April 2017

Convolutional Neural Network Visualization for fMRI Brain Disease Classification Tasks

Ezra Philip Davis
Worcester Polytechnic Institute

Miyabi Ida Gaskell
Worcester Polytechnic Institute

Ryan Baylor Killea
Worcester Polytechnic Institute

Convolutional Neural Network Visualization for fMRI Brain Disease Classification Tasks

A Major Qualifying Project
Submitted to the Faculty
Of the
Worcester Polytechnic Institute
In partial fulfillment of the requirements for the
Degree in Bachelor of Science in Computer Science
By

Ezra Philip Darius Davis

Miyabi Ida Gaskell

Ryan Baylor Killea

Date: April 22, 2017

Approved By:

Xiangnan Kong

Abstract

Neurologists face an increasingly overwhelming amount of data that they must use to determine diagnoses for patients with potential brain diseases. Our project aims to supplement the upcoming technology of automated brain disease classification using deep convolutional neural networks to develop a suite of visualization tools for these high dimensional deep convolutional neural networks. For this project we developed high quality visualizations for convolutional neural networks that classify fMRI brain scans. Visualizations such as those developed in this project could be used by doctors to identify which parts of the brain are indicators for mental conditions or by data scientists to understand more about how their networks work.

Acknowledgements

We would like to thank our advisor, Professor Xiangnan Kong, for his guidance and advice on this project. We would also like to thank Constance Moore of the University of Massachusetts Medical Center for providing us with the sample patient fMRI scans used in this project.

Introduction

The effectiveness of neural network classifiers is well known (Zhang et al., 2000), and in the field of fMRI brain disease classification there are several such classifiers that are convolutional neural networks (Lee et al., 2017). However, these classifiers can be difficult to understand (Zeiler and Ferguson, 2013) and, in the case of 4D fMRI images, present an extra challenge because visualizing high dimensional data is so complicated. The neural network techniques for fMRI images also differ from standard neural networks, presenting further difficulty. Such visualization methods are important as they are closely connected with understanding modern ideas in deep learning (Zeiler and Ferguson, 2013). In this MQP, we created powerful convolutional neural network classifiers to assist in automatic labeling of brain diseases and developed visualizations to understand these classifiers. Deep learning is currently used in many applications such as voice and image recognition (Schmidhuber, 2015) and the results of this MQP support novel work in visualizing convolutional networks, deep auto-encoders, and convolutional network design.

Background

fMRI

Magnetic resonance imaging (MRI) scans are images formed using magnetic fields, typically images of the human brain. Originally introduced in the 1980s, MRI scans are used to produce structural images of organs, such as the central nervous system (Logothetis, 2008). Functional MRI (fMRI) scans, however, record information from each voxel across time, as introduced in a paper for *Science* in 1991 (Belliveau, 1991). fMRI scans take separate images of the brain in both the resting and active states. The fMRI then observes whether there is any change in hemoglobin by detecting the presence or absence of oxygen. The result is a series of images showing a map of brain activity. fMRI scans are currently used to detect dynamic patterns of activity in the working human brain (Buxton, 2013). Thus, fMRI scans can provide crucial data on the higher-level visual cortex using a nonintrusive window into brain function with whole-brain coverage and reasonable spatial recognition (Cox and Savoy, 2003).

Cognitive research has used fMRI scans to map brain activity to further investigate neural diseases, including Parkinson's, Alzheimer's, and Huntington's. One study, for example, used fMRI to observe what brain circuits were associated with timing reproduction in patients with Parkinson's (Elsinger et al., 2003). Another study investigating the neural networks of Alzheimer's disease using fMRI identified a deficit of visuospatial abilities in patients, which is a sign of early onset Alzheimer's (Vanini et al., 2008). A third study found that fMRI is sensitive to neural dysfunction occurring more than 12 years prior to the onset of Huntington's disease (Zimbelman et al., 2007). As such, there is enormous capacity for fMRI scans to enable further progress in understanding the associations between different areas of the brain for these and many other neural diseases. The Huntington's study also reflects the potential uses of fMRI for predicting information about a disease in a neural network using machine learning, a point we investigated in this project.

Machine Learning and Neural Networks

Data mining and machine learning are used to determine patterns in large amounts of data, like fMRI scans. Discovering these patterns allows us to predict conditions from the data. For fMRI data, we would be able to create tentative diagnoses of various diseases. There are many different approaches to machine learning, including decision tree learning, artificial neural networks, support vector machines, Bayesian networks, reinforcement learning, and genetic algorithms. Each model has its own benefits and drawbacks and it is always a good idea to carefully pick which machine learning method(s) to use (Raschka, 2015).

Neural networks are a programming construct originally inspired by biological neurons and the brain's biological process. In a neural network there are usually many neurons, each of which has a number of inputs and one output. The neuron computes a linear combination of its inputs and applies a nonlinear transform. Often a neuron is used to determine the likelihood for whether a particular input is a member of a particular class of inputs, e.g. given an input of flower petal length and sepal length, a properly trained neuron could say whether the flower is likely to belong to a certain species or not. The weights in the neuron's linear combination are carefully selected to make the output of the neuron tell something useful about the input data. Weight selection is usually done via a process called "training", where the neuron is tested against data where the desired output is known (Raschka, 2015; Goodfellow, 2016).

Neural networks used on classification tasks, by virtue of the problem definition, admit tractable loss functions to express the difference between their prediction and the known classification for items in the training set. A popular choice among these loss functions is Categorical Cross-entropy.The way that neural networks are typically trained is through a method called Stochastic Gradient Descent (SGD) with Nesterov Momentum, RMSProp, and some form of variance reduction. SGD takes "mini-batches" of the training set, which are random subsets of a certain size, and computes the partial derivative of the parameters of the network with respect to the loss of the model on the inputs from the current mini-batch (Bottou, 2010). This partial derivative is used to move the parameters down a sloping gradient function, away from the loss (see L1 and L2 from figure 1).

Nesterov Momentum has been shown to dramatically accelerate SGD by incorporating a second order "momentum" term, which intuitively moves the search away from plateaus or flat regions of the loss function and toward deeper minima as the search accrues momentum that requires hitting opposing directions of loss in order to be reduced. Nesterov adds another parameter to be tuned and tends to amplify the variance of the parameters across updates, as well as reduce the chance of convergence in practice. The best guarantee for convergence of Nesterov is $O(1/T^2)$ (Sutskever et al., 2013). Another technique to speed up training is RMSProp, which operates in a similar way to Nesterov. However, instead of tracking a second order term, RMSProp normalizes the updates with respect to the magnitude of a running windowed average of gradients. In this way, it escapes areas with relatively small gradients and ideally converges to the solution at a fixed rate (Hinton, et al. 2014).

Variance reduction is a technique by which SGD can use more aggressive parameters for Nesterov and RMSProp while still having a chance of convergence. It relies on the fact that updates deviate wildly on a per-mini-batch basis, but if an average piecewise-linear model is kept, they can be made to adjust a “running average” model that is almost forced to converge. The last modification used in this project, and that most modern neural networks that are trained employ some variation of, was learning rate annealing. The key point of this method is that the gradient should be scaled to a finer granularity as the model gets closer to convergence to avoid missing the minima. This method, combined with more advanced third-order “negative momentum” methods, have recently been introduced that have better provable convergence guarantees (Allen-Zhu, 2016), but are not yet implemented in Keras and were therefore unavailable to use for our visualizations.

Shallow neural networks are good tools for determining simple classifications of data. Complex tasks such as image recognition, however, greatly benefit from increased depth. For more difficult tasks, multiple shallow networks are used, with each neural network - or “layer” - feeding into the next and the last neural network providing the output classification (Raschka, 2015; Goodfellow, 2016). Usually, between each layer, all values are passed through a nonlinear activation function, which prevents the more traditional linear layers from combining into a single collapsible linear function; this process is called deep learning. Activation functions include nonlinearities between layers which are usually either a sigmoid (a rescaled hyperbolic tangent function), or a rectified linear unit (ReLU, or `max(0, x)`). While the sigmoid activation function is more biologically motivated, ReLU trains faster and is generally superior for training multi-layer networks (Glorot et al., 2011).

For some problems, using traditional fully connected neurons - unconstrained matrices - for all layers of the deep learning network is inefficient. For example, for a neural network that classifies images of dogs, a dog in a slightly different location or orientation would cause a whole separate set of neurons to fire. This leads to high memory costs, a need for larger datasets, and long training times. One solution to this problem is the use of Convolutional Neural Networks (CNNs), which are a type of neural network usually used in deep learning and uses a specialized linear operation called convolution in place of general matrix multiplication in one or more layers (Goodfellow et al., 2016). Convolutional Layers, the primary component of CNNs, use a set of “filters” that are convolved over the input to the layer, and each filter produces an output, called an activation map, of almost the same size and shape as the input (LeCun and Bengio, 1995). All of the activation maps are then passed into the next layer.

Because convolutional layers’ filters focus on low level patterns and classification problems often depend on high level patterns (e.g. classifying dog and cat pictures), it is necessary to make activation maps significantly smaller. This is achieved using subsampling, which are also called pooling layers. Typically, pooling layers combine local values into a single value by a simple mathematical formula. For instance, a 3x3 max pooling layer - the most common type of pooling in CNNs - combines squares of 9 pixels into a single pixel by picking the maximum (Cohen and Shashua, 2016).

In addition to the basic building blocks of CNNs, there are a number of techniques that make neural networks more reliable and train faster that can be applied to most networks. One of these techniques is to apply Dropout. Dropout is a process that is only applied during training, where the output from individual neurons (or locations in activation maps in convolutional neural networks) is ignored randomly. This forces the neural network to become more robust and redundant, providing better predictions (Srivastava et al., 2014).

Another technique is using a Residual Network. In a Residual Network, the output from a convolutional layer is a combination of the activation maps of its filters and the inputs from the previous layer, which provides continuity over the filters across multiple layers. During training, individual layers can be left out much in the same manner as Dropout, and allows the training of significantly deeper neural networks (He et al., 2016). While using Residual Network and Dropout improves performance and training time, a neural network can also have its training time reduced by apply Batch Normalization, even somewhat obviating the need for Dropout. Batch Normalization is the name for a process where the activation maps from one layer to the next are normalized to avoid extreme values in activation maps, or an Internal Covariate Shift, and can speed up the training process by a factor of 14 (Sioffe and Szegedy, 2015).

The capacity of a Convolutional Neural Network can be controlled by varying the number of convolutional layers - called depth - and the number and size of the filters - called breadth - to make strong and mostly correct assumptions about the nature of images, including stationarity of statistics and locality of pixel dependencies (Krizhevsky et al., 2012).

Figure 1: Math Table

Layer or Concept	Math
1D Convolution	$(\vec{f} * \vec{g})(t) = \sum_{x=-w}^{w} \vec{f}_x \cdot \vec{g}_{t-x}$
3D Convolution	$(\mathbf{F} * \mathbf{G})(\vec{t}) = \sum_x \sum_y \sum_z \mathbf{F}_{x,y,z} \cdot \mathbf{G}_{\vec{t}_x - x, \vec{t}_y - y, \vec{t}_z - z}$
Sigmoid	$S(x) = \frac{1}{1 + e^{-x}}$
ReLU	$f(x) = \max(x, 0)$
Max Pool	$f(\vec{t}, \mathbf{X}) = \max_{\substack{\vec{t}_x - w \le i \le \vec{t}_x + w \\ \vec{t}_y - w \le j \le \vec{t}_y + w \\ \vec{t}_z - w \le k \le \vec{t}_z + w}} \mathbf{X}_{i,j,k}$
Average Pool	$f(\vec{t}, \mathbf{X}) = \sum_{i=\vec{t}_x - w}^{\vec{t}_x + w} \sum_{j=\vec{t}_y - w}^{\vec{t}_y + w} \sum_{k=\vec{t}_z - w}^{\vec{t}_z + w} \frac{\mathbf{X}_{i,j,k}}{8w^3}$
Dense Layer	$f(\vec{x}) = \text{ReLU}(\mathbf{W}\vec{x} + \vec{b})$
L2	$\|x\|_2 = \left(\sum_i (x_i^2) \right)^{1/2}$
L1	$\|x\|_1 = \sum_i (\lvert x_i \rvert)$

Machine Learning Using Brain Scans

fMRI scans are useful for diagnosing neurological disorders. For instance, one study conducted to discern patients with unipolar or bipolar depression was able to identify traits common to each type of disease, respectively, but determined that larger sample sizes are necessary for greater accuracy (Grotegard et al., 2013). Unfortunately, fMRIs are complex enough that it is easy to misdiagnose patients (Sing and Rajput, 2006). One possible solution to this problem is to automate the process with machine learning.

Various types of machine learning have been used to diagnose brain diseases. For instance, support vector machines and random forests have been used to diagnose Alzheimer's disease and Schizophrenia with over 96% and 80% accuracy respectively (Sarraf and Tofighi, 2016; Raventós and Zaidi, 2015), and convolutional neural networks have been used on ADHD data (Kong, 2014). In a standard model process, the machine learning algorithm takes in input in the form of an image or set of images - such as a batch of fMRI scans - and commences training the algorithm on the input data, making predictions for the diagnoses that hopefully get closer and closer to the actual values.

Convolutional Neural Network Visualization Techniques

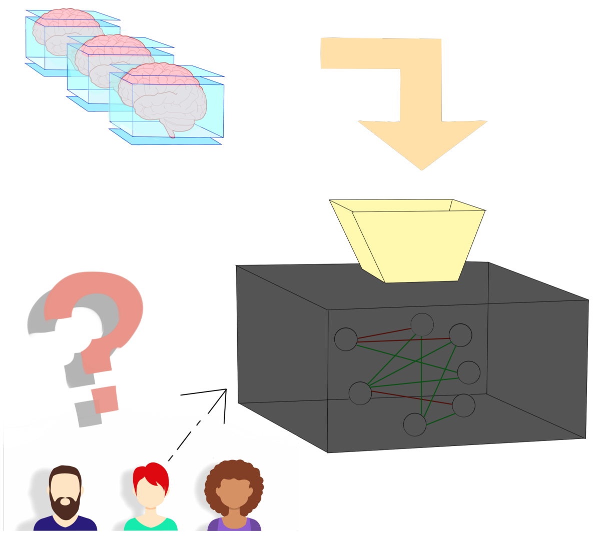


Figure 2: The Black Box Problem

Neural networks are useful tools, but it is not obvious how any given neural network operates. Unfortunately, due to the large number of non-linear interacting parts it is difficult to determine how the internals of the network work, especially for filters in the deeper layers of the network. A common comparison often made is to a "black box" (pictured in figure 2) that obscures the layers of the training set from outside sources (Yosinski et al., 2015). Since neurologists observing associations between data points within a neural network need to have access to the training filters in order to better understand the associations in an enormous dataset, any techniques used in this project needed to also address this problem.

Visualizations of neural networks can show information to serve several different purposes: showing the layout of a neural network (Lee et al., 2017), to deconstruct how a given neural network operates, to show where improvements could be made to the network (Zeiler and Fergus, 2013), or as an educational tool for explaining how a neural network works (Smilcov and Carter, 2017). This paper focuses on visualizations unique to convolutional neural networks, especially those that are affected by increasing the dimensionality of the input data from 2D images to 3D tensors.

Max-Patch

One of the most basic approaches for understanding what a filter of a convolutional neural network detects is determining the patches of the input image that cause maximal activations. It is so basic that in some papers the technique is not even named (Zeiler and Fergus, 2013). It is relatively easy to understand for 2D images: If the top 10 patches of pictures that cause a filter to maximally activate all show a certain feature (e.g. eyes in a dataset of facial recognition images), then that filter is good at detecting that feature. There are possible issues with this kind of visualization - sometimes it is not clear what the patches have in common, or the filter may detect more than one feature from the original images (e.g. the filter activates for both eyes and pictures of doorknobs). It is also necessary to have an appropriate dataset on hand to pull the patches from. Despite this, Max-Patch visualizations are one of the most intuitive ways of detecting the purpose of a filter on a convolutional layer. Figure 3 and figure 4 are examples of a Max-Patch visualization on a convolutional neural network that classifies images of hand-drawn digits from LeCun's MNIST dataset. This visualization shows that filter 31 is consistently detecting a different shape than filter 21.

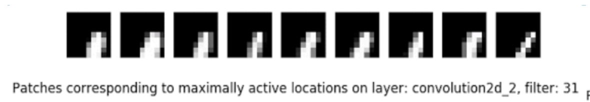
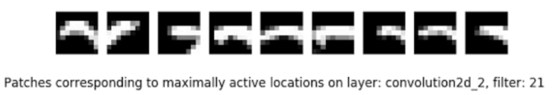

Patches corresponding to maximally active locations on layer: convolution2d_2, filter: 31

Patches corresponding to maximally active locations on layer: convolution2d_2, filter: 21

Figure 3 and Figure 4: MNIST Max-Patch Examples

There is also spatial data shown by the location of the maximum activations. This can be useful for determining where the filter activates, and get a more general idea of what the filter is trying to identify. Figure 5 uses the same classifier as the examples above, but this example also shows the location of the maximum activation on the original image. It is clear that filter 0 generally activates towards the top of the MNIST images' digits.

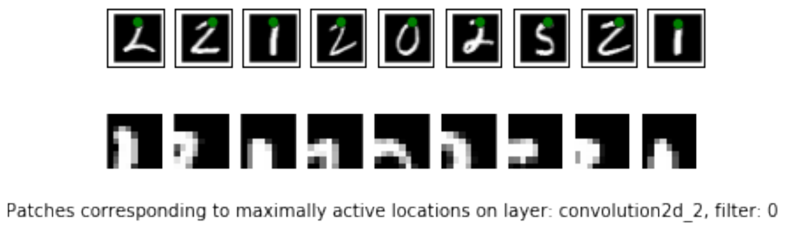

Patches corresponding to maximally active locations on layer: convolution2d_2, filter: 0

Figure 5: Max-Patch Example with Original Location Marked

Live Activations

Perhaps the next most straightforward method of visualization is showing the activation maps of a filter on an image. This allows a quick visualization of what each filter detects, and is fast enough to run on live video (Yosinki et al., 2015). Each activation map is always equal to or smaller than the original image. In figure 6, the leftmost image is the original image from the MNIST dataset, and the right three images are activation maps for different filters. This visualization shows where the filter activates - for instance, the filter corresponding to the rightmost image detects (activates on) the bottom of lines on the original image.

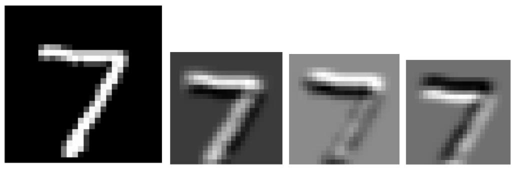


Figure 6: Live Activation Example

Guided Back Propagation and Saliency Maps

Guided Back Propagation is a technique that aims to visualize what parts of the inputs for a neural network are used to determine its output. This technique can be easily extended to account for an individual feature instead of the output of the network by treating the loss that is back-propagated to the output of the layer containing the feature to be the loss output of the classifier and applying the same techniques.

Saliency Maps use guided back-propagation with a special way of back-propagating through ReLU layers to determine a feature on an image. The idea is to highlight the areas of the image that influence the activation of that feature significantly. When using guided back-propagation, one discards all negative gradients each time they pass through a ReLU layer, which sparsifies and concentrates the end result on areas that correlate positively with the visualized feature (Simonyan et al., 2013). We also propose adding the gradient map to a map with only negative gradients if the number of ReLU layers considered is even, and the input ReLU positive if the number of ReLU layers considered is odd. Because we constrain the sign and two negative signs become positive, both of these together give a sparse footprint of how that feature activates on the image.

Saliency Maps encode the location of an object of a given class in an image and are computed by finding a derivative using a single back-propagation pass, and then rearranging elements in the vector w, where the number of elements is equal to the number of pixels in the given image. The Saliency Maps are extracted using a Convolutional Neural Network classification trained on the image labels and are then visualized (Simonyan et al., 2013). This visualized map can then be used in deconvolution.

Deconvolution

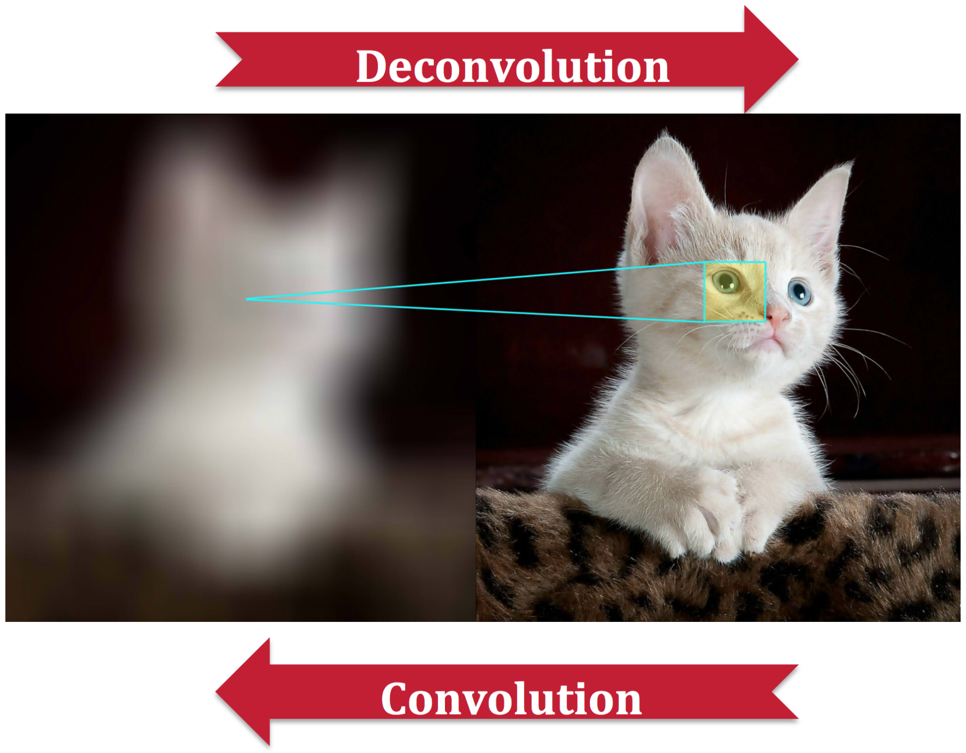


Figure 7: Basic Example of Convolution and Deconvolution

A Deconvolutional Network is a Convolutional Model that uses the same components (filtering, pooling) in reverse (Zeiler and Fergus, 2013). Each layer successively unpools, rectifies, and filters the activity in the layer beneath it until it reaches input pixel space. Figure 7 is a simple visual aid demonstrating the inverse relationship between convolution and deconvolution. Unpooling is an approximate inverse of max pooling that takes place in the convolutional equivalent, which then switches from the convolutional network and places reconstructions on upper layers in order to preserve the structure (Springenberg et al. 2015). The reconstructed signal is then passed through a ReLU non-linearity before each filter is transposed vertically and horizontally.

Training a Deconvolutional Neural Network on the output classifications of a neural network allows patterns to be drawn from what the neural network thinks a typical member of that classification looks like. It is also possible to train Deconvolutional Neural Networks to, using a feature map produced by a convolutional layer, attempt to find what the original image looked like. A network trained in this way can be fed a one-hot vector to attempt to find a "perfect" image for activating a single feature at that location.

Auto-Encoder

An auto-encoder, or auto-association, neural network learns a map from the high-dimensional space of fMRIs to a low dimensional space of significantly lower size and a paired map from the low dimensional space back to the input space using a fully convolutional approach with pooling. Because the size of the low dimensional space is less than that of the space of fMRIs, it is called an “undercomplete” space as it cannot represent all vectors that could exist in the fMRI space (Goodfellow et al., 2016). Auto-encoders that are completely linear have been shown to be equivalent to Principal Component Analysis (PCA) when trained using L2 loss and L2 regularization that is sufficient to keep the spectral norm of the resulting matrices below a threshold (Murray, 2016). PCA finds orthogonal vectors which define a basis such that when the input is projected to said basis, expected Euclidean distance to the original input is minimized over some data-set. In this way, deep auto-encoders can be thought of as optimizing for principle-component half-spaces which separate the data in a way that classifies it into categories for which there is a good principal component representation (Basri and Jacobs, 2017).

Auto-encoders have been shown to be useful as pre-training, or pre-conditioning, methods for deep neural networks. Before the introduction of sound random initializations (in a paper by He et al. in 2015 and another method by Glorot and Bengio in 2010), auto-encoder initializations were standard for deep neural network training. Deep auto-encoders have been shown to be equivalent to certain forms of manifold learning which aids in the intuition that their construction is used to maintain smooth areas of the input space as smooth in the feature space (Basri and Jacobs, 2017; Park 2015).

Convolutional auto-encoders consist of a CNN with a low dimensional output (say 5x5x3 instead of an input image that is 96x96x50) to perform the encoding, followed by a Deconvolutional Neural Network to decode the data back to its original dimensions (for our example, 96x96x50) (Masci, 2011).

Results and Visualizations

Setup and tools

For this project, we developed a model that we trained on a set of fMRI scans based on the patient’s diagnosis for bipolar disorder. This fell into four categories: at risk - manic, at risk - genetic, has bipolar disorder, and does not have bipolar disorder. We used the following visualization techniques: Max-Patch, Guided Back Propagation, Saliency Maps, Deconvolution, and an auto-encoder. Visualizations were developed using `Keras` backend and pre-processing was performed using a Python script. The network was trained on the WPI Ace cluster server.

Model Structure

Network Model

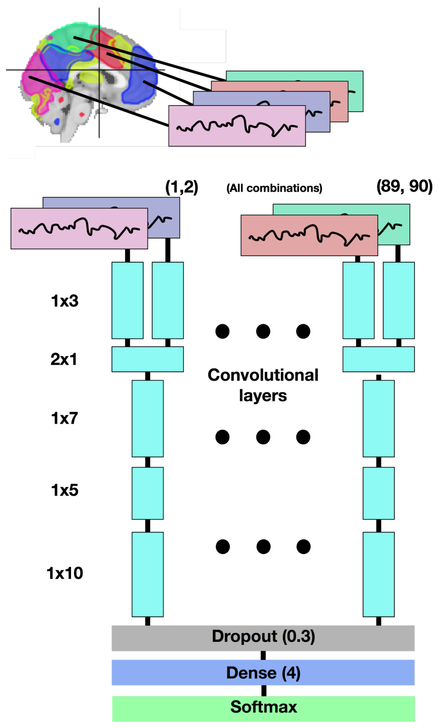


Figure 8: Network Model Architecture

One common way of analyzing brain scans is by analyzing pairs of regions of the brain and figuring out important interactions between them. This method has had success in using statistical analysis to show similarities in bipolar disorder and schizophrenia (Chai et al., 2011), among other neural disorders. We created patterned one of our model's using analysis of pairwise signals from brain scans, starting with the architecture of the best performing model from Lee's 2017 paper (also by our advisor, Xiangnan Kong). This original model began with a 2x1 "valid" layer which Through testing the model on our held-out testing set we determined that we were slightly underfitting so we added another layer of 1D convolutions before the 1x2 Convolution proposed in the model to arrive at 1x3, 2x1, 1x7, 1x5, 1x10 layers, all of which outside of 2x1 were Residual. This difference allowed our model (pictured in figure 8) to train faster and have more representational power than the model originally proposed in Lee's paper.

Fully Convolutional Model

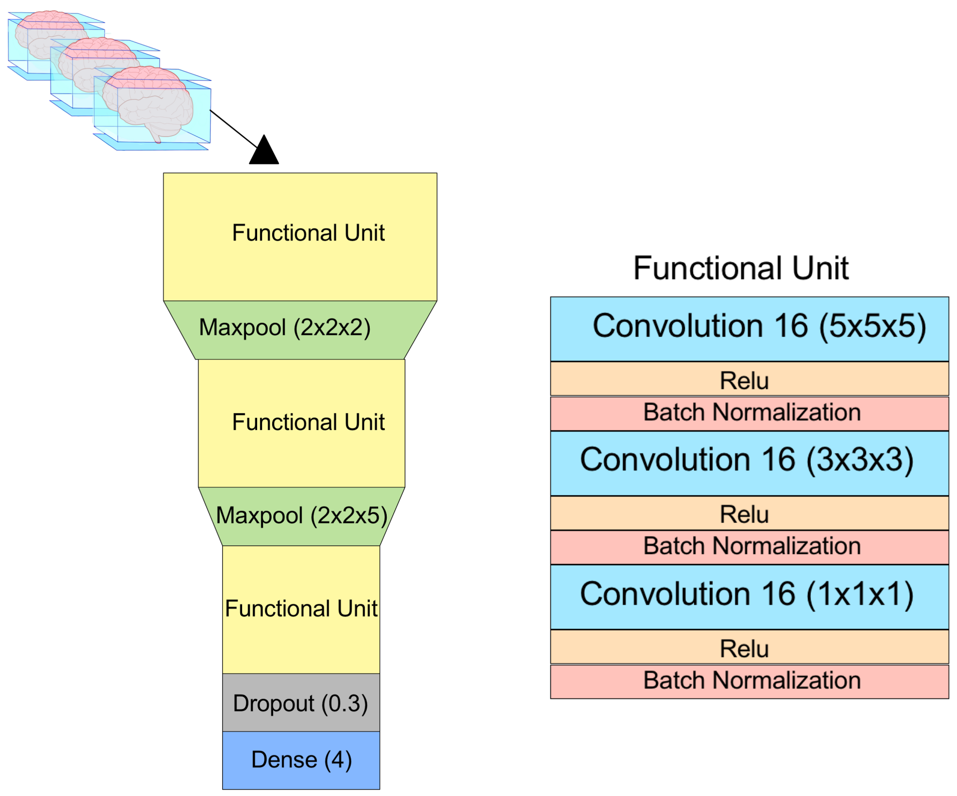


Figure 9: Fully Convolutional Model Architecture

Our other model directly convolves on the fMRI images, consisting of 9 convolutional layers. Our Convolutional layers are divided into three “Functional Units”, each containing 1 5x5x5, 1 3x3x3, and 1 1x1x1 Residual Convolutional layers, all of which have 16 filters. Each of our Functional Units is separated from the next by a Max-Pooling layer. Our rationale for this architecture comes from the understanding that larger convolutional filters can detect features that smaller ones cannot, but take significantly longer to train, and a network that mixes the two provides a good balance (Smith and Topin, 2016; Zagoruyko and Komodakis, 2017). A constraint of our model is that because we are using Residual layers all the way through, it is impossible to create a network which follows the previously standard pyramid-shape architecture for neural network design which is to have more filters closer to the input and few towards the output of the neural network. There do exist workarounds to this problem (Han et al., 2016), however this was out of scope for our implementation. This also creates a neural network that has a straightforward architecture to create visualizations from.

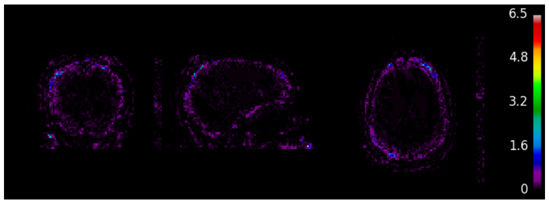
This model produces a classifier for a single brain image in a sequence of 150 brain images, which we used to “boost” the model to obtain a better result than what we achieved on the single brain image.

Model Fit/Loss

The fully convolutional model was trained to recognize patients with the 'DX' label into one of four categories, diagnosed, not-bipolar, at risk: manic and at risk: depressive. After revising the network to train by categorical loss, a lower learning rate was required.

For our models, we trained each with a holdout of 13 and 15 items for our network and our fully convolutional model respectively (we decided to use a larger holdout after worrying that our good result for the network model was due in part to variance) for our test set, leaving the rest of the data for the model to learn from. We achieved state-of-the-art (to our knowledge) results for both of our models, with an 86.7% (13/15) accuracy for our boosted fully convolutional neural network and an 84.6% (11/13) accuracy for our network model. This is a higher classification success rate than 64%, the accuracy of the model our network model is based on, though that network is diagnosing ADHD (Lee et al., 2017), which may be a significantly harder problem.

Saliency Maps



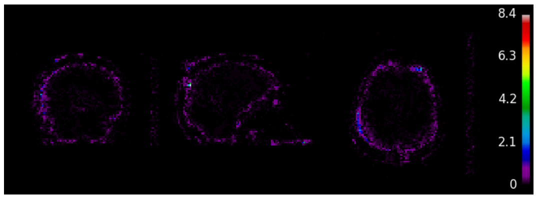


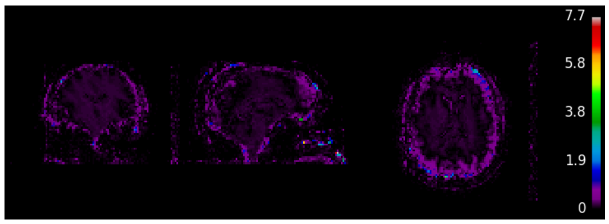
Figure 10 and figure 11: Saliency Map Examples on our Fully Convolutional Network

Our implementation of Saliency Maps, as shown in figure 10 and figure 11, (Simonyan et al., 2013) operates on a 3D CNN, and can visualize the output of any filter of any Residual layer in the model (or as initially described in the paper (Simonyan et al., 2013) on the output layer) as a video, each frame being a single saliency map of the prediction that is being used to boost the classifier accuracy over the whole fMRI sequence. To make this efficiently computable, we used `Keras` built-in `K.gradient` function on a black-box `Keras` "functional model" that we created by taking the input of the model and the output of the layer we are concerned with and pairing that with a loss that is the negative magnitude (in the Euclidean sense) of the output of the feature/layer pair in question. If we were to perform gradient descent on this it would allow us to train a deep-dream model, however such a visualization would not make sense in the already difficult-to-interpret case of brain fMRIs.

In order to modify the gradient passed through the ReLU gates to produce our Saliency Maps, our functional model must actually run through each Residual layer and then remove the results that are not the correct sign by using a second pass done using simple `numpy` operations.

Upon observing the results from our Saliency Map videos we were able to distinguish a few features that seemed omnipresent across filters, layers, and brains. The most significant of these features was a clear pattern of communication between regions of the brain and an area near the back of the top of the brain which we identified to be the parietal lobe, which has been associated with bipolar disorder (Najt, 2013).

We also created Leave One Out (LOO), pictured below in figures 12 and 13, visualizations that show a similar result to the Saliency Map visualizations. LOO visualizations are concerned with the impact of all the filters that are not visualized by the Saliency Map on how the later layers use the output of the filter visualized. The LOO visualization is performed by subtracting the normalized Saliency Map output for a single filter from the normalized Saliency Map output of all filters on a given layer and clipping the loss to be above zero. This produces the points of the image that are taken into account by the other filters to the exclusion of the filter being visualized.



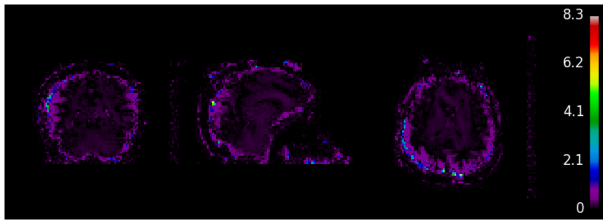


Figure 12 and 13: Examples of Leave One Out (LOO) Visualizations

Max-Patch

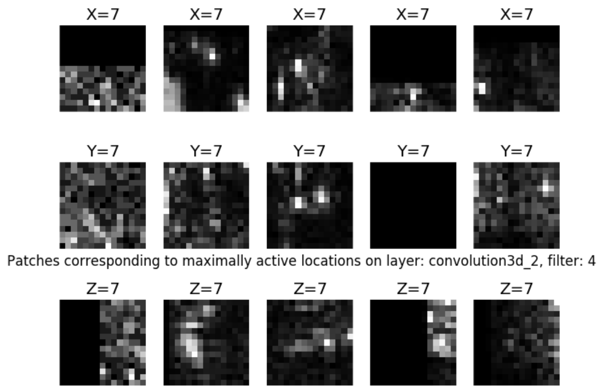
We applied several different versions of the Max-Patch visualization to our network, and our process evolved over time.

One major consideration for Max-Patch is selecting the size of the patches. Ideally, the patch includes all pixels that influence the maximum location on the activation map. We considered using back-propagation to detect what influences the pixel, but it is much simpler to perform a simple calculation (figure 14) using the hyper-parameters of each of the previous layers.

```
Data: $patch\_size = (1,1,1)$
for each $layer$ in $model$ do
    if $layer$ is $convolutional$ then
        $patch\_size = patch\_size \oplus layer.filter\_size$
    end
    if $layer$ is $pooling$ then
        $patch\_size = patch\_size \otimes layer.pooling\_size$
    end
end
```

Figure 14: Simple Patch Size Algorithm

This method does break down on more complex neural networks where layers are not linearly arranged, such as the Inception Architecture (Szegedy, 2015), but works quite well on simpler neural networks, such as our fully convolutional model.
The next difficulty we come across is how to display the patches once we gather them. Initially, we showed a single cross section, but that clearly did not show enough information to identify what features the filter detects. In figures 14 and 15, each column corresponds to a different patch and each row corresponds to a different cross section of that patch.


X=7
Y=7
Patches corresponding to maximally active locations on layer: convolution3d_2, filter: 4
Z=7

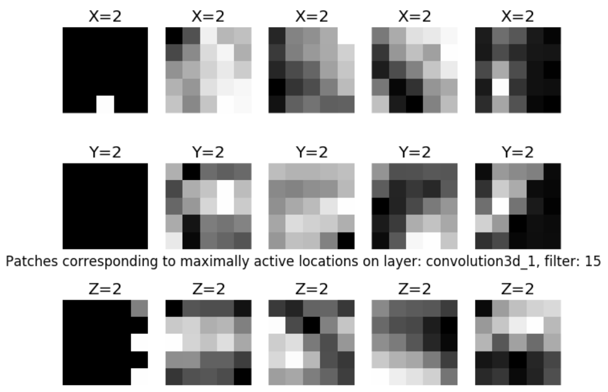

X=2
Y=2
Patches corresponding to maximally active locations on layer: convolution3d_1, filter: 15
Z=2

Figure 14 and 15: Two Max-Patch Visualizations

After that, we tried a few other methods displaying multiple cross sections, one for each axis, or one for each Z value. These methods were better, especially the latter, which shows all the data, but quickly becomes untenable as the neural network becomes deeper and the size of each patch increases. We also started to add a chart displaying the location of the patch in the original fMRI (shown in figure 17).

Figure 16 (left image): Max-Patch Visualization with Small Patches

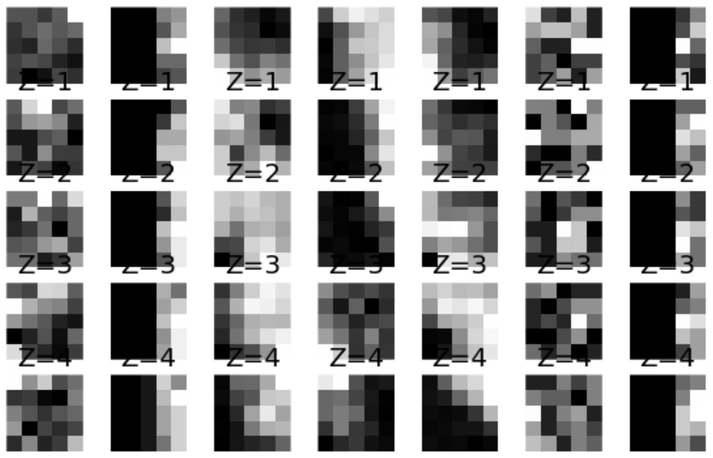
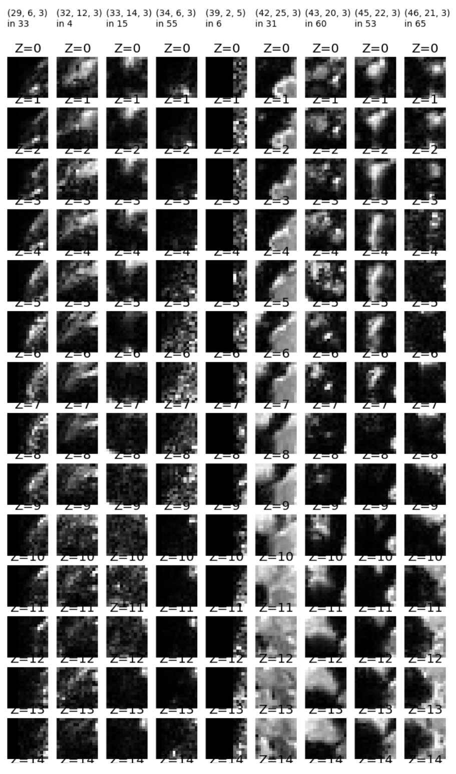


Figure 17 (right image): Later in Network Max-Patch Visualization



Finally, we settled on a 3D view placing the patch in its original location in the fMRI (figures 18 and 19). This allows us to not only get information about what each patch consists of, but also what area of the brain causes a filter to activate the most.

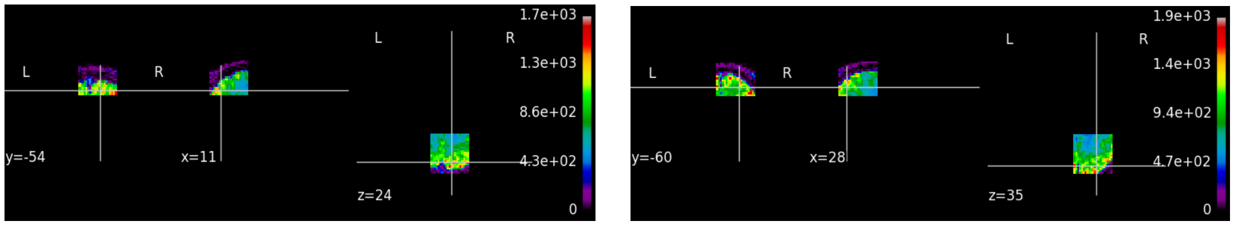


Figure 18 and 19: Two Max-Patch Visualizations with Patches in Original Position in Brain

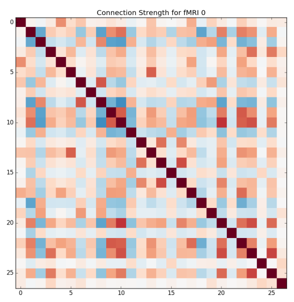
Video Rendering

We used `Matplotlib`'s (Hunter, 2007) video rendering functionality and rendered directly to files. It is important to note that this allowed us to string together plots with full resolution and not drop any frames as a result of "lossiness" that another method (i.e. using a different video rendering software with a more hands-off approach) may have caused. Because there are 88 distinct fMRI samples, we have 88 distinct videos for Saliency Maps, LOO, and our dictionary-learned activations.

Network Model Visualizations

Our network model makes its predictions using patterns between pairwise regions of the brain, and showing which pairs of regions are important for determining our prediction for a particular brain gives a lot of insight into what the neural network considers important.

We created two visualizations that show just that: a heatmap of the importance of the patterns between pairs of regions, and an activation graph network visualization superimposed on a drawing of a real brain. Both of these visualizations show the same information, but the heatmap is more complete while the graph network is easier to understand without memorizing the location of each of the regions (see figure 20 and 21).



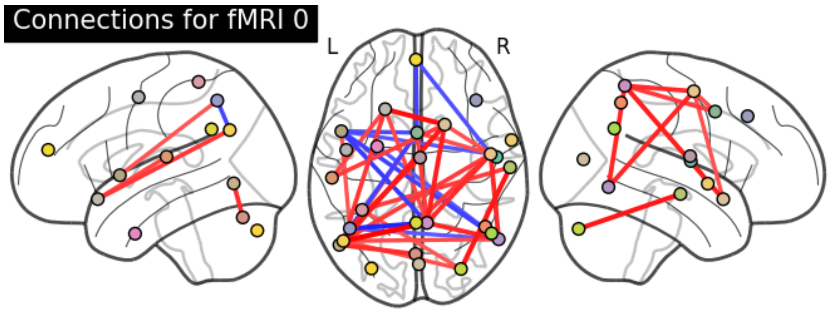
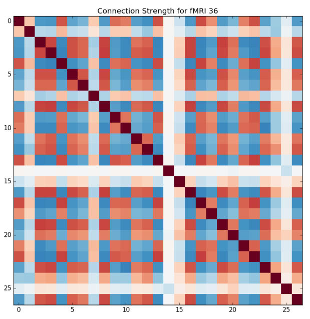


Figure 20 and 21: Visualizations of the First fMRI in the Dataset

In the heatmap, each region of the brain according to our atlas is given a row and a column, where the colored squares represent the signal between the regions indicated by its row and column. The diagonals are dark because we do not try to find patterns meaningful to our diagnosis between a region and itself. In the graph network, lines indicate a strong activation for the patterns detected between the regions at the endpoints of the edge.

In these visualizations, red indicates a positive number output from the convolutional section of the network and blue indicates a negative one, but the color is far less important than the saturation of the squares or the presence of an edge, which indicates the strength of the patterns between those two areas.

Strong vs. Weak



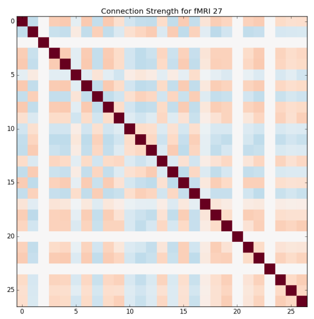
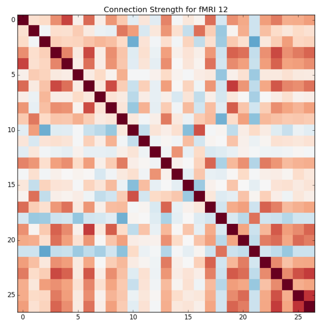
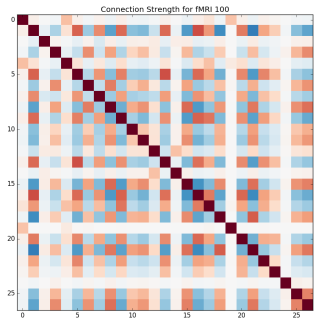


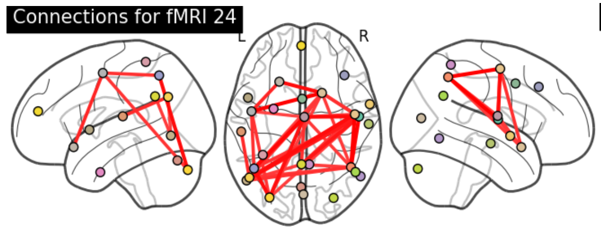
Figure 22 and 23:Strong vs. Weak

Figure 22 and Figure 23 indicate that fMRI 36 has many more of the patterns the network is looking for than fMRI 27.

Positive vs. Negative







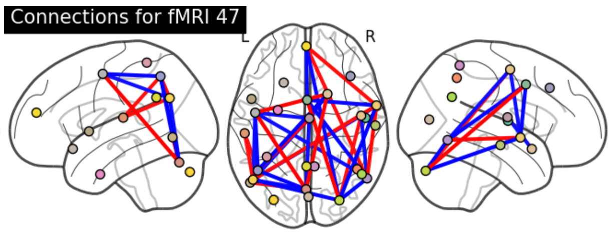


Figure 24, 25, 26, and 27: Positive vs. Negative

These four visualizations (figures 24 - 27) are examples of fMRIs that had a particularly large number of positive or negative pattern activations. Note - while this does not mean all that much because the dense layer is perfectly capable of reversing these activation. The patterns are typically strongly activated positively across our dataset or strongly negatively activated, but not both.

Negative Results

Iteratively trained model

Another model we developed was trained iteratively, one or two layers at a time. Similar methods are not uncommon, for instance there has been research into unsupervised training of individual layers as far back as 2007 (Bengio et al., 2007). This allows the training of very deep neural networks and avoids putting the entire network into memory at once. The basic idea is to train a shallow neural network (a convolutional layer followed by a dense layer to provide output) to classify the brain scans, and then take the activation maps' output from the convolutional layer, and train another convolutional layer and a dense layer to classify the activation maps from the previous convolutional layer. This is repeated until the output of the classification has stopped improving with the addition of new layers. The convolutional layers then are stacked together with the last dense layer, as the previous dense layers are discarded, to provide a classifier that works on the fMRI images. This allowed us to train each convolutional layer independently.

Unfortunately, the network we built this way was not terribly good at diagnosing the brain scans. Deep Neural Networks rely on earlier layers adapting (via training with back-propagation) to layers closer to the output, and this method contravenes that. Methods like this are usually used as pre-training - making the final training process converge much faster (Erhan, 2010). We attempted to ameliorate this issue by training two layers at a time, training each convolutional layer twice, but this did not significantly increase results. We could have followed up training each layer individually or in pairs by training the entire network at once, but by this point we had developed the other models sufficiently that this model was no longer a priority.

4D Convolutional layers

Early on we attempted to use a 4D convolutional neural network directly, however there were several obstacles that stymied our progress. One such obstacle was that in order to do this we would need to turn the built-in 3D convolution support into 4D convolution support in `Keras`, as not even the `CuDNN` toolkit on which `Keras` is based has optimized 4D convolutions. This was a huge performance hit. Additionally, in order to develop a performant and deep neural network, we were faced with a challenge of either cutting depth or filters by a factor of three as the 3x3x3x3 convolution uses 81 floats times the width of the previous layer, instead of 27 for a 3x3x3 convolution.

Auto-Encoder

We also trained a convolutional auto-encoder on our fMRI scans. The goal of training this auto-encoder was to effectively determine a piecewise-linear feature description of the input space of fMRI scans that would enable our network model to perform better on our dataset than the conventional dictionary learning approach. The performance benefit that we aimed to achieve was due to the higher performance of deep auto-encoders as dimensionality reduction techniques when used to pre-train neural networks as described in Erhan's 2009 paper and Le Paine's 2015 paper.

We trained an auto-encoder with the same general architecture to produce several different dimensions as output. The highest performing auto-encoder was naturally the 90-dimensional one, as it was the least constrained of the three, however what we had to weigh against this performance gain was how well our network model performed when it was fed pairs from the 90-dimensional input vs. the 5 or 30 dimensional inputs. The network model requires a number of weights that is quadratic in the number of 1-dimensional inputs it takes. This means that the layout of the deep network model allows for a much higher number of trainable convolutional filters per layer on the 5 model than the 90 model.

Conclusions

During the course of this project we created two very different, highly accurate predictive models using convolutional neural networks and a number of visualizations that show the function of parts of our models. We found that non-overlapping regions of the brain are more useful than our convolutional auto-encoder for classifying bipolar depressive disorder in our dataset. Additionally, we observed meaningful motifs in visualizations both networks, in particular interactions with the parietal lobe, which agrees with previous research on bipolar depressive disorder (Najt et al. 2013). Visualizations for the two models produced similar results, so we can conclude that they work consistently.

Future work

There are a number of avenues that one could take to pursue this research further: more visualizations, increasing the training set, and applying our approach to related problems. There are many visualizations that are commonly used on 2D Convolutional neural networks that we did not investigate in this paper, such as deconvolution or displaying activation maps, which may be informative and useful. Additionally, for this project we had access to fMRIs from 88 unique patients, which is a small dataset for a machine learning problem. Training with more fMRIs will show if our networks overfit the data and possibly suggest different visualizations and understanding of the problem.

Finally, we trained our network to determine whether a patient has bipolar disorder, but there are many more problems relevant to doctors and patients that remain unexplored. For instance, around 60% of patients with bipolar disorder are misdiagnosed as having major depressive disorder (Hirschfeld et al., 2003; Singh and Rajput, 2006) because differentiating the symptoms between the two disorders is difficult. As such, potential future applications of this project could be used to identify the different patterns associated with bipolar disorder and major depressive disorder using fMRI scans of patients.

References

Allen-Zhu, Z. (2016). Katyusha: The first direct acceleration of stochastic gradient methods.

Basri, R., & Jacobs, D. (2017). *Efficient Representation of Low-Dimensional Manifolds using Deep Networks*. Paper presented at the ICLR.

Belliveau, J., Kennedy Jr, D., McKinstry, R., Buchbinder, B., Weisskoff, R., Cohen, M., et al. (1991). Functional mapping of the human visual cortex by magnetic resonance imaging (Vol. 254, pp. 716-719). Science.

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2006). *Greedy layer-wise training of deep networks*. Paper presented at the Proceedings of the 19th International Conference on Neural Information Processing Systems.

Blumensath, T., Jbabdi, S., Glasser, M. F., Van Essen, D. C., Ugurbil, K., Behrens, T. E. J., et al. (2013). Spatially constrained hierarchical parcellation of the brain with resting-state fmri (Vol. 76, pp. 313 - 324). NeuroImage.

Bottou, L. (2010). *Large-scale machine learning with stochastic gradient descent*. Paper presented at the Computer Statistics.

Brosch, T., & Tam, R. (2013). Manifold learning of brain mris by deep learning (Vol. 8150, pp. 633 - 640). Medical Image Computing and Computer-Assisted Intervention–MICCAI.

Buxton, R. B. (2013). The physics of functional magnetic resonance imaging (fMRI). *Reports on Progress in Physics, 76*(9), 096601.

Chai, X. J., Whitfield-Gabrieli, S., Shinn, A. K., Gabrieli, J. D., Castanón, A. N., McCarthy, J. M., et al. (2011). Abnormal medial prefrontal cortex resting-state connectivity in bipolar disorder and schizophrenia. *Neuropsychopharmacology, 36*(10), 2009-2017.

Cohen, N., & Shashua, A. (2016). *Convolutional rectifier networks as generalized tensor decompositions.*Paper presented at the International Conference on Machine Learning (ICML).

Cox, D. D., & Savoy, R. L. (2003). Functional magnetic resonance imaging (fMRI) “brain reading”: detecting and classifying distributed patterns of fMRI activity in human visual cortex (Vol. 19, pp. 261 - 270).

Elsinger, C. L., Rao, S. M., Zimbelman, J. L., Reynolds, N. C., Blindauer, K. A., & Hoffmann, R. G. (2003). Neural basis for impaired time reproduction in Parkinson's disease: an fMRI study. *Journal of the International Neuropsychological Society, 9*(07), 1088-1098.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., & Bengio, S. (2010). Why Does Unsupervised Pre-training Help Deep Learning? *Journal of Machine Learning Research, 11*, 625-660.

Erhan, D., Manzagol, P.-A., Bengio, Y., Bengio, S., & Vincent, P. (2009). *The Difficulty of Training Deep Architectures and the Effect of Unsupervised Pre-Training.* Paper presented at the Artificial Intelligence and Statistics.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Artificial Intelligence and Statistics*.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*: MIT Press.

Grotegerd, D., Suslow, T., Bauer, J., Ohrmann, P., Arolt, V., Stuhrmann, A., et al. (2013). Discriminating unipolar and bipolar depression by means of fMRI and pattern classification: a pilot study. *European archives of psychiatry and clinical neuroscience, 263*(2), 119-131.

Han, D., Kim, J., & Kim, J. (2016). Deep Pyramidal Residual Networks. *arXive preprint*.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. Paper presented at the International Conference on Computer Vision.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep residual learning for image recognition.* Paper presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

Henaff, M., Bruna, J., & LeCun, Y. (2015). Deep convolutional networks on graph-structured data. *arXive preprint*.

Hinton, G., Srivastava, N., & Swersky, K. (2014). CSC321 Lecture 6: Neural Networks for Machine Learning. University of Toronto.

Hirschfeld, R., Lewis, L., & Vornik, L. A. (2003). Perceptions and impact of bipolar disorder: how far have we really come? Results of the national depressive and manic-depressive association 2000 survey of individuals with bipolar disorder. *Journal of Clinical Psychiatry*.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing In Science and Engineering, 9*(3), 90-95.

Ioffe, S., & Szegedy, C. (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* Paper presented at the Proceedings of the 32nd International Conference on Machine Learning (ICML-15).

Kong, X., & Yu, P. S. (2014). Brain network analysis: a data mining perspective (Vol. 15, pp. 30 - 38). Special Interest Group on Knowledge Discovery and Data Mining Explorations.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *Imagenet classification with deep convolutional neural networks*, Neural Information Processing Systems.

Le Paine, T., Khorrami, P., Han, W., & Huang, T. (2014). An Analysis of Unsupervised Pre-training in Light of Recent Advances. *arXive preprint*.

LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks, 3361*(10), 1995.

LeCun, Y., Cortes, C., & Burges, C. J. C. (1998). The MNIST Database of handwritten digits.

Lee, J. B., Kong, X., Bao, Y., & Moore, C. (2017). *Identifying Nonlinear Contrasting Networks from Time*

Series Data: Application to Brain Network Analysis. Paper presented at the SIAM International Conference on Data Mining.

Liu, X., Kong, X., & Yu, P. S. (2017). *Collective Discovery of Brain Networks*

with Unknown Groups. Paper presented at the International Joint Conference on Neural Networks.

Logothetis, N. K. (2008). What we can do and what we cannot do with fMRI. *Nature, 453*(7197), 869-878.

Masci, J., Meier, U., Cireşan, D., & Schmidhuber, J. (2011). *Stacked convolutional auto-encoders for hierarchical feature extraction*. Paper presented at the International Conference on Artificial Neural Networks.

Murray, I. (2016). Autoencoders and Principal Components Analysis (PCA). University of Edinburgh.

Najt, P., Bayer, U., & Hausmann, M. (2013). Right fronto-parietal dysfunction underlying spatial attention in bipolar disorder. *Psychiatry Research, 210*(2), 479-484.

Park, E. (2015). Manifold Learning with Variational Auto-encoder for Medical Image Analysis.

Plant, C., Teipel, S. J., Oswald, A., Böhm, C., Meindl, T., Mourao-Miranda, J., et al. (2010). Automated detection of brain atrophy patterns based on MRI for the prediction of Alzheimer's disease. *Neuroimage, 50*(1), 162-174.

Plis, S. M., Hjelm, D. R., Salakhutdinov, R., Allen, E. A., Bockholt, H. J., Long, J. D., et al. (2014). Deep learning for neuroimaging: a validation study. *8*, 229.

Raschka, S. (2015). *Python machine learning*: Packt Publishing Ltd.

Raventós, A., & Zaidi, M. (2015). Automating Neurological Disease Diagnosis Using Structural MR Brain Scan Features.

Sarraf, S., & Tofighi, G. (2016). Classification of Alzheimer's Disease Using fMRI Data and Deep Learning Convolutional Neural Networks. *arXive preprint*.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks, 61*, 85-117.

Simonyan, K., Vedaldi, A., & Zisserman, A. (2013). Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps.

Singh, T., & Rajput, M. (2006). Misdiagnosis of bipolar disorder. *Psychiatry (Edgmont), 3*(10), 57.

Smilcov, D., & Carter, S. (Producer). A Neural Network Playground. retrieved from http://playground.tensorflow.org

Smith, L. N., & Topin, N. (2016). Deep Convolutional Neural Network Design Patterns. *arXive preprint*.

Springenburg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2015). *Striving for Simplicity - The All Convolutional Net*. Paper presented at the International Conference on Learning Representations.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research, 15*(1), 1929-1958.

Sutskever, I., Martens, J., Dahl, G. E., & Hinton, G. E. (2013). On the importance of initialization and momentum in deep learning. *Proceedings on the International Conference on Machine Learning, 28*(3), 1139-1147.

Vannini, P., Lehmann, C., Dierks, T., Jann, K., Viitanen, M., Wahlund, L.-O., et al. (2008). Failure to Modulate Neural Response to Increased Task Demand in Mild Alzheimer's Disease: FMRI Study of Visuospatial Processing (Vol. 31, pp. 287 - 297). Neurobiology of Disease.

Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., & Lipson, H. (2015). *Understanding Neural Networks Through Deep Visualization*. Paper presented at the International Conference on Machine Learning.

Zagoruyko, S., & Komodakis, N. (2016). *Wide Residual Networks*. Paper presented at the British Machine Vision Conference.

Zeiler, M., & Fergus, R. (2013). *Visualizing and Understanding Convolutional Networks*. Paper presented at the European conference on computer vision.

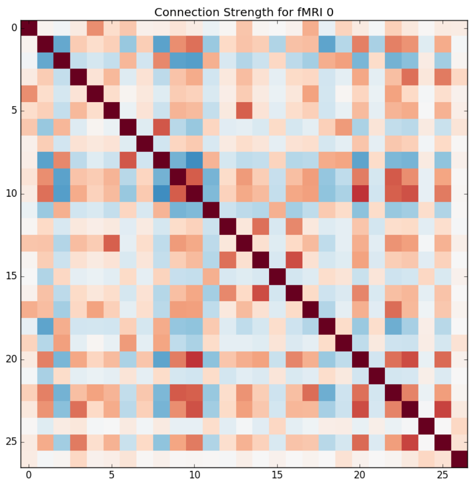
Zhang, G. P. (2000). Neural networks for classification: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, 30*(4), 451-462.

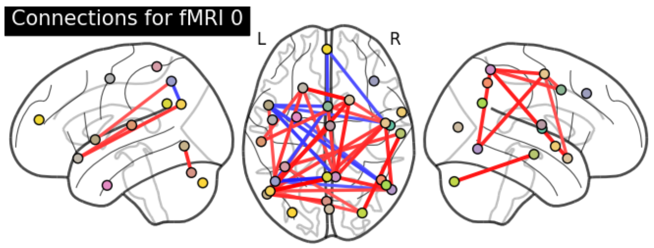
Zimbelman, J. L., Paulsen, J. S., Mikos, A., Reynolds, N. C., Hoffmann, R. G., & Rao, S. M. (2007). fMRI Detection of Early Neural Dysfunction in Preclinical Huntington's Disease (Vol. 13, pp. 758-769). Journal of the International Neuropsychological Society.

Appendix

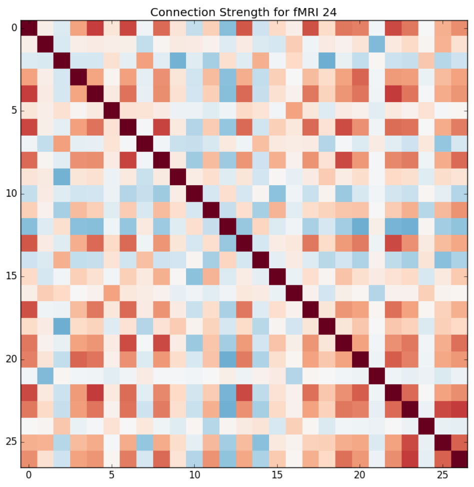
Selected Network Model Visualizations

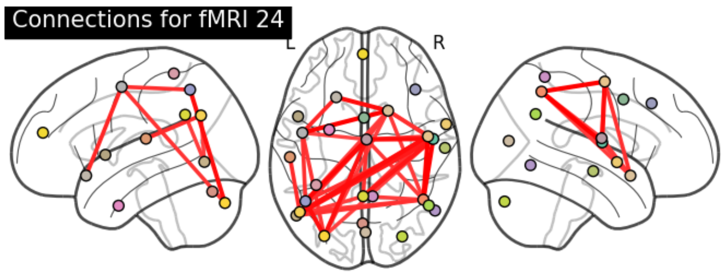
fMRI 0



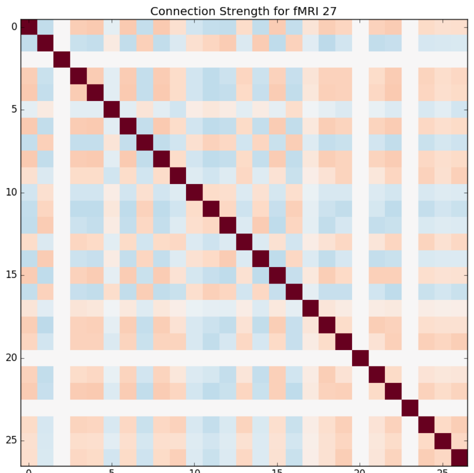


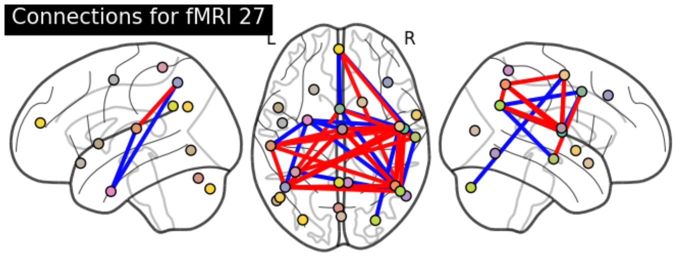
fMRI 24



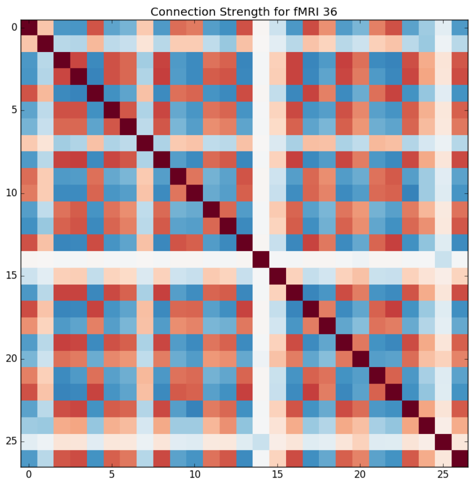


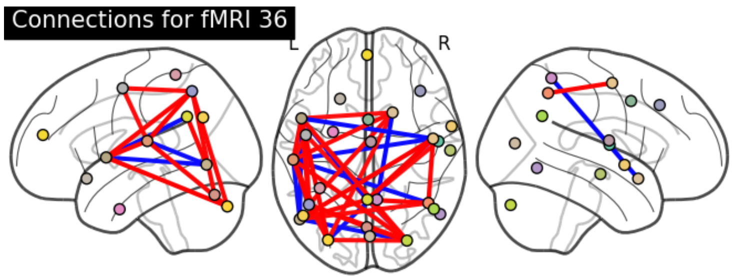
fMRI 27



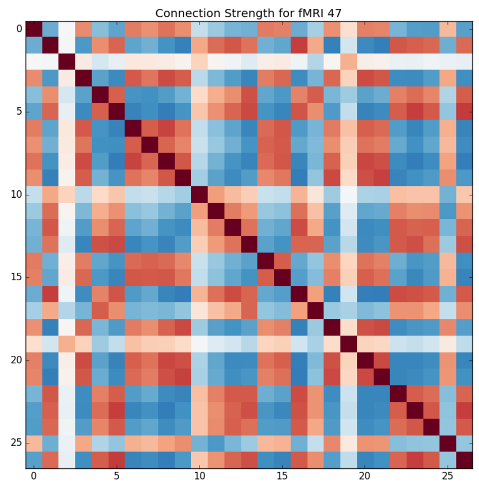


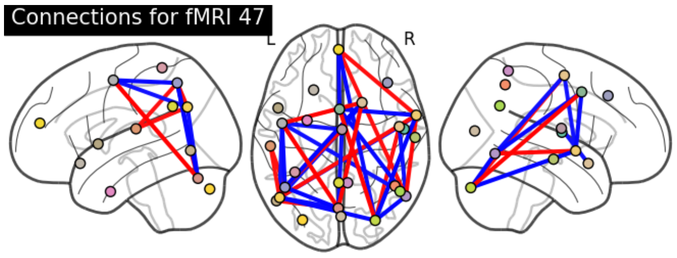
fMRI 36





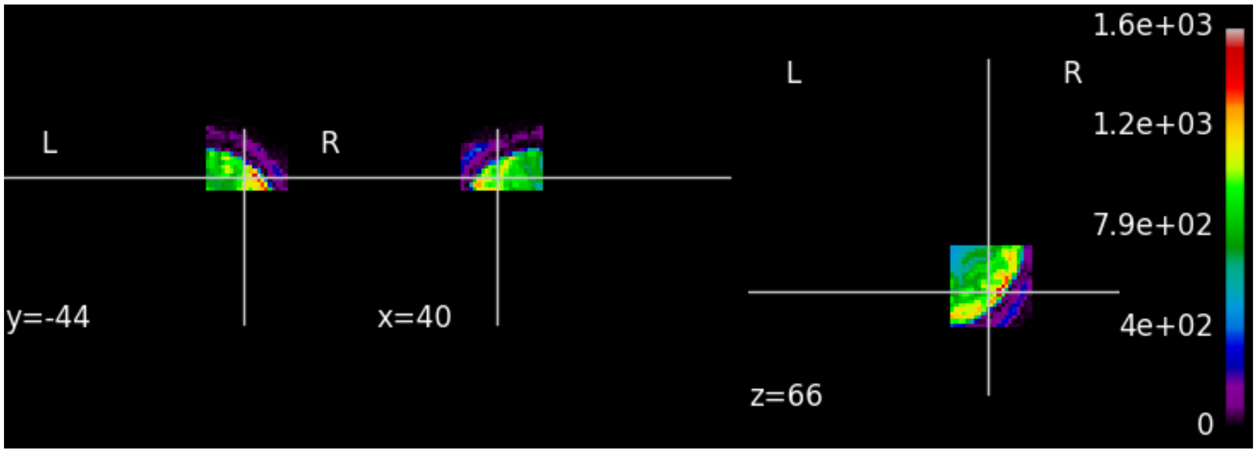
fMRI 47

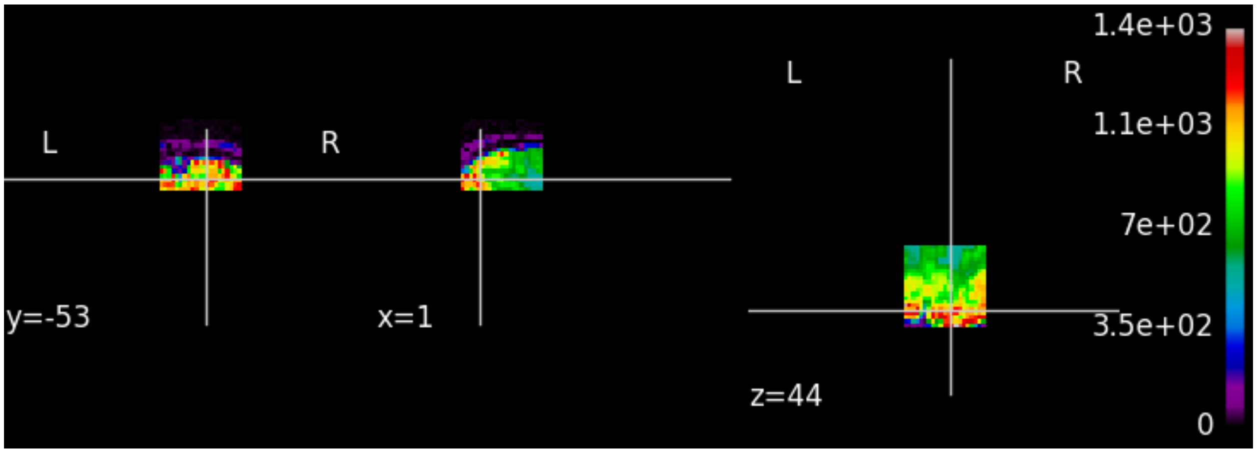

Connection Strength for fMRI 47
0
5
10
15
20
25
0
5
10
15
20
25

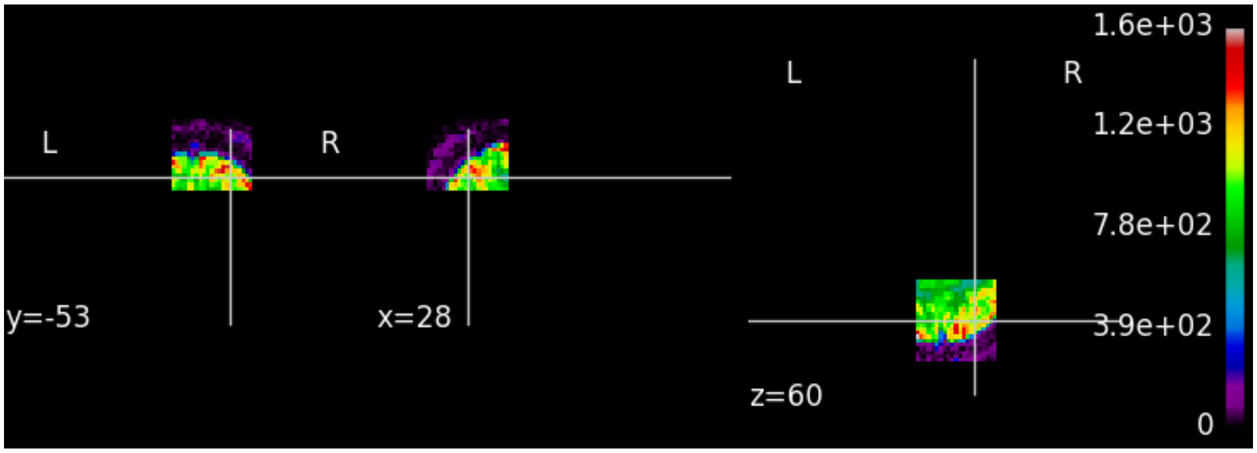

Connections for fMRI 47
L
R

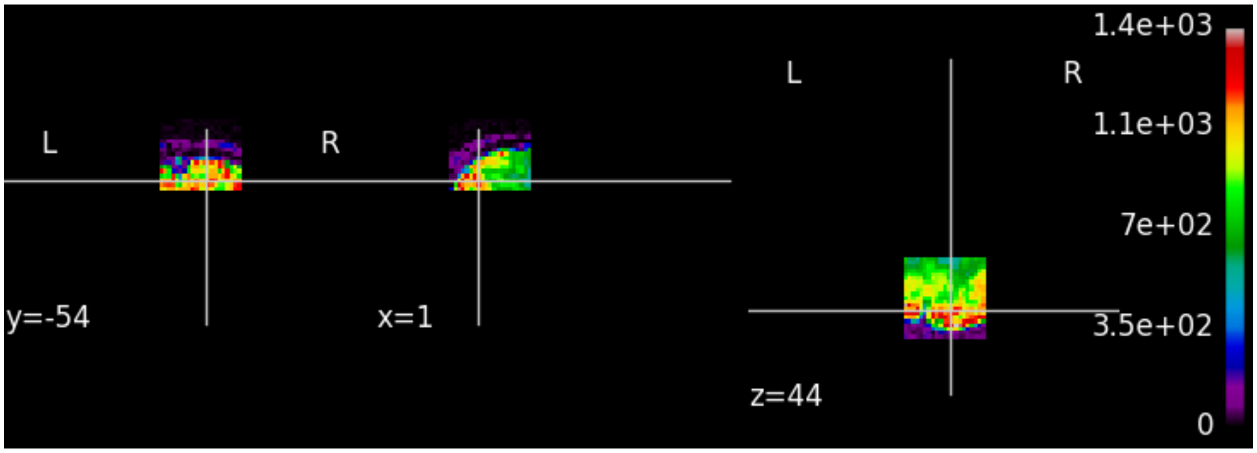
Selected Fully Convolution Model Visualizations:
Max-Patch visualizations of Filter 1 and Filter 2 from the last Convolutional layer in the 2nd Functional Unit

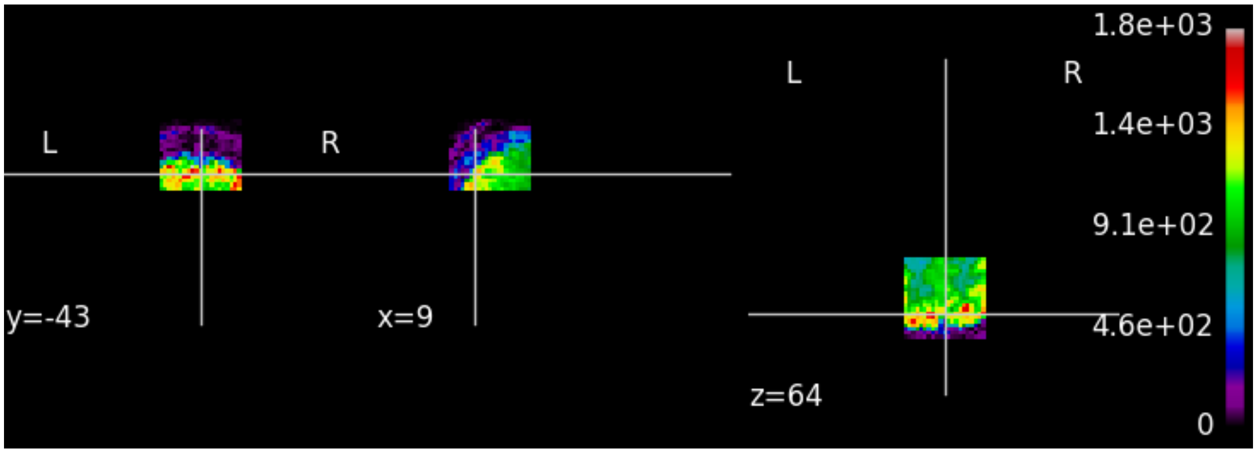
Filter 1's patches:

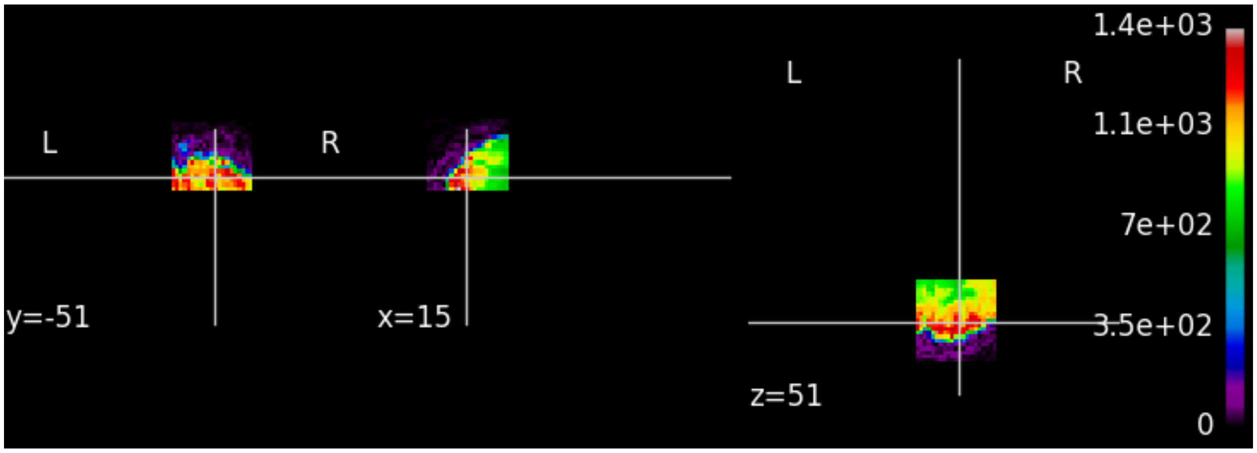


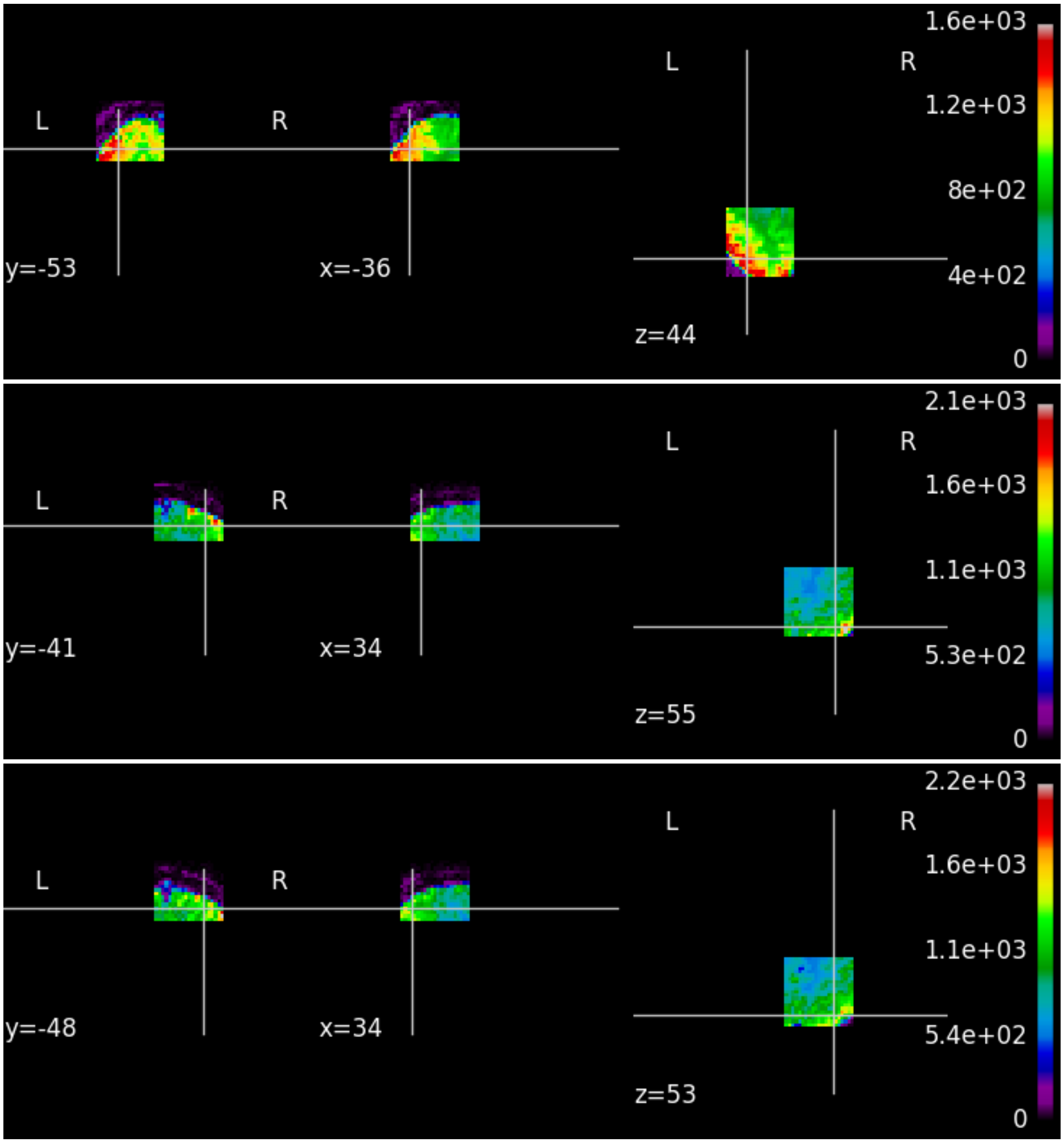


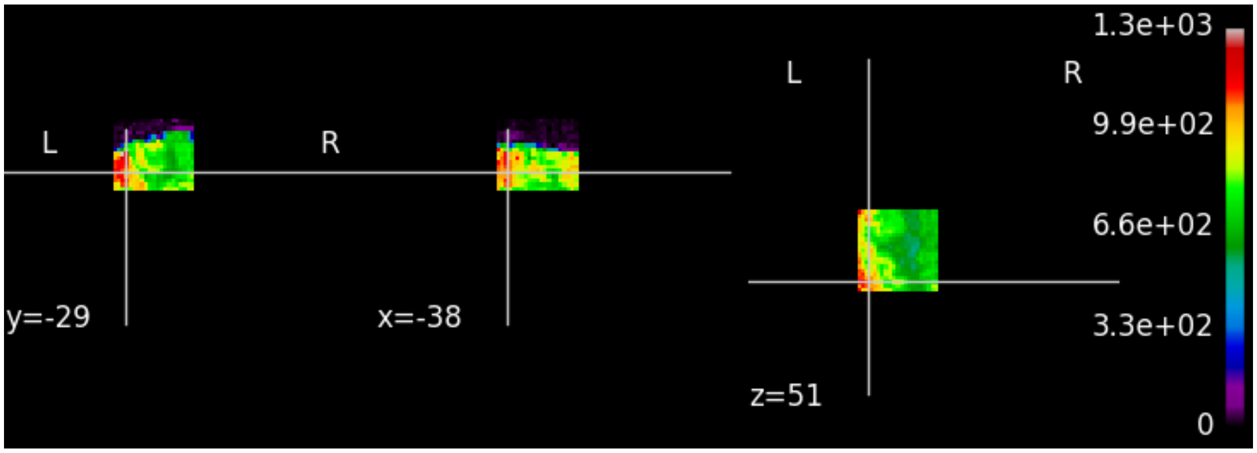



L
R
y=-54
x=1
L
R
z=44
1.4e+03
1.1e+03
7e+02
3.5e+02
0

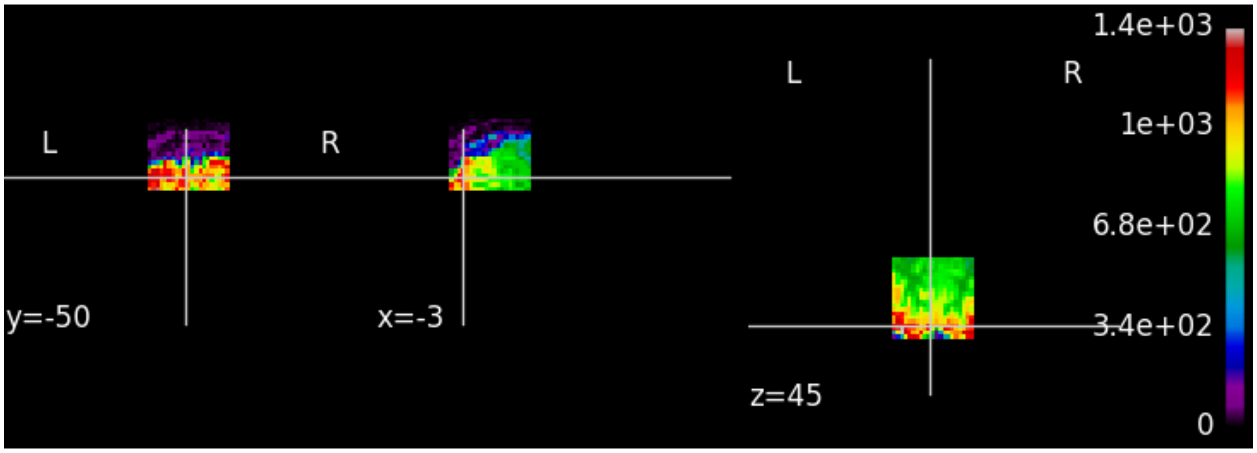

L
R
y=-43
x=9
L
R
z=64
1.8e+03
1.4e+03
9.1e+02
4.6e+02
0

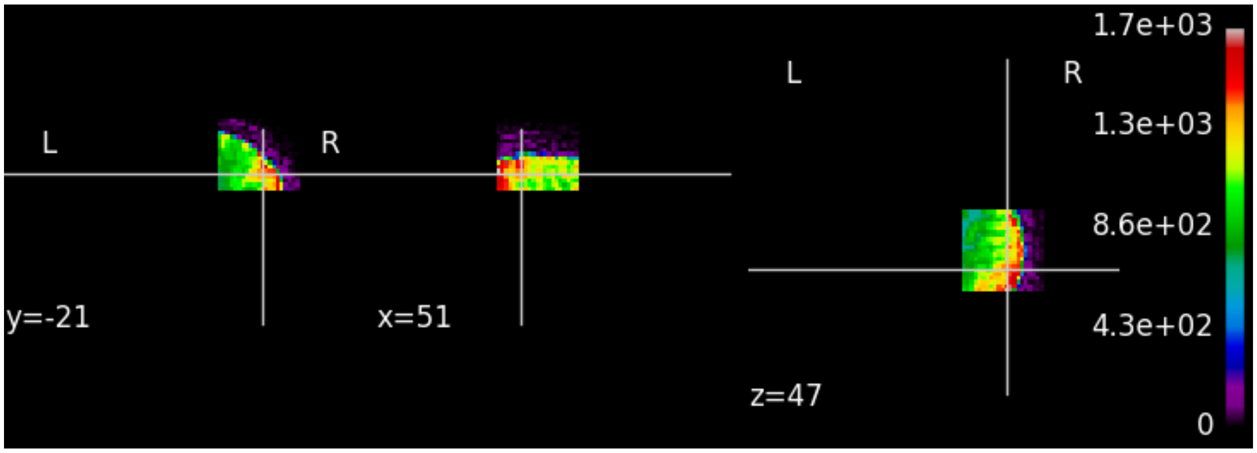

L
R
y=-51
x=15
L
R
z=51
1.4e+03
1.1e+03
7e+02
3.5e+02
0

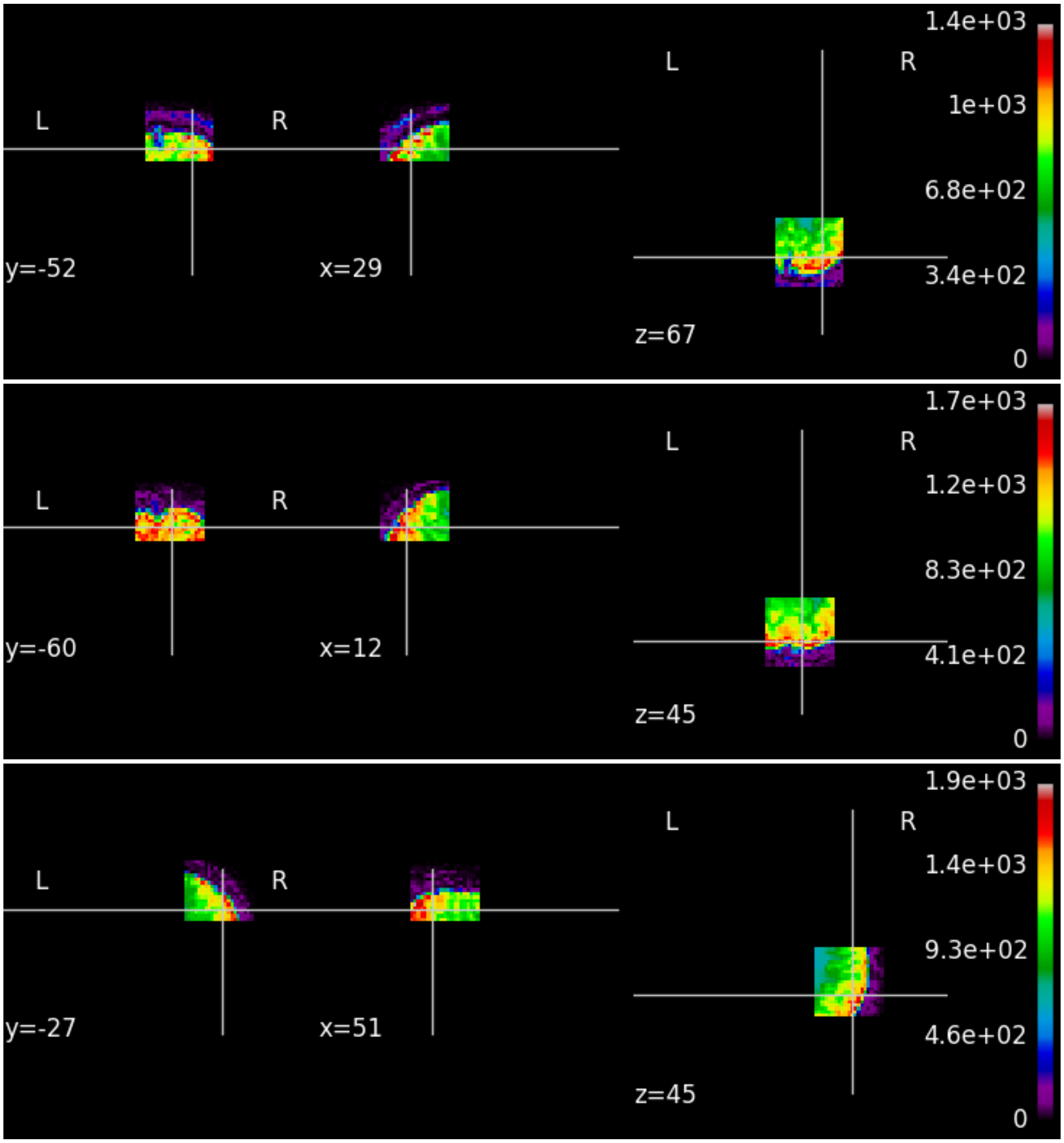

L
R
L
R
y=-53
x=-36
z=44
1.6e+03
1.2e+03
8e+02
4e+02
0
L
R
L
R
y=-41
x=34
z=55
2.1e+03
1.6e+03
1.1e+03
5.3e+02
0
L
R
L
R
y=-48
x=34
z=53
2.2e+03
1.6e+03
1.1e+03
5.4e+02
0

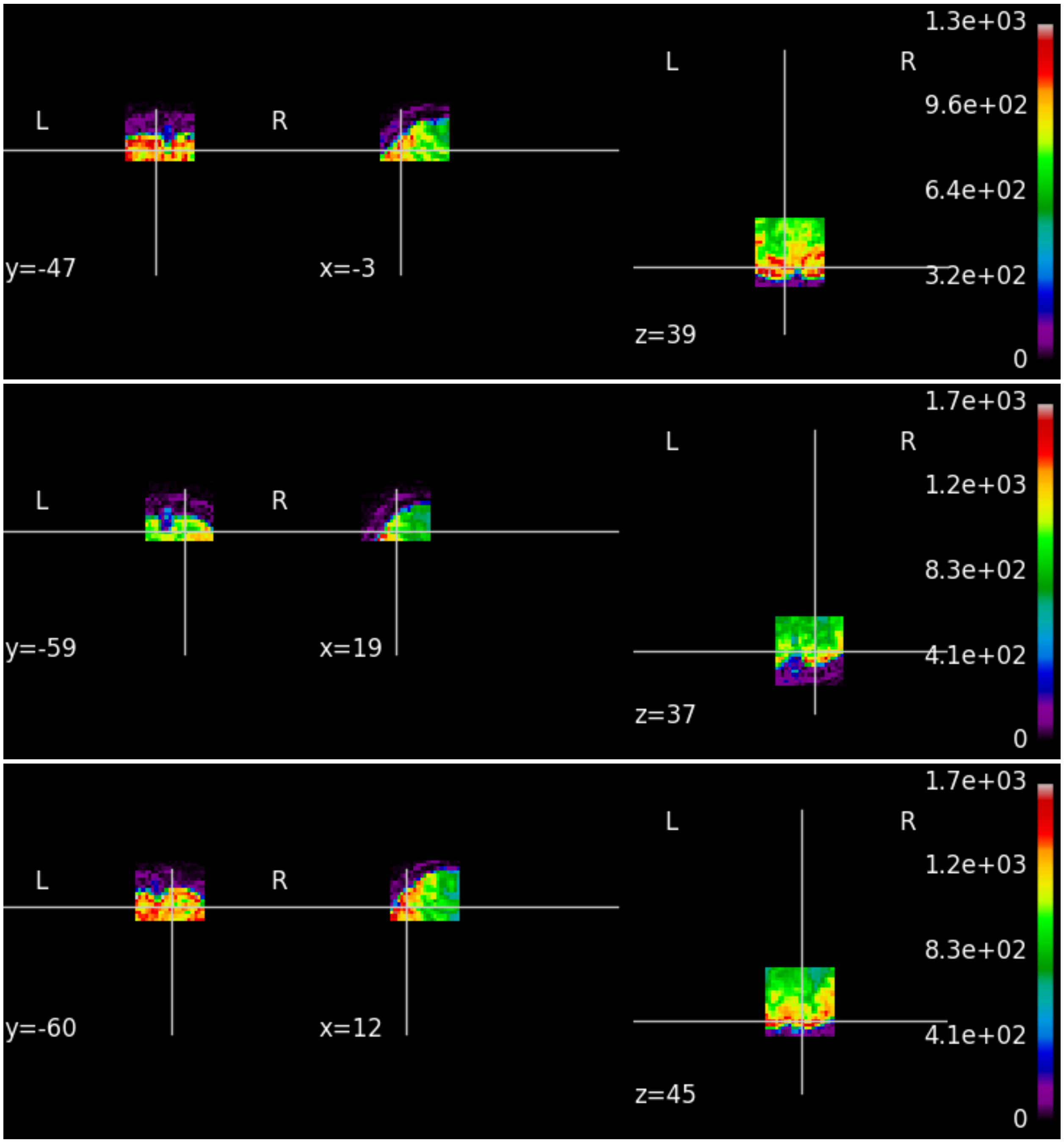


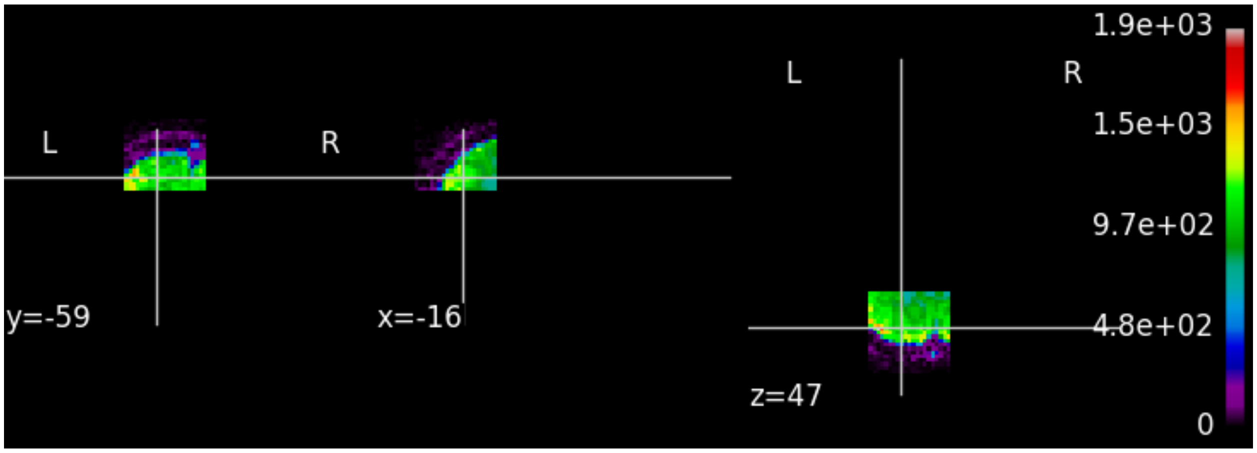
Filter 2's patches:

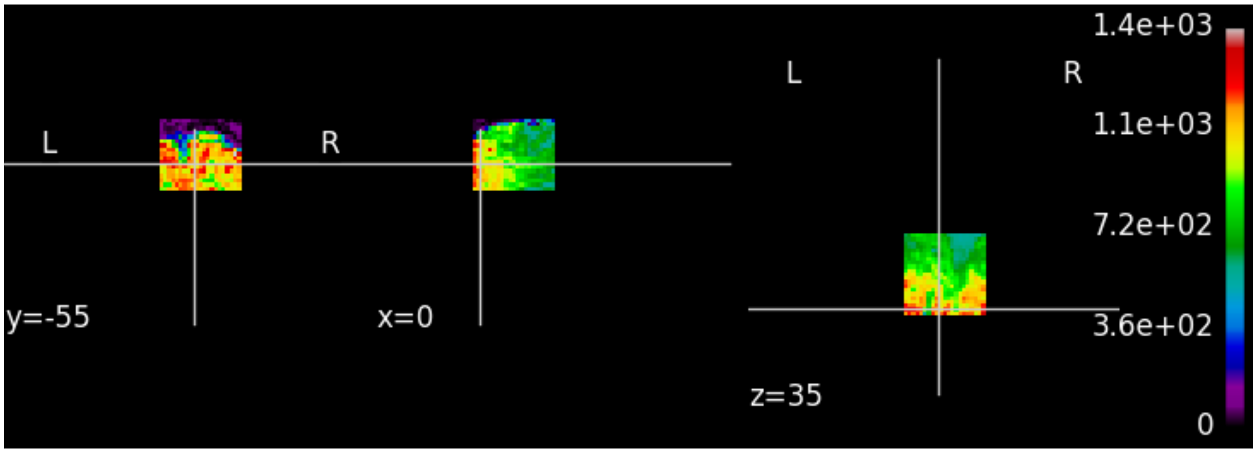





L
R
y=-52
x=29
L
R
z=67
1.4e+03
1e+03
6.8e+02
3.4e+02
0
L
R
y=-60
x=12
L
R
z=45
1.7e+03
1.2e+03
8.3e+02
4.1e+02
0
L
R
y=-27
x=51
L
R
z=45
1.9e+03
1.4e+03
9.3e+02
4.6e+02
0


L
R
y=-47
x=-3
z=39
1.3e+03
9.6e+02
6.4e+02
3.2e+02
0
L
R
y=-59
x=19
z=37
1.7e+03
1.2e+03
8.3e+02
4.1e+02
0
L
R
y=-60
x=12
z=45
1.7e+03
1.2e+03
8.3e+02
4.1e+02
0


L
R
y=-59
x=-16
L
R
z=47
1.9e+03
1.5e+03
9.7e+02
4.8e+02
0


L
R
y=-55
x=0
L
R
z=35
1.4e+03
1.1e+03
7.2e+02
3.6e+02
0