Name: Lingxiao Zhang

Student ID: 20475043

Lab Report

Introduction

The goal of this project is to apply and implement image recognition models, CNN, to classify different classes of flowers based on three datasets, which are training data, validation data and the test data.

Lab Procedure

This project can be separated into four steps, which are read and scale the input data, apply training models, compare performance, and make predictions and write the result to a csv file.

Load the data

First, I used the Pandas read_table() function to create a table to classify the content for each txt file. In general, I created three tables in total and one for each file. For tables of training data and validation data, I assigned two columns of attributes, which are “path” and “labels” since the content of both files contains the path of all the pictures and the actual class/type they belong to. For the test data, it does not have the “label” attribute and our goal is to predict its “label” attributes after training the model and testing the performance on the performance on the validation data. The code is shown below:

```
data_path = '/Users/lingxiaozhang/Documents/6000b/project2/'

train_path = pd.read_table("train.txt", sep=' ', header=None)
train_path.columns = ['path', 'label']

val_path = pd.read_table("val.txt", sep=' ', header=None)
val_path.columns = ['path', 'label']

test_path = pd.read_table("test.txt", sep=' ', header=None)
test_path.columns = ['path']

length_train = len(train_path)
length_val = len(val_path)
length_test = len(test_path)

train_X = np.zeros((length_train, height, width, 3))
train_Y = train_path['label']
#train_Y = train_Y.values.reshape((-1, 1))
val_X = np.zeros((length_val, height, width, 3))
val_Y = val_path['label']
#val_Y = val_Y.values.reshape((-1, 1))
test_X = np.zeros((length_test, height, width, 3))


print("begin...load data")
for i in range(length_train):
    image = imread(train_path.iloc[i, :].path)
    image = imresize(image, (height, width))
    train_X[i, :, :, :] = image


for j in range(length_val):
    image = imread(val_path.iloc[j, :].path)
    image = imresize(image, (height, width))
    val_X[j, :, :, :] = image


for m in range(length_test):
    image = imread(test_path.iloc[m, :].path)
    image = imresize(image, (height, width))
    test_X[m, :, :, :] = image

print(train_X.shape[1:])
```

Train the model

For me, I have made and tried several models for this project, including self-made and models in the Keras package. For example, I have tried several practices of created my own CNN designs with 3-5 layers of combinations of convolutional and max-pooling accompanied with the activation function and dropout function. In the final layer, we need to use Flatten() and Dense() function to reduce the dimension and to specify the number of classes for the output. For the models in the Keras package, I have used Vgg16, Resnet50 and Xception. Basically,

there are three steps to train a model using Keras. First, we need build the structure/template of the mode. For the model designed by myself, I need to write the self-designed code of the structure of my cnn, which I have mentioned above. For the model I used from Keras, only a single line needed, which we just need to call that model's name from Keras. Second, we need to choose the optimizer for the model. For this part, I've tried three optimizers, which are Rmsprop, Adam and Adadelta. After choosing the optimizer, we need to call the function compile() to "compile" our model with the optimizer chosen. Finally, we need to feed the input data to our model. Here, in some cases we need to normalize the input data, for me, I normalized the input data in the model designed by myself and Xception. However, for the Resnet50 and Vgg16, normalization cause poor performance on the accuracy so I didn't normalize the input data for these two models.

```
#base = VGG16(weights='imagenet', include_top=False)
#base = VGG16(weights=None, include_top=False, input_tensor=Input((224, 224, 3)))
#base_model = ResNet50(weights='imagenet', include_top=False, input_tensor=Input((224, 224, 3)))
base = Xception(include_top=False, weights='imagenet', input_tensor=Input((299, 299, 3)))
output = base.output
output = Flatten()(output)
#output = Dense(512, activation='relu')(output)
output = Dropout(0.6)(output)
output = Dense(5, activation='softmax')(output)

# Assign connections
model = Model(input=base.input, outputs=output)

# Define optimizer
optimizer = keras.optimizers.rmsprop(0.0001, decay=1e-6)
#optimizer = keras.optimizers.rmsprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
#optimizer = keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
#optimizer=keras.optimizers.Adadelta()

# Train the model using RMSprop
model.compile(loss='categorical_crossentropy', #sparse_categorical_crossentropy, categorical_crossentropy
              optimizer=optimizer,
              metrics=['accuracy'])


# Normalize
train_X = train_X.astype('float32')
train_X /= 255
val_X = val_X.astype('float32')
val_X /= 255
test_X = test_X.astype('float32')
test_X /= 255

# Feed inputs and labels
print("Start training...")
model.fit(train_X, train_Y,
          batch_size=_batch_size,
          epochs=epochs,
          validation_data=(val_X, val_Y)
          )
```

Compare the performance and make predictions

After comparing the performance on the self-made models and keras's models, it turns out that Xception has the has the highest performance on the validation data after 3 epochs, which the accuracy is around 92.5%. The ones that I made have a greatest performance around 65%. So I used and saved the model generated from Xception to make predictions on the test data.

```
In [5]: %run cnn1.py

Using TensorFlow backend.

begin...load data
(299, 299, 3)

/Users/lingxiaozhang/Documents/6000b/project2/project2/data/cnn1.py:139: UserWarning: Update your `Model` call to the
Keras 2 API: `Model(outputs=Tensor("de..., inputs=Tensor("in...)`
  model = Model(input=base_model.input, outputs=x)

(2569, 299, 299, 3)
(2569, 5)
(550, 299, 299, 3)
(550, 5)
Train on 2569 samples, validate on 550 samples
Epoch 1/3
2569/2569 [==============================] - 10034s - loss: 0.6176 - acc: 0.8163 - val_loss: 0.5137 - val_acc: 0.8345
Epoch 2/3
2569/2569 [==============================] - 11955s - loss: 0.0571 - acc: 0.9840 - val_loss: 0.2668 - val_acc: 0.9145
Epoch 3/3
2569/2569 [==============================] - 9734s - loss: 0.0141 - acc: 0.9957 - val_loss: 0.2158 - val_acc: 0.9255
550/550 [==============================] - 441s
Test loss: 0.215828706121
Test accuracy: 0.925454545455
[[  0.00000000e+00   0.00000000e+00   2.94534639e-31   0.00000000e+00
    1.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   1.76214685e-36   0.00000000e+00
    1.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   4.76486547e-27   0.00000000e+00
    1.00000000e+00]
 ...,
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
    1.00000000e+00]
 [  1.16993693e-34   0.00000000e+00   7.89142565e-15   0.00000000e+00
    1.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   3.64738485e-22   0.00000000e+00
    1.00000000e+00]]
```

Conclusion

To sum up, I gained some experience on understanding the operations of CNN and training a CNN model and then test on a test set in this project. On the other hand, I gained some experience on using the Kera package.