Full Name	Aziz Alexandros Tarek
Student Id	@00571858
Username	AEE826
Module Title	Deep Learning
Module Leader	Dr. Sunil Vadera

Developing a CNN for Image Classification

Contents

1 Introduction

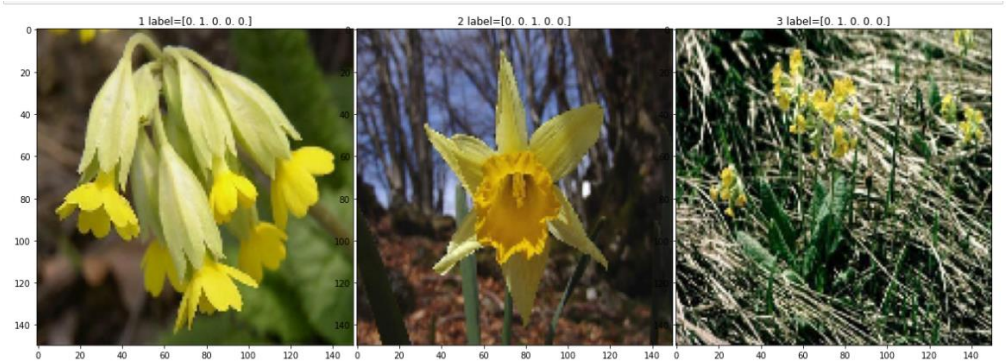
This report provides documentation of the development of two Convolutional Neural Networks a simple and an advanced model. Both models have the purpose of classifying 5 categories of flowers (Buttercup, Daffodil, Sunflower, Cowslip, Windflower) with the highest training and validation accuracy possible as well as the lowest training and validation loss possible. The data consists of 400 images 80 for each category. When the python file is executed and the absolute path of where the images are located is given the data will be automatically split into 250 images for training, 100 for validation and 50 testing. To produce the advanced model which is an improved version of the simple model extensive experimentation has been conducted using various methods such as data augmentation, data visualization, transfer learning, methods to control overfitting [4] : Dropout with values ranging from 0.2-0.5 and Regulariler l1, l2 with a learning rate of 0.001/0.01, and more.

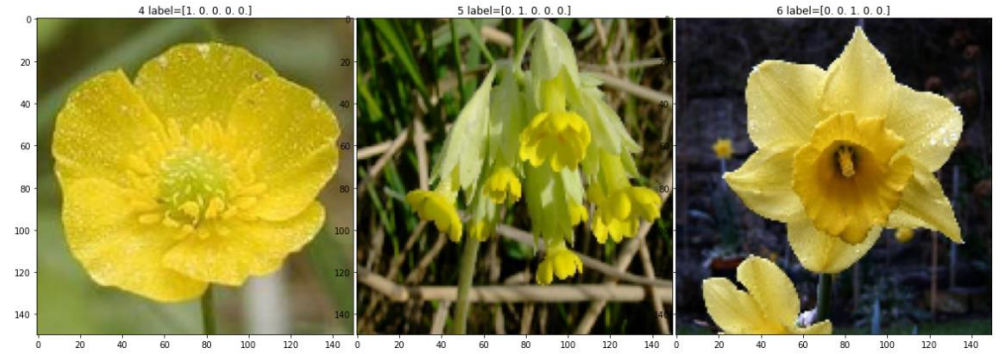
2 Simple and Advanced Model

2.1 Similarities

Both models use the same method that creates 3 folders (test, train and validation) and each of these folders contain 5 folders one for each type of flower. The models also use the ImageDataGenerator class to create generators that read all the images by using the 'flow_from_directory' method. In this method the class mode has been set to categorical rather than binary as the data has multiple classes (types of flowers), the target size property resizes the input images and has been set to 150, the batch size is the number of images that the generator will use per batch, 10 have been set for the training and testing data and 5 for the validation.

Furthermore, after the creation of generators to test them, their shapes are printed out and they are plotted using matplotlib which displays some images and their labels.





2.2 Simple Model

The simple model uses Conv2D and Maxpooling operations 4 times. Maxpooling is one of the most common pooling methods to gradually shrink the spatial size of the representation in order to minimize the number of parameters and processes in the network. The input shape is 150(height),150(width),3(Colour channels). Finally, 2 dense layers are used with the last layer having a Softmax activation while all the other layers have a ReLu(Rectified Linear activation Unit) activation. As this is a multiclass [1] [2] [3] single label problem Softmax activation was used in the last layer with 5 neurons as there are 5 classes.

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_4 (MaxPooling 2D)	(None, 74, 74, 32)	0
conv2d_5 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_5 (MaxPooling 2D)	(None, 36, 36, 64)	0
conv2d_6 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_6 (MaxPooling 2D)	(None, 17, 17, 128)	0
conv2d_7 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_7 (MaxPooling 2D)	(None, 7, 7, 128)	0
flatten_1 (Flatten)	(None, 6272)	0
dense_2 (Dense)	(None, 512)	3211776
dense_3 (Dense)	(None, 5)	2565

```
from keras import models, layers

network = models.Sequential()
network.add(layers.Conv2D(32,(3,3),activation='relu',input_shape=(150,150,3)))
network.add(layers.MaxPooling2D(2,2))
network.add(layers.Conv2D(64, (3,3), activation="relu"))
network.add(layers.MaxPooling2D(2,2))
network.add(layers.Conv2D(128, (3,3), activation="relu"))
network.add(layers.MaxPooling2D(2,2))
network.add(layers.Conv2D(128, (3,3), activation="relu"))
network.add(layers.MaxPooling2D(2,2))
network.add(layers.Flatten())
network.add(layers.Dense(512,activation="relu"))
network.add(layers.Dense(5,activation='softmax')) #output
```

The Categorical_crossentropy loss function was used as this is a multi-class single problem and the results that were obtained are: (Figure 0)

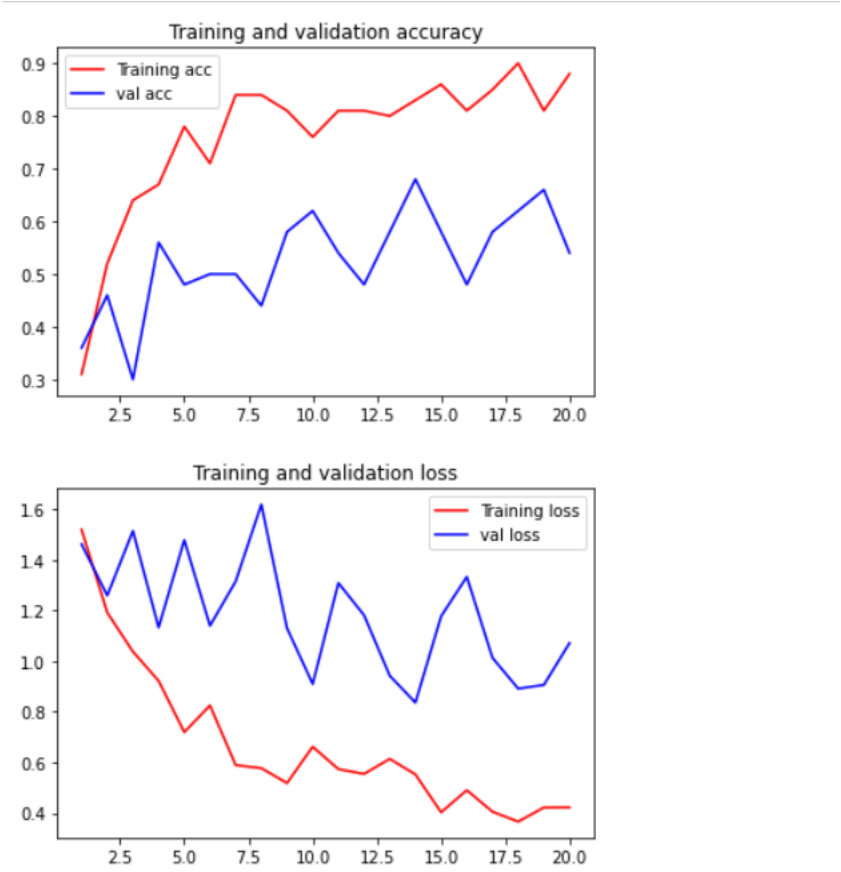


Figure 0

With the use of RMSprop optimisation algorithm and a learning rate of 1e-4 for the duration of 20 epochs with 10 steps per epoch and 10 validation steps the total accuracy on the test set was 46% and the loss 131%. Which is quite low, and the loss is extremely high and allows plenty of room for improvement.

2.3 Advanced Model

2.3.1 Experimentation

In order to improve the accuracy of which is the focus of this assessment various methods have been implemented such as Data Augmentation, Data Visualization, two types of transfer learning using the VGG16 database and more to create the model with the best accuracies and reasonable losses.

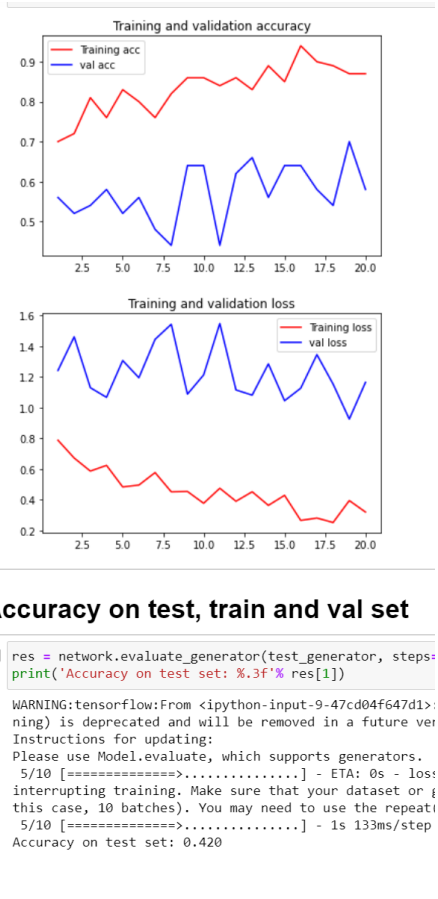


Figure 1

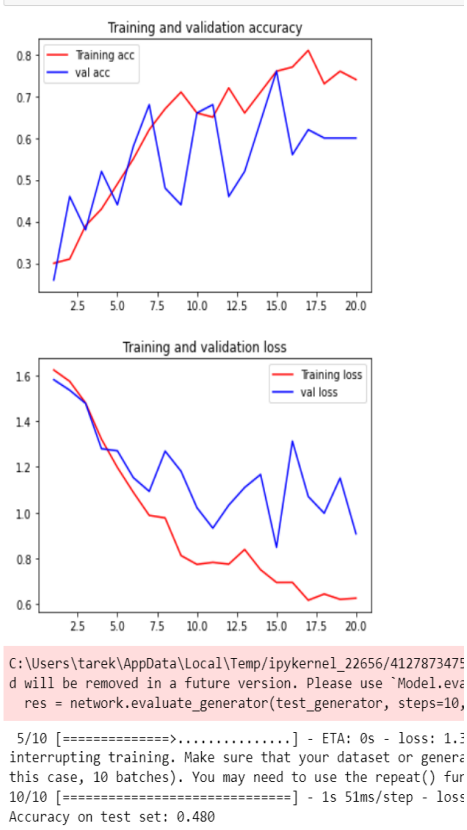


Figure 2

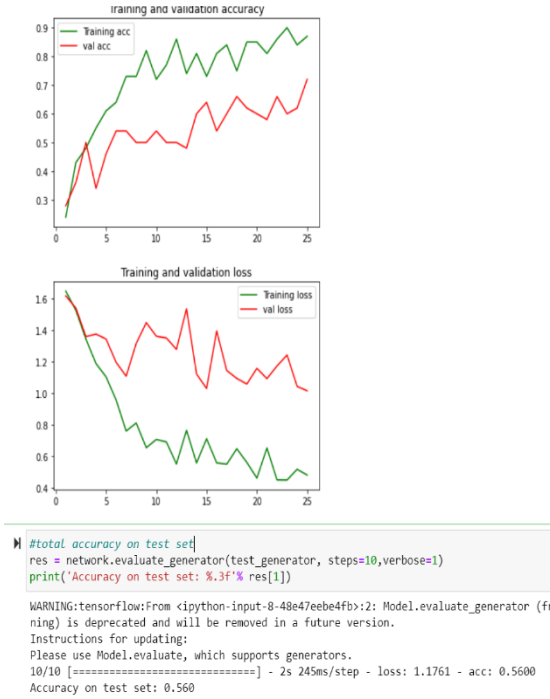


Figure 3

[Model1] With the addition of 3 Dense layers to the simple model and the use of ReLu activations the first 512 nodes, the second with 256 nodes and the third with 128 nodes, before the last layer. After running the model with 10 steps per epoch and 10 validation steps for 20 epochs the network evaluation with 10 steps and a verbose of 1 which enables an animated bar showed that the accuracy did not improve (42%), instead it got slightly worse. (Figure1)

[Model2] Dropout of 0.2 was added between the previously added dense layers to control the overfitting as the validation loss is extremely high. After running the model again without making any changes to the network compilation, fitting or the evaluation method, the results indicate that the validation loss seemed to have dropped slightly and the accuracy improved on the test set with an accuracy of 48%. (Figure2)

[Model3] After undoing the changes that were made above a Dropout of 0.5 was added before the first Dense layer which resulted in a higher

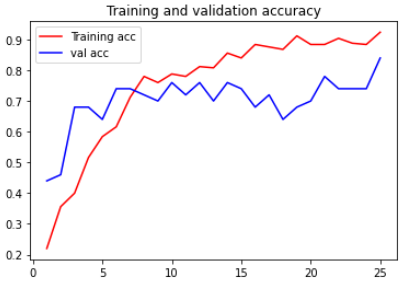
accuracy compared to the 2 models above, the accuracy on the test set was 56%. (Figure)

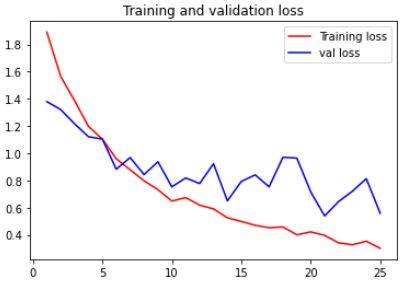
After making other small changes like changing the number of nodes of the dense the layers or using different number of pooling layers and dens layers the results did not seem to improve as much, and so other techniques were introduced to the model to improve the accuracies.

```
network = models.Sequential()
network.add(vgg_base)
network.add(layers.Flatten())
network.add(layers.Dense(256,activation='relu'))
network.add(layers.Dropout(0.5))
network.add(layers.Dense(5,activation = 'softmax'))

network.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_4 (Dense)              (None, 256)               2097408
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_5 (Dense)              (None, 5)                 1285
=================================================================
Total params: 16,813,381
Trainable params: 16,813,381
Non-trainable params: 0
_________________________________________________________________
```

Figure 4






```
#report the accuracy on test set
test_datagenerator = test_datagen.flow_from_directory

test_loss , test_acc = network.evaluate_generator(tes
print("test accuracy  =",test_acc)
```

```
Found 50 images belonging to 5 classes.
WARNING:tensorflow:Your input ran out of data; interr
least `steps_per_epoch * epochs` batches (in this cas
ur dataset.
test accuracy  = 0.6200000047683716
```

Figure 5

[Model 4] Transfer learning by directly using the VGG16 base was implemented. With a dropout of 0.5 between 2 dense layers and a Softmax activation (Figure4) over the duration of 25 epochs the model was fit steps per epoch, 10 validation steps and had compiled with a learning rate of 2e-5 resulted in 62% accuracy on the test set. Which is an 8% improvement from the last model. (Figure5)

[Model 5] To improve this model even further fine tuning was applied by making a particular layer trainable(block5) from the VGG16 base layers and disabling every other layer from being trainable.

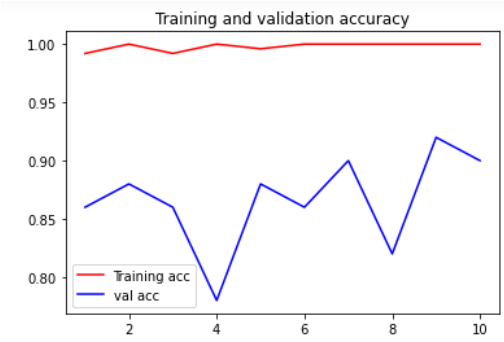
With the use of a loop (Figure 6) a selected convolutional block5 layer turned into a trainable one and 2 if statements within the loop. One that checks if a given layer name matches the one that has been selected to be trainable; if it is true, it will set a Boolean variable to true. The other if

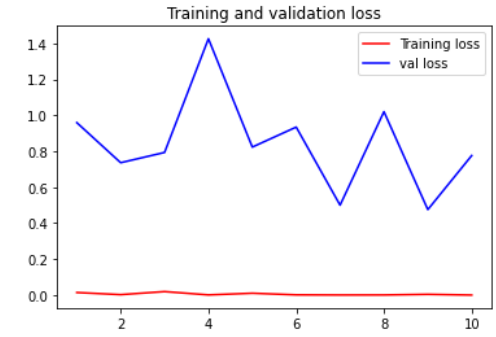
statement checks a Boolean is true, if the condition is met then the given layer's trainable property will be set to true else it will be set to false. The model was compiled with a learning rate of 1e-5 and was fit with the same settings as the last model.

```
# set all layers to trainable
vgg_base.trainable =True
```

```
#set block 5 layers as trainable, rest not as trainable
set_trainable=False
for layer in vgg_base.layers:
    if layer.name =='block5_conv1':
        set_trainable=True
    if set_trainable:
        layer_trainable=True
    else:
        layer_trainable=False
```

Figure 6






```
#report the accuracy on test set
test_datagenerator = test_datagen.flow_from_directory(tes

test_loss , test_acc = network.evaluate_generator(test_da
print("test accuracy of fine tuned version =",test_acc)
```

```
Found 50 images belonging to 5 classes.
WARNING:tensorflow:Your input ran out of data; interrupti
least `steps_per_epoch * epochs` batches (in this case, 1
ur dataset.
test accuracy of fine tuned version = 0.7200000286102295
```

Figure 7

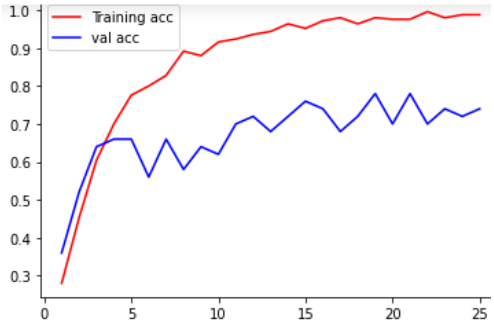
The accuracy on the test set increased to 72% (Figure 7), 10% higher than the last model.

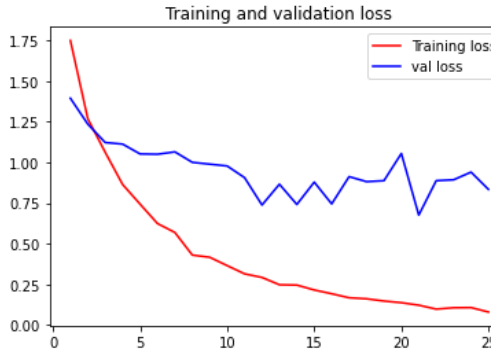
Models 5 and 6 also used Data Augmentation to improve the results. The ImageDataGenerator class has properties that can be used to create images with different properties on the fly (do not exist after runtime). The properties that were used are:

Rotation range:	40
Rescale:	1./255
Width shift range:	0.2
Height shift range:	0.2
Shear range:	0.2
Horizontal flip:	True
Fill mode:	Nearest (Stretches pixels to make the image fit)

[Model 5] Without the use of Data Augmentation (Figure11)

[Model 4] Without the use of Data Augmentation (Figure12)

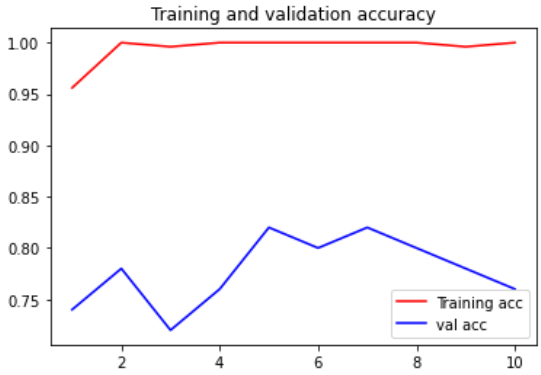


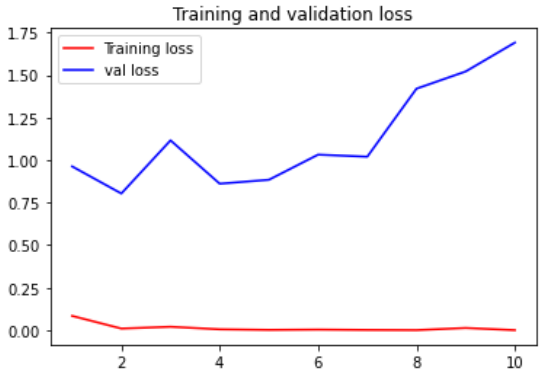



```
#report the accuracy on test set
test_datagenerator = test_datagen.flow_from_dir

test_loss , test_acc = network.evaluate_generat
print("test accuracy  =",test_acc)
```

```
Found 50 images belonging to 5 classes.
WARNING:tensorflow:From <ipython-input-8-45426f
g) is deprecated and will be removed in a futur
Instructions for updating:
Please use Model.evaluate, which supports gener
WARNING:tensorflow:Your input ran out of data;
least `steps_per_epoch * epochs` batches (in th
ur dataset.
test accuracy  = 0.5600000023841858
```

Figure 11






```
#report the accuracy on test set
test_datagenerator = test_datagen.flow_from_directory(tes

test_loss , test_acc = network.evaluate_generator(test_da
print("test accuracy of fine tuned version =",test_acc)
```

```
Found 50 images belonging to 5 classes.
WARNING:tensorflow:Your input ran out of data; interrupti
least `steps_per_epoch * epochs` batches (in this case, 1
ur dataset.
test accuracy of fine tuned version = 0.6000000238418579
```

Figure 12

Model 5's accuracy dropped by 12% (72%->60%)

Model 4's accuracy dropped by 10% (66%->56%)

2.3.2 Best Possible Model

The best results were produced using transfer learning and VGG16 but this time with a feature extraction method. (Figure 8)

Transfer Learning: Feature extraction (Alt Approach)

```
from keras.applications import VGG16
vgg_base = VGG16(weights = 'imagenet', include_top = False, input_shape=(150,150,3))
#vgg_base.summary()
```

```
def extract_features(dir, sample_count):
    features = np.zeros((sample_count,4,4,512))
    labels =  np.zeros((sample_count, 5))
    generator = datagen.flow_from_directory(dir,target_size=(150,150),batch_size = bat

    i = 0

    for inputs_batch, labels_batch in generator:
        features_batch = vgg_base.predict(inputs_batch)

        features[i*batch_size:(i+1)*batch_size] = features_batch
       # print(features.shape)
        labels[i*batch_size:(i+1)*batch_size] = labels_batch
        i +=1

        if i *batch_size >= sample_count:
            break

    return features,labels
```

```
train_features , train_labels = extract_features(train_dir,250)
validation_features, validation_labels = extract_features(val_dir,50)
test_features,test_labels = extract_features(test_dir,100)
```

```
Found 250 images belonging to 5 classes.
Found 50 images belonging to 5 classes.
Found 100 images belonging to 5 classes.
```

```
train_features = np.reshape(train_features,(250,4*4*512))
validation_features = np.reshape(validation_features,(50,4*4*512))
test_features = np.reshape(test_features,(100,4*4*512))
```

Figure 8

A method that extracts features and labels by taking the directory and the number of inputs as its parameters. After the features and labels are extracted, they are reshaped.

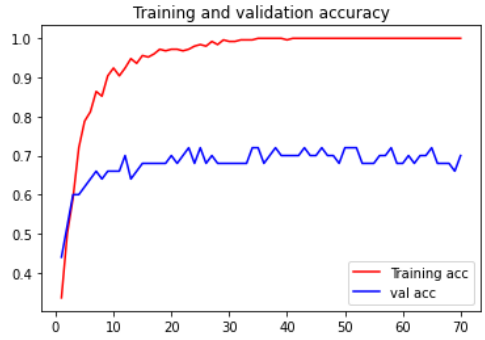
```
from keras import models,layers, optimizers

network = models.Sequential()
network.add(layers.Dense(256,activation = 'relu', input_shape = (4*4*512,)))
network.add(layers.Dropout(0.5))
network.add(layers.Dense(5,activation = 'softmax'))


network.compile(optimizer= optimizers.RMSprop(lr=2e-5),loss = 'categorical_crossentropy', metrics = ['acc'])
network.summary()
```

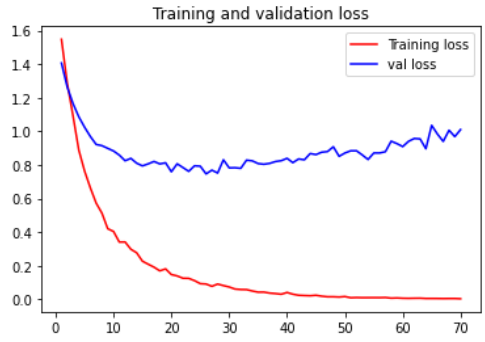
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 256)               2097408
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 1285
=================================================================
Total params: 2,098,693
Trainable params: 2,098,693
Non-trainable params: 0
_________________________________________________________________
```

```
history = network.fit(train_features,train_labels ,
                      epochs = 70,batch_size = 10, validation_data = (validation_features,validation_labels)
```

Figure 9

Only two dense layers were used, a Dropout of 0.5 and the last layer has a Softmax activation with 5 neurons for five classes.






```
4/4 [==============================] - 0s 2ms/step - loss: 0.8014 - acc: 0.7700
test accuracy  = 0.7699999809265137
```

Figure 10

The model was compiled with a learning rate of 2e-5 and was fit with train and validation labels, and features for 70 epochs. The results display a training accuracy that reaches 100%, a validation accuracy that is stable at 70%, a training loss that is very close to zero while the validation loss starts to overfit after 30 epochs. All in all, the test accuracy is roughly 77% a 5% improvement from the last model.

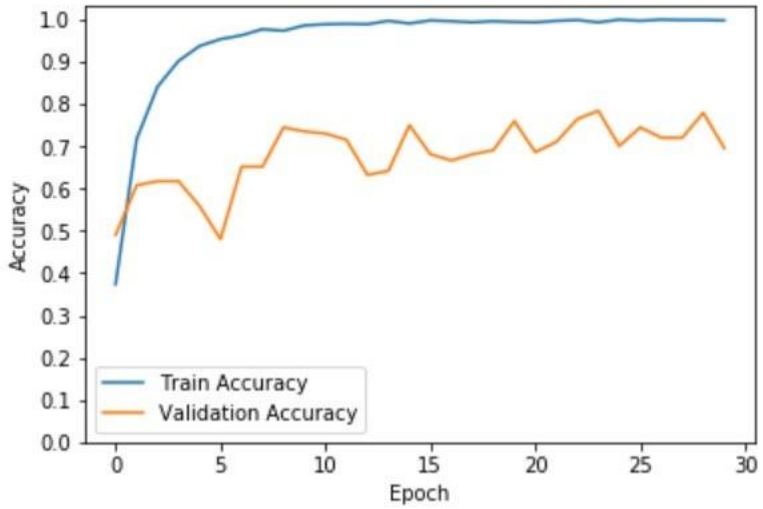
3 Comparison of the 2 models and contrast with results of published work

3.1.1 Comparison of the 2 models

The simple model has a total accuracy of 46% while the advanced model has an accuracy of 77%, a 30% improvement. The simple model uses Convolutional layers, Maxpooling and two Dense layers with an input shape of (150,150,3) compared to the advanced model that has 2 dense layers and a dropout of 0.5 with an input shape of (4*4*512,). The simple model's highest accuracy almost reached 90% and its validation accuracy never

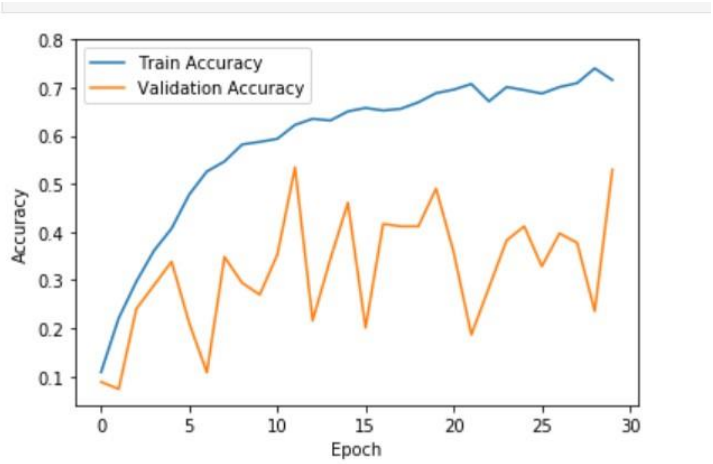
surpassed 70%, the advanced model was able surpass 90% after 20 epochs and its validation accuracy had reached 70% after 15 epochs and had steady a rate never surpassing 70% for the remainder of the epochs which makes the advanced model a slightly more accurate model. The simple model's loss started from 160% and decreased all the way down to 40% in 20 epochs and its validation loss started from 140% and consistently stayed over 100% except a few times where it reached 90% until the end. The advanced model's loss started from 160% and decreased in 50 epochs to reach an almost 0%, its validation loss was able to reach 80% and after 30 epochs it started to overfit.

3.1.2 Published work

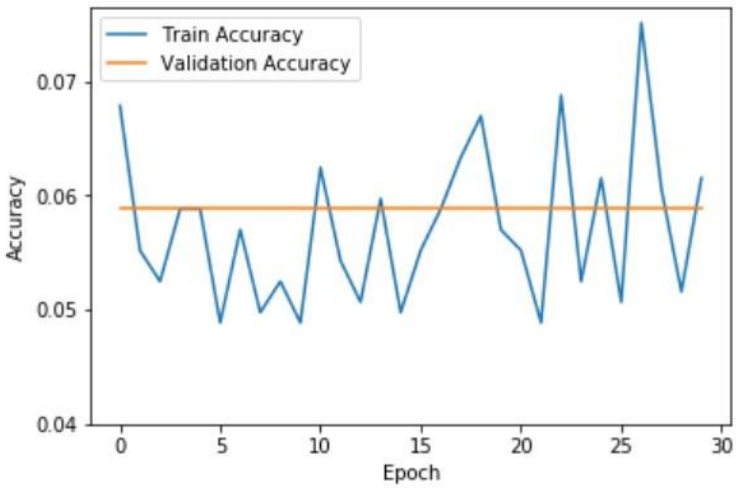


The publisher [5] used transfer learning with similar layers only two dense layers were used, a Dropout of 0.5 and the last layer has a Softmax activation with 5 neurons for five classes. The model was compiled with a learning rate of 2e-4 and was fit with train and validation labels, and features for 70 epochs. The results display a training accuracy that reaches 100%, a validation accuracy that is stable at 70%.

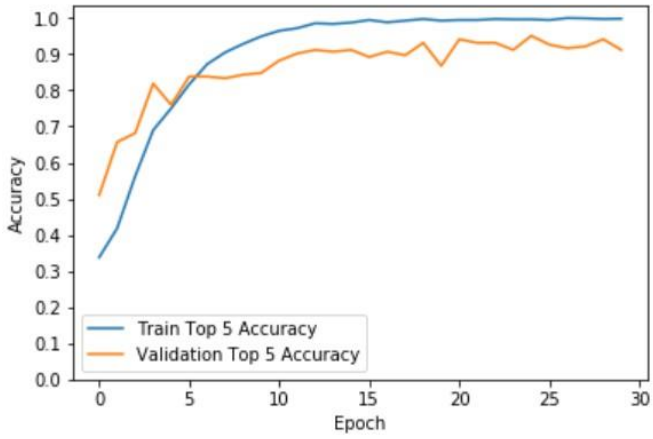
For their second run they added an extra dense layer and a regularizer l2 with a learning rate of 0.1 on the first dense layer and a regularizer l2 of 0.01 on the second the dense layer. The network is fit with an optimiser that uses a learning rate of 0.001.



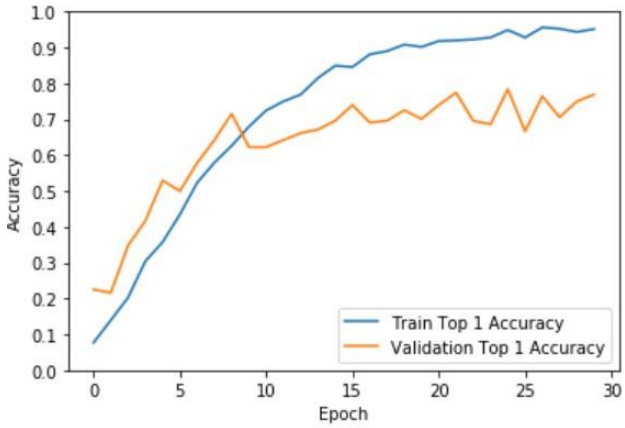
The results show a training accuracy that reaches a little over 70%, the validation accuracy that is below 60%.



For the third run the publisher changed the learning of their regularizers to 0.1 and used regularizer l1 which made accuracy reach close to 0% for both training and validation



The publisher increased the accuracy of their model by adding a dropout of 0.5 to each of their 3 dense layers. The top 5 accuracies of the training data go up to 100% and the validation accuracies reach near 90% after 15 epochs.



The final model's top1 training accuracy is little under 100% while the validation's top 1 after the 7th epoch consistently stays at 70%.

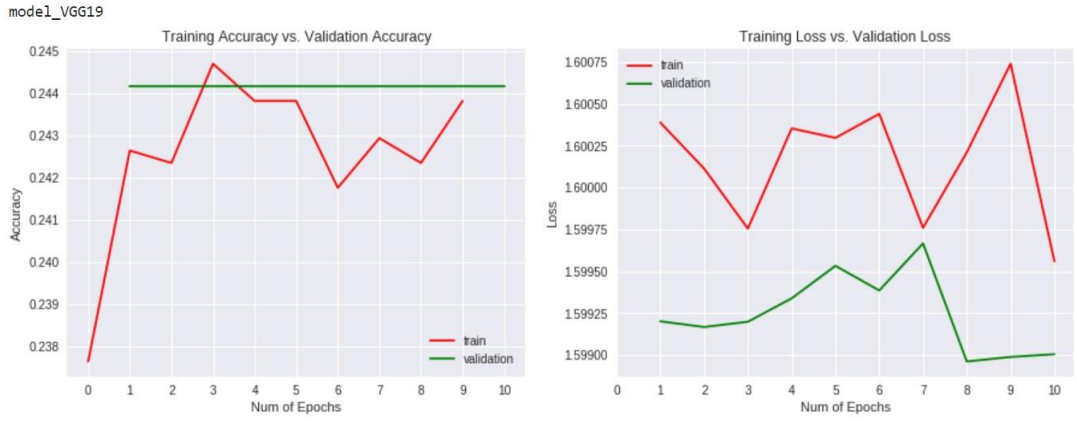
All in all, the results of my advanced model did not exceed the results from this publisher's model the accuracies were roughly about the same, as for the comparison of the simple model the publisher did not have a simple model to allow comparison with mine.

The publisher [6]

For their first model used 6 convolutional layers 3 Maxpooling layers with a dropout of 0.25 every 3 layers until 2 layers before the final layer where a dropout of 0.5 was used , the last layer has a Softmax activation.

For 50 epochs the training and validation accuracy lines seem to be very close to each other and have an accuracy of 70% in comparison my advanced model has a higher training accuracy and a similar validation accuracy.

For 50 epochs the training and validation accuracy lines again seem to follow to each other, they start from 160% and have a loss of 80% at their best in comparison my advanced model has a better training loss as it almost reaches 0% and the validation loss stays over 100%.



For the second model 2 dense layers were used and a dropout of 0.5.

The accuracies reached a little over 24% while the losses were over 140% ,compared to my model their model did not perform well.

4 Lessons learned

After running extensive tests by changing the number of layers and activation types I have not seen a big difference. On the other hand, techniques like augmentation have improved the accuracy considerably. Data visualization was not useful for the purpose of this assignment which is to have the highest accuracy possible on the test set of the advanced model. Regularizers seem to help with overfitting just like Dropouts although in some cases they increased the loss of training and validation. Transfer learning which uses a pretrained database that has generalised features of many objects and beings is a useful technique that helped with increasing the accuracy considerably, in this module two methods were used in the workshops I tested both and had better results than my previous models that did not use it. In particular one method used the VGG16 database directly as a layer and then fine tuned it by disabling some layers from being trainable, this allowed for a 10 percent improvement in accuracy. The other method of transfer learning which is feature extraction extracts features and labels from the input that it is given and after fitting the model it was able to produce the higher accuracy on the test than any other model I had created. Lastly, I understood the importance of data augmentation as it can edit input images on the fly without having to save the changes, images can be often not easily readable and so applying some settings a property ‘nearest’ in the

ImageDataGenerator class can stretch an image to make it more realistic and more readable rather than it is having black borders.

5 References

[1] Ou, G. and Murphey, Y.L., 2007. Multi-class pattern classification using neural networks. *Pattern Recognition*, *40*(1), pp.4-18. https://doi.org/10.1016/j.patcog.2006.04.041

[2] GeeksforGeeks, 2020. An introduction to multlabel classification. An introduction to MultiLabel classification - GeeksforGeeks

[3] Pushprajmaraje, 2020. Simple multi-class classification using CNN for custom dataset. Simple Multi-Class Classification using CNN for custom Dataset. | by Pushprajmaraje | Analytics Vidhya | Medium

[4] Programming Review, Regularization for Deep Learning: A taxonomy (Jan Kukacha, Vladimir Golkov, and Daniel Cremers). HOW TO PREVENT THE OVERFITTING | REGULARIZATION — PROGRAMMING REVIEW (programming-review.com)

[5] Ujaitley. Convolutional Neural Network Convolutional-Neural-Network--17-Flower-Category-Classification/CNN_Image_Classification.ipynb at master · ujaitley/Convolutional-Neural-Network--17-Flower-Category-Classification · GitHub

[6] RockyXu66. Kaggle flowers classification keras. Kaggle_Flowers_Classification_Keras/Flower_Classification_Keras.ipynb at master · RockyXu66/Kaggle_Flowers_Classification_Keras · GitHub