

Alzheimer's Disease Prediction

Machine Learning - Mini Project

Team Members:

- | | | | |
|---|-----------------------------|---|--------------------|
| ❑ | Shashanka Venkatesh | - | 18 5001 145 |
| ❑ | Suraj Jain | - | 18 5001 177 |
| ❑ | Vishakan Subramanian | - | 18 5001 196 |
| ❑ | Vishnu K Krishnan | - | 18 5001 200 |

About Alzheimer's

Alzheimer's Disease

Alzheimer's disease (AD), also referred to simply as Alzheimer's, is a **neurodegenerative disease** that usually starts slowly and progressively worsens. It is the cause of **60–70%** of cases of dementia.

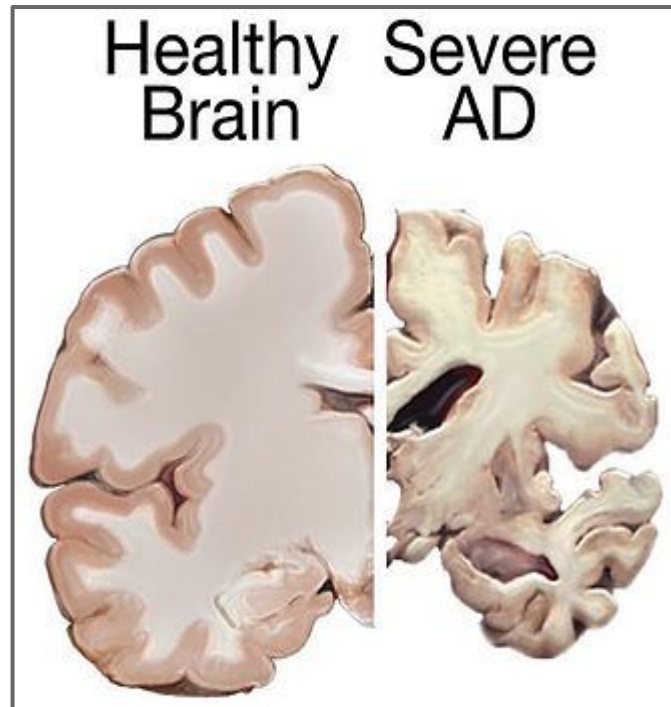
The most common early symptom is difficulty in remembering recent events. As the disease advances, symptoms can include problems with language, disorientation (including easily getting lost), mood swings, loss of motivation, self-neglect, and behavioral issues.

The course of Alzheimer's is generally described in three stages, with a progressive pattern of **cognitive and functional impairment**. The three stages are described as early or mild, middle or moderate, and late or severe. The disease is known to target the hippocampus which is associated with memory, and this is responsible for the first symptoms of memory impairment. As the disease progresses so does the degree of memory impairment.

Alzheimer's Disease

Advanced medical imaging with **computed tomography** (CT) or **magnetic resonance imaging** (MRI), and with **single-photon emission computed tomography** (SPECT) or **positron emission tomography** (PET) can be used to help exclude other cerebral pathology or subtypes of dementia. Moreover, it may predict conversion from prodromal stages (mild cognitive impairment) to Alzheimer's disease.

Our dataset comprises of **MRI scan** images.



Application

The **early stages** of Alzheimer's disease are often **very difficult to diagnose**, leading to complications once the disease develops more prominently. Early diagnosis would help in preventive treatment and in taking proper healthcare measures. One of the ways in which that can be done is identifying the existence of dementia through MRI scans of people at risk. The machine learning model aids both as a tool and a reliable source of diagnosis for doctors and healthcare experts to properly analyze the patient's medical history.

The use of ML in medicine is growing exponentially in the modern day and this is one such application where machines can predict the outcome with a high degree of confidence.

Clinical decision-making is a very sound skill that can be transferred from human intelligence to machine appropriately, thus providing **quality standards of diagnosis**. Automation in this field will reduce workload of doctors as well.

About the Dataset

Dataset

The dataset was obtained from from Kaggle: [Alzheimer's Dataset](#).

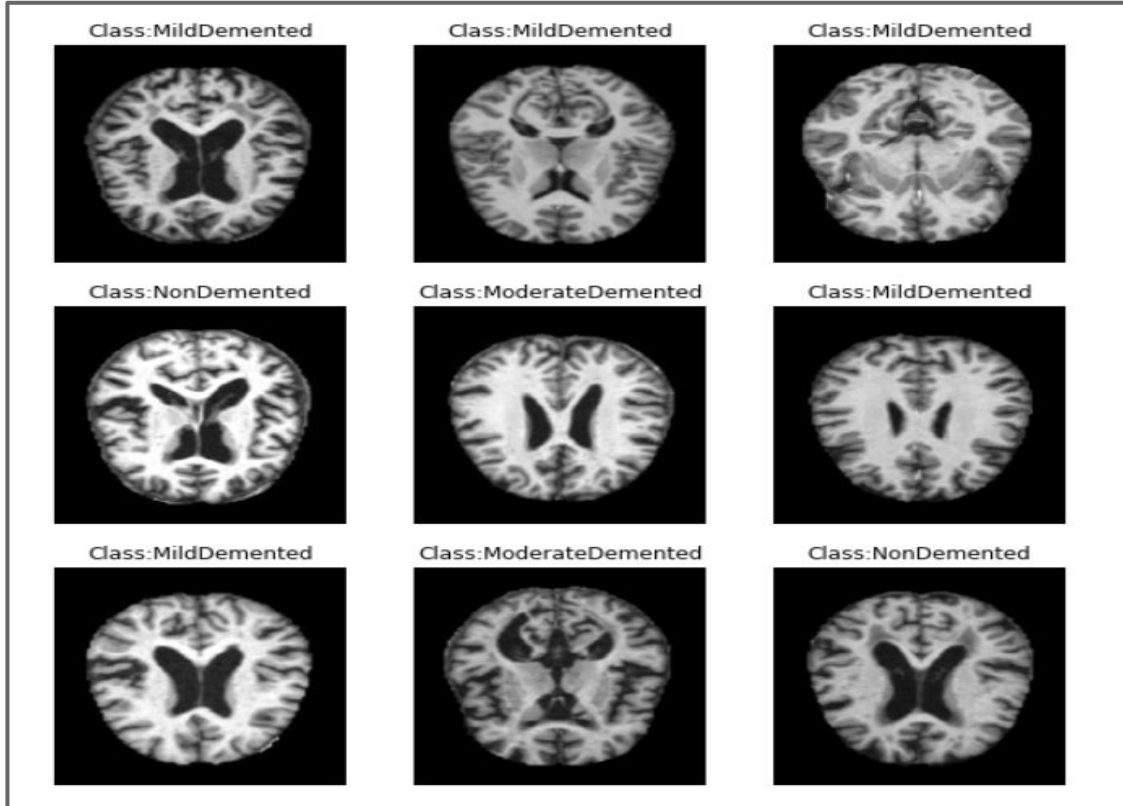
It contains a total of **6400** images, split into **4 classes**, namely:

- Non Demented
- Mild Demented
- Very Mild Demented
- Moderate Demented

All of these images are of size **178 x 208** pixels.

The goal of our project is to build a **Convolutional Neural Network** from scratch to learn the features from these images and perform accurate classification of the data.

Sample Data



Here we can see some of the sample images of the dataset that we are going to use.

Data Preprocessing

Image Augmentation

We use the inbuilt **ImageDataGenerator** class under Keras' preprocessing package to perform Image Augmentation under the following categories:

- Horizontal Flip
- Zoom
- Brightness
- Fill Mode

The advantage of using this is that we artificially create more samples for the model to learn from, and it also helps prevent overfitting.

Over Sampling

Oversampling is a key preprocessing idea we chose to employ here, mainly because the dataset is imbalanced and we do not have many samples in the **ModerateDemented** class to work with. This may lead to the model completely ignoring the features of that class, which oversampling prevents.

For this task, we employ the **SMOTE (Synthetic Minority Oversampling TEchnique)** Over Sampler under the `imblearn.over_sampling` package under `scikit-learn`. This synthesizes new examples for the minority class.

This is indeed compute intensive, but the trade-off is necessary to build an impartial model.

Train-Test-Validation Split

We split the data after it passes through the various preprocessing stages, which yields us around **12800 samples** to work with.

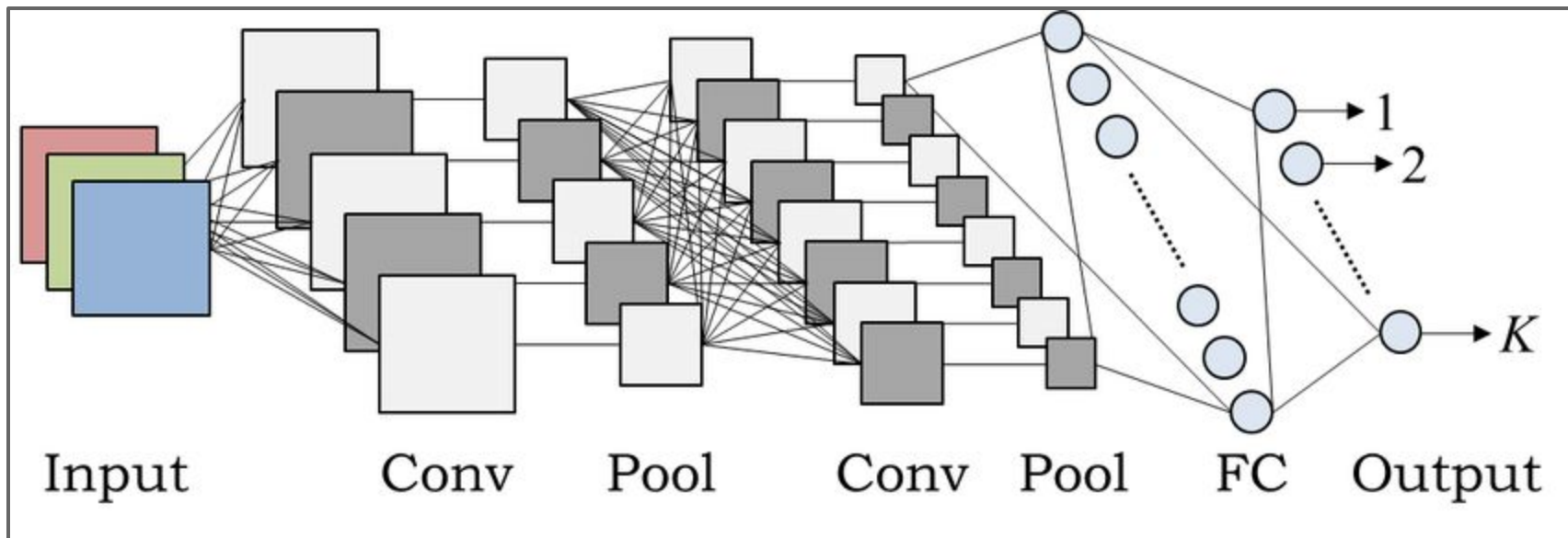
The split we chose:

- Training: **60%**
- Validation: **20%**
- Testing: **20%**

Our Proposed Architectures

Our CNN Model

Convolutional Neural Network



Given here is a typical CNN architecture, and our architecture generally resembles the one shown above.

Convolutional Neural Network

Our CNN architecture comprises of:

- ❖ An **Input Layer** that feeds the image matrix of size 176x176.
- ❖ Two **Convolutional (2D) Layers** of 16x3 kernel size, followed by a **Max Pooling (2D) Layer**.
- ❖ Three custom-defined **convolution blocks** of size 32x3, 64x3 and 128x3 (defined in the next slide)
- ❖ A **Dropout Layer** that drops 20% of the neurons at random during each training epoch, thus acting as a regularizer of sorts, and preventing overfitting.
- ❖ A custom-defined **convolution block** of size 256x3.
- ❖ Another **Dropout Layer**.
- ❖ A **Flatten Layer**, which converts the final 2D matrix outputted by the Convolution phase into a 1D array for the Dense Neural Network to use.
- ❖ Three custom-defined **dense fully connected blocks** of size 512, 128 & 64 (defined in the next slide)
- ❖ A **Softmax Layer** at the end to predict in terms of probabilities as to which class the given data sample might belong to.

Convolutional Block

Our custom-defined convolutional block consists of:

- Two **Separable Convolution (2D) Layers** - which factorizes a convolutional kernel into two smaller kernels
- A **Batch Normalization Layer** - to perform batch-wise normalization to maintain the mean of the output towards 0 with standard deviation close to 1. This helps the training to go faster and the learning rate to be kept on a comparatively higher side. It also keeps the training phase stable, as it regularizes the model.
- A **Max Pooling (2D) Layer** - that performs an aggregation of the input matrix it receives, and extracts the maximum value around each sub-block of predefined size, thus halving the input matrix dimensions.

Dense Block

Our custom-defined dense block consists of:

- A **Dense Layer** of predefined size (a simple neural network layer), that will be fully connected to the next layer.
- A **Batch Normalization Layer** for similar purposes as discussed in the previous slide.
- A **Dropout Layer**, so that the Dense Layer becomes more robust and ensures that each neuron learns some feature. It acts as a regularizer of sorts and prevents the model from lazily overfitting. It also speeds up the training process and backpropagation process.

Summary Of Our CNN Model

Model: "cnn_model"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 176, 176, 16)	448

conv2d_1 (Conv2D)	(None, 176, 176, 16)	2320

max_pooling2d (MaxPooling2D)	(None, 88, 88, 16)	0

sequential (Sequential)	(None, 44, 44, 32)	2160

sequential_1 (Sequential)	(None, 22, 22, 64)	7392

sequential_2 (Sequential)	(None, 11, 11, 128)	27072

dropout (Dropout)	(None, 11, 11, 128)	0

sequential_3 (Sequential)	(None, 5, 5, 256)	103296

dropout_1 (Dropout)	(None, 5, 5, 256)	0

flatten (Flatten)	(None, 6400)	0

sequential_4 (Sequential)	(None, 512)	3279360

sequential_5 (Sequential)	(None, 128)	66176

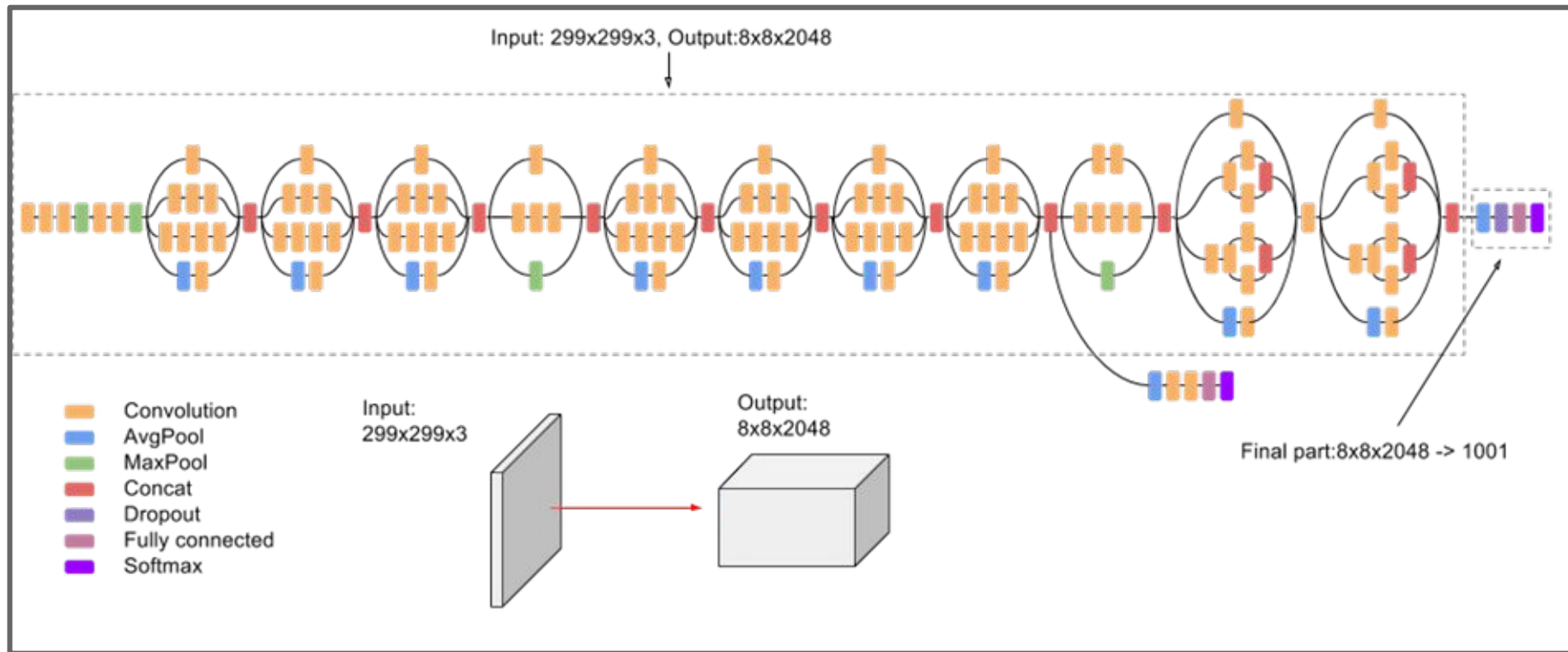
sequential_6 (Sequential)	(None, 64)	8512

dense_3 (Dense)	(None, 4)	260
=====		
Total params: 3,496,996		
Trainable params: 3,494,628		
Non-trainable params: 2,368		

Transfer Learning:

InceptionV3 Model

Inception V3 Architecture



About Inception V3

InceptionV3 is a convolutional neural network for assisting in image analysis and object detection, and got its start as a module for **Googlenet**. It is the third edition of Google's Inception Convolutional Neural Network, originally introduced during the ImageNet Recognition Challenge. Inception-v3 is a convolutional neural network that is **48 layers deep**. The pretrained model can classify over **1000** unique classes of images.

It is available for us to use under the **TensorFlow** library.

The inspiration behind the name “*Inception*”, a dialogue from Christopher Nolan’s famous film, Inception.



Transfer Learning CNN Architecture

The Transfer Learning CNN architecture comprises of:

- ❖ The **Inception V3 Architecture** without its Fully Connected Layers. This part of the architecture is used as is without making any changes to the weights or layers. It is used as is and is not trained by us, as training this part of the model will take a very long time. The weights are **frozen**.
- ❖ Four custom-defined **dense fully connected blocks** of size 512, 256, 128 & 64 (as defined in the next slide)
- ❖ A **Softmax Layer** at the end to predict in terms of probabilities as to which class the given data sample might belong to.

Dense Block

Our custom-defined dense block consists of:

- A **Dense Layer** of predefined size (a simple neural network layer), that will be fully connected to the next layer.
- A **Batch Normalization Layer** for similar purposes as discussed in the previous slide.
- A **Dropout Layer**, so that the Dense Layer becomes more robust and ensures that each neuron learns some feature. It acts as a regularizer of sorts and prevents the model from lazily overfitting. It also speeds up the training process and backpropagation process.

Summary Of Inception Based CNN Model

Model: "inception_cnn_model"

Layer (type)	Output Shape	Param #
inception_v3 (Functional)	(None, 4, 4, 2048)	21802784
dropout (Dropout)	(None, 4, 4, 2048)	0
global_average_pooling2d (G1	(None, 2048)	0
flatten (Flatten)	(None, 2048)	0
batch_normalization_94 (Batc	(None, 2048)	8192
dense (Dense)	(None, 512)	1049088
batch_normalization_95 (Batc	(None, 512)	2048
dropout_1 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
batch_normalization_96 (Batc	(None, 256)	1024

dropout_2 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32896
batch_normalization_97 (Batc	(None, 128)	512
dropout_3 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 64)	8256
dropout_4 (Dropout)	(None, 64)	0
batch_normalization_98 (Batc	(None, 64)	256
dense_4 (Dense)	(None, 4)	260
Total params: 23,036,644		
Trainable params: 1,227,844		
Non-trainable params: 21,808,800		

Concepts Used

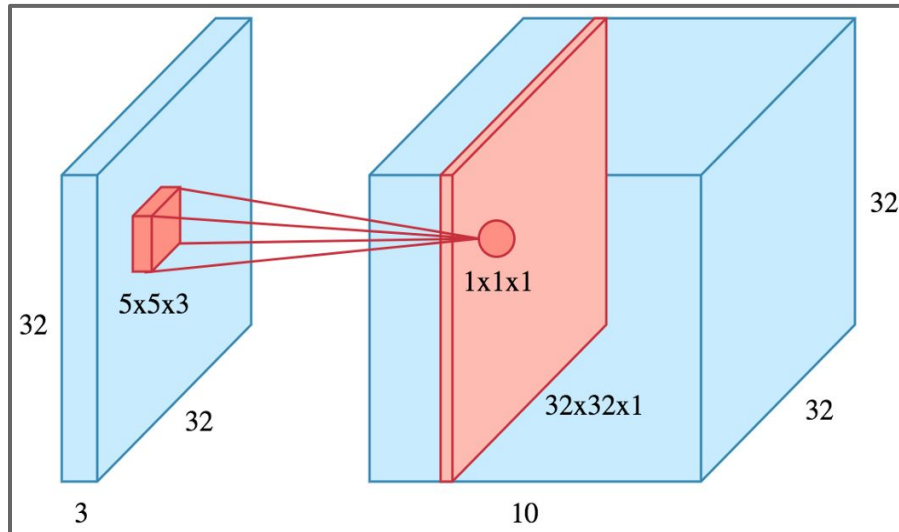
Visualizing Convolution

1	1x1	1x0	0x1	0
0	1x0	1x1	1x0	0
0	0x1	1x0	1x1	1
0	0	1	1	0
0	1	1	0	0

Input x Filter

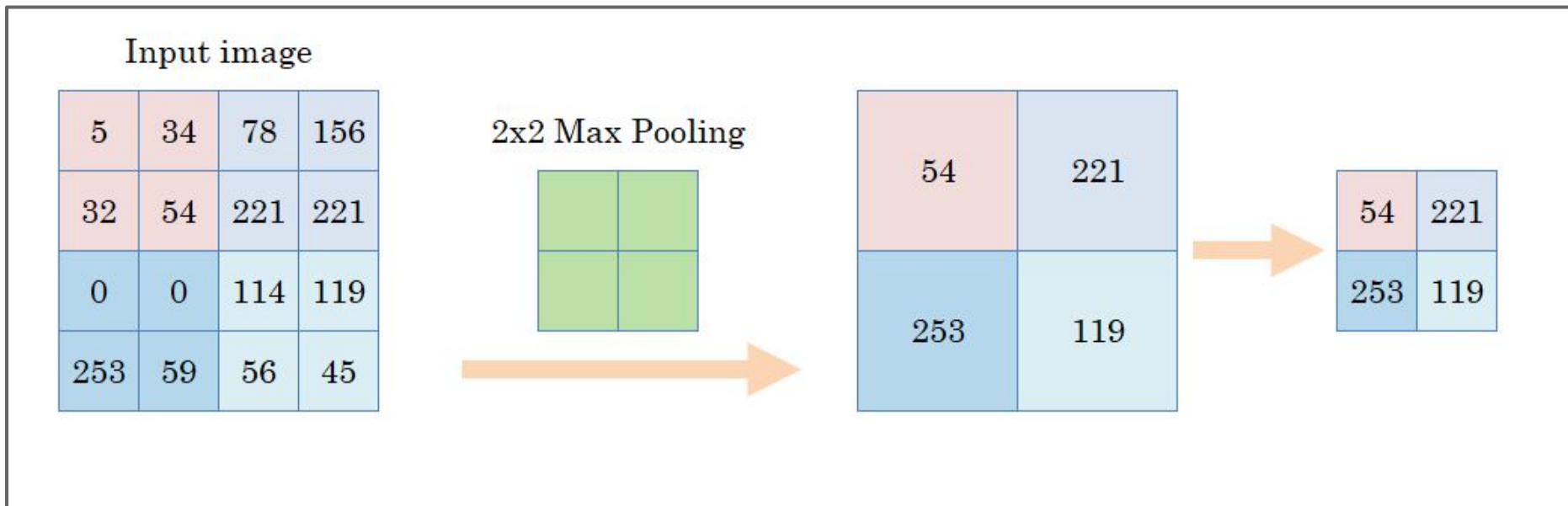
4	3	

Feature Map



Convolution: A mathematical matrix operation, where a kernel “slides” over the 2D input data, performing an elementwise multiplication with the part of the input it is currently on, and then summing up the results into a single output pixel.

Visualizing Max-Pooling



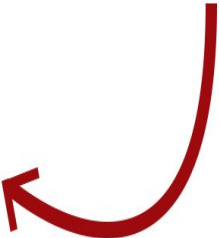
Max Pooling: We take groups of pixels (for example, groups of 2x2 pixels) and perform an aggregation over them. Then we take the maximum value of the pixels in the group.

Softmax Activation Function

$$P(y=j \mid \theta^{(i)}) = \frac{e^{\theta^{(i)}}}{\sum_{j=0}^k e^{\theta_j^{(i)}}}$$

where $\theta = w_0x_0 + w_1x_1 + \dots + w_kx_k = \sum_{i=0}^k w_i x_i = w^T x$

Softmax function



Softmax: is an activation function that scales numbers/logits into probabilities.

The output of a Softmax is a vector (say v) with probabilities of each possible outcome. The probabilities in vector v sums to one for all possible outcomes or classes.

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$

Batch Normalization: focuses on standardizing the inputs to any particular layer(i.e. activations from previous layers). Standardizing the inputs mean that inputs to any layer in the network should have approximately zero mean and unit variance.

Advantages:

- ★ Allows training to go faster.
- ★ Allows higher learning rates.
- ★ Provides regularization.
- ★ Provides better results.

Other Noteworthy Points

For the activation function, we use the **ReLU** (Rectified Linear Unit) function as it is very popular and widely used, especially with image-based applications.

We have run the model for **100 epochs** in total, and also defined three **callback** functions (**Custom Callback**, **EarlyStopping** & **ReduceLROnPlateau**) to prevent training in case the model overfits to the data, or in case validation loss does not improve over the span of 3 iterations.

Training the model roughly takes around **40-60 minutes** on the resources allotted by Kaggle. This includes the usage of a **GPU Accelerator** to hasten the training process.

Performance Metrics

Loss Function & Optimizer

For the CNN model & the Inception V3 model that we developed, we used the **Categorical Cross Entropy** loss as the loss function, which is the most apt, and is the de-facto standard for multi-class classification tasks in general.

For the optimizer, we preferred to use the **Adam** optimizer for our model, since it's very popularly used since it is a stochastic gradient descent method that is based on **adaptive estimation of first-order and second-order moments**, and uses less memory and is very computationally efficient.

For the Inception V3 model, we used the **RMSProp** optimizer for the purpose of comparison, which uses **plain momentum** and maintains a **moving average of the gradients**, and uses that average to estimate the variance. The performance of RMSProp is comparable to the Adam optimizer.

Performance Metrics

We evaluate the model using the **Categorical Accuracy**, the **AUC (Area Under ROC Curve)** & the **F1-Score** metrics.

For further performance analysis, we use the following metrics as well:

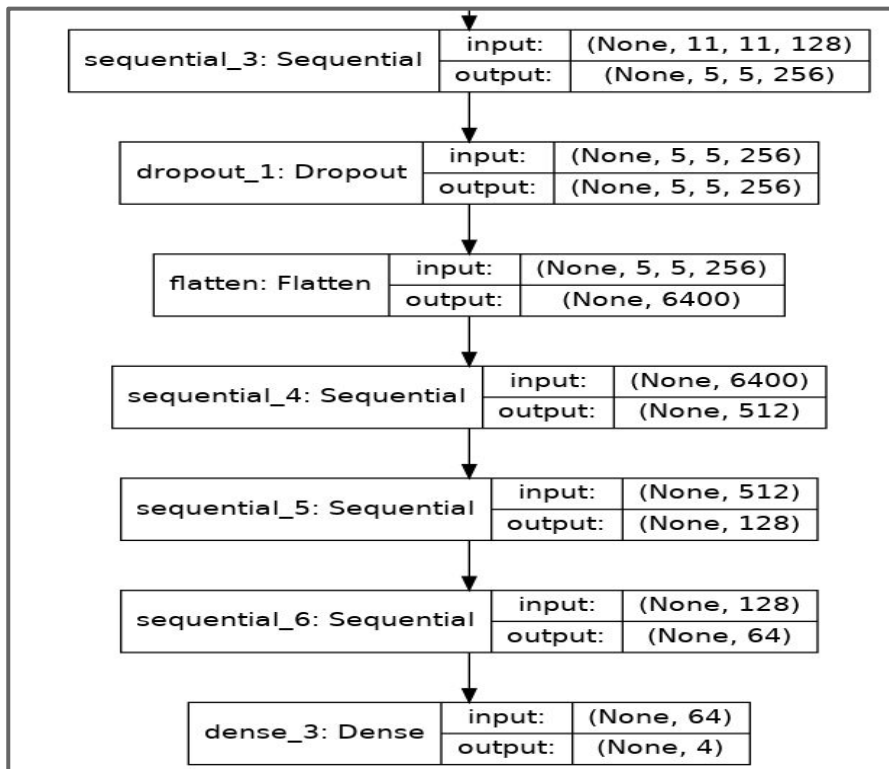
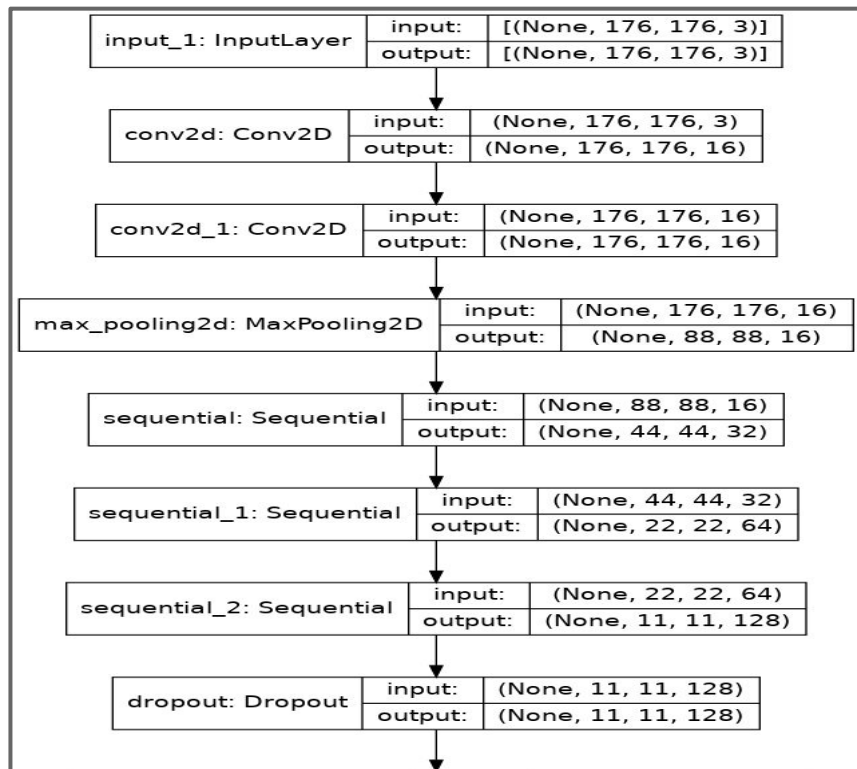
- Confusion Matrix
- Classification Report
- Balanced Accuracy Score
- Matthew's Correlation Coefficient.

The last two metrics are most essential for this task as the dataset is imbalanced, i.e. the **ModerateDemented** class comprises only **1%** of the total data.

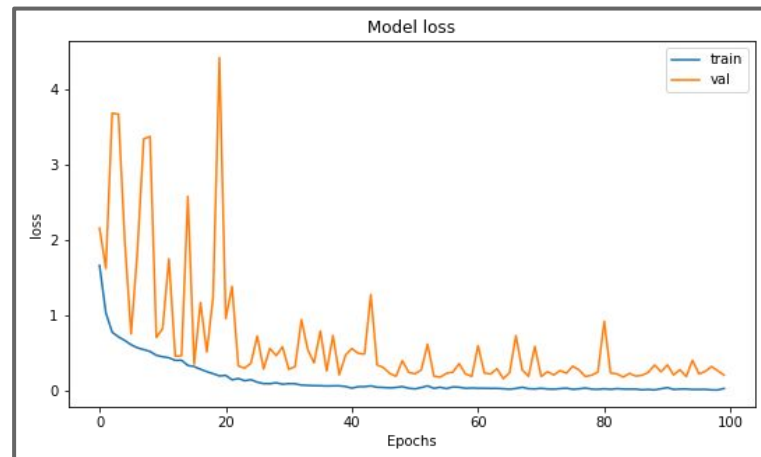
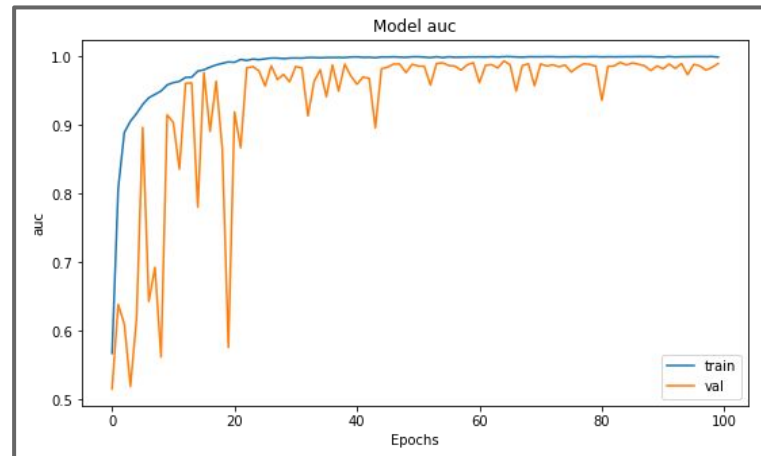
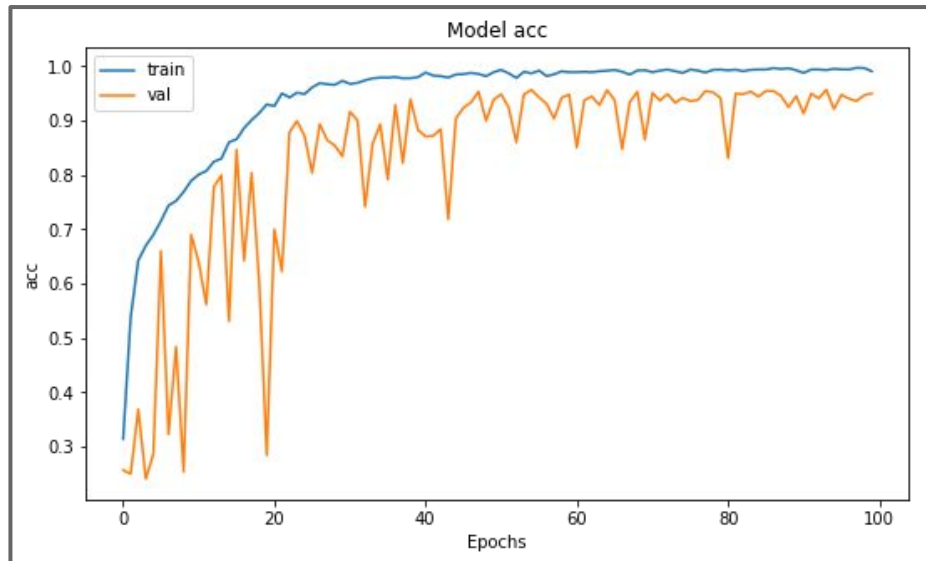
Results

Our CNN Model

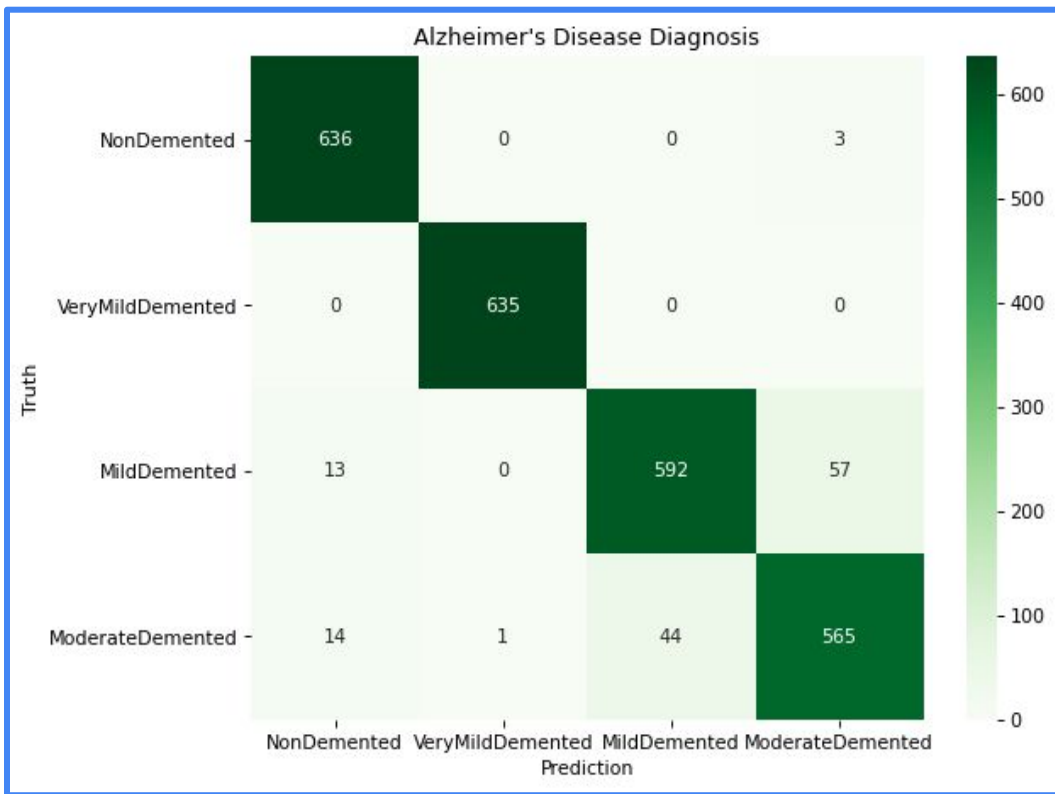
Model's Plot



Model's Performance



Confusion Matrix



Given here is the confusion matrix resulting from the predictions done on the training set by our **CNN architecture**. It performs upto expectations, and there are only few misclassifications. The model does not seem to be biased towards any one class in particular, and thus we can conclude that there is **no overfitting**.

The model also yielded a **Balanced Accuracy Score** of **94.88%** and **Matthew's Correlation Coefficient** of **93.14%**, which implies that the model has performed very well.

Classification Report

	precision	recall	f1-score	support
NonDemented	0.96	1.00	0.98	639
VeryMildDemented	1.00	1.00	1.00	635
MildDemented	0.93	0.89	0.91	662
ModerateDemented	0.90	0.91	0.90	624
micro avg	0.95	0.95	0.95	2560
macro avg	0.95	0.95	0.95	2560
weighted avg	0.95	0.95	0.95	2560
samples avg	0.95	0.95	0.95	2560

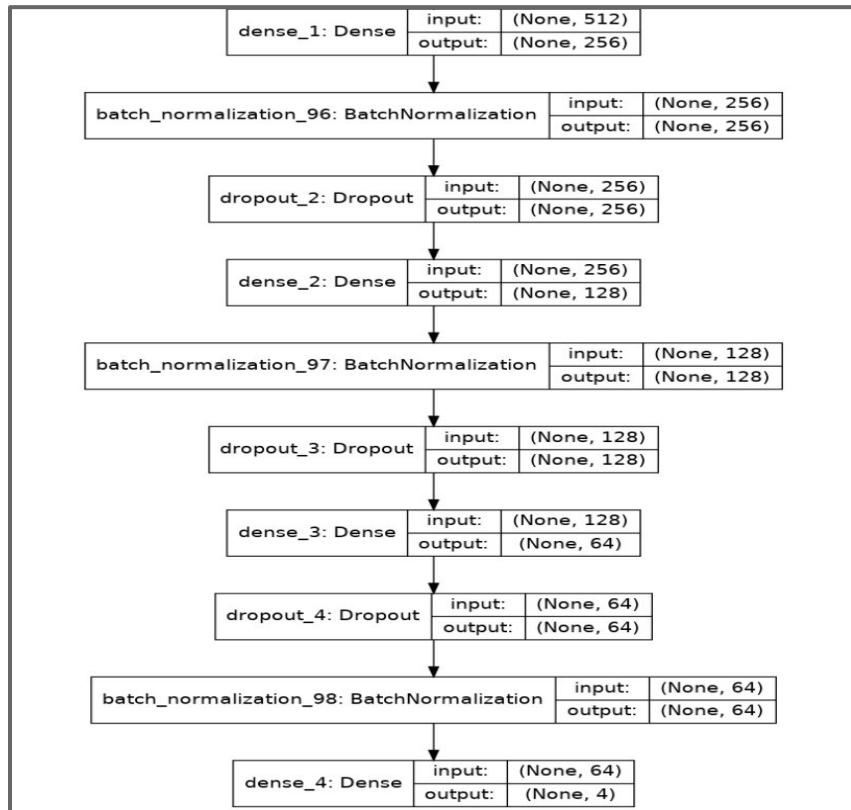
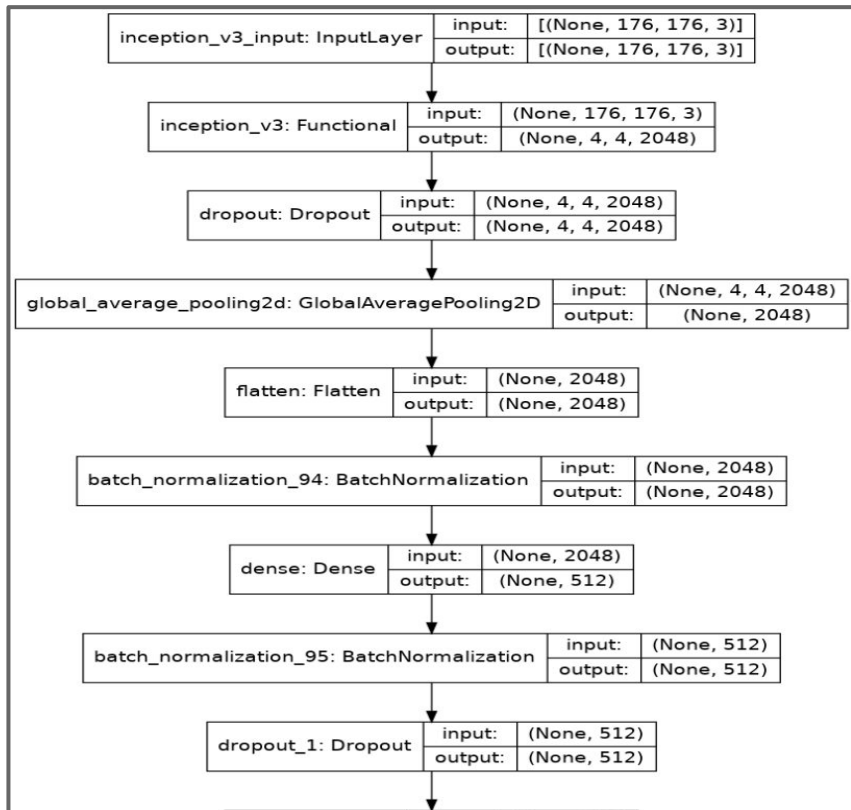
Our **CNN model** fares well in each class when it comes to precision, recall and F1-score.

Although, it is to be noted that the model can be improved to work better in the **MildDemented** & **ModerateDemented** classes (classes 3 & 4 in the classification report)

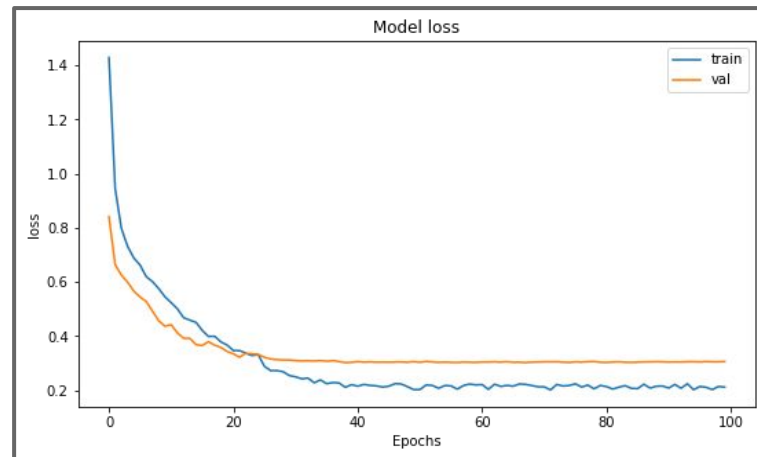
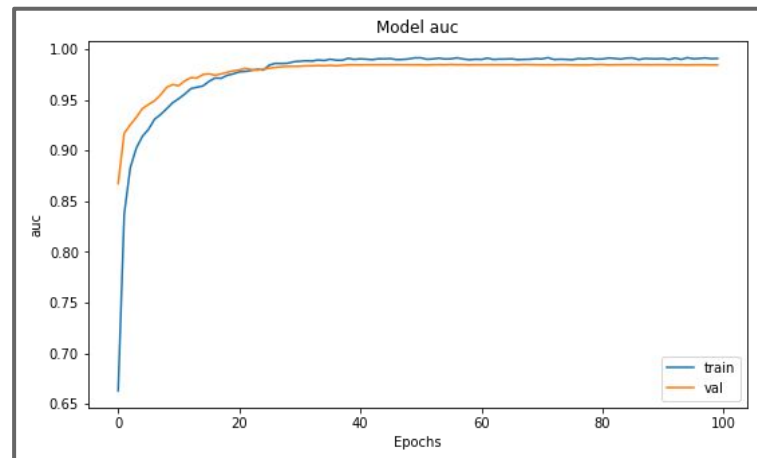
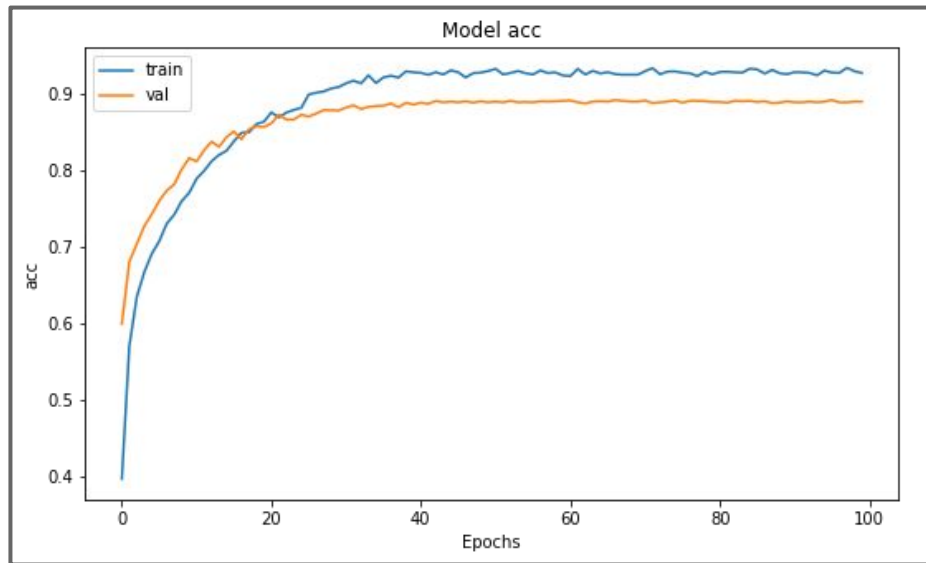
Transfer Learning:

InceptionV3 Model

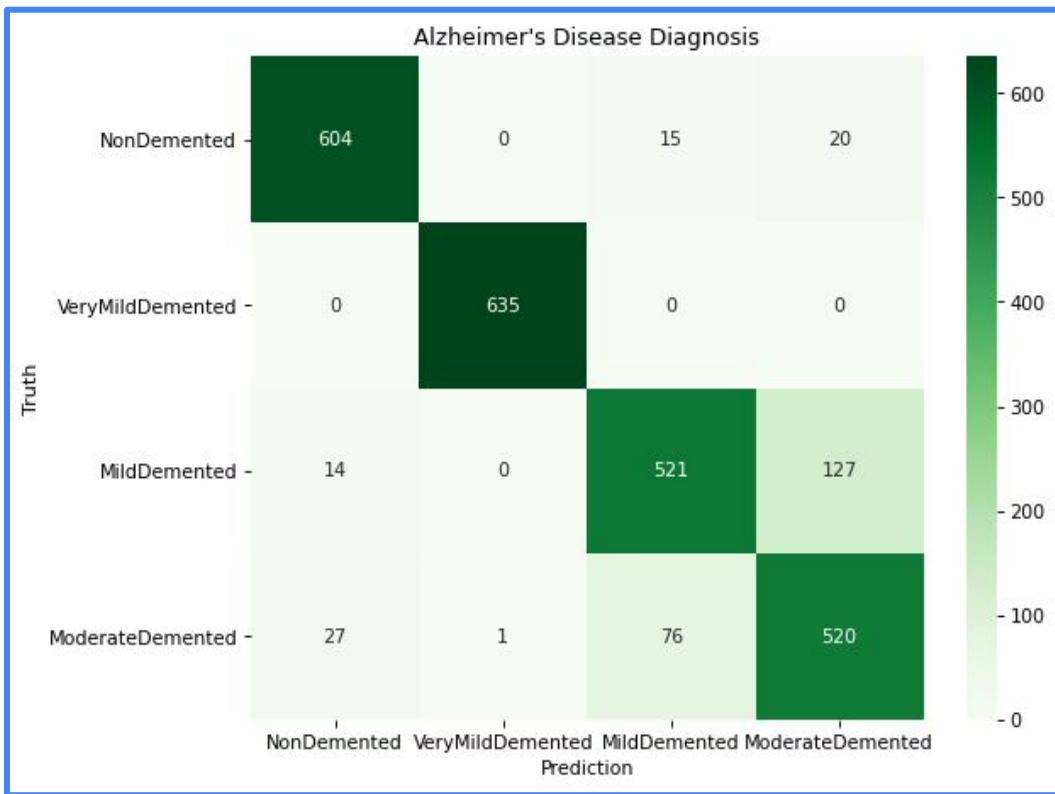
Model's Plot



Model's Performance



Confusion Matrix



Given here is the confusion matrix resulting from the predictions done on the training set by our **Inception architecture**. It performs upto expectations, and there are only few misclassifications. The model does not seem to be biased towards any one class in particular, and thus we can conclude that there is **no overfitting**.

The model also yielded a **Balanced Accuracy Score** of **89.14%** and **Matthew's Correlation Coefficient** of **85.46%**, which implies that the model has performed very well.

Classification Report

	precision	recall	f1-score	support
NonDemented	0.94	0.95	0.94	639
VeryMildDemented	1.00	1.00	1.00	635
MildDemented	0.85	0.79	0.82	662
ModerateDemented	0.78	0.83	0.81	624
micro avg	0.89	0.89	0.89	2560
macro avg	0.89	0.89	0.89	2560
weighted avg	0.89	0.89	0.89	2560
samples avg	0.89	0.89	0.89	2560

The **Inception model** fares well in each class when it comes to precision, recall and F1-score.

Although, it is to be noted that the model can be improved to work better especially in the **MildDemented** & **ModerateDemented** classes (classes 3 & 4 in the classification report)

Comparison Chart

<u>Metric</u>	<u>Our CNN Model</u>	<u>Inception V3 Model</u>
Training Accuracy	99.21 %	92.26 %
Validation Accuracy	95.02 %	88.96 %
Testing Accuracy	94.84 %	89.06 %
Training AUC	99.91 %	99.11 %
Validation AUC	98.98 %	98.44 %
Testing AUC	99.11 %	98.54 %
Training F1-Score (Average)	99.21 %	92.30 %
Validation F1-Score (Average)	94.99 %	88.89 %
Testing F1-Score (Average)	94.83 %	89.09 %
Balanced Accuracy Score	94.88 %	89.14 %
Matthew's Correlation Coefficient	93.14 %	85.46 %

Using The Model

We have exported both our CNN model & the InceptionV3 model in the form of a **HDF5** format, which can be imported into other machine learning modules and reused/retrained/improved upon.

The exported file of our model includes the architecture and the pre-trained weights as well, and comes to around **54.6 MB** in size, which is portable.

The exported file of the Inception V3 model is however almost double in size, owing to its huge architecture and number of parameters. It comes to around **94.6 MB** in size.

Going Further

Using Pre-Trained Models

For the task, we did not choose to use existing Image Classification Architectures like the **VGGNet, LeNet, ResNet, AlexNet** etc. which are very powerful and deeply trained for other popular image classification tasks, as the **Inception V3** model itself did not show that much of an improvement over our model.

Since the main focus in our project was to learn-on-the-go, we decided to build a CNN model from scratch, taking ideas and inspirations from these architectures. Our custom built architecture also proved to be powerful, as it yielded a **94.84%** accuracy on the **testing** data, as well as **99.21%** accuracy on the **training** data.

The high accuracy score obtained by our model made us decide that there might not be much significant improvement in using pre-trained architectures.

Performing Further Pre-Processing

We were limited by Kaggle's resource quota limitations, and thus we were not able to perform deep pre-processing, as the RAM quota exceeds the given **13 GB**. SMOTE is **very resource intensive**, and thus limits the amount of preprocessing and augmentation that can be done.

Given access to more compute resources, further preprocessing tasks can be done to enhance the model's performance.

Our notebooks can be found [HERE \(Custom CNN\)](#) & [HERE \(InceptionV3\)](#) for reference.

References

References

- ❏ Kaggle - <https://www.kaggle.com/>
- ❏ Towards Data Science - <https://towardsdatascience.com/>
- ❏ TensorFlow Documentation - <https://www.tensorflow.org/guide>
- ❏ Wikipedia - <https://www.wikipedia.org/>

“Take care of your brain, and it will take care of you.”



Thank You!