Report Cat and Dog Classification

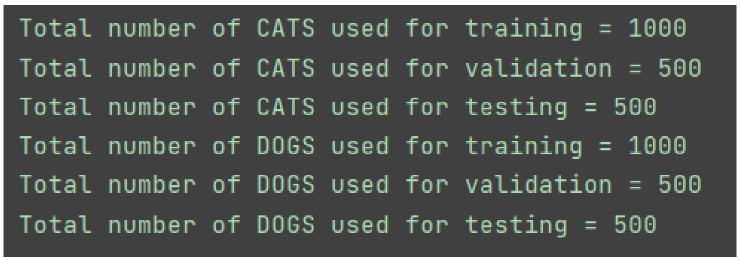
Zemin XU

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 6 cats validation folder (validation_cats_directory) and the following 500 into the cat testing folder (test_cats_directory).

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).

Answer:

The two exercises are successfully done with the following output.



```
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.

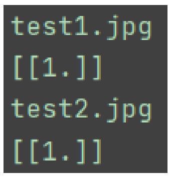
Answer:

The accuracy on the testing dataset is 0.7250, which is close to val_accuracy.

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.

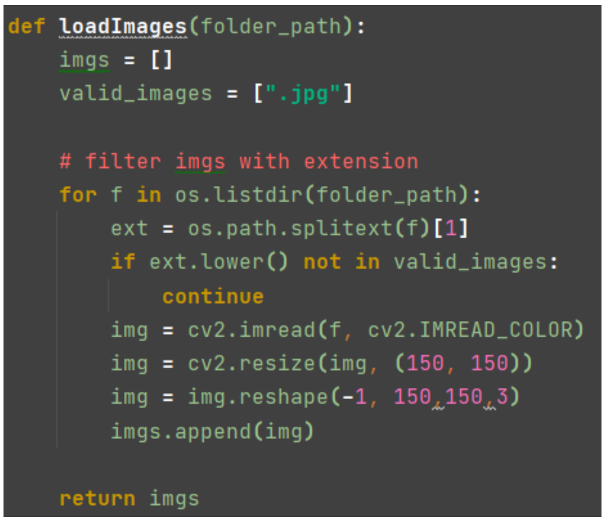
Answer:

The prediction for these two images are all dogs, even though the first one is a cat.



```
test1.jpg
[[1.]]
test2.jpg
[[1.]]
```

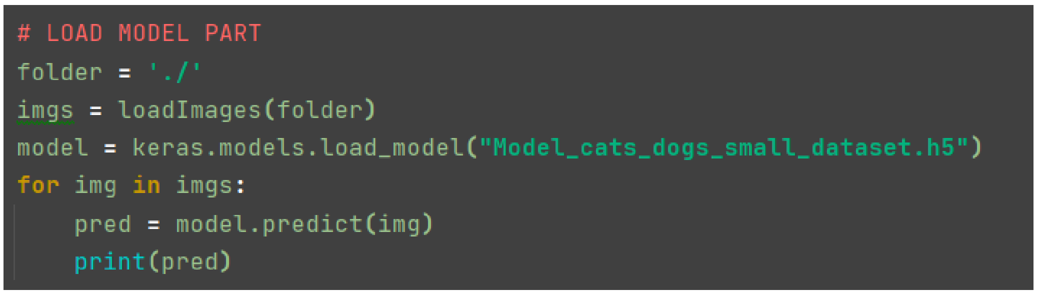
The following is the snippet for loading and predicting, in “exercise4_4.py”.



```python
def loadImages(folder_path):
    imgs = []
    valid_images = [".jpg"]

    # filter imgs with extension
    for f in os.listdir(folder_path):
        ext = os.path.splitext(f)[1]
        if ext.lower() not in valid_images:
            continue
        img = cv2.imread(f, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (150, 150))
        img = img.reshape(-1, 150,150,3)
        imgs.append(img)

    return imgs
```

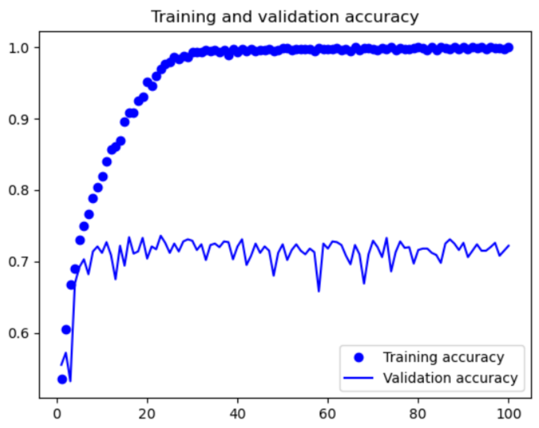


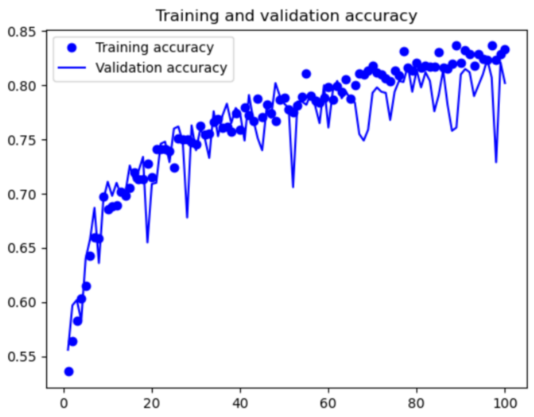
```python
# LOAD MODEL PART
folder = './'
imgs = loadImages(folder)
model = keras.models.load_model("Model_cats_dogs_small_dataset.h5")
for img in imgs:
    pred = model.predict(img)
    print(pred)
```

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.

In order to further improve the system performance, increase the dataset used for training by performing some data augmentation techniques. Let’s train the network for 100 epochs using data augmentation techniques as presented below. How is the system accuracy influenced by the data augmentation techniques? Compare the graphs with the figure saved at Step 6.

Answer:





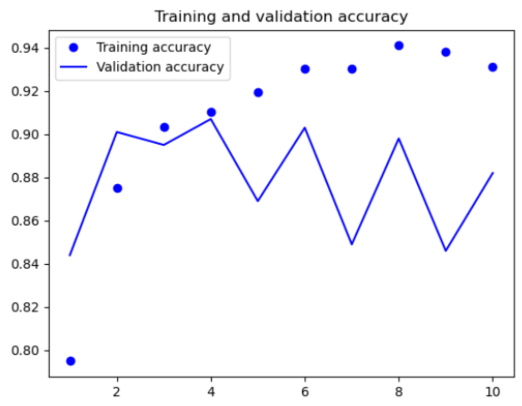
In order to compare the system accuracy, I firstly do the training without dropout layer and data augmentation. We can see the output from the graph on the left that it starts having overfitting problem after 5 epochs and the accuracy is at 0.7250. Then I do the training for 100 epochs with dropout layer and data augmentation, the accuracy keep increasing and the system accuracy stops at 0.8090, which is a great improvement.

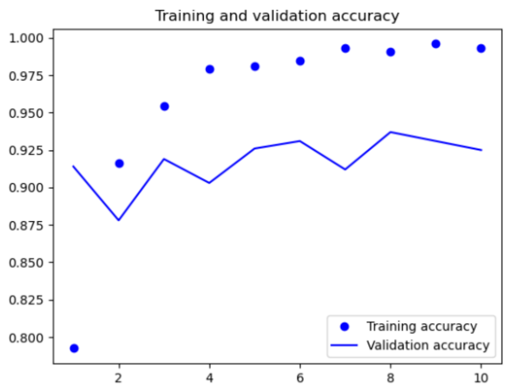
Exercise 6: In this exercise, you will need to classify images of cats and dogs by using transfer learning from a pre-trained network. In the defineCNNModelVGGPretrained method, load the VGG16network, previously trained on ImageNet, to extract interesting features from cat and dog images, and then (using the pretrained filters) train a dogs-versus-cats classifier on top of these features.

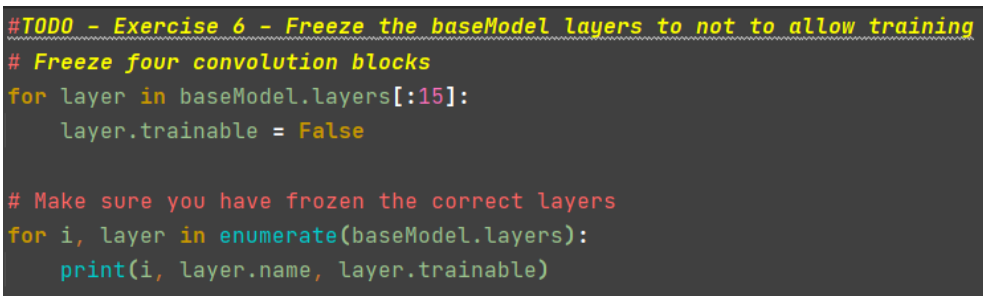
Exercise 7: In this exercise, you will need to classify images of cats and dogs by using transfer learning and fine-tunning. Write a Python script that uses the VGG16 network, previously trained on ImageNet, to extract interesting features from cat and dog images, unfreeze some of the top layers of the 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.

Answer:

With the snippet below, we can make the last layers trainable to better fit this task. The graph below on the left is the result with all layers in VGG16 frozen. Its val_accuracy is about 0.87. The graph on the right is the result with last layers trainable. Its val_accuracy is about 0.92, which is better than the one on left. What's more, the "fine-tune" method makes the loss decreasing more smoothly than before. If the val_accuracy has too much shaking in graph, it is a sign of overfitting.







```
#TODO - Exercise 6 - Freeze the baseModel layers to not to allow training
# Freeze four convolution blocks
for layer in baseModel.layers[:15]:
    layer.trainable = False

# Make sure you have frozen the correct layers
for i, layer in enumerate(baseModel.layers):
    print(i, layer.name, layer.trainable)
```

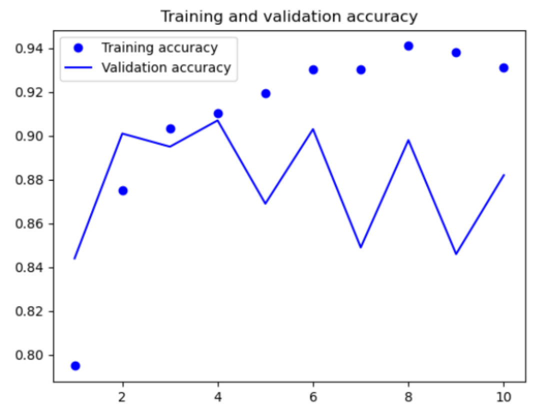
Exercise 8: Write a Python script that uses the network architectures presented in Table 1, previously trained on ImageNet, to extract interesting features from cat and dog images, and then (using the pretrained filters) train a dogs-versus-cats classifier on top of these features. How is the system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

Answer:

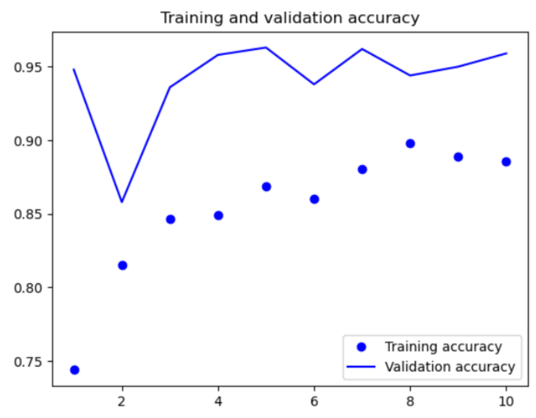
Table 1. System performance evaluation for various numbers of neurons on the hidden layers

CNN architecture	VGG16	Xception	Inception	ResNet50	MobileNet
System accuracy	0.87	0.96	0.95	0.50	0.85

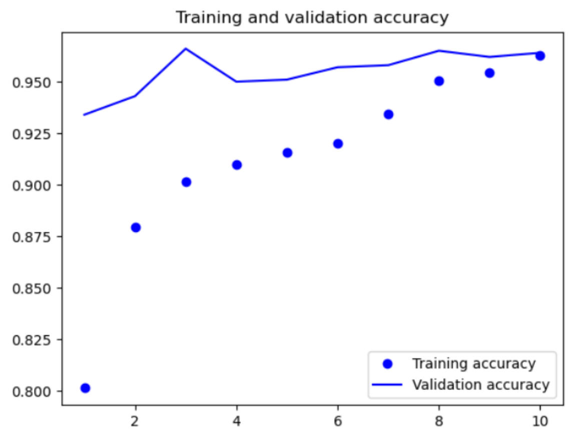
In the experiment, I use the pretrained filters without using "fine-tune" technique. The epochs are 10. The accuracy graphs are showed below. From the table above we can see that Xception and Inception have the best performance. VGG and MobileNet are acceptable as accuracy, however they suffer from overfitting. The worst is ResNet50 with only 0.5 as accuracy.



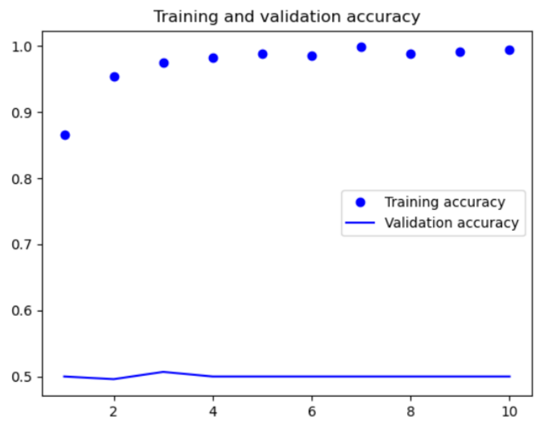
VGG



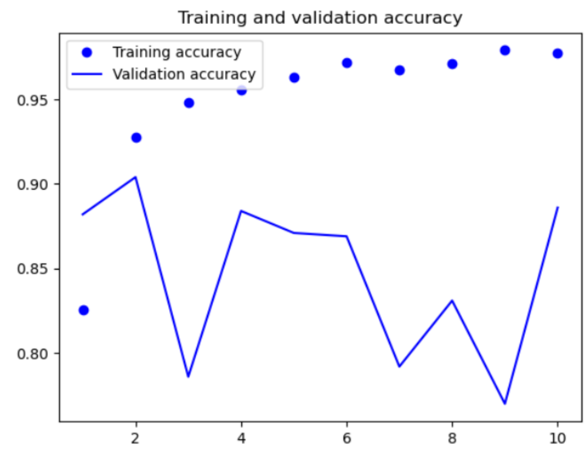
Inception



Xception



ResNet50


Training and validation accuracy
Training accuracy
Validation accuracy
0.95
0.90
0.85
0.80
2
4
6
8
10

MobileNet