The Thesis Committee for Naman Maheshwari
certifies that this is the approved version of the following thesis:

Estimating the Minimum Bit-Width Precision for Stable Deep Neural Networks Utilizing Numerical Linear Algebra

APPROVED BY

SUPERVISING COMMITTEE:

Jaydeep P. Kulkarni, Supervisor

Sudhanva Gurumurthi

Estimating the Minimum Bit-Width Precision for Stable Deep Neural Networks Utilizing Numerical Linear Algebra

by

Naman Maheshwari

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2019

Dedicated to my Family.

Acknowledgments

I wish to thank the multitudes of people who helped me. I would like to thank my advisor Dr. Jaydeep P. Kulkarni and mentor Dr. Sudhanva Gurumurthi for giving me the opportunity to do a co-operative research internship at Advanced Micro Devices (AMD) Research, Austin, in Fall 2018, which formed as a starting point for this research work. Their guidance and support throughout the course of this project was invaluable. I would also like to thank Dr. Nicholas P. Malaya and Dr. Scott Moe for being great mentors and helping me with my work on a day to day basis. I appreciate all the inputs I got from them, which helped me progress with this work in a streamlined fashion. I would also like to thank all my team-mates at AMD Research and Circuit Research Lab, University of Texas at Austin for being amazing co-workers.

AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Estimating the Minimum Bit-Width Precision for Stable Deep Neural Networks Utilizing Numerical Linear Algebra

Naman Maheshwari, M.S.E.
The University of Texas at Austin, 2019

Supervisor: Jaydeep P. Kulkarni

Understanding the bit-width precision is critical in compact representation of a Deep Neural Network (DNN) model with minimal degradation in the inference accuracy. While DNNs are resilient to small errors and noise as pointed out by many prior sources, there is a need to develop a generic mathematical framework for evaluating a given DNN's sensitivity to input bit-width precision. In this work, we derive a bit-width precision estimator which incorporates the sensitivity of DNN inference accuracy to round-off errors, noise, or other perturbations in inputs. We use the tools of numerical linear algebra, particularly stability analysis, to establish the general bounds that can be imposed on the precision. Random perturbations and 'worst-case' perturbations, via adversarial attacks, are applied to determine the tightness of the proposed estimator. The experimental results on AlexNet and VGG-19

showed that minimum 11 bits of input bit-width precision is required for these networks to remain stable. The proposed bit-width precision estimator can enable compact yet highly accurate DNN implementations.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Deep neural networks (DNNs) achieve very high accuracy in many machine learning tasks such as object recognition, speech recognition, and natural language processing. Many kernels in deep learning, particularly in convolutional neural networks (CNNs) are designed for computer vision tasks and are dominated by computation. For example, AlexNet [Krizhevsky et al., 2012] has nearly 62.5 million parameters, 0.65 million neurons, 2.3 million weights (4.6 MB of storage) and requires 666 million MACs per 227x227 image (13 kMACs/pixel); whereas VGG-16 [Simonyan and Zisserman, 2015] possesses 14.7 million weights (29.4 MB of storage) and requires 15.3 billion MACs per 224x224 image (306kMACs/pixel) [Chen et al., 2016]. The trend in current architectures is towards networks with more layers, which require more storage and computation per pixel.

1.1 A Basic Neural Network

Figure 1.1 shows the example of a very simple fully-connected neural network. The first layer is called the input layer, which takes the desired form of input, such as images in case of CNNs designed for image recognition tasks.

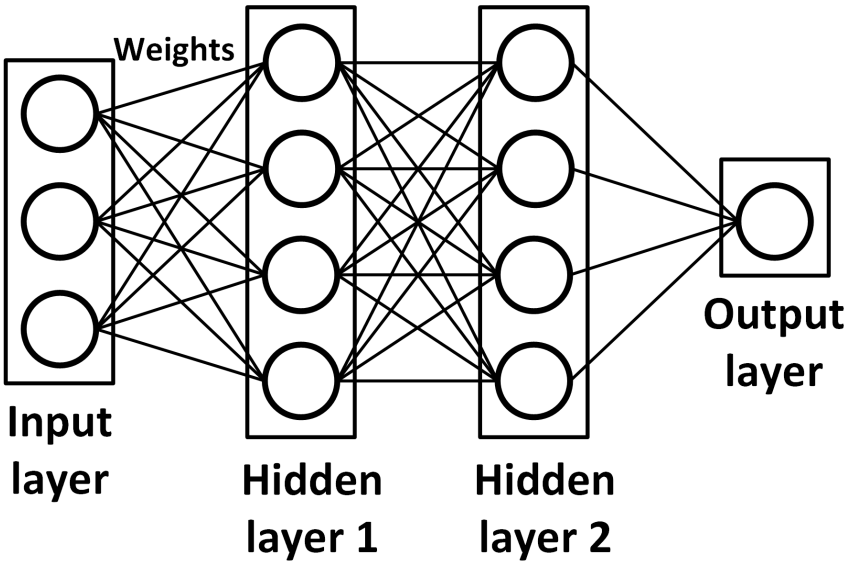


Figure 1.1: A simple fully-connected neural network.

The circles inside each layer are called neurons. For the input layer, these circles represent the pixel values of the image being applied to the network. The input layer is followed by several hidden layers, two in this case, which are responsible for feature extraction. The output of each neuron is called activation and the layers are connected by parameters called weights. Finally, the output layer can take various forms. For example, for an image recognition task, this layer consists of the set of classifiers and the classifier with the highest value is the predicted value of the neural network.

1.2 Convolutional Neural Network Basics

Convolutional Neural Network (CNN) is a class of deep neural network that is used for Computer Vision or analyzing visual imagery. A representative

CNN is shown in Figure 1.2. The components of a CNN are discussed in the following subsections.

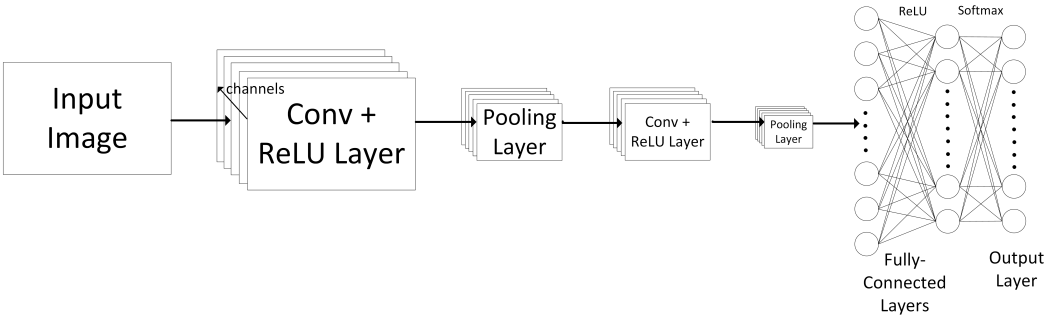


Figure 1.2: A representative convolutional neural network.

1.2.1 Convolutional Layer

An image is read as pixels and expressed as a matrix of dimension height by width by depth (NxNx3). Images makes use of three channels (RGB) resulting in a depth of 3. The convolutional layer makes use of a set of learnable filters. A filter is used to detect the presence of specific features or patterns present in the original image (input). It is usually expressed as a matrix (MxMx3), with a smaller dimension but the same depth as the input layer. This filter is convolved (slid) across the width and height of the input layer, and a dot product is computed to give an activation map. Different filters which detect different features are convolved on the input layer and a set of activation maps is outputted which is passed to the next layer in the CNN. Convolutional layers are used to learn different low-level features from the images.

1.2.2 Activation Function

The activation function is the non-linear transformation done over the output of the convolution operation and helps in deciding if the neuron would fire or not. This transformed output is then sent to the next layer of neurons as input. Most commonly used activation function is the Rectified Linear Unit (ReLU), which is discussed in Chapter 3. Some other common activation functions are discussed in Appendix 1.

1.2.3 Pooling Layer

The pooling layer can be seen between convolution layers in a CNN architecture and is used to reduce the amount of parameters and computation in the network, controlling overfitting by progressively reducing the spatial size of the network. Two most common pooling techniques are average pooling and maximum pooling, which take the average or maximum from the pool respectively. Unlike the convolution layer, the pooling layer does not alter the depth of the network.

1.2.4 Fully-Connected Layer

In the fully-connected layer, the neurons have complete connections to all the activations from the previous layers. Their activations can hence be computed with a matrix multiplication followed by a bias offset. Fully-connected layers are used to learn higher level features from the images and provide the last phase for a CNN network. The last fully-connected layer is

Softmax, which converts the outputs to probabilities based on an exponential function.

1.3 Motivation to Reduce Precision

Lowering bit-widths in both training and inference is advantageous for these systems because it permits faster computation and reduced power use, particularly when the underlying hardware can natively support reduced precision. However, higher performance must be balanced by the need for accuracy, particularly in safety-critical systems such as autonomous driving and health-care solutions where failure can be catastrophic. While many CNN architectures are resilient to lower precision, no rigorous and general result exists that provides *a-priori* estimates of the accuracy impact from reduced precision. Similarly, inference is shown to be more amenable to reduced precision than training through empirical analysis, but no theoretical results exist demonstrating why this is the case in our knowledge.

Noting that modern DNNs are governed by computational arithmetic, and so are amenable to the tools of numerical analysis, the approach presented in this thesis develops an estimator that predicts the impact of small perturbations to the inputs of a neural network on its output. This analysis must be performed on a network-by-network basis. This estimator can further be used to predict the minimal input precision required for that particular neural network such that the network does not become less stable for the perturbations to its input. We then numerically study this estimator on AlexNet and

VGG-19, using an adversarial perturbation technique to generate sample perturbations. We also characterize the effect of perturbations more commonly seen in the real world by studying the estimator with random perturbations.

This analysis used pre-trained models of two standard CNNs designed for object recognition tasks on the ILSVRC2012 ImageNet dataset [Russakovsky et al., 2015]: AlexNet [Krizhevsky et al., 2012] and VGG-19 [Simonyan and Zisserman, 2015]. AlexNet has five convolutional layers and three fully-connected layers, while VGG-19 is a deeper network with sixteen convolutional and three fully-connected layers.

1.4 Thesis Organization

This thesis is organized as follows. Related work in reduced precision analysis for neural networks is explored in Chapter 2. Chapter 3 presents the estimator of the condition number for a single neuron and a multi-layer neural network. Chapter 4 discusses the techniques used to test the estimator with the aid of adversarial perturbation generation methods. Chapter 5 discusses the techniques employed to compute the estimator, followed by the estimator results on two prominent CNN architectures, AlexNet and VGG-19 and Chapter 6 concludes the work.

Chapter 2

Related Work

Bit-Width of neural network parameters has been extensively studied as reducing precision can enable significant improvements in memory storage requirements and computational efficiency. Recent research efforts have shown that neural networks are amenable to reduced precision and have taken different directions to study the bit-width precision requirements for neural networks. Prior DNN precision research work can be broadly classified into four categories. The first category is the line of work which explores binarization and ternarization techniques for different neural network parameters during training and inference to compress the network with minimum or no loss in accuracy. The second category is an empirical exploration of bit-width precision for neural network parameters across different layers to achieve same accuracy as the full precision model. The third category is based on the derivation of theoretical bounds for neural networks with limited precision. The last category of prior work, which most closely relates to this thesis, uses of the tools of numerical linear algebra such as matrix norms to study the properties of DNNs and establishes analytical bounds on their characteristics. The following four sections summarize the related works in each of the four categories respectively.

2.1 Binary and Ternary Networks

Training CNNs as well as inference with binary or ternary weights and activations can achieve comparable accuracy to full precision networks ([Courbariaux et al., 2015], [Li et al., 2016], [Courbariaux et al., 2016], [Rastegari et al., 2016]). [Zhou et al., 2016] generalized this finding to enable different precision settings for weights and activations and also demonstrated how low precision can be used for gradients at training time. Recently, [Choi et al., 2019] proposed 2-bit Quantized Neural Networks (QNNs) using techniques to individually target weight and activation quantizations which can achieve higher accuracy compared to the previous quantization schemes. [Ott et al., 2017] explored reducing the numerical precision of weights and biases for different Recurrent Neural Networks (RNNs) empirically and concluded that weight binarization techniques do not work for RNNs while ternarization schemes yield similar or even higher accuracy than the baseline versions. All these methods help in greatly compressing the size of the neural networks, however, these lack an analytical method of arriving at the precision settings.

2.2 Empirical Exploration of Bit-Width Precision

[Judd et al., 2015] studied per layer quantization and observed that the tolerance of CNNs to reduced precision data varies not only across networks, but also within different layers of a network. They proposed an empirical method to find a low precision configuration for a network while maintaining high accuracy. Figure 2.1 shows the proposed empirical analysis for AlexNet,

where the authors claim that nine bits for representation of weights, ten bits for integer portion of the activations and nine bits for the fraction part of the activations are enough to achieve the similar accuracy as the baseline network with all single precision parameters. However, this method requires simulations and does not provably guarantee no loss in accuracy. [Judd et al., 2016] is an extension of the previous work, where the authors proposed a method called Proteus which analyzes a given DNN implementation and maintains the native precision of the compute engine by converting to and from a fixed-point reduced precision format used in memory. This enables using different representation per layer for neuron activations as well as weights and helps in reducing storage footprint and data movement, thus resulting in reduced power consumption. [Lacey et al., 2018] presented a learning scheme to enable heterogeneous allocation of precision across layers for a fixed precision budget. The scheme is based on stochastic exploration for the DNN to be able to learn an optimal precision configuration and leads to a favorable regularization effect, thus preventing overfitting and improving generalization. However, the optimal value of the precision budget is not known beforehand.

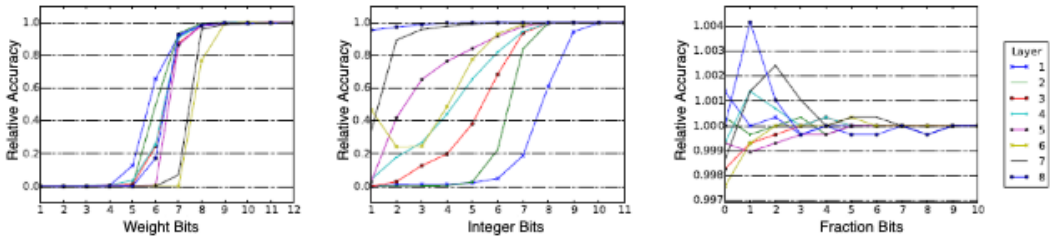


Figure 2.1: Empirical per layer quantization of weights, integer portion of activations and fraction portion of activations for AlexNet [Judd et al., 2015].

2.3 Theoretical Bounds

[Sakr et al., 2017] derived theoretical bounds on the misclassification rate in the presence of limited precision. Models pre-trained in floating point precision are taken and bounds are established to limit misclassification after quantizing activations and weights to a fixed-point format. However, this is limited in scope because it just bounds the accuracy between floating-point and fixed-point assignments. [Gupta et al., 2015] studied the impact of limited numerical precision on neural network training and the impact of rounding scheme in determining networks behavior during training. It shows that 16-bit fixed-point representation incurs little accuracy degradation by using stochastic rounding but does not study the precision requirements during inference. [Zhang et al., 2017] explored training at reduced precision but is mainly limited to linear models.

2.4 Use of Numerical Linear Algebra

In this thesis, we leverage the tools of numerical linear algebra, particularly stability analysis, to establish the general bounds that can be imposed on the precision. Such an approach is not without precedent, and investigations of the properties imposed on a network by the measure of the weight matrix can be at least traced back to [Bartlett, 1996], who showed that the generalization performance of a well-trained neural network with small training error depends on the size of the weights, not the number. Reasoning about the generalization ability of a neural network in terms of the size, or norm, of

its weight vector is called norm-based capacity control and [Neyshabur et al., 2015] evaluated this for feed-forward neural networks. Along with capacity, they also investigated the convexity and characterization of the neural networks.[Bartlett et al., 2017] evaluated the spectral complexity of the neural networks using the Lipschitz constant, i.e., the product of the spectral norms of their weight matrices. It is used to derive a margin-based multiclass generalization bound and showed that stochastic gradient descent selects predictors whose complexity scales with the difficulty of the learning task. [Liang et al., 2017] showed that the Fisher-Rao norm provides an estimate of the size of the network weights and associates this with the trained networks generalization capacity. In a very recent work which is closely related to this thesis, [Lin et al., 2019] show that because of the error amplification effect, the vanilla quantized models become more prone to adversarial attacks by means of an empirical study and propose a Defensive Quantization (DQ) method which controls the Lipschitz constant of the network during quantization. This thesis uses the measure of the weight matrix to study the stability of the neural networks to reduced precision and derive the precision requirements based on the estimator presented in Chapter 3.

Chapter 3

Stability Bounds on Forward Propagation

In numerical analysis, the condition number of a system is a measure of the change in the output value of a function or a network for a small change in the input argument. If the condition number of a system is $\kappa(A) = 10^k$, then up to k digits of accuracy may be lost on top of the loss of precision from arithmetic methods. This work derives an analogue of the numerical analysis condition number for neural networks as,

$$\kappa = \left(\frac{\|\delta_y\| \,/\, \|y\|}{\|\delta_x\| \,/\, \|x\|} \right). \tag{3.1}$$

We will define the variables used in Equation (3.1) in the Section 3.1. This quantity estimates the susceptibility of a network to perturbations. For most neural networks, this quantity can only be approximated, as shown in Section 3.1. We begin with a single neuron example to demonstrate the technique and generate intuition, and then generalize to an n-layer network.

3.1 Single Neuron Estimator Derivation

Consider first a single neuron, neglecting bias, as shown in Figure 3.1. Each input $(x_1, x_2, \ldots, x_n)$ is multiplied by an associated weight $(\theta_1, \theta_2, \ldots, \theta_n)$.

The results are then passed through an activation function, f to produce the output, y.

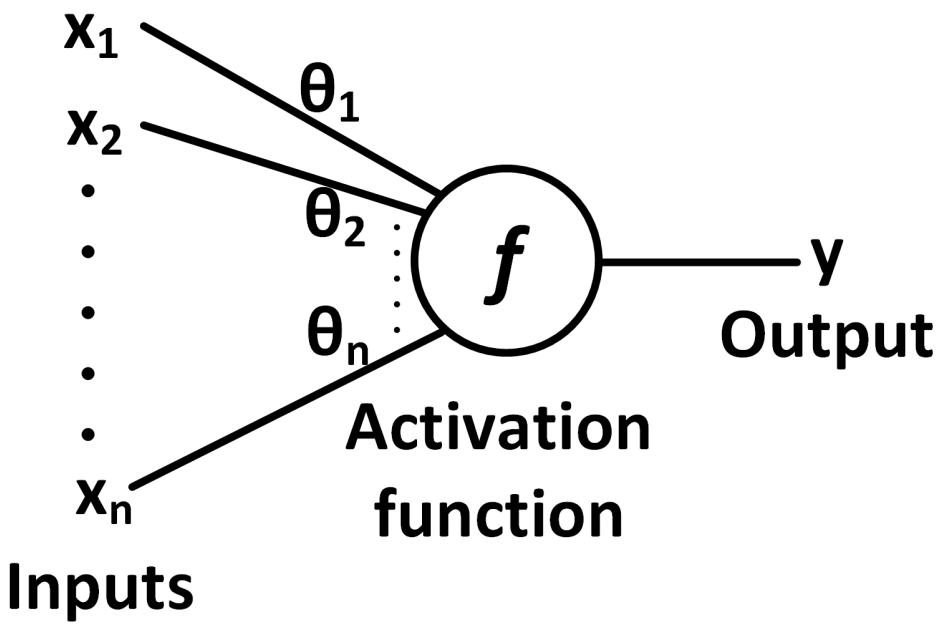


Figure 3.1: A single neuron example.

This is expressed as,

$$y = f(\sum_i \theta_i x_i) = f(\theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n).$$

Consider a common and representative activation function, the rectified linear unit (ReLU),[1]

[1]Other common activation functions are also considered and results are derived in Appendix 1.

$$f(z) = \begin{cases} z, & \text{for } z > 0 \\ 0, & \text{for } z \leq 0 \end{cases}$$

ReLU, due to its simplicity, has the added appeal of making subsequent computations more straightforward. Figure 3.2 shows the graph of ReLU. Notice that ReLU is simply $\max(0, z)$.

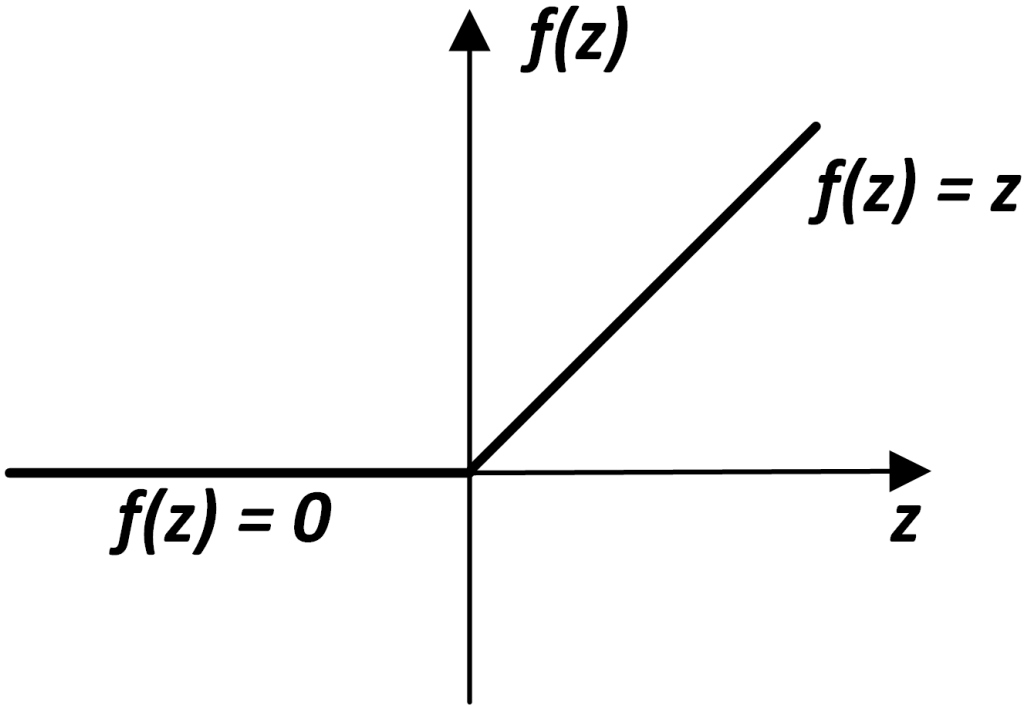


Figure 3.2: Rectified Linear Unit (ReLU) activation function.

In the rest of this thesis, we will use the shorthand $\theta\, x$ to indicate multiplication of a matrix of weights θ by a matrix x. Additionally, if f is a scalar defined function and x is a matrix, we will use the shorthand $f(x)$ to indicate f applied to each entry of x. Then, the output of a single layer neural

net satisfies,

$$\|y\| = \|f(\theta x)\| = \|\max(0, \theta x)\|. \tag{3.2}$$

Where $\|\|$ denotes the generalized norm. To analyze the stability of the single layer neural network, we introduce a small perturbation, δ_x, to the inputs. The resulting perturbation to the outputs is defined as δ_y. Then,

$$\delta_y := f(\theta(x + \delta_x)) - f(\theta x),$$
$$\|\delta_y\| = \|f(\theta(x + \delta_x)) - f(\theta x)\| = \|f(\theta x + \theta(\delta_x)) - f(\theta x)\|. \tag{3.3}$$

It can be shown that, by the triangle inequality,

$$\|f(\theta x + \theta(\delta_x)) - f(\theta x)\| \leq \|f(\theta x + \theta(\delta_x) - \theta x)\| = \|f(\theta(\delta_x))\|. \tag{3.4}$$

Using Equations (3.2), (3.3) and (3.4),

$$\|\delta_y\| \leq \|f(\theta(\delta_x)\| \leq \|\theta\delta_x\| \leq \|\theta\| \, \|\delta_x\|, \tag{3.5}$$
$$\frac{\|\delta_y\|}{\|\delta_x\|} \leq \|\theta\|.$$

This result indicates that a perturbation to the input of a single layer neural network is bounded by the norm of the weight matrix. However, this quantity is only significant relative to the overall magnitude of the data. For

example, what if the network $f(\theta x)$ shrinks the magnitude of every input? Then, even though the quantity $\frac{\|\theta\delta y\|}{\|\delta x\|}$ is small, it may be a large perturbation relative to the magnitude of the initial quantity, $\frac{\|y\|}{\|x\|}$. This intuition drives the motivation that the pertinent quantity to be studied is therefore,

$$\frac{\|\delta_y\|}{\|\delta_x\|} / \left(\frac{\|y\|}{\|x\|}\right) = \left(\frac{\|\delta_y\| \,/\, \|y\|}{\|\delta_x\| \,/\, \|x\|}\right).$$

This is the quantity that was introduced in Equation (3.1). Our estimator is the maximum value of this quantity,

$$\tilde{\kappa} = \max_{x \neq 0} \kappa = \max_{x \neq 0} \left(\frac{\|\delta_y\| \,/\, \|y\|}{\|\delta_x\| \,/\, \|x\|}\right). \tag{3.6}$$

Since we have $\|y\|$ in the denominator which can be zero, this estimator cannot be computed exactly unless it is known that $\|y\| \geq C > 0, \forall\, x \neq 0$ for some positive constant C. Therefore, the estimator $\tilde{\kappa}$ is approximated by calculating a set of κ's for many perturbations δ_x and inputs x.

3.2 Multi-layer Network Estimator Derivation

The analysis of the previous section is now generalized to many-layer networks. Stability and rounding error concerns become more complicated when considering additional layers because each layer could introduce an error from rounding, which is amplified in the subsequent layers. The multi-layer case has the form,

$$\frac{\|\delta_y\|}{\|\delta_x\|} \leq \prod_{j=0}^{i-1} \|\theta_{n-j}\| \,. \tag{3.7}$$

Where $\|\theta_i\|$ is the norm of the weight matrix of layer i from the input side. Considering perturbations at every single layer due to rounding error for a simple feedforward neural network with n layers results in,

$$\frac{\|\delta_y\|}{\|\delta_x\|} \leq \sum_{i=1}^{n} \left(\prod_{j=0}^{i-1} \|\theta_{n-j}\| \right). \tag{3.8}$$

This is the product of all current and previous weight matrices, summed over each layer. The above equation indicates that the deeper layers cause the perturbations to grow because they amplify the rounding errors from previous layers. To aid in intuition, Figure 3.3 shows the computation of $\frac{\|\delta_y\|}{\|\delta_x\|}$ for a simple three layer neural network. The first layer contributes to only one term, while the second layer contributes to two terms, and the last layer is a product of all the previous terms of the estimator.

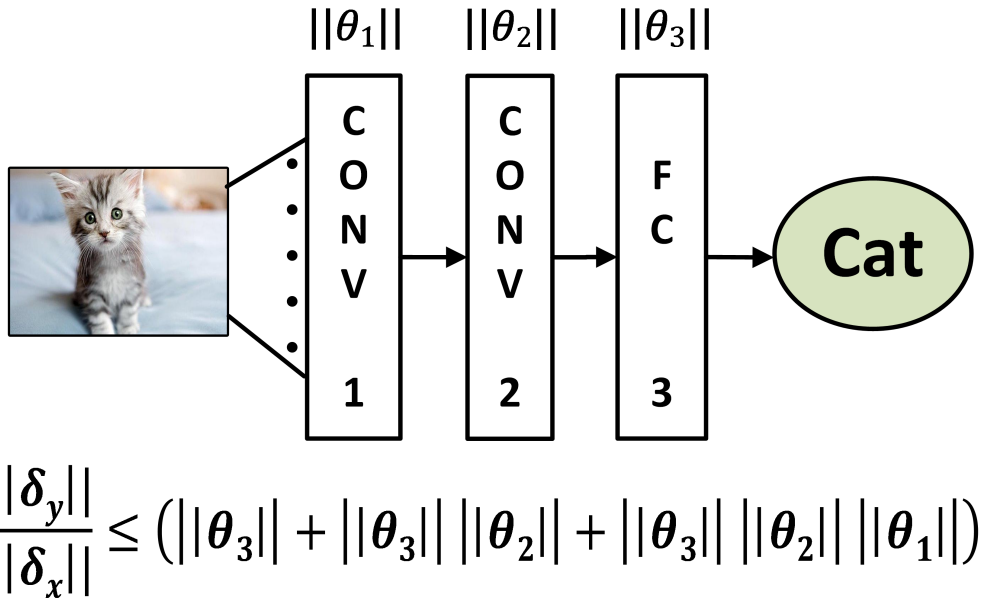


Figure 3.3: Example calculation of estimator in Equation (3.8) for a 3-layer neural network.

Chapter 4

Testing the Estimator with Adversarial Perturbations

The estimator presented in Equation (3.6) provides an upper-bound of the susceptibility of a network to a perturbation in input. A natural question is how close is this bound to perturbations in the network. To explore the tightness of the estimator presented in Equation (3.6), we minimize the quantity, $\|\delta_x\|$. In particular, the smallest perturbations $\|\delta_x\|$ that in turn maximizes the quantity, $\|\delta_y\|$. To efficiently generate such perturbations, we leverage the existing body of work on adversarial perturbations.

Adversarial attacks are methods designed to alter the solution or classifier output of a learning system. For CNNs, an adversarial perturbation, $\Delta(\boldsymbol{x}; \hat{k})$, refer to small perturbations $\boldsymbol{r}$ added to an input image $\boldsymbol{x}$ such that the network classifier $\hat{k}(\boldsymbol{x})$ changes, leading to misprediction. This is formally presented in Equation (4.1) as,

$$\Delta(\boldsymbol{x}; \hat{k}) = \min_{\boldsymbol{r}} \|\boldsymbol{r}\| \text{ subject to } \hat{k}(\boldsymbol{x} + \boldsymbol{r}) \neq \hat{k}(\boldsymbol{x}). \tag{4.1}$$

One common example to explain adversarial perturbations is shown in Figure 4.1. A cat image is being recognized as a cat by the neural network as

expected, but when an adversarial perturbation is added to it, the classifier of the network completely changes and now the cat is identified as guacamole even though the image is visually imperceptible.

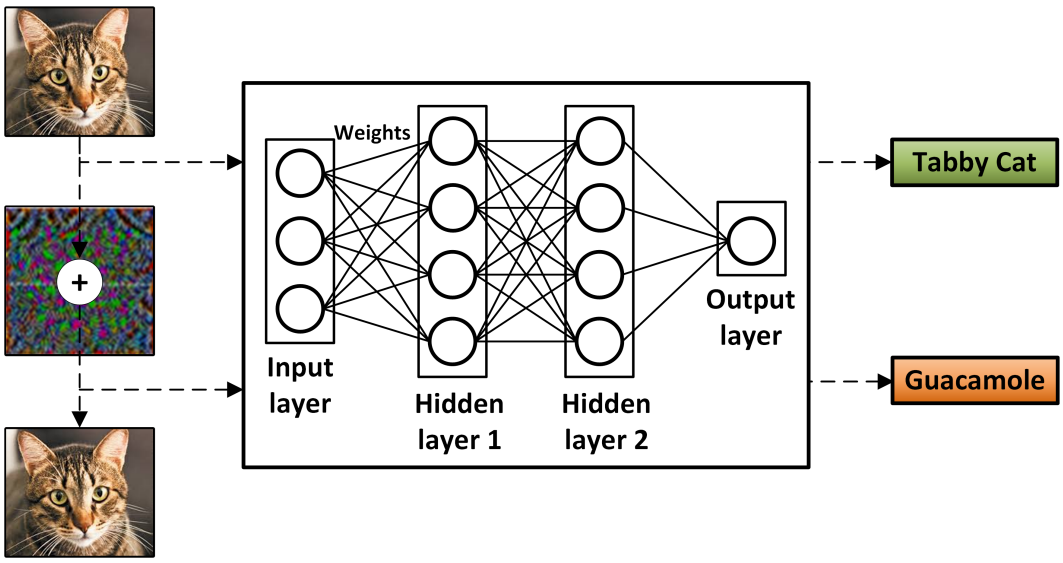


Figure 4.1: One common example of adversarial perturbation.

The techniques used to generate these perturbations are discussed in Section 4.1, and then the results of these perturbations are presented in Section 4.2.

4.1 Techniques to Generate Adversarial Perturbations

Adversarial instability was first explored in [Szegedy et al., 2014]. The authors calculated the adversarial examples by solving a constrained optimization problem and conjectured that those exist because of the complex nature of the neural networks and the probability of adversarial examples is too low

for those to be seen in the test set. However, this method is not scalable to large datasets because the optimization method is time-consuming.

[Nguyen et al., 2015] proposed a method to generate crafted unrecognizable images, which are predicted with high confidence by DNNs. [Tsai and Cox, 2015] have provided a software to misclassify a given image in a specified class on multiple DNNs by using different existing algorithms, without necessarily finding the minimal perturbation vectors. [Papernot et al., 2016] introduced Jacobian-based Saliency Map Attack (JSMA) to construct adversarial saliency maps that identify features of the input that most significantly impact output classification, enabling the construction of network-dependent and input-independent adversarial samples.

[Goodfellow et al., 2015] introduced *"fast gradient sign (FGS)"*, an efficient method to compute adversarial perturbations, where the cost function is linearized around the current value of θ, obtaining an optimal max-norm constrained perturbation $\eta = \epsilon sign(\nabla_x J(\theta, x, y))$, where $J(\theta, x, y)$ is the cost function of the neural network. Essentially, this tries to generate the perturbation vector in the direction of the gradient of the cost function, which is presumably the direction of maximum ascent. In order to keep the perturbation vector small, a small number ϵ is multiplied. However, the unique gradient step often leads to sub-optimal solutions, hence, not yielding optimal perturbations. [Kurakin et al., 2017] improved the FGS method by iteratively taking smaller steps α in the direction of the gradient and clipping the intermediate values to ensure that they are in the ϵ-neighbourhood of the original image.

This is done to generate smaller perturbations which are closer to the minimal ones. However, FGS-based methods do not guarantee misclassification.

Using existing adversarial attacks permits leveraging existing techniques that represent the best-known methods to reliably produce misclassification from small perturbations to input. Existing techniques also have the advantage of providing standard and peer-reviewed methods, making the results more broadly accessible to the community. Based on these criteria, this work focused on the common adversarial attack, DeepFool, which is discussed below.

[Moosavi-Dezfooli et al., 2016] proposed the untargeted attack technique known as DeepFool, which is optimized for the L_2 distance metric. This method is based on an iterative linearization of the classifier to generate minimal perturbations sufficient to change the classification label. Figure 4.2 shows the linearized classification boundaries in dotted line and the actual classification boundaries in solid line. Initially, it is assumed that the neural networks are completely linear, with classifiers separated by hyper-planes. Since neural networks are non-linear, the linearization process is iterated until the classification index changes. In this work, the iterator was observed to converge in less than four iterations for most images. The process of generating the perturbations is computationally inexpensive and this is therefore an effective technique to generate small adversarial perturbations.

[Moosavi-Dezfooli et al., 2017] presents an extension of DeepFool which generates a small, image-agnostic perturbation vector which causes misclassification on a large set of images across a wide variety of classifiers. This

means that an image-specific perturbation vector need not be generated, and a universal perturbation when added to different input images can cause misclassification with probability of about 70% across different networks. This is particularly interesting from the perspective of this thesis, because our method directly provides a bound on the magnitude of the universal perturbation.

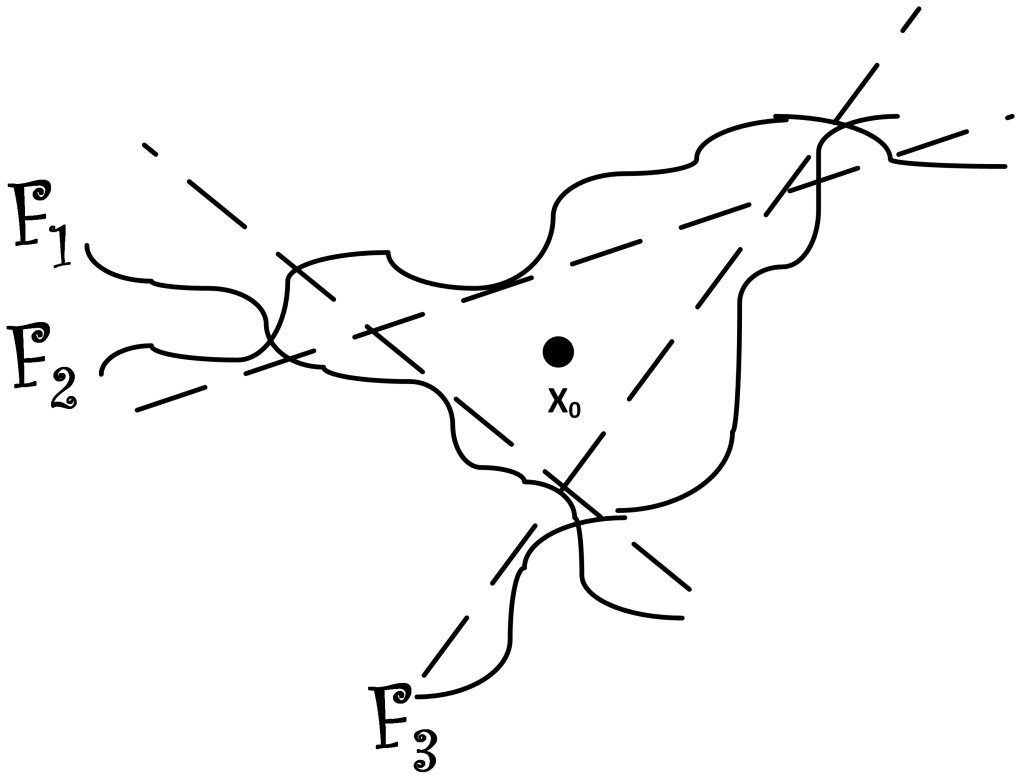
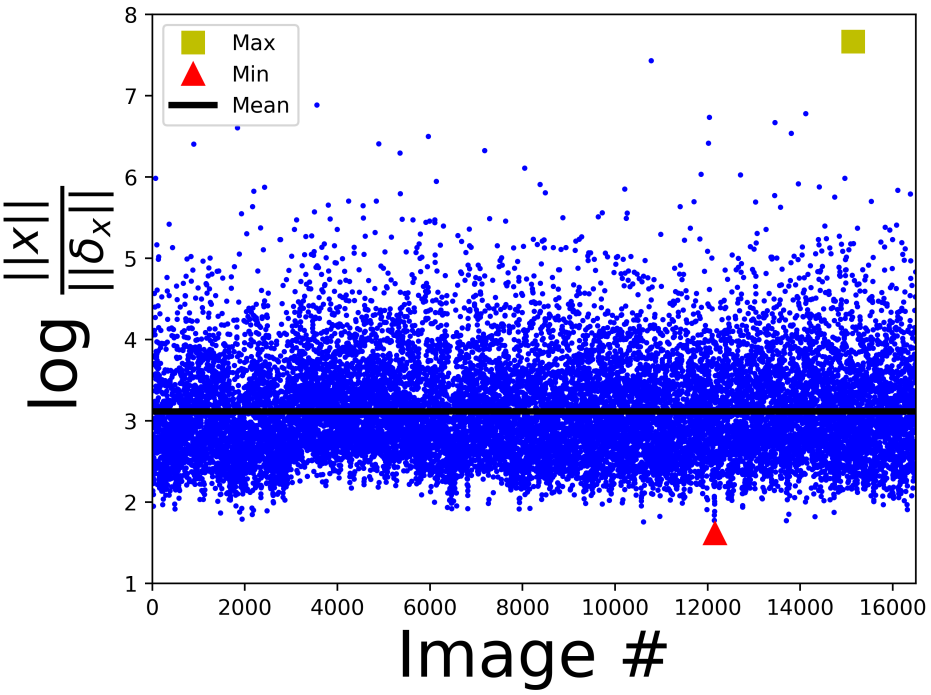


Figure 4.2: Linearization of classification boundaries [Moosavi-Dezfooli et al., 2016].

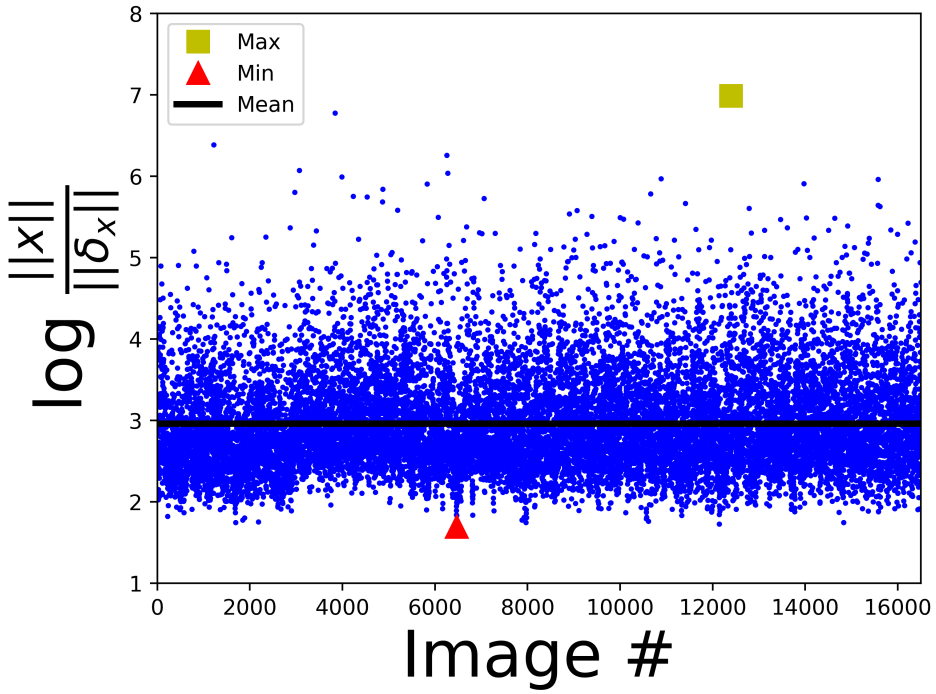
4.2 Generated DeepFool Perturbations

Figure 4.3 shows the magnitudes of DeepFool perturbations generated for AlexNet and VGG-19 measured relative to the original image, or, $\|\delta_x\|$, over the norm of the input image, $\|x\|$. Since this quantity is much less than 1, the results are plotted by the reciprocal on a logarithmic scale, or $\log(\frac{\|x\|}{\|\delta_x\|})$. The x-axis spans different images, and the y-axis details the resulting perturbation size. For this analysis, 16,500 test images were used. This sampled ImageNet with 20 randomly chosen images from each class.



(a) AlexNet

Figure 4.3: Continued on next page.



(b) VGG-19

Figure 4.3: Relative magnitudes of generated DeepFool perturbations to cause misclassification.

Table 4.1 summarizes the statistics in Figure 4.3. The smallest perturbations relative to the image are of the order of 10^{-8} for AlexNet and 10^{-7} for VGG-19. Even on average, the perturbations are of the order of 10^{-3} for the two networks.

Table 4.1: Statistics of generated DeepFool perturbations.

Statistics	**AlexNet**	**VGG-19**
Mean	1.53E-03	2.05E-03
Maximum	2.43E-02	2.03E-02
Minimum	2.16E-08	1.03E-07

Note that all these generated perturbations, when added to the original image, lead to misclassification by the network. Figure 4.4 shows some interesting misclassifications by AlexNet when DeepFool perturbations are added to the input images. In each case, the resulting perturbed images are indistinguishable to the human eye and yet, result in misclassification by the network.

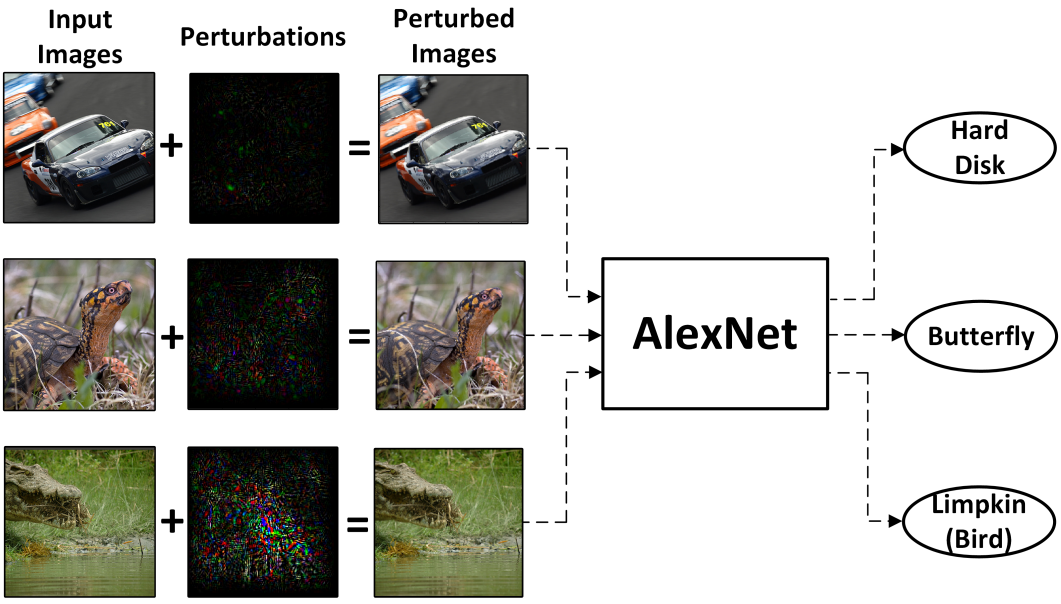


Figure 4.4: Some interesting misclassifications.

Chapter 5

Results

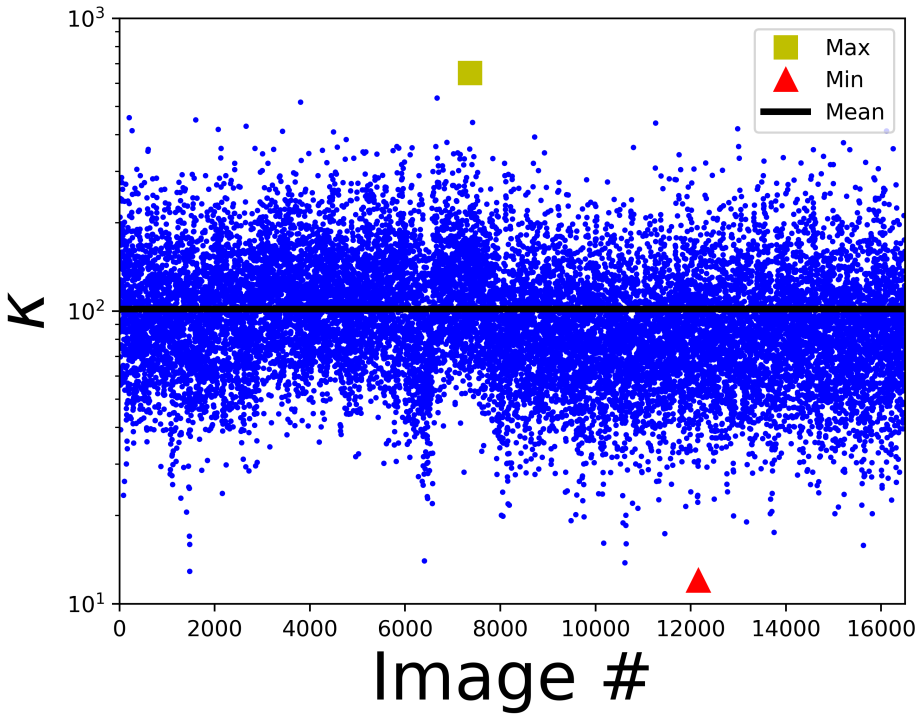
This section compares the condition number (calculated with Equation (3.6)) to the generalized of small perturbation vectors from the previous section. Note that in this quantity, $\frac{\|\delta_y\|}{\|\delta_x\|}$ and $\|\delta_y\|$ are computed as the magnitude of the element-wise difference of the pre-final output layer, i.e. the layer before the Softmax layer between the original and perturbed scenarios. We then compare the estimator to empirical data generated via adversarial attacks and random noise.

5.1 DeepFool Perturbations

The quantity κ presented in Equation (3.6) was computed for AlexNet and VGG-19 using the generated DeepFool perturbations applied to each of the 16500 images discussed in Chapter 4. The κ values are shown in Figure 5.1. The X-axis is the image number and Y-axis shows the the maximum value of κ for each image. All of the 16,500 data are rendered as blue circles. The red triangle indicates the perturbation that resulted in the smallest change in output to the network. The yellow square indicates the perturbation that caused the largest change in the network output. These plots are similar,

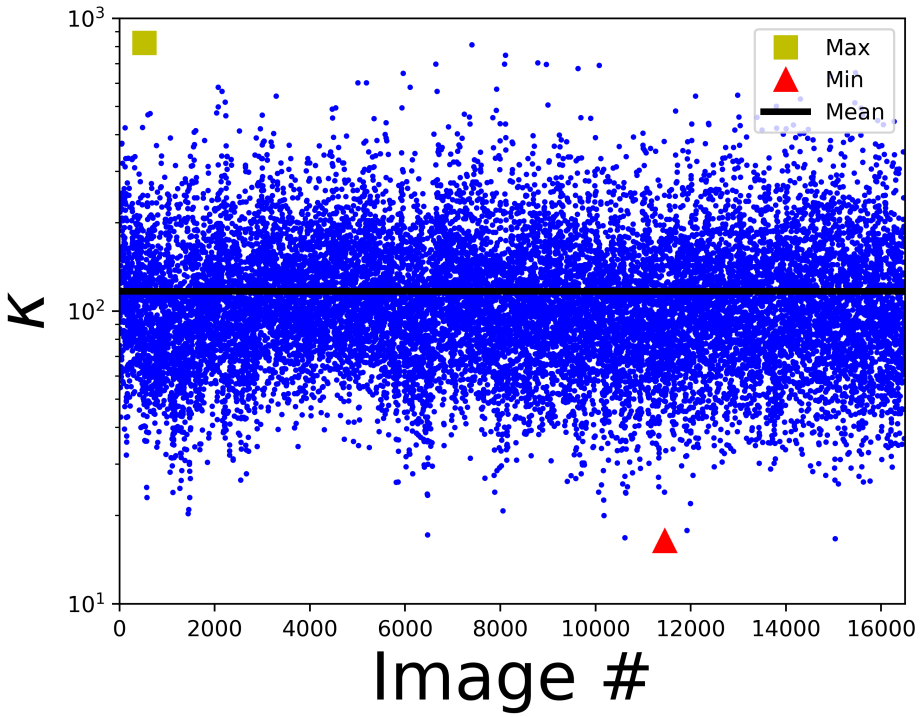
indicating that the networks have similar susceptibility to perturbations in input. For either network, the range of results spans many orders of magnitude, indicating a wide range of possible impact from a perturbation.

The maximum κ value created by applying the DeepFool perturbations to inputs to AlexNet is about 651, the average image has a maximum κ value of about 100, and all images have a maximum κ value of at least of 12. For VGG-19, the maximum κ value is about 825, the average image has a maximum κ value of about 117, and all images have a maximum κ value of at least 16.



(a) AlexNet

Figure 5.1: Continued on next page.



(b) VGG-19

Figure 5.1: κ estimator values for DeepFool perturbations.

Table 5.1 shows the precision requirements for input representations of AlexNet and VGG-19 based on the mean, maximum and minimum value of κ. The maximum value of κ gives maximum amount a small perturbation to the input of the neural network maybe amplified by the network.

In practice, the error in the output of the network can be expected to be $\epsilon\tilde{\kappa}$, where ϵ is machine epsilon which gives the maximum relative error due to rounding in floating point arithmetic and $\tilde{\kappa}$ is defined in Equation (3.6). Thus, $\tilde{\kappa}$ gives an estimate of the minimum precision that should be used with

a particular neural network. The minimum digits required can be calculated as $log_{10}(\kappa)$. The minimum number of bits is calculated as shown in Equation (5.1).

$$\text{Min. \# of bits} = \lceil log_2(\kappa) \rceil + 1 \tag{5.1}$$

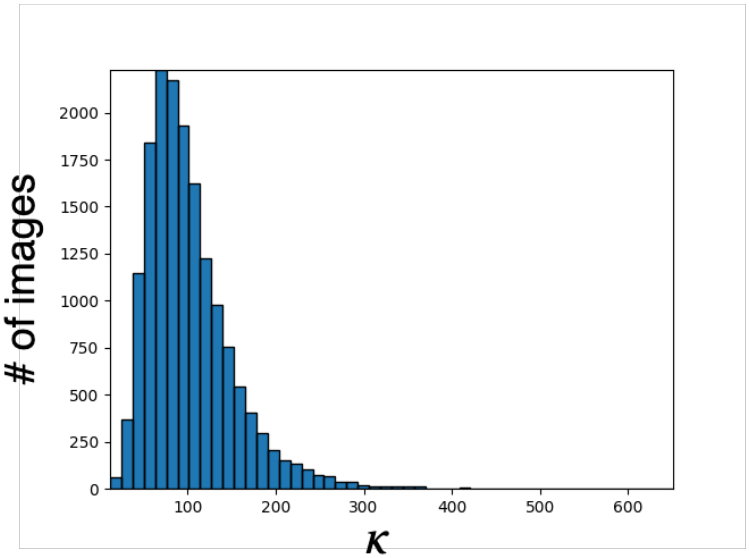
Table 5.1: Precision requirements for input representations based on DeepFool perturbations.

Network	**AlexNet**	**VGG-19**
Mean κ	101.78	116.84
Minimum Digits	2.01	2.07
Minimum Bits	8	8
Maximum κ	651.06	824.53
Minimum Digits	2.81	2.92
Minimum Bits	11	11
Minimum κ	12.00	16.37
Minimum Digits	1.08	1.21
Minimum Bits	5	6

For the worst-case generated perturbations, 10 bits of precision could be lost for both AlexNet and VGG-19 and hence, 11 bits are required for input representations at a minimum. For the mean scenario, 8 bits of precision are required for both networks, whereas the minimum κ says that AlexNet requires 5 bits of precision while VGG-19 requires 6 bits. This shows that moving to *"int8"*, a commonly supported precision in most processors, for representing inputs of these networks may make them more likely to misclassify inputs. However, these perturbations are very specially generated to maximize their

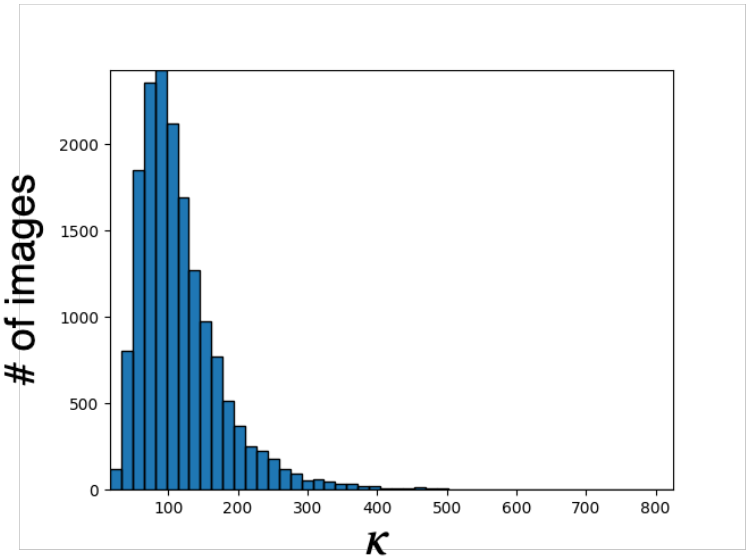
impact on the network and one may question how often perturbations like this will be seen in practice.

Figure 5.2 shows the distribution of 16500 images over κ values for both AlexNet and VGG-19. Clearly, most of the values are distributed around the mean and the distribution is positively skewed, meaning that the median of the distribution is smaller than the mean. For most images, κ value is either lesser or slightly higher than the mean. κ value is much higher than the mean only for some particular images.



(a) AlexNet

Figure 5.2: Continued on next page.



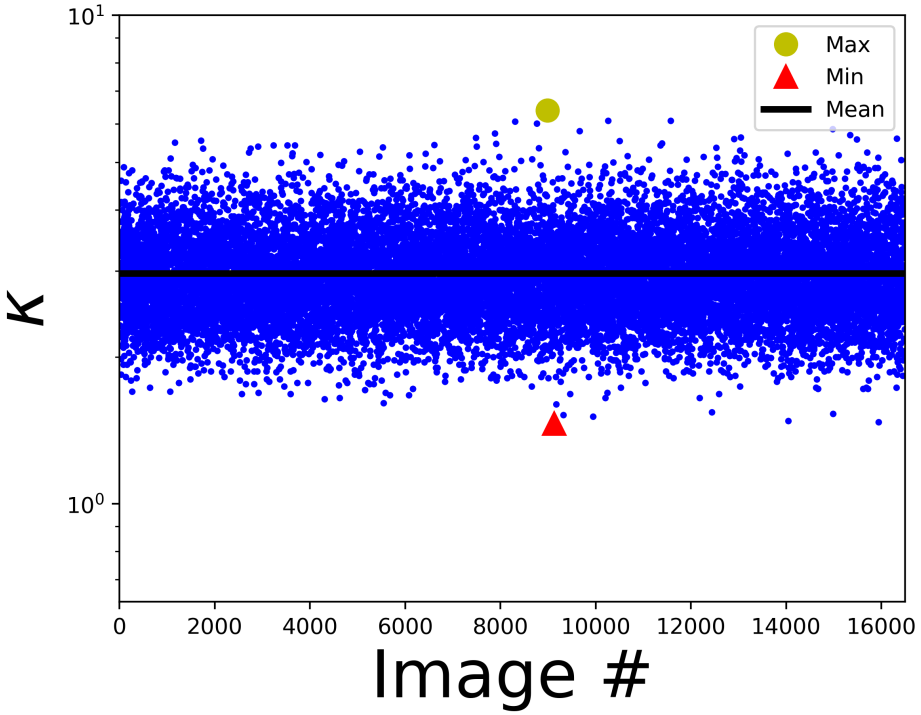
(b) VGG-19

Figure 5.2: Distribution of 16500 images over κ estimator values.

5.2 Random Perturbations

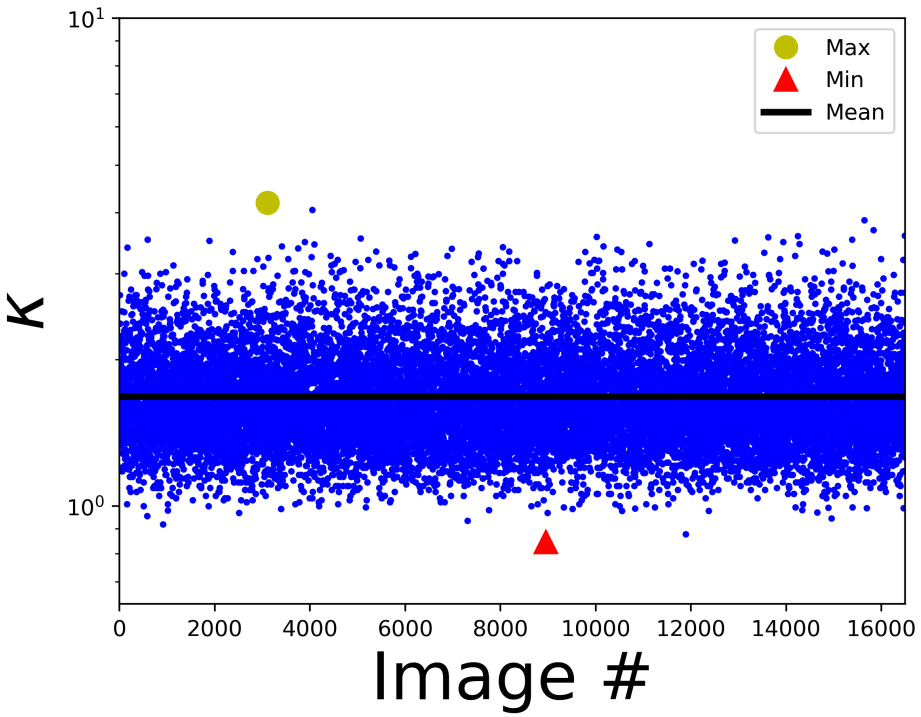
The DeepFool perturbations are deliberately constructed to cause misclassification by the network. An important question is what impact will indiscriminate sources of perturbations, such as rounding error or noisy data, have on the output of neural networks. To test this, 16500 random perturbations of same magnitude as the DeepFool perturbations were generated. Those perturbations were added those to the same set of images used in Section 5.1 and again the maximum value of κ was estimated for each image. The κ values are shown in Figure 5.3. The X-axis is the image number and Y-axis shows the the maximum value of κ for each image. All of the 16,500 data are rendered

as blue circles. The red triangle indicates the perturbation that resulted in the smallest change in output to the network. The yellow square indicates the perturbation that caused the largest change in the network output.



(a) AlexNet

Figure 5.3: Continued on next page.



(b) VGG-19

Figure 5.3: κ estimator values for random perturbations.

As can be seen from Table 5.2, the κ values observed for these random perturbations are much smaller than in Section 5.1. This suggests that for applications where the consequences of infrequent misclassification are not severe, and even lower precision than 11 bits may suffice.

Table 5.2: Precision requirements for input representations based on random perturbations.

Network	**AlexNet**	**VGG-19**
Mean Estimator	1.68	2.96
Minimum Digits	0.23	0.47
Minimum Bits	2	3
Maximum Estimator	4.19	6.39
Minimum Digits	0.62	0.81
Minimum Bits	4	4

Chapter 6

Conclusion and Future Work

In this thesis, we use the methods of numerical analysis to derive estimators which can predict the sensitivity of a neural network to random as well as crafted perturbations. These estimators can help in estimating the minimum precision requirements for input representations of various neural networks. We show the results for two widely studied CNN architectures, AlexNet and VGG-19.

This work can have an impact at different levels of design hierarchy. It can help in the decision of precision support required at the hardware level when designing machine learning accelerators, thus enabling energy-efficient edge computing where power consumption is a major bottleneck. At the software level, this can help in estimating the maximum evaluation error required for certification, validation or quantification of uncertainty. At the algorithmic level, this work can help in better understanding the underlying characteristics of deep neural networks and enable the design of efficient DNN architectures which can be resistant to noise or adversarial attacks.

In the future, we would like to extend this work by deriving the analytical expression for the condition number of a neural network such that it can

be made free of any empirical analysis. Further, we need to introduce perturbations at each layer to mimic perturbing the weights and make the estimator more robust as this may amplify the effect of random perturbations. That would also help us in establishing a tight bound for the estimator and extend it to calculate the precision requirements for activations, weights and biases for each layer. The final objective would be to come up with an estimator which can be extended to a wide variety of neural networks - Recurrent Neural Networks (RNNs), Reinforcement Learning (RL), Generative Adversarial Networks (GANs), etc. and not just be limited to CNNs.

Appendix

Appendix 1

Derivation of bound for other activation functions

The bound presented in Chapter 3 was derived for the most common activation function, Rectified Linear Unit (ReLU). Here, we derive the bounds for other commonly used activation functions as well and show that the results presented in the thesis hold true.

1.1 Leaky Rectified Linear Unit (Leaky ReLU)

$$f(z) = \begin{cases} z, & \text{for } z > 0 \\ \alpha z, & \text{for } z \leq 0; \quad \alpha = \text{small constant (e.g. } 0.1) \end{cases}$$

$$\|f(z)\| = \|\text{Max}(\alpha z, z)\| \leq \|z\|$$

1.2 Exponential Linear Unit

$$f(z) = \begin{cases} z, & \text{for } z > 0 \\ \alpha(e^z - 1), & \text{for } z \leq 0; \quad \alpha = \text{small constant (e.g. } 0.1) \end{cases}$$

$$0 < \|e^z\| \leq 1 \text{ for } z \leq 0$$

$$0 \leq \|e^z - 1\| < 1 \text{ for } z \leq 0$$

$$\|a\| \, \|e^z - 1\| \leq \|\alpha\| \text{ for } z \leq 0$$

$$\|f(z)\| = \|\text{Max}(\alpha, z)\|$$

1.3 Sigmoid

$$f(z) = \frac{1}{1 + e^{-z}}$$

$$\left\|1 + e^{-z}\right\| > 1$$

$$\|f(z)\| < 1$$

1.4 Hyberbolic Tangent

$$f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$\|e^z\| > 0, \left\|e^{-z}\right\| > 0$$

$$(\|e^z\| - \left\|e^{-z}\right\|) < (\|e^z\| + \left\|e^{-z}\right\|)$$

$$\|f(z)\| < 1$$

Bibliography

Peter L. Bartlett. For valid generalization, the size of the weights is more important than the size of the network. In *International Conference on Neural Information Processing Systems (NIPS)*, 1996. URL `http://dl.acm.org/citation.cfm?id=2998981.2999000`.

Peter L. Bartlett, Dylan J. Foster, and Matus Telgarsky. Spectrally-normalized margin bounds for neural networks. In *International Conference on Neural Information Processing Systems (NIPS)*, 2017. URL `http://dl.acm.org/citation.cfm?id=3295222.3295372`.

Yu-Hsin Chen, Tushar Krishna, Joel Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In *IEEE International Solid-State Circuits Conference (ISSCC)*, 2016. doi: 10.1109/ISSCC.2016.7418007. URL `https://ieeexplore.ieee.org/document/7418007`.

Jungwook Choi, Swagath Venkataramani, Vijayalakshmi Srinivasan, Kailash Gopalakrishnan, Zhuo Wang, and Pierce Chuang. Accurate and efficient 2-bit quantized neural networks. In *Conference on Systems and Machine Learning (SysML)*, 2019. URL `https://www.sysml.cc/doc/2019/168.pdf`.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *International Conference on Neural Information Processing Systems (NIPS)*, 2015. URL http://dl.acm.org/citation.cfm?id=2969442.2969588.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. In *arXiv preprint arXiv:1602.02830*, 2016. URL https://arxiv.org/abs/1602.02830.

Ian J. Goodfellow, Jonathan Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/abs/1412.6572.

Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning (ICML)*, 2015. URL http://proceedings.mlr.press/v37/gupta15.html.

Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, Raquel Urtasun, and Andreas Moshovos. Reduced-precision strategies for bounded memory in deep neural nets. In *arXiv preprint arXiv:1511.05236*, 2015. URL https://arxiv.org/abs/1511.05236.

Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M. Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Proteus: Exploiting numeri-

cal precision variability in deep neural networks. In *International Conference on Supercomputing (ICS)*, 2016. URL http://doi.acm.org/10.1145/2925426.2926294.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *International Conference on Neural Information Processing Systems (NIPS)*, 2012. URL http://dl.acm.org/citation.cfm?id=2999134.2999257.

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *International Conference on Learning Representations (ICLR) Workshop Track*, 2017. URL https://arxiv.org/abs/1607.02533.

Griffin Lacey, Graham W. Taylor, and Shawki Areibi. Stochastic layer-wise precision in deep neural networks. In *arXiv preprint arXiv:1807.00942*, 2018. URL https://arxiv.org/abs/1807.00942.

Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. In *arXiv preprint arXiv:1605.04711*, 2016. URL https://arxiv.org/abs/1605.04711.

Tengyuan Liang, Tomaso Poggio, Alexander Rakhlin, and James Stokes. Fisher-rao metric, geometry, and complexity of neural networks. In *arXiv preprint arXiv:1711.01530*, 2017. URL https://arxiv.org/abs/1711.01530.

Ji Lin, Chuang Gan, and Song Han. Defensive quantization: When efficiency meets robustness. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://arxiv.org/abs/1904.08444.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. URL https://arxiv.org/abs/1511.04599.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. URL https://arxiv.org/abs/1610.08401.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *Conference on Learning Theory (COLT)*, 2015. URL https://arxiv.org/abs/1503.00036.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. URL https://arxiv.org/abs/1412.1897.

Joachim Ott, Zhouhan Lin, Ying Zhang, Shih-Chii Liu, and Yoshua Bengio. Recurrent neural networks with limited numerical precision. In *arXiv preprint arXiv:1608.06902*, 2017. URL https://arxiv.org/abs/1608.06902.

Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *IEEE European Symposium on Security & Privacy (EuroS&P)*, 2016. URL `https://arxiv.org/abs/1511.07528`.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision (ECCV)*, 2016. URL `https://arxiv.org/abs/1603.05279`.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3): 211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Charbel Sakr, Yongjune Kim, and Naresh Shanbhag. Analytical guarantees on numerical precision of deep neural networks. In *International Conference on Machine Learning (ICML)*, 2017. URL `http://proceedings.mlr.press/v70/sakr17a.html`.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015. URL `https://arxiv.org/abs/1409.1556`.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural

networks. In *International Conference on Learning Representations (ICLR)*, 2014. URL https://arxiv.org/abs/1312.6199.

C.-Y. Tsai and D. Cox. Are deep learning algorithms easily hackable?, 2015. URL http://coxlab.github.io/ostrichinator.

Hantian Zhang, Jerry Li, Kaan Kara, Dan Alistarh, Ji Liu, and Ce Zhang. Zipml: Training linear models with end-to-end low precision, and a little bit of deep learning. In *International Conference on Machine Learning (ICML)*, 2017. URL http://proceedings.mlr.press/v70/zhang17e.html.

Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. In *arXiv preprint arXiv:1606.06160*, 2016. URL https://arxiv.org/abs/1606.06160.

Vita

Naman Maheshwari was born in Delhi, India, the son of Mr. Mukesh Maheshwari and Dr. Krishna Maheshwari. He received the Bachelor of Engineering (Hons.) degree in Electrical and Electronics from Birla Institute of Technology & Science - Pilani, Pilani Campus, India in 2015 and had the opportunity to work on Approximate Multiplier Circuits during his internship at the University of Alberta, Edmonton, Canada. He worked as an ASIC Design Engineer in Design-for-Testability (DFT) domain as part of the Automotive Radar team at Texas Instruments, Bangalore, India. He started his graduate studies in Electrical and Computer Engineering (Integrated Circuits & Systems track) at The University of Texas at Austin in August, 2017. He interned with the SoC Physical Design team at Apple Inc., Cupertino in Summer 2018 and did a co-op in the Applications, Software & Technology team at Advanced Micro Devices (AMD) Research, Austin in Fall 2018.

Email address: naman@utexas.edu

This thesis was typeset with LaTeX[†] by 'the author'.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.