Transfer Learning Assignment

Download all the data in this rar_file , it contains all the data required for the assignment. When you unrar the file you'll get the files in the following format: **path/to/the/image.tif,category**

```
where the categories are numbered 0 to 15, in the following order:
```

```
0 letter
1 form
2 email
3 handwritten
4 advertisement
5 scientific report
6 scientific publication
7 specification
8 file folder
9 news article
10 budget
```

Saving...

```
13 questionnaire
14 resume
15 memo
```

There is a file named as 'labels_final.csv' , it consists of two columns. First column is path which is the required path to the images and second is the class label.

```
#the dataset that you are dealing with is quite large 3.7 GB and hence there are two methods to import the data to Colab
# Method 1- you can use gdown module to get the data directly from Google drive to Colab
# the syntax is as follows !gdown --id file_id , for ex - running the below cell will import the rvl-cdip.rar dataset

```

```
#!pip install gdown
```

```
#!gdown --id 1Z4TyI7FcFVEx8qdl4jO9qxvxaqLSqoEu
```

```
# Method -2 you can also import the data using wget function
#https://www.youtube.com/watch?v=BPUfVq7RaY8

```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#unrar the file
#get_ipython().system_raw("unrar x rvl-cdip.rar")
#!winrar x 'rvl-cdip.rar'
```

```
!unrar x 'drive/MyDrive/Transfer-Learning-25/rvl-cdip.rar'
```

Saving...

```
Creating    data_final/imagesz/z/z/y/zzy95c00                        OK
Extracting  data_final/imagesz/z/z/y/zzy95c00/2072568903a.tif        OK
Creating    data_final/imagesz/z/z/z                                 OK
Creating    data_final/imagesz/z/z/z/zzz01f00                        OK
Extracting  data_final/imagesz/z/z/z/zzz01f00/0001252956.tif         OK
Creating    data_final/imagesz/z/z/z/zzz46d00                        OK
Extracting  data_final/imagesz/z/z/z/zzz46d00/40015701-5701.tif      OK
Creating    data_final/imagesz/z/z/z/zzz97c00                        OK
Extracting  data_final/imagesz/z/z/z/zzz97c00/527802957+-2958.tif    OK
All OK
```

```
!pip install patool
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting patool
  Downloading patool-1.12-py2.py3-none-any.whl (77 kB)
     |████████████████████████████████| 77 kB 7.4 MB/s
Installing collected packages: patool
Successfully installed patool-1.12
```

```
import patoolib
patoolib.extract_archive("drive/MyDrive/Transfer-Learning-25/rvl-cdip.rar")
```

```
patool: Extracting drive/MyDrive/Transfer-Learning-25/rvl-cdip.rar ...
patool: running /usr/bin/unrar x -- /content/drive/MyDrive/Transfer-Learning-25/rvl-cdip.rar
patool:     with cwd='./Unpack_jx5_4zfv'
patool: ... drive/MyDrive/Transfer-Learning-25/rvl-cdip.rar extracted to `rvl-cdip' (multiple files in root).
```

Saving...

```
#!pip install opencv-python
#!pip install matplotlib
```

2. On this image data, you have to train 3 types of models as given below You have to split the data into Train and Validation data.

```
#import all the required libraries
import os
import numpy as np
import pandas as pd
import cv2
from tqdm import tqdm
from tensorflow.keras import models, layers
from tensorflow.keras.models import Model
from tensorflow.keras.layers import BatchNormalization, Activation, Flatten
from tensorflow.keras.optimizers import Adam
import random as rn
from numpy import expand_dims
import numpy as np
import tensorflow as tf
import keras
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras import regularizers
from matplotlib import pyplot
from tensorflow.keras.layers import Dense,Input,Conv2D,MaxPooling2D,Flatten,Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import TensorBoard
import datetime
from tensorflow.keras.callbacks import ModelCheckpoint
import warnings
warnings.filterwarnings("ignore")
```

```
data=pd.read_csv('drive/MyDrive/Transfer-Learning-25/labels_final.csv',dtype=str)
data.head(5)
```

	path	label
0	imagesv/v/o/h/voh71d00/509132755+-2755.tif	3
1	imagesl/l/x/t/lxt19d00/502213303.tif	3
2	imagesx/x/e/d/xed05a00/2075325674.tif	2
3	imageso/o/j/b/ojb60d00/517511301+-1301.tif	3
4	imagesq/q/z/k/qzk17e00/2031320195.tif	7

Saving... ×

```
0     3016
13    3007
14    3006
12    3006
3     3005
8     3003
10    3002
9     3002
7     3000
5     2999
15    2996
4     2994
1     2994
2     2993
11    2992
6     2985
Name: label, dtype: int64
```

```
data.shape
```

```
(48000, 2)
```

```
#Due RAM issues currently I am training the model on 20k data points
#data = data.sample(n = 20000)
```

```
dir_path='rvl-cdip/data_final'
```

```
#!unzip animals10.zip -d "/content/drive/MyDrive/Data_science/animals10"
```

3. Try not to load all the images into memory, use the gernarators that we have given the reference notebooks to load the batch of images only during the train data. or you can use this method also https://medium.com/@vijayabhaskar96/tutorial-on-keras-imagedatagenerator-with-flow-from-dataframe-8bd5776e45c1

https://medium.com/@vijayabhaskar96/tutorial-on-keras-flow-from-dataframe-1fd4493d237c

Note- In the reference notebook you were dealing with jpg images, in the given dataset you are dealing with tiff images. Imagedatagenrator works with both type of images. If you want to use custom data pipeline then you have to convert your tiff images to jpg images.

4. You are free to choose Learning rate, optimizer, loss function, image augmentation, any hyperparameters. but you have to use the same architechture what we are asking below.
5. Use tensorboard for every model and analyse your gradients. (you need to upload the screenshots for each model for evaluation)
6. You can check about Transfer Learning in this link - https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

https://www.appliedaicourse.com/lecture/11/applied-machine-learning-online-course/3426/code-example-cats-vs-dogs/8/module-8-neural-networks-computer-vision-and-deep-learning

Saving... ×

aw model_plots for each of the model.

Model-1

```
work without Fully Connected layers and initilize all the weights with Imagenet trained weights.
out FC layers, add a new Conv block ( 1 Conv layer and 1 Maxpooling ), 2 FC layers and an output layer to classify 16 classes. You ar
e INPUT --> VGG-16 without Top layers(FC) --> Conv Layer --> Maxpool Layer --> 2 FC layers --> Output Layer
plot the architecture of the model.

, FC layers, output layer. Don't train the VGG-16 network.
```

```
# import VGG16 model
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.models import Model
```

```
imageflow = ImageDataGenerator( rescale=1./255.,validation_split=0.25,rotation_range=40,width_shift_range=0.3,
                                height_shift_range=0.2,shear_range=0.2,zoom_range=0.2,horizontal_flip=True,)

train_generator=imageflow.flow_from_dataframe(dataframe=data,directory=dir_path,x_col="path",y_col="label",subset="training",
batch_size=32,seed=42,shuffle=True,class_mode='categorical',target_size=(224,224))
```

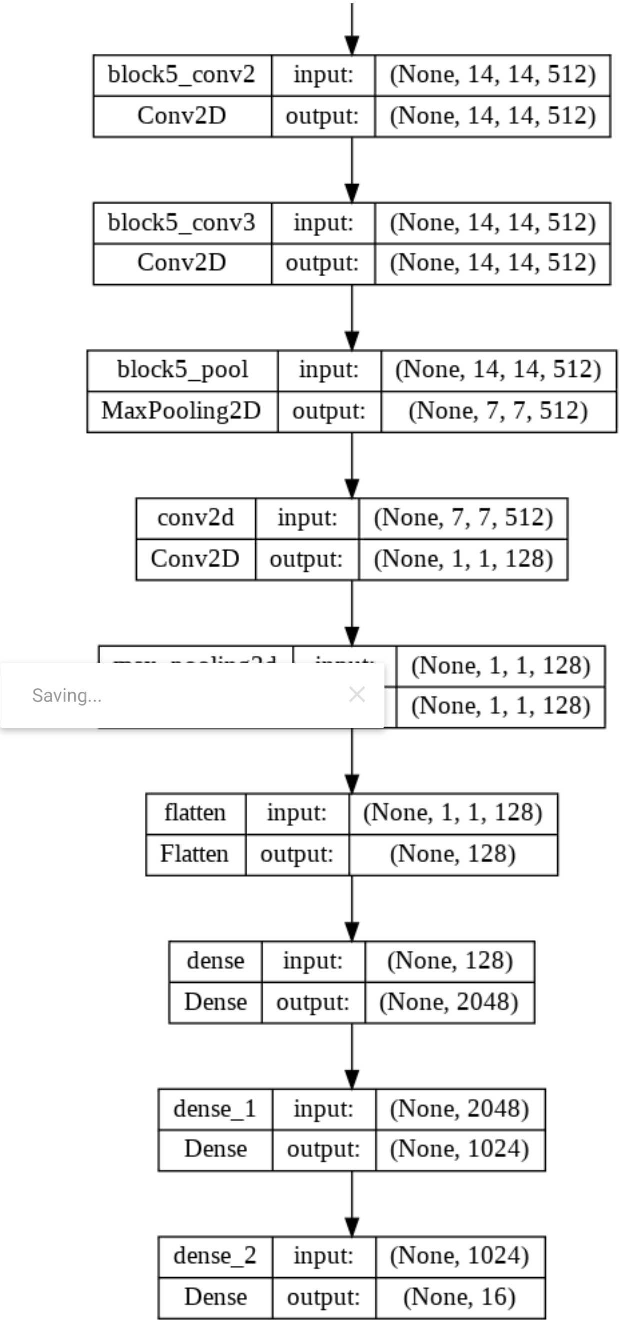
```

valid_generator=imageflow.flow_from_dataframe(dataframe=data,directory=dir_path,x_col="path",y_col="label",subset="validation",
batch_size=32,seed=42,shuffle=True,class_mode='categorical',target_size=(224,224))

```

```
Found 36000 validated image filenames belonging to 16 classes.
Found 12000 validated image filenames belonging to 16 classes.
```

```
tf.keras.backend.clear_session()

## Set the random seed values to regenerate the model.
np.random.seed(0)
rn.seed(0)

## loading vgg16 model without FC layers
vgg16 = VGG16(weights='imagenet',include_top=False,input_shape = (224,224, 3))

## freezing the model  / Not trainable weights
vgg16.trainable = False

vgg16_output = vgg16.output
conv_layer = Conv2D(filters=128, kernel_size=7, activation='relu',padding='valid')(vgg16_output)
maxpool_layer = MaxPooling2D(pool_size=1,strides=2)(conv_layer)
flatten = Flatten()(maxpool_layer)
FC_1 = Dense(2048,activation='relu')(flatten)
FC_2 = Dense(1024,activation='relu')(FC_1)
  Saving...          n='softmax')(FC_2)

final_model=Model(inputs = vgg16.input, outputs = output_layer)
final_model.summary()

```

```
 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    
                                                                 
 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0         
 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 conv2d (Conv2D)             (None, 1, 1, 128)         3211392

 max_pooling2d (MaxPooling2D  (None, 1, 1, 128)        0
 )

 flatten (Flatten)           (None, 128)               0

 dense (Dense)               (None, 2048)              264192

 dense_1 (Dense)             (None, 1024)              2098176

 dense_2 (Dense)             (None, 16)                16400

=================================================================
Total params: 20,304,848
  Saving...                              8
_________________________________________________________________
```

```
plot_model(final_model,'image.png',show_shapes=True)
```


block5_conv2 | input: | (None, 14, 14, 512)
Conv2D | output: | (None, 14, 14, 512)
block5_conv3 | input: | (None, 14, 14, 512)
Conv2D | output: | (None, 14, 14, 512)
block5_pool | input: | (None, 14, 14, 512)
MaxPooling2D | output: | (None, 7, 7, 512)
conv2d | input: | (None, 7, 7, 512)
Conv2D | output: | (None, 1, 1, 128)
max_pooling2d | input: | (None, 1, 1, 128)
(None, 1, 1, 128)
Saving...
flatten | input: | (None, 1, 1, 128)
Flatten | output: | (None, 128)
dense | input: | (None, 128)
Dense | output: | (None, 2048)
dense_1 | input: | (None, 2048)
Dense | output: | (None, 1024)
dense_2 | input: | (None, 1024)
Dense | output: | (None, 16)

```
!rm -rf ./model1_logs/
```

```
log_dir="model1_logs/fit/"+ datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard=TensorBoard(log_dir=log_dir,histogram_freq=1)

filepath="best_model_1.h5"
checkpoint=ModelCheckpoint(filepath,monitor='val_accuracy',verbose=1,mode='auto',save_best_only=True)
```

```
#from IPython.display import display
#from PIL import Image
```

```
optimizer =  tf.keras.optimizers.Adam(learning_rate=0.0001)

final_model.compile(optimizer=optimizer,loss='categorical_crossentropy',metrics=['accuracy'])
steps_per_epoch = len(train_generator) // 32
print("steps_per_epoch:",steps_per_epoch)
```

```
    steps_per_epoch: 35
```

```
# fits the model on batches with real-time data augmentation:
final_model.fit_generator(train_generator,validation_data=valid_generator,epochs=10,steps_per_epoch = steps_per_epoch,callbacks=[tensorboard])
```

Saving...

```
                                 ======] - 210s 6s/step - loss: 2.4152 - accuracy: 0.2384 - val_loss: 2.2512 - val_accuracy: 0.2701
    Epoch 2/10
    35/35 [==============================] - 194s 6s/step - loss: 2.2345 - accuracy: 0.2705 - val_loss: 2.1170 - val_accuracy: 0.3164
    Epoch 3/10
    35/35 [==============================] - 185s 5s/step - loss: 2.0666 - accuracy: 0.3277 - val_loss: 2.0555 - val_accuracy: 0.3516
    Epoch 4/10
    35/35 [==============================] - 190s 6s/step - loss: 2.0806 - accuracy: 0.3098 - val_loss: 1.9717 - val_accuracy: 0.3557
    Epoch 5/10
    35/35 [==============================] - 223s 7s/step - loss: 1.9214 - accuracy: 0.3920 - val_loss: 1.9365 - val_accuracy: 0.3781
    Epoch 6/10
    35/35 [==============================] - 189s 6s/step - loss: 1.9825 - accuracy: 0.3571 - val_loss: 1.8786 - val_accuracy: 0.4059
    Epoch 7/10
    35/35 [==============================] - 185s 5s/step - loss: 1.9049 - accuracy: 0.3768 - val_loss: 1.8917 - val_accuracy: 0.3913
    Epoch 8/10
    35/35 [==============================] - 193s 6s/step - loss: 1.7933 - accuracy: 0.4205 - val_loss: 1.8636 - val_accuracy: 0.4156
    Epoch 9/10
    35/35 [==============================] - 192s 6s/step - loss: 1.8041 - accuracy: 0.4179 - val_loss: 1.8351 - val_accuracy: 0.4128
    Epoch 10/10
    35/35 [==============================] - 191s 6s/step - loss: 1.7969 - accuracy: 0.4170 - val_loss: 1.7774 - val_accuracy: 0.4305
    <keras.callbacks.History at 0x7f03523322b0>
```

```
steps_per_epoch = len(train_generator) // 25
print(steps_per_epoch)
final_model.fit_generator(train_generator,validation_data=valid_generator,epochs=30,steps_per_epoch = steps_per_epoch,callbacks=[tensorboard])
```

```
    45/45 [==============================] - 185s 4s/step - loss: 1.7772 - accuracy: 0.4347 - val_loss: 1.7478 - val_accuracy: 0.4375
```

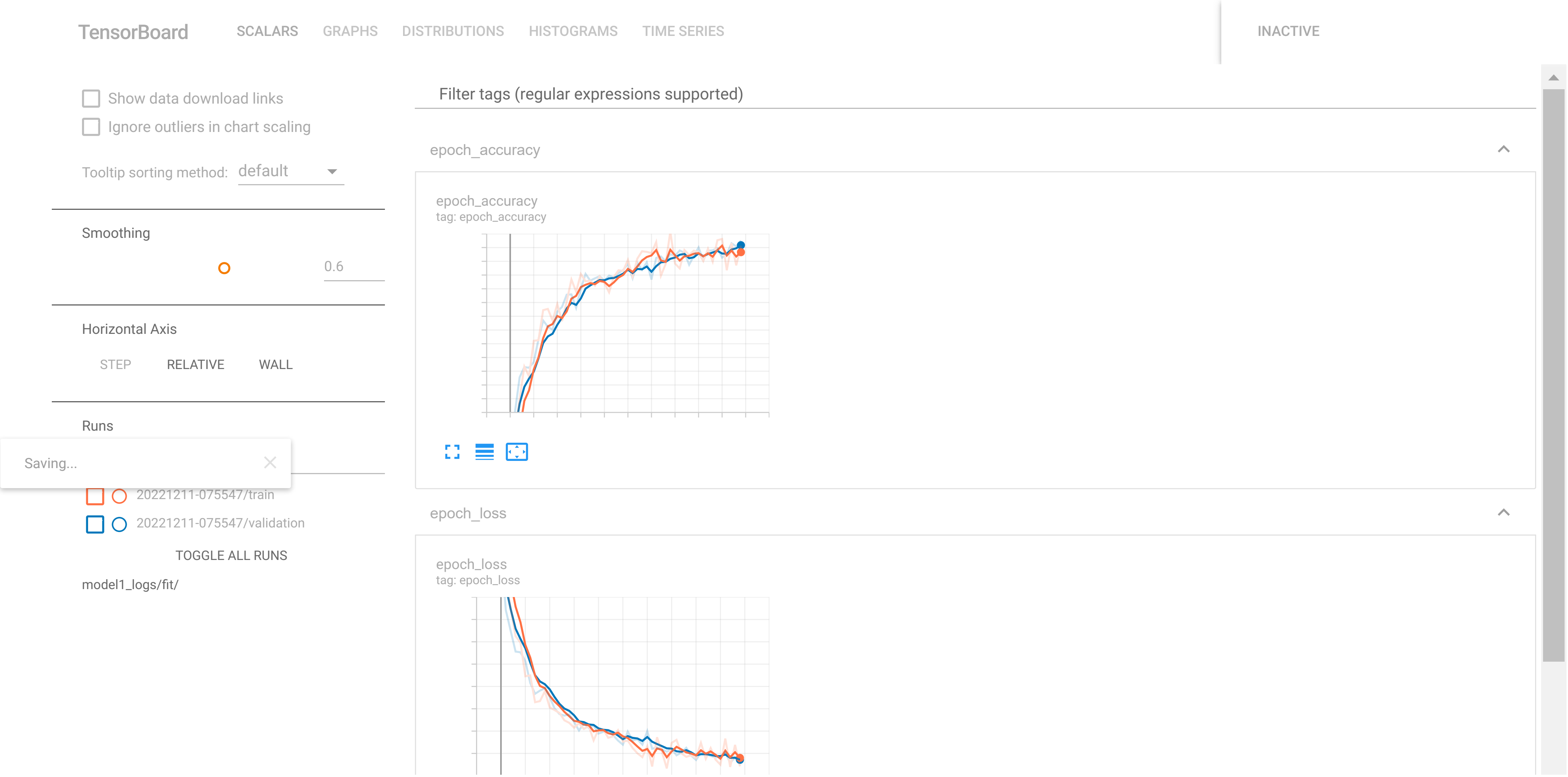
```
Epoch 5/30
45/45 [==============================] - 188s 4s/step - loss: 1.7381 - accuracy: 0.4326 - val_loss: 1.6980 - val_accuracy: 0.4559
Epoch 6/30
45/45 [==============================] - 186s 4s/step - loss: 1.7195 - accuracy: 0.4451 - val_loss: 1.7349 - val_accuracy: 0.4499
Epoch 7/30
45/45 [==============================] - 185s 4s/step - loss: 1.6592 - accuracy: 0.4708 - val_loss: 1.7177 - val_accuracy: 0.4545
Epoch 8/30
45/45 [==============================] - 184s 4s/step - loss: 1.6229 - accuracy: 0.4924 - val_loss: 1.7286 - val_accuracy: 0.4484
Epoch 9/30
45/45 [==============================] - 185s 4s/step - loss: 1.6650 - accuracy: 0.4667 - val_loss: 1.6620 - val_accuracy: 0.4705
Epoch 10/30
45/45 [==============================] - 186s 4s/step - loss: 1.6570 - accuracy: 0.4604 - val_loss: 1.7361 - val_accuracy: 0.4442
Epoch 11/30
45/45 [==============================] - 184s 4s/step - loss: 1.6166 - accuracy: 0.4750 - val_loss: 1.6537 - val_accuracy: 0.4705
Epoch 12/30
45/45 [==============================] - 185s 4s/step - loss: 1.6932 - accuracy: 0.4799 - val_loss: 1.6463 - val_accuracy: 0.4709
Epoch 13/30
45/45 [==============================] - 188s 4s/step - loss: 1.6788 - accuracy: 0.4688 - val_loss: 1.6996 - val_accuracy: 0.4607
Epoch 14/30
45/45 [==============================] - 185s 4s/step - loss: 1.5968 - accuracy: 0.4979 - val_loss: 1.6995 - val_accuracy: 0.4577
Epoch 15/30
45/45 [==============================] - 186s 4s/step - loss: 1.6164 - accuracy: 0.4993 - val_loss: 1.6234 - val_accuracy: 0.4807
Epoch 16/30
45/45 [==============================] - 186s 4s/step - loss: 1.6044 - accuracy: 0.4972 - val_loss: 1.6212 - val_accuracy: 0.4798
Epoch 17/30
45/45 [==============================] - 184s 4s/step - loss: 1.5950 - accuracy: 0.4910 - val_loss: 1.6635 - val_accuracy: 0.4687
Epoch 18/30
45/45 [==============================] - 184s 4s/step - loss: 1.6447 - accuracy: 0.4806 - val_loss: 1.6180 - val_accuracy: 0.4811
Epoch 19/30
Saving...
======] - 185s 4s/step - loss: 1.6567 - accuracy: 0.4639 - val_loss: 1.5903 - val_accuracy: 0.4945

======] - 185s 4s/step - loss: 1.5869 - accuracy: 0.4903 - val_loss: 1.5877 - val_accuracy: 0.4939
Epoch 21/30
45/45 [==============================] - 184s 4s/step - loss: 1.5770 - accuracy: 0.4958 - val_loss: 1.5912 - val_accuracy: 0.4887
Epoch 22/30
45/45 [==============================] - 183s 4s/step - loss: 1.5709 - accuracy: 0.4958 - val_loss: 1.5979 - val_accuracy: 0.4923
Epoch 23/30
45/45 [==============================] - 188s 4s/step - loss: 1.6106 - accuracy: 0.5014 - val_loss: 1.6320 - val_accuracy: 0.4832
Epoch 24/30
45/45 [==============================] - 188s 4s/step - loss: 1.6448 - accuracy: 0.4896 - val_loss: 1.5726 - val_accuracy: 0.4958
Epoch 25/30
45/45 [==============================] - 192s 4s/step - loss: 1.5821 - accuracy: 0.4903 - val_loss: 1.5900 - val_accuracy: 0.4807
Epoch 26/30
45/45 [==============================] - 184s 4s/step - loss: 1.5755 - accuracy: 0.4986 - val_loss: 1.5593 - val_accuracy: 0.5092
Epoch 27/30
45/45 [==============================] - 188s 4s/step - loss: 1.5616 - accuracy: 0.5118 - val_loss: 1.5896 - val_accuracy: 0.4997
Epoch 28/30
45/45 [==============================] - 192s 4s/step - loss: 1.5154 - accuracy: 0.5194 - val_loss: 1.5309 - val_accuracy: 0.5104
Epoch 29/30
45/45 [==============================] - 194s 4s/step - loss: 1.5934 - accuracy: 0.4910 - val_loss: 1.5686 - val_accuracy: 0.5013
Epoch 30/30
45/45 [==============================] - 190s 4s/step - loss: 1.5354 - accuracy: 0.5097 - val_loss: 1.5644 - val_accuracy: 0.4967
<keras.callbacks.History at 0x7f03323416a0>
```

```
steps_per_epoch = len(train_generator) // 32
print(steps_per_epoch)
final_model.fit_generator(train_generator,validation_data=valid_generator,epochs=50,steps_per_epoch = steps_per_epoch,callbacks=[tensorboard])
```

```
Epoch 24/50
35/35 [==============================] - 186s 5s/step - loss: 1.6765 - accuracy: 0.4670 - val_loss: 1.6819 - val_accuracy: 0.4627
Epoch 25/50
35/35 [==============================] - 185s 5s/step - loss: 1.7043 - accuracy: 0.4661 - val_loss: 1.6577 - val_accuracy: 0.4674
Epoch 26/50
35/35 [==============================] - 185s 5s/step - loss: 1.6530 - accuracy: 0.4821 - val_loss: 1.6391 - val_accuracy: 0.4763
Epoch 27/50
35/35 [==============================] - 184s 5s/step - loss: 1.6574 - accuracy: 0.4580 - val_loss: 1.6983 - val_accuracy: 0.4554
Epoch 28/50
35/35 [==============================] - 185s 5s/step - loss: 1.6243 - accuracy: 0.4839 - val_loss: 1.6582 - val_accuracy: 0.4780
Epoch 29/50
35/35 [==============================] - 186s 5s/step - loss: 1.6001 - accuracy: 0.4946 - val_loss: 1.6626 - val_accuracy: 0.4675
Epoch 30/50
35/35 [==============================] - 185s 5s/step - loss: 1.5831 - accuracy: 0.4938 - val_loss: 1.6393 - val_accuracy: 0.4785
Epoch 31/50
35/35 [==============================] - 184s 5s/step - loss: 1.6324 - accuracy: 0.4920 - val_loss: 1.7001 - val_accuracy: 0.4534
Epoch 32/50
35/35 [==============================] - 185s 5s/step - loss: 1.5415 - accuracy: 0.5080 - val_loss: 1.6209 - val_accuracy: 0.4849
Epoch 33/50
35/35 [==============================] - 185s 5s/step - loss: 1.6726 - accuracy: 0.4589 - val_loss: 1.6270 - val_accuracy: 0.4871
Epoch 34/50
35/35 [==============================] - 185s 5s/step - loss: 1.6077 - accuracy: 0.4759 - val_loss: 1.6186 - val_accuracy: 0.4803
Epoch 35/50
35/35 [==============================] - 185s 5s/step - loss: 1.5329 - accuracy: 0.5232 - val_loss: 1.6080 - val_accuracy: 0.4903
Epoch 36/50
35/35 [==============================] - 184s 5s/step - loss: 1.6473 - accuracy: 0.4768 - val_loss: 1.6199 - val_accuracy: 0.4881
Epoch 37/50
35/35 [==============================] - 184s 5s/step - loss: 1.6610 - accuracy: 0.4696 - val_loss: 1.5915 - val_accuracy: 0.4958
Epoch 38/50
                  ======] - 185s 5s/step - loss: 1.6002 - accuracy: 0.4964 - val_loss: 1.6041 - val_accuracy: 0.4925
  Saving...
                  ======] - 185s 5s/step - loss: 1.5848 - accuracy: 0.4893 - val_loss: 1.6254 - val_accuracy: 0.4750
Epoch 40/50
35/35 [==============================] - 185s 5s/step - loss: 1.5918 - accuracy: 0.4955 - val_loss: 1.5905 - val_accuracy: 0.4879
Epoch 41/50
35/35 [==============================] - 186s 5s/step - loss: 1.5800 - accuracy: 0.4929 - val_loss: 1.5707 - val_accuracy: 0.5002
Epoch 42/50
35/35 [==============================] - 184s 5s/step - loss: 1.6478 - accuracy: 0.4804 - val_loss: 1.6042 - val_accuracy: 0.4816
Epoch 43/50
35/35 [==============================] - 185s 5s/step - loss: 1.5760 - accuracy: 0.4982 - val_loss: 1.5967 - val_accuracy: 0.4952
Epoch 44/50
35/35 [==============================] - 193s 6s/step - loss: 1.6239 - accuracy: 0.4795 - val_loss: 1.5891 - val_accuracy: 0.4960
Epoch 45/50
35/35 [==============================] - 191s 6s/step - loss: 1.5723 - accuracy: 0.5107 - val_loss: 1.5816 - val_accuracy: 0.4991
Epoch 46/50
35/35 [==============================] - 194s 6s/step - loss: 1.5499 - accuracy: 0.5125 - val_loss: 1.5867 - val_accuracy: 0.4853
Epoch 47/50
35/35 [==============================] - 196s 6s/step - loss: 1.6666 - accuracy: 0.4670 - val_loss: 1.6061 - val_accuracy: 0.4907
Epoch 48/50
35/35 [==============================] - 194s 6s/step - loss: 1.5397 - accuracy: 0.5063 - val_loss: 1.5574 - val_accuracy: 0.5062
Epoch 49/50
35/35 [==============================] - 194s 6s/step - loss: 1.6383 - accuracy: 0.4741 - val_loss: 1.5821 - val_accuracy: 0.5017
Epoch 50/50
35/35 [==============================] - 196s 6s/step - loss: 1.5421 - accuracy: 0.5027 - val_loss: 1.5571 - val_accuracy: 0.5102
<keras.callbacks.History at 0x7f4ad0028340>
```

```
%load_ext tensorboard
%tensorboard --logdir model1_logs/fit/
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 434), started 2:53:14 ago. (Use '!kill 434' to kill it.)
```



□ Show data download links

□ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis

Runs

Saving...

□ ○ 20221211-075547/train

□ ○ 20221211-075547/validation

model1_logs/fit/

Filter tags (regular expressions supported)

epoch_accuracy

epoch_accuracy
tag: epoch_accuracy

epoch_loss

epoch_loss
tag: epoch_loss

Model-2

1. Use VGG-16 pretrained network without Fully Connected layers and initilize all the weights with Imagenet trained weights.
2. After VGG-16 network without FC layers, don't use FC layers, use conv layers only as Fully connected layer.Any FC
layer can be converted to a CONV layer. This conversion will reduce the No of Trainable parameters in FC layers.
For example, an FC layer with K=4096 that is looking at some input volume of size 7×7×512 can be equivalently expressed as a CONV l
In other words, we are setting the filter size to be exactly the size of the input volume, and hence the output will
simply be 1×1×4096 since only a single depth column “fits” across the input volume, giving identical result as the
initial FC layer. You can refer this link to better understanding of using Conv layer in place of fully connected layers.
3. Final architecture will be VGG-16 without FC layers(without top), 2 Conv layers identical to FC layers, 1 output layer for 16 cl
4. 4.Print model.summary() and plot the architecture of the model.
Reference for plotting model
5. Train only last 2 Conv layers identical to FC layers, 1 output layer. Don't train the VGG-16 network.

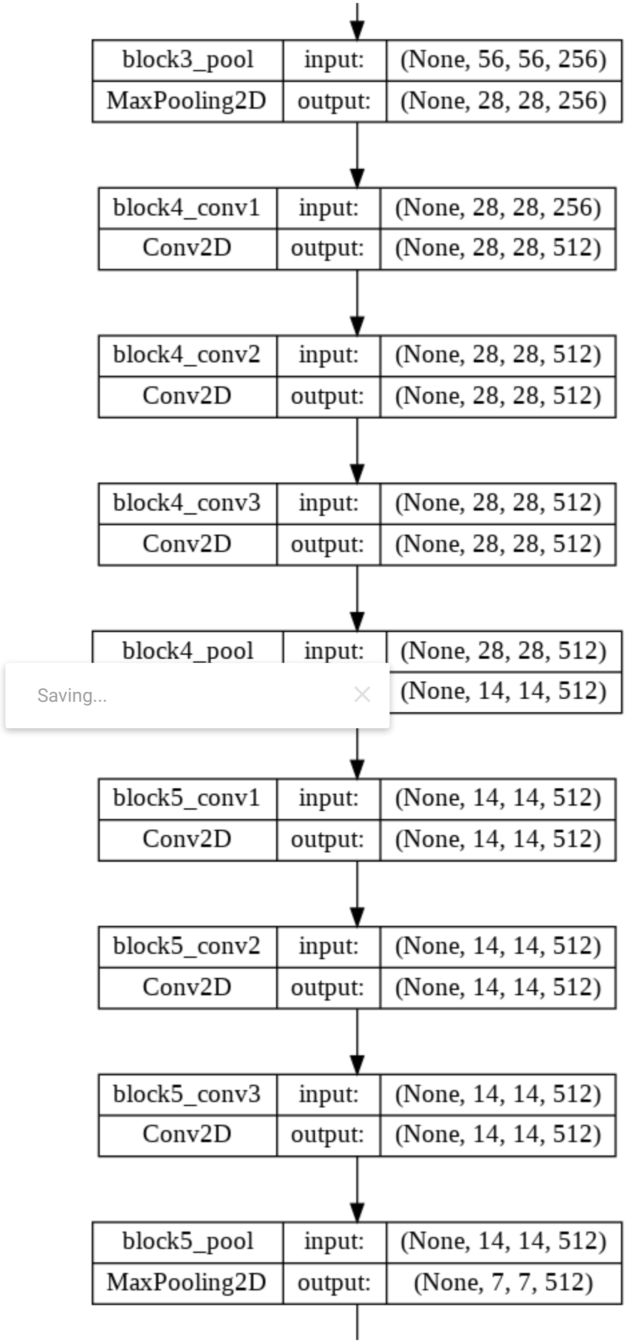
Saving...

```
tf.keras.backend.clear_session()

np.random.seed(0)
rn.seed(0)

## loading vgg16 model without FC layers
vgg16 = VGG16(weights='imagenet',include_top=False,input_shape = (224,224, 3))

  layer.trainable=False

vgg16_op = vgg16.output

## convolution layers as FC layers
conv_layer_1 = Conv2D(filters=32, kernel_size=7, activation='relu',padding='valid')(vgg16_op)
conv_layer_2 = Conv2D(filters=32, kernel_size=1, activation='relu',padding='valid')(conv_layer_1)

flatten = Flatten()(conv_layer_2)

output_layer_2 = Dense(16,activation='softmax')(flatten)

final_model_2 = Model(inputs = vgg16.input, outputs = output_layer_2)
final_model_2.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

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 Saving...                    one, 1, 1, 32)           802848

 conv2d_1 (Conv2D)           (None, 1, 1, 32)          1056

 flatten (Flatten)           (None, 32)                0

 dense (Dense)               (None, 16)                528

=================================================================
Total params: 15,519,120
Trainable params: 804,432
Non-trainable params: 14,714,688
_________________________________________________________________
```

```
input=model.output
conv1=Conv2D(filters=32,kernel_size=7,strides=1,activation='relu')(input)
conv2=Conv2D(filters=32,kernel_size=1,strides=1,activation='relu')(conv1)
flatten=Flatten()(conv2)
output=Dense(16,activation='softmax')(flatten)
model2=Model(inputs=model.input,outputs=output)
model2.summary()
```

```
plot_model(final_model_2,'image_model_2.png',show_shapes=True)
```



block3_pool input: (None, 56, 56, 256)
MaxPooling2D output: (None, 28, 28, 256)

block4_conv1 input: (None, 28, 28, 256)
Conv2D output: (None, 28, 28, 512)

block4_conv2 input: (None, 28, 28, 512)
Conv2D output: (None, 28, 28, 512)

block4_conv3 input: (None, 28, 28, 512)
Conv2D output: (None, 28, 28, 512)

block4_pool input: (None, 28, 28, 512)
(None, 14, 14, 512)

Saving...

block5_conv1 input: (None, 14, 14, 512)
Conv2D output: (None, 14, 14, 512)

block5_conv2 input: (None, 14, 14, 512)
Conv2D output: (None, 14, 14, 512)

block5_conv3 input: (None, 14, 14, 512)
Conv2D output: (None, 14, 14, 512)

block5_pool input: (None, 14, 14, 512)
MaxPooling2D output: (None, 7, 7, 512)

```
!rm -rf ./model2_logs/
```

```
log_dir="model2_logs/fit/"+ datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_2=TensorBoard(log_dir=log_dir,histogram_freq=1)

filepath="best_model_2.h5"
checkpoint_2=ModelCheckpoint(filepath,monitor='val_accuracy',verbose=1,mode='auto',save_best_only=True)
```

```
optimizer =  tf.keras.optimizers.Adam(learning_rate=0.0001)
final_model_2.compile(optimizer=optimizer,loss='categorical_crossentropy',metrics=['accuracy'])
```

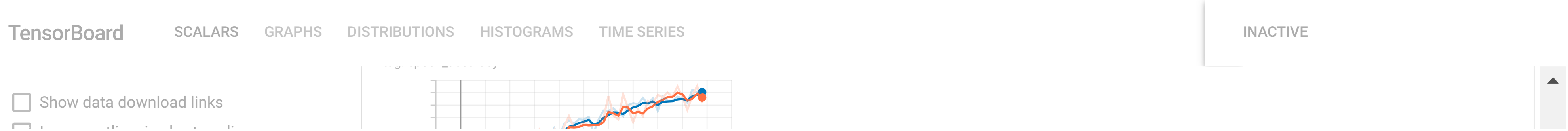
```
##fitting generator
steps_per_epoch = len(train_generator) // 32
print(steps_per_epoch)
final_model_2.fit_generator(train_generator,validation_data=valid_generator,epochs=50,steps_per_epoch = steps_per_epoch,callbacks=[tensorboard_2,checkpoint_2])
```

Saving...

```
Epoch 41: val_accuracy did not improve from 0.43183
35/35 [==============================] - 194s 6s/step - loss: 1.8427 - accuracy: 0.4366 - val_loss: 1.8840 - val_accuracy: 0.4108
Epoch 42/50
35/35 [==============================] - ETA: 0s - loss: 1.8748 - accuracy: 0.4330
Epoch 42: val_accuracy improved from 0.43183 to 0.43417, saving model to best_model_2.h5
35/35 [==============================] - 194s 6s/step - loss: 1.8748 - accuracy: 0.4330 - val_loss: 1.8293 - val_accuracy: 0.4342
Epoch 43/50
35/35 [==============================] - ETA: 0s - loss: 1.8510 - accuracy: 0.4402
Epoch 43: val_accuracy did not improve from 0.43417
35/35 [==============================] - 196s 6s/step - loss: 1.8510 - accuracy: 0.4402 - val_loss: 1.8394 - val_accuracy: 0.4270
Epoch 44/50
35/35 [==============================] - ETA: 0s - loss: 1.8155 - accuracy: 0.4339
Epoch 44: val_accuracy did not improve from 0.43417
35/35 [==============================] - 222s 7s/step - loss: 1.8155 - accuracy: 0.4339 - val_loss: 1.8236 - val_accuracy: 0.4272
Epoch 45/50
35/35 [==============================] - ETA: 0s - loss: 1.7934 - accuracy: 0.4500
Epoch 45: val_accuracy did not improve from 0.43417
35/35 [==============================] - 197s 6s/step - loss: 1.7934 - accuracy: 0.4500 - val_loss: 1.8194 - val_accuracy: 0.4338
```

```
%load_ext tensorboard
%tensorboard --logdir model2_logs/fit/
```

Saving... ×



Model-3

1. Use same network as Model-2 '**INPUT --> VGG-16 without Top layers(FC) --> 2 Conv Layers identical to FC --> Output Layer**' and tra

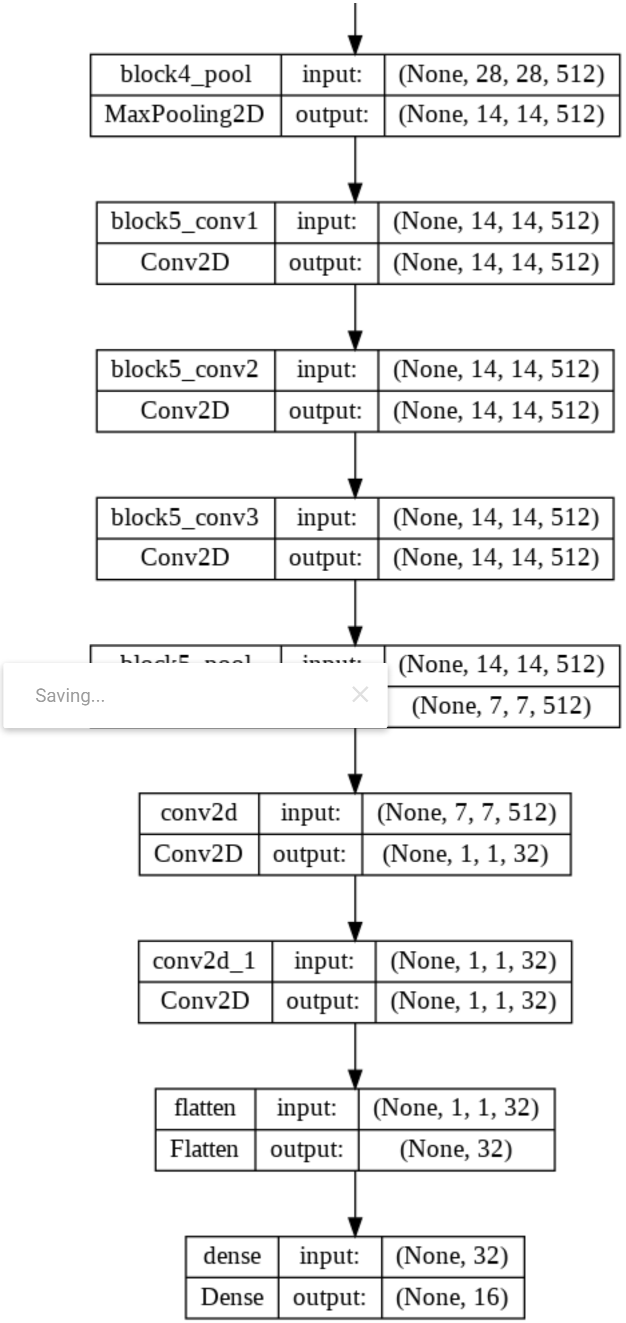
```
tf.keras.backend.clear_session()

np.random.seed(0)
rn.seed(0)

vgg16 = VGG16(weights='imagenet',include_top=False,input_shape = (224,224, 3))

## making only the last 6 layers of VGG16 as trainable
for l in vgg16.layers[:-6]:
```

Saving...

```
vgg16_op = vgg16.output
conv_layer_1 = Conv2D(filters=32, kernel_size=7, activation='relu',padding='valid')(vgg16_op)
conv_layer_2 = Conv2D(filters=32, kernel_size=1, activation='relu',padding='valid')(conv_layer_1)
flatten = Flatten()(conv_layer_2)
output_layer_3 = Dense(16,activation='softmax')(flatten)

final_model_3=Model(inputs = vgg16.input, outputs = output_layer_3)
final_model_3.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 4s 0us/step
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

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 conv2d (Conv2D)             (None, 1, 1, 32)          802848

 conv2d_1 (Conv2D)           (None, 1, 1, 32)          1056

 flatten (Flatten)           (None, 32)                0

   Saving...                  one, 16)                 528

=================================================================
Total params: 15,519,120
Trainable params: 10,243,664
Non-trainable params: 5,275,456
_________________________________________________________________
```

```
plot_model(final_model_3,'image_model_3.png',show_shapes=True)
```


block4_pool
MaxPooling2D
input: (None, 28, 28, 512)
output: (None, 14, 14, 512)
block5_conv1
Conv2D
input: (None, 14, 14, 512)
output: (None, 14, 14, 512)
block5_conv2
Conv2D
input: (None, 14, 14, 512)
output: (None, 14, 14, 512)
block5_conv3
Conv2D
input: (None, 14, 14, 512)
output: (None, 14, 14, 512)
block5_pool
input: (None, 14, 14, 512)
(None, 7, 7, 512)
Saving...
conv2d
Conv2D
input: (None, 7, 7, 512)
output: (None, 1, 1, 32)
conv2d_1
Conv2D
input: (None, 1, 1, 32)
output: (None, 1, 1, 32)
flatten
Flatten
input: (None, 1, 1, 32)
output: (None, 32)
dense
Dense
input: (None, 32)
output: (None, 16)

```
!rm -rf ./model3_logs/
```

```
log_dir="model3_logs/fit/"+ datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_3=TensorBoard(log_dir=log_dir,histogram_freq=1)

filepath="best_model_3.h5"
checkpoint_3=ModelCheckpoint(filepath,monitor='val_accuracy',verbose=1,mode='auto',save_best_only=True)
```

```
optimizer =  tf.keras.optimizers.Adam(learning_rate=0.0001)
final_model_3.compile(optimizer=optimizer,loss='categorical_crossentropy',metrics=['accuracy'])
```

```
steps_per_epoch = len(train_generator) // 25
print(steps_per_epoch)
```

```
45
```

```
##fitting generator
final_model_3.fit_generator(train_generator,validation_data=valid_generator,epochs=50,steps_per_epoch=steps_per_epoch,callbacks=[tensorboard_3,checkpoint_3])
```

Saving...

```
45/45 [==============================] - 207s 5s/step - loss: 1.3426 - accuracy: 0.5938 - val_loss: 1.3734 - val_accuracy: 0.5919
Epoch 46/50
45/45 [==============================] - ETA: 0s - loss: 1.3110 - accuracy: 0.6021
Epoch 46: val_accuracy did not improve from 0.59192
45/45 [==============================] - 205s 5s/step - loss: 1.3110 - accuracy: 0.6021 - val_loss: 1.3895 - val_accuracy: 0.5821
Epoch 47/50
45/45 [==============================] - ETA: 0s - loss: 1.4374 - accuracy: 0.5778
Epoch 47: val_accuracy did not improve from 0.59192
45/45 [==============================] - 226s 5s/step - loss: 1.4374 - accuracy: 0.5778 - val_loss: 1.3557 - val_accuracy: 0.5880
Epoch 48/50
45/45 [==============================] - ETA: 0s - loss: 1.3399 - accuracy: 0.5993
Epoch 48: val_accuracy did not improve from 0.59192
45/45 [==============================] - 193s 4s/step - loss: 1.3399 - accuracy: 0.5993 - val_loss: 1.3873 - val_accuracy: 0.5883
Epoch 49/50
45/45 [==============================] - ETA: 0s - loss: 1.4072 - accuracy: 0.5840
Epoch 49: val_accuracy improved from 0.59192 to 0.59383, saving model to best_model_3.h5
45/45 [==============================] - 193s 4s/step - loss: 1.4072 - accuracy: 0.5840 - val_loss: 1.3496 - val_accuracy: 0.5938
Epoch 50/50
45/45 [==============================] - ETA: 0s - loss: 1.3359 - accuracy: 0.5951
Epoch 50: val_accuracy did not improve from 0.59383
45/45 [==============================] - 190s 4s/step - loss: 1.3359 - accuracy: 0.5951 - val_loss: 1.3586 - val_accuracy: 0.5878
<keras.callbacks.History at 0x7f44300351c0>
```

```
%load_ext tensorboard
%tensorboard --logdir model3_logs/fit/
```

Saving...