POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Analysis of CNN Computational Profile Likelihood on Adversarial Attacks and Affine Transformations

MIRA MARHABA
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Avril 2022

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Analysis of CNN Computational Profile Likelihood on Adversarial Attacks and Affine Transformations

présenté par **Mira MARHABA**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Heng LI, président
Ettore MERLO, membre et directeur de recherche
Michel DESMARAIS, membre

DEDICATION

To my beloved grandmother Laila Mawlawi,
a strong advocate for education,
whom I lost along the way...

ACKNOWLEDGEMENTS

I am very lucky to have received much support and encouragement throughout this journey.

First and foremost, I would like to thank my supervisor, Professor Ettore Merlo, for his invaluable guidance and from whom I learned a great deal. You were always available for any assistance or advice, despite your busy schedule, and for that I am extremely grateful.

I would like to thank the DEEL (DEpendable Explainable Learning) projet and all its participants, funded by the National Science and Engineering Research Council of Canada (NSERC) and the Consortium for Research and Innovation in Aerospace in Quebec (CRIAQ), together with its industrial partners Thales Canada Inc., Bell Textron Canada Ltd., CAE Inc. and Bombardier Inc.

I would also like to thank the jury members, Professor Heng Li and Professor Michel Desmarais, for accepting to review my thesis.

In addition, thank you to my loving and supportive parents who have always done their best to inspire me to be the best version of myself. I wouldn't be the person I am today without you.

I would also like to thank my other half, my husband Tarek, without whom I would not have been able to complete this thesis. You were my rock throughout it all, and I couldn't have done it without you.

Lastly but certainly not least, I would like to thank my siblings, always providing me with advice and emotional support in times of stress, as well as my parents-in-law, who always did their best to help out whenever possible.

RÉSUMÉ

Les réseaux d'apprentissage automatique peuvent être vulnérables lorsqu'il s'agit de classer les entrées qui sont hors de la distribution observée pendant l'entraînement. Pour identifier ces cas, qui pourraient être interprétés différemment et éventuellement mal classés à cause de cette différence, nous proposons une approche non paramétrique basée sur les classes, inspirée de Surprise Adequacy Deep Learning Likelihood (SADL).

Nous ciblons la partie entièrement connectée des réseaux de neurones convolutifs (CNN) et estimons la probabilité conditionnelle des niveaux d'activation des neurones internes d'un réseau pendant la reconnaissance. Les distributions observées lors de l'entraînement et des tests sont prises comme point de référence et sont comparées à celles observées lors du traitement des profils de calcul d'autres cas non familiers tels que les attaques adversariales et les transformations affines. Deux catégories de ces entrées non familières sont considérées: celles qui sont dans la distribution des entrées d'entraînement (In-Distribution - InD) et celles qui sont hors distribution des entrées d'entraînement (Out-Of-Distribution - OOD). Un seuil linéaire basé sur sigma est utilisé pour séparer les cas InD des cas OOD.

Toutes les expériences sont effectuées sur des CNN entraînés à l'aide d'un sous-ensemble de la base de données MNIST-fashion, qui est un ensemble d'images de vêtements accessible au public. Pour éliminer les biais associés à l'utilisation d'un seul réseau, des expériences finales sont effectuées sur dix réseaux avec des structures différentes et cinq variantes d'un réseau entraîné sur les mêmes entrées.

Les résultats expérimentaux présentés montrent que les probabilités de calcul des cas adversarials et des transformations affines couvrent une plage beaucoup plus large et étendue par rapport aux cas de l'ensemble d'apprentissage. Une minorité de ces cas se situent dans la plage de distribution de l'entraînement, tandis que les entrées mal classées correspondent très souvent à des profils de calcul OOD différents de ceux obtenus lors de l'entraînement. Les expériences montrent que l'identification OOD permet de réduire jusqu'à 70% à 90% les erreurs de classification, en les filtrant, souvent sans réduire de manière significative la quantité d'entrées correctement prédites. De plus, les résultats expérimentaux indiquent que toutes les classes de sortie ne sont pas également sensibles et vulnérables aux entrées contradictoires et aux transformations affines.

Cette identification des entrées OOD peut être bénéfique dans des domaines sensibles et critiques tels que l'aérospatiale, la médecine, la cybersécurité et bien d'autres, où il peut être difficile de prévoir des échantillons appropriés et représentatifs de cas inconnus ou inattendus.

ABSTRACT

Machine learning networks can be vulnerable when it comes to classification of inputs that are out of the distribution that is observed during training. To identify these cases, which could be interpreted differently and possibly misclassified due to this difference, we propose a non-parametric class-based approach based on Surprise Adequacy Deep Learning Likelihood (SADL).

We target the fully connected part of Convolutional Neural Networks (CNNs) and estimate the conditional probability of a network's internal neuron activation levels during recognition. The distributions observed during training and tests are taken as a point of reference and compared with those observed when processing computational profiles of other unfamiliar cases such as adversarial attacks and affine transformations. Two categories of these unfamiliar inputs are considered: those that are In-Distribution (InD) of the training inputs, and those that are Out-Of-Distribution (OOD) of the training inputs. A linear sigma-based threshold is used to separate the InD cases from the OOD cases.

All experiments are performed on CNNs trained using a subset of the MNIST-fashion database, which is a publicly available set of clothing images. To eliminate bias associated with using one single network, final experiments are performed on ten networks with different structures and five variants of one network trained on the same inputs.

Presented experimental results show that the computational likelihoods of the adversarial cases and affine transformations span a much wider and extended range with respect to the training set cases. A minority of those cases lie in the training distribution range, while misclassified inputs very often correspond to OOD computational profiles that are different than those obtained during training. Experiments show that the OOD identification allows up to 70% to 90% reduction of misclassifications, by filtering them out, often without significantly reducing the amount of correctly predicted inputs. Furthermore, experimental results indicate that not all output classes are equally sensitive and vulnerable to adversarial inputs and affine transformations.

The identification of OOD computations may be beneficial in sensitive and critical domains such as aerospace, medicine, cyber-security, and many others, where it may be hard to forecast proper and representative samples of unknown or unexpected cases.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

CPL	Computational Profile Likelihood
CNN	Convolutional Neural Network
DNN	Deep Neural Network
FNN	Feed Forward Neural Network
InD	In Distribution
OOD	Out of Distribution

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

1.1 Motivation

These days, convolutional neural networks, more commonly referred to as CNNs, are increasingly becoming more integrated in large and industrial software systems in many fields. The ideal favorable outcome would be that after deployment, with new and unfamiliar inputs, the system continues to behave similarly as it does with the data that it was previously trained and tested on. However, this is generally not the case, and systems tend to perform significantly worse on operational data than they do on the training and testing data.

Furthermore, this leaves the network vulnerable and gives opportunity to the malicious inputs of adversarial attacks or affine transformations of inputs that lead the CNN to make erroneous predictions. This can have a very negative impact in sensitive and critical domains, such as aerospace, medicine, finance, or cyber-security, which require a higher confidence in CNN decisions and more robustness against unfamiliar inputs.

Thus arises the need to find approaches that increase a network's reliability after being deployed, and testing for cases that may decrease or eliminate its reliability. There are many challenges to testing a CNN system. One difficult challenge is the oracle problem, in which it is impossible or very expensive to expect to know all the correct answers to all possible inputs being passed into the CNN, which limits the ability of verifying the CNN's performance and improving it when it comes to unknown inputs. Another challenge faced during validation and verification of the network is when a faulty CNN is passed some input but is able to correctly make a prediction solely due to mere coincidence. This makes it very difficult to discover the defect of the program, particularly with unfamiliar inputs, as no failure is detected.

There has been much recent research revolving around improving a network's prediction accuracy and confidence, particularly for the detection of out-of-distribution data to recognize when the presented data is different from the data that the network is used to seeing. When the distribution of operational data is different from the distribution of the data used for training and testing the system, the network may have reduced performance and unexpected results in the model's outputs, unless there is a way for it to differentiate between the familiar cases and those cases that seem to come from different distributions.

1.2 Our Approach

In this thesis, we investigate the response of CNNs to compare original training images to test cases that were not drawn from the same distribution of the training sets. To be more specific, we investigate the use of synthetic test cases obtained from affine transformations of legitimate inputs and from adversarial attacks.

We propose a novel statistical white-box approach to quantify the "unusual reasoning" of the decisions that the network is making, which we refer to as CPL: a non-parametric class-based variant of Surprise Adequacy Deep Learning Likelihood (SADL) [1] that doesn't require retraining. This measure essentially indicates the likelihood of a network's prediction, when compared to a standard input distribution, where a larger CPL indicates a less probable prediction.

This likelihood measurement is based on the inference of a stochastic model of vectors of activation (excitation) levels produced by the neurons in a network's layers. We call these vectors computational profiles, and we refer to the likelihood measure as Computational Profile Likelihood (CPL). During classification, we compare the CPL values of an incoming input, that has been best classified into an output class by the network, against the distribution of computational profiles observed during training for the same output class. The comparison aims at the identification of those inputs whose likelihoods are separable or are Out-of-Distribution (OOD) with respect to the network's training set computational profiles.

Our motivation for this "reasoning" measurement of likelihood comes from the intuition that a network's precision and performance observed during training and tests cannot be assumed to be the same when dealing with unfamiliar operational cases that traverse very different computational paths in the network during prediction. Indeed, we don't know the a-priori precision and performance of a network in these "different" or OOD cases. Sometimes the prediction is correct, but very often it's incorrect, especially for the corner cases that are tests specifically designed to test rare and unusual events and that may cause a network to fail.

The proposed approach is described in detail in Section 4. CNN training has been performed on images from the MNIST-fashion database [2] and non-parametric statistical models have been estimated during training, without using a secondary classifier on adversarial or affine examples. Many of our experiments have been performed on different architectures, each with training and testing accuracy of over 90%, to reduce the bias coming from initial values of parameters in the learning process.

Presented results show that for investigated affine and adversarial cases, the likelihood of

computational profiles observed during prediction is often significantly lower than that observed for computational profiles corresponding to training and test sets, and are quite separable from the original data distribution. Furthermore, by applying a simple linear threshold separation on computational profile likelihoods, we are successful in identifying many OOD computational profiles corresponding to cases that were not drawn from a distribution similar to that of the training set.

With some additional research, this presented approach may be useful as a defence against unusual cases, however we focus on using it as an OOD detection tool to filter and classify the computational likelihood of incoming inputs. Potentially, these difficult and uncertain cases can be flagged as unreliable predictions or bugs in the system to be reviewed by developers.

1.3 Research Objectives

Objectives of this thesis are to take a closer look at a network's activation levels, in the form of Computational Profile Likelihoods, as an attempt to assess the robustness of a network and recognize unlikely decisions made by this network. This can be done either by comparing an input distribution to a more familiar training distribution and separating between the two sets, or by simply setting a threshold that separates any incoming input from the familiar distribution.

Ideally, this works towards the goal of certification in Machine Learning, as a threshold can be customized to suit different fields and even different classes depending on the needs of the domain and on the trustworthiness of the class.

1.4 Thesis Contribution

Major contributions of this thesis can be summarized as follows:

- the definition of non-parametric statistical models based on computational profiles
- the definition of computational profile likelihood (CPL)
- experimental results of computational profiles likelihood computed on inputs belonging to legitimate train and test cases
- experimental results of computational profiles likelihood computed on inputs belonging to affine and adversarial cases generated from legitimate cases

- experimental results of per-set separation of computational profile likelihoods between train cases and other non-train inputs
- experimental results of per-input OOD analysis of computational profile likelihoods of non-train inputs

Throughout this thesis, the following two publications were made:

- Paper in iMLSE 2020 (2nd International Workshop on Machine Learning Systems Engineering) [3]
- Poster in ICSE CAIN 2022 (1st International Conference on AI Engineering - Software Engineering for AI) [4]

1.5 Thesis Outline

The rest of this document is organized as follows:

- Chapter 2 reviews concepts related to deep learning, unfamiliar CNN inputs, and detection of Out-Of-Distribution inputs. These fundamental topics are necessary to understand the work done in our experiments.
- Chapter 3 includes a review of relatively recent works that are related to our research work and that helped guide many of our experimental decisions.
- Chapter 4 outlines our methodology for the different experiments as well as details pertaining to the setup of our experiments and the datasets used.
- Chapter 5 presents and discusses the results obtained from our different approaches.
- Chapter 6 resumes the goals, methods, and results presented in this thesis. It also concludes the thesis and discusses limitations on the work and future potential research to build upon what was done.

CHAPTER 2 BACKGROUND

2.1 Deep Learning and DNNs

Deep Learning, a subset of Machine Learning and Artificial Intelligence, builds on traditional linear algorithms by stacking multiple levels of representations and creating a hierarchy of layers that progressively increase the level of complexity and extract abstracted features from inputs passed through the layers. This statistical model is particularly beneficial for high-dimensional data as the goal is to learn different features as well as to find patterns, creating elaborate mapping functions between raw inputs and final decision-based predictions.

Deep learning accomplishes this mainly with the use of Deep Neural Networks (DNNs), biologically-inspired deeply-connected networks multiple layers with multiple layers of neurons. Through the different layers, these DNNs take raw data as input and, after a series of transformations and abstractions, output a decision or prediction relevant to the domain and problem at hand.

This complex computational approach is relatively new, as processors are becoming more and more powerful. In the past, it was difficult to implement DNNs. The more layers they had, the more neurons and, by extension, parameters there were to keep track of. It was heavy both in terms of computation and in terms of memory needed to store data, parameters, and hyperparameters.

However, in recent days, processors are getting more powerful and have become capable of implementing DNNs with many more layers. Furthermore, data is becoming increasingly more available as more and more information is collected, organized, and shared. These two facts have encouraged the use of DNNs in research projects all over the world. In fact, they have become a very popular tool to extract patterns from data and use it in real-world applications.

This work focuses primarily on Feed Forward Neural Networks (FNN), one of the most widely used in the Deep Learning domain, although with some extra research the methods presented could be applied on other types of models as well. FNNs are especially useful for classification, as they take input of a constant size with the goal of mapping it to some sort of output of a constant size, such as a probability assigned to each possible class. An example of an FNN is shown in 2.1. The data strictly flows from the left of the network to the right without any chance of information flowing back to previous layers. This is why they are called Feed Forward Networks.

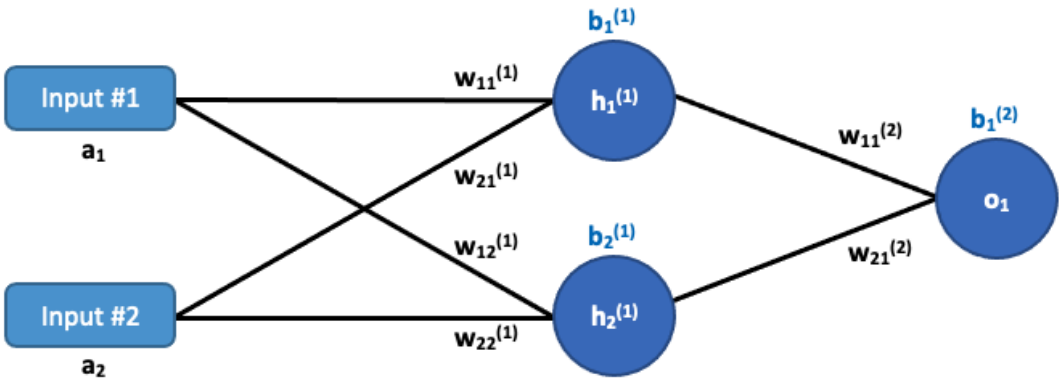

Figure 2.1 Demonstration of the Inner Components of an FNN

Between the input and output layers, FNNs have any arbitrary number of hidden layers. These intermediate levels of computation are referred to as *hidden* due to the fact that they contain only intermediary forms of data, as opposed to the user-facing data seen as input or output. Each neuron in the intermediate layers takes a weighted sum of the outputs of the layer before, and passes it through a function that adds non-linearity in the mapping. This is called the activation function, and it helps with avoiding sensitivity to variations of the input that should not affect the output. The more neurons or layers the FNN has, the more complex relationships it is able to approximate. This is why deep learning has benefitted from using it, as researchers are able to model more complex behaviours by adding more and more layers and/or neurons.

A specific type of FNN used in this work is a Convolutional Neural Network (CNN), which adds specialized layers of convolution and pooling to better transform and process data that is in the form of multiple arrays. This is a widely used type of network in recent works, and has achieved much success when it comes to processing images or videos.

2.2 Metamorphic Transformations

The use of metamorphic relations when analyzing or testing software was first mentioned in 1998 [5]. Since then, it has grown to be used by many in all sorts of software domains, one of which is deep learning and DNNs. For our purposes, a metamorphic relation is a property that should hold true regardless of how many times it is executed and regardless of which input it is used on. A simple example would be the function $f = sin(x)$ and the fact that

$sin(x) = sin(x + 4\pi)$ should always hold true, thus this property would be considered a metamorphic relation. Similarly, going from x to $x + 4\pi$ to be used as input to the function f would be considered a metamorphic transformation.

In the case of using metamorphic transformations with a DNN, the point is to take an image for which the true class is known and the image is correctly predicted by the network. The transformation is then applied to the image which should preserve the relations within the image and ideally result in it being predicted correctly as belonging to the same original class.

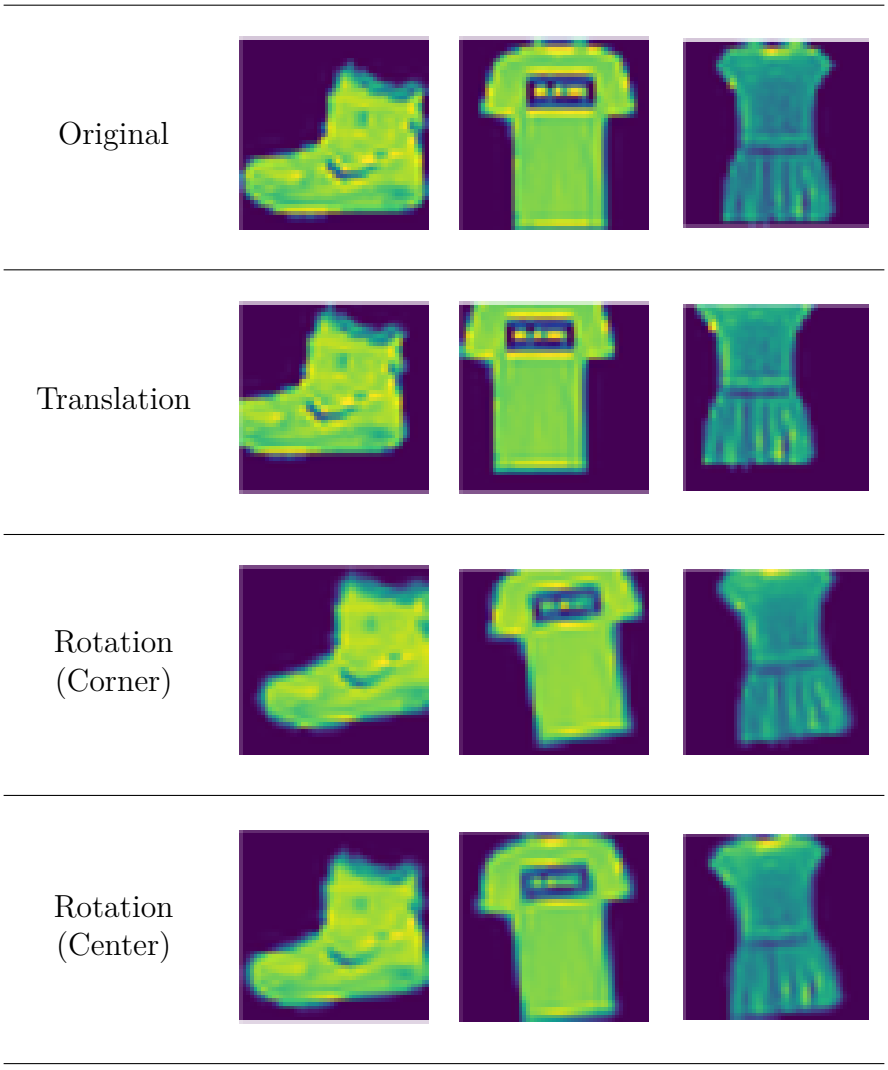

Figure 2.2 Example Transformations

Particular examples of metamorphic transformation used in this work are translation or rotation on the images. Examples of the transformations are shown in Figure 2.2. Translation

is a transformation that shifts the pixels of an image by a particular unit of pixels along the x and/or y axis. As for the rotation of the images, it is performed in two different ways. The image is either rotated by the top left corner, in which it slowly shifts off the screen, or the image is rotated strictly by its centre. We discuss these transformations and their role in our particular experiments in greater detail in Section 4.7.

2.3 Adversarial Attacks

Deep Neural Networks are vulnerable to crafted adversarial attacks. In these attacks, an adversary takes an input that is initially correctly classified by the network and applies a sort of perturbation to the input or attempts to transform it in such a way that the network now incorrectly classifies it with very high confidence. There are many types of adversarial attacks with varying strengths and many different ways to apply them, but their purpose remains the same. Unfortunately, although there exist many recent works implementing defenses against adversarial attacks, many of the techniques can be defeated, and DNNs are still susceptible to being fooled by these malicious inputs.

There are many good examples in the literature, though one particular one that comes to mind [6] shows a very clear example of an adversarial attack. In this example, we see an input image x which is correctly classified by the network as *panda* with a confidence of only 57.7%. By simply adding a slight layer of perturbation with an ϵ of 0.007 to x, the adversary succeeds as the network now classifies the image as *gibbon* with a very high confidence of 99.3%.

Some examples of adversarial images used in this work are Fast Gradient Sign [6], Jacobian Saliency Map [7], DeepFool [8] and Carlini & Wagner [9]. More details about these attacks and how they were used in the experiments can be found in Section 4.7.

2.4 DNN Out-Of-Distribution Detection

Many DNNs, particularly those with the softmax activation function, tend to have very high confidences even when incorrectly classifying inputs. With these DNNs it is very difficult to recognize cases where the network may be mistaken. It is therefore beneficial to develop a method that statistically analyzes a DNN's computations to ideally separate between the predictions that are likely to be correct and those that have low probability of being correct.

Using such a method, it is then possible to flag the cases that are unlikely to be correct as Out-Of-Distribution (OOD), as they are deemed to have a different distribution than the

legitimate inputs that the network is able to properly label. Conversely, the cases considered as In-Distribution (InD) with the same approach would be the cases that are likely to be correctly predicted by the network as they have a similar distribution to the legitimate inputs.

This is especially vital in critical scenarios where a deep learning model's prediction can have dangerous consequences when incorrect. It wouldn't be too worrisome to take the risk of some incorrect predictions when asking for product recommendations, but can be fatal if an autonomous car were to make an incorrect decision with high confidence. In the latter example, a method of identifying and flagging the decisions that are very different (OOD) from the distributions of proper decisions could help avoid the suspicious erroneous behaviours.

There can be many different ways of defining an input set's distribution, and thus the approach of deciding whether the inputs are In- or Out-Of-Distribution differs as well. In this work, the distribution is found using the Computation Profiles computed by the network when passing inputs through. Deciding whether an input is In- or Out-Of-Distribution can then be performed by setting a linear threshold to one-dimensionally separate the inputs. More details about our method can be found in Section 4.

CHAPTER 3 LITERATURE REVIEW

Deep neural networks (DNNs) are more and more integrated in large and industrial software systems across many fields. Unfortunately, these systems are vulnerable to unknown inputs including affine transformations of images and adversarial attacks on benign images. Many sensitive and critical domains, such as aerospace, medicine, finance, or cyber-security require higher confidence in the network's prediction along with robustness against unfamiliar inputs and certification of these systems according to field-appropriate standards. Recently, much research is being applied towards making progress in these directions, particularly with deeper "white-box" investigation of networks.

This chapter reviews relatively recent related works that make progress towards the goal of Dependable, Certifiable and Explainable AI systems for critical scenarios. Section 3.1 presents several prominent works that analyze the internal computations performed by a neural network to assess and evaluate robustness or to leverage the information to better understand the network, namely using the values of the DNN's activation levels or the neuron coverage of the network. Section 3.2 covers the investigation of one particular layer as opposed to the investigation of all layers of a network as a whole. Section 3.3 discusses recent approaches to DNN robustness against unknown inputs, particularly metamorphic transformations and adversarial attacks.

3.1 White-Box Testing

3.1.1 Extraction and Use of DNN Activation Levels

Many recent works investigate how using the activation levels, which are the inner computations of a neural network, can contribute to getting more information and better understanding the network. This type of information would potentially contribute to the complex goals of explainability of the network or robustness of a network against different inputs other than the training data.

Papernot [10] [11] presents distillation of neuron activation levels from training and adversarial inputs and trained a secondary DNN as a defensive measure for robustness. Another paper [12] investigated neuron activation levels and a secondary classifier based on the nearest neighbors method computed using locality-sensitive hashing.

Xiao et al. [13] present an approach that also takes a deeper look into the internal DNN layers and activations. Their approach is only applied after the model has been trained and does

not affect training. It uses kernel density estimation (KDE) to extrapolate the probability density distributions of each layer's output by evaluating the DNN on the training data, per class. It then infers the density probabilities of each layer for any given test instance for each class. The higher the values of the class, the more similar the features of the instance in this layer are to the specific class. Search-based optimization is then used to find the most optimal layer combinations to be able to give an alarm if the layer features are inconsistent with the final prediction, and give advice in the form of an alternative prediction.

Guerriero et al. [14] introduce DeepEST, which samples from operational data (highly representative data) many failing tests to learn from while building a set of tests that are suited for the DNN accuracy estimate. They do this with the goal of providing accuracy estimates that are much more precise and efficient as well as the practical advantage of enabling retraining for further improvement. In other words, they look for the mispredictions that are the "most informative" to then be able to use them for future training or improvement of the DNN. This is done with an approach based on Surprise Adequacy, however they suggest that the distance metric by itself is not enough. They suggest that both confidence and distance are metrics that should be used, and they introduce an equation that uses a combination of the two.

Ravi Mangal et al. [15] presented an approach for robustness of neural networks based on non-adversarial real-world input probability distributions. We share the perspective of stochastic modeling, but we concentrate on computational profiles of networks with no assumptions about input distributions. Another notable example is DeepCon [16], in which the authors introduce an approach to measure the testing adequacy of a DNN that uses neuron values and their weights. In contrast to their work, we rely on the distribution not being Gaussian, and our analysis is class-dependent.

In the work [17], the authors also present a somewhat similar approach to use a network's activation levels to better understand it, however this approach uses Kernel-Density Estimation and they look at several different layers of the network, with their main focus on how to best aggregate these layers. They do not elaborate on the use of their approaches with respect to any unknown or adversarial input.

3.1.2 Neuron Coverage

In a recent short article [18], the authors look at several kinds of coverage, including neuron coverage as a percentage of activated neurons based on inputs. They find that there is a low correlation between the number of misclassified inputs in a test set and its coverage, and thus that coverage (including neuron coverage) is actually misleading. They also look

at adversarial inputs and consider how coverage can be used to measure the robustness of a DNN. They find that the adversarial examples are pervasive and thus the number of adversarial misclassified images found when trying to achieve high coverage depends on the search method used. However, they specify that their conducted experiments were only preliminary and that more will need to be conducted for more conclusive results.

In another later work, Kim [19] shows that neuron coverage is not significantly statistically related to adversarial case detection, and that seeking higher neuron overage doesn't necessarily provide better detection of adversarial cases.

RAID [20] considers the activation levels of a subset of relevant neurons with the highest difference values with respect to adversarial inputs to retrain a secondary classifier and assess the confidence of the predictions. DeepXplore [21] is another neuron coverage approach that flags incorrect behaviour in deep learning systems without the need of manually labeled data, reducing the need for a 'ground truth' oracle. It analyzes the neurons in the network in order to compute the rate of those that are activated, and it relies on training a new model or retraining the original. The problem is framed as a joint optimization problem and uses gradient ascent to solve it efficiently.

Ma et al. [22] also use neuron coverage when presenting DeepGauge, a set of multi-granularity testing criteria to better analyze and measure the quality and effectiveness of the data being used to test the deep learning system. They leverage the use of neuron coverage criteria at different layer levels, to identify corner-case behaviours, and discuss the possibility of using these criteria for automating white-box tests for deep learning systems.

3.2 Analyzing Different Layers

In [23], the authors investigate a way to solve a DNN from "overthinking", in which a final (correct) prediction may be reached within the DNN before reaching the final layer. They propose Shallow-Deep Networks, by training multiple classifiers on the different layers, and an algorithm of confidence-based decisions of exiting the computation early, with the goal of reducing computation cost and preserving or increasing accuracy. They argue that a DNN's last layer by itself is not enough to characterize a DNN, and that sometimes the last layer is what turns a correct prediction into an incorrect one. However, their arguments and experiments are preliminary and are not verified against adversarial inputs, and thus perhaps there is not sufficient research to conclude that this discovery is always applicable.

On the other hand, other works align well with using only a particular layer to allow faster computation, since less neurons are considered and they have higher statistical independence

from previous layers, in contrast with analyzing the whole network [1, 12, 20].

3.3 DNN robustness against Affine Transformations and Adversarial Attacks

Engstrom et al. [24] discuss applying affine transformations, namely translation and rotation to test robustness of deep learning vision-based models. They show that these simple black-box transformations are able to easily fool CNNs without complicated or advanced optimization techniques as used in other works.

Many works [25] [26] show how vulnerable DNN systems are when faced with adversarial examples. Some demonstrate how effective it could be to add these aggressive cases to the training data after identifying them, and show that it could lead to increased resilience of the model, as well as use the incorrectly predicted cases by the network to identify violations of safety properties in the system.

In a recent work, Dissector [27] is presented as a fault tolerance approach, in which the authors introduce the concept of within-inputs (inputs that are within the model's handling capability) and beyond-inputs (inputs that are beyond the model's handling capability). They relate the concept of the model's confidence to whether the inputs are within or beyond. They attempt to distinguish the "beyond" inputs from the "within" inputs, with the hope of identifying the inputs that are more likely to be incorrectly predicted. They do this without the need of model retraining or redeploying applications. They have the goal of defending against adversarial inputs and they also analyze the inner DNN workings to compare between known data and new different data. However their approach first creates sub-models to represent different levels of knowledge in the model. They then pass the inputs through the submodels and make "snapshots" of each for the different layers (which they refer to as prediction profiles). These prediction profiles are then compared to calculate a confidence score to check whether the sample is within input.

Other approaches use activation levels to detect adversarial techniques. Some examples are [28–30]. They all demonstrate satisfactory results and, very similarly to RAID [20], they all rely on training a secondary classifier to be effective, while our approach does not.

Other approaches to detect adversarial images vary from using techniques like Principal Component Analysis [31–33] to Kernel-Density Estimation and Bayesian Neural-Network Uncertainty [17, 34]. However each and every one of these techniques can be defeated by choosing a specific loss function depending on the defense [35].

The analysis of a network's internal computation values to differentiate between adversarial samples and benign inputs has been explored and presented in the literature [36, 37]. This

presented thesis extends the investigation to include metamorphic tests.

3.4 DNN Out-Of-Distribution Detection

It is known that neural networks generalize well when it comes to sampling from the same distribution for training and testing data. This is demonstrated in several research works [38] [39] [40] [41] where the authors successfully train and test networks with great accuracy on large and challenging datasets. However, it is often very difficult to obtain such high accuracies when it comes to operational data.

Nguyen et al. [42] show how a neural network can have very high confidence for its predictions when it comes to inputs that are not even recognizable. Similarly, other works [43] [44] [45] show very high confidence for inputs that are completely irrelevant.

The idea of detecting out-of-distribution inputs is not new. It has previously been studied for many applications, as described in the review by [46], and referred to as "Novelty Detection". Low-dimensional out-of-distribution inputs are shown to be detected using conventional approaches such as nearest neighbour, clustering analysis, and density estimation [47] [48] [49]. Nevertheless, research works suggest that these methods are still not reliable for data that has many dimensions across multiple fields [50] [51]. In another work, Schlegl et al. [52] demonstrate training a generative adversarial network (GAN) with the goal of detecting out-of-distribution inputs in a clinical environment. Hendrycks and Gimpel [44] present an effective heuristic that serves as a simple baseline to detect Out-Of-Distribution cases based on a threshold, using the probabilities obtained from their softmax distributions. They notice that the maximum softmax values tend to be larger when the prediction is correct, compared to the lower maximum values seen for faulty predictions flagged as out-of-distribution cases.

3.5 Certification of Safety-Critical Systems

In a recent White Paper of the DEEL project [53], the authors summarize machine learning techniques and present a taxonomy of them, within the context of using machine learning in sensitive domains that are safety-critical (namely aerospace and automotive domains). Seven challenges of machine learning in critical systems associated with certification are presented, including but not limited to resilience, explainability, and robustness.

Another recent work [54] also addresses the need of certification when it comes to safety-critical systems. This systematic literature review considers research papers published within the last five years that relate to machine learning systems certification. In the 217 reviewed

papers, they reach several notable conclusions.

First, though the community has expressed much interest in certification of ML systems, there is still a significant lack of diversity when it comes to use cases used in these investigations as well as the types of ML models tested. They also encourage developing deeper connections between academic and industrial domains for research on this subject. The majority of papers found were purely academic, and enriching the experiments with more practical industrial data could prove to be very useful when assessing safety-critical scenarios.

The authors also emphasize six topics that are considered to be the most relevant concepts when it comes to certification: robustness, uncertainty, explainability, verification, safe reinforcement learning, and direct certification. In the reviewed papers, the authors notice that these concepts seem to be studied independently from each other, and suggest that instead connections should be built between them to further investigate them on a deeper level.

3.6 Inspiration and Proposed Differences

The approach introduced in this thesis is inspired by the Surprise Adequacy for Deep Learning Systems (SADL) approach [1]. SADL uses neuron activation values to calculate the level of "surprise" between training images and adversarial images, which is a test criterion to measure how different a neural network's input is from its training data. They then use these "surprise" values to retrain a classifier to avoid the misclassification of those adversarial images.

Our approach differs in several ways. Firstly, ours is non-parametric, and uses histograms, whereas SADL relies on having a Gaussian distribution. Furthermore, a novel aspect of our approach is the fact that it is class-based, allowing a finer analysis of each class independently. It is also worth mentioning that our approach does not require retraining the same model or any new model, and can be used dynamically on existing systems.

CHAPTER 4 METHODOLOGY AND EXPERIMENTS

4.1 Computational Profile Extraction

As mentioned in Section 2.1, a neural network is typically composed of several layers of neurons that carry and propagate activation levels across layers using weights and transfer functions. Our work proposes a novel statistical approach to measure the "reasoning" likelihood of a network during its predictions.

This likelihood measurement, further explained below in Section 4.2, is computed using vectors of activation levels of the neurons in the layers of the network. These vectors are what we call computational profiles. We compute the Computational Profile Likelihood (CPL) of incoming inputs, that have been best classified into output classes by the network, and compare them against the distribution of computational profiles observed during training for the same output classes.

Motivation for reasoning using the measurement of likelihood comes from the intuition that a network's precision and performance observed during training and tests cannot be assumed to be the same when dealing with unfamiliar cases that traverse very different computational paths in the network during prediction. These could, for example, be affine transformations of images or adversarial inputs and indeed, we don't know the a-priori precision and performance of a network in these cases that have different distributions than the cases that the network is used to processing. Sometimes the prediction may even be correct, but with differently traversed computational paths. However, it is very often incorrect, especially for the "corner cases" that are tests specifically designed to test rare and unusual events and that may cause a network to fail.

In a recent work, Kim [19] showed that neuron coverage is not significantly statistically related to adversarial case detection. Indeed, we share the opinion and we don't seek higher neuron coverage, but we consider the actual neuron activation levels as a sort of statistical signature used to identify unusual computational profiles during prediction. Investigating the CPL's of a network enables investigation of the potential reasoning behind the network's predictions, and allows comparison between the distributions of these corner cases and the familiar training cases.

For each experiment, we consider a single architecture a from the set A of all architectures. Each architecture a is trained on the training set X_a, a subset of all possible training inputs X, for the set of all classes K. We take the subset $X_a^{'}$ as all the correctly classified images

by a from X_a.

To extract the computational profiles, we consider the activation level $actLev(i, j, x, k, a)$ of the i-th neuron in the j-th layer of architecture a, where x is an input from $X'_{k,\ a}$, belonging to a class k from K. We compute in Equations 4.1 and 4.2 respectively the average and standard deviation of $actLev(i, j, x, k, a)$ across the different inputs x.

$$refNodeAvgLev(i, j, k, a) = \frac{1}{\mid X'_{k,\ a} \mid} \cdot \sum_{x \in X'_{k,\ a}} actLev(i, j, x, k, a) \tag{4.1}$$

$$\begin{aligned} &refNodeStdDev(i, j, k, a) = \\ &= \sqrt{\frac{\sum_{x \in X'_{k,\ a}} (actLev(i, j, x, k, a) - refNodeAvgLev(i, j, k, a))^2}{\mid X'_{k,\ a} \mid}} \end{aligned} \tag{4.2}$$

4.2 Computational Profile Likelihood

Bin frequencies $bFreq(b, i, j, k, a)$ are computed during training by counting how many times inputs from training set X'_a and belonging to class k have produced an activation level that falls into bin b for neuron (i, j).

Since we discovered that the distributions of the computational profiles are not normal, we use non-parametric statistics [55] to estimate the density of the underlying distribution of the neuron activation levels using histograms. The resolution of distribution estimation is determined by the width of histogram slots. To have a uniform resolution across the neuron distributions, bins of variable size $width(i, j, k, a) = c * refNodeStdDev(i, j, k, a)$ are used for a neuron (i, j).

In the presented experiments, $c = 1$ has been used, but $c = 0.5$ could be used for a finer estimation of distributions. The average $refNodeAvgLev(i, j, k, a)$ and the standard deviation $refNodeStdDev(i, j, k, a)$ of each neuron activation level has been computed during training.

Bin probabilities have been computed from bin frequencies as follows:

$$p(b,i,j,k,a) = \frac{1}{\mid X'_{k,\ a} \mid} \cdot bFreq(b,i,j,k,a) \tag{4.3}$$

To smooth probabilities over the bins, a very low probability is nevertheless assigned to all bins with null frequencies.

The Bayesian likelihood $L(y,j,k,a)$ of the any input y from an arbitrary class Y, given class k for layer j of architecture a is computed as the joint probability of all neurons composing a layer. We convert it to logarithmic likelihood for practical reasons and refer to it as Computational Profile Likelihood (CPL), as follows:

$$\begin{aligned} L(y,j,k,a) &= \prod_i p(b,i,j,k,a) \\ CPL(y,j,k,a) &= -\sum_i log(p(b,i,j,k,a)) \end{aligned} \tag{4.4}$$

In the presented experiments, we concentrated on only the second to last layer and results have been obtained using $CPL(y, N-1, k, X)$, where N is the number of layers in a network and $(N-1)$ is the layer before the output layer. A discussion about this choice is presented in section 4.7.2.

We thus define the computation used in our experiments that we refer to as Architectural CPL in Equation 4.5, which is the CPL computed for any input y predicted to be from class k by architecture a, considering only the second to last layer.

$$archCPL(y,k,a) = CPL(y, N-1, k, a) \tag{4.5}$$

4.3 Per-Set Best Separation

Once we compute the architectural CPL of each image of a set as per Equation 4.5, we end up with a distribution of Computational Profiles for that particular set. Given one set of legitimate basic images and one set of input images, we would like to compute the point at which separating the two sets results in the best precision of both sets, without compromising the precision of the other. In our case, the primary legitimate set would be the set of training

images, and the input set would be whichever group of images we are currently experimenting with, such as a type of transformation or adversarial attack.

If we were dealing with normal distributions, the easiest way to separate between the two distributions would have been to simply find the intersection of the distributions. However since the data we are dealing with is not of a normal distribution, we must employ a different method of finding the best separation point between the distributions.

This optimization is done in a straightforward manner, where we place a threshold at the right extremity of the rightmost distribution. Then, we simply loop through each point and compute the precision of separation for each distribution at that threshold point. Since we keep track of the previous iteration's separation point, we are able to identify the point at which the difference between the two sets' precisions flips sign, that is, goes from negative to positive or from positive to negative. Another way to describe this separation point would be the critical point at which the absolute value of the difference of the precisions decreases to 0 and then starts increasing again.

This separation cutpoint at which the sign is flipped is computed along with the precisions of separation for the first legitimate set of images (training images) and the second input set of images. This method is performed for each set of input images, each true class, and each best predicted class. The results are presented and discussed in Section 5.2.1.

4.4 Sigma-Order Threshold

For each architecture a and each class k, we compute the architectural CPL mentioned in eq. 4.5 using each image x from the architecture's correctly predicted set of class k training images $X'_{k,\ a}$. The average (eq. 4.6) and standard deviation (eq. 4.7) of these architectural CPL values are computed for each class k of the architecture, and used to obtain a sigma normalized version of the architectural CPL in equation 4.8 for any y input from arbitrary set of inputs Y.

$$refArchAvg(k,a) = \frac{1}{\mid X'_{k,\ a} \mid} \cdot \sum_{x \in X'_{k,\ a}} archCPL(x,k,a) \tag{4.6}$$

$$refArchStdDev(k,a) = \sqrt{\frac{\sum\limits_{x \in X'_{k,\ a}} (archCPL(x,k,a) - refArchAvg(k,a))^2}{\mid X'_{k,\ a} \mid}} \tag{4.7}$$

$$normArchCPL(y,k,a) = \frac{archCPL(y,k,a)}{refArchStdDev(k,a)} \tag{4.8}$$

4.5 Per-Input Out-Of-Distribution Computation

Each input y's $normArchCPL$ can then be directly compared to any sigma-based threshold $sepTh(k,a)$ to obtain a boolean value of whether the input is identified as OOD or not. If the $normArchCPL$ is larger than the threshold, then the input is considered as out-of-distribution, as per equation 4.9a. Conversely, if it is smaller or equal to the threshold the input is considered as in-distribution, as per equation 4.9b.

$$OOD(y,k,a) = normArchCPL(y,k,a) > sepTh(k,a) \tag{4.9a}$$

$$InD(y,k,a) = \neg OOD(y,k,a) = normArchCPL(y,k,a) <= sepTh(k,a) \tag{4.9b}$$

We then introduce the following definitions, for the arbitrary set of inputs Y:

- $numCorrectOOD(Y,k,a)$: the number of identified out-of-distribution cases that the model correctly labels
- $numIncorrectOOD(Y,k,a)$: the number of identified out-of-distribution cases that the model incorrectly labels
- $numCorrectInD(Y,k,a)$: the number of identified in-distribution cases that the model correctly labels
- $numIncorrectInD(Y,k,a)$: the number of identified in-distribution cases that the model incorrectly labels

With these values, it is possible to get the total number of images that would pose no threat to the model when filtering against OOD, as it is the sum of incorrectly predicted Out-Of-

Distribution images and all correctly predicted images:

$$numNoHarm(Y,k,a) = \\ numIncorrectOOD(Y,k,a) + numCorrectOOD(Y,k,a) + numCorrectInD(Y,k,a) \tag{4.10}$$

Once it is clear which images are considered OOD and which are InD, it is possible to compute a *No-Harm%* ratio that describes the ratio of aggressive test cases identified as OOD and non-harmful correctly predicted images, with regards to the total number of images considered. This ratio is presented in Equation 4.11, and is the measure that we use to present our results in section 5. Similarly, we could also compute the ratio of OOD images as in Equation 4.12 and the ratio of InD images as in Equation 4.13.

$$\textit{No-Harm\%} = \frac{numNoHarm(Y,k,a)}{\mid Y \mid} \tag{4.11}$$

$$OOD\% = \frac{numCorrectOOD(Y,k,a) + numIncorrectOOD(Y,k,a)}{\mid Y \mid} \tag{4.12}$$

$$InD\% = \frac{numCorrectInD(Y,k,a) + numIncorrectInD(Y,k,a)}{\mid Y \mid} = 1 - OOD\% \tag{4.13}$$

We also consider a measure that simply represents the ratio of incorrectly predicted images found below the threshold and considered as in-distribution, with regards to the total number of images. This measure is defined in Equation 4.14.

$$\textit{Misclassified\%} = \frac{numIncorrectInD(Y,k,a)}{\mid Y \mid} \tag{4.14}$$

Similarly, we obtain a measure that is the ratio of correctly predicted in-distribution images, with regards to the total number of images, defined in Equation 4.15. This measure would be equivalent to the subtraction of the $OOD\%$ value from the *No-Harm%* value.

$$\textit{CorrClassified\%} = \frac{numCorrectInD(Y,k,a)}{\mid Y \mid} = \textit{No-Harm\%} - OOD\% \tag{4.15}$$

4.6 Research Questions

The following research questions were used to guide our experiments and to frame the conclusions that were drawn from the results. We address the answers to these questions when presenting and discussing the results.

- RQ1: What are the computational profile distributions of affine and adversarial cases compared to those of the training set?
- RQ2: What is the degree of separation between affine and adversarial cases with respect to the training set?
- RQ3: What effect does changing the threshold have when separating between adversarial/affine inputs and training data?
- RQ4: How can we identify aggressive test cases with regards to their distributions in comparison with training data?

4.7 Experiment Setup

Our experiments were designed to allow comparison between a CNN's training images and other cases that were not taken from the same distribution of the training set, by generating synthetic images from the training images using affine transformations and adversarial attacks.

More specifically, we use random pixels as error control-data for initial experiments, as well as different intervals of affine transformations (translation, corner rotation, center rotation) and adversarial images generated using Fast Gradient [6], Carlini & Wagner [9], Jacobian Saliency Maps [7], and DeepFool [8].

Successful adversarial attacks and affine transformations can be considered as aggressive test cases against CNN models and it can be very useful to identify those cases that are further away in distribution from the training set.

4.7.1 Architectures

We perform our experiments on a range of convolutional neural networks, to eliminate the bias that we would have if we had only one single architecture, and to look for any trends between different architecture structures and/or between different instances of the same architecture

structure. Ten different structures were used throughout our experiments, with identifiers v0-v9 and their details can be found in Appendix B.

We began with introductory experiments using one instance of v0, which was based on a Tensorflow variant of LeNet [56]. In this thesis, experiments for Section 4.3 were performed using this instance. As for the following Sections 4.4 and 4.5, the experiments were carried out using all ten structures v0-v9. Furthermore, we also perform these experiments on five different instances of the v0 structure, because of its highest cumulative recognition precision on training and test sets among the different structures and due to the fact that it is based on LeNet [56].

Each instance was obtained by training the same structure on a different set of training images. To obtain each training set, we start with the original set of MNIST-Fashion images, and randomly sample 60000 images with a different random seed. The remaining 10000 images become the test set. This allows for five different training sets that may intersect with one another differently, later used to train the five instances of v0. Each instance was trained starting with a different seed for the initial randomized weights, but the rest of the hyperparameters such as learning rate, epochs, and mini-batch size remain the same, so that the only difference between the architectures is the data that they have been trained on.

4.7.2 Layers

In our experiments, though it is possible obtain activations for all layers, we considered the neurons only in the penultimate layer. Some works such as [23] may discourage this decision, as their experiments demonstrate that the last layer may often be the one to turn a correct prediction to an incorrect one. However, their arguments and experiments are not verified against any sort of affine transformations or adversarial inputs, and thus do not give sufficient evidence to conclude that this discovery is applicable in our case. In principle, other works more relevant to our topic align well with our use of this second-to-last layer to improve computational efficiency, since we consider less neurons and have higher statistical independence from previous layers, in contrast with analyzing the whole network [1, 12, 20]. Furthermore, recent research [1] suggests that layer sensitivity varies and is more effective on the final layers of a network than on the initial layers for the MNIST dataset.

4.7.3 Image Datasets

Our experiments take five image sets into account: train, test, random, affine transformations and adversarial attacks. The affine transformations and adversarial attacks are synthetic

cases created for each architecture from its training images. More details on the creation of these images can be found below.

Train and Test Images

For these sets, we decided to use the MNIST Fashion library of 70000 images [2], and split them into 60000 training images and 10000 testing images. As mentioned, we repeat the split for each of the five architecture instances so that we end up with five sets of train and test images. In contrast, the ten architecture structures were trained using the same sets of train and test images.

Random Images

As some sort of control data, 60000 images of the same MNIST-Fashion size (28x28) were generated by creating completely random grayscale values for each pixel, using a uniform distribution. Each row's pixels were randomly generated in sequence, for each image, without re-initializing the random generator between rows or between images. Although these images are best classified by the network into some classes, we assumed that all these predictions be errors. We use these random images in introductory experiments of Section 4.3.

Affine Transformed Images

For each architecture, we applied three different types of affine transformations on the train images. The transformations did not need any prior knowledge of the model, but only knowledge of the training and testing images used. For each transformation, we started with the original train image and incrementally increased the parameter values of the transformation. Each increment represents a different interval of transformation.

We describe below the different types of transformations used.

- Translation: This transformation type consists of a translation along the x and y axes. Each interval parameter amount represents the pixel translation along both axes.
- Corner Rotation: This transformation type consists of a rotation of the image around the pixel in the upper left corner of the image. Each interval parameter amount represents the degree of rotation. Note that since we are rotating along the corner, we eventually reach a point where the image is completely out of the visible frame.
- Center Rotation: This transformation type consists of a rotation of the image around its center. Each interval parameter amount represents the degree of rotation.

Adversarial Images

Starting from the correctly predicted images of each training set, we use IBM's Adversarial Robustness Toolbox [57] to create adversarial images with Fast Gradient Sign [6], Jacobian Saliency Map [7], and DeepFool [8]. We also use CleverHans [58] to create adversarial images with Carlini & Wagner [9]. The parameters used for each attack are kept constant throughout our experiments, and were chosen based on other examples used in other works, such as the epsilon of 0.6 that we used for the Fast Gradient Sign images. For all the attacks we performed, the attack crafted the images to attack a particular architecture, and thus knowledge of the model is needed. In most of our attacks, the adversarial images were crafted without a target. In other words, they were created simply with the goal of making the network fail, without any emphasis on classes. However, in the case of the Jacobian Saliency Maps attack, we input a target class t, for the goal of modifying the original image specifically to incorrectly be classified as an image from class t.

CHAPTER 5 RESULTS AND DISCUSSION

5.1 Computing and Visualizing CPL

In this section, we discuss the application of our measure of surprise CPL as presented in Equation 4.4. We apply it to our experimental datasets described in Section 4.7. We present our results in the form of example visualizations demonstrating our method, along with a discussion.

5.1.1 Per-Class Overview

As described in section 4, we use the Bayesian likelihood computation to obtain a measure that we refer to as CPL. This measure represents how probable it is that, during classification of an image, the neuron activation levels belong to the distribution of the original training set. In other words, this measure represents the probability of a network layer's activation levels with respect to the distributions observed and estimated during training for the given layer and the predicted class.

Several recent works [10] [11] [12] attempt the use of neuron activation levels from training and adversarial inputs and train a secondary DNN as a defensive measure for robustness or based on the nearest neighbors method computed using locality-sensitive hashing. We share the extraction of activation levels with the approaches, but we model them with non-parametric statistical Bayesian likelihood of an input given a class, without the need of secondary training. Some other approaches [13] [17] don't use a secondary classifier but use different methods such as Kernel Density Estimation.

For each experimental dataset mentioned in 4.7, we compute the measure of CPL of the images, per class. To be more specific: for each class, we take the images that originally belong to the class in question but are incorrectly predicted by the network, and compute their CPL from the training set for that class.

To better visualize the results we obtain and to identify any patterns or trends, we create a cartesian plot with the CPL values for each class. We do this for random adversarial attacks (including random images) and affine transformations. Examples of these plots can be seen in Figures 5.1 and 5.2.

In Figure 5.1, we see four different example plots, each representing a different class. Along the x axis, we see the different image sets in question. In these plots, each point represents

a single image.

For each class, we have columns containing the train and test images of the class in question, along with five proceeding columns containing, in order, the randomly generated images (rnd), the Fast Gradient images (fgrad), the Carlini & Wagner images (cw), the Deepfool images (df), and the Jacobian Saliency Maps Images (jsma_targeted). For clear and visible identification, the training set is in dark blue, the test set is in cyan, and the adversarial attacks are in red, including the randomly generated images. The y axis, labeled as *dist*, refers to the measure of surprise CPL that a particular image has from that class's training set. The larger the CPL value for a particular image, the further that image is from the training distribution. The position of each image within the columns along the x axis is of no significance; it was chosen randomly when generating the plots, to avoid all the points being placed on top of one another.

Each plot represents a class, where the train and test columns contain the correctly predicted images from the training and testing sets for that class, respectively. The adversarial columns however, contain the incorrectly predicted images that have that class as the true label. The only exception is the set of randomly generated images, where an image does not have a true class. For these images, we place these images on the plot of the class corresponding to the class that the network predicted them to belong to.

For all the attacks we performed, except for the randomly generated images, the attack crafted the images to attack a particular architecture, and thus knowledge of the model is needed. In most of our attacks, the adversarial images were crafted without a target. In other words, they were created simply with the goal of making the network fail, without any emphasis on classes. However, in the case of the Jacobian Saliency Maps attack, we input a target class t, with the goal of modifying the original image specifically to incorrectly be classified as an image from class t. For this attack, we repeat the experiments 10 times, each time with the target class as a different class.

In Figure 5.1, the plots are shown for classes 1, 3, 5, and 9. These were chosen purely for demonstrative purposes, to cover various different behaviours observed. The first thing to notice is that the *jsma_targeted* columns contain many more images than the other columns. This is to be expected, as for each class c, the *jsma_targeted* column contains all the images from class c that are incorrectly predicted, and includes the 10 different target class possibilities.

When comparing between the different classes, we can clearly see that some classes seem to produce many more images with higher CPLs than others. For example, in Class 1, we see many images with CPLs above 3000 as well as above 4000. This could signify a high

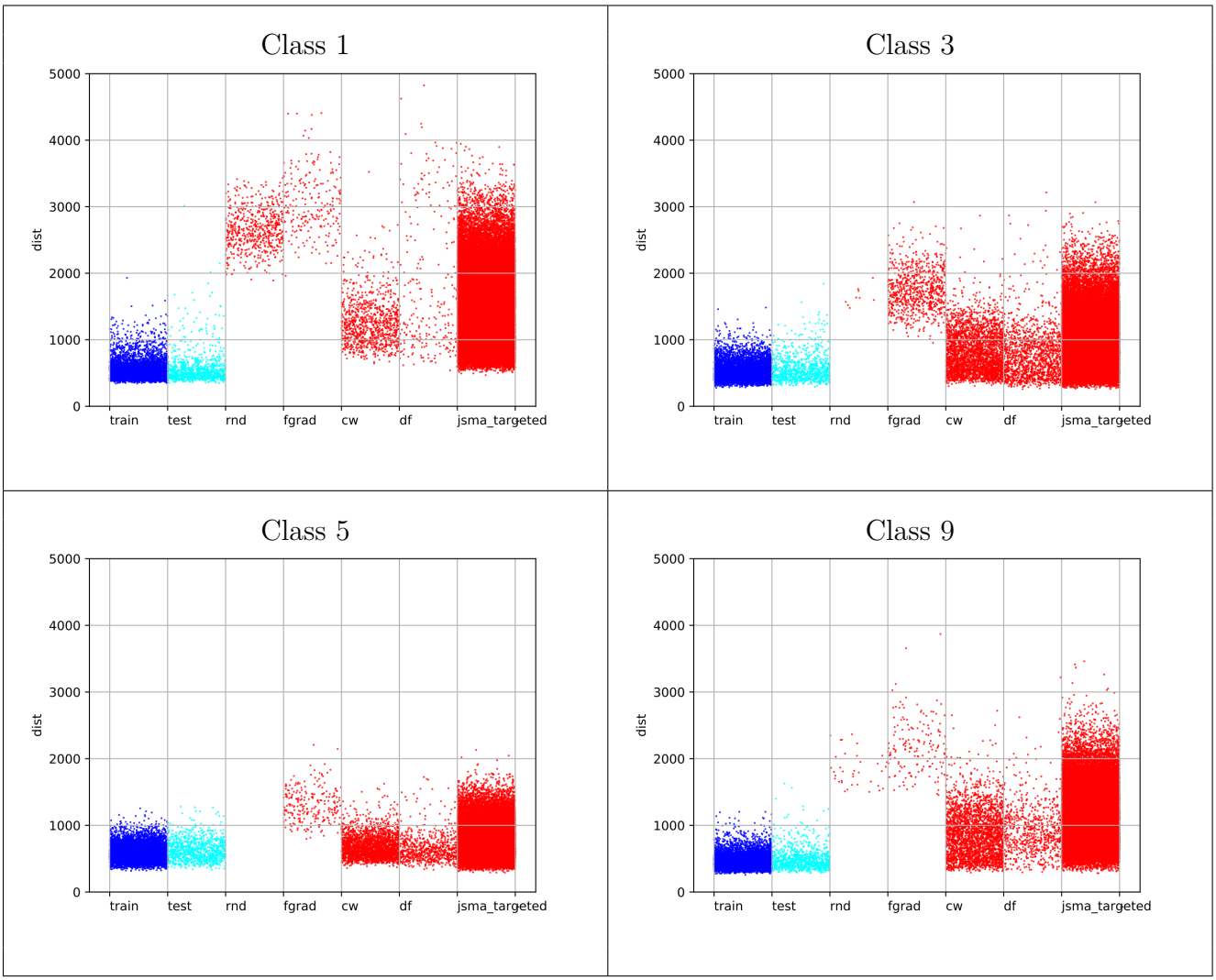

Figure 5.1 CPL Plots - Incorrectly Predicted Adversarial Images

robustness for Class 1, as it seems to recognize how different the adversarial images are from the training set. Class 9 and Class 3 seem to have a lower recognition of these differences, as the overall CPLs tend to be lower than those of Class 1. Finally, we can see that Class five seems to have the lowest recognition of these images, as all the images have much lower CPLs than those of the other classes, with all values below 2500. In fact, we chose Class 5 as one of the example plots precisely because it seems to be the worst at assigning larger CPLs to the adversarial images.

It is also interesting to notice that the randomly generated images seem to collect in certain classes, such as Class 1. In contrast, not a single randomly generated image was predicted to be in Class 5. Some seem to end up in Class 9, though it is a very small amount compared to the total amount of randomly generated images. With some further investigation, these types of trends could be useful in characterizing the different classes and learning more about their behavior.

In all cases, it is important to note that the testing images seem to, for the most part, be within distribution of the training set. This is to be expected, as the training and testing images were all sampled from the same original set of images.

Similarly to the adversarial attacks, we also create the same type of plot for transformed images. We see examples of these plots in Figure 5.2, where the incorrectly predicted images of Class 0, 4, 6, and 9 are shown for translated images. For each example, we visualize the images after they have been translated at different interval units, namely 1, 3, 5, 7, 9, and 11. Once again, the y axis labeled as *dist* corresponds to the measure of CPL and the x axis corresponds to the different image sets. The first two columns contain the train and test images in dark blue and cyan, respectively. The next six columns show, in red, the transformed images for each interval.

The first thing to notice is the clear pattern in CPLs as we increase the interval of affine transformation. After only one interval of transformation, the CPLs seem indistinguishable from those of the training set. But after looking at the next interval, we see right away that in general the CPLs start to move upwards on the plot. The more we translate the image, the further the CPLs move away from the training class for the class in question. We eventually reach a point where almost all the CPLs are out of the distribution of the training set.

Similarly to the adversarial attack plots in Figure 5.1, we can see that the various classes demonstrate different behaviors. In this case, Class 0 presents CPLs that are closer together for each interval, whereas Class 9 shows CPLs that are more spread out across the y axis for the different columns. At interval 11, we also see that Class 0 still has several images that overlap with the training set, while Class 4 and Class 9 contain a very minimal amount of

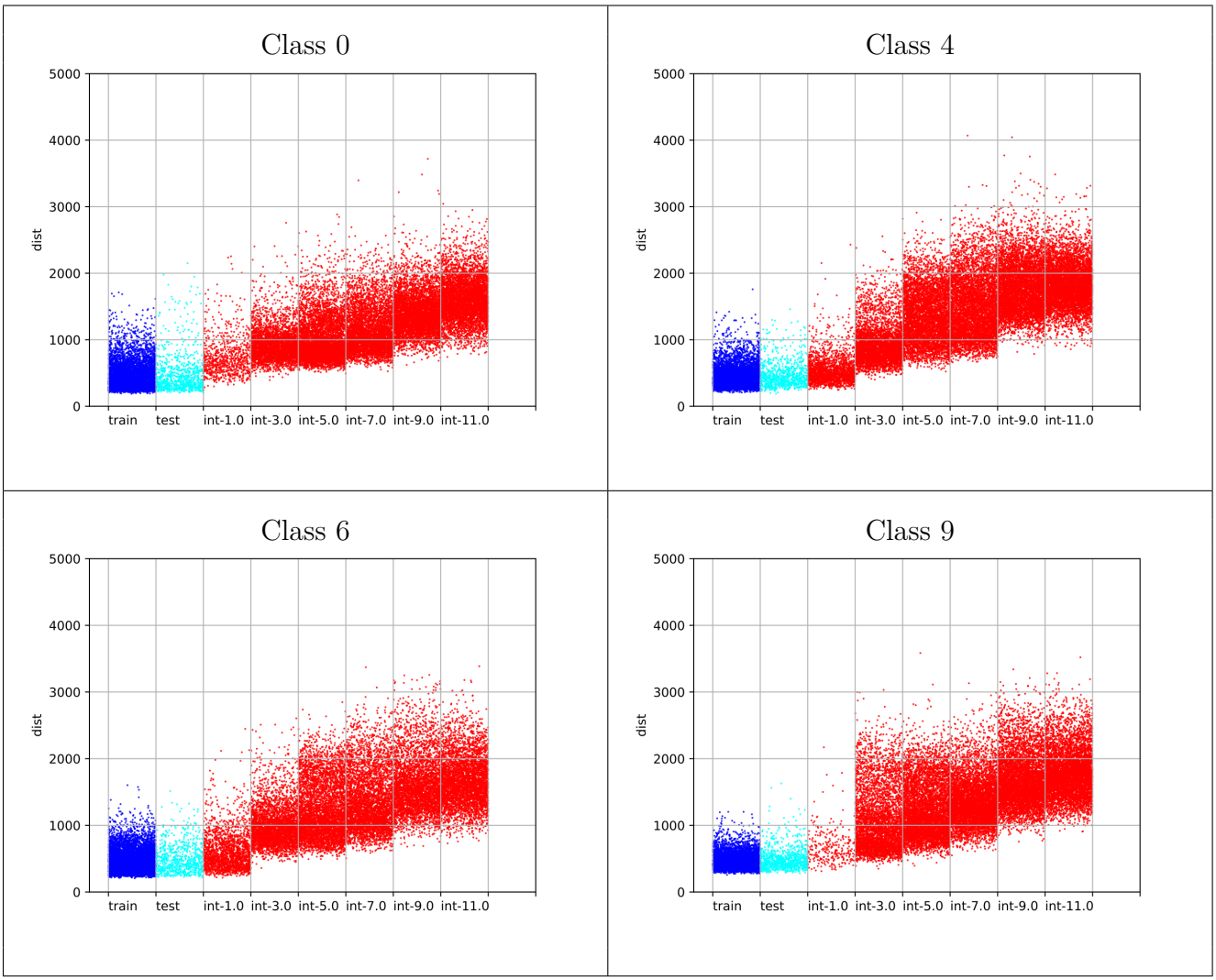

Figure 5.2 CPL Plots - Incorrectly Predicted Translated Images

overlap.

It is also interesting to see that after only one interval of translation, relatively very few images get classified as Class 9, whereas many are classified as Class 4 or Class 6. This could be related to the observations we made about Class 9 for the adversarial attacks, and once again further investigation is warranted to analyze this behaviour.

5.1.2 Statistical Comparison Tests

In this section, we present several two by two 2-sample statistical tests, to compare the different sets of images and numerically get a sense for how similar they may be, based on the distribution measures described and visualized in Section 5.1.1. The tests we performed are Anderson-Darling, Cliff's Delta, Epps-Singleton, Kolmogorov-Smirnov, and Wilcoxon Rank-Sum test for continuous and discrete variables. Most of the tests were computed using the python SciPy library for scientific computing [59], with the exception of Cliff's Delta which was taken from Ernst's Python Implementation [60] derived from Torchiano's in R [61].

In our tests, we statistically compared each class separately, allowing the independent assessment of different classes.

Table 5.1 Statistical 2-Sample Tests: training / test

	Anderson-Darling		Cliff's		Epps-Singleton		Kolmogorov-Smirnov		Wilcoxon Rank-Sum	
class	stat	sig-level	delta	size	stat	p-value	stat	p-value	stat	p-value
0	3.82	0.009	0.0158	negligible	30.89	0.000	0.04	0.134	0.76	0.447
1	5.47	0.002	-0.0388	negligible	14.13	0.007	0.05	0.057	-1.95	0.051
2	1.48	0.080	-0.0269	negligible	11.44	0.022	0.04	0.169	-1.29	0.199
3	17.98	0.001	-0.0809	negligible	40.94	0.000	0.08	0.000	-3.96	0.000
4	4.27	0.006	-0.0474	negligible	16.09	0.003	0.05	0.088	-2.27	0.023
5	36.26	0.001	-0.1392	negligible	59.45	0.000	0.12	0.000	-7.02	0.000
6	4.53	0.005	0.0031	negligible	25.27	0.000	0.05	0.047	0.14	0.890
7	3.91	0.009	-0.0428	negligible	9.21	0.056	0.05	0.034	-2.16	0.031
8	13.86	0.001	-0.0820	negligible	24.96	0.000	0.08	0.000	-4.14	0.000
9	12.06	0.001	-0.0698	negligible	28.11	0.000	0.07	0.001	-3.45	0.001

Table 5.2 Statistical 2-Sample Tests: training / random

	Anderson-Darling		Cliff's		Epps-Singleton		Kolmogorov-Smirnov		Wilcoxon Rank-Sum	
class	stat	sig-level	delta	size	stat	p-value	stat	p-value	stat	p-value
0	5,627.63	0.001	-1.0000	large	77,212,398.89	0.000	1.00	0.000	-91.11	0.000
1	1,673.03	0.001	-1.0000	large	3,828.08	0.000	1.00	0.000	-40.48	0.000
2	1,019.37	0.001	-1.0000	large	639.36	0.000	1.00	0.000	-28.60	0.000
3	69.34	0.001	-1.0000	large	229.90	0.000	1.00	0.000	-5.19	0.000
4	5,243.65	0.001	-1.0000	large	118,389,283.14	0.000	1.00	0.000	-87.72	0.000
6	14,796.74	0.001	-0.9943	large	318,458.42	0.000	0.97	0.000	-124.12	0.000
8	7,561.29	0.001	-0.9931	large	141,370.12	0.000	0.96	0.000	-104.61	0.000
9	88.52	0.001	-1.0000	large	16.67	0.002	1.00	0.000	-5.99	0.000

Table 5.3 Statistical 2-Sample Tests: training / rotation

	Anderson-Darling		Cliff's		Epps-Singleton		Kolmogorov-Smirnov		Wilcoxon Rank-Sum	
class	stat	sig-level	delta	size	stat	p-value	stat	p-value	stat	p-value
0	3,056.99	0.001	-0.8748	large	11,339.27	0.000	0.73	0.000	-65.74	0.000
1	1,626.62	0.001	-0.9944	large	968.99	0.000	0.95	0.000	-40.19	0.000
2	238.08	0.001	-0.4090	medium	274.16	0.000	0.29	0.000	-18.23	0.000
3	1,454.17	0.001	-0.7465	large	2,987.04	0.000	0.63	0.000	-42.39	0.000
4	268.95	0.001	-0.4549	medium	184.10	0.000	0.42	0.000	-14.03	0.000
5	68.87	0.001	-0.2562	small	112.08	0.000	0.17	0.000	-9.38	0.000
6	1,099.99	0.001	-0.6745	large	1,998.38	0.000	0.55	0.000	-37.44	0.000
7	2,209.32	0.001	-0.9542	large	7,651.01	0.000	0.86	0.000	-51.63	0.000
8	816.47	0.001	-0.6740	large	1,624.30	0.000	0.53	0.000	-32.81	0.000
9	1,029.63	0.001	-0.9337	large	620.31	0.000	0.86	0.000	-30.28	0.000

Table 5.4 Statistical 2-Sample Tests: training / fgrad

	Anderson-Darling		Cliff's		Epps-Singleton		Kolmogorov-Smirnov		Wilcoxon Rank-Sum	
class	stat	sig-level	delta	size	stat	p-value	stat	p-value	stat	p-value
0	1,246.81	0.001	-0.9988	large	2,815.36	0.000	0.98	0.000	-33.02	0.000
1	355.59	0.001	-1.0000	large	112.30	0.000	1.00	0.000	-14.10	0.000
2	507.50	0.001	-0.9994	large	138.51	0.000	0.99	0.000	-17.91	0.000
3	605.78	0.001	-0.9997	large	128.13	0.000	0.99	0.000	-20.04	0.000
4	1,185.94	0.001	-0.9986	large	527.40	0.000	0.98	0.000	-31.94	0.000
5	267.43	0.001	-0.9924	large	1,629.95	0.000	0.96	0.000	-12.46	0.000
6	6,594.35	0.001	-0.9965	large	133,565.21	0.000	0.97	0.000	-98.52	0.000
7	34.34	0.001	-1.0000	large	68,942.21	0.000	1.00	0.001	-3.46	0.001
8	2,828.45	0.001	-0.9949	large	75.74	0.000	0.96	0.000	-59.27	0.000
9	183.02	0.001	-1.0000	large	0.00	0.000	1.00	0.000	-9.30	0.000

We first compared the training and test image sets (Table 5.1). As expected, the tests produce results that show that the images are not necessarily from different distributions. The sig- and p- values are relatively larger in all tests for many classes, and the Cliff's delta is negligible for all the classes.

We then compared the training set with the set of random images (Table 5.2). In contrast to the train/test comparison, this comparison shows that the distributions of the images are quite different, as expected since the random images have a much larger CPL from the training set than the test images do. The sig- and p- values are usually 0.000, with the exception of the Anderson-Darling test, which caps the sig-values at 0.001 once they go below that value. So in reality, they are probably have values even smaller than 0.001, with the statistics being very large values in comparison to the train/test statistical tests. Furthermore, we can also see that the Cliff's delta is always large.

Next, we took a set of images where each image is taken from the training set and is rotated just until the point where the network's prediction for it turns incorrect. We compared these rotated images with the original training set (Table 5.3). Though not as extreme as the random images, the results still indicate that the distribution of these rotated images is quite different from that of the original training images. Similar to the random images, the sig- and

p- values are always 0.000 except for Anderson-Darling where they are capped at 0.001, while the statistics are very large values in comparison to the train/test statistical tests. In this case, the Cliff's delta is often large, except for rotated classes 2 and 4 that show a medium delta and class 5 for which the delta is small, showing that the transformations likelihood lie in an extended range with respect to training, tests, and noise.

Lastly, to statistically compare some adversarial images with the training set, we perform the same statistical tests for the training set and a set of training images that were adversarially attacked using the Fast Gradient method (Table 5.4). Once again, all sig- and p- values are smaller or equal to 0.001, and the statistic values are relatively quite large. Cliff's delta also shows a large size for all classes, emphasizing that the distributions of these two images are in fact quite different from one another.

It is worth mentioning that Cliff's delta is a non parametric test [62]. For Cliff's delta d, the magnitude is assessed using thresholds on d and is classified as follows: "negligible", if $d < 0.147$; "small", if $d < 0.33$; "medium", if $d < 0.474$; and "large" otherwise [63]. Furthermore, the alpha level can be considered as the possibility of making a type one error, which refers to incorrectly declaring a difference, effect or relationship to be true due to chance. We can in principle set the alpha level for one test at 0.05. With this alpha level, the probability of the test showing "something" when there is actually nothing should not occur in more than one in twenty statistical tests. Since we carried out more than one statistical test, the probability of finding at least one statistically significant test due to chance and incorrectly declaring a difference or relationship to be true, increases.

We hypothesize that there is a substantial difference between the training images and the random, rotated, and fast gradient images for the different classes. Since we formulated one hypotheses per class (10 in total), we should correct our statistical tests accordingly, using the Bonferroni correction, and assume an alpha value of 0.005, instead of 0.05.

With this in mind, we point out that comparing the training set with random, rotated, and fast gradient images produces sig- and p- values that are much smaller to the Bonferroni correction threshold of 0.005.

5.2 Separation Using CPL

In this section, we use the measure of CPL to go further in detail. We visualize not only each true class of images, but we break them further down into their predicted classes of images, to look for any patterns or trends. Using these groups of images, we show the results for two different ways of comparing the images with the training set: a per-set best

comparison between the input images and the training set of their predicted class, and a per-input sigma-order threshold comparison between a single input and the training set of its true class.

5.2.1 Per-Set Best Comparison

The first method of separation we attempted is what we refer to as the best separation between the two distributions. We assume full knowledge of the whole set of images being considered to compute this separation, and attempt to place a linear threshold-based separator using that knowledge. For example, if we are computing the best separation between a set of fast gradient images and their training corresponding training images, we take all of the fast gradient and training images into consideration for the computation.

As described in Section 4.3, we compute the best separation by finding the appropriate CPL cutpoint value and placing a linear threshold at that value. This way, we can attempt to separate between the training set and the set of images in question.

We do this for each image set, for each true class, and for each best (predicted) class. Once again, we include some example visualizations to better understand the process. Figure 5.3 shows plots for classes 1, 3, 5, and 9. These plots are similar to those previously shown in Figure 5.1, but a key difference is that each visualization is based on a particular set of images. In this case, Figure 5.3 shows the visualizations specifically for Jacobian Salience Maps Attack (JSMA). Here we see the same train and test columns per-class, but the next 10 columns represent the predictions the neural network has made for images that were originally in the class. For example, any point in the best3 column of the Class 1 plot signifies an image that is truly a class 1 image, but predicted to be class 3 by the network after the adversarial attack. With the same logic, images that are in the Class 1 plot and best1 column are images that were correctly predicted even after adversarially attacking them.

We apply the best separation method for each column, to separate the images from the training set of the class they are predicted to be in. To better visualize the results, we color-code the points based on the three categories they belong to: green points are images that are below the separation line but correctly predicted by the network, red points are images that are below the separation but incorrectly predicted by the network, and the grey points are images that fall above the separation line regardless of whether they were correctly or incorrectly predicted.

This way of visualizing allowed us to not only compare between images' true classes, but also between the predicted classes. The differences in color also allow us to clearly see where the

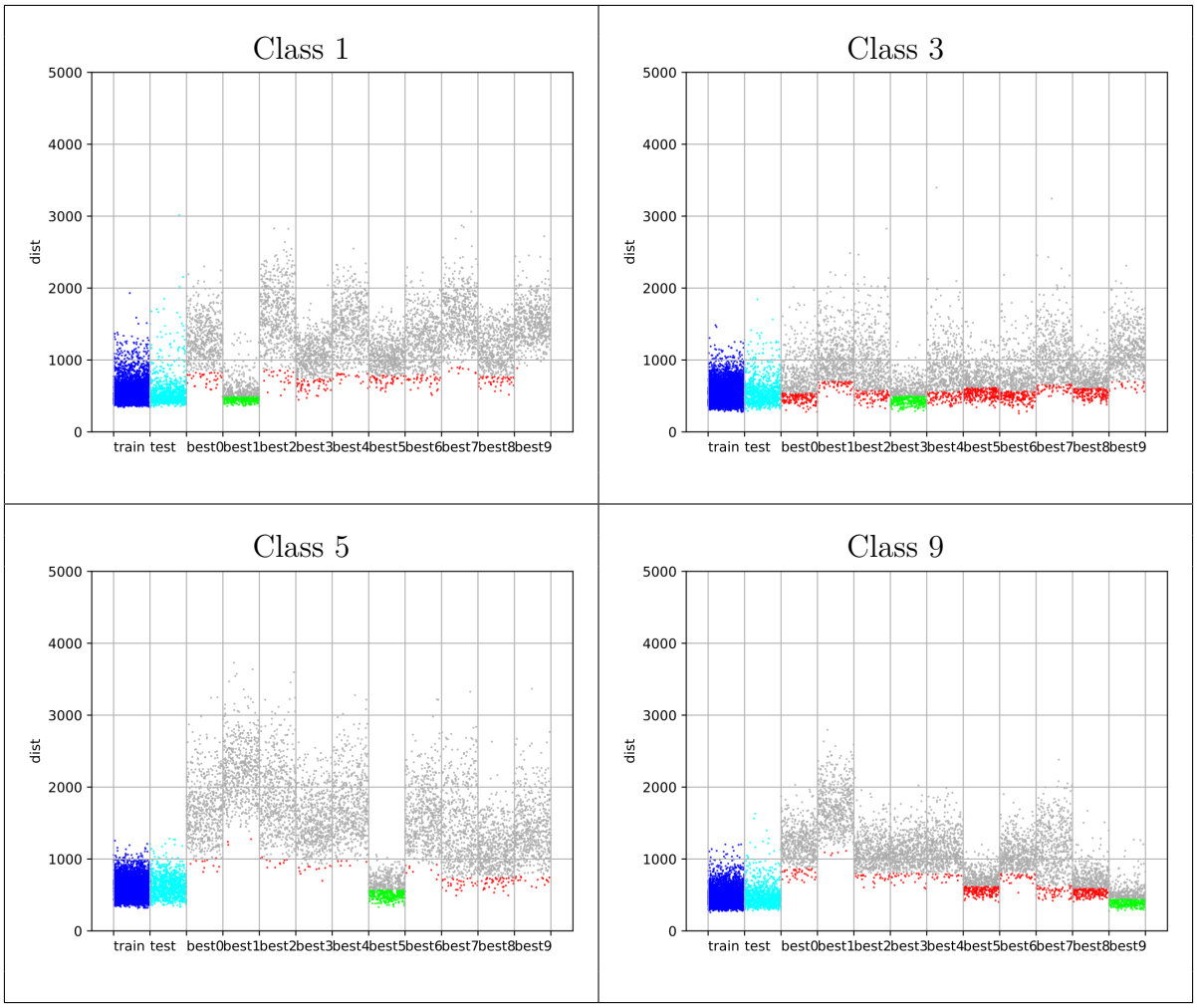

Figure 5.3 Best-Separation CPL Plots for JSMA: Classes 1, 3, 5, 9

linear threshold for separation was placed in each column. In Figure 5.3, we can clearly see in all cases that the separation for the correctly predicted class (green) is always lower than the separations for all the other incorrectly predicted classes (red). This usually means we are separating with less precision in the training set, and implies that the distributions of the correctly predicted images are much closer to the training set than those of the incorrectly predicted images, as they are harder to separate with good precision.

As expected, we see that the columns with generally higher CPLs (such as Class 5 - best0) will have higher separation thresholds than those with lower CPLs (such as Class 5 - best8). In general, the higher the separation threshold, the better the separation precision for the training set. However this is not always the case for the precision of the adversarial images.

Using the same plot formats, we can visualize the affine transformations as well. Figure 5.4 presents the CPL plots for Class 0 images that were translated at various intervals. Similarly to the adversarial images, the plots show images as points indicating their CPLs from the training set, and are organized in columns of their predicted classes. One difference is that we create a plot for each true class as well as for each interval of transformation. For lack of redundancy and to save space, we include examples of Intervals 3, 7, 11, and 15. These examples were chosen purely for demonstrative reasons, as we can see a few key trends that we would like to highlight. Nevertheless, the same type of plot can be created for all classes as well as all intervals of transformation.

In Figure 5.4, the first thing to notice is that as we increase the parameters of transformation, the CPLs of the images tend to grow larger. In other words, they start to move upwards in the plots, and start to move out away from the training distributions. When the interval is 3, we see most of the points have CPLs under 1000, whereas when the interval is 15, there are rarely any images with a CPL below 1000. As we increase the interval of transformation, the computed separation threshold also tends to increase. For example, the best6 column seen in Figure 5.4 places the separation threshold at a CPL of 673 when the interval is 3, 912 when the interval is 7, 1138 when the interval is 11, and 1160 when the interval is 15. In these cases, the separation precision of the adversarial images, in other words the ratio of adversarial images that are found above the separation threshold, also increases. To be more precise, the separation precision of the best6 column is 84.6% when the interval is 3, 97.4% when the interval is 7, 99.5% when the interval is 11, and 99.6% when the interval is 15.

We also see certain classes that most images tend to pool in, regardless of whether the images are correctly or incorrectly predicted. In our example in Figure 5.4, most images tend to be predicted as class 0, class 3, class 6, or class 9. In contrast, we rarely see any images get sorted into class 1, class 2, class 5, class 7, and class 9. This interesting pattern of pooling

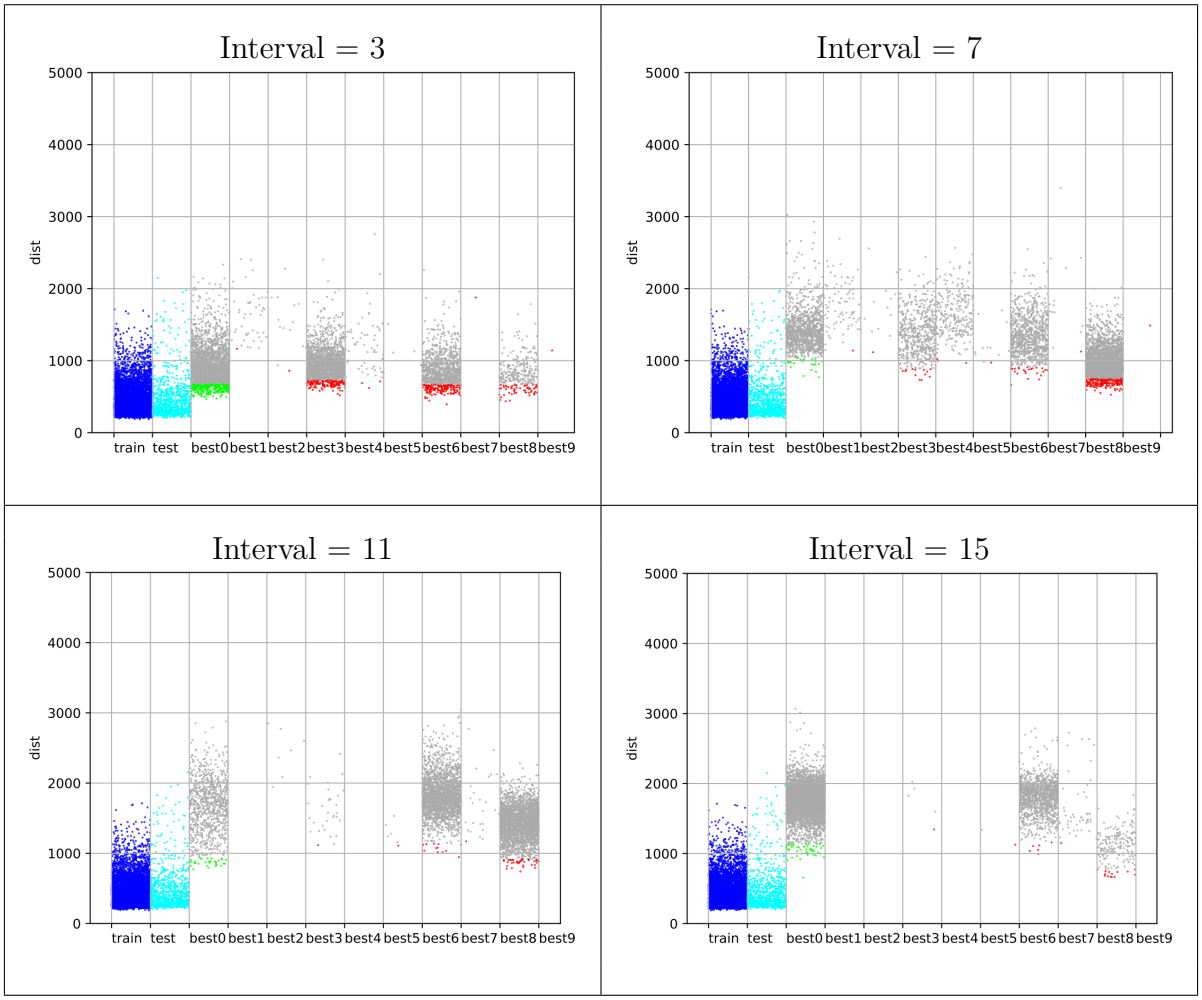

Figure 5.4 Best-Separation CPL Plots for Translation: Architecture v0.0, Class 0

in certain classes happened across most of the adversarial attacks and affine transformations that we studied, across various classes.

To take a more quantitative approach, Tables 5.5, 5.6, 5.7, and 5.8 are presented to show the exact numerical values of best separation for JSMA for each combination of true class and best class. In these tables, the Separation Point column gives the value of CPL at which the separation threshold was placed to separate between the different adversarial attacks and the training set. The next column, Train Separation, gives the ratio of training images of the particular class that are found under the separation threshold, relative to the total number of training images considered for that class. Similarly, the JSMA Separation column gives the ratio of JSMA images that lie above the separation threshold.

The format of these tables allows to compare the separation precisions of each true class and each best class. We can link them back to Figure 5.3 to verify the patterns we previously noticed from the plots. Indeed, we see that in all cases the separation for the correctly predicted class is lower than the separations for all the other incorrectly predicted classes. For example, when True Class is 0 and Best Class is 0, we see that the separation point is below 400, and the Train and JMSA separation precisions are quite low at 50% each. These correctly predicted images were considered to be very similar to the training set, and were more difficult to separate. Meanwhile, when True Class is 0 but Best Class is any class other than 0, we see much higher CPLs and separation precisions. For example, when Best Class is 1, the separation point is over 700 and the separation precisions are over 90% for both sets. The same pattern holds for all combinations of True Class and Best Class. Thus, we can reasonably say that when an image is correctly predicted, it usually presents a computational profile that is likely to be similar to the training distribution.

The choice of these tables is purely demonstrative, and the same tables can be computed for all experiments. The rest of the tables for all adversarial attacks can be found in Section A.

Table 5.5 Linear Best Separability - Jacobian Saliency Map - Classes 0, 1, 2

True Class	Best Class	Separation Point	Train Separation	JSMA Separation
0	0	397.5314	0.5007	0.5007
	1	703.2719	0.9032	0.9033
	2	463.6105	0.6715	0.6716
	3	537.9626	0.6059	0.606
	4	525.7214	0.7604	0.7604
	5	579.1344	0.5123	0.5123
	6	465.6688	0.5089	0.509
	7	633.958	0.8789	0.879
	8	564.5886	0.514	0.5141
	9	615.9107	0.9	0.9001
1	0	831.7829	0.9355	0.9355
	1	495.5164	0.5029	0.5031
	2	860.8222	0.9682	0.9682
	3	742.3261	0.9178	0.918
	4	813.7388	0.9647	0.9648
	5	784.5104	0.912	0.9121
	6	803.4066	0.9411	0.9411
	7	931.6438	0.9828	0.983
	8	778.2969	0.9358	0.9359
	9	919.686	0.9949	0.995
2	0	528.217	0.7211	0.7212
	1	774.9406	0.937	0.937
	2	390.5676	0.5005	0.5006
	3	592.9788	0.7402	0.7403
	4	459.1883	0.623	0.6231
	5	599.5464	0.5716	0.5716
	6	462.9844	0.5029	0.503
	7	689.631	0.9144	0.9145
	8	578.8158	0.5685	0.5687
	9	616.8163	0.9002	0.9003

Table 5.6 Linear Best Separability - Jacobian Saliency Map - Classes 3, 4, 5

True Class	Best Class	Separation Point	Train Separation	JSMA Separation
3	0	542.0361	0.7385	0.7385
	1	706.7524	0.9048	0.905
	2	576.6976	0.8375	0.8375
	3	503.5266	0.5007	0.5008
	4	563.2345	0.8152	0.8152
	5	616.7371	0.6156	0.6157
	6	565.6911	0.6998	0.6999
	7	659.3092	0.8959	0.896
	8	609.7839	0.6728	0.673
	9	703.4379	0.9572	0.9572
4	0	559.6922	0.757	0.757
	1	756.7171	0.9292	0.9292
	2	461.4536	0.6669	0.6671
	3	562.5864	0.6699	0.6701
	4	418.4148	0.4999	0.5003
	5	602.9499	0.5796	0.5798
	6	457.222	0.4892	0.4892
	7	668.1598	0.9011	0.9012
	8	559.6019	0.4943	0.4945
	9	600.4999	0.8838	0.884
5	0	1011.8756	0.9764	0.9765
	1	1276.6832	0.9977	0.9978
	2	1032.5779	0.9883	0.9885
	3	912.4925	0.983	0.9831
	4	973.5321	0.9883	0.9885
	5	575.2078	0.5013	0.5016
	6	931.3186	0.9779	0.978
	7	735.8141	0.9379	0.9381
	8	747.5837	0.9145	0.9146
	9	755.3217	0.9739	0.974

Table 5.7 Linear Best Separability - Jacobian Saliency Map - Classes 6, 7

True Class	Best Class	Separation Point	Train Separation	JSMA Separation
6	0	523.5387	0.7164	0.7165
	1	744.9751	0.924	0.9241
	2	474.1108	0.6922	0.6923
	3	584.34	0.7238	0.724
	4	487.0512	0.6824	0.6825
	5	609.8284	0.5966	0.5966
	6	461.7968	0.5002	0.5003
	7	675.398	0.9066	0.9067
	8	581.7034	0.5795	0.5796
	9	615.4827	0.8995	0.8996
7	0	826.7217	0.934	0.9341
	1	1123.2175	0.9923	0.9925
	2	844.6525	0.9646	0.9647
	3	805.3618	0.9506	0.9507
	4	784.0894	0.956	0.9561
	5	562.3694	0.5418	0.5419
	6	797.0386	0.9388	0.939
	7	452.6432	0.5023	0.5026
	8	633.4204	0.7368	0.7369
	9	587.8089	0.8686	0.8688

Table 5.8 Linear Best Separability - Jacobian Saliency Map - Classes 8, 9

True Class	Best Class	Separation Point	Train Separation	JSMA Separation
8	0	821.2453	0.9323	0.9323
	1	1087.2327	0.9895	0.9896
	2	726.8218	0.9318	0.9319
	3	771.1901	0.9325	0.9327
	4	735.4302	0.9378	0.9379
	5	693.8133	0.7844	0.7845
	6	726.5925	0.8955	0.8956
	7	783.8004	0.9563	0.9564
	8	561.2151	0.5005	0.5007
	9	688.0899	0.9505	0.9505
9	0	871.8646	0.9492	0.9493
	1	1117.0811	0.9917	0.9918
	2	799.3414	0.9529	0.953
	3	806.9166	0.9519	0.9519
	4	791.4185	0.9574	0.9574
	5	623.7709	0.6316	0.6317
	6	799.9306	0.9399	0.9399
	7	633.1024	0.8781	0.8782
	8	597.2288	0.6318	0.632
	9	447.5632	0.503	0.503

5.2.2 Per-Input OOD Threshold Comparison

In this section, we show a different method of using the CPLs obtained to compare them to the training set.

We consider two different categories of CPLs: those that are In-Distribution (InD) cases and those that are Out-Of-Distribution (OOD) cases. As mentioned in Section 4, we identify these two different categories by using a linear threshold based likelihood measure, where those images with CPLs that fall above the threshold are considered as OOD and those that are below the threshold are considered InD. Categorizing the images as InD or OOD could be used both to filter the InD aggressive test cases and to ward off the OOD test cases.

As mentioned in Section 4.7.1, experiments in this section were performed on ten different network structures, as well as five additional instances of one of these structures (v0). The recognition precisions of the ten trained structures are shown in Table 5.9 and recognition precisions of the five trained instances are shown in Table 5.10.

Table 5.9 Precision Measures Across the Ten Network Structures

network	train	test
v0	99.1	91.5
v1	98.5	91.6
v2	98.2	92.1
v3	95.7	92.9
v4	94.4	92.4
v5	98.2	91.3
v6	98.4	91.1
v7	98.2	91.7
v8	95.1	92.7
v9	94.3	92.2

We start by computing the training set's average and standard deviation of all the training images for a specific class, using Equations 4.6 and 4.7, respectively. Once we've obtained the average and standard deviation, we can use Equation 4.8 to normalize the CPLs of each input image with respect to the class it comes from. This would allow them to directly be compared to any arbitrary sigma-order threshold.

In our case, we consider the linear thresholds of 1.5, 2.0, and 3.0 orders of sigma. For the purposes of comparison, we also consider an infinite sigma threshold, which would be equivalent to having no upper threshold, named "none". These thresholds are based on the

Table 5.10 Precision Measures Across the Five Instances from v0

network	train	test
v0.0	98.3	91.0
v0.1	98.2	91.9
v0.2	97.7	91.5
v0.3	97.9	91.5
v0.4	98.0	91.8

number of standard deviations above the mean of each class' training set. As per Equations 4.9a and 4.9b, each image is directly compared to the chosen threshold, to determine whether the image should be considered as Out-Of-Distribution or In-Distribution.

In our experiments, we compute the respective separation of all the images when applying the threshold at various places corresponding to different sigma values with respect to the training set. Logically, as the threshold value decreases, more and more cases are considered as OOD, which allows a certain level of control in how strictly to separate them.

As previously mentioned in Section 4.7.1, we use ten different architecture structures and five different architecture instances in our experiments to reduce the bias of having a single architecture. For better readability of our results and to use the space more efficiently, we simply show the average, minimum, and maximum values of the results across the different structures and across the different instances.

Table 5.11 InD Distribution of Training and Test Sets Across All Network Structures

Set	$SepTh$	$InD\%$		
		Avg	Min	Max
train	None	100	100	100
	3.0	98.6	98.5	98.7
	2.0	95.4	95.3	95.6
	1.5	91.8	91.6	92.2
test	None	100	100	100
	3.0	95.7	94.6	97.4
	2.0	91.6	90.3	94.3
	1.5	87.9	85.6	91

We present in Table 5.11 the $InD\%$ values for the training and testing images of the ten

Table 5.12 InD Distribution of Training and Test Sets Across All Instances from v0

Set	$SepTh$	InD%		
		Avg	Min	Max
train	None	100	100	100
	3.0	98.6	98.5	98.7
	2.0	95.4	95.3	95.4
	1.5	91.7	91.6	92
test	None	100	100	100
	3.0	95.8	95.6	96
	2.0	91.2	90.8	91.6
	1.5	87	86.3	87.9

structures. Defined in Equation 4.13, this measure represents the percentage of cases that are found as in-distribution for each threshold value. In this table, we see a very large number of training images (above 98%) within the distribution at 3.0 orders of sigma, and the number slightly decreases as the order of sigma decreases, but remains above 90% even at 1.5 orders of sigma. Similarly, the testing images for all structures start with a separation average above 95% for 3.0 orders of sigma and continue to decrease while remaining above 85% for 1.5 orders of sigma. Of course, with no separation threshold, InD% remains at 100% as there is no separation being performed. The high values for InD% after separation are to be expected, as we are comparing the training and testing images of a particular class to that class's training images themselves. Thus, the training set should in fact mostly be within distribution, and since the train and test sets were sampled from the same original set of images, it is expected that the testing set also have very high values of InD%.

Table 5.12 shows the InD% values for the training and testing images of the ten instances. In this table we notice very similar trends as those seen for Table 5.11. InD% values for the training set remain above 91% at $SepTh = 1.5$ for all instances, and those for the testing set remain above 86%.

Next, we report the results obtained from our experiments for the adversarial attacks and the affine transformations.

We see in Table 5.13 the results for the adversarial attacks for each order of sigma for the ten structures. In this table, the average (Avg), minimum (Min), and maximum (Max) *No-Harm%* values over all ten architectures can once again be found, as well as those for the OOD% measure, as defined in Equations 4.11 and 4.12 respectively. We also include the

Table 5.13 Separation Measures Across All Ten Structures: Adversarial Attacks

Attack	*SepTh*	OOD%			No-Harm%			Misclassified%			CorrClassified%		
		Avg	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
cw	none	<0.05	<0.05	<0.05	15.8	0.4	32.2	84.2	67.8	99.6	15.8	0.4	32.2
	3.0	33.9	19.4	62.4	49.5	35.5	62.9	50.5	37.1	64.5	15.6	0.4	31.6
	2.0	44.6	27.5	73.3	59.7	47.7	73.8	40.3	26.2	52.3	15.2	0.3	30.5
	1.5	51.2	33.4	79.7	65.8	54.6	80.1	34.2	19.9	45.4	14.6	0.3	29.3
df	none	<0.05	<0.05	<0.05	16	5.5	26.6	84	73.4	94.5	16	5.5	26.6
	3.0	46.7	24.8	62.2	60.4	35.7	72.8	39.6	27.2	64.4	13.7	5.2	25.7
	2.0	55.7	39.4	67.8	68.4	48.5	76.9	31.6	23.1	51.5	12.7	4.6	24.7
	1.5	61	48.1	72.9	72.9	55.9	79.6	27.1	20.4	44.1	11.9	3.8	23.6
fgrad	none	<0.05	<0.05	<0.05	7.9	3.8	11.7	92.1	88.3	96.2	7.9	3.8	11.7
	3.0	74.1	34	100	76.8	40.2	100	23.2	<0.05	59.8	2.7	<0.05	6.3
	2.0	84.7	52.4	100	86.5	57.3	100	13.5	<0.05	42.7	1.8	<0.05	4.9
	1.5	89.2	62.5	100	90.5	66.4	100	9.5	<0.05	33.6	1.3	<0.05	3.9
jsma	none	<0.05	<0.05	<0.05	11.3	8.7	14.4	88.7	85.6	91.3	11.3	8.7	14.4
	3.0	69.6	52.2	86.6	77.5	58.8	91.4	22.5	8.6	41.2	7.9	4.7	10.1
	2.0	77.8	63.8	89.4	85.2	69.8	94.8	14.8	5.2	30.2	7.4	4.5	9.7
	1.5	81.8	70.3	90.9	88.9	76	96.1	11.1	3.9	24	7.1	4.4	9.1

Table 5.14 Separation Measures Across All Five Instances: Adversarial Attacks

Attack	*SepTh*	OOD%			No-Harm%			Misclassified%			CorrClassified%		
		Avg	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
cw	none	<0.05	<0.05	<0.05	28.7	26.9	31.2	71.3	68.8	73.1	28.7	26.9	31.2
	3.0	17.6	15.3	19.3	46	44.5	46.6	54	53.4	55.5	28.5	26.7	31
	2.0	26.2	23.9	27.8	54	52.6	54.4	46	45.6	47.4	27.8	26	30.3
	1.5	32.4	30.4	33.8	59.3	58	59.8	40.7	40.2	42	26.9	25	29.4
df	none	<0.05	<0.05	<0.05	21.6	20.3	22.8	78.4	77.2	79.8	21.6	20.3	22.8
	3.0	49.2	45.7	51.3	69.9	66.9	73.2	30.1	26.8	33.1	20.7	19.3	21.8
	2.0	56.2	54.5	56.7	76.3	75.3	77.8	23.8	22.2	24.7	20.1	18.8	21.2
	1.5	59.9	58.7	60.8	79.2	78	80.5	20.8	19.5	22	19.4	18.1	20.4
fgrad	none	<0.05	<0.05	<0.05	8.2	8	8.6	91.8	91.4	92	8.2	8	8.6
	3.0	95.5	91.8	99.7	95.7	92.1	99.7	4.3	0.3	7.9	0.2	<0.05	0.4
	2.0	99.1	98.1	100	99.1	98.2	100	0.9	<0.05	1.8	<0.05	<0.05	0.08
	1.5	99.7	99.3	100	99.7	99.3	100	0.3	<0.05	0.7	<0.05	<0.05	<0.05
jsma	none	<0.05	<0.05	<0.05	10	10	10	90	90	90	10	10	10
	3.0	47	45.7	48.5	56.8	55.6	58.3	43.2	41.7	44.4	9.9	9.8	9.9
	2.0	59	57.5	60.3	68.5	67	69.8	31.5	30.2	33	9.5	9.5	9.6
	1.5	65.6	64	66.8	74.8	73.3	76	25.2	24.1	26.8	9.2	9.1	9.3

results for the $Misclassified\%$ and $CorrClassified\%$ measures, defined in Equations 4.14 and 4.15, respectively. On average, for all attacks, we start with very low *No-Harm%* values below 20% when no threshold is placed (equivalent to $SepTh = \infty$). The *Misclassified%* rate is high, in some cases reaching values above 90%, and the *CorrClassified%* rate is equivalent to the low *No-Harm%* values since no separation is performed at this point. When applying a $SepTh$ of 3, we see a large increase in *No-Harm%* cases, with averages above 50% and reaching high seventies for Fast Gradient and Jacobian Saliency Maps. Impressively, these large increases occur with a relatively low decrease of *CorrClassified%* values. For example, for Deepfool, applying a $SepTh$ of 3 increases the non-harmful cases by 44% while only decreasing the false negatives by less than 3%. Of course, reducing the threshold value also increases the $OOD\%$ measures, and by filtering out these OOD images also reduces the error observed with the $Misclassified\%$ value. As we continue to decrease $SepTh$, we continue to see these trends. $OOD\%$ and *No-Harm%* values keep increasing while $Misclassified\%$ and $CorrClassified\%$ continue to decrease.

As for the results of the adversarial attacks on the five architecture instances, though they are slightly different, the same trends can still be seen and are presented in Table 5.14. Without a threshold, *No-Harm%* values are all on average below 30%, while $Misclassified\%$ values are above 70%. As we decrease the threshold by reducing the order of sigma to 3, the *No-Harm%* jumps to 96% for Fast Gradient, 70% for DeepFool, 57% for Jacobian Saliency Maps, and 46% for Carlini & Wagner. Meanwhile, in most cases, the $CorrClassified\%$ values decrease by less than 1%. As we continue to reduce the order of sigma to 1.5, the *No-Harm%* measure continues to increase, reaching almost 100% for Fast Gradient, 79% for DeepFool, 75% for Jacobian Saliency Maps, and 59% for the Carlini & Wagner. Once again, in most cases, the $CorrClassified\%$ values decrease by less than 3%. Of course, our approach's success varies across different attacks. Though Fast Gradient sees an enormous increase of more than 80% for *No-Harm%*, the $CorrClassified\%$ value jumps down to 0.2% when $SepTh = 3$, and goes down below 0.05% as we further decrease $SepTh$. However, it is also important to note that $CorrClassified\%$ was very low to begin with, where the decrease is only by about 8%, and that this behaviour for Fast Gradient seems to be unique among all the other attacks, where the average decreases are all less than 3%.

The clear correlation between the different thresholds and obtained measures shows that it is possible to filter out many incorrectly predicted images, at the cost of a very small amount of originally correctly predicted images being filtered out as well. This allows for flexibility in deciding how many false positives one would like to tolerate to successfully protect against a much larger number of false negatives.

Table 5.15 Separation Measures Across All Ten Structures: Transformations - Center Rotations

Interval	*SepTh*	OOD%			No-Harm%			Misclassified%			CorrClassified%		
		Avg	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
6	none	<0.05	<0.05	<0.05	92.8	90.1	96.2	7.2	3.8	9.9	92.8	90.1	96.2
	3.0	4.6	1.1	8.1	93.8	90.7	97.3	6.2	2.7	9.3	89.2	86.2	92.3
	2.0	8.5	2.9	12.9	94.4	91.2	97.7	5.6	2.3	8.9	85.8	82.3	89.2
	1.5	12	5.2	16.6	94.8	91.5	98	5.2	2	8.5	82.8	78.9	86.3
12	none	<0.05	<0.05	<0.05	80	69	89.9	20	10.1	31	80	69	89.9
	3.0	12	2.2	21.4	83.3	71.6	93.8	16.7	6.3	28.4	71.4	62.1	79.7
	2.0	19.9	8	32.2	85.3	74.3	95	14.7	5	25.7	65.4	55	73.2
	1.5	25.9	12.2	39.8	86.8	76.9	95.8	13.2	4.2	23.1	60.9	49.7	68.4
18	none	<0.05	<0.05	<0.05	59.7	50.1	72.8	40.3	27.2	49.9	59.7	50.1	72.8
	3.0	24.7	4.2	45.1	70.3	55.6	89.7	29.7	10.3	44.4	45.6	36.8	56.1
	2.0	36.1	10.9	55.9	75.4	59.3	92.7	24.6	7.3	40.7	39.3	28.5	48.6
	1.5	43.6	15.7	64	78.7	61.9	94.2	21.3	5.8	38.1	35.1	23.4	46.1
24	none	<0.05	<0.05	<0.05	44	37.7	53.1	56	46.9	62.3	44	37.7	53.1
	3.0	36	4.6	60.5	64.6	44.4	84.9	35.4	15.1	55.6	28.6	18	39.8
	2.0	48.8	12.3	74.3	72	48.6	89.9	28	10.1	51.4	23.2	12.9	36.3
	1.5	56.7	20.1	81.2	76.7	53.1	93	23.3	7	46.9	20	10.3	33
30	none	<0.05	<0.05	<0.05	32.3	26.9	39.9	67.7	60.1	73.1	32.3	26.9	39.9
	3.0	42.1	6.8	69.2	61.2	37.5	83.7	38.8	16.3	62.5	19.1	10.2	30.7
	2.0	55.3	15.7	81.6	70.7	42.5	89.9	29.3	10.1	57.5	15.4	7.5	26.9
	1.5	63.2	22.9	87.7	76.4	47.4	93.6	23.6	6.4	52.6	13.3	5.9	24.5

Table 5.16 Separation Measures Across All Five Instances: Transformations - Center Rotations

Interval	*SepTh*	OOD%			No-Harm%			Misclassified%			CorrClassified%		
		Avg	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
6	none	<0.05	<0.05	<0.05	91.4	90.6	92.3	8.6	7.8	9.5	91.4	90.6	92.3
	3.0	3.3	2.9	3.9	92	91.1	92.8	8	7.2	8.9	88.7	87.8	89.6
	2.0	6.9	6.4	7.9	92.4	91.6	93.3	7.6	6.7	8.4	85.5	84.4	86.3
	1.5	10.4	9.6	11.7	92.8	92	93.6	7.2	6.4	8	82.4	81	83.2
12	none	<0.05	<0.05	<0.05	76.4	75	79.9	23.6	20.1	25	76.4	75	79.9
	3.0	7.1	6.1	8.1	78.3	77	81.3	21.8	18.7	23	71.1	68.9	74.6
	2.0	14.1	11.9	15.6	79.7	78.5	82.4	20.3	17.6	21.5	65.6	63.3	68.1
	1.5	19.5	16.5	21.1	80.9	79.4	83.3	19.1	16.7	20.6	61.4	58.9	63.2
18	none	<0.05	<0.05	<0.05	53.4	52.3	54.6	46.6	45.4	47.7	53.4	52.3	54.6
	3.0	13.6	10.2	17	60	58.3	63.3	40	36.7	41.7	46.5	43.8	48.3
	2.0	23.5	18.4	27	64.4	62.4	67.7	35.6	32.3	37.6	41	37.7	44
	1.5	31	24.6	34.5	67.8	65.1	70.7	32.2	29.4	34.9	36.7	33.7	40.5
24	none	<0.05	<0.05	<0.05	38.3	37.8	39.3	61.7	60.7	62.3	38.3	37.8	39.3
	3.0	18.6	13.8	22.5	48.9	46.3	52.3	51.1	47.7	53.7	30.3	25.7	34
	2.0	30.9	25.3	34.7	55.9	53.4	59.8	44.1	40.2	46.6	25	20.7	28.9
	1.5	39.5	33.8	43.3	61.1	59.1	64.7	38.9	35.3	41	21.5	17.6	25.2
30	none	<0.05	<0.05	<0.05	29.1	28	30.5	70.9	69.6	72	29.1	28	30.5
	3.0	19.9	16	23.9	40.9	39.3	43.6	59.1	56.4	60.7	21	17.8	23.6
	2.0	31.8	27.5	35.5	49.1	47.3	50.6	50.9	49.4	52.7	17.3	14.4	19.8
	1.5	40.8	35.9	44.3	55.9	53.5	57.5	44.1	42.5	46.5	15.1	12.3	17.6

Table 5.17 Separation Measures Across All Ten Structures: Transformations - Corner Rotations

Interval	*SepTh*	OOD%			No-Harm%			Misclassified%			CorrClassified%		
		Avg	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
4	none	<0.05	<0.05	<0.05	87.8	82.7	92	12.2	8	17.3	87.8	82.7	92
	3.0	8.3	2.2	13.6	90.2	85.1	94.8	9.8	5.2	14.9	81.8	79.1	87.1
	2.0	14.6	5.1	21.8	91.4	86.1	95.6	8.6	4.4	13.9	76.8	72.8	82.7
	1.5	19.8	8.1	27.9	92.3	86.8	96.2	7.7	3.8	13.2	72.5	67.2	78.8
8	none	<0.05	<0.05	<0.05	69.7	60.7	77.9	30.3	22.1	39.4	69.7	60.7	77.9
	3.0	31.7	13.3	44.3	82.8	72	94.5	17.2	5.6	28	51.1	43.3	58.7
	2.0	45.2	22.1	58.5	87.6	76.2	96.4	12.4	3.6	23.8	42.4	33.9	54.1
	1.5	53.4	29.7	66.7	90.4	79.4	97.3	9.6	2.7	20.6	37	28.2	49.7
12	none	<0.05	<0.05	<0.05	47.2	40.7	58.7	52.8	41.3	59.3	47.2	40.7	58.7
	3.0	58	34.5	73.4	80.9	66.1	96.7	19.1	3.3	33.9	22.9	15.9	31.6
	2.0	71.8	47.9	85.4	88.7	74.3	98.5	11.3	1.5	25.8	16.9	11.2	26.4
	1.5	78.7	56	89.9	92.5	78.7	99.1	7.5	0.9	21.3	13.8	8.3	22.7
16	none	<0.05	<0.05	<0.05	33.3	25.5	43.6	66.7	56.4	74.5	33.3	25.5	43.6
	3.0	69.1	48.5	88	81.2	61.3	94	18.8	6	38.7	12	6	16.2
	2.0	81.2	61.2	94.7	89.5	74.4	97.9	10.5	2.1	25.6	8.3	3.2	13.3
	1.5	86.6	67.9	96.8	93.1	79.4	98.9	6.9	1.1	20.6	6.5	2.2	11.6
20	none	<0.05	<0.05	<0.05	22.6	14.9	29.1	77.4	70.9	85.1	22.6	14.9	29.1
	3.0	76.1	55.6	89.7	83.8	64.8	93.9	16.2	6.1	35.2	7.7	3.3	10.6
	2.0	87	72.2	96.2	91.8	77.7	97.8	8.2	2.2	22.3	4.8	1.6	8.4
	1.5	91.4	81	98	94.9	85.9	99	5.1	1	14.1	3.5	1	7
24	none	<0.05	<0.05	<0.05	16.6	9.6	22	83.4	78	90.4	16.6	9.6	22
	3.0	80.2	67.1	91.6	85.5	72.6	94.7	14.5	5.3	27.5	5.3	2.8	7.1
	2.0	90	81.6	97.2	93	84.4	98.4	7	1.6	15.6	3.1	1.2	5.1
	1.5	93.8	88.2	98.8	96	89.6	99.3	4	0.7	10.4	2.2	0.6	4.3

Table 5.18 Separation Measures Across All Five Instances: Transformations - Corner Rotations

Interval	*SepTh*	OOD%			No-Harm%			Misclassified%			CorrClassified%		
		Avg	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
4	none	<0.05	<0.05	<0.05	86.2	85.4	86.9	13.8	13.1	14.7	86.2	85.4	86.9
	3.0	5.2	4.5	5.7	87.6	86.7	88.4	12.4	11.6	13.4	82.4	82	82.9
	2.0	10.8	9.4	11.9	88.7	87.6	89.6	11.3	10.4	12.4	77.9	77.1	78.7
	1.5	15.8	13.8	17.1	89.7	88.4	90.5	10.3	9.5	11.6	73.9	72.7	74.9
8	none	<0.05	<0.05	<0.05	66.2	64.5	68.7	33.8	31.3	35.5	66.2	64.5	68.7
	3.0	19.2	13.7	21.4	74.2	71.9	75.3	25.8	24.8	28.1	55	50.8	60.2
	2.0	34.3	28.6	38.4	80.7	79.2	82.5	19.3	17.5	20.8	46.4	41.9	50.7
	1.5	45.3	39.5	50.2	85.4	83.2	87.4	14.6	12.6	16.9	40.1	35.6	44.9
12	none	<0.05	<0.05	<0.05	42.4	40.6	46.2	57.6	53.8	59.5	42.4	40.6	46.2
	3.0	44.6	39.5	52.7	71.2	65.8	75.2	28.8	24.8	34.2	26.6	22.5	30.9
	2.0	63.2	58.5	68.5	82.8	78.7	86.5	17.2	13.5	21.3	19.6	15.8	24.2
	1.5	73.3	68.8	77.5	89.1	87.3	91.6	10.9	8.5	12.7	15.8	12.5	19.8
16	none	<0.05	<0.05	<0.05	30	28	31.3	70	68.8	72	30	28	31.3
	3.0	60.2	56.5	65.4	74.7	72.1	79	25.3	21	27.9	14.5	12.6	16.7
	2.0	76.7	74	79.7	86.9	83.3	89.6	13.1	10.5	16.7	10.2	8.4	11.6
	1.5	85	83.5	86.4	92.9	90.6	94.3	7.1	5.7	9.4	7.8	6.2	8.8
20	none	<0.05	<0.05	<0.05	20.2	17.1	21.9	79.8	78.1	82.9	20.2	17.1	21.9
	3.0	73.4	66.8	79	82.1	74.8	87	17.9	13.1	25.2	8.6	8	10.3
	2.0	87.4	81.8	91.1	92.5	86.6	95.8	7.5	4.2	13.4	5.1	4.1	6
	1.5	93	89.7	95	96.4	92.9	98.2	3.6	1.8	7.1	3.5	2.5	4.4
24	none	<0.05	<0.05	<0.05	15	12.5	16.8	85	83.2	87.5	15	12.5	16.8
	3.0	82.6	76.1	87.5	88	81.8	92.2	12	7.8	18.2	5.4	4.6	7.4
	2.0	93	87.3	95.4	95.7	90.6	98	4.3	2	9.5	2.7	2	3.3
	1.5	96.5	93.2	97.7	98.1	95.2	99.4	1.9	0.6	4.8	1.6	1	2

Table 5.19 Separation Measures Across All Ten Structures: Transformations - Translations

Interval	*SepTh*	OOD%			No-Harm%			Misclassified%			CorrClassified%		
		Avg	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
1	none	<0.05	<0.05	<0.05	90.4	86.6	94.7	9.6	5.3	13.4	90.4	86.6	94.7
	3.0	9.3	5.4	11.7	92.4	87.8	96.4	7.6	3.6	12.2	83.1	80	86.5
	2.0	16.3	11.1	19.4	93.5	88.9	97	6.5	3	11.1	77.1	72.4	81.8
	1.5	22.1	16.3	26	94.2	89.8	97.4	5.8	2.6	10.2	72.2	66.1	77.9
3	none	<0.05	<0.05	<0.05	44.1	34	55.9	55.9	44.1	66	44.1	34	55.9
	3.0	57.1	41.7	71.3	78.6	65.4	87.8	21.4	12.2	34.6	21.5	15.4	27.4
	2.0	71	56.1	82.5	87	76.3	93.5	13	6.5	23.7	16	10.9	21
	1.5	78.1	64.7	87.5	91.1	81.9	96	8.9	4	18.1	13	8.5	17.4
5	none	<0.05	<0.05	<0.05	14.1	9.9	17.7	85.9	82.3	90.2	14.1	9.9	17.7
	3.0	63.3	38.3	78.6	70.6	48.2	85	29.4	15.1	51.8	7.3	4.6	10
	2.0	77.7	52.7	90.5	83.1	61.2	94.7	16.9	5.3	38.8	5.5	3.4	8.5
	1.5	84.2	61.6	94.3	88.6	69	97.4	11.4	2.6	31	4.4	2.7	7.4
7	none	<0.05	<0.05	<0.05	8.9	6	13	91.1	87	94	8.9	6	13
	3.0	65	34.7	90.5	68.7	39.4	92.6	31.3	7.4	60.6	3.7	1.8	6.8
	2.0	79.2	53.1	97.4	81.6	56.8	98.3	18.4	1.7	43.2	2.4	0.9	5.2
	1.5	85.8	65.6	99	87.6	68.7	99.5	12.4	0.5	31.3	1.7	0.5	4.2
9	none	<0.05	<0.05	<0.05	6.4	4.1	9.6	93.6	90.4	96	6.4	4.1	9.6
	3.0	68.2	37	96.7	70.5	40.8	97.2	29.5	2.8	59.2	2.2	0.3	4.2
	2.0	81.6	60	99.6	82.9	62.4	99.7	17.1	0.3	37.6	1.3	0.08	2.6
	1.5	87.9	73.2	99.9	88.8	73.9	99.9	11.2	0.07	26.1	0.9	<0.05	1.8
11	none	<0.05	<0.05	<0.05	8.1	5	13.6	91.9	86.4	95	8.1	5	13.6
	3.0	70.7	29.1	98.9	73.2	35.6	99	26.8	1	64.4	2.5	0.06	7.2
	2.0	83.3	51.9	99.8	84.6	56	99.9	15.4	0.2	44	1.3	<0.05	4.1
	1.5	89	66.4	99.9	89.8	69.2	99.9	10.2	0.06	30.8	0.9	<0.05	2.8
13	none	<0.05	<0.05	<0.05	10.4	6.7	14.1	89.6	85.9	93.3	10.4	6.7	14.1
	3.0	69.7	24.8	99.7	73.2	33.7	99.7	26.8	0.3	66.3	3.5	<0.05	8.9
	2.0	83.2	50.2	99.9	85.2	55.6	100	14.8	0.05	44.5	2	<0.05	5.3
	1.5	89.2	64.2	100	90.5	67.7	100	9.5	<0.05	32.3	1.4	<0.05	4
15	none	<0.05	<0.05	<0.05	12.4	8.3	16.1	87.6	83.9	91.7	12.4	8.3	16.1
	3.0	65.2	17.1	99.7	69.9	25.9	99.7	30.1	0.3	74.1	4.7	<0.05	8.9
	2.0	79.6	44.6	99.9	82.2	49.5	99.9	17.8	0.08	50.5	2.6	<0.05	7.2
	1.5	86.4	60.5	100	88.2	63.2	100	11.8	<0.05	36.8	1.8	<0.05	6.2

Table 5.20 Separation Measures Across All Five Instances: Transformations - Translations

Interval	*SepTh*	OOD%			No-Harm%			Misclassified%			CorrClassified%		
		Avg	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
1	none	<0.05	<0.05	<0.05	88	86.7	88.9	12	11.1	13.3	88	86.7	88.9
	3.0	7.6	6.9	8.5	89.5	88.5	90.2	10.5	9.8	11.5	81.9	81.1	82.7
	2.0	15.1	14.5	15.8	90.7	89.9	91.2	9.3	8.8	10.1	75.6	75.1	76.3
	1.5	21.6	21.3	22.4	91.7	91	92.1	8.3	7.9	9	70	69.7	70.7
3	none	<0.05	<0.05	<0.05	41.5	37.2	44.4	58.5	55.6	62.8	41.5	37.2	44.4
	3.0	48.7	45.1	52	73.4	70.4	76.1	26.6	23.9	29.6	24.6	23	26.4
	2.0	69.8	66.1	72.9	86.8	84.4	88.5	13.2	11.5	15.6	16.9	15.5	18.3
	1.5	79.8	76.9	82	92.6	90.8	93.7	7.4	6.3	9.2	12.8	11.7	14
5	none	<0.05	<0.05	<0.05	12.9	11.1	14.4	87.1	85.7	88.9	12.9	11.1	14.4
	3.0	69	58.8	77.3	75.3	64.6	82.7	24.7	17.4	35.4	6.3	5.3	7.5
	2.0	83.6	74.7	89.5	88	79.1	93.2	12	6.8	21	4.4	3.7	5.4
	1.5	90.3	84.8	94.1	93.8	88.4	97	6.2	3.1	11.6	3.4	2.9	4.3
7	none	<0.05	<0.05	<0.05	8.5	6.3	10.4	91.5	89.6	93.7	8.5	6.3	10.4
	3.0	82.1	73.8	91.9	84.6	76.3	93.1	15.4	6.9	23.7	2.5	1.2	3.3
	2.0	94.4	92	98.1	95.7	93.5	98.6	4.3	1.5	6.5	1.3	0.4	1.9
	1.5	97.7	96.5	99.4	98.5	97.8	99.6	1.5	0.4	2.2	0.8	0.2	1.3
9	none	<0.05	<0.05	<0.05	7.5	5.8	8.3	92.5	91.7	94.3	7.5	5.8	8.3
	3.0	90.7	83.4	98	91.4	85	98.2	8.6	1.8	15	0.7	0.2	1.6
	2.0	98	95.7	99.8	98.2	96.2	99.8	1.8	0.2	3.8	0.2	<0.05	0.5
	1.5	99.4	98.4	100	99.4	98.6	100	0.6	<0.05	1.4	0.08	<0.05	0.2
11	none	<0.05	<0.05	<0.05	8.7	7.1	9.2	91.3	90.8	92.9	8.7	7.1	9.2
	3.0	97.3	94.7	99.1	97.5	95	99.3	2.5	0.7	5	0.2	0.05	0.3
	2.0	99.6	99.1	99.9	99.6	99.2	99.9	0.4	0.06	0.8	<0.05	<0.05	0.07
	1.5	99.9	99.7	100	99.9	99.8	100	0.09	<0.05	0.2	<0.05	<0.05	<0.05
13	none	<0.05	<0.05	<0.05	9.5	7.6	11	90.5	89	92.4	9.5	7.6	11
	3.0	98.8	97.3	99.9	98.9	97.5	99.9	1.1	0.1	2.5	0.1	<0.05	0.2
	2.0	99.8	99.6	100	99.9	99.7	100	0.1	<0.05	0.3	<0.05	<0.05	<0.05
	1.5	100	99.9	100	100	99.9	100	<0.05	<0.05	0.06	<0.05	<0.05	<0.05
15	none	<0.05	<0.05	<0.05	10.6	8.8	13.6	89.4	86.4	91.2	10.6	8.8	13.6
	3.0	98.2	96.8	99.8	98.3	97	99.8	1.7	0.2	3	0.1	<0.05	0.2
	2.0	99.8	99.7	100	99.8	99.7	100	0.2	<0.05	0.3	<0.05	<0.05	<0.05
	1.5	99.9	99.9	100	100	99.9	100	0.05	<0.05	0.09	<0.05	<0.05	<0.05

In Tables 5.15 and 5.16, results for the same four measures are presented for images that were transformed with Center Rotation (as per Section 4.7.3) for the ten structures and the five instances, respectively. The first column, titled *Interval*, represents the interval of transformation being applied. We also present the results of Center Rotations on the Structures, Center Rotations on the Instances, Translations on the Structures, and Translations on the Instances, in Tables 5.17, 5.18, 5.19 and 5.20, respectively. Similarly to the adversarial attacks, we consistently see that for all transformations both the *No-Harm%* and $OOD\%$ measures decrease and $Misclassified\%$ and $CorrClassified\%$ measures increase as we raise the $sepTh$ value. In most cases, we also see the $OOD\%$ measure consistently increase as we increase the parameter of transformation.

To visualize the results for each test case, after computing the images' CPLs we place them as points on a cartesian map, for each class. Similarly to previous sections, each plot shows the CPLs calculated for the train and test sets of that class, along with ten columns representing the ten different network classes and indicating the network's prediction of the image in question, whether it is correct or incorrect. For example, a point in the best3 column of a class 0 plot signifies that the image was originally from class 0 but has been predicted to be from class 3.

For each class, we visualize a separation threshold once again at the same orders of sigma seen in the tables presented: 1.5, 2.0, 3.0, and *none* (no threshold). Each threshold can be seen on the plot as the black line in the train column.

Examples of the Jacobian Saliency Maps visualizations for architecture v0.0 can be found in Figure 5.5 for class 1, which shows a plot for each of the four sigma thresholds. Similarly, for the purposes of comparison, we also present the results for class 3 in Figure 5.6. These plots were chosen as useful examples purely for demonstrative purposes, as there would be too many plots to display them all. There are indeed visualizations that identify many more OOD cases (such as in Fast Gradient Sign method) and some that identify less (such as for Carlini & Wagner). All other results can also be illustrated in the same way, and we invite the reader to consult Tables 5.13 and 5.14 for more information and to compare between the different attacks and sigma thresholds.

Those images that fall below the black threshold t can be thought of as InD, as described in Section 4.5, while those that fall above the threshold are considered OOD. To help with visualizing, each column is split into two colours: green, which represents either the incorrectly predicted images that are OOD (can be easily warded off) or any correctly predicted image (not dangerous), and red, which represents the incorrectly predicted image that are InD.

In our visualizations, we can clearly see how increasing the sigma threshold classifies more

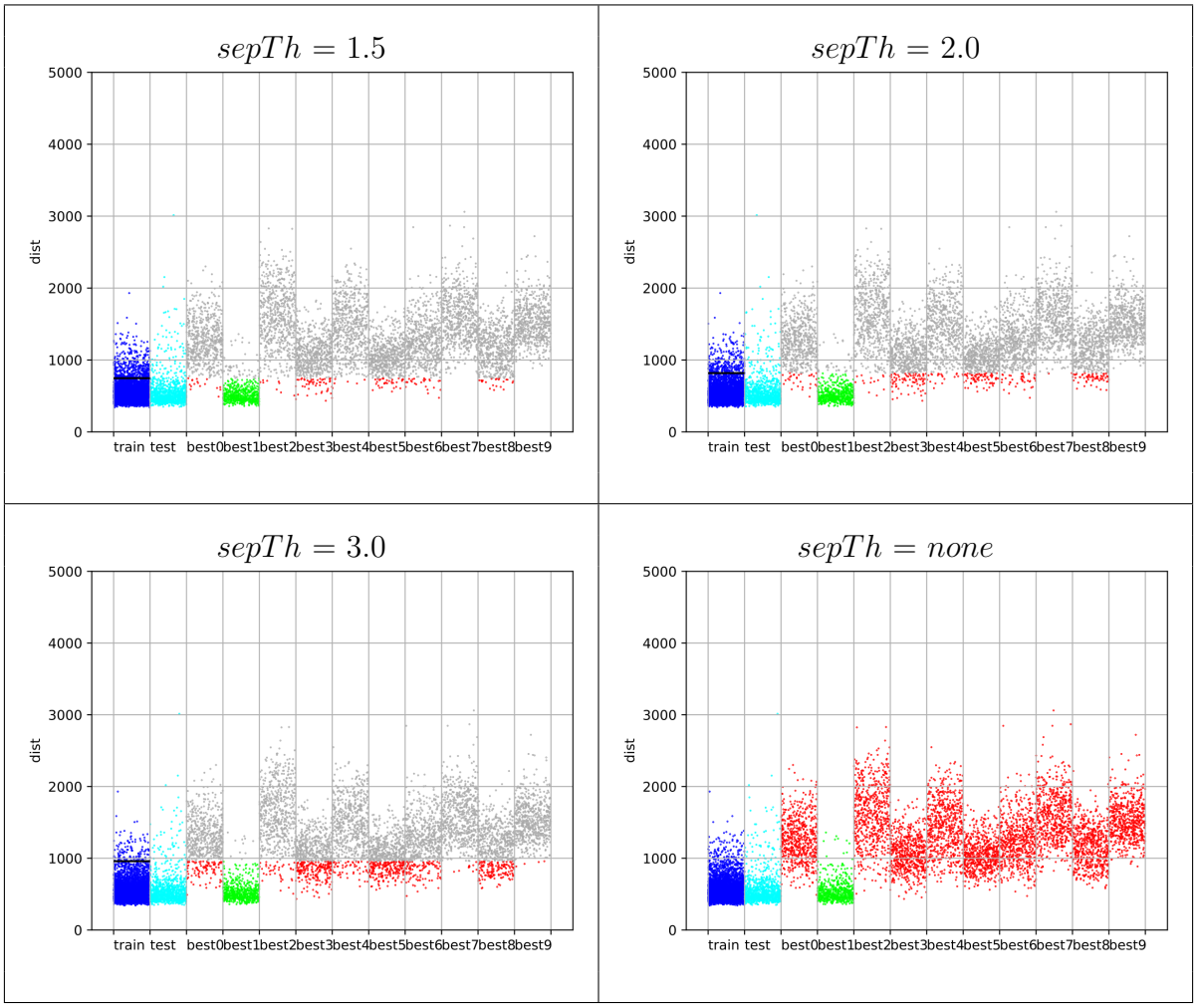

Figure 5.5 CPL Plots for Jacobian-based Saliency Maps Attack Per Order of Sigma: Architecture v0.0, Class 1

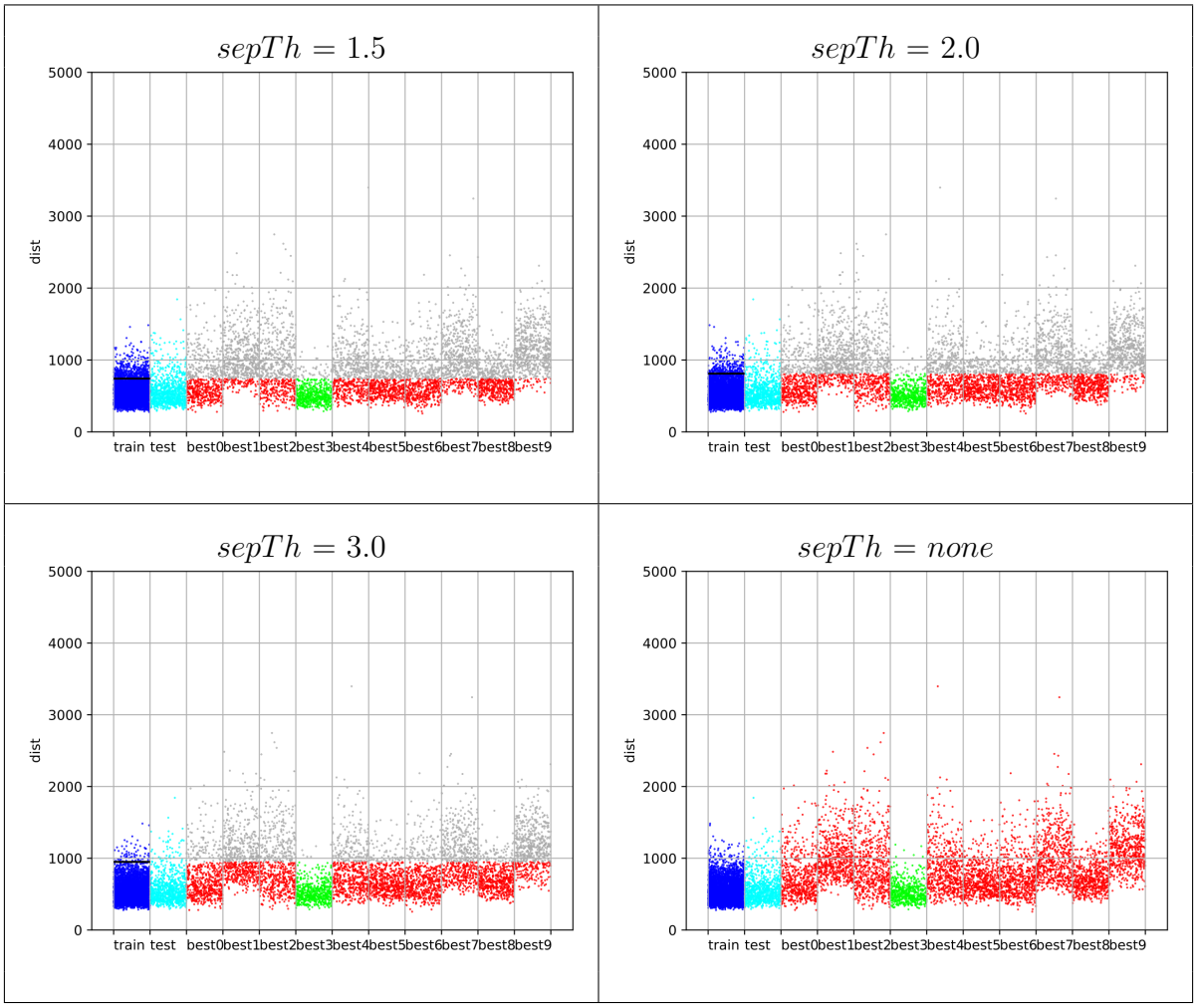

Figure 5.6 CPL Plots for Jacobian-based Saliency Maps Attack Per Order of Sigma: Architecture v0.0, Class 3

images as In-Distribution. For example, in Figure 5.5, for Architecture v0.0's class 1 of Jacobian Saliency Maps attacks, we manage to identify more than 95% of incorrectly predicted OOD cases when considering a threshold of 1.5 orders of sigma. As we increase the threshold value to 2.0 and then 3.0 orders of sigma, the numbers fall to approximately 90% and then 80% respectively, for some best predictions. For the latter, in some cases the values fall below 70%, such as for best3 which is at 67% and best5 which is at 59%. Of course, when looking at an infinite value threshold (i.e. no effective threshold), we see 0% of cases as OOD.

A similar trend can be seen in Figure 5.6, for Architecture v0.0's class 3 of Jacobian Saliency Maps attacks. For this class, when separating at 1.5 orders of sigma, the first thing we notice is that there is a lot of red on the graph. The values across different prediction classes varies greatly. For example, we manage to identify only 36% and 29% of incorrectly predicted OOD cases for best0 and best5 respectively. On the other hand, for best1 and best9, the values shoot up to 85% and 98% respectively. As we saw for class 1, we notice for class 3 that as we increase the threshold, the number of identified incorrectly predicted OOD cases decreases, however we still see a wide range of results for this class. When the threshold is at 2.0 orders of sigma, best0 and best5 have 25% and 21% of identified cases respectfully, while best1 and best9 are at 73% and 96%. When the threshold is at 3.0 orders of sigma, best0 and best5 have 12% and 10% of identified cases respectfully, while best1 and best9 are at 50% and 87%. This is an example of one of the worst classes and shows how different classes behave differently to the same approach. The trends of increasing the sigma value are also further demonstrated by the separation tables (Tables 5.11, 5.12, 5.13, 5.14) as we compare the numbers for different sigma units and notice that in all cases, for all attacks and across all architectures, both the *No-Harm%* and *OOD%* measures decrease as we raise the *sepTh* value.

In all cases, we notice that the column for the correctly predicted classes (with green images) tends to have very low CPLs, and rarely has images that are classified as out of distribution. We also observe particular classes that tend to regularly have more incorrectly predicted images labeled as In-Distributions, for all computed orders of sigma. Some attacks seem to be more prone to creating images that fall within the train distributions, as opposed to others that have many more images that are OOD. For example, when comparing the plots in Figure 5.5 for class 1 we see some best prediction columns with image points that are much further away than the training distribution than with other columns. This trend also appears in Figure 5.6 for class 3.

Figures 5.7 and 5.8 show similar plots for Classes 0 and 6 respectively for translations when the threshold is set to 1.5 orders of sigma. Similarly to the translation plots presented in

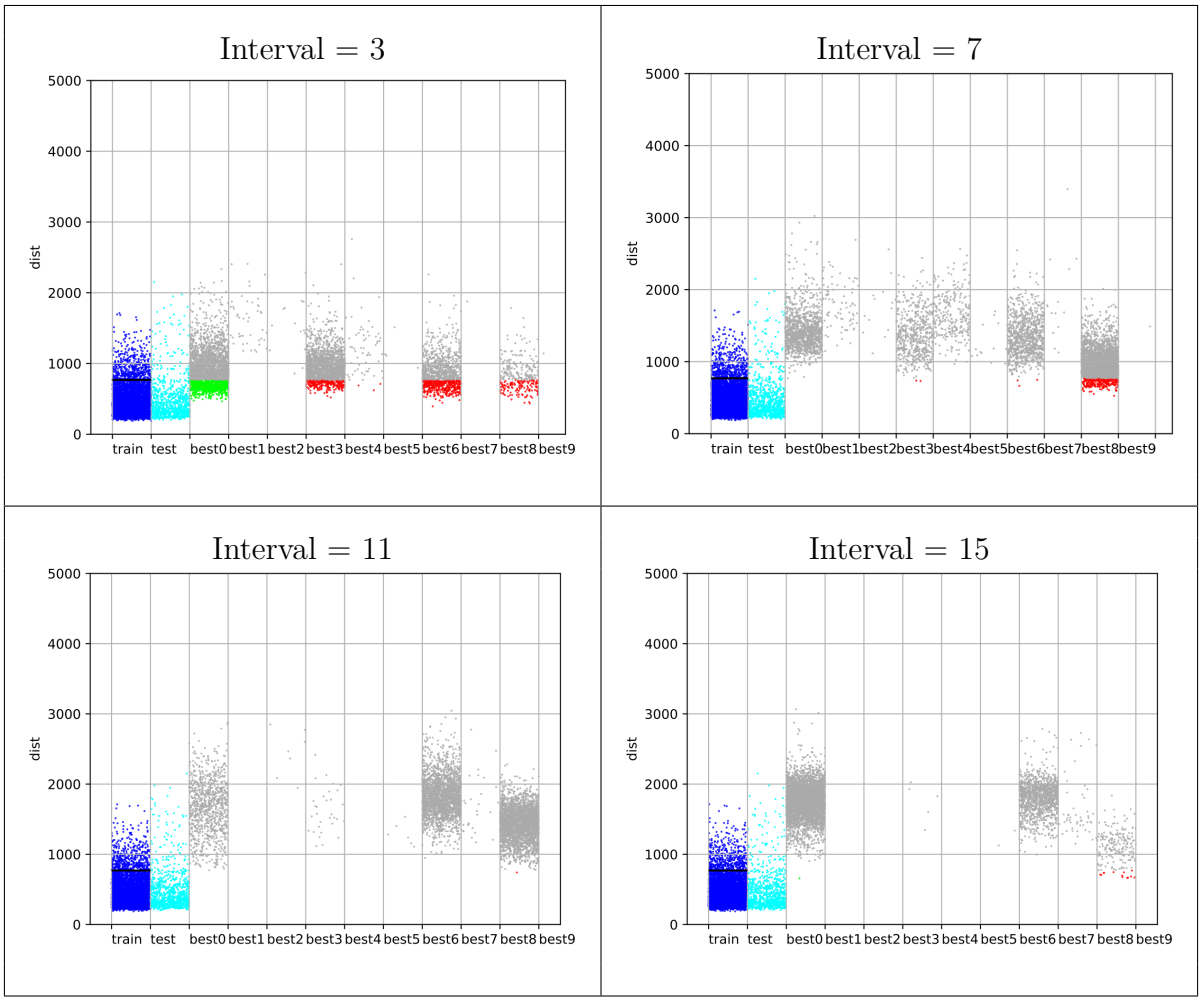

Figure 5.7 CPL Plots for Transformation - Translation - $sepTh = 1.5$, Architecture v0.0, Class 0

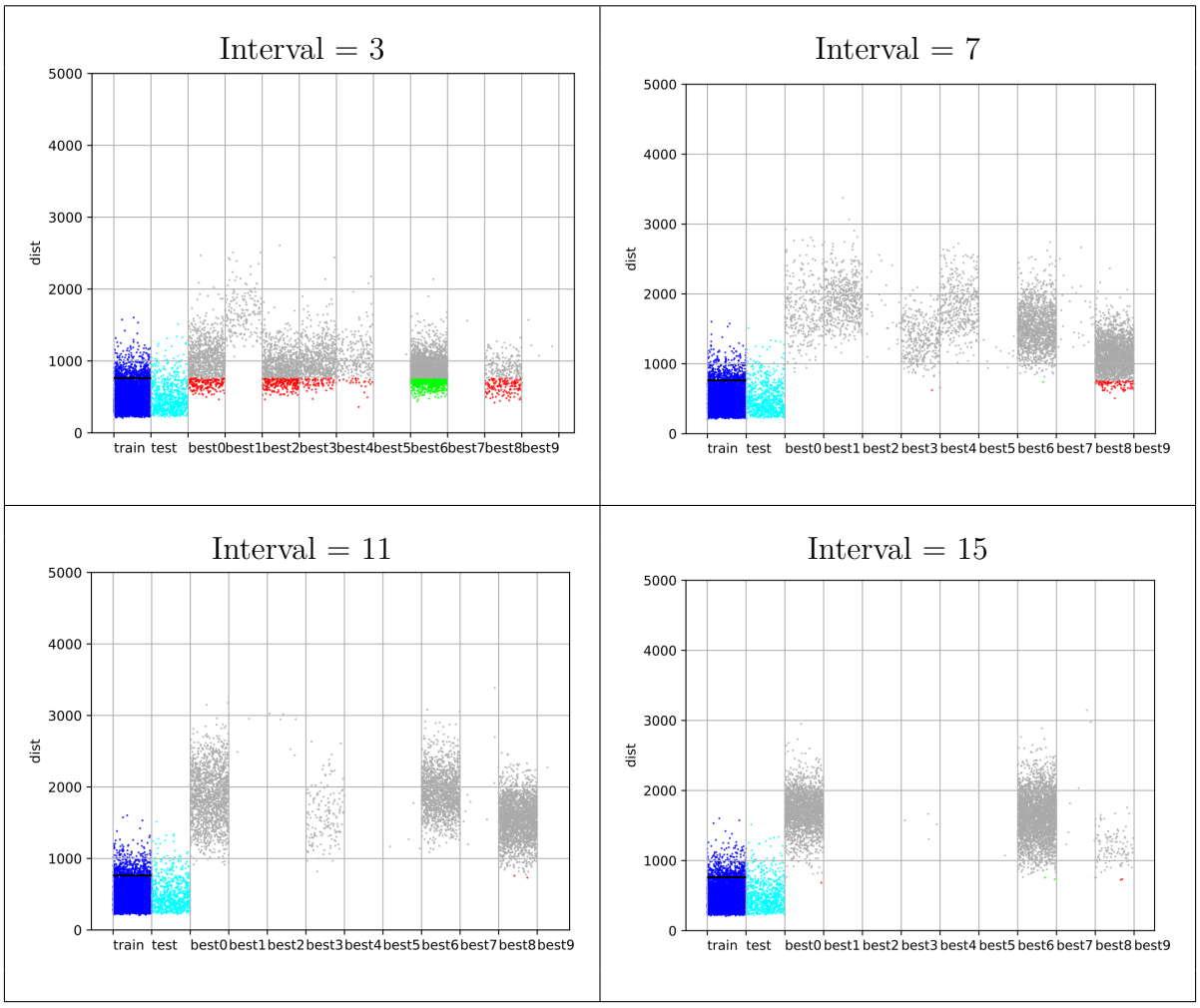

Figure 5.8 CPL Plots for Transformation - Translation - $sepTh = 1.5$, Architecture v0.0, Class 6

Section 5.2.1, we create a plot for each true class as well as for each interval of transformation. For lack of redundancy and to save space, we include examples of Intervals 3, 7, 11, and 15.

For Class 0, we notice that when the interval is 3 units, there are several columns such as best3 and best6 that have many incorrectly predicted images still considered as within distribution (shown in red). Upon increasing the interval to 7 units, most of these images get identified as out of distribution. With higher intervals of 11 and 15, we reach a point where it is rare that any of these images are considered as within the distribution. This demonstrates how transforming an image generally brings it further away from the training distribution, and thus the more the image is transformed, the more likely it is to be considered as out of distribution, as the computational profile becomes more and more different.

The same can be noticed in Class 6, as we start off with several columns with many images classified to be within distribution when the interval value is at 3 units. Then, as we increase the interval of transformation, more and more images gradually become considered as out of distribution of the training set, until it becomes very rare to find any images within distribution.

For all adversarial attacks and affine transformations, further research can be performed to further investigate the cases that are wrongly classified by the network, but nevertheless categorized as In-Distribution. Depending on the application and field, cases like these could be classified as bugs within the CNN and be sent back to the developers for inspection. If it is decided by a human oracle, they may even be added to the training data as extra examples for certain classification classes.

Our experiments also make way to further investigate computational profiles in the future as a measure of robustness in a model, to indicate levels of vulnerability of a model when it faced with aggressive testing or adversarial attacks. The ratio of aggressive tests that can be fended off, presented in our experiments as $OOD\%$ may serve to indicate how well our model can be protected against these cases.

This idea can be investigated even further when performing assessment by class, since the behaviour of classes when faced with aggressive testing is different across the different classes. Whether the computational profiles could be used to assess the robustness of a network on a per-class basis would still need to be determined. But if so, the robustness could then be assessed independently for each class and with respect to particular domain-based risk levels.

CHAPTER 6 CONCLUSION

In this chapter, we review the work that has been done in this thesis and conclude with the limitations of our experiments as well as future directions that additional research may take.

6.1 Summary of Work

In this thesis, we investigate the activation levels of CNNs and compare the computational profiles of original training images to those of test cases that were not drawn from the same distribution of the training sets. Namely, we investigate the use of synthetic test cases obtained from affine transformations of legitimate inputs and from adversarial attacks.

We present a novel statistical white-box approach to investigate the "reasoning" of the decisions that the network is making, based on the inference of a non-parametric model of vectors of activation levels produced by the neurons in a network's layers. These vectors are what we introduce as computational profiles, and we present their likelihood as Computational Profile Likelihood (CPL), which indicates the likelihood of a network's prediction, when compared to a standard input distribution. In the context of our experiments, a larger CPL represents a less probable prediction.

We consider this a non-parametric variant of SADL [1]. Our approach also allows for more refined analysis of each class, since it is class-based. Our proposed stochastic models have been estimated during training, and in contrast to SADL which retrains the system, ours is used without further retraining of the system and without the need of a secondary classifier on any image features.

We compare the legitimate training and testing cases to inputs generated by applying affine transformations or by attacking with adversarial attacks. We place them on plots to better visualize their differences. When looking at the experimental CPL values, we also notice that CPL distributions for training and test sets are very similar to eachother, and are often much higher than those for the affine transformations and adversarial attacks.

We first attempt to differentiate the CPL's of legitimate inputs from the unfamiliar inputs per-set, by finding the best linear separation between the familiar training set and the unfamiliar set. This shows several interesting trends, such that the CPLs for the correctly predicted classes tend to be lower than those for the incorrectly predicted classes. We also find that for most classes the best separation precisions between the training set and the set in question are quite high and depend not only on the true class that they belong to but also on the best

class they are predicted to be in. This demonstrates how different classes behave differently and that the different classes can and should be assessed independently.

We then attempt to separate between legitimate inputs and unfamiliar inputs drawn from a different distribution on a per-image basis, using a linear threshold set by particular orders of sigma, where any case found above the threshold is considered as Out-Of-Distribution. In our experiments, we succeed in separating between In-Distribution and Out-Of-Distribution cases with respect to the training distribution, as the CPL's for the affine and adversarial cases are often significantly lower than those observed for the training sets. In our experiments, we see that our OOD analysis of CPLs allows the successful identification and filtering of many affine and adversarial cases, where our OOD identification ratio of incorrectly predicted and harmful images is often quite high relative to a small decrease of correctly predicted images.

6.2 Limitations

Several of our experiments have been repeated on different networks that present fully connected output layers with the RELU transfer function. Although these architectures are quite common and frequent, obtained results are not generalizable to different structures and additional experiments have to be performed and repeated.

Furthermore, our experiments were done using one large database of clothing images [2]. It would thus be useful to perform more experiments on other datasets and other domains.

As for our generated images, we used a limited number of affine transformations (translation, corner rotation, center rotation) and adversarial attacks(Fast Gradient, Jacobian Saliency Map, DeepFool, Carlini & Wagner). Additional experiments should be repeated with more types of transformations and attacks.

We also used a limited set of intervals for the transformations and a fixed set of hyper-parameters for each attack. Once again, for generalizability, further experiments should be run with more combinations of transformation intervals and attack hyper-parameters.

As mentioned in Section 4.7.2, in our experiments we considered neurons only the second to last layer for computational efficiency and due to recent research suggesting that layer sensitivity is more effective on the final layers of a network than on the initial layers for our particular experiments using the MNIST database. Conversely, the deeper (initial) layers are more effective for the CIFAR dataset, and so our presented results may not necessarily generalize on all other datasets. Therefore more layers should be investigated to compute the CPLs.

6.3 Future Research

We focus on using our presented approach as an OOD detection tool to filter and classify the computation likelihood of incoming inputs. However, future research could be done to actually apply our approach as a defence against unusual cases in operational settings.

Potentially, the difficult and low-likelihood cases that our approach finds could be flagged as unreliable predictions or bugs in the system to be reviewed by developers.

As mentioned in Section 6.2, future directions could include broadening the scope of experiments by performing them on different datasets and models, with additional transformations, attacks, intervals, and hyper-parameters. This would greater improve chances of generalizability and discovering additional patterns or trends.

Further research could also be performed on each class to understand their different characteristics and why they behave so differently.

It would also be interesting to build on our work by using a combination of techniques, such as our approach combined with the application of ensemble models (such as boosting or bagging), to potentially further increase separability.

REFERENCES

[1] J. Kim, R. Feldt, and S. Yoo, "Guiding deep learning system testing using surprise adequacy," in *Proceedings of the 41st International Conference on Software Engineering*, ser. ICSE '19. IEEE Press, 2019, p. 1039–1049.

[2] "Mnist-fashion," https://github.com/zalandoresearch/fashion-mnist.

[3] E. Merlo *et al.*, "Models of computational profiles to study the likelihood of dnn metamorphic test cases," *IMLSE '20*, 2020. [Online]. Available: https://drive.google.com/file/d/12TMkERu0d85h_fCPx7FzYm0f1yUhZaia/view

[4] M. Marhaba *et al.*, "Identification of out-of-distribution cases of cnn using class-based surprise adequacy," *ICSE/CAIN '22*, 2022.

[5] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: A new approach for generating next test cases," *Department of Computer Science, The Hong Kong University of Science and Technology*, 1998.

[6] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *ICLR '15*, 2015.

[7] N. Papernot *et al.*, "The limitations of deep learning in adversarial settings," *2016 IEEE European Symposium on Security and Privacy*, 2016.

[8] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," *CVPR '16*, 2016.

[9] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," 2016. [Online]. Available: https://arxiv.org/abs/1608.04644

[10] N. Papernot *et al.*, "Distillation as a defense to adversarial perturbations against deep neural networks," *CoRR*, vol. abs/1511.04508, 2015. [Online]. Available: http://arxiv.org/abs/1511.04508

[11] N. Papernot *et al.*, "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 582–597.

[12] N. Papernot and P. D. McDaniel, "Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning," *CoRR*, vol. abs/1803.04765, 2018. [Online]. Available: http://arxiv.org/abs/1803.04765

[13] Y. Xiao *et al.*, "Self-checking deep neural networks in deployment," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, May 2021, pp. 372–384.

[14] A. Guerriero, R. Pietrantuono, and S. Russo, "Operation is the hardest teacher: Estimating dnn accuracy looking for mispredictions," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, May 2021, pp. 348–358.

[15] R. Mangal, A. V. Nori, and A. Orso, "Robustness of neural networks: a probabilistic and practical approach," in *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results, ICSE (NIER) 2019, Montreal, QC, Canada, May 29-31, 2019*, A. Sarma and L. Murta, Eds. IEEE / ACM, 2019, pp. 93–96. [Online]. Available: https://doi.org/10.1109/ICSE-NIER.2019.00032

[16] Z. Zhou *et al.*, "Deepcon: Contribution coverage testing for deep learning systems," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2021, pp. 189–200.

[17] M. Wardat, W. Le, and H. Rajan, "Deeplocalize: Fault localization for deep neural networks," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2021, pp. 251–262. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICSE43902.2021.00034

[18] Z. Li *et al.*, "Structural coverage criteria for neural networks could be misleading," in *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results*, ser. ICSE-NIER '19. IEEE Press, 2019, p. 89–92. [Online]. Available: https://doi.org/10.1109/ICSE-NIER.2019.00031

[19] F. Harel-Canada *et al.*, "Is Neuron Coverage a Meaningful Measure for Testing Deep Neural Networks?" in *28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020.

[20] H. Ferit Eniser, M. Christakis, and V. Wüstholz, "RAID: Randomized Adversarial-Input Detection for Neural Networks," *arXiv e-prints*, p. arXiv:2002.02776, Feb. 2020.

[21] K. Pei *et al.*, "Deepxplore: Automated whitebox testing of deep learning systems," *Commun. ACM*, vol. 62, no. 11, p. 137–145, Oct. 2019. [Online]. Available: https://doi.org/10.1145/3361566

[22] L. Ma *et al.*, "Deepgauge: multi-granularity testing criteria for deep learning systems," *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, Sep 2018. [Online]. Available: http://dx.doi.org/10.1145/3238147.3238202

[23] Y. Kaya, S. Hong, and T. Dumitras, "Shallow-deep networks: Understanding and mitigating network overthinking," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 3301–3310. [Online]. Available: http://proceedings.mlr.press/v97/kaya19a.html

[24] L. Engstrom *et al.*, "A rotation and a translation suffice: Fooling CNNs with simple transformations," 2019. [Online]. Available: https://openreview.net/forum?id=BJfvknCqFQ

[25] S. Gu and L. Rigazio, "Towards deep neural network architectures robust to adversarial examples," 2015.

[26] K. Pei *et al.*, "Towards practical verification of machine learning: The case of computer vision systems," 2017.

[27] H. Wang *et al.*, "Dissector: Input validation for deep learning applications by crossing-layer dissection," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, Oct 2020, pp. 727–738.

[28] Z. Gong, W. Wang, and W. Ku, "Adversarial and clean data are not twins," *CoRR*, vol. abs/1704.04960, 2017. [Online]. Available: http://arxiv.org/abs/1704.04960

[29] K. Grosse *et al.*, "On the (statistical) detection of adversarial examples," *CoRR*, vol. abs/1702.06280, 2017. [Online]. Available: http://arxiv.org/abs/1702.06280

[30] J. H. Metzen *et al.*, "On detecting adversarial perturbations," in *Proceedings of 5th International Conference on Learning Representations (ICLR)*, 2017.

[31] A. Bhagoji, D. Cullina, and P. Mittal, "Dimensionality reduction as a defense against evasion attacks on machine learning classifiers," *ArXiv*, vol. abs/1704.02654, 2016.

[32] D. Hendrycks and K. Gimpel, "Early methods for detecting adversarial images," 2017.

[33] X. Li and F. Li, "Adversarial examples detection in deep networks with convolutional filter statistics," *CoRR*, vol. abs/1612.07767, 2016. [Online]. Available: http://arxiv.org/abs/1612.07767

[34] R. Feinman *et al.*, "Detecting adversarial samples from artifacts," 2017.

[35] N. Carlini and D. A. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," *CoRR*, vol. abs/1705.07263, 2017. [Online]. Available: http://arxiv.org/abs/1705.07263

[36] S. Ma *et al.*, "NIC: detecting adversarial samples with neural network invariant checking," in *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019.

[37] J. Wang *et al.*, "Adversarial sample detection for deep neural network through model mutation testing," in *Proceedings of the 41st International Conference on Software Engineering*, ser. ICSE '19. IEEE Press, 2019, p. 1245–1256.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira *et al.*, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[39] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[40] K. He *et al.*, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[41] C. Zhang *et al.*, "Understanding deep learning requires rethinking generalization," *ArXiv*, vol. abs/1611.03530, 2017.

[42] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," 2015.

[43] C. Szegedy *et al.*, "Intriguing properties of neural networks," *CoRR*, vol. abs/1312.6199, 2014.

[44] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," *CoRR*, vol. abs/1610.02136, 2016. [Online]. Available: http://arxiv.org/abs/1610.02136

[45] S.-M. Moosavi-Dezfooli *et al.*, "Universal adversarial perturbations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[46] M. A. F. Pimentel *et al.*, "A review of novelty detection," *Signal Process.*, vol. 99, pp. 215–249, 2014.

[47] P. Vincent and Y. Bengio, "Manifold parzen windows," in *Proceedings of the 15th International Conference on Neural Information Processing Systems*, ser. NIPS'02. Cambridge, MA, USA: MIT Press, 2002, p. 849–856.

[48] A. Ghoting, S. Parthasarathy, and M. Otey, "Fast mining of distance-based outliers in high-dimensional datasets," *Data Mining and Knowledge Discovery*, vol. 16, pp. 349–364, 06 2008.

[49] L. Devroye, L. Györfi, and G. Lugosi, *A Probabilistic Theory of Pattern Recognition*, ser. Stochastic Modelling and Applied Probability. Springer New York, 2013. [Online]. Available: https://books.google.ca/books?id=Y5bxBwAAQBAJ

[50] L. Wasserman, *All of Nonparametric Statistics (Springer Texts in Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[51] L. Theis, A. van den Oord, and M. Bethge, "A note on the evaluation of generative models," 2017.

[52] T. Schlegl *et al.*, "Unsupervised anomaly detection with generative adversarial networks to guide marker discovery," 2017.

[53] H. Delseny *et al.*, "White paper machine learning in certified systems," 2021.

[54] F. Tambon *et al.*, "How to certify machine learning based safety-critical systems? a systematic literature review," 2021.

[55] O. Strauss, F. Comby, and M. . Aldon, "Rough histograms for robust statistics," in *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, vol. 2, 2000, pp. 684–687 vol.2.

[56] "Tensorflow - lenet," https://github.com/tensorflow/models/blob/master/research/slim/nets/lenet.py

[57] "Adversarial robustness 360 toolbox." [Online]. Available: https://developer.ibm.com/technologies/analytics/projects/adversarial-robustness-toolbo

[58] N. Papernot *et al.*, "Technical report on the cleverhans v2.1.0 adversarial examples library," *arXiv preprint arXiv:1610.00768*, 2018.

[59] P. Virtanen *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," pp. 261–272, 2020.

[60] N. Ernst, "cliffsdelta," https://github.com/neilernst/cliffsDelta, 2019.

[61] "Marco Torchiano - Effect size implementation in R." [Online]. Available: https://github.com/mtorchiano/effsize

[62] M. Hess and J. Kromrey, "Robust confidence intervals for effect sizes: A comparative study of cohen's d and cliff's delta under non-normality and heterogeneous variances," *Paper Presented at the Annual Meeting of the American Educational Research Association*, 01 2004.

[63] J. Romano *et al.*, "Appropriate statistics for ordinal level data: Should we really be using t-test and cohen's d for evaluating group differences on the nsse and other surveys?" in *Annual meeting of the Florida Association of Institutional Research*, 2006.

APPENDIX A LINEAR BEST SEPARABILITY TABLES

This section includes all the detailed separation tables for the Per-Set Best Separation results mentioned in section 5.2.1.

Table A.1 Linear Best Separability - Carlini Wagner - Classes 0, 1, 2

True Class	Best Class	Separation Point	Train Separation	CW Separation
0	0	486.413	0.6673	0.675
	1	881.088	0.9643	1
	2	501.9865	0.7389	0.7399
	3	543.0706	0.6203	0.6207
	4	588.5641	0.8446	0.8485
	5	751.5596	0.8749	0.875
	6	472.5352	0.5228	0.5228
	7	1589.3591	0.9998	1
	8	639.5854	0.7498	0.75
	9	1203.2364	0.9998	1
1	0	754.4971	0.9095	0.9096
	1	484.3397	0.5378	0.5381
	2	757.9953	0.9428	0.9429
	3	669.1065	0.8479	0.848
	4	663.8182	0.9028	0.9068
	5	823.4882	0.943	1
	6	743.5708	0.9073	0.9268
	7	-	-	-
	8	767.7848	0.9285	0.9286
	9	-	-	-
2	0	574.6692	0.7744	0.7746
	1	1270.8184	0.9975	1
	2	513.8088	0.7565	1
	3	609.765	0.7691	0.7692
	4	434.125	0.553	0.5534
	5	637.4314	0.6665	0.6667
	6	462.7645	0.5024	0.5026
	7	1562.9799	0.9998	1
	8	661.3385	0.795	0.8214
	9	-	-	-

Table A.2 Linear Best Separability - Carlini Wagner - Classes 3, 4, 5

True Class	Best Class	Separation Point	Train Separation	CW Separation
3	0	586.5664	0.7873	0.7875
	1	813.5144	0.9488	0.949
	2	632.3241	0.8816	0.8828
	3	548.2284	0.6358	0.6366
	4	588.8298	0.8449	0.8453
	5	815.773	0.9375	0.9375
	6	628.055	0.7963	0.7967
	7	1271.0368	0.9978	1
	8	696.9212	0.8602	0.8602
	9	-	-	-
4	0	709.192	0.8832	0.8833
	1	898.3938	0.9687	0.9765
	2	452.556	0.6476	0.6478
	3	532.569	0.5929	0.593
	4	530.3369	0.767	1
	5	772.1786	0.8993	1
	6	482.1886	0.5425	0.5427
	7	1589.3591	0.9998	1
	8	608.0472	0.6665	0.6667
	9	-	-	-
5	0	1078.5612	0.9831	0.9867
	1	1697.4489	0.9998	1
	2	1081.5074	0.9914	1
	3	1059.6019	0.995	1
	4	1175.0918	0.9964	1
	5	583.4189	0.5244	0.5248
	6	1006.5368	0.9871	0.9884
	7	675.2723	0.9064	0.9065
	8	785.9562	0.9412	0.9412
	9	667.1461	0.941	0.9412

Table A.3 Linear Best Separability - Carlini Wagner - Classes 6, 7

True Class	Best Class	Separation Point	Train Separation	CW Separation
6	0	480.3539	0.655	0.6551
	1	1028.939	0.9858	1
	2	473.3951	0.6912	0.6916
	3	572.9218	0.6969	0.6984
	4	438.5021	0.5662	0.5662
	5	857.7804	0.9613	1
	6	630.2202	0.7999	0.8
	7	1297.5639	0.9985	1
	8	661.0956	0.794	0.7941
	9	-	-	-
7	0	840.9614	0.9383	0.9455
	1	1930.2526	0.9998	1
	2	913.3432	0.9761	0.9875
	3	813.2522	0.9561	0.9561
	4	1045.5479	0.9919	1
	5	584.9735	0.5283	0.5284
	6	805.7899	0.9419	0.9458
	7	487.1605	0.6108	0.6113
	8	692.9856	0.8545	0.8546
	9	527.8497	0.7598	0.7599

Table A.4 Linear Best Separability - Carlini Wagner - Classes 8, 9

True Class	Best Class	Separation Point	Train Separation	CW Separation
8	0	895.0191	0.9544	0.9554
	1	1374.7135	0.9992	1
	2	684.1054	0.9133	0.9143
	3	732.9613	0.9102	0.9103
	4	696.3801	0.9193	0.9226
	5	687.5396	0.7737	0.7738
	6	759.2384	0.9169	0.9181
	7	721.4962	0.9319	0.9321
	8	561.8863	0.5033	0.5034
	9	682.378	0.9483	0.9583
9	0	1006.2344	0.9757	0.9817
	1	1161.5854	0.9938	1
	2	805.6243	0.9534	1
	3	781.3352	0.9391	0.9565
	4	875.2234	0.977	1
	5	664.4764	0.729	0.7292
	6	867.5256	0.965	0.9659
	7	612.7433	0.8601	0.8601
	8	683.6053	0.8412	0.875
	9	455.8203	0.5346	0.5348

Table A.5 Linear Best Separability - Fast Gradient - Classes 0, 1, 2

True Class	Best Class	Separation Point	Train Separation	FGRAD Separation
0	0	1138.3257	0.9888	1
	1	1930.2526	0.9998	1
	2	1071.8412	0.9912	1
	3	1398.0666	0.9997	1
	4	1151.1303	0.9959	1
	5	1254.7209	0.9998	1
	6	936.3612	0.9782	0.9783
	7	-	-	-
	8	880.9739	0.9782	0.9783
	9	-	-	-
1	0	1141.6702	0.9891	0.9892
	1	1740.7595	0.9998	1
	2	1125.596	0.9932	1
	3	1052.7765	0.995	0.9954
	4	927.48	0.9827	0.9828
	5	1254.7209	0.9998	1
	6	953.5616	0.9805	0.9805
	7	-	-	-
	8	904.371	0.9835	0.9844
	9	-	-	-
2	0	1318.4542	0.9956	1
	1	1930.2526	0.9998	1
	2	1582.5551	0.9998	1
	3	1256.8887	0.9995	1
	4	1039.3327	0.9919	0.9927
	5	1231.0216	0.9998	1
	6	977.355	0.9832	0.9833
	7	-	-	-
	8	902.4543	0.9832	0.9833
	9	-	-	-

Table A.6 Linear Best Separability - Fast Gradient - Classes 3, 4, 5

True Class	Best Class	Separation Point	Train Separation	FGRAD Separation
3	0	1173.698	0.9904	0.991
	1	1930.2526	0.9998	1
	2	1058.8121	0.9905	1
	3	1185.2537	0.9985	1
	4	950.5759	0.9849	0.985
	5	-	-	-
	6	900.9432	0.9726	0.9727
	7	-	-	-
	8	899.0744	0.9828	0.9832
	9	-	-	-
4	0	1205.4373	0.9926	0.9944
	1	1930.2526	0.9998	1
	2	1059.0793	0.9905	0.9917
	3	1258.1349	0.9995	1
	4	1046.3609	0.9921	1
	5	-	-	-
	6	980.9563	0.9837	0.9839
	7	-	-	-
	8	886.2017	0.9807	0.9839
	9	-	-	-
5	0	1317.7125	0.9956	0.997
	1	1930.2526	0.9998	1
	2	1347.031	0.9992	1
	3	1484.1148	0.9998	1
	4	1387.5193	0.9995	1
	5	899.1557	0.975	0.975
	6	1111.1435	0.9943	0.9945
	7	1589.3591	0.9998	1
	8	893.8664	0.9818	0.982
	9	1203.2364	0.9998	1

Table A.7 Linear Best Separability - Fast Gradient - Classes 6, 7

True Class	Best Class	Separation Point	Train Separation	FGRAD Separation
6	0	1162.7623	0.9898	0.9898
	1	1930.2526	0.9998	1
	2	1174.0449	0.9958	0.9959
	3	1146.5674	0.9977	1
	4	1064.8951	0.9927	0.9928
	5	1052.4563	0.9972	1
	6	1006.0901	0.9871	0.9873
	7	-	-	-
	8	890.3251	0.9812	0.9812
	9	-	-	-
7	0	1270.2336	0.9945	1
	1	1930.2526	0.9998	1
	2	1373.2906	0.9992	1
	3	1304.1733	0.9995	1
	4	1533.4598	0.9998	1
	5	891.2279	0.9737	0.9737
	6	1052.0699	0.9906	0.9907
	7	-	-	-
	8	891.5769	0.9813	0.9815
	9	1203.2364	0.9998	1

Table A.8 Linear Best Separability - Fast Gradient - Classes 8, 9

True Class	Best Class	Separation Point	Train Separation	FGRAD Separation
8	0	1279.8113	0.9946	0.9958
	1	1930.2526	0.9998	1
	2	1299.6044	0.9988	1
	3	1484.1148	0.9998	1
	4	1115.8472	0.9953	0.9953
	5	905.0185	0.9772	0.9773
	6	1002.946	0.9866	0.9867
	7	1589.3591	0.9998	1
	8	892.3893	0.9813	0.9814
	9	1203.2364	0.9998	1
9	0	1474.9484	0.9988	1
	1	1930.2526	0.9998	1
	2	1579.0507	0.9998	1
	3	1484.1148	0.9998	1
	4	1330.8115	0.999	1
	5	934.1037	0.9843	0.9915
	6	1050.1007	0.9904	0.9905
	7	1589.3591	0.9998	1
	8	864.986	0.9742	0.9743
	9	1203.2364	0.9998	1

Table A.9 Linear Best Separability - DeepFool - Classes 0, 1, 2

True Class	Best Class	Separation Point	Train Separation	DF Separation
0	0	397.3051	0.4998	0.5
	1	830.0255	0.952	1
	2	476.9613	0.6986	0.6987
	3	517.8053	0.5457	0.5464
	4	551.8951	0.8003	0.8235
	5	1240.4871	0.9998	1
	6	586.2332	0.7336	0.7336
	7	-	-	-
	8	818.7998	0.9587	0.9588
	9	-	-	-
1	0	603.0585	0.8034	0.8051
	1	480.6315	0.5561	0.5563
	2	764.0869	0.9443	0.9444
	3	630.5862	0.7988	0.7989
	4	583.0866	0.8392	0.8393
	5	-	-	-
	6	818.3586	0.9483	0.9483
	7	-	-	-
	8	848.2412	0.9695	0.9695
	9	-	-	-
2	0	599.8891	0.8012	0.8057
	1	1366.2107	0.999	1
	2	378.6778	0.4695	0.4696
	3	559.5572	0.6632	0.6719
	4	425.5947	0.5243	0.5243
	5	-	-	-
	6	609.1533	0.768	0.7682
	7	-	-	-
	8	848.6935	0.9698	0.97
	9	-	-	-

Table A.10 Linear Best Separability - DeepFool - Classes 3, 4, 5

True Class	Best Class	Separation Point	Train Separation	DF Separation
3	0	513.6662	0.7038	0.7039
	1	727.7063	0.916	0.916
	2	531.8012	0.7824	0.7863
	3	544.1488	0.6236	0.6237
	4	507.9933	0.7267	0.7292
	5	1254.7209	0.9998	1
	6	677.6648	0.8499	0.8499
	7	-	-	-
	8	829.7965	0.9622	0.9622
	9	-	-	-
4	0	712.719	0.8859	0.8955
	1	744.9231	0.924	1
	2	409.7061	0.5484	0.5485
	3	497.6689	0.4807	0.4821
	4	425.8125	0.5251	0.5253
	5	-	-	-
	6	625.1961	0.7915	0.7915
	7	-	-	-
	8	837.4417	0.9655	0.9672
	9	-	-	-
5	0	933.3215	0.962	0.9655
	1	1930.2526	0.9998	1
	2	1273.4437	0.9983	1
	3	879.8979	0.977	1
	4	1428.5297	0.9998	1
	5	590.2318	0.5439	0.5441
	6	1008.298	0.9875	0.9877
	7	601.2628	0.846	0.8462
	8	774.2352	0.933	0.9331
	9	625.0678	0.908	0.908

Table A.11 Linear Best Separability - DeepFool - Classes 6, 7

True Class	Best Class	Separation Point	Train Separation	DF Separation
6	0	482.4215	0.6607	0.6608
	1	1009.1897	0.9837	1
	2	457.4047	0.659	0.659
	3	555.4727	0.6544	0.6583
	4	473.9152	0.6521	0.6523
	5	1254.7209	0.9998	1
	6	589.802	0.7383	0.7384
	7	-	-	-
	8	833.7191	0.9642	0.9642
	9	-	-	-
7	0	1006.9874	0.976	0.9789
	1	1930.2526	0.9998	1
	2	858.8008	0.9677	0.9765
	3	776.0027	0.9345	0.9347
	4	1159.5345	0.9963	1
	5	574.1013	0.4981	0.4982
	6	928.5942	0.9776	0.9778
	7	527.142	0.7226	0.7226
	8	744.7976	0.9128	0.913
	9	556.9501	0.8207	0.8208

Table A.12 Linear Best Separability - DeepFool - Classes 8, 9

True Class	Best Class	Separation Point	Train Separation	DF Separation
8	0	801.1912	0.9264	0.9265
	1	1621.4893	0.9998	1
	2	599.4261	0.8566	0.8667
	3	703.9217	0.8843	0.8919
	4	652.8615	0.896	0.8992
	5	655.9069	0.7082	0.7083
	6	779.5464	0.9303	0.9304
	7	761.1703	0.9481	0.9483
	8	568.2715	0.5287	0.5291
	9	654.0213	0.9304	0.9306
9	0	876.9131	0.9506	0.9556
	1	1131.5011	0.9932	1
	2	643.3476	0.8888	0.8889
	3	925.0744	0.9846	1
	4	713.9375	0.9285	0.9286
	5	599.3681	0.5713	0.5714
	6	812.4309	0.9448	0.9449
	7	559.4547	0.788	0.7886
	8	732.0642	0.902	0.9022
	9	471.336	0.5931	0.5931

Table A.13 Linear Best Separability - Jacobian Saliency Map - Classes 0, 1, 2

True Class	Best Class	Separation Point	Train Separation	JSMA Separation
0	0	397.5314	0.5007	0.5007
	1	703.2719	0.9032	0.9033
	2	463.6105	0.6715	0.6716
	3	537.9626	0.6059	0.606
	4	525.7214	0.7604	0.7604
	5	579.1344	0.5123	0.5123
	6	465.6688	0.5089	0.509
	7	633.958	0.8789	0.879
	8	564.5886	0.514	0.5141
	9	615.9107	0.9	0.9001
1	0	831.7829	0.9355	0.9355
	1	495.5164	0.5029	0.5031
	2	860.8222	0.9682	0.9682
	3	742.3261	0.9178	0.918
	4	813.7388	0.9647	0.9648
	5	784.5104	0.912	0.9121
	6	803.4066	0.9411	0.9411
	7	931.6438	0.9828	0.983
	8	778.2969	0.9358	0.9359
	9	919.686	0.9949	0.995
2	0	528.217	0.7211	0.7212
	1	774.9406	0.937	0.937
	2	390.5676	0.5005	0.5006
	3	592.9788	0.7402	0.7403
	4	459.1883	0.623	0.6231
	5	599.5464	0.5716	0.5716
	6	462.9844	0.5029	0.503
	7	689.631	0.9144	0.9145
	8	578.8158	0.5685	0.5687
	9	616.8163	0.9002	0.9003

Table A.14 Linear Best Separability - Jacobian Saliency Map - Classes 3, 4, 5

True Class	Best Class	Separation Point	Train Separation	JSMA Separation
3	0	542.0361	0.7385	0.7385
	1	706.7524	0.9048	0.905
	2	576.6976	0.8375	0.8375
	3	503.5266	0.5007	0.5008
	4	563.2345	0.8152	0.8152
	5	616.7371	0.6156	0.6157
	6	565.6911	0.6998	0.6999
	7	659.3092	0.8959	0.896
	8	609.7839	0.6728	0.673
	9	703.4379	0.9572	0.9572
4	0	559.6922	0.757	0.757
	1	756.7171	0.9292	0.9292
	2	461.4536	0.6669	0.6671
	3	562.5864	0.6699	0.6701
	4	418.4148	0.4999	0.5003
	5	602.9499	0.5796	0.5798
	6	457.222	0.4892	0.4892
	7	668.1598	0.9011	0.9012
	8	559.6019	0.4943	0.4945
	9	600.4999	0.8838	0.884
5	0	1011.8756	0.9764	0.9765
	1	1276.6832	0.9977	0.9978
	2	1032.5779	0.9883	0.9885
	3	912.4925	0.983	0.9831
	4	973.5321	0.9883	0.9885
	5	575.2078	0.5013	0.5016
	6	931.3186	0.9779	0.978
	7	735.8141	0.9379	0.9381
	8	747.5837	0.9145	0.9146
	9	755.3217	0.9739	0.974

Table A.15 Linear Best Separability - Jacobian Saliency Map - Classes 6, 7

True Class	Best Class	Separation Point	Train Separation	JSMA Separation
6	0	523.5387	0.7164	0.7165
	1	744.9751	0.924	0.9241
	2	474.1108	0.6922	0.6923
	3	584.34	0.7238	0.724
	4	487.0512	0.6824	0.6825
	5	609.8284	0.5966	0.5966
	6	461.7968	0.5002	0.5003
	7	675.398	0.9066	0.9067
	8	581.7034	0.5795	0.5796
	9	615.4827	0.8995	0.8996
7	0	826.7217	0.934	0.9341
	1	1123.2175	0.9923	0.9925
	2	844.6525	0.9646	0.9647
	3	805.3618	0.9506	0.9507
	4	784.0894	0.956	0.9561
	5	562.3694	0.5418	0.5419
	6	797.0386	0.9388	0.939
	7	452.6432	0.5023	0.5026
	8	633.4204	0.7368	0.7369
	9	587.8089	0.8686	0.8688

Table A.16 Linear Best Separability - Jacobian Saliency Map - Classes 8, 9

True Class	Best Class	Separation Point	Train Separation	JSMA Separation
8	0	821.2453	0.9323	0.9323
	1	1087.2327	0.9895	0.9896
	2	726.8218	0.9318	0.9319
	3	771.1901	0.9325	0.9327
	4	735.4302	0.9378	0.9379
	5	693.8133	0.7844	0.7845
	6	726.5925	0.8955	0.8956
	7	783.8004	0.9563	0.9564
	8	561.2151	0.5005	0.5007
	9	688.0899	0.9505	0.9505
9	0	871.8646	0.9492	0.9493
	1	1117.0811	0.9917	0.9918
	2	799.3414	0.9529	0.953
	3	806.9166	0.9519	0.9519
	4	791.4185	0.9574	0.9574
	5	623.7709	0.6316	0.6317
	6	799.9306	0.9399	0.9399
	7	633.1024	0.8781	0.8782
	8	597.2288	0.6318	0.632
	9	447.5632	0.503	0.503

APPENDIX B ARCHITECTURE STRUCTURES

Table B.1 Different architecture structures used

v0	conv 2d (32 filters, 5x5), max pooling (2x2), conv 2d (64 filters, 5x5), max pooling (2x2), flatten, dense (1024, relu), dense (10, softmax)
v1	conv 2d (16 filters, 5x5), max pooling (2x2), conv 2d (32 filters, 5x5), max pooling (2x2), flatten, dense (512, relu), dense (10, softmax)
v2	conv 2d (32 filters, 3x3), max pooling (2x2), flatten, dense (128, relu), dense (10, softmax)
v3	conv 2d (32 filters, 3x3), conv 2d (64 filters, 3x3), max pooling (2x2), dropout (0.5), conv 2d (128 filters, 3x3), max pooling (2x2), dropout (0.5), flatten, dense (256, relu), dense (10, softmax)
v4	conv 2d (16 filters, 3x3), conv 2d (32 filters, 3x3), max pooling (2x2), dropout (0.5), conv 2d (64 filters, 3x3), max pooling (2x2), dropout (0.5), flatten, dense (128, relu), dense (10, softmax)
v5	conv 2d (32 filters, 5x5), max pooling (2x2), conv 2d (64 filters, 5x5), max pooling (2x2), flatten, dense (1024, relu), dense (1024, relu), dense (10, softmax)
v6	conv 2d (16 filters, 5x5), max pooling (2x2), conv 2d (32 filters, 5x5), max pooling (2x2), flatten, dense (512, relu), dense (512, relu), dense (10, softmax)
v7	conv 2d (32 filters, 3x3), max pooling (2x2), flatten, dense (128, relu), dense (128, relu), dense (10, softmax)
v8	conv 2d (32 filters, 3x3), conv 2d (64 filters, 3x3), max pooling (2x2), dropout (0.5), conv 2d (128 filters, 3x3), max pooling (2x2), dropout (0.5), flatten, dense (256, relu), dense (256, relu), dense (10, softmax)
v9	conv 2d (16 filters, 3x3), conv 2d (32 filters, 3x3), max pooling (2x2), dropout (0.5), conv 2d (64 filters, 3x3), max pooling (2x2), dropout (0.5), flatten, dense (128, relu), dense (128, relu), dense (10, softmax)