THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF MECHANICAL ENGINEERING

Categorizing Synthetic Aperture Sonar Imagery Using Pre-Trained Neural Networks and Artificial Data

RAIID AHMED
SPRING 2021

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Mechanical Engineering
with honors in Mechanical Engineering

Reviewed and approved* by the following:

Daniel C. Brown
Assistant Research Professor of Acoustics
Thesis Supervisor

Bo Cheng
Associate Professor of Mechanical Engineering
Honors Adviser

* Electronic approvals are on file.

ABSTRACT

In the past 10 years, advancements in graphical and cloud computing have made it possible to conduct a wide variety of experiments applying machine learning tools to real-world physical problems. In this thesis, we explore the effectiveness of applying these tools to an image classification problem dealing with SAS (Synthetic Aperture Sonar) imagery. Lack of reference SAS imagery means that training an image recognition algorithm would require creating a training dataset from the ground up. SAS imagery is computationally expensive to create which makes training an image classification algorithm non-trivial. Therefore, in this experiment we will leverage the use of artificially generated images from MATLAB to train our image classification algorithm. This algorithm will be validated on 3D-printed versions of the artificially generated images. The MNIST dataset will be used as a basis for both the set of artificially generated images and the set of 3D-printed models. The Fast.ai library will be used as the source for our image classification models. First returns of data show the classifier is able to categorize MATLAB generated training set at an accuracy rate of 99.5%. However, results are inconclusive for SAS data. Possible continuations of this study would explore the possibility of using numerical data rather than images, or categorizing scans based on material properties (ex: density).

TABLE OF CONTENTS

LIST OF FIGURES iii

LIST OF TABLES iv

ACKNOWLEDGEMENTS v

Chapter 1 Introduction 1

Chapter 2 Background and Literature Review 5

2.1 Methods of Image Categorization 5
2.2 Overview of Neural Networks 7
2.3 ImageNet Database 12
2.4 Fast.ai Overview 14
2.5 The MNIST Database 16
2.6 Synthetic Aperture Sonar 17
2.7 Similar Works 20

Chapter 3 Motivation and Goals 23

3.1 Sound Hunter 23
3.2 Goals and Parameters 25

Chapter 4 Data Methodology and Tool Design 28

4.1 Forms of Data 28
4.2 Creating Test Models 29
4.2 Creating Training Imagery 33
4.4 Image Classifier 35

Chapter 5 Experimental Procedure 40

5.1 Timeline 40
5.2 Procedure 42

Chapter 6 Results and Analysis 44

Chapter 7 Conclusion and Future Steps 48

BIBLIOGRAPHY 50

LIST OF FIGURES

Figure 1. A representative model of a single function in a neural network adapted from Roos *et al.* [10] 6

Figure 2. A representative model of a neural network with multiple input and output units adapted from Fausett [3] 7

Figure 3. A representative model for a backpropagation network with one hidden layer adapted from Fausett [3] 8

Figure 4. An example function to be approximated by a backpropagation network 9

Figure 5. The plot generated from the equation in Figure 4 10

Figure 6. The plot generated by a backpropagation network after 1000 epochs 10

Figure 7. The plot generated by a backpropagation network after 10000 epochs 11

Figure 8. Detected bounding boxes around objects shown in the ImageNet dataset adapted from Deng *et al.* [5] 12

Figure 9. Examples of histopathology scans used in the dataset adapted from Kieffer *et al.* [12] 13

Figure 10. Command Hierarchy for the Fast.ai library adapted from Howard *et al.* [13] 14

Figure 11. Example code for setting up an image classifying model in Fast.ai adapted from Howard *et al.* [13] 15

Figure 12. Examples of skin lesions and their types adapted from Reddy [14] 15

Figure 13. A set of example imagery contained within the MNIST dataset adapted from Deng [15] 16

Figure 14. The three modalities used for collecting SAS imagery adapted from Blanford *et al.* [7]17

Figure 15. The progenitor to the AirSAS setup used for this project adapted from Blanford et al. Top is the actual setup, while bottom is the setup diagram [7] 18

Figure 16. Acoustic Time Series for a 30 second clip [4] 20

Figure 17. Spectrogram for a 30 second clip of music [4] 21

Figure 18. Example of buried object detection in a shallow lakebed adapted from Brown [19] 23

Figure 19. Plot showing training loss, validation loss, and accuracy for a neural network [4]26

Figure 20. The Delta Rule adapted from Fausett [3] 26

Figure 21. Setting initial parameters for the readMNIST command 29

Figure 22. The main loop of MNIST2STL 30

Figure 23. MNIST2STL surface plot output 31

Figure 24. Models in slicing software 31

Figure 25. Response from various 3D printed specimens 32

Figure 26. For loop used to set height of each pixel in training images 33

Figure 27. Example image for the number 2 used for training 34

Figure 28. Initializing parameters and libraries in the image classifier 36

Figure 29. Folder structure of Jupyter Notebook 37

Figure 30. Code cells for creating learner and data objects 38

Figure 31. Example batch from the input dataset 38

Figure 32. Confusion matrix example for training data 39

Figure 33. Test set prediction code cell 39

Figure 34. The number 5 as scanned by the AIRSAS (top) and generated by MATLAB (bottom)41

Figure 35. Response of training loss, accuracy, and validation loss over a 10-epoch training period 44

Figure 36. Response of training loss, accuracy, and validation loss over a 100-epoch training period 45

Figure 37. Confusion matrices detailing correct and incorrect predictions for 10 epochs (left) and 100 epochs (right) 46

Figure 38. Example tensor classifying 0 as 8 from a 5-epoch trained network 47

LIST OF TABLES

Table 1. Results from testing deep vs. shallow networks on 30 second clips [4] 21

Table 2. Observed accuracy values in response to training duration 46

ACKNOWLEDGEMENTS

First off, I would like to thank my advisors Dr. Brown and Dr. Blanford for assisting me every step of the way during this process. From our initial meeting in the summer of 2019 during my internship throughout the COVID-19 pandemic they have had answers and guidance for every single one of my questions.

Furthermore, I would like to thank Dr. Bo Cheng for advising me through the highs and lows of my Mechanical Engineering study with the Schreyer Honors college.

Finally, I would like to thank my family for supporting me 24/7 without pause for the past 21 years.

Chapter 1

Introduction

Recent advancements in computer vision technology have made the implementation of previously theoretical algorithms possible for anyone with a laptop and an internet connection. These advancements have given rise to the explosion of popularity enjoyed by the field of machine learning and data science as a whole. Previously impossible problems like classifying images of animals by breed have been made trivial [1]. While the animal/breed example is commonplace now as an introduction to implementing classification algorithms, it is far from the only unique example. Classification algorithms have been applied to a multitude of different areas, from birdsong syllable detection to gesture and language recognition [2].

Classification algorithms operate on sets of neural networks. Neural networks are algorithms based on mathematical representations of biological neurons. They operate based on previous assumptions from input data sets. Mainly, networks are first presented with labeled data called a “training set”. Random assumptions are made on parameters such as data types and what the data may represent. These random assumptions are compared with the labels and recorded. The network then adjusts parameters based on the agreement between the predictions and the verified labels. Using these parameters, a network can then make predictions on new, uncategorized datasets usually referred to as “validation sets” [3]. This method of categorization is useful in areas of study where classifying a large, unlabeled dataset is time-consuming for a user. For example, there are many cases where a user would like to identify a song playing on the radio. Previously, this task would have required many Google searches and vague descriptions to possibly identify a song title. Now, it is possible for a novice programmer to use neural networks by downloading an image classifying model. Then, the user could use a Short Time Fourier Transform to convert the audio into a spectrogram which can be used in an image classifying model. Finally, the user can use the pre-trained model on the spectrogram to determine subjective properties like genre, and objective properties like song title [4].

While neural networks can be used to great extent in classifying audio samples of popular music, there are many cases where a large labeled dataset does not exist. In this case, it would be necessary to produce this input dataset. This option is less attractive when real cost factors come into play (ex: time, computing resources). Therefore, it can be necessary to use other strategies when creating an image classification model. With the existence of astronomically large datasets, it is possible to generalize a network to fit any type of problem [5]. These generalized, pre-trained models are widely available but are not suitable for deeply specialized problems. They function well in detecting and classifying common features such as animals and humans, but fail to do the same on esoteric datasets. Because of this, generalized networks cannot be applied to all problems. However, it is possible to artificially create data based on predictions of what types of data the network may encounter. Artificially generated data is particularly useful for tasks where it is only necessary to identify a set of predetermined patterns. This strategy also has limitations in how the generated data must closely resemble the field data to an accurate degree.

Synthetic Aperture Sonar is a method to generate high quality acoustic imagery for use in underwater applications [6]. A benchtop system called the AirSAS at the Applied Research Laboratory at the Pennsylvania State University (ARL/PSU) makes it possible to acquire this data and observe acoustic responses on dry land. The system has been built using off the shelf components and is a valuable tool for instruction. This method cuts out many issues with acquiring underwater imagery including but not limited to: non-stationary conditions, high financial costs, and unwanted platform movement [7].

Researchers have expressed interest in applying an image classification algorithm based on neural networks to automatically classify object based on properties such as material properties and object geometry. The lack of a large set of existing input data for potential objects makes this implementation difficult, and collecting this set would require a large amount of time and resources making the effort costly. Due to many different teams running projects on this system, collecting continuous data on the scale necessary for building a training set is virtually impossible. This problem represents one of the main issues with using neural networks to for real world tasks. Since image classification algorithms are dependent on large, digital, existing datasets, they normally cannot be applied to specialized problems without data augmentation. In this paper, we will explore the possibility of procedurally generating an input dataset for the classification of scans obtained through Synthetic Aperture Sonar.

The dataset that was used for generating our input dataset and physical specimens will be the MNIST dataset of handwritten letters/numbers. These letters were observed in to two mediums: optical and physical. Optical representations were generated using surface plots in MATLAB. These surface plots were then used to generate .STL files for 3D printing. The training set was allocated with the visual, artificially generated scans from MATLAB while the validation set was built using AirSAS scans of the 3D printed models. By examining the accuracy of the network when classifying these scans of 3D printed models, we determined how effective this method would be when applied to normally encountered underwater specimens. The Fast.ai library was used as the framework for designing and evaluating our responses in this experiment. Fast.ai was chosen specifically for its simplicity. In addition, Fast.ai has many tools available for visually representing classification results. In this experiment, we varied several parameters such as learning rate, training cycles, and batch size. In addition, we also iterated on the process of creating effective 3D models for use with the AirSAS. Our final results determined that the network performs very well on the MATLAB generated data, but yields inconclusive results on the AirSAS scans.

Chapter 2

Background and Literature Review

2.1 Methods of Image Categorization

The problem of image categorization has been revolutionized in recent years with developments in graphical computing hardware. A 2012 study by Parkhi *et al.* [1] investigating image categorization methods for animal breeds reached an accuracy level of about 59%. This result was considered a remarkable achievement at the time, as contemporary methods were not able to reach similar accuracy levels for the breed classification problem. Just five years later, Hu *et al.* [8] solved a much larger image classification problem using 1000 categories with an accuracy level of 97.749%. This dramatic leap in effectiveness for image classification is the result of a revolution in graphical computing beginning with the implementation of Deep Neural Networks (DNN) for image classification on Graphical Processing Units (GPU) created by Ciregan *et al.* [9]. This approach used a hierarchal system similar to neurons and synapses in the brain. Images are separated into small layers, which are passed to individual functions with random initial weights and verified using an indexed control set. Figure 1 illustrates how these weights would be arranged with a function to reach an output.

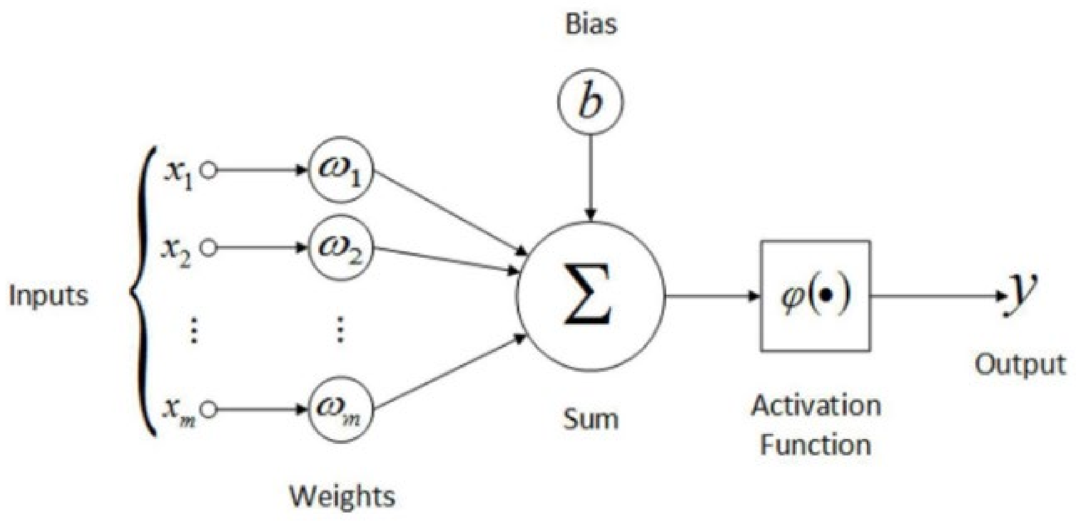


Figure 1. A representative model of a single function in a neural network adapted from Roos *et al.* [10]

The organization of the diagram is meant to simulate the function of a biological neuron. At the far left, inputs are received as integers representing some information about an image (color, pixel location, etc.). These inputs are multiplied by pre-determined weights relevant to the type of input that is being received. The weighted inputs are then summed together. A bias term is added to this sum, modeling the sensitivity of the neuron. Finally, an activation function is used to decide how relevant the inputs are to the classification of the image. The weights and bias are adjusted based on the accuracy of the classification in a process known as “training”. Each cycle of presenting inputs, obtaining outputs, and adjusting weights is referred to as an “epoch” [3]. The next section will go over a deeper explanation on how neural networks are constructed.

2.2 Overview of Neural Networks

Traditional, logic-based computing normally excels in a wide variety of situations but lacks in cases where more "human" behavior is expected. For example, a child can recognize the letter A from the letter B and distinguish a cat from a bird. This association is not deduced using laws and theorems - rather, it is learned by a teacher telling the child that a picture of a cat is called and the picture of the bird is called a bird. Neural networks take this approach and extrapolate it to a form that can be processed on a computer. Neural networks provide an alternative form of problem solving from conventional programming. This solution makes problems like pattern classification, natural language processing, and speech recognition possible.

Figure 1 represents the construction of a single neuron. Neural networks in practice utilize this model in quantities on the order of thousands or even greater. These neurons are normally presented in "layers" of neurons that are interconnected. The architecture of these networks varies based on the type of problem and form of input data. Figure 2 illustrates a possible organization of these layers [3].

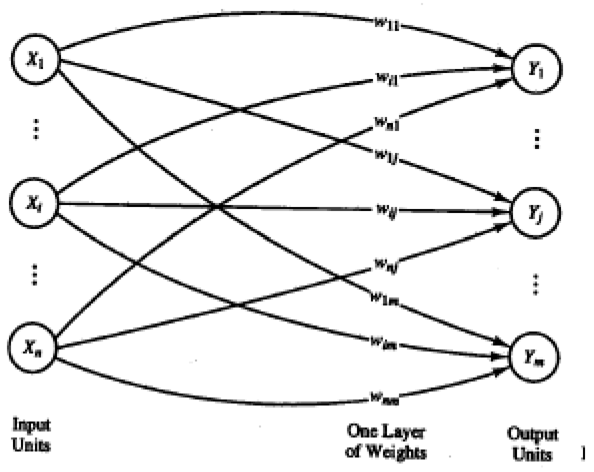


Figure 2. A representative model of a neural network with multiple input and output units adapted from Fausett [3]

Figure 2 is a single layer net with one layer of connection weights. There is a finite number of possible outputs and these outputs do not influence each other's respective weights. More complicated problems require the use of multilayer networks. Figure 3 illustrates a possible representation of one of these networks.

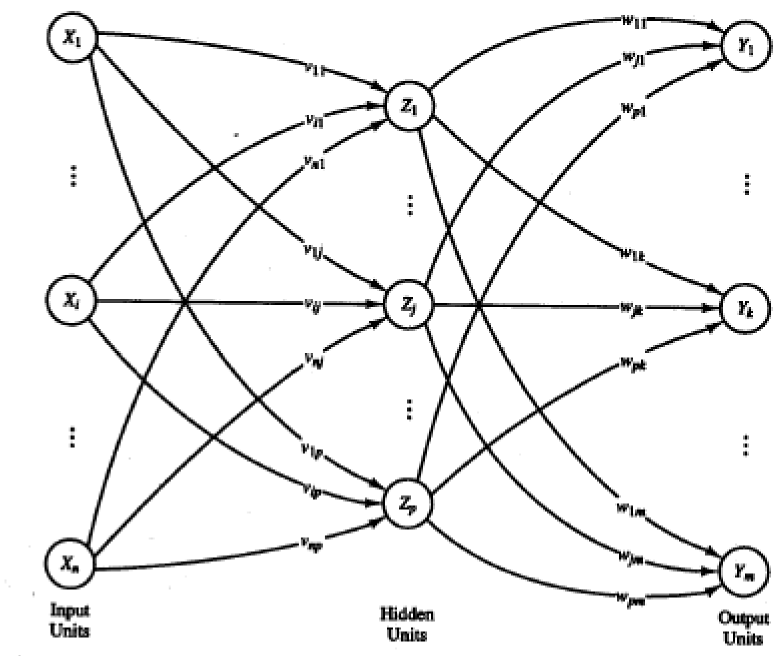


Figure 1.5 A multilayer neural net.

Figure 3. A representative model for a backpropagation network with one hidden layer adapted from Fausett [3]

These networks allow for more complex problems to be solved with the use of several input/output layers that can extract and represent features contained deeply within the input data. These connections mimic the associations created in human learning, i.e., based on association versus underlying laws [3]. Multiple input/output units are linked together with weights, allowing for the extraction of features not visible in single layer networks. This style of network is called a Deep Neural Network (DNN), and the use of these networks is referred to as deep learning [9]. This paper relies heavily on the associations built in deep learning due to the lack of complete training data. Backpropagation must be implemented in order for networks of this type to create nonlinear associations and work with input data such as handwritten text. Training by backpropagation involves four stages: forward inputs of the input training pattern, calculation of errors between the generated and desired output, backpropagation of the associated error, and the adjustment of all weights. By calculating and training based on the error through all layers of the network, it is possible for a network to learn any continuous mapping to an arbitrary accuracy [3]. For example, Figure 4 shows an equation that can generate a relatively complex surface plot using two sine functions. Figure 5 shows the plot normally generated from this equation, while Figure 6 shows the plot generated by a multilayer network with backpropagation implemented after 1000 epochs of training. Figure 7 shows the plot generated by the network after 10000 epochs of training, clearly showing an increase in overall accuracy from the additional epochs. The accuracy was quantified by calculating the Euclidean distance between the real and predicted vectors. For 1000 epochs, this value was 3.0553. After training for 10000 epochs, this value was reduced to .74405. Training data was produced by randomly generating a set of 20 pairs of x1 and x2 pairs. The corresponding y values of these points were then calculated and presented in a random order to the network.

$$\mathbf{y = \sin(2\pi x_1)\sin(2\pi x_2)}$$

Figure 4. An example function to be approximated by a backpropagation network

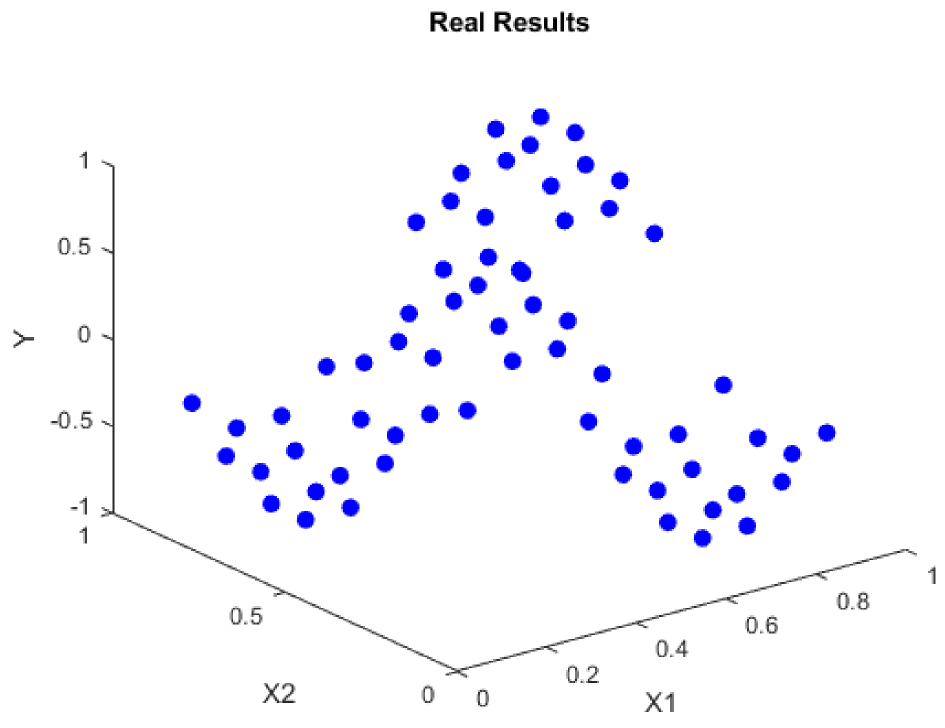


Figure 5. The plot generated from the equation in Figure 4

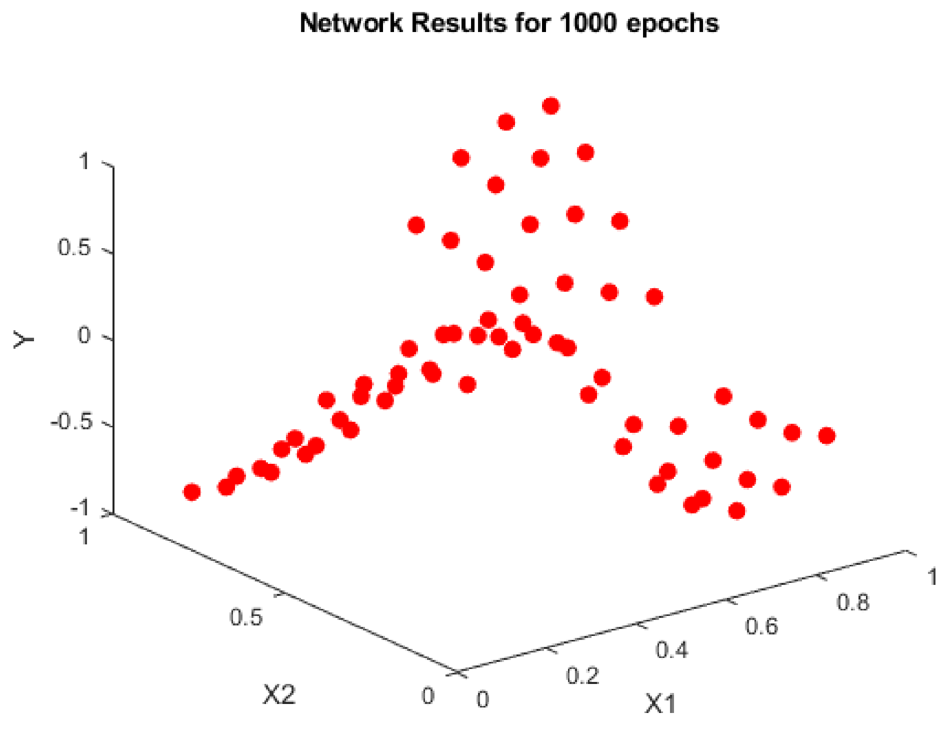


Figure 6. The plot generated by a backpropagation network after 1000 epochs

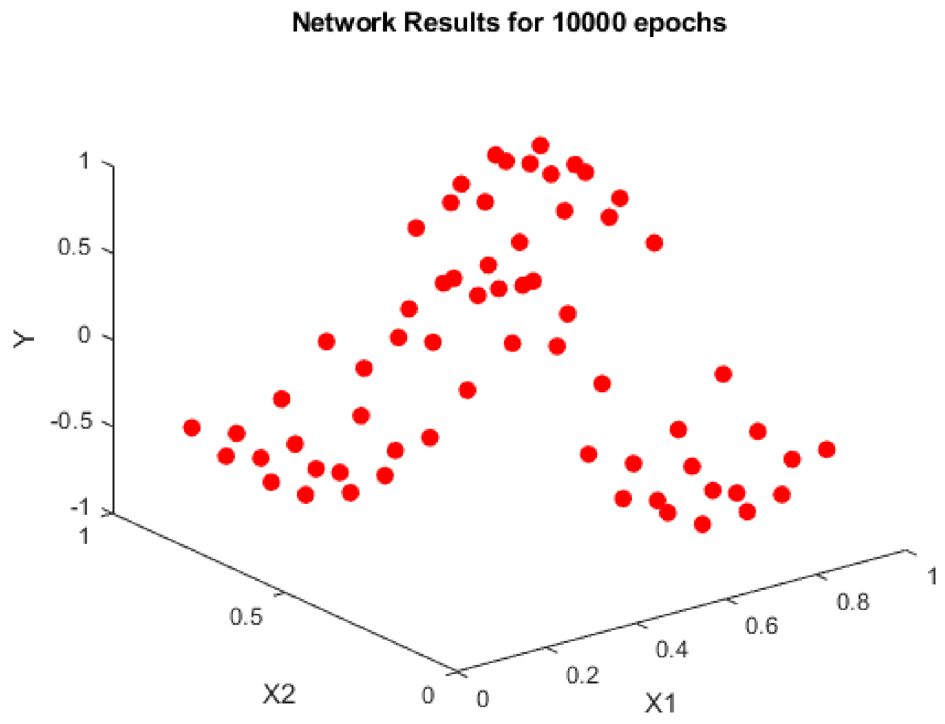


Figure 7. The plot generated by a backpropagation network after 10000 epochs

For ideal situations like the one previously mentioned with the sinusoidal function, it is possible to acquire a complete input set in a short period of time. This is not the case with many real-world applications however, as neural networks are widely used to approximate situations with missing data. A study completed by Smieja *et al.* [11] explores a mechanism for these types of problems. In the paper, a method using probability density functions is used to approximate uncertainty on missing information. Using this data, it is possible to generalize each neuron's response based on the uncertainty and applying expected values to the network. From a theoretical perspective, this approach has been shown to give comparable results to methods using complete data. Another approach using this methodology is illustrated in the next section, overviewing how large datasets can be used to generalize a network for any application.

2.3 ImageNet Database

An incredibly large dataset would be necessary to generalize DNNs for a wide variety of image classification applications. The ImageNet database created by Deng *et al.* [5] became the standard for this need, emerging as an indexed database of millions of categorized and annotated images. Users around the world were able to participate in this study, individually labeling 3.2 million images in order to create a multipurpose image dataset. Thanks to this collection of images, it has been possible to adapt the use of DNNs for a wide variety of vision-based tasks such as object recognition, image classification, and object localization. Figure 8 shows how the ImageNet dataset was used to draw bounding boxes around recognizable objects in various pictures. A single network can be generalized to recognize these specific features from a wide range of categories mainly because ImageNet is so large. ImageNet does not specialize in any one direction, which makes it an applicable dataset to almost all vision tasks.

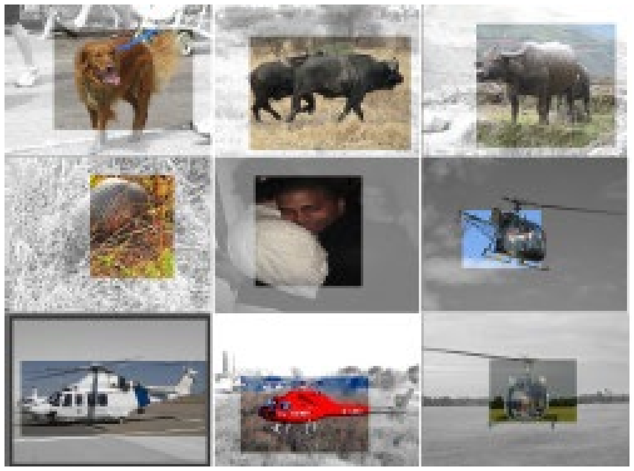


Figure 8. Detected bounding boxes around objects shown in the ImageNet dataset adapted from Deng *et al.* [5]

The method presented by Kieffer *et al.* [12] is an example of how a generalized input set can be used for specialized applications, showing the possible application of a DNN pre-trained on the ImageNet database in the application of categorizing medical imagery. This study utilizes an image classifier designed around the ImageNet database to categorize a collection of 27,055 histopathology scans in 24 categories. As mentioned in section 1.2, multilayer networks were used to extract deep features within the scans. While it is borderline impossible to observe these deep features with the naked eye, they were used in the project to categorize the input patches. Figure 9 shows an example of these input patches, illustrating the complexity of the present imagery.

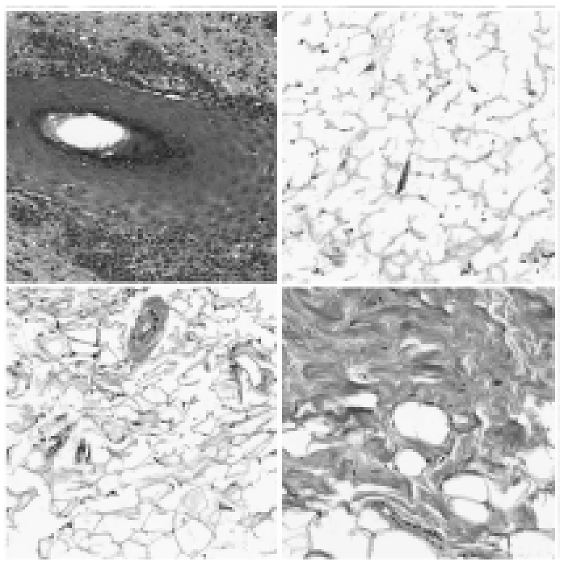


Figure 9. Examples of histopathology scans used in the dataset adapted from Kieffer *et al.* [12]

The lack of existing annotated datasets for histopathology scans meant that an existing dataset like ImageNet would have to be used as a basis for training. This classifying method was able to compete with state-of-the-art methods reported in the literature for the paper. After fine tuning parameters, the network was able to achieve an accuracy of 76.10% using whole scans [12]. ImageNet does not contain any specific labeled histopathology images within the data set. Despite this setback, the classifier performed reasonably well. This shows that a network pre-trained on a general dataset like ImageNet can still be used in specific problems after fine tuning.

2.4 Fast.ai Overview

The Fast.ai library for Python, a general-use programming language, was originally created with the goal of democratizing the use of neural networks for data classification in all areas of research. It was created with a computer vision library allowing algorithms to be implement in 4-5 lines of code. Furthermore, the reporting system allows for generation of easy-to-read diagrams at any stage of learning. Fast.ai is built as a high-level API on top of several lower-level functions within the Python language [13]. Figure 10 illustrates the breakdown of the Fast.ai library.

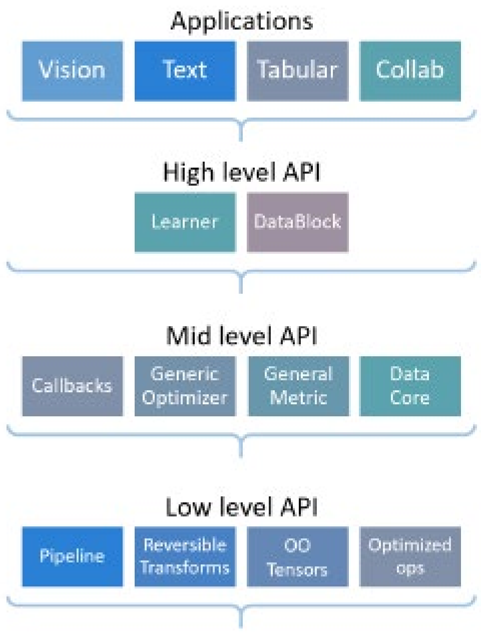


Figure 10. Command Hierarchy for the Fast.ai library adapted from Howard *et al.* [13]

This library was created with the intention of use by practitioners applying existing deep learning methods, making it a suitable choice considering my background in mechanical engineering. Fast.ai also offers access to lower-level parameters, allowing us to view the effect of testing multiple layer types. Figure 11 illustrates example code for creating and training a vision learning model in Fast.ai. While the case of vision models is applied in this behavior, Fast.ai allows for the categorization of text and tabular data as well.

```
from fastai.vision.all import *
path = untar_data(URLs.PETS)
dls = ImageDataLoaders.from_name_re(path=path, bs=64,
    fnames = get_image_files(path/"images"), pat = r'/([^/]+)_\d+.jpg$',
    item_tfms=RandomResizedCrop(450, min_scale=0.75),
    batch_tfms=[*aug_transforms(size=224, max_warp=0.), Normalize.from_stats(*imagenet_stats)])
learn = cnn_learner(dls, resnet34, metrics=error_rate)
learn.fit_one_cycle(4)
```

Figure 11. Example code for setting up an image classifying model in Fast.ai adapted from Howard *et al.* [13]

Each line of code is meant to perform a single task in setting up the model. According to a survey referenced in the Fast.ai publication, 10% of researches in the Kaggle community make use of Fast.ai [13]. One example of the Fast.ai library being applied to a domain outside of data science is shown in the study by Reddy [14]. In this project, the Fast.ai vision model is used to classify dermoscopy images of skin lesions into 7 distinct disease categories. This classification study was conducted with the goal of eventually being able to recognize the early signs of skin lesions. Using the Fast.ai model, accuracy with the input set reached 91.0%. This was achievable after only 19 epochs of training, showing that a relatively low-cost method for classifying skin lesions is possible. Figure 12 shows an example of a lesion from each possible category.

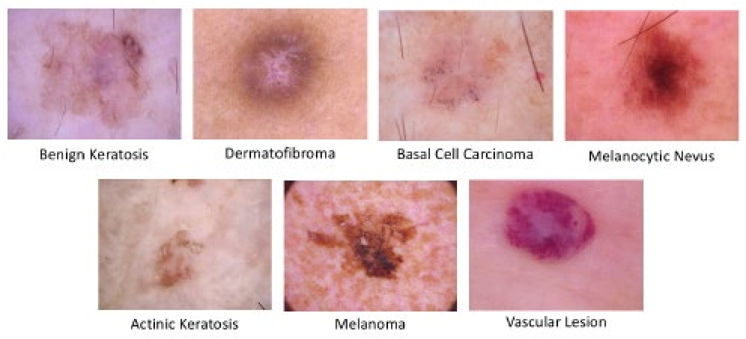


Figure 12. Examples of skin lesions and their types adapted from Reddy [14]

2.5 The MNIST Database

One of the main issues looked at in the areas of pattern recognition is the classification of handwritten digits. The MNIST database is one of the freely available standard collections available for study. MNIST is particularly useful for a basis in this study, as it is focused on simpler recognition tasks. The set consists of 60,000 training images and 10,000 test images. The digits are presented in black and white in a fixed size of 28x28 pixel images. The nature of this standardized set lends itself to be applicable in simple projects that require real world data, but with time constraints that make collecting said data impossible. Normal applications of this set yield accuracy rates around 99.73% [15]. Figure 13 shows an example of some of the specimens from this dataset.

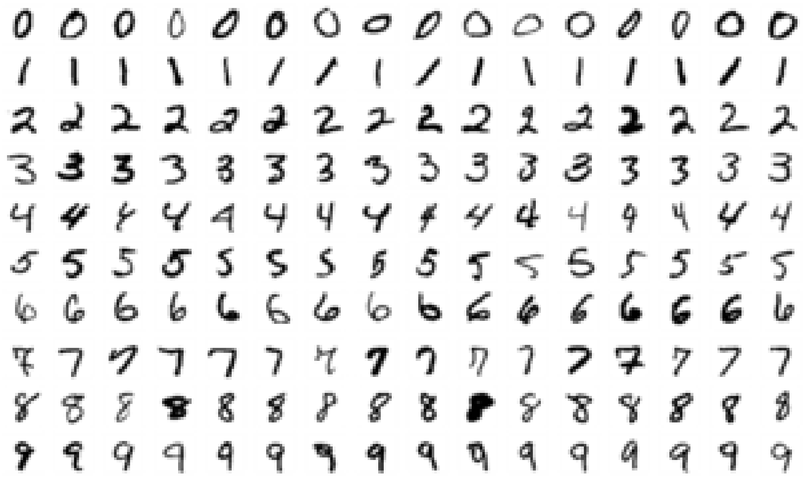


Figure 13. A set of example imagery contained within the MNIST dataset adapted from Deng [15]

The next section will provide a brief description of the acoustic methods used in this project.

2.6 Synthetic Aperture Sonar

SAS is an underwater acoustic technique for generation of high-quality imagery through the combination of sonar returns from a subject. SAS on the relative motion between a sonar device and the subject to form a synthetic response. Several modalities can be used to collect this imagery. These options are represented in Figure 14.

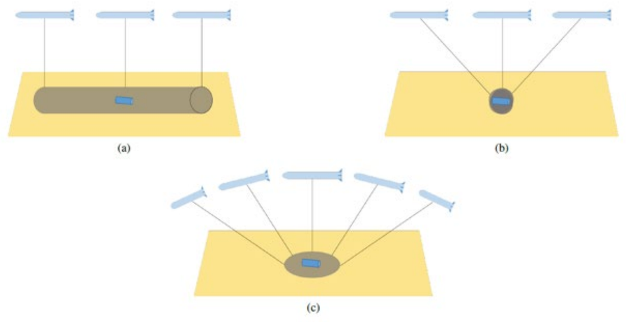
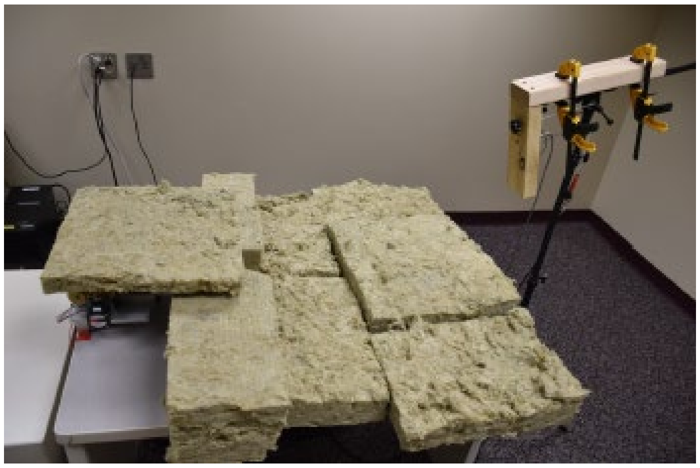


Figure 14. The three modalities used for collecting SAS imagery adapted from Blanford *et al.* [7]

During the summer of 2019 in the Applied Research Laboratory, Blanford *et al.* [7] designed a system that would be able to collect acoustic data using this technique aboveground, eliminating many of the challenges associated with collecting traditional underwater sonar responses. Recently, the need to generate an indexed database of imagery collected by the AirSAS has risen due to variety of projects conducted on the platform. While the cost of the AirSAS is much less than the cost associated with collecting underwater SAS imagery, the sheer volume of scans needed to build an indexed database is impractical with the current schedule. It took the AirSAS a few minutes to scan each object for the test set. Similar projects in the field require datasets orders of magnitude larger than the one used in this paper. With the enormous demands placed on SAS systems in the field, machines are often booked out for weeks at a time. Therefore, scheduling the data collection for even only the 50 objects in this project is a non-trivial task. The setup used for this project is shown in Figure 15. The AirSAS is made from a loudspeaker, an amplifier, a microphone/preamplifier, a sound card, a microcontroller, and a stepper motor. In addition, the Data Acquisition System is run on a computer.



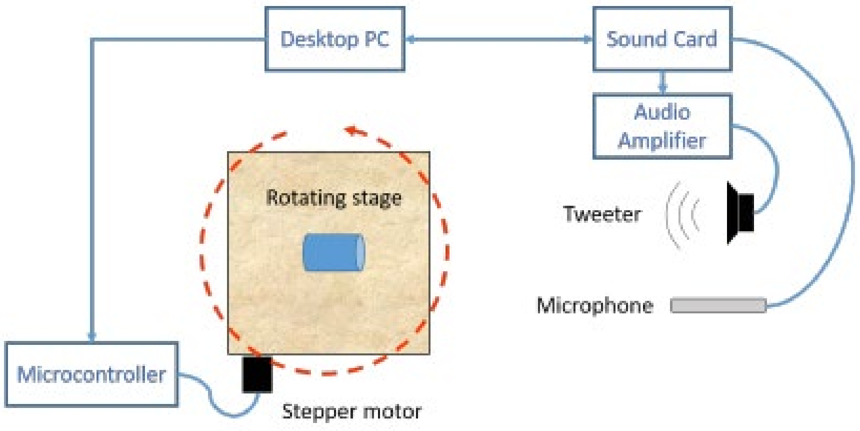


Figure 15. The progenitor to the AirSAS setup used for this project adapted from Blanford et al. Top is the actual setup, while bottom is the setup diagram [7]

SAS imagery is generated from the AirSAS raw data by reorganizing the returned data in terms of time and space. Doing this forms a grid of pixels for the imaging scene. This approach is known as backprojection reconstruction [16].

Previous works in the area of combining acoustic imagery with sonar imagery have proven effective, albeit with some interesting findings. The study conducted by Williams [17] employed the use of reducing the size of the network to reduce the need of large datasets. Networks that are too complex for the input dataset tend to over-constrain themselves to a set sequence. William's approach seeks to use tiny networks in order to classify datasets with fewer parameters. The final section in this literature review will present similar projects in this area of research.

2.7 Similar Works

An interesting application of the theory of applying neural networks to sinusoidal waveforms is music recognition. Most base-level approaches to this problem make the use of a short form Fourier transform, that can convert a waveform into a visual representation containing all the identifying information for a specific song. This visual representation is a spectrogram, which shows each individual frequency present in a signal as they vary with time. Then, it is possible to train an image classification algorithm on clips from different genres. This allows an algorithm to classify songs based on genre, artist, etc… after viewing only a few seconds of a clip. A project demonstrating this was conducted by Costa *et al.* [18]. Here, clips from several music clip datasets were used in a classification study. The network was trained in two modes. One used the visual representations of each clip in the form of a spectrogram, while the other used numerical features from the waveform. The highest accuracy reached in the project was 87.4 percent for the visual approach.

In preparation for this project, I put together a music genre classifier of my own. To cut down on training time, I found that the numerical method using acoustic features worked best. Figure 16 shows an example of a waveform obtained in this project, while Figure 17 shows a spectrogram for the visual recognition case.

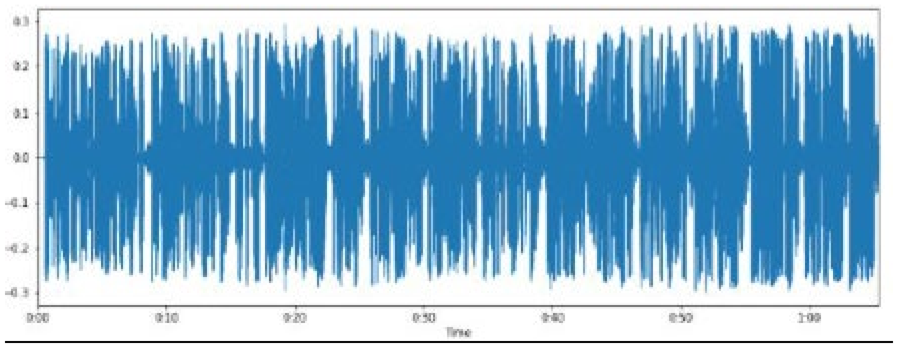


Figure 16. Acoustic Time Series for a 30 second clip [4]

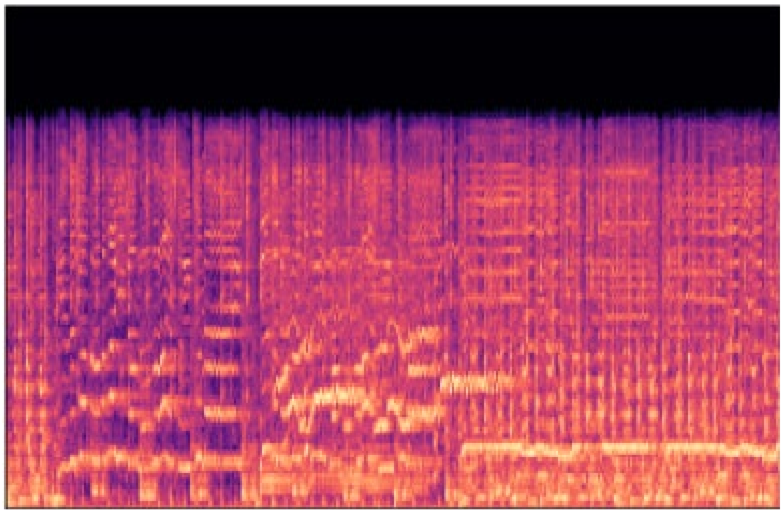


Figure 17. Spectrogram for a 30 second clip of music [4]

In this project, I applied the Fast.ai tabular learner to sort acoustic features from an input song into one of several genres. For training, I extracted acoustic features from the GTZAN dataset. This dataset contains 1000 clips of songs at 30 seconds each, sorted into 10 distinct genres. I tested a 3-layer and a 2-layer network while varying the number of nodes per layer from low to high. I also tested how the accuracy would vary by limiting the number of genres to be sorted in some cases. I found that I obtained my best results with the configurations shown below. [4]

Table 1. Results from testing deep vs. shallow networks on 30 second clips [4]

Genre set	Layers	Epochs	Accuracy
rock, pop	[200, 100]	13	.925
rock, pop, classical, jazz	[100, 50]	32	.962025
rock, pop, classical, jazz, hiphop, disco, metal	[200, 100, 50]	9	.820144
rock, pop, classical, jazz, hiphop, disco, metal, reggae, country, blues	[100, 50, 25]	29	.7236

Sorting clips into 4 possible genres with a smaller, shallower network gave me a high accuracy rating of 96.20%. While this experiment will not employ the use of generated spectrograms or acoustic features, I will employ the same methodology of limiting the complexity of the network for a less complicated problem.

Chapter 3

Motivation and Goals

3.1 Sound Hunter

My original motivation for this project came after I worked on the Soundhunter research platform in the Applied Laboratory during the summer of my sophomore year. The platform is intended to extend SAS reconstruction to detect objects buried in shallow lakebeds. Figure 18 shows an example of expected imagery from this process.

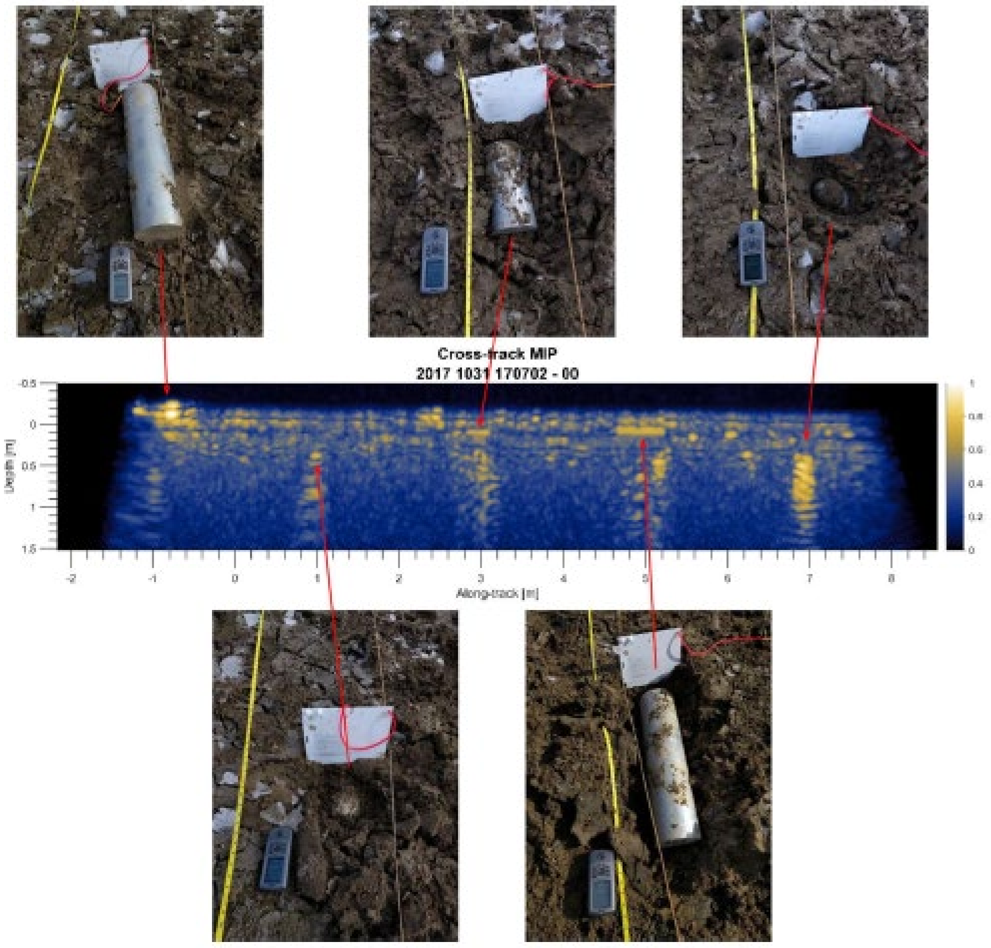


Figure 18. Example of buried object detection in a shallow lakebed adapted from Brown [19]

Cylindrical stock was buried in the sediment layers of a man-made lake during the winter. The lake was drained at the time, making this possible [19]. During my work period, we made several passes to collect new data. While my role was mainly focused on designing hardware for sensors, I saw that there was an opportunity for research with this methodology. The process of identifying objects of different types within the scans bore resemblance to an image classification problem. It would be possible to train a neural network to recognize our specimens we had the means to build a database of scans of all the types of cylindrical stock we would use. Additionally, we would also have the ability to identify links between SAS imagery and other properties of the stock like density or surface roughness. There was not enough time to explore this problem during my time on that project, but the 3D printed MNIST dataset gave us a good representation of what this approach would yield.

3.2 Goals and Parameters

The primary goal of this thesis is to determine whether there is a case to be made for classifying SAS test imagery on a neural network trained using generated MATLAB imagery. The primary gauge for success was to achieve an accuracy rate of at least 80% on our testing set. We can define the accuracy rate as the fraction of scans from the AirSAS that are classified correctly in our training set. In order to define a hit or a miss, a label was defined in each scan's filename corresponding to the number that scan is supposed to represent. We then organized the scans based on whether their classified category and labeled were equal or not. Plots of validation loss and training loss were also used as indicators showing the network performance. Loss is used to gauge the prediction error of the neural network during the training phase [3]. Plotting the training loss would ideally show an exponentially decreasing curve as the network optimizes itself for a certain training set. Due to the random nature of picking a validation set from the training set, validation loss can be more varied but is normally expected to decrease similarly to the training loss. Figure 19 shows an example plot of accuracy, training loss, and validation loss from my pilot experiment.

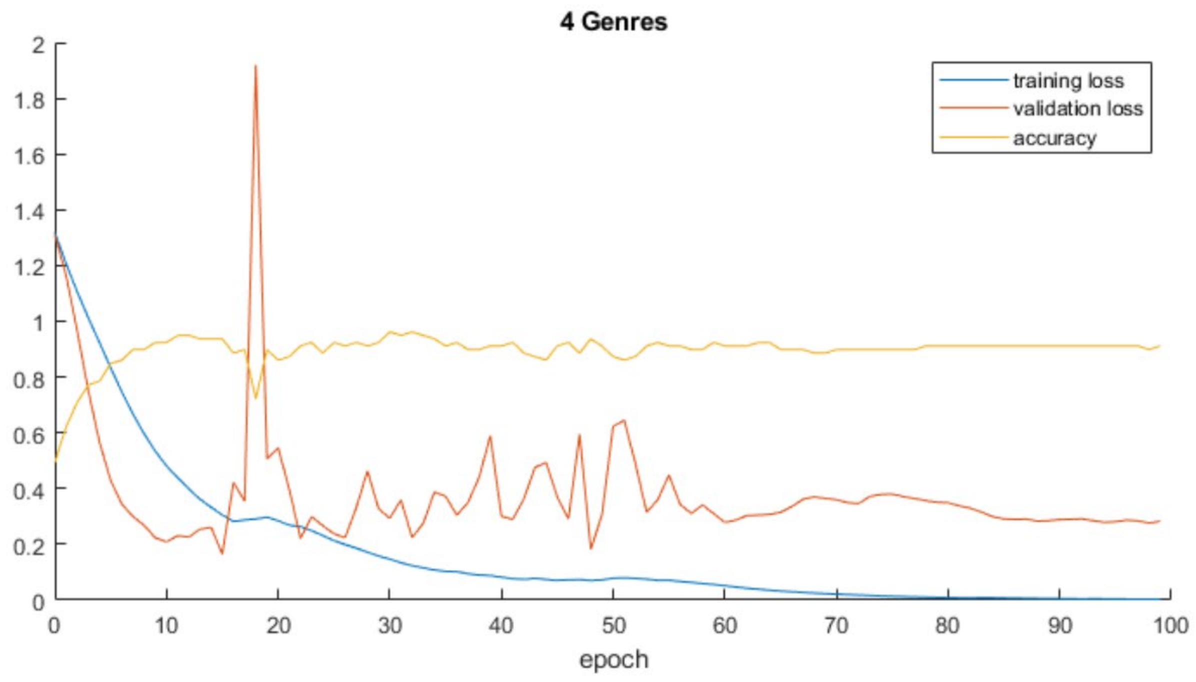


Figure 19. Plot showing training loss, validation loss, and accuracy for a neural network [4]

For this run, each indicator behaved as expected with the accuracy rate reaching > 90%. To adjust the behavior of the network from run to run, several parameters were adjusted. The first of these parameters was the learning rate. The learning rate is defined as a constant between 0 and 1. This constant is used in the formula for the weight adjustment of each neuron in the training process [3]. Figure 20 outlines the formula used in training simple neural networks, the Delta rule.

$$\Delta w_I = \alpha(t - y\_in)x_I.$$

Figure 20. The Delta Rule adapted from Fausett [3]

The learning rate is defined by α. t is the expected output, while y_in is the output determined by the network. x_i is the input for neuron i, and Δw_i is the resulting weight change for the current epoch of training. Varying the learning rate explores the possibilities of a network that changes more or less per epoch. A learning rate that is too high can lead to a network that is overfitted to a specific expected training set, while a learning rate that is too low can cause the network to be over-generalized. The next parameter to consider is the training duration of the network. This value is defined as the number of epochs to train the network with the chosen test set. The number of epochs is usually chosen in conjunction with the learning rate, as both affect the network's tendency to gravitate towards the test set.

The last parameter to consider is the overall structure of the network. As stated earlier in section 2.6, a networks complexity should generally be equivalent to the complexity of the inputs. This consideration also applies to the number of inputs available to train the network. Simpler networks are suited for use in situations with small amounts of training data. In order to explore the possibilities available, a few different network structures were considered. I used 2-layer and 3-layer networks to gauge the response of deep vs. shallow networks. To vary the size of each layer, I alternated between low and high node counts. The pilot experiment showed that there were applications where smaller, simpler networks were ultimately more effective due to the simpler nature of the problem.

Chapter 4

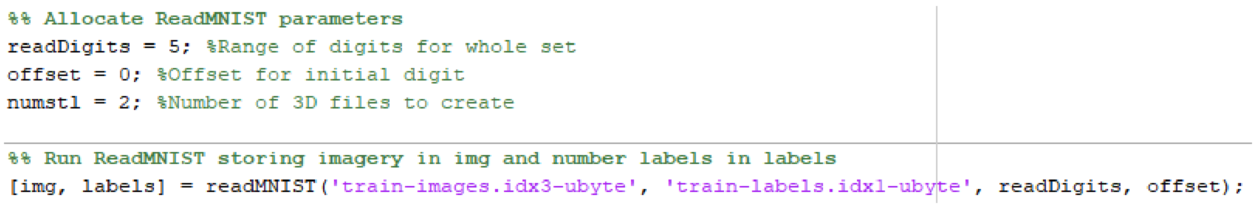
Data Methodology and Tool Design

4.1 Forms of Data

The first and most impactful step to a data classification study is defining and processing the training/testing datasets. While the MNIST set requires little processing to be used in 2-D graphical studies, the AirSAS requires a physical model. This physical model must also be designed in a way which is quickly printed and yields a defined response from the AirSAS [15]. There must be some extrapolation in the Z axis to create scattering phenomena in our output plots [7]. Therefore, this experiment required MNIST dataset to be transformed into two datatypes. First, we had to convert the raw MNIST data into imagery that could be used as the training set for the Fast.ai classifier. This was followed by converting the same raw MNIST data into .STL files that could be used to 3D print our physical specimens. An iteration process followed this where we determined how to generate thin, small models that could yield clear imagery from the AirSAS. Finally, the classifier functions were rewritten to organize these datasets.

4.2 Creating Test Models

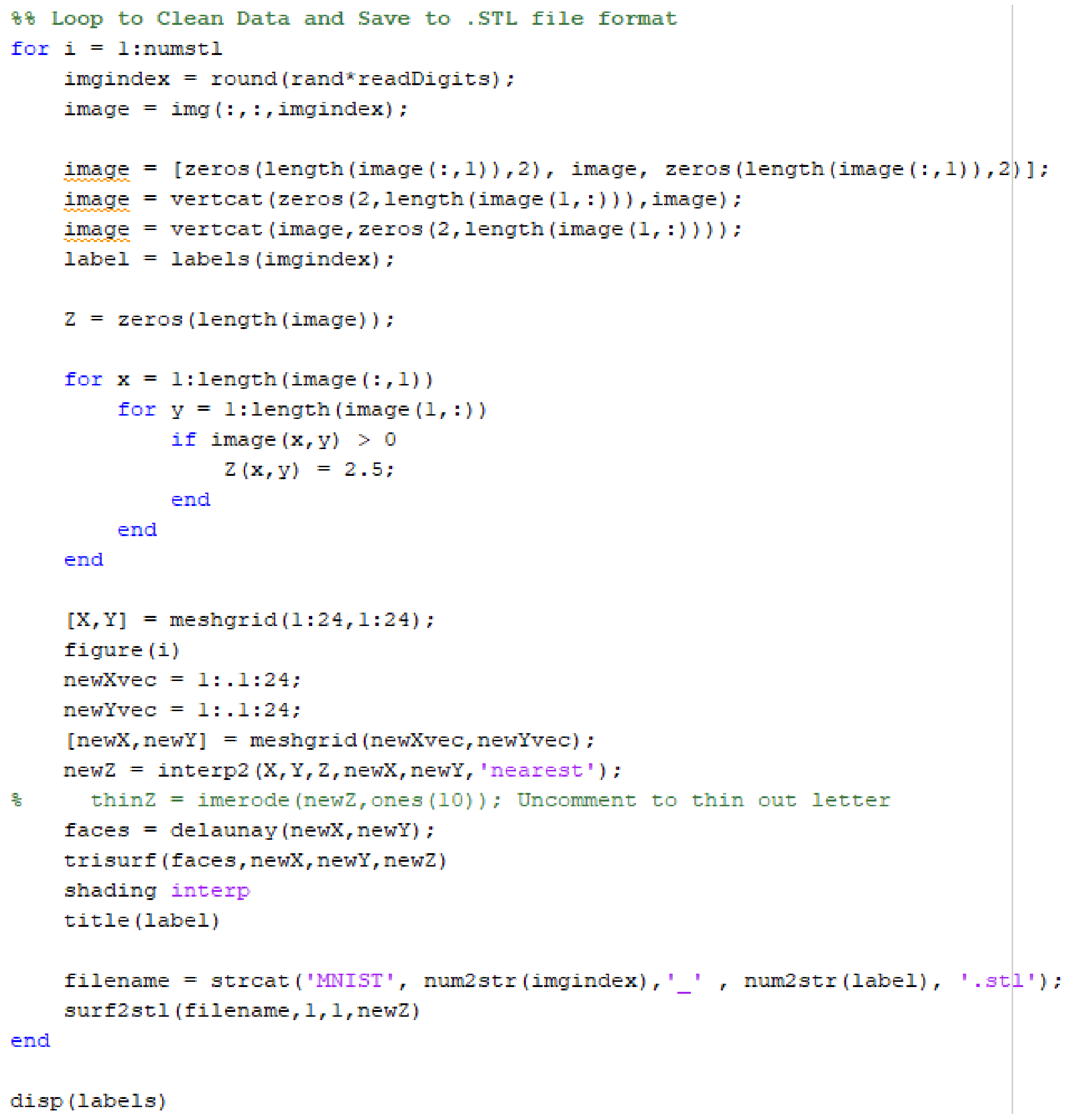
As stated in section 2.5, the MNIST dataset is organized into 50,000 training images each with a map of 28x28 pixels [15]. MATLAB was chosen to process these maps due to its ease of use when processing large matrices. The initial loading of the MNIST set from its input form was handled by readMNIST function. This function reads a chosen number of digits from the MNIST file with a chosen offset. It outputs data in the form of a 3-dimensional array that can be cycled through to view all images. Figure 21 shows the steps taken to run the readMNIST command.



```
%% Allocate ReadMNIST parameters
readDigits = 5; %Range of digits for whole set
offset = 0; %Offset for initial digit
numstl = 2; %Number of 3D files to create

%% Run ReadMNIST storing imagery in img and number labels in labels
[img, labels] = readMNIST('train-images.idx3-ubyte', 'train-labels.idx1-ubyte', readDigits, offset);
```

Figure 21. Setting initial parameters for the readMNIST command

The main loop is called the readMNIST function stores our number grids in the 3-dimensional array img and the labels for each number in the vector label. This loop is run a predetermined number of times and randomly picks a number from the range specified by readDigits. First, the chosen number is padded around the edges with a few rows of empty space. This padding allows the surface plot to create a fully enclosed 3D image with no open edges. Proceeding this is a short loop leveling all pixels in the chosen number at 2.5 millimeters in height. The commands finishing out the loop construct a surface plot by creating and interpolating a mesh grid. Interpolation is necessary to limit the slanted edges that may be produced in a 3D printed model. Finally, the model is named and saved. Figure 22 shows the rest of the code for this function.



```
%% Loop to Clean Data and Save to .STL file format
for i = 1:numstl
    imgindex = round(rand*readDigits);
    image = img(:,:,imgindex);

    image = [zeros(length(image(:,1)),2), image, zeros(length(image(:,1)),2)];
    image = vertcat(zeros(2,length(image(1,:))),image);
    image = vertcat(image,zeros(2,length(image(1,:))));
    label = labels(imgindex);

    Z = zeros(length(image));

    for x = 1:length(image(:,1))
        for y = 1:length(image(1,:))
            if image(x,y) > 0
                Z(x,y) = 2.5;
            end
        end
    end

    [X,Y] = meshgrid(1:24,1:24);
    figure(i)
    newXvec = 1:.1:24;
    newYvec = 1:.1:24;
    [newX,newY] = meshgrid(newXvec,newYvec);
    newZ = interp2(X,Y,Z,newX,newY,'nearest');
%     thinZ = imerode(newZ,ones(10)); Uncomment to thin out letter
    faces = delaunay(newX,newY);
    trisurf(faces,newX,newY,newZ)
    shading interp
    title(label)

    filename = strcat('MNIST', num2str(imgindex),'_' , num2str(label), '.stl');
    surf2stl(filename,1,1,newZ)
end

disp(labels)
```

Figure 22. The main loop of MNIST2STL

Each model is plotted as a surface plot and saved as an STL file. Figure 23 shows the surface plot output, and Figure 24 shows 4 numbers loaded into a 3D printer slicing software.

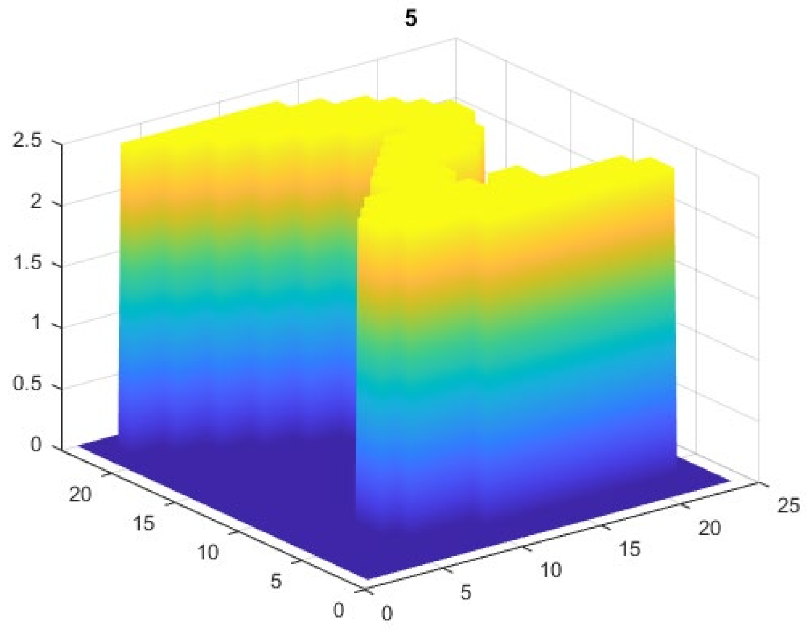


Figure 23. MNIST2STL surface plot output

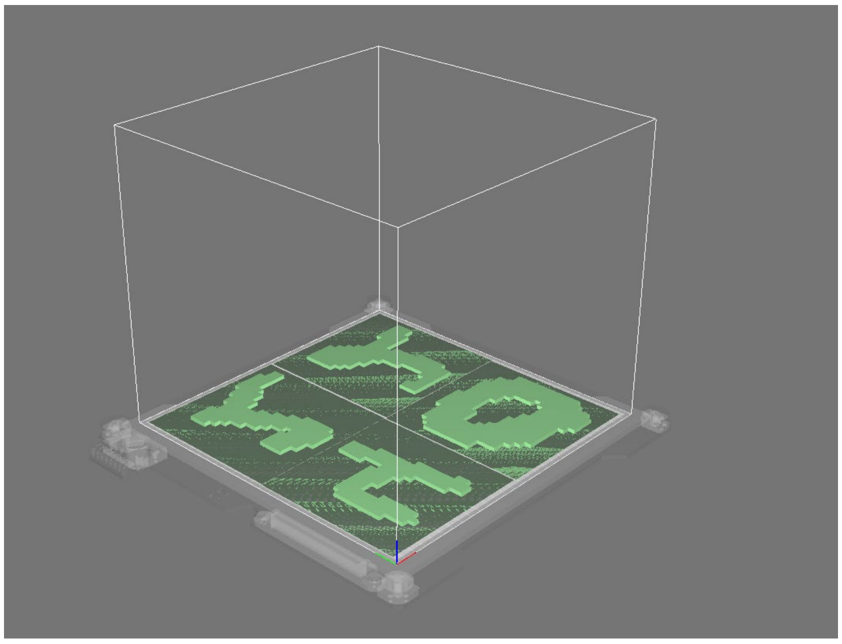


Figure 24. Models in slicing software

The models required iteration to get to the point where useable data could be produced for the experiment. Main factors considered were time cost, size, and potential to scatter acoustic waves. We found that numbers that were interpolated properly had edges at 90 degrees, and overall produced much clearer images. Size was varied as well. Initial letters were small on the order of 1.5in^2 while later iterations hit 7in^2. Eventually, 4in^2 was chosen as a middle ground between print speed and scan clarity. Figure 25 shows each number iteration and their respective acoustic scan. Notice how the graphed response increases in both clarity and density.

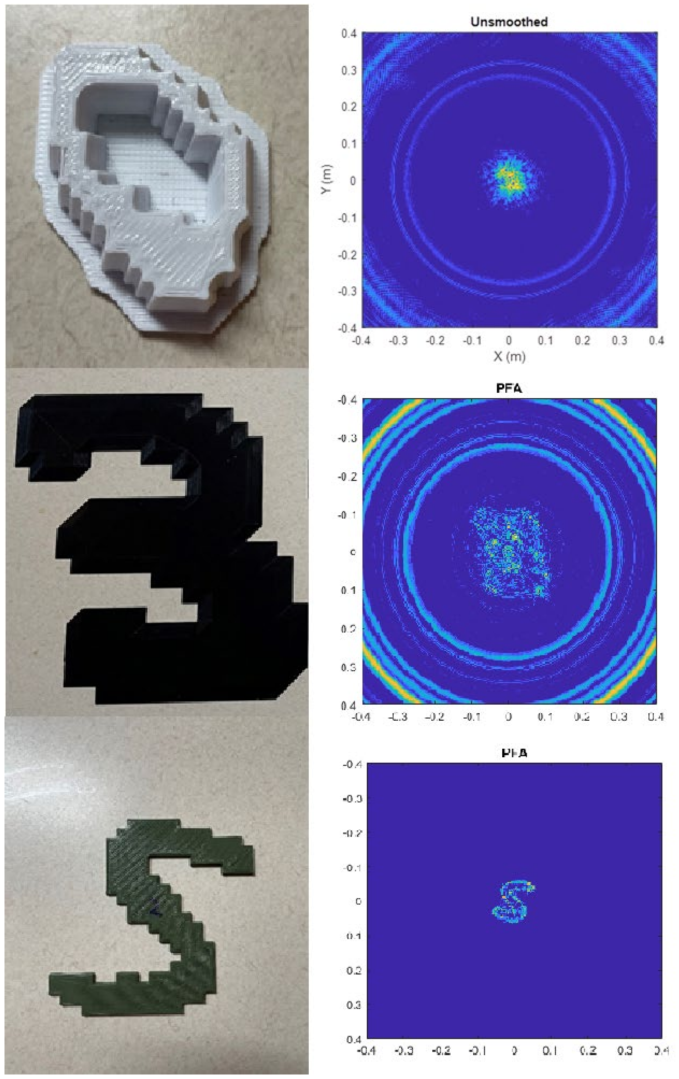
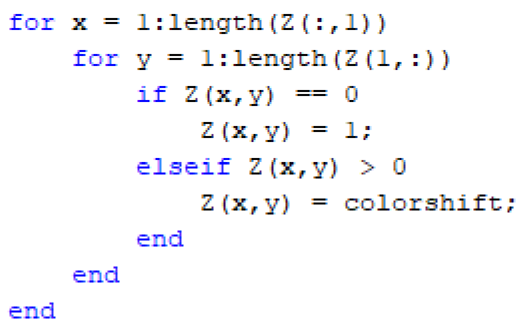


Figure 25. Response from various 3D printed specimens

4.2 Creating Training Imagery

Creating the training imagery for the network posed a unique problem on its own. The algorithm would have to that would make usable images without editing the source material greatly. Too much filtering would make this situation less of a MNIST training problem and more of a question on how we can create simulated AirSAS imagery. Therefore, the script to develop these images was more barebones than originally envisioned.

The script starts with the same calls as the test set script but adds variables to determine empty padding space around letters and a color shifting value. The main loop is similar up until the for loops with the exception of the padding step which uses the predetermined value. In the for loop, all zero pixels are set to a height of one millimeter while active pixels are set to a height equal to the color shifting value. Figure 26 shows the code used for this process.



```
for x = 1:length(Z(:,1))
    for y = 1:length(Z(1,:))
        if Z(x,y) == 0
            Z(x,y) = 1;
        elseif Z(x,y) > 0
            Z(x,y) = colorshift;
        end
    end
end
```

Figure 26. For loop used to set height of each pixel in training images

Next, a mesh grid is setup to create the surface plot for the image. The same interpolation process from the test script. The number is then mapped out on a plot, with the view set to overhead. The colorbar and title are both turned off. When testing the script, imagery is generated. However, creating multiple figure windows exhausts memory quickly. Therefore, visible plotting is turned off when producing training sets. Figure 27 shows an example image from one of these runs.

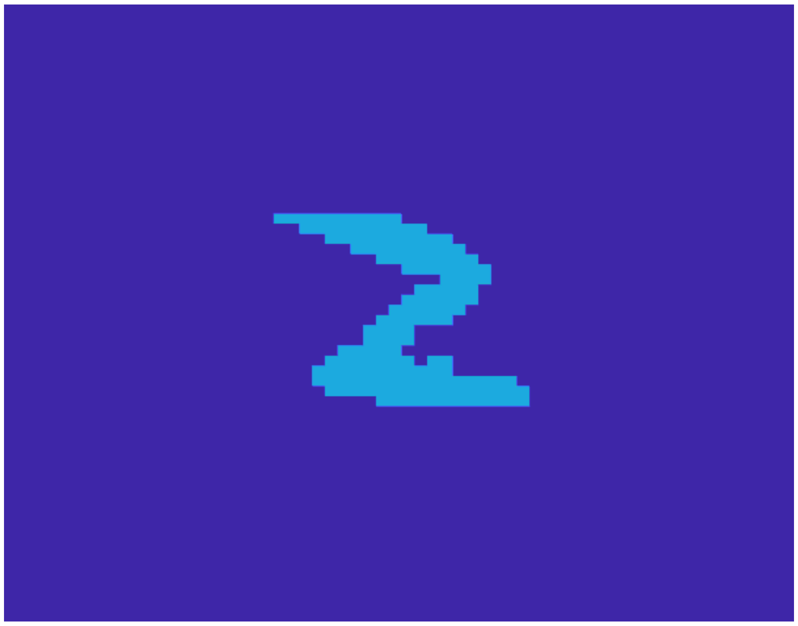
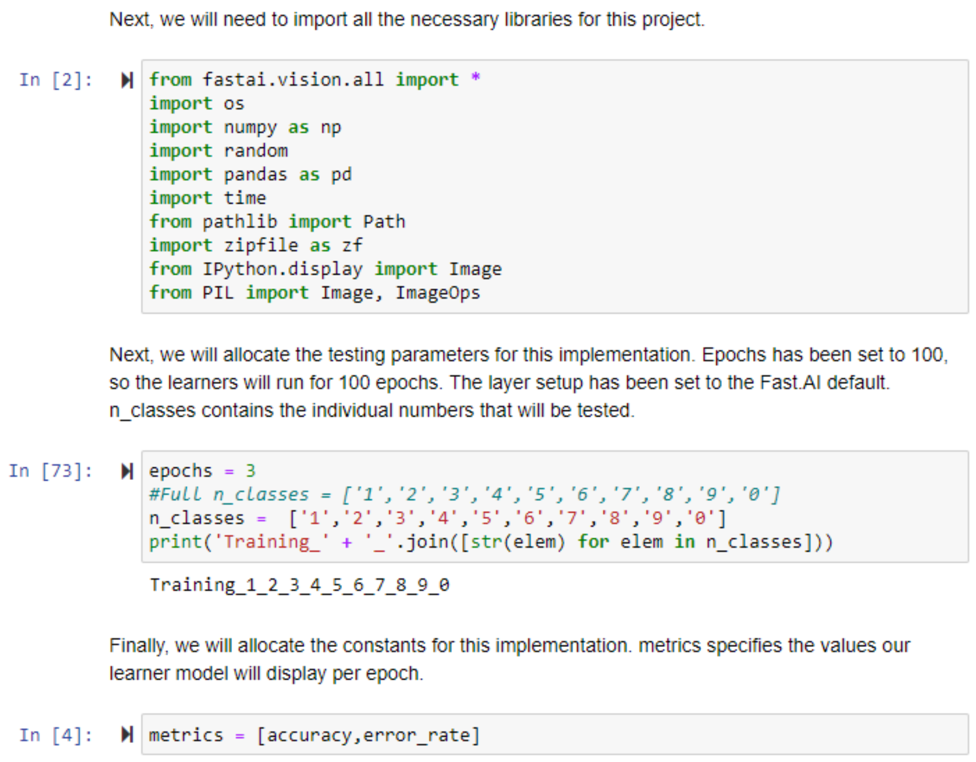


Figure 27. Example image for the number 2 used for training

4.4 Image Classifier

As outlined in section 2.4, the Fast.ai library was the tool of choice for constructing the image classifier. Fast.ai allows the use of high-level components to create and train pre-made image classifying models. The library uses the python language, giving us access to simple components to organize and label files. This is imperative as labels are usually acquired through filename parsing. Training images are expected to have the filename form of MNIST(Dataset #)_(Label)_(Dataset). The dataset # signifies the order in which the numbers are generated, while the label identifies what the number is supposed to represent. The final parameter, Dataset is meant to describe any generations the generation script has applied.

The format used for this and many other machine learning projects is the Jupyter Notebook. The Jupyter notebook allows for a cohesive combination of code cells, documentation, and imagery. Code cells are self-contained and ran independently of each other, making it possible for practitioners to replicate experiments. Jupyter Notebooks are not normally run locally. In this case, the Jupyter notebook was run on a paid cloud computing service called Datacrunch.io. The first few cells define the libraries to be imported and allocate certain metrics/variables. These operations and descriptions are shown in Figure 28.


Next, we will need to import all the necessary libraries for this project.

```
In [2]:  from fastai.vision.all import *
         import os
         import numpy as np
         import random
         import pandas as pd
         import time
         from pathlib import Path
         import zipfile as zf
         from IPython.display import Image
         from PIL import Image, ImageOps
```

Next, we will allocate the testing parameters for this implementation. Epochs has been set to 100, so the learners will run for 100 epochs. The layer setup has been set to the Fast.AI default. n_classes contains the individual numbers that will be tested.

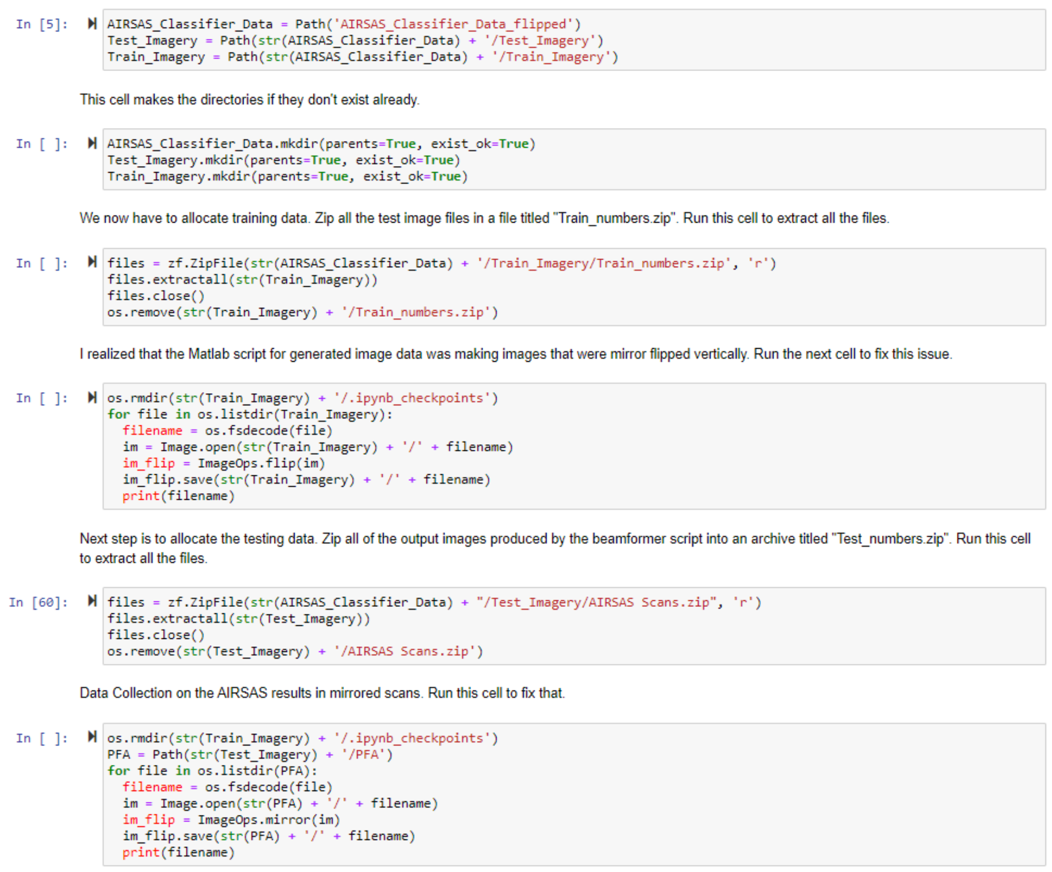
```
In [73]: epochs = 3
         #Full n_classes = ['1','2','3','4','5','6','7','8','9','0']
         n_classes =  ['1','2','3','4','5','6','7','8','9','0']
         print('Training_' + '_'.join([str(elem) for elem in n_classes]))

         Training_1_2_3_4_5_6_7_8_9_0
```

Finally, we will allocate the constants for this implementation. metrics specifies the values our learner model will display per epoch.

```
In [4]:  metrics = [accuracy,error_rate]
```

Figure 28. Initializing parameters and libraries in the image classifier

In the next portion of the notebook, I set up the folder structure and file organization. Two main directories were made to hold the training set and test set. Immediately after creating the main directories and their paths, instructions are shown to upload the sets to Datacrunch in the form of .zip files. Next, these files are unzipped and organized. Finally, the Training Images from MATLAB flipped vertically while the scans from the AirSAS are mirror flipped. This is done so every image is set to the same general orientation. These operations are detailed in Figure 29.



```
In [5]:  AIRSAS_Classifier_Data = Path('AIRSAS_Classifier_Data_flipped')
         Test_Imagery = Path(str(AIRSAS_Classifier_Data) + '/Test_Imagery')
         Train_Imagery = Path(str(AIRSAS_Classifier_Data) + '/Train_Imagery')
```

This cell makes the directories if they don't exist already.

```
In [ ]:  AIRSAS_Classifier_Data.mkdir(parents=True, exist_ok=True)
         Test_Imagery.mkdir(parents=True, exist_ok=True)
         Train_Imagery.mkdir(parents=True, exist_ok=True)
```

We now have to allocate training data. Zip all the test image files in a file titled "Train_numbers.zip". Run this cell to extract all the files.

```
In [ ]:  files = zf.ZipFile(str(AIRSAS_Classifier_Data) + '/Train_Imagery/Train_numbers.zip', 'r')
         files.extractall(str(Train_Imagery))
         files.close()
         os.remove(str(Train_Imagery) + '/Train_numbers.zip')
```

I realized that the Matlab script for generated image data was making images that were mirror flipped vertically. Run the next cell to fix this issue.

```
In [ ]:  os.rmdir(str(Train_Imagery) + '/.ipynb_checkpoints')
         for file in os.listdir(Train_Imagery):
           filename = os.fsdecode(file)
           im = Image.open(str(Train_Imagery) + '/' + filename)
           im_flip = ImageOps.flip(im)
           im_flip.save(str(Train_Imagery) + '/' + filename)
           print(filename)
```

Next step is to allocate the testing data. Zip all of the output images produced by the beamformer script into an archive titled "Test_numbers.zip". Run this cell to extract all the files.

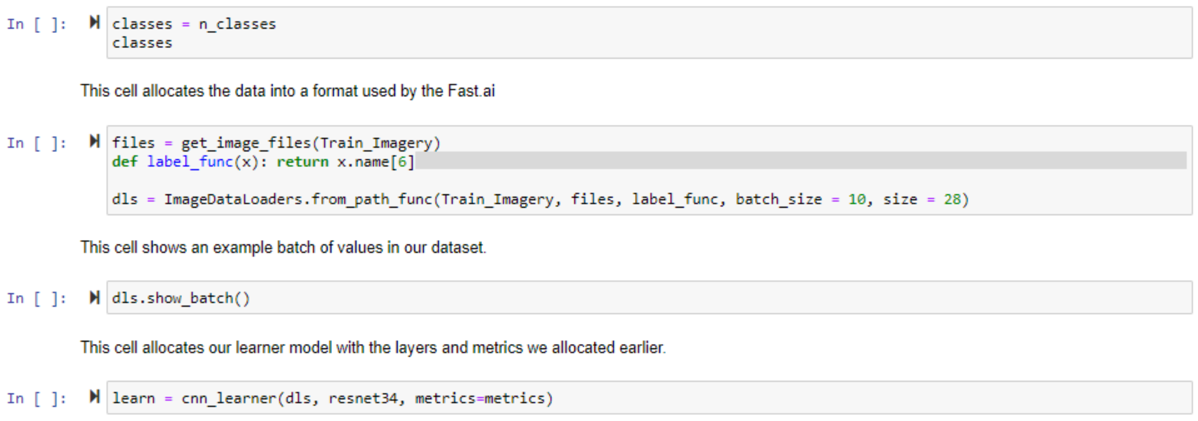
```
In [60]:  files = zf.ZipFile(str(AIRSAS_Classifier_Data) + "/Test_Imagery/AIRSAS Scans.zip", 'r')
          files.extractall(str(Test_Imagery))
          files.close()
          os.remove(str(Test_Imagery) + '/AIRSAS Scans.zip')
```

Data Collection on the AIRSAS results in mirrored scans. Run this cell to fix that.

```
In [ ]:  os.rmdir(str(Train_Imagery) + '/.ipynb_checkpoints')
         PFA = Path(str(Test_Imagery) + '/PFA')
         for file in os.listdir(PFA):
           filename = os.fsdecode(file)
           im = Image.open(str(PFA) + '/' + filename)
           im_flip = ImageOps.mirror(im)
           im_flip.save(str(PFA) + '/' + filename)
           print(filename)
```

Figure 29. Folder structure of Jupyter Notebook

After creating the folder, some example commands are run to illustrate the functionality of the notebook. Here, a random image is chosen from the training set and displayed for viewing. In order for image classifier to be implemented, a few conditions must be met. First, the data must be organized into an object organizing each individual image into a class denoted by its label. In this project, this label is kept in the filename. Each class is meant to be a number on the scale of zero to nine, creating 10 classes in all. When making the object, the class data is lifted by a label function that reads the digit in the -16th position of the image filename. This position corresponds to the label, which is used to organize the data into its respective class. After the object is created. the next code cell shows a single batch of data with its individual labels. Finally, the object is loaded into a learner object. This learner object can be thought of the image classifier itself, which is allocated with a chosen architecture and display metrics. These operations and example batch are detailed in Figures 30 and 31.



```
In [ ]:  classes = n_classes
         classes
```

This cell allocates the data into a format used by the Fast.ai

```
In [ ]:  files = get_image_files(Train_Imagery)
         def label_func(x): return x.name[6]

         dls = ImageDataLoaders.from_path_func(Train_Imagery, files, label_func, batch_size = 10, size = 28)
```

This cell shows an example batch of values in our dataset.

```
In [ ]:  dls.show_batch()
```

This cell allocates our learner model with the layers and metrics we allocated earlier.

```
In [ ]:  learn = cnn_learner(dls, resnet34, metrics=metrics)
```

Figure 30. Code cells for creating learner and data objects

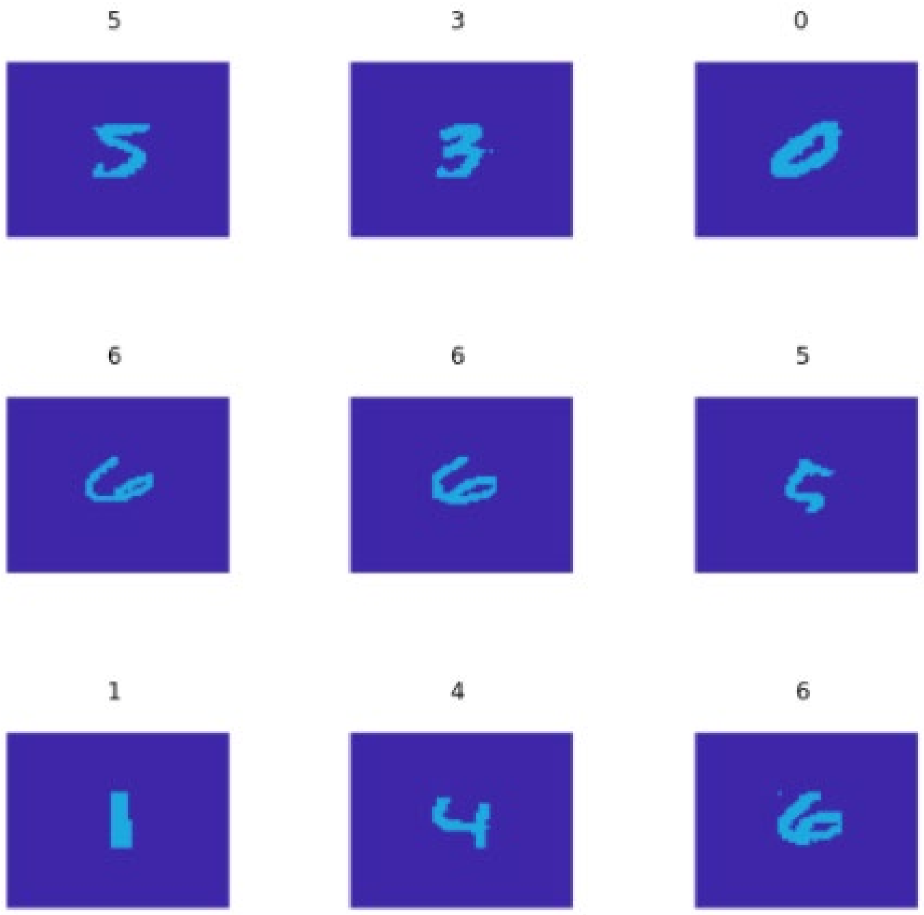


Figure 31. Example batch from the input dataset

Training occurs after creating the learner object. The learner is trained to recognize the class of each image in the data object. This is done with the fit_one_cycle() method, which runs the entire dataset through the network for a predetermined number of epochs. Following this operation, the model is saved and exported for later evaluation. Finally, a confusion matrix is constructed where we can see the comparison between the true and predicted class of each number. Following this is a cell where the classes of test images are predicted and printed to the console. Figures 32 and 33 shows these final two processes.

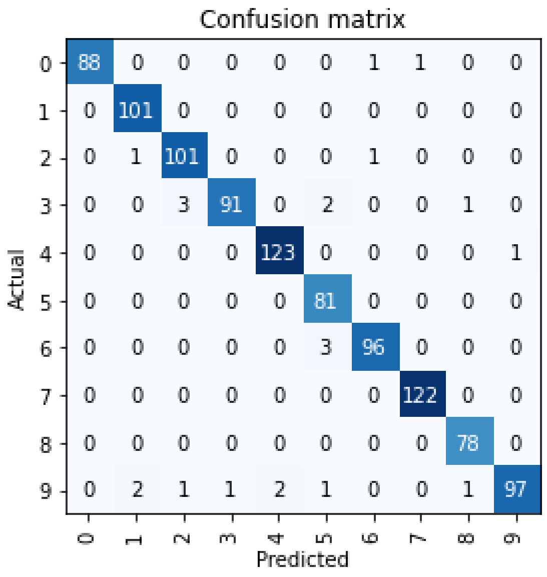
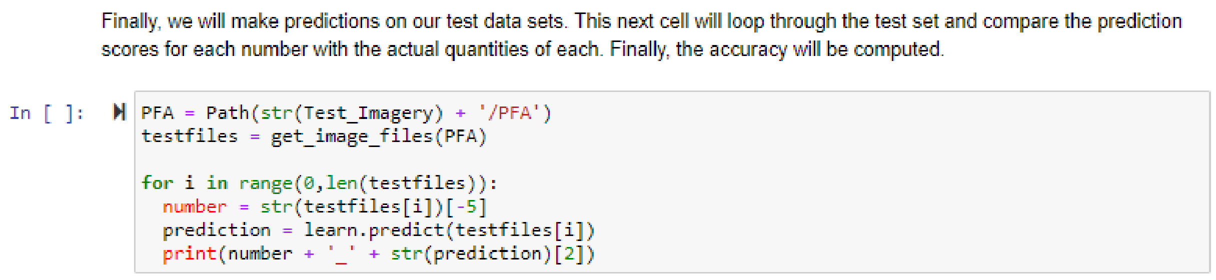


Figure 32. Confusion matrix example for training data



Finally, we will make predictions on our test data sets. This next cell will loop through the test set and compare the prediction scores for each number with the actual quantities of each. Finally, the accuracy will be computed.

```
In [ ]:  PFA = Path(str(Test_Imagery) + '/PFA')
         testfiles = get_image_files(PFA)

         for i in range(0,len(testfiles)):
           number = str(testfiles[i])[-5]
           prediction = learn.predict(testfiles[i])
           print(number + '_' + str(prediction)[2])
```

Figure 33. Test set prediction code cell

Chapter 5

Experimental Procedure

5.1 Timeline

This experiment was conducted over the past few months while the effects of COVID-19 were affecting normal operations. This meant that meeting in the main lab was not an option until absolutely necessary, and most work had to be done in a remote format. Luckily, I was able to access a 3D printer during this period due to my involvement in Digi Digits, a student organization at Penn State. The first major milestone covered in this project was conducting background research, outlining experiment parameters, and learning to use the Fast.ai library. This was done in the period from Fall 2019 to Winter 2020. At the end of this period, I conducted the pilot experiment detailed in Section 2.7.

The first step in obtaining the printed models was to configure the 3D printer and the respective scripts to create 3D imagery. This process was completed during Winter 2021, where afterwards the printer would be continuously creating the models for the test set. Following this, I focused on creating the scripts for the testing imagery and fit the classifier model from the pilot experiment to this thesis. By early march, both the framework and imagery of the experiment were ready. In mid-March, I gained access to the ARL building again where I could continuously scan the 50 numbers printed previously. Example scans are shown below in Figure 34.

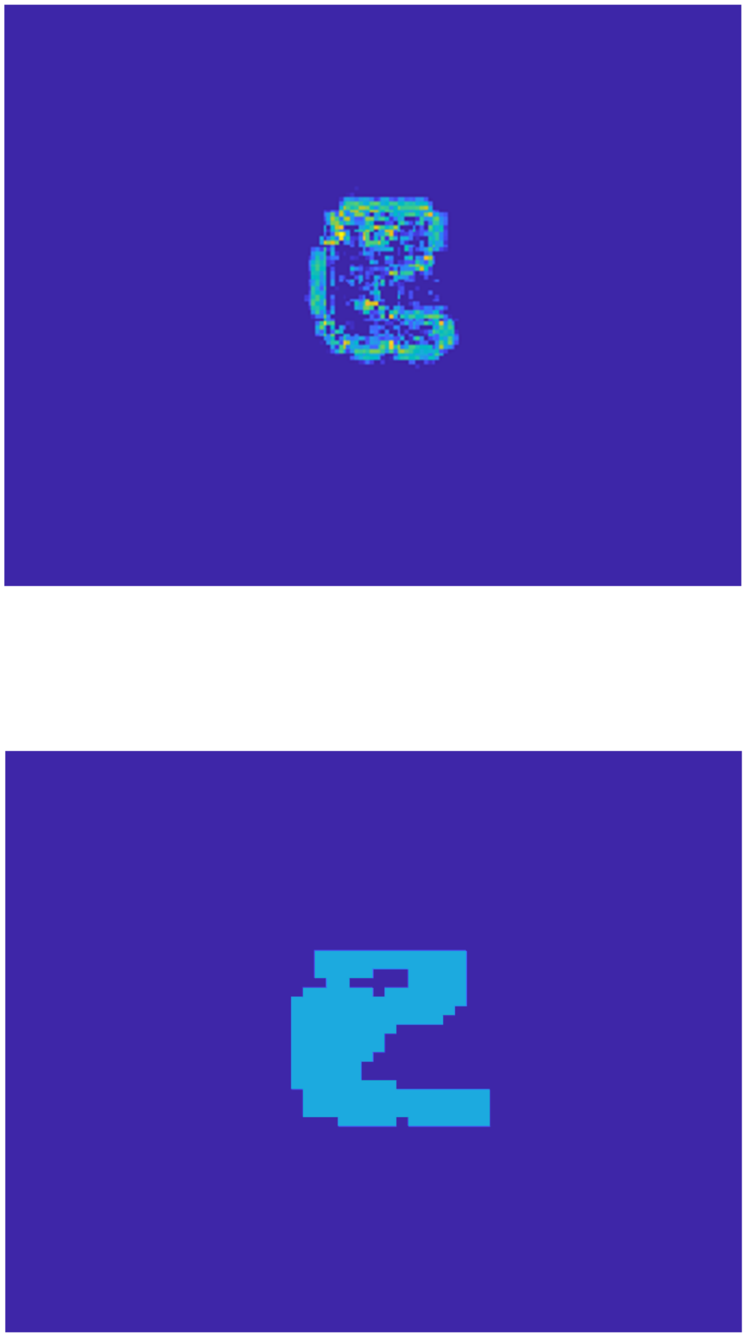


Figure 34. The number 5 as scanned by the AIRSAS (top) and generated by MATLAB (bottom)

5.2 Procedure

Generating training data was the part of putting the experiment into production. The MNIST set originally housed 60000 images for its testing set. A dataset this large would be cumbersome when migrating between the ARL systems, my PC, and online storage with my current hardware access. Therefore, 5000 images were sampled at random from this set to construct the training set. These images were generated with the MATLAB script outlined in Section 4.3. These images were generated over the course of a few days.

Generating the test set required the use of the MATLAB script from Section 4.2. Each number was printed in sets of 4 at a size of around 4x4 inches each. Printing each individual number required around 3 hours, so this process was conducted over the course of a few weeks. 50 numbers were printed, again sampled at random from the set of 5000 numbers from the training set.

The process of scanning each letter began after the 50 letters were printed. Each number was scanned individually on the AirSAS test platform at the ARL. The numbers were examined for response from a ping at a sampling frequency of 100khz for 1-degree increments. A full set of pings were obtained for 360 degrees. After all letters were collected, I utilized a beam-former script to generate the test imagery set. This set was evaluated and the parameters were tweaked until a useable response was shown.

The image classifier was fitted with the 5000-number training set for durations of 10 and 100 epochs with batch sizes ranging from 2-60 until a usable response was observed. Training using small batch sizes resulted in a classifier that could not categorize the testing set with a reliable accuracy. Training using large batch sizes exhausted usable memory quickly, causing the system to crash. It was observed that a batch size of 10 was best for this study. This batch size allowed the classifier to reliably make predictions on the training set while not exhausting the resources of the system. Finally, predictions were made on the test set using the network, comparing the predicted values to their respective labels.

Chapter 6

Results and Analysis

The response observed from the training set from MATLAB was positive. A max accuracy rate of 99.6% was observed after 100 epochs of training. Similar degrees of success were observed whilst training for lower periods. 10 epochs of training yielded an accuracy rate of 98.7 percent. Figures 35 and 36 shows the graphs of training loss, validation loss, and accuracy for these two modes of training.

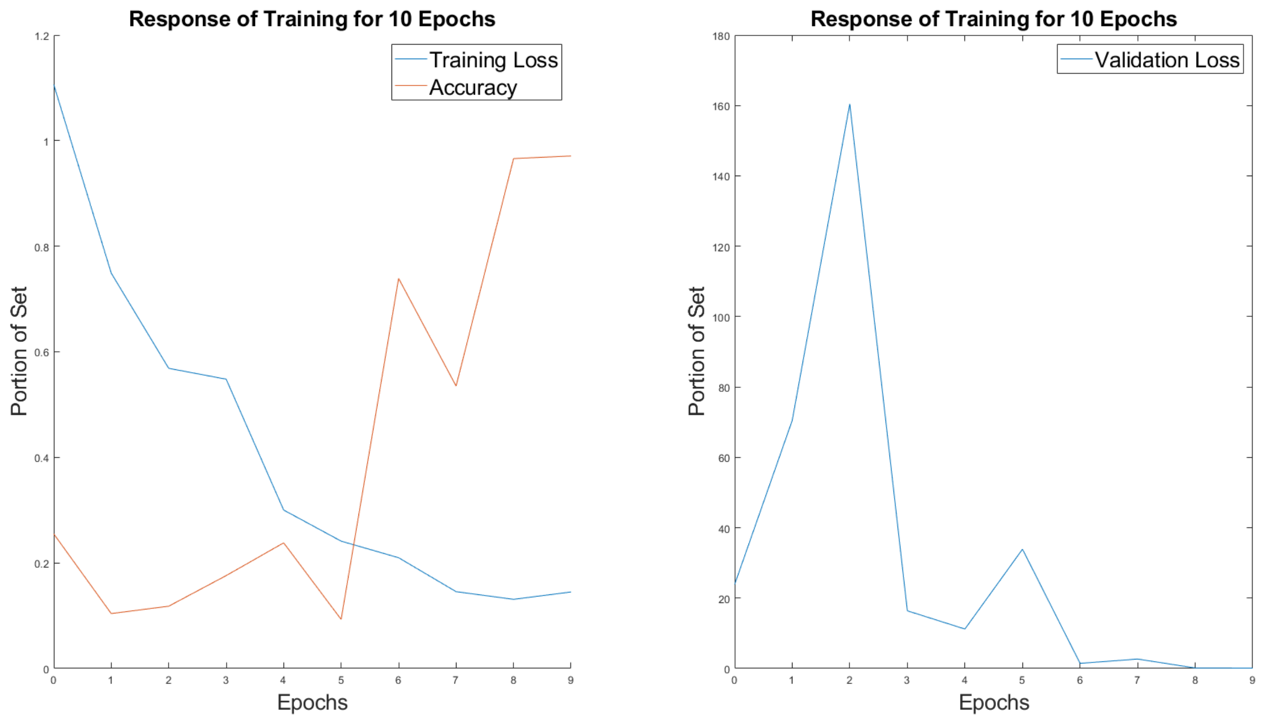
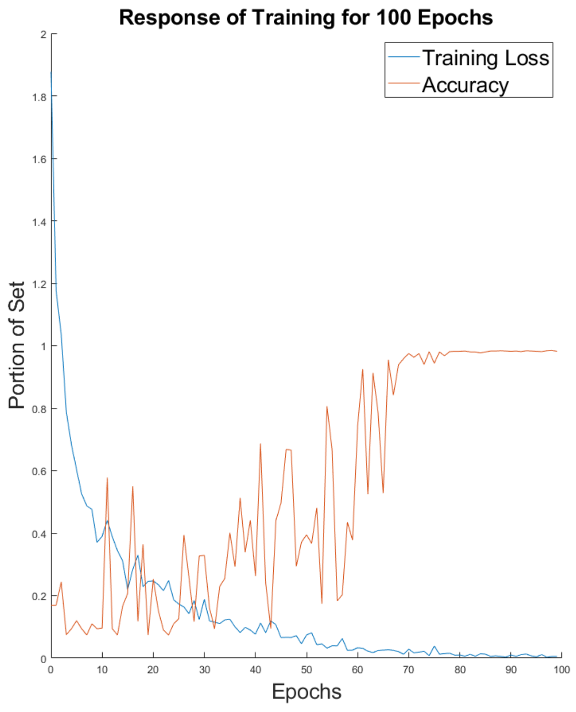


Figure 35. Response of training loss, accuracy, and validation loss over a 10-epoch training period



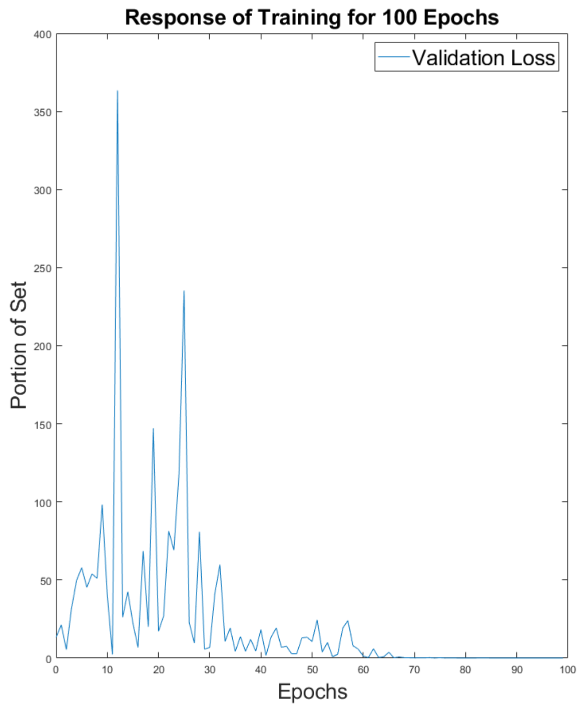


Figure 36. Response of training loss, accuracy, and validation loss over a 100-epoch training period

The values of training loss and validation loss are meant to signify the error between the calculated values after network prediction and the truth values for each number in the training set. The network reserves a majority of the entire training set to train the network, while a small part of the same set is reserved to validate the network after each epoch of training. For both graphs, training loss behaves as expected being an inverse from accuracy curve. However, the validation set exhibits a value several orders of magnitude higher than the training loss for the first portion of training. It is not until after about 1/3rd of the epochs has run where the validation loss stabilized to normal values. In the pilot experiment, this value showed similar behavior to the training loss with a similar curve on the same order of magnitude. This may be a sign that the network has been fitted tightly to the training set. Running the network for smaller durations shows similar behavior, with accuracy normally reaching the low 90 percent range. This behavior is tabulated in Table 2. In addition, confusion matrices are shown for the 10 and 100 epoch training durations in Figure 37.

Table 2. Observed accuracy values in response to training duration

Epochs	Accuracy Rate
3	.9352
4	.9815
5	.9694
10	.9870
100	.9956

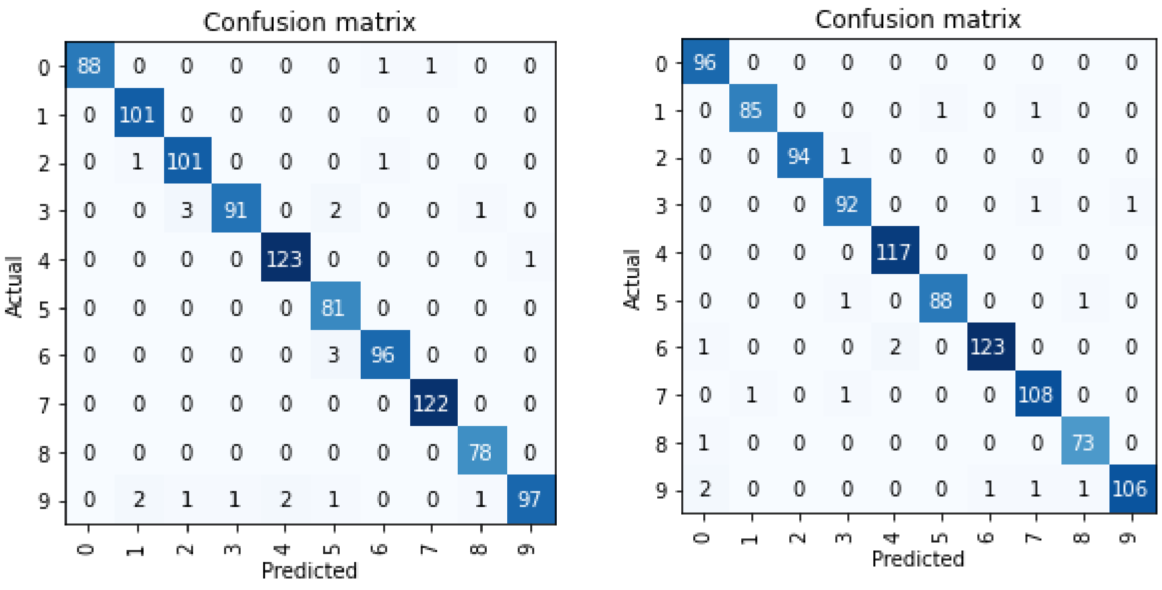
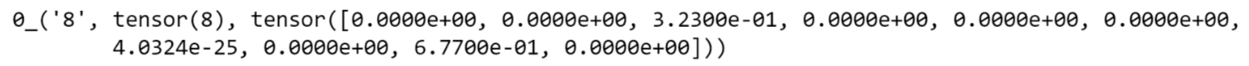


Figure 37. Confusion matrices detailing correct and incorrect predictions for 10 epochs (left) and 100 epochs (right)

Applying the testing set of the AirSAS scans did not yield nearly as promising values. Observing the output tensors of each scan showed that most runs resulted in no distinction being made by the classifier. Images where a prediction was made all seemed to be the same number, only varying when the script was reset. Figure 38 shows an example tensor response to the testing set.



```
0_('8', tensor(8), tensor([0.0000e+00, 0.0000e+00, 3.2300e-01, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        4.0324e-25, 0.0000e+00, 6.7700e-01, 0.0000e+00]))
```

Figure 38. Example tensor classifying 0 as 8 from a 5-epoch trained network

This is likely due to the network specializing to our training set to enough of a degree where classifying a different type of imagery is a non-trivial task. This heterogeneity between the training and testing set seems to pose an issue outside of the scope of this paper. However, the existence of some tensor response in the testing set and the high degree of accuracy whilst training the testing set gives rise to some possibilities in future experiments.

A few key differences exist between the AIRSAS data and the MATLAB data that likely contributed greatly to error in this test. Referring back to the backpropagation network illustrated in section 2.6, complex neural networks extract deep features from the input data and use those as classification markers. The MATLAB image set was a 1:1 translation of the MNIST dataset, while the AIRSAS scans introduced much more scattering and dispersed pixilation. The sheer difference between how the two datasets looked had a large impact on the training of the network, and likely contributed to the ultimate failure.

Another difference is the issue of scaling. This is mainly due to the AirSAS scans being images of real specimens and the MATLAB scans being of pixel maps. It is a project in and of itself to get AirSAS imagery with the exact same dimensions of the MATLAB scans, but this will most likely be necessary in order to get a constructive response for the network. Since the classifier processes each image pixel by pixel, multiple discontinuities between the scan can add up and impact other sets of weights.

Chapter 7

Conclusion and Future Steps

In this experiment, it was found that an image classifier using the Fast.AI library can classify a set of input data generated from MATLAB plots to a high degree of accuracy greater than 99.5%. However, the same network cannot be used to classify imagery of a different datatype despite some responses being observed. Furthermore, it was observed that training the dataset resulted in a sharp increase in accuracy and a sharp decrease in validation loss after a set fraction of the total training period, showing different performance from contemporary experiments.

Future areas to explore lie in a few domains. The first solution is to apply more preprocessing techniques to the input data. This can be done from studying the responses of AirSAS imagery and simulating these responses using MATLAB filtering commands. In addition, it may also be useful to remove the use of imagery entirely and instead categorize each object based on spectral coefficients obtained from a waveform. This manner of training would be much less computationally expensive but would require much more background research to implement.

An earlier incarnation of this project utilized the Microsoft Kinect to create depth maps as training imagery rather than MATLAB generated imagery. These depth maps would have been collected by fixing the Kinect to the AirSAS setup, allowing both datasets to be collected simultaneously. A future project may yield better results with this type of testing set.

Finally, it would be beneficial to migrate the image classifier off of the Fast.ai library for future experiments. While the classifier was useful for this experiment due to its high-level features, the high number of bugs and lack of customizability may be an issue for future experiments.

BIBLIOGRAPHY

[1] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar, "Cats and dogs," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 3498–3505, 2012, doi: 10.1109/CVPR.2012.6248092.

[2] X. MU, "BIRDSONG SYLLABLE DETECTION USING CONVOLUTIONAL NEURAL NETWORK," The Pennsylvania State University Schreyer Honors College, 2019.

[3] L. Fausett, "Fundamentals of Neural Networks Archiiitectures, Algorithms, and Applications," pp. 1–476, 1994.

[4] R. Ahmed, "Raiid Ahmed Determining Music Genre Preferences Using Neural Networks CMPEN 497: Introduction to Physics Assisted Neural Networks."

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," *2009 IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 248–255, 2010, doi: 10.1109/cvpr.2009.5206848.

[6] R. Hansen, *Introduction to Synthetic Aperture Sonar*. Intech.

[7] T. E. Blanford, J. D. McKay, D. C. Brown, J. D. Park, and S. F. Johnson, "Development of an in-air circular synthetic aperture sonar system as an educational tool," *177th Meet. Acoust. Soc. Am.*, vol. 36, no. 3, p. 070002, 2019, doi: 10.1121/2.0001025.

[8] J. Hu, L. Shen, and G. Sun, "Squeeze-and-Excitation Networks," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 7132–7141, 2018, doi: 10.1109/CVPR.2018.00745.

[9] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 3642–3649, 2012, doi: 10.1109/CVPR.2012.6248110.

[10] M. Roos, "Deep Learning Neurons versus Biological Neurons," *Towar. Data Sci.*, 2019.

[11] M. Smieja, Ł. Struski, J. Tabor, B. Zielinski, and P. Spurek, "Processing of missing data by neural networks," *Adv. Neural Inf. Process. Syst.*, vol. 2018-Decem, no. Section 4, pp. 2719–2729, 2018.

[12] B. Kieffer, M. Babaie, S. Kalra, and H. R. Tizhoosh, “Convolutional neural networks for histopathology image classification: Training vs. Using pre-trained networks,” *Proc. 7th Int. Conf. Image Process. Theory, Tools Appl. IPTA 2017*, vol. 2018-Janua, pp. 1–6, 2018, doi: 10.1109/IPTA.2017.8310149.

[13] J. Howard and S. Gugger, “Fastai: A layered api for deep learning,” *Inf.*, vol. 11, no. 2, pp. 1–27, 2020, doi: 10.3390/info11020108.

[14] “Classification of Dermoscopy Images using Deep Learning ND Reddy, BS 1 1.”

[15] L. Deng, “The MNIST database of handwritten digit images for machine learning research,” *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, 2012, doi: 10.1109/MSP.2012.2211477.

[16] J. D. Park, T. E. Blanford, and D. C. Brown, “Late return focusing algorithm for circular synthetic aperture sonar data,” *JASA Express Lett.*, vol. 1, no. 1, p. 014801, 2021, doi: 10.1121/10.0003063.

[17] D. P. Williams, “On the Use of Tiny Convolutional Neural Networks for Human-Expert-Level Classification Performance in Sonar Imagery,” *IEEE J. Ocean. Eng.*, vol. 46, no. 1, pp. 236–260, 2021, doi: 10.1109/JOE.2019.2963041.

[18] Y. M. G. Costa, L. S. Oliveira, and C. N. Silla, “An evaluation of Convolutional Neural Networks for music classification using spectrograms,” *Appl. Soft Comput. J.*, vol. 52, pp. 28–38, 2017, doi: 10.1016/j.asoc.2016.12.024.

[19] D. C. Brown, “MR-2545 : Sediment Volume Search Sonar,” no. May, 2018.

ACADEMIC VITA

Raiid Ahmed
raiidahmed@gmail.com

Education

B.S. Mechanical Engineering
The Pennsylvania State University Schreyer Honors College — May 2021

Work and Research Experience

General Electric – Schenectady, NY — June 2020 – August 2020
Manufacturing Intern
- Increased efficiency by writing Python scripts to extract data from a 40-year-old inventory database
- Led a team to complete several statistical audits in preparation for 2020 Product Inventory Day
- Proposed a new shipping configuration for gas generators in order to reduce shipping costs

Applied Research Laboratory – State College, PA — June 2019 – Present
Mechanical Engineering Intern
- Writing an honors thesis on the effectiveness of organizing sonar responses using machine learning
- Designed sensor mounting hardware in SolidWorks and implemented a weather station with Arduino
- Used MATLAB to visualize and beamform acoustic waveforms

THRED LAB – State College, PA — August 2018 – December 2018
REU Undergraduate Researcher
- Conducted literature review and acquired CERI Research grant for the Fall 2018 Semester
- Developed Python Script to navigate and create a functional basis from an Excel patent database
- Analyzed functional basis vectors of individual patents using cosine similarity

Extracurricular Involvement

Penn State Digi Digits – State College, PA — May 2020 – Present
President
- Leading a team of 20+ students in the service and design of 3D printed prosthetic devices
- Integrated 3D printers into a wireless network using a headless Raspberry Pi
- Leading project to create a fully operational 3D printed hand operated through an Arduino

Phi Kappa Sigma – State College, PA — April 2019 – May 2020
Academic Vice President
- Maintained and updated a comprehensive academic file of all fraternity members
- Facilitated the completion of all IFC education programs by all fraternity members
- Implemented a program incentivizing academic improvement and continued achievement

Tecnun Universidad De Navarra - San Sebastián, Spain — May 2018 – June 2018
Study Abroad
- Completed an intensive course on the fundamentals of engineering design on international teams
- Conducted customer discovery research on local demographics while crossing a language barrier
- Designed and 3D printed a prototype for a bike seat fixing issues cited by potential customers

Coursework Highlights

Physics-Assisted Neural Networks, Computational Methods for Engineering in MATLAB, Fluid Dynamics, Heat Transfer, Race and Ethnic Relations, Rhetoric in Civic Life