Classification of Alzheimer's Disease with Machine Learning

Machine Learning
Professor Sergio Alvarez
Riteng Zhang, Qiyuan Zhou, Anirudh Iyer, Ryan Bennett, Roman Kozdrowicz
December 2022

1. Abstract

Alzheimer's disease is an incurable type of dementia that typically affects those over the age of sixty. Given that there is no cure, early detection is paramount in starting treatments early to try to curb the disease's effect. In this paper, we address the image classification problem focused on MRI scans to determine Alzheimer's disease progression using core machine learning techniques. We experiment with a variety of learning algorithms, including Naive Bayes, Logistic Regression, K-Nearest Neighbors (KNN), and transfer learning with Inception, NASNetMobile, and ResNet. We also include results from our own CNN based model, tweaking parameters to improve performance. The results from the traditional machine learning algorithms underperformed, reaching a highest accuracy of 75% from KNN. We found that the complex transfer learning models overfit, with training accuracies at 90+% and validation scores between 60-89%. These "fancy" transfer learning models were also more challenging to train properly, as many of them took up to 12 hours to train. Ultimately, our own CNN-based model resulted in the best accuracy of 97%, including classifying non-demented MRI scans with an accuracy of 100%. These results embody the importance of machine learning in healthcare and the effectiveness of deep learning for complex tasks.

2. Introduction

Alzheimer's disease is a progressive disease resulting from brain cell degeneration, impacting fifty million people globally and 12% of Americans over sixty. The main symptoms of Alzheimer's disease include memory issues, anosmia, and impaired reasoning. There is no cure for the disease and the only treatment is cognition enhancing medications. Unfortunately, the symptoms of Alzheimer's disease overlap with other neurodivergent disorders, often leading to delayed or misdiagnosed conditions, which severely reduces the effectiveness of treatment for patients.

As a result, the early detection and classification of Alzheimer's disease is a major challenge faced by the medical industry and there are extensive efforts to develop better models to achieve these goals. One paper by Sheng et al. published in *Nature* in 2022, aimed to use brain scan imagery and machine learning methods to classify the scans between Alzheimer's afflicted brains, healthy controls, and multiple stages of mild cognitive impairment. In it, they used a feature selection program to identify certain imaging based and genetic features that yielded classification accuracies between 72-98% for different classification sub-tasks. Another paper, this one published in the journal *Neural Computing and Applications* investigated the application of a convolutional neural network (CNN) framework to classify the disease from brain MRI scans. In this paper, AbdulAzeem et al. identify how deep learning techniques offer an advantage over conventional machine learning methods like KNN and decision trees. By removing the need for pre-processing and manual feature selection, their attempts to develop a CNN framework achieved 99.6%, 99.8%, and 97.8% classification accuracies.

With this research in mind, this project will attempt to build a machine learning model, leveraging deep neural networks, to classify MRI image scans of brains on a comprehensive spectrum: 1) non demented, 2) very mildly demented 3) mildly demented, or 4) moderately demented.

3. Methods

We used a dataset from Kaggle titled *Alzheimer MRI 4 Classify Fasiai*. This dataset contains both the original dataset and the augmented dataset. In this project, we are only using the augmented dataset because the categories are relatively balanced. Overall, this dataset contains 33984 augmented MRI images in 4 categories: nondemented (8960 images), very mildly demented (8960 images), mildly demented (8960 images), and moderately demented (6464 images). Our methodology for this project is divided into mainly two groups: non deep learning methods and deep learning methods. Non deep learning methods include Naive Bayes, KNN, and Logistic regression; deep learning models include transfer learning with Inception, NASNetMobile, ResNet, and our CNN Based Model. However, the preprocessing steps for non deep learning methods are very naive and their results only act as a baseline to compare with.

3.1. Non Deep Learning Methods

For data preprocessing of non deep learning models, *cv2* and *imutils* API along with two helper functions were used. The original images are over 180x180 pixels, which is unnecessarily large and caused the models to take a long time to run. The first helper function shrinks the input image into 32x32 pixels and flattens the RGB pixel intensities into a single list of numbers. Given that there are three channels for each red, green, and blue component, the output feature vector of this function will be a list of 32x32x3 = 3072 numbers. The second helper function is more like an abstract of the first helper function. It constructs a color histogram for each input image to characterize the color distribution of the image in terms of hue, saturation, and value. Since we set 8 bins for each of the hue, saturation, and value channels, the final feature vector will be the size of 8x8x8 = 512. Using these two helper functions, we are able to create three matrices: raw pixel intensities matrix, features matrix, and a labels list. Since there are 33984 raw images in total, the shapes are (33984, 3072), (33984, 512), and (33984,) respectively. The train/test split we used for all models is 90% to 10% with a random state of 42.

3.1.1. Naive Bayes

Naive Bayes is a simple and straightforward probabilistic classifier, and there is nothing to tune. The accuracy for two different kinds of inputs are shown below. The advantage of this model is that it is not time consuming to run. However, since Naive Bayes assumes that all features are independent, which rarely happens in real life, we are not taking this result too seriously. These results are aimed to provide a comparison to the models we tried later on.

```
[INFO] raw pixel accuracy for Naive Bayes: 34.21%

[INFO] histogram representation accuracy for Naive Bayes: 28.68%
```

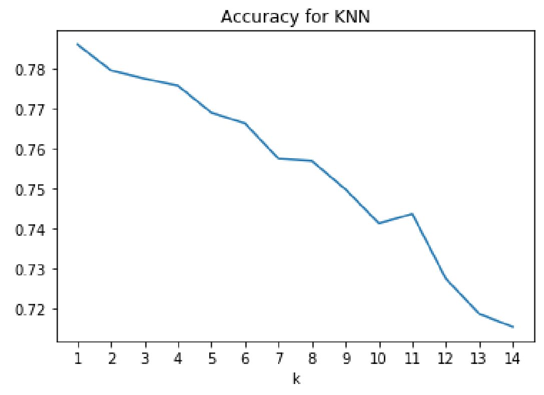
3.1.2. Logistic regression

We used grid search on the following parameters: solvers = ['lbfgs', 'sag','saga'], penalty = ['l1', 'l2', 'elasticnet'], c_values = [10, 1.0, 0.1]. The accuracy for the two different kinds of inputs are shown below.

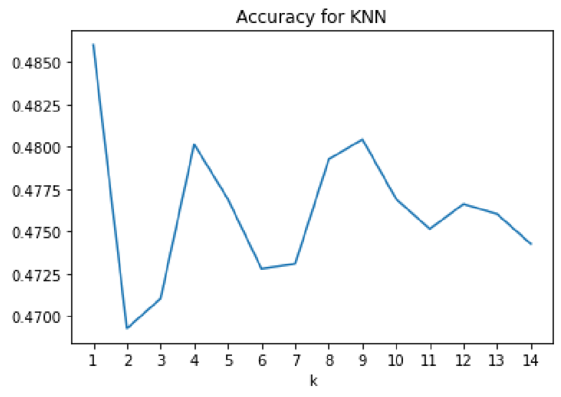
```
[INFO] evaluating raw pixel accuracy for Log reg...
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'penalty': 'l2', 'solver': 'sag'}
accuracy : 0.5587013774581998

[INFO] evaluating histogram representation accuracy for Log reg...
tuned hpyerparameters :(best parameters)  {'C': 10, 'penalty': 'l1', 'solver': 'saga'}
accuracy : 0.40411966650318776
```

3.1.3. KNN

Since k is the only hyperparameter we can tune in KNN, we drew an accuracy plot with respect to the value of k. The k values range from 1 to 15. The plot on the left is for raw pixels as input, and the plot on the right is for histogram representation as input.



accuracy for raw pixels



accuracy for histogram representation

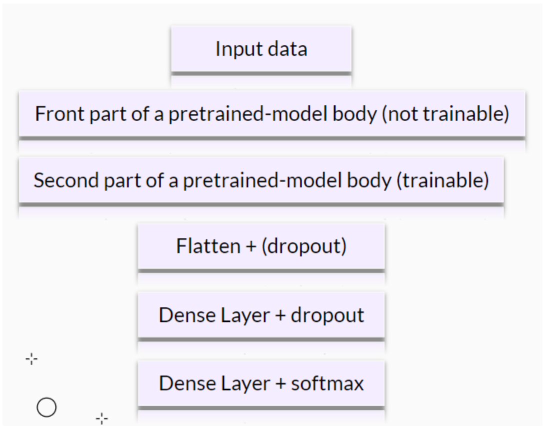
Only the euclidean distance metric was used here, since a covariance matrix is needed for Mahalanobis distance metric, which was far too large to calculate for our dataset. KNN is easy to implement and there is no training step; however, KNN struggles to predict the output as the numbers of variables grow.

3.2 Deep Learning Methods

For all of the deep learning methods that we used, we first converted all of the images into tensors of shape (224, 224, 3) with padding (if any image is not this large). We save 10% of the data as our validation dataset, use Adam as optimizer (so we don't write any scheduler function), and use categorical cross-entropy as the loss function. Initial learning rate is always set to 0.001 in Adam. This initial learning rate works well for all cases so we never decided to change it. We evaluate our results by comparing accuracy between the different models.

3.2.1 Transfer learning

We first tried to do transfer learning with three classic pre-trained models, including Inception, ResNet, and NASNet. The figure below shows the overall structure of our transfer learning model.


Input data
Front part of a pretrained-model body (not trainable)
Second part of a pretrained-model body (trainable)
Flatten + (dropout)
Dense Layer + dropout
Dense Layer + softmax

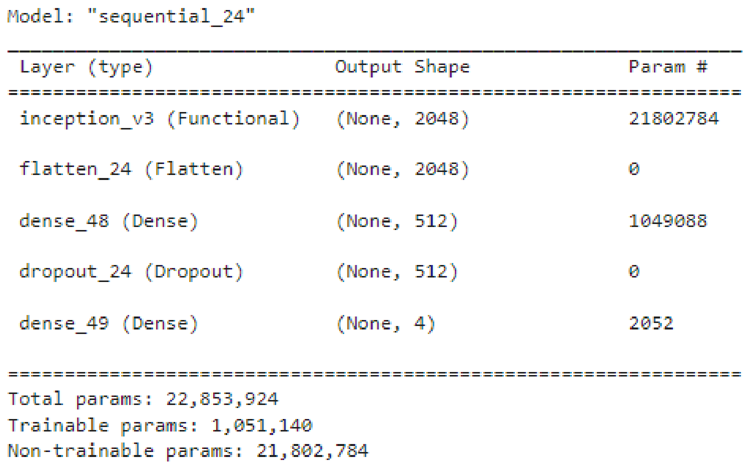
The model body, or the pretrained model is split into two parts: a trainable front and a non trainable back. For example, take a pre-trained model with n layers. We can split up this model into two parts from any layer range from layer 0 to layer n. If we split it up from layer 0, then the entire pre-trained model is trainable; if we split it up from layer n, then the entire pre-trained model is not trainable; if we split it up from layer x, where x is not 0 or n, then we train part of the pre-trained model. Where to split can't be decided before we run any model, so we have to start by training the whole pre-trained model, see what happens, and then adjust.

The output of the pre-trained model is then converted into a 1d array, and this array becomes the input of our classifier, or the model head. The classifier consists of several denses layers with activation function of reLu. For the final layer, its output has the shape of (1, 4), so the final prediction will be the index of the largest values in these 4 numbers. Thus, we add a softmax layer to make the choice of our final dense layer with output of 4 more obvious.

Note: all the non mentioned detailed parameters are not decided before running any model, so they will vary a lot during our running and will be mentioned in later sections.

3.2.1.1 Inception

The Inception model was created to solve the problem of variation in the location of the important information; choosing the right kernel size for the convolution operation can be very difficult (Szegedy et al., 2015). Instead of simply trying to stack layers with different kernel sizes, Inception uses kernels of different sizes and concatenates them together; thus the problem is solved at a very low computational cost (Szegedy et al., 2015).



```
Model: "sequential_24"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_v3 (Functional)   (None, 2048)              21802784

 flatten_24 (Flatten)        (None, 2048)              0

 dense_48 (Dense)            (None, 512)               1049088

 dropout_24 (Dropout)        (None, 512)               0

 dense_49 (Dense)            (None, 4)                 2052

=================================================================
Total params: 22,853,924
Trainable params: 1,051,140
Non-trainable params: 21,802,784
```

The figure above shows our best-version model using Inception as encoder, the other variants of this are only different in few details.

We first tried to train the entire pre-trained model body together with the classifier model head, and we assumed it would overfit. Indeed it is overfitting since it's too large and complicated for our task. Then, we tried to freeze the front part of Inception up to 100, 247 or 311. Note that layer 311 is the final layer in Inception. We chose these numbers because they are usually the last layer in some stack so it makes sense to freeze several numbers of stacks and unfreeze some other stacks instead of cutting one stack into two parts which makes no sense.

```
 100 <keras.layers.merging.concatenate.Concatenate object
248 <keras.layers.merging.concatenate.Concatenate object
```

```
311 <keras.layers.pooling.global_average_pooling2d.GlobalAveragePooling2D object
```

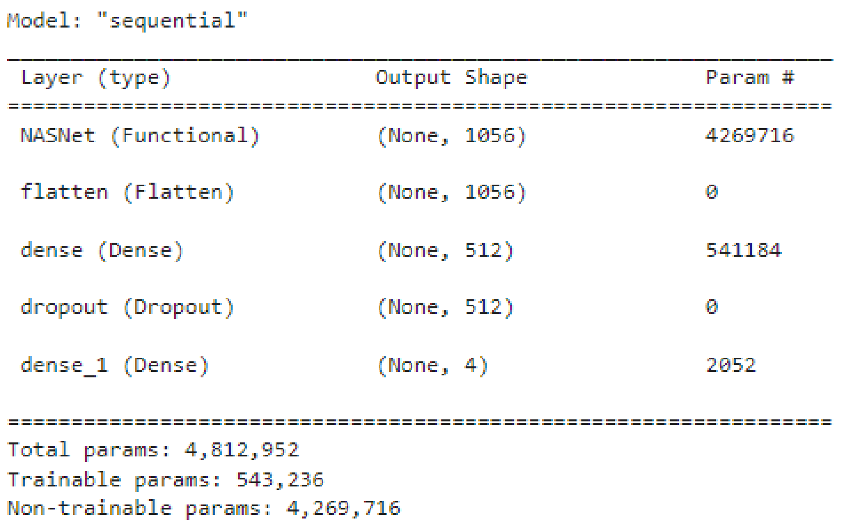
The reason why we chose these numbers in increasing order is that it was apparent that the previous result model will end up overfitting. Even the last model, in which we froze the entire Inception, experienced overfitting. Thus, we could only let Inception be a non-trainable encoder for our task.

This method gives us much better results, but to further overcome the overfitting problem we adjust dropout rate and also add a dropout layer after the flatten layer, which indeed gives us better results (86% validation accuracy). There is still overfitting but it's difficult to keep regularizing it because these mentioned regularizations already made it very difficult to fit the training data.

This covers all the variant models that we've tried using pre-trained Inception. The numerical result will be shown in more detail in Section 4.

3.2.1.2 NASNetMobile

NASNetMobile is another deep learning architecture developed by Google Brain, and the NAS stands for "Neural Architecture Search". The special thing about this model is that the blocks are chosen by reinforcement learning and the structure within each block is chosen by RNN (Zoph et al., 2018). Thus the architecture of this model is the result of machine learning already.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 NASNet (Functional)         (None, 1056)              4269716   
                                                                 
 flatten (Flatten)           (None, 1056)              0         
                                                                 
 dense (Dense)               (None, 512)               541184    
                                                                 
 dropout (Dropout)           (None, 512)               0         
                                                                 
 dense_1 (Dense)             (None, 4)                 2052      
                                                                 
=================================================================
Total params: 4,812,952
Trainable params: 543,236
Non-trainable params: 4,269,716
```

The figure above shows our best-version model using NASNet as encoder, the other variants of this are only different in few details.

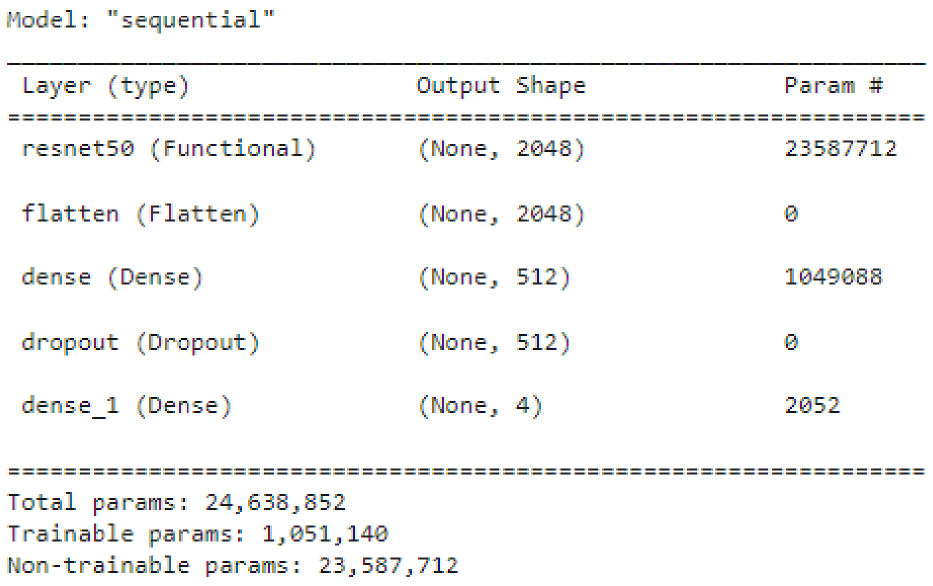
Since the last large model, Inception, overfits no matter what regularization we have, we started trying transfer-learning with NASNet by freezing the entire model (making it non-trainable). Then we tried to use several different dropout rates ranging from 0.05 to 0.2. Even with the highest dropout rate it is still overfitting (This gives the best result though for the NASNet section, with 89% validation accuracy).

We added a dropout layer after the flatten layer, then tried a different dropout rate from 0.05 to 0.2 again. It takes much longer to train but the result is not as good as the models without dropout layer after flatten layer. Again, it's difficult to keep regularizing it because these mentioned regularizations already made it very difficult to fit the training data.

This covers all the variant models that we've tried using pre-trained NesNet. The numerical result will be shown in more detail in Section 4.

3.2.1.3 ResNet

Residual Network (ResNet) is a deep learning network architecture developed by researchers at Microsoft. One crucial feature of the model is the use of skip connections which allow information to bypass one or more layers, increasing learning efficiency and reducing the risks of vanishing gradients (He et al., 2016). Additionally, another unique aspect of the model is normalizing the activations of each layer in the network using batch normalization (He et al., 2016).

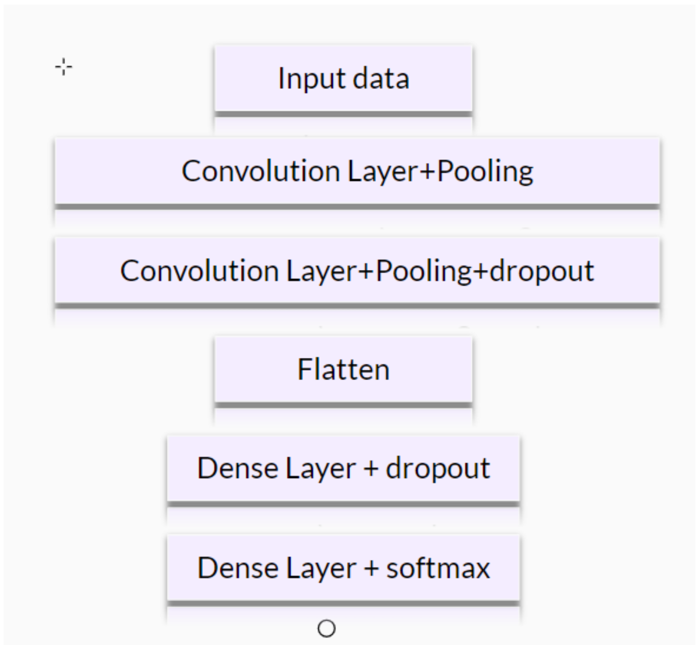


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 2048)              23587712

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 512)               1049088

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 4)                 2052

=================================================================
Total params: 24,638,852
Trainable params: 1,051,140
Non-trainable params: 23,587,712
```

The model did not fit our MRI images well when we made ResNet non-trainable. Although we attempted to open certain layers to train together, this approach led to substantial overfitting. Our optimal result led to a training accuracy of 0.9 and validation accuracy of only 0.62 (when opening layers after layer 143), which pales in comparison to the other learning models.

This covers all of the variant models that we've tried using pre-trained ResNet. The numerical result will be shown in more detail in Section 4.

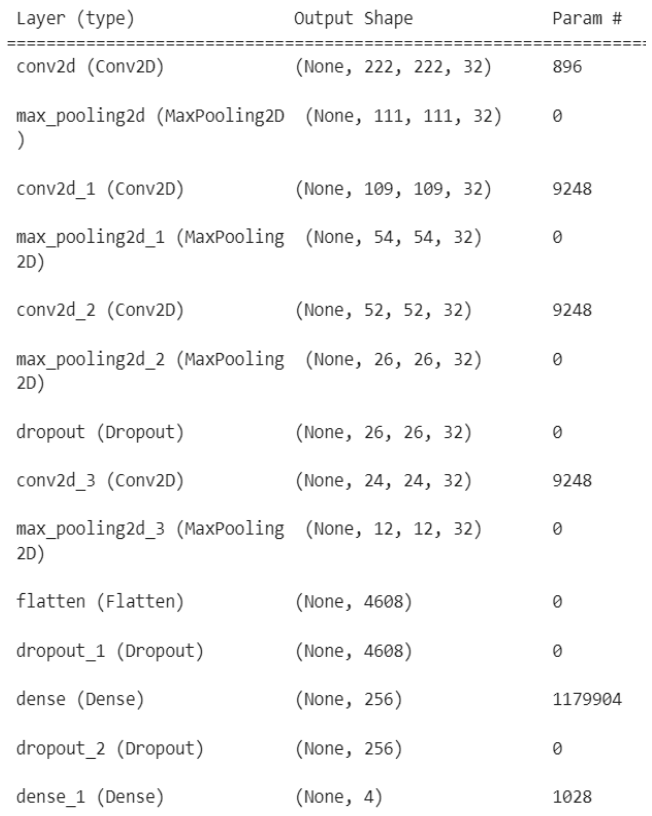
3.2.1.3 Custom CNN Based Model



Since most of our models using pre-trained models are overfitting, we decided to make a simple custom approach. Thus, we wrote our own version of a CNN based model. The above figure is the architecture of the variants of this model. It proceeds with blocks of a convolution layer, max pooling layer without dropping out anything, followed by blocks of convolution layer, and a max pooling layer with dropping out layer. Then we flatten the output from the last dropped out pooling layer, followed by several dense layers with dropping out. Finally, the dense layer with softmax gives us the final result.

First, we tried this architecture without dropping out the output of any layer before the flatten layer, which gave us a somewhat overfitting result. However, the result is much better than the result we got from previous transfer learnings, using a much fewer number of iterations.

We added a dropout layer after certain convolutional stacks (after some of the pooling layers, to be more specific). One of these variants gave us the best result in our project, and the details are shown below.



```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2D  (None, 111, 111, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 109, 109, 32)      9248

 max_pooling2d_1 (MaxPooling  (None, 54, 54, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 52, 52, 32)        9248

 max_pooling2d_2 (MaxPooling  (None, 26, 26, 32)       0
 2D)

 dropout (Dropout)           (None, 26, 26, 32)        0

 conv2d_3 (Conv2D)           (None, 24, 24, 32)        9248

 max_pooling2d_3 (MaxPooling  (None, 12, 12, 32)       0
 2D)

 flatten (Flatten)           (None, 4608)              0

 dropout_1 (Dropout)         (None, 4608)              0

 dense (Dense)               (None, 256)               1179904

 dropout_2 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 4)                 1028
```

Pooling layers are chosen to be max_pooling, and dropout rate is fixed to 0.2. We also tried a 0.3 dropout rate to see if it can further regularize the model, but there's no improvement and it takes much longer to train.

More detailed results will be discussed in the next section.

4. Results and Discussion

4.1 Non-deep learning baseline results

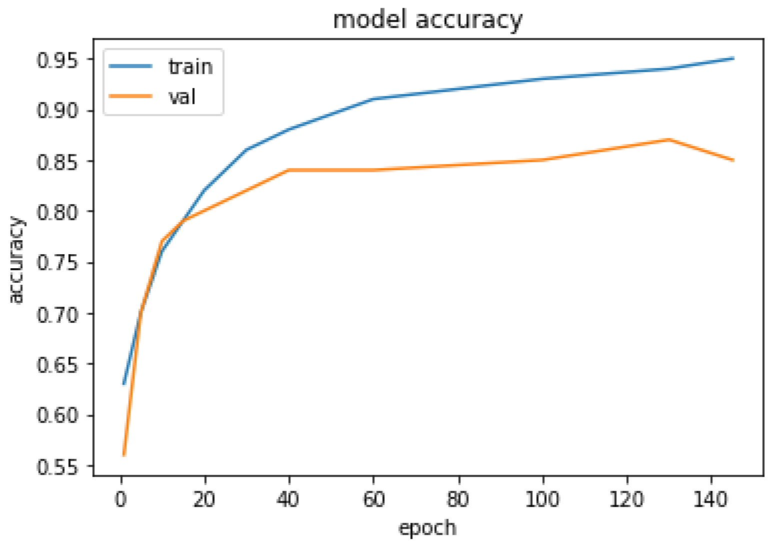
Comparing the results for non deep learning models, we find that KNN with raw pixels as input performs the best, with an accuracy of 75%. The worst model is Naive Bayes taking histogram representation as an input, with an accuracy of 28.7%. However, even the worst accuracy performs better than the random baseline of 25% (since our dataset is balanced), and the majority baseline of 26.4%.

However, once again, we didn't choose the features carefully for any of these basic non-deep learning methods so the result is more like a baseline to compare with. We believe that they would work much better if we had a larger focus on them, especially if features like edge detection are flushed out.

4.2 Deep learning results

4.2.1 Inception results

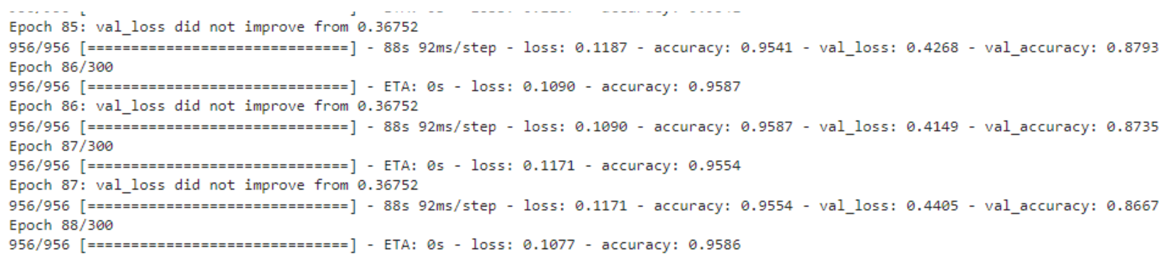
For transfer learning with Inception, the best result we got is 88% validation accuracy and a 96% training accuracy. As the figure below shows, the gap between train accuracy and validation accuracy is massive while training accuracy is converging.



A plausible explanation for poor performance is that we are not utilizing the strength of the Inception model here. Inception, as mentioned, deals with the problem of variance in location of the important information in different images. However, for our task, most of the important information in the brain images is located in the very central part of the image, so it's not necessary for us to use Inception here. In addition, since the simple model we wrote can fit the training data perfectly as mentioned, this huge model is prone to overfit because of its high complexity.

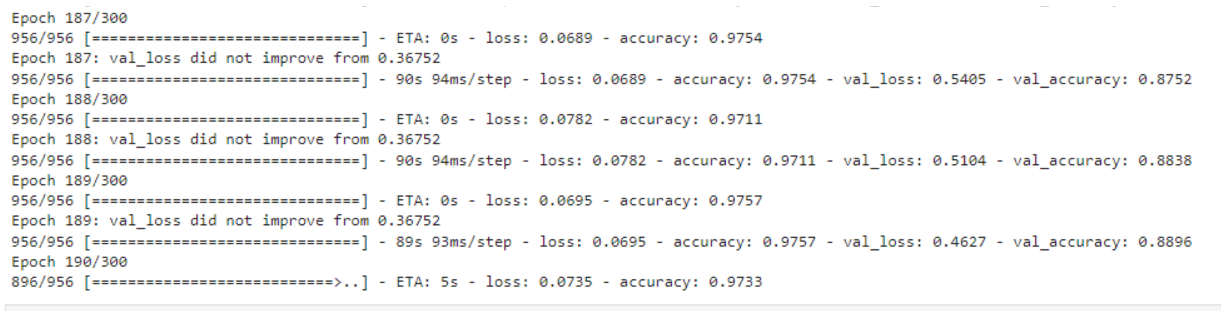
4.2.2 NASNet results

For transfer learning with NASNet, the best result we got is 88% validation accuracy and 97% training accuracy. As the figures below shows, the gap between train accuracy and validation accuracy is always huge while training accuracy is converging. Around epoch 85, the training accuracy is 95% and validation accuracy is 87%.



```
Epoch 85: val_loss did not improve from 0.36752
956/956 [==============================] - 88s 92ms/step - loss: 0.1187 - accuracy: 0.9541 - val_loss: 0.4268 - val_accuracy: 0.8793
Epoch 86/300
956/956 [==============================] - ETA: 0s - loss: 0.1090 - accuracy: 0.9587
Epoch 86: val_loss did not improve from 0.36752
956/956 [==============================] - 88s 92ms/step - loss: 0.1090 - accuracy: 0.9587 - val_loss: 0.4149 - val_accuracy: 0.8735
Epoch 87/300
956/956 [==============================] - ETA: 0s - loss: 0.1171 - accuracy: 0.9554
Epoch 87: val_loss did not improve from 0.36752
956/956 [==============================] - 88s 92ms/step - loss: 0.1171 - accuracy: 0.9554 - val_loss: 0.4405 - val_accuracy: 0.8667
Epoch 88/300
956/956 [==============================] - ETA: 0s - loss: 0.1077 - accuracy: 0.9586
```

However, the following screenshot shows that the validation accuracy doesn't improve after epoch 90, and validation loss is oscillating.

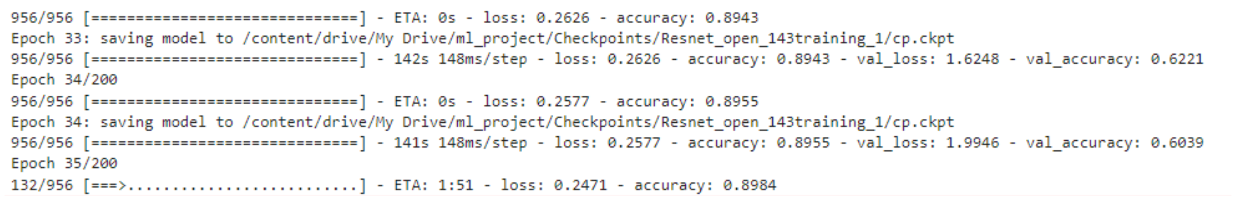

```
Epoch 187/300
956/956 [==============================] - ETA: 0s - loss: 0.0689 - accuracy: 0.9754
Epoch 187: val_loss did not improve from 0.36752
956/956 [==============================] - 90s 94ms/step - loss: 0.0689 - accuracy: 0.9754 - val_loss: 0.5405 - val_accuracy: 0.8752
Epoch 188/300
956/956 [==============================] - ETA: 0s - loss: 0.0782 - accuracy: 0.9711
Epoch 188: val_loss did not improve from 0.36752
956/956 [==============================] - 90s 94ms/step - loss: 0.0782 - accuracy: 0.9711 - val_loss: 0.5104 - val_accuracy: 0.8838
Epoch 189/300
956/956 [==============================] - ETA: 0s - loss: 0.0695 - accuracy: 0.9757
Epoch 189: val_loss did not improve from 0.36752
956/956 [==============================] - 89s 93ms/step - loss: 0.0695 - accuracy: 0.9757 - val_loss: 0.4627 - val_accuracy: 0.8896
Epoch 190/300
896/956 [===========================>..] - ETA: 5s - loss: 0.0735 - accuracy: 0.9733
```

Our explanation that NASNet is not working well is that we are not taking advantage of the strength of the NASNet model here. NASNet's strength is to choose convolution layers more wisely in its pretrained process. These chosen layers are helpful for the NASNet pre-trained task and maybe many other more complicated tasks in general, but for our task, it might not be the best choice because the size of the model is unnecessarily big and complex, so it's prone to overfit.

Instead of using the pretrained model, if we train a NASNet model of our own based on our dataset, the results might be a lot more promising.

4.2.3 ResNet results

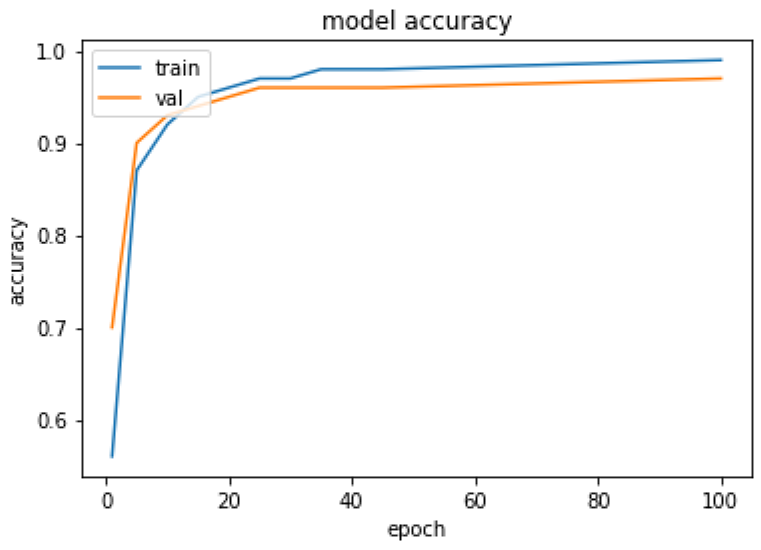
For transfer learning with ResNet, the best result we got is 62% validation accuracy and 90% training accuracy. The accuracy numbers here are somewhat meaningless because the validation loss is very high while the training loss is converging to 0, as the figure below shows. So it's no different from taking a random guess when evaluating this model in the validation dataset. Yet, this is the best result we've got from transfer learning with ResNet (when opening layers after layer 143). The rest of the variants model either can't fit or overfit just as this one does. The difference between the result of ResNet and previous pretrained model is that it can't fit the training data when the pretrained model is non-trainable, while NASNet and Inception still tend to overfit in that case.


```
956/956 [==============================] - ETA: 0s - loss: 0.2626 - accuracy: 0.8943
Epoch 33: saving model to /content/drive/My Drive/ml_project/Checkpoints/Resnet_open_143training_1/cp.ckpt
956/956 [==============================] - 142s 148ms/step - loss: 0.2626 - accuracy: 0.8943 - val_loss: 1.6248 - val_accuracy: 0.6221
Epoch 34/200
956/956 [==============================] - ETA: 0s - loss: 0.2577 - accuracy: 0.8955
Epoch 34: saving model to /content/drive/My Drive/ml_project/Checkpoints/Resnet_open_143training_1/cp.ckpt
956/956 [==============================] - 141s 148ms/step - loss: 0.2577 - accuracy: 0.8955 - val_loss: 1.9946 - val_accuracy: 0.6039
Epoch 35/200
132/956 [===>..........................] - ETA: 1:51 - loss: 0.2471 - accuracy: 0.8984
```

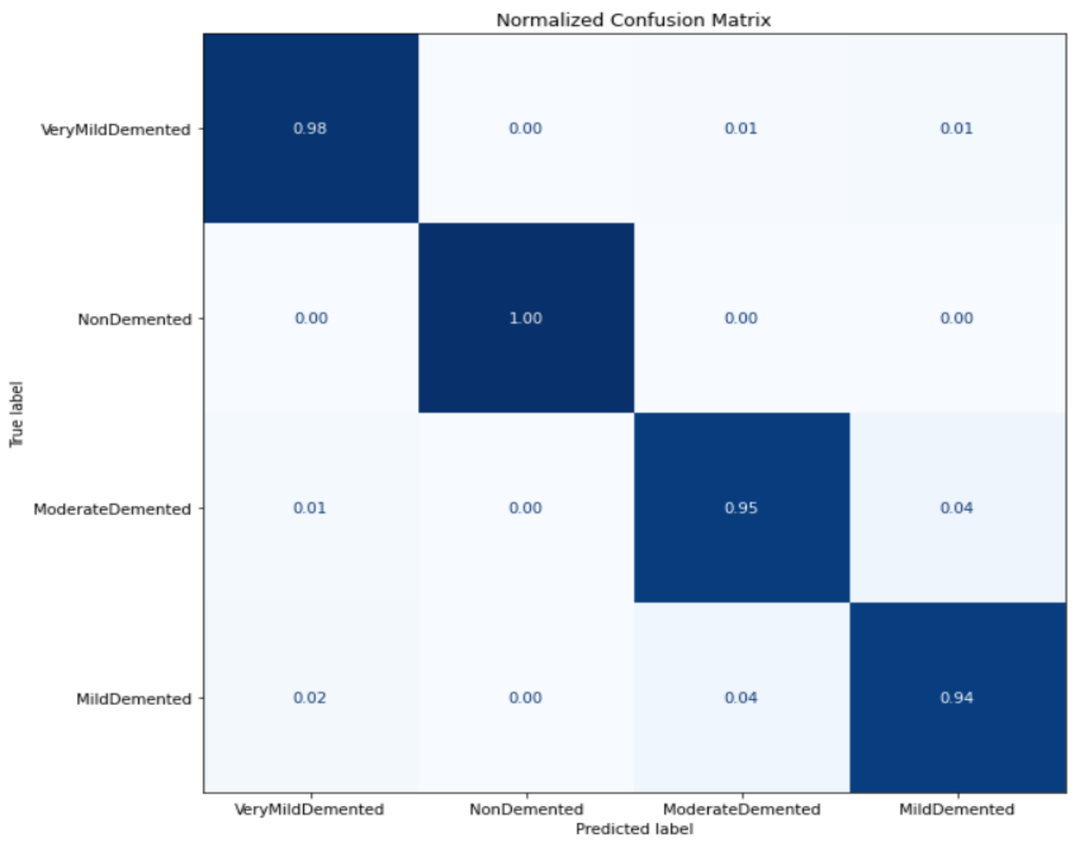
Our explanation for ResNet's poor performance is that we are not using the strength of the ResNet model here. ResNet's strength is its ability to solve the problem of the vanishing gradient; however, for our task, we do not have enough layers for this vanishing gradient to be problematic. In addition, since the simple model we wrote can fit the training data perfectly as mentioned, this huge model is prone to overfit because of its high complexity.

4.2.4 Custom CNN based model results

Previous failures with the transfer learnings lead us to creating our own version of a CNN based model, which gives us the best result. As the figure shows, the model training accuracy converges very fast in comparison to the previous transfer learning models. At the same time, the gap between training accuracy and validation accuracy is very small, towards the end of around epoch 100 it gives us the validation accuracy around 97% (oscillating around this number), and a training accuracy approaching 100% slowly.



This figure below is the confusion matrix of this best result model when evaluated in the validation dataset. It has perfect accuracy for the class non demented, which means that if our task is binary classification, we will most likely get a perfect validation accuracy. For the rest of the three classes, there are more confusion between moderate and mild demented classes, which makes sense since these two classes are rather similiar and thus prone to be confused even by medical professionals.



Although the model is the best model for our particular dataset, it might not be the best model in the real world or with other input data. Overall, for our dataset the performance is very good considering time needed to train and also the accuracy of its predictions. The complexity of this model is also the best we've tried so far, since it fits the training data perfectly and resulted in the smallest gap between validation and training loss. We wanted to use cross validation on our dataset but since we did the train/test split differently many times and evaluated them and the resulting accuracy is always the same, we can confidently say that cross validation will give the same good results.

5. Conclusions

When applying deep learning methods for this classification task on Alzheimer's disease progression, a simple CNN model would be a better choice over the complex classic models. Given the experience we have now, our approach might be different: we would start from simple CNN models instead of starting with transfer learnings. This is also the most important "take-home message" in our result: Occam's razor. Don't overthink on any task, try easier models first, find the problem, and then try the large ones. Otherwise, there might be a great waste on computational resources when you start with many unnecessary overfitting models. However, despite testing these unnecessary models, we did arrive at a result that achieved our goal of correctly classifying Alzheiemer's disease in an MRI scan. Our final results are on par with prior research described in the Introduction.

There is also more work to be done in the future. Our model could be tried on other datasets on this same topic. MRI scans can vary depending on the dataset, so testing our model on other datasets of MRI scans could determine if it has more real-world applications. Additionally, there is still room for improvement in our CNN model to achieve better accuracy. We could test different building blocks in our model architecture to see how we can further improve the model. Finally, although the transfer learning models did not yield the best results, we can create our own architecture that uses the building blocks of NASNet that works for our task.

Overall, the results of our project show the power and importance of machine learning. Being able to correctly classify an MRI scan (or any type of medical scan, such as an X-Ray or mammography) is a lifesaving technique. Early diagnosis of diseases like Alzehiemer's is critical in combating the disease as it gets worse. Having a model that can classify such images at a 97% accuracy will take the pressure off of medical professionals to identify life-threatening symptoms.

Here's our github repository link:
https://github.com/zhangcoj/ML_Project.git

6. References

AbdulAzeem, Y., Bahgat, W. M., & Badawy, M. (2021). A CNN based framework for classification of Alzheimer's disease. *Neural Computing and Applications*, 33(16), 10415-10428.

Alzheimer MRI 4 Classify Fasiai. Retrieved September 30, 2022 from hhttps://www.kaggle.com/code/stpeteishii/alzheimer-mri-4-classify-fasiai/notebook.

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).

Sheng, J., Xin, Y., Zhang, Q., Wang, L., Yang, Z., & Yin, J. (2022). *Predictive classification of Alzheimer's disease using brain imaging and genetic data.* Scientific Reports, 12(1), 1-9.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).

Tsang, S.-H. (2021, July 27). *Review: Nasnet-neural architecture search network (image classification)*. Medium. Retrieved December 9, 2022, from https://sh-tsang.medium.com/review-nasnet-neural-architecture-search-network-image-cl assification-23139ea0425d.

Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 8697-8710).