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

```
## 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
)
```

```
WARNING:absl:Variant folder /root/tensorflow_datasets/tf_flowers/3.0.1 has no dataset_info.json
Downloading and preparing dataset Unknown size (download: Unknown size, generated: Unknown size, total: Unknown size) to /ro
Dl Completed...: 100%     1/1  [00:08<00:00,   8.59s/ url]

Dl Size...: 100%     218/218  [00:08<00:00,  29.75  MiB/s]
Dataset tf_flowers downloaded and prepared to /root/tensorflow_datasets/tf_flowers/3.0.1. Subsequent calls will reuse this d
```

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

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

```
## Resizing images
train_ds = tf.image.resize(train_ds, (150, 150))
test_ds = tf.image.resize(test_ds, (150, 150))
train_ds[0].shape
```

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

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

```
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)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_ke
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
```

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

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

```
## model details
base_model.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer_7 (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1,792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36,928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73,856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147,584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295,168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590,080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590,080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0

Total params: 14,714,688 (56.13 MB)
Trainable params: 0 (0.00 B)
Non-trainable params: 14,714,688 (56.13 MB)

```
#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
])
```

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

```
history=model.fit(train_ds, train_labels, epochs=10, validation_split=0.2, batch_size=32)
```

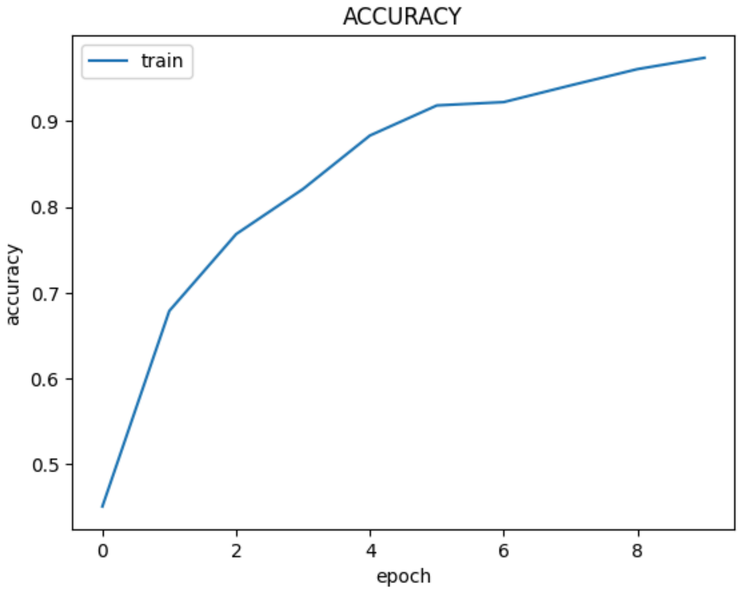
```
Epoch 1/10
65/65 ━━━━━━━━━━━━━━━━━━━━ 664s 10s/step - accuracy: 0.3738 - loss: 2.7304 - val_accuracy: 0.5720 - val_loss: 1.1737
Epoch 2/10
65/65 ━━━━━━━━━━━━━━━━━━━━ 668s 10s/step - accuracy: 0.6704 - loss: 0.8873 - val_accuracy: 0.6381 - val_loss: 1.0835
Epoch 3/10
65/65 ━━━━━━━━━━━━━━━━━━━━ 646s 10s/step - accuracy: 0.7781 - loss: 0.6040 - val_accuracy: 0.6634 - val_loss: 1.0166
Epoch 4/10
65/65 ━━━━━━━━━━━━━━━━━━━━ 650s 10s/step - accuracy: 0.8262 - loss: 0.4386 - val_accuracy: 0.6868 - val_loss: 0.9902
Epoch 5/10
65/65 ━━━━━━━━━━━━━━━━━━━━ 653s 10s/step - accuracy: 0.8881 - loss: 0.2982 - val_accuracy: 0.6926 - val_loss: 1.0807
Epoch 6/10
65/65 ━━━━━━━━━━━━━━━━━━━━ 676s 10s/step - accuracy: 0.9193 - loss: 0.2474 - val_accuracy: 0.6946 - val_loss: 1.0494
Epoch 7/10
65/65 ━━━━━━━━━━━━━━━━━━━━ 685s 10s/step - accuracy: 0.9223 - loss: 0.2063 - val_accuracy: 0.6946 - val_loss: 1.1531
Epoch 8/10
65/65 ━━━━━━━━━━━━━━━━━━━━ 695s 10s/step - accuracy: 0.9328 - loss: 0.1853 - val_accuracy: 0.7179 - val_loss: 1.1633
Epoch 9/10
```

```
65/65 ━━━━━━━━━━━━━━━━━━━━ 650s 10s/step - accuracy: 0.9633 - loss: 0.1128 - val_accuracy: 0.7257 - val_loss: 1.1242
Epoch 10/10
65/65 ━━━━━━━━━━━━━━━━━━━━ 679s 10s/step - accuracy: 0.9725 - loss: 0.0828 - val_accuracy: 0.7101 - val_loss: 1.2591
```

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

```
35/35 ━━━━━━━━━━━━━━━━━━━━ 285s 8s/step - accuracy: 0.9832 - loss: 0.0681
Loss:  0.0677466094493866 Accuracy:  0.9809264540672302
```

```
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()
```



```
Start coding or generate with AI.
```