discussed in Sec. 7.3.1, we thoroughly ablate our model and compare its performance to other state-of-the-art methods (Sec. 7.5.2).

Repurposing NuScenes Tracking Dataset. nuScenes [22] recently introduced a large-scale multi-modal dataset recorded in Boston and Singapore. It provides 1000 twenty-second logs that are fully annotated with 3D bounding boxes. This work explicitly focuses on forecasting based on LiDAR data, obtained with a 32 beam LiDAR sensor recorded at 20 Hz, covering 360-degree view. We follow the official protocol and evaluate forecasting on the *car* class up to 3 seconds in the future. We evaluate forecasting performance on the *pedestrian* class in the appendix. As the test set is hidden, we follow [135] and conduct our analysis on the official validation split.

7.5.1 Metric Breakdown Analysis

In this section, we analyze different evaluation metrics by comparing a trivial constant position model to several state-of-the-art forecasting methods [135, 149, 189]. For the task of end-to-end forecasting from raw data, methods report both detection and forecasting confidence scores. For the simple constant position model, we threshold trajectories such that we only report those where the final position overlaps with the initial position (i.e the object is stationary) with high confidence. A good forecasting evaluation metric should indicate that the trivial constant position baseline is not a good predictor, as it only correctly predicts future locations of stationary objects and explicitly assumes a stationary world. Do existing metrics reveal this?

	$ADE@60$ (↓)	$FDE@60$ (↓)	$ADE@90$ (↓)	$FDE@90$ (↓)	$ADE\ avg.$ (↓)	$FDE\ avg.$ (↓)	$AP_f^{stat.}$(↑)	$AP_f^{lin.}$(↑)	$AP_f^{non-lin.}$(↑)	mAP_f(↑)
Constant Position (CP)	**0.38**	**0.63**	**0.48**	**0.76**	**0.37**	**0.64**	**66.3**	0	0	22.1
PnPNet [135]	0.58	0.93	0.68	1.04	-	-	-	-	-	-
PnPNet w/o Tracker [135]	0.69	1.09	0.75	1.14	-	-	-	-	-	-
Trajectron++ [189]	1.13	2.54	1.25	2.71	1.08	2.42	59.2	8.1	2.8	23.4
SPF2 [227]	-	-	-	-	1.04	1.04	-	-	-	-
Fast and Furious* (FaF) [149]	0.74	1.59	0.83	1.69	0.73	1.56	64.8	**22.2**	**7.5**	**31.5**

Table 7.1: Metric Breakdown Analysis: We compare our simple constant position model to state-of-the-art prediction models, highlighting differences among various proposed metrics. ADE/FDE based metrics measured at different recall rates favor our trivial constant position baseline over state-of-the-art methods [135, 149, 189]. Only our proposed *forecasting mAP* (mAP_f) favors state-of-the-art models over the constant position baseline. We report numbers for PnPNet [135] and SPF2 [227] from their respective papers. **Note: Lower ADE/FDE is better and higher AP_f is better.**

To answer this question, we report the results of our analysis in Table 7.1. We analyze the results using average and final displacement (ADE and FDE) errors at $\{60, 90\}\%$ recall [135], and a variant that averages results over all recall thresholds [227] (see Sec. 7.3.1 for a discussion of these metrics). Our trivial constant position baseline yields state-of-the-art results under the aforementioned evaluation settings. Current metrics are "gameable" by this trivial forecaster.

What about our *forecasting mAP* (Sec. 7.3.2)? We analyze these methods both through the lens of each motion class ($AP_f^{stat.}$, $AP_f^{lin.}$ and $AP_f^{non-lin.}$), and as an aggregate. First, we observe that the constant position model AP_f evaluated over static cars performs better than FaF*, a state-of-the-art end-to-end forecaster. However, when we evaluate on the subset of cars that are in motion, $AP_f^{lin.}$ and $AP_f^{non-lin.}$ confirms that our metric behaves as expected: we obtain $0AP$ with the constant position model, indicating that it fails to predict the motion of moving cars. On the other hand, FaF* obtains 7.5 $AP_f^{non-lin.}$, indicating that motion forecasting from the raw sensory data is a very challenging problem. We observe that Trajectron++ outperforms the constant position model for moving objects (8.1 $AP_f^{lin.}$), but does not reach the performance of the constant position model or FaF* on stationary objects.

A good metric should summarize the performance on the full set of cars, *i.e.*, in addition to predicting the motion of moving cars, a good model should correctly predict that parked cars are unlikely to move in the near future. This is achieved by our *forecasting mAP* that averages $AP_f^{stat.}$, $AP_f^{lin.}$ and $AP_{non-lin.}$. Our mAP_f ranks state-of-the-art FaF* (31.5 mAP_f) favourably over the constant position baseline (22.1 mAP_f), as expected. Based on this analysis, we are confident we have the right tools to analyze *FutureDet* thoroughly!

7.5.2 Ablation and Comparison to State-of-the-Art

	K=1								K=5							
	$AP^{stat.}$		$AP^{lin.}$		$AP^{non-lin.}$		mAP		$AP^{stat.}$		$AP^{lin..}$		$AP^{non-lin.}$		mAP	
	$\mathrm{AP_{det.}}$	$\mathrm{AP_f}$	$\mathrm{AP_{det.}}$	$\mathrm{AP_f}$	$\mathrm{AP_{det.}}$	$\mathrm{AP_f}$	$\mathrm{mAP_{det.}}$	$\mathrm{mAP_f}$	$\mathrm{AP_{det.}}$	$\mathrm{AP_f}$	$\mathrm{AP_{det.}}$	$\mathrm{AP_f}$	$\mathrm{AP_{det.}}$	$\mathrm{AP_f}$	$\mathrm{mAP_{det.}}$	$\mathrm{mAP_f}$
Detection + Constant Velocity	**70.3**	**66.0**	65.8	21.2	90.0	6.5	**75.4**	31.2	70.3	66.0	65.8	21.2	90.0	6.5	**75.4**	31.2
Detection + Forecast (*c.f.* [149])	69.1	64.7	**66.1**	22.2	86.3	7.5	73.8	31.5	69.1	64.7	**66.1**	22.2	86.3	7.5	73.8	31.5
Trajectron++ [189]	**70.3**	59.2	65.8	8.1	90.0	2.8	**75.4**	23.4	70.3	61.7	65.8	9.8	90.0	4.3	75.4	25.3
FutureDet	70.1	65.5	62.9	**24.9**	91.8	**10.1**	74.9	**33.5**	70.1	67.3	62.9	27.7	91.7	11.7	74.9	35.6
FutureDet-PointPillars	70.1	64.1	63.4	24.8	**92.4**	9.6	**75.4**	32.8	**70.7**	**67.5**	63.4	**28.8**	**92.0**	**11.9**	**75.4**	**36.1**
FutureDet + Map	70.2	65.5	62.7	24.3	91.7	9.4	74.9	33.1	70.2	**67.5**	62.7	27.1	91.7	11.0	74.9	35.2

Table 7.2: Joint car detection and forecasting evaluation on nuScenes. We adopt top-K evaluation for forecasting and evaluate under two settings of $K = 1$ and $K = 5$ (for forecasting only). We further breakdown the performance of each model by examining the detection AP ($AP_{det.}$) and forecasting AP (AP_f) on static, linear, and non-linearly moving sub-categories. First, we find that methods that are trained to detect objects in the current frame have higher overall $AP_{det.}$ (Detection + Constant Velocity, row 1), while methods that are trained to detect objects in future frames have higher overall AP_f (c.f. *FutureDet*, row 4), which is expected by design. For forecasting, surprisingly, Trajectron++ (row 3) is outperformed by constant velocity predictions (row 1), suggesting that this is indeed a challenging problem and constant velocity is a strong baseline. *FutureDet* consistently outperforms other baselines on non-linear trajectories. Notably, for $K = 5$, we improve the non-linear object forecasting accuracy by 4% over FaF*. *FutureDet* trained with a PointPillars backbone provides modest improvement across metrics, and performs best overall.

After confirming that our proposed *forecasting mAP* is a suitable metric for joint object detection and forecasting, we compare *FutureDet* to a number of baselines and two state-of-the-art methods.

Detection + Constant Velocity. We begin with an surprisingly simple, yet strong baseline which is often overlooked in forecasting literature. This baseline takes the detections, as well as the estimated velocity from our CenterPoint detector [241], and simply extrapolated forecasts as if objects are moving with constant velocity. Since CenterPoint is such a strong detector, this baseline produces strong results. Most of the ground-truth objects are approximately moving with a constant velocity, either moving directly forward or are stationary. We expect this model to under-perform on non-linear trajectories.

Detection + Forecast (*FaF**, c.f. [*149*]). This variant predicts a different velocity offset at every time step into the future for each detection, and derives trajectories by integrating velocities in the forward direction. This is precisely what Fast and Furious (FaF) does [149]. For an apples-to-apples comparison, we re-implement FaF using a CenterPoint-backbone and denote this model as FaF*. This method predicts a single trajectory per detection.

Trajectron++. We compare all of the aforementioned variants and ablations of our method to the state-of-the-art auto-regressive trajectory prediction model, Trajectron++ [189]. This model is indicative of current state-of-the-art approaches for the traditional forecasting task where ground truth tracks are given. With this comparison, we wish to outline how the standard three-stage detection-tracking-forecasting approach compares with our end-to-end forecasting method. To construct this baseline, we begin with off-the-shelf detection and tracking results from CenterPoint [241]. CenterPoint performs tracking using the velocity offset estimates to match detections in each frame using a greedy matching between the current frame detections and previous frame detections. Trajectron++ then takes these predicted trajectories as input for forecasting.

FutureDet. . We detect objects directly in future frames and backcast these future detections to the reference frame. Intuitively, the advantage of this variant over simple forecasting (FaF) is in that it encourages the network to learn a better feature representation for forecasting by placing ”multiple bets” on the future position of objects in the current frame. As shown in Figure 7.2, this method naturally allows for a multiple-future interpretation of the observed sensory data (as discussed in Sec. 7.4). In Figure 7.3, we show qualitatively that our method can represent multiple futures. We note that the highest confidence future trajectories looks like constant velocity predictions as the training data is biased towards static and linearly moving objects. *FutureDet* is able to learn road geometries without map information, as indicated by the curved trajectories.

Discussion. We compare the results of the aforementioned variants to *FutureDet* as well as Trajectron++ [189] in Table 7.2. First, we notice that moving object forecasting under our end-to-end setting is a challenging problem — none of the methods we study have high AP_f, suggesting the need for the community to focus on this problem. Second, despite the constant velocity model being conceptually simple, it performs on par with our FaF

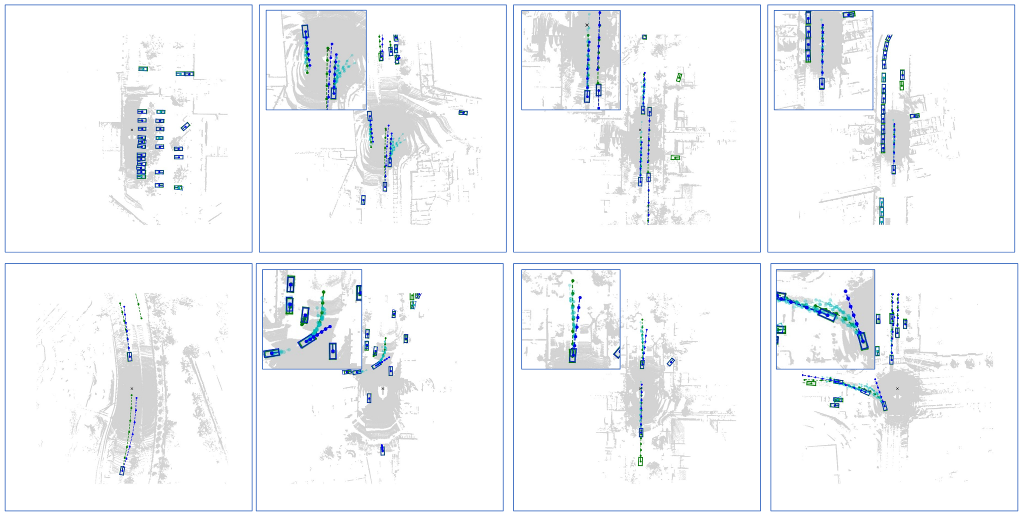


Figure 7.3: We qualitatively evaluate forecasts from *FutureDet*. We denote ground-truth trajectories with **green** and multiple future predictions with **blue** for the highest confidence forecast and **cyan** for the remaining multiple-future predictions. Since we repurpose CenterPoint, a state-of-the-art detector, current frame detection performs well. Often, our model predicts that moving objects may be moving with constant velocity with high confidence. Given the data bias, where most objects are either stationary or are moving with constant velocity, this is a reasonable output. We highlight the multiple-future detection output in the top left.

re-implementation and improves on Trajectron++ by +7.8 mAP_f. Unfortunately, this constant velocity baseline is usually under-emphasized in the literature. We argue here that it still serves as an important baseline. The poor performance of Trajectron++ might also hint that performing direct end-to-end forecasting is advantageous over a three-stage approach of detection-tracking-forecasting, where errors can easily compound. *FutureDet* takes a different approach compared to existing methods. Our method improves upon all other baselines in terms non-linear object AP_f and the motion category-averaged mAP_f (our primary metric). In addition, this multi-future interpretation also allows the performance to be improved in the $K = 5$ evaluation, where the forecast with minimum FDE from the top 5 ranked forecasts for each detection is evaluated. Note that for *FutureDet* AP_f^{static}, $K = 1$ results slightly decrease because bundling multiple trajectory estimates into one multi-future prediction for a single object reduces recall. However this is more than made up for in the increase in performance for detection and forecasting moving objects at $K = 5$. We train a version of our model with road masks as an additional input channel into the BEV feature representation (after the sparse-voxel backbone). This brings very little change to the results. We hypothesize that adding the map information does not provide additional information as it can be easily be learnt from the raw LiDAR input. However, further exploration is required to evaluate how to best fuse map information.

7.6 Conclusion

This paper presents a new end-to-end method for trajectory forecasting directly from LiDAR sensor data. Our proposed *FutureDet* is a natural forecasting-by-detection framework that allows for a multi-future interpretation of the observed evidence and establishes a new state-of-the-art. We provide thorough analysis of existing evaluation metrics for end-to-end forecasting and reveal that they can be gamed by a simple constant position model. To this end, we propose a new set of evaluation metrics based on the average precision metric that comprehensively evaluates joint detection and forecasting performance. This allows us to conduct a thorough analysis that reveals that a constant velocity model is a surprisingly strong baseline that should be considered in future forecasting work.

Limitations. As we do not explicitly enforce diverse trajectory generation, many of our multiple-futures are closely clustered. While *FutureDet* presents the first method for end-to-end forecasting from raw sensory data, capable of multi-future predictions, generation of diverse, multi-modal predictions remains an open challenge.

7.A Appendix

7.A.1 Examining *FutureDet*'s Predictions

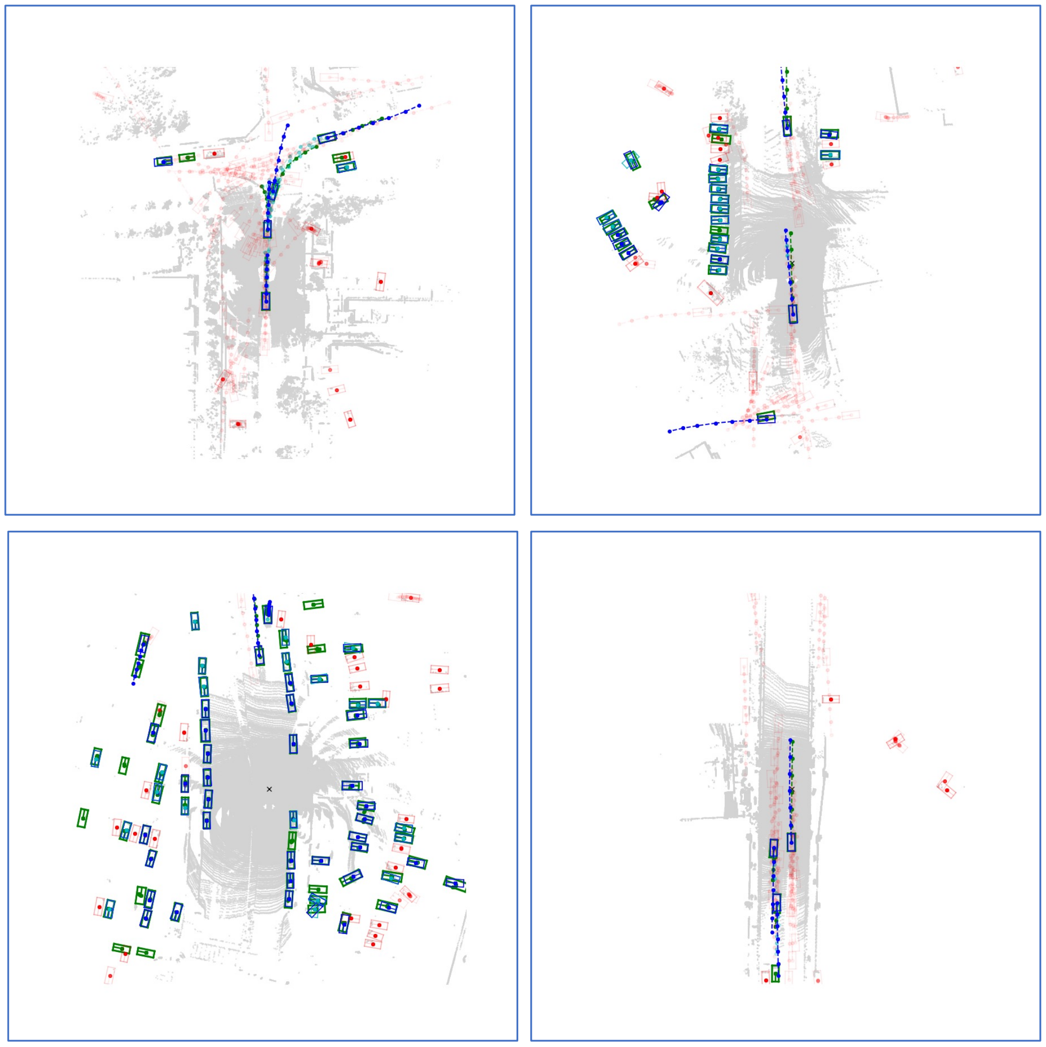


Figure 7.4: The raw output of *FutureDet* includes many false positive detections and forecasts (shown in **red**). Further post-processing is required to leverage the output of our end-to-end model in further downstream tasks.

One of the challenges of forecasting from raw sensor data is appropriately handling false positive detections and forecasts. The standard forecasting setup allows us to build models in isolation from other factors. However, the standard assumption of having perfect input trajectories is not feasible in practice as it critically depends on *perfect* object tracks (and by extension perfect detections) as inputs, which are nearly impossible to obtain in practice. As seen in Figure 7.4, the raw output of *FutureDet* makes it challenging to use in practice. Intuitively, training a model to make "multiple bets" about the position of objects may induce more false positives. Further post-processing is required to leverage the output of our end-to-end model in further downstream tasks.

7.A.2 Evaluating Pedestrian Forecasting

In this section, we evaluate pedestrian forecasting on the nuScenes dataset. Forecasting pedestrian movement can be considerably more challenging than car forecasting because pedestrians are more dynamic. Given our 3 second forecasting horizon, pedestrians typically do not move very far from their initial position. As a result, we define tighter match thresholds for pedestrian forecasting. A successful match in the current frame is determined based on the distance from the center, averaged over distance thresholds of $\{0.125, 0.25, 0.5, 1\}m$. A successful match in the final timestep is determined based on the distance from center, averaged over distance thresholds of $\{0.25, 0.5, 1, 2\}m$ respectively. We highlight the results of pedestrian forecasting in Table 7.3.

We see that *FutureDet* performs the best overall, with 26.9 mAP_f. Looking to Figure 7.5, it is clear that pedestrian detections are tightly clustered together, making back-casting less effective overall. We also find that many of the predicted multiple-futures are very similar to one another, indicating that the model is not able to model dynamic pedestrian futures. However, *FutureDet* still consistently improves over FaF* by 1% on AP_f metrics.

We train a version of our model with road masks as an additional input channel into the BEV feature representation (after the sparse-voxel backbone). This brings very little change to the results. We hypothesize that adding the map information does not provide additional information. However, further exploration is required to evaluate how to best fuse LiDAR and map information.

7.A.3 *FutureDet* Architecture

In this section, we further describe the implementation details of *FutureDet*. Specifically, we focus on the detector head architecture, and the sampling

	K=1								K=5							
	$AP^{stat.}$		$AP^{lin.}$		$AP^{non-lin.}$		mAP		$AP^{stat.}$		$AP^{lin..}$		$AP^{non-lin.}$		mAP	
	$\mathrm{AP_{det.}}$	$\mathrm{AP_f}$	$\mathrm{AP_{det.}}$	$\mathrm{AP_f}$	$\mathrm{AP_{det.}}$	$\mathrm{AP_f}$	$\mathrm{mAP_{det.}}$	$\mathrm{mAP_f}$	$\mathrm{AP_{det.}}$	$\mathrm{AP_f}$	$\mathrm{AP_{det.}}$	$\mathrm{AP_f}$	$\mathrm{AP_{det.}}$	$\mathrm{AP_f}$	$\mathrm{mAP_{det.}}$	$\mathrm{mAP_f}$
Detection + Constant Velocity	**55.1**	33.3	73.5	27.8	96.9	12.4	**75.2**	24.5	**55.1**	33.3	73.5	27.8	96.9	12.4	**75.2**	24.5
Detection + Forecast (*c.f.* [149])	53.7	**35.0**	**73.9**	30.8	**97.2**	13.3	74.9	26.4	53.7	35.0	**73.9**	30.8	**97.2**	13.3	74.9	26.4
Trajectron++ [189]	**55.1**	16.4	73.5	7.8	96.9	5.2	**75.2**	9.8	55.1	18.1	73.5	9.0	96.9	6.9	**75.2**	11.3
FutureDet	53.1	33.3	72.4	**32.6**	95.3	**14.7**	73.6	**26.9**	53.1	**35.1**	72.4	**34.0**	95.2	**15.0**	73.6	**28.0**
FutureDet-PointPillars	41.0	20.7	69.1	29.8	93.3	13.3	67.8	21.3	41.0	22.9	69.2	31.0	93.1	13.5	67.7	22.5
FutureDet + Map	52.4	33.0	71.8	32.0	95.3	14.4	73.2	26.5	52.4	34.8	71.8	33.4	95.2	14.8	73.2	27.7

Table 7.3: Joint pedestrian detection and forecasting evaluation on nuScenes. We adopt top-K evaluation for forecasting and evaluate under two settings of $K = 1$ and $K = 5$ (for forecasting only). We further breakdown the performance of each model by examining the detection AP ($AP_{det.}$) and forecasting AP (AP_f) on static, linear, and non-linearly moving sub-categories. Note that since pedestrians have smaller displacement over a 3 second forecasting horizon, we tighten the match thresholds as described above. *FutureDet* performs the best, improving over FaF* by 0.5 mAP_f. As with car forecasting, FaF* and the constant velocity baseline beat Trajectron++ by 14.4 % and 16.6 % mAP_f respectively. Notably, training with a PointPillars backbone dramatically reduces *FutureDet* performance on all metrics. In addition, we find that using a road mask does not significantly change the performance of *FutureDet*, indicating that the model might already be reasoning about spatial context.

strategy used to improve nonlinear trajectory forecasting.

Recurrent Features. We re-purpose CenterPoint for our implementation of *FutureDet*. However, CenterPoint is designed to detect objects in the current frame. It uses a shared feature representation for all classes. Although this effectively captures object spatial location, it does not allow for a robust representation of forecasted features. Specifically, since *FutureDet* detects `cars` and `future-cars`, we expect that the features required to detect these temporally offset classes should be different. To this end, we allow the model to learn a shallow network that transforms current features into future features as shown in Figure 7.6.

Trajectory Sampler. The distribution of static, linear, and non-linear trajectories in the nuScenes dataset is unbalanced. Since most cars are parked, we find that 60% of the trajectories are static. In order to address this data imbalance, we leverage copy-paste augmentation proposed by [254] to oversample linear and nonlinear trajectories during training. Importantly, we ensure that our copy-paste augmentation samples at the trajectory level, instead of at the class level, allowing consistent augmented trajectories across all detection heads (i.e. classes).

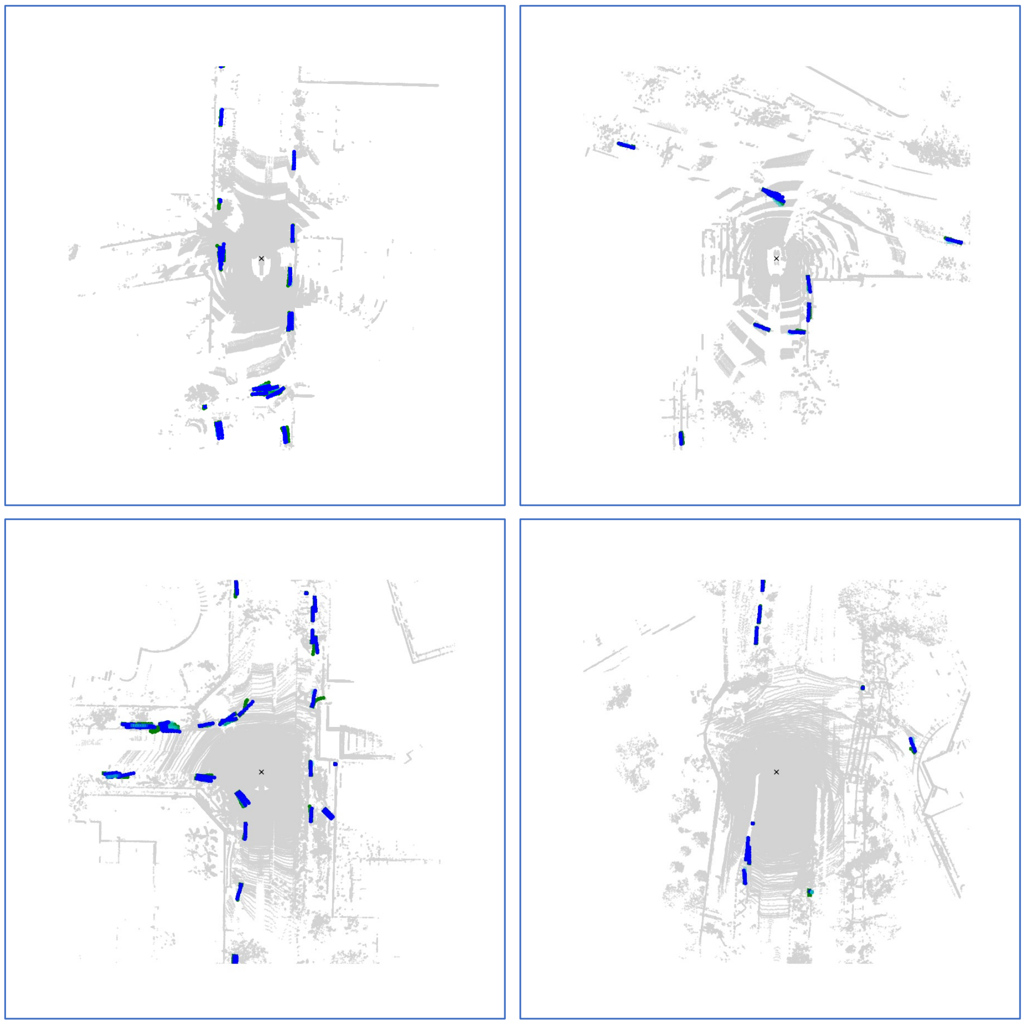


Figure 7.5: We qualitatively evaluate pedestrian forecasts from *FutureDet* (we denote the ground-truth trajectories with **green** and multiple future predictions with **blue** for the highest confidence forecast and **cyan** for the remaining future predictions). Pedestrian forecasting is more difficult than car forecasting due to the dynamic movement of pedestrians. *FutureDet* struggles to accurately forecast pedestrians because they often travel in crowds. This makes it difficult to accurately detect and forecast individual pedestrian motion. Often, the predicted multiple futures are all linearly moving, and are often similar to each other.

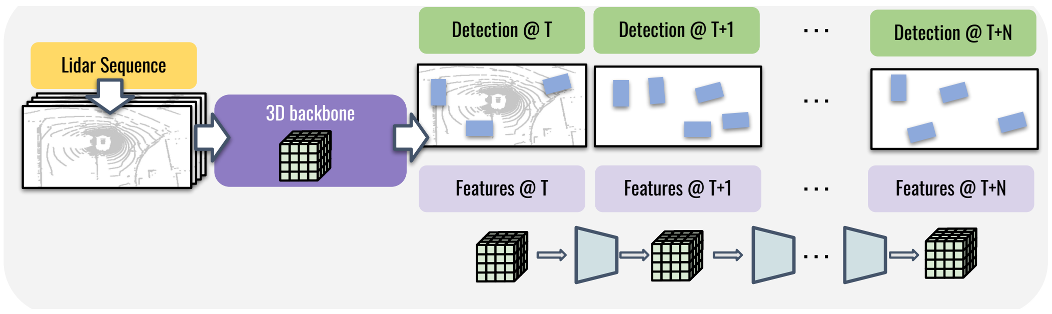


Figure 7.6: *FutureDet*'s detector head architecture adapts CenterPoint's architecture for the task of forecasting. Importantly, CenterPoint shares one set of features for all classes (i.e. cars, trucks, pedestrians, etc.). Since we adapt the architecture to forecast `cars` and `future-cars`, the single shared feature may not be able to effectively model long-term forecasting. To this end, we allow the model to learn a shallow network that transforms current features into future features.

7.A.4 Computing Motion Subclass AP

Computing subclass average precision is straightforward in principle if both predictions and ground-truth have subclass labels; one can simply treat the sub-class as a class and apply standard precision-recall metrics. In our case, predictions do not come with a subclass label. Instead, we match predictions to ground-truth at a class-level, and assign the ground-truth sub-class to the true positive matched predictions. However, this will not produce any sub-class labels for false positive predictions (that are unmatched). Instead, the metric evaluation code *derives* sub-class labels for false positive predictions, by applying the same logic used to derive sub-class labels for the ground-truth. We follow this procedure as it is also used to produce small-vs-large sub-class precision-recall metrics for standard detection toolkits [139]. Finally, although we use the language of sub-classes, our formalism can apply to any attributes associated with a detection.

We derive the subclass label as a function of the (ground truth or predicted) trajectory. For each trajectory, we first compute the IoU between bounding boxes at the first and last timestep. If the IoU is greater than 0, this trajectory is considered to be static. Next, using the velocity of the first timestep bounding box, we apply a constant velocity forecast to the initial position to compute a target box. If the IoU between the last timestep box and target box is greater than 0, this trajectory is considered to be linear. All trajectories that are not classified as static or linear as considered to be non-linear trajectories.

7.A.5 Broader Impact

Autonomous agents will play an important role in the automation of tasks that can be considered unsafe (*e.g.*, due to a high number of traffic accidents). Forecasting is at the heart of autonomous vehicle navigation: safe navigation necessitates motion prediction of surrounding agents to ensure driving safety. By leveraging LiDAR sensory data to accomplish this task, we can better understand world geometry and dynamics. Moreover, establishing the proper metrics, particularly considering the performance of moving and static car trajectories, is essential for building safe embodied robotics systems.

Chapter 8

Conclusion and Future Work

This thesis provides principled evaluation frameworks and novel constraint-aware solutions for various computer vision problems under resource constraints. By fully embracing resource constraints into design principles, we revisit well-established problems through novel perspectives, and come with up with creative algorithms for efficient visual understanding.

For evaluation frameworks, we introduce budgeted training, a formal setting to study training strategies that optimize for a given computation budget. Next, we propose streaming accuracy to evaluate latency and accuracy coherently with a single metric for real-time online perception. More broadly, building upon this metric, we introduce the streaming perception meta-benchmark that is applicable to any single-frame understanding tasks. Lastly, we propose forecasting AP to evaluate end-to-end detection, tracking and forecasting, eliminating the gameability in existing true-positive-only metrics.

For constraint-aware solutions, we propose a budget-aware learning rate schedule to effectively optimize training for a given budget. We propose dynamic scheduling and asynchronous forecasting for streaming perception. Additionally, we propose biologically-inspired attentional warping for 2D object detection, and discusses its future extension to arbitrary image-based tasks. We also propose a progressive distillation approach for learning lightweight detectors from a sequence of teacher models. To complete the perception stack, we propose future object detection with backcasting for end-to-end detection, tracking, and forecasting.

In the following subsections, I discuss several future directions based on the above thesis projects. These directions include online and streaming forecasting as an extension to streaming perception, task-agnostic attentional warping as an extension to the detection-specific foveated image magnification, and multi-model 3D detection as alternative approach to far-field 3D detection.

8.1 Online and Streaming Forecasting

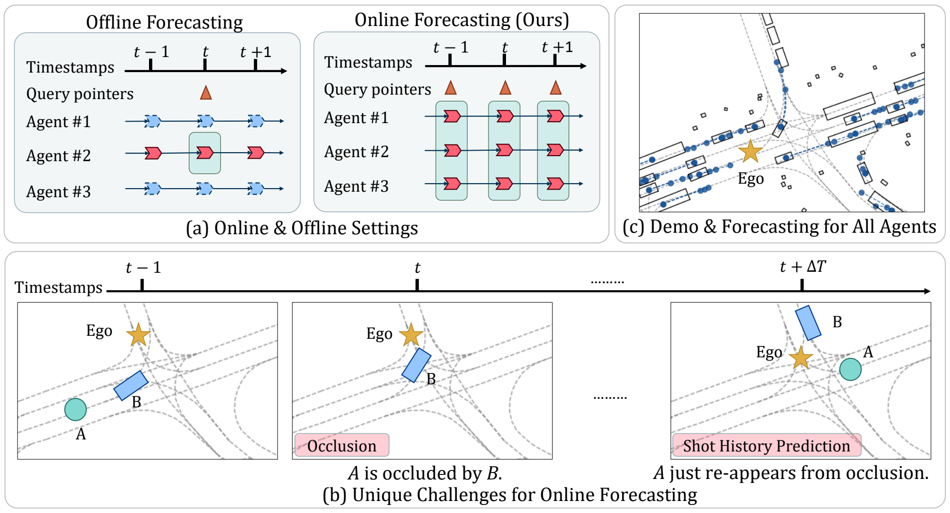

Offline Forecasting
Online Forecasting (Ours)
Timestamps
Query pointers
Agent #1
Agent #2
Agent #3
$t-1$
t
$t+1$
(a) Online & Offline Settings
Ego
(c) Demo & Forecasting for All Agents
$t+\Delta T$
A
B
Occlusion
Shot History Prediction
A is occluded by *B*.
A just re-appears from occlusion.
(b) Unique Challenges for Online Forecasting

Figure 8.1: Overview of our proposed online forecasting project. (a) **Setting comparison.** Embodied perception operates under an *online* setting, where motion forecasting module needs to concern every agent on continuous timestamps. However, current benchmarks are mainly *offline* as they concentrate on a small subset of agents on a set of temporally discontinuous timestamps. The contrast motivates us to set up a benchmark for "online forecasting". (b) **Unique questions for online forecasting**. Online forecasting reveals additional challenges. For example, agent A experiences occlusion (frame t) and de-occlusion (frame $t+\Delta T$) during its lifetime and raises the challenge of forecasting for occluded agents (frame t) and forecasting with short history observation (frame $t+\Delta T$). (c) A demo showing what the expected predictions are for *all* agents in the scene. For clarity, we only show the top-3 predictions (blue) for moving agents.

In streaming perception, we build a meta-benchmark by simply querying the state of the world at the current time for every timestamps. If we query a future time instead of the current time, we arrive at a streaming forecasting benchmark, which evaluates the performance of full-stack algorithms with a focus on long-term motion forecasting. Since full-stack perception may be complex, we also consider a simplified version where we remove the real-time constraint, and this resulting version is equivalent to online motion forecasting.

Motion forecasting has been largely studied in an offline and snapshot-based setting. While such a setup promotes the field through simplification and standardization, it also sweeps many problems under the rug. In an online world, a forecasting algorithm needs to make not only correct, but also temporally coherent predictions. Since objects frequently appear and disappear, *occlusion reasoning* emerges as a new challenge for forecasting algorithms. As shown in Fig 8.1, we plan to construct the a benchmark for online forecasting: we require algorithms to continuously output fixed-horizon future trajectories for every agent given the perception results till the current time. Such a requirement for continuous and complete understanding aligns well with the nature of the problem, and is necessary for practical deployment. Implementation-wise, we can create the benchmark by re-purposing the publicly available Argoverse tracking dataset for online forecasting [29]. We hope to provide a platform and inspiration for future research into this under-explored yet crucial problem.

8.2 Learning to Zoom and Unzoom

While the thesis have shown that attentional warping improves algorithms efficiency, the specific form of warping prevents us to apply our warping construction to tasks with dense spatial outputs such as semantic segmentation (see Fig 4.3 for more details on the original warping formulation). We plan to design a novel invertible and differentiable warping. As long as the base model architecture contains some intermediate stage with spatially aligned features, we can "unwarp" at that point to revert any spatial deformations, and this is the case with most modern convolutional backbones. Therefore, the proposed warp is largely model- and task-agnostic, and also does not require any changes to the original loss formulation. We plan to demonstrate the versatility on a variety of tasks: 2D object detection with RetinaNet [138] and Faster-RCNN [180] on Argoverse-HD [29], semantic segmentation with PSPNet [251] on Cityscapes [42], and monocular 3D detection with FCOS3D [220] on nuScenes [22].

8.3 Fusion-Based Far-Field 3D Detection

Chapter 6 introduces a novel representation for far-field 3D detection based on LiDAR input. We observe that most autonomous vehicles also have RGB cameras equipped and their dense signal may be provide additional information to objects at a distance. Therefore, we plan to explore multi-modal 3D detection, which takes both LiDAR and RGB images as input. For this problem, we notice there are several issues in existing evaluation framework. Contemporary AV benchmarks such as nuScenes underemphasize this problem because they evaluate performance only up to a certain distance (50m). One reason behind this limitation is that obtaining far-field 3D annotations is difficult, particularly for LiDAR sensors that produce very few point returns for far-away objects. Lack of a good benchmark impoverishes the exploration of far-field detection methods, and even leaves the conventional wisdom unjustified that high-resolution RGB is more effective when one looks far enough "out". Towards a far-field detection benchmark, we plan to develop a method to find well-annotated scenes, and a better distance-aware metric for reliable far-field evaluation. We also plan to propose novel fusion-based solutions for far-field 3D object detection.

Bibliography

[1] Sungsoo Ahn, Shell Xu Hu, Andreas Damianou, Neil D Lawrence, and Zhenwen Dai. Variational information distillation for knowledge transfer. In *CVPR*, 2019. 104

[2] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *CVPR*, June 2016. 155, 158, 160

[3] Javad Amirian, Jean-Bernard Hayet, and Julien Pettré. Social ways: Learning multi-modal distributions of pedestrian trajectories with gans. In *Conference on Computer Vision and Pattern Recognition Workshop*, 2019. 155, 158, 163

[4] Khalid Ashraf, Bichen Wu, Forrest N Iandola, Mattthew W Moskewicz, and Kurt Keutzer. Shallow networks for high-accuracy road object-detection. *arXiv preprint arXiv:1606.01561*, 2016. 127

[5] Mehmet Aygün, Aljoša Ošep, Mark Weber, Maxim Maximov, Cyrill Stachniss, Jens Behley, and Laura Leal-Taixé. 4d panoptic lidar segmentation. In *CVPR*, 2021. 158

[6] Olivier Bachem, Mario Lucic, and Andreas Krause. Practical coreset constructions for machine learning. *arXiv preprint arXiv:1703.06476*, 2017. 9, 26

[7] Marko Bacic. On hardware-in-the-loop simulation. In *IEEE Conference on Decision and Control*, 2005. 40

[8] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, J. Gall, and C. Stachniss. Towards 3D LiDAR-based semantic scene understanding of 3D point cloud sequences: The SemanticKITTI Dataset. *IJRR*, 40(8-9):959–967, 2021. 158

[9] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. *ACM SIGGRAPH computer graphics*, 26(2):35–42, 1992. 79, 80

[10] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, 2009. 108

[11] Philipp Bergmann, Tim Meinhardt, and Laura Leal-Taixé. Tracking without bells and whistles. In *ICCV*, 2019. 47, 68, 77

[12] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *ICIP*, 2016. 39, 66

[13] Apratim Bhattacharyya, Bernt Schiele, and Mario Fritz. Accurate and diverse sampling of sequences based on a "best of many" sample objective. In *CVPR*, 2018. 155, 163

[14] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020. 77, 94

[15] Mark Boddy and Thomas L Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 1994. 36

[16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Miguel Pozuelo Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Junbo Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. 8

[17] Léon Bottou. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91*, Nimes, France, 1991. EC2. 14, 19

[18] Léon Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998. revised, oct 2012. 8

[19] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60:223–311, 2018. 9, 14, 19, 29

[20] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *KDD*, 2006. 102, 105

[21] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 2016. 36

[22] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020. 75, 134, 136, 141, 155, 156, 158, 159, 160, 161, 165, 179

[23] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *AAAI*, 2018. 22

[24] Zhaowei Cai and Nuno Vasconcelos. Cascade R-CNN: Delving into high quality object detection. In *CVPR*, 2018. 56, 71, 129, 131

[25] Shengcao Cao, Xiaofang Wang, and Kris M Kitani. Learnable embedding space for efficient neural architecture compression. *ICLR*, 2019. 8, 22

[26] João Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. *CVPR*, pages 4724–4733, 2017. 11, 18, 31

[27] Yuning Chai, Benjamin Sapp, Mayank Bansal, and Dragomir Anguelov. Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction. *arXiv preprint arXiv:1910.05449*, 2019. 159

[28] Yuning Chai, Pei Sun, Jiquan Ngiam, Weiyue Wang, Benjamin Caine, Vijay Vasudevan, Xiao Zhang, and Dragomir Anguelov. To the point: Efficient 3d object detection in the range image with graph convolution kernels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009, 2021. 136

[29] Ming-Fang Chang, John W Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, and James Hays. Argoverse: 3D tracking and forecasting with rich maps. In *CVPR*, 2019. 42, 52, 53, 74, 75, 129, 135, 141, 155, 159, 160, 179

[30] Akshay Chawla, Hongxu Yin, Pavlo Molchanov, and Jose Alvarez. Data-free knowledge distillation for object detection. In *WACV*, 2021. 105

[31] Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. Learning efficient object detection models with knowledge distillation. In *NeurIPS*, 2017. 105, 107, 108

[32] Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. Hybrid task cascade for instance segmentation. In *CVPR*, 2019. 43, 50, 55, 71, 77, 86, 105, 106, 112

[33] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 71, 98, 101, 112

[34] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI*, 40:834–848, 2018. 13

[35] Qiang Chen, Yingming Wang, Tong Yang, Xiangyu Zhang, Jian Cheng, and Jian Sun. You only look one-level feature. *CVPR*, 2021. 86, 94, 97

[36] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *CVPR*, pages 1907–1915, 2017. 135

[37] Yuntao Chen, , Chenxia Han, Yanghao Li, Zehao Huang, Yi Jiang, Naiyan Wang, and Zhaoxiang Zhang. SimpleDet: A simple and versatile distributed framework for object detection and instance recognition. *JMLR*, 2019. 72

[38] Yu Cheng, D. Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *ArXiv*, abs/1710.09282, 2017. 75

[39] Jang Hyun Cho and Bharath Hariharan. On the efficacy of knowledge distillation. In *ICCV*, 2019. 103, 104, 108

[40] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *ICLR*, 2016. 20

[41] MMDetection3D Contributors. MMDetection3D: OpenMMLab next-generation platform for general 3D object detection. `https://github.com/open-mmlab/mmdetection3d`, 2020. 143, 153

[42] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016. 18, 26, 179

[43] Xiaodong Cun and Chi-Man Pun. Defocus blur detection via depth distillation. In *ECCV*, 2020. 105

[44] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. *arXiv preprint arXiv:1605.06409*, 2016. 105

[45] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 764–773, 2017. 78

[46] Xing Dai, Zeren Jiang, Zhao Wu, Yiping Bao, Zhicheng Wang, Si Liu, and Erjin Zhou. General instance distillation for object detection. In *CVPR*, 2021. 105, 112, 115

[47] Patrick Dendorfer, Sven Elflein, and Laura Leal-Taixé. Mg-gan: A multi-generator model preventing out-of-distribution samples in pedestrian trajectory prediction. In *ICCV*, 2021. 155, 158, 160, 163

[48] Patrick Dendorfer, Aljosa Osep, and Laura Leal-Taixe. Goal-gan: Multimodal trajectory prediction based on goal position estimation. In *ACCV*, 2020. 155, 158, 160, 163

[49] Patrick Dendorfer, Aljoša Ošep, Anton Milan, Konrad Schindler, Daniel Cremers, Ian Reid, and Stefan Roth Laura Leal-Taixé. Motchallenge: A benchmark for single-camera multiple target tracking. *IJCV*, 2020. 158

[50] P. Dendorfer, H. Rezatofighi, A. Milan, J. Shi, D. Cremers, I. Reid, S. Roth, K. Schindler, and L. Leal-Taixé. CVPR19 tracking and detection challenge: How crowded can it get? *arXiv:1906.04567*, 2019. 47

[51] Chunfang Deng, Mengmeng Wang, Liang Liu, Yong Liu, and Yunliang Jiang. Extended feature pyramid network for small object detection. *IEEE Transactions on Multimedia*, 2021. 105

[52] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In *ECCV*, 2021. 102

[53] Piotr Dollár, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: An evaluation of the state of the art. *PAMI*, 34, 2012. 98

[54] Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*, 2015. 71

[55] Felix Dräxler, Kambis Veschgini, Manfred Salmhofer, and Fred A. Hamprecht. Essentially no barriers in neural network energy landscape. In *ICML*, 2018. 11

[56] Dawei Du, Yuankai Qi, Hongyang Yu, Yi-Fan Yang, Kaiwen Duan, Guorong Li, Weigang Zhang, Qingming Huang, and Qi Tian. The unmanned aerial vehicle benchmark: Object detection and tracking. In *ECCV*, 2018. 53

[57] Simon S Du, Jason D Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. *arXiv preprint arXiv:1811.03804*, 2018. 11

[58] Simon S. Du, Jason D. Lee, Yuandong Tian, Barnabás Póczos, and Aarti Singh. Gradient descent learns one-hidden-layer cnn: Don't be afraid of spurious local minima. In *ICML*, 2018. 11

[59] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *TPAMI*, 2017. 36

[60] M. Everingham, L. Van Gool, C.K.I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (VOC) challenge. *IJCV*, 88(2):303–338, 2010. 159, 161

[61] Lue Fan, Xuan Xiong, Feng Wang, Naiyan Wang, and Zhaoxiang Zhang. Rangedet: In defense of range view for lidar-based 3d object detection. In *ICCV*, pages 2918–2927, 2021. 134

[62] Meherwar Fatima and Maruf Pasha. Survey of machine learning algorithms for disease diagnostic. *Journal of Intelligent Learning Systems and Applications*, 9(01):1, 2017. 8

[63] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. In *ICCV*, 2019. 71, 148

[64] Hamid Reza Feyzmahdavian, Arda Aytekin, and Mikael Johansson. An asynchronous mini-batch algorithm for regularized stochastic optimization. *IEEE Transactions on Automatic Control*, 61(12):3740–3754, 2016. 9

[65] Davi Frossard and Raquel Urtasun. End-to-end learning of multi-sensor 3d tracking by detection. In *ICRA*, 2018. 158

[66] Takashi Fukuda, Masayuki Suzuki, Gakuto Kurata, Samuel Thomas, Jia Cui, and Bhuvana Ramabhadran. Efficient knowledge distillation from an ensemble of teachers. In *Interspeech*, 2017. 105

[67] Jiyang Gao, Chen Sun, Hang Zhao, Yi Shen, Dragomir Anguelov, Congcong Li, and Cordelia Schmid. Vectornet: Encoding hd maps and agent dynamics from vectorized representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11525–11533, 2020. 159

[68] Mingfei Gao, Ruichi Yu, Ang Li, Vlad I Morariu, and Larry S Davis. Dynamic zoom-in network for fast object detection in large images. In *CVPR*, pages 6926–6935, 2018. 49, 78, 86, 96, 98

[69] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P. Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *NeurIPS*, 2018. 11

[70] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *CVPR*, 2012. 141, 158

[71] Thomas Gilles, Stefano Sabatini, Dzmitry Tsishkou, Bogdan Stanciulescu, and Fabien Moutarde. Home: Heatmap output for future motion estimation. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 500–507. IEEE, 2021. 159

[72] Rohit Girdhar, João Carreira, Carl Doersch, and Andrew Zisserman. A better baseline for AVA. *ArXiv abs/1807.10066*, 2018. 71

[73] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, pages 580–587, 2014. 77

[74] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Neural Information Processing Systems*, 2014. 158

[75] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. *ICLR*, 2019. 11, 21

[76] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 31

[77] Junru Gu, Chen Sun, and Hang Zhao. DenseTNT: End-to-end trajectory prediction from dense goal sets. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15303–15312, 2021. 159

[78] Jianyuan Guo, Kai Han, Yunhe Wang, Han Wu, Xinghao Chen, Chunjing Xu, and Chang Xu. Distilling object detectors via decoupled features. In *CVPR*, 2021. 105, 108, 112, 115

[79] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks. In *CVPR*, 2018. 155, 158, 160, 163

[80] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016. 102

[81] Moritz Hardt, Benjamin Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *ICML*, 2016. 11

[82] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. In *ICCV*, 2017. 8, 11, 13, 17, 18, 22, 42, 43, 52, 56, 71, 83, 105, 106, 110, 112

[83] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37:1904–1916, 2015. 77, 83

[84] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, pages 770–778, 2016. 8, 13, 16, 17, 85, 86, 106

[85] Kevin Healy, Luke McNally, Graeme D Ruxton, Natalie Cooper, and Andrew L Jackson. Metabolic rate and body size are linked with perception of temporal information. *Animal behaviour*, 86(4):685–696, 2013. 1

[86] Dirk Helbing and Péter Molnár. Social force model for pedestrian dynamics. *Physical Review E*, 1995. 158

[87] Byeongho Heo, Minsik Lee, Sangdoo Yun, and Jin Young Choi. Knowledge transfer via distillation of activation boundaries formed by hidden neurons. In *AAAI*, 2019. 105

[88] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *NeurIPS Workshop*, 2014. 102, 103, 104, 105

[89] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural computation*, 1997. 117

[90] Eric J Horvitz. *Computation and action under bounded resources*. PhD thesis, Stanford University, 1990. 36

[91] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets:

Efficient convolutional neural networks for mobile vision applications. *ArXiv abs/1704.04861*, 2017. 35

[92] Bo Yang Hsueh, Wei Li, and I-Chen Wu. Stochastic gradient descent with hyperbolic-tangent decay on classification. *WACV*, 2019. 11, 13, 16

[93] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, pages 7132–7141, 2018. 134, 140

[94] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11108–11117, 2020. 136

[95] Chen Huang, Simon Lucey, and Deva Ramanan. Learning policies for adaptive tracking with deep feature cascades. In *Proceedings of the IEEE international conference on computer vision*, pages 105–114, 2017. 78

[96] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. Snapshot ensembles: Train 1, get m for free. *ICLR*, 2017. 11

[97] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. *CVPR*, pages 2261–2269, 2017. 8, 11, 15, 17

[98] Rui Huang, Wanyue Zhang, Abhijit Kundu, Caroline Pantofaru, David A. Ross, Thomas A. Funkhouser, and Alireza Fathi. An lstm approach to temporal 3d object detection in lidar point clouds. In *ECCV*, 2020. 134

[99] Zehao Huang and Naiyan Wang. Like what you like: Knowledge distill via neuron selectivity transfer. *arXiv preprint arXiv:1707.01219*, 2017. 104

[100] Zeyi Huang, Yang Zou, Vijayakumar Bhagavatula, and Dong Huang. Comprehensive attention self-distillation for weakly-supervised object detection. In *NeurIPS*, 2020. 105

[101] Boris Ivanovic and Marco Pavone. The trajectron: Probabilistic multi-agent trajectory modeling with dynamic spatiotemporal graphs. In *ICCV*, 2019. 155, 158, 160, 163

[102] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *arXiv preprint arXiv:1506.02025*, 2015. 76, 78, 79, 80, 81, 83, 93, 96

[103] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM Multimedia*, pages 675–678, 2014. 13, 16

[104] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960. 45

[105] Will Kay, João Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Apostol Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset. *CoRR*, abs/1705.06950, 2017. 18

[106] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *ICLR*, 2017. 117

[107] Aleksandr Kim, Aljoša Ošep, and Laura Leal-Taix'e. Eagermot: 3d multi-object tracking via sensor fusion. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021. 155

[108] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015. 9, 12, 20, 27

[109] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014. 158

[110] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. Pointrend: Image segmentation as rendering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9799–9808, 2020. 78

[111] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983. 14

[112] Robert D. Kleinberg, Yuanzhi Li, and Yang Yuan. An alternative view: When does sgd escape local minima? In *ICML*, 2018. 11

[113] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *ICML*, 2019. 109

[114] Vineet Kosaraju, Amir Sadeghian, Roberto Martín-Martín, Ian Reid, Hamid Rezatofighi, and Silvio Savarese. Social-BiGAT: Multimodal trajectory forecasting using bicycle-gan and graph attention networks. In *Neural Information Processing Systems*, 2019. 155, 158, 163

[115] Robert J Kosinski. A literature review on reaction time. *Clemson University*, 10, 2008. 33

[116] Matej Kristan, Aleš Leonardis, Jiri Matas, Michael Felsberg, Roman Pflugfelder, Luka Čehovin Zajc, Tomas Vojir, Gustav Häger, Alan Lukežič, Abdelrahman Eldesokey, and Gustavo Fernandez. The visual object tracking VOT2017 challenge results, 2017. 36

[117] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 15

[118] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS*, 2012. 8, 158

[119] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom

Duerig, and Vittorio Ferrari. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *arXiv:1811.00982*, 2018. 8

[120] Xu Lan, Xiatian Zhu, and Shaogang Gong. Knowledge distillation by on-the-fly native ensemble. In *NeurIPS*, 2018. 108

[121] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *CVPR*, 2019. 132, 134, 135, 138, 142, 144, 153, 155, 158

[122] Adam M. Larson and Lester C. Loschky. The contributions of central versus peripheral vision to scene gist recognition. *Journal of vision*, 9 10:6.1–16, 2009. 75

[123] Laura Leal-Taixé, Gerard Pons-Moll, and Bodo Rosenhahn. Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker. In *ICCV Workshops*, 2011. 158

[124] Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *CVPR*, 2016. 102

[125] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher Bongsoo Choy, Philip H. S. Torr, and Manmohan Krishna Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. *CVPR*, 2017. 155, 158, 163

[126] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. Crowds by Example. *Comput. Graph. Forum*, 2007. 158, 160, 162

[127] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *NeurIPS*, 2018. 117, 118

[128] Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet S. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *JMLR*, 18:185:1–185:52, 2017. 8

[129] Mengtian Li, Yuxiong Wang, and Deva Ramanan. Towards streaming perception. In *ECCV*, 2020. 75, 76, 77, 85, 88, 90, 101, 110, 111, 116, 128, 130, 132, 143, 157, 159

[130] Mengtian Li, Ersin Yumer, and Deva Ramanan. Budgeted training: Rethinking deep neural network training under resource constraints. In *ICLR*, 2020. 53, 86, 113

[131] Quanquan Li, Shengying Jin, and Junjie Yan. Mimicking very efficient network for object detection. In *CVPR*, 2017. 105

[132] Xiaoxiao Li, Ziwei Liu, Ping Luo, Chen Change Loy, and Xiaoou Tang. Not all pixels are equal: Difficulty-aware semantic segmentation via deep layer cascade. In *CVPR*, pages 3193–3202, 2017. 78

[133] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *NIPS*, pages 5330–5340, 2017. 9

[134] Ming Liang, Bin Yang, Rui Hu, Yun Chen, Renjie Liao, Song Feng, and Raquel Urtasun. Learning lane graph representations for motion forecasting. In *European Conference on Computer Vision*, pages 541–556. Springer, 2020. 159

[135] Ming Liang, Bin Yang, Wenyuan Zeng, Yun Chen, Rui Hu, Sergio Casas, and Raquel Urtasun. Pnpnet: End-to-end perception and prediction with tracking in the loop. In *CVPR*, 2020. 155, 157, 159, 160, 165, 166

[136] Ji Lin, Chuang Gan, and Song Han. TSM: Temporal shift module for efficient video understanding. In *ICCV*, pages 7083–7093, 2019. 77

[137] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, pages 2117–2125, 2017. 77, 85, 106, 110, 134, 150

[138] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017. 33, 35, 51, 52, 71, 86, 94, 97, 105, 106, 110, 112, 179

[139] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014. 17, 36, 42, 52, 53, 93, 103, 104, 111, 161, 162, 175

[140] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *ICLR*, 2019. 8, 102

[141] Lanlan Liu and Jia Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. In *AAAI*, 2018. 78

[142] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. In *ECCV*, 2016. 33, 35, 51, 71, 77

[143] Yifan Liu, Ke Chen, Chris Liu, Zengchang Qin, Zhenbo Luo, and Jingdong Wang. Structured knowledge distillation for semantic segmentation. In *CVPR*, 2019. 105

[144] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *ICLR*, 2017. 11, 13, 16, 21, 27

[145] Pauline Luc, Camille Couprie, Yann LeCun, and Jakob Verbeek. Predicting future instance segmentations by forecasting convolutional features. In *ECCV*, 2018. 34, 73, 74, 163

[146] Alan Lukezic, Tomás Vojír, Luka Cehovin Zajc, Jiri Matas, and Matej Kristan. Discriminative correlation filter with channel and spatial reliability. In *CVPR*, 2017. 39

[147] Hao Luo, Wenxuan Xie, Xinggang Wang, and Wenjun Zeng. Detect or track: Towards cost-effective video object detection/tracking. In *AAAI*, volume 33, pages 8803–8810, 2019. 77

[148] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. *ICLR*, 2019. 9, 12, 27

[149] Wenjie Luo, Bin Yang, and Raquel Urtasun. Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net. In *CVPR*, 2018. 134, 155, 159, 160, 163, 165, 166, 167, 168, 173

[150] W. Ma, S. Wang, R. Hu, Y. Xiong, and R. Urtasun. Deep rigid instance scene flow. In *CVPR*, pages 1–9, 2019. 11

[151] Huizi Mao, Xiaodong Yang, and William J Dally. A delay metric for video object detection: What average precision fails to tell. In *ICCV*, 2019. 35

[152] Dmitrii Marin, Zijian He, Peter Vajda, Priyam Chatterjee, Sam Tsai, Fei Yang, and Yuri Boykov. Efficient segmentation: Learning downsampling near semantic boundaries. In *ICCV*, pages 2131–2141, 2019. 78, 93

[153] David McAllester, Bart Selman, and Henry Kautz. Evidence for invariants in local search. In *AAAI*, pages 321–326, 1997. 14

[154] Peter McLeod. Visual reaction time and high-speed ball games. *Perception*, 1987. 33

[155] Rakesh Mehta and Cemalettin Ozturk. Object detection at 200 frames per second. In *ECCV Workshop*, 2018. 105

[156] Gregory P Meyer, Ankit Laddha, Eric Kee, Carlos Vallespi-Gonzalez, and Carl K Wellington. Lasernet: An efficient probabilistic 3d object detector for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12677–12686, 2019. 136

[157] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant. In *AAAI*, 2020. 103, 104, 105

[158] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of convolution neural network advances on the imagenet. *Computer Vision and Image Understanding*, 161:11–19, 2017. 11

[159] Ravi Teja Mullapudi, Steven Chen, Keyi Zhang, Deva Ramanan, and Kayvon Fatahalian. Online model distillation for efficient video inference. In *ICCV*, 2019. 49

[160] Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19:1574–1609, 2009. 9

[161] Yaghout Nourani and Bjarne Andresen. A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, 31(41):8373, 1998. 14

[162] SeongUk Park and Nojun Kwak. Feed: Feature-level ensemble for knowledge distillation. *arXiv preprint arXiv:1909.10754*, 2019. 108

[163] S. Pellegrini, Andreas Ess, and L. Gool. Improving data association by joint modeling of pedestrian trajectories and groupings. In *ECCV*, 2010. 158, 160, 162

[164] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *ICML*, 2018. 22

[165] PyTorch. ImageNet training in PyTorch v1.0.1, 2019. Retrieved from `https://github.com/pytorch/examples/tree/master/imagenet` on Mar 4, 2019. 17

[166] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *CVPR*, 2017. 158

[167] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, pages 652–660, 2017. 135

[168] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017. 136

[169] Siyuan Qiao, Liang-Chieh Chen, and Alan Yuille. Detectors: Detecting objects with recursive feature pyramid and switchable atrous convolution. *arXiv preprint arXiv:2006.02334*, 2020. 77, 105, 106, 112, 131

[170] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source robot operating system. In *ICRA workshop on open source software*. Kobe, Japan, 2009. 48

[171] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, F. Durand, and Saman P. Amarasinghe. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2013. 75

[172] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *NeurIPS*, 2017. 109

[173] Piyush Rai, Hal Daumé, and Suresh Venkatasubramanian. Streamed learning: one-pass svms. In *IJCAI*, 2009. 8

[174] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-NET: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016. 102

[175] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *AAAI*, 2019. 22

[176] Adria Recasens, Petr Kellnhofer, Simon Stent, Wojciech Matusik, and Antonio Torralba. Learning to zoom: a saliency-based sampling layer for neural networks. In *ECCV*, pages 51–66, 2018. 76, 78, 79, 80, 81, 82, 83, 86, 93, 96, 97, 98, 101

[177] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *ICLR*, 2018. 9, 12, 20, 21, 24, 27

[178] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016. 33, 35, 51, 105

[179] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 77, 105

[180] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 77, 83, 85, 105, 131, 158, 179

[181] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 658–666, 2019. 83

[182] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951. 9, 10, 14, 29

[183] Alexandre Robicquet, Amir Sadeghian, Alexandre Alahi, and Silvio Savarese. Learning social etiquette: Human trajectory understanding in crowded scenes. In *ECCV*, 2016. 158, 160, 162

[184] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In *ICLR*, 2015. 104, 105

[185] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 115(3):211–252, 2015. 8, 17, 77

[186] Stuart Jonathan Russell and Eric Wefald. *Do the right thing: studies in limited rationality*. MIT press, 1991. 36

[187] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. 9

[188] Amir Sadeghian, Vineet Kosaraju, Ali Sadeghian, Noriaki Hirose, Hamid Rezatofighi, and Silvio Savarese. Sophie: An attentive GAN for predicting paths compliant to social and physical constraints. In *Conference on Computer Vision and Pattern Recognition*, 2019. 155, 158, 163

[189] Tim Salzmann, Boris Ivanovic, Punarjay Chakravarty, and Marco Pavone. Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data. In *ECCV*, 2020. 155, 160, 164, 165, 166, 167, 168, 173

[190] Bharat Bhusan Sau and Vineeth N Balasubramanian. Deep model compression: Distilling knowledge from noisy teachers. *arXiv preprint arXiv:1610.09650*, 2016. 104, 105, 108

[191] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *ICLR*, 2018. 9

[192] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013. 77

[193] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In *CVPR*, 2020. 155, 158

[194] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud. In *CVPR*, 2019. 136, 155, 158

[195] Shaoshuai Shi, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. From Points to Parts: 3D Object Detection from Point Cloud with Part-aware and Part-aggregation Network. *PAMI*, 2020. 155

[196] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental learning of object detectors without catastrophic forgetting. In *ICCV*, 2017. 105

[197] H. Simon. A behavioral model of rational choice. *Quarterly Journal of Economics*, 69:99–118, 1955. 1

[198] Leslie N. Smith. Cyclical learning rates for training neural networks. *WACV*, pages 464–472, 2017. 11

[199] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *CVPR*, 2020. 53, 75, 136, 141, 155, 158, 159, 160

[200] Pei Sun, Weiyue Wang, Yuning Chai, Gamaleldin Elsayed, Alex Bewley, Xiao Zhang, Cristian Sminchisescu, and Dragomir Anguelov. Rsn: Range sparse net for efficient, accurate lidar 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5725–5734, 2021. 136

[201] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2017. 11

[202] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, pages 1–9, 2015. 11

[203] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, pages 2818–2826, 2016. 11

[204] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. *CVPR*, pages 2815–2823, 2019. 1, 102

[205] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *CVPR*, pages 10781–10790, 2020. 77, 106, 116, 132, 143, 150

[206] Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. Searching efficient 3d architectures with sparse point-voxel convolution. In *European conference on computer vision*, pages 685–702. Springer, 2020. 136

[207] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *NeurIPS*, 2018. 108

[208] Russell H Taylor, Arianna Menciassi, Gabor Fichtinger, and Paolo Dario. Medical robotics and computer-integrated surgery. *Springer handbook of robotics*, pages 1199–1222, 2008. 8

[209] Chittesh Thavamani, Mengtian Li, Nicolas Cebron, and Deva Ramanan. Fovea: Foveated image magnification for autonomous navigation. In *ICCV*, 2021. 143

[210] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *ICCV*, 2019. 106, 112

[211] T. Tieleman and G. Hinton. RMSProp: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 2012. 12, 27

[212] Burak Uzkent and Stefano Ermon. Learning when and where to zoom with deep reinforcement learning. In *CVPR*, pages 12345–12354, 2020. 78

[213] V. Vanhoucke, A. Senior, and M. Mao. Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS*, 2011. 75

[214] Thomas Verelst and Tinne Tuytelaars. Dynamic convolutions: Exploiting spatial sparsity for faster inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2320–2329, 2020. 78

[215] Paul Voigtlaender, Michael Krause, Aljoša Ošep, Jonathon Luiten, Berin Balachandar Gnana Sekar, Andreas Geiger, and Bastian Leibe. MOTS: Multi-object tracking and segmentation. In *CVPR*, 2019. 53

[216] Jayakorn Vongkulbhisal, Phongtharin Vinayavekhin, and Marco Visentini-Scarzanella. Unifying heterogeneous classifiers with distillation. In *CVPR*, 2019. 104, 105

[217] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4: Scaling cross stage partial network. In *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, pages 13029–13038, 2021. 132

[218] Liwei Wang, Lunjia Hu, Jiayuan Gu, Yue Wu, Zhiqiang Hu, Kun He, and John Hopcroft. Towards understanding learning representations: To what extent do different neural networks learn the same representation. In *NeurIPS*, 2018. 109

[219] Tao Wang, Li Yuan, Xiaopeng Zhang, and Jiashi Feng. Distilling object detectors with fine-grained feature imitation. In *CVPR*, 2019. 105, 108

[220] Tai Wang, Xinge Zhu, Jiangmiao Pang, and Dahua Lin. Fcos3d: Fully convolutional one-stage monocular 3d object detection. *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 913–922, 2021. 179

[221] Wanwei Wang, Wei Hong, Feng Wang, and Jinke Yu. Gan-knowledge distillation for one-stage object detection. *IEEE Access*, 2020. 105

[222] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. *CVPR*, 2018. 11, 18

[223] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019. 136

[224] Yu-Xiong Wang, Deva Ramanan, and Martial Hebert. Growing a brain: Fine-tuning by increasing model capacity. In *CVPR*, pages 2471–2480, 2017. 9

[225] Longyin Wen, Dawei Du, Zhaowei Cai, Zhen Lei, Ming-Ching Chang, Honggang Qi, Jongwoo Lim, Ming-Hsuan Yang, and Siwei Lyu. DETRAC: A new benchmark and protocol for multi-object detection and tracking. *arXiv abs/1511.04136*, 2015. 53

[226] Xinshuo Weng, Jianren Wang, David Held, and Kris Kitani. 3D Multi-Object Tracking: A Baseline and New Evaluation Metrics. In *IROS*, 2020. 155, 158

[227] Xinshuo Weng, Jianren Wang, Sergey Levine, Kris Kitani, and Nick Rhinehart. Inverting the pose forecasting pipeline with SPF2: Sequential pointcloud forecasting for sequential pose forecasting. In *CoRL*, 2020. 157, 159, 166

[228] Xinshuo Weng, Yongxin Wang, Yunze Man, and Kris Kitani. Gnn3dmot: Graph neural network for 3d multi-object tracking with multi-feature learning. In *CVPR*, 2020. 155, 158

[229] Xinshuo Weng, Ye Yuan, and Kris Kitani. PTP: Parallelized tracking and prediction with graph neural networks and diversity sampling. *Robotics and Automation Letters*, 2021. 155, 159, 160

[230] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *NIPS*, pages 4148–4158, 2017. 12

[231] Ancong Wu, Wei-Shi Zheng, Xiaowei Guo, and Jian-Huang Lai. Distilled person re-identification: Towards a more scalable system. In *CVPR*, 2019. 108

[232] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *CVPR*, 2016. 102

[233] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017. 106, 129

[234] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *ICCV*, 2015. 11

[235] K. Yamaguchi, A.C. Berg, L.E Ortiz, and T.L. Berg. Who are you with and where are you going? In *CVPR*, 2011. 158

[236] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018. 132, 134, 136, 150

[237] Jing Yang, Brais Martinez, Adrian Bulat, and Georgios Tzimiropoulos. Knowledge distillation via softmax regression representation learning. In *ICLR*, 2021. 107

[238] Linjie Yang, Yuchen Fan, and Ning Xu. Video instance segmentation. In *ICCV*, 2019. 40, 53

[239] Yiming Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1:69–90, 2004. 151

[240] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *CVPR*, 2017. 105

[241] Tianwei Yin, Xingyi Zhou, and Philipp Krahenbuhl. Center-based 3d object detection and tracking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11784–11793, 2021. 132, 134, 155, 158, 162, 167, 168

[242] Zhichao Yin, Trevor Darrell, and Fisher Yu. Hierarchical discrete distribution decomposition for match density estimation. *CVPR*, 2019. 11

[243] Shan You, Chang Xu, Chao Xu, and Dacheng Tao. Learning from multiple teacher networks. In *KDD*, 2017. 105, 108

[244] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *CVPR*, June 2020. 76, 91

[245] Fei Yuan, Linjun Shou, Jian Pei, Wutao Lin, Ming Gong, Yan Fu, and Daxin Jiang. Reinforced multi-teacher selection for knowledge distillation. In *AAAI*, 2021. 104

[246] Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *ICLR*, 2017. 102, 104

[247] Chenrui Zhang and Yuxin Peng. Better and faster: Knowledge transfer from multiple self-supervised learning tasks via graph distillation for video classification. In *IJCAI*, 2018. 108

[248] Linfeng Zhang and Kaisheng Ma. Improve object detection with feature-based knowledge distillation: Towards accurate and efficient detectors. In *ICLR*, 2021. 105, 108, 112, 114, 115, 128, 129

[249] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018. 35

[250] Yang Zhang, Zixiang Zhou, Philip David, Xiangyu Yue, Zerong Xi, Boqing Gong, and Hassan Foroosh. Polarnet: An improved grid representation for online lidar point clouds semantic segmentation. In *CVPR*, 2020. 136, 158

[251] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. *CVPR*, pages 6230–6239, 2017. 13, 18, 26, 179

[252] Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. Tracking objects as points. In *ECCV*, 2020. 158

[253] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *CVPR*, pages 4490–4499, 2018. 135, 155, 164

[254] Benjin Zhu, Zhengkai Jiang, Xiangxin Zhou, Zeming Li, and Gang Yu. Class-balanced grouping and sampling for point cloud 3d object detection. *arXiv preprint arXiv:1908.09492*, 2019. 164, 173

[255] Xizhou Zhu, Jifeng Dai, Lu Yuan, and Yichen Wei. Towards high performance video object detection. In *CVPR*, pages 7210–7218, 2018. 77

[256] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results. In *CVPR*, 2019. 129

[257] Xizhou Zhu, Yujie Wang, Jifeng Dai, Lu Yuan, and Yichen Wei. Flow-guided feature aggregation for video object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 408–417, 2017. 77

[258] Xinge Zhu, Hui Zhou, Tai Wang, Fangzhou Hong, Yuexin Ma, Wei Li, Hongsheng Li, and Dahua Lin. Cylindrical and asymmetrical 3d convolution networks for lidar segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9939–9948, 2021. 136

[259] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI magazine*, 1996. 37, 48

[260] Shlomo Zilberstein and Abdel-Illah Mouaddib. Optimal scheduling of progressive processing tasks. *International Journal of Approximate Reasoning*, 2000. 37

[261] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*, pages 928–936, 2003. 9

[262] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *ICLR*, 2017. 8, 22, 102