50.039 Theory & Practice of Deep Learning

Project Final Report

Alzheimer's MRI Detection using Convolutional Neural Networks

Chong Pok Shun (1004992)

Adelle Chan (1005418)

Goh Yu Fan (1005054)

Abstract

Alzheimer's disease (AD) is a progressive neurodegenerative condition leading to memory loss, particularly affecting episodic memory functions such as context recall. It stands as the leading cause of dementia, significantly impacting memory, thinking, language, judgment and behavior. In addition, Alzheimer's disease is irreversible and progressive. Alzheimer's is evaluated by identifying certain symptoms and ruling out other possible causes of dementia, often utilizing medical exams like CT, MRI or PET scans of the head. While there is no cure, medications can help to slow the disease's progression and manage the symptoms.

The project aims to evaluate pre existing computer vision models like CNN, AlexNet, and Inception v3 for Alzheimer's detection. It seeks to determine the most effective model for prediction accuracy.

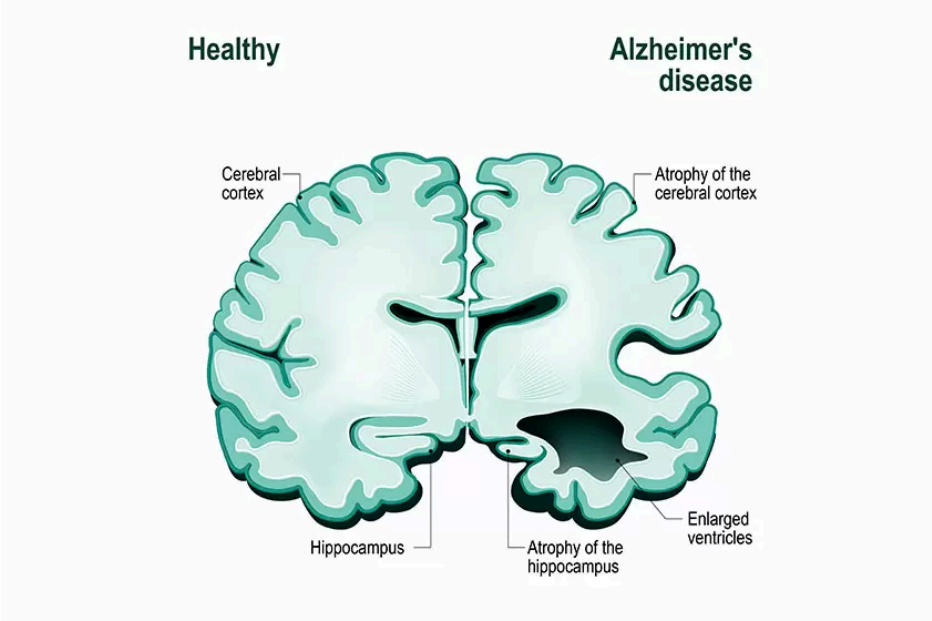

Figure 1. Simplified illustration on Alzheimer’s disease

Methodology

Given the abstract, we modeled our deep learning task as follows, with Task, Input, Output:
Task: Correctly classify the MRI scan to its respective Alzheimer's severity label
Input: MRI scans with Alzheimer's classification
Output: 4 classes of Alzheimer's - non-demented, mild demented, moderate demented,very mild demented

Approach

The approach we aim to take is as follows:

1. Examine effectiveness of Inception (transfer learning)
2. Examine effectiveness of Inception (trained)
3. A Convolutional Neural Network (CNN) will be created from scratched and trained on the Examine effectiveness of pure CNN approach
4. Examine effectiveness of CNN approach + with a WeightedRandomSampler

Dataset

The dataset (obtained from Kaggle[1]) comprises a total of 6400 MRI images of the brain, categorized into four distinct classes. All the images have been resized into a size of 128 x 128 pixels. These classes are as follows:

- Mild Demented: 896 images.
- Moderate Demented: 64 images.
- Non Demented: 3200 images.
- Very Mild Demented: 2240 images.

Our images are saved into separate files and need to be processed with cv2.imread. Which reads raw .jpg files.

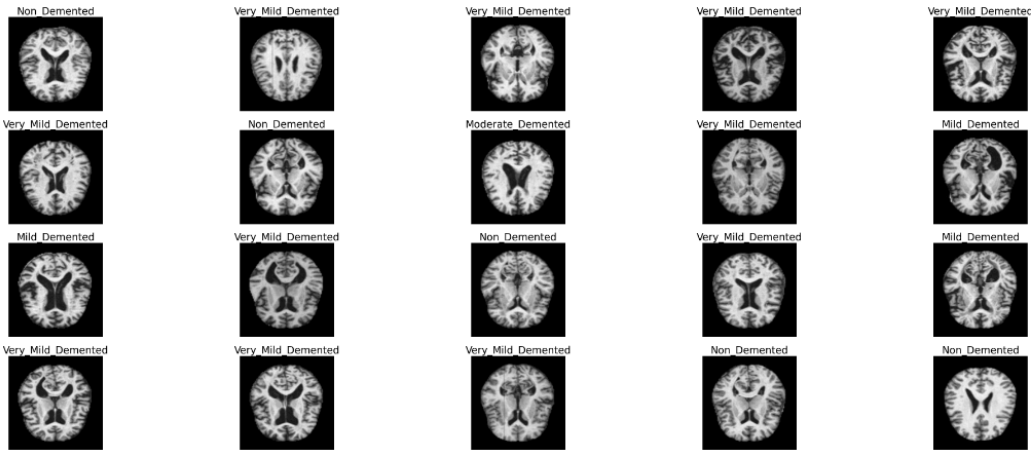

Figure 2: dataset sample.

[1] https://www.kaggle.com/datasets/sachinkumar413/alzheimer-mri-dataset

Dataset Preprocessing

To preprocess the data, the labels were encoded into numerical numbers: Non_Demented: 0, Very_Mild_Demented: 1, Mild_Demented: 2, Moderate_Demented: 3.

It is also observed that the number of samples for the various classes were imbalanced, with the distribution skewed toward Non-Demented and Very-Mild-Demented. In an attempt to balance out the dataset, Weighted Random Sampler was implemented.

Weighted Random Sampler

Class imbalance is a machine learning problem where the number of examples in each class is significantly different. In real-world datasets, it is common for one majority class to vastly outnumber the other minority classes. Handling class imbalance is essential due to poor performance by many machine learning algorithms on training data where one class dominates.

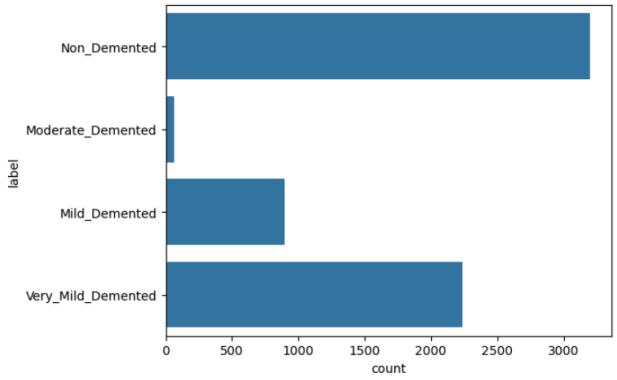

Figure 3: Sampled Data before WRS

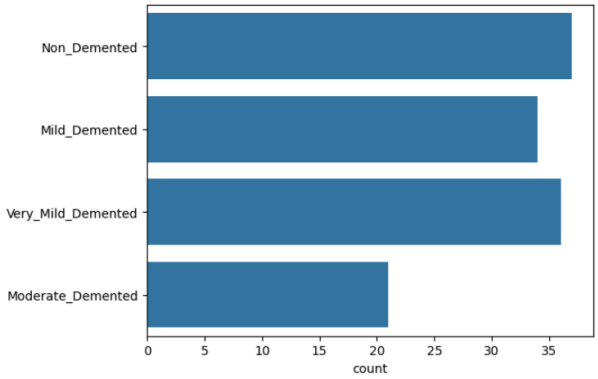

Figure 4: Sampled data after WRS

From the sample data, it is clear our data is biased against the “moderate_demented” minority class. From the study “**A systematic study of the class imbalance problem in convolutional neural networks”**, an oversampling of the minority class is the best way to handle biased data, since it increases the frequency that images from minority classes are seen by the model during training. To eliminate bias from the dataset, a pytorch weighted random sampler is used, with the results as seen in Figure 2 above.

Inception v3

To identify signs of Alzheimer's through the MRI scans, we decided to reference sample papers that have worked on the model. Many research papers revealed use of complicated Convolution Neural Network models, using state of the art solutions for this image processing problem[2].

Inception-v3 is an image recognition model renowned for its accuracy, surpassing 78.1% on the ImageNet dataset. It's the result of collaborative efforts by numerous researchers and is based on the seminal paper by Szegedy et al[3]. The model comprises both symmetric and asymmetric building blocks, incorporating various techniques such as convolutions, pooling, concatenations, dropouts, and fully connected layers. Notable enhancements include the integration of Label Smoothing, Factorized 7 x 7 convolutions, and an auxiliary classifier to propagate label information throughout the network, alongside batch normalization for sidehead layers.

Therefore, we decided to start with Inceptionv3 before fine tuning the solution. For our first Inceptionv3 model, data is preprocessed in datasetv2.py. In datasetv2, data is resized to 299x299 pixels per inception model requirements and normalized to N(mean = 0.1307, var = 0.3081)

```
transform_data = Compose([ToTensor(),
                          Resize(299),
                          Normalize((0.1307,), (0.3081,))])
```

We make no changes to the standard model, using pretrained weights to understand the effect it has on our data. The model is run through a standard feedforward network, with a standard adam optimizer and cross entropy loss.

```
model = models.inception_v3(pretrained=True)
model.aux_logits = False
# standard feedforward network
model.fc = nn.Sequential(nn.Linear(model.fc.in_features, 4))
optimizer = torch.optim.Adam(
    filter(lambda p: p.requires_grad, model.parameters()), lr=1e-3
)
loss = nn.CrossEntropyLoss()
```

[2] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10093003/)

[3] https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Szegedy_Rethinking_the_Inception_CVPR_2016_paper.pdf

Inception v3 evaluation

Overall numbers look decent, with accuracy of the dataset averaging at about 68.2% while the accuracy of label 3, Moderate_dementia, the dataset we have the least data for, reaches 91.6%. However, the loss and accuracy data from the model's training data and test data reveals severe overfitting in the model.

Firstly, the training loss data does not decrease smoothly, sporadically increasing. (blue line)

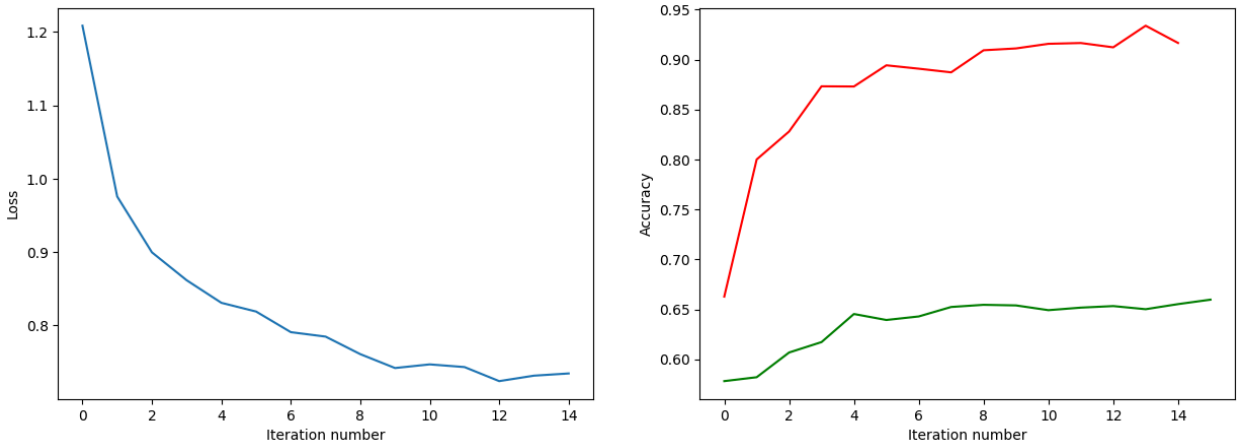

Figure 5: Training Loss and Accuracy curves for Inception v3 (No Batchnorm)

The green line is total accuracy. The red line is accuracy for the moderate_demented, our smallest dataset.

Next, we see that test data accuracy is much lower than the original dataset. Despite high label 3 accuracy, there is a relatively high loss and a much lower total average accuracy.

Loss 0.781, Accuracy: 0.589062511920929, Lbl3 Acc: 0.9287733182589033

	precision	recall	f1-score	support
Non_Demented	0.79	0.70	0.74	33
Very_Mild_Demented	0.38	0.48	0.43	27
Mild_Demented	0.59	0.87	0.70	31
Moderate_Demented	1.00	0.51	0.68	37
accuracy			0.64	128
macro avg	0.69	0.64	0.64	128
weighted avg	0.72	0.64	0.65	128

Inception v3 with Batch Normalization

In our attempt to reduce overfitting, we proceeded to further generalize the model with batch normalization.

On top of the standard feedforward network used earlier, a new batchnormalization layer is added after each Conv2d layer.

```
model = models.inception_v3(pretrained=True)
model.aux_logits = False

for name, module in model.named_children():
    if isinstance(module, nn.Conv2d):
        # Insert BatchNorm layer after each Conv2d layer
        setattr(model, name, nn.Sequential(module,
nn.BatchNorm2d(module.out_channels)))

model.fc = nn.Sequential(nn.Linear(model.fc.in_features, 4))
```

Like above, we use a standard adam optimizer and cross entropy loss.

Inception v3 with Batch Normalization evaluation

Our attempts to generalize the model worked: the loss more smoothly decreases over time. Average overall accuracy barely changed from 68.2% to 68.1% and moderate_dementia accuracy increased from 91.6% to93.1%. However, we still see 2 challenges. Firstly, the loss is still sporadic and still shows signs of overfitting. (blue line)

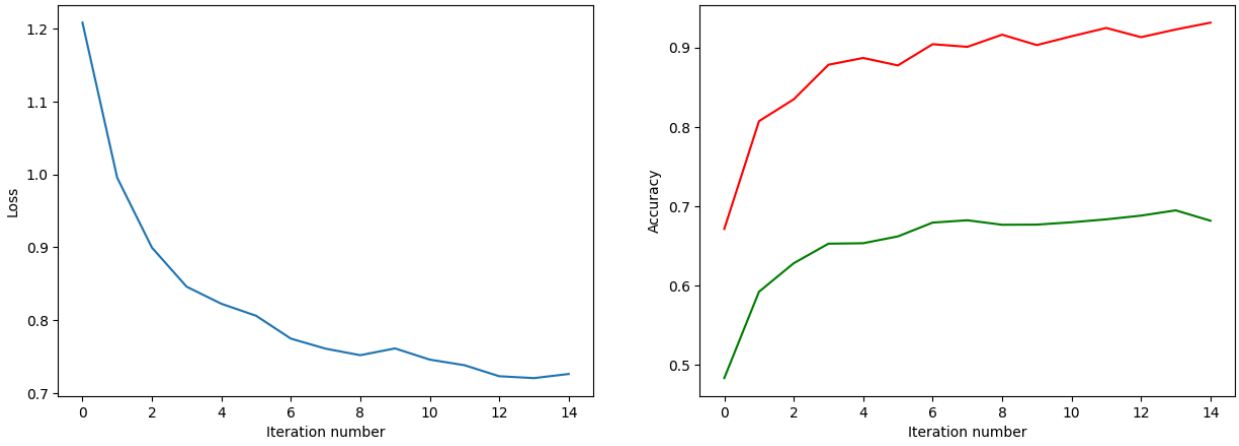

Figure 6: Training Loss and Accuracy curves for Inceptionv3 (With Batchloss)

The green line is total accuracy. The red line is accuracy for the moderate_demented, our smallest dataset.

In addition, our test set appears to have a worse accuracy and loss than the previous model. We visibly see an increase in loss and a decrease in accuracy from 59% to 56%.

Loss 0.8181416015326978, Accuracy: 0.5570312738418579, Lbl3 Acc: 0.9316730523627076

	precision	recall	f1-score	support
Non_Demented	0.68	0.44	0.54	34
Very_Mild_Demented	0.46	0.35	0.40	34
Mild_Demented	0.41	0.82	0.55	28
Moderate_Demented	0.96	0.72	0.82	32
accuracy			0.57	128
macro avg	0.63	0.58	0.58	128
weighted avg	0.63	0.57	0.57	128

It was clear that the model was generalizing even less now, and that this was not the way to go. Hence, we decided instead to reduce the complexity of our model, reducing the number of layers to 2. This is done with the assumption that the model was over complicated and hence overfitting.

2-Layer CNN

For our initial CNN model, data was preprocessed in datasetv3 with function to be resized into 28x28 pixels and normalized into N(mean = 0.1307, var = 0.3081)

```
transform_data = Compose([ToTensor(),
                          Resize(28),
                          Normalize((0.1307,), (0.3081,))])
```

The model had a standard 2 convolutional layers with size 32 and 64 respectively of kernel_size=3 and stride=1, with 2 fully connected layers and 3 iterations.

```
# two convolutional layers
self.conv1 = nn.Conv2d(3,32, kernel_size=3, stride=1, padding=1 )
self.conv2 = nn.Conv2d(32,64, kernel_size=3, stride=1, padding=1)
# two fully connected layers
self.fc1 = nn.Linear(28*28*64, 128)
self.fc2 = nn.Linear(128, 4)
```

The model uses a standard adam optimizer. A ReLU activation function is also used to introduce non-linearity to the model.

```
optimizer = optim.Adam(model.parameters(), lr = lr)
X = F.relu(x)
```

The dimensions of the feature maps are preserved throughout the convolutional layers due to the use of padding. However, the size of the feature maps is reduced when passing through the fully connected layers, leading to a flattened shape before entering the fully connected layers.

2-Layer CNN Evaluation

It appears our choice to reduce the model complexity worked, and we were able to attain a model that could fit our data. Surprisingly, this worked well too for the minority dataset.

--- Epoch 13/13: Train loss: 0.0629, Train accuracy: 0.9801

Above we see that at just 13 epochs, the training loss had reached 0.0629 and training accuracy was 0.9801. Final accuracy of Label 3 is also 1.0.

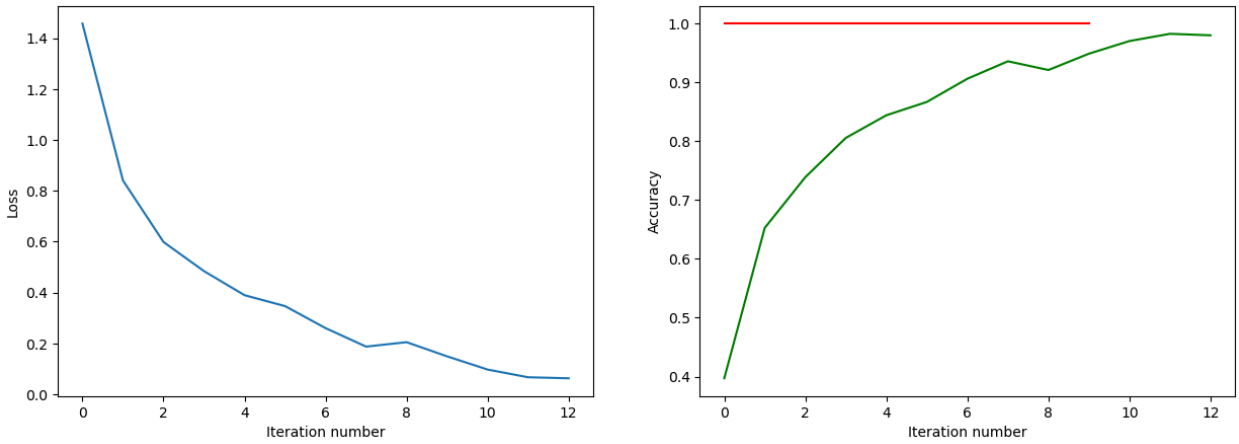

Figure 7: Training Loss and Accuracy Curves for 2-Layer CNN

The test set was also able to do well. Below we have the classification report. Our minority class has an F1 score of 1.0, and overall accuracy of 0.98.

	precision	recall	f1-score	support
Non_Demented	1.00	0.94	0.97	33
Very_Mild_Demented	0.90	1.00	0.95	27
Mild_Demented	1.00	0.97	0.98	29
Moderate_Demented	1.00	1.00	1.00	39
accuracy			0.98	128
macro avg	0.97	0.98	0.97	128
weighted avg	0.98	0.98	0.98	128

Evaluation & Analysis

For the base CNN, we obtained a result of test loss: 0.5617 and test accuracy: 0.7500. The next step was to bring the accuracy of the model up and try to lower the test loss. We went ahead to increase the number of epochs for training, from 3 to 10. With that we achieved Train loss: 0.1465, Train accuracy: 0.9506, Test loss: 0.2329, Test accuracy: 0.9117

The experimentation for this portion can be seen in **DL_proj.ipynb**, with results in saved_models as

1. Az_CNN (1e-3)
2. Az_CNN_5e_4 (5e-4)
3. Az_CNN_1e_4 (1e-4)

After reaching the final iteration, by no free lunch theorem, we decided to try different learning rates in hopes that another learning rate will improve the model.

Given 10 epochs

Learning rate	Train loss	Train accuracy	Test loss	Test accuracy
1e-3	0.1465	0.9506	0.2329	0.9117
5e-4	0.0950	0.9783	0.1860	0.9398
1e-4	0.4647	0.8100	0.5148	0.7945

Upon trying it with our final model, we noticed that the test accuracy decreased to 88% instead, a whole 10% decrease for the same settings. Despite our best efforts, it appears that Adam Optimizer's default settings will always win. Hence our final model uses a learning rate of 1e-3.

Conclusion

To conclude, the 2 layer CNN model turned out to be the most proficient at working the dataset. Additionally, even with a resizing of 28x28 from a 224x224 model, with 13 epochs the model was performing better than many of the models in current research papers. We have learnt that simple is better and that often, a good model has few layers with good optimization.

Despite Inceptionv3 being state of the art, a model of that complexity was simply not required for our use case: a black and white 2D image with 4 classes. A simple model is sufficient and useful enough for the prediction, coupled with balancing of the dataset in the preprocessing of data.

Appendix

Github Repository: https://github.com/benjaminchong99/deepLearning_AlzheimerDetection
Head to the finalsubmission branch.

Contributions

Adelle: Weighted random sampler, Standard 2 layer CNN, report
Yu Fan: Inceptionv3 model inclusion, report
Pok Shun: Inclusion of batch normalization, experimentation with learning rates, report

Instructions to Recreate

1. Ensure ad_labels.py, datasetv2.py and archive.zip are all in your environment.
2. Unzip archive.zip
3. Run inceptionv3_smote_3.ipynb for a full breakdown of inceptionv3 models and standard 2-layer CNN
4. Run DL_proj for learning rate experiments.

References

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3312396/
https://www.radiologyinfo.org/en/info/alzheimers
https://www.kaggle.com/datasets/sachinkumar413/alzheimer-mri-dataset