Report of Cats/Dogs classification using a reduced training dataset

Xuan ZHOU
Institut Polytechnique de Parise
xuan.zhou@telecom-sudparis.eu

Part I. Application 1.

This study focuses on training the cat/dog classification problem using convolutional neural networks (CNNs). In this experiment, we use a smaller dataset with 4000 images, where cats and dogs have 2000 images each. The system will use 2000 images for training, 1000 for validation and 1000 for testing. The data set for this experiment is relatively small, and to handle the limited amount of data, we can explore two methods: dropout regularization and data augmentation. The aim is to solve the problem of overfitting.

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset. Training deep learning neural network models on more data can result in more skillful models, and the augmentation techniques can create variations of the images that can improve the ability of the fit models to generalize, what they have learned, to new images. Data augmentation can also act as a regularization technique, adding noise to the training data, and encouraging the model to learn the same features, invariant to their position in the input.

Dropout regularization is a technique where randomly selected neurons are ignored during training. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

In this experiment, the dataset contains 25,000 images of cats and dogs, 12,500 images per class. We will create a new dataset containing three subsets: a training set of 1. 000 samples per class, a validation set with 500 samples per class, and a test set with 500 samples per class.

A new dataset with 1000 images for the training set, 500 images for the validation set and 500 images for the test set is created first. Figure 1 is shown the new dataset.

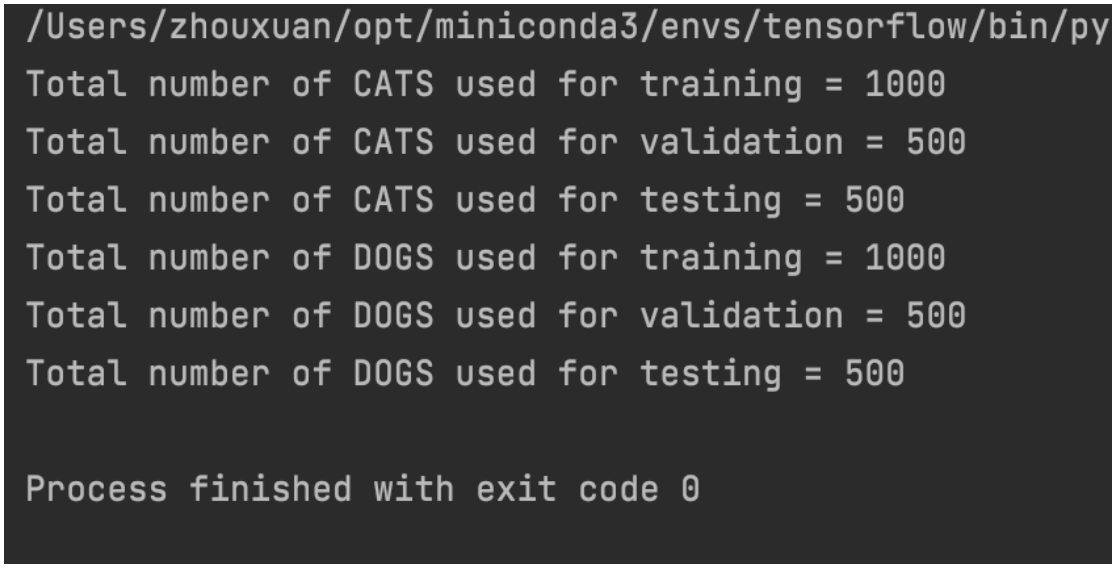

Figure 1. The result of the new dataset.

After rebuilding the dataset, we next preprocess the images and then build a CNN model to train the model. Figure 2 shows the results of model training. Figure 3 shows the numbers generated by the CNN model trained at 100 epochs. To evaluate the accuracy of the model on the dataset, first, a data generator is created for the test dataset to obtain the loss and accuracy of the function. Figure 4 illustrates the loss and accuracy.

Layer (type)	Output Shape	Param #
conv2d_20 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_20 (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_21 (Conv2D)	(None, 73, 73, 64)	8256
max_pooling2d_21 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_22 (Conv2D)	(None, 35, 35, 128)	32896
max_pooling2d_22 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_23 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_23 (MaxPooling2D)	(None, 7, 7, 128)	0
flatten_5 (Flatten)	(None, 6272)	0
dense_10 (Dense)	(None, 512)	3211776
dense_11 (Dense)	(None, 1)	513

Total params: 3,401,921
Trainable params: 3,401,921
Non-trainable params: 0

Figure 2. The result of CNN model

```
Epoch 98/100
100/100 [==============================] - 13s 132ms/step - loss: 0.0082 - accuracy: 0.9975 - val_loss: 2.4316 - val_accuracy: 0.7330
Epoch 99/100
100/100 [==============================] - 13s 131ms/step - loss: 5.5784e-05 - accuracy: 1.0000 - val_loss: 2.3800 - val_accuracy: 0.7370
Epoch 100/100
100/100 [==============================] - 13s 131ms/step - loss: 0.0013 - accuracy: 0.9995 - val_loss: 2.4866 - val_accuracy: 0.7420
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: UserWarning: `Model.evaluate_generator` is deprecated and will be removec
  if sys.path[0] == '':
```

Figure 3. performing the training for 100 epochs.

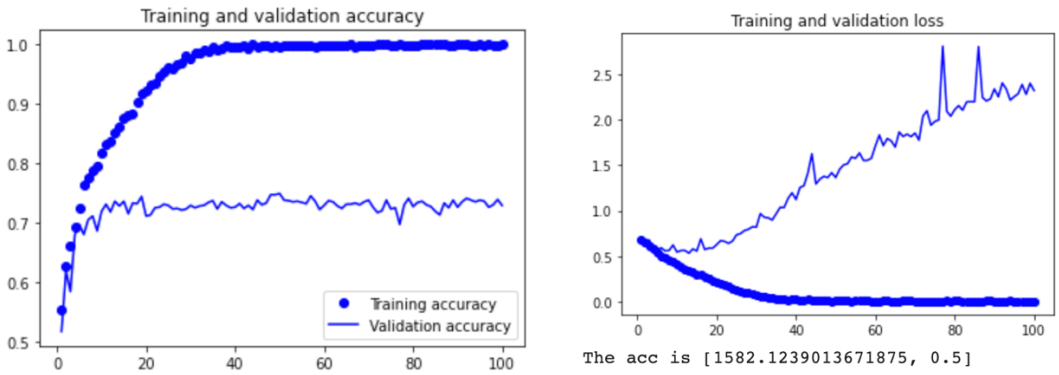

Figure 4. The accuracy and loss

After fitting, we can first save the model as a *.h5 file. We can use our saved model to predict the images of a cat and a dog, as shown in Figure 5. The prediction will be performed simultaneously on the new image using the same batch of images. The results are shown in Figure 6.

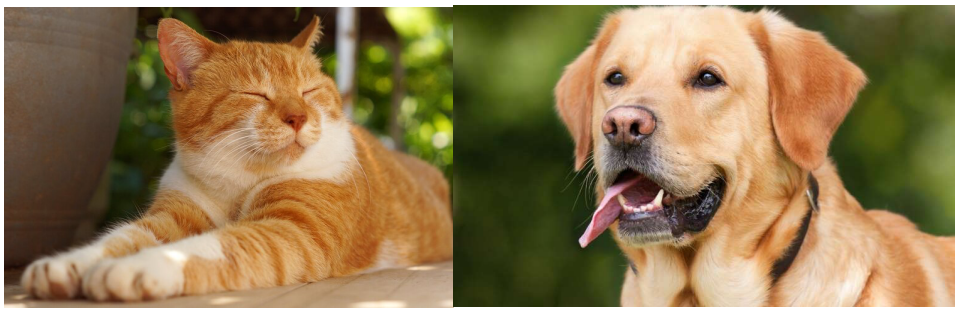

Figure 5. predict the images of a cat and a dog

```
1/1 [==============================] - 0s 152ms/step
[[0.]]
1/1 [==============================] - 0s 10ms/step
[[1.]]

Process finished with exit code 0
```

Figure 6. The result of predict the images of a cat and a dog

In order to increase the system performance by reducing overfitting. we can add a dropout layer into the model of CNN that randomly excludes 50% of neurons. To further improve the system performance, increase the dataset used for training by performing some data augmentation techniques. We can train the network for 100 epochs using data augmentation techniques. Observe the impact of data enhancement techniques on the accuracy of the system as shown in Figure 7.

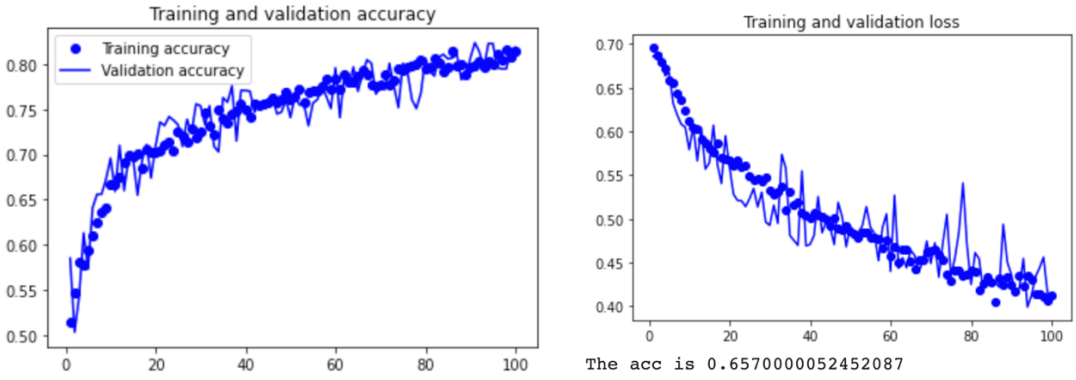

Figure 7. The result of data augmentation for this model

After using data augmentation, the testing accuracy was improved to 65.7%. The use of data augmentation can also avoid the problem of model overfitting. And it can improve the robustness of the model and reduce the sensitivity of the model to images. It can improve the generalization ability of the model.

Part II. Application 2.

Next, we analyze the cat and dog images by performing migration learning using a pre-trained network and load the previously trained VGG16Net. onto ImageNet to extract features from the cat and dog images. The dog and cat classifiers are then trained on the basis of these features. First, the VGG16 structure is loaded, then the architecture of VGG16 is visualized, the volume base of CNN is frozen, and finally, the CNN architecture is created. The results are shown in Figure 8.

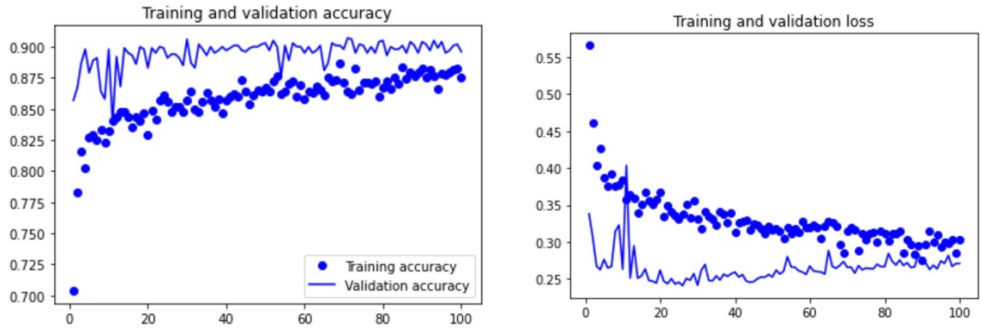

Figure 8. The result of VGG16

After extracting interesting features from the images of cats and dogs using the VGG16 network structure and training the classifier based on it, the results of the VGG16 network topology on the accuracy of the system are shown in Figure 9.

```
Epoch 100/100
100/100 [==============================] - 22s 223ms/step - loss: 0.3039 - accuracy: 0.8750 - val_loss: 0.2712 - val_accuracy: 0.8960
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:19: UserWarning: `Model.evaluate_generator` is deprecated and will be removed
VGG16 testing acc is 0.843999981880188
```

Figure 9. The accuracy of VGG16

Next, we use the network structures Xception, Inception, ResNet50 and MobileNet to

predict the classification of cats and dogs. The effect of different network topologies on the accuracy of the system is determined. The results are shown in Table 1.

CNN architecture	VGG16	Xception	Inception	ResNet50	MobileNet
System accuracy (%)	84%	56%	56%	50%	51%

Table 1. The results of system performance evaluation for various numbers of neurons on the hidden layers

The results of the Xception run are shown in Figure 10, and the accuracy is 56% as shown in Figure 11. The results of the Inception pre-training model are shown in Figure 12, and the Inception accuracy is 56% as shown in Figure 13. The results of the ResNet50 pre-training model are shown in Figure 14, and the accuracy is 50% as shown in Figure 15. the MobileNet pre-training model The results are shown in Figure 16 and the accuracy is 51% as shown in Figure 17.

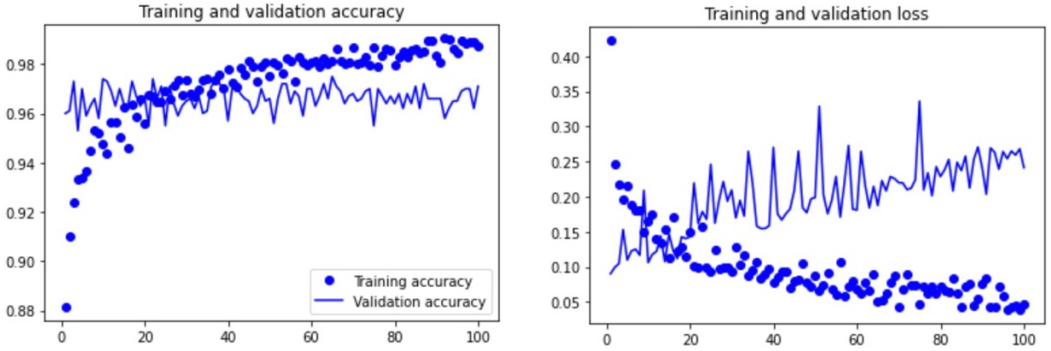

Figure 10. The result of Xception pre-trained models

```
Epoch 100/100
100/100 [==============================] - 22s 224ms/step - loss: 0.0472 - accuracy: 0.9875 - val_loss: 0.2415 - val_accuracy: 0.9710
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:19: UserWarning: `Model.evaluate_generator` is deprecated and will be removed
Xception testing acc is 0.5550000071525574
```

Figure 11. The testing accuracy of Xception pre-trained models

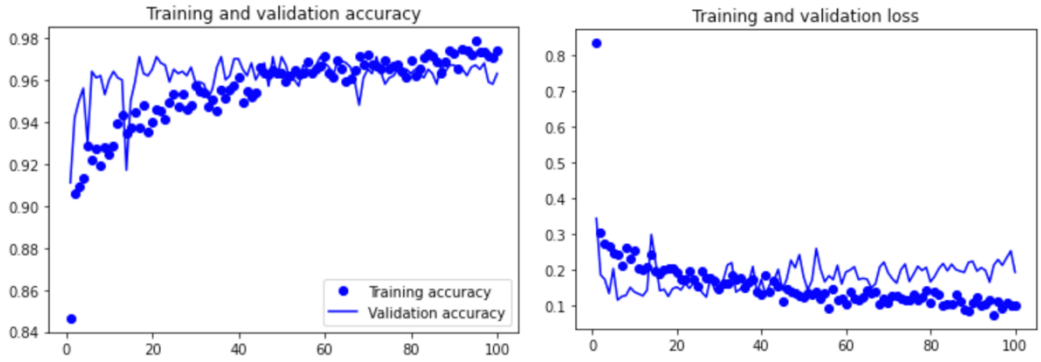

Figure 12. The result of Inception of pre-trained models

```
/usr/local/lib/python3.7/dist-packages/ipykernel_
  del sys.path[0]
Inception testing acc is 0.5550000071525574
```

Figure 13. The Testing accuracy of Inception pre-trained models

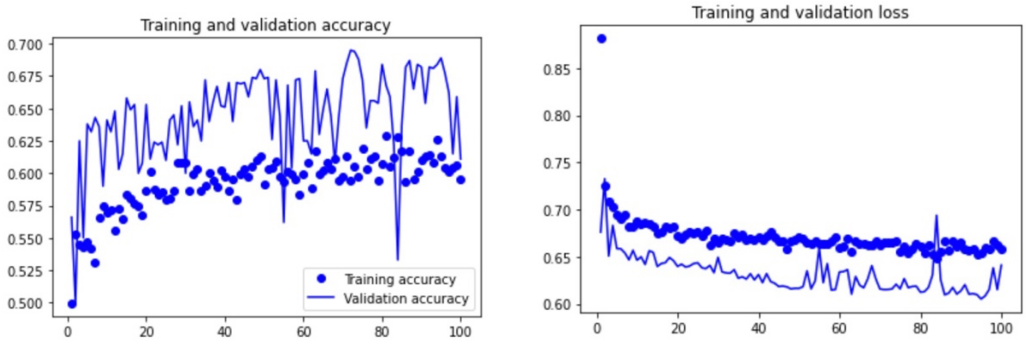

Figure 14. The result of ResNet50 of pre-trained models

```
Epoch 100/100
100/100 [==============================] - 23s 226ms/step - loss: 0.6579 - accuracy: 0.5950 - val_loss: 0.6410 - val_accuracy: 0.6110
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:19: UserWarning: `Model.evaluate_generator` is deprecated and will be removed
ResNet50 testing acc is 0.503000020980835
```

Figure 15. The accuracy of ResNet50 of pre-trained models

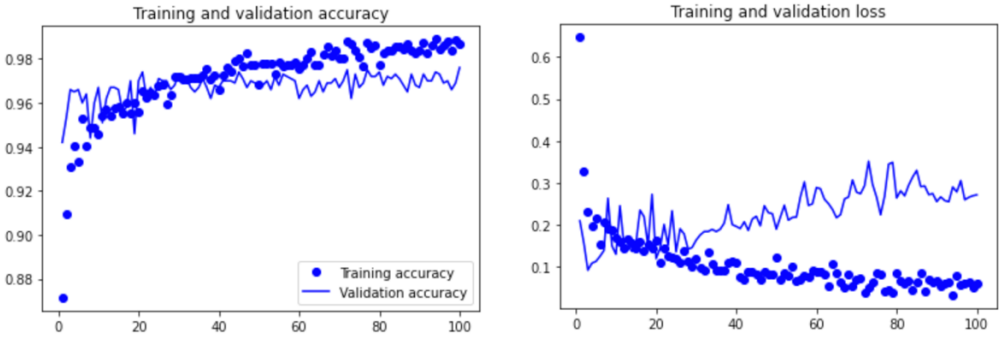

Figure 16. The result of MobileNet of pre-trained models

```
Epoch 100/100
100/100 [==============================] - 22s 216ms/step - loss: 0.0612 - accuracy: 0.9865 - val_loss: 0.2713 - val_accuracy: 0.9760
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:13: UserWarning: `Model.evaluate_generator` is deprecated and will be remo
  del sys.path[0]
MobileNet testing acc is 0.5130000114440918
```

Figure 17. The result of MobileNet of pre-trained models

The results of the five pre-trained models are shown in Table1, and a comparison of the accuracy of the isomorphic pair tests shows that the accuracy of the pre-trained model of VGG16 is the highest, reaching 84%. In this experiment, the VGG16 pre-training model is the best choice among them.

Appendix

Code for Application 1- - - Exercise 1&2

```
from keras import layers
from keras import models
from keras.optimizers import rmsprop_v2
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
from keras.applications.vgg16 import VGG16
import os
import shutil
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation, Dropout

def visualizeTheTrainingPerformances(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)
    pyplot.title('Training and validation accuracy')
    pyplot.plot(epochs, acc, 'bo', label = 'Training accuracy')
    pyplot.plot(epochs, val_acc, 'b', label = 'Validation accuracy')
    pyplot.legend()
    pyplot.figure()
    pyplot.title('Training and validation loss')
    pyplot.plot(epochs, loss, 'bo', label = 'Training loss')
    pyplot.plot(epochs, val_loss, 'b', label = 'Validation loss')
    pyplot.legend
    pyplot.show()
    return

def defineCNNModelFromScratch(input_shape):
    model = models.Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu',
                     input_shape=(input_shape, input_shape, 3)))
    model.add(MaxPooling2D(2, 2))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(2, 2))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D(2, 2))
    model.add(Conv2D(128, (3, 3), activation='relu'))
```

```
        model.add(MaxPooling2D(2, 2))
        model.add(Flatten())
        model.add(Dense(512, activation='relu'))
        model.add(Dense(1, activation='sigmoid'))
        model.summary()
        model.compile(optimizer=rmsprop_v2.RMSprop(learning_rate=0.0001),
loss='binary_crossentropy', metrics=['accuracy'])
        return model

def imagePreprocessing(base_directory):
        train_directory = base_directory + '/train'
        validation_directory = base_directory + '/validation'
        test_directory = base_directory + '/test'
        train_datagen = ImageDataGenerator(rescale=1. / 255)
        validation_datagen = ImageDataGenerator(rescale=1. / 255)
        test_datagen =    ImageDataGenerator()
        test_generator                                                                    =
test_datagen.flow_from_directory(test_directory,target_size=(150,150))
        train_generator          =          train_datagen.flow_from_directory(train_directory,
target_size=(150, 150), batch_size=20,

class_mode='binary')
        va\lidation_generator                                                             =
validation_datagen.flow_from_directory(validation_directory, target_size=(150, 150),

batch_size=20, class_mode='binary')
        for data_batch, labels_batch in train_generator:
            print('Data batch shape in train: ', data_batch.shape)
            print('Labels batch shape in train: ', labels_batch.shape)
            break
        for data_batch, labels_batch in validation_generator:
            print('Data batch shape in validation: ', data_batch.shape)
            print('Labels batch shape in validation: ', labels_batch.shape)
            break
        return train_generator, validation_generator, test_generator

def visualizeTheTrainingPerformances(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)
    pyplot.title('Training and validation accuracy')
    pyplot.plot(epochs, acc, 'bo', label = 'Training accuracy')
```

```
    pyplot.plot(epochs, val_acc, 'b', label = 'Validation accuracy')
    pyplot.legend()
    pyplot.figure()
    pyplot.title('Training and validation loss')
    pyplot.plot(epochs, loss, 'bo', label = 'Training loss')
    pyplot.plot(epochs, val_loss, 'b', label = 'Validation loss')
    pyplot.legend
    pyplot.show()
    return

def main():
        base_directory =
"/content/drive/MyDrive/Kaggle_Cats_And_Dogs_Dataset_Small_1"
        train_generator, validation_generator,test_generator =
imagePreprocessing('/content/drive/MyDrive/Kaggle_Cats_And_Dogs_Dataset_Smal
l')
        input_shape = 150
        model = defineCNNModelFromScratch(input_shape)
        history = model.fit_generator(train_generator, steps_per_epoch=100, epochs=100,
                                      validation_data=validation_generator,
validation_steps=50)
        acc = model.evaluate_generator(test_generator)
        visualizeTheTrainingPerformances(history)
        print("The acc is {}".format(acc(1)))
        model.save('Model_cats_dogs_small_dataset.h5')
        return

if __name__ == '__main__':
        main()
```

Comment:

Code for Exercise 3

```
from keras import layers
from keras import models
from keras.optimizers import rmsprop_v2
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
from keras.applications.vgg16 import VGG16
import numpy as np
import os
import shutil
```

```
import cv2
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation, Dropout
from keras.models import load_model
import tensorflow as tf


def visualizeTheTrainingPerformances(history):
        acc = history.history['accuracy']
        val_acc = history.history['val_accuracy']
        loss = history.history['loss']
        val_loss = history.history['val_loss']
        epochs = range(1, len(acc) + 1)
        pyplot.title('Training and validation accuracy')
        pyplot.plot(epochs, acc, 'bo', label = 'Training accuracy')
        pyplot.plot(epochs, val_acc, 'b', label = 'Validation accuracy')
        pyplot.legend()
        pyplot.figure()
        pyplot.title('Training and validation loss')
        pyplot.plot(epochs, loss, 'bo', label = 'Training loss')
        pyplot.plot(epochs, val_loss, 'b', label = 'Validation loss')
        pyplot.legend
        pyplot.show()
        return

def defineCNNModelFromScratch(input_shape):
        model = models.Sequential()
        model.add(Conv2D(32, (3, 3), activation='relu',
                                input_shape=(input_shape, input_shape, 3)))
        model.add(MaxPooling2D(2, 2))
        model.add(Conv2D(64, (3, 3), activation='relu'))
        model.add(MaxPooling2D(2, 2))
        model.add(Conv2D(128, (3, 3), activation='relu'))
        model.add(MaxPooling2D(2, 2))
        model.add(Conv2D(128, (3, 3), activation='relu'))
        model.add(MaxPooling2D(2, 2))
        model.add(Flatten())
        model.add(Dense(512, activation='relu'))
        model.add(Dense(1, activation='sigmoid'))
        model.summary()
        model.compile(optimizer=rmsprop_v2.RMSprop(learning_rate=0.0001),
loss='binary_crossentropy', metrics=['accuracy'])
        return model

def imagePreprocessing(base_directory):
```

```
    train_directory = base_directory + '/train'
    validation_directory = base_directory + '/validation'
    test_directory = base_directory + '/test'
    train_datagen = ImageDataGenerator(rescale=1. / 255)
    validation_datagen = ImageDataGenerator(rescale=1. / 255)
    test_datagen =  ImageDataGenerator()
    test_generator =
test_datagen.flow_from_directory(test_directory,target_size=(150,150))
    train_generator = train_datagen.flow_from_directory(train_directory,
target_size=(150, 150), batch_size=20,

class_mode='binary')
    validation_generator =
validation_datagen.flow_from_directory(validation_directory, target_size=(150, 150),

batch_size=20, class_mode='binary')
    for data_batch, labels_batch in train_generator:
        print('Data batch shape in train: ', data_batch.shape)
        print('Labels batch shape in train: ', labels_batch.shape)
        break
    for data_batch, labels_batch in validation_generator:
        print('Data batch shape in validation: ', data_batch.shape)
        print('Labels batch shape in validation: ', labels_batch.shape)
        break
    return train_generator, validation_generator, test_generator

def visualizeTheTrainingPerformances(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)
    pyplot.title('Training and validation accuracy')
    pyplot.plot(epochs, acc, 'bo', label = 'Training accuracy')
    pyplot.plot(epochs, val_acc, 'b', label = 'Validation accuracy')
    pyplot.legend()
    pyplot.figure()
    pyplot.title('Training and validation loss')
    pyplot.plot(epochs, loss, 'bo', label = 'Training loss')
    pyplot.plot(epochs, val_loss, 'b', label = 'Validation loss')
    pyplot.legend
    pyplot.show()
    return
# Exercise 3
```

```
def loadImages(path):
    image = cv2.imread(path)
    resized_image = cv2.resize(image,(150,150))
    return resized_image

def main():
    input_shape = 150
    model = defineCNNModelFromScratch(input_shape)
    model = load_model('/Users/zhouxuan/Desktop/VAR_AI/Lab4_CatsAndDogs_Classification/Model_cats_dogs_small_dataset.h5')
    image_1 = loadImages('/Users/zhouxuan/Desktop/VAR_AI/Lab4_CatsAndDogs_Classification/test1.jpg')
    image_2 = loadImages('/Users/zhouxuan/Desktop/VAR_AI/Lab4_CatsAndDogs_Classification/test2.jpg')
    #print(image_2.shape)
    images =[]
    images.append(image_1)
    images.append(image_2)
    tensor_image = tf.convert_to_tensor(images)
    for element in tensor_image:
        element = tf.expand_dims(element, 0)
        label = model.predict(element)
        print(label)
    return

if __name__ == '__main__':
    main()
```

Comment: we loaded the model to predict the new images of a cat and a dog.

Code for Exercise 5

```
def defineCNNModelVGGPretrained():
    baseModel = InceptionV3(weights='imagenet', include_top=False,input_shape=(150, 150, 3))
    baseModel.summary()
    for layer in baseModel.layers:
       layer.trainable = False
    VGG_model = models.Sequential()
    VGG_model.add(baseModel)
```

```
        VGG_model.add(Flatten())
        VGG_model.add(Dropout(0.5))
        VGG_model.add(Dense(512,activation = 'relu'))
        VGG_model.add(Dense(1, activation='sigmoid'))
        VGG_model.compile(optimizer=rmsprop_v2.RMSprop(learning_rate=0.0001),
loss='binary_crossentropy', metrics=['accuracy'])
        return VGG_model
#    Exercise 5
def imagePreprocessing(base_directory):
        train_directory = base_directory + '/train'
        validation_directory = base_directory + '/validation'
        test_directory = base_directory + '/test'
        train_datagen                                                                    =
ImageDataGenerator(rescale=1./255,rotation_range=40,width_shift_range=0.2,height
_shift_range=0.2,shear_range=0.2,zoom_range=0.2,horizontal_flip=True,fill_mode='
nearest')
        validation_datagen = ImageDataGenerator(rescale=1. / 255)
        test_datagen =    ImageDataGenerator()
        test_generator=test_datagen.flow_from_directory(test_directory,target_size=(150
,150),class_mode='binary')
        train_generator          =          train_datagen.flow_from_directory(train_directory,
target_size=(150, 150), batch_size=20,

class_mode='binary')
        validation_generator                                                             =
validation_datagen.flow_from_directory(validation_directory, target_size=(150, 150),

batch_size=20, class_mode='binary')
for data_batch, labels_batch in train_generator:
      print('Data batch shape in train: ', data_batch.shape)
      print('Labels batch shape in train: ', labels_batch.shape)
      break
for data_batch, labels_batch in validation_generator:
      print('Data batch shape in validation: ', data_batch.shape)
      print('Labels batch shape in validation: ', labels_batch.shape)
      break
return train_generator, validation_generator, test_generator

def main():
        base_directory                                                                   =
"/content/drive/MyDrive/Kaggle_Cats_And_Dogs_Dataset_Small_1"
        train_generator,                validation_generator,test_generator              =
imagePreprocessing('/content/drive/MyDrive/Kaggle_Cats_And_Dogs_Dataset_Smal
l')
```

```
    model =defineCNNModelVGGPretrained()
    history = model.fit_generator(train_generator, steps_per_epoch=100, epochs=100,
                                  validation_data=validation_generator,
validation_steps=50)
    acc = model.evaluate_generator(test_generator)
    print("Inception testing acc is {}".format(acc[1]))
    visualizeTheTrainingPerformances(history)
    model.save('Model_cats_dogs_small_dataset.h5')
    return
```

Comment: Define VGG16 pre-training model, other parts are the same as application, the code changes have been marked in pink.

Code for Exercise 6

```
def defineCNNModelVGGPretrained():
    baseModel=I MobileNet (weights='imagenet',include_top=False,
input_shape=(150, 150, 3))
    baseModel.summary()
    for layer in baseModel.layers:
       layer.trainable = False
    VGG_model = models.Sequential()
    VGG_model.add(baseModel)
    VGG_model.add(Flatten())
    VGG_model.add(Dropout(0.5))
    VGG_model.add(Dense(512,activation = 'relu'))
    VGG_model.add(Dense(1, activation='sigmoid'))
    VGG_model.compile(optimizer=rmsprop_v2.RMSprop(learning_rate=0.0001),
loss='binary_crossentropy', metrics=['accuracy'])
    return VGG_model
```

Comment: Other parts of the code are not changed, and the code changes have been marked in purple. Observe the change in the accuracy of the model by changing the preprocessing model.