THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

THE ACCURACY OF RHYTHM RECOGNITION WITH CONVOLUTIONAL NEURAL NETWORKS ON TRUENORTH PROCESSOR

ERIC GALANTE
SPRING 2018

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

Reviewed and approved* by the following:

Vijaykrishnan Narayanan
Distinguished Professor of Computer Science and Engineering
Thesis Supervisor and Honors Advisor

John Sampson
Assistant Professor of Computer Science and Engineering
Faculty Reader

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

Convolutional Neural Networks (CNNs) are widely used for image processing and have shown a high accuracy in recognizing properties of inputs in order to classify them [5]. However, these CNNs can also be altered to work with other data inputs, such as sound files. In this research, we explore the accuracy and efficiency of training a CNN to recognize 7 classes of drum rhythms. In particular, we explore altering the existing Eedn training code to work with a dataset comprised of 15 recordings of each class and training the network accordingly. We demonstrate that Eedn network can recognize the drum rhythms with an accuracy that increases in conjunction with the size of the data width. Because of the nature of a drum, each hit is basically indistinguishable from another, providing a difficult dataset to learn. Therefore, as the data width increases, each data point represents a longer amount of time for each audio sample and the accuracy increases because the network has the ability to recognize more patterns in the rhythm. However, this increase in data size also increases the memory required to handle the dataset, with the ideal size exceeding well over 100GB. This leads to the inability to test the network to its full extent, which would result in a fully trained rhythm recognition network.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank Professor Vijaykrishnan Narayanan for helping to form the idea for my research as well as introducing me to the process of Neuromorphic Computing. He has been a guide to me throughout this entire experience and has always kept me on the right track, helping with any problem that I may have had.

I would like to thank Jinhang Choi, graduate student at Penn State University, for teaching me about the specifics about of the Eedn and Convolutional Neural Networks in general. Without his expertise, none of this could have been possible.

Finally, I would like to thank my friends and family for their constant support and encouragement throughout the course of my education. In particular, I would like to thank my good friends Zachary Wiegand and Steven Hatten for helping me record the drum rhythms used in this research.

This work was supported in part by NSF Expeditions in Computing Program: Visual Cortex on Silicon CCF 1317560.

The views and findings expressed in this work are those of the author and not associated with the funding agencies.

Chapter 1

Introduction

Throughout the past several years, audio and speech recognition has become an increasingly prevalent part of the modern computing world. As speech recognition gets closer and closer to 99% accurate, many experts believe that it will become the main way that people interact with computers. By 2023, the speech and voice recognition market is projected to be valued at $18 billion, which is about a 300% increase from its current value today [13]. Every day, more and more consumers are speaking to devices made by the major technology companies, such as Apple, Google, and Amazon. Voice and speech recognition truly is the future, and companies are focusing their efforts on advancing towards it.

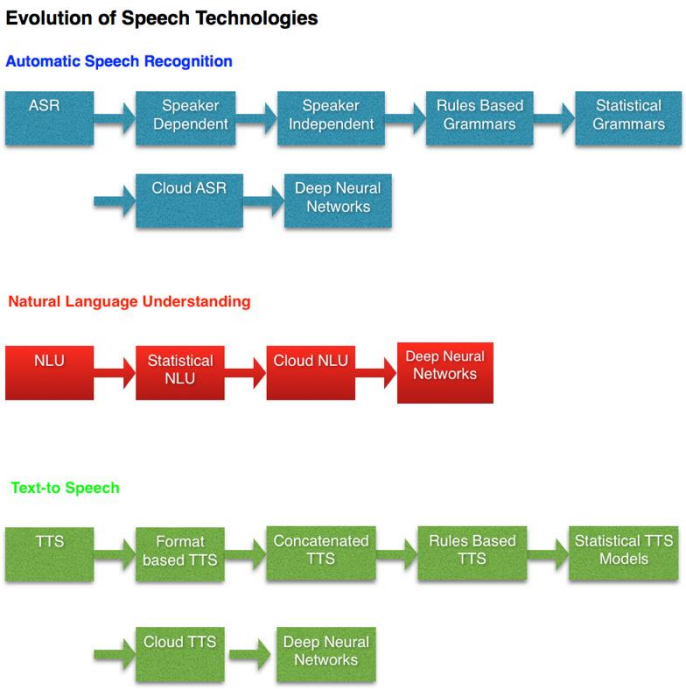

Figure 1. Evolution of Speech Technologies (Adapted From [11])

One of the most prominent uses for speech recognition is search. Comscore projects that by 2020, about 50% of searches will be done by voice [4]. Digital assistants such as Apple's Siri, Amazon's Alexa, and Google's assistant make it incredibly easy for people to search for information quickly with their voice, and this ease of use is the main reason for their rise in popularity.

The driving force behind these smart assistants and devices in the increase in the use of deep neural networks. Neuromorphic computing provides architecture modeled after the human brain and can solve a wide array of recognition problems while still meeting energy, size and speed requirements [7]. By joining the memory and computation units, it eliminates the bottleneck effect that occurs in traditional computers. Because of their structure, neuromorphic architectures can effectually implement Neural Networks used for image and audio recognition.

One type of neural network typically designed for image recognition is the Convolutional Neural Network (CNN). CNNs are a multilayer neural network whose layers are neurons that perform a convolutional filtering first on the input, then on the output of the previous layer [7]. By using a CNN over hardware designed with neuromorphic architecture, the results can very accurate without using an obscene amount of energy. Although inputs for a CNN are typically image-based, we chose to use one to work with audio signals. In this research, we use a modified CNN originally designed to be used with the CIFAR-10 images to explore the accuracy rate when seven classes of drum rhythms are applied over the TrueNorth hardware. We recorded fifteen different samples for each of the seven classes and extracted a feature set able to be used as appropriate data. We modify the dataset and labels in order for them to be used with the existing LMDB framework. Once the dataset was successfully configured, we successfully train and test the network. We observe the changes in accuracy when there is an increase in data value

width, representing the time duration of the audio data. For example, when data width increases from 100ms to 200ms, it results in an increase in accuracy. We also explore the effects of changing the learning rate for training and how that affects the learning curve and overall accuracy of training and testing.

We begin with a review in Chapter 2 of all topics essential to understanding the research, including fast Fourier transforms, neural networks, and CNNs. Following this, we break down the data configurations and experiments and analyze the results in Chapter 3, and the final chapter consists of the conclusion to this research.

Chapter 2

Literature Review

In this chapter, we summarize the relevant topics from sources that are important in understanding the research performed, such as Fast Fourier Transforms, neural networks, and CNNs.

2.1 Fourier Transform

The Fourier Transform is a powerful tool used in several scientific fields to convert waveform data in the time domain to data in the frequency domain [8]. This can be done on any type of waveform, from sound waves to electrical waves. This mathematical conversion breaks down the original time-based wave into several sinusoidal-based properties, including magnitude, and phase [8]. When these sinusoidal functions are added together, then we can exactly replicate the original waveform. However, the separation of these functions makes it much more manageable than working in the time domain. For example, Figure 1 in [8] displays how plotting the amplitude versus the frequency of 5kHZ and 10kHZ sine waves creates a power spectrum, which is the time series response displayed in the frequency domain.

The Fast Fourier Transform (FFT) is a program developed to be a much more efficient version of a normal Fourier Transform. The key to the FFT is speed which it acquires by decreasing the total number of calculations required to analyze the waveform [8]. By restricting

the range of the waveform being analyzed, it increases the speed but can lead to spectral leakage [8].

2.2 Human Neuron Structure and Function

The Human Nervous System is comprised of the Central Nervous System and the Peripheral Nervous System. The Central Nervous System (CNS) is the system in the human body comprised of the brain and the spinal cord and is where the analysis of information occurs [12]. The Peripheral Nervous System (PNS) consists of the sensory and motor neurons found throughout the rest of the body [12]. There are three classes of neurons within these two systems: sensory, motor, and interneurons. All of these neurons have the basic function to receive signals, determine whether these signals must be passed along to another neuron, and to communicate signals to neighboring cells in the body [12]. Figure 2 in [12] shows the basic structure of human neurons and their many parts, including dendrites, axons, synapses, and soma.

The cell body of a neuron is called the soma, which contains the nucleus and produces protein for the neuron. Dendrites are responsible for receiving and processing inbound signals which can either fire the neuron or keep it from firing. Axons, on the other hand, are responsible for sending signals and split up into several branches known as axon terminals. These terminals make connections on target cells. Synapses are the connections between the axons of the sending cell and the dendrites of the receiving cell where information is transferred [12].

2.3 TrueNorth Neurosynaptic Chip

The purpose of neuromorphic computing is to replicate the energy efficiency and fault tolerance of the human brain onto a physical chip. The possibilities of this technology on an architecture that can be very scalable are endless. This is why IBM created the TrueNorth Neurosynaptic Chip: a chip consisting of 4096 neurosynaptic cores with 1 million digital neurons and 256 million synapses connected and controlled by an event routing organization [1]. This powerful chip ensures a one-to-one correspondence between hardware and software with an architecture consisting of 5.4 billion transistors. Figure 2 in [1] shows a graph a bipartite graph on the left representing a small part of a larger neural network. This graph represents the action that takes place on one neurosynaptic core, shown on the right.

Each of these cores contains neurons for computation and memory to store neuron parameters and connectivity. They have input buffers that receive spikes from the network, axons (horizontal lines) that send signals along, and dendrites (vertical lines) that receive these signals. The points where these two meet is the synapses in the neural design. The output of each neuron (triangle) is linked to the input buffer of the axon that it interconnects with [1]. If the axon that it communicates with is in a different core than the neuron, then they communicate over a routing network. All of this computation must in the current synchronization signal (called a tick), which lasts about 1ms [1]. With 4096 of these cores on one chip, each having 256 neurons and 64k synapses, TrueNorth provides a highly scalable and very parallel architecture capable of very quick and low-energy complex computation.

Because of its high configurability, there is a wide variety of networks that can be mapped to the TrueNorth chip. In regards to sound recognition, it has been proven that

TrueNorth can be used for end-to-end audio processing. LATTE is an audio feature extractor designed specifically to make use of the highly parallel computation that TrueNorth offers [14]. This network has been proven to be reconfigurable to satisfy a wide array of energy, speed, and accuracy requirements using the capabilities of the TrueNorth chip [14]. Because of its proven success in audio recognition, we are going use the TrueNorth chip to train and test our network of rhythms.

2.4 Artificial Neural Networks

In order to take advantage of the hardware provided by the TrueNorth chip, there needs to software that contains an appropriate neural structure. An Artificial Neural Network (ANN) is an electronic model based on the neural structure of the brain. Recent biological research shows that the human brain can store information as complex patterns, allowing us to recognize different objects or people from different angles [6]. Thought to be the next wave of modern computing, these networks have the ability to replicate this by storing information as patterns, analyzing them, and using them to solve new problems or recognize data arrangements. There are three different training strategies used in ANNs to learn: supervised learning, unsupervised learning, and reinforcement learning [2]. Supervised learning involves the ANN working with data and making guesses about the answers, at which point the teacher provides the network with the correct answers [2]. The network then compares its answers with the correct answers and makes adjustments. In contrast, unsupervised learning involves searching for a hidden pattern where there is no correct answer provided [2]. Reinforcement learning involves the ANN making

decisions based on its environment and recording the outcome of each decision to be taken into consideration next time [2].

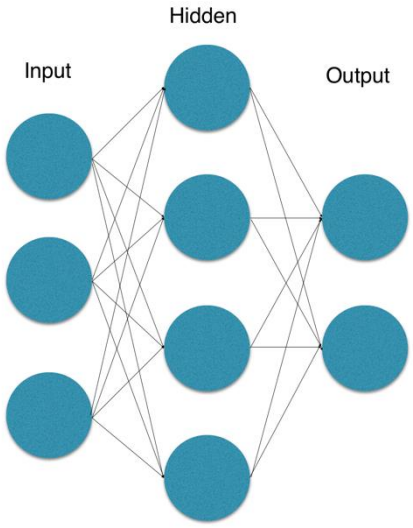

Figure 2. ANN Diagram (Adapted From [3])

Figure 2 displays the structure of a basic ANN. There are three different types of layers: input, hidden, and output. The input layer of neurons receives the data from the input files. Following this layer, there may be several hidden layers. These layers receive signals from the layers above them and transfer the information along to the next layer [6]. Each of the nodes in every layer is associated with a transfer function and a weight value. The weight value is added to the value that is received from the previous layer, and the transfer function is applied, producing an output that is sent along to the next layer [6]. The output layer receives the final product and sends it along to either the outside world or to a secondary computing procedure. An adjusted version of this simple ANN model is the feedback ANN. The key difference here is that the output of one layer is compared to the desired output and then it routes back to a previous layer with an adjusted weight [6]. This helps eliminate any discrepancies in output data.

2.5 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are similar to regular neural networks, but they have neurons arranged in three dimensions: width, height, and depth [5]. CNNs are typically designed for images, but they may be used for other types of inputs. CNNs benefit in the fact that they contain the strong ability to recognize spatial relation within their feature space [5]. The neurons in each layer are only connected to a small region in the previous layer, rather than all of the neurons being connected. Figure 1 in [5] shows how a CNN is a sequence of different layers, each converting one volume into another through a differentiable function [5]. There are five main types of layers in a CNN: input, convolution, pooling, RELU, and fully connected.

- The **input layer** contains the raw input values, in the shape of a three-dimensional value matrix.
- The **convolutional layer** performs much of the heavy lifting, by computing the dot product between the weights and a small region of the input volume. There are three main parameters in this layer:
    - Depth: referring to the number of filters being used to convolve
    - Stride: referring to the number of data input values being skipped between each dot product
    - Zero Padding: the layer of zeros that border the input volume used to control the output size.

  It applies a filter to the data by sliding (convolving) each filter across the dimensions of the input volume. While sliding it computes the dot products between the filter and input at any position of the input matrix [5]. A 2D activation map is created after convolving

the entire input volume which will give the response of that filter at every spatial point [5]. Once convolution is done throughout the entire convolutional layer, the stack of 2D activation maps created will be stacked along the depth (third) dimension, producing the output volume [5]. The convolution layer is summarized by the following equation [5]:

Given input size W, stride S, filter size F, number of filters K, zero padding P:

Output is given by (W-F +2P)/S + 1

- The **pooling layer** gradually reduces the spatial size of the data depiction in order to reduce the amount of total computation throughout the network [5]. The pooling layer is most commonly used for down sampling its input volume to reduce the total amount of parameters.
- The **ReLU** (rectified linear unit) layer applies an activation function, such as max(0,x) (x is the input to a neuron) in an elementwise manner, so that the volume remains the same [5].
- Finally, the **fully connected** layer is similar to a regular neural network layer, such that is has full connections to the previous layer's activations [5]. The activations of the fully connected layer can then be computed using matrix multiplication.

Because of its strong capability to recognize spatial relation within a dataset, we chose to use a CNN with a time-dependent audio dataset. In this research, we used a modified version of the energy efficient deep neuromorphic (Eedn) CNN in order to accept the input data being drum rhythms, rather than images. Now that we have completed the review of literature used in this research, we will begin discussing the experiments and analysis.

Chapter 3

Experiments and Analysis

This chapter will describe the experiments conducted using the dataset built of 15 samples of 7 different drum rhythm classes and a modifies version of Eedn's Convolutional Neural Network. We then present the results of the training and testing and analyze them. We start by explaining how the dataset was constructed and then examine the experiments and analysis.

3.1 Network Description

The platform used throughout this research is the Eedn network. This network using the TrueNorth hardware to create convolutional networks whose neurons and weights have been adapted to run interface tasks [7]. This provides high performance of the network with the energy efficiency obtains by the unique neuromorphic hardware of TrueNorth. Eedn uses binary neurons with trinary (-1,0,1) synapses to adapt backpropagation in order to incorporate energy efficient neuromorphic dynamics into the network [7]. The Eedn network consists of 15 layers, where each layer extracts features from the output of the previous layer. The weights are altered using backpropagation in order to reduce the error between the correct label and the network's guess. The network also has a learning rate used to control how the weights will be adjusted, while the learning rate decay controls how much the learning rate decreases over time as the accuracy increases.

The first layer in the Eedn network is the data layer which consists of the 7 classes of input audio data. The convolution layer is then used to transform the input data into spiking representation using binary channels. Then, the neuron layers start with a preprocessing layer and follow up with a combination of pooling, topological, and network-in-network (NIN) layers. Network-in-Network layers are used to enhance local patch model refinement in the receptive field by using average pooling over feature maps, which makes it easier for the classification layer to interpret [10]. All of these layers vary in filter size, stride, padding size, and number of groups. Finally, the predict layer is used to classify the input volume into one of the 7 classes of rhythms. Although we have modified items such as the stride and filter size of each layer, the overall structure of the network remains similar to that of the original version meant for the CIFAR-10 image set.

3.2 Feature Extraction

The dataset used in this research is comprised of 7 different classes of snare drum rhythms, ranging from simple eight notes to more complex roll patterns. Each of the rhythms was recorded 15 times using Apple's Garageband, each at 140 beats per minute (BPM).

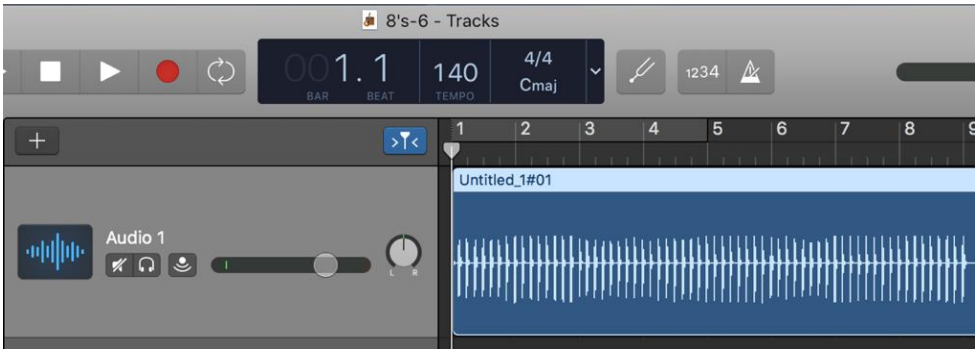

Figure 3. Sample Audio Recording

Each of the individual audio files was then converted into a .wav file and uploaded to the Micro Design Lab server. From there, the data needed to be formatted in a way that we could have some sort of feature extraction. To accomplish this, we started with a function that is used to simplify speech signals: the MFCC function.

The mel frequency cepstral coefficient (MFCC) function computes the coefficients from a speech signal at a certain sampling frequency, where the signal is then run through a short-time Fourier transform analysis of a set frame duration and frame shift [15]. After running a few of the audio files through the function we were able to produce the following graph:

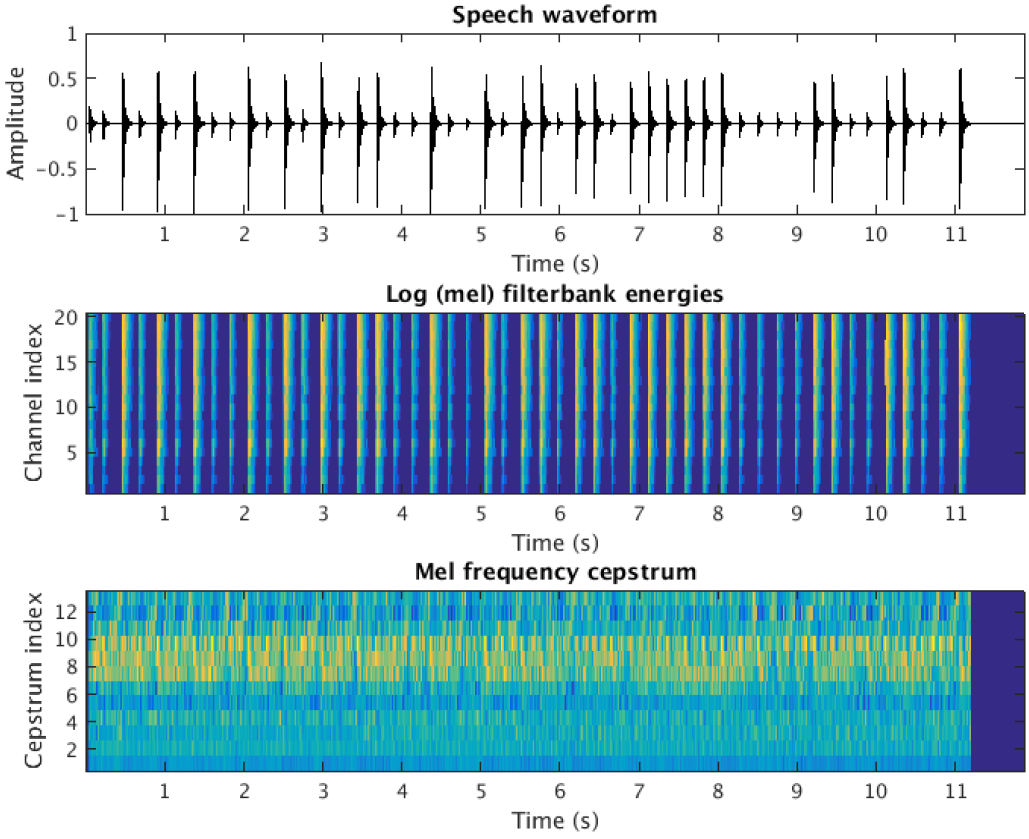

Figure 4. MFCC Output

Figure 4 shows the original waveform of the audio file, the mel filterbank energies, and the MFCCs of the waveform. Since there was no distinguishable feature in the MFCC graph that

related to the structure of the original waveform, we could not use it for our feature extraction. Fortunately, there another feature extraction method built into the MFCC function: Magnitude Spectrum Computation. This calculation consists of the Fast Fourier transform of the frames of each of the of the audio files as column vectors [15]. The output of the function is presented as rows with a constant length of 2048 representing the length of the Fourier transform and columns representing time in centiseconds (1 column for every 10ms). When plotted, the magnitude spectrum is as follows:

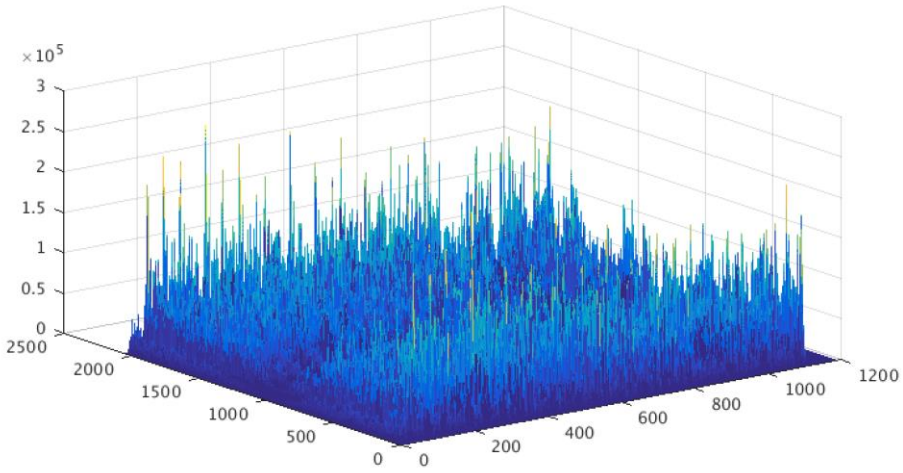

Figure 5. Magnitude Spectrum Example 1

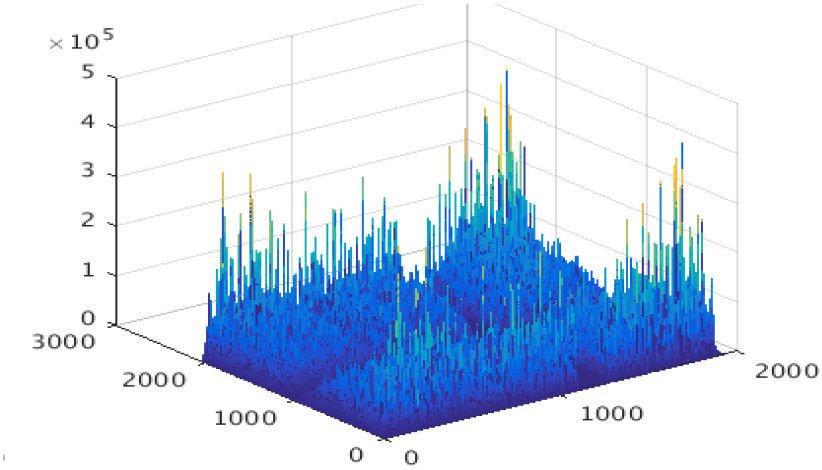

Figure 6. Magnitude Spectrum Example 2

Figure 5 displays an example plot of an accent pattern that is about 11.5 seconds long, and Figure 6 displays a sample plot of a rhythm pattern that is about 19.3 seconds long with a four-count rest at around 11 seconds in. It is clear that there are distinguishable features between the two plots, meaning that we are able to use the magnitude spectrum for each audio file as our feature extraction. This data extracted using the MFCC function provides a feature set with strong spatial relations, making it perfect to be used with the CNN in the Eedn network.

3.3 Building the Dataset

Now that we have a feature extraction for the data, the next step is to build the dataset. Since we are using a modified version of the Eedn network, we modified the th_cifar_dataset.m file in order to fit our audio dataset. The most important part of the dataset composition is how each data value is structured. When testing audio, the sample signal could start at any point throughout the duration of the rhythm. Therefore, we want to eliminate time as a factor throughout our training and testing of the dataset. To accomplish this, we concatenate each of the 15 feature sets for each class into one file resulting in 7 different data files. From there, the data is trimmed of any zero elements and is split into two parts: 90% for training and 10% for testing. This is the basic principle of hold-out validation and is a simple and effective way to gain a test set of data. From there, the training and testing data for each of the 7 classes is reshaped in a way that it overlaps. For our first trial, there are 196,688 total data values and each of the data points are 10 columns (100ms) long, the first data point would go from column 1-10, the second data point would go from column 2-11, and this would continue until there are 196,678 total data points, each with a shape of [10 2048 1]. The purpose of this is to improve the overall accuracy

by ensuring that multiple data training points can represent a small chunk of each audio sample, which should make testing more efficient. Following this, we create the labels (1-7) for each of the 7 classes by creating an array for each class with the length of each individual dataset. This is done for each class's training and testing dataset. All the pieces are now created, so the total training and testing datasets and their labels are all concatenated together.

Now that the training and testing dataset has been structured, the training dataset must be randomized so that it can be properly tested. To do this, we create an incrementing index array with a size equal to the size of the training dataset and randomize it. We then loop through the training data and training labels and reassign them such that:

```
new_rand_dataTraining{i} = new_dataTraining{rand_index(i)};
new_rand_labelsTraining(i) = labelsTraining(rand_index(i));
```

After the dataset is randomized and shaped correctly, it is ready to go through preprocessing. The preprocessing for the Eedn consists of a scaling of the dataset with the largest value so that all of the data point values are between 0 and 1. This allows for more efficient learning during training. Once the training and testing dataset have both been scaled, the data is now ready for training.

3.4 Experiments

3.4.1 First Test: Input Size 10X2048X1

As mentioned above, the Eedn network consists of 15 layers beginning with preprocessing and continuing with an assortment of topological, NIN, and pooling layers. In each of these layers, the filter being applied is some type of square (3x3, 4x4, 2x2, etc), as the network

is meant to be used with 32x32 images. However, since the dataset we are working with has a shape of 10x2048, we altered these filters to be more of a rectangular shape (2x4, 2x8, etc) in order to better represent the overall shape as it moves through the layers. We also needed to adjust the stride and padding for each layer in order to cut down on core usage, as the estimated usage of cores for the first attempt was about 33,000 (TrueNorth only has 4096 cores). By increasing the striding pattern on the preprocessing, pooling, and topological layers, we are able to decrease the number of cores to 3,548, as the total amount of data being processed is decreased tremendously (Note: All of these changes were done using the equation for convolutional layers described in Chapter 2.5, in order to ensure the output of each layer would be an integer). Table 1 shows the structure of each of the layers (excluding NIN layers which remained unchanged), for a two-dimensional training input of (10x2048):

Table 1. Layer Settings for 10x2048x1 Input

Layer	**Input**	**Filter**	**Padding**	**Stride**	**Output**
Preprocessing	10x2048	2x6	2	3	5x683
Pool	5x683	3x5	3	4	3x172
Pool	3x172	3x8	2	4	2x43
Topological	2x43	2x7	2	4	2x11
Pool	2x11	2x7	2	4	2x3
Topological	2x3	2x3	2	4	2x2
Pool	2x2	2x2	0	1	1x1

We also need to change the total amount of training iterations so that the network trains for 10 epochs, where: $1\ epoch = \frac{\#\ iterations * batch\ size}{size\ of\ dataset}$. Therefore, with a batch size of 58, and a dataset size of 196,678, the total number of training iterations we used was 33,910. For the total number of test iterations, we simply divide the test dataset (in this case 21,796), by the batch size (58), so we used an estimated 376 iterations for testing. Table 2 shows the values obtained for initial accuracy, loss, and time over 33,910 iterations, while Figure 7 displays the related graph:

Table 2. Values Obtained With 10x2048x1 Input

Iterations	Average Training Accuracy	Average Testing Accuracy	Loss
33,910	15.27	11.15	5.84

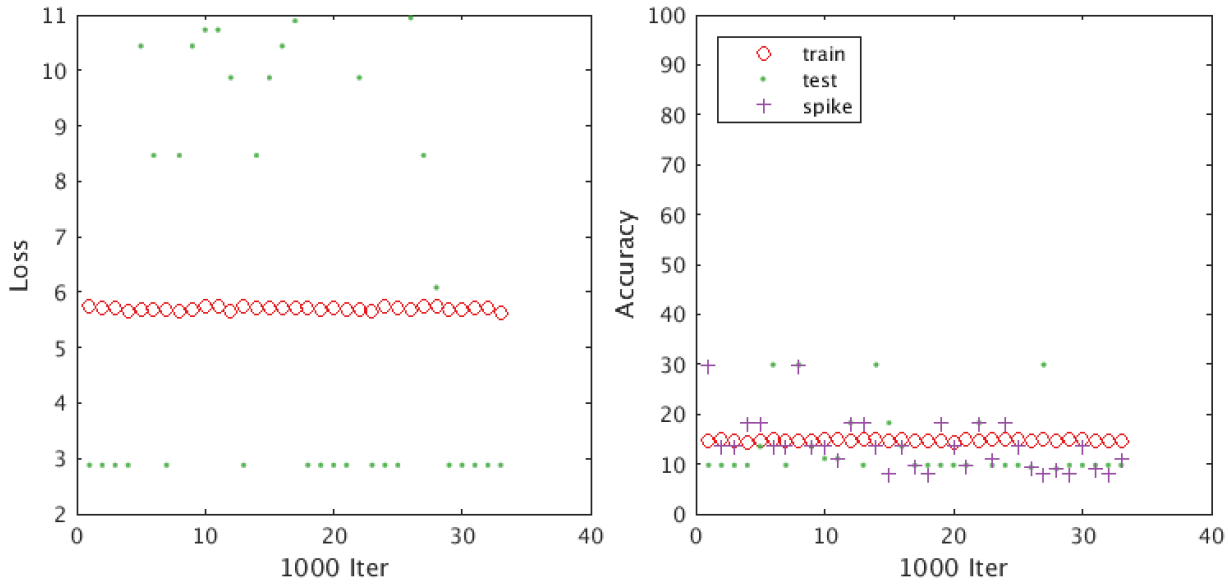

Figure 7. Accuracy and Loss Plots for 10x2048x1 Input with Learning Rate of 20

Clearly, there is no evidence of learning being done by the network, as the training accuracy over 10 epochs is represented by a horizontal line. In order to fix this, we changed the learning rate in order to affect how the weights are adjusted. We began by increasing the learning

rate from 20 (the rate for original CIFAR-10 dataset) to 40 in order to see how the curve would be affected. Table 3 displays the values obtained for this test's accuracy and loss over 33,910 iterations, while Figure 8 displays the related graph:

Table 3. Values Obtained With 10x2048x1 Input with Learning Rate of 40

Iterations	Average Training Accuracy	Average Testing Accuracy	Loss
33,910	13.81	10.68	4.59

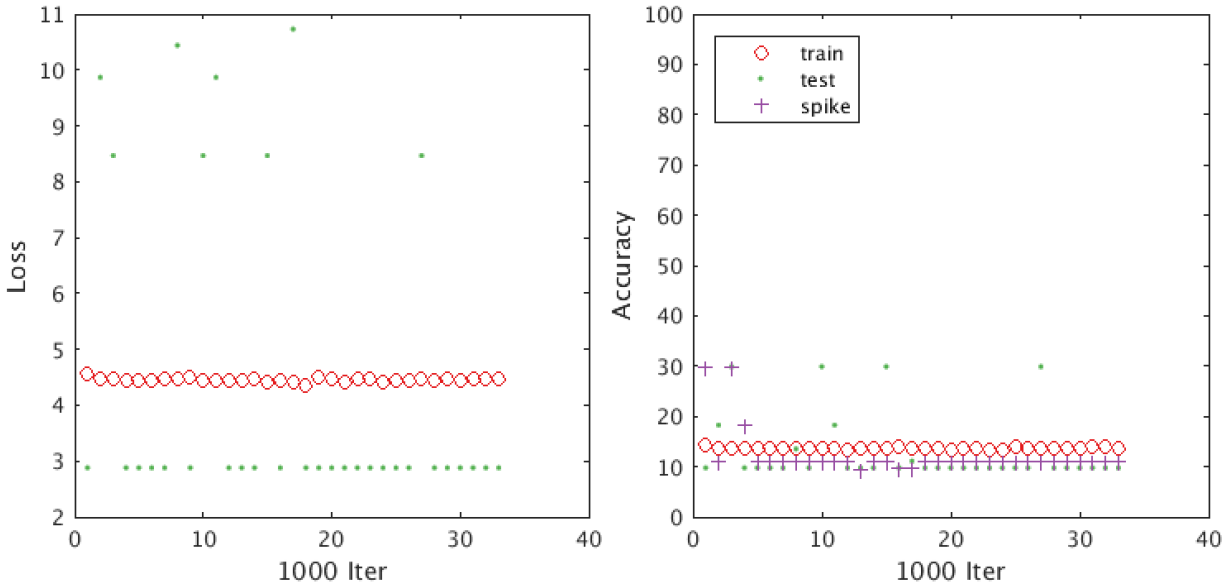

Figure 8. Accuracy and Loss Plots for 10x2048x1 Input with Learning Rate of 40

Since these results were even worse than the original results, we also tested the same dataset with a learning rate less than 20: 0.1. Table 4 displays the values obtained for this test's accuracy and loss over 33,910 iterations, while Figure 9 displays the related graph:

Table 4. Values Obtained With 10x2048x1 Input with Learning Rate of 0.1

Iterations	Average Training Accuracy	Average Testing Accuracy	Loss
33,910	17.34	14.78	6.87

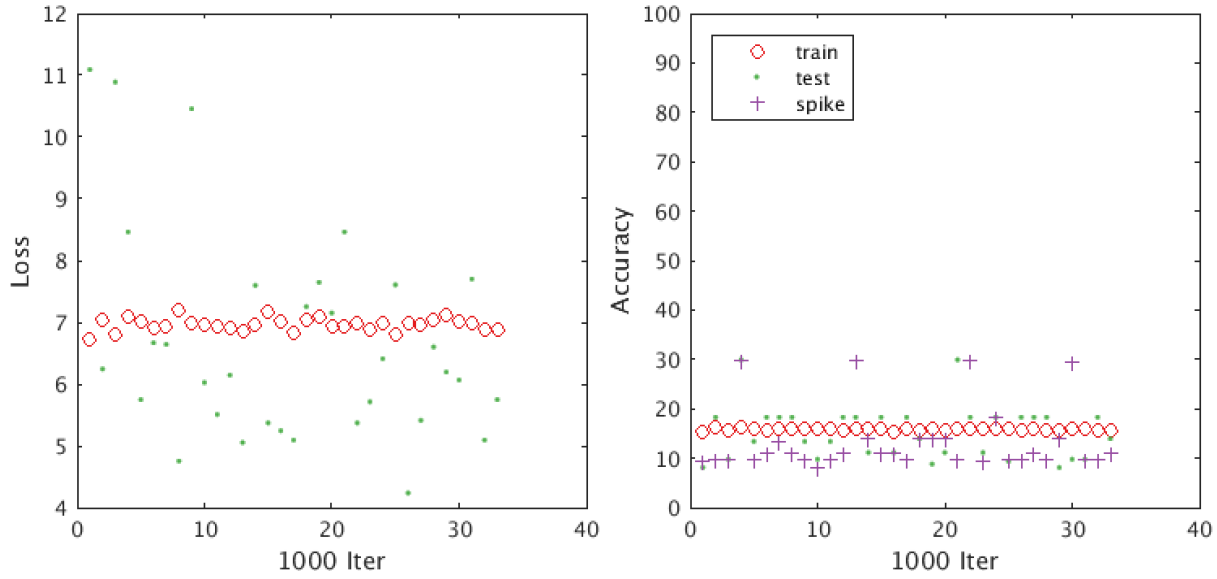

Figure 9. Accuracy and Loss Plots for 10x2048x1 Input with Learning Rate of 0.1

Even though the accuracy is slightly higher for this experiment, there is still no evidence of learning happening in the network. Therefore, we decided to change the size of the dataset in order to increase the chance that the network could recognize patterns between data points.

3.4.1 Second Test: Input Size 15X2048X1

We begin with a small increase in the data size in order to see if there would be any recognizable change in the results. By going back to the dataset file of our network and restructuring the size of each data point, each input size used in this experiment is now

15X2048X1, where the width of 15 represents 150ms of audio data. By changing the input size of each point, the structure of our convolutional layers also must change in order to account for integer output values at each layer, as well as keeping the total number of cores used below the limit of 4096. Table 5 shows the structure of each of the layers (excluding NIN layers which remained unchanged), for a two-dimensional training input of (15x2048):

Table 5. Layer Settings for 15x2048x1 Input

Layer	Input	Filter	Padding	Stride	Output
Preprocessing	15x2048	2x4	1	3	6x683
Pool	6x683	2x7	2	4	3x171
Pool	3x171	3x7	2	4	2x43
Topological	2x43	2x7	2	4	2x11
Pool	2x11	2x7	2	4	2x3
Topological	2x3	2x3	2	4	2x2
Pool	2x2	2x2	0	1	1x1

The total number of iterations also changed because the total number of data points decreased. By keeping the batch-size at 58, and with a new total of inputs of 196,643, for 10 epochs we needed 33,903 iterations. Keeping the learning rate at 0.1, Table 6 displays the values obtained for this test's accuracy and loss over 33,903 iterations, while Figure 10 displays the related graph:

Table 6. Values Obtained With 15x2048x1 Input with Learning Rate of 0.1

Iterations	Average Training Accuracy	Average Testing Accuracy	Loss
33,903	18.26	18.12	7.04

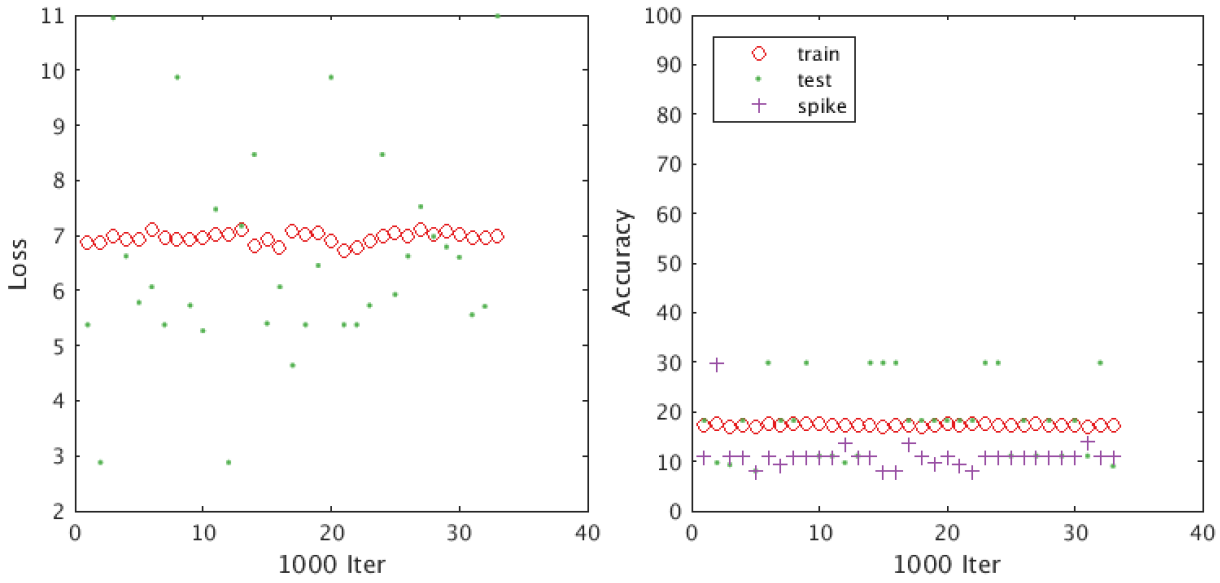

Figure 10. Accuracy and Loss Plots for 15x2048x1 Input with Learning Rate of 0.1

The results of this experiment show a slight increase in training and testing accuracy due to the increase in data point size, even though there is still no evidence of learning. We then increased the data point size even more to look for a more significant change.

3.4.3 Third Test: Input Size 20X2048X1

Since the input width of 15 still showed no signs of learning, we then increased the width to 20, which represents 200ms of audio data. We then restructured the different convolution

layers in order to adjust for integer outputs at each layer, as well as keeping the core usage under 4096. Table 7 shows the structure of each of the layers (excluding NIN layers which remained unchanged), for a two-dimensional training input of (20x2048):

Table 7. Layer Settings for 20x2048x1 Input

Layer	Input	Filter	Padding	Stride	Output
Preprocessing	20x2048	2x6	1	4	6x512
Pool	6x512	2x4	2	4	3x129
Pool	3x129	3x5	2	4	2x33
Topological	2x33	2x5	2	4	2x9
Pool	2x9	2x5	2	4	2x3
Topological	2x3	2x3	2	4	2x2
Pool	2x2	2x2	0	1	1x1

The total number of iterations also changed because the total number of data points decreased. By keeping the batch-size at 58, and with a new total of inputs of 196,608, for 10 epochs we needed 33,897 iterations. Keeping the learning rate at 0.1, Table 8 displays the values obtained for this test's accuracy and loss over 33,897 iterations, while Figure 11 displays the related graph:

Table 8. Values Obtained With 20x2048x1 Input with Learning Rate of 0.1

Iterations	Average Training Accuracy	Average Testing Accuracy	Loss
33,897	19.76	20.98	7.24

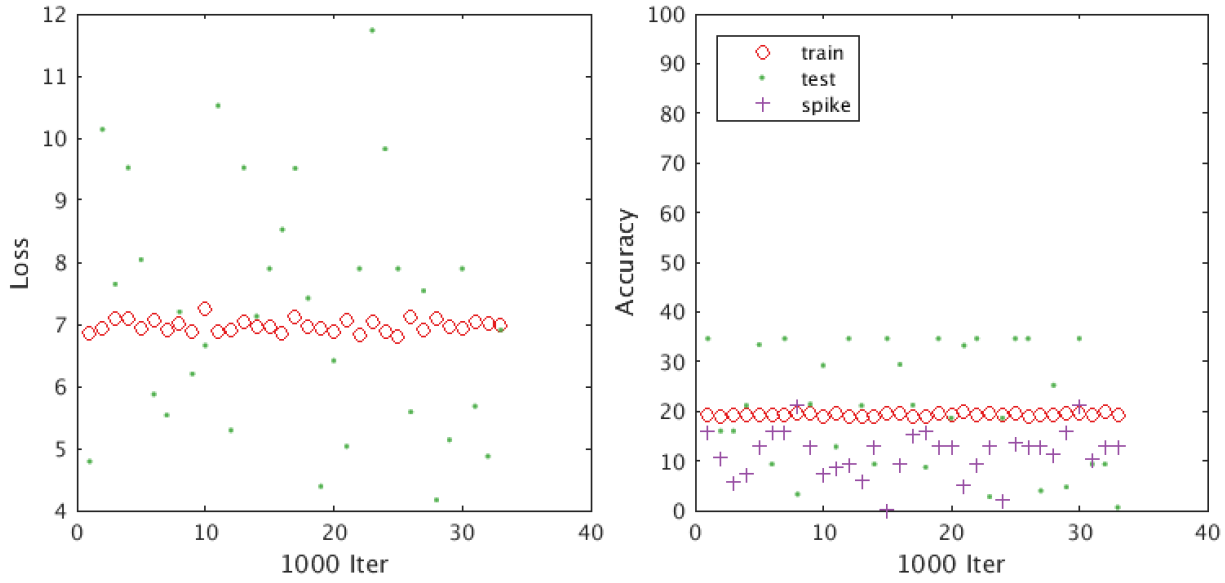

Figure 11. Accuracy and Loss Plots for 20x2048x1 Input with Learning Rate of 0.1

Although there is still no evidence of a learning curve, the accuracy of the tests is increasing as the data point width increases. Therefore, it is clear that we are headed in the right direction, as increasing the data width allows for longer audio samples for each point. This allows for the network to better recognize patterns in the audio.

3.4.5 Further Experiments

Upon trying to increase the width of each data point further, we hit a road block. The server that we have been using for these experiments only has 64 GB of memory. Because the data that we are working with is as large as it is, when trying to structure the dataset to have widths any size larger than 20 (200ms), the program runs out of the memory. The dataset is simply too large for the server to handle, and we were unable to modify it anymore to improve results. The ultimate goal was to set a data width of 45 (450ms), as the amount of time between

two quarter notes at 140 beats per minute (BPM) is 428.57ms [9]. Because this is the tempo that all of the audio files were recorded at, there would have been a definite distinction between data points of this size. Unfortunately, this dataset would have been over 100GB in size, which is well over the server limit of 64GB. Therefore, we were unsuccessful in being able to increase the data width to this size with the current resources we are using.

3.5 Analysis of Results

The modified version of the Eedn used in these experiments can recognize the input set of 7 different class of drum rhythms with low accuracy. This lower accuracy is most likely due to the size of the width of the data points being input into the network. By increasing the width of each data point from 10 to 15 and from 15 to 20 (100ms, 150ms, and 200ms respectively), we can see that the accuracy of training and testing also increased. Between a width of 10 and a width of 20, the average testing accuracy increased by 6.2%, as the network receives twice the audio data per input, making it easier to recognize similar patterns. We believe that if the data width could be increased to 45 (450ms), we would receive much higher accuracy ratings, as each data point would be able to encapsulate one quarter note in each rhythm. The network would have a much easier job of recognizing the patterns, and we would likely see some sort of learning curve in the results.

Chapter 4

Conclusion

In this research, we used a modified version of the Eedn network's Convolutional Neural Network (CNN) designed to process the CIFAR-10 image set with IBM's TrueNorth hardware in order to process 7 classes of drum rhythms. The dataset used consists of 15 different samples of each class passed through a Fast Fourier Transform used for feature extraction. We modified the existing dataset function used in the Eedn network in order to construct a dataset containing overlapping data point samples so that the testing of the dataset could start at any point in the rhythm. After the preprocessing of that data was complete, we needed to alter the settings for each of the convolutional layers in the network in order to provide an integer output at each layer, as well as keep the core usage under 4096. The first dataset tested had a data point size of 10X2048X1 and resulted in an average test accuracy of 11.15% with no apparent learning curve. To fix this, we adjusted the learning rate and found that a lower learning rate of 0.1 provided a slightly better accuracy. We then moved on to increase the data point width from 10 to 15 and from 15 to 20 (100ms, 150ms, and 200ms respectively), and found that this increase corresponded with an increase in both training and testing accuracy, as the network received more audio data to help in pattern recognition. This is due to the nature of the drum rhythms we were working with, as one hit is mostly indistinguishable from another. Therefore, the network requires more time for each data point in order to provide a distinguishable pattern to learn. When trying to increase the data size further, however, we were unable to due to the server not

having the required amount of memory needed for such a large dataset. Because of the 140 BPM tempo of each recording, if we were able to increase the data point size to 45 (450ms), this would surely lead to not only a much higher accuracy rating, but a true training curve for the network, as each data point would be able to see at least a full quarter note of audio data. In future work, one would work towards achieving this by working with more available memory in order to display the network's ability to truly learn to recognize the 7 classes of drum rhythms.

BIBLIOGRAPHY

[1] Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip. (2015). *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,34*(10), 1537-1557. Retrieved March 24, 2018, from http://doi.org/10.1109/TCAD.2015.2474396

[2] Artificial Intelligence Neural Networks. (2018, January 08). Retrieved March 25, 2018, from https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_neural_networks.htm

[3] Artificial Neural Network Design. (2017, November 6). Retrieved March 25, 2018, from https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/

[4] Boyd, C. (2018, January 10). The Past, Present, and Future of Speech Recognition Technology. Retrieved March 23, 2018, from https://medium.com/swlh/the-past-present-and-future-of-speech-recognition-technology-cf13c179aaf

[5] Convolutional Neural Networks (CNNs / ConvNets). (n.d.). Retrieved April 09, 2018, from https://cs231n.github.io/convolutional-networks/

[6] DACS. (n.d.). Artificial Neural Networks Technology. Retrieved March 25, 2018, from http://www.psych.utoronto.ca/users/reingold/courses/ai/cache/neural2.html

[7] Esser, S. K., Merolla, P. A., Arthur, J. V., Cassidy, A. S., Appuswamy, R., Andreopoulos, A., et al. (2016, March 27). "Convolutional Networks for Fast, Energy-Efficient Neuromorphic Computing." arXiv.org. National Acad Sciences. http://doi.org/10.1073/pnas.1604850113

[8] FFT (Fast Fourier Transform) Waveform Analysis. (n.d.). Retrieved March 24, 2018, from https://www.dataq.com/data-acquisition/general-education-tutorials/fft-fast-fourier-transform-waveform-analysis.html

[9] LeFevre, N. (n.d.). BPM to MS. Retrieved March 25, 2018, from http://nickfever.com/music/blog/2014/bpm-to-ms

[10] Lin, M., Chen, Q., & Yan, S. (2014, March 04). Network In Network. Retrieved April 06, 2018, from https://arxiv.org/abs/1312.4400

[11] Nuance. (n.d.). Evolution of Speech Technologies [Digital image]. Retrieved March 2, 2018, from https://www.techrepublic.com/article/how-we-learned-to-talk-to-computers/

[12] Overview of neuron structure and function. (n.d.). Retrieved March 24, 2018, from https://www.khanacademy.org/science/biology/human-biology/neuron-nervous-system/a/overview-of-neuron-structure-and-function

[13] Rohan. (n.d.). HOME › Press Releases › Speech and Voice Recognition Market worth 18.30 Billion USD by 2023. Retrieved March 22, 2018, from https://www.marketsandmarkets.com/PressReleases/speech-voice-recognition.asp

[14] Tsai, W., Barch, D. R., Cassidy, A. S., DeBole, M. V., Andreopoulos, A., & Jackson, B. L., et al. (2017, June). Always-On Speech Recognition Using TrueNorth, a Reconfigurable, Neurosynaptic Processor. Retrieved April 09, 2018, from http://doi.org/10.1109/TC.2016.2630683

[15] Wojcicki, K. (2011, September 19). HTK MFCC MATLAB. Retrieved March 27, 2018, from https://www.mathworks.com/matlabcentral/fileexchange/32849-htk-mfcc-matlab?focused=5199998&tab=function

Academic Vita

Eric Galante

eag5357@gmail.com

EDUCATION

The Pennsylvania State University University Park, PA

- *Schreyer Honors College Scholar* Fall 2014-Present
    - Bachelor of Science, Computer Engineering
    - Graduation: May 2018
- *Relevant Course-Work*: Application development (GUIs), systems programming, working in a Unix environment, polymorphism, inheritance

WORK EXPERIENCE

Vanguard Summer, 2017

- Worked as a UI/Mid-tier development intern for the Retail Systems division
- Developed team-based skills in an Agile work environment, partaking in daily Scrums while using tools such as Jira and Confluence
- Used Atlassian tools such as Bitbucket and Bamboo to have a seamless work stream with my team

Camelback Mountain Resort Tannersville, PA

Segway Tour Guide Summer 2014, 2015

- Instructed guests how to ride Segways and conducted tours across Camelback Mt. in the Poconos
- Utilized strong communication skills in providing tours to as many as five groups per day

Ski Instructor Winter 2011 – 2015

- Worked both in a team or individually, teaching levels beginner to advanced, ages 3-adult
- Developed teaching ability, patience, and communication skills in teaching people of all ages and skill levels

SKILLS

- Experience writing AngularJS, HTML, CSS, Jasmine tests, Java, C++, C, Matlab, and Verilog
- Proficient experience with Microsoft Office, Bamboo, Bitbucket, Git, STS, Webstorm, Visual Studio, Netbeans

ACTIVITIES AND HONORS

Blue Band Fall 2014-Present

- Member of the Percussion section in The Pennsylvania State University Marching Blue Band

Blue Band THON Fall 2014-Present

- Independent organization within Penn State's philanthropic dance marathon THON
- Participated in THON activities, such as collecting money through fundraising trips

Eagle Scout, Boy Scouts of America