Creating Synthetic Images to Achieve Desired Brain Activation Patterns

Authors: Cameron Lester, Florrie Li, Jesse Chen

Project Advisor: Dr. Amy Kuceyeski

Cornell University

Table of Contents

ABSTRACT

The goal of our project was two-fold, first we utilized the BOLD5000: Brain, Object, Landscape Dataset produced by Chang *et. al* in order to build a model which would predict brain activation levels in the visual cortex. We then planned to develop a generative model to create synthetic images able to achieve desired brain activation levels in our predictive model. The BOLD5000 dataset contained fMRI data of four participants who viewed over 5,000 images while in an MRI machine; we note that the fourth participant failed to complete all MRI sessions. The corresponding blood oxygen level dependent signals were recorded for each respective participant as they viewed these images and were used to interpret the neuronal activation of each participant during respective viewings. In order to use this data, we had to first transform the raw neuronal signals. The data was manipulated such that we could reproduce all correlation graphs present in the Chang *et. al* paper to ensure we had data in the proper format for our research (Figures 1-3 in the appendix). We proceeded to run an AlexNet feature extraction on the mean brain activation levels of the three participants to fully complete their MRI sessions and reduced the resulting matrix using a Principal Component Analysis (PCA) for dimensionality reduction. Once this resulting matrix was in a workable form, we utilized it as the input for an ElasticNet regression in order to create our final prediction model. We proceeded to train this model using ten unique training and testing pairs containing all presented images from the BOLD5000 dataset.

Unfortunately, our final model ended up being incomplete. Due to computational limitations, we were only able to train our prediction model on the first convolutional layer of the AlexNet model. It is well documented that the removal of any layer from AlexNet

results in immediate and significant deterioration of the model's performance; therefore, we cannot conclude the validity of our final predictive model based on its current state (Krizhevsky). Due to a combination of computational limitations forcing us to use a reduced model and time lost due to the current pandemic, we were unable to complete the secondary goal of our project, building a generative model which creates synthetic images that achieve desired brain level activation. However, this research project is planned to continue well into the summer until it is ready for publication under the direction of Dr. Amy Kuceyeski.

INTRODUCTION

Background

The field of vision science has seen dramatic growth and development in the last decade. With stunning advances in the fields of artificial intelligence, neuroscience, engineering, and statistics, this interdisciplinary field has uncovered some of the fundamental strategies that our brains utilize in order to grant us sight. Turning the organic into the algorithmic, recent machine learning techniques such as deep learning have enabled the prediction of brain activation patterns based on given stimuli. In attempting to emulate the method by which neurons in the brain fire and communicate, deep learning networks are able to extract important features from images which may then be used as weights for predictive or generative models. These generative models may then be used in the creation of visual stimuli which specifically activate a targeted region of the brain.

While these models may be strong predictors, in order for any of them to be accurate, they must have robust and reliable data to train from. For a task as complex as mimicking how the human brain processes visual stimuli, a model would have to be trained on an incredibly vast and varied dataset; this was exactly the goal of the team behind AlexNet. AlexNet is a deep convolutional neural network which was trained on 1.2 million high-resolution images from the ImageNet dataset. The team developed AlexNet with eight learned layers, five convolutional and three fully-connected, with the output of the last fully-connected layer being fed into a 1000-way softmax which then produces a distribution over the 1000 class labels for final image classification (Krizhevsky). AlexNet was a record breaking neural network when its paper was published in 2012 due to its robust nature and usage of a common image dataset, ImageNet, and it remains a strong competitor amongst image classifiers.

The powerful pattern recognition developed in AlexNet is exceptionally helpful in dealing with image classification and feature extraction. For our purposes, we input the BOLD5000 dataset and extract features which will later be fed into a regression model. BOLD5000 is a human function MRI (fMRI) study that includes nearly 5,000 distinct images from the three most commonly used image datasets: Scene UNderstanding (SUN), Common Objects in Context (COCO), and ImageNet (Chang). The study presented these nearly 5,000 images to four participants who viewed them across sixteen fMRI sessions, recording the image presented and blood oxygen level dependent brain activation signals each time an image was viewed by the respective participants. The dataset includes nearly 20 hours of MRI scanning per each of the four participants, with the most noticeable exception being that the fourth participant completed only 10 of the 16 sessions and thus had significantly

less data compared to the other participants. A total of 5,254 image viewings were conducted, of which 4,916 viewings were unique, with 112 images repeated four times and one image repeated three times. Due to the fact that the BOLD5000 uses ImageNet for a portion of the dataset, the AlexNet model will already have a level of exposure to the data and have an easier time with initial classifications. While this could lead to problematic overfitting, and any predictions from this model may be overly accurate due to training or testing on repeated images, AlexNet is merely a single step towards our prediction model and these issues will not affect the final prediction model. Furthermore, AlexNet's architecture already takes steps to prevent overfitting; these details are explored under the AlexNet architecture section of our methodology.

Combining the power of AlexNet with the robustness of the BOLD5000 dataset, we are able to extract relevant features from the images that each participant was presented with. Then, running these results through ElasticNet, a regularized regression method which linearly combines the L1-norm and L2-norm penalties of the lasso and ridge methods, we are able to predict the brain activation patterns when our model "views" an image. The LASSO (least absolute shrinkage and selection operator) method shrinks coefficients of the matrix to exactly zero; on the other hand, the ridge method attempts to shrink coefficients as close as possible to 0 without forcing them to 0. The ratio by which ElasticNet utilizes these methods is an important choice which can dramatically affect the regression and strength of our predictive model; how we made this choice is outlined further in our methodology. Once we are able to predict brain activation patterns based on the features extracted from viewed images, we will be able to create synthetic images that specifically target and activate a chosen area of the brain.

We will utilize a generative model to create our synthetic images. The model is planned to be a variant of AlexNet and CaffeNet, which has been proven to be a reliable generative model (Ponce). CaffeNet's architecture is similar to AlexNet, but as outlined in the Ponce paper it will consist of nine total layers, three fully-connected layers and six deconvolutional, instead of the eight total layers within AlexNet. The generative model will be fed the same images from the BOLD5000 dataset as was AlexNet. As we iterate over the generations, we expect the synthetically produced images to be able to cause specific brain activation patterns within our prediction model.

METHODOLOGY

Materials and Data

The BOLD5000 dataset that we utilized is available for free online via Chang *et. al*'s paper and Github. All Python packages, including our pre-trained AlexNet model, are open sourced and documented within our code attached at the end of this paper (TorchVision). Notably, we chose the *split_folders* package designed by MIT in order to evenly split the SUN, COCO, and ImageNet folders. This package allowed us to easily split the presented stimuli sub folders and create multiple training and testing sets while remaining significantly less computationally expensive than our originally constructed function.

Pre-Processing of Data

To predict how the brain responds to visual stimuli, we created a population-level activation map based on the fMRI data obtained from the BOLD5000 dataset. We built a

machine learning model to predict brain activation levels within different regions of interest (ROIs) of the brain based utilizing the BOLD5000 images as the inputs.

Due to the fact that the fourth participant was unable to complete the 16 total MRI sessions, they are missing data and as such were not a focus of our research. We chose instead to build our model using only the first three participants as they had completed all sessions and viewed each image the same amount of times. While we later plan to include the fourth participant, their initial elimination from our model allowed us to utilize all images and ROIs while avoiding possible inconsistencies in our analysis.

Since the BOLD5000 fMRI data contains brain activation levels for each individual neuron across the different regions of interest, analysis quickly became computationally expensive and cumbersome to interpret if we chose to keep all of this data. Since we weren't interested in the way that each neuron responds, but rather the overall way in which visual stimuli activates regions of interest, we instead calculated the mean of the neuronal activation within these regions of interest and retained it as our relevant information instead of the voxel-wise neuron values. Keeping data on a neuronal level didn't align with our goal either, so instead of using the individual values we took a mean and retained the mean as the activation level for each region of interest.

To make sure that our data was manipulated and handled in the same way as the BOLD5000 dataset, which would allow for comparative analysis, we reproduced Pearson correlation maps for the first three subjects emulating Fig. 5 in the Chang paper. In this analysis we didn't use average neuron activation level, that is to say, we used the original fMRI data without our previous manipulation. This allowed us to verify that we had manipulated our data to the same extent as the BOLD5000 paper as the data that is open

sourced lacks much of the transformations that were present in the paper. Figures 1-3 in the appendix represent these Pearson correlation maps and can be compared against the Chang paper for similarity.

$$r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\,\sqrt{n\sum y_i^2 - (\sum y_i)^2}}.$$

(Figure 4. Formula for the Pearson correlation coefficient.)

We conducted the calculation on two levels, repeated images and non-repeated images. Each repeated image was shown to participants either three or four times (112 were shown four times, one was shown three times) so within each region of interest we calculated the average correlation for the same image, repeated the process for all 113 repeated images, and averaged over 113 correlation coefficients. To calculate the average correlation for the same images, we took neuron activation levels of different image appearance as our input. For example, neuron data of the first appearance of image_1 was the X_i and neuron data of the second appearance of image_1 was the Y_i. For the non-repeated images, we still used the same 113 images but only the data from the first appearance was used so that the two could be considered distinct images. Here, we were able to simply use the neuron data of different inputs and calculated the respective mean correlation coefficients. After the calculation, a bar chart was made to visualize the result and check against similar figures present in the Chang paper. Once we knew that our data was properly manipulated and handled, we were able to begin building our models and predicting brain activation levels across the ROIs.

Data Partitioning for Training and Testing

In order to train our models in a way that would predict brain activation patterns, we had to take several steps to assure we would avoid inconsistencies in our data. There were three key concerns in training our models: How would we assure that we have a robust enough training set to assure that we don't have biases towards certain features? What is the optimal way to train our models, individually or as a sum of participants? Do repeated images need to be accounted for when training these models?

Towards robustness, by utilizing the split_folders package we were able to generate ten uniquely and randomly seeded train/test sets. By training our models across these ten sets, we were able to analyze the feature extractions and regression variables from the ten different models to spot any significant outliers or discrepancies.

Towards optimality, we then chose to take the sum of the three participants who fully completed the 16 fMRI sessions as our initial input. The ROIs for the first three participants were averaged and then fed through ElasticNet for prediction model training. We made sure to track which ROIs corresponded to which images, so that the ElasticNet could be trained from the feature extraction and tested against the true response for validation.

Towards image accountability, we initially chose to account for repeated images and removed them from our initial training and testing sets. However, we came to the conclusion that this will likely only hurt us later down the line. This would be due to the fact that our model would never encounter the images that would be in our "gold standard set." If our model never encountered these images, it would likely hurt our end result; the

slight chance of overfitting that is caused by including these 113 images is negligible when weighed against the fact that 113 images comprise barely 2% of the total image pool.

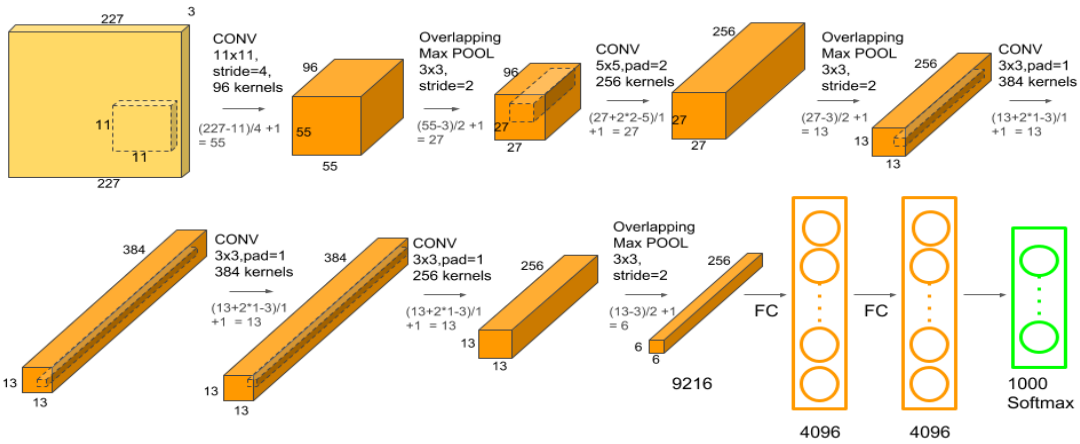
Once we had decided on these characteristics for our training and testing sets the ten sets were produced from ten seeds which were randomly generated from the interval 1-9999. This range was chosen because it includes the default seed of 1337 without increasing the range of the seeds by a magnitude of the default value. For replication purposes, the following seeds were generated in this order, with each model built sequentially: 1337, 6700, 1640, 8218, 3074, 9510, 6358, 2934, 8799, and 7181. Due to the nature of the split_folders function, when replicating this split it is necessary to manually move images out of each subfolder created if utilizing this paper's code. We did initially attempt to use a function to solve this issue. But the time spent getting it to work quickly approached the time cost to manually sort them, as such we chose the latter.

AlexNet Usage and Architecture

AlexNet is the model we used for image feature extraction. We used the open source code of AlexNet from the PyTorch package utilizing a pretrained model. As previously discussed, the architecture of AlexNet consists of eight layers: five convolutional layers and three fully-connected layers. AlexNet utilizes the ReLU activation function, rather than the tanh and sigmoid functions, across its layers and most of the convolutional layers are followed by max-pooling layers. For our purposes we were only interested in the convolutional layers. Namely, we extracted all the weights from each of the five convolutional layers whenever we ran an image input through the model. In order to accomplish this, we initialized five new instances of AlexNet wherein each would stop at

different convolutional layers and only return the features from said layers. However, due to limitations of our hardware, we could only extract the first layer during our initial runs. All five layers will be extracted in future research utilizing AWS cloud computing. While we are currently only able to extract the first layer, analysis of it is still prudent and provides a glimpse at the potential accuracy of our full model. We do recognize the fact that AlexNet accuracy is known to deteriorate significantly as layers are removed. With this in mind, our current analysis should not be representative of the full model's success.

As noted in the introduction, we chose AlexNet because it was trained on over 1.2 million images from ImageNet. While this could lead to overfitting issues, AlexNet addresses overfitting through data augmentation of the images, such as horizontal reflections or translations, and introduces a dropout to the model. Considering this, we were comfortable using AlexNet as it already took the necessary steps to ensure model integrity. While we could have chosen more recent models, AlexNet has been a stable and well-understood model for nearly a decade which allowed us to focus on our predictive model rather than worry about how our classifier model would function and identify significant features for extraction.



(Figure 5. AlexNet Architecture colored and retouched by Muneeb ul Hassan.)

To feed our images into AlexNet, transformations of the images had to be done as the model only accepts a particular image format. We needed to ensure that our images had a size of 224x224x3, transform them into tensor objects, and then normalize them for AlexNet to be able to process the images. As we were already working with data that was partially processed by the BOLD5000 paper, only a few of our images weren't in a formatted square shape. For irregular images, we needed to perform a center crop in addition to resizing. The total number of "irregularly-shaped" images was too small to make an impact on the analysis, and as such a center crop was acceptable to fit them for training and didn't introduce significant irregularities or discrepancies.

The transformed training images were then fed into the new AlexNet model we created and the results were appended. We wanted to use these results as the input variables, our X matrix, for the training process. However, the output of each image after feeding them through the AlexNet model was large and computationally expensive, making it unrealistic to handle due to the amount of time required for each iteration. Furthermore, the output had significantly more columns than rows, approximately 150,000 columns to

5,000 rows, making this output not desirable for our training process. To solve this issue, we decided to run a Principal Component Analysis (PCA) covering 95% of the variance on the input variables. This successfully reduced the input matrix to a much more manageable size of around 1,200 columns.

For the response variable, our Y matrix, we used each image's corresponding average activation level across the regions of interest computed from the fMRI data of the first three participants. We averaged across all three participants and chose not to use each neuron's individual activation level but rather the average activation for each region. As such, we made ten models, one for each region, to produce our final matrix.

ElasticNet Usage and Theory

We chose Elastic Net to fit our model because we wanted both coefficient shrinkage and further dimension reduction of the data. This was in order to reduce error as the number of columns within the data was still quite large after PCA reduction, overcoming some limitations of the LASSO method. The LASSO method is one of the two regression analysis methods ElasticNet utilizes, the other being the ridge method as mentioned in the introduction. Since LASSO encourages variable exclusion by shrinking the coefficients of the matrix to exactly 0, the multicollinearity of this large matrix caused long computation times on our hardware which made the method inefficient. Therefore, we had to find a proper ratio for the LASSO and ridge method's respective L1 and L2 values.

$$\hat{\beta} \equiv \underset{\beta}{\operatorname{argmin}}(\|y - X\beta\|^2 + \lambda_2 \|\beta\|^2 + \lambda_1 \|\beta\|_1).$$

(Figure 6. Calculation of the estimates from the ElasticNet method.)

Determining the ratio of L1 and L2 penalties is crucial when applying the Elastic Net method, so we implemented a ten-fold cross validation process to choose the best ratio. The function MultiTaskElasticNetCV from sklearn.linear_model was used for this task wherein the optimization objective is as follows:

```
(1 / (2 * n_samples)) * ||Y - XW||^Fro_2
+ alpha * l1_ratio * ||W||_21
+ 0.5 * alpha * (1 - l1_ratio) * ||W||_Fro^2
```

(Figure 7. Optimization objective.)

In the model specification, we set alpha to be picked automatically and run a list, *l1_ratio,* containing the values [.1, .5, .7, .9, .95, .99, 1]. In the end, our ElasticNet model was biased towards the ridge method at a ratio of 0.1:0.9 for the L1 and L2 values respectively. The ridge method is much better at handling large matrices with multicollinearity. Understanding this, it makes sense that we would have an L2 bias and our calculations prove this to be true.

Once this ratio was chosen, our ElasticNet was able to run a penalized regression method. The ridge regression shrinks the coefficients in a way similar to LASSO. However, it aims to shrink the coefficients as close as possible to 0 rather than exactly to 0. Unlike LASSO, ridge performs well even with multicollinear models and instances of a large amount of predictors with a small amount of observations. With our ratios, this meant that the penalty was primarily a shrinkage towards 0 rather than a forced elimination to 0 across the data.

Model Validation

Once we had a fully trained model, we needed a way to find the best possible accuracy that we can get to compare against our average accuracy. This would allow us to interpret our results against the best possible performance for our model. Based on our data, the highest accuracy of our model should occur whenever one of the 113 repeated images were viewed. To discover the gold standard we utilize two new sets, S_1 and S_2. S_1contains the average neuron activation levels for all ROIs during the first viewing of each image. S_2 is equivalent, however, it also contains the second appearance.

First, we calculated the Pearson correlations between S_1 and S_2 in each region. Next we computed the correlations between S_1 and the predicted average activation levels on repeated images in each region. To calculate the predicted value, we applied the same transformation process we originally performed on the training images but this time solely on the 113 repeated images. Then we used the trained Elastic Net model to acquire the predicted values. The latter set of correlation values divided by the former gave us the gold standard we were looking for.

Having the gold standard, we could start testing our model performance against the highest accuracy it could achieve. We started by looking at both the training and testing mean squared errors. The formula is provided below:

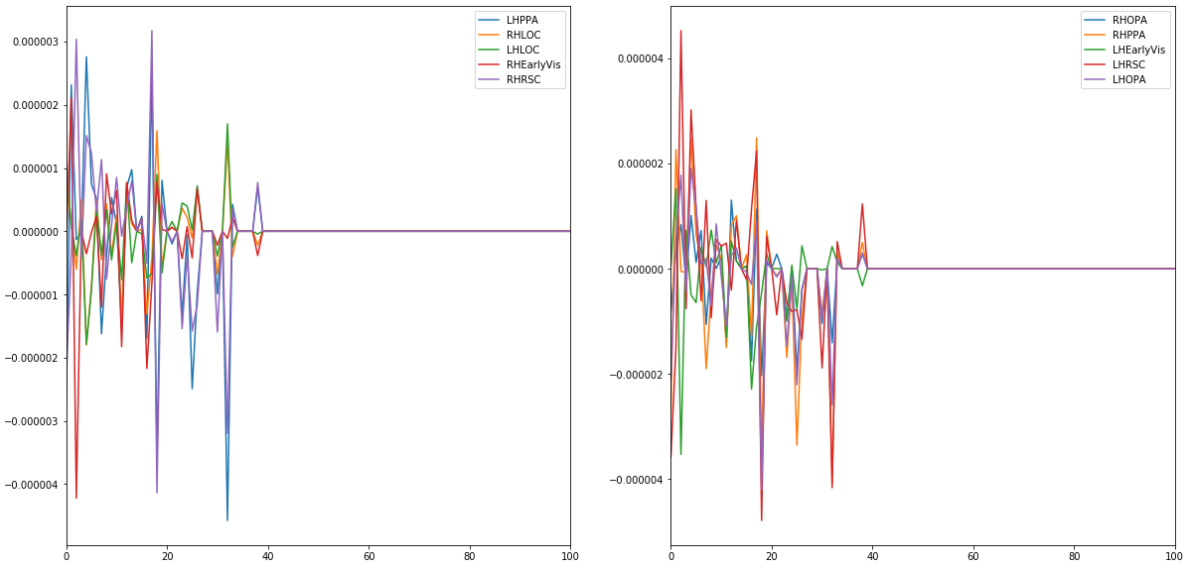
$$MSE = \frac{1}{n}\Sigma\left(\underbrace{y - \hat{y}}_{\text{The square of the difference between actual and predicted}}\right)^2$$

(Figure 8. Formula for the Mean Squared Error, MSE.)

The MSE results weren't too informative, so we calculated the correlation between our true response variables (actual average neuron activation level of the test images) and the predicted response variables (predicted average neuron activation level of the test images). We then compared this correlation to the correlation between S_1 and S_2.

The correlations among repeated and non-repeated images clearly indicated that the average brain activation levels are similar when people are looking at the same image, which is what we would expect. Although the maximum correlation is around 0.1, these repeated images correlations are much more significant than the uncorrelated ones. It is also clear that correlations are much stronger in some particular regions of interest, but there is no clear overall pattern of which region is more significant across each test subject. Presumably this is caused by the difference of each subject's brain.

After running the ten-fold cross validation, we get an alpha value of 0.053 and L1 ratio of 0.1 which means our model is leaning much more towards the ridge method rather than the LASSO method as expected. We note that the coefficients of each of the regions are different as they vary across the ten different models we built; observe that all the coefficients after approximately the 40th are zero.



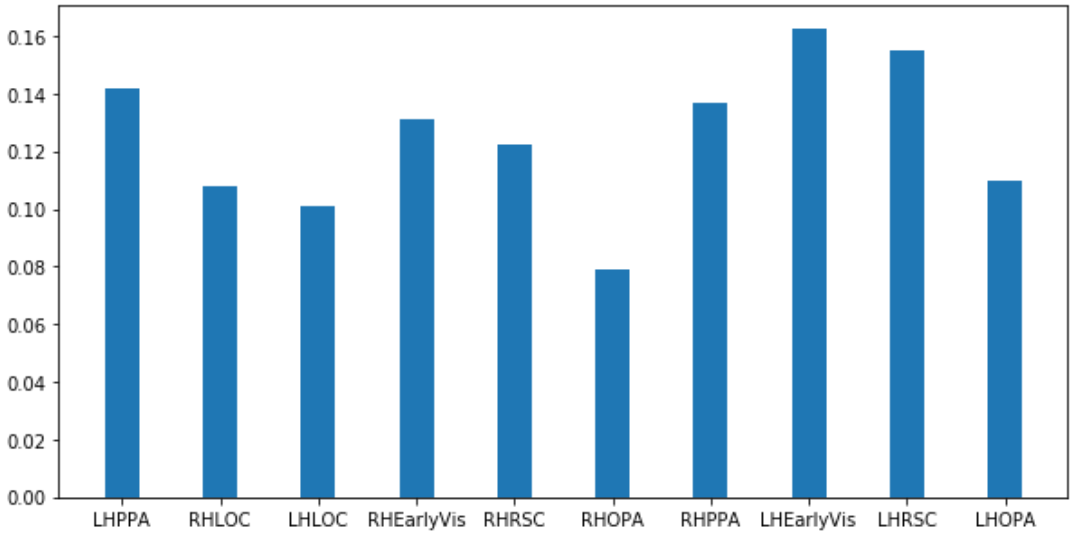
(Figure 9. A graph of coefficient values across regions.)

More investigation confirmed that a very small amount of the coefficients are nonzero.

```
LHPPA 28
RHLOC 28
LHLOC 28
RHEarlyVis 28
RHRSC 28
RHOPA 28
RHPPA 28
LHEarlyVis 28
LHRSC 28
LHOPA 28
```

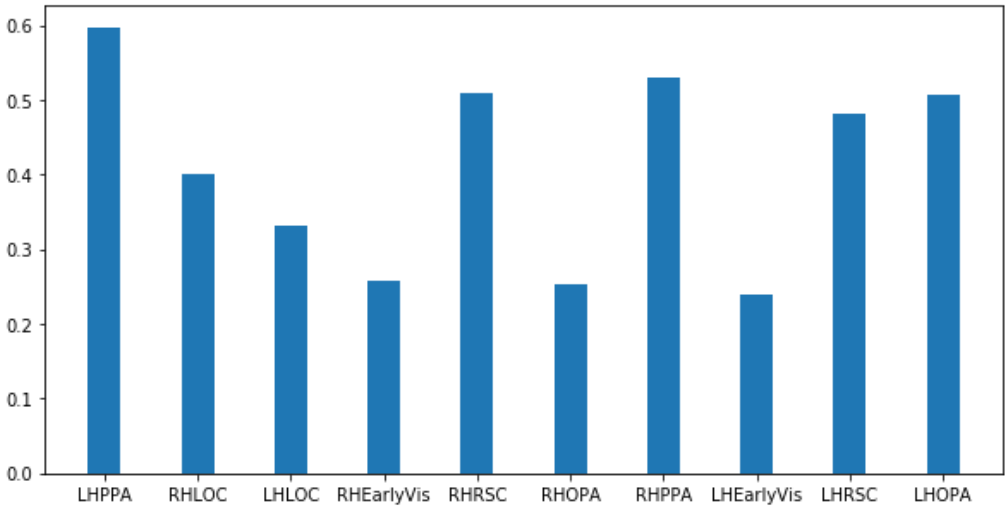
(Figure 10. The number of nonzero coefficients in each region across the models.)

Our training mean squared error and testing mean squared were 1.04e-05 and 1.07e-05 respectively. Although these two numbers may seem to be small, they don't necessarily mean that we reached a perfect model. This is because our response variables already had very small magnitude so their respective errors would logically be miniscule as well. However, this does show that we didn't overfit or underfit the data. The correlations between true response variables and the predicted response variables suggests that our model has a relatively good fit as we highlight in Figure 11.



(Figure 11. Correlations of true test set vs predicted test set.)

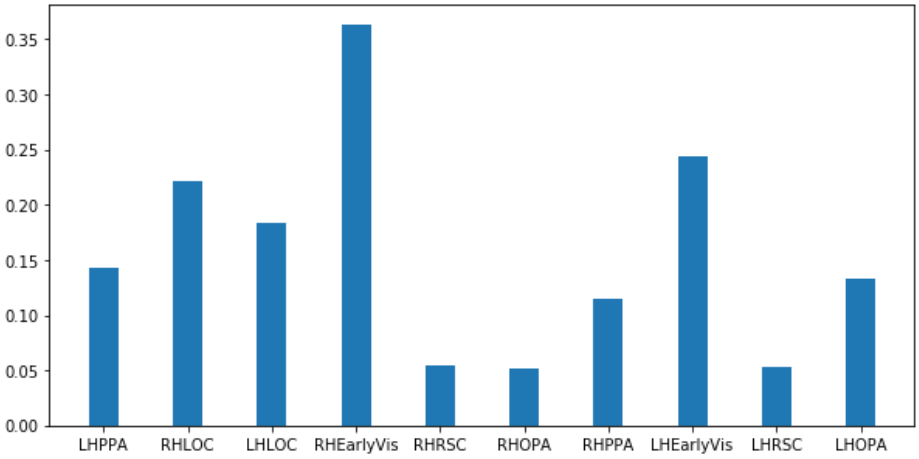
Now we look at the correlations of S_1 vs. S_2.



(Figure 12. Correlations of S_1 vs. S_2.)

The correlations illustrated in Figure 12 indicate that even if it was the second time a person saw the same image, their brain still behaved differently, even if just marginally. The nature of how a person interprets or interacts with a secondary viewing might be the reason for the brain behaving differently. This is reflected in the early visual regions having relatively low correlations. When it's the second time a repeated image appears, the

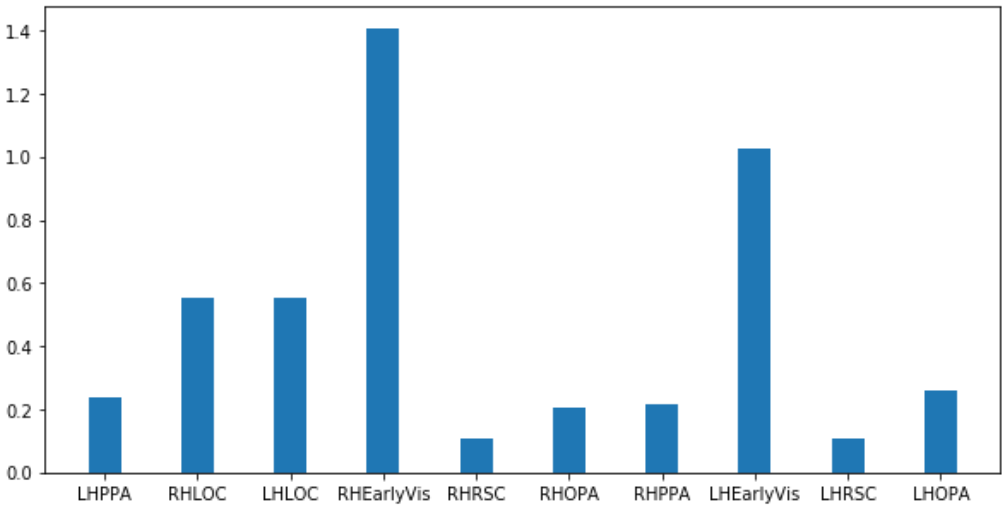
early visual regions won't activate that much because the person might realize they have seen this image before and these regions reflect the early activities. We compare the correlations found between True S_1 vs Predicted S_1, which figure 13 illustrates below:



(Figure 13. Correlations of True S_1 vs Predicted S_1.)

We see that the correlations between True and Predicted are much smaller than expected in several areas. We do note that the right hemisphere early visual (RHEarlyVis) and left hemisphere early visual (LHEarlyVis) cortexes were still very well correlated. As discussed, that is likely because the immediate viewing of the image, prior to recognition or classification, doesn't introduce as many variabilities in the early visual cortexes as compared to other regions of the brain.

Finally, we compared the ratio between the two sets of correlations. The ratio indicates that both the early visual regions were very well predicted, some even better predicted than the second appearance as Figure 14 demonstrates:



(Figure 14. Ratio of the two set of correlations)

Even without a robust and complete model, we can still observe that the early visual cortex regions of both the left and right hemisphere appear the most consistent and correlated in our predicted model. When we extrapolate this to the biological, this makes sense as the early visual cortex of our brain doesn't perform most of the complex categorization and recognition as other parts of the visual cortex. Therefore we wouldn't expect this correlation to vary too wildly from each viewing.

Overall, this initial and incomplete model still shows promising correlations and predictive capacity. It is as logically valid as we could make it considering our computational limitations forcing us to only extract one layer from AlexNet. We do want to openly recognize and concede that the incompleteness of our model could cause our analyses to seem inappropriate as the architecture of AlexNet is not meant to be analyzed in the deconstructed state that we have. However, as we plan to continue this project until we complete all of our models, we treat this analysis as an initial rather than final conclusion.

CONCLUSION

Results

We were able to extract features from the first layer of our AlexNet model and run regression on it utilizing an ElasticNet model. The results showed a low correlation between repeated images, which we interpret as even its best case scenario still has low predictability. This makes sense as, due to computational limitations, we were unable to build and execute a full AlexNet model on our hardware. However, the fact that the predictability was nonzero, at 10%, shows that a full model could prove very successful. These results are promising as we continue to build our model. At the minimum, it is proof that our current model and path of research produces some level of predictability which we can further develop. It is worth noting that our model analysis revealed an incredibly sparse matrix which hinders our predictive abilities; this is entirely due to the fact that we only extracted the features present in the first layer of AlexNet. This weak predictive potential is not indicative of the full model's potential success.

We were unable to fully accomplish our goal of synthetic image generation due to hardware limitations. However, we have recently acquired an AWS account thanks to our project advisor and plan to begin building the generative model soon. Our plans for how the model will be built remain the same, an emulation of CaffeNet (Ponce). Full details on this future generative model are present in our further research section.

Discussion

Although the correlation value between first and second viewings were low in our model validation, there are several reasons why we should still find this result promising.

We've already conceded that our model would have significant issues due to the fact that we were only using the first convolutional layer of our AlexNet, greatly deteriorating its performance. Then, considering the fact that we are averaging the viewing of three separate participants, 0.1 correlation is fairly high. The ways in which the participants were viewing separate parts of the image during separate viewings, their interpretations of the images, and what they were thinking during the viewing for each image would rationally be incredibly variable and unique. Therefore, achieving 10% correlation on a deteriorated model is a promising sign. As we build a more robust model with our newly acquired computational tools, we should expect this correlation to improve.

Regardless of these issues, our research is still important and promising. If we are able to correctly predict and stimulate the brain activation patterns of our artificial brain, this would provide evidence that the same should be possible with actual people. Being able to target specific regions of the brain via visual stimuli may provide further insight into how visual triggers affect certain neurological conditions, such as epilepsy, and how these visual stimuli are rendered, interpreted, and reacted to by the brain.

Recommendations

As we ran into some computational limitations with this paper, for reproducibility purposes, we note here some of the minimum specifications required to run all code we have produced for the project: Intel i7-8565U 1.80 GHz CPU or better, 16 GB minimum memory, Intel UGH Graphics 620 GPU or better. Although for a majority of the research process we had no issues with computational ability, when building our model we ran into significant issues. Extracting all five convolutional layers from AlexNet resulted in the

crashing of all three laptops, and a single desktop (24 GB RAM + GPU memory), due to the RAM and graphical memory required. We would highly recommend utilizing cloud computing or hardware much better than the specifications laid out above. Additionally, if it is required to use a system similar to ours, practice frequent data backup and ensure a valid system restore is present on your computer.

Non-hardware recommendations include performing different transformations on the input data. This paper chose to use an average of the three participants as our input data to complete the study, but there are several other valid transformations that could also suffice. An interesting transformation could be a method which weights the neuronal activation of regions of interest based on the total viewings of that image; this would allow the inclusion of the fourth participant without introducing abnormalities to the data. We could also keep the data at the individual level and the model training could instead be individualized for each participant. This would require building an AlexNet and ElasticNet model for each participant, although it may be the most thorough approach possible to this research.

Limitations

There were several limitations previously mentioned that we would like to further discuss. First and foremost, we conducted this project during the COVID-19 pandemic which caused us to lose a significant amount of workable hours during the semester. This caused us to run out of time before being able to build a generative model to produce synthetic images. However, we were still able to produce multiple models and complete all data manipulation required for further research. Most of the groundwork has now been laid out in such a way that synthetic image generation should be just a few weeks away.

Next, as mentioned extensively in our methodology, our group ran into severe computational limitations. When we began this project, we had very few reasons to suspect that such computational limitations would occur as our system specifications were well above average. However, as we progressed towards a full model we began to experience system instabilities and realized that continued development without the assistance of cloud computing would be futile. A majority of our computations were conducted on our laptops which were hardware upgrade limited. We have recently begun to acquire an Amazon Web Services cloud computing account and plan to complete our computations once we have full access to it.

Further Research

Current plans for further research include finishing the remaining model building utilizing AWS and working towards a generative model for synthetic images. We plan to use the Ponce paper as inspiration for the generation of synthetic images. Where the Ponce paper tested their generative models on monkeys, we plan to test them on our neural net models after we have fully extracted the features from the complete AlexNet model via ElasticNet.

Next, we will build separate models for each of the participants and observe whether or not these synthetic generation models are significantly different. If there is significant observed variability, we may have to rethink how our model should be structured. The variability may be extreme between participants if variables are lost by not utilizing individual neuronal activity for our prediction model. We may also observe

extreme variability if our underlying assumptions of regularity and overall neurological similarity regarding the brain activation across the participants are violated.

Once we have concluded whether or not we constructed our model in a logically sound manner, we will proceed towards the generation of synthetic images. We plan to replicate a generative model similar to the one discussed in the Ponce paper. The model will likely use CaffeNet which is a variation of AlexNet. We will emulate the architecture present in the Ponce paper as closely as possible, as we expect our model to contain nine total layers, three fully-connected layers and six deconvolutional. This model will be used to alter the images present in the BOLD5000 dataset until we are able to activate specific regions of the brain within our predictive model. At this point we will note which areas are targeted by which images and attempt to find underlying correlations and patterns between such images. Potential analyses on these images may reveal what classifications of visual stimuli produce the strongest activations in targeted regions of the brain.

Depending on how successfully we are able to target and activate our ten regions of interest within the brain, we will consider restructuring our model in a way to predict and activate broader areas of the brain beyond just these ROIs. Once we are confident in our approach, the CaffeNet model will be trained to either target overall brain regions or specific neuron groupings. We hypothesize that our synthetic images should be able to reliably target any region of our choice as long as our model's underlying assumptions and efficacy are true.

BIBLIOGRAPHY

Chang, Nadine, et al. "BOLD5000, a Public FMRI Dataset While Viewing 5000 Visual Images." Nature News, Nature Publishing Group, 6 May 2019, www.nature.com/articles/s41597-019-0052-3. 20 Dec 2019.

--- (2019) *BOLD5000: Brain, Object, Landscape Dataset*. Github, bold5000.github.io.

Hassan, Muneeb ul. "AlexNet - ImageNet Classification with Convolutional Neural Networks." AlexNet - ImageNet Classification with Convolutional Neural Networks, 1 Nov. 2018, neurohive.io/en/popular-networks/alexnet-imagenet-classification-with-deep-convolutional-neural-networks/. 3 Mar 2020.

Krizhevsky, Alex, et al. ImageNet Classification with Deep Convolutional Neural Networks. University of Toronto, 2012, papers.nips.cc/paper/4824-Imagenet-classification-with-deep-convolutional-neural-networks.pdf. 2 Feb 2020.

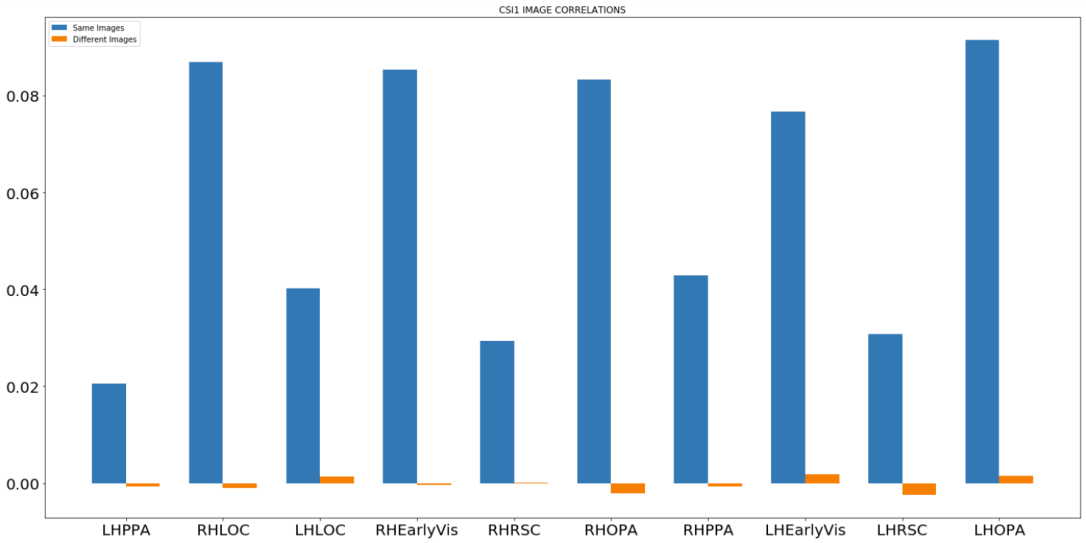
MIT. *split_folders.* MIT. 30 Jul 2019. Pypi.org. https://pypi.org/project/split-folders/.

Ponce CR, Xiao W, Schade PF, et al (2019) Evolving super stimuli for real neurons using deep generative networks. bioRxiv 516484. https://doi.org/10.1101/516484. 20 Dec 2019.

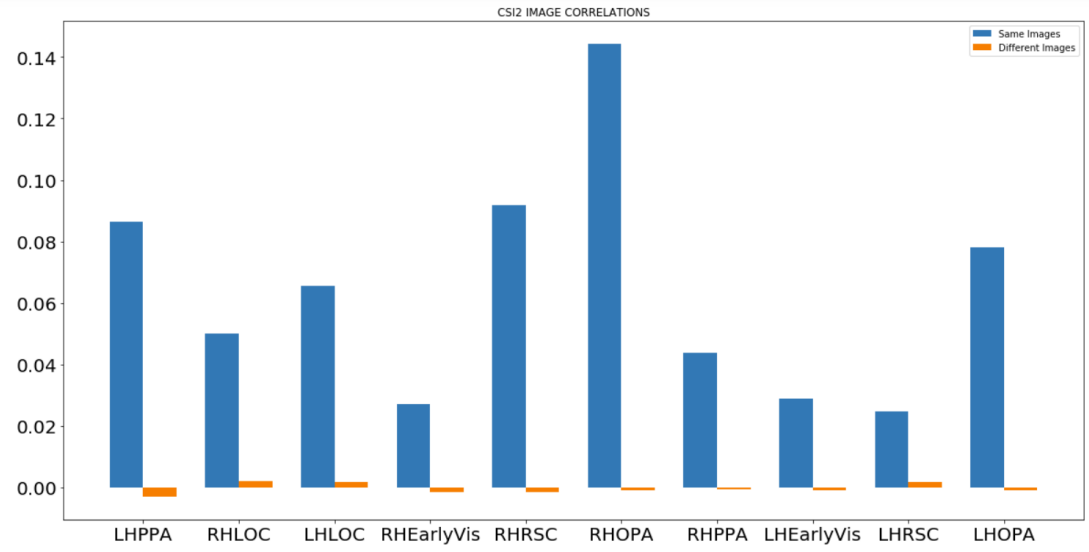
Torchvision.models.alexnet. "Source Code for Torchvision.models.alexnet." PyTorch Master Documentation. pytorch.org/docs/stable/_modules/torchvision/models/alexnet.html#alexnet.

APPENDIX

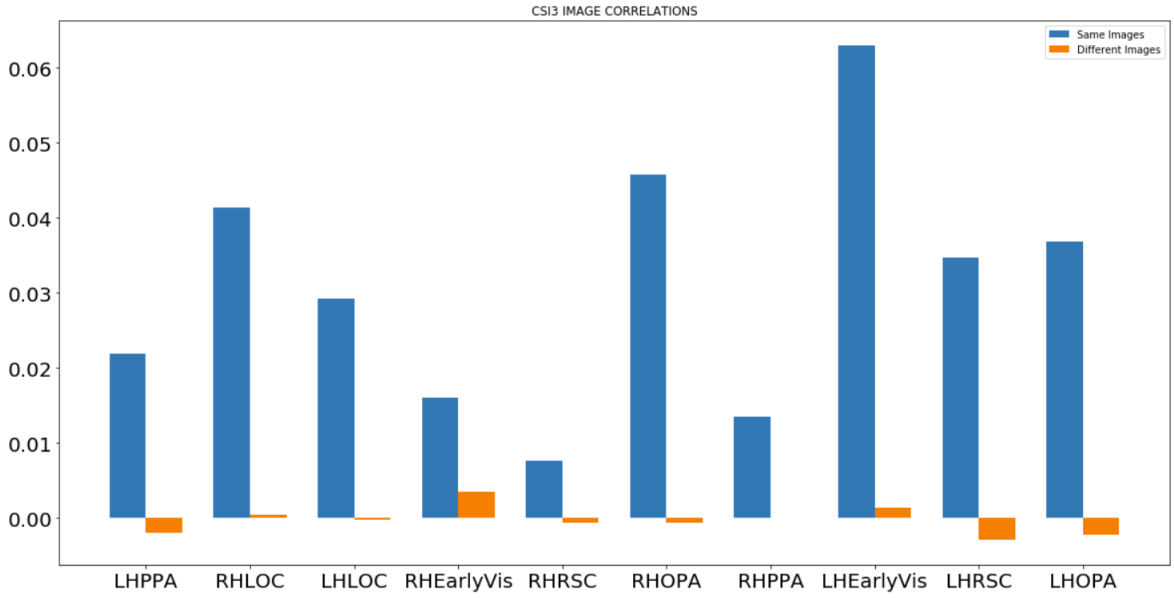
Figures



(Figure 7, the correlation for participant 1)



(Figure 2, the correlation for participant 2)



(Figure 3. The correlation for participant 3.)

Code

All data was gathered from the BOLD5000 github (Chang). In order to use our code, make sure to change directory paths to your system specifications and personal choice. Furthermore, as noted in our methodology, we do not build a function to pool the training and testing divisions of the stimuli subfolders. As such, it will be necessary to either create such a function or manually pool the images as outlined in our paper.

```
#!/usr/bin/env python
# coding: utf-8

# # Setting up our project

# In[1]:


#import nibabel as nib
import scipy.io as sci
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
import glob
import torch
import torch.nn as nn
from torchvision import models
from torchvision import transforms
from PIL import Image
import random
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import ElasticNetCV
from sklearn.linear_model import MultiTaskElasticNetCV
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
import os
from sklearn.model_selection import train_test_split
import split_folders


# In[2]:


#load roi files, this is the fMRI data for each test subject
csi1 =
sci.loadmat('C:/Users/nehcy/Desktop/stsci_5999/BOLD5000_ROIs/ROIs/CSI1/mat/CSI1_
ROIs_TR34.mat')
csi2 =
sci.loadmat('C:/Users/nehcy/Desktop/stsci_5999/BOLD5000_ROIs/ROIs/CSI2/mat/CSI2_
ROIs_TR34.mat')
csi3 =
sci.loadmat('C:/Users/nehcy/Desktop/stsci_5999/BOLD5000_ROIs/ROIs/CSI3/mat/CSI3_
ROIs_TR34.mat')
csi4 =
sci.loadmat('C:/Users/nehcy/Desktop/stsci_5999/BOLD5000_ROIs/ROIs/CSI4/mat/CSI4_
ROIs_TR34.mat')

#load the rep images labels
rep_list =
open('C:/Users/nehcy/Desktop/stsci_5999/BOLD5000_Stimuli/BOLD5000_Stimuli/Scene
_Stimuli/repeated_stimuli_113_list.txt').read().splitlines()

#create roi list and patient list for looping
roi_list = ['LHPPA', 'RHLOC', 'LHLOC', 'RHEarlyVis', 'RHRSC', 'RHOPA', 'RHPPA',
'LHEarlyVis', 'LHRSC', 'LHOPA']
subject_list = [csi1,csi2,csi3,csi4]
```

```
subject_string = ['csi1','csi2','csi3','csi4']


# In[4]:


#list of images presented, in order, four patients
csi1_pre =
open('C:/Users/nehcy/Desktop/stsci_5999/BOLD5000_ROIs/ROIs/stim_lists/CSI01_stim_l
ists.txt').read().splitlines()
csi2_pre =
open('C:/Users/nehcy/Desktop/stsci_5999/BOLD5000_ROIs/ROIs/stim_lists/CSI02_stim_l
ists.txt').read().splitlines()
csi3_pre =
open('C:/Users/nehcy/Desktop/stsci_5999/BOLD5000_ROIs/ROIs/stim_lists/CSI03_stim_l
ists.txt').read().splitlines()
csi4_pre =
open('C:/Users/nehcy/Desktop/stsci_5999/BOLD5000_ROIs/ROIs/stim_lists/CSI04_stim_l
ists.txt').read().splitlines()


# We get the average roi brain activision level over four patients
# but actually we end up using only three patients average

# In[5]:


#empty dictionary to contain all the patients' average data
subject_average = {}
#loop through each patient
for p, q in zip(subject_list,subject_string):
    #create empyty average activation dataframe for each patient
    roi_average = pd.DataFrame()
    for i in roi_list:
        #compute row average for each region, average activation for each stimuli with each
region
        roi_average[i] = pd.DataFrame(p[i]).mean(axis = 1)
    #put all patients' data into one dictionary
    subject_average[q] = roi_average


# In[8]:


#pull out the index rep images appear in the image presentation list
rep_index = []
```

```
for i, j in enumerate(csi1_pre):
    for k in rep_list:
        if j == k:
            rep_index = rep_index+[i]
#the number of index should be greater than 113 since they appeared multiple times
#I believe my for loop is not wrong, it does check mutiple appearances

#For each repeated image, we find the corresponding repeating index, store them in
rep_pairs
def rep_extract(csi_pre):
    rep_pairs = []
    for k in rep_list: #rep_list should be pre-defined
        rep_index = []
        for i, j in enumerate(csi_pre):
            if j.replace('rep_','') == k:
                rep_index = rep_index+[i]
        rep_pairs.append(rep_index)
    return rep_pairs

csi1_rep_pairs = rep_extract(csi1_pre)
csi2_rep_pairs = rep_extract(csi2_pre)
csi3_rep_pairs = rep_extract(csi3_pre)


# In[9]:


#In this function, first we find the average corr of the same image, then we find it for all 113
repeated images
#Finally we we average over 113 and get a number for each roi
def corr_avg(csi, rep_pairs):
    result = []
    for j in roi_list: #roi_list is pre-defined it contains all the roi regions
        total_mean = []
        for i in range(0,113):
            df = pd.DataFrame(csi[j])
            df = df-df.mean()
            df = df.loc[rep_pairs[i],:]
            corr = df.T.corr(method='pearson').unstack().reset_index(name="corr")['corr']
            not1_bool = (corr != 1) #we exclude diagonal correlations
            single_img_mean = corr[not1_bool].mean()
            total_mean.append(single_img_mean)
        result.append(sum(total_mean)/len(total_mean))
    return result

csi1_corr = corr_avg(csi1, csi1_rep_pairs)
```

```
# In[10]:


csi2_corr = corr_avg(csi2, csi2_rep_pairs)
csi3_corr = corr_avg(csi3, csi3_rep_pairs)


# In[11]:


#distinct images correlation
def corr_avg_unrep(csi):
    result = []
    for j in roi_list: #roi_list is pre-defined it contains all the roi regions
        df = pd.DataFrame(csi[j])
        df = df-df.mean()
        df = df.loc[rep_index,:] #rep_index is predefinds this is single unrep
        corr = df.T.corr(method='pearson').unstack().reset_index(name="corr")['corr']
        not1_bool = (corr != 1) #we exclude diagonal correlations
        single_mean = corr[not1_bool].mean()
        result.append(single_mean)
    return result

csi1_corr_unrep = corr_avg_unrep(csi1)


# In[12]:


csi2_corr_unrep = corr_avg_unrep(csi2)
csi3_corr_unrep = corr_avg_unrep(csi3)


# In[13]:


#plotting CSI1 IMAGE CORRELATIONS
labels = roi_list
x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars

fig, ax = plt.subplots(figsize=(20, 10))
same = ax.bar(x - width/2, csi1_corr, width, label='Same Images')
different = ax.bar(x + width/2, csi1_corr_unrep, width, label='Different Images')
```

```
ax.set_title('CSI1 IMAGE CORRELATIONS')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()
ax.tick_params('both',labelsize=20)

fig.tight_layout()


# In[14]:


#plotting CSI2 IMAGE CORRELATIONS
labels = roi_list
x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars

fig, ax = plt.subplots(figsize=(20, 10))
same = ax.bar(x - width/2, csi2_corr, width, label='Same Images')
different = ax.bar(x + width/2, csi2_corr_unrep, width, label='Different Images')
ax.set_title('CSI2 IMAGE CORRELATIONS')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()
ax.tick_params('both',labelsize=20)


# In[15]:


#plotting CSI3 IMAGE CORRELATIONS
labels = roi_list
x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars

fig, ax = plt.subplots(figsize=(20, 10))
same = ax.bar(x - width/2, csi3_corr, width, label='Same Images')
different = ax.bar(x + width/2, csi3_corr_unrep, width, label='Different Images')
ax.set_title('CSI3 IMAGE CORRELATIONS')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()
ax.tick_params('both',labelsize=20)


# # Essential AlexNet step
```

```
# Extract each convolutional layer

# In[9]:


original_model  = models.alexnet(pretrained=True)

class AlexNetConv1(nn.Module):
    def __init__(self):
        super(AlexNetConv1, self).__init__()
        self.features = nn.Sequential(
        # stop at conv1
        *list(original_model.features.children())[:0]
        )
    def forward(self, x):
        x = self.features(x)
        return x

class AlexNetConv2(nn.Module):
    def __init__(self):
        super(AlexNetConv2, self).__init__()
        self.features = nn.Sequential(
        # stop at conv2
        *list(original_model.features.children())[:3]
        )
    def forward(self, x):
        x = self.features(x)
        return x

class AlexNetConv3(nn.Module):
    def __init__(self):
        super(AlexNetConv3, self).__init__()
        self.features = nn.Sequential(
        # stop at conv3
        *list(original_model.features.children())[:6]
        )
    def forward(self, x):
        x = self.features(x)
        return x

class AlexNetConv4(nn.Module):
    def __init__(self):
        super(AlexNetConv4, self).__init__()
        self.features = nn.Sequential(
        # stop at conv4
```

```
        *list(original_model.features.children())[:8]
        )
    def forward(self, x):
        x = self.features(x)
        return x

class AlexNetConv5(nn.Module):
    def __init__(self):
        super(AlexNetConv5, self).__init__()
        self.features = nn.Sequential(
        # stop at conv5
        *list(original_model.features.children())[:10]
        )
    def forward(self, x):
        x = self.features(x)
        return x

model1 = AlexNetConv1()
model2 = AlexNetConv2()
model3 = AlexNetConv3()
model4 = AlexNetConv4()
model5 = AlexNetConv5()


# # Make 10 splits of training and testing images

# This Chunk of code is created by Cameron so we are seeing different file paths

# In[ ]:


#Setting up prefixes for the respective folders
root_dir = r'C:\Users\camle\Documents\MRI
Project\BOLD5000_Stimuli\Scene_Stimuli\Presented_Stimuli'
coco = r'\COCO'
image_net = r'\ImageNet'
scene = r'\Scene'
target_dir = r'C:\Users\camle\Documents\MRI Project'

#Creating relevant directories; commented out as they now exist

#os.makedirs(target_dir + r'\train')
#os.makedirs(target_dir + r'\test')

train_ratio = 0.7
test_ratio = 0.3
```

```
#Tuple of directory names to cycle through later
all_dirs = (coco, image_net, scene)

#Now that the directories exist, creating their readable paths for functions

train_path = r'C:\Users\camle\Documents\MRI Project\train'
test_path = r'C:\Users\camle\Documents\MRI Project\test'

#Setting up lists of directories and removing reps of all folders
COCO_files = os.listdir(root_dir + coco)
IN_files = os.listdir(root_dir + image_net)
Scene_files = os.listdir(root_dir + scene)

print(len(COCO_files) + len(IN_files) + len(Scene_files))


COCO_ready = []
IN_ready = []
Scene_ready = []

Function that will remove reps
def clear_reps (file_list):
  new_list = []
  new_list = file_list
  for file in file_list:
     for rep in rep_list:
        if file == rep:
           new_list.remove(file)
  return new_list

def check_reps (file_list):
  for file in file_list:
     for rep in rep_list:
        if file == rep:
           #new_list.remove(file)
           print("Match found")
  return

#For some reason I need to double-sweep the lists
COCO_ready = clear_reps(clear_reps(COCO_files))
IN_ready = clear_reps(clear_reps(IN_files))
Scene_ready = clear_reps(clear_reps(Scene_files))

Coco_check = check_reps(COCO_files)
IN_check = check_reps(IN_files)
```

```
Scene_check = check_reps(Scene_files)

print(len(COCO_ready) + len(IN_ready) + len(Scene_ready))


list_of_files = (COCO_files, IN_files, Scene_files)


# In[ ]:


move_path = r'C:\Users\camle\Documents\MRI
Project\BOLD5000_Stimuli\Scene_Stimuli\repeats'

for root, subFolders, files in os.walk(root_dir):
  for file in files:
    for rep in rep_list:
      if file == rep:
        subFolder = os.path.join(move_path, file[:5])
        if not os.path.isdir(subFolder):
          os.makedirs(subFolder)
        shutil.move(os.path.join(root, file), move_path)#


# Now that the file lists are ready, we can randomly pull from each of the folders and create
our training and testing sets

# In[ ]:


##Commented out so I don't accidentaly run it again

dirs = [x[0] for x in os.walk(root_dir)]
files = [x[2] for x in os.walk(root_dir)]
print(dirs)
seed_list = [1337, 6700, 1640, 8218, 3074, 9510, 6358, 2934, 8799, 7181] #randomly
generated seeds for split_folders
for j in seed_list:
  for d in dirs:
    split_folders.ratio(d, output = train_path + "_" + str(j), ratio = (.7,.3), seed = j)

for j in seed_list:
  dirs = [x[0] for x in os.walk(train_path + "_" + str(j))]
  other = [x[1] for x in os.walk(train_path + "_" + str(j))]
  print(dirs)
  print(other)
```

```
for d in dirs:
  files = os.listdir(d)
  for file_name in files:
     #print(file_name)
     for rep in list_of_files:
        print(file_name)
        print(rep)
        if file_name == rep:
           full_file_name = os.path.join(d, file_name)

           if os.path.isfile(full_file_name):
              shutil.move(file_name, r"C:\\Users\\camle\\Documents\\image_test" )



# In[ ]:


#moving new files around for training

for j in seed_list:
  dirs = [x[0] for x in os.walk(train_path + "_" + str(j))]

#End of Cameron's code


# # Create train/test y martix

# Create training result matrix/y matrix

# In[10]:


#y matrix is basicly the average roi
#but we need to only select data for train/test images and we need to reorder them
#name_fit serves this fuction
def name_fit(csi_pre, roi, names):
   csi_matrix = pd.concat([pd.Series(csi_pre, name = 'image_label'), roi], axis = 1)
   csi_matrix = csi_matrix[csi_matrix['image_label'].isin(names)]
   csi_matrix.set_index('image_label',inplace = True)
   csi_matrix = csi_matrix.reindex(names)
   return csi_matrix

#we are using the training images we created with seed number 1337
```

```
train_names = glob.glob1(f"C:/MRI-Project/training_sets/train_1337/train/","*")

csi1_train_ymatrix = name_fit(csi1_pre,subject_average['csi1'], train_names)
csi2_train_ymatrix = name_fit(csi2_pre,subject_average['csi2'], train_names)
csi3_train_ymatrix = name_fit(csi2_pre,subject_average['csi3'], train_names)
csi4_train_ymatrix = name_fit(csi2_pre,subject_average['csi4'], train_names)

#We are going to use the average roi level of the first three patient FOR NOW
train_ymatrix = (csi1_train_ymatrix+csi2_train_ymatrix+csi3_train_ymatrix)/3
train_ymatrix = train_ymatrix.reset_index(drop = True)


# Create test y matirx

# In[11]:


#we are using the test images we created with seed number 1337
test_names = glob.glob1(f"C:/MRI-Project/training_sets/train_1337/val/","*")

csi1_test_ymatrix = name_fit(csi1_pre,subject_average['csi1'], test_names)
csi2_test_ymatrix = name_fit(csi2_pre,subject_average['csi2'], test_names)
csi3_test_ymatrix = name_fit(csi2_pre,subject_average['csi3'], test_names)

test_ymatrix = (csi1_test_ymatrix+csi2_test_ymatrix+csi3_test_ymatrix)/3
test_ymatrix = test_ymatrix.reset_index(drop = True)


# # Create train/test x martix

# Image tranformation

# In[12]:


transform = transforms.Compose([
   transforms.Resize(224),
   transforms.CenterCrop(224),
   transforms.ToTensor(),
   transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])


# create train x_matrix

# In[28]:
```

```
images = []
batch_t = []
for f in glob.iglob(f"C:/MRI-Project/training_sets/train_1337/train/*"):
   images.append(Image.open(f))

for i in range(len(images)):
   batch_t.append(torch.unsqueeze(transform(images[i]), 0))


# In[71]:


#stacking all five convolutional layer
#this is a limitation! NOT ENOUGH MEMORY for ALL FIVE LAYERS
tensor_list = []
for i in range(len(batch_t)):
   layer1 = model1(batch_t[i]).flatten()
   #layer2 = model2(batch_t[i]).flatten()
   #layer3 = model3(batch_t[i]).flatten()
   #layer4 = model4(batch_t[i]).flatten()
   #layer5 = model5(batch_t[i]).flatten()
   #row = torch.cat([layer1,layer2,layer3,layer4,layer5], dim=0)
   tensor_list.append(layer1)


# In[73]:


train_xmatrix = torch.stack(tensor_list).numpy()


# create test x_matrix

# In[74]:


images = []
batch_t = []
for f in glob.iglob(f"C:/MRI-Project/training_sets/train_1337/val/*"):
   images.append(Image.open(f))

for i in range(len(images)):
   batch_t.append(torch.unsqueeze(transform(images[i]), 0))
```

```
# In[77]:


tensor_list = []
for i in range(len(batch_t)):
   layer1 = model1(batch_t[i]).flatten()
   #layer2 = model2(batch_t[i]).flatten()
   #layer3 = model3(batch_t[i]).flatten()
   #layer4 = model4(batch_t[i]).flatten()
   #layer5 = model5(batch_t[i]).flatten()
   #row = torch.cat([layer1,layer2,layer3,layer4,layer5], dim=0)
   tensor_list.append(layer1)


# In[78]:


test_xmatrix = torch.stack(tensor_list).numpy()


# In[79]:


#we remove not useful variables to save some ram
del globals()['tensor_list']
del globals()['images']
del globals()['batch_t']


# # the x_matrix is too large, we need to do pca

# In[80]:


#We standardize data
scaler = StandardScaler()
scaler.fit(train_xmatrix)
train_xmatrix = scaler.transform(train_xmatrix)


# In[81]:


test_xmatrix = scaler.transform(test_xmatrix)
```

```
# In[82]:


from sklearn.decomposition import PCA
# Make an instance of the Model
pca = PCA(.95)


# In[83]:


pca.fit(train_xmatrix)


# In[86]:


train_xmatrix = pca.transform(train_xmatrix)


# In[89]:


test_xmatrix = pca.transform(test_xmatrix)


# # Elastic Net

# Try cross validation to determine best alpha and l1 ratio values

# In[14]:


#Cross Validation fitting
regrCV = MultiTaskElasticNetCV(cv=10, l1_ratio = [.1, .5, .7, .9, .95, .99, 1])
regrCV.fit(train_xmatrix, train_ymatrix)


# In[31]:


#We plot the coefficients of our ten models, want to discover a pattern a their variation
roi_list = ['LHPPA', 'RHLOC', 'LHLOC', 'RHEarlyVis', 'RHRSC', 'RHOPA', 'RHPPA',
'LHEarlyVis', 'LHRSC', 'LHOPA']
fig = plt.figure(figsize=(20,10))
```

```
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)

ax1.plot(regrCV.coef_[0], label = 'LHPPA')
ax1.plot(regrCV.coef_[1], label = 'RHLOC')
ax1.plot(regrCV.coef_[2], label = 'LHLOC')
ax1.plot(regrCV.coef_[3], label = 'RHEarlyVis')
ax1.plot(regrCV.coef_[4], label = 'RHRSC')
ax1.set_xlim(0,100)
ax1.legend(loc=1)

ax2.plot(regrCV.coef_[5], label = 'RHOPA')
ax2.plot(regrCV.coef_[6], label = 'RHPPA')
ax2.plot(regrCV.coef_[7], label = 'LHEarlyVis')
ax2.plot(regrCV.coef_[8], label = 'LHRSC')
ax2.plot(regrCV.coef_[9], label = 'LHOPA')
ax2.set_xlim(0,100)
ax2.legend(loc=1)

plt.show()


# In[36]:


#find the number of non-zero coefficients of each model
po_co = []
for i,j in zip(range(0,10),roi_list):
   number = len(regrCV.coef_[i][regrCV.coef_[i]!=0])
   print(j, number)
   po_co.append(number)


# In[37]:


#discover the value of alpha and l1 ratio
print(regrCV.alpha_)
print(regrCV.l1_ratio_)


# In[51]:


#find training and testing mse
yhat_train = regrCV.predict(train_xmatrix)
```

```
train_mse = mean_squared_error(train_ymatrix, yhat_train)
print(train_mse)
yhat_test = regrCV.predict(test_xmatrix)
test_mse = mean_squared_error(test_ymatrix, yhat_test)
print(test_mse)


# In[52]:


#model r score of test images
score = regrCV.score(test_xmatrix, test_ymatrix)
print(score)


# In[53]:


#correlations of ytrue vs yhat
ytrue_yhat_co = test_ymatrix.reset_index(drop =
True).corrwith(pd.DataFrame(yhat_test,columns = roi_list),axis = 0)
ytrue_yhat_co


# In[54]:


labels = roi_list
x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars

fig, ax = plt.subplots(figsize=(10, 5))

ax.bar(x, ytrue_yhat_co, width)
ax.set_xticks(x)
ax.set_xticklabels(labels)

plt.show()


# # The following chunk of code are for repeated images
#

# In[15]:
```

```
#This cell only need to run once to create rep_images folder
rep_images = []
for f in glob.iglob(r"C:\MRI-Project\total_images\*"):
   if f[28:] in rep_list:
      Image.open(f).save('C:/MRI-Project/rep_images/'+f[28:])


# In[42]:


#load repeated images names
rep_names = glob.glob1(f"C:/MRI-Project/rep_images/","*")


# Creat set 1 and set 2

# In[44]:


def rep_subset(rep):
   rep_sub = []
   for alist in rep:
      rep_sub.append(alist[0:2])
   return rep_sub

#csi1_rep_pairs were created when we first setting up our project
csi1_rep_subset = rep_subset(csi1_rep_pairs)
csi2_rep_subset = rep_subset(csi2_rep_pairs)
csi3_rep_subset = rep_subset(csi3_rep_pairs)


def set1_set2(rep_subset):
   set1 = []
   set2 = []
   new_list = rep_subset.copy()
   for alist in new_list:
      one = random.choice(alist)
      alist.remove(one)
      set1.append(one)
      set2.append(alist[0])

   df = pd.DataFrame(set1, index = rep_list)
   df = df.reindex(rep_names)
   set1 = list(df[0])

   df = pd.DataFrame(set2, index = rep_list)
```

```
    df = df.reindex(rep_names)
    set2 = list(df[0])

    return set1, set2

set1_1, set2_1 = set1_set2(csi1_rep_subset)
set1_2, set2_2 = set1_set2(csi2_rep_subset)
set1_3, set2_3 = set1_set2(csi3_rep_subset)


# In[45]:


#Need to use average response
set1_matrix = (subject_average['csi1'].iloc[set1_1].reset_index(drop = True)+
            subject_average['csi2'].iloc[set1_2].reset_index(drop = True)+
             subject_average['csi3'].iloc[set1_3].reset_index(drop = True))/3
set2_matrix = (subject_average['csi1'].iloc[set2_1].reset_index(drop = True)+
            subject_average['csi2'].iloc[set2_2].reset_index(drop = True)+
             subject_average['csi3'].iloc[set2_3].reset_index(drop = True))/3


# In[98]:


images = []
batch_t = []
for f in glob.iglob(r"C:\MRI-Project\rep_images\*"):
    images.append(Image.open(f))

for i in range(len(images)):
    batch_t.append(torch.unsqueeze(transform(images[i]), 0))

tensor_list = []
for i in range(len(batch_t)):
    layer1 = model1(batch_t[i]).flatten()
    #layer2 = model2(batch_t[i]).flatten()
    #layer3 = model3(batch_t[i]).flatten()
    #layer4 = model4(batch_t[i]).flatten()
    #layer5 = model5(batch_t[i]).flatten()
    #row = torch.cat([layer1,layer2,layer3,layer4,layer5], dim=0)
    tensor_list.append(layer1)

test_rep_xmatrix = torch.stack(tensor_list).numpy()
```

```
# In[99]:


del globals()['tensor_list']
del globals()['images']
del globals()['batch_t']


# In[100]:


test_rep_xmatrix = scaler.transform(test_rep_xmatrix)


# In[101]:


test_rep_xmatrix = pca.transform(test_rep_xmatrix)


# In[55]:


#correlations of set1 vs set2
set1_set2_co = set1_matrix.corrwith(set2_matrix,axis = 0)
set1_set2_co


# In[56]:


#plotting correlations of set1 vs set2
labels = roi_list
x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars

fig, ax = plt.subplots(figsize=(10, 5))

ax.bar(x, set1_set2_co, width)
ax.set_xticks(x)
ax.set_xticklabels(labels)

plt.show()


# In[57]:
```

```
#correlations of set 1 vs predicted set 1
yhat_rep_test = regrCV.predict(test_rep_xmatrix)
yhat_rep_test_df = pd.DataFrame(yhat_rep_test,columns = roi_list)

set1_pre_set1_co = set1_matrix.corrwith(yhat_rep_test_df,axis = 0)
set1_pre_set1_co

# sse = mean_squared_error(set1_matrix.iloc[:,0], set2_matrix.iloc[:,0])*113
# sst = sum((set2_matrix.iloc[:,0]-set2_matrix.iloc[:,0].mean())**2)
# print(1-sse/sst)


# In[50]:


#plotting correlations of set 1 vs predicted set 1
labels = roi_list
x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars

fig, ax = plt.subplots(figsize=(10, 5))

ax.bar(x, set1_pre_set1_co, width)
ax.set_xticks(x)
ax.set_xticklabels(labels)

plt.show()


# In[58]:


#correlations ratio
ratio = set1_matrix.corrwith(yhat_rep_test_df,axis =
0)/set1_matrix.corrwith(set2_matrix,axis = 0)
ratio


# In[59]:


#plotting correlations ratio
labels = roi_list
x = np.arange(len(labels))  # the label locations
```

```
width = 0.35  # the width of the bars

fig, ax = plt.subplots(figsize=(10, 5))

ax.bar(x, ratio, width)
ax.set_xticks(x)
ax.set_xticklabels(labels)

plt.show()
```