6-25-2019

Inverted cone convolutional neural network for deboning MRIs

Oliver John Palumbo
Rowan University

INVERTED CONE NEURAL NETWORK FOR DEBONING MRIS

by

Oliver Palumbo

A Thesis

Submitted to the
Department of Electrical & Computer Engineering
College of Engineering
In partial fulfillment of the requirement
For the degree of
Master of Science in Electrical and Computer Engineering
At
Rowan University
November 4, 2018

Thesis Chair: Nidhal Bouaynaya, Ph.D.

Dedications

To my parents and family.

Acknowledgment

This work was supported by the National Science Foundation under Award Numbers NSF DUE-1610911 and NSF ACI-1429467.

Abstract

Oliver Palumbo
INVERTED CONE NEURAL NETWORK FOR DEBONING MRIS
2018-2019
Nidhal Bouaynaya, Ph.D.
Master of Science in Electrical and Computer Engineering

Data plenitude is the bottleneck for data-driven approaches, including neural networks. In particular, Convolutional neural networks (CNNs) require an abundant database of training images to achieve a desired high accuracy. Current techniques employed for boosting small datasets are data augmentation and synthetic data generation, which suffer from computational complexity and imprecision compared to original datasets. In this paper, we intercalate prior knowledge based on spatial relation between images in the third dimension by computing the gradient of subsequent images in the dataset to remove extraneous information and highlight subtle variations between images. The approach is coined ``Inverted Cone" because the volume of brain images below the level of the eyes is ordered to form an inverted cone geometry.

The application explored in this work is deboning, or brain extraction, in brain magnetic resonance imaging (MRI) scans. The difficulty of obtaining ground truth for this application prevents the ability of obtaining a large quantity of training images to train the CNN. We considered a limited dataset of 23 patients with and without malignant glioblastoma. Deboning was performed by employing an optimized CNN architecture with and without the Inverted Cone processing. The classic CNN without prior knowledge achieved a validation accuracy of 77 %, while the Inverted Cone CNN model achieved a validation accuracy of 86 % in a dataset of 451 brain MRI slices.

Table of Contents

Table of Contents (Continued)

List of Figures

List of Figures (Continued)

Figure Page

List of Tables

Chapter 1

Introduction

In this chapter, we will outline the motivation and problem in deboning brain MRI scans as well as the current approaches taken in medical image segmentation. Additionally, we will elaborate on the contributions of this thesis work.

1.1 Motivation, Problem Statement and Background

Deep learning has been widely utilized in object detection and recognition. Convolutional neural networks (CNNs) allow processing and analyzing large sets of image data into classification of predefined classes. Beginning with the AlexNet architecture developed in 2012 by the SuperVision group [1], CNNs have been proven to outperform classical modeling for object detection. AlexNet ranked with the top-5 error rate of models for the *ImageNet Large Scale Visual Recognition Challenge* with only 15.3 % error by classifying 1.2 million images into 1,000 categories [1].

Since then, deep learning for object recognition has been expanded into many different applications, such as pothole detection for intelligent transportation systems [2] and medical brain tumor segmentation as an aide for medical diagnoses [3], [4]. Image segmentation is a concentrated application of object detection that distills an image into a series of patches such that every pixel in an image can be classified.

The Multimodal Brain Tumor Segmentation (BRATS) competition [5] strives to improve the brain tumor segmentation application by evaluating a set of image segmentation models, both classical and deep learning, to determine which technique produces the highest accuracy. The task is to develop a model that detects and classifies

5 distinct regions in a brain magnetic resonance imaging (MRI) scan (normal tissue, necrosis, edema, non-enhancing, and enhancing tumor). A CNN model was awarded first place in the 2015 BRATS challenge with Dice Similarity Coefficients of 0.88, 0.83, and 0.77 in the complete, core, and enhanced regions, respectively, as computed by the BRATS organization [4].

A vital issue that arises in specific applications such as brain MRI segmentation or pothole detection is the requirement of a large database of images to train the network. The BRATS competition employed a training dataset comprising 276 patients' four modalities MRIs with each MRI modality containing approximately 150 images. Often times, especially for medical image applications, image datasets are limited with a small number of images available for training the network, which can result in over fitting of the model to the images in the training database and not being able to generalize well on unseen images. The Inverted Cone CNN introduced in this work serves to increase the classification accuracy of a CNN on brain MRI scans by leveraging prior knowledge of the ordered nature of MRI scans to reduce the complexity of the dataset for training purposes.

1.2 Research Contributions

The contributions of this research involve deriving the Inverted Cone framework for deboning magnetic resonance images (MRIs). This approach exploits the temporal structure of the MRIs to improve classification results by machine learning algorithms. A fundamental limitation of neural networks and other machine learning techniques is the quantity of thoroughly-labeled data which is required to train an accurate system.

The Inverted Cone method relies on the known ordered nature of a brain MRI scan sequence to remove difficult to classify regions in a size-restricted dataset. By calculating the gradient between the area of subsequent scans, large and complex sinus structures can be removed in preprocessing that significantly improve the accuracy of a standard CNN model.

The Inverted Cone framework was implemented on brain MRIs, with and without tumors, provided by the University of Alabama at Birmingham School of Medicine. The brain MRI scans of 23 patients were analyzed with each patient scan consisting of 17-38 slices.

1.3 Organization

This thesis work will be organized in the following way.

In Chapter 2, we perform a literature review for the current methods of brain MRI deboning. The techniques used in the field include by hand analysis of the MRI scans as well as software that relies upon a manually adjusted parameter and visual inspection. The automation that is granted by the work in this thesis will remove the need for manually adjusted parameters and lessen the work of visually inspecting the scans. Furthermore, we discuss the foundations for neural networks and CNNs, and how each of these can be designed to solve classification problems.

In Chapter 3, an overall perspective is taken of the brain MRI processing procedure. Each step of the brain diagnosis process is broken down and specific analysis taken on the deboning stage. Additionally, the brain MRI deboning problem is framed in this section as a classification problem on which a neural network could be employed.

In Chapter 4, the Inverted Cone CNN framework is introduced. We will explore the motivation behind utilizing the Inverted Cone method to preprocess brain MRIs prior to deboning via CNN. The database of MRI scans provided by the University of Alabama at Birmingham will be detailed in this section and the process of preparing each scan with the Inverted Cone method prior to deboning with neural network architecture.

In Chapter 5, we will describe the implementation of the Inverted Cone CNN technique for brain MRI deboning on the high-performance computer (HPC) at Rowan University. The specifications of the HPC system will be described along with the languages and libraries utilized to design the Inverted Cone CNN.

In Chapter 6, the simulation results of deboning brain MRIs with the Inverted Cone CNN will be discussed. We will show that the CNN employing the Inverted Cone preprocessing technique outperforms a standard CNN in validation accuracy measures. Example segmentation results will also be provided.

In Chapter 7, we will summarize this work by explaining the contributions and successes uncovered in the development of our thesis. The research procedure and findings will be outlined with conclusions drawn. Furthermore, we will speculate upon future developments to the thesis work.

Chapter 2

Literature Review

In this chapter, we will review the current techniques being employed in the medical field in order to debone brain MRI scans. Additionally, we will review the fundamentals contributions of neural networks with particular interest in CNNs.

2.1 Neural Networks

Neural networks are powerful machine learning tools that create systems with unparalleled capability to extract features from datasets and classify inputs based on these features. The core principle of neural networks is training a series of artificial neurons with a large quantity of training data for the network to understand the features of this dataset and can make predictions on the classifications of new input data.

Neurons are arranged in a parallel fashion in the form of layers. An input and output layer exists at the input and output of the network respectively. The input layer consists of a weighted connection between each data point of the input with each neuron in the first hidden layer. The output layer consists of several neurons corresponding with the desired number of classes to be predicted or to combine the data in the desired fashion. Between the input and output layers reside hidden layers containing several parallel neurons. Weighted connections relate each neuron in one layer to every neuron in the next layer. A simple diagram of a two-hidden layer neural network is pictured in figure 1.

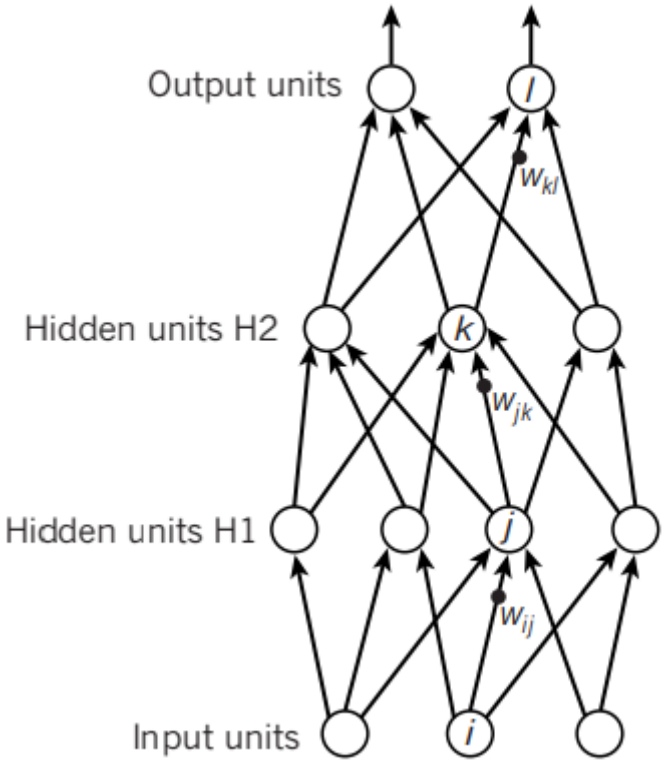

Figure 1. A simple artificial neural network with two hidden layers [12]

Each neuron in an artificial neural network (ANN) acts as an activation function to translate a summed input of weights and biases to a combined output [12]. Weights are scalar values that function as connections between neurons in a network while biases perturb the network to prevent over fitting. Figure 2 represents the connections between the neurons of an input layer with a single neuron hidden layer (also known as a single layer perceptron).

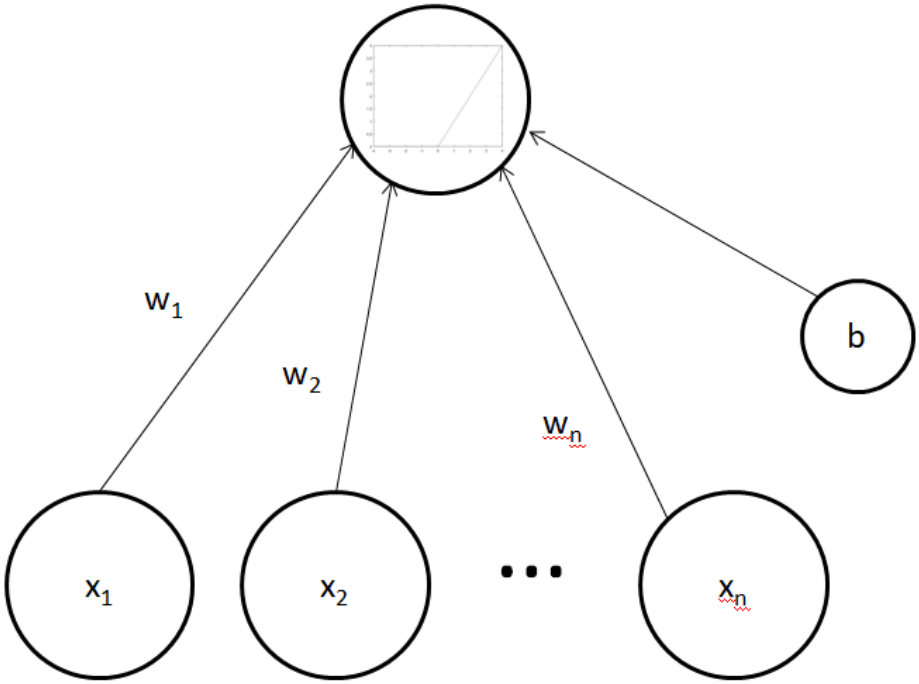

Figure 2. An artificial neuron with weight connections and bias

The activation function of a neuron acts as a classifier based on the input weights and biases. The input data is scaled based upon the weights connecting to the neuron in the next layer and each weighted input is then summed at the input of each neuron. The equation for the input of the neuron (pre-activation) can be seen in (1) where a(x) describes the input to the activation function while the transformed output equation g(x) can be seen in (1). The vector **w** represents the weights of the input layer while the vector **x** represents each input value. The scalar value b represents the bias.

$$a(x) = b + \mathbf{w}^T \mathbf{x} = b + \sum_i w_i x_i \tag{1}$$

$$h(x) = g(a(x)) = g(b + \sum_i w_i x_i) \tag{2}$$

A common activation function to utilize for a neuron is the sigmoid function. There are multiple benefits to using a sigmoid function: non-negativity (negative input values set to zero), upper bounded at 1 and lower bounded at 0, and always increasing; furthermore, the sigmoid function introduces non-linearity to the transformation of the input data which allows for the development of complex classifiers. The equation for a sigmoid function can be seen in (3) with a graphical representation in figure 3.

$$g(a) = sigm(a) = \frac{1}{1+\exp(-a)} \tag{3}$$

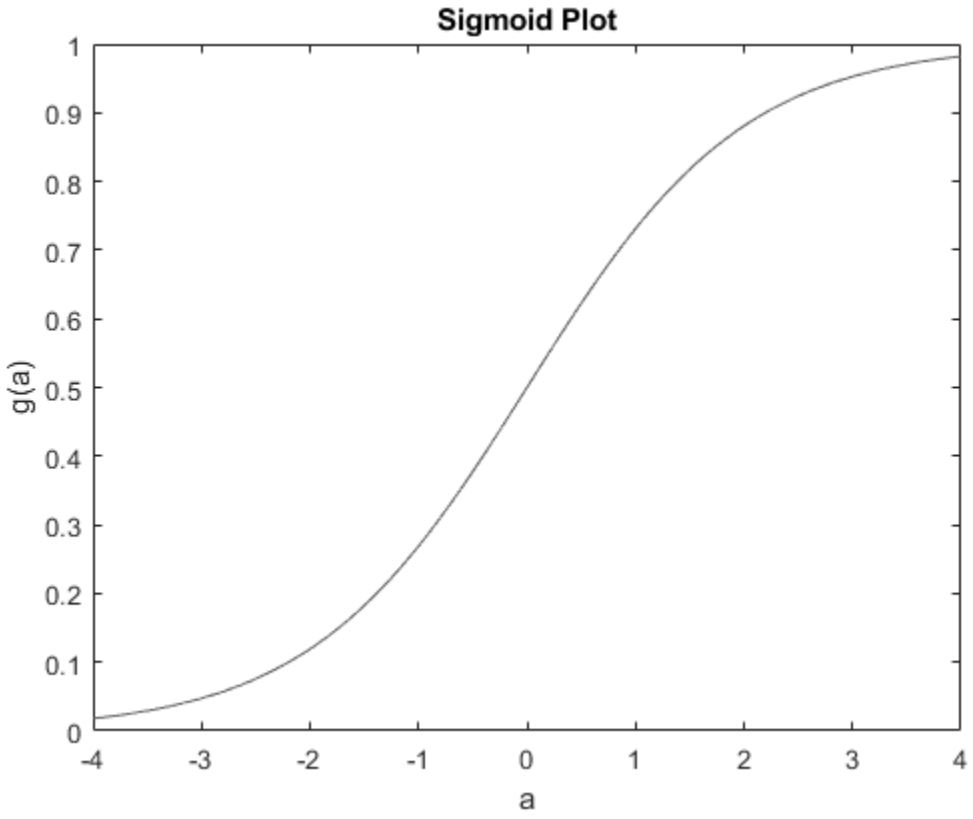

Figure 3. Sigmoid activation function

There are numerous choices for activation functions, but the rectified linear unit (ReLU) function has become the most popular in neural network design. The ReLU function can be calculated by taking the maximum value between 0 and the input value; thereby, restricting the output to non-negative values while retaining the value of non-negative inputs. The ReLU function has gained such notoriety as an activation function since the monotonically increasing positive portion of the function retains the positive inputs directly and the negative portion sets all negative inputs to zero, allowing the ReLU function to be non-linear while reducing processing requirements [13]. A graphical view of the ReLU function can be seen in figure 4.

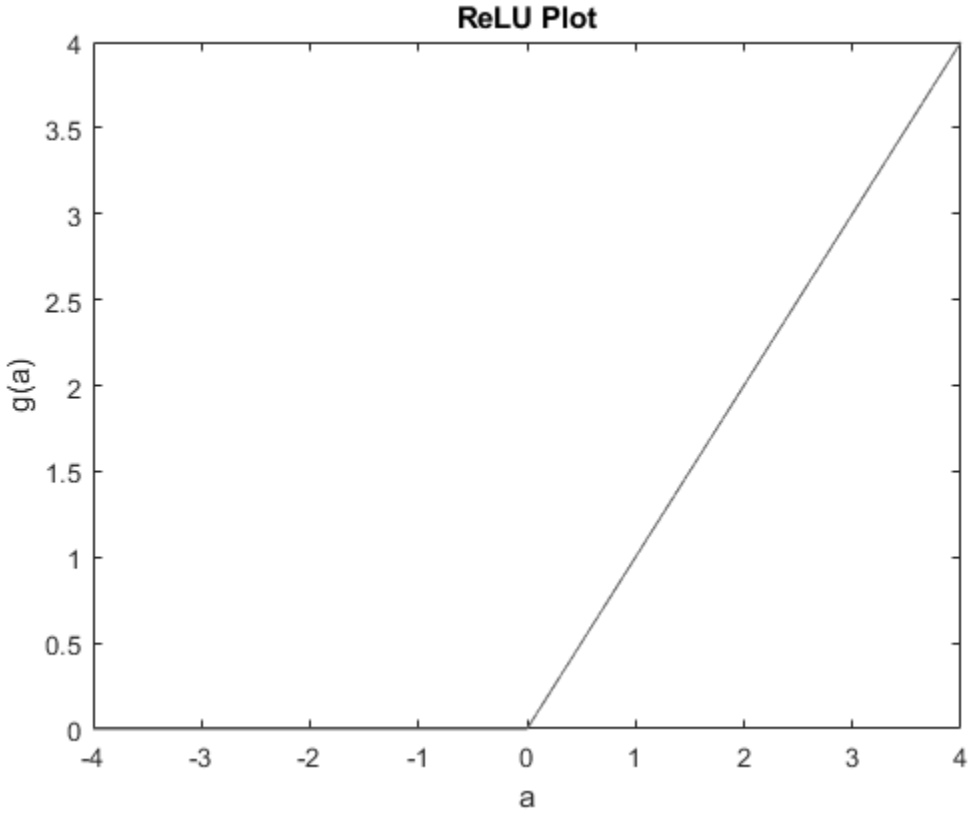

Figure 4. ReLU activation function

Consequently, the non-linearity provided by the aforementioned activation functions allows a neural network to make increasingly more complex classifications based upon the number of hidden layers in the network. At every layer, every neuron is connected to every neuron in the next layer; therefore, the non-linear transformation of the input data at the output of the first hidden layer will be subjected to even greater non-linearity at the output of the next hidden layer. A visualization of a more complex, two-layer network can be seen in figure 1 while a detailed view of a multiple-neuron single layer network can be seen in figure 5. The values x_1 through x_3 represent the inputs, the values w_{11} through w_{33} represent the weights between the input layer and the hidden layer, the values w_1 through w_3 represent the weights between the hidden layer and the output layer, and the values b_1 and b_2 represent the bias for the hidden layer and output layer respectively.

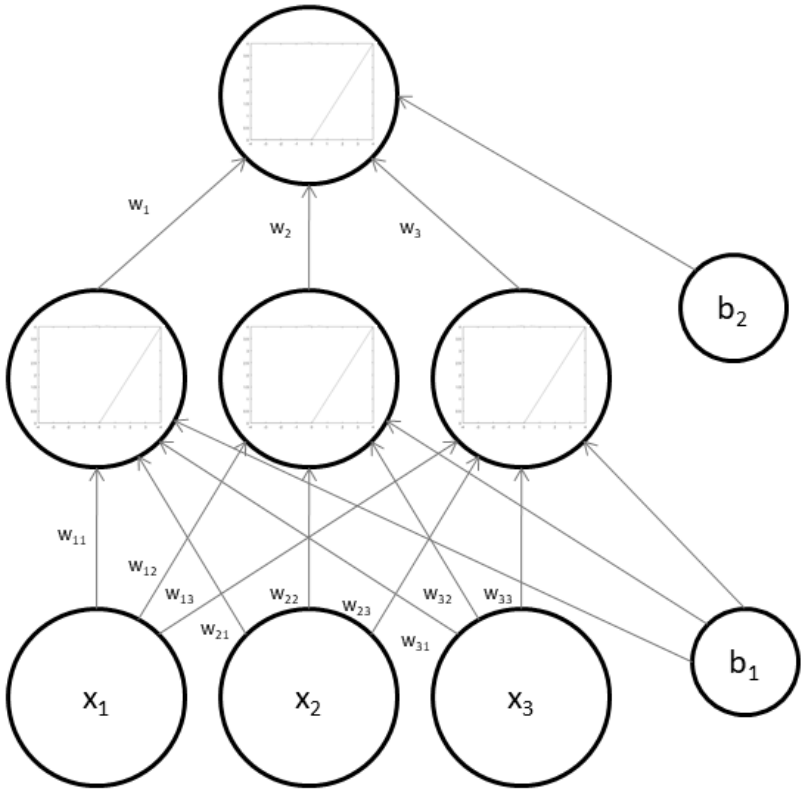

Figure 5. Multiple-neuron single hidden layer ANN

Neural network architecture primarily consists of the parameters: numbers of hidden layers, neurons within each hidden layer, input values at the input layer, and classes at the output layer [14]. The activation function is chosen as the ReLU function for CNN applications.

Training is performed once the neural network architecture is established. A database of labeled data is provided as input to the system. The network learns to classify this data based on the class labels. An error function is formulated to determine the accuracy of classification after each forward-pass through the neural network. Empirical risk is determined by comparing the predictions made at the output layer with the predetermined classes of the dataset [14]. The formula for empirical risk

minimization can be seen in (4) where l is the loss function, $f(x_i)$ is the predicted classification of the input x_i, y_i is the correct classification, and n is the number of inputs.

$$E_n(f) = \frac{1}{n}\sum_{i=1}^{n} l(f(x_i), y_i) \tag{4}$$

Subsequently, the network learns features of the dataset during back-propagation. Minimization of the error function calculated on the outset of a forward-pass is performed through standard optimization techniques. Stochastic gradient descent is a common choice in optimization algorithms that operates by subtracting the gradient of the gradient of the loss function from the weight values of the forward-pass to determine the adjusted weight values [15]. A step size is chosen to adjust the speed of convergence for the optimization algorithm. In machine learning, this step size is known as the learning rate. The stochastic gradient descent equation can be seen in (5) where Q is the loss function, Υ is the step size, and w is the weight value being calculated.

$$w_{t+1} = w_t - \Upsilon_t \Delta_w Q(z_t, w_t) \tag{5}$$

Ultimately, the weights of connections between each layer are updated every iteration of back-propagation until the desired number of iterations or desired accuracy measure has been achieved.

2.2 Convolutional Neural Networks

A central motivation of computer vision is object recognition: to take as an input 2D array describing an image and output a known class in which this image belongs. One property which allows CNNs to excel at object recognition is that local connectivity is emphasized. Object recognition can be accomplished with a typical ANN by vectorizing the image into a 1 dimensional vector and applying this vector as input to the

network, but this technique fails to consider the relationships between adjacent pixels in an image that highlight important features such as corners, edges, or textures.

CNNs are an extension of ANNs that calculate weights via back-propagation in a 2-dimensional space in order to create optimized filters to extract complex features. Each entry in the simple Gaussian blur filter in figure 6 becomes a variable weight. The optimized weight values produced by a CNN can extract accurate features that machine learning designers themselves cannot understand; therefore, CNNs are especially suited to excel at tasks in computer vision. The convolution operation is performed by taking the calculating the product of one function with another over all points of the original function. The formula for convolution between two generic functions *f* and *g* can be seen in (6) where *t* is the independent variable and τ is the variable that shifts the functions.

$$(f * g)(t) = \int_0^t f(\tau) g(t - \tau) d\tau \tag{6}$$

In computer vision, convolution can be leveraged to pass a relatively small filter (or kernel) over the entirety of an image. A kernel is a matrix of values that can manipulate an input image. The kernel representing a Gaussian blur filter can be seen in figure 6.

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Figure 6. Gaussian blur filter

2D convolution can be used to process images and extract features such as corners or edges. Gaussian blur filters, such as in figure 6, operate by taking the weighted product of all pixels starting at coordinate [0, 0] with the filter, and then taking the average over the sum of weights in the filter [16]. Through convolution, this product is taken between the filter and every pixel region as the filter passes over the image. An example image before and after convolution with a Gaussian blur filter can be seen in figure 7.

Figure 7. Image filtered with Gaussian blur

Each neuron in a hidden layer of a CNN is connected to all pixels in a small region of the image (determined by desired filter size) known as the receptive field for that activation unit. Additionally, the total number of computations required by the network to back-propagate through the weights is vastly decreased as compared to a fully connected network with each pixel having a weighted connection to each neuron. An example of receptive field connectivity can be seen in figure 8.

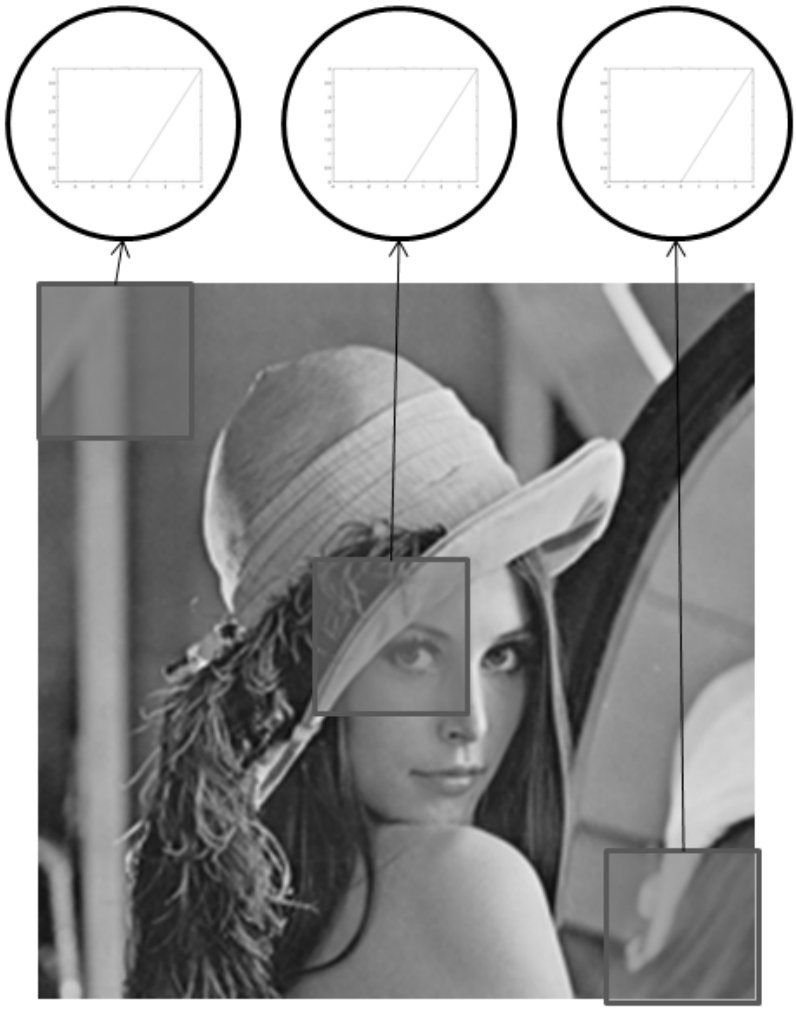

Figure 8. Local connectivity achieved through receptive fields for each neuron

Furthermore, neurons in a CNN are arranged into feature maps that determine complex features in an image and further reduce the computational complexity of the network. Feature maps contain enough neurons to cover the entirety of the input image in relation to the size of the filters being produced; thereby, calculating the weights for every available receptive field within each feature map. The weighted connections that determine the filter convolved with the receptive fields are shared between all neurons in a feature map, also known as parameter sharing, which allows for robust filter

development without higher computational cost. An example of parameter sharing through feature maps can be seen in figure 9, with each color representing shared kernel parameters.

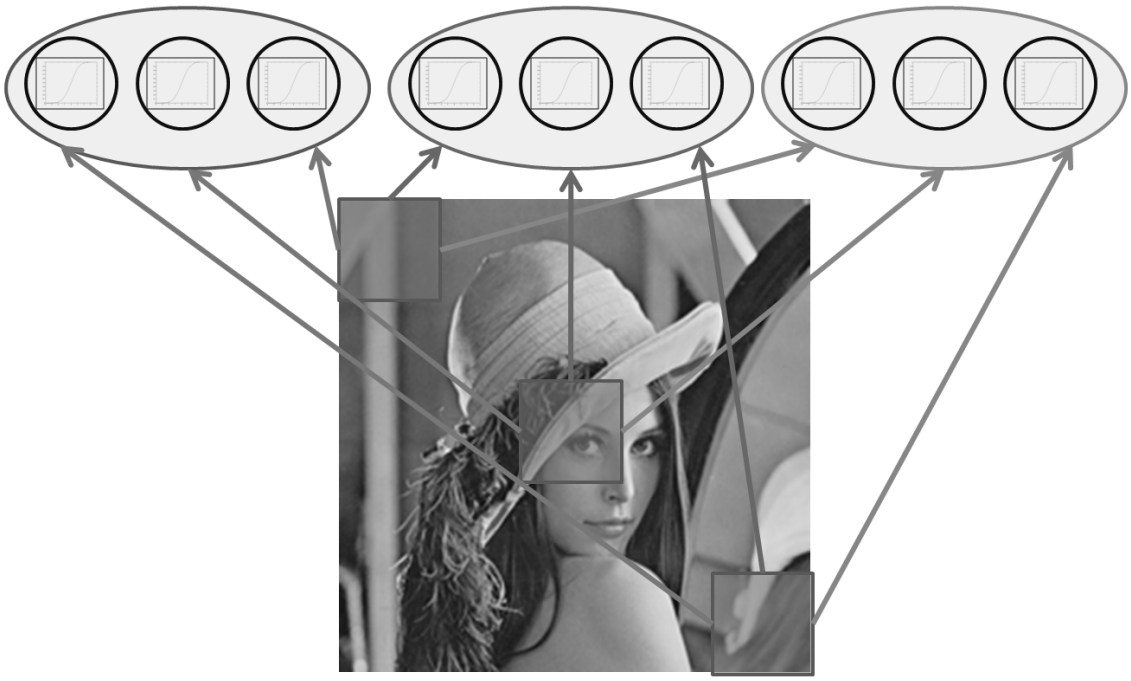

Figure 9. Parameter sharing via feature maps

Filters are calculated for every feature map in a hidden layer and then convolved with every receptive field in the image. The result of this operation is a singular feature extraction across the entire image for each feature map. Consequently, the number of feature maps determines the number of features that are extracted from the image at a certain hidden layer. The number of feature maps can vary between each layer depending on the desired number of features to be used for classification.

The weight matrix, denoted as $\boldsymbol{k_{ij}}$, contains the weights for the i^{th} channel in the image and j^{th} feature map. The kernel matrix is then convolved with the corresponding receptive field in the image, denoted as $\boldsymbol{x_{ij}}$. Then the output of the feature map after activation, denoted as $\boldsymbol{y_j}$, is calculated by taking the summation of the convolution across all channels in the image as input to the activation function as seen in (7).

$$y_j = g_j \tanh\left(\sum_i k_{ij} * x_i\right) \tag{7}$$

Expanding the feature maps into a cascading series forms the foundation for hidden layers in a CNN. As an image passes from one hidden layer to the next, increasingly complex features are extracted by the network. The initial hidden layer extracts features directly from receptive fields in the input image while subsequent layers extract features from the features extracted in the previous layer [17]. An example of features extracted by a deep neural network (one with many hidden layers) can be seen in figure 10.

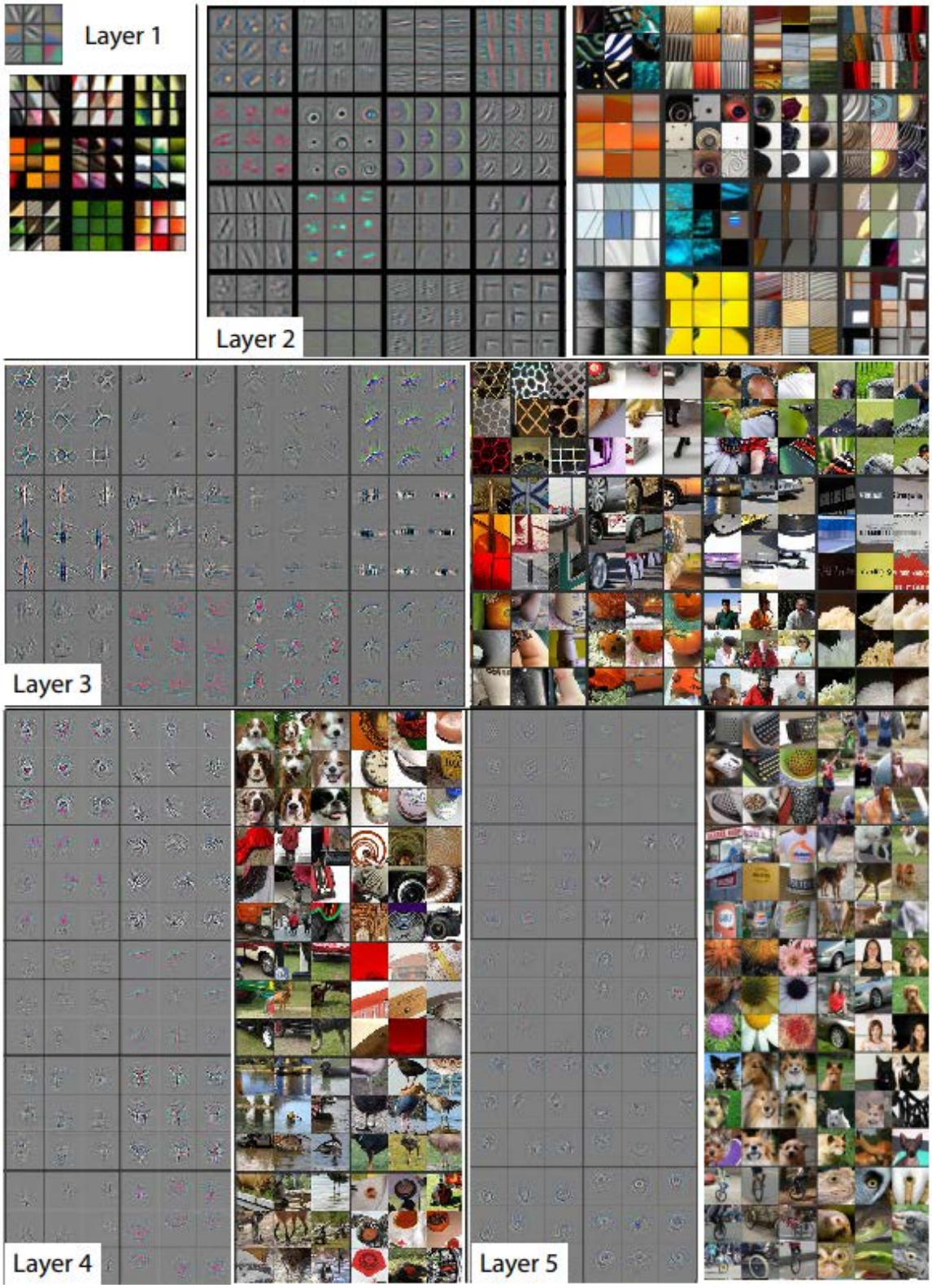

Figure 10. Features extracted by a deep convolutional neural network [17]

Additionally, the first and last layers of the network are the input and output layers, similar to the ANN. Images are passed to the activation functions of the first hidden layer through the weighted connections of feature maps. At the output layer exists a neuron for each of the predetermined classes that could describe the input image [14]. A full CNN architecture can be seen in figure 11.

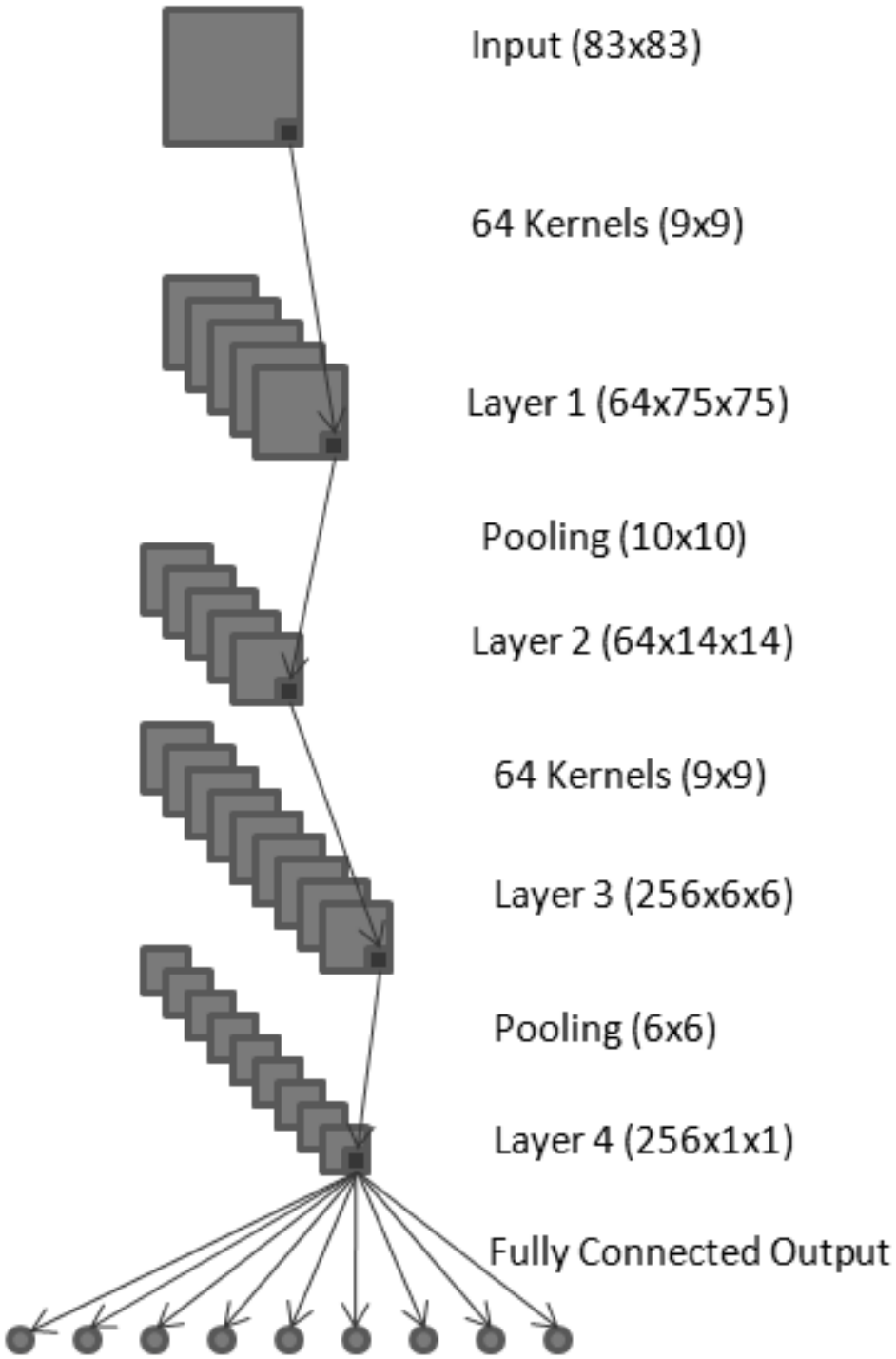

Figure 11. Block diagram of convolutional neural network

At the first hidden layer, the network calculates weights for the kernel connections between the layer and the input image. The weights are adjusted during each pass of back-propagation to extract the optimal features from the image for the desired classification criteria. The output of this layer is a filtered version of the input image per feature map [17]. For example, if a network was designed with 64 feature maps in the first layer, then the output of the first layer would contain 64 copies of the input image with each copy filtered by a separate kernel.

Subsequently, subsampling and pooling layers are typically utilized in CNN architectures to reduce the size of the filtered images propagating through the network; thereby, reducing computational complexity and increasing runtime efficiency. Furthermore, subsampling layers serve to highlight relevant features that span across local regions of the image and prevents over fitting by disrupting the input information [18]. This operation is performed until, at the input to the final layer, there is a vectorized string of 1x1 elements describing the features extracted from the image by the network. This vector is attached to a fully connected layer which is then classified in the output layer by the neurons signifying the possible class sets.

2.3 Data Augmentation and Synthetic Data Generation

Several preprocessing techniques have been developed to alleviate some of the issues that arise with the limited datasets. Data augmentation is one way to artificially increase the size of a database by duplicating and performing transformations on the original dataset [6], [7] and [8]. For example, one could perform a series of 90°, 180° and 270° rotations on each image to effectively quadruple the size of their database [4].

Furthermore, these transformations would make the model rotationally invariant, allowing accurate object detection regardless of how the object is oriented within the test images. However, data augmentation increases the computational complexity, which is undesirable especially for medical image analysis intended for diagnosis purposes.

Additionally, a database can be expanded through synthetic data generation [9]. Originally proposed as a solution to imbalanced classes, the Synthetic Minority Over-sampling technique has been utilized to increase the amount of training data in an underrepresented class [10], [11]. Each training class could be manipulated through this technique until all classes contain an equal quantity of training images. This concept could then be expanded to the dataset; generating synthetic data to increase the total size of the database. However, medical images are usually very difficult to imitate and include critical information that need to be extracted. By synthesizing these medical images, we may be losing some information that might help in diagnosis and treatment.

Chapter 3

The Big Picture

In this chapter, we will discuss the current procedure employed to generate brain MRI scans, debone each scan, segment the results, quantify brain structure information, and perform diagnostic predictions based upon this data. Furthermore, we will analyze the brain MRI deboning problem as a classification problem.

3.1 Brain MRI Processing

Magnetic resonance imaging (MRI) machines are employed in the medical field as a non-invasive diagnostic tool which can produce images of internal organs and bones. Brain MRI scans are often taken when patients are at risk of malignant brain tumors, changes in volume indicating Alzheimer's disease, stroke, multiple sclerosis, and many others. First, a patient must be scanned for several minutes in the MRI machine to produce suitable images. An example of an MRI scan can be seen in figure 12.

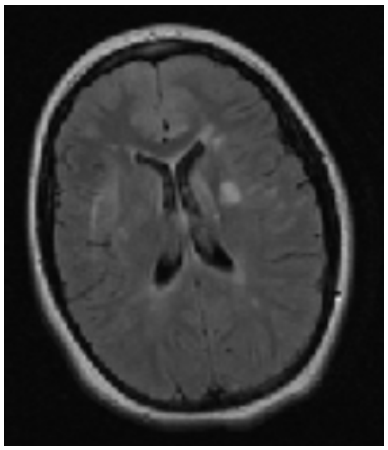

Figure 12. Standard MRI scan from volume of images produced by an MRI machine

Prior to diagnoses, skull structures that are unrelated to the illness under evaluation must be removed from each image. Currently, this process is performed either by hand or by manually adjusting a single variable within preprocessing software, such as the *FSL* software [19]. This variable must be readjusted for every image in each MRI set, thus increasing the time and labor for each diagnosis. The interface for the *FSL* software can be seen in figure 13.

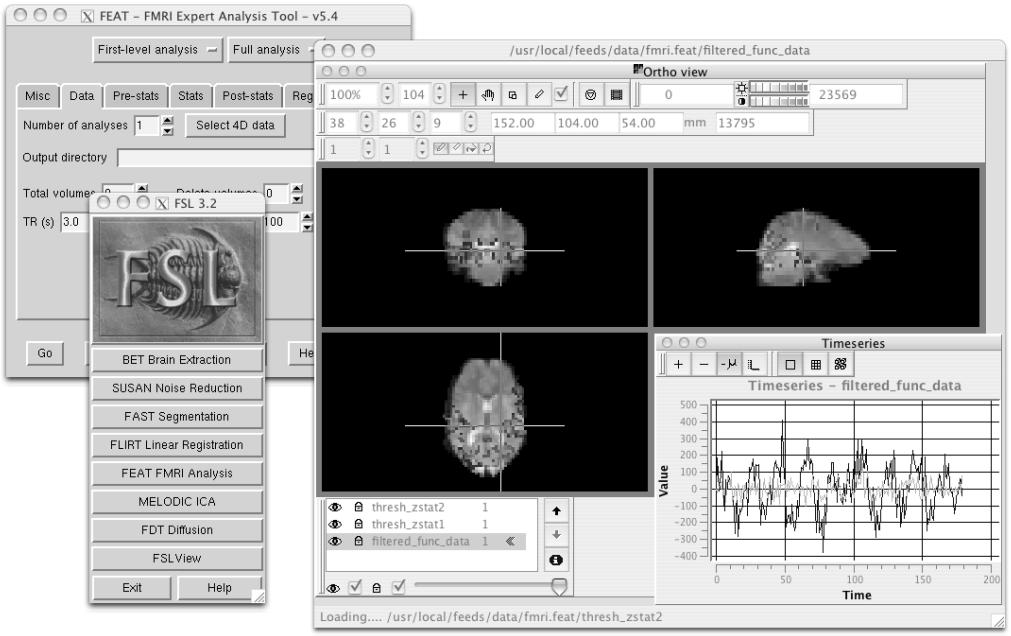

Figure 13. FSL software interface for deboning brain MRI scans

From there, the deboned brain MRI scan is segmented either by hand or through an automated method based on statistical modeling. One such modeling method is the Active Contour, which utilizes an energy minimization function to propagate a boundary toward connected structures in an image. Typically, physicians will segment brain MRI

scans by hand and through visual inspection. Once the structures of the brain are properly distinguished – with area and volumes quantified – then a diagnosis can be made based on this information.

3.2 Brain MRI Deboning as a Classification Problem

Brain MRI deboning can be framed as a classification problem by analyzing each scan on a pixel-by-pixel basis. Segmentation is performed to classify multiple objects or classes within a single image. In the deboning application, segmentation could be formulated as a binary classification problem of skull vs. brain. The process of segmentation involves patching to be applied to the dataset to extract local regions of a specified size from a variety of areas. An equal number of patches are extracted from the training set of images for each of the two classes. Each patch is classified by the central pixel in that patch. Once a balanced library of patches is derived from the dataset, the network is then trained on that library. The weights in the network are adjusted after each pass through the library of patches until all epochs are completed.

After training has been completed, testing can be performed on images that the network has not yet seen. The input images to the system are decomposed into patches of a specified size and input into the network. Each patch would then be analyzed by the network to classify the central pixel of that patch into one of the predefined classes of the training set. Image segmentation is complete once all pixels have been classified by the network. An example patch library extracted from a database of brain MRI images can be seen in figure 14.

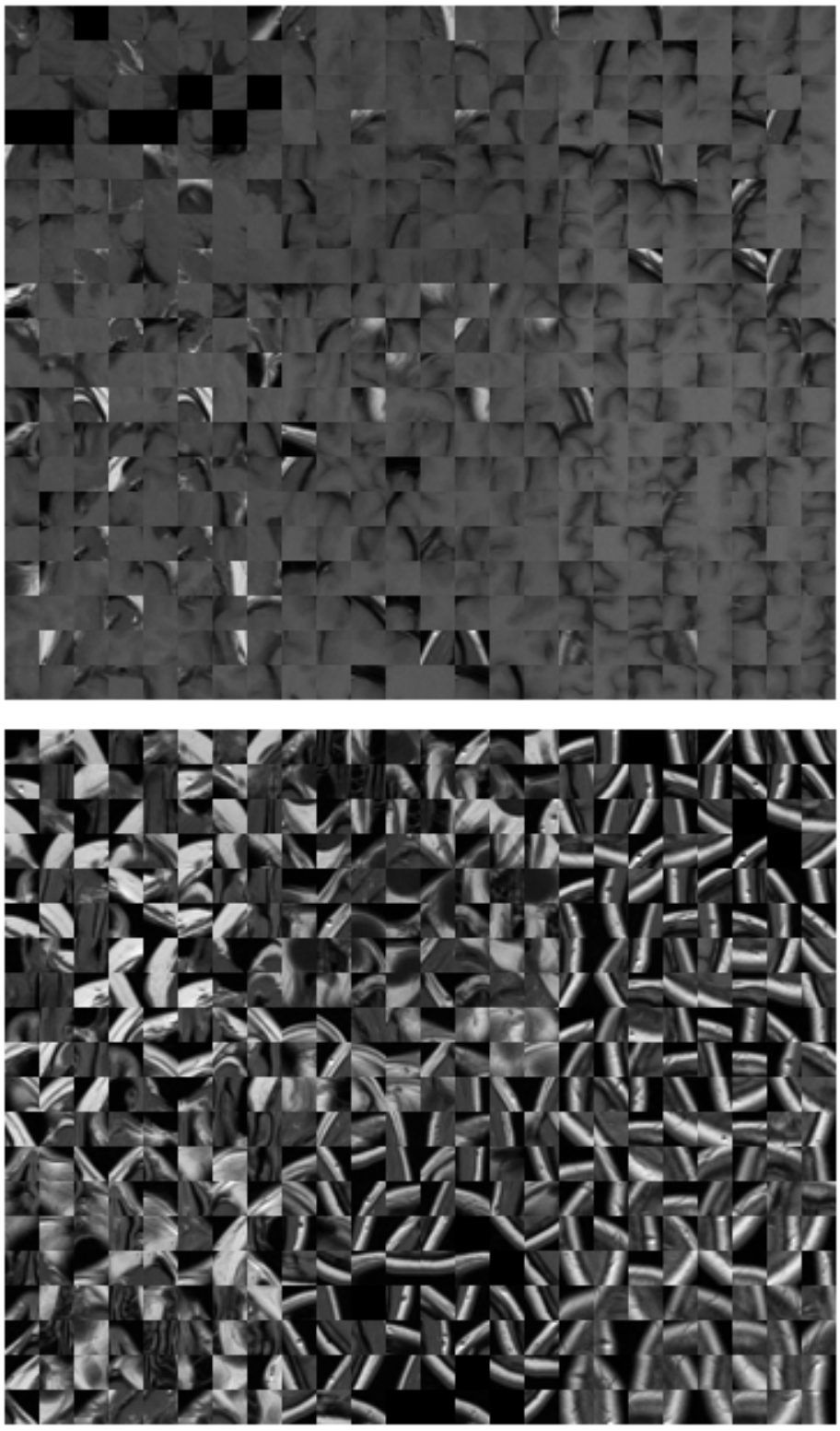

Figure 14. Patch library extracted from brain MRI database with brain patches in the top image and skull patches in the bottom image

Each pixel of the brain MRI database can be determined to exist in one of two classes: skull or non-skull. A training database can be constructed containing skulls that have been labeled by hand (each pixel of skull as a 1 and all other pixels as 0) by breaking all images in this dataset down into patches. Each small patch taken from an image in the dataset, such as a 15x15 square, can be utilized by a CNN to classify the central pixel of that patch based on the features of the surrounding 15x15 region. An example of a skull classified patch and a non-skull classified patch can be seen in figure 15.

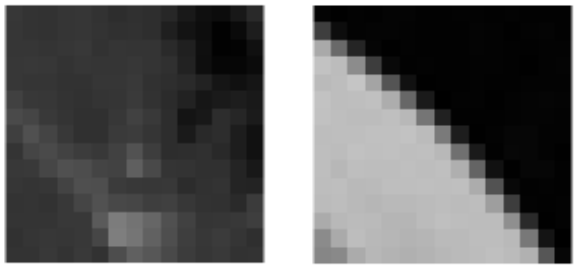

Figure 15. Skull and non-skull classified patches

Once a CNN has been trained upon the patch library extracted from the brain MRI database then new brain MRI scans can be segmented by the system. At the input to the network, each new scan will be broken down into a library of patches that encompasses all pixels in the image. Then central pixel of each patch will be classified based upon the trained network classifier. The result will be a pixel-by-pixel prediction by the network of where the skull exists in the image. A brain MRI along with the segmented result via CNN can be seen in figure 16.

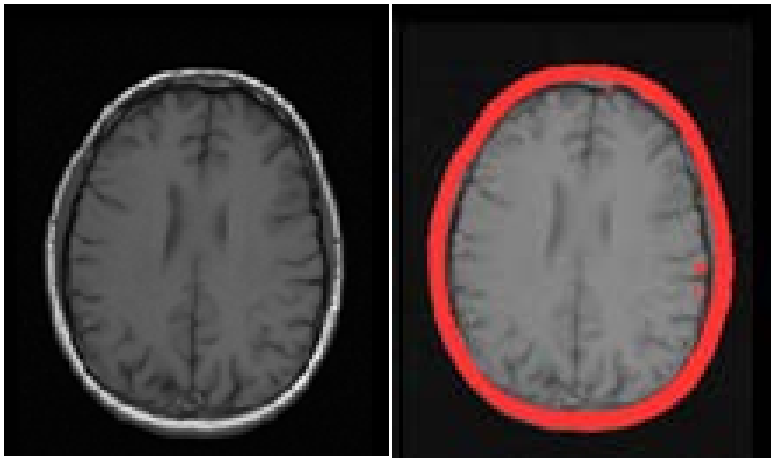

Figure 16. Skull segmentation of brain MRI scan

3.3 Inverted Cone Preprocessing

Although CNN architectures are robust for general object recognition with large and diverse training data, faults arise when specific applications lack a suitable database. The standard approach to dealing with this deficiency would be to artificially expand the training data through data augmentation and synthetic data generation. Both of these techniques involve adding additional data points to the database in the preprocessing step that do not truly exist in the dataset. The solution proposed in this thesis is to impose a constraint on the dataset in both preprocessing and during the testing of the CNN architecture based upon prior knowledge of the dataset.

The application explored in this paper is the deboning process for brain MRI scans. The proposed modification, named the Inverted Cone Method, utilizes the known order of images in an MRI dataset to remove the most complex skull structures prior to process by the CNN.

Chapter 4

Inverted Cone CNN

In this chapter, we will be exploring the Inverted Cone CNN to debone brain MRI scans. We will discuss how the nature of the dataset can be leveraged to increase accuracy when spatial relationships are determined. The database of MRI scans used for this work will be described and the Inverted Cone CNN will be dissected.

4.1 Temporal Derivative

The Inverted Cone method relies on a sequentially ordered dataset, such as in a stream of video or a series of MRI scans, to isolate relevant information in more complex images. In the deboning application, all images in an individual patient's MRI scan are ordered in layers from the base to the top of the skull and separated by a constant thickness per slice.

MRI scan slices that are taken from the top to the middle of the skull are easily segmented by both visual inspection and through CNN processing. At these locations, the skull is present in a well-defined ring around the brain. As the scans descend further into the skull, sinus cavities begin to appear in the skull structure as the area occupied by the brain shrinks.

MRI slices that have been taken closer to the base of the skull introduce highly irregular areas and deviate greatly from the typical central slice. In these lower images, the sinus cavities and eye sockets create more complex structures to classify. An example of a complex MRI slice containing eye sockets can be seen in figure 17.

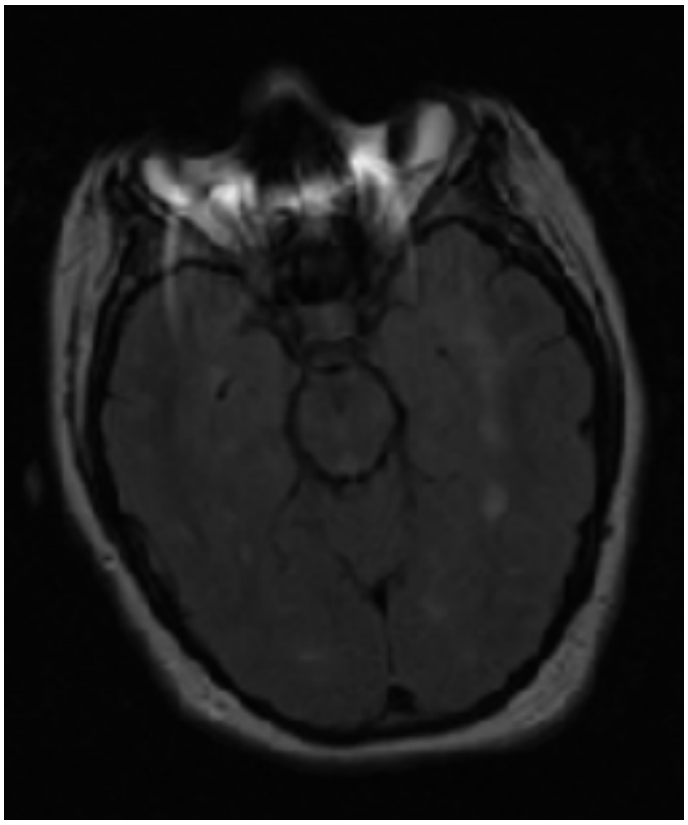

Figure 17. Complex MRI scan containing eye sockets and sinus structures

The ordered and related nature of the brain MRI images allows the use of preprocessing to remove the most difficult to classify sections of the skull. By working from the central slices outward, the MRI images with the largest area of brain can be leveraged to remove extraneous skull structures in the more complex scans.

The difference is taken between each image with a larger area of brain and the next image in the dataset to highlight the relevant area of analysis in that next image. The skull structures that result from this difference can be removed from the subsequent images to reduce presence of the most difficult to classify areas.

4.2 The Inverted Cone CNN

To construct the Inverted Cone CNN, a preprocessing system was created that must be applied during both the training and the testing phases of the CNN. The training set is prepared with this system prior to training the CNN, and then applied in a feedback loop during the testing of the network. A block diagram depicting the overall process for the Inverted Cone CNN can be seen in figure 18.

First, the training set of images must be preprocessed through the Inverted Cone method. The preprocessing was performed by finding the MRI slice for a specific patient with the largest area of brain in the ground truth. The resultant image would be from the central area of the MRI volume, and the difference would then be taken between this image and the subsequent image for that patient's MRI. This difference would include all irrelevant skull structures that fall outside the immediate area surrounding the brain. This process was then performed upon all the following images; thereby, the MRI sets for each patient were simplified based on a central slice for each patient. A CNN architecture would then be trained upon this modified dataset.

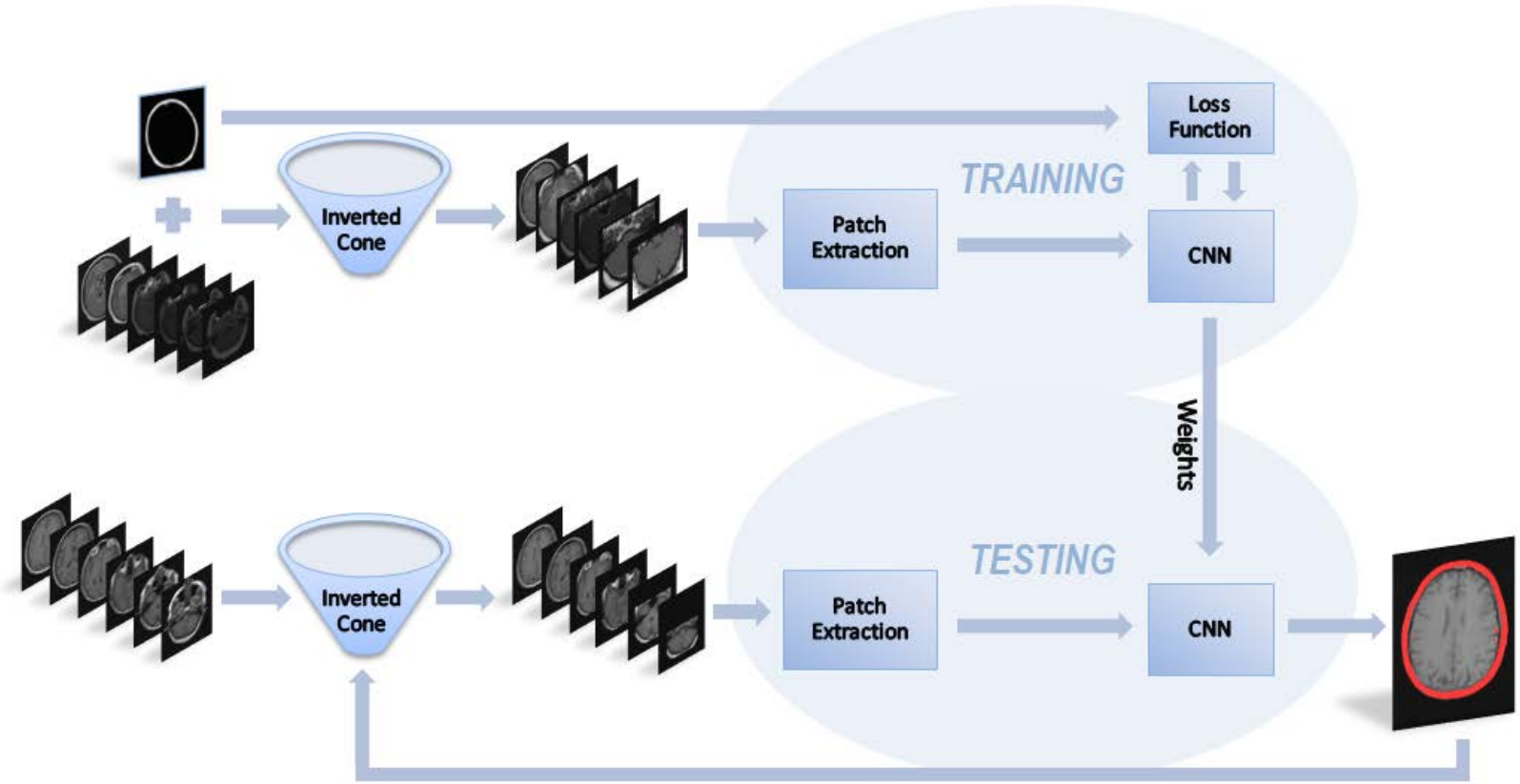

Figure 18. Block diagram of Inverted Cone CNN

During the testing phase, the Inverted Cone method was applied in a similar fashion. When classifying a new set of MRI scans, the scans were input into the system starting with the slices that are taken at the top of the skull. The area of the brain that is identified in each slice will be stored and compared to the area in the subsequent slice, until the central slice with the largest area of brain is discovered. This central slice would then be utilized to identify extraneous skull structures in the following image. Once these structures are removed from the slice, the image would be processed as an input to the network. The segmented result is then fed back as an input to the network to compare with the next image in the set, thus creating a gradient of change around the immediate area of the brain. This process is performed on each subsequent image until the entire MRI set for a patient is classified and deboned.

4.3 Database

The Inverted Cone CNN was applied to a database of anonymized gadoliniumenhanced T1-Weighted MRI images of human brain with and without malignant glioblastoma multiforme, a malignant brain tumor. The database, provided by the University of Alabama at Birmingham School of Medicine, was comprised of patient files containing a series of MRI scan slices for each patient. Patient files which held MRI scans that did not equal the common size of 256x256 were removed from the database. The resulting size of the dataset that was utilized in the training of this system was 23 patients with 17-38 slices per MRI scan. Ultimately, the system was trained with 451 MRI slices.

4.4 Preprocessing

For the application of deboning, a library of 300,000 patches was created for a data set consisting of 23 patients and between 17-38 slices per MRI scan. The total number of images in the dataset was 451 MRI slices. Patch sizes of 33x33 and 15x15 were investigated in this model. Larger patch sizes allow for a larger region of features to be analyzed around the pixel being classified which makes the network more robust to local structures, yet increases computational complexity and overfitting due to a loss of resolution. Ultimately, a patch size of 15x15 was chosen for the final model.

The Inverted Cone method was applied to the deboning dataset during preprocessing to remove complex skull structures at the base of the skull. The ground truth for each image following the central slice was used to filter the subsequent slice for that patient. Once the difference was taken between the areas of the brain in the previous image with the second image, the difference could then be removed from the second image as extraneous skull structures. A series of brain MRI slices preprocessed using the Inverted Cone method can be seen in figure 19 with the processed scans on the left hand side and unprocessed scans on the right hand side.

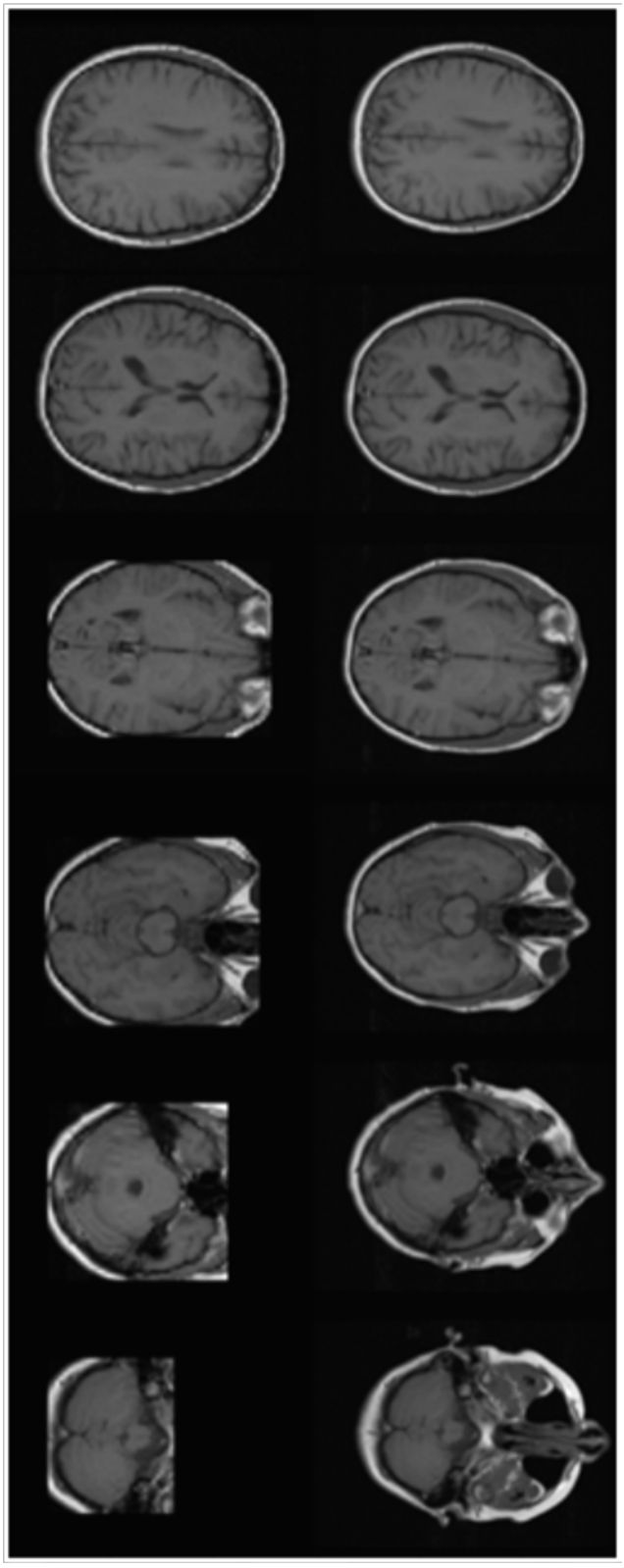

Figure 19. Brain MRI scans preprocessed by Inverted Cone

Chapter 5

HPC Implementation

In this chapter, we will discuss the application of high-performance computing technologies to efficiently train and test the CNN employed in the brain MRI deboning problem.

5.1 System Specifications

The Rowan University high-performance computer provided the capability to train and test the Inverted Cone CNN with an extensive dataset consisting of 300,000 15x15 patches extracted from 451 brain MRI scans. Each node of the HPC acts as a powerful processing unit with high memory nodes being allocated even greater resources. A diagram of a typical HPC cluster architecture can be seen in figure 20.

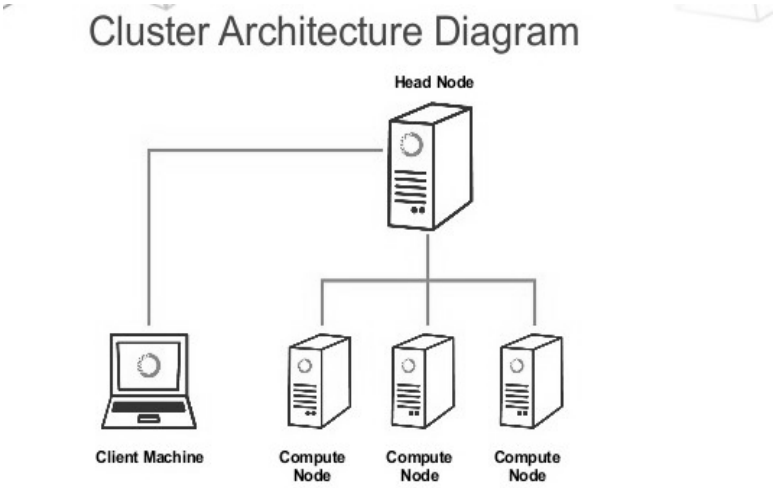

Figure 20. HPC node cluster diagram

The HPC at Rowan University is comprised of 59 nodes and 1,372 physical processing cores with 7,616 GB of RAM. An example image of the Rowan University HPC can be seen in figure 21.

Figure 21. HPC at Rowan University

Three types of nodes exist on the Rowan University HPC: compute nodes, high-memory nodes, and graphics processing unit (GPU) nodes. The GPU nodes – on which this system was trained – contain 64 GB of RAM, 10 core processors, and 2 NVIDIA Tesla K20 graphics cards. A detailed specifications list for the GPU nodes, high-memory nodes, and compute nodes can be seen in table 1.

Table 1

Specification list for Rowan University HPC

	Compute Nodes
CPU	2x Intel Xeon E5-2670v3 2.3 GHz (12-Core)
RAM	64 GB (8 x 8 GB)
System Disks	1x 240 GB Intel DC S3500 Series MLC SSD
Storage Disks	1x 1 TB Seagate Constellation ES.3
Switch	Mellanox 1-Port FDR Infiniband
GPU	N/A
	High Memory Nodes
CPU	2x Intel Xeon E5-2670v3 2.3 GHz (12-Core)
RAM	512 GB (16 x 32 GB)
System Disks	1x 240 GB Intel DC S3500 Series MLC SSD
Storage Disks	1x 1 TB Seagate Constellation ES.3
Switch	Mellanox 1-Port FDR Infiniband
GPU	N/A
	GPU Nodes
CPU	2x Intel Xeon E5-2670v3 2.3 GHz (12-Core)
RAM	64 GB (8 x 8 GB)
System Disks	2x 240 GB Intel DC S3500 Series MLC SSD
Storage Disks	1x 1 TB Seagate Constellation ES.3
Switch	Mellanox 1-Port FDR Infiniband
GPU	2x NVIDIA Tesla K20m GPU 5 GB GDDR5

The size of the dataset that can be used for training the neural network was primarily restricted by the quantity of RAM on the computing system since the patch library must be stored in memory; therefore, the utilization of the HPC allowed for a

much greater variety of training data with the expanded memory capabilities. Additionally, multiple models could be run on separate nodes in order to rapidly prototype an array of hyperparameter values to find those ideal for the brain MRI deboning application.

5.2 Python, Tensorflow, and Keras

Python was chosen as the language to complete this thesis as there are extensive libraries for neural networking and machine learning that are constantly being expanded by community development. Limitations arose in attempting to develop using MATLAB deep learning toolboxes since they are not an open-source development platform, and thus the toolboxes restricted the ability to train a complex convolutional system. Training the Inverted Cone CNN model on the HPC required that a GPU node be configured to execute Python code with Tensorflow and Keras neural network libraries. A series of steps were involved in connecting to the HPC and properly setting up the environment therein.

Firstly, a virtual environment was created and activated to work within the HPC cluster. Within this environment, the Python *pip* functionality must then be upgraded to allow for the installation of Tensorflow.

Secondly, the Tensorflow machine learning framework was installed on the virtual environment. Tensorflow is an open-source platform designed by the Google Brain team that has been a primary tool in machine learning development since its release to the public on November 9th, 2015 [20]. Additionally, the open-source API Keras was installed to run on top of Tensorflow to provide additional tools and ease-of-use. These

libraries allow the simple creation of layers with inputs and outputs to construct neural networks of any variety.

Finally, the Compute Unified Device Architecture (CUDA) and the CUDA Deep Neural Network (cuDNN) library were installed in the environment in order to access the full functionality of the NVIDIA GPUs on the GPU node. The CUDA API allows for parallel computing tasks involving the GPU while the cuDNN library allows for faster performance of machine learning algorithms with optimization of runtimes. A diagram of the processing flow for the CUDA API can be seen in figure 22.

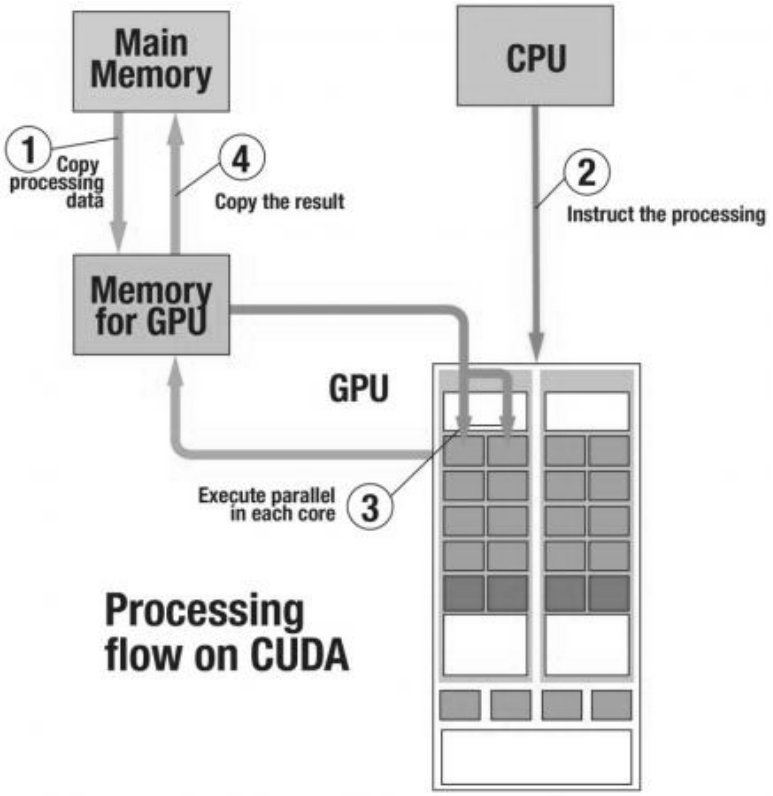

Figure 22. Processing flow diagram for CUDA API

Data is copied from the main memory to the memory for the GPU which can pass that data for processing to the GPU cores in a parallel execution by instructions from the central processing unit (CPU). Once the GPU cores process the data, the results can be copied back into the main memory of the system.

5.3 Utilizing the HPC

Once an account is obtained, the high-performance computer can be accessed directly through the Rowan University campus network or virtually through a virtual private network (VPN) connection. The Cisco AnyConnect Client was used to access the Rowan University network when off-site. An image of the VPN client can be seen in figure 23.

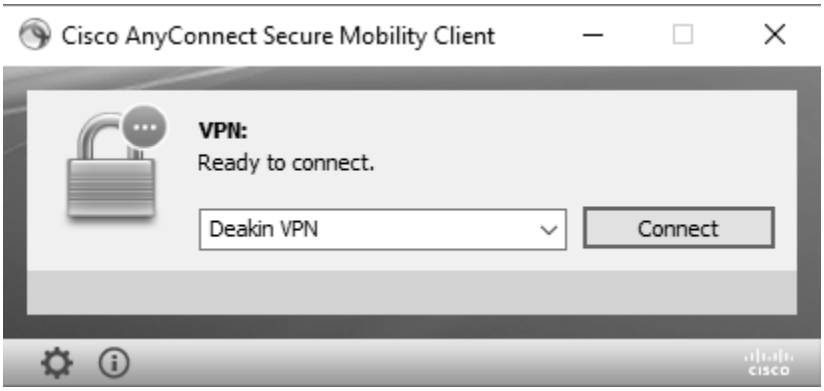

Figure 23. Cisco AnyConnect Secure Mobility Client interface

Once connected to the network, a Secure Socket Shell (SSH) client was used to remotely connect to the HPC cluster. From there, Tensorflow and Keras were imported to allow execution of the Python-based Inverted Cone CNN architecture. MobaXterm

was used to simplify file management on the system. The interface of this software can be seen in figure 24.

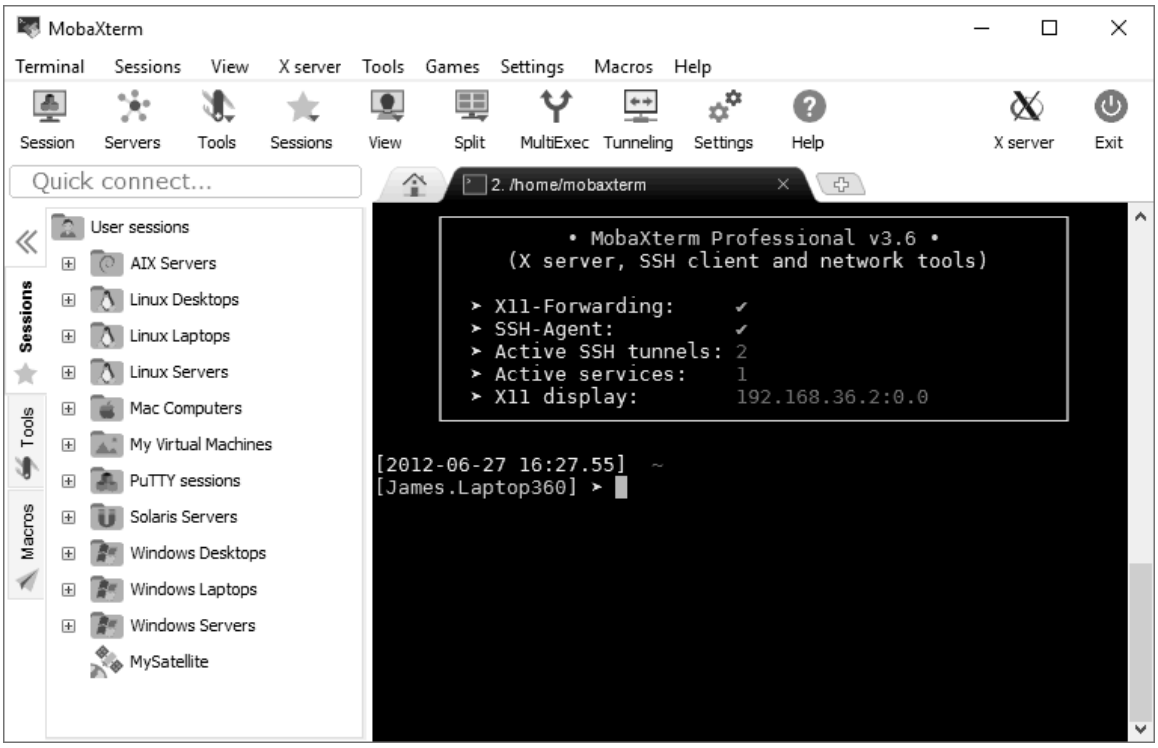

Figure 24. MobaXTerm SSH Client interface

The database of 451 brain MRI scans was copied onto the HPC along with the code-base for the Inverted Cone CNN method. At runtime, a library of 300,000 patches was stored into memory on the operating GPU node and trained on two NVIDIA Tesla K20 graphics processors. The system then iterated through a directory containing test images unseen by the system to determine validation accuracy.

Chapter 6

Simulation Results and Discussion

In this chapter, we will evaluate the performance of the Inverted Cone CNN on deboning brain MRI scans against a standard CNN model and discuss the implications of those results.

6.1 Performance Evaluation and Discussion

The architecture utilized in the Inverted Cone CNN was a three-layer convolutional network followed by two densely connected layers and a classification layer at the output. Three hidden layers in this network could extract a better feature-set from the data then a shallow network with only a single layer. An increase in the number of layers past three resulted in a decrease of overall accuracy due to over fitting. A standard ReLU activation function was used in each layer along with batch normalization to prevent over fitting. Batch normalization ensures that the mean activation of the previous layer is close to zero and the standard deviation is close to one. The full list of hyperparameters for this network is displayed in Table 1.

Kernel sizes were chosen to be cascading in size from 7x7 for the first layer, 5x5 for the second layer, and 3x3 for the third layer. Smaller kernel size allowed for a three-hidden layer design with a small patch size of 15x15. Larger kernels were employed at the outer layers to extract features with more locality information. The kernels decrease in size in the two subsequent layers in order to reduce the number of weights and deter over fitting.

Table 2

List of hyperparameters for Inverted Cone CNN

Hyperparamter	Value
Patch dimensions	15x15
Patch quantity	300000
Kernel dimensions	7x7, 5x5, 3x3
Kernel quantity	8, 8, 8
Learning rate	0.001
Decay	0.01
Momentum	0.9
Epochs	10
Batch size	256
Database size	451

The architecture described in Table 1 was trained with and without using the Inverted Cone method to process the inputs. The results from the Inverted Cone CNN deboning were compared to a standard CNN as well as the widely used *FSL* deboning software. The accuracy measurements for the *FSL* software was acquired by manually adjusting the parameter for the central slice of each test MRI set and using this parameter for all other slices in the set. Accuracy, validation accuracy, and loss measurements for the two CNN techniques are shown in Table 2. The Inverted Cone CNN outperformed the standard CNN model and the *FSL* software on this dataset. Accuracy and validation accuracy were plotted for both CNN models over the 10 epochs, as seen in figure 25.

Table 3

Accuracy, validation accuracy, and loss

Measurement	Standard CNN	Inverted Cone CNN
Loss	0.5112	0.4333
Accuracy	0.8204	0.8431
Validation Accuracy	0.7748	0.8567

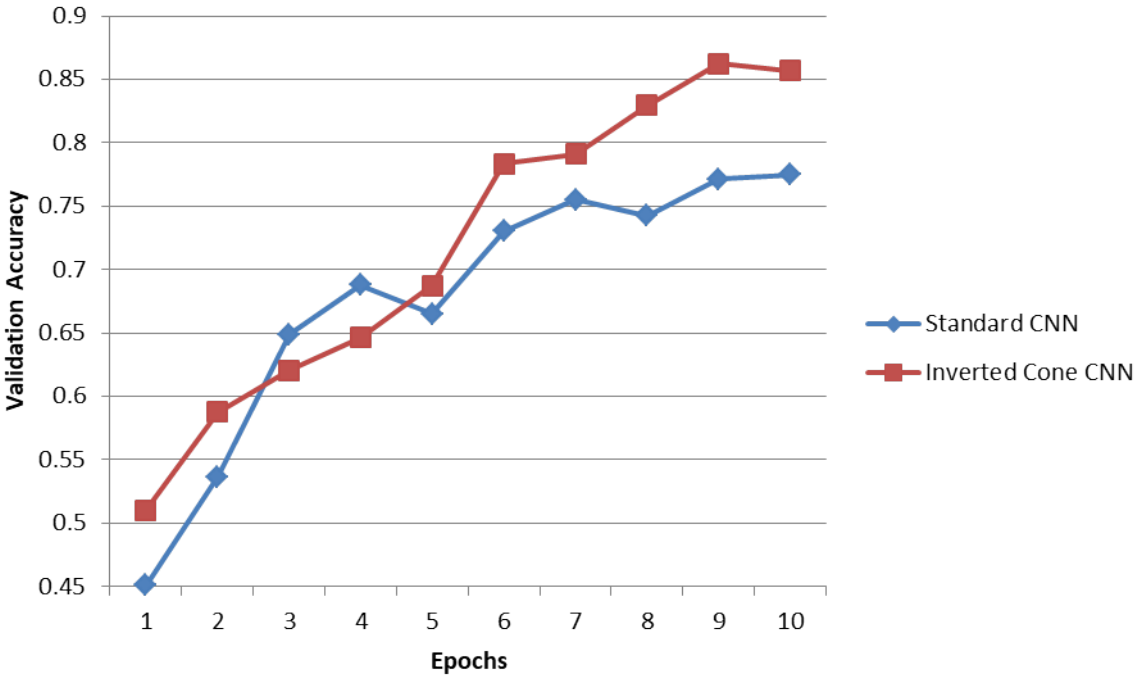

Figure 25. Validation accuracy for standard and inverted cone CNN.

The Inverted Cone CNN outperformed the standard model in validation accuracy. The results show that utilizing the ordered nature of the brain MRI scans during preprocessing can reduce the complexity of the dataset and provide the system with a

greater capacity to learn the data. Figure 26 contains both simple and complex images segmented using the standard CNN model and the modified Inverted Cone CNN. As can be observed, the Inverted Cone CNN could segment the validation set more accurately as compared to the ground truth than the standard CNN architecture for both complex and simple MRI slices. In the complex image, the Inverted Cone model was able to identify skull structure immediately bordering the brain more readily than the standard model.

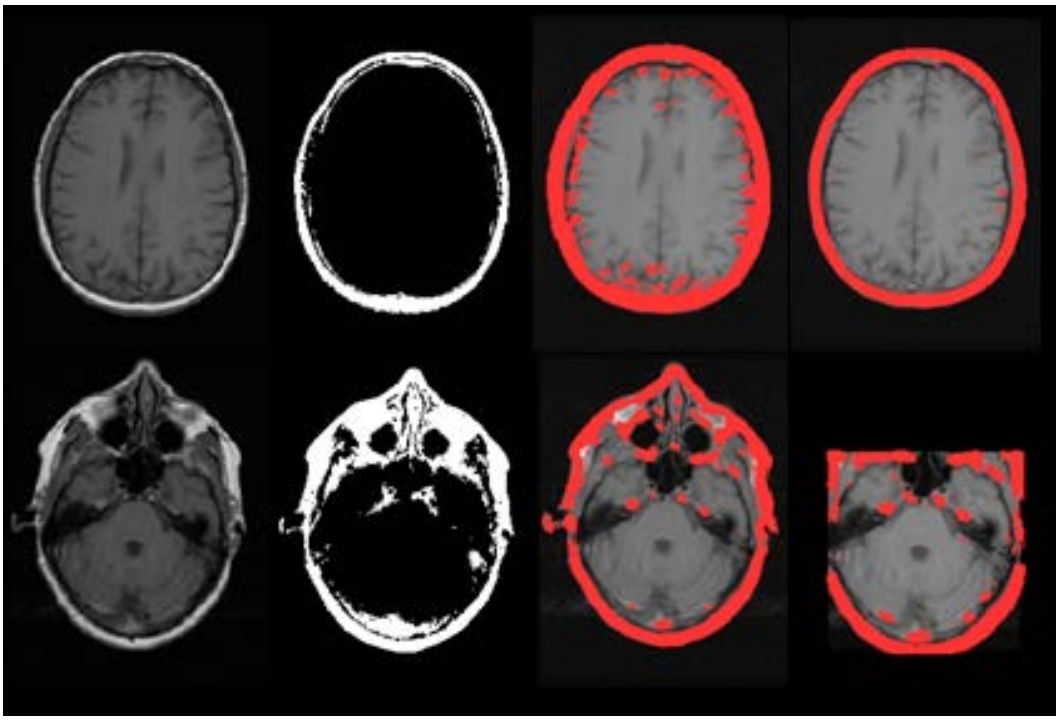

Figure 26. Segmentation Results: 1st column: Original MRI scans; 2nd column: Ground truth deboning; 3rd column: Segmentation of the standard CNN; 4th column: Segmentation of the Inverted Cone CNN; 5th column: Deboning of the FSL software.

Chapter 7

Summary and Future Work

Modifications to training data for a CNN are vital in situations where the quantity of that data is limited. In the past, data augmentation has been used to multiply the size of the database through duplicating and transforming available images. Synthetic data generation has also been employed to create additional, artificial images to the dataset. Knowledge of intrinsic attributes for a set of images can be leveraged in the training and testing of a network through specific preprocessing operations to increase the overall accuracy of the system. Two such attributes, order and spatial relation, were incorporated into the preprocessing for the application of deboning brain MRI scans.

Since MRI scans are oriented from the top to the base of the skull in sequential order, each scan is a slight gradient from the previous scan. When descending further into the slices of a scan, the area of the brain occupying the slice grows. Knowledge of the ordered nature of the data along with the relationship between subsequent images allows the Inverted Cone preprocessing method to be employed. Once the slice with the largest area of brain is determined, which will be a central slice in the scan, each slice afterwards can be filtered by the previous slice. The only area of interest -- the brain -- decreases in size from the central slice to the base of the skull while the size and complexity of the skull structures increase.

By ignoring the skull structures that fall outside of the area occupied by the brain in the previous slice, the classification problem becomes much simpler for the neural network to learn. The most difficult to classify images, which contain sinus cavities,

become more similar in shape and relative area of each class when the inverted cone processing is applied; thereby increasing the overall accuracy of the system.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks.” NIPS, pp. 1106 – 1114, 2012.

[2] Y. J. Cha, W. Choi, and O. Buyukozturk, “Deep learning-based crack damage detection using convolutional neural network.” Computer-Aided Civil and Infrastructure Engineering, vol. 32, pp. 361-378, 2017.

[3] P. Moeskops, M. A. Viergever, A. Mendrik, L. S. de Vries, M. J. N. L. Benders, and I. Igum, “Automatic segmentation of mr brain images with a convolutional neural network,” IEEE Transactions on Medical Imaging, vol. 35, 2016.

[4] S. Pereira, A. Pinto, V. Alves, and C. A. Silva, “Brain tumor segmentation using convolutional neural networks in MRI images,” IEEE Transactions on Medical Imaging, vol. 35, pp. 1240 – 1251, 2016.

[5] B. H. Menze, A. Jakab, and et al, “The multimodal brain tumor image segmentation benchmark (brats),” IEEE Transactions on Medical Imaging, vol. 34, pp. 1993 – 2024, 2015.

[6] J. Lemley, S. Bazrafkan, and P. Corcoran, “Smart augmentation learning an optimal data augmentation strategy,” IEEE Access, vol. 5, pp. 5858 – 5869, 2017.

[7] J. Ding, X. Li, and V. N. Gudivada, “Augmentation and evaluation of training data for deep learning,” 2017 IEEE International Conference on Big Data (BIGDATA), 2017.

[8] I. Oliveira, J. Medeiros, and V. de Sousa, “A data augmentation methodology to improve age estimation using convolutional neural networks,” 2016 29th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI), 2016.

[9] M. Babaee and A. Nilchi, “Synthetic data generation for x-ray imaging,” 21st Iranian Conference on Biomedical Engineering (ICBME 2014), 2014.

[10] J. W. Anderson, K. E. Kennedy, and L. B. Ngo, “Synthetic data generation for the internet of things,” 2014 IEEE International Conference on Big Data, 2014.

[11] S. C. Wong, A. Gatt, and V. Stamatescu, “Understanding data augmentation for classification: When to warp?” 2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA), 2016.

[12] Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436-444. DOI: 10.1038/nature14539

[13] Glorot, X., Bordes, A. & Bengio. Y. Deep sparse rectifier neural networks. In Proc. 14th International Conference on Artificial Intelligence and Statistics 315–323 (2011).

[14] Schmidhuber, Jürgen. “Deep Learning in Neural Networks: An Overview.” *Neural Networks*, vol. 61, 2015, pp. 85–117., doi:10.1016/j.neunet.2014.09.003.

[15] Wijnhoven, R.g.j., and P.h.n. De With. “Fast Training of Object Detection Using Stochastic Gradient Descent.” *2010 20th International Conference on Pattern Recognition*, 2010, doi:10.1109/icpr.2010.112.

[16] E. S. Gedraite and M. Hadad, “Investigation on the effect of a gaussian blur in image filtering and segmentation,” in ELMAR Proceedings, 2011.

[17] Zeiler, Matthew D., and Rob Fergus. “Visualizing and Understanding Convolutional Networks.” *Computer Vision – ECCV 2014 Lecture Notes in Computer Science*, 2014, pp. 818–833., doi:10.1007/978-3-319-10590-1_53.

[18] Scherer, Dominik, et al. “Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition.” *Artificial Neural Networks – ICANN 2010 Lecture Notes in Computer Science*, 2010, pp. 92–101., doi:10.1007/978-3-642-15825-4_10.

[19] S. Smith, M. Jenkinson, M. Woolrich, C. Beckmann, T. Behrens, H. Johansen-berg, P. Bannister, M. Luca, I. Drobnjak, D. Flitney, R. Niazy, J. Saunders, J. Vickers, Y. Zhang, N. Stefano, J. Brady, and P. Matthews, “Advances in functional and structural MR image analysis and implementation as FSL,” NeuroImage, vol. 23, pp. 208 – 219, 2004.

[20] M. Abadi, A. Agarwal et al., “Tensorflow: Large-scale machine learning on heterogeneous distributed systems,” arXiv:1603.04467, 2016.