Clinical Brain Imaging Multi-class Classification and Interpretation – CNN Model and Layer-wise Relevance Propagation Analysis

Yijian Pang (yp2496@columbia.edu)
Fulei Zhang (fz2273@columbia.edu)

November 19, 2019

Abstract

This report covered some deep learning applications to clinical brain imaging, from training CNN models to interpreting and visualizing the results. We configured cloud computing machine to utilize all of the ∼10,000 MRI images to build three binary classification models (AD/CN, AD/MCI, MCI/CN) and one six-class classification model (CN/EMCI/MCI/LMCI/AD/SMC). We have achieved the same level of top accuracy ratios for binary classification as the start-of-art (>96%). The accuracy ratio of our six-class models reaches 77.11%. In this project, there are several challenges that we have solved: (1) generate TFRecords and read them into TensorFlow backend; (2) train, save and read a CNN model using multi-GPUs; (3) implement layer-wise relevance propagation to brain features resulting in our predictions. We also compared the results of two different LRP methods. Our results will provide some insights for clinicians in the field of Alzheimer's disease.

Key Words: Deep Learning, Convolutional Neural Networks, Layer-wise Relevance Propagation

1 Introduction and Objects

Alzheimer's disease (AD) is an irreversible and progressive brain disorder that destroys memory and other important mental functions eventually. AD is ranked as the sixth leading cause of death in the U.S. so it is meaningful to accurately identify AD progression for the patients.

Our project[1] has two main goals. Firstly, we seek to predict the risk for Alzheimer's disease progression based on the Magnetic resonance imaging (MRI) scans using deep learning models such as Convolutional Neural Networks (CNNs). Secondly, we try to provide information about underlying brain features resulting in the decision from deep learning models by state-of-the-art Deep Learning interpretation methods such as Layer-wise Relevance Propagation (LRP) method. Figure 1 illustrates the framework of our project.

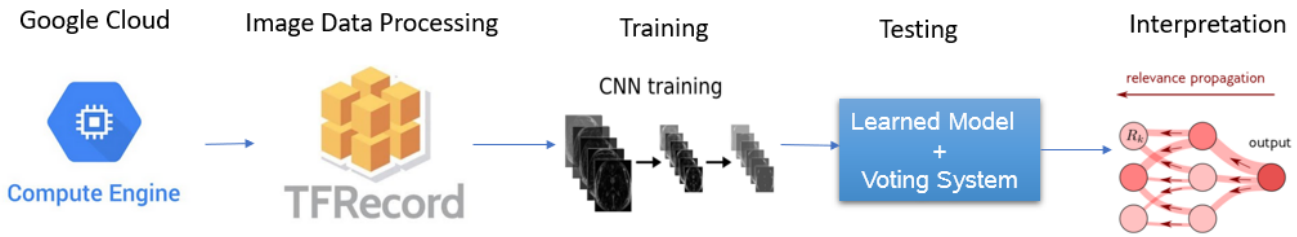

Figure 1: Project Framework

2 Google Cloud Configuration

Google Cloud is a user friendly platform for cloud computing. There are mainly two methods to configure the cloud computing engine. The first method is to create a Virtue Machine (VM) instance where only the operating

[1]Our Codes and Results can be found here. (Login with Lion Mail to access them)

system is set automatically. `TensorFlow` and Jupyter Notebook needs to be configured manually on the virtual machine. Google Cloud shell is also needed to be setup on local machine and connect to the virtual machine for file uploading or downloading. The second method is to create a notebook instance on Google Cloud AI platform. It is much easier for instant use. However, the cons for the second way is that the versions of Python and `TensorFlow` are set by Google Cloud platform. Python 3.5 is the defaulted version for Google Cloud AI platform notebook instance, whereas the first method gives more freedom on setting up the versions and craft the machine for other needs. We implemented the first method at the beginning given that we were not clear whether the defaulted version worked or not. We then moved to the second method when we found it was compatible and easy for instant use. The second method is recommended if specific versions of Python or `TensorFlow` are not required.

For the first method, it is hard to describe in a few paragraphs. We followed the materials and codes provided by Professor Manuel Balsera. The references are provided on the shared folder. (`test_gpu.py` file is used to test if CUDA framework for GPU is successfully installed.)

We will show how to configure the Google Cloud machine using the second method step by step. Google Cloud Platform has a restriction on the number of GPUs per user can use by default. A request is needed to increase the number of GPUs available.

- To increase the quota, firstly go to 'IAM & admin' and click on 'Quotas'.

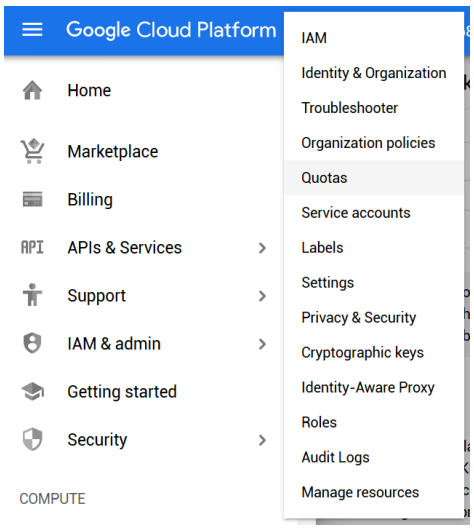

Figure 2: Google Cloud Platform Interface

First, increase the quotas for GPUs in all regions, which is the total number of GPUs available to use. Select the metrics as in Figure 3 and click on 'Edit Quotas'. Then, increase the number of a specific type of GPUs. For example, I select 'NVIDIA K80 GPUs' in us-east1 and edit the quota again. The change will usually takes effect in a few hours.

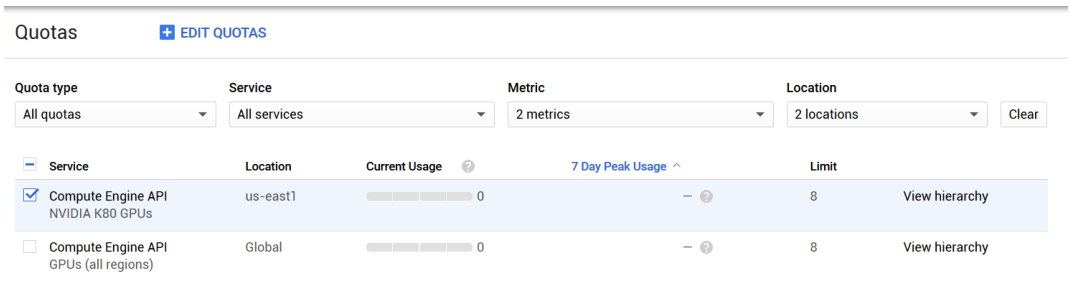

Figure 3: Google Cloud Quotas Interface

- To create a notebook instance, click the 'AI Platform' and select 'Notebooks'.

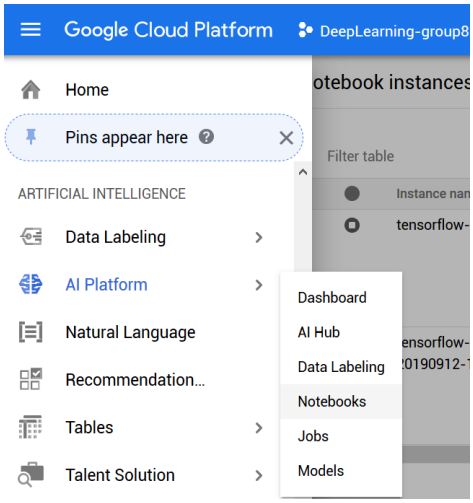

Figure 4: Google Cloud - AI Platform

Click 'New Instance' and 'Customize Instance'. Then select region and zone. Usually the defaulted choice in the U.S. will be OK, sometimes when the resources in one region are not available, try to select other regions or zones in the U.S.

- Next step is to customize the environment, number of CPUs and GPUs, and boot disk size. Based on our own experience, we selected 2000GB for the boot disk. Since the size of all 3D MRI images are around 700G and we need to leave as the same size as it for the TFRecords generated from original 3D images.

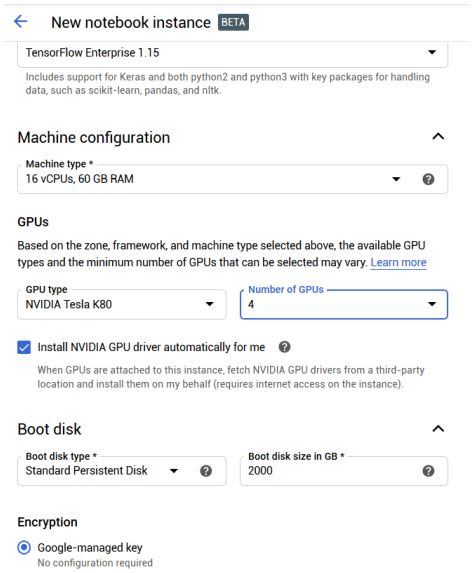

Figure 5: Google Cloud Creating New Notebook Instance Interface

Once the instance is created, start the instance and click 'Open JupyterLab'. This is the coding environment we use to write codes and train models.

3 Data Processing

3.1 Dataset Overview

For this project, we have access to the following database: ADNI (Alzheimer's Disease Neuroimaging Initiative). ADNI has around 10,000 brain structure MRI and their clinical phenotype data. Our MRI data has been pre-processed using standard brain imaging analysis pipeline (denoised, bias corrected, and registered to the standard space). We are also provided with the labels for the data. The data are in six classes: Cognitive Normal (CN), Early MCI (EMCI), Mild Cognitive Impairment (MCI), Late MCI (LMCI), Alzheimer's Disease (AD), and Subjective Memory Complaint (SMC).

The count of images per class is shown in Table 1.

Table 1: Scan Counts Classified by Class

Class	Count
CN	3031
EMCI	1785
MCI	2704
LMCI	866
AD	1241
SMC	343

As you can see from Table 1, our data set are unbalanced. Most classification algorithms are sensitive to imbalance in the predictor classes. In order to get better performance, we need to balance the dataset before training the model. We will discuss how we balance the data in section 3.5.

3.2 Download Images Data to VM

The image data is stored on the Google Cloud Storage. We used the credential to connect to the Google Cloud Storage stored in the bucket. Then every file can be downloaded to the VM with its name specified. We provide the credential file and code which can be used directly to download the data. The time estimation for downloading all the MRI images to VM is about 6 to 7 hours.

List all the file names in the bucket

```
1 blob_names = [blob.name for blob in bucket.list_blobs()]
```

```
1 blob_names
```

```
['ADNI_t1_list_with_fsstatus_20190111.csv',
 'T1_brain.tar',
 'data/',
 'data/002_S_0295_S110476-T1_T1_brain_mni305.nii',
 'data/002_S_0295_S13408_T1_brain_mni305.nii',
 'data/002_S_0295_S150055-T1_T1_brain_mni305.nii',
 'data/002_S_0295_S21856_T1_brain_mni305.nii',
 'data/002_S_0295_S32678_T1_brain_mni305.nii',
 'data/002_S_0295_S54061_T1_brain_mni305.nii',
 'data/002_S_0295_S67612_T1_brain_mni305.nii',
 'data/002_S_0295_S84944_T1_brain_mni305.nii',
 'data/002_S_0413_S111989-T1_T1_brain_mni305.nii',
 'data/002_S_0413_S111992-T1_T1_brain_mni305.nii',
 'data/002_S_0413_S13893_T1_brain_mni305.nii',
 'data/002_S_0413_S14781-T1_T1_brain_mni305.nii',
 'data/002_S_0413_S150697-T1_T1_brain_mni305.nii',
 'data/002_S_0413_S189127-T1_T1_brain_mni305.nii',
 'data/002_S_0413_S217526-T1_T1_brain_mni305.nii',
 'data/002_S_0413_S22557_T1_brain_mni305.nii',
```

Figure 6: Data Filenames Sample

3.3 Image Visualization

`Nibabel` and `SimpleITK` are libraries under Python to read `nii` files. Both of them can read a `nii` 3D image into a 256*256*256 array. `SimpleITK` provides more processing functions for medical images.[1]

Figure 7 shows three slices of one 3D image aligning with x, y and z axis separately. Each row in Figure 8 is a sequence of slices along x/y/z axis. This visualization helps us to select the best slice which contains the most information of the brain.

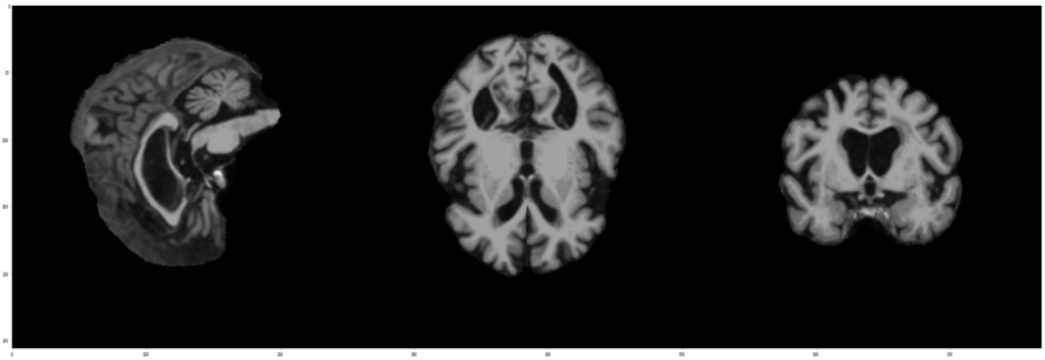

Figure 7: Sample Image Visualization 1

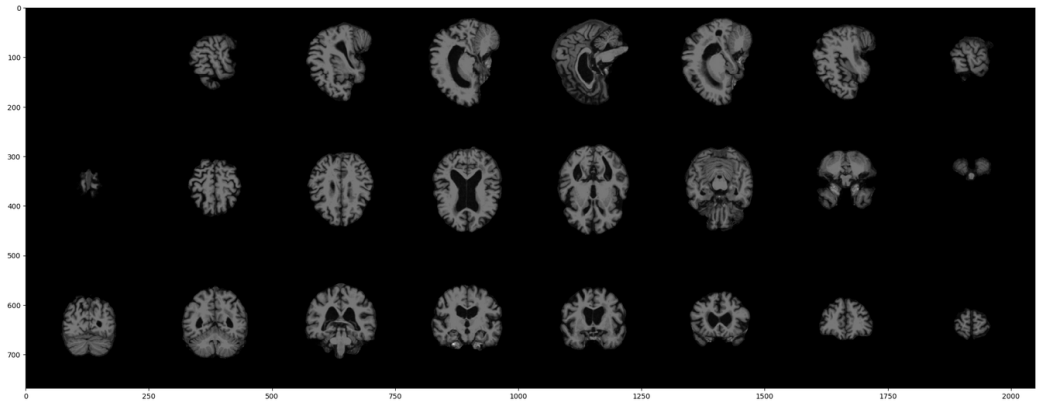

Figure 8: Sample Image Visualization 2

3.4 Generate TFRecords

As we know, one of the biggest challenges for the project is that the size of 3D MRI data is so large that it is hard to read all the image data into RAM at one time when training the model. TFRecord is a type of file created to solve the problem. It enables `TensorFlow` to read data sequentially when needed within the deep learning model.

The process of creating a TFRecord is basically saving the image array, label and its name as one feature and iterate through all images. The time needed to create a TFRecord varies based on the number of images and sizes of each image. It took about 30 minutes to create a TFRecord of 3000 256*256 images and about 2 to 3 hours for only 300 256*256*256 images. We used a machine with 360GB memory and 96vCPU.

To avoid utilizing too much memory space when reading images, we created three TFRecords for three axes and six classes separately. When randomly splitting training set and test set, we use seed in the shuffle function to make sure that we use the same training and test images for three axes.

3.5 Dataset Processing before Training

Before starting to create TFRecords for the data, we sliced the data from the middle of each axis to convert each 3D MRI image of size 256*256*256 to three 2D images of size 256*256*1. (We also tried to slice diagonally in three directions, however the test result was not as good as slicing align with the axis.)

As discussed in section 3.1, our dataset is unbalanced. Some common ways to deal with unbalanced datasets are oversampling minority classes and undersampling majority classes. We chose oversampling way to fully use our original data. Before doing oversampling, we split our data to training sets and test sets with the ratio of 8:2 to avoid the exact same data to be present in both the train the test sets, which may cause overfitting and poor generalization to the test set.

We noted that EMCI, LMCI, AD and SMC are minority classes in our dataset, so we duplicated original EMCI, LMCI, AD and SMC train sets by 1, 2, 2, 8 time(s) separately to create a more balanced dataset. The numbers of elements in our balanced train set and test set are shown in Table 2.

Table 2: Balanced Training and Test Set Counts Classified by Class

Class	Balanced Training Set Count	Test Set Count
CN	2425	606
EMCI	2856	357
MCI	2163	541
LMCI	2078	173
AD	2978	248
SMC	2470	69

4 Classification and Interpretation Models

4.1 Classification Models

4.1.1 Model Introduction

Convolutional Neural Networks (CNNs) is the most popular neural network model being used for image classification problem. In this project, we used 2D CNNs to classify MRI images.[2] We did four different scenarios of classifications, three of them are binary classes (AD and CN, AD and MCI, MCI and CN) and another is 6 classes. For each scenario, we trained the data from three axis separately and then combined the results by creating a voting system to get our final decision.

4.1.2 Model Structure and Parameters

Our best performing model structure configurations are as follows:

(1) First Convolutional Layer: filters = 64, kernel size = 3, activation = 'elu'
(2) Max Pooling: pool size = 3
(3) Dropout Layer
(4) Second Convolutional Layer: filters = 16, kernel size=3, activation = 'elu'
(5) Max Pooling: pool size=3
(6) Dropout Layer
(7) Third Convolutional Layer: filters = 32, kernel size = 3, activation = 'elu'
(8) Fourth Convolutional Layer: filters = 16, kernel size = 3, activation = 'elu'
(9) Max Pooling: pool size = 3
(10) Dropout Layer
(11) Flatten Layer
(12) Fully connected Layer: Output 2/6 nodes, activation = 'sigmoid'/ 'softmax'

For each training, we set number of epochs as 150 and we stored our best model with highest train accuracy for later use. We also set learning rate as 0.001 and drop rate as 0.3 for our results.

4.1.3 Training Speed Up

To speed up our training time, we made the following improvements to our existing model.

- Use two GPUs to train the models. `keras.utils.multi_gpu_model` allows single-machine multi-GPU data parallelism.
- Set early stopping. The train loss at each epoch is monitored. After the test loss has not improved after ten epochs, training will be interrupted.

Using these methods, we successfully decreased our training time from approximately 40 minutes to 20 minutes per training.

4.1.4 Voting System

Since we trained our models on three different axes separately, our predictions were also given separately. A voting system is needed to get the final prediction. The predictions on classes are in digits and larger number will have larger probability in that class given that our last layer's activation function is sigmoid or softmax. We assume that models trained in each axis play the same role on the final decision making so we simply defined our voting system as predicting a MRI scan by choosing the largest class from the element wise sum of three different prediction logits.

4.2 Interpretation Method

CNN model itself will not provide underlying information to clinicians because of its black-box property. Layer-wise relevance propagation is a back propagation technique of interpreting deep neural networks. It provides analysis on how each pixel of the images contributes to the classification function through each layer. By leveraging this technique, we are able to bring insightful interpretations to human experts.
Suppose the classification function is $f(x)$, x is the pixel of the image, and V is the total number of pixels. $f(x)$ is a mapping from R^v to R^1. In a binary classification case, $f(x) > 0.5$, i.e. $f(x) - 0.5 > 0$ means the prediction is classified as one class. In order to analyze the influence of each pixel on $f(x)$, let S_i be the score of pixel i's contribution to $f(x)$, then $f(x)$ can be decomposed by: $f(x) \approx \sum_{i=1}^{V} Pi$. If $Pi > 0$, it means the pixel has a positive effect on the final classification and a negative effect when $P_i < 0$. In a multi-layer neural network, we not only have the relevance score P_i in the input layer but for each layer before the output layer. Let R_i^L be the relevance score of unit i at layer L, then the sum of scores of all units R_i^L in layer L should satisfy the conservation rule as follows: [3]

$$f(x) \approx \sum_{i \in L+1} R_i^{L+1} = \sum_{i \in L} R_i^L = ... = \sum_{i}^{V} R_i^1 \tag{1}$$

There are three possible ways of relevance propagation, including LRP-Z, LRP-Epsilon, and LRP-AlphaBeta. LRP-Epsilon is an improvement over LRP-Z method by adding an term in the denominator to avoid infinity relevance score. LRP-AlphaBeta treats the activating and inhibiting neurons separately by setting alpha and beta parameters.[4] Thus we apply both LRP-Epsilon and LRP-AlphaBeta and make a comparison between them.

According to forward propagation, we know that from layer J to K, the connected neurons satisfy the following equation, where $\sigma(x)$ is the activation function:

$$a_K = \sigma(\sum w_{JK} a_J + b_K), \tag{2}$$

Combining equations (1) and (2), we can derive the formula to calculate relevance scores backwards until the input layer in two methods mentioned above. R_j^K is the relevance score of neuron j in layer K and R_i^J is the relevance score of neuron i in layer J.

- LRP-Epsilon

$$R_i^J = \sum_{j \in K} \frac{a_i w_{ij}}{\sum_i a_i w_{ij} + \varepsilon sign(\sum_i a_i w_{ij})} R_j^K \tag{3}$$

- LRP-AlphaBeta

$$R_i^J = \sum_{j \in K} (\alpha \frac{a_i w_{ij}^+}{\sum_i a_i w_{ij}} - \beta \frac{a_i w_{ij}^-}{\sum_i a_i w_{ij}}) R_j^K \tag{4}$$

To perform LRP analysis, we used `iNNvestigate`[5] which is a newly developed library providing popular analytic algorithms for neural networks, including Gradient Saliency Map, SmoothGrad, IntegratedGradients, Deconvnet, GuidedBackprop, PatternNet, PatternAttribution, DeepTaylor, and LRP.

To implement LRP and visualize the relevance score of each pixel, we did it in a comparison way based on the model we trained. For example, to analyze the AD/CN binary classification model, we draw two images which are the average of all AD and CN images in the training set respectively. Then we overlay the LRP result from true positives and true negatives onto two averaged images separately. In this way, it is easy to compare the same region in brain and whether the region is positively or negatively related with AD or CN.

5 Results

5.1 Binary Classification Models

Tables 3 - 5 are the results from different models on test set. The accuracy ratios on y axis are the best in all three binary classification models. It is because y axis slices have more information than x or z axis slices. After conducting voting systems, the overall accuracy improves about 2% over the best axis without doing harm to precision, which means the voting system works well.

AD/CN classification model has the best result among the three models, which is consistent with other researchers. It is also reasonable since the difference between AD and CN is easier to tell. However, it is important to diagnose MCI from CN or AD from MCI at an early stage so that patients will get treatment earlier. Table 6 shows the comparisons of results with different models and images. Our models achieved an accuracy ratio of 97.97% on AD/MCI and 96.68% on MCI/CN classification, which are better compared than other researchers' results using non deep learning techniques.

Table 3: AD and CN Classification Results on Test Set

	x axis	y axis	z axis	Voting System
Accuracy	93.68%	96.25%	92.74%	98.36%
True Positive Rate	91.25%	95.87%	90.26%	99.01%
True Negative Rate	99.60%	97.18%	98.79%	95.56%
Precision	99.81%	98.81%	99.45%	98.04%
Negative Predictive Value	82.06%	90.60%	80.59%	97.53%

Table 4: AD and MCI Classification Results on Test Set

	x axis	y axis	z axis	Voting System
Accuracy	88.32%	96.45%	95.81%	97.97%
True Positive Rate	83.15%	95.00%	94.44%	97.04%
True Negative Rate	99.19%	99.60%	98.79%	99.60%
Precision	99.78%	99.81%	99.42%	99.81%
Negative Predictive Value	73.21%	90.15%	89.09%	94.27%

Table 5: MCI and CN Classification Results on Test Set

	x axis	y axis	z axis	Voting System
Accuracy	90.40%	94.85%	93.63%	96.68%
True Positive Rate	93.89%	95.21%	93.89%	98.35%
True Negative Rate	86.48%	94.44%	93.33%	94.81%
Precision	88.63%	95.06%	94.05%	95.51%
Negative Predictive Value	92.66%	94.62%	93.16%	98.08%

Table 6: Results Compared with Published Papers

Reference	Model	Image	AD/CN	AD/MCI	MCI/CN
Suk et al.[6]	SAE+SVM	MRI+PET+CSF	95.9%	N/A	85%
Suk et al.[7]	SAE+SVM	MRI+PET	95.4%	N/A	85.7%
Zhu et al.[8]	MSLF+SVM	MRI+PET+CSF	95.9%	N/A	82%
Zu et al.[9]	MTFS+SVM	MRI+PET+CSF	96%	N/A	80.3%
Liu et al.[10]	SAE+SVM	MRI+PET	91.4%	N/A	82.1%
Liu et al.[11]	MFE+SVM	MRI	93.8%	N/A	89.1%
Li et al.[12]	PCA+SVM	MRI+PET+CSF	91.4%	70.1%	77.4%
Payan et al.[13]	2D SAE	MRI	95.4%	86.8%	92.1%
Hossein-Asl et al.[14]	3D SAE	MRI	97.6%	N/A	N/A
Proposed model	**CNN**	**MRI**	**98.36%**	**97.97%**	**96.68%**

Abbreviations in the table:
CSF: cerebrospinal fluid; CAE: Convolutional Autoencoder; SAE: Sparse Autoencoder;
CNN: Convolutional Neural Networks; MSLF: Matrix-Similarity based Loss Function;
MTFS: Multi-task feature selection; MFE: Multiview Feature Extraction.

5.2 Six-Class Classification Model

As for the six-class model, we used the same architecture with only the activation function of the output layer changed to six classes. To evaluate the model, we predicted on test set and reported the results (Table 7) after voting system. True rate for class i is defined as $\frac{\text{Number of correctly predicted samples in class i}}{\text{Number of predictions in class i}}$. The results show that SMC has the highest true rate, which means all of our predictions as SMC are true SMC. On the other hand, the precision for SMC is 70.83%, which means among all true SMCs, our six-class model successfully predicted 70.83% of them. The six-class model performs well in identifying LMCI since its true rate and precision are both high. The weighted average accuracy ratio over all six classes is 77.11%.

Table 7: Six Classes Classification Results on Test Set

Class	True Rate	Precision
CN	68.81%	86.51%
EMCI	94.40%	74.23%
MCI	86.85%	76.89%
LMCI	96.53%	86.53%
AD	92.34%	72.93%
SMC	100.00%	70.83%

Below are the confusion matrices (Figures 9 - 10) of the six-class model's predictions on test set. Before normalization, MCI has the darkest color because the number of accurately predicted MCI is the largest in the testing set. The normalized confusion matrix provides a more reasonable explanation as it will take the number of scans into consideration. CN has the lightest blue now due to its 68.81% true rate. We can see that for scans predicted as CN, some of them are actually EMCI or MCI. For other samples which are true CN, the model predicts

them as EMCI or MCI, and its probability is higher than the former case. It reveals that the performance of six-class model was not as stable as binary model for the disease at an early stage. This model performs better in classifying Alzheimer's disease after MCI stage.

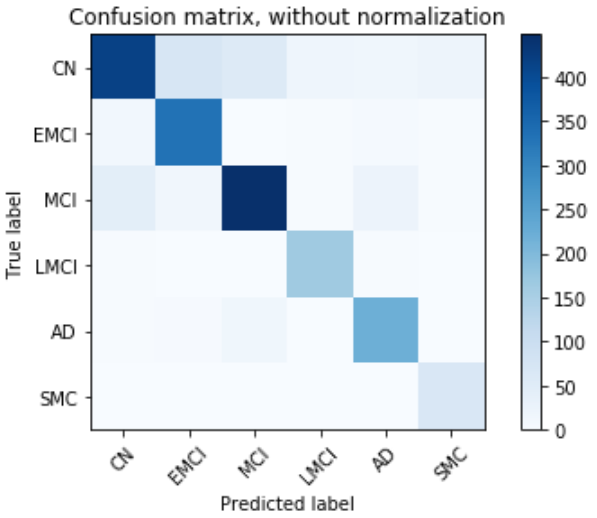

Figure 9: Confusion Matrix, Without Normalization

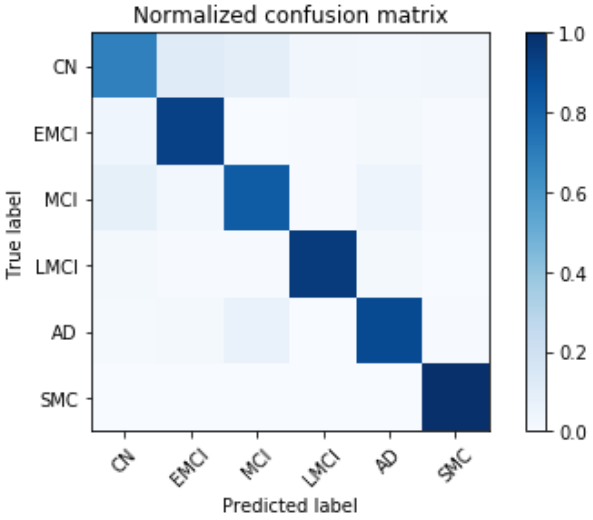

Figure 10: Normalized Confusion Matrix

5.3 LRP Interpretation

In this section, LRP-Epsilon and LRP-AlphaBeta are implemented to interpret the AD/CN classification model. Both of the two methods visualize how each pixel on MRI image contributes to classification. The results of AD/CN visualization using LRP-Epsilon are shown in Figures 11 - 13 and the results using LRP-AlphaBeta are shown in Figures 14 - 16. The red represents pixels that are positively related to the class, whereas the blue represents pixels negatively related to the class. Epsilon is set as 0.001; alpha is set as 3 and beta is set as 2.

The visualization from LRP-Epsilon method shows that the pixels inside the brain contribute the most in the classification. There is no obvious pattern of regions shown and the pixels are not concentrated enough. It is because that based on equation (2) in section 4.2, the epsilon term is likely to absorb part of the relevance score. The x-axis slice catches the most effective pixels compared to y or z axis. LRP-Epsilon visualizations for AD/MCI and MCI/CN model are shown in Appendix (section 8.2).

In comparison, LRP-AlphaBeta shows that the key to classifying AD versus CN lies on the boundary of regions in the brain. The patterns are more obvious on y and z axes. There are several small regions on the y slices of AD that are red, which contributes the most in classifying as AD.

Both methods provides insights of the same AD/CN classification model. Clinicians can utilize the results below and analyze the colored regions with expertise. These two methods can be used in complementation given that LRP-Epsilon selects pixels in the middle and LRP-AlphaBeta focuses more on the frontier.

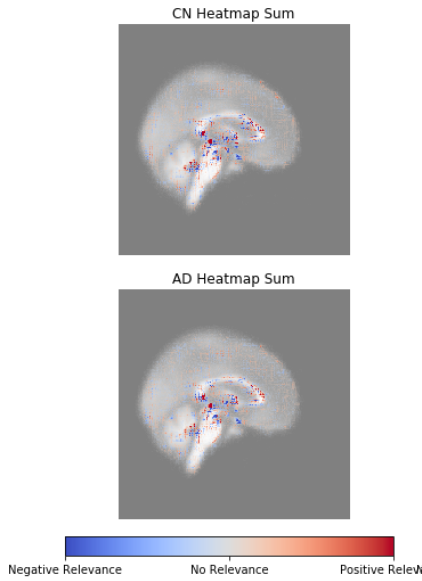

Figure 11: AD/CN Visualization using LRP-Epsilon (x axis)

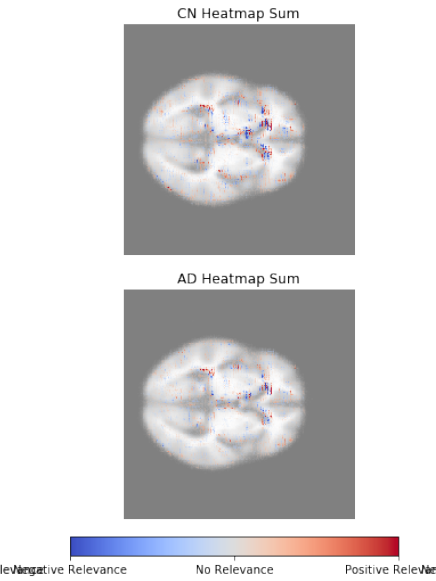

Figure 12: AD/CN Visualization using LRP-Epsilon (y axis)

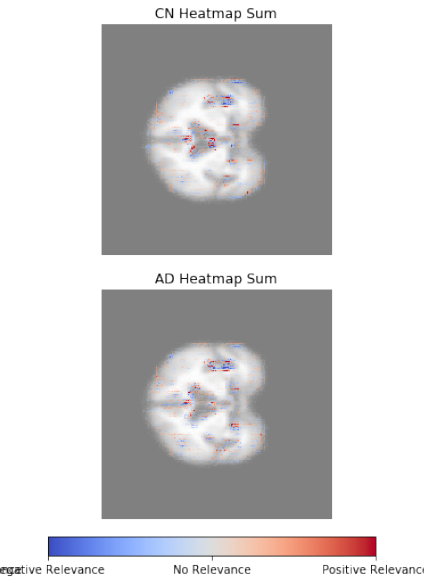

Figure 13: AD/CN Visualization using LRP-Epsilon (z axis)

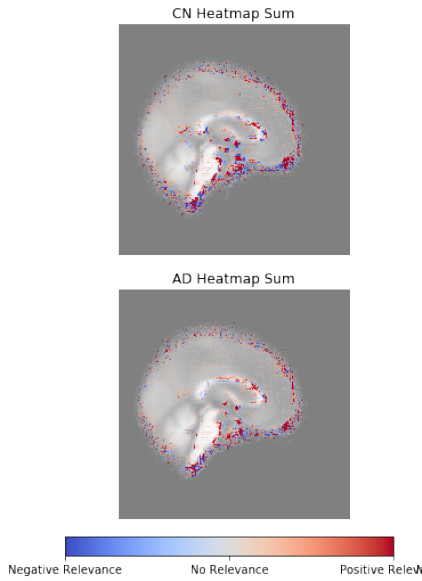

Figure 14: AD/CN Visualization using LRP-AlphaBeta (x axis)

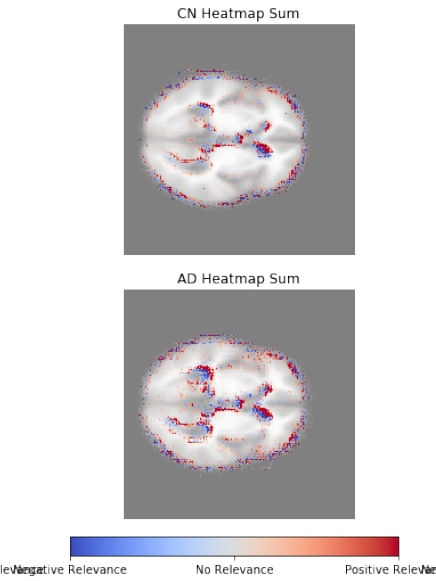

Figure 15: AD/CN Visualization using LRP-AlphaBeta (y axis)

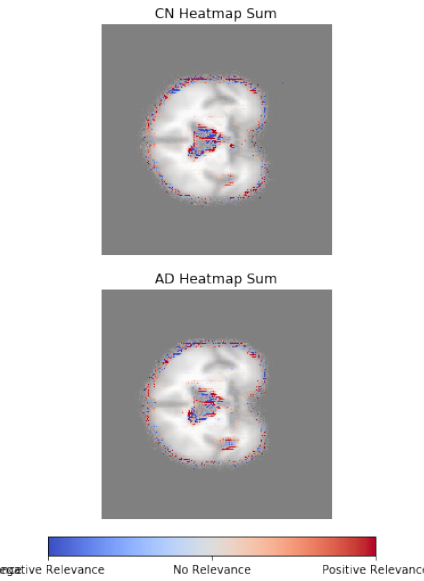

Figure 16: AD/CN Visualization using LRP-AlphaBeta (z axis)

6 Future Improvements

Based on the resources we have explored and our experience, we suggest that the following aspects deserve further research and improvements.

- **Computational Power Utilization**
  In this project, there are mainly two parts which are computation costly, one is generating TFRecords and the other is training models. We have tackled the challenge of training models using multi-gpus and all following procedures after that, such as saving and reading a multi-gpu trained model and input the weights into LRP interpretation process. However, there is still some room to improve which is using multi-cpus to generate TFRecords. It would shorten the time spent on generating TFRecords, especially when using 3D images.

- **Enhanced Voting System**
  In our model, we selected the middle slices from x, y and z axis and created a voting systems combining the three predictions. It is worthwhile to try selecting multiple slices from three axises and build a larger voting system. We have shown that the voting system is effective in improving the accuracy of binary and six-class classification model. Selecting multiple slices means utilizing more information and thus possible to increase the overall accuracy.

- **Interpretation Method**
  We have tested the result of interpretation by using LRP method. (We also tried gradient method but it was not effective.) There are other interpretation methods which also could provide insights into neural network models, such as Deconvnet, GuidedBackprop, PatternNet and Pattern Attribution.

7 Conclusion

In this paper, We applied CNNs for predicting the risk for Alzheimer's disease progression based on the Magnetic resonance imaging (MRI) scans and using LRP-Epsilon and LRP-AlphaBeta methods to uncover brain features resulting in our predictions.

We built an end-to-end system of diagnosing Alzheimer's disease. We started from configuring the google cloud virtual machine and processing MRI image data. We provided the functionalities including:
(1) Creating TFRecords
(2) Reading TFRecords into TensorFlow backend
(3) Splitting it into train and test set, balancing the training data
(4) Training and saving binary-class and six-class CNN models using multi-GPUs
(5) Prediction result analysis on test set
(6) MRI visualization and LRP interpretation

This project is a complete implementation of leveraging deep learning techniques in diagnosing Alzheimer's disease. This task is challenging and meaningful at the same time. In conclusion, the models demonstrates the possibility of correctly predicting Alzheimer's disease progression even from early stages and classifying some brain features that have effects on the decision, where future improvements can be made.

References

[1] Z. Yaniv, B. C. Lowekamp, H. J. Johnson, and R. Beare, "Simpleitk image-analysis notebooks: a collaborative environment for education and reproducible research," *Journal of Digital Imaging*, vol. 31, no. 3, pp. 290–303, Jun 2018. [Online]. Available: https://doi.org/10.1007/s10278-017-0037-8

[2] F. Eitel, E. Soehler, J. Bellmann-Strobl, A. U. Brandt, K. Ruprecht, R. M. Giess, J. Kuchling, S. Asseyer, M. Weygandt, J.-D. Haynes, and et al., "Uncovering convolutional neural network decisions for diagnosing multiple sclerosis on conventional mri using layer-wise relevance propagation," *NeuroImage: Clinical*, vol. 24, p. 102003, 2019. [Online]. Available: http://dx.doi.org/10.1016/j.nicl.2019.102003

[3] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PLOS ONE*, vol. 10, no. 7, pp. 1–46, 07 2015. [Online]. Available: https://doi.org/10.1371/journal.pone.0130140

[4] S. Lapuschkin, A. Binder, G. Montavon, K.-R. Müller, and W. Samek, "The lrp toolbox for artificial neural networks," *Journal of Machine Learning Research*, vol. 17, no. 114, pp. 1–5, 2016. [Online]. Available: http://jmlr.org/papers/v17/15-618.html

[5] M. Alber, S. Lapuschkin, P. Seegerer, M. Hägele, K. T. Schütt, G. Montavon, W. Samek, K.-R. Müller, S. Dähne, and P.-J. Kindermans, "iNNvestigate neural networks!" *arXiv e-prints*, p. arXiv:1808.04260, Aug 2018.

[6] H.-I. Suk and D. Shen, "Deep learning-based feature representation for ad/mci classification," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2013*, K. Mori, I. Sakuma, Y. Sato, C. Barillot, and N. Navab, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 583–590.

[7] H.-I. Suk, S.-W. Lee, and D. Shen, "Hierarchical feature representation and multimodal fusion with deep learning for ad/mci diagnosis," *NeuroImage*, vol. 101, pp. 569–582, 2014.

[8] X. Zhu, H.-I. Suk, Y. Zhu, K.-H. Thung, G. Wu, and D. Shen, "Multi-view classification for identification of alzheimer's disease," in *Machine Learning in Medical Imaging*, L. Zhou, L. Wang, Q. Wang, and Y. Shi, Eds. Cham: Springer International Publishing, 2015, pp. 255–262.

[9] C. Zu, B. Jie, M. Liu, S. Chen, D. Shen, D. Zhang, and A. The Alzheimer'S Disease Neuroimaging Initiative, "Label-aligned multi-task feature learning for multimodal classification of alzheimer's disease and mild cognitive impairment," *Brain Imaging and Behavior*, 11 2015.

[10] S. Liu, S. Liu, W. Cai, H. Che, S. Pujol, R. Kikinis, D. Feng, M. Fulham, and ADNI, "Multimodal neuroimaging feature learning for multiclass diagnosis of alzheimer's disease," *IEEE Transactions on Biomedical Engineering*, vol. 62, no. 4, pp. 1132–1140, 4 2015.

[11] M. Liu, D. Zhang, E. Adeli, and D. Shen, "Inherent structure-based multiview learning with multitemplate feature representation for alzheimer's disease diagnosis," *IEEE Transactions on Biomedical Engineering*, vol. 63, no. 7, pp. 1473–1482, July 2016.

[12] F. Li, L. Tran, K. Thung, S. Ji, D. Shen, and J. Li, "A robust deep model for improved classification of ad/mci patients," *IEEE Journal of Biomedical and Health Informatics*, vol. 19, no. 5, pp. 1610–1616, 9 2015.

[13] A. Payan and G. Montana, "Predicting alzheimer's disease: a neuroimaging study with 3d convolutional neural networks," 2015.

[14] E. Hosseini-Asl, R. Keynton, and A. El-Baz, "Alzheimer's disease diagnostics by adaptation of 3d convolutional network," *2016 IEEE International Conference on Image Processing (ICIP)*, Sep 2016. [Online]. Available: http://dx.doi.org/10.1109/ICIP.2016.7532332

8 Appendix

8.1 Train/Test Performance

8.1.1 AD/CN

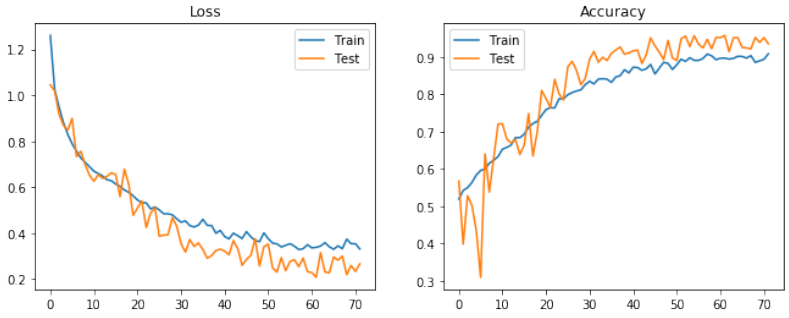

Figure 17: AD/CN Performance (x axis)

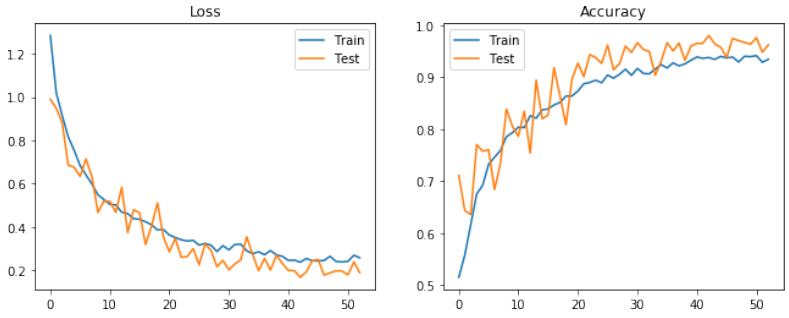

Figure 18: AD/CN Performance (y axis)

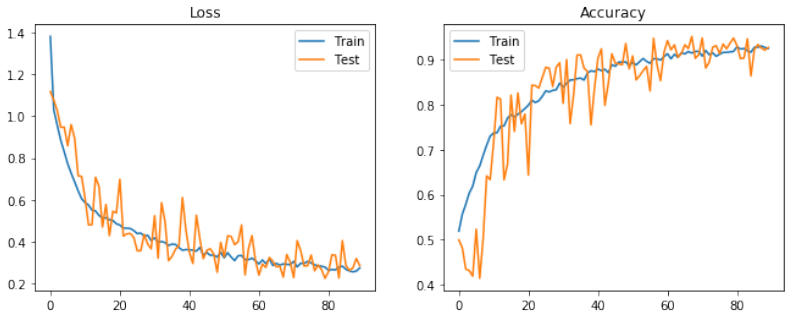

Figure 19: AD/CN Performance (z axis)

8.1.2 AD/MCI

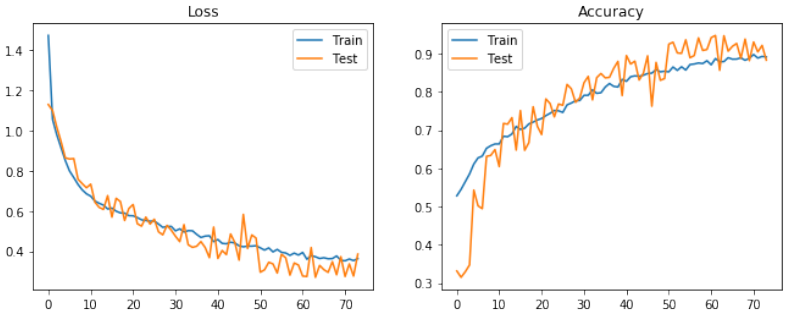

Figure 20: AD/MCI Performance (x axis)

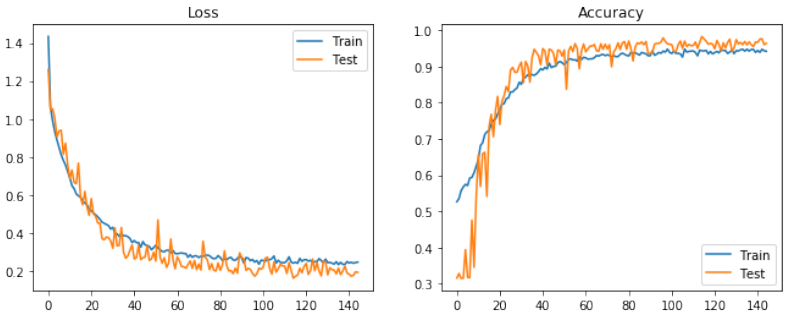

Figure 21: AD/MCI Performance (y axis)

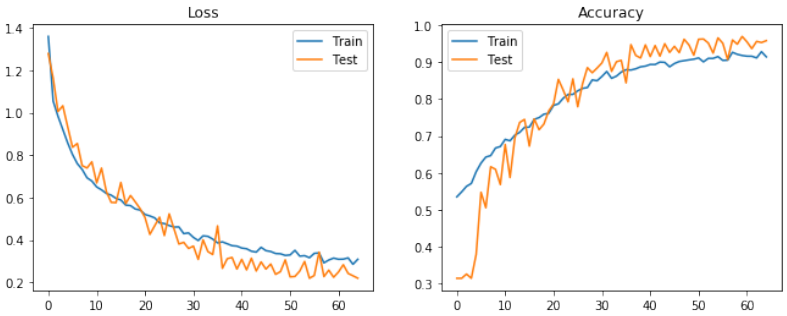

Figure 22: AD/MCI Performance (z axis)

8.1.3 MCI/CN

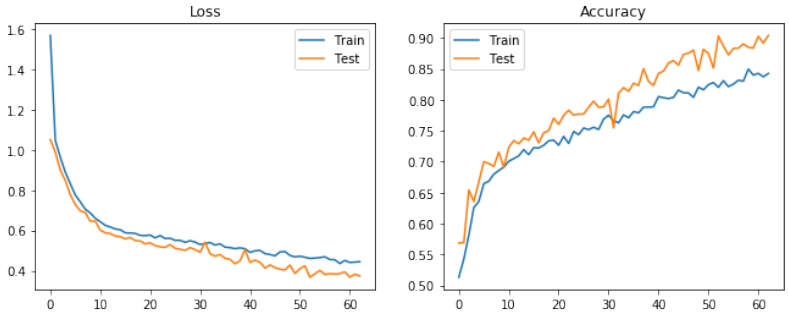

Figure 23: MCI/CN Performance (x axis)

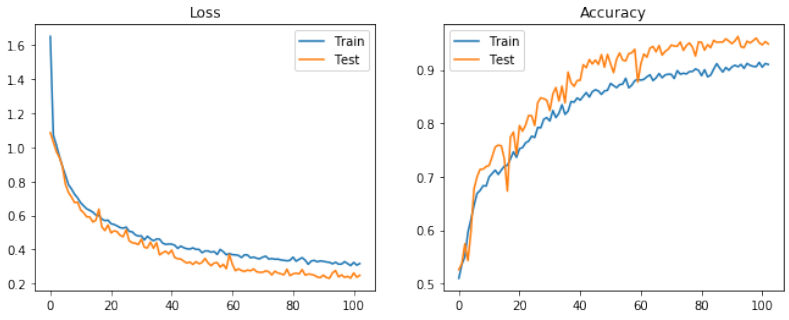

Figure 24: MCI/CN Performance (y axis)

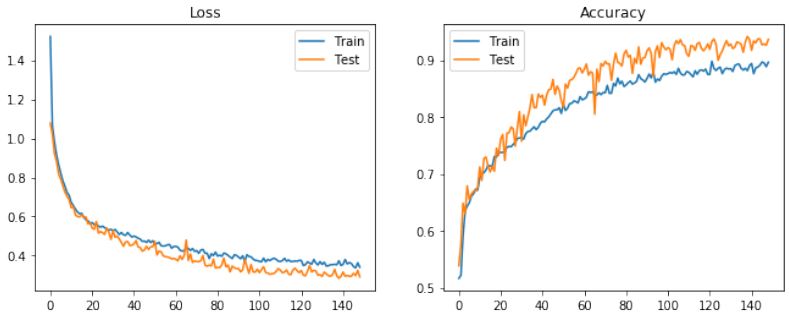

Figure 25: MCI/CN Performance (z axis)

8.1.4 6 Classes

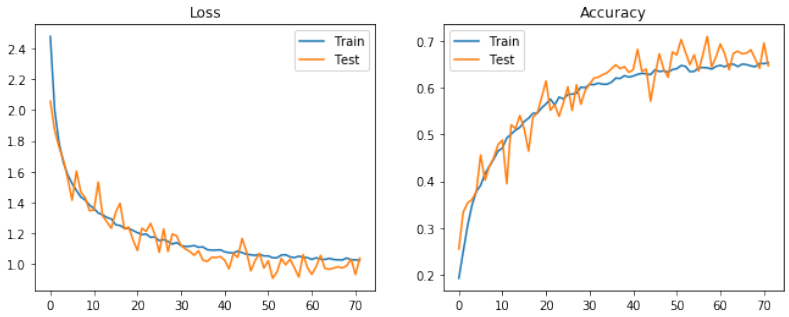

Figure 26: 6 Classes Performance (x axis)

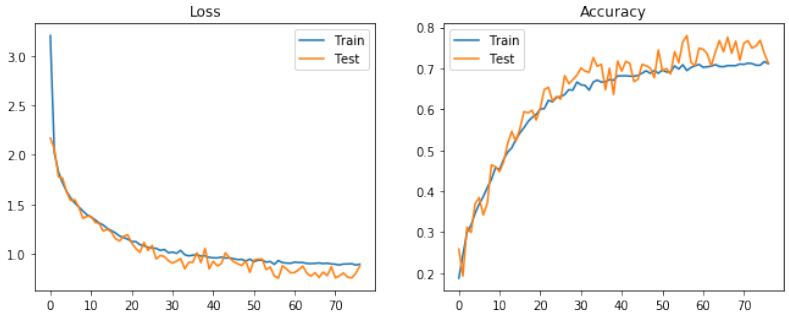

Figure 27: 6 Classes Performance (y axis)

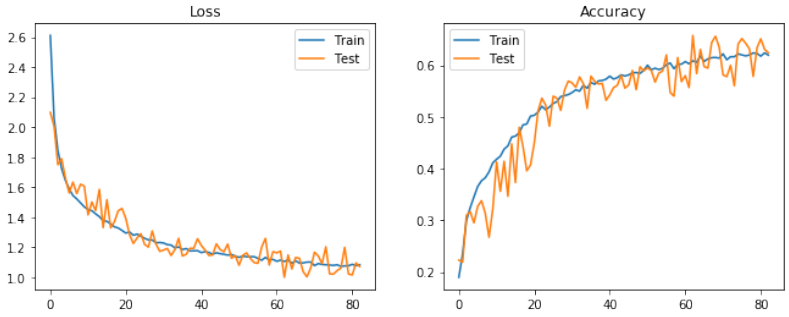

Figure 28: 6 Classes Performance (z axis)

8.2 LRP Interpretation and Visualization

8.2.1 AD/MCI

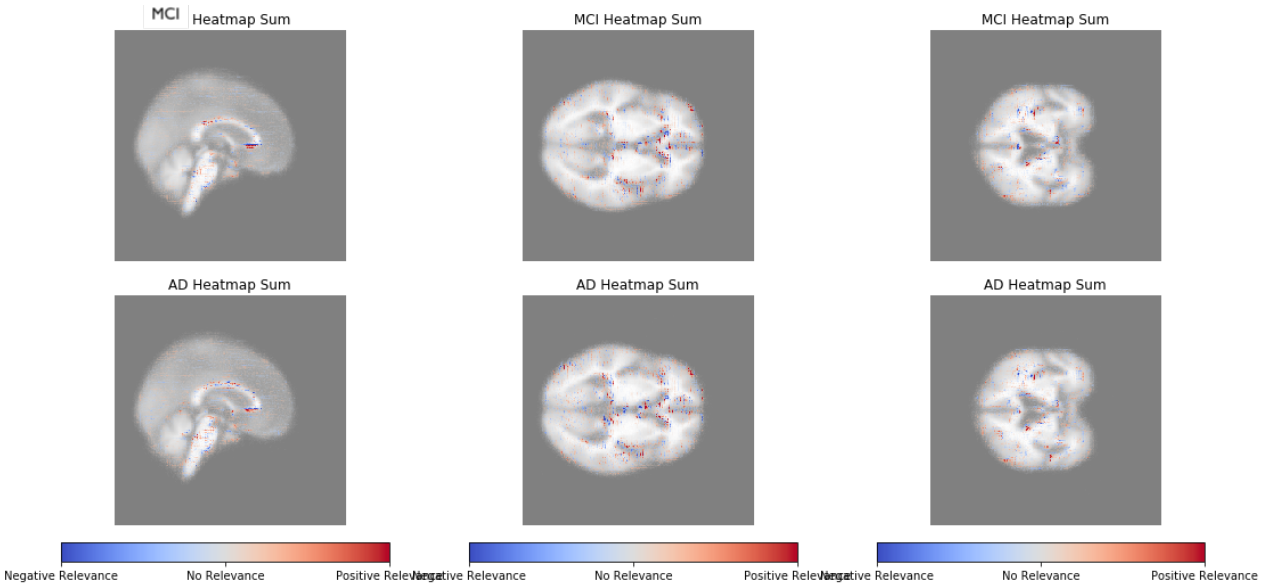

Figure 29: AD/MCI Visualization using LRP-Epsilon (x axis)

Figure 30: AD/MCI Visualization using LRP-Epsilon (y axis)

Figure 31: AD/MCI Visualization using LRP-Epsilon (z axis)

8.2.2 MCI/CN

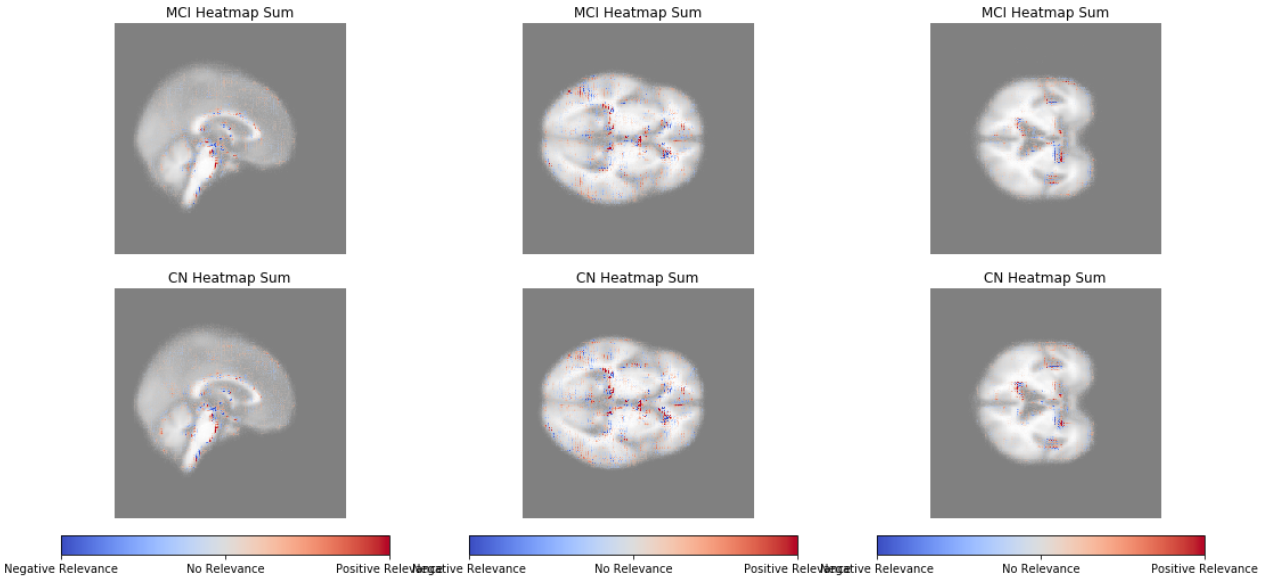

Figure 32: MCI/CN Visualization using LRP-Epsilon (x axis)

Figure 33: MCI/CN Visualization using LRP-Epsilon (y axis)

Figure 34: MCI/CN Visualization using LRP-Epsilon (z axis)