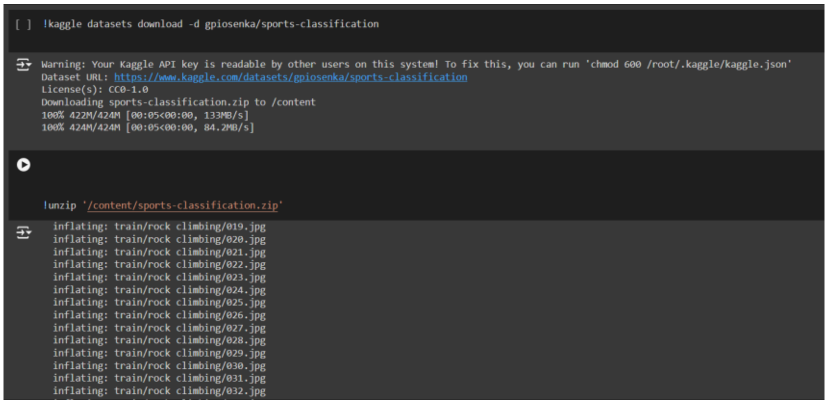
Data Collection and Preprocessing Phase

Date	10 June 2024
Team ID	SWTID1720158677
Project Title	SportSpecs: Unraveling Athletic Prowess With Advanced Transfer Learning For Sports.
Maximum Marks	6 Marks

Preprocessing Template

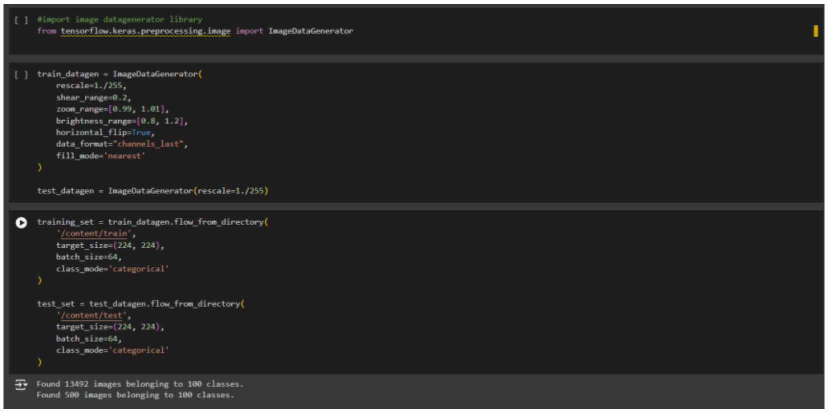
The images will be preprocessed by resizing, normalizing, augmenting, denoising, adjusting contrast, detecting edges, converting color space, cropping, batch normalizing, and whitening data. These steps will enhance data quality, promote model generalization, and improve convergence during neural network training, ensuring robust and efficient performance across various computer vision tasks.

Section	**Description**
Data Overview	Provide an overview of the dataset, which includes labeled images from seven different sports classes: cricket, wrestling, tennis, badminton, soccer, swimming, and karate. The dataset is used to train a deep learning model for classifying sports activities.
Resizing	Resize images to a specified target size (e.g., 150x150 pixels) to ensure uniformity and compatibility with the pre-trained model used for transfer learning.
Normalization	Normalize pixel values to a specific range (e.g., 0 to 1) to standardize the input data and improve the model's performance.
Data Augmentation	Apply augmentation techniques such as flipping, rotation, shifting, zooming, or shearing to increase the diversity of the training data and prevent overfitting.

<table>
<tr><td>Denoising</td><td>Apply denoising filters to reduce noise in the images, enhancing the quality of the input data for better model accuracy.</td></tr>
<tr><td>Edge Detection</td><td>Apply edge detection algorithms to highlight prominent edges in the images, which can help in identifying key features relevant to different sports activities.</td></tr>
<tr><td>Color Space Conversion</td><td>Convert images from one color space to another (e.g., RGB to grayscale) if necessary, to simplify the data and focus on essential features.</td></tr>
<tr><td>Image Cropping</td><td>Crop images to focus on the regions containing objects of interest, ensuring that the model learns from the most relevant parts of the images.</td></tr>
<tr><td>Batch Normalization</td><td>Apply batch normalization to the input of each layer in the neural network to stabilize and accelerate the training process. Apply batch normalization to the input of each layer in the neural network to stabilize and accelerate the training process.</td></tr>
<tr><td colspan="2"><b>Data Preprocessing Code Screenshots</b></td></tr>
<tr><td>Loading Data</td><td>
<pre>
[ ] !kaggle datasets download -d gpiosenka/sports-classification

Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Dataset URL: https://www.kaggle.com/datasets/gpiosenka/sports-classification
License(s): CC0-1.0
Downloading sports-classification.zip to /content
100% 422M/424M [00:05<00:00, 133MB/s]
100% 424M/424M [00:05<00:00, 84.2MB/s]

!unzip '/content/sports-classification.zip'

  inflating: train/rock climbing/019.jpg
  inflating: train/rock climbing/020.jpg
  inflating: train/rock climbing/021.jpg
  inflating: train/rock climbing/022.jpg
  inflating: train/rock climbing/023.jpg
  inflating: train/rock climbing/024.jpg
  inflating: train/rock climbing/025.jpg
  inflating: train/rock climbing/026.jpg
  inflating: train/rock climbing/027.jpg
  inflating: train/rock climbing/028.jpg
  inflating: train/rock climbing/029.jpg
  inflating: train/rock climbing/030.jpg
  inflating: train/rock climbing/031.jpg
  inflating: train/rock climbing/032.jpg
</pre></td></tr>
<tr><td>Resizing</td><td>
<pre>
[ ] #import image datagenerator library
    from tensorflow.keras.preprocessing.image import ImageDataGenerator

[ ] train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=[0.99, 1.01],
        brightness_range=[0.8, 1.2],
        horizontal_flip=True,
        data_format="channels_last",
        fill_mode='nearest'
)

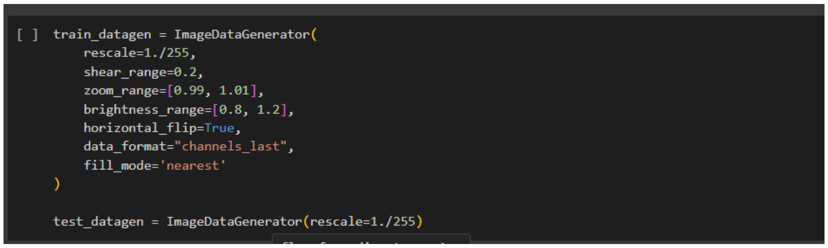
    test_datagen = ImageDataGenerator(rescale=1./255)

    training_set = train_datagen.flow_from_directory(
        '/content/train',
        target_size=(224, 224),
        batch_size=64,
        class_mode='categorical'
)

    test_set = test_datagen.flow_from_directory(
        '/content/test',
        target_size=(224, 224),
        batch_size=64,
        class_mode='categorical'
)

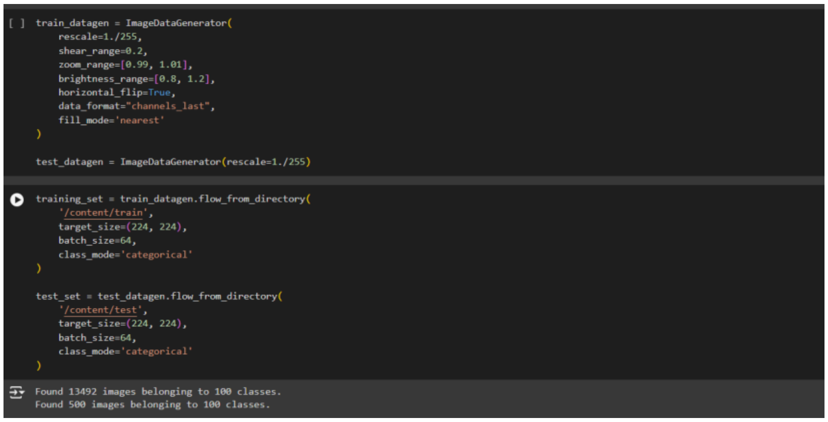
    Found 13492 images belonging to 100 classes.
    Found 500 images belonging to 100 classes.
</pre></td></tr>
</table>

<table>
<tr>
<td>Normalization</td>
<td>



```
[ ]  train_datagen = ImageDataGenerator(
         rescale=1./255,
         shear_range=0.2,
         zoom_range=[0.99, 1.01],
         brightness_range=[0.8, 1.2],
         horizontal_flip=True,
         data_format="channels_last",
         fill_mode='nearest'
     )

     test_datagen = ImageDataGenerator(rescale=1./255)
```

</td>
</tr>
<tr>
<td>Data Augmentation</td>
<td>

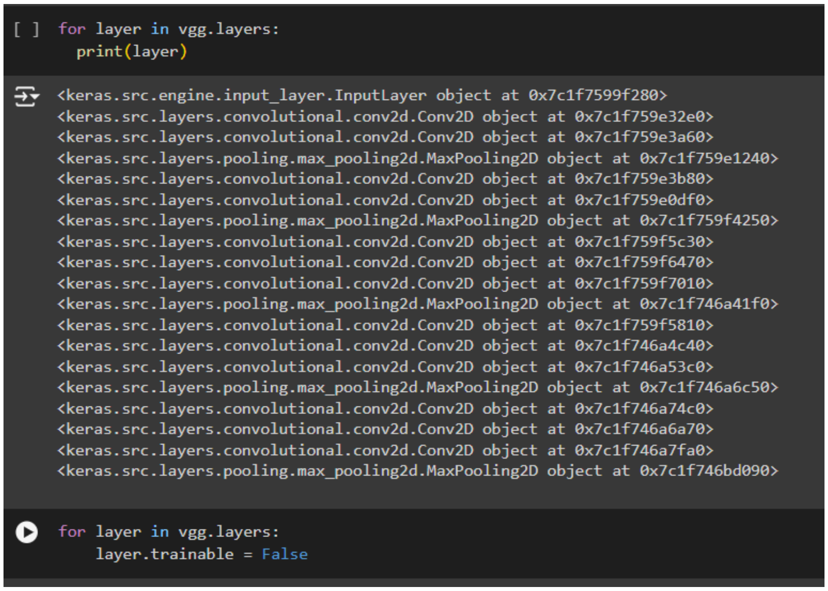


```
[ ]  train_datagen = ImageDataGenerator(
         rescale=1./255,
         shear_range=0.2,
         zoom_range=[0.99, 1.01],
         brightness_range=[0.8, 1.2],
         horizontal_flip=True,
         data_format="channels_last",
         fill_mode='nearest'
     )

     test_datagen = ImageDataGenerator(rescale=1./255)
```

```
     training_set = train_datagen.flow_from_directory(
         '/content/train',
         target_size=(224, 224),
         batch_size=64,
         class_mode='categorical'
     )

     test_set = test_datagen.flow_from_directory(
         '/content/test',
         target_size=(224, 224),
         batch_size=64,
         class_mode='categorical'
     )
```

```
Found 13492 images belonging to 100 classes.
Found 500 images belonging to 100 classes.
```

</td>
</tr>
<tr>
<td>Denoising</td>
<td>



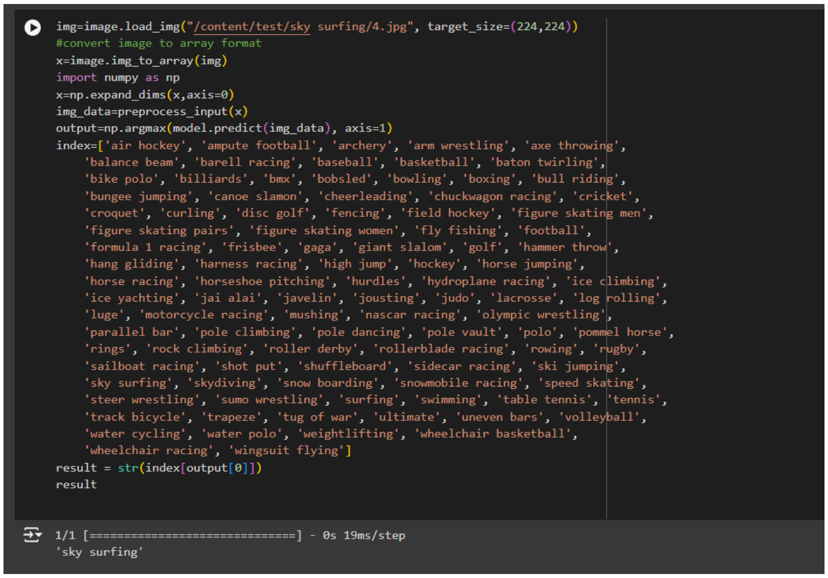
```
[ ]  for layer in vgg.layers:
       print(layer)
```

```
<keras.src.engine.input_layer.InputLayer object at 0x7c1f7599f280>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7c1f759e32e0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7c1f759e3a60>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7c1f759e1240>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7c1f759e3b80>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7c1f759e0df0>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7c1f759f4250>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7c1f759f5c30>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7c1f759f6470>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7c1f759f7010>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7c1f746a41f0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7c1f759f5810>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7c1f746a4c40>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7c1f746a53c0>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7c1f746a6c50>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7c1f746a74c0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7c1f746a6a70>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7c1f746a7fa0>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7c1f746bd090>
```

```
     for layer in vgg.layers:
         layer.trainable = False
```

</td>
</tr>
</table>

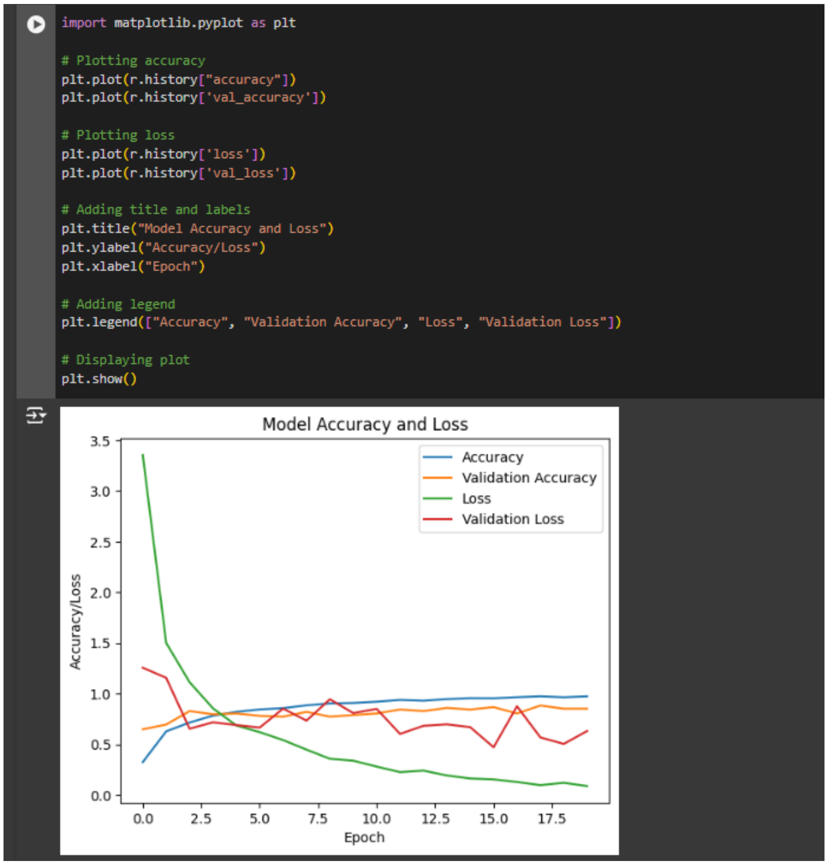
<table>
<tr>
<td>Edge Detection</td>
<td>



```
img=image.load_img("/content/test/sky surfing/4.jpg", target_size=(224,224))
#convert image to array format
x=image.img_to_array(img)
import numpy as np
x=np.expand_dims(x,axis=0)
img_data=preprocess_input(x)
output=np.argmax(model.predict(img_data), axis=1)
index=['air hockey', 'ampute football', 'archery', 'arm wrestling', 'axe throwing',
    'balance beam', 'barell racing', 'baseball', 'basketball', 'baton twirling',
    'bike polo', 'billiards', 'bmx', 'bobsled', 'bowling', 'boxing', 'bull riding',
    'bungee jumping', 'canoe slamon', 'cheerleading', 'chuckwagon racing', 'cricket',
    'croquet', 'curling', 'disc golf', 'fencing', 'field hockey', 'figure skating men',
    'figure skating pairs', 'figure skating women', 'fly fishing', 'football',
    'formula 1 racing', 'frisbee', 'gaga', 'giant slalom', 'golf', 'hammer throw',
    'hang gliding', 'harness racing', 'high jump', 'hockey', 'horse jumping',
    'horse racing', 'horseshoe pitching', 'hurdles', 'hydroplane racing', 'ice climbing',
    'ice yachting', 'jai alai', 'javelin', 'jousting', 'judo', 'lacrosse', 'log rolling',
    'luge', 'motorcycle racing', 'mushing', 'nascar racing', 'olympic wrestling',
    'parallel bar', 'pole climbing', 'pole dancing', 'pole vault', 'polo', 'pommel horse',
    'rings', 'rock climbing', 'roller derby', 'rollerblade racing', 'rowing', 'rugby',
    'sailboat racing', 'shot put', 'shuffleboard', 'sidecar racing', 'ski jumping',
    'sky surfing', 'skydiving', 'snow boarding', 'snowmobile racing', 'speed skating',
    'steer wrestling', 'sumo wrestling', 'surfing', 'swimming', 'table tennis', 'tennis',
    'track bicycle', 'trapeze', 'tug of war', 'ultimate', 'uneven bars', 'volleyball',
    'water cycling', 'water polo', 'weightlifting', 'wheelchair basketball',
    'wheelchair racing', 'wingsuit flying']
result = str(index[output[0]])
result
```

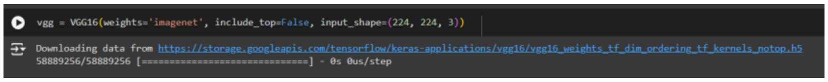
```
1/1 [==============================] - 0s 19ms/step
'sky surfing'
```

</td>
</tr>
<tr>
<td>Color Space Conversion</td>
<td>



```
import matplotlib.pyplot as plt

# Plotting accuracy
plt.plot(r.history["accuracy"])
plt.plot(r.history['val_accuracy'])

# Plotting loss
plt.plot(r.history['loss'])
plt.plot(r.history['val_loss'])

# Adding title and labels
plt.title("Model Accuracy and Loss")
plt.ylabel("Accuracy/Loss")
plt.xlabel("Epoch")

# Adding legend
plt.legend(["Accuracy", "Validation Accuracy", "Loss", "Validation Loss"])

# Displaying plot
plt.show()
```

</td>
</tr>
<tr>
<td>Image Cropping</td>
<td>

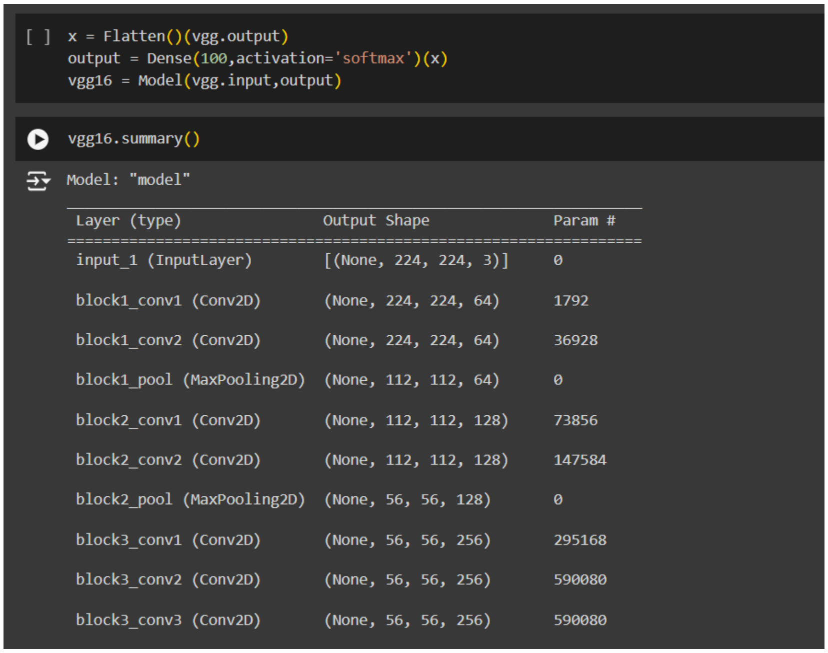


```
vgg = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

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

</td>
</tr>
</table>

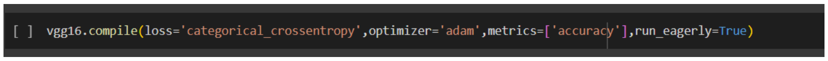
Batch Normalization



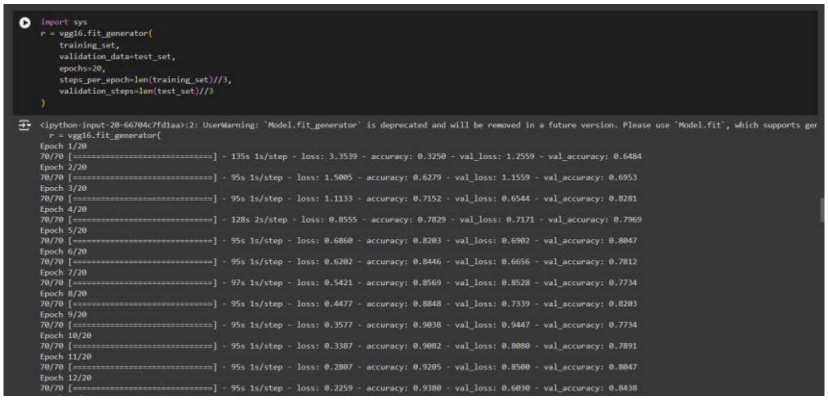
```
[ ] x = Flatten()(vgg.output)
    output = Dense(100,activation='softmax')(x)
    vgg16 = Model(vgg.input,output)
```

```
vgg16.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (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
```



```
[ ] vgg16.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'],run_eagerly=True)
```



```
import sys
r = vgg16.fit_generator(
    training_set,
    validation_data=test_set,
    epochs=20,
    steps_per_epoch=len(training_set)//3,
    validation_steps=len(test_set)//3
)
```

```
<ipython-input-20-66704c7fd1aa>:2: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports ger
  r = vgg16.fit_generator(
Epoch 1/20
70/70 [==============================] - 135s 1s/step - loss: 3.3539 - accuracy: 0.3250 - val_loss: 1.2559 - val_accuracy: 0.6484
Epoch 2/20
70/70 [==============================] - 95s 1s/step - loss: 1.5005 - accuracy: 0.6279 - val_loss: 1.1559 - val_accuracy: 0.6953
Epoch 3/20
70/70 [==============================] - 95s 1s/step - loss: 1.1133 - accuracy: 0.7152 - val_loss: 0.6544 - val_accuracy: 0.8281
Epoch 4/20
70/70 [==============================] - 128s 2s/step - loss: 0.8555 - accuracy: 0.7829 - val_loss: 0.7171 - val_accuracy: 0.7969
Epoch 5/20
70/70 [==============================] - 95s 1s/step - loss: 0.6860 - accuracy: 0.8203 - val_loss: 0.6902 - val_accuracy: 0.8047
Epoch 6/20
70/70 [==============================] - 95s 1s/step - loss: 0.6202 - accuracy: 0.8446 - val_loss: 0.6656 - val_accuracy: 0.7812
Epoch 7/20
70/70 [==============================] - 97s 1s/step - loss: 0.5421 - accuracy: 0.8569 - val_loss: 0.8528 - val_accuracy: 0.7734
Epoch 8/20
70/70 [==============================] - 95s 1s/step - loss: 0.4477 - accuracy: 0.8848 - val_loss: 0.7339 - val_accuracy: 0.8203
Epoch 9/20
70/70 [==============================] - 95s 1s/step - loss: 0.3577 - accuracy: 0.9038 - val_loss: 0.9447 - val_accuracy: 0.7734
Epoch 10/20
70/70 [==============================] - 95s 1s/step - loss: 0.3387 - accuracy: 0.9082 - val_loss: 0.8080 - val_accuracy: 0.7891
Epoch 11/20
70/70 [==============================] - 95s 1s/step - loss: 0.2807 - accuracy: 0.9205 - val_loss: 0.8500 - val_accuracy: 0.8047
Epoch 12/20
70/70 [==============================] - 95s 1s/step - loss: 0.2259 - accuracy: 0.9380 - val_loss: 0.6030 - val_accuracy: 0.8438
```