SC2

FYP Proposal

Deep Testing of Advanced Learning Systems

by

Jung Chan

SC2

Advised by

Prof. Shi-Ching, Chueng

Submitted in partial fulfillment

of the requirements for COMP 4982

in the

Department of Computer Science

The Hong Kong University of Science and Technology

2018-2019

Date of submission: April 17, 2019

Abstract

While deep learning techniques may have evolved from statistical analysis, there is, to the best of our knowledge, little research into applying statistical methods into the study of deep learning systems. The testing of deep neural networks is also a relatively new field, with few up to date, comprehensive surveys in the topic.

This thesis presents a comprehensive survey of the field of deep neural network testing, with additional detail in related research, like adversarial generation. This thesis also explores applying statistical analysis of the neuron outputs of certain layers of a deep neural networks, with some surprising insights being gleaned. Using this statistical information, we implement a novel iteration based on existing tools for testing and evaluating deep neural network. This novel tool is compared with existing methods, showing that it is more effective within certain use cases. Finally, we suggest an implementation of this tool for the DeepXplore algorithm.

The code for the project may be found here:

https://github.com/j-chan-hkust/deep_testing_of_advanced_learning_systems

Acknowledgements

I would like to express my gratitude to Professor Shing-Chi Cheung, for his guidance on this FYT; as well as Victor Yongqiang Tian for his technical guidance on the implementation of this FYT; without which this FYT would not be possible.

Table of Contents

1 Introduction

1.1 Overview

DNNs – Machine learning and AI are two terms that are rather overused and misunderstood by the general public. Machine learning refers to the field of computer science that uses statistical techniques to train self-improving systems to solve specific problems without explicit human instructions, instead they rely on statistical models and inference [1]. While there are many different approaches to achieving this, one of the most common approaches – in both research and commercial usage – are variants of deep neural networks (DNNs), or deep learning systems.

In the past few years, research in DNNs and commercial usage of DNNs have soared. DNNs have long shown their usefulness in numerous practical applications (OCR [2], speech-to-text [3], autonomous vehicles [4], Go [5], Starcraft 2 [6], to name a few). More recently, DNNs have become commercially available through the cloud platforms provided by huge companies like Amazon [7], Microsoft [8], and Google [9]. At this point, their ability to outperform human-written code in certain applications is undeniable.

DNN Testing – As the technology has matured, research work in DNNs has gradually shifted from questions of feasibility to questions of reliability. Much of the difficulty that comes with testing these systems is built into the nature of DNNs. Because no human intervention is present during the training of DNNs, it can be very difficult to understand how the data is being processed by the network. It is practically impossible to test all the huge input space for erroneous behavior [10]. Thus, we arrive at our core question. If we do not completely understand how input data is being processed, and it is impossible to test everything, how can you systematically test DNNs?

1.2 Objectives

This final year thesis aims to fulfill the goals outlined in the previous progress report: "To produce a useful generalized framework for DNN testing", and "To improve/build upon existing methodology with new tools/techniques". To achieve these goals, this final year thesis presents the following contributions.

To fulfill the first goal, a comprehensive survey of the current state of research into adversarial attack and DNN testing and will be mostly discussed in section 1.3, the literature survey. Hopefully it will present the reader with significant insight into the similarities in techniques used by modern researchers.

The second question will be covered in the remainder of this report. Building upon the insights from the literature survey, this paper proposes new tools to better evaluate, test and understand DNNs. This paper presents the following: tools to aggregate the outputs of individual neurons in a DNN layer across the entire training dataset; statistical analysis tools for interpreting these outputs; adapting this statistical analysis to iterate on well-established DNN evaluation criterion (for example: DeepTest and DeepXplore; see: neuron coverage [10, 11]); the evaluation of the effectiveness of this new DNN evaluation criterion; and finally, present an example of how this coverage criterion may be used to guide adversarial generation.

1.3 Literature Survey

The topic of DNN testing is a fast-growing subject with new research being added every month, and as such the scope of my research has expanded a lot since the last report. This literature survey is organized as such. Macro-level concepts are introduced first, giving a brief overview of core concepts used for the Design. As we move down, more micro-level concepts are discussed. Finally, papers which explore interesting concepts but happen to be orthogonal to our work are discussed.

It is only necessary to read through the subheadings 1.3.1 -1.3.9 to gain enough context to understand the work in the methodology segment later.

1.3.1 Deep Neural Networks

DNNs are software structures consisting of layers, each layer containing of multiple neurons. Each neuron applies an activation function to the sum of its inputs (if the neurons are not on the first layer, the inputs are the output of neurons in the previous layer), and the output is passed to the neuron on the next layer. Each connection between each neuron has an associated weight and bias. The main activation functions are sigmoid, hyperbolic tangent and ReLU [12].

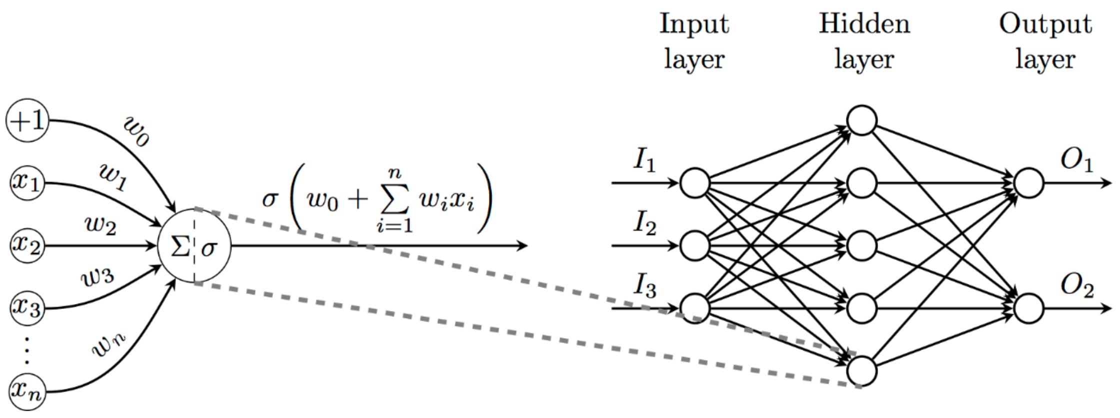


Figure 1: A simple DNN, with more detail on the neuron

The network is tested against the training dataset and evaluated using a loss function. To train the network, gradient descent using backpropagation is used [13] to minimize the loss function.

1.3.2 Adversarial Attack

Adverserial attack is a popular research topic for DNNs. The technique takes advantage of the fact that DNNs allow for easy gradient calcuulation, thereby making it easy to explore the input space for misclassified images [14] [15] [16]. Some versions may begin with a valid input, where others may begin from scratch. These attacks are difficult to defend against because they make use of inherent properties of DNNs, namely the existent of gradients.

Adverserial attack is shown to work even when the resultant adverserial image is printed, and scanned with a phone [17]. This is shown to work even when the DNN is made to not show certainty/masked (thus removing direct access gradient ascent), because adversarial inputs obtained for different models that perform a similar function also produce erroneous behavior in the masked DNN [18].

Two proposed defences are defensive distillation [19] – outputing multiple classes instead of a single output, something that doesn't quite work for autonomous steering – and adverserial training – using the generated adverserial data to train the model – an idea that is used in many other papers [10] [20].

This idea of using adverserial attack to produce "difference-inducing inputs" is quite common throughout the research in the field, and forms the core structure of many of the papers discussed.

The core flow of this is we have a heruristic, which we use to guide a generation method, which can be additive functions inputs [21], or single pixels changes [16]. We will be discussing other generation methods later on.

1.3.3 Decision Boundaries

For neural networks that output a single output, decision boundaries between two outputs often have rougher gradients, meaning erroneous behaviors are more common. Also, models of similar function will have similar (though not identical) decision boundaries. Erroneous behaviors are thought to lie around decision boundaries. However, my research has not explored the nature of decision boundaries are like for models that have continuous outputs (like autonomous vehicle steering). It will be worth doing further research into the topic.

1.3.4 Metamorphic Relations

There is this idea that certain transforms (like image rotation, mirroring, color inversion), while they can significantly alter the image (increase the distance between the images), they do not change the structure of the data. It follows that this changed data can be used to train a new model, and this new model will be able to successfully categorize the transformed inputs. Mislabeled or dummy training data, however will not be correctly categorized when transformed, as their structure is not consistent with their labeling. Thus, we can use these to clean our datasets of dirty data. These ideas are presented in the paper by Dwarakanath et al. [22], and concepts from metamorphic testing – using the idea that certain transforms to the input should alter the output of a system – are referenced in DeepTest [20], DeepRoad [23], and others [11] [16] [24].

1.3.5 Main Behaviors Vs. Corner Case Behaviors

One dichotomy that is presented in DeepGauge [11] is main behaviors – the input space that is covered by the training data – and corner-case behavior – input space that is not covered by training data. While adversarial inputs can lie in both the spaces, different testing criterion prioritize different spaces, for example neuron coverage favors main behaviors, whereas criterion like K-multisection Neuron Coverage [11] favor corner-case behaviors. DeepHunter [25] suggests that more adversarial inputs can be found by exploring corner-case behaviors, which is something that follows logically. If a certain input/behavior set is not trained for, it seems natural that the DNN would be more likely to make a mistake.

1.3.6 Traditional Software Engineering and DNN Software Engineering

Many of the papers we will be discussing use traditional software testing criterions and techniques, then attempt to adapt the essence of these methods to DNNs. Forms of code coverage, taken from traditional programming are adapted to DNNs. Test case generation techniques, like concolic testing and fuzzing have also been adapted.

1.3.7 Neuron Coverage

The team behind DeepXplore [10] proposed a testing criterion. Adapted from the concept of "code coverage" in traditional programming, they introduced "neuron coverage" a measure of how much of the model's neurons had been used. Neuron coverage is calculated using the following method: for a certain input, if some neurons exceeds a certain activation threshold, we count them as being "activated". Over the entire test set, we see how many neurons in the DNN have been activated at least once. They show that existing methods of adversarial attack generation do not produce high neuron coverage. In using this criterion in conjunction with adversarial attacks' difference inducing gradient ascent, we produce adversarial inputs that also maximize neuron coverage. This concept of neuron coverage is used in many proceeding papers, however, there are some key critiques. Some papers found that neuron coverage was easily satisfied with a few images [24] [25]. DeepGauge [11] posits that the neuron coverage is too abstracted and does not take into account the statistical properties of the activation function, i.e. some neurons are almost always highly activated, some are difficult to activate, and this is not reflected in the coverage.

1.3.8 Further Coverage Criterion

Many different papers have continued to propose newer criterions. DeepGauge [11] considered many multi-granular coverage criterions that were more difficult to satisfy. Other papers have also shown other coverage criterion, many inspired by existing software testing methods, like MC/DC [26], Lipschitz continuity [27] and combinatorial testing [28]. The paper DeepHunter [25] has implemented Neuron Coverage from DeepXplore [10]; as well as K-multisection Neuron Coverage, Neuron Bound. Coverage, Strong Neuron Act. Coverage, Top-k Neuron Coverage from DeepGauge [11]; and gives their assessment of these coverage criterion, namely that neuron coverage is easily satisfied, and that the other 4 from DeepGauge skew towards exploring corner-case behavior, rather than main behaviors. They proposed K-multisection Neuron Coverage as a criterion to quantify exploring main behaviors.

1.3.9 DeepGauge's Multi-Granular Coverage

The work by DeepGauge introduced the innovation of using training data to find a single neuron's maximum and minimum output, using this as a rough boundary for main function behavior and corner-case behavior. They also introduced considering the permutations of activations of neurons to give a more difficult to satisfy coverage criterion.

1.3.10 Modified Condition/Decision Coverage Based Coverage Criterion

MC/DC is a coverage criterion that requires the following [29]:

1. Each entry and exit point is invoked
2. Each decision takes every possible outcome
3. Each condition in a decision takes every possible outcome
4. Each condition in a decision is shown to independently affect the outcome of the decision.

DeepCover [26] explored criterions based on MC/DC concepts. Sign-sign neuron coverage – looking at permutations of the signs (negative and positive) of pairs of neurons – is presented in the paper by DeepCover and is reused in TensorFuzz [24]. It is similar to the next coverage metric, combinatorial coverage.

1.3.11 Combinatorial Testing Coverage

The papers by DeepCover [26] have mentioned the use of concepts from combinatorial testing, namely in sign-sign neuron coverage, and these ideas are fully explored in the work on DeepCT by Ma et al. [28]. They reduce each neuron to being activated/de-activated. Then, they can consider the permutations of activating/de-activating groups of neurons of size *k*. By considering how extensively these permutations are explored, they evaluate the coverage.

The main benefit they claim is that erroneous behaviors occur when pairs of factors interact. Thus, by considering permutations of pairs of neurons, we obtain a more comprehensive look logic coverage. While their coverage criteria are harder to fulfill than neuron coverage, they do not consider the statistical properties of the neurons.

1.3.12 Lipschitz Coverage

In the DeepConcolic [27] they use Lipschitz Continuity: a form of continuity that states that the maximum slope of a continuous function is limited by a real number c. In application they use this concept to define Lipschitz Coverage: an input is Lipschitz Covered iff in an open ball of size b (a parameter), there exists no inputs which cause outputs that differ by more than c (another parameter). i.e. if the open ball centered around an input fluctuates in output greatly, then it is not Lipschitz Covered.

1.3.13 Fuzzing

Fuzzing is a technique from traditional software testing [30], which uses seeds to produce random inputs which can be used to expose erroneous behavior. These seeds can be saved so the error inducing input may be reproduced and examined. There are a few differentiators. The first is when the Fuzzer mutates existing inputs or creates inputs from scratch; second is whether they are aware of input structure or not; third is they can be white, grey, or black box [30]. Papers like TensorFuzz [24], DeepHunter [25], and DLFuzz [31] have reapplied these methods with DNNs, and incorporated the coverage criterions discussed in DeepXplore [10] and DeepGauge [11]. By mutating inputs and using coverage criterion as a heuristic, we can obtain higher coverage. Due to the small size of the perturbations, we use metamorphic relations to assume that the output should not change significantly. Thus, we can obtain adversarial inputs. One novel benefit of this method is due to its random nature, there is no inherent order in which we must process the data. This means that using a probabilistically weighted batch implementation, it is possible to parallelize the testing and increase throughput, which may be useful in operation [25].

1.3.14 Concolic Testing [27]

Concolic testing is a software testing technique that uses concrete testing (running with a concrete input) and symbolic testing (analyzing the program to find what inputs are needed to get a code block to execute).

The paper "Concolic Testing for Deep Neural Networks" discusses implementing concolic testing techniques for DNN. However, while sounding quite novel, in practice symbolic analysis is basically the gradient ascent and criterion-based heuristics used by other papers. They even reuse sign-sign coverage from MC/DC [26] and Neuron coverage from DeepXplore [10]. Their most novel contribution is a new criterion called "Lipschitz Coverage".

1.3.15 DeepCruiser or Markov Decision Process Testing [32]

An adaption of some of the ideas involving input space coverage proposed by DeepXplore [10] and DeepGauge [11] to RNNs (recurrent neural networks). Some of the main difficulties of adapting neuron coverage and similar coverage criterions is they do not cover the attention-vector used by RNNs. This paper uses the paths of the attention vectors for training data (called "traces" in the paper), and by collectively analyzing the traces, we obtain an MDP. Using this, we can test how extensively our test data explores our abstracted MDP.

While their techniques are not relevant to our work, we would like to highlight that they use analysis of how the model handles training data to quantify whether test data is within the main function body or is outside standard functionality. They also use metamorphic relations as a basis for assuming the outputs of the generated adversarial data.

1.3.16 Training Set Reduction [33]

By reducing the size of the dataset, we may train models more quickly. Considers the optimal method to cut the original dataset, including random selection, distance based selection, and loss-function based selection. It concludes that loss-function based selection is most optimal. The paper is quite orthogonal to our area of research.

1.3.17 Data Cleaning

One widely cited use of these methods is their ability to clean training data sets of mislabeled/malicious data [22]. This is a worthy application that we may be able to explore further down the line, if time allows.

1.3.18 Feature Based Adversarial Testing [34]

Using the feature extraction tool SIFT, they extract the key features of an image. They find that when single pixel perturbations are applied near the features, erroneous output is more common. This is another alternative heuristic to gradient ascent that can be to guide adversarial input finding.

1.3.19 DeepMutation or Mutation Testing [35]

An adaption of the technique of mutation testing. The program to be tested is mutated in several ways, and faults are introduced to the program. Then, the mutants and original program are subjected to the test suite, and failures to distinguish between mutant and original are noted.

2 Methodology

2.1 Design

The literature survey played a big role in many of the choices I made in the design of my thesis. I wanted to do an expansion of neuron coverage [10], to develop a new coverage criterion to quantify how well the test data engages the logic of a DNN.

As I read the papers DeepGauge [11] and DeepCruiser [32], the concept of using training idea to map out the input space as "Main Behaviors" and Corner-case Behaviors" was very interesting. From this I got the idea of storing the outputs of neurons for training inputs and storing them for analysis.

From DeepGauge and DeepCT [28], I considered using a more combinatorial-testing based approach, pushing me towards creating a metric that would analyze the interactions between seemingly independent neurons. For DNNs, we know that neurons on the same layer can be thought of as being responsible for identifying a separate feature in the input, and that these neurons are thought to be independent of each other [10]. However, while they might be independents, the training data can still have statistical relationships for when certain features occur. It would be very interesting to show the statistical relationships between the features that neurons identify, and whether giving statistically unlikely (i.e. corner-case) features would cause more erroneous behaviors.

One thing I didn't like about Neuron Coverage is the arbitrary threshold for activation. Neuron Coverage does not consider the statistical properties of the neuron output. For example, a neuron might have a high mean output, meaning that it is easily activated, whereas another may have low mean output, meaning it is harder to activate. This implies that neuron activation is exponentially hard to fulfill. While DeepGauge attempts to fix these issues by considering the maximum output in training, it doesn't allow us to set different levels of "unexpectedness" of the intensity of a neuron output (i.e. a certain feature) is, only being able to say that it has never seen this intensity in training before. From this, I wanted a criterion whose threshold for activation would fix these issues.

In this design section, I present a methodology that aims to fix the issues I have mentioned.

2.1.1 Data Collection

For each input for the training data, we can collect the neuron output of the penultimate neuron layer, before the activation function.

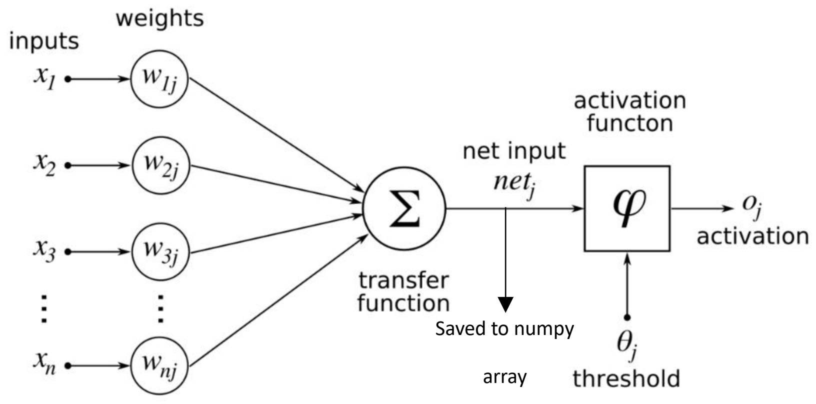


Figure 2: A visualization of data flow

This is so that the distribution of the neuron output is not altered by the activation function

2.1.2 Statistical Analysis

With this collected data, we analyze the statistical properties of the neuron output. We can calculate the mean, variance, as well as the covariance between neurons.

2.1.3 Criterion Design

Using the mean, variance, and covariance calculated from the previous stage, we can approximate the distribution of the neuron output. We define a new form of coverage, k-size neuron group orthant coverage, or orthant coverage as I will refer to it in the remainder of the paper. (note: orthant is the analogue to a quadrant in n-dimensional space)

Essentially, by approximating the neurons/groups of neurons as being normally distributed, we can analyze the statistical distributions of neuron groups of size k and identify neuron outputs that are statistically unlikely. I will go over some examples for different sizes of k below.

In the case where k=1, we are simply looking at the statistical distribution of a single neuron. For each neuron, using their approximated normal distribution, we define their upper threshold, which is mean plus n standard deviations, and lower threshold, which is mean minus n standard deviations. We will consider a threshold to be "covered" as long as it has been exceeded at least once. This is so that we can change n to quantify more extreme behaviors. This offers a benefit over DeepGauge and DeepXplore, namely the threshold matches the distribution of each neuron. We then sum the number of covered thresholds, then divide by the number of neurons in the layer times 2, similar to neuron coverage.

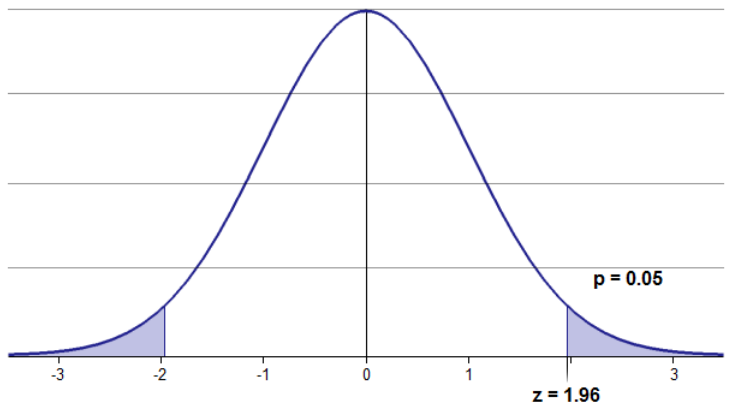


Figure 3: We want outputs on the edges of the distribution

In the case where k=2, we take the distribution of a pair of neurons, we assume they are a binormal distribution. At the mean, we split the distribution horizontally and vertically. We will consider an orthant to be covered if there is an input in the test set that causes the output of the pair of neurons to be within that orthant and be more than n standard deviations away from mean. We sum the number of orthants covered and divide by total number of quadrants to obtain the coverage percentage, similar to neuron coverage

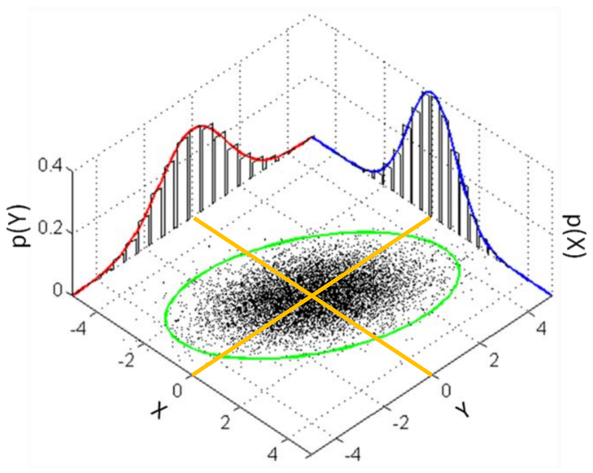


Figure 4: We are looking for points in each quadrant, outside of the green line

And so on and so forth for higher dimensions of k. For k = 3, each neuron group would have 8 orthants that need covering.

2.1.4 Base Set Selection and Adversarial Input Generation

We select a 'base set' of images from the test set of each dataset. For each model, we use the same adversarial attacks as described in DeepGauge to generate a set of adversarial images unique to each model. These images will then be used to evaluate how well each coverage criterion preforms during evaluation.

2.1.5 Coverage Evaluation

Similar to DeepGauge, we will be seeing how the inputs from each type of adversarial attack effect the neuron/orthant coverage. However, instead of just calculating coverage after evaluating against the entire set of adversarial images, we will be looking at how coverage changes as new adversarial images are gradually added. To do this, we first obtain the coverage amount for the base set of test images. Then, for each set of adversarial images, we randomly sort these adversarial attack images (with a fixed seed). For each image in this randomized list, we update the neuron/orthant coverage, and plot this data on a graph. We do this because certain adversarial images create disproportionally large spikes for certain models, and in order to alleviate this, we repeat this process for different fixed seeds, and display an averaged graph with error bands. Note that the final coverage obtained will always be the same, as we are still using the same set of adversarial inputs.

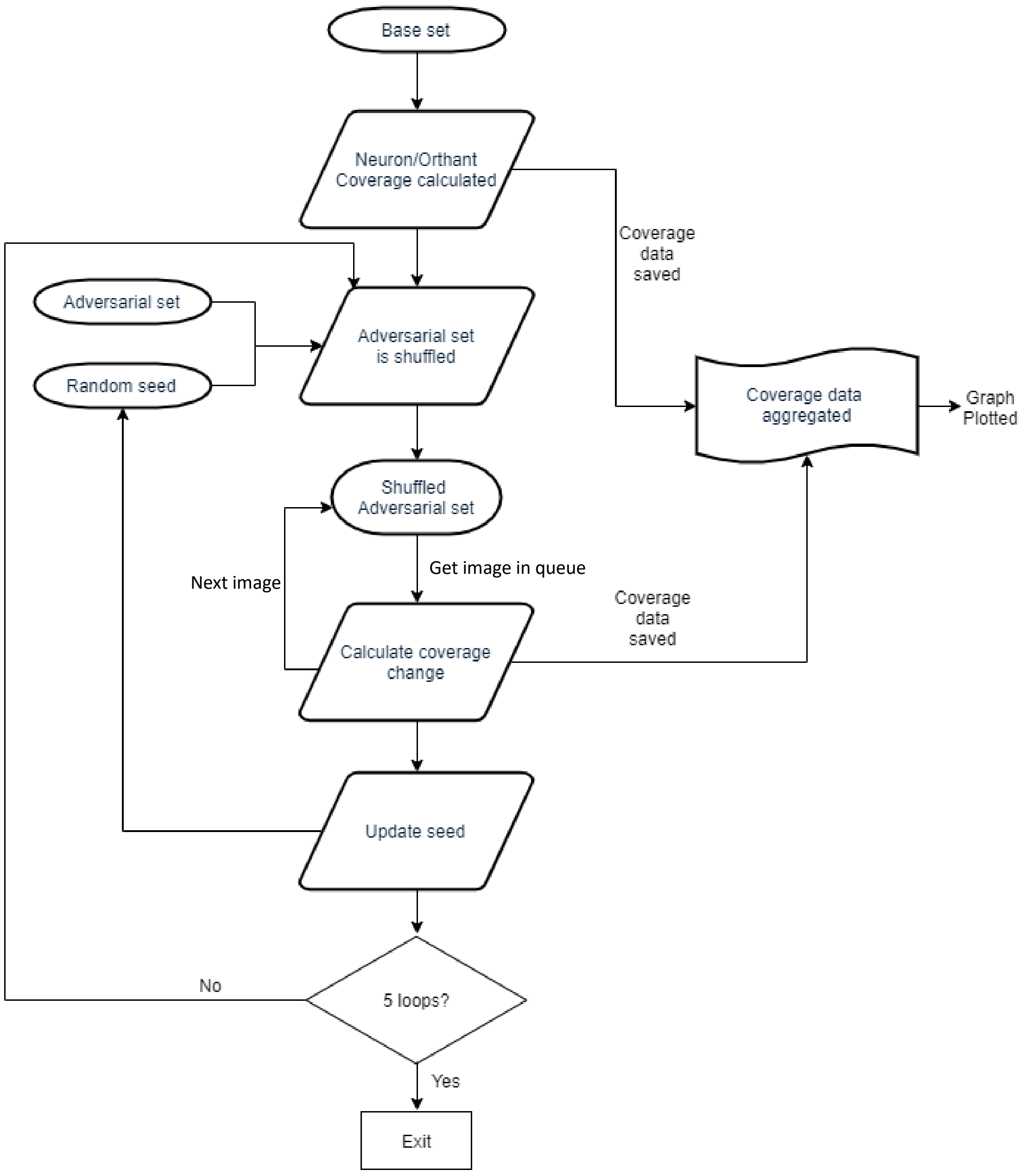


Figure 5: Flow chart of coverage evaluation process

2.1.6 Further Work: Guided Adversarial Attack

The paper also presents an implementation of the DeepXplore [10] adversarial generation algorithm using k-neuron group orthant coverage as a heuristic. However, due to time constraints work on this is only at an initial stage and may be found on the repository for this project.

2.2 Implementation

2.2.1 Train/Download Models for Testing

The models for the MNIST dataset are implemented in the LeNet-1, LeNet-4, LeNet-5 architectures [36]. They were trained with the Keras library, and achieved accuracy of 1.38%, 0.98%, and 0.92% respectively. The models for the CIFAR10 and CIFAR100 datasets are implemented in the VGG-16 architecture [37], with the implementation coming from [38]. The CIFAR-10 model has been verified to have a validation accuracy of 93.56% and the CIFAR-100 has been verified to have a validation accuracy of 70.48%.

2.2.2 Data Collection

Using the Keras [39] and Numpy [40] library to implement. This stage ended up being more complicated than I initially anticipated, because the API did not support getting the output of a layer before the activation function. I spent some time devising a workaround to get this to work (of which can be accessed on the github repo). The workaround involves saving the original weights, replacing the layer with a new layer with no activation function, and moving the weights to this new layer. The collected data is stored in a numpy datafile.

2.2.3 Statistical Analysis

Using the data from the numpy array, the covariance between the dataset of each neuron is calculated, and stored in a covariance matrix, describing the distribution of all neurons in the layer. The mean of each neuron output is calculated and stored in a mean vector. The PyPlot and Seaborn libraries were also used to plot some of the distributions.

The main difficulty that arose were memory errors, due to the size of some of these vectors, combined with the memory intensity of the numpy covariance function. To fix this, I implemented a covariance function that would use less memory. The method still uses the numpy.cov() function, but makes use of checkpoints so that the memory doesn't cap out. The implementation may be found in the repository for this project.

2.2.4 Criterion Implementation

In one dimension when we calculate the number of standard deviations between a datapoint and the mean of a distribution, we are first taught of the following formula:

$$z = \frac{x - \mu}{\sigma}$$

Where the resultant z is the number of standard distributions the datapoint x is away from the mean μ. We can visualize this as shifting and stretching the normal distribution such that the distance from the mean translates cleanly to the number of standard distributions away from the mean.

For higher dimensions, this concept is captured in the Mahalanobis distance, represented in the following formula:

$$D_M(\vec{x}) = \sqrt{(\vec{x} - \vec{\mu})^T S^{-1} (\vec{x} - \vec{\mu})}.$$

Where x is the input vector, μ is the mean vector, S is the covariance matrix.
Using the covariance matrix and mean vector, we may calculate the Mahalanobis distance for any group of neurons.

Using this concept, I tweaked the structures and functions from DeepXplore. Instead of checking the output of a neuron against a fixed threshold, we first check if the orthant has been covered before, if it has not, we calculate the Mahalanobis distance of the input, identify which orthant the input lies by comparing it against appropriate neuron mean values, and update the coverage tracker accordingly.

2.2.5 Base Set Selection and Adversarial Input Generation

Using the datasets provided through the Keras API, we select a subset of test images, which we will call the base set. We only select a subset, because there is a big difference in computational time. To produce the adversarial images, the CleverHans adversarial attack library [41] is used. The library accepts the base set and a model, and outputs adversarial inputs. The following attacks were used on the MNIST dataset models, matching the choices in DeepGauge: Basic Iterative Method (BIM), Fast Gradient Sign Method (FGSM), Jacobian-based Saliency Map Attack (JSMA), and Carlini/Wagner Attack (CW). BIM attack was used on the CIFAR10 and CIFAR100 models.

Carlini Wagner was found to be the slowest by a large margin, taking almost 20 minutes to generate 100 adversarial inputs, as opposed to 0.5-2 minutes for the other attacks.

2.2.6 Coverage Evaluation

Drawing upon the skills I developed from my studies, I implemented the core evaluation loop. All the previous data processes/functions were used here. The pandas DataFrame was used here to plot the graph on seaborn, to better visualize the range of values of the coverage.

Surprisingly, everything went very smoothly, with no real implementation issues. There was a deprecation warning with the Keras Backend, however the issue was not pressing enough to fix. The main issue was that for larger group sizes, evaluation takes an incredibly long time, taking 1+ hours on my machine.

2.3 Testing

Due to time constraints, testing was quite minimal.

2.3.1 Model Accuracy Validated

Made sure that model accuracy was close to the accuracy described in literature.

2.3.2 Adversarial Inputs Validated

Check to see if adversarial inputs result in reduced accuracy.

2.3.3 Sanity Tests for Coverage

In the process of implementation, check if the code behaves correctly. E.g. increasing the size of the input data set results in higher coverage, increasing thresholds decreases coverage and vice versa, essentially, making sure the outputs make sense.

2.4 Evaluation

Whether the literature survey has been successful at fulfilling the first goal, generalizing the techniques used by researchers, is not something I can objectively answer here. I personally believe the literature survey section of this paper has presented a broad and encompassing exploration into DNN testing. However, the ultimate decision is left up to the reader.

To assess progress towards the second goal, "to improve/build upon existing methodology with new tools/techniques", we compare the behavior of neuron coverage with that of orthant coverage.

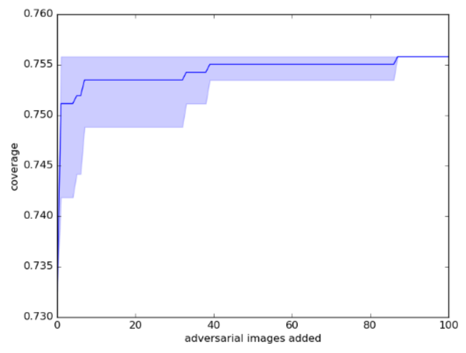
The main frustrations I expressed in with neuron coverage in the design stage is how it does not make use of the main-body behavior/edge-case behavior, it does not consider the interactions between neurons/features in the inputs, and that the threshold for activation is arbitrary. I believe that the design and implementation have fulfilled all these qualitative requirements.

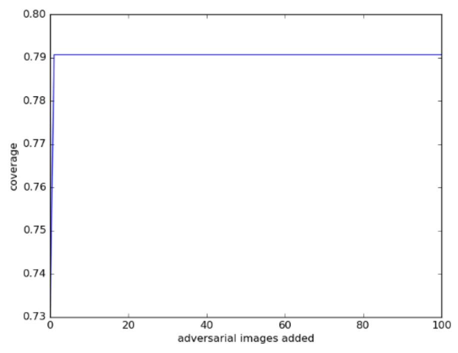
In order to more quantitatively compare neuron coverage and orthant coverage, I will be looking through the following quantifiers:

- Is it harder to satisfy than neuron coverage?
- Does orthant coverage perform better computationally?

When comparing the graphs of orthant coverage against neuron coverage, we note the following:

- For all types, there is an initial spike. This is probably because adversarial output makes use of different logic loops.
- After this initial spike, neuron coverage in small bursts.
- For orthant coverage, however, increases gradually, with more of the images contributing to increased coverage. This smoothness in coverage increase is more pronounced for larger k.
- Orthant coverage increases more overall. After the initial spike, orthant coverage continues to increase by at least 0.02. This difference is more pronounced for larger k. For k=3, coverage continued to increase by 0.2 after the initial spike.
- By comparison, neuron coverage increases less than 0.01 after the initial spike. In some cases, 0 coverage change was observed.
- When looking at the differences between attacks, we note that BIM caused the largest increases, followed by FGSM, then JSMA, then CW.

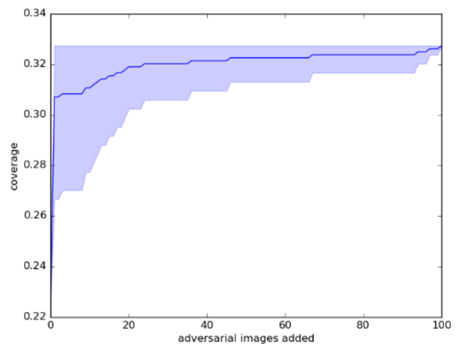


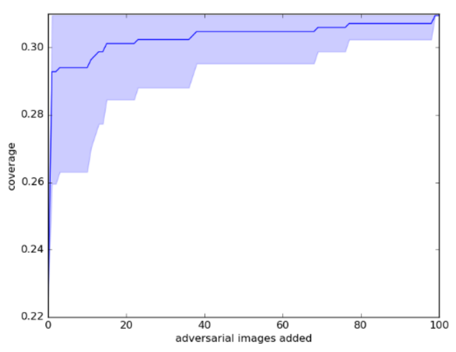


Figs 6-7: Neuron coverage graph, MNIST LeNet5, threshold = 0.6

Left: Fast Gradient Sign Method adversarial input

Right: Jacobian Saliency Map Attack adversarial input

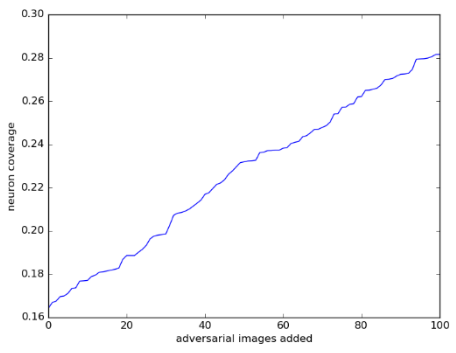


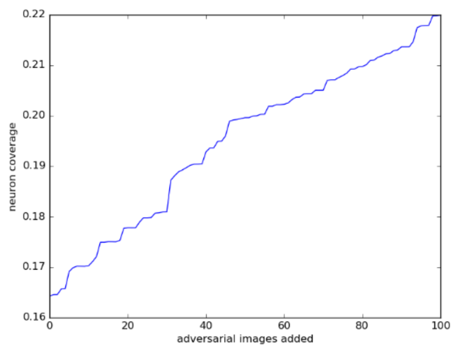


Figs 8-9: Orthant coverage graph, MNIST LeNet5, threshold = 2.5, group size 1

Left: Fast Gradient Sign Method adversarial input

Right: Jacobian Saliency Map Attack adversarial input





Figs 10-11: Orthant coverage graph, MNIST LeNet5, threshold = 3, group size 3

Left: Fast Gradient Sign Method adversarial input

Right: Jacobian Saliency Map Attack adversarial input

RQ 1: Is it harder to satisfy than neuron coverage?

Many researchers have noted that neuron coverage is quite trivial to maximize [24] [25], with only a few images being needed to maximize coverage. Following their logic, an improvement over neuron coverage would be a coverage criterion that is more difficult to satisfy with few images. Neuron coverage often increases in large bursts, whereas orthant coverage increases more gradually. Thus, we may expect more images to be needed to obtain full coverage, especially for large k. Thus, we can posit that orthant coverage requires more images to obtain full coverage.

RQ 2: Does orthant coverage perform better computationally?

As results were being generated, it was noted that for k=1, the program would finish running after ~20 seconds, significantly faster than neuron coverage, which took around ~5 minutes. However, for k=2, the program took ~10 minutes, and was significantly worse for k=3, taking 2+ hours. This is easily explainable once we consider the number of groups we are evaluating, as well as the orthants per group. For k=1, the number of groups is equal to the last layer, and the number of orthants is 2. For k=3, the number of groups is equal to number of neurons in last layer choose 3 (a *huge* number), and the number of orthants is 8. Time complexity increases at *least* exponentially with k.

In summary, computational efficiency of orthant coverage is similar to neuron coverage, up to k=2.

3 Discussion

While the results obtained are generally positive, there were numerous challenges faced during the implementations, flaws in the design that were amended, as well as further questions that were not fully explored in this paper.

3.1 Implementation challenges

There were numerous challenges and constraints that forced the scope of this thesis to change.

3.1.1 Fast Pace of DNN field

DNNs is quite a new field, and DNN testing is even more new. Most of the papers regarding DNN testing were written within the last 2 years. Because API changes in libraries are common, and it is not unusual for relatively recent implementations (2 years old) to be completely incompatible with the modern versions. This was the case when working with the code for the Udacity self-driving car dataset, which simply would not work on the most recent version of Keras.

While many libraries were touted as being universally applicable, often there would be incompatibility between DNN model and library. During the adversarial generation stage, I learned that the CleverHans library is not made to be strongly compatible with all Keras models and is not made to work on weaker machines like mine. For this reason most attacks failed for the CIFAR10 VGG16 models. Upon a deeper look into the documentation, and issues raised on the GitHub page, it seems that the developers did not prioritize universal compatibility. Due to time constraints, it wasn't possible for me to figure out how to reimplement the model to work with CleverHans.

3.1.2 Poor Documentation

As a student used to exclusively working with well-maintained, open-source software, working with some of the software was a very different experience than I expected. The documentation of model implementations was often unclear. For instance, while I was working out how to run and test the Udacity models, the lack of documentation meant I spent a lot more time than I should have. In papers like DeepXplore, I opted to not use their automated vehicle steering models, because details explaining training dataset used and preprocessing procedures left out.

3.1.3 Lack of Open Access

Another thing I did not expect was that pre-trained models for well-known architectures (e.g. LeNet-1, CNN) cannot be found online. Meaning time was needed to implement them myself, or low quality implementations had to be used.

In earlier stages of planning, I was hoping to use more algorithms, and compare orthant coverage with more criterions. Namely, I was hoping to use the code from DeepHunter and DeepGauge. However, implementations for these papers were not publicly available, and the writers of these papers did not respond when asked. This meant I either had to implement them myself, or change the scope. Due to time, I opted for the latter.

3.2 Design Flaws

Due to my own lack of experience, quite a few of the decisions I made when outlining the scope for this project did not work out.

3.2.1 Dataset Size

I had originally planned to evaluate using models trained against the ImageNet [42], and Udacity self-driving [43] datasets. However, the size of the ImageNet and Udacity datasets would have made testing very difficult. There was also difficulty in downloading a comprehensive dataset in the case of ImageNet.

3.2.2 Not appropriate for Autonomous vehicles

As I researched the implementation of autonomous steering models, I found that many modern implementations of steering mechanisms use some version of internal state to recall past events – mainly LSTM and attention.

Neuron/orthant coverage only tracks neuron activity, and do not evaluate the internal state of the model. Using orthant coverage to explore these models would be inherently flawed. This means that DeepTest [20], with its use of neuron coverage to guide adversarial generation, is flawed.

For such recurrent models, tools that evaluate exploration of internal state may be more useful, like DeepCruiser [32].

3.2.3 Scaling issues

The methodology proposed by this paper requires analysis over entire training data set, which may not be scalable for commercial DNNs, which are trained on huge, ever increasing datasets.

For complex commercial DNNs, even the penultimate layer may have thousands of neurons, meaning a huge number of unique neuron groups that need to be explored.

These issues were very apparent during the testing and evaluation stages, where evaluating for k=4 took far too long to be practical.

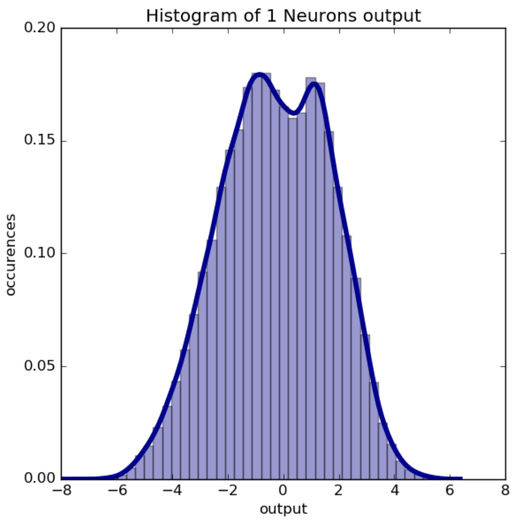
While these issues are inherent to the design of the criterion, they are not unfixable. Potential fixes to the scalability issue are discussed in the following section.

3.3 Further Questions and Research

While working on the thesis, some arbitrary decisions were made, and some potential research paths were not fully explored in the interest of keeping the scope within a workable size. I will address most of them here.

3.3.1 Statistical distributions

When designing the coverage criterion, I guessed that the neurons would be approximately normally distributed. This was somewhat confirmed, with most neurons being roughly normally distributed, albeit with a dip around 0.



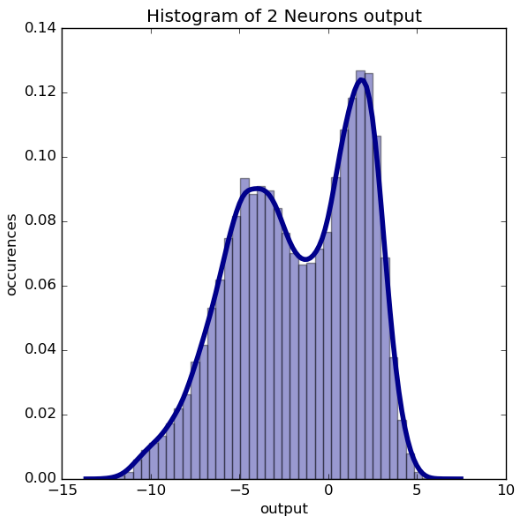
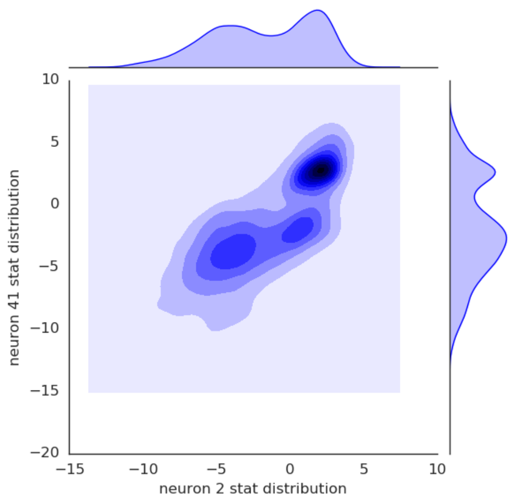


Figure 12: Plot of neuron 1 for MNIST LeNet5 model

Figure 13: Plot of neuron 2 for MNIST LeNet5 model

However, as seen in the figures below, some extreme cases differ from the normal distribution significantly.



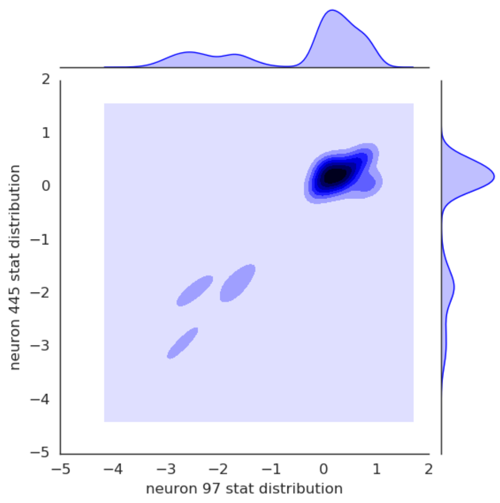
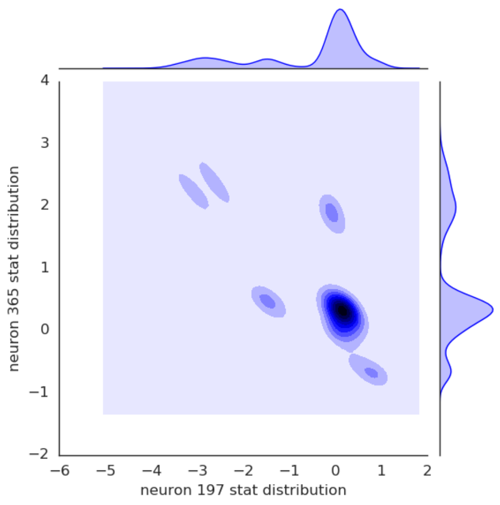


Figure 14: Plot of highly covariant neurons for MNIST LeNet5 model

Figure 15: Plot of highly covariant neurons of the CIFAR10 VGG16 model



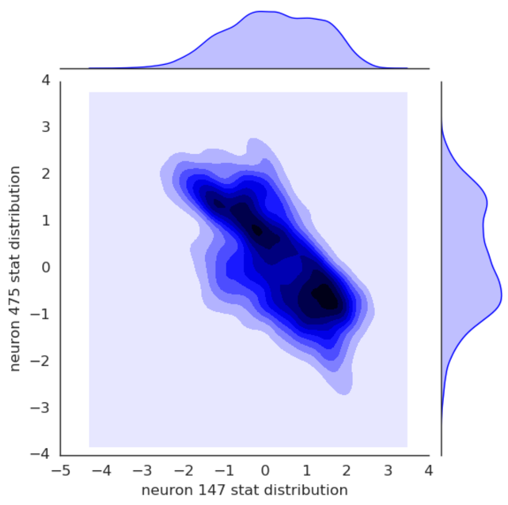


Figure 16: Plot of negatively covariant neurons for CIFAR10 VGG16 model

Figure 17: Plot of negatively covariant neurons of the CIFAR100 VGG16 model

While these results are surprising, they do not invalidate the results of this thesis, as outputs far away from the center of the neuron are still unlikely, and the majority of neurons are well approximated by normal distribution.

It can be noted that these strange distributions are more significant for VGG16 models and is especially prominent for the CIFAR10 VGG16 model. One hypothesis I considered is that since VGG16 is an overly powerful architecture for the CIFAR10 dataset, perhaps overfitting is causing this strange effect. It would be a worthy topic to follow-up on in future research.

It is also worth considering if as concept like orthant coverage could be made to work with for mixture distributions

3.3.2 Penultimate layer

During the data collection, I decided to only analyze penultimate layer of each model. A single layer was chosen mostly in the interest of keeping the workload lower. As for why the penultimate layer in particular, the final layers would be looking for more macro-level features, and it would be closer to the essence of combinatorial testing to look for more general/macro-level features. The final layer outputs the final result, so the output here would probably be heavily mutually exclusive, and not worth analyzing.

3.3.3 Partial datasets and orthant groups

While this paper proposes to use the entire training dataset to evaluate the statistical distribution of the model, it may be unnecessary. For commercial use cases, it may be impractical/impossible to evaluate the dataset against the entire training set. Future research could be done to see if only a subset of the training data is necessary to obtain a rough approximation of the distribution of the neuron output.

A similar logic could be expressed towards exploring every single neuron group. Perhaps only a subset of all neuron groups is already sufficient to get respectable results.

3.3.4 Parallelization

While evaluating orthant coverage can be quite slow, it is a parallelizable process. It could be implemented with a shared coverage tracker being updated by multiple computers. An implementation with parallelization could help orthant coverage scale for commercial use and is worth looking into in further research.

3.3.5 Edge-case vs main-body behaviors

Adversarial attack methods create adversarial output by exploiting the decision boundaries of DNN models. A hypothesis I considered is that these adversarial inputs exist close to these decision boundaries because the training data did not have enough inputs to teach the model the correct boundary. Thus, the adversarial inputs produced can be thought of as edge-case, as they explored poorly trained areas of input space. Since orthant coverage responds more strongly to adversarial output than neuron coverage, perhaps it is better at identifying exploration of these decision boundaries/edge-case behaviors. The content of this thesis did not contain enough evidence to support this hypothesis and it would be worth exploring in further research.

3.3.6 Adversarial generation

Due to time constraints, the effectiveness of orthant coverage in guiding adversarial exploration was not explored.

3.3.7 A Bitter Lesson?

We close off discussion with a sobering blog post [44] by Rich Sutton, one of the founding fathers of reinforcement learning. He suggests that attempting to bake in human logic into our machine learning systems is not as efficient as relying on raw computational power in the long run. Indeed, trying to adapt testing techniques from traditional coding for testing DNN may not be useful in the long run.

This thesis has been inspired by and built on top of existing research which takes traditional software testing protocols, tools developed by humans, and try to apply their logic to evaluating DNNs. This falls squarely into the type of behavior Sutton prescribes us to avoid. Whether the testing of DNNs will fall into the trends described by Sutton, remains to be seen.

4 Conclusion

This FYT has presented a comprehensive survey into recent developments in the field of deep neuron network testing. This process helped us gain insight into the subject, and heavily informed the design of further research.

From this research, we designed and demonstrated a new coverage criterion, orthant coverage, for testing and evaluating DNNs. We have shown it to be comparable in quality to that of Neuron Coverage. In the process of implementing orthant covereage, we made use of available libraries, and in cases when the functions given were not adequate, we made small tweaks to get them to work. The results of this paper gave promising initial results supporting the usefulness of orthant coverage.

The limitations of this implementation were discussed, especially in the context of scaling the methodology for commercial use. Some surprising results were also found and were discussed. If we had had more time, we would have decided to focus on answering the following questions:

- Is parallelization a valid method for decreasing the computation time?
- Is a partial dataset enough to approximate the results of orthant coverage?
- Is orthant coverage really better at analyzing edge case behaviors?
- What causes the strangeness of distribution of the neuron outputs?

5 Project Planning

GANTT Chart

Task	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr
Study fundamentals of neural networks	■								
Study relevant literature	■	■	■						
Validate the testing methodology		■	■	■					
Do further research into DNN testing				■	■				
Design generalized testing framework						■	■	■	
Research into potential limitations						■	■	■	
Retrain and recreate the self-driving models	■	■	■						
Implement and verify testing methods			■	■	■				
Apply generalized framework to other applications						■	■	■	■
Implement improved experiment methods						■	■	■	■
Test training models	■	■	■						
Test original testing methods			■	■	■				
Test new testing methods						■	■	■	■
Test generalized framework									
Write the Proposal		■							
Write the Monthly Reports		■	■	■	■		■		
Write the Progress Report							■		
Write the Final Report									■
Prepare for the Presentation									■
Design the Project Poster									■

7 Required Hardware & Software

7.1 Hardware

Server PC: Linux system with i7-8700k CPU and titan V GPU

Laptop: Personal laptop

7.2 Software

Key libraries used:

Python 3.6.6

Tensorflow 1.12.0

Keras 2.2.4

8 References

[1] B. C. M., Pattern Recognition and Machine Learning, 2006.

[2] Google Cloud, "Detect Text (OCR) | Cloud Vision API Documentation | Google Cloud," 18 January 2019. [Online]. Available: https://cloud.google.com/vision/docs/ocr.

[3] Microsoft, "Speech to Text API | Microsoft Azure," 2019. [Online]. Available: https://azure.microsoft.com/en-us/services/cognitive-services/speech-to-text/.

[4] Waymo, "Waymo – Waymo," 2019. [Online]. Available: https://waymo.com/.

[5] C. JX, "The evolution of computing: AlphaGo," *Computing in Science & Engineering,* 2016 Jul 18.

[6] deepmind.com, "AlphaStar: Mastering the Real-Time Strategy Game StarCraft II," 24 Jan 2019. [Online]. Available: https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/.

[7] Amazon Web Services, Inc., "Amazon Rekognition – Video and Image - AWS," [Online]. Available: https://aws.amazon.com/rekognition/.

[8] Microsoft, "Azure Machine Learning Service | Microsoft Azure," [Online]. Available: https://azure.microsoft.com/en-us/services/machine-learning-service/.

[9] Google Cloud, "Cloud AI | Cloud AI | Google Cloud," [Online]. Available: https://cloud.google.com/products/ai/.

[10] P. K, C. Y, Y. J and J. S., "Deepxplore: Automated whitebox testing of deep learning systems," *In Proceedings of the 26th Symposium on Operating Systems Principles,* pp. 1-18., 2017 Oct 14.

[11] J.-X. F. Z. F. S. J. X. M. L. B. C. C. S. T. L. L. L. Y. Z. J. Ma L, "Deepgauge: Multi-granularity testing criteria for deep learning systems," *In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering,* pp. 120-131, 2018 Sep 3.

[12] N. V and H. GE, "Rectified linear units improve restricted boltzmann machines," *InProceedings of the 27th international conference on machine learning,* 2010.

[13] R. DE, H. GE and W. RJ, "Learning representations by back-propagating errors," *nature,* 1986 Oct.

[14] S. Yaghoubi and G. Fainekos, "Gray-box Adversarial Testing for Control Systems with Machine Learning Component," *arXiv.org,* 2018.

[15] A. Nguyen, J. Yosinski and J. Clune, "Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images," *arXiv.org,* no. arXiv:1412.1897, 2014 .

[16] J. Su, D. V. Vargas and S. Kouichi, " One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation,* 2019 Jan 4.

[17] A. Kurakin, I. Goodfellow and S. Bengio, "Adversarial examples in the physical world," *arXiv preprint arXiv:1607.02533,* 2016 Jul 8.

[18] N. Papernot, P. McDaniel, I. Goodfellow, Z. B. C. Somesh Jha and A. Swami, "Practical Black-Box Attacks against Machine Learning," *arXiv.org,* 2016.

[19] H. G, V. O and D. J., " Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531,* 2015 Mar 9.

[20] Y. Tian, K. Pei, S. Jana and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," *InProceedings of the 40th international conference on software engineering ,* 2018 May 27.

[21] I. J. Goodfellow, J. Shlens and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *arXiv.org,* 2014.

[22] D. A, A. M, R. R. Sikand S, D. N. Bose RP and P. S., "Identifying Implementation Bugs in Machine Learning based Image Classifiers using Metamorphic Testing," *InProceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis ,* 2018 Jul 12.

[23] Z. Y. Z. L. L. C. K. S. Zhang M, "DeepRoad: GAN-based Metamorphic Autonomous Driving System Testing," *arXiv preprint arXiv:1802.02295,* 2018 Feb 7.

[24] A. Odena and I. Goodfellow, "TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing," *arXiv preprint arXiv:1807.10875,* 2018 Jul 28.

[25] X. X, M. L, J.-X. F, C. H, X. M, L. B, L. Y, Z. J, Y. J and S. S, "DeepHunter: Coverage-Guided Fuzzing for Deep Neural Networks," *arXiv preprint arXiv:1809.01266,* 2018 Sep 4.

[26] X. H. D. K. Youcheng Sun, "Testing Deep Neural Networks," *arXiv preprint arXiv:1803.04792,* 2018 Mar 10.

[27] S. Y, W. M, R. W, H. X, K. M and K. D, " Concolic Testing for Deep Neural Networks," *arXiv preprint arXiv:1805.00089,* 2018 Apr 30.

[28] M. L, Z. F, L. B. Xue M, Z. J. Liu Y and W. Y., "Combinatorial Testing for Deep Learning Systems," *arXiv preprint arXiv:1806.07723,* 2018 Jun 20.

[29] K. Hayhurst, D. Veerhusen, J. Chilenski and L. Rierson, "A Practical Tutorial on Modified Condition/Decision Coverage," *NASA,* 2001 May.

[30] J. Neystadt, " Automated Penetration Testing with White-Box Fuzzing," Microsoft, February 2008. [Online]. [Accessed 2019].

[31] J. Guo, Y. Jiang, Y. Zhao, Q. Chen and J. Sun, "DLFuzz: Differential Fuzzing Testing of Deep Learning Systems," *arXiv preprint arXiv:1808.09413,* 2018 Aug 28.

[32] D. X, X. X, L. Y, M. L and L. Y. Zhao J, "DeepCruiser: Automated Guided Testing for Stateful Deep Learning Systems.," *arXiv preprint arXiv:1812.05339,* 2018 Dec 13.

[33] S. H and G. A, "Towards Testing of Deep Learning Systems with Training Set Reduction," *arXiv preprint arXiv:1901.04169,* 2019 Jan 14.

[34] W. M, H. X and K. M, "Feature-guided black-box safety testing of deep neural networks," *arXiv preprint arXiv:1710.07859,* 2017 Oct 21.

[35] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao and Y. Wang, "DeepMutation: Mutation Testing of Deep Learning Systems," *arXiv preprint arXiv:1805.05206,* 2018 May 14.

[36] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," 1988.

[37] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 2014.

[38] Y. Geifman. [Online]. Available: https://github.com/geifmany/cifar-vgg.

[39] F. Chollet, "Keras," [Online]. Available: https://github.com/keras-team/keras.

[40] T. Oliphant, "A guide to NumPy," 2006.

[41] N. Papernot, "https://github.com/tensorflow/cleverhans," [Online]. Available: https://github.com/tensorflow/cleverhans.

[42] J. Deng, "ImageNet: A Large-Scale Hierarchical Image Database".

[43] Udacity, "udacity/self-driving-car," GitHub, [Online]. Available: https://github.com/udacity/self-driving-car. [Accessed 2019].

[44] J. S, B. SS, C. J, K. ZT and I. RK, "Avfi: Fault injection for autonomous vehicles," *In2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)* , pp. 55-56, 2018 Jun 25.

[45] A. Davies, "What Is A Self-Driving Car? The Complete WIRED Guide," WIRED, 2018. [Online]. Available: https://www.wired.com/story/guide-self-driving-cars/. [Accessed 20 Sept 2018].

[46] W. Knight, "DARPA Is Funding Projects That Will Try To Open Up AI'S Black Boxes," MIT Technology Review, 2018. [Online]. Available: https://www.technologyreview.com/s/603795/the-us-military-wants-its-autonomous-machines-to-explain-themselves/. [Accessed 20 Sept 2018].

[47] S. a. J. W. Levin, "Self-Driving Uber Kills Arizona Woman In First Fatal Crash Involving Pedestrian," The Guardian, 2018. [Online]. Available: https://www.theguardian.com/technology/2018/mar/19/uber-self-driving-car-kills-woman-arizona-tempe. [Accessed 20 Sept 2018].

9 Appendix A: Meeting Minutes

9.1 Minutes of the 1st Project Meeting

Date: August 15, 2018

Minutes for 15th Aug

Clarified on the goals of the paper with prof. cheung.

Current goal is to reproduce the results in the paper, and gain some experience working with tensorflow and keras.

Set up tensorflow and keras on laptop.

Fixed issues with version

9.2 Minutes of the 2nd Project Meeting

Date: September 5, 2018

Fixed some confusion with labeling format of the labelled datasets needed for training

Fixed some issues with file structure of the final format dataset needed to do training.

Clarified some issues with formatting of data into .npy array

Discussed existing code, as well as certain changes that need to be made so that it runs on modern versions of Keras and Tensorflow.

Discussion of greater goals beyond that of replicating existing research

9.3 Minutes of the 3rd Project Meeting

Date: September 12, 2018

Discussion of difficulties with chauffeur implementation.

Discussion of proposal contents, and the hardware and software requirements

Arrangement of meeting on the 17th

Discuss the need for remote server access needed to train the models.

9.4 Minutes of the 4th Project Meeting

Date: November 30, 2018

Discussed technical difficulties getting the codebase to run correctly,

Victor agreed to help me out with the implementation on my virtual machine

9.5 Minutes of the 5th Project Meeting

Date: January 1, 2019

Discussed the feasibility of new framework that combines adversarial attack and realistic transforms.

Discuss the utility of this new framework.

Action Items:

- Figure out how my proposal improves the quality of testing and training for self-driving cars.
- Figure out a quantifiable metric in which I can measure how my proposal improves the quality, so that I can assess whether the work in the thesis is a success or not.

9.6 Minutes of the 6th Project Meeting

Date: January 14, 2019

Discussed a new metric involving the covariance of the neuron outputs.

Struggled to find a way to convert these ideas into a concrete testing metric

9.7 Minutes of the 7th Project Meeting

Date: February 15, 2019

Met with Professor Cheung to discuss future direction of the thesis, as well as the potential for publication.

9.8 Minutes of the 8th Project Meeting

Date: March 6, 2019

Met with Professor Cheung to begin to nail down the key quantifiers that could be used to evaluate the quality of the new criterion.