Deep Learning: Adversarial Robustness and Improved Generalisation Through Weight Variability

Thomas Arnaez-Selhi
Supervised by Michael Spratling
BSc Computer Science
Student ID: 1858820

April 29, 2022

Abstract

Deep hierarchical neural networks, such as convolutional neural networks (CNNs), have been shown to exhibit very strong performance in a wide variety of pattern recognition applications. However, the vulnerabilities of such methods to certain manipulations in the input data to produce what are known adversarial examples has been cited as a significant drawback. Additionally, such deep learning models are have not been able to demonstrate high robustness to commonly occurring, incidental corruptions to their input.

A recent paper has claimed that by introducing a weight variability feature, motivated by neuroscience and howt the neural variability the brain exhibits during responses to the same stimulus, the issues of overfitting and catastrophic forgetting that damage deep learning methods can been be solved at negligible costs. However, this paper did not address the susceptibility of their method to the aforementioned adversarial attacks, nor did it test its effectiveness against other input corruptions. This project will carry out that evaluation of adversarial robustness and generalisation.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary.

I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Acknowledgements

I would like to thank my supervisor, Michael Spratling, who has provided me with invaluable help and support throughout the project.

Thomas Arnaez-Selhi

8th April 2021

Contents

Chapter 1

Introduction

In the last five to ten years, machine learning has become pervasive with their impact affecting every industry, from social media feed suggestions to the detection of cancer cells. With these techniques becoming a critical part of so many key systems and industries, research into the reliability, security and trustworthiness of such machine learning algorithms are of significant concern.

One such concern are adversarial attacks, a malicious technique that attempts to to fool machine learning systems by either providing a model with faulty or mis-representative data during the training phase, or by inputting manipulated data to a trained model such that network produces incorrect results. Such attacks have shown to expose a severe lack of robustness in modern deep learning.

Alongside the intentional adversity caused by bad actors utilising the attacks mentioned, there also exists the issue of incidental adversity whereby input data becomes corrupted. In the field of computer vision it is often the case that images are non-maliciously altered by weather effects, various forms of blurring and noise (usually as a result of the image being compressed), all of which have been shown to reveal poor general robustness in computer vision networks that otherwise show exceptional results on clean test data taken from the same distribution that the data the networks was trained on. For this project we denote such adversity as *input corruption*.

For this project we evaluate a new machine learning that proposes that introducing a form of noise variability into the weights of its during the training phase can reduce overfitting and improved its generalisation to unseen data. However it remains unknown how effective this technique is in defending the techniques stated above. The goal of this project is then to test the ability if the proposed algorithm to defend against certain malicious techniques one might use to fool a network, these include *adversarial attacks* and *input corruptions*.

Chapter 2

Background

The suggestion put forward in the paper behind algorithm we will evaluate in this project is that deep learning is held back my two major issues; overfitting and catastrophic forgetting. It points towards the evidence that such problems are rarely occurring in the natural neural systems i.e. the brain, which today's neural networks are already heavily inspired by and proposes to replicate what it refers to as *neural variability* whereby a system shows considerable variation to the same inputs. With this newly implemented feature that introduces variability into the weightings of a neural network, known as *artificial neural variability*, the authors claim it offers significant improvements in terms of generalisation and reduced overfitting.

For this project we look into different concepts of robustness and adversarial issues to help evaluate this proposed algorithm.

An adversarial attack is a malicious attempt to manipulate data to produce wrong outputs when entered into a machine learning model. If we consider a data point x_0 which belongs to class C_i, the attack attempts to change x_0 as slightly as possible so that model misclassifies as belonging to a different class. The concept of adversarial examples can be formulated mathematically by:

$$\text{minimise } ||x - x'|| \text{ such that } C(x) \neq C(x') \tag{2.1}$$

Where x is a data input occurring in a dataset, x' is an adversarial example and C is a classifier's prediction for a given input. Some adversarial attacks are what are known as *targeted attackers* where the attacker aims to produce adversarial examples to be are predicted as a specific class, and conversely there are *non-targeted attacks* that only have the goal of causing the target classifier to misclassify the adversarial image.

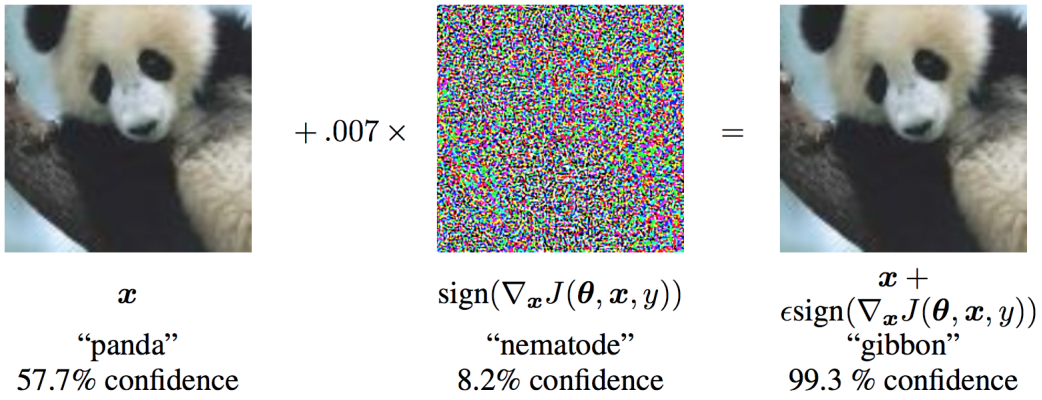

Figure 2.1: Examples of FGSM attack performed on a panda image to change its predicted label. Credit: PyTorch website

Adversarial attacks can be separated into two categories; white box and black box attacks. In a white box attack, the attacker has full knowledge about the machine learning algorithm it is attempting to fool, such as its parameters, hyper parameters and network architecture and is therefore able to generate adversarial examples based on this properties. Conversely, a black box attack has no knowledge of the intrinsic properties of its target algorithm available to it. Expectedly, white-box attacks have been shown to be considerably out perform black box attacks because of their ability to learn from the target model. Most highly effective white-box attacks employ gradient-based methods to construct adversarial examples. Such an example is the Fast Gradient Sign Method (FGSM), where adversarial examples are found by altering the image with a small perturbation in the direction of its loss function with respect to the input image. The mathematical formula behind it is as follows:

$$x_{adv} = x + \epsilon * sign(\delta_x L(x, y)) \tag{2.2}$$

Where $L(x, y)$ represents the loss function with respect to x. The FGSM attack is one of the algorithms we make use of, along with the Projected Gradient Descent algorithm (PGD) which can be seen as applying the FGSM algorithm iteratively until our knowledge of the classifier it will be misclassified. Chapter 6 is where we focus on applying such adversarial attacks.

Adversarial transferability is another security concern to neural networks that has exposed disappointing robustness performance, where adversarial attacks targeting one model have been shown to be successful against different, and possibly unseen models. With many of the popular used deep learning models being made publicly available, it is possible that even without complete access to a model's parameters to train against one can get relatively close by generating adversarial examples on a model with the same or highly similar architecture.

We explore the issue of adversarial transferability on the proposed method in section 6.4.

As for methods to improve robustness against malicious actors, adversarial training has been shown to be one of the most effective methods deep learning methods to defend against adversarial attacks [1]. The idea is to simply construct a set of adversarial examples and incorporate them into the training phase of the model, and the deep learning network learns to not be fooled by the small perturbations of such attacks. Although very effective, it can be highly resource consuming when working on very large datasets such as ImageNet. In 6.3 we evaluate the gains of the proposed variability algorithm from adversarial training.

Whilst most of the work done to improve robust performance on deep learning methods is done with adversarial examples in mind, other issues found in the performance of neural networks is how well they respond to naturally occurring "corruptions" found in input data. In the terms of computer vision this could be the effects of weather, blur, noise, compression and changes in image quality. Such phenomena are not adversarial attacks intentionally crafted to fool neural networks, but are very often found in real-life examples. It has been shown that models still considerably underperform humans when it comes to evaluating images that contain small, incidental perturbations and naturally occurring image corruptions. To that end, several datasets containing commonly found corruptions have been constructed to aid in benchmarking out-of-distribution robustness, such as the MNIST-C and CIFAR-10-C datasets which we make use of in Chapters 4 and 5.

Note that in much of the analysis ahead we refer to the proposed algorithm as *VSGD*, which is the object name for the optimizer provided by the paper.

Chapter 3

Methodology

First and foremost, for any machine learning we identify the datasets to be used. The MNIST[3] database is consists of 80,000 black and white images showing handwritten digits, with each image paired with a label between 0 and 9 corresponding the digit shown in the image. It is a highly convenient database for implementing machine learning techniques due its simplistic nature, requiring only the minimum of time spent and preprocessing inputs before proceeding to training and testing. The MNIST-C [6] is a complimentary dataset consisting of a set of 15 corruptions applied to the MNIST test dataset, an example of the aforementioned input corruptions which will allow for benchmarking the out-of-distribution robustness of the model.

We also make use of the CIFAR dataset, which consists of 60,000 colour images of 32 by 32 pixels showing images of various objects. To further evaluate robustness to input corruption and perturbation with the CIFAR-10-C and CIFAR-10-P datasets presented by the Hendyrcks paper [4].

For developing the algorithm the Jupyter Notebook web framework was used, due to its convenience for working with data owing to its live code editing and quick visualisation features. The .ipynb Jupyter Notebook files are provided in the source code attachments.

The PyTorch machine learning framework was the logical choice to carry out this project with, since the authors of the paper of the proposed method have provided a PyTorch implementation of their algorithm [7]. The PyTorch API allows for accessing commonly used ML training sets MNIST and CIFAR. To perform adversarial attacks, PyTorch library Torchattacks [5] to produce adversarial examples for many adversarial attacks and will be used to further verify the robustness of the proposed algorithm. PyTorch allows for the saving of model parameters to file, and so the model weights used to produce the results in this report will be provided.

To further evaluate the robustness of the algorithm when against corrupted inputs, such as the MNIST-C and the CIFAR-10-C, we utilise the metrics provided by the Hendyrcks paper [4]: Corruption Error (CE), and relative Corruption Error (rCE). Averaging over these errors for the set of corruptions gives us the mCE and relative mCE.

$$CE^f = E_c^f / E_c^{Base} \tag{3.1}$$

Corruption Error gives us the error relative to a baseline model, in this project we will use a simple CNN trained with the SGD optimizer.

$$\text{relative } CE^f = (\sum_{s=1}^{5} E_{s,c}^f - E_{clean}^f)/(\sum_{s=1}^{5} E_{s,c}^{Base} - E_{clean}^{base}) \tag{3.2}$$

Relative corruption error is a more complex metric that is designed to measure how much a classifier's performance declines on a corrupted dataset relative compared to a clean dataset. The code to implement these metrics can be found in the "py/ce.py" file.

To produce the charts to evaluate the algorithm in this report, we use the Matplotlib and pandas Python packages to process and plot the training and test results of the algorithms.

Chapter 4

MNIST-C: A Robustness Benchmark For Computer Vision

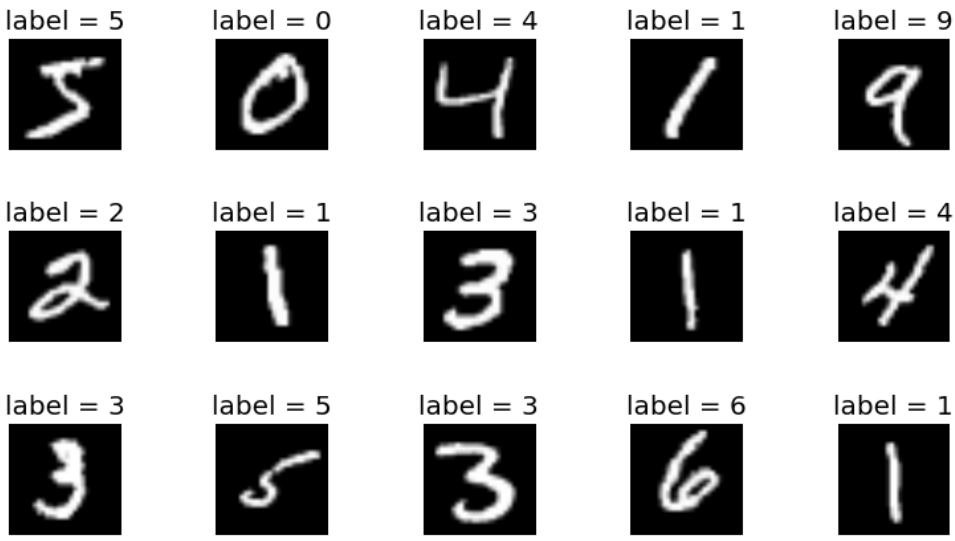

Figure 4.1: Examples of MNIST dataset images and labels

4.1 Model and Training

The MNIST dataset is very convenient for training machine learning techniques due to the minimal amount of time needed to be spent on preprocessing and formatting the data, particularly with the PyTorch API providing easy access to the dataset. Its short training time makes it very convenient for a student project.

Both the base model and the VSGD models were train for over 3 epochs, with a learning rate of 0.003 and momentum value of 0.9. The variability parameter of the VSGD algorithm was varied to produce different models to test its effective on the adversarial robustness.

For training the base model and initial VSGD model with a variability of 0.01, from Figure 4.1 we can see that both algorithms settle at a similar training loss value, but the base SGD algorithm takes longer to reach this point, possibly due to introduction of weight variability has allowed to algorithm to generalise and learn more from fewer examples. The SGD algorithm achieved an accurate of 96.88% on the test set, and the VSGD achieved 98.52%.

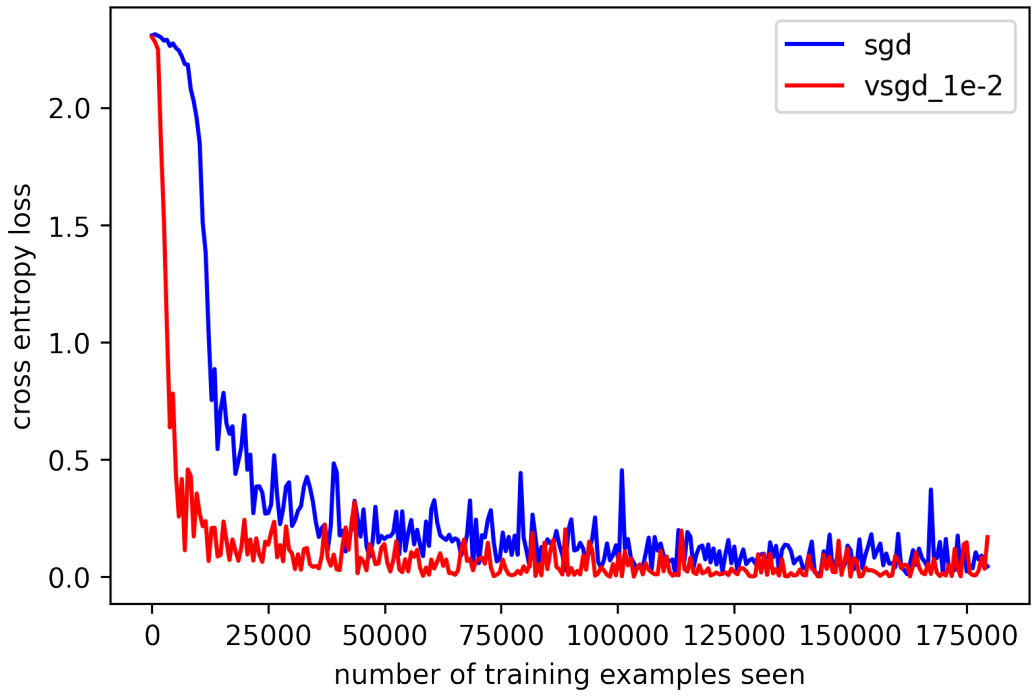

Figure 4.2: Comparing the training loss of the base and VSGD model

Table 4.1: Model Accuracy on individual classes on the MNIST test set (%)

Network	0	1	2	3	4	5	6	7	8	9
SGD	99	98	99	98	98	96	98	98	96	95
VSGD 0.01	100	99	97	99	98	96	98	98	100	95

4.2 Testing Input Corruption Robustness

We make use of the MNIST-C dataset, consisting of 15 commonly occurring visual corruptions to evaluate general out-of-distribution robustness performance. These are generated corruptions for the MNIST dataset that aim to be "semantically invariant" [4], meaning that a human won't be fooled by the corruption), and are designed to be plausible in the real world and contain little overlap.

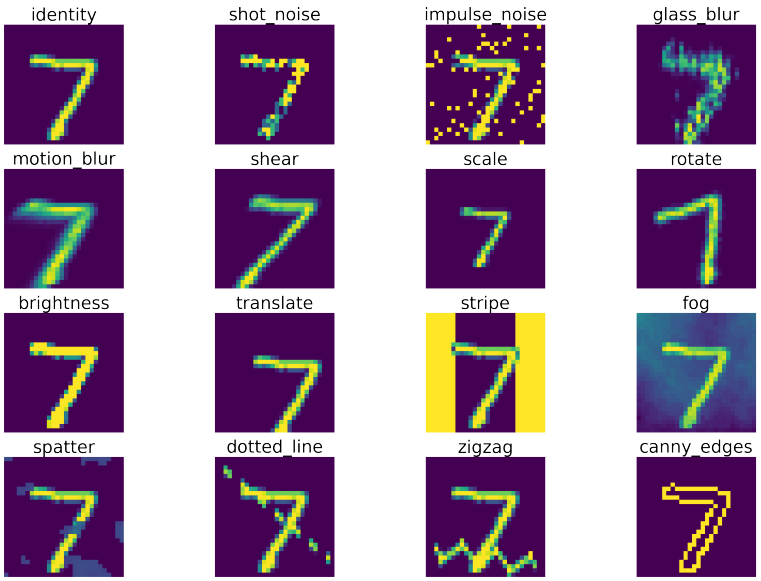

Figure 4.3: An example off all 15 corruptions of the MNIST-C dataset applied to image with label '7'

We can observe from Figure 4.2 that the VSGD algorithm with a variability of 0.01 showed significantly lower test error across the board when compared to the base model on the corruptions of the MNIST-C, with huge reductions in error rate found on brightness and fog weather effects in particular.

The relative corruption errors (calculated by equation 3.2) are shown in Figure [**?**]. These results would indicate that the vsgd model is considerably robust to input corruptions than the base model.

Using the corruption error equation outlined in 3.1 we calculate the corruption errors on model for each corruption, which are displayed in Table 4.2.

The data then appears to indicate that for a large variability of 0.1 results in huge decreases in accuracy on the corruption set, but the values ¡ 0.1 all showed lower mean relative corruption errors implying considerably robustness improvements.

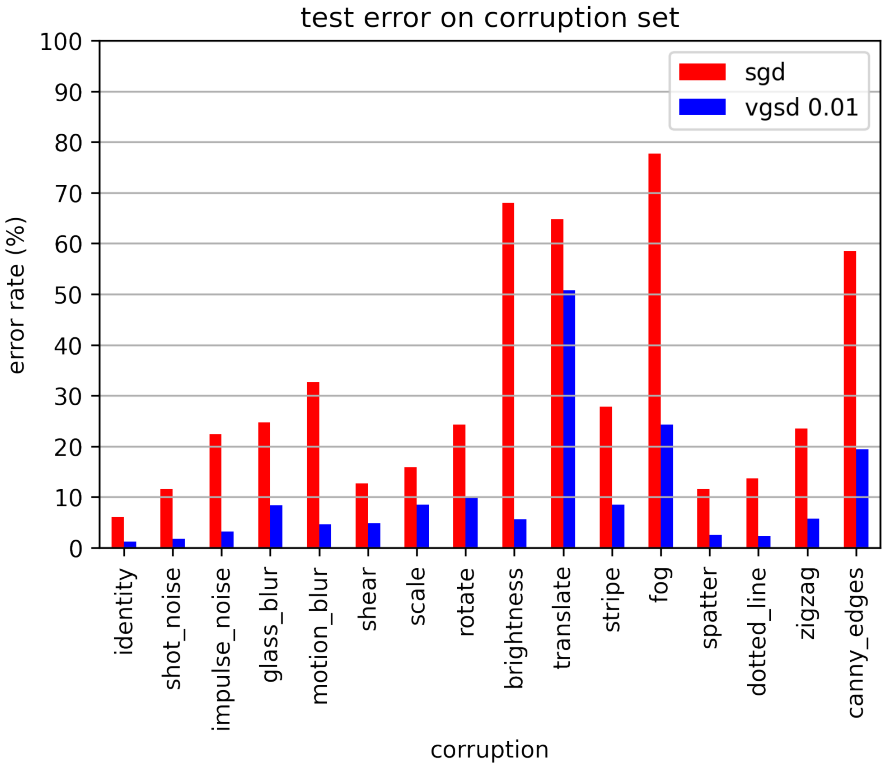

Figure 4.4: Comparing the error rates of the base algorithm and VSGD algorithm for each MNIST-C corruption

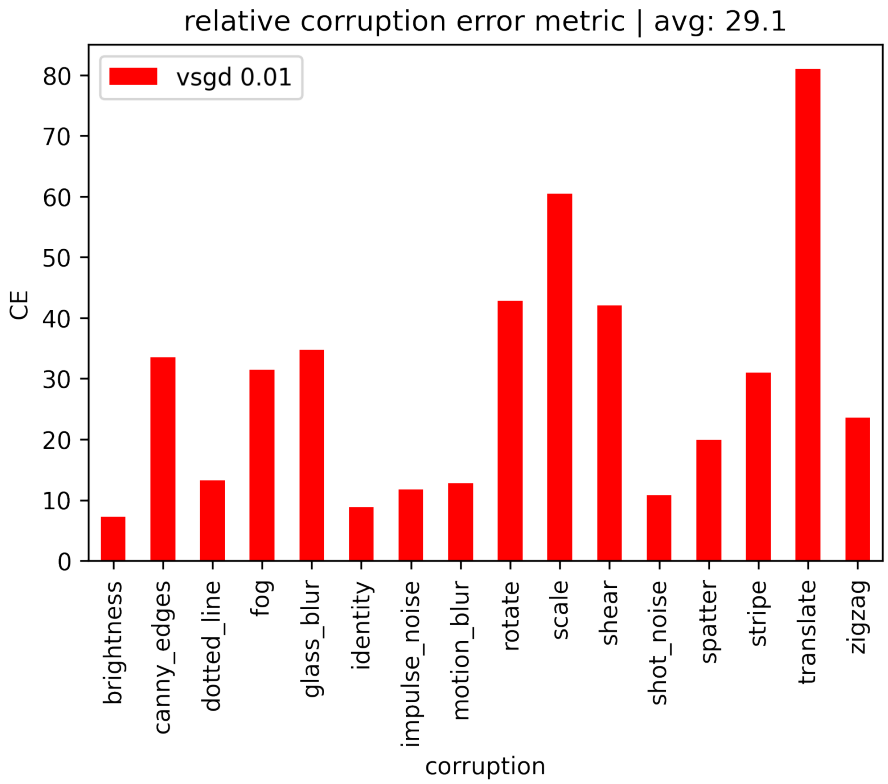

Figure 4.5: Relative corruption error metric for the VSGD algorithm

Table 4.2: Relative Corruption Error data for model on the corruptions of the MNIST-C dataset. Each model is trained on the clean MNIST dataset.

Model	mRCE	Bright	Canny	Dotted	Fog	Glass	Ident	Impulse	Motion	Rotate	Scale	Shear	Shot	Spat	Stripe	Trans	Zigzag
VSGM 0.1	628	2961.3	999.8	432.5	386.6	281.8	876.3	657.6	395.1	128.3	134.8	337.8	111.5	990.2	798.7	409.7	150.3
VGSM 0.001	28.7	14.6	17.0	18.1	27.1	17.9	27.5	40.1	44.1	8.5	79.2	31.6	18.5	25.7	14.0	32.8	43.7
VGSM 0.02	34.4	12.3	29.3	20.3	35.4	28.4	25.3	62.1	36.3	1.8	80.2	34.8	18.5	33.7	18.9	45.8	68.0
VSGSM 0.05	39.2	19.5	21.0	26.1	46.9	28.3	41.9	39.5	42.8	21.2	84.5	36.0	48.8	26.8	31.0	52.7	59.5
VGSM 0.01	29.2	15.5	12.7	10.9	20.0	8.7	37.0	48.7	38.6	5.6	72.5	62.6	14.0	25.1	13.9	33.5	48.5

Chapter 5

CIFAR-10

The CIFAR-10 dataset is a collection of images regularly used to train computer vision and machine learning algorithms. It contains 60,000 colour images. Each image belongs to one of 10 different classes; airplanes, cats, birds, deer, dogs, frogs, horses, ships and trucks. The low resolution of 32 by 32 pixels makes it a very convenient dataset for quick development due to the low training time of algorithms.

We will be using it further robustness against input corruptions and perturbations. The Hendyricks paper [4] provides two datasets, the CIFAR-10-C and the CIFAR-10-P. Note when testing that these datasets are heavy in disk space, taking up 2.9 GB and 18.3 GB respectively.

Figure 5.1: Example of each class from the CIFAR-10 dataset

5.1 Model and Training

Moving from the greyscale MNIST dataset to the colour CIFAR dataset requires a different convolutional neural network to handle its three channel image data. The first architecture of the first model tested was outlined by the official PyTorch documentation and in initial tests achieved a fairly low accuracy of 62%. After preparing a more complex CNN, we achieved an accuracy of 72%.

To help improve our prediction accuracy, for training on the CIFAR dataset we include a validation set to periodically measure validation loss on during the testing stage. As we can observe from Figure 5.2 the validation loss indicates that training the model for over 6 epochs leads to worse performance on the validation set, suggesting the model has begun to overfitting. Table 5.1 shows the final testing accuracy achieved for each mode.

Due to the longer training time on the CIFAR dataset, the model weights for the CIFAR models have been produced with the source code and the Jupyter Notebook to visualise the results is separated from the Notebook used to train the models.

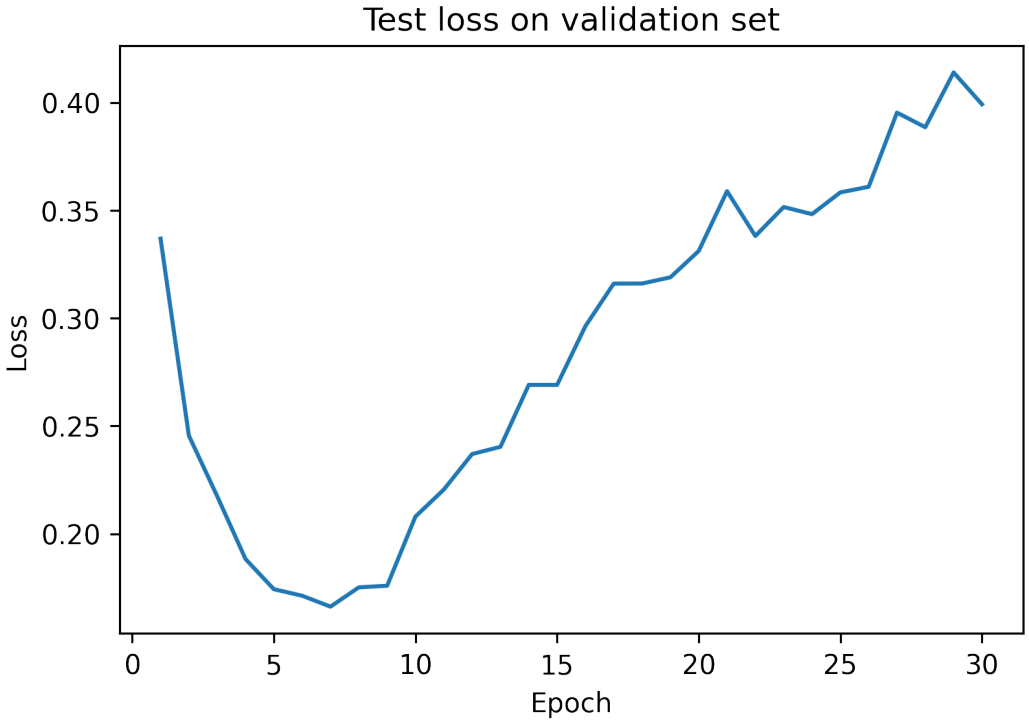

Figure 5.2: Validation loss during training for the CIFAR10

Table 5.1: Test accuracies on the clean CIFAR-10 dataset for SGD and VSGD with different variabilities

Model	SGD	VGSM 1e-1	VGSM 1e-2	VGSM 1.5e-3	VGSM 2e-3	VGSM 5e-3
Test Acc (%)	72.41	21.47	71.84	70.86	71.52	71.72

5.2 CIFAR-10-C Robustness Benchmark

The CIFAR-10-C dataset consists of 15 algorithmically generated corruptions applied to to the test images of the CIFAR-10 dataset at 5 different levels of severity, as observed in Figure 5.3.

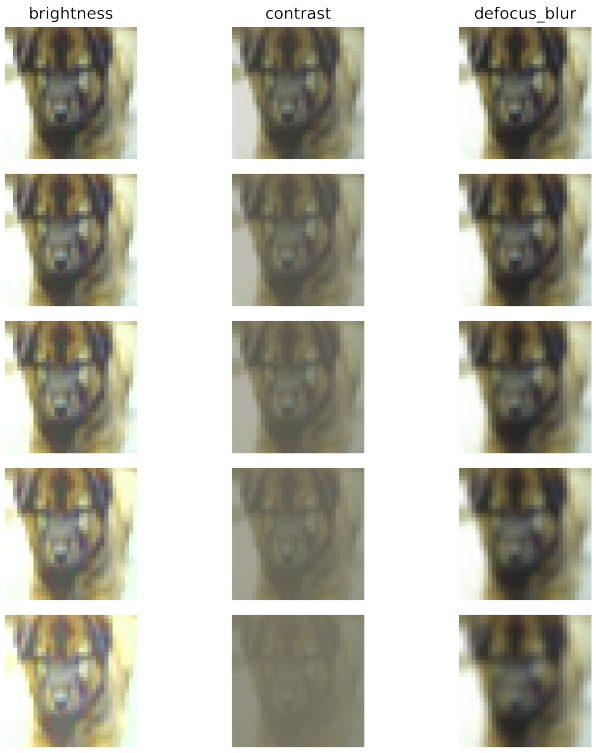

Figure 5.3: Example of three different corruptions applied to dog image at increasing severity

Figure 5.2 shows the corruption errors on the different datasets to be similar, which is notably different from the significant improvements that were found when testing on on the MNIST-C corruption dataset.

Using the equations outlined in 3.2 we calculate the relative corruption error metrics for the proposed variability algorithm at different variabilities, the results are displayed in Table 5.2. This does reveal the proposed VGSM algorithm for values of variability below 0.1 generally shows improves robustness to input corruptions. Noting that a variability of 0.1 seems to produce negative error rates, this is most likely a statistical quirk due to the algorithm performing better on the corruption set than the clean set on some tests due it predicting labels completely randomly.

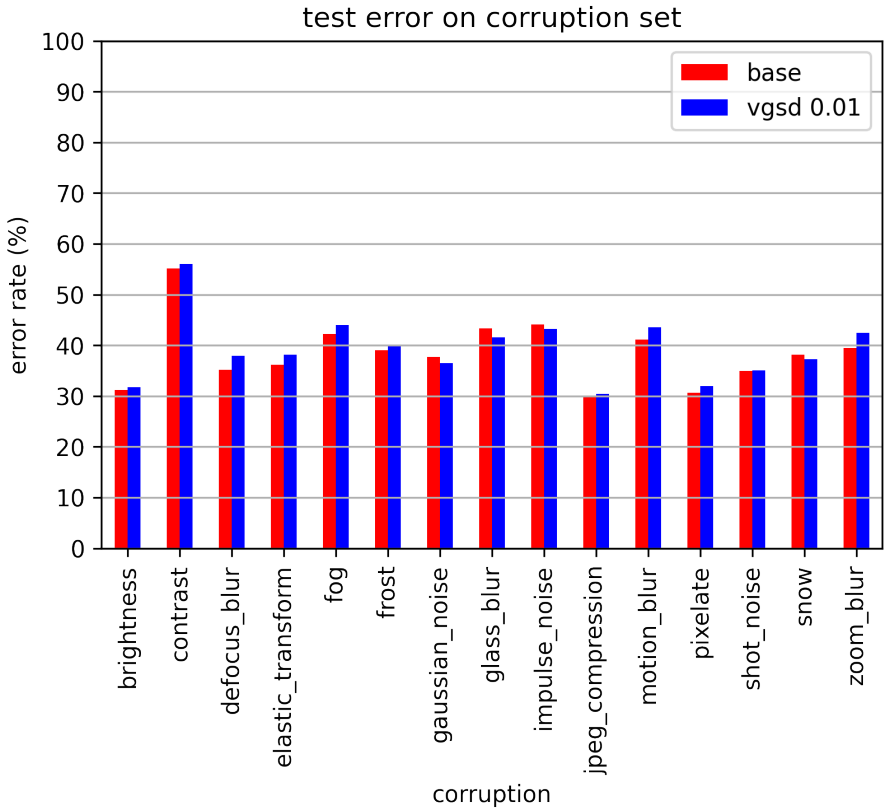

Figure 5.4: Comparing the error rates of the base algorithm and VSGD algorithm for each CIFAR10-C corruption

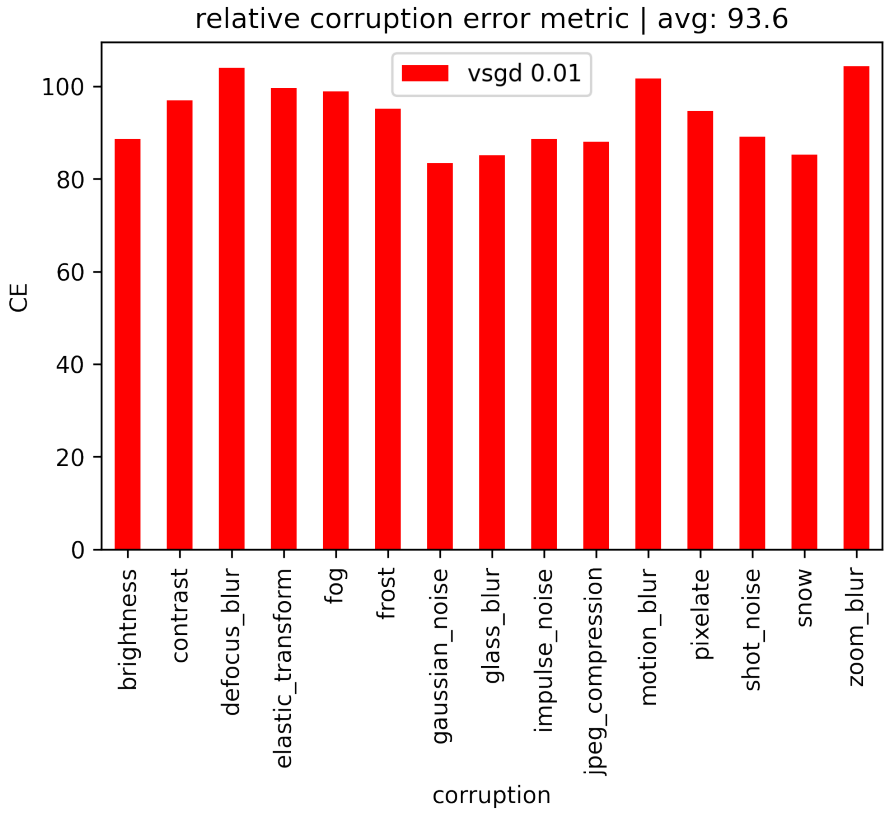

Figure 5.5: Comparing the error rates of the base algorithm and VSGD algorithm for each MNIST-C corruption

Table 5.2: Relative Corruption Error data for model on the corruptions from the CIFAR-10-C dataset, relative to a base model trained using SGD. Each model is trained on the clean CIFAR10 training set.

Model	mRCE	bright	contrast	defocus	elastic	fog	frost	gaussian	glass	impulse	jpeg	motion	pixel	shot	snow	zoom
VSGM 0.1	0.996	-0.1	5.3	0.8	3.0	4.1	5.7	-2.3	0.0	-0.4	0.0	1.8	-3.1	-1.7	-0.5	2.3
VGSM 0.001	72.	66.5	76.4	80.5	82.3	78.1	76.5	61.4	67.1	65.6	64.8	80.7	72.4	64.1	67.9	83.8
VGSM 0.02	80.6	71.1	75.4	84.9	88.8	78.2	73.8	81.6	81.6	83.7	78.1	85.2	85.9	83.0	74.0	85.0
VGSM 0.05	63.6	66.2	72.2	75.7	77.1	68.1	68.7	44.8	58.8	51.5	55.3	75.3	60.1	44.6	56.4	79.7
VGSM 0.01	66.9	66.5	75.3	69.2	69.8	70.6	72.3	62.4	64.8	65.6	55.0	75.1	55.8	63.9	63.5	74.7

5.3 CIFAR10-P Robustness Benchmark

The second benchmark we will use for the CIFAR-10 dataset is input perturbation via the CIFAR10-P, also provided by Hendyricks paper on robustness [4].

This benchmark is designed to test a model's perturbation robustness. Perturbations involving a small modification to the input data of a classifier that fool it into predicting the wrong output, whilst attempting to remain imperceptible to human inspection. The CIFAR10-P does not contain labels and is only a benchmark to measure the stability of a classifier's predictions when given noisy data.

Each perturbation is applied to an image to produce a sequence of over 30 frames. With exception of the noise perturbations which are repeatedly applied to the original image at greater severity, the rest of the perturbations are applied temporally meaning that the each frame is a result of the perturbation applied to the previous frame.

The Jupyter Notebook used to run evaluations on the CIFAR-10-P is found as "cifar10p.ipynb".

To evaluate performance on the CIFAR10-P we can utilise on the Flip Probability (FP) metric, which measures how often the predicted label changes in response to a perturbation. To measure the Flip Probability of classifier f on perturbation sequence p we can use the following equations, also provided by the Hendyricks paper [4]:

$$FP_p^f = \frac{1}{m(n-1)} \sum_{i=1}^{m} \sum_{j=2}^{n} (f(x_j^{(i)}) \neq f(x_{j-1}^i)) \tag{5.1}$$

For noise perturbations which are not temporally applied, we instead compare each result with the original, unaltered image found at index 1:

$$FP_p^f = \frac{1}{m(n-1)} \sum_{i=1}^{m} \sum_{j=2}^{n} (f(x_j^{(i)}) \neq f(x_1^i)) \tag{5.2}$$

In terms of implementing this in Python, we need to understand the dimensions of dataset being inputted to the model. PyTorch deals in Tensors as their central data construct, which are multi-dimensional matrices holding elements of simple data such as ints or floats. As we iterate over the DataLoader, we receive a Tensor of shape [x, y, 3, 32, 32]. x represents the number of images in a batch, and y corresponds to the number of frames making up the perturbation sequence. To predict labels we need to squeeze these two dimensions to have the right shaped Tensor to be inputted into our model, after which we "unsqueeze" the Tensor and for each image x, iterate over each perturbation sequence y to count the number of labels that have been flipped. The PyTorch function view is very useful in this case, as it allows us to do fast reshaping and slicing whilst maintaining the underlying shape of the data.

Table 5.3 and Figure 5.6 demonstrate how the proposed variability algorithm was able to reduce the likelihood of the predicted label flipping compared to the base model for all variabilities, which provides further evidence to a general trend improvement to input corruption.

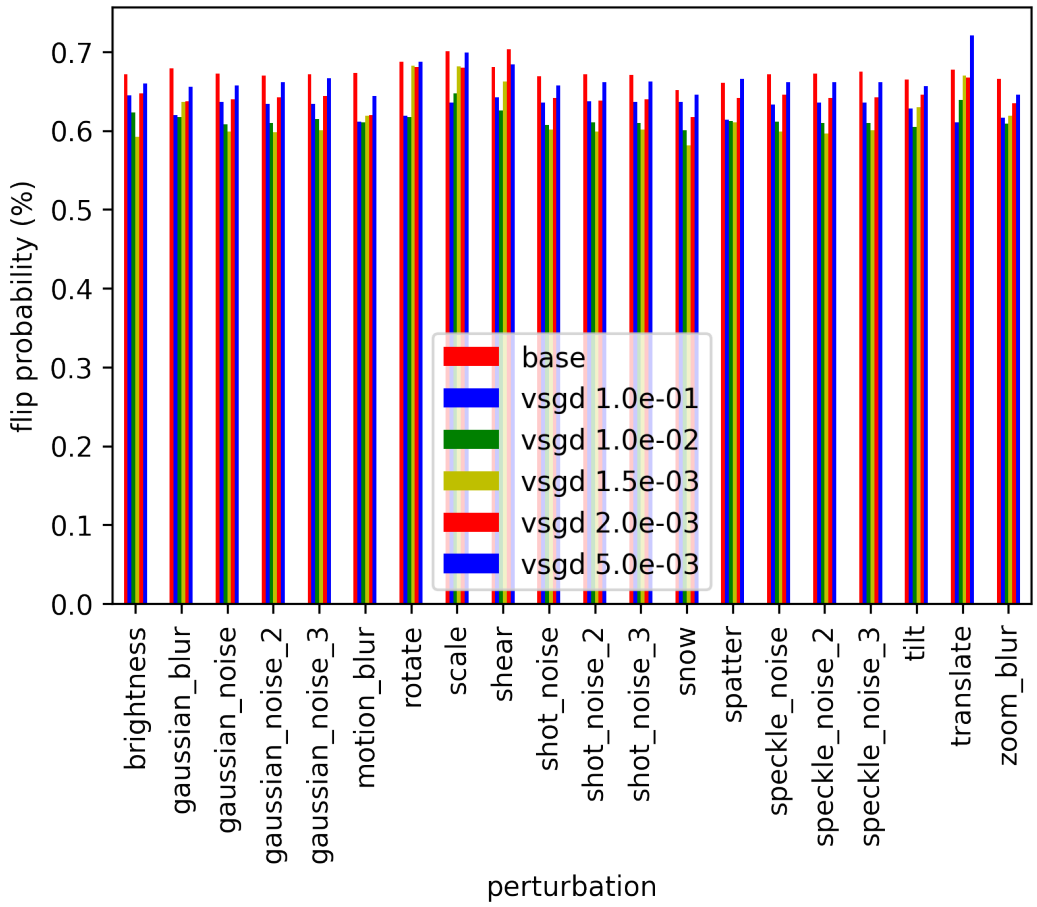

Figure 5.6:

Table 5.3: Mean flip rates for networks

SGD	VGSM (0.1)	VGSM (0.01)	VGSM (0.002)	VGSM (0.015)	VGSM (0.005)
0.673	0.630	0.615	0.619	0.647	0.666

Chapter 6

Attacks

Using the package Torchattacks [5], which is well integrated with the PyTorch framework, we make use of a variety of adversarial attacks in this section. Unlike the previous experiments which were based on non-malicious techniques, here we test the proposed variability algorithm's adversarial robustness.

6.1 Setup

Trial and error was needed to find parameters for attack methods on the base model that were able to produce effective adversarial examples that fooled the algorithm whilst remaining semantically invariant. In some cases attacks would wipe all non-0 values of the Tensor leaving just a black image. The parameters used to generate attacks used in these experiments can be found in the appendix at A.

Testing on an example image from the MNIST dataset, we can observe the result of applying these different attacks shown in Figure 6.1. We can still clearly see the images still resemble the original label, with the FGSM and PGD adding a lot of minor "noise" whilst the CW attack appears completely imperceptible.

Noting that the Torchattacks documentation suggests to examples should be scaled from [0, 1] we don't apply the normalization transformation applied in the previous sections, which from testing appears to reduce accuracy on the test set.

Figure 6.1: Adversarial attacks applied to MNIST image with label '5'

6.2 Robustness to Adversarial Attacks

Through the Torchattacks API [5] we can generate a set of augmented images for a given dataset for a wide range of attacks. From there we can test how a model trained on a clean MNIST dataset holds up when tested on these attacked images. The set of attacks we make use of to assess the robustness are the Fast Gradient Signed Method (FGSM), Carlini & Wagner (CW) [2] and the Projected Gradient Descent (PGD).

The Jupyter notebook to run the code used to carry out this evaluation can be found under "mnist_attacks.ipynb". Note that generating adversarial attacks for the Carlini Wagner attack is time consuming and resource intensive.

Table 6.1: Standard and Robustness Accuracy to Various Attacks (%)

Network	Clean	FGSM	PGD	CW
SGD	98.97	6.23	0.01	50.51
VGSM 0.001	98.89	5.41	0.17	56.47
VGSM 0.01	98.47	9.03	0.15	57.21
VGSM 0.03	98.3	13.32	0.42	54.56
VGSM 0.005	96.09	16.43	0.64	52.24
VGSM 0.1	11.35	10.04	10.09	10.09

The test accuracy results shown in 6.1 clearly showcase that Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD) adversarial attacks are highly effective at fooling both the base classifier and the proposed variability algorithm to near around 10% after both algorithms having achieved very high accuracy on the clean test set, whilst the Carlini & Wagner only reduces effective to around 50%.

The VSGD algorithm is demonstrably highly susceptible to adversarial attacks, showing significant decreases in accuracy under all attacks. When compared to the base SGD algorithm, it does exhibit a marginally higher accuracy to

all attacks for the variability parameters 0.01, 0.03 and 0.005. At the highest variability of 0.1 the algorithm appears to degrade into making random guesses, achieving approximately 10% on each test set whilst we know there are 10 classes to choose from. The results may suggest that weight variability may marginally improve adversarial robustness whilst bringing achieving slightly worse performance on the clean dataset.

6.3 Adversarial Training

Adversarial training is a machine learning technique whereby we generate adversarial examples from training data using some prior knowledge of adversarial attacks and feed them into our model during the training phase. This very simple method has been shown to be one of the most effective ways of improving an algorithm's adversarial robustness, possibly due to the fact that many attackers are using similar techniques to produce their adversarial examples which creates a high level of transferability.

In this case we tested the improvements of the proposed algorithm through adversarial training on the MNIST dataset using the Fast Gradient Sign Metho, Projected Gradient Descent and Carlini & Wagner adversarial attacks, and compare its results to that of our base model. Note that training on examples produced by the Carlini & Wagner algorithm attack has a significantly long training time.

Table 6.2: Test accuracy on datasets after adversarial training using the FGSM attack

Network (var)	Clean	FGSM	CW	PGD
SGD	94.96	97.89	86.15	97.87
VSGD (0.001)	99.07	98.51	96.70	98.49
VSGD (0.01)	99.05	98.53	96.90	98.49
VSGD (0.1)	98.94	98.33	96.59	98.32

Table 6.3: Test accuracy on MNIST dataset after adversarial training using the PGD attack

Network (var)	Clean	FGSM	CW	PGD
SGD	94.96	93.86	90.00	93.79
VSGD (0.001)	88.04	90.64	87.92	90.58
VSGD (0.01)	92.76	94.67	92.57	94.57
VSGD (0.1)	51.12	48.54	50.99	48.39

Table 6.4: Test accuracy on MNIST dataset after adversarial training using the CW attack

Network (var)	Clean	FGSM	PGD	CW
SGD	99.15	6.23	0.01	50.51
VSGD (0.001)	98.98	9.03	0.17	56.47
VSGD (0.01)	0	90.64	0.15	90.58
VSGD (0.1)	0	94.67	92.57	94.57

From the results shown in Tables 6.2 we see that when training on adversarial examples produced by the FGSM attack, the VSGD model showed improved robustness compared to the SGD for all test datasets. The data on the adversarial training with the PGD attack in Table 6.3 shows the VSGD model accuracy on the test data is much more sensitive to the variability parameter of the of the optimizer algorithm, but a value of 0.01 showed more robustness overall compared to the base model.

6.4 Transfer Attack

Adversarial attacks against a deep learning model have been shown to demonstrate transferability against a different and potentially unseen networks. For this section we will use an outsider deep learning model to generate a set of adversarial examples, and feed this augmented inputs into VSGD model to evaluate its robustness to transfer attacks. Such an attack is an example of a black box attack as we are not using any intrinsic knowledge of the targeted model.

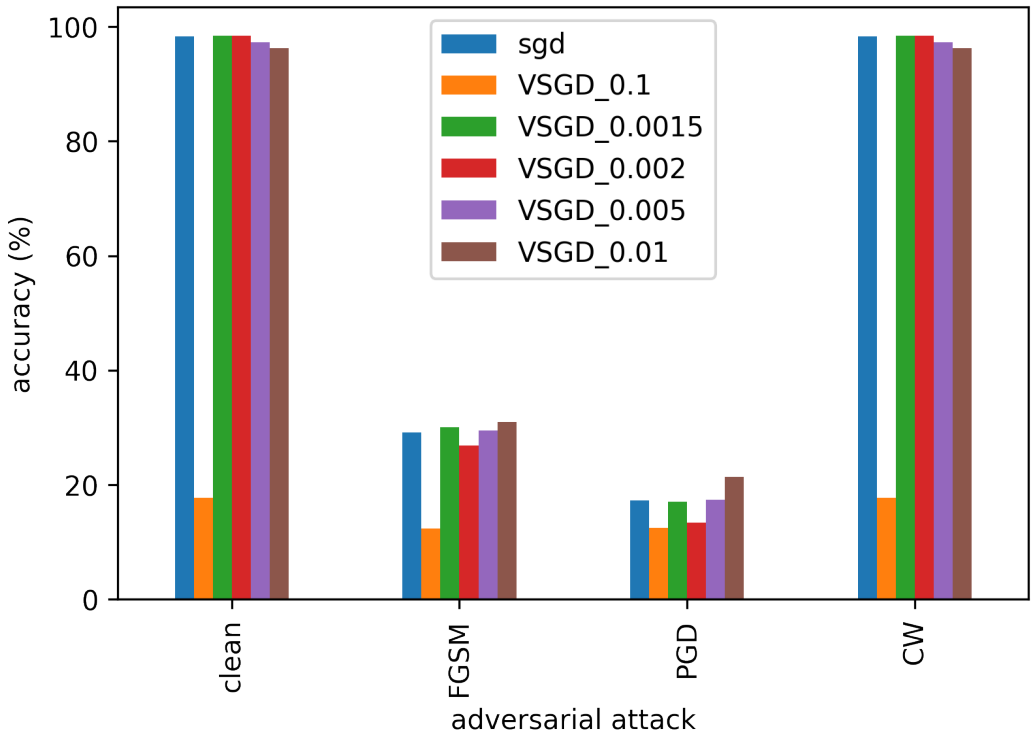

Figure 6.2: Transfer attacks performed with the MNIST dataset

The results in Figure 6.2 show broadly that setting a higher variability value of 0.1 causes low accuracy across all attacks and the clean dataset. The CW attack was also shown to be ineffective at producing dangerous adversarial examples in this instance. Judging from the effective attacks of the FGSM and PGD, we we observe that the middling variability values of 0.01 and 0.005 result in improved robustness after adversarial training compared to the base model. This would suggest that incorporating weight variability into algorithm can allow it to better learn from adversarial training.

Table 6.5: Test accuracy on datasets after performing adversarial training

Network (var)	Clean	FGSM	PGD	CW
SGD	98.27	29.12	17.31	98.27
VSGD 0.0015	98.45	30.09	17.03	98.45
VSGD 0.002	98.43	26.85	13.39	98.43
VSGD 0.005	97.24	94.67	17.41	97.24
VSGD 0.01	96.28	30.93	21.44	96.28
VSGD 0.1	17.68	12.38	12.44	17.68

Chapter 7

Awareness of Professional Issues

Professional issues in the machine learning field revolve less about the algorithms themselves, but more-so in how the data is collected and used. Recent events such as the Facebook scandal surrounding Cambridge Analytica highlights such issues the way user data has been stored and used to target certain demographics without the user's awareness or consent. While this project only used datasets that have been made publicly available are intended for academic use and has no security implications, the issue of data protection when working on machine learning projects is something that should always be kept in mind.

Chapter 8

Conclusion

In conclusion, there appears to be some evidence that point towards the proposed *artificial neural variability* method improving generalisation with its variable stochastic gradient descent method (VSGD) optimizer algorithm. We observed significant improves on input corruptions for both the MNIST-C dataset, and marginal improvements on the CIFAR-10 dataset. On evaluating robustness to adversarial attacks, in Section 6.2 we also found that at certain variability parameters exhibits improved robustness compared to a base model trained on stochastic gradient descent. However, the proposed variability model was still highly susceptible to these adversarial attacks and exhibited a huge reduction in accuracy, decreasing from over 98% to below 10% in some cases. In section 6.3 we observe that adversarial training did seem to be more effective when learning from the FGSM algorithm, the differences were very minor. In section 6.4 on transfer attacks we also observe accuracy values that are similar and in some cases marginal improvements by the VSGD algorithm compared to the base when defending against transfer attacks from the FGSM and PGD attack. It definitely seems to the case that introducing weight variability improves input corruption robustness more than adversarial robustness.

This analysis was quite limited in terms of the number of tests ran and tuning of variables to enhance performance so it in all likelihood it cannot be used to arrive at strong conclusions about the performance of the algorithm. For example, during the testing of adversarial attacks it was readily apparent the tuning of the number of steps, epsilon value and constants used by the attacker algorithm produced significant changes in the Carlini and Wagner attack's effectiveness, and multiple attacks that were tested were not used in the final report as they didn't appear to generate effect adversarial examples. This would need to be explored much further before fully concluding the algorithm's robustness to adversarial attacks.

As to how this project could be taken further, testing with a wider range of base models to compare with the proposed neural variability algorithm would

provide a better overview of how its robustness performance compares to some of the modern machine learning methods. For computer vision tasks this could be AlexNet, ResNet and VGG models which are very popular models for difficult object recognition problems. More challenging training data could be used, as in the interest of time and computer resources this analysis focused on relatively trivial object recognition datasets MNIST and CIFAR-10 of low resolution. ImageNet and ObjectNet are examples of more challenging datasets. Furthermore, the domain could be extended to tasks other than object recognition in images; face and voice recognition and natural language processing are examples of such fields where security and trustworthiness and, as a result adversarial robustness is of significant important.

Appendix A

Adversarial Attack Paramaters

```
FGSM(model, eps=0.3),
PGD(model, eps=0.3, alpha=0.1, steps=7),
CW(model, c=5, lr=0.001),
```

Bibliography

[1] Tao Bai, Jinqi Luo, Jun Zhao, Bihan Wen, and Qian Wang. Recent advances in adversarial training for adversarial robustness, 2021.

[2] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks, 2016.

[3] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[4] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *Proceedings of the International Conference on Learning Representations*, 2019.

[5] Hoki Kim. Torchattacks: A pytorch repository for adversarial attacks. *arXiv preprint arXiv:2010.01950*, 2020.

[6] Norman Mu and Justin Gilmer. Mnist-c: A robustness benchmark for computer vision, 2019.

[7] Zeke Xie, Fengxiang He, Shaopeng Fu, Issei Sato, Dacheng Tao, and Masashi Sugiyama. Artificial neural variability for deep learning: On overfitting, noise memorization, and catastrophic forgetting. *Neural Computation*, 33(8):2163–2192, 2021.