In [1]:
```
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"
```

Imports

In [2]:
```
import tensorflow_datasets as tfds
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
```

```
c:\Users\Gaurav\AppData\Local\Programs\Python\Python39\lib\site-packages\tqdm\auto.p
y:22: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See ht
tps://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

Load Data

In [3]:
```
## Loading images and labels
(train_ds, train_labels), (test_ds, test_labels) = tfds.load("tf_flowers",
    split=["train[:70%]", "train[:30%]"], ## Train test split
    batch_size=-1,
    as_supervised=True,  # Include labels
)
```

Image Preprocessing

In [4]:
```
## check existing image size
train_ds[0].shape
```

Out[4]:
```
TensorShape([442, 1024, 3])
```

In [5]:
```
## Resizing images
train_ds = tf.image.resize(train_ds, (150, 150))
test_ds = tf.image.resize(test_ds, (150, 150))
```

In [6]:
```
train_labels
```

Out[6]:
```
<tf.Tensor: shape=(2569,), dtype=int64, numpy=array([2, 3, 3, ..., 0, 2, 0], dtype
=int64)>
```

In [7]:
```
## Transforming labels to correct format
train_labels = to_categorical(train_labels, num_classes=5)
test_labels = to_categorical(test_labels, num_classes=5)
```

In [8]:
```
train_labels[0]
```

Out[8]:
```
array([0., 0., 1., 0., 0.], dtype=float32)
```

Use Pretrained VGG16 Image Classification model

Load a pre-trained CNN model trained on a large dataset

In [9]:
```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
```

In [10]:
```
train_ds[0].shape
```

Out[10]:
```
TensorShape([150, 150, 3])
```

In [11]:
```
## Loading VGG16 model
base_model = VGG16(weights="imagenet", include_top=False, input_shape=train_ds[0].s
```

In [12]:
```
## will not train base mode
# Freeze Parameters in model's lower convolutional layers
base_model.trainable = False
```

In [13]:
```
## Preprocessing input
train_ds = preprocess_input(train_ds)
test_ds = preprocess_input(test_ds)
```

In [14]:
```
## model details
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
_________________________________________________________________
```

Add custom classifier with two dense layers of trainable parameters to model

In [15]:
```
#add our layers on top of this model
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')
```

```python
prediction_layer = layers.Dense(5, activation='softmax')


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

Train classifier layers on training data available for task

In [16]:
```python
from tensorflow.keras.callbacks import EarlyStopping

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

In [17]:
```python
es = EarlyStopping(monitor='val_accuracy', mode='max', patience=5,  restore_best_we
```

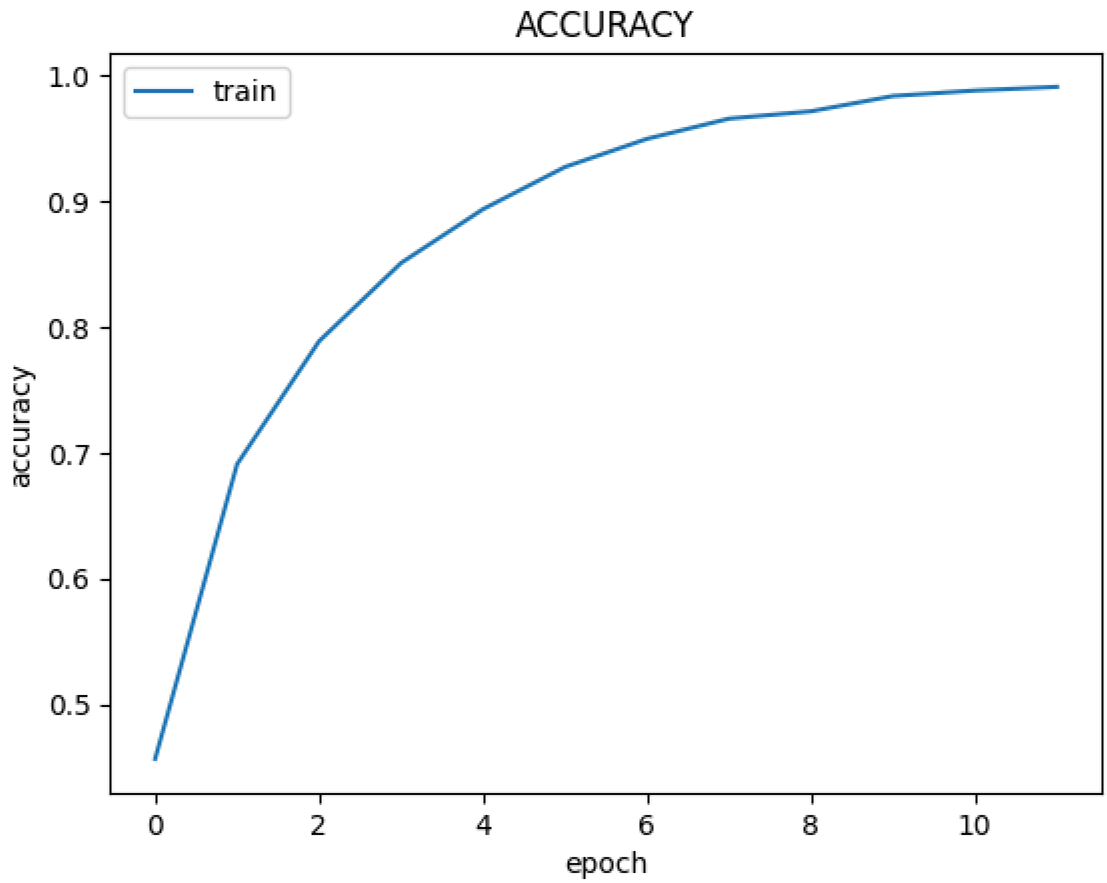
In [18]:
```python
history=model.fit(train_ds, train_labels, epochs=50, validation_split=0.2, batch_si
```

```
Epoch 1/50
65/65 [==============================] - 91s 1s/step - loss: 1.6600 - accuracy: 0.45
64 - val_loss: 1.1794 - val_accuracy: 0.5486
Epoch 2/50
65/65 [==============================] - 88s 1s/step - loss: 0.8261 - accuracy: 0.69
10 - val_loss: 1.0397 - val_accuracy: 0.6226
Epoch 3/50
65/65 [==============================] - 90s 1s/step - loss: 0.5542 - accuracy: 0.78
88 - val_loss: 0.9917 - val_accuracy: 0.6479
Epoch 4/50
65/65 [==============================] - 93s 1s/step - loss: 0.3986 - accuracy: 0.85
11 - val_loss: 0.9414 - val_accuracy: 0.7062
Epoch 5/50
65/65 [==============================] - 95s 1s/step - loss: 0.2973 - accuracy: 0.89
39 - val_loss: 0.9425 - val_accuracy: 0.7043
Epoch 6/50
65/65 [==============================] - 95s 1s/step - loss: 0.2008 - accuracy: 0.92
75 - val_loss: 1.0838 - val_accuracy: 0.7004
Epoch 7/50
65/65 [==============================] - 96s 1s/step - loss: 0.1542 - accuracy: 0.94
99 - val_loss: 1.0646 - val_accuracy: 0.7257
Epoch 8/50
65/65 [==============================] - 98s 2s/step - loss: 0.1176 - accuracy: 0.96
59 - val_loss: 1.0945 - val_accuracy: 0.7237
Epoch 9/50
65/65 [==============================] - 99s 2s/step - loss: 0.0899 - accuracy: 0.97
18 - val_loss: 1.1666 - val_accuracy: 0.7062
Epoch 10/50
65/65 [==============================] - 100s 2s/step - loss: 0.0597 - accuracy: 0.9
839 - val_loss: 1.2617 - val_accuracy: 0.7121
Epoch 11/50
65/65 [==============================] - 100s 2s/step - loss: 0.0428 - accuracy: 0.9
883 - val_loss: 1.3025 - val_accuracy: 0.6984
Epoch 12/50
65/65 [==============================] - 100s 2s/step - loss: 0.0380 - accuracy: 0.9
912 - val_loss: 1.3514 - val_accuracy: 0.7121
```

In [19]:
```
los,accurac=model.evaluate(test_ds,test_labels)
print("Loss: ",los,"Accuracy: ", accurac)
```

```
35/35 [==============================] - 44s 1s/step - loss: 0.1068 - accuracy: 0.96
91
Loss:  0.10678977519273758 Accuracy:  0.9691190123558044
```

In [20]:
```
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.title('ACCURACY')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train'],loc='upper left')
plt.show()
```



In [21]:
```
import numpy as np
import pandas as pd
y_pred = model.predict(test_ds)
y_classes = [np.argmax(element) for element in y_pred]
#to_categorical(y_classes, num_classes=5)
#to_categorical(test_labels, num_classes=5)
print(y_classes[:10])
print("\nTest")
print(test_labels[:10])
```

```
35/35 [==============================] - 44s 1s/step
[2, 3, 3, 4, 3, 0, 0, 0, 0, 1]

Test
[[0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 1. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]]
```