SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

MRI Dementia Severity Classification

50.039 Deep Learning Project Report

Github: https://github.com/a-nnza-r/MCC-Dementia.git

Mohammed Ansar Ahmed	Ng Zheng Wei	Tan Pheng Yuan Ryner
1006015	1006014	1005982

1. Introduction

Problem statement: How might we use a Deep Learning model for Alzheimer's detection via the patient's MRI scans?

Addressing the problem: Build a Deep Learning model that can classify a patient's severity of Alzheimer's into the following categories (non-demented, very mild dementia, mild dementia, moderate dementia) via the use of MRI medical imaging processing.

Reason for tackling this problem: By building accurate Deep Learning models that can classify the severity of Alzheimer's Disease, we are able to detect the disease earlier and provide better treatment ahead of time. This is greatly impactful in the medical industry.

How to recreate the model:

1. git clone https://github.com/a-nnza-r/dlProj.git
2. pip install -r requirements.txt
3. Download dataset from Kaggle, unzip it and place Data file in root directory
4. Open respective notebooks and simply run cell by cell
    a. All training and evaluation code required to recreate our data is present in the notebooks
    b. We have 5 notebooks in total. 2 for CNN models, 1 for ResNet, 1 for CRNN model and 1 for visualizing the dataset and the common functions we use across the various notebooks.

2. Problem Definition and Algorithm

2.1 Task Definition

Our dataset consists of MRI scans of patients belonging to 4 categories of severity of dementia. Non-demented, very mild dementia, mild dementia and moderate dementia.

The goal of our project is to develop a model which takes in an MRI image and gives us an accurate prediction of the patient's dementia severity. Additionally, we will explore giving the model a sequence of 61 images, each from the same patient, to make a prediction on the severity of dementia of this particular patient. This is to explore CRNNs which can capture the spatial temporal relationship between the sequence of images.

In this deep learning project, we mainly considered neural network-based methods.

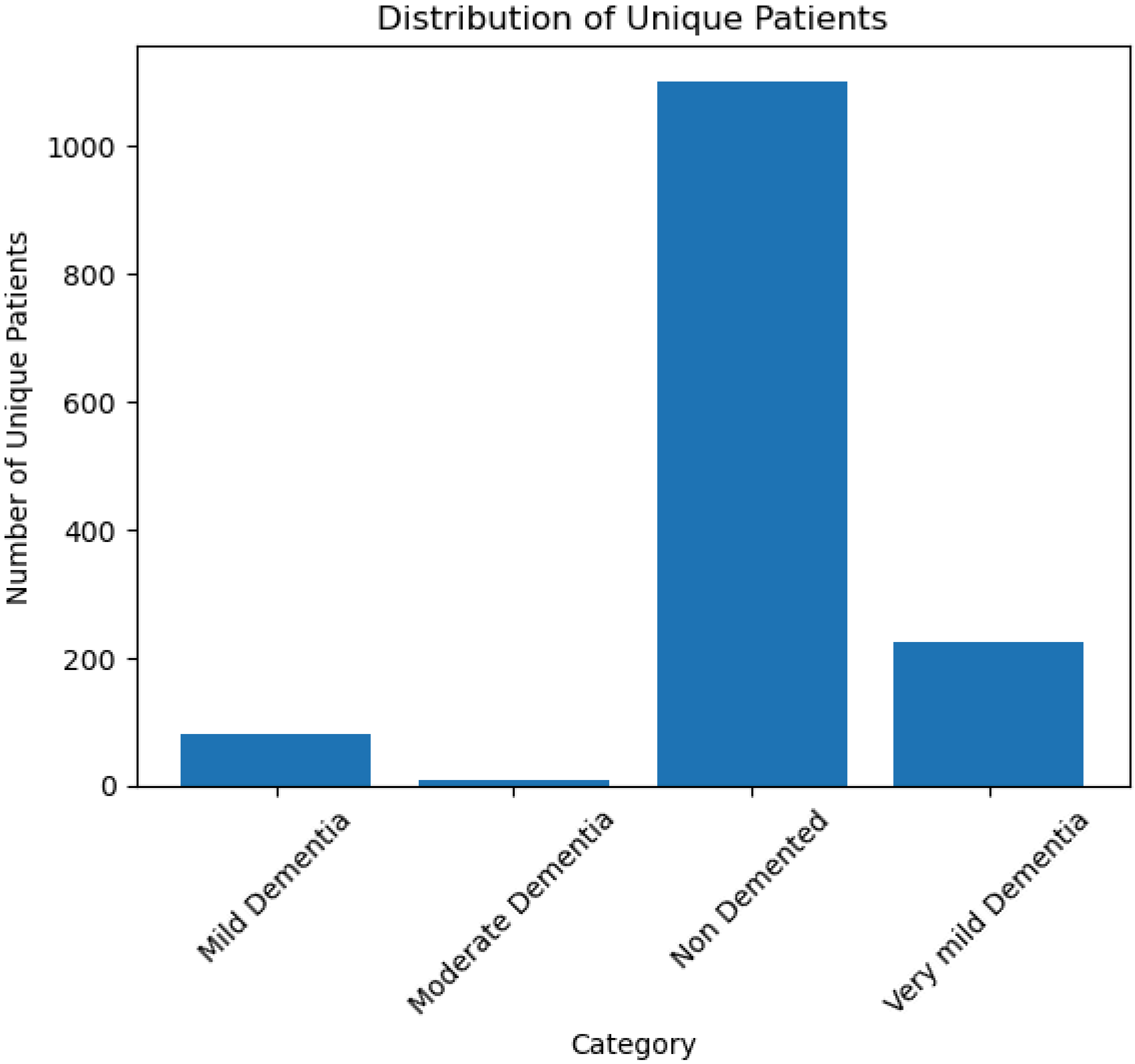
2.2 Dataset

Dataset Name: OASIS Alzheimer's Detection

Dataset Source:https://www.kaggle.com/datasets/ninadaithal/imagesoasis?resource=download-directory

Dataset size: 80,000 brain MRI images

Dimension and Format: MRI scan grayscale images (248 x 496 px) in jpg format



Class	Number of patients
Mild Dementia	82

Moderate Dementia	8
Non Demented	1102
Very mild Dementia	225

The distribution of images for each class of dementia in the dataset is not equal. We have **significantly less samples of “Moderate Dementia”** compared to the rest of the classes and significantly more samples of “Non Demented”.
If we trained our models on this dataset without resolving this imbalance, we will face issues where the model is biassed towards the majority class, “Non demented”, giving us poor generalisation while having a high accuracy.

Solution:
We used **random weighted sampling** and **random transformations** (horizontal flipping and rotations) to artificially increase the size and diversity of the training dataset by creating modified versions of existing data points.
This is a common approach that is especially valuable in fields like medical imaging, where the accurate classification of less common conditions (which might be critical to diagnosing and treating severe diseases) is as important as the more common ones.

Additional Remarks about dataset Collection and Preparation techniques:

The brain images were sliced along the z-axis into 256 pieces, and slices ranging from 100 to 160 were selected from each patient. This effectively means that we have **61 cross-section images from each patient’s** brain and their associated labels.

Patient classification was performed based on the provided metadata and Clinical Dementia Rating (CDR) values, resulting in four classes: demented, very mild demented, mild demented, and non-demented. These classes enable detecting and studying different stages of Alzheimer's disease progression.

2.3 Models

S/N	Model	Accuracy	Number of Parameters
1	CNN2 (3 Conv2d, 2 Linears) - model unable to converge due to too little parameters	-	223,497
2	SimpleCNN (3 Conv2d, 3 Linears)	97.83%	454,530

	SimpleCNN with Adversarial training	77.76%	454,530
3	BatchNormDropOutCNNModel (CNN)	98.86 %	7,168,100
	BatchNormDropOutCNNModel (CNN) with Adversarial training	97.64%	7,168,100
4	Convolutional Recurrent Neural Networks (CRNN)	98.36%	257,628
5	Residual Network (ResNet)	99.94 %	23,509,956
6	Vision Transformer (ViT) - could not train due to large number of parameters	-	85,408,516

In this project, we considered models which were suitable for image classification.

We created a benchmark with the ResNet customised for our use case, which was grayscale images with 4 classes output, and obtained an accuracy of 99.94%.

We then began our exploration with the architecture documented in our proposal, started off with an initial Convolutional Neural Network (CNN), and obtained an accuracy of 99.68%. We began to tune the hyperparameters and obtained a much better result of 99.86% in our third iteration of the CNN after some tinkering.
This final CNN model produced a performance comparable to the ResNet.

To further improve the robustness of the CNN model, we performed adversarial training on the CNN model, and discovered that accuracy changed from 99.86% to 97.64%.

To further our investigation of different architectures, we explored Convolutional Recurrent Neural Networks(CRNN) which can take advantage of the sequential information in cross-section of the brain and work better with temporal time-series data, since each patient has 61 images of their brain. We achieved 95.32% accuracy on this model.

For more details on the model and algorithm definition, please refer to Section 3.

2.3.1 Loss Function and Optimizer

For all models, we used **cross-entropy loss** as the loss function as this is a multi-class classification problem.

We used **Adam optimizer** as the default optimizer as it is highly efficient and is the defacto optimizer.

3. Experimental Evaluation

3.1 Methodology

3.1.1 Initial Benchmarking with ResNet (State of the Art)

In this project, we started off by benchmarking the separability of the dataset by using a state of the art model, ResNet, which is known for its effectiveness in image recognition tasks. The model is then adapted to the task of Alzheimer's severity classification by modifying the output layer to distinguish among four classes: non-demented, very mild dementia, mild dementia, and moderate dementia.

3.1.2 Developing a Compact CNN Model

Following the benchmarking phase, we developed a Convolutional Neural Network (CNN) with the goal of **reducing model complexity while maintaining similar levels of accuracy**. This includes fewer convolutional layers, reduced depth, and simpler connectivity, focusing on preserving the most significant features for the task.

The CNN is trained from scratch, implementing techniques such as dropout, batch normalisation, pooling and using ReLu as the activation function.

Comparison to ResNet (State of the Art):

We managed to **reduce the number of model parameters by approximately 69.5%** from 23,509,956 in ResNet to 7,168,100 in CNN, while **maintaining the accuracy** as we obtained 99.94% in ResNet compared to 98.86% in CNN (BatchNormDropOutCNNModel).

3.1.3 Experimenting with CRNN (Convolutional Recurrent Neural Network)

For further learning, we decided to implement CRNN which can take advantage of the sequential images of the cross-section of the brain, since each patient has a sequence of 61 images of the cross section of their brain.

The CRNN model combines convolutional layers with recurrent neural network layers (LSTM). The convolutional layers extract spatial features from MRI images, while the recurrent layers are designed to **capture dependencies and patterns across different sections of the image and across different images** that might be relevant for assessing the progression of Alzheimer’s disease.

The CRNN is trained from scratch, implementing techniques such as dropout, batch normalisation, pooling, skip connections and using ReLu as the activation function.

Comparison to ResNet (State of the Art):

Number of parameters used in CRNN is 257,628, **1.1% the number of model parameters used in Resnet** and 1.8% the number of model parameters compared to CNN model. CRNN obtained an **accuracy of 95.327%** which is higher than the accuracy of radiologists(65% - 95%) and comparable to the SVM results(95%) achieved in this paper[1].

3.2 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are very effective for analysing images. They are designed to learn spatial hierarchies of features present in images through backpropagation, utilising kernel masks to pick up key features present in images, and creating activation maps. After multiple convolutions, the features picked up from images form activation maps which represent higher-level features and this is used in making predictions in the final fully connected linear layers. This makes them highly suited for medical image analysis, especially in our case of classifying MRI images into different classes of severity of dementia.

3.2.1 Model Definition

We took an iterative approach to tune the hyperparameters and experiment with different model architectures. Our CNN exploration includes 3 models, namely:

- CNN2
- SimpleCNN
- BatchNormDropOutCNNModel.

Their respective architectures are shown below:

CNN2:

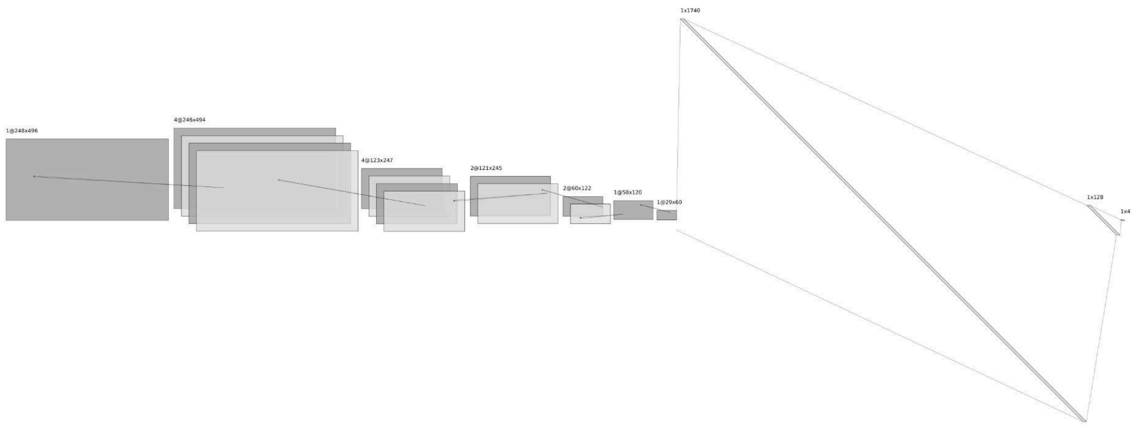


Fig: Visual of CNN2 Model

```
CNN2(
  (conv1): Conv2d(1, 4, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(4, 2, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(2, 1, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=1740, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=4, bias=True)
)
```

SimpleCNN

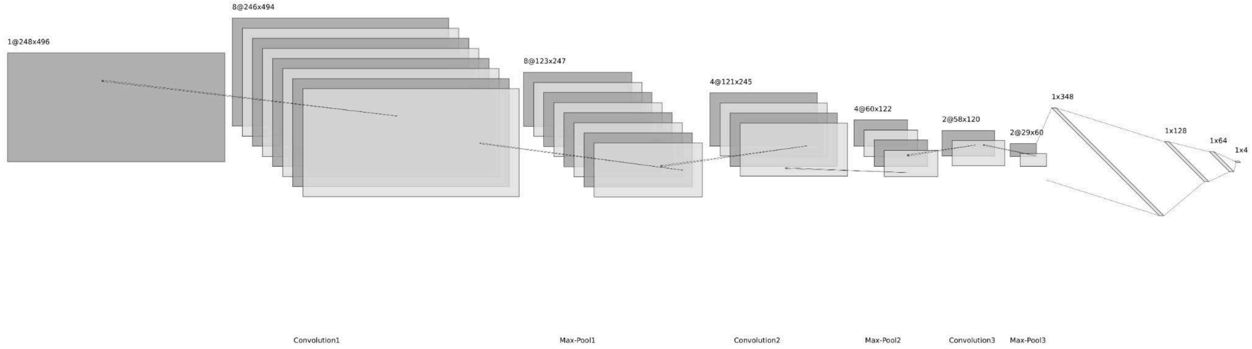


Fig: Visual of SimpleCNN (first linear layer is not accurate as image could not fit 3480 inputs)

```
SimpleCNN(
  (conv1): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(8, 4, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(4, 2, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=3480, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=4, bias=True)
)
```

BatchNormDropOutCNNModel

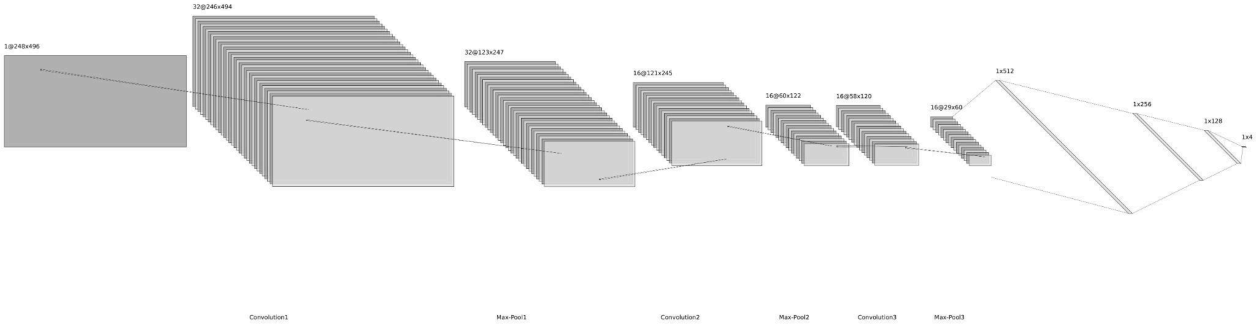


Fig: Visual of CNN model (first linear layer is not accurate as image could not fit 27,840 inputs)

```
BatchNormDropOutCNNModel(
  (conv1): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1))
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout1): Dropout2d(p=0.25, inplace=False)
  (conv2): Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1))
  (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout2): Dropout2d(p=0.25, inplace=False)
  (conv3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
  (bn3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout3): Dropout2d(p=0.25, inplace=False)
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (fc1): Linear(in_features=55680, out_features=256, bias=True)
  (dropout4): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=256, out_features=128, bias=True)
  (dropout5): Dropout(p=0.5, inplace=False)
  (fc3): Linear(in_features=128, out_features=4, bias=True)
)
```

3.2.2 Experiment Results

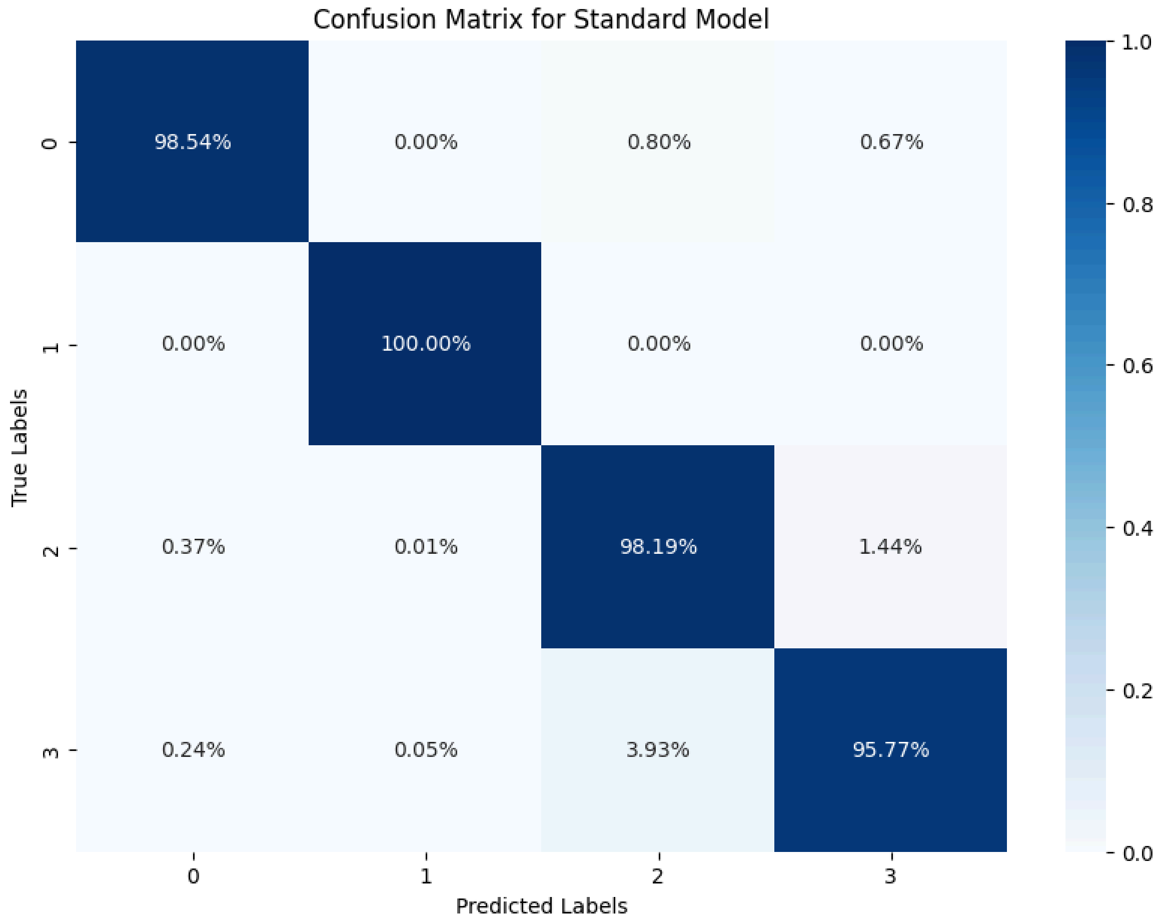
Structure:

- Summary of model performances
- Various train/validation visualisations

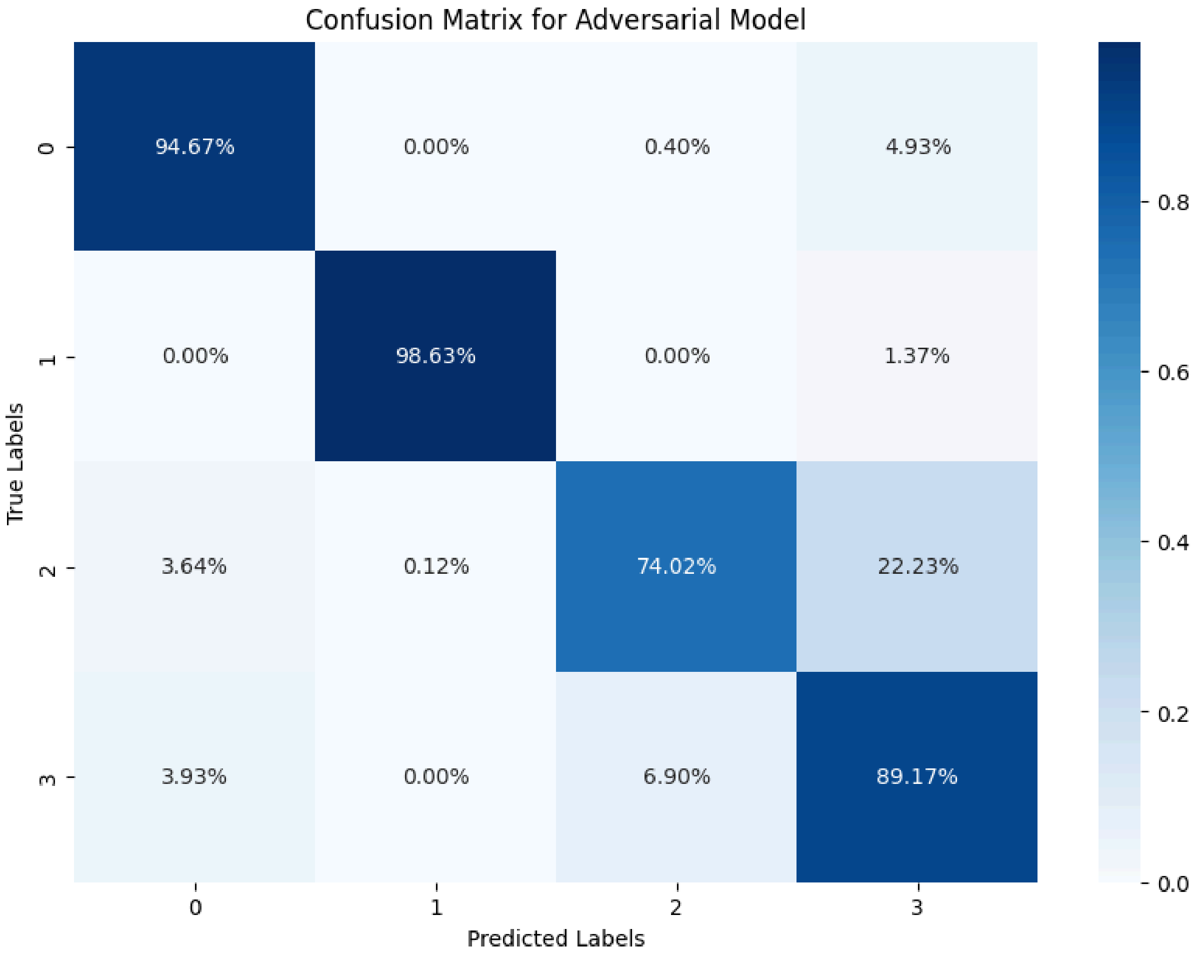
Summary of CNN model performances:

Model	Accuracy	F1 score	Loss
CNN2	**-**	**-**	**-**
SimpleCNN	**97.83%**	**0.9704**	**0.0558**
BatchNormDropOut CNNModel	**98.86%**	**0.9888**	**0.0363**

SimpleCNN



SimpleCNN with Adversarial Training



BatchNormDropOutCNNModel

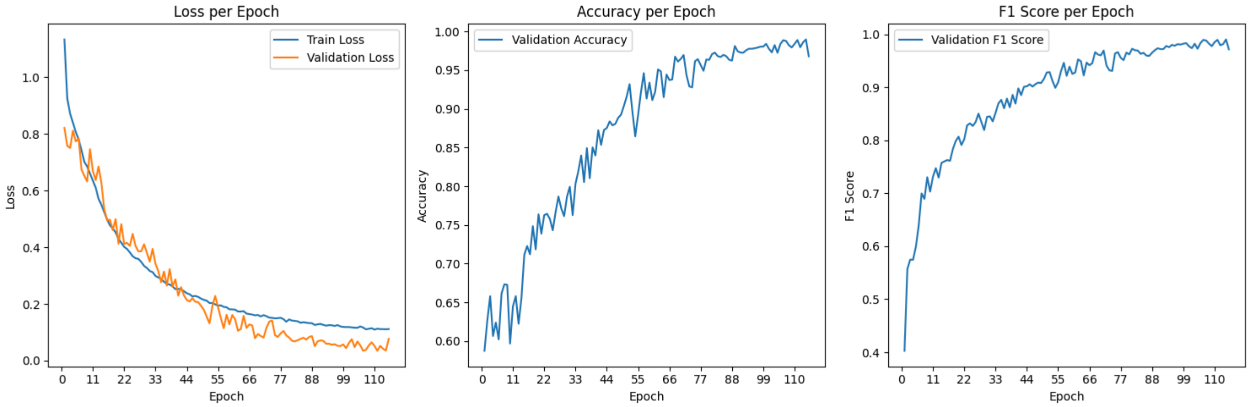
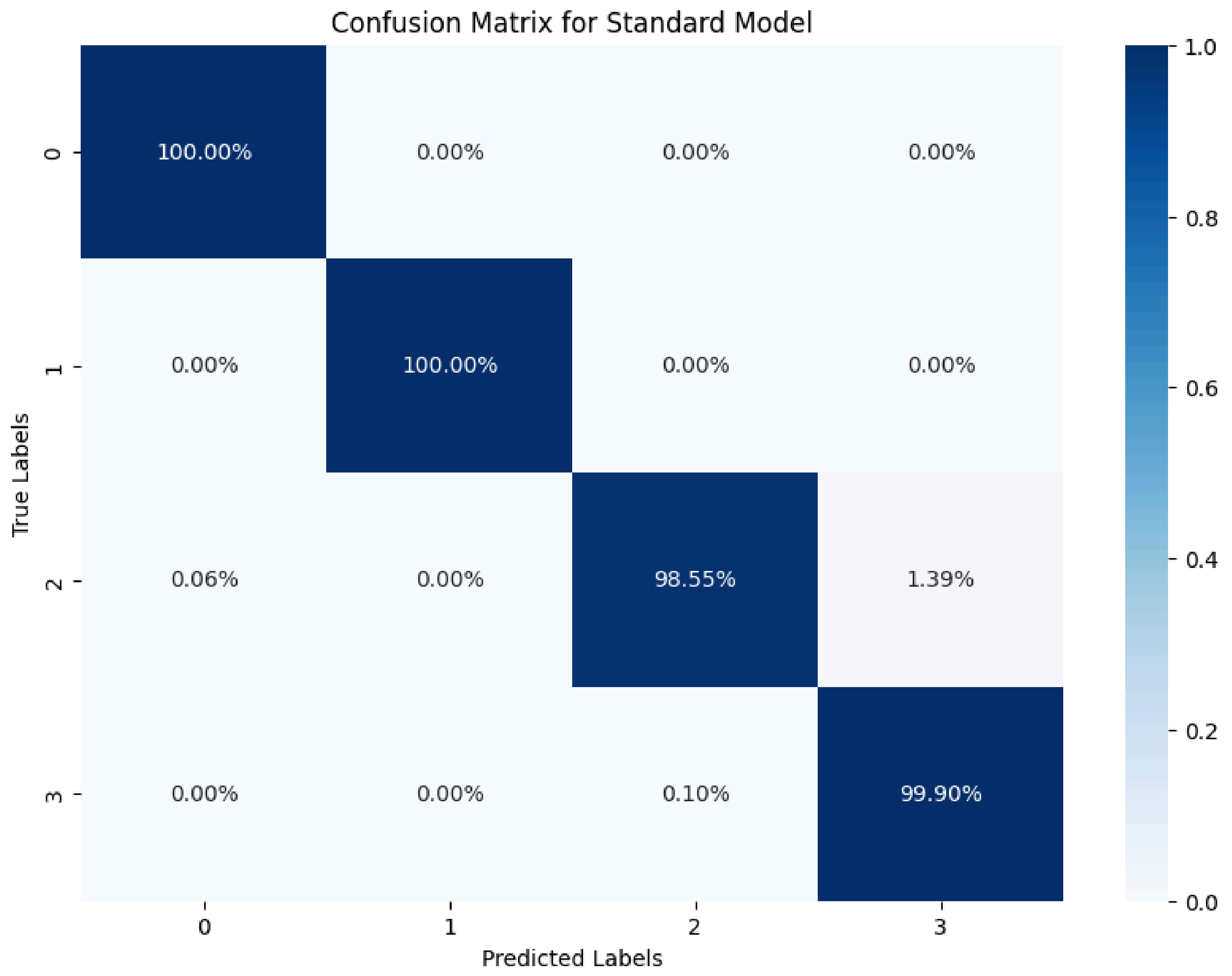


Fig: Train Loss, Accuracy and F1 score after 116 Epoch for BatchNormDropOutCNNModel



Using BatchNormDropOutCNNModel,

Final accuracy obtained	98.86%
Final F1 score obtained	98.88%
Final loss obtained	0.0363

3.2.3 Experiment Findings

CNN2 was unable to converge as there were too few parameters, while SimpleCNN and BatchNormDropOutCNNModel completed training with high accuracy. For these models, we performed adversarial training to improve their robustness and discovered that accuracy decreased as expected.

The model that produced the best performance was the BatchNormDropOutCNNModel, which obtained a final accuracy of 98.86%.

Diving deeper into the analysis of **BatchNormDropOutCNNModel's** performance, we arrive at the following findings:

Generalisation of the BatchNormDropOutCNNModel

Training and validation loss shows a steady decline with every epoch showing that the model is learning well to fit the training data without overfitting.

F1 score and accuracy for both training and validation data improve with every epoch, both following the same trend, suggesting that the model is generalising well.

Achieving a final accuracy of 99.86% on test data is proof that the BatchNormDropOutCNNModel managed to learn essential features in the images needed to classify the different severities of dementia.

Usage of BatchNorm

Before the implementation of the BatchNorm operation, the model could not achieve a test accuracy above 90%.
This led us to implement BatchNorm as we believe it would help to reduce internal covariate shifts in the model and speed up the training process, giving us a more robust model in less time.

BatchNormDropOutCNNModel's performance on each class

Judging by the **confusion matrix and high F1 score**, the pre and post-adversarial variations of the BatchNormDropOutCNNModel achieved high precision and recall, and has managed to **successfully generalise across all classes** as it has a balanced performance across them.

This means that the random weighted sampling and random transformations **successfully allowed the model to learn to classify the 4 classes of dementia severity without developing a bias to the class with the most data** (Non demented).

Significance of our findings in medical application

This is particularly important in a **clinical setting as the implications are significant. For Alzheimer's disease diagnosis and staging**, being able to accurately identify all stages of dementia is crucial for patient care and treatment planning.

This is a concern since there is a much smaller sample size for moderately demented patients so theoretically, we can still achieve a high accuracy despite wrongly classifying many moderately demented patients.

Motivation behind F1 score as evaluation metric in clinical context

In the medical field, the consequences of false negatives can be very severe (e.g., a person with moderate dementia might not receive the care they need if not identified), as can the consequences of false positives (e.g., a person without dementia might undergo unnecessary stress and treatment). Hence, we used F1 score as one of the major performance metrics since it gives a more balanced view of how well the model is performing in terms of these clinically significant outcomes.

3.2.4 Files and Notebooks

Here is the folder containing all information relevant to the CNN model.

- Model + Training + Adversarial training:
  - CNN2, SimpleCNN
  - BatchNormDropOutCNNModel
- Log of CNN training: /CNN/DebuggedModelCheckpoints/SimpleCNN
- Log of BatchNormDropOutCNNModel training: /CNN/DebuggedModelCheckpoints/BatchNormDropOutCNNModel
- Model checkpoints containing saved parameters: /CNN/DebuggedModelCheckPoints

To run training, please download the dementia dataset from Kaggle and open CNN2, SimpleCNN or BatchNormDropOutCNNModel. Training cells are indicated clearly in the notebook with further descriptions and explanations.

3.2.5 Comparison with other research

Compared to a research paper with similar exploration scope titled “Automated Classification of Alzheimer's Disease Based on MRI Image Processing using Convolutional Neural Network (CNN) with AlexNet Architecture”[2], they reported to have achieved an accuracy of 95% while we obtained 99.86%.
We obtained a higher accuracy likely due to the higher quality of dataset used where there was minimal noise present in the images.

3.3 Convolutional Recurrent Neural Network (CRNN)

3.2.1 Model Definition

A Convolutional Recurrent Neural Network (CRNN) is a hybrid neural network architecture that combines the spatial hierarchical feature extraction capabilities of Convolutional Neural Networks (CNNs) with the sequence modelling capabilities of Recurrent Neural Networks (RNNs). This combination allows CRNNs to effectively process data that has both spatial and temporal dimensions, making them particularly well-suited for this task. As the MRI scans present in this dataset are **61 cross-sectional images per patient**, we can use CRNN to learn the sequential features within these 61 images to make a better prediction.

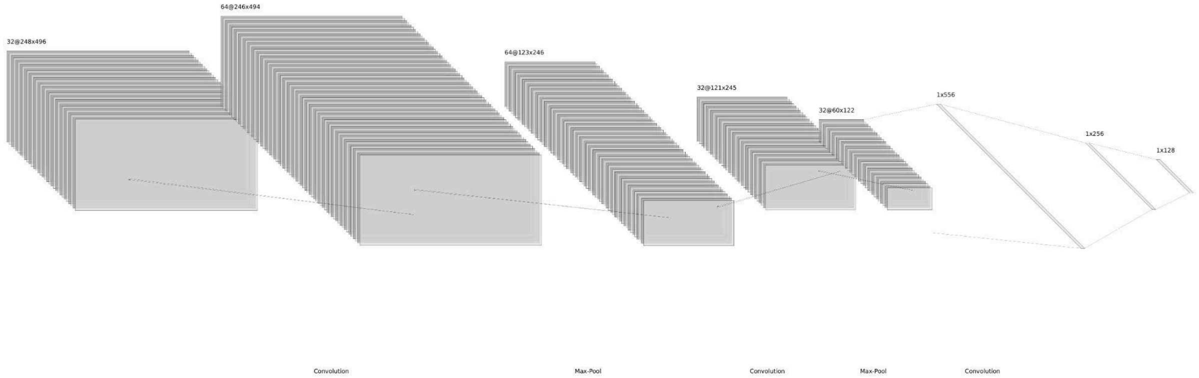


Fig: Visual of CRNN Model (note first linear layer is not accurate as image could not fit 1920 inputs)

```
CRNN(
  (conv1): Conv2d(61, 61, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn1): BatchNorm2d(61, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(61, 61, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn2): BatchNorm2d(61, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv3): Conv2d(61, 61, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn3): BatchNorm2d(61, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=1922, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=4, bias=True)
  (lstm): LSTM(64, 64, batch_first=True)
)
```

3.4.2 Experiment Results

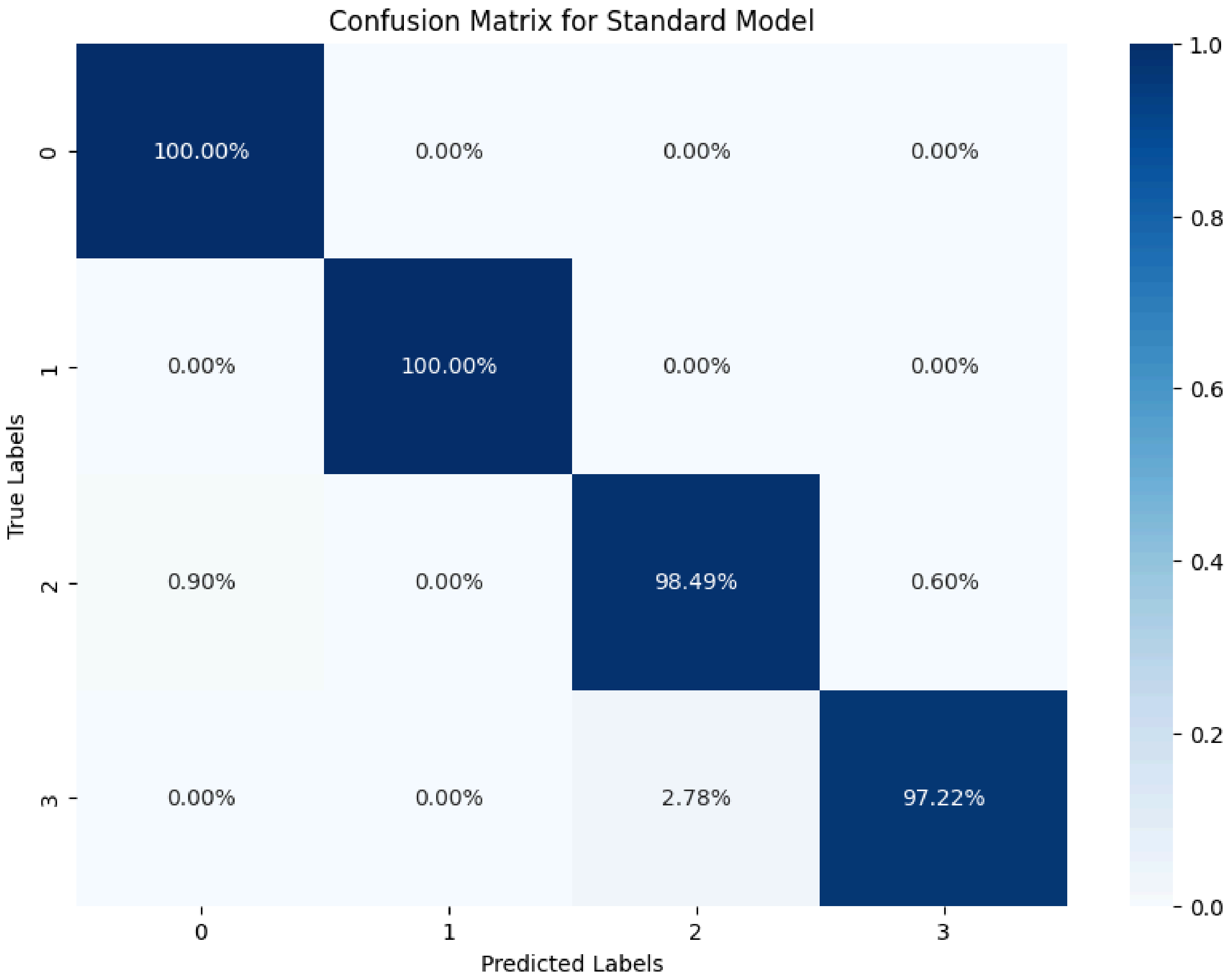


Fig: Valuation + Test Data Confusion Matrix after 20 Epoch

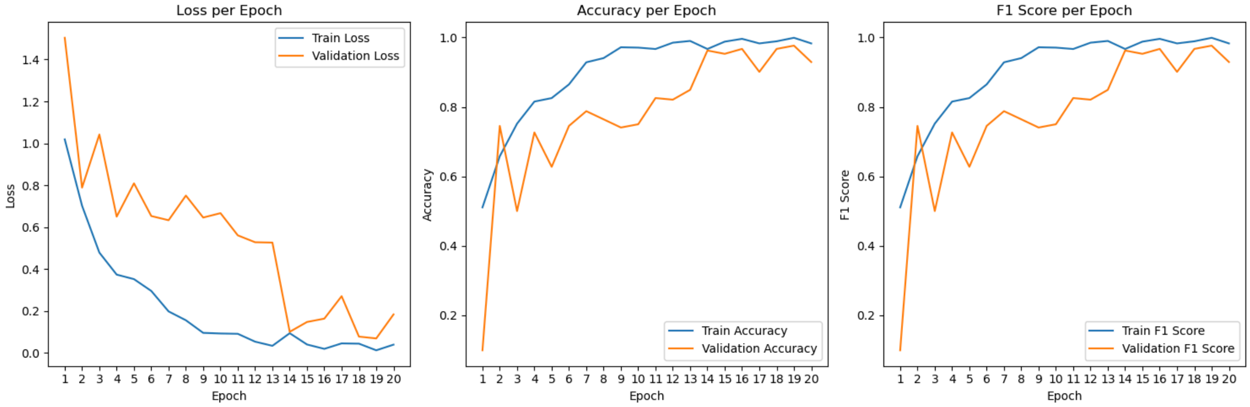


Fig: Final Loss, Accuracy and F1 score after 20 Epoch

Final accuracy obtained	98.36%
Final F1 score obtained	97.37%
Final loss obtained	0.0573

3.3.3 Experiment Findings

Confirmation of sequential images theory

The high performance of the model on classifying patients with varying dementia with significantly fewer model parameters confirms the theory that we can take advantage of the sequential nature of the cross-sectional images from each patient. Since we have the cross-sectional image of each patient from images 160 to 200, it allows us to use an LSTM in the model to capture the intricate spatial-temporal patterns that might be indicative of Alzheimer's disease progression.

Generalisation

The loss graph shows that both training and validation loss decreases over time, with the validation loss closely mirroring the training loss, indicating good generalisation without overfitting.

The accuracy graph shows the model’s accuracy on both the training and validation sets improving over time. Notably, the validation accuracy follows the training accuracy closely but starts to plateau towards the later epochs.

The F1 score for both training and validation increases over epochs, with the validation F1 score showing a bit more fluctuation but generally increasing. The F1 score's performance is especially relevant given the unbalanced dataset, indicating a more balanced classification performance across classes.

Usage of Skip Connection

Before the implementation of skip connections, our CRNN model struggled with training effectively. The depth of the model, combined with the complexity of extracting spatial-temporal features from sequential MRI images, **led to difficulties with vanishing gradients, where the gradient signal becomes too small to make meaningful updates** to the weights in earlier layers. This was evidenced by poor convergence of the loss function and subpar performance on both training and validation sets.

With the introduction of skip connections, these challenges were substantially mitigated. Skip connections promote the flow of gradient during backpropagation by providing an alternative pathway for the gradient to flow through, effectively addressing the vanishing gradient problem. This allowed for deeper layers of the network to learn from the gradient signal that would otherwise diminish in earlier layers.

Performance on each class
CRNN Model - Loss: 0.0573, Accuracy: 0.9836, F1 Macro: 0.9737

Outcome/Class	Mild Dementia	Moderate Dementia	Non Demented	Very mild Dementia
True Positive	100%	100%	98.49%	97.22%
False Negative	0	0	1.51%	2.78%

Significance of our findings in medical application
Judging by the **macro F1 score = 0.9737 and high accuracy in confusion matrix**, the model achieved high precision and recall, and has managed to **successfully generalise across all classes** as it has a balanced performance across them.
This means that the random weighted sampling and random transformations successfully allowed the model to learn to classify the 4 classes of dementia severity without developing a bias to the class with the most data (Non demented).
This is particularly important in a clinical setting as the implications are significant. For Alzheimer's disease diagnosis and staging, being able to accurately identify all stages of dementia is crucial for patient care and treatment planning.

3.3.4 Files and Notebooks

Here is the folder containing all information relevant to the CRNN model.

- Model + Training + Adversarial training: /CRNN/CRNN.ipynb
- Log of model training: /CRNN/log
- Model checkpoints containing saved parameters: /CNN/ModelCheckPoints

To run training, please download the dementia dataset from Kaggle and open /CRNN/CRNN.ipynb. Training cells are indicated clearly in the notebook with further descriptions and explanations.

3.4 Residual Network (ResNet)

Residual Networks (ResNet) is a class of deep neural network architectures that was introduced by Kaiming He et al. in their 2015 paper, "Deep Residual Learning for Image Recognition."[3] ResNets are designed to enable the training of much deeper networks by addressing the vanishing gradient problem through the use of residual blocks. This breakthrough allows ResNet models to achieve outstanding performance across a variety of tasks, including image classification, object detection, and segmentation.

In our project for classification of severity of Alzheimer’s, we recognised that we needed to make changes to ResNet’s architecture to suit this task. Thus, we made **2 key modifications** (highlighted below):

1. Decreased *inchannels* from 3 to 1 as our dataset consists of grayscale images only
2. Increased *numclasses* to 4 as we have 4 distinct classifications for severity of Alzheimer's

We trained ResNet from scratch on the dataset and did not use any pre-trained weights.

3.4.1 Model Definition

```
ResNet(
  (conv1): Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer3): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (downsample): Sequential(
    (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
    (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
(1): Bottleneck(
  (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
)
(2): Bottleneck(
  (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
)
(3): Bottleneck(
  (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
)
(4): Bottleneck(
  (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
)
(5): Bottleneck(
  (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=4, bias=True)
)
```

3.4.2 Experiment Results

Experimenting with various hyperparameter values

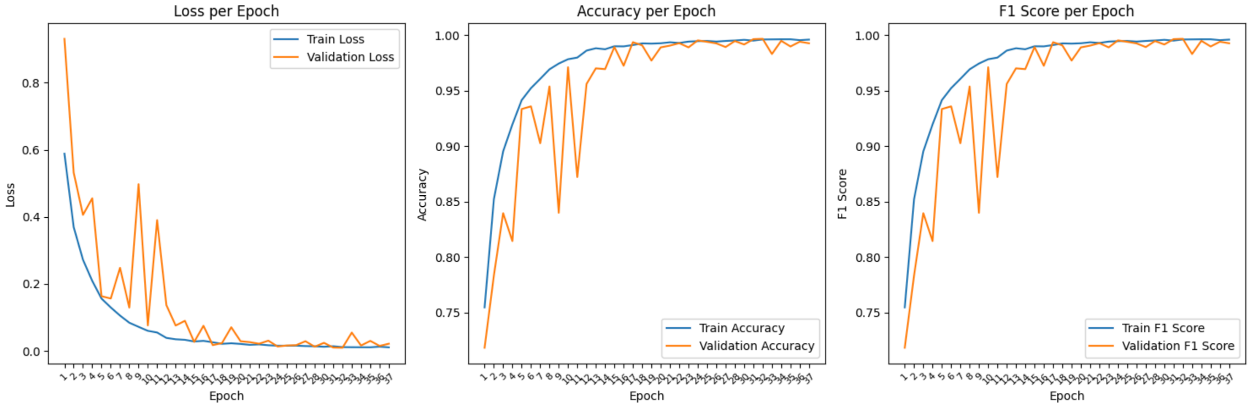


Fig: Final Loss, Accuracy and F1 score after 37 Epochs

Final accuracy obtained	**99.77%**
Final F1 score obtained	**99.69%**
Final loss obtained	**0.0073**

3.4.3 Experiment Findings

State-of-the-art performance

The architecture for ResNet-50 does exceptionally well even for the modified use case of dementia diagnosis, this is unsurprising as Resnet architecture consists of a deep structure that enables the learning of complex feature correlations. This is why we used ResNet as a benchmark, for the earlier mentioned models to meet.

3.4.4 Files and Notebooks

Here is the folder containing all information relevant to the CRNN model.

- Model + Training: /RESNET/ResNet.ipynb
- Log of model training: /ResNet/DebuggedModelCheckpoints/interrupted_performance_24_36.json
- Model checkpoints containing saved parameters: /ResNet/DebuggedModelCheckpoints

To run training, please download the dementia dataset from Kaggle and open /RESNET/ResNet.ipynb. Training cells are indicated clearly in the notebook with further descriptions and explanations.

3.5 Vision Transformer (ViT) - could not train

Apart from CNN, CRNN and ResNet, we explored the Vision Transformer architecture as well.

For context, Vision Transformers (ViTs) are a recent development in computer vision that apply the transformer architecture, originally designed for natural language processing, to image classification tasks. Unlike CNNs, which process images using convolutions and pooling operations, ViTs process images as sequences of tokens.

In our implementation of the Vision Transformer, we ensured that the first Conv2d layer would only take in **1 inchannel instead of 3** as our input images are **grayscale instead of RGB.**

We also applied various transformations to ensure that they fit well with the patches division as illustrated in part (a) in the architecture diagram below.

More information about the architecture can be found below, under 3.5.1 Model Definition.

3.5.1 Model Definition

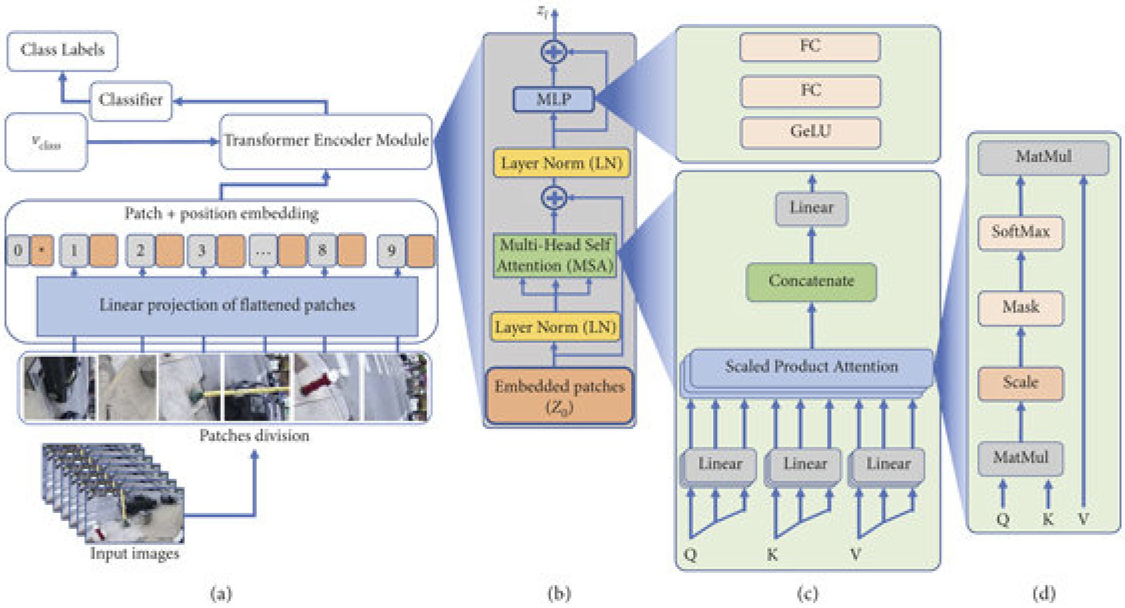


Image credits: ResearchGate

```
ViT(
  (patches): CreatePatches(
    (patch): Conv2d(1, 768, kernel_size=(16, 16), stride=(16, 16))
  )
  (attn_layers): ModuleList(
    (0-11): 12 x AttentionBlock(
      (pre_norm): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attention): MultiheadAttention(
        (out_proj): NonDynamicallyQuantizableLinear(in_features=768, out_features=768, bias=True)
      )
      (norm): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (MLP): Sequential(
        (0): Linear(in_features=768, out_features=3072, bias=True)
        (1): GELU(approximate='none')
        (2): Dropout(p=0.0, inplace=False)
        (3): Linear(in_features=3072, out_features=768, bias=True)
        (4): Dropout(p=0.0, inplace=False)
      )
    )
  )
  (dropout): Dropout(p=0.0, inplace=False)
  (ln): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
  (head): Linear(in_features=768, out_features=4, bias=True)
)
```

3.5.2 Experiment Results

Our implementation with the above architecture was unable to train even with a V100 GPU (provided by the school cluster), due to the vast amount of parameters at 85M. We experimented with decreasing the number of attention blocks down to 6, 3 and 1, but were still unable to train with 7M total parameters.

3.5.3 Experiment Findings

We experimented with different architectures to match the compute power we were provided with, but were ultimately limited by compute and were unable to train the Vision Transformer we have implemented.

3.5.4 Files and Notebooks

Here is the folder containing all information relevant to the CRNN model.

- Model + Training: ViT/vit final.ipynb

To train the Vision Transformer, please download the dementia dataset from Kaggle and open ViT/vit final.ipynb. Training cells are indicated clearly in the notebook with further descriptions and explanations.

4. Conclusion

Summary of model performances

Model Name	Number of	Number of	Test Accuracy	Test F1 score	FN rate (from

	training Samples	Parameters			Mid Moderate Very mild)
ResNet	86,437	23,509,956	99.77%	0.9969	0.13% 0% 0.1%
CNN (BatchNormDropOutCNNModel)	86,437	14,316,356	98.86 %	0.9888	0 % 0 % 0.1 %
CRNN	992	257,628	98.36%	0.9737	0% 0%, 2.78%
ViT	86,437	85,408,516	NIL	NIL	NIL

Accuracy

In medical applications, particularly in diagnostics like MRI-based dementia severity classification, the accuracy and reliability of the models are paramount due to the severe consequences of misdiagnosis. Misdiagnoses can lead to incorrect treatments, delayed proper care, and significant emotional distress for patients and their families.

- **ResNet**: With an accuracy of 99.77%, the **ResNet model** stands out for its exceptional performance in MRI-based dementia severity classification. This model, due to its **deep architecture and residual connections**, might be the most reliable for clinical use
- **CNN**: Our best performing **BatchNormDropOutCNNModel** achieves an accuracy of 98.86%, slightly lower than ResNet but still **well above the general threshold required for medical diagnostics** where radiologists correctly categorised 65-95% of scans and the SVM implemented in this research reached 95% accuracy.[1]
- **CRNN**: The CRNN model, with an accuracy of 98.36%, meets the minimum threshold **generally considered acceptable in medical applications but stands at the lower bound**. It suggests a need for **further validation and refinement**(discussed in future work) before it could be recommended for sensitive medical applications like dementia diagnosis.

FN rate

More critical than the accuracy rate of False-negative diagnosis i.e how often for each class of demented individual does the model misclassify as no demented? In these metrics, the CNN model is the one that performs the best as it did not misclassify any samples of Mild or moderate Dementia class, and only misclassify a couple of samples from the Very mild dementia class. The ResNet model however also makes some False Negative errors for the Mild Dementia class (a more serious stage). The CRNN has a concerningly large amount of FN for the Very Mild

Dementia class. Further hyperparameter tuning must be done to alleviate this issue. (See future consideration)

Number of Parameters and Implications

Models with a large number of parameters take a high amount of c**omputation resources and time during inference.**

In clinical settings, although the inference time is not critical, we might require models to be run on portable or on-device medical diagnostic tools **where hardware capabilities are limited**.

- **ResNet**: Housing over 23 million parameters, ResNet's complexity requires significant computational resources, **which may not be feasible to run on small medical devices.**
- **CNN**: With about 14 million parameters, the CNN model strikes a balance between complexity and computational demand. It can be a viable option for integration into medical devices where some computational capacity is available but is not extensive.
- **CRNN**: As the model with the least number of parameters (around 257,628), the CRNN offers a **potential solution for deployment in environments with stringent hardware limitations**. However, the trade-off in accuracy might be a concern, necessitating a **careful evaluation of where and how it is deployed.**
- **ViT:** As the model with the requirement for the greatest amount of computational resources due to the sheer amount of parameters at 86 million parameters, it is **least feasible** amongst the other models we have explored, despite the high accuracy it might bring, due to the large tradeoff for computational resources.

5. Future consideration

Generic Considerations for All Models:

- Continuous validation against new and emerging data (considering changes in technology used and possibly expanding the models to incorporate inputs from other modalities) to ensure the models' reliability and accuracy are maintained.

Specific Considerations for the CRNN Model:

- The CRNN model's relatively lower accuracy could be improved by addressing the small number of training samples through input sequence length reduction.
- Testing with shorter continuous subsets of images, such as sequences of 5 or 10, could allow for a more extensive training dataset and potentially improve model performance.
- Optimal sequence length determination will require rigorous hyperparameter tuning during the training phase.
- The goal is to find an effective balance between capturing essential temporal features for accurate diagnosis and providing a sufficiently large and varied training dataset.

6. Contribution to Project

Mohammed Ansar Ahmed

- Wrote training and dataset functions
- Developed various visualisation code
- Developed CNN and fine tuned CRNN model

Ng Zheng Wei

- Developed CRNN model
- Visualisation of training data
- Wrote report

Tan Pheng Yuan Ryner

- Developed ResNet, ViT model
- Wrote report

References

[1]	Klöppel, Stefan & Stonnington, Cynthia & Barnes, Josephine & Chen, Frederick & Chu, Carlton & Good, Catriona & Mader, Irina & Mitchell, L & Patel, Ameet & Roberts, Catherine & Fox, Nick & Jack, Clifford & Ashburner, John & Frackowiak, Richard. (2008). Accuracy of dementia diagnosis - A direct comparison between radiologists and a computerized method. Brain : a journal of neurology. 131. 2969-74. 10.1093/brain/awn239.
[2]	Fu'Adah, Y. N., Wijayanto, I., Pratiwi, N. K. C., Taliningsih, F. F., Rizal, S., & Pramudito, M. A. (2021, March). Automated classification of Alzheimer's disease based on MRI image processing using convolutional neural network (CNN) with AlexNet architecture. In Journal of physics: conference series (Vol. 1844, No. 1, p. 012020). IOP Publishing.
[3]	He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).