

Final Project – Deep Learning - 99006

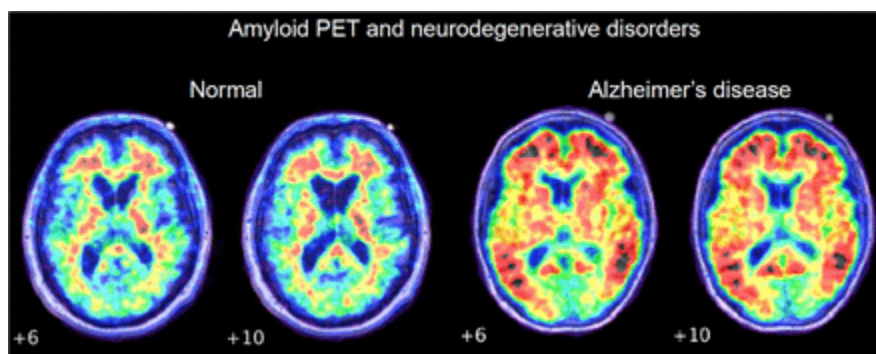
Alzheimer's MRI Classification -

Project Report

1. Introduction

Alzheimer's disease

Alzheimer's disease is a neurodegenerative dementia, responsible for 60-80% of all dementias. It is associated with an accumulation and deposition of cerebral amyloid- β ($A\beta$). Ref.1,2



Amyloid PET comparison of Non-AD vs. AD brain MRI image - from https://jnm.snmjournals.org/content/63/Supplement_1/13S

Low clinical differentiation of neurodegenerative dementias Ref.3

*“Alzheimer's disease (AD) and Lewy body disease (LBD) are the two most common neurodegenerative dementias and can occur in combination (AD+LBD). Due to overlapping biomarkers and symptoms, clinical differentiation of these subtypes could be difficult. Clinical diagnosis of AD+LBD has poor sensitivities: Over 61% of participants with autopsy-confirmed AD+LBD were diagnosed clinically as AD. **Clinical diagnosis of AD has a low sensitivity at the early dementia stage and low specificities at all stages.** Among participants diagnosed as AD in the clinic, over 32% had concurrent LBD neuropathology at autopsy. Among participants diagnosed as LBD, 32% to 54% revealed concurrent autopsy-confirmed AD pathology. With increasing dementia stages, the clinical diagnosis accuracy of black participants became significantly worse than other races, and diagnosis quality significantly improved for males but not females.”*



“These findings demonstrate the problem with Clinical diagnosis of AD, LBD, and AD+LBD are inaccurate and suffer from significant disparities on race and sex. They provide important implications for clinical management, anticipatory guidance, trial enrollment and applicability of potential therapies for AD (and promote research into better biomarker-based assessment of LBD pathology).”

Our Goal - Very High Accuracy on MRI Classification of AD

Since **clinical diagnosis** of AD has its problems (as seen above), we wanted to see if there is a way to get very high accuracies for detection of AD and its different stages. We were able to do so by applying Deep Learning techniques on a dataset of **MRI images of patients' brains**. Our best model achieved an accuracy of 99.53% (using pretrained Inception-V3 weights). **In our project, we focused only on AD prediction** and classified patients into 4 different stages: Non Demented, Very Mild Demented, Mild Demented and Moderate Demented.

Prior research

In their article “Towards Alzheimer’s Disease Classification through Transfer Learning” ^{Ref.4} - Hon and Khan state that:

*“Detection of AD from neuroimaging data such as MRI through machine learning have been a subject of intense research in recent years. Recent success of deep learning in computer vision has progressed such research further. However, common limitations with such algorithms are reliance on a large number of training images, and requirement of careful optimization of the architecture of deep networks. In this paper, we attempt **solving these issues with transfer learning**, where state-of-the-art architectures such as VGG and Inception are initialized with pre-trained weights from large benchmark datasets consisting of natural images, and the fully-connected layer is re-trained with only a small number of MRI images. We employ image entropy to select the most informative slices for training. Through experimentation on the OASIS MRI dataset, we show that with training size almost 10 times smaller than the state-of-the-art, we reach comparable or even better performance than current deep-learning based methods.”*

Their experiments are summarized in this table, where we can clearly see that classification of AD yields far better accuracies when using transfer learning on a relatively small dataset:

Model	Avg. Acc. (st. dev.) (%)
VGG16 (from scratch)	74.12 (1.55)
VGG16 (transfer learning)	92.3 (2.42)
Inception V4 (transfer learning)	96.25 (1.2)

Table I shows the accuracy results (with standard deviation) for the three models. As can be seen, VGG16, when trained from scratch, results in poor performance. This can be attributed towards the small training size that we used, which can result in overfitting/underfitting. However, using pre-trained model for transfer learning, the accuracy significantly increases. Finally, using Inception V4 with transfer learning results in high accuracy. This signifies the value of transfer learning, where using even a relatively smaller training dataset can result in an excellent automated system for AD detection.



2.Dataset

Description

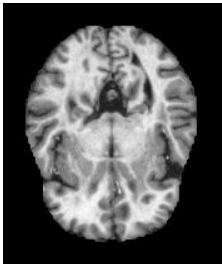
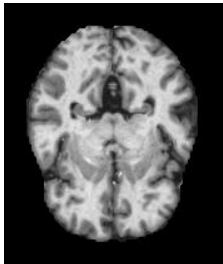
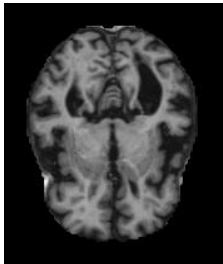
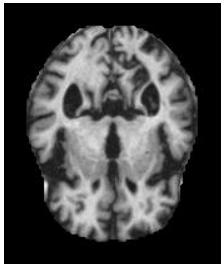
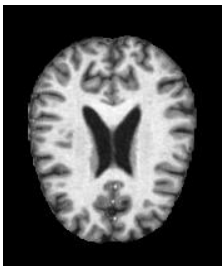

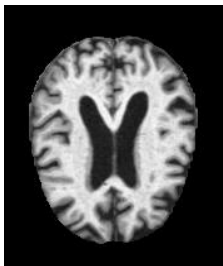
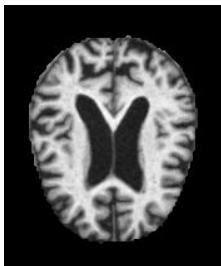
Data may be found at:

<https://www.kaggle.com/datasets/tourist55/alzheimers-dataset-4-class-of-images>

The Data was hand collected from various websites with each and every label verified.
It consists of 6400 MRI images and has four classes of images in both training and testing sets:

Number of Images per Set and Stage	Non Demented	Very Mild Demented	Mild Demented	Moderate Demented
Train (5121 images)	2560	1792	717	52
Test (1279 images)	640	448	179	12

Examples

Example of Images	Non Demented	Very Mild Demented	Mild Demented	Moderate Demented
Train				
Test				



Dataset Issues

Size related issues

Volume

We compared our data size with the size of the Oasis MRI dataset (used in the “*Towards Alzheimer’s Disease Classification through Transfer Learning*” article):

The Oasis dataset comprises 416 + 20 nondemented subjects.

Our data seems larger in comparison.

Yet, we still believe the Pre-training approach used in the article will help us here as well, since:

- Some of the categories on the training dataset are not big enough for accurate training (for example - Moderate Demented only contains 52 images)
- Our GPU resources are limited (even when using Google Colab) and we wish to be as quick and efficient as possible on the training phase.
- The available pre-trained models are state-of-the-art models, optimized with the most suitable hyperparameters and (as seen in the article) yield better accuracies overall.

Another way to increase the volume of the data is by using Data Augmentation (see “*Methods*” in the “*Process*” section).

Imbalance

Imbalanced training data is an issue that has to be addressed when dealing with classification.

The train dataset is very imbalanced - ratio is:

49.2 (Non Dem.) : 34.5 (Very Mild Dem.) : 13.8 (Mild Dem.) : 1 (Moderate Dem.)

We will try and correct this imbalance by using 2 techniques (see full description on “*Methods*” in the “*Process*” section):

1. Applying SMOTE on Train Dataset - Synthetic Minority Oversampling Technique.
2. Adjusting the class_weight hyperparameter when fitting our models.

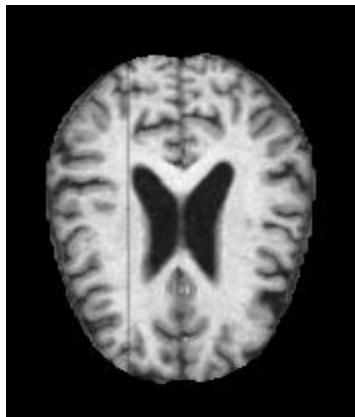


Visual issues

When “eyeballing” the data, we can see that it looks relatively clean but there are still some apparent issues:

Noise

For example - the black vertical line on the 26 (54).jpg file:



Since this example was found on the Test dataset (on the Very Mild Demented Class), we decided it should stay as part of our data.

Shade intensity

Some of the images are darker than others: for example - the image given as an example for the Mild Demented class from the Train Dataset (see “Examples” section above) has a darker shade compared to the other images. Since this dark shade doesn’t seem to be region-specific, we concluded that some preprocessing should probably be done.

Train vs. Test data

Probably the most apparent visual problem on the “*Examples*” section above is the difference between the Train set and the Test set. Since we are not domain experts in brain imaging, we can only try and describe this as “butterfly-shaped black holes” in the middle of the Test dataset examples (see table above), which we do not see on the Train dataset examples. At first, when we looked at the **full** train and test sets, we didn’t quite grasp this difference, since there are some images on the full Train dataset that do have this special shape on them. We considered looking at additional data (from other sources) or even merge and resplit the data into 2 new Train and Test datasets to gain greater similarity between the sets.



3.Process

End-to-end Deep Learning Pipeline:

Load

- All data was downloaded to our Google Drive from:
<https://www.kaggle.com/datasets/tourist55/alzheimers-dataset-4-class-of-images>
- Load images in grayscale from Train and Test directories (set on Google Drive)

Preprocess

Transforming the Images: (done on both Train and Test datasets)

- Remove image noise using OpenCV bilateralFilter (which reduces unwanted noise while keeping edges fairly sharp)
- Produce a pseudo-colored image using applyColorMap from OpenCV.
- Resize images.
- Normalize Images into the range 0 to 1.
- Categorize images to classes 0 to 3.

Taking care of training data Size and Imbalance:

- At this stage we thought about using some more data from other datasets ([ADNI](#) for example - The Alzheimer's Disease Neuroimaging Initiative), since some classes on our train dataset didn't seem to have enough data on them. But first, we tried some coding approaches:
 - Create synthetic images by applying SMOTE (see *Methods*). This should add more samples to the minority class and deal with the imbalance.
 - Use Data Augmentation (see *Methods*) to create additional images by rotating the original images (in some notebooks we also used range shifts and flips). This is not meant to fix the imbalance as the new images are added across all classes, but it does give us more data to learn from.

Taking care of the difference between Train and Test datasets:

- As mentioned in the *Dataset Issues* section, there is some kind of inexplicable difference in how the train data and the test data look. This became even more suspicious once we got to the test and evaluation stages of our work (see *Evaluation* below) - while validation accuracy reached 99%, test accuracy reached only 82% on our best try. At this stage we mixed together the Train and Test datasets (by merging the datasets into one directory and shuffling), and then re-split them into new train(4106 images) - val(1027 images) - test(1284 images) datasets - almost the same proportions as the original Train and Test datasets.



Short EDA:

- Show volumes of datasets before merge.
- Show volumes of datasets after merge.
- Show some examples of images (after preprocessing) with labeling.

Building the Neural Networks (Using Transfer Learning)

We decided to use transfer learning to build our networks (as it was proved this way is the most effective in terms of training time, handling the complexity and low volume of our data and reaching high accuracies - see *Prior Research* section and *Dataset Issues - Volume* section).

We used the following workflow (as mentioned in the *Methods* section):

Take layers from a previously trained model:

We used 5 different pretrained base models - VGG16, ResNet50, InceptionV3, ResNet152V2 and InceptionResNetV2 (for more information about these models - see *Base Models' Architecture* section on Appendix). We chose these models since three of them - VGG, ResNet and Inception - were mentioned on the article we read on our prior research (^{Ref.4})

Freeze them:

This is done to avoid destroying any of the information they contain during future training rounds (such as the weights).

Remove the top layer:

We removed the top layer of the pretrained model. For example:

- in VGG16 we removed the last fully connected layer which classifies the data into 1000 classes (and uses the Softmax activation function).
- in ResNet50 we removed the last AvgPool + fully connected layer (classifying the data into 1000 classes).

Add some new trainable layers on top of the frozen layers:

On top of our base models, we added the following layers -

- A GlobalAveragePooling2D layer (in the ResNet models it just replaced the old one). This layer is important since it reduces the very large number of parameters.
- In some of the experiments we added 2 additional layers: a Dropout layer to prevent overfitting (we chose to drop 50% of the neurons) and a fully connected layer (with ReLU activation function).
- Finally, we replaced the old top fc layer with a new one - this time, classifying into our 4 classes (we used Softmax as in the original models).

Train the new layers on our dataset:

The models will learn to turn the old features into predictions on our new and small dataset.



Model Compilation

- Optimizers (For more info on Adam vs. SGD - see *Methods* section on Appendix):
 - We mostly used Adam to optimize our model (with a learning rate of 0.0001).
 - In a few experiments we used SGD (with a learning rate of 0.01).
- Note: We also used ReduceLROnPlateau in the training process, to adjust our learning rate when we saw that the validation loss didn't improve for a few epochs (see *Methods*).
- Loss Function: We used Categorical Crossentropy as our loss function. It's a softmax activation plus a Cross-Entropy loss used for multiclass classification. Using this loss, we can train our network to output a probability over the 4 classes for each image.

Training

- To reach the best training, we used 100 epochs with the early stopping method (described in *Methods* section - *Training Epochs with Early Stopping*):
 - We created a model checkpoint callback to save the best model during training.
 - We created an early stopping callback to stop the training if the validation loss doesn't improve for 5 epochs (later, we increased this to 8 epochs)
 - We have created a reduced learning rate on plateau callback to reduce the learning rate by 0.2 if the validation loss doesn't improve for 3 epochs.
- To fix the imbalance on our train dataset, we experimented with different class weights:
 - In some experiments we used the 1:1:1:1 default
 - In some - we calculated the "balanced" weights (see calculation on *Methods*)
 - And finally, we also manually adjusted the weights until reaching the best ratio.

Evaluation (for more info - see *Best Models, Learning Curves and Classification Reports*)

- Training and Validation: we looked at the following curves to understand and improve our experiments (by changing the hyperparameters to get optimal results) -
 - Show Loss curves - Loss vs. Epoch.
 - Show Accuracy curves - Accuracy vs. Epoch.
- Test:
 - We applied our model on the Test dataset.
 - We then examined the Classification Report: we noticed the huge gap between the validation and test accuracies and the low recalls (especially on the MildDemented class), which made us look for some imbalance fixes (as described above).
 - Finally, we resorted to merging our Train and Test datasets, shuffling and re-splitting the unified dataset into new train-val-test datasets. We re-ran our models: this time reaching test accuracies above 93% for all of our models and highly balanced precision and recall scores (and our best result - 99.53% test accuracy on our InceptionV3 based model).



4. Experiments and Results

Table of experiments

Notes:

- Accuracies can be found on the table. For more metrics - see *Best Models* on Appendix
- For Base Models' Architectures (including top layer) - see *Base Models' Architectures*
- Class weights ratio is presented in the following order:
'MildDemented', 'ModerateDemented', 'NonDemented', 'VeryMildDemented'

Base Model	Additional Layers (On top of base model instead of original top layer)	Batch Size	Epochs	Data Aug?	Smote ?	Class Weights	Other manipulations	Test accuracy %
VGG16	GlobalAveragePooling2D() Dense(4, activation=SOFTMAX)	20	14	Rot. range 10	No	1:1:1:1	optimizer=adam learning_rate=0.0001	74.98
ResNet50	GlobalAveragePooling2D() Dense(4, activation=SOFTMAX)	20	15	Rot. range 10	No	1:1:1:1	optimizer=adam learning_rate=0.0001	82.72
ResNet50	GlobalAveragePooling2D() Dense(4, activation=SOFTMAX)	20	13	Rot. range 30 W. shift=0.05 H. shift=0.05 H. flip=True	No	1:1:1:1	optimizer=adam learning_rate=0.0001	76.54
ResNet50	GlobalAveragePooling2D() Dense(4, activation=SOFTMAX)	20	18	Rot. range 10	No	1:1:1:1	optimizer= SGD learning_rate= 0.01	79.59
ResNet50	GlobalAveragePooling2D() Dropout(0.5) Dense(1024, activation=RELU) Dense(4, activation=SOFTMAX)	20	15	Rot. range 10	No	1:1:1:1	optimizer=adam learning_rate=0.0001	76.62
ResNet50	GlobalAveragePooling2D() Dropout(0.5) Dense(1024, activation=RELU) Dense(4, activation=SOFTMAX)	16	15	Rot. range 10	No	1:1:1:1	optimizer=adam learning_rate=0.0001	79.05
ResNet50	GlobalAveragePooling2D() Dropout(0.5) Dense(1024, activation=RELU) Dense(4, activation=SOFTMAX)	20	16	Rot. range 10	No	1.79 :24.62 :0.5 :0.71 (balanced)	optimizer=adam learning_rate=0.0001	75.84
ResNet50	GlobalAveragePooling2D() Dense(4, activation=SOFTMAX)	20	12	Rot. range 10	No	1:5:1:1.5	optimizer=adam learning_rate=0.0001	79.20
ResNet50	GlobalAveragePooling2D() Dropout(0.5) Dense(1024, activation=RELU) Dense(4, activation=SOFTMAX)	20	18	Rot. range 10	No	1:5:1:1.5	optimizer=adam learning_rate=0.0001	79.83



ResNet50	GlobalAveragePooling2D() Dropout(0.5) Dense(1024, activation=RELU) Dense(4, activation=SOFTMAX)	20	14	Rot. range 10	Yes	1:5:1:1.5	optimizer=adam learning_rate=0.0001	77.56
ResNet50	GlobalAveragePooling2D() Dense(4, activation=SOFTMAX)	20	17	Rot. range 10	Yes	1:1:1:1	optimizer=adam learning_rate=0.0001	78.66
ResNet50	GlobalAveragePooling2D() Dropout(0.5) Dense(1024, activation=RELU) Dense(4, activation=SOFTMAX)	20	17	Rot. range 10	Yes	1:1:1:1	optimizer=adam learning_rate=0.0001	80.53
ResNet50	GlobalAveragePooling2D() Dropout(0.5) Dense(1024, activation=RELU) Dense(4, activation=SOFTMAX)	16	11	Rot. range 10	Yes	1:1:1:1	optimizer=adam learning_rate=0.0001	75.45
InceptionV3	GlobalAveragePooling2D() Dropout(0.5) Dense(1024, activation=RELU) Dense(4, activation=SOFTMAX)	20	9	Rot. range 10	No	1:1:1:1	optimizer=adam learning_rate=0.0001	75.89
ResNet152V2	GlobalAveragePooling2D() Dropout(0.5) Dense(1024, activation=RELU) Dense(4, activation=SOFTMAX)	20	13	Rot. range 10	No	1:1:1:1	optimizer=adam learning_rate=0.0001	72.87
InceptionResNetV2	GlobalAveragePooling2D() Dropout(0.5) Dense(1024, activation=RELU) Dense(4, activation=SOFTMAX)	20	24	Rot. range 10	No	1:1:1:1	optimizer=adam learning_rate=0.0001	74.65
THE RESULTS BELOW WERE ACHIEVED BY MERGING TRAIN AND TEST DATASETS, SHUFFLING AND RE-SPLITTING (best accuracy per base model - in Green)								
VGG16	GlobalAveragePooling2D() Dense(4, activation=SOFTMAX)	20	24	Rot. range 10	No	1:5:1:1.5	Merged Train+Test, shuffled and re-split optimizer=adam learning_rate=0.0001	98.75
VGG16	GlobalAveragePooling2D() Dense(4, activation=SOFTMAX)	20	24	Rot. range 10	Yes	1:1:1:1	Merged Train+Test, shuffled and re-split optimizer=adam learning_rate=0.0001	98.91
ResNet50	GlobalAveragePooling2D() Dense(4, activation=SOFTMAX)	20	14	Rot. range 10	No	1:1:1:1	Merged Train+Test, shuffled and re-split optimizer=adam learning_rate=0.0001	98.36



ResNet50	GlobalAveragePooling2D() Dense(4, activation=SOFTMAX)	20	22	Rot. range 10	No	1:5:1:1.5	Merged Train+Test, shuffled and re-split Patience = 8 optimizer=adam learning_rate=0.0001	99.07
InceptionV3	GlobalAveragePooling2D() Dropout(0.5) Dense(1024, activation=RELU) Dense(4, activation=SOFTMAX)	32	37	Rot. range 10	No	1:1:1:1	Merged Train+Test, shuffled and re-split Patience = 5 optimizer=adam learning_rate=0.0001	98.21
InceptionV3	GlobalAveragePooling2D() Dense(4, activation=SOFTMAX)	20	15	Rot. range 10	No	1:1:1:1	Merged Train+Test, shuffled and re-split Patience = 5 optimizer=adam learning_rate=0.0001	99.53
ResNet152V2	GlobalAveragePooling2D() Dropout(0.5) Dense(1024, activation=RELU) Dense(4, activation=SOFTMAX)	32	15	Rot. range 10	No	1:1:1:1	Merged Train+Test, shuffled and re-split Patience = 5 optimizer=adam learning_rate=0.0001	94.78
InceptionResNetV2	GlobalAveragePooling2D() Dropout(0.5) Dense(1024, activation=RELU) Dense(4, activation=SOFTMAX)	32	14	Rot. range 10	No	1:1:1:1	Merged Train+Test, shuffled and re-split Patience = 5 optimizer=adam learning_rate=0.0001	93.92

Test Results Comparison

- The best test accuracy - 99.53% was reached with our InceptionV3 based model.
- The biggest accuracy improvement was made when we decided to merge and resplit the Train and Test data sets.
- Also - we can see better performance of ResNet50, VGG16 and Inception V3, compared to the ResNet152V2 and InceptionResNetV2 best runs.
- Adding the Dropout layer and an extra fully connected layer (the one with the ReLU activation function) didn't improve the model and sometimes even harmed it.
- Adam got us better results than SGD.
- Running with up to 100 epochs while using Early Stopping and Reduce LR On Plateau techniques proved itself very effective.
- Adding more permutations of Data Augmentation didn't improve the models, and it was enough to use a Rotation range of 10.
- SMOTE didn't significantly improve the results.



- The calculated “balanced” class weights didn’t improve the results, while our manually adjusted class weights were able to achieve better results (by trial and error).

5. Discussion

Conclusions

In this project, we propose a transfer learning-based method to detect AD stages from structural MRI images. We test 5 popular architectures: VGG16, ResNet50, Inception V3, ResNet152V2 and InceptionResNetV2. Through pre-trained weights from ImageNet and some additional layers, we can only use a small number of training images to obtain highly accurate results. Moreover, we employ some techniques to clean, augment and balance our data in order to get the most value from our small dataset. Our method proved itself right, providing performance comparable to state-of-the-art models despite having a training set many times smaller.

To reproduce our research, our models, dataset and code can be accessed through our drive at: <https://drive.google.com/drive/folders/17wtRXwCr6CQn9EcioSrQDNgWSuwysill?usp=sharing>

Future Work

Now that we were able to reach high accuracies on MRI classification of Alzheimer’s disease, using pretrained model, we can try and do the following:

- Apply our best models on MRI images of other major neurodegenerative diseases associated with dementia, including frontotemporal lobar degeneration, dementia with Lewy bodies (LBD - as seen on our introduction), and Huntington’s disease. This way, with some fine tuning such as changing the final layer number of classes, we might be able to get high accuracies for these other disease classifications as well.
- Go beyond dementia and try to classify with our models other diseases and disorders that affect the brain. For example, take the brain MRI dataset from ABIDE and try to classify Autism spectrum disorder ([ABIDE](#) - The Autism Brain Imaging Data Exchange).

6. References

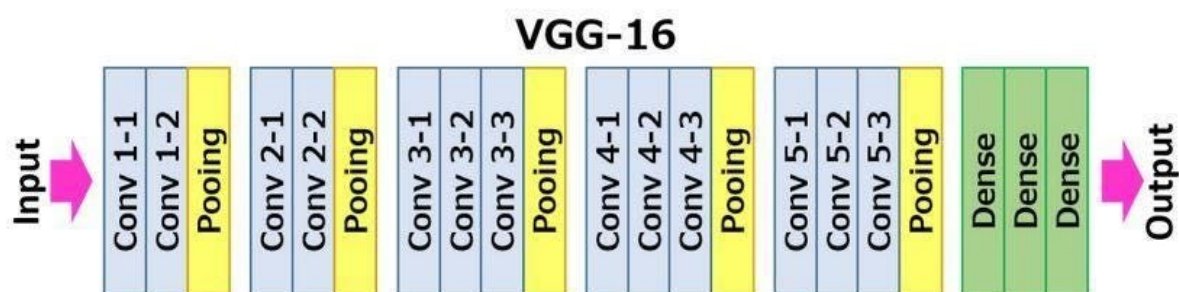
1. Jalbert J, Daiello L, Lapane K. Dementia of the Alzheimer Type. Epidemiol Rev. 2008;30(1):15-34. doi:10.1093/epirev/mxn008 - Pubmed
2. Miller-Thomas M, Sipe A, Benzinger T, McConathy J, Connolly S, Schwetye K. Multimodality Review of Amyloid-Related Diseases of the Central Nervous System. Radiographics. 2016;36(4):1147-63. doi:10.1148/rg.2016150172 - Pubmed
3. Wei Hui, Masurkar Arjun V., Razavian Narges On gaps of clinical diagnosis of dementia subtypes: A study of Alzheimer’s disease and Lewy body disease (2023 in Frontiers in Aging Neuroscience) doi.org/10.3389/fnagi.2023.1149036 - Frontiers
4. M. Hon and N. Khan, "Towards Alzheimer's disease classification through transfer learning," in 2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Kansas City, MO, USA, 2017 pp. 1166-1169. doi: 10.1109/BIBM.2017.8217822 - IEEE
5. <https://datagen.tech/guides/computer-vision/vgg16/#>
6. <https://machinelearningmastery.com/>
7. <https://www.analyticsvidhya.com/>
8. <https://www.datacamp.com/>
9. https://www.tensorflow.org/guide/keras/transfer_learning

Appendix

Base Models' Architectures Ref.5

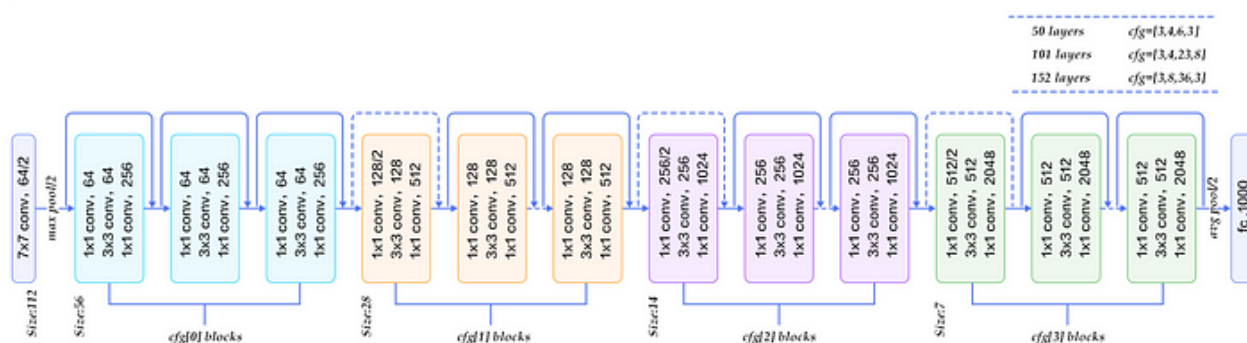
VGG-16

VGG16 refers to the VGG model, also called VGGNet. It is a convolution neural network (CNN) model supporting 16 layers. The VGG16 model can achieve a test accuracy of 92.7% in ImageNet, a dataset containing more than 14 million training images across 1000 object classes. VGG16 improves on AlexNet and replaces the large filters with sequences of smaller 3×3 filters. In AlexNet, the kernel size is 11 for the first convolutional layer and 5 for the second layer. The researchers trained the model for several weeks using NVIDIA Titan Black GPUs.



ResNet-50

ResNet-50 is a 50-layer convolutional neural network (48 convolutional layers, one MaxPool layer, and one average pool layer). Residual neural networks are a type of artificial neural network (ANN) that forms networks by stacking residual blocks. The model uses the “Skip Connection” technique - Adding the original input to the output of the convolutional block.





Inception-V3

InceptionV3 is a convolutional neural network (CNN) model developed for image classification. It was introduced by Google researchers in 2015 as an evolution of earlier Inception models. The main idea behind InceptionV3 is to create a network architecture that performs efficient and accurate feature extraction by utilizing multiple different convolutional filters at each layer.

The model is composed of multiple layers, including convolutional layers, pooling layers, and fully connected layers. It employs a unique "Inception module" that combines multiple filter sizes (1x1, 3x3, and 5x5) within a single layer to capture different levels of abstraction in the input image. This helps the model to efficiently process information at various scales and capture both fine-grained and high-level features.

InceptionV3 also incorporates other techniques like batch normalization and factorized convolutions to improve training and inference efficiency. It includes auxiliary classifiers at intermediate layers, which provide additional supervision during training to alleviate the vanishing gradient problem and improve the gradient flow.

The model was pretrained on a large dataset called ImageNet, which contains millions of labeled images across thousands of classes. As a result, InceptionV3 has learned to recognize a wide range of objects and patterns, making it suitable for tasks like image classification, object detection, and feature extraction. It has achieved high accuracy on benchmark datasets and has been widely used as a base model for various computer vision applications.

ResNet-152-V2

ResNet-152-V2 is a deep convolutional neural network (CNN) model that belongs to the ResNet (Residual Network) family. It was introduced by Microsoft Research in 2016 as an enhancement to the original ResNet architecture. The primary objective of ResNet-152-V2 is to address the problem of vanishing gradients and enable the training of very deep neural networks.

The "ResNet" name comes from the concept of residual learning, which introduces skip connections or shortcuts in the network. These connections allow the model to bypass certain layers and directly propagate information from earlier layers to later layers. By doing so, ResNet-152-V2 overcomes the degradation problem that occurs when the network becomes too deep, making it easier to train and improving performance.

ResNet-152-V2 consists of 152 layers, including convolutional layers, pooling layers, and fully connected layers. The model adopts a "bottleneck" structure, where 1x1 convolutional layers are used to reduce the dimensionality of the input, followed by 3x3 convolutions for feature extraction. This design reduces computational complexity while capturing meaningful features.



Additionally, ResNet-152-V2 utilizes batch normalization and rectified linear unit (ReLU) activations to improve the training process and alleviate the vanishing gradient problem. It also incorporates residual blocks with identity shortcuts, allowing the network to learn residual functions and focus on the incremental changes in the features.

ResNet-152-V2 has been pretrained on large-scale datasets such as ImageNet, enabling it to recognize a wide variety of objects and patterns. The model has achieved state-of-the-art performance on various image recognition tasks, including image classification and object detection. It serves as a powerful tool for deep learning practitioners working in computer vision domains, providing high accuracy and strong feature representation capabilities.

Inception-ResNet-V2

Inception-ResNet-V2 is a deep convolutional neural network (CNN) model that combines the Inception architecture with residual connections from the ResNet model. It was introduced by Google researchers as an advancement of both the Inception and ResNet models, aiming to leverage the benefits of both architectures.

The model inherits the Inception concept of using multiple filters of different sizes within a single layer to capture diverse features at different scales. It employs the Inception module, which consists of parallel convolutional layers with varying filter sizes (1x1, 3x3, and 5x5) and pooling operations. This allows the model to effectively extract features of different complexities and resolutions.

Inception-ResNet-V2 also integrates the residual connections from the ResNet architecture. These connections enable the direct propagation of information from earlier layers to later layers, mitigating the vanishing gradient problem and facilitating the training of very deep networks. By combining residual connections with the Inception modules, the model achieves improved training efficiency and performance.

The model incorporates additional techniques such as batch normalization, factorized convolutions, and global average pooling to enhance its capabilities. It employs aggressive use of factorized convolutions, which factorize large convolutions into smaller convolutions, reducing computational complexity without compromising accuracy.

Inception-ResNet-V2 has been pretrained on large-scale datasets like ImageNet, enabling it to recognize a broad range of objects and patterns. The model has achieved state-of-the-art results on various image recognition tasks, including image classification, object detection, and feature extraction. It serves as a powerful tool for computer vision applications, offering a strong combination of accuracy, efficiency, and the ability to capture features at multiple scales.



Methods

Transfer Learning with Image Data Ref.6, Ref.9

It is common to perform transfer learning with predictive modeling problems that use image data as input.

It may be a prediction task that takes photographs or video data as input.

For these types of problems, it is common to use a deep learning model pre-trained for a large and challenging image classification task such as the [ImageNet](#) 1000-class photograph classification competition.

This approach is effective since the models were trained on a large corpus of photographs and make predictions on a relatively large number of classes, in turn, requiring that the model efficiently learn to extract features from photographs in order to perform well on the problem.

The most common incarnation of transfer learning in the context of deep learning is the following workflow:

1. Take layers from a previously trained model.
2. Freeze them, so as to avoid destroying any of the information they contain during future training rounds.
3. Add some new, trainable layers on top of the frozen layers. They will learn to turn the old features into predictions on a new dataset.
4. Train the new layers on your dataset.

Adjusting Class Weights Ref.7

Most machine learning algorithms are not very useful with biased class data. But, we can modify the current training algorithm to take into account the skewed distribution of the classes. This can be achieved by giving different weights to both the majority and minority classes. The difference in weights will influence the classification of the classes during the training phase.

The whole purpose is to penalize the misclassification made by the minority class by setting a higher class weight and at the same time reducing weight for the majority class.

By default, the value of `class_weight=None`, i.e. both the classes have been given equal weights. Other than that, we can either give it as 'balanced' or we can pass a dictionary that contains manual weights for both the classes.

When the `class_weights = 'balanced'`, the model automatically assigns the class weights inversely proportional to their respective frequencies.

To be more precise, the formula to calculate this is: $w_j = n_samples / (n_classes * n_samples_j)$

Note: Passing the value as 'balanced' gives good results in most cases but sometimes for extreme class imbalance, we can try giving weights manually.



Data Augmentation Ref.8

Data augmentation is a technique of artificially increasing the training set by creating modified copies of a dataset using existing data. It includes making minor changes to the dataset or using deep learning to generate new data points.

Augmented data is driven from original data with some minor changes. In the case of image augmentation, we make geometric and color space transformations (flipping, resizing, cropping, brightness, contrast) to increase the size and diversity of the training set.

When Should You Use Data Augmentation?

- To prevent models from overfitting.
- The initial training set is too small.
- To improve the model accuracy.
- To Reduce the operational cost of labeling and cleaning the raw dataset.

Limitations of Data Augmentation

- The biases in the original dataset persist in the augmented data.
- Finding an effective data augmentation approach can be challenging.

SMOTE Ref.6

A problem with imbalanced classification is that there are too few examples of the minority class for a model to effectively learn the decision boundary.

One way to solve this problem is to oversample the examples in the minority class. This can be achieved by simply duplicating examples from the minority class in the training dataset prior to fitting a model. This can balance the class distribution but does not provide any additional information to the model.

An improvement on duplicating examples from the minority class is to synthesize new examples from the minority class. This is a type of data aug. for tabular data and can be very effective.

Perhaps the most widely used approach to synthesizing new examples is called the Synthetic Minority Oversampling TEchnique, or SMOTE for short. This technique was described by Nitesh Chawla, et al. in their 2002 paper named for the technique titled ["SMOTE: Synthetic Minority Over-sampling Technique."](#)

SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line. Specifically, a random example from the minority class is first chosen. Then k of the nearest neighbors for that example are found (typically $k=5$). A randomly selected neighbor is chosen and a synthetic example is created at a randomly selected point between the two examples in feature space. This procedure can be used to create as many synthetic examples for the minority class as are required. As described in the paper, it suggests first using random undersampling to trim the number of examples in the majority class, then using SMOTE to oversample the minority class to balance the class distribution.

The approach is effective because new synthetic examples from the minority class are created that are relatively close in feature space to existing examples from the minority class.



A general downside of the approach is that synthetic examples are created without considering the majority class, possibly resulting in ambiguous examples if there is a strong overlap for the classes. Also, we can get some sample overlapping, noise interference, and blindness of neighbor selection which might actually decrease the performance. It is also heavy in terms of GPU usage.

Optimizers - SGD vs. Adam Ref.7

SGD is a very basic algorithm and is hardly used in applications now due to its slow computation speed. One more problem with that algorithm is the constant learning rate for every epoch. Moreover, it is not able to handle saddle points very well.

The Adam optimizer inherits the good features of RMSProp and other algorithms. The results of the Adam optimizer are generally better than every other optimization algorithm, have faster computation time, and require fewer parameters for tuning. Because of all that, Adam is recommended as the default optimizer for most of the applications. Choosing the Adam optimizer for your application might give you the best probability of getting the best results. But, even Adam optimizer has some downsides and there are cases when algorithms like SGD might be beneficial and perform better than Adam optimizer (it all depends on the data).

Training Epochs with Early Stopping Ref.6

A problem with training neural networks is in the choice of the number of training epochs to use. Too many epochs can lead to overfitting of the training dataset, whereas too few may result in an underfit model. Early stopping is a method that allows you to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a hold out validation dataset.

Callbacks provide a way to execute code and interact with the training model process automatically. Callbacks can be provided to the `fit()` function via the “callbacks” argument. The loss function chosen to be optimized for your model is calculated at the end of each epoch. To callbacks, this is made available via the name “loss.”

Often, the first sign of no further improvement may not be the best time to stop training. This is because the model may coast into a plateau of no improvement or even get slightly worse before getting much better: We can account for this by adding a delay to the trigger in terms of the number of epochs on which we would like to see no improvement. This can be done by setting the “patience” argument.

We can also use patience with `ReduceOnPlateau`: `ReduceLROnPlateau` is a scheduling technique that decreases the learning rate when the specified metric stops improving for longer than the patience number allows. Thus, the learning rate is kept the same as long as it improves the metric quantity, but the learning rate is reduced when the results run into stagnation.

The `EarlyStopping` callback will stop training once triggered, but the model at the end of training may not be the model with best performance on the validation dataset.

An additional callback is required that will save the best model observed during training for later use. This is the `ModelCheckpoint` callback. We will use it to save the best model observed during training as defined by a chosen performance measure on the validation dataset.



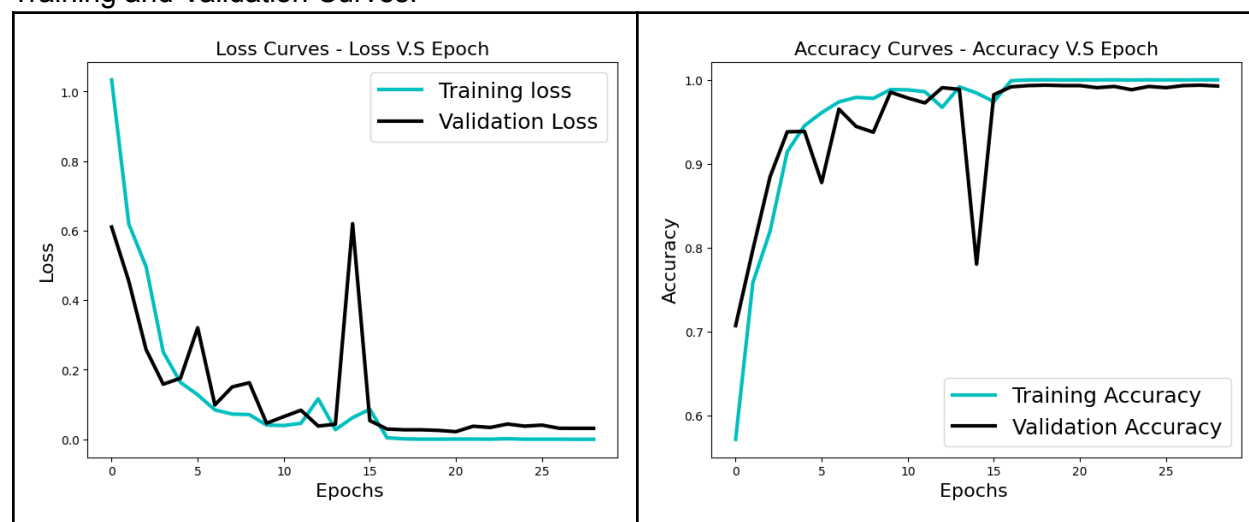
Best Models, Learning Curves and Classification Reports

Best VGG-16 based model

Model Summary:

Base Model	Additional Layers (On top of base model instead of original top layer)	Batch Size	Epochs	Data Aug?	Smote ?	Class Weights	Other manipulations	Test accuracy %
VGG16	GlobalAveragePooling2D() Dense(4, activation=SOFTMAX)	20	24	Rot. range 10	Yes	1:1:1:1	Merged Train+Test, shuffled and re-split optimizer=adam learning_rate=0.0001	98.91

Training and Validation Curves:



Test Classification Report:

	precision	recall	f1-score	support
0	0.99	0.98	0.99	179
1	1.00	1.00	1.00	10
2	0.98	1.00	0.99	664
3	1.00	0.98	0.99	431
accuracy			0.99	1284
macro avg	0.99	0.99	0.99	1284
weighted avg	0.99	0.99	0.99	1284

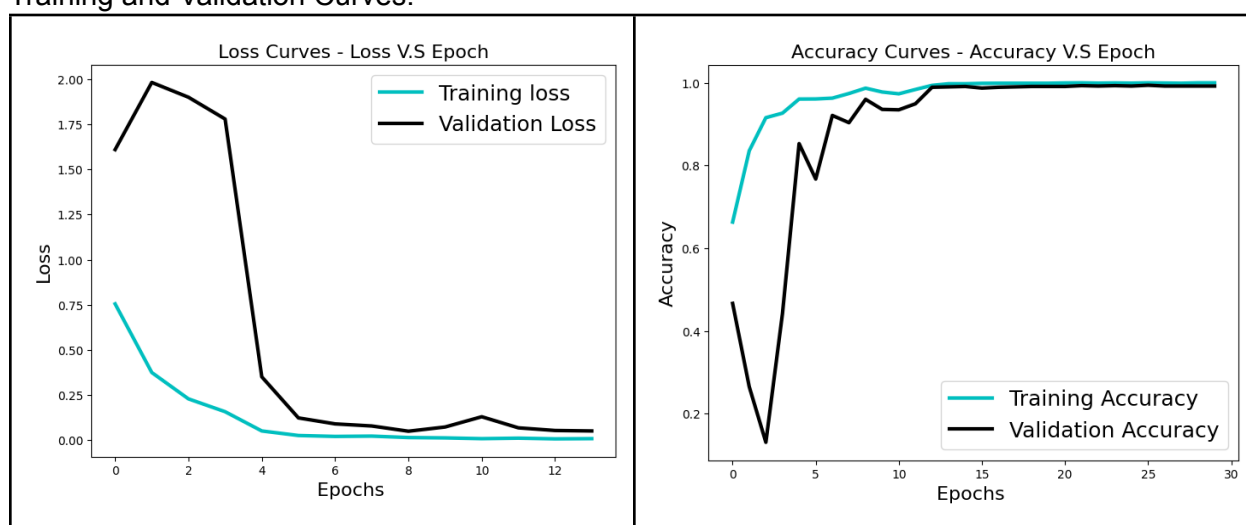


Best ResNet-50 based model

Model Summary:

Base Model	Additional Layers (On top of base model instead of original top layer)	Batch Size	Epochs	Data Aug?	Smote ?	Class Weights	Other manipulations	Test accuracy %
ResNet50	GlobalAveragePooling2D() Dense(4, activation=SOFTMAX)	20	22	Rot. range 10	No	1:5:1:1.5	Merged Train+Test, shuffled and re-split Patience = 8 optimizer=adam learning_rate=0.0001	99.07

Training and Validation Curves:



Test Classification Report:

	precision	recall	f1-score	support
0	1.00	0.99	0.99	179
1	1.00	1.00	1.00	10
2	0.99	1.00	0.99	664
3	0.99	0.98	0.99	431
accuracy			0.99	1284
macro avg	0.99	0.99	0.99	1284
weighted avg	0.99	0.99	0.99	1284

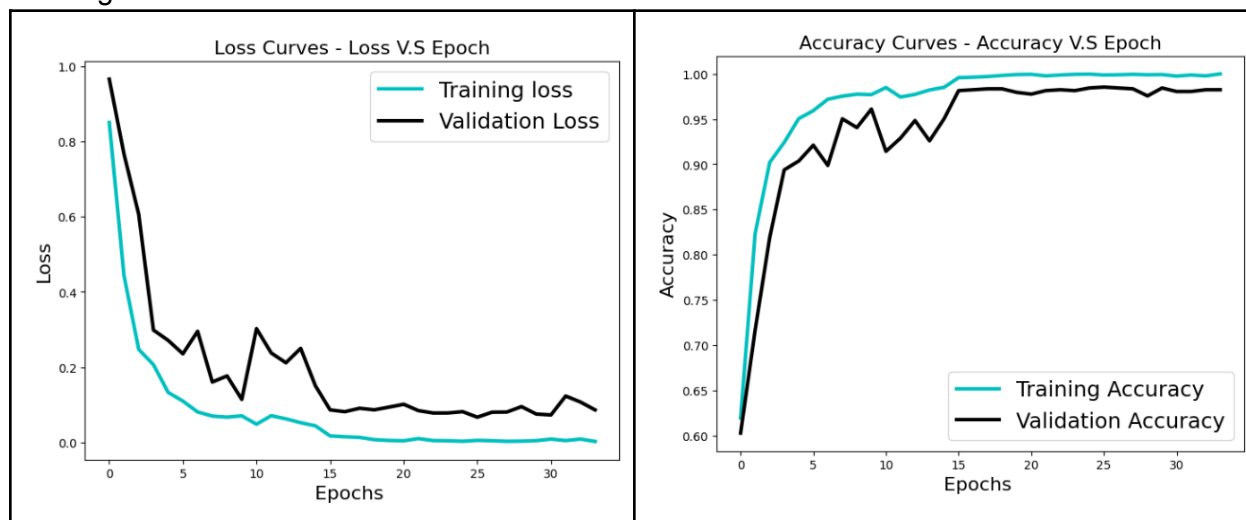


Best Inception-V3 based model

Model Summary:

Base Model	Additional Layers (On top of base model instead of original top layer)	Batch Size	Epochs	Data Aug?	Smote ?	Class Weights	Other manipulations	Test accuracy %
Inception V3	GlobalAveragePooling2D() Dense(4, activation=SOFTMAX)	20	15	Rot. range 10	No	1:1:1:1	Merged Train+Test, shuffled and re-split Patience = 5 optimizer=adam learning_rate=0.0001	99.53

Training and Validation Curves:



Test Classification Report:

	precision	recall	f1-score	support
0	0.99	1.00	0.99	167
1	0.89	1.00	0.94	8
2	1.00	1.00	1.00	670
3	1.00	0.99	0.99	438
accuracy			1.00	1283
macro avg	0.97	1.00	0.98	1283
weighted avg	1.00	1.00	1.00	1283

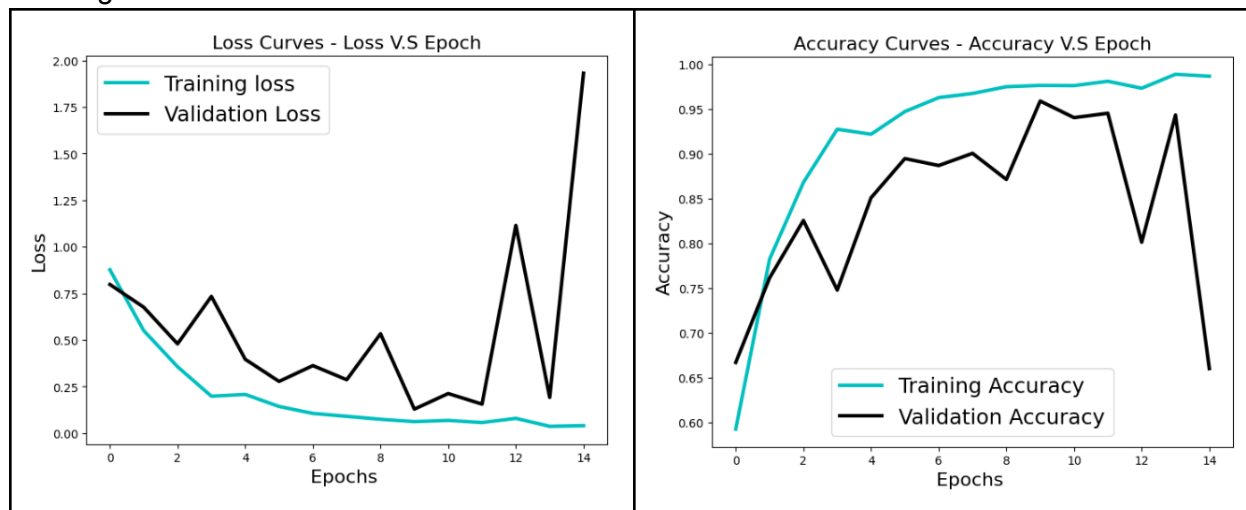


Best ResNet-152-V2 based model

Model Summary:

Base Model	Additional Layers (On top of base model instead of original top layer)	Batch Size	Epochs	Data Aug?	Smote ?	Class Weights	Other manipulations	Test accuracy %
ResNet-152-V2	GlobalAveragePooling2D() Dense(4, activation=SOFTMAX)	32	15	Rot. range 10	No	1:1:1:1	Merged Train+Test, shuffled and re-split Patience = 5 optimizer=adam learning_rate=0.0001	94.78

Training and Validation Curves:



Test Classification Report:

	precision	recall	f1-score	support
0	0.85	0.99	0.92	183
1	0.92	1.00	0.96	12
2	0.97	0.96	0.96	617
3	0.97	0.91	0.94	471
accuracy			0.95	1283
macro avg	0.93	0.97	0.95	1283
weighted avg	0.95	0.95	0.95	1283

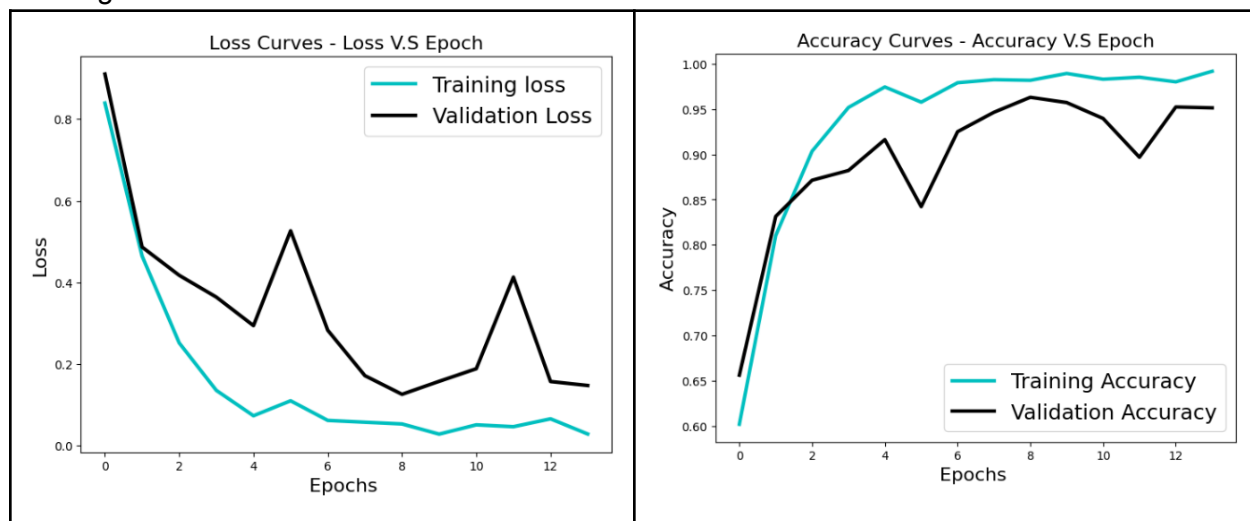


Best Inception-ResNet-V2 based model

Model Summary:

Base Model	Additional Layers (On top of base model instead of original top layer)	Batch Size	Epochs	Data Aug?	Smote ?	Class Weights	Other manipulations	Test accuracy %
Inception-ResNet-V2	GlobalAveragePooling2D() Dense(4, activation=SOFTMAX)	32	14	Rot. range 10	No	1:1:1:1	Merged Train+Test, shuffled and re-split Patience = 5 optimizer=adam learning_rate=0.0001	93.92

Training and Validation Curves:



Test Classification Report:

	precision	recall	f1-score	support
0	0.97	0.92	0.94	183
1	1.00	0.75	0.86	12
2	0.93	0.98	0.95	617
3	0.94	0.90	0.92	471
accuracy			0.94	1283
macro avg	0.96	0.89	0.92	1283
weighted avg	0.94	0.94	0.94	1283