

Imperial College London
Faculty of Engineering
Department of Computing

Understanding and Mitigating Universal Adversarial Perturbations for Computer Vision Neural Networks

Kenneth T. Co

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of the Imperial College London
October 2022

Statement of Originality

I certify that the work in this thesis is my own and that all else has been properly acknowledged.

Copyright Declaration

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-Non Commercial 4.0 International Licence (CC BY-NC). Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose. When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes. Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Abstract

Deep neural networks (DNNs) have become the algorithm of choice for many computer vision applications. They are able to achieve human level performance in many computer vision tasks, and enable the automation and large-scale deployment of applications such as object tracking, autonomous vehicles, and medical imaging. However, DNNs expose software applications to systemic vulnerabilities in the form of Universal Adversarial Perturbations (UAPs): input perturbation attacks that can cause DNNs to make classification errors on large sets of inputs.

Our aim is to improve the robustness of computer vision DNNs to UAPs without sacrificing the models' predictive performance. To this end, we increase our understanding of these vulnerabilities by investigating the visual structures and patterns commonly appearing in UAPs. We demonstrate the efficacy and pervasiveness of UAPs by showing how Procedural Noise patterns can be used to generate efficient zero-knowledge attacks for different computer vision models and tasks at minimal cost to the attacker. We then evaluate the UAP robustness of various shape and texture-biased models, and found that applying them in ensembles provides marginal improvement to robustness.

To mitigate UAP attacks, we develop two novel approaches. First, we propose the Jacobian of DNNs to measure the sensitivity of computer vision DNNs. We derive theoretical bounds and provide empirical evidence that shows how a combination of Jacobian regularisation and ensemble methods allow for increased model robustness against UAPs without degrading the predictive performance of computer vision DNNs. Our results evince a robustness-accuracy trade-off against UAPs that is better than those of models trained in conventional ways. Finally, we design a detection method that analyses the hidden layer activation values to identify a variety of UAP attacks in real-time with low-latency. We show that our work outperforms existing defences under realistic time and computation constraints.

Acknowledgements

I am grateful to my supervisor Professor Emil Lupu for providing guidance, giving me confidence in my work, and for all the support both inside and outside college. I am also grateful to my secondary supervisor Professor Ben Glocker for providing many valuable insights and suggestions for my research topic.

I would like to thank the Resilient Information Systems Security Group (RISS), and in particular Aaron, Luca, and Luis for the wonderful discussions and the invaluable lessons learned together. A special thanks also to my industry sponsor DataSpartan, and in particular David and Leslie for their continued support and exchange of ideas.

Lastly, I wish to thank my family and friends: my parents for their support without which none of this would be possible, and my friends for filling me with joy and providing me with the motivation to face the challenges of this work.

Publications

The central content of this thesis appears in the following publications and peer-reviewed work:

- “Procedural Noise Adversarial Examples for Black-Box Attacks on Deep Convolutional Networks”. K. T. Co, L. Muñoz-González, S. De Maupéou, E. C. Lupu. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, ACM, 2019.
- “Sensitivity of Deep Convolutional Networks to Gabor Noise”. K. T. Co, L. Muñoz-González, E. C. Lupu. *ICML 2019 Workshop: On Identifying and Understanding Deep Learning Phenomena*, 2019.
- “Universal Adversarial Robustness of Texture and Shape-Biased Models”. K. T. Co, L. Muñoz-González, L. Kanthan, B. Glocker, E. C. Lupu. In *Proceedings of the 2021 International Conference on Image Processing (ICIP)*, IEEE, 2021.
- “Jacobian Regularization for Mitigating Universal Adversarial Perturbations”. K. T. Co, D. Martinez, E. C. Lupu. In *Proceedings of the 2021 International Conference on Artificial Neural Networks (ICANN)*, Springer, 2021.
- “Improving Robustness-Accuracy Trade-offs with Jacobian Ensembles”. K. T. Co, D. Martinez, Z. Y. Hau, E. C. Lupu. In *Proceedings of the 2022 International Conference on Artificial Neural Networks (ICANN)*, Springer, 2022.
- “Real-time Detection of Practical Universal Adversarial Perturbations”. K. T. Co, L. Muñoz-González, L. Kanthan, E. C. Lupu. Under review for the *2023 International Conference on Applied Cryptography and Network Security (ACNS)*.

Contents

Abstract	5
Acknowledgements	6
1 Introduction	21
1.1 Goal and Scope	22
1.2 Summary of Contributions	24
1.3 Thesis Outline	26
2 Background	27
2.1 Preliminaries	27
2.1.1 Neural Networks	27
2.1.2 Adversarial Examples	28
2.1.3 Universal Adversarial Perturbations	29
2.2 Threat Model & Metrics	29
2.2.1 Adversarial Goals	29
2.2.2 Adversarial Capabilities	31
2.2.3 Adversarial Knowledge	34

2.3	Universal Attack Algorithms	36
2.3.1	Fast Gradient Sign	36
2.3.2	Iterative DeepFool	37
2.3.3	Stochastic Gradient Descent	38
2.3.4	Generalisable Data-free UAP	38
2.3.5	Singular Vector Attack	40
2.3.6	Fourier Basis Functions	41
2.3.7	Adversarial Patches	42
2.4	Defending Against Universal Attacks	43
2.4.1	Categorising Defences	44
2.4.2	Adversarial Training	45
2.4.3	Perturbation Rectifying Network	46
2.4.4	Selective Feature Regeneration	47
2.4.5	SentiNet: Detecting Localised Universal Attacks	48
2.5	Summary	49
3	Procedural Noise Attacks	50
3.1	Background: Procedural Noise	52
3.1.1	Types of Procedural Noise Functions	52
3.1.2	Colour Maps	54
3.1.3	Procedural Noise Attack	55
3.2	Vulnerability to Procedural Noise	57
3.2.1	Experimental Setup	57

3.2.2	Effectiveness of Procedural Noise	60
3.2.3	Model Sensitivity on Inputs	63
3.2.4	Discussion	66
3.3	Background: Bayesian Optimization	70
3.3.1	Gaussian Processes	71
3.3.2	Acquisition Functions	72
3.4	Efficient Zero-knowledge Attack	73
3.4.1	Universal Zero-knowledge Attack	73
3.4.2	Input-dependent Zero-knowledge Attack	75
3.5	Beyond Image Classification	78
3.5.1	Background: Object Detection	78
3.5.2	Experimental Setup	80
3.5.3	Results & Discussion	81
3.5.4	Beyond Images	83
3.6	Preliminary Defence	84
3.7	Conclusion	86
4	Robustness of Texture and Shape-biased Models	88
4.1	Shape-biased Training	89
4.2	Robustness of Shape-biased Models	91
4.2.1	Experimental Setup	91
4.2.2	Transferability Across Training Datasets	92
4.2.3	Transferability Across Architectures	94

4.3	Dominant Label Analysis	94
4.3.1	Effectiveness of Targeted Universal Attacks	95
4.3.2	Dominant Labels of Untargeted UAPs	97
4.3.3	Effectiveness of Targeted Universal Attacks	99
4.3.4	ImageNet vs. Stylized-ImageNet Training	100
4.4	Mitigating Attacks with Ensembles	101
4.5	Conclusion	103
5	Mitigating UAPs Through Jacobian Regularisation	104
5.1	Background & Notations	105
5.1.1	Preliminary Definitions	105
5.1.2	Jacobian Regularisation	106
5.1.3	UAPs for Jacobians	108
5.2	Bounds for the Stacked Jacobian	109
5.2.1	Upper Bound for the Frobenius Norm	109
5.2.2	Measuring Alignment of Jacobians	112
5.3	Robustness of Jacobian Regularised Models	113
5.3.1	Experimental Setup	113
5.3.2	Jacobian Regularisation Mitigates UAPs	113
5.3.3	Jacobian Alignment of Input Pairs	116
5.4	Jacobian Ensembles	117
5.4.1	Model Ensembles	117
5.4.2	Bounds for UAP Robustness on Model Ensembles	118

5.5	Improved Trade-offs with Jacobian Ensembles	122
5.5.1	Experimental Setup	122
5.5.2	Improvements with Jacobian Ensembles	123
5.5.3	Accuracy-Robustness Trade-off	125
5.6	Conclusions	127
6	HyperNeuron: Detecting UAPs via Activation Values	129
6.1	HyperNeuron Detection Algorithm	130
6.1.1	Motivation	131
6.1.2	HyperNeuron Algorithm	133
6.1.3	Measuring Detector Performance	136
6.2	Small-scale Images	137
6.2.1	Experimental Setup	137
6.2.2	Layer Calibration	139
6.2.3	Detection Results	139
6.3	Large-scale Images	141
6.3.1	Experimental Setup & Calibration	141
6.3.2	Adversarial Noise Evaluation	143
6.3.3	Targeted UAP Evaluation	144
6.3.4	Adversarial Patch Evaluation	145
6.4	Detection in Practice	147
6.4.1	Performance Metrics in Practice	148
6.4.2	Practical Evaluation of HyperNeuron	149

6.5	Comparison with Existing Defences	151
6.5.1	Related Work	151
6.5.2	Comparison & Results	152
6.6	Defence-aware Adaptive Attack	155
6.7	Application: Perceptual Ad-blocking	156
6.8	Limitations & Trade-offs of HyperNeuron	158
6.9	Discussion	159
6.10	Conclusion	160
7	Conclusion	162
7.1	Summary of Thesis Achievements	162
7.2	Future Work	165
	Bibliography	166

List of Figures

2.1	UAPs for various models generated using iDeepFool [1].	37
2.2	UAPs for various models generated with GD-UAP [2].	39
2.3	UAPs generated for VGG models using the singular vector method [3] on VGG-19.	41
2.4	Visualisations of (left) Fourier basis in 8×8 space and (right) example of generated Fourier basis attack [4].	42
3.1	Adversarial example generated with a procedural noise function. From left to right: original image, adversarial example, and procedural noise (magnified for visibility). Below the first two images are the classifier's top 5 output probability labels.	51
3.2	Samples of lattice gradient noise with colour maps: (top) greyscale and (middle) greyscale sine function.	55
3.3	Procedural noise patterns with (top) Gabor noise and (bottom) Perlin noise, both with decreasing frequency from left to right.	57
3.4	UAPs generated for VGG-19 using the Singular Vector Attack [3].	57
3.5	Universal Evasion Rate	60
3.6	Average Sensitivity	60
3.7	Histogram of (a) UER over all perturbations and (b) average sensitivity over all inputs across all models. For example, VGG-19 in (a) shows that about 12% of the Gabor noise perturbations have a UER of approximately 0.8.	60

3.8	Correlations between UER with (top) Gabor noise parameters ($\delta_{\text{gab}} = \{\sigma, \omega, \lambda, \xi\}$) and (bottom) Perlin noise parameters ($\delta_{\text{per}} = \{\lambda_x, \lambda_y, \phi_{\text{sine}}, \omega\}$).	63
3.9	Visualisation of YOLOv3 outputs on (left) images from the MS COCO dataset [5] and (right) the same images adversarially perturbed with procedural noise. .	79
3.10	Histogram of metrics over all perturbations on YOLOv3. Results for (top) all classes and (bottom) the “person” class.	81
3.11	Histogram of metrics over all perturbations on Inception v3 with and without median denoising for (left) Gabor noise and (right) Perlin noise for ℓ_∞ -norm $\varepsilon = 16$	85
4.1	Examples of UAPs generated from (top) the different ResNet-50 [6] models and (bottom) VGG [7] and DenseNet [8] models. For the top row, the titles for each indicate how the model was trained, complete details for these are in Section 4.1.	90
4.2	UER (in %) of untargeted UAPs for different perturbation values ε . Plot titles indicate the evaluated model.	92
4.3	Confusion matrix showing the UER (in %) of untargeted UAPs with $\varepsilon = 10$ when applied as transfer attacks to different models. VGG19 and DenseNet121 are labeled as V19 and D121 respectively.	93
4.4	Examples of targeted UAPs generated for IN, SIN, and SIN+IN.	96
4.5	Distribution of TSR of targeted UAPs on the ResNet50 models, with one UAP for each ImageNet class labels.	96
4.6	Top five output labels for each tested model given the UAP used to perturb the dataset. In each plot we label the source of the untargeted UAP generated via SGD for $\varepsilon = 10$	97
5.1	Test accuracy of LeNet on MNIST (left) and Fashion-MNIST (right) for various Jacobian Regularisation strengths λ_{JR}	114

5.2	Effectiveness of untargeted UAPs for various ℓ_∞ -norm perturbation constraints ε . Plots are shown for various models with different degrees of Jacobian Regularisation.	114
5.3	Average Targeted Success Rate (TSR) of targeted UAPs generated for each class, with error bars showing standard deviation across UAPs for different classes. Plots are shown for various models with different degrees of Jacobian Regularisation.	115
5.4	Jacobian similarity for pairs of inputs on MNIST (left) and Fashion-MNIST (right) for LeNet with and without Jacobian Regularisation (JR). Median similarity values on MNIST are 0.18 and 0.58; and on Fashion-MNIST are 0.11 and 0.46 with and without JR respectively.	116
5.5	Clean accuracy of LeNet on MNIST (top) and Fashion-MNIST (bottom) for various JR strengths λ_{JR} and with varying number of base learners per ensemble method.	123
5.6	Model robustness against UAP with $\varepsilon = 0.15$ of LeNet on MNIST (top) and Fashion-MNIST (bottom) for various JR strengths λ_{JR} and with varying number of base learners per ensemble method.	124
5.7	Clean accuracy versus robustness trade-off of LeNet on MNIST (top) and Fashion-MNIST (bottom) labeled by λ_{JR} for various UAP attack strengths ε	125
6.1	Top 10 percentile of mean feature activation values of ResNet50 [6] on residual block 3.0 for ImageNet images with and without UAPs. We plot the average over 10,000 randomly chosen validation set images.	131
6.2	Visualizations for an image from the ImageNet dataset (left), with an adversarial noise (center), and with an adversarial patch (right). The clean image is classified as a great white shark (class #2) while the other two are classified as brain coral (class #109) by a ResNet50 model. The second and third rows visualise selected features from chosen residual blocks of the model. The features' computed mean and standard deviation (std) are shown below and are noticeably larger for the perturbed inputs.	133

6.3	Mean and standard deviation of AUC per layer for ResNet18 against SGD-Layer for calibration at $\varepsilon = 10$	138
6.4	Evaluation metrics for ResNet18 against UAPs for different ε . Top row are untargeted UAPs, bottom row are targeted UAPs for class label indices 0, 4, and 5.	140
6.5	Mean AUC for HyperNeuron with each combination of observed layer and aggregation function against SGD-Layer attacks with $\varepsilon = 10$ on ResNet50.	143
6.6	Evaluation metrics of HyperNeuron - 3.0M on ResNet50 against a variety of UAP attacks for different ε	143
6.7	Distribution of TSR of targeted UAPs across all 1,000 ImageNet class labels with $\varepsilon = 10$ on ResNet50.	144
6.8	Mean AUC for HyperNeuron with each combination of observed layer and aggregation function against Adversarial Patch attacks with 16% patch size on ResNet50.	146
6.9	Top row shows metrics for targeted adversarial patches against ResNet50 on ImageNet. There is significant overlap between the Best AUC and Block 4.2 AUC. Bottom row shows the AUC per layer.	147
6.10	Defence-aware attacks against HyperNeuron for ResNet50 on ImageNet validation. ASR of SGD on the undefended model is shown on the right subplot in gray for reference.	156
6.11	AUC per layer of HyperNeuron for YOLOv3 against overlay and overflow attacks at $\varepsilon = 1$	157
6.12	Evaluation metrics for YOLOv3 against overlay and overflow attacks for different ε . The plots on the right show the total number of detected ads.	158

List of Tables

3.1	UER (in %) by percentile for random and procedural noise on Inception v3. . . .	61
3.2	Input-dependent evasion rate (in %) for random and procedural noise perturbations. Original refers to the top 1 error on the unaltered original images. Strongest attack on each classifier is highlighted.	65
3.3	Top 5 input-dependent evasion rate (in %) for random and procedural noise perturbations. Original refers to the top 5 error on the unaltered original images. The strongest attack on each classifier is highlighted.	65
3.4	Comparison on Inception v3 between universal Perlin and Gabor noise ZK attacks at ℓ_∞ -norm $\varepsilon = 16$. Universal evasion rates (%) of the optimised perturbations are shown for their respective training set and the validation set.	75
3.5	Highest UER (in %) of Fourier and Procedural noise UAPs for models at ℓ_∞ -norm $\varepsilon = 16$. Best attack is in bold.	75
3.6	Comparison on Inception v3 between our input-dependent Perlin and Gabor noise attacks and the bandit attack [9] for different query limits at ℓ_∞ -norm $\varepsilon = 16$	77
4.1	Percentiles of TSR (in %) of targeted UAPs on the ResNet50 models on the clean ImageNet validation set.	98
4.2	Top and bottom three class labels ranked according to their average targeted success rate (in %) across the ResNet50 models.	100

4.3	Accuracy (in %) of ResNet50 models and ensembles for the clean ImageNet validation set and on the worst-case UAP at $\epsilon = 10$. Highest model accuracy is highlighted.	102
5.1	Performance metrics (in %) of LeNet. Jacobian Regularisation (JR) uses $\lambda_{JR} = 0.05$. UAP evaluations are for ℓ_∞ -norm attacks at $\epsilon = 0.2$ for MNIST and $\epsilon = 0.15$ for Fashion-MNIST. Lowest values indicate the best robustness and are highlighted.	115
5.2	Weighted accuracy (in %) of LeNet for MNIST (top) and Fashion-MNIST (bottom). The first 3 rows show the models with highest weighted accuracy. The bottom 3 rows show the <i>best</i> weighted accuracy of models trained with only JR, only ensemble, and “standard training” (neither JR nor ensemble).	126
6.1	Metrics for targeted UAPs at $\epsilon = 10$ against ResNet18 on CIFAR-10.	140
6.2	Metrics for targeted UAPs at $\epsilon = 10$ against ResNet50 on ImageNet.	145
6.3	Detection performance for adversarial noise and patch UAPs against ResNet50.	149
6.4	Metrics (in %) summarizing metrics from Table 6.3 configurations against adversarial noise and patch UAPs on ResNet50.	150
6.5	Average time for inference and detection in milliseconds (ms) on a batch size of 64 images on ResNet50.	150
6.6	Comparison of HyperNeuron with non-detection defences over noise and patch UAPs on ResNet models for ImageNet. Lower ASR and higher CP indicate better model performance. All metrics are in %.	153
6.7	Comparison of HyperNeuron with non-detection defences on noise and patch UAPs on ResNet models for ImageNet. First six rows show ASR for each attack. Metrics are in %.	153

Chapter 1

Introduction

Advances in computation and data have enabled deep neural networks (DNNs) to transform the field of computer vision. The dramatic rise in popularity of DNNs for computer vision can be traced back to their breakthrough “ImageNet moment”. Krizhevsky et al. dominated the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) competition with the only neural network-based solution [10]. Their method achieved 41% better performance than the next best competitor, and it was this seminal moment that highlighted the performance of DNNs and demonstrated the viability of deep learning algorithms for large-scale computer vision tasks. In the following years, the top submissions to the ILSVRC competition were all based on DNNs [11]. Unlike traditional software, DNNs are not explicitly told through code how to solve a task. Instead, these models are often trained to learn a solution from training data that contains examples of inputs matched with their corresponding expected outputs. The result of this is a model that has learned some approximate solution to the task posed by its training dataset. In recent times, DNNs have been able to achieve human-level performance in popular computer vision tasks such as image classification [10] and object detection [12]. Aside from these tasks, DNN models are also used in highly sensitive and security-relevant computer vision applications such as facial recognition [13], object tracking [14], autonomous vehicles [15, 16, 17], and medical imaging [18, 19, 20]. Thus, the integrity of the models’ predictions become of paramount importance; integrity here referring to the faithfulness of

model predictions with their expected outputs.

Despite the state of the art performance and widespread use of DNNs in computer vision tasks, there remains a huge gap in the understanding of their vulnerabilities. One of the most prominent types of attacks on these models comes in the form of *adversarial examples*—seemingly benign inputs that are specifically designed to fool the model [21, 22]. This is known as an *evasion attack* as it causes the model to make a mistake during inference, thus “evading” the model. Because of this, adversaries that can manipulate the inputs to these models are able to compromise the integrity of the model predictions.

Many adversarial example attacks are based on “input-dependent” adversarial perturbations—where a different perturbation is computed for every benign input to create a unique adversarial example [23, 24, 25]. This threat model may seem unrealistic in practice as it requires an adversary with significant computation power and exclusive levels of access to the model and data. However, researchers have demonstrated the existence of **Universal Adversarial Perturbations (UAPs)**—single adversarial perturbations that can be applied to large sets of data to fool the model on many of the affected inputs [1]. UAPs are an extremely concerning threat as they exploit systemic vulnerabilities in the model, enable physically-realizable universal adversarial attacks [26, 27], and drastically reduce the cost and access required for performing adversarial attacks on computer vision DNNs [28].

As UAPs are a more powerful version of adversarial perturbations, they pose a greater threat to computer vision DNNs in practice. It is, thus, imperative that universal attacks be efficiently mitigated to secure computer vision systems and disincentivise attacks on DNNs. Although input-dependent adversarial perturbations have been shown to be extremely difficult to defend against [29, 30, 31], the mitigation of *universal* attacks against DNNs is relatively understudied.

1.1 Goal and Scope

The primary goal of this thesis is to understand and mitigate the effectiveness of UAPs in the context of computer vision DNNs. We focus on computer vision tasks and in particular, image

classification. Adversarial Machine Learning research has predominantly been computer vision driven [32, 33] due to (a) the accessibility of labelled images, (b) wide adoption of benchmark datasets, and (c) the intuitive nature of vision for humans. Humans are equipped with their own vision systems which makes it easier to identify and verify adversarial examples as well as understand manipulations applied to images. As a result, research is more challenging for other applications such as malware detection and reinforcement learning because the notion of an adversarial perturbation is not as intuitive or easy to evaluate and verify in these domains. Our focus is primarily on image classification as it is one of the core computer vision problems. It is standard practice that feature extraction layers of models pre-trained on image classification benchmarks like ImageNet [11] are also used as the initial layers for computer vision DNNs in more advanced tasks such as image segmentation [34, 35] and object detection [12].

To understand UAPs, we wish to investigate two primary aspects of UAPs. The first objective is to replicate and characterise the distinct patterns present when visualising UAPs in the input space. This is important as the patterns present in UAPs are one of its most salient visual features, and these are not always present in input-dependent adversarial perturbations. The second objective is to investigate the effect of UAPs on the hidden layer activation values of the DNN during inference. This is interesting as several UAP attacks rely on saturating hidden layer activations [3, 36, 2], its prevalence in the literature can lead to generalisations about UAPs that could be used to detect them. To demonstrate a more complete understanding of UAPs, we construct novel attacks and defences based on these characterisations and observations.

There are many ways to investigate and characterise UAPs, the impact of input perturbations is mainly observed through either the input image space or the DNN activation space. We choose to investigate both these aspects as they represent the clearest manifestations of UAPs that can be described and addressed, and we find that our exploration of each is sufficient to improve our understanding and to construct new attacks and defences. To mitigate attacks, it is necessary to identify inescapable trade-offs that attackers must face. We first show in this thesis that there is a trade-off between the UAP attack effectiveness and the ease of evading a defence. Indeed there is no recourse if an attack is highly effective and can not be defended against. Second, from the defender’s perspective, the main challenge is to address the accuracy-robustness trade-

off, i.e. how to achieve the model’s desired robustness to UAPs without sacrificing its accuracy on clean inputs. Thus, our goal is to propose new techniques that demonstrate and improve upon these two key trade-offs.

For defences, other DNNs can be applied to automatically detect or filter malicious inputs. However, we refrain from introducing additional DNN components to our proposed defences as this raises further challenges, and in particular the vulnerability of these components themselves to adversarial examples. Thus, we focus instead on improving the core training algorithms or only introducing the most basic explainable defence. Finally, while our proposed defences are not exclusively the only ways to defend models, they help us understand and improve on the key trade-offs previously mentioned. Thus, our methods can serve as a guide to make more effective defences in future work.

1.2 Summary of Contributions

Procedural noise as task-agnostic UAPs and efficient zero-knowledge attacks. In contrast to per-instance adversarial perturbations, UAPs appear to exhibit clear visual structures and patterns in computer vision problems. We use this observation to replicate these visual patterns with procedural noise functions, and we show that procedural noise acts as effective UAPs. Procedural noise functions are fast, scalable, and lightweight functions for generating structured image patterns. Not only are they effective UAPs, but since their generation is agnostic to the model or computer vision task, we demonstrate extremely query-efficient zero-knowledge attacks on various computer vision models by utilising their universality. The visual appearance of these procedural noise closely resembles that of the features that DNNs learn in the earlier layers, which suggests that this vulnerability to procedural noise originates from those layers—and these early layers often represent more textural image patterns.

Showing diversity of UAPs and characterising them by dominant class labels. In accordance with the observations above, there is a prevailing notion that increasing DNN’s shape-bias makes them more robust in general. Indeed, existing work has shown that an increase

in shape-bias improves model performance against common corruptions and noise. However, we demonstrate that shape-bias does not improve the “worst-case” adversarial robustness against UAPs. This result implies that a model’s texture or shape-bias is likely not the underlying cause for its vulnerability to adversarial examples and universal attacks. Moreover, we use UAPs as a tool to visualise the features that each model is more vulnerable to. We show that the visualisation of UAPs for shape-biased models appears to be more visually interpretable than that of the texture-biased models. We also characterise the different types of UAPs based on that UAP’s most dominant output class label. Finally, the analysis with UAPs shows that the models have very diverse vulnerabilities. Based on this, we propose an ensemble voting mechanism as a way to mitigate vulnerability to universal attacks.

Improving accuracy-robustness trade-offs with Jacobian ensembles. A number of studies have shown that Jacobian regularisation and model ensembles can improve the *stability and accuracy* of machine learning algorithms. We theoretically prove and then empirically verify that Jacobian regularisation improves model robustness to UAPs. Moreover, we propose Jacobian Ensembles, a combination of Jacobian regularisation and model ensembles, and show that the combination can greatly improve robustness against UAPs whilst maintaining and sometimes even improving the clean accuracy. This stands in contrast to existing defences, which, up to now have improved adversarial robustness at a great cost to clean accuracy. By showing that the accuracy-robustness trade-off can be greatly improved we put defences for computer vision DNNs against UAP attacks on a more practical footing.

Efficient detection of universal attacks. We observe that the neuron activations for UAPs are noticeably much larger than that of clean images, i.e. that UAPs “hyper-activate” neuron values. From this observation, we construct and propose a novel detection mechanism HyperNeuron and show that it can reliably detect both adversarial noise and adversarial patch UAPs for image classification and object detection tasks. Our method also performs as well as or even better than existing defences. We demonstrate that HyperNeuron can detect universal attacks for a variety of attacker objectives, revealing a trade-off between a UAP’s detectability and its generalisability across inputs. Finally, we apply HyperNeuron to defend real-time perceptual ad-blocking models at low latency, showing that it is an effective and practical defence.

1.3 Thesis Outline

The remainder of this thesis is organised as follows. In Chapter 2, we outline the basic definitions and threat models that we will use. We will also review the existing literature relevant to UAP attacks and defences in the context of computer vision DNNs. In Chapter 3, we construct procedural noise attacks that replicate the visual patterns of UAPs and demonstrate that procedural noise can act as effective zero-knowledge UAP attacks. In Chapter 4, we analyse the UAP robustness of models trained on texture and shape-biased datasets, and we demonstrate that the most frequent output labels of UAPs can be used to quantitatively describe the vulnerabilities of these models. In Chapter 5, we theoretically prove and empirically show that Jacobian regularisation and model ensembles improve robustness to UAPs. We then propose Jacobian Ensembles to improve UAP robustness whilst preserving model accuracy. In Chapter 6, we introduce a novel detection mechanism that uses hidden layer activation values to detect a wide variety of UAP attacks. Finally, in Chapter 7, we summarise our achievements and discuss potential avenues for future work.

Chapter 2

Background

This chapter is an overview of the current Adversarial Machine Learning literature focusing on Universal Adversarial Perturbations (UAP). Existing attacks and defences for deep neural networks (DNNs) are predominantly characterised in computer vision settings [32, 33] due to the availability of datasets, benchmarks, and models. The focus in this chapter is on the threat model as well as UAP attack and defence algorithms.

2.1 Preliminaries

2.1.1 Neural Networks

In this thesis, we focus more narrowly on neural networks used for image classification and UAP evasion attacks. We use the terms “model”, “network”, “classifier”, and “DNN” interchangeably.

In classification problems, a DNN can be written as a function f which takes an input $x \in \mathbb{R}^D$ and outputs a probability vector $f(x) \in \mathbb{R}^C$, where D is the dimensionality of inputs and C are the number of class labels. The output label assigned by the model is given by the class label with the highest probability $F(x) = \arg \max f(x)$.

The DNN is a hierarchical composition of n parametric functions f_i that model input x , these f_i are also referred to as *layers* of the neural network [37]. Each function f_i for $i \in 1, \dots, n$ can be modeled as a layer of neurons, these neurons are computing units that apply an *activation function* to the previous layer's weighted outputs to generate a new output, typically to be fed to the next layer. Each layer is parametrised by a weight vector θ_i which affects neuron activation values. Thus, we can write f as a composition of its layers in the following way:

$$f(x) = f_n(\theta_n, f_{n-1}(\theta_{n-1}, \dots, f_2(\theta_2, f_1(\theta_1, x)))) \quad (2.1)$$

f is implicitly dependent on model parameters θ , which includes all weight vector and bias parameters. However, when evaluating the robustness of these models against attacks these parameters are fixed, so for convenience we do not show the dependence on θ from hereon.

Each layer makes use of some non-linear activation function, typical choices for these are tanh [38], sigmoid, or Rectified Linear Units (ReLU) [39]. We focus our work on networks that use a ReLU activation function, as it currently is the most widely used activation function [38, 37]. This activation function is defined by $\text{ReLU}(z) = \max\{0, z\}$.

2.1.2 Adversarial Examples

DNNs have been shown to be vulnerable to attacks that can be performed during model inference [21, 22, 40]. An attacker can perform evasion attacks where they generate *adversarial examples*, these are legitimate inputs modified slightly to produce model errors. These adversarial examples greatly undermine the integrity and reliability of model predictions.

We first introduce a formal notion of adversarial examples. As defined in the last subsection, let f be a classifier with class label output $F(x) = \arg \max f(x)$. Let $\tau(x)$ denote the true class label of an input x . An adversarial example x' is an input that satisfies $F(x') \neq \tau(x)$, despite x' being almost equal to x according to some distance metric (implicitly, $\tau(x) = \tau(x')$). The difference $\delta = x' - x$ is referred to as an *adversarial perturbation* and it is often constrained

to $\|\delta\|_p \leq \varepsilon$, for some ℓ_p -norm and small $\varepsilon > 0$ [22]. We discuss the choice of perturbation distance metric, such as the ℓ_p -norm, in greater detail later in Section 2.2.2.

2.1.3 Universal Adversarial Perturbations

For a dataset $x \in X$ and classifier $f : X \subset \mathbb{R}^D \rightarrow \mathbb{R}^C$ a **Universal Adversarial Perturbation (UAP)** is an adversarial perturbation $\delta \in \mathbb{R}^D$ that satisfies $f(x + \delta) \neq \tau(x)$ for sufficiently many $x \in X$ of a benign dataset X . That is, $x + \delta$ is an adversarial example for sufficiently many inputs in the dataset [1].

UAPs show that models have systemic vulnerabilities that can be exploited regardless of the input [28, 41, 42]. As they are optimised over multiple inputs, UAPs exploit global model sensitivities rather than per-input characteristics. UAPs are also *transferable* across models [1]. This property is exploited in practical transfer attacks [43, 44, 45]. We discuss existing attack and defence algorithms for UAPs later in Section 2.3. But first, we discuss the threat model which is necessary to give context to UAP attacks.

2.2 Threat Model & Metrics

Threat modeling allows for the formalization of attack scenarios. The structure of this is laid out by Szegedy et al. where the threat model can be described in terms of the adversary’s goals, capabilities, and knowledge [22, 25]. We describe each of these three categories in the context of UAP evasion attacks for computer vision DNNs. In each subsection, we outline the metrics or constraints we use that allow us to formally define the threat model.

2.2.1 Adversarial Goals

In the general case, *adversarial goals* can be put into four broad categories according to impact on: confidentiality, integrity, availability, and privacy. The first three come from the classical

CIA security model whilst the latter is proposed by Papernot et al. [33]. These categories can be grouped into two pairs due to their inherent similarities, (1) confidentiality and privacy, and (2) integrity and availability. Evasion attacks such as adversarial examples primarily concern the latter two as these are constructed to undermine the model’s outputs.

Integrity and Availability. These refer to the faithfulness of model predictions with their expected outputs. Attacks on integrity aim to control model outputs in a way that the adversary desires. Closely related, attacks on availability look to reduce the quality or performance of the machine learning system, making it unreliable. While they have differences, the attack vectors in our case are similar.

Integrity is a crucial factor for evaluating machine learning models as it is the basis of the performance metrics used for them. This includes metrics such as accuracy, f-scores, and false positive rates. It has been shown that the integrity of machine learning systems can be compromised by adversarial examples as these are inputs designed to make the algorithm have incorrect outputs. These have been shown to exist and occur in several machine learning models, including state-of-the-art neural networks. For example, a classifier could assign the wrong class to an authentic image or confidently classify a meaningless image [46].

Availability is about the prevention of access to a feature, making the target model inconsistent or unreliable when some of their input features are corrupted or missing [47]. Availability attacks can tamper with images, making the prediction unreliable in general [33]. For example, a compromised input feed can allow for adversarial perturbations to be applied on all images that pass through, so that the model predictions become unreliable for the user.

Goals for UAP Attacks. Choosing the right metrics for adversarial goals allows them to formalise objective functions to optimise their attacks. UAP attacks can be broadly classified into the two following categories based on their *primary attack success metric*. We will be using these two metrics in this thesis.

1. **Untargeted attack.** Sometimes referred to as “indiscriminate” attacks, the goal of the UAP here is to maximise the number of inputs the model makes an error on. In image

classification, an error is when the resulting image is classified as anything except its true label ($F(x') \neq \tau(x')$). For an attacker, this is useful in the general case where they indiscriminately apply the UAP to the input feed to cause model error, thus damaging overall performance and affecting *availability*. Attack success for this can be measured with an **Universal Evasion Rate (UER)** which is the proportion of the input dataset that the model makes errors on when the UAP is applied. Formally, the UER of perturbation δ for model f on dataset $x \in X$ can be written as:

$$\text{UER}(X, \delta) = \frac{|\{x \in X : \arg \max f(x + \delta) \neq \tau(x)\}|}{|X|} \quad (2.2)$$

where $|\cdot|$ denotes the cardinality of the set.

2. **Targeted attack.** The goal of the UAP here is to maximise classification towards a particular class label y_{tgt} as chosen by the attacker ($F(x') = t$). This is more challenging to execute than untargeted attacks as it requires classification of as many inputs as possible towards the desired “sink” class y_{tgt} .

This is more insidious in practice as it can be used for more targeted applications (e.g. in autonomous driving, funneling classification of any road signs to “Speed Limit”), undermining model *integrity*. Attack success for this can be measured with a **Targeted Success Rate (TSR)** which is the proportion of the input dataset that the model classifies as y_{tgt} . Formally, the TSR of perturbation δ for model f on dataset $x \in X$ targeting class y_{tgt} can be written as:

$$\text{TSR}(X, \delta, y_{\text{tgt}}) = \frac{|\{x \in X : \arg \max f(x + \delta) = y_{\text{tgt}}\}|}{|X|} \quad (2.3)$$

2.2.2 Adversarial Capabilities

Adversarial capabilities refer to how an adversary can modify aspects of the machine learning pipeline from the input to the model. As we are primarily concerned with studying adversarial perturbations, we restrict the adversarial capabilities to input modifications. For the rest of this

discussion, we describe the types of modifications that the adversary can apply to the input.

Image Representation. We are focused on computer vision applications, so the input images we consider come in the form of:

1. *Grey-scale* images $x \in \mathbb{R}^{hw}$ where x is a two-dimensional $h \times w$ -pixel grey-scale image and pixel intensities are bounded in the range $[0, 1]$.
2. *Coloured* images $x \in \mathbb{R}^{3hw}$ similar to grey-scale images, but now with three colour channels. We do not convert RGB images to other representations such as HSV or HSL as the DNN acts on the raw pixel values.

Perturbation Distance Metrics. Given a benign image x that is correctly classified by the model, the idea of adversarial perturbations is to modify the image so that it remains *perceptually similar* with the original, yet is misclassified by the model. There is currently no distance metric that perfectly replicates human perception. As a result, existing distance metrics only serve as a *proxy* for perceptual similarity. Determining the most optimal metric is not within the scope of this thesis. However, constructing and evaluating distance metrics for adversarial examples remains an important research question we leave to future work.

In the general case, we have $d(x, x') \leq \varepsilon$ for some distance metric d and small $\varepsilon > 0$ to indicate that the original input x and adversarial example $x' = x + \delta$ are similar. The most commonly used metrics in the literature are the ℓ_p -norm metrics with $p = 0, 2$ and ∞ . The ℓ_p distance metric is written as $\|x - x'\|_p = \|\delta\|_p$ with p -norm $\|\cdot\|_p$ defined by

$$\|v\|_p = \left(\sum_{i=1}^n |v_i|^p \right)^{\frac{1}{p}}$$

1. The ℓ_0 -norm measures the number of input coordinates i such that $x_i \neq x'_i$. In the context of images, this corresponds to the number of pixels that have been manipulated by an adversary. Papernot et al. argue for the use of this distance metric [48].
2. The ℓ_2 -norm measures the root mean square distance and was used in the first adversarial example paper [22]. This distance can remain small when changes in pixel values are small

in magnitude. However, in the context of UAPs, small changes for one image could be large for another, making it difficult to optimise UAPs under this metric.

3. The ℓ_∞ -norm measures the maximum change for any individual pixel:

$$\|x - x'\|_\infty = \max(\{|x_i - x'_i|\}_{i=1}^n)$$

where i are the pixel coordinates for images with n total pixels. In this case, the attacker is able to modify any pixel so long as it is within the maximum perturbation budget defined by the ℓ_∞ -norm. Warde-Farley and Goodfellow argue that this ℓ_∞ is the optimal distance metric to use [49].

Adversarial perturbations are primarily generated through optimisation algorithms. A challenge when applying distance metrics is enforcing their constraints without hampering the optimisation. This is why ℓ_p -norms have persisted as the most used metrics as they are relatively simple to apply in existing adversarial attacks, whereas other distance metrics like Structural similarity index measure (SSIM) [50] and Wasserstein distance [51] require their own bespoke adversarial attack algorithms to account for the more computationally involved constraints these metrics introduce [52, 51].

When generating UAPs, the resulting perturbation has to be effective and must satisfy the constraints *simultaneously* across all the inputs. As a result, it becomes difficult to optimise UAPs under the ℓ_2 -norm constraint due to large variations in pixel values between different images. This leaves us with ℓ_0 and ℓ_∞ -norms as the most feasible to apply with current UAP generation algorithms—in line with the existing UAP literature [1, 2, 53].

Patches & Noise. Two of the most common types of UAPs for computer vision models come either in the form of patches or noise, depending on whether the perturbation is constrained to a part of the image or diffuse across all its pixels. **Adversarial noise** are more generally considered as bounded by an ℓ_∞ -norm and **adversarial patches** by an ℓ_0 -norm.

Although there are exponentially many pixel combinations that can be chosen when implementing ℓ_0 -norm attacks, the most physically realisable are those that manipulate a continuous

region in the image. In the adversarial patch setting, the attacker can modify a continuous 2D region in the image, typically in the shape of a square or circle under ℓ_0 -norm constraints. In practice, this is done by inserting a patch or 2D texture into the camera’s field of view. We primarily consider the adversarial patch for the ℓ_0 -norm scenario since it is the most practical and physically realisable attack for this threat model [54, 27, 26]. Although these ℓ_p -norms do not directly represent real-world scenarios, they are important tools for basic research as it allows for clarity, quantitative comparisons, and benchmarking to advance the field [29, 55].

2.2.3 Adversarial Knowledge

Adversarial knowledge in our work refers to the level of access the attacker has to read the internal parameters of the victim model as well having access to a dataset that is representative of the victim model’s training dataset. Knowing the internal parameters (weight vectors) of the victim model can greatly aid the attacker in generating more effective adversarial attacks, and many existing adversarial attack algorithms rely on this knowledge to successfully generate adversarial examples. In the general case, attackers have the ability to query the model with an input and observe the output. The adversary’s knowledge can be broadly classified into three categories from most to least access [56]:

1. **Perfect-Knowledge (PK).** This is sometimes referred to as “*white-box*” in the literature. This strong adversary has full knowledge of the victim model and its internal parameters. Many adversarial attack require this PK access to use their gradient-based optimisation attacks [22, 23, 25, 24].

A feasible real-world scenario of this is with open-sourced models and datasets. Although in practical settings it is often difficult to obtain access to this sensitive model information—typically requiring insider knowledge or reverse-engineering of the victim model. Nonetheless, a strong PK attacker should still be accounted for when evaluating model robustness [33].

2. **Limited-Knowledge (LK).** This is sometimes referred to as “*grey-box*” in the literature.

This adversary does not have read access to the victim model and its internal parameters. However, it has full access for a similar “surrogate” model trained on the same task as the victim model with a dataset of similar size and quality.

LK adversaries exploit the *transferability* property, where the same adversarial example is able to fool models other than the one it was generated from [43]. They generate PK adversarial examples on their surrogate and then query these examples to the target victim model [44]. UAPs are known to transfer across models trained on the same task [1, 28]. Transfer attacks are not always successful, especially if there are significant differences between the victim and surrogate models.

3. **Zero-Knowledge (ZK).** This is sometimes referred to as “*black-box*” in the literature. This adversary has no access to the internal parameters of the victim model or any comparable surrogate models. They solely rely on the outputs generated from their queries to the victim model.

When evaluating ZK attacks, it is desirable to be *query-efficient* in that successful adversarial examples are generated with as few queries to the victim model as possible. This is to reduce the query budget needed by the attacker, e.g. if querying the target model is a paid service, or to avoid detection from the defender by lowering the volume of suspicious queries. If we allow for an unlimited number of queries, the adversary can simply reconstruct the victim model, which makes this scenario redundant.

As we are focused on widely-available image data from common benchmark datasets, we assume that our attackers have access to datasets that are representative of the victim model’s training dataset. LK attacks can be thought of as extensions to PK attacks as they use PK attack algorithms on surrogate models then transfer the adversarial examples to the victim model. Hence, the algorithms themselves for generating adversarial perturbations can be split into just two categories: either PK or ZK.

2.3 Universal Attack Algorithms

In this section, we aggregate the existing adversarial attack literature relevant to UAPs in the context of computer vision DNNs.

2.3.1 Fast Gradient Sign

Goodfellow et al. proposed the **Fast Gradient Sign Method (FGSM)** algorithm for generating adversarial examples in an efficient manner with minimal computation steps [23]. Although this work does not pertain exactly to UAPs, this was the first of many methods that utilised gradient-based optimisation for crafting adversarial examples. Using the notation introduced in Section 2.1, we define for a model f its loss function $\mathcal{L}(x, y)$. A common choice in classification tasks for this loss is cross-entropy [37]. FGSM computes the adversarial example with:

$$x' = x + \alpha \operatorname{sgn}(\nabla_x \mathcal{L}(x, y))$$

where $y = \tau(x)$, small step size $\alpha > 0$, and $\operatorname{sgn}(\cdot)$ is the piece-wise function:

$$\operatorname{sign}(x) = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}$$

At each step, FGSM effectively adjusts each pixel value towards the direction that maximises the loss function. This method is simple and efficient as it is a single-step gradient computation, although its attack success rate can be improved by more computationally expensive iterative methods which we later discuss. It is worth noting that computing the gradients through the victim model's loss function does require PK access to the model. This makes gradient-based optimisation attacks that explicitly use the model's loss necessarily PK in nature, and therefore requires a strong adversary.

2.3.2 Iterative DeepFool

Moosavi-Dezfooli et al. were the first to demonstrate the existence of UAPs for DNNs [1] using **iDeepFool**—an iterative version of their **DeepFool** algorithm originally made for input-dependent adversarial perturbations. The goal of DeepFool was to find the *minimal* perturbation needed to generate an adversarial example for a single input. Minimal in their definition refers to finding perturbations δ with the smallest magnitude under the ℓ_p -norm metric $\|\delta\|_p$. To generate δ , DeepFool iteratively updates the perturbation in the direction of $\nabla_x \mathcal{L}(x, k)$ similar to FGSM, but they choose the direction with class label k , $k \neq \tau(x)$ that minimises the magnitude of the step size taken. They demonstrate the efficacy of this algorithm for $p = 2$ and ∞ . For iDeepFool, they extend the DeepFool algorithm formulation to work on batches of images rather than individual ones by jointly updating δ over the average of $\sum_i \nabla_{x_i} \mathcal{L}(x_i, k)$. Examples of UAPs generated in this way can be seen in Figure 2.1. They also show that these UAPs can generalise across models, demonstrating transferability.

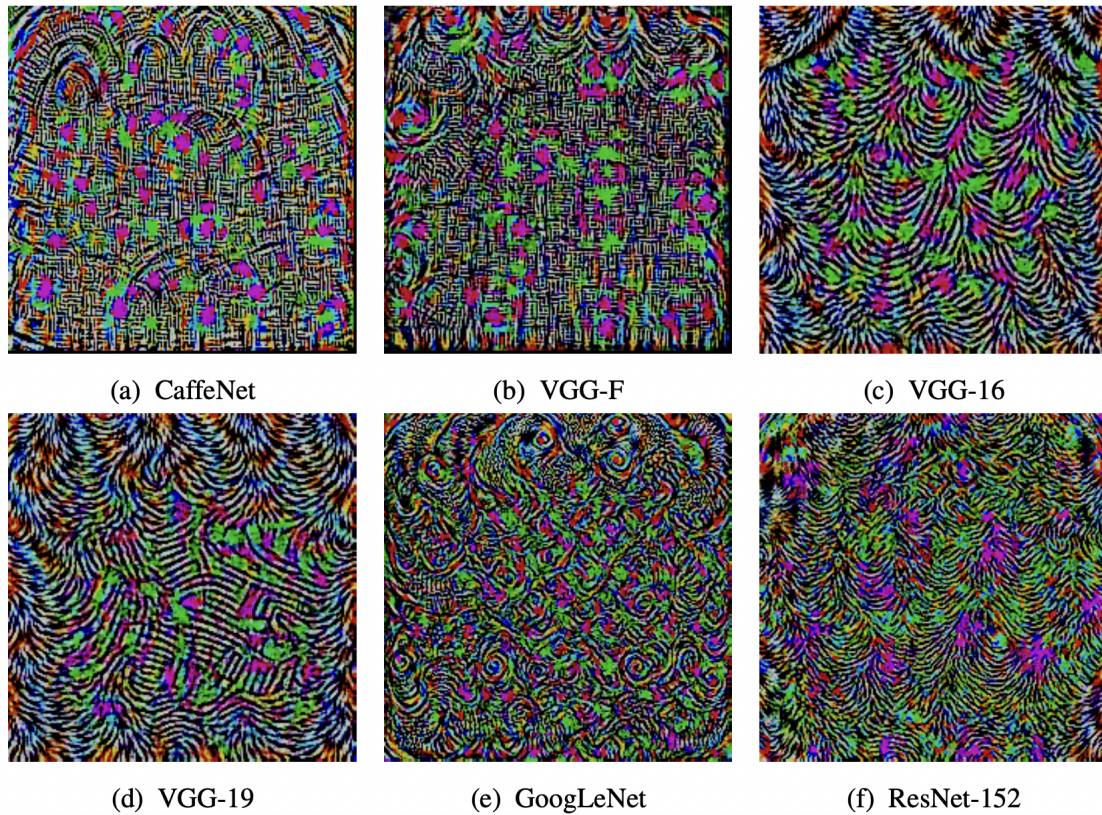


Figure 2.1: UAPs for various models generated using iDeepFool [1].

2.3.3 Stochastic Gradient Descent

Kurakin et al. and Madry et al. improve on FGSM by applying it iteratively. Most notable is a multi-step variant FGSM^k [57] which is then modified to be the Projected Gradient Descent (PGD) attack on the negative loss function [24]. The step taken at iteration $t + 1$ is given by

$$x^{t+1} = \text{clip}_{x+\mathcal{S}}(x^t + \alpha \text{sgn}(\nabla_x \mathcal{L}(x, y)))$$

where $\mathcal{S}$ is the space of allowed perturbations and α is the step-size per iteration. Unlike FGSM, this has a higher attack success rate at the cost of greater computation time due to multiple iterations rather than just a single one. And in contrast to DeepFool, PGD is more efficient as it is not designed to find the *minimal* perturbation. It is sufficient for PGD to find a perturbation so long as it evades the model and is within the perturbation constraints.

To adapt PGD to generate UAPs, the **Stochastic Gradient Descent (SGD)** algorithm was proposed [53]. UAPs are generated by maximizing the total loss across multiple inputs $\sum_i \mathcal{L}(x_i + \delta)$ using an iterative stochastic gradient descent algorithm: where $\{x_i\}$ are batches of inputs. Updates to δ are done in mini-batches in the direction of $-\sum_i \nabla \mathcal{L}(x_i + \delta)$. To generate targeted UAPs, we use the same stochastic gradient descent as in the untargeted case, but modify the loss to be minimised instead with all resulting inputs $x_i + \delta$ classified towards the target class y_{tgt} . SGD is a versatile and effective attack. Variations of it have been used to generate UAPs in real applications such as adversarial patches [54] and perceptual ad-blocking [58]. SGD is also the most effective among the PK UAP attacks, outperforming iDeepFool [1] and superseding the attacks [36, 2, 3] that we will discuss later in this section [53], making it the most suitable PK attack for evaluating robustness to UAPs.

2.3.4 Generalisable Data-free UAP

Mopuri et al. introduce a data-free objective function for generating UAPs by choosing perturbations that saturate hidden layer neuron activation values [36, 2]. Their primary motivation is to create UAP that are independent of the data. Their attack named **GD-UAP** aims to

find a perturbation δ that maximises hidden layer activations when applied to any input $x + \delta$. The intuition is that spurious firing of neurons in the hidden layer will result in inefficient information propagating and thus cause misclassification in the final layer regardless of x .

Let $f_l(\cdot)$ be the output of the l -th hidden layer of the network. Similar to iterative gradient-based techniques like DeepFool and PGD, GD-UAP updates their perturbation with gradient descent. However, instead of using the training loss of the model, they instead optimise a custom loss function to spuriously maximise all hidden layer activations, with ℓ_∞ -norm perturbation constraints for δ .

$$\mathcal{L}_{\text{GD-UAP}} = - \sum_i \log \left(\prod_l \|f_l(g_i + \delta)\|_2 \right) \quad (2.4)$$

At iteration $t + 1$ they have:

$$\delta^{t+1} = \delta^t + \alpha \text{sgn}(\nabla \mathcal{L}_{\text{GD-UAP}})$$

They propose several variations depending on the assumptions on the available data g . In the data-free case, $g = 0$. In the range prior case, they draw $g \sim \mathcal{N}(\mu, \sigma)$. When data is available, they draw from that dataset $g = x$. Naturally, they found that using more information improves the effectiveness of their generated UAPs.

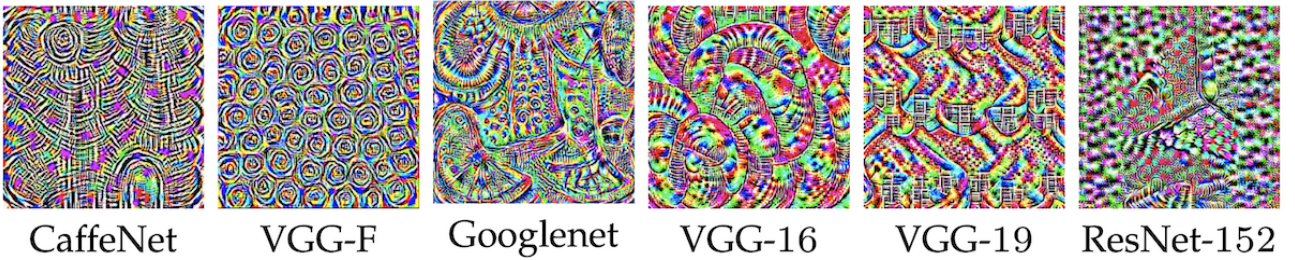


Figure 2.2: UAPs for various models generated with GD-UAP [2].

Examples of resulting UAPs generated can be found in Figure 2.2. Although GD-UAP is versatile depending on access to data, it is still not as effective as SGD for generating strong UAP attacks even when given full access to data. As both require PK access to the model, SGD is a more effective attack to evaluate models with. Nonetheless, the attack vector of GD-UAP targeting hidden layer activations is noteworthy for our work.

2.3.5 Singular Vector Attack

Khrulkov & Oseledets introduced their **Singular Vector Attack (SVA)**, a method for generating UAPs through the singular vectors of Jacobian matrices for intermediate layers of the model. Their strategy was to generate a perturbation vector δ that maximises the difference in hidden layer outputs between the original and perturbed image. The idea is that as the difference propagates further into the network, the change will result in dramatically different predicted outputs for x and x' [3].

Let $f_l(\cdot)$ be the outputs of the l -th hidden layer of the network. They want perturbation δ such that it maximises the quantity $f_l(x + \delta) - f_l(x)$ for a chosen layer l with an ℓ_q -norm constraint for the perturbation $\|\delta\|_q = \varepsilon$ for some $\varepsilon > 0$. They do this by using the approximation $f_l(x + \delta) - f_l(x) \approx J_l(x)\varepsilon$ where $J_l(x) = \frac{\partial f_l}{\partial x}$ is the Jacobian matrix of f_l . Thus, for any ℓ_q -norm

$$\|f_l(x + \delta) - f_l(x)\|_q \approx \|J_l(x) \delta\|_q$$

Maximizing the difference on the left hand side is equivalent to finding the vector δ that maximises $\|J_l(x) \delta\|_q$ under the constraint $\|\delta\|_q = \varepsilon$. Because this is invariant to scaling, it is sufficient to solve it when $\varepsilon = 1$. The solution to this problem is called the (p, q) -singular vector of $J_l(x)$. The computation for this solution can be done efficiently with the Power Method algorithm [59]. For UAPs, they then extend the formulation to multiple images $x_j \in X_b$ where $X_b \subset X$ is a random subset (batch) of the whole dataset X .

$$\arg \max_{\delta} \sum_{x_j \in X} \|J_l(x_j) \delta\|_q, \quad \text{where } \|\delta\|_q = \varepsilon$$

Let $\mathbf{J}_l$ be the matrix obtained by stacking $J_l(x_j)$ for all $x_j \in X$. This results in their final formulation where they compute for the singular vector δ :

$$\arg \max_{\delta} \|\mathbf{J}_l \delta\|_q, \quad \text{where } \|\delta\|_q = \varepsilon$$

Using this method, they were able to empirically identify the more fragile hidden layers and

found that UAPs generated from the earlier layers had larger success in evading the model [3]. Examples of their generated UAPs can be seen in Figure 2.3. The overall effectiveness of UAPs from SVA is noticeably less than those from the original iDeepFool method whilst still needing PK access to calculate the hidden layer Jacobians.

What is interesting with SVA and the previously discussed GD-UAP is their targeting of the hidden layer activations to facilitate UAP attacks. This observation will be useful later on when we develop our own defence against UAPs.

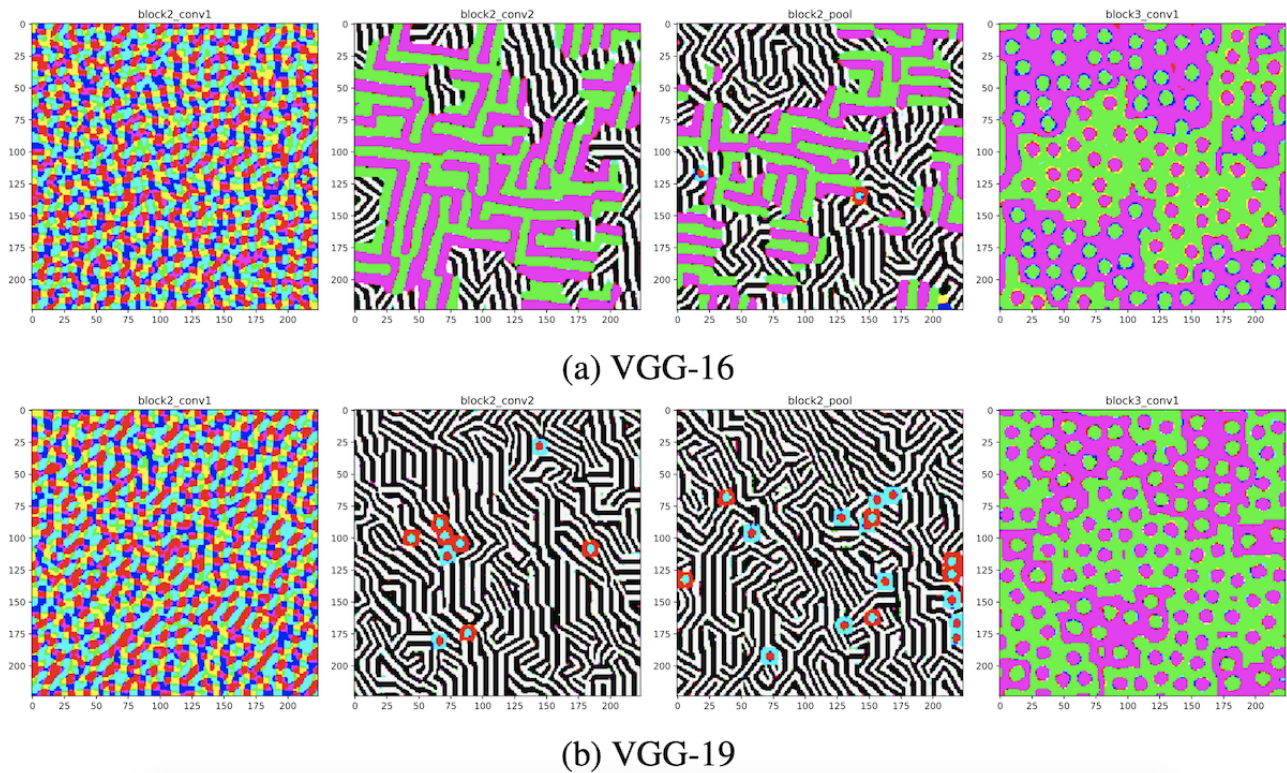


Figure 2.3: UAPs generated for VGG models using the singular vector method [3] on VGG-19.

2.3.6 Fourier Basis Functions

Tsuzuku & Sato were able to show that computer vision DNNs are sensitive to the directions of Fourier basis functions [4]. They show mathematically that *linear* convolutional neural networks are sensitive to a combinations of Fourier basis functions. They then hypothesise that it is a similar case for existing non-linear convolutional DNNs, assuming that a linear approximation holds well for DNNs.

They propose their **Signed Single Fourier Attack (SSFA)** where they optimise the frequencies of the Fourier basis functions under ℓ_∞ -norm perturbation constraints. They use an exhaustive search in their parameter space to find the best results, as their proposed function only has a few frequency parameters. SSFA is compelling as it is one of the few ZK UAP attacks, not requiring any internal knowledge of the victim model. However, their attack success rate is considerably low even for large ℓ_∞ -norm constraints of $\varepsilon = 10$ and 20.

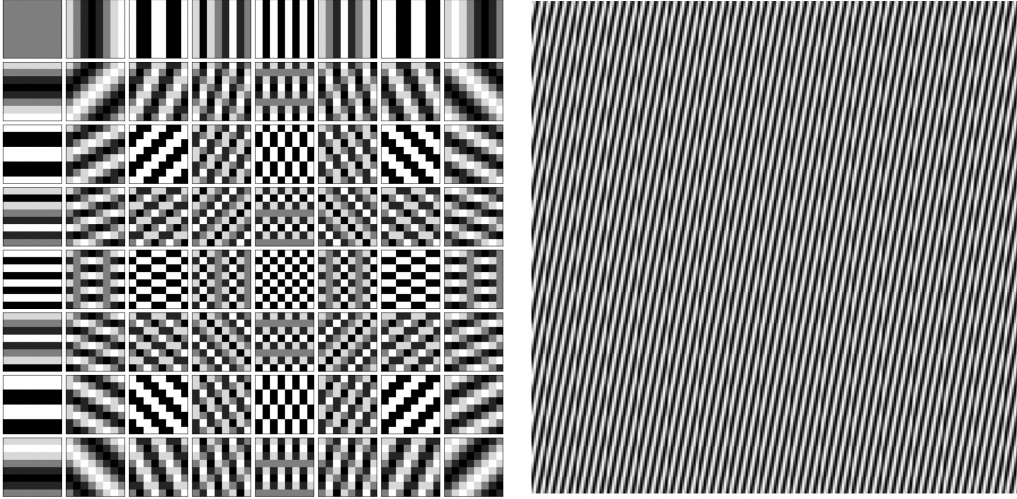


Figure 2.4: Visualisations of (left) Fourier basis in 8×8 space and (right) example of generated Fourier basis attack [4].

2.3.7 Adversarial Patches

Physically realisable universal attacks in the form of *adversarial patches* have been demonstrated against various computer vision tasks including: image classification [54], person recognition [27], facial recognition [13], and object detection [60, 61, 62]. They consist of inserting a 2D patch within the image frame to force classification towards the adversary’s desired class for the adversarial patch.

We first define the patch application function A for the resulting perturbed image $x' = A(p, x, t, l)$ where p is the patch, x is the original input image, t is a valid transformation on the patch, and l is the location of the patch within the image. We also define $\Pr(y_{\text{tgt}}|x)$ as the model’s output probability for the target class y_{tgt} given input x .

Athalye et al. summarises the most general form for generating robust physical adversarial attacks in their proposed Expectation Over Transformations (EOT) formulation [26], and Brown et al. adapt this for 2D image patches [54]. EOT aims to optimise the trained patch $\hat{p}$

$$\begin{aligned}\hat{p} &= \arg \max_p \mathbb{E}_{x \sim X, t \sim T, l \sim L} [\Pr(y_{\text{tgt}} | A(p, x, t, l))] \\ &= \arg \max_p \mathbb{E}_{x \sim X, t \sim T, l \sim L} [\Pr(y_{\text{tgt}} | x')]\end{aligned}\tag{2.5}$$

This objective to find the desired adversarial patch $\hat{p}$ is maximised via Stochastic Gradient Descent (SGD). Gradient values for the expectation is approximated through sampling transformations independently at each gradient descent step and differentiating through the transformation. As SGD applies gradient descent through the model’s internal parameters, this attack requires PK knowledge. Other works modify EOT for their specific threat model. For example, the Robust Physical Perturbation (RP₂) objective proposed by Eykholt et al. also optimises an EOT, with their objective function including an additional term that captures printability constraints [60, 61]. For our purposes, the formulation in Equation 2.5 is sufficient for generating adversarial patches for the threat models that we consider.

2.4 Defending Against Universal Attacks

In this section, we aggregate and put into context the existing defence literature relevant to UAPs for computer vision DNNs. Most of the literature has focused on defending against input-dependent adversarial perturbations. As the general principles behind defending input-dependent attacks can also be applied to UAPs, we first go over the broader context of defending adversarial examples before focusing on defences against UAPs in particular.

To construct robust defences for DNNs, we must first be able to evaluate the robustness of any given DNN. In general, there are two ways in which security is evaluated for a system [63]:

1. **A formal proof of robustness** is a set of mathematical statements that guarantee the robustness of the system under a stated threat model.

2. **An empirical demonstration of robustness** relies on applying existing and novel attacks against the system. Although stronger attacks than those found may exist, the system is no more secure than the best attack found so far.

In this thesis, we adopt an empirical approach to evaluating the robustness of models. For many practical computer vision tasks, formal proofs are more difficult to construct and apply than empirical demonstrations of robustness. The set of assumptions in formal proofs are often so restrictive that the guaranteed security is minimal in practice [64, 65, 66]. Empirical defences so far have been able to generalise to the complex tasks present in computer vision [67], so we will be applying this to investigate robustness and trade-offs in accuracy for DNNs.

2.4.1 Categorising Defences

Defences can be classified into three general categories depending on how they manage adversarial examples. Details for specific defences are explained in later sections.

- **Type I - Defences that stabilise predictions in the presence of perturbations.**

These defences are designed with the explicit goal of ensuring that model predictions remain the same for both the adversarial example and the original unperturbed image. They acknowledge the existence of adversarial examples and harden the model’s predictions against such attacks. These defences can be categorised further:

- **Data augmentation** is where perturbed images are introduced into the training set. A primary example of this is Adversarial Training (AT) where adversarial examples are added to the training dataset [22].
- **Denoising or filtering inputs** attempt to remove perturbations from the input. These apply functions or transformations to the input so that it gets mapped to a form that is closer to the model’s learned distribution.
- **Denoising or filtering in the activation space** attempt to correct hidden layer activations to what they define to be “normal”.

- **Type II - Defences that indirectly improve robustness.** These methods robustify the model without explicitly correcting its predictions on perturbed images. This can be done by diversifying learned features [68, 69] or penalising model instability through regularisation [70, 71].
- **Type III - Defences that detect adversarial perturbations.** These defences detect or “flag” inputs for adversarial perturbations. These can be classified depending on which part of the machine learning pipeline the detector focuses on:
 - *Input-based detection* analyse the input image directly.
 - *Activation-based detection* look to the model’s internal hidden layer activations when the input image passes through the model.

In the remainder of this section, we discuss existing UAP defence algorithms in greater detail.

2.4.2 Adversarial Training

Adversarial Training (AT) was the first proposed defence for adversarial examples [22] and remains one of the most analysed defences to date [31]. The idea of AT is to train the model on a mixture of clean and adversarial examples, with the goal of classifying both correctly. Goodfellow et al. proposed using adversarial examples generated via FGSM to make AT more efficient [23]. Later work by Madry et al. found that these previous versions of AT trained on too weak an adversary, and instead they demonstrated that AT on adversarial examples generated via PGD lead to more robust models [24]. Formally, AT can be defined as a min-max objective:

$$\min_{\theta} \max_{\delta \in \mathcal{P}} \mathcal{L}(\theta, x', y) \quad (2.6)$$

where $\mathcal{L}$ is the training loss. AT attempts to learn model parameters θ while finding the strongest adversarial examples (inner problem), whose classification loss is minimised (outer problem) [24]. Although the above Equation 2.6 is the general objective function for AT, there are many variations in the optimisation algorithms deployed for achieving this [25, 72].

AT is currently considered one of the best ways to defend against adversarial attacks [30, 31], though it still has some disadvantages. Training on a particular type of adversarial attack does not ensure robustness to different or more novel types of attacks. There are also trade-offs in performance and robustness when training on different types of adversarial attacks [73]. Note that this flaw is not limited to AT, as other defences that train on specific adversarial attacks are not guaranteed to cover novel adversarial attacks that they have not been previously encountered. AT also requires generating adversarial examples during training time and requires more data than standard training to achieve robust generalisation [74]. These factors combined make AT difficult to scale for large natural-image datasets, such as the ImageNet [11] image classification benchmark. Convergence of AT for the min-max problem shown in Equation 2.6 is difficult to achieve on such large datasets since the space of possible adversarial attacks is so large, often resulting in significant declines in model accuracy.

In the context of UAPs, **Universal Adversarial Training (UAT)** was proposed where they introduce UAPs generated via SGD into the training set [53]. Experiments were done on the natural-image ImageNet benchmark [11] where widely-used models such as VGG [7] and ResNet [6] achieve 75-80% accuracy. Despite this, the resulting ResNet models trained with UAT suffered a large degradation in performance with less than 50% accuracy on the clean dataset. This demonstrates the sacrifice in clean accuracy that AT often induces. We now go over defences designed specifically for UAP attacks.

2.4.3 Perturbation Rectifying Network

Akhtar et al. propose their **Perturbation Rectifying Network (PRN)** defence for UAPs [75]. It is a two-step system with a “rectifier” for denoising inputs and a “detector” for detecting perturbed inputs. This is a combination of a Type I - denoising and Type III - detection defence as discussed in Section 2.4.1. They designed their method so that the original DNN used for image classification does not require any modification.

The first component is the PRN which is designated as their perturbation “rectifier”. It is placed as “pre-input” layers to the original model. Let F be the output probability vector

of the original DNN. Given an input x , and the perturbed input x' . The PRN is optimised to compute a transformation T on the perturbed images such that it minimises the difference $\|F(x) - F(T(x'))\|_2$. It is trained with gradient descent to transform perturbed inputs such that the original model classifies the resulting “rectified” images in the same way as it does the unperturbed image. The perturbations they use in training are “synthetic perturbations”, which is their variation of the ℓ_∞ -norm attack based on the DeepFool [76] algorithm, but generated on only a few samples (< 100) due to reported computation constraints.

The second component is a perturbation detector. Given an input x and the PRN output $T(x)$, they use a Discrete Cosine Transform (DCT) denoted by D to generate features $D(x - T(x))$. They then train a binary classifier B on clean and perturbed data to map the generated features $D(x - T(x))$ to $[0, 1]$, indicating whether it is adversarial or not. If the input is classified as adversarial, their system uses the output of the PRN for prediction, otherwise it uses the prediction of the original network.

A disadvantage of their method is that they train only on a specific type of adversarial perturbation, and any compromise in either the detector or rectifier will leave the model vulnerable. Their evaluation also did not include UAPs generated via SGD nor a defence-aware attacker, indicating that their defence is likely more fragile than reported. PRN is outperformed by the next defence that we will cover.

2.4.4 Selective Feature Regeneration

Borkar et al. propose **Selective Feature Regeneration (SFR)** to defend against UAPs [77]. This is done by first identifying convolutional features within the hidden layers of the DNN that are the most vulnerable to adversarial noise. They then apply their trainable *Feature Regeneration Units (FRU)* to these layers to “regenerate” or correct activations that are the result of adversarial perturbations. SFR is a Type I - denoising defence that acts on the hidden layer activations of the model.

Their FRUs are shallow residual blocks [6] that consist of two stacked 3×3 convolutional layers

sandwiched between several 1×1 convolutional layers and a single skip connection. Let $f_l(x)$ be the activations at convolutional filter l for input x . Define the FRU at convolutional filter l as Φ_l . They train FRUs such that $\Phi_l(f_l(x + \delta)) \approx f_l(x)$ and $\Phi_l(f_l(x)) \approx f_l(x)$. FRUs try to correct or “regenerate” activation values for perturbed images, whilst maintaining the original activation values for non-perturbed images. Ideally, these FRU only modify the activation patterns that make inputs adversarial.

They assess the vulnerability of individual convolutional filters by using the ℓ_1 -norm of their kernel weights. They show that the ℓ_1 -norm of these weights is proportional to the maximum possible activation value of that filter. Thus, they determined that convolutional filters with the highest ℓ_1 -norm weights are the most susceptible to adversarial perturbations. For their experiments, they applied their FRUs to the top 50% most vulnerable convolutional filters. Their results show that SFR outperforms other UAP defences like PRN [75] for recovering model accuracy on adversarial noise UAPs [77].

2.4.5 SentiNet: Detecting Localised Universal Attacks

SentiNet is a UAP defence designed to detect localised universal attacks such as adversarial patch UAPs [67] without modifying the defended model. This is a Type III - detection-based defence. Their defence can be broken down into two main steps. First, SentiNet uses model explainability techniques and object detection to identify and localise the input image region that most highly influences the model prediction. Then second, the extracted region is applied to a set of trusted test inputs to observe the model output. SentiNet compares the behaviour of the model on this synthetic dataset versus its behaviour on trusted inputs to reach a prediction on whether the original input image is affected by an adversarial attack.

The initial extraction for the region of interest is done using a model-interpretability technique Grad-CAM [78], this uses gradients computed in the final layers of a network to calculate the regions of the input that contributed the most to the classification score. In the second part, SentiNet overlays the suspected region on a set of benign test images. These new images are fed back into the network, and the number of adversarial examples are counted. Intuitively,

the higher the number of modified images that can fool the model, the more likely it is that the suspected region is an adversarial attack. For the region of interest, they plot two metrics: the number of images fooled and the average confidence value of the region. They observed that clean and adversarial inputs they have tested on can be separated by a parabolic curve. A linear approximation technique was used to derive the decision boundary. The confidence threshold for their detector is then determined by the margin, i.e. how far the plotted point is from the decision boundary. Overall they are able to demonstrate that SentiNet is capable of detecting adversarial patch UAPs reasonably well. They also show its efficacy against defence-aware attackers, with only a mild degradation in performance.

2.5 Summary

In this chapter, we defined UAPs in the context of computer vision DNNs. We outlined the various threat models that we consider and gave our rationale for each scenario. The threat model is crucial for giving context to the UAP attack evaluation and their later mitigation. We have reviewed the related work and aggregated the existing adversarial attack and defence literature relevant to UAPs. The following are the key takeaways from the literature:

1. Among UAP generation algorithms, SGD is the most potent PK attack, making it the best method for evaluating UAP robustness of models.
2. Hidden layer activations is likely a distinguishing characteristic of UAPs, as UAP attacks GD-UAP [2] and SVA [3] maximise this, whilst UAP defences PRN [75] and SFR [77] attempt to correct this. This indicates that there may be properties of hidden layer activations that can be used to detect UAPs.
3. Defences such as AT improve robustness to adversarial examples, however they suffer considerable drops in accuracy. The existing defence literature has yet to thoroughly address the accuracy-robustness trade-off for UAPs.

Chapter 3

Procedural Noise Attacks

Universal Adversarial Perturbations (UAPs) exhibit interesting visual structures [1] that are not yet fully understood. Existing approaches like those outlined in Section 2.3, primarily craft UAPs using perfect-knowledge (PK) of the model’s learned parameters [1, 79, 3]. What was missing at the time¹ was a method for generating UAPs in a *zero-knowledge (ZK)* manner, i.e. without requiring access to the model’s learned parameters. UAPs are interesting as their universality reveals more generic or class-agnostic features that machine learning algorithms appear to be sensitive to. That is, regardless of the original input, the model’s outputs are affected by small perturbations in the form UAPs. In contrast, input-dependent adversarial perturbations, though less detectable in many cases, can “overfit” and affect only the inputs they were designed for [80]. The visual structures present in UAPs are also not evident in their input-dependent counterparts. Due to the prevalence of these visual patterns in UAPs, we construct procedural noise functions to generate patterns that are visually similar to those of UAPs. Through rigorous experimentation we then demonstrate that our procedural noise can also act as UAPs. As a ZK attack, procedural noise has a lot of appealing traits that make it a good base for efficient, practical, and scalable attacks on computer vision DNNs. Procedural noise is simple to implement, fast to compute, and does not require the additional overhead of building, training, or accessing a DNN to generate adversarial perturbations. The parametrisation of procedural noise is simple and this results in a reduced search space, which

¹Zero-knowledge UAPs based on Fourier Basis functions [4] was published around the same time as our work.

can enable query-efficient ZK attacks. This is useful in large-scale applications like natural-image classification where existing attacks often explore the entire input space, which has very high dimensionality ($\geq 100,000$).

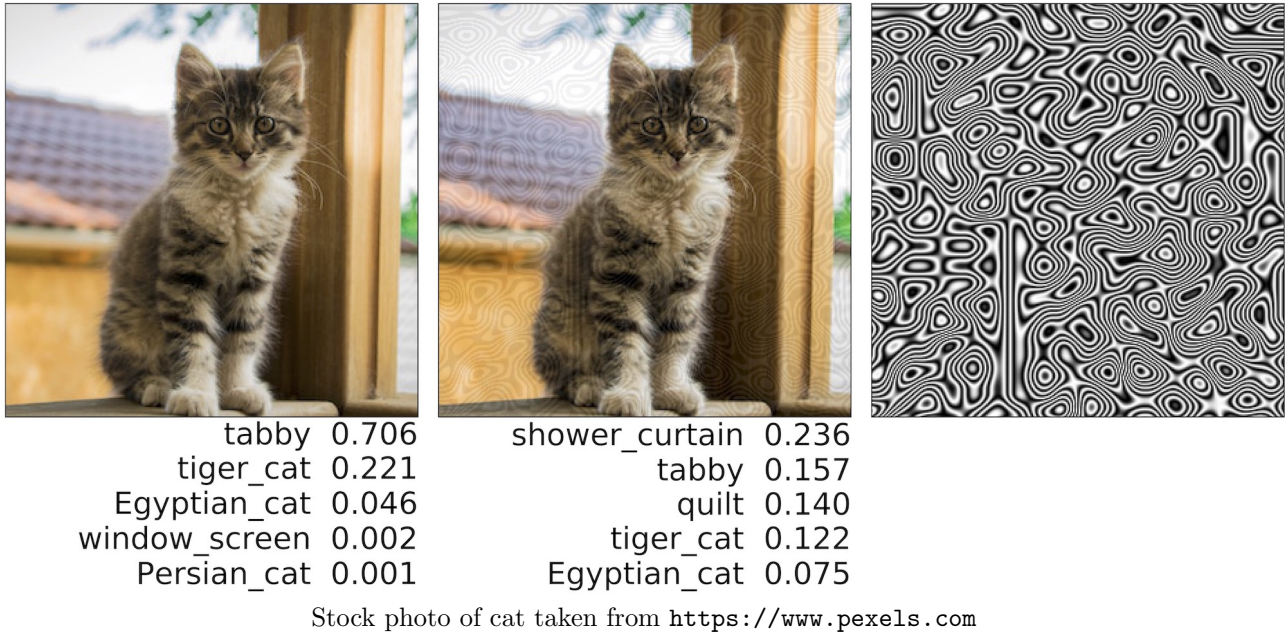


Figure 3.1: Adversarial example generated with a procedural noise function. From left to right: original image, adversarial example, and procedural noise (magnified for visibility). Below the first two images are the classifier’s top 5 output probability labels.

In this chapter, we propose a novel approach for generating UAPs based on the use of procedural noise functions. Such functions are commonly used in computer graphics and designed to be parametrisable, fast, and lightweight [81]. Their primary purpose is to algorithmically generate textures and patterns on the fly. Procedurally generated noise patterns have interesting structures that are visually similar to those in existing universal adversarial perturbations [1, 3]. We empirically demonstrate that DNNs are fragile to procedural noise and these noise act as effective UAPs. Our experimental results on the large-scale *ImageNet* classifiers show that our proposed ZK attacks can fool classifiers on up to 98.3% of input examples. The attack also transfers to the object detection task, showing that it has an obfuscating effect on objects against the YOLOv3 object detector [12]. These results suggest that large-scale indiscriminate ZK attacks against DNN-based machine learning services are not only possible but can be realised at low computational costs.

3.1 Background: Procedural Noise

UAPs have some visual resemblance with patterns generated by *procedural noise functions* as seen in Figure 3.2. Procedural noise functions are procedural techniques used for generating image patterns. The term “procedural” refers to the noise being generated algorithmically. These noise functions are widely used in computer graphics and have numerous applications in film, video game, and graphics production [81] amongst others. They are used to generate textures for the creation of *natural details* to enhance images. Specific examples include textures for glass, wood, marble, and animations such as cloud, fire, and ripple effects. Given these properties, we hypothesise that procedural noise perturbations added to benign images can fool image classification algorithms.

Procedural noise functions are designed to scale to large dimensions, be fast to evaluate, and have low memory footprints. They are also parametrisable, allowing the generation of different noise patterns from the same function. These attributes make them desirable for generating computationally inexpensive perturbations useful for efficient ZK attacks.

3.1.1 Types of Procedural Noise Functions

Procedural noise functions can be classified into three categories: lattice gradient noise, sparse convolution noise, and explicit noise [81]. **Lattice gradient noise** is generated by interpolating random values or gradients at the points of an integer lattice, with Perlin noise as a representative example as it is the most widely used. **Sparse convolution noise** is a sum of randomly positioned and weighted kernels², with Gabor noise as a representative example due to its popularity.

Though the construction of explicit noise is not strictly procedural, it is relevant to computer graphics with Anisotropic and Wavelet noise as examples. Explicit noise is not considered for this study as it induces a large memory cost from its reliance on the storage and retrieval of pre-computed images [81], which limits its feasibility as a scalable attack.

²A kernel in image processing refers to a matrix used for image convolution.

1. **Lattice Gradient Noise.** Perlin noise is described here because of its ease of use, popularity, and simplicity. Perlin noise was developed as a technique to produce natural-looking textures for computer graphics, with its initial application in motion pictures, and it has remained a staple in the industry [81]. Although Perlin noise may not be the most expressive noise function, it has a simple implementation which makes it suitable for inexpensive ZK attacks as it only has a few parameters.

This is a summary of the formal construction of two-dimensional Perlin noise as described in [82, 83, 84]: The value at a point (a, b) is derived in the following way: Let (i, j) define the four lattice points of the lattice square where $i = |a|, |a| + 1$ and $j = |b|, |b| + 1$. The four gradients are given by $g_{ij} = \mathbf{V}[\mathbf{Q}[\mathbf{Q}[i] + j]]$ where pre-computed arrays $\mathbf{Q}$ and $\mathbf{V}$ contain a pseudo-random permutation and pseudo-random unit gradient vectors respectively. The four linear functions $g_{ij} \cdot (a - i, b - j)$ are then bilinearly interpolated by $s(a - |a|)$ and $s(b - |b|)$, where $s(t) = 6t^5 - 15t^4 + 10t^3$. The result is the Perlin noise value $p(a, b)$ for coordinates (a, b) .

The Perlin noise function has several parameters that determine the visual appearance of the noise. The frequencies ν_a, ν_b and number of octaves ω contribute to the most visual change. Frequency corresponds to the rate of change between adjacent pixel values, it affects how visually smooth the image is. The number of octaves refers to the number of additional noise signals added onto the image where the next signal's frequency is double the previous signal's. Increasing this makes the visual appearance more detailed. The resulting gradient noise value at point (a, b) with parameters $\theta_{\text{LGrad}} = \{\nu_a, \nu_b, \omega\}$ becomes

$$S_{\text{LGrad}}(a, b) = \sum_{n=1}^{\omega} p(a \cdot \nu_a 2^{n-1}, b \cdot \nu_b 2^{n-1}).$$

2. **Sparse Convolution Noise.** Gabor noise is described here as it has more accurate spectral control over other procedural noise [85]. Spectral control refers to the ability to control the appearance of noise as measured by its energy along each frequency band. The improved spectral control of Gabor noise is because it can be quickly evaluated at any point in space while being described by only a few parameters such as orientation,

frequency, and bandwidth [85]. Gabor noise is generated by the sparse convolution of the Gabor kernel. The Gabor kernel g for parameters $\{\kappa, \alpha, \nu, \omega\}$ is the product of a circular Gaussian and a harmonic function,

$$g(a, b) = \kappa e^{-\pi\alpha^2(a^2+b^2)} \cos[2\pi\nu(x \cos \omega + y \sin \omega)]$$

where κ and α are the magnitude and inverse width of the Gaussian respectively, and ν and ω are the frequency and orientation of the harmonic respectively [81]. The value $S(a, b)$ at point (a, b) is the sparse convolution noise with a Gabor kernel,

$$S_{\text{SConv}}(a, b) = \sum_{i=1}^N w_i g(a - a_i, b - b_i; \kappa, \alpha, \nu, \omega)$$

where $\{(a_i, b_i)\}_i$ are random points, $\{w_i\}_i$ are random weights, and N is the number of random points [81].

Although the procedural noise functions are already quite expressive in the range of patterns they are able to generate, it still does not possess the visual structures seen in UAPs. These patterns with only the noise function are shown in the top row of Figure 3.2, and on its own, its appearance does not resemble that of UAPs as seen in Figure 3.4. To improve on this design, we augment the procedural noise with colour maps.

3.1.2 Colour Maps

Colour maps are used to create additional variation in the appearance of the image. An example of how the colour map affects the visual appearance can be seen in Figure 3.2. To achieve patterns with many edges like in UAPs, we apply a sine function with a frequency parameter ν_{sine} . This is defined for the noise value q with the function $C(q) = \sin(q \cdot 2\pi\nu_{\text{sine}})$. The periodicity of this sine function creates distinct bands in the image to achieve a high frequency of edges.

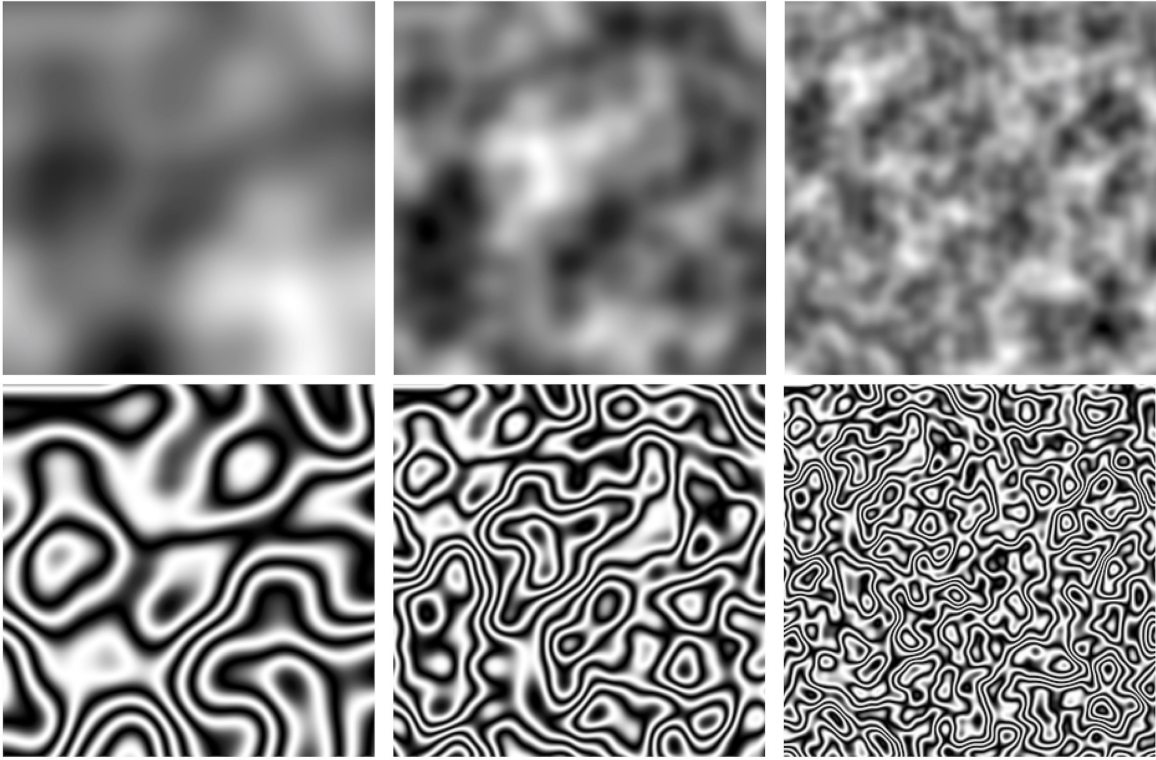


Figure 3.2: Samples of lattice gradient noise with colour maps: (top) greyscale and (middle) greyscale sine function.

Colour maps are versatile in that other transformations aside from sine and greyscale colour mappings can be used. Different variations and gradients of colours can be applied depending on the application, and various functions can be chosen depending on the desired visual appearance. For our purposes, the sine function we chose is sufficient in replicating the visual appearance of UAPs.

3.1.3 Procedural Noise Attack

Here we describe the noise generating functions used in the actual attacks, one each for lattice gradient noise and sparse convolution noise. We tailor their parametrisation so that they achieve a similar appearance with the structured patterns of UAPs.

Perlin Noise. To achieve more distinct visual patterns, we use a sine colour map with an additional frequency parameter ϕ_{sine} . This colour map for the noise value p is defined by $C(p) = \sin(p \cdot 2\pi\phi_{\text{sine}})$. The periodicity of the sine function creates distinct bands in the image

to achieve a high frequency of edges. The resulting noise generating function G_{per} at point (x, y) is defined as the composition of our Perlin noise function and the sine colour map,

$$G_{\text{per}}(x, y) = C(S_{\text{per}}(x, y)) = \sin((S_{\text{per}}(x, y) \cdot 2\pi\phi_{\text{sine}}))$$

with combined parameters $\delta_{\text{per}} = \{\lambda_x, \lambda_y, \phi_{\text{sine}}, \Omega\}$.

Gabor Noise. Gabor noise is an expressive noise function and will have a large number of dimensions if each random point is assigned different kernel parameters. To simplify the implementation, we use the same parameters and weights for each random point (x_i, y_i) . This results in noise patterns that have a uniform texture.

We add an additional discrete parameter ξ to control the isotropy of the Gabor noise. Having $\xi = 1$ results in anisotropic noise, which is oriented in one direction. Progressively larger ξ makes it more uniform in all directions. Implementing these changes gives an updated formulation

$$S_{\text{gab}}(x, y) = \frac{1}{\xi} \sum_{n=1}^{\xi} \sum_i g(x - x_i, y - y_i; \sigma, \lambda, \omega + \frac{n\pi}{\xi}).$$

To achieve high-frequency patterns and remove unwanted low-contrast oscillations, we normalise the variance spectrum of the Gabor noise using the algorithm described by [86]. This results in min-max oscillations to give the distinct structured patterns as seen in UAPs. Note that we refer specifically to the *frequency of edges*, i.e. how often edges appear one after the other on an image. This is different from the classical frequency used in spectral analysis. For simplicity we use “low-frequency” or “high-frequency” when referring to patterns with low or high edge frequency in the image. The final noise generating function G_{per} with a normalised variance spectrum has parameters $\delta_{\text{gab}} = \{\sigma, \lambda, \omega, \xi\}$.

As a result of this construction, Figure 3.3 visualises examples of perturbations generated from the proposed procedural noise functions. Comparing this with an existing UAP attack in Figure 3.4, we see the visual similarities in the structure of the noise patterns in Figures 3.3 and 3.4.

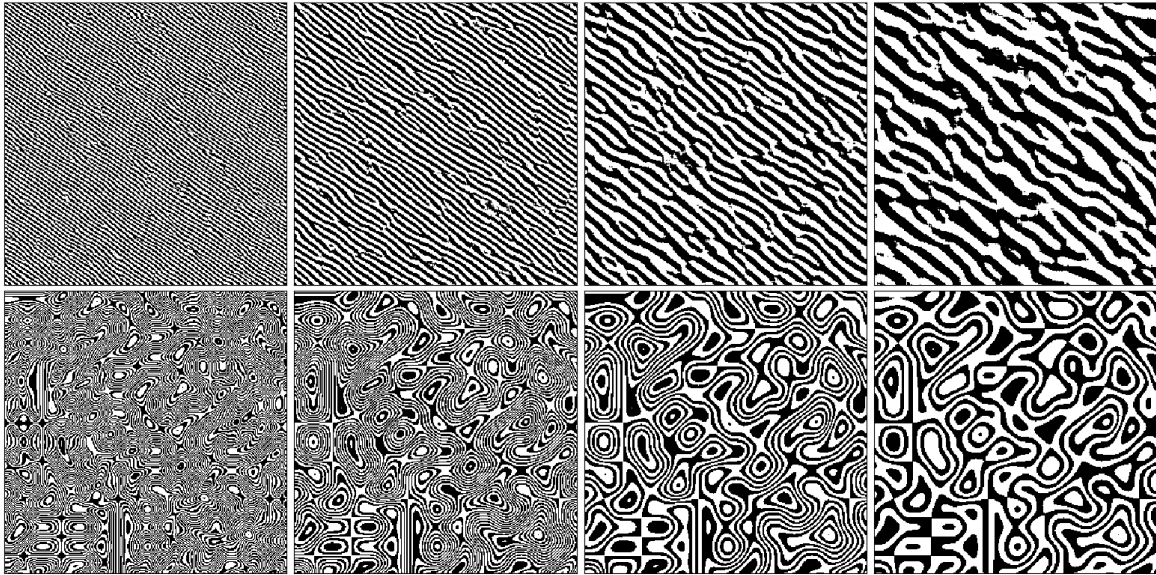


Figure 3.3: Procedural noise patterns with (top) Gabor noise and (bottom) Perlin noise, both with decreasing frequency from left to right.

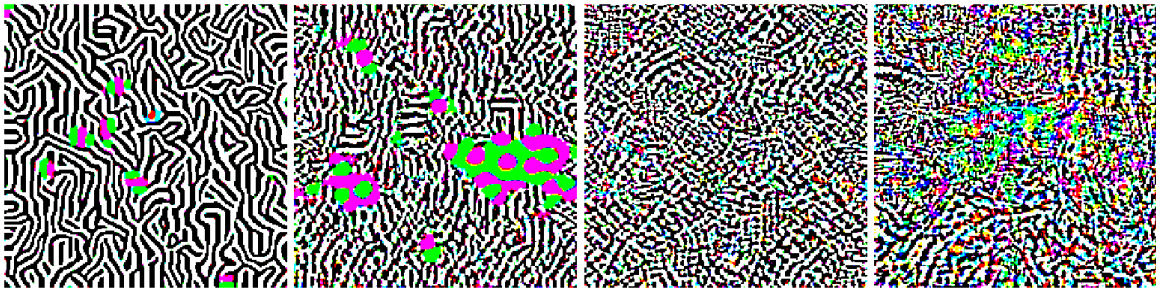


Figure 3.4: UAPs generated for VGG-19 using the Singular Vector Attack [3].

3.2 Vulnerability to Procedural Noise

In this section, we evaluate the vulnerability of five different popular DNN architectures to procedural noise for the ImageNet benchmark.

3.2.1 Experimental Setup

The data used are 5,000 random images from the validation set of the ILSVRC 2012 ImageNet classification task [11] or ImageNet for short. It is a widely used object recognition benchmark

with images taken from various search engines and manually labelled to 1,000 distinct object categories where each image is assigned one ground truth label.

Models. We use four popular distinct DNN architectures pre-trained on ImageNet: VGG-19 [7], ResNet-50 [6], Inception v3 [87], and Inception ResNet v2 [88]. We abbreviate Inception ResNet v2 to **IRv2**. Inception v3 and Inception ResNet v2 take input images with dimensions $299 \times 299 \times 3$ while the remaining two networks take images with dimensions $224 \times 224 \times 3$.

We also take an Inception ResNet v2 model trained with **Ensemble Adversarial Training (EAT)**: Tramer et al. [72] fine-tuned the IRv2 network by applying EAT, making it more robust to gradient-based attacks. In contrast to conventional AT, EAT augments the dataset of a model with adversarial examples generated from several other models. We refer to this new model trained with EAT as **IRv2_{ens}**. It has the same architecture, but different weights when compared to the first IRv2.

Perturbations. The Perlin noise parameters $\lambda_x, \lambda_y, \phi_{\text{ sine}}$ and Gabor noise parameters σ, λ are positive and bounded above by the image’s side length d . Increasing the parameters beyond this will have no impact on the resulting image as it gets clipped by the image dimension. The isotropy $\xi \in [1, 12]$ and number of octaves $\Omega \in [1, 4]$ are discrete, and evaluations show negligible change in the image for $\xi > 12$ and $\Omega > 4$. The range on the angle $\omega \in [0, 2\pi]$ covers all directions. We fix the kernel size and number of random points for Gabor noise so that the Gabor kernels always populate the entire image. The parameters δ_{gab} of the Gabor noise have the greater influence on the resulting visual appearance of the noise pattern.

To provide a baseline, we also test the models against uniform random noise perturbations: $\text{sgn}(r) \cdot \varepsilon$ where $r \in \mathcal{U}(-1, 1)^{d \times d \times 3}$, d is the image’s side length, and ε is the ℓ_∞ -norm constraint on the perturbation. This is an ℓ_∞ -optimised uniform random noise, and it is reasonable to say that attacks that significantly outperform this baseline are non-trivial.

Threat Model. For adversarial capabilities, we use the ℓ_∞ -norm threat model as it is the most used in the UAP literature and is straightforward to impose for procedural noise UAPs. We refer the reader to Section 2.2.2 for a more thorough discussion on this choice.

The adversary for our procedural noise UAP only requires ZK of the model. This means no access to the internal parameters of the victim model, whilst having the ability to query the model and read the output labels. For the adversarial goals, we first measure how well our proposed procedural noise attack works as a universal attack, then as an input-dependent attack. To measure how well a UAP generalises across a dataset, we use the Universal Evasion Rate (UER) as defined in Equation 2.2. UER is the proportion of the dataset that the model makes errors on when the UAP is applied.

We introduce two additional metrics to improve the analysis of our results. To measure the model’s sensitivity against the perturbations for each input, we define the model’s **average sensitivity** on an input x over a set of perturbations $\delta \in \mathcal{P}$ as

$$\frac{|\{\delta \in \mathcal{P} : \arg \max f(x + \delta) \neq \tau(x)\}|}{|\mathcal{P}|} \quad (3.1)$$

An input x with a high sensitivity means that any perturbation leads to model error, meaning that the model is unstable for its prediction on x . Whereas a low sensitivity means that the model is not likely to make errors on x in the presence of perturbations. This will help determine the proportion of the dataset on which the model is more vulnerable.

Finally, the **input-dependent evasion rate** is given by

$$\frac{|\{x \in X : \exists \delta \in \mathcal{P} \text{ such that } \arg \max f(x + \delta) \neq \tau(x)\}|}{|X|} \quad (3.2)$$

This measures how many inputs in the dataset can be evaded with perturbations from the set $\mathcal{P}$. For example, if the input specific evasion rate is high, then for most inputs x we can find some adversarial perturbation δ in the set $\mathcal{P}$ such that the model makes an error on $x + \delta$.

Experiment. We evaluate our procedural noise perturbations on 5,000 random images from the validation set. The perturbations generated are from 1,000 Gabor noise, 1,000 Perlin noise, and 10,000 uniform random perturbations. 1,000 queries for the procedural noise functions was sufficient for our results and its search space only has four bounded parameters. We use an ℓ_∞ -norm constraint of $\varepsilon = 16$. The pixels of the perturbations are first clipped to $[-\varepsilon, \varepsilon]$

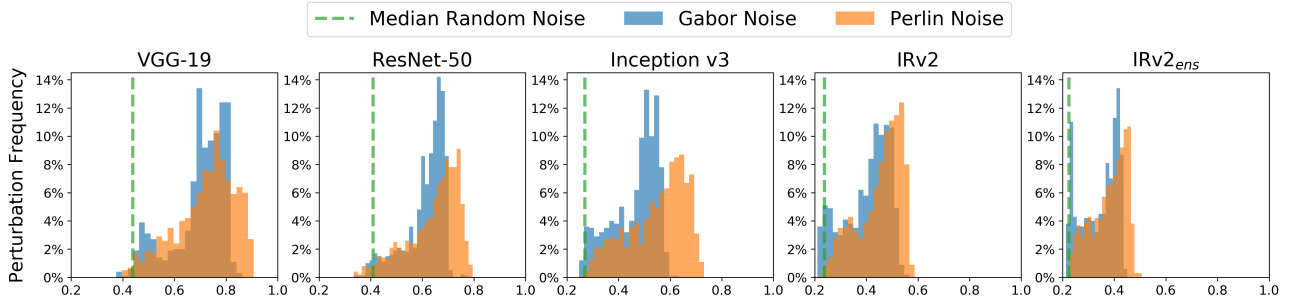


Figure 3.5: Universal Evasion Rate

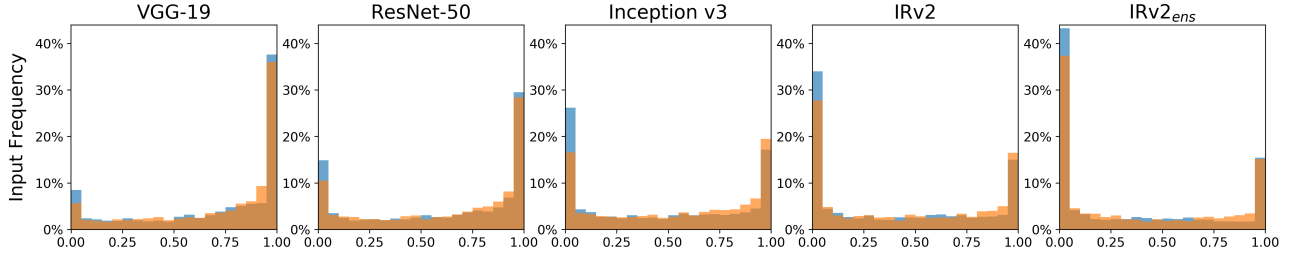


Figure 3.6: Average Sensitivity

Figure 3.7: Histogram of (a) UER over all perturbations and (b) average sensitivity over all inputs across all models. For example, VGG-19 in (a) shows that about 12% of the Gabor noise perturbations have a UER of approximately 0.8.

and the resulting adversarial example’s pixels are clipped to the image space $[0, 255]$. This is an untargeted ZK attack with exactly 1,000 queries per image for procedural noise. Note that the adversary has no knowledge of the target model and only requires the top label from the model’s output. Results are reported in Figure 3.7 and discussed in the following section.

3.2.2 Effectiveness of Procedural Noise

The procedural noise perturbations is both universal and transferable across models. The results in Figure 3.7 show that the models in order from least to most robust against the perturbations are: VGG-19, ResNet-50, Inception v3, IRv2, and then IRv2_{ens}. This is not surprising, as the generalization error for these models appear in the same order, with larger generalisation error indicating a less robust model. IRv2_{ens} was hardened against adversarial examples through EAT, so it was expected to be less affected. However, the EAT did not fully mitigate the impact of the procedural noise.

Both Gabor and Perlin noise have significantly higher UER than random noise. In Figure 3.5,

we represent random noise using the median (which, in this case, is very close to the mean) of its universal evasion rate over all 10,000 perturbations, as the variance is very small (less than 10^{-5}) for each model. Procedural noise functions create a distribution whose modes have high UER. Table 3.1 shows an example of this on Inception v3, where more than half the Perlin noise perturbations achieve evasion on more than 57% of the dataset. Between the two procedural noises, Perlin noise is a stronger UAP, though not by a large margin. In Figure 3.5, the distribution of results for Gabor noise is bimodal, especially with the IRv2 classifiers. This bimodal aspect shows that there are two large disjoint subsets of Gabor noise: one with high UER and another with low UER. This was not as prominent for Perlin noise.

Table 3.1: UER (in %) by percentile for random and procedural noise on Inception v3.

Percentile	Random	Gabor	Perlin
min	26.0	24.7	26.4
25th	26.8	39.9	47.2
50th	27.0	49.1	57.7
75th	27.1	53.3	63.7
max	27.9	61.8	73.2

Best Parameters. Given the distribution of UER in Figure 3.5, it is interesting to observe which parameters of the noise functions contribute most to the evasion rate. Because there are only four parameters for each procedural noise function, the linear correlations between these parameters and the UER give a general overview of their influence.

For Gabor noise, the parameters σ , ω , and ξ have low correlation (≤ 0.16) with UER. The remaining parameter λ however has a high correlation (≥ 0.52) with the UER across all classifiers; this peaks at 0.90 for IRv2_{ens}. λ corresponds to the wavelength of the harmonic kernel. Visually, lower values of λ indicate thinner and more frequent bands, which is why λ can also be thought of as an inverse frequency. The correlations suggest that low-frequency Gabor noise patterns correlate with high UER, and this appears to be more pronounced as we move to the more robust models.

For Perlin noise, the parameters λ_x and λ_y have low correlation (≤ 0.25) with UER. The UER have a moderate negative correlation with the number of octaves Ω and have a moderately-high

positive correlation with ϕ_{sine} for the non-adversarially trained models. This suggests that high-frequency Perlin noise patterns correlate with high universal evasion. The number of octaves indicate the amount of detail or the number of curvatures within the image, so a slight negative correlation indicates that some patterns that achieve high evasion have fewer curvatures and details. Visually, the frequency of the sine function ϕ_{sine} can be thought of as the thinness of the bands in the image. This is the opposite of the λ in Gabor noise, so we have a similar result where IRv2_{ens} is more susceptible to low-frequency patterns.

Low-frequency patterns are also more effective on IRv2_{ens} because this model was adversarially trained on gradient-based attacks, which we hypothesise generated mostly *high-frequency* perturbations. The frequency appears to be an important factor in determining the strength of the UAP even when Gabor and Perlin noise have opposite correlations between their frequency and their universal evasion. This difference shows that these noise patterns have notably different characteristics. Figure 3.8 shows the correlation matrices between the procedural noise parameters with the corresponding UER on the various ImageNet classifiers. These correlations quantify the effects of each parameter and the transferability of UAPs across models.

Cross-model Universality. The UAPs with high universal evasion on one model often have high universal evasion on the other models. For example, a Perlin noise perturbation has 86.7%, 77.4%, 73.2%, 58.2%, and 45.9% UER on VGG-19, ResNet-50, Inception v3, IRv2, and IRv2_{ens} respectively, which is the top percentile in UER amongst the procedural noise. The correlation of the universal evasion rates across models is high (≥ 0.80), which shows the transferability of our UAPs across models. For Gabor noise, there is a high correlation across all models. For Perlin noise, there appears to be a noticeable difference between IRv2_{ens} and the remaining four models. The strongest Perlin noise on the undefended models have high-frequency patterns that the adversarial training partially mitigated. Overall, the results show that procedural noise perturbations act as UAPs across all models. This is the first ZK generation of cross-model UAPs, and the procedural noise allows us to draw perturbations from a distribution with high average universal evasion. We now look at the results from an input-dependent perspective.

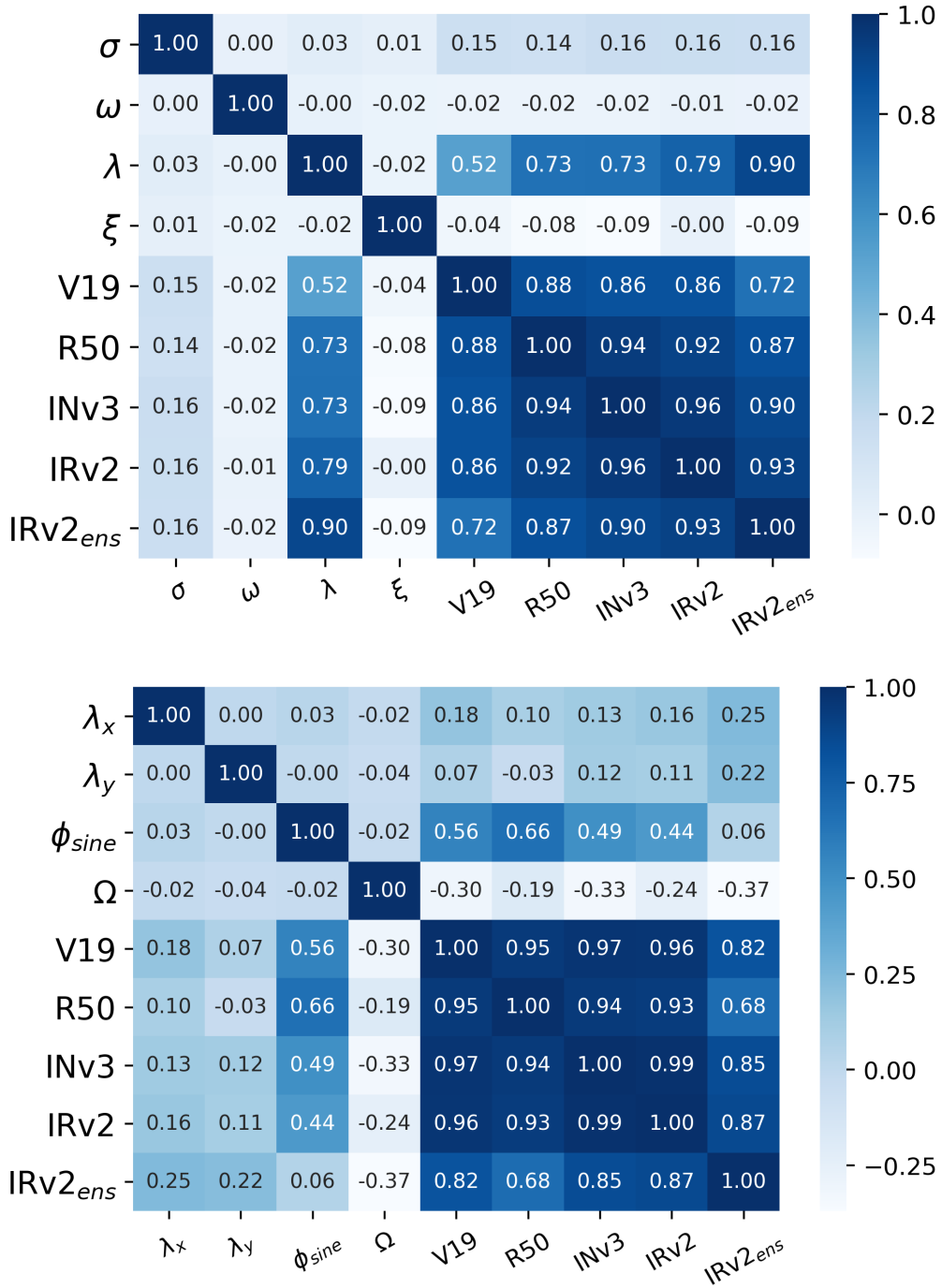


Figure 3.8: Correlations between UER with (top) Gabor noise parameters ($\delta_{\text{gab}} = \{\sigma, \omega, \lambda, \xi\}$) and (bottom) Perlin noise parameters ($\delta_{\text{per}} = \{\lambda_x, \lambda_y, \phi_{\text{sine}}, \omega\}$).

3.2.3 Model Sensitivity on Inputs

The model's sensitivity could vary across the input dataset, i.e. the model's predictions are stable for some inputs while others are more susceptible to small perturbations. We measure

this with the average sensitivity of single inputs over all perturbations. Results in Figure 3.6 show that the average sensitivity of the dataset is bimodal for all models and both procedural noise functions. There are two distinct subsets of the data: one on the right that is very sensitive and the other on the left that is very insensitive to the perturbations. The remaining data points are somewhat uniformly spread in the middle. The number of points on the left peak of the histogram in Figure 3.6 is larger for the most robust models. Similarly to Figure 3.5, this progression indicates that the order of most to least sensitive models align with the most to least robust models. This manifests as a tall peak on the left and a short peak on the right.

When comparing the average sensitivity of inputs between the two procedural noise functions on the same models, the correlations range between 0.89-0.92, which shows that both procedural noise perturbations affect very similar groups of inputs for each model. The correlation between the average sensitivities for each input across the Inception models is at least 0.79, which suggests that these models are sensitive to procedural noise on similar inputs. This is less so between ResNet-50 and VGG-19 whose correlations with the other models range from 0.56-0.81.

Input-dependent Evasion. We consider the case when our untargeted ZK attack is used as an input-dependent attack. This is the traditional metric that adversarial attacks are evaluated with where adversarial perturbations are generated for each input, hence being “input-dependent”, with the goal of fooling the model on as many inputs as possible [23, 24, 25]. This is in contrast to universal attacks where the adversary crafts a *single* perturbation that can fool the model on as many inputs as possible. UAPs are more practical for the attacker if applying unique adversarial perturbations on each input is too computationally expensive.

Getting into the results, we first note that the random noise is optimised for the ℓ_∞ -norm. We draw the random noise perturbation from $\{-\varepsilon, \varepsilon\}^{d \times d \times 3}$. Thus, for each pixel the added noise is either $-\varepsilon$ or ε , rather than drawing from the continuous domain $(-\varepsilon, \varepsilon)$. It is reasonable to think that a larger perturbation is more likely to cause evasion.

Table 3.2 shows that VGG-19 and ResNet-50 are particularly fragile as even random noise greatly degrades their performance. Although the Inception models are more robust, both procedural noise perturbations still achieve more than 72% evasion on all of them. Although

Table 3.2: Input-dependent evasion rate (in %) for random and procedural noise perturbations. Original refers to the top 1 error on the unaltered original images. Strongest attack on each classifier is highlighted.

Classifier	Original	Random	Gabor	Perlin
VGG-19	29.4	57.1	97.7	98.3
ResNet-50	25.9	55.7	96.2	96.3
Inception v3	22.3	46.8	89.2	93.6
IRv2	20.1	38.7	81.1	87.0
IRv2 _{ens}	20.1	37.5	72.7	79.4

Table 3.3: **Top 5** input-dependent evasion rate (in %) for random and procedural noise perturbations. Original refers to the top 5 error on the unaltered original images. The strongest attack on each classifier is highlighted.

Classifier	Original	Random	Gabor	Perlin
VGG-19	9.5	28.0	90.2	92.6
ResNet-50	8.4	29.5	82.5	85.3
Inception v3	6.3	18.0	66.9	79.5
IRv2	4.5	13.2	56.3	66.4
IRv2 _{ens}	5.1	12.1	44.1	53.6

ensemble adversarial training improved the robustness of IRv2, it still does not mitigate the impact of the procedural noise attack. The idea of ensemble adversarial training was to decouple the generation of the adversarial training set from the original model, but this was limited since it was only done for gradient-based attacks. We argue that defences should be more input-agnostic to avoid having to train against all types of attacks.

In Table 3.3, we show the top 5 input-dependent evasion rate where the true class label is outside the top 5 class labels in the perturbed image $x + s$. The top 5 error is often used alongside the top 1 error to measure the performance of classifiers on ImageNet. Top 5 evasion is more difficult than top 1 as the confidence in the true label has to be degraded sufficiently enough for it to be below five other class labels. Despite that, Perlin noise is able to achieve top 5 evasion on more than half the inputs for all tested models. We see that our procedural noise remains a strong attack even when considering the top 5 error metric.

Label Analysis. The procedural noise perturbations were not designed to be a targeted attack, but intuitively, the same universal perturbation would cause misclassification towards

class labels that have visually similar textures to the procedural noise pattern. We find that this holds true for only a few of the procedural noise UAPs.

For a given procedural noise perturbation, we define its *top label* to be the class label it causes the most misclassification towards. When looking at procedural noise UAPs across all models, about 90% of Gabor noise perturbations have their top label apply to at most 9% of the inputs. For Perlin noise, about 80% of its perturbations have their top label on at most 10% of the input. There are however a few outliers that have their top label appear above 10% of the inputs. For example, on Inception v3, the top Gabor noise with 61.8% universal evasion has “window screen” as its top label and it applies for 37.8% of the evaded inputs. In contrast, another Gabor noise perturbation with 58.9% universal evasion has “quilt” as its top label, but it only applies for 6.0% of the evaded inputs. As a consequence, it is still possible to use procedural noise to create universal targeted UAPs aimed at specific class labels like “window screen” or “brain coral”. However the class labels we can target is dependent on the procedural noise and the overall success of universal targeted attacks may be limited, as it is more difficult to make a perturbation both universal and targeted.

Perlin noise has a relatively large amount of “brain coral” classifications, with other labels such as “maze” and “shower curtain” also appearing frequently in the top five most classified labels per classifier. For Gabor noise, there was no label that consistently appeared at the top across all models. These results indicate that Perlin noise has a larger bias towards certain classes, while Gabor noise is more indiscriminate. That the model is more vulnerable to these particular class labels may indicate that it is more sensitive to lower level textural features, as these class

3.2.4 Discussion

Adversarial examples exploit the model’s sensitivity, causing large changes in the model’s output by applying specific small changes to the input. More specifically, recent work has shown adversarial examples exploit “non-robust” features that the model learns but that are incomprehensible to humans [41]. It is likely that UAPs may be exploiting “universal” non-robust features that the DNN has learned.

Textures. Previous results have shown ImageNet-trained DNNs to be more reliant on textures rather than shapes [89]. Though not explicitly shown in the later Inception architectures (Inception v3, IRv2), these are still likely to have a strong texture-bias due to similarities in the training. The texture bias however does not fully explain why small, and sometimes imperceptible, changes to the texture can drastically alter the classification output. We attribute adversarial perturbations more to the sensitivity of the model. Moreover, we test our attack against an object detection model in Section 3.5, and show that it also degrades the performance of models that have a spatial component in their learning task.

Generalizability of Procedural Noise. From the label analysis of procedural noise UAPs, we observe that most perturbations with high universal evasion do not have a strong bias towards any particular class label—no particular class is targeted more than 10% for over 80% of the procedural noise UAPs. This suggests that UAPs leverage more generic low-level features that the model learns, which would explain their *universality* and indiscriminate behaviour in causing misclassification.

Amongst UAP attacks in the PK scenario, our procedural noise has the closest visual appearance to perturbations from the Singular Vector Attack (SVA) by [3] as seen when comparing Figures 3.3 and 3.4. They generate UAPs targeting specific layers of DNNs as discussed in Section 2.3, and found that targeting earlier layers of the network generated more successful UAPs. The patterns obtained for these earlier layers share a visual appearance with procedural noise. These layers also correspond to the low-level features learned by the network. Other evidence also suggests that procedural noise exploits low-level features, as feature visualization of earlier layers in neural networks share the same visual appearance with some procedural noise patterns. Convolutional layers induce a prior on DNNs to learn local spatial information [37], and DNNs trained on natural-image datasets learn convolution filters that are similar in appearance to Gabor kernels and colour blobs [90, 91]. Gabor noise appears to be a simple collection of low-level features whereas Perlin noise seems to be a more complex mixture of low-level features. This difference in the complexity of their visual appearance may explain why Perlin noise is a stronger attack than Gabor noise.

Procedural noise attacks transfer with high correlation across the models most likely because they share the same training set (ImageNet), learning algorithms (e.g. backpropagation), and have similar components in their architectures (e.g. convolutional layers). This increases the likelihood that they share input-agnostic vulnerabilities. In this way, our results appear to support the idea that DNNs are *sensitive* to aggregations of low-level features.

Security Implications. In transfer learning, a model trained on one task is re-purposed as an initialization or fixed feature extractor for another similar or related task. When used as a feature extractor, the initial layers are often frozen to preserve the low-level features learned. The subsequent layers, closer to the output, are then re-trained for the new task [90]. Hidden layers from models pre-trained on the ImageNet dataset are often re-used for other natural-image classification tasks [92]. This makes it a notable target as vulnerabilities that exploit low-level features encoded in the earlier layers carry over to the new models.

Transfer learning has many benefits as training entire models from scratch for large-scale tasks like natural-image classification can be costly both computationally in training and in terms of gathering the required data. However, this creates a systemic threat, as subsequent models will also be vulnerable to attacks on low-level features like procedural noise. Precisely characterizing the extent of the vulnerability of re-purposed models to the same attack is an interesting direction for future work.

Procedural noise is an accessible and inexpensive way for generating UAPs against existing image classifiers. In our experiments the Gabor and Perlin noise functions modified only four parameters, and each parameter was bounded in a closed interval. Drawing from this space of perturbations has generated UAPs with high universal evasion rates. Attackers can take advantage of the small search space by using procedural noise to craft query-efficient untargeted ZK attacks as we will show in Section 3.4.

Strengths & Limitations. Procedural noise is one of the first ZK generation of UAPs. Other existing ZK attacks often optimise for input-dependent evasion, and other existing UAP generation methods are designed for PK or LK scenarios. Whilst procedural noise is also a generative model, it differs from other generative models like Bayesian networks, Generative

Adversarial Networks (GANs), or Variational Autoencoders (VAEs) as it does not require the additional overhead of building and training these generative models—which often requires more resources and stronger adversaries to execute successfully.

Comparing procedural noise attacks with existing ZK attacks is not straightforward, as procedural noise naturally has high universal evasion rates. This gives procedural noise an advantage in that, despite having no access to the target model, randomly drawn procedural noise patterns are likely to have high universal evasion. The search space for our procedural noise has only four dimensions, whereas most attacks are designed for the whole input space of hundreds of thousands of dimensions. However, this does come at a cost, as procedural noise functions do not capture adversarial perturbations outside their codomain. Other attacks that explore the whole input space are able to take full advantage of a stronger adversary (i.e. PK or LK setting) or larger query limits. We explore this trade-off further with input-dependent ZK attacks in Section 3.4.

Another limitation of this experiment was the use of an ℓ_∞ -norm constraint. Although it is often used in the literature as a proxy for human perception, ℓ_p -norms have limitations [55] e.g., low-frequency patterns appear to be more visible than high-frequency patterns for the same ℓ_p -norm. This frequency could be used as an additional constraint and developing a more reliable proxy for human perception remains an interesting avenue for future work.

Summary. Procedural noise attacks specialise as untargeted ZK perturbations with naturally high universal evasion. Even when parameters are chosen at random, procedural noise perturbations act as effective UAPs against DNNs. Also, when considering input-dependent evasion, procedural noise UAPs fool each model on more than 72% of the dataset. Offensively, this has applications as a large-scale indiscriminate attack and, defensively, as a standard test on machine learning services.

To be able to deploy procedural noise attacks in practice, we require more optimal ways for choosing noise function parameters. As our procedural noise attack operates under the ZK threat model, we require a zero-knowledge optimisation technique.

3.3 Background: Bayesian Optimization

In this section, we present the background knowledge needed to understand Bayesian optimization. Bayesian optimization is a sequential model-based optimization algorithm primarily used to find optimal parameters θ for black-box objective functions using a minimal number of queries [93]. This is relevant for creating query-efficient ZK attacks. Bayesian optimisation has proven to be effective in solving various problems such as hyperparameter tuning, reinforcement learning, and combinatorial optimization [94].

Consider a set of functions parameterised by θ . Let $\mathcal{D}$ denote the observed data. We treat θ as a latent variable with a prior distribution $p(\theta)$, this contains our prior beliefs on the values of θ . Given the observations $\mathcal{D}$ and a likelihood $p(\mathcal{D} \mid \theta)$ we can compute the posterior $p(\theta \mid \mathcal{D})$ using Bayes' rule.

$$p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta)p(\theta)}{p(\mathcal{D})}.$$

The posterior distribution is our updated belief on θ given the observed data $\mathcal{D}$ [95, 94]. In Bayesian optimization this model is updated to encode the new beliefs over the parameters as more observations are made from each query, with the goal of maximizing an objective function g . This algorithm is query efficient as it uses the information provided by past queries while also considering the uncertainty of the model.

Bayesian optimization consists of two components, first a probabilistic surrogate model, typically a **Gaussian Process (GP)**, and, second, an acquisition function that guides its queries. GP regression is used to update the beliefs on the parameters with respect to the objective function after each query, and the acquisition function is used to select the next “best” input to query the target classifier. The algorithm stops once the best objective function value ceases to improve significantly or when it reaches its query budget [94].

3.3.1 Gaussian Processes

The first component in Bayesian optimization is a probabilistic surrogate model for our objective function. A GP is the generalization of Gaussian distributions to a distribution over functions [95]. We use GPs as they induce a posterior distribution over the objective function that is analytically tractable. This allows us to update our beliefs of what the objective function looks like after each iteration [93].

A Gaussian Process $\mathcal{GP}(m, k)$ is fully described by a prior mean function $m : \mathcal{X} \rightarrow \mathbb{R}$ and positive-definite kernel or covariance function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$. We describe GP regression to understand how our GP is updated given more observations. Let $\mathbf{X} = \mathbf{x}_{1:n}$ be a collection of n points and define variables $f_i = f(\mathbf{x}_i)$ and $\mathbf{y}_{1:n}$ to be unknown function values and noisy observations respectively. We assume that $\mathbf{f} = f_{1:n}$ are jointly Gaussian and $\mathbf{y} = y_{1:n}$ follow a normal distribution given $\mathbf{f}$. The following expressions give the GP prior and Gaussian likelihood respectively

$$p(f \mid \mathbf{X}) = \mathcal{N}(m(\mathbf{X}), \mathbf{K})$$

$$p(y \mid f, \mathbf{X}) = \mathcal{N}(f(\mathbf{X}), \sigma^2 \mathbf{I})$$

where $\mathcal{N}$ denotes a normal distribution. Elements of the mean and covariance matrix are given by $m_i = m(\mathbf{x}_i)$ and $K_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$. Given observations $\{\mathbf{X}, \mathbf{y}\}$ and an arbitrary point $\mathbf{x}$, the updated posterior mean and covariance on the n -th query are given by

$$m_n(\mathbf{x}) = m(\mathbf{x}) - (\mathbf{K} + \sigma^2 \mathbf{I})^{-1}(\mathbf{y} - m(\mathbf{X}))$$

$$k_n(\mathbf{x}, \mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - k(\mathbf{x}, \mathbf{X})(\mathbf{K} + \sigma^2 \mathbf{I})^{-1}k(\mathbf{X}, \mathbf{x}).$$

The mean function is taken to be zero $m \equiv 0$ to simplify evaluation and since no prior knowledge can be incorporated into the mean function [94]. The ability of GPs to model a rich distribution of functions rests on its kernel function which controls important properties of the function distribution such as differentiability, periodicity, and amplitude [95, 94]. Any prior knowledge of the target function is encoded in the kernel's hyperparameters, but in ZK settings a more

general kernel function such as the the Matérn 5/2 kernel is suitable [93].

$$k_{5/2}(\mathbf{x}, \mathbf{x}') = \left(1 + \frac{\sqrt{5}r}{l} + \frac{5r^2}{3l^2}\right) \exp\left(-\frac{\sqrt{5}r}{l}\right)$$

Where $r = \|\mathbf{x} - \mathbf{x}'\|$ and l is the length-scale parameter. This results in twice-differentiable functions, an assumption that corresponds to those made in popular black-box optimization algorithms like quasi-Newton methods [93].

3.3.2 Acquisition Functions

The second component in Bayesian optimization is an acquisition function that describes how optimal a query is. Intuitively, the acquisition function evaluates the utility of candidate points for the next evaluation, and it is normally defined so that high acquisition corresponds to potentially optimal values of the objective function [96].

The acquisition function has to balance the trade-off between exploration and exploitation. Exploration seeks high variance regions, i.e. those where the uncertainty about the value of the objective function is high. Exploitation seeks places where the uncertainty and estimated mean of the objective function are low [94], i.e. the model is confident about the estimated value of the objective function. Too little exploration can get it trapped at local extrema, while too much exploration does not fully utilise past observations.

The aim is to maximise an objective function g and choose a general purpose acquisition function considering the ZK scenario. The two most popular choices are the *Expected Improvement* (EI) and Gaussian Process *Upper Confidence Bound* (UCB) [94]. First define $\mu(\mathbf{x})$ and $\sigma^2(\mathbf{x})$ as the predictive mean and variance of $g(\mathbf{x})$ respectively. Let $\gamma(\mathbf{x}) = \frac{g(\mathbf{x}_{\text{best}}) - \mu(\mathbf{x})}{\sigma(\mathbf{x})}$. The acquisition functions are given by

$$\alpha_{\text{EI}}(\mathbf{x}) = \sigma(\mathbf{x})(\gamma(\mathbf{x})\Phi(\gamma(\mathbf{x})) + \mathcal{N}(\gamma(\mathbf{x}) \mid 0, 1))$$

$$\alpha_{\text{UCB}}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}), \quad \kappa > 0$$

where Φ is the normal cumulative distribution function. EI and UCB have both been shown to be effective and data-efficient in real black-box optimization problems [93]. However, most studies have found that EI converges near-optimally, requires no tuning of its own parameters, and is better-behaved than UCB in the general case [96, 94, 93]. This makes EI the best candidate for a acquisition function.

3.4 Efficient Zero-knowledge Attack

In the previous Section 3.2, we have shown that computer vision DNNs are vulnerable to procedural noise adversarial perturbations, another advantage our procedural noise functions bring is their low-dimensional search space. This enables the use of query-efficient black-box optimization techniques that otherwise do not scale well to high-dimensional problems. In this section, we compare several black-box optimization techniques for both input-dependent and universal attacks and show that Bayesian optimization is an efficient method for choosing parameters in such ZK attacks.

3.4.1 Universal Zero-knowledge Attack

For a universal ZK attack using procedural noise, the goal is to find the optimal parameters δ^* for the procedural noise generating function G so that the universal evasion rate of perturbation $G(\delta^*)$ generalises to unknown inputs X_{val} . The attacker has a smaller set of inputs X_{train} and optimises their perturbation δ^* for this dataset. The performance of the attack is measured by its universal evasion rate over the validation set X_{val} . In a practical setting, this is where the attacker optimises their procedural noise UAP over a small dataset, then injects that perturbation to other inputs—with the goal of causing as many misclassifications as possible.

To demonstrate how a ZK attack can be applied in practice, we use the Inception v3 model, ℓ_∞ -norm $\varepsilon = 16$, and the same 5,000 data points tested in Section 3.2 as X_{val} . X_{train} are points from the ILSVRC2012 validation set not in X_{val} . We test for training set sizes of 50, 125, 250,

and 500, which corresponds to 1%, 2.5%, 5%, and 10% of the validation set size.

We compare Bayesian optimization with Limited-memory BFGS (L-BFGS) [97], a quasi-Newton optimization algorithm that is often used in black-box optimization and machine learning. As the procedural noise functions are non-differentiable, we estimate gradients using finite difference approximation. This gradient approximation is similar to what is used for other ZK attacks like zeroth-order optimization [98], but here it is applied to a significantly smaller search space. When L-BFGS converges, possibly to a local optima, we restart the optimization with a different random initial point, stopping when the query limit is reached and choosing the best optima value found.

We set a maximum query limit of 1,000 universal evasion evaluations on the training set X_{train} . In practice, this limit was not necessary as both algorithms converged faster to their optimal values: within the first 100 queries for Bayesian optimization and within the first 250 queries for L-BFGS. These are untargeted universal ZK attacks where the adversary only requires the top label from the model’s outputs.

Results. The best procedural noise UAP computed from the training sets generalised well to the much larger validation set, consistently reaching 70% or more universal evasion on the validation set for Perlin noise. This is a surprising result as the training sets were 10-100 times smaller than the validation set. This may be due to the inherent universality of our procedural noise perturbations. We focus on Perlin noise as it outperforms Gabor noise, with the latter averaging 58% universal evasion on the validation set.

Table 3.4 shows that Bayesian optimization (BayesOpt) reliably outperforms L-BFGS in terms of universal evasion rate on the training sets and the resulting universal evasion rates on the validation set. For comparison, random parameters for Perlin noise in Section 3.2 had a 98th percentile of 70.2% and a maximum of 73.1% universal evasion. Bayesian optimization consistently reached or passed this 98th percentile while L-BFGS did not. It is reasonable for these optimization algorithms not to beat the maximum since the training sets were significantly smaller. Similar trends appear between Bayesian optimization, random selection, and L-BFGS for Gabor noise perturbations.

Table 3.4: Comparison on Inception v3 between universal Perlin and Gabor noise ZK attacks at ℓ_∞ -norm $\varepsilon = 16$. Universal evasion rates (%) of the optimised perturbations are shown for their respective training set and the validation set.

Train Size	BayesOpt _{per}		L-BFGS _{per}		BayesOpt _{gab}		L-BFGS _{gab}	
	Train	Val.	Train	Val.	Train	Val.	Train	Val.
50	78.0	71.4	74.0	69.9	64.0	57.6	58.0	51.6
125	77.6	70.2	76.0	71.5	64.6	58.0	58.4	54.6
250	71.6	71.2	71.2	69.7	60.0	58.4	58.8	56.0
500	75.0	72.9	73.4	70.8	64.8	62.4	59.8	58.0

Table 3.5: Highest UER (in %) of Fourier and Procedural noise UAPs for models at ℓ_∞ -norm $\varepsilon = 16$. Best attack is in bold.

Percentile	Fourier [4]	Gabor	Perlin
VGG-19	66.1	85.5	90.1
ResNet-50	57.5	77.3	79.7
Inception v3	49.7	61.8	72.9

Comparison with Fourier Basis Functions. Fourier basis functions [4] is so far the only other UAP attack designed for a ZK threat model like our procedural noise attack. Their SSFA algorithm to generate UAPs based on Fourier basis functions is detailed in the related work—Section 2.3. To compare our Procedural noise with Fourier basis functions, we choose the UAPs with the highest UER for models trained on ImageNet. Results in Table 3.5 demonstrate that under similar threat models of ℓ_∞ -norm $\varepsilon = 16$, UAPs generated via procedural noise outperform those generated with Fourier basis functions by up to 1.47 times. Procedural noise holds the advantage as it is a more expressive noise function. SSFA only adjust the frequency parameters of their Fourier basis functions, whereas our procedural noise attack adjusts a variety of parameters (octaves, frequencies, colour map parameters).

3.4.2 Input-dependent Zero-knowledge Attack

Now that we have seen the effectiveness of our procedural noise as a ZK UAP, we now show how it is effective as a ZK input-dependent attack. The goal of the adversary is to evade as many inputs in the dataset, maximizing the input-dependent evasion rate as defined in Equation 3.2.

Another important metric here is the query-efficiency of these attacks, as requiring large volumes

of queries *per sample* becomes undesirable and oftentimes impractical in real-world scenarios. Ideally, the best attack would evade the model for an input with as few queries as possible.

Experiment. We use the Inception v3 model, ℓ_∞ -norm $\varepsilon = 16$, and the same 5,000 data points from Section 3.2. For a given input x , the goal is to achieve evasion with $\arg \max f(x+s) \neq \tau(x)$ by minimizing the probability of the true class label $\tau(x)$. In this case we allow the attacker to access the model’s output probability vector, as the black-box optimization algorithms gain minimal usable information if the output is binary ($\tau(x)$ or $\neg\tau(x)$).

As in the previous section, we compare Bayesian optimization and L-BFGS with a limit of 1,000 queries per input and number of restarts when it converges. We use sparse GPs to better scale Bayesian optimization [99], as the standard GP scales cubically with the number of observations. We also compare with uniform random parameter selection from Section 3.2. These are untargeted input-dependent ZK attacks where the adversary has no knowledge of the target model and only requires the output probability vector of the model.

Results. Table 3.6 shows that Bayesian optimization reached a high 91.6% success rate with just 7 queries per successful evasion on average, under a restrictive 100 query limit. This vastly improves on the query efficiency over random parameters by 5-fold whilst attaining the same accuracy. The improvement for Bayesian optimization increasing the query limit from 100 to 1,000 was very incremental. This suggests that the limited codomain of the procedural noise function is setting an upper bound on the attack’s success rate.

L-BFGS performed the worst as we observe it can get trapped and waste queries at *poor* local optima, which requires several restarts from different initial points to improve its performance. There were similar trends for Gabor noise where Bayesian optimization had 78.3% success with 9.8 queries on average. For both procedural noise functions, Bayesian optimization improved the average query efficiency over L-BFGS and random parameter selection by up to 7 times while retaining a success rate greater than 83%.

Comparison. We compare our results with bandits [9] in Table 3.6, where they formalise their attack as a gradient estimation problem (like in PK gradient-based attacks), and use a bandit

Table 3.6: Comparison on Inception v3 between our input-dependent Perlin and Gabor noise attacks and the bandit attack [9] for different query limits at ℓ_∞ -norm $\varepsilon = 16$.

Attack ^a	Query Limit	Average Queries	Success Rate (%)
BayesOpt _{per}	100	7.0	91.6
BayesOpt _{per}	1,000	8.4	92.8
L-BFGS _{per}	100	19.8	71.7
L-BFGS _{per}	1,000	70.1	86.5
Random _{per}	1,000	36.3 ^b	91.6
BayesOpt _{gab}	100	9.8	83.1
BayesOpt _{gab}	1,000	10.9	83.6
L-BFGS _{gab}	100	6.0	44.7
L-BFGS _{gab}	1,000	6.0	44.7
Random _{gab}	1,000	62.2	86.1
Bandits _{TD}	100	29.0	36.7
Bandits _{TD}	1,000	224	73.6
Bandits _{TD}	10,000	888	96.9

^aSubscripts “per” and “gab” refer to Perlin and Gabor noise respectively.

^bFor each input, we divide the total number of perturbations by the number of those that evade that input to get an expected number of queries.

optimization framework to solve it. We test the bandits attack on the same model, dataset, and ℓ_∞ -norm for maximum query limits of 100, 1,000, and 10,000.

The results show that the evasion rate of our input-dependent procedural noise attack greatly outperforms the bandits attack when the query limit per image is a thousand or less. For significantly larger query limits, the Perlin noise Bayesian optimization attack has a competitive success rate at a drastically better query efficiency—needing *100* times less queries on average.

The procedural noise attack is optimised for universal evasion and a restrictively small amount of queries. Most existing methods explore the entire image space which has a hundred thousand dimensions, and some attacks reduce their search space with techniques like tiling, but the dimensionality is still much larger than the four parameters of our procedural noise. Other input-dependent ZK attacks [100, 101, 98] require tens to hundreds of thousands of queries on realistic natural-image datasets, which makes them inefficient, but almost certain to find an adversarial example given enough queries. The bandits method makes a small sacrifice in success rate for more query efficiency, and our procedural noise attack takes it further for greater query-efficiency.

Procedural noise perturbations have naturally high evasion rates, although the expressiveness of our chosen functions can be less than attacks whose larger codomains can capture more kinds of adversarial perturbations. These other attacks make this trade-off by sacrificing efficiency in ZK scenarios, as the number of queries they need to craft successful adversarial examples is large. On the other hand, we can increase the expressiveness of our procedural noise functions by introducing more parameters by, for example, using different colour maps. However, this may come at the cost of its naturally high universal evasion.

Summary. Black-box optimization techniques like Bayesian optimization are able to take advantage of the drastically reduced search space of procedural noise. With its naturally high UER, we have shown that procedural noise gives rise to inexpensive yet potent untargeted ZK attacks, whether it be input-dependent or universal. It is also shown to be competitive with existing attacks that often require orders of magnitude more queries and resources.

3.5 Beyond Image Classification

In this section, we show how procedural noise attacks extend to object detection DNNs by attacking the YOLOv3 object detection model. We then discuss how exploiting the sensitivity of DNNs to low-level features, like in procedural noise attacks, can be generalised to craft universal attacks for DNNs in other application domains.

3.5.1 Background: Object Detection

Object detection is the task of locating and then identifying objects within an image. This differs from the image classification problem in that object detection requires both the localisation (where in the image) and classification (what it is) of any objects within an image. For each object detected, the model outputs a bounding box to identify the location of an object and assigns that box an output class label.

Figure 3.9 shows an example of an object detector’s output. For the clean images on the left,

the model is able to localise and classify more than objects within an image. Note that our procedural noise attacks on the right are able to (top) hide objects from the detector and (bottom) cause misclassification of objects.

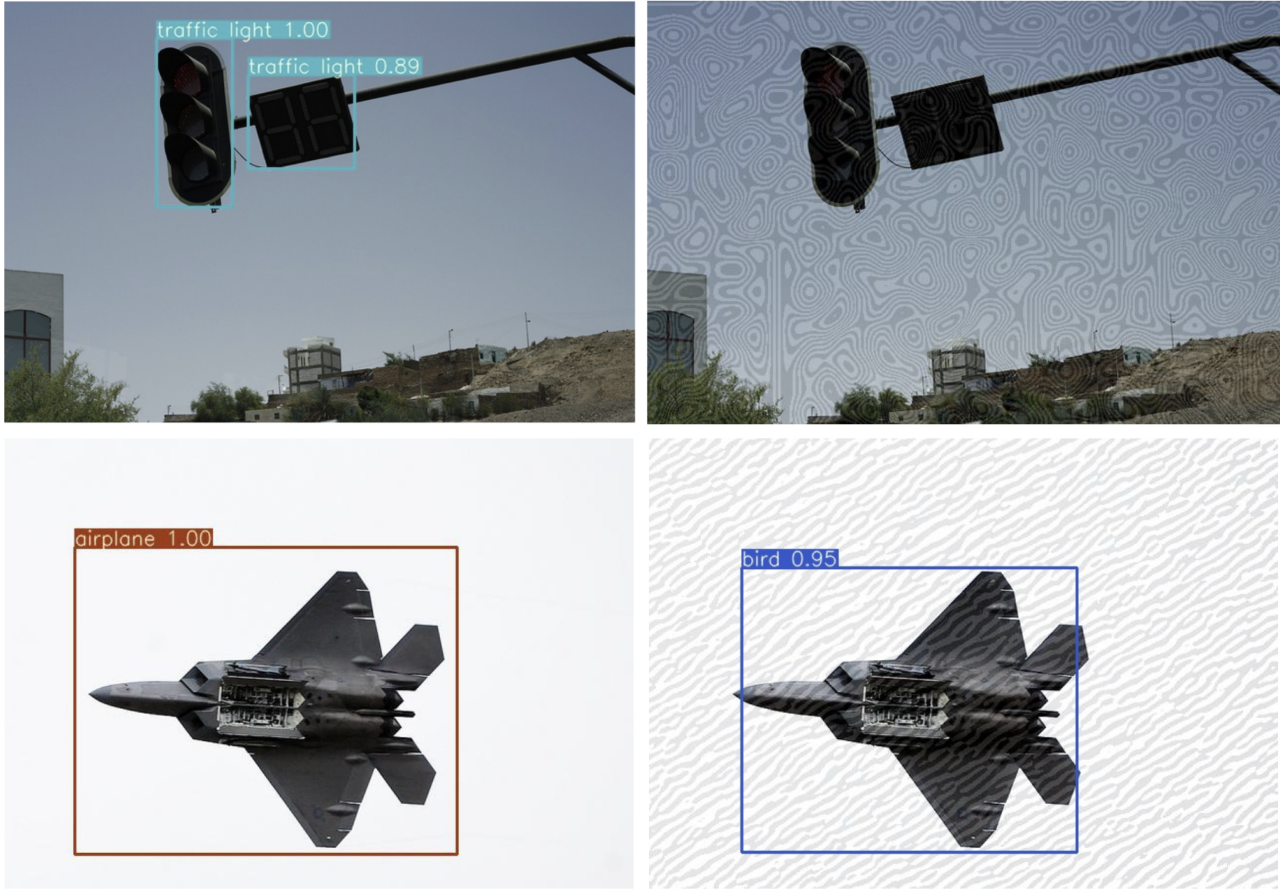


Figure 3.9: Visualisation of YOLOv3 outputs on (left) images from the MS COCO dataset [5] and (right) the same images adversarially perturbed with procedural noise.

Single Shot Detectors (SSD) are a type of object detection DNNs that generate their predictions after a single pass over the input image. This allows SSDs to process images in real-time speeds while maintaining high accuracy. A notable example of SSD models are the different versions of YOLO [102, 103, 12]. Other types of object detectors such as Fast R-CNN [104] and Faster R-CNN [35] are based on region proposals which take several more steps than SSDs. The R-CNN family of models have comparable accuracy, but do not achieve the same image processing speed as SSDs like YOLOv3. We choose to evaluate our procedural noise attack against YOLOv3 [12], one of the most widely-used iterations of YOLO, which achieves high accuracy and is capable of detecting objects in real time.

Mean Average Precision (mAP) is the metric used to evaluate the overall performance of object detectors. Before explaining mAP, we first define the following sub-metrics mAP is based on: Intersection Over Union (IOU), recall, and precision.

Intersection Over Union (IOU) is the ratio of the intersection area over the union area between the predicted and ground truth bounding boxes. A threshold is usually set to determine what IOU constitutes a positive classification. True positives occur when the IOU is larger than the threshold and the class is identified correctly. False negatives occur when the threshold is not met with the correct class for a ground truth bounding box. False positives occur when the IOU is less than the threshold, there are no intersecting ground truth boxes, or there are duplicate predicted bounding boxes. mAP values are calculated over recall values from 0 to 1. *Average Precision (AP)* is calculated as the weighted mean of precision at each threshold; the weight is the increase in recall from the prior threshold. mAP is the average of AP for each class.

3.5.2 Experimental Setup

Threat Model. For adversarial capabilities, we use the ℓ_∞ -norm threat model. For adversarial knowledge, a ZK threat model is sufficient for our procedural noise attack. The goal of this adversary is to decrease the overall mAP of the model.

Experiment. We use the MS COCO dataset [5] which contains 80 different classes, with a large proportion of targets being the “person” class. This has become one of the benchmark datasets for object detection. We use standard settings as described in [12], with input dimensions $416 \times 416 \times 3$ and an IOU threshold of 0.5.

We use ℓ_∞ -norm $\varepsilon = 16$ and apply each of our procedural noise perturbations on the 1,000 random images from the validation set. The perturbations generated are from 1,000 Gabor noise, 1,000 Perlin noise, and 1,000 ℓ_∞ -optimised uniform random perturbations. The parameters are chosen uniformly at random so that we can analyze the results. This is a universal untargeted ZK attack.

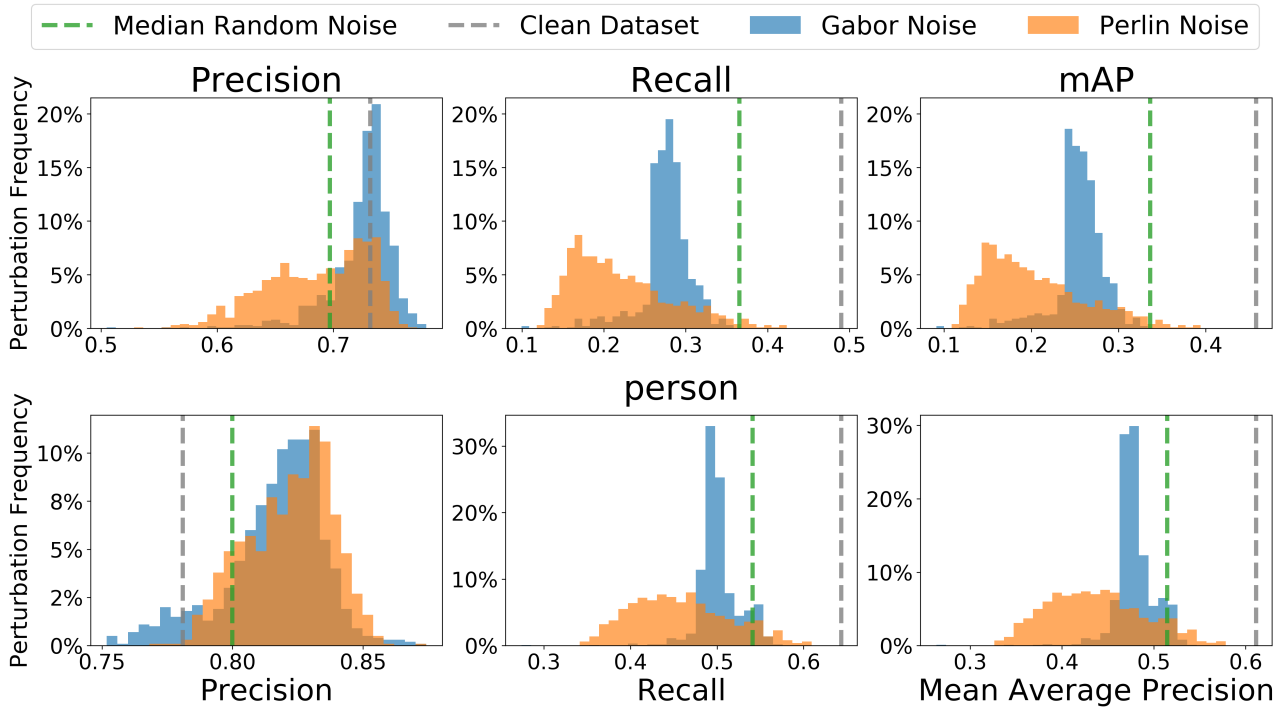


Figure 3.10: Histogram of metrics over all perturbations on YOLOv3. Results for (top) all classes and (bottom) the “person” class.

3.5.3 Results & Discussion

The procedural noise perturbations significantly decrease the mAP of the model, with 75% of Perlin noise perturbations halving the mAP or worse. The precision and recall also decreased in the presence of our perturbations. Again, Perlin noise outperforms Gabor noise on average but Gabor noise has a smaller minimum, with at least one perturbation causing 0.09 mAP. Compared to uniform random noise, which maintained around 0.34 mAP, both Perlin and Gabor noise had larger impact on the model’s performance. We represent uniform random noise using the median (which, in this case, is very close to the mean) on each metric as the variance is very small (less than 10^{-5}).

In Figure 3.10, for the metrics across all classes, we observe that precision was maintained or decreased while recall and mAP decreased. Some perturbations decreased all three metrics, which indicate that these consistently cause both false positives and false negatives. When the perturbation increases the precision while decreasing the recall and mAP, then the noise is likely

masking objects rather than introducing new objects to the image. This masking effect has serious implications for safety-critical applications like surveillance and autonomous vehicles.

We focus on the “person” class as it constitutes a majority of the targets and is semantically meaningful in the context of some applications. In Figure 3.10, metrics for “person” follow a similar trend to the metrics across all classes. However, the increase in precision is very small compared to the large drops in recall and mAP caused by Perlin noise. The higher precision indicates fewer false positives, while the decrease in recall indicates more false negatives. This indicates that the classifier is making fewer predictions overall, which means that the adversarial noise is masking persons in the image.

Whilst for the most relevant “person” class, our procedural noise appears to have an obfuscating effect, for other classes like “zebra” all the metrics decrease – indicating that there are many false positive and false negatives. However, for three classes, “backpack”, “book”, and “toaster”, all three metrics improve for the classifier. These labels did not have as much representation in the test set which may explain this anomalous result.

The frequency of the sine ϕ_{sine} for Perlin noise had the largest inverse correlation of less than -0.72 with each of the three metrics. This means that high-frequency patterns decrease the model’s performance metrics, similarly to what we have observed for the image classifiers. The thickness λ of Gabor noise is moderately correlated at 0.4 with the precision, but not the recall or mAP. This suggests that thicker Gabor noise perturbations decrease the number of false positives relative to the other perturbations. This can be seen in Figure 3.8, which shows the correlation matrices between the procedural noise parameters with the corresponding universal evasion rates on the various ImageNet classifiers. These correlations quantify the effects of each parameter and the transferability of UAPs across models.

Object detection is a more complex task than image classification as it has to identify multiple objects and their locations within an image. Although YOLOv3 has a different architecture, task, and dataset from the ImageNet classifiers, we see that the same procedural noise perturbations are still able to greatly degrade its performance. This shows that it is more likely caused by the models’ sensitivity towards perturbations rather than a texture bias, as the object

detection task has a spatial component to it. This suggests that our procedural noise attacks may generalise to other DNNs on computer vision tasks with natural-image data.

In our discussion in Section 3.2, we hypothesised that the procedural noise perturbations are an aggregation of low-level features at high frequencies that the DNNs strongly respond to. The prior that convolutional layers induce and similarities across natural-image datasets may be why DNNs are sensitive to these noise patterns. Additionally, like in Section 3.4, we can also create more efficient ZK attacks by applying black-box optimization techniques such as Bayesian optimization to enhance the attack against object detectors.

3.5.4 Beyond Images

One of the main strengths of procedural noise is that it allows to describe a distribution of UAPs with only a few parameters. This compressed representation allows for an efficient generation of UAPs both for undermining a machine learning service or, defensively, for testing the robustness of a model. In practice, compressed representations can be learned by generative models such as GANs [79, 105] or Variational Autoencoders (VAEs), however these incur additional training, calibration, and maintenance costs. Training algorithms or defences that incorporate domain-specific knowledge may be needed to mitigate the sensitivity towards attacks that make use of these compact representations.

DNNs in other applications may also be vulnerable to aggregations of low-level features due to the use of convolutional layers. Future attacks can exploit how DNNs rely on combining low-level features rather than understanding the more difficult global features in the input data. A natural next step would be to apply these ideas in exploratory attacks or sensitivity analysis on sensory applications like speech recognition and natural language processing. To expand our procedural noise attack framework to other applications, it is worth identifying patterns in existing adversarial examples for domains like speech recognition [106, 107] and reinforcement learning [108, 109] to find analogues of procedural noise. As a starting point, these patterns can be found by finding perturbations that maximise the hidden layer difference as in the

Singular Vector Attack [3] or by using feature visualization on earlier layers of DNNs to infer the low-level features that a model learns.

3.6 Preliminary Defence

The DNNs we tested are surprisingly fragile to procedural noise as UAPs. Improving their robustness to adversarial examples is not a straightforward task. A robust defence needs to defend not only against existing attacks but also future attacks, and often proposed defences are shown to fail against new or existing attacks [31].

Among existing defences, adversarial training (AT) appears to be more robust than others [30]. However, we have shown in Table 3.2 that ensemble AT against gradient-based attacks did not significantly diminish the input-dependent evasion rate of our procedural noise attack. This suggests that such defences do not generalise well as the attacks used for adversarial training do not sufficiently represent the entire space of adversarial examples. Training against all types of adversarial attacks would become computationally expensive, especially for high-dimensional tasks like ImageNet. Thus, defences that regularise the model or incorporate domain-specific knowledge may be more efficient strategies.

DNNs’ weakness to adversarial perturbations across inputs may be a result of their sensitivity—small changes to the input cause large changes to the output. Input-agnostic defences that minimise the impact of small perturbations may be effective in reducing the models’ sensitivity without the need to train against all types of attacks. As a preliminary investigation, we briefly explore using denoising to defend against the universality of our procedural noise perturbations.

Defence. Denoising with spatial filters is a common pre-processing step in signal processing and is thus an attractive defence due to its simplicity and pervasiveness across signal processing applications. Median filtering is a smoothing operation that replaces each entry with the median of its neighbouring entries, and it often preserves edges while removing noise. The idea is to smooth out the high-frequency noise, which we know to be highly correlated with the universal evasion rates for Perlin noise on Inception v3.

Experiment. We retrain Inception v3 with a median denoising filter applied as a pre-processing step. For this evaluation, we use the same 1,000 of each procedural noise, 1,000 uniform random perturbations, and 5,000 validation set points as in Section 3.2. We test how denoising improves robustness for ℓ_∞ attack norms $\varepsilon = 4, 8, 12$, and 16. The decrease in the number of uniform random perturbations is due to results in Section 3.2, where we found that its universal evasion rate had very low variance across 10,000 samples, and our results show it is similar for 1,000 samples.

Results. The min-max oscillations present in the procedural noise perturbations may have allowed the noise patterns to persist despite the denoising, as the universal evasion rates are not completely mitigated by the defence. Across the different ℓ_∞ -norm values, the denoising decreased the median and mean universal evasion rates by a consistent amount when compared with no defence: 7.2-10.8% for Gabor noise and 13.2-16.9% for Perlin noise. Figure 3.11 shows that the decrease in effectiveness of Perlin noise is much greater than that for Gabor noise, suggesting that the denoising appears to slightly mitigate the effectiveness of high-frequency noise patterns. This denoising defence grants a reasonable increase in robustness despite the simplicity of the algorithm. However, this measure has not fully mitigated the sensitivity to procedural noise.

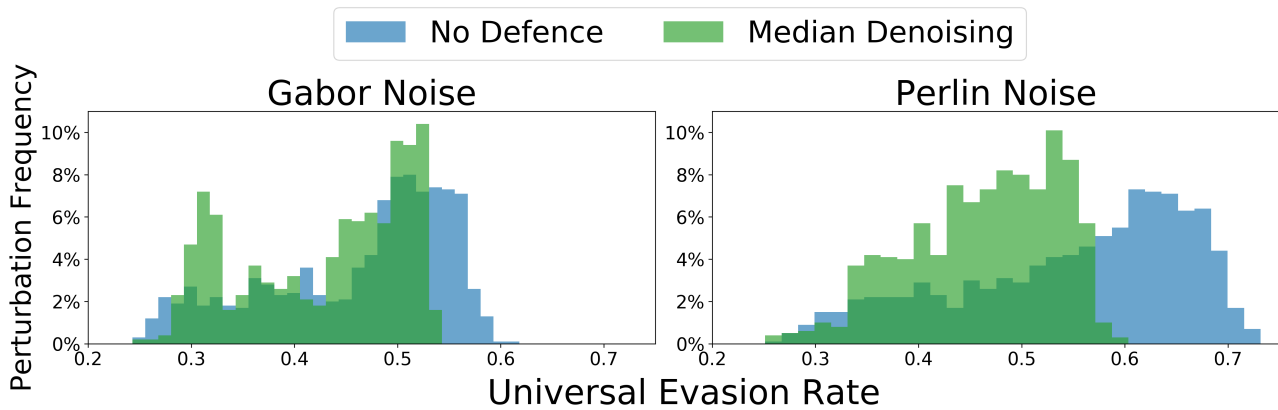


Figure 3.11: Histogram of metrics over all perturbations on Inception v3 with and without median denoising for (left) Gabor noise and (right) Perlin noise for ℓ_∞ -norm $\varepsilon = 16$.

3.7 Conclusion

We highlight the strengths of procedural noise as an indiscriminate and inexpensive ZK attack on DNNs. We have shown that popular DNN architectures for image classification have systemic vulnerabilities to procedural noise perturbations, with Perlin noise UAPs achieving 58% or higher universal evasion across non-adversarially trained models. This weakness to procedural noise can be used to craft efficient universal and input-dependent ZK attacks. Our procedural noise attack augmented with Bayesian optimization attains competitive input-dependent success rates whilst improving the query efficiency of existing ZK methods over 100 times. Moreover, we show that procedural noise attacks also work against the YOLOv3 model for object detection, where it has an obfuscating effect on the “person” class.

We have shown the difficulty of defending against such novel approaches like our procedural noise attack. In particular, Ensemble Adversarial Training was unable to significantly diminish the effectiveness of our procedural noise attack. We suggest that future defences take more input-agnostic approaches to avoid the costs in defending and retraining against all possible attacks. To summarise, we enumerate the main findings of this chapter:

- We show a novel and intuitive vulnerability of DNNs in computer vision tasks to procedural noise perturbations. These functions characterise a distribution of noise patterns with high universal evasion, and universal perturbations optimised on small datasets generalise to datasets that are 10 to 100 times larger. To our knowledge, this was the first model-agnostic ZK generation of UAPs at the time.
- We propose and demonstrate that *Bayesian optimization* is an effective tool for augmenting ZK attacks on DNNs. In particular, we show that it can use our procedural noise to craft inexpensive universal and input-dependent ZK attacks. It improves on the query efficiency of random parameter selection by 5-fold and consistently outperforms the popular L-BFGS optimization algorithm. Against existing query-efficient ZK attacks, we achieve a 100 times improvement on the query efficiency while maintaining a competitive success rate.

- We show evidence that our procedural noise UAPs exploit low-level features in DNNs, and that this vulnerability may be exploited to create universal attacks across applications. We also highlight the shortcomings of adversarial training and suggest input-agnostic defences to reduce model sensitivity to adversarial perturbations.

Chapter 4

Robustness of Texture and Shape-biased Models

To better characterise and mitigate the vulnerabilities of computer vision Deep Neural Networks (DNNs) to Universal Adversarial Perturbations (UAPs), it is crucial to visualise and understand the types of features that these DNNs learn and primarily use for classification. By investigating these features we will be able to determine what leads to the vulnerability of these DNNs.

The previous chapter on procedural noise demonstrates that it is these low-level textural patterns that mainly comprise adversarial perturbations. To elucidate the types of features that computer vision DNNs learn on image classification tasks such as the ImageNet benchmark [11], recent work have investigated the effect of texture and shape-bias that these models have on their performance [110, 111].

Previously, there have been two competing hypotheses for what features models primarily use for classification as described by [111]: first is the *shape hypothesis*, where the intuition is that DNNs aggregate low-level features such as curves and edges to recognise high-level global object shapes. Second is the *texture hypothesis*, which suggests that image textures are more important than object shapes in determining the output. Evidence from [110] and [111] support the latter texture hypothesis as they show it is sufficient for models to use image textures to achieve high accuracy on the widely-used ImageNet benchmark [11].

Geirhos et al. proposed Stylized-ImageNet [111], a modified ImageNet dataset that requires recognizing object shapes rather than textures to discriminate images in the dataset. They claimed that biasing models towards shapes improves their robustness, and they showed that these shape-biased models have improved robustness against common corruptions. However, to have a complete characterization of a model’s robustness, it is important to also study the worst-case distortions. These worst-case distortions come in the form of *adversarial perturbations* [21, 22] as these attacks can greatly undermine the integrity and trust in model predictions.

In this chapter, we evaluate the robustness of shape and texture-biased models used for image classification to UAP attacks. Utilising both targeted and untargeted UAPs, we perform a novel analysis on the most frequent output labels to better characterise the vulnerabilities of both shape and texture-biased models. As UAPs are input-agnostic, they represent global features that greatly influence the model output. Rather than using subjective notions of “shape” or “texture”, we are able to quantify the sensitive features of the model by measuring the most frequent class label for each UAP. These global features are worth studying as they reveal the underlying features that models use for classification. Other studies have claimed some improvements in adversarial robustness to input-dependent attacks by increasing shape-bias [112], however our results show that this is not the case against universal attacks. Our contributions in this chapter have important implications on the understanding of neural networks and brings a different perspective to the shape versus texture discussion.

4.1 Shape-biased Training

ImageNet is a widely used computer vision benchmark with 1,000 distinct object categories [11]. Geirhos et al. created a shape-biased version of this dataset in *Stylized-ImageNet*, which replaces textures of ImageNet images while retaining the global shape of the original objects [111]. This dataset requires a model to use shapes rather than textures to correctly identify and discriminate objects. Stylized-ImageNet is generated by applying style transfer of different uninformative textures onto ImageNet images.

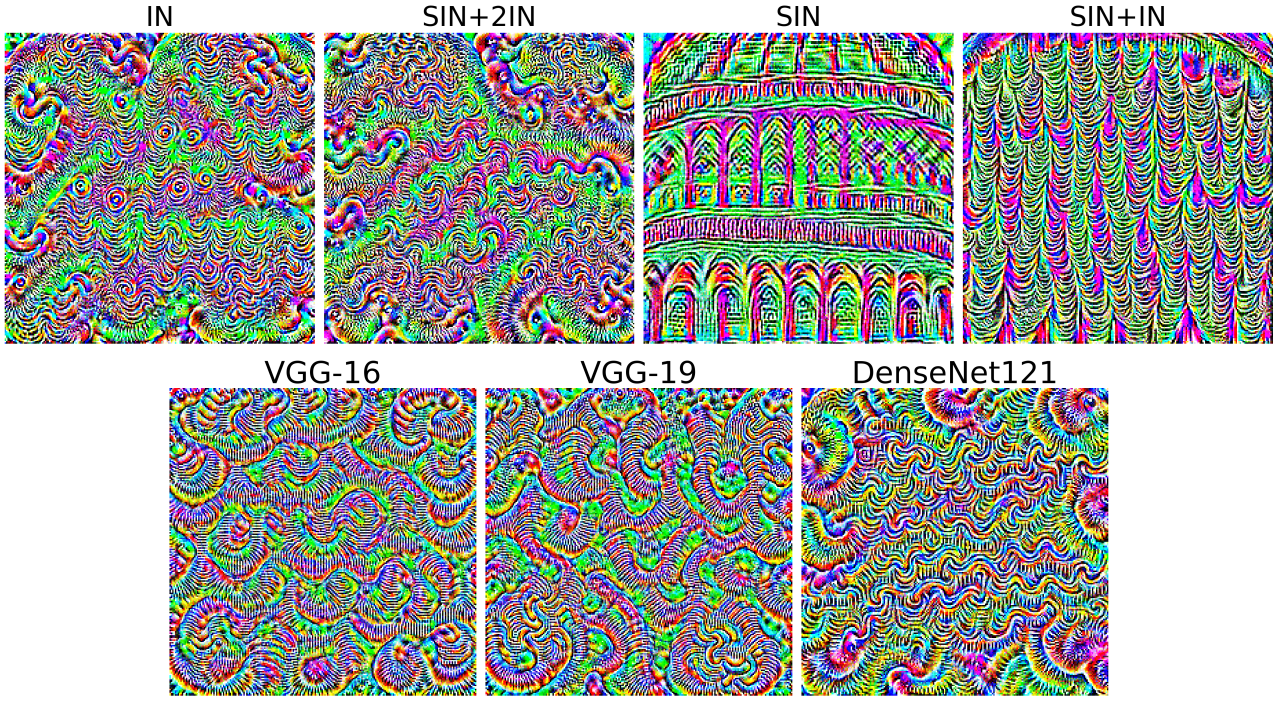


Figure 4.1: Examples of UAPs generated from (top) the different ResNet-50 [6] models and (bottom) VGG [7] and DenseNet [8] models. For the top row, the titles for each indicate how the model was trained, complete details for these are in Section 4.1.

Models. We use the ResNet50 [6] architecture with varying degrees of training on ImageNet and Stylized-ImageNet. This model takes input images with dimensions $224 \times 224 \times 3$. We test these four ResNet50 models, named according to the training dataset used:

- **IN:** only ImageNet
- **SIN:** only Stylized-ImageNet
- **SIN+IN:** both Stylized-ImageNet and ImageNet
- **SIN+2IN:** both Stylized-ImageNet and ImageNet, and then fine-tuned on ImageNet

From here onwards, we use IN and SIN to refer to the trained ResNet50 models rather than the datasets.

4.2 Robustness of Shape-biased Models

We first compare the robustness of ResNet50 models under different training regimes by measuring the effectiveness and transferability of untargeted UAPs generated from the SGD attack. Our results show that models trained on Stylized-ImageNet are still as vulnerable to these UAPs as models trained on ImageNet.

4.2.1 Experimental Setup

Models. Aside from the four different ResNet50 models we described previously (IN, SIN, SIN+IN, SIN+2IN), we also evaluate untargeted UAPs on other model architectures trained on ImageNet: DenseNet121 [8] and VGG19 [7]. Like ResNet50, these models take input images of dimension $224 \times 224 \times 3$ and have similar clean dataset accuracy.

Threat Model. As we are evaluating the worst-case robustness, we use the PK threat model. That is, the adversary has knowledge of the internal model parameters. For adversarial capabilities, perturbations are applied under ℓ_∞ -norm constraints as it is the standard in the UAP and adversarial machine learning literature. The ℓ_∞ -norm constraint on δ ensures that the perturbation is small and does not greatly alter the visual appearance of the resulting image. We refer the reader to Section 2.2.2 for a more thorough discussion on this choice.

For the adversarial goals, we measure the effectiveness of untargeted and targeted universal attacks. Thus, we use the Universal Evasion Rate (UER) as Targeted Success Rate (TSR) defined in Equations 2.2 and 2.3 respectively. UER is the proportion of the dataset that the model makes errors on when the UAP is applied. TSR is the proportion of the dataset where that the model classifies as the target class y_{tgt} when the targeted UAP is applied. We evaluate the robustness of these models to UAP attacks generated via SGD [53]. As discussed in Section 2, SGD is chosen as it SGD is the most potent PK attack, making it the most appropriate for evaluating the UAP robustness of models in a PK setting.

Summary. We consider the four ResNet50 models as outlined in Section 4.1. We generate and

optimise the SGD-based perturbations on each model for various ℓ_∞ -norms ($2 \leq \varepsilon \leq 12$) over the entire 50,000 image ImageNet validation set. The UAP literature often focuses on $\varepsilon = 10$, so we use this as our primary benchmark, but we also provide results for other ε values to measure its effectiveness under different perturbation constraints. For each UAP, we evaluate the UER and TSR on the model it was generated from (PK) and on the other remaining models (transfer attack).

Initial Result. All perfect-knowledge attacks, i.e. where the UAP is generated from the same tested model, achieve high attack success rates. They consistently reach greater than 90% Universal Evasion Rate for $\varepsilon = 10$ on all models as shown in Figure 4.2. This shows that all these models remain as vulnerable to universal attacks independent of the Stylized-ImageNet training.

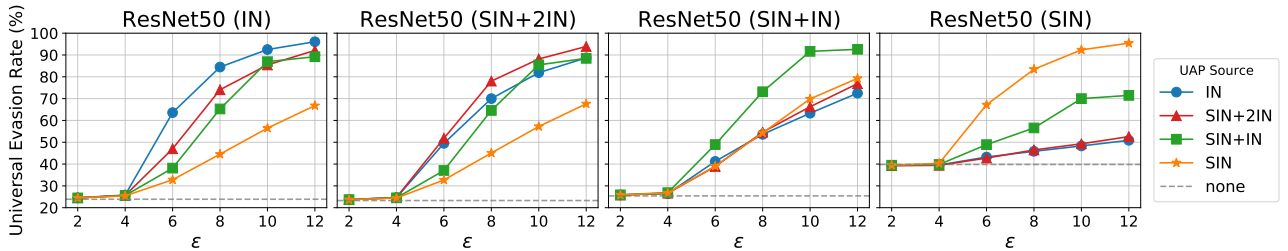


Figure 4.2: UER (in %) of untargeted UAPs for different perturbation values ε . Plot titles indicate the evaluated model.

4.2.2 Transferability Across Training Datasets

In investigating the transferability of UAPs, we are able to quantitatively measure to what extent these models share their vulnerabilities. For transfer attacks, the UAP is first generated from an origin model, then that UAP is used in robustness evaluations on models distinct from the original one.

We observe that the UER consistently rises for $\varepsilon > 4$. SIN appears to be the most resilient to transfer attacks against all other models trained on the ImageNet dataset. However, this robustness to transfer attacks comes at the cost of having the highest clean error on ImageNet.

UAP Source	IN	92.5	81.9	63.3	48.3	68.2	59.3
	SIN+2IN	85.4	88.2	66.3	49.3	65.7	54.5
	SIN+IN	86.9	85.5	91.7	70.0	78.9	70.5
	SIN	56.5	57.2	69.9	92.4	60.2	50.5
	V19	50.1	46.8	38.7	44.6	93.9	44.4
	D121	66.0	60.6	51.1	49.9	71.0	88.5
		IN	SIN+2IN	SIN+IN	SIN	V19	D121
Tested Model							

Figure 4.3: Confusion matrix showing the UER (in %) of untargeted UAPs with $\varepsilon = 10$ when applied as transfer attacks to different models. VGG19 and DenseNet121 are labeled as V19 and D121 respectively.

That UAPs are still effective against SIN shows that Stylized-ImageNet training does not diminish the effects of UAPs. Instead, Stylized-ImageNet training shifts the set of features the model has learned, making it vulnerable to a different set of features from what the ImageNet-trained models have learned. Hence, what features models are most vulnerable to depends more on the data distribution of its training dataset. It is interesting to note in Figure 4.2 that UAPs from IN and SIN+2IN have near-identical effectiveness against SIN.

Impact of fine-tuning. Although SIN+2IN achieves the best clean dataset accuracy, our results show that its response to UAPs is very similar to that of the model IN that was trained only on ImageNet. In Figures 4.2 and 4.3, the Universal Evasion Rates of UAPs from IN and SIN+2IN are highly similar across all tested models. Additionally, there are large visual similarities between the UAPs generated for SIN+2IN and IN as seen in Figure 4.1. These suggest that there are large similarities in the features that SIN+2IN and IN are vulnerable to, despite the additional training on Stylized-ImageNet. The additional fine-tuning on ImageNet for SIN+2IN could have resulted in the model “overfitting” where it no longer uses features it has learned from both datasets, and instead focuses only on features learned from ImageNet.

4.2.3 Transferability Across Architectures

We now consider the transferability of the UAPs from these ResNet50 models to other architectures: DenseNet121 [8] and VGG19 [7]. Like ResNet50, these models take input images of dimension $224 \times 224 \times 3$ and have comparable clean dataset accuracy. Compared to ResNet50, VGG19 has fewer layers, whilst the DenseNet121 has more layers and uses dense blocks. The architectures have approximately 8, 25, and 140 million parameters for DenseNet121, ResNet50, and VGG19 respectively. We focus our analysis on attacks with $\varepsilon = 10$.

In Figure 4.3, VGG19 is noticeably more vulnerable to all UAPs, whereas DenseNet121 is more resilient to the UAPs from the other models. When focusing only on the ImageNet-trained models (IN, VGG19, DenseNet121), we see that the architectures with more layers and fewer parameters appear to be more robust. Compared to IN, SIN does not improve on the transferability of UAPs to these other architectures. However, the UAP from SIN+IN has noticeably higher transferability against all models, achieving *at least* 70% UER against each model. We hypothesise that SIN+IN has learned more generalised representations because it was trained equally on both ImageNet and Stylized-ImageNet without overfitting to either, thus allowing it to have a stronger transfer attack to the other model architectures. These results also show that models with more depth and fewer parameters may be more robust to UAPs.

4.3 Dominant Label Analysis

To better understand the effects of shape-based training, we evaluate the robustness of the ResNet50 models that have varying degrees of training on ImageNet and Stylized-ImageNet. For each of these models, we generate and compare the effectiveness of UAPs targeting each of the 1,000 different ImageNet class labels. This is the first study to investigate the effectiveness of targeted UAPs across all class labels of the ImageNet dataset. Our analysis with UAPs reveals interesting similarities and differences in the most and least sensitive features of these models. We are able to quantitatively describe the models' sensitivities based on the dominant class labels of their UAPs. We show that we are able to more precisely characterise the vulnerabilities

of the models using these targeted UAPs.

The distribution of output labels when evaluated on the clean dataset is approximately uniform for all models, with a frequency of about 0.1% for each of the 1,000 class labels. When the dataset is perturbed with an untargeted UAP, it makes models predominantly classify towards specific output labels despite the attack not explicitly optimizing for this a priori. These dominant labels have been previously observed [1], but they have not measured nor studied their effectiveness over the entire validation set across all class labels. In this section, we analyze the most frequent output labels for untargeted UAPs over the ImageNet validation set. We also demonstrate the use of targeted UAPs to visualise and study the sensitivity of different class labels on the ResNet50 models.

4.3.1 Effectiveness of Targeted Universal Attacks

Targeted UAPs can be used to visualise and analyze the sensitivity of the ResNet50 models to UAPs for each class label. These are generated targeting each of the 1,000 ImageNet class labels at $\varepsilon = 10$ on each of the ResNet50 models and their effectiveness is measured by their targeted success rate (TSR). We omit the analysis for SIN+2IN as its robustness to universal attacks is highly similar to that of IN.

Figure 4.4 shows an example of how the different training regimes result in targeted UAPs with differing visual appearances despite targeting the same class. For the middle column, the UAP for SIN appears to capture more of the overall shape of the objects, such as the vehicle in the top row. Whereas IN appears to show more abstract patterns.

Figure 4.5 shows that the TSR ranges from approximately 30-80% and the majority of the 1,000 targeted UAPs for each model are centered the around 50-70% TSR range. What stands out is that SIN+IN appears to be slightly more robust to targeted UAPs as its TSR distribution is shifted to the left when compared to IN and SIN. Since SIN+IN blends elements from both datasets, its larger and more diverse training may have made it more robust to targeted UAPs.

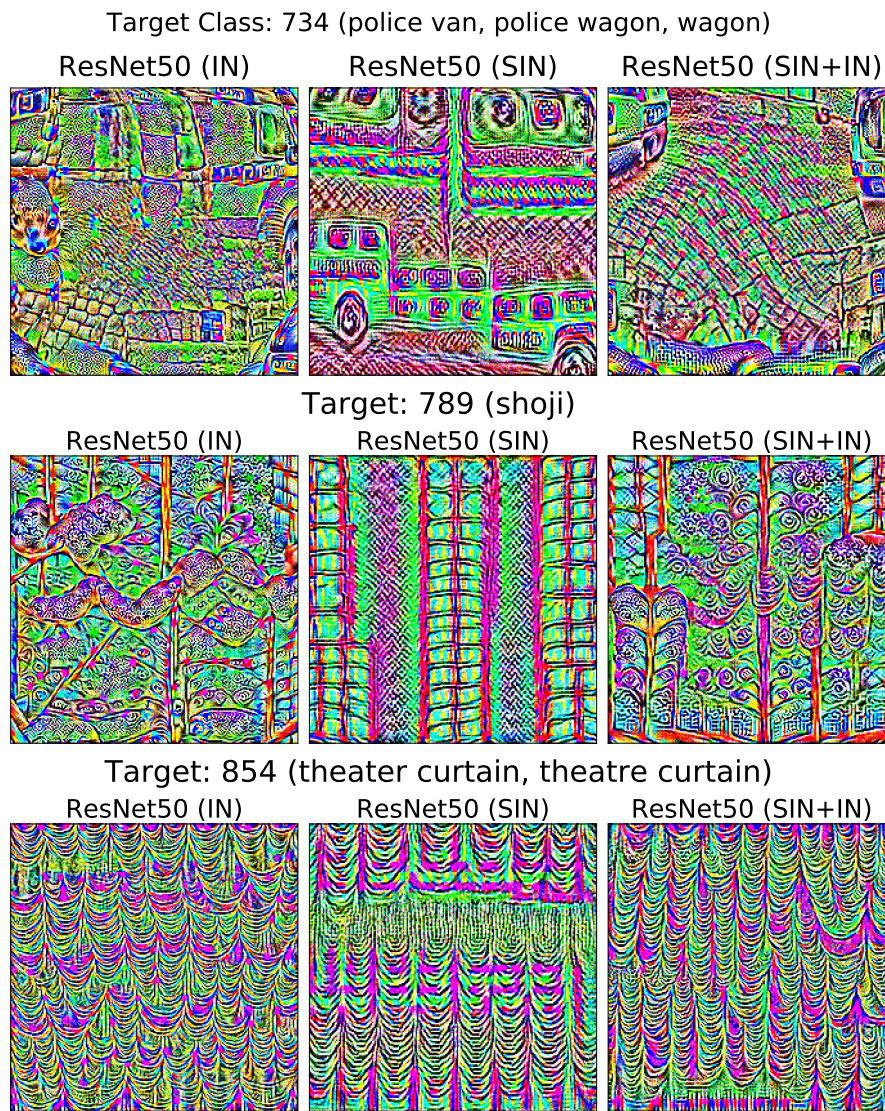


Figure 4.4: Examples of targeted UAPs generated for IN, SIN, and SIN+IN.

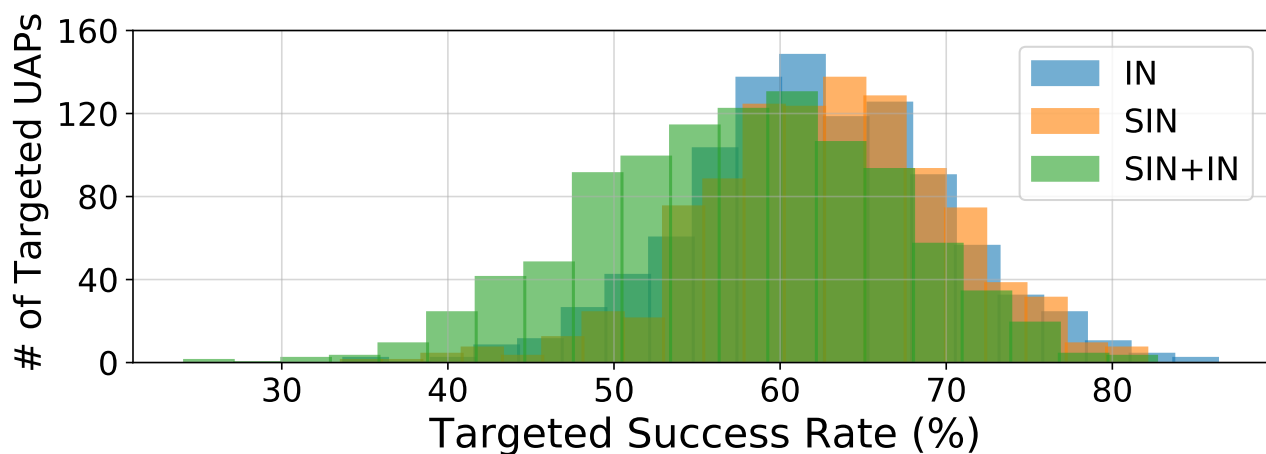


Figure 4.5: Distribution of TSR of targeted UAPs on the ResNet50 models, with one UAP for each ImageNet class labels.

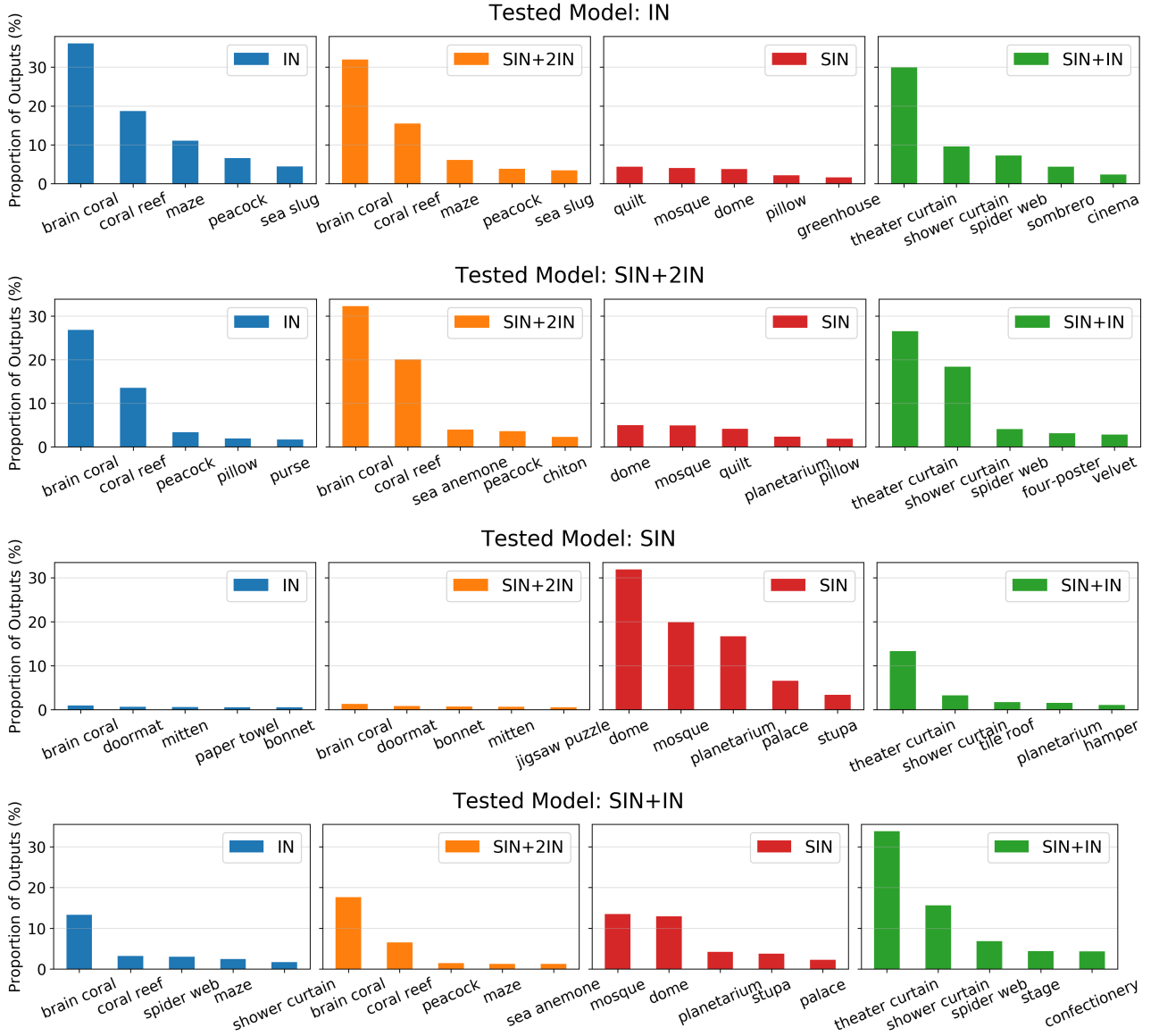


Figure 4.6: Top five output labels for each tested model given the UAP used to perturb the dataset. In each plot we label the source of the untargeted UAP generated via SGD for $\varepsilon = 10$.

4.3.2 Dominant Labels of Untargeted UAPs

This analysis uses the results shown in Figure 4.6. We see that each UAP has at least 2-3 dominant output labels, with each dominant label comprising 10-36% of the predictions on the perturbed dataset. For example, in the top row and first column of Figure 4.6, when IN is attacked with the UAP generated from IN, the three most common output labels are: *brain coral* (36.1%), *coral reef* (18.7%), and *maze* (11.0%). VGG19 and DenseNet121 trained on ImageNet exhibit the same phenomenon, having brain coral and coral reef as the dominant labels of their untargeted UAPs.

For the resulting output labels caused by the UAPs, the dominant labels from UAPs for V16, V19, and D121 are still *brain coral* and *coral reef*, each comprising 20-30% of the output labels. The vulnerability to the same texture-based class labels appears to be an effect of training on the ImageNet dataset. However, the different architectures most likely result in different “interpretations” of these labels. This could explain the slightly differing visual appearances in their UAP patterns as seen in Figure 4.1. The UAP generated by SIN+IN still causes misclassification towards the curtain class labels for V16, V19, and D121 in a similar way that it did against IN.

Table 4.1: Percentiles of TSR (in %) of targeted UAPs on the ResNet50 models on the clean ImageNet validation set.

Percentile:	<i>min</i>	<i>10th</i>	<i>50th</i>	<i>90th</i>	<i>max</i>
IN	33.74	52.56	62.01	71.64	86.33
SIN	33.59	53.71	63.01	71.88	82.07
SIN+IN	24.15	46.17	57.78	68.58	82.66

Looking again at Figure 4.6, UAPs from IN have a negligible effect on SIN as the classified labels appear to be evenly distributed, although *brain coral* is still the most frequently occurring output label. UAPs from SIN do not have as great an effect on IN, but we see that the labels *quilt*, *mosque*, *dome* are still favoured at about 4% each. The UAP from SIN+IN appears to have a noticeable effect against all models, heavily favouring labels with curtains.

We notice the following trends for each of the models: UAPs generated from IN appear to favour more texture-based objects, most notably *brain coral* and *coral reef*. The preference of UAPs on ImageNet-trained models towards these classes was also shown by procedural noise UAPs by [28, 113].

The untargeted UAPs generated for SIN appears to favour structures and buildings. This may be because SIN is more reliant on making decisions based on shapes, so it is more susceptible to misclassifying objects towards labels that have distinctive shapes. Often the shape of buildings in the dataset are clear due to the sky being a backdrop in these images. Just as IN is vulnerable to *brain coral* and *maze* due to the distinct textures of these classes, SIN is vulnerable to buildings which are objects with distinct shapes. UAPs generated from SIN+IN favour curtains.

It is not clear why curtains is the class for SIN+IN, and we explore this further with targeted UAPs in the next section.

These observations on the dominant labels for each model are consistent with the visual appearances of the untargeted UAPs as seen in Figure 4.1. Additionally, these results also align with our findings that training on different datasets makes models more vulnerable to different sets of features. In this case, the models’ vulnerable features manifest as the models’ interpretation of these dominant labels.

4.3.3 Effectiveness of Targeted Universal Attacks

Dominant labels arise in untargeted UAPs even when these perturbations are not optimised to prioritise any specific output label. Targeted UAPs on the other hand are specifically designed to maximise the number of outputs classified as a desired class label. It then becomes natural to investigate how effective targeted UAPs are for each class label and how does ImageNet and Stylized-ImageNet training affect these UAPs.

Targeted UAPs can be used to visualise and analyze the sensitivity of the ResNet50 models to UAPs for each class label. We omit the analysis for SIN+2IN as its robustness to universal attacks is highly similar to that of IN.

In this experiment, we generate targeted UAPs at $\varepsilon = 10$ for each of the 1,000 ImageNet class labels for each of the ResNet50 models and evaluate their effectiveness by measuring their targeted success rate (TSR). We demonstrate how targeted UAPs can be used as a tool to measure and identify the most vulnerable features and class labels of the ResNet50 models under ImageNet and Stylized-ImageNet training.

Table 4.1 shows that the TSR ranges from approximately 30-80% and the majority of the 1,000 targeted UAPs for each model has 50-70% TSR. Figure 4.5 shows the distribution of TSR for on IN, SIN, and SIN+IN. What stands out is that SIN+IN appears to be slightly more robust to targeted UAPs as its TSR distribution is shifted to the left when compared to IN and SIN, this shift indicates that the average and median TSR for SIN+IN is lower. Figure 4.5 shows

that the TSR ranges from approximately 30-80% and the majority of the 1,000 targeted UAPs for each model are centered the around 50-70% TSR range. What stands out is that SIN+IN appears to be slightly more robust to targeted UAPs as its TSR distribution is shifted to the left when compared to IN and SIN. Since SIN+IN blends elements from both datasets, its larger and more diverse training may have made it more robust to targeted UAPs.

Untargeted vs. Targeted UAPs. Another interesting aspect is the visual similarity between untargeted UAPs and their corresponding targeted UAP from the same model. For example, the targeted UAP for *theatre curtain* (#854) as seen in Figure 4.4 is similar in appearance to the untargeted UAP for SIN+IN that can be seen in Figure 4.1. The untargeted UAP has 34% of the outputs classified *theatre curtain* while it is 81% for the targeted UAP. This difference could be attributed to the untargeted UAP optimizing solely for the untargeted success rate. The visual similarities is likely due to *theatre curtain* being a such a dominant class for SIN+IN. A similar phenomenon occurs for *brain coral* on IN and *dome* on SIN.

4.3.4 ImageNet vs. Stylized-ImageNet Training

In this section, we analyse the class labels associated with the most and least effective targeted UAPs on all three models IN, SIN, and SIN+IN. Generating targeted UAPs for each model and for each of the ImageNet class labels reveals a rich set of visualizations. The TSR of targeted UAPs for each class label allows us to quantitatively measure, compare, and contrast the features that these models learned in terms of the dataset class labels rather than vague notions of shape or texture bias. We analyse two categories of class labels below.

Table 4.2: Top and bottom three class labels ranked according to their average targeted success rate (in %) across the ResNet50 models.

Class ID	Label	Average TSR (%)
854	theater curtain	82.86
611	jigsaw puzzle	81.09
879	umbrella	79.82
960	chocolate sauce	39.65
74	Aranea diademata	37.15
72	Argiope aurantia	31.04

High TSR on all three models. Table 4.2 shows the top and bottom three class labels based on their average TSR across the models. The class labels *theater curtain* and *jigsaw puzzle* do not have distinct shapes and are primarily identified by more their textural pattern. The class *umbrella* is a simple object that can be identified by its shape. What is interesting is that the targeted UAPs from each of the three models are all visually similar, as can be seen for *theater curtain* in Figure 4.4. That these class labels are vulnerable to targeted UAPs on all three models indicates that the models learn similar features for identifying these classes on both ImageNet and Stylized-ImageNet. Especially since SIN+IN shares features with both IN and SIN individually.

Low TSR on all three models. Looking at the classes with the least effective targeted UAPs, *chocolate sauce* does not have a clear shape and is often a solid brown colour. *Aranea diademata* and *Argiope aurantia* are both varieties of garden spiders, most notably differing in their texture. A spider can be identified by its shape, with the eight thin appendages, while the specific species is identified through its texture. The low TSR for targeted UAPs on all three classifiers shows that this class is difficult to activate, which indicates difficulty in learning and a low sensitivity to this combination of shape and texture.

This analysis reveals the class labels which are the most and least difficult for all three models to learn. Thus, targeted UAPs can be used to assess the difficulty that models have in learning certain class labels, or it may be indicative on the quality of data points in the training set. That these undefended do not have as strong a targeted UAP response may mean that the models have not learned any distinguishing features for these class labels.

4.4 Mitigating Attacks with Ensembles

Based on results from previous sections, we see that models trained on the different datasets Stylized-ImageNet and ImageNet have dissimilarities in their vulnerable features. This can be seen by the performance of the UAPs across the three models (IN, SIN, SIN+IN), as they do not transfer perfectly, their targeted UAPs evidence diversity in visual appearances, and variance

in effectiveness of vulnerable features learned. Thus, we test an ensemble voting mechanism to improve the overall robustness to universal attacks and maintain clean performance. The idea is that for a UAP to fool the entire ensemble, it will have to fool majority of the models, and due to the diversity in the features learned, this is unlikely to succeed. At the same time, all models are trained to have high accuracy, so the majority classify clean inputs correctly.

Ensembles. Ensembles are created by combining the outputs of multiple base learners (models) to generate an improved prediction. To combine IN, SIN, and SIN+IN, we consider two ensemble voting schemes: a *hard* voting scheme where the ensemble outputs the majority class label, and a *soft* voting scheme where the ensemble outputs the class label with the highest average probability after the softmax layer. For both voting schemes, the ensemble outputs no prediction if there is no majority.

Experimental setup. To compare the ensemble scheme with the original models, we measure their accuracy on the clean dataset and the lowest accuracy achieved on any of the UAPs generated so far at $\epsilon = 10$. This is to consider the worst-case robustness of these models to universal attacks. For IN, SIN, and SIN+IN, this would be their accuracy against their corresponding untargeted UAP in a PK scenario.

Table 4.3: Accuracy (in %) of ResNet50 models and ensembles for the clean ImageNet validation set and on the worst-case UAP at $\epsilon = 10$. Highest model accuracy is highlighted.

Model	Clean	Worst-case UAP
IN	76.13	7.50
SIN	60.18	7.65
SIN+IN	74.59	8.33
Ensemble (hard)	73.24	17.25
Ensemble (soft)	76.02	20.37

Results. In Table 4.3, both ensembles improve on the robustness to the worst universal attack when compared to the individual models, with soft voting achieving the highest accuracy of 20.37%, more than 10 points better than the individual models. The soft ensemble also achieves a clean accuracy of 76.02%, which is very close to the highest clean accuracy of IN at 76.13%. This demonstrates that the ensemble with soft voting is able to maintain clean accuracy while greatly improving adversarial robustness to universal attacks. Although the adversarial robust-

ness of the ensemble could still be improved, this is a promising direction and demonstrates that ensemble methods can improve adversarial robustness work while maintaining clean accuracy.

4.5 Conclusion

In this chapter, we demonstrated how UAPs can be used to better understand and evaluate the robustness of DNNs with differing degrees of shape-based and texture-based training. Rather than relying on informal notions of “texture” or “shape”, we are able to more rigorously quantify the features that the models have learned and are vulnerable to in terms of the dominant labels of their UAPs and the effectiveness of their targeted UAPs for each of the 1,000 ImageNet class labels. To summarise, we enumerate the main findings of this chapter:

1. We investigate the adversarial robustness of models trained on Stylized-ImageNet. Our results show that shape-based training does not mitigate vulnerabilities to UAPs, but instead shifts the set of features that the models use for classification.
2. We demonstrate how our Dominant Label Analysis of UAPs can be used to quantify the most and least vulnerable classes of each model. Thus, moving away from more subjective notions of texture and shape.
3. Finally, we propose ensembles of texture and shape-biased models to maintain strong performance and improve robustness to universal attacks at minimal cost to clean accuracy.

Towards the end of this chapter, we were able to show that ensembles made of base learners trained on the different datasets IN, SIN, and SIN+IN could result in an aggregate model that achieves both superior performance and robustness. Although the robustness to UAPs is not absolute, it is a considerable improvement given that there was minimal decline in clean accuracy. In the next chapter, we work towards improving this accuracy-robustness trade-off further by exploring Jacobian Regularisation.

Chapter 5

Mitigating UAPs Through Jacobian Regularisation

An understudied aspect of existing defences against UAPs is the trade-off between its clean accuracy to its robustness against UAPs. A model with increased robustness to UAPs is indeed desirable, but a decrease in clean accuracy would translate to reduced utility for the model in practice. In Section 2.4, we showed that existing defences often neglect the costs they incur on the model’s performance for the original task. For example, defences like Adversarial Training (AT) can improve robustness but at the cost of greatly reduced accuracy [42, 71, 114]. A number of studies have investigated the use of *Jacobian Regularisation (JR)* to improve the stability of machine learning algorithms [71, 115, 116, 114, 117, 118]. JR has two characteristics that make it a good candidate for improving both accuracy and robustness:

- JR has been shown to be an effective model-agnostic way of increasing classification margins and model generalisation. These factors correlate with improved accuracy.
- In contrast to more empirically-based methods such as AT that require many parameters to be tuned, JR is simpler to tune as it has only one regularisation parameter. This allows for the more measured control of any accuracy-robustness trade-offs involving JR.

Existing work so far have considered JR only for the stability of models against input-dependent

adversarial perturbations (“per-input”)—where adversarial perturbations need to be crafted *for each single input*. This leaves the topic of JR’s effect on UAP robustness unexplored.

In this chapter, we study JR in the context of UAP robustness, derive formal proofs showing its efficacy, and develop a defence based on JR that improves on the accuracy-robustness trade-off for UAPs. In Sections 5.2 and 5.3, we expand on the theoretical formulation of JR to UAPs and derive upper bounds on the effectiveness of UAPs based on properties of the Jacobian matrices on inputs. In Sections 5.4 and 5.5, we propose a novel approach in Jacobian Ensembles—a combination of Jacobian Regularisation and model ensembles to significantly increase the robustness against UAPs whilst maintaining or improving model accuracy. We provide both theoretical proofs and empirical results that support the use of JR and Jacobian Ensembles.

5.1 Background & Notations

5.1.1 Preliminary Definitions

Before moving forward, we introduce the following notations. We then reiterate some basic properties of matrices and their norms:

- $\mathbf{A}'$ denotes the transpose of $\mathbf{A}$.
- We write the entries of a matrix $\mathbf{A}$ using lowercase letters a_{ij} , where i and j indicates the row and column respectively.
- For a rectangular matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, let $\sigma_i(\mathbf{A})$ denote the i -th largest singular value of $\mathbf{A}$. Note that this is equal to the square root of the i -th largest eigenvalue of $\mathbf{A}\mathbf{A}'$. For the following definitions, let r be the rank of $\mathbf{A}$.
- For matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$, define the inner product in $\mathbb{R}^{m \times n}$ as:

$$\langle \mathbf{A}, \mathbf{B} \rangle = \text{Tr}(\mathbf{A}'\mathbf{B}) = \sum_{i=1}^m \sum_{j=1}^n a_{ij}b_{ij} \quad (5.1)$$

where $\text{Tr}(\cdot)$ is the trace. This inner product is associated with the Frobenius norm $\|\cdot\|_F$.

- The Frobenius norm is given by:

$$\|\mathbf{A}\|_F = \sqrt{\langle \mathbf{A}, \mathbf{A} \rangle} = \sqrt{\text{Tr}(\mathbf{A}'\mathbf{A})} = \left(\sum_{i=1}^m \sum_{j=1}^n a_{ij}^2 \right)^{\frac{1}{2}} = \left(\sum_{i=1}^r \sigma_i^2(\mathbf{A}) \right)^{\frac{1}{2}} \quad (5.2)$$

- The induced 2-norm of a matrix is equal to its largest singular value:

$$\|\mathbf{A}\|_2 = \sigma_1(\mathbf{A}) \quad (5.3)$$

- These two matrix norms we defined are related by the following inequalities:

$$\|\mathbf{A}\|_2 \leq \|\mathbf{A}\|_F \leq \sqrt{r} \|\mathbf{A}\|_F \leq r \|\mathbf{A}\|_2 \quad (5.4)$$

5.1.2 Jacobian Regularisation

For the purposes of our formal proof, we update some definitions and notations. Let $f : \mathcal{X} \subset \mathbb{R}^D \rightarrow \mathbb{R}^C$ denote the logits of a piece-wise linear classifier which takes as input $\mathbf{x} \in \mathcal{X}$. D represents the input dimensions and C represents the number of output class labels. Given that $f(\mathbf{x})$ is the logit output of the classifier for input $\mathbf{x}$, we write $\mathbf{J}_f(\mathbf{x})$ to denote the input-output Jacobian of f at $\mathbf{x}$. We can linearise f within a neighbourhood around $\mathbf{x}$ as follows using the Taylor series expansion:

$$f(\mathbf{x} + \delta) = f(\mathbf{x}) + \mathbf{J}_f(\mathbf{x})\delta + O(\delta^2) \quad (5.5)$$

For a sufficiently small neighbourhood $\|\delta\|_p \leq \varepsilon$ with $\varepsilon > 0$, the higher order terms of δ can be neglected and the stability of the prediction is determined by the Jacobian.

$$f(\mathbf{x} + \delta) \simeq f(\mathbf{x}) + \mathbf{J}_f(\mathbf{x})\delta \quad (5.6)$$

and equivalently, for any q -norm, we have:

$$\|f(\mathbf{x} + \delta) - f(\mathbf{x})\|_q \approx \|\mathbf{J}_f(\mathbf{x})\delta\|_q \quad (5.7)$$

The intuition is that an attacker can maximise the change in $f(x)$ if $\mathbf{v}$ is a singular vector for $\mathbf{J}_f(\mathbf{x})$. To improve the stability of model outputs to small perturbations δ , existing works have proposed regularizing the Frobenius norm [71, 115, 116] or the Spectral norm [114, 117, 118] of this data-dependent Jacobian $\mathbf{J}_f(\mathbf{x})$ for each input. It was also shown that adversarial perturbations align with the dominant singular vectors of these Jacobian matrices [114].

Now for training with Jacobian Regularisation (JR), we optimise the following joint loss and use the algorithm as proposed by [71]:

$$\mathcal{L}_{\text{joint}}(\boldsymbol{\theta}) = \mathcal{L}_{\text{train}}(\{\mathbf{x}_i, \mathbf{y}_i\}_i, \boldsymbol{\theta}) + \frac{\lambda_{\text{JR}}}{2} \left(\frac{1}{B} \sum_i \|\mathbf{J}(\mathbf{x}_i)\|_F^2 \right) \quad (5.8)$$

where $\boldsymbol{\theta}$ represent the parameters of the model, $\mathcal{L}_{\text{train}}$ is the standard cross-entropy training loss, $\{\mathbf{x}_i, \mathbf{y}_i\}$ are input-output pairs from the mini-batch, and B is the mini-batch size. This optimization uses a regularisation parameter λ_{JR} , which lets us adjust the trade-off between regularisation and classification loss.

The main idea is to reduce the Frobenius norm of the input-output Jacobian $\|\mathbf{J}(\mathbf{x}_i)\|_F$ to decrease the model's sensitivity to small perturbations such as UAPs. However, it can often simultaneously decrease the model's clean accuracy especially for large values of λ_{JR} [71].

Although Khrulkov & Oseledets considered Jacobians in the context of UAPs [3], they only focused on the computation of δ as an attack and did not perform any theoretical or empirical analysis for mitigating the effects of UAPs. Prior studies that explore JR focused solely on improving robustness to single-input perturbations and did not explain nor consider the effectiveness of JR for UAPs. Thus, we extend these formulations to achieve a more concrete formulation and understanding for how JR mitigates UAPs.

5.1.3 UAPs for Jacobians

We now formulate UAPs in the context of JR. To generate an adversarial perturbation for a small $\varepsilon > 0$, we want the δ that maximises the right hand side of Equation 5.7 in order to sufficiently change the original output and fool the model. With constraint $\|\delta\|_p \leq \varepsilon$, this is equivalent to finding the (p, q) singular vector for $\mathbf{J}_f(\mathbf{x})$ [3]. When computing a UAP δ that fools the model across multiple inputs $\{\mathbf{x}_i\}_{i=1}^N$, one would optimise:

$$\max_{\delta: \|\delta\|_p=1} \sum_{i=1}^N \|\mathbf{J}_f(\mathbf{x}_i)\delta\|_q \quad (5.9)$$

This extends the intuition from Equation 5.7 to many inputs, and due to the homogeneity of the norm, it is sufficient to solve this for a fixed norm value $\|\delta\|_p = 1$ [3]. The solution to δ for Equation 5.9 is equivalent to finding the (p, q) singular vector for the **stacked Jacobian** matrix $\bar{\mathbf{J}}_N$, the matrix formed by vertically stacking the Jacobians of the first N inputs.

$$\max_{\delta: \|\delta\|_p=1} \|\bar{\mathbf{J}}_N \delta\|_q \quad \text{where} \quad \bar{\mathbf{J}}_N = \begin{bmatrix} \mathbf{J}_f(\mathbf{x}_1) \\ \mathbf{J}_f(\mathbf{x}_2) \\ \vdots \\ \mathbf{J}_f(\mathbf{x}_N) \end{bmatrix} \quad (5.10)$$

This is the same as the data-dependent (p, q) -operator norm of $\mathbf{J}_f(\mathbf{x})$ defined as [114]:

$$\|\mathbf{J}_f(\mathbf{x})\|_{p,q} = \max_{\mathbf{v}: \|\mathbf{v}\|_p=1} \|\mathbf{J}_f(\mathbf{x})\mathbf{v}\|_q \quad (5.11)$$

which is a data-dependent measure of the maximal amount of signal-gain that can be induced when propagating a norm-bounded input vector through the linearised network.

5.2 Bounds for the Stacked Jacobian

It is in restricting the Frobenius norm of this stacked Jacobian $\bar{\mathbf{J}}_N$ that we will improve the overall stability and robustness of the classifier for UAP attacks. In this section, we will derive upper bounds for this metric to show that JR improves robustness to UAPs.

5.2.1 Upper Bound for the Frobenius Norm

To obtain an upper bound for the (p, q) -operator norm shown in Equation 5.10, note that it is bounded above by its Frobenius norm denoted by $\|\bar{\mathbf{J}}_N\|_F$:

$$\|\bar{\mathbf{J}}_N \delta\|_q \leq \|\bar{\mathbf{J}}_N\|_F \|\delta\| \quad (5.12)$$

Thus, mitigating the effectiveness of a UAP across multiple inputs can be achieved by limiting the Frobenius norm of the stacked Jacobian $\|\bar{\mathbf{J}}_N\|_F$. Before proceeding, let us define the inner product induced by the Frobenius norm for two real matrices. Given $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$, let the inner product in $\mathbb{R}^{m \times n}$ be defined as:

$$\langle \mathbf{A}, \mathbf{B} \rangle = \text{Tr}(\mathbf{A}'\mathbf{B}) = \sum_{i=1}^m \sum_{j=1}^n a_{ij} b_{ij} \quad (5.13)$$

where $\mathbf{A}'$ denotes the transpose of $\mathbf{A}$, the lowercase letters a_{ij} are the entries of the matrix $\mathbf{A}$, and $\text{Tr}(\cdot)$ is the trace. This inner product is associated with the Frobenius norm $\|\cdot\|_F$. Now we introduce the following proposition.

Proposition 1. *For matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$, we have:*

$$\langle \mathbf{A}, \mathbf{B} \rangle \leq \|\mathbf{A}\|_F \|\mathbf{B}\|_F \quad (5.14)$$

with equality if and only if $\mathbf{A}$ and $\mathbf{B}$ share singular directions and their singular values satisfy $\sigma_i(\mathbf{A}) = s \cdot \sigma_i(\mathbf{B})$ for all i for a constant scalar $s > 0$, where $\sigma_i(\cdot)$ is the singular value that corresponds to the i -th largest singular value.

Proof. Consider the singular value decomposition of $\mathbf{A} = \mathbf{U}_A \mathbf{\Sigma}_A \mathbf{V}'_A$ and $\mathbf{B} = \mathbf{U}_B \mathbf{\Sigma}_B \mathbf{V}'_B$, where $\mathbf{U}_A, \mathbf{U}_B, \mathbf{V}_A, \mathbf{V}_B$ are orthogonal matrices and $\mathbf{\Sigma}_A, \mathbf{\Sigma}_B$ are diagonal matrices whose diagonal entries $\sigma_i(\mathbf{A})$ and $\sigma_i(\mathbf{B})$ are non-negative and in descending order. Let $r = \max(\text{rank}(\mathbf{A}), \text{rank}(\mathbf{B}))$.

$$\begin{aligned}
\langle \mathbf{A}, \mathbf{B} \rangle &= \text{Tr}(\mathbf{A}'\mathbf{B}) \\
&= \text{Tr}(\mathbf{V}_A \mathbf{\Sigma}'_A \mathbf{U}'_A \mathbf{U}_B \mathbf{\Sigma}_B \mathbf{V}'_B) \\
&= \text{Tr}(\mathbf{V}'_B \mathbf{V}_A \mathbf{\Sigma}'_A \mathbf{U}'_A \mathbf{U}_B \mathbf{\Sigma}_B) \quad \text{cyclic property of trace}
\end{aligned}$$

Note that since $\mathbf{U}_A, \mathbf{U}_B, \mathbf{V}_A, \mathbf{V}_B$ are all orthogonal matrices, $\|\mathbf{U}'_A \mathbf{U}_B\|_2 \leq \|\mathbf{U}'_A\|_2 \|\mathbf{U}_B\|_2 = 1$, and in a similar way, $\|\mathbf{V}'_B \mathbf{V}_A\|_2 \leq 1$.

$$\begin{aligned}
\langle \mathbf{A}, \mathbf{B} \rangle &= \text{Tr}(\mathbf{V}'_B \mathbf{V}_A \mathbf{\Sigma}'_A \mathbf{U}'_A \mathbf{U}_B \mathbf{\Sigma}_B) \\
&= \sum_{i=1}^r \sum_{j=1}^r z_{ij} \cdot \sigma_i(\mathbf{A}) \sigma_j(\mathbf{B}) \quad \text{where } \sum_{i=1}^r |z_{ij}| \leq 1, \sum_{j=1}^r |z_{ij}| \leq 1 \\
&\leq \sum_{i=1}^r \sigma_i(\mathbf{A}) \sigma_i(\mathbf{B}) \quad \text{equality} \iff z_{ij} \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases} \\
&\leq \left(\sum_{i=1}^r \sigma_i^2(\mathbf{A}) \right)^{\frac{1}{2}} \left(\sum_{i=1}^r \sigma_i^2(\mathbf{B}) \right)^{\frac{1}{2}} \quad \text{Cauchy-Schwarz Inequality} \\
&= \|\mathbf{A}\|_F \|\mathbf{B}\|_F \quad \square
\end{aligned}$$

The equality conditions for the above require $z_i = 1, \forall i$, which implies that $\mathbf{U}'_A \mathbf{U}_B$ and $\mathbf{V}'_B \mathbf{V}_A$ are identity matrices. This requires $\mathbf{U}_A = \mathbf{U}_B$ and $\mathbf{V}_A = \mathbf{V}_B$, i.e. $\mathbf{A}$ and $\mathbf{B}$ share the same singular vectors. Equality under Cauchy-Schwarz requires the singular values to be scalars of one another: $\sigma_i(\mathbf{A}) = s \cdot \sigma_i(\mathbf{B})$ for a constant scalar $s > 0, \forall i$.

We opted for the proof above to show conditions for equality, although a straightforward proof

for the inequality can also be written as follows:

$$\begin{aligned}
\langle \mathbf{A}, \mathbf{B} \rangle &= \sum_{i=1}^m \sum_{j=1}^n a_{ij} b_{ij} \\
&\leq \left(\sum_{i=1}^m \sum_{j=1}^n a_{ij}^2 \right)^{\frac{1}{2}} \left(\sum_{i=1}^m \sum_{j=1}^n b_{ij}^2 \right)^{\frac{1}{2}} && \text{Cauchy-Schwarz Inequality} \\
&= \|\mathbf{A}\|_F \|\mathbf{B}\|_F && \square
\end{aligned}$$

The equality conditions for the above requires $z_{ii} = 1, \forall i$ as the σ_i are in descending order. This implies that $\mathbf{U}'_A \mathbf{U}_B$ and $\mathbf{V}'_B \mathbf{V}_A$ are identity matrices, which requires $\mathbf{U}_A = \mathbf{U}_B$ and $\mathbf{V}_A = \mathbf{V}_B$, i.e. $\mathbf{A}$ and $\mathbf{B}$ share the same singular vectors. Equality under Cauchy-Schwarz requires the singular values to be scalars of one another: $\sigma_i(\mathbf{A}) = s \cdot \sigma_i(\mathbf{B})$ for the same scalar $s > 0, \forall i$.

This proposition is significant as it gives us upper bounds for the inner product and equality conditions to achieve this upper bound. Applying this result to the stacked Jacobian matrix $\bar{\mathbf{J}}_N$ gives us the following:

$$\begin{aligned}
\|\bar{\mathbf{J}}_N\|_F^2 &= \text{Tr}(\bar{\mathbf{J}}_N' \bar{\mathbf{J}}_N) \\
&= \text{Tr} \left(\sum_{i=1}^N \sum_{j=1}^N \mathbf{J}_f(\mathbf{x}_i)' \mathbf{J}_f(\mathbf{x}_j) \right) \\
&= \sum_{i,j} \text{Tr}(\mathbf{J}_f(\mathbf{x}_i)', \mathbf{J}_f(\mathbf{x}_j)) \\
&= \sum_{i,j} \langle \mathbf{J}_f(\mathbf{x}_i), \mathbf{J}_f(\mathbf{x}_j) \rangle && \text{Frobenius inner product} \\
&\leq \sum_{i,j} \|\mathbf{J}_f(\mathbf{x}_i)\|_F \|\mathbf{J}_f(\mathbf{x}_j)\|_F && \text{Proposition 1}
\end{aligned}$$

With equality if and only if, for all pairs of inputs $(\mathbf{x}_i, \mathbf{x}_j)$, we have $\mathbf{J}_f(\mathbf{x}_i)$ and $\mathbf{J}_f(\mathbf{x}_j)$ sharing singular vectors and their corresponding singular values are constant up to a fixed scalar $s > 0$.

Our result can be summarised with the following equation:

$$\|\bar{\mathbf{J}}_N\|_F \leq \left(\sum_{i,j} \|\mathbf{J}_f(\mathbf{x}_i)\|_F \|\mathbf{J}_f(\mathbf{x}_j)\|_F \right)^{\frac{1}{2}} \quad (5.15)$$

From a defence perspective, this shows that regularizing the Frobenius of the Jacobian for the $\mathbf{x}_i$ decreases the upper bound of the Frobenius norm of the stacked Jacobian and hinders the overall effectiveness of a UAP. Thus, data-dependent JR across inputs should make it more difficult to generate effective UAPs.

5.2.2 Measuring Alignment of Jacobians

To measure alignment between Jacobians of two distinct inputs, we use the **cosine similarity** between their respective Jacobians under the inner product induced by the Frobenius norm:

$$\text{sim}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\langle \mathbf{J}_f(\mathbf{x}_i), \mathbf{J}_f(\mathbf{x}_j) \rangle}{\|\mathbf{J}_f(\mathbf{x}_i)\|_F \|\mathbf{J}_f(\mathbf{x}_j)\|_F} \leq 1 \quad (5.16)$$

This is precisely the formula given in Proposition 1, with the above ratio equal to one if and only if the singular vectors of the Jacobians are the same. This shows us that alignment of Jacobians can be evaluated with this similarity measure, and combined with Equation 5.15, this ratio allows us to measure how closely two inputs share adversarial perturbations. Although the Jacobian is a first-order derivative, we show later that our Jacobian similarity measure also improves robustness against multi-step or iterative UAP attacks. Thus, demonstrating that it is an effective measure to determine the “universality” of adversarial vulnerability even against iterative adversaries like SGD.

Having a similarity measure like this is beneficial as this allows us to easily determine if two inputs are likely to share adversarial perturbations. This is more advantageous than manually generating adversarial perturbations for each pair of inputs as one would have to consider many additional attack parameters when generating adversarial attacks, including the ε bounds, chosen ℓ_p -norm, step size, number of attack iterations, and so on.

5.3 Robustness of Jacobian Regularised Models

Having formally shown how JR can improve robustness to UAPs, we now evaluate its UAP robustness empirically on common benchmark datasets.

5.3.1 Experimental Setup

Models & Datasets. We consider the benchmark datasets MNIST [119] and Fashion-MNIST [120]. These are widely-used image classification datasets, each with 10 classes, whose images are 28 by 28 pixels, and their pixel values range from 0 to 1. For the neural network architecture, we use a modernised version of LeNet-5 [119, 71] as it is a commonly used benchmark neural network. We refer to this model as LeNet.

Threat Model. We evaluate the UAP robustness of these models to perfect-knowledge attacks, so we use SGD with 100 iterations and a batch size of 200 to generate UAP attacks. Perturbations are applied under ℓ_∞ -norm constraints. The ε we consider in our attacks for this norm are from 0.1 to 0.3, this perturbation magnitude is equivalent to 10%-30% of the maximum total possible change in pixel values. For adversarial goals, we generate both untargeted and targeted attacks. For targeted UAPs, we generate one UAP for each of 10 classes of each dataset. Evaluations are done on the entire 10,000 sample test set.

The effectiveness of untargeted attacks are measured with the UER, defined as the proportion of inputs that are misclassified. Targeted UAPs are evaluated according to their TSR, the proportion of inputs classified as the target class. For more details on these metrics, we refer the reader to how our threat models are defined as in Section 2.2.

5.3.2 Jacobian Regularisation Mitigates UAPs

Regular training without JR (i.e. $\lambda_{JR} = 0$) achieves 99.08% and 90.84% test accuracy on MNIST and Fashion-MNIST respectively. Figure 5.1 shows that increasing the weight of JR

decreases the resulting model’s test accuracy. Note, however, that this decrease appears to be negligible for very small $\lambda_{JR} \leq 0.1$.

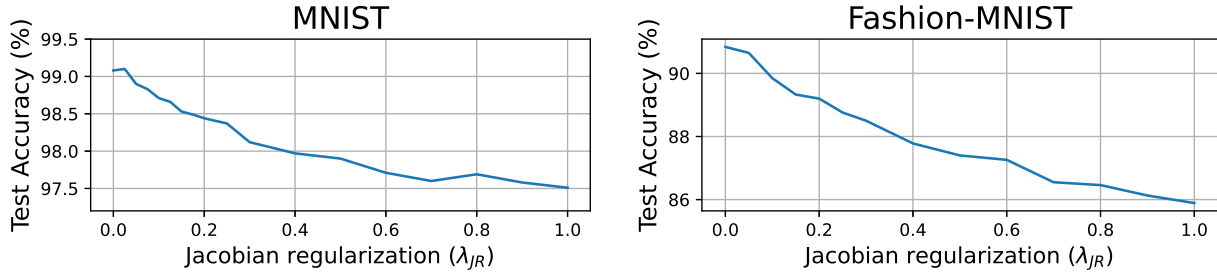


Figure 5.1: Test accuracy of LeNet on MNIST (left) and Fashion-MNIST (right) for various Jacobian Regularisation strengths λ_{JR} .

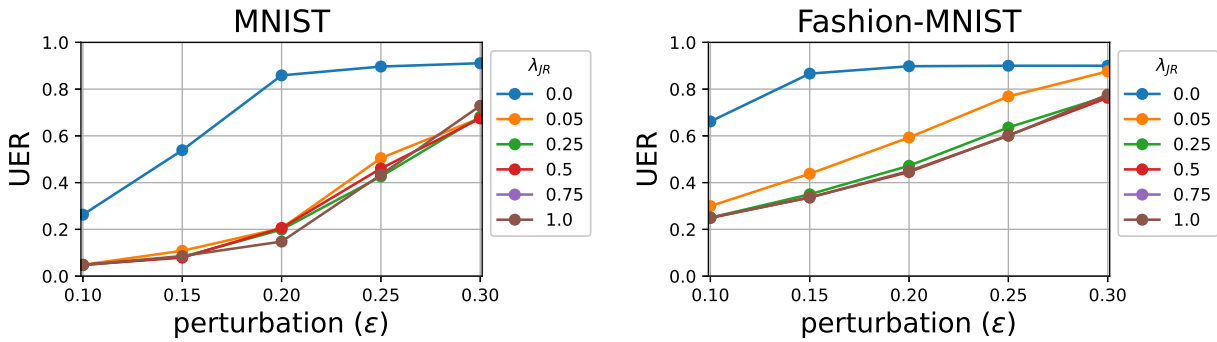


Figure 5.2: Effectiveness of untargeted UAPs for various ℓ_∞ -norm perturbation constraints ϵ . Plots are shown for various models with different degrees of Jacobian Regularisation.

Untargeted UAPs. Figure 5.2 presents the effectiveness of our untargeted UAP attacks on different LeNet with varying JR strengths. The regularly trained model is especially vulnerable to UAP attacks on both datasets, with untargeted UAPs achieving above 80% UER for $\epsilon \geq 0.2$ on both datasets. On MNIST, UAP attacks seem to gain reasonable success only after $\epsilon \geq 0.25$. This is permissible as the adversary perturbs the input by 25% of its maximum possible value in this case, which entails an enormous change. What is striking is that JR has a protective effect for $\epsilon \leq 0.2$, even for small amounts of regularisation at $\lambda_{JR} = 0.05$. Here, UAP effectiveness is down from 80% to 20% at $\epsilon = 0.2$. Increasing the strength of the regularisation likely has diminishing returns for robustness as stronger regularisation also begins to damage clean accuracy, and thus the model’s generalization. Fashion-MNIST can be seen to be less robust since it begins with a lower clean accuracy at around 91%. This means that the model is overall less robust to begin with than the model trained for MNIST, so we can expect it to be less

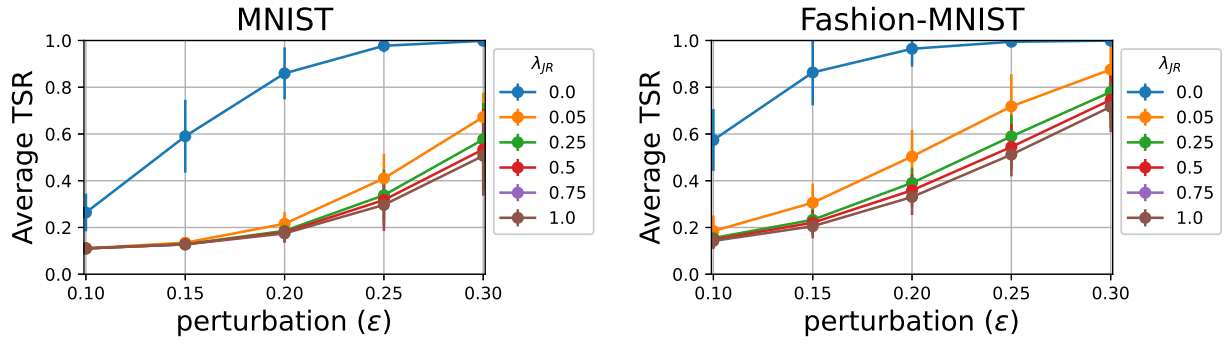


Figure 5.3: Average Targeted Success Rate (TSR) of targeted UAPs generated for each class, with error bars showing standard deviation across UAPs for different classes. Plots are shown for various models with different degrees of Jacobian Regularisation.

robust to UAP attacks in general. Nonetheless, we still see a protective effect from JR for $\varepsilon \leq 0.15$ even with only a minor degree of regularisation $\lambda_{JR} = 0.05$.

Targeted UAPs. Figure 5.3 shows our results for the effectiveness of targeted UAPs. These plots follow a similar trend as with untargeted UAPs, suggesting that JR is able to improve model robustness against a diverse array of UAP attacks and not only against untargeted UAPs. Even a minor amount of regularisation in $\lambda_{JR} = 0.05$ provides up to a 4-times decrease in effectiveness of UAPs while maintaining the model’s performance on the clean test set, as seen in Table 5.1.

Table 5.1: Performance metrics (in %) of LeNet. Jacobian Regularisation (JR) uses $\lambda_{JR} = 0.05$. UAP evaluations are for ℓ_∞ -norm attacks at $\varepsilon = 0.2$ for MNIST and $\varepsilon = 0.15$ for Fashion-MNIST. Lowest values indicate the best robustness and are highlighted.

	MNIST			Fashion-MNIST		
	Standard	UAT [53]	JR	Standard	UAT [53]	JR
Test Error	0.92	1.81	0.90	9.16	16.66	9.15
Untargeted UER	85.88	27.49	20.47	86.63	34.10	29.96
Average TSR	85.94	24.05	21.57	86.33	26.64	30.59

Comparison with Adversarial Training. We compare JR with Universal Adversarial Training (UAT) [53], where adversarial training is done on UAPs. UAT models in Table 5.1 are trained on $\varepsilon = 0.2$ and $\varepsilon = 0.15$ adversaries for MNIST and Fashion-MNIST respectively. Although UAT improves robustness to UAPs compared to standard training, *it doubles the test error on both clean datasets*. In contrast, JR achieves better robustness than UAT without

damaging clean accuracy. Adversarial training relies on training against specific UAP perturbations. The heuristic quality of UAT makes improving robustness against all possible perturbations computationally difficult. Our results show that regularizing a more general property of the model, in the norm of the Jacobian, leads to better robustness while maintaining accuracy.

5.3.3 Jacobian Alignment of Input Pairs

We now investigate how the cosine similarity of input Jacobians as introduced in Equation 5.16 correlates with the models' robustness to UAPs. We consider LeNet with Jacobian Regularisation ($\lambda_{JR} = 0.05$) and without ($\lambda_{JR} = 0.0$). The performance of the models on the test sets is the same as the ones in Table 5.1. We take a random subset of 1,000 test set images with a uniform distribution on the output classes and measure the pairwise similarity. The results we

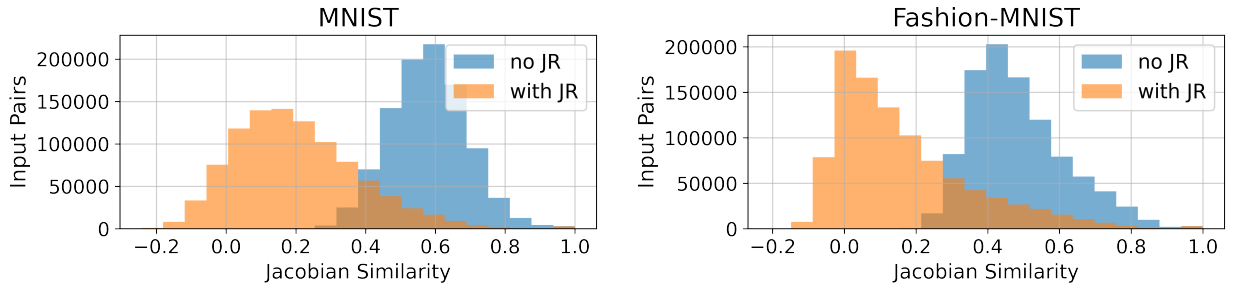


Figure 5.4: Jacobian similarity for pairs of inputs on MNIST (left) and Fashion-MNIST (right) for LeNet with and without Jacobian Regularisation (JR). Median similarity values on MNIST are 0.18 and 0.58; and on Fashion-MNIST are 0.11 and 0.46 with and without JR respectively.

have found in Figure 5.4 verifies several of our intuitions on generating UAPs:

- Regularising the Frobenius (Spectral) norm of the Jacobian *for any* of the $\mathbf{x}_i$ decreases the Frobenius norm of the stacked Jacobian, and thus hinders the overall effectiveness of a shared adversarial perturbation.
- Increasing the number of points we wish to evade (i.e. increasing n for $\{\mathbf{x}_i\}_{i=1}^n$) leads to weaker upper bounds for the stacked Jacobian, and thus an overall less effective shared adversarial perturbation. This is because the equality conditions require the singular directions of Jacobians to align.

5.4 Jacobian Ensembles

In the ongoing effort to defend models against adversarial attacks, it is often difficult to reconcile the trade-offs in model accuracy and robustness to attacks. Additionally, it is this trade-off aspect of existing defences that is not sufficiently explored in existing work. Although we have just shown that JR on its own can improve robustness against UAPs with minimal degradation of performance, we still see that JR can damage the clean accuracy at larger amounts of regularization [42, 71, 114]. *Model ensembles* on the other hand has widely been shown to achieve better classification performance and stability than a single classifier [121, 122]. Ensembles are created by combining the outputs of multiple base learners to generate an improved prediction. Thus, we propose combining JR and model ensembles during training to create *Jacobian Ensembles*. JR is used to drastically improve the model’s robustness to UAPs while the ensemble methods stabilise the model’s predictions and improve its clean accuracy. JR and model ensembles individually each have been shown to improve UAP robustness but at some cost to clean accuracy [42, 123]. We show that Jacobian Ensembles greatly improve on the accuracy-robustness trade-off when compared to either JR or model ensembles individually. First, we theoretically show that increasing the number of base learners in a model ensemble improves the expected robustness of classifiers. Then, we empirically verify our theoretical findings by applying JR with popular ensemble methods bagging [124], snapshot ensembles [125], soft voting [122] to DNNs trained on the popular benchmark datasets: MNIST [119], Fashion-MNIST [120] and then evaluating their robustness against UAPs.

5.4.1 Model Ensembles

An ensemble consists of combining multiple classifiers (base learners) to obtain a resulting ensemble that has better accuracy or predictive performance on aggregate than any individual base learner. In practice, it is widely accepted that combining multiple classifiers can achieve better classification performance than a single “best” classifier [121, 122]. Ensembles are typically generated in two ways: sequentially and in parallel. After generating the base learners

the combination of their outputs is taken rather than choosing a single “best” learner [122].

In this section, we will analyze ensemble methods that aggregate their base learners in a convex combination. Formally, we define an ensemble $\mathcal{F}$ as a convex combination of M base learners f_i : $\mathcal{F}(x) = \sum_{i=1}^M c_i f_i(x)$ where $\sum_{i=1}^M c_i = 1$ and $0 < c_i < 1, \forall i$. This is typical as many ensemble methods aggregate their methods via averaging or a similar form of weighted sum [122]. Note however that this will exclude some boosting algorithms such as AdaBoost that are typically not convex combinations [126].

5.4.2 Bounds for UAP Robustness on Model Ensembles

In this section, we derive theoretical bounds for the expectation and variance of the Frobenius norm of the Jacobian of model ensembles. We only consider ensembles that take a convex combination of their base learners. Similar to [42], we restrict the Frobenius norm of the Jacobian of a model to improve robustness against UAPs.

We show that using model ensembles result in tighter bounds on the Frobenius norm of the Jacobian, suggesting improved robustness and stability versus a single classifier. Our main result in this section is **Theorem 5.1**, which shows that increasing the number of base learners decreases both the upper and lower bounds of the expectation and variance for the Frobenius norm of the ensemble’s Jacobian.

Proposition 2. *Let x_i be independent random variables drawn from a Normal distribution $x_i \sim \mathcal{N}(\mu, \sigma^2)$ for a fixed mean μ and variance σ^2 . Define $\bar{x} = \sum_{i=1}^M c_i x_i$ where $\sum_{i=1}^M c_i = 1$ and $0 < c_i < 1, \forall i$. We then have the following:*

$$\frac{\sigma^2}{M} \leq \sum_{i=1}^M c_i^2 \sigma^2 < \sigma^2 \quad (5.17)$$

Proof. By linearity of Normal distributions: $\bar{x} \sim \mathcal{N}(\mu, \sum_{i=1}^M c_i^2 \sigma^2)$. We then derive bounds for

$\sum_{i=1}^M c_i^2$ when $M \geq 2$:

$$\begin{aligned} \sum_{i=1}^M c_i^2 &= \sum_{i=1}^M c_i^2 + 2 \sum_{i=1}^M \sum_{j \neq i}^M c_i c_j - 2 \sum_{i=1}^M \sum_{j \neq i}^M c_i c_j \\ &= \left(\sum_{i=1}^M c_i \right)^2 - 2 \sum_{i=1}^M \sum_{j \neq i}^M c_i c_j \\ &= 1 - 2 \sum_{i=1}^M \sum_{j \neq i}^M c_i c_j \end{aligned}$$

Since $c_i c_j > 0$ for all pairs i, j , then it follows that we have the upper bound $\sum_{i=1}^M c_i^2 < 1$. Note that equality: $\sum_{i=1}^M c_i^2 = 1$ is only possible in the degenerate case (when $c_i = 1$ for exactly one i and $c_j = 0$ for $i \neq j$). For the lower bound, we use Cauchy-Schwarz inequality to get:

$$\left(\sum_{i=1}^M (c_i \cdot 1) \right)^2 \leq \left(\sum_{i=1}^M c_i^2 \right) \left(\sum_{i=1}^M 1^2 \right) = \left(\sum_{i=1}^M c_i^2 \right) \cdot M$$

The left hand side reduces to 1, so it follows that $\sum_{i=1}^M c_i^2 \geq \frac{1}{M}$ with equality when $c_i = \frac{1}{M}$ for all i . Multiplying all sides with σ^2 gives the desired bounds. $\square$

Let $\mathcal{F}$ be an ensemble of M base learners f_i : $\mathcal{F}(x) = \sum_{i=1}^M c_i f_i(x)$ where $\sum_{i=1}^M c_i = 1$ and $0 < c_i < 1, \forall i$. Let $\mathbf{J}_{\mathcal{F}}$ denote the Jacobian of $\mathcal{F}$ and $\mathbf{J}_i$ the Jacobian of f_i for all i . It follows that $\mathbf{J}_{\mathcal{F}} = \sum_{i=1}^M c_i \mathbf{J}_i$.

Theorem 5.1. *Let each matrix $\mathbf{J}_i \in \mathbb{R}^{C \times D}$ be comprised of the independent random variables ${}_i a_{pq} \sim \mathcal{N}(\mu, \sigma^2)$, where ${}_i a_{pq}$ is the element on the p -th row and q -th column of matrix $\mathbf{J}_i$. It follows that their convex combination $\mathbf{J}_{\mathcal{F}}$ satisfies:*

$$CD \left(\frac{\sigma^2}{M} + \mu^2 \right) \leq E(\|\mathbf{J}_{\mathcal{F}}\|_F^2) < E(\|\mathbf{J}_i\|_F^2) \quad (5.18)$$

$$CD \left(\frac{4\mu^2\sigma^2}{M} + \frac{2\sigma^4}{M^2} \right) \leq \text{Var}(\|\mathbf{J}_{\mathcal{F}}\|_F^2) < \text{Var}(\|\mathbf{J}_i\|_F^2) \quad (5.19)$$

Proof. Taking the square of Frobenius norm, we have the following for the Jacobian of a model:

$$\|\mathbf{J}_i\|_F^2 = \sum_{p=1}^C \sum_{q=1}^D |{}_i a_{pq}|^2$$

The moments of $\|\mathbf{J}_i\|_F$ are proportional to the moments of the random variables ${}_i a_{pq}^2$. These follow a chi-squared distribution with 1 degree of freedom, and have expectation and variance:

$$\begin{aligned} \mathbb{E}({}_i a_{pq}^2) &= \text{Var}({}_i a_{pq}) + [\mathbb{E}({}_i a_{pq})]^2 \\ &= \sigma^2 + \mu^2 \\ \text{Var}({}_i a_{pq}^2) &= \mathbb{E}({}_i a_{pq}^4) - [\mathbb{E}({}_i a_{pq}^2)]^2 \\ &= \mu^4 + 6\mu^2\sigma^2 + 3\sigma^4 - \mu^4 - 2\mu^2\sigma^2 - \sigma^4 \\ &= 4\mu^2\sigma^2 + 2\sigma^4 \end{aligned}$$

To summarise:

$$\begin{aligned} \mathbb{E}({}_i a_{pq}^2) &= \sigma^2 + \mu^2 \\ \text{Var}({}_i a_{pq}^2) &= 4\mu^2\sigma^2 + 2\sigma^4 \end{aligned}$$

Define $\bar{a}_{pq} = \sum_{i=1}^M c_i {}_i a_{pq}$, the elements of $\mathbf{J}_{\mathcal{F}}$. Note that $\bar{a}_{pq} \sim \mathcal{N}(\mu, \sum_{i=1}^M c_i^2 \sigma^2)$. Thus for the ensemble model's Jacobian, we have:

$$\|\mathbf{J}_{\mathcal{F}}\|_F^2 = \sum_{p=1}^C \sum_{q=1}^D |\bar{a}_{pq}|^2$$

It is clear that the moments of $\|\mathbf{J}_{\mathcal{F}}\|_F$ are proportional to that of $\bar{a}_{pq}^2$. These random variables follow a chi-squared distribution with M degrees of freedom, and have the following expectation and variance:

$$\begin{aligned} \mathbb{E}(\bar{a}_{pq}^2) &= \sum_{i=1}^M c_i^2 \sigma^2 + \mu^2 \\ \text{Var}(\bar{a}_{pq}^2) &= 4\mu^2 \sum_{i=1}^M c_i^2 \sigma^2 + 2 \left(\sum_{i=1}^M c_i^2 \sigma^2 \right)^2 \end{aligned}$$

Applying **Proposition 2**, we then have the following bounds for the expectation and variance

for these random variables:

$$\begin{aligned}\frac{\sigma^2}{M} + \mu^2 &\leq \mathbb{E}(\bar{a}_{pq}^2) < \mathbb{E}(a_{pq}^2) \\ \frac{4\mu^2\sigma^2}{M} + \frac{2\sigma^4}{M^2} &\leq \text{Var}(\bar{a}_{pq}^2) < \text{Var}(a_{pq}^2)\end{aligned}$$

As the random variables are independently drawn, our desired result follows:

$$\begin{aligned}CD\left(\frac{\sigma^2}{M} + \mu^2\right) &\leq \mathbb{E}(\|\mathbf{J}_{\mathcal{F}}\|_F^2) < \mathbb{E}(\|\mathbf{J}_i\|_F^2) \\ CD\left(\frac{4\mu^2\sigma^2}{M} + \frac{2\sigma^4}{M^2}\right) &\leq \text{Var}(\|\mathbf{J}_{\mathcal{F}}\|_F^2) < \text{Var}(\|\mathbf{J}_i\|_F^2)\end{aligned}$$

□

Conclusion. From these derivations, ensembles decrease both the expected value and variance of the Jacobian’s Frobenius norms when compared to that of a single model’s. As M increases, the lower bounds of both the expectation and variance decreases.

Averaging is one of the most common methods for aggregating base learner outputs in an ensemble [121, 122], so it is important to consider this case. When the ensemble is done via averaging, i.e. $c_i = \frac{1}{M}$ for all i , this achieves the equality condition for the lower bounds of both $\mathbb{E}(\bar{a}_{pq}^2)$ and $\text{Var}(\bar{a}_{pq}^2)$. Therefore, increasing the number of models in the ensemble *strictly decreases* the expectation and variance of the ensemble’s Jacobian’s norm $\|\mathbf{J}_{\mathcal{F}}\|_F$.

This theoretical result gives us the motivation on how model ensembles also improve the stability of models and thus their robustness to UAPs. We show this by deriving the above bounds on the Frobenius norm on their Jacobian. As model ensembles have also been shown to have improved performance over a single classifier, this makes it an ideal candidate for improving both model accuracy and robustness.

5.5 Improved Trade-offs with Jacobian Ensembles

Now that we have shown formally how ensembles can improve robustness to UAPs, we combine JR with ensembles to empirically evaluate the accuracy and robustness of Jacobian Ensembles. Popular ensemble algorithms like *Bagging* [124] and newer methods like *Snapshot Ensembles* [125] take the average of their base learners. Bagging uses bootstrap to generate multiple subsets from the original training dataset. The base learners are then trained on different bootstrapped subsets in parallel, their results are then averaged. Snapshot ensembles use snapshots of the model during training as the base learners for the ensemble. Other methods like *Soft Voting* [122] use a convex combination of their model outputs to vote.

5.5.1 Experimental Setup

We evaluate the following ensemble methods: *Bagging* [124], *Snapshot Ensembles* [125], and *Soft Voting* [122]. For these, we evaluate all experiments with ensembles trained on 1, 3, 6, and 9 base learners. Effectively, one base learner is similar to using no ensemble method at all.

Jacobian Ensembles. To apply Jacobian Ensembles, we include the Jacobian Regularisation as described in Equation 5.1.2 to the joint loss of standard ensemble methods. The Jacobian Regularisation parameter λ_{JR} is tested for values between 0 and 2: where the resulting models manage to maintain good accuracy as informed by the previous sections' results.

Models & Datasets. We use the MNIST [119] and Fashion-MNIST [120] datasets. These are popular image classification benchmarks, each with 10 classes, and 28 by 28 pixel images whose pixel values range from 0 to 1. For the DNN architecture, we use a version of LeNet-5 [119, 71], which we refer to as LeNet.

UAP Attacks. We again evaluate the robustness of these models to UAPs generated via iterative SGD with 100 iterations and a batch size of 200. Perturbations are applied under ℓ_∞ -norm constraints. The ε we consider in our attacks for this norm are from 0.10 to 0.25, this perturbation magnitude is equivalent to 10%-25% of the maximum total possible change

in pixel values. UAPs are generated over 50 different random seeds, and we report UAPs with the highest attack success rate, as this would represent the worst-case scenario.

Metrics. The following metrics are evaluated on the entire 10,000 sample test sets for each dataset. *Clean accuracy* is the accuracy of the model on the test set. *Model robustness* measures the accuracy of the model when a UAP is applied. This is the opposite of UER which measures model error rather than accuracy in the presence of a UAP. We then average the model robustness over all the UAP attack scenarios that we consider, ℓ_∞ -norm of $\varepsilon = 0.10, 0.15, 0.2, 0.25$, to get a *mean UAP Accuracy*.

5.5.2 Improvements with Jacobian Ensembles

Clean Accuracy. In Figure 5.5 there is a rapid degradation in clean accuracy when λ_{JR} is large. This is when JR is more heavily weighted, and this is consistent with previous work [42, 71] as too much regularisation damages accuracy on the test set. However, having a small amount of JR is still noticeably more beneficial than no JR as indicated by the clean accuracy when considering λ_{JR} in the range of 0 to 0.1 for both datasets across all settings.

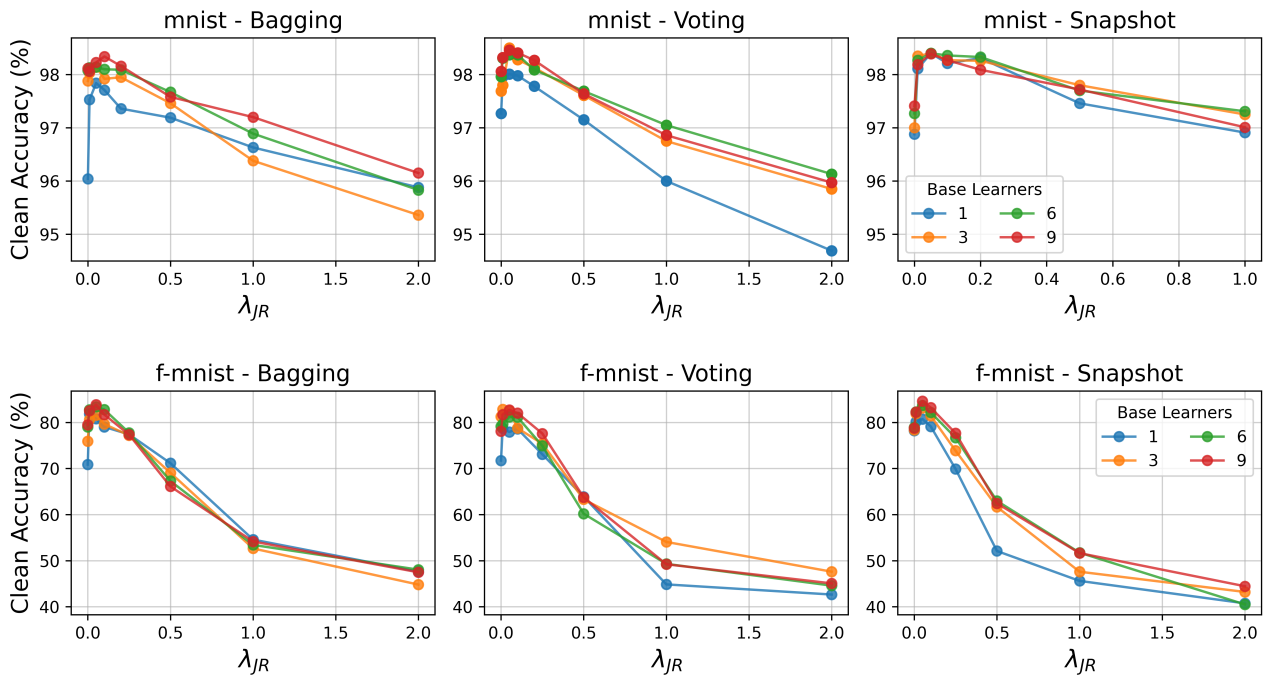


Figure 5.5: Clean accuracy of LeNet on MNIST (top) and Fashion-MNIST (bottom) for various JR strengths λ_{JR} and with varying number of base learners per ensemble method.

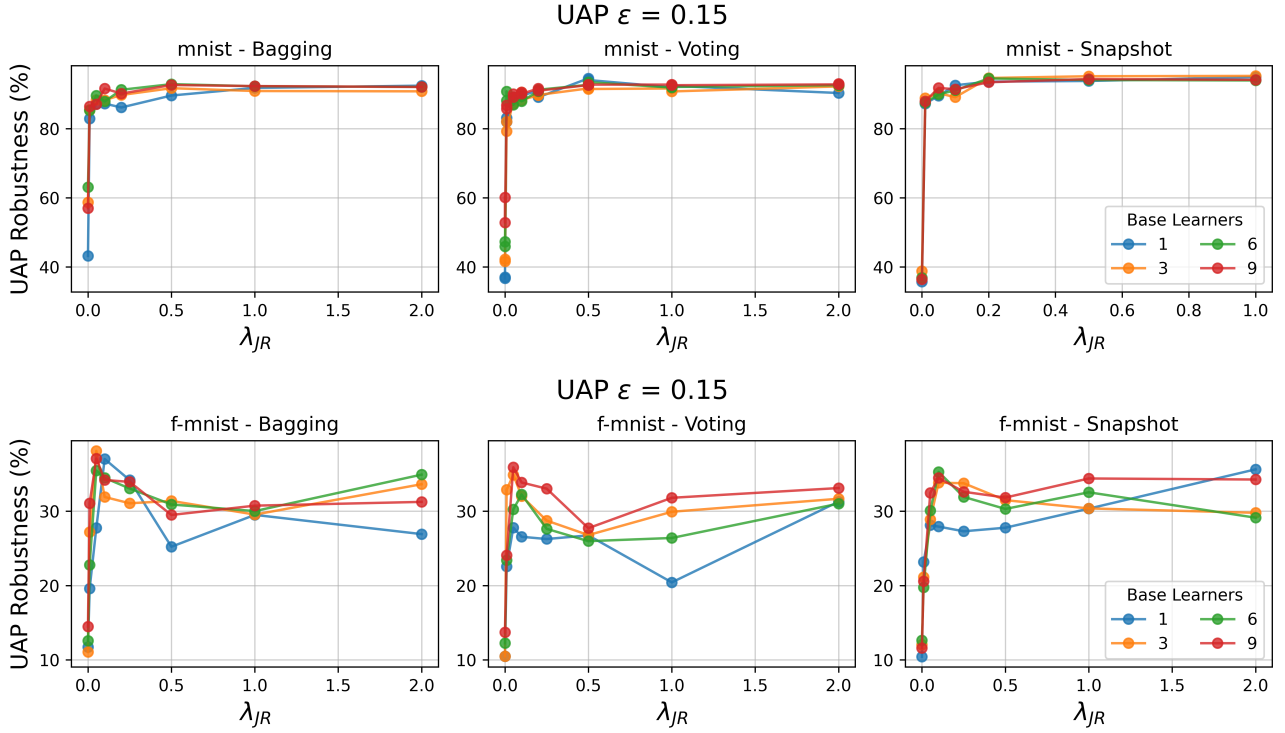


Figure 5.6: Model robustness against UAP with $\varepsilon = 0.15$ of LeNet on MNIST (top) and Fashion-MNIST (bottom) for various JR strengths λ_{JR} and with varying number of base learners per ensemble method.

We also see benefits of using ensembles: models with more than one base learner have noticeably better clean accuracy across all settings in Figure 5.5. Overall, Jacobian Ensembles, which is a combination of ensemble methods and JR, achieves the best performance in our experiments.

Model Robustness. As an example, we present the robustness results for a particular UAP attack strength $\varepsilon = 0.15$ in Figure 5.6. In terms of robustness, trends for both datasets are slightly different since Fashion-MNIST is a more difficult dataset than MNIST: a regularly trained LeNet on MNIST can be expected to have 98-99% clean accuracy whereas it is 90-91% for the same model on Fashion-MNIST. Thus, it is expected that models on Fashion-MNIST are considerably less robust. For most settings, the general trends in Figure 5.6 show that ensembles have a robustness benefit. JR monotonically increases robustness for MNIST and for Fashion-MNIST has a range between $0 < \lambda_{JR} < 0.5$ that achieves the best robustness.

Across both datasets Snapshot ensembles have the best aggregate performance in terms of accuracy and robustness for the various λ_{JR} and differing number of base learners. Thus, our

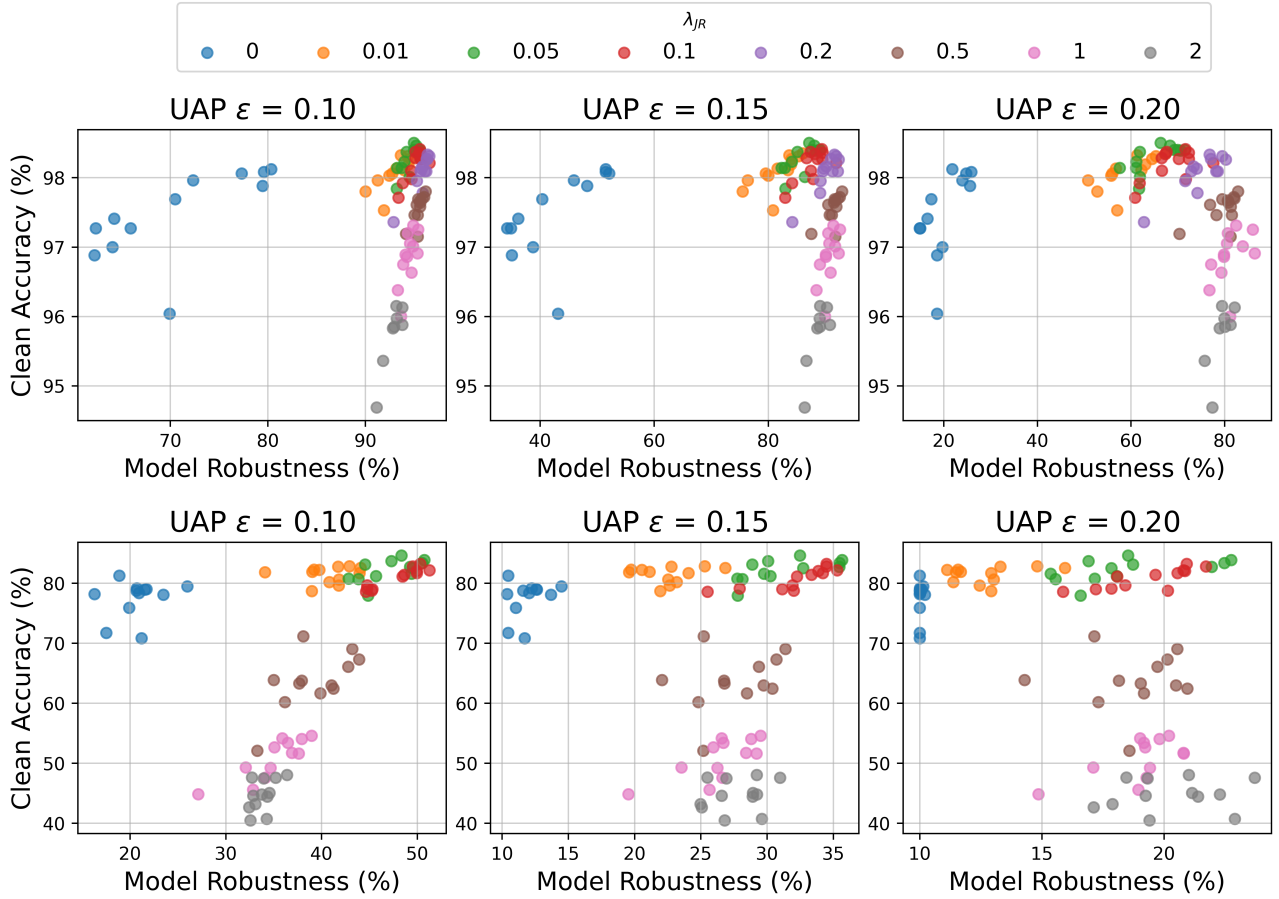


Figure 5.7: Clean accuracy versus robustness trade-off of LeNet on MNIST (top) and Fashion-MNIST (bottom) labeled by λ_{JR} for various UAP attack strengths ε .

subsequent analysis will focus on Snapshot as the ensemble method of choice. Clean accuracy drops greatly for larger λ_{JR} , so the JR range $0 < \lambda_{JR} < 0.5$ appears ideal for both datasets.

In conclusion, ensemble methods with more than one base learner introduce measurable advantages in both accuracy and robustness. These advantages become even more pronounced when combined with JR. Next we analyze the general trade-off between accuracy and robustness.

5.5.3 Accuracy-Robustness Trade-off

In Figure 5.7, we plot the accuracy versus robustness of the various models we trained under the different configurations accounting for various ensemble methods, number of base learners, and λ_{JR} values. The best models achieve both high accuracy and robustness, so these will be on the top right side of the graph.

Table 5.2: Weighted accuracy (in %) of LeNet for MNIST (top) and Fashion-MNIST (bottom). The first 3 rows show the models with highest weighted accuracy. The bottom 3 rows show the *best* weighted accuracy of models trained with only JR, only ensemble, and “standard training” (neither JR nor ensemble).

Model			MNIST Accuracy (%)		
Ensemble	Learners	λ_{JR}	Clean	Avg. UAP	Weighted
Snapshot	3	0.50	97.8	82.8	90.3
Snapshot	9	0.50	97.7	82.3	90.0
Snapshot	3	0.20	98.3	81.5	89.9
JR Only	1	0.05	98.4	74.2	86.3
Bagging	6	0	98.1	42.8	70.4
Standard	1	0	99.1	31.9	65.5
Model			Fashion-MNIST Accuracy (%)		
Ensemble	Learners	λ_{JR}	Clean	Avg. UAP	Weighted
Bagging	9	0.05	83.9	43.2	63.5
Bagging	6	0.05	83.4	43.0	63.2
Snapshot	9	0.10	83.2	42.4	62.8
JR Only	1	0.10	79.0	31.2	55.1
Bagging	9	0	91.2	12.7	51.2
Standard	1	0	90.8	12.1	50.9

In a scenario when robustness is not accounted for (i.e. $\lambda_{JR} = 0$), models would appear in the top left. They are trained to have good accuracy, but remain extremely vulnerable to adversarial attacks like UAPs. On the other hand, models that overcompensate for robustness such as those with very high regularisation (e.g. $\lambda_{JR} = 2$), will appear on the bottom right. As these models sacrifice a significant amount of clean accuracy for improved robustness, especially against UAP attacks with larger strength ε . These delineations become clear when labeling the models according to their λ_{JR} value as in Figure 5.7. Thus, the role of λ_{JR} is evident in improving overall robustness.

To better capture the model performance on both benign and adversarial inputs, we compute the *Weighted Accuracy* by averaging the clean accuracy and mean UAP accuracy. In practice, the defender can adjust the weighting of each accuracy metric in their final assessment to better match their application and risk profile. We choose the mean as the base setting.

Next, we perform an ablation study on the effect of only JR and only ensemble models compared against the top 3 Jacobian Ensembles with the best weighted accuracy in Table 5.2. We find

that Jacobian Ensembles achieve the best weighted accuracy. To compare, we also show in the bottom 3 rows for each table the best models with only JR, only ensembles, and neither JR nor ensembles. The best Jacobian Ensembles have a clear advantage with average UAP accuracy over the non-Jacobian Ensembles whilst maintaining very close clean accuracy. This difference is even more pronounced on the Fashion-MNIST dataset.

Differences between the two datasets MNIST and Fashion-MNIST also show in Table 5.2. Since MNIST is an easier dataset, performance degradation by large $\lambda_{JR} = 0$ are not as prominent, so larger λ_{JR} are favored by the combined score. For Fashion-MNIST, a large λ_{JR} is detrimental as the base model begin with a relatively low clean accuracy ($< 91\%$). In both cases, ensembles demonstrate a noticeably large boost in weighted accuracy, and further tuning is likely to improve their performance.

5.6 Conclusions

In this chapter, we are the first to derive upper bounds on the impact of UAPs. We theoretically show and then empirically verify that data-dependent Jacobian Regularisation significantly reduces the effectiveness whilst maintaining clean performance of UAPs, and finally we propose cosine similarity of Jacobians to measure the strength of shared adversarial perturbation between inputs. We also propose a new method in Jacobian Ensembles to significantly increase model robustness against UAPs whilst maintaining the clean accuracy of models. Our results show that Jacobian Ensembles takes the advantages of both Jacobian Regularisation and model ensembles to achieve superior accuracy and robustness than each of these methods on their own. We also derive theoretical bounds on the robustness to UAPs for model ensembles, showing that increasing the number of base classifiers in the models' ensembles reduces the expected Frobenius norm of their Jacobian and thus improves stability. We empirically verify our results and show that a combination of both JR and ensembles achieves the best performance. These results give us confidence in recommending Jacobian Ensembles as a general methodology for training models as UAPs present a great threat to model adoption and safety. To summarise,

we enumerate the main findings of this chapter:

- We extend theoretical formulations and are the first to show that the effectiveness of UAPs is bounded above by the norms of data-dependent Jacobians and that for model ensembles the robustness to UAPs increases monotonically with the number of base learners.
- We show that even a minimal amount of Jacobian Regularisation reduces effectiveness of UAPs by up to 4-times, whilst leaving clean accuracy relatively unaffected. We also show that Jacobian Ensembles, a combination of Jacobian Regularisation and ensembles, achieves the best accuracy-robustness trade-off as measured by a combined weighted accuracy metric.
- We propose the use of cosine similarity for Jacobians of inputs to measure the strength of shared adversarial perturbations between distinct inputs. Our empirical evaluations on benchmark datasets demonstrate that this similarity measure is an effective proxy for measuring robustness to UAPs.

Jacobian Ensembles represent an advancement in the accuracy-robustness trade-off against UAP attacks. In the presence of small perturbations to the input, we were able to greatly improve UAP robustness and maintain clean accuracy. However, there remains a trend where attack success dramatically increases once perturbations become *sufficiently large*, e.g. when $\varepsilon \geq 0.2$ for MNIST and Fashion-MNIST. As inputs are shifted away from the natural data distribution, it is not unexpected for such degradation in model performance to occur. Intuitively, the larger the perturbations to the input, the easier it would be to detect. Such a detection-based defence would complement our proposed Jacobian Ensembles as JR primarily improves stability against small perturbations. Thus, in the next chapter, we look to explore a defence mechanism to show that attackers must also face a trade-off between the effectiveness of their attack and their ability to evade detection.

Chapter 6

HyperNeuron: Detecting UAPs via Activation Values

We have so far shown that, from the defender’s perspective, progress can be made with the accuracy-robustness trade-off. To further mitigate adversarial attacks on machine learning models, it is also necessary to identify similarly inescapable trade-offs that the *attacker* must face. Indeed there is no recourse if an attack is highly effective and can not be defended against. Several works in the adversarial machine learning literature have shown that it is difficult to consistently defend against input-dependent adversarial attacks [30, 29, 25, 127], as adversarial perturbations can be optimised to evade both the model and the defence on a single input without any significant penalty to the attacker’s objectives. This highlights the immense challenge in developing truly robust defences for input-dependent attacks. In contrast, for UAPs, a single adversarial perturbation has to *simultaneously* evade both the model and the defence on a large set of inputs. Thus, properly constructed defences should be able to force UAP attacks into trade-offs between fooling the model and evading the defence on many inputs. We refer to this as the **defendability-universality trade-off** for UAP attacks. This observation has significant implications as it enables more reliable defences against practical UAP attacks, in contrast to input-dependent perturbations.

We aim to improve on the defendability-universality trade-off as it is an important step towards

more robust machine learning systems in real applications. To properly address and mitigate the threat of UAPs, we also have to consider defending UAPs with practical constraints such as feasibility and run-time in mind. UAPs enable efficient attacks as only one perturbation needs to be generated to fool a model on an entire dataset. It is this property that allows universal attacks to be a significant threat in practice, and therefore defences against these also need to be as efficient, practical, and scalable. Another aspect of putting defences on a more practical footing includes developing robust defences against a variety of UAPs. Recall from Section 2.2.2, that the two most common modalities of UAP attacks for computer vision models come in the form of **adversarial noise** and **adversarial patches**, where noise are diffuse perturbations across the image and patches are continuous 2D regions in the image modified by an adversary. A UAP defence that simultaneously addresses both noise and patch attacks in a computationally efficient and scalable way has yet to be proposed in the literature.

In this chapter, we propose HyperNeuron, a fast and scalable method to detect universal attacks against computer vision DNNs. HyperNeuron flags inputs that sufficiently saturate hidden layer neurons past a trusted baseline. We demonstrate how HyperNeuron can reliably detect a wide variety of both adversarial noise and adversarial patch UAPs with comparable or better performance than existing UAP defences. And, crucially, our results reveal a trade-off between a UAP’s detectability and its generalisability across different inputs; we show that defence-aware attacks necessarily sacrifice UAP effectiveness against the model to increase their evasion of our defence.

6.1 HyperNeuron Detection Algorithm

We first give the motivation behind HyperNeuron and put it into context relative to the related work. We then discuss the detection algorithm itself and how we will be evaluating the defence.

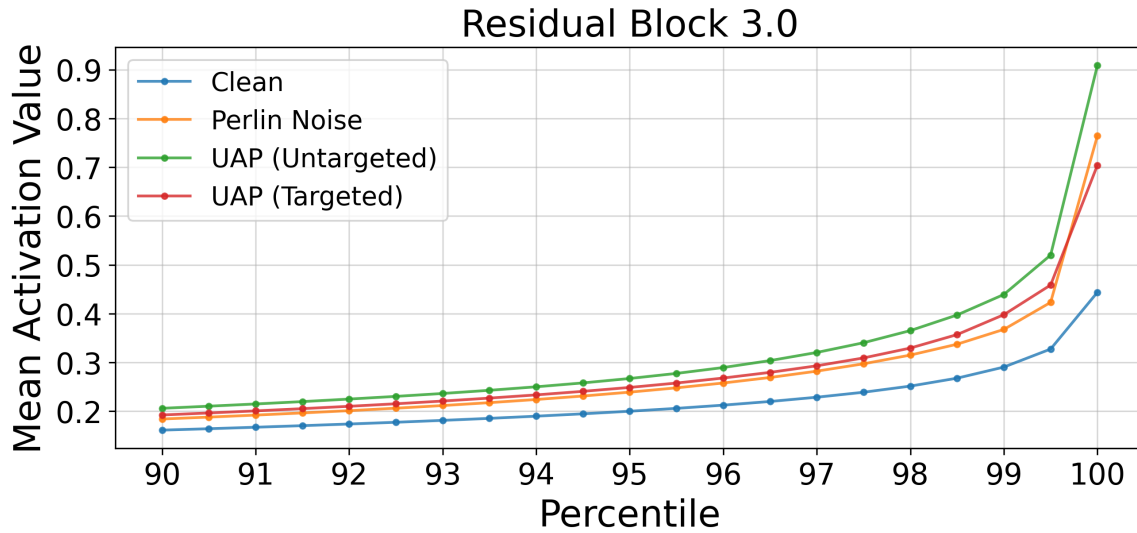


Figure 6.1: Top 10 percentile of mean feature activation values of ResNet50 [6] on residual block 3.0 for ImageNet images with and without UAPs. We plot the average over 10,000 randomly chosen validation set images.

6.1.1 Motivation

Many of the existing UAP defences discussed in Section 2.4 have required training on existing UAP attacks to be effective [77, 75, 53]. This is problematic as these leave defences open to more novel adversarial attacks. For example, in Section 3.2 we were able to demonstrate that procedural noise UAPs were still able to drastically degrade the performance of adversarially trained models. Instead of focusing on specific types of UAP attacks, we try to identify persistent characteristics of UAPs, and construct a detection-based defence based from this. Unlike input-dependent adversarial perturbations, where a perturbation can be tailored and made undetectable for a single input, UAPs necessarily have to exploit more systemic or global features learned by the model and thus they exhibit more discernible characteristics [28, 41, 42].

UAPs are designed to alter the model’s decision on many inputs. For UAPs to saliently affect model predictions across a diverse array of model inputs, we hypothesised that they elicit sufficiently large activations in the hidden layers to consistently affect the model’s output. Intuitively, the more inputs a UAP can evade, the more noticeable its hidden layer activations would be. This is partially supported by the existing literature as several UAP attack algorithms [2, 3] explicitly maximise hidden layer activations, and several UAP defences attempt to rectify

hidden layer activations [77, 75]. However, none of these past works have attempted to detect UAPs based on this property nor have they adequately demonstrated and characterised the defendability-universality trade-off in practice.

From a mechanistic perspective, the Rectified Linear Unit (ReLU) is a likely pathway that enables the “hyper-activation” or extreme increase in neuron values within the model’s hidden layers. In modern neural networks, ReLU has become the activation function of choice, as it achieves significantly better generalization and test performance over previously used activation functions like sigmoid or hyperbolic tangent. Although there are models like shallow Radial Basis Function (RBF) networks that appear to be naturally immune to adversarial examples, they often achieve high error and do not generalise well in relevant practical applications [23]. ReLU, given by $\text{ReLU}(x) = \max(0, x)$, has allowed improvements in model performance and the proliferation of large-scale neural networks [37]. That ReLU is not bounded above may be a mechanism that helps facilitate adversarial attacks. To test this theory, we first compared the neuron activation values of clean inputs versus those with UAPs.

These observations are seen in Figure 6.1, where we plot the average top percentile of mean neuron activation values for the observed layer. On the rightmost side of the graph, we notice that the top percentile of values are much greater for inputs with UAPs (non-blue curves) compared to those of clean inputs (blue curves), leading credence to our hypothesis.

Figure 6.2 shows an example where UAPs increase the activation values (mean) and statistical dispersion (standard deviation) in the hidden layers. For this example, we can see in the second and third rows that darker cells correspond to the regions that the adversarial perturbations occupy. In the adversarial patch case on the rightmost column, it shows that the activation values of the patch are significantly larger than the rest of the image. Statistically, this shows up as an increase in standard deviation (std).

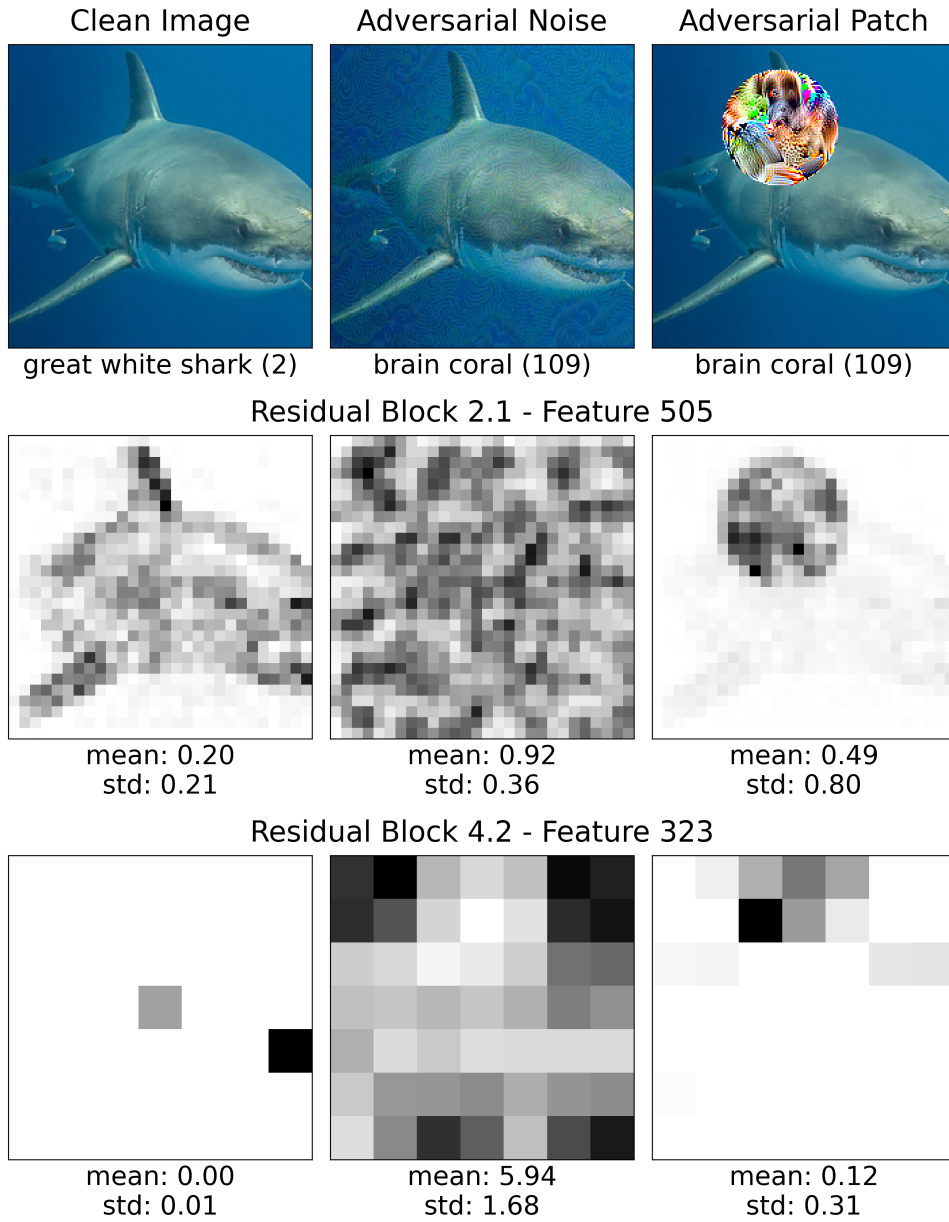


Figure 6.2: Visualizations for an image from the ImageNet dataset (left), with an adversarial noise (center), and with an adversarial patch (right). The clean image is classified as a great white shark (class #2) while the other two are classified as brain coral (class #109) by a ResNet50 model. The second and third rows visualise selected features from chosen residual blocks of the model. The features' computed mean and standard deviation (std) are shown below and are noticeably larger for the perturbed inputs.

6.1.2 HyperNeuron Algorithm

We first describe the HyperNeuron algorithm on a high level. For a given deep neural network F , HyperNeuron observes the hidden layer activations $L_k(\cdot)$ at chosen layers k . For each layer,

Algorithm 1 HyperNeuron Algorithm**Require:** X_{clean} , X_{test} , aggregation function A , layer k , top percentile t , FP r , $\mu_k \leftarrow \text{average}_{x \in X_{\text{clean}}} A_k(x)$ $\sigma_k \leftarrow \text{standard-deviation}_{x \in X_{\text{clean}}} A_k(x)$ **for** $x_i \in X_{\text{test}}$ **do** $Z_k(x_i) \leftarrow \text{z-score}(A_k(x_i), \mu = \mu_k, \sigma = \sigma_k)$ $z_{\text{top-pct},i} = \{\text{top-percentile}(Z_k(x_i), t)\}$, top- t percentile $z_{\text{top-avg},i} \leftarrow \text{average}(z_{\text{top-pct},i})$ **if** $z_{\text{top-avg},i} > \theta_{k,r}$ **then** $\text{flag}(x_i) = 1$ **else** $\text{flag}(x_i) = 0$ **end if****end for****return** $\{z_{\text{top-avg},i}\}, \{\text{flag}(x_i)\}$

it establishes a baseline mean μ_k and standard deviation σ_k of the top percentile of neuron activation values based on a trusted clean dataset X_{clean} . When given a new input x , the hidden layer activations $L_k(x)$ are computed and compared with the baseline statistics μ_k, σ_k to generate a z-score $Z_k(x)$. If the resulting aggregate score exceeds a pre-defined threshold, then the input is flagged and considered anomalous.

Design Choices. Based on our categorisation of defences outlined in Section 2.4.1, this is a Type III detection defence. We choose our detection mechanism with the assumption that the distribution of hidden layer activations for benign data points will be consistent with that of our trusted clean dataset. This is in-line with the intuition that UAPs hyper-active hidden layer activations, and avoids the need to train on any particular type of adversarial attack.

For outlier detection, we use the widely-used z-score metric because of its simplicity and fast computation; simplicity to avoid overly complex mechanisms in the defence, and fast computation as we plan to deploy HyperNeuron for DNNs in practical settings and in real-time tasks like object detection. We now explain HyperNeuron in greater detail.

Feature Aggregation. Let the output of hidden layer $L_k(x)$ have dimensions $a_k \times a_k \times d_k$. This can be thought of as a d_k -dimensional vector of $a_k \times a_k$ features. Figure 6.2 shows examples of these $a_k \times a_k$ features for outputs of residual blocks 2.1 and 4.2 in the ResNet50 [6] model. We aggregate the statistics of each $a_k \times a_k$ feature in $L_k(x)$ to obtain an aggregated feature

vector $A_k(x) \in \mathbb{R}^{d_k}$. For each input, we use the aggregation function $A_k(\cdot)$, either the mean $A_k = M_k$ or the standard deviation $A_k = S_k$. The aggregation of features is needed to reduce the memory requirements and latency that detector introduces. Without it, the activations of 100 images from a single layer can take up to 22GB of memory. This aggregation step reduces the size of these vectors down to 0.6MB. As we will show later on, the aggregated feature vectors $A_k(x)$ are sufficient to detect UAPs.

Setting a Baseline. To set a baseline for detection, we use points from the clean training dataset X_{clean} . For each layer k , we compute the mean μ_k and standard deviation σ_k of the aggregate feature vector $A_k(\cdot)$ over the clean dataset X_{clean} . For each new input x_i , we compute its z-score vector $Z_k(x_i)$, where $Z_k(x_i) = (A_k(x_i) - \mu_k)/\sigma_k$ for each component in A_k , relative to the clean baseline vectors μ_k and σ_k . We use the top- t percentile of the z-scores in the vector $Z_k(x')$ to identify the features that have the largest deviation from the baseline. Once we have the average $z_{\text{avg},i}$ of the top- t percentile values for input x_i on layer k , we compare this to a threshold $\theta_{k,r}$ to determine whether or not to flag the input for that layer, where $\theta_{k,r}$ is chosen to have a desired False Positive (FP) rate r over the trusted clean dataset. A flag is set to $\text{flag}(x_i) = 1$ to signal that input x_i deviates from the baseline on layer k .

In *summary*, our detector uses a trusted clean dataset X_{clean} as a baseline and first aggregates the statistics of the hidden layer activations $L_k(x)$ to a feature vector $A_k(\cdot)$ for each feature at the chosen layer k . When given test inputs X_{test} , we compute each sample's aggregate feature vectors $A_k(x_i)$ and calculate its z-score $Z_k(x_i)$ with respect to the baseline mean μ_k and standard deviation σ_k vectors. The average of the top- t percentile z-scores are then compared to a threshold $\theta_{k,r}$ and the input is flagged $\text{flag}(x_i) = 1$ if it exceeds the threshold. This process is described in Algorithm 1.

Detector Calibration. Given the model, trusted clean dataset X_{clean} , and test set X_{test} , we discuss the process for choosing the remaining parameters of the detector. Since we refrain from training on any particular UAP attack a priori, we propose an **SGD-Layer** algorithm to generate perturbations for calibration. This method uses Stochastic Gradient Descent (SGD) optimization to generate a perturbation that maximises hidden layer activations, similar to

[3, 2]. The objective SGD-Layer optimises is:

$$-\sum_i \log \left(\prod_k \|L_k(x_i + \delta)\|_2 \right), \text{ such that } \delta \in \mathcal{P}, \quad (6.1)$$

where L_k is the model’s output for hidden layer k . SGD optimisation is chosen as it has been shown to be the best perfect-knowledge (PK) algorithm for UAP attacks. Since it is able to be tailored to attack specific layers, we use SGD-Layer to calibrate our detector, as the primary mechanism of our detector relies on identifying elevated hidden layer activations.

To calibrate the detector we need to choose: aggregation function A , layer observed k , and top percentile t . Given candidate layers $k \in K$ for observation, we perform the SGD-Layer attack for each k . For each of the generated SGD-Layer UAPs, we compute the AUC for both aggregation functions M_k, S_k for each $k \in K$. The best layer k^* and aggregation method A^* are chosen to maximise the average AUC for these attacks. The detector then observes the single layer that has the best aggregate performance. Grid search is used to find the best top-percentile value t . For our initial experiments, we choose to only monitor a single layer with HyperNeuron to better evaluate the method. It may be possible to combine multiple layers to identify UAPs in a more complex learning task, however, this may have drawbacks for the scalability, explainability, and robustness of the detector.

6.1.3 Measuring Detector Performance

The initial goal of our detector is to find inputs that have been perturbed by UAPs and to detect adversarial perturbations even if they have not been successful as attacks. This allows the defender to detect attempts to attack the model. We consider as true positives inputs that contain a UAP even if they did not change the model’s original prediction. To evaluate HyperNeuron, we take a subset of the training set as the trusted dataset X_{clean} for computing the baseline vectors μ_k, σ_k . Tests are then conducted on a separate validation set. We apply the UAP to samples from the validation set to measure the True Positive (TP) rate of the detector. The False Positive (FP) rate is the number of clean inputs falsely flagged by the detector.

Defendability-Universality Trade-Off. A properly constructed defence should be able to force the attacker to face a defendability-universality trade-off. This is where the effectiveness of the UAP attack on the model (universality) is at odds with its success in evading the defence put in place (defendability). There is no recourse if an attack is highly effective and can not be defended against. To evaluate HyperNeuron’s detection performance, we use the *area under the receiver operating characteristic* (AUC-ROC), which we shorthand as *area under the curve* (AUC). The receiver operating characteristic curve (ROC) is defined by plotting TP and FP on the x and y-axes respectively. AUC is the area under ROC and is commonly used to capture the overall performance of a binary classifier.

In evaluating the defendability-universality trade-off for HyperNeuron, we use the AUC to measure the overall “defendability” to an attack and the stated attack success metric to measure the “universality” of a UAP attack. The attack success metric could be the UER or TSR as previously defined in Section 2.2.1, depending on the attacker’s goals. Ideally, a good defence forces the attacker to trade-off UAP detectability versus effectiveness, making it more challenging to evade both the model and the detector on many inputs with a single UAP. This would translate to an inverse correlation between AUC and UER (or TSR).

6.2 Small-scale Images

We first evaluate HyperNeuron on an image classifier for the CIFAR-10 dataset [128]. This is a small-image dataset that is commonly used as a benchmark for image classification and often used in the machine learning literature.

6.2.1 Experimental Setup

The CIFAR-10 dataset [128] has 10 class labels and $32 \times 32 \times 3$ images. We evaluate HyperNeuron on a ResNet18 [6] model that has 94.02% accuracy on the test set. We use HyperNeuron on the layer outputs of its 8 residual blocks numbered 1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 4.0, 4.1, taking

the output after the ReLU activation function. We compute the clean baseline vectors, μ_k , σ_k , for each layer based on activation values from the 50,000 training set images.

Threat Model. We evaluate HyperNeuron against perfect-knowledge UAP attacks. For adversarial capabilities, perturbations are applied under ℓ_∞ -norm constraints. For the adversarial goal, we evaluate HyperNeuron against both untargeted and targeted attacks. The effectiveness of untargeted attacks are measured with the UER, defined as the proportion of inputs that are misclassified. Targeted UAPs are evaluated according to their TSR, the proportion of inputs classified as the target class. For more details on these metrics, we refer the reader to how our threat models are defined in Section 2.2.

We evaluate with SGD untargeted and targeted attacks. We also test the detector with SGD transfer attacks from a VGG16 model [7] that achieves 93.79% accuracy on the test dataset. As a sanity check, we also evaluate against uniform random noise.

Detector Calibration. From the results shown in Figure 6.3, we select the mean $M_k(\cdot)$ as our aggregation function and $k = 1.1$ as the layer to observe as this combination has the highest average AUC across the UAP attacks. We refer to this HyperNeuron configuration of observing layer 1.1 with mean aggregation as 1.1M. For the top percentile t , we found $t = 85$ to have good performance on average, though the detection results are not too sensitive to this parameter for $65 < t < 99$.

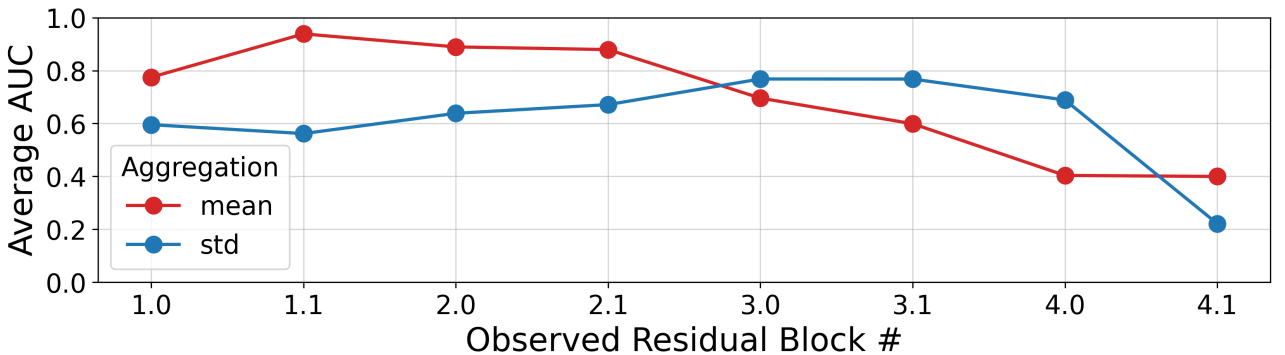


Figure 6.3: Mean and standard deviation of AUC per layer for ResNet18 against SGD-Layer for calibration at $\varepsilon = 10$.

6.2.2 Layer Calibration

In this preliminary study, we use HyperNeuron to understand the relationship between the universality and detectability of UAPs. We therefore analyze the differences in detection rates across layers and for different perturbation magnitudes.

Layer Differences. Figure 6.3 shows that the AUC decreases in the later residual blocks, in particular, for 4.0 and 4.1. A possible explanation could be that perturbations which mimic low-level features primarily stimulate neurons in the earlier layers, as is the case for most UAPs, but does not translate to increased activations for high-level features. So activations of inputs with UAPs have much lower values in the later layers when compared to the clean baseline. This shows that the tested UAPs appear to hyper-activate neurons in the earliest layers of the model. This observation is line with the observation that these perturbations often resemble low-level features reported in our previous chapters and previous work [28, 123, 3].

6.2.3 Detection Results

For untargeted attacks, we see in the top row of Figure 6.4 that UAPs with high universality (UER) are correlated with high detectability (AUC). And for smaller perturbation values, a lower AUC corresponds to a drastically reduced UER. This confirms the hypothesis that universality and detectability are related. In the same figure, the targeted UAP attack against class 5 ('dog') is consistently detected even when the TSR and perturbation are very small. This example shows that the model is highly sensitive to the features that this targeted UAP exploits. Even at very low $\varepsilon = 2$ perturbations can be detected by HyperNeuron at an AUC of 0.8. On the other side, attacks that do not exploit sensitive features of the model require larger perturbations to achieve high success rates, but they can be detected by HyperNeuron at these large ε values.

We see in Table 6.1 that targeted UAPs for 8 out of the 10 labels have TSR higher than 75%. Despite the model's fragility to these targeted UAPs, HyperNeuron is able to successfully detect them with a high AUC of at least 0.87.

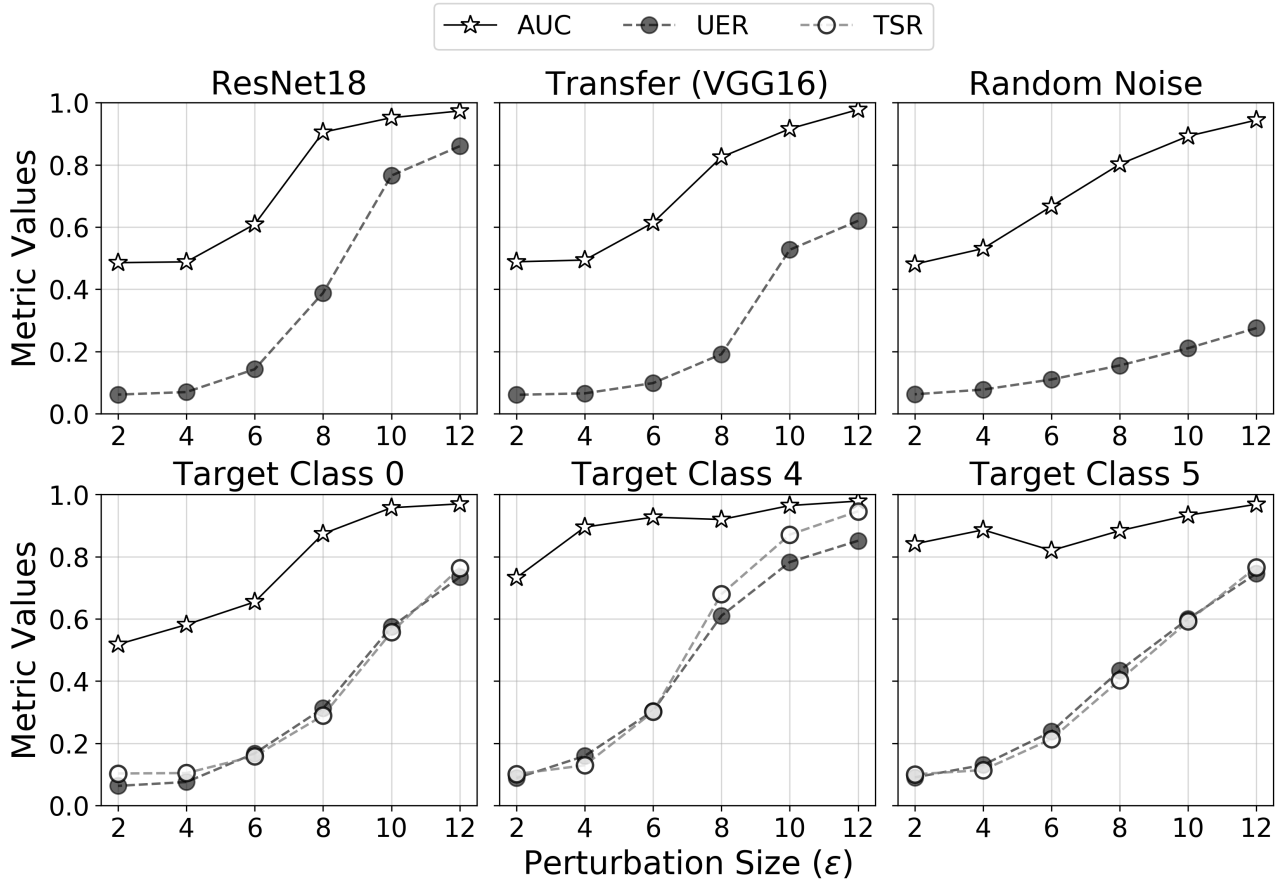


Figure 6.4: Evaluation metrics for ResNet18 against UAPs for different ε . Top row are untargeted UAPs, bottom row are targeted UAPs for class label indices 0, 4, and 5.

Differences in the TSR are likely a result of how the model was trained, as some class labels could take up more space in the output manifold than the others depending on randomization, dataset balance, training epochs, and such. Each class label may have specific sets of excitatory and inhibitory neurons, so it is likely that the computed UAPs exploit these, leading to larger signals on these particular neurons depending on the targeted class. Figure 6.4 shows that the AUC for class IDs 4 and 5 are very high even for very low ε values. This is notable also as class

Table 6.1: Metrics for targeted UAPs at $\varepsilon = 10$ against ResNet18 on CIFAR-10.

Class ID	0	1	2	3	4	5	6	7	8	9
Class Label	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
UER (%)	55.56	73.32	74.62	68.24	78.05	58.39	75.36	69.74	74.14	73.25
TSR (%)	55.73	81.88	83.22	75.12	87.10	59.26	83.57	76.49	83.11	82.19
Best AUC	0.958	0.957	0.961	0.957	0.965	0.933	0.972	0.869	0.965	0.904

ID 5 has one of the lowest TSR while class ID 4 has the highest.

To summarise, we identify the two scenarios where UAP has high universality and how these are detected in each case. In the first case, the UAP has naturally high UER even for small perturbation values. These are detected with high AUC. In the second case, the UAP has high UER only for larger perturbation values. It is these large perturbations that are then readily detected with high AUC. These results confirm a trade-off between UAP universality and detectability, and demonstrates the viability of HyperNeuron as a UAP detector. This validates our detection method before applying it to the large-scale ImageNet benchmark.

6.3 Large-scale Images

In this section, we evaluate HyperNeuron’s detection capabilities against adversarial noise and adversarial patch UAP attacks. This evaluation is done on the ImageNet [11] dataset, a standard benchmark for computer vision tasks with 1,000 class labels. Classifiers pre-trained on ImageNet are frequently used as feature extractors [92] in DNNs for other tasks such as object detection [102] and scene segmentation [34]. ImageNet’s pervasive use across real-world computer vision applications makes models trained on it an important case to study.

6.3.1 Experimental Setup & Calibration

We evaluate HyperNeuron on ResNet50 [6] which takes ImageNet images with dimensions $224 \times 224 \times 3$. Our chosen model achieves 76.13% accuracy on the clean validation set and we calibrate HyperNeuron on the layer outputs of its 16 residual blocks numbered 1.0 to 4.2. Clean baseline vectors μ_k , σ_k for each layer are based on the activation values from 50,000 random training set images, with 50 examples for each of the 1,000 class labels. To determine the sensitivity of the detector to this baseline, we have repeated the evaluations for 10 different sets of training images and found no significant variance in detection performance.

Threat Model. For adversarial capabilities, we consider both adversarial noise and adversarial

patch attacks. For adversarial noise, perturbations are applied under ℓ_∞ -norm constraints. For the adversarial goals, we evaluate HyperNeuron against both untargeted and targeted attacks. The effectiveness of untargeted attacks are measured with the UER, defined as the proportion of inputs that are misclassified. Targeted UAPs are evaluated according to their TSR, the proportion of inputs classified as the target class. For more details on these metrics, we refer the reader to how our threat models are defined as in Section 2.2.

UAP Attacks. Evaluations are done on the 50,000 image validation set. For adversarial noise, we test on a variety of universal attacks: SGD (Untargeted, Targeted, Transfer), SGD-Layer, Perlin Noise. Then, we test our defence against adversarial patch UAPs [54]. We expect PK attacks to be effective, but easy to detect. We also expect the remaining attacks to require larger perturbation values to be more effective, and thus, be easily detectable.

For the transfer attacks, UAPs are generated on a separate model before being used to attack the evaluated model. The UAPs we used for this are generated from a VGG16 model [7] trained on ImageNet, a ResNet50 model trained on Stylized-ImageNet (SIN) [111, 123], and a ResNet50 model trained on both Stylized-ImageNet and ImageNet (SIN+IN). Stylized-ImageNet is a variation of ImageNet using similar images, but designed to be differentiated based on high-level shape features rather than low-level textural features [111]. All UAPs we use in the transfer attacks achieve at least 90% UER at $\varepsilon = 10$ on the models they were generated from. For coverage, we include an evaluation against Perlin Noise as these UAPs have high effectiveness and are generated in a different way from the PK attacks. We use Bayesian Optimization to find optimal attack parameters for Perlin Noise.

Calibration. From the results shown in Figure 6.5, we select the mean $M_k(\cdot)$ as our aggregation function and $k = 3.0$ as the layer to observe as this combination has the highest average AUC across the SGD-Layer attacks. We refer to this HyperNeuron configuration of layer 3.0 with mean aggregation as 3.0M. For the top percentile t , we found $t = 95$ to have good performance on average, though the detection results are not too sensitive to this parameter for $90 < t < 99$.

We now separate the analysis into of targeted and untargeted UAP noise attacks into two separate sections, then go over results for adversarial patch UAPs.

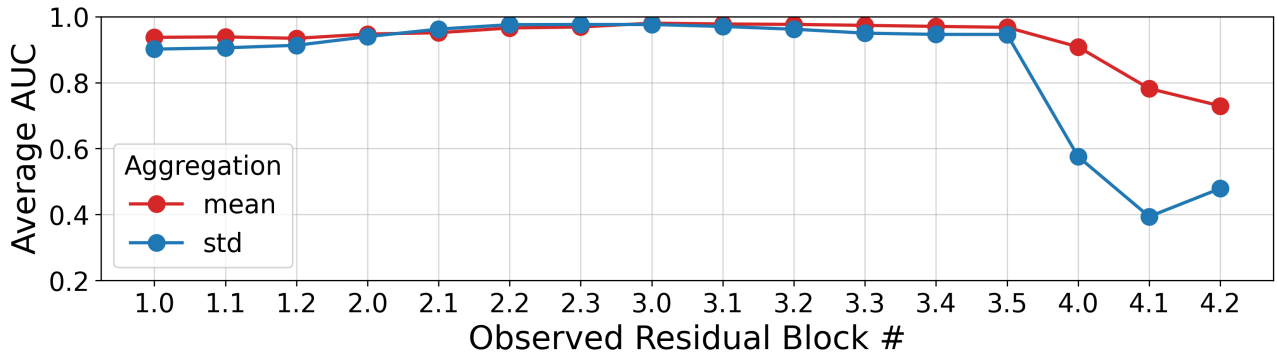


Figure 6.5: Mean AUC for HyperNeuron with each combination of observed layer and aggregation function against SGD-Layer attacks with $\varepsilon = 10$ on ResNet50.

6.3.2 Adversarial Noise Evaluation

For adversarial noise UAPs, we see in Figure 6.6 that UAPs with high universality (UER) are correlated with high detectability (AUC). And for smaller perturbation values, a lower AUC corresponds to a lower UER. The detection rates remain high for all attacks.

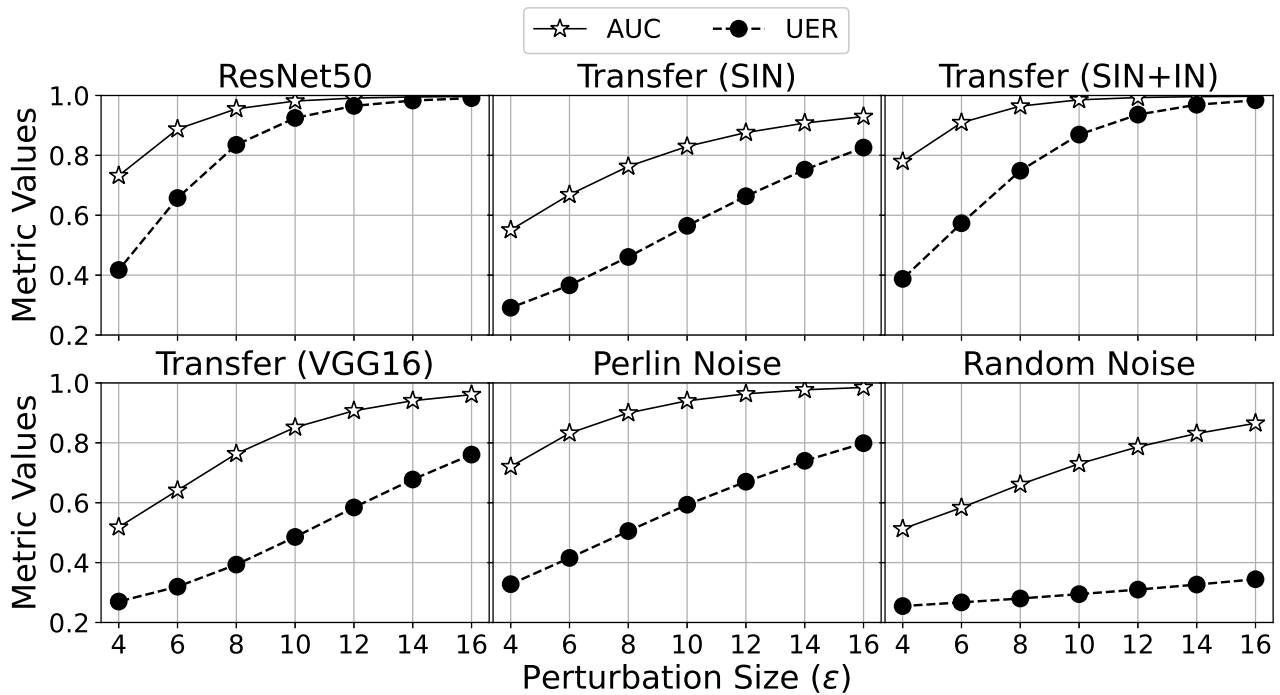


Figure 6.6: Evaluation metrics of HyperNeuron - 3.0M on ResNet50 against a variety of UAP attacks for different ε .

Among the different types of UAPs, PK attacks exhibit high detectability as they were generated on the same model. More surprisingly, transfer attacks also have high detectability,

perhaps as a result of either the shared model architecture or training datasets. In comparison, procedural noise attacks have lower universality and require much larger perturbations to be effective. UAPs are naturally detected for larger perturbations as they drift from the baseline distribution. UAPs at lower perturbation values but with high UER are also detected often like the SGD at $\varepsilon = 8$ with $\text{AUC} = 0.95$ and $\text{UER} = 83.51\%$. UAPs that are not as frequently detected have much lower universality. These findings confirm a defendability-universality trade-off.

6.3.3 Targeted UAP Evaluation

ImageNet contains 1,000 distinct class labels and we first generate one targeted UAP for each class at $\varepsilon = 10$. We find that the TSR ranges from 33.74% to 86.33% depending on the target class label. The high TSR for several class labels is alarming. This further highlights the importance of detecting and defending against these types of universal attacks.

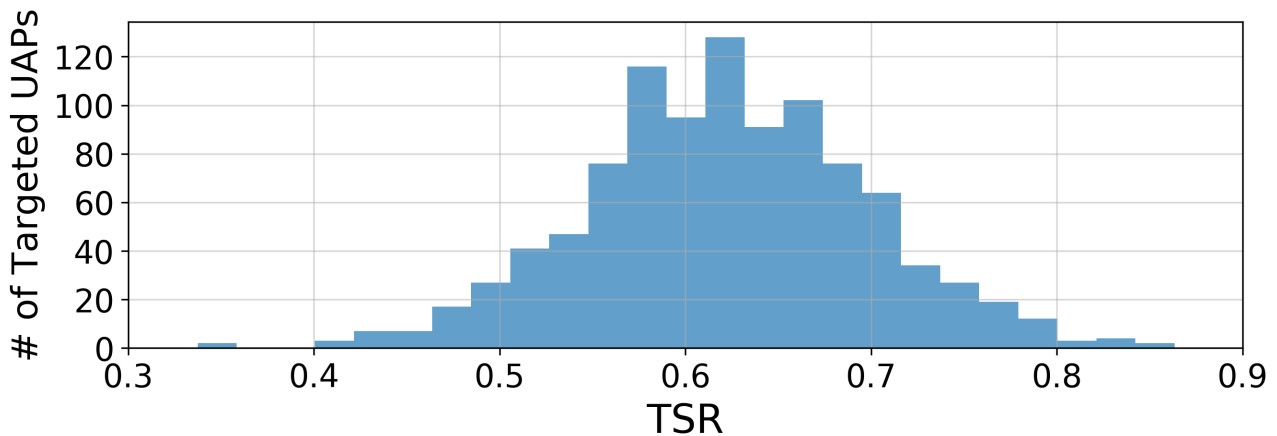


Figure 6.7: Distribution of TSR of targeted UAPs across all 1,000 ImageNet class labels with $\varepsilon = 10$ on ResNet50.

We sample targeted UAPs for 5 random class labels and report their performance and detection rates in Table 6.2. Targeted UAPs with high TSR are all detected with AUC above 0.94. This demonstrates the efficacy of HyperNeuron against targeted UAPs. On the other hand, targeted UAPs with much lower TSR have AUC within the range 0.74-0.93. There is no direct correlation between TSR and AUC, but this difference is likely a product of how the model’s

learned representations for these classes affect their detectability. It is possible that activations for some class labels could be dispersed over a wide number of neurons rather than having notably higher activations for a small percentile of neurons, which makes it more difficult to detect with HyperNeuron. This is in line with our results from CIFAR, where UAPs with high

Table 6.2: Metrics for targeted UAPs at $\varepsilon = 10$ against ResNet50 on ImageNet.

Class ID	698	109	815	854	611
Class Label	palace	brain coral	spider web	theater curtain	jigsaw puzzle
UER (%)	65.13	73.40	70.57	78.82	81.82
TSR (%)	70.60	83.35	83.41	85.04	86.33
Best AUC	0.970	0.980	0.945	0.991	0.940

targeted success are also detected at a very high AUC. Since these UAPs are highly optimised, they will be easier to detect with HyperNeuron. On the lower end of the targeted success rates, class ID 960 is easy to detect, whereas those for class IDs 72 and 984 have slightly lower detection rates. This is in contrast to CIFAR-10, where targeted UAPs for 80% had consistently high TSR of at least 75.12% and high detection rates (0.89-0.99). This can be explained by the difference in the number of class labels, 10 for CIFAR-10 and 1,000 for ImageNet.

The final layers again fail to detect UAPs similar to what happened in CIFAR-10. It is likely that UAPs primarily hyper-activate early neurons, and the top activation values of perturbed samples in these last layers naturally do not deviate as much from the baseline distribution. For various ε values, the shape of the AUC curves remains consistent.

6.3.4 Adversarial Patch Evaluation

Adversarial patches are image patches that work as targeted UAPs and are robust to rotation and translation [54, 62, 27]. These properties allow it to be printed for physical targeted attacks on computer vision models. We evaluate against the ResNet50 model trained on ImageNet. In contrast to previous experiments, circular adversarial patches are constrained by a special case of ℓ_0 -norm and are also optimised to be robust to transformations.

To generate targeted adversarial patches we use SGD with an Expectation Over Transformations (EOT) objective, we refer the reader to our background discussion in Section 2.3.7 for more details. We optimise the patches over 2,500 random images from the training set. This is sufficient to generate effective targeted adversarial patches that achieve at least 80% TSR on the 50,000 image validation set when covering 16% of the image area.

We limit adversarial patches to 16% of the image area as it becomes unreasonable to consider patches that occupy much larger portions of the image where they can obstruct the original object—with the patch effectively replacing the true class label of the original clean input. We select three class labels to evaluate our defence against: *jigsaw puzzle* (#611), *toaster* (#859), and *chocolate sauce* (# 960). Toaster was chosen as it was the primary example used by Brown et al. [54], jigsaw puzzle and chocolate sauce were chosen as they had the highest and lowest TSR amongst the targeted adversarial noise UAPs respectively.

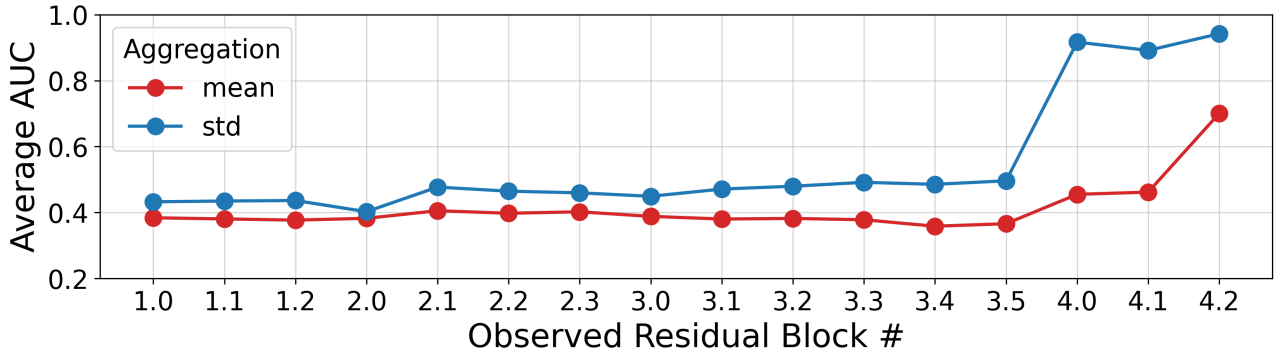


Figure 6.8: Mean AUC for HyperNeuron with each combination of observed layer and aggregation function against Adversarial Patch attacks with 16% patch size on ResNet50.

HyperNeuron Results. For detection of patches, we select the layer that has the best detection performance. Based on results from Figure 6.8, we chose the standard deviation $S_k(\cdot)$ as our aggregation function and $k = 4.2$ as the layer to observe as this combination has the highest average AUC. We refer to this HyperNeuron configuration as 4.2S.

Under these constraints, adversarial patches likely exploit higher-level features that are more robust to transformations and can be better recognised across images. It is these higher-level features that correspond to the final layers of the model. 4.2S is able to detect adversarial patches introduced with very high AUC (>0.9) for all adversarial patches that cover 8% or

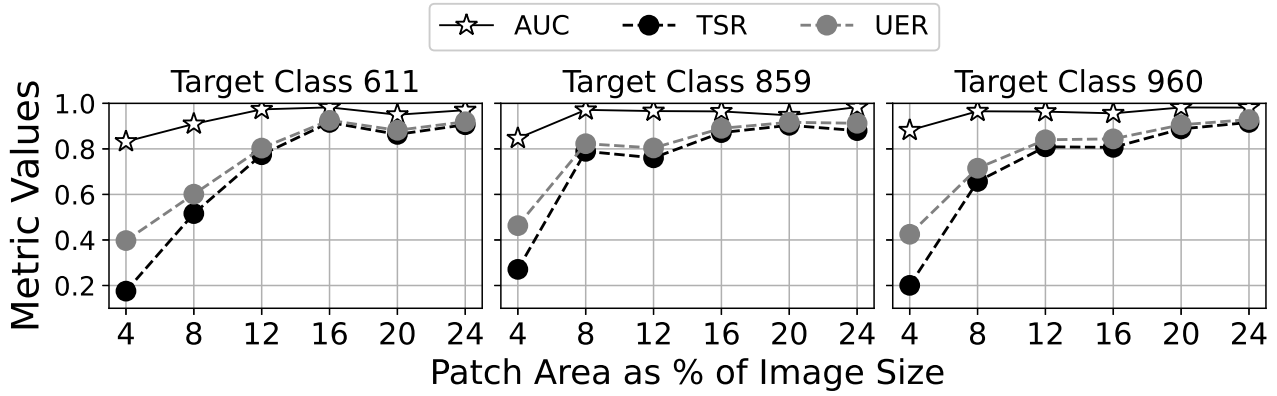


Figure 6.9: Top row shows metrics for targeted adversarial patches against ResNet50 on ImageNet. There is significant overlap between the Best AUC and Block 4.2 AUC. Bottom row shows the AUC per layer.

more of the image as seen in Figure 6.9. Using standard deviation makes sense intuitively as the adversarial patch creates a large disparity in activation values between the original image and the patch’s region.

Even when a patch is not the most effective at 4% of the image size, the patch is still reliably detected at 0.8 AUC or more. The disparity in activations between the patch and the original image allows the patch to be detected. That the patch has to be robust to translation across the image means that it has to hyper-activate neurons which correspond to the area it occupies. To conclude, our results show that both adversarial noise and patch UAPs with high UER are detected by HyperNeuron with high AUC. We next evaluate the performance of HyperNeuron with more practical applications in mind.

6.4 Detection in Practice

We have shown that HyperNeuron is able to reliably detect both adversarial noise and patch UAPs. Our results also show that UAPs with high universality often have high detectability with HyperNeuron. We now apply HyperNeuron in practice by first, combining the detection for both noise and patch UAPs; and second, by measuring the latency our detector introduces.

6.4.1 Performance Metrics in Practice

It is also useful to understand how the detector affect the overall performance of the system. When applying a UAP to a set of inputs, the attacker’s objective is to maximise the model error for perturbed inputs. Thus, perturbed inputs not flagged by the detector achieve this goal only if they are also misclassified. When we measure the performance of the model on the clean dataset, the detector should ideally not flag clean inputs that the model classifies correctly. To measure the performance of HyperNeuron in practice, we assume that inputs flagged by the detector ($\text{flag}(x) = 1$) are discarded. Non-discarded inputs ($\text{flag}(x) = 0$) are classified. Thus, we define two metrics: the Attack Success Rate (ASR) as the proportion of non-flagged perturbed inputs that have been misclassified, and the Clean Performance (CP) as the proportion of non-flagged clean inputs that have been correctly classified. The former penalises the detector when it fails to discard misclassified perturbed inputs, while the latter penalises the detector when it discards correctly classified clean inputs.

$$\text{ASR}(\delta) = |\{x \in X : \text{flag}(x') = 0 \wedge f(x') \neq \tau(x')\}| \cdot \frac{1}{|X|} \quad (6.2)$$

$$\text{CP} = |\{x \in X : \text{flag}(x) = 0 \wedge f(x) = \tau(x)\}| \cdot \frac{1}{|X|} \quad (6.3)$$

where $x' = x + \delta$. ASR does not penalise flagging perturbed inputs that are correctly classified as the perturbation may indicate a potential malicious adversary. CP does not penalise flagging clean inputs that are misclassified by the model, as it is more detrimental to serve incorrect predictions without flagging them. For the defender, lower ASR and higher CP are preferred. Note that the ASR and CP deviate from the usual performance metrics UER outlined in Section 2.2 since the system has the additional detection component. In this evaluation, we use the ASR, CP, and AUC to evaluate the detector’s and attacker’s performance.

6.4.2 Practical Evaluation of HyperNeuron

In our previous results, we have shown that we can configure HyperNeuron to observe layer 3.0 with mean aggregation (3.0M) and to observe layer 4.2 with standard deviation aggregation (4.2S) to get good performance in detecting adversarial noise and patch UAPs respectively. However, in more realistic scenarios, it is not always known beforehand what type of adversarial perturbation an attacker may introduce. Thus, it is useful to have a detector that can flag both adversarial noise and adversarial patch UAPs.

Combining Mask and Patch Detection. In the general case, we combine the detection of both noise and patch UAPs by flagging an input when it is flagged by either 3.0M or 4.2S. We refer to this configuration as 3.0M+4.2S. To measure the ASR and CP, we set the detector thresholds to have 5% false positive rate on the clean baseline for each of 3.0M and 4.2S. Our results in Table 6.4 show that this naive rule is sufficient to create a detector that reliably flags both types of UAPs.

In Table 6.4, the combined detection 3.0M+4.2S simultaneously reduces the average ASR on both noise and patch UAPs by 85% compared to the ASR on no defence, while retaining more than 95% of the original CP. When averaging over both noise and patch attacks, 3.0M+4.2S achieves much better performance only at a marginal cost of CP. Additionally, the layer that flags the input indicates to the defender whether the input is more likely to have a noise or a patch UAP. Table 6.3 shows that 3.0M and 4.2S are good for decreasing ASR on noise and patch attacks respectively.

Table 6.3: Detection performance for adversarial noise and patch UAPs against ResNet50.

Perturbation	Attack	UER	AUC			ASR (%)		
			3.0M	4.2S	3.0M+4.2S	3.0M	4.2S	3.0M+4.2S
Mask ($\epsilon = 10$)	SGD	92.50	0.981	0.332	0.966	6.29	92.11	6.14
	SGD 611	90.05	0.933	0.509	0.897	24.80	85.70	22.43
	Perlin Noise	59.33	0.940	0.853	0.919	12.20	58.86	11.98
Patch (16% Area)	Target 611	92.53	0.381	0.982	0.966	90.53	5.15	4.90
	Target 859	89.03	0.416	0.965	0.935	86.60	14.83	14.46
	Target 960	84.39	0.385	0.956	0.922	82.75	14.14	13.81

Table 6.4: Metrics (in %) summarizing metrics from Table 6.3 configurations against adversarial noise and patch UAPs on ResNet50.

	No Defence	3.0M	4.2S	3.0M+4.2S
Avg. ASR: Mask	80.27	14.43	78.89	13.52
Avg. ASR: Patch	88.65	86.63	11.37	11.06
Clean Performance (CP)	76.13	73.55	74.17	72.91

Table 6.5: Average time for inference and detection in milliseconds (ms) on a batch size of 64 images on ResNet50.

	No Defence	3.0M	4.2S	3.0M+4.2S
Time per batch	75.53	110.02	108.27	130.82
Time per image	1.18	1.72	1.69	2.04
Latency per image	-	0.54	0.51	0.86

Runtime Analysis. We measure the latency of HyperNeuron when defending the ResNet50 model from both adversarial noise and adversarial patch attacks in Table 6.5. The detection component of HyperNeuron is run on the CPU, and the model is run on a GeForce RTX 2080 Ti GPU. When an input is given to the model, HyperNeuron saves the aggregated statistics of the activations on the observed layers, computes the mean of the top percentile, then compares it to the threshold set. We show the runtime performance of HyperNeuron with the three detection configurations: 3.0M, 4.2S, 3.0M+4.2S. These three correspond to the noise-only, patch-only, and combined detection configurations respectively.

Our results show that the runtime with HyperNeuron remains on the same order of magnitude as inference without the defence. This is in contrast to other defences like SentiNet that introduce orders of magnitude more latency per batch at 2 seconds (2,000ms) [67] on similar hardware. The per-image latency that HyperNeuron introduces is a fraction of a thousandth of a second demonstrates that it is extremely efficient and suitable for real-time applications such as image classification or object detection in videos of up to 60 frames per second. The added 55 millisecond latency from HyperNeuron would then be negligible.

To summarise, HyperNeuron simultaneously decreases the ASR of both adversarial patch and adversarial noise UAPs at low latency without significantly degrading the model’s CP. These advantages make it viable for practical applications as it can be run in real-time and detect

both types of perturbations. To better put the performance of HyperNeuron into context, we next compare the performance of HyperNeuron against existing defences designed for UAPs.

6.5 Comparison with Existing Defences

Existing defences for adversarial noise UAPs denoise the input or retrain the model to correct its output on perturbed inputs [75, 129, 53, 77]. These approaches are good for correcting model predictions on tested universal attacks without having to intervene during model inference. However, no information is gained as it is unclear if the errors made by the model are caused by maliciously altered inputs or by natural model error. HyperNeuron takes a different approach by performing detection instead. Detection is more advantageous as it alerts the user of active attacks, and helps identify input sources with compromised integrity. So long as existing neural networks remain largely vulnerable to adversarial examples and do not achieve near-perfect accuracy on clean datasets, detecting attacks would have this advantage.

6.5.1 Related Work

HyperNeuron is the first UAP defence to demonstrate simultaneous efficacy against both adversarial noise and adversarial patch UAP attacks. To be able to compare with the existing literature, we compare our method against defences designed for each of these threats individually. We give a brief summary of the defences we compare against, these were discussed in greater detail in Section 2.4.

For adversarial noise UAPs, we compare against Selective Feature Regeneration (SFR) [77] and Adversarial Training (AT) [23]. SFR works by regenerating the top 50% of adversarially susceptible activations and retraining it with feature regeneration units (FRU). They first identify features of the pre-trained model that are most susceptible to adversarial attacks and then they deploy these FRUs which are trained to transform some convolutional filter activations into more resilient features that are robust to universal attacks. SFR is chosen as it outperforms

other UAP defences for recovering model accuracy on adversarial noise UAPs [77].

AT is chosen as it has been the benchmark defence in the adversarial machine learning literature. It is a straightforward method in that it augments the training set with adversarial examples. Additionally, we have yet to see work that shows the effectiveness of AT in improving model robustness against UAPs. We use the fast AT algorithm [130], which significantly speeds up AT by computing the gradients for adversarial perturbations and model weights simultaneously.

For adversarial patch UAPs, we compare HyperNeuron with SentiNet, a defence that has been shown to be able to detect adversarial patch UAPs [67]. Given an input image, SentiNet first uses model explainability techniques and object detection to identify the input image regions that most highly influence the model prediction. It then applies this extracted region on a set of trusted test inputs to observe the model output. SentiNet then compares the behaviour of the model on this synthetic dataset versus its behaviour on the trusted inputs to reach a prediction on whether the original input image is affected by an adversarial perturbation.

6.5.2 Comparison & Results

To compare HyperNeuron with non-detection defences, we use Attack Success Rate (ASR) and Clean Performance (CP) metrics. Note that ASR and CP are defined with detection in mind. For non-detection defences, ASR is equivalent to UER and CP is the accuracy on the clean dataset as there is no flagging mechanism to reject inputs. These metrics make sense as they measure the relevant performance of the model on the perturbed and clean datasets for both detection and non-detection defences. When comparing methods, *a better defence will have lower ASR and higher CP.*

Adversarial Noise. We compare against SFR and a fast AT algorithm with ℓ_∞ -norm = 4 on a ResNet50 model. Table 6.6 shows that HyperNeuron (3.0M+4.2S) achieves much lower average ASR for both adversarial noise and patch attacks when compared to SFR and AT. HyperNeuron decreases the ASR closer to zero as it can reject perturbed inputs, whereas the other non-detection defences are restricted by their error rate on the clean dataset. These

Table 6.6: Comparison of HyperNeuron with non-detection defences over noise and patch UAPs on ResNet models for ImageNet. Lower ASR and higher CP indicate better model performance. All metrics are in %.

	No Defence	SFR[77]	AT[130]	M+S (Ours)
Avg. ASR: Mask	80.27	29.77	48.72	13.52
Avg. ASR: Patch	88.65	38.41	78.88	11.06
CP	76.13	75.47	53.83	72.91

Table 6.7: Comparison of HyperNeuron with non-detection defences on noise and patch UAPs on ResNet models for ImageNet. First six rows show ASR for each attack. Metrics are in %.

Perturbation	Attack	SFR ^a [77]	AT ^b [130]	3.0M+4.2S ^b (Ours)
Mask ($\varepsilon = 10$)	SGD	25.39	51.43	6.14
	SGD 611	25.12	46.75	22.43
	Perlin	38.80	47.97	11.98
Patch (16% Area)	Target 611	38.28	79.10	4.90
	Target 859	38.60	79.43	14.46
	Target 960	38.36	78.10	13.81
Avg. ASR: Mask		29.77	48.72	13.52
Avg. ASR: Patch		38.41	78.88	11.06
CP (No Attack)		77.47	53.83	68.48

^aBase model is a ResNet152 with 79% accuracy undefended.

^bBase model is a ResNet50 with 76% accuracy undefended.

defences like SFR and AT attempt to restore the model’s clean performance on the perturbed images. This means that the ASR of UAP attacks on these defences would be bounded below by their error on non-perturbed images. Thus, if the model’s performance on the clean dataset is far from perfect (e.g. ResNet50 has 23.87% error on benign ImageNet) then the ASR is greater than this value.

SFR is able to decrease the ASR of SGD much closer to its clean error. However Perlin Noise, which is a non-gradient based attack, achieves a better ASR than SGD. This shows that SFR does not generalise as well to unseen UAPs when compared to HyperNeuron. For noise UAPs against AT, we see in Table 6.6 that their average ASR of 48.72% is very close to its error on the clean dataset which is 46.17%. This means that AT with FGSM does make the model insensitive to noise UAPs. However, as the CP is very low at 53.83%, the model still makes many errors on both perturbed and non-perturbed images. Additionally, AT does not improve the robustness against adversarial patch attacks. This is in line with recent results showing

that AT defences have difficulty achieving robustness against multiple types of perturbations simultaneously [131, 73]. Unlike SFR and AT, HyperNeuron does not attempt to recover the correct class label but can give information on the integrity of input sources. The automatic recovery of labels is an advantage if the model has near-perfect accuracy or if intervening with the deployed model is costly. Feedback on the integrity of inputs is useful if the defender actively monitors model inputs for malicious attacks or data drift. The defender has to consider that aspect when evaluating and deploying a defence.

Adversarial Patch. We compare HyperNeuron with SentiNet [67]. Since their defence is a detector, we can directly compare the TP and FP rates on adversarial patch attacks. We measure 98.5% TP at 4.8% FP for an adversarial patch that is roughly 25% of the image area, which targets the *toaster* (#611) class on a VGG16 trained for ImageNet. In a similar attack on VGG16, HyperNeuron achieves 97.9% TP at 4.9% FP for 3.0M+4.2S. SentiNet appears to achieve slightly better performance, but with a latency of 2 seconds on similar hardware. In contrast, HyperNeuron is primarily run on the CPU and introduces a drastically reduced latency of 0.055 seconds for a batch of 64 images. Thus, HyperNeuron is orders of magnitude faster, and whilst SentiNet is not viable for real-time application HyperNeuron (0.055 seconds) can be used for live detection. Taking these differences into consideration, HyperNeuron could be used to complement more accurate but computationally expensive defences like SentiNet by acting as a pre-filter for inputs to analyze. A defender should consider these trade-offs of speed and performance depending on their model’s application.

To summarise, HyperNeuron has the distinct advantage of being able to simultaneously defend against both adversarial noise and adversarial patch UAPs with significantly lower latency, while maintaining competitive or better ASR and CP when compared to existing defences. HyperNeuron has sufficiently low latency, to be easily applied to existing models without significantly increasing computation costs, and this makes it a practical choice for deflecting UAP attacks. Now that we have shown its efficacy, we evaluate its performance in the presence of a defence-aware adversary.

6.6 Defence-aware Adaptive Attack

We consider the case where the attacker has PK of the model, including knowledge of the detection mechanism. The attacker’s goal is to maximise ASR by generating a UAP that also evades HyperNeuron. ASR is an appropriate metric as it measures the proportion of unflagged misclassifications made on a perturbed set of inputs. UAP generation and evaluation is done on the 50,000 ImageNet validation set, and the detector thresholds are set to have 5% FP rate on the clean baseline for 3.0M and 4.2S. 3.0M+4.2S flags the input if it is flagged by either. To generate an adaptive attack, we focus primarily on adversarial noise UAPs and assume the defender uses 3.0M for detection as it has shown to be the most capable for detecting noise UAPs. We apply SGD optimization to a new loss function which we call the **Baseline-Feature attack**:

$$\sum_i (\mathcal{L}(x_i + \delta) - \text{ReLU}(M_k(x_i + \delta) - \mu_k)) \quad (6.4)$$

The first term is taken from SGD to maximise the model’s loss, while the second term minimises the resulting UAP’s detectability. The ReLU penalises the perturbation when its activation exceeds the value of the baseline vector, an assumption that HyperNeuron relies on. This formulation is similar to that of the feature adversary [132], but instead of optimizing towards the activations of a guide image, we optimise against the mean baseline μ_k used by the detector. We compare this Baseline-Feature attack with SGD, the primary attack for maximizing ASR on an undefended model.

Figure 6.10 shows that the Baseline-Feature attack improves the overall ASR even when HyperNeuron is present. It is able to recover an average of 53% of the original ASR when compared to SGD on an undefended model. However, the Baseline-Feature UAP generated has significantly less ASR on the undefended model (note $\text{ASR} = \text{UER}$ when there is no detector). This shows that the detector still mitigates a large fraction of the UAP attack as it forces the attacker to sacrifice attack UER for a decrease in detectability, supporting the idea of the defendability-universality trade-off.

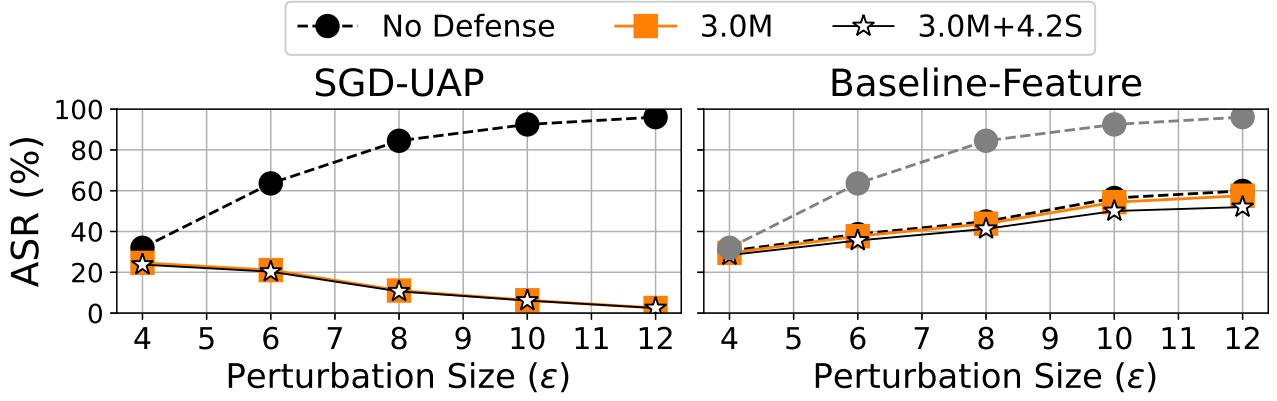


Figure 6.10: Defence-aware attacks against HyperNeuron for ResNet50 on ImageNet validation. ASR of SGD on the undefended model is shown on the right subplot in gray for reference.

6.7 Application: Perceptual Ad-blocking

Having seen the effectiveness of HyperNeuron against both adversarial noise and patch UAP attacks on ImageNet, we now test against perceptual ad-blocking as another example of a practical setting, but this time in the digital domain. The previous practical attack for the physical domain came in the form of adversarial patches. This case is of interest as the application requires real-time detection of ads to block them from the user. We implement two UAP attacks against page-based perceptual ad-blockers and evaluate HyperNeuron on these attacks.

Perceptual ad-blocking detects online advertisements on a web page based on their visual content. However, Tramer et al. have shown that existing methods are vulnerable to UAPs [58]. As a webpage has very dynamic and constantly changing content, adversarial perturbations applied would necessarily have to be *universal* so that they are robust to arbitrary changes in the content of the web page. We take advantage of this property to detect the attacks.

Model. Page-based perceptual ad-blockers, such as *Sentinel* [133], identify images on rendered web pages. This is an object detection model that outputs the bounding box coordinates of the detected ads and a confidence score. For details on the object detection task, we refer the reader to Section 3.5.1. For ad-blocking, there are only two class labels (“ad” and “not ad”). The model used is based on YOLOv3 with a DarkNet53 feature extractor [102], which has

comparable performance to ResNet50 on ImageNet. We use the pre-trained object detector made available by Tramer et al. [58], which achieves an 80% true positive rate for detecting ads on our test set of 225 web page screenshots.

Attacks. We focus on two types of attacks: the *overlay* attack, where the UAP enables ads to evade detection, and the *overflow* attack, where the UAP overflows the model with false ad predictions [58]. Perturbations are generated by computing a small noise, then tiling the web page with the computed noise. SGD is used to optimise the noise for either hiding all ads or overflowing the model. These can be seen as targeted UAP attacks. We apply the two attacks on the test set of 225 web page screenshots for $\varepsilon = 0.3, 0.5, 1, 2, 4, 8$ under the ℓ_∞ -norm perturbation constraints. For the overlay attack, we measure the total number of ads detected and “ad decrease”—the percentage of pages where the UAP has decreased the number of ads detected. For the overflow attack, we measure the total number of ads detected and “ad increase”—the percentage of pages where the UAP has increased the number of ads detected. These attacks are extremely effective even for very low perturbation values as seen in Figure 6.12. The overlay attack halves the total number of ads detected at $\varepsilon = 1$ and the overflow attack doubles the total number of ads detected at $\varepsilon = 0.3$. These results demonstrate the fragility of the ad-blockers to visually imperceptible UAPs. This ε is extremely small as it is a fraction of the maximum possible pixel perturbation of 255.

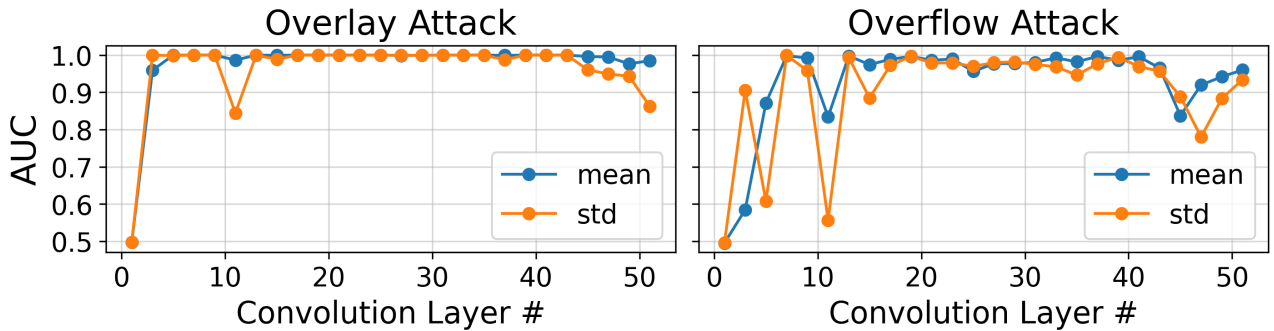


Figure 6.11: AUC per layer of HyperNeuron for YOLOv3 against overlay and overflow attacks at $\varepsilon = 1$.

Detection Results. To detect attacks, we monitor the convolution blocks of the DarkNet53. We see in Figures 6.11 and 6.12 the performance for each layer and find that mean aggregation on layers 7, 13, 19, and 39 all achieve near perfect (>0.99) AUC in detecting both types of

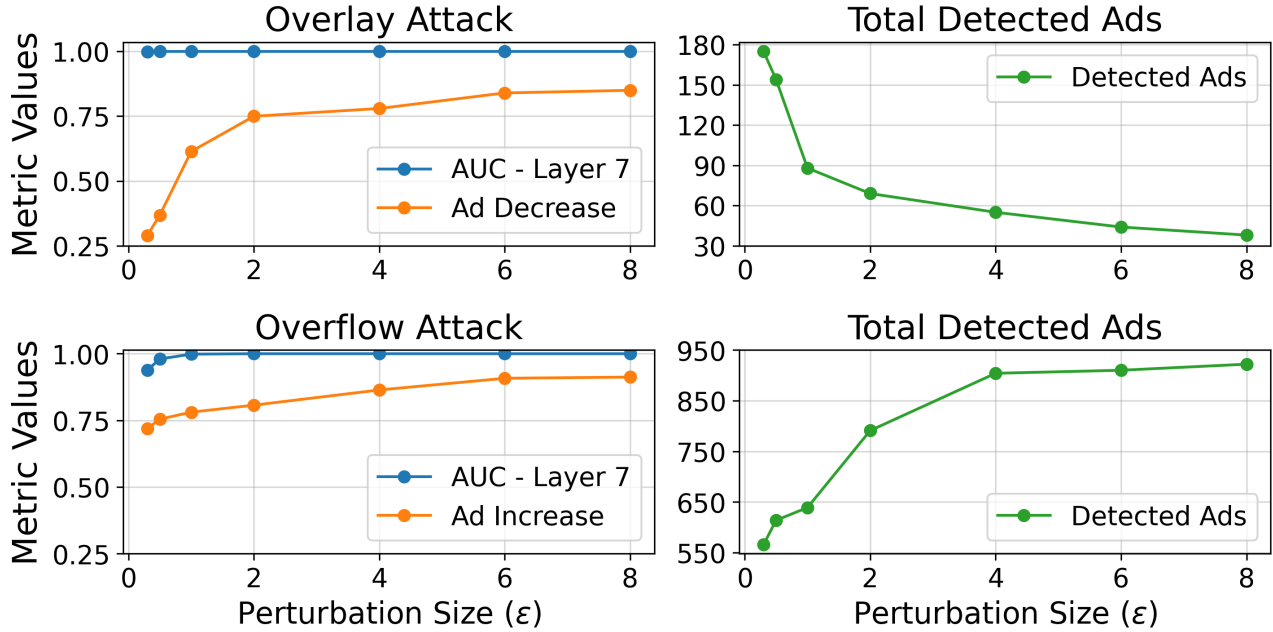


Figure 6.12: Evaluation metrics for YOLOv3 against overlay and overflow attacks for different ϵ . The plots on the right show the total number of detected ads.

attacks, even for small perturbation values. HyperNeuron is able to detect all attacks with near-perfect AUC, as seen in Figure 6.12. This shows that YOLOv3 is highly vulnerable to UAPs but these UAPs are highly detectable by HyperNeuron. Thus, in tasks where the universality of adversarial perturbations is important in generating realistic and successful attacks, we show that HyperNeuron is able to detect UAPs with near-perfect AUC for small perturbations. The attacks are most evident in the middle layers. There is a hyper-activation of neurons in both attacks, which suggests that there may be distinct sets of neurons that inhibit and excite the detection of ads and the overlay and overflow UAPs activate distinct subsets.

6.8 Limitations & Trade-offs of HyperNeuron

Although HyperNeuron is not perfect in reducing the ASR of all UAPs to 0, it is successful in mitigating the ASR on a large variety of both adversarial noise and patch UAPs, and it is readily scalable and applicable in practice. We also see that UAPs that evade our detector have reduced UER on an undefended model. HyperNeuron compares favorably against existing UAP defences while adding minimal computational overhead.

Simplicity & Scalability. Simple heuristics are used in HyperNeuron (mean, standard deviation, z-score, top- t percentile) to reduce the dimensionality of activation vectors and the complexity of the detector. This is reflected by the incredibly low latency it introduces. An additional advantage of using these heuristics is that their failure cases are well understood and can be explained; we are able to calibrate and interpret our detection results when it fails.

Limitations. For adversarial patches, SentiNet is an example of a detector with slightly better performance, but with orders of magnitude larger latency which makes it impractical for live detection. In contrast, HyperNeuron is more practical and can be combined with other defences. HyperNeuron complements these more effective but expensive defences as it can act as a pre-filter to decrease the total computational cost.

Decreasing our detector’s ASR against the defence-aware attacks covered in this section would require increasing the number of FPs and negatively affecting CP. This would favor the defender more if there is a higher frequency of attacks on the model, but not necessarily when attacks are rare or infrequent. This trade-off should be made depending on the application and a risk assessment of both the *likelihood and severity* for each type of error (FPs and FNs).

We have empirically shown that the defendability-universality trade-off holds for known UAP attacks on ImageNet. This includes a novel adaptive UAP feature attack that is designed specifically to evade HyperNeuron. Increasing the dispersion of a UAP’s hidden layer activations improves its chances of evading detection, though this still greatly sacrifices its universality when compared to the SGD attack.

6.9 Discussion

We highlight observations made when evaluating our defence and the implications of our results.

1. Defendability-universality trade-off. The success of HyperNeuron and our extensive results have led us to a better understanding of how to detect UAPs via the hyper-activation of neurons. Although defending against input-dependent attacks is extremely difficult [30, 29,

25, 127], we present a more hopeful picture against universal attacks. We demonstrate how it can be made more challenging for an adversary to simultaneously evade both the model and detector on a large subset of inputs when using a single UAP, thus forcing a trade-off between UAP universality and detectability. As the universality of adversarial perturbations is a cornerstone for many practical and physically-realizable attacks, this gives us confidence that more robust defences against adversarial attacks can be developed for real-world machine learning applications.

2. Disadvantages of existing non-detection defences. Non-detection defences do not distinguish between adversarial or natural model error when detection can. Thus, for models that are actively monitored, detection-based defences provide more information on the integrity of their input sources. When the model makes an error, it is not clear if it was due to an adversary or model error. This disadvantage became evident when we compared the number of misclassifications on perturbed images using the Attack Success Rate (ASR) for various defences. The ASR of non-detection defences could only be as good as their error on the clean inputs, whereas detection-based defences like HyperNeuron are able to flag any suspicious inputs. This is problematic for non-detection defences especially when the model has far from perfect accuracy on the clean dataset, which is the case for ImageNet. Thus, detection-based defences seem better in this case, as it gives the defender more information on the integrity of their input source. HyperNeuron was able to reduce ASR much better than the non-detection defences like AT and SFR since the model had high error on the clean dataset to begin with. The error on the clean dataset acted as a lower bound for the ASR on these non-detection defences. This result shows that for models with far from perfect clean accuracy, recovering clean accuracy on a perturbed dataset is not enough to significantly reduce ASR.

6.10 Conclusion

In this chapter, we have designed HyperNeuron, a new detection algorithm that successfully identifies UAP attacks across the entire range of experiments that we have conducted. The

extensive testing on benchmark datasets, a broad selection of UAP attacks, and on real applications such as adversarial patches and perceptual ad-blocking heavily suggests the presence of a defendability-universality trade-off inherent in UAPs. Equally important, this evaluation has led to a new understanding of how UAPs manifest in DNNs through neuron hyper-activations. HyperNeuron runs with minimal latency of 0.86ms per image for ResNet50 image classification and greatly outperforms existing UAP defences in terms of latency, while comparing favorably in terms of detection performance. It is also a general algorithm that can be applied to many tasks including image classification and object detection. To summarise, we enumerate the main findings of this chapter:

- We propose HyperNeuron, a fast and scalable detector that is able to simultaneously detect both adversarial noise and adversarial patch universal attacks with extremely low latency. This is based on our observation that UAPs hyper-activate neuron values on specific layers of a DNN, which allows for their detection.
- We show that UAPs exhibit a trade-off between detectability and universality, even for defence-aware attacks. Our results provide a more general methodology for comparing and evaluating existing and future universal attacks and defences.
- We highlight the role of UAPs in existing practical attacks on DNNs, and show that HyperNeuron can successfully detect attacks in realistic scenarios, such as adversarial patches against image classification and adversarial noise on perceptual ad-blocking.
- Our analysis also demonstrates the advantages and disadvantages of existing UAP defences, and how a detection-based defence can fare better than non-detection ones for models with *far from perfect* accuracy.

Chapter 7

Conclusion

Advances in DNNs have transformed the field of computer vision as they have demonstrated increased utility in many popular and security-relevant computer vision applications such as image recognition, object detection, autonomous vehicles, and medical imaging. Despite this, computer vision DNNs have been shown to be vulnerable to systemic attacks in the form of UAPs. The efficiency and practicality of UAP attacks is a major threat as it enables many concrete attacks to be deployed, drastically lowering the bar for adversaries to undermine the integrity and reliability of these machine learning systems. It is therefore important that we can effectively mitigate UAP attacks in practical and scalable ways without compromising on model accuracy and utility. In this chapter, we briefly summarise our main achievements and discuss future research directions.

7.1 Summary of Thesis Achievements

In this thesis, we have advanced the understanding of and mitigation of UAP attacks. For understanding UAPs, we used procedural noise (Chapter 3) to replicate the distinct visual patterns present in UAPs in the input space. We showed that our procedural noise patterns can be used in very efficient zero-knowledge UAP attacks. These results have notable implications. The universality of our ZK method introduces the possibility for large-scale attacks on

DNN-based machine learning services, and the use of Bayesian optimization makes ZK attacks significantly more realisable. Compared to the state-of-the-art at the time, we were able to reduce the number of queries required to perform successful input-dependent ZK attacks by up to 2 orders of magnitude: from 888 to 8.4 queries on average. The ease in which procedural noise is generated and the lack of model access required for the attack to be successful tips the balance in favour of adversaries. The results also suggested that procedural noise attacks exploit low-level features that DNNs are sensitive towards. This has worrying implications on the safety of transfer learning and model reuse, as it is often these low-level features that are preserved when retraining models or reusing components of pre-trained models for new tasks. That our procedural noise attack generalises across models and computer vision tasks is of great concern and presents a real threat to computer vision DNNs as we were able to use the same procedural noise algorithms to attack another computer vision task in object detection. This highlights the need to harden models against UAPs.

To investigate the susceptibility of low-level features learned by models to adversarial attacks, we explored the UAP robustness of shape-biased and texture-biased models (Chapter 4). Although compelling, neither the shape nor texture hypotheses seem to adequately address the phenomenon of adversarial perturbations. Having evaluated the robustness of these methods, we see that training on the shape-biased Stylized-ImageNet does not mitigate any vulnerability to UAPs, but instead shifts the sets of features that these models are vulnerable to. In our analysis, we found that generated UAPs favour classification towards specific class labels, and these dominant class labels align with the visual appearance of their corresponding UAPs. Our findings give insight into how the choice of training datasets affect the visual appearance of UAPs and the resulting robustness of the models to certain class labels. That there are generally successful targeted UAPs for almost all of 1,000 ImageNet class labels for each of the models, regardless if they were trained on ImageNet or Stylized-ImageNet, shows the breadth of the attack surface and model vulnerabilities to UAPs. When the same model architecture is trained on different datasets, the features it is vulnerable to shifts. Despite UAPs being a weakness, we argue that the novel analysis based on UAP’s dominant class labels introduced in our study can be adopted for evaluating and understanding the robustness of computer vision

models for various tasks, datasets, and model architectures. Finally, we propose ensembles of texture and shape-biased models, showing that it can maintain clean performance while improving on universal adversarial robustness.

The initial improvement in using ensembles to increase UAP robustness without sacrificing clean performance is a step in the right direction. To take this idea further, we formulated the accuracy-robustness trade-off that model owners have to face, where the goal is to robustify models to UAPs without significantly degrading accuracy. Our results showed that Jacobian Regularisation (JR) combined with model ensembles has many advantages in greatly improving model robustness to UAPs without compromising on model accuracy (Chapter 5). We proved that applying JR and increasing the number of base learners for a model ensemble each on their own improves the theoretical bounds indicative of increased robustness to UAPs. We then empirically verified our theoretical derivations by applying JR to neural networks trained on popular benchmark datasets and evaluating their robustness to various UAPs. We found that even a small amount of JR drastically improves model robustness against many universal attacks with negligible downsides to clean performance. To augment this, our proposed Jacobian Ensembles achieves previously unseen levels of accuracy and robustness, greatly improving over previous methods that tend to skew towards only either accuracy or robustness. Thus, it is indeed possible to get the best of both worlds.

Jacobian Ensembles greatly improve the stability of models to small perturbations. However, degradation in model performance is inevitable for sufficiently large perturbations. To solve this problem, we demonstrated that larger perturbations are necessarily more detectable by constructing a robust detection-based defence in HyperNeuron (Chapter 6). Our results showed the effectiveness and generalisability of HyperNeuron to attacks on DNNs under different constraints (noise, patches) and tasks (image classification, object detection), as well as realistic use-cases such as adversarial patches and perceptual ad-blocking. HyperNeuron introduces minimal latency, making it usable in real-time applications such as object detection and perceptual ad-blocking. It can also be a viable pre-filter to complement more computationally expensive defence mechanisms. And our evaluations with HyperNeuron have led to a new understanding of how UAPs manifest in DNNs through neuron hyper-activations. Finally, we developed a

defence-aware adaptive attack against HyperNeuron to show that, even in the worst-case scenario, the trade-off between detectability and universality holds as the attack greatly sacrificed its effectiveness to reduce its detectability. This demonstrates the defendability-universality trade-off, where attackers necessarily have to sacrifice their attack’s universality in exchange for the ability to evade our defence. Advancing these two key trade-offs, with the other being the accuracy-robustness trade-off, gives us hope in the future of robust machine learning. Overall, our findings enable us to put defences for neural networks against realistic and systemic UAP attacks on more practical footing.

7.2 Future Work

We believe that constructing defences to universal attacks is an important component towards solving the wider problem of adversarial examples for DNNs. As input-dependent adversarial attacks have been shown to be very difficult to defend against, thoroughly defending against UAPs is a more achievable first step. This is an important challenge that must be overcome to secure DNNs in widely-used and security-critical domains.

From our studies with procedural noise and models trained on texture and shape-biased datasets, we revealed that many UAPs exploit patterns that either correspond to low-level features or particular class labels. This finding can perhaps be extended to other applications outside computer vision, particularly in signal processing domains, where analogues to low-level features or procedural noise can be found. Bayesian optimisation can be similarly used to enhance novel ZK attacks, and the analysis of dominant labels of UAPs can reveal learned features and vulnerabilities of models in different application domains. The characterisation of the entire UAP attack surface remains an open challenge. From our initial results, we found class labels of the dataset to be a useful tool in describing these. However, in applications where labeled data is not as descriptive like in binary classification or unsupervised learning, these labels may not be as informative. In these cases, a more detailed analysis of either the training dataset or the hidden layer activations can make for better characterisations of the attack surface.

For the mitigation of UAP attacks, we have made considerable progress on two sides. The first being improvements in model stability with Jacobian Ensembles, leading to robustness against UAP attacks that use small perturbations. The second being improvements in the detection of UAPs with HyperNeuron, with notable success in detecting significantly large perturbations. When it comes to practice, the ideal defence should be to handle UAP attacks that make use of both small and large perturbation sizes. Thus, a logical next step would be to combine two such approaches that can combat each threat individually as these would complement each other. We encourage future work to explore defence frameworks that can unify multiple types of defences to achieve superior performance across various attacks without sacrificing clean accuracy. The main challenge would come in navigating the accuracy-robustness trade-off while taking into account various levels of attacker strength such as perturbation magnitude to reconcile any conflicts when combining different defences.

We also hope that future work will explore in more depth a closer integration between attack detection and model retraining. Although we lightly touched on this with HyperNeuron, improving how defenders deal with detected adversarial attacks could further enhance the robustness of the entire machine learning system to universal attacks. It is possible that an optimal solution has to be modular in nature to deal with the increasing diversity of adversarial threats to machine learning models, as such, a unifying framework that can manage different threats and defensive mechanisms is worth exploring.

Bibliography

- [1] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1765–1773, 2017.
- [2] Konda R. Mopuri, Aditya Ganeshan, and Venkatesh B. Radhakrishnan. Generalizable data-free objective for crafting universal adversarial perturbations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.
- [3] Valentin Khruikov and Ivan Oseledets. Art of singular vectors and universal adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8562–8570, 2018.
- [4] Yusuke Tsuzuku and Issei Sato. On the structural sensitivity of deep convolutional networks to the directions of fourier basis functions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 51–60, 2019.
- [5] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 740–755. Springer, 2014.
- [6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

- [7] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [8] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, 2017.
- [9] Andrew Ilyas, Logan Engstrom, and Aleksander Madry. Prior convictions: Black-box adversarial attacks with bandits and priors. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.
- [10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1097–1105, 2012.
- [11] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.
- [12] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.
- [13] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16*, page 1528–1540, New York, NY, USA, 2016. Association for Computing Machinery.
- [14] Guanghan Ning, Zhi Zhang, Chen Huang, Xiaobo Ren, Haohong Wang, Canhui Cai, and Zhihai He. Spatially supervised recurrent convolutional neural networks for visual object tracking. In *2017 IEEE international symposium on circuits and systems (ISCAS)*, pages 1–4. IEEE, 2017.

- [15] Sampo Kuutti, Richard Bowden, Yaochu Jin, Phil Barber, and Saber Fallah. A survey of deep learning applications to autonomous vehicle control. *IEEE Transactions on Intelligent Transportation Systems*, 22(2):712–733, 2020.
- [16] Zhongyuan Hau, Kenneth T. Co, Soteris Demetriou, and Emil C. Lupu. Object removal attacks on lidar-based 3d object detectors. *arXiv preprint arXiv:2102.03722*, 2021.
- [17] Zhongyuan Hau, Soteris Demetriou, Luis Muñoz-González, and Emil C. Lupu. Shadow-catcher: Looking into shadows to detect ghost objects in autonomous vehicle 3d sensing. In *European Symposium on Research in Computer Security (ESORICS)*, pages 691–711. Springer, 2021.
- [18] Konstantinos Kamnitsas, Christian Ledig, Virginia FJ Newcombe, Joanna P. Simpson, Andrew D. Kane, David K. Menon, Daniel Rueckert, and Ben Glocker. Efficient multi-scale 3d cnn with fully connected crf for accurate brain lesion segmentation. *Medical image analysis*, 36:61–78, 2017.
- [19] Spyridon Bakas, Mauricio Reyes, Andras Jakab, Stefan Bauer, Markus Rempfler, Alessandro Crimi, Russell Takeshi Shinohara, Christoph Berger, Sung Min Ha, Martin Rozycki, et al. Identifying the best machine learning algorithms for brain tumor segmentation, progression assessment, and overall survival prediction in the brats challenge. *arXiv preprint arXiv:1811.02629*, 2018.
- [20] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.
- [21] Battista Biggio, Iginio Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-KDD)*, pages 387–402. Springer, 2013.

- [22] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Dumitru Erhan, Joan Bruna, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.
- [23] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.
- [24] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.
- [25] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy (S&P)*, pages 39–57. IEEE, 2017.
- [26] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In *International Conference on Machine Learning (ICML)*, pages 284–293. PMLR, 2018.
- [27] Simen Thys, Wiebe Van Ranst, and Toon Goedemé. Fooling automated surveillance cameras: adversarial patches to attack person detection. In *CVPRW: Workshop on The Bright and Dark Sides of Computer Vision: Challenges and Opportunities for Privacy and Security*, 2019.
- [28] Kenneth T. Co, Luis Muñoz González, Sixte de Maupeou, and Emil C. Lupu. Procedural noise adversarial examples for black-box attacks on deep convolutional networks. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS '19*, pages 275–289, New York, NY, USA, 2019. ACM.
- [29] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (AISec)*, pages 3–14. ACM, 2017.

- [30] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 274–283, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.
- [31] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness. *arXiv preprint arXiv:1902.06705*, 2019.
- [32] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6:14410–14430, 2018.
- [33] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P. Wellman. Sok: Security and privacy in machine learning. In *Proceedings of the 2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 399–414. IEEE, 2018.
- [34] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2961–2969, 2017.
- [35] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 91–99, 2015.
- [36] Konda Reddy Mopuri, Utsav Garg, and R. Venkatesh Babu. Fast feature fool: A data independent approach to universal adversarial perturbations. *arXiv preprint arXiv:1707.05572*, 2017.
- [37] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [38] Dmytro Mishkin and Jiri Matas. All you need is a good init. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

- [39] Andrew L. Maas, Awni Y. Hannun, Andrew Y. Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning (ICML)*, 2013.
- [40] Luis Munoz-Gonzalez and Emil C. Lupu. The secret of machine learning. *ITNow*, 60(1):38–39, 2018.
- [41] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.
- [42] Kenneth T. Co, David Martinez Rego, and Emil C. Lupu. Jacobian regularization for mitigating universal adversarial perturbations. In *International Conference on Artificial Neural Networks (ICANN)*, pages 202–213. Springer, 2021.
- [43] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in Machine Learning: From Phenomena to Black-box Attacks using Adversarial Samples. *arXiv preprint arXiv:1605.07277*, 2016.
- [44] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 506–519. ACM, 2017.
- [45] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 321–338, 2019.
- [46] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436, 2015.
- [47] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In

- Proceedings of the 2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387. IEEE, 2016.
- [48] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Proceedings of the 2016 IEEE Symposium on Security and Privacy (S&P)*, pages 582–597. IEEE, 2016.
- [49] David Warde-Farley and Ian Goodfellow. Adversarial Perturbations of Deep Neural Networks. In *Perturbations, Optimization, and Statistics*. The MIT Press, 12 2016.
- [50] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.
- [51] Eric Wong, Frank Schmidt, and Zico Kolter. Wasserstein adversarial examples via projected sinkhorn iterations. In *International Conference on Machine Learning (ICML)*, pages 6808–6817. PMLR, 2019.
- [52] Mahmood Sharif, Lujo Bauer, and Michael K. Reiter. On the suitability of lp-norms for creating and preventing adversarial examples. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1605–1613, 2018.
- [53] Ali Shafahi, Mahyar Najibi, Zheng Xu, John Dickerson, Larry S. Davis, and Tom Goldstein. Universal adversarial training. *arXiv preprint arXiv:1811.11304*, 2018.
- [54] Tom B. Brown and Dandelion Mané. Adversarial patch. *arXiv preprint arXiv:1712.09665*, 2017.
- [55] Ian Goodfellow. Defense against the dark arts: An overview of adversarial example security research and future research directions. *arXiv preprint arXiv:1806.04169*, 2018.
- [56] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2154–2156, 2018.

- [57] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.
- [58] Florian Tramèr, Pascal Dupré, Gili Rusak, Giancarlo Pellegrino, and Dan Boneh. Adversarial: Perceptual ad blocking meets adversarial machine learning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS '19*, page 2005–2021, New York, NY, USA, 2019. Association for Computing Machinery.
- [59] David W. Boyd. The power method for lp norms. *Linear Algebra and its Applications*, 9:95–101, 1974.
- [60] Kevin Eykholt, Ivan Evtimov, Earlene Fernandes, Bo Li, Amir Rahmati, Florian Tramer, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Physical adversarial examples for object detectors. In *12th USENIX Workshop on Offensive Technologies (WOOT 18)*, 2018.
- [61] Kevin Eykholt, Ivan Evtimov, Earlene Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1625–1634, 2018.
- [62] Xin Liu, Huanrui Yang, Ziwei Liu, Linghao Song, Hai Li, and Yiran Chen. Dpatch: An adversarial patch attack on object detectors. *arXiv preprint arXiv:1806.02299*, 2018.
- [63] Nicholas Carlini. *Evaluation and Design of Robust Neural Network Defenses*. PhD thesis, University of California at Berkeley, 2018.
- [64] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.
- [65] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning (ICML)*, pages 1310–1320. PMLR, 2019.

- [66] Aman Sinha, Hongseok Namkoong, and John Duchi. Certifiable distributional robustness with principled adversarial training. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.
- [67] Edward Chou, Florian Tramèr, and Giancarlo Pellegrino. Sentinet: Detecting localized universal attacks against deep learning systems. In *DLS: Deep Learning and Security Workshop*, 2020.
- [68] Sanjay Kariyappa and Moinuddin K Qureshi. Improving adversarial robustness of ensembles with diversity training. *arXiv preprint arXiv:1901.09981*, 2019.
- [69] Tianyu Pang, Kun Xu, Chao Du, Ning Chen, and Jun Zhu. Improving adversarial robustness via promoting ensemble diversity. In *International Conference on Machine Learning (ICML)*, pages 4970–4979. PMLR, 2019.
- [70] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning (ICML)*, pages 854–863. PMLR, 2017.
- [71] Judy Hoffman, Daniel A. Roberts, and Sho Yaida. Robust learning with jacobian regularization. *arXiv preprint arXiv:1908.02729*, 2019.
- [72] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.
- [73] Florian Tramèr and Dan Boneh. Adversarial training and robustness for multiple perturbations. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5858–5868, 2019.
- [74] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31. Curran Associates, Inc., 2018.

- [75] Naveed Akhtar, Jian Liu, and Ajmal Mian. Defense against universal adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.
- [76] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2574–2582, 2016.
- [77] Tejas Borkar, Felix Heide, and Lina Karam. Defending against universal attacks through selective feature regeneration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 709–719, 2020.
- [78] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 618–626, 2017.
- [79] Konda Mopuri, Utkarsh Ojha, Utsav Garg, and Venkatesh R. Babu. Nag: Network for adversary generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 742–751, 2018.
- [80] Wen Zhou, Xin Hou, Yongjun Chen, Mengyun Tang, Xiangqi Huang, Xiang Gan, and Yong Yang. Transferable adversarial perturbations. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 452–467, 2018.
- [81] Ares Lagae, Sylvain Lefebvre, Rob Cook, Tony DeRose, George Drettakis, David S Ebert, John P Lewis, Ken Perlin, and Matthias Zwicker. A survey of procedural noise functions. In *Computer Graphics Forum*, volume 29, pages 2579–2600. Wiley Online Library, 2010.
- [82] Ken Perlin. An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3):287–296, 1985.
- [83] Ken Perlin. Improving noise. In *ACM transactions on graphics (TOG)*, volume 21, pages 681–682. ACM, 2002.

- [84] Ken Perlin. Improved noise reference implementation. 2002. Accessed: 2018-06-09.
- [85] Ares Lagae, Sylvain Lefebvre, George Drettakis, and Philip Dutré. Procedural noise using sparse gabor convolution. *ACM Transactions on Graphics (TOG)*, 28(3):54, 2009.
- [86] Fabrice Neyret and Eric Heitz. *Understanding and controlling contrast oscillations in stochastic texture algorithms using spectrum of variance*. PhD thesis, LJK/Grenoble University-INRIA, 2016.
- [87] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.
- [88] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, volume 4, page 12, 2017.
- [89] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. *arXiv preprint arXiv:1811.12231*, 2018.
- [90] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3320–3328, 2014.
- [91] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2(11), 2017.
- [92] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1717–1724, 2014.

- [93] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2951–2959, 2012.
- [94] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando De Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.
- [95] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.
- [96] Eric Brochu, Vlad M. Cora, and Nando De Freitas. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *arXiv preprint arXiv:1012.2599*, 2010.
- [97] Dong C. Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.
- [98] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (AISec)*, pages 15–26. ACM, 2017.
- [99] Mitchell McIntire, Daniel Ratner, and Stefano Ermon. Sparse gaussian processes for bayesian optimization. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence, UAI’16*, pages 517–526, Arlington, Virginia, United States, 2016. AUAI Press.
- [100] Arjun Nitin Bhagoji, Warren He, Bo Li, and Dawn Song. Black-box attacks on deep neural networks via gradient estimation. In *ICLR Workshop*, 2018.
- [101] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248*, 2017.

- [102] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.
- [103] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7263–7271, 2017.
- [104] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.
- [105] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. Generating adversarial examples with adversarial networks. *arXiv preprint arXiv:1801.02610*, 2018.
- [106] Nicholas Carlini, Pratyush Mishra, Tavish Vaidya, Yuankai Zhang, Micah Sherr, Clay Shields, David Wagner, and Wenchao Zhou. Hidden voice commands. In *USENIX Security Symposium*, pages 513–530, 2016.
- [107] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 1–7. IEEE, 2018.
- [108] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.
- [109] Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, and Min Sun. Tactics of adversarial attack on deep reinforcement learning agents. *arXiv preprint arXiv:1703.06748*, 2017.
- [110] Wieland Brendel and Matthias Bethge. Approximating cnns with bag-of-local-features models works surprisingly well on imagenet. In *International Conference on Learning Representations (ICLR)*, 2019.

- [111] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.
- [112] Baifeng Shi, Dinghuai Zhang, Qi Dai, Zhanxing Zhu, Yadong Mu, and Jingdong Wang. Informative dropout for robust representation learning: A shape-bias perspective. In *International Conference on Machine Learning (ICML)*, pages 8828–8839. PMLR, 2020.
- [113] Kenneth T. Co, Luis Muñoz-González, and Emil C. Lupu. Sensitivity of deep convolutional networks to gabor noise. *arXiv preprint arXiv:1906.03455*, 2019.
- [114] Kevin Roth, Yannic Kilcher, and Thomas Hofmann. Adversarial training is a form of data-dependent operator norm regularization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
- [115] Daniel Jakubovitz and Raja Giryes. Improving dnn robustness to adversarial attacks using jacobian regularization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 514–529, 2018.
- [116] Roman Novak, Yasaman Bahri, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. In *International Conference on Learning Representations (ICLR)*, 2018.
- [117] Jure Sokolić, Raja Giryes, Guillermo Sapiro, and Miguel RD Rodrigues. Robust large margin deep neural networks. *IEEE Transactions on Signal Processing*, 65(16):4265–4280, 2017.
- [118] Dániel Varga, Adrián Csiszárík, and Zsolt Zombori. Gradient regularization improves accuracy of discriminative models. *arXiv preprint arXiv:1712.09936*, 2017.
- [119] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

- [120] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.
- [121] Ludmila I. Kuncheva. *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons, 2014.
- [122] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.
- [123] Kenneth T. Co, Luis Muñoz-González, Leslie Kanthan, Ben Glocker, and Emil C. Lupu. Universal adversarial robustness of texture and shape-biased models. In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 799–803. IEEE, 2021.
- [124] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.
- [125] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.
- [126] Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(771-780):1612, 1999.
- [127] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. *arXiv preprint arXiv:2002.08347*, 2020.
- [128] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.
- [129] Chaithanya Kumar Mummadi, Thomas Brox, and Jan Hendrik Metzen. Defending against universal perturbations with shared adversarial training. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 4928–4937, 2019.
- [130] Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

- [131] Daniel Kang, Yi Sun, Tom Brown, Dan Hendrycks, and Jacob Steinhardt. Transfer of adversarial robustness between perturbation types. *arXiv preprint arXiv:1905.01034*, 2019.
- [132] Sara Sabour, Yanshuai Cao, Fartash Faghri, and David J Fleet. Adversarial manipulation of deep representations. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.
- [133] Adblock Plus. Sentinel. 2020.