

Sensitivity of Feedforward Neural Networks to Harsh Computing Environments

Austin P. Arechiga

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Alan J. Michaels, Chair

Jonathan T. Black

Ryan K. Williams

June 22, 2018

Blacksburg, Virginia

Keywords: Machine Learning, Fault Tolerance, Single Event Upsets

Copyright 2018, Austin P. Arechiga

Sensitivity of Feedforward Neural Networks to Harsh Computing Environments

Austin P. Arechiga

(ABSTRACT)

Neural Networks have proven themselves very adept at solving a wide variety of problems, in particular they excel at image processing. However, it remains unknown how well they perform under memory errors. This thesis focuses on the robustness of neural networks under memory errors, specifically single event upset style errors where single bits flip in a network's trained parameters. The main goal of these experiments is to determine if different neural network architectures are more robust than others. Initial experiments show that MLPs are more robust than CNNs. Within MLPs, deeper MLPs are more robust and for CNNs larger kernels are more robust. Additionally, the CNNs displayed bimodal failure behavior, where memory errors would either not affect the performance of the network, or they would degrade its performance to be on par with random guessing. VGG16, ResNet50, and InceptionV3 were also tested for their robustness. ResNet50 and InceptionV3 were both more robust than VGG16. This could be due to their use of Batch Normalization or the fact that ResNet50 and InceptionV3 both use shortcut connections in their hidden layers. After determining which networks were most robust, some estimated error rates from neutrons were calculated for space environments to determine if these architectures were robust enough to survive. It was determined that large MLPs, ResNet50, and InceptionV3 could survive in Low Earth Orbit on commercial memory technology and only use software error correction.

Sensitivity of Feedforward Neural Networks to Harsh Computing Environments

Austin P. Arechiga

(GENERAL AUDIENCE ABSTRACT)

Neural networks are a new kind of algorithm that are revolutionizing the field of computer vision. Neural networks can be used to detect and classify objects in pictures or videos with accuracy on par with human performance. Neural networks achieve such good performance after a long training process during which many parameters are adjusted until the network can correctly identify objects such as cats, dogs, trucks, and more. These trained parameters are then stored in a computer's memory and then recalled whenever the neural network is used for a computer vision task. Some computer vision tasks are safety critical, such as a self-driving car's pedestrian detector. An error in that detector could lead to loss of life, so neural networks must be robust against a wide variety of errors. This thesis will focus on a specific kind of error: bit flips in the parameters of a neural network stored in a computer's memory. The main goal of these bit flip experiments is to determine if certain kinds of neural networks are more robust than others. Initial experiments show that MLP (Multilayer Perceptions) style networks are more robust than CNNs (Convolutional Neural Network). For MLP style networks, making the network deeper with more layers increases the accuracy and the robustness of the network. However, for the CNNs increasing the depth only increased the accuracy, not the robustness. The robustness of the CNNs displayed an interesting trend of bimodal failure behavior, where memory errors would either not affect the performance of the network, or they would degrade its performance to be on par with random guessing. A second set of experiments were run to focus more on CNN robustness because CNNs are much more capable than MLPs. The second set of experiments focused on

the robustness of VGG16, ResNet50, and InceptionV3. These CNNs are all very large and have very good performance on real world datasets such as ImageNet. Bit flip experiments showed that ResNet50 and InceptionV3 were both more robust than VGG16. This could be due to their use of Batch Normalization or the fact that ResNet50 and InceptionV3 both use shortcut connections within their network architecture. However, all three networks still displayed the bimodal failure mode seen previously. After determining which networks were most robust, some estimated error rates were calculated for a real world environment. The chosen environment was the space environment because it naturally causes a high amount of bit flips in memory, so if NASA were to use neural networks on any rovers they would need to make sure the neural networks are robust enough to survive. It was determined that large MLPs, ResNet50, and InceptionV3 could survive in Low Earth Orbit on commercial memory technology and only use software error correction. Using only software error correction will allow satellite makers to build more advanced satellites without paying extra money for radiation-hardened electronics.

Dedication

To my family who encouraged me, and my friends who made it bearable

Acknowledgments

In addition to my committee, Chris Headley provided much sage advice and guidance through my graduate school experience. I'd also like to thank my labmates for providing proofreading, bug fixing, and camaraderie.

Contents

List of Figures	x
List of Tables	xiv
1 Introduction	1
1.1 Contributions	6
2 Neural Network Basics and Previous Work	9
2.1 Neural Network Background	9
2.1.1 Multilayer Perceptron Description	9
2.1.2 Convolutional Neural Network Description	10
2.2 Datasets	13
2.2.1 MNIST	13
2.2.2 CIFAR-10	13
2.2.3 ImageNet	14
2.3 Hardware	15
2.3.1 Memory Hierarchy	15
2.3.2 Memory Technology	16
2.4 Related Work	18

3	Comparing MLP and CNN Robustness	22
3.1	Tested Architectures	22
3.2	Results	24
3.2.1	Methodology	24
3.2.2	MLP Results	25
3.2.3	CNN Results	29
3.3	Future Work	36
3.4	Conclusion	36
4	Robustness of Deep Learning Architectures	37
4.1	Tested Architectures	38
4.1.1	VGG16	38
4.1.2	ResNet50	38
4.1.3	InceptionV3	39
4.2	Results	40
4.2.1	Methodology	40
4.2.2	VGG16 Results	41
4.2.3	ResNet50 Results	43
4.2.4	InceptionV3 Results	45
4.2.5	Comparison of Results	48

4.3	Future Work	49
4.4	Conclusion	49
5	Survivability in Real World Environments	51
5.1	Neural Network Space Application	51
5.2	Calculating Error Rates	56
5.3	Final Remarks	61
6	Conclusions	63
6.1	Future Work	66
	Bibliography	69

List of Figures

2.1	A single neuron in a neural network along with the equation describing its behavior. W_n represent the weights on each input, while B represents the Bias of the whole neuron. Sigma represents an activation function.	10
2.2	Illustration of how the neurons are connected in a small MLP. Notice that every neuron in one layer is connected to every other neuron in the next layer.	11
2.3	In convolutional neural networks the neurons are arranged into kernels, which then convolve or slide across the input to generate the output data. The size and number of kernels can be adjusted.	12
2.4	Example of how a max pooling layer filters the data by only allowing the max value in its pooling window through. In this case, the window is 2x2 and overlaps by one.	12
2.5	Three examples of images from the MNIST dataset. From left to right are the numeral zero, numeral six, and numeral three. The numeral six illustrates that some examples can be difficult to classify.	13
2.6	Three examples of images from the Cifar10 dataset. From left to right are an example of a truck, automobile, and ship. The low resolution of the images can make classifying these images challenging.	14
2.7	Three examples of images from the ImageNet dataset. From left to right are an example of a Shetland sheepdog, crane, and safe. The classes in ImageNet are very specific and the images are not all the same size.	14

2.8	This figure shows a diagram for a six transistor SRAM memory cell. Transistors T1, T2, T3, and T4 make up the dual inverters, while T5 and T6 are the access transistors. WL is the word line and BL is the bit line.	17
2.9	This figure shows a diagram for a standard DRAM cell. Zero voltage stored in the capacitor stands for a logical 0, while high voltage stands for a logical 1. The transistor is used to access the data in the capacitor.	18
3.1	The performance of the OneLayer MLP network with weight errors. The bar height represents the number of times a certain accuracy occurs over the 200 trials for each number of weights in error.	26
3.2	The performance of the ThreeLayer MLP network with weight errors. The bar height represents the number of times a certain accuracy occurs over the 200 trials for each number of weights in error.	27
3.3	The performance of the FiveLayer MLP network with weight errors. The bar height represents the number of times a certain accuracy occurs over the 200 trials for each number of weights in error.	28
3.4	The performance of the TwoLayersSmall network with weight errors. The bar height represents the number of times a certain accuracy occurs over the 200 trials for each number of weights in error.	30
3.5	The performance of the TwoLayersLarge network with weight errors. The bar height represents the number of times a certain accuracy occurs over the 200 trials for each number of weights in error.	31

3.6	The performance of the FourLayersSmall network with weight errors. The bar height represents the number of times a certain accuracy occurs over the 200 trials for each number of weights in error.	32
3.7	The performance of the FourLayersLarge network with weight errors. The bar height represents the number of times a certain accuracy occurs over the 200 trials for each number of weights in error.	34
4.1	An example of a residual connection. Note how the shortcut connection skips two weight layers, not just one. In ResNet50 these shortcut connections are only over convolutional layers. Sometimes the shortcut connection will have 1x1 convolutions in it to correct the dimensions of the data.	39
4.2	An example of an inception layer. This is a simple example from early versions of the Inception architecture. In InceptionV3 there are multiple layers between the input and concatenation stage.	40
4.3	The performance of the VGG16 model under various weight errors. For each percentage of weights in error shown on the x axis, the trend lines show how many of the 100 trials ended up at greater than 80% accuracy or less than 10% accuracy.	42
4.4	The performance of the ResNet model under various weight errors. For each percentage of weights in error shown on the x axis, the trend lines show how many of the 100 trials ended up at greater than 80% accuracy or less than 10% accuracy.	44

4.5	The performance of the ResNet model under various weight errors. For each percentage of weights in error shown on the x axis, the trend lines show how many of the 100 trials ended up at greater than 80% accuracy or less than 10% accuracy.	45
4.6	The performance of the InceptionV3 model under various weight errors. For each percentage of weights in error shown on the x axis, the trend lines show how many of the 100 trials ended up at greater than 80% accuracy or less than 10% accuracy.	46
4.7	The performance of the InceptionV3 model under various weight errors. For each percentage of weights in error shown on the x axis, the trend lines show how many of the 100 trials ended up at greater than 80% accuracy or less than 10% accuracy.	47
5.1	An example of imagery from the Open California dataset. This image shows part of the San Francisco Bay.	52
5.2	The output of the Canny edge detection algorithm. The detected edge pixels are white, while all other pixels are black	53
5.3	This figure shows different image chips from sliding an 80x80 window over the satellite image after edge detection. (a) is from over land, and has a high number of white pixels. (b) is a boat on the water and has a small number of white pixels. (c) is over open ocean and has zero white pixels.	54
5.4	The output of the ship detection algorithm on the image Sfbay_3. The green boxes represent image chips classified as a ship. There are a few false positives, but no missed ships.	55

List of Tables

3.1	Overview of MLP Architectures	23
3.2	Overview of CNN Architectures	24
3.3	Neural Network Performance Under 100 Weight Errors	36
4.1	Tested Model Data	41
4.2	Overview of Results	50
5.1	Error Rates of Neural Networks in Different Space Radiation Environments .	58
5.2	Survival Times of Neural Networks in Different Space Radiation Environments	61

List of Abbreviations

CNN Convolutional Neural Network

ECC Error Correction Code

ISS International Space Station

LEO Low Earth Orbit

MLP Multi Layer Perceptron

MLPs are a type of neural network characterized by dense connections between the layers. Every neuron in one layers is connected to every neuron in the next layer.

CNNs are a type of neural network that use different sized filters convolved across the input to generate the next layer of data.

The ISS is a large space station used by the international community for research. It orbits in LEO and houses astronauts from various countries for months long missions.

LEO is a range of altitudes where the lowest satellites orbit the earth. In general the range of LEO orbits is between 300 km and 1000 km.

ECC is a method for for detecting single or multi-bit errors in memory. This is done by adding extra bits to data in memory, and when that data is read the extra bits are used to check and correct errors.

Chapter 1

Introduction

Neural networks are enjoying a strong resurgence. They are rapidly moving out of research labs and into the commercial world. Neural networks' greatest successes have been in computer vision tasks [28] [52] [1], but they have also been applied to tasks such as language processing [14], robotic control [20], and signal processing [18] [63].

Now that neural networks are being used in real world environments, the robustness of the neural networks is becoming an important research field. The robustness of neural networks is important because neural networks are being used for applications where lives are at risk. Take for instance a neural network used to classify signs for a driverless car. The consequences of mis-classifying a stop sign could be fatal. Another example is the use of facial recognition software. If law enforcement use facial recognition to help identify criminals, and the neural network powering the facial recognition outputs an incorrect name, then people could be accused or convicted of crimes they did not commit. These are strong arguments for why neural networks need to have high accuracy, but also why they must be robust against induced errors that could degrade their accuracy.

Robustness is an important design parameter across many different fields. From biology, Kitano [25] defines robustness as “a property that allows a system to maintain its functions despite external or internal perturbations.” From computer engineer, Gribble [15], similarly defines robustness as “the ability of a system to continue to operate correctly across a wide range of operational conditions, and to fail gracefully outside of that range.” Finally, Dr.

Genichi Taguchi [56], a pioneer in robust design, defined robust products as delivering “a strong signal regardless of external noise.” From all of these quotes, a common idea of robustness emerges. In this thesis, the robustness of neural networks will be defined as how well the neural network maintains its trained classification accuracy in the presence of errors during execution.

There are a variety of reasons for errors to occur during the execution of a neural network. Bad sensor readings might end up outside the expected input range of a neural network, leading to bad outputs. If a neural network was trained on a large server machine using floating point math, but deployed to a small embedded device only capable of fixed point math, then many errors could occur during execution. Memory errors are another cause of performance degradation, where bits can flip in data stored on memory chips such as Static or Dynamic Random Access Memory (SRAM or DRAM). In this thesis, the effect of memory errors on the execution of neural networks is explored. Memory errors are important to consider because neural networks can take up a large amount of memory and their trained parameters must exist in memory during execution.

There are several reasons for memory errors to occur during execution. One natural cause of errors are cosmic rays impacting memory hardware and causing bit flips [50]. Although Earth’s magnetic field keeps most harmful radiation away from the surface of the Earth, data centers can still be affected because they offer a large target for stray particles [36] [22] [48]. In Meza et al. [36], the error probability of Facebook servers is examined, and they found that around 9% of their servers experienced a correctable memory error, and the most common cause of those errors are spurious events such as cosmic ray strikes or alpha particle decay from the memory packaging. So neural networks being executed on servers are vulnerable to the higher error rates of servers. This is a popular software architecture to bring machine learning to mobile devices [31]. Instead of having constrained devices such as cell

phones execute neural networks, the devices just collect relevant data and send the data to a server for processing. To continue the Facebook example, if Facebook uses the DeepFace architecture to identify users in images [57], mislabeling could occur if the server running DeepFace is one of the unlucky 9% experiencing errors.

It is rare for personal computers or embedded systems to encounter memory errors at the same rate as large cloud computing centers in nominal conditions, but there are environments that pose hazards to any computers. One such example is the area around the Fukushima reactors. After a large earthquake and tsunami, three reactor cores at the Fukushima Daiichi suffered meltdowns due to a total loss of power. Radiation levels were very high in the reactor buildings and some areas were deemed unsafe for humans. Instead, robots were used to explore the wreckage and help clean up radioactive debris. These robots needed to be shielded against the high level of radiation that cause damage to silicon chips, so retrofitted robots were made to survive the radiation environment inside Fukushima Daiichi [38]. Inspired by the challenges of Fukushima, the DARPA Robotics Challenge took place in 2015 [16], tasking teams with developing advanced robots to help respond to disaster situations. For those robots to take advantage of modern neural networks, those networks must be resilient against hardware failures such as memory degradation caused by radiation.

Another dangerous radiation environment is outer space. Earth’s magnetic field keeps most of the cosmic rays away from computers on the ground (aside from data centers as discussed before), however in orbit there is less protection, so more cosmic rays have a chance to interact with computer hardware. Single Event Effect (SEE) type errors are caused by high energy particles passing through computer chips. High energy particles can cause bit flips in memory locations, or high currents, which can permanently disable transistors [44]. Campbell et al. [5] gives an overview of memory errors over the life of the Combined Release and Radiation Effects Satellite (CRRES), which specifically studied SEE rates of various kinds

of memory. Data from the satellite showed that memory error rates can change a lot based on the space environment, and that some areas such as the Van Allen Radiation Belts are quite dangerous. Additionally, solar flares can drastically increase the rate of memory errors, during the CRRES mission a solar flare caused the memory error to increase to almost 36 memory bit flips every 15 minutes. If neural networks are to be used on the next generation of planetary rovers or to augment the growing commercial space imagery industry, they must be robust to memory errors caused by the harsh space environment.

Another cause of bit flips in memory can be due to malicious actors exploiting the physical properties of memory used in computer systems. So called Rowhammer attacks can flip bits in memory by repeatedly accessing neighboring memory locations and generating a large amount of electromagnetic noise [24]. These hardware bugs are being used by researchers to attack computer systems. A team from Google showed how Rowhammer attacks could be used to escape software sandboxes and escalate kernel privileges [42]. Another group of researchers used the Rowhammer bug to flip bits in RSA encryption keys and force a victim computer to download and install arbitrary software packages [39]. These kinds of attacks could be used to flip bits of neural network parameters and change the networks performance.

The causes of memory errors described in the preceding paragraphs are not often considered in neural network research, although there is some prior work covered in Chapter 2. Most research is focused on the capabilities of the networks or achieving better performance at computer vision contests. That kind of research has proved the utility of neural networks for a wide variety of tasks. But now that neural networks are subjected to real world conditions such as bit flips in memory caused by malicious code, nuclear radiation, or cosmic rays, research should include studying how neural networks behave in the field. This thesis contributes to that research focus by presenting the results of controlled bit flips inside neural

networks as though memory errors had occurred. The main goals of the experiments in this thesis are first, to determine how robust neural networks are to memory errors and second, if certain neural network architectures are more robust than others. The rest of the thesis is organized as follows:

Chapter 2 provides relevant background for the research shown in the thesis. An overview of neural networks is provided, including specifics about the network architectures tested throughout the thesis. Additionally, the various datasets used in this work are described in detail. Last, a literature review of related neural network robustness work is covered and differentiating aspects of this thesis are highlighted.

Chapter 3 covers the first set of experiments exploring how bit flips affect neural network performance. These experiments primarily focus on comparing the performance of two different kinds of neural networks: Multi-Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs). Additionally, the hyper-parameters (such as depth or kernel size) of the MLPs and CNNs were adjusted to see how they affected the robustness of the networks. This chapter showed that both MLPs and CNNs could resist a small amount of memory errors, but that MLPs are more robust. For the MLPs, increasing the depth of the network increases robustness, but a similar relationship does not exist for the CNNs. For the CNNs, robustness can be increased by using larger kernel sizes. An interesting takeaway from these experiments is the bimodal output of the network accuracies while experiencing bit flips. In general the bit flips either barely changed the accuracy of the network or the accuracy dropped all the way down to the level of random guessing.

Chapter 4 covers an additional round of experiments designed with lessons learned from the experiments covered in Chapter 3. First, this chapter focuses exclusively on CNNs. Although earlier experiments showed MLPs to be more robust, CNNs have significantly better performance on computer vision tasks. On some datasets, CNNs perform with similar accu-

racy to humans. The CNNs tested in Chapter 3 did not have state-of-the-art performance and were relatively small compared to modern CNNs. In this chapter, three modern CNN architectures are tested for their robustness against memory errors. The three tested architectures are VGG16, ResNet50, and InceptionV3. Each of these CNNs had, at one time, record setting classification accuracy on the ImageNet dataset and have significantly different architectures. The tests showed that ResNet50 and InceptionV3 were both more robust than VGG16, and various reasons for why they are more robust are discussed.

In Chapter 5, real world memory error rates are calculated to compare against the robustness of the tested networks. Various locations with high radiation such as near Fukushima and several different orbits are used as reference locations. An expected number of bit flips at each location is calculated and compared against the number of bit flips that the tested CNNs are able to withstand.

Chapter 6 finishes with overall takeaways and future work. The next steps for research are outlined alongside shortcomings of the previous experiments.

1.1 Contributions

This thesis contributes to the large body of neural network research in several novel ways. First, the results of Chapter 3 show that simple MLP networks seem to be more robust than simple CNNs. The MLPs tested did a good job of maintaining their trained performance under an increasing number of memory errors, and deeper MLPs resisted the memory errors better than the shallow MLPs. The deeper networks have a much higher number of parameters so that could be why the deep MLPs are more robust than the shallow MLPs. The relationship between depth and robustness does not exist for small CNNs. Instead, there is a strong relation between the size of the kernels and the robustness of the CNNs. CNNs

using larger kernels are more robust than CNNs using smaller kernels. However, none of the CNNs maintained their trained performance under weight errors as well as the MLPs. One last contribution from Chapter 3 is the fact that the failure trend of the CNNs is bimodal. Under bit flips, CNNs either maintain their trained performance, or have an accuracy on par with random guessing. These results were written into a paper, “The Effect of Weight Errors on Neural Networks” which was accepted to the IEEE Computing and Communications Workshop and Conference and subsequently published.

Chapter 4 contributed more to understanding the robustness of CNNs. Experiments show that VGG16 is less robust than ResNet50 or InceptionV3. ResNet50 and InceptionV3 could withstand an order of magnitude higher percentage of their parameters in error than VGG16 before falling to the same level of performance as VGG16. This shows that there is not a direct relationship between the number of parameters in a CNN and its robustness, because VGG16 has the an order of magnitude more parameters than both ResNet50 and VGG16. Possible reasons for the increased robustness of ResNet50 and InceptionV3 are the use of Batch Normalization or the use of shortcut connections. A final contribution from Chapter 4 is that the bimodal failure mode found in the small CNNs from Chapter 3 continues in large CNNs. VGG16, ResNet50, and InceptionV3 all had bimodal failure trends. These results have been accepted to the IEEE High Performance Extreme Computing conference and will be published in a paper titled “The Robustness of Modern Deep Learning Architectures Against Single Event Upset Errors”.

Chapter 5 contributes to the field by showing how well the robustness of the neural networks examined in this thesis stand up to estimated real-world memory error rates. The estimated error rates are all for space environments, including Low Earth Orbit (LEO) which is seeing increased investment by commercial companies. This chapter shows that ResNet50, InceptionV3, and large MLPs have enough robustness to survive in LEO on commercial SRAM

using only software error detection and correction. This will allow satellite manufacturers to save money by not purchasing radiation hardened parts. Additionally, this chapter gives a brief overview of an application that could be run onboard a satellite in LEO, and would make use of large neural networks. These results have been accepted to the IEEE National Aerospace and Electronics Conference and will be published in a paper titled “Onboard Image Processing for Small Satellites”.

Chapter 2

Neural Network Basics and Previous Work

This chapter provides basic background material before moving on to the detailed experiments. The first sections cover neural networks and their architecture. Background on how neural network are trained is not given, because that knowledge is not required to understand the rest of this thesis. This chapter also provides details on the datasets used for training the networks in future chapters. A brief discussion of hardware technologies used for computer memory is included to help with the last chapter involving real world memory error rates. Finally, there is a discussion of related research and how this thesis is novel compared to the previous work.

2.1 Neural Network Background

2.1.1 Multilayer Perceptron Description

A Multilayer Perceptron (MLP) [41] is a special kind of feedforward neural network made up of an input layer, one or more hidden layers, and an output layer. Each layer is made of individual neurons as shown in Figure 2.1. A neuron has several inputs each with a unique weight. The weights are multiplied by their input before being summed together and passing

through an activation function.

Figure 2.2 shows how neurons are connected in an MLP to form a neural network. The first layer consists of input neurons. Input neurons represent the data being sent into the network. The next layer is the hidden layer where the weights are applied and the activation function is executed. The output of the hidden layer can go to the input of another hidden layer, creating a deeper network. At the end of the hidden layers is the output layer where the result of the neural network is exposed.

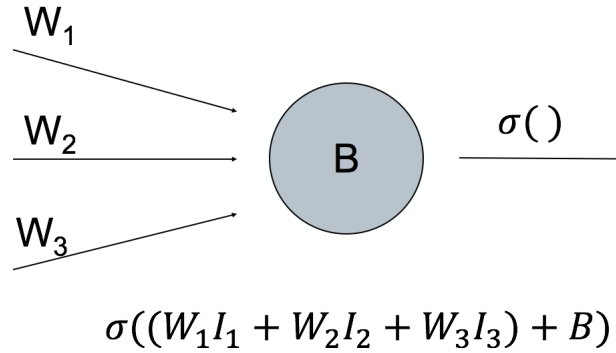


Figure 2.1: A single neuron in a neural network along with the equation describing its behavior. W_n represent the weights on each input, while B represents the Bias of the whole neuron. Sigma represents an activation function.

2.1.2 Convolutional Neural Network Description

Convolutional neural networks (CNNs) are a more modern architecture than MLPs. CNNs perform much better than MLPs at computer vision tasks such as classification of objects in an image. CNNs use similar neurons as MLPs but arranged differently as shown in Figure 2.3. The neurons in a CNN are arranged into several small groups called kernels that convolve across the input to form the next layer of data. Each kernel can be said to learn a specific feature of the image classes the network trains on. The output from a set of kernels can then be sent to another set of kernels, forming a stack of convolutional layers. However,

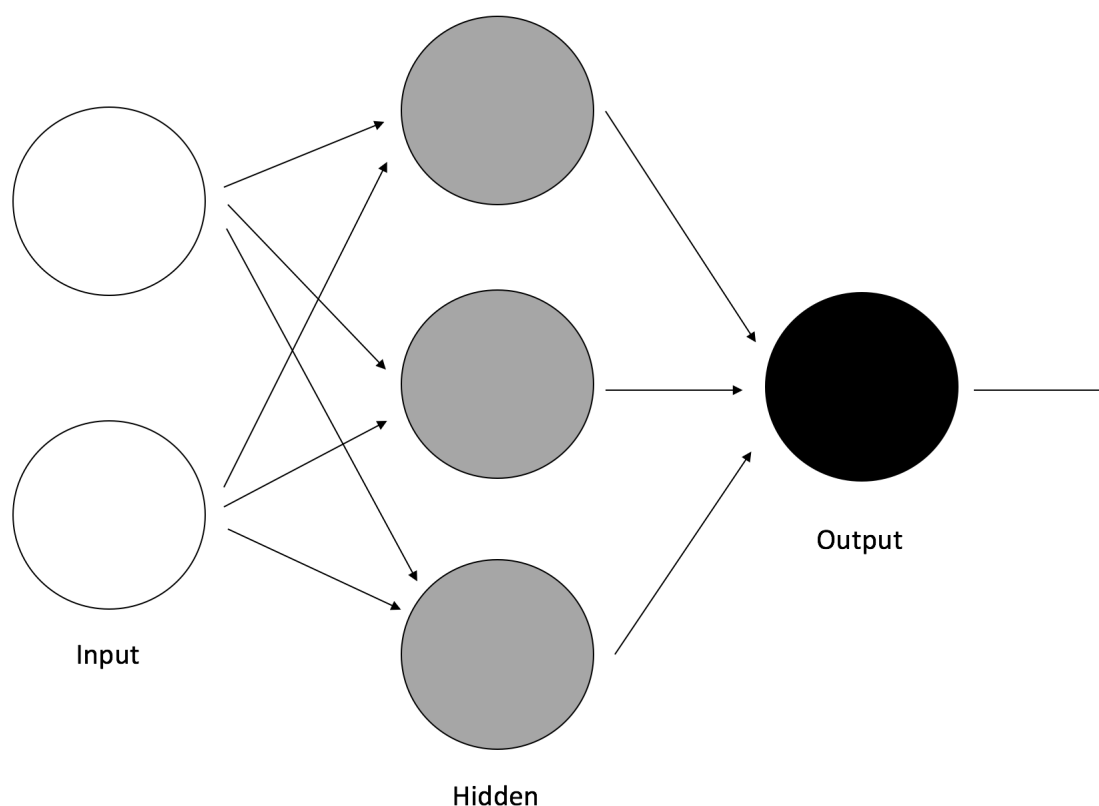


Figure 2.2: Illustration of how the neurons are connected in a small MLP. Notice that every neuron in one layer is connected to every other neuron in the next layer.

CNNs are not usually built by just stacking convolutional layers on top of each other. A special kind of layer called a pooling layer is generally inserted after a few convolutional layers. Figure 2.4 shows how a pooling layer takes several inputs and outputs a single value representing the highest value of the input. This operation removes unnecessary data from the network. For example, the kernels before the pooling layer may have detected that the picture to be classified has pointy ears. The pooling layer will remove information about the exact orientation of the ears since that is not important to the final classification answer. After convolutional layers and pooling layers, CNNs end with a few fully connected layers that match the architecture of a MLP. This is done to convert the information of which features were detected by the kernels to the name of the object in the picture.

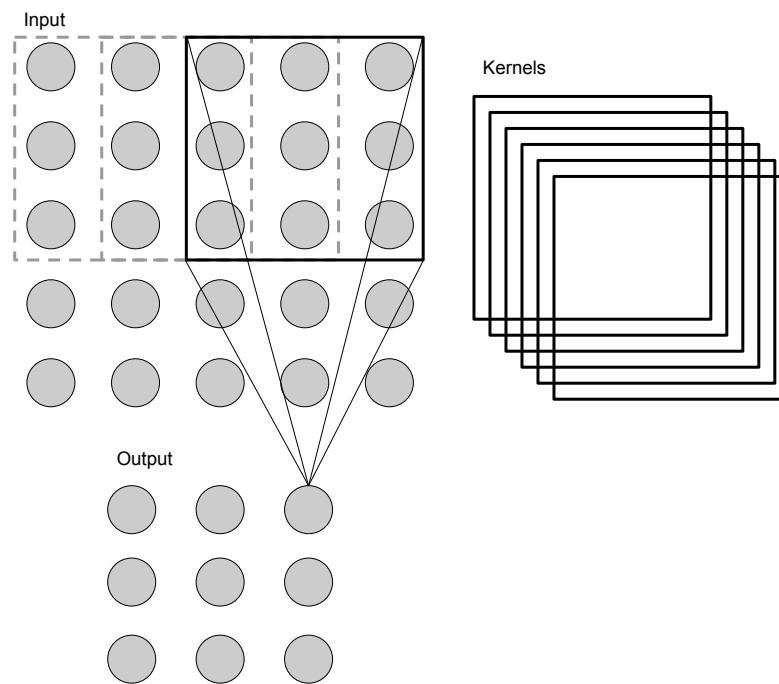


Figure 2.3: In convolutional neural networks the neurons are arranged into kernels, which then convolve or slide across the input to generate the output data. The size and number of kernels can be adjusted.

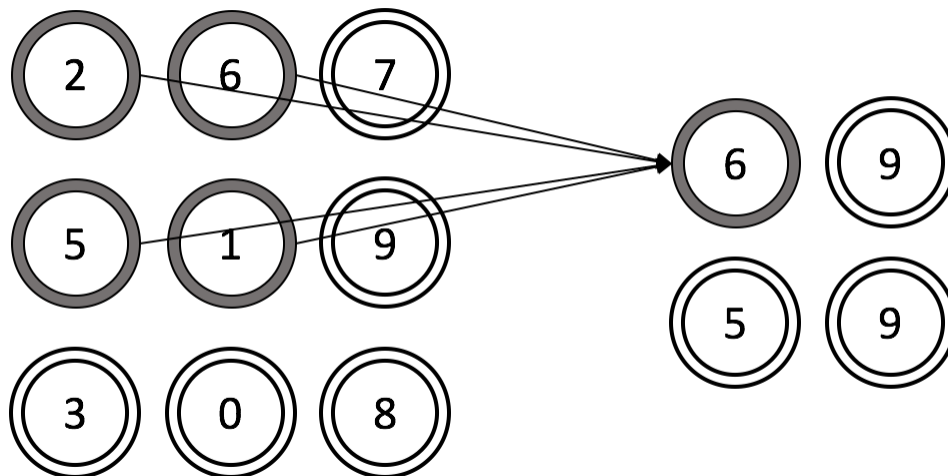


Figure 2.4: Example of how a max pooling layer filters the data by only allowing the max value in its pooling window through. In this case, the window is 2x2 and overlaps by one.

2.2 Datasets

2.2.1 MNIST

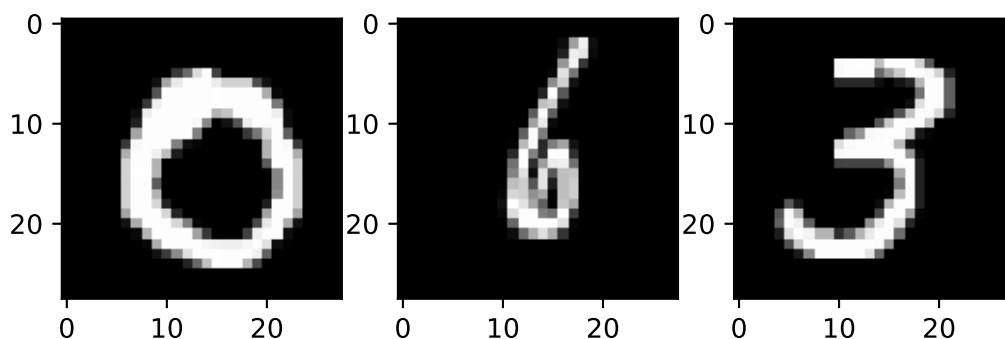


Figure 2.5: Three examples of images from the MNIST dataset. From left to right are the numeral zero, numeral six, and numeral three. The numeral six illustrates that some examples can be difficult to classify.

The MNIST (Modified National Institute of Standards and Technology) dataset is a database of handwritten digits primarily used for computer vision research [33]. The dataset includes 70,000 images, generally split into 60,000 images for training and 10,000 images for testing. The images contain just greyscale values and are 28 pixels wide by 28 pixels tall. Each image gives one example of a handwritten digit written by either an employee of the American Census Bureau or high school students in America. The digits span the range from zero to nine and are centered in the 28x28 window based on calculating the center of mass of the image.

2.2.2 CIFAR-10

The CIFAR-10 (Canadian Institute for Advanced Research) dataset is a collection of small images representing 10 different classes primarily used for computer vision research [27]. The images in CIFAR-10 represent ten different classes: airplane, automobile, bird, cat, deer, dog,

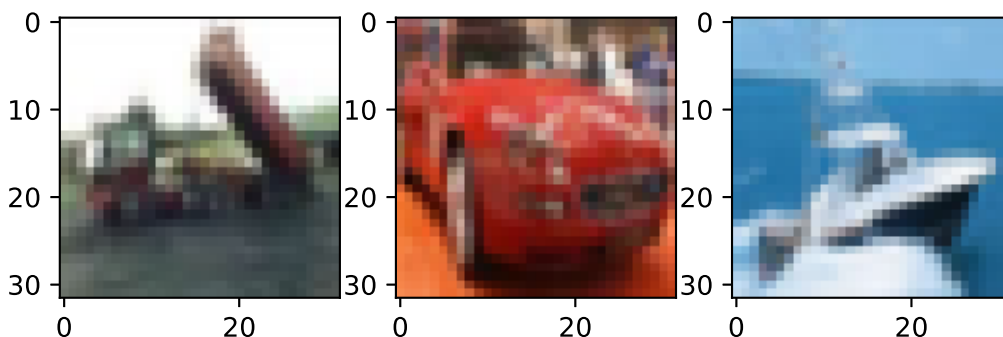


Figure 2.6: Three examples of images from the Cifar10 dataset. From left to right are an example of a truck, automobile, and ship. The low resolution of the images can make classifying these images challenging.

frog, horse, ship, and truck. The classes are all mutually exclusive and ambiguous objects such as pickup trucks which could span two classes are not present. Each class in the dataset has 6,000 example images, so the whole dataset has 60,000 images. Generally the dataset is split into 50,000 training images and 10,000 test images. The images are RGB color images, each 32 pixels wide and 32 pixels tall. The images come from a collection of 80 million tiny images collected by researchers at MIT.

2.2.3 ImageNet



Figure 2.7: Three examples of images from the ImageNet dataset. From left to right are an example of a Shetland sheepdog, crane, and safe. The classes in ImageNet are very specific and the images are not all the same size.

The ImageNet dataset is a very large database with over 14 million images across 21,000 categories. The images come in a variety of sizes, but generally are about 400 pixels tall by 400 pixels wide and all the images come in RGB colors. The images are not equally spread across all the categories, but some of the broader categories have several hundred example images. The ImageNet dataset is used for the ImageNet Large Scale Visual Recognition Competition (ILSVRC) which is hosted every year to test the performance of different computer vision algorithms. In this thesis, only the validation dataset from the 2012 ILSVRC is used which contains 50,000 images across 1,000 categories.

2.3 Hardware

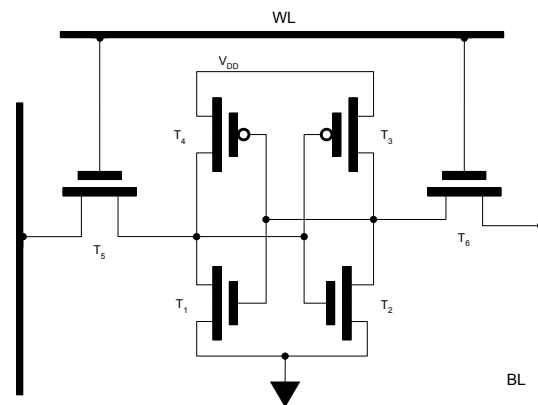
2.3.1 Memory Hierarchy

The computer memory that stores the neural network parameters in these experiments exists as a hierarchy, with small amounts of fast memory nearest to the computer processor and large amounts of slower memory stored farther away. The closest memory to a computer processor are registers. The x86_64 instruction set architecture has 16 general purpose registers, and many more pre-defined registers that hold values the processor is currently using. Instructions directly call on values stored in these registers to perform computations. If a value does not exist in a register, then it must be fetched from the cache. The cache is the next fastest kind of memory. Generally, data from the cache can be moved into a register in approximately 10ns. This speed is possible because the cache is located on the same silicon chip as the rest of the processing core. In multi-core systems, each core will have its own cache, and sometimes there is a separate shared cache that all the cores can use. The size of the shared cache is generally around 10 MBs, while the cache for each individual core

is around 500 KBs. After the cache, the next level of the memory hierarchy is the main memory, also called Random Access Memory or RAM. RAM is generally several GBs large, over 64GB in some desktop computers. The time to access data from RAM is about twice as slow as getting data from the cache, because RAM exists on separate silicon chips than the CPU, and the data must travel over long wire buses. Finally, if data is not in the cache or RAM, then it lives in the hard drive, the slowest memory. Hard drives have the largest data capacity at over a TB, but accessing data from the hard drive is usually several orders of magnitude slower than accessing data in the RAM. With the invention of solid state hard drives, latency times are down to a single order of magnitude greater. This means that in general, hard drives are used for long term data storage. For example, if a neural network is not being used, then all its trainable parameters like weights and biases may be stored on the hard drive. Once the neural network is being executed, the data moves to fill as much of the cache as possible while the rest sits in the RAM for easy access. Since the neural network parameters will be in the cache or RAM during execution, this chapter focuses on error rates of those memory devices as opposed to the error rates of solid state hard drives.

2.3.2 Memory Technology

Based on the needs of the different levels of the memory hierarchy, different technologies are used to build each level. Registers and cache have speed as a very high priority, while RAM and hard drives focus more on density and capacity. In the case of cache and RAM, there are two dominant technologies for implementing each one: Static Random Access Memory (SRAM) and Dynamic Random Access Memory (DRAM).



Static Random Access Memory

Static Random Access Memory (SRAM) is the preferred memory technology for the cache of a computer. SRAM is used for the cache because each cell can be accessed with low latency. SRAM cells are made up of two inverters wired together along with two access transistors, for a total of six transistors per SRAM cell (see Figure 2.8). Each SRAM cell is volatile, meaning that without a constant source of power the data stored in the cell will be lost. However, as long as there is power to the cell, the data will not need to be refreshed, and both reading and writing are non-destructive.

Dynamic Random Access Memory (DRAM) is the preferred memory technology for the main memory or RAM of a computer. DRAM is used for main memory because DRAM is much more dense than SRAM, so it is possible to have several gigabytes of DRAM as opposed to a few megabytes of SRAM. This density is achieved by using a different cell design that only

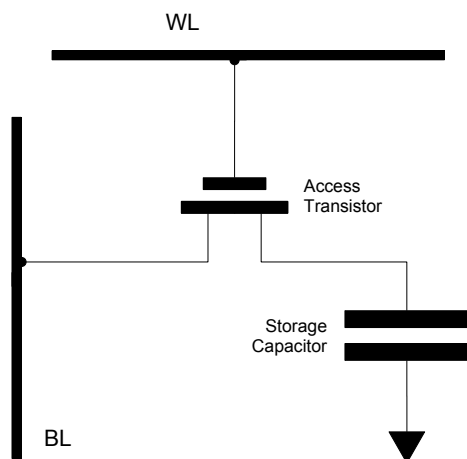


Figure 2.9: This figure shows a diagram for a standard DRAM cell. Zero voltage stored in the capacitor stands for a logical 0, while high voltage stands for a logical 1. The transistor is used to access the data in the capacitor.

requires a single transistor and a single capacitor (see Figure 2.9). Unfortunately, this design has several disadvantages when compared to SRAM. First, the capacitor is leaky and must be recharged to maintain the high “1” voltage. Additionally reading from the capacitor is destructive and the charge must be restored. These disadvantages lead to the slower access time for DRAM as opposed to SRAM, which is why SRAM is used for cache. But the increased density of DRAM makes it a good technology choice for main memory.

2.4 Related Work

The theoretical underpinnings of machine learning are several decades old, and a seminal paper on weight errors in neural networks was published in 1990 [51]. That paper applied weight perturbation ratios and input errors to individual adaline units (similar to the neurons discussed above) and then calculated the probability that the output of the adaline would cross its decision boundary. This method was also used to test the performance of a

madaline network (similar to MLP). However the madaline network was not actually trained on a specific task. Rather, given a random initialization of weights, the paper tested the probability that a certain weight perturbation ratio would change the original output mapping. The results from this thesis differ in several ways from [51]. First, weight perturbation ratios and bit flips are different kinds of errors and will produce different altered states of the neural network. Bit flips will have a much higher range of error values, especially when using floating point numbers. Second, in this thesis, fully trained networks for a specific task are tested. Additionally, madaline networks are very old and it is in open question as to if the results on madaline networks will apply to new neural network architectures such as CNNs and their variations.

Choi and Choi [7] extends the work done by Stevenson et al. [51]. In [7], the effect of weight perturbations on MLPs with different initial weights is explored. Additionally, their analysis switched between two kinds of weight perturbations, additive or multiplicative. The paper is very interesting in that it shows that certain weight initializations are more robust than others, but a serious limitation of the paper is that it focuses primarily on single-output MLPs as compared to the more complex and capable networks explored in this thesis. Additionally, additive or multiplicative weight perturbations are not quite the same as bit flips on floating point numbers due to the much wider range of change possible due to the bit flips. By changing the weights more, there could be a larger affect on the execution on the network.

An interesting set of experiments by a group of French researchers also explored the robustness of older neural networks in the late 1990s [58, 59, 60, 61]. These experiments focused on developing a neural network that could survive in the space environment. They envisioned several applications for space-based neural networks such as classifying land images based on their texture, i.e. industrial land, residential land, ocean, etc. or detecting whistlers in the outer magnetosphere. Unlike the previous papers, these papers were testing the effect

of bit flips on neural networks. Their experiments showed that their network for classifying land use could withstand 2% of its weights to be in error before its classification accuracy went below 50%. However, the network tested by this group of researchers is still an old MLP, and not a modern CNN. Additionally they do not compare different neural networks to determine which aspects of a neural network cause better or worse performance when subjected to bit flips.

A modern paper on neural network weight error robustness [29] focuses on one specific type of CNN, LeNet5, and trains the network on the MNIST dataset. Kwon et al. [29] does not directly simulate bit flips, but instead applies artificial Gaussian noise to the weights of LeNet5, which restricts how much the weights could be changed compared to bit flips. In some ways, this thesis replicates the results of that paper, while also going further. One way this thesis goes further is by using a different, more complicated dataset for the CNNs. The more complicated dataset mirrors the real world better and will make it harder for the CNNs to maintain their performance under weight errors. This thesis also expands the networks being tested. Instead of just testing one type of CNN, many different types of CNN architectures are explored to determine which variable in CNN design most affects the robustness of the network. Additionally, this thesis tests both MLPs and CNNs so the two networks can be compared directly.

Aside from just research on weight errors, there are researchers focusing on other aspects of neural network robustness. Sequin and Clay [43] and Mhamdi and Guerraoui [37] are two papers that explore the robustness of neural networks to the loss of an entire neuron after training. This is an interesting contrast to using Dropout during training, which purposefully turns off neurons to prevent the model from memorizing the dataset and not generalizing across the entire problem set. [43] actually concluded that disabling neurons during training just like Dropout helped make networks more robust to the loss of neurons during inference.

They also found that if a neural network permanently lost a neuron, it could be retrained back up to similar performance. In [37], they removed neurons after training and found a relationship between the robustness of the network and the steepness of the activation function. The steeper the activation function, the less robust the network. Both of these papers focused on MLP networks as opposed to CNNs, although [37] claimed that their theoretical work could apply to CNNs.

An exciting new research trend investigating the robustness of neural networks is the discovery of adversarial examples [55] [13]. Adversarial examples are inputs to a neural network that have been specifically designed to have the wrong output. For example, the correct amount of random pixels added to an image of a panda can make a neural network classify the image as a gibbon. The amount of noise required to fool the network is so small that it can be imperceptible to humans. These kinds of errors affect modern deep learning architectures such as those examined in this thesis, and have started raising interesting questions about how trustworthy neural networks are and challenge assumptions about what exactly neural networks are learning. This work has been followed up by other researchers looking to exploit neural networks with techniques such as adversarial patches that, when added to a scene, can change the classification output of the network [4]. Adversarial examples are fascinating, especially since they work against modern deep learning architectures, but they focus on errors exclusively on the input to the networks as compared to errors within the hidden layers which is the focus of this thesis.

Chapter 3

Comparing MLP and CNN

Robustness

This chapter presents the initial memory error experiments. The overall goal of the thesis is to determine if there are certain architectural traits that give some neural networks more resilience against bit flips than other networks. However, for a fair comparison, the two neural networks should both be able to accomplish a similar task. Therefore, this chapter compares the robustness of MLPs and CNNs. Both types of networks can be used for computer vision tasks, while also having significant architectural differences. In addition to comparing the two different kinds of neural networks, various parameters of each network type are adjusted to determine what aspects of MLPs or CNNs have the greatest contribution to their robustness.

3.1 Tested Architectures

Since this thesis only focuses on the result of weight errors, the networks tested in this thesis are based off other successful architectures. The networks developed in [9] serve as a model for the MLP networks in this thesis. That paper developed several MLP networks that focus on classification of handwritten digits from the MNIST dataset, a widely used benchmark for testing neural networks [33]. The networks in [9] had record-setting performance at the

Table 3.1: Overview of MLP Architectures

	OneLayer	ThreeLayers	FiveLayers
Input	784	784	784
Hidden	100	1000	2000
Hidden	-	500	1500
Hidden	-	100	1000
Hidden	-	-	500
Hidden	-	-	100
Output	10	10	10

time of publication, and even now have barely been surpassed. A difference in implementation between this thesis and [9] is in how the networks were trained. The original paper deforms the MNIST images to increase the size of the training data. This thesis simply trains on default training data. This means that the maximum accuracy in this thesis is lower than the original paper by about one to two percent.

Table 3.1 shows the number of neurons per layer that were used during the experiments. The networks all have 784 input neurons corresponding to the pixels in MNIST images, and 10 output neurons for signaling which digit has been detected.

In addition to the MLPs described above, this chapter also tests some small, custom CNNs to focus on how specific CNN parameters affect robustness. The primary parameters tested by the custom networks is kernel size and depth. The two kernel sizes explored were 3x3 kernels and 7x7 kernels. Each size of kernel was tested in a network with two convolutional layers, and a network with four convolutional layers. The four layer networks had a larger number of kernels in layers 3 and 4. All of these small, custom networks were trained with Dropout on the CIFAR10 dataset. The CIFAR10 dataset has 60,000 color images of size 32 x 32. There are 10 object classes in the dataset and each class has 6,000 images. This dataset has more complicated images than the MNIST dataset and is harder to get high accuracies on. Table 3.2 gives a detailed view of the custom architectures tested in this chapter.

Table 3.2: Overview of CNN Architectures

TwoLayersSmall		TwoLayersLarge	FourLayersSmall	FourLayersLarge
Input 32x32x3				
Hidden	Conv 3, 64	Conv 7, 64	Conv 3, 64	Conv 7, 64
Hidden	Conv 3, 64	Conv 7, 64	Conv 3, 64	Conv 7, 64
Maxpool 2x2				
Hidden	-	-	Conv 3, 128	Conv 7, 128
Hidden	-	-	Conv 3, 128	Conv 7, 128
—			Maxpool 2x2	
Dense 128				
Output 10				

For both the MLPs and CNNs, a Rectified Linear Unit (RELU) activation function is used between layers. This is a difference between our experiments and those done several decades ago. RELUs are a popular activation function in modern neural networks because they avoid the exploding gradient problem, so they train much faster than sigmoid activation functions [12]. All the output layers use a softmax activation function to provide the final classification answer.

3.2 Results

3.2.1 Methodology

The Keras [8] Python library was used with Tensorflow [3] as the backend to design and run the neural networks. GPUs on several machines were utilized to accelerate training and testing. The errors were introduced to the neural networks after training was complete and all the weights had been set. The errors were simulated by selecting a weight at random from the trained networks, and then flipping a randomly chosen bit in that weight. The network's accuracy was then re-evaluated with the new weight. Each weight is represented by 32-bit

floating point numbers. The number of weights in error was increased to determine how robust the networks were to weight errors and to classify the behavior of the networks under increasing errors. For each neural network, the weights in error were increased from 10 to 100, and the classification accuracy of each network was recorded. 200 trials were recorded for each number of weights in error. The weights in error were randomly selected between each trial, and the bit to be flipped was randomly selected for each weight.

3.2.2 MLP Results

Each MLP was tested as described above. The biggest main differences between the MLP models is their depth and the number of neurons in each layer. Therefore, the following experiments should determine if there is a correlation between the depth of an MLP and its robustness to weight errors.

Figure 3.1 shows the result of flipping bits in the OneLayer MLP network. That network trained to an accuracy of 97.97% at classifying the handwritten digits in MNIST. Looking at the figure, the histogram at the back shows that with only 10 weights in error the classification accuracy is not affected very much. The accuracy stays at the trained accuracy for most of the 200 trials, although there are a few runs of the network with accuracies below 50 percent. As the number of weights in error decreases, the number of trials near the trained accuracy also decrease. By the time 100 weights are in error, less than 40 of the 200 trials were at the trained accuracy. An interesting trend does appear as the number of weights in error increases. The histograms begin showing a bimodal shape, with a cluster of trails at very low accuracies and a cluster near the trained accuracy. As more weight errors occur, the network will not have a smooth drop off in performance. Instead, some executions of the network behave perfectly, while others look like random guessing. This illustrates that each

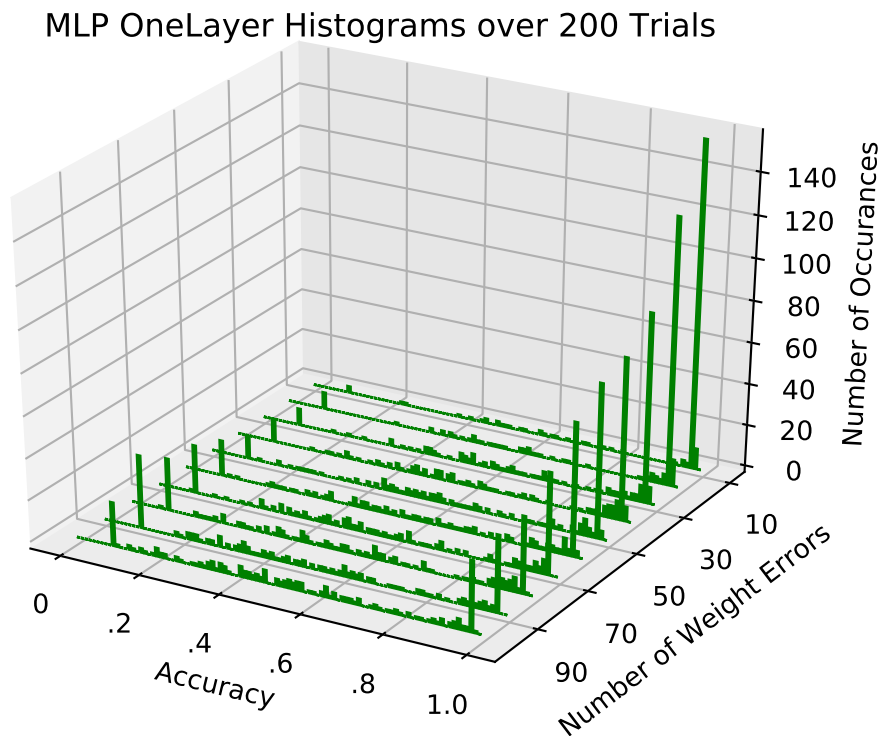


Figure 3.1: The performance of the OneLayer MLP network with weight errors. The bar height represents the number of times a certain accuracy occurs over the 200 trials for each number of weights in error.

weight in the network affects the response of all the different output nodes. The changing of a few weights doesn't just break the output node that indicates if the digit is a 9, but instead breaks all of the output nodes.

Figure 3.2 shows the result of flipping bits in the ThreeLayer MLP network. This network trained to an accuracy of 98.14% at classifying the handwritten digits in MNIST. Similar to the OneLayer network, this network is relatively unaffected by only 10 weights in error. In fact, the ThreeLayer network is more robust because with 10 weight errors it has over 175 trials at the trained accuracy, compared to only 160 trials for the OneLayer network. As

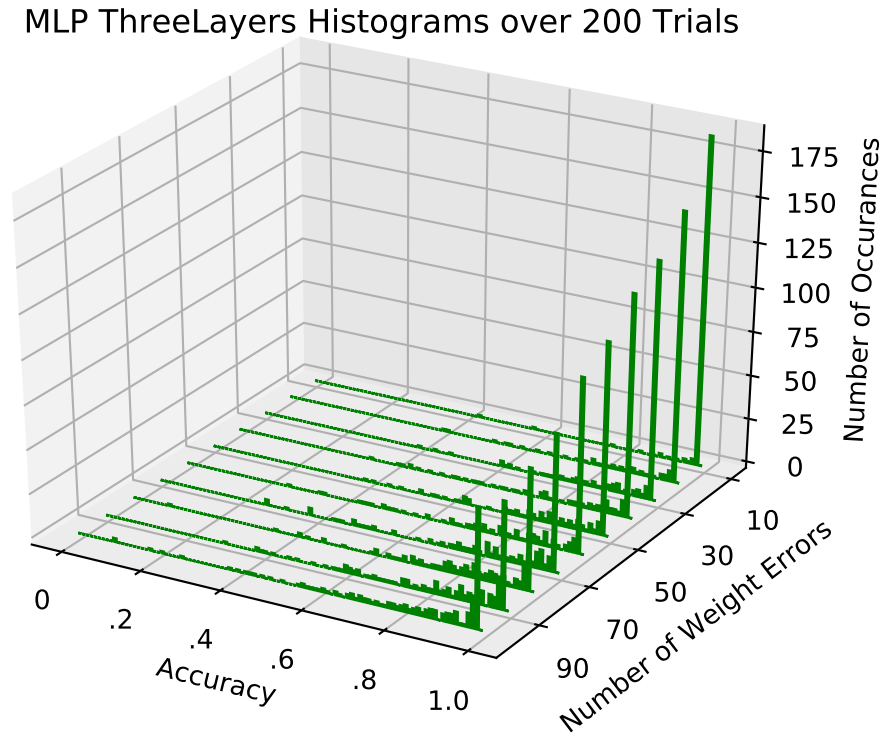


Figure 3.2: The performance of the ThreeLayer MLP network with weight errors. The bar height represents the number of times a certain accuracy occurs over the 200 trials for each number of weights in error.

the number of weight errors increase, the number of trials at the trained accuracy decreases. However the final run with 100 weight errors has over 50 trials at the trained accuracy, again showing that the ThreeLayer network is more robust than the OneLayer network. Additionally the ThreeLayer network is not showing the same bimodal behavior as the OneLayer network. Most of the runs seem to be over 80% classification accuracy with no cluster near the 10% accuracy point. The lack of bimodal behavior may be due to the larger size of the ThreeLayer network. Even though each weight in the network is still attached to each output neuron, the larger size of the ThreeLayer network means that each weight has

a smaller effect on the outputs. In the OneLayer network there are 78,500 weights between the input and a single output node, while in ThreeLayer network there are 1,334,100 weights between the input and a single output node. So the only 100 weight errors have a much smaller effect on each individual output node in the ThreeLayer network than they do in the OneLayer network.

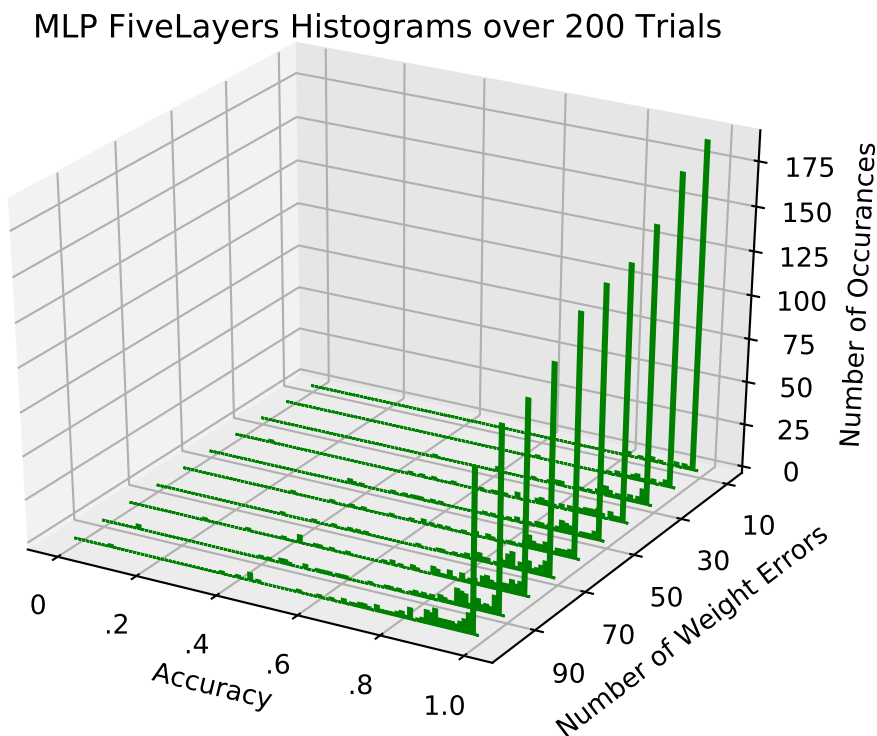


Figure 3.3: The performance of the FiveLayer MLP network with weight errors. The bar height represents the number of times a certain accuracy occurs over the 200 trials for each number of weights in error.

Figure 3.3 shows the result of flipping bits in the FiveLayer MLP model. This network trained to an accuracy of 98.5% at classifying the handwritten digits in MNIST. This network starts performing similarly to the ThreeLayer network with 10 weight errors. Both networks are

above 175 trials at their trained accuracies so they are more robust than the OneLayer network. As the weight errors increase, the FiveLayer network seems to be even more robust than the ThreeLayer network. The number of trials at the trained accuracy with 100 weight errors is at about 75 with the FiveLayer network, compared to the ThreeLayer network at about 50 trials. The FiveLayer network also does not have the bimodal behavior seen in the OneLayer network at the tested number of weight errors. This is because, just like for the ThreeLayer network, there are more weights between the input and each individual output node in the FiveLayer network than the OneLayer network.

From these three experiments, a simple correlation can be seen between the size of an MLP and its robustness to weight errors. As the size and depth of an MLP network increases, it is better at maintaining its trained classification accuracy under bit flips in its weights. This is believed to be due to the fact that deeper MLPs have more weights. When the OneLayer network has 100 weights in error, this is a much larger proportion of the total weights in the network than when the FiveLayer network has 100 weights in error.

3.2.3 CNN Results

The first experiments analyzed are the tests on the small custom CNNs explained by Table 3.2. The primary purpose of these experiments is to determine if there is a correlation between the size of the kernels in each convolutional layer and the network’s robustness to weight errors.

Figure 3.4 shows the result of flipping bits in the TwoLayersSmall CNN model. This network trained to an accuracy of 68.29% at classifying objects in CIFAR10. The figure shows that the TwoLayersSmall network is relatively robust to weight errors. With only 10 weights in error, over 140 of the trials were still at the trained accuracy. As the number of weights in

CNN TwoLayersSmall Histograms over 200 Trials

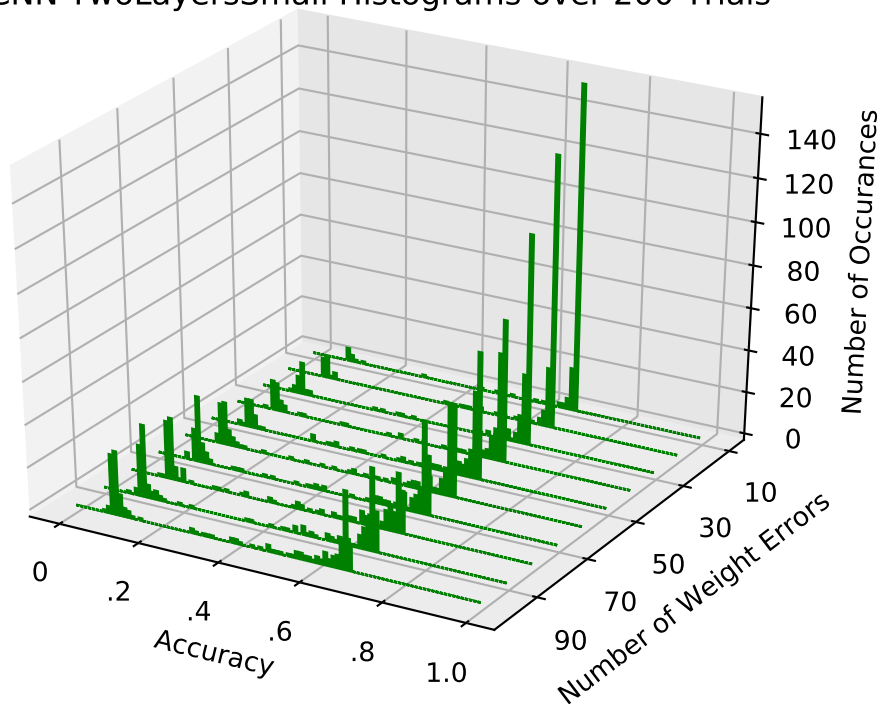


Figure 3.4: The performance of the TwoLayersSmall network with weight errors. The bar height represents the number of times a certain accuracy occurs over the 200 trials for each number of weights in error.

error increases, the number of runs at the trained accuracy decreases. Overall, the results here are similar to the results for the MLP OneLayer network (Fig. 3.1). Both figures show bimodal behavior as the number of weight errors increases and both figures show similar accuracy, relative to the trained accuracy, at the minimum and maximum number of weights in error. One difference is that the CNN histograms are slightly more spread out. The spread is most clear at the lower accuracies, where for CNN TwoLayersSmall there are two significant peaks near 10% accuracy, while for MLP OneLayer there is only one peak right at 10%. The bimodal behavior reveals how important each individual weight is to each output

node. This is a little odd because CNNs are not fully connected. However, the response of each kernel is important to each output node so when there are flipped bits causing a kernel to break, each output node is effected.

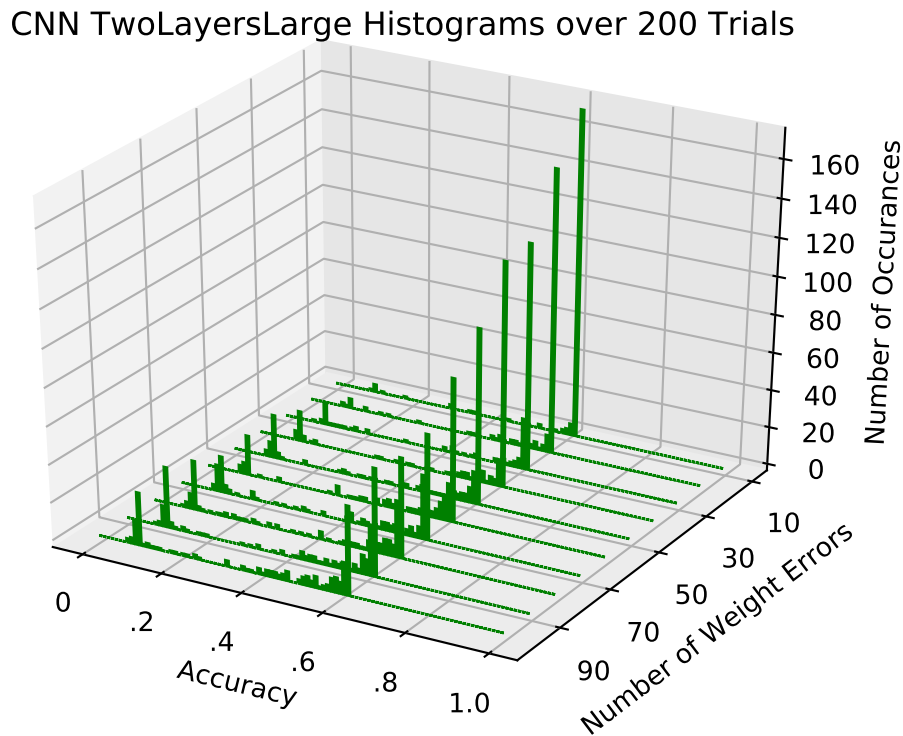


Figure 3.5: The performance of the TwoLayersLarge network with weight errors. The bar height represents the number of times a certain accuracy occurs over the 200 trials for each number of weights in error.

Figure 3.5 shows the result of flipping bits in the TwoLayersLarge CNN model. This network trained to an accuracy of 62.44% at classifying objects in CIFAR10. The figure shows that TwoLayersLarge is also fairly robust to weight errors. When compared to TwoLayersSmall (Fig. 3.4), TwoLayersLarge is actually more robust. With only ten weights in error, TwoLayersLarge has over 160 trials at the trained accuracy while TwoLayersSmall is only

over 140. Similarly, once 100 weights are in error the TwoLayersLarge network has more runs at the trained accuracy than the TwoLayersSmall network. So in general the TwoLayersLarge network is more robust to weight errors than the TwoLayersSmall network. This is probably because TwoLayersLarge has more weights than TwoLayersSmall, similar to how the deeper MLPs were more robust than the shallow ones. Both networks however do show bimodal behavior. There is not a smooth decrease in classification accuracy as the number of weights in error increases. Instead there is simply a higher chance that the network will sharply drop to performance approximating random guessing.

CNN FourLayersSmall Histograms over 200 Trials

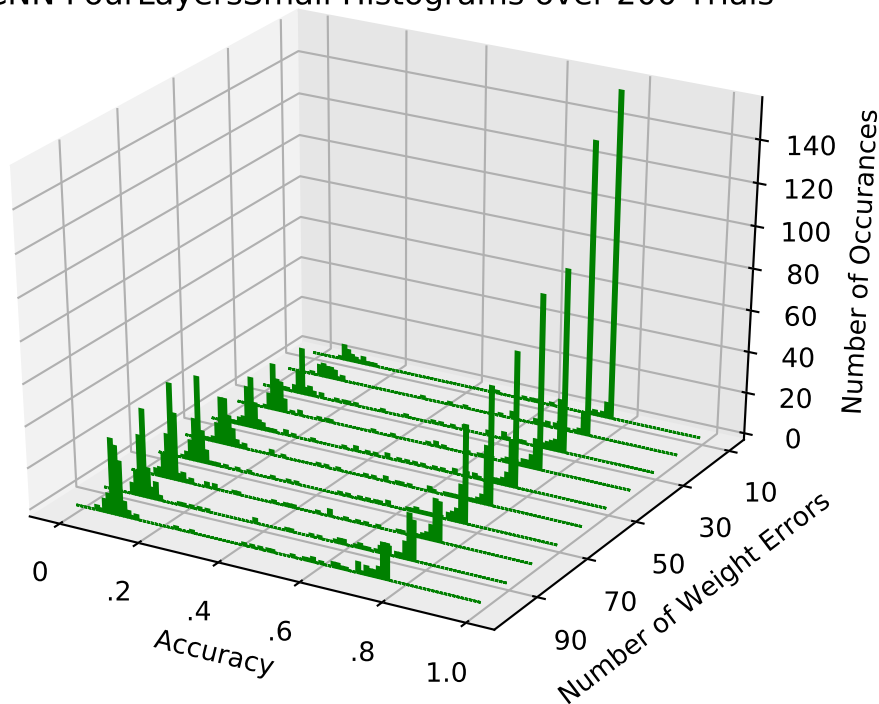


Figure 3.6: The performance of the FourLayersSmall network with weight errors. The bar height represents the number of times a certain accuracy occurs over the 200 trials for each number of weights in error.

Figure 3.6 shows the result of flipping bits in the FourLayersSmall CNN model. This network trained to an accuracy of 77.2% at classifying objects in CIFAR10. The results shown in this figure have some similarities to the results from testing TwoLayersSmall (Fig. 3.4). Both networks have over 140 runs at their trained accuracies when only 10 weights are in error. Additionally both networks have an increasing number of trials around random guessing accuracy as the number of weight errors increases. Finally, both networks show the bimodal behavior which is also shown by TwoLayersLarge (Fig. 3.5) and MLP OneLayer (Fig. 3.1). One difference between this network and TwoLayersSmall is their performance when 100 weights are in error. FourLayersSmall has less than 20 runs at its trained accuracy while TwoLayersSmall has over 30 runs. This breaks the trend from the MLPs where the deeper networks were more robust than the shallow networks. This may be due to the fact that CNNs are sparsely connected, meaning that each node is not connected to every preceding node. However, the nodes can interact indirectly if the network is deep enough. Since the FourLayersSmall network is deeper than the TwoLayersSmall network there are more indirect interactions between nodes and the input and nodes at the output, so a random distribution of flipped bits will affect more of the FourLayersSmall network. FourLayersSmall also has a wider distribution of accuracies when its performance degrades. Figure 3.6 shows three peaks reaching 20 runs at around 10% accuracy when 100 weights are in error compared to only two peaks in Figure 3.4 and only one peak in Figure 3.1.

Figure 3.7 shows the result of flipping bits in the FourLayersLarge CNN model. This network trained to an accuracy of 74.28% at classifying objects in CIFAR10. The results shown in this figure have some similarities to the results from testing TwoLayersLarge (Fig. 3.5). Both networks hit 160 runs at their trained accuracies when only 10 weights were in error but that number steadily decreases as the number of weights in error increases. Both networks also show the bimodal behavior seen in the TwoLayerSmall and TwoLayerLarge networks, as well

CNN FourLayersLarge Histograms over 200 Trials

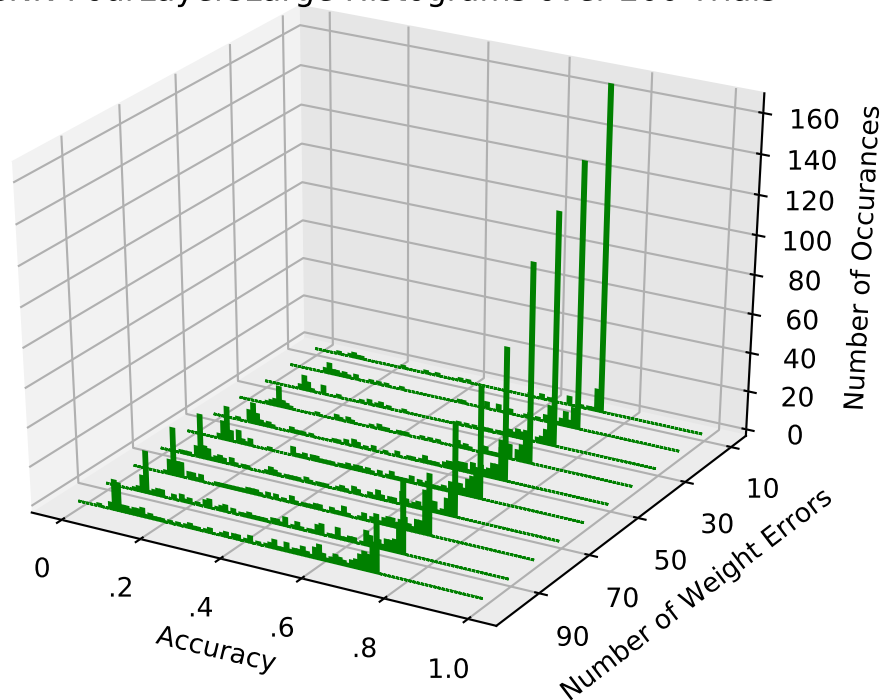


Figure 3.7: The performance of the FourLayersLarge network with weight errors. The bar height represents the number of times a certain accuracy occurs over the 200 trials for each number of weights in error.

as the OneLayer MLP network. However, FourLayersLarge has worse performance when 100 weights are in error than TwoLayersLarge. FourLayersLarge has about 30 runs at its trained accuracy with 100 weights in error while TwoLayersLarge has about 40 runs at its trained accuracy with the same number of weights in error.

The preceding four experiments show that there is a strong correlation between the size of the kernels in the network and their robustness to weight errors. For networks with equivalent depth, the network with the larger kernels retains higher classification accuracy under increasing weight errors. The networks with larger kernels at equal depths had more

runs at their trained classification accuracy for all numbers of weights in error. This may be because each kernel has more weights, so it is harder to affect the output of a single kernel with individual weight bit flips. When comparing networks with the same size of kernel, but different depths, the correlation is not as strong. In general though, the deeper networks performance degraded faster than the shallower networks, and under the maximum number of weight errors, the deeper networks had fewer runs at their trained accuracies than the shallower networks. This may be due to the sparse connections within CNNs. This means that shallower networks have fewer indirect connections between the input and output, while the deeper networks have more indirect connections. More indirect connections means that bit flips in one part of the network have a larger affect on the rest of the network.

Comparing the MLPs and CNNs, one clear pattern emerges. All of the tested CNNs showed bimodal behavior, while only the smallest MLP network was bimodal. Additionally the ThreeLayers and FiveLayers MLP networks both seemed to perform better than any of the CNN networks under weight errors. Both ThreeLayers(Fig. 3.2) and FiveLayers(Fig. 3.3) achieved over 175 runs at their trained accuracies with only 10 weights in error. The best result by the CNNs for that many weight errors is only above 160 runs. Additionally, ThreeLayers and FiveLayers maintain their accuracy better under increasing weight errors than the CNNs. FiveLayers especially has good performance, only dipping down to about 75 runs at its trained accuracy with 100 weight errors compared to the best CNN (TwoLayersLarge) dipping down to 40 runs. In general, it appears that the MLP networks are more robust to weight errors than the CNNs. This could be due to the number of weights in each network. The ThreeLayers MLP network has 1,335,000 weights, while the FourLayersSmall CNN network has 1,060,224 weights. Additionally, the MLPs were working on a much simpler dataset compared to the CNNs which could also help explain the difference in robustness.

Table 3.3: Neural Network Performance Under 100 Weight Errors

	% of Trials at Trained Accuracy
MLP OneLayer	17.5%
MLP ThreeLayers	33.5%
MLP FiveLayers	47%
CNN TwoLayersSmall	4.5%
CNN TwoLayersLarge	23%
CNN FourLayersSmall	7.5%
CNN FourLayersLarge	12%

3.3 Future Work

There are several directions this work could take in the future. First, larger CNNs should be tested. Some CNNs such as VGGNet [45] have up to 16 convolutional layers compared to the four convolution layers in the networks in this chapter. Another area that should be explored with the networks is determining which layers are most sensitive to weight errors. This is especially interesting to consider with the CNNs since they have convolutional layers as well as densely connected layers.

3.4 Conclusion

In this chapter, the relationship between weight errors and classification accuracy of modern MLPs and CNNs was explored. Table 3.3 shows a snapshot of the results of the experiments run in this chapter. The results of the experiments show that MLPs are more robust than CNNs to bit flips in their weights. This is an important result because CNNs currently have state-of-the-art performance in computer vision tasks. However, in dangerous environments where the integrity of the hardware or software running the network is in question, CNNs may not be the best option. Alternatively, if CNNs must be used, then networks with larger kernels are a better option than networks with small kernels.

Chapter 4

Robustness of Deep Learning Architectures

The previous chapter leads to some insights for future experiments. Even though the fully connected MLPs are more robust than the CNNs, CNNs are much more capable especially when it comes to computer vision tasks. Thus this chapter will focus on further improving the investigation of CNN robustness in several ways. First, the CNNs tested in the previous chapter were very small compared to newer CNNs. Additionally the CIFAR10 dataset, while a good starting point, is not the gold standard when it comes to determining if a network is ready for real-world applications. So in this chapter, instead of developing small, custom CNNs, cutting edge CNNs from other researchers are tested for their robustness. The networks chosen are VGG16, ResNet50, and InceptionV3. Each of these networks at one point had the highest classification performance on the ImageNet dataset [10]. ImageNet is a very large dataset containing several million images and over 1,000 classes.

4.1 Tested Architectures

4.1.1 VGG16

VGG16 was introduced in 2014 by the Visual Geometry Group at Oxford [46]. Their main contribution was that they used only one filter size in their convolutional layers: 3x3. By using one small filter size they could build extremely deep networks. This architecture allowed the team to place first and second in ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014. The VGG16 architecture has 13 convolutional layers, 5 max pooling layers, and 3 fully connected layers. Most of the convolutional layers have a 3x3 filter size and use a stride of 1, but there are also 3 layers that use 1x1 filters. The max pooling layers all use 2x2 windows with a stride of 2. Two fully connected layers have 4096 nodes, and one has only 1000 nodes. The output of the network is a softmax function. Dropout [49] was used in training between the first two fully connected layers.

4.1.2 ResNet50

ResNet50 was introduced in 2015 by researchers at Microsoft [19]. Their main contribution was the addition of shortcut or residual connections between different layers. An example residual connection is shown in Figure 4.1. These residual connections ease the training of deep convolutional networks and allowed Microsoft to build extremely deep networks. In the case of this chapter their 50 layer network is tested, although Microsoft did implement an 152 layer network. Like VGG16, ResNet50 uses mostly 3x3 convolutional filters although there are a few 1x1 convolutional filters as well. Some differences are that ResNet50 uses a stride of 2 in the convolutional layers, does not use dropout during training, and utilizes batch normalization [23] between the convolutional layers.

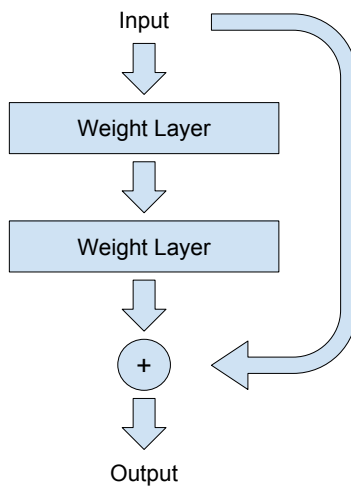


Figure 4.1: An example of a residual connection. Note how the shortcut connection skips two weight layers, not just one. In ResNet50 these shortcut connections are only over convolutional layers. Sometimes the shortcut connection will have 1x1 convolutions in it to correct the dimensions of the data.

4.1.3 InceptionV3

InceptionV3 was introduced in 2016 by researchers at Google [54]. InceptionV3 is an iteration of the network developed in [53], which introduced the inception architecture. Inception layers are convolutional layers that contain multiple different filter sizes in the same layer. The output of all the different filters is concatenated together before being sent to the next layer. Figure 4.2 gives an example of an inception layer from the first paper. The overall architecture of InceptionV3 includes a mix of normal convolutional layers, several inception layers, max pooling layers, and a single dense layer. InceptionV3 was trained with dropout like VGG16 and used batch normalization like ResNet50.

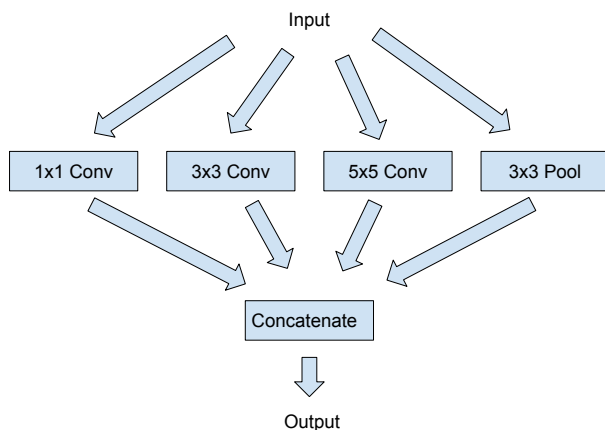


Figure 4.2: An example of an inception layer. This is a simple example from early versions of the Inception architecture. In InceptionV3 there are multiple layers between the input and concatenation stage.

4.2 Results

4.2.1 Methodology

The architectures tested in this chapter were implemented in the Keras Python library with Tensorflow as the backend. GPUs were utilized for executing the networks, but no training was done. No training was needed because the models were instantiated with weights provided by the original researchers for classification against the ImageNet dataset. ImageNet is a very large dataset of over 1 million images with 1,000 classes to distinguish. For the tests below, only the validation data of 50,000 images was used to calculate the performance of the neural networks. For the baseline accuracy, the networks were run on the entire 50,000 image validation dataset, but for testing against weight errors the networks were only tested against 5,000 validation images. The performance of the networks is shown in Table 4.1. Internal errors were simulated by choosing a random trainable parameter in the network and flipping a random bit in said parameter. The parameters are represented as single-precision floating point numbers, so a single bit flip could have a very small effect

Table 4.1: Tested Model Data

Model	Accuracy	Total Parameters
VGG16	85.46%	138,357,544
ResNet50	88.44%	25,636,712
InceptionV3	93.08%	23,851,784

if it is in the fraction, or a very large effect if it is in the exponent or sign bit. The number of parameters to affect was determined based on the total number of parameters in the network (also shown in Table 4.1). The parameters that could be affected are the input weights to convolutional and dense layers, the bias parameter in convolutional and dense layers, and batch normalization parameters in the case of ResNet50 and InceptionV3. The number of parameters affected started at $1 * 10^{-5}$ % of the total parameters, and increased to $10 * 10^{-5}$ % over the course of testing. For each percentage value there were 100 different runs to determine the overall performance of the networks. An example execution on VGG16 would go as follows. Since there are 138,357,544 total parameters, at $1 * 10^{-5}$ % only 14 simulated bit flips occur. After choosing 14 random parameters, and flipping a random bit in each parameter, the classification accuracy of VGG16 is tested on 5,000 validation images. This process is repeated 99 more times with a different 14 parameters affected by a bit flip each time, before moving on to the $2 * 10^{-5}$ % or 27 of the weights being in error.

4.2.2 VGG16 Results

Figure 4.3 shows the performance of the VGG16 network under memory errors. This network had a baseline accuracy of 85.46% on the 50,000 validation images in ImageNet. With SEUs affecting the network, the behavior of the network is extremely bimodal. The classification accuracy during tests at each percentage tend to cluster at or near the baseline accuracy, or extremely low accuracy displaying random guessing behavior. There are very few runs that

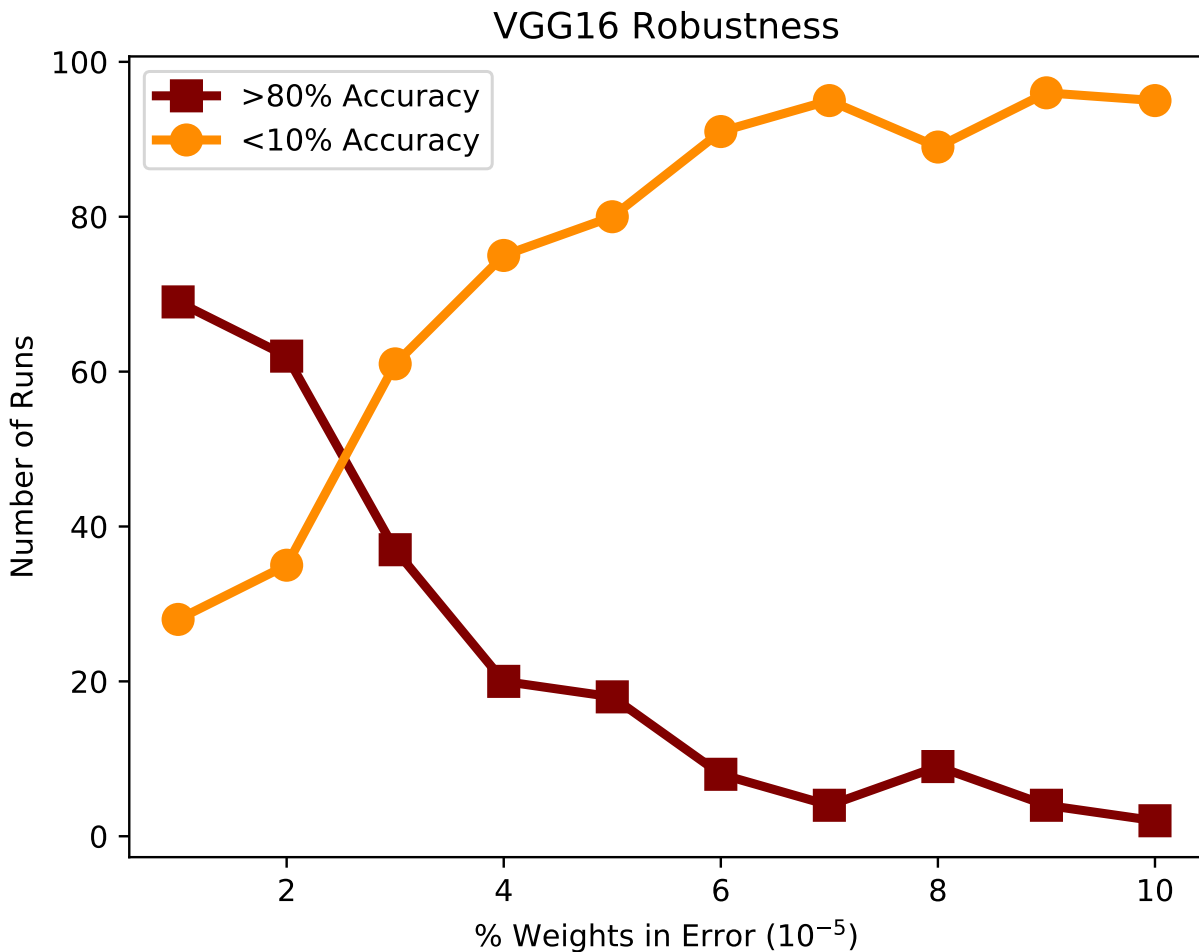


Figure 4.3: The performance of the VGG16 model under various weight errors. For each percentage of weights in error shown on the x axis, the trend lines show how many of the 100 trials ended up at greater than 80% accuracy or less than 10% accuracy.

return a classification accuracy somewhere in the middle. Figure 4.3 shows that with only 1×10^{-5} % of the total weights in error, VGG16 had about 70 runs above 80% accuracy and about 30 runs below 10%. At 3×10^{-5} %, the number of runs below 10% accuracy exceeds the number of runs above 80% accuracy for the first time. By the time 7×10^{-5} % of the parameters are affected by bit flips, VGG16 is effectively unusable. The percentages tested are very small due to the huge size of VGG16, to put them in context 3×10^{-5} % (the point where most of the runs are below 10% accuracy), would be 42 simultaneous bit flips

in memory. So in some ways VGG16 does not seem very robust to bit flips, because very few of the total weights need to be affected to significantly reduce performance. But at the same time, the likelihood of tens of bit flips occurring in memory must be considered when it comes to practical applications.

4.2.3 ResNet50 Results

Figure 4.4 shows the performance of the ResNet50 network under memory errors. This network had a baseline accuracy of 88.44% on the 50,000 validation images in ImageNet. ResNet50 displayed the same bimodal behavior as VGG16 with most of the test run accuracies ending near the baseline accuracy or very low accuracy. Figure 4.4 shows that with only 1×10^{-5} % of the total parameters in error, ResNet50 maintained very good performance with almost all 100 runs executing at over 80% accuracy and no runs coming in at under 10% accuracy. All the way to 10×10^{-5} % of the parameters in error, ResNet50 has more runs over 80% accuracy than under 10% accuracy. At 10×10^{-5} %, approximately 55 runs were over 80%, and approximately 30 runs were under 10%. This shows that ResNet50 is significantly more robust than VGG16 at resisting bit flips in its trained parameters. However, it should be noted that ResNet50 has significantly fewer parameters than VGG16. ResNet50 has 25,636,712 parameters compared to VGG16's 138,357,544, an order of magnitude fewer. This means that for ResNet 10×10^{-5} % of parameters in error is only 27 simultaneous errors compared to the 138 simultaneous errors for VGG16 at 10×10^{-5} %.

Due to ResNet50's increased robustness at the same percentage of parameter errors as VGG16, more testing was done to see at what point ResNet50 became unusable. Figure 4.5 shows the results of testing ResNet50 with more weights in error. Notice that the scale for parameters in error has changed from 10^5 to 10^4 . Now there is a point where outputs runs of

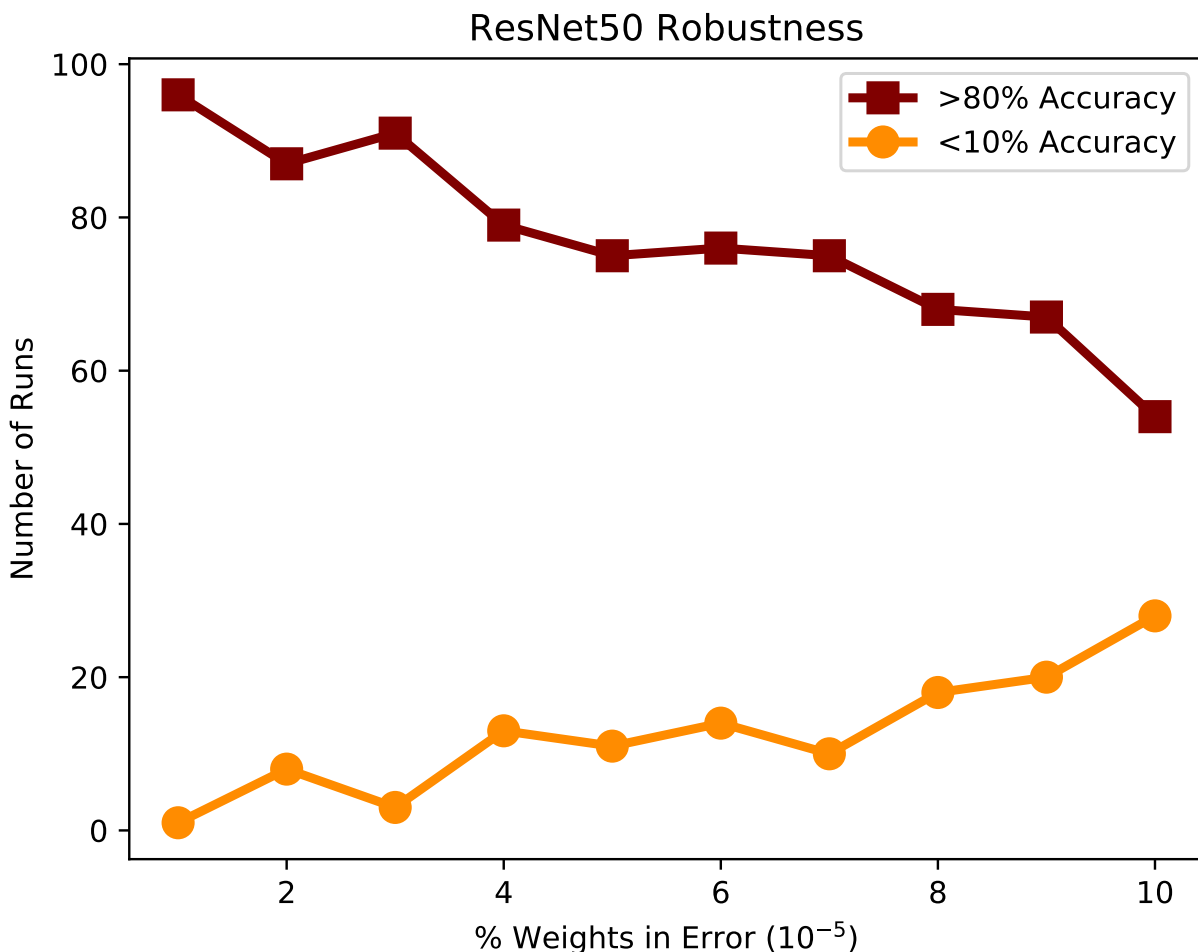


Figure 4.4: The performance of the ResNet model under various weight errors. For each percentage of weights in error shown on the x axis, the trend lines show how many of the 100 trials ended up at greater than 80% accuracy or less than 10% accuracy.

less than 10% accuracy exceed runs of greater than 80% accuracy: 2×10^{-4} % of parameters in error. By the time 6×10^{-4} % of the weights are in error approximately 90 of the runs are less than 10% classification accuracy, making ResNet50 essentially unusable. So in terms of percentages, ResNet50 can withstand an order of magnitude more memory errors than VGG16. In terms of raw numbers though, ResNet50 and VGG16 are quite similar. VGG16 crossed from more runs greater than 80% to more runs less than 10% at 42 simultaneous errors, while for ResNet50 that point occurs at 51 simultaneous errors.

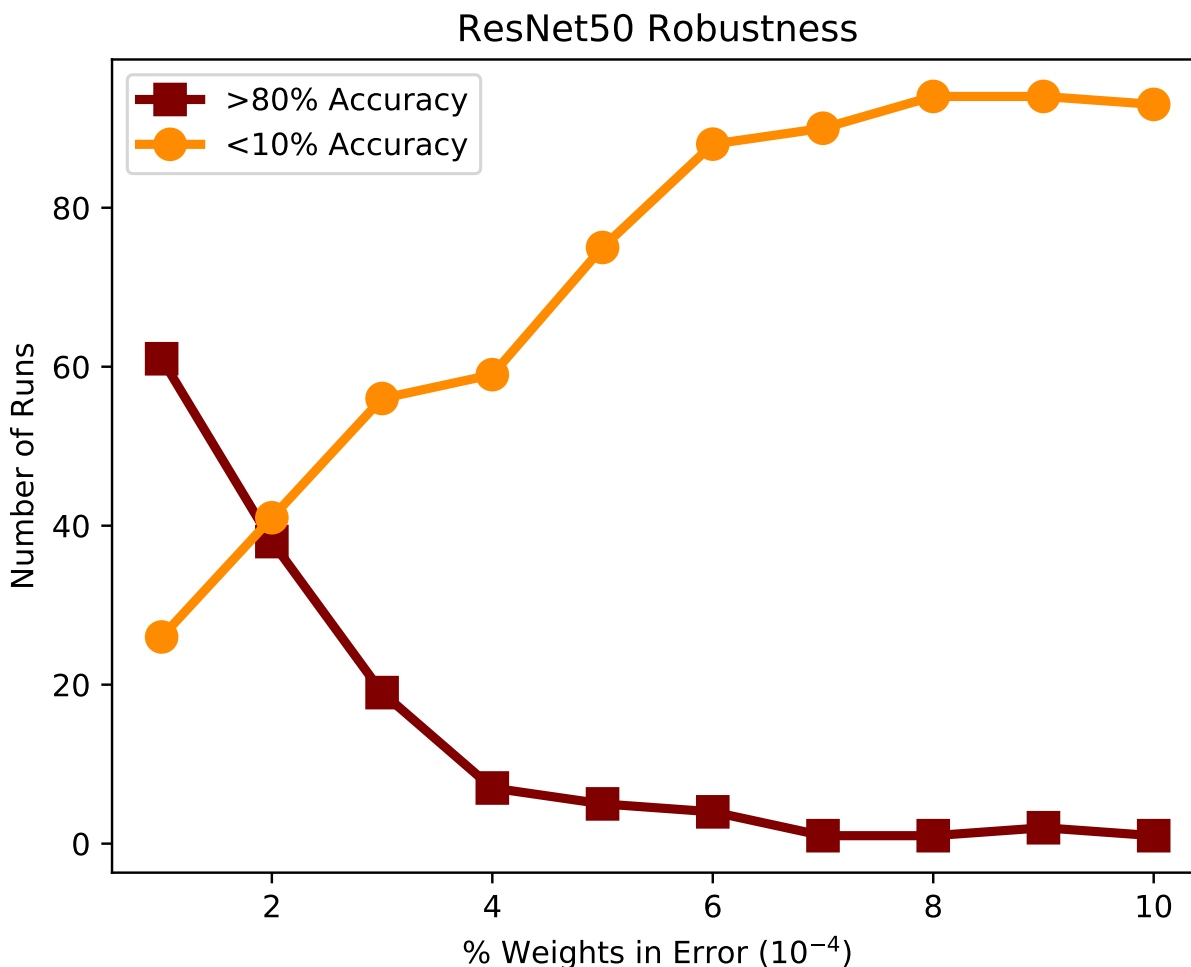


Figure 4.5: The performance of the ResNet model under various weight errors. For each percentage of weights in error shown on the x axis, the trend lines show how many of the 100 trials ended up at greater than 80% accuracy or less than 10% accuracy.

4.2.4 InceptionV3 Results

Figure 4.6 shows the performance of the InceptionV3 network under memory errors. This network had a baseline accuracy of 93.08% on the 50,000 validation images in ImageNet. InceptionV3 showed the same bimodal behavior as VGG16 and ResNet50 with most of the test run accuracies returning near the baseline accuracy or at very low accuracies. Figure 4.6 shows that with only 1×10^{-5} % of the total parameters in error, InceptionV3 maintained

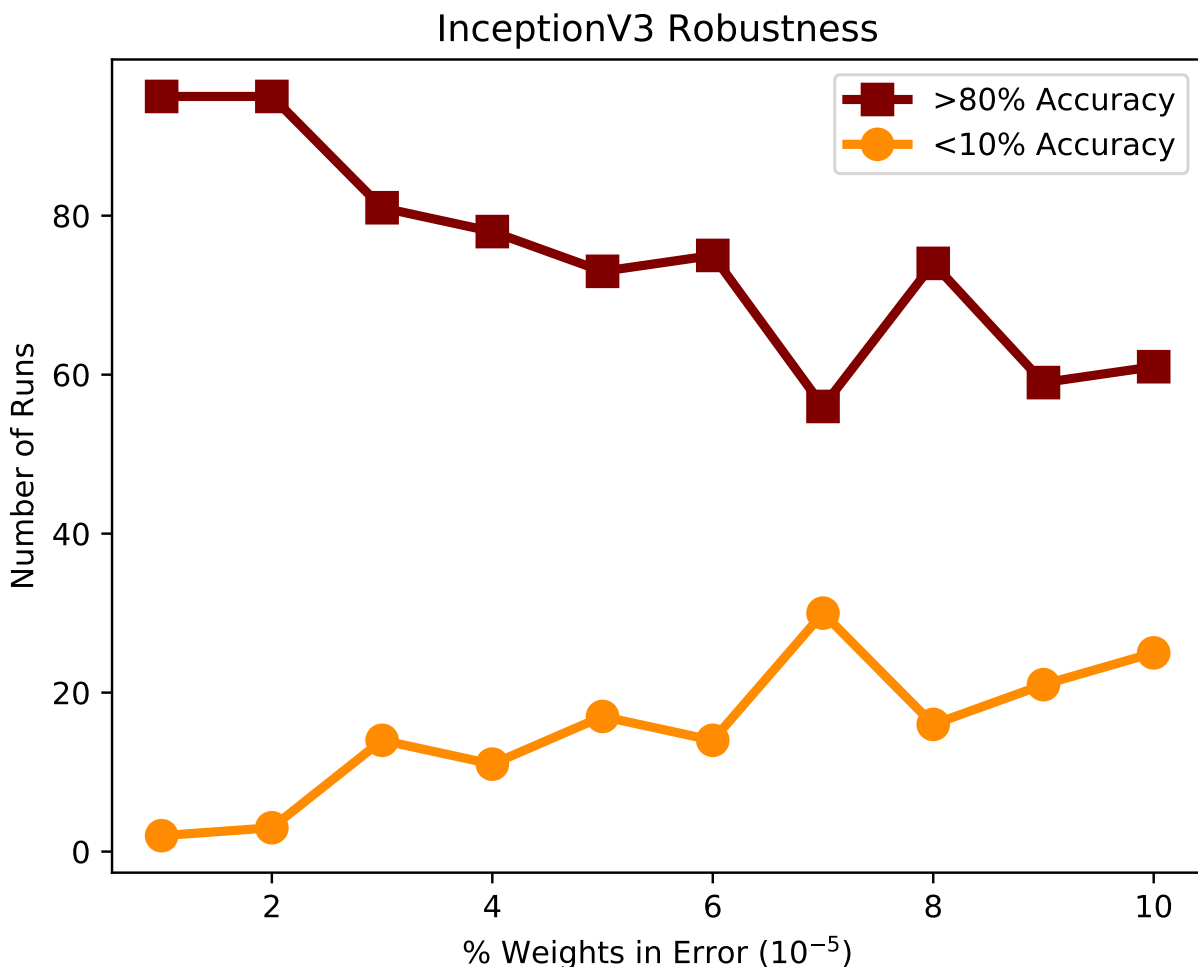


Figure 4.6: The performance of the InceptionV3 model under various weight errors. For each percentage of weights in error shown on the x axis, the trend lines show how many of the 100 trials ended up at greater than 80% accuracy or less than 10% accuracy.

very good performance with almost all the test runs returning over 80% classification accuracy. All the way to 10×10^{-5} % of the total parameters in error InceptionV3 has more test runs over 80% accuracy than under 10% accuracy, just like ResNet50. However there is some anomalous performance at 7×10^{-5} %, where the number of runs below 10% accuracy jump significantly before returning back to the regular trend line. This goes to show that, even though in general InceptionV3 maintains its performance fairly well, if the wrong parameters are affected then performance can decrease significantly. Overall, the performance of Incep-

tionV3 is very similar to ResNet50. InceptionV3 is significantly more robust to weight errors than VGG16, and has an order of magnitude fewer parameters than VGG16. Compared to ResNet50 there are signs that InceptionV3 may be a little more robust. At $10 \times 10^{-5} \%$ of the total parameters in error, ResNet50 had approximately 55 runs over 80% accuracy, while InceptionV3 had at least 60 runs over 80%. Also of note InceptionV3 has slightly less parameters than ResNet50.

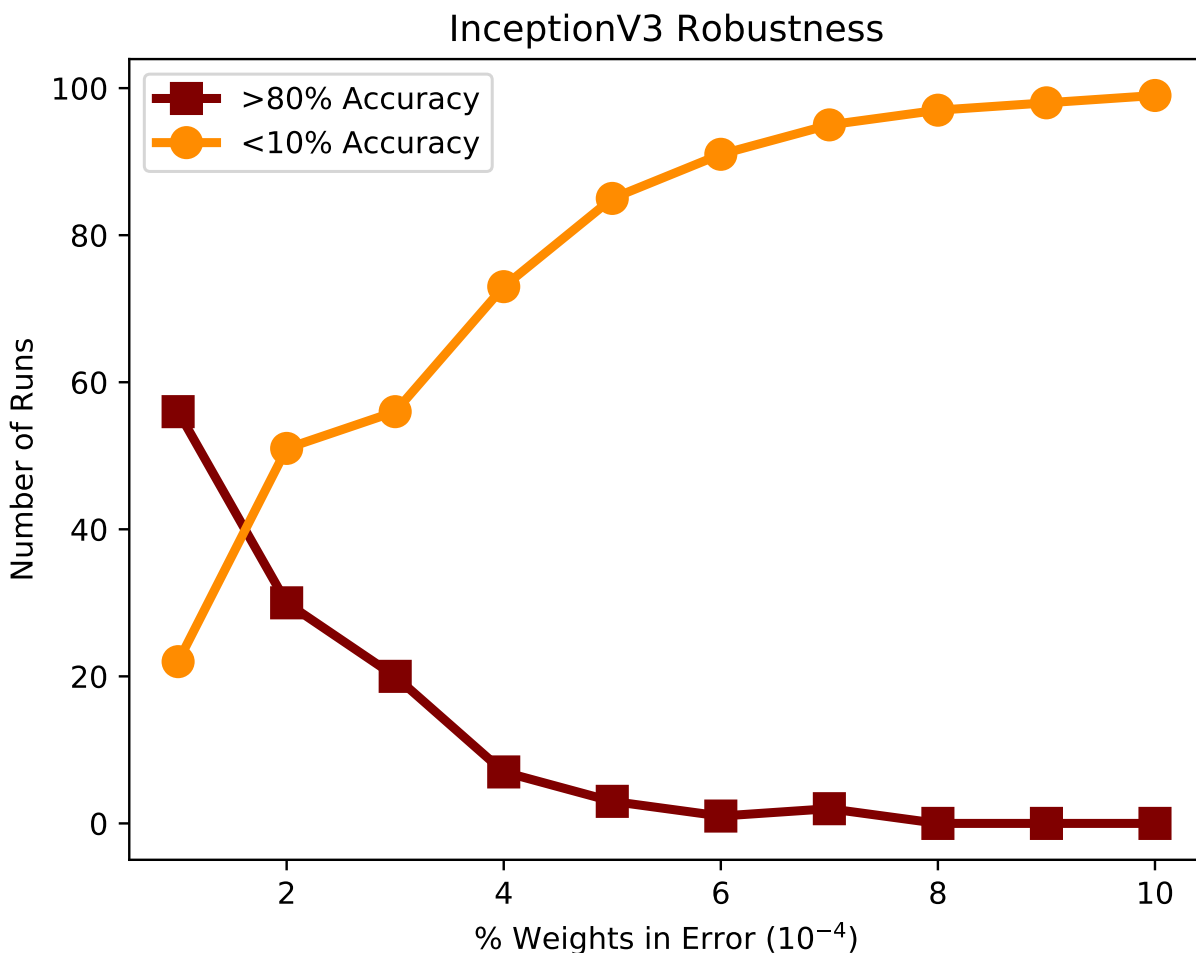


Figure 4.7: The performance of the InceptionV3 model under various weight errors. For each percentage of weights in error shown on the x axis, the trend lines show how many of the 100 trials ended up at greater than 80% accuracy or less than 10% accuracy.

Since InceptionV3 was more robust than VGG16, additional testing was done at higher

percentage values to find out when InceptionV3 became unusable. Figure 4.7 shows the results of testing InceptionV3 with more memory errors. Notice that the scale for parameters in error has changed from 10^{-5} to 10^{-4} . At 2×10^{-4} % of the total parameters in error the number of runs less than 10% accuracy exceeds the number of runs at greater than 80% accuracy. By the time 0.000006% of the total parameters are affected InceptionV3 is unusable with approximately 90 runs executing at less than 10% accuracy. So again InceptionV3 has very similar performance to ResNet50. Based on the run at 1×10^{-4} %, it was theorized that InceptionV3 may be more robust than ResNet50, but in the end both networks had the same points where the 10% accuracy runs exceeded the 80% accuracy runs.

4.2.5 Comparison of Results

After simulating bit flips in the parameters of all three neural networks, their performance can be compared against each other. VGG16 was much less robust than ResNet50 or InceptionV3. It took a much smaller percentage of the total parameters to be in error for VGG16 to become unusable than for ResNet50 or InceptionV3. ResNet50 and InceptionV3 had very similar robustness, with each network becoming unusable at the same percentage of parameters in error. ResNet50 and InceptionV3 are fairly different convolutional neural networks, but one thing they have in common that VGG16 does not have is the use of batch normalization. Batch normalization reduces covariance shift and normalizes the input to each layer in the network which helps speed up learning. It may also help the network resist errant bit flips during inference. A second commonality between ResNet50 and InceptionV3 is shortcut layers. Shortcut layers appear in InceptionV3 in the form of 1x1 convolutions which essentially pass the data from the input layer straight to the concatenation layer. These shortcut connections could help robustness by allowing deeper layers in the network to see data that wasn't tainted by bit flips. Another interesting trend is that, although the

percentage where VGG16 breaks down is much smaller than ResNet50 or InceptionV3, the total number of errors is fairly similar. When testing VGG16, it takes 42 simultaneous bit flips for the number of test runs outputting less than 10% classification accuracy to exceed the number of test runs over 80% classification accuracy. For ResNet50, it takes 51 simultaneous bit flips to reach the same point and InceptionV3 takes 48 simultaneous bit flips. So although ResNet50 and InceptionV3 are still more robust than VGG16 when measured in terms of raw number of bit flips, the gap seems smaller.

4.3 Future Work

To test the theory that batch normalization is responsible for robustness, it should be added to the VGG16 network. Additionally the parameter errors could be concentrated in either the batch normalization layers, the convolutional layers, or the dense layers to see which kind of layer is most harmed by bit flips. There are also other architectures besides just convolutional neural networks to test. Recurrent neural networks for example are often used for natural language processing and it is possible to use batch normalization with those networks [32]. Additionally these experiments can be rerun as new neural network architectures are developed.

4.4 Conclusion

In this chapter, three modern, deep, convolutional neural networks were tested for their robustness against memory errors. Memory errors can be caused by malicious actors, nuclear radiation, or cosmic rays and any system that requires high reliability must be resilient against such errors. After testing VGG16, ResNet50, and InceptionV3 against bit flips in

Table 4.2: Overview of Results

Model	Runs < 10% Exceed Runs > 80% as % of total parameters	Runs < 10% Exceeds 90 as % of total parameters
VGG16	$3 \times 10^{-5} \%$	$5 \times 10^{-5} \%$
ResNet50	$2 \times 10^{-4} \%$	$6 \times 10^{-4} \%$
InceptionV3	$2 \times 10^{-4} \%$	$6 \times 10^{-4} \%$

their trained parameters, it was shown that ResNet50 and InceptionV3 were shown to be much more robust than VGG16 (see Table 4.2). It was hypothesized that the use of batch normalization is what made ResNet50 and InceptionV3 resilient against memory errors.

Chapter 5

Survivability in Real World Environments

In Chapter 3 and Chapter 4, experimental results for testing multiple neural network architectures were presented. The results compared the robustness of the different architectures in a vacuum, with no context for how the networks will perform in real world scenarios. A classic real world scenario where a neural network would need to resist memory errors is if the network was running on a satellite in space. This chapter will explore a space application which is enabled by neural networks, and then compare if the neural network architectures explored in previous chapters are robust enough to survive in the space environment.

5.1 Neural Network Space Application

Due to decreasing launch costs and the development of more capable small satellites, commercial activity in space is picking up. In particular, there are several new companies focusing on the earth imagery market, which is a good application to apply neural networks. An application developed in this thesis that combines neural networks and satellite imagery is an open ocean ship detector.

The satellite imagery comes from Planet’s Open California dataset. Open California contains images from Planet’s RapidEye satellite since 2013, and Dove satellites since 2015. The

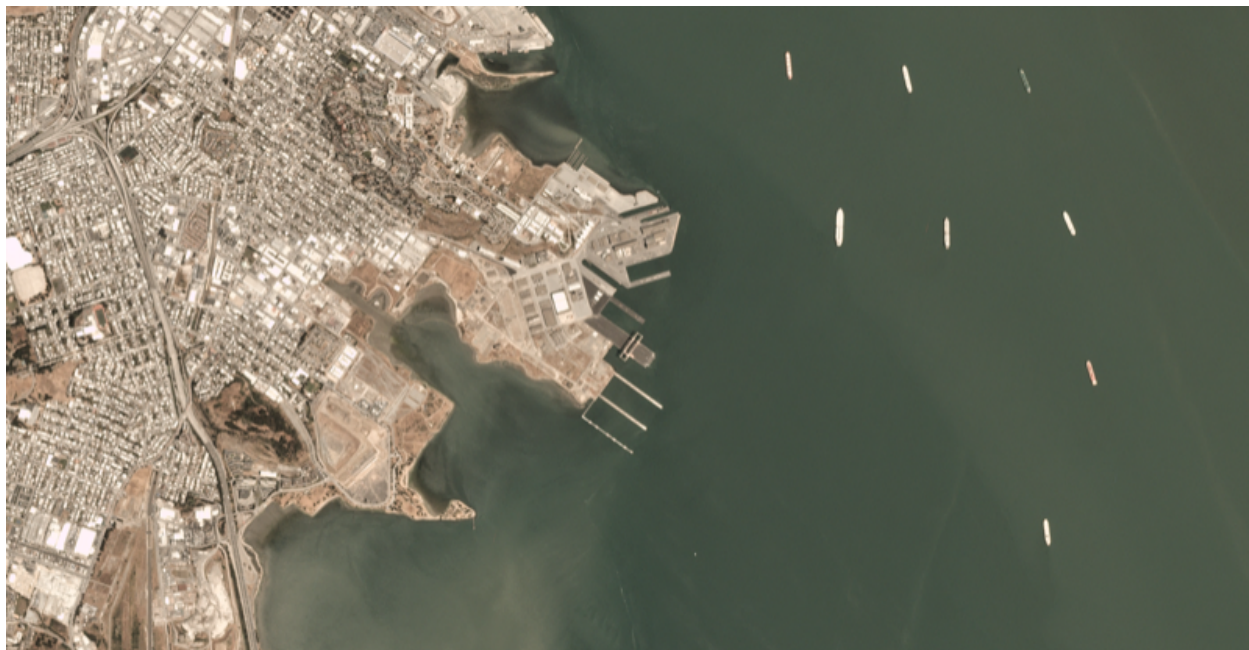


Figure 5.1: An example of imagery from the Open California dataset. This image shows part of the San Fransisco Bay.

images come in RGB or multispectral format, and have a maximum resolution of 3 meters. An example of a scene from the Open California dataset is shown in Figure 5.1.

The application is comprised of two main parts. First, objects are detected on the water. Then those objects are classified as a ship or not. Detection of objects on the water is done with traditional computer vision techniques. The first step in the object detection stage is executing a Canny edge detection algorithm on the satellite image, which produces an image as shown in Figure 5.2. A sliding window of size 80 x 80 pixels with a stride of 20 pixels is then moved over the edge image to generate image chips. To save processing time, not every single chip is sent on to the detection stage. Instead, only chips whose white pixel count lands in the correct range are sent on to the classification stage. Chips with a high number of white pixels means there were many edges detected, which means the image chip is probably over a section of land where there would not be a ship. Chips with almost zero white pixels means there were almost no edges detected, so the chip is probably over open ocean where

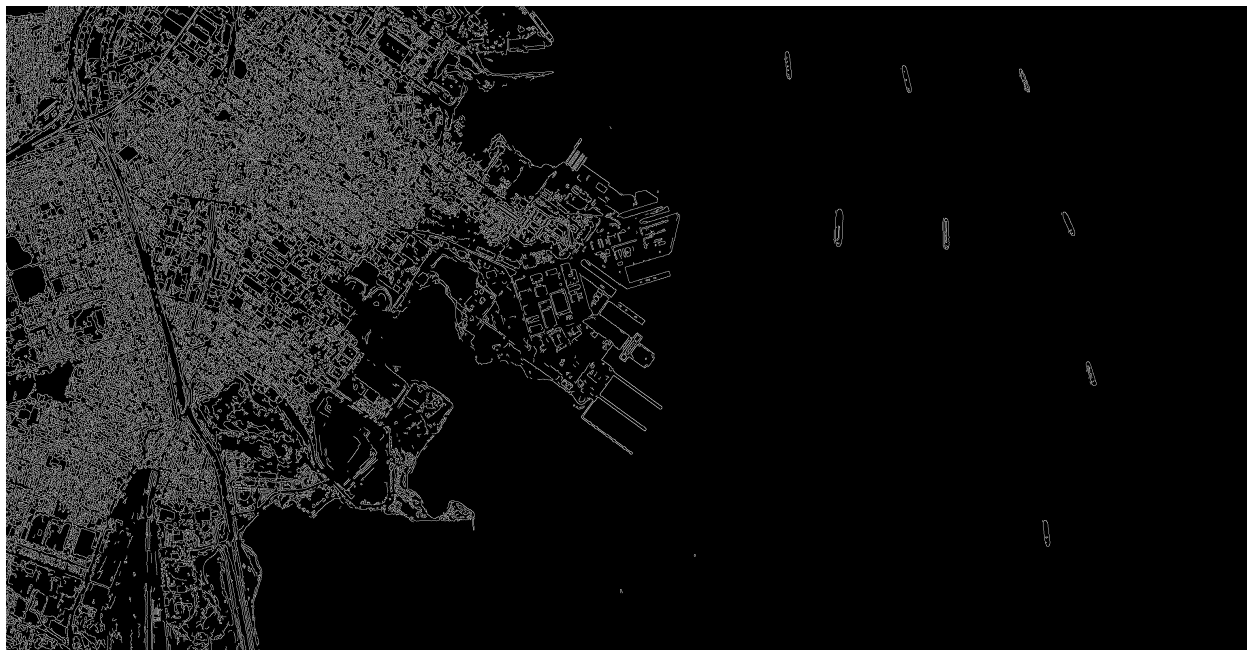


Figure 5.2: The output of the Canny edge detection algorithm. The detected edge pixels are white, while all other pixels are black

there is not a ship. Somewhere in the middle are image chips in which the correct number of white pixels indicates there could be a ship present in the image chip. Figure 5.3 Shows examples of image chips with different white pixel counts. After some experimentation, it was determined that a range of 50 to 600 white pixels was appropriate and any image chips whose white pixel count landed in that range would be considered a possible ship.

The sliding window is applied over the entire image and all image chips that pass the white pixel threshold are saved. All of the saved image chips are then sent to the classification stage. The classification stage is made up of a deep convolutional neural network inspired by the design of VGGnet [46], although not quite as deep. The architecture is trained with dropout to prevent overfitting and uses the relu activation function throughout. The output is a single sigmoid layer where zero represents no boat and one represents boat. The neural network was trained on the Ships in Satellite imagery dataset on Kaggle [17]. The dataset images are all RGB and sized 80 x 80. There are 700 images of ships, and 2100 images of

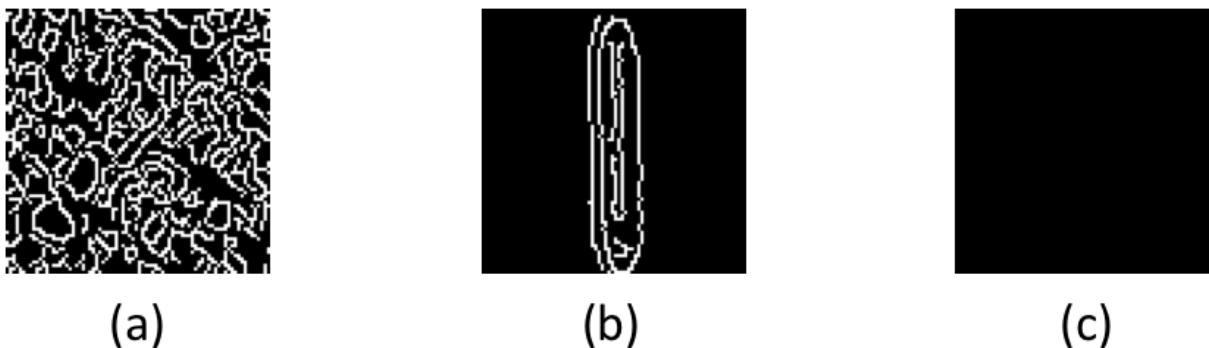


Figure 5.3: This figure shows different image chips from sliding an 80x80 window over the satellite image after edge detection. (a) is from over land, and has a high number of white pixels. (b) is a boat on the water and has a small number of white pixels. (c) is over open ocean and has zero white pixels.

not ships. 600 images of ships and 1700 images of not ships were used for training, while the rest were used for validation. The not ship class includes land, ocean, partial ships, and the wake from possible ships. The images all from from Planet Open California images, so they have the same resolution and parameters as described above. During training a small amount of data augmentation is used, specifically the images are randomly rotated between 0 and 40 degrees and horizontally flipped. The neural network was built in Keras with the Tensorflow backend, and trained on a Nvidia GTX 1080 Ti.

After being trained to a high accuracy with low false positive counts, the neural network along with the sliding window detector was moved onto the Nvidia TX2 to test its performance. Figure 5.4 shows the output of the ship detection algorithm on the image Sfbay_3. The algorithm successfully detected all the ships in the image, along with a few false positives. In the upper left corner there is a false positive where half the image chip was ocean and the other half was an artificial dock. There are also some false positives in the middle right due to some wake. Also notice how the algorithm can detect individual boats multiple times. This is because the sliding window only moves over and down by 10 pixels, so neighboring image chips can sometimes both fit inside the correct white pixel count range and thus they



Figure 5.4: The output of the ship detection algorithm on the image Sfbay_3. The green boxes represent image chips classified as a ship. There are a few false positives, but no missed ships.

are both sent to the neural network classifier.

This kind of algorithm perfectly combines the growing satellite imagery market with the success that CNNs have had with image processing. Hosting the CNN application onboard satellites opens up new opportunities for satellite imagery companies. For example the satellites could prioritize downlinking specific photos based on what the neural networks detect. Or the satellites could alert each other in real time to events unfolding on the ground, such as a natural disaster, so that there are multiple instruments looking at the event at the same time. Additionally, rovers could use advanced object and terrain detection to move through difficult terrain more quickly and perform a wider range of science experiments. However, this is all reliant on the robustness of neural network algorithms and if they can survive in the harsh space environment.

5.2 Calculating Error Rates

To determine if the neural network algorithms examined in previous chapters are robust enough, their performance must be compared against real-world error rates. As mentioned previously, these errors can be caused by high energy particles penetrating the memory chip and causing bits to flip. The number of particles passing through a volume at any given time is the particle flux, which is known for several different locations. According to the JEDEC JESD89A standard [2], the total 10 MeV neutron flux at New York City is $3.6 \times 10^{-3} \text{ cm}^{-2}\text{s}^{-1}$. By combining the flux with the size of the memory taken up by a neural network, the number of particles passing through the neural network each second can be calculated. However, every neutron passing through a silicon chip does not cause an error. Average error rates change depending on which memory technology (such as SRAM or DRAM) is subject to the high energy particles. Additionally the size of the neural network in memory will depend on the memory technology storing the network's parameters.

The memory technology chosen for these initial calculations will be SRAM. As described before, SRAM is generally used for the cache of the computer so it is where most of the trained parameters will be stored during execution of the network. To calculate the expected number of memory errors the error rate of SRAM must be known as well as the size of the neural network when stored in SRAM. From [47], the error rate of SRAM was determined to be 10^{-3} FIT/bit . FIT stands for failure in time and 1 FIT is one error per billion device-hours. Another way to say that is: given one billion transistors, in one hour of operation, one of them would fail. The size of the neural network can be determined based on how many bits the neural network takes up and the size of a single SRAM cell. A single six transistor SRAM cell, when built using 16nm technology, is $0.07 \mu^2$ or $7 \times 10^{-10} \text{ cm}^2$ [6]. The first network to be tested for real-world error rates is VGG16. VGG16 has 138,357,544 trainable parameters, each stored as a single precision floating point number that takes up 32 bits.

The total number of bits taken up by VGG16 is $138,357,544 * 32 = 4,427,441,408$ bits which leads to:

$$4,427,441,408 \text{ bits} \times 7 \times 10^{-10} \text{ cm}^2 = 3.1 \text{ cm}^2$$

VGG16 takes up 3.1 cm^2 when stored on an SRAM implemented in a 16nm technology node. Given that area and the known neutron flux at New York City, the total number of neutrons per second passing through VGG16 at New York City can be calculated:

$$\frac{3.6 \times 10^{-3} \text{ neutrons}}{\text{cm}^2 \text{ sec}} \times 3.1 \text{ cm}^2 = \frac{0.01116 \text{ neutrons}}{\text{sec}}$$

After calculating the number of neutrons per second that pass through VGG16, the error rate of the SRAM can be used to determine how many errors per second would occur for VGG16 at New York City.

$$\frac{10^{-3} \text{ errors}}{10^9 \text{ hours bit}} \times 4,427,441,408 \text{ bits} = \frac{4,427,441 \text{ errors}}{10^9 \text{ hours}} = \frac{0.0044 \text{ errors}}{\text{hour}}$$

Finally, the error rate per hour can be combined with the neutron rate per second to determine how many particles it takes to induce an error in VGG16 when stored on 16 nm SRAM.

$$\frac{0.004 \text{ errors}}{\text{hour}} \times \frac{1 \text{ hour}}{3600 \text{ sec}} \times \frac{1 \text{ sec}}{0.01116 \text{ neutrons}} = \frac{1.1 \times 10^{-4} \text{ errors}}{\text{neutron}} = \frac{9,074 \text{ neutrons}}{\text{error}}$$

By knowing how many neutrons it takes to produce an error for VGG16 when stored on SRAM, it is possible to calculate the predicted error rates for environments outside of New York City. All it takes is knowing the neutron flux value for the new environment. For example, several experiments have been run on the International Space Station (ISS) to determine neutron flux in Low Earth Orbit (LEO) [11] [26]. From those sources the neutron

Table 5.1: Error Rates of Neural Networks in Different Space Radiation Environments

	ISS 20 neutrons/cm ²	Moon Surface 1 × 10 ⁻¹ neutrons/cm ²	Mars Surface 1 × 10 ⁶ neutrons/cm ²
MLP FiveLayers	<u>28 error</u> day	<u>0.14 error</u> day	<u>1.4 × 10⁶ error</u> day
VGG16	<u>590 errors</u> day	<u>3 errors</u> day	<u>2.95 × 10⁷ errors</u> day
ResNet50	<u>112 errors</u> day	<u>0.56 errors</u> day	<u>5.6 × 10⁶ errors</u> day
InceptionV3	<u>102 errors</u> day	<u>0.51 errors</u> day	<u>5.1 × 10⁶ errors</u> day

flux on the ISS is about 20 neutrons/cm² sec which can be used to calculate the expected error rate of VGG16 if it were executed in an SRAM chip on a satellite in a LEO orbit.

$$\frac{20 \text{ neutrons}}{\text{cm}^2 \text{ sec}} \times 3.1 \text{ cm}^2 = \frac{62 \text{ neutrons}}{\text{sec}}$$

$$\frac{9,074 \text{ neutrons}}{\text{error}} \times \frac{1 \text{ sec}}{62 \text{ neutrons}} = \frac{146.4 \text{ sec}}{\text{error}} \times \frac{1 \text{ day}}{86,400 \text{ sec}} = \frac{0.0017 \text{ days}}{\text{error}} = \frac{590 \text{ errors}}{\text{day}}$$

This process can be repeated for multiple different location as well as for the other neural networks presented in this thesis. The results of those calculations is shown in Table 5.1

As mentioned before, the ISS neutron flux numbers come from studies of its radiation environment, primarily to make sure astronauts are not receiving dangerous levels of radiation. Getselev et al. [11] reports a maximum flux of 40 neutrons/cm² sec from their Monte Carlo simulations. These simulations are confirmed by also simulating neutron flux numbers for other satellites that have real world neutron flux measurements. The simulation flux includes all particles with energy greater than 10 MeV. Koshiishi et al. [26] however only reports a flux of 5 × 10⁻² neutrons/cm² sec which is based on measurements from onboard the ISS. However, that flux number is only the particles with an energy of 10 MeV, not particles with an energy higher than that. Additionally, the measurements taken by Koshiishi et al. [26] were during a particularly active time of the solar cycle, which would reduce the amount of Galactic Cosmic Rays (GCRs) reaching earth and thus reduce neutron count. GCRs produce neutrons

by interacting with the shielding on the ISS, so differences in the shielding assumptions in the simulations and the real world shielding could also account for the discrepancy in the two papers. For this thesis, a flux of $20 \text{ neutrons/cm}^2 \text{ sec}$ was chosen because it is in between the numbers from [11] and [26], but favors [11] due to the fact that it reports integral flux and assumes a harsher space environment.

The neutron flux numbers for the Moon’s surface came from simulations by Lingenfelter et al. [34], which were later confirmed after the Apollo missions [35]. The neutron flux numbers were used to estimate the surface composition of the moon, down to depths of tens of centimeters. These neutrons are created when GCRs impact the lunar surface and interact with the materials in the regolith, similarly to how neutrons are produced inside the ISS. Unfortunately, the simulation does not report integral neutron flux, so the chosen flux of $1 \times 10^{-1} \text{ neutrons/cm}^2 \text{ sec}$ is only neutrons with an energy of exactly 10 MeV.

Finally, the Mars surface neutron flux numbers come from Wilson et al. [62] and Khler J. et al. [30]. Similar to the ISS numbers, those sources present simulations and then real world measurements, but are much more in agreement than the ISS papers so $10^6 \text{ neutrons/cm}^2 \text{ sec}$ is very near the results of both papers. The source of neutrons on Mars is from GCRs similar to the ISS and the Moon, but is much higher because there is more material for the GCRs to interact with on Mars. On the ISS, there is just aluminum shielding and on the Moon there is just the regolith. On Mars, there is both the regolith and a small atmosphere for the GCRs to interact with, leading to a much higher neutron flux.

In general, the number of errors expected per day scales as the neutron flux increases and as the size of the network increases. These results are as expected. These error rates now must be compared against the calculated robustness of each neural network. MLPFiveLayer was the most robust MLP tested in chapter 3. It took approximately 80 simultaneous weight errors for MLPFiveLayer to have about 50% of its test runs execute at its trained accuracy.

This is enough robustness to easily survive on the Lunar surface and even LEO orbits similar to the ISS orbit. Assuming that there is zero radiation hardening and no software bit error correction, MLPFiveLayer could maintain acceptable performance on the lunar surface for:

$$80 \text{ errors} \times \frac{1 \text{ day}}{0.14 \text{ errors}} = 571 \text{ days}$$

which is longer than any previous lunar rover survived on the surface of the moon. For a LEO orbit like the ISS orbit though, MLPFiveLayers would only survive:

$$80 \text{ errors} \times \frac{1 \text{ day}}{28 \text{ errors}} = 3 \text{ days}$$

without any sort of shielding or error correction coding (ECC). Table 5.2 shows the expected survival time of each architecture. An architecture is no longer surviving when less than half of the expected runs of the network are at the trained accuracy. Adding ECC or using memory scrubbing techniques such as those used on NASA space probes [40] to the memory holding the trained parameters of MLPFiveLayers would allow it to survive much longer in LEO, because the error rate is very low. The error rate on the Martian surface though is extremely high, with 16 bit flips happening every second. Surviving on the Martian surface requires radiation hardened memory.

VGG16 had the worst performance of the CNNs when it came to resisting bit flip errors. It took 42 simultaneous bit flips for the number of test runs at less than 10% classification accuracy to exceed the runs at greater than 80% classification accuracy. In addition to having low robustness, VGG16 is very large, which leads to a higher expected error rate at each environment. However, VGG16 should still be able to survive for a short time on commercial SRAM chips on the Lunar surface. Without any shielding or ECC, VGG16 could survive

Table 5.2: Survival Times of Neural Networks in Different Space Radiation Environments

	Earth Surface	ISS	Moon Surface	Mars Surface
MLP FiveLayers	15,725 Days	3 Days	571 Days	5.7×10^{-5} Days
VGG16	395 Days	0.07 Days	3 Days Days	1.4×10^{-6} Days
ResNet50	2,524 Days	0.45 Days	91 Days	9.1×10^{-6} Days
InceptionV3	2,609 Days	0.47 Days	94 Days	9.4×10^{-6} Days

for only 14 days, but again the error rate is low enough that ECC would significantly extend the life of VGG16 on the moon. In LEO, VGG16 can expect an error rate of less than one error per minute. This rate should also be low enough for ECC to handle. However, on Mars the expected error rate for VGG16 is 341 bit flips per second, which would require radiation hardened memory to prevent.

In chapter 4 it was determined that ResNet50 and InceptionV3 had approximately similar robustness capabilities. The crossing point between runs over 80% and less than 10% accuracy for ResNet50 was 51 simultaneous bit flips and for InceptionV3 it was 48 simultaneous bit flips. These numbers do not seem that much higher than VGG16, but ResNet50 and InceptionV3 are significantly smaller than VGG16, which leads to overall lower error rates and thus better robustness. ResNet50 and InceptionV3 would both survive longer in commercial SRAM on the Lunar Surface, 91 days for ResNet50 and 94 days for InceptionV3, and LEO would be more manageable compared to VGG16. ResNet50 would have errors of 5 per hour while InceptionV3 would have about 4 per hour. The Martian surface would still pose a challenge though, with each network suffering approximately 60 errors per second.

5.3 Final Remarks

By calculating real-world error rates to compare against the predicted robustness from previous chapters, the ability of neural networks to survive harsh computing environments has

been determined. Overall the results are good. With simple Error Correction Codes, highly capable neural networks such as ResNet50 and InceptionV3 could survive a variety of space environments, most importantly the LEO area which is seeing a large amount of commercial interest. Additionally, older architectures such as MLPs have high enough robustness to be worth considering for certain applications.

It should be noted that the error rate numbers in this chapter were only calculated for SRAM chips, but DRAM is another popular memory technology and very large neural networks that exceed the capacity of the cache which uses SRAM would have to exist in main memory, which uses DRAM. DRAM error rates have been decreasing as the DRAM design rule has decreased. Modern DRAM has an error rate of about 10^{-9} FIT/bit while the error rate used above for SRAM was 10^{-3} FIT/bit. However, DRAM has a significant amount of control logic that is becoming more susceptible to soft errors from high energy particles [47]. Errors in the control logic can lead to several kilobits being flipped simultaneously, which would be disastrous for any of the neural networks studied in this thesis.

Chapter 6

Conclusions

This thesis explored the robustness of various neural network architectures to bit flips in their trained parameters. These trained parameters included the weights and biases of individual neurons, as well as batch normalization means and variances if the network used batch normalization layers. The networks tested for robustness in this thesis were based off the best-performing designs of other researchers, although with different enough architectures that the experiments would provide insights into what makes a neural network robust to weight errors.

Chapter 3 focused primarily on MLP networks and small CNNs. The MLPs were trained on the MNIST dataset, and the depth of the MLPs was varied to determine how it affected their robustness. In general, the MLPs showed good robustness against bit flips. There is a direct correlation in the depth of an MLP and its robustness against bit flips. The `MLPOneLayer` network could only withstand 30 simultaneous bit flips before less than half of the trials were less than the trained accuracy. Meanwhile the `MLPFiveLayer` network could withstand up to 80 simultaneous bit flips before less than half of the trials were less than the trained accuracy. The added robustness of the deeper networks is probably due to the increased number of weights in the deeper networks. By having more weights, each bit flip has a smaller effect on the whole network. The small CNNs were trained on the CIFAR10 dataset and the depth and size of the kernels were adjusted to see how they affected the robustness. An interesting trend found while testing for robustness was the bipolar nature of the results.

After flipping bits in a CNN, the performance of the network tends to either remain near the trained accuracy, or decrease drastically to unusable levels on par with random guessing. The bipolar trend seems to indicate that each weight in the networks has some bearing on the behavior of each different output node, as opposed to only affecting one or two of the output nodes. There was no strong correlation between the depth of the CNNs and robustness, but larger kernels were more robust than smaller kernels. The lack of correlation between the depth of a CNN and its robustness could be due to the increased amount of indirect connections between the input and output as CNNs become deeper. So with a deeper CNN, random bit flips have a larger effect on the rest of the network. Additionally the CNNs tested were less robust than the MLPs. Both `MLPFiveLayers` and `MLPThreeLayers` had a larger percentage of their test trials at their trained accuracies than any of the tested CNNs. `MLPFiveLayers` had 47% of its test runs at its trained accuracies with 100 weight errors, while `CNNTwoLayersLarge`, the best performing CNN, had only 27% of its trials at its trained accuracy with 100 weight errors. This could be due to the different datasets each type of network were trained on. The MLPs were trained on MNIST, which is much easier than CIFAR10 that the CNNs were tested on. Additionally, the MLPs have more weights than the CNNs, so each the bit flips have a smaller effect in the MLPs than the CNNs.

Chapter 4 took lessons from the experiments in Chapter 3 to design a new set of tests. In Chapter 4, significantly larger CNNs were tested on a much harder and more realistic dataset: ImageNet. The three CNNs tested were VGG16, ResNet50, and InceptionV3. Each of these networks at one point had record setting performance on classifying images in the ImageNet dataset while having significantly different architectures. VGG16 is a fairly simple CNN made up of many stacked layers of 3x3 convolutions. ResNet50 utilizes shortcut connections in between layers to decrease training time and enable deeper networks. InceptionV3 has multiple pathways in each layer so the neural network can decide for itself which size kernel is

the best. VGG16 had the weakest performance of the three, while ResNet50 and InceptionV3 had very similar robustness. VGG16 had more runs at less than 10% accuracy than over 80% accuracy when only 3×10^{-7} % of its parameters were in error. To get to the same point in ResNet50 and InceptionV3 it required 2×10^{-6} % of their trained parameters to be in error, an order of magnitude more. Two features that ResNet50 and InceptionV3 have that VGG16 does not are batch normalization layers and shortcut connections, either of which could be responsible for increased robustness. However, all three architectures still had the bimodal response to bit flips. This means that CNNs in general do not fail gracefully as errors accumulate but rather begin to output extremely low classification accuracies without warning.

These robustness numbers were placed into context in Chapter 5. That chapter presented a method for calculating expected bit flips in specific kinds of memory based on the neutron flux of the environment where the neural network would be operating. The environments explored were the orbit of the International Space Station, the Lunar surface, and the Martian surface. These tests showed that MLPFiveLayers, VGG16, ResNet50, and InceptionV3 could all survive on the Lunar surface because the flux of neutrons on the moon is relatively low. Only VGG16 would be expected to suffer more than one error per day. The ISS orbit is a harsher environment, but even still the expected error rates are low. VGG16 had the highest error rate at 590 errors/day, but ResNet50 and InceptionV3 were both near 100 errors/day and MLPFiveLayers was down at 20 errors/day. This means that these networks could probably survive in LEO with only software error correction without the need for expensive radiation hardened memory. The Martian surface however is very harsh and all the networks would experience more than 10^6 errors/day. As such, the neural networks would require radiation hardened memory to survive. These are important results because certain space missions could take advantage of powerful neural networks and save money by using commercial

memory technologies as opposed to expensive and less capable radiation-hardened parts. Additionally, these experiments showed that neural networks are not too fragile to prevent their use on safety critical systems. A driverless car will not begin misclassifying pedestrians when a rare cosmic ray flips a bit or two in memory.

6.1 Future Work

There are several ways that this work can be expanded. The first would be to test other kinds of neural networks with different architectures to find more correlations between architecture and robustness. One example of a different architecture that should be tested for robustness is a recurrent neural network (RNN). RNNs are commonly used for natural language processing or other tasks with time dependent data, but have a very different architecture than MLPs or CNNs. RNNs are not feedforward neural networks, instead RNNs have loopback connections within the network so they can save states between inputs. This means they could have a very different response to bits being flipped in their trained parameters. Perhaps their performance will degrade faster if a bit flip occurs in a loopback connection, or maybe the loopback connections will act like shortcut connections and increase robustness.

Additionally, there are other kinds of feedforward neural networks to test. MobileNets are a class of CNNs designed to be extremely efficient for use on embedded devices such as cell phones [21]. The cost of this efficiency is a small performance hit, but perhaps MobileNets are also less robust than their larger cousins such as ResNet50 or InceptionV3. This would be a particularly interesting experiment to run because satellites or planetary rovers would probably prefer to use MobileNets as computing power would be constrained on those platforms.

Beyond other types of architectures to be tested, the networks in this thesis could be tested on

a deeper level to determine what makes them robust to bit flips in their trained parameters. For example, the bit flips could be concentrated on specific layers of the networks. With smaller networks like the MLPs it would be relatively easy to test all the layers, but for the large CNNs it would be better to test specific groupings of layers. For example, just flip bits in the last five or first five layers of ResNet50 or InceptionV3. Bit flips could also be concentrated in layers based on what type of layer they are. ResNet50 and InceptionV3 both have convolutional layers and densely connected layers, so concentrating the bit flips in either of those type of layers would reveal which type is more important to the correct functioning of the network. Similar experiments can be done by focusing bit flips in the shortcut connections or exclusively in the batch normalization layers. Concentrating bit flips in certain areas of the neural network can do more than just determine what makes them robust, it can also help uncover how neural networks learn. Understanding the deeper workings of neural networks is a hot research topic, and the techniques used in this thesis to measure robustness could also be used in that work. Lastly, if floating point numbers are still being used, then bit flips could be focused on specific parts of the number such as the exponent or the fraction. This could be compared against the robustness of networks using fixed-point numbers.

The next step for this research would be to design a neural network with robustness in mind. For CNNs this thesis predicted that larger kernels, batch normalization, or shortcut layers could be what increase robustness, while for MLPs added depth increases robustness as well as performance. Additionally, MLPs seemed more robust to bit flips than CNNs, so aspects of MLP architectures could be combined with CNNs to make a network with the robustness of MLPs and the performance of CNNs. The goal is to create a style of neural networks specifically for use in harsh computing environments. Additionally, special kinds of training could be used to try and increase robustness. Changing the feedback loop to

prioritize robustness over accuracy or flipping bits during training might help.

Lastly, the estimated real-world memory error rates should be confirmed. Running a neural network in a radiation testing chamber or launching an experimental CubeSat with modern 16nm SRAM chips are two ways to confirm that neural networks could survive in the space environment.

Bibliography

- [1] An On-device Deep Neural Network for Face Detection - Apple. URL <https://machinelearning.apple.com/2017/11/16/face-detection.html>.
- [2] Measurement and reporting of alpha particle and terrestrial cosmic ray-induced soft errors in semiconductor devices. Technical Report JESD89A, October 2006.
- [3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL <http://tensorflow.org/>. Software available from tensorflow.org.
- [4] Tom Brown, Dandelion Mane, Aurko Roy, Martin Abadi, and Justin Gilmer. Adversarial patch. 2017. URL <https://arxiv.org/pdf/1712.09665.pdf>.
- [5] A. Campbell, P. McDonald, and K. Ray. Single event upset rates in space. *IEEE Transactions on Nuclear Science*, 39(6):1828–1835, December 1992. ISSN 0018-9499. doi: 10.1109/23.211373.
- [6] Y. H. Chen, W. M. Chan, W. C. Wu, H. J. Liao, K. H. Pan, J. J. Liaw, T. H. Chung, Q. Li, C. Y. Lin, M. C. Chiang, S. Y. Wu, and J. Chang. A 16 nm 128 Mb SRAM in

- High- κ Metal-Gate FinFET Technology With Write-Assist Circuitry for Low-VMIN Applications. *IEEE Journal of Solid-State Circuits*, 50(1):170–177, January 2015. ISSN 0018-9200. doi: 10.1109/JSSC.2014.2349977.
- [7] Jin Young Choi and Chong-Ho Choi. Sensitivity analysis of multilayer perceptron with differentiable activation functions. *IEEE Transactions on Neural Networks*, 3(1):101–107, January 1992. ISSN 1045-9227. doi: 10.1109/72.105422.
- [8] François Chollet et al. Keras. <https://github.com/fchollet/keras>, 2015.
- [9] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Juergen Schmidhuber. Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition. *Neural Computation*, 22(12):3207–3220, December 2010. ISSN 0899-7667, 1530-888X. doi: 10.1162/NECO_a_00052. URL <http://arxiv.org/abs/1003.0358>. arXiv: 1003.0358.
- [10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.
- [11] I. Getselev, S. Rumin, N. Sobolevsky, M. Ufimtsev, and M. Podzolko. Absorbed dose of secondary neutrons from galactic cosmic rays inside the international space station. *Advances in Space Research*, 34(6):1429–1432, January 2004. ISSN 02731177. doi: 10.1016/j.asr.2004.04.002. URL <http://linkinghub.elsevier.com/retrieve/pii/S0273117704004612>.
- [12] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Aistats*, volume 15, page 275, 2011.
- [13] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv:1412.6572 [cs, stat]*, December 2014. URL <http://arxiv.org/abs/1412.6572>. arXiv: 1412.6572.

- [14] A. Graves, A. r Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013. doi: 10.1109/ICASSP.2013.6638947.
- [15] S. D. Gribble. Robustness in complex systems. In *Proceedings Eighth Workshop on Hot Topics in Operating Systems*, pages 21–26, May 2001. doi: 10.1109/HOTOS.2001.990056.
- [16] E. Guizzo and E. Ackerman. The hard lessons of DARPA’s robotics challenge [News]. *IEEE Spectrum*, 52(8):11–13, August 2015. ISSN 0018-9235. doi: 10.1109/MSPEC.2015.7164385.
- [17] Bob Hammell. Ships in Satellite Imagery, September 2017. URL <https://www.kaggle.com/rhammell/ships-in-satellite-imagery>.
- [18] S. C. Hauser, W. C. Headley, and A. J. Michaels. Signal detection effects on deep neural networks utilizing raw IQ for modulation classification. In *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, pages 121–127, October 2017. doi: 10.1109/MILCOM.2017.8170853.
- [19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. URL <http://arxiv.org/abs/1512.03385>. arXiv: 1512.03385.
- [20] W. He, Y. Chen, and Z. Yin. Adaptive Neural Network Control of an Uncertain Robot With Full-State Constraints. *IEEE Transactions on Cybernetics*, 46(3):620–629, March 2016. ISSN 2168-2267. doi: 10.1109/TCYB.2015.2411285.
- [21] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convo-

- lutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL <http://arxiv.org/abs/1704.04861>.
- [22] Andy A. Hwang, Ioan A. Stefanovici, and Bianca Schroeder. Cosmic Rays Don’T Strike Twice: Understanding the Nature of DRAM Errors and the Implications for System Design. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pages 111–122, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-0759-8. doi: 10.1145/2150976.2150989. URL <http://doi.acm.org/10.1145/2150976.2150989>.
- [23] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*, February 2015. URL <http://arxiv.org/abs/1502.03167>. arXiv: 1502.03167.
- [24] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 361–372, June 2014. doi: 10.1109/ISCA.2014.6853210.
- [25] Hiroaki Kitano. Biological robustness. *Nature Reviews Genetics*, 5(11):826–837, November 2004. ISSN 1471-0064. doi: 10.1038/nrg1471. URL <https://www.nature.com/articles/nrg1471>.
- [26] H. Koshiishi, H. Matsumoto, A. Chishiki, T. Goka, and T. Omodaka. Evaluation of the neutron radiation environment inside the International Space Station based on the Bonner Ball Neutron Detector experiment. *Radiation Measurements*, 42(9):1510–1520, October 2007. ISSN 13504487. doi: 10.1016/j.radmeas.2007.02.072. URL <http://linkinghub.elsevier.com/retrieve/pii/S1350448707001679>.

- [27] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.
- [28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [29] S. Kwon, K. Lee, Y. Kim, K. Kim, C. Lee, and W. W. Ro. Measuring error-tolerance in SRAM architecture on hardware accelerated neural network. In *2016 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, pages 1–4, October 2016. doi: 10.1109/ICCE-Asia.2016.7804818.
- [30] Khler J., Zeitlin C., Ehresmann B., WimmerSchweingruber R. F., Hassler D. M., Reitz G., Brinza D. E., Weigle G., Appel J., Bttcher S., Bhm E., Burmeister S., Guo J., Martin C., Posner A., Rafkin S., and Kortmann O. Measurements of the neutron spectrum on the Martian surface with MSL/RAD. *Journal of Geophysical Research: Planets*, 119(3):594–603, February 2014. ISSN 2169-9097. doi: 10.1002/2013JE004539. URL <https://agupubs.onlinelibrary.wiley.com/doi/10.1002/2013JE004539>.
- [31] Nicholas D. Lane and Petko Georgiev. Can Deep Learning Revolutionize Mobile Sensing? In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications, HotMobile '15*, pages 117–122, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3391-7. doi: 10.1145/2699343.2699349. URL <http://doi.acm.org/10.1145/2699343.2699349>.
- [32] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio. Batch normalized recurrent neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2657–2661, March 2016. doi: 10.1109/ICASSP.2016.7472159.

- [33] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.
- [34] R. E. Lingenfelter, E. H. Canfield, and W. N. Hess. The lunar neutron flux. *Journal of Geophysical Research*, 66(9):2665–2671, September 1961. ISSN 01480227. doi: 10.1029/JZ066i009p02665. URL <http://doi.wiley.com/10.1029/JZ066i009p02665>.
- [35] R. E. LINGENFELTER, E. H. CANFIELD, and V. E. HAMPEL. The lunar neutron flux revisited. *Earth and Planetary Science Letters*, 16(3):355–369, November 1972. ISSN 0012-821X. doi: 10.1016/0012-821X(72)90153-7. URL <https://www.sciencedirect.com/science/article/pii/0012821X72901537>.
- [36] J. Meza, Q. Wu, S. Kumar, and O. Mutlu. Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 415–426, June 2015. doi: 10.1109/DSN.2015.57.
- [37] E. M. El Mhamdi and R. Guerraoui. When Neurons Fail. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1028–1037, May 2017. doi: 10.1109/IPDPS.2017.66.
- [38] K. Nagatani, S. Kiribayashi, Y. Okada, K. Otake, K. Yoshida, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, M. Fukushima, and S. Kawatsuma. Gamma-ray irradiation test of electric components of rescue mobile robot Quince. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 56–60, November 2011. doi: 10.1109/SSRR.2011.6106756.
- [39] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip feng shui: Hammering a needle in the software stack. In *25th USENIX*

- Security Symposium (USENIX Security 16)*, pages 1–18, Austin, TX, 2016. USENIX Association. ISBN 978-1-931971-32-4. URL <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/razavi>.
- [40] M. Reid and G. Ottman. Software controlled memory scrubbing for the Van Allen Probes Solid State Recorder (SSR) memory. In *2014 IEEE Aerospace Conference*, pages 1–6, March 2014. doi: 10.1109/AERO.2014.6836406.
- [41] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- [42] Mark Seaborn and Thomas Dullien. Exploiting the dram rowhammer bug to gain kernel privileges. March 2015.
- [43] C. H. Sequin and R. D. Clay. Fault tolerance in artificial neural networks. In *1990 IJCNN International Joint Conference on Neural Networks*, pages 703–708 vol.1, June 1990. doi: 10.1109/IJCNN.1990.137651.
- [44] F. W. Sexton. Destructive single-event effects in semiconductor devices and ICs. *IEEE Transactions on Nuclear Science*, 50(3):603–621, June 2003. ISSN 0018-9499. doi: 10.1109/TNS.2003.813137.
- [45] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL <http://arxiv.org/abs/1409.1556>.
- [46] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*, September 2014. URL <http://arxiv.org/abs/1409.1556>. arXiv: 1409.1556.

- [47] C. Slayman. Soft error trends and mitigation techniques in memory devices. In *2011 Proceedings - Annual Reliability and Maintainability Symposium*, pages 1–5, January 2011. doi: 10.1109/RAMS.2011.5754515.
- [48] V. Sridharan and D. Liberty. A study of DRAM failures in the field. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–11, November 2012. doi: 10.1109/SC.2012.13.
- [49] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014. ISSN 1532-4435. URL <http://dl.acm.org/citation.cfm?id=2627435.2670313>.
- [50] E. G. Stassinopoulos and J. P. Raymond. The space radiation environment for electronics. *Proceedings of the IEEE*, 76(11):1423–1442, November 1988. ISSN 0018-9219. doi: 10.1109/5.90113.
- [51] M. Stevenson, R. Winter, and B. Widrow. Sensitivity of feedforward neural networks to weight errors. *IEEE Transactions on Neural Networks*, 1(1):71–80, March 1990. ISSN 1045-9227. doi: 10.1109/72.80206.
- [52] M. Szarvas, A. Yoshizawa, M. Yamamoto, and J. Ogata. Pedestrian detection with convolutional neural networks. In *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, pages 224–229, June 2005. doi: 10.1109/IVS.2005.1505106.
- [53] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015. doi: 10.1109/CVPR.2015.7298594.

- [54] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, June 2016. doi: 10.1109/CVPR.2016.308.
- [55] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv:1312.6199 [cs]*, December 2013. URL <http://arxiv.org/abs/1312.6199>. arXiv: 1312.6199.
- [56] Genichi Taguchi, Don Clausing, et al. Robust quality. *Harvard Business Review*, 68(1): 65–75, 1990.
- [57] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, June 2014. doi: 10.1109/CVPR.2014.220.
- [58] R. Velazco, A. Assoum, N. E. Radi, R. Ecoffet, and X. Botey. SEU fault tolerance in artificial neural networks. *IEEE Transactions on Nuclear Science*, 42(6):1856–1862, December 1995. ISSN 0018-9499. doi: 10.1109/23.489227.
- [59] R. Velazco, A. Assoum, P. Cheynet, M. Olmos, and R. Ecoffet. SEU experiments on an Artificial Neural Network implemented by means of digital processors. *IEEE Transactions on Nuclear Science*, 43(6):2889–2896, December 1996. ISSN 0018-9499. doi: 10.1109/23.556882.
- [60] R. Velazco, P. Cheynet, J. D. Muller, R. Ecoffet, and S. Buchner. Artificial neural network robustness for on-board satellite image processing: results of upset simulations and ground tests. *IEEE Transactions on Nuclear Science*, 44(6):2337–2344, December 1997. ISSN 0018-9499. doi: 10.1109/23.659057.

- [61] R. Velazco, P. Cheynet, A. Tissot, J. Haussy, J. Lambert, and R. Ecoffet. Evidences of SEU tolerance for digital implementations of artificial neural networks: one year MPTB flight results. In *1999 Fifth European Conference on Radiation and Its Effects on Components and Systems. RADECS 99 (Cat. No.99TH8471)*, pages 565–568, 1999. doi: 10.1109/RADECS.1999.858648.
- [62] J.W. Wilson, M.S. Cloudsley, F.A. Cucinotta, R.K. Tripathi, J.E. Nealy, and G. De Angelis. Deep space environments for human exploration. *Advances in Space Research*, 34(6):1281–1287, January 2004. ISSN 02731177. doi: 10.1016/j.asr.2003.10.052. URL <http://linkinghub.elsevier.com/retrieve/pii/S0273117704002030>.
- [63] L. J. Wong, W. C. Headley, and A. J. Michaels. Estimation of transmitter I/Q imbalance using convolutional neural networks. In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 948–955, January 2018. doi: 10.1109/CCWC.2018.8301715.