CS582 Project 2

Deep learning: Vision and NLP
Machine learning, CS, MIU

Cong Trang Truong – 612731
Khoa Nam Nguyen – 612744
Thanh Nhan Nguyen – 612745
Thai Trung Phan – 612819
Thai Binh Duong Nguyen – 612700
Anh Thong Tran – 612723

Professor: **Anthony Sander**

Aug 12, 2021

Contents

Abstract

Transfer learning is a method of reusing a pre-trained model knowledge for another task. Transfer learning can be used for classification, regression and clustering problems. This project uses one of the pre-trained models – VGG16 with Deep Convolutional Neural Network to classify images from the flowers dataset.

1 Problem Statement

We picked a **Computer Vision** task. The data used in this study is called: *Flowers Recognition from Kaggle*[1].

2 EDA and Data Preparation

2.1 Dataset

This dataset contains 4590 images of flowers. The data collection is based on the data flickr, google images, yandex images. We can use this data set to recognize plants from the photo. The pictures are divided into five classes: chamomile, tulip, rose, sunflower, dandelion. For each class there are about 800 photos. Photos are not high resolution, about 320x240 pixels. Photos are not reduced to a single size, they have different proportions.

2.2 EDA

The number of samples in each category present:

```
Total number of flowers in the dataset:  4590
Flowers in each category:
tulip        1200
dandelion    1060
rose          784
sunflower     782
daisy         764
Name: category, dtype: int64
```

Figure 1: Number of samples

Let's visualize it

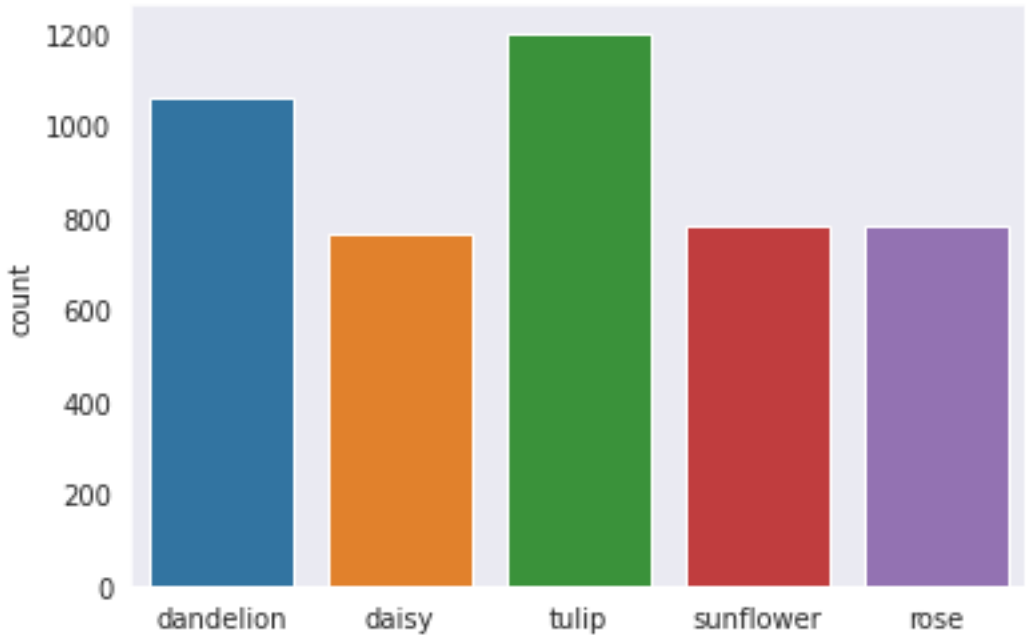

Figure 2: Distribution histogram

The data set seems to be balanced as for each training label , enough training examples exist.

Let's review random 10 images:

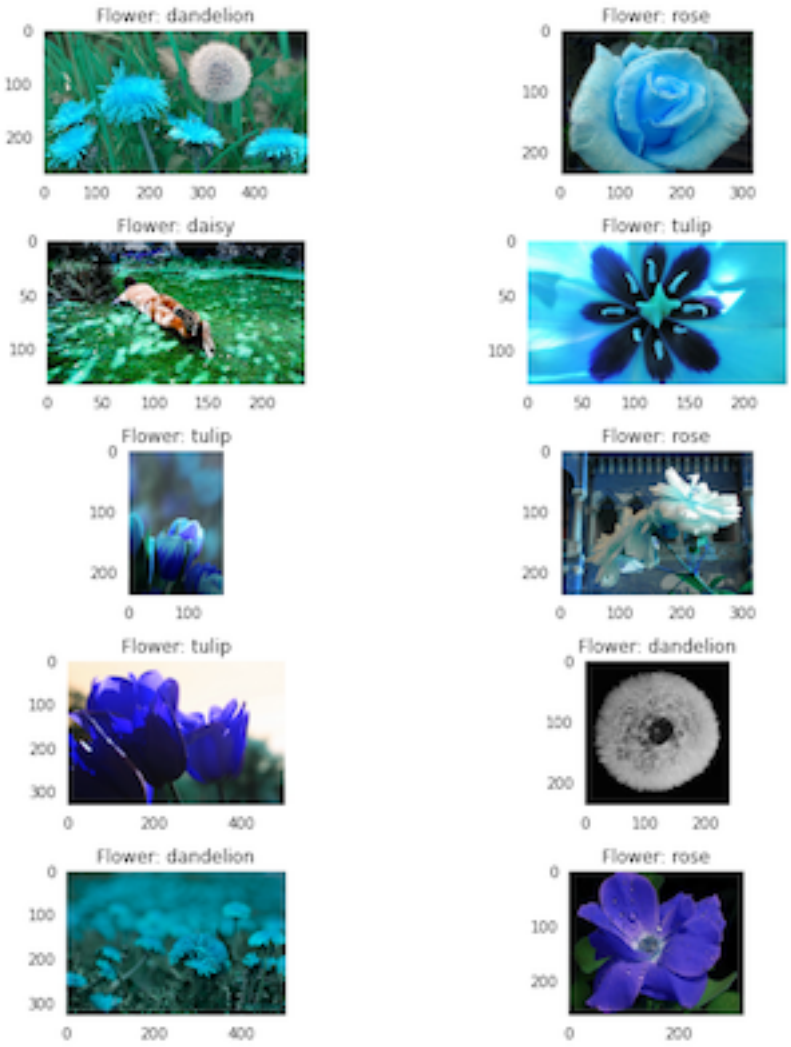

Figure 3: Random images

After visualizing data, we perform data preparation. We will use the api *ImageDataGenerator()* of Keras to do both image resize and data normalization. Because images of the datasets have different proportions, and the input layer of VGG16 is 224 x 224 so we need to resize all images to 224 x 224. We also the grayscale normalization to convert an input image into a range of pixel values that are more familiar or normal to the senses.
Moreover, we split the input data into two sets: a training set and a test set.

```
train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
training_set = train_datagen.flow_from_directory('data/flowers',
    target_size=(224, 224),
    subset='training')
test_set = train_datagen.flow_from_directory('data/flowers',
    target_size=(224, 224),
    subset='validation')
```

```
Found 3674 images belonging to 5 classes.
Found 916 images belonging to 5 classes.
(224, 224, 3)
```

Figure 4: Training set and Test set

3 Model Architect

3.1 Obtain the pre-trained model

For our transfer learning, we will use the pre-trained VGG16 model, which is a convolutional neural network trained on 14 million images to classify 1000 different classes. We chose VGG16 because it was able to achieve around 92.7% top-5 test accuracy in ImageNet and its weights are freely available and can be loaded and used in any models and applications.

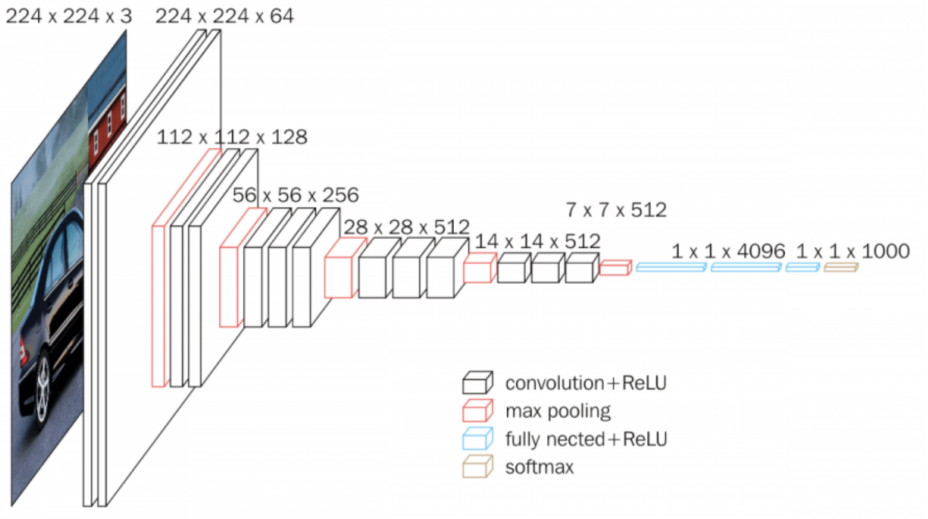

Figure 5: VGG16 Architecture [2]

It will be easier to look at the layered architecture of VGG16:

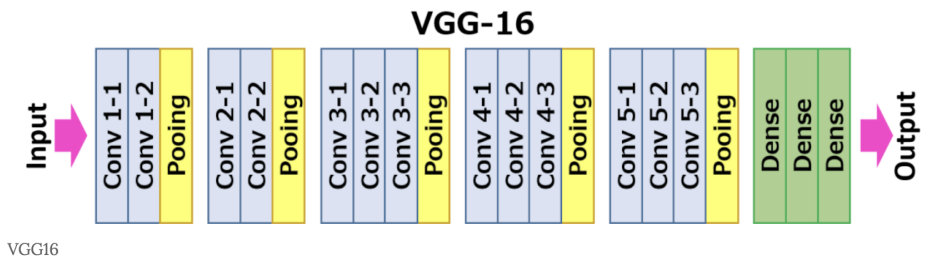

Figure 6: VGG16 Layered Architecture [2]

From the the layered architecture, we can see that VGG16 have 5 Convolution block and 1 fully connected block. Each convolution block having several convolution layers and 1 pooling layer. Convolution layers are using 3×3 filters. Pooling layers are using 2×2 Max Pooling. Fully connected block contains 3 fully connected layers, which will be replaced by defining a custom model on top of pre-trained model

3.2 Freeze Layers

By freezing or setting layer.trainable = False, we prevent the weights in a given layer from being updated during training. If they are updated, then we will lose all the learning that has already taken place. Therefore it will lead to no different from training the model from scratch. In our case, we are freezing all the convolutional block of VGG16.

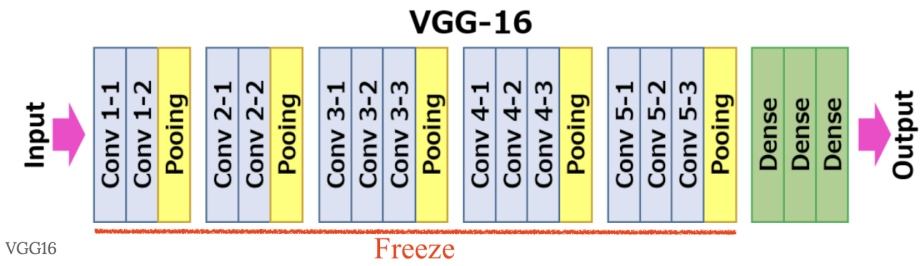

Figure 7: Freeze Layers

3.3 Replace the fully connected layers

We defined custom layers on top of the pre-trained layers in the VGG16 model. In our model, we have replaced it with our own three dense layers of dimension 512×128 with ReLU activation and then the output layer for classification.

```
Model: "model_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_11 (InputLayer)        [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
global_max_pooling2d_10 (Glo (None, 512)               0
_________________________________________________________________
dense_30 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_31 (Dense)             (None, 128)               65664
_________________________________________________________________
dense_32 (Dense)             (None, 5)                 645
=================================================================
Total params: 15,043,653
Trainable params: 328,965
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 8: Model Summary

3.4 Train Model

Here we have created a new model by combining the input of the pre-trained VGG16 model and new added layers. Once we have a new model we will compile it:

```
model.compile(optimizer=optimizers.Adam(),
loss=losses.categorical_crossentropy, metrics=['accuracy'])
```

We used the Adam optimizer with a default learning rate, categorical_crossentropy as our loss, and accuracy as our metrics.

Now we can train our model on the training set and evaluating it on the test set:

```
history = model.fit(training_set, validation_data=test_set, epochs=20,
callbacks=[early_stop_callback, checkpoint_callback])
```

We called model.fit() function, which will train the model and repeatedly iterating over the entire dataset for a given 20 number of epochs. We passed in callback parameters with ModelCheckpoint callback to save best weights and EarlyStopping callback to stop train. Then we returned a history object that holds a record of the loss values and accuracy values during training.

```
Epoch 00002: val_loss improved from 0.81267 to 0.57440, saving model to /content/drive/MyDrive/deeplearning/flowers/trained_model/vgg16_flowers_model_1.h5
Epoch 3/50
109/109 [==============================] - 17s 153ms/step - loss: 0.5053 - accuracy: 0.8186 - val_loss: 0.5096 - val_accuracy: 0.8035

Epoch 00003: val_loss improved from 0.57440 to 0.50963, saving model to /content/drive/MyDrive/deeplearning/flowers/trained_model/vgg16_flowers_model_1.h5
Epoch 4/50
109/109 [==============================] - 17s 155ms/step - loss: 0.6087 - accuracy: 0.7903 - val_loss: 0.5270 - val_accuracy: 0.8000

Epoch 00004: val_loss did not improve from 0.50963
Epoch 5/50
109/109 [==============================] - 16s 149ms/step - loss: 0.3992 - accuracy: 0.8588 - val_loss: 0.5169 - val_accuracy: 0.8093

Epoch 00005: val_loss did not improve from 0.50963
Epoch 6/50
109/109 [==============================] - 16s 149ms/step - loss: 0.3709 - accuracy: 0.8701 - val_loss: 0.5572 - val_accuracy: 0.8035

Epoch 00006: val_loss did not improve from 0.50963
Epoch 7/50
109/109 [==============================] - 17s 154ms/step - loss: 0.3459 - accuracy: 0.8756 - val_loss: 0.4824 - val_accuracy: 0.8209

Epoch 00007: val_loss improved from 0.50963 to 0.48241, saving model to /content/drive/MyDrive/deeplearning/flowers/trained_model/vgg16_flowers_model_1.h5
Epoch 8/50
109/109 [==============================] - 17s 153ms/step - loss: 0.2946 - accuracy: 0.8933 - val_loss: 0.5234 - val_accuracy: 0.8081

Epoch 00008: val_loss did not improve from 0.48241
Epoch 9/50
109/109 [==============================] - 16s 149ms/step - loss: 0.2574 - accuracy: 0.9051 - val_loss: 0.6316 - val_accuracy: 0.7860

Epoch 00009: val_loss did not improve from 0.48241
Epoch 10/50
109/109 [==============================] - 16s 146ms/step - loss: 0.2748 - accuracy: 0.8982 - val_loss: 0.6336 - val_accuracy: 0.8093

Epoch 00010: val_loss did not improve from 0.48241
Epoch 11/50
109/109 [==============================] - 17s 155ms/step - loss: 0.2644 - accuracy: 0.9028 - val_loss: 0.6577 - val_accuracy: 0.7953

Epoch 00011: val_loss did not improve from 0.48241
Epoch 12/50
109/109 [==============================] - 17s 154ms/step - loss: 0.2143 - accuracy: 0.9228 - val_loss: 0.5910 - val_accuracy: 0.8198
Restoring model weights from the end of the best epoch.

Epoch 00012: val_loss did not improve from 0.48241
Epoch 00012: early stopping
Evaluate model
27/27 [==============================] - 3s 125ms/step - loss: 0.4824 - accuracy: 0.8209
Accuracy on TestSet: 82.093
```

Figure 9: Train Model

Evaluating this model on the test set we got a 82.093% accuracy.

Here we visualize loss and accuracy during training from the history object:

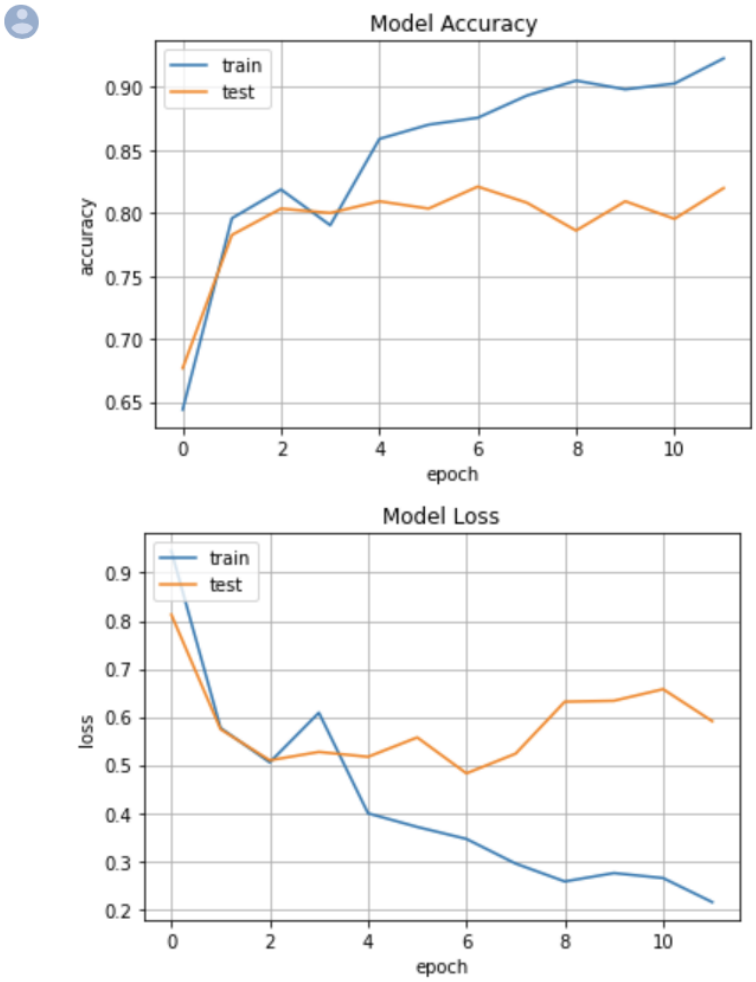

Figure 10: Model Accuracy and Loss

4 Model Tuning

Below here is what we will try to fine tune the model to get a higher accuracy and a lower loss score.

As we already trained the base model with epoch = 20, learning_rate = 0.001, batch_size = 32, we got a evaluation on the test set with the loss = 0.4824 and accuracy = 0.8209 We have not trained the model with any augmentation techniques or add any drop-out or batch-norm layers to the model so far.

So let us start to fine-tune the model by adding drop-out layer and conducting the augmentation before training

4.1 Base Model + Drop-out layers added + Augmentation conducted

The augmentation now includes the following characteristics:

* shear_range=0.2 that means shear the image by 20%

* zoom_range=0.2 that means zoom the image by 20%

* rotation_range=10 that means rotating images between 0 and 10 degrees

* horizontal_flip=True for mirror reflection.

The structure of the model now turn to be as follows:

```
Model: "model_63"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_36 (InputLayer)        [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
global_max_pooling2d_35 (Glo (None, 512)               0
_________________________________________________________________
dense_105 (Dense)            (None, 512)               262656
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_106 (Dense)            (None, 128)               65664
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_107 (Dense)            (None, 5)                 645
=================================================================
Total params: 15,043,653
Trainable params: 328,965
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 11: The structure of the model added drop-out layers and augmentation

4.1.1 drop-out = 0.5

That mean 50% of nodes (neurons) retain is implemented as 50% of nodes (neurons) is dropped out.

After evaluation on testset we got the loss = 0.5196 and the accuracy = 0.8058. The loss increases a bit a bout 0.0372 and its accuracy is decreased by about 1.51%, which is bad to use it.

Therefore, We tried a lower drop-out of 0.2 to check it if it helps.

4.1.2 drop-out = 0.2

That mean 80% of nodes (neurons) retain is implemented as 20% of nodes (neurons) is dropped out.

After evaluation on testset we got the loss = 0.5112 and accuracy = 0.8302. Although the loss increase a bit a bout 0.0288, but its accuracy is increased by about 0.93% (1%).

Below is the learning curve of the accuracy and the loss of the model:

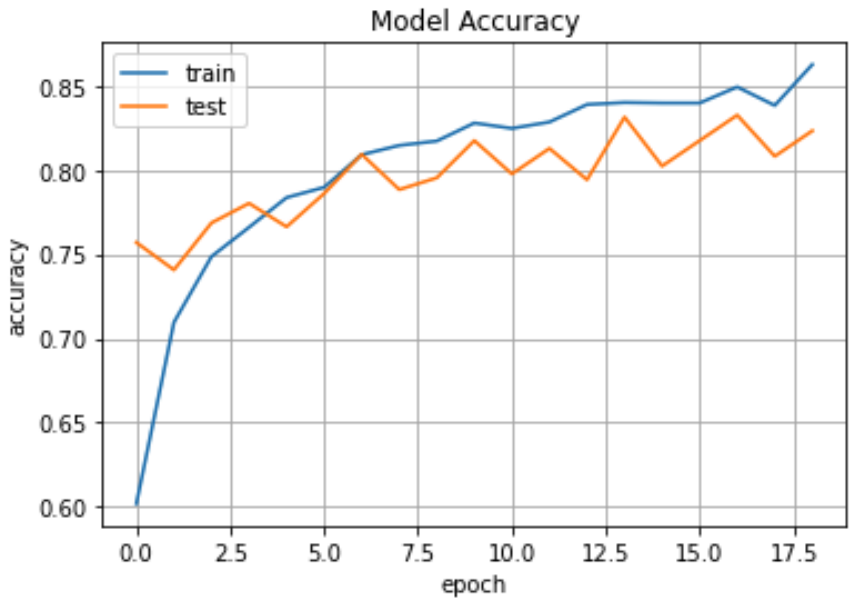

Figure 12: The learning curve of the accuracy

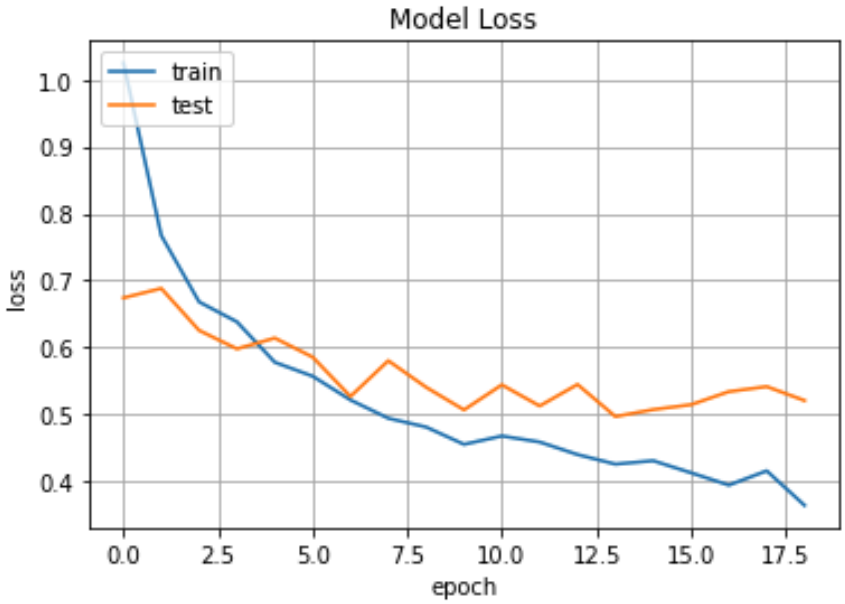

Figure 13: The learning curve of the loss

Since we use EarlyStoping callback, after 5 epochs without any improvement in validation loss, the model stopped training at 19th epoch.

4.2 Base Model + Drop-out + Augmentation + Try Multiple Batch_Size

We tried the model added dropout layers and augmentation with multiple values of batch_size of 32, 64, or 128 to check if the more batch_size is , the more the accuracy or the less the loss is or it makes the otherwise.

With batch_size = 32, we had the result of the loss = 0.5112 and the accuracy = 0.8302 as shown in the the chapter 4.1

With batch_size = 64, we had the result of the loss loss = 0.5477 and the accuracy = 0.8035 which are much worse than the previous one of batch_size = 32

Therefore, we will continue to fine tune the model with batch_size = 32 because if we keep decreasing the batch_size down to the lower level, the downside of using a smaller batch size is that the model is not guaranteed to converge to the global optima. This is intuitively explained by the fact that smaller batch sizes allow the model to "start learning before having to see all the data.". In general, batch size of 32 is a good starting point.

4.3 Base Model + Drop-out + Augmentation + ReduceLROnPlateau

ReduceLROnPlateau is the callback function that will be called to reduce learning rate when a metric has stopped improving. The learning rate will be modified whenever a new epoch starts (based on that function). Here the "val_loss" is set as a metric of ReduceLROnPlateau that is used in our case.

After evaluation on testset we got the loss = 0.4790 and the accuracy = 0.8360. The loss now decreased by about 0.0034 and the accuracy increased by about 1.51 compared with the base model and the loss decreased by about 0.0322 and the accuracy increased by about 0.58 compared with the model added drop-out and augmentation, which is the best one so far.

Below is the learning curve of the accuracy and the loss score of the model:

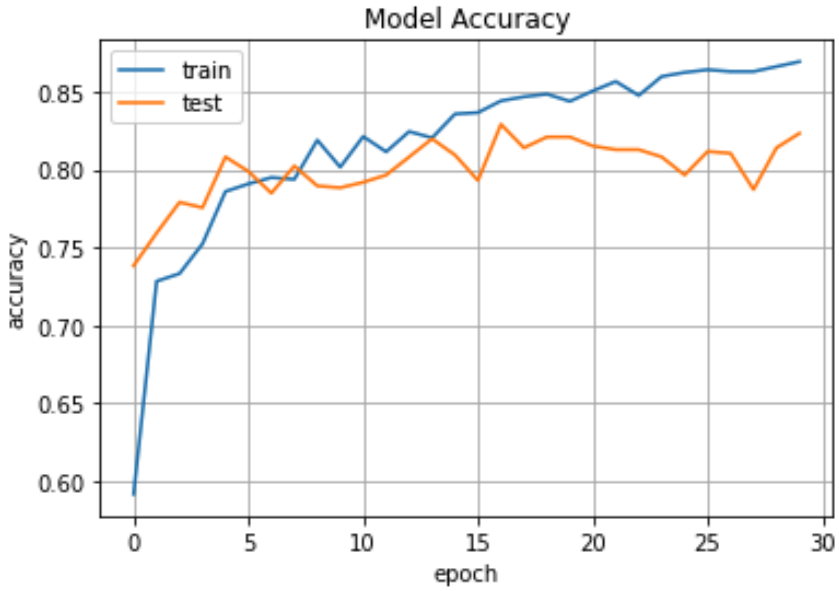

Figure 14: The learning curve of the accuracy

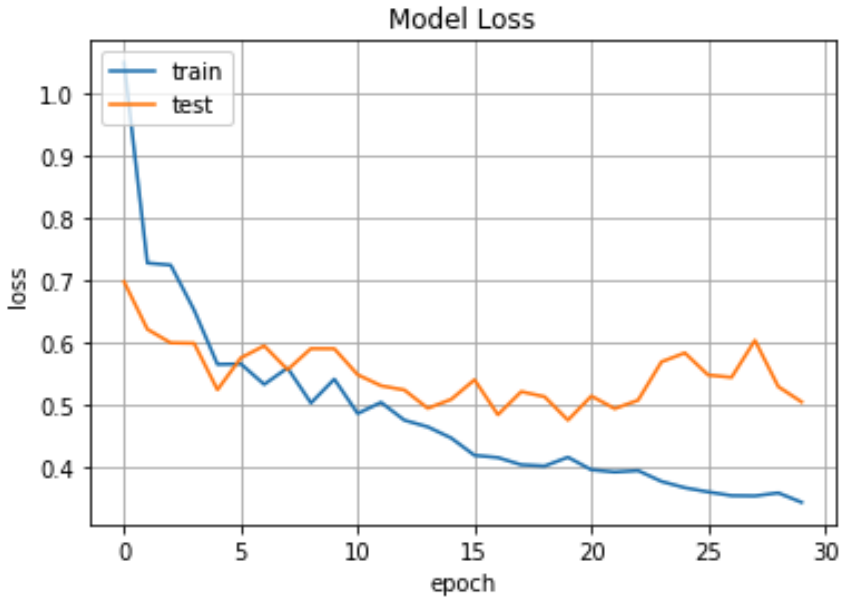

Figure 15: The learning curve of the loss

4.4 Base Model + Drop-out + Augmentation + ReduceLROnPlateau + featurewise_center = True

In previous models, we re-scaled 1./255 for transforming every pixel value from range [0,255] to [0,1]. However, after researching about pretrained model VGG16 we know that the VGG16 model was trained on a specific ImageNet challenge dataset. As such, the model expects images to be centered. That is, to have the mean pixel values from each channel (red, green, and blue) as calculated on the ImageNet training dataset subtracted from the input.

We can achieve this effect with the image data generator, by setting the featurewise center argument to True and manually specifying the mean pixel values to use when centering as the mean values from the ImageNet training dataset: [123.68, 116.779, 103.939].

After evaluation on testset we got the loss = 0.4891 and the accuracy = 0.8407. The loss now increased by about 0.0067, but the accuracy increased by about 1.98% compared with the base model.

Below it is the learning curve of the accuracy score and the loss score:

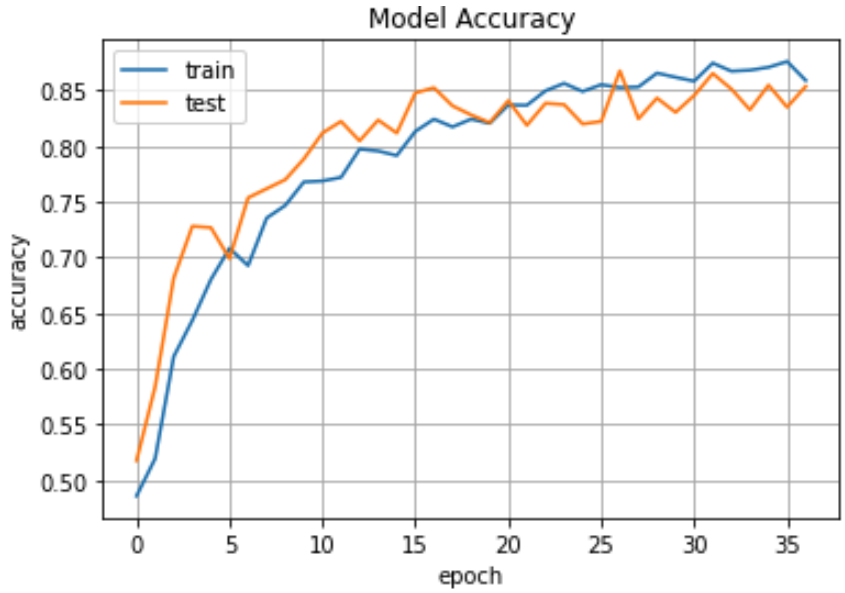

Figure 16: The learning curve of the accuracy

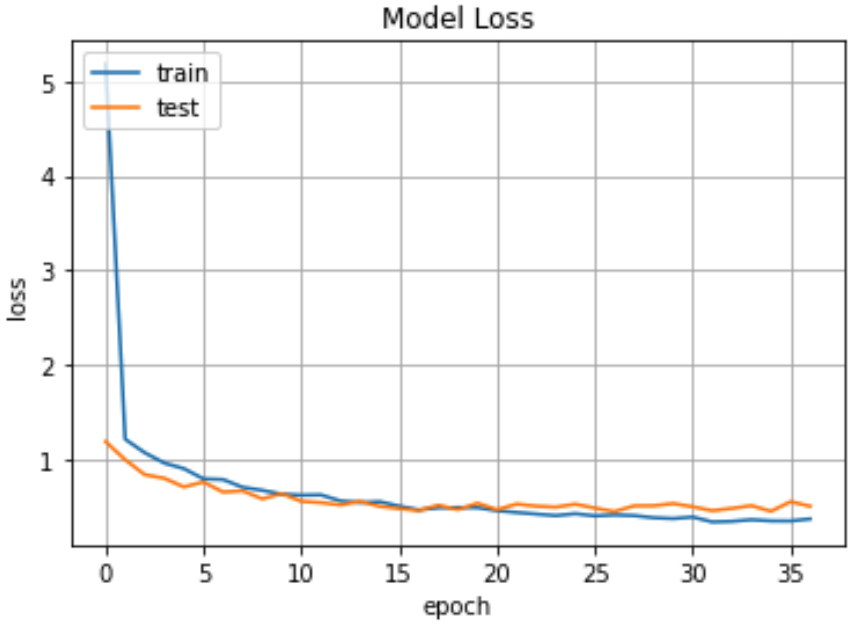

Figure 17: The learning curve of the loss

4.5 Base Model + Drop-out + Augmentation + ReduceLROnPlateau + featurewise_center = True + Batch_norm layers

We also try to add batch_norm layers to the dense layers of the model to check if it is more outperformed than the others, after evaluation we got the loss: 0.3971 and the accuracy: 0.8547, which is created by the best hyper-parameters sor far. The loss decreased by 0.0853 and the accuracy increased by 3.38%.

This is the learning curve for this model:

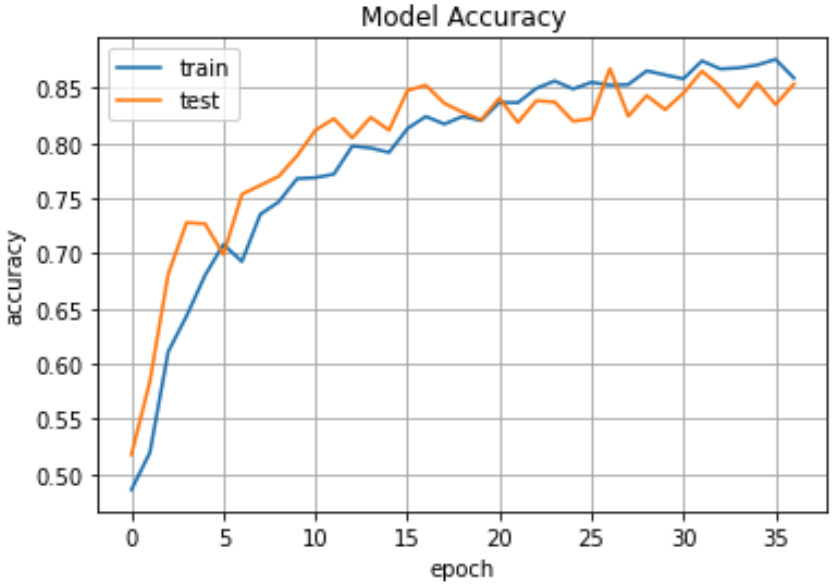

Figure 18: The learning curve of the accuracy

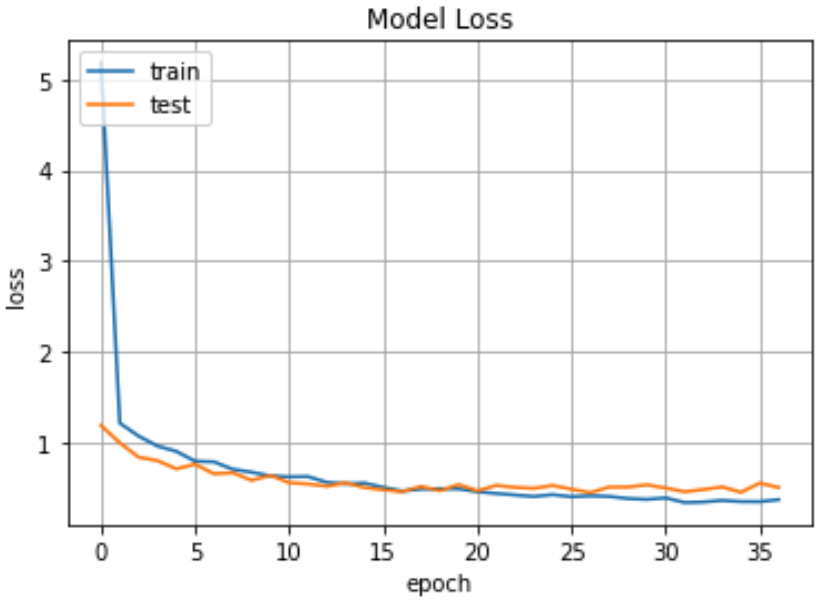

Figure 19: The learning curve of the loss

4.6 Conclusion for Model Tuning

After a series of training processes conducted, we can conclude that:

- The Model will perform well (the loss will decreases, and the accuracy will increase) when the drop_out layers, batch_norm layers are added, augmentation is conducted.
- the percentage of neurons that is dropped out also affected to the accuracy or the loss score, 20% of neurons dropping out is fine for our model, but it would be not outperformed if it is 50% or more.
- Before using ReduceLROnPlateau, the training process usually stops earlier at epochs less than 20 by the callback EarlyStop called because the model detected overfitting, but after using ReduceLROnPlateau, it adjusted the value of the learning_rate to make the model much better, prevented the model from overfitting so the training stopped later after epochs of 30 or 35.
- After a series of fine tuning conducted, the loss and the accuracy of the model changed considerably from 0.4824 and 82.09% to 0.3971 and 85.47% respectively.

5 CAM(Class Activation Maps)

We chose a technique for producing "visual explanations" for decisions from a large class of CNN-based models, making them more transparent. Our approach - Gradient-weighted Class Activation Mapping (Grad-CAM).
Why we chose Grad-CAM?
Because The technique does not require any modifications to the existing model architecture

and this allows it to apply to any CNN based architecture.

This is sample of my project about Visual Explanations from Deep Networks via Gradient-based Localization

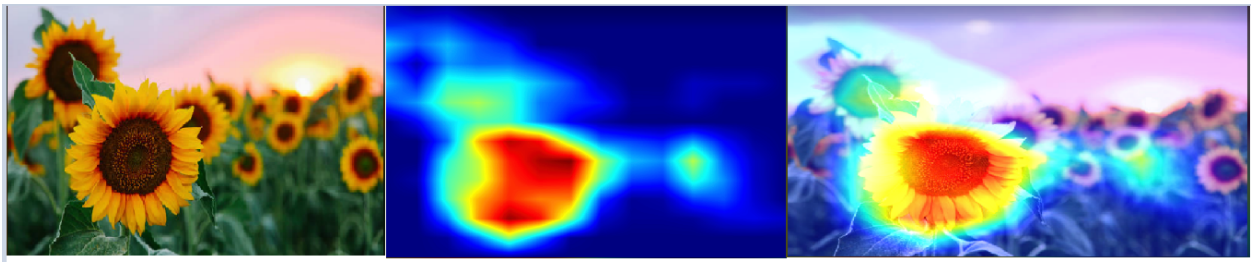

Figure 20: Sun flower

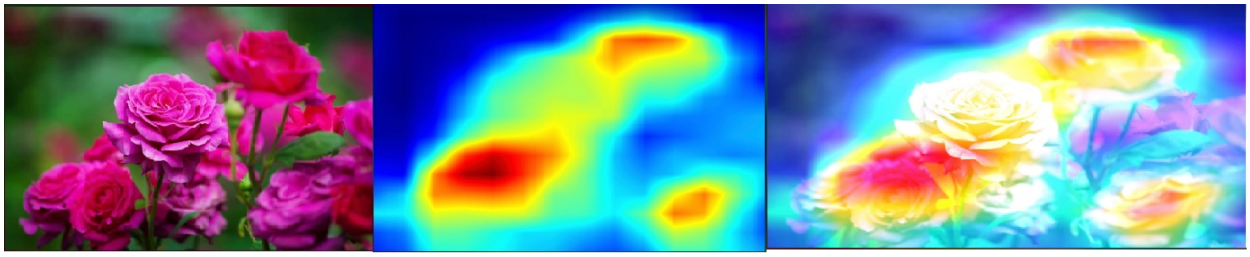

Figure 21: Rose

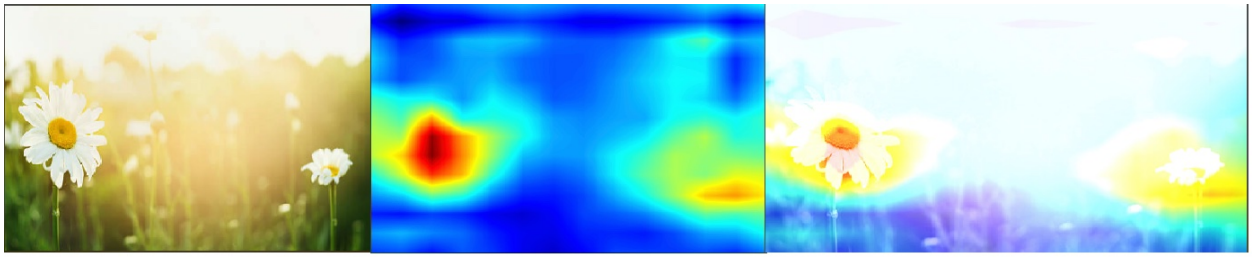

Figure 22: Daisy

The usage of CAMs allows you to not only see the class the network predicts, but also the part of the image the network is especially interested in. This helps to get more insights into the learnings of the network and to also ease up debugging, because the user gets an object localization of the predicted class without having to explicitly label the bounding box for this object

6 Conclusion

In this project, we used a pre-trained VGG16 model to classify images from flowers dataset using transfer learning. We applied some fine-tune techniques such as added drop out layers, batch normalization layers and did some augmentation, adjusted batch size, learning rate, etc, we was able to get a better accuracy score and a lower loss score. We applied Grad-CAM technique for producing visual explanations or interpreting the results of our model, making it more transparent. Therefore, we was able to achieve our goal which is to classify the types of flowers from the photos.

References

[1] A. Mamaev, "Flowers recognition." https://www.kaggle.com/alxmamaev/flowers-recognition/code.

[2] "Vgg16 – convolutional network for classification and detection." https://neurohive.io/en/popular-networks/vgg16/.