Big Data Final Project

Qiyun Zhang
Aug 17, 2019

Contents

1 Introduction

1.1 Background

New data is generated in our daily life. Statistically, there are 2.5 quintillion bytes of data created each day (Marr). This data is important not only because it reflects what is happening now but also because it contributes a lot to the future reference. In this case, an efficient way of dealing with data is required, including collecting and transferring data, data storage, and making predictions based on the data we have already generated.

1.2 Problem Restatement

The goal of this project is to predict the numbers in the corresponding images uploaded by users of the program. This is crucial because the numbers in those images are in handwritten form, and is more difficult for computers to recognize compared with printed numbers. Also, to make sure of the efficiency of transferring and storing data, the virtual container Docker is included in this project.

Specifically, Convolutional Neural Network(CNN), a trained model of which is deployed into Docker, is used to predict the handwriting numbers. The results of predictions are given to two locations at the same time. First, by RESTful API, results can be transferred to the website that this project established earlier, users can see the results predicted on the website directly. Second,

the predictions of the results are recorded in the phpMyAdmin databases, including both the numbers and the time used to make those predictions correspondingly.

2 CNN Model

Training a CNN Model aims to prepare for the future prediction process. How well the model is trained decides how accurate the predicted results are. The way to train this model is to implement the MNIST code, and users are required to run it before other operations. The trained model will be stored in a file named “model”.

MNIST reshapes the images uploaded by users, and transforms the images into standardized images centered in the 28*28 field in conv1: (in MNIST.py)

```
with tf.variable_scope("conv1"):
    # 随机初始化权重
    w_conv1 = weight_variavles([5, 5, 1, 32])
    b_conv1 = bias_variavles([32])

    # 对x进行形状的改变[None, 784] ----- [None,28,28,1]
    x_reshape = tf.reshape(x, [-1, 28, 28, 1])  # 不能填None,不知道就填-1

    # [None,28, 28, 1] -------- [None, 28, 28, 32]
    x_relu1 = tf.nn.relu(tf.nn.conv2d(x_reshape, w_conv1, strides=[1, 1, 1, 1], padding="SAME")
        + b_conv1)

    # 池化 2*2，步长为2, 【None, 28,28, 32]--------[None,14, 14, 32]
    x_pool1 = tf.nn.max_pool(x_relu1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding="SAME")
```

The target numbers in those images are identified by grey levels. The training set contains 60,000 examples, and the test set contains 10,000 examples (Yann, Corinna, Christopher). Once the

MNIST model is available, the training times of CNN Model is set. In this project, it is 3000.

The accuracy of the process is returned per 100 steps.

```
with tf.Session() as sess:
    sess.run(init_op)
    for i in range(3000):
        mnist_x, mnist_y = mnist.train.next_batch(50)
        if i % 100 == 0:
            # 评估模型准确度，此阶段不使用Dropout
            train_accuracy = accuray.eval(feed_dict={x: mnist_x, y_true: mnist_y, keep_prob:
                1.0})
            print("step %d, training accuracy %g" % (i, train_accuracy))

        # 训练模型，此阶段使用50%的Dropout
        train_op.run(feed_dict={x: mnist_x, y_true: mnist_y, keep_prob: 0.5})
        # 将模型保存在你自己想保存的位置
    saver.save(sess, "D:/Candy/model/fc_model.ckpt")
```

3 The User Interface

Two parts are included in the user interface. First, Flask is implemented to make sure that users can upload images, which are automatically transported to the trained model of CNN. Second, the trained model of CNN gives feedbacks, which are the results of the predictions. The results are returned to the website the users can view.

For the upload part, the code needed is in the page resource of upload.html:

```
<html>

    <head>
        <title>Upload   new File</title>
        <h1>Upload  your    testing image   please</h1>
        <form   action = '/pic_identity' method="POST"  enctype="multipart/form-data">
            <input  type="file" name="file">
            <input  type="submit"   value="Predict">
        </form>                 '''

    </head>
</body>
</html>
```

One thing needed to be mention here is that the allowed extensions in this project are set. But if anyone wants to add or delete the extensions allowed, it can also be achieved simply by modifying the allowed_file( ) function and ALLOWED_EXTENSIONS in init1.py:

```
ALLOWED_EXTENSIONS = 'png,jpg'
```

```
def allowed_file(filename):
        return '.' in filename and filename.rsplit('.',1)[1].lower() in ALLOWED_EXTENSIONS
```

For the prediction part, the function upload_file( ) is defined in init1.py:

```
#Authenticates the login
@app.route('/pic_identity',methods=['GET','POST'])
def upload_file():
        if  request.method  =='POST':
                if 'file' not in request.files:
                        flash('No file part')
                        return  redirect(request.url)
                file = request.files['file']
                if  file.filename   ==  '':
                        flash('No   selected    file')
                        return  redirect(request.url)
                if  file    and allowed_file(file.filename):
                        filename    =   secure_filename(file.filename)
                        ext = filename.rsplit('.',1)[1]
                        new_filename = 'uploaded'+'.'+ext
                        file.save(os.path.join(UPLOAD_PATH,new_filename))
                        process_image.main(new_filename,ext)
                        result,result_int = output.conv_fc('processed_image.'+ext)
                        query = 'INSERT INTO image_information VALUES(%s,%s,%s)'
                        cursor.execute(query,(filename,time.strftime('%Y.%m.%d %H:%M:%S
                            ',time.localtime(time.time())),int(result_int)))
                        db.commit()



                        return  render_template('result.html',result = result)
```

4 Docker implementation

The purpose of implementing Docker container is making the service be used by a wider range of users. The implementation will not be bothered by the environment, and it is more efficient compared with traditional methods.

After building the service, building the Docker image is the next step. The following are what should be included in Docker.

First, as shown above, init1.py contains the part of prediction, and upload.html contains the part of uploading images.

Second, requirements.txt contains all the open source libraries where the functions in this project can be found:

```
bleach==1.5.0
certifi==2016.2.28
Click==7.0
cycler==0.10.0
enum34==1.1.6
Flask==1.1.1
html5lib==0.9999999
itsdangerous==1.1.0
Jinja2==2.10.1
kiwisolver==1.1.0
Markdown==3.1.1
MarkupSafe==1.1.1
matplotlib==3.0.3
numpy==1.17.0
opencv-python==4.1.0.25
Pillow==6.1.0
protobuf==3.9.0
pyparsing==2.4.2
python-dateutil==2.8.0
setuptools==36.4.0
six==1.12.0
tensorflow==1.4.0
tensorflow-tensorboard==0.4.0
Werkzeug==0.15.5
wheel==0.29.0
wincertstore==0.2
PyMySQL == 0.9.3
```

Third, Dockerfile helps to run Docker:

```
FROM  python:3.5
ADD ./mnist /code
WORKDIR /code
RUN pip install --default-timeout=2000 --no-cache-dir -r requirements.txt
CMD ["python","/code/package/init1.py"]
```

5 PhpMyAdmin Database

This project uses phpMyAdmin database to record data we generated during the process. Two things need to be done here.

First, python should be connected with phpMyAdmin database. “db” is the command used to make this connections:

```
db =
    pymysql.connect(host='192.168.99
    .100',port=3306,user='root',password='root',db='mnist',charset='utf8mb4',cursorclass=pymysql
    .cursors.DictCursor)
cursor = db.cursor()
ALLOWED_EXTENSIONS = 'png,jpg'
UPLOAD_PATH = os.getcwd()
#Define a route to hello function
```

Second, all the results of predictions including the numbers, and the corresponding time need to be put into the database:

```
        result,result_int = output.conv_fc('processed_image.'+ext)
        query = 'INSERT INTO image_information VALUES(%s,%s,%s)'
        cursor.execute(query,(filename,time.strftime('%Y.%m.%d %H:%M:%S
            ',time.localtime(time.time())),int(result_int)))
        db.commit()
```

6 Docker Construction

The last work is Docker Construction. Users are suggested to follow the instructions to make sure the process goes well.

1) Download phpMyAdmin and MySQL from Docker registry:

docker pull phpmyadmin/phpmyadmin

docker pull mysql:5.6

2) We need two Docker containers, which are the container runs the main service and records data in database, and the container implements the database and receives the data.

Start the two Docker containers :

docker run --name test-mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=root -d mysql:5.6

docker run --name test-phpmyadmin -p 8080:80 --link test-mysql:db -d phpmyadmin/ phpmyadmin:latest

In this case, two containers expose phpMyAdmin and MySQL to port 8080 and port 3306.

3) Go to the website by typing “the default IP of Docker machine” + “:8080”, and then sign in to phpMyAdmin with “username” and “password” (both of them are root).

4) Create “database mnist”, and create table “picture_information”.

5) Use the following SQL query to create table image_information： CREATE TABLE `test`.`image_information` (`name` VARCHAR(50) NOT NULL , `time`DATETIME NOT NULL , `output` INT NOT NULL)

7 Conclusion

7.1 Results

Based on the above implementation, the numbers in the images submitted by users can be predicted, and the results can be stored in the database. Moreover, the whole process is deployed into Docker, making it more convenient for users to get access to the service.

7.2 Improvements

Here are still things that can be improved which are not covered in this project.

First, when training the MNIST model, the training times is set to be 3000. The reason why 3000 is chosen is that in this case the training accuracy is within the range from 0.98 to 1. However, there might be a better number of training times that can increase the accuracy of the trained model. But we do not know the best answer. On the one hand, if the number is too small, the accuracy can be low either. On the other hand, if the number is too large, it may be overtrained. More work needs to be done to find the best solution.

Second, when recognizing the numbers in images, we use grey levels to identify the numbers. This can identify black numbers with white background, or numbers of other colors that are darker than white. However, there might be the situation that some users submit images of white numbers with black background. In this case, the MNIST model can not identify those numbers. Potential solution can be adding a new function to shifting colors and implementing the binary images.

Third, this project uses phpMyAdmin database, which can be substituted by other kinds of database, such as Cassandra.

Finally, this project requires individual users to operate on their own laptop. What are potential ways that different users can make connections with each other, so that a group of people can share the predictions of results, and work together to identify thousands of images, to gain higher accuracy of the model, which can even recognize “messy” handwriting and the more human thing in general?

Work Cited

[1] MNIST code
https://github.com/tensorflow/tensorflow/blob/r1.4/tensorflow/examples/tutorials/mnist/mnist_softmax.py

[2]Data
https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#2cede47d60ba

[3] MNIST
http://yann.lecun.com/exdb/mnist/