MONASH University

Advances in Decision Forests and Ferns with Applications in Deep Representation Learning for Computer Vision

by

Yan Zuo

Supervisor: Prof. Tom Drummond

A thesis submitted for the degree of Doctor of Philosophy at
Monash University, May, 2019.

Copyright Notice

Abstract

In recent years, the field of computer vision has seen remarkable progress that has led to the emergence of new and useful technologies which have both aided our day-to-day lives and improved upon various aspects of industry. This is in part due to the rise in popularity of deep learning, which has shown that powerful, learned representative features can offer a level of performance that is often hard to match using more traditional handcrafted features. Currently, deep learning has largely embedded itself within computer vision as an extremely useful tool for accomplishing a range of vision-based tasks such as image classification, object detection and semantic segmentation. With this surge in popularity in deep learning, more traditional computer vision methods such as decision forests have somewhat fallen out of favour in the community despite their benefits. This thesis hopes to bridge this gap; it focuses on advancing the ensemble methods of decision forests and ferns and incorporating them within deep learning frameworks for computer vision applications.

First, a novel ensemble learning approach that constructs a decision forest model which takes cues from boosting approaches under a residual framework setting is presented. This framework creates an ensemble of decision trees designed to cooperate with one another, using the complementary information between each other to minimise a global loss. This approach allows for highly compact shallow decision forests to be constructed without the expected drop in performance normally associated with shallow tree models.

Subsequently, this thesis also investigates methods for incorporating a decision forest within a deep learning framework. A piecewise method is offered, whereby a pretrained Convolutional Neural Network is used as a feature extractor alongside the residual forest classifier to perform various semantic segmentation tasks.

This is then extended upon such that the residual forest classifier is used to learn representation features, offering an end-to-end approach for learning both features and a classifier. This is demonstrated to improve training speed over the baseline pure deep learning approach and offers improved performance in semantic segmentation tasks on data sets with limited training data.

Following this, the conditioning of deep neural networks is investigated where it is shown incorporating a decision forest in these frameworks greatly improves conditioning of the network and aids training stability. This is used in an application of decision forests within a Generative Adversarial Network to significantly improve performance on the task of image generation.

Finally, this thesis investigates the application of decision ferns for controlled operations in the latent space of a Variational Autoencoder. A novel decision fern controller for operating on latent variables is introduced and demonstrates improved performance over spatial transformation tasks and the more complex inference task of video prediction.

Declaration

This thesis contains no material which has been accepted for the award for any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Print Name: Yan Zuo

Date: Tuesday 21st May, 2019

Acknowledgements

First and foremost, I would like to sincerely thank my supervisor Professor Tom Drummond. You have provided me with guidance and support and pushed me to pursue interesting and worthwhile research under your supervision. I have been fortunate enough to be your student and have learned a lot under your guidance during the course of my PhD.

Thank you to the Australian Government Research Training Program and the Australian Centre for Robotic Vision, which have supported me financially and enabled me to pursue research that is both interesting and challenging within the field of robotics and vision.

I would like to thank my fellow colleagues Andrew, Ben Harwood, Ben Meyer, Gil, Thanuja, Vincent, Winston and Yanming. I have thoroughly enjoyed the numerous coffee runs, board game nights and discussions on research we have had during my PhD. In particular, I would like to thank Gil, whom I have collaborated with for multiple works which have contributed significantly to the research contained in this thesis.

I would like to thank my family for all their love and support. Thank you to my father, Xinyi, for encouraging me during the difficult times of my PhD. Thank you to my mother, Qing, who has always looked after and taken care of me. Thank you to my sister, Helen, who has

always lent a listening ear and been there for me.

Finally, I would like to thank my partner Sophia, for her continued patience, love and understanding during this arduous (but often times enjoyable) process. The support you have provided me, both physically and emotionally, was an integral part of this journey and I could not have done this without you.

Contents

Chapter 1

Introduction

The field of Artificial Intelligence (AI) is defined as the study of *intelligent agents*; machines which are able to perceive their environment, form understandings and take actions which maximise their chance of achieving a set of objectives [164]. On the sensory side, researchers have long sought to create such machines which can extract semantic information from their surroundings through equipping them with a variety of sensors.

Of these sensors, cameras have provided machines with arguably the most important of the five senses for visualising the world around them: sight. Often, visual perception of the real world is all humans need to gain an understanding of a scene. We exploit knowledge from both past experiences, as well as from the local scene to give the required context to make complex inferences such as interpreting visual information. Enabling a machine to recognise and learn through the visual data it receives is an incredibly important task; this has opened up an entire field of research known as Computer Vision.

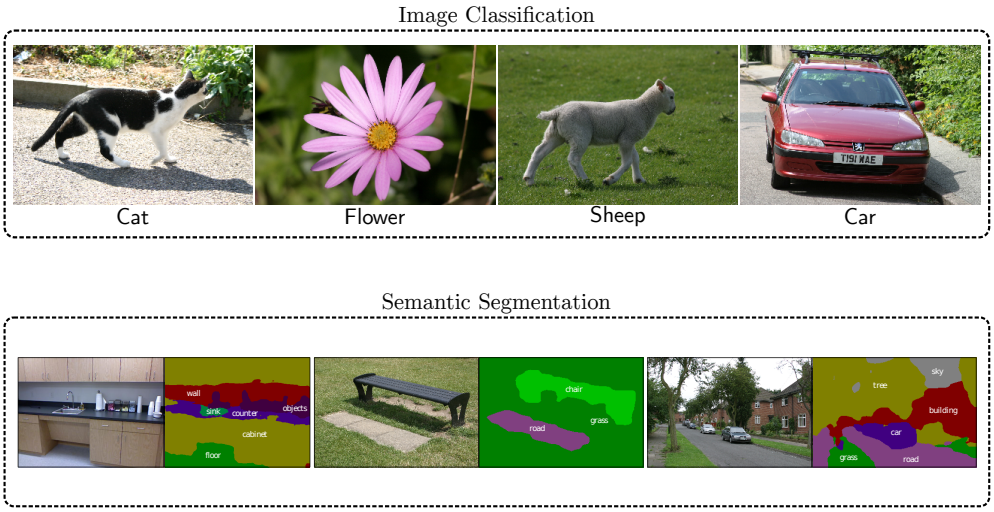


Figure 1.1: Visual recognition tasks of image classification (top) and semantic segmentation (bottom). The examples provided for semantic segmentation are actual results from the work done in Chapter 5.

1.1 Computer Vision

Computer vision is an interdisciplinary field that is concerned with enabling machines to gain a high-level understanding from digital images or videos. One of the long-standing goals of computer vision is to allow computers to perform complex visual inference tasks in a general and robust manner. From an engineering perspective, computer vision investigates the automation of tasks that are performed by the human visual system [202]. This ability to process visual information is incredibly important and the estimates are that the human brain dedicates more than 30% of its capacity towards visual processing [109].

1.1.1 Applications of Computer Vision

Imbuing machines with the power of the human visual system serves as one of the inspirations for this investigation of computer-based visual recognition approaches. Aside from scientific curiosity, advancing the area of computer visual

recognition improves the feasibility of several important applications:

Robotics & Automation: In robotics, the field has drastically advanced in recent years. Humanoid robots are now able to perform a range of actions; from simple ones such as grasping objects to more advanced actions such as climbing a ladder. A better visual understanding of the world will allow robotics to find applications of automation in the real world.

Image Search: In the current information age, the world-wide web is an ocean of information. Search engines can now reliably index the vast quantity of textual information, allowing for access to billions of documents that exists (and growing). However, visual data such as images and videos still cannot be as reliably indexed. The capability to accurately access images and videos on the web would be immensely beneficial in many areas including research, web browsing, data collection, amongst others. The ability to index images and videos based on extracted semantic content (rather than just relying on meta-data and accompanying textual information) would prove invaluable.

Medicine: Diagnosis of tumours and other health-related abnormalities from medical scan is often times difficult and requires expert knowledge. An automated system would be able to improve diagnosis accuracy, improve efficiency in diagnosis and cut down on human-related errors.

Transport: Improving safety systems of vehicles is still a very active area of research. Modern vehicles are equipped with a wide array of sensors to improve the driving experience. For example, it is now common to find anti-lock braking or anti-skid braking systems (ABS) in most vehicles which automatically assist

drivers during braking, preventing the wheels from locking up and avoiding uncontrolled skidding. By enabling technology which equip computers with scene understanding, we are now starting to see the deployment of self-driving vehicles to further improve road safety.

Security: Automated recognition of individuals of interest or suspicious activity would improve safety in public spaces. This same technology could also be extended towards private home security systems; a security system with semantic understanding of its environment would be able to differentiate between false alarms such as movements from domestic animals and real alarms such as disturbances created by an intruder.

1.1.2 Difficulties of Computer Vision

Often it is easy for us to forget just how difficult the task of visual understanding of the real world is since this task feels so natural and effortless to perform. As an example, looking at Figure 1.1, it would be an almost trivial task to classify the depicted images of various objects shown (as in the case of image classification) or to form an understanding of the scene by segmenting out objects of interest in a scene and assigning them correct semantic labels (as in the case of semantic segmentation). Although we would like machines to inherit this ability to distil semantic information effortlessly from visual data, the method for achieving this is not so clear.

First, a computer represents an image as a large array of numbers which indicates brightness and colour at a position in the image. Each number in this array is referred to as a *pixel*. Any given image may contain millions of these pixels. In order for a machine to understand the content of an image, it must transform these pixels from patterns of brightness and colour into high level features which represent the semantic content within the image. Adding to this difficulty, any

changes to the visual information, whether it be different lighting conditions, differences in camera viewpoints or variances in object pose could alter these brightness and colour patterns to be completely different. This presents the difficult challenge of creating a model that accurately describes a mapping from low-level pixel patterns in the image to high-level semantic concepts which can generalise across variations in images under similar semantic content.

1.1.3 Machine Learning for Computer Vision

A lot of the practical applications in computer vision we wish to perform can be described as finding a mapping from some input space to a corresponding target output space (*i.e.* we are trying to find some model which describes the mapping from a non-meaningful input to a meaningful output). As an example, consider the task of image recognition - something we perform several times on a day to day basis with relative ease. Although we are able to do this with relative ease, for a machine, this task is far from a trivial one. The difficulties associated with such problems become more apparent upon closer inspection: how does one explicitly specify a set of tangible properties which attribute an image of some object to a particular class or meaningful label (*e.g.* an image contains cat or dog objects)?

Machine learning benefits from the fact that often it is far easier to obtain samples of the desired mapping we are wishing to model. It covers a class of algorithms that are data-driven; rather than trying to explicitly construct a model given a set of observations, machine learning algorithms differ from conventional algorithms by learning from samples drawn from a data distribution. Mitchell's definition of machine learning describes it as the following: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [145].

In real world tasks, there exists a wide variety of experiences, tasks and perfor-

mance measurements. For the task of vision-related learning, these experiences usually involves observing some set of visual examples such as RGB images. Given visual recognition tasks such as that of trying to describe a face, a conventional non-machine learning approach may be to try to model this by identifying the important attributes of what compose a face: an oval or circular coloured shape, darker areas where the eyes and mouth are expected, an elevated contour in the centre for a nose and so forth. Normally, these characteristics would be explicitly specified by the programmer. By contrast, a machine learning algorithm would not have such a coded definition, instead relying on a learn-by-examples approach; it would eventually learn the defining characteristics of a face using a collected dataset of faces and face annotations (commonly referred to as *labels*) provided by an expert (typically a human annotator).

Data-driven machine learning approaches have played a large role in semantic understanding of image data for computer vision. These approaches aim to learn some non-linear mapping from an input image (commonly RGB images) to a target output which represents the underlying semantics of the input. Modern computer vision techniques rely heavily on machine learning and more recently, deep learning algorithms. As discussed earlier, an important aspect of vision is representation; that is, the ability to extract key features which help in assigning semantic meaning to raw image data. Machine learning approaches allow for automated learning of these representations and this has been a key component for their dominance in solving a variety of computer vision related tasks.

1.1.4 Deep Learning

Despite the difficult nature of computer vision tasks, in recent times we have seen significant progression in the area of visual recognition. In particular, performance in image classification tasks has dramatically risen since the introduction of state-of-the-art deep models based on Convolutional Neural Networks (CNNs). The ability for these models to learn powerful *representations* which capture the

underlying intrinsic characteristics of images have enabled the categorising of images belonging to thousands of classes at performance levels comparable to humans [117] or more recently, surpassing them [78, 79]. In related tasks such as objection detection and semantic segmentation, we have seen similar levels of improvements in performance [25,63,64,135,170,229,230]. These recent advances have enabled real world applications of tasks that would otherwise have been infeasible such as face detection and recognition, self-driving cars and advanced image searching to name a few.

1.1.5 Remaining Challenges

In spite of the benefits of deep learning, there still remain open challenges within computer vision. Although deep learning techniques have dramatically improved our ability to perform many visual understanding tasks, they are known to be very data-hungry, often requiring millions of hand labelled samples to achieve the levels of performance in visual recognition they are known for [182]. In addition, deep models are often computationally expensive both to train and to infer. There exists a tradeoff between accuracy and speed; we have yet to achieve human levels of performance in visual recognition tasks at real time levels of performance.

1.2 Decision Forests

Without a doubt, deep learning has propelled the field of computer vision forward in recent years; we are now able to perform a range of vision-related tasks which would otherwise have been infeasible without modern CNN approaches. However, with the success that modern deep learning techniques have found in computer vision, classical computer vision methods have now taken a backseat in the field. Of these classical approaches, the once-popular ensemble method of decision forests has seen a steady decline as a topic of interest in relation to visual learning tasks.

Despite this, ensemble methods such as decision forests possess several favourable attributes which can help address many of the relevant challenges associated with modern deep learning for vision-based applications. They are compact and efficient learners, offering lots of non-linearity in a small space [21]. This allows decision forests to train efficiently, build compact models and learn with less data when compared to their deep learning counterparts.

With this in mind, we show that integrating decision forests within deep networks can bring many of the aforementioned benefits into the deep learning space. Our findings indicate that the incorporation of decision forests allow for improved performance across a range of vision tasks, increased training speed and efficiency and extra robustness to limited training data. As we will show in Chapter 7, the usage of decision forests in a deep learning context can significantly improve the conditioning of the learning problem during training. This subsequently increases training stability and offers insights towards why they can be a beneficial inclusion within deep learning frameworks.

1.3 Contributions

This thesis investigates and explores improving upon ensemble learning approaches related to decision forests and ferns and the subsequent deep learning applications for vision with these ensemble methods. A majority of the work presented centres around the gains that decision forests and ferns can offer towards improving existing deep learning approaches for a variety of computer vision applications. Throughout our investigations, we find that decision forests and ferns offer improvements ranging from improved training speed, more favourable tradeoffs to model capacity to model complexity and improved performances for a wide variety of inference tasks.

Chapter 4 begins with the introduction of a novel ensemble learning method which utilises a technique for combining residual information between base decision tree learners in the ensemble. The aim of this chapter is to exploit the complementary information between decision trees in the ensemble and minimise a global loss for the ensemble, rather than using a locally greedy, entropy minimising approach found in conventional random forests. We demonstrate that our model can be optimised in closed form, simplifying the model tuning process. Additionally, this approach allows for the construction of much shallower models which results in a large compression in model size, with minimal tradeoff to model performance.

In Chapter 5, we look at incorporating our decision forests within a deep learning framework. For this, we implement a hybrid model consisting of a Convolutional Neural Network (CNN) component and a decision forest component. The CNN is pretrained on a general, large scale image dataset where it learns to extract generic features to be used by the decision forest for classification. We show that by using our decision forest to replace the solver component of the CNN, we are able to achieve improved performance on scene segmentation tasks.

Integrating a decision forest within a deep learning framework is a difficult task.

Although it is relatively straightforward to directly combine a pretrained CNN and decision forest where the former acts as a general feature extractor and the latter classifies, learning features for the decision forest in an end-to-end training context is not so straightforward. Since the routing function of decision forests is not differentiable and much of deep learning relies on backpropagation to train models, this presents an issue in applying decision forests within a deep learning framework. In Chapter 6, we continue to look for ways to address this issue. We develop a method which approximates the forward pass of a decision forest using a novel loss blending technique during the backward pass which compute gradients. This allows for the derivation of backward gradients to update the CNN using the decision forest as a classifier. Additionally, we show that generating the statistics stored in the leaves of a decision tree can be treated as an optimisation problem and learned using gradient descent.

Following this, Chapters 7 & 8 explore the learning of latent representations using decision forests and ferns. Chapter 7 presents the first application which looks at using decision forests to improve image generation tasks by combining decision forests with Generative Adversarial Networks (GANs). We demonstrate how a decision forest can greatly improve the conditioning of model learning when applied to the discriminator network of the GAN. This in effect addresses a known issue in GAN training by improving training stability, which subsequently leads to increased performance on image generation tasks. Additionally, we develop the residual forest framework to use soft decisions, where instead of hard routing to a single prediction leaf node, a proportion is routed to all prediction nodes. This allows the decision forest to be treated as a layer within a deep learning model and trained in true end-to-end fashion.

Finally, Chapter 8 shows another potential application of ensemble methods within generative models. We show how an ensemble of decision ferns can serve as an effective controller within the latent space of a Variational Autoencoder and perform complex inference tasks such as video prediction. We demonstrate how decision ferns can be constructed in this latent space using common neural

network components such as fully-connected linear layers and hyperbolic tangent functions through a reparameterisation trick. We show that our approach can offer performance improvements over other contemporary works which also perform video prediction.

1.4 Collaborations

On top of the research work carried out by myself under the supervision of Prof. Tom Drummond, I have been fortunate to be part of a successful collaborative effort with Gil Avraham at Monash University during the last year of my PhD.

My collaboration with Gil occurred internally within the ECSE Robotics Laboratory at Monash University, which is a research node of the Australian Centre of Excellence for Robotic Vision. The collaborative work between Gil and myself mainly looked at the benefits that ensemble methods could offer when working with latent representations of data which targeted both our research interests. The work presented in Chapters 7 & 8 are drawn from submitted work or publications jointly authored with Gil Avraham. Gil and I are equal contributors in both these publications and in both chapters, I described the percentage breakdown which describe the contributions I have personally made for the production of the referenced work, which both Gil and I have agreed upon.

1.5 Publications

The main body chapters of this thesis are based on the following peer-reviewed or currently under review articles:

- **Residual Likelihood Forests**
  Y. Zuo & T. Drummond
  Submitted to *Conference on Computer Vision and Pattern Recognition (CVPR), June 2019*

- **Fast Residual Forests: Rapid Ensemble Learning for Semantic Segmentation**
  Y. Zuo & T. Drummond
  In *Conference on Robot Learning (CoRL), November 2017* [234]

- **Generative Adversarial Forests for Better Conditioned Adversarial Learning**
  Y. Zuo[*], G. Avraham[*] & T. Drummond
  Submitted to *Conference on Computer Vision and Pattern Recognition (CVPR), June 2019* [232]

- **Traversing Latent Space with Decision Ferns**
  Y. Zuo[*], G. Avraham[*] & T. Drummond
  In *Asian Conference on Computer Vision (ACCV), December 2018* [233]

[*]indicates equal contribution

1.6 Thesis Layout

In Chapter 2, we perform a review of existing works in the literature with a focus on decision forest and boosting ensemble methods as well as neural networks. In Chapter 3, we layout the necessary preliminaries for concepts used in this thesis which describe to mathematics of related ensemble methods and fundamentals of deep neural networks.

Following this, the primary contributions outlined in this introduction are expanded upon in Chapters 4 to 8. Chapter 4 introduces the Residual Likelihood Forest framework, a novel ensemble learning approach which serves as the baseline framework for a majority of this thesis. Chapter 5 extends upon the Residual Likelihood Forest framework, demonstrating how the method can be transitioned into a hybrid deep learning framework. Chapter 6 further develops upon the Residual Likelihood Forest framework; we describe a technique which allows for deep representation features to be learned alongside the forest classifier. In Chapters 7 & 8, we move from discriminative tasks to the generative tasks and describe some applications of decision forests and ferns within this area. Chapter 7 looks at incorporating decision forests together with Generative Adversarial Networks to improve model conditioning and hence their training stability. Chapter 8 describes the application of decision ferns within the latent space of a Variational Autoencoder to perform complex inference tasks such as video prediction.

Finally, we conclude the thesis in Chapter 9, provide a summary of the contributions and discuss future avenues of research.

Chapter 2

Background

In this chapter, we outline the past and current research in the areas relevant to the work present in this thesis. This includes various ensemble methods such as decision forests and boosting, perceptron-based approaches including multi-layered perceptron models and deep learning methods utilising convolutional neural networks for various computer vision tasks such as classification and segmentation.

2.1 Ensemble Methods

The practice of combining the predictions of multiple learning algorithms to produce a single model is referred to as an *ensemble method*. The resulting ensemble model generally performs at a level higher than any of the individual classifiers making up the ensemble [157]. The individual learning algorithm can be any classification method; commonly chosen methods are support vector machines, decision trees and neural networks. In this section, we focus our discussion around ensembles of decision trees, decision ferns and boosting algorithms due to their relevance to this thesis.

2.1.1 Decision Forests

Breiman first introduced the term Classification and Regression Trees (CART) [17] to refer to decision tree algorithms that can be used for classification and regression predictive modelling tasks. In general, this algorithm is simply referred to as *decision trees* or *decision forests*. For the remainder of this thesis, unless otherwise specified, we refer to this class of algorithms as decision forests. Decision forests have established themselves as a prominent ensemble learning method, finding a wide range of applications across various fields including computer vision, where they are used in numerous tasks including object recognition [58], semantic segmentation [198] and data clustering [149]. Although decision forests are well known for their strong discriminating power [166], they initially suffered from variance and stability issues, having a tendency to overfit to a problem.

2.1.1.1 Induction of Decision Trees

One of the earliest and most well-known algorithms in the literature for building decision trees is the Iterative Dichotomiser 3 (ID.3) algorithm [166]. This algo-

rithm builds decision trees from a set of training data, using an information gain maximisation criteria (or entropy minimisation) to determine a splitting decision that separates data most effectively. Trees are constructed in a recursive manner and optimisation of split node decision is performed in a greedy, top-down manner, where the optimal split is found for each local decision node. This ensures that at each node, a feature of the data is chosen so that the data is split into subsets such that they are enriched in largely one class. The ID.3 algorithm then recursively applies itself on the smaller subsets in the same manner until a leaf node is reached and a prediction can be made about input data. The C4.5 algorithm [167] was subsequently developed as an extension to the ID.3 algorithm. C4.5 builds decision trees from a set of training data in the same way as ID.3, however several improvements were made over the original algorithm. These included the ability to accept both continuous and discrete features, handling incomplete data point and using a bottom-up leaf pruning technique to address overfitting.

2.1.1.2 Random Forests

The general method of random forests was first proposed by [83] as a way to overcome the variance and stability issues of binary decision trees. The work showed that an ensemble of binary tree classifiers could improve in performance as the trees are grown but did not overfit, as long as the subspace of feature for training was randomly chosen. A subsequent follow up work to this showed that this extended to other splitting methods as long as feature subspace was randomly restricted [84]. At the time, this was in strong contrast to the belief that the complexity of a classifier can only grow to a certain level before classification performance was hurt by overfitting. Amit & Geman introduced the idea of using a restricted random subset of available features for making a split decision when growing a single tree [1]. Dietterich proposed the concept of Randomised Node Optimisation in which a randomly selected subset of possible parameters is trained at each node [36].

These aforementioned works were used by [16] to finally compile the random forest method into its current form, with the final step of incorporating the bootstrap aggregation technique in [13] into the random forest approach. This method enhanced the original random forest approach, bootstrapping the data from the original training set by creating many random subsets of the data set (with replacement) and training each individual tree classifier on a selected subset. Geurts *et al.* introduced Extremely Randomised Trees which selected an attribute common amongst all instances in a subset of data to be classified and drew a random cut-point uniformly between the minimum and maximum values that the selected attribute can take [62]. It then performs a split on the data by comparing the value of the selected attribute in all instances of data with the randomly generated cut-point.

2.1.1.3 Applications of Decision Forests

Due to their efficiency, flexibility, good generalisation capability and inherent ability to handle multi-class problems, decision forests have found numerous applications in the field of computer vision. Moosmann *et al.* introduced Extremely Randomised Clustering Forests which used Extremely Randomised Trees as a baseline to efficiently perform data clustering [149], using various visual descriptors including SIFT [137] and Haar wavelet transforms. They found that this approach was more robust to image background clutter, offered performance improvements and was much faster in training and testing than conventional K-means approaches. The work of Bosch *et al.* proposed to use random forests as a classifier in conjunction with various feature descriptors to perform image classification [11]. Shotton *et al.* extended upon this method, using it to perform image categorisation and semantic segmentation [198]. A random forest was used as a global image classifier to quickly and efficiently provide priors which was subsequently pipelined into the framework of [199] to improve performance in semantic segmentation. Their work differed from [149] by learning directly from pixels instead of visual descriptors and maintaining hierarchical clusters within branch nodes. Schroff *et al.* used random forests to perform semantic seg-

mentation using HOG features as inputs [190] and Brostow *et al.* demonstrated that Extremely Randomised Trees could be used with motion and structure cues alongside appearance-based features to improve object recognition in challenging, dynamic environments [18].

The work of Gall *et al.* introduced Hough Forests, which extended upon the random forest method by incorporating the Hough Transformation to perform object detection [58]. The approach used class-specific random forests to learn a direct mapping between the appearance of an image patch and its corresponding Hough vote. In [197], random decision forests were used to predict 3D positions of body joints from a single depth image. The approach used an object recognition approach, separating the body into intermediate parts and allowing for per-pixel classification of each part. This work was extended upon by Keskin *et al.* which used random decision forests for hand skeleton tracking [105]. Subsequently, Kontschieder *et al.* extended on the work of [58] using priority queuing in order to incorporate contextual features into the training process of a decision forest [113]. Dollar *et al.* showed that random decision forests could act as a structured learning framework for predicting local edge masks [37]. The approach used structured labels to train random forests, allowing them to map these structured labels to a discrete set of labels of a given node in a decision tree. Ultimately, this enabled random forests to be trained using standard information gain maximisation procedures.

The work of Montillo *et al.* proposed the use of entangled random forests to augment the feature space [147]. The entangled random forests were grown in a breadth-first manner where the progress of the trained subtree was used to refine the feature space using the class posterior distributions. Schulter *et al.* combined the global loss minimisation benefit of boosting approaches whilst retaining the benefits from the original random forests method [191]. An alternating update algorithm approach was proposed which alternated between weight updates and parallelised tree growing in a breadth-first manner. In [169], a global optimisation scheme was used to globally refine a random forest after it had been constructed.

This approach used a global loss function to iteratively refine values in the leaf node to minimise redundancy in information between tree classifiers in the ensemble. This was combined with leaf pruning to significantly reduce model size for various classification and regression tasks whilst retaining a competitive level of accuracy.

2.1.2 Random Decision Ferns

A related ensemble classifier to random forests are random decision ferns. Decision ferns are closely related to decision trees, with the key difference that ferns are non-hierarchical in nature; they can be thought of as a constrained version of a decision tree where the same question is asked at each level in the tree. Decision ferns represent a faster, alternative solution to random decision trees and allows for more parallelised construction of the ensemble classifier due to its non-hierarchical nature, with a tradeoff being that less questions can be asked for the same given model size. Of the works that use decision ferns in computer vision, arguably the work of Ozuysal *et al.* is the most well known, which proposed a method using random decision ferns to classify image patches for keypoint detection [159]. For each image patch instance, each random decision fern would perform a series of pixel intensity comparisons. This allowed for the implementation of a simple, but efficient and robust keypoint detector where the outputs of multiple decision ferns were combined to give consensus on valid keypoints. The work of Villamizar *et al.* proposed a boosting algorithm which used random decision ferns as a base weak learner and binary features in the HOG space to perform efficient rotation invariant object detection [214]. In a follow up work, Villamizar *et al.* employed random decision ferns for object detection and classification [215]. This approach used random ferns to create a shared feature pool, and subsequently used this shared feature pool with class specific ensemble classifiers composed of random ferns as the base weak learner.

2.1.3 Boosting

The core idea behind boosting algorithms is based around the idea of combining multiple weak learners into a single strong learner. This concept differs from the bootstrap aggregation approaches described in the previous section. Bootstrap aggregation relies on averaging large numbers of individual learners to create a single learner; it relies on the law of large numbers to reduce variance and over-fitting in the final model. In contrast, the boosting approach aims to iteratively create a strong learner; it does this by giving emphasis on misclassified data for subsequently created weak learners. Boosting algorithms are based on the PAC learning framework which proposes an idea of weak learnability versus strong learnability.

2.1.3.1 PAC Framework

Valiant introduced the notion of strong learnability where a class of concepts is defined as such by the existence of a polynomial-time algorithm which achieves low error with high confidence for all concepts in the class [211]. This led to the initial development of the Probably Approximately Correct (PAC) learning framework that boosting algorithms would later extend upon. The PAC learning framework tries to define a learner which attempts to select a generalisation function (model) from a certain class of possible functions to fit a set of sample data. This defines the overall goal of PAC learning: selection of a function that will *probably* have low generalisation error and hence be *approximately correct*. Under this criteria, the learner must be able to learn the concept given any arbitrary approximation ratio, probability of success, or distribution of the samples. Subsequently, Kearns defined a model to have weak learnability if it could not achieve arbitrarily high accuracy but could output a hypothesis which performed slightly better than random guessing [103]. This left open the question of whether there was an equivalency between the notions of weak and strong learnability.

2.1.3.2 Discrete Boosting

The work of Schapire addressed this question, showing that weak and strong PAC learning are equivalent [188], which introduced the concept of boosting, using an algorithm called the Recursive Majority Gate of Hypotheses which selectively combined several weak hypotheses into a single strong hypothesis. Freund extended upon this algorithm to combine the outputs of all weak hypotheses [47] and Kearns & Valiant showed that under a uniform distribution on the instance space, monotone Boolean functions are weakly, but not strongly, learnable [104]. This indicated that strong and weak learnability were not equivalent when certain restrictions are placed on the instance space distribution and that strong and weak learning models would prove to be inequivalent under the distribution-free case as well.

Freund introduced the Boost by Majority algorithm which improved upon the algorithm of [188] through increasing efficiency of the algorithm [48]. The recursive element of the algorithm was removed, favouring a final hypothesis serving as a single majority gate which combined the outputs of all weak hypotheses in the ensemble. Freund & Schapire would subsquently develop a method which took advantage of the adaptability of weak learners, leveraging this into an adaptive boosting algorithm called Adaptive Boosting (AdaBoost) where each subsequent weak learner was tuned towards favouring misclassified training instances through increased weighting of these data points [50]. Although AdaBoost was a successful algorithm which saw wide use in practice due to its simplicity and adaptivity, [36] showed that AdaBoost was particularly susceptible to classification noise when compared to other ensemble methods such as bootstrap aggregation. The work showed that AdaBoost had a tendency to assign higher weights to noisy examples over other examples. This resulted in hypotheses generated in later iterations which were overfit to noise. Freund would later address this with the BrownBoost algorithm, which ignored instances that were repeatedly misclassified, under a core assumption that instances which are constantly misclassified are afflicted with noise [49]. This allows the classifier to forfeit correctly classifying these noisy instances in favour of making the final classifier more robust to

noise.

2.1.3.3 Gradient Boosting

Breiman first observed that boosting could be interpreted as an optimisation algorithm on a suitable cost function [14]. Following this, in [15], Breiman recast the Adaboost algorithms as a statistical framework in which they were referred to as ARCing (Adaptive Reweighting and Combining) algorithms. This class of algorithms was described as stagewise additive; each new weak learner is added one at a time and existing learners in the model were left frozen and unchanged. This led to the generalisation of AdaBoost and related algorithms into a class of algorithms known as Gradient Boosting. Similar to boosting approaches, gradient boosting iteratively builds a prediction model composed of an ensemble of weak prediction models (typically, the base weak learner is a decision tree or neural network). However, it generalises the boosting approach by allowing optimisation over an arbitrary differentiable loss function. This allowed boosting to be cast as a numerical optimisation problem which used a procedure similar to gradient descent to minimise the loss of the model through adding of weak learners. This framework was subsequently further developed by Friedman and called Gradient Boosting Machines [55] which was later known as simply Gradient Boosted Trees [54]. This framework extended gradient boosting to decision trees of a fixed size as base learners and a modification to the gradient boosting method allowed for an improved quality of fit for each base tree learner.

2.2 Representations of Data

In its raw form, real world images collected via sensors such as an RGB camera is usually high dimensional, non-linear and correlated in nature. For a machine to perform a direct interpretation of data in this form is difficult; an extensive line of research tries to address this issue by preprocessing or preparing real world data into a form that is more digestible for machines to interpret. This process is known as *feature extraction* where an initial set of measurements (data) is processed such that redundant information is removed so that the derived values (features) are more informative.

2.2.1 Feature Extraction

Feature extraction is in large part motivated by the fact that many tasks we wish for machines to learn (such as image classification) often require an input which is both mathematically and computationally convenient to process. However, most real world data such as images, video, and sensor measurement does not come in a form which fits these properties: they are usually highly non-linear, often redundant and highly variable. Thus, feature extraction offers a method for discovering useful features or representation from raw data such that it is easier to process. Commonly, feature extraction is a dimensionality reduction process where raw data that is high dimensional and usually redundant is reduced to a lower dimensional representation which still accurately captures and describes the information in the original data [209].

Feature extraction typically consists of two distinct stages. The first stage involves finding points of interest in the images; these are usually spatial locations or points in the image which are interesting or stand out in the image. Good keypoints are typically chosen to be scale and rotation invariant such that after distortions to the image, the same keypoints can still be recovered from the distorted image.

The second stage involves describing keypoints once they are detected using a *feature descriptor*. These feature descriptors are engineered by a human using prior knowledge of the real world. This allows for a set of descriptive features to be created which ideally should capture the intrinsic properties of the raw input data.

Lowe first introduced the Scale Invariant Feature Transform (SIFT) descriptor, which detected interest points in a grayscale image, accumulating statistics of local gradient directions of image intensities to give a summarising description of local image structures in a local neighbourhood around each interest point [137]. Similarly, in [31], Dalal & Triggs proposed the Histogram of Oriented Gradients (HOG) descriptor for describing local object appearances and shapes using distributions of intensity gradients. The method divided the image into small connected regions called cells and a histogram of gradients directions is created for the pixels within each cell. The HOG descriptor was the concatenation of these histograms and was applied to pedestrian detection in static images. Bay *et al.* proposed Speeded Up Robust Features (SURF), using a blob detector to find points of interest and Haar wavelet response to describe these points [6]. This was followed by the work by Calonder *et al.* which proposed the Binary Robust Independent Elementary Features (BRIEF) descriptor [20]. This method generated a binary descriptor by directly computing binary strings from point pairs within image patches. The BRIEF descriptor was extended upon by the ORB descriptor [180] and the BRISK descriptor [128]. The former approach combined the Features from Accelerated Segment Test (FAST) detector [176] with the BRIEF descriptor, modifying the feature descriptor by computing unambiguous orientations and making it robust to viewpoint changes. The latter approach introduced the Binary Robust Invariant Scalable Keypoints (BRISK); unlike BRIEF or ORB descriptors, pixels were sampled in a predefined pattern over concentric rings. It achieved rotation invariance by rotating the sampling pattern by the computed orientation of a given keypoint.

2.2.2 Feature Learning

Feature learning (also known as *representation learning*) is a set of techniques which allows for automatic discovery of representation features from raw data. These representation features are effectively a transformation of raw data input to a representation that can be effectively exploited in machine learning tasks. As discussed in the previous section, feature extraction involves the use of hand-crafted features. This requires expert knowledge about the problem at hand, and often does not generalise well across tasks. This serves as motivation for the design of efficient feature learning techniques which are able to generalise well across tasks. Feature learning removes the need for manual feature engineering and instead allows a machine to learn both a specific task and the necessary features themselves. In recent times, this approach has been widely adopted within the machine learning community as the method of choice for a variety of vision-based tasks such as image classification [79, 117, 201], object detection [63, 64, 229] and semantic segmentation [44, 77, 135]. The power of representation learning allows for both features as well as classifier to be jointly learned and has recently demonstrated enormous success in tasks where input data is high dimensional and complex (as is often the case with image data). Feature learning can be separated into two categories: supervised and unsupervised learning. In supervised feature learning, features are learned which represent the semantic labels underlying the data, whilst unsupervised feature learning involves discovering features that capture some underlying structure about the input data.

One of the earliest forms of representation learning can be traced back to Principal Component Analysis (PCA) [161] which was a statistical method for converting a set of observations that were possibly correlated into a set of values that were linearly uncorrelated called principal components. Following this, Linear Discriminant Analysis (LDA) was proposed by Fisher. LDA offered a way to reduce dimensionality while preserving as much of the class discrimination information as possible, allowing for boundaries around clusters of classes to be found [46]. Data points are projected and separated out on a line with each cluster occupy-

ing areas in a relatively close distance to a centroid. This method was closely related to PCA as both methods searched for linear combinations of variables which best explain the observed data. The work of Rumelhart *et al.* introduced what is arguably one of the most well-known and widely used methods for learning representations [181]. This method allowed for the weights of a model to be estimated via backward propagation of errors (backpropagation). The method is commonly used to train deep neural networks and is often used by the gradient descent optimisation algorithm to learn representations by calculating the gradient of the loss function with respect to parameters of the model. Comon proposed the method of Independent Component Analysis (ICA) which was a technique for revealing latent factors that underlie a set of random variables [28]. Local methods have also been deployed to discover the intrinsic structure of high dimensional data such as Isometric Feature Mapping (Isomap) [207] and Locally Linear Embedding (LLE) [179].

The first successful application of deep neural networks to dimensionality reduction by Hinton & Salakhutdinov introduced the concept of *deep representation learning* [82]. Following this, Deng & Yu defined the concept of deep learning as a class of machine algorithms that utilise a stack of many layers with nonlinear processing units for feature extraction and transformation, with each successive layer using the output from the previous layer as its input [35]. In this setup, higher level features are derived from the lower level feature to form a hierarchical representation of the data. Vincent *et al.* proposed the idea of stacked denoising autoencoders to locally remove noise from corrupted versions of the input data [217]. The introduction of the AlexNet framework paved the way for powerful representation (feature) learning methods. Donahue *et al.* proposed a feature extraction method which used an ImageNet [34] pretrained AlexNet and repurposed these featured to novel generic tasks. These features were named Deep Convolutional Activation Features (DeCAF) [38]. Subsequently, Razavian *et al.* used these features to improve performance on standard image classification tasks [196]. The work of Zeiler & Fergus visualised the features learned by AlexNet in order to try and form a better understanding of why they performed so well [227].

2.3 Artificial Neural Networks

2.3.1 Perceptron Models

Perceptron-based models are approaches to learning which try to simulate the human brain in a computer. They follow the principals of roughly how a brain operates. A perceptron tries to artificially simulate a *brain neuron* in action. This brain neuron is a cell made of two significant components: some *dendrites* and an *axon*; these components were used to receive and send impulses respectively. Axons and dendrites propagate signals to each other via a connection called a *synapse*. Over time, the connections between axons and dendrites change; the brain *learns* as the conditions under which neurons become excited evolve. Rosenblatt first proposed the concept of a perceptron as an algorithm for supervised classification [175]. Since then, various linear perceptron methods would be proposed [51, 133]. In the following section, we detail the more relevant multi-layer perceptrons, which serve as a precursor to modern neural network models.

2.3.1.1 Multi-Layer Perceptrons

For most real applications, we use data that is complex and non-linear, which is difficult for linear models to deal with. If instances are not linearly separable, learning will never reach a point where all instances are classified properly. Multi-layer perceptrons are a type of feed-forward neural network designed to try to address this problem. A multi-layer perceptron (feed-forward network) consists of a large number of connected units (neurons). The units in the network usually consist of three classes: input units which receive information to be processed, output units which output the results of processing, and hidden units which lie in between the input and output units in the network and process the data. Feed-forward Networks (FNNs) allow signals to travel one way only, from input to output. A FNN relies on three fundamental aspects: input and activation

functions of the unit, architecture of the network and the weight of each input connection. Since the first two aspects are fixed, the behaviour of the FNN is determined by the current values of the input connection weights. The network is initially seeded with random values for the connection weights, and then instances of the training set are repeatedly exposed to the network. The values for the input of an instance are placed on the input units and the output of the network is compared with the desired output for this instance. Weights in the network are then adjusted slightly so that the output values of the network converge towards the values for the desired output.

Prior to the use of backpropagation to train multi-layer networks, Ivakhnenko proposed the first general working learning algorithm using feed-forward multi-layer perceptrons for supervised learning [93] alongside other early works on multi-layer perceptrons with a single hidden layer [100, 213]. Ivakhnenko subsequently proposed a deep network consisting of 8 layers [97], trained using the Group Method of Data Handling (GMDH) algorithm [96]. The units within these GMDH networks had activation functions that implemented Kolmogorov-Gabor polynomials [94] which were a more general form of other widely used activation functions used today. The method involved incrementally growing layers and training them using regression analysis [127] on a training set. The layers were then pruned using a validation set using Decision Regularisation to remove redundant units in the layer. This was followed by numerous applications of GMDH-style networks [45, 90, 95, 110, 111, 139, 222].

Hornik *et al.* demonstrated that standard multi-layer feed-forward networks were a class of universal approximators, being capable of approximating any Borel measurable function from a finite dimensional space, provided that enough hidden units were available [85]. In [146], feed-forward networks were trained using genetic algorithms to improve classification on sonar images over networks trained via backpropagation. Hagan & Menhaj showed that the Marquardt algorithm for non-linear least squares could be incorporated into the backpropagation algorithm to train feed-forward networks, demonstrating that this method was more efficient

for small networks of a few hundred weights [74]. Huang *et al.* developed the Extreme Learning Machine (ELM), which randomly assigned the weight values in the hidden layers of a feed-forward network and instead tuned the output weights; this allowed the networks to be trained extremely fast [87]. Liang *et al.* extended upon this approach, by presenting a fast online sequential learning algorithm for training ELMs [129]. The work of [65] looked at understanding why deep feed-forward networks were so difficult to train. Specifically, Glorot & Bengio wanted to understand why learning using gradient descent from random initialisation fails in deep feed-forward networks. This led to the proposal of a new initialisation scheme that allowed for significantly faster convergence in deep neural networks.

2.3.2 Convolutional Neural Networks

Fukushima proposed the first ever Convolutional Neural Network (CNN) in the form of the Neocognitron [56], which was based on the idea of simple and complex cells, with simple cells basically performing what is known as the convolution operation and complex cells performing average pooling. This model was inspired by a model proposed by Hubel & Wiesel which had found simple and complex cells in the visual cortex and proposed a model which cascaded these two types of cells for solving pattern recognition tasks [88]. Although architecturally, the Neocognitron shares many similarities with the modern CNN designs, it is not trained using supervised backpropagation; rather, it adopts local Winner-Take-All-based unsupervised learning rules during training [57].

The first use of backpropagation to a neural network specific application was described in [221]. Several related works followed this which further explored the use of backpropagation within neural network architectures [123, 126, 160]. LeCun *et al.* proposed [124], where the standard backpropagation algorithm was first applied to a CNN framework to recognise handwritten ZIP codes in mail. This framework would serve as an essential baseline for many modern CNN works

to follow and also introduced one of the most famous benchmarks in machine learning, the MNIST data set. Following this, CNNs were used across a variety of applications including fingerprint recognition [5] and object recognition [125]. Bengio *et al.* introduced a method called Greedy Layerwise Unsupervised Pre-training, in which a hierarchy of features was learned one layer at a time [8]. The approach utilised unsupervised feature learning to identify new transformations at each layer composed from transformations learned at the previous layer. This allowed each iteration of transformations to add one layer of weights to the deep neural network so that the layers could be assembled together to initialise a deep supervised predictor such as a neural network classifier.

However, for computer vision tasks, CNNs would still arguably take the backseat to other machine learning methods such as support-vector machines (SVMs) and decision forests which would typically outperform them on several benchmarks. The hallmark work of [117] would change the field through the introduction of the AlexNet framework for large scale image classification on the ImageNet data set [34]. This CNN-based framework introduced the now popular Rectified Linear Unit (ReLU) activation, combining this with dropout [203] and weight decay regularisation [118]. The combined result allowed AlexNet to achieve unrivalled performance on the image classification task. This was subsequently followed by the Visual Geometry Group Network (VGGNet) [201]. This network offered a deeper architecture and used smaller kernel spatial sizes of 3×3 windows which would go on to become a standard for subsequent network architectures. He *et al.* would create the next baseline CNN architecture which featured the use of "skip" connections and employed a residual learning framework [79]. The resulting framework was aptly named Residual Networks and enabled the training of networks with hundreds to thousands of layers by addressing a key limitation of vanishing gradients in (very) deep networks at the time. Since these developments, CNNS have been successfully applied by numerous works to a wide-variety of vision-related tasks including image classification [79, 86, 117, 201], object detection [63, 64, 170], semantic segmentation [131, 132, 135, 230] and depth estimation [41, 120, 134, 187].

2.4 Generative Neural Network Models

Generative neural network models are a subset of generative models which try to learn a data distribution given samples from a distribution. Generally, there are two major categories of generative neural network models: Variational Autoencoders (VAE) [108], and Generative Adversarial Networks (GAN) [68]. In this section, we briefly outline recent works in both categories. Whilst there are many works that approximate probability distributions, here we have described a selection that are most related to the work in this thesis. For the interested reader, we encourage them to refer to more comprehensive reviews in [7] and more recently [174].

2.4.1 Variational Autoencoders

Autoencoders are known for being an efficient method for reducing the dimensionality of data, which rapidly gained popularity since [82] first implemented the encoder-decoder setup using neural networks. Kingma & Welling introduced the Variational Autoencoder (VAE) that minimises the Variational Lower Bound by formulating it as an encoder-decoder scheme [108]. Since its introduction, VAEs have found a wide use of applications. The work of Salimans *et al.* introduced a variational approximation to the Markov Chain Monte Carlo (MCMC) method [185], proposing the Conditional Variational Autoencoder (CVAE) which models latent variables and data both conditioned to some random variables. Rezende *et al.* used variational inference to train a generative model using an approach which performed stochastic backpropagation through a latent Gaussian representation [172]. The work of Kulkarni *et al.* used a method to divide the learned latent space of a VAE into extrinsic and intrinsic components [119]. This allowed the extrinsic variables to represent controllable parameters of the image whilst the intrinsic parameters represented aspects of the image that were invariant to the extrinsic values. Walker *et al.* performed forward prediction from

static images using a probabilistic prediction framework using VAEs [219].

2.4.2 Generative Adversarial Networks

Generative Adversarial Networks [68] are a another subset of generative models - instead of explicitly trying to form an expression for the density function of the input data space, GANs learn to sample from the distribution by learning a mapping from some source of noise to the input data space. Following their introduction by Goodfellow *et al.* , GANs have faced various training issues due to the unstable nature of the minimax loss functions they were typically trained on. Several works since have contributed ways to ease training of GANs; most notably the work of Radford *et al.* explored architectural changes and utilised deep convolutional networks which opened up using regularisation methods such as batch normalisation [92], significantly improving on training stability and mitigating mode collapse [168]. Other works looked at modifying the objective function GANs were trained on. In particular, Arjovsky *et al.* proposed the Wasserstein Generative Adversarial Network which considered using the Earth-Mover's Distance (EMD) as an alternative loss function [2]; this better coupled training with quality of generated samples. This was followed up by a extension work by Gulrajani *et al.* which improved upon this by using gradient penalisation to enforce the Lipschitz continuity instead of the weight clipping method of the original work [70]. Modern works in the literature investigate the task of high resolution image generation [102, 228] by adopting a popular technique of progressively growing a GAN, which first tries to generate a low resolution image or estimate a low dimensional distribution, and gradually stack layers of higher resolutions, using the lower resolutions as a preconditioning. Recently, the conditioning of a GAN has been studied in order to improve stability of training. The work of Mescheder *et al.* showed that local convergence and stability properties of GAN training was tied to the eigenvalues of the Jacobian of its associated gradient vector field [143]. This work was followed up with a large scale study which investigated the convergence properties of various GAN training methods,

which found that simple descent-ascent update rules might not converge in a GAN setting [141].

Chapter 3

Preliminaries

In order to make this thesis as self-contained as possible, this chapter presents mathematical concepts that are commonly used throughout the remainder of the thesis. First, we give a brief outline of machine learning and its basic premise. We then detail some assumptions made about our models and data. Following this, we cover some of the common optimisation methods used including gradient descent and its stochastic variant. We provide an overview of ensemble methods related to this thesis, including various forms of decision forests and boosting algorithms. Subsequently, we summarise various deep learning concepts including various neural network layer components, how these components are initialised as well as various methods for regularising these deep models. Finally, we conclude this chapter by outlining the software packages used to implement the frameworks described in this thesis.

3.1 Model and Data

In the context of learning from data, we make some general assumptions about the nature of the data itself as well as the functions obtained from our learning algorithms. Here, we detail the key assumptions about our data as well as our model which learns a function to map input data to target outputs.

3.1.1 Independent and Identically Distributed Data

A key ability that a machine learning algorithm should possess is the ability to generalise to new data that was not seen during training. Generally, we assume there is some common structure between the data in order to generalise from the training set to a test set. We typically make a *i.i.d* assumption; that is, data is independently and identically distributed. In other words, we assume that each sample is generated independently from other samples, with each sample drawn from the same distribution p_{data} [29]. We can express this formally as:

$$p_{data}(X, y) = \prod_i p_{data}(X_i, y_i) \tag{3.1}$$

3.1.2 Model Smoothness

Smoothness is an underlying assumption in machine learning, since prediction is feasible when the behaviour of a system is locally smooth. A key assumption we make is that the "true" model we are trying to learning is smooth - the underlying function that the model represents which maps data features to target outputs is not too steep anywhere: an input X produces an output Y and a corresponding input close to X should produce an output proportionally close to Y (*i.e.* similar samples should have similar target outputs). Indeed, the

accuracy of correct predictions improves with increasing smoothness of the local neighbourhood between samples from the data [42]; the addition of more training data makes this smoothness more probable since these neighbourhoods are now smaller.

3.1.3 Limited Model Complexity

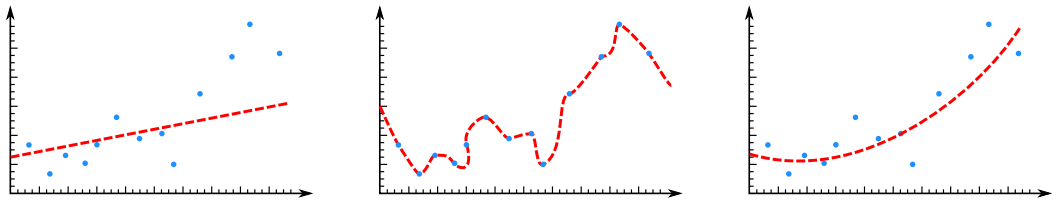
We assume that the model is not overly complex and can be represented fairly simply. This assumption follows Occam's razor [10], where given two explanations for an occurrence, the explanation that requires less speculation is usually better. This assumption allows for regularisation methods to be applied to our model which correspond to priors from a Bayesian viewpoint.

3.1.4 Model Selection

For supervised learning tasks, when learning in high-dimensional input spaces with limited training data (as is often the case with image data), we make a simplifying assumption on the structure of the best model f^*. More specifically, for supervised learning algorithms, we assume that f^* (or a good enough approximation) belongs to a family of candidate models $\mathcal{F}$, also known as hypotheses, of restricted structure. Under this setting, model selection is defined as finding the best model amongst $\mathcal{F}$ given the training data.

3.1.4.1 Model Bias and Variance

When selecting models, generally there are two important aspects to consider: its bias and variance. The *bias* of a model is the error due to incorrect assumptions made by the model's learning algorithm and describes the overall mean error of the model. A high model bias indicates that the model may not be able to infer

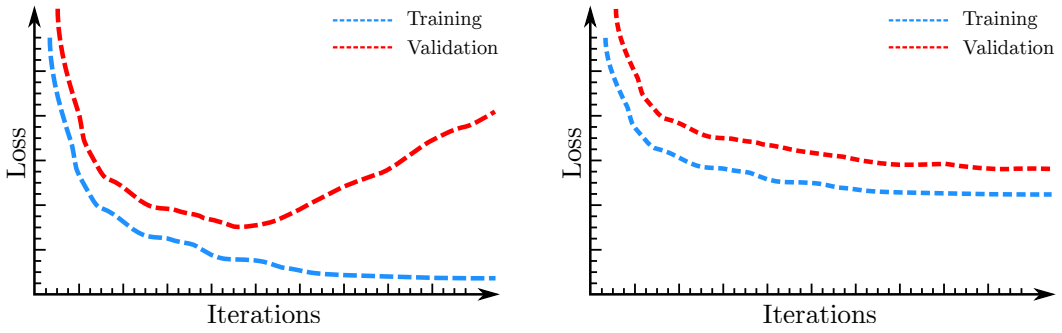


(a) Too Simple **(b)** Too Complex **(c)** Good

Figure 3.1: (a) When the model is too simple, it lacks the capacity to capture the structure of the data and relate this to the target output resulting in poor predictive performance on both seen and unseen data. (b) When the model is too complex, it mistakenly relates random noise to the underlying structure of the data and relates this to this target output which leads to poor performance on unseen data. (c) A good model finds a compromise between bias and variance in which there is enough complexity in the model to capture the regularities of the data, but not too much such that small irregularities in the training data are also captured.

the structure of the data and relate this to the target output. This is illustrated in Fig. 3.1a. The *variance* of a model describes the error due to sensitivity to small irregularities in the training data. A model with high variance can fit to the random noise in training data, instead of capturing the relations between the data's features and target output. This is illustrated in Fig. 3.1b.

Ideally, we would like to select a model which minimises both bias and variance. However, the *bias-variance tradeoff* makes this a difficult task; it describes a property in predictive models where models with a lower bias in parameter estimation tend to exhibit higher variance in parameter estimates across samples and vice versa. This makes it difficult to choose a model which both accurately captures the regularities in the training data but also generalises well to unseen data: complex models have fitting power which enables them to represent the training data well, however they are at risk of over-fitting to the noise in the training data. In contrast, simpler models tend to under-represent the training data and although offer lower variance predictions between training and unseen data, they may fail to capture key relevant relations between features in the training data and target outputs. Good model selection involves selecting a model that offers a compromise between bias and variance (shown in Fig. 3.1c).



(a) Over-fitting **(b)** Under-fitting

Figure 3.2: (a) A model over-fitting to the training data. Initially, validation loss decreases proportionally with training loss as the model learns from the data. As training progresses, the model shows signs of over-fitting to the training data and the validation loss begins to diverge from the training loss and rapidly increases. (b) A model under-fitting to the training data. Over the training schedule, although the validation loss never drastically diverges from training loss, the model lacks the capacity to fully learn from the training data resulting in poor predictive performance on the training data and subsequently the validation data.

3.1.4.2 Over-fitting and Under-fitting

The ability for a model to generalise is a key criteria that defines its ability to perform robust pattern recognition. For a learning algorithm to be useful, not only does it need to perform well on the training data it has learned from, but it must be able to generalise to samples *unseen* during the learning process. An over-fitted model will be unable to generalise to unseen samples since it has been too finely tuned towards its training set; in this case, the model is considered to contain more parameters than can be justified by its training data [43]. Over-fitted models tend to perform very well when considering samples from its training set since they have essentially memorised the entire training set, but tend to perform poorly when shown samples outside of its training set. This is due to the model capturing the noise in its training data. This is illustrated in Fig. 3.2a. Generally, over-fitted models exhibit low bias but high variance. On the other hand, a model is said to be under-fitted if it does not possess the capacity to correctly capture the underlying trend or structure of its training data. Intuitively, under-fitting occurs in models that cannot fit the data well enough because the model

is too simple. Under-fitted models tend to show low variance, but a high bias. This is illustrated in Fig. 3.2b. Both over-fitted and under-fitted models result in poor performance. Generally, in deep models, over-fitting is more common than under-fitting and it is important to utilise a *validation* set to assess a model's true predictive power.

3.2 Learning from Data

Many problems that we wish to solve in the real world can be thought of as finding a mapping from some input space X to a corresponding output space Y (*i.e.* we are trying to find some function f which describes this mapping $f : X \rightarrow Y$). For example, in image classification tasks we require some image data X to be correctly classified with their corresponding class labels Y. In this case, we can specify Y to be some probability value in the interval of $[0, 1]$ which indicates the probability of that an image X contains an object of the required class. However, specifying a function f which performs this mapping is non-trivial using conventional means. It is not clear how one could hand-engineer a function which recognises an image of a cat.

3.2.1 Maximum Likelihood Estimation

One of the most well-known approaches for modelling data is Maximum likelihood estimation (MLE). This method estimates parameters of a statistical model given observations of some data, through finding parameter values which maximise the likelihood of the observed data, given the parameters of the model. MLE defines a probabilistic model that is controlled by a set of parameters θ. This model gives a probability distribution $p_{model}(\boldsymbol{x}; \theta)$ over the examples $\boldsymbol{x}$. We can then find the correct value of $\theta \in \Theta$ where Θ defines the set of all possible values. Assuming all samples are drawn from a data distribution p_{data} and are *i.i.d*, we can define the likelihood function as:

$$\mathcal{L}(\theta) = \prod_i p_{model}(X_i; \theta) \tag{3.2}$$

Hence, the estimator for maximum likelihood estimation as:

$$\theta^* = \arg\max_{\theta \in \Theta} \prod_i p_{model}(X_i; \theta) \tag{3.3}$$

Equation 3.3 indicates that the objective of maximum likelihood estimation is to choose parameters θ^* which maximises the probability that the given model will generate the training data. We can do this by finding the values for θ such that the first-order derivatives equal zero, *i.e*:

$$\frac{\partial \mathcal{L}}{\partial \theta} = 0 \tag{3.4}$$

Normally, instead of optimising Equation 3.2, we exploit the monotonically increasing property of the logarithm to optimise the log likelihood, which is maximised for the same value of θ. Maximising the log likelihood addresses two key issues that can arise in digital numerical computation; namely numerical underflow as a result of taking the product of several factors in the interval of $[0, 1]$ as well as conveniently decomposing this product into a sum of separate examples which is much more convenient from a mathematical perspective. Hence, in practice we find the estimator which maximises the log likelihood:

$$\theta* = \arg\max_{\theta \in \Theta} \sum_{i} \log p_{model}(X_i; \theta) \tag{3.5}$$

We can recover the true probability distribution p_{data} using maximum likelihood estimation, if we fulfil two criteria:

1. $p_{model}(\boldsymbol{x}; \theta)$ possesses enough model capacity such that $p_{data}(\boldsymbol{x})$ is contained in the set of samples generated by $p_{model}(\boldsymbol{x}; \theta)$.
2. We have an infinite number of samples drawn from p_{data} which are independent and identically distributed.

Practically of course, the training set contains a finite set of data in which case the maximum likelihood estimator must also generalise well. This is generally infeasible when very little data is available. In the case of very little data being available, maximum likelihood estimation of parametric models can perform quite poorly as its sample set is simply too small to properly represent the true distribution it is trying to estimate.

3.2.2 Supervised Learning

In supervised learning, we have a training data set of n examples with their corresponding labels $(x_1, y_1), (x_2, y_2), ...(x_n, y_n)$. Again, we assume our data is made of independent and identically distributed $(i.i.d)$ samples drawn from a data distribution p_{data} $(i.e.\ (x_i, y_i) \sim p_{data} \forall i)$. f is then a function which learns the mapping from X to Y, $F : X \rightarrow Y$. In this case, f belongs to a class of potential functions $\mathcal{F}$ that we can choose from. Supervised learning involves finding the f which best fits the training examples. This learning is enabled through a training signal commonly known as a loss function L. Here, $L(\hat{y}, y)$ denotes a function which outputs a scalar-value which measures the dissimilarity between the annotated label y_i (commonly referred to as *ground truth*) and the predicted label output by $\hat{y} = f(x_i)$. Ideally, we would like to find $f^* \in \mathcal{F}$ that satisfies the following:

$$f^* = \arg\min_{f \in \mathcal{F}} E_{(x,y)\sim D}[L(f(x), y)] \tag{3.6}$$

This indicates that in order to find f^*, we need to minimise the expected loss over the data generating distribution P_{data}. f^* describes the mapping function we could learn if we had access to an infinite amount of training data; once f^* is learned we can then use it to map any element from X to Y. However, this optimisation problem is intractable since we do not have access to all the elements in the data distribution P_{data}. In practice, we use the $i.i.d$ assumption of our training data so that we can approximate the expected loss in Equation 3.6 by averaging loss over available samples from the training data:

$$f^* \approx \arg\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} L(f(x_i), y_i) \tag{3.7}$$

3.3 Optimisation

Optimisation lies at the heart of machine learning; a majority of machine learning problems can be reduced to optimisation problems. In this section, we detail some of the optimisation techniques that are relevant to the work presented in this thesis.

3.3.1 Gradient Ascent

In Section 3.2.1, we discussed a closed-form optimisation solution in the form of maximum likelihood estimation. In other words, we can define an objective function given by the log likelihood:

$$\mathcal{L}(\theta) = \sum_i \log p_{model}(X_i; \theta) \tag{3.8}$$

solving for the optimisation problem:

$$\begin{aligned} &\text{maximise } \mathcal{L}(\theta) \\ &\text{subject to } \theta \in \Theta \end{aligned}$$

Sometimes, as in the case of maximum likelihood estimation, this can be achieved simply by solving $\nabla_\theta \mathcal{L}(\theta) = 0$ for θ. However, there is often no closed-form that can be recovered to Equation 3.8 in which case an iterative optimisation method must be employed to obtain the solution.

One such iterative optimisation method is *gradient ascent* for maximising likelihood functions, or *gradient descent* if we are minimising loss functions. The key idea behind this algorithm is that $\nabla_\theta \mathcal{L}(\theta)$ gives the direction where $\mathcal{L}$ increases most rapidly in a local neighbourhood around θ. Hence, in order to optimise $\mathcal{L}$, we should iteratively take small steps in the direction given by the gradient $\nabla \mathcal{L}$.

Using the gradient ascent algorithm, we can compute the update to apply to θ for iteration t using the following:

$$\theta^{(t)} = \theta^{(t-1)} + \alpha(t)\nabla_\theta \mathcal{L}(\theta) \tag{3.9}$$

where $\alpha(t)$ controls the step size to be taken in the direction given by $\nabla_\theta \mathcal{L}(\theta)$ (also commonly known as the *learning rate*) [27].

3.3.2 Stochastic Gradient Ascent

Consider a data set with a large redundancy in examples; it may only take a small number of examples from this data set to get a good representation of the data set. Subsequently, to get a good estimate of the direction of the gradient from the current value of θ, only a small number of examples are required as well. Using gradient ascent, we would need to compute the gradients contributed by every example in the data set; this would be the case even if some examples are the exact same as others in the data set. Moreover, we know that the mean standard error of our estimate of the true gradient will decrease at a rate of slower than linearly as more examples are added. Since computation of the estimate increases linearly, it is usually not computationally cost effective to use the entire data set to estimate the gradient.

Fortunately, an alternative algorithm to gradient ascent resolves this problem. *Stochastic Gradient Ascent* [27] is an approximation of the gradient ascent algorithm and uses a modified update rule:

$$\theta^{(t)} = \theta^{(t-1)} + \alpha(t)\nabla_\theta \sum_{i \in \mathbb{B}} \log p_{model}(X_i; \theta) \tag{3.10}$$

where $\mathbb{B}$ is a randomly selected subset of $1, ..., m$. This randomly selected set of examples are called a *mini-batch*. Typically, mini-batch sizes range from 1 to 128. Stochastic gradient ascent alleviates the computational load issues associated

with gradient ascent by assuming that a randomly (stochastically) selected set of samples represents the entire data set well and thus can provide accurate estimates of the true gradient. Stochastic gradient ascent is also believed to offer other benefits aside from reducing redundancy in computations. The main additional benefit would be improving learning through injection of noise which allows saddle points or local maxima to be escaped [59] and incidentally reducing over-fitting.

So far, we have looked at optimisation in terms of ascending the log likelihood. In practice however, optimisation is most commonly performed as stochastic gradient descent (SGD). Instead of maximising the log likelihood, we can think of learning as descending a cost function instead. Referring back to ascending the log likelihood, we can of course treat the problem as descending the negative log likelihood and estimate the same values for θ.

3.3.3 Newton's Method

Newton's method (also known as the Newton-Raphson method) is an efficient approach to gradient ascent. It is an iterative root-finding algorithm which uses the first few terms of the Taylor series of a function f around the neighbourhood of a candidate root. Given a function f, the Taylor series of f about the point $x = \hat{x} + \epsilon$ is given by:

$$f(\hat{x} + \epsilon) = f(\hat{x}) + f'(\hat{x})\epsilon + 0.5f''(\hat{x})\epsilon^2 + ... \tag{3.11}$$

where $\hat{x}$ is the estimate of x and ϵ is an offset needed to land closer to the root starting at an initial guess $\hat{x}^0$. Assuming that f is convex around the neighbourhood of the true solution x^*, we can approximate f using the first-order Taylor series expansion for the current iteration i around the estimate of the model parameters $\hat{x}^i$:

$$f(\hat{x}^i + \epsilon) \approx f(\hat{x}^i) + f'(\hat{x}^i)\epsilon \tag{3.12}$$

This can be expanded to the gradient of f, where the solution x^* should exist where $f'(x^*) = 0$, where we can again approximate this function using the first-order Taylor series expansion for the gradient of the function f for the current iteration i:

$$f'(\hat{x}^i + \epsilon) \approx f'(\hat{x}^i) + f''(\hat{x}^i)\epsilon \tag{3.13}$$

We start with an initial estimate $\hat{x}^0$ and set $f'(\hat{x}^0 + \epsilon) = 0$. Solving Eq. 3.13 for $\epsilon \equiv \epsilon_0$ yields the update step needed to land closer to the root starting from the initial guess $\hat{x}^0$:

$$\epsilon_0 = -\frac{f'(\hat{x}^0)}{f''(\hat{x}^0)} \tag{3.14}$$

This update can then be iteratively applied to obtain an update procedure:

$$\hat{x}^{n+1} = \hat{x}^n - \frac{f'(\hat{x}^n)}{f''(\hat{x}^n)} \tag{3.15}$$

For $n = 1, 2, 3, ...$ and, given an initial good estimate of the root's position $\hat{x}^0$, safe convergence of Newton's method is called an *approximate zero*.

3.3.4 Gauss-Newton Method

For minimisation problems in which the objective function is expressed as a sum of squares, it is often beneficial to take advantage of the special structure of the problem. We can express the objective function as the norm squared of a residual error $r(x)$:

$$f(x) = r(x)^2 \tag{3.16}$$

The first and second-order derivatives of Eq. 3.16 are:

$$\begin{aligned} f'(x) &= 2r(x)r'(x) \\ f''(x) &= 2(r(x)r''(x) + (r'(x))^2) \end{aligned} \tag{3.17}$$

The Gauss-Newton method uses the approximation to the second derivative $f''(x)$. It assumes that the Hessian of the residual $r''(x)$ becomes more and more

negligible as the error reduces and hence drops this term from $f''(x)$ in Eq. 3.17. The approximation to $f''(x)$ is then:

$$f''(x) \approx 2(r'(x))^2 \tag{3.18}$$

Hence the update procedure for the Gauss-Newton method becomes:

$$\hat{x}^{n+1} = \hat{x}^n - \frac{2r(\hat{x}^n)r'(\hat{x}^n)}{2(r'(\hat{x}^n))^2} \tag{3.19}$$

This allows the Gauss-Newton method to be an efficient approximation of Newton's method: this approximation does not lose the robustness of Newton's method but significantly reduces the average computation cost due to removing the requirement of computing second-order derivatives.

3.4 Ensemble Methods

Ensemble methods are meta-algorithms which combine several machine learning methods (or base learners) into a single predictive model with the intent of decreasing variance, bias or improving overall predictions. Generally, ensemble methods can be divided into two groups:

- Sequential ensemble methods generate base learners iteratively, with each base learner conditioned upon the information learned by the existing ensemble (*e.g.* Adaptive Boosting). These methods exploit the dependence between base learners in the ensemble and their main mechanism for improving overall performance is by assigning higher weights to previously misclassified samples for each iteration of base learner being learned.

- Parallel ensemble methods generate base learners in parallel (*e.g.* Random Forests). These methods exploit the independence between base learners since variance (and in general, error) can be reduced by averaging across multiple base learners.

In this section, we cover some details related to the ensemble methods relevant to this thesis.

3.4.1 Derivative-Free Optimisation

In the previous section, we discussed some optimisation methods which utilised gradient information to update the parameters in a model. An alternative approach to solving an optimisation problem is to use derivative-free optimisation. As the name implies, this approach does not rely on derivative information to find optimal solutions. This could be due to the derivative of the objective function

f being unavailable, unreliable or impractical to acquire (*i.e.* f could be non-smooth, expensive to evaluate, or possess noise such that methods which rely on derivatives are of little use).

Finding the optimal points for these types of problems is referred to as derivative-free optimisation; this involves using a stochastic hill-climbing method that effectively guesses and checks potential θ which are randomly selected from some distribution. This method involves drawing a large number of θ at random, checking each one and making a selection based on the parameter that maximises the objective. This approach ties derivative-free methods closely to black-box optimisation since both approaches do not obtain any gradient information for optimisation [4].

Ensemble methods commonly use derivative-free optimisation techniques to train their models, particularly in the case of those using decision trees as their base learner since it is infeasible to obtain derivatives for the routing operations they perform (although there also exist gradient-based optimisation methods such as Gradient Boosted Trees [55]).

3.4.2 Decision Forests

Decision forests are an ensemble learning method for performing tasks such as classification and regression. This approach constructs a series of decision trees during the training schedule. Each individually constructed decision tree outputs a distribution which is computed from observed class labels sampled from the training data set and these distributions are subsequently aggregated to form an overall prediction for the entire ensemble as whole.

3.4.2.1 Decision Trees

The basic premise for a decision tree is to use a splitting criteria to separate out data. This is achieved through routing of data through the internal *decision nodes* of the decision tree which determines the path taken to the terminating *leaf node*. The internal nodes of each decision tree are a set of decision nodes, $\mathcal{D} = \{d_0, \cdots, d_{N-1}\}$, each of which holds a decision function $d_k(\boldsymbol{x}; \theta_k)$, where θ_k are the parameters for decision node k. For binary decision trees, the decision function for each decision node is defined as $d_k(\boldsymbol{x}; \theta_k) : \mathcal{X} \rightarrow [0, 1]$, which routes a sample to its left or right child node. Collectively the decision nodes route data, x, through the tree until a terminal leaf node is reached: $\ell = \delta(x; \Theta)$. Here, δ signifies the routing function which directs data x to the terminal leaf node ℓ, given the tree's decision node parameters Θ. This is illustrated in Fig. 3.3. The leaf nodes contain classification information in the form of class label probability distributions, $\boldsymbol{q} = Q(\ell)$. These distributions are formed from the training data and accompanying ground truth class labels. For a terminal leaf node ℓ, the stored probability for class j is given as:

$$q_{\ell,j} = \frac{\sum_i c_{ij}}{n_\ell} \tag{3.20}$$

where n_ℓ is the total number of samples routed into leaf node ℓ and c_{ij} is the observed class label for sample i in class j *i.e*:

$$c_{ij} = \begin{cases} 1, & \text{if sample } i \text{ has class label } j \\ 0, & \text{otherwise} \end{cases} \tag{3.21}$$

Typically, decision trees are trained greedily at the local decision node level, where each decision node selects a decision function which best splits the incoming data according to some criteria such as Shannon entropy or the Gini index [16].

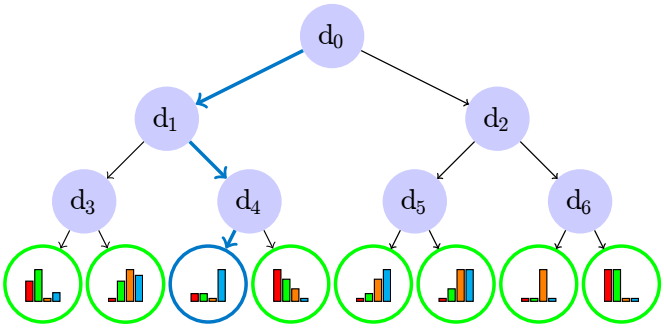


Figure 3.3: A typical decision tree layout with a data sample being routed highlighted in blue

3.4.2.2 Ensembles of Decision Trees

An ensemble of $\mathcal{T}$ number of decision trees forms a decision forest. Each decision tree in the decision forest outputs a prediction about the data in the form of a marginal probability distribution as described in Eq. 3.20. The final prediction offered by the decision forest is the average of the marginal probability distributions delivered by each decision tree:

$$P(\text{class } j|\boldsymbol{x}, \boldsymbol{\Theta}, \boldsymbol{Q}) = \frac{1}{\mathcal{T}} \sum_{t=1}^{\mathcal{T}} Q^t(\delta^t(\boldsymbol{x}; \Theta^t))_j \tag{3.22}$$

where Q^t, δ^t and Θ^t are the respective distributions, routing functions and parameters of tree t, while $\boldsymbol{\Theta}$ and $\boldsymbol{Q}$ are the collected parameters of decisions and distributions of the ensemble.

3.4.3 Induction of Decision Trees

Training decision trees involves locally searching and choosing the best split function at each node in the decision tree. The most common methods for training a decision tree are the Iterative Dichotomiser 3 (ID.3) [166] and C4.5 algorithms (an extension to the earlier ID.3 algorithm) [167]. Both ID.3 and C4.5 utilise the concept of information entropy as a criteria for selecting split nodes during the

tree construction process.

3.4.3.1 Information Entropy

Information entropy is used to measure the amount of information there is in a state. In general, the more uncertainty or randomness there is in the state, the more information it will contain (increasing information will result in a decrease in uncertainty or entropy). Whilst training decision trees, at each decision node, the state represents the class distribution of observed labels from the data set. In this context, information entropy for a discrete random variable X is measured for each possible data point i using the negative logarithm of its probability mass function p_i: $H(X) = -\sum_i p_i \log_b p_i$, where b is the logarithm base (usually 2).

3.4.3.2 Iterative Dichotomiser 3 Algorithm

One of the earliest algorithms for constructing decision trees was the Iterative Dichotomiser 3 (ID.3) algorithm [166]. The ID.3 algorithm starts with a root node and recursively splits the data set into subsets. For each decision node, a set of potential attributes A, is randomly selected for the node. From this, each potential parameter in the set is independently evaluated and the weighted information entropy of the potential child nodes is measured. The parameter which produces the lowest weighted entropy (and subsequently offers the largest information gain) is selected. The set of data is then partitioned according to the selected parameter to produce subsets of the data. This is illustrated in Fig. 3.4.

Information Gain The splitting criteria that ID.3 uses for determining the local optimal decision node split is to select a decision which maximises the *information gain*. For a split function being considered, the information gain is measured as the difference in weighted information entropy (or Shannon entropy [195])

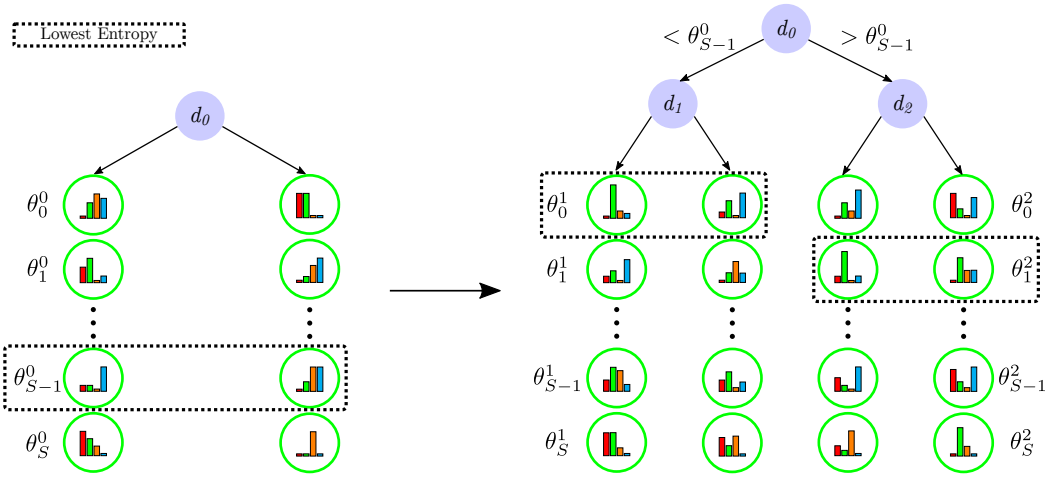


Figure 3.4: The ID3 algorithm selects a decision function based on an entropy minimising criteria. A decision tree is recursively constructed in this manner.

between the pre-split observed class distributions and post-split observed class distributions in the resulting child nodes. Formally, the expected information gain IG is the change in information entropy H from its prior state to its state after it takes some information as a result of splitting on a selected split attribute a.

Let T denote the set of all training samples, with each sample in the form $(\boldsymbol{x}, y) = (x_1, x_2, x_3, ..., x_k, y)$, where x_a is the value of the a^{th} attribute of sample $\boldsymbol{x}$ and y is the corresponding class label. The information gain for the attribute a can be defined in terms of the information entropy $H(\cdot)$, which is the difference between the *a priori* information entropy $H(T)$ and the conditional entropy $H(T|a)$:

$$IG(T, a) = H(T) - H(T|a) \tag{3.23}$$

The ID.3 algorithm will recursively apply the same procedure to each child node created and stop when one of the following stopping criteria is met:

- a subset only contains elements of the same class. In this case, a leaf node

is created and labelled with the class of the elements in the subset.

- there are no attributes left to select. In this case, a leaf node is created and labelled with the most common class amongst elements in the subset.
- the subset contains no elements. In this case, a leaf node is created with the most common class amongst elements in the parent node.
- a maximum specified tree depth is reached. In this case, a leaf node is created and labelled with the most common class amongst elements in the subset

Algorithm 3.1 summarises the full procedure for the ID3 algorithm for constructing a single decision tree. Whilst the ID.3 algorithm was a powerful algorithm

Algorithm 3.1 ID3 Algorithm

Require: $\mathcal{N}$: training set
Require: Feature pool size S
Require: Maximum tree depth D_{max}
  while $m < D_{max}$ **do**
    In parallel:
    for all $s \in \{1, ..., 2^{D_{max}-1}\}$ **do**
      In parallel:
      for all $p \in \{1, ..., S\}$ **do**
        Choose an attribute a
        Calculate total weighted entropy H for both child nodes
      end for
      Choose value for a for node d that minimises H
    end for
  end while=0

it had some shortcomings. Attributes must be nominal values, the data set was not allowed to include missing data and the algorithm had a tendency to over-fit to the training data.

3.4.3.3 C4.5 Algorithm

The ID.3 algorithm was extended into the C4.5 algorithm, which made a number of improvements over the the original ID.3 algorithm. Similar to the ID.3 algorithm, C4.5 builds decision trees from a set of training data using information entropy as a splitting criteria for selecting the internal decision split functions in the tree. However, the C4.5 algorithm addressed several of the limitations of the original ID.3 algorithm.

A key improvement was that C4.5 allowed for handling of both continuous and discrete attributes; for continuous attributes, C4.5 uses a threshold which partitions the list of samples to those whose attribute value lies above the threshold and those whose attribute are less than or equal to it. Additionally, to account for ID.3's limitation to being overly sensitive to features with large values, it used the information gain ratio for selecting decision functions. This was a modification of the information gain splitting criteria which reduced the bias towards favouring attributes with a large number of distinct values.

Information Gain Ratio The information gain ratio describes the ratio between the mutual information of two random variables and the entropy of one of them. This value is guaranteed to be in the range of $[0, 1]$, except for cases where it is undefined. We first define the split information value SI which represents the potential information generated when splitting the training data T into m partitions, which corresponds to m outcomes on the chosen attribute a (*i.e.* the intrinsic value of the attribute):

$$SI(T, a) = -\sum_{j=1}^{m} \frac{T_j}{|T|} \log_b \frac{T_j}{|T|} \tag{3.24}$$

Where T_j corresponds to the j^{th} partition of the training set T. Hence, the split information penalises information gain criteria; it is used as a normalisation term which scales the information gain depending on the number of choices available

for a given attribute. We can subsequently compute the gain ratio GR as the ratio between the information gain and split information:

$$GR(T, a) = \frac{IG(T, a)}{SI(T, a)} \tag{3.25}$$

3.4.3.4 C5.0 Algorithm

The C5.0 Algorithm further extended upon the C4.5 algorithm. The largest improvement was towards C4.5's speed where C5.0 was significantly faster by several orders of magnitude. C5.0 offered further memory usage improvements via training of smaller decision trees when compared to C4.5. Additionally, support for boosting was added as well as a weighting mechanism for handling different cases and misclassification types.

3.4.4 Boosting

Boosting algorithms are a set of related ensemble learning methods which iteratively construct a *strong classifier* by combining a series of *weak classifiers* (or weak learners). The core concept of boosting algorithms relates together strong and weak learners through a notion of weak and strong learnability from the Probably Approximately Correct learning framework. The choice of the weak learner can be any classifier, although decision trees and kernel-based classifiers such as support vector machines have historically been a popular choice [52, 54, 188].

3.4.4.1 Probably Approximately Correct Learning

Probably approximately correct (PAC) learning theory helps analyse whether a learner $\mathcal{H}$ will *probably* output an *approximately correct* classifier, and under what conditions this occurs. The notion of weak learnability was first introduced

in [103] which first considered the equivalence between weak and strong learnability. First we give the definitions of weak and strong learnability as follows:

Definition 3.4.1 *(Weak PAC Learnability). A concept class $\mathcal{C}$ is **weakly PAC learnable** using a model class $\mathcal{F}$ if there exists a learner $\mathcal{H}$ and a value γ ¿ 0 such that for any $c \in \mathcal{C}$, for any distribution P over the input space, for any $\delta \in (0, 0.5)$, given access to a polynomial (in $1/\delta$) number of examples drawn i.i.d. from P and labelled by c, $\mathcal{H}$ outputs a function $f \in \mathcal{F}$ such that with probability at least $1 - \delta$, $err(f) \leq 0.5 - \gamma$.*

Definition 3.4.2 *(Strong PAC Learnability). A concept class $\mathcal{C}$ is **strongly PAC learnable** using a model class $\mathcal{F}$ if there exists a learner $\mathcal{H}$ such that for any $c \in \mathcal{C}$, for any distribution P over the input space, for any $\epsilon \in (0, 0.5)$ and $\delta \in (0, 0.5)$, given access to a polynomial (in $1/\epsilon$ and $1/\delta$) number of examples drawn i.i.d. from P and labelled by c, $\mathcal{L}$ outputs a function $f \in \mathcal{F}$ such that with probability at least $1 - \delta$, $err(f) \leq \epsilon$.*

In other words, the learner $\mathcal{H}$ will output a classifier with probability bounded by δ which will achieve an error over the distribution of inputs bounded by ϵ. If we know that a target concept is PAC-learnable, we can bound the sample size n required to probably learn an approximately correct classifier:

$$n \geq \frac{1}{\epsilon}(\ln |\mathcal{F}| + \ln \frac{1}{\delta}) \tag{3.26}$$

From Eq. 3.26, we observe the following: as the allowable error decreases, the required sample size must increase. Likewise, n increases with the probability of an approximately correct learner and with the size of the model space $\mathcal{F}$ (the set of models that the algorithm considers). Hence, n represents the *sample complexity*; that is, the minimum number of training samples necessary to successfully learn a target concept, given an error ϵ and probability of an approximately correct

learner δ [212].

3.4.4.2 Discrete AdaBoost

The Adaptive Boosting (AdaBoost) algorithm [188] can be used to improve the performance of any given learning algorithm. Like other boosting methods, it enables the construction of a strong classifier using a set of weak classifiers, where the resulting strong classifier has an improved performance over its weak classifier components. The AdaBoost algorithm achieves this by selecting and combining discriminative features using an iterative process that weights training samples based on how difficult they are to classify.

The following describes the Discrete AdaBoost algorithm for a two-class classification problem. Suppose we have a set of training samples $(x_1, y_1), (x_2, y_2), ...(x_N, y_N)$ where x_n refers to a training sample from the sample space X and y_n is its corresponding class label $Y \in \{-1, +1\}$. Our boosted classifier is composed of a set of weak classifiers $\{h_1, ..., h_K\}$, each of which outputs a prediction $h_k(x_i) \in \{-1, +1\}$ for each training sample. Each iteration constructs and adds a weak classifier to our boosted classifier $\mathcal{H}$ where after the $(k-1)$-th iteration the boosted classifier can be expressed as a linear combination of the weak classifiers:

$$\mathcal{H}_{k-1}(x_i) = \alpha_1 h_1(x_i) + ... + \alpha_{k-1} h_{k-1}(x_i) \tag{3.27}$$

For the k-th iteration, we want to boost $\mathcal{H}$ such that it improves in performance after the k-th weak classifier is added to the ensemble:

$$\mathcal{H}_k(x_i) = \mathcal{H}_{k-1}(x_i) + \alpha_k h_k(x_i) \tag{3.28}$$

Hence, we would like to determine the best choice for the weak classifier h_k as well as its weight value when being added to the boosted classifier $\mathcal{H}_k$. At iteration k, the total loss of the strong classifier L_k is the sum of its exponential loss on

each sample:

$$L_k = \sum_{i=1}^{N} \exp^{-y_i \mathcal{H}_k(x_i)} \tag{3.29}$$

Initially, each training sample is assigned a weight value w_i^k which determines their contribution to the expected loss of the strong classifier. Initially, w is uniform across all samples and is adaptively adjusted over iterations according to the loss by the strong classifier (*i.e.* we let $w_i^1 = 1$ and $w_i^k = \exp^{-y_i \mathcal{H}_{k-1}(x_i)}$ for $k > 1$ for each training sample x_i). Hence, the total weighted loss of $\mathcal{H}$ at the k-th iteration is given by:

$$L_k = \sum_{i=1}^{N} w_i^k \exp^{-y_i \alpha_k h_k(x_i)} \tag{3.30}$$

Eq. 3.30 can be split into correctly classified samples and misclassified samples:

$$L_k = \sum_{y_i = h_k(x_i)} w_i^k \exp^{-\alpha_k} + \sum_{y_i \neq h_k(x_i)} w_i^k \exp^{\alpha_k} \tag{3.31}$$

$$= \sum_{i=1}^{N} w_i^k \exp^{-\alpha_k} + \sum_{y_i \neq h_k(x_i)} w_i^k \left(\exp^{\alpha_k} - \exp^{-\alpha_k}\right) \tag{3.32}$$

Since the only term in Eq. 3.32 that depends on the selection of the weak classifier h_k is $\sum_{y_i \neq h_k(x_i)} w_i^k$, we select the weak classifier that minimises the total weighted loss L_k of the boosted classifier $\mathcal{H}_k$ at iteration k.

Selecting our weak classifier h_k allows its weight value α_k to be determined. For this, we again wish to choose α_k which minimises the total weight loss L_k. Differentiating the total error with respect to α_k, setting the partial derivative to zero and solving for α_k yields:

$$\alpha_k = \frac{1}{2} \ln \left(\frac{\sum_{y_i = h_k(x_i)} w_i^k}{\sum_{y_i \neq h_k(x_i)} w_i^k} \right) \tag{3.33}$$

At iteration k, the weighted classification error ϵ_k of the weak classifier is defined as the ratio of the sum of weights of incorrectly classified samples to the total

sum of weights of all samples (*i.e.* $\epsilon_k = \sum_{y_i \neq h_k(x_i)} w_i^k / \sum_{i=1}^N w_i^k$). Hence, Eq. 3.33 can be expressed as:

$$\alpha_k = \frac{1}{2} \ln \left(\frac{1 - \epsilon_k}{\epsilon_k} \right) \tag{3.34}$$

This derives the AdaBoost algorithm. For the k-th iteration, we summarise the procedure of adding a weak classifier to the ensemble as follows:

1. choose the weak classifier h_k which minimises the total weighted error $\sum_{y_i \neq h_k(x_i)} w_i^m$
2. use the total weighted error to compute the classification error rate $\epsilon_k = \sum_{y_i \neq h_k(x_i)} w_i^k / \sum_{i=1}^N w_i^k$
3. use ϵ_k to calculate the weight value assigned to the weak classifier h_k, $\alpha_k = \frac{1}{2} \ln \left(\frac{\sum_{y_i = h_k(x_i)} w_i^k}{\sum_{y_i \neq h_k(x_i)} w_i^k} \right)$
4. update the boosted classifier using α_k and h_k: $\mathcal{H}_k = \mathcal{H}_{k-1} + \alpha_k h_k$

3.4.5 Gradient Boosted Trees

Gradient Boosted Trees (GBT) [54] are another popular ensemble method that utilises the gradient boosting algorithm alongside decision trees as its weak learner. Like Random Forests, Gradient Boosted Trees also use an ensemble of multiple trees to create more powerful prediction models for classification and regression. Unlike the Random Forest method which builds and combines a forest of randomly different trees in parallel, in GBTs, a series of trees are built in a sequential order, where each subsequent tree attempts to correct the mistakes made by the previous tree built in the ensemble. Like boosting methods, gradient boosting builds a model in a sequential manner. However, it generalises the method by allowing optimisation on an arbitrary loss function that is differentiable.

In general, we wish to solve some task by learning a model $\mathcal{H}$ which predicts values $\hat{y} = \mathcal{H}(x)$ and minimising some loss function $\frac{1}{N} \sum_i^N L(y_i, \hat{y}_i)$ where y_i is

the corresponding output value from training sample i from a training set of size N. For each iteration k, gradient boosting assumes there is some imperfect model $\mathcal{H}_k$ which can be improved on by constructing a new model that adds an estimator h to improve on $\mathcal{H}_k$: $\mathcal{H}_{k+1} = \mathcal{H}_k(x) + h(x)$. To find the estimator h, gradient boosting begins with the scenario of a perfect h, which would allow $\mathcal{H}_{k+1}$ to predict the exact output value y:

$$\mathcal{H}_{k+1}(x) = \mathcal{H}_k(x) + h(x) = y \implies h(x) = y - \mathcal{H}_k(x) \tag{3.35}$$

Gradient boosting fits h on the residual term $y - \mathcal{H}_k(x)$ in Eq. 3.35. Like in other boosting methods, each subsequent iteration of the strong classifier $\mathcal{H}_{k+1}$ attempts to correct the errors of the previous iteration $\mathcal{H}_k$. In the simplest case of least square regression where we require a model $\mathcal{H}$ to predict the values $\hat{y} = \mathcal{H}(x)$ and want to minimise the mean squared error $\frac{1}{N}\sum_i(\hat{y}_i - y_i)^2$, we can observe that the residuals $y - \mathcal{H}(x)$ for a given model are the negative gradients ($w.r.t\ \mathcal{H}(x)$) of the square error loss function $\frac{1}{2}(y - \mathcal{H}(x))^2$.

3.5 Artificial Neural Networks

3.5.1 Feed-forward Neural Networks

A Feed-forward Neural Network (FNN) [181] is a model composed of several *fully-connected linear layers*. We first review a neural network model for a single hidden layer; this is done for didactic purposes and generalisation to multiple layers is straightforward. For models with a single hidden layer, we denote $\boldsymbol{W}_1$ as a *weight matrix* which represents the first fully-connected (FC) layer with a set of parameters which provides a linear mapping of the input (or *input layer*) $\boldsymbol{x}$ to the output $\boldsymbol{h}_1$. $\boldsymbol{b}_1$ is the FC linear layer's set of biases which translate the input $\boldsymbol{x}$ under the affine transformation $\boldsymbol{h}_1 = \boldsymbol{W}_1 + \boldsymbol{b}_1$. Following this, an element-wise non-linear activation $\sigma(\cdot)$ (such as ReLU [152] or Hyperbolic Tangent) is usually applied to $\boldsymbol{h}_1$. $\sigma(\boldsymbol{h}_1)$ is referred to as a *hidden layer* and each element in the hidden layer is referred to as a *network unit*. The hidden layer is followed by a second linear transformation applied by a second FC linear layer with weight matrix $\boldsymbol{W}_2$ with a set of biases $\boldsymbol{b}_2$ which maps the hidden layer to the model's *output layer*. We can express a standard FNN with a single hidden layer which maps an input $\boldsymbol{x}$ to an output $\hat{\boldsymbol{y}}$ as:

$$\hat{\boldsymbol{y}} = \sigma(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2 \tag{3.36}$$

We can see that Eq. 3.36 can be easily extended to multiple hidden layers. A simplified network architecture of a FNN with multiple hidden layers is shown in Fig 3.5.

One immediate issue with FNNs is that they do not scale well to full image data. For example, consider a small RGB image of size $32 \times 32 \times 3$ (32 pixels high, 32 pixels wide and 3 colour channels). This would indicate that the input layer would require $32 \times 32 \times 3 = 3072$ neurons. For larger images of respectable size (*e.g.* a $256 \times 256 \times 3$ image) would require many more neurons ($256 \times 256 \times 3 = 1732608$)

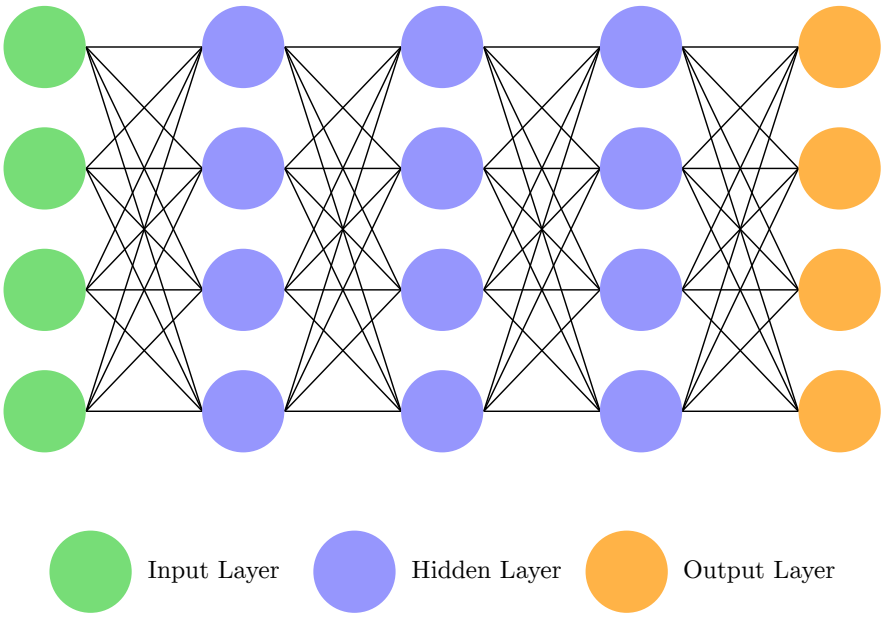


Figure 3.5: A typical feed-forward network architecture. The network consists of stacked fully-connected linear layers (non-linear activations are not shown). Each neuron of each layer is connected to all neurons in the proceeding layer.

and since for a FNN we would like to have several hidden layers in order to represent complex functions, the number of parameters can quickly grow out of control. The full connectivity of the linear layer does not account for the structure of image data and this is largely wasteful in terms of model parameters, leading to over-fitting of the model.

3.5.2 Convolutional Neural Networks

For image processing, the popular tool of choice are Convolutional Neural Networks (CNNs) [117, 135, 201]. Similarly to FNNs, CNN models are made of a recursive application of network layers. The key difference is that instead of being composed of just FC linear layers, CNNs are typically made up of convolution and pooling layers, with FC linear layers found at the end of the network. Combined with a Softmax [9] layer, these FC linear layers serve as classifiers whilst the convolution layers extract the required features used to classify. Fig 3.6 shows a

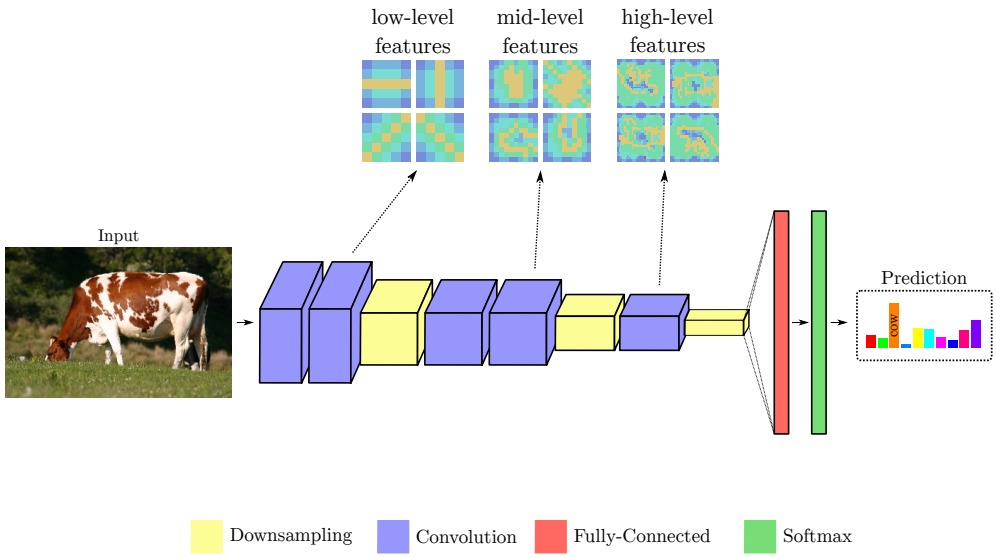


Figure 3.6: A common CNN architecture used for image classification consisting of convolution and downsampling layers, followed by a fully connected linear layer and Softmax [9] layer. As shown, features extracted by filters earlier in the CNN architecture are simpler, extracting points and edges. These earlier layers build up to more complex features in deeper convolution layers in the CNN to detect more complex shapes.

common CNN architecture that would be used for image classification. Features that are learned in the deeper parts of the CNN contain more semantic content about the input image, whilst earlier layers in the CNN extract low-level features such as edges and points [63].

3.5.2.1 Convolution Layer

A convolution layer is the core building block of a CNN. A convolution layer's parameters are composed of a set of learnable filters (also commonly referred to as *kernels*). Each kernel occupies a small spatial region (along width and height), but extend across all channels of the input. As an example, a typical kernel size of the first convolution layer of a CNN may be of size $5 \times 5 \times 3$, representing 5 pixels in height and width, across the 3 colour channels of an 3D input image of size $h_{in} \times w_{in} \times 3$. During the forward pass, each kernel convolves across the entire height and width of the input volume, where the dot product between the

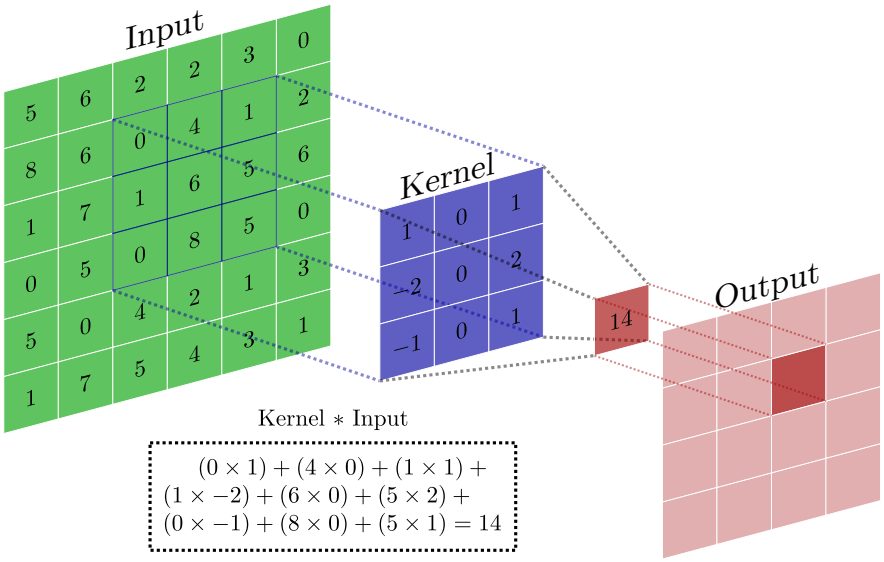


Figure 3.7: A convolution operation on an image patch. The kernel is operated on a sliding window across the entire input image where it is convolved with patches in the input image. The result of each convolution of an image patch with the kernel results in an output pixel value as shown.

weights in the kernel and the values in the input volume at a given (x, y) position is computed. This convolution operation is illustrated in Fig. 3.7.

Each convolution layer is composed of a set of kernels of size $n_{kernel} \times c_{kernel} \times h_{kernel} \times w_{kernel}$, representing the number of kernels, input channels, spatial heights and widths of each kernel respectively. When applied to an input, h_{kernel} and w_{kernel} determine the spatial region of a patch the kernel operates on, c_{kernel} matches the number of channels of the input so that each kernel is applied across all channels in the input and n_{kernel} dictates the number of channels in the output. Each kernel performs a sliding window convolution operation across patches in the input image (shown in Fig 3.8), producing a 2D matrix of values. This convolution operation is a linear transformation which preserves spatial information in the input, making it particularly suited towards image data. Unlike the FC linear layer, neurons in a convolution layer are only connected to a small region of the layer preceding it.

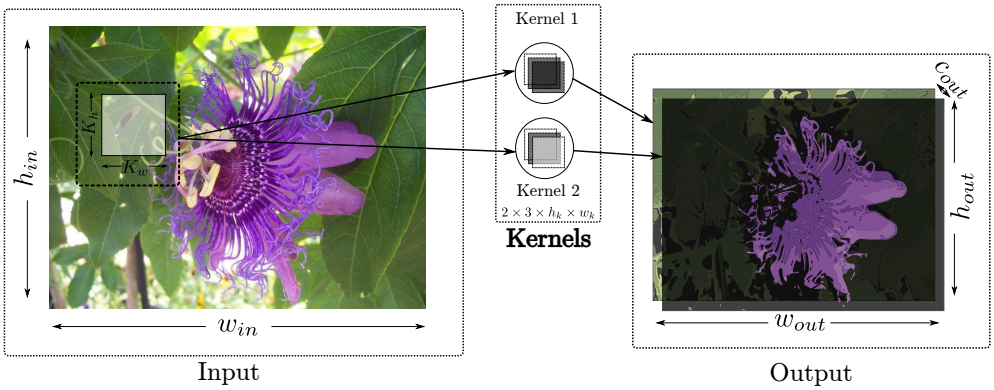
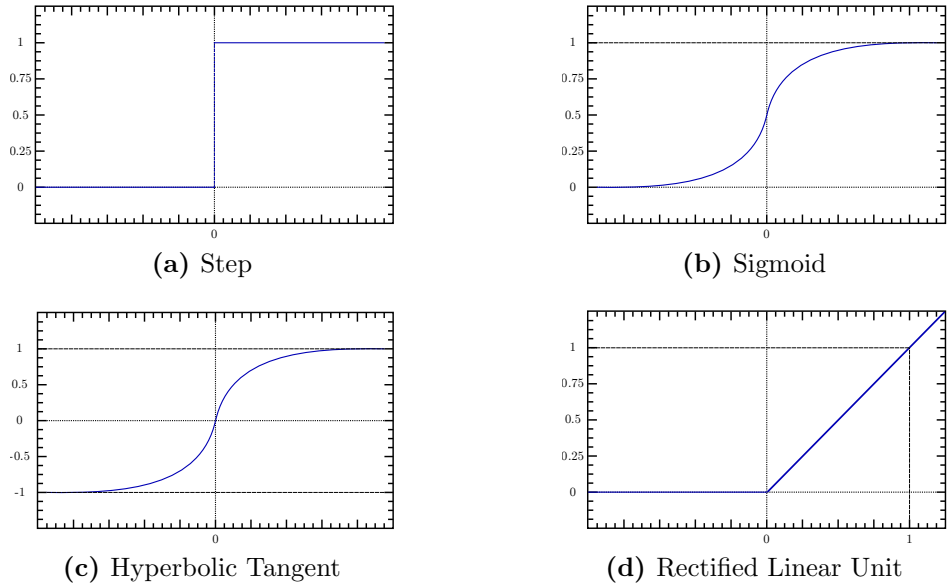


Figure 3.8: A convolution layer in a CNN. In this case, the input is an RGB image with height h_{in}, width h_w, and channels c_{in} (in this case 3 colour channels). Each of the two kernels is convolved with a patch in the input image corresponding to the size of the kernel $h_{in} \times w_{in}$. Each convolution operates across all channels of the input for a given patch area: we can see that Kernel 1 preserves the green channel, whilst Kernel 2 preserves the red and blue channels, outputting blue and purple pixels.

3.5.2.2 Activation Layers

Although convolution layers excel at learning good representations that capture important features of the underlying data, the convolution operation is a linear operation. For many real world problems, the goal of the model is to find a *non-linear* mapping from the input to output. Hence, the recursive stacking of convolution layers in deep models only achieves this required non-linearity if there exists some non-linearity between convolution layers as we would be able to represent multiple stacked convolution layers with a single convolution layer since their convolution operations are linear. Activation layers are a way to break this linearity and allows neural networks to model complex, non-linear functions that are useful for real world data. Generally, an activation layer follows a convolution layer; its main function is to introduce non-linearity into the model. In a neural network, an activation function decides whether a neuron should be "fired" or not. Here, we briefly outline some of the commonly used activation layers [67] and plot them in Fig. 3.9.

Step The Step function (shown in Fig. 3.9a) is a threshold based activation



(a) Step **(b)** Sigmoid

(c) Hyperbolic Tangent **(d)** Rectified Linear Unit

Figure 3.9: Commonly used activation functions employed in deep neural networks. Each activation function serves to break the linearity that exists between successive convolution or fully connected layers in a network, allowing them to model complex non-linear functions.

function. In other words, if the value of the input is above a threshold value, it will activate. Typically, the threshold value is zero [12]. The step activation can be expressed as the following:

$$y = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.37}$$

Sigmoid The Sigmoid function (shown in Fig. 3.9b) is a smooth, step-like function. It is non-linear in nature and hence combinations of this function are also non-linear. Since it is smooth for $x \in \mathbb{R}$, it will have a smooth gradient too. The sigmoid activation can be expressed as the following:

$$y = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.38}$$

Hyperbolic Tangent The Hyperbolic Tangent (TanH) function (shown in Fig. 3.9c) is a scaled sigmoid function where:

$$\tanh x = 2\sigma(2x) - 1 \tag{3.39}$$

As such, it has characteristics similar to the sigmoid function, but with stronger derivatives since its slope is steeper. The TanH activation can be expressed as the following:

$$y = \frac{2}{1 + e^{-2x}} - 1 \tag{3.40}$$

Rectified Linear Unit The Rectified Linear Unit (ReLU) [152] (shown in Fig. 3.9d) is linear (identity) for all $x > 0$ and zero for all $x < 0$. It offers a non-linearity at its "elbow" (*i.e.* the point of discontinuity where $x = 0$). As such, it is non-linear in nature and combinations of ReLU are also non-linear. Unlike the previous activation functions, ReLU is unbounded for $x > 0$ and its range is $[0, \text{inf}]$. The ReLU activation can be expressed as the following:

$$y = \max(x, 0) \tag{3.41}$$

3.5.2.3 Downsampling Layers

Downsampling layers play an important role in a CNN. They are periodically inserted between successive convolution layers where their function is to reduce the spatial size of the representation. This serves two important purposes: first, each downsampling layer increases the effective receptive field of the proceeding convolution layer, allowing its kernels to capture more context of the surrounding area around a given centre point. Fig. 3.11 shows how downsampling can increase the effective receptive field of a kernel. Second, periodic downsampling progressively reduces the spatial size of the representation which serves to decrease the number of parameters. This yields more manageable models from a computational cost

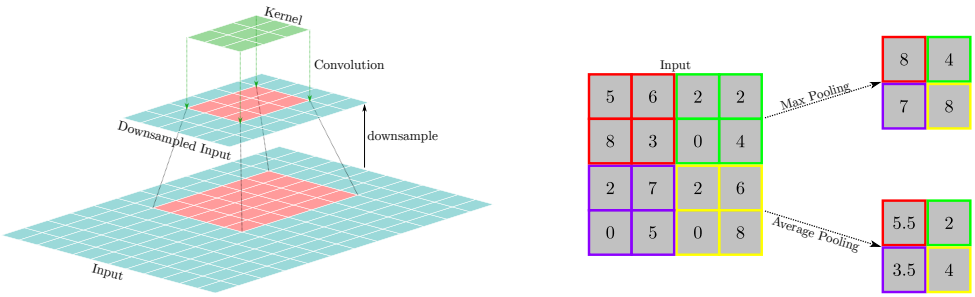


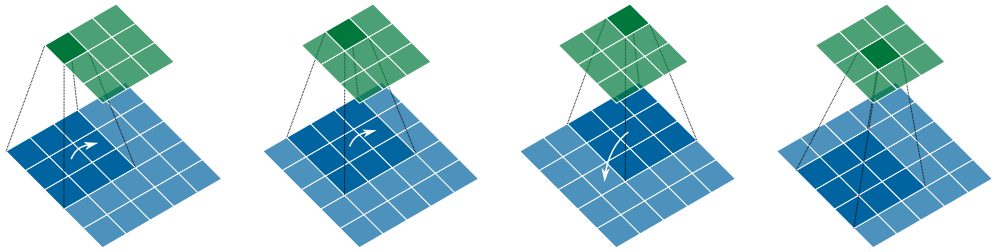
Figure 3.10: (a) For a 3×3 kernel operating on a downsampled input, the effective receptive field the kernel has on the original input is increased to 5×5. In this way, downsampling layers allow smaller kernels to capture more context of the surrounding area around a given centre point. (b) Two common downsampling operations, max pooling and average pooling.

perspective and limits the model's ability to over-fit to the data.

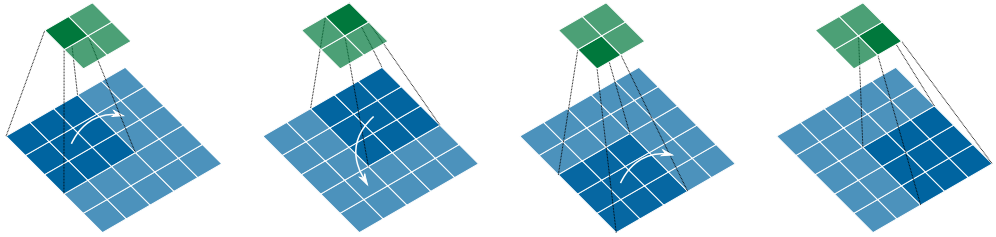
Typically there are two commonly used types of downsampling methods: max pooling and average pooling [67]. In max pooling, the maximum value over a window region is taken as the output, whilst in average pooling, the average over a window region is taken as the output. Commonly, the window is a 2×2 region with a stride of 2 (as shown in Fig. 3.10b). As shown, this has the effect of downsampling the spatial size of the input. Generally for a 3D input, pooling operates on every channel independently and resizes it spatially [67].

3.5.2.4 Strided Convolution Layer

An alternative to using downsampling layers is the use of strided convolutions [67]. Usually, a convolution layer operates densely on an input where each pixel in the input acts as a centre point and a patch around the pixel is convolved with each kernel (shown in Fig. 3.11a). With strided convolutions, the convolution layer instead samples pixels in a sparse manner. For example, a convolution layer with stride size of 2 samples every second pixel along the spatial dimensions of the input (shown in Fig. 3.11b). This sparse sampling has a similar effect to pooling



(a) Convolution



(b) Strided Convolution

Figure 3.11: (a) A 2D convolution layer operating on the input with a stride of 1 and a kernel size of 3×3. In this case, each pixel is densely sampled and the output is not downsampled (reduction in size of the output is due to the size of the kernel). (b) A strided 2D convolution layer with 3×3 kernel size and stride of 2. Due to the increased stride, the spatial size of output is downsampled from the spatial size of the input.

and results in a downsampling of the output.

3.5.2.5 Transposed Convolution Layer

A hypothetical deconvolution operation can be described as follows: given the output of a convolution operation, if we were to feed this output to a deconvolution operation, we should get the original input again. A transposed convolution attempts to mimic this described process by producing an output of the same spatial resolution that a hypothetical deconvolution layer would. However, instead of producing the mathematical inverse of the convolution operation, a transposed convolution layer instead uses padding on the input along with regular convolution operations to revert the spatial transformation of a convolution. Fig. 3.12 illustrates this idea. Transposed convolution layers are sometimes conveniently named deconvolution layers, but it is important to clarify that from a numerical

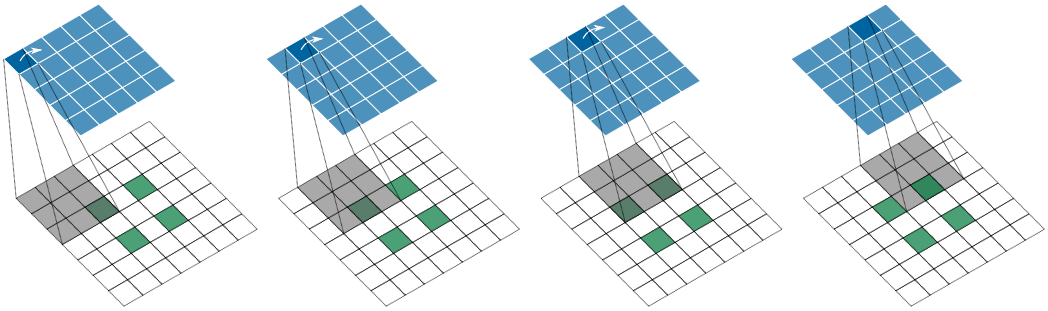


Figure 3.12: A transposed 2D convolution with stride size of 2, kernel size of 3 and no padding. This operation produces a 5×5 output with a 2×2 input.

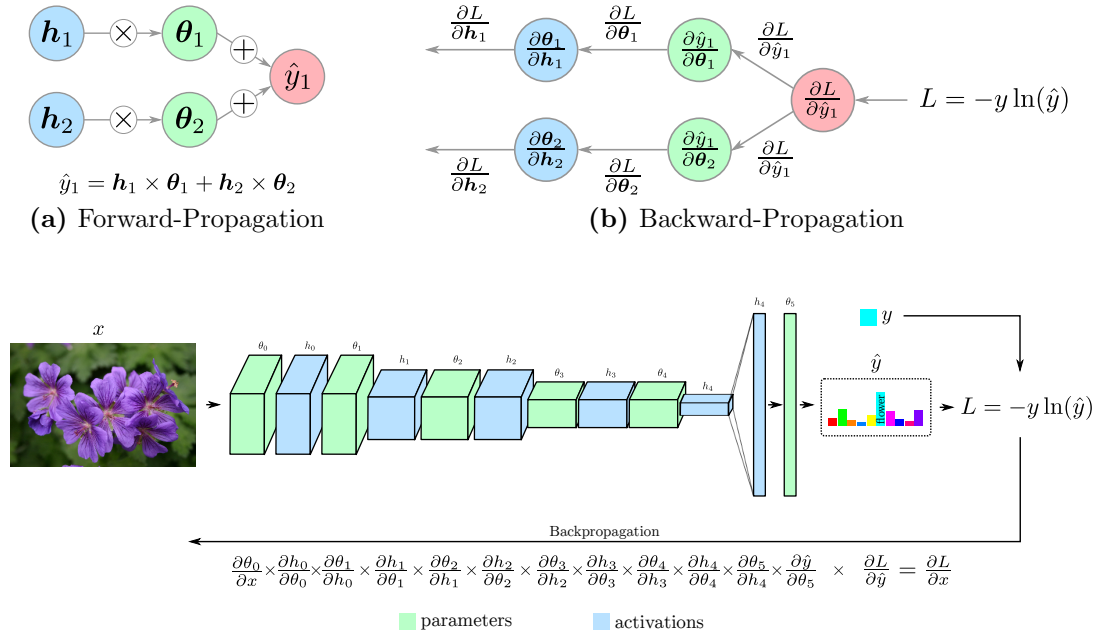
standpoint, the transposed convolution does not reverse the process of a convolution; it simply reconstructs the spatial resolution of the pre-convolution input and performs a convolution, relying on its parameters to learn an inverse mapping of its target convolution operation.

3.5.3 Training Neural Networks

The most common method for training a neural network is to use a (stochastic) gradient descent approach [67], taking small steps in the direction of the negative gradient which minimises the loss function. The loss function defines a cost that the model is trying to minimise. Loss functions vary depending on the training task: for a discrete problem such as classification, the cost function could be the logarithmic loss of the model's prediction of the input image given its ground truth class label. For a continuous problem such as depth estimation, the loss could be the difference between the model's predicted depth and ground truth depth. Here, we briefly discuss some key aspects for training a network.

3.5.3.1 Backpropagation

Optimising the model through gradient descent involves optimising the model's trainable parameters with respect to the loss function, updating the parameters



(a) Forward-Propagation **(b)** Backward-Propagation

(c) Backward-Propagation through Model

Figure 3.13: (a) Forward propagation on the last part of a neural network. Input activations $\boldsymbol{h}_i$ are propagated through and multiplied with the parameters of the network $\boldsymbol{\theta}_i$. (b) Backward propagation propagates derivatives back through the network to the corresponding parameters in the network via use of the chain-rule. (c) Backward propagation through a model for an image classification task. Gradients are computed via the chain rule.

of the model to minimise the loss. This requires computing the gradient of each parameter with respect to the loss function which is intractable for a model with millions of parameters (typical for deep neural networks). Fortunately, we can take advantage of the fact that backward propagation of gradients through the model is possible via application of the chain rule as shown in Fig. 3.13c. Through a combination of backpropagation and the stochastic gradient descent optimisation method which operates on batches of data, we can learn relatively deep models which would otherwise be infeasible to train.

Mini-batch Training Neural networks are commonly trained using batches to reduce training time [67]. Optimisation methods that work at the batch level

(such as stochastic gradient descent) relies on the fact that on *average*, the gradients from a batch approximate "well-enough" the average of the gradients from the entire training set, provided we select a reasonable sized batch [189]. Furthermore, regularisation methods such as momentum also help mitigate any issues that may arise from inconsistencies between the batch and the entire training set. Batches are generated by randomly grouping training samples together and the entire batch is forward propagated through the network in a single pass which generates gradients for backpropagation. The values of the weights in the network are subsequently updated by the average gradient for the batch.

Learning Rate When optimising models using gradient descent methods, choice of learning rate is extremely important to the overall performance of the model. Training a model using gradient descent is an iterative optimisation process which involves gradually stepping towards a solution that the computed gradients predict will most minimise a loss. Under this setting, the learning rate α is a hyperparameter that controls how much we adjust the weights of our model with respect to the loss gradient. The lower this value is, the slower we traverse along the downward slope to the local minima. Low learning rates ensure that local minimas are not missed, but could also result in long training times. Model convergence is intricately tied to the learning rate; if the learning rate is either too high or low, the model may never learn either because it continuously overshoots the minima or approaches it too slowly. Fig. 3.14 shows the effects that different learning rates can have on the final performance of a model. In practice, learning rates are reduced via an annealing or step schedule from a base learning rate and this has shown to help training [67], with the hope that the model finds an optimal local minima.

3.5.3.2 Regularisation During Training

A well-posed problem is defined by [73] if it fulfils the following three criteria:

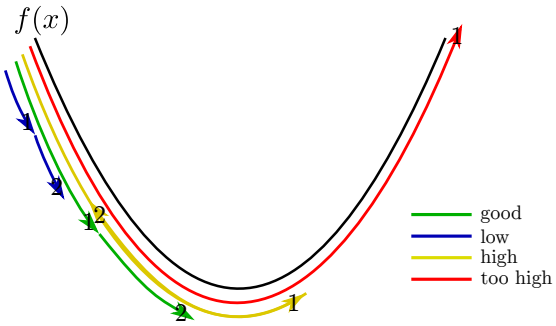
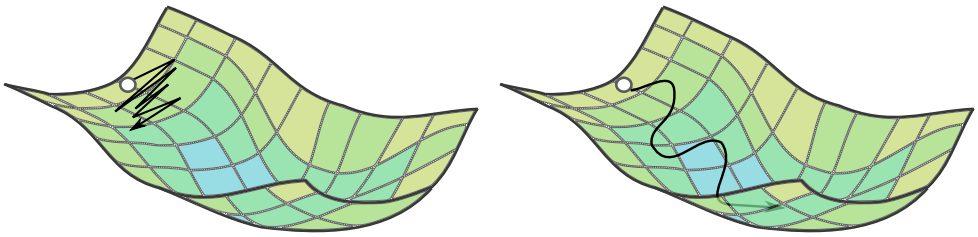


Figure 3.14: When the learning rate is too high (red) or too low (blue), the model either fails to converge to a solution or converges very poorly. Setting a relatively high learning rate (yellow) may lead to short term gains in performance but this can quickly saturate as the model approaches the local minima and begins to overshoot it. A good learning rate (green) should be able to learn quickly but should not miss any local minima it can converge towards.

1. it has a solution.
2. the solution is unique.
3. the solution depends continuously on data and parameters.

If a problem does not fulfil any of these three criteria, it is said to be ill-posed. Generally, many of the real world problems that we wish to solve are ill-posed [101]. As such, for training deep models, typically we apply various forms of regularisation in order to introduce additional information to the model to solve an ill-posed optimisation problem or prevent over-fitting. Regularisation constrains or shrinks the model parameters toward zero and by extension discourages learning a more complex or flexible model [19]. Regularisation can adopt many forms; in the following, we discuss some of the more common methods for regularising training of a model.

Momentum A technique that can enhance stochastic gradient descent for training deep neural networks is *momentum*. Momentum is a relatively easy modification on the stochastic gradient descent algorithm that adds a low computational overhead. The key idea is that parameters should move with a *velocity* which is



(a) (b)

Figure 3.15: (a) Without momentum, learning may be unstable and non-smooth, resulting in a failure to converge. (b) Momentum stabilises learning, smooths out convergence and may help the model converge towards a solution.

influenced by the estimated gradient at each time step:

$$\boldsymbol{v}^{(t)} = \mu(t)\boldsymbol{v}^{(t-1)} + \alpha(t)\nabla_\theta \sum_{i \in \mathbb{B}} \log p_{model}(X_i; \theta) \tag{3.42}$$

$$\theta^{(t)} = \theta^{(t-1)} + \boldsymbol{v}^{(t)} \tag{3.43}$$

Momentum partially observes the curvature of the function; it adds an inertia which both smooths and accelerates the downwards traversal towards the minimum. It dampens oscillations, allowing the algorithm to roll through narrow valleys, small humps and local minima. This has been shown to stabilise learning, allowing this method to perform on par with more complicated second-order methods such as Hessian-free optimisation [205]. Fig 3.15 shows how momentum can aid a model converge towards a solution.

Weight Decay Another common form of regularisation is weight decay [118]. Weight decay is implemented through minimisation of the $L2$ norm of the parameters in the model. Weight decay is a regularisation term which penalises large weight values of parameters, with the goal of reducing large, imbalanced weight value distributions across parameters in the model. This has the effect of stabilising the network. The weight decay term is typically scaled by a small value (usually between $1e$-4 to $5e$-4) [117] before being added to the main loss function so that it is given comparatively low priority by the model when considering the

objective.

3.5.3.3 Exploding and Vanishing Gradients

Exploding and vanishing gradients were significant initial issues with training deep networks [67]. Vanishing gradients are the result of gradients exponentially decreasing as they are backpropagated through a network, where the choice of sigmoid or hyperbolic tangent activation layers reduces these gradients at each stage [65]. For deep enough networks, this results in the gradient reaching such small numerical values that they effectively vanish and thus the network stops learning. In contrast to this, exploding gradients demonstrate the exact opposite issue, where large initial gradients are backpropagated through the network, resulting in instabilities in training or non-convergence of models [78].

3.5.3.4 Initialisation

Along with excessively large or low learning rates, exploding and vanishing gradients have been attributed to incorrect initialisation of weight values of the parameters in the model [144]. Glorot & Bengio first showed that initialisation of weight values for parameters in a model from a standard normal distribution can lead to either poor convergence or in some cases, stops convergence altogether [65]. The main outcome was that initialising all weight values in a model from the same distribution was suboptimal since different layers in the network contained varying numbers of parameters. As a remedy for this, a new type of initialisation heuristic was proposed called Xavier initialisation where the set of weight values for parameters in a layer were initialised to account for the size of the previous layer. These weight values were initialised by randomly sampling from a uniform distribution $[-r, r]$ with the range r defined as:

$$r = \sqrt{\frac{1}{FAN_{in} \times FAN_{out}}} \tag{3.44}$$

where FAN_{in} and FAN_{out} are the respective number of inputs and outputs of the layer being initialised. Following this, He *et al.* subsequently demonstrated that when training deep networks with ReLU activation layers, the weight values of the model should be initialised from a distribution with twice the range since the ReLU activation zeroes out approximately half the activations [78]. A slight modification was made to initialisation heuristic, where weight values would be initialised from a normal distribution with variance σ^2, where:

$$\sigma^2 = \sqrt{\frac{2}{FAN_{in} \times FAN_{out}}} \tag{3.45}$$

3.5.3.5 Batch Normalisation

The method of batch normalisation is a technique for normalising outputs across a mini-batch to stabilise gradients and increase the speed of model training. This method was first introduced by Ioffe and Szegedy in [92]. Generally, a batch normalisation layer is placed between a convolution layer and activation layer where its role is to normalise values across the mini-batch to have zero mean and unit variance. The key idea is to reduce the amount that the hidden unit values in a model shift around (called *covariate shift*), which should improve stability of the network. To achieve this, for the output of each convolution layer in the model, the following affine transformation is applied:

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \tag{3.46}$$

Where x_i is the input, $\hat{x}_i$ is the normalised output and $\mu_{\mathcal{B}}$ and $\sigma^2_{\mathcal{B}}$ are the respective mean and variance of the batch $\mathcal{B}$. However, this transformation has the potential to change the underlying information that the activations represent which may impede learning. Hence, two trainable parameters are used to address this potential issue which enables the layer to learn the inverse operation

of Eq. 3.46 and allows the layer to simply represent an identity transformation:

$$y_i = \gamma \hat{x}_i + \beta \tag{3.47}$$

Where γ and β are the two trainable parameters that scale and offset the input respectively. During the forward pass in training, each batch normalisation layer also keeps track of a moving average global mean and variance, which is used during the inference stage after the model is trained. Since both Eq. 3.46 and Eq. 3.47 are differentiable, backpropagation can be easily applied with batch normalisation layers in a model. Employment of batch normalisation has since become standard practice for training deep models and has heavily mitigated the requirement for careful initialisation to achieve convergence during training.

3.5.3.6 Residual and Highway Connections

Another recent technique which mitigated vanishing gradients involved training a network which utilised "skip" connections to preserve gradient flow. This was achieved by learning either residual functions [79] or a gating mechanism that directed the flow of gradients [204] during training. These approaches were shown to make the model more mathematically stable; the use of skip connections aided gradient flow which was particularly useful for deep networks. Residual and highway networks offer two branches for gradient to flow through: a main branch which maintains the overall state of the learned representations and a residual branch which learns the residual information at a layer level which is added to the representations of the main branch. The main branch is composed of a small number of convolution layers and skip connections (represented by the identity function). This allows gradients to flow through this path relatively unchanged since the gradient is simply scaled by a unit value when flowing through an identity function. At the layer level, the residual branch does most of the heavy lifting when it comes to learning rich representations and layers in the residual branch are "turned on" when they become useful during training; otherwise, gradients are able to flow relatively unimpeded through the main branch.

3.6 Software

The following software libraries were used to produce the work described in this thesis:

- **libCVD (Cambridge Vision Dynamics)** is a portable and high performance C++ based library for computer vision, image and video processing. This is available at https://github.com/edrosten/libcvd.

- **OpenCV (Open Computer Vision)** is an open source computer vision and machine learning software library which supports C++, Python, Java and MATLAB interfaces.
  This is available at https://github.com/Itseez/opencv.

- **Caffe (University of California, Berkeley)** is one of the earlier machine learning frameworks built for Python and C++.
  This is available at https://caffe.berkeleyvision.org.

- **TensorFlow (Google Brain)** is an open source software library for numerical computation using data flow graphs.
  This is available at https://www.tensorflow.org.

- **PyTorch (Facebook AI Research)** is an open-source machine learning library for Python, based on Torch, used for deep learning applications.
  This is available at https://www.pytorch.org.

- **MATLAB (MathWorks)** is a numerical computing environment. This was mainly used for some early prototyping of ideas, pre-processing of datasets and data analysis.
  This is available at https://www.mathworks.com/products/matlab.html.

Chapter 4

Residual Likelihood Forests

In this chapter, we first look to improve upon existing ensemble learning approaches. Here, we introduce an ensemble learning method called Residual Likelihood Forests (RLF) which will serve as a baseline framework and will be built upon in subsequent chapters.

In contrast to other ensemble methods, our weak learners produce conditional likelihoods that are sequentially optimised using global loss in the context of previous learners within a boosting-like framework (rather than probability distributions that are measured from observed data) and are combined multiplicatively (rather than additively). This dramatically increases the efficiency of our strong classifier, allowing for the design of classifiers which are extremely compact in terms of model capacity. In contrast to gradient boosting, our method computes the residual likelihoods in closed form which simplifies model tuning.

We apply our method to several machine learning classification tasks, showing significant improvements in performance. When compared against several ensemble approaches including Random Forests and Gradient Boosted Trees, RLF offers a significant improvement in performance whilst drastically reducing required model size.

4.1 Introduction

The method of Random Forests (RF) was introduced by Breiman [16] and was quickly shown to be a powerful and efficient learning method. Since then, RFs have found use across a wide range of computer vision tasks including applications in image classification [11], semantic segmentation [198], object recognition [58] and data clustering [149]. RFs have shown the appealing properties of dealing well with data that is non-linear and high-dimensional [22], being well-suited for parallel hardware architectures and being inherently well-equipped to handle multi-class problems [190].

Despite their success, much is still unknown about RFs from both a theoretical and practical perspective. Theoretically, RFs are constructed such that each individual tree in the ensemble is learned greedily at the decision node level. This approach can be suboptimal in terms of objective maximisation: there are no guarantees that a global loss is being minimised and learning in this manner does not leverage the complementary information that can potentially exist between different trees in the ensemble [191]. Practically, for RFs to fit complex real data, the non-linearity offered by (very) deep trees is usually required, resulting in ensembles with large overlaps of information between trees and redundancy in the model.

Whilst there have been several works which have improved upon the original RF approach in certain aspects, there still remain key limitations which have yet to be addressed. Notably, the ensemble approaches of Friedman [55] and Schulter *et al.* [191] utilised gradient boosting via fitting of weak learners to pseudo-residuals or to a set of adaptive weights allowing for minimisation of a global loss via gradient descent. Although these methods have demonstrated improved performance over their RF counterpart, they are harder to tune and are more susceptible to overfitting. An alternative approach proposed by Ren *et al.* tried to retain RFs as a baseline classifier, employing a global refinement

technique combined with leaf pruning to gain improvements in performance and reduce model size [169]. However, this method still required the relatively large overhead of constructing the original RF before refinement could be performed.

4.1.1 Contributions

In this chapter, we propose a novel method which constructs a decision forest model that benefits from the boosting approach of creating complementary base tree learners through minimisation of a global loss, yet retains the simplicity in model tuning found in the bootstrap aggregation approach of RFs. Our approach actively seeks to construct weak learners for the ensemble such that the mutual information between learners in the ensemble is accounted for as each new weak learner is added to the ensemble. We show that our method allows for construction of extremely compact and shallow forest models that yield much higher performance when compared to their competing counterparts. The main contributions of this work are as follows:

- A sequential ensemble learning approach which constructs weak learners in the context of previously added learners, explicitly taking account of the mutual information between them, so that their combined outputs offer a more powerful predictor of a class label.

- Unlike gradient boosting methods which also minimise a global loss using gradient descent, our method of optimising residual likelihoods permits the use of large efficient Gauss-Newton like steps.

- Empirically, we show that our method offers vastly improved parameter efficiency when selecting decision functions to construct a tree. In consequence, our method significantly outperforms several competing ensemble learning baselines across several standard machine learning datasets, whilst offering significant reductions in model size.

4.2 Related Work

The work in this chapter builds upon works related to Random Forests and Boosting which have already been outlined in Section 2.1. Additionally, there is residual representation component in the work which we outline in the following section.

4.2.1 Residual Representations

Arguably, the Fisher Vector (FV) was the first representation of a residual vector, describing the direction that the parameters of a model should be modified to best fit some set of data [162]. Vector of Linearly Aggregated Descriptors (VLAD) extended upon this by encoding residual vectors with respect to a visual vocabulary dictionary for image recognition [99]. Both these methods used residual vectors for data fitting to indirectly model the original data and this provided the insight that for vector quantisation, mapping residual vectors is more effective than mapping the original, unreferenced vectors [98]. Finally, He *et al.* transferred residual fitting into deep learning, demonstrating their power by using extremely deep network architectures with "skip" paths and achieving state-of-the-art results in image classification tasks [79].

4.3 Residual Likelihood Forests

In contrast to other ensemble methods, our framework constructs weak learners to output likelihoods (rather than probability distributions). We illustrate why we take this approach with the following didactic example: first, consider two weak learners evaluating some input data to perform a classification task: we denote ${}^1q_j$ and ${}^2q_j$ to represent the probabilities given by each weak learner for the underlying class label j given the input. The correct approach to combining these two weak learners depends on the correlation between them. If ${}^1q_j$ and ${}^2q_j$ are independent of one another, the correct class probability should be given by normalised product for the class j:

$$P(\text{class } j) = \frac{{}^1q_j {}^2q_j}{\sum_k {}^1q_k {}^2q_k} \tag{4.1}$$

On the other hand, when the two weak learners are fully correlated (*i.e.* they are reporting the same information), then ${}^1q_j = {}^2q_j$. In this instance, applying Eq. 4.1 would result in an incorrect estimate of the class probabilities as the normalised squared distribution and we should instead simply average their marginal distributions.

We can see that the correlation between learners can be problematic if we wish to have a consistent approach for correctly combining weak learners within an ensemble since the approach that should be adopted depends on this correlation. Since it is difficult to determine whether weak learners are learning independent information or correlated information, we approach the problem from a different perspective: instead of deciding how to combine distributions from each weak learner, we can instead modify the stored q distributions of each weak learner and simply choose a consistent approach to combining weak learners which gives a correct estimate irrespective of their correlation. In the case of Eq. 4.1, we note that manipulating the stored distributions and replacing ${}^1q_j$ and ${}^2q_j$ with their

square roots gives the correct answer for the fully correlated case:

$$P(\text{class } j) = \frac{\sqrt{{}^1q_j}\sqrt{{}^2q_j}}{\sum_k \sqrt{{}^1q_k}\sqrt{{}^2q_k}} = {}^1q_j = {}^2q_j \tag{4.2}$$

4.3.1 Weak Learners Generating Likelihoods

In practice, ${}^1\boldsymbol{q}$ and ${}^2\boldsymbol{q}$ will rarely be fully independent or fully correlated; rather there will be some overlap in information between them. As such, deciding the amount each weak learner should contribute to the strong classifier's prediction is a non-trivial task. Typically, ensemble methods combine their weak learners by averaging their marginal distributions [13, 47, 188], relying on the law of large numbers to obtain a good estimate of the underlying distribution of the data and smooth out variance between learners.

If we were to manipulate the stored distributions as shown in Eq. 4.2, each weak learner would no longer output a probability distribution. Instead, we can think of each weak learner generating a *likelihood* which is designed to be combined with likelihoods from other weak learners in the ensemble. This forms the basis for our ensemble learning framework: we construct the $\boldsymbol{q}$ values stored in the leaves of each decision tree as likelihoods, conditioned upon the information stored in the trees previously incorporated into the strong classifier. Henceforth, we refer to the $\boldsymbol{q}$ generated by our weak learners as *residual likelihoods*. We can generate these residual likelihoods such that they can be treated as independent sources of information, forming the strong classifier distribution by taking their product and normalising:

$$P(\text{class } j|\boldsymbol{x}, \boldsymbol{\Theta}, \boldsymbol{Q}) = \frac{\prod_{t=1}^{\mathcal{T}} Q^t(\delta^t(\boldsymbol{x}; \Theta^t))_j}{\sum_k \prod_{t=1}^{\mathcal{T}} Q^t(\delta^t(\boldsymbol{x}; \Theta^t))_k} \tag{4.3}$$

4.3.2 Residual Forest Framework

Our method departs from the greedy, entropy minimisation approaches of RFs and instead minimises a global loss function, similar to gradient boosting approaches [54]. The key difference in our approach is that the importance allocated towards hard-to-classify samples during the training process is implicit rather than explicit, as in the case of gradient boosted trees. Our method still chooses decisions from a random subset of the feature space (as in RFs) but these decisions are selected with the objective of minimising a global loss function. Incidentally, this removes the need for weighted training samples or specialised loss functions found in traditional boosting techniques and simplifies the model tuning process.

4.3.2.1 Minimising a Global Loss

For a new tree to be added to the ensemble, we can derive a solution which optimises the residual likelihood it contributes. Let us consider a tree to be added to the ensemble. We wish to find $\{q_j\}$ such that the loss function is minimised when this tree is added to the ensemble. We define the following terms:

- P_{ij}^{-} is the value of class j for sample i obtained by combining likelihoods from all existing weak learners in the ensemble (excludes the new tree to be added).

- q_j is the stored value for class j of the residual likelihood to be contributed by the new tree.

- P_{ij}^{+} is the normalised probability of class j for sample i of the combined likelihoods from the ensemble including the new tree:

$$P_{ij}^{+} = \frac{P_{ij}^{-} q_j}{\sum_k P_{ik}^{-} q_k} \tag{4.4}$$

- c_{ij} is the ground truth class label for sample i taking the class label j, as defined in Eq. 3.21 in the Preliminaries section. For completeness, we redefine it here:

$$c_{ij} = \begin{cases} 1, & \text{if sample } i \text{ has class label } j \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

We can define the loss function L for the ensemble after adding each new tree as:

$$\begin{aligned} L &= -\sum_i \sum_j c_{ij} \log\left(P_{ij}^{+}\right) \\ &= -\sum_i \sum_j c_{ij} \log\left(\frac{P_{ij}^{-} q_j}{\sum_k P_{ik}^{-} q_k}\right) \\ &= -\sum_i \sum_j c_{ij} \left(\log\left(P_{ij}^{-} q_j\right) - \log\left(\sum_k P_{ik}^{-} q_k\right)\right) \end{aligned} \tag{4.6}$$

Hence, we require a q_j to be chosen such that L is minimised (*i.e.* the derivative of L with respect to the q_j is zero):

$$\frac{\partial L}{\partial q_j} = 0, \quad \forall j \tag{4.7}$$

$$\begin{aligned} \implies & -\sum_i c_{ij} \frac{P_{ij}^{-}}{P_{ij}^{-} q_j} + \sum_i \frac{P_{ij}^{-}}{\sum_k P_{ik}^{-} q_k} = 0 \\ \implies & \frac{n_j}{q_j} = \sum_i \frac{P_{ij}^{-}}{\sum_k P_{ik}^{-} q_k} \\ \implies & n_j = \sum_i \frac{P_{ij}^{-} q_j}{\sum_k P_{ik}^{-} q_k} \\ \implies & n_j = \sum_i P_{ij}^{+} \end{aligned} \tag{4.8}$$

where n_j is the total number of samples with class label j within the given leaf. We can show this is a convex optimisation problem where a single global solution

exists. For a convex loss function:

$$\frac{\partial^2 L}{\partial q_j^2} \geq 0 \tag{4.9}$$

Hence, under the assumption that our loss function is convex:

$$\begin{aligned} \frac{\partial^2 L}{\partial q_j^2} &= \sum_i c_{ij} \frac{1}{q_j^2} - \sum_i \frac{(P_{ij}^-)^2}{(\sum_k P_{ik}^- q_k)^2} \geq 0 \\ \implies \sum_i c_{ij} &\geq \sum_i \frac{(P_{ij}^- q_j)^2}{(\sum_k P_{ik}^- q_k)^2} = \sum_i (P_{ij}^+)^2 \end{aligned} \tag{4.10}$$

Thus, our loss function is convex since Eq. 4.10 holds for each sample i where $0 < \sum_k (P_k^+)^2 \leq 1$ and $\sum_k c_k = 1$ across all classes k.

4.3.2.2 Computing Residual Likelihoods

Eq. 4.8 indicates that for a leaf node and samples routed to that leaf node, the solution for the residual likelihood q exists when the sum of strong classifier probabilities across samples is equal to the sum of class labels associated with those samples for a given class. For computing a q that fulfils this criteria, we adopt a straightforward strategy: using Eq. 4.8, we first initialise the residual likelihoods of the new tree $\boldsymbol{q} = \mathbf{1}$ and solve for the updated residual:

$$\text{Initialise: } q_j \leftarrow 1 \tag{4.11}$$

$$\text{Iterate: } q_j \leftarrow \frac{q_j n_j}{\sum_i P_{ij}^+} \tag{4.12}$$

which converges quickly and stabilises when Eq. 4.8 holds, solving for q_j for all j. In theory, the estimate for q_j at the solution can be improved arbitrarily by continuously iterating on Eq. 4.12; in practice, we found that only a single iteration was required for Eq. 4.8 to hold.

This update strategy will design subsequent weak learners to focus on harder-to-

classify samples and as such, it shares similarities with Adaptive Boosting [50]. However, the key difference is that our method *implicitly* raises the importance of misclassified examples through their contribution to the derivative of the loss function defined in Eq. 4.6 (*i.e.* each weak learner constructed tries to maximise information gain of the entire ensemble through its residual contribution), rather than adapting a global set of weights tied to misclassifications of samples. As we will show empirically, this implicit accounting of mutual information between weak learners in the ensemble during the forest construction phase allows the residual forest framework to drastically increase its parameter efficiency and maximise information gain.

4.3.3 Implementing RLF

4.3.3.1 Selecting Decision Node Splits

As previously mentioned, our RLF method differs from RFs [16] since it does not adopt an entropy minimisation strategy to decide decision node splits. Instead, we select decisions which minimise the global loss defined in Eq. 4.6. Similarly to [16], starting at a root decision node for a tree to be added to the ensemble, we randomly select a subset of the feature space and evaluate each feature/threshold pair. Residual likelihoods are generated for each feature/threshold pair (according to Eq. 4.11 and 4.12) and these residuals are added to the ensemble to determine the log loss for each feature/threshold pair. We then select the decision split for the decision node that minimises the overall global loss. This decision node is then fixed and we grow the tree by one level before repeating for the next level, until a specified maximum tree depth is reached. This is summarised as part of our overall learning procedure specified in Algorithm 4.2.

4.3.3.2 Residual Rescaling

Since the residuals of each tree is conditioned towards maximising information gain given the information already learned by the ensemble, this naturally conditions early constructed trees in the ensemble to generate residuals with more information than trees constructed later in the ensemble. Under this setting, the ensemble will be at risk to overfit a small set of early constructed trees in the ensemble. To address this, we adopt a simple residual rescaling method as a form of regularisation: for each tree added to the ensemble, we first convert each likelihood to a *log-likelihood*, obtain the largest absolute log-likelihood value for each tree and use this value to downscale all log-likelihood values in the tree. This guarantees that the maximum residual contribution of any given log-likelihood from a tree in the ensemble will lie in the range of $[-1, 1]$. This corresponds to raising the likelihood values to a fractional power, with a scale factor that is less than 1, ensuring that information will be spread more evenly across the trees and helps smooth out variance in the ensemble's overall prediction.

4.3.4 Summary

We summarise our method in Algorithm 4.2: each weak learner ensemble generates a class label distribution from observed class labels of instances routed to its leaf nodes. This class label distribution serves as a 'truth' distribution for instances routed to the leaf. Each instance routed to a leaf node has a combined distribution from the ensemble and contributes to the stored averaged combined probability distribution in that leaf node. This averaged probability distribution maintains the overall 'state' of the information learned by trees in the ensemble for instances routed to this particular leaf node. Each routed instance also contributes a residual distribution that tries to shift the prediction of the ensemble towards what it views as the true underlying class distribution. This represents the *additional* information that tree has learned *given* what the ensemble has learned. Our framework uses weak learners to model this residual information and relies on the ensemble to maintain the overall state of information learned.

Algorithm 4.2 RLF Training

Require: $\mathcal{N}$: training set
Require: Number of trees in forest $\mathcal{T}$
Require: Feature pool size S
Require: Maximum tree depth D_{max}
0: **for all** $t \in \{1, ..., \mathcal{T}\}$ **do**
0: **for all** $m \in \{1, ..., D_{max}\}$ **do**
0: In parallel:
0: **for all** $s \in \{1, ..., 2^{D_{max}-1}\}$ **do**
0: In parallel:
0: **for all** $p \in \{1, ..., S\}$ **do**
0: Choose a random θ_p
0: Calculate c_j and $\sum_i P_{ij}^-$ for both leaves
0: Calculate q_j for both leaves
0: Rescale q_j within the range of $[-1, 1]$
0: Calculate the loss for both leaves
0: **end for**
0: Choose θ_p for node d that minimises loss
0: **end for**
0: **end for**
0: Copy the winning q_j into the leaves of the tree t
0: **end for**=0

Data set	# Train	# Test	# Features	# Classes
G50c [24]	50	500	50	2
Letter [3]	16000	4000	16	26
USPS [3]	7291	2007	256	10
MNIST [89]	60000	10000	784	10
Chars74k [33]	66707	7400	64	62

Table 4.1: Properties of datasets used in our experiments

4.4 Experiments

To evaluate our proposed RLF classifier, we perform machine learning experiments on several standard classification benchmarks. Throughout our experiments we use 5 standard machine learning datasets to compare RLFs against competing related approaches as well as investigating various parameter settings. The properties of these datasets are summarised in Table 4.1.

4.4.1 Experiment Settings

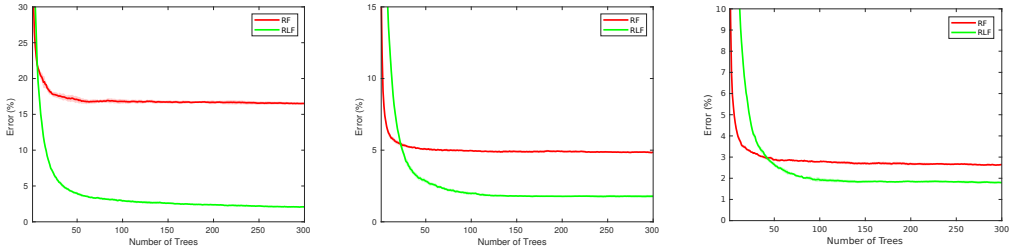
For all our experiments, we follow the settings described in [191]. We set the default number of trees in an ensemble to 100, keeping in line with the experiments in [169, 191]. For each data set, we test a number of random features equal to the square root of the feature dimensions, as recommended in [16], allocating 10 random thresholds per feature. In each case, we report mean error and standard deviation across 10 separate runs to account for variance due to randomness during training, except for the *G50c* data set which we report the mean error and standard deviation across 250 separate runs as was done in [191]. Due to our training method differing from the conventional entropy minimisation schemes of competing approaches, we do not specify an early node termination policy. Instead, we rely on the efficiency of our decision node split choices and construct considerably shallower models. Hence, for practical purposes, we construct trees

up to a maximum specified depth of 15. For the overall results shown in Table 4.2, we follow the tree depth settings in [191], except for the *Chars74k* data set in which we set a tree depth of 15 (as opposed to 25 in [191]).

4.4.2 Comparison with Random Forests

We vary hyperparameters of RLFs and RFs for the *MNIST* data set, observing the classification error and log loss on the test data. The results of these experiments are shown in Figs. 4.1 and 4.2. In Fig. 4.2, we observe that when the maximum tree depth is limited, our method vastly outperforms the competing RF approach. In Fig. 4.2a and 4.2b, we show the classification error and log loss of RFs and RLF when the number of weak learners is fixed to 100 trees and the total number of split nodes is changed by varying the maximum tree depth between the range of $[1, 15]$. We observe that our RLF method can achieve a classification error of 6.6% using just 300 split nodes, compared to the approximately 25,000 split nodes required by its RF counterpart. Fig. 4.2b shows that this property also transitions over to log loss, where the difference between RFs and RLF is even more apparent. To achieve a log loss of 0.22, our method again only requires approximately 300 split nodes, compared to the approximately 400,000 split nodes required by a conventional RF. This is indicative of the efficiency of split node selection that our method offers; the classification errors indicate that our RLF requires vastly less split nodes to acquire similar levels of information when compared to a conventional RF.

Additionally, we perform an ablation study comparing our method to RFs on the *Chars74k* data set, where we vary hyperparameters of number of trees $\mathcal{T}$ and tree depth D_{max} between the ranges of $[1, 300]$ and $[5, 15]$ respectively. We list our results in Table 4.3. We observe similar trends as with the *MNIST* data set: at shallower tree depths (< 5) there is a significant performance gap between RLF and RFs. Additionally, we note that across all maximum tree depth levels, the performance of RFs begins to saturate after around 50 trees have been constructed



(a) **(b)** **(c)**

Figure 4.1: Classification error of RLF vs. RFs varying the number of trees from $[1, 300]$ for a fixed maximum tree depth of (a) 5, (b) 10 and (c) 15 on the *MNIST* test data. For shallower trees, our RLFs significantly outperforms RFs. In each case, we report mean result across 10 separate runs.

for the ensemble. Our RLF continues to improve as trees are constructed past the 50 tree mark; this is especially noticeable at shallower tree depths.

4.4.3 Comparison with Gradient Boosting

Next, we compare our RLF method to two gradient boosted ensemble approaches, Gradient Boosted Trees (GBT) [54] and the more recent method, Alternating Decision Forests (ADF) [191]. Table 4.2 gives an overall comparison between our RLF method, Gradient Boosted Trees and Alternating Decision Forests across all 5 datasets. We observe that RLF offers significant improvements over both gradient boosted baselines.

Furthermore, we perform an ablation study to compare RLF with ADFs on the *Chars74k* data set. We vary the number of trees $\mathcal{T}$ as well as the maximum tree depth D_{max} between the ranges of $[1, 300]$ and $[5, 15]$ respectively. The results are shown in Table 4.3. Similarly to the comparison with RFs, we observe similar trends with ADFs: classification performance begins to saturate when the number of trees in the ensemble approaches 50. At lower maximum tree depths, the difference in performance between RLF and ADFs is apparent.

	RF [16]	GBT [54]	ADF [191]	RLF
G50c	18.91±1.33	18.90±1.31	18.71±1.27	**17.75±1.20**
Letter	4.75±0.10	4.70±0.18	3.52±0.12	**2.59±0.06**
USPS	5.96±0.21	5.93±0.27	5.59±0.16	**4.89±0.09**
MNIST	3.21±0.07	3.15±0.05	2.71±0.10	**1.81±0.03**
Chars74k	17.76±0.13	17.59±0.29	16.67±0.21	**16.33±0.25**

Table 4.2: Overall performance of our method when compared with its main competitors on 5 datasets. The best performing methods are bolded. For [16, 54, 191], we list results given by training under the settings specified by [191]. We train our RLF model using the same settings as described by [191] for each of the 5 datasets shown. In each case, we report mean result across 10 separate runs except for the *G50c* dataset where the mean result across 250 separate runs is shown.

4.4.4 Parameter Efficiency of RLF

Referring to Table 4.3, we can observe that initially our RLF appears to underperform the competition when the number of weak learners is low (≤ 10) but a general trend emerges at all maximum tree depth levels: as the number of weak learners increases, the performance of RLF rapidly overtakes the performance of both [16] and [191]. We can attribute this to the residual rescaling we employ to mitigate overfitting of the model; since we explicitly dampen the contribution of a single tree in the ensemble to make room for other trees in the ensemble to contribute, when the number of learners in the ensemble is low, the ensemble holds back the contribution of weak learners in the ensemble, expecting additional weak learners to join the ensemble. Incidentally, this accounts for the observed phenomenon of early classification performance saturation that both RFs and ADFs suffer from which does not appear to affect our RLF method as much: for each competing method, there is no expectation of additional weak learners joining the ensemble and hence no mechanism which accounts for this. Each weak learner under this scheme has a sole objective of maximising information gain which leads to rapid overlapping of information between learners accumulating as new trees are added to the ensemble. As a result, the classification performance begins to saturate as additional weak learners added to the ensemble are unable to contribute additional information and are simply there to smooth out variance in the final classifier's output.

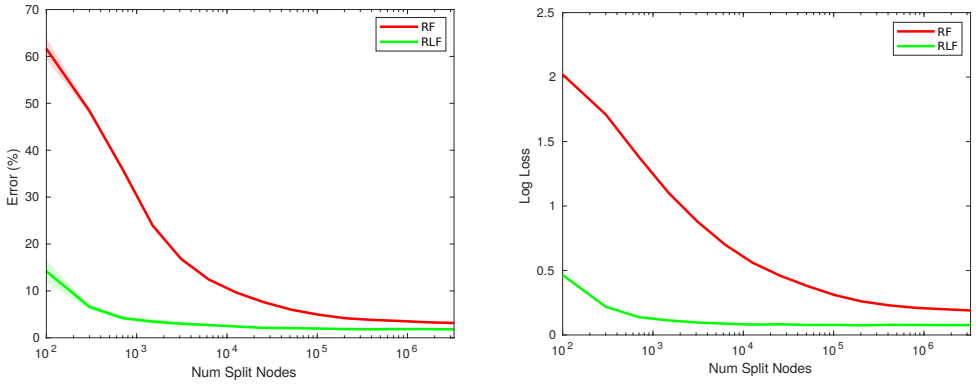
		Number of trees $\mathcal{T}$						
D_{max}	Model	1	10	25	50	100	200	300
5	RF [16]	79.82 ± 0.81	55.78 ± 1.02	51.07 ± 0.67	49.08 ± 0.38	48.25 ± 0.15	47.87 ± 0.18	47.78 ± 0.23
	ADF [191]	83.12 ± 0.68	53.75 ± 0.69	48.15 ± 0.58	46.34 ± 0.54	44.97 ± 0.29	44.20 ± 0.13	44.38 ± 0.14
	RLF	88.09 ± 1.03	46.95 ± 0.38	34.35 ± 0.25	27.52 ± 0.15	23.04 ± 0.24	20.62 ± 0.40	**19.85 ± 0.24**
10	RF [16]	47.62 ± 1.02	32.40 ± 0.33	30.57 ± 0.46	29.82 ± 0.16	29.68 ± 0.23	29.38 ± 0.15	29.31 ± 0.14
	ADF [191]	55.63 ± 1.75	34.32 ± 0.35	32.17 ± 0.37	31.23 ± 0.19	30.60 ± 0.25	30.49 ± 0.08	30.40 ± 0.15
	RLF	84.08 ± 1.53	43.90 ± 0.23	29.59 ± 0.22	20.93 ± 0.24	18.06 ± 0.17	16.92 ± 0.08	**16.65 ± 0.15**
15	RF [16]	34.92 ± 0.51	22.18 ± 0.24	20.05 ± 0.14	19.02 ± 0.19	18.52 ± 0.24	18.31 ± 0.12	18.12 ± 0.11
	ADF [191]	39.76 ± 1.18	24.64 ± 0.43	21.66 ± 0.38	20.38 ± 0.17	19.79 ± 0.20	19.61 ± 0.13	19.41 ± 0.14
	RLF	81.90 ± 0.98	42.88 ± 0.36	28.89 ± 0.29	18.84 ± 0.27	16.33 ± 0.25	16.11 ± 0.20	**15.86 ± 0.22**

Table 4.3: RLF compared with RFs and ADFs on the *Chars74k* data set. Various parameter choices of number of weak learners $\mathcal{T}$ and maximum tree depth D_{max} are shown. We note RLF offers dramatic performance improvements in classification when maximum tree depth is limited, indicating an efficiency in choosing split node decisions. In each case, we report mean result across 10 separate runs.

In contrast, with our residual ensemble approach accounting for mutual information between weak learners in the ensemble, performance saturates at a much later stage (around 100 trees in the ensemble) and more importantly, classification error saturates at a significantly lower value. This is especially evident when the weak learners are shallow in depth, as we can observe that our RLF method more than *halves* the overall classification error when compared to its competitors. These observations are very encouraging for supporting the use of a residual framework. It indicates that under our residual framework where the information learned by the ensemble is considered during the construction phase, we can construct weak learners that are much more efficient in the split decisions they choose. This efficiency in choosing split decisions allows for construction of significantly more compact models which perform as well or better than their competing, deeper ensemble counterparts.

4.4.5 Comparison with Global Refined Forests

A recent work that tries to infuse global information into the RF approach is the method of [169]. This approach uses a global leaf refining scheme along with leaf pruning as a means to allow weak learners to account for mutual information between each other as well as reducing model size. However, this approach still requires the training of the full RFs before iterative refinement and pruning can



(a) **(b)**

Figure 4.2: (a) Classification error and (b) log loss of RFs compared to RLF across a range of depths from $[1, 15]$ on the *MNIST* test data, fixing the number of weak learners in the ensemble to 100 trees. Tree depth is measured as total number of decision nodes in the model. We can observe that when the trees in the ensemble are shallow and the number of decision nodes is limited, there is a dramatic difference in classification performance between models. We also note a similar trend for log loss between RFs and our approach. In each case, we report mean result across 10 separate runs.

occur as global information is not injected until after the initial forest construction stage has completed. We compare the accurate refined model (Refined-A) and economic refined model (Refined-E) from [169] with a shallow and deep variant of RLF (RLF-S and RLF-D respectively). The maximum tree depth for RLF-S and RLF-D models is varied depending on the data set; we list the full conditions for each data set regarding maximum tree depth in Table 4.5. The maximum tree depths were chosen such that competitive or better results were obtained in comparison to the models in [169]. This allows us to investigate the accuracy/memory trade off between shallow and deep models and offer a fair comparison.

We compare both model performance in classification error as well as model compactness using the compression ratio defined in [169] between our approach and the approach in [169]. Similar to [169], the compression ratio is defined relative to model capacity of the original RF [16] and ADF [191] which have similar model sizes.

Data set	Performance Error (%)			
	Refined-A [169]	Refined-E [169]	RLF-D (Ours)	RLF-S (Ours)
USPS	5.10±0.10	5.69±0.15	**5.01±0.04**	5.46±0.16
Letter	2.98±0.15	4.33±0.08	**2.68±0.06**	4.30±0.22
MNIST	2.05±0.02	2.95±0.03	**1.81±0.03**	2.41±0.05
Chars74k	**15.40±0.1**	18.00±0.09	16.33±0.25	18.51±0.17
Data set	Compression Ratio			
	Refined-A [169]	Refined-E [169]	RLF-D (Ours)	RLF-S (Ours)
USPS	2.86	15.14	**5.58**	22.32
Letter	2.33	30.32	**3.49**	446.42
MNIST	6.29	76.92	**6.98**	111.61
Chars74k	1.70	37.04	**5.75**	368.30

Table 4.4: Residual Likelihood Forests compared with [169] for performance/compression tradeoff on 4 datasets. For a fair comparison, the best performing method for each data set comparing across Refined-A and RLF-D is **bolded**, and the best performing method comparing across Refined-E and RLF-S is shaded in blue. We note that when compared to the best performing accurate model of [169], our method (RLF-D) achieves higher performance on 3 out of the 4 datasets and achieves competitive results on the *Chars74k* data set, using a much smaller fraction of the model capacity of [169]. When compared to the economic model of [169], our method is able to achieve better than or competitive performance whilst further reducing model capacity by more than an order of magnitude. In each case, we report mean result across 10 separate runs.

4.4.5.1 Model Performance

In terms of classification error, our deep model (RLF-D) handily outperforms the accurate version of the refined model in [169] on 3 out of the 4 datasets compared against. It is worth noting that even on the *Chars74k* data set, we achieve competitive results, despite limiting our maximum tree depth to 15 (compared to the 25 depth trees used in [169]). In Table 4.4, we show our RLF model's results compared to the model of [169].

	Maximum Tree Depth		
Data set	RF\ADF\ [169]	RLF-D	RLF-S
USPS	10	7	**5**
Letter	15	13	**6**
MNIST	15	12	**7**
Chars74k	25	15	**10**

Table 4.5: Comparison of maximum tree depths of our method compared to competing ensemble methods. Our method is able to train significantly shallower models without any significant to classification performance.

4.4.5.2 Model Compactness

Here, we offer a discussion on the efficiency of our method when utilising the feature space to decide split nodes. The right hand side of Table 4.4 shows the compression ratio of our RLF-D and RLF-S models when compared to the Refined-A and Refined-E models of [169]. We note that our model demonstrates a drastic decrease in model capacity whilst only giving a minor trade off in accuracy. These results further highlight the efficiency of our RLF method in constructing compact models which offer strong classification performance. Unlike the method in [169] which uses leaf refinement and pruning to gain a reduction in model size and accuracy improvement, our method yields the reduction in model size during the construction process of the strong classifier, rather than as a step after the entire forest has been constructed. This is beneficial for our RLF method as we are able to avoid a large part of the process of building deep trees and subsequent leaf pruning our forest for model compactness that the method in [169] requires. Compared with the global refined models Refined-A and Refined-E, our method manages to further improve model compression whilst offering additional improvements to classification performance.

4.4.5.3 Computation Complexity

Finally, we provide a brief discussion on the model computation complexity of RLF. Our RLF model offers improvements when compared to other decision forest models which is evidenced by the large decreases model sizes. Table 4.4 demonstrates that RLF can achieve competitive performances using significantly less model parameters. This is largely in part due to our RLF method requiring much shallower trees when compared to other ensemble methods. Our empirical results in Fig. 4.2 and Tables 4.3, 4.4 and 4.5. Furthermore, our experiments also show that RLFs require less tree learners in the ensemble when compared to Random Forests which also contribute to the reduced computation complexity of our model. In terms of training samples, we trained all our RLF models with the same standard datasets as the competing ensemble methods to ensure a fair comparison between models.

4.5 Discussion and Summary

In this chapter, we have proposed a novel classifier called Residual Likelihood Forests which offers a new approach for combining weak learners within an ensemble learning framework. Our method shows that weak learners in an ensemble can be constructed to optimise a global loss in a complementary manner through the generation of residual likelihoods instead of probability distributions in the base tree weak learner. Empirically, we show that this allows for construction of much shallower and more compact models whilst yielding higher classification performance over competing ensemble methods across several machine learning datasets.

Chapter 5

A Hybrid Deep Learning Model using Forests

In this chapter, we extend on the framework introduced in the previous chapter to utilise the rich space of representative features of a CNN. Additionally, we illustrate how the framework can be incorporated as part of a CNN framework to create a hybrid model. We apply our hybrid model to the more complex task of semantic segmentation and show that our model can outperform several pure deep learning approaches.

We highlight some key architecture choices in our framework to enable it to perform semantic segmentation, using a Fully Convolutional Network (FCN) [135] as a baseline network, replacing its convolutional solver component with our RLF classifier. We demonstrate empirical results which show that the CNN modified with our RLF classifier is able to offer improved performance in segmentation on the PASCAL VOC, NYUv2-40 and MSRC-21 datasets.

5.1 Introduction

Recently, deep learning approaches have become more popular; deep networks have demonstrated that targeted, learned features allow for a powerful distributed representation of data and have used this to surpass the competition in tasks such as image classification [79, 117, 201]. In this chapter, we combine the powerful representational capabilities offered by deep convolutional networks with the concentrated non-linear discriminative capabilities of decision trees. The early work of Sethi described a method for converting a decision tree into a neural network. These tree structured networks were then retrained and called Entropy Nets [194]. In this work, we take an approach parallel to this which aligns with earlier work where decision forests have been used to learn intermediate representations of data [91, 177]. The work of Montillo *et al.* [148] and Kontschieder *et al.* [115] used intermediate predictions of the input data and injected the input space with this information, which can be seen as an earlier version of representational learning using decision forests. Fanello *et al.* [183] learned a cascade of image priors, using a random forest which held a linear prediction model within each leaf nodes.

5.1.1 Contributions

The work presented in this chapter builds upon the decision forest framework introduced in Chapter 4, extending it to utilise the activations from various convolution layers in a CNN to perform semantic segmentation tasks. Due to the use of representation features of CNNs with the Residual Likelihood Forests of Chapter 4, we refer to this approach as Residual Representation Forests (RRF). This chapter details how through specific model architecture choices, the two different frameworks of a decision forest and CNN can be combined into a single hybrid approach. This chapter highlights the following contributions:

- We develop a hybrid framework which combines decision forests with CNNs to perform semantic segmentation.

- We empirically demonstrate that our decision forest classifier can be successfully used to replace the solver component of a CNN and gain significant improvements over the baseline on the PASCAL VOC 2012, NYUv2-40 and MSRC-21 datasets.

5.2 Related Work

5.2.1 Random Forests in Semantic Segmentation

Random forests have found a wide use of applications in vision-related problems, including segmentation. The early work by Shotton *et al.* approached semantic segmentation using dense hand-crafted features as inputs to graph-based methods [199]. Shotton *et al.* subsequently extended upon this work with random forests to generate global priors of images for the framework [198]. Schroff *et al.* used single-histogram class models which were mapped within a random forest classifier for segmentation tasks [190]. Following this, other graph-based techniques incorporated more scene information such as contextual and joint relations between objects [69, 80, 223, 224]. The work of Kontschieder *et al.* used random forests to learn structured class labels which incorporated joint statistics around a small neighbourhood and used this to perform semantic labelling [112]. Kontschieder *et al.* introduced Geodesic forests which enriched the input space with intermediate predictions from a random forest and used this feature space to perform semantic segmentation [115]. Montillo *et al.* used random forests to search for boundaries in a high-dimensional feature space. These boundaries were then utilised to perform segmentation on tumours in medical images [148].

5.2.2 Deep Learning in Semantic Segmentation

A number of recent approaches have found success in semantic segmentation tasks by utilising CNNs. Pinheiro *et al.* employed a feed-forward network which performed pixel-wise classification on pooled, raw pixel data [163] . Similarly, Sermanet *et al.* adopted fully convolutional computation to perform sliding window detection by densely sampling image patches [193]. The work of Long *et al.* introduced the Fully Convolutional Network (FCN), which adapted CNNs to be trained end-to-end for semantic segmentation by coupling the learning of

the weights with a linear solver that combines residual information from multiple layers in the network using a coarse-to-fine approach [135]. Noh *et al.* utilised a deconvolutional network to upsample and combine features from different convolutional layers of the FCN [154].

5.3 System Overview

This section discusses the modifications made to the Residual Likelihood Forests framework presented in Chapter 4 which allow it to be integrated into a CNN framework and perform semantic segmentation. The baseline CNN used for the feature extraction step is a Fully Convolutional Network (FCN) [135]. Fig. 5.1 shows a system overview of RRF, showing the modifications made to the baseline FCN model. In the following section, we discuss the implementation details of our method, including the model architecture, how the model is trained and particular approximations made to the objective function.

5.3.1 Using CNN Features

We enrich the input space of our decision trees with features extracted from the FCN baseline model [135]. Top layer features have large effective receptive fields which extract global information about a scene. Features from lower layers of the CNN have much smaller effective receptive fields and describe local information that tells the classifier "where" the semantic information is located.

We treat a pretrained FCN as a generic feature extractor and train our RRF on features extracted from various convolution layers of the network. Each convolution layer outputs a feature map consisting of C channels. The decision function in each decision node consists of a channel index and a threshold value. The decision function's channel index is a randomly selected channel from a corresponding convolutional layer's output. The threshold value is randomly chosen from a normal distribution generated from the mean and variance across all observed instances routed to its corresponding decision node. Thus, for a given decision node, the parameters θ consist of a channel and a threshold: $\theta = \{c, t_n\}$.

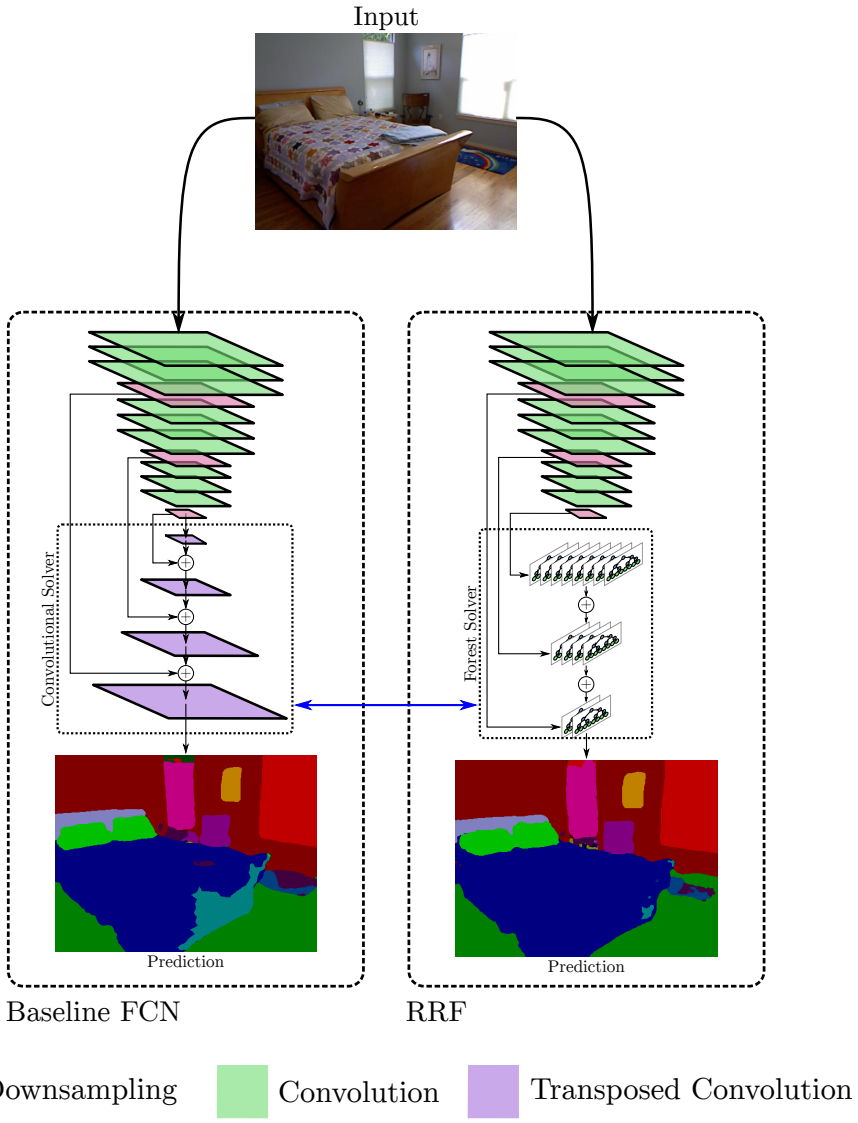


Figure 5.1: Our RRF model replaces the fully convolutional solver component of a baseline FCN. Instead of using skip connections to the activations of earlier convolution layers in the network, we build a decision forest directly from the activations of specified convolution layers in the CNN.

The decision function $d(\boldsymbol{x};\theta)$ is defined as:

$$d(\boldsymbol{x};\theta=\{c,t_n\}) = \begin{cases} 0, & \text{if } \boldsymbol{x}_c \leq t_n \\ 1, & \text{otherwise} \end{cases} \tag{5.1}$$

where $\boldsymbol{x}_c$ is the feature value of a sample in channel c in the convolutional layer's output, and t_n is the threshold value belonging to decision node n of a tree. We extract features *before* they are passed through the Rectified Linear Unit (ReLU) to increase their expressiveness.

5.3.1.1 Choosing Convolution Layers

We train directly on features extracted from FCN convolutional layers, adopting a coarse-to-fine approach when building forests in our RRF. FCN outputs features which capture various levels of context depending on the convolution layer the feature maps are extracted from. Our model uses a group of forests, with a forest dedicated to learning from features for a selected convolution layer in the FCN model. A top down approach is employed; we start by building forests for the coarse, stride-32 feature maps first (`fc7` layer) before transitioning to the stride-16 feature maps (`conv5_3` layer). This is followed by the stride-8 feature maps (`conv4_3` layer) and finally, the stride-4 feature maps (`conv3_3` layer). At this point, we stop training additional forests since this yields diminishing returns beyond the stride-4 feature maps.

5.3.1.2 Coarse-to-Fine Upsampling

For an input feature map of $H \times W$ dimensions, each tree in its corresponding forest outputs an image of $H \times W$ dimensions, where each pixel in the image is mapped to an image of residual likelihoods for each tree in the forest. These images are then multiplied and normalised to generate an image of pixel-wise probability distributions for the entire forest. Following this, we transition to the

next convolution layer which outputs a feature map at a higher resolution (*i.e.* from the `fc7` layer to the `conv5_3` layer), where a new forest is constructed. In order to combine images from a lower resolution (*i.e.* `fc7`) with images from a higher resolution (*i.e.* `conv5_3`), we need to upsample the lower resolution image; for this, we perform bilinear interpolation on the image of per-pixel probability distributions. The upsampled image of distributions is used as the image prior for the new forest in the proceeding convolutional layer.

5.3.2 Learning Residual Representation Trees

Similarly to the Residual Likelihood Trees detailed in Chapter 4, our learning approach departs from the entropy minimisation methods of conventional random forests. Rather than trying to locally minimise entropy of the left and right class label distributions for each decision node in the tree, we instead minimise a global loss function which accounts for all trees.

5.3.2.1 Decision Function Selection

Each tree is added iteratively to the RRF and grows one level at a time, starting with a root decision node. Similar to the Residual Likelihood Forest framework of Chapter 4, to select the best decision function from the pool of available decision functions at each node, we generate potential residual likelihoods for the tree to be added and evaluate the log loss of the RRF, inclusive of the residual likelihoods of the currently constructed tree. A decision function is selected such that it locally minimises the global log loss. Since each decision node in the level is independent of one another, choosing decision functions this way allows us to minimise overall log loss for decision functions selected for the level in the tree. After the decision functions for a level in the tree is selected, we fix these decision functions for each decision node and continue to grow the tree by one level. This process described above is repeated and we continue to add levels of decision nodes to

the incomplete tree until a specified maximum depth is reached, at which point we add the completed decision tree to the RRF and begin building the next tree.

5.3.2.2 Batch Learning

We adopt a batch learning method, where a mini-batch of images is used for the selection of each decision node. In the context of decision forests, this approach can be likened to the bootstrap aggregation method in [16]. For each tree constructed for the ensemble, we randomly select a batch of images from the training data to generate leaf node likelihoods. For each level of decision nodes to be added to the current tree being built, we select a new batch of images, doubling the initial batch size for each level added (to account for the doubling of nodes as we move deeper into the tree). The initial batch size is chosen such that the leaf likelihoods are generated from approximately $N \approx 1000$ training samples.

5.3.2.3 Updating Residuals

As each tree is iteratively added to the ensemble, we apply mini-updates to the overall ensemble to ensure each tree's residual likelihoods are regularly updated and conditioned upon not only the residual likelihoods of trees built before it, but subsequent trees added later in the ensemble. After we add a tree to the ensemble, we select a random batch of images (with batch size equal to the number of samples used to train the deepest level of a tree) and simultaneously generate residual likelihood updates for all existing trees in the ensemble in this manner. Each residual likelihood update term is downscaled by a factor equal to the current number of trees in the ensemble before being applied to each residual likelihood in the ensemble.

5.3.3 Objective Function Approximations

5.3.3.1 Class Label Approximation

The class label distributions are generated from observed samples from the training set which are paired with each CNN feature being routed through the decision forest. The training data consists of pairs of raw RGB images and corresponding dense, per-pixel class label images. Once the RGB image is passed through the CNN, features from the coarse-level layers will have multiple pixel-wise class labels associated with it, which makes it difficult to construct observed class label distributions for these feature maps. To address this, we downsample the ground truth class label image to match the size of the CNN feature map for which we are constructing a forest. The area in the original class label image that corresponds to the coarse-level feature being considered is condensed into a class label distribution. The coarse-level feature is routed through each tree as per usual and is aggregated with the class label distribution stored in the corresponding leaf node.

5.3.3.2 Loss Function Approximation

During training, for the coarse features extracted in the deeper layers of the network, we make an approximation on the training loss computed to select decision functions for each decision node. The loss function defined in Eq. 4.6 from Chapter 4 represents the average global log loss of the RRF at full image resolution. For completeness, we repeat the loss function here:

$$L = -\sum_{i}\sum_{j} c_{ij} \log\left(P_{ij}^{+}\right) \tag{5.2}$$

Where P_{ij}^{+} is the normalised probability of class j for coarse feature i of the combined likelihoods from the ensemble including the new tree. When using coarse feature maps from top level convolutional layers, we modify our loss function as

an approximation:

$$\mathbb{L}_{approx} = -\sum_{i}\sum_{j} s_{ij} \log\left(P_{ij}^{+}\right) \tag{5.3}$$

where s_{ij} is the ratio of pixels with class label j for a coarse feature i with n number of corresponding pixels to the coarse feature i:

$$s_j = \frac{c_j}{n} \tag{5.4}$$

Here, we are approximating the loss function since we take the *average* loss for class j over the area of pixels the coarse-level feature corresponds to from the full resolution image. We find this method is a good approximation for the actual loss function and minimise on this approximate loss function during training.

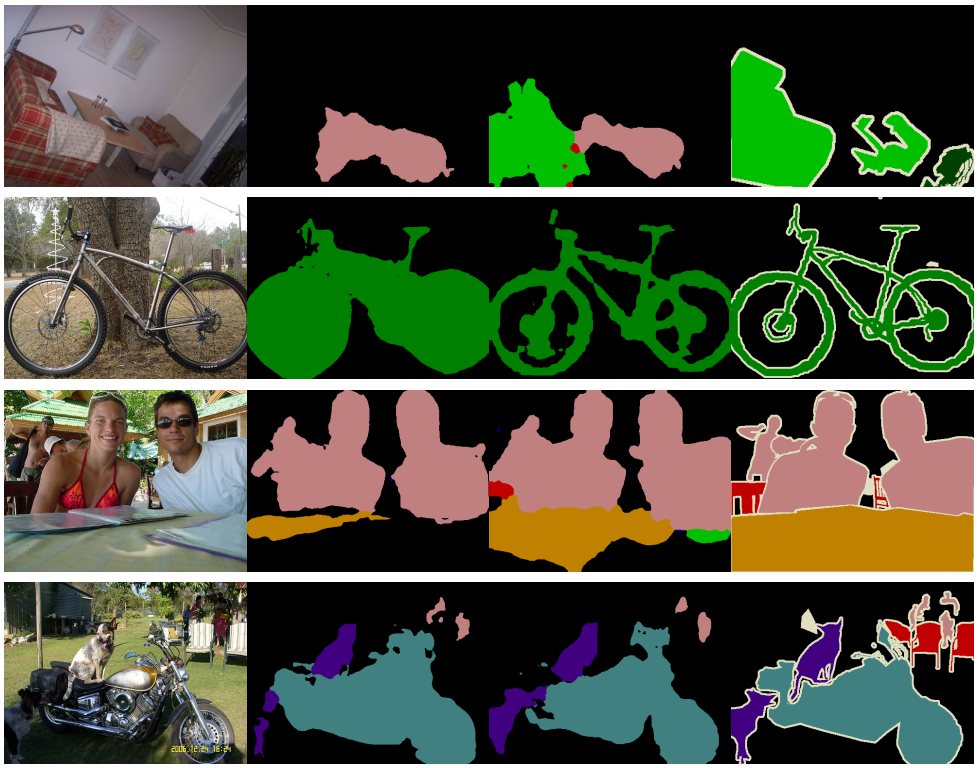
5.4 Experiments

For our experiments, we use the highest performing pretrained models: all models used in our experiments were obtained from the online repository of [135] (*https://github.com/shelhamer/fcn.berkeleyvision.org*). These models are collectively referred to as `fcn-heavy` and trained using the SGD algorithm with a momentum of 0.99 and mini-batch size of 1.

We fix the weights of the pretrained models up to and including the `fc7` layer and replace the solver layers of the network with our classifier (as shown in Fig. 5.1). We use a 4 stride model of RRF (RRF-4s), starting at 25 trees for the finest resolution convolution layer, `conv3_3`, and double the number of trees as we move up to coarse resolution layers. This is done in a similar manner to [135], which roughly doubles the modelling capacity between transpose convolution layers as we transition to coarse resolutions with more semantic information. We adjust the depth of our trees in accordance to the size of the data set, using deeper trees (more modelling capacity) for larger datasets. To assess and compare our model's performance, we use the three metrics defined in [135]: mean IU, mean accuracy and global pixel accuracy.

5.4.1 Pascal VOC

We train our RRF-4s model using the training and validation data of the PASCAL VOC 2012 data set in addition to the augmented PASCAL data set of [75] (12031 total images) and evaluate on the test data (1456 images), specifying a tree depth of 7 levels. Table 5.1 shows the performance of our RRF-4s on the test data of PASCAL VOC 2012 and compares with other methods that work directly with features from a CNN. We outperform the methods of [30,76,135] and achieve very competitive results with [154]. In Fig. 5.2, we show qualitative results comparing our method the baseline FCN model [135].



(a) Image **(b)** FCN-8s **(c)** RRF-4s **(d)** Truth

Figure 5.2: RRF-4s results on PASCAL VOC 2012 Validation data: Column (b) shows the segmentations produced by the model in [135]. The output of our highest performing model, RRF-4s, is shown in column (c). Comparatively, our system seems to be more robust at dealing with irregular camera rotations (first row), handles occlusion better (second row), better discerns between inconsistent surface appearances (third row) and is more robust to irregular lighting conditions (fourth row).

	mean IU (%)
SDS [76]	51.6
CFM [30]	61.8
FCN-8s-heavy [135]	67.2
DeconvNet [154]	**69.6**
RRF-4s (Ours)	69.4

Table 5.1: PASCAL VOC 2012 Results. We report mean result across 10 separate runs.

5.4.2 NYUDv2

We report results on the NYUDv2 [200] data set, using the 40 class semantic segmentation set with pixel-wise labels provided by [71]. We use the standard split of 795 training images and 654 testing images. We lower our tree depth to 5 levels to account for the relatively smaller data set size in comparison to PASCAL VOC data set.

Table 5.2 shows our model trained on only RGB colour information and compared to other colour-only models from [132] and [135]. To offer a fair comparison, we compare our model to the baseline CNN with sliding pyramid pooling of [132] (up to the parts that interact directly with the feature maps generated by the CNN). Here, we demonstrate the advantage of our method with datasets with limited training data; we offer a significant increase in performance in the mean IU, mean accuracy and global accuracy metrics compared to [132] and [135].

	pixel acc. (%)	mean acc. (%)	mean IU (%)
FCN-32s-heavy RGB [135]	60.0	42.2	29.2
Lin *et. al* [132]	63.5	45.3	32.4
RRF-4s (Ours)	**64.1**	**46.3**	**33.7**

Table 5.2: NYUDv2 Results (RGB only). The mean result across 10 separate runs is shown.

We also train our model using both colour and depth information following the procedure from [72] and compare against methods from [41], [72] and [135]. We

open up each decision node in our trees to choose from both colour and depth information resulting in hybrid trees with a mixture of decision nodes that learn based on either colour or depth. We list our comparison in Table 5.3, outperforming all other methods significantly across mean IU, mean accuracy and global accuracy metrics. In Fig. 6.6, we show qualitative results comparing our method the baseline FCN model [135].

	pixel acc. (%)	mean acc. (%)	mean IU (%)
Gupta *et al.* [72]	60.3	35.1	28.6
FCN-32s-heavy RGBD [135]	61.5	42.4	30.5
FCN-32s-heavy RGB-HHA [135]	64.3	44.9	32.8
FCN-16s-heavy RGB-HHA [135]	65.4	46.1	34.0
Eigen *et al.* [41]	65.6	45.1	34.1
RRF-4s (Ours)	**67.3**	**46.9**	**36.2**

Table 5.3: NYUDv2 Results (RGB + Depth). The mean result across 10 separate runs is shown.

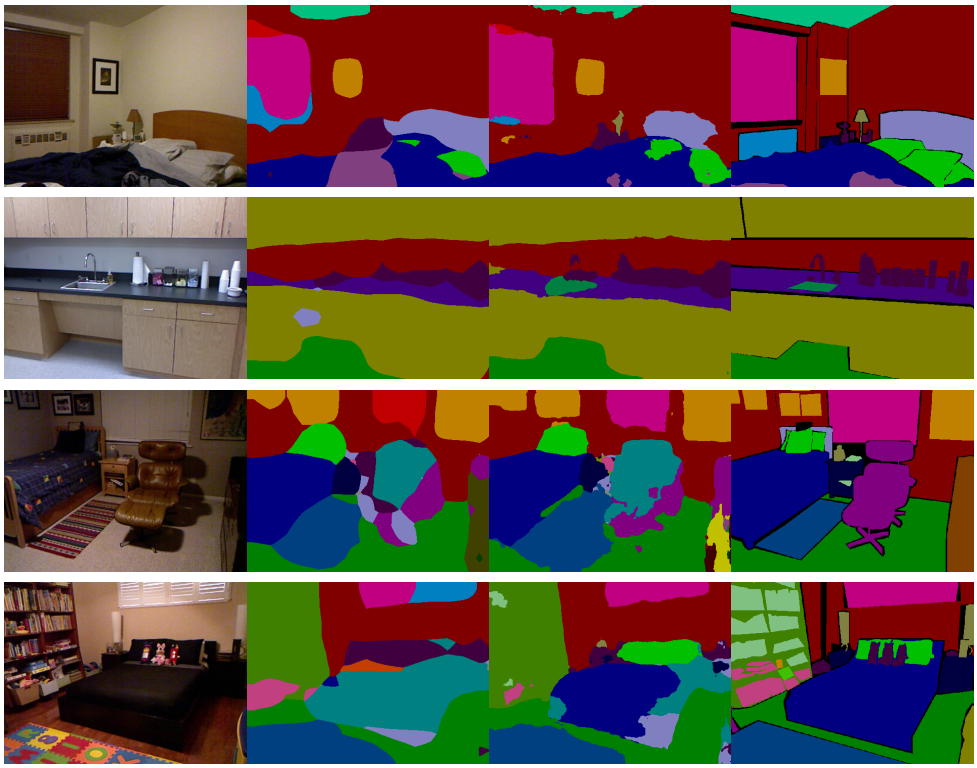
5.4.3 MSRC-21

Finally, we show performance of our method on the MSRC-21 data set [199]. It consists of 591 colour images with 21 different class categories. We train our RRF-4s model on this data set. Once again, to account for the comparatively smaller size of the MSRC-21 data set, we scaled back the depth of our trees using a tree depth of 3 levels.

	pixel acc. (%)	mean acc. (%)	mean IU (%)
Yao *et al.* [224]	86.2	79.3	-
FCN-8s-heavy * [135]	91.2	85.7	76.6
Ours	**94.0**	**91.5**	**85.4**

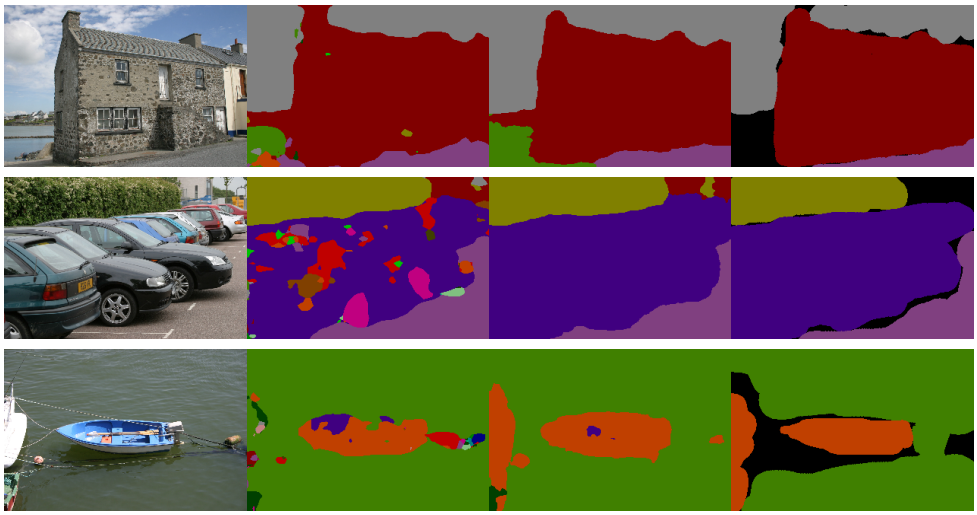
Table 5.4: MSRC-21 Results. The mean result across 10 separate runs is shown.

In Table 5.4, we compare the performance of RRF against the previous methods. Our results offer a large performance gain over the pure deep learning approach



(a) Image **(b)** FCN-32s **(c)** RRF-4s **(d)** Truth

Figure 5.3: RRF-4s results on NYUv2-40 Test data: Column (b) shows the segmentations produced by the baseline CNN model [135]. Column (c) shows the output of our highest performing model, RRF-4s. Our system is able to recover more information when dealing with irregular textures (first row), recover fine structures (second row), handle poor lighting conditions better (third row) and deal with ambiguity between similar class objects (fourth row).



(a) Image **(b)** FCN-8s **(c)** RRF-4s **(d)** Truth

Figure 5.4: RRF-4s results on MSRC-21 Test data: Column (b) shows the segmentations produced by the baseline CNN model [135]. The output of our highest performing model, RRF-4s, is shown in column (c). Comparatively, our system seems to be more robust at detecting changes in texture (first row), handles occlusion better (second row) and better handles irregular lighting conditions (third row).

of the FCN baseline across global accuracy, mean accuracy and mean IU metrics. In Fig. 5.4, we show qualitative results comparing our method the baseline FCN model [135].

*Since no trained network was available, we fine-tuned a pretrained FCN-8s network using the procedure detailed in [135] on the MRSC-21 training set for 100 epochs, with batch size of 1 and high momentum of 0.99, initialising weights from a FCN-8s network model trained on the PASCAL-Context data set [150]

5.5 Discussion and Summary

The system presented in this chapter combines a CNN with a decision forest framework and offers a hybrid model to perform semantic segmentation tasks. This approach shows that the incorporation of decision forests to an existing CNN baseline can offer noticeable improvements to performance and in particular, shows a greater robustness to datasets with limited training data over its pure deep learning counterpart. The current implementation of the model relies on the CNN being a pretrained component which extracts features for the decision forest classifier to learn from. In the following chapter, we show how this framework can be extended upon to additionally learn powerful representation features which have been tuned to the decision forest classifier.

Chapter 6

Fast Residual Forests for Deep Representation Learning

This chapter further develops upon the decision forest framework described the previous two chapters. In Chapter 5, we offered a method for combining decision forests with a pretrained CNN to create a hybrid model where the CNN acts as a feature extractor and the decision forest acts as a classifier using the extracted features of the CNN as an input. However, under this setting, the CNN is fixed after its initial pretraining and cannot utilise the information learned by the decision forest to adjust its parameters and offer more suitable features for the decision forest to learn from. To address this, we extend upon our Residual Forest framework to efficiently learn both deep representations and a classifier.

This work introduces an efficient learning approach which utilises decision forests; it substantially reduces the time required to learn and provides much higher performance when training data is limited. Our model demonstrates noticeable performance improvements over its pure deep learning baseline, notably on datasets with limited training data. We apply our method to the outdoor and indoor segmentation datasets of KITTI and NYUv2-40, outperforming multiple pure deep learning methods whilst using a fraction of training time normally required.

6.1 Introduction

Recently, deep learning approaches have shown success in a range of computer vision tasks, learning both feature representations together with their classifiers, yielding large performance gains over classical methods that rely on conventional feature descriptor and classifier frameworks [79, 117]. However, these models are often difficult to train, both in terms of the training time required as well as the amount of data necessary to ensure generalisation in the trained model [201]. This can be especially problematic in applications where learning speed and robustness to limited training data is required such as in robotic learning. Whilst effective, pure deep learning approaches can often fall short for these types of robotic applications due to their data hungry nature.

In the domain of robotics, learning the task of semantic segmentation plays a vital role in scene understanding since the spatial information in an environment is often just as important as the semantic information within it. Deep learning has been successfully applied to curated semantic segmentation tasks [135, 154] such as the PASCAL VOC data set. For segmentation datasets relevant to robotics, segmentation tasks are not as straightforward; the scenes are often cluttered with many classes and are considered challenging due to the high ratio of classes to training data [72, 200]. Furthermore, due to the nature of segmentation tasks, the amount of training data available is often limited by the difficulty of providing accurate labelled data to learn from [61, 200]. Despite this, research towards investigating approaches that generalise well when there is insufficient training data available has been limited. The popular approach for training deep neural networks for semantic segmentation tasks with limited training data involves fine-tuning from network weights that have been trained on a much larger, more general data set [30, 135, 163]. However, even this does not properly address the situation when the training data is too limited to sufficiently fine-tune the network to perform adequately on a given task.

6.1.1 Contributions

In this chapter, we present a learning approach called Fast Residual Forests (FRF) which not only dramatically reduces time required to learn, but significantly outperforms many pure deep learning methods on datasets with limited training data. Our method learns feature representations from a CNN which are used to train a decision forest framework. We formulate a method which enables the joint training of both features from the CNN *together* with a decision forest classifier, unifying the two different frameworks. The benefits of our approach include:

- Joint optimisation of information across all the leaf node predictions of trees in the ensemble, allowing for large numbers of tree classifiers to be trained in parallel. This drastically reduces the required training time of the model.

- Our model combines highly non-linear random decision forests with a convolutional neural network, using a novel technique to train the entire system via backpropagation.

- Empirically, our approach demonstrates a significant increase in performance on datasets with limited training data, such as the KITTI 6-class and NYUv2 40-class segmentation datasets, when compared against multiple pure deep learning methods.

6.2 Related Work

6.2.1 Deep Learning with Decision Forests

As deep learning rose to prominence, it quickly became the de facto standard for several vision related tasks, including semantic segmentation. Initial works used the strong features learned by neural networks and leveraged their rich information to train conventional, off-the-shelf classifiers [63,72,76]. Following this, end-to-end deep methods which jointly trained CNN features in parallel with a classifier emerged as a natural progression from the typical decoupled frameworks of CNN extractors and classifier pairs [135, 154, 163, 193]. Since then, works in the literature have sought to train conventional CNNs with methods inspired by decision trees [91, 151, 173]. Most notably, our approach shares similarities with methods that look to directly combine convolutional neural networks with decision forests [114, 177]. Bulo *et al.* replaced the decision nodes in decision trees with multi-layer perceptrons and used these modified forests to perform semantic segmentation [177]. Kontschieder *et al.* reformulated split node functions as a soft, differentiable stochastic function, enabling backpropagation to learn both the weights in the network and a decision forest classifier [114].

6.3 Framework

In contrast to [114] and [177], our method does not modify the hard split function conventionally found in decision trees. Instead, it maintains the expected behaviour of a decision forest during the forward pass and employs a technique to approximate the gradient during the backward pass. Additionally, our method reformulates the problem of learning leaf node distributions to learning leaf node likelihoods which subsequently allows for parallelised training of thousands of tree classifiers at once. In the following section, we discuss the details of how our model learns prediction leaf nodes and CNN features.

This approach builds upon the Residual Forest framework detailed in the previous two chapters. We now extend the method so that the representation features produced by a deep convolutional neural network can be jointly optimised alongside our decision forest classifier. Unlike the Residual Forest framework discussed in the previous two chapters, instead of iteratively optimising each decision tree, we now look to jointly optimise all decision trees in parallel across the entire ensemble.

6.3.1 Assigning Channels to Decision Nodes

The decision forest classifier component of the FRF model must first be initialised before the model can be trained. This involves selecting which CNN feature activations will serve as the inputs to the decision forest classifier component of the model. For this, we simply observe the number of channels in the feature map we are constructing the forest for and randomly choose a channel index for our desired activation value for each decision node in the forest (*e.g.* the `fc7` convolution layer outputs a feature map with 4096 channels; for each decision node, we randomly select a discrete channel index value in the range of $[1, 4096]$). This assigns each decision function for a decision node in the forest with a correspond-

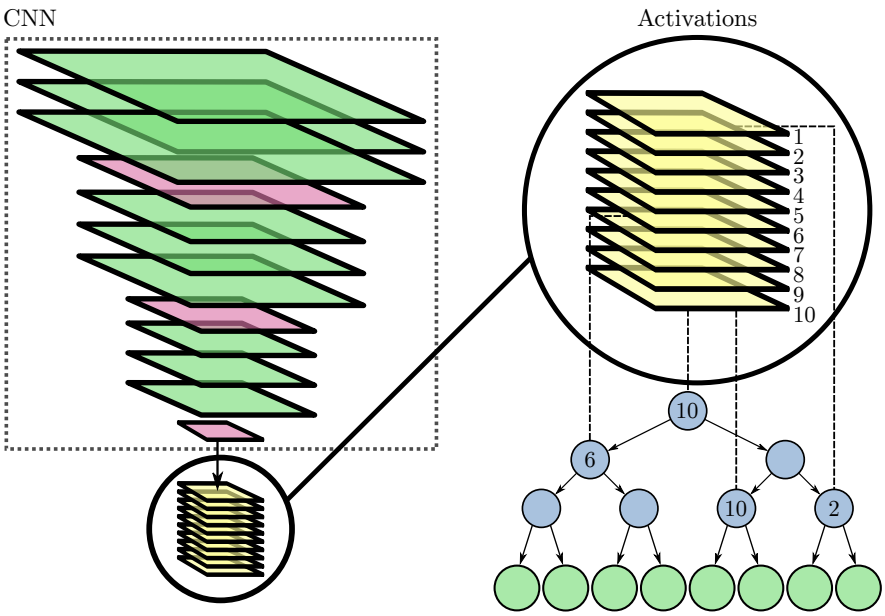


Figure 6.1: Each decision node is initialised by being randomly assigned a channel index from the feature activation map of its corresponding convolution layer. The decision function in each decision node consists of a channel index value which indexes into a channel of the input feature map to obtain an activation value. This is then compared with the decision node's assigned threshold value to determine routing direction.

ing CNN activation value for the convolution layer it is constructed for. This is illustrated in Fig. 6.1. Additionally, each decision node is assigned a randomly sampled threshold term, which is generated by sampling from a normal distribution where the mean and standard deviation is computed from all the values of the activation tensor. This threshold term is used for comparing against its assigned activation value.

Recall from Chapter 5 that for a given decision node, its parameters θ consist of a channel index and a threshold value: $\theta = \{c, t_n\}$. The decision function $d(\boldsymbol{x}; \theta)$ is defined as:

$$d(\boldsymbol{x}; \theta = \{c, t_n\}) = \begin{cases} 0, & \text{if } \boldsymbol{x}_c \leq t_n \\ 1, & \text{otherwise} \end{cases} \tag{6.1}$$

where $\boldsymbol{x}_c$ is the feature value of a sample in channel c in the convolutional layer's output, and t_n is the threshold value belonging to decision node n of a tree. Thus,

forward inference through any tree in the forest proceeds as usual; for a decision node, when the selected activation value exceeds its assigned threshold value, the sample is routed right, otherwise it is routed left.

6.3.2 Learning Prediction Nodes

To optimise the residual likelihoods in the leaf prediction nodes, we can treat the task as a standard optimisation problem across all residual likelihoods, where we would like to minimise the global log loss of the model, given the representation features we have selected from the output of the convolution layers of the CNN.

For any tree d in the ensemble, we can derive the required update term for its leaf prediction nodes. Recall the following terms from Chapter 4:

- P_{ij}^{-} is the value of class j for sample i obtained by combining likelihoods from all existing weak learners in the ensemble (excludes the residual likelihood term added by tree d).

- q_j is the stored value for class j of the residual likelihood to be contributed by the tree d.

- P_{ij}^{+} is the normalised probability of class j for sample i of the combined likelihoods from the all the trees in the ensemble (including tree d):

$$P_{ij}^{+} = \frac{P_{ij}^{-} q_j}{\sum_k P_{ik}^{-} q_k} \tag{6.2}$$

- c_{ij} is the ground truth class label for sample i taking the class label j.

Hence, for each residual $\{q_j\}$ contributing to the final prediction, we look to

minimise the following global loss term:

$$L = -\sum_{i}\sum_{j} c_{ij} \log\left(P_{ij}^{+}\right) \tag{6.3}$$

We can construct the learning of our prediction nodes in the ensemble of trees as a convex optimisation problem; we use Stochastic Gradient Descent (SGD) to jointly optimise all residuals likelihoods in all leaves, across all trees in the ensemble. Substituting Eq. 6.2 into Eq. 6.3:

$$\begin{aligned} L &= -\sum_{i}\sum_{j} c_{ij} \log\left(\frac{P_{ij}^{-} q_j}{\sum_k P_{ik}^{-} q_k}\right) \\ &= -\sum_{i}\sum_{j} c_{ij} \left(\log\left(P_{ij}^{-} q_j\right) - \log\left(\sum_k P_{ik}^{-} q_k\right)\right) \end{aligned} \tag{6.4}$$

Thus, for the class l of a particular instance i, we can generate its first-order derivatives of the log loss with respect to the residual term, q_l, to minimise the global loss term:

$$\begin{aligned} \frac{\partial L}{\partial q_l} &= -c_{il}\frac{P_{il}^{-}}{P_{il}^{-} q_l} + \frac{P_{il}^{-}}{\sum_k P_{ik}^{-} q_k} \\ &= -\frac{c_{il}}{q_l} + \frac{P_{il}^{-}}{\sum_k P_{ik}^{-} q_k} = 0 \\ \implies \frac{\partial L}{\partial q_l} &= -(c_{il} - P_{il}^{+}) \end{aligned} \tag{6.5}$$

Where for the coarser feature maps produced by deeper convolution layers in the model, c_{il} represents a ground truth probability value of instance i for the given class l. We use Stochastic Gradient Descent (SGD) to minimise our loss with respect to the residual likelihoods stored in our ensemble of trees:

$$q_l^{(t+1)} = q_l^{(t)} + \mu\phi \tag{6.6}$$

Where μ is the momentum and ϕ is an update term given by:

$$\phi = -\eta \frac{\partial L}{\partial q_l} - \eta \omega q_l^{(t)} \tag{6.7}$$

Where $\eta > 0$ is the learning rate and $\omega > 0$ is the weight decay. This allows us to build all residual likelihoods in all trees in the decision forest in *parallel*, where each residual contribution from each tree is jointly optimised in the context of contributions from all other trees in the forest.

6.3.3 Learning Features

A standard CNN is composed of convolution layers, downsampling layers and non-linear activation layers with a final classifier layer typically made up of some fully connected linear layers (for classification) or additional convolution layers (for segmentation). This is usually followed by a loss layer such as Softmax [9] which provides a loss to drive the training. The key is that each of these components apply operations which are differentiable and enables training of the entire model via propagation.

Since we are replacing the classifier component of the model with a decision forest, which performs a routing in the forward pass, our model is no longer completely differentiable in the conventional sense and training the model via backpropagation is no longer viable. Here, we offer a modification to the backwards pass of the model, which generates the required gradients for training through approximation of each decision node's routing function and the loss generated by the forest.

6.3.3.1 Approximating the Loss Function

We generate derivatives to drive training by considering an *actual* and *divergent* loss from each node in each tree in the decision forest. For each instance routed

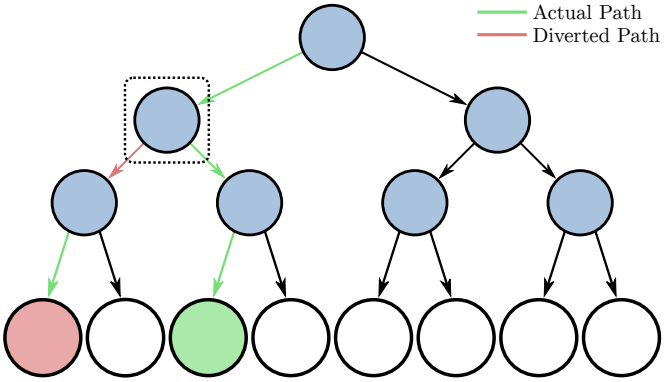


Figure 6.2: For an input activation and a selected decision node (highlighted with dotted box), we reroute the sample from its intended path to a diverted path. This routes the sample to two different prediction leaf nodes, of which one is its actual leaf node and the other which is a diverted leaf node. As a result, two different losses are generated; comparing these two losses allows us to generate a gradient to push the input activation value closer or further away from the selected node's threshold value.

through each tree in the decision forest, we define the actual loss of that instance to be the loss computed if the instance was routed normally through each tree. We define the divergent loss assigned by a node in the tree to be the computed loss if, for a node n in tree t, the instance is instead routed the other direction (but proceeding through all other nodes in the tree as normal). This is illustrated in Fig.6.2.

Using this concept, we can form an approximation of the loss function for our decision forest which is differentiable and use this to generate derivatives to drive training of the weights in the CNN. For each input instance, we approximate the loss function of our forest solver as a blend of sigmoids of the actual and divergent loss, for a given decision node n:

$$L = \frac{1}{1 + e^{\alpha(x - t_n)}} L_1 + \frac{1}{1 + e^{-\alpha(x - t_n)}} L_2 \tag{6.8}$$

Where x is the feature value, t_n is the split threshold which the feature is compared against for a given decision node, d_n. For $x \leq t_n$, L_1 and L_2 are the respective actual and divergent losses; for $x > t_n$, L_1 and L_2 are the respective divergent and actual losses. α indicates the steepness of the sigmoid function which dictates

how close a feature needs to be to its respective threshold to affect the backward derivative generated.

6.3.3.2 Generating Backward Gradients

We can use the loss function defined in Eq. 6.8 to generate backward derivatives and use SGD to train the weights in the CNN. The first-order derivatives of the loss L in Eq. 6.8, with respect to the feature activations x is given by:

$$\begin{aligned}\frac{\partial L}{\partial x} &= \delta\left(-\frac{\alpha e^{\alpha(x-t_n)}}{(1+e^{\alpha(x-t_n)})^2}L_1 + \frac{\alpha e^{-\alpha(x-t_n)}}{(1+e^{-\alpha(x-t_n)})^2}L_2\right) \\ &= \delta\frac{\alpha e^{-\alpha(x-t_n)}}{(1+e^{-\alpha(x-t_n)})^2}(L_1 - L_2)\end{aligned} \tag{6.9}$$

Where δ is defined by:

$$\delta = \begin{cases} -1, & \text{if } x > t_n \\ 1, & \text{otherwise} \end{cases} \tag{6.10}$$

Eq. 6.9 indicates that for a given feature activation x, the backward gradient generated is dependent on the difference between actual and divergent losses computed at the node n the instance was diverted on. For any divergence point, if the divergent loss is lower than the actual loss, this indicates that the feature activation would be better off on the other side of its threshold value. Hence, a gradient is generated which pulls x towards t_n, such that it may cross the threshold. Likewise, if the actual loss is lower than the divergent loss, the gradient generated would push x away from t_n.

For each sample being routed through a tree, we divert its path once for each depth level in the tree (*i.e.* for a tree with 3 levels, each instance routed through the tree will generate a set of 3 divergent losses). This means that each sample generates 3 sets of gradients for each tree in the ensemble. From this, we can compose a tensor of gradients for the feature map of activations in the CNN and transplant this gradient tensor as initial gradients for the standard backpropaga-

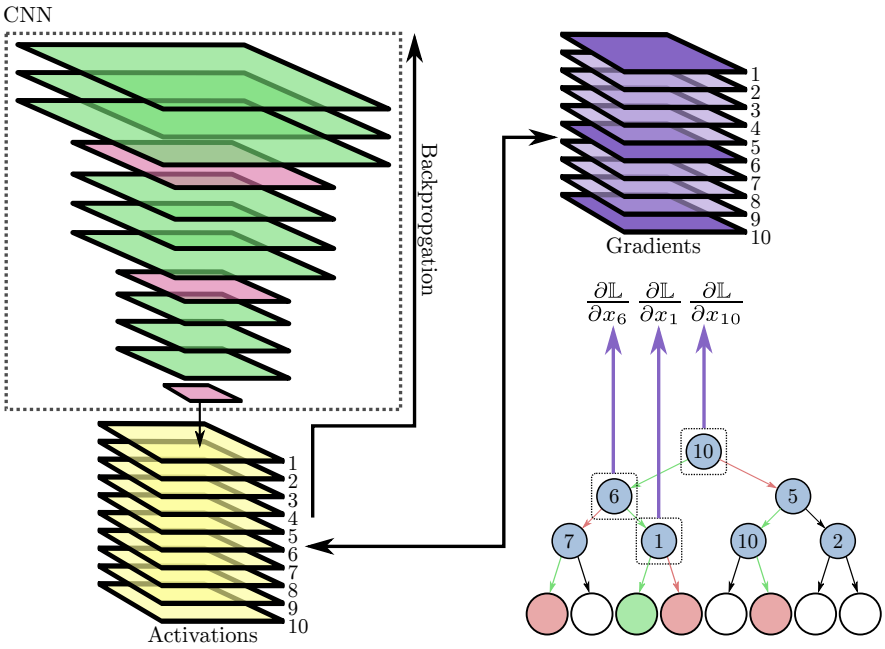


Figure 6.3: Our method generates gradients from our decision forest classifier using an approximation of the loss function using an actual and diverted loss. Each sample is routed through the decision tree as normal until it reaches the decision node for the selected tree depth level. At this decision node, the sample is diverted to the other direction of its normally routed path. After this single diversion, it proceeds as normal. This generates two separate losses which generate a differentiable loss function to compute gradients from. We then transplant these gradients and assign them to the input activation tensor, using backpropagation to update the entire model.

tion algorithm to compute updates for the entire model. Fig. 6.3 illustrates this technique.

6.4 Experiments

We use FCN-8s [135] as a feature extractor to which we attach our forest classifier as a solver; FCN-8s also serves as a baseline comparison in all experiments. All training experiments were performed on a PASCAL GeForce GTX1080 to ensure consistent timing values for training and inference. We use features extracted from the `fc7`, `conv5_3` and `conv4_3` layers of the network in [135] to train our decision forest solver. For all our experiments, we build three forests comprising of 1024, 512 and 256 10-level depth trees for each of the `fc7`, `conv5_3` and `conv4_3` convolutional layers respectively. We train the "at-once" FCN-8s model of [135] with default specified hyperparameters (high momentum of 0.99, mini-batch size of 1, weight decay of $5e{-}4$ and fixed learning rate of $1e$-6). For our model, we use the exact same hyperparameter settings, and we set our value of $\alpha = 1$ for all experiments. We initialised both our model and [135] with the same network weights, trained on PASCAL-Context [150].

We present results in a small-size outdoor scene segmentation data set (KITTI) [61] and a medium-size indoor scene segmentation data set (NVYv2) [200]. These smaller datasets are used to highlight the robustness of our model to datasets with limited training data and demonstrate the improvements we gain in terms of speed of learning and segmentation performance over our baseline model. To measure our performance, we use the metrics defined in [135]: mean IU, mean accuracy and global pixel accuracy.

6.4.1 KITTI

KITTI is a small outdoor scene segmentation data set with 11 classes, comprising of 146 images in total [61] (100 training and 46 testing images). We follow [220] and exempt under-represented classes like poles and pedestrians for evaluation. This turns the data set into a 6 class problem and we follow [220],

using intersection-over-union (IU) as a measurement of performance.

We compare our results in Table 6.1, outperforming all other listed methods significantly in 4 out of 6 classes and offering a significant improvement to the mean IU metric. We train the model of [135] for 20,000 iterations after which it shows performance improvements over the graph-based methods of [171, 208] and [220]. Comparatively, our approach further improves on the result of [135], using the same amount of training data but vastly less training iterations (2000 iterations).

	sky	building	road	sidewalk	vegetation	car	overall
Ren *et al.* [171]	87.4	78.7	72.6	41.3	80.9	59.5	71.9
Tighe *et al.* [208]	81.41	72.7	51.2	17.3	69.9	52.3	60.7
Wang *et al.* [220]	**88.6**	80.1	80.9	43.6	81.6	63.5	74.8
FCN-8s-heavy (20k iterations) [135]	78.9	84.4	87.3	68.3	86.6	**80.4**	81.0
FRF (Ours) (2k iterations)	84.5	**85.9**	**92.3**	**78.8**	**87.8**	80.3	**84.9**

Table 6.1: KITTI test data performance results: intersection-over-union. The mean result across 10 separate runs is shown.

Next, we demonstrate improvements towards speed of learning that our method offers. The graph on the left side of Fig. 6.5 shows the relative performance of our method compared to [135] across a range of training iterations. We can see that given the same base learning rate (1e-6), our model learns more than an order of magnitude faster than [135]. After approximately 20,000 iterations, the model of [135] begins to converge at a performance point of around 81% mean IU. In contrast, our model reaches this performance point much earlier (within 500 iterations of training) and continues to improve in performance for another 1,500 iterations before converging at the significantly higher point of approximately 85% mean IU. This represents a reduction by a factor of 40 in the number of training iterations required by our model compared to [135].

Table 6.4 shows the timings for both training and inference between our model and the model in [135]. We drastically reduce the training time required, using only *4.81* minutes compared to [135], which requires 150.11 minutes to reach the same mean IU performance point of 81%. This represents an increase in training speed by a factor of approximately 31. Inference across both models is approximately the same; we gain a small reduction of 2.67 milliseconds in inference time per inference iteration compared to the model in [135].

6.4.1.1 Ablation Study for Tree Depth

Additionally, we perform an ablation study on performance of our model versus tree depth selection. Table 6.2 shows the mean IU performance from shallow trees (1 depth) up to the relatively deep trees used in our final evaluation (10 depth). This shows some limitations of our model - performance converges to approximately the same mean IU (84.5%), regardless of tree depth up to 4 depth. However, for any depth shallower than 4 depth, mean IU performance begins to suffer, indicating that our trees cannot be too shallow if we want to converge in learning. The results also seem to indicate that deeper trees learn faster (iterations 100 to 500); this is possibly due to the extra non-linearity in the classifier and higher modelling capacity offered by deeper trees.

Tree Depth	No. of Training Iterations				
	100	200	500	1000	2000
1	36.3	58.9	58.3	54.1	53.9
2	62.5	48.7	57.2	63.6	71.6
3	66.5	66.3	65.0	68.6	70.5
4	41.2	62.9	64.4	72.1	84.3
5	39.5	65.3	73.8	81.3	84.6
6	54.8	60.6	67.5	81.9	84.7
7	51.6	74.4	80.4	81.7	84.9
8	57.1	57.6	75.1	**83.9**	**85.0**
9	53.7	74.2	79.7	79.4	84.5
10	**63.8**	**75.3**	**81.5**	82.0	84.9

Table 6.2: Ablation study on tree depth for KITTI

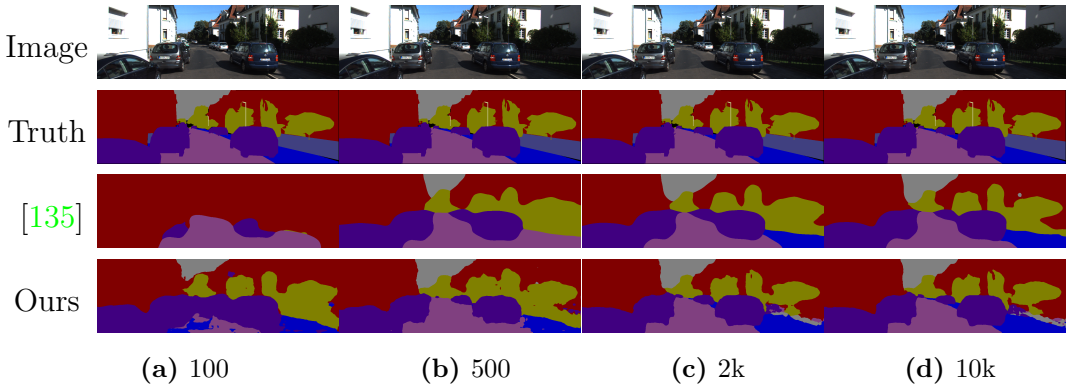


Figure 6.4: Qualitative results on KITTI Test data. The first and second rows show the test image and corresponding ground truth segmentation respectively. The third row shows the output of [135] over training iterations. Our model's output is shown in the fourth row over the same training iterations.

6.4.2 NYUDv2

We now show that our method can be extended to larger, more complex scene segmentation datasets. The NYUDv2 [200] data set is a challenging 40 class indoor scene segmentation problem with pixel-wise labels provided by [71], considered a medium-sized data set (1449 total images). We use the standard split of 795 training images and 654 testing images. First, we list our best results comparison in Table 6.3 - note that our method and [132] use only colour information for training whilst [41, 72, 135] use the method of [72] to utilise additional depth information for training. We show that we outperform all colour only methods by a significant margin across all metrics. Furthermore, we even outperform methods that use both colour and depth information for training across the mean accuracy and mean IU metric, and obtain a competitive result in global accuracy compared to [41].

We again offer a more in-depth analysis of the performance of our model against our baseline model [135]. The graph on the right side of Fig. 6.5 compares the performance of our method to [135] across a range of training iterations. After approximately 100,000 iterations, the model of [135] begins to converge at a per-

	pixel acc. (%)	mean acc. (%)	mean IU (%)
Gupta *et al.* (RGB-HHA) [72]	60.3	35.1	28.6
FCN-32s-heavy (RGBD) [135]	61.5	42.4	30.5
FCN-32s-heavy (RGB-HHA) [135]	64.3	44.9	32.8
FCN-16s-heavy (RGB-HHA) [135]	65.4	46.1	34.0
Eigen *et al.* (RGB-HHA) [41]	65.6	45.1	34.1
FCN-8s (100k iterations) (RGB) [135]	60.6	41.6	29.0
Lin *et al.* (Basemodel) (RGB) [132]	63.5	45.3	32.4
Ours (10k iterations) (RGB)	**64.6**	**48.3**	**34.3**

Table 6.3: NYUDv2 test results: the models below the horizontal line only use RGB information to perform segmentation. The mean result across 10 separate runs is shown.

formance point of around 29% mean IU (consistent with the published results in [135]). Our model reaches this performance point much earlier (after approximately 4000 iterations of training) and continues to improve in performance until 10,000 iterations of training, at a significantly higher point of approximately 34% mean IU. This represents a reduction by a factor of 25 in the number of training iterations required by our model compared to [135].

Table 6.4 shows the timings for both training and inference between our model and the model in [135]. We improve on training time, using only *21.08* minutes compared to [135], which requires 533.21 minutes to reach the same mean IU performance point of 29%. This represents an increase in training speed by a factor of approximately 25. We gain a significant reduction in inference time, cutting inference down by more than 20 milliseconds per iteration of inference over the model in [135].

6.4.3 Training Computation Complexity

The timings in Table 6.4 indicates that our FRF benefits from an improved level of training computation complexity. A large part of the heavily reduced training

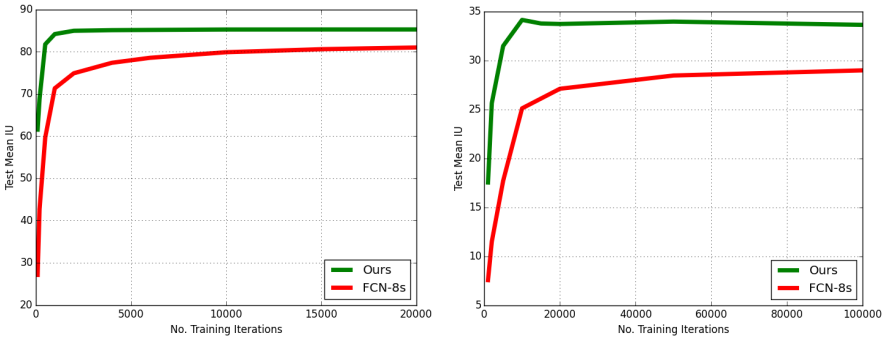


Figure 6.5: Training iteration timings vs. mean IU performance on KITTI (left) and NYUv2 (right) test data. The mean result across 10 separate runs is shown.

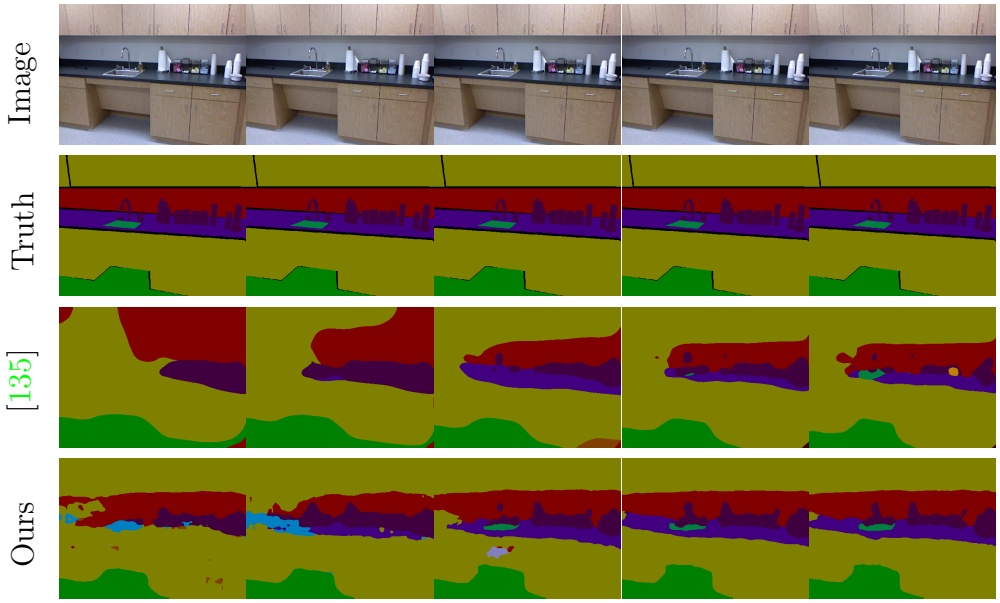


Figure 6.6: Qualitative results on NYUv2-40 Test data. The first and second rows show the test image and corresponding ground truth segmentation respectively. The third row shows the output of [135] over training iterations. Our model's output is shown in the fourth row over the same training iterations.

Method	KITTI		NYUv2	
	Total training time (min)	Avg. inference time (ms)	Total training time (min)	Avg. inference time (ms)
FCN-8s-heavy [135]	150.11	109.02	533.21	69.46
Ours	**4.81**	**106.35**	**21.08**	**48.36**

Table 6.4: Training and inference timings for KITTI and NYUv2. The mean result across 10 separate runs is shown.

times for FRF across both the KITTI and NYUv2 datasets in Table 6.4 is due to FRF requiring far fewer iterations than the competing FCN model. However, training complexity at the iteration level is also further reduced due to the nature of the forest solver component in FRF. When compared to a fully-connected linear solver, in which a point-wise multiplication of the output of the preceding linear layer occurs, the decision forest solver of FRF only requires computation of a small part of the decision forest (*i.e.* along the branches routes of the actual and diverted paths), with the large majority of the forest remaining untouched during a forward pass through the decision forest. This in turn affects the computation complexity of the backwards pass and in turn, the training complexity of the model.

6.5 Discussion and Summary

This work presented an ensemble learning method for segmentation which demonstrates vast improvements in training speed through the use of decision forests. We introduce a hybrid-model which utilises the representational learning of CNNs together with the discriminating power of decision forests. Moreover, we formulate a method that allows for end-to-end learning of both representations and leaf distributions in our decision forest solver. We use this approach to demonstrate successful segmentation results on the KITTI and NYUv2 datasets, outperforming multiple pure deep learning approaches and cutting down training time by more than an order of magnitude.

Chapter 7

Soft Residual Forests in Generative Adversarial Networks

The following chapter is largely drawn from a collaborative work performed with Gil Avraham and will likely appear in a similar form in his thesis. The relative contribution between myself and Gil was equal in essentially all relevant areas. This work looks at the application of decision forests in another prominent area of deep learning within the domain of generative models. Specifically, we look towards an implicit type of generative model called the Generative Adversarial Network (GAN) which are known to be highly unstable in training. For this, we look at improving the conditioning of GANs by using a soft form of the decision forest framework from the previous chapters.

The relevant contribution percentages from my end to produce this work are listed as follows (mutually agreed upon by both first authors):

- Idea conception (50%)
- Network architecture design (50%)
- Loss function design (50%)
- Coding the network (50%)

- Training of models (50%)
- Testing and evaluation (50%)

In this chapter, we demonstrate how the training speed and stability of a deep network is governed by the condition number of the Jacobian matrix of partial derivatives of the square root loss of each training sample *w.r.t.* each model parameter. We illustrate this with reference to a didactic example (multi-dimensional XOR) and a classification example that speeds up training on the CIFAR-10 data set. Following this, we show how including a highly non-linear layer at the output stage of the discriminator/critic network of a GAN in the form of a decision forest can substantially improve its conditioning, increase stability and in return offer greater learning efficiency.

7.1 Introduction

In numerical analysis, conditioning is used to measure the sensitivity of a function towards changes or errors in the input [32]. Well-conditioned learning problems have played an integral role towards the recent significant advances within the field of deep learning. The key contribution of the Rectified Linear Unit (ReLU) [152] over Sigmoid enabled deep networks to be trained by preventing derivatives from vanishing. Leaky ReLUs further improved upon this by dealing with the "dying ReLU" problem (which can arise due to bad initialisation) where large numbers of negative activations zero out too many derivatives [138]. Batch Normalisation [92] explicitly reconditioned the learning problem at the layer level: scaling activations ensured the derivatives on all weights were of similar magnitudes, while shifting the activations to zero mean removed the systematic correlation between derivatives on the weights and those on the biases. Regularisation methods such as Dropout [117] discouraged a network from being sensitive to small differences between highly correlated activations and hence discouraged such high correlations, which in turn improves its conditioning. Residual Networks [79] replaced $X' = f(X)$ with $X' = X + f(X)$. This continually re-centers the learning problem to a point near the solution (f only has to learn a small correction, rather than having to model the entire function). This improves the conditioning on learning f and also provides a means of propagating significant derivatives back up the processing chain, much like Highway Networks [204].

Despite the benefits these techniques have provided for various vision-based tasks that utilise deep learning (such as image classification), this has not fully translated towards the realm of Generative Adversarial Networks (GANs) [68]. Ill-conditioned models are particularly problematic in GANs [68], where the adversarial training setup creates a fragility in the learning process. Recent works in the literature have approached the task of improving GAN stability by making modifications on the loss [2] or adding regularisation to the model [70, 168, 178]. Ultimately, these techniques can be seen as an implicit conditioning on the back-

propagated gradients which leads to improving the stability of training [2, 155]. This indicates that improving the conditioning of learning a GAN may be crucial towards stable learning.

In this work, we analyse the conditioning of a deep neural network by inspecting the condition number of a matrix containing the individually backpropagated derivatives for each sample within a training set. Other works have examined the condition number of the *forward* computation (*i.e.* of the model parameters). This has been shown to be effective in obtaining models which are resilient to adversarial attacks [192] and less sensitive to small perturbations of the input data [165]. Although this is a desired property to have in the inference stage, this does not give as strong insights regarding characteristics of the learning procedure. We adopt an approach similar to [143] and instead observe the conditioning of a model from a backward perspective. The key difference in our method is that rather than observing the behaviour of the average gradient for a set of examples (as was done in [143]), we look at the distribution of per-exemplar gradients *within* the set. Empirically, we show that using a highly nonlinear layer in the form of a decision forest improves the conditioning of the learning problem and as a result, gives increased training stability and efficiency. Whereas ReLU (or other activation functions) are sensitive to the distribution of activations in each channel independently, decision forests are able to disentangle complex joint dependencies between *different* channels, leveraging their ability to model complex real data [23].

7.1.1 Contributions

In this chapter, we introduce Generative Adversarial Forests (GAF). This method incorporates a decision forest as a layer into the discriminator/critic network of a GAN. Our results show that this improves GAN conditioning and subsequently increases stability in training. In summary, we make the following contributions:

- We introduce a methodology which connects the conditioning of a deep neural network with its training performance. Through examples, we show how better conditioned gradients accelerates learning and increases stability.

- This leads us to propose a novel and general approach which incorporates decision forests into deep neural networks. We demonstrate how a decision forest can be represented as a network layer which allows it to be embedded within an end-to-end trainable framework.

- We empirically show that a GAN possessing a well-conditioned discriminator/critic with a decision forest offers both qualitative and quantitative improvements over its baseline counterpart. We show large improvements in FID scores over GAN baselines on the CIFAR-10, CUB Birds and Oxford Flowers datasets.

7.2 Related Work

7.2.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs) were first introduced in [68]; they are a member in the taxonomy tree of Generative Models [66] which attempt to derive an explicit estimate of the density distribution [53, 156]. Generative Adversarial Networks however, do not explicitly form an expression for the density function, but implicitly allow for sampling of the probability distribution once trained. The original GAN formulation [68] introduced the adversarial loss and showed that with enough capacity, the true data distribution can be recovered. However, issues prevalent with the unstable nature of training these networks due to their minimax loss functions gave rise to several works that attempt to address this issue, either by network architectural solutions or modifications to the loss function. Most notably, Radford *et al.* proposed the Deep Convolutional Generative Adversarial Network (DCGAN), an architecture involving deep convolutional networks and regularisation methods such as batch normalisation and empirically showed this to help with stabilising the training of a GAN and delay mode collapse [168]. Other works offered regularisation techniques which built around the base architecture of the DCGAN to try and stabilise training by altering its training scheme or adding components around it [121, 178, 231]. Arjovsky *et al.* introduced the Wasserstein GAN, which proposed a modification on the loss function by considering the Earth-mover's distance (EMD), showing how progress of training can be better coupled with the evolution of generated samples using this new objective [2]. Other works have offered regularisation techniques which were built around the base architecture of DCGAN to try and stabilise training by altering its training scheme or adding components around it [121, 155, 178, 206, 231].

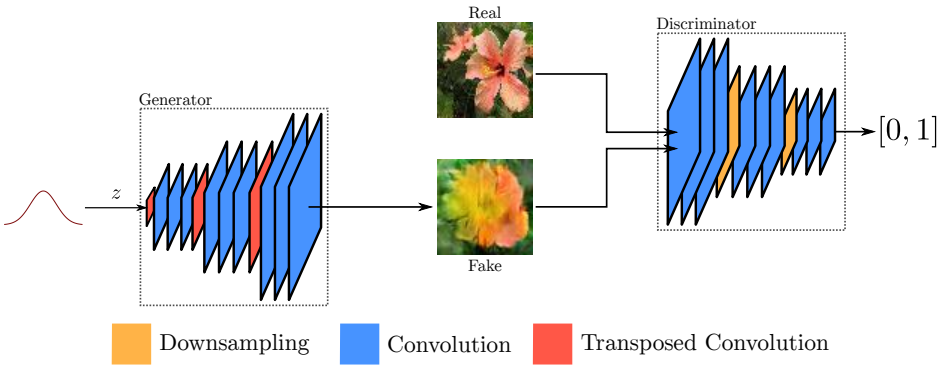


Figure 7.1: A typical generator/discriminator setup in the original formulation of the GAN. The generator's role is to generate fake images from a source of energy (typically Gaussian or uniform distribution) that can fool the discriminator, whose role is to distinguish between real and fake images. Correct training between these two adversaries results in an equilibrium state where they are evenly matched

7.3 Background

7.3.1 Generative Adversarial Networks

Generative Adversarial Networks implicitly estimate a data distribution using the following minimax objective loss function:

$$\begin{aligned} \min_G \max_D V(D,G) = E_{x\sim P_r(x)}[\log D(x)] \\ + E_{z\sim P_z(z)}[\log(1 - D(G(z)))] \end{aligned} \tag{7.1}$$

This loss function narratively describes a game between two opponents, a discriminator and a generator. The discriminator's goal, given an example, is to judge whether that example was drawn from the true distribution $P_r(x)$, or from the generated fake distribution $G(z)$. This is illustrated in Fig 7.1. Hence, the correct optimisation over the loss function in Eq. 7.1 results in an equilibrium state where the discriminator and generator are evenly matched and the discriminator cannot distinguish between real and fake generated samples, assigning a half probability to any sample it receives [68].

7.3.2 Wasserstein Generative Adversarial Networks

A relatively new development in GANs is the Wasserstein GAN [2], which tries to stabilise training of GANs by approaching the problem of matching distributions from a perspective of mass transportation. In this context, the Wasserstein distance is the measure between two distributions. For the set of all joint probability distributions $\Gamma(x_r, x_g)$, we have probability distributions P_r, P_g over $\mathcal{X} \subseteq \mathbb{R}^d$ and the cost function $c(x_r, x_g) : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}^+$. The transport plan $\gamma^* \in \Gamma(x_r, x_g)$ minimises the following:

$$W_c(P_r, P_g) = \inf_{\gamma \in \Gamma(P_r, P_g)} \mathbb{E}_{(x_r, x_g) \sim \gamma}[c(x_r, x_g)] \tag{7.2}$$

Arjovsky *et al.* compared different distance metrics for measuring distribution distances [2] and showed the advantages of using the Wasserstein distance over other probability distances including the Jensen-Shannon (JS) distance which was used in the original GAN formulation. To address the expensive nature of estimating Eq. 7.2 for distributions of high dimensions, Arjovsky *et al.* [2] suggested using the Kantorovich-Rubinstein duality [216]:

$$W_1(P_r, P_g) = \sup_{f \in \mathbb{F}_{Lip}} \mathbb{E}_{x_r \sim P_r}[f(x_r)] - \mathbb{E}_{x_g \sim P_g}[f(x_g)] \tag{7.3}$$

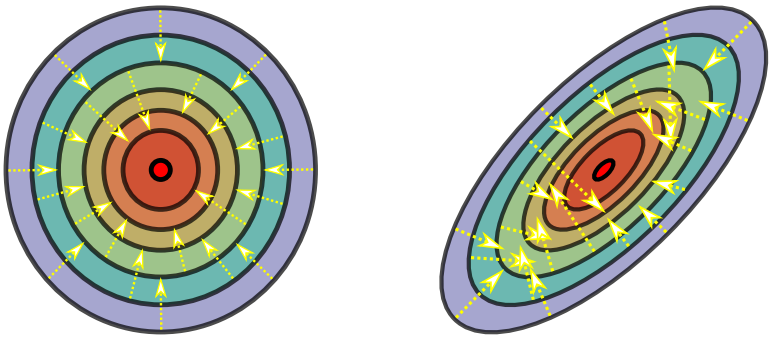
With the special case of $(\mathcal{X}, c)$ as a metric space and $\mathbb{F}_{Lip}$ are bounded 1-Lipschitz functions. This form of computing the Wasserstein distance replaced the conventional way of estimating the JS distance in Eq. 7.1, resulting in a new GAN objective:

$$\begin{aligned} \min_G \max_D V(D, G) = \min_{\theta \in \Theta} \max_{\omega \in \Omega} & \mathbb{E}_{x_r \sim P_r}[D_\omega(x_r)] \\ & - \mathbb{E}_{z \sim P_z}[D_\omega \circ G_\theta(z)] \end{aligned} \tag{7.4}$$

With G, D being neural networks parameterised by θ, ω and P_r, P_z being the data distribution and noise distribution respectively. In the original formulation of the Wasserstein GAN, Arjovsky *et al.* [2] adopted a weight clipping approach to maintain the 1-Lipschitz bound on D; this was recently extended upon by

Gulrajani *et al.* [70] which used gradient penalty instead to enforce the Lipschitz constraint, by directly constraining the gradient norm of the critic's output with respect to its input. The method enforced a soft version of the constraint by modifying the objective function with a penalty term on the gradient norm for random samples $\hat{x} \sim P_{\hat{x}}$, where $P_{\hat{x}}$ is defined as sampling uniformly along straight lines between pairs of points sampled from the data distribution P_r and the generator distribution P_g:

$$\begin{aligned} \min_G \max_D V(D,G) = \min_{\theta \in \Theta} \max_{\omega \in \Omega} \mathbb{E}_{x_r \sim P_r}[D_\omega(x_r)] \\ -\mathbb{E}_{z \sim P_z}[D_\omega \circ G_\theta(z)] + \lambda \mathbb{E}_{\hat{x} \sim P_{\hat{x}}}[(||\nabla_{\hat{x}} D_\omega(\hat{x})||_2 - 1)^2] \end{aligned} \tag{7.5}$$



(a) **(b)**

Figure 7.2: (a): Isotropic loss surfaces of a well-conditioned network. This circular loss surface shows how the minima can be reached from all directions (b): Loss contours of an ill-conditioned model. Due to its non-ideal shape, there are several ways to overshoot and leave the surrounding well, or never enter it.

7.4 A Better Conditioned Discriminator

We first take a step back and look at the underlying problem at hand. More importantly, what does it mean for a model to be ill-conditioned? From the perspective of gradient-based optimisation (as is often the case with learning methods), networks which learn in a stable manner should possess an isotropic loss surface which contain local minimas that are surrounded by spherical-shaped wells. These spherical wells allow the minima to be reached from any direction as well as reducing the chance of overshooting the minima. This is illustrated in Fig. 7.2a. On the other hand, the source of instability in learning can often be traced back to minimas that are surrounded by non-spherical wells [226]; the minimas will be hard to reach and it is easy to overshoot and leave the local area surrounding the minima. Fig. 7.2b illustrates this idea with a ellipsoid-shaped well. These non-spherical wells around a local minima indicate an ill-conditioning of the model.

Here, we propose a method for observing the conditioning of a model. Minimising the loss function *w.r.t* the parameters in the network can be treated as an iterative

least squares problem:

$$\begin{aligned} \mathbb{L}(\boldsymbol{\Theta}) = \sum_{i \in \mathcal{N}} \mathbb{L}(i, \boldsymbol{\Theta}) = \sum_{i \in \mathcal{N}} \left(\sqrt{\mathbb{L}(i, \boldsymbol{\Theta})} \right)^2 \\ \implies \Delta\boldsymbol{\Theta} = -\left[\frac{\partial \sqrt{\mathbb{L}(i, \boldsymbol{\Theta})}}{\partial \boldsymbol{\Theta}} \right]^{\dagger} \sqrt{\mathbb{L}(i, \boldsymbol{\Theta})} \end{aligned} \tag{7.6}$$

where $\mathbb{L}$ represents the loss, and i are samples from the data, $\mathcal{N}$. The first line of Eq. 7.6 converts the global loss into a sum of root losses per example so that the problem is re-expressed as a sum of squares minimisation. Then, defining the matrix M of partial derivatives and vector v of root loss errors:

$$M_{ij} = \frac{\partial \sqrt{\mathbb{L}(i, \boldsymbol{\Theta})}}{\partial \boldsymbol{\Theta}_j} \quad \text{and} \quad v_i = \sqrt{\mathbb{L}(i, \boldsymbol{\Theta})} \tag{7.7}$$

would enable a Gauss-Newton update: $\Delta\boldsymbol{\Theta} = -M^{\dagger}\boldsymbol{v}$ (*i.e.* the second line of Eq. 7.6). The condition number of this matrix M corresponds to the eccentricity of the iso-loss ellipsoids in Fig. 7.2 and hence the efficiency of gradient descent.

This offers a general method for obtaining the condition number of the Jacobian of the backpropagated gradients on a per-exemplar basis for any given model. In the following sections, we provide examples of increasing difficulty, tying observations from this approach to successful learning of a given task.

7.4.1 Example: XOR

We start with a didactic example to illustrate how poorly conditioned networks can fail to learn an apparently simple 3-dimensional XOR function. For this task, we construct two models: the first model consists of two FC linear layers with a ReLU non-linearity in between, the second model consists of a FC linear layer connected to a decision tree of depth 2. Across both models, the first FC linear layer consists of 3 hidden nodes which ensures both models have the same modelling capacity.

Theoretically, we would expect both models to learn the XOR function. Looking at the model with two FC linear layers, one possible solution for the 3-dimensional XOR problem using 3 hidden nodes is the following: for the first FC layer, its parameters consist of a 3×3 matrix of ones, W_1 and biases $b_1 = 2$, $b_2 = 0$ and $b_3 = -2$. The output of this FC layer before a ReLU activation is applied is shown in the second column of Table 7.1. The ReLU activation zeroes out any negative output values and is shown in the third column of Table 7.1. For the second FC layer, its 4 parameters consist of a 3×1 vector $W_2 = [1, -3, 5]$ and a bias of $b_4 = -0.5$. This gives the output shown in the final column of Table 7.1 which solves the 3 dimensional XOR problem.

Input			FC1			ReLU			FC2
x_1	x_2	x_3	h_1	h_2	h_3	h_1	h_2	h_3	y
1	1	1	5	3	1	5	3	1	0.5
1	1	-1	3	1	-1	3	1	0	-0.5
1	-1	1	3	1	-1	3	1	0	-0.5
1	-1	-1	1	-1	-3	1	0	0	0.5
-1	1	1	3	1	-1	3	1	0	-0.5
-1	1	-1	1	-1	-3	1	0	0	0.5
-1	-1	1	1	-1	-3	1	0	0	0.5
-1	-1	-1	-1	-3	-5	0	0	0	-0.5

Table 7.1: A possible solution to the 3 dimensional XOR problem using 2 FC linear layers using 3 hidden nodes.

Similarly, for the FC linear layer and 2 depth decision tree, one possible solution for the 3-dimensional XOR problem using 3 hidden nodes is the following: for the first FC layer, its parameters consist of a 3×3 matrix of ones, W_1 and biases $b_1 = -1$, $b_2 = 1$ and $b_3 = -1$. The output of this FC layer is shown in the second column of Table 7.2. This is followed by a 2 depth soft decision tree with 4 leaf parameters $\ell_1 = -1$, $\ell_2 = 10$, $\ell_3 = -40$ and $\ell_4 = 20$. This gives the output shown in the final column of Table 7.2 which solves the 3 dimensional XOR problem.

However, in practice we do not observe the model of two FC linear layers with ReLU learning this function; in contrast, the FC linear layer with decision tree is able to learn the function. Fig. 7.3a shows the log loss across 1000 epochs of

Input			FC1			FC2
x_1	x_2	x_3	h_1	h_2	h_3	y
1	1	1	2	4	2	12.48
1	1	-1	0	2	0	-0.66
1	-1	1	0	2	0	-0.66
1	-1	-1	-2	0	-2	0.05
-1	1	1	0	2	0	-0.66
-1	1	-1	-2	0	-2	0.05
-1	-1	1	-2	0	-2	0.05
-1	-1	-1	-4	-2	-4	-0.39

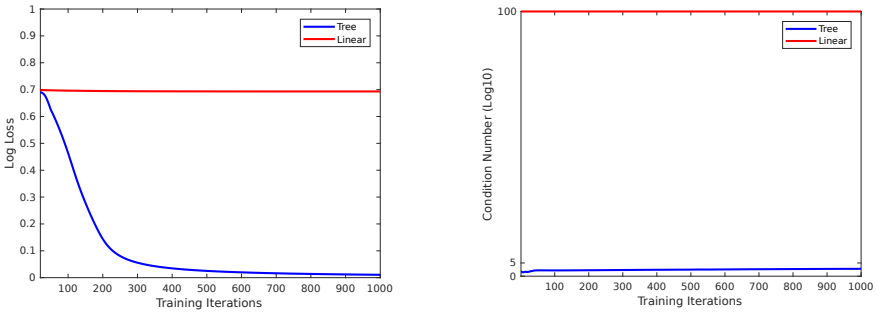
Table 7.2: A possible solution to the 3 dimensional XOR problem using an FC linear layer and 2 depth decision tree using 3 hidden nodes.

training for the two models. We can see that the 2 FC linear layer model with ReLU fails to learn the XOR function and resorts to random guessing. The FC linear layer with decision tree successfully learns the XOR function and its log loss quickly converges towards zero.

Fig. 7.3b shows the condition number as a result of applying Eq. 7.6 to the two models. We observe that the FC linear layer model with a decision tree has a much lower condition number compared to its 2 FC linear layer counterpart. This result provides an insight into why the FC linear layer with decision tree is able to learn the XOR function: it is better conditioned and thus provides gradients that reliably decrease the loss function. We observe this result consistently across higher dimensions of XOR.

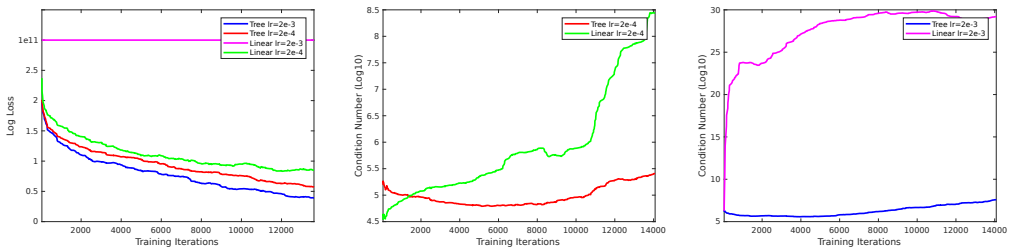
7.4.2 Example: CIFAR-10

Transitioning to real data, we construct two models for classification on the CIFAR-10 data set [116]: the first model uses the discriminator in DCGAN [168]; the second model is a modified version of the DCGAN discriminator with the last FC layer replaced with a decision forest. We train the DCGAN discriminator for image classification on the CIFAR-10 data set [116], using the settings described



(a) **(b)**

Figure 7.3: (a): Log loss of 2 FC linear layer model and FC linear + Decision Tree solving a 3-dimensional XOR function. (b): Conditioning of the 2 FC linear layer and FC linear + Decision Tree models, shown on a logarithmic scale.



(a) **(b)** **(c)**

Figure 7.4: Using a DCGAN discriminator architecture for CIFAR-10 supervised classification task. (a): Log loss of DCGAN discriminator with FC layer and DCGAN discriminator with decision forest for various learning rates. (b), (c): Condition numbers of DCGAN discriminator with FC layer and DCGAN discriminator with decision forest for various learning rates. We note the increased efficiency in learning that a well-conditioned model can provide; in this example, we can increase the learning rate by a factor of 10 for the tree variant of the model, maintain well-conditioned gradients and subsequently learns the given task at an accelerated rate. For the same learning rate ($2e$-3), the baseline model suffers from ill-conditioning and as a consequence, fails to converge.

in its respective paper [168]. Similarly, we also train the DCGAN discriminator with decision forest on the same task. The maximum forest tree depth is chosen such that the number of parameters between both models are similar. Fig. 7.4a shows the log loss of the two models where we can clearly see that the DCGAN discriminator with decision forest converging to a lower loss compared to its counterpart. Similarly, we can use Eq. 7.6 to compute the condition number of the gradients backpropagated from the FC layer and decision forest of their respective models. Fig 7.4b shows the conditioning of the two models where we can observe that the conditioning for the discriminator with the decision forest remains relatively well-conditioned compared to the discriminator with the FC layer.

Incidentally, looking at Fig. 7.4, one of the key takeaways from this analysis is that the conditioning of backpropagated gradients allows for points of instability to be identified during the training process. The same cannot be said when merely observing the training log loss. With this insight, we would expect the better conditioned model to learn in a more stable way if we increase the learning rate. Figs. 7.4a and 7.4c show the effect of increasing the learning rate by a factor of 10 on both models to their respective log losses and condition numbers. The DCGAN discriminator with decision forest is able to train in a stable manner whereas the vanilla DCGAN discriminator cannot; in this case, both the log loss and condition number reflect this. In the following section, we describe implementation details of our method; we show how a decision forest can be constructed as a network layer and specified as a component within the discriminator/critic of a GAN.

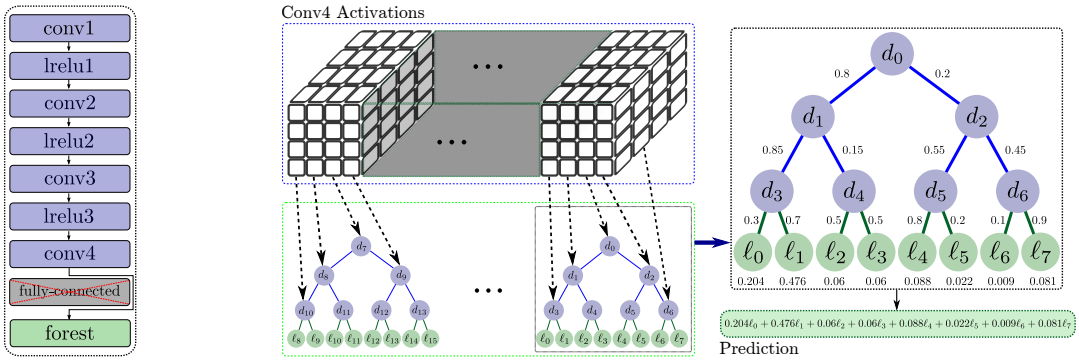
7.5 Generative Adversarial Forests

Our approach modifies the architecture of the discriminator/critic network in a GAN by replacing the final fully-connected (FC) layer of the network with a decision forest. This is shown in Fig. 7.5a. We reformulate the decision nodes in our decision forest such that they are differentiable; hence, the decision forest can be inserted seamlessly into the discriminator/critic network and the whole model can be trained end-to-end. Our method is similar to the approach used in [114] in that we replace the normally hard decision routing function in each decision node with a soft, differentiable sigmoid function. However, we differ in two important aspects:

1. We reconstruct the task of learning leaf node values to jointly learn all values in parallel across the ensemble instead of iteratively learning the values.

2. This in turn allows the use of the soft functionality of decision nodes in our ensemble instead of requiring a stochastic hard routing approximation on the forward pass through the trees, as was done in [114].

In this way, we ensure that the forward pass through our model is consistent with its backward pass and maintain symmetry. Furthermore, in contrast to our method from Chapter 6, this method allows for updating both decision and leaf nodes simultaneously instead of alternating between updates of the two. Our decision forest is used to replace the last FC layer of the discriminator/critic network in a GAN. The set of activations output from the last convolution layer (`conv4`) of the discriminator are reshaped and assigned across the decision nodes in our decision forest (shown in Fig. 7.5b).



(a) (b)

Figure 7.5: An overview of our proposed changes. (a) shows our architecture which modifies the discriminator/critic network by replacing its fully-connected layer with a decision forest (Batch Normalisation/Layer Normalisation is not shown). (b) shows reshaping of `conv4` activations in DCGAN/WGAN-GP to form our decision node parameters.

7.5.1 Soft Decision Trees

Fig. 7.5b shows how each soft decision tree is constructed in our forest. Each tree in the ensemble outputs a single prediction value which is the result of blending the values in all leaves in the tree according to their generated proportion values.

7.5.1.1 Soft Decision Functions

Each decision function in a decision node delivers a value that indicates the proportion of each left and right subtree:

$$d_n(\boldsymbol{x}, \boldsymbol{\Theta}) = \sigma(\alpha_n(x_n - b_n)) \tag{7.8}$$

where $\sigma(x) = (1 + e^{-x})^{-1}$ is a sigmoid function which is similar to the sigmoidal splits described in [194], with α_n indicating its steepness. x_n and b_n are the respective activation and bias values assigned to decision node n. We define $\mu_\ell(\boldsymbol{x}, \boldsymbol{\Theta})$ as the blending function which determines the proportion of contribution

by leaf ℓ towards the tree's final output:

$$\mu_\ell(\boldsymbol{x}|\boldsymbol{\Theta}) = \prod_{n \in \mathcal{N}} d_n(\boldsymbol{x}, \boldsymbol{\Theta})^{1_{\ell \swarrow n}} \bar{d}_n(\boldsymbol{x}, \boldsymbol{\Theta})^{1_{\ell \searrow n}} \tag{7.9}$$

where $\bar{d}_n(\boldsymbol{x}, \boldsymbol{\Theta}) = 1 - d_n(\boldsymbol{x}, \boldsymbol{\Theta})$. 1_C is an indicator function which equals 1 when its condition C is met and 0 otherwise. $\ell \swarrow n$ and $\ell \searrow n$ is defined in [114], and is true if ℓ belongs to either the left or right subtree of node n respectively. Hence, the final prediction value generated by a soft decision tree is given by:

$$Q(\boldsymbol{x}, \boldsymbol{\Theta}) = \sum_{\ell} \mu_\ell(\boldsymbol{x}|\Theta) q_\ell \tag{7.10}$$

Our decision forest changes the role of each output activation of the last convolution layer of the discriminator network; instead of delivering the final prediction, the activation drives the blending proportions output by its assigned decision node. Furthermore, by enforcing our decision trees to make soft decisions which blend leaf values instead of hard routing samples, our model becomes fully differentiable and we are able to easily generate gradients to update our model via backpropagation.

7.5.2 Soft Residual Forest

To combine our ensemble of soft decision trees, we extend upon the method in Chapter 6 and create a layer that acts as an ensemble of residual decision trees which are designed to be jointly optimised in parallel to model the underlying input data. Each decision tree in the ensemble contributes a residual value which is combined with all other residual contributions from other trees in the ensemble. This is achieved by multiplicatively combining the predictive contributions from each tree. Hence, for an input sample the forest outputs a final score value, S, given by:

$$S(\boldsymbol{x}, \boldsymbol{\Theta}, \boldsymbol{Q}) = \prod_{t=1}^{\mathcal{T}} Q^t(D^t(\boldsymbol{x}, \Theta^t)) \tag{7.11}$$

We modify our residual forest framework by adopting a soft approach in both the forward pass and backward passes. Unlike the training schemes in [114] and the approach in Chapter 6, which relied on alternating between updates of the leaf nodes and split nodes, our modification allows for *true* end-to-end training of the decision forest, where both leaf and split nodes are simultaneously updated.

7.6 Experiments

For our experiments, we compared our GAF model to two well-established GAN baselines, DCGAN [168] and WGAN-GP [70], across three datasets: CIFAR-10 [116], CUB Birds [218] and Oxford Flowers [153]. We use both DCGAN and WGAN-GP as baseline GANs in which we replace the final fully-connected (FC) layer in their discriminator/critic networks with our forest layer. This creates two variants of our GAF model and for all our experiments, we refer to our two forest variants of DCGAN and WGAN-GP as DCGAN-Forest and WGAN-GP-Forest respectively.

7.6.1 Experiment Settings

For choice of parameters used in the soft residual forest, we deployed a 16 tree forest with each tree being 8 levels in depth for the CIFAR-10 data set and 9 levels in depth for the CUB Birds and Oxford Flowers datasets. Specifying forest configurations in this manner allows us to maintain approximately the same number of parameters as the FC layer that is being replaced. The CIFAR-10 models consist of 2.6M parameters and the larger models (Oxford Flowers and CUB Birds) consist of 9.7M parameters. We set $\alpha = 1$ and keep all other aspects of the baseline networks consistent in our model, including the hyperparameters with a batch size of 64. For all settings in our models, we keep them consistent with the settings specified in the respective papers of DCGAN and WGAN-GP.

7.6.2 Datasets

7.6.2.1 CIFAR-10

The CIFAR-10 data set [116] consists of 50,000 32×32 training images and 10,000 32×32 testing images evenly distributed across 10 different object class categories. We trained our models along with the DCGAN [168] and WGAN-GP [70] baseline models for 100k iterations.

7.6.2.2 CUB Birds

The CUB Birds data set [218] consists of 11,788 images in 200 different bird categories. We trained our models along with the DCGAN [168] and WGAN-GP [70] baseline models for 50k generator iterations. In Fig. 8.9, we show qualitative samples of our models where a significant improvement in sample quality can be observed over the DCGAN and WGAN-GP baselines.

7.6.2.3 Oxford Flowers

The Oxford Flowers data set consists of 8,189 images separated into 102 different flower categories [153]. We trained our models along with the DCGAN [168] and WGAN-GP [70] baseline models for 50k generator iterations. In Fig. 8.9, we show qualitative samples of our models where we can again observe a significant improvement in sample quality over the DCGAN and WGAN-GP baselines.

7.6.3 Quantitative Results

For quantitatively comparing our model's performance against their respective baselines, we evaluate using the Frechet-Inception Distance (FID) [81] which is now generally considered an improvement on the previously used Inception

(a) DCGAN [168] **(b)** DCGAN-Forest (Ours) **(c)** WGAN-GP [70] **(d)** WGAN-GP-Forest (Ours)

Figure 7.6: Qualitative results on CIFAR10, Oxford Flowers and CUB Birds datasets. We show considerable image quality improvements over both DCGAN and WGAN-GP baselines. Note the increased level of detail in our generated samples, resulting in sharper looking images.

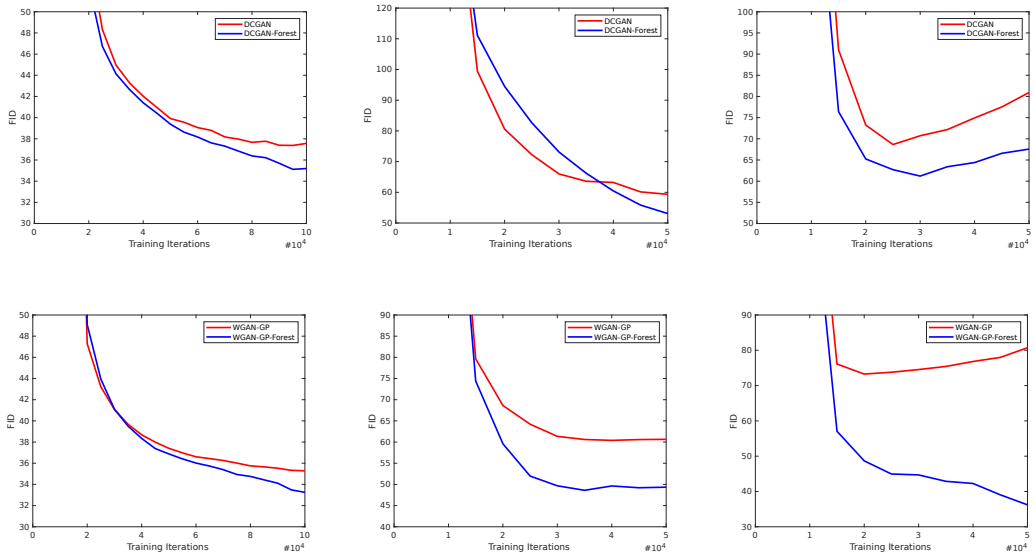
FID Score : $\boldsymbol{m} - \boldsymbol{m}_{w_2}^2 + Tr(\boldsymbol{C} + \boldsymbol{C}_w - 2(\boldsymbol{C}\boldsymbol{C}_w)^{1/2}$		
	DCGAN [168]	DCGAN-Forest (Ours)
CIFAR-10 [116]	37.7	**35.2**
Oxford Flowers [153]	82.0	**67.2**
CUB Birds [218]	59.0	**53.4**
	WGAN-GP [70]	WGAN-GP-Forest (Ours)
CIFAR-10 [116]	35.3	**33.2**
Oxford Flowers [153]	80.4	**35.3**
CUB Birds [218]	60.3	**49.6**

Table 7.3: FID scores (lower is better) for DCGAN and WGAN-GP compared to DCGAN-Forest and WGAN-GP-Forest on CIFAR-10, Oxford Flowers and CUB Birds datasets. The mean result across 5 separate runs is shown.

Score [184]. Hence, the FID score is commonly used as a quantitative measure of a GAN's performance where a lower FID score indicates the better performing GAN. In Table 7.3, we list FID scores of the baseline DCGAN and WGAN-GP models, comparing them with the FID scores of our DCGAN-Forest and WGAN-GP-Forest models. We can see that our models offer a significant improvement over both baselines on FID scores across all three datasets. Note in particular the large improvement that our WGAN-GP-Forest model offers over the baseline WGAN-GP model on the Oxford Flowers data set, which is also consistent with the sample quality shown in Fig. 8.9.

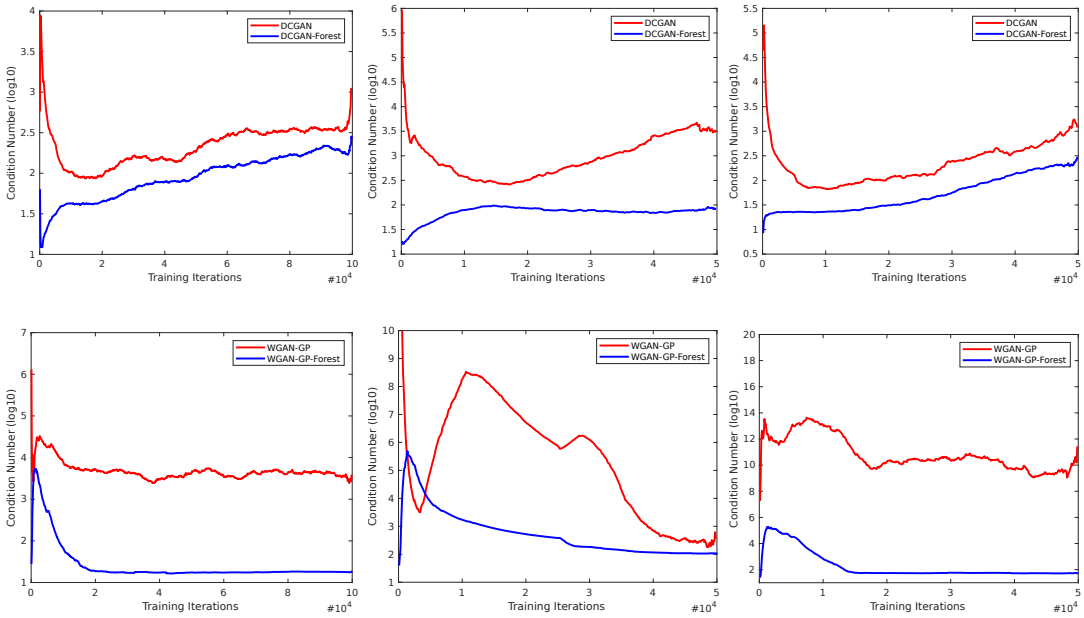
7.6.4 FID Scores and Conditioning

In Fig. 7.7 we show the FID score curves for the CIFAR-10, CUB Birds and Oxford Flowers datasets over 50/100k training iterations. The top row of Fig. 7.7 observes the FID score performance of our DCGAN-Forest model and the baseline DCGAN model. We can observe that our DCGAN-Forest model achieves a lower FID score curve when compared to the baseline DCGAN model across all three datasets. Similarly, the bottom row of Fig. 7.7 observes the FID score performance of our WGAN-GP-Forest model and the baseline WGAN-GP model. We can again observe that our WGAN-GP-Forest model achieves a lower FID score curve when



(a) CIFAR-10 **(b)** CUB Birds **(c)** Oxford Flowers

Figure 7.7: Top: FID curves for DCGAN and DCGAN-Forest models on the CIFAR-10, CUB Birds and Oxford Flowers datasets. **Bottom**: FID curves for WGAN-GP and WGAN-GP-Forest models on the CIFAR-10, CUB Birds and Oxford Flowers datasets. Across both baseline models of DCGAN and WGAN-GP, we note that our model achieves a lower FID score over 100k training iterations for CIFAR-10 and 50k training iterations for CUB Birds and Oxford Flowers.



(a) CIFAR-10 **(b)** CUB Birds **(c)** Oxford Flowers

Figure 7.8: Top: Condition Number curves for DCGAN and DCGAN-Forest models on the CIFAR-10, CUB Birds and Oxford Flowers datasets. **Bottom**: Condition Number curves for WGAN-GP and WGAN-GP-Forest models on the CIFAR-10, CUB Birds and Oxford Flowers datasets. Across both baseline models of DCGAN and WGAN-GP, we note that our model is better conditioned than the baseline over 100k training iterations for CIFAR-10 and 50k training iterations for CUB Birds and Oxford Flowers.

compared to the baseline WGAN-GP model across all three datasets.

Correspondingly, looking at Fig. 7.8, we show the condition number curves for the CIFAR-10, CUB Birds and Oxford Flowers datasets over the same 50/100k training iterations. The top row of Fig. 7.8 observes the condition number curves of our DCGAN-Forest model and the baseline DCGAN model. We can observe that our DCGAN-Forest model achieves a consistently lower condition number curve when compared to the baseline DCGAN model across all three datasets. Similarly, the bottom row of Fig. 7.8 observes the condition number curves of our WGAN-GP-Forest model and the baseline WGAN-GP model where we can again observe that our WGAN-GP-Forest model achieves a lower condition number curve when compared to the baseline WGAN-GP model across all three datasets.

(a) DCGAN (Top) DCGAN-Forest (Bottom)

(b) WGAN-GP (Top) WGAN-GP-Forest (Bottom)

Figure 7.9: Evolution of generator samples on the CIFAR-10, CUB Birds and Oxford Flowers datasets over 50k training iterations for DCGAN, DCGAN-Forest, WGAN-GP and WGAN-GP-Forest models.

7.6.5 Training Computation Complexity

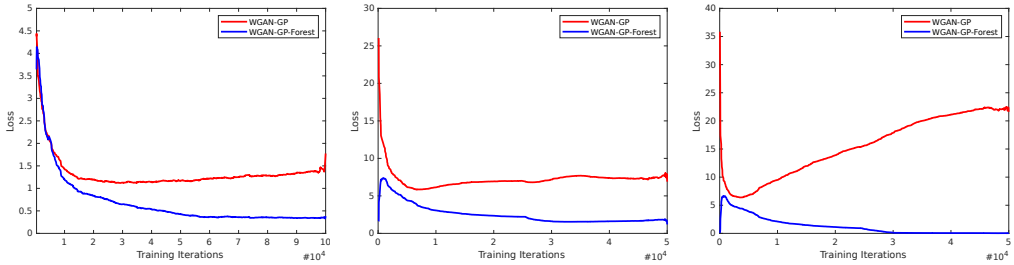
In Table 7.4, we show the training time required for our GAF models and respective GAN baselines for the Oxford Flowers and CUB Birds datasets. Since our forest now makes soft decisions, each branch in each tree in the ensemble must now be evaluated during the forward pass to compute the full set of routing probabilities for leaf nodes. However, our GAF models incur only a minor training time penalty of a relative 2.6% and 9.8% when compared to their respective DCGAN and WGAN-GP baselines.

Training Time (*iters/sec*)			
DCGAN	DCGAN-Forest	WGAN-GP	WGAN-GP-Forest
0.379	0.369	0.113	0.103

Table 7.4: Training and inference timings for Oxford Flowers and CUB Birds datasets. The mean result across 5 separate runs is shown.

7.6.6 Measuring the Critic Loss

Finally, we look towards the critic loss for further evidence that our method improves GAN training. As demonstrated in [2], one of the advantages the Wasserstein distance offers in the critic network is an interpretable loss func-



(a) CIFAR-10 **(b)** CUB Birds **(c)** Oxford Flowers

Figure 7.10: Critic loss curves for WGAN-GP and WGAN-GP-Forest over (a) 100k training iterations for CIFAR-10 (b) 50k training iterations for CUB Birds and (c) 50k training iterations for Oxford Flowers. Across all 3 datasets, we note that the critic loss for our WGAF-GP model shows better critic loss convergence over the WGAN-GP baseline.

tion for assessing the progression of GAN training. [2] showed that the critic loss decreases consistently as training progresses, along with an increase in sample quality. In Fig. 7.10, we show critic losses of the baseline WGAN-GP model with the critic losses of our WGAN-GP-Forest model over 100k training iterations for the CIFAR-10 data set and 50k training iterations for CUB Birds and Oxford Flowers datasets. We can observe that across all 3 datasets, the critic loss of our WGAN-GP-Forest model converges significantly better than the baseline WGAN-GP model. In Fig. 7.9b, we show sample evolution of our WGAN-GP-Forest model compared to the WGAN-GP baseline model on the CUB Birds and Oxford Flowers datasets. Qualitatively, the evolution of generated samples for our DCGAN-Forest and WGAN-GP-Forest models appears to be better than their respective DCGAN and WGAN-GP baselines. The generator's sample evolution for WGAN-GP and WGAN-GP-Forest for CUB Birds and Oxford Flowers also appears consistent with the critic loss curves shown in Figs. 7.10b and 7.10c - the samples from our WGAN-GP-Forest model appears to evolve to a higher level of visual quality when compared to samples from the WGAN-GP baseline.

7.7 Discussion and Summary

This work investigates the optimisation of deep neural networks from a perspective of conditioning on per-exemplar backpropagated gradients $w.r.t$ its parameters. Our analysis gives a key insight that the training speed and stability of a deep network is tightly connected to this distribution of gradients. As such, we offer an architectural modification which improves conditioning of deep networks; we demonstrate that decision forests can provide better conditioning to achieve greater training stability and offer increased learning efficiency. We assess our method by applying it to GANs and offer comparisons with two well-established GAN baselines. We show significant improvements in FID scores on 3 datasets (CIFAR-10, CUB Birds and Oxford Flowers) demonstrating both qualitative and quantitative performance gains in image generation. Finally, we show that our better conditioning of a GAN correlates favourably with critic loss curves in a WGAN-GP setup when compared to the baseline.

Chapter 8

Traversing Latent Space using Decision Ferns

The following chapter is largely drawn from a collaborative work performed with Gil Avraham and will likely appear in a similar form in his thesis. In this work, we investigated how a constructed latent space can be explored in a controlled manner and demonstrate that this complements well founded inference methods. We use a Variational Autoencoder for constructing the latent space, using a novel, decision fern controller to smoothly traverse the latent space. This approach allowed for improved performance of complex inference tasks such as video prediction over existing methods.

The relevant contribution percentages from my end to produce this work are listed as follows (mutually agreed upon by both first authors):

- Idea conception (10%)
- Network architecture design (70%)
- Loss function design (50%)
- Coding the network (50%)
- Training of models (80%)
- Testing and evaluation (20%)

8.1 Introduction

Interpreting visual information from the surrounding environment to subsequently perform meaningful actions is a large part of human perception and understanding. However, these interpretations are not applied directly to what is being seen; rather, there exists an abstraction mechanism from the image space into a more informative space so that complex inferences can be made in this abstracted space [210]. Similarly in machine learning, we would like machines to inherit this ability to abstract as it is the key towards understanding and learning about data from the real world.

Although deep learning has shown great success across various vision-related tasks such as image classification, object detection and semantic segmentation [63, 79, 117, 135, 201], this level of success has yet to transition across to more complicated tasks such as video prediction. Many of the popular deep learning methods approach these more challenging tasks in a similar manner to image classification or segmentation, choosing to learn directly from the image space [136, 140]. This presents a challenge because often the image space is high-dimensional and complex; there exists a large semantic gap between the input pixel representation of the data and the desired transformation of said data for complex inference tasks (such as prediction).

8.1.1 Contributions

To address this challenge, we leverage the compact encoding space provided by a Variational Autoencoder (VAE) [108] to learn complex, higher order tasks in a more feasible way. Inference in the latent space takes advantage of solving a far more tractable problem than performing the operation in the image space as it gives a more natural way of separating the task of image construction and inferring semantic information. Other works have similarly utilised a compact encoding

space to learn complex functions [119, 130, 186, 225]; however, even this encoding space can be strongly entangled and highly non-linear. As such, we construct a residual decision fern based architecture which serves as a controller module that provides the necessary non-linearity to properly disentangle encodings in the latent space. To this end, we introduce a novel framework for controlled traversal of the latent space called the Latent Space Traversal Network (LSTNet) and offer the following contributions:

- We discuss the benefits of operating in a latent space for complex inference tasks, introducing a novel decision fern based controller module which enables the use of control variables to traverse the latent space.

- We create a unified, end-to-end trainable framework which incorporates our controller module with a residual VAE framework, offering an encoding space for learning high order inference tasks on real world data. Additionally, this framework offers a key insight into separating the tasks of pixel reconstruction and high order inference.

- We demonstrate significant qualitative and quantitative improvements offered by LSTNet over popular models that impose geometrical and kinematic constraints on the prediction search space across the MNIST and KITTI datasets.

8.2 Related Work

8.2.1 Learning Representations for Complex Inference Tasks

Operating in the latent space for demonstrating certain properties has been shown in several works. The work of Radford *et al.* applied a convolution architecture to a GAN framework [168] and observed the construction of the latent space. They showed that applying arithmetic operations between two latent vectors observes the corresponding semantic logic in the image space we would expect from such an operation. Similarly, InfoGAN [26] added a regularisation term which maximises mutual information between image space and some latent variables, allowing visual aspects of images generated from their corresponding latent variables to be controlled. Both these works claim to yield a disentangled latent vector representation of the high dimensional input data, demonstrating this by choosing a specific latent variable and interpolating across two values and showing smooth image transformation. However, due to their unsupervised nature, there are no guarantees on what attributes they will learn and how this will distort the intrinsic properties of the underlying data.

The work of Kulkarni *et al.* divides the learned latent space in a VAE into extrinsic and intrinsic variables [119]. The extrinsic variables are forced to represent controllable parameters of the image and the intrinsic parameters represent the appearance that is invariant to the extrinsic values. Although it is trained in a VAE setting, this method requires full supervision for preparing training batches. Yoo *et al.* demonstrated a fully supervised method, introducing a Gaussian Process Regression (GPR) in the constructed latent space of a VAE [225]. However, using GPR imposes other limitations and assumptions that do not necessarily apply in training a VAE. Santana *et al.* used a semi-supervised approach for video prediction, training a VAEGAN [121] instead of a standard VAE [186].

The VAEGAN leads to sharper looking images but imposes a discriminator loss which complicates the training procedure drastically. This work is similar to ours in that control variables are used to guide learning in the latent space. However, this method is not end-to-end trainable and uses a simple framework for processing the latent space; Lotter *et al.* [136] showed this framework underperformed in next frame prediction and next frame steering prediction.

Mathieu *et al.* [140] and Liang *et al.* [130] offered a video prediction framework which combines an adversarial loss along with an image gradient difference/optical flow loss function to perform next frame prediction and multi-step frame prediction. Similarly, Lotter *et al.* [136] performed predictive coding by implementing a convolutional LSTM network that is able to predict the next frame as well as perform multi-step frame prediction in a video sequence. In contrast to our work and the work of Santana *et al.* [186], these works optimise learning in the image space. This approach suffers from poor semantic inference (especially over large time intervals) and will be a focus of investigation in our work.

8.3 Background

8.3.1 Variational Autoencoders

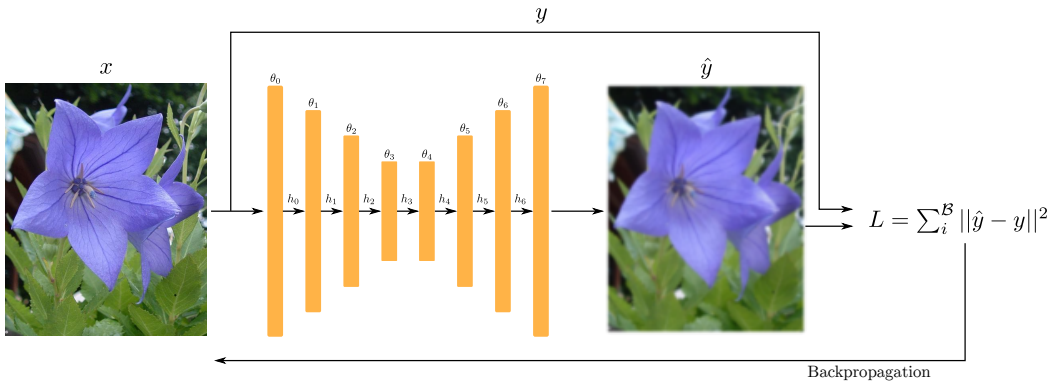


Figure 8.1: An autoencoder with encoder-decoder setup. The autoencoder attempts to encode the input into a latent vector (shown here as h_3) through the encoder network and then subsequently reconstructs the latent vector back to the input image via the decoder network. The loss is generated using the mean-squared error across all pixels between the reconstruction $\hat{y}$ and the input y.

Variational Autoencoders (VAEs) minimise the Evidence Lower Bound L_{ELBO} or variational bound:

$$\begin{aligned} \log P(X) - \mathcal{D}[\mathcal{Q}(z|X)||P(z|X)] = \\ E_{z\sim\mathcal{Q}}[logP(X|z)] - \mathcal{D}[\mathcal{Q}(z|X)||P(z)] \end{aligned} \tag{8.1}$$

As the name implies, VAEs use an encoder-decoder setup of autoencoders as shown in Fig. 8.1. They minimise the variational bound by constructing an encoder to act as $Q(z|X)$ and decoder for $P(X|z)$. The prior probability $P(z)$ is conveniently chosen to be a normal distribution so that the KL divergence on the right hand side of Eq. 8.1 can be computed in a closed form. In the case that the encoder matches the decoder perfectly (meaning the encoded latent vector z can perfectly reconstruct sample X), the KL divergence on the left hand side of Eq. 8.1 will zero out, and the lower bound will be hit. Minimising

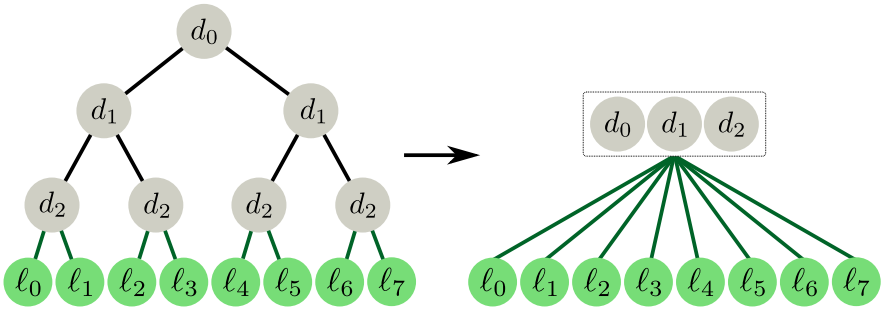


Figure 8.2: A decision tree can be easily converted into a decision fern by changing the decision function for each decision node at each tree level depth to be the same. In this way, each decision can be evaluated independently, regardless of the outcomes of preceding decision nodes.

the KL divergence between approximation of the true distribution towards the true distribution is equivalent to Maximum Likelihood Estimation. As a result, Variational Autoencoders are characterised as generative models that produce a diverse set of generated samples, which attempt to minimise an objective function that satisfies all the samples in an empirical true distribution.

8.3.2 Decision Ferns

A decision fern is related to a decision tree. Similarly to decision trees, decision ferns are non-hierarchical models which route samples from the root decision node to a prediction leaf node. The key difference between a decision fern and decision tree is that the former consists of a set of binary tests which are evaluated simultaneously rather than in a sequential, hierarchical manner [158]. In Fig 8.2, we illustrate how a decision tree can be reconstructed as a decision fern. As shown, a decision tree can easily be converted into a decision fern by using the same decision function for all nodes at any given tree depth level. Thus, a decision fern can be thought of as a constrained decision tree, where the same binary test is performed at each depth level of the tree. This allows all decisions in a decision fern to be evaluated simultaneously, rather than in the hierarchical manner of decision trees.

8.4 Operating in Latent Space

To operate in latent space, there must exist a mechanism which allows for transition between the latent space and image space. Hence, there is an inherent trade off between obtaining a good semantic representation of the image space via its corresponding latent space, and the reconstruction error introduced when transitioning back to the image space. In this work, we show that the benefits of working in a compact latent space far outweigh the loss introduced by image reconstruction. The latent space emphasises learning of the underlying changes in semantics related to an inference task which is paramount when learning high order inference tasks.

8.4.1 Constructing a Latent Space

When constructing a latent space, several viable options exist as candidates. The most naive method would be to perform Principal Component Analysis (PCA) directly on images [209], selecting the N most dominant components. However, for high dimensional spaces such as the natural image space, this method can result in a large loss in information and is less than ideal. Generative Adversarial Networks (GANs) [2, 68, 168] can produce realistic looking images from a latent vector but lack an encoder for performing inference and are difficult to train. Techniques which combine GANs with Variational Autoencoders [39, 40, 121, 142, 174] offer an inference framework for encoding images but prove to be cumbersome to train due to many moving parts and instability accompanying their adversarial training schemes.

Hence, to construct our latent space, we use the relatively straightforward framework of a Variational Autoencoder (VAE) [108]. VAEs contain both a decoder as well as an encoder; the latter of which is used to infer a latent vector given an image. This encoding space offers a low dimensional representation of the

input and has many appealing attributes: it is semi-smooth and encapsulates the intrinsic properties of image data. It is separable by construction; it maintains an encoding space that embeds similar objects in the image space near each other in the latent space. Furthermore, a VAE can be trained in a stable manner and in an unsupervised way, making it an ideal candidate to learn complex higher order inference tasks.

8.4.2 Traversing the Latent Space

Recent works attempted to learn the latent space transformation using various models, such as RNNs [186] and a Gaussian Process Regression [225]. In our work, we recognise that although the original input data is reduced to a lower dimension encoding space, inference on this space is a complex operation over a space which, if constructed correctly, has no redundant dimensions; hence all latent variables should be utilised for the inference task. Under the assumption of a smooth constructed manifold and a transformation that traverses this smooth manifold under a narrow constraint (in the form of a control variable or side information), a reasonable model for the controller module is:

$$z_{t+h} = z_t + F(z_t, z_{t-1}, z_{t-2}, ..., \theta) \tag{8.2}$$

where $\{z_t, z_{t-1}, z_{t-2}...\}$ are the latent vectors corresponding to input data $\{x_t, x_{t-1}, x_{t-2}...\}$, θ is the control variable and z_{t+h} is the output of the model corresponding to given the inputs. The operator $F(\mathbf{z}, \theta)$ can be interpreted as:

$$\begin{aligned} \frac{z_{t+h} - z_t}{h} &= \frac{1}{h} F(z_t, z_{t-1}, z_{t-2}, ..., \theta) \\ \frac{\partial z_t}{\partial h} &= F(z_t, z_{t-1}, z_{t-2}, ..., \theta) \end{aligned} \tag{8.3}$$

where $\frac{1}{h}$ can be absorbed into $F(\mathbf{z}, \theta)$ and by doing so, we can interpret it as a residual term that is added for smoothly traversing from input z_t to z_{t+h}, given

side information θ and the history $\{z_t, z_{t-1}, z_{t-2}...\}$. This construction allows us to implement Eq. 8.3 using a neural network, which we denote as the *Transformer Network*. Eq. 8.2 encapsulates the complete controller, which is a residual framework that delivers the final transformed latent vector (denoted as the *Controller Module*). The Transformer Network will be doing the heavy lifting of inferring the correct step to take for obtaining the desired result $\hat{z}_{t+h}$ and as such, should be carefully modelled. In the following section, we discuss our chosen implementation and the considerations that were taken when constructing the Transformer Network.

8.4.3 Latent Space Traversal Network

Our Latent Space Traversal Network (LSTNet) consists of two main components:

1. A VAE with an encoder and decoder. The encoder learns a latent representation to encode the real set of training images into this space. The decoder learns a mapping from latent space back to the image space.

2. A Controller Module (C) with a Transformer Network (TN), that applies an operation in the latent space offered by the VAE.

An overview of our model is shown in Fig. 8.3. In the remainder of this section, we detail and justify the choice of architecture for the components of our model. To construct our latent space, we adopt the approach in [107] and construct a residual encoder and decoder to form our VAE. This residual VAE offers a low-dimensional, dense representation of input image data, allowing for a latent space which makes higher order inference tasks easier to learn in. LSTNet is trained

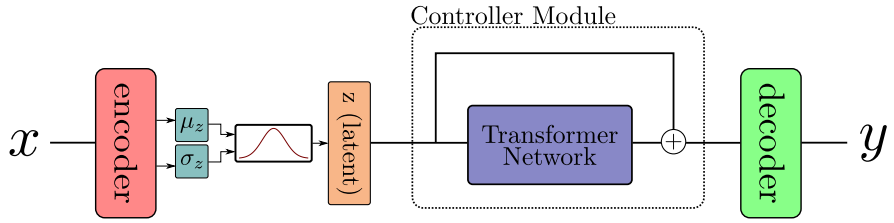


Figure 8.3: Overview of our LSTNet model architecture. x denotes input data and y denotes the output with a transformation applied in the latent space.

on a loss function composed of three terms:

$$\mathbb{L}_{VAE} = \frac{1}{N}\sum_{i\in\mathcal{B}} \hat{I}_i - {I_i}^2 + \frac{1}{N}\sum_{i\in\mathcal{B}} KL(z_i, \mathcal{N}(0, U)), \\ z \sim \mathcal{N}(\mu_{enc}(I), \sigma^2_{enc}(I)) \tag{8.4}$$

$$\mathbb{L}_z = \frac{1}{N}\sum_{i\in\mathcal{B}} z_{target} - {\hat{z}_{target}}^2, \hat{z}_{target} = C(z_{t-n}, ..., z_{t-1}, \theta), \\ z_{target} = \mu_{enc}(I_t) \tag{8.5}$$

$$\mathbb{L}_I = \frac{1}{N}\sum_{i\in\mathcal{B}} I_{target} - {\hat{I}_{target}}^2, \hat{I}_{target} = \mathcal{P}(I_{t-n}, ..., I_{t-1}, \theta), \\ I_{target} = I_t \tag{8.6}$$

where $\mathbb{L}_{VAE}$ is the loss for the VAE which updates the encoder and decoder parameters, $\mathbb{L}_z$ is the controller loss which updates the controller network's parameters and $\mathbb{L}_I$ is the predicted image loss which updates the controller and decoder of the VAE. In this case, I is an image, z is a latent vector in the encoding space, $\mathcal{B}$ is a minibatch, U is an identity matrix, μ_{enc} and σ^2_{enc} are the respective mean and variance of the encoder's output, C is the controller network and $\mathcal{P}$ denotes the LSTNet (passing an input image(s) through the encoder, controller and decoder to generate a transformation/prediction).

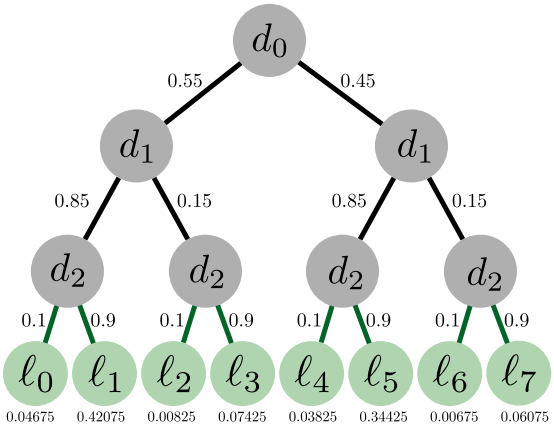


Figure 8.4: A decision fern with soft routing. Each decision node determines a routing portion to subsequent child nodes. The decision nodes along a route determine the final routing portion to all terminating leaf nodes, instead of the hard route approach which selects a single path to a single leaf node.

8.4.3.1 Fern-based Transformer Network

Even in the latent space, learning complex tasks such as video prediction can be difficult. In the experiments section, we motivate the use of the fern-based controller over a linear variant composed of stacked fully connected layers with ReLU non-linearities.

Layer Reparameterisation of Ferns Our transformer network employs an ensemble of soft decision ferns as a core component. The use of soft decision ferns allows them to be differentiable such that they can be integrated and trained within an end-to-end framework. One way to achieve this is to construct decision functions which apply a sigmoid to each input activation biased with a threshold value, yielding a soft value between $[0, 1]$:

$$d_n(\boldsymbol{x}, \boldsymbol{t}) = \sigma((x_n - t_n)) \tag{8.7}$$

where $\sigma(x)$ is a sigmoid function. x_n and t_n are the respective input activation and corresponding threshold values assigned towards the decision. To illustrate this, for a depth two fern using two activations which create the soft routes to its

corresponding four leaves, its output Q is:

$$\begin{aligned} Q = q_0 \times p_0 \times p_1 + q_1 \times p_0 \times (1 - p_1) \\ + q_2 \times (1 - p_0) \times p_1 + q_3 \times (1 - p_0) \times (1 - p_1) \end{aligned} \tag{8.8}$$

where p_0 and p_1 are the respective probability outputs of the decision functions of the decision ferns. q_0, q_1, q_2 and q_3 are the corresponding leaf nodes of the decision fern (illustrated in Fig. 8.4). Eq. 8.8 can be reparameterised as:

$$Q = b + d_0 \times w_1 + d_1 \times w_2 + d_0 \times d_1 \times w_3 \tag{8.9}$$

where:

$$\begin{aligned} d_0 = \tanh(x_0), d_1 = \tanh(x_1) \\ b = \frac{1}{2^h} \times (q_0 + q_1 + q_2 + q_3), w_1 = \frac{1}{2^h} \times (q_0 - q_1 + q_2 - q_3) \\ w_2 = \frac{1}{2^h} \times (q_0 + q_1 - q_2 - q_3), w_3 = \frac{1}{2^h} \times (q_0 - q_1 - q_2 + q_3) \end{aligned} \tag{8.10}$$

x_0 and x_1 are the assigned activations to the decision fern and h is the fern depth. The biases for all ferns are collected together into a single bias term b. w_1, w_2, w_3 can be represented by fully connected linear layers which encapsulates all decision ferns in the ensemble.

Fig. 8.5a shows the architecture of our transformer network. We adopt the residual framework in [79], modifying it for a feed-forward network (FNN) and adding decision fern blocks along the residual branch in the architecture. In Fig.8.5b, we outline the construction of a decision fern building block in the TN, consisting of ferns of two levels in depth. The decision nodes of the fern are reshaped from incoming activations, to which Batch Normalisation [92] and a Hyperbolic Tangent function is applied. This compresses the activations between the range of $[-1, 1]$ and changes their role to that of making decisions on routing to the leaf nodes. A split and multiply creates the conditioned depth two decisions of the fern, which is concatenated with the depth one decisions. Finally, a FC linear layer serves to interpret the decisions made by the decision fern and form the leaf nodes which

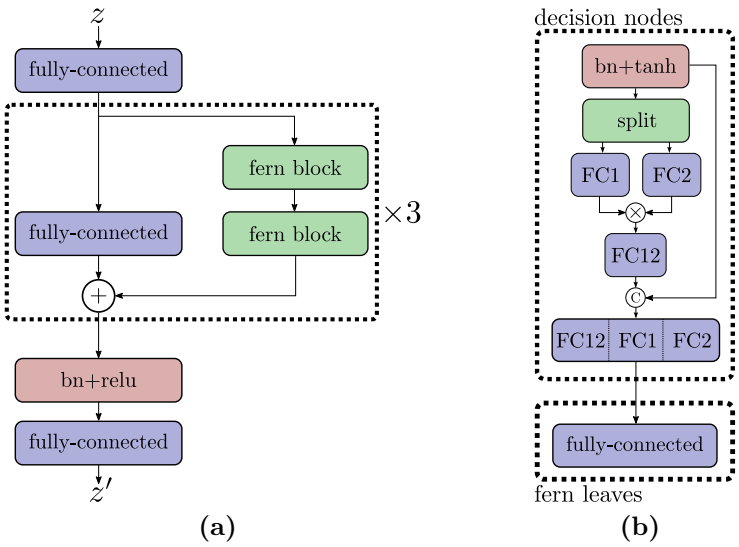


Figure 8.5: (a) The architecture of our proposed transformer network. (b) The structure of a fern block used in our transformer network. We use a reparameterisation trick which allows for the construction of soft decision ferns using common network components such as FC linear layers and Hyperbolic Tangent activation layers.

are free to take any range of values.

Hence, our final controller network is expressed as:

$$C(z_{t-n}, ..., z_{t-1}, \theta) = z_{t-1} + TN(z_{t-n}, ..., z_{t-1}, \theta) \tag{8.11}$$

8.5 Experiments

In our experiments, we explore operation sets that can be achieved by working in the latent space. We choose two applications to focus on - the first application looks towards imposing a spatial transformation constraint on the latent vector whereas the second application is more complex, looking at video prediction. For the first application, we use the MNIST data set [122] as a toy example and investigate rotating and dilation operations on the data set. For the second application, we use the KITTI data set [60] and perform video prediction and steering prediction. For all experiments, we trained using the ADAM Optimiser [106] with learning rate of $1e$-4, first and second moment values of 0.9 and 0.999 respectively, using a batch size of 64.

8.5.1 Imposing Spatial Transformation

For imposing spatial transformations, we present rotating and dilation operations and demonstrate how to constrain the direction a latent vector will traverse by using a spatial constraint. This constrained version of LSTNet applies a transformation to a single image which either rotates or erodes/dilates the given image. For both rotation and dilation experiments, we use a small residual VAE architecture along with our specified controller module with 1 residual layer with decision fern blocks (refer to Fig. 8.5a & 8.5b for details). We choose a latent vector size of 100 dimensions. The encoder consists of 2 residual downsampling layers (refer to [107] for details on the residual layers), Batch Normalisation and ReLU activations in between, ending with 2 fully connected linear layers for emitting the mean and variance of the latent vector. The decoder also consists of 2 residual upsampling layers, Batch Normalisation and ReLU activations in between, ending with a Hyperbolic Tangent function. To compare our method, we use two baselines. The first method is the most obvious comparison; we implement a baseline CNN which learns a target transformation, given an input image and corresponding

control variable θ (CNN-baseline). This CNN-baseline is composed of 2 strided 3x3 convolution layers, 2 FC linear layers and 2 strided 3x3 deconvolution layers with ReLU non-linearities used for activation. The number of output channels in the hidden layers was kept at a constant 128. Additionally, we implement the Deep Convolutional Inverse Graphics Network [119] as specified in their paper for comparison. For each of the three methods compared, we use the same conditions by providing the control variable θ as an input during training and testing.

8.5.1.1 Rotation

We create augmented, rotated samples from the MNIST data set. Specifically, we randomly choose 600 samples from the data, ensuring an even distribution of 60 samples per class label are chosen. For each sample, we generate 45 rotation augmentations by rotating the sample in the range of $-45° < \theta < 45°$. We add this augmented set to the original MNIST data and train the VAE with controller module end-to-end for 20k iterations. To inject the control variable into the input, we concatenate it to the encoded latent vector before feeding it to the controller module.

To train our controller module, we note that there are $\binom{45}{2}$ possible pairs in each example, giving us much more training data than needed. Hence, for every iteration of training, we randomly choose a batch of triplets (I_i, I_j, θ), where θ is a rotation control variable specifying the rotation in radians. For inference, we randomly sample images from the MNIST data set that were not selected to be augmented and perform a rotation parameter sweep. This results in smooth rotation of the image, while preserving the shape (see Fig. 8.6a). Note that other works (*i.e* [26]) have shown that by altering a variable in the latent space, a rotated image can be retrieved, but fails to preserve image shape, leading to distortion. This indicates that the specific variable for rotation is not only responsible for rotation, but has influence over other attributes of the image. In contrast, we observed this difference across several variables between z_i and z_j. This gives the insight that in order to perform a rotation (or similar spatial trans-

(a) (b)

Figure 8.6: (a) Rotation and dilation operations performed by LSTNet on samples from MNIST. The top two rows show rotation, whilst the bottom two rows show dilation. The original samples are shown in the middle highlighted in blue (b) LSTNet applying a combination of rotation and dilation operations to a sample from MNIST.

form operation) and preserve the original image shape, several variables in the latent vector need to change which justifies the use of a highly non-linear network to approximate this operation.

8.5.1.2 Thickening

For learning the dilation operation we created augmentations in a similar manner to rotation operations. We randomly choose 5000 samples (evenly distributed across the 10 class labels) from the original data set, and augmented every sample 4 times with 2 steps of dilation (thickening) and 2 steps of erosion (thinning). We train our VAE and controller module in a similar way to the rotation operation for 20k iterations, specifying batches of triplets (I_i, I_j, θ); θ changes its role to specifying a dilation factor that takes one of 5 discrete values in the range of $[-2, 2]$. Note that although the network was trained on 5 discrete levels of dilation, it manages to learn to smoothly interpolate when performing the operation sweep during inference (as shown in Fig. 8.6a).

8.5.1.3 Combining Operations

One immediate extension that LSTNet offers when performing spatial transformation operations is the ease in which multiple spatial transformation operations

can be combined together into a single framework. In Fig. 8.6b, we show the samples produced by an LSTNet with 2 controller modules, sharing a single latent space offered by the VAE. It is important to note here that neither the rotation or dilation controller modules saw the other's training data. Hence, both operations are applied consecutively in the latent space and decoded back for visualisation.

In Table 8.1, we show the mean squared error (MSE) of LSTNet, comparing against the two baseline methods across rotation, dilation and combined rotation plus dilation operations. We can see that LSTNet outperforms in Rotation and Dilation MSE and handily outperforms in the combined operation. This indicates a generality in learning in the latent space; LSTNet has learned the semantics behind rotation and dilation operations and thus can seamlessly combine these two operations.

Model	Rotation (MSE)	Dilation (MSE)	Rotation+Dilation (MSE)
DCIGN [119]	0.07373484	0.02599725	0.08349568
CNN-baseline	0.02819574	0.00841950	0.06508310
LSTNet	**0.02380177**	**0.00835836**	**0.04410466**

Table 8.1: Mean squared error values on MNIST for rotation, dilation and rotation+dilation operations across CNN-baseline, DCIGN and LSTNet. Note the significant improvement LSTNet offers for the combined operation of rotation and dilation, indicating modularity.

8.5.2 Imposing Kinematics

We now move towards the more complex inference task of video prediction. Similar to imposing spatial transformation, we use a larger, residual VAE architecture along with a larger controller module to account for the more complex inference task. We increase the base dimension of our latent vector to 256 dimensions. The VAE's encoder consists of 3 residual downsampling layers, Batch Normalisation and ReLU activations in between, ending with a fully connected linear layer. The decoder consists of 3 residual upsampling layers, Batch Normalisation and ReLU

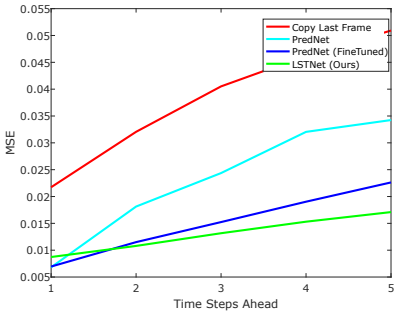
activations in between, ending with a Hyperbolic Tangent function. For the controller network we experiment using two variants. The first is the fern-based controller as described in Section 8.4.3. For motivating the use of the fern-based controller, a linear variant controller network is also used: this is a feed-forward linear network consisting of 4 FC linear layers with ReLU activations, matching the fern-based controller in terms of model capacity.

Similarly to imposing spatial transformations, we randomly create batches of triplets (I_i, I_j, θ), where I_i and I_j are the respective current frame and target future frame (randomly chosen to be within 5 time steps of the current frame) and θ is the corresponding time step from the current frame to the target frame. Our controller module receives the latent vectors of the current frame as well as a sequence of latent vectors belonging to the previous 5 frames to the current frame. We train our framework end-to-end for 150k iterations; the VAE is trained to minimise reconstruction loss along with KL regularisation, the controller is minimised using the latent error between the target latent and predicted latent vectors and both decoder and controller are jointly optimised using the error between target frames and predicted frames (refer to Eqs. 8.4, 8.5 and 8.6). Empirically, we found that training the VAE at a ratio of 5:1 against the controller module resulted in the best performance to training time tradeoff.

Fig. 8.9 shows qualitative results comparing our model with the PredNet model in [136]. We observe that the further the prediction is over time, the less accurate PredNet becomes, whilst LSTNet remains much more robust to changes in the scene over time. Observing the samples of PredNet, a recurring phenomena is that in areas of the frame where object movement should occur, moving objects are instead smeared and blurred. In the case of LSTNet, predicted movement is better observed (particularly over large time steps), whilst vehicles and street furniture (*i.e.* road signs and markings) are better placed. This gives an indication that LSTNet is more reliable for prediction mechanism within the context of the task. PredNet implicitly learns to predict on a fixed timestep and hence predicts over a large time interval by rolling out over its predicted images. On the other

Model	Avg. MSE	Avg. SSIM
Copy Last Frame	0.03829	0.615
PredNet [136]	0.02436	0.604
PredNet (Finetuned) [136]	0.01524	0.679
Linear Controller Variant	0.02083	0.631
LSTNet (Ours)	**0.01316**	**0.694**

(a)



(b)

Figure 8.7: (a) Averaged MSE and SSIM over 5 timesteps of frame prediction (b) Individual MSE for each predicted time step plotted for Copy Last Frame, PredNet and LSTNet.

hand, LSTNet offers a more general approach via the control variable θ. It learns a transformation which allows it to directly predict the target video frame without the computational overhead of rolling out fixed timesteps to reach the target. In Table 8.7a, we depict the average MSE and SSIM over 5 future frames (500ms time lapse) for the KITTI test set. We note that LSTNet outperforms the compared methods of Copy Last Frame, PredNet and a Linear Controller Variant (as discussed in Section 8.4.3). Looking at Fig. 8.7b, we can see that LSTNet achieves lowest MSE for all timesteps, except $\hat{I}_{t+1}$, where PredNet has slightly lower MSE. However, despite slightly higher initial MSE, the performance of LSTNet quickly exceeds both methods as inferring good prediction begins playing a larger factor over time (see Section 8.5.3 for a full discussion). These quantitative results correlate well with our qualitative results and again indicate that LSTNet is able to outperform on the task of prediction rather than on image reconstruction.

8.5.3 Latent Space for Prediction

In addition to the computational and memory footprint benefits, we show that projecting an image onto the latent space, operating on the latent vector using

our controller module and reprojecting back into the pixel space has on average a lower MSE in the pixel space over operations that are increasingly harder to perform; for example: chaining spatial transformations and predicting more than 1 time step into the future.

8.5.3.1 Moving vs. non-moving objects

An attribute that does not favour a latent space framework is the inference of the fine grained detail in images with a lot of static, non-moving components. Fig. 8.8 shows an example of a $\hat{I}_{t+1}$ prediction for PredNet (Finetuned) [136], our LSTNet and the ground truth for comparison. Across these three images, red and green highlighted patches show texture and movements of objects respectively. Across the patches, it is visually apparent that LSTNet does not generate a fine detailed texture of the tree leaves, although it captures the movement of a car well as the camera viewpoint changes. Conversely, PredNet is able to capture the fine grain texture of the tree leaves, but fails to capture the movement of the stationary car from a camera viewpoint change.

We perform a simple patch-based test to illustrate the importance of emphasising prediction on moving objects versus non-moving objects. For $\hat{I}_{t+1}$, we randomly choose 20 patches of 20×20 pixels that we identify where movement occurs; computing the average MSE yielded values for LSTNet=**0.0020** and PredNet=0.0134 showing we significantly improve on predicting movement. Similarly, we sample 20 patches identified as being static with no moving objects which yielded an MSE for LSTNet=0.0024 and PredNet=**0.0021**, with the results favouring PredNet. This result correlates well with the competitive results shown on $\hat{I}_{t+1}$ in Fig. 8.7b; between two consecutive frames, the majority of parts in a scene are static with little to no movement. Hence, prediction plays a smaller role in pixel space MSE over such a short time frame. However, as the time between predicted and current frame increases, getting better predictions on movements plays a larger role, which accounts for the results shown in Fig. 8.7b.

Model	Steering Angle MSE (Degrees^2)
Copy Last Frame	1.3723866
PredNet (FineTuned) [136]	2.4465750
LSTNet (Ours)	**0.5267124**

Table 8.2: Steering angle prediction MSE on the KITTI test data.

8.5.3.2 Auxiliary Parameter Predictions

Furthermore, inferring auxiliary tasks such as steering angle does not require the fine detailed knowledge contained in the pixel space of a scene. For performing such inference tasks, the main requirement is the semantic information contained in the scene. For this task, we used a pretrained LSTNet and added a FC layer to the controller to output a single value and finetuned it to learn the steering angle. This is where LSTNet shines; it considerably outperforms PredNet [136] and copying of steering angles from the last seen frame as shown in Table 8.2. This further correlates with our MSE prediction results that LSTNet is inherently able to distill the semantics from a scene for complex inference tasks.

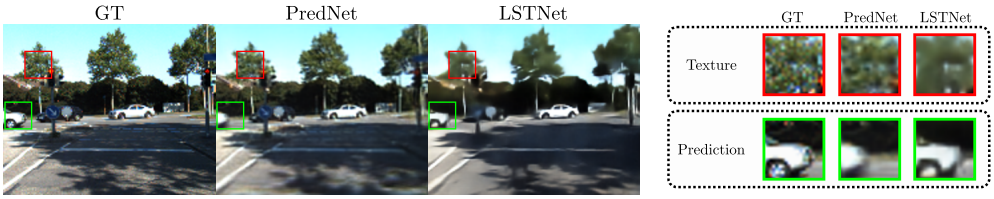


Figure 8.8: On the left hand side we present the next frame prediction $(t+1)$ for the Ground-Truth, PredNet (Finetuned) and our LSTNet. On the right hand side are patches that match the rectangular markings on the images with corresponding labels. Our LSTNet excels at predicting the semantic changes that are important for maintaining the correct structure of a scene; and at times may fail (as shown) at outputting the fine-grained details of the scene objects, due to reconstruction.

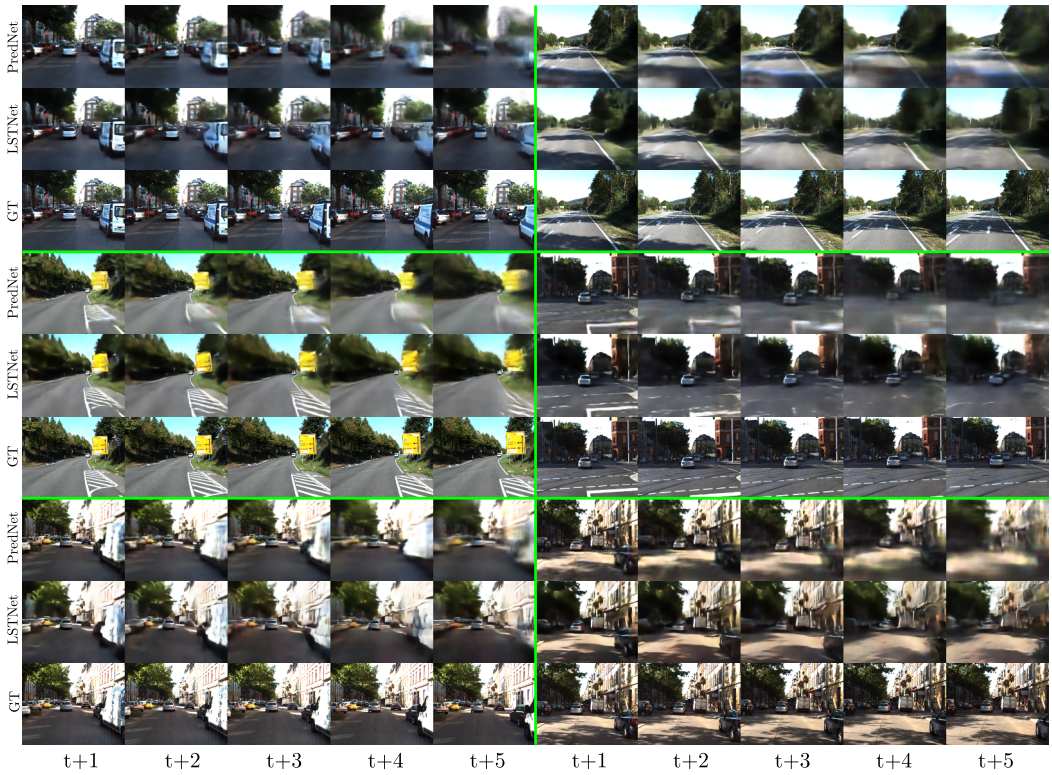


Figure 8.9: Multi-Frame predictions. This figure depicts of 6 sequences selected from the KITTI test set where $\{I_{t+1}, I_{t+2}, I_{t+3}, I_{t+4}, I_{t+5}\}$ are predicted using a past sequence of 5 frames. Each example is organised into 3 rows depicting PredNet (FineTuned) [136], our LSTNet and the ground-truth for comparison.

8.6 Discussion and Summary

In this work, we present a novel, end-to-end trainable framework for operating on a latent space constructed using a VAE. We explore the power of learning in latent space on two operations: spatial transformations and video prediction, and show semantic gains for increasingly harder inference tasks which subsequently translates to a more meaningful result in the pixel space. Furthermore, as a direct extension to this work, the use of a VAE presents an opportunity to explore multi-model predictions for further robustness in predictive tasks.

Chapter 9

Conclusion

In this thesis, we present a number of research outcomes which aim to improve the ensemble learning approaches of decision forests and ferns. Furthermore, we develop a method for integrating these ensemble learning methods into deep learning frameworks, showing that this results in improved performance across a variety of computer vision tasks such as image classification, semantic segmentation and video prediction. We discuss in detail the new techniques introduced and compare our methods with previous related approaches.

9.1 Summary of Contributions

The primary contributions of this thesis are as follows:

- In Chapter 4, we introduce a novel ensemble learning approach called Residual Likelihood Forests, which constructs a decision forest model that combines the benefits from boosting approaches with random forests. This allows our method to leverage the complementary information between base tree learners via minimising a global loss, but still retain the simplicity in model tuning attributed to random forest. We show that this framework demonstrates improved parameter efficiency when selecting split node decisions, which enables the construction of much shallower forest models but retains the performance levels of the deeper tree models of competing decision forest approaches.

- Chapter 5 investigates an approach to extend the residual forest framework towards utilising deep representational features. To this end, we use a pretrained Convolutional Neural Network (CNN) as a feature extractor, which serves to extract meaningful features from images to serve as input to our residual forest classifier. We use the activations of various convolution layers in the CNN to construct our decision forest and apply this model to various semantic segmentation tasks. We detail various architecture choices in our model, objective function modifications and parameter settings which enable our model to combine the two different frameworks of a decision forest and CNN into a single cohesive framework.

- Following this, we extend the residual framework to learn both representations and a classifier in Chapter 6 of this thesis. We develop an efficient learning approach which is able to enables end-to-end training of a hybrid model consisting of a CNN and decision forest. We apply this model towards semantic segmentation tasks; compared to the pure deep learn-

ing CNN baseline, this approach substantially reduces training time and provides noticeable improvements in segmentation performance on training data sets with limited data.

- In Chapter 7, we demonstrate that our decision forests have applications within the domain of generative models. We first show that the application of a soft version of our residual forest framework within a Generative Adversarial Network (GAN) can significantly improve model conditioning and by extension, improve the stability of the model during training. We demonstrate how a soft decision forest can be represented as a network layer, using it to replace the last fully-connected linear layer of the discriminator of the GAN and show that the improved stability yields higher performances over baselines on image generation tasks across several datasets.

- Finally, we show in Chapter 8 that decision ferns can be used to effectively learn controlled traversal within the latent space of a Variational Autoencoder (VAE). We show how a soft variant of decision ferns can be constructed using common neural network building blocks through a reparameterisation trick. Empirically, we show that the additional non-linearity offered by decision ferns can aid in complex inference tasks in the latent space and apply this to image spatial transformation and video prediction tasks. We demonstrate that our decision fern controller can offer significant improvements over competing methods.

9.2 Discussion and Future Work

The contributions outlined in this thesis opens up several areas of extension to the ideas presented. Here, we discuss some of the possible future avenues of research in this area.

Extending Residual Likelihood Forests The proposed method presented in Chapter 4 can be further explored. Since the method determines the splits made by the decision nodes of a decision tree using a global loss minimising criteria rather than a local, greedy entropy minimisation criteria, in this regard, investigating different global loss criteria may be of interest. The current method uses cross-entropy loss as its global loss function; this leaves room for a variety of different loss functions to be explored which may improve performance of model such as Hinge Loss, Exponential Loss and Tangent Loss to name a few.

Binary Representation Forests A natural extension of the decision forest framework presented in Chapter 6 is in the area of binary neural networks. In the context of operating on the activations of a CNN, a decision forest effectively binarises the input activation value to $[0, 1]$ by comparing it against a threshold term. Once the activation value is binarised, there is a loss of information that cannot be recovered from this process. The tradeoff from this process is that decision trees offer a large amount of non-linearity for a given model size. We can avoid the loss of information from the binarising process of decision trees if the original activation inputs are binary. Since binary networks use binary representations as their activations throughout, this makes them an ideal candidate for investigating a hybrid framework which incorporates decision forests or ferns.

Human Pose Generation The method presented in Chapter 7 offers a general

method for improving the conditioning of a Generative Adversarial Network. As such, there are numerous applications such a framework could be employed towards as future avenues of research. One area is in the relatively recent task of performing human pose generation. Given the history of decision trees in human pose estimation tasks, this would be an interesting task to investigate whether a model with decision forests can offer performance benefits. In this case, we can modify the method introduced in Chapter 7 to that of a conditional Generative Adversarial Network which receives an input image and a target pose (*e.g.* as motion captured points) as inputs and tries to generate an output image where the pose of the input image changes to that of the target pose. This could be used to enhance performance on pose estimation tasks whereby the training data is augmented by generated image poses.

Latent Space Traversal for Reinforcement Learning The video prediction framework presented in Chapter 8 could also serve as a baseline for improving upon reinforcement learning methods. The proposed video prediction framework could serve as a mechanism for forward predicting into the future for an agent in a reinforcement learning framework. This could allow the agent to predict future events as a result of taking some specified action, along with the associated reward from that action. Allowing the agent to forward predict using this video prediction framework could allow it to decide upon certain actions that are deemed beneficial as well as eliminate actions that could potentially harm its progress later into the future.

References

[1] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural computation*, 9(7):1545–1588, 1997.

[2] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223, 2017.

[3] A. Asuncion and D. Newman. Uci machine learning repository, 2007.

[4] C. Audet and M. Kokkolaras. Blackbox and derivative-free optimization: theory, algorithms and applications, 2016.

[5] P. Baldi and Y. Chauvin. Neural networks for fingerprint recognition. *Neural Computation*, 5(3):402–418, 1993.

[6] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[7] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[8] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[9] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[10] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's razor. *Information processing letters*, 24(6):377–380, 1987.

[11] A. Bosch, A. Zisserman, and X. Munoz. Image classification using random forests and ferns. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007.

[12] R. N. Bracewell and R. N. Bracewell. *The Fourier transform and its applications*, volume 31999. McGraw-Hill New York, 1986.

[13] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[14] L. Breiman. Arcing the edge. Technical report, Technical Report 486, Statistics Department, University of California at Berkeley, 1997.

[15] L. Breiman. Prediction games and arcing algorithms. *Neural computation*, 11(7):1493–1517, 1999.

[16] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[17] L. Breiman. *Classification and regression trees*. Routledge, 2017.

[18] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla. Segmentation and recognition using structure from motion point clouds. In *European conference on computer vision*, pages 44–57. Springer, 2008.

[19] P. Bühlmann and S. Van De Geer. *Statistics for high-dimensional data: methods, theory and applications*. Springer Science & Business Media, 2011.

[20] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[21] R. Caruana, N. Karampatziakis, and A. Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 96–103. ACM, 2008.

[22] R. Caruana, N. Karampatziakis, and A. Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 96–103. ACM, 2008.

[23] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168. ACM, 2006.

[24] O. Chapelle, B. Scholkopf, and A. Zien. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

[25] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[26] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180, 2016.

[27] M. B. Christopher. *PATTERN RECOGNITION AND MACHINE LEARNING.* Springer-Verlag New York, 2016.

[28] P. Comon. Independent component analysis, a new concept? *Signal processing*, 36(3):287–314, 1994.

[29] T. M. Cover and J. A. Thomas. *Elements of information theory.* John Wiley & Sons, 2012.

[30] J. Dai, K. He, and J. Sun. Convolutional feature masking for joint object and stuff segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3992–4000, 2015.

[31] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[32] B. David, E. Kuh, and R. Welsch. Regression diagnostics: identifying influential data and sources of collinearity, 1980.

[33] T. E. De Campos, B. R. Babu, M. Varma, et al. Character recognition in natural images. *VISAPP (2)*, 7, 2009.

[34] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[35] L. Deng, D. Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

[36] T. G. Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[37] P. Dollár and C. L. Zitnick. Structured forests for fast edge detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1841–1848, 2013.

[38] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655, 2014.

[39] J. Donahue, P. Krähenbühl, and T. Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

[40] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2016.

[41] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2650–2658, 2015.

[42] G. Einicke. Smoothing, filtering and prediction: Estimating the past, present and future. rijeka, croatia: Intech. Technical report, ISBN 978-953-307-752-9, 2012.

[43] B. Everitt and A. Skrondal. *The Cambridge dictionary of statistics*, volume 106. Cambridge University Press Cambridge, 2002.

[44] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1915–1929, 2013.

[45] S. J. Farlow. *Self-organizing methods in modeling: GMDH type algorithms*, volume 54. CrC Press, 1984.

[46] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[47] Y. Freund. Boosting a weak learning algorithm by majority. In *COLT*, volume 90, pages 202–216, 1990.

[48] Y. Freund. Boosting a weak learning algorithm by majority. *Information and computation*, 121(2):256–285, 1995.

[49] Y. Freund. An adaptive version of the boost by majority algorithm. *Machine learning*, 43(3):293–318, 2001.

[50] Y. Freund and R. E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

[51] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296, 1999.

[52] Y. Freund, R. E. Schapire, et al. Experiments with a new boosting algorithm. In *icml*, volume 96, pages 148–156. Citeseer, 1996.

[53] B. J. Frey. *Graphical models for machine learning and digital communication*. MIT press, 1998.

[54] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, NY, USA:, 2001.

[55] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[56] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[57] K. Fukushima. Artificial vision by multi-layered neural networks: Neocognitron and its advances. *Neural networks*, 37:103–119, 2013.

[58] J. Gall and V. Lempitsky. Class-specific hough forests for object detection. In *Decision forests for computer vision and medical image analysis*, pages 143–157. Springer, 2013.

[59] R. Ge, F. Huang, C. Jin, and Y. Yuan. Escaping from saddle pointsonline stochastic gradient for tensor decomposition. In *Conference on Learning Theory*, pages 797–842, 2015.

[60] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[61] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE, 2012.

[62] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[63] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[64] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Region-based convolutional networks for accurate object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):142–158, 2016.

[65] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[66] I. Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[67] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[68] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[69] S. Gould, T. Gao, and D. Koller. Region-based segmentation and object detection. In *Advances in neural information processing systems*, pages 655–663, 2009.

[70] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

[71] S. Gupta, P. Arbelaez, and J. Malik. Perceptual organization and recognition of indoor scenes from rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 564–571, 2013.

[72] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik. Learning rich features from rgb-d images for object detection and segmentation. In *European Conference on Computer Vision*, pages 345–360. Springer, 2014.

[73] J. Hadamard. Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton university bulletin*, pages 49–52, 1902.

[74] M. T. Hagan and M. B. Menhaj. Training feedforward networks with the marquardt algorithm. *IEEE transactions on Neural Networks*, 5(6):989–993, 1994.

[75] B. Hariharan, P. Arbeláez, L. Bourdev, S. Maji, and J. Malik. Semantic contours from inverse detectors. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 991–998. IEEE, 2011.

[76] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Simultaneous detection and segmentation. In *European Conference on Computer Vision*, pages 297–312. Springer, 2014.

[77] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.

[78] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[79] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, 2016.

[80] G. Heitz, S. Gould, A. Saxena, and D. Koller. Cascaded classification models: Combining models for holistic scene understanding. In *Advances in Neural Information Processing Systems*, pages 641–648, 2009.

[81] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017.

[82] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[83] T. K. Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE, 1995.

[84] T. K. Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.

[85] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[86] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017.

[87] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 2, pages 985–990. IEEE, 2004.

[88] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

[89] J. J. Hull. A database for handwritten text recognition research. *IEEE Transactions on pattern analysis and machine intelligence*, 16(5):550–554, 1994.

[90] S. Ikeda, M. Ochiai, and Y. Sawaragi. Sequential gmdh algorithm and its application to river flow prediction. *IEEE Transactions on Systems, Man, and Cybernetics*, (7):473–479, 1976.

[91] Y. Ioannou, D. Robertson, D. Zikic, P. Kontschieder, J. Shotton, M. Brown, and A. Criminisi. Decision forests, convolutional networks and the models in-between. *arXiv preprint arXiv:1603.01250*, 2016.

[92] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[93] A. Ivakhnenko. *Cybernetic predicting devices*.

[94] A. Ivakhnenko. Heuristic self-organization in problems of engineering cybernetics. *Automatica*, 6(2):207–219, 1970.

[95] A. Ivakhnenko and G. Ivakhnenko. The review of problems solvable by algorithms of the group method of data handling (gmdh). *Pattern Recognition And Image Analysis C/C Of Raspoznavaniye Obrazov I Analiz Izobrazhenii*, 5:527–535, 1995.

[96] A. G. Ivakhnenko. The group method of data of handling; a rival of the method of stochastic approximation. *Soviet Automatic Control*, 13:43–55, 1968.

[97] A. G. Ivakhnenko. Polynomial theory of complex systems. *IEEE transactions on Systems, Man, and Cybernetics*, (4):364–378, 1971.

[98] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2011.

[99] H. Jegou, F. Perronnin, M. Douze, J. Sánchez, P. Perez, and C. Schmid. Aggregating local image descriptors into compact codes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(9):1704–1716, 2012.

[100] R. D. Joseph. *Contributions to perceptron theory*. Cornell Univ., 1961.

[101] S. I. Kabanikhin. Definitions and examples of inverse and ill-posed problems. *Journal of Inverse and Ill-Posed Problems*, 16(4):317–357, 2008.

[102] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[103] M. Kearns. Learning boolean formulae or finite automata is as hard as factoring. *Technical Report TR-14-88 Harvard University Aikem Computation*

Laboratory, 1988.

[104] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.

[105] C. Keskin, F. Kıraç, Y. E. Kara, and L. Akarun. Hand pose estimation and hand shape classification using multi-layered randomized decision forests. In *European Conference on Computer Vision*, pages 852–863. Springer, 2012.

[106] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[107] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, pages 4743–4751, 2016.

[108] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[109] B. Kolb and I. Q. Whishaw. *Fundamentals of human neuropsychology*. Macmillan, 2009.

[110] T. Kondo. Gmdh neural network algorithm using the heuristic self-organization method and its application to the pattern identification problem. In *SICE'98. Proceedings of the 37th SICE Annual Conference. International Session Papers*, pages 1143–1148. IEEE, 1998.

[111] T. Kondo and J. Ueno. Multi-layered gmdh-type neural network self-selecting optimum neural network architecture and its application to 3-

dimensional medical image recognition of blood vessels. *International Journal of innovative computing, information and control*, 4(1):175–187, 2008.

[112] P. Kontschieder, S. R. Bulo, H. Bischof, and M. Pelillo. Structured class-labels in random forests for semantic image labelling. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2190–2197. IEEE, 2011.

[113] P. Kontschieder, S. R. Bulò, A. Criminisi, P. Kohli, M. Pelillo, and H. Bischof. Context-sensitive decision forests for object detection. In *Advances in neural information processing systems*, pages 431–439, 2012.

[114] P. Kontschieder, M. Fiterau, A. Criminisi, and S. Rota Bulo. Deep neural decision forests. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1467–1475, 2015.

[115] P. Kontschieder, P. Kohli, J. Shotton, and A. Criminisi. Geof: Geodesic forests for learning coupled predictors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 65–72, 2013.

[116] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[117] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[118] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[119] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum. Deep convolutional inverse graphics network. In *Advances in Neural Information Processing Systems*, pages 2539–2547, 2015.

[120] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab. Deeper depth prediction with fully convolutional residual networks. In *3D Vision (3DV), 2016 Fourth International Conference on*, pages 239–248. IEEE, 2016.

[121] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015.

[122] Y. LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

[123] Y. LeCun. Une procedure d'apprentissage ponr reseau a seuil asymetrique. *proceedings of Cognitiva 85*, pages 599–604, 1985.

[124] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[125] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer, 1999.

[126] Y. LeCun, D. Touresky, G. Hinton, and T. Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pages 21–28. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988.

[127] A. M. Legendre. *Nouvelles méthodes pour la détermination des orbites des comètes*. F. Didot, 1805.

[128] S. Leutenegger, M. Chli, and R. Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555. IEEE, 2011.

[129] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan. A fast and accurate online sequential learning algorithm for feedforward networks. *IEEE Transactions on neural networks*, 17(6):1411–1423, 2006.

[130] X. Liang, L. Lee, W. Dai, and E. P. Xing. Dual motion gan for future-flow embedded video prediction. *arXiv preprint*, 2017.

[131] G. Lin, A. Milan, C. Shen, and I. D. Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In *Cvpr*, volume 1, page 5, 2017.

[132] G. Lin, C. Shen, A. van den Hengel, and I. Reid. Efficient piecewise training of deep structured models for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3194–3203, 2016.

[133] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and computation*, 108(2):212–261, 1994.

[134] F. Liu, C. Shen, and G. Lin. Deep convolutional neural fields for depth estimation from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5162–5170, 2015.

[135] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for se-

mantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[136] W. Lotter, G. Kreiman, and D. Cox. Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104*, 2016.

[137] D. G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[138] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[139] H. R. Madala and A. G. Ivakhnenko. *Inductive learning algorithms for complex systems modeling*, volume 368. cRc press Boca Raton, 1994.

[140] M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

[141] L. Mescheder, A. Geiger, and S. Nowozin. Which training methods for gans do actually converge? In *International Conference on Machine Learning*, pages 3478–3487, 2018.

[142] L. Mescheder, S. Nowozin, and A. Geiger. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. *arXiv preprint arXiv:1701.04722*, 2017.

[143] L. Mescheder, S. Nowozin, and A. Geiger. The numerics of gans. In *Advances in Neural Information Processing Systems*, pages 1825–1835, 2017.

[144] D. Mishkin and J. Matas. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015.

[145] T. M. Mitchell et al. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877, 1997.

[146] D. J. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In *IJCAI*, volume 89, pages 762–767, 1989.

[147] A. Montillo, J. Shotton, J. Winn, J. E. Iglesias, D. Metaxas, and A. Criminisi. Entangled decision forests and their application for semantic segmentation of ct images. In *Biennial International Conference on Information Processing in Medical Imaging*, pages 184–196. Springer, 2011.

[148] A. Montillo, J. Tu, J. Shotton, J. Winn, J. Iglesias, D. Metaxas, and A. Criminisi. Entangled forests and differentiable information gain maximization. *Decision Forests in Computer Vision and Medical Image Analysis*, 5:1, 2013.

[149] F. Moosmann, B. Triggs, and F. Jurie. Fast discriminative visual codebooks using randomized clustering forests. In *Twentieth Annual Conference on Neural Information Processing Systems (NIPS'06)*, pages 985–992. MIT Press, 2006.

[150] R. Mottaghi, X. Chen, X. Liu, N.-G. Cho, S.-W. Lee, S. Fidler, R. Urtasun, and A. Yuille. The role of context for object detection and semantic segmentation in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 891–898, 2014.

[151] V. N. Murthy, V. Singh, T. Chen, R. Manmatha, and D. Comaniciu. Deep decision network for multi-class image classification. In *Proceedings of*

the IEEE Conference on Computer Vision and Pattern Recognition, pages 2240–2248, 2016.

[152] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[153] M.-E. Nilsback and A. Zisserman. A visual vocabulary for flower classification. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1447–1454. IEEE, 2006.

[154] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1520–1528, 2015.

[155] A. Odena, J. Buckman, C. Olsson, T. B. Brown, C. Olah, C. Raffel, and I. Goodfellow. Is generator conditioning causally related to gan performance? *arXiv preprint arXiv:1802.08768*, 2018.

[156] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

[157] D. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal of artificial intelligence research*, 11:169–198, 1999.

[158] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast keypoint recognition using random ferns. *IEEE transactions on pattern analysis and machine intelligence*, 32(3):448–461, 2010.

[159] M. Ozuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. In *Computer Vision and Pattern Recognition, 2007. CVPR'07.*

IEEE Conference on, pages 1–8. Ieee, 2007.

[160] D. B. Parker. Learning logic. 1985.

[161] K. Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[162] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

[163] P. H. Pinheiro and R. Collobert. Recurrent convolutional neural networks for scene labeling. In *ICML*, pages 82–90, 2014.

[164] D. Poole, A. Mackworth, and R. Goebel. Computational intelligence: a logical approach. 1998.

[165] H. Qian and M. N. Wegman. L2-nonexpansive neural networks. *arXiv preprint arXiv:1802.07896*, 2018.

[166] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[167] J. R. Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[168] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[169] S. Ren, X. Cao, Y. Wei, and J. Sun. Global refinement of random forest. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 723–730, 2015.

[170] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2017.

[171] X. Ren, L. Bo, and D. Fox. Rgb-(d) scene labeling: Features and algorithms. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2759–2766. IEEE, 2012.

[172] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[173] D. L. Richmond, D. Kainmueller, M. Yang, E. W. Myers, and C. Rother. Mapping stacked decision forests to deep and sparse convolutional neural networks for semantic segmentation. *arXiv preprint arXiv:1507.07583*, 2015.

[174] M. Rosca, B. Lakshminarayanan, D. Warde-Farley, and S. Mohamed. Variational approaches for auto-encoding generative adversarial networks. *arXiv preprint arXiv:1706.04987*, 2017.

[175] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[176] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[177] S. Rota Bulo and P. Kontschieder. Neural decision forests for semantic image labelling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 81–88, 2014.

[178] K. Roth, A. Lucchi, S. Nowozin, and T. Hofmann. Stabilizing training of generative adversarial networks through regularization. *arXiv preprint arXiv:1705.09367*, 2017.

[179] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[180] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011.

[181] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[182] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[183] S. Ryan Fanello, C. Keskin, P. Kohli, S. Izadi, J. Shotton, A. Criminisi, U. Pattacini, and T. Paek. Filter forests for learning data-dependent convolutional kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1709–1716, 2014.

[184] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[185] T. Salimans, D. Kingma, and M. Welling. Markov chain monte carlo and variational inference: Bridging the gap. In *International Conference on Machine Learning*, pages 1218–1226, 2015.

[186] E. Santana and G. Hotz. Learning a driving simulator. *arXiv preprint arXiv:1608.01230*, 2016.

[187] A. Saxena, S. H. Chung, and A. Y. Ng. Learning depth from single monocular images. In *Advances in neural information processing systems*, pages 1161–1168, 2006.

[188] R. E. Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[189] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[190] F. Schroff, A. Criminisi, and A. Zisserman. Object class segmentation using random forests. In *BMVC*, pages 1–10, 2008.

[191] S. Schulter, P. Wohlhart, C. Leistner, A. Saffari, P. M. Roth, and H. Bischof. Alternating decision forests. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 508–515, 2013.

[192] H. Sedghi, V. Gupta, and P. M. Long. The singular values of convolutional layers. *arXiv preprint arXiv:1805.10408*, 2018.

[193] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *International Conference on Learning Representations (ICLR 2014)*, page 16. CBLS, 2013.

[194] I. K. Sethi. Entropy nets: from decision trees to neural networks. *Proceedings of the IEEE*, 78(10):1605–1613, 1990.

[195] C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[196] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813, 2014.

[197] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1297–1304. Ieee, 2011.

[198] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *Computer vision and pattern recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[199] J. Shotton, J. Winn, C. Rother, and A. Criminisi. Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *European conference on computer vision*, pages 1–15. Springer, 2006.

[200] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. In *European Conference on Computer Vision*, pages 746–760. Springer, 2012.

[201] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[202] M. Sonka, V. Hlavac, and R. Boyle. *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

[203] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[204] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385, 2015.

[205] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[206] C. Tao, L. Chen, R. Henao, J. Feng, and L. C. Duke. Chi-square generative adversarial network. In *International Conference on Machine Learning*, pages 4894–4903, 2018.

[207] J. B. Tenenbaum, V. De Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[208] J. Tighe and S. Lazebnik. Finding things: Image parsing with regions and per-exemplar detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3001–3008, 2013.

[209] M. A. Turk and A. P. Pentland. Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pages 586–591. IEEE, 1991.

[210] D. J. Utgoff, P. E.; Stracuzzi. "many-layered learning". *Neural Computation*, 8:2497–2529, 2002.

[211] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[212] V. Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[213] S. Viglione. 4 applications of pattern recognition technology. In *Mathematics in Science and Engineering*, volume 66, pages 115–162. Elsevier, 1970.

[214] M. Villamizar, F. Moreno-Noguer, J. Andrade-Cetto, and A. Sanfeliu. Efficient rotation invariant object detection using boosted random ferns. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1038–1045. IEEE, 2010.

[215] M. Villamizar, F. Moreno-Noguer, J. Andrade-Cetto, and A. Sanfeliu. Shared random ferns for efficient detection of multiple categories. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 388–391. IEEE, 2010.

[216] C. Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

[217] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.

[218] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[219] J. Walker, C. Doersch, A. Gupta, and M. Hebert. An uncertain future: Forecasting from static images using variational autoencoders. In *European Conference on Computer Vision*, pages 835–851. Springer, 2016.

[220] S. Wang, S. Fidler, and R. Urtasun. Holistic 3d scene understanding from a single geo-tagged image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3964–3972, 2015.

[221] P. J. Werbos. Applications of advances in nonlinear sensitivity analysis. In *System modeling and optimization*, pages 762–770. Springer, 1982.

[222] M. Witczak, J. Korbicz, M. Mrugalski, and R. J. Patton. A gmdh neural network-based approach to robust fault diagnosis: Application to the damadics benchmark problem. *Control Engineering Practice*, 14(6):671–683, 2006.

[223] Y. Yang, S. Hallman, D. Ramanan, and C. Fowlkes. Layered object detection for multi-class segmentation. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 3113–3120. IEEE, 2010.

[224] J. Yao, S. Fidler, and R. Urtasun. Describing the scene as a whole: Joint object detection, scene classification and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 702–709. IEEE, 2012.

[225] Y. Yoo, S. Yun, H. J. Chang, Y. Demiris, and J. Y. Choi. Variational autoencoded regression: high dimensional regression of visual data on complex manifold. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3674–3683, 2017.

[226] Y.-x. Yuan. Step-sizes for the gradient method. *AMS IP Studies in Advanced Mathematics*, 42(2):785, 2008.

[227] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[228] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *arXiv preprint*, 2017.

[229] N. Zhang, J. Donahue, R. Girshick, and T. Darrell. Part-based r-cnns for fine-grained category detection. In *European conference on computer vision*, pages 834–849. Springer, 2014.

[230] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2881–2890, 2017.

[231] J. Zhao, M. Mathieu, and Y. LeCun. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*, 2016.

[232] Y. Zuo, G. Avraham, and T. Drummond. Generative adversarial forests for better conditioned adversarial learning. *arXiv preprint arXiv:1805.05185*, 2018.

[233] Y. Zuo, G. Avraham, and T. Drummond. Traversing latent space using decision ferns. *arXiv preprint arXiv:1812.02636*, 2018.

[234] Y. Zuo and T. Drummond. Fast residual forests: Rapid ensemble learning for semantic segmentation. In *Conference on Robot Learning*, pages 27–36,

2017.