Transfer Learning - Flower Classifier

CAB320 - Assignment 2 - Semester 1 - 2020

Jarryd Stringfellow - n9734074
Chase Dart - n10171517

1.0 Introduction

The aim of this assignment is to build a flower classifier using transfer learning on a neural network trained on the ImageNet dataset. A range of flowers including a daisy, dandelion, rose, sunflower, and tulip have been chosen for this task. This task will be undertaken by coding in the Python language. Using the pre-trained dataset provided, the goal is to generate a process to be able to identify the flowers provided. The learning process will be documented in this report to show the steps that have been taken.

Transfer learning is a machine learning method where we reuse a model trained on a first dataset called the source dataset as the starting point for training a model on a second dataset called the target dataset. Generally, the source dataset is a large dataset like MobileNetV2 and the target dataset is a much smaller dataset relevant to a new application domain.

The method of applying transfer learning in through deep learning is as such:

1. A previously trained model will be sourced to remove the initial heavy lifting. This layers from this model will be used.
2. To avoid destroying any information they contain, the weights are frozen.
3. New trainable layers are added above the frozen layers. These are the layers that will learn to turn the old features into predictions on a new dataset.
4. Train the new layers on the provided dataset.

2.0 Methodology

2.1 Dataset

The dataset used contained 1000 images of 5 classes of flowers which comprised five different flower types: daisy, rose, tulip, sunflower, and dandelions. Each class had 200 unique images of a centered and easily identifiable flower related to its class. The resolution of each image was 320x240.

Each of the images are broken into their relative folder for ease of access during the process. For this assignment, the images are in a dataset folder one layer outside the python file. This was to allow the data to be accessible and not require the large quantity of files to be sent to GitHub for collaboration.

2.2 Data Preprocessing & Splitting

The dataset provided required little data preprocessing. The input of the N-MobileNetV2 model required an maximum input of (224, 224, 3) and the images were (320, 240, 3) by default. The images were scaled down to match the input of the model.

The data was split into the training, testing and validation; 60:20:20 split. This was decided because a bigger split for training to (80:10:10) would take too much computational resources when training the data on the N-MobileNetV2 model.

The first dataset is training. This is where the images will be passed through the classifier to help train the new model.

The next category is the validation category. This provides the image with a known identifier for the model to check its accuracy by providing unbiased evaluation of a model fit for the training dataset. It also helps to tune the model hyperparameters.

The final category, the testing dataset, provides an unbiased evaluation of a final model fit on the training dataset.

2.3 MobileNetV2 & Fine Tuning

The MobileNetV2 model is a convolutional neural network architecture that provides accurate image classification on low computational devices like mobile devices (Tsang, 2019). The pretrained model is trained on the ImageNet dataset containing over a million images provided by google. Due to the model's architecture and large training dataset, it is very effective at feature extraction for object detection and segmentation (Sinha, 2018) on a wide variety of subjects.

As seen in fig 1 it is a very large model made to handle extremely large datasets, however, by training the model on the flower dataset, the number of classes are reduced from thousands to only five. To accommodate the change to the model's intended extremely large dataset, fine tuning to the MobileNetV2 architecture is applied for better suitability to the flower dataset. The final dense layer of the model is reduced to an input of 5 which will change the dimensionality of the final output of previous layers within the model. This change to the model should provide a good intended output.

Model	**Size(MB)**	**Top-1 Accuracy**	**Top-5 Accuracy**	**Parameters**	**Depth**	**Time (ms) per inference step (CPU)**	**Time (ms) per inference step (GPU)**
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8

Fig 1. Statistics about MobileNetV2

3.0 Implementation

3.1 Computer Setup

The system setup that is used will affect the outcomes of testing from the runtimes through to the amount of resources required for memory. The device used for all of the testing was kept consistent and no other programs were run simultaneously to the testing.

The device that was used was a Dell laptop running a Windows OS. The device was equipped with 16GB of RAM and used a Intel Core i7 Processor. This device allowed for the tests to be conducted in a suitable environment to test the capabilities of the flower classifier.

3.2 Model Research Process

3.2.1 Initial Model Development

The model was compiled with the SDG3 optimiser that used a learning rate of 0.01, momentum of 0 and a false nesterov. A categorical cross-entropy loss function was added to match the framing of our multi-class classification model. A epoch of 40 was decided after it reached the computational limits.

3.2.2 Initial Development Results

The results of the initial model parameters show clear results of overfitting. It's most likely because the learning rate for the optimiser is too high. The final results show that the model came to a conclusion too early in the epochs. For later models, the learning rate must be lower.

This observation that the learning rate was too high was made evident from the results shown below. The first graph in Fig.2 shows that the distance between the testing and training accuracy is far too wide. A more acceptable reading would result in a higher accuracy between the testing and trained data. The second graph shows that the loss rate between the two is not bad but could be improved.

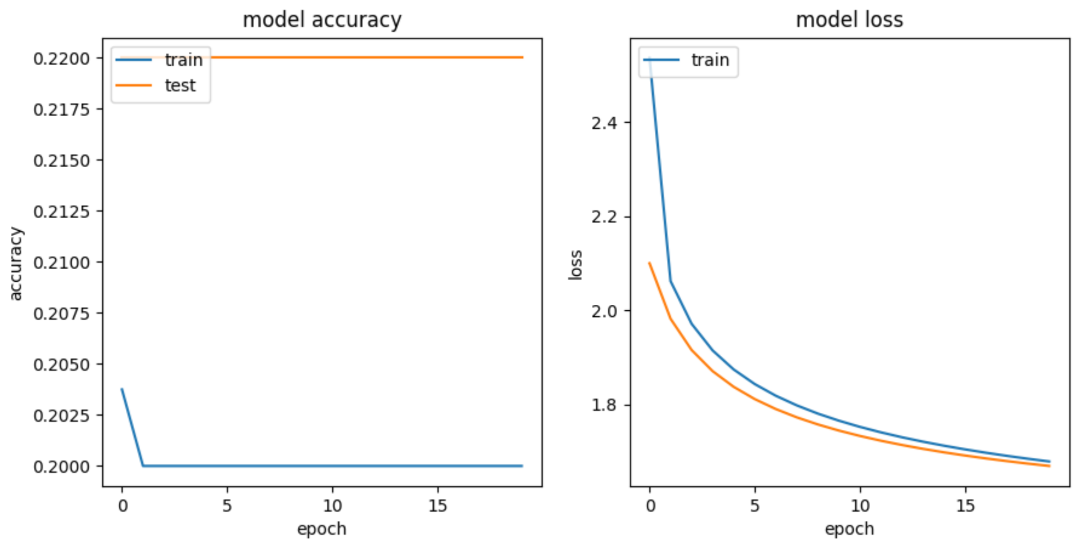

Fig.2 Testing and Training Accuracy and Loss

The combination of both of these results has had the effect of overfitting in the confusion table shown in Fig.3. The provided figure shows an aligned match of 1.0 for the predicted sunflower across the entire range of samples. This has also been found for the testing and validation categories.

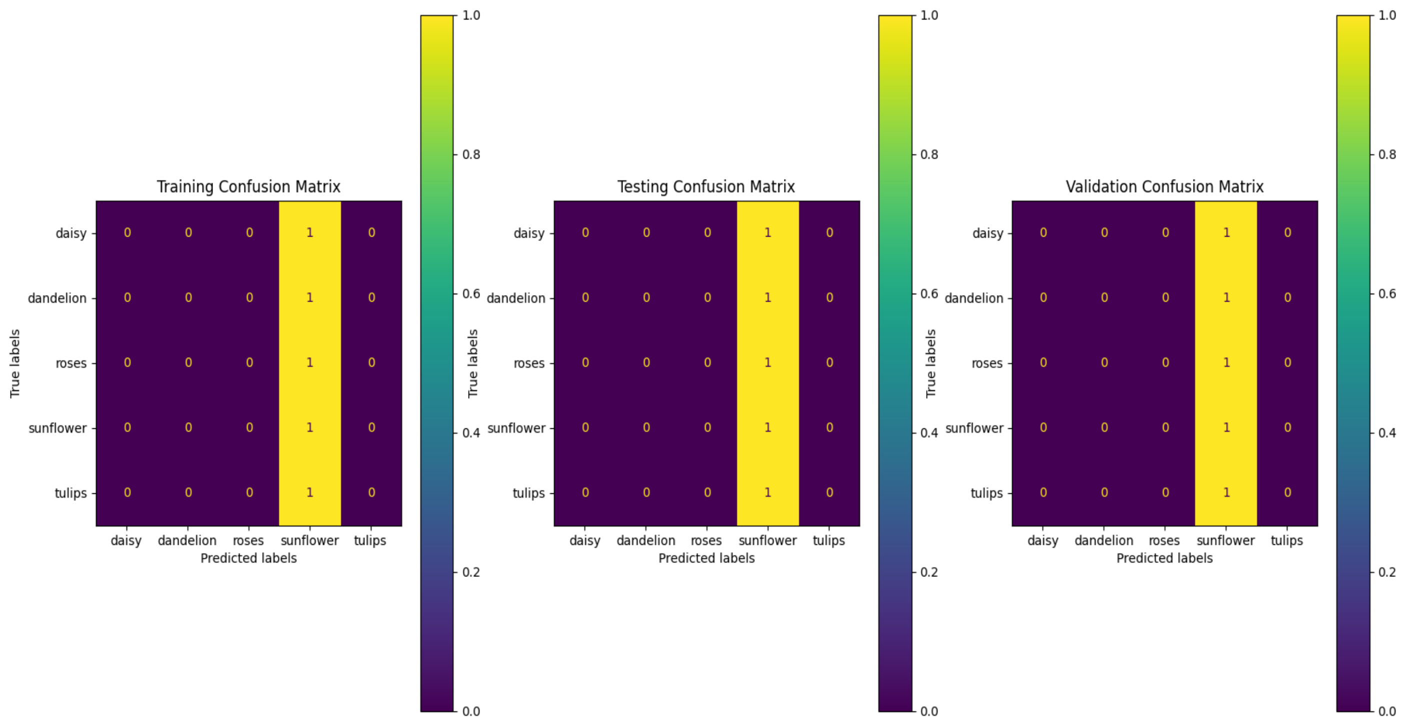

Fig.3 Testing, Training and Validation Confusion table

3.2.3 Refined Model Development

The first refined model has the same parameter as the initial model but with an learning rate of 0.001, fig 4. This was then modified for the second model which has a learning rate of 0.0001, fig 5. The third model has a new learning rate of 0.01 and momentum of 1 fig 6. The learning results were decided upon after the first model's learning rate was too high. The lower learning rates were found to need more computational power which limited the final results. Because of this, the epochs of all refined models were brought down to 10 in order to provide sufficient final results.

3.2.4 Refined Model Results

It's clear that a lower learning rate improved the overfitted accuracy from fig 2. The learning rate of fig 4 shows a tighter accuracy gap between the training and testing data than what both fig 5 and fig 6 have shown results in. There are not enough epochs to understand the full results of the refined models. Bringing down the learning rate and epoch seem to be counter intuitive as the learning rate draws out along the number of epochs. However, decreasing the epochs has made the loss model's result a mystery about whether decreasing the learning rate improved the final results.
It can be said that applying momentum has made the model overfitted. However, much like the learning rate, final results are indefinite.

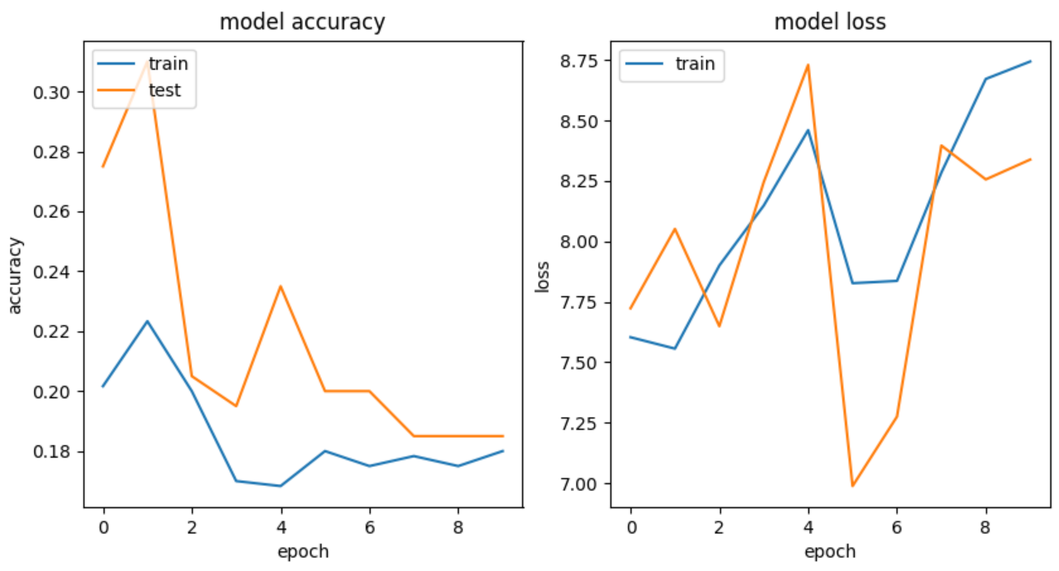

Fig 4. Learning rate = 0.001

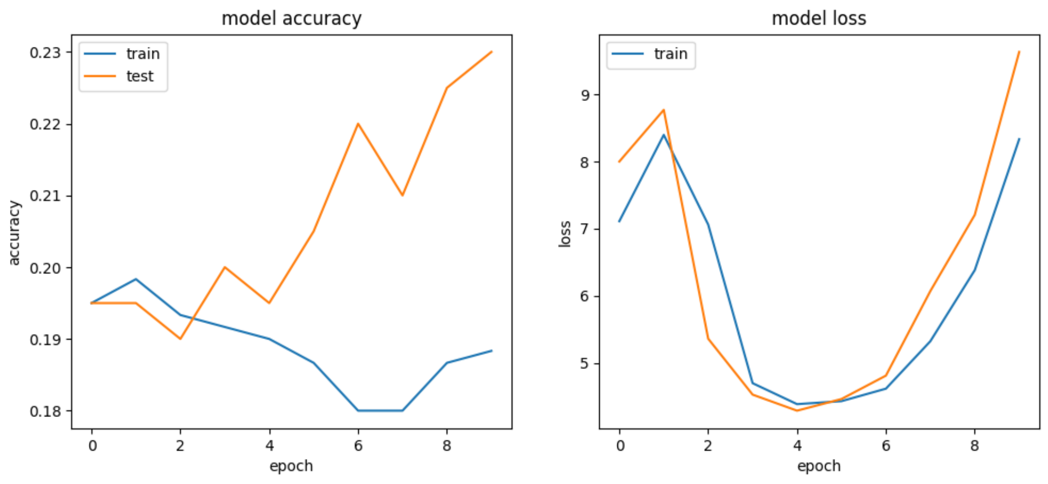

Fig 5. Learning rate = 0.0001

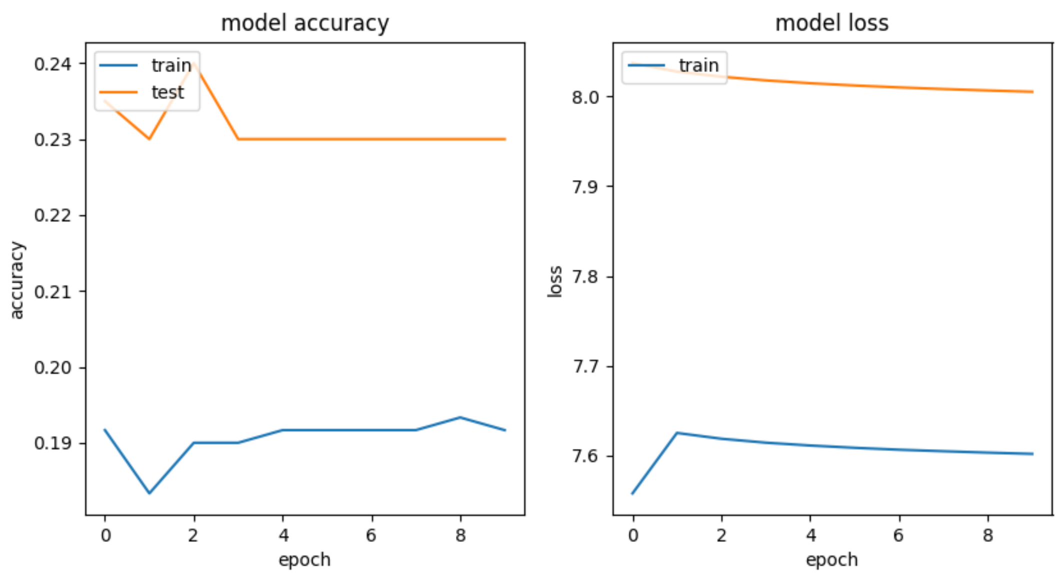

Fig 6. Learning rate = 0.01, momentum = 1

4.0 Conclusion

As can be seen from the results and discussion provided above, there has been an improvement in the model accuracy from the changes that have been made. With original modifications to the model, an overfitting result was established. This was primarily due to a learning rate that was too high. After improving the system by dropping the learning rate and adding momentum to the system, a much better result was found. The issue with this new result was that it required more epochs in the system to be able to understand the results better.

Some of the issues that we came across included the computational power and issues with running the program onsome of the devices. One of the largest issues found occurred when trying to run past 10 epochs. The lack of computational power was the reason behind this as we were running out of RAM on some of the devices. Some models were unable to run past this computational limitation which severely hindered proper analysis of the final results.

Another issue occurred when the program did not return any results. This resulted in the graphs all showing a flat line. This was due to the results not returning correctly but rather resulting in a series of zeroes. This issue was solved by using a different system.