

Final Project  
Case Study in Finance - House Rooms Classification

CSCI E-89 Deep Learning, Fall 2023

Harvard University Extension School

Prof. Zoran B. Djordjević

**Abstract**

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 tyep of room such as diningroom vs bedroom vs et al enables these investment decisions.

The code in this project helps with binary classification of Dining room vs Bedroom 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

The data set leveraged is from Kaggle’s [Home Rooms Image Data Set](https://www.kaggle.com/datasets/robinreni/house-rooms-image-dataset/data)\*.

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

With the Part 1, Convolutional Neural Network the validation accuracy was 82.5%.

With Part 2, Pre-trained Convolutional Neural Network VGG16 and after fine tuning the validation accuracy is 95% and test accuracy is 94.04%.

**Final Result**

**With Part 3, Pre-trained Convolutional Neural Networks - Xception and after fine tuning the validation accuracy is 97% and test accuracy is 98.13%.**

You Tube Videos

* Two minute (short): https://youtu.be/uqDYF-L85D8
* 15 minutes (long): https://youtu.be/GPKKnFPIraM
* [https://www.kaggle.com/datasets/robinreni/house-rooms-image-dataset/data](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fwww.kaggle.com%2Fdatasets%2Frobinreni%2Fhouse-rooms-image-dataset%2Fdata)

**Detailed Report**

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 tyep of room such as Dining Room vs Bed Room vs et al enables these investment decisions.

Technologies Used

Google Colab

Keras

Tensor Flow

Approach

The code in this project helps with binary classification of Dining room vs Bedroom 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

Data Set

The Data is downloaded from Kaggle [https://www.kaggle.com/datasets/robinreni/house-rooms-image-dataset/data](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fwww.kaggle.com%2Fdatasets%2Frobinreni%2Fhouse-rooms-image-dataset%2Fdata)

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

This Kaggle Data set has 1248 bedroom pictures and 1158 dining room pictures which was split along the below lines.

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

=================================================================

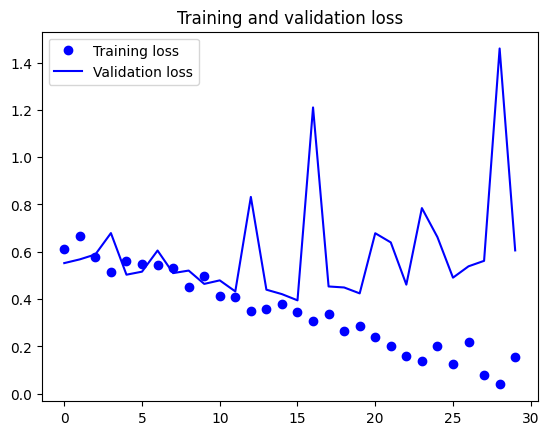
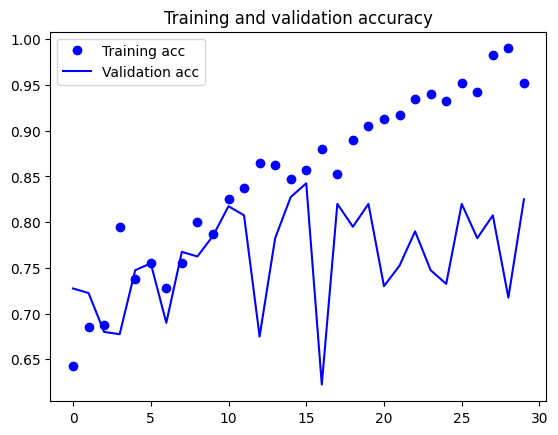
Total params: 3453121 (13.17 MB)

Trainable params: 3453121 (13.17 MB)

Non-trainable params: 0 (0.00 Byte)

Part 1 - Convolutional Neural Networks

The network was built along the below architecture.



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

Tried to Add 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

=================================================================

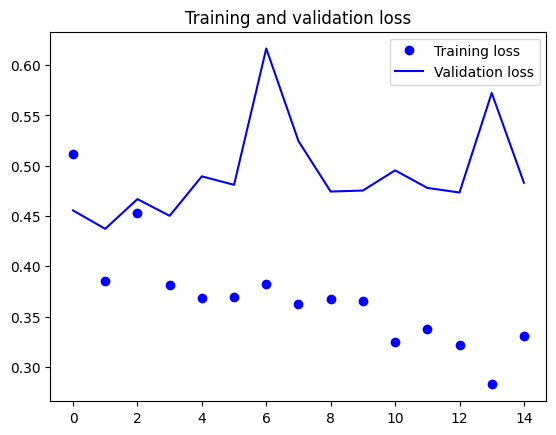
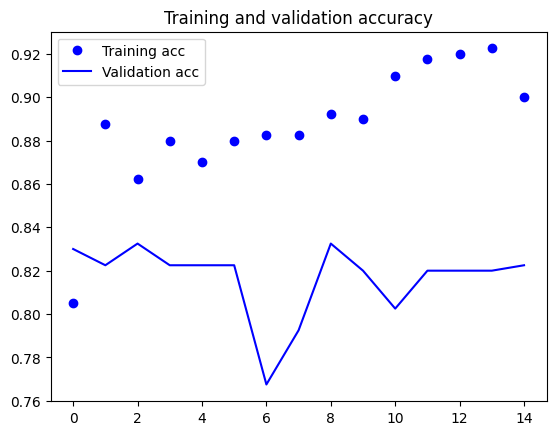
Total params: 3453121 (13.17 MB)

Trainable params: 3453121 (13.17 MB)

Non-trainable params: 0 (0.00 Byte)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The network was built along the below architecture.



Regularization does not help 82.5% vs 82.2%

Epoch 15/15

20/20 [==============================] - 3s 132ms/step - loss: 0.3307 - acc: 0.9000 - val\_loss: 0.4830 - val\_acc: 0.8225

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]']

bleConv2D)

max\_pooling2d\_11 (MaxPooli (None, 10, 10, 256) 0 ['separable\_conv2d\_7[0][0]']

ng2D)

batch\_normalization\_8 (Bat (None, 10, 10, 256) 1024 ['max\_pooling2d\_11[0][0]']

chNormalization)

activation\_8 (Activation) (None, 10, 10, 256) 0 ['batch\_normalization\_8[0][0]'

]

separable\_conv2d\_8 (Separa (None, 10, 10, 512) 133376 ['activation\_8[0][0]']

bleConv2D)

batch\_normalization\_9 (Bat (None, 10, 10, 512) 2048 ['separable\_conv2d\_8[0][0]']

chNormalization)

activation\_9 (Activation) (None, 10, 10, 512) 0 ['batch\_normalization\_9[0][0]'

]

separable\_conv2d\_9 (Separa (None, 10, 10, 512) 266752 ['activation\_9[0][0]']

bleConv2D)

max\_pooling2d\_12 (MaxPooli (None, 5, 5, 512) 0 ['separable\_conv2d\_9[0][0]']

ng2D)

conv2d\_13 (Conv2D) (None, 5, 5, 512) 131072 ['max\_pooling2d\_11[0][0]']

add (Add) (None, 5, 5, 512) 0 ['max\_pooling2d\_12[0][0]',

'conv2d\_13[0][0]']

global\_average\_pooling2d ( (None, 512) 0 ['add[0][0]']

GlobalAveragePooling2D)

dropout (Dropout) (None, 512) 0 ['global\_average\_pooling2d[0][

0]']

dense\_4 (Dense) (None, 1) 513 ['dropout[0][0]']

==================================================================================================

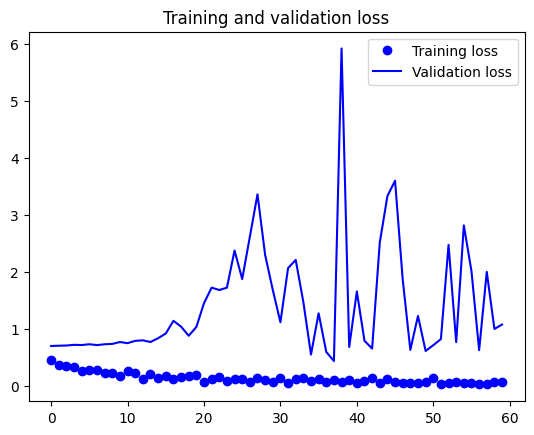
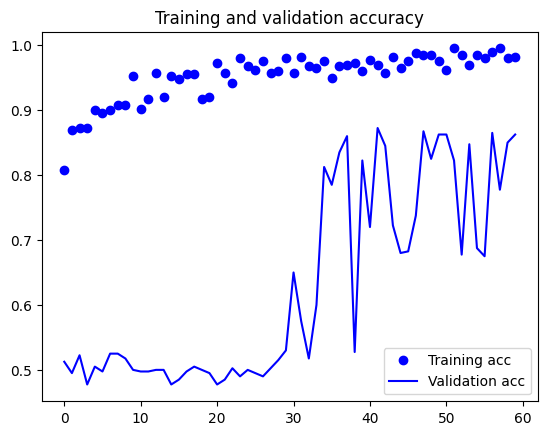
Total params: 677825 (2.59 MB)

Trainable params: 674817 (2.57 MB)

Non-trainable params: 3008 (11.75 KB)

Replacing with Separable Conv2D

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

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

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

Epoch 60/60

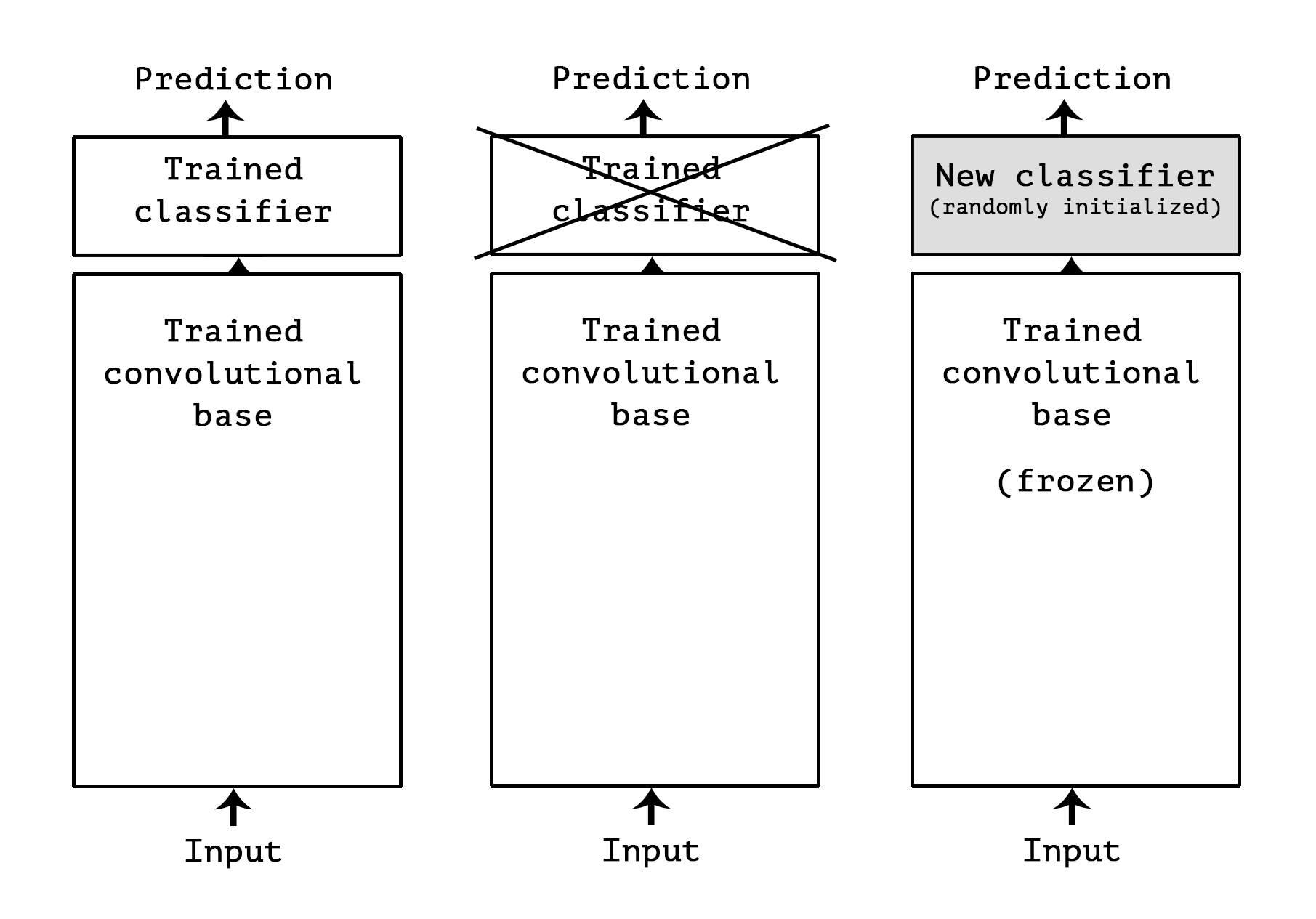
20/20 [==============================] - 3s 131ms/step - loss: 0.0567 - acc: 0.9825 - val\_loss: 1.0717 - val\_acc: 0.8625

**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

Feature extraction consists of using the representations learned by a previous network to extract interesting features from new samples. These features are then run through a new classifier, which is trained from scratch.

As we saw previously, convnets used for image classification comprise two parts: they start with a series of pooling and convolution layers, and they end with a densely-connected classifier. The first part is called the "convolutional base" of the model. In the case of convnets, "feature extraction" will simply consist of taking the convolutional base of a previously-trained network, running the new data through it, and training a new classifier on top of the output.

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)

## Feature extraction

Feature extraction consists of using the representations learned by a previous network to extract interesting features from new samples.

These features are then run through a new classifier, which is trained from scratch.

As we saw previously, convnets used for image classification comprise two parts: they start with a series of pooling and convolution

layers, and they end with a densely-connected classifier. The first part is called the "convolutional base" of the model. In the case of

convnets, "feature extraction" will simply consist of taking the convolutional base of a previously-trained network, running the new data

through it, and training a new classifier on top of the output.

![swapping FC classifiers](https://s3.amazonaws.com/book.keras.io/img/ch5/swapping\_fc\_classifier.png)

Why only reuse the convolutional base? Could we reuse the densely-connected classifier as well? In general, it should be avoided. The

reason is simply that the representations learned by the convolutional base are likely to be more generic and therefore more reusable: the

feature maps of a convnet are presence maps of generic concepts over a picture, which is likely to be useful regardless of the computer

vision problem at hand. On the other end, the representations learned by the classifier will necessarily be very specific to the set of

classes that the model was trained on -- they will only contain information about the presence probability of this or that class in the

entire picture. Additionally, representations found in densely-connected layers no longer contain any information about \_where\_ objects are

located in the input image: these layers get rid of the notion of space, whereas the object location is still described by convolutional

feature maps. For problems where object location matters, densely-connected features would be largely useless.

Note that the level of generality (and therefore reusability) of the representations extracted by specific convolution layers depends on

the depth of the layer in the model. Layers that come earlier in the model extract local, highly generic feature maps (such as visual

edges, colors, and textures), while layers higher-up extract more abstract concepts (such as "cat ear" or "dog eye"). So if your new

dataset differs a lot from the dataset that the original model was trained on, you may be better off using only the first few layers of the

model to do feature extraction, rather than using the entire convolutional base.

In our case, since the ImageNet class set did contain multiple dog and cat classes, it is likely that it would be beneficial to reuse the

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

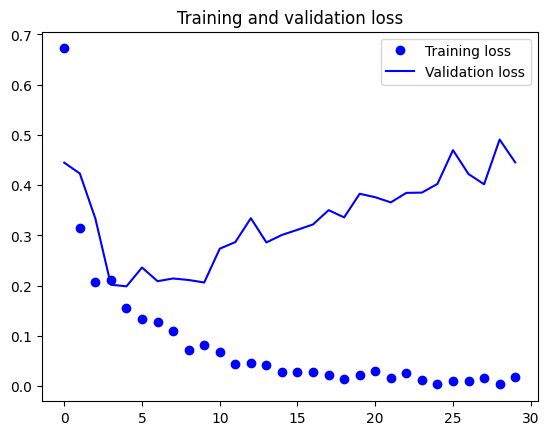
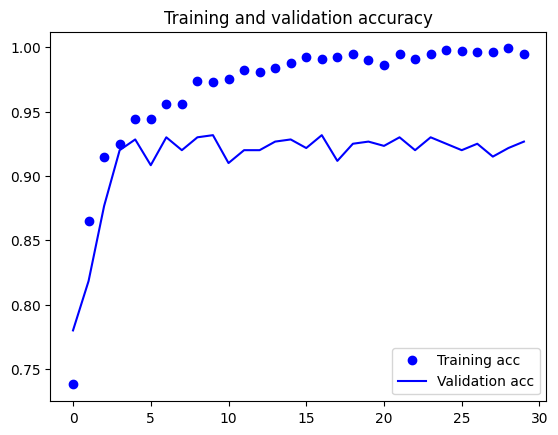
=================================================================

Total params: 14714688 (56.13 MB)

Trainable params: 14714688 (56.13 MB)

Non-trainable params: 0 (0.00 Byte)

information contained in the densely-connected layers of the original model. However, we will chose not to, in order to cover the more

general case where the class set of the new problem does not overlap with the class set of the original model.

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

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

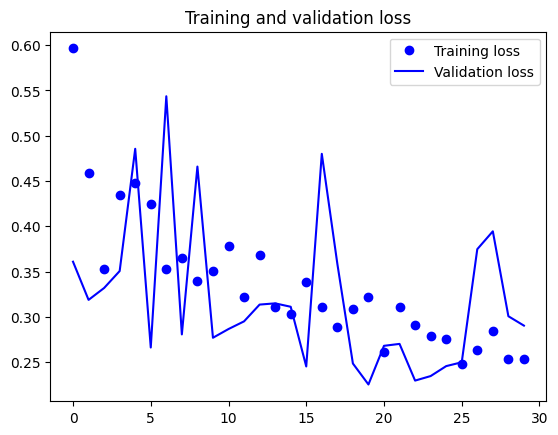
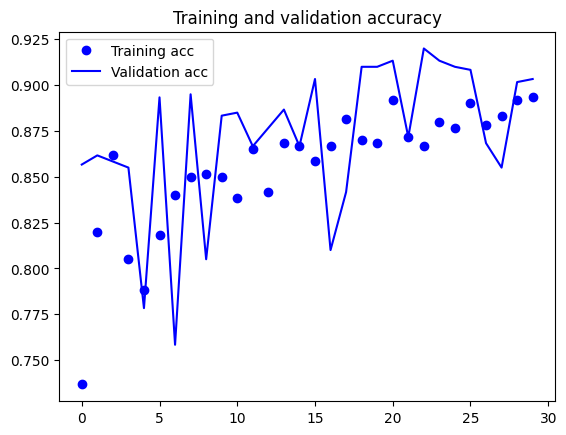
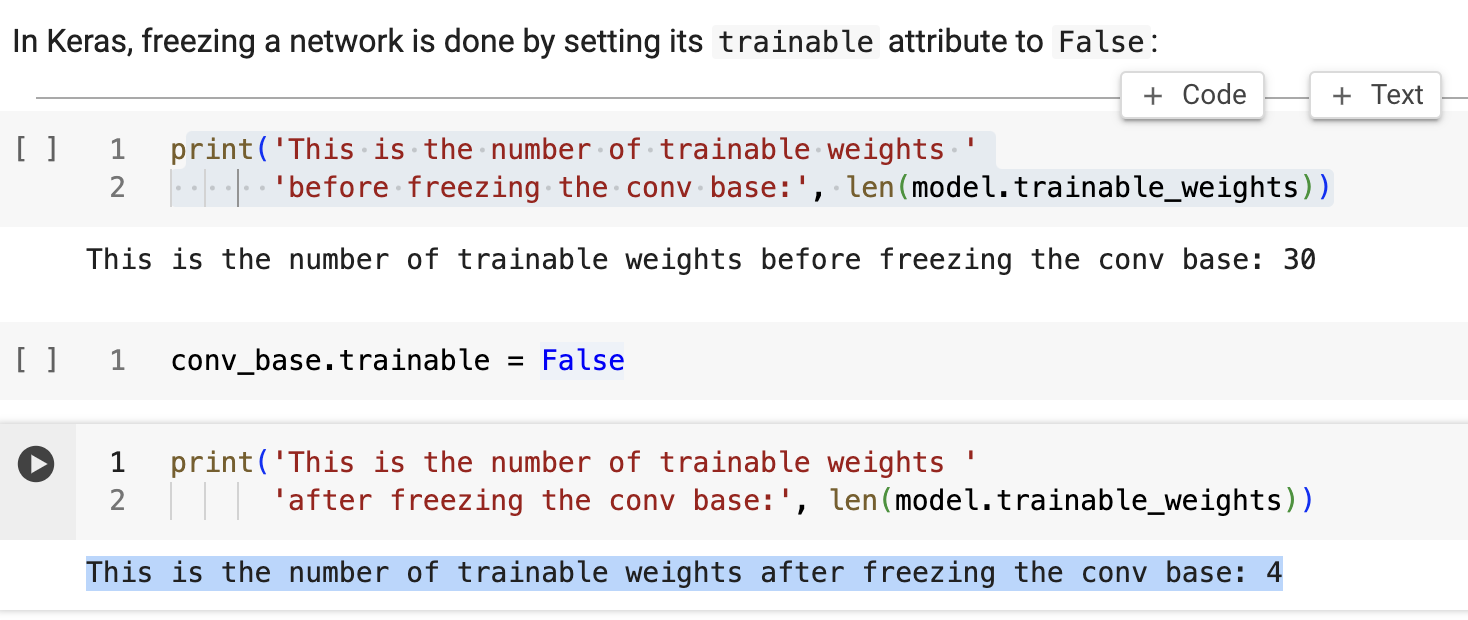
dense\_3 (Dense) (None, 1) 257

=================================================================

Total params: 16812353 (64.13 MB)

Trainable params: 16812353 (64.13 MB)

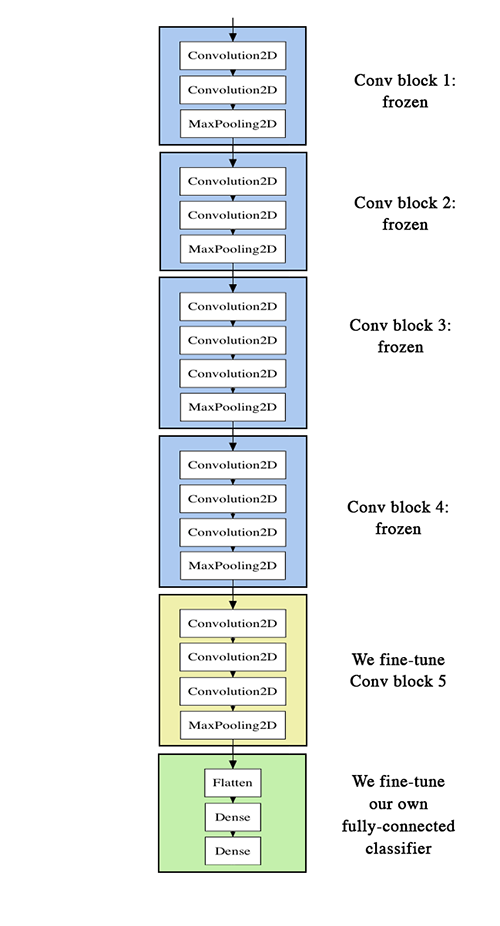
Non-trainable params: 0 (0.00 Byte)

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

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

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

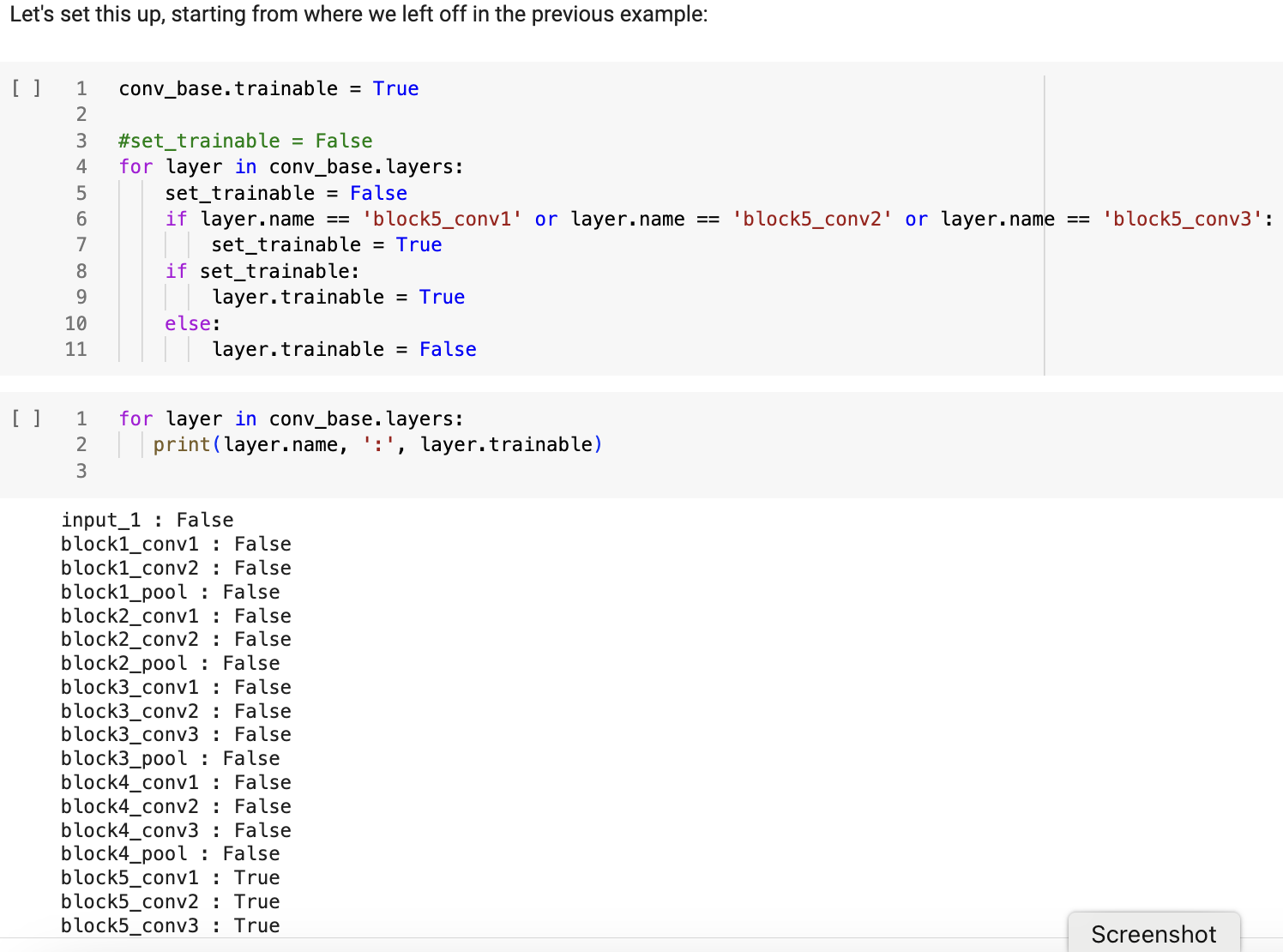
block5\_pool (MaxPooling2D) (None, 4, 4, 512) 0

=================================================================

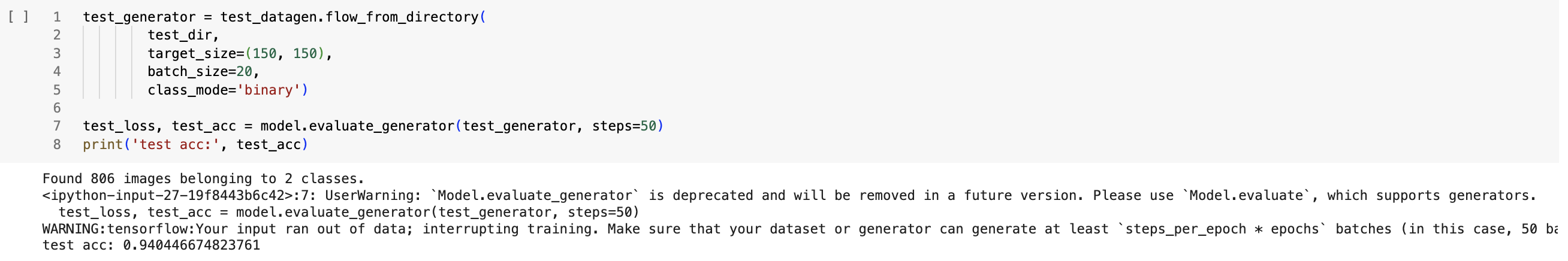
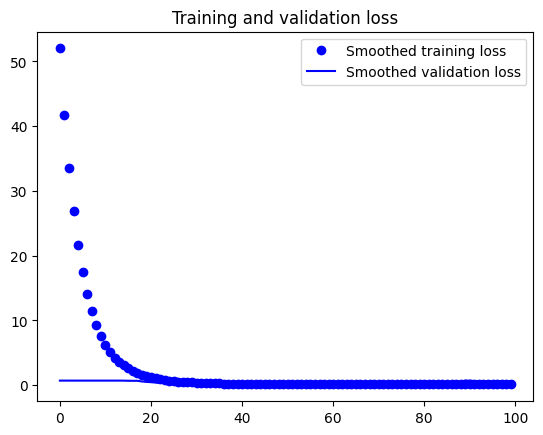
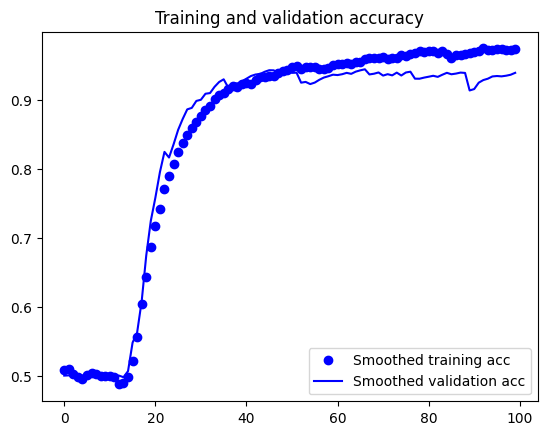
Total params: 14714688 (56.13 MB)

Trainable params: 0 (0.00 Byte)

Non-trainable params: 14714688 (56.13 MB)



**Fine-tune the last 3 convolutional layers**



Test Accuracy 94.04% & Validation Accuracy 95%

Epoch 100/100

30/30 [==============================] - 6s 217ms/step - loss: 0.0695 - acc: 0.9767 - val\_loss: 0.5351 - val\_acc: 0.9500

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)

==================================================================================================

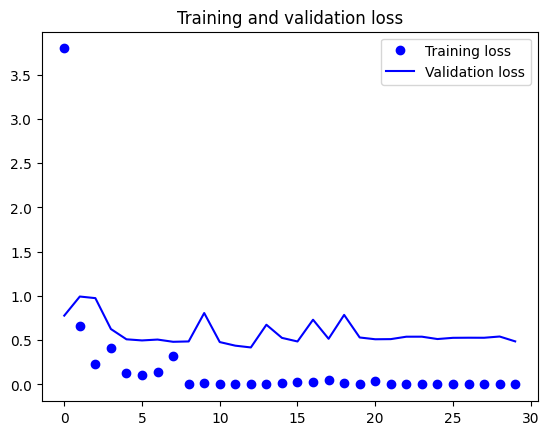
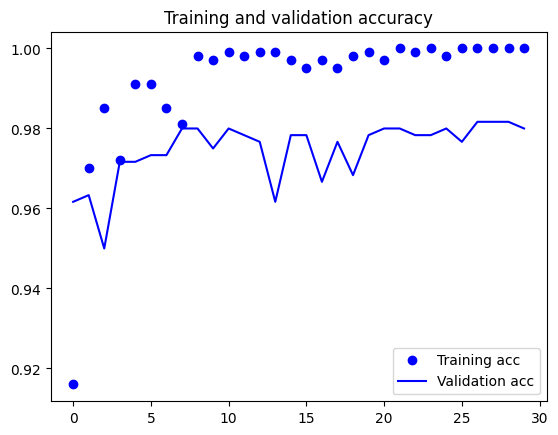
Total params: 20861480 (79.58 MB)

Trainable params: 20806952 (79.37 MB)

Non-trainable params: 54528 (213.00 KB)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

…..



Validation Accuracy 98%

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

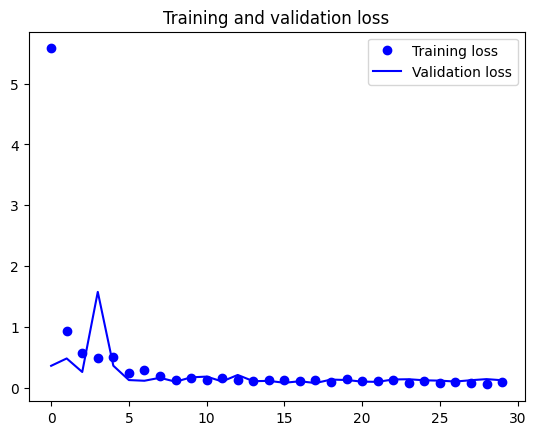
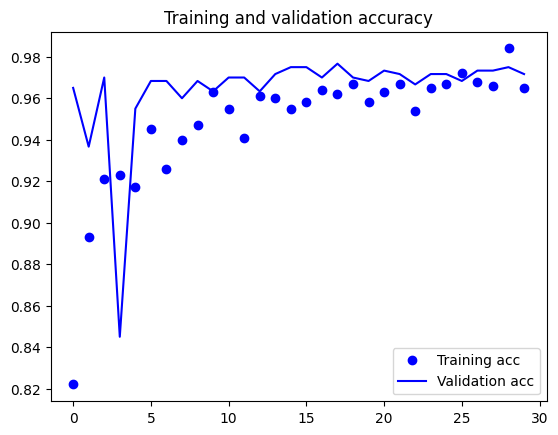
=================================================================

Total params: 73290793 (279.58 MB)

Trainable params: 73236265 (279.37 MB)

Non-trainable params: 54528 (213.00 KB)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Part 3 - Pre-trained Convolutional Neural Networks - Xception

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)

==================================================================================================

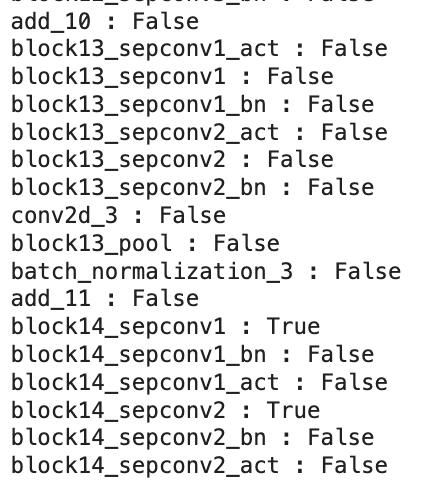
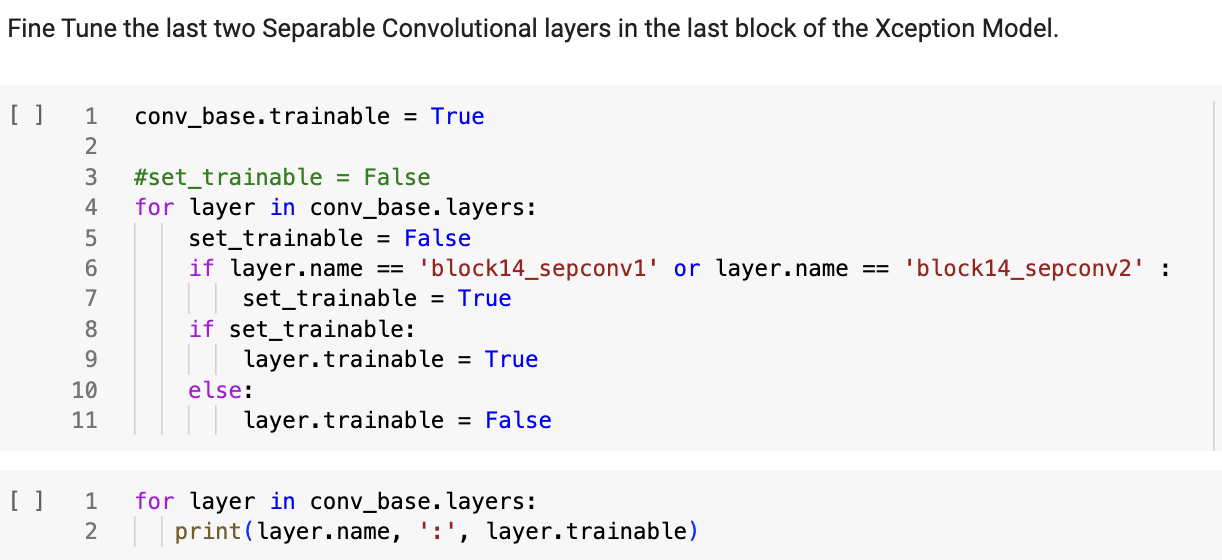
Total params: 20861480 (79.58 MB)

Trainable params: 0 (0.00 Byte)

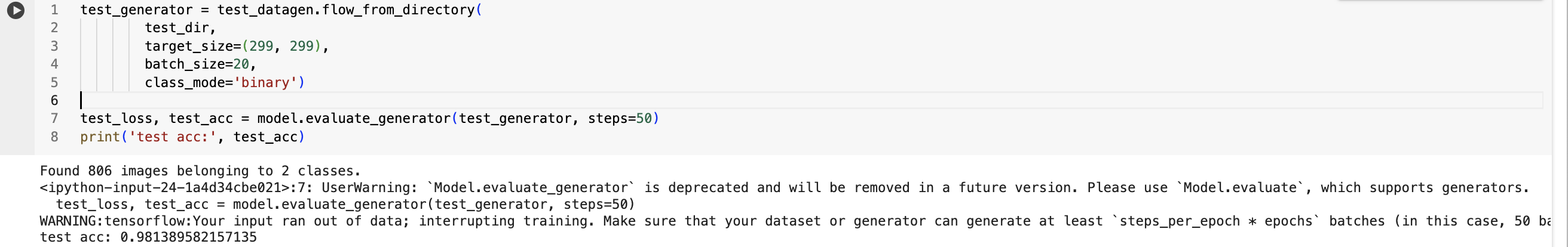
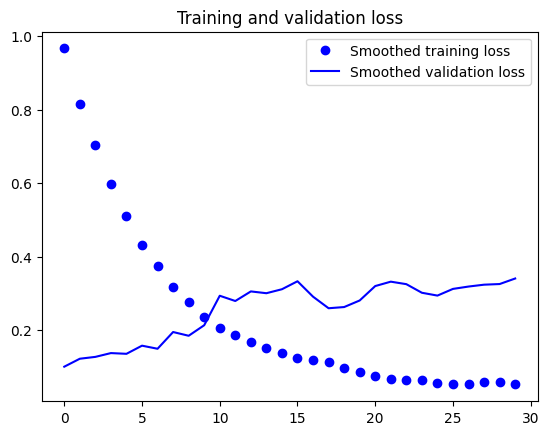
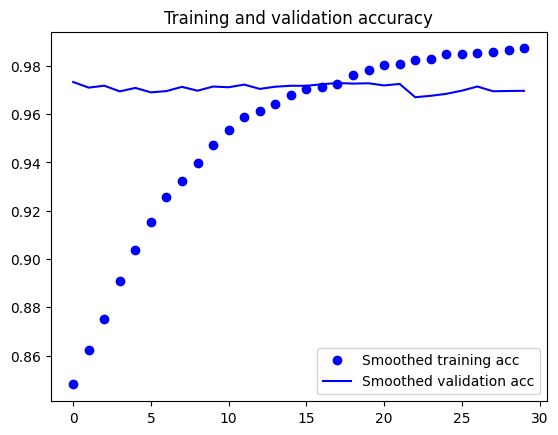
Non-trainable params: 20861480 (79.58 MB)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

….



Fine Tune the last 2 Separable Conv. Layers



Epoch 30/30

50/50 [==============================] - 31s 628ms/step - loss: 0.0295 - acc: 0.9900 - val\_loss: 0.4008 - val\_acc: 0.9700

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