Supplementary Materials: Can Deep Networks be Highly Performant, Efficient and Robust simultaneously?

Madan Ravi Ganesh[1, *2]
madan.raviganesh@us.bosch.com
Salimeh Yasaei Sekeh[3]
salimeh.yasaei@maine.edu
Jason J. Corso[2]
jjcorso@umich.edu

[1] Bosch Center for Artificial Intelligence
2555 Smallman Drive,
Pittsburgh, Pennsylvania, USA

[2] University of Michigan
500 S State Street,
Ann Arbor, Michigan, USA

[3] University of Maine
168 College Avenue,
Orono, Maine, USA

A CAPER: Modification For ILSVRC2012

CAPER scales across the size and depth of a DNN as well as the number of samples in a dataset. To efficiently execute CAPER on ILSVRC2012, which has over 1 million images, we re-purposed the algorithm to function in two phases instead of one. In the first phase, we compute $D(.)$ from Eq. 6 (in the main paper) across samples of each label and summarize them using their mean value to ascertain the difference heuristic over labels. In the second phase, we refine our search space to samples across the 10 labels with the highest difference in values and re-capture the heuristic across the samples from only these labels. Doing so allows us to avoid comparing statistics across a million samples, instead we simplify the comparison to samples across 10 labels (approximately 13000).

B CAPER: Multi-layer Extension

In this section, we extend CAPER to work in a multi-layer setting. We provide details on the extension and relevant results below.

B.1 Sensitivity Constraint

When collecting features across a layer we implicitly assume uniform importance for all filters. However, from DNN pruning literature [1, 7] we know that there are a number of filters which provide redundant information and reducing their contribution does not hurt the performance of DNNs. Following this line of thought, we adopt the notion of sensitivity [2]

* indicates work done when affiliated with

to capture features from a subset of filters that provide important information. While there are many different ways to utilize sensitivity, in this work we threshold the value of sensitivity to obtain a subset of the filters ($\tilde{O}^{(l)}$) from which we derive our features. Doing so allows us to leverage the learned structure of the weight matrices in identifying sensitive filters while also reducing the overall memory consumed to store features.

B.2 Computing the Binary Mask

While $\Delta\hat{f}$ captures the distance between features from a specific layer, we expand the formulation of CAPER to include sensitivity when aggregating distances across multiple layers of the DNN. To do so, we include $\xi_i^{(l)}$, the instability of a sample measured as the average $\Delta\hat{f}$ across filters in a given layer.

$$\xi_i^{(l)} = \frac{\sum_{q=1}^{\tilde{O}^{(l)}} \Delta\hat{f}(i,q)}{\tilde{O}^{(l)}}, \;\; i \in \{1,\ldots,N\}. \tag{1}$$

By combining the contributions of $\xi_i^{(1)}, \xi_i^{(2)}, \ldots, \xi_i^{(L)}$ across multiple layers we obtain the overall instability of a sample, ξ_i, given as,

$$\xi_i = \xi_i^{(1)}\alpha^{(1)} + \ldots + \xi_i^{(L)}\alpha^{(L)}, \tag{2}$$

where $\alpha()$ denotes a window function that provides scalar multipliers used to combined the instability values obtained from different layers.

To identify the optimal values of α we would have to solve the system of equations shown below,

$$\begin{bmatrix} \xi_1^{(1)} & \xi_1^{(2)} & \cdots & \xi_1^{(L)} \\ \vdots & \vdots & \cdots & \vdots \\ \xi_N^{(1)} & \xi_N^{(2)} & \cdots & \xi_N^{(L)} \end{bmatrix} \in \mathbb{R}^{(N\times L)} \begin{bmatrix} \alpha(1) \\ \vdots \\ \alpha(L) \end{bmatrix} \in \mathbb{R}^{(L)}_{\geq 0}, \tag{3}$$

where the final accuracy is the metric over which we need to optimize. Given the practical constraints in solving this system of equations, where the LHS is ill-defined and the size of the system matrix forces any operation on it to be expensive, we explore a restricted set of functions, including $\mathbb{1}_{1:\frac{L}{2}}$, $\mathbb{1}_{\frac{L}{2}:L}$, a gaussian distribution and finally $\mathbb{1}_L$, to find the best performing α. Once we set α, we can evaluate ξ_i. Using these values, we compute m as:

$$m_i = \begin{cases} 0 & \text{if } \xi_i \text{ is in the top } \gamma \text{ values of } \xi \\ 1 & \text{o.w} \;. \end{cases} \tag{4}$$

By controlling γ, we use m_i to reduce the amount of the training data held in memory as well as the overall FLOPs required during training. Once m is applied, the DNN is then trained with the remaining subset of data from epochs τ to E.

B.3 Results

Across all the results presented in Tables 1 and 2 in the main paper, we use $\alpha = \mathbb{1}_L$ which results in the collection of features from the last convolutional layer. In this section, we compare and contrast four different window functions to identify the impact of comparing

Window	VGG16	MobileNet	DenseNet	ResNet50
Baseline	94.04	93.50	95.13	95.63
$\alpha = \mathbb{1}_L$	94.47 ($\gamma = 125$)	93.62 ($\gamma = 125$)	95.16 ($\gamma = 50$)	95.75 ($\gamma = 25$)
$\alpha = \mathbb{1}_{1:\frac{L}{2}}$	**94.49** ($\gamma = 300$)	**93.66** ($\gamma = 150$)	**95.28** ($\gamma = 50$)	**95.78** ($\gamma = 50$)
$\alpha = \mathbb{1}_{\frac{L}{2}:L}$	94.41 ($\gamma = 300$)	93.59 ($\gamma = 150$)	95.22 ($\gamma = 50$)	95.72 ($\gamma = 50$)
$\alpha = \mathcal{N}(0,1)$	94.43 ($\gamma = 250$)	93.61 ($\gamma = 125$)	95.16 ($\gamma = 300$)	95.74 ($\gamma = 100$)

Table 1: Assessing the susceptibility of samples to noise using multiple layers boosts γ as well as Accuracy (%). $\alpha = \mathbb{1}_{1:\frac{L}{2}}$ provides the best performance. Optimal result for each CIFAR10-DNN-α combination, up to $\gamma = 350$, is provided in the table.

features across multiple layers of a DNN and its potential benefits. For the sake of consistency, we restrict the set of possible γ values to a maximum of 350. Based on Table 1, there are two main observations. First, the use of additional layers in assessing the susceptibility of samples to noise often allows for an increase in γ when compared to the case of $\mathbb{1}_L$, with minor trade-offs in performance. Second, in conjunction with the first observation, $\alpha = \mathbb{1}_{1:\frac{L}{2}}$ shows the best Accuracy(%) across our restricted set of window functions. These results highlight the regularization effect our method imposes on the DNN, regardless of the location at which we ascertain the distance between features. Further, by assessing distances across layers other than the final one in the DNN, we can reduce the relationship between specific task-oriented information and how we assess noisy samples, allowing CAPER to be more extensible to alternative tasks.

B.4 Hyper-parameters

In studying the effects of a variety of window functions, we observe an improvement in overall γ as well as the final testing Accuracy (%). We list the number of filters, post sensitivity, and the ε used to compute the final performance for each DNN.

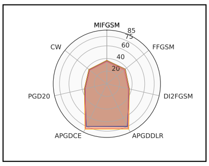
- For VGG16, we use a subset of 17 filters and $\varepsilon = 0.7$.
- For MobileNet, we use a subset of 16 filters and $\varepsilon = 0.5$.
- For DenseNet, we use a subset of 12 filters and $\varepsilon = 0.0$.
- For ResNet50, we use a subset of 12 filters and $\varepsilon = 0.3$. We list the optimal results from $\tau = 100$ for $\alpha = \mathbb{1}_{\frac{L}{2}:L}$ and $\alpha = \mathcal{N}(0,1)$.

C Adversarial Robustness

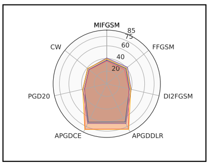
In the following section, we discuss adversarial robustness to attacks generated from multiple DNN architectures.

C.1 Adversarial Sources $\neq$ Target

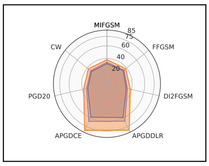
We define in-Transferability as the robustness of DNNs to attacks designed on a variety of DNN backbones. To measure in-Transferability, we use the mean and standard deviation of Robust Accuracy (%) when a selected model is attacked using adversaries generated from all four of the DNN architectures used in our experiments. We specifically demand that standard



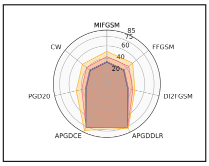
(a) VGG16



(b) MobileNet

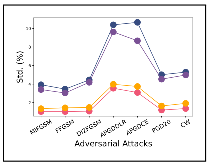


(c) DenseNet

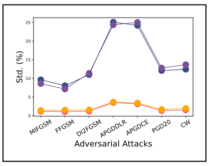


(d) ResNet50

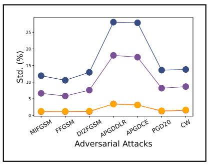
Figure 1: CAPER-based training boosts the mean Robustness Accuracy(%) across multiple sources of adversaries. In our experiments, we use all four possible DNN architectures to generate attacks. γ values are 125/12, 250/125, 25/5 and 12/25 for CAPER+[6] and CAPER +[5] respectively across VGG16, MobileNet, DenseNet and ResNet50.



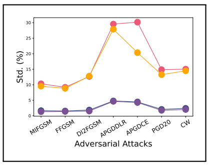
(a) VGG16



(b) MobileNet



(c) DenseNet



(d) ResNet50

Figure 2: CAPER-based adversarial training algorithms consistently have some of the lowest deviation in performance, thus ensuring in-Transferability.

deviation in performance is minimized, in addition to high average performance, since a high deviation is indicative of robustness being dependent on the type of DNN backbone used to generate adversaries.

In Figs. 1 and 2, we highlight the mean and standard deviation of Robust Accuracy(%) when we take into account adversaries generated across all 4 possible DNN architectures. Across almost all DNN-adversarial attack combinations, the average Robust Accuracy(%) for CAPER-based approaches improves on the original adversarial training approach. The only exception is on VGG16 for APGDCE and APGDDLR attacks, which fall within 1 standard deviation of the original approach's performance. From the standard deviation point of view, Shafahi *et al.* [5]-based training had lower values when compared to Wong *et al.* [6]-based training across all DNNs except ResNet50, both with and without CAPER. Overall, our results closely resemble the standard deviation of existing efficient adversarial training approaches while improving the in-Transferability of DNNs to a variety of adversarial attacks.

A broader takeaway from our results is the distinctly visible patterns of performance between [6] and [5], with some degree of architectural specificity (Residual vs. the rest). A further study into their relationship could help highlight factors essential to designing adversarial training approaches. Overall, the common trend of improved mean Robust Accuracy(%) with low standard deviations highlights our CAPER-based adversarial training as a definite way to ensure in-Transferability across a number of DNN architectures.

D Discussion

ε Selection Throughout our experiments, we found ε to be closely related to the combination of Dataset-DNN and invariant to γ, τ or the choice of training algorithm. Hence, we ran experiments across a standard combination of ε values and provide the best settings in Section E.

Algorithm	Accuracy(%)			
	VGG16	MobileNet	DenseNet	ResNet50
Best from Table 1 in main paper	**94.44**	**93.62**	**95.20**	**95.83**
Wong *et al.* [6]	76.58	80.66	74.10	79.19
CAPER+[6]	79.75	83.29	75.96	79.28
Shafahi *et al.* [5]	80.86	82.17	83.69	91.64
CAPER+[5]	82.62	82.97	84.73	92.55

Table 2: There is a still a significant gap between the improved Accuracy(%) achieved by adversarial training and results from Table 1 in main paper, which suggests there is still room for improvement in this domain. Results for [5]-based experiments are across 1 trial.

τ Selection The value of τ is directly tied to the expected decrease in training FLOPs. Typically, we choose a value of 50, when training up to and beyond 300 epochs, and 35 when training up to 100 epochs. Only in a very limited set of circumstances, where performances did not match our expected improvement, did we investigate the impact of varying τ to 75 and 100 epochs. Otherwise, all our experiments used predetermined values for τ.

Noise injected We intentionally avoid the use of any adversarial perturbations and use simple gaussian noise since our focus is not only to highlight samples susceptible to a particular distribution of adversaries but those that generally detract from performance in regular training as well. For this purpose we keep the perturbation generic. Apart from the type of noise used to highlight features, we needed to ensure that there was a strong enough impact of the noise on the features. A very small magnitude would force behaviors similar to fluctuations due to data augmentations while too large a magnitude would force irregular behavior from all samples, thus we settled intuitively on 0.5.

Performance vs. Adversarial Robustness The trade-off between performance and adversarial robustness is a commonly know and accepted fact. A number of works like [4] and others try to resolve this by aiming to improve on both fronts simultaneously. From our results on adversarial robustness we observe that there is a sharp drop in Accuracy(%) when we apply any form of adversarial training, regardless of the underlying DNN architecture. Only CAPER+[5] on ResNet50-CIFAR10 comes within 4% of the highest performance we achieve in curriculum-based comparison (Table 2). While there are differences in the settings and hyper-parameters suggested by the original authors of the adversarial training works, we find that there is still room for improvement when it comes to bridging the gap between improving Accuracy(%) and Robust Accuracy(%) simultaneously.

DenseNet Performance Over the course of our experiments on adversarial robustness and curriculum-based comparisons DenseNet has offered the smallest magnitude of robustness and improvement in performance. While there could be a number of contributing factors, we hypothesize that DenseNet's foundational building block of continuously retaining and combining features from previous blocks is one of the main reasons why standard mini-batch, DIHCL and even CAPER-based training does not provide a strong improvement in adversarial robustness. A more comprehensive combination of the features across the entire DNN might help overcome the current issue and improve the overall adversarial robustness.

Analysis of removed samples On analyzing the removed data points we found a number of common patterns, (1) the distribution of data removed across each label of a dataset is non-uniform, (2) when we observe the distribution of data removed from different labels across mutiple DNN backbones there are common labels where a high percentage of samples are removed relative to others, and (3) there are characteristic peaks in samples removed across other labels that are specific to the DNN considered. When looking at the cross-section of samples removed, (1) across multiple trials on the same Dataset-DNN combination there is about 33% overlap, which highlights the impact of the stochasticity during training, and (2) across multiple DNN architectures the overlap could be between $27-10\%$. This indicates that there is definitely a core set of samples that can be removed across the entire dataset, with the remainder being more DNN-specific. In general, we observe that our method avoids targeting a specific class and instead naturally selects across the entire dataset. To avoid bias, explicitly highlighting samples as being part of the core dataset and using cut-mix/mixup-like augmentations could help soften their impact on the learning process and assimilate them. Interestingly, when visually inspecting a certain subset of classes there were no clear demarcations or differences in the images that were discarded. This leads us to believe that the samples removed were identified based on the perception/understanding of the DNN, slightly different to our understanding of the images.

Potential Negative Impacts Since we reduce the total amount of training data provided to the model, we risk losing some of the representational depth and complexity in the learned features. This is especially important when considering the impact of weaker pretraining on downstream tasks. In addition, our core idea revolves around removing datapoints that have a high proclivity to being ambiguous. One possible implication of the removal of such datapoints could be a reduction in the fairness of the overall model since such datapoints could be part of an underrepresented set of data. From an adversarial robustness perspective, when using the l_2 metric distance as a sensitivity measure, we risk exposing our feature embeddings to alternative forms of adversarial attacks.

E Experimental Setup

E.1 Abbreviations

Throughout the supplementary materials, we use shorthand notations to simplify the discussion of certain DNN architecture or hyper-parameter names. We outline their full meaning below,

- Sched. : Learning rate step schedule
- Opt. : Optimizer
- Decay : Weight decay
- Mult. : Multiplier
- Mtm. : Momentum
- Bandit Alg. : Bandit Algorithm
- Loss Fb. : Loss feedback

	VGG16	MobileNet
Epochs	300 / 200	350 / 200
Batch	128 / 128	128 / 128
Lr	0.1 / 0.1	0.1 / 0.1
Sched.	90,180,260 / 60,120,160	150,250 /90,180,260
Opt.	SGD / SGD	SGD / SGD
Decay	0.0005 / 0.0005	0.00004 / 0.0001
Mult.	0.2 / 0.2	0.1 / 0.2
Mtm.	True / True	False / True

Table 3: Training setups for mini-batch SGD (Baseline) on CIFAR-10 / CIFAR-100 respectively. Here, MobileNet uses cosine LR scheduling for CIFAR-100

	DenseNet	ResNet50
Epochs	300 / 300	300 / 300
Batch	64 / 64	128 / 128
Lr	0.1 / 0.1	0.1 / 0.1
Sched.	150,225 / 150,225	90,180,260 /90,180,260
Opt.	SGD / SGD	SGD / SGD
Decay	0.0001 / 0.0001	0.0002 / 0.0002
Mult.	0.1 / 0.1	0.1 / 0.1
Mtm.	False / False	True / True

Table 4: Training setups for mini-batch SGD (Baseline) on CIFAR-10 / CIFAR-100 respectively.

E.2 Curriculum Comparison

Tables 3, 4 and 5, describe the hyper-parameters used for our baseline (SGD) models while Tables 6 and 7 describe the hyper-parameters used for the DIHCL algorithm [8]. For the ILSVRC2012 experiments, we use Epoch=100, Batch=64, Lr=0.1, Sched. = 30,60,90, Opt.=SGD, Decay=0.00003, Mult.=0.1 and Mtm= True, with $\tau = 15$. Code for the DIHCL algorithm was provided from `https://github.com/tianyizhou/DIHCL`. For CAPER, we re-use the hyper-parameters in Tables 3, 4 and 5 while experimenting on values for γ and ε, after setting $\tau = 50$. Only for DenseNet on and ResNet50 on CIFAR-10 we set $\tau = 75, 100$ respectively. The final values of ε and γ for the results in Tables 1 and 2 (in the main paper) are,

- For the CIFAR-10 experiments, $\gamma = 2500, 125, 100, 1000$ and $\varepsilon = 0.7, 0.1, 0.0, 0.3$ for VGG16, MobileNet, DenseNet and ResNet50 respectively.
- For the CIFAR-100 experiments, $\gamma = 1250, 10000, 1250, 1250$ and $\varepsilon = 0.5, 0.7, 0.1, 0.2$ for VGG16, MobileNet, DenseNet and ResNet50 respectively.
- For the miniImagenet experiments, $\gamma = 2500, 5000, 5000, 2500$ and $\varepsilon = 0.1, 0.7, 0.3, 0.1$ for VGG16, MobileNet, DenseNet and ResNet50 respectively.
- Finally, for the ILSVRC2012 experiment, $\gamma = 11700$ and $\varepsilon = 0.3$.

	VGG16	MobileNet	DenseNet	ResNet50
Epochs	300	200	300	300
Batch	64	128	64	128
Lr	0.01	0.1	0.1	0.1
Sched.	90,180,260	90,180,260	150,225	90,180,260
Opt.	SGD	SGD	SGD	SGD
Decay	0.0005	0.0001	0.0001	0.0002
Mult.	0.2	0.2	0.1	0.1
Mtm.	True	True	False	True

Table 5: Training setups for mini-batch SGD (Baseline) miniImagenet respectively

	VGG16	MobileNet	DenseNet	ResNet50
Epochs	300 / 300	350 / 300	300 / 300	300 / 300
Bandit Alg.	EXP3 / EXP3	EXP3 / EXP3	EXP3 / EXP3	EXP3 / EXP3
Mean Teacher	True / True	True / True	True / True	True / True
Loss Fb.	True / True	True / True	True / True	True / True
Batch Size	128 / 128	128 / 128	128 / 128	128 / 128

Table 6: Training setups for DIHCL on CIFAR-10 / CIFAR-100 respectively. MobileNet uses a schedule of [0 5 10 15 20 30 40 60 90 140 210 300 350]

	VGG16	MobileNet	DenseNet	ResNet50
Epochs	300	300	300	300
Bandit Alg.	TS	TS	TS	TS
Mean Teacher	True	True	True	True
Loss Fb.	False	False	False	False
Batch Size	128	128	64	128

Table 7: Training setups for DIHCL on miniImagenet.

E.3 Adversarial Robustness

The adversarial training algorithms we used were cloned from `https://github.com/locuslab/fast_adversarial`. Most of the adversarial attacks were cloned from `https://github.com/Harry24k/adversarial-attacks-pytorch`

while PGD20 and CW loss-based attacks were ported from `https://github.com/zjfheart/Friendly-Adversarial-Training`.

E.3.1 Adversarial Attacks

In general, we use the default settings provided for all the adversarial attacks throughout our experiments. We highlight some of the specifications (variable names) for each attack below,

- MIFGSM: $\varepsilon = 8/255.$, $\alpha = 2/255.$, decay=1.0, iterations=5.
- FFGSM: $\varepsilon = 8/255.$, $\alpha = 10/255.$.
- DI2FGSM: $\varepsilon = 8/255.$, $\alpha = 2/255.$, decay=0.0, steps=20, resize_rate=0.9, diversity_prob=0.5, random_state=False.

	PreActResNet18
Epochs	200
Batch	128
LR schedule	piecewise
LR max	0.1
LR one drop	0.01
LR one drop epoch	100
Attack	PGD
Epsilon	8
Attack iters	10
Restarts	1
PGD-alpha	2

Table 8: Training setup for [4] on CIFAR-10

	ResNet18
Epochs	100
Batch	128
Decay	0.0002
LR	0.1
Mtm.	0.9
Epsilon	0.031
Steps	10
Step size	0.007

Table 9: Training setup for [1] on CIFAR-100

- APGD: $\varepsilon = 8/255.$, steps=100.
- CWLoss: steps=30,$\varepsilon = 0.031$, step_size=$0.031/4$, category='Madry', rand _init= True.
- PGD20: steps=20, $\varepsilon = 0.031$, step_size=$0.031/4$, category='Madry', rand _init = True.

E.3.2 Standard Adversarial Training

In this section, we list the hyper-parameters used to train models based on Rice *et al.* [4] and Cui *et al.* [1] in Tables 8 and 9. For CAPER, we re-use the hyper-parameters from the original algorithms alongside our selection of γ and ε while setting $\tau = 35$. Specifically,

- For the CAPER+[4], $\gamma = 245$ and $\varepsilon = 0.1$.
- For the CAPER+[1], $\gamma = 400$ and $\varepsilon = 0.1$.

E.3.3 Efficient Adversarial Training

We list the hyper-parameters used to train models based on Wong *et al.* [6] and Shafahi *et al.* [5] in Tables 10 and 11. For CAPER, we re-use the hyper-parameters from the original

	VGG16	MobileNet	DenseNet	ResNet50
Epochs	300	350	300	300
Batch	128	128	64	128
LR min	0.0	0.0	0.0	0.0
LR max	0.1	0.1	0.1	0.1
Sched.	Cyclic	Cyclic	Cyclic	Cyclic
Opt.r	SGD	SGD	SGD	SGD
Decay	0.0005	0.00004	0.0001	0.0002
epsilon	8	8	8	8
alpha	10	10	10	10
delta-init	Random	Random	Random	Random
Mtm.	True	True	True	True

Table 10: Training setup for [6] on CIFAR-10

	VGG16	MobileNet	DenseNet	ResNet50
Epochs	300	350	300	300
Batch	128	128	64	128
LR min	0.0	0.0	0.0	0.0
LR max	0.1	0.1	0.1	0.1
Sched.	Cyclic	Cyclic	Cyclic	Cyclic
Opt.r	SGD	SGD	SGD	SGD
Decay	0.0005	0.00004	0.0001	0.0002
epsilon	8	8	8	8
Mtm.	True	True	True	True

Table 11: Training setup for [5] on CIFAR-10

algorithms alongside our selection of γ and ε while setting $\tau = 50$. Specifically,

- For the CAPER+[6], $\gamma = 125, 250, 25, 12$ and $\varepsilon = 0.1, 0.5, 0.2, 0.3$ for VGG16, MobileNet, DenseNet and ResNet50 respectively.
- For the CAPER+[5], $\gamma = 450, 125, 5, 25$ and $\varepsilon = 0.5, 0.3, 0.1, 0.7$ for VGG16, MobileNet, DenseNet and ResNet50 respectively.

References

[1] Jiequan Cui, Shu Liu, Liwei Wang, and Jiaya Jia. Learnable boundary guided adversarial training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15721–15730, 2021.

[2] Madan Ravi Ganesh, Dawsin Blanchard, Jason J Corso, and Salimeh Yasaei Sekeh. Slimming neural networks using adaptive connectivity scores. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[3] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *5th International Conference on Learning Representations,*

ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net, 2017. URL https://openreview.net/forum?id=rJqFGTslg.

[4] Leslie Rice, Eric Wong, and Zico Kolter. Overfitting in adversarially robust deep learning. In *International Conference on Machine Learning*, pages 8093–8104. PMLR, 2020.

[5] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *Advances in Neural Information Processing Systems*, 32:3358–3369, 2019.

[6] Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=BJx040EFvH.

[7] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

[8] Tianyi Zhou, Shengjie Wang, and Jeff A Bilmes. Curriculum learning by dynamic instance hardness. *Advances in Neural Information Processing Systems*, 33, 2020.