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-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

Model-1

Model-2

Model-3

In [1]:

```
import tensorflow as tf
import os
```

```
import numpy as np
import pandas as pd
```

In [2]:

```
from tensorflow.keras.layers import
Dense,Input,Conv2D,MaxPool2D,Activation,Dropout,Flatten,GlobalAveragePooling2D
from tensorflow.keras.models import Model
import random as rn
```

In [3]:

```
!gdown --id 1Z4TyI7FcFVEx8qdl4jO9qxvxaqLSqoEu


get_ipython().system_raw("unrar x rvl-cdip.rar")
```

```
Downloading...
From: https://drive.google.com/uc?id=1Z4TyI7FcFVEx8qdl4jO9qxvxaqLSqoEu
To: /content/rvl-cdip.rar
4.66GB [01:25, 54.8MB/s]
```

In [4]:

```
os.listdir()
```

Out[4]:

```
['.config', 'data_final', 'rvl-cdip.rar', 'labels_final.csv', 'sample_data']
```

In [5]:

```
import pandas as pd

data = pd.read_csv('labels_final.csv') #reading the csv file
```

In [6]:

```
data.head()
```

Out[6]:

	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

In [7]:

```
from sklearn.model_selection import train_test_split


#traindf, validationdf = train_test_split(data, test_size=0.3, random_state=42,shuffle=True)

train_path, validation_path, train_label, validation_label = train_test_split(data['path'], data['l
abel'], test_size=0.2, random_state=42)
```

In [8]:

```
labels_dict = { 0 : 'letter',1: 'form',2: 'email',3 :'handwritten',4 :'advertisement',
                5 : 'scientific report',6 : 'scientific publication',7 :'specification',8 : 'file fo
lder',
                9 : 'news article', 10 : 'budget', 11 : 'invoice',12 : 'presentation',
```

```
13 : 'questionnaire', 14 : 'resume', 15 : 'memo'}
```

In [9]:

```
labels_dict.values()
```

Out[9]:

```
dict_values(['letter', 'form', 'email', 'handwritten', 'advertisement', 'scientific report', 'scie
ntific publication', 'specification', 'file folder', 'news article', 'budget', 'invoice',
'presentation', 'questionnaire', 'resume', 'memo'])
```

In [10]:

```
for subfolder_name in list(labels_dict.values()):
    os.makedirs(os.path.join('train_images', subfolder_name))
```

In [11]:

```
os.listdir('train_images')
```

Out[11]:

```
['advertisement',
 'handwritten',
 'memo',
 'form',
 'invoice',
 'email',
 'file folder',
 'scientific publication',
 'presentation',
 'budget',
 'news article',
 'questionnaire',
 'letter',
 'resume',
 'scientific report',
 'specification']
```

In [12]:

```
#https://thispointer.com/python-how-to-copy-files-from-one-location-to-another-using-shutil-copy/
import shutil
from tqdm import tqdm
for file, label in tqdm(zip(train_path, train_label)):
  shutil.copy('data_final/'+file,'train_images/'+labels_dict[label]+'/')
```

```
38400it [02:08, 299.08it/s]
```

In [13]:

```
for subfolder_name in list(labels_dict.values()):
    os.makedirs(os.path.join('validation_images', subfolder_name))
```

In [14]:

```
for file, label in tqdm(zip(validation_path, validation_label)):
    shutil.copy('data_final/'+file ,'validation_images/'+labels_dict[label]+'/')
```

```
9600it [00:34, 279.55it/s]
```

In [15]:

```
dir_path= 'train_images'
for i in os.listdir(dir_path):
    print("No of Images in ",i," category is ",len(os.listdir(os.path.join(dir_path,i))))
```

```
No of Images in  advertisement  category is  2393
No of Images in  handwritten  category is  2413
No of Images in  memo  category is  2401
No of Images in  form  category is  2407
No of Images in  invoice  category is  2397
No of Images in  email  category is  2379
No of Images in  file folder  category is  2351
No of Images in  scientific publication  category is  2373
No of Images in  presentation  category is  2428
No of Images in  budget  category is  2422
No of Images in  news article  category is  2392
No of Images in  questionnaire  category is  2395
No of Images in  letter  category is  2461
No of Images in  resume  category is  2415
No of Images in  scientific report  category is  2379
No of Images in  specification  category is  2391
```

In [16]:

```
dir_path= 'validation_images'
for i in os.listdir(dir_path):
    print("No of Images in ",i," category is ",len(os.listdir(os.path.join(dir_path,i))))
```

```
No of Images in  advertisement  category is  601
No of Images in  handwritten  category is  592
No of Images in  memo  category is  595
No of Images in  form  category is  587
No of Images in  invoice  category is  595
No of Images in  email  category is  614
No of Images in  file folder  category is  652
No of Images in  scientific publication  category is  612
No of Images in  presentation  category is  578
No of Images in  budget  category is  580
No of Images in  news article  category is  609
No of Images in  questionnaire  category is  612
No of Images in  letter  category is  554
No of Images in  resume  category is  590
No of Images in  scientific report  category is  620
No of Images in  specification  category is  609
```

In [17]:

```
dir_path= 'train_images'
ImageFlow = tf.keras.preprocessing.image.ImageDataGenerator()
ImageGenerator_train =
ImageFlow.flow_from_directory(dir_path,target_size=(224,224),seed=10,batch_size=32)
```

```
Found 38397 images belonging to 16 classes.
```

In [18]:

```
dir_path='validation_images'
ImageGenerator_validation =
ImageFlow.flow_from_directory(dir_path,target_size=(224,224),seed=10,batch_size=32)
```

```
Found 9600 images belonging to 16 classes.
```

In [19]:

```
##Checking time taken to load images.
import time
start = time.time()
total_batches = 0

batches = 0
per_batch = 32
for x_batch, y_batch in ImageGenerator_train:
    batches += 1
    if batches >= 6899/per_batch:
        total_batches = total_batches + batches
        break
```

```
end = time.time()
duration = end-start
print("{} batches: {} s".format(total_batches, duration))
print("{:0.5f} Images/s".format(per_batch*total_batches/duration))
```

```
216 batches: 34.13801717758179 s
202.47222 Images/s
```

1. Use VGG-16 pretrained network without Fully Connected layers and initilize all the weigh ts 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 --> Max pool Layer --> 2 FC layers --> Output Layer**
4. Train only new Conv block, FC layers, output layer. Don't train the VGG-16 network.

In [20]:

```
import os
os.environ['PYTHONHASHSEED'] = '0'

##https://keras.io/getting-started/faq/#how-can-i-obtain-reproducible-results-using-keras-during-development
## Have to clear the session. If you are not clearing, Graph will create again and again and graph size will increses.
## Varibles will also set to some value from before session
tf.keras.backend.clear_session()

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


#VGG-16
base_model=tf.keras.applications.VGG16(weights='imagenet', include_top=False,input_shape=(224,224,3))


# add a global spatial average pooling layer
x=base_model.output

#print(x)

# first: train only the top layers (which were randomly initialized)
# i.e. freeze all convolutional InceptionV3 layers
for layer in base_model.layers:
     layer.trainable = False

#base_model.trainable=False

#Conv Layer
Conv1 = Conv2D(filters=32,kernel_size=(3,3),strides=(1,1),padding='valid',data_format='channels_last',
               activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=0),name='Cconv1')(x)
#MaxPool Layer
Pool1 = MaxPool2D(pool_size=(2,2),strides=(2,2),padding='valid',data_format='channels_last',name='Pool1')(Conv1)

flatten = Flatten(data_format='channels_last',name='Flatten')(Pool1)

#FC layer
FC1 = Dense(units=64,activation='relu',kernel_initializer=tf.keras.initializers.glorot_normal(seed=32),name='FC1')(flatten)

#FC layer
FC2 = Dense(units=32,activation='relu',kernel_initializer=tf.keras.initializers.glorot_normal(seed=33),name='FC2')(FC1)

#output layer
```

```
#Output layer
Out =
Dense(units=16,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_normal(seed=3)
,name='Output')(FC2)

#Creating a model
model_1 = Model(inputs=base_model.input,outputs=Out)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 0s 0us/step
```

In [21]:

```
model_1.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
_________________________________________________________________
Conv1 (Conv2D)               (None, 5, 5, 32)          147488
_________________________________________________________________
Pool1 (MaxPooling2D)         (None, 2, 2, 32)          0
_________________________________________________________________
Flatten (Flatten)            (None, 128)               0
_________________________________________________________________
FC1 (Dense)                  (None, 64)                8256
_________________________________________________________________
FC2 (Dense)                  (None, 32)                2080
_________________________________________________________________
Output (Dense)               (None, 16)                528
=================================================================
Total params: 14,873,040
Trainable params: 158,352
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [22]:

```
model_1.compile(optimizer='rmsprop', loss='categorical_crossentropy',metrics=["accuracy"])
```

In [23]:

```
history=model_1.fit(ImageGenerator_train,

validation_data=ImageGenerator_validation,epochs=3,steps_per_epoch=len(ImageGenerator_train),valida
tion_steps=len(ImageGenerator_validation))
```

```
Epoch 1/3
1200/1200 [==============================] - 277s 203ms/step - loss: 2.2030 - accuracy: 0.3689 - v
al_loss: 1.5048 - val_accuracy: 0.5334
Epoch 2/3
1200/1200 [==============================] - 222s 185ms/step - loss: 1.4179 - accuracy: 0.5628 - v
al_loss: 1.4221 - val_accuracy: 0.5724
Epoch 3/3
1200/1200 [==============================] - 188s 157ms/step - loss: 1.2840 - accuracy: 0.6057 - v
al_loss: 1.4315 - val_accuracy: 0.5847
```

In [24]:

```
history.history
```

Out[24]:

```
{'accuracy': [0.45391565561294556, 0.5649399757385254, 0.6053076982498169],
 'loss': [1.7981104850769043, 1.4176541566848755, 1.3013049364089966],
 'val_accuracy': [0.5334374904632568, 0.5723958611488342, 0.5846874713897705],
 'val_loss': [1.5048243999481201, 1.422074317932129, 1.4315128326416016]}
```

In [25]:

```
from keras.preprocessing.image import ImageDataGenerator
```

In [ ]:

```
df=pd.read_csv('./labels_final.csv',dtype=str)
traindf, validationdf = train_test_split(df, test_size=0.3, random_state=42,shuffle=True)
```

```
Found 0 validated image filenames belonging to 0 classes.
Found 0 validated image filenames belonging to 0 classes.
```

```
/usr/local/lib/python3.7/dist-packages/keras_preprocessing/image/dataframe_iterator.py:282:
UserWarning: Found 33600 invalid image filename(s) in x_col="path". These filename(s) will be
ignored.
  .format(n_invalid, x_col)
/usr/local/lib/python3.7/dist-packages/keras_preprocessing/image/dataframe_iterator.py:282:
UserWarning: Found 33600 invalid image filename(s) in x_col="path". These filename(s) will be
ignored.
  .format(n_invalid, x_col)
```

In [ ]:

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

train_generator=datagen.flow_from_dataframe(dataframe=traindf,directory='./train_images/',x_col="p
ath",
y_col="label",batch_size=32,seed=42,shuffle=False,class_mode="categorical",target_size=(224,224,3),
subset='training')

valid_generator=datagen.flow_from_dataframe(dataframe=traindf,directory='./validation_images/',x_c
ol="path",
y_col="label",batch_size=32,seed=42,shuffle=False,class_mode="categorical",target_size=(224,224,3),
subset='validation')

# Define your model
```

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

#
model.fit(train_generator,validation_data=validation_generator,epochs=10,steps_per_epoch=len(train_
rator),validation_steps=len(validation_generator))
```

1. Use VGG-16 pretrained network without Fully Connected layers and initilize all the weigh ts 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 ex actly 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 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 VG G-16 network.

In [39]:

```
import os
os.environ['PYTHONHASHSEED'] = '0'


tf.keras.backend.clear_session()

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


#VGG-16
base_model_2=tf.keras.applications.VGG16(weights='imagenet', include_top=False,input_shape=(224,224
,3))
print(base_model_2.input)

# add a global spatial average pooling layer
x=base_model_2.output

for layer in base_model_2.layers:
    layer.trainable = False



Conv1 = Conv2D(4096,kernel_size=[7,7],strides=(1,1),padding='valid', activation='relu',name='Conv1'
)(x)

Conv2 = Conv2D(4096,kernel_size=[1,1],strides=(1,1),padding='valid',data_format='channels_last',
               activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=0),name='C
onv2')(Conv1)

flatten = Flatten(data_format='channels_last',name='Flatten')(Conv2)
output =
Dense(units=16,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_normal(seed=3)
,name='Output')(flatten)

# #Creating a model
model_2 = Model(inputs=base_model_2.input,outputs=output)
```

```
KerasTensor(type_spec=TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name='input_1'), nam
e='input_1', description="created by layer 'input_1'")
```

In [40]:

```
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
_________________________________________________________________
Conv1 (Conv2D)               (None, 1, 1, 4096)        102764544
_________________________________________________________________
Conv2 (Conv2D)               (None, 1, 1, 4096)        16781312
_________________________________________________________________
Flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
Output (Dense)               (None, 16)                65552
=================================================================
Total params: 134,326,096
Trainable params: 119,611,408
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [41]:

```
model_2.compile(optimizer='rmsprop', loss='categorical_crossentropy',metrics=['accuracy'])

history_2=model_2.fit(ImageGenerator_train,

validation_data=ImageGenerator_validation,epochs=3,steps_per_epoch=len(ImageGenerator_train),valida
tion_steps=len(ImageGenerator_validation))
```

```
Epoch 1/3
1200/1200 [==============================] - 209s 173ms/step - loss: 21.8210 - accuracy: 0.3893 -
val_loss: 1.4671 - val_accuracy: 0.6195
Epoch 2/3
1200/1200 [==============================] - 207s 173ms/step - loss: 1.2502 - accuracy: 0.6540 - v
al_loss: 1.2839 - val_accuracy: 0.6566
Epoch 3/3
1200/1200 [==============================] - 207s 173ms/step - loss: 1.0990 - accuracy: 0.7062 - v
```

al_loss: 1.2574 - val_accuracy: 0.6760

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.

MODEL 3

In [43]:

```
import os
os.environ['PYTHONHASHSEED'] = '0'


tf.keras.backend.clear_session()

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


#VGG-16
base_model_3=tf.keras.applications.VGG16(weights='imagenet', include_top=False,input_shape=(224,224
,3))


# add a global spatial average pooling layer
x=base_model_3.output



for layer in base_model_3.layers[:13]:
  layer.trainable = False


# # #Conv Layer
Conv1 = Conv2D(4096,kernel_size=[7,7],strides=(1,1),padding='valid', activation='relu',name='Conv1'
)(x)

Conv2 = Conv2D(4096,kernel_size=[1,1],strides=(1,1),padding='valid',data_format='channels_last',
              activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=0),name='C
onv2')(Conv1)
flatten = Flatten(data_format='channels_last',name='Flatten')(Conv2)
output =
Dense(units=16,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_normal(seed=3)
,name='Output')(flatten)

# #Creating a model
model_3 = Model(inputs=base_model_3.input,outputs=output)
```

In [44]:

```
model_3.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
_________________________________________________________________
Conv1 (Conv2D)                (None, 1, 1, 4096)        102764544
_________________________________________________________________
Conv2 (Conv2D)                (None, 1, 1, 4096)        16781312
_________________________________________________________________
Flatten (Flatten)             (None, 4096)              0
_________________________________________________________________
Output (Dense)                (None, 16)                65552
=================================================================
Total params: 134,326,096
Trainable params: 129,050,640
Non-trainable params: 5,275,456
_________________________________________________________________
```

In [45]:

```
model_3.compile(optimizer='rmsprop', loss='categorical_crossentropy',metrics=['accuracy'])

history_3=model_3.fit(ImageGenerator_train,

validation_data=ImageGenerator_validation,epochs=3,steps_per_epoch=len(ImageGenerator_train),valida
tion_steps=len(ImageGenerator_validation))
```

```
Epoch 1/3
1200/1200 [==============================] - 284s 236ms/step - loss: 227.2454 - accuracy: 0.0603 -
val_loss: 2.7729 - val_accuracy: 0.0604
Epoch 2/3
1200/1200 [==============================] - 284s 236ms/step - loss: 2.7727 - accuracy: 0.0593 - v
al_loss: 2.7731 - val_accuracy: 0.0577
Epoch 3/3
1200/1200 [==============================] - 284s 236ms/step - loss: 2.7728 - accuracy: 0.0641 - v
al_loss: 2.7731 - val_accuracy: 0.0577
```

In [46]:

```
history_3.history
```

Out[46]:

```
{'accuracy': [0.06013490632176399, 0.0614110492169857, 0.06370289623737335],
 'loss': [39.831398010253906, 2.772803544998169, 2.7727930545806885],
 'val_accuracy': [0.06041666492819786,
  0.05770833417773247,
  0.05770833417773247],
 'val_loss': [2.772860527038574, 2.773062229156494, 2.7731451988220215]}
```

MODEL 1 Accuracy

In [50]:

```
history.history['accuracy'][2]*100
```

Out[50]:

```
60.53076982498169
```

MODEL 2 accuracy

In [51]:

```
history_2.history['accuracy'][2]*100
```

Out[51]:

```
70.18256783485413
```

MODEL 3 accuracy

In [52]:

```
history_3.history['accuracy'][2]*100
```

Out[52]:

```
6.370289623737335
```

The Last model Accuracy was low because we tried to train the weight of the vgg model with our own data and thus it didnt perform well.