Transfer Learning Assignment

Download all the data in this rar_file (https://drive.google.com/open?id=1Z4TyI7FcFVEx8qdl4jO9qxvxaqLSqoEu) , 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
    11 invoice
    12 presentation
    13 questionnaire
    14 resume
    15 memo
```

In [1]:

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

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

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.

In [2]:
```
#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
```

In [3]:
```
!gdown --id 1Z4TyI7FcFVEx8qdl4jO9qxvxaqLSqoEu
```

```
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Option `--id` was deprecated in version 4.3.1 a
nd will be removed in 5.0. You don't need to pass it anymore to use a file ID.
  category=FutureWarning,
Access denied with the following error:

        Too many users have viewed or downloaded this file recently. Please
        try accessing the file again later. If the file you are trying to
        access is particularly large or is shared with many people, it may
        take up to 24 hours to be able to view or download the file. If you
        still can't access a file after 24 hours, contact your domain
        administrator.

You may still be able to access the file from the browser:

         https://drive.google.com/uc?id=1Z4TyI7FcFVEx8qdl4jO9qxvxaqLSqoEu (https://drive.google.com/uc?id=1Z4TyI7FcFVEx
8qdl4jO9qxvxaqLSqoEu)
```

In [4]:
```
# Method -2 you can also import the data using wget function
#https://www.youtube.com/watch?v=BPUfVq7RaY8
```

In [5]:
```
#unrar the file
get_ipython().system_raw("unrar x /content/drive/MyDrive/rvl-cdip.rar")
```

In [5]:
```

```

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-imagedatagenerator-with-flow-from-dataframe-8bd5776e45c1)

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

Model 1

1. Use VGG-16 (https://www.tensorflow.org/api_docs/python/tf/keras/applications/VGG16) pretrained network without Fully Connected layers and initilize all the weights with Imagenet trained weights.
2. After VGG-16 network without 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 are free to choose any hyperparameters/parameters of conv block, FC layers, output layer.
3. Final architecture will be **INPUT --> VGG-16 without Top layers(FC) --> Conv Layer --> Maxpool Layer --> 2 FC layers --> Output Layer**
4. Print model.summary() and plot the architecture of the model. Reference for plotting model (https://www.tensorflow.org/api_docs/python/tf/keras/utils/plot_model)
5. Train only new Conv block, FC layers, output layer. Don't train the VGG-16 network.

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.

In [6]:

```
import matplotlib.pyplot as plt # importing the libraries
import pandas as pd
import numpy as np
import seaborn as sns
import tensorflow as tf
import datetime, os
from tensorflow import keras
from keras.models import Model

df=pd.read_csv("labels_final.csv")
```

In [7]:

```
!wget --header="Host: doc-0k-3k-docs.googleusercontent.com" --header="User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x
```

```
--2022-11-04 11:30:20--  https://doc-0k-3k-docs.googleusercontent.com/docs/securesc/8h8uvuh5ifib89027b7ihknt22nu5vgc/pb
2qojp66b5mdedm2vf2dtj92nda0jem/1593320700000/00484516897554883881/05866892802988797180/1Z4TyI7FcFVEx8qdl4jO9qxvxaqLSqoE
u?e=download&authuser=0&nonce=cvcfvs0ct36a2&user=05866892802988797180&hash=ilt33fv0qrj4td17qaeas607uq6g796v (https://do
c-0k-3k-docs.googleusercontent.com/docs/securesc/8h8uvuh5ifib89027b7ihknt22nu5vgc/pb2qojp66b5mdedm2vf2dtj92nda0jem/1593
320700000/00484516897554883881/05866892802988797180/1Z4TyI7FcFVEx8qdl4jO9qxvxaqLSqoEu?e=download&authuser=0&nonce=cvcfv
s0ct36a2&user=05866892802988797180&hash=ilt33fv0qrj4td17qaeas607uq6g796v)
Resolving doc-0k-3k-docs.googleusercontent.com (doc-0k-3k-docs.googleusercontent.com)... 142.251.8.132, 2404:6800:4008:
c15::84
Connecting to doc-0k-3k-docs.googleusercontent.com (doc-0k-3k-docs.googleusercontent.com)|142.251.8.132|:443... connect
ed.
HTTP request sent, awaiting response... 403 Forbidden
2022-11-04 11:30:20 ERROR 403: Forbidden.
```

In [8]:

```
labels_dict={ 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",
    11 :"invoice",
    12 :" presentation",
    13 :"questionnaire",
    14 :"resume",
    15: "memo"}
```

In [9]:

```
df['label']=df['label'].apply(lambda x:labels_dict[x])
```

In [10]:

```
df.head(5)
```

Out[10]:

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

generating the image

In [11]:

```
from keras_preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(rescale=1/255., validation_split=0.2)
```

In [12]:

```
 # train data
print("------TRAIN DATA-------")
train_generator = datagen.flow_from_dataframe(dataframe=df, directory="/content/data_final",
                                             x_col='path',
                                             y_col='label', # using flow from data frame
                                    target_size=(256,256),
                                             class_mode='categorical',
                                             batch_size=32,
                                             subset='training',
                                             seed=0)
```

```
------TRAIN DATA-------
Found 38400 validated image filenames belonging to 16 classes.
```

In [13]:

```
# cross validation data

print("------CROSS VALIDATION DATA-------") # cross validation data
validation_generator = datagen.flow_from_dataframe(dataframe=df, directory="/content/data_final",
                                             x_col='path',
                                             y_col='label',
                                             target_size=(256,256),
                                             class_mode='categorical',
                                             batch_size=32,
                                             subset='validation',
                                             seed=0)
```

```
------CROSS VALIDATION DATA-------
Found 9600 validated image filenames belonging to 16 classes.
```

IMPORTING THE NECESSARY LIBRARIES

In [14]:

```
from keras.layers import Input, Lambda, Dense, Flatten
from keras.models import Model
from keras.applications.vgg16 import VGG16
from keras.callbacks import Callback
from keras.callbacks import TensorBoard
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
from keras.layers import Dense, Conv2D, MaxPool2D , Flatten
```

In [15]:

```
%load_ext tensorboard
```

plotting the tensor board

In [16]:

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

HERE WE ARE DOWNLODING THE PRE TRAINED VGG MODEL

In [17]:

```
IMAGE_SIZE = [256, 256]
model = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_
tf_kernels_notop.h5 (https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_t
f_kernels_notop.h5)
58889256/58889256 [==============================] - 3s 0us/step
```

#pre trained vgg16 model

In [18]:
```
model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0         
                                                                 
 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0         
                                                                 
 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808   
                                                                 
 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0         
                                                                 
 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808   
 block5_pool (MaxPooling2D)  (None, 8, 8, 512)        0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

#MODEL_1(INPUT --> VGG-16 without Top layers(FC) --> Conv Layer --> Maxpool Layer --> 2 FC layers --> Output Layer)

In [19]:

```
# model_1
for layer in model.layers:
  layer.trainable = False
# Adding custom Layers
x = model.output
x = Conv2D(filters=30,kernel_size=(3,3),padding="same", activation="relu")(x)

# here we are using the maxpooling
x = MaxPool2D(2,2)(x)
x = Flatten()(x)

# dense layer with relu activation function
# dense fully connected layer
x = Dense(64, activation="relu")(x)
x = Dense(32, activation="relu")(x)

output = Dense(16, activation="softmax")(x)
# creating the final model
model_1 = Model(inputs = model.input, outputs = output)
# compile the model
model_1.compile(loss = "categorical_crossentropy", optimizer ='Adam', metrics=["accuracy"])
```

MODEL_1 SUMMARY

In [20]:
```
model_1.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0         
                                                                 
 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0         
                                                                 
 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808   
                                                                 
 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0         
                                                                 
 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808   
 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0

 conv2d (Conv2D)             (None, 8, 8, 30)          138270

 max_pooling2d (MaxPooling2D  (None, 4, 4, 30)         0
 )

 flatten (Flatten)           (None, 480)               0

 dense (Dense)               (None, 64)                30784

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 16)                528

=================================================================
Total params: 14,886,350
Trainable params: 171,662
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [21]:

```
train_steps = train_generator.n//train_generator.batch_size
validation_steps = validation_generator.n//validation_generator.batch_size
```

FITTIING the model_1

In [22]:
```
#fitting the model_1
model_1.fit_generator(train_generator,steps_per_epoch=train_steps, epochs=5,
                              validation_data=validation_generator,validation_steps=validation_steps,callbacks=[tensorbo
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: UserWarning: `Model.fit_generator` is deprecated and wi
ll be removed in a future version. Please use `Model.fit`, which supports generators.
  This is separate from the ipykernel package so we can avoid doing imports until

Epoch 1/5
1200/1200 [==============================] - 318s 254ms/step - loss: 1.4826 - accuracy: 0.5415 - val_loss: 1.2036 - val
_accuracy: 0.6263
Epoch 2/5
1200/1200 [==============================] - 307s 256ms/step - loss: 1.0382 - accuracy: 0.6810 - val_loss: 1.0439 - val
_accuracy: 0.6864
Epoch 3/5
1200/1200 [==============================] - 307s 256ms/step - loss: 0.9120 - accuracy: 0.7217 - val_loss: 0.9398 - val
_accuracy: 0.7202
Epoch 4/5
1200/1200 [==============================] - 308s 256ms/step - loss: 0.8229 - accuracy: 0.7462 - val_loss: 0.9334 - val
_accuracy: 0.7225
Epoch 5/5
1200/1200 [==============================] - 308s 256ms/step - loss: 0.7502 - accuracy: 0.7663 - val_loss: 0.9511 - val
_accuracy: 0.7201
```

Out[22]:
```
<keras.callbacks.History at 0x7f4404a34110>
```

In [23]:
```
%tensorboard --logdir logs
```

```
<IPython.core.display.Javascript object>
```

#OBSERVATION

##We can see that no of epochs we have taken 5 and after 5 epochs accuracy is 76% i think if i increase the no of epoch then may be we can increase the accuracy ##We can see the increasing the number of epoch the loss is getting decreasing significantly ##Due to less computing resources we have decreased the no of paramters otherwise we can increase the accuracy ##imp thing what i have observed that all the parameters are not trainable

Model-2

```
1. Use VGG-16 (https://www.tensorflow.org/api_docs/python/tf/keras/applications/VGG16) pretrained network withou
t 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.An
y 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 ex
pressed as a CONV layer with F=7,P=0,S=1,K=4096.
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 t
he
initial FC layer. You can refer this (http://cs231n.github.io/convolutional-networks/#convert) link to better un
derstanding 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 out
put layer for 16 class classification. INPUT --> VGG-16 without Top layers(FC) --> 2 Conv Layers identical to FC
-->Output Layer
4. 4.Print model.summary() and plot the architecture of the model.
Reference for plotting model (https://www.tensorflow.org/api_docs/python/tf/keras/utils/plot_model)
5. Train only last 2 Conv layers identical to FC layers, 1 output layer. Don't train the VGG-16 network.
```

In [24]:

```
!rm -rf ./logs/
```

In [25]:

```
#model_2
for layer in model.layers:
  layer.trainable = False

#Adding custom Layers with strides using relu activation function
x = model.output
x = Conv2D(filters=512,kernel_size=8 ,strides=1,activation="relu")(x)
x = Conv2D(filters=512,kernel_size=1 ,strides=1,activation="relu")(x)
x = Flatten()(x)

# creating the final model
output= Dense(16, activation="softmax")(x)
model_2 = Model(inputs = model.input, outputs = output)

# compile the model
model_2.compile(loss="categorical_crossentropy",optimizer = 'Adam',metrics=['accuracy'])
```

In [26]:

```
# summary of the model_2
model_2.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0         
                                                                 
 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0         
                                                                 
 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808   
                                                                 
 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0         
                                                                 
 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808   
 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0

 conv2d_1 (Conv2D)           (None, 1, 1, 512)         16777728

 conv2d_2 (Conv2D)           (None, 1, 1, 512)         262656

 flatten_1 (Flatten)         (None, 512)               0

 dense_3 (Dense)             (None, 16)                8208

=================================================================
Total params: 31,763,280
Trainable params: 17,048,592
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [27]:
```
#fitting model_2
model_2.fit_generator(train_generator,steps_per_epoch=train_steps, epochs=5,verbose=1,
                              validation_data=validation_generator,validation_steps=validation_steps,callbacks=[tensorbo
```

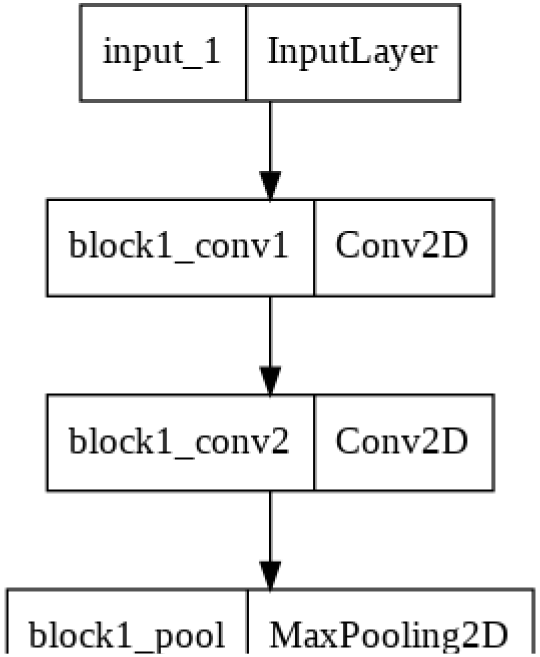
```
Epoch 1/5

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: UserWarning: `Model.fit_generator` is deprecated and wi
ll be removed in a future version. Please use `Model.fit`, which supports generators.
  This is separate from the ipykernel package so we can avoid doing imports until

1200/1200 [==============================] - 333s 277ms/step - loss: 1.2643 - accuracy: 0.6189 - val_loss: 0.9703 - val
_accuracy: 0.7105
Epoch 2/5
1200/1200 [==============================] - 332s 277ms/step - loss: 0.8806 - accuracy: 0.7305 - val_loss: 0.9784 - val
_accuracy: 0.7088
Epoch 3/5
1200/1200 [==============================] - 332s 276ms/step - loss: 0.7280 - accuracy: 0.7756 - val_loss: 0.9525 - val
_accuracy: 0.7143
Epoch 4/5
1200/1200 [==============================] - 332s 276ms/step - loss: 0.6193 - accuracy: 0.8071 - val_loss: 1.0516 - val
_accuracy: 0.7161
Epoch 5/5
1200/1200 [==============================] - 332s 276ms/step - loss: 0.5441 - accuracy: 0.8310 - val_loss: 0.9664 - val
_accuracy: 0.7267
```

Out[27]:
```
<keras.callbacks.History at 0x7f43302108d0>
```

In [28]:

```
# model graphs
tf.keras.utils.plot_model(
    model_2, to_file='model_2.png', show_shapes=False, show_layer_names=True,
    rankdir='TB', expand_nested=False, dpi=96
)
```

Out[28]:



In [29]:

```
%tensorboard --logdir logs
```

```
Reusing TensorBoard on port 6006 (pid 522), started 0:27:41 ago. (Use '!kill 522' to kill it.)
```

```
<IPython.core.display.Javascript object>
```

#OBSERVATION

#IN the model 2 we have increased the no of trainble parameters but i didn't got improvement in accuracy may be due to same no of epochs as model 1

#As per given instruction we have increased no of trainable paramters so it took more time than model 1

#yes in this model also we can see the decreasing in loss with increasing no of epochs

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 train only Last 6 Layers of VGG-16 network, 2 Conv layers identical to FC layers, 1 output layer.

training last 6 layers of vgg16

In [30]:

```
for layer in model.layers[-6:]: # training last 6 layers of vgg16
    layer.trainable = True
    print("Layer '%s' is trainable" % layer.name)
```

```
Layer 'block4_conv3' is trainable
Layer 'block4_pool' is trainable
Layer 'block5_conv1' is trainable
Layer 'block5_conv2' is trainable
Layer 'block5_conv3' is trainable
Layer 'block5_pool' is trainable
```

In [31]:

```
#model_3

#Adding custom Layers
x = model.output

# intilizing with relu activation function
x = Conv2D(filters=512,kernel_size=8 ,strides=1,activation="relu")(x)
x = Conv2D(filters=512,kernel_size=1 ,strides=1,activation="relu")(x)
x = Flatten()(x)

# creating the final model
output = Dense(16, activation="softmax")(x)
model_3 = Model(inputs = model.input, outputs = output)

# compile the model with Adam optimizer
model_3.compile(loss="categorical_crossentropy",optimizer = 'Adam',metrics=['accuracy'])
```

MODEL_3 SUMMARY

In [32]:
```
model_3.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0

 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792

 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928

 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0

 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856

 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584

 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0

 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168

 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080

 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0

 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160

 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0

 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808
 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0         
                                                                 
 conv2d_3 (Conv2D)           (None, 1, 1, 512)         16777728  
                                                                 
 conv2d_4 (Conv2D)           (None, 1, 1, 512)         262656    
                                                                 
 flatten_2 (Flatten)         (None, 512)               0         
                                                                 
 dense_4 (Dense)             (None, 16)                8208      
                                                                 
=================================================================
Total params: 31,763,280
Trainable params: 26,487,824
Non-trainable params: 5,275,456
_________________________________________________________________
```

FITTING model_3

In [33]:

```
model_3.fit_generator(train_generator,steps_per_epoch=train_steps, epochs=5,
                              validation_data=validation_generator,validation_steps=validation_steps,callbacks=[tensorbo
```

```
Epoch 1/5

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: UserWarning: `Model.fit_generator` is deprecated and wi
ll be removed in a future version. Please use `Model.fit`, which supports generators.
  This is separate from the ipykernel package so we can avoid doing imports until

1200/1200 [==============================] - 395s 328ms/step - loss: 2.7836 - accuracy: 0.0623 - val_loss: 2.7728 - val
_accuracy: 0.0601
Epoch 2/5
1200/1200 [==============================] - 386s 322ms/step - loss: 2.7728 - accuracy: 0.0623 - val_loss: 2.7728 - val
_accuracy: 0.0614
Epoch 3/5
1200/1200 [==============================] - 386s 322ms/step - loss: 2.7728 - accuracy: 0.0620 - val_loss: 2.7731 - val
_accuracy: 0.0584
Epoch 4/5
1200/1200 [==============================] - 386s 322ms/step - loss: 2.7728 - accuracy: 0.0616 - val_loss: 2.7731 - val
_accuracy: 0.0601
Epoch 5/5
1200/1200 [==============================] - 386s 322ms/step - loss: 2.7728 - accuracy: 0.0602 - val_loss: 2.7729 - val
_accuracy: 0.0611
```

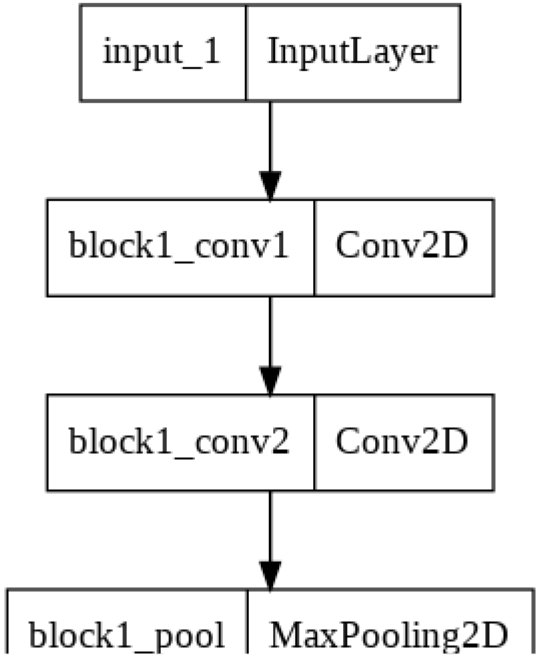
Out[33]:

```
<keras.callbacks.History at 0x7f43186d4550>
```

In [34]:

```
# ploting the acrchitecture
tf.keras.utils.plot_model(
    model_3, to_file='model_3.png', show_shapes=False, show_layer_names=True,
    rankdir='TB', expand_nested=False, dpi=96
)
```

Out[34]:



In [35]:

```
%tensorboard --logdir logs
```

```
Reusing TensorBoard on port 6006 (pid 522), started 1:00:02 ago. (Use '!kill 522' to kill it.)
```

```
<IPython.core.display.Javascript object>
```

#OBSERVATION

#Its really very usefull insights i have learned from this assignment that its not important that if we increase the no of parameters will not directly improve the accuracy it also requires more number of epochs

#so with this model 3 we have got 61% accuracy with 5 number of epochs if we increased the number of epochs may be improve the accuracy

one more important thing in this model i observed that loss is not improve much even increasing the number of parameters

if i increased the number of epochs like 45-50 i think we could get accuracy 90-95%

NOW HERE WE ARE IMPORTING THE PRETTY TO SUMMERIZE THE RESULT OF ALL THREE MODELS

REFERENCE : http://zetcode.com/python/prettytable/ (http://zetcode.com/python/prettytable/)

In [2]:

```
from prettytable import PrettyTable
from prettytable import ALL as ALL


table=PrettyTable(hrules=ALL)
table.field_names = [ "Sl.N0", "Model", "Number of epochs", " val_accuracy"] # # http://zetcode.com/python/prettytable/
table.add_row([1, "model_1", "5", 0.7225])
table.add_row([2, "model_2","5", 0.7265])
table.add_row([3, "model_3","5", 0.0614])
print(table)
```

```
+-------+---------+------------------+---------------+
| Sl.N0 |  Model  | Number of epochs |  val_accuracy |
+-------+---------+------------------+---------------+
|   1   | model_1 |        5         |     0.7225    |
+-------+---------+------------------+---------------+
|   2   | model_2 |        5         |     0.7265    |
+-------+---------+------------------+---------------+
|   3   | model_3 |        5         |     0.0614    |
+-------+---------+------------------+---------------+
```

In [36]: