1. Download all the data in this folder https://drive.google.com/open?id=1Z4TyI7FcFVEx8qdl4jO9qxvxaqLSqoEu. it contains two file both images and labels. The label file list the images and their categories 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

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.

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)

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)

Note: fit_genarator() method will have problems with the tensorboard histograms, try to debug it, if you could not do use histgrams=0 i.e don't include histograms, check the documentation of tensorboard for more information.

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)

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 a 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. Train only new Conv block, FC layers, output layer. Don't train the VGG-16 network.

Model-2

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, 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 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 the initial FC layer. You can refer this (http://cs231n.github.io/convolutional-networks/#convert) 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 class classification. **INPUT --> VGG-16 without Top layers(FC) --> 2 Conv Layers identical to FC --> Output Layer**
3. Train only last 2 Conv layers identical to FC layers, 1 output layer. Don't train the VGG-16 network.

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.

In [ ]:

```
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
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: Fut
ureWarning: pandas.util.testing is deprecated. Use the functions in the publ
ic API at pandas.testing instead.
  import pandas.util.testing as tm
Using TensorFlow backend.
```

In [ ]:

```
!wget --header="Host: doc-0k-3k-docs.googleusercontent.com" --header="User-Agent: Mozi
```

```
--2020-06-28 05:06:02--  https://doc-0k-3k-docs.googleusercontent.com/doc
s/securesc/8h8uvuh5ifib89027b7ihknt22nu5vgc/pb2qojp66b5mdedm2vf2dtj92nda0j
em/1593320700000/00484516897554883881/05866892802988797180/1Z4TyI7FcFVEx8q
dl4jO9qxvxaqLSqoEu?e=download&authuser=0&nonce=cvcfvs0ct36a2&user=05866892
802988797180&hash=ilt33fv0qrj4td17qaeas607uq6g796v (https://doc-0k-3k-doc
s.googleusercontent.com/docs/securesc/8h8uvuh5ifib89027b7ihknt22nu5vgc/pb2
qojp66b5mdedm2vf2dtj92nda0jem/1593320700000/00484516897554883881/058668928
02988797180/1Z4TyI7FcFVEx8qdl4jO9qxvxaqLSqoEu?e=download&authuser=0&nonce=
cvcfvs0ct36a2&user=05866892802988797180&hash=ilt33fv0qrj4td17qaeas607uq6g7
96v)
Resolving doc-0k-3k-docs.googleusercontent.com (doc-0k-3k-docs.googleuserc
ontent.com)... 74.125.204.132, 2404:6800:4008:c04::84
Connecting to doc-0k-3k-docs.googleusercontent.com (doc-0k-3k-docs.googleu
sercontent.com)|74.125.204.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/rar]
Saving to: ‘rvl-cdip.rar’

rvl-cdip.rar            [       <=>          ]   4.34G  39.5MB/s    in 85s

2020-06-28 05:07:27 (52.2 MB/s) - ‘rvl-cdip.rar’ saved [4660541790]
```

In [ ]:

```
get_ipython().system_raw("unrar x rvl-cdip.rar") # extracting the uploaded file
```

In [ ]:

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

In [ ]:

```
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 [ ]:

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

In [ ]:

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

Out[7]:

	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

In [ ]:

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

In [ ]:

```
print("------TRAIN DATA-------")   # train data
train_generator = datagen.flow_from_dataframe(dataframe=df, directory="/content/data_f
                                              x_col='path',
                                              y_col='label', # using flow from data fra
                                    target_size=(256,256),
                                              class_mode='categorical',
                                              batch_size=32,
                                              subset='training',
                                              seed=7)
```

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

In [ ]:

```
print("------CROSS VALIDATION DATA-------") # cross validation data
validation_generator = datagen.flow_from_dataframe(dataframe=df, directory="/content/da
                                                   x_col='path',
                                                   y_col='label',
                                                   target_size=(256,256),
                                                   class_mode='categorical',
                                                   batch_size=32,
                                                   subset='validation',
                                                   seed=7)
```

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

In [ ]:

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

In [ ]:

```
%load_ext tensorboard
```

In [ ]:

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

In [ ]:

```
IMAGE_SIZE = [256, 256] #pre trained vgg16 model
model = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

```
Downloading data from https://github.com/fchollet/deep-learning-models/relea
ses/download/v0.1/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5 (http
s://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg16_we
ights_tf_dim_ordering_tf_kernels_notop.h5)
58892288/58889256 [==============================] - 5s 0us/step
```

In [ ]:

```
model.summary() #pre trained vgg16 model
```

```
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 [ ]:

```
#model_1
for layer in model.layers:
  layer.trainable = False
#Adding custom Layers
x = model.output
x = Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu")(x)
x = MaxPool2D(2,2)(x)
x = Flatten()(x)
x = Dense(256, activation="relu")(x)
x = Dense(128, 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=["accura
```

In [ ]:

```
# summary of the model_1
model_1.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, 8, 8, 512)         2359808
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_3 (Dense)              (None, 16)                2064
=================================================================
Total params: 19,206,864
Trainable params: 4,492,176
```

```
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [ ]:

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

In [ ]:

```
#fitting the model_1
model_1.fit_generator(train_generator,steps_per_epoch=train_steps, epochs=5,
                              validation_data=validation_generator,validation_steps=va
```

```
Epoch 1/5
1200/1200 [==============================] - 715s 596ms/step - loss: 1.2791
- accuracy: 0.6035 - val_loss: 0.6011 - val_accuracy: 0.7000
Epoch 2/5
1200/1200 [==============================] - 703s 586ms/step - loss: 0.8746
- accuracy: 0.7310 - val_loss: 0.9771 - val_accuracy: 0.7421
Epoch 3/5
1200/1200 [==============================] - 703s 586ms/step - loss: 0.7227
- accuracy: 0.7766 - val_loss: 0.7426 - val_accuracy: 0.7505
Epoch 4/5
1200/1200 [==============================] - 703s 586ms/step - loss: 0.6244
- accuracy: 0.8055 - val_loss: 0.4625 - val_accuracy: 0.7480
Epoch 5/5
1200/1200 [==============================] - 704s 587ms/step - loss: 0.5377
- accuracy: 0.8326 - val_loss: 0.6681 - val_accuracy: 0.7444
```

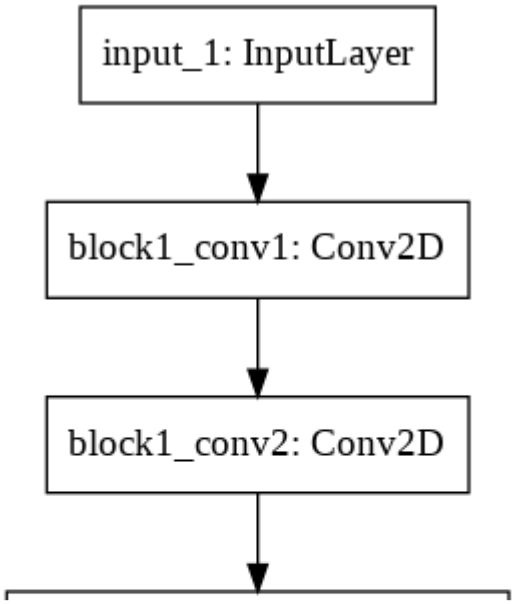
Out[19]:

```
<keras.callbacks.callbacks.History at 0x7f6a6041f2e8>
```

In [ ]:

```
# model graphs
tf.keras.utils.plot_model(
    model_1, to_file='model_1.png', show_shapes=False, show_layer_names=True,
    rankdir='TB', expand_nested=False, dpi=96
)
```

Out[20]:



In [ ]:

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

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

observations

- out of total parameters only 1/4 are trainable.
- after running only 5 epochs we get accuracy of .75 ,that is if we increase the epoch number we can increase accuracy and reduce loss.
- from histogram we can understand that after 5 epoch added layers forms same distribution as pretrained layers as the number of neurons in the layers decreases range of dsistribution(mean=0) of weight increases *after every epochs weight distribution reduces its ranges centered towards mean(standard deviation reduces).

In [ ]:

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

MODEL_2 (INPUT --> VGG-16 without Top layers(FC) --> 2 Conv Layers identical to FC --> Output Layer)

In [ ]:

```
#model_2
for layer in model.layers:
  layer.trainable = False
#Adding custom Layers
x = model.output
x = Conv2D(filters=4096,kernel_size=8 ,strides=1,activation="relu")(x)
x = Conv2D(filters=4096,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 [ ]:

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

```
Model: "model_4"
_________________________________________________________________
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_6 (Conv2D)            (None, 1, 1, 4096)        134221824
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 1, 1, 4096)        16781312
_________________________________________________________________
flatten_4 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_6 (Dense)              (None, 16)                65552
=================================================================
Total params: 165,783,376
Trainable params: 151,068,688
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [ ]:

```
#fitting model_2
model_2.fit_generator(train_generator,steps_per_epoch=train_steps, epochs=5,verbose=1,
                      validation_data=validation_generator,validation_steps=va
```

```
Epoch 1/5
1200/1200 [==============================] - 1410s 1s/step - loss: 1.4058 -
accuracy: 0.6052 - val_loss: 1.5813 - val_accuracy: 0.6809
Epoch 2/5
1200/1200 [==============================] - 1409s 1s/step - loss: 0.9071 -
accuracy: 0.7223 - val_loss: 1.2431 - val_accuracy: 0.7105
Epoch 3/5
1200/1200 [==============================] - 1410s 1s/step - loss: 0.7603 -
accuracy: 0.7663 - val_loss: 0.6085 - val_accuracy: 0.7309
Epoch 4/5
1200/1200 [==============================] - 1409s 1s/step - loss: 0.6576 -
accuracy: 0.7952 - val_loss: 0.8318 - val_accuracy: 0.7357
Epoch 5/5
1200/1200 [==============================] - 1409s 1s/step - loss: 0.5836 -
accuracy: 0.8188 - val_loss: 0.7888 - val_accuracy: 0.7315
```

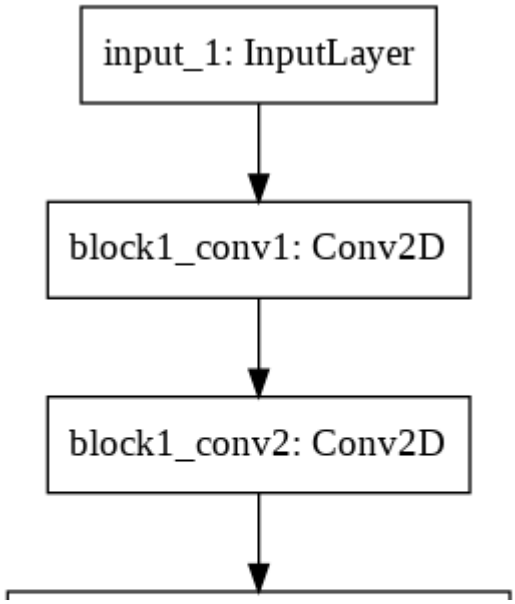
Out[32]:

```
<keras.callbacks.callbacks.History at 0x7f6a08762668>
```

In [ ]:

```
# 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[33]:



In [ ]:

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

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

observations

- number of trainable parameter large as compared to model_1 so time taken for each epoch is increases
- as number of trainable parameters increases and we use epoch number as previous model we got lesser accuracy and higher loss value(we got only .73 accuracy)
- in the conv2d_6 layer change of weights after every epoch is very small because that layer consist of large number of parameter.

In [ ]:

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

Model-3

In [ ]:

```
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 [ ]:

```
#model_3
#Adding custom Layers
x = model.output
x = Conv2D(filters=4096,kernel_size=8 ,strides=1,activation="relu")(x)
x = Conv2D(filters=4096,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
model_3.compile(loss="categorical_crossentropy",optimizer = 'Adam',metrics=['accuracy'
```

In [ ]:

```
1 model_3.summary()
```

```
Model: "model_5"
_________________________________________________________________
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_8 (Conv2D)            (None, 1, 1, 4096)        134221824
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 1, 1, 4096)        16781312
_________________________________________________________________
flatten_5 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_7 (Dense)              (None, 16)                65552
=================================================================
Total params: 165,783,376
Trainable params: 160,507,920
Non-trainable params: 5,275,456
_________________________________________________________________
```

In [39]:

```
#fitting model_3
model_3.fit_generator(train_generator,steps_per_epoch=train_steps, epochs=5,
                              validation_data=validation_generator,validation_steps=va
```

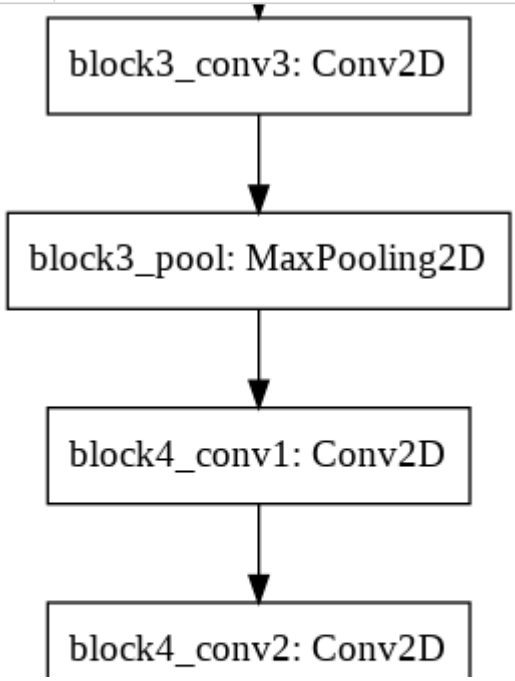
```
Epoch 1/5
1200/1200 [==============================] - 2155s 2s/step - loss: 2.8802 -
accuracy: 0.0624 - val_loss: 2.7756 - val_accuracy: 0.0601
Epoch 2/5
1200/1200 [==============================] - 2181s 2s/step - loss: 2.7728 -
accuracy: 0.0608 - val_loss: 2.7724 - val_accuracy: 0.0617
Epoch 3/5
1200/1200 [==============================] - 2155s 2s/step - loss: 2.7728 -
accuracy: 0.0615 - val_loss: 2.7754 - val_accuracy: 0.0584
Epoch 4/5
1200/1200 [==============================] - 2160s 2s/step - loss: 2.7728 -
accuracy: 0.0602 - val_loss: 2.7708 - val_accuracy: 0.0601
Epoch 5/5
1200/1200 [==============================] - 2162s 2s/step - loss: 2.7728 -
accuracy: 0.0597 - val_loss: 2.7729 - val_accuracy: 0.0601
```

Out[39]:

```
<keras.callbacks.callbacks.History at 0x7f6a0c669eb8>
```

In [40]:

```
# model graphs
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
)
```



In [42]:

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

```
Reusing TensorBoard on port 6006 (pid 5579), started 3:18:59 ago. (Use '!kil
l 5579' to kill it.)
```

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

observations

- we get only .0601 accuracy on model_3.
- from the histograms of last layers we understand that layers are not completely learned .
- low accuracy is because of model_3 consist of very large number of trainable parameters so that it need more epoch to attain more accuracy.
- as trainable parameters increases more epoch is needed to get good accuracy loss convergence.
- if we run the model for 25-30 epochs we get accuracy above 90

In [3]:

```
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:/
table.add_row([1, "model_1", "5", 0.7444])
table.add_row([2, "model_2","5", 0.7315])
table.add_row([3, "model_3","5", 0.0601])
print(table)
```

```
+-------+---------+------------------+---------------+
| Sl.N0 |  Model  | Number of epochs |  val_accuracy |
+-------+---------+------------------+---------------+
|   1   | model_1 |        5         |     0.7444    |
+-------+---------+------------------+---------------+
|   2   | model_2 |        5         |     0.7315    |
+-------+---------+------------------+---------------+
|   3   | model_3 |        5         |     0.0601    |
+-------+---------+------------------+---------------+
```

In [ ]:

```

```