CNN for Classification of Swedish Tree leaves

Javier Rojas García

Javier990507@gmail.com

Abstract –

The aim of this paper is to present the development of a CNN to classify the images in the Swedish Leaf Dataset. The effects of transfer learning and data augmentation will be discussed, as well as the limitations of the model proposed and some difficulties found while developing it, and how to overcome them. The transfer learning based model reached an accuracy of 96.6% on the test set and an accuracy of 54.6% on images from the internet which did not came from a similar distribution.

Keywords: CNN, k-fold cross-validation, transfer learning, data augmentation

1. INTRODUCTION

Plant species recognition is a task that usually requires extensive knowledge on the subject of taxonomy or extensive experience at dealing with plants. Taxonomers can identify plant species by different means, most of the times, by just looking at a leaf, they are able to determine the species from which it came from. They do so by noticing important features in the leaf, such as its shape, color, size, venous system, border or even it's texture.

Convolutional Neural Networks (CNNs) are good at extracting certain features from a set of images and then using those same features to classify images that the network has not previously seen. For this reason, CNNs are extensively used in computer vision systems across many industries: medical, manufacturing, research, transport, security, etc.

Machine learning applications are allowing common people to do tasks that otherwise would require vast amount of knowledge in specific domains. It is in this context that many systems for automatic plant recognition have been developed, enabling its users with the ability to recognize, not only the species of a plant, but also other factors such as diseases, and hydric or nutritional stresses.

The Swedish Leaf Dataset contains leaves from 15 tree classes. It was created by the Computer Vision Laboratory at Linköping University and is publicly available. Each class contains 75 images, which need to be separated in training, validation and testing sets. The images were obtained by digitally scanning the leaves, this results in a great detail of features, like the veinous system of each leaf.

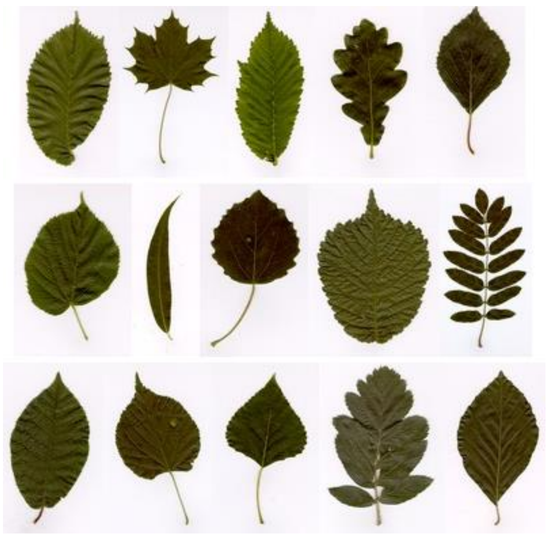


Figure 1: Example instances of each tree class in the Swedish Leaf Dataset.[1]

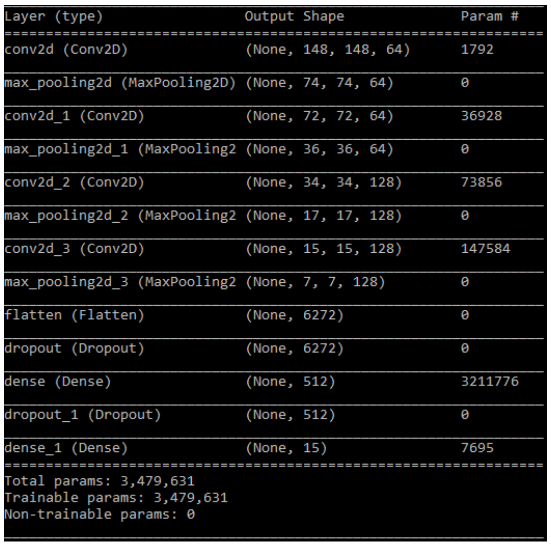
2. DATASET SPLIT

Instead of manually selecting the images that would go into each set, a Python script that implements the dataset split into train, validation and test sets was created. The size of each set can be modified in the script. This was useful to try different split proportions. At the end, due to not having a bigger dataset, a 75/20/5 split was chosen. Each class had 56 examples in the train set, 15 examples in the validation set, and 4 examples in the test set.

3. BASE CNN MODEL

3a.- Architecture

The model's architecture was defined as follows:



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 64)      1792
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dropout (Dropout)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3211776
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 15)                7695
=================================================================
Total params: 3,479,631
Trainable params: 3,479,631
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: Base CNN model architecture

The dropout layer was added to avoid overfitting of the data, which often occurs whit smaller datasets.

3b.- Data Augmentation

Once the training and validation directories were created, training and validation instances of image data generators were created to perform data augmentation in the training set, as well as rescaling the inputs to normalize both sets.

The data augmentation parameters used were:

- Rotation: 90 degrees
- Width & height shifts: 20%
- Shear: 20%
- Zoom: 20%
- Horizontal & vertical flips: True
- Fill mode: 'nearest'

In order to account for the different shapes and orientations of images that might be input into the network.

3c.- Optimizer & Batch Size

The chosen optimizer was RMSprop, and the learning rate used was 0.001, the default value of the Keras optimizer.

The batch size for the training and validation data was of 5 examples. In order to use all of the data in each epoch, the steps_per_epoch and validation_steps parameters, in the fit method, were set to 9, and 3, respectively.

3d.- Loss vs. Accuracy

In Figure 3, the training and validation accuracies and losses were plotted for each of the 250 epochs. Although there are many spikes in each of the plots, both accuracies and losses tend to increase and decrease, respectively. This indicates that the model is not overfitting and is learning correctly.

The spikes indicate that the local minima computed for a certain epoch and batch, is very different from the next one, thus the pronounced increase in either the training or validation losses. The curves would be smoother if a bigger dataset were to be used, so that more examples could be fitted into each of the train and validation batches. [2]

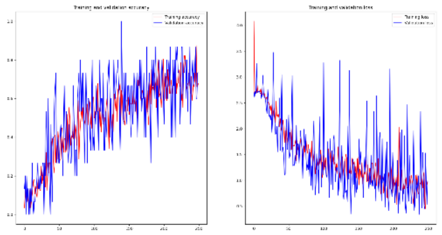


Figure 3: Training and Validation: Accuracy (left), Loss (right). See Appendix A.

3e. Test Accuracy

With this model, the test accuracy on the test set, which contains images that have not been previously seen by the model, reaches 73.3%. Which is already better than a random guess of $\frac{1}{15} = 0.0\overline{6}\%$. Keep in mind, that these results are achieved with images that came from the same distribution of where the training images came.

With transfer learning, better results can be achieved, as shown in the next section.

4. TRANSFER LEARNING CNN MODEL

Transfer learning takes advantage of previously trained, usually more complex and computationally expensive, models. Well trained CNNs extract some features from the data that has been fed to them during training. These features can be reused by creating a backbone model that will receive the previously trained model's weights; removing the top layers of it and locking the remaining ones; and inserting new layers with random weights on top of this model.

The only layers that will be trained are those that are not locked, i.e., the ones that we inserted on top.

The pre-trained model chosen for transfer learning was the Inception v3 model proposed by Szegedy et al. in 2015 [3]. The weights were obtained from [4]. This model was trained with the ImageNet dataset, which contains over 1 million training examples and 1000 classes. [5]

4a. Architecture

The Inception v3's output layer, chosen as input for the new model, was the 'Mixed7' layer. Although, it could be changed to other layers to perhaps obtain better results, depending on the application [6]. If the input images for the new application are not as related to the input images used to train the Inception v3 model, the best would be to go even further and take previous layers in order to stack on top some convolutional layers to detect features specific to the new input data. Finally, over those convolutional layers, one should insert some fully connected, or 'Dense' layers.

On top of the 'Mixed7' layer, a fully connected layer with 1024 units and a 'relu' activation function was added. Followed by a Dropout layer to prevent the network from overfitting.

The last layer of the model contained 15 units and had a 'Softmax' activation function, needed to classify each prediction into the 15 different classes of the dataset. (see Appendix E to see the summary of the last layers)

4b. Data Augmentation

The same parameters used for the previous model where implemented in this model.

The same train and validation set were used to train this model.

4c. Optimizer & Batch Size

The optimizer again was RMSprop, with the default learning rate of 0.001. This learning rate proved to give the similar results when compared with a learning rate of 0.0001, but converged faster.

4d. Loss vs. Accuracy

In figure 4 the training and validation accuracies and losses over 250 epochs were plotted. Again the spikes are there, for the same reasons, but the overall trend is that the training and validation accuracies go up, while their losses go down. This implies there is no overfitting and that the algorithm is learning correctly.

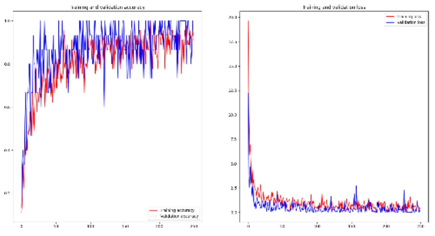


Figure 4: Training and validation: Accuracy (left), Loss (right). See Appendix B

4e. Test Accuracy

The same test set was used for testing and comparing this model's performance. The accuracy on the test set improved considerably to 95% of the images correctly classified after being trained on 250 epochs.

However, the same model trained again, but now with a callback telling it to stop after reaching over 98% accuracy on the training set, managed to achieve 96.6% of accuracy on the test set after being trained for nearly 240 epochs, with similar training and validation accuracies and losses and no overfitting. (See Appendix C)

5. FINAL MODEL

Once the model was validated, a final model was trained using the 95% of the data – 71 images – for training and just the remaining 5% for validation, which could've been omitted since the model was already validated. As expected, since we had slightly more training data, the spikes on the training accuracy and loss curves were not as abrupt as in the previous trainings. However, since the validation data was significantly less, and could basically just oscillate between 4 values, the spikes are bigger and more frequent.

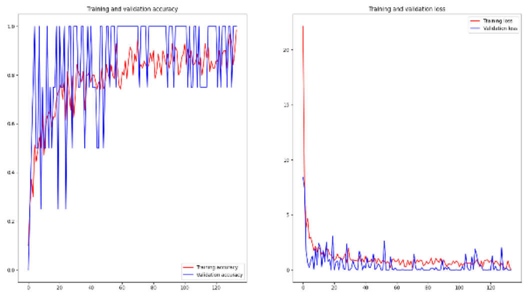


Figure 5: Final model's training and validation: Accuracy (left), Loss (right). See Appendix D

This model is expected to have a similar to the previous model performane when dataset from the same distribution is fed to it (digitally scanned images of those classes of leafs).

6. LIMITATIONS

These networks were trained using a small dataset of images containing only 75 images per class. Considering the immense variability that a specific leaf can have due to factors such as: temperature, disease, deformations, incompleteness, perspective, developing stage of the plant, and many more, it is safe to say that these models will not work as reported in their test accuracies. In order to amend these deficiencies, a greater dataset should be gathered, one that reflects this variability in the real world data.

To further prove this point, a set of 75 images (5 images per class) was recollected from the internet. Images were preprocessed (cut) to try to match the data with which the model was trained. Even after this preprocess step, the new test set images were completely different from the training ones. Then the final model was tested with this new set. The results, as expected, were not as great, only obtaining a 54.6% of accuracy (a slight improvement from the model trained with only 56 images, which obtained a 52% accuracy).

It is worth noting that the most difficult classes for the model to predict, were those that resembled considerably in large features such as the form as the leaf. The model learned good representations for species such as Quercus Robus or Acer Platanoides, which have a characteristic form, but failed to discern between different species of the Ulmus family of trees. This indicates that the training data contained features, which helped the model discern between these classes, that are not present in this new test set.

The classes for which the model performed well are: Sorbus Aucuparia, Sorbus Hybrida, Fragus Sylvatica, Acer Platanoides, Quercus Robus, Salix Fragilis and Populus Tremula. These classes have in common a distinctive shape from the other leaves. The best would be to use the model with only these classes, this could increase the accuracy considerably

Moreover, the test accuracies reported may not reflect the exact value of the model accuracy, since no k-fold cross-validation was implemented. For small datasets, this type of cross-validation is recommended in order to get a mean performance of the model as the input data it receives varies. The test set was meant to work as a proxy for this k.fold cross-validation, considering the small number of examples available per class, as well as for the data expected to be received after the model were to be deployed.

7. RECOMMENDATIONS

Two easily surmountable problems were found during the development of the CNN models. The first one was related to the 'fit' method that Keras uses to train the model using the train and validation data generators. When setting up the steps_per_epoch and validation_steps parameters, one must consider the number of examples that each directory contains, and the batch size defined in each data generator. The ideal scenario is that the

$$batch\_size * steps\_per\_epoch = training\_set\_examples$$

and,

$$batch\_size * validation\_steps = validation\_examples$$

If the left hand side product is smaller than the right hand side, some data will not be used at each epoch. If it is greater than the right hand side, there won't be enough training data to perform the training of the model and an error will be issued.

The second problem arose during the testing step. It is known that we want the test set to come from the same distribution that the train and validation set came. During the training portion of the code, I rescaled the train and validation data in the ImageDataGenerator to be 1/255 of the original value in order to normalize them. The problem came because, at first, I omitted this rescaling process for the test set during the testing portion of the code, and I was getting only wrong predictions usually biased towards one unique class.

The solution to this last problem was pretty straightforward: rescale the input images in the testing phase by the same 1/255 factor. Once I did this, the networks started to perform as expected from their training and validation accuracies and losses.

8. CONCLUSIONS

The model trained using transfer learning was by far superior to the one done implementing just some convolution layers followed by fully connected layers. Even though the dataset was small, it managed to perform well without overfitting using data that came from the same distribution of images with which it was trained. It did not generalize as well with all of the images on the internet, whose characteristics are completely different, but it proved to work well with nearly half of the classes. The model worked as expected, and its performance could be enhanced by training it with more data, which actually reflects the variability of leaves in the real world, and by using it only with those classes that have characteristic shape features like the ones listed previously. Other species plant classifiers could be built using this model's architecture.

9. REFERENCES

[1] Linköping University. (2016). Swedish Leaf Dataset. Recovered from: https://www.cvl.isy.liu.se/en/research/datasets/swedish-leaf/

[2] Tunguz, B. (2018). What could be explanations for validation loss zig-zagging when training a deep neural network? Recovered from: https://www.quora.com/What-could-be-explanations-for-validation-loss-zig-zagging-when-training-a-deep-neural-network

[3]Szegedy,C., Vanhoucke,V., Ioffe, S., & Shlens, J. (2015). Rethinking the Inception Architecture for Computer. Recovered from: Rethinking the Inception Architecture for Computer Vision

[4]https://storage.googleapis.com/mledu-datasets/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5

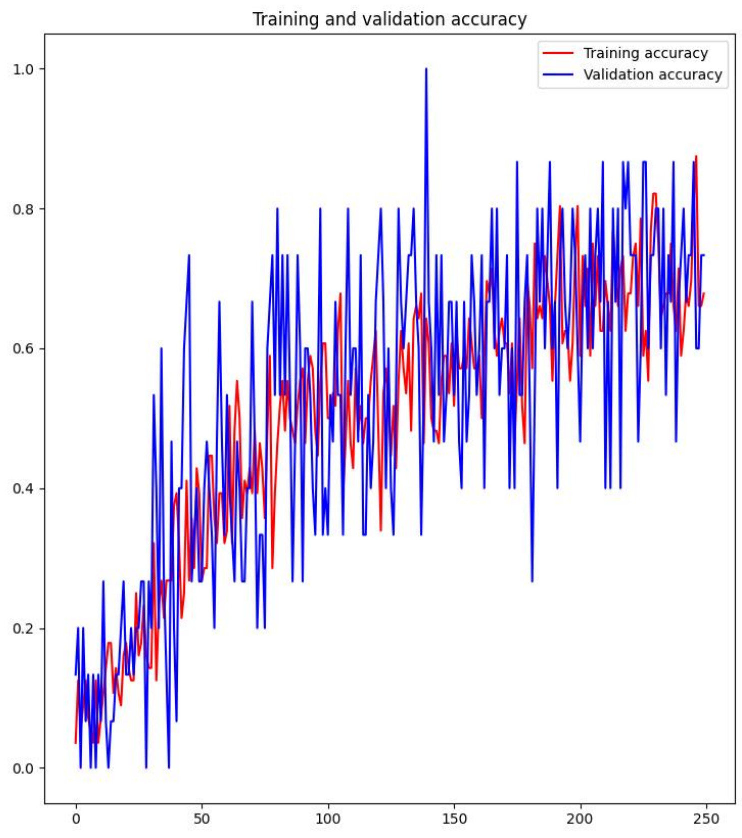
[5]Milton-Barker, A. (2019). Inception V3 Deep Convolutional Architecture for Classifying Actue Myeloid/Lymphoblastic Leukemia. Recovered from: https://software.intel.com/content/www/us/en/develop/articles/inception-v3-deep-convolutional-architecture-for-classifying-acute-myeloidlymphoblastic.html

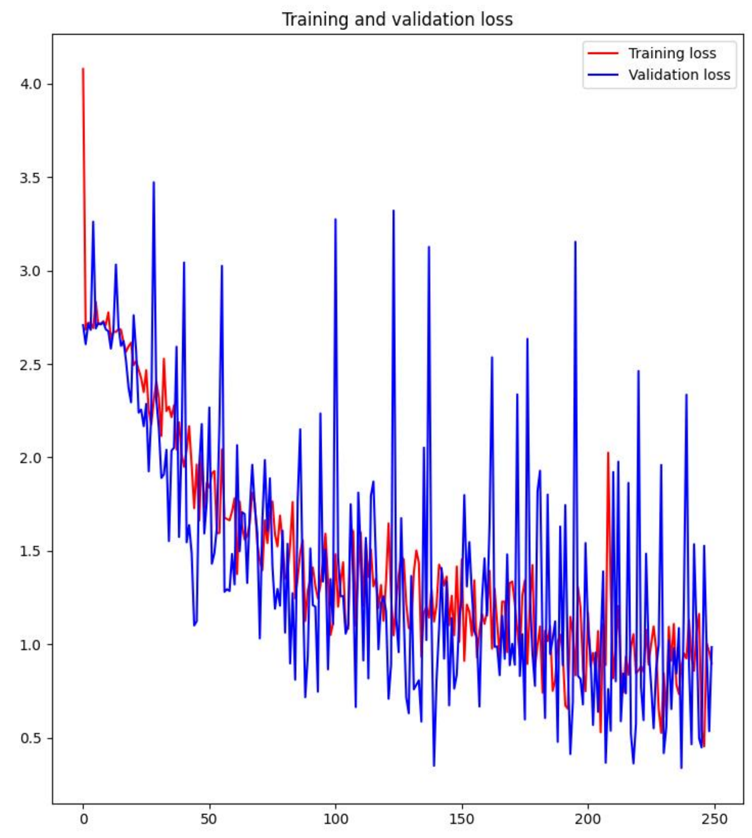
[6]Ng, A. & Moroney, L. (n.d.) Convolutional Neural Networks in Tensorflow. Recovered from: https://www.coursera.org/professional-certificates/tensorflow-in-practice

[7] Tensorflow. (2021). Transfer Learning and Fine-Tuning. Recovered from: https://www.tensorflow.org/tutorials/images/transfer_learning

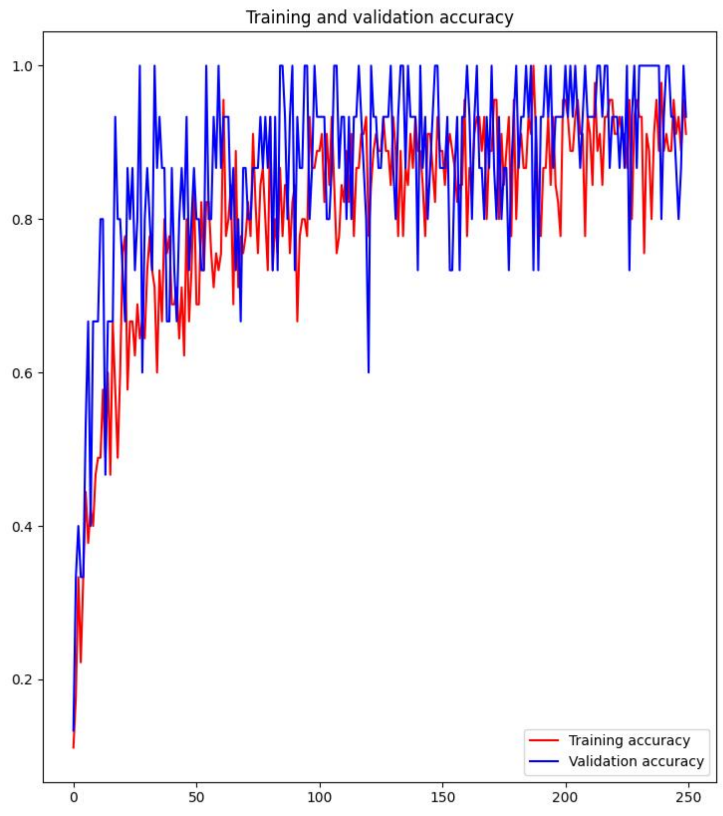
[8]Söderkvist, O. (2001) Computer Vision Classification of Leaves from Swedish Trees. (Thesis). Recovered from: http://www.diva-portal.org/smash/get/diva2:303038/FULLTEXT01.pdf

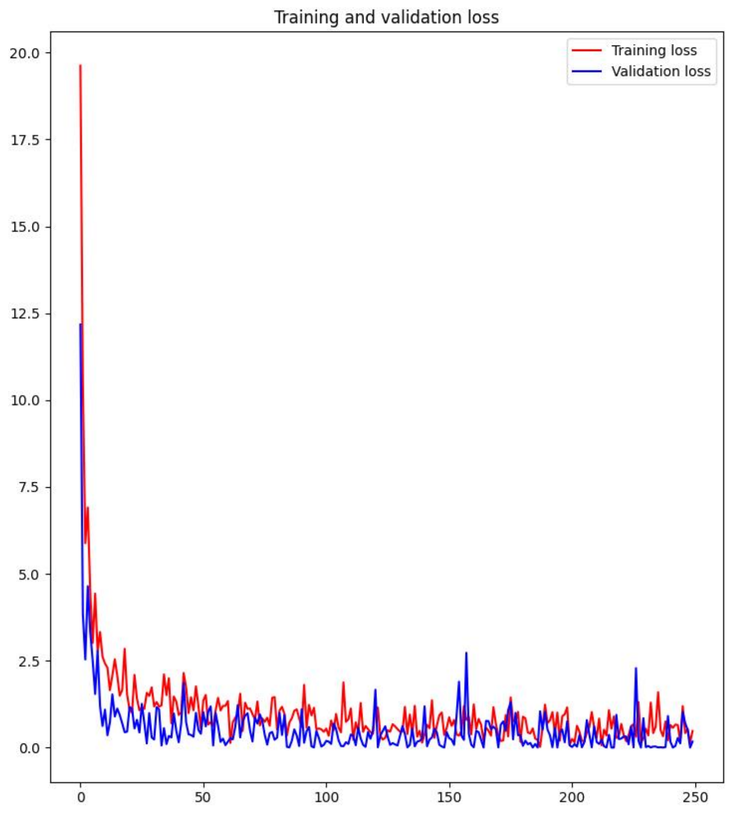
APPENDIX A



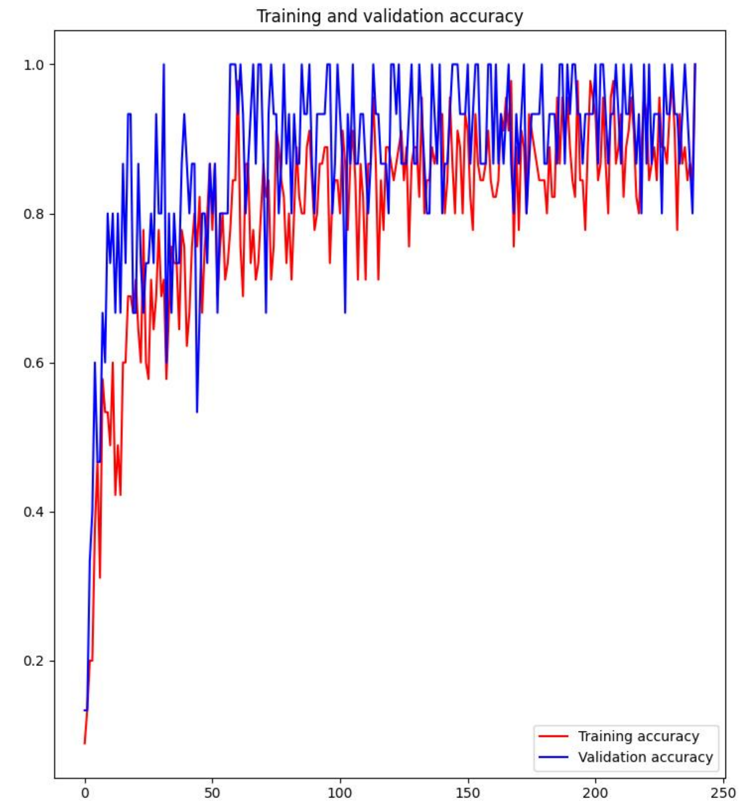


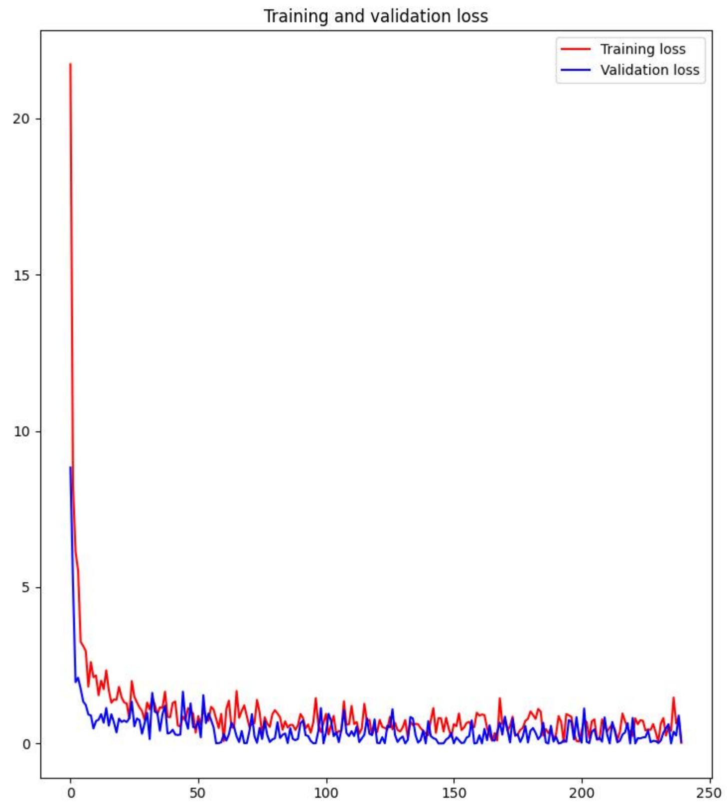
APPENDIX B



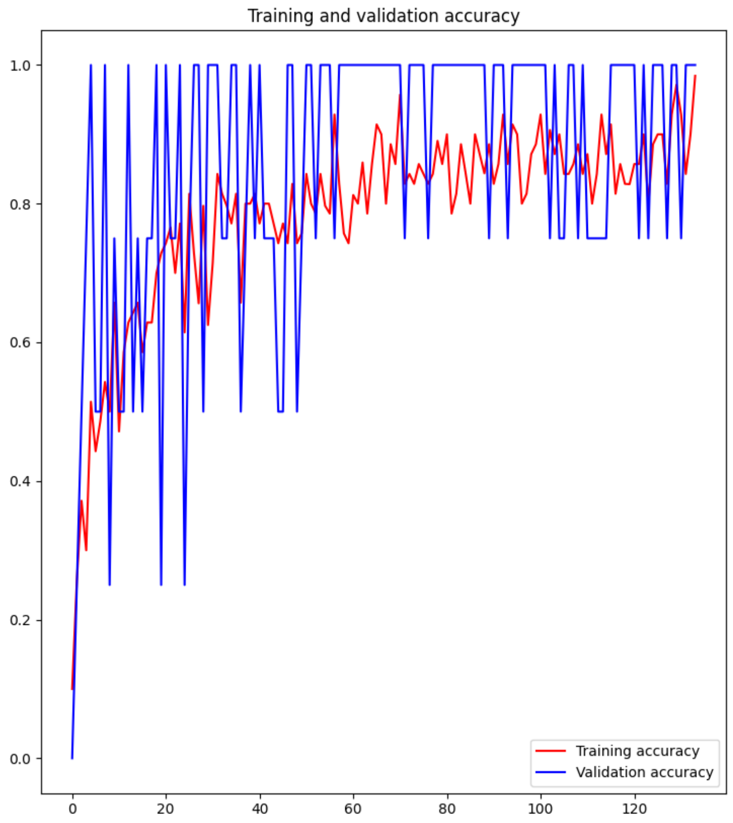


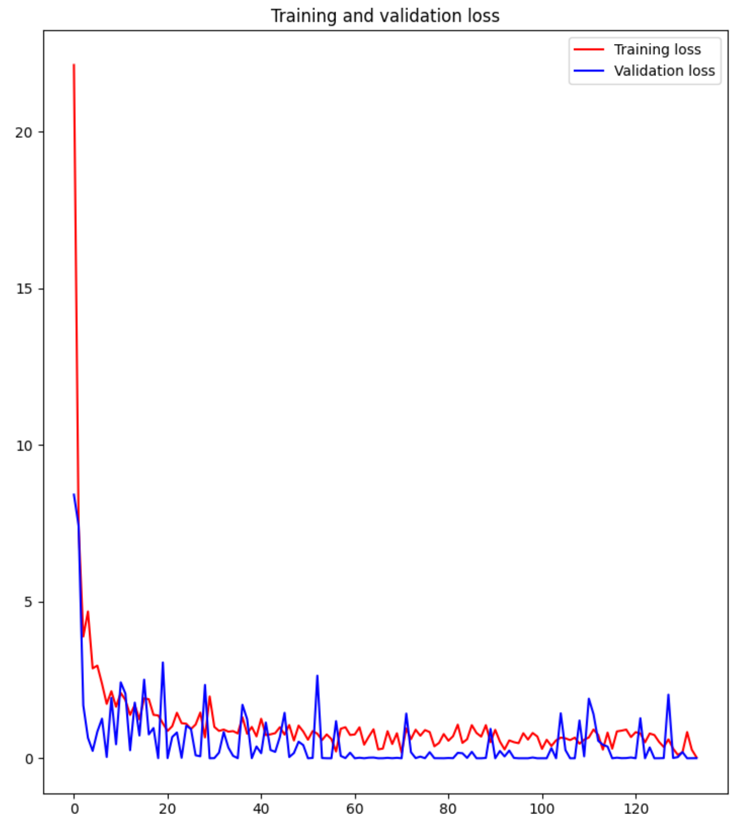
APPENDIX C



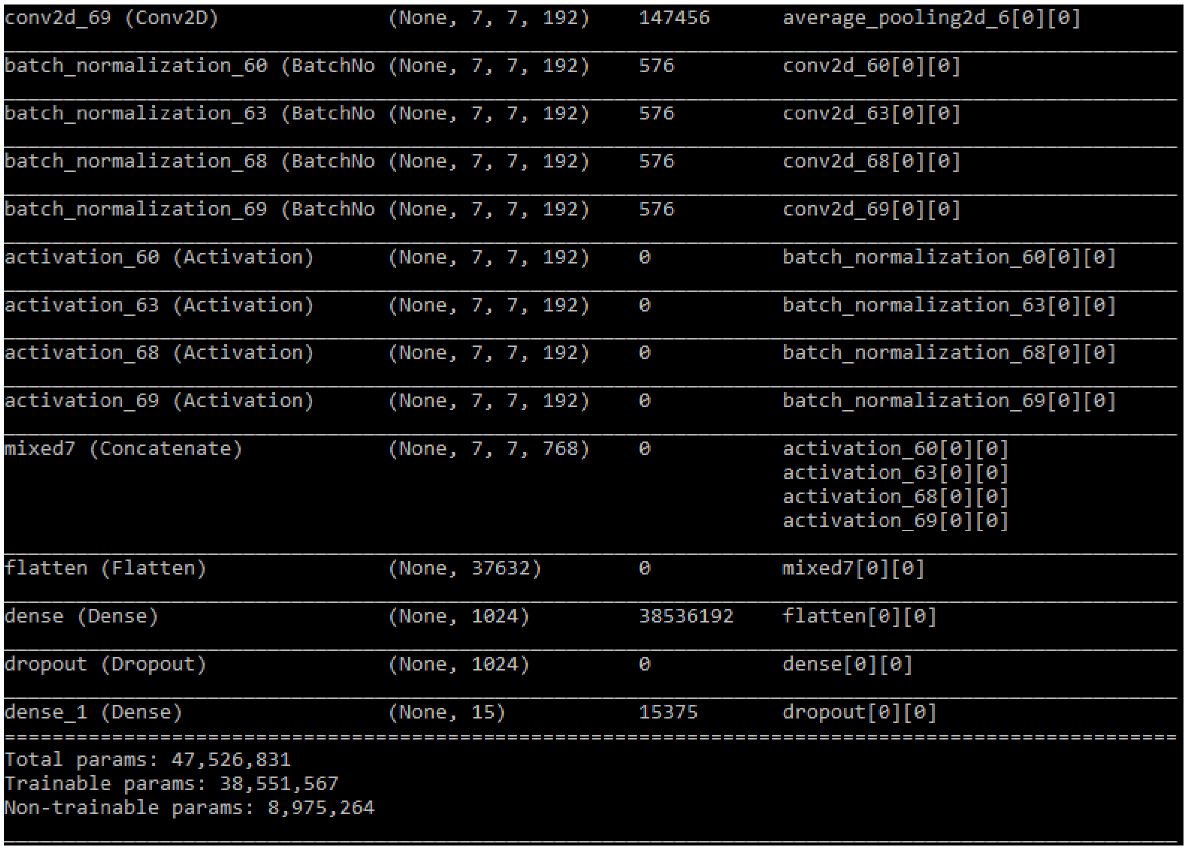


APPENDIX D





APPENDIX E



```
conv2d_69 (Conv2D)              (None, 7, 7, 192)    147456      average_pooling2d_6[0][0]
__________________________________________________________________________________________________
batch_normalization_60 (BatchNo (None, 7, 7, 192)    576         conv2d_60[0][0]
__________________________________________________________________________________________________
batch_normalization_63 (BatchNo (None, 7, 7, 192)    576         conv2d_63[0][0]
__________________________________________________________________________________________________
batch_normalization_68 (BatchNo (None, 7, 7, 192)    576         conv2d_68[0][0]
__________________________________________________________________________________________________
batch_normalization_69 (BatchNo (None, 7, 7, 192)    576         conv2d_69[0][0]
__________________________________________________________________________________________________
activation_60 (Activation)      (None, 7, 7, 192)    0           batch_normalization_60[0][0]
__________________________________________________________________________________________________
activation_63 (Activation)      (None, 7, 7, 192)    0           batch_normalization_63[0][0]
__________________________________________________________________________________________________
activation_68 (Activation)      (None, 7, 7, 192)    0           batch_normalization_68[0][0]
__________________________________________________________________________________________________
activation_69 (Activation)      (None, 7, 7, 192)    0           batch_normalization_69[0][0]
__________________________________________________________________________________________________
mixed7 (Concatenate)            (None, 7, 7, 768)    0           activation_60[0][0]
                                                                 activation_63[0][0]
                                                                 activation_68[0][0]
                                                                 activation_69[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 37632)        0           mixed7[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 1024)         38536192    flatten[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 1024)         0           dense[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 15)           15375       dropout[0][0]
==================================================================================================
Total params: 47,526,831
Trainable params: 38,551,567
Non-trainable params: 8,975,264

__________________________________________________________________________________________________
```

Appendix A: Layers added on top of the 'Mixed7' convolutional layer from the Inception v3 architecture.