Written Report

1. Data Set

Description of the data set and description of what you are trying to do with the neural net.

The purpose of the project is to apply a multilabel classification task to brain MRI images of distinct people who faces Alzheimer and predict the category of their disease. For this purpose, we have utilized the data set given in the link below.

Link: https://www.kaggle.com/tourist55/alzheimers-dataset-4-class-of-images

In our dataset, we have 4 categories that reflect the stages of the disease. Our aim is to build a Convolutional Neural Network which is complex enough to be able to capture the given MRI images, discover its characteristic features and successfully classify the stage of the Alzheimer according to the images we will train it.

2. Exploratory Data Analysis

- Descriptive statistics for the features and labels

As we have mentioned above, there are 4 stages of the disease which are named as: Non-Demented, Mild Demented, Moderate Demented and Very Mild Demented. Images are given in .jpg format. Each of them has size (176 x 208 px). In total, there are 6400 of them. In the given dataset, they are already uploaded separately for each class and for training and testing. To have a deeper look we have the following:

- All of the data:
  - Non Demented (3200 images)
  - Very Mild Demented (2240 images)
  - Mild Demented (896 images)
  - Moderate Demented (64 images)
  - Total: 6400 jpegs

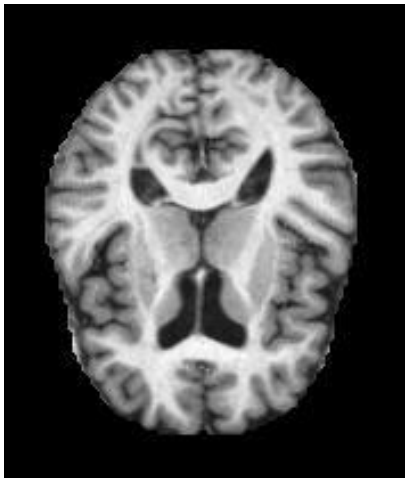
- Training Data:
  - Non Demented (2560 images)
  - Very Mild Demented (1792 images)
  - Mild Demented (717 images)
  - Moderate Demented (52 images)
  - Total: 5121 jpegs (176 x 208 px)

- Testing Data:
  - Non Demented (640 images)
  - Very Mild Demented (448 images)
  - Mild Demented (179 images)
  - Moderate Demented (12 images)
  - Total: 5121 jpegs (176 x 208 px)

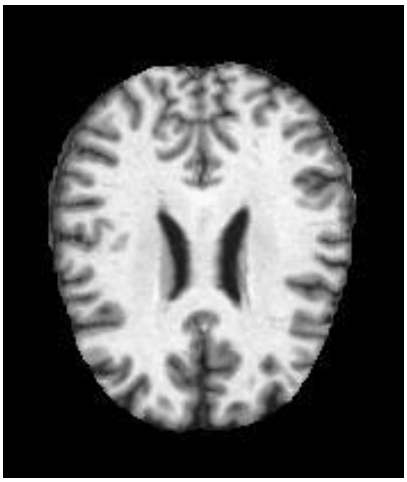
- Some visualization of the data and state why you chose the visualizations you choose

Below you can view sample MRI images of each class. This allows us to visually distinguish that the classes are actually different from each other and have their own characteristics.

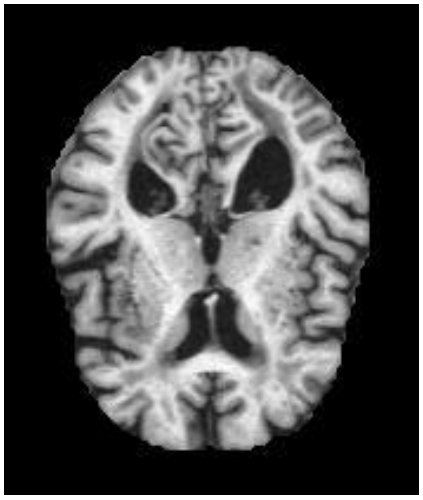
- **Non Demented - Train**



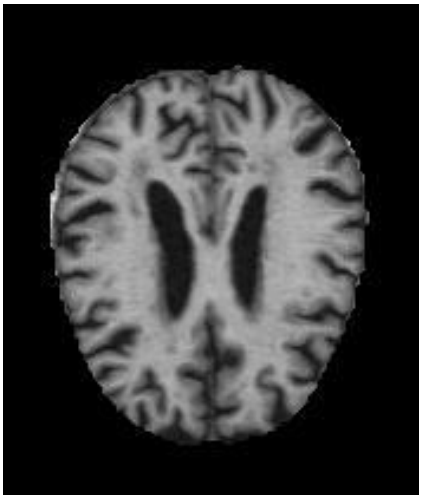
Non Demented - Test



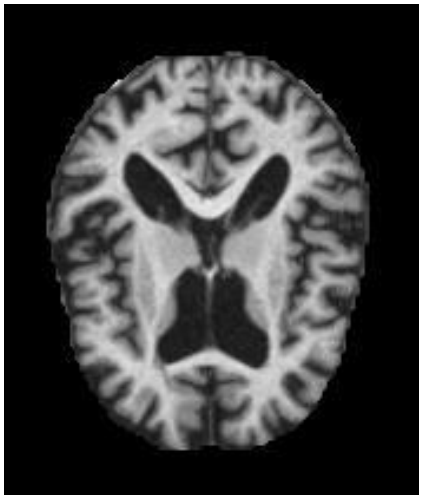
- **Mild Demented -Train**



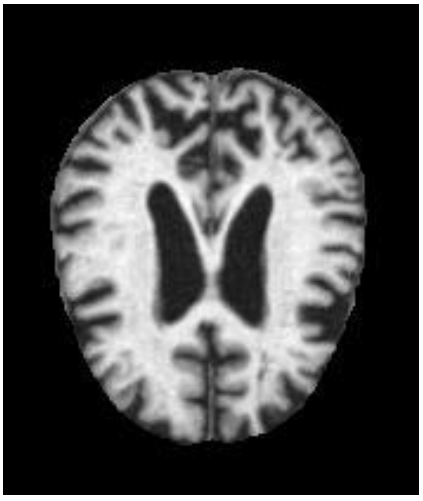
Mild Demented - Test



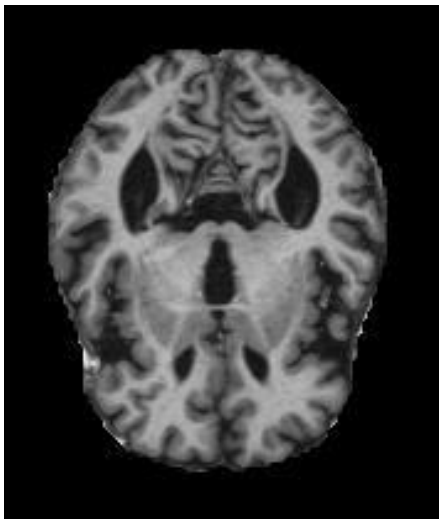
- **Moderate Demented - Train**

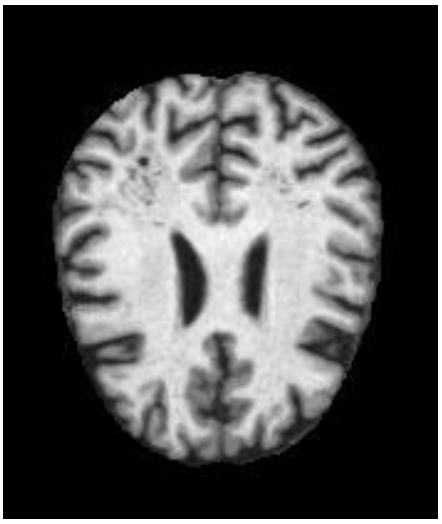


Moderate Demented - Test



- **Very Mild Demented - Train** **Very Mild Demented - Test**





From the MRI images of each class, we can clearly indicate that the train MRI images does not look so similar to the corresponding test MRI images in each class. Therefore, we initially expect to have lower test results compared to the validation results. Displaying random images for each class and for both training and testing data helps us to interpret the results we obtained after the validation and test process. Such difference between train and corresponding test data also points out this fact.

3. Data Pre-Processing

Describe any data pre-processing techniques (scaling, etc.) and why you used them.

– RGB to Grayscale:

We can observe that the images are already in grayscale format even we have displayed them with their RGB values. That points out the fact that we do not necessarily require to contain all 3 channels and we can reduce the dimensionality by converting the images in grayscale.

– Normalization:

The aim of the normalization of the data is to change the values of each feature in the dataset to a common scale without distorting differences in ranges of values. The reason we is because in the process of training our network, we're going to be multiplying (weights) and adding to (biases) these initial inputs in order to cause activations that we then update our weighs according to the loss function we chose. We would like in this process for each feature to have a similar range so that our gradients don't go out of control.

– Oversampling:

As it can be observed from the exploratory data analysis section, we have an unbalanced data set if we especially consider the moderate demented class. In order to overcome this issue, we have considered two different techniques:

- Class Weighting:

The aim of using class weighting is to penalize the misclassification made by the minority class by setting a higher-class weight and at the same time reducing the weight for the majority class. In order to achieve this, we have set the weights inversely proportional with the number of samples we have. Which yields the following weights:

- Class Weights:
    - Non Demented - **2**
    - Very Mild Demented – **3**
    - Mild Demented - **7**
    - Moderate Demented - **98**

This technique is not only useful for the minority class but also provides the balance of whole dataset.

- SMOTE (Synthetic Minority Oversampling Technique):

  This technique and its variants are also popular to have images in which all classes are in the same size of majority class. However, this approach takes most of the RAM used by Google Colab. Therefore, we could not train the model enough since session crashes.

4. Upload Data and Training/Test Split

Describe your procedure for taking the original data, and how you split it into training/testing sets, training/testing/validation sets, or used k-fold cross-validation.

The data is uploaded to a Google Drive account in a zipped format. Then it is imported to the Google Colab and the images are unzipped to the folder of the Google Colab. After that, the path that images belong are utilized to fetch each image and then these images are converted into numpy arrays.

As it is mentioned in the exploratory data analysis section, our data set is already split into training and testing data. Therefore, we are left with applying k-fold cross validation utilizing from the training data and finally apply the optimal model to the test data.

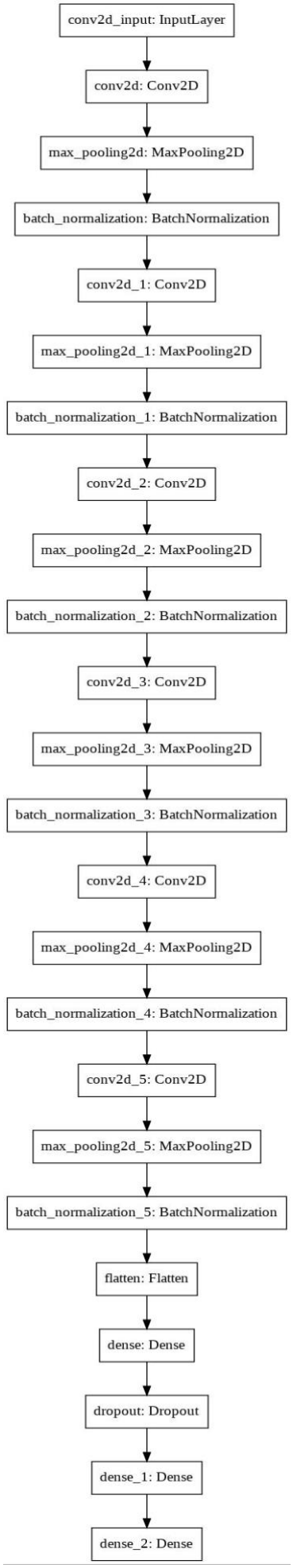
First of all, considering the number of samples we have, the value of k is chosen as 5 that means we have applied 5-fold cross validation. In each fold, from the training data, we have 20% of the data (~600) for validation and 80% of the data (~2400 samples) for training. The reason behind this is that, we had problems with the RAM and it crashes when we run the rest of the notebook together with k-fold cross validation section. In addition to that, utilizing only a random sub-part of the dataset did not prevent us from tuning the parameters succeffully. Using a for loop of fold size, for each iteration, we have fetched these data randomly, initialize the model, compile the model, fit the chosen training data and validate it using the chosen validation data.

During our validation phase we also noted the training and validation losses

5. Neural Network Architecture

Following Convolutional Neural Network is used to train, validate and test the model:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 208, 176, 16)      416
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 104, 88, 16)       0
_________________________________________________________________
batch_normalization (BatchNo (None, 104, 88, 16)       64
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 104, 88, 16)       6416
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 52, 44, 16)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 52, 44, 16)        64
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 52, 44, 16)        6416
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 26, 22, 16)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 26, 22, 16)        64
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 26, 22, 32)        4640
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 13, 11, 32)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 13, 11, 32)        128
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 11, 32)        9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 6, 5, 32)          0
_________________________________________________________________
batch_normalization_4 (Batch (None, 6, 5, 32)          128
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 6, 5, 32)          9248
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 3, 2, 32)          0
_________________________________________________________________
batch_normalization_5 (Batch (None, 3, 2, 32)          128
_________________________________________________________________
flatten (Flatten)            (None, 192)               0
_________________________________________________________________
dense (Dense)                (None, 64)                12352
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 16)                1040
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 68
=================================================================
Total params: 50,420
Trainable params: 50,132
Non-trainable params: 288
```


conv2d_input: InputLayer
conv2d: Conv2D
max_pooling2d: MaxPooling2D
batch_normalization: BatchNormalization
conv2d_1: Conv2D
max_pooling2d_1: MaxPooling2D
batch_normalization_1: BatchNormalization
conv2d_2: Conv2D
max_pooling2d_2: MaxPooling2D
batch_normalization_2: BatchNormalization
conv2d_3: Conv2D
max_pooling2d_3: MaxPooling2D
batch_normalization_3: BatchNormalization
conv2d_4: Conv2D
max_pooling2d_4: MaxPooling2D
batch_normalization_4: BatchNormalization
conv2d_5: Conv2D
max_pooling2d_5: MaxPooling2D
batch_normalization_5: BatchNormalization
flatten: Flatten
dense: Dense
dropout: Dropout
dense_1: Dense
dense_2: Dense

It can be observed that, we have used 6 convolutional layers with max pooling and batch normalization layers between them. In the final layers we have flattened the output and used 2 fully connected layers with a dropout layer between them. Models that are less complex compared to this one was able to capture and extract the features of our training data. Note that, we tried to include many parameters that belongs to convolutional layers. We achieved this by making the parameter padding = same. That prevents any decrement of dimensions and kept more parameters.

6. Hyper-Parameter Tuning

For hyper-parameter tuning, we have optimized 6 crucial parameters.

- Hyperparameters:
  - Type of Optimizer
  - Number of Epochs
  - Batch Size
  - Dropout rate
  - Inclusion of batch normalization
  - Learning Rate

The results we have obtained from hyper-parameter tuning phase is given below. Note that the tuning phase is processed based on the results obtained. That is, we have chosen the parameters based on a logic rather than applying all combinations since there are 6 parameters to tune, this yields a lot of possibilities that mostly contains inconsistent pairs. In the explanation of the hyperparameter tuning, we will refer to the index values of the trials.

Index	Learning Rate	Type of Optimizer	Number of Epochs	Batch Size	Dropout Rate	Inclusion of Batch Normalization	Accuracy	Balanced Accuracy
1	0.1	AdaDelta	50	8	0.25	Yes	%69.26	%69.72
2	0.1	AdaDelta	50	32	0.25	Yes	%65.25	%67.06
3	0.1	AdaDelta	50	64	0.25	Yes	%65.74	%65.24
4	0.01	AdaDelta	50	8	0.25	Yes	%51.71	%59.73
5	1	AdaDelta	90	8	0.25	Yes	%74.65	%77.29
6	1	AdaDelta	70	8	0.25	Yes	%76.17	%75.41
7	0.001	Adam	90	8	0.25	Yes	%74.67	%76.01
8	0.01	Adam	90	8	0.25	Yes	%30.04	%25.67
9	**0.003**	**Adam**	**90**	**8**	**0.25**	**Yes**	**%78.17**	**%78.98**
10	0.003	Adam	150	4	0.25	Yes	%77.37	%78.62
11	0.003	Adam	90	8	0.40	Yes	%80.8	%71.79
12	0.003	Adam	90	8	0.10	Yes	%79.26	%72.60
13	0.003	Adam	90	8	0.25	No	%76.51	%79.78
14	0.005	Adam	90	8	0.25	Yes	%77.01	%77.32

- Type of Optimizer
  For type of optimizer, we have chosen to try Adam optimizer and AdaDelta optimizer.
  - Adadelta optimizer is a more robost optimizer and it extends the optimizer called Adagrad. It is an adaptive optimizer which modifies learning rate based on moving window of gradient updates. That means, it does not accumulate all past gradients.
  - Adam optimizer is an optimized version of stochastic gradient descent for deep learning. It gathers the best properties of AdaGrad and RMSprop algorithms.

It can be observed that AdaDelta and Adam optimizer are used in index 1-6 and 7-13 respectively. From the results obtained for each trial of optimizer we got the best result of Adadelta in 5 and best result of Adam optimizer in 9.

- Learning Rate

Learning Rate basically determines how quickly or slowly the neural network model learns a problem. Keeping it too high may obstruct you from converging to a minima by making oscillations while keeping it too low would require too much time to converge to a minima or stuck in a local minima and couldn't improve more. In other words, it controls the amount of apportioned error that the weights of the model are updated with each time they are updated.

  - In the paper of Adadelta, we have observed that it tends to benefit from higher learning rates. Therefore we have used 0.1 and 1 as the learning rates of AdaDelta (index 1-6) and just used 0.01 to ensure that small learning rate is not useful with AdaDelta optimizer (index 4).
  - For Adam optimizer, compared to Adadelta, it benefits from smaller learning rates. For this reason, we have utilized 0.001 and 0.003 as the learning rates of Adam (index 7-13) and just used 0.01 to ensure that higher learning rate is not useful with Adam optimizer (index 8).

- Number of Epochs

The number of epochs is a hyperparameter of training phase that controls the number of complete passes through the training dataset. As we have mentioned in the learning rate section above, a small learning rate requires a longer training phase since the step size is smaller. This demonstrates us that number of epochs should be inversely proportional with learning rate. For this reason, we have tried the following learning rate – epoch pairs, (0.003, 150) and (0.01, 90) (index 8,10).

- Batch Size

Batch size determines the number of samples that are processed before the update of the weights occur. In theory, batch size = 1 is the optimal value to train the model since it accounts each data sample individually rather than averaging multiple ones according to batch size. However, this takes so much time that does not really effect the result. For this reason, we tried to capture the optimal batch size. It can be observed that the values 8, 32, 64 are investigated and observed that 8 is the optimal one (index 1-3). Since we have observed a significant increase between the epoch size 8 and 32, in further steps, we have compared the validation accuracy by applying 4 and 8 batch sizes (index 9,10). We have observed that decrementing the batch size does not really benefit that much even we have gone through more epochs in the phase we have 4 batch size.

- Dropout Rate

Dropout rate enables us to determine the amount of the neurons that will be discarded in order to prevent overfitting by sticking only a certain characteristic. In order to not throw away so much or throw away a few, we should optimize this value. Hence, we have varied the dropout rate between the values 0.10, 0.25 and 0.40 (index 9, 11, 12). We have observed that keeping the dropout rate comparably high (0.40) or low (0.10) yields a notable difference between accuracy and balanced accuracy.

- Inclusion of Batch Normalization

Batch Normalization has the effect of stabilizing the learning process by keeping the outputs of the layers in a certain range. It also accelerates the learning phase. Hence, one can choose a higher learning rate. However, we should note that for Adam optimizer the learning rate should be

considerably small. Therefore, we did not apply a dramatic increase, so we compared the results of validation when 0.005 and 0.003 is used as learning rates while first one has batch normalization layers and second one has not (index 13, 14). We have observed that there is an increase in both accuracy and balanced accuracy when we include the batch normalization layer. However, one should also compare the results (index 9, 14) and note that increase in learning rate is not beneficial when you consider Adam optimizer. Therefore, we decide to include the batch normalization layers with a learning rate higher than 0.001 and lower than 0.005.

Overall, when we take into account the comparisons we have done, we can observe that picking the index 9 as the optimal combination of hyperparameters would be the correct choice since it has a comparably high accuracy and balanced accuracy while having small difference between each other.

7. Over-Fitting

Describe what you did to address over-fitting.

In order to prevent over-fitting, we have applied 5-fold cross validation which ensures that all different sections of the training data consistently yield an accurate result.

During our 5-fold cross validation phase, we were able to tune 6 crucial hyperparameters and ensured that each part of the data ends up with an accurate result. Therefore, we avoided building a biased model and chose the parameters in a way that all of the training data contributed both in training and validation phase. That means, the model does not have any unseen data except the test set.

Maxpool layers enables us to control the number of parameters by applying a max operation to pool sets of features. Increasing the number of features causes the model to capture every tiny detail. Which means, your model completely fits to the data you have trained it with. To prevent this, maxpool layer is utilized to reduce dimensionality.

Drop out layers prevent the model to rely on any single feature by ensuring that this feature is not available all the time. That drives the model to look for different potential characteristics instead of sticking into a single one. Hence this also prevents overfitting.

L1 regularization is also known as L1 norm which prevents overfitting by shrinking the parameters towards 0. When we use L1 norm, we penalize absolute value of weights. We have also added an L1 regularization to the end of activation function in the fully connected layer section. This benefits us to overcome any potential overfitting. When the parameter is too much, the model underfitted so it should also be optimized.

We have utilized class weights in order to prevent overfitting to a specific data. During the learning phase, if a sample that belongs to minority class is misclassified then model is punished proportional to the weight assigned to that class so that the model cannot ignore the minority classes and shape its learning procedure by giving more weight to minority classes than majority classes to have balance between them.