Neural Neural Network

Major Qualifying Project

Advisors:

XIANGNAN KONG
TIAN GUO

Written By:

RYAN K. RACINE
KARSTEN H. ROBERTS
ISAAC W. WOODS

A Major Qualifying Project
WORCESTER POLYTECHNIC INSTITUTE

Submitted to the Faculty of the Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Bachelor of Science in Computer Science.

AUGUST 2018 - MARCH 2019

Abstract

FMRI is a modern technique employed to help diagnose internal brain injuries. Analyzing the resulting data from them is currently facilitated by medical professionals. Having these professionals involved can increase the inherent cost of this process with regards to time and monetary value. For our project we aimed to address this problem by creating a neural network that can analyze unprocessed fMRI data and identify brain injuries. The result of this was a model that could predict whether a given fMRI contained a concussion with 70% accuracy.

Acknowledgements

We would like to thank our advisor, Professor Xiangnan Kong, for all his help throughout this project. We would also like to thank our Co-Advisor Tian Guo for her advice and contributions throughout the project. Additionally we would like to thank the team managing the ACE machines on campus for their help finding us the resources we needed to run a neural network.

Table of Contents

Page

Abstract **ii**

List of Figures **iv**

1 Introduction **1**

2 Background **2**

2.1 History 2

2.2 Techniques 5

2.3 fMRI Applications 10

2.3.1 Data Pre-processing 11

3 Methodology **12**

3.1 The Opportunity 12

3.2 Techniques 13

4 Results **16**

5 Conclusions and Future Work **17**

A Appendix A: Result Graphs **18**

B Appendix B: Concussion Scan Information **20**

Bibliography **21**

List of Figures

Figure **Page**

1.1 Slicing a four dimensional fMRI scan into set of three dimensional slices, each representing a cross-section over time. 1

2.1 A simple logical neuron 2
2.2 Frank Rosenblatt's perceptron 3
2.3 A multi-layer neural network 4
2.4 Convolutional and Fully Connected neural layers 6
2.5 Common loss functions 6
2.6 Two common activation functions 7
2.7 An example of max pooling 8
2.8 An illustration of the symptoms of over- and under-fitting 8
2.9 An example of cross validation 9

3.1 Example of concussion fMRI scan 13
3.2 A diagram of our model 14
3.3 A graph of the training accuracy of the model 15
3.4 A graph of the test accuracy of the model 15

4.1 A visual representation of the final model structure 16

Introduction

FMRI is a technique that is used to measure blood oxygenation to determine regions of neural activity within a patient's brain[1]. Patients get one generally when there is suspicion of an unseen traumatic brain injury. Data from fMRIs can be analyzed to determine whether a patient has a particular brain injury and the effect of the injury on the brain's function.

Currently analyzing the fMRI data to diagnose injuries is left to medical professionals, usually a radiologist. After a patient gets an fMRI, the fMRI data is sent to a professional who performs analysis on it and gets results. These results are then sent back to the physician or doctor who conveys the diagnosis to the patient. While a computer may aid in the analysis process, the process itself still requires professionals to facilitate it[2]. This creates a promising opportunity to create a fully automated systems that can analyze the fMRI data, which is a growing area of interest [3].

There are several downsides to having medical professionals involved in this process. There is an inherent overhead cost that is incurred by having the medical professional involved. It is an added time and financial cost for patients that could be reduced with the help of emerging computational techniques. On top of this the process itself can still be quite slow. This can be a major issue, especially if it is a critical injury.

For this project we created a convolutional neural network that can analyze unprocessed four dimensional fMRI data in order to aid doctors in determining whether a patient has a brain injury. It looked at 4 dimensional data (fMRI images over time) and made a decision on whether there is an injury or not. We focused specifically on concussions for this project, though the idea and potentially the model could be adjusted to detect other injuries.

Our final model showed promising results. It managed to correctly predict concussions with approximately a 70% accuracy. It used two convolutional layers with maxpooling in between them and then two linear layers followed by softmax. It worked on 4 dimensional fMRI data by looking at slices of the brain over time. While this may not be accurate enough to fully replace the current medical process, it does indicate that something can be done towards this goal.

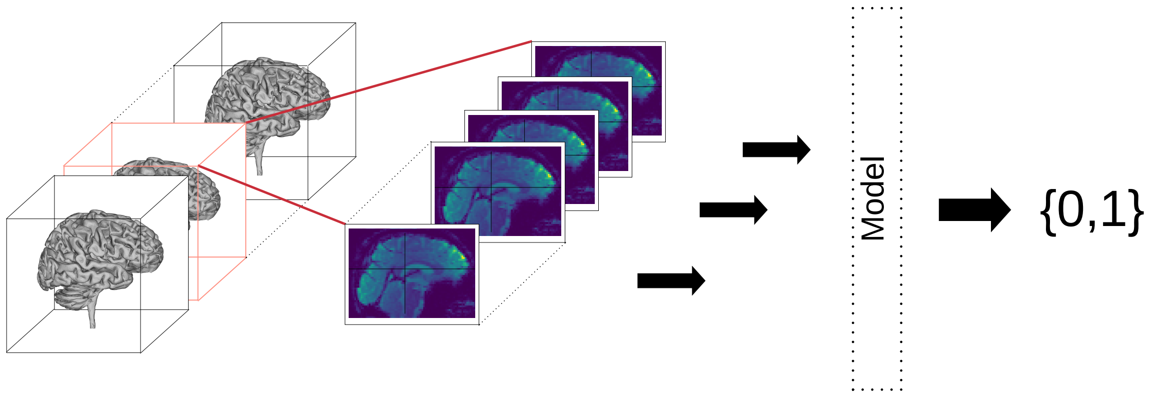


Figure 1.1: *Slicing a four dimensional fMRI scan into set of three dimensional slices, each representing a cross-section over time.*

BACKGROUND

Over the past century, the idea of a neural network capable of transcribing human writing has gone from a dream to becoming an almost trivial task. Neural networks started out as just a mathematical concept, not something that could be done with the technology level of the time, but over time the ideas grew and the technology finally caught up[4].

2.1 History

The progenitor of all modern neural network models was the neuron structure proposed by the neuroscientist Warren McCulloch and mathematician Walter Pitts in 1943. While simple in idea, it was capable of modelling linear separable systems, such as logical operators AND, OR, and NOT. The neuron would be given a list of boolean inputs (either 0 or 1), sum them, and then pass the sum to a threshold function that would return 1 if the sum exceeded the threshold and 0 if it failed to. Each input could be either excitatory or inhibitory. An excitatory input would be one which affects the output together with the state of other inputs, whereas an inhibitory input would be one whose state can determine the output on its own[5].

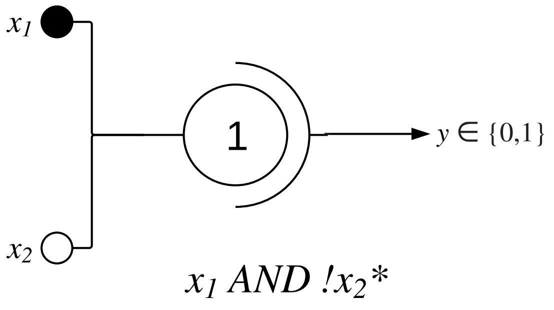


Figure 2.1: *A simple logical neuron*

Figure 2.1 is an example of how a neuron would be set up to compute $x_1 AND !x_2$. By making $!x_2$ an inhibitory input, there are only two possible cases: $x_1 = 0$ and $x_2 = 0$, and $x_1 = 1$ and $x_2 = 0$. Clearly the expression only evaluates to true if $x_1 = 1$, and so case 2 is the only valid one. In order to reflect this, we set a threshold of 1. The sum of case 1 is 0, which fails the threshold and returns 0. However, the sum of case 2 is 1, which passes, and the neuron outputs one. While innovative, there were several limitations the McCulloch-Pitts neuron. Inputs were limited to 0's and 1's, every neuron's threshold had to be coded independently, the input values were all equally weighted during aggregation, and non-linearly separable functions (i.e. XOR) were impossible to model limiting the neurons applications. However, in 1958, the psychologist Frank Rosenblatt improved upon this model by addressing several of these limitations[5].

Frank Rosenblatt's model, known as a perceptron, was the first neuron model specifically proposed with the purpose of building a computer that could learn (Figure 2.2). The idea of the perceptron is very similar to that of McCulloch-Pitts neuron. The perceptron took a list of binary

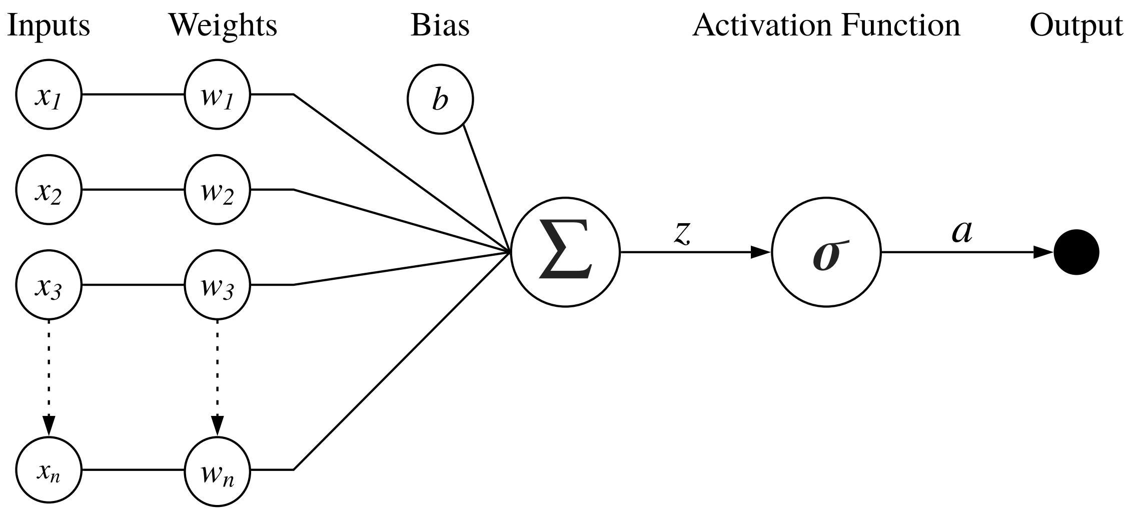


Figure 2.2: *Frank Rosenblatt's perceptron*

inputs, aggregated them, and then passed the aggregation to a threshold activation function that output either 0 or 1. However, there was a key difference[6].

Rather than the inputs being passed directly to the aggregator, they were each multiplied by a scalar called a weight. Each input had its own weight, which was initialized as a random positive number. With this modification, each input could now be weighted independently in proportion to the inputs relative importance to the decision. On top of the inputs given by the user, one other input-weight pair was added known as the bias. The input was always one, but the weight could be modified. This allowed for a constant offset to be applied to the aggregation, increasing the versatility of the perceptron. These modifications allowed for a new concept: training a computer neuron[6].

The process of training the perceptron was fairly simple. First, a training dataset needed to be created. This consisted of a set of sample input sets from the problem domain, known as the training set, and the expected output for each input known as a label. To begin with, all weights are randomized. Then, the first input set would be sent to the neuron and the output would be calculated. If the output was 1 but should have been 0, the weights of inputs with value 0 would be increased. If the output was 0 but should have been 1, the weights of inputs with value 1 would be increased. Then the next input set in the training set would be sent, and the weights modified again. By repeating this process until the neuron always output the correct number, the neuron would eventually be trained[6].

While a single perceptron can only output 0 or 1 and thus distinguish between two states, n parallel perceptrons could distinguish between n states. These parallel perceptrons are called a layer in a network. Each input would be sent to all perceptrons, and their outputs would be compared to a label the same as with a single perceptron. However, this label would be a list n

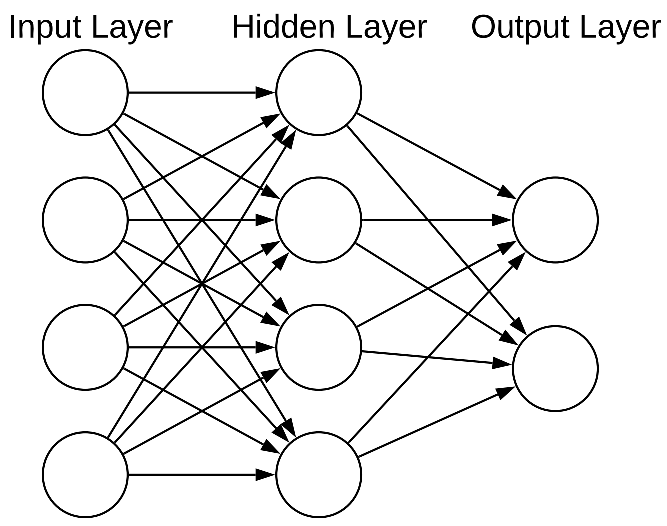


Figure 2.3: *A multi-layer neural network*

long consisting of all zeros except for a one indicating which perceptron represented the correct output. With this method, a network could be trained to distinguish between shapes in 2D images. Although much more powerful than the McCulloch-Pitts neuron, not all limitations had been addressed[6].

The most critical limitation remaining was the fact that perceptrons were still unable to model any problem that was not linearly separable. Thanks to research by M. Minsky and S. A. Papert[7], it was determined to be impossible to model non-linearly separable problems without the use of multiple layers of neurons, or “hidden” layers. Up until this point, models had had a single layer, the output layer, with all inputs being sent directly to every output in the layer. Figure 2.3 illustrates how the new hidden layer would work. According to A. Kurenkov[6],

> “The reason hidden layers are good, in basic terms, is that the hidden layers can find features within the data and allow following layers to operate on those features rather than the noisy and large raw data”.

Without this noise reduction, perceptrons would be a dead end. Perceptrons were unable to be used in multi-layer models because it modified the weights based on the end output of of the whole model. This worked fine when there was a single layer, since there was only one set of weights for each perceptron, and they were the sole modifiers of the input. However, with a second layer, there were suddenly multiple weights modifying each other that affected the output. There was no way to know the outputs of the hidden layer in order to modify those weights each run, and so it couldn’t be trained [6].

It took seventeen years before this barrier was finally overcome in 1986 as described in the paper "Learning representations by back-propagatiog errors" by D. E. Rumelhart, G. E. Hinton, and R. J. Williams [8]. In it, the researchers outline a method for setting weights in hidden layers called backpropagation. "The key realization was that if the neural net neurons were not quite perceptrons, but were made to compute the output with an activation function that was still nonlinear but also differentiable ... not only could the derivative be used to adjust the weight to minimize error, but the chain rule could also be used to compute the derivative for all the neurons in a prior layer and thus the way to adjust their weights would also be known"[6]. The concept of backpropagation opened the way for the development of what is today known as fully-connected and convolutional neural networks that are capable of surpassing human performance at certain tasks.

2.2 Techniques

There are many modern techniques used in current neural networks ranging from types of neural layers, to ways of adjusting the model over time. Each of these techniques have strengths and weaknesses making them optimal for different applications.

One of these techniques is the recurrent neural layer which takes into consideration things previously seen by the model essentially giving it a short term "memory". This "memory" allows it to perform sequence recognition. A model without memory that looks at the numbers 1, 2, 3, 4 and 5 would have no idea what the next number would be because it would not be able to recognize the order or pattern in the inputs. A recurrent neural layer, however, would know what the last few numbers it looked at were, and from that can try to identify the next based on patterns it has previously analyzed. For this reason recurrent neural networks are used heavily in problems involving text or speech data and often performs some sort of prediction or generation task[9].

Another common technique in neural networks is the fully connected layer, which is often used at the end of networks which are focused on types of problems like classification. They are used to convert the number of output channels from the layer before it to the number of output channels needed to classify (1 output channel for each choice). One benefit of fully connected layers is that they give the model a lot of flexibility because each bit of input data can directly have an effect on each bit of output. However, this means that the model must have the number of inputs multiplied by the number of outputs connections for that layer. When the input data is large, calculating and maintaining a weight for each of these can have a high time and space cost. A more resource efficient option for dealing with data on a larger scale is the convolutional neural layer. A convolutional layer will relate bits that are close to each other in the input by outputting one bit per input grouping. These groupings can be any size but are generally consistent across the model. For example, in the convolutional layer in Figure 2.4 the kernel size is three so the first three input bits (1, 2 & 3) are the sole input to the first output bit, the next three input bits (2, 3 & 4) are the sole input to the second output bit and so on. This significantly reduces the number of connections in the layer compared to a fully connected layer. The number of outputs

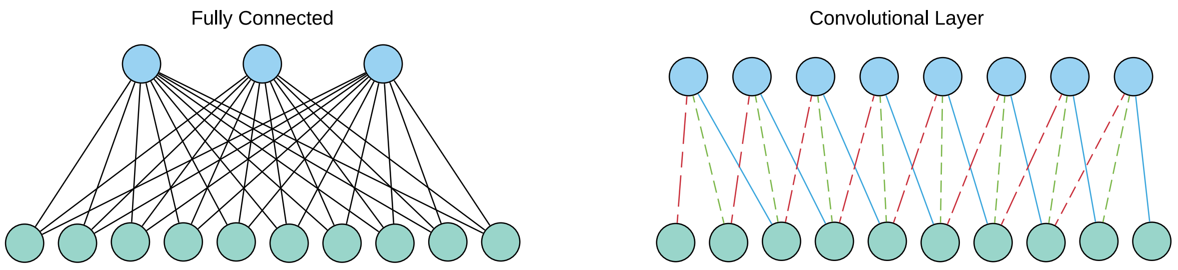


Figure 2.4: *Convolutional and Fully Connected neural layers*

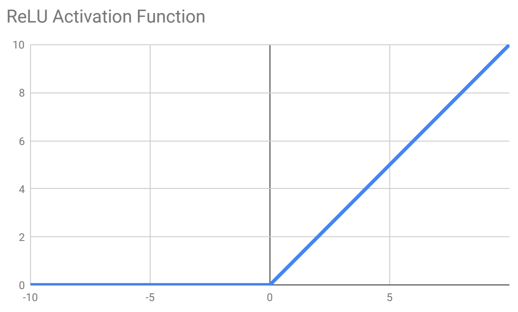
multiplied by the kernel size is better than the number of outputs multiplied by the input size because a reasonable kernel will be smaller than the number of inputs. The larger the kernel size, the less focus on each individual bit in the area and more focused on the general kernel area as a whole it becomes. Making use of this method enables convolutional neural layers to reduce the size of the data in a productive way that avoids losing to much information.

Another technique integral to neural networks is the loss function. Loss functions calculate a loss from the predicted value and the actual value. They are critical to any neural network as they influence training through backpropagation. Backpropagation determines how much the model's weights should change based on the predicted and actual values. Figure 2.5 shows the formulas for a few of the more common loss functions. Mean Squared Error (MSE) penalizes the model heavily for larger differences in the predicted and actual values. There is a logarithmic version of this (Mean Squared Logarithmic Error) that can be used in cases where you want to avoid penalizing it too much for huge differences. Mean Absolute Error (MAE) is more generous to outliers as well since it does not square the difference between the predicted and actual values at all. Both MSE and MAE are used most commonly in predictive modeling. Cross Entropy loss is commonly used in classification problems (the equations shown in figure 2.5 is for binary classification, there is also a multi-class version). Negative Log Likelihood is similar to Cross Entropy mathematically and is a very widely used loss function.

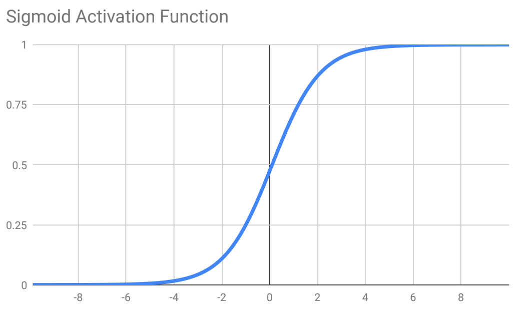
Activation functions are a technique used between layers to convert the outputs, which could be anywhere between negative and positive infinity, into more applicable ranges like -1 to 1, 0 to

Mean Squared Error	Loss	$= \frac{1}{n} \sum_{i=1}^{n} (y^i - \hat{y}^i)^2$
Mean Absolute Error	Loss	$= \frac{1}{n} \sum_{i=1}^{n} \lvert y^i - \hat{y}^i \rvert$
Cross Entropy	Loss	$= -\frac{1}{n} \sum_{i=1}^{n} [y^i \cdot log(\hat{y}^i) + (1 - y^i) \cdot log(1 - \hat{y}^i)]$
Negative Log Likelihood	Loss	$= -\frac{1}{n} \sum_{i=1}^{n} log(\hat{y}^i)$

Figure 2.5: *Common loss functions*



(a) $R(z) = max(0, z)$



(b) $\sigma(z) = \dfrac{1}{1+e^{-z}}$

Figure 2.6: *Two common activation functions*

1, or 0 to infinity. They can also be used to perform some other conversions on the data. Sigmoid will convert all negative numbers into a 0 to 0.5 range and all positive numbers into a 0.5 to 1 range (as seen in figure 2.6). Tanh, another commonly used activation function, is very similar to sigmoid except its range is -1 to 1 instead of 0 to 1. ReLU is a commonly used activation function; it converts all negative values into zero. This can cause an issue called neuron death. Neuron death happens when the output of the neuron starts being less than zero consistently. When this happens the entire neuron will output zero and will have a hard time recovering from this state since the gradient for any value less than zero in the ReLU function is 0. To address this issue there is another activation function called Leaky ReLU that multiples all negative values by a factor between 0 and 1 essentially reducing the influence of the negatives without fully eliminating it.

A common technique to reduce the size of the data, and thus performance, is pooling. Maxpool in 2 dimensions works as shown on the right in Figure 2.7. The filter size and the stride tells it which sections to look at and for each section it simplifies it into just one output that is the largest value from the whole section. This helps preserve as much of the data as possible while still reducing the size of the data that the model has to process. This technique is commonly used in between layers to keep reducing the size of the data.

Hyperparameters are used to tune a model to successfully learn from training data[10]. Hyperparameters like learning rate can sometimes be hard to optimize. Often a particular learning rate can enable the model to approach a certain level of accuracy, but cannot continue to improve without lowering the learning rate. However, if you begin with a low learning rate the model will take significantly longer to train. So to help the model train faster while still allowing it to improve later a technique called learning rate decay is used.

A problem that some neural networks can have is overfitting, especially those with small datasets[10]. Overfitting is when the model latches onto some patterns in the data that allow it to make accurate predictions for the training set, but do not necessarily hold true for the testing

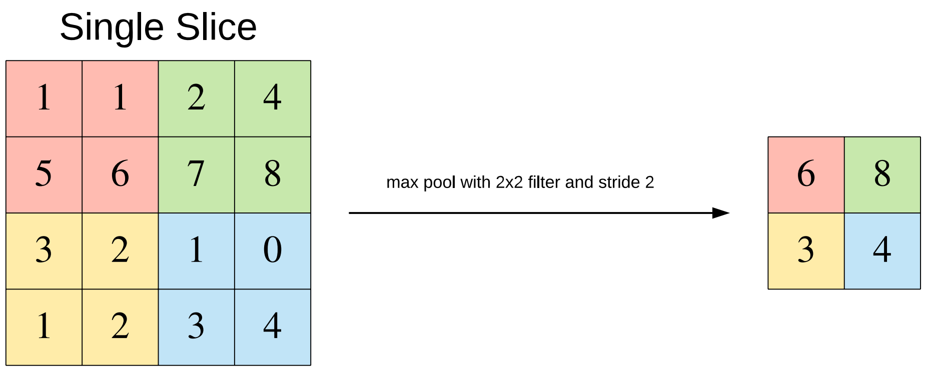


Figure 2.7: *An example of max pooling*

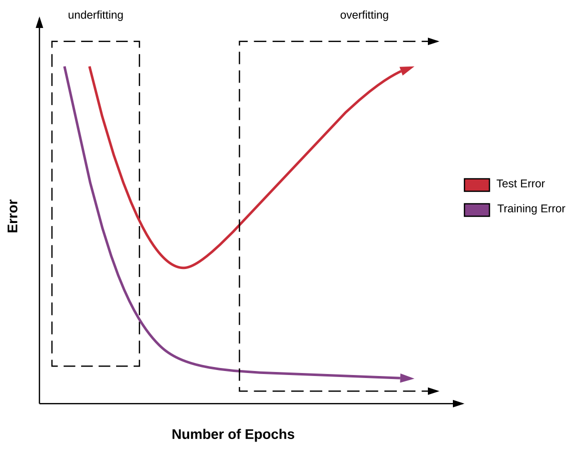


Figure 2.8: *An illustration of the symptoms of over- and under-fitting*

set or the problem domain as a whole. An example of this is if training a fruit classifier with a training set consisting of an apple, an apple, and a banana. When it encounters a strawberry the model will probably think the strawberry is an apple because it is red and all red things it has seen so far are apples. In this case the neural network has identified a particular pattern that is all it cares about. This pattern allows it to correctly predict everything in the test set so it believes that it is accurate all the time. However, we know that this pattern does not hold true when applied to the whole set, or at least when applied to a test set that includes a strawberry. So in general, overfitting increases test accuracy while severely decreasing test accuracy and therefore should be avoided.

There are several techniques to address this problem. The simplest of them is just training on a more diverse dataset. Another technique is to stop training the model when you notice the test accuracy no longer decreases. The model reaches a certain point where most of the patterns left to consider that increase the accuracy on the training set are not generalizable to the set as a whole. Unfortunately this is hard to locate or predict as the first thing it decides to look at could be specific to the training data and all the model knows is it is accurate on the training data which is all it cares about. The third technique for preventing overfitting is called regularization. The idea behind it is to tell the model that the higher its internal weights are, the worse the accuracy is. If the model is blindly focusing on a single input bit, the weights along the path from the bit to the outcome will be extremely high. Regularization forces the model to try to spread the

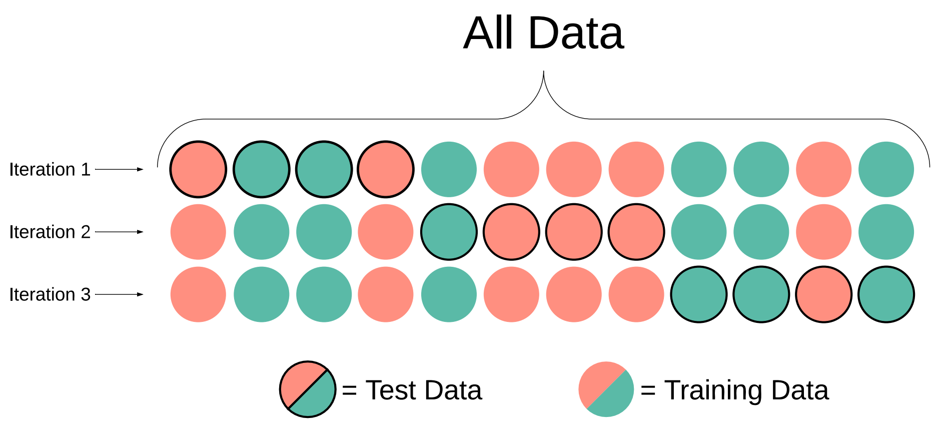


Figure 2.9: *An example of cross validation*

weight across more of the input bits meaning the model takes more features into consideration, which tends to make it less likely to fixate on a particular piece that will lead to overfitting. Regularization influences the model by calculating a loss value based on the weight values, which it factors into the regular loss while training.There are two main types of regularization loss that are commonly used: L1 loss and L2 loss. L1 loss calculates the loss from the weights in a linear way while L2 squares it. Because of this L2 attributes more loss to large weights helping to quickly reduce these weights. This can help to reduce overfitting by preventing the model from fixating on any patterns.

When there is not a sufficient amount of data for the model to test or train on, it is difficult to ensure that the testing and training sets do not have any bias. For example, if the set used to train has half bananas and half apples and then you test with only pears the results will probably be different than if both the training and test sets each consist of an even distribution of apples, pears, and bananas. One technique that attempts to address this is cross validation[10, 11]. The idea behind cross validation is that you can run the model multiple times on multiple sections, called folds, of the data provided to reduce chances for picking a split that has bias in any direction. For example, pretend there is a dataset of 4000 images. If we want the test data to be 25% of it we can split it into 4 folds as seen on the right in figure 2.9. How the data is split or grouped could be changed but the general idea behind it is the same; every image is in the test set for one run and the training set for the others. Doing multiple runs with different sets provides proof that the model works without a biased testing set.

There are many vastly different problems and tasks that neural networks aim to solve, all with varying types and complexities of data. However, simple data does not inherently mean a simple model. For example, one application of a convolutional neural network to one dimensional data is text understanding. Human languages, particularly idiomatic ones like English, are full of meaning that cannot be understood by explicitly reading the text, making it an extremely difficult thing for a network to interpret. The network cannot simply look at each character one by one, it must look at all characters together to build a context from which to understand what is written.

On the other side of the spectrum is using a convolutional network to analyze four dimensional data. The most common examples are three dimensional images in a time series, in other words a three dimensional video such as an MRI. However, as T. Wang[12] discusses, four dimensional data by nature is extremely difficult to model, as CNN architectures have not yet been developed for four-dimensional data. With added complexity comes more noise and more computation requiring more preprocessing and training to successfully train a model. T. Wang[12] discusses several methods of circumventing these issues, one of which we used in our own project. In order to reduce their four dimensional data he and his team averaged their data across a dimension, allowing pre-existing CNN architectures to be used.

2.3 fMRI Applications

Brain injuries and diseases, such as concussions or dementia, are conditions that can be both debilitating and difficult to diagnose. These specific examples are also widespread and affect people of all ages, which in turn affects those around them[13][14]. For example. dementia affects an estimated 24 million people worldwide and yet it is often undiagnosed in private practice settings, especially when it is at an early stage[15][16]. For another example, this article [17] reported that 300,000 concussions occur per year, but estimates that the real number of occurrences is 3.4 million. Early detection and intervention for these diseases is important, but often difficult. Cases of concussions and other brain injuries, unlike brain diseases, are easier to detect immediately but become more difficult to detect the later the diagnostic is done. One review of three widely used computerized tests for detecting sport-related concussions found that their sensitivity of detection peaked when applied within 24 hours of the injury, with a maximum rate of 67.8%. However, if administered after more than eight days, their sensitivity had diminished nearly to their false positive rates. Despite this, these computerized neurocognitive tests are currently the most reliable method of detecting a sport-related concussion[18].

Systems and methods that allow us to analyze the brain directly could give us a better ability to detect these conditions. Functional magnetic resonance imaging (fMRI) is a technique that allows us to record and monitor the activity, or function, that is happening in a brain. Magnetic resonance imaging (MRI) works by detecting the changes in the magnetic resonance (MR) of the body, which can be applied to the brain to map its structure. In order to detect the activity of the brain, fMRI machines monitor fluctuations in the MR signal in the brain, which correlate with fluctuations in its activity. The correlation between these factors is not direct; in fact MR signal actually correlates directly to the level the of blood-oxygenation which in turn correlates to brain activity[1]. This technique of detecting brain activity can be used to investigate and diagnose conditions which affect the functionality of the brain, instead of just the structure. This can be particularly useful for conditions such as concussions, which are believed to affect only the function of the brain and not its structure[19]. There is also evidence that concussions, and other traumatic brain injuries, create a detectable change the in the functionality of the brain. This change in functionality can be detected even if there is no longer any noticeable change in task performance of the individual, which is the standard method of diagnosis[20].

The information acquired from MRI and fMRI can be analyzed to determine the health and functionality of a brain, but this analysis can be difficult. The data produced by an fMRI can be very difficult to interpret due to its high resolution and noise. Interpreting the data from an fMRI and making a reliable diagnosis from it requires years of experience. Finding automated ways to perform this diagnosis would be valuable for situations where an expert is unavailable or too costly. Until recent years, the issue of resolution and noise made it difficult to develop a computerized solution. However, the improvement and growth of machine learning techniques has allowed us to deal with these issues. One study used support vector machines, an alternate method of supervised machine learning, to classify sporadic Alzheimer's disease using data from MRI. They found that it could classify sporadic Alzheimer's disease in 93% of cases. This compared to radiologist's with different levels of experience, who had rates ranging 80% to 90%[20]. While this does not prove that an expert will always be out-done by a computerized system, it does show that there is an opportunity for computerized analysis of data from MRI and fMRI.

2.3.1 Data Pre-processing

When working with high complexity data, such as that from fMRI and MRI, a common tool to improve results is preprocessing the data before giving it to the model. This preprocessing is intended to reduce the noise, variability, and complexity in the data. One research team[21] preprocessed MRI data obtained from the Alzheimer's Disease Neuroimaging Initiative (ADNI) database. They reduced noise by removing non-relevant parts of the body that were in the scan, such as the skull and cerebellum, normalizing the intensity, and smoothing the data with a Gaussian kernel. They reduced complexity by using positron emission tomography (PET) images to align the MRI images (when available), segmenting the MRI data and only using the gray matter map, and downsampling the final images. To reduce the variability, they also normalized the spatial component of the grey matter map into what they called a template space. Another team[22] used a four step process to preprocess fMRI scans. They began by correcting the phase shifts of the signal in a given scan, followed by correcting for the motion of the subject. Their next step were to normalize the spatial dimensions to reduce differences between subjects, and the last was to smooth the spatial dimension, again using a Gaussian kernel. The first and third steps were to reduce variability between subjects and scans, while second and fourth steps were to reduce the noise in the data. A third research team[23] used resting state fMRI data which had been already preprocessed by the "connecttome" project, which included segmenting the brain into 90 regions, which reduced the complexity of the data by adding meta-data. However, these methods, while improving the quality of the data that the neural network is given, also require an overhead cost to implement.

METHODOLOGY

3.1 The Opportunity

MRI and fMRI techniques, in combination with the advancement of machine learning, introduce a promising opportunity to create automated system that can analyze the data they produce. As discussed in the previous section, exploration into this area has already begun. The goal of this exploration is to build systems that can help analyze and diagnose conditions in a more timely, accurate, and cost efficient way. In this project, we aspired to contribute to this exploration. Our initial plan was to build a neural network that could directly analyze fMRI scans of a brain and detect an injury or disease, without any pre-processing of the data. The specific type of network and condition was initially undefined, so the neural network model we built evolved over time.

The data set we ultimately used consisted of scans from 29 athletes in a resting state, with 14 of them confirmed to have a concussions, and 15 without. Their ages ranged from 18 to 22, and they were of both sexes. The sports that the athletes participated in also varied. The time between the concussion occurring and the fMRI scan ranged from 6 days to 180 days. The scans were provided in the NIfTI-1 data format, which was created by the Neuroimaging Informatics Technology Initiative (NIfTI). More information about the scans can be found in appendix B. An example of a scan can be seen in 3.1.

For this project, we used Python as a base language due to the existence of several high quality, domain specific libraries available for it such as TensorFlow, PyTorch, and Nilearn. Both PyTorch and TensorFlow are powerful libraries that provide tools for building custom neural networks while abstracting much of their complexity. We decided to use PyTorch over TensorFlow because it allowed us to build a more flexible model, which enabled us to easily experiment with different ideas of how the model should be built while we were building it. Another library that we used significantly was Tensorboard, which is a visualizer tool for neural networks. This allowed for graphing of any set of metrics in real time while the network wastraining, which enabled us to monitor the model. This gave us the ability to catch errors, or a mis-behaving model, much earlier. It also allowed us to have graphical representations of how the training went. This made

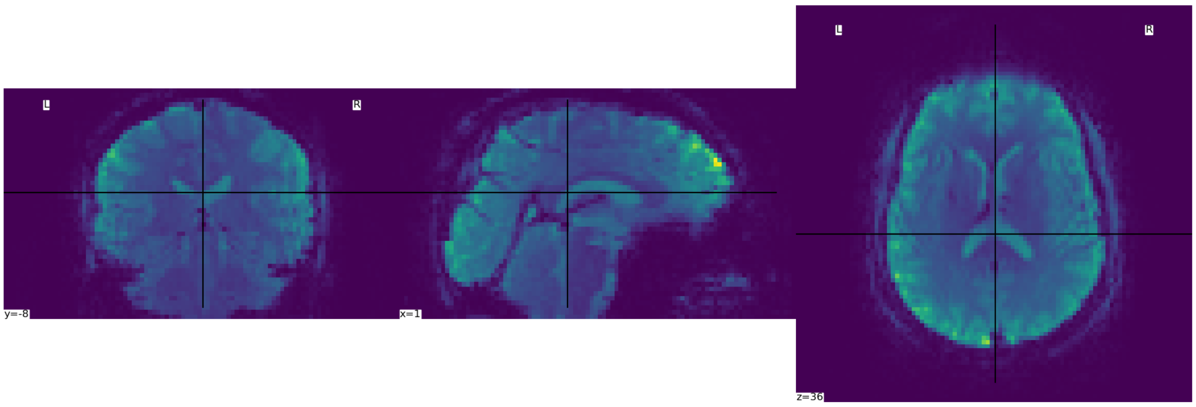


Figure 3.1: *Example of concussion fMRI scan*

reviewing and evaluating different versions of the model both quicker and more in depth, because it allowed us to compare more than just the final result. While it was originally designed to work with TensorFlow, we found the library TensorboardX that allowed us to integrate Tensorboard with PyTorch.

The hardware that we used for training our model was the WPI ACE computer cluster. These machines are provided for students to use for long term computations and simulations, which was a valuable resource for training a neural network model. Each node in the cluster has 128 GB of memory, and a minimum of 32 CPUs. The system has 24 NVIDIA K20 GPUs total. It uses the Slurm Workload Manager to manage jobs. The batch manager limits the time that a computation can run for to 12 hours, and limits the number of CPUs a job can use to 40 and the number of GPUs to 2.

3.2 Techniques

The initial network that we designed was a simple fully connected model with two layers. The model processed the four dimensional fMRI image by starting with a large number of inputs, one for each voxel, and reducing it to 1000 outputs which was then further reduced to 2. Each final output represented either a positive or negative concussion diagnosis. The loss function we used was NLLLoss, which is similar to cross entropy loss except it doesn't perform softmax on the output. We included a ReLU activation function in between the fully connected layers, and a softmax function on the final outputs. While this was a good place to start it gave us poor results. These results were due to several problems. The two layer rapid reduction of the data was unlikely to get good results because it did not give the model enough room to extract features. Additionally, the data was so large that the number of outputs of the first layer could not be much more than 1000 without the model being too big to load into memory. This meant that we could not add more layers or increase the layer sizing to fix the problem.

In order to resolve this we tried reducing the data in a meaningful way by averaging out the fourth dimension, time. We reasoned that the image's change over time would be the least significant dimension for a diagnosis. We had to make a sacrifice in the quantity of data to reduce the model size, and averaging gave each time slice an influence on the resulting data. This reduced the number of inputs from 71,884,800 to a much smaller size of 460,800. This allowed us to increase the complexity of the model. However, the model still used fully connected layers which treat the problem linearly and ignore the locality of the data.

To address this we decided to add two convolutional neural layers to the beginning of the model to help it break down the data while considering locality. The first convolutional layer takes in one input channel with all the data, applies a 2x2x2 kernel to it and outputs 11 channels. ReLU was then applied to each output, and maxpool was used with a 2x2x2 kernel size to reduce the data further. The next convolutional layer received the 11 output channels from the previous layer, applied a 2x2x2 kernel to each, and output 30 channels. After applying the same ReLU and max pooling to the outputs, they were then passed into the linear layers which now had less data to deal with. The diagram for this model can be seen in Figure 3.2.

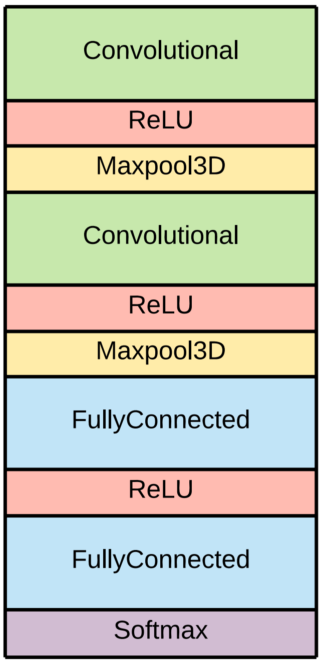


Figure 3.2: *A diagram of our model*

One issue the model still had was it was having trouble learning from such a small dataset. One paper found that even 350 samples were not sufficient for a high difficulty problem when using whole brain MRI scans[24]. Small datasets can be biased both in the data they contain and in how they are split into test and training sets. Splitting the dataset into test and training sets in such a way that at least one of them is uneven leads to bias. For example, if there are only 9 images in a binary classification problem's training set then it is guaranteed to see one label more than the other.

Our method of addressing this problem for our small dataset was to implement cross validation. This allowed us to test various configurations of the testing and training data split, which helped reduce the chance of bias in the splitting of the dataset. Different configurations, or folds, produced varying results which we could then average to get a relatively unbiased result. While this reduced bias, it did not significantly improve the models performance.

At this point, the main issue was that the model could not perform better than random, even after 60 epochs of training. Because of our small dataset, we expected our model to overfit quickly. Instead we found the accuracy of the model to be 51% at best. We noticed that the loss

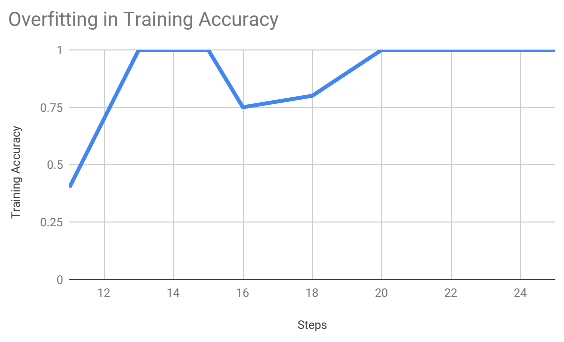


Figure 3.3: *A graph of the training accuracy of the model*

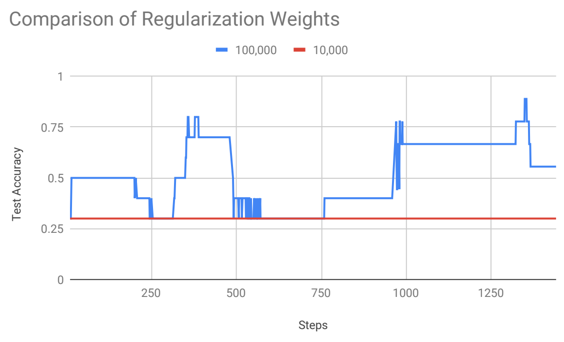


Figure 3.4: *A graph of the test accuracy of the model*

during training was extremely chaotic, leading us to believe the learning rate was too high. To minimize the length it took to overfit while still having a small enough learning rate, we implemented learning rate decay. In order to improve our understanding of how the model was training, we also integrated tensorboard at this time. This allowed us to better understand how the hyperparameters we were testing affected training performance. Eventually we found a set of parameters that enabled the model to overfit within 20 epochs (Figure 3.3). In order to compare training runs to each other, we set the random seed of the program. This meant that if one ran two runs with exactly the same parameters at two different times, the model training should match exactly. This allowed us to determine which set of hyperparameters was best for any set of runs. The next step was to prevent overfitting in order to increase our test accuracy, which was still chaotic at this point.

The method we used to reduce overfitting was L2 weight regularization. We experimented with different magnitudes of this hyperparameter, from 10^{-8} up to 100,000 and found that larger values performed much better. Figure 3.4 shows the test accuracy of two identical runs over 1,400 steps, except that orange has a regularization weight of 1000 and green has a regularization weight of 100,000. As you can see, while neither reached optimal accuracy, the higher regularization weight significantly improved the test accuracy. However, we needed some way to increase test accuracy further.

In the end, we elected to manipulate the input data in a unique way. Instead of averaging each input over the time dimension to get a 3D input, we generated a series of independent slices of each fMRI across the x-axis. This converted each 4D input into a series of 3D inputs, with each 3D input representing a 2D slice over time. We hoped that by more closely associating a slice with its change in state over time the model would be able to better see patterns in brain activity. At the same time, since we now had fifty times as many inputs, we removed cross validation and instead randomly distributed 30% of slices from each scan to the test set and the remaining to the training set by random sampling.

However, the downside of having much more input data is that it took significantly longer to train. We found that the limit of 12 hours set by ACE was not nearly enough to complete a full training run, meaning we needed to find a way to restart our model from a saved state. To do this, we used PyTorch's built-in model exporting functionality. Every five minutes we saved the state of the model, as well as the current epoch and step. When a run was terminated, we could restart the run and it would automatically load the state of the model from the previously saved checkpoint. This enabled us to train our model until the accuracy was no longer meaningfully increasing, allowing it to reach higher test accuracy.

Results

After considerable experimentation with the model (Figure 4.1), we found that it performed best with a learning rate of 10^{-10} and a regularization weight of 1000. After roughly 70 hours of training, which consisted of 400 epochs, the model reached a test accuracy of 70% and a training accuracy of 74%. While not more accurate than a human doctor, it represents a significant step in computer-aided healthcare. In order to reach this level of accuracy, we compared the performance of our model using several different hyperparameters.

First, we selected a regularization weight. From previous experiments we knew it would need to be large, so we compared four values to the model without weight regularization: 1,000; 10,000; 50,000; and 100,000. The learning rate for all five runs was 10^{-9}, as we had found that to work well in the past. The best performing hyperparameters were 100,000 and 1,000 with 68% test accuracy and 73% training accuracy, so we selected 1,000. (Appendix A page 1). Second, we selected a learning rate. We knew that 10^{-9} worked well, and so centered our hyperparameter testing around it. We tested learning rates of 10^{-7}, 10^{-8}, 10^{-9}, 10^{-10}, and 10^{-11}. Of these five, 10^{-10} by far performed the best, with a test accuracy of 70% and a training accuracy of 74%. This can be seen in Appendix A page 2.

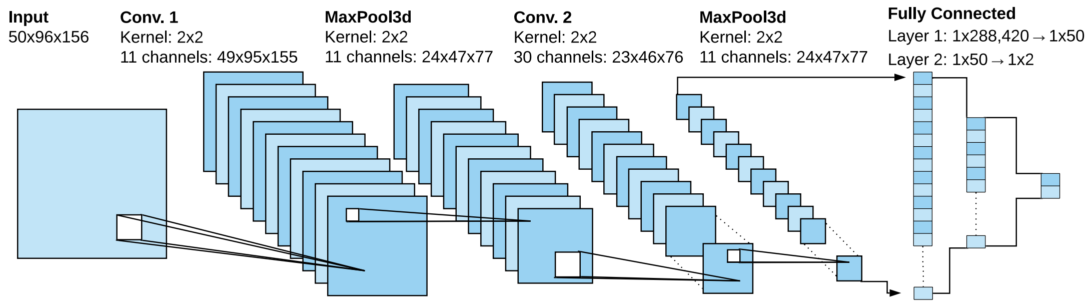


Figure 4.1: *A visual representation of the final model structure*

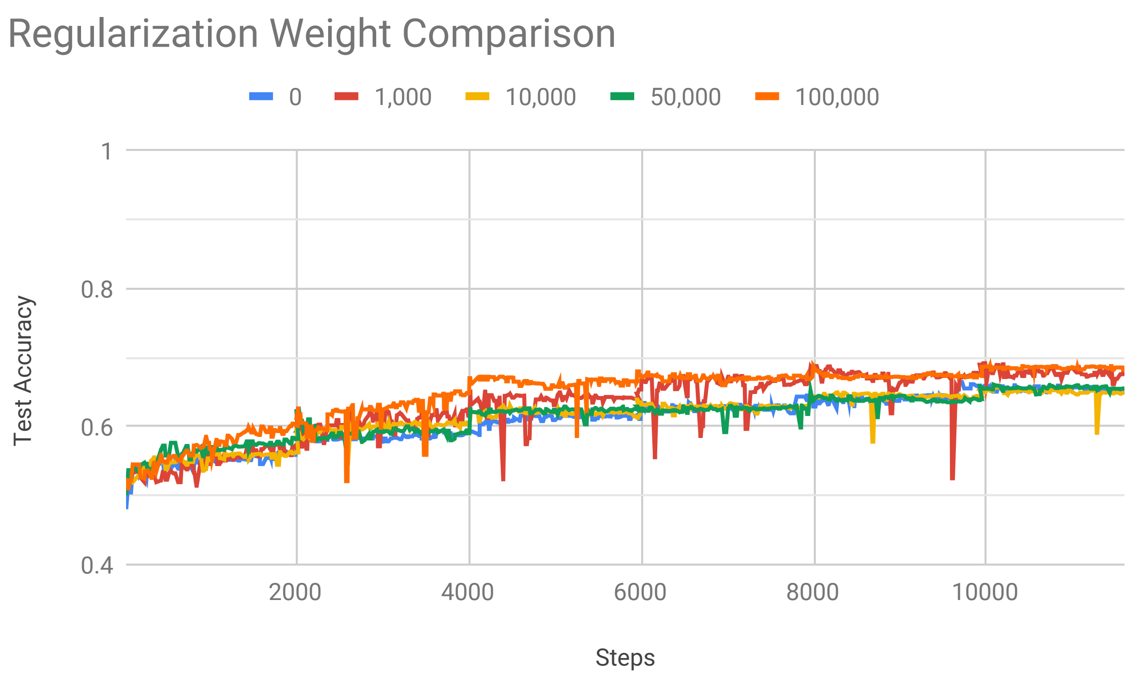
Having found what we believed to be the optimal hyperparameters for our model on this data, we compared it to two versions of our previous design which averaged across the data, one using a fully connected network and the other using a convolutional neural network. As can be seen in Appendix A. Each model was run for 400 epochs, allowing each to reach its maximum potential test accuracy. As can be seen in Appendix A page 3, our final model, the Slicing CNN, far outperformed the other two, which never made it above random.

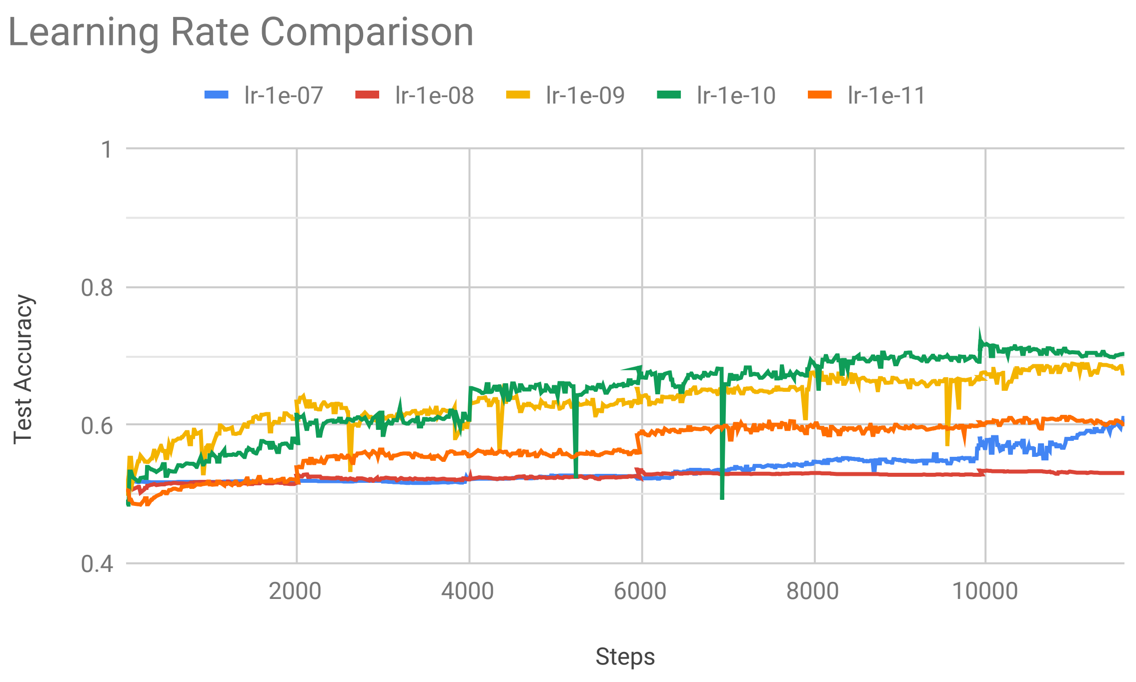
Conclusions and Future Work

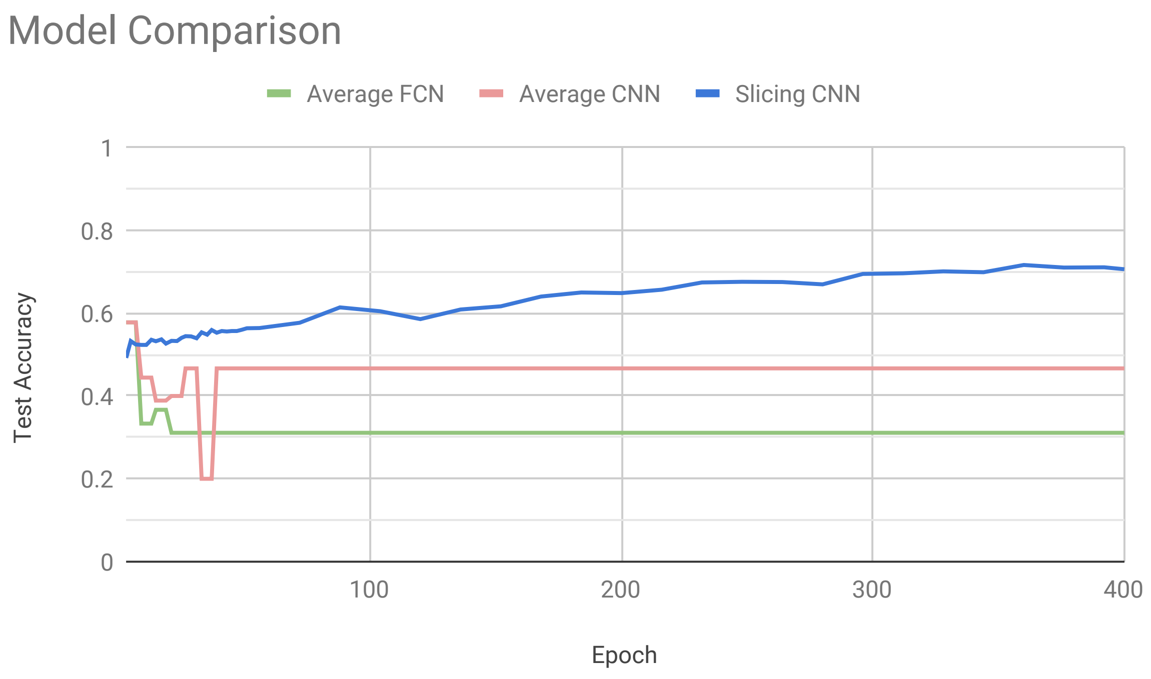
The results of this model are promising. A neural network that can detect and diagnose conditions similar to a doctor could help in the medical industry by diagnosing patients faster and for a lower cost. This in turn could make good healthcare more affordable for lower income families and increase their quality of life.

This being said, our model still has room for expansion. The model is currently limited by the number of fMRIs it has to analyze. 30 pieces of data is extremely small for any neural network, not to mention one that works with such complex data. If the neural network could train on more data it would certainly result in a significant improvement in its overall accuracy, potentially exceeding the current 70% diagnosis accuracy. Our data was also taken at varying times after the concussion was potentially received. If all the fMRI scans were done at similar times after the concussion, this would improve the consistency of the data. This allows the model to see patterns that occur in the scan around one particular time and not be confused by patterns that appear and disappear at various times. Another limitation in the model is the fact that it has no preprocessing. The model currently looks at only the raw image voxels but preprocessing could be a valuable tool for expanding the models power, especially if more data is acquired. Preprocessing also requires an in-depth knowledge of the domain, in this case brain function and other related fields, which unfortunately our group did not possess. With preprocessing, more complex model structures can be applied. This is due to the fact that preprocessing removes noise and helps the model focus in on particular aspects of the data, reducing its complexity. Our model was also limited by the computational power of the computers it trained on. Training it for 400 epochs took roughly 70 hours on the machines we used and so the model could make use of more computational power to train longer and faster, especially if the data was expanded. Despite being beyond the scope of this project, these could improve the models performance and make it a more feasible solution. Ultimately, this project's results show that some application of neural networks to data in this area is well within reach with modern techniques.

Appendix A: Result Graphs






Model Comparison
Average FCN
Average CNN
Slicing CNN
Test Accuracy
1
0.8
0.6
0.4
0.2
0
100
200
300
400
Epoch

APPENDIX B: CONCUSSION SCAN INFORMATION

Subject ID	Sport	Concussed?	Post-concussion 31 P	Post-concussion GABA	Age at Enrollment	Gender	Handedness	Height (cm)	Weight(kg)
TBI001	Lacrosse	No	N.A.	N.A.	21	1	0	181	88
TBI002	Lacrosse	No	N.A.	N.A.	22	1	0	173	73
TBI003	Swimming	No	N.A.	N.A.	20	1	0	177	76
TBI004	Lacrosse	Yes	27	44	22	1	0	173	76
TBI005	Track	No	N.A.	N.A.	19	0	0	156	65
TBI006	Lacrosse	Yes	180	180	19	1	0	172	82
TBI007	Lacrosse	No	N.A.	N.A.	21	0	0	161	59
TBI008	Football	No	N.A.	N.A.	22	1	0	190	129
TBI009	Softball	Yes	76	76	20	0	0	162	68.5
TBI010	Field Hockey	No	N.A.	N.A.	20	0	0	169	68
TBI011	Football	Yes	77	77	20	1	1	180	101
TBI012	Volleyball	No	N.A.	N.A.	19	0	0	170	104
TBI013	Lacrosse	Yes	89	89	22	1	1	177	71
TBI014	Football	No	N.A.	N.A.	18	1	0	186	110
TBI015	Basketball	Yes	23	23	21	0	0	182	82
TBI016	Ice Hockey	Yes	166	166	20	0	N.A.	158	59
TBI017	Gymnastics	No	N.A.	N.A.	18	1	0	163	62
TBI018	Ice Hockey	Yes	185	185	19	1	0	170	68
TBI019	Ice Hockey	Yes	184	184	20	1	0	172	86
TBI020	Field Hockey	No	N.A.	N.A.	19	0	0	159	53
TBI021	Ice Hockey	No	N.A.	N.A.	21	0	0	170	65
TBI022	Basketball	No	N.A.	N.A.	19	0	0	160	52
TBI023	Volleyball	Yes	21	21	19	1	0	168	86
TBI024	Soccer	No	N.A.	N.A.	22	1	0	178	75
TBI025	Football	Yes	6	6	20	1	0	173	79
TBI026	Field Hockey	No	N.A.	N.A.	19	0	0	160	60
TBI027	Ice Hockey	Yes	13	13	19	1	0	183	77
TBI028	Basketball	Yes	9	9	19	1	0	178	73
TBI029	Skiing	Yes	14	14	21	0	0	170	76

Bibliography

[1] S. A. Huettel, A. W. Song, G. McCarthy *et al.*, *Functional magnetic resonance imaging*. Sinauer Associates Sunderland, MA, 2004.

[2] T. Madhyastha, M. Peverill, N. Koh, C. McCabe, J. Flournoy, K. Mills, K. King, J. Pfeifer, and K. A. McLaughlin, “Current methods and limitations for longitudinal fmri analysis across development,” *Developmental Cognitive Neuroscience*, vol. 33, pp. 118 – 128, 2018.

[3] K. A. Norman, S. M. Polyn, G. J. Detre, and J. V. Haxby, “Beyond mind-reading: multi-voxel pattern analysis of fmri data,” *Trends in Cognitive Sciences*, vol. 10, no. 9, pp. 424 – 430, 2006.

[4] M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects,” *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[5] A. C. Lagandula. (2018) Mcculloch-pitts neuron - mankind’s first mathematical model of a biological neuron. Accessed: 2019-02-7.

[6] A. Kurenkov. (2015) A ’brief’ history of neural nets and deep learning. Accessed: 2019-02-7.

[7] M. Minsky and S. A. Papert, *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 9, pp. 533–536, 1986.

[9] W. Zaremba, I. Sutskever, and O. Vinyals, “Recurrent neural network regularization,” *Computer Research Repository*, vol. abs/1409.2329, 2014.

[10] S. Lemm, B. Blankertz, T. Dickhaus, and K.-R. Müller, “Introduction to machine learning for brain imaging,” *NeuroImage*, vol. 56, no. 2, pp. 387 – 399, 2011, multivariate Decoding and Brain Reading.

[11] R. Kohavi *et al.*, “A study of cross-validation and bootstrap for accuracy estimation and model selection,” in *International Joint Conference on Artificial Intelligence*, vol. 14, no. 2, 1995, pp. 1137–1145.

[12] T.-C. Wang, J.-Y. Zhu, E. Hiroaki, M. Chandraker, A. A. Efros, and R. Ramamoorthi, "A 4d light-field dataset and cnn architectures for material recognition," in *European Conference on Computer Vision*, 2016, pp. 121–138.

[13] M. Wortmann, "Dementia: a global health priority - highlights from an adi and world health organization report," *Alzheimer's Research & Therapy*, vol. 4, no. 5, p. 40, 2012.

[14] R. S. Moser, "The growing public health concern of sports concussion: The new psychology practice frontier," *Professional Psychology: Research and Practice*, vol. 38, no. 6, pp. 699–704, 2007.

[15] C. P. Ferri, M. Prince, C. Brayne, H. Brodaty, L. Fratiglioni, M. Ganguli, K. Hall, K. Hasegawa, H. Hendrie, Y. Huang, A. Jorm, C. Mathers, P. R. Menezes, E. Rimmer, and M. Scazufca, "Global prevalence of dementia: a delphi consensus study," *The Lancet*, vol. 366, no. 17, pp. 2112–2117, 2005.

[16] V. G. Valcour, K. H. Masaki, J. D. Curb, and P. L. Blanchette, "The detection of dementia in the primary care setting," *Archives of Internal Medicine*, vol. 160, no. 19, pp. 2964–2968, 2000.

[17] M. E. Halstead, K. D. Walter, and , "Sport-related concussion in children and adolescents," *Pediatrics*, vol. 126, no. 3, pp. 597–615, 2010.

[18] L. D. Nelson, A. A. LaRoche, A. Y. Pfaller, E. B. Lerner, T. A. Hammeke, C. Randolph, W. B. Barr, K. Guskiewicz, and M. A. McCrea, "prospective, head-to-head study of three computerized neurocognitive assessment tools (cnts): reliability and validity for the assessment of sport-related concussion," *Journal of the International Neuropsychological Society*, vol. 22, no. 1, p. 24–37, 2016.

[19] A. Ptito, J.-K. Chen, and K. M. Johnston, "Contributions of functional magnetic resonance imaging (fmri) to sport concussion evaluation," *NeuroRehabilitation*, vol. 22, no. 3, pp. 217–227, 2007.

[20] T. McAllister, A. Saykin, L. Flashman, M. Sparling, S. Johnson, S. Guerin, A. Mamourian, J. Weaver, and N. Yanofsky, "Brain activation during working memory 1 month after mild traumatic brain injury," *Neurology*, vol. 53, no. 6, pp. 1300–1300, 1999.

[21] R. Li, W. Zhang, H.-I. Suk, L. Wang, J. Li, D. Shen, and S. Ji, "Deep learning based imaging data completion for improved brain disease diagnosis," in *Medical Image Computing and Computer-Assisted Intervention*, P. Golland, N. Hata, C. Barillot, J. Hornegger, and R. Howe, Eds., 2014, pp. 305–312.

[22] E. E. Tripoliti, D. I. Fotiadis, M. Argyropoulou, and G. Manis, “A six stage approach for the diagnosis of the alzheimer’s disease based on fmri data,” *Journal of Biomedical Informatics*, vol. 43, no. 2, pp. 307 – 320, 2010.

[23] A. Riaz, M. Asad, S. M. M. R. A. Arif, E. Alonso, D. Dima, P. Corr, and G. Slabaugh, “Deep fmri: An end-to-end deep network for classification of fmri data,” in *International Symposium on Biomedical Imaging*, 2018, pp. 1419–1422.

[24] C. Chu, A.-L. Hsu, K.-H. Chou, P. Bandettini, and C. Lin, “Does feature selection improve classification accuracy? impact of sample size and feature selection on classification using anatomical magnetic resonance images,” *NeuroImage*, vol. 60, no. 1, pp. 59 – 70, 2012.