Transfer Learning Assignment

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

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

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

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

#!gdown --id 1Z4TyI7FcFVEx8qdl4jO9qxvxaqLSqoEu

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

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

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

```
'''#import all the required libraries
import tensorflow as tf
import os
import numpy as np
import pandas as pd

df=pd.read_csv('labels_final.csv',dtype=str)'''
```

{"type":"string"}

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

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

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

1. 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.

2. Use tensorboard for every model and analyse your gradients. (you need to upload the screenshots for each model for evaluation)

3. You can check about Transfer Learning in this link - https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

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

1. Do print model.summary() and draw model_plots for each of the model.

Import Section

```
import matplotlib.pyplot as plt # importing the libraries
import pandas as pd
import numpy as np
import os
import datetime
import seaborn as sns
import tensorflow as tf
import datetime, os
from tensorflow import keras
from keras_preprocessing.image import ImageDataGenerator
from keras.layers import Input, Lambda, Dense, Flatten , Conv2D,
MaxPool2D
from tensorflow.keras.callbacks import
ModelCheckpoint,EarlyStopping,LearningRateScheduler,ReduceLROnPlateau,
TensorBoard
from keras.models import Model
from keras.applications.vgg16 import VGG16,preprocess_input
from keras.preprocessing import image
from keras.callbacks import Callback,TensorBoard
from prettytable import PrettyTable
from prettytable import ALL as ALL
#!pip install -U --no-cache-dir gdown --pre
from tensorflow.keras.utils import plot_model
import warnings
warnings.filterwarnings('ignore')
```

Lets Pull the Dataset

```
!pip install -q kaggle
from google.colab import files
files.upload()
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

```
!chmod 600 /root/.kaggle/kaggle.json
!mkdir dataset
! kaggle datasets download -d brahma0545/aaic-assignment-tl
! unzip /content/aaic-assignment-tl.zip -d dataset
```

```
dataframe=pd.read_csv("/content/dataset/labels_final.csv")
```

```
dataframe.head()
```

```
                                          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
```

```
labels_dict={ 0 :"letter",1 :"form",2 :"email",3 :"handwritten",4 :"ad
vertisement",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"}
```

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

```
                                          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
```

Lets Create Data Generator

```
datagenerator = ImageDataGenerator(rescale=1/255.,
validation_split=0.2) #image generator
```

```
print("------TRAIN DATA-------")   # train data
train_generator =
datagenerator.flow_from_dataframe(dataframe=dataframe,
directory="/content/dataset/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=7)
```

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

```python
print("------TEST DATA-------") # cross validation data
test_generator =
datagenerator.flow_from_dataframe(dataframe=dataframe,
directory="/content/dataset/data_final",
                                             x_col='path',
                                             y_col='label',
                                             target_size=(256,256),
                                             class_mode='categorical',
                                             batch_size=32,
                                             subset='validation',
                                             seed=7)
```

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

```python
result_dataset = pd.DataFrame(data = np.zeros((3,2)),index =
['Model_1','Model_2','Model_3'],columns =
['Model_Loss','Model_Accuracy'])
result_dataset
```

	Model_Loss	Model_Accuracy
Model_1	0.0	0.0
Model_2	0.0	0.0
Model_3	0.0	0.0

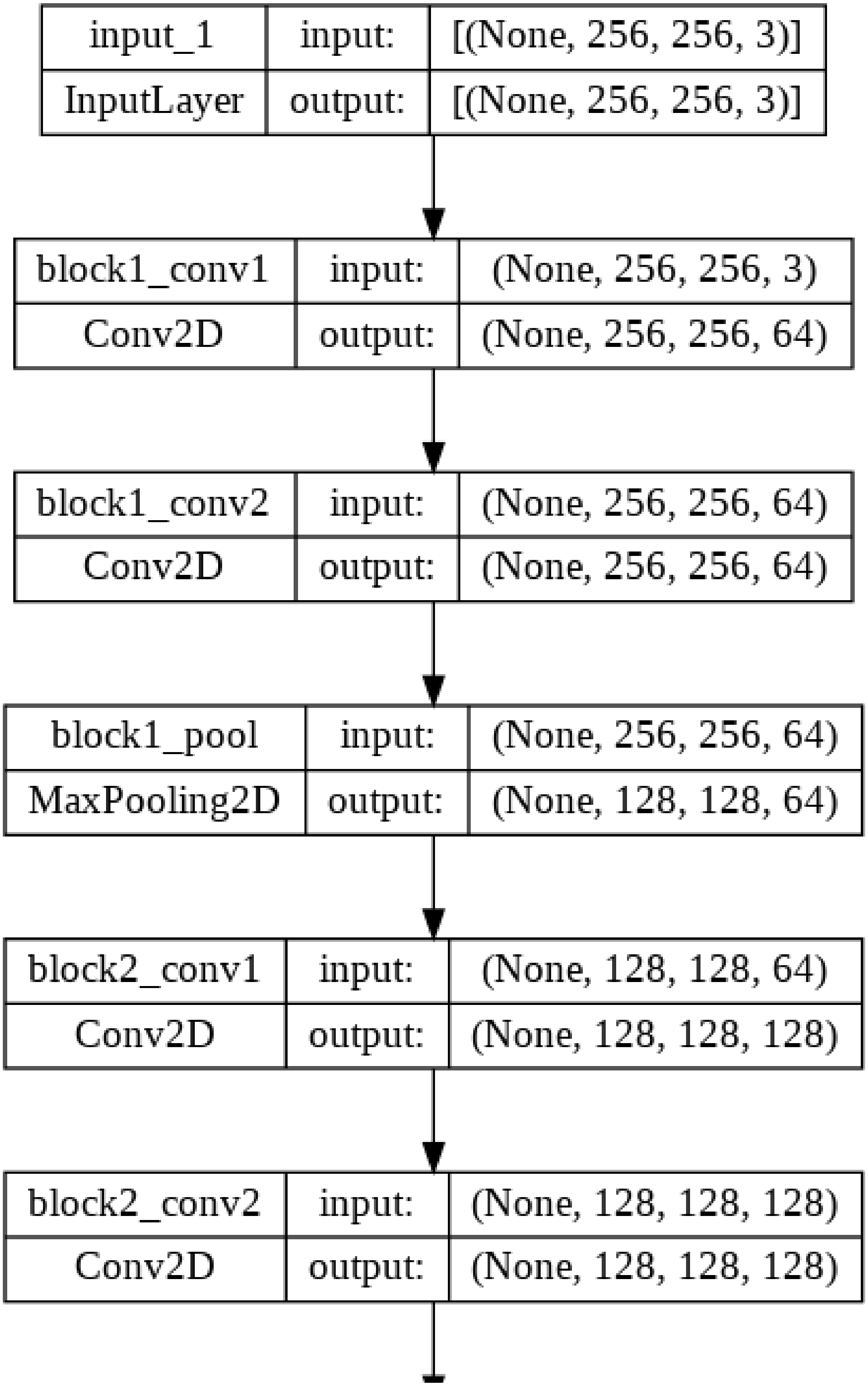
Model-1

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

vgg_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
_________________________________________________________________
```

```
plot_model(vgg_model,show_shapes = True)
```


input_1
InputLayer
input:
output:
[(None, 256, 256, 3)]
[(None, 256, 256, 3)]
block1_conv1
Conv2D
input:
output:
(None, 256, 256, 3)
(None, 256, 256, 64)
block1_conv2
Conv2D
input:
output:
(None, 256, 256, 64)
(None, 256, 256, 64)
block1_pool
MaxPooling2D
input:
output:
(None, 256, 256, 64)
(None, 128, 128, 64)
block2_conv1
Conv2D
input:
output:
(None, 128, 128, 64)
(None, 128, 128, 128)
block2_conv2
Conv2D
input:
output:
(None, 128, 128, 128)
(None, 128, 128, 128)

Lets Build Model 1

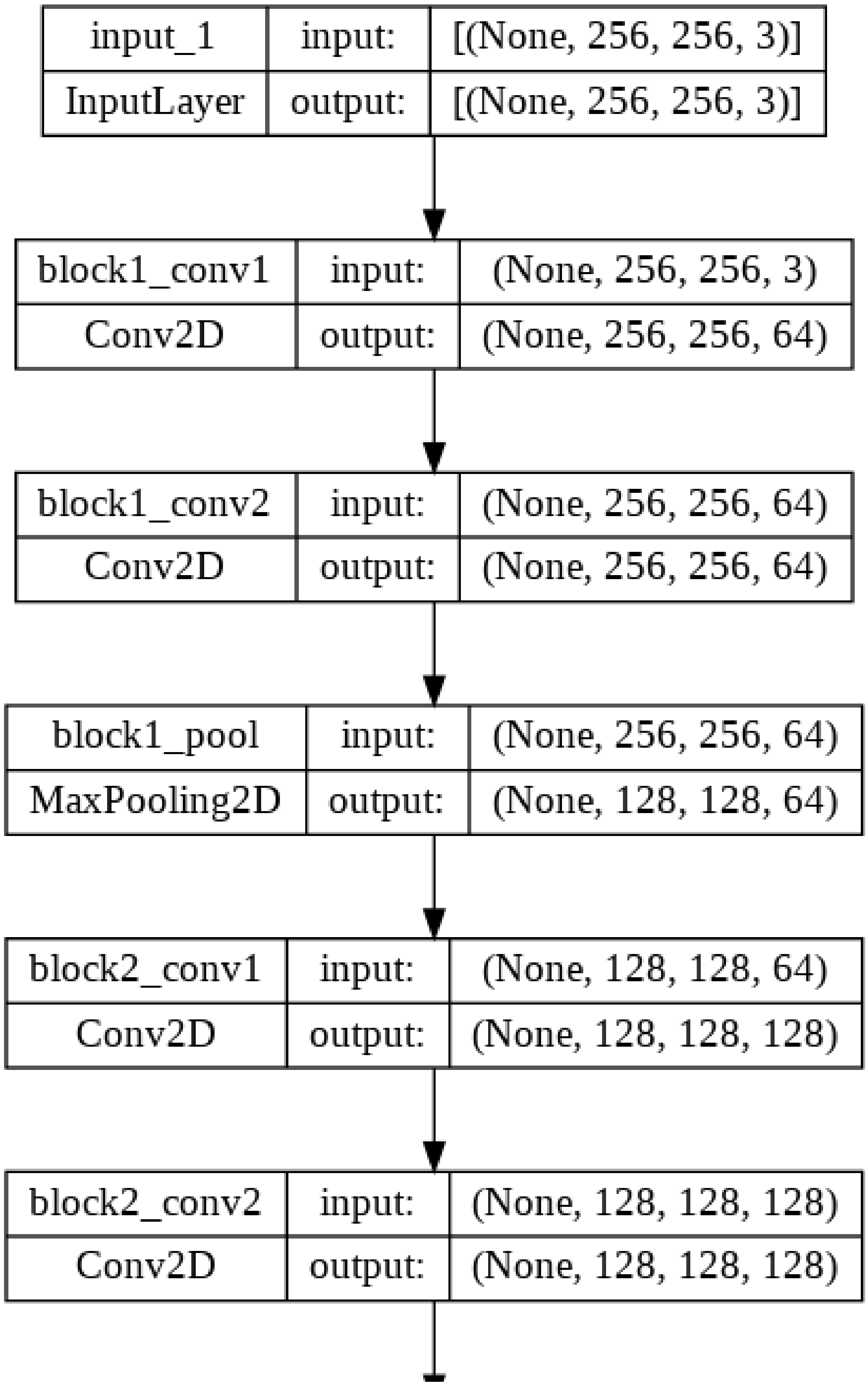
- MODEL_1(INPUT --> VGG-16 without Top layers(FC) --> Conv Layer --> Maxpool Layer --> 2 FC layers --> Output Layer)

```
# lets make vgg model layers non trainable
for layer in vgg_model.layers:
  layer.trainable = False
#Adding custom Layers
vgg_out = vgg_model.output
conv2d_1 =
Conv2D(filters=512,kernel_size=(3,3),padding="same",kernel_initializer
='he_normal',activation="relu")(vgg_out)
maxpool_1 = MaxPool2D(2,2)(conv2d_1)
flatten_1 = Flatten()(maxpool_1)
dense_1 = Dense(256, activation="relu",kernel_initializer
='he_normal')(flatten_1)
dense_2 = Dense(128, activation="relu",kernel_initializer
='he_normal')(dense_1)
output = Dense(16, activation="softmax")(dense_2)
# creating the final model
model_1 = Model(inputs = vgg_model.input, outputs = output)
# compile the model
model_1.compile(loss = "categorical_crossentropy", optimizer ='Adam',
metrics=["accuracy"])

# summary of the model_1
model_1.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_9 (Conv2D)           (None, 8, 8, 512)         2359808

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 512)        0
 2D)

 flatten_5 (Flatten)         (None, 8192)              0

 dense_7 (Dense)             (None, 256)               2097408

 dense_8 (Dense)             (None, 128)               32896

 dense_9 (Dense)             (None, 16)                2064

=================================================================
Total params: 19,206,864
Trainable params: 4,492,176
Non-trainable params: 14,714,688
_________________________________________________________________
```

```
#lets plot model 1
plot_model(model_1,to_file='model_1.png',show_shapes = True)
```


input_1
input:
[(None, 256, 256, 3)]
InputLayer
output:
[(None, 256, 256, 3)]
block1_conv1
input:
(None, 256, 256, 3)
Conv2D
output:
(None, 256, 256, 64)
block1_conv2
input:
(None, 256, 256, 64)
Conv2D
output:
(None, 256, 256, 64)
block1_pool
input:
(None, 256, 256, 64)
MaxPooling2D
output:
(None, 128, 128, 64)
block2_conv1
input:
(None, 128, 128, 64)
Conv2D
output:
(None, 128, 128, 128)
block2_conv2
input:
(None, 128, 128, 128)
Conv2D
output:
(None, 128, 128, 128)

```
# HyperParameters
numberofepochs = 5
batch_size = 32 #note should be same as imagedatagenerator

filepath="model_1_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath,
monitor='val_accuracy', verbose=1, save_best_only=True, mode='auto')

earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0.01,
patience=3, verbose=1)

# Load the TensorBoard notebook extension
%load_ext tensorboard
log_dir = os.path.join("logs",'fits',
datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback =
tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1,write_
graph=True)
%reload_ext tensorboard

reduce_lr = ReduceLROnPlateau(monitor='val_loss',
factor=0.2,patience=5, min_lr=0.000001)

#fitting the model_1
model_1.fit_generator(train_generator,epochs=numberofepochs,validation
_data=test_generator,callbacks=[checkpoint,earlystop,reduce_lr,tensorb
oard_callback])
```

```
Epoch 1/5
1200/1200 [==============================] - ETA: 0s - loss: 1.2606 -
accuracy: 0.6137
Epoch 1: val_accuracy improved from -inf to 0.68667, saving model to
model_1_save/weights-01-0.6867.hdf5
1200/1200 [==============================] - 321s 255ms/step - loss:
1.2606 - accuracy: 0.6137 - val_loss: 1.0319 - val_accuracy: 0.6867 -
lr: 0.0010
Epoch 2/5
1200/1200 [==============================] - ETA: 0s - loss: 0.8603 -
accuracy: 0.7344
Epoch 2: val_accuracy improved from 0.68667 to 0.70781, saving model
to model_1_save/weights-02-0.7078.hdf5
1200/1200 [==============================] - 296s 246ms/step - loss:
0.8603 - accuracy: 0.7344 - val_loss: 0.9810 - val_accuracy: 0.7078 -
lr: 0.0010
Epoch 3/5
1200/1200 [==============================] - ETA: 0s - loss: 0.7237 -
accuracy: 0.7761
Epoch 3: val_accuracy improved from 0.70781 to 0.75104, saving model
to model_1_save/weights-03-0.7510.hdf5
1200/1200 [==============================] - 296s 247ms/step - loss:
0.7237 - accuracy: 0.7761 - val_loss: 0.8553 - val_accuracy: 0.7510 -
```

```
lr: 0.0010
Epoch 4/5
1200/1200 [==============================] - ETA: 0s - loss: 0.6082 -
accuracy: 0.8094
Epoch 4: val_accuracy improved from 0.75104 to 0.76448, saving model
to model_1_save/weights-04-0.7645.hdf5
1200/1200 [==============================] - 296s 247ms/step - loss:
0.6082 - accuracy: 0.8094 - val_loss: 0.8355 - val_accuracy: 0.7645 -
lr: 0.0010
Epoch 5/5
1200/1200 [==============================] - ETA: 0s - loss: 0.5162 -
accuracy: 0.8361
Epoch 5: val_accuracy improved from 0.76448 to 0.76875, saving model
to model_1_save/weights-05-0.7688.hdf5
1200/1200 [==============================] - 297s 247ms/step - loss:
0.5162 - accuracy: 0.8361 - val_loss: 0.8470 - val_accuracy: 0.7688 -
lr: 0.0010
```

```
<keras.callbacks.History at 0x7ff7f4e94370>
```

```
model1_score = model_1.evaluate(test_generator, verbose=1)
print('Test loss:', model1_score[0])
print('Test accuracy:', model1_score[1])
result_dataset['Model_Loss']['Model_1'] = model1_score[0]
result_dataset['Model_Accuracy']['Model_1'] = model1_score[1]
```

```
300/300 [==============================] - 58s 189ms/step - loss:
0.8470 - accuracy: 0.7688
Test loss: 0.8469867706298828
Test accuracy: 0.768750011920929
```

```
result_dataset
```

	Model_Loss	Model_Accuracy
Model_1	0.846987	0.76875
Model_2	0.000000	0.00000
Model_3	0.000000	0.00000

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

```
Output hidden; open in https://colab.research.google.com to view.
```

Model-2

Lets Build Model 2

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

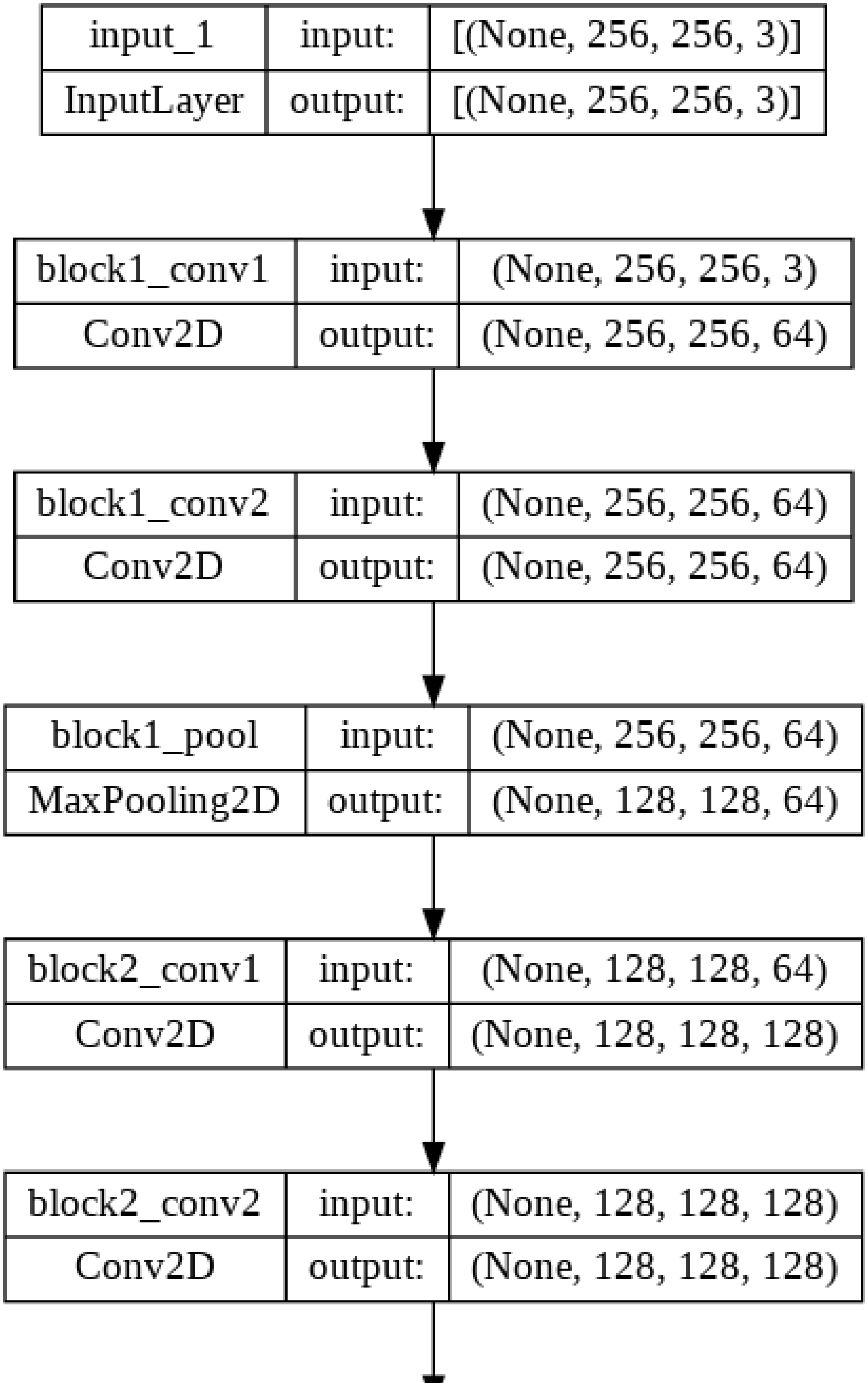
```
#model_2
for layer in vgg_model.layers:
  layer.trainable = False
```

```python
#Adding custom Layers
vgg_model_output = vgg_model.output
conv2d_1 =
Conv2D(filters=256,kernel_size=8 ,strides=1,kernel_initializer =
'he_normal',activation="relu")(vgg_model_output)
conv2d_2 =
Conv2D(filters=128,kernel_size=1 ,strides=1,kernel_initializer =
'he_normal',activation="relu")(conv2d_1)
flatten_1 = Flatten()(conv2d_2)
# creating the final model
output= Dense(16, activation="softmax")(flatten_1)
model_2 = Model(inputs = vgg_model.input, outputs = output)
# compile the model
model_2.compile(loss="categorical_crossentropy",optimizer =
'Adam',metrics=['accuracy'])

# summary of the model_2
model_2.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_4 (Conv2D)           (None, 1, 1, 256)         8388864

 conv2d_5 (Conv2D)           (None, 1, 1, 128)         32896

 flatten_2 (Flatten)         (None, 128)               0

 dense_2 (Dense)             (None, 16)                2064

=================================================================
Total params: 23,138,512
Trainable params: 8,423,824
Non-trainable params: 14,714,688
_________________________________________________________________
```

```
plot_model(model_2,to_file = 'model_2.png',show_shapes = True)
```


input_1
input:
[(None, 256, 256, 3)]
InputLayer
output:
[(None, 256, 256, 3)]
block1_conv1
input:
(None, 256, 256, 3)
Conv2D
output:
(None, 256, 256, 64)
block1_conv2
input:
(None, 256, 256, 64)
Conv2D
output:
(None, 256, 256, 64)
block1_pool
input:
(None, 256, 256, 64)
MaxPooling2D
output:
(None, 128, 128, 64)
block2_conv1
input:
(None, 128, 128, 64)
Conv2D
output:
(None, 128, 128, 128)
block2_conv2
input:
(None, 128, 128, 128)
Conv2D
output:
(None, 128, 128, 128)

```python
filepath="model_2_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath,
monitor='val_accuracy', verbose=1, save_best_only=True, mode='auto')

#fitting the model_2
model_2.fit_generator(train_generator,epochs=numberofepochs,validation
_data=test_generator,callbacks=[checkpoint,earlystop,reduce_lr,tensorb
oard_callback])
```

```
Epoch 1/5
1200/1200 [==============================] - ETA: 0s - loss: 1.2592 -
accuracy: 0.6249
Epoch 1: val_accuracy improved from -inf to 0.68313, saving model to
model_2_save/weights-01-0.6831.hdf5
1200/1200 [==============================] - 325s 260ms/step - loss:
1.2592 - accuracy: 0.6249 - val_loss: 1.0398 - val_accuracy: 0.6831 -
lr: 0.0010
Epoch 2/5
1200/1200 [==============================] - ETA: 0s - loss: 0.8692 -
accuracy: 0.7333
Epoch 2: val_accuracy improved from 0.68313 to 0.72260, saving model
to model_2_save/weights-02-0.7226.hdf5
1200/1200 [==============================] - 314s 261ms/step - loss:
0.8692 - accuracy: 0.7333 - val_loss: 0.9323 - val_accuracy: 0.7226 -
lr: 0.0010
Epoch 3/5
1200/1200 [==============================] - ETA: 0s - loss: 0.7271 -
accuracy: 0.7765
Epoch 3: val_accuracy improved from 0.72260 to 0.74406, saving model
to model_2_save/weights-03-0.7441.hdf5
1200/1200 [==============================] - 313s 260ms/step - loss:
0.7271 - accuracy: 0.7765 - val_loss: 0.8823 - val_accuracy: 0.7441 -
lr: 0.0010
Epoch 4/5
1200/1200 [==============================] - ETA: 0s - loss: 0.6209 -
accuracy: 0.8063
Epoch 4: val_accuracy improved from 0.74406 to 0.74542, saving model
to model_2_save/weights-04-0.7454.hdf5
1200/1200 [==============================] - 310s 258ms/step - loss:
0.6209 - accuracy: 0.8063 - val_loss: 0.9209 - val_accuracy: 0.7454 -
lr: 0.0010
Epoch 5/5
1200/1200 [==============================] - ETA: 0s - loss: 0.5350 -
accuracy: 0.8292
Epoch 5: val_accuracy did not improve from 0.74542
1200/1200 [==============================] - 309s 258ms/step - loss:
0.5350 - accuracy: 0.8292 - val_loss: 0.9859 - val_accuracy: 0.7440 -
lr: 0.0010

<keras.callbacks.History at 0x7f176ce63df0>
```

```
model2_score = model_2.evaluate(test_generator, verbose=1)
print('Test loss:', model2_score[0])
print('Test accuracy:', model2_score[1])
result_dataset['Model_Loss']['Model_2'] = model2_score[0]
result_dataset['Model_Accuracy']['Model_2'] = model2_score[1]
```

```
300/300 [==============================] - 56s 187ms/step - loss:
0.9209 - accuracy: 0.7454
Test loss: 0.9209381341934204
Test accuracy: 0.7454166412353516
```

```
result_dataset
```

```
         Model_Loss  Model_Accuracy
Model_1    0.846987        0.768750
Model_2    0.920938        0.745417
Model_3    0.000000        0.000000
```

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

```
Output hidden; open in https://colab.research.google.com to view.
```

```
tf.keras.backend.clear_session
```

```
<function keras.backend.clear_session()>
```

Model-3

Lets Build Model-3

- '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.

```
# lets set last 6 layers trainable
for layer in vgg_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
```

```
#model_3
#Adding custom Layers
vgg_output = vgg_model.output
conv2d_1 =
Conv2D(filters=256,kernel_size=8,strides=1,kernel_initializer =
'he_normal',activation="relu")(vgg_output)
conv2d_2 =
```

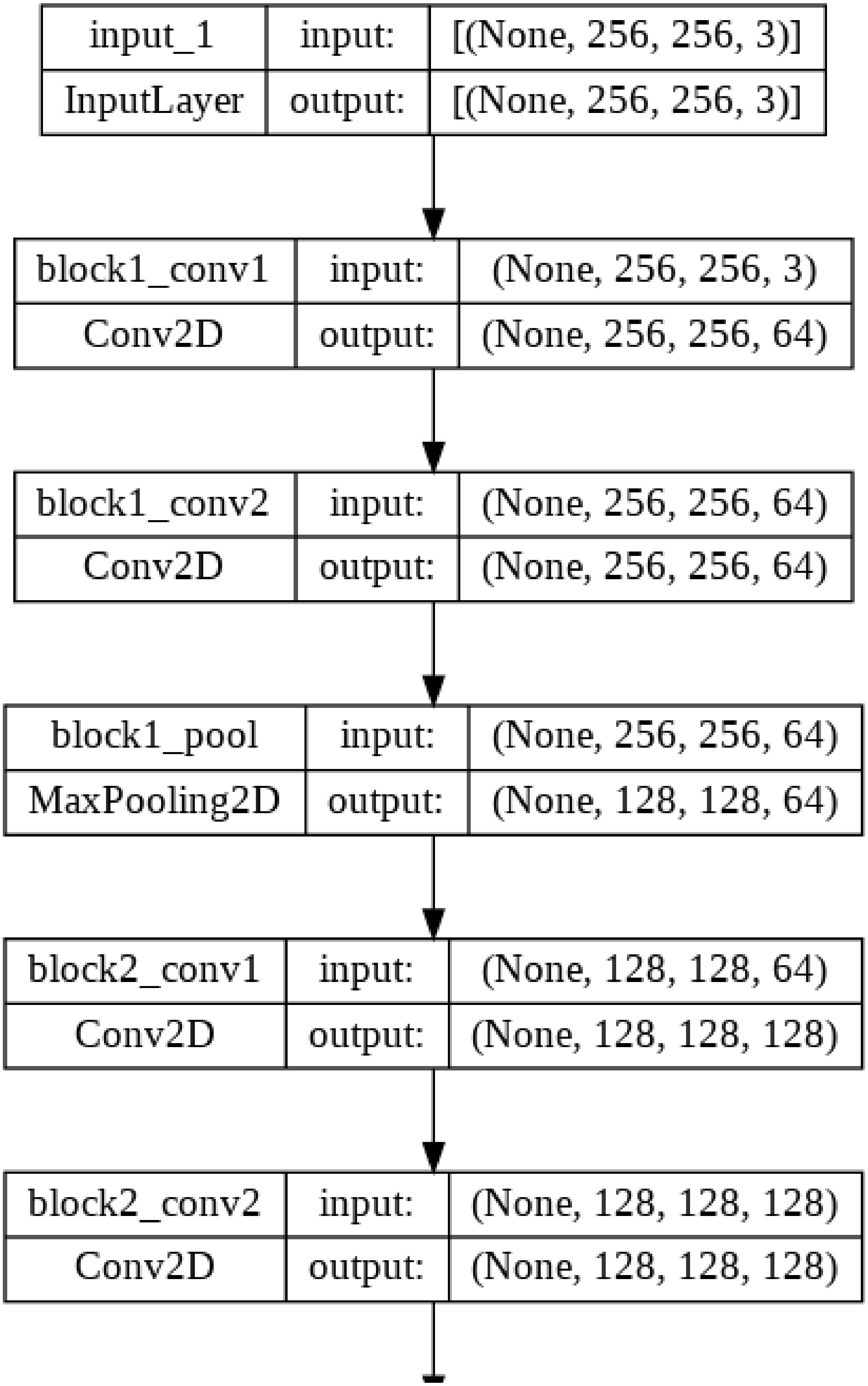
```python
Conv2D(filters=128,kernel_size=1 ,strides=1,kernel_initializer =
'he_normal',activation="relu")(conv2d_1)
flatten_1 = Flatten()(conv2d_2)
# creating the final model
output = Dense(16, activation="softmax")(flatten_1)
model_3 = Model(inputs = vgg_model.input, outputs = output)
# compile the model
model_3.compile(loss="categorical_crossentropy",optimizer =
'Adam',metrics=['accuracy'])
```

```python
model_3.summary()
```

```
Model: "model_3"

_________________________________________________________________
 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, 256)         8388864

 conv2d_7 (Conv2D)           (None, 1, 1, 128)         32896

 flatten_3 (Flatten)         (None, 128)               0

 dense_3 (Dense)             (None, 16)                2064

=================================================================
Total params: 23,138,512
Trainable params: 17,863,056
Non-trainable params: 5,275,456
_________________________________________________________________
```

```
plot_model(model_3,to_file = 'model_3.png',show_shapes = True)
```


input_1
input:
[(None, 256, 256, 3)]
InputLayer
output:
[(None, 256, 256, 3)]
block1_conv1
input:
(None, 256, 256, 3)
Conv2D
output:
(None, 256, 256, 64)
block1_conv2
input:
(None, 256, 256, 64)
Conv2D
output:
(None, 256, 256, 64)
block1_pool
input:
(None, 256, 256, 64)
MaxPooling2D
output:
(None, 128, 128, 64)
block2_conv1
input:
(None, 128, 128, 64)
Conv2D
output:
(None, 128, 128, 128)
block2_conv2
input:
(None, 128, 128, 128)
Conv2D
output:
(None, 128, 128, 128)

```
filepath="model_3_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath,
monitor='val_accuracy', verbose=1, save_best_only=True, mode='auto')

#fitting the model_3
model_3.fit_generator(train_generator,epochs=numberofepochs,validation
_data=test_generator,callbacks=[checkpoint,earlystop,reduce_lr,tensorb
oard_callback])
```

```
Epoch 1/5
1200/1200 [==============================] - ETA: 0s - loss: 2.7925 -
accuracy: 0.0608
Epoch 1: val_accuracy improved from -inf to 0.05844, saving model to
model_3_save/weights-01-0.0584.hdf5
1200/1200 [==============================] - 411s 341ms/step - loss:
2.7925 - accuracy: 0.0608 - val_loss: 2.7729 - val_accuracy: 0.0584 -
lr: 0.0010
Epoch 2/5
1200/1200 [==============================] - ETA: 0s - loss: 2.7729 -
accuracy: 0.0607
Epoch 2: val_accuracy did not improve from 0.05844
1200/1200 [==============================] - 355s 295ms/step - loss:
2.7729 - accuracy: 0.0607 - val_loss: 2.7730 - val_accuracy: 0.0584 -
lr: 0.0010
Epoch 3/5
1200/1200 [==============================] - ETA: 0s - loss: 2.7728 -
accuracy: 0.0594
Epoch 3: val_accuracy did not improve from 0.05844
1200/1200 [==============================] - 355s 296ms/step - loss:
2.7728 - accuracy: 0.0594 - val_loss: 2.7732 - val_accuracy: 0.0584 -
lr: 0.0010
Epoch 4/5
1200/1200 [==============================] - ETA: 0s - loss: 2.7728 -
accuracy: 0.0620
Epoch 4: val_accuracy improved from 0.05844 to 0.06115, saving model
to model_3_save/weights-04-0.0611.hdf5
1200/1200 [==============================] - 355s 296ms/step - loss:
2.7728 - accuracy: 0.0620 - val_loss: 2.7730 - val_accuracy: 0.0611 -
lr: 0.0010
Epoch 4: early stopping

<keras.callbacks.History at 0x7f16e66519a0>
```

```
model3_score = model_3.evaluate(test_generator, verbose=1)
print('Test loss:', model3_score[0])
print('Test accuracy:', model3_score[1])
result_dataset['Model_Loss']['Model_3'] = model3_score[0]
result_dataset['Model_Accuracy']['Model_3'] = model3_score[1]
```

```
300/300 [==============================] - 58s 190ms/step - loss:
2.7730 - accuracy: 0.0611
```

```
Test loss: 2.7729671001434326
Test accuracy: 0.0611458346247673
```

```
result_dataset
```

```
         Model_Loss  Model_Accuracy
Model_1    0.846987        0.768750
Model_2    0.920938        0.745417
Model_3    2.772967        0.061146
```

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

```
Output hidden; open in https://colab.research.google.com to view.
```

Observations

Please write your observations or a brief summary of the results that you get after performing transfer learning with reference to model1, model2 and model3

```
from tabulate import tabulate
print(tabulate(result_dataset, headers='keys', tablefmt='psql'))
```

```
+---------+--------------+------------------+
|         |   Model_Loss |   Model_Accuracy |
|---------+--------------+------------------|
| Model_1 |     0.846987 |        0.76875   |
| Model_2 |     0.920938 |        0.745417  |
| Model_3 |     2.77297  |        0.0611458 |
+---------+--------------+------------------+
```

Observations

model 1:

- We can see that transfer learning using top layers have very good accuracy as the weights are well converged imagenet weights. And accuracy is around 77% percentage for 5 epochs, if we increase the epochs we will get very good accuracy in few epochs.and we have around 19 million weights for this model.and train time it is around 300 seconds.

##model 2:

- here we are using transfer learning using top layers with conv layers instead of dense layers to retain the spatial information .And accuracy is around 74.5% percentage for 5 epochs, if we increase the epochs we will get very good accuracy in few epochs.and we have around 23 million parameters for this model.which increased when compared to model 1.and train time increased to 325 seconds.

##model 3:

- here we are using transfer learning using top layers and last six layers trainable with conv layers instead of dense layers to retain the spatial information .And accuracy is around 6.11% percentage for 5 epochs as we are re taining six layers from vgg 16 it takes time to converge for our dataset .and trainable parameters increased to 23.1 million parameters for this model.which increased when compared to model 2.and train time increased to 355 seconds.