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:

    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
```

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

In [ ]:
```
#the dataset that you are dealing with is quite large 3.7 GB and hence there are tw
# Method 1- you can use gdown module to get the data directly from Google drive to
# the syntax is as follows !gdown --id file_id , for ex - running the below cell wi
```

In [ ]:
```
#!gdown --id 1Z4TyI7FcFVEx8qdL4jO9qxvxaqLSqoEu
```

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

In [ ]:
```
#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.

In [1]:
```
#import all the required libraries
```

```python
import tensorflow as tf
import os
import numpy as np
import pandas as pd
from tensorflow.keras.layers import MaxPool2D, Conv2D, Flatten, Dense
from tensorflow.keras import Model
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
import datetime
import os
import random as rn
from tensorflow.keras.utils import plot_model

%load_ext tensorboard
```

In [2]:
```python
df=pd.read_csv('labels_final.csv',dtype=str)
```

In [3]:
```python
df.head()
```

Out[3]:

	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 [4]:
```python
df.shape
```

Out[4]:
```
(48000, 2)
```

In [5]:
```python
# https://vijayabhaskar96.medium.com/tutorial-on-keras-flow-from-dataframe-1fd4493d
ImageFlow = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255.,validat



train_generator=ImageFlow.flow_from_dataframe(dataframe=df,
directory="./data_final/",
x_col="path",
y_col="label",
subset="training",
batch_size=64,
seed=0,
shuffle=True,
class_mode="categorical",
target_size=(128,128))

validation_generator=ImageFlow.flow_from_dataframe(dataframe=df,
directory="./data_final/",
x_col="path",
y_col="label",
subset="validation",
batch_size=64,
```

```
seed=0,
shuffle=True,
class_mode="categorical",
target_size=(128,128))
```

```
Found 36000 validated image filenames belonging to 16 classes.
Found 12000 validated image filenames belonging to 16 classes.
```

In [6]:
```
train_steps_per_epoch = np.ceil(36000/64)
validation_steps_per_epoch = np.ceil(12000/64)
```

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

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

4. You are free to choose Learning rate, optimizer, loss function, image augmentation, any hyperparameters. but you have to use the same architechture what we are asking below.

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

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

https://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

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

Model-1

1. Use VGG-16 pretrained network without Fully Connected layers and initilize all the weights with Imagenet trained weights.
2. After VGG-16 network without FC layers, add a new Conv block (1 Conv layer and 1 Maxpooling), 2 FC layers and an output layer to classify 16 classes. You are free to choose any hyperparameters/parameters of conv block, FC layers, output layer.
3. Final architecture will be **INPUT --> VGG-16 without Top**

layers(FC) --> Conv Layer --> Maxpool Layer --> 2 FC layers --> Output Layer
4.Print model.summary() and plot the architecture of the model.
Reference for plotting model
5. Train only new Conv block, FC layers, output layer. Don't train the VGG-16 network.

In [7]:
```
os.environ['PYTHONHASHSEED'] = '0'

##https://keras.io/getting-started/faq/#how-can-i-obtain-reproducible-results-using
## Have to clear the session. If you are not clearing, Graph will create again and
## 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)
tf.random.set_seed(0)
```

In [9]:
```
vgg_model = tf.keras.applications.VGG16(
    include_top=False,
    weights="imagenet",
    input_shape=(128,128,3),
)
vgg_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         [(None, 128, 128, 3)]     0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 128, 128, 64)      1792      
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 128, 128, 64)      36928     
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 64, 64, 64)        0         
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 64, 64, 128)       73856     
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 64, 64, 128)       147584    
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 32, 32, 128)       0         
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 32, 32, 256)       295168    
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 32, 32, 256)       590080    
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 32, 32, 256)       590080    
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 16, 16, 256)       0         
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 16, 16, 512)       1180160   
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 16, 16, 512)       2359808   
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 16, 16, 512)       2359808   
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 8, 8, 512)         0         
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 8, 8, 512)         2359808   
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 8, 8, 512)         2359808   
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 8, 8, 512)         2359808   
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0         
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

In [9]:

```python
#https://blog.keras.io/building-powerful-image-classification-models-using-very-lit
for layer in vgg_model.layers:
    layer.trainable = False
input_from_vgg = vgg_model.output
conv1 = Conv2D(filters = 512, kernel_size = (3,3), padding = 'same',
               activation = 'relu', kernel_initializer='HeUniform') (input_from_vgg
max_pool1 = MaxPool2D(pool_size=(2, 2)) (conv1)
flat = Flatten() (max_pool1)
dense_1 = Dense(256, activation = 'relu', kernel_initializer='HeUniform') (flat)
```

```
dense_2 = Dense(128, activation = 'relu', kernel_initializer='HeUniform') (dense_1)
output = Dense(16,activation='softmax')(dense_2)

model_one = Model(inputs = vgg_model.input, outputs = output)
model_one.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 128, 128, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 128, 128, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 128, 128, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 64, 64, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 64, 64, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 64, 64, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 32, 32, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 32, 32, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 32, 32, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 32, 32, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 16, 16, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 16, 16, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 8, 8, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
conv2d (Conv2D)              (None, 4, 4, 512)         2359808
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 2, 2, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 256)               524544
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 16)                2064
=================================================================
Total params: 17,634,000
Trainable params: 2,919,312
```

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

In [10]:

```python
filepath="model_save/best_model_one.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy',  verbose=1,

earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0.01, patience=2, verbo

tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir= ("logs/fits/"+dateti
                                                      histogram_freq=1,write_graph=

call_back_list = [ earlystop, checkpoint, tensorboard_callback]
```

In [11]:

```python
model_one.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accur

model_one.fit( train_generator,steps_per_epoch = train_steps_per_epoch,epochs=3,
                        validation_data = validation_generator, validation_steps =
                        callbacks=[call_back_list])
```

```
Epoch 1/3
563/563 [==============================] - 178s 300ms/step - loss: 1.5444 - accura
cy: 0.5238 - val_loss: 1.2278 - val_accuracy: 0.6213

Epoch 00001: val_accuracy improved from -inf to 0.62133, saving model to model_sav
e\best_model_one.hdf5
Epoch 2/3
563/563 [==============================] - 150s 266ms/step - loss: 1.1656 - accura
cy: 0.6388 - val_loss: 1.1875 - val_accuracy: 0.6358

Epoch 00002: val_accuracy improved from 0.62133 to 0.63583, saving model to model_
save\best_model_one.hdf5
Epoch 3/3
563/563 [==============================] - 149s 265ms/step - loss: 1.0246 - accura
cy: 0.6807 - val_loss: 1.1538 - val_accuracy: 0.6527

Epoch 00003: val_accuracy improved from 0.63583 to 0.65267, saving model to model_
save\best_model_one.hdf5
```

Out[11]:

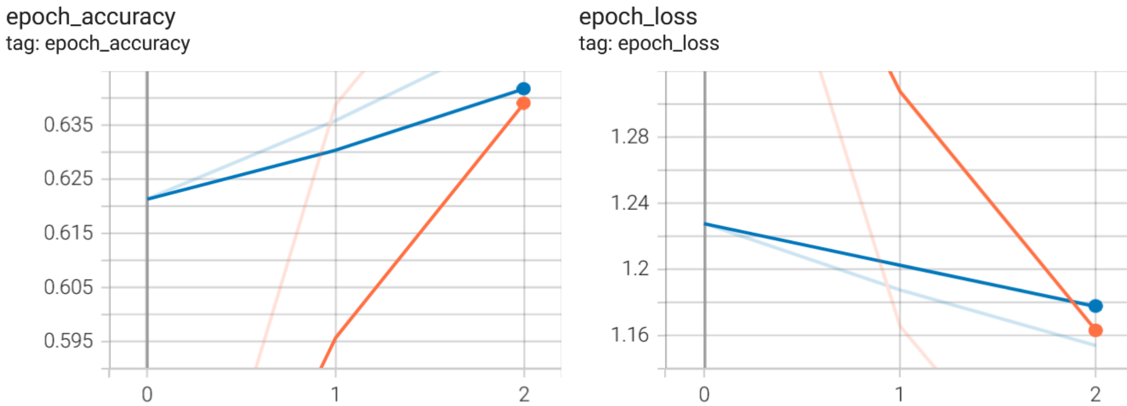
```
<keras.callbacks.History at 0x1fe081a2c40>
```

In [52]:

```
%tensorboard --logdir logs/fits
```

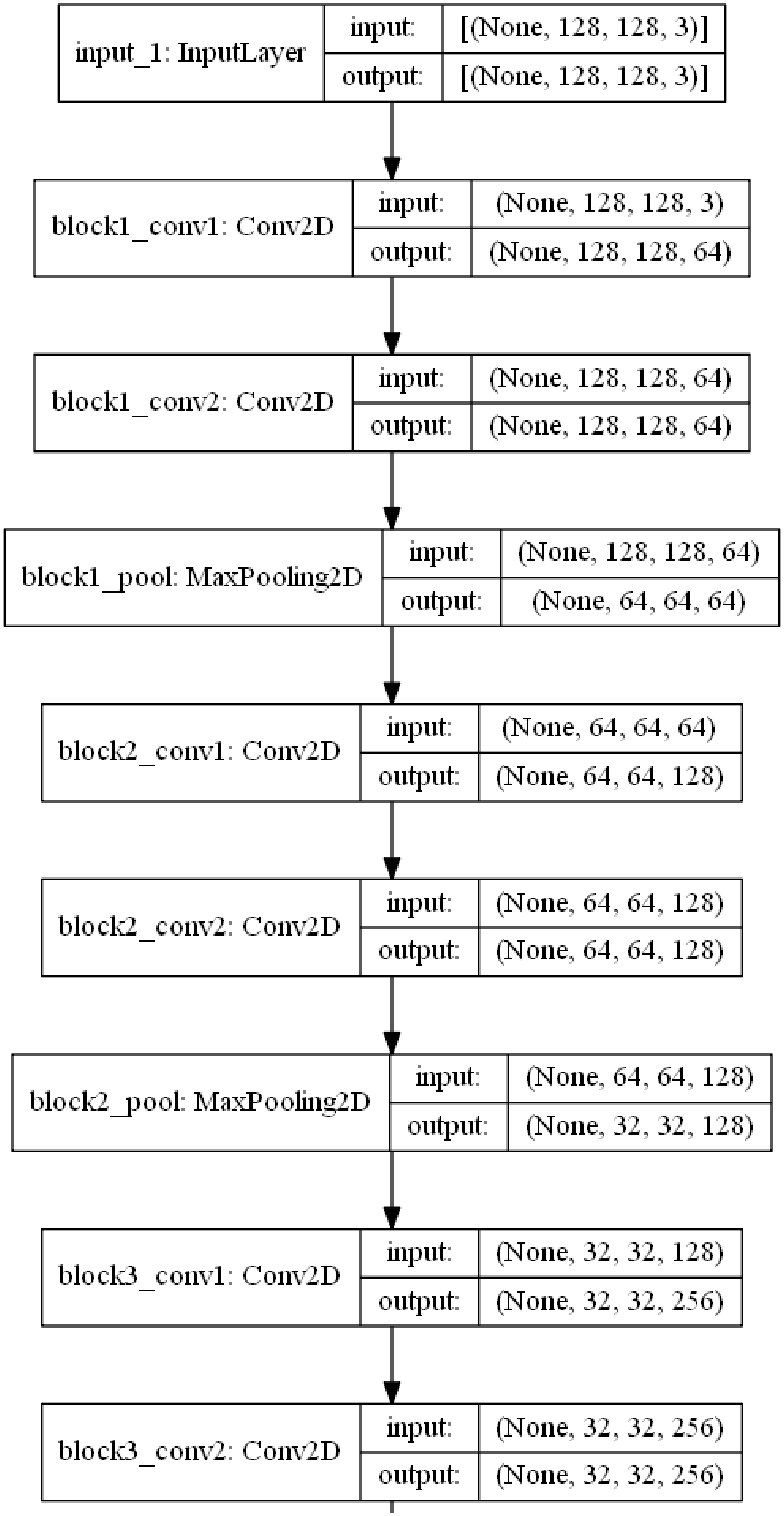
```
Reusing TensorBoard on port 6006 (pid 880), started 0:01:06 ago. (Use '!kill 880'
to kill it.)
```

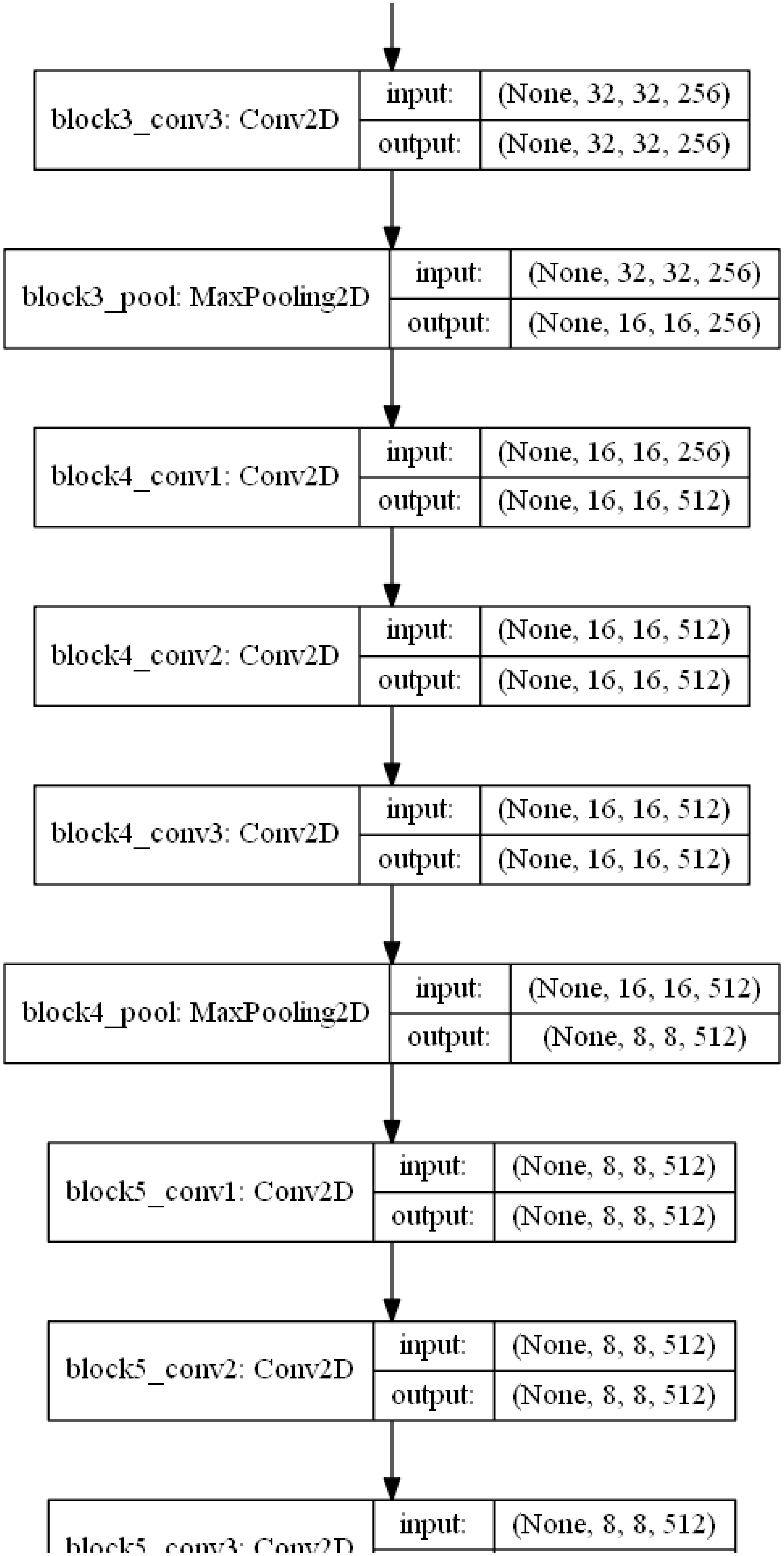
model_one Tensorboard accuracy and loss



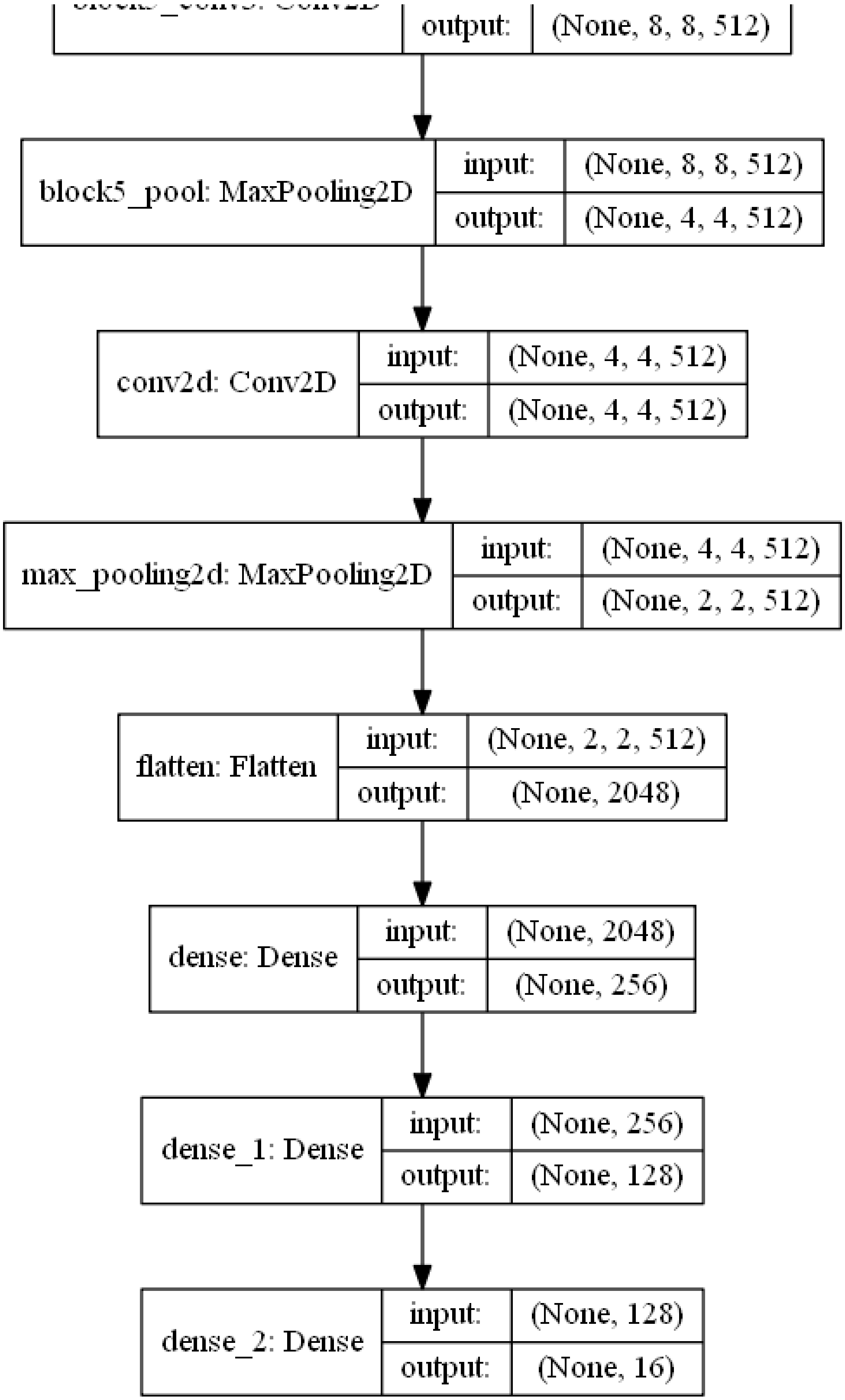
In [16]:
```
plot_model(model_one, to_file='model_one.png', show_shapes=True)
```

Out[16]:


input_1: InputLayer
input:
[(None, 128, 128, 3)]
output:
[(None, 128, 128, 3)]
block1_conv1: Conv2D
input:
(None, 128, 128, 3)
output:
(None, 128, 128, 64)
block1_conv2: Conv2D
input:
(None, 128, 128, 64)
output:
(None, 128, 128, 64)
block1_pool: MaxPooling2D
input:
(None, 128, 128, 64)
output:
(None, 64, 64, 64)
block2_conv1: Conv2D
input:
(None, 64, 64, 64)
output:
(None, 64, 64, 128)
block2_conv2: Conv2D
input:
(None, 64, 64, 128)
output:
(None, 64, 64, 128)
block2_pool: MaxPooling2D
input:
(None, 64, 64, 128)
output:
(None, 32, 32, 128)
block3_conv1: Conv2D
input:
(None, 32, 32, 128)
output:
(None, 32, 32, 256)
block3_conv2: Conv2D
input:
(None, 32, 32, 256)
output:
(None, 32, 32, 256)



block3_conv3: Conv2D
input:
(None, 32, 32, 256)
output:
(None, 32, 32, 256)
block3_pool: MaxPooling2D
input:
(None, 32, 32, 256)
output:
(None, 16, 16, 256)
block4_conv1: Conv2D
input:
(None, 16, 16, 256)
output:
(None, 16, 16, 512)
block4_conv2: Conv2D
input:
(None, 16, 16, 512)
output:
(None, 16, 16, 512)
block4_conv3: Conv2D
input:
(None, 16, 16, 512)
output:
(None, 16, 16, 512)
block4_pool: MaxPooling2D
input:
(None, 16, 16, 512)
output:
(None, 8, 8, 512)
block5_conv1: Conv2D
input:
(None, 8, 8, 512)
output:
(None, 8, 8, 512)
block5_conv2: Conv2D
input:
(None, 8, 8, 512)
output:
(None, 8, 8, 512)
input:
(None, 8, 8, 512)



Model-2

```
1. Use VGG-16 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 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**
4. 4.Print model.summary() and plot the architecture of the model.
Reference for plotting model
5. Train only last 2 Conv layers identical to FC layers, 1 output layer. Don't train the VGG-16 network.

In [12]:
```
tf.keras.backend.clear_session()
```

In [13]:
```
#https://blog.keras.io/building-powerful-image-classification-models-using-very-lit
for layer in vgg_model.layers:
    layer.trainable = False
input_from_vgg = vgg_model.output

conv_1 = Conv2D(filters = 2048, kernel_size = (4,4), padding = 'valid',
               activation = 'relu', kernel_initializer='HeUniform') (input_from_vgg


conv_2 = Conv2D(filters = 2048, kernel_size = (1,1), padding = 'valid',
               activation = 'relu', kernel_initializer='HeUniform') (conv_1)


output_two = Conv2D(filters = 16, kernel_size = (1,1), activation='softmax')(conv_2

flat_ = Flatten() (output_two)

model_two = Model(inputs = vgg_model.input, outputs = flat_)
model_two.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_1 (InputLayer)        [(None, 128, 128, 3)]     0

block1_conv1 (Conv2D)       (None, 128, 128, 64)      1792

block1_conv2 (Conv2D)       (None, 128, 128, 64)      36928

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

block2_conv1 (Conv2D)       (None, 64, 64, 128)       73856

block2_conv2 (Conv2D)       (None, 64, 64, 128)       147584

block2_pool (MaxPooling2D)  (None, 32, 32, 128)       0

block3_conv1 (Conv2D)       (None, 32, 32, 256)       295168

block3_conv2 (Conv2D)       (None, 32, 32, 256)       590080

block3_conv3 (Conv2D)       (None, 32, 32, 256)       590080

block3_pool (MaxPooling2D)  (None, 16, 16, 256)       0

block4_conv1 (Conv2D)       (None, 16, 16, 512)       1180160

block4_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

block4_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

block4_pool (MaxPooling2D)  (None, 8, 8, 512)         0

block5_conv1 (Conv2D)       (None, 8, 8, 512)         2359808

block5_conv2 (Conv2D)       (None, 8, 8, 512)         2359808

block5_conv3 (Conv2D)       (None, 8, 8, 512)         2359808

block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

conv2d (Conv2D)             (None, 1, 1, 2048)        16779264

conv2d_1 (Conv2D)           (None, 1, 1, 2048)        4196352

conv2d_2 (Conv2D)           (None, 1, 1, 16)          32784

flatten (Flatten)           (None, 16)                0

=================================================================
Total params: 35,723,088
Trainable params: 21,008,400
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [14]:
```
filepath="model_save/best_model_two.hdf5"
```

```python
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy',  verbose=1,

earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0.01, patience=2, verbo

tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir= ("logs/fits/"+dateti
                      histogram_freq=1,write_graph=True)

call_back_list = [ earlystop, checkpoint, tensorboard_callback]
```

In [15]:

```python
model_two.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accur

model_two.fit( train_generator,steps_per_epoch = train_steps_per_epoch,epochs=3,
                        validation_data = validation_generator, validation_steps =
                        callbacks=[call_back_list])
```

```
Epoch 1/3
563/563 [==============================] - 262s 447ms/step - loss: 1.6402 - accura
cy: 0.5293 - val_loss: 1.2483 - val_accuracy: 0.6107

Epoch 00001: val_accuracy improved from -inf to 0.61067, saving model to model_sav
e\best_model_two.hdf5
Epoch 2/3
563/563 [==============================] - 162s 286ms/step - loss: 1.1337 - accura
cy: 0.6476 - val_loss: 1.1635 - val_accuracy: 0.6364

Epoch 00002: val_accuracy improved from 0.61067 to 0.63642, saving model to model_
save\best_model_two.hdf5
Epoch 3/3
563/563 [==============================] - 160s 284ms/step - loss: 1.0044 - accura
cy: 0.6868 - val_loss: 1.1059 - val_accuracy: 0.6607

Epoch 00003: val_accuracy improved from 0.63642 to 0.66075, saving model to model_
save\best_model_two.hdf5
```

Out[15]:

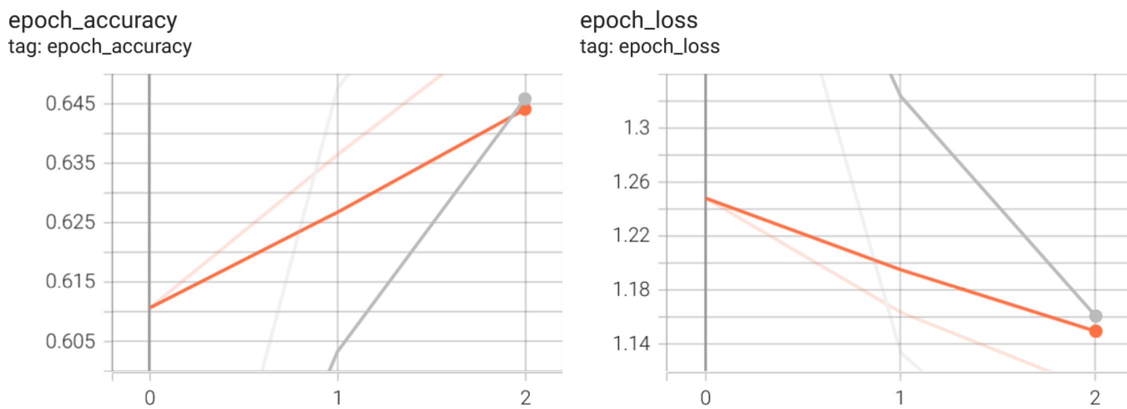
```
<keras.callbacks.History at 0x21f003adfa0>
```

In [15]:

```python
%tensorboard --logdir logs/fits
```

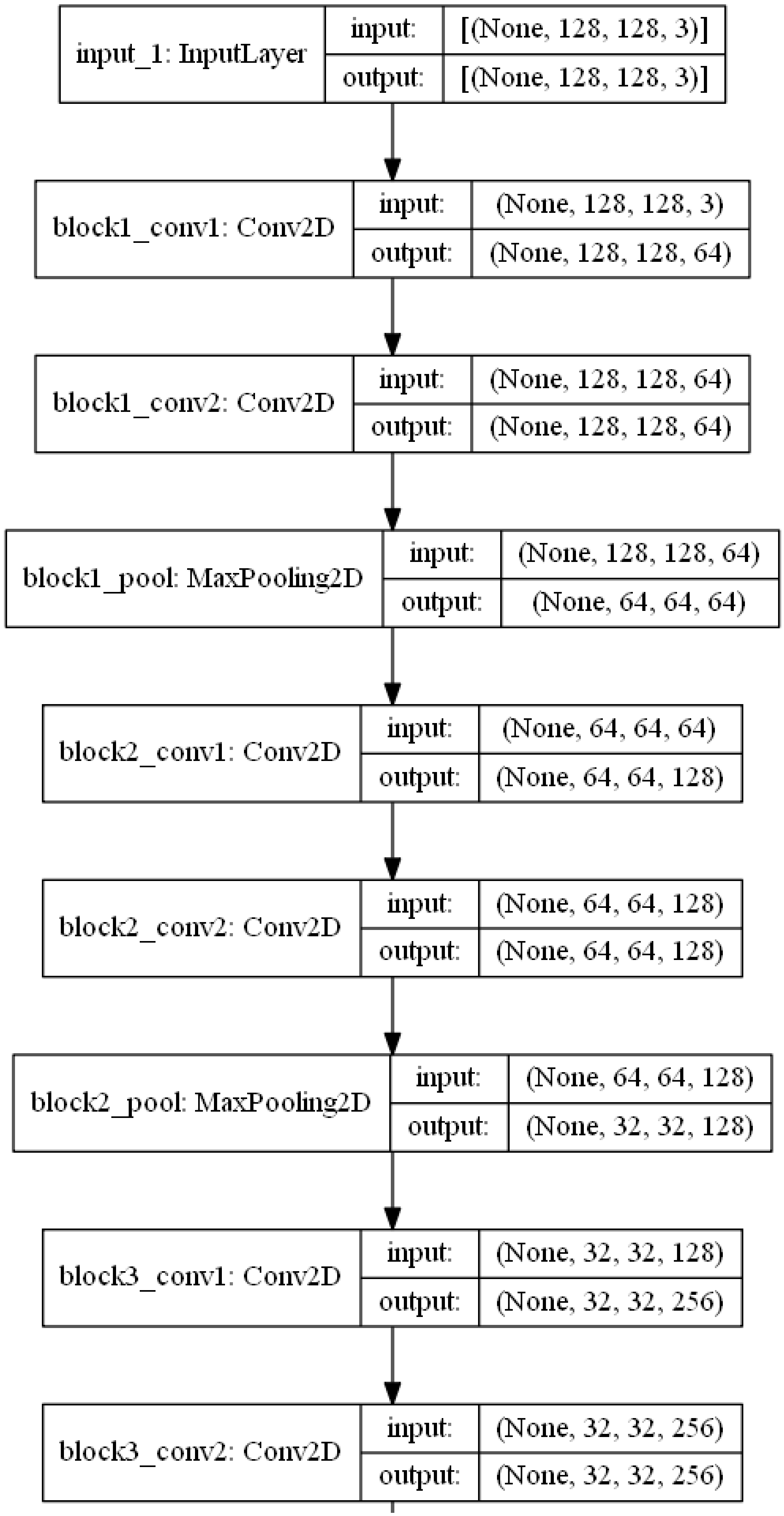
```
Reusing TensorBoard on port 6006 (pid 11692), started 3:18:56 ago. (Use '!kill 116
92' to kill it.)
```

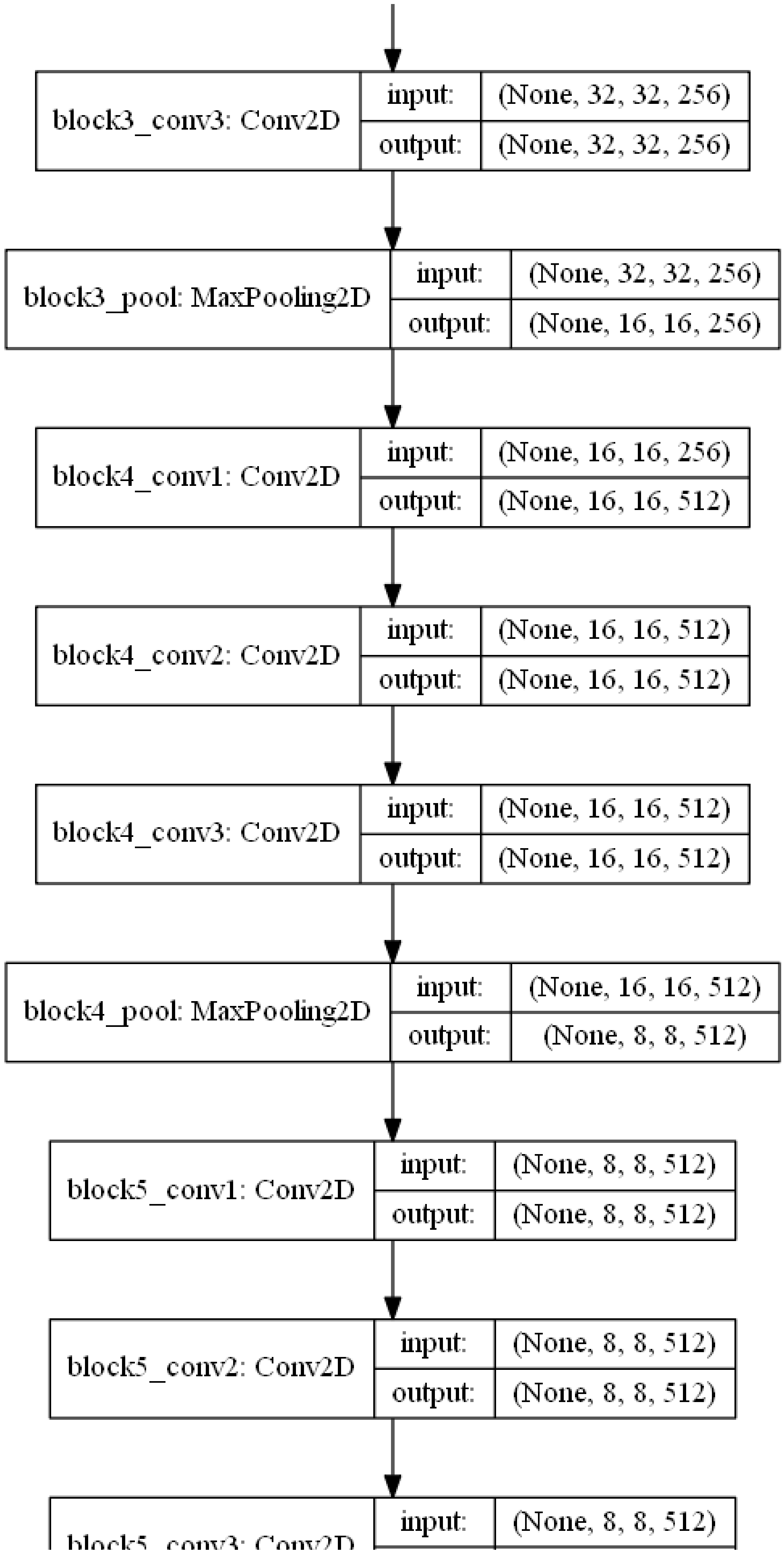
model_two Tensorboard accuracy and loss

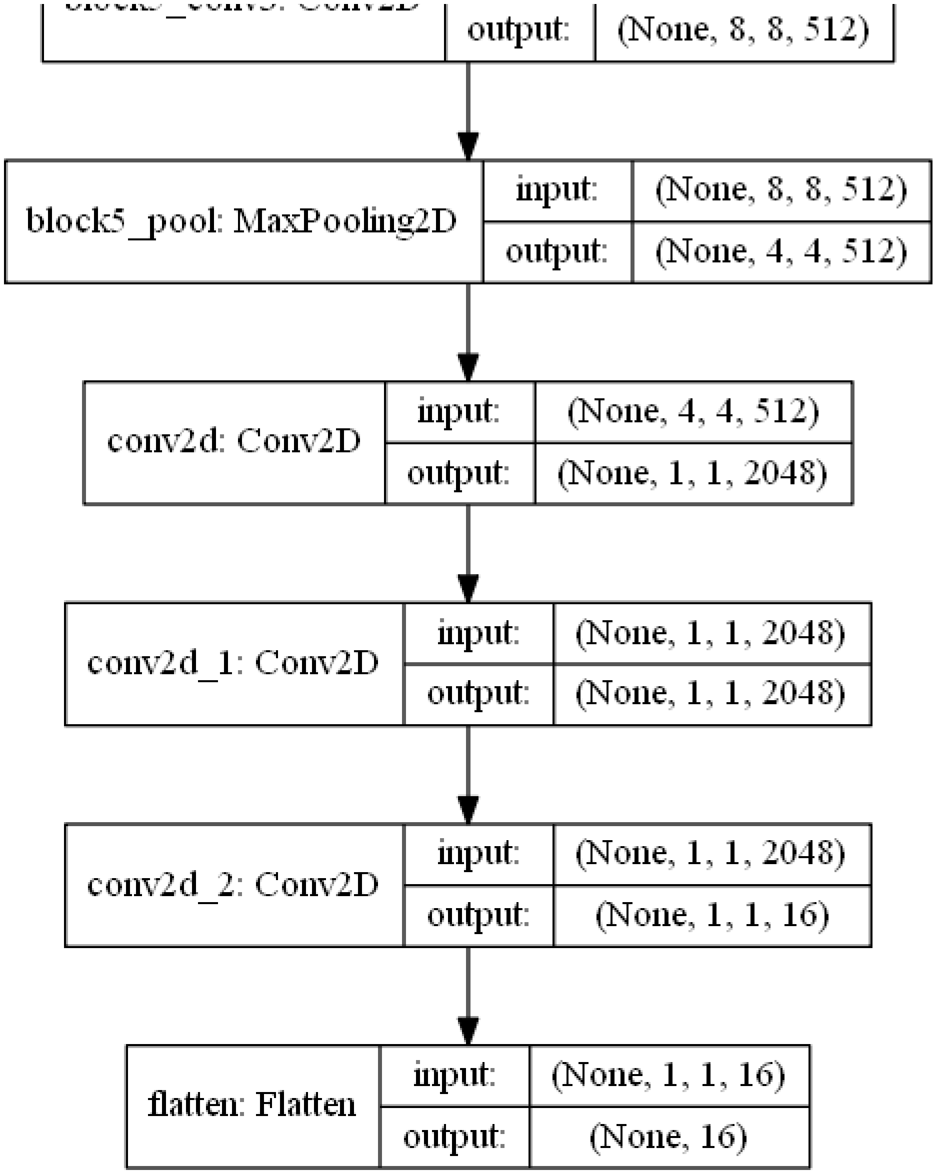


In [36]:
```
plot_model(model_two, to_file='model_two.png', show_shapes=True)
```

Out[36]:


input_1: InputLayer
input:
[(None, 128, 128, 3)]
output:
[(None, 128, 128, 3)]
block1_conv1: Conv2D
input:
(None, 128, 128, 3)
output:
(None, 128, 128, 64)
block1_conv2: Conv2D
input:
(None, 128, 128, 64)
output:
(None, 128, 128, 64)
block1_pool: MaxPooling2D
input:
(None, 128, 128, 64)
output:
(None, 64, 64, 64)
block2_conv1: Conv2D
input:
(None, 64, 64, 64)
output:
(None, 64, 64, 128)
block2_conv2: Conv2D
input:
(None, 64, 64, 128)
output:
(None, 64, 64, 128)
block2_pool: MaxPooling2D
input:
(None, 64, 64, 128)
output:
(None, 32, 32, 128)
block3_conv1: Conv2D
input:
(None, 32, 32, 128)
output:
(None, 32, 32, 256)
block3_conv2: Conv2D
input:
(None, 32, 32, 256)
output:
(None, 32, 32, 256)


block3_conv3: Conv2D
input: (None, 32, 32, 256)
output: (None, 32, 32, 256)
block3_pool: MaxPooling2D
input: (None, 32, 32, 256)
output: (None, 16, 16, 256)
block4_conv1: Conv2D
input: (None, 16, 16, 256)
output: (None, 16, 16, 512)
block4_conv2: Conv2D
input: (None, 16, 16, 512)
output: (None, 16, 16, 512)
block4_conv3: Conv2D
input: (None, 16, 16, 512)
output: (None, 16, 16, 512)
block4_pool: MaxPooling2D
input: (None, 16, 16, 512)
output: (None, 8, 8, 512)
block5_conv1: Conv2D
input: (None, 8, 8, 512)
output: (None, 8, 8, 512)
block5_conv2: Conv2D
input: (None, 8, 8, 512)
output: (None, 8, 8, 512)
input: (None, 8, 8, 512)



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]:
```
tf.keras.backend.clear_session()
```

In [31]:
```
#https://blog.keras.io/building-powerful-image-classification-models-using-very-lit
for layer in vgg_model.layers[-6:]:
    layer.trainable = True

for layer in vgg_model.layers[:-6]:
    layer.trainable = False

input_from_vgg = vgg_model.output
```

```
conv_1_ = Conv2D(filters = 2048, kernel_size = (4,4), padding = 'valid',
               activation = 'relu', kernel_initializer='HeUniform') (input_from_vgg


conv_2_ = Conv2D(filters = 2048, kernel_size = (1,1), padding = 'valid',
               activation = 'relu', kernel_initializer='HeUniform') (conv_1_)


output_three = Conv2D(16, kernel_size = (1,1), activation='softmax')(conv_2_)

flat_3 = Flatten() (output_three)

model_three = Model(inputs = vgg_model.input, outputs = flat_3)
model_three.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_1 (InputLayer)        [(None, 128, 128, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)       (None, 128, 128, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)       (None, 128, 128, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)  (None, 64, 64, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)       (None, 64, 64, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)       (None, 64, 64, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)  (None, 32, 32, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)       (None, 32, 32, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)       (None, 32, 32, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)       (None, 32, 32, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)  (None, 16, 16, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)       (None, 16, 16, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)       (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)       (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)  (None, 8, 8, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)       (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)       (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)       (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0
_________________________________________________________________
conv2d (Conv2D)             (None, 1, 1, 2048)        16779264
_________________________________________________________________
conv2d_1 (Conv2D)           (None, 1, 1, 2048)        4196352
_________________________________________________________________
conv2d_2 (Conv2D)           (None, 1, 1, 16)          32784
_________________________________________________________________
flatten (Flatten)           (None, 16)                0
=================================================================
Total params: 35,723,088
Trainable params: 30,447,632
Non-trainable params: 5,275,456
_________________________________________________________________
```

In [32]:

```
filepath="model_save/best_model_three.hdf5"
```

```
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy',  verbose=1,

earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0.001, patience=2, verb

tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir= ("logs/fits/"+dateti
                       histogram_freq=1,write_graph=True)

call_back_list = [ earlystop, checkpoint, tensorboard_callback]
```

In [33]:
```
model_three.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['acc

model_three.fit( train_generator,steps_per_epoch = train_steps_per_epoch,epochs=5,
                        validation_data = validation_generator, validation_steps =
                        callbacks=[call_back_list])
```

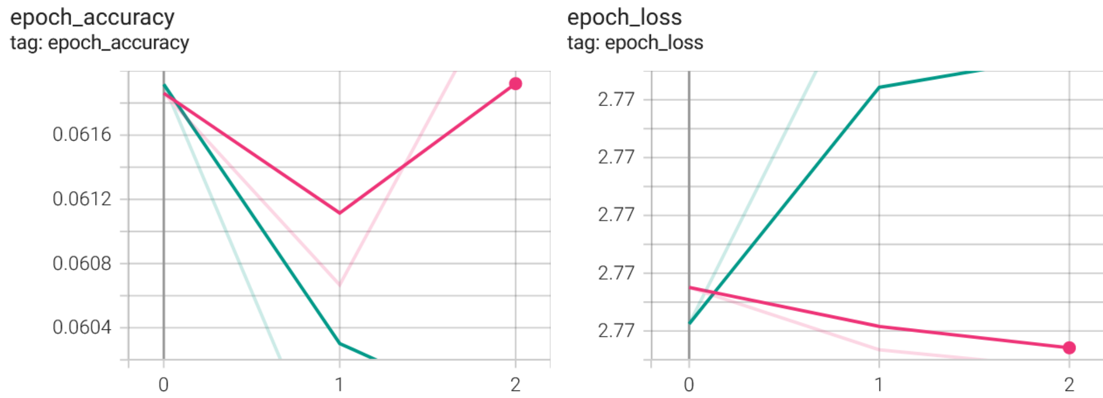
```
Epoch 1/5
563/563 [==============================] - 273s 483ms/step - loss: 2.7728 - accura
cy: 0.0619 - val_loss: 2.7727 - val_accuracy: 0.0619

Epoch 00001: val_accuracy improved from -inf to 0.06192, saving model to model_sav
e\best_model_three.hdf5
Epoch 2/5
563/563 [==============================] - 279s 496ms/step - loss: 2.7727 - accura
cy: 0.0607 - val_loss: 2.7730 - val_accuracy: 0.0593

Epoch 00002: val_accuracy did not improve from 0.06192
Epoch 3/5
563/563 [==============================] - 278s 493ms/step - loss: 2.7727 - accura
cy: 0.0627 - val_loss: 2.7730 - val_accuracy: 0.0593

Epoch 00003: val_accuracy did not improve from 0.06192
Epoch 00003: early stopping
```

Out[33]:
```
<keras.callbacks.History at 0x2204d567eb0>
```

In [12]:
```
%tensorboard --logdir logs/fits
```

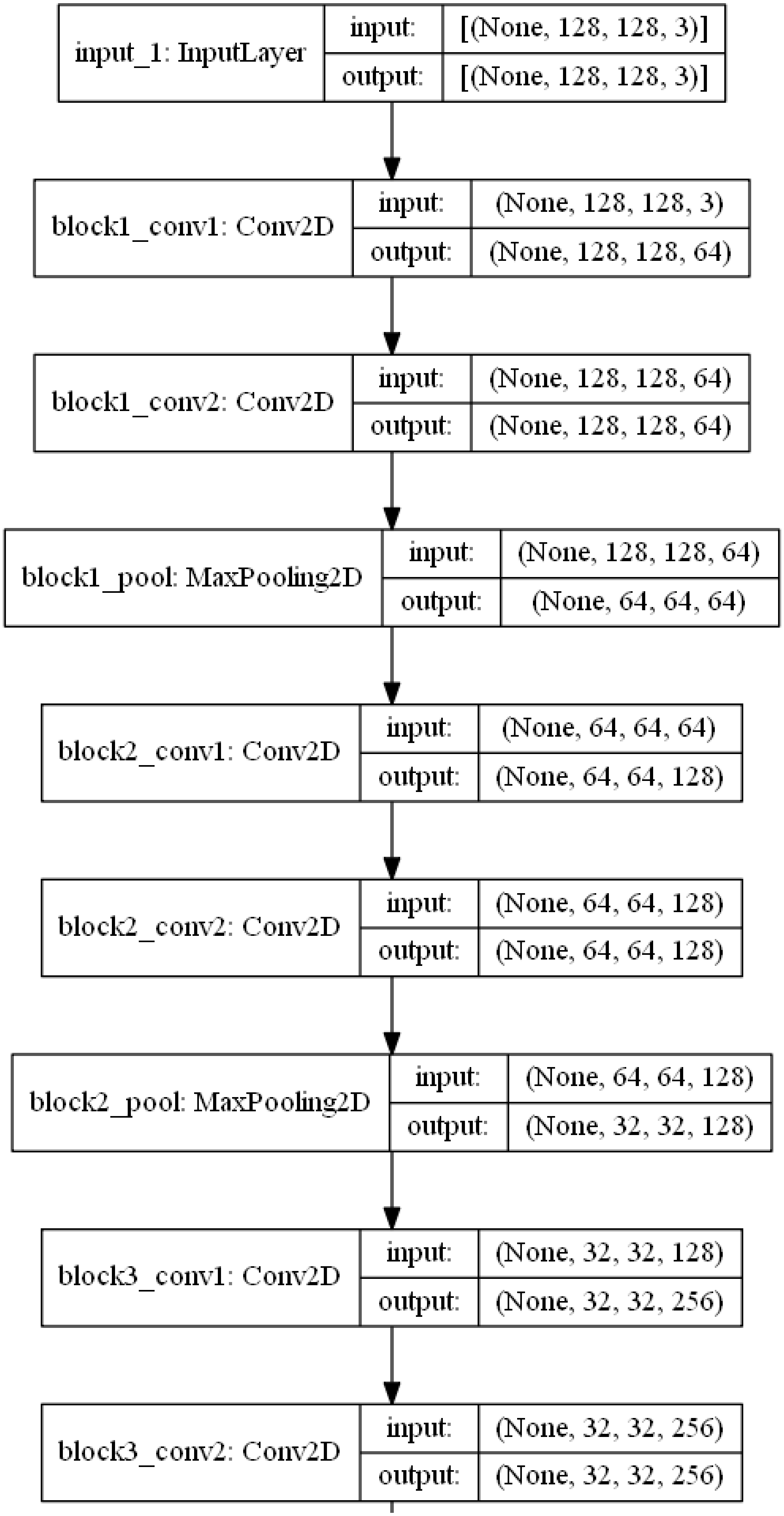
```
Reusing TensorBoard on port 6006 (pid 11692), started 15:28:36 ago. (Use '!kill 11
692' to kill it.)
```

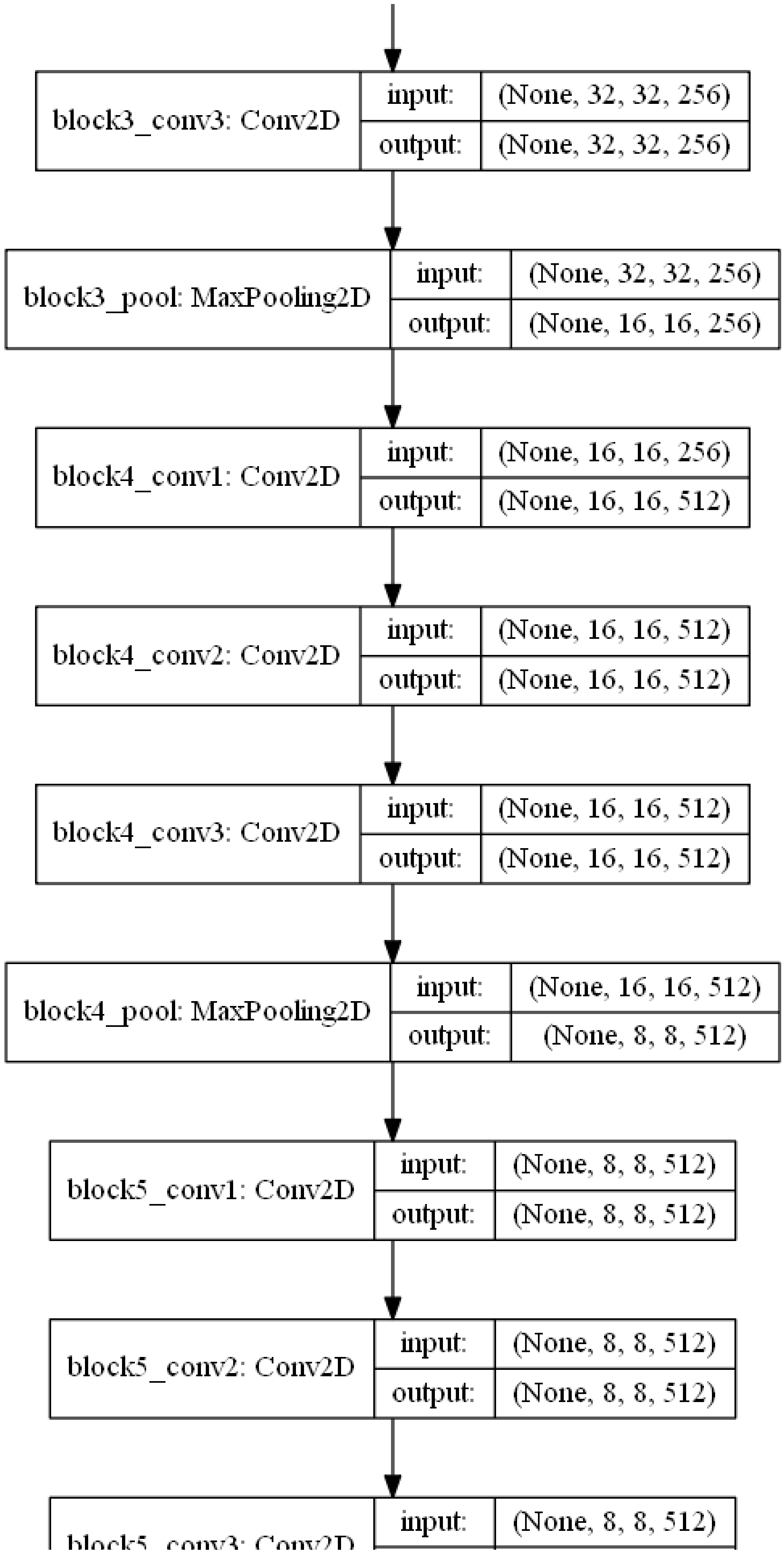
model_three Tensorboard accuracy and loss

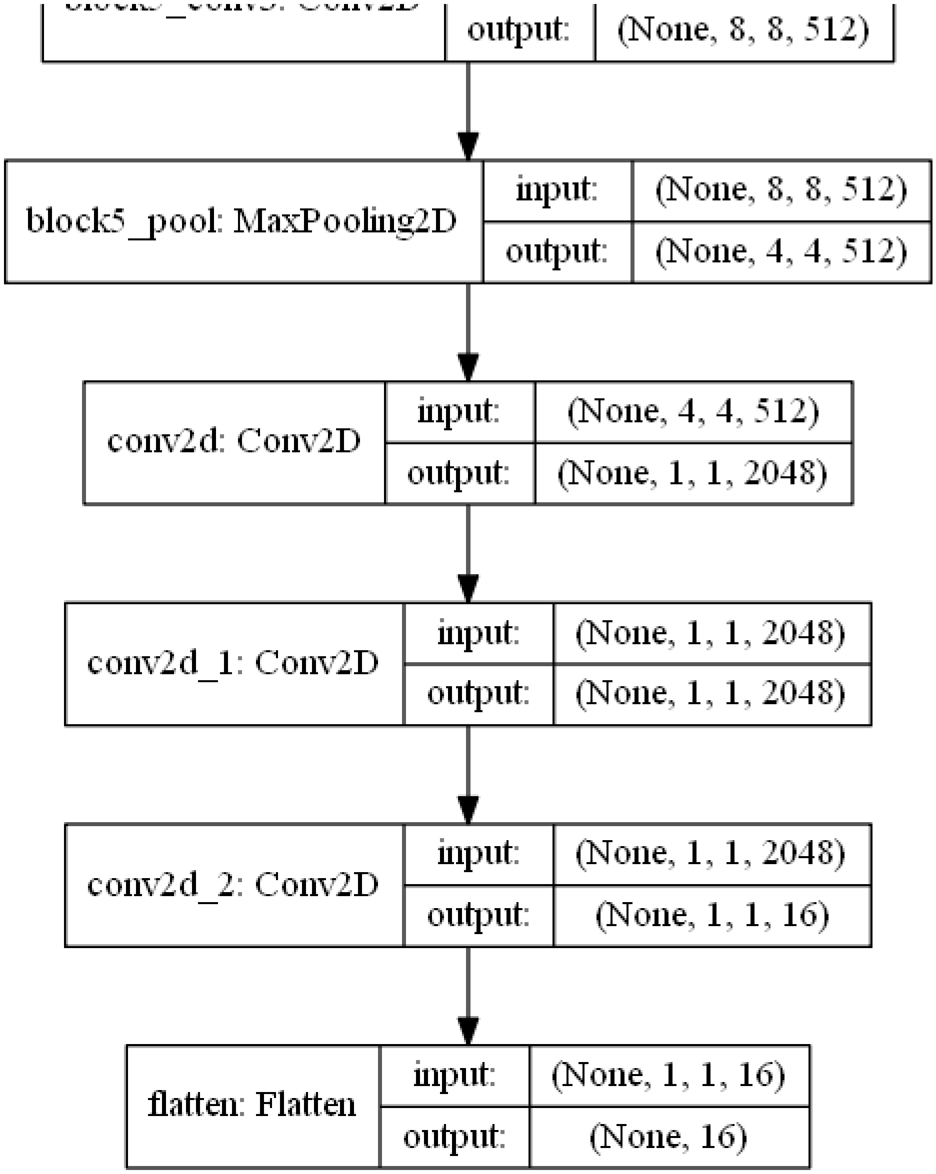



```
In [35]: plot_model(model_three, to_file='model_three.png', show_shapes=True)
```

Out[35]:


input_1: InputLayer
input:
[(None, 128, 128, 3)]
output:
[(None, 128, 128, 3)]
block1_conv1: Conv2D
input:
(None, 128, 128, 3)
output:
(None, 128, 128, 64)
block1_conv2: Conv2D
input:
(None, 128, 128, 64)
output:
(None, 128, 128, 64)
block1_pool: MaxPooling2D
input:
(None, 128, 128, 64)
output:
(None, 64, 64, 64)
block2_conv1: Conv2D
input:
(None, 64, 64, 64)
output:
(None, 64, 64, 128)
block2_conv2: Conv2D
input:
(None, 64, 64, 128)
output:
(None, 64, 64, 128)
block2_pool: MaxPooling2D
input:
(None, 64, 64, 128)
output:
(None, 32, 32, 128)
block3_conv1: Conv2D
input:
(None, 32, 32, 128)
output:
(None, 32, 32, 256)
block3_conv2: Conv2D
input:
(None, 32, 32, 256)
output:
(None, 32, 32, 256)


block3_conv3: Conv2D
input:
(None, 32, 32, 256)
output:
(None, 32, 32, 256)
block3_pool: MaxPooling2D
input:
(None, 32, 32, 256)
output:
(None, 16, 16, 256)
block4_conv1: Conv2D
input:
(None, 16, 16, 256)
output:
(None, 16, 16, 512)
block4_conv2: Conv2D
input:
(None, 16, 16, 512)
output:
(None, 16, 16, 512)
block4_conv3: Conv2D
input:
(None, 16, 16, 512)
output:
(None, 16, 16, 512)
block4_pool: MaxPooling2D
input:
(None, 16, 16, 512)
output:
(None, 8, 8, 512)
block5_conv1: Conv2D
input:
(None, 8, 8, 512)
output:
(None, 8, 8, 512)
block5_conv2: Conv2D
input:
(None, 8, 8, 512)
output:
(None, 8, 8, 512)
input:
(None, 8, 8, 512)



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

In [34]:

```
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Model", "Epochs", "val_accuracy"]

x.add_row(["model_one", "3","0.6526"])
x.add_row(["model_two", "3","0.6607"])
x.add_row(["model_three", "3","0.0619"])

print(x)
```

```
+-------------+--------+--------------+
|    Model    | Epochs | val_accuracy |
+-------------+--------+--------------+
|  model_one  |   3    |    0.6526    |
|  model_two  |   3    |    0.6607    |
| model_three |   3    |    0.0619    |
+-------------+--------+--------------+
```

Observations:

1. model_one only had fully connected layers after vgg-16 model as trainable layers and also had the least trainable parameters among all three models (2,919,312).
2. model_two used convolution layers instead of fully connected layers after vgg-16 model as trainable layers and had the second most trainable parameters among all three models (21,008,400).
3. model_three used the same architecture of model_two, but also had the last 6 layers in vgg-16 model to be set as trainable. Because of training these additional layers, it had the highest number of traininable parameters among all three models (30,447,632).
4. model_one took the least time to train because it had the least number of trainable parameters while model_two took longer and model_three took the longest time to train as they had the higher number of trainable parameters.
5. Since model_three has 9 layers and high number of parameters to train, it needs significant number of epochs to get higher accuracy. This is the reason why model_three only had 6% accuracy after three epochs.
6. Since model_one and mode_two only had 3 layers to be trained and used the pre-trained weights from vgg-16 model, they have more than 60% accuracy after 3 epochs