Flower Recognition

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!

```
import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow.keras.utils import to_categorical
import numpy as np
import·os
import·PIL
import·PIL.Image
```

Step 1: Load Dataset

```
# Load the data
(train_ds, train_labels), (test_ds, test_labels) = tfds.load("tf_flowers", split=["train[:70%]", "train[70%:]"], batch_size = -1, as_supervis
## Loading images and labels
#(train_ds, train_labels), (test_ds, test_labels) = tfds.load("tf_flowers",split=["train[:70%]", "train[:30%]"], batch_size=-1, as_supervised
 ## Train test split
 # Include labels


## Resizing images
train_ds = tf.image.resize(train_ds, (150, 150))
test_ds = tf.image.resize(test_ds, (150, 150))


## Transforming labels to correct format
train_labels = to_categorical(train_labels, num_classes=5)
test_labels = to_categorical(test_labels, num_classes=5)
```

Step 2: Load the Model

```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input


## Loading VGG16 model
base_model = VGG16(weights="imagenet", include_top=False, input_shape=train_ds[0].shape)
base_model.trainable = False ## Not trainable weights
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.
58889256/58889256 [==============================] - 1s 0us/step
```

```
## Preprocessing input
train_ds = preprocess_input(train_ds)
test_ds = preprocess_input(test_ds)
```

We use Include_top=False to remove the classification layer that was trained on the ImageNet dataset and set the model as not trainable. Also, we used the preprocess_input function from VGG16 to normalize the input data.

```
base_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 150, 150, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 150, 150, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 150, 150, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 75, 75, 64)        0         
```

```
 block2_conv1 (Conv2D)       (None, 75, 75, 128)       73856

 block2_conv2 (Conv2D)       (None, 75, 75, 128)       147584

 block2_pool (MaxPooling2D)  (None, 37, 37, 128)       0

 block3_conv1 (Conv2D)       (None, 37, 37, 256)       295168

 block3_conv2 (Conv2D)       (None, 37, 37, 256)       590080

 block3_conv3 (Conv2D)       (None, 37, 37, 256)       590080

 block3_pool (MaxPooling2D)  (None, 18, 18, 256)       0

 block4_conv1 (Conv2D)       (None, 18, 18, 512)       1180160

 block4_conv2 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_conv3 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 9, 9, 512)         0

 block5_conv1 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv2 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv3 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_________________________________________________________________
```

The model has over 14 Million trained parameters and ends with a maxpooling layer that belongs to the Feature Learning part of the network.

Step 3: Add dense layers specific for your problem

```
from tensorflow.keras import layers, models


flatten_layer = layers.Flatten()
dense_layer_1 = layers.Dense(50, activation='relu')
dense_layer_2 = layers.Dense(20, activation='relu')
prediction_layer = layers.Dense(5, activation='softmax')


model = models.Sequential([
base_model,
flatten_layer,
dense_layer_1,
dense_layer_2,
prediction_layer
])
```

Step 4: Compile and Fit

```
from tensorflow.keras.callbacks import EarlyStopping


model.compile(
optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'],
)


es = EarlyStopping(monitor='val_accuracy', mode='max', patience=5, restore_best_weights=True)


history = model.fit(train_ds, train_labels, epochs=50, validation_split=0.2, batch_size=32, callbacks = [es])
```

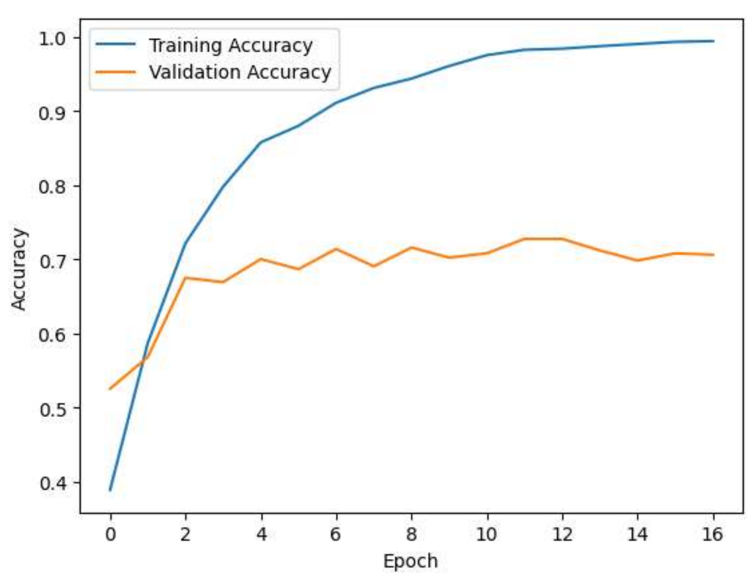
```
Epoch 1/50
65/65 [==============================] - 698s 11s/step - loss: 1.7729 - accuracy: 0.3888 - val_loss: 1.2351 - val_accuracy: 0.5253
Epoch 2/50
65/65 [==============================] - 721s 11s/step - loss: 1.0734 - accuracy: 0.5869 - val_loss: 1.1252 - val_accuracy: 0.5681
Epoch 3/50
65/65 [==============================] - 689s 11s/step - loss: 0.7829 - accuracy: 0.7217 - val_loss: 1.0363 - val_accuracy: 0.6751
Epoch 4/50
65/65 [==============================] - 688s 11s/step - loss: 0.5382 - accuracy: 0.7981 - val_loss: 0.9771 - val_accuracy: 0.6693
Epoch 5/50
65/65 [==============================] - 688s 11s/step - loss: 0.3969 - accuracy: 0.8579 - val_loss: 0.9983 - val_accuracy: 0.7004
Epoch 6/50
65/65 [==============================] - 685s 11s/step - loss: 0.3184 - accuracy: 0.8803 - val_loss: 1.0611 - val_accuracy: 0.6868
Epoch 7/50
65/65 [==============================] - 686s 11s/step - loss: 0.2439 - accuracy: 0.9114 - val_loss: 1.1541 - val_accuracy: 0.7140
Epoch 8/50
65/65 [==============================] - 688s 11s/step - loss: 0.1983 - accuracy: 0.9314 - val_loss: 1.2012 - val_accuracy: 0.6907
Epoch 9/50
65/65 [==============================] - 687s 11s/step - loss: 0.1746 - accuracy: 0.9440 - val_loss: 1.2286 - val_accuracy: 0.7160
Epoch 10/50
65/65 [==============================] - 686s 11s/step - loss: 0.1404 - accuracy: 0.9611 - val_loss: 1.2987 - val_accuracy: 0.7023
Epoch 11/50
65/65 [==============================] - 686s 11s/step - loss: 0.0957 - accuracy: 0.9757 - val_loss: 1.2924 - val_accuracy: 0.7082
Epoch 12/50
65/65 [==============================] - 764s 12s/step - loss: 0.0715 - accuracy: 0.9830 - val_loss: 1.3818 - val_accuracy: 0.7276
Epoch 13/50
65/65 [==============================] - 679s 10s/step - loss: 0.0561 - accuracy: 0.9844 - val_loss: 1.3954 - val_accuracy: 0.7276
Epoch 14/50
65/65 [==============================] - 683s 11s/step - loss: 0.0504 - accuracy: 0.9878 - val_loss: 1.4481 - val_accuracy: 0.7121
Epoch 15/50
65/65 [==============================] - 674s 10s/step - loss: 0.0353 - accuracy: 0.9908 - val_loss: 1.5437 - val_accuracy: 0.6984
Epoch 16/50
65/65 [==============================] - 658s 10s/step - loss: 0.0290 - accuracy: 0.9937 - val_loss: 1.6474 - val_accuracy: 0.7082
Epoch 17/50
65/65 [==============================] - 668s 10s/step - loss: 0.0242 - accuracy: 0.9946 - val_loss: 1.6147 - val_accuracy: 0.7062
```

Exercise 1 :

(i) Plot the training and validation accuracy

(ii) Print Confusion matrix and classification report

```
import matplotlib.pyplot as plt


# Plot the training and validation accuracy curves
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



```
# Evaluate the model on the test set
loss, accuracy = model.evaluate(test_ds, test_labels)
print('Test Loss:', loss)
print('Test Accuracy:', accuracy)
```

```
35/35 [==============================] - 291s 8s/step - loss: 2.0362 - accuracy: 0.6585
Test Loss: 2.0361859798431396
Test Accuracy: 0.6584922671318054
```

```
from sklearn.metrics import confusion_matrix, classification_report
```

```
pred = model.predict(test_ds)
```

```
35/35 [==============================] - 293s 8s/step
```

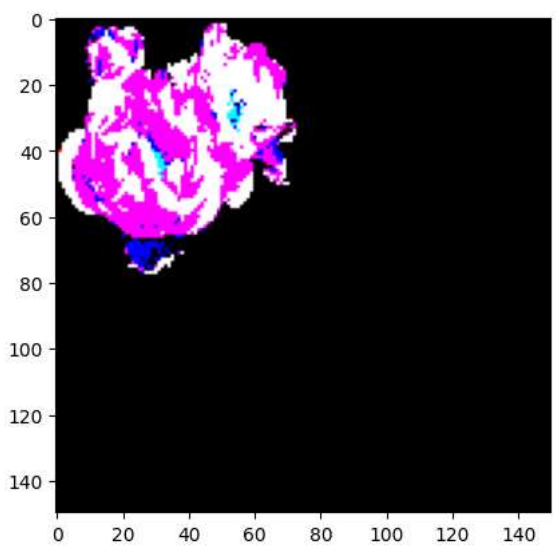
```
# Get the predicted labels for the test set
test_pred = np.argmax(pred, axis=1)
```

```
np.argmax(test_labels, axis = 1)
```

```
array([4, 0, 1, ..., 2, 0, 3])
```

```
index = 235
test_img = test_ds[index]
print("actual class: {}, predicted class: {}".format(test_labels[index].argmax(), test_pred[index].argmax()))
plt.imshow(test_img)
plt.show()
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RG
actual class: 4, predicted class: 0
```




```
# Print the confusion matrix
print('Confusion Matrix:')
print(confusion_matrix(np.argmax(test_labels, axis = 1), test_pred))
```

```
Confusion Matrix:
[[160  46  13  33   7]
 [ 17 146  10   9   6]
 [  6  30 175  15  42]
 [  6  29   9 148  11]
 [  9   7  66   5  96]]
```

```
# Print the classification report
print('Classification Report:')
print(classification_report(np.argmax(test_labels, axis = 1), test_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.62      0.70       259
           1       0.57      0.78      0.65       188
           2       0.64      0.65      0.65       268
```

```
              3       0.70      0.73      0.72       203
              4       0.59      0.52      0.56       183

       accuracy                           0.66      1101
      macro avg       0.66      0.66      0.66      1101
   weighted avg       0.67      0.66      0.66      1101
```

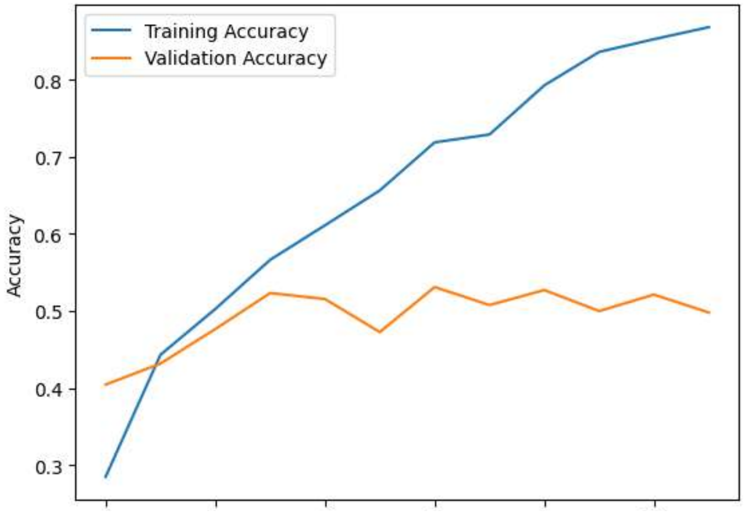
Exercise 2:

Build a custom convnet model . Compare and contrast the results

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras import layers, models


# Define the model architecture
SConvNet = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(128, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(5, activation='softmax')
])


# Compile the model
SConvNet.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])


# Train the model
history = SConvNet.fit(train_ds, train_labels, epochs=50, validation_split=0.2, batch_size=32, callbacks = [es])
```

```
Epoch 1/50
65/65 [==============================] - 117s 2s/step - loss: 14.6343 - accuracy: 0.2852 - val_loss: 1.3566 - val_accuracy: 0.4047
Epoch 2/50
65/65 [==============================] - 111s 2s/step - loss: 1.3344 - accuracy: 0.4433 - val_loss: 1.6632 - val_accuracy: 0.4319
Epoch 3/50
65/65 [==============================] - 111s 2s/step - loss: 1.1883 - accuracy: 0.5027 - val_loss: 1.2270 - val_accuracy: 0.4767
Epoch 4/50
65/65 [==============================] - 111s 2s/step - loss: 1.0742 - accuracy: 0.5664 - val_loss: 1.2794 - val_accuracy: 0.5233
Epoch 5/50
65/65 [==============================] - 115s 2s/step - loss: 0.9489 - accuracy: 0.6112 - val_loss: 1.2622 - val_accuracy: 0.5156
Epoch 6/50
65/65 [==============================] - 118s 2s/step - loss: 0.8721 - accuracy: 0.6564 - val_loss: 1.3045 - val_accuracy: 0.4728
Epoch 7/50
65/65 [==============================] - 117s 2s/step - loss: 0.7436 - accuracy: 0.7187 - val_loss: 1.2978 - val_accuracy: 0.5311
Epoch 8/50
65/65 [==============================] - 119s 2s/step - loss: 0.7511 - accuracy: 0.7290 - val_loss: 1.4214 - val_accuracy: 0.5078
Epoch 9/50
65/65 [==============================] - 115s 2s/step - loss: 0.5532 - accuracy: 0.7927 - val_loss: 1.3570 - val_accuracy: 0.5272
Epoch 10/50
65/65 [==============================] - 118s 2s/step - loss: 0.4388 - accuracy: 0.8360 - val_loss: 1.6672 - val_accuracy: 0.5000
Epoch 11/50
65/65 [==============================] - 119s 2s/step - loss: 0.4030 - accuracy: 0.8526 - val_loss: 1.5623 - val_accuracy: 0.5214
Epoch 12/50
65/65 [==============================] - 111s 2s/step - loss: 0.3782 - accuracy: 0.8681 - val_loss: 1.7283 - val_accuracy: 0.4981
```

```
import matplotlib.pyplot as plt


# Plot the training and validation accuracy curves
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




```
# Evaluate the model on the test set
loss, accuracy = SConvNet.evaluate(test_ds, test_labels)
print('Test Loss:', loss)
print('Test Accuracy:', accuracy)
```

```
35/35 [==============================] - 17s 463ms/step - loss: 1.9294 - accuracy: 0.4550
Test Loss: 1.9293853044509888
Test Accuracy: 0.4550408720970154
```

```
from sklearn.metrics import confusion_matrix, classification_report
```

```
pred = SConvNet.predict(test_ds)
```

```
35/35 [==============================] - 16s 464ms/step
```

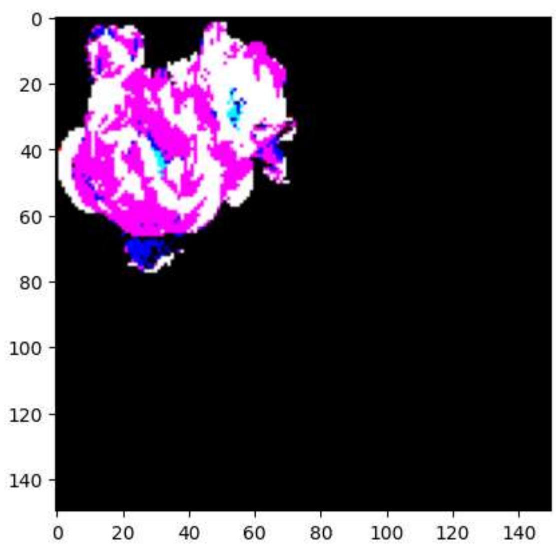
```
# Get the predicted labels for the test set
test_pred = np.argmax(pred, axis=1)
```

```
np.argmax(test_labels, axis = 1)
```

```
array([4, 0, 1, ..., 2, 0, 3])
```

```
index = 235
test_img = test_ds[index]
print("actual class: {}, predicted class: {}".format(test_labels[index].argmax(), test_pred[index].argmax()))
plt.imshow(test_img)
plt.show()
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RG
actual class: 4, predicted class: 0
```



```
# Print the confusion matrix
print('Confusion Matrix:')
print(confusion_matrix(np.argmax(test_labels, axis = 1), test_pred))
```

```
Confusion Matrix:
[[109  58  36  47   9]
 [ 56  82  30  12   8]
 [ 15  30 182  13  28]
 [ 13   3  81 100   6]
 [ 12  25 113   5  28]]
```

```
# Print the classification report
print('Classification Report:')
print(classification_report(np.argmax(test_labels, axis = 1), test_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.53      0.42      0.47       259
           1       0.41      0.44      0.42       188
           2       0.41      0.68      0.51       268
           3       0.56      0.49      0.53       203
           4       0.35      0.15      0.21       183

    accuracy                           0.46      1101
   macro avg       0.46      0.44      0.43      1101
weighted avg       0.46      0.46      0.44      1101
```