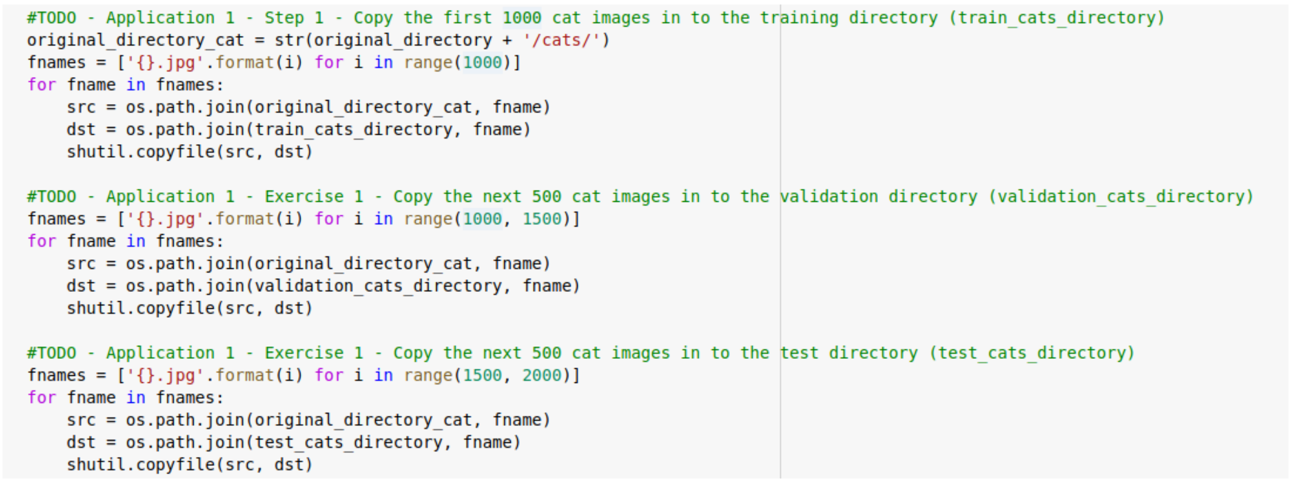
Exercise 1: Copy the first 1000 images with cats from the `original_directory` into the cats training folder (`train_cats_directory`). Copy the next 500 cat images from the `original_directory` into the cats validation folder (`validation_cats_directory`) and the following 500 into the cat testing folder (`test_cats_directory`).

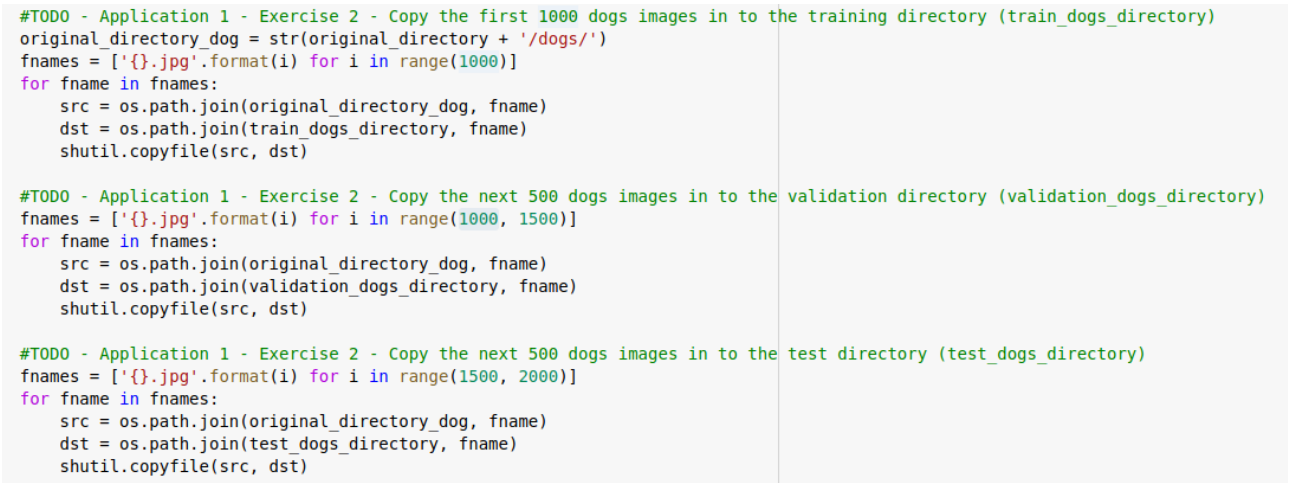
For cats;



```
#TODO - Application 1 - Step 1 - Copy the first 1000 cat images in to the training directory (train_cats_directory)
original_directory_cat = str(original_directory + '/cats/')
fnames = ['{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
    src = os.path.join(original_directory_cat, fname)
    dst = os.path.join(train_cats_directory, fname)
    shutil.copyfile(src, dst)

#TODO - Application 1 - Exercise 1 - Copy the next 500 cat images in to the validation directory (validation_cats_directory)
fnames = ['{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
    src = os.path.join(original_directory_cat, fname)
    dst = os.path.join(validation_cats_directory, fname)
    shutil.copyfile(src, dst)

#TODO - Application 1 - Exercise 1 - Copy the next 500 cat images in to the test directory (test_cats_directory)
fnames = ['{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(original_directory_cat, fname)
    dst = os.path.join(test_cats_directory, fname)
    shutil.copyfile(src, dst)
```

Exercise 2: Copy the first 1000 images with dogs from the `original_directory` into the dogs training folder (`train_dogs_directory`). Copy the next 500 dog images from the `original_directory` into the cats validation folder (`validation_dogs_directory`) and the following 500 into the dogs testing folder (`test_dogs_directory`).

For dogs;



```
#TODO - Application 1 - Exercise 2 - Copy the first 1000 dogs images in to the training directory (train_dogs_directory)
original_directory_dog = str(original_directory + '/dogs/')
fnames = ['{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
    src = os.path.join(original_directory_dog, fname)
    dst = os.path.join(train_dogs_directory, fname)
    shutil.copyfile(src, dst)

#TODO - Application 1 - Exercise 2 - Copy the next 500 dogs images in to the validation directory (validation_dogs_directory)
fnames = ['{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
    src = os.path.join(original_directory_dog, fname)
    dst = os.path.join(validation_dogs_directory, fname)
    shutil.copyfile(src, dst)

#TODO - Application 1 - Exercise 2 - Copy the next 500 dogs images in to the test directory (test_dogs_directory)
fnames = ['{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(original_directory_dog, fname)
    dst = os.path.join(test_dogs_directory, fname)
    shutil.copyfile(src, dst)
```

As a sanity check,

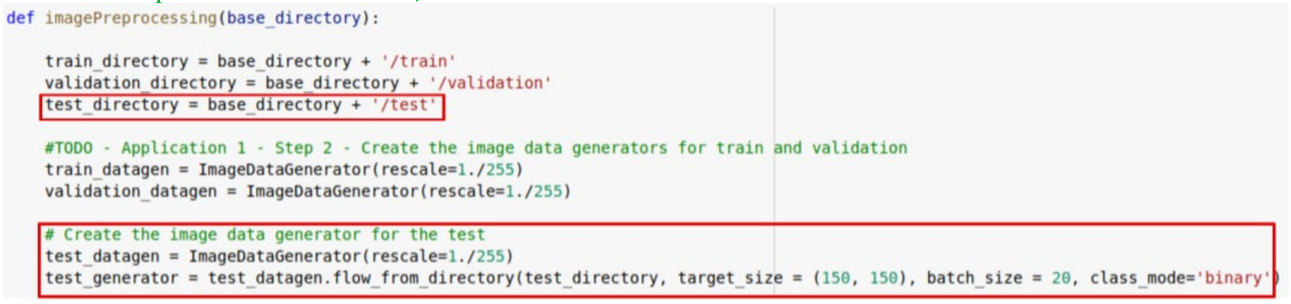
```
[7]  prepareDatabase(original_directory, base_directory)
```

```
Total number of CATS used for training = 1000
Total number of CATS used for validation = 500
Total number of CATS used for testing = 500
Total number of DOGS used for training = 1000
Total number of DOGS used for validation = 500
Total number of DOGS used for testing = 500
```

Exercise 3: Evaluate the model accuracy on the testing dataset.

Hint: First, create a data generator for the testing dataset as for the training and the validation datasets (*cf.* Step 2). Next, use the `model.evaluate_generator` function to obtain the loss and accuracy.

We have implemented as follow;

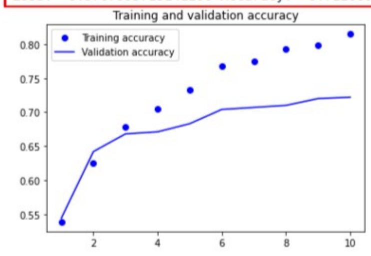


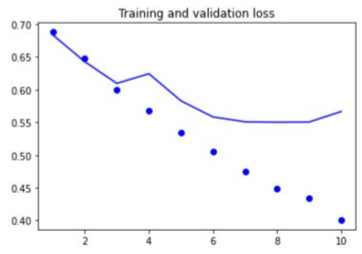
```
def imagePreprocessing(base_directory):

    train_directory = base_directory + '/train'
    validation_directory = base_directory + '/validation'
    test_directory = base_directory + '/test'

    #TODO - Application 1 - Step 2 - Create the image data generators for train and validation
    train_datagen = ImageDataGenerator(rescale=1./255)
    validation_datagen = ImageDataGenerator(rescale=1./255)

    # Create the image data generator for the test
    test_datagen = ImageDataGenerator(rescale=1./255)
    test_generator = test_datagen.flow_from_directory(test_directory, target_size = (150, 150), batch_size = 20, class_mode='binary')
```

Now we have modified the **imageProcessing(base-directory)** function so that it returns **test-generator** as well. The **Loss: 0.5767995715141296** and the **Accuracy: 0.7120000123977661**

```
    return train_generator, validation_generator, test_generator
```

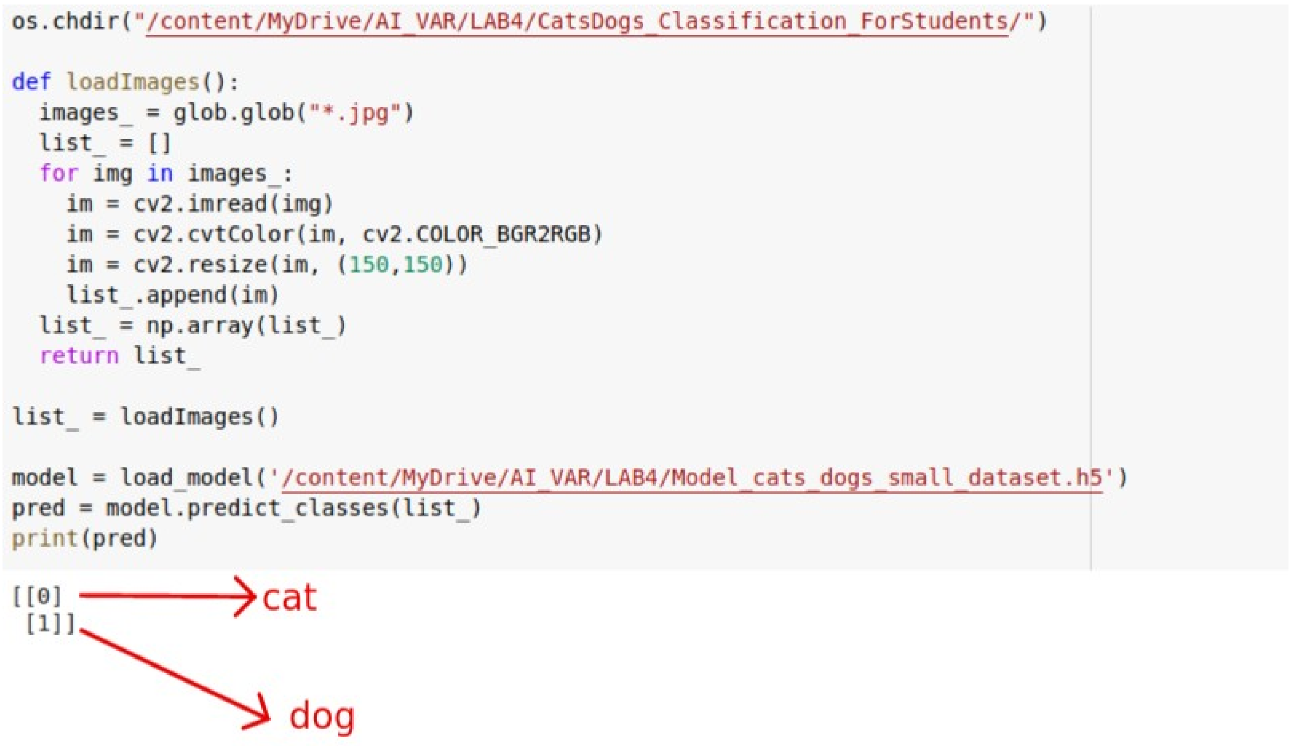
```
100/100 [==============================] - 12s 118ms/step - loss: 0.4480 - accuracy: 0.7925 - val_loss: 0.5502 - val_accuracy: 0.7100
Epoch 9/10
100/100 [==============================] - 12s 117ms/step - loss: 0.4335 - accuracy: 0.7985 - val_loss: 0.5505 - val_accuracy: 0.7200
Epoch 10/10
100/100 [==============================] - 12s 118ms/step - loss: 0.4008 - accuracy: 0.8150 - val_loss: 0.5666 - val_accuracy: 0.7220
WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `
Loss:  0.5767995715141296 Accuracy:  0.7120000123977661
```





Exercise 4: Once fit, we can save the final model to an *.h5 file by calling the *save()* function on the model and pass in the selected filename (`model.save('Model_cats_dogs_small_dataset.h5')`). We can use our saved model to make a prediction on new images. Using the pre-trained model (saved above) write a Python script able to make an automatic prediction regarding the category for the following images "test1.jpg" and "test2.jpg". The prediction will be performed simultaneously for the two images using a batch of images.

It was implemented as follow. Here the class 0 belongs to cat and the class 1 belongs to dog. The loadImages return a tensor of size of **(2, 150, 150, 3)**.



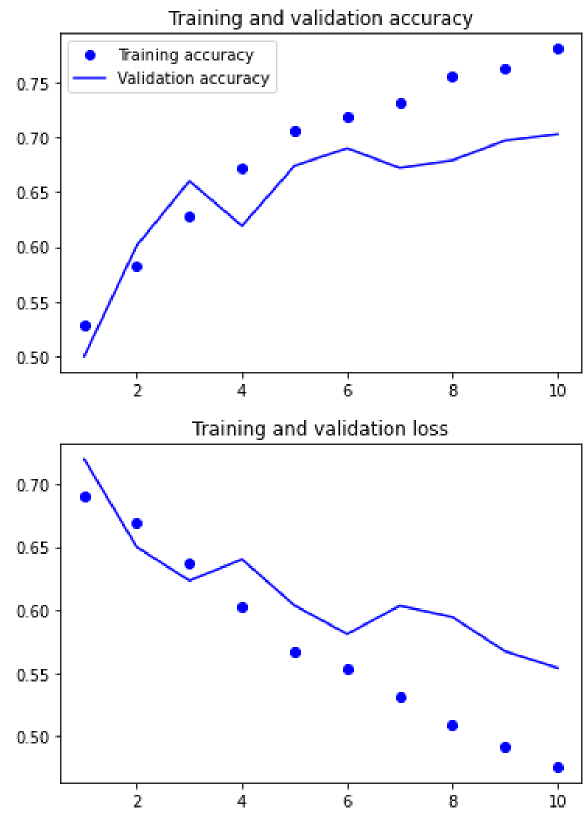
```python
os.chdir("/content/MyDrive/AI_VAR/LAB4/CatsDogs_Classification_ForStudents/")

def loadImages():
  images_ = glob.glob("*.jpg")
  list_ = []
  for img in images_:
    im = cv2.imread(img)
    im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
    im = cv2.resize(im, (150,150))
    list_.append(im)
  list_ = np.array(list_)
  return list_

list_ = loadImages()

model = load_model('/content/MyDrive/AI_VAR/LAB4/Model_cats_dogs_small_dataset.h5')
pred = model.predict_classes(list_)
print(pred)
```

```
[[0]
 [1]]
```

Exercise 5: Try to further increase the system performances (by reducing the overfitting) by adding a dropout layer to your model that randomly excludes 50% of neurons. The layer should be added right before the densely connected classifier.

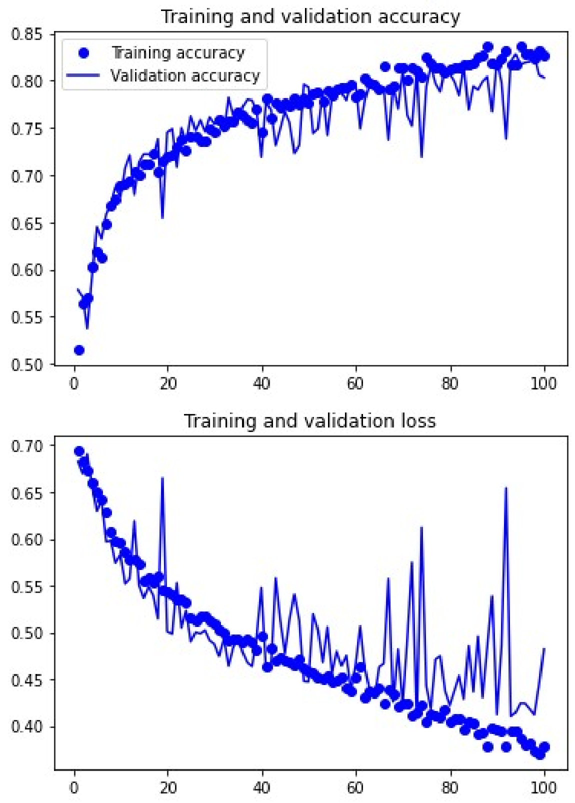
If we add the dropout layer of 50%, we get the following **accuracy** and the **loss**:

```
Loss:  0.565795361995697 Accuracy:  0.7139999866485596
```



Lets apply data augmentation; and the **epoch size 100.** Now we can see the accuracy is 0.79

```
Loss:  0.5287520885467529 Accuracy:  0.796999990940094
```



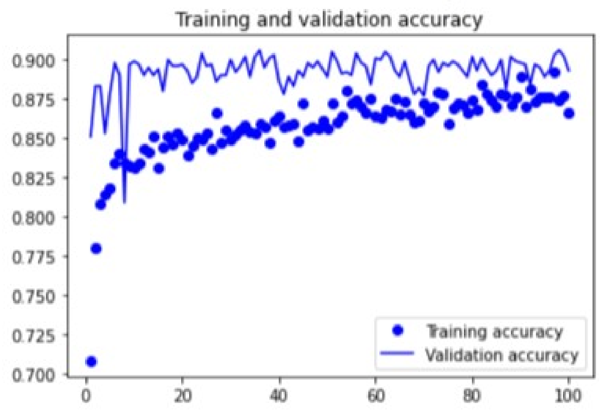
Exercise 6: In this exercise, you will need to classify images of cats and dogs by using transfer learning from a pre-trained network. In the **defineCNNModelVGGPretrained** method, load the VGG16 network, previously trained on ImageNet, to extract interesting features from cat and dog images, and then (using the pretrained filters) train a dogs-versus-cats classifier on top of these features.

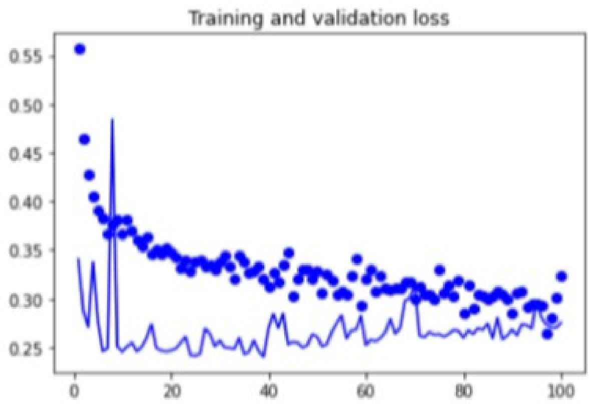
The following function shows the implement of the **VGG16** as a pre-trained network.

```python
def defineCNNModelVGGPretrained():

    #TODO - Exercise 6 - Load the pretrained VGG16 network in a variable called baseModel
    #The top layers will be omitted; The input_shape will be kept to (150, 150, 3)
    baseModel = VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))

    #TODO - Exercise 6 - Visualize the network arhitecture (list of layers)
    baseModel.summary()

    #TODO - Exercise 6 - Freeze the baseModel layers to not to allow training
    for layer in baseModel.layers:
      layer.trainable = False

    #Create the final model and add the layers from the baseModel
    VGG_model = models.Sequential()
    VGG_model.add(baseModel)

    # TODO - Exercise 6 - Add the flatten layer
    VGG_model.add(Flatten())

    # TODO - Exercise 6 - Add the dropout layer
    VGG_model.add(Dropout(0.5))

    # TODO - Exercise 6 - Add a dense layer of size 512
    VGG_model.add(Dense(units = 512, activation = "relu"))

    # TODO - Exercise 6 - Add the output layer
    VGG_model.add(Dense(1, activation='sigmoid'))

    # TODO - Exercise 6 - Compile the model
    VGG_model.compile(optimizer=optimizers.RMSprop(lr=1e-4), loss='binary_crossentropy', metrics=['accuracy'])

    return VGG_model
```

We apply data augmentation; and the **epoch size was 100.** Now we can see the accuracy is **0.89** and the loss is **0.27**

```
Loss:  0.26615312695503235 Accuracy:  0.890999972820282
```

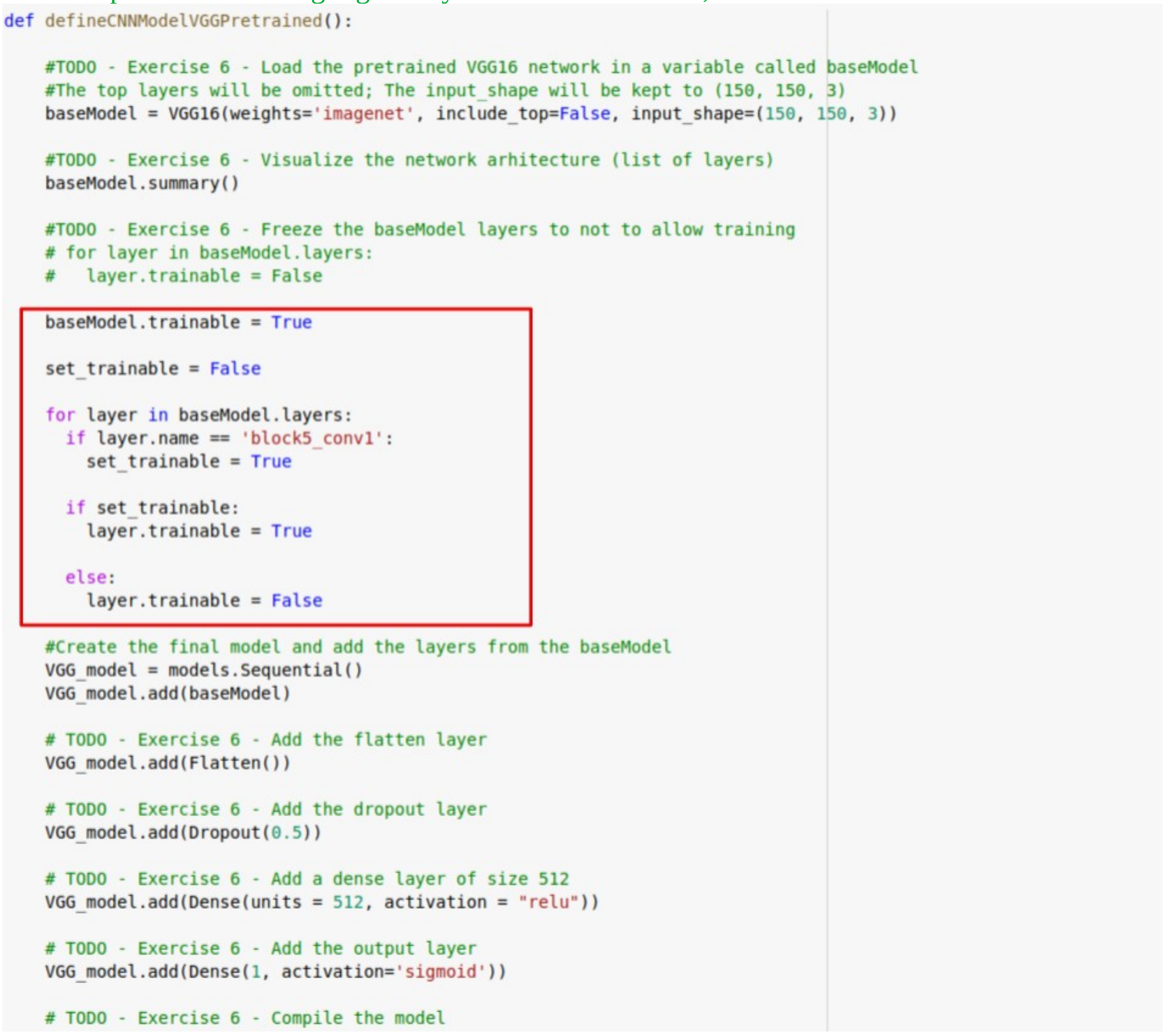




Exercise 7: In this exercise, you will need to classify images of cats and dogs by using transfer learning and fine-tunning. Write a Python script that uses the VGG16 network, previously trained on ImageNet, to extract interesting features from cat and dog images, unfreeze some of the top layers of the frozen model and jointly train both the newly-added classifier layers and the last layers of the base model. This allows us to "fine-tune" the higher-order feature representations in the base model in order to make them more relevant for the specific task.

You'll fine-tune the last three convolutional layers, which mean that all layers up to `block4_pool` should be frozen, and the other layers `block5_conv1`, `block5_conv2`, and `block5_conv3` should be trainable.

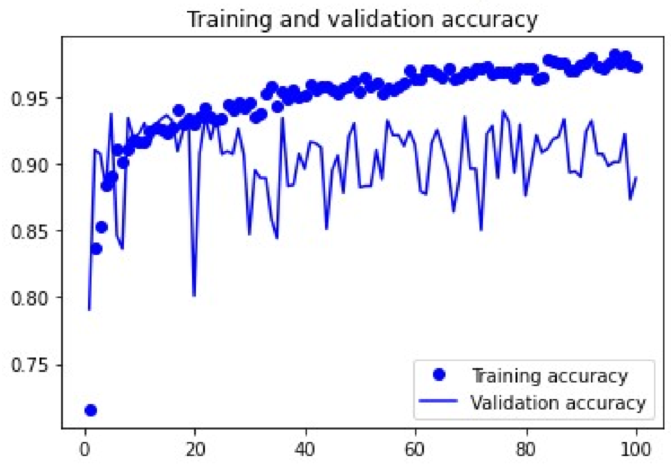
It was implemented like highlighted by the red box as below;

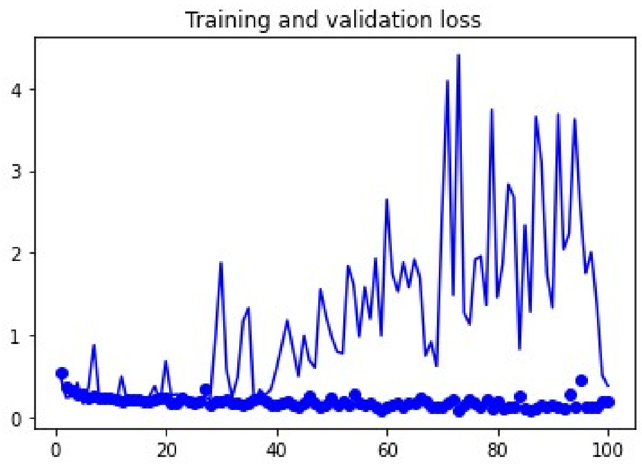


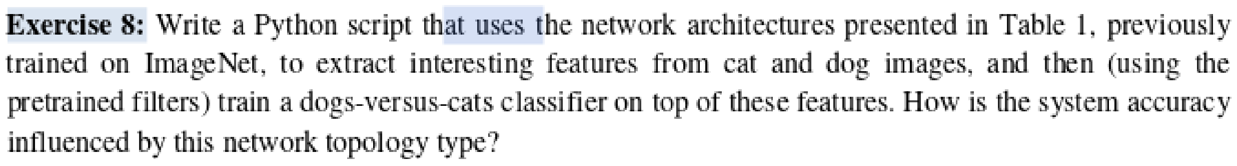
```python
def defineCNNModelVGGPretrained():

    #TODO - Exercise 6 - Load the pretrained VGG16 network in a variable called baseModel
    #The top layers will be omitted; The input_shape will be kept to (150, 150, 3)
    baseModel = VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))

    #TODO - Exercise 6 - Visualize the network arhitecture (list of layers)
    baseModel.summary()

    #TODO - Exercise 6 - Freeze the baseModel layers to not to allow training
    # for layer in baseModel.layers:
    #   layer.trainable = False

    baseModel.trainable = True

    set_trainable = False

    for layer in baseModel.layers:
      if layer.name == 'block5_conv1':
        set_trainable = True

      if set_trainable:
        layer.trainable = True

      else:
        layer.trainable = False

    #Create the final model and add the layers from the baseModel
    VGG_model = models.Sequential()
    VGG_model.add(baseModel)

    # TODO - Exercise 6 - Add the flatten layer
    VGG_model.add(Flatten())

    # TODO - Exercise 6 - Add the dropout layer
    VGG_model.add(Dropout(0.5))

    # TODO - Exercise 6 - Add a dense layer of size 512
    VGG_model.add(Dense(units = 512, activation = "relu"))

    # TODO - Exercise 6 - Add the output layer
    VGG_model.add(Dense(1, activation='sigmoid'))

    # TODO - Exercise 6 - Compile the model
```

The following figures show the training accuracy and the validation accuracy.

```
Loss:  2.449563980102539 Accuracy:  0.878000020980835
```



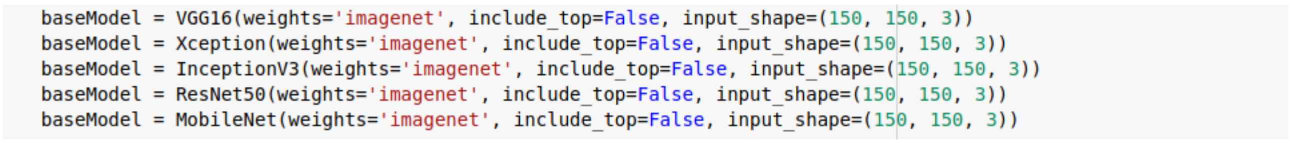



Exercise 8: Write a Python script that uses the network architectures presented in Table 1, previously trained on ImageNet, to extract interesting features from cat and dog images, and then (using the pretrained filters) train a dogs-versus-cats classifier on top of these features. How is the system accuracy influenced by this network topology type?

The following table shows the system accuracy for the different pre-trained model. Everything was tested for 50 epoch size. We have tested few times and took the average.

CNN architecture	VGG16	Xception	Inception	ResNet50	MobileNet
System Accuracy	**0.89**	**0.97**	**0.96**	**0.57**	**0.97**

We selected one at a time to train the model.


```
baseModel = VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))
baseModel = Xception(weights='imagenet', include_top=False, input_shape=(150, 150, 3))
baseModel = InceptionV3(weights='imagenet', include_top=False, input_shape=(150, 150, 3))
baseModel = ResNet50(weights='imagenet', include_top=False, input_shape=(150, 150, 3))
baseModel = MobileNet(weights='imagenet', include_top=False, input_shape=(150, 150, 3))
```

Xception, Inception and MobileNet perform better compared to the VGG16 & ResNet50. It provides an accuracy of 96-97%.