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 fu nction, image augmentation, any hyperparameters. but you ha ve to use the same architechture what we are asking below.

5. Use tensorboard for every model and analyse your gradien ts. (you need to upload the screenshots for each model for evaluation)

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

6. You can check about Transfer Learning in this link - [htt ps://blog.keras.io/building-powerful-image-classification-m odels-using-very-little-data.html (https://blog.keras.io/building-powerful-image-classificati on-models-using-very-little-data.html)](https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html)

In [1]:
```
!curl --header 'Host: storage.googleapis.com' --user-agent 'Mozilla
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time
Time  Current
                                 Dload  Upload   Total   Spent
Left  Speed
100 4440M  100 4440M    0     0  51.7M      0  0:01:25  0:01:25 --
:--:-- 29.0M
```

In [ ]:
```
!unzip data_final.zip -d data_final
```

In [3]:
```
%tensorflow_version 2.x
```

In [4]:
```
import tensorflow as tf
import os
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
```

In [6]:
```
data = pd.read_csv('labels_final.csv')
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]:
```
df = data.groupby('label').tail(2000).reset_index(drop=True)
```

In [8]:
```
df.shape
```

Out[8]:
```
(32000, 2)
```

In [9]:
```
df['label'].value_counts()
```

Out[9]:
```
15    2000
14    2000
13    2000
12    2000
11    2000
10    2000
9     2000
8     2000
7     2000
6     2000
5     2000
4     2000
3     2000
2     2000
1     2000
0     2000
Name: label, dtype: int64
```

In [10]:
```
df.head()
```

Out[10]:

	path	label
0	imagest/t/r/e/tre03e00/2042692622_2042692636.tif	13
1	imagesp/p/n/o/pno92e00/2045613028_2045613030.tif	13
2	imagesa/a/p/d/apd03e00/2042789021_2042789038.tif	13
3	imagesv/v/w/n/vwn30e00/87563496.tif	6
4	imagesu/u/b/w/ubw39c00/2505223727.tif	6

In [11]:
```
final = df.sample(frac=1).reset_index(drop=True)
final['label'] = final['label'].apply(str)
```

In [12]:
```
final.head()
```

Out[12]:

	path	label
0	imagesu/u/s/b/usb91a00/1003537721.tif	9
1	imagesf/f/z/x/fzx53a00/1001896119_6122.tif	5
2	imagesa/a/j/a/aja94c00/96383603.tif	0
3	imagesv/v/h/m/vhm79e00/2050743741.tif	13
4	imagesk/k/y/n/kyn20a00/10167827_10167833.tif	6

In [13]:
```
from keras.preprocessing.image import ImageDataGenerator

train = final[:19200]
cv = final[19200:25600]
test = final[25600:]
```

In [14]:
```
imageFlow = ImageDataGenerator(rescale=1./255)
```

In [15]:
```
train_gen = imageFlow.flow_from_dataframe(train, directory="/conten
                                          classes=None, class_mode=
cv_gen = imageFlow.flow_from_dataframe(cv, directory="/content/data
                                          classes=None, class_mode=
test_gen = imageFlow.flow_from_dataframe(test, directory="/content/
                                          classes=None, class_mode=
```

```
Found 19200 validated image filenames belonging to 16 classes.
Found 6400 validated image filenames belonging to 16 classes.
Found 6400 validated image filenames belonging to 16 classes.
```

Transfer 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 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.

In [16]:
```
from keras import applications, callbacks
from tensorflow.keras.layers import Dense,Input,Conv2D,MaxPool2D,Ac
from keras.models import Model
import random as rn
```

In [17]:
```
model = applications.VGG16(weights = "imagenet", include_top=False,
model.summary()
```

```
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_tf_kernels_notop.h5)
58892288/58889256 [==============================] - 0s 0us/step
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 156, 256, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 156, 256, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 156, 256, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 78, 128, 64)       0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 78, 128, 128)      73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 78, 128, 128)      147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 39, 64, 128)       0
```

```
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 39, 64, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 39, 64, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 39, 64, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 19, 32, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 19, 32, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 19, 32, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 19, 32, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 16, 512)        0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 16, 512)        2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 16, 512)        2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 16, 512)        2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 8, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

In [18]:
```
for layer in model.layers:
    layer.trainable = False
    # print(layer.name)

#adding custom layers
x = model.output

#Conv Layer
Conv1 = Conv2D(filters=32,kernel_size=(3,3),strides=(1,1),padding='
               activation='relu',kernel_initializer=tf.keras.initial
#MaxPool Layer
Pool1 = MaxPool2D(pool_size=(2,2),strides=(2,2),padding='valid',dat

#flatten before adding FC
x = Flatten()(Pool1)

#2 FC layer
fc1 = Dense(256, activation="relu")(x)
fc2 = Dense(256, activation="relu")(fc1)

predictions = Dense(16, activation="softmax")(fc2)

# creating the final model
model_final = Model(model.input, predictions)

# compile the model
model_final.compile(loss = "categorical_crossentropy", optimizer =
```

In [19]:
```
model_final.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 156, 256, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 156, 256, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 156, 256, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 78, 128, 64)       0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 78, 128, 128)      73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 78, 128, 128)      147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 39, 64, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 39, 64, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 39, 64, 256)       590080
_________________________________________________________________
```

```
block3_conv3 (Conv2D)        (None, 39, 64, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 19, 32, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 19, 32, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 19, 32, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 19, 32, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 16, 512)        0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 16, 512)        2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 16, 512)        2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 16, 512)        2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 8, 512)         0
_________________________________________________________________
Conv1 (Conv2D)               (None, 2, 6, 32)          147488
_________________________________________________________________
Pool1 (MaxPooling2D)         (None, 1, 3, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 96)                0
_________________________________________________________________
dense (Dense)                (None, 256)               24832
_________________________________________________________________
dense_1 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_2 (Dense)              (None, 16)                4112
=================================================================
Total params: 14,956,912
Trainable params: 242,224
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [20]:
```
history=model_final.fit_generator(train_gen, steps_per_epoch=600, e
```

```
Epoch 1/3
600/600 [==============================] - 132s 207ms/step - loss:
1.8984 - accuracy: 0.4009 - val_loss: 1.3349 - val_accuracy: 0.598
0
Epoch 2/3
600/600 [==============================] - 97s 162ms/step - loss:
1.2557 - accuracy: 0.6172 - val_loss: 1.2291 - val_accuracy: 0.627
5
Epoch 3/3
600/600 [==============================] - 91s 152ms/step - loss:
1.0868 - accuracy: 0.6673 - val_loss: 1.1736 - val_accuracy: 0.645
5
```

In [21]:
```
score = model_final.evaluate_generator(test_gen, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
Test score: 1.1859544515609741
Test accuracy: 0.6417187452316284
```

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.

In [22]:
```
imageFlow = ImageDataGenerator(rescale=1./255)
```

In [23]:
```
train_gen = imageFlow.flow_from_dataframe(train, directory="/conten
                                          classes=None, class_mode=
cv_gen = imageFlow.flow_from_dataframe(cv, directory="/content/data
                                          classes=None, class_mode=
test_gen = imageFlow.flow_from_dataframe(test, directory="/content/
                                          classes=None, class_mode=
```

```
Found 19200 validated image filenames belonging to 16 classes.
Found 6400 validated image filenames belonging to 16 classes.
Found 6400 validated image filenames belonging to 16 classes.
```

In [24]:
```
model2 = applications.VGG16(weights = "imagenet", include_top=False
model2.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (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
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

In [25]:
```
for layer in model2.layers[:13]:
    # layer.trainable = False
    print(layer.name)
```

```
input_2
block1_conv1
block1_conv2
block1_pool
block2_conv1
block2_conv2
block2_pool
block3_conv1
block3_conv2
block3_conv3
block3_pool
block4_conv1
block4_conv2
```

In [26]:
```
for layer in model2.layers:
    layer.trainable = False


#adding custom layers
x = model2.output

#Conv Layers
Conv1 = Conv2D(4096, kernel_size=[7,7], strides=(1,1), padding='val

Conv2 = Conv2D(4096, kernel_size=[1,1], strides=(1,1), padding='val

#flatten before output
x = Flatten()(Conv2)

#output layer
predictions = Dense(16, activation="softmax")(x)

# creating the final model
model2_final = Model(model2.input, predictions)

# compile the model
model2_final.compile(loss = "categorical_crossentropy", optimizer =
```

In [27]:
```
model2_final.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
```

```
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
conv2d (Conv2D)              (None, 1, 1, 4096)        102764544
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 1, 1, 4096)        16781312
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_3 (Dense)              (None, 16)                65552
=================================================================
Total params: 134,326,096
Trainable params: 119,611,408
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [28]:
```
history=model2_final.fit_generator(train_gen, steps_per_epoch=600,
```

```
Epoch 1/3
600/600 [==============================] - 193s 318ms/step - loss:
2.8943 - accuracy: 0.4394 - val_loss: 1.2508 - val_accuracy: 0.619
1
Epoch 2/3
600/600 [==============================] - 191s 318ms/step - loss:
1.0299 - accuracy: 0.6823 - val_loss: 1.0883 - val_accuracy: 0.674
5
Epoch 3/3
600/600 [==============================] - 190s 317ms/step - loss:
0.8419 - accuracy: 0.7379 - val_loss: 1.0230 - val_accuracy: 0.692
7
```

In [29]:
```
score = model2_final.evaluate_generator(test_gen, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
Test score: 1.0471042394638062
Test accuracy: 0.6887500286102295
```

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 [30]:
```
for layer in model2_final.layers[:13]:
    layer.trainable = False
```

In [32]:
```
# stopping = callbacks.EarlyStopping(monitor='val_accuracy')
# compile the model
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.

model2_final.compile(loss = "categorical_crossentropy", optimizer =
history=model2_final.fit_generator(train_gen, steps_per_epoch=600,
```

```
Epoch 1/3
600/600 [==============================] - 191s 318ms/step - loss:
0.7549 - accuracy: 0.7639 - val_loss: 1.0614 - val_accuracy: 0.703
1
Epoch 2/3
600/600 [==============================] - 190s 317ms/step - loss:
0.6212 - accuracy: 0.8038 - val_loss: 1.0824 - val_accuracy: 0.705
0
Epoch 3/3
600/600 [==============================] - 190s 317ms/step - loss:
0.5538 - accuracy: 0.8255 - val_loss: 1.0929 - val_accuracy: 0.716
9
```

In [33]:
```
score = model2_final.evaluate_generator(test_gen, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
Test score: 1.126118540763855
Test accuracy: 0.7098437547683716
```