Final Year Project Report

Full Unit - Final Report

Comparison of Image Classification Models with Transfer Learning

James Arnott

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Li Zhang

Department of Computer Science

Royal Holloway, University of London

March 6, 2023

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: XXXX

Student Name: James Arnott

Date of Submission: March 6, 2023

Video Link:

Signature:

Table of Contents

Abstract

Transfer learning is a common, efficient method for training deep learning models. It involves using a pre-trained model to extract features from a dataset, and then training a new model on top of the extracted features. This project report gives a comparison of the performance of several image classification models when using transfer learning and evaluates their performance on a dataset with MRI scans of Alzheimer's patients. The models compared are MobileNetv2, MobileNetv3-Small, MobileNetv3-Large, InceptionV3, ResNet101 and VGG19.

Project Specification

Aims: To implement and compare on benchmark data sets various machine learning algorithms Background: Machine learning allows us to write computer programs to solve many complex problems: instead of solving a problem directly, the program can learn to solve a class of problems given a training set produced by a teacher. This project will involve implementing a range of machine learning algorithms, from the simplest to sophisticated, and studying their empirical performance using methods such as cross-validation and ROC analysis. In this project you will learn valuable skills prized by employers using data mining. Early Deliverables

- You will implement simple machine learning algorithms such as nearest neighbours and decision trees.

- They will be tested using simple artificial data sets.

- Report: a description of 1-nearest neighbour and k-nearest neighbours algorithms, with different strategies for breaking the ties;

- Report: a description of decision trees using different measures of uniformity.

Final Deliverables

- May include nearest neighbours using kernels and multi-class support vector machines.

- The algorithms will be studied empirically on benchmark data sets such as those available from the UCI data repository and the Delve repository.

- For many of these data set judicious preprocessing (including normalisation of attributes or examples) will be essential.

- The performance of all algorithms will be explored using a hold-out test set and cross-validation.

- The overall program will have a full object-oriented design, with a full implementation life cycle using modern software engineering principles.

- Ideally, it will have a graphical user interface.

- The report will describe: the theory behind the algorithms.

- The report will describe: the implementation issues necessary to apply the theory.

- The report will describe: the software engineering process involved in generating your software.

- The report will describe: computational experiments with different data sets and parameters.

Suggested Extensions

- Modifications of known algorithms.

- Dealing with machine learning problems with asymmetrical errors (such as spam detection) and ROC analysis.

- Nontrivial adaptations of known algorithms to applications in a specific area, such as medical diagnosis or option pricing.

- Exploring the cross-validation procedure, in particular the leave-one out procedure. What is the optional number of folds?

- Comparative study of different strategies of reducing multi-class classifiers to binary classifiers (such as one-against-one, one-against-the-rest, coding-based).

Reading

- Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The Elements of Statistical Learning. Second edition. Springer, New York, 2009.

- Tom M. Mitchell. Machine Learning. McGrow-Hill, New York, 1997.

- Vladimir N. Vapnik. The Nature of Statistical Learning Theory. Second edition. Springer, New York, 2000.

- Vladimir N. Vapnik. Statistical Learning Theory. Wiley, New York, 1998.

- Seth Hettich and Steven D. Bay. The UCI KDD Archive. University of California, Department of Information and Computer Science, Irvine, CA, 1999, http://kdd.ics.uci.edu.

- Delve archive. http://www.cs.toronto.edu/ delve.

Addendum

As I was not keen on doing the pre-assigned generic project, I had asked for permission to do a project of my own choosing. I was given permission to do so, and I chose to do a project on comparison of image classification models using transfer learning. I chose this project as I was interested in transfer learning as I had used it in a previous project and wanted to learn more about it and it seemed far more interesting than the generic project and I believe this is generally a more applicable problem to solve in the real world.

Chapter 1: **Introduction**

My project is about the comparison of image classification models with transfer learning. In this report I shall cover the background of transfer learning, the models I have chosen to compare, the datasets I have chosen to use, the results of my experiments and the conclusions I have drawn from them.

The importance of transfer learning cannot be overstated, it makes training models for anyone with a computer and an internet connection possible. It opens up possibilities for people who would otherwise not be able to train models as far less data and computing power is required.
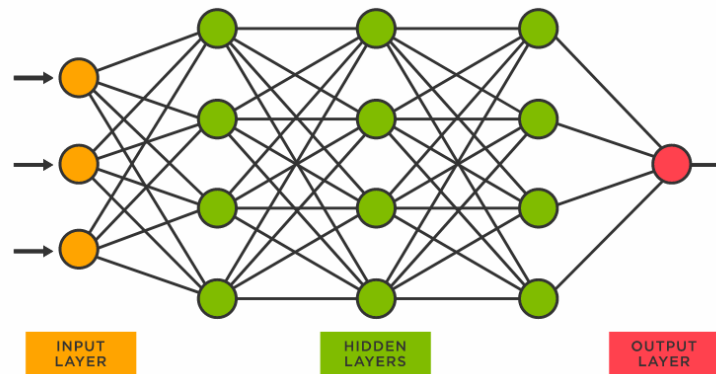
Chapter 2: **Neural Networks**

Figure 2.1: A diagram of a neural network [15]

2.1   What is a Neural Network

A neural network is a machine learning model that is inspired by the way the human brain works. It is made up of a series of layers, each layer is made up of a number of neurons. Each neuron is connected to every neuron in the next layer, and each connection has a weight associated with it. The weights are used to determine how much each neuron in the next layer is affected by the current neuron. The weights are updated during training, and the process of updating the weights is known as backpropagation.

2.2   Why use a Neural Network

The use of neural networks has increased dramatically in recent years, and they are now used in a variety of different applications. The reason for this is that they are very good at learning complex patterns in data, and they are able to learn these patterns without being explicitly programmed to do so. This saves a lot of time and effort when compared to other machine learning techniques.

They are far more flexible than many of the traditional strategies for image classification, due to the fact they can adapt to new data and are resilient to noise in the data or changes such as lighting or rotation.

2.3 What are the alternatives to Neural Networks in Image Classification

This is not an exhaustive list for image classification, but it is a list of some of the most common alternatives to neural networks that produce the best results.

2.3.1 Support Vector Machines

There are a number of different alternatives to neural networks, one common alternative is Support Vector Machines. Support Vector Machines are a type of supervised machine learning model that can be used for both classification and regression. They work by finding a hyperplane that separates the data into different classes, and then classifying new data based on which side of the hyperplane it is on. They are very good at classifying data that is linearly separable, otherwise they are not as good, and they are also very sensitive to noise in the data.

In one study using SVMs to classify Alzheimer's patients, the researchers managed to get an accuracy of 62.64% using MRI scans of the brain. They managed to get between 83% and 90% accuracy when using SVMs with clinical parameters, however combining the two methods in this study did not improve the accuracy. [25]

2.3.2 Content Based Image Retrieval

Content Based Image Retrieval (CBIR) is a type of image retrieval system that uses the actual content of an image as the basis for retrieving similar images from a large database. It works by extracting the features from the image and then using those features to compare to other images in the database. The features are usually based on a combination of color, texture, shape, and other attributes. The system then produces a list of images that are similar to the one used for the search query. CBIR systems are particularly useful for searching for images that can't be easily described using traditional keywords.

CBIR can be used to identify images that are not in the database by analyzing the content of the images. By extracting features from the image such as color, texture, and shape, CBIR can be used to compare the content of the query image with the content of the images in the database. It can then return images that are similar in content to the query image even if they are not present in the database.

For example, if the target image was a blue sky with white clouds, then the algorithm would return images that had similar features such as a blue sky and white clouds to the target image.

Although this method of image classification would not seem to be very useful for classifying images of Alzheimer's patients, a study was done using this method to classify Alzheimer's patients, and they had managed to get an accuracy of 87% using MRI scans[1] and getting feedback from the physicians who were treating the patients.

For classifying images of flowers it could also be very useful, as it would allow the user to search for images of a specific flower. This could help identify flowers by recognizing the different characteristics of each flower, such as the shape of the petals, the color of the petals, and the overall structure of the flower.

2.4   The different optimizers

In deep learning, optimisers are algorithms used to adjust the parameters of a model in order to minimise a loss function. Optimisers are used to improve the accuracy of a model by reducing the error rate. Common optimisers used in deep learning include Stochastic Gradient Descent (SGD), Adaptive Moment Estimation (Adam), and Root Mean Square Propagation (RMSProp). There are many different optimizers that can be used to train a model, and the one that is used depends on the type of problem that is being solved.

The impact using different optimisers can have is dependent on the type of problem being solved. For example, SGD is often used for linear regression problems, while Adam is better suited for deep learning problems. Additionally, the choice of optimiser can also affect the speed of convergence and the accuracy of the model.
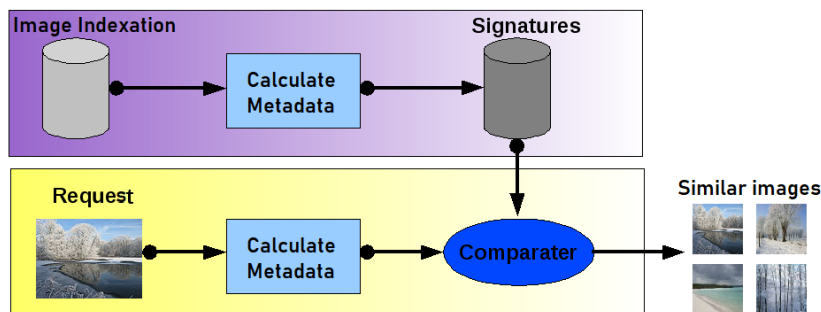

Figure 2.2: A diagram of Content Based Image Retrieval [5]

2.5   What is a Convolutional Neural Network

A convolutional neural network (CNN) is a type of neural network that is used for image classification. It is made up of a series of convolutional layers, pooling layers, and fully connected layers. The convolutional layers are used to extract features from the image, and the pooling layers are used to reduce the size of the image. The fully connected layers are used to classify the image. They are often much more efficient than other types of neural networks, and are commonly used for image classification.

2.6   Types Of Layers

- Convolutional Layer: This layer extracts features from an input image and creates a feature map.

- Pooling Layer: This layer reduces the dimensionality of a feature map while preserving its most important features.

- Dropout Layer: This layer randomly ignores nodes during training to reduce overfitting.

- Fully Connected Layer: This layer connects all the neurons of the previous layer to every neuron in the next layer. It helps in mapping input to output.

2.7   What transfer learning is

Transfer learning is a common, efficient method for training deep learning models. There are many different ways to implement transfer learning and here I hope to compare the performance of a variety of different methods with different pre-trained models.

The applications for transfer learning are vast, and it is a common method for training deep learning models, This is because it is a very efficient method for training models as most of the work is done by the pre-trained model. The pre-trained model is used to extract features from the dataset, and then a new model is trained on top of the extracted features. The pre-trained model is usually frozen so that it does not change during training, and only the new model is trained.

After the new model is trained on top, a process known as fine-tuning can be used to further improve the performance of the model. Fine-tuning involves training the new model on the dataset, but with a lower learning rate than the initial training. This is also where the model is usually unfrozen, so that the pre-trained model can be trained as well, which helps to improve the overall performance of the model.

2.8   The Datasets I will be using

The datasets I have chosen to use for this project are the Oxford 102 Flower Dataset[18] and the Alzheimer's MRI Dataset[2].

The choice of datasets is based off of the availability of the dataset, the complexity and the number of classes. The aim of the Alzheimer's dataset was to see if the model could learn subtle differences between the different classes, and if it could learn to classify the different classes. The aim with the Oxford flowers 102 dataset[18] was to see how the models would perform with a wider variety of classes but more significant differences between the classes.

Chapter 3: **Transfer Learning**

3.1　**Why use Transfer Learning**

Transfer learning can enable a model to be trained much more efficiently than if it was trained from scratch, as most of the work is done by the pre-trained model. Aside from the extra time it takes to train a model from scratch, it often requires more training data, which is not always available and can be expensive to obtain. Transfer learning can often get better results than training a model from scratch, as the pre-trained model has already learned about different features from a large dataset, and the transfer learning model can then use these features to learn about the new dataset.

3.2　**Drawbacks of Transfer Learning**

The main drawback of transfer learning is that it is not always possible to use it, as the pre-trained model may not be able to extract the features from the new dataset. With image classification, the pre-trained model can typically extract features from any image, but it may not be able to extract the features that are needed for the new dataset. Transfer learning can also be less efficient than training a model from scratch, as the pre-trained model will have all the weights from the original dataset, which may not be needed for the new dataset. This can cause the model to be less efficient, as it will have to train on weights that are not needed but will still affect the performance of the model.

3.3   The code behind the transfer learning

The code for the transfer learning is shown below.

```
# This imports the model from the keras library
MobileNetV3Small = keras.applications.MobileNetV3Small(
    input_shape=(224, 224, 3),
    include_top=False,
    weights='imagenet')

# Create the model
model = keras.Sequential()

# Freezing all but 10 layers of the pre-trained model
for layer in MobileNetV3Small.layers[:-10]:
    layer.trainable = False

# Adding the pre-trained model to the model, along with
# a dropout layer and a fully connected layer
model.add(MobileNetV3Small)
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(512, activation='relu'))
model.add(keras.layers.Dropout(0.6))
model.add(keras.layers.Dense(512, activation='relu'))
model.add(keras.layers.Dropout(0.6))
model.add(keras.layers.Dense(4, activation='softmax'))
```

The code above shows how to create a model using transfer learning, creating the model structure consists of adding the pre-trained model to the model, and then adding a few layers on top of it. The Flatten layer is used to flatten the output of the pre-trained model as that is the input for the fully connected layers. The fully connected layers are used to classify the images.

The dropout layers are used to prevent overfitting, and the dense layers are used to classify the images. The dense layers are potentially the most important part of the model, as they are the ones that are actually learning about the dataset. They get trained on the features extracted by the pre-trained model, and the weights of the dense layers are what is used to classify the images.

The final layer of the model is a Softmax layer, which is used to classify the images. It is used to classify the images into one of the classes in the dataset, in this example, the 4 levels of Alzheimer's disease that are in the Alzheimer's MRI Dataset.

3.4   How to implement Transfer Learning

3.4.1   Extracting features

Feature extraction is the process of extracting features from the images in the dataset, these features in the images could be anything, such as the colour, shape, or the texture of the image. All of these features are used to classify the images, and having these features already extracted can make the training process much faster. Most of the pre-trained model is usually frozen so that it does not change during the initial stages training, and only the new model is trained, and on top of the extracted features from the base model.

3.4.2   Fine-tuning

This is the process of training the new model on the dataset, but with a lower learning rate than the initial training and the model is usually unfrozen, so that the pre-trained model can be trained as well, which helps to improve the overall performance of the model. This was where I managed to get the best results for my models.

3.5   How to fine-tune a model

Fine-tuning a model is a very simple process, and can be done in just a few lines of code. The first step is to unfreeze the pre-trained model, which can be done by setting the trainable attribute of the layers to True. The next step is to set the learning rate to a lower value, which can be done by setting the learning rate attribute of the optimizer. The final step is to recompile the model, which can be done by calling the compile method of the model. Here is an example of how I fine-tuned a model using Keras:

```
base_model.trainable = True
optimizer.learning_rate = 0.0001
model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
```

```
base_model.trainable = True
optimizer.learning_rate = 0.0001
model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
```

3.6   My training setup

For this project, I have set up my gaming laptop as a server, with Manjaro[13] as the operating system. I've chosen my laptop to do the training on due to it's powerful GPU, which is a NVIDIA GeForce GTX1070[16]. This can be used with Nvidia CUDA[6] to speed up the training process, which is what I have done. Training the models on my laptop has been very useful, as it has allowed me to train the models much easier as I can leave it to train overnight without having to constantly check on it.

Chapter 4: **Evaluating the performance**

4.1 The different metrics

Accuracy is typically the metric used to compare models and is defined as the number of correct predictions divided by the total number of predictions. It is a very simple metric to understand, and is therefore often used to compare the performance of different models. There is accuracy for both the training and validation sets, and the accuracy of the validation set is used to determine how well the model generalises to new data, and a validation accuracy that is much lower than the training accuracy is a sign that the model is overfitting. However, it is not always the best metric to use, as it can be misleading in some cases.

Loss is a very common and useful method to evaluate the performance of a model, it is calculated by taking the difference between the predicted value and the actual value. There are however many different types of loss functions, and the one that is used depends on the type of problem that is being solved. These could be binary cross entropy, categorical cross entropy, or mean squared error. For my project I used binary cross entropy due to it's simplicity and the fact that it is the most common loss function used for binary classification problems.

The validation loss and accuracy are used to determine how well the model generalises to new data, and a validation loss that is much higher than the training loss is a sign that the model is overfitting. For the validation set, I have assigned 20% of the data to be used.

4.2 Data Augmentation

Data augmentation is the process of artificially increasing the size of a dataset by applying random transformations or filters to the images. This can help to improve the performance of a model, as it can help to reduce overfitting, and can also help to improve the generalisation of the model. The transformations that are applied to the images are usually random, and can include things like rotating the image, flipping the image, or changing the contrast/saturation of the image.

In the case of the Alzheimer's MRI Dataset, it is extremely useful to have more data to train the model on, as there are a limited number of brain scans available. MRI scans are also very expensive, so using data augmentation can significantly reduce the cost of training the model.

In the Alzheimer's MRI Dataset, the majority of the archive consists of augmented images already however this caused me problems which I will go on to talk about. With the flower dataset however, I did apply data augmentation to the images, as there were not enough images per class to train the model on.

4.3 Problems with the Alzheimer's MRI Dataset and how I fixed them

4.3.1 The problems

The Alzheimer's MRI dataset that I had chosen early on in the project was far from ideal as I found out later on. The pre-augmentation that was applied to the dataset caused problems with the training/validation split, as the images were already augmented, and the validation set was made up of images that were in the training set, but with different transformations applied to them. This gave me amazing results on the validation set, but when I tested the model on new data, it performed very poorly. Here the image shows the last 30 epochs of the training process, and the validation accuracy is very high, higher than what other models from other research papers have achieved.
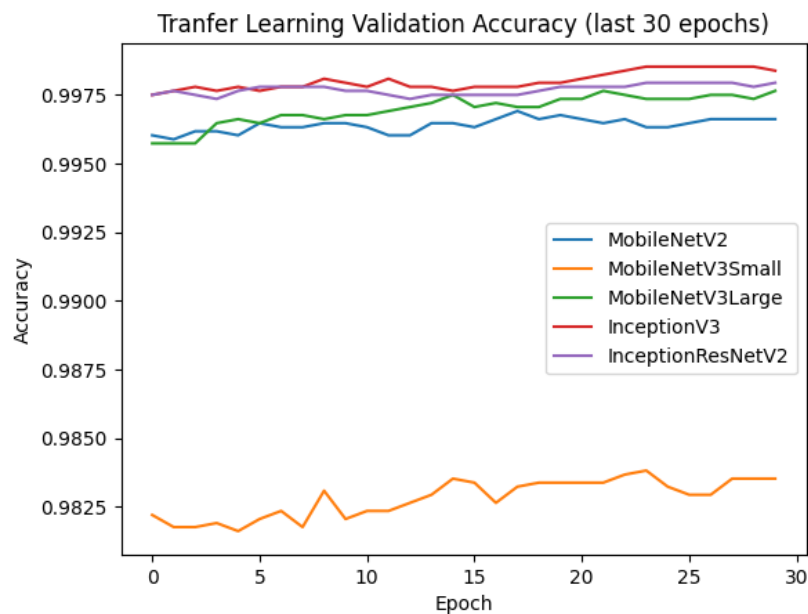

Figure 4.1: The training results from the various models (Last 30 epochs)

4.3.2 The solution

The solution to this problem was to use the original images, and not the pre-augmented images. This meant that I had to re-download the dataset, and re-split the dataset into training and validation sets, and then apply data augmentation to the images. This gave me much better results, and the model was able to generalise to new data much better. Here is the image of the training results from the various models, and the validation accuracy is much lower than before.

Here's the snippet of code that I used to augment the images:

```
train_datagen = ImageDataGenerator(rescale=1./255,
    validation_split=0.2,
    featurewise_center=True,
    featurewise_std_normalization=True,
    rotation_range=360,
    width_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    vertical_flip=True,
)
```

4.4   Evaluating the given metrics

The metrics of accuracy and loss generally tell us how well the model is performing, but they do not tell us how well the model is performing for each class. They do also not tell us how well the model would work in a real world scenario, as they are only calculated on the training and validation sets, of which we have a limited amount of data.

The validation dataset is also from the same distribution as the training dataset, so it is not a good representation of how well the model would work in a real world scenario as the scans could be of a different quality.

I have used the augmented dataset for the training and validation sets as this should give a better representation of how well the model would work in a real world scenario.

Figure 4.2: A confusion Matrix[4]

For the final report I intend on using confusion matrices to evaluate the performance of the models, as they can show which classes the model has the best or worst performance on. They can also represent nicely in a table, all the different hyperparameter combinations that I will try to use to get the best possible results.
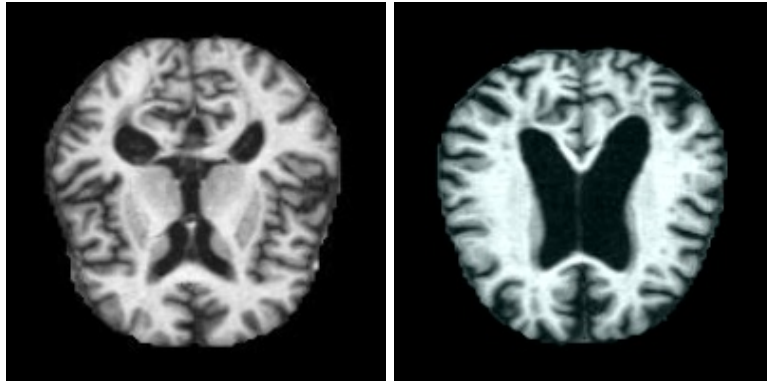
4.5   The different datasets

Figure 4.3: Samples from the Alzheimer's MRI Dataset[2]

The Alzheimer's MRI Dataset is a dataset that contains 4 different levels of Alzheimer's disease, it contains 33984 images in the augmentation folder and the images are all 256x256 pixels in size. They require resizing to 224x224 pixels in order to be used with the pre-trained models as they are expecting images of that size. Not using all the resolution of the images is a problem as it could reduce the accuracy of the model, however it is a necessary due to the models being trained on images of that size.
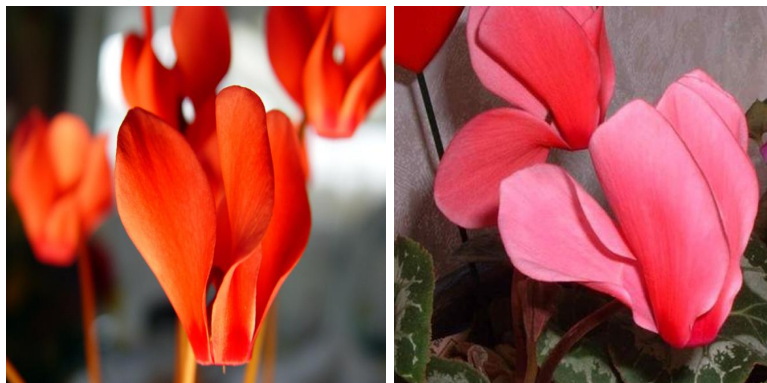

Figure 4.4: Samples from the Oxford 102 Flower Dataset[18]

The Oxford 102 Flower Dataset is a dataset that contains 102 different types of flowers, the images are all different sizes, with 3 channels for colour and are in the JPG format. I have chosen this dataset as it contains a wide variety of images with different backgrounds and lighting conditions, which should help to test the generalisation of the model, along with the ability for the base model to extract features from the images. Per class, there are fewer examples than in the Alzheimer's MRI Dataset, however there are still enough to train the model on. This will be a good test for the various base models to see which one is the best at extracting features from the images. For the interim report, I did not use this dataset as I was having too many problems with it, and the training time was too long for me to be able to have adequate results for the interim report.

Chapter 5: **The different models that are being compared**

All the pre-trained models are from the Keras library[12] and were trained on the ImageNet dataset[10].

5.1   Mobilenetv3

As described in the "Searching for Mobilenetv3" paper[8], this model is designed for mobile devices, and is a successor to the Mobilenetv2 model. It is a comparatively small model as it is optimised for mobile devices, and is also very fast.

The resulting models from this paper are:

- Mobilenetv3-Large

- Mobilenetv3-Small

The idea to have a large and small model is to specialise the model for different use cases and different hardware, increasing the speed or accuracy of the model. The large model is designed for high accuracy, and the small model is designed for high speed, there may also be cases where the smaller model is the only viable option due to hardware constraints.
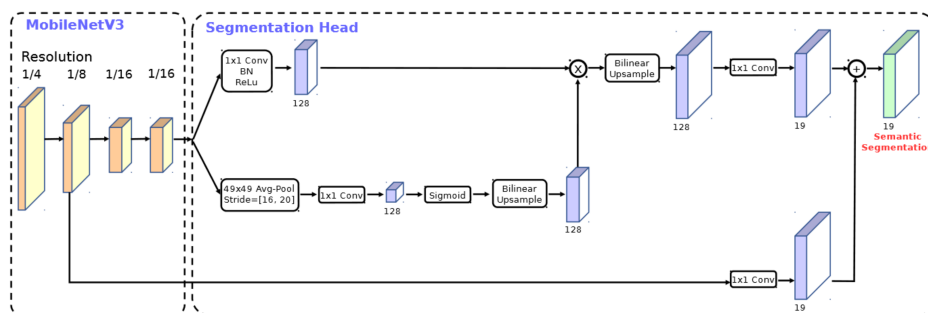

Figure 5.1: The proposed segmentation head for MobileNetV3[8]

5.2   InceptionV3

InceptionV3[23], it's one of the most popular models for image classification as it is comparatively fast and accurate. It's feature extraction capabilities are much more accurate than traditional methods such as hand-crafted feature extraction. It is a deep learning model that uses a combination of convolutional layers, pooling layers, and fully-connected layers. It uses auxiliary classifiers, these are additional classifiers used in a neural network to predict the output of the main classifier. Auxiliary classifiers are used to improve the accuracy of the main classifier by providing additional input data. They can also help to reduce the amount of overfitting that can occur when using a single classifier.
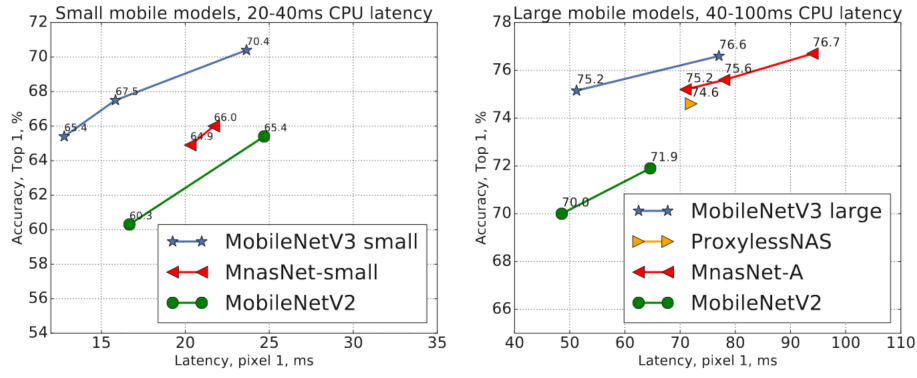
Figure 5.2: A comparison of latency and accuracy[8]

5.3   ResNet101

Resnet101[7] is another popular model, it's the largest model that is being compared and different due to the amount of layers that it has, which helps with the accuracy of the model, however it is also much slower than the other models to train and to run inference on.

5.4   VGG19

VGG stands for Visual Geometry Group, it is a model that was developed by the Visual Geometry Group at Oxford University. VGG19[22] is a model which has 19 convolutional layers
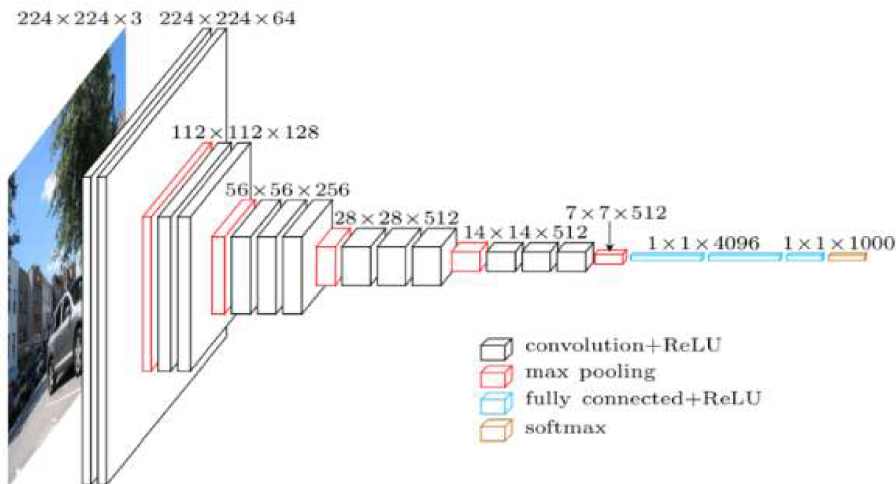

Figure 5.3: The structure of VGG19[22]

Chapter 6: **Software Engineering**

6.1   The code

The way I have structured the code is by having a main.py file that contains the code to train the model, and a Main.ipynb file that contains the code to test setting up the model.

6.2   My Development Environment

I have used the following tools to develop this project:

- Python 3.8.5[20] - To run the code

- Jupyter Notebook[11] - To help me develop the code and generate the graphs

- Visual Studio Code[26] - As my IDE to develop in, I have used the SSH feature of VsCode to SSH into my Linux server and modify the code there quickly

- GitLab[21] - To store the code and to track the changes

- tmux[24] - To run the code in the background

- CUDA[6] - To train the models on the GPU

6.3   Using Git

I have used Git to manage the code, and have used the RHUL GitLab server to host the code. This was not the first time I have used Git, however it was the first time I have used it with GitLab. I used separate branches for the different stages of the project, and then merged them into the master branch when they were complete. The branches were made to create the different planning papers, and then to set up the development environment. I also used branches to create the different models, and then to test them too.

All of my commits were in the format of Commitlint[3] to make it easier to read the commit messages, make them more consistent, and to make it easier to go through the history and find a specific change or commit to cherry pick or revert.

6.4   Testing with Jupyter Notebook

I have used Jupyter Notebook to test the code, and to generate the graphs. This was the most useful tool for me as it allowed me to test the code quickly and easily, I had many issues with setting up the code as this was the first time I have trained any model.

6.5 Using tmux

I have used tmux to run the code in the background, this was useful as I could SSH into my server and run the code, and then disconnect from the server. I would then periodically SSH back into the server to check on the progress of the code, and to see if there were any errors. This wasn't ideal as I could have set up a canary to email me if there were any errors, however I didn't have time to do this.

6.6 Using Test Driven Development

I used PyTest[19] to test the code, and to make sure that everything was working as expected. This helped me quickly test and debug my code, I would write a test to define how I wanted my code to work, and then write the code to make the test pass. This sped up development as I could quickly test the code and make sure that it was working as expected.

The other tests I wrote were making the graphs that I generated, as I was testing the code by generating the graphs using Jupyter Notebook[11] and matplotlib[14], and then checking that the graphs were looking correct. This was really the only way I could test the training code, as the models take 10+ hours to train, and I didn't have the time to train the models multiple times to see if unit tests that I had written were correct.

In the future I would like to use unit tests as I could set up an automated pipeline to train the models and then run the unit tests, and then if the unit tests fail, then the pipeline would fail, and I would be notified. This would allow me to test the code more easily, and to make sure that the code is working correctly. I would also be able to test the code on a smaller dataset, which would allow me to train the models quicker, and to test the code quicker.

Aside from testing the models, I could set up unit tests for an API that I could create for anyone to upload their brain scans to and get a prediction of whether they have Alzheimer's or not. This would give me an opportunity to write unit tests for the API, and to test the code that I would write for the API.

6.7 Different learning rates

When training a model, the learning rate is the rate at which the model learns from the data. A low learning rate will cause the model to learn slower, and a high learning rate will cause the model to learn quicker but it can also cause the model to get stuck in a local minimum. A combination of a high learning rate whilst the base model is frozen, and a low learning rate whilst the base model is unfrozen is a good way to train a model as it allows the model to learn quickly at the start, and then fine-tune the model at a slower rate.

A local minimum is when the model can't decrease its loss any further by adjusting its parameters. It is stuck at a 'valley' in the loss landscape with the parameters that it currently has. A global minimum is the same concept, but it is the absolute lowest the model can take its loss. It is the lowest point in the entire loss landscape no matter what parameters the model has.

6.8   Batch sizes

The batch size is the number of images that are passed to the model at a time. A larger batch size will cause the model to learn quicker, however it will also use more memory whereas having a smaller batch size will cause the model to learn slower, however it will use less memory. When I started off, I did not understand the concept of batch sizes, and I was using a batch size of 1000, this lead to my training times being very long and I had to wait days for a model to train. I have found that a batch size of 32 works well for the models that I have trained and my GTX1070 8 GB GPU[16] doesn't often run out of memory.

6.9   Dataset Augmentation

When training a deep learning model, it is important to have a large dataset, as this will allow the model to learn from a wide variety of data. Unfortunately I do not have as many MRI scans as I would like, so I have used data augmentation to increase the size of the dataset. Using the ImageDataGenerator class from Keras[12], I have been able to generate new the new images. This helps training the model as it allows the model to learn from a wider variety of data, and it also helps to prevent overfitting. It is not optimal to be required to use data augmentation, however it's significantly better than not having enough data to train the model. If these models were to be used in a real world scenario, augmentation helps to make the model more resilient to the quality of the scans.

Chapter 7: **Identifying annd Mitigating Bias**

The OASIS-1 dataset[17] that I used for this project contains 416 patients with 434 scans.

7.1   nWBV vs Age

Giving the age of the patient to the machine learning algorithm may allow for bias, this is because there are very few patients who are young and have Alzheimer's disease, in the dataset. This also means it would likely be problematic to give the normalized whole brain volume to the machine learning algorithm, as this is likely to be highly correlated with age, and will therefore allow for bias.
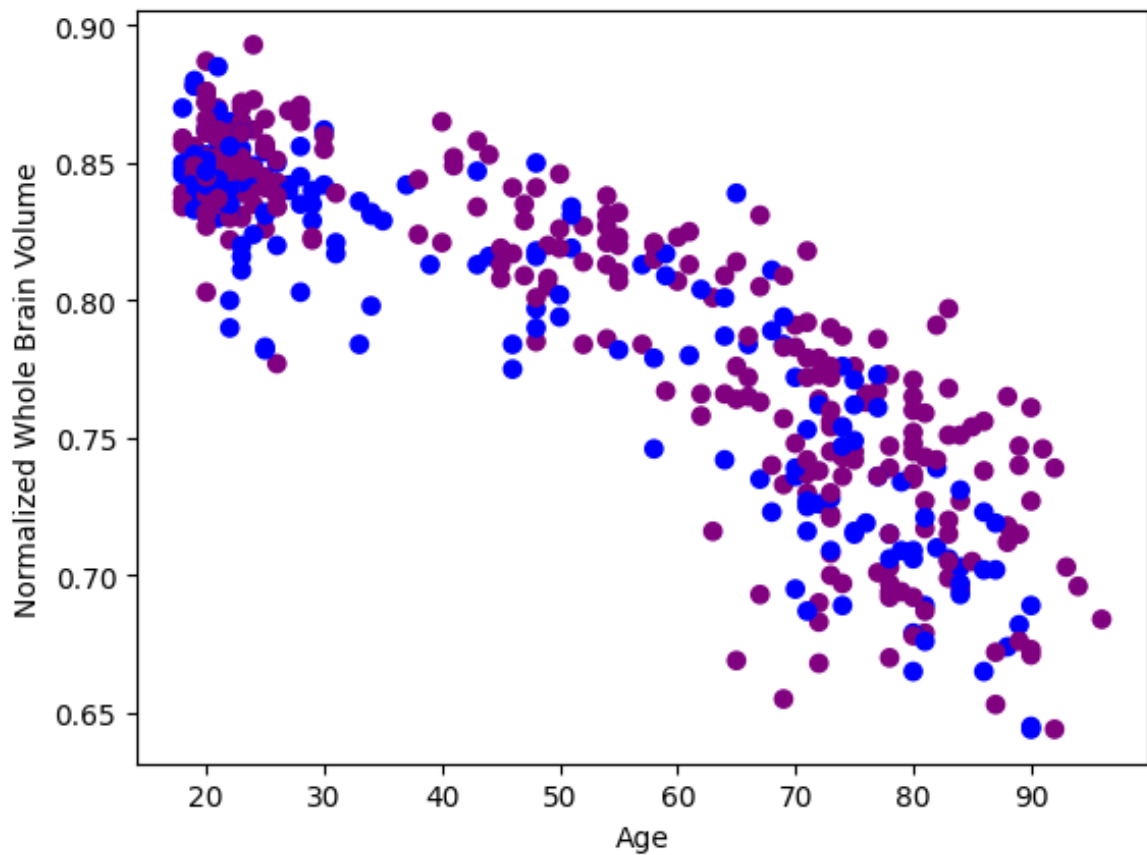
Figure 7.1: The nWBV of the patients in the dataset

7.2   eTIV vs Age

This scatter graph is significantly more random than the one for age and whole brain volume, which means it is likely better to give to the machine learning algorithm, as it shouldn't give any strong indication of the age of the patient, and it should be useful for the machine learning algorithm. One potential issue with this parameter is that women are more likely to have Alzheimer's disease, and this graph shows how the men and women are separated, which may allow for bias based off of the gender of the patient.
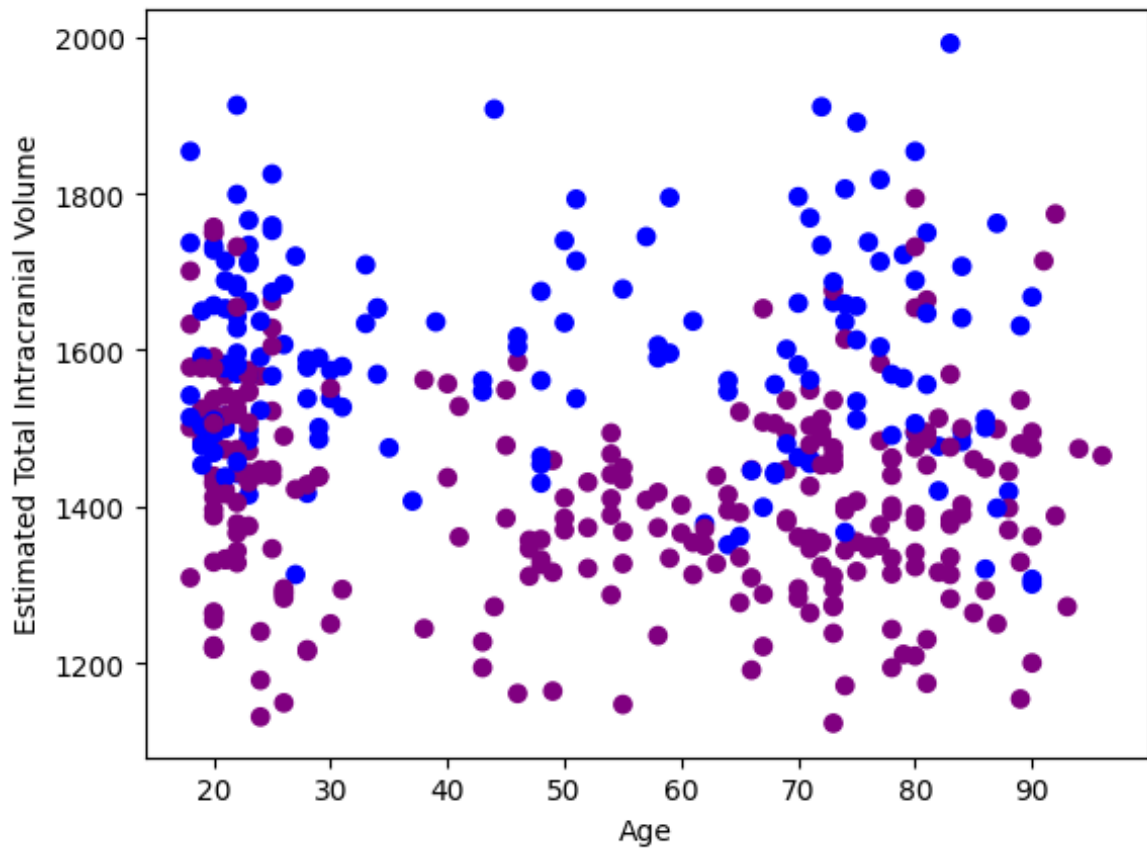

Figure 7.2: The eTIV of the patients in the dataset

Chapter 8: **My initial training results**

8.0.1   The comparison of the different models

Here is a graph of all the different models that I have trained properly with the Alzheimer's dataset. The graph shows the accuracy of the models on the validation set along with the accuracy on the training set.
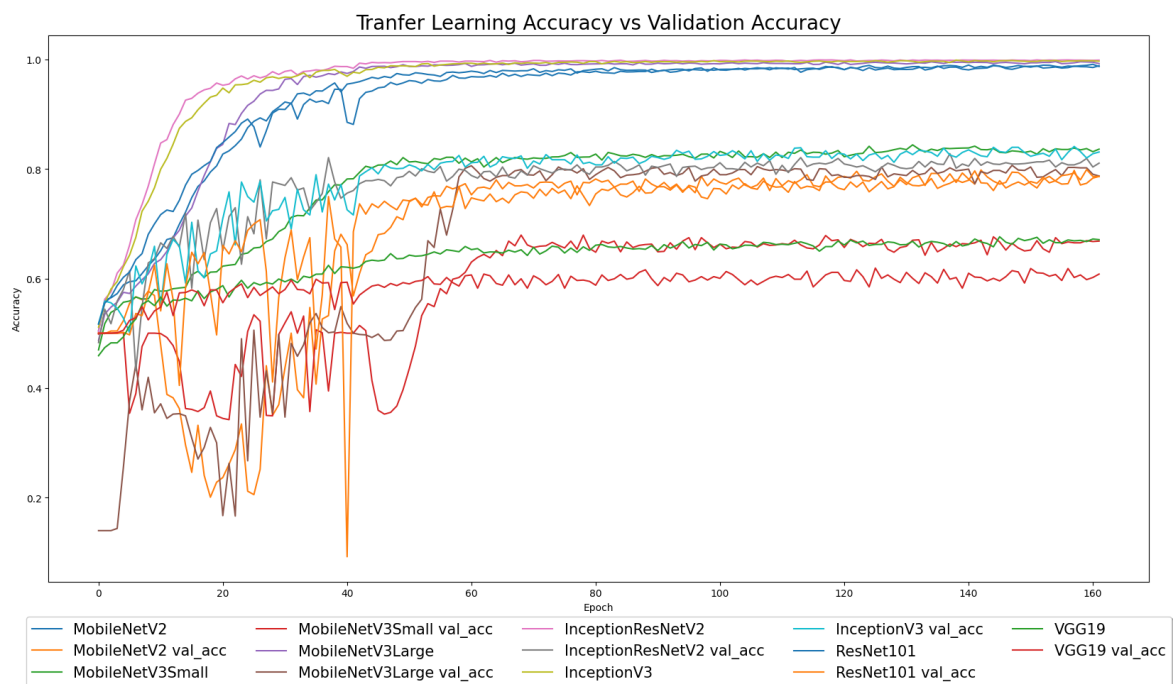

Figure 8.1: The accuracy of the different models

8.0.2 InceptionV3

For my training results, when everything was working correctly, I was able to get the highest validation accuracy of 84% and a loss lowest of 0.30.
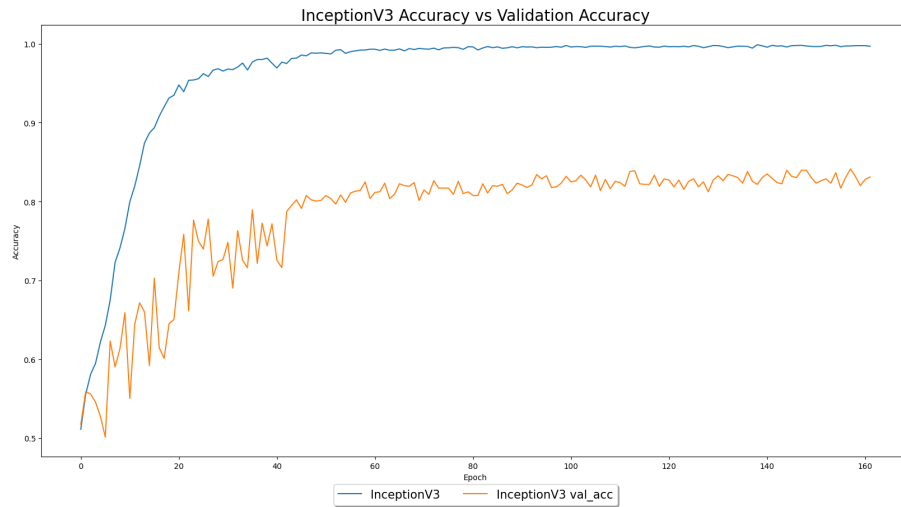

Figure 8.2: The training results using InceptionV3[23] as the base model

This graph clearly indicates that there was overfitting, as the validation accuracy is significantly lower than the training accuracy. This is due to the fact that the model is learning from the training data, and it is not generalising well to the validation data. To help mitigate overfitting, I did use data augmentation and dropout layers, however I think that the base model it's self is overfitting to the data.

I will need to do more research into regularisation techniques to help mitigate overfitting, and to help improve the performance of the model. To avoid this in the future, I will use more advanced techniques such as hyperparameter tuning and regularisation.

L2 and L1 regularisation techniques work by adding a penalty to the weights of the model, which helps to reduce overfitting. Hyperparameter tuning is a process of optimising the hyperparameters of a model, such as learning rate, number of layers, etc. This helps to improve the accuracy of the model and reduce overfitting.

8.0.3 MobileNetV3 Small

MobilenetV3 Small is the smallest model in size and is consistently the least accurate model and has the highest loss.
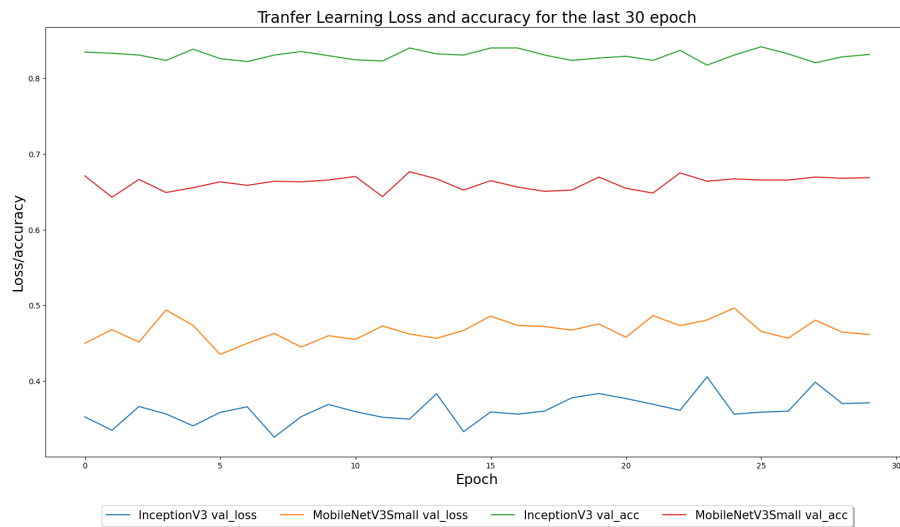

Figure 8.3: The last 30 epochs of the training results

Here's another graph that clearly shows the difference in performance between Inceptionv3 and MobilenetV3 Small. Although InceptionV3 is slightly over 2x the size of MobilenetV3 Small, the difference in performance is clearly worth the trade-off in size. The lack of performance appears to mostly be down to the fact that MobilenetV3 Small is a smaller model, and it is not able to learn as well as InceptionV3. MobilenetV2 and V3 Large are mostly in line with the other models in terms of performance, however they are still not as good as InceptionV3.

8.0.4   The model sizes

Here's a bar chart comparing the sizes of the models in megabytes. They are uncompressed and include the full model, after transfer learning.
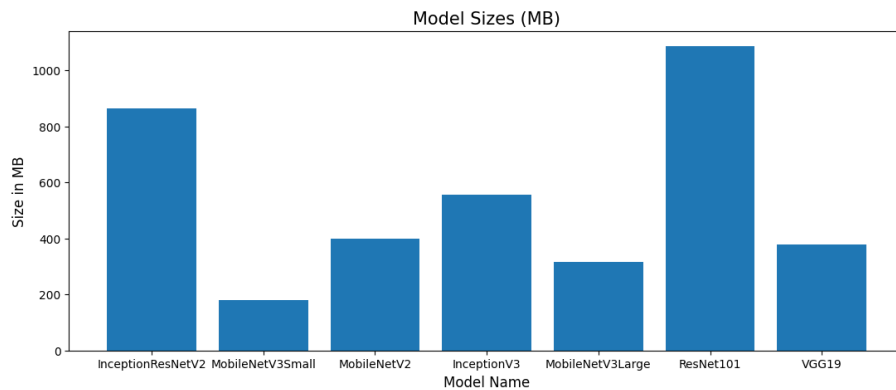

Figure 8.4: The different model sizes in MB

From this graph, it's clear to see how large some of the models are in comparison to each other. The largest model, ResNet101 is just over 1 GB in size and has a far worse accuracy than the top performing model, InceptionV3.
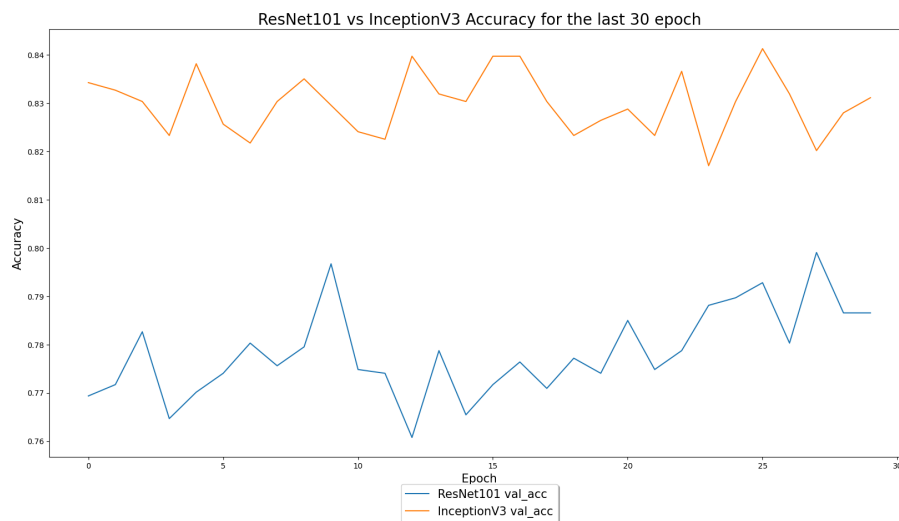

Figure 8.5: Here's it's performance in comparison to InceptionV3

The ResNet101[7] model was created by Microsoft and is a 101 layer deep neural network. It was created to compete in the ImageNet Large Scale Visual Recognition Challenge[9] in 2015. Inceptionv3 is a 48 layer deep neural network, and was created by Google to compete in the same challenge in 2015.

Although the ResNet model managed to win the challenge, it is clear to see that Inceptionv3 is a far better model for this problem. Both models were trained on the same dataset (ImageNet), however the ImageNet dataset is very different to the Alzheimer's dataset. The ImageNet dataset contains a wide variety of images, whereas the Alzheimer's dataset contains a very small number of images, and they are all MRI scans, with only 4 classes. This could suggest that the ResNet model performs better on a larger dataset, however it is not as good at classification with a smaller dataset and fewer classes.

Chapter 9: **The new training results**

Chapter 10: **How to run the code**

10.1 Prerequisites

Go to the directory in your terminal (with python 3.7 ¿ installed) and run the following command:

```
pip install −r requirements.txt
```

10.2 Running the code

To run the code in the notebook, follow the markdown instructions in the notebook. To run the training program, run automated-testing-training/main.py (it will likely take a long time to run).

To run the tests for the AutomatedTestingLibary run the following command:

```
python −m pytest
```

Run main.py in the AutomatedTestingLibary directory to run the training program.

The flower training notebook does not work as I am having issues with Keras and the dataset (specifically the way it is loaded).

Chapter 11: **Diary**

19/10/2022

Today I have been deciding the specific classes of images I want to use for my project. I had initially intended on using only 1 class, however I have decided to use 2 classes instead as I think it will be more interesting to see the results of the transfer learning on 2 classes rather than 1, It will also demonstrate better the different strengths and weaknesses of the individual transfer learning methods.

I have decided to use the following sets of images:

- Flowers Dataset This dataset is likely to be an easier dataset for the models to get high accuracy on, as the different flowers are quite distinct from each other.

- Alzheimer's Dataset This dataset is going to be more difficult for the models as the changes are more subtle between the different scan images, and the images are not as distinct from each other.

I am going to follow the tutorial here which should help me to get a good understanding of how to use the datasets and how to train the models to start with. I have no prior experience training models so this will be a good starting point for me. I will then need to use parts of the code from the tutorial to transfer run the transfer learning.

26/10/2022

Today I have been working on the transfer learning tutorial, I've made more progress through it and I've managed to get Mobilenetv3 to run on the flowers dataset, however I'm having some issues with the alzheimer's dataset. This is significant progress as this was a major blocker for me as I was not sure how to go forward with the prediction.

01/11/2022

Today I've worked more on the transfer learning tutorial and it can now identify the different flowers in the flowers dataset, with varying degrees of accuracy, around 80% was the best I got however this was only after 5 epochs and was the first time I have tried transfer learning.

09/11/2022

I had lost some of my git history for my diary, so I have had to rewrite some diary entries. I have found more flower dataset images here and I've decided to change over to this as soon as possible; https://www.kaggle.com/competitions/tpu-getting-started/overview

14/11/2022

I am training the model "Mobilenetv3 Small" with the Alzheimer's dataset, I have been able to get it to run and it is currently training. The power required to train the models is making it difficult to use trial and error for the parameters, so I am going to try and find a way to use the GPU on my laptop to train the models as soon as possible. I'm also going to optimise the way I am training the models as to not use more power than necessary. This means I've got to get a better understanding of the models I am training with and how they work. I have found this paper Searching For MobileNetV3[8] which I think will be useful for me to read through and understand the structure of Mobilenetv3.

15/11/2022

I've had trouble getting the Alzheimer's classification to work with transfer learning, so I decided to try and run other people's code to see if I could get it to work. I have found that my transfer learning could be significantly more efficient and accurate if I use the existing code, so I'll have to try and identify where I went wrong with my code and make improvements.

21/11/2022

Today I managed to get the accuracy up to 90% on the alzheimer's dataset, this is a significant improvement from the 60% I was getting before. My model appears to still be overfitting however this progress is good. This was done through transfer learning on the Mobilenetv3 small model.

Output from the model during fine-tuning:

```
loss: 0.0314 − accuracy: 0.9883 − val_loss: 0.1337 − val_accuracy: 0.9036
```

22/11/2022

Here's some graphs of the model's predictions on the alzheimer's dataset: This graph was made using the MobilenetV3Small model and transfer learning on the Alzheimer's dataset, with all but 3 layers of the mobilenetv3 model frozen. ![image](/images/better-output.png)

Here is a graph of where I was fine-tuning the model however I had overfitted the model and it was not generalising well to the test data:

![image](./images/maxing-out.png)

I have also setup my laptop properly using Manjaro Linux to train the models on the GPU, this should make it much easier to train the models and to try different parameters and should allow me to create a confusion matrix for different parameters.
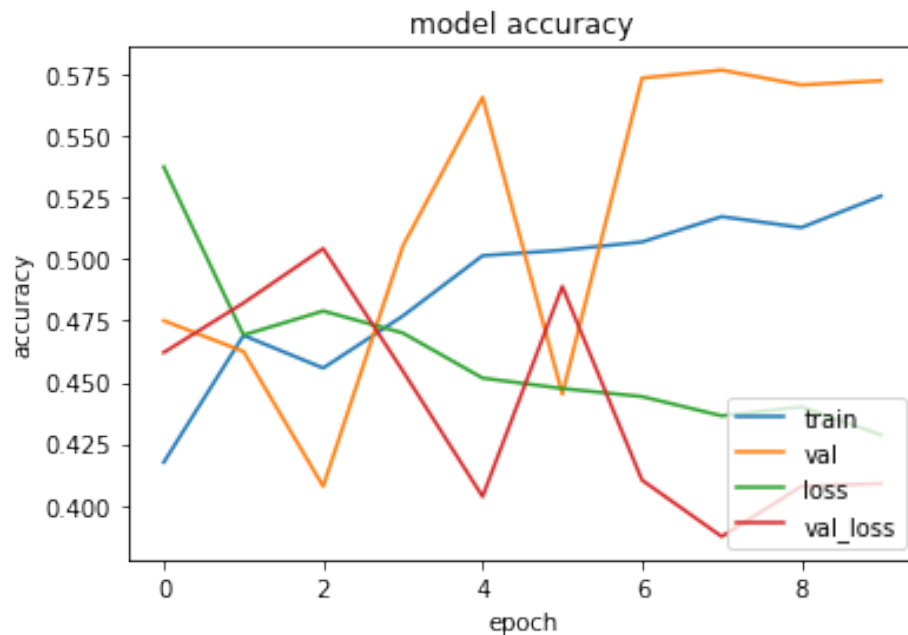
Figure 11.1: Graph of the model's predictions on the alzheimer's dataset

23/11/2022

I have achieved 99% accuracy on my model identifying Alzheimer's (validation dataset). This was unfortunatley not saved however as it crashed, however I believe I can get it to that level again. I have written a scritpt to use different base models for transfer learning and to output the results of training to a JSON file so I can analyse the results later, it also saves the model to a file so I can use it later. This should make it a lot easier to use different models, datasets and parameters to train the models and to analyse the results and make the confusion matrix.

27/11/2022

I have been working lots on the interim report and getting it ready to submit, during this time I have also been setting my laptop off to train the models as I work. I discovered that my '99%' accuracy was from the model learning the validation set through the augmented images, so I have had to retrain all the models with images I have augmented myself.

Soon I'll finish the Alzheimer's dataset training and I'll start working on the flowers dataset.

28/11/2022

Today I have had trouble with the training, my GPU kept running out of memory due to me not releasing the ram after the keras model was trained. This meant I had to keep on checking and restarting the training, every few models. This is now fixed and I have been able to train the models without any issues. They are currently training and there is a lot of progress to be made.

```python
MobileNetV3Small = keras.applications.MobileNetV3Large(
    input_shape=(224, 224, 3),
    include_top=False,
    weights='imagenet')

# Create the model

model = keras.Sequential()

for layer in MobileNetV3Small.layers:
    layer.trainable = False

model.add(MobileNetV3Small)
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(1024, activation='relu'))
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.Dense(4, activation='sigmoid'))
```

✓ 1.9s

```python
# Compile the model

model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    metrics=['accuracy'],
    loss='binary_crossentropy',
)

# Train the model
history = model.fit(
    train_ds,
    epochs=10,
    validation_data=test_ds,
)
```

✓ 11m 32.7s

Figure 11.2: Code used to generate the graph

29/11/2022

Today I have decided to start work on the training and testing library which I can use to speed up training and increase reliability of my training, through unit tests and consistency. This library will allow me to adjust parameters for fitting and compilation to let me easily do hyperparameter tuning.
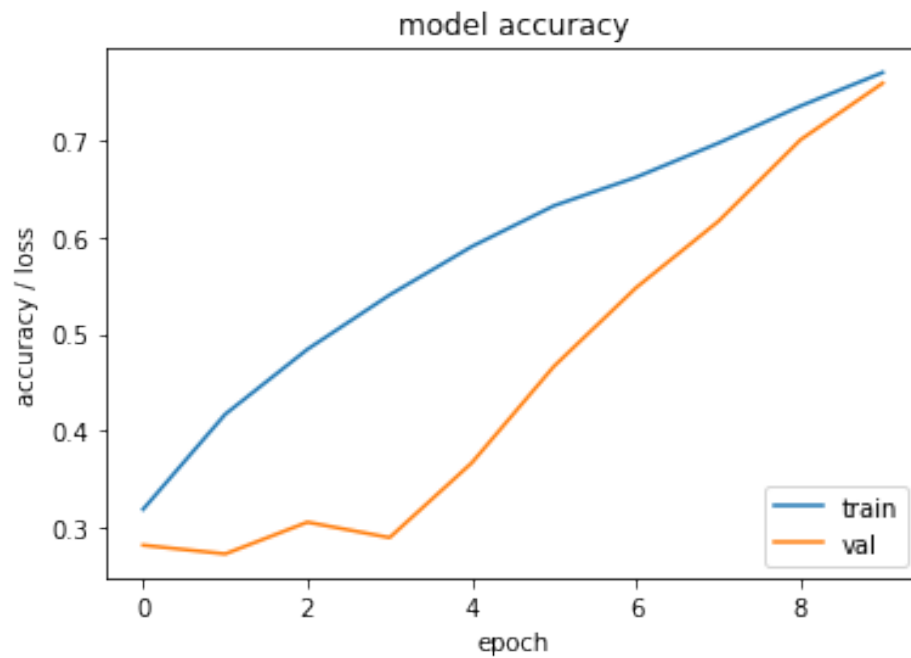
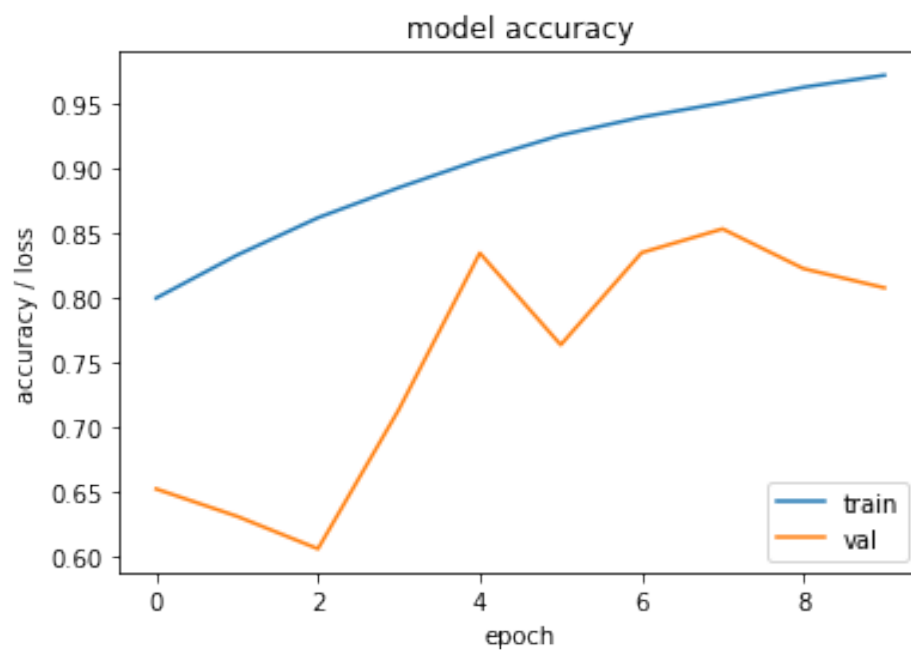Figure 11.3: Positive progress from unfreezing all the layers of the model and "fine-tuning" the model

Figure 11.4: Graph of where I was fine-tuning the model however I had overfitted the model and it was not generalising well to the test data

1/12/2022

I have been working lots on my report and the training library, this has taken a while to get all the information I feel I need in my report, however I have now mostly finished it and will submit it soon.

I have made lots of graphs from the history of the models in training, this made it a lot easier
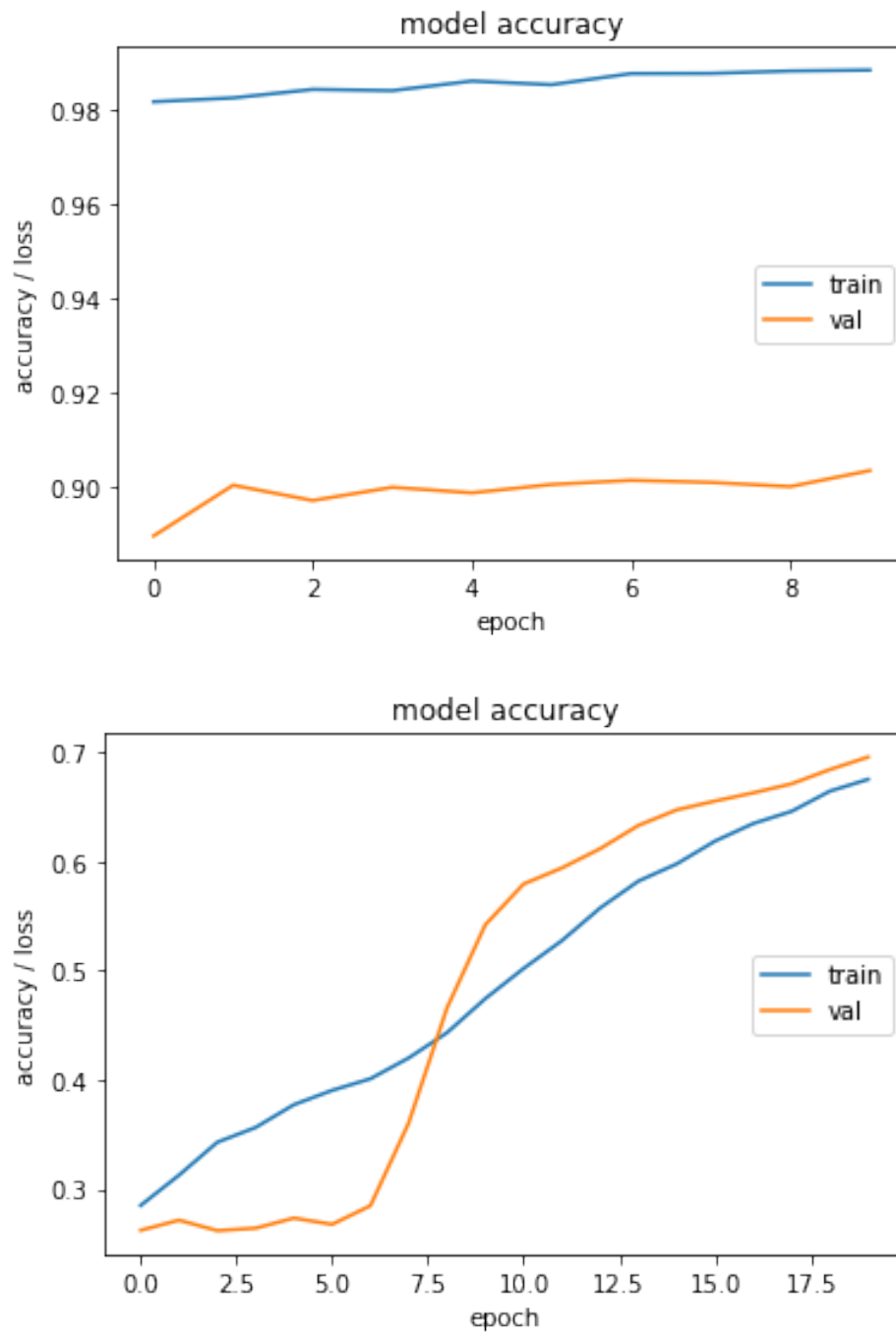
Figure 11.5: First attempt at training the mobilenetV3Large model after unfreezing all the layers

to talk about the different strengths and weaknesses of the different base models.

Chapter 12: **Conclusion**

Bibliography

[1] Mayank Agarwal and Javed Mostafa. "Content-based image retrieval for Alzheimer's disease detection". In: *2011 9th International Workshop on Content-Based Multimedia Indexing (CBMI)*. 2011, pp. 13–18. DOI: 10.1109/CBMI.2011.5972513.

[2] *Alzheimer's Dataset*. URL: https://www.kaggle.com/datasets/uraninjo/augmented-alzheimer-mri-dataset.

[3] *CommitLint*. URL: https://github.com/conventional-changelog/commitlint.

[4] *Confusion Matrix*. URL: https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826.

[5] *Content-Based Image Retrieval Diagram*. URL: https://en.wikipedia.org/wiki/Content-based_image_retrieval.

[6] *CUDA Toolkit Documentation*. URL: https://docs.nvidia.com/cuda/.

[7] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[8] Andrew Howard et al. "Searching for MobileNetV3". In: *CoRR* abs/1905.02244 (2019). arXiv: 1905.02244. URL: http://arxiv.org/abs/1905.02244.

[9] *ILSVRC*. URL: http://www.image-net.org/challenges/LSVRC/.

[10] *ImageNet*. URL: http://www.image-net.org/.

[11] *JupyterLab is a web-based interactive development environment for Jupyter notebooks, code, and data*. URL: https://jupyter.org/.

[12] *Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano*. URL: https://keras.io/.

[13] *Manjaro Linux*. URL: https://manjaro.org/.

[14] *Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python*. URL: https://matplotlib.org/.

[15] *Neural Network Diagram*. URL: https://www.tibco.com/reference-center/what-is-a-neural-network.

[16] *NVIDIA GeForce GTX 1070*. URL: https://www.nvidia.com/en-gb/geforce/products/10series/geforce-gtx-1070/.

[17] *OASIS*. URL: https://oasis-brains.org/#data.

[18] *Oxford Flowers 102*. URL: https://www.robots.ox.ac.uk/~vgg/data/flowers/102/.

[19] *PyTest*. URL: https://docs.pytest.org/en/stable/.

[20] *Python*. URL: https://www.python.org/.

[21] *RHUL GitLab*. URL: https://gitlab.cim.rhul.ac.uk/.

[22] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations*. 2015.

[23] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *CoRR* abs/1512.00567 (2015). arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567.

[24] *tmux*. URL: https://github.com/tmux/tmux/wiki.

[25]   Yudthaphon Vichianin et al. "Accuracy of Support-Vector Machines for Diagnosis of Alzheimer's Disease, Using Volume of Brain Obtained by Structural MRI at Siriraj Hospital". In: *Frontiers in Neurology* 12 (2021). ISSN: 1664-2295. DOI: 10.3389/fneur.2021.640696. URL: https://www.frontiersin.org/articles/10.3389/fneur.2021.640696.

[26]   *VsCode*. URL: https://en.wikipedia.org/wiki/Visual_Studio_Code.