Artificial intelligence for VR/AR

Report for Lab4 Session

Xuefeng Wei
Institut Polytechnique de Paris
xuefeng.wei@ip-paris.fr

Part I. Building CNN to classify Cats and Dogs

The dataset used in this experiment is the cat and dog dataset, which contains images of cats and dogs in various poses and scenes, because the original dataset is large, in this experiment, the authors only used a part of the data to train the neural network and test, the dataset used contains a total of 4000 images, 2000 images in each category, including 1000 images in the training set, 500 images in the test set, and 500 images in the validation set. The results of the data set classification can be seen in Figure 1.

```
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

Process finished with exit code 0
```

Figure1. The results shown in PyCharm console for separating of data

This dataset was then used by the authors for CNN training, and the structure of this CNN is shown in Figure 2, The hyperparameter is set to epochs=100, the learning rate is 0.0001, and the loss is binary_crossentropy, the results of the run can be seen in Figure 3.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_3 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dense (Dense)	(None, 512)	3211776
dense_1 (Dense)	(None, 1)	513

Figure2. The architecture of the CNN used in exercise1

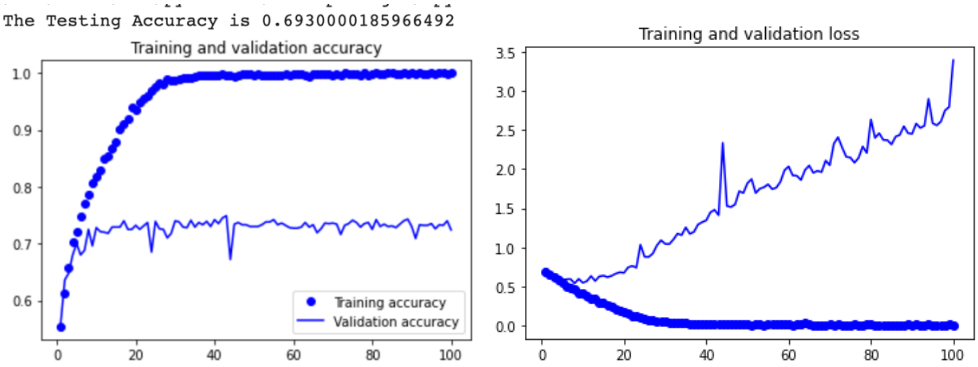


Figure3. Results shown in Google Colab console for exercise1&2

From Table III, the training set converged well, while the validation set started to overfit after 15 epochs, and the final accuracy bracelet of the validation set was around 75%, while the performance of the accuracy on the test set was 69%. The trained model was then saved for subsequent testing, in which the authors chose two images, one of a cat and one of a dog, as shown in Figure 4.

Figure4. Testing sample for the save model

When the weight file is loaded, we will fold two images read as one batch and then test, the results of the test are shown in Figure 5.

```
1/1 [==============================] - 0s 122ms/step
The 1st prediction label is Cat
1/1 [==============================] - 0s 20ms/step
The 2st prediction label is Dog

Process finished with exit code 0
```

Figure5. Results shown in PyCharm console for exercise3

In order to avoid overfitting, dropout layer and data augmentation operations are introduced into the CNN model, with the proportion of dropout being 50% and the specific operations of data augmentation being: rotation_range=40,width_shift_range=0.2,height_shift_range=0.2,shear_range=0.2,zoom_range=0.2,fill_mode='nearest',horizontal_flip=True. The experimental results are shown in Figure 6. In terms of test accuracy, after the data enhancement and dropout operations, the accuracy has been improved, which indicates that these two operations can make our model more robust and have stronger generalization ability. From the learning curve, the phenomenon of overfitting is moderated to a greater extent, and the overall loss value is decreasing, but there are large fluctuations, so we can consider using a smaller learning rate to get a smoother curve.

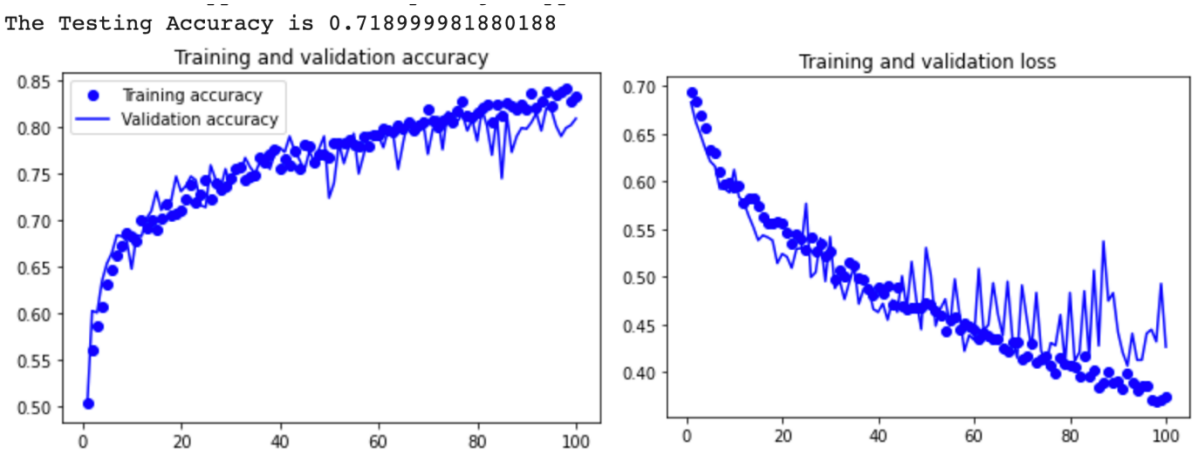


Figure6. Learning curve for exercsie4

Part II. Transfer Learning

In this section, migration learning is used by the authors on this dataset by loading imagenet pre-training weights and a more advanced network structure in order to expect the model to perform better on this dataset, First, the model of VGG16 is loaded with the same settings of hyperparameters as appliicaton1, and the model behaves as shown in Figure 7.

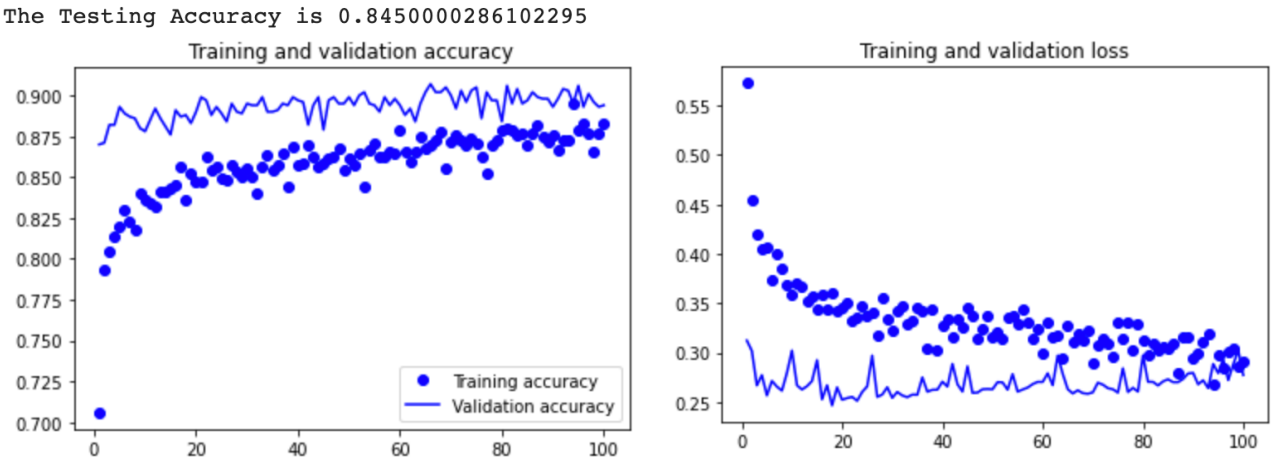
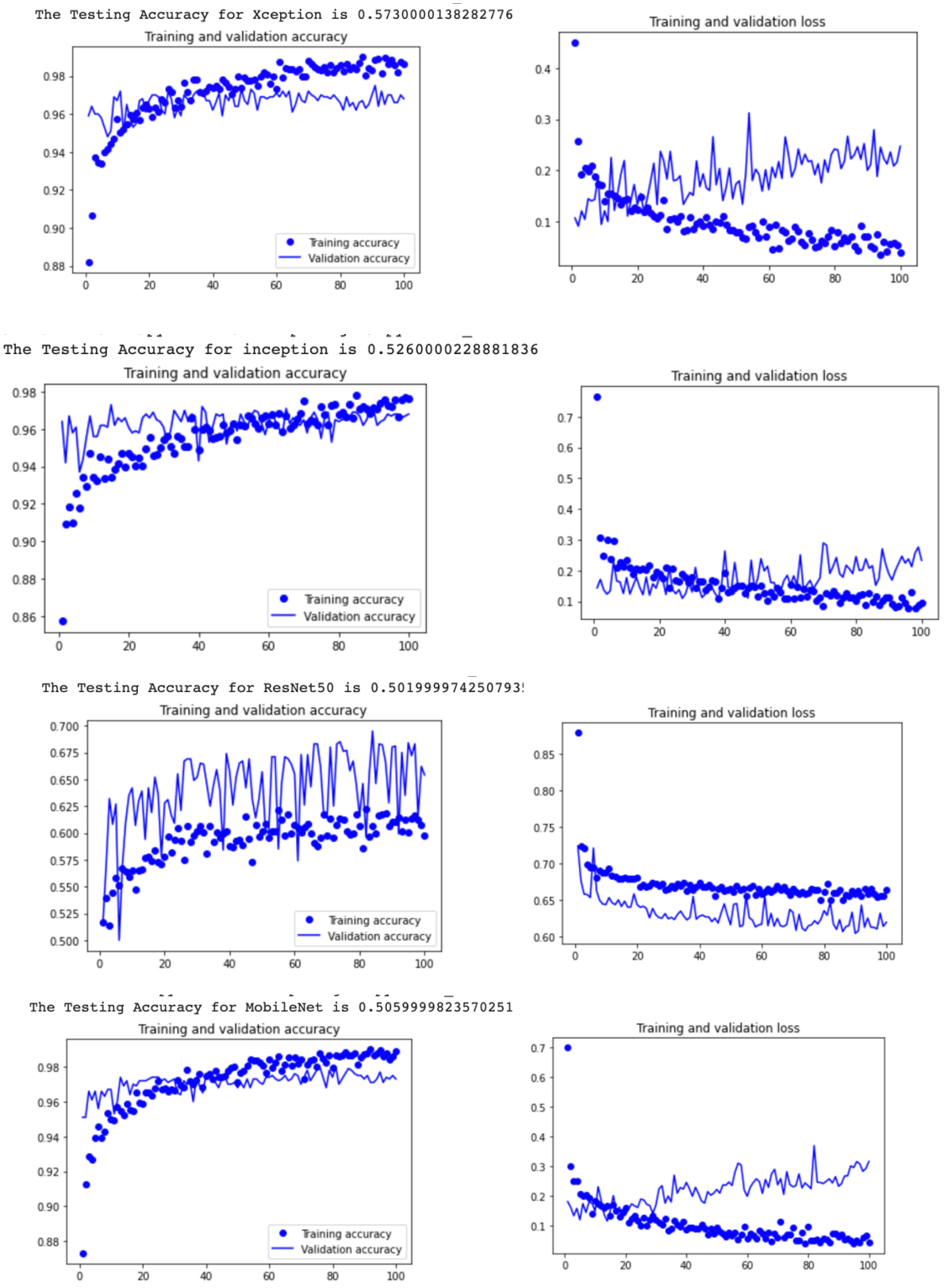


Figure6. Learning curve for exercsie5

As seen in Figure 6, by loading the vgg model and pre-training the weights, our model achieves a prediction accuracy of 84.5% on the test dataset, which is better than any previous results, and the learning curve does not fluctuate much. Finally, not only the model of VGG16 was used for testing, but Xception, Inception, ResNet50 and MobileNet were also called for testing and all loaded with the pre-training weights of Imagenet, and the results are shown in Table 1 and Figure7.

CNN architecture	VGG16	Xception	Inception	ResNet50	MobileNet
System Accuracy (%)	84.50	57.30	52.60	50.19	50.59

Table2. System performance evaluation for various CNN architecture



From the results in Table 2, the VGG16 model has better performance on this dataset.

Appendix

Code for Exercise 1&2&3

```
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 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()
    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')
    test_generator = test_datagen.flow_from_directory(test_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 defineCNNModelFromScratch():

    model = models.Sequential()
    model.add(Conv2D(32, (3, 3), input_shape=(150, 150, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(64, (3, 3),activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(128, (3, 3),activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.summary()
    model.compile(loss='binary_crossentropy',
optimizer=rmsprop_v2.RMSProp(learning_rate=0.0001),metrics=['accuracy'])

    return model

def main():

    #original_directory = "./Kaggle_Cats_And_Dogs_Dataset"
    #base_directory = "./Kaggle_Cats_And_Dogs_Dataset_Small_1"

    #prepareDatabase(original_directory, base_directory)
    train_generator, validation_generator,test_generator =
imagePreprocessing('/content/drive/MyDrive/Kaggle_Cats_And_Dogs_Dataset_Small_1')
    model = defineCNNModelFromScratch()

    history = model.fit_generator(train_generator, steps_per_epoch=100, epochs=100,
validation_data=validation_generator, validation_steps=50) #change the number of epochs here
    scores = model.evaluate_generator(test_generator) #get the accuracy value by using
model.evaluate_generator

    model.save('Model_cats_dogs_small_dataset.h5') #Save the trained model here
    print("The Testing Accuracy is {}".format(scores[1]))
    visualizeTheTrainingPerformances(history). #show the learning curve here

    return

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

Code for Exercise 4

```
def defineCNNModelFromScratch():

    model = models.Sequential()
    model.add(Conv2D(32, (3, 3), input_shape=(150, 150, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(64, (3, 3),activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(128, (3, 3),activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.5)) #adding the dropout layer here
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.summary()
    model.compile(loss='binary_crossentropy',
optimizer=rmsprop_v2.RMSProp(learning_rate=0.0001),metrics=['accuracy'])

    return model

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,
        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').    #adding the data augmentation here

    validation_datagen = ImageDataGenerator(rescale=1./255)
    test_datagen = ImageDataGenerator()
    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')
    test_generator = test_datagen.flow_from_directory(test_directory,target_size=
```

```
(150,150),class_mode='binary')
    #TODO - Application 1 - Step 2 - Analyze the output of the train and validation generators
    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
```

Code for Exercise 5

```
def defineCNNModelVGGPretrained():
    baseModel = VGG16(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

def main():

    #original_directory = "./Kaggle_Cats_And_Dogs_Dataset"
    #base_directory = "./Kaggle_Cats_And_Dogs_Dataset_Small_1"
    #prepareDatabase(original_directory, base_directory)

    train_generator, validation_generator,test_generator =
imagePreprocessing('/content/drive/MyDrive/Kaggle_Cats_And_Dogs_Dataset_Small_1')


    model = defineCNNModelVGGPretrained() #change the model to VGG16 here
    history = model.fit_generator(train_generator, steps_per_epoch=100, epochs=100,
validation_data=validation_generator, validation_steps=50)
```

```
    scores = model.evaluate_generator(test_generator)
    print("The Testing Accuracy is {}".format(scores[1]))
    visualizeTheTrainingPerformances(history)

    return
```

Code for Exercise 6

```
from keras.applications.vgg16 import VGG16
from keras.applications.xception import Xception
from keras.applications.inception_v3 import InceptionV3
from keras.applications.resnet import ResNet50
from keras.applications.mobilenet import MobileNet

def defineCNNModelVGGPretrained():

    #TODO - Application 2 - Step 1 - 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 = MobileNet(weights='imagenet',include_top=False,input_shape=(150,150,3))
   #change the baseModel in MobileNet, VGG16, ResNet50, Xception and Inceptin here
    baseModel.summary()
    for layer in baseModel.layers:
        layer.trainable = False
    VGG_model = models.Sequential()
    VGG_model.add(baseModel)                    #Uncomment this
    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
```
