Final Project

Case Study in Finance - House Rooms Classification

Rachapudi, Vinod

CSCI E-89 Deep Learning, Fall 2023

Harvard University Extension School

Prof. Zoran B. Djordjević

Introduction

In FinTech Mortgage business investment (Mortgage Portfolio) decisions are made based on a house and its rooms.

Therefore, having the ability to classify the rooms of a house to identify the type of room such as Dining Room vs Bed Room vs et al enables these investment decisions.

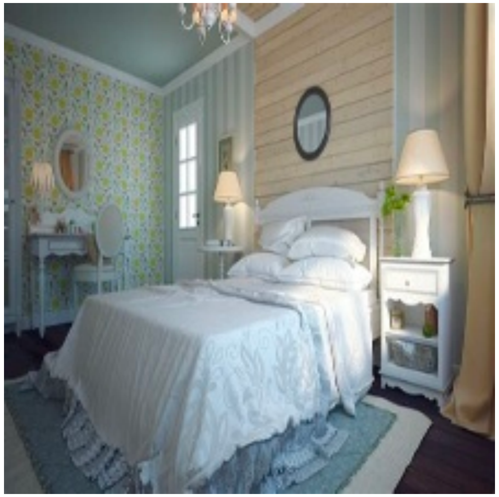
The code in this project helps with binary classification of Dining Room vs Bed Room by using

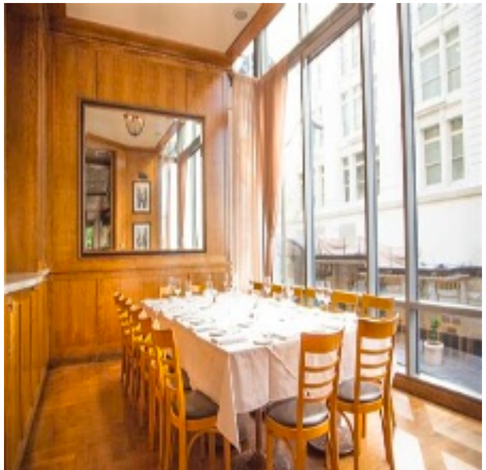
- **Convolutional Neural Networks in Part 1**
- **Pre-trained Convolutional Neural Network VGG16 in Part 2 and**
- **Pre-trained Convolutional Neural Networks - Xception in Part 3**

Downloading the Data

The Data is downloaded from Kaggle https://www.kaggle.com/datasets/robinreni/house-rooms-image-dataset/data

This Kaggle Data set has 1248 bedroom pictures and 1158 dining room pictures





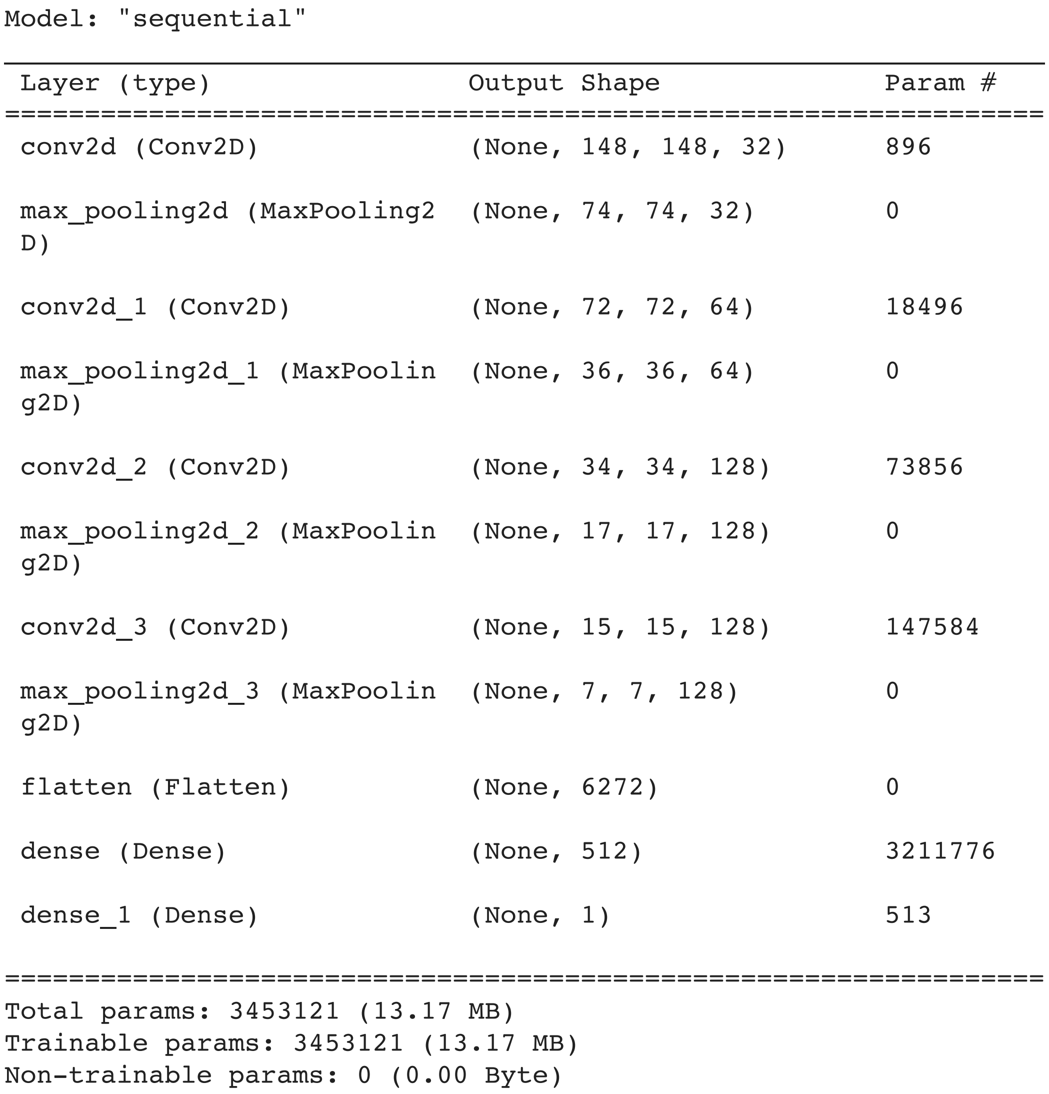
Splitting of Data into Training, Validation & Test DataSets

This Kaggle Data set has 1248 bedroom pictures and 1158 dining room pictures

```
Total training bedroom images: 500
Total training diningroom images: 500
Total validation bedroom images: 300
Total validation diningroom images: 300
Total test bedroom images: 448
Total test diningroom images: 358
```

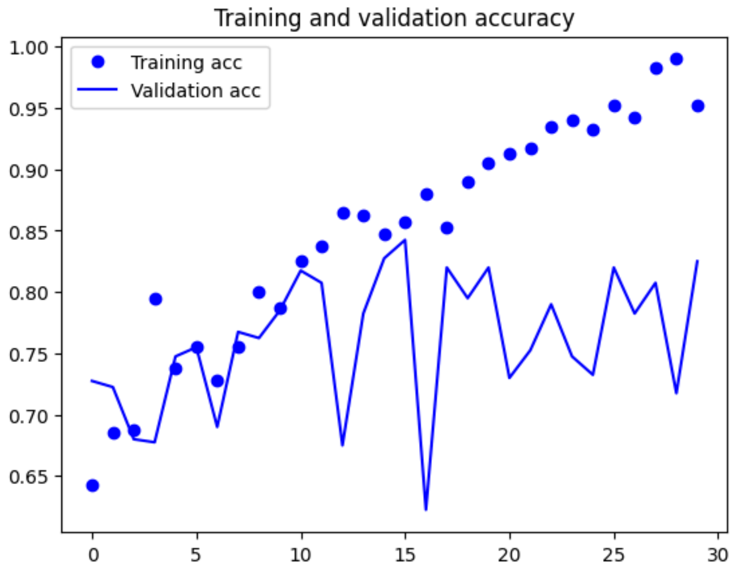
Part 1 - Convolutional Neural Networks

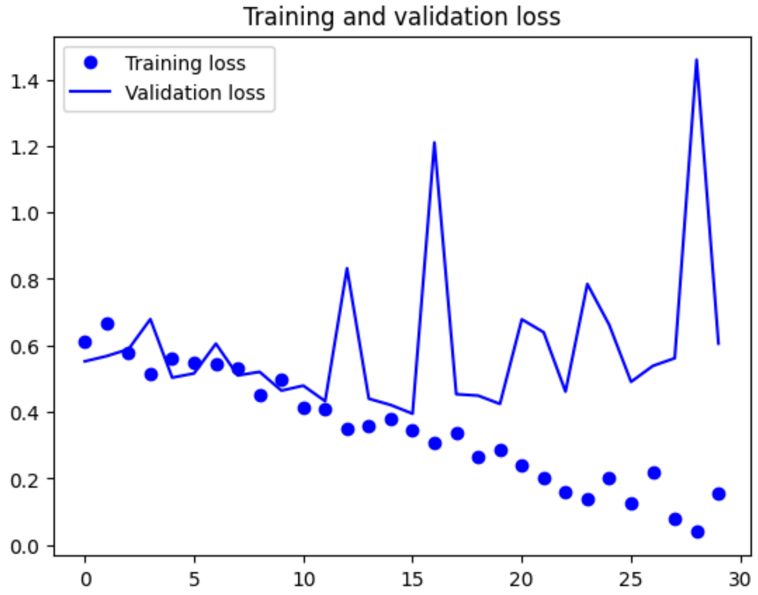
Conv Neural Network



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 max_pooling2d (MaxPooling2  (None, 74, 74, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 36, 36, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 17, 17, 128)       0
 g2D)

 conv2d_3 (Conv2D)           (None, 15, 15, 128)       147584

 max_pooling2d_3 (MaxPoolin  (None, 7, 7, 128)         0
 g2D)

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 512)               3211776

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 3453121 (13.17 MB)
Trainable params: 3453121 (13.17 MB)
Non-trainable params: 0 (0.00 Byte)
```

Validation Accuracy is at 82.5%



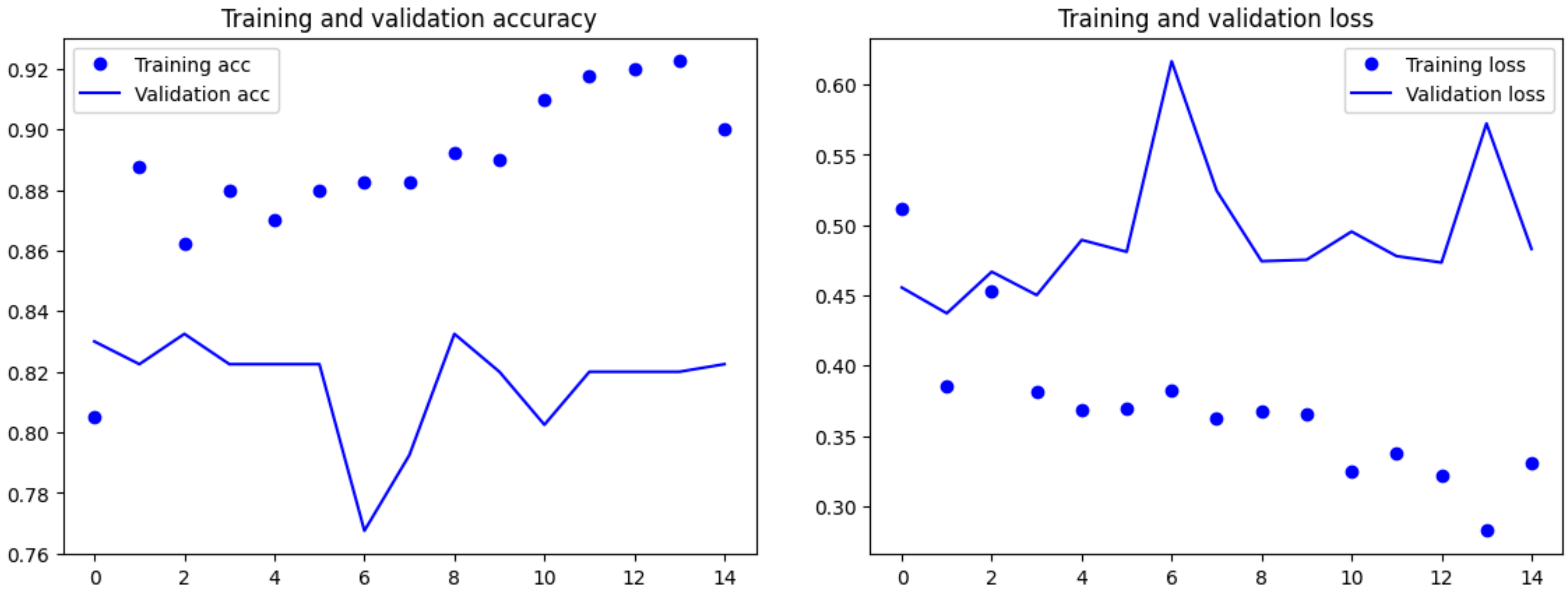


```
Epoch 30/30
20/20 [==============================] - 3s 136ms/step - loss: 0.1534 - acc: 0.9525
- val_loss: 0.6058 - val_acc: 0.8250
```

Added the L1 Regularizer (0.0001) on the Dense Layer

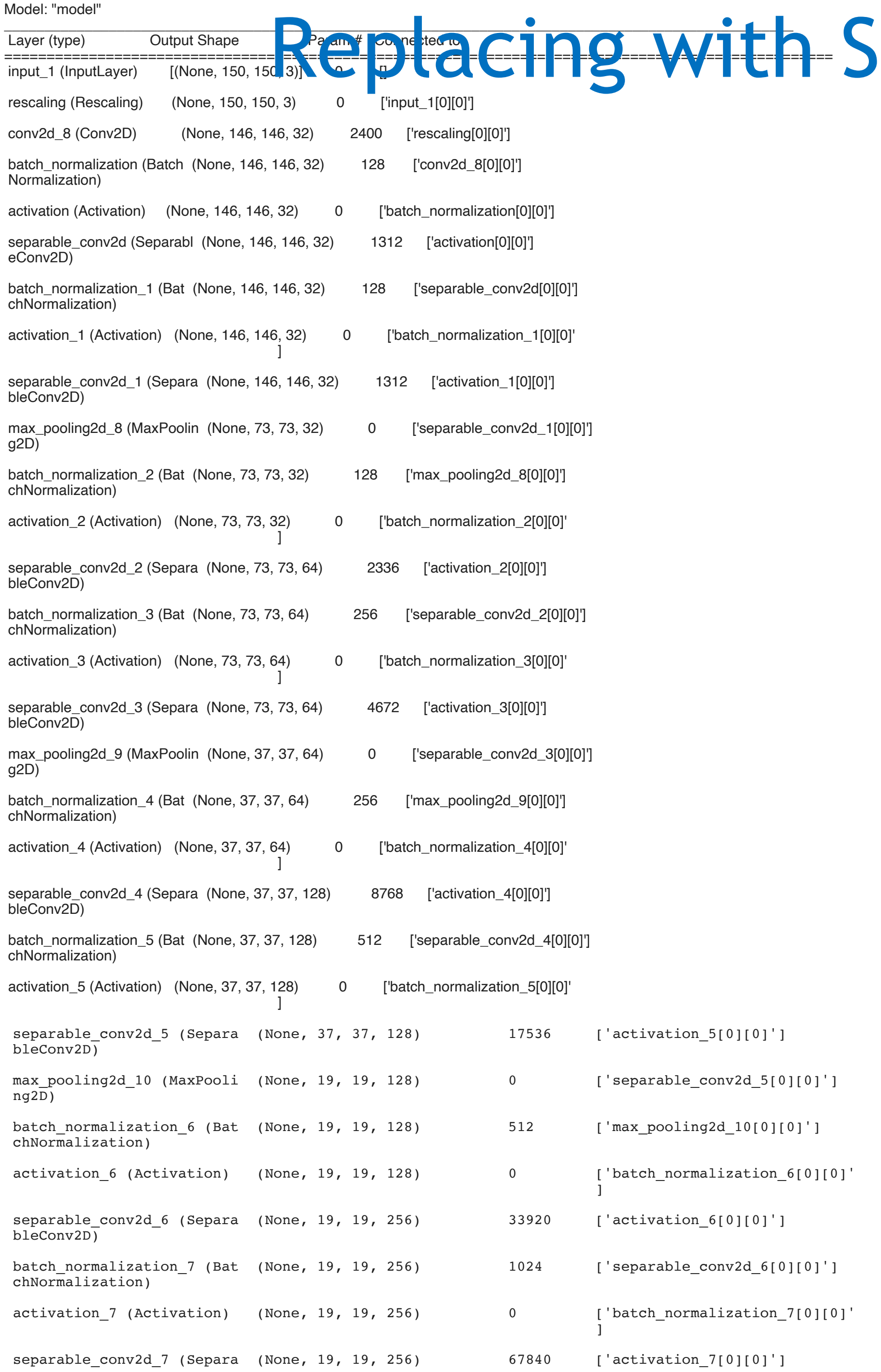
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_4 (Conv2D)           (None, 148, 148, 32)      896       
                                                                 
 max_pooling2d_4 (MaxPoolin  (None, 74, 74, 32)        0         
 g2D)                                                            
                                                                 
 conv2d_5 (Conv2D)           (None, 72, 72, 64)        18496     
                                                                 
 max_pooling2d_5 (MaxPoolin  (None, 36, 36, 64)        0         
 g2D)                                                            
                                                                 
 conv2d_6 (Conv2D)           (None, 34, 34, 128)       73856     
                                                                 
 max_pooling2d_6 (MaxPoolin  (None, 17, 17, 128)       0         
 g2D)                                                            
                                                                 
 conv2d_7 (Conv2D)           (None, 15, 15, 128)       147584    
                                                                 
 max_pooling2d_7 (MaxPoolin  (None, 7, 7, 128)         0         
 g2D)                                                            
                                                                 
 flatten_1 (Flatten)         (None, 6272)              0         
                                                                 
 dense_2 (Dense)             (None, 512)               3211776   
                                                                 
 dense_3 (Dense)             (None, 1)                 513       
                                                                 
=================================================================
Total params: 3453121 (13.17 MB)
Trainable params: 3453121 (13.17 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Regularization does not help 82.5% vs 82.2%



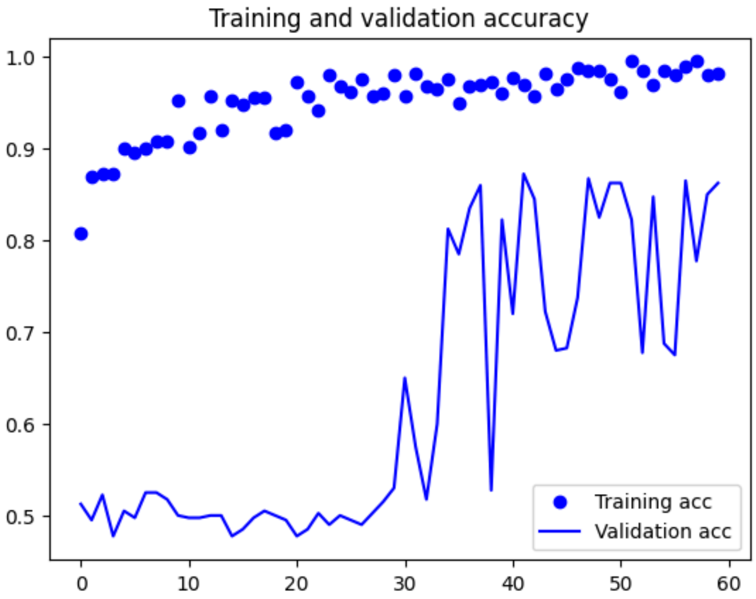
```
Epoch 15/15
20/20 [==============================] - 3s 132ms/step - loss: 0.3307 - acc: 0.90
- val_loss: 0.4830 - val_acc: 0.8225
```

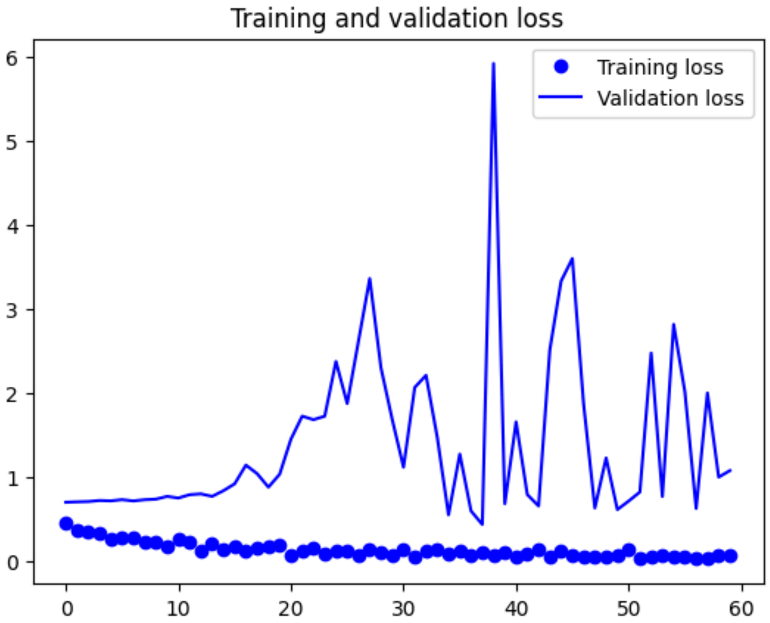
Replacing with Separable Conv2D



```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 150, 150, 3)]  0         []

 rescaling (Rescaling)       (None, 150, 150, 3)  0           ['input_1[0][0]']

 conv2d_8 (Conv2D)           (None, 146, 146, 32)  2400       ['rescaling[0][0]']

 batch_normalization (Batch  (None, 146, 146, 32)  128        ['conv2d_8[0][0]']
 Normalization)

 activation (Activation)     (None, 146, 146, 32)  0          ['batch_normalization[0][0]']

 separable_conv2d (Separabl  (None, 146, 146, 32)  1312       ['activation[0][0]']
 eConv2D)

 batch_normalization_1 (Bat  (None, 146, 146, 32)  128        ['separable_conv2d[0][0]']
 chNormalization)

 activation_1 (Activation)   (None, 146, 146, 32)  0          ['batch_normalization_1[0][0]'
                                                              ]

 separable_conv2d_1 (Separa  (None, 146, 146, 32)  1312       ['activation_1[0][0]']
 bleConv2D)

 max_pooling2d_8 (MaxPoolin  (None, 73, 73, 32)   0           ['separable_conv2d_1[0][0]']
 g2D)

 batch_normalization_2 (Bat  (None, 73, 73, 32)   128         ['max_pooling2d_8[0][0]']
 chNormalization)

 activation_2 (Activation)   (None, 73, 73, 32)   0           ['batch_normalization_2[0][0]'
                                                              ]

 separable_conv2d_2 (Separa  (None, 73, 73, 64)   2336        ['activation_2[0][0]']
 bleConv2D)

 batch_normalization_3 (Bat  (None, 73, 73, 64)   256         ['separable_conv2d_2[0][0]']
 chNormalization)

 activation_3 (Activation)   (None, 73, 73, 64)   0           ['batch_normalization_3[0][0]'
                                                              ]

 separable_conv2d_3 (Separa  (None, 73, 73, 64)   4672        ['activation_3[0][0]']
 bleConv2D)

 max_pooling2d_9 (MaxPoolin  (None, 37, 37, 64)   0           ['separable_conv2d_3[0][0]']
 g2D)

 batch_normalization_4 (Bat  (None, 37, 37, 64)   256         ['max_pooling2d_9[0][0]']
 chNormalization)

 activation_4 (Activation)   (None, 37, 37, 64)   0           ['batch_normalization_4[0][0]'
                                                              ]

 separable_conv2d_4 (Separa  (None, 37, 37, 128)  8768        ['activation_4[0][0]']
 bleConv2D)

 batch_normalization_5 (Bat  (None, 37, 37, 128)  512         ['separable_conv2d_4[0][0]']
 chNormalization)

 activation_5 (Activation)   (None, 37, 37, 128)  0           ['batch_normalization_5[0][0]'
                                                              ]

 separable_conv2d_5 (Separa  (None, 37, 37, 128)  17536       ['activation_5[0][0]']
 bleConv2D)

 max_pooling2d_10 (MaxPooli  (None, 19, 19, 128)  0           ['separable_conv2d_5[0][0]']
 ng2D)

 batch_normalization_6 (Bat  (None, 19, 19, 128)  512         ['max_pooling2d_10[0][0]']
 chNormalization)

 activation_6 (Activation)   (None, 19, 19, 128)  0           ['batch_normalization_6[0][0]'
                                                              ]

 separable_conv2d_6 (Separa  (None, 19, 19, 256)  33920       ['activation_6[0][0]']
 bleConv2D)

 batch_normalization_7 (Bat  (None, 19, 19, 256)  1024        ['separable_conv2d_6[0][0]']
 chNormalization)

 activation_7 (Activation)   (None, 19, 19, 256)  0           ['batch_normalization_7[0][0]'
                                                              ]

 separable_conv2d_7 (Separa  (None, 19, 19, 256)  67840       ['activation_7[0][0]']
```

The validation accuracy went up to 86.25% with the SeparableConv2D replacement from the original (only Conv2D withOUT regularization) 82.5%


Training and validation accuracy


Training and validation loss

```
Epoch 60/60
20/20 [==============================] - 3s 131ms/step - loss: 0.0567 - acc: 0.9825
- val_loss: 1.0717 - val_acc: 0.8625
```

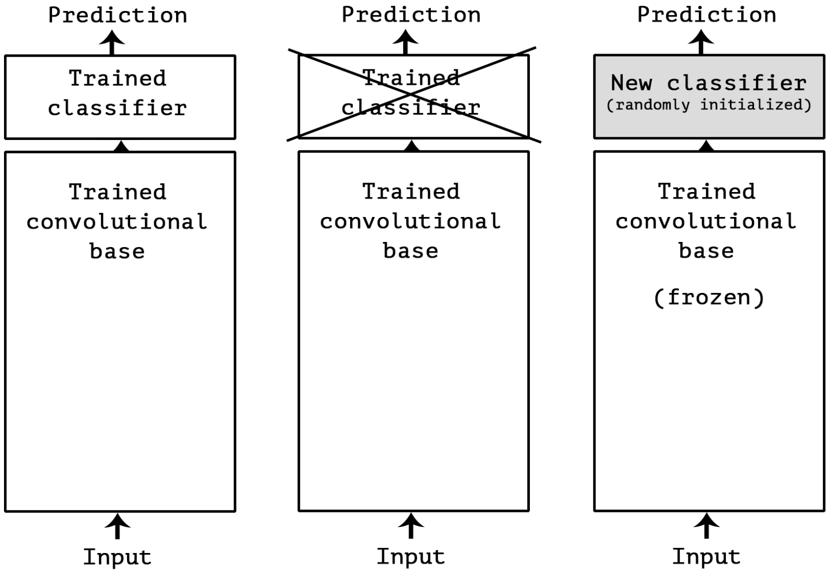
This shows that Separable CONV2D leads to ~4% increase in accuracy which is a big deal. There is still some overfitting.

The of Trainable params decreased to 674,817 from the original

Trainable params: 3,453,121 - which is good as with fewer parameters we are getting a 4% accuracy lift.

Part 2 - Pre-trained Convolutional Neural Networks - VGG16

Pre-trained Convolutional Neural Network VGG16

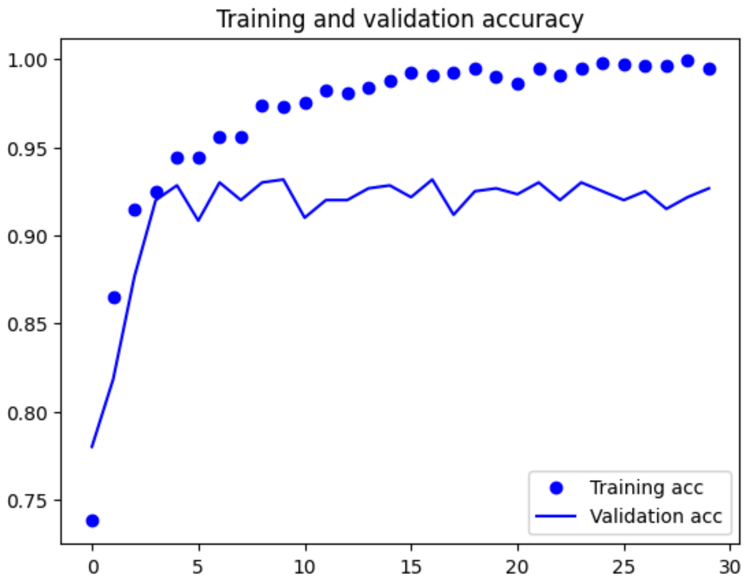


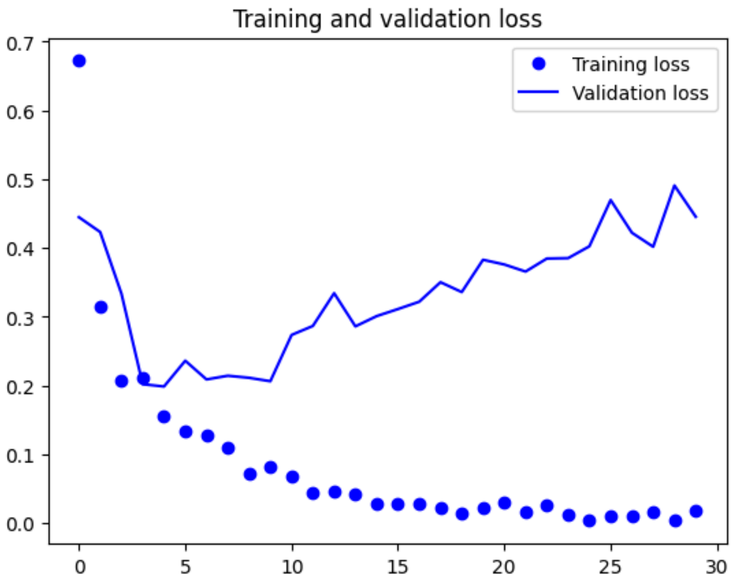
```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 150, 150, 3)]     0

 block1_conv1 (Conv2D)       (None, 150, 150, 64)      1792

 block1_conv2 (Conv2D)       (None, 150, 150, 64)      36928

 block1_pool (MaxPooling2D)  (None, 75, 75, 64)        0

 block2_conv1 (Conv2D)       (None, 75, 75, 128)       73856

 block2_conv2 (Conv2D)       (None, 75, 75, 128)       147584

 block2_pool (MaxPooling2D)  (None, 37, 37, 128)       0

 block3_conv1 (Conv2D)       (None, 37, 37, 256)       295168

 block3_conv2 (Conv2D)       (None, 37, 37, 256)       590080

 block3_conv3 (Conv2D)       (None, 37, 37, 256)       590080

 block3_pool (MaxPooling2D)  (None, 18, 18, 256)       0

 block4_conv1 (Conv2D)       (None, 18, 18, 512)       1180160

 block4_conv2 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_conv3 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 9, 9, 512)         0

 block5_conv1 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv2 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv3 (Conv2D)       (None, 9, 9, 512)         2359808

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

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 14714688 (56.13 MB)
Non-trainable params: 0 (0.00 Byte)
```

Densely-connected classifier

```
from keras import models
from keras import layers
from keras import optimizers

model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_dim=4 * 4 * 512))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizers.RMSprop(lr=2e-5),
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(train_features, train_labels,
                    epochs=30,
                    batch_size=20,
                    validation_data=(validation_features, validation_labels))
```

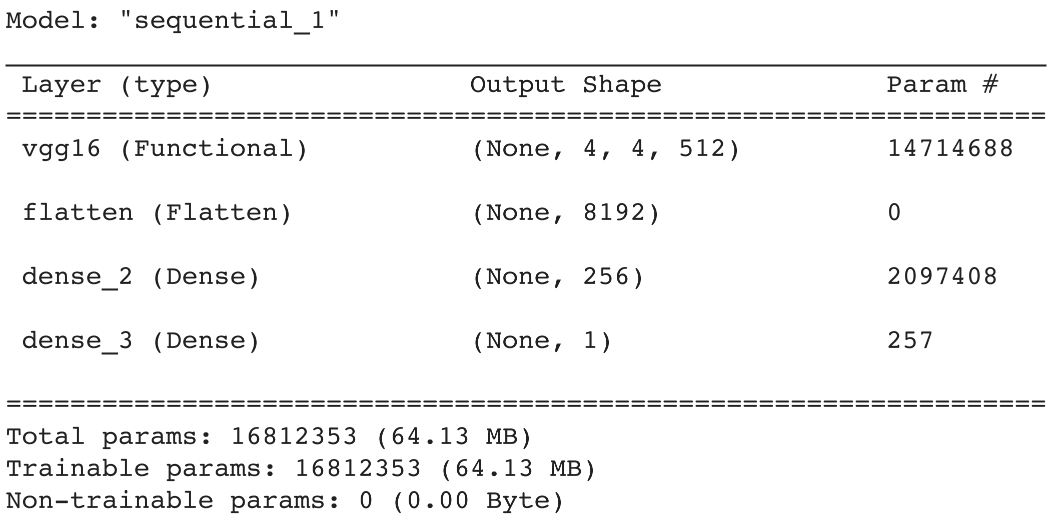
Validation Accuracy 92.67%



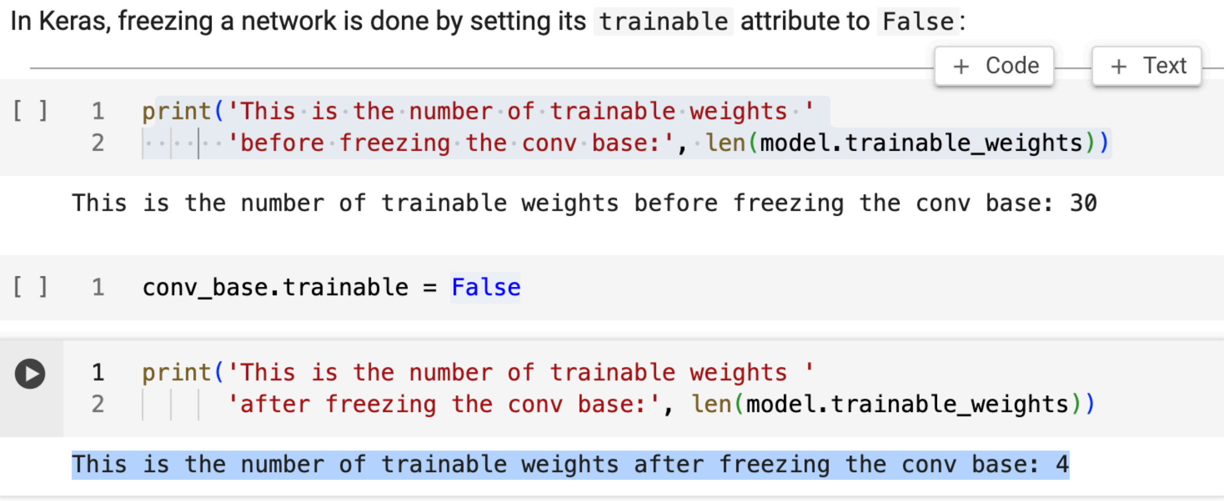


```
Epoch 30/30
50/50 [==============================] - 0s 8ms/step - loss: 0.0191 - acc: 0.9950 -
val_loss: 0.4455 - val_acc: 0.9267
```

Building the VGG16 and layering the dense layers and then freezing them



```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 4, 4, 512)         14714688

 flatten (Flatten)           (None, 8192)              0

 dense_2 (Dense)             (None, 256)               2097408

 dense_3 (Dense)             (None, 1)                 257

=================================================================
Total params: 16812353 (64.13 MB)
Trainable params: 16812353 (64.13 MB)
Non-trainable params: 0 (0.00 Byte)
```



In Keras, freezing a network is done by setting its `trainable` attribute to `False`:

```
print('This is the number of trainable weights '
      'before freezing the conv base:', len(model.trainable_weights))
```

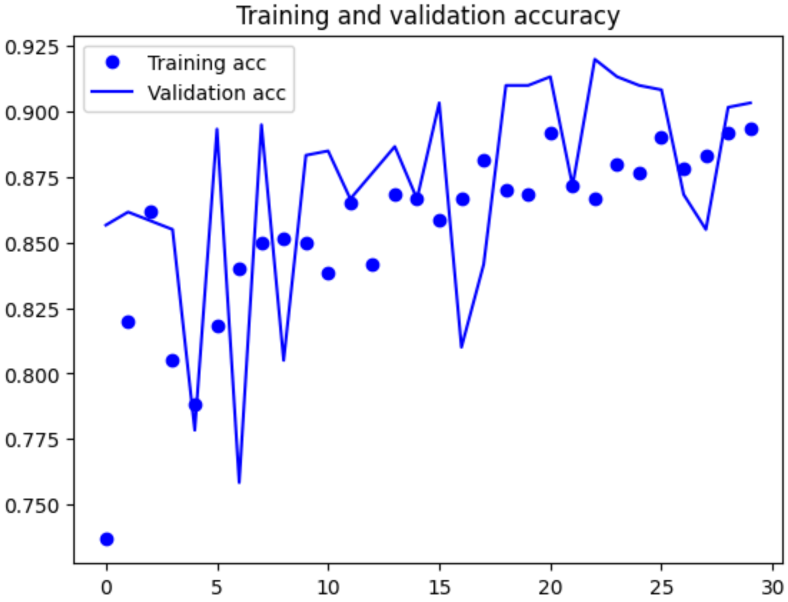
```
This is the number of trainable weights before freezing the conv base: 30
```

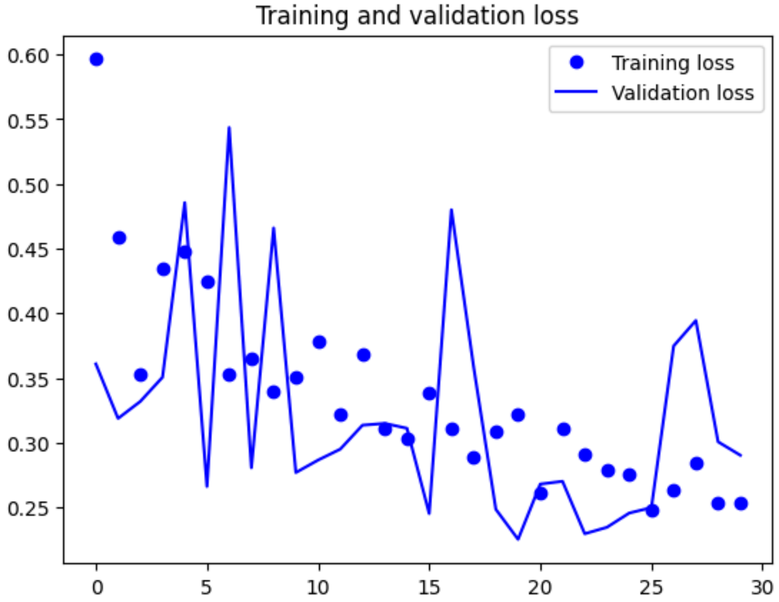
```
conv_base.trainable = False
```

```
print('This is the number of trainable weights '
      'after freezing the conv base:', len(model.trainable_weights))
```

```
This is the number of trainable weights after freezing the conv base: 4
```

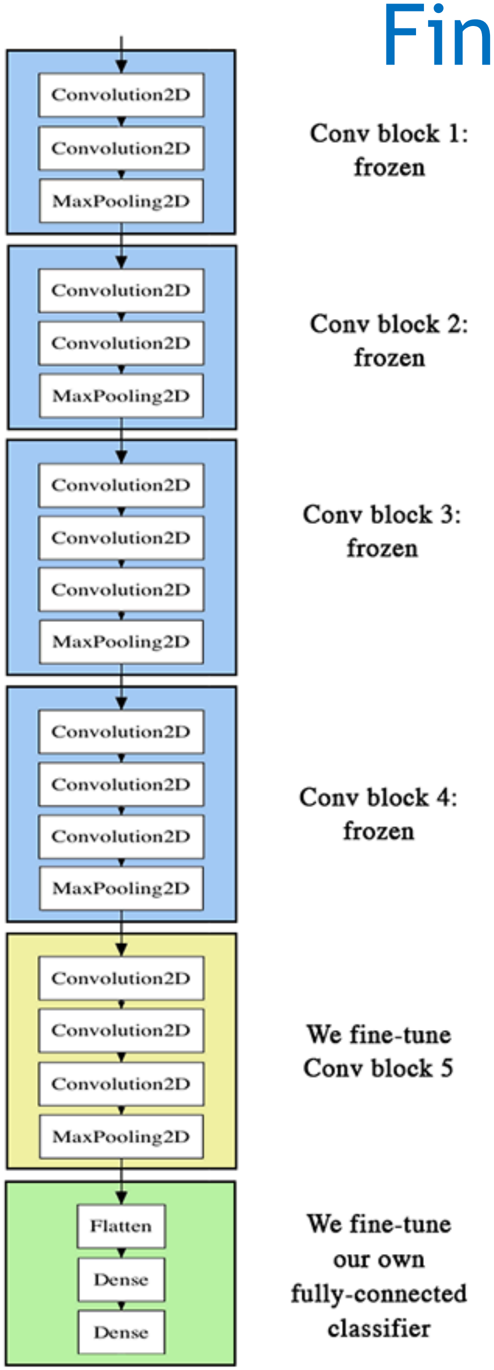
Validation Accuracy is at 90.33%





```
Epoch 30/30
30/30 - 12s - loss: 0.2539 - acc: 0.8933 - val_loss: 0.2903 - val_acc: 0.9033 - 12s/
epoch - 411ms/step
```

Fine-Tuning



Conv block 1: frozen

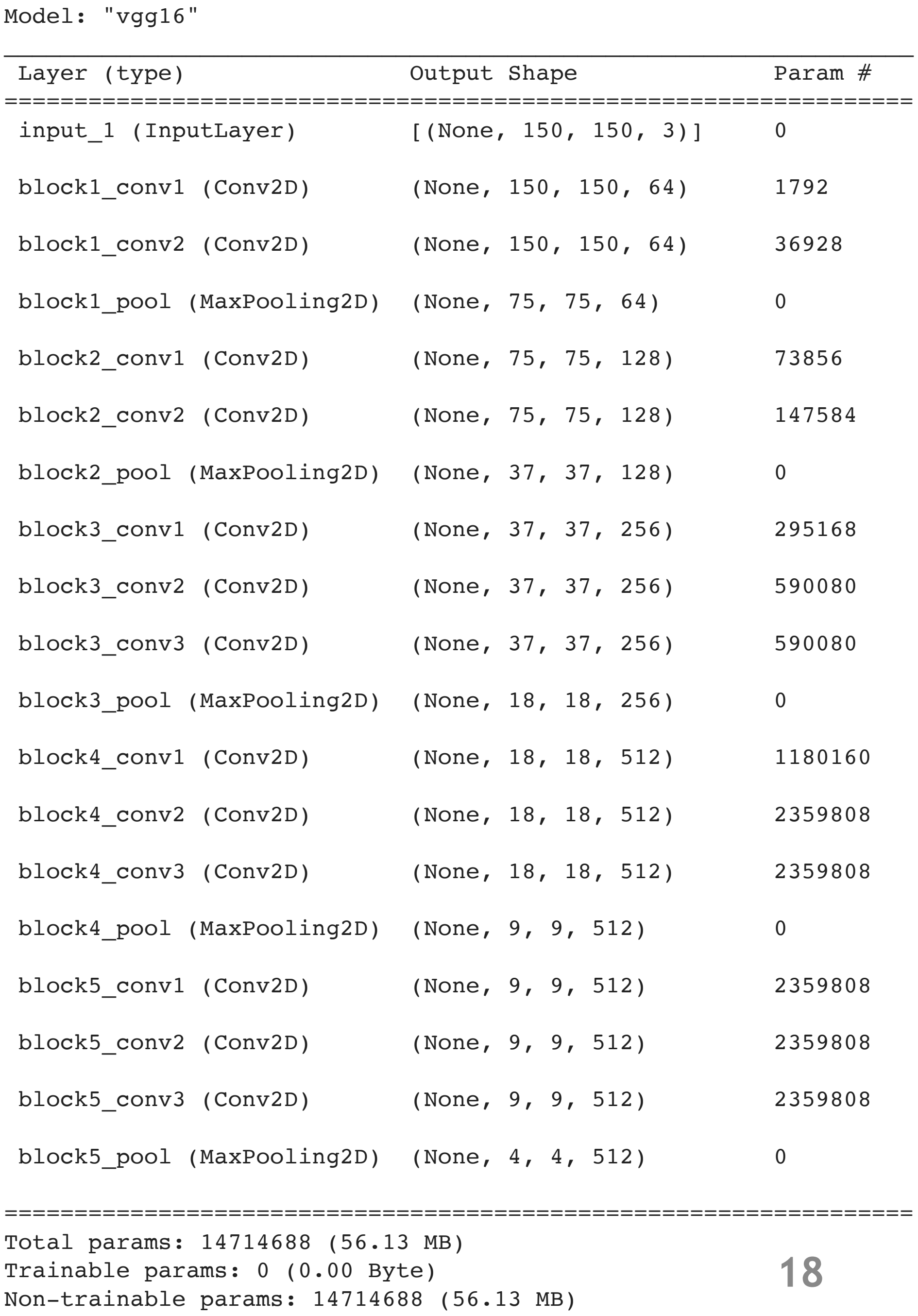
Conv block 2: frozen

Conv block 3: frozen

Conv block 4: frozen

We fine-tune Conv block 5

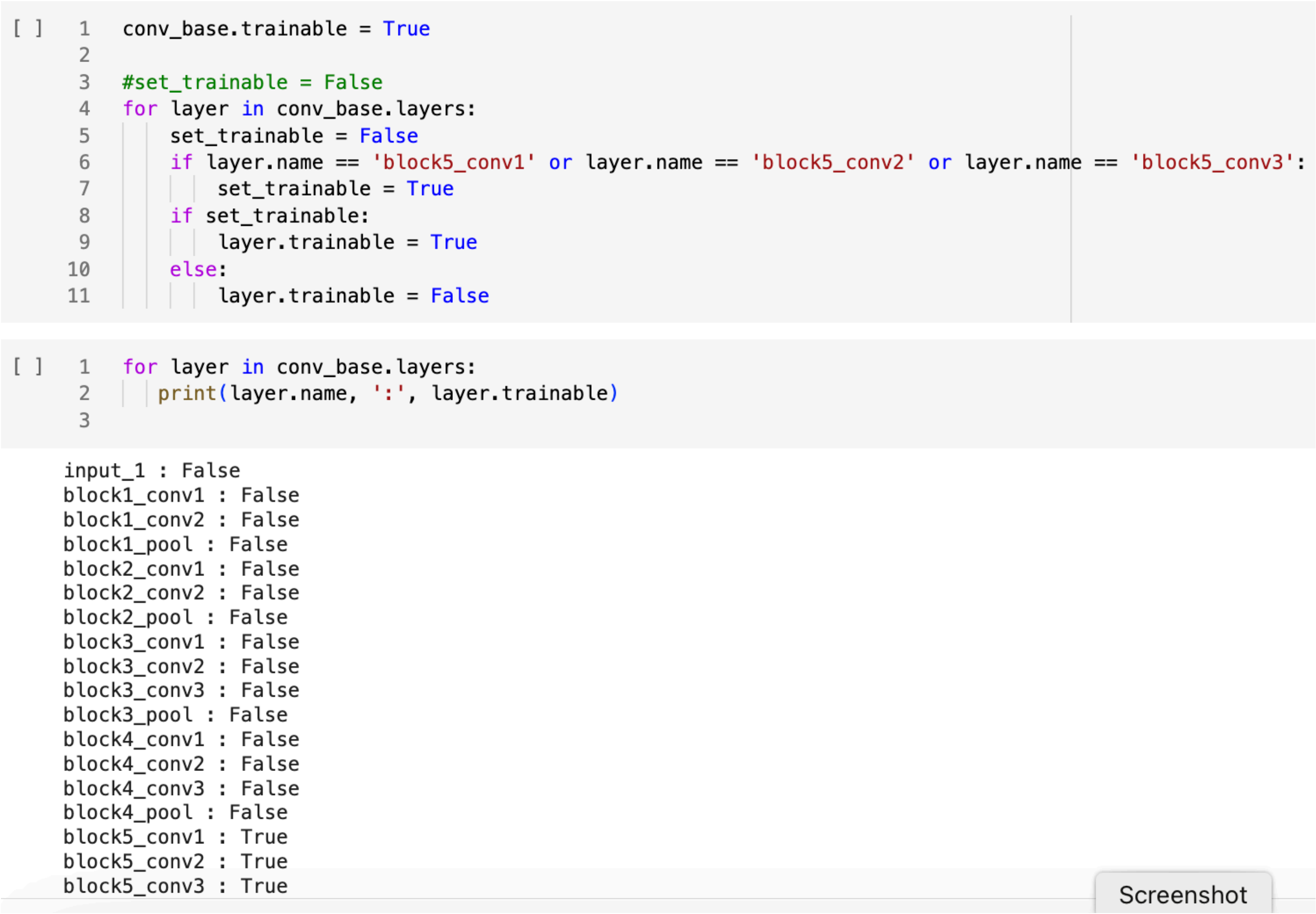
We fine-tune our own fully-connected classifier



```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 150, 150, 3)]     0

 block1_conv1 (Conv2D)       (None, 150, 150, 64)      1792

 block1_conv2 (Conv2D)       (None, 150, 150, 64)      36928

 block1_pool (MaxPooling2D)  (None, 75, 75, 64)        0

 block2_conv1 (Conv2D)       (None, 75, 75, 128)       73856

 block2_conv2 (Conv2D)       (None, 75, 75, 128)       147584

 block2_pool (MaxPooling2D)  (None, 37, 37, 128)       0

 block3_conv1 (Conv2D)       (None, 37, 37, 256)       295168

 block3_conv2 (Conv2D)       (None, 37, 37, 256)       590080

 block3_conv3 (Conv2D)       (None, 37, 37, 256)       590080

 block3_pool (MaxPooling2D)  (None, 18, 18, 256)       0

 block4_conv1 (Conv2D)       (None, 18, 18, 512)       1180160

 block4_conv2 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_conv3 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 9, 9, 512)         0

 block5_conv1 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv2 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv3 (Conv2D)       (None, 9, 9, 512)         2359808

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

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 0 (0.00 Byte)
Non-trainable params: 14714688 (56.13 MB)
```

Fine-tune the last 3 convolutional layers

Let's set this up, starting from where we left off in the previous example:

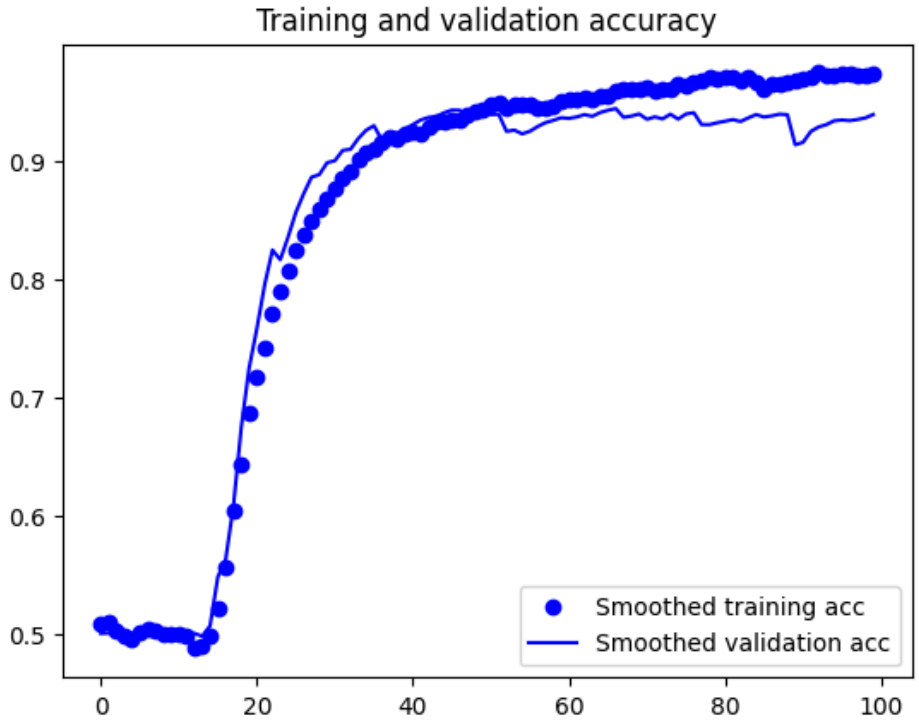


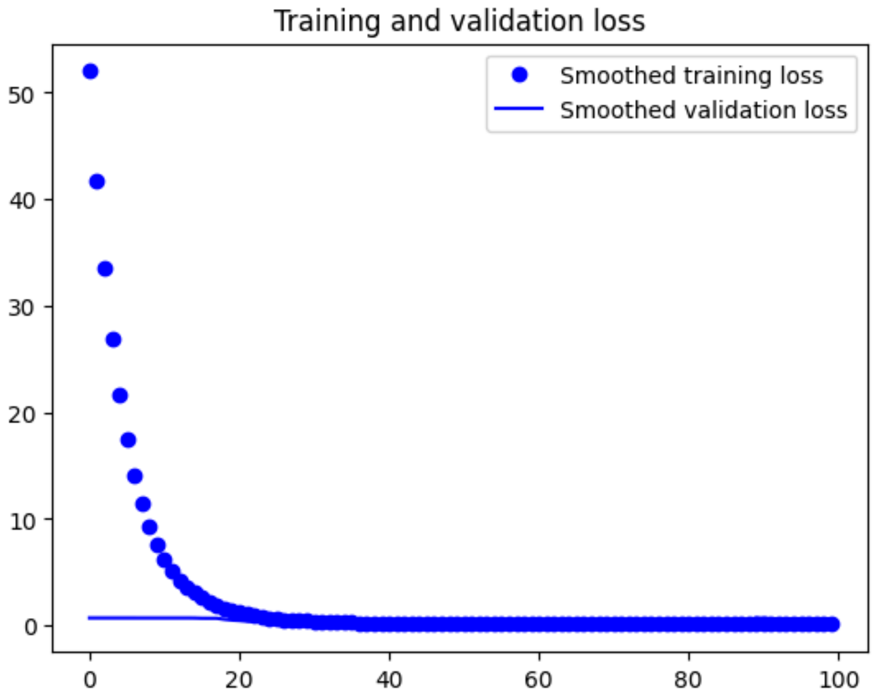
```
conv_base.trainable = True

#set_trainable = False
for layer in conv_base.layers:
    set_trainable = False
    if layer.name == 'block5_conv1' or layer.name == 'block5_conv2' or layer.name == 'block5_conv3':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

```
for layer in conv_base.layers:
  print(layer.name, ':', layer.trainable)

```

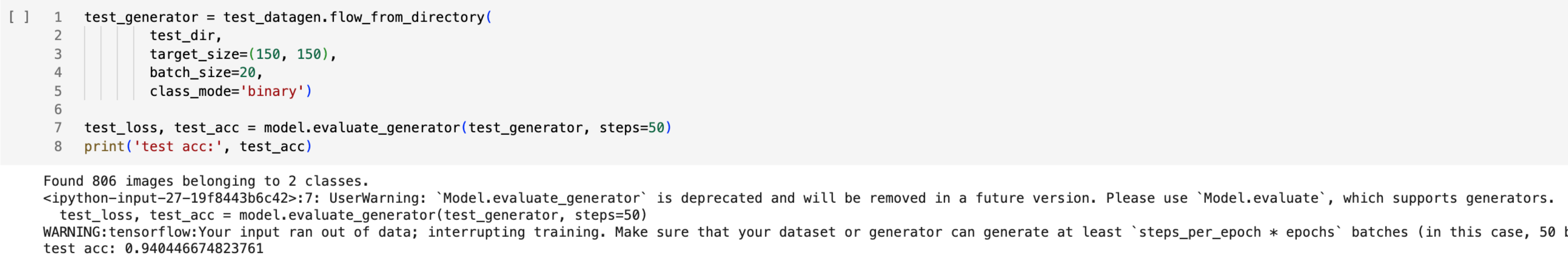
```
input_1 : False
block1_conv1 : False
block1_conv2 : False
block1_pool : False
block2_conv1 : False
block2_conv2 : False
block2_pool : False
block3_conv1 : False
block3_conv2 : False
block3_conv3 : False
block3_pool : False
block4_conv1 : False
block4_conv2 : False
block4_conv3 : False
block4_pool : False
block5_conv1 : True
block5_conv2 : True
block5_conv3 : True
```

Test Accuracy 94.04% & Validation Accuracy 95%


Training and validation accuracy


Training and validation loss

```
Epoch 100/100
30/30 [==============================] - 6s 217ms/step - loss: 0.0695 - acc: 0.9767
- val_loss: 0.5351 - val_acc: 0.9500
```

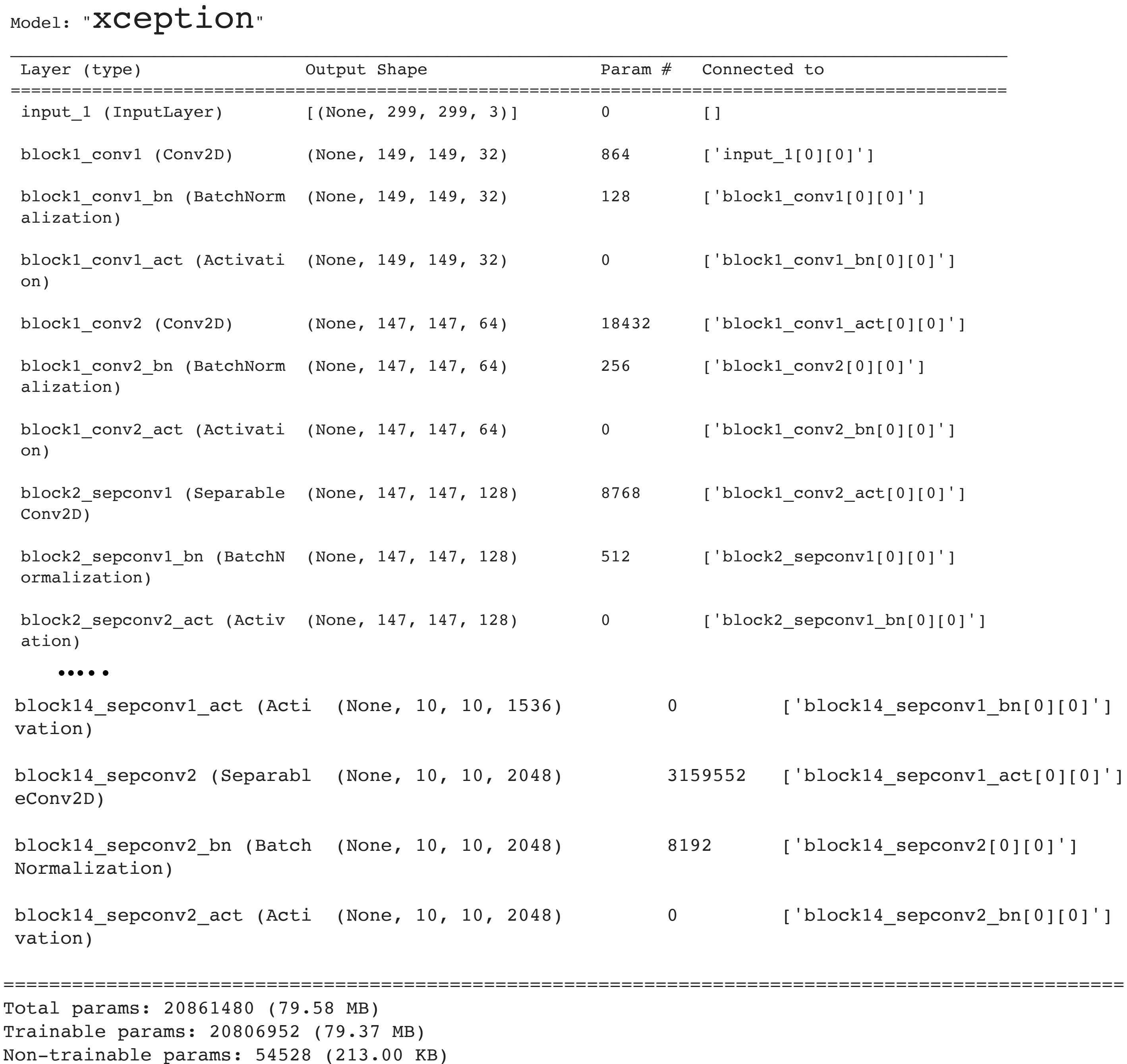


```
[ ]  test_generator = test_datagen.flow_from_directory(
             test_dir,
             target_size=(150, 150),
             batch_size=20,
             class_mode='binary')

     test_loss, test_acc = model.evaluate_generator(test_generator, steps=50)
     print('test acc:', test_acc)
```

```
Found 806 images belonging to 2 classes.
<ipython-input-27-19f8443b6c42>:7: UserWarning: `Model.evaluate_generator` is deprecated and will be removed in a future version. Please use `Model.evaluate`, which supports generators.
  test_loss, test_acc = model.evaluate_generator(test_generator, steps=50)
WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches (in this case, 50 b
test acc: 0.940446674823761
```

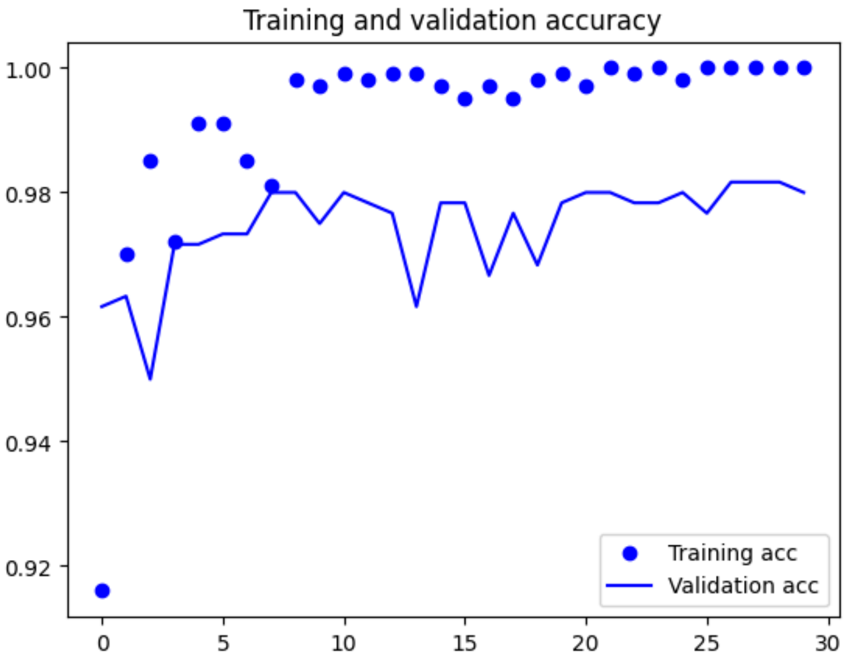
Part 3 - Pre-trained Convolutional Neural Networks - Xception

Pretrained Convolutional Neural Networks - Xception

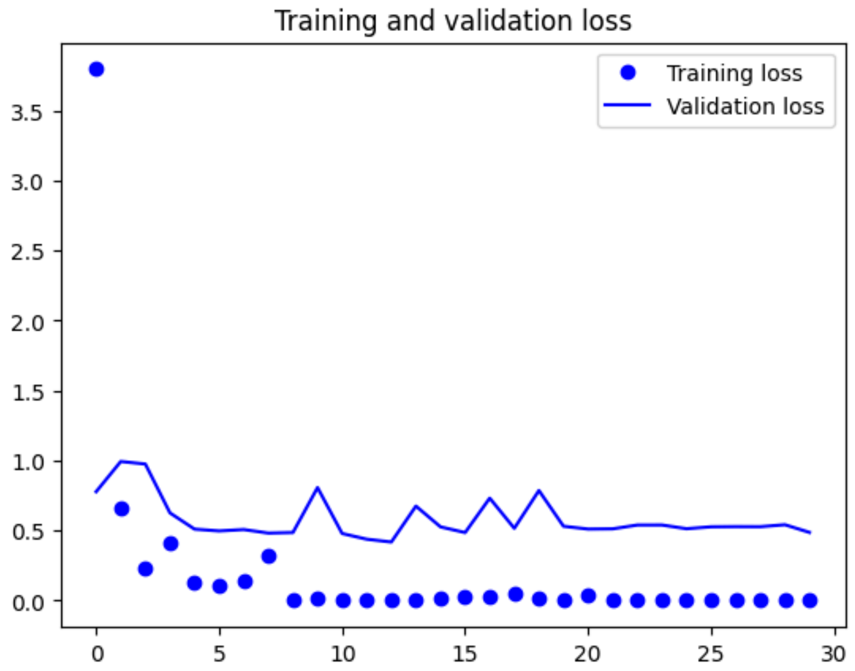


```
Model: "xception"
____________________________________________________________________________________________________
 Layer (type)                   Output Shape             Param #   Connected to
====================================================================================================
 input_1 (InputLayer)           [(None, 299, 299, 3)]    0         []

 block1_conv1 (Conv2D)          (None, 149, 149, 32)     864       ['input_1[0][0]']

 block1_conv1_bn (BatchNorm     (None, 149, 149, 32)     128       ['block1_conv1[0][0]']
 alization)

 block1_conv1_act (Activati     (None, 149, 149, 32)     0         ['block1_conv1_bn[0][0]']
 on)

 block1_conv2 (Conv2D)          (None, 147, 147, 64)     18432     ['block1_conv1_act[0][0]']

 block1_conv2_bn (BatchNorm     (None, 147, 147, 64)     256       ['block1_conv2[0][0]']
 alization)

 block1_conv2_act (Activati     (None, 147, 147, 64)     0         ['block1_conv2_bn[0][0]']
 on)

 block2_sepconv1 (Separable     (None, 147, 147, 128)    8768      ['block1_conv2_act[0][0]']
 Conv2D)

 block2_sepconv1_bn (BatchN     (None, 147, 147, 128)    512       ['block2_sepconv1[0][0]']
 ormalization)

 block2_sepconv2_act (Activ     (None, 147, 147, 128)    0         ['block2_sepconv1_bn[0][0]']
 ation)
    .....
 block14_sepconv1_act (Acti   (None, 10, 10, 1536)         0         ['block14_sepconv1_bn[0][0]']
 vation)

 block14_sepconv2 (Separabl   (None, 10, 10, 2048)         3159552   ['block14_sepconv1_act[0][0]']
 eConv2D)

 block14_sepconv2_bn (Batch   (None, 10, 10, 2048)         8192      ['block14_sepconv2[0][0]']
 Normalization)

 block14_sepconv2_act (Acti   (None, 10, 10, 2048)         0         ['block14_sepconv2_bn[0][0]']
 vation)

=================================================================================================
Total params: 20861480 (79.58 MB)
Trainable params: 20806952 (79.37 MB)
Non-trainable params: 54528 (213.00 KB)
```

Validation Accuracy 98%



Training and validation accuracy

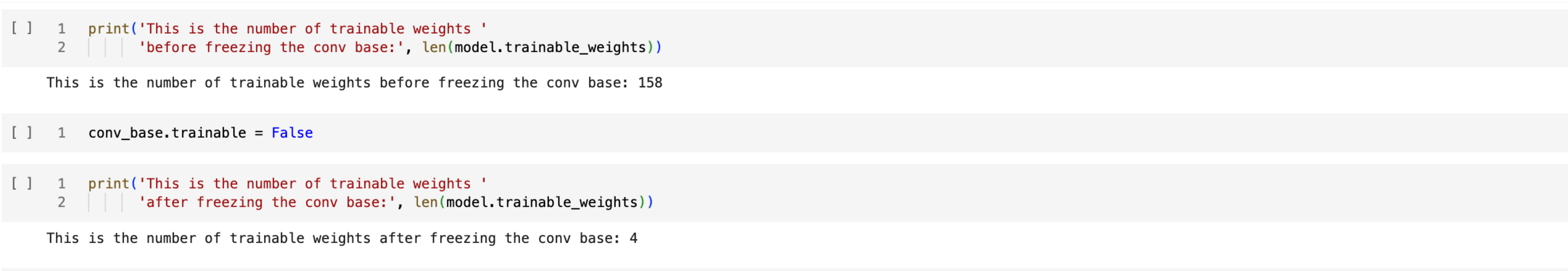


Training and validation loss

```
Epoch 30/30
50/50 [==============================] - 1s 19ms/step - loss: 1.7528e-04 - acc:
1.0000 - val_loss: 0.4869 - val_acc: 0.9800
```

Building the Exception and layering the dense layers and then freezing them

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 xception (Functional)       (None, 10, 10, 2048)      20861480

 flatten (Flatten)           (None, 204800)            0

 dense_2 (Dense)             (None, 256)               52429056

 dense_3 (Dense)             (None, 1)                 257

=================================================================
Total params: 73290793 (279.58 MB)
Trainable params: 73236265 (279.37 MB)
Non-trainable params: 54528 (213.00 KB)
_________________________________________________________________
```



```python
print('This is the number of trainable weights '
      'before freezing the conv base:', len(model.trainable_weights))
```

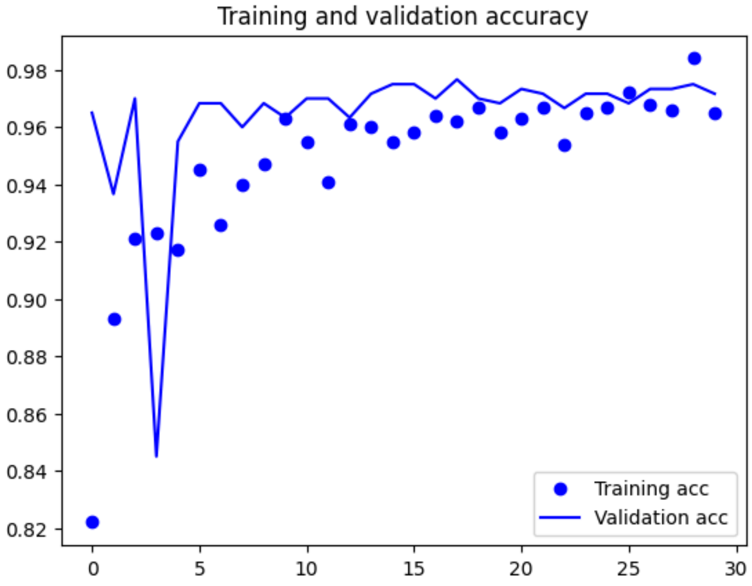
```
This is the number of trainable weights before freezing the conv base: 158
```

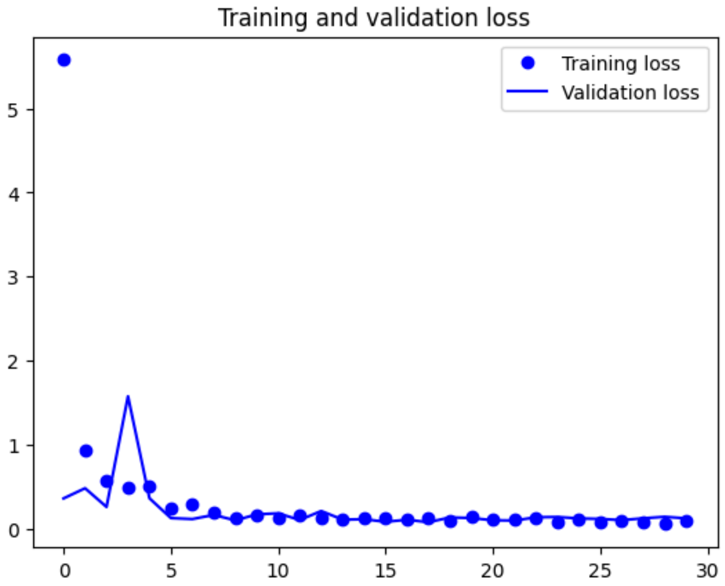
```python
conv_base.trainable = False
```

```python
print('This is the number of trainable weights '
      'after freezing the conv base:', len(model.trainable_weights))
```

```
This is the number of trainable weights after freezing the conv base: 4
```

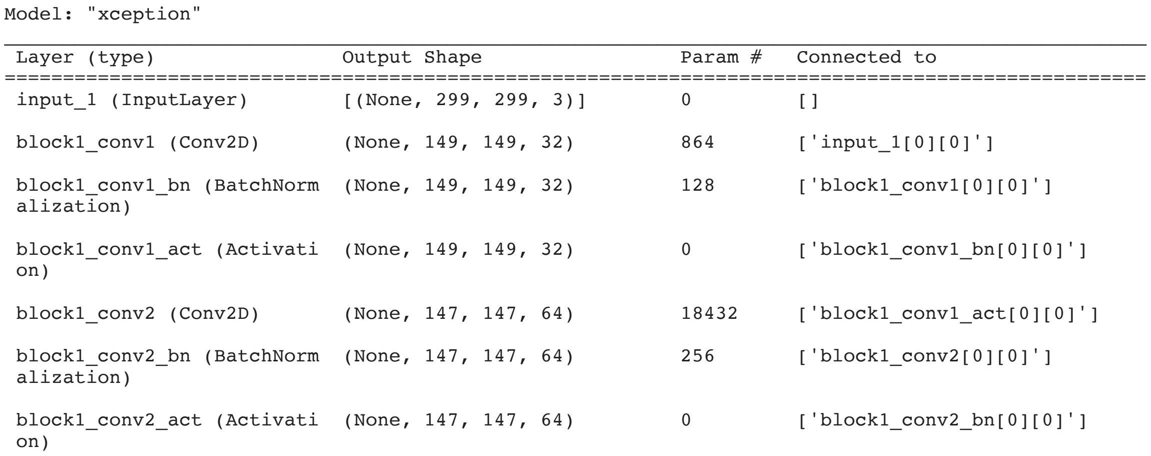
Validation Accuracy 97.17%





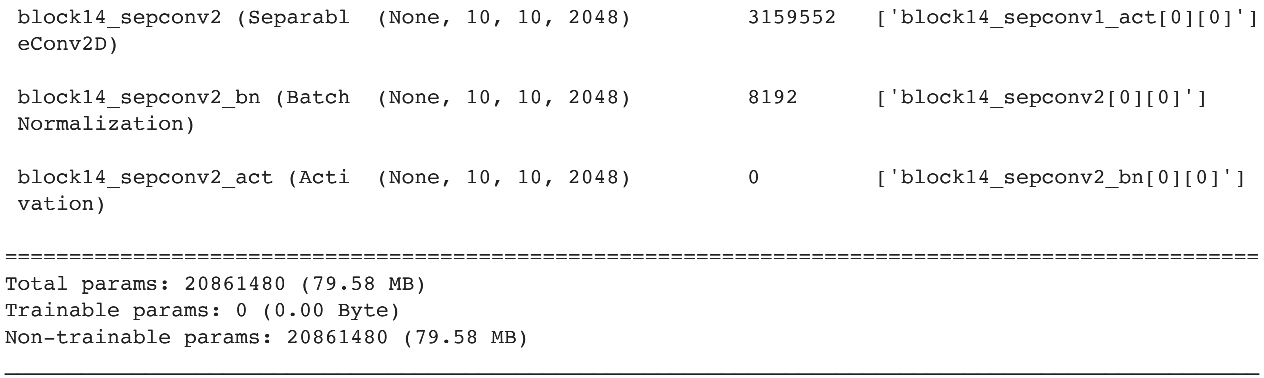
```
Epoch 30/30
50/50 - 29s - loss: 0.0867 - acc: 0.9650 - val_loss: 0.1237 - val_acc: 0.9717 - 29s/
epoch - 575ms/step
```

Fine Tuning



```
Model: "xception"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 299, 299, 3)]  0         []

 block1_conv1 (Conv2D)          (None, 149, 149, 32)  864        ['input_1[0][0]']

 block1_conv1_bn (BatchNorm     (None, 149, 149, 32)  128        ['block1_conv1[0][0]']
 alization)

 block1_conv1_act (Activati     (None, 149, 149, 32)  0          ['block1_conv1_bn[0][0]']
 on)

 block1_conv2 (Conv2D)          (None, 147, 147, 64)  18432      ['block1_conv1_act[0][0]']

 block1_conv2_bn (BatchNorm     (None, 147, 147, 64)  256        ['block1_conv2[0][0]']
 alization)

 block1_conv2_act (Activati     (None, 147, 147, 64)  0          ['block1_conv2_bn[0][0]']
 on)
```

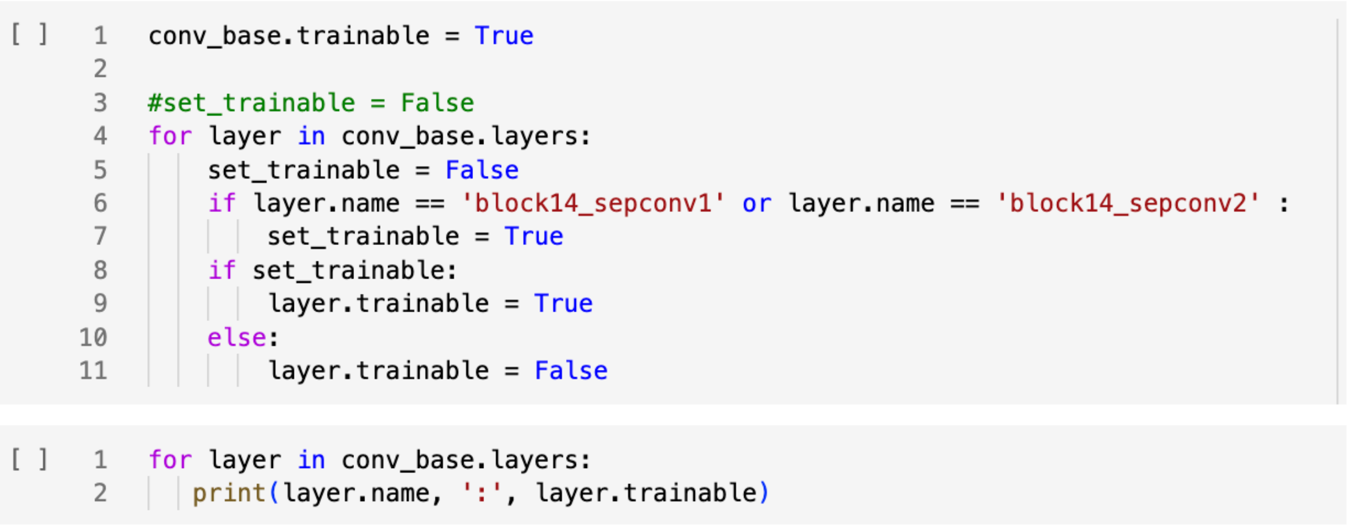
....



```
 block14_sepconv2 (Separabl  (None, 10, 10, 2048)         3159552   ['block14_sepconv1_act[0][0]']
 eConv2D)

 block14_sepconv2_bn (Batch  (None, 10, 10, 2048)         8192      ['block14_sepconv2[0][0]']
 Normalization)

 block14_sepconv2_act (Acti  (None, 10, 10, 2048)         0         ['block14_sepconv2_bn[0][0]']
 vation)

==================================================================================================
Total params: 20861480 (79.58 MB)
Trainable params: 0 (0.00 Byte)
Non-trainable params: 20861480 (79.58 MB)
__________________________________________________________________________________________________
```

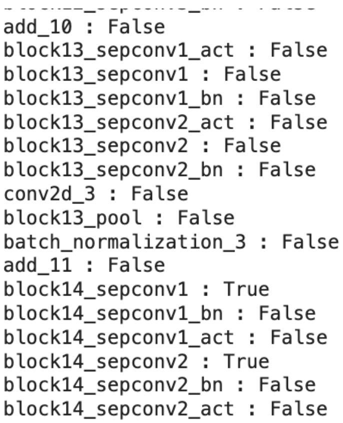
Fine Tune the last 2 Separable Conv. Layers

Fine Tune the last two Separable Convolutional layers in the last block of the Xception Model.



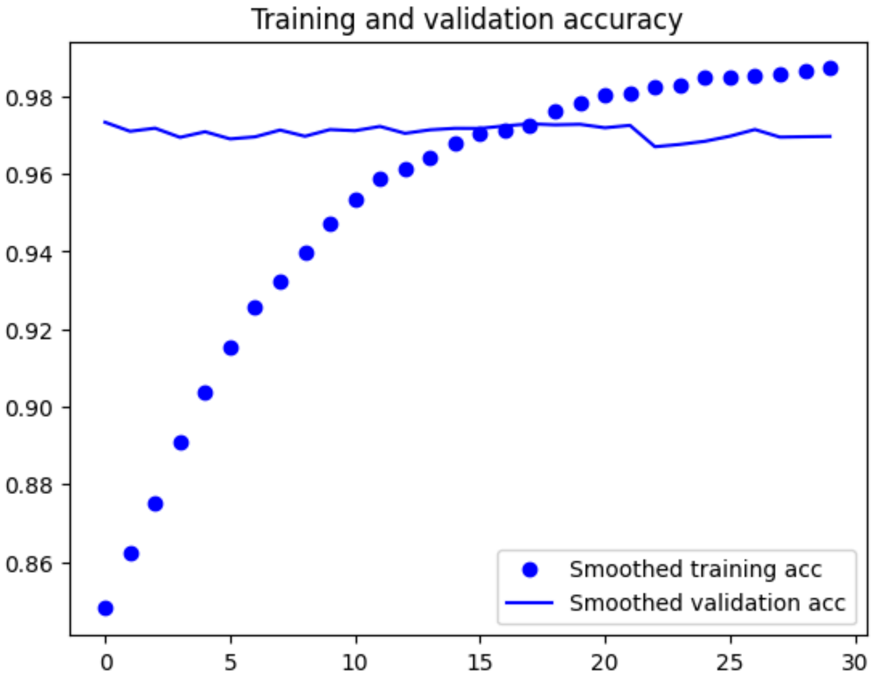
```python
conv_base.trainable = True

#set_trainable = False
for layer in conv_base.layers:
    set_trainable = False
    if layer.name == 'block14_sepconv1' or layer.name == 'block14_sepconv2' :
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

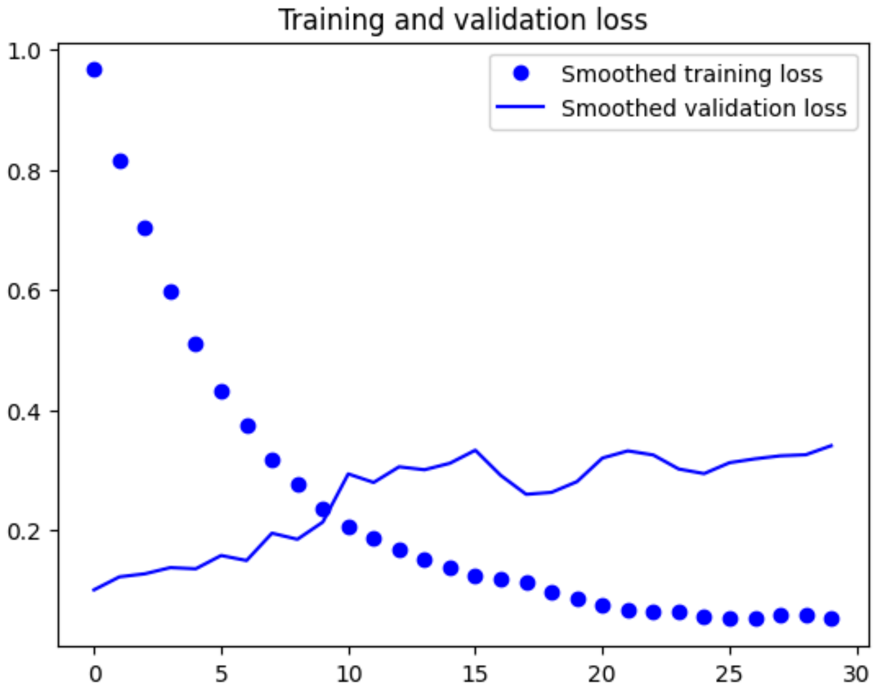
```python
for layer in conv_base.layers:
  print(layer.name, ':', layer.trainable)
```



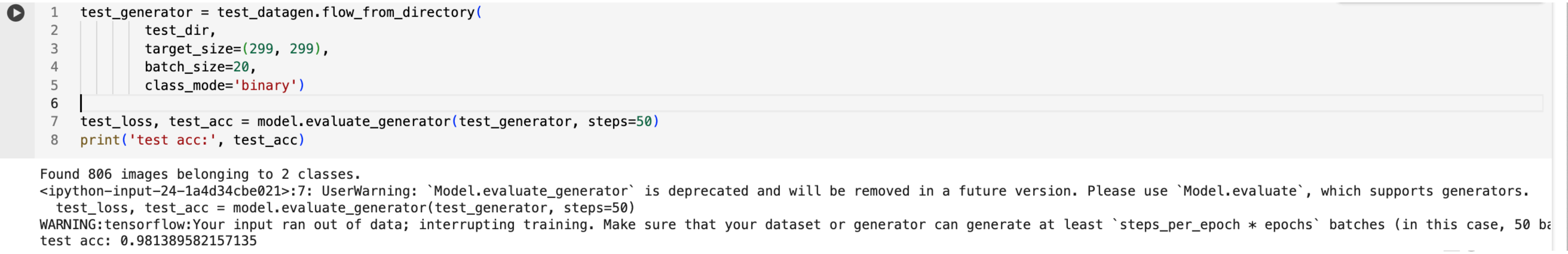
```
add_10 : False
block13_sepconv1_act : False
block13_sepconv1 : False
block13_sepconv1_bn : False
block13_sepconv2_act : False
block13_sepconv2 : False
block13_sepconv2_bn : False
conv2d_3 : False
block13_pool : False
batch_normalization_3 : False
add_11 : False
block14_sepconv1 : True
block14_sepconv1_bn : False
block14_sepconv1_act : False
block14_sepconv2 : True
block14_sepconv2_bn : False
block14_sepconv2_act : False
```

Accuracy is extremely high with Xception model and fine tuning it at 98.13% vs that of VGG16 at 94.04%





```
Epoch 30/30
50/50 [==============================] - 31s 628ms/step - loss: 0.0295 - acc: 0.9900
- val_loss: 0.4008 - val_acc: 0.9700
```



```
test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=(299, 299),
        batch_size=20,
        class_mode='binary')

test_loss, test_acc = model.evaluate_generator(test_generator, steps=50)
print('test acc:', test_acc)
```

```
Found 806 images belonging to 2 classes.
<ipython-input-24-1a4d34cbe021>:7: UserWarning: `Model.evaluate_generator` is deprecated and will be removed in a future version. Please use `Model.evaluate`, which supports generators.
  test_loss, test_acc = model.evaluate_generator(test_generator, steps=50)
WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches (in this case, 50 ba
test acc: 0.9813895821571350
```

YouTube URLs, Last Page

- Two minute (short): https://youtu.be/uqDYF-L85D8
- 15 minutes (long): https://youtu.be/GPKKnFPIraM