IMPORTS

In [1]:

```
import tensorflow_datasets as tfds
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
```

LOAD DATA

In [2]:

```
## 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
)
```

```
Downloading and preparing dataset Unknown size (download: Unknown size, generate
d: Unknown size, total: Unknown size) to C:\Users\LENOVO\tensorflow_datasets\tf_
flowers\3.0.1...



Dataset tf_flowers downloaded and prepared to C:\Users\LENOVO\tensorflow_dataset
s\tf_flowers\3.0.1. Subsequent calls will reuse this data.
```

IMAGE PREPROCESSING

In [3]:

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

Out[3]:

```
TensorShape([442, 1024, 3])
```

In [4]:

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

In [5]:

```
train_labels
```

Out[5]:

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

In [6]:

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

In [7]:

```
train_labels[0]
```

Out[7]:

```
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 [8]:

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

In [9]:

```
train_ds[0].shape
```

Out[9]:

```
TensorShape([150, 150, 3])
```

In [10]:

```
## Loading VGG16 model
base_model = VGG16(weights="imagenet", include_top=False, input_shape=train_ds[0
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
ns/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 10s 0us/step
58900480/58889256 [==============================] - 10s 0us/step
```

In [11]:

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

In [12]:

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

In [13]:

```
## 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 [14]:

```
#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')
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 [15]:

```
from tensorflow.keras.callbacks import EarlyStopping

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

In [16]:

```
es = EarlyStopping(monitor='val_accuracy', mode='max', patience=5,  restore_best
```

In [17]:

```
history=model.fit(train_ds, train_labels, epochs=50, validation_split=0.2, batch
```

```
Epoch 1/50
65/65 [==============================] - 245s 4s/step - loss: 2.0215 - accuracy:
0.4891 - val_loss: 1.1629 - val_accuracy: 0.5759
Epoch 2/50
65/65 [==============================] - 246s 4s/step - loss: 0.7979 - accuracy:
0.7202 - val_loss: 1.0837 - val_accuracy: 0.6479
Epoch 3/50
65/65 [==============================] - 248s 4s/step - loss: 0.5177 - accuracy:
0.8209 - val_loss: 1.0236 - val_accuracy: 0.6634
Epoch 4/50
65/65 [==============================] - 248s 4s/step - loss: 0.3281 - accuracy:
0.8881 - val_loss: 0.9744 - val_accuracy: 0.7004
Epoch 5/50
65/65 [==============================] - 249s 4s/step - loss: 0.2454 - accuracy:
0.9197 - val_loss: 1.0080 - val_accuracy: 0.7043
Epoch 6/50
65/65 [==============================] - 250s 4s/step - loss: 0.1795 - accuracy:
0.9499 - val_loss: 1.2187 - val_accuracy: 0.7004
Epoch 7/50
65/65 [==============================] - 251s 4s/step - loss: 0.1394 - accuracy:
0.9591 - val_loss: 1.2054 - val_accuracy: 0.7062
Epoch 8/50
65/65 [==============================] - 806s 13s/step - loss: 0.0943 - accurac
y: 0.9747 - val_loss: 1.1620 - val_accuracy: 0.7004
Epoch 9/50
65/65 [==============================] - 247s 4s/step - loss: 0.0677 - accuracy:
0.9839 - val_loss: 1.1926 - val_accuracy: 0.7101
Epoch 10/50
65/65 [==============================] - 249s 4s/step - loss: 0.0455 - accuracy:
0.9912 - val_loss: 1.2542 - val_accuracy: 0.7140
Epoch 11/50
65/65 [==============================] - 250s 4s/step - loss: 0.0340 - accuracy:
0.9942 - val_loss: 1.2863 - val_accuracy: 0.7257
Epoch 12/50
65/65 [==============================] - 248s 4s/step - loss: 0.0285 - accuracy:
0.9937 - val_loss: 1.3212 - val_accuracy: 0.7140
Epoch 13/50
65/65 [==============================] - 246s 4s/step - loss: 0.0230 - accuracy:
0.9951 - val_loss: 1.3357 - val_accuracy: 0.7276
Epoch 14/50
65/65 [==============================] - 263s 4s/step - loss: 0.0227 - accuracy:
0.9946 - val_loss: 1.3657 - val_accuracy: 0.7335
Epoch 15/50
65/65 [==============================] - 276s 4s/step - loss: 0.0179 - accuracy:
0.9956 - val_loss: 1.4143 - val_accuracy: 0.7218
Epoch 16/50
65/65 [==============================] - 278s 4s/step - loss: 0.0177 - accuracy:
0.9961 - val_loss: 1.4197 - val_accuracy: 0.7276
Epoch 17/50
65/65 [==============================] - 256s 4s/step - loss: 0.0124 - accuracy:
0.9976 - val_loss: 1.4324 - val_accuracy: 0.7315
Epoch 18/50
65/65 [==============================] - 249s 4s/step - loss: 0.0102 - accuracy:
0.9976 - val_loss: 1.4354 - val_accuracy: 0.7237
Epoch 19/50
65/65 [==============================] - 249s 4s/step - loss: 0.0080 - accuracy:
0.9976 - val_loss: 1.4546 - val_accuracy: 0.7296
```

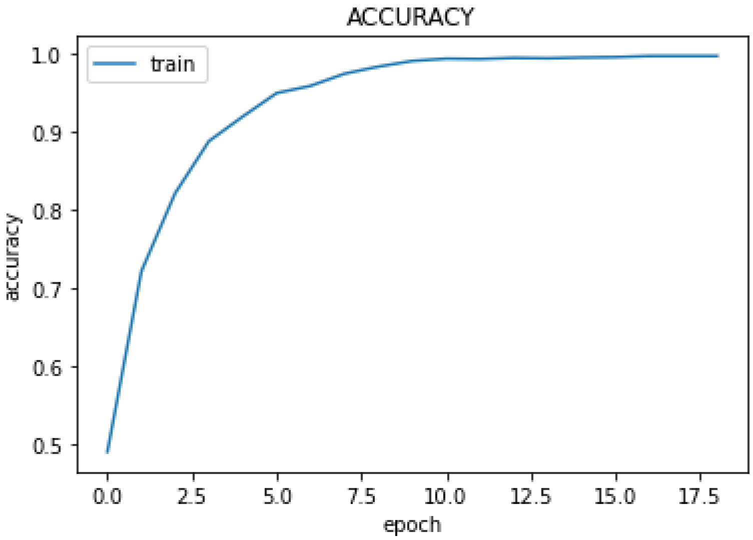
In [18]:

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

```
35/35 [==============================] - 104s 3s/step - loss: 0.0228 - accuracy: 
0.9936
Loss:  0.022801034152507782 Accuracy:  0.9936421513557434
```

In [19]:

```
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 [20]:

```
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])
```

```
[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.]]
```

In [ ]: