Project Report

Sun Jian

2019/8/26

Abstract

Project Overview

The main job of the project is to take fully use of docker to deploy a user handwriting digit recognition program into container. The goal of the project is to learn the frontier technology such as container Cassandra, Hadoop, Spark.etc and take these technologies into practices. Besides, we adopted supervisor Dr. Zhang Fan's valuable suggestions regarding to computer science's foreground and prospect and we listen to his experienced work history, which we deeply benefited from these valuable lessons. We started the project on July 7th, the first course we learned fundamental and required tools for the whole project and understood the implementation process of the project. Docker was introduced to us in the second course and we are engaged with container technology. In the third courses we learned Cassandra, a distributed, wide column store, NoSQL database. I successfully took use of these techniques into practice and suited purpose of supervisor Dr.Zhang's requirements.

Required Knowledge Review

(1)Neural network

Let's have a brief overview of how neural networks work.Suppose we get some input X1, X2 and X3. We try to map these three inputs two two outputs, cat and dog. In these case cat and dog are two neurones. We could use a single hidden layer. Then each of the input data connected to each of the neutrons in that first hidden layer, and each of these connection has its own unique weight. Even though we can connect the first hidden layer to the output layer, here is the problem that the relationship between X1 and cat or dog all the other inputs would only be linear relationships. Thus, if we're looking forward to map a nonlinear relationships which is probably going to be the case in a complex question. Then we need to have two or more Hidden Layer. Here we know that one Hidden Layer means we just have a neural network, and two or more Hidden Layers means we have a quote-unquote deep neural network. Then after these neurones fully connected all unique weights, each of those links (black lines) has a unique weight associated with it. Let's say at the final output layer, this output layer is almost certain to have just a sigmoid activation function and then we are going to say the cat is 0.81 and the dog is 0.19. Thus, in this case we have 81% confidence it is a cat and 19% confidence it is a dog. Then we think it is a cat.

(2)Convolution Neural Network

In this project, we use convolution to build our Keras Model.

Thus, in general, we have the following process.

IMAGE —> CONVONLUTION —> MAX POOLING —>

CONVOLUTION —>MAX POOLING —> FULLY CONNECTED —>

FULLY CONNECTED —>CLASSIFIER

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(5, 5), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

(The image above is what it looks like in project)

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=128, epochs=4, verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Then we compile this model by model.compile(),

Here we use the optimizer Adam.

After running this python file, we get 99.29% test accuracy which is

pretty good.

(3)Mnist

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. It was created by "re-mixing" the samples from NIST's original datasets. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments. Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.

The MNIST database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset. There have been a number of scientific papers on attempts to achieve the lowest error rate; one paper, using a hierarchical system of convolutional neural networks, manages to get an error rate on the MNIST database of 0.23%. The original creators of the database keep a list of some of the methods tested on it. In

their original paper, they use a support-vector machine to get an error rate of 0.8%. An extended dataset similar to MNIST called EMNIST has been published in 2017, which contains 240,000 training images, and 40,000 testing images of handwritten digits and characters.

In our project, we import mnist data set from keras.datasets as below

```
from keras.datasets import mnist
```

(4)Flask

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools. Extensions are updated far more frequently than the core Flask program.

In our project, the Flask Router will link to the HTML5 file.

The Router then saves the handwritten image and requests the prediction from the MNIST model. After the result returns, the Router forwards the result to HTML5 and show it to user via browser and submits all two data to the Cassandra Database Container.

(5)Docker

Docker is mainly a software development platform and a kind of virtualization technology that makes it easy for us to develop and deploy apps inside of neatly packaged virtual containerized environments.

This project is constructed by two docker Containers: an Application Docker Container and a Database Docker Container. Both the Application Docker Container and Database Docker Container are connected by Docker Network Bridge, so that they can communicate with each other.

In my project, I installed docker and developed my project in windows10 environment instead of Linux environment, which caused some unexpected errors during the developing period. I will introduce these problems afterwards. Overall, windows environment is not recommended to apply docker tool. （如果程序需要映射目录并且对应的目录需要权限设置，这个操作在win下操作会出现问题。网络和端口转接方面也和Linux有所不同。总之，正式环境下不建议在win中使用docker）

(6)Cassandra

Cassandra is a free open source no sequel database. It stores that values in the form of key value pairs. Cassandra in highly robust because it has a masterless replication so basically Cassandra works in the principle of clustering.We can Strat the Cassandra by creating a KEYSPACE.The following image how we create a new KEYSPACE in the terminal.

```
CREATE  KEYSPACE [IF NOT EXISTS] keyspace_name
   WITH REPLICATION = {
      'class' : 'SimpleStrategy', 'replication_factor' : N }
   }
```

In the project, we use Cassandra-driver, which may have a little different with the above attachment.

First we need to give the input of the variable KEYSPACE.

```
KEYSPACE = "mnist_key"
```

In the project, we name KEYSPACE “mnist_key”. Then similar to the work we did in the terminal, we will assign the details of this KEYSPACE.

```
def createKeySpace():
    cluster = Cluster(contact_points=['192.168.99.100'], port=9042)
    session = cluster.connect()
    log.info("Creating keyspace...")
    try:
        session.execute("""
            CREATE KEYSPACE %s
            WITH replication = { 'class': 'SimpleStrategy', 'replication_factor': '3' }
            """ % KEYSPACE)

        log.info("Setting keyspace...")
        session.set_keyspace(KEYSPACE)
```

Then let's start to create the table inside this KEYSPACE.

We can also create the table through terminal. We must tell Cassandra which variable is PRIMARY KEY, since it specifies which node will hold a particular table row. In the project we create the table by using Cassandra-driver.

```
CREATE TABLE record_data(
    date timestamp,
    result text,
    PRIMARY KEY (date)
)
""")
```

It is very similar to the way we did in the terminal. Here we let "date" be the PRIMARY KEY.

The last thing we need to do is inserting the data into our table.

```
Insert into keyspace_name.table_name(
    ColumnName1,
    ColumnName2,
    ColumnName3 . . . .)
values (
    Column1Value,
    Column2Value,
    Column3Value . . . .)
```

```
def insertData(date, result):
    cluster = Cluster(contact_points=['192.168.99.100'], port=9042)
    session = cluster.connect(KEYSPACE)
    log.info("Inserting data...")
    try:
        session.execute("""
            INSERT INTO record_data (date, result)
            VALUES(%s, %s);
            """, (date, result))
    except Exception as e:
        log.error("Unable to insert data")
        log.error(e)
```

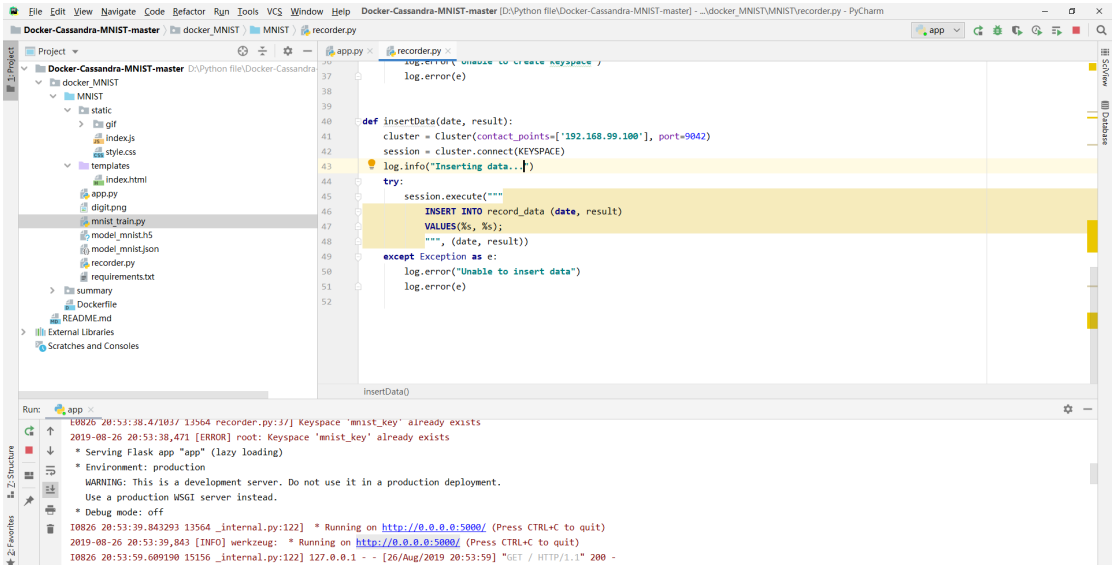
We must notice that the contact point should be set into 192.168.99.100, since docker is configured to use the default machine with IP 192.168.99.100.

Project Technical Process

The main job for this project is to recognize the user's handwritten digit. User can upload an image file, and it will return the result on the right side of the page. The prediction and date will be recorded in the Cassandra database.

This project consist of 2 procedures. The basic work we need to do is that run the digit recognition program in local environment.

Here we must notice that the site in browser is ‘local:5000’.



The next step is to deploy the project into docker.This project is constructed by two docker containers: an application docker container and a database docker container.

While the user submits ports in any browser, the Flask Router will link to the HTML5 file. The Router then saves the handwritten image and requests the prediction from the MNIST model. After the result returns, the Router forwards the result to HTML5 and show it to user via browser and submits all two data to the Cassandra Database Container through the Docker Network Bridge.

My Product

I think the core of the project consist of following functions:

(1) imagepredict

```
def imagepredict(image_path):
    os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
    result=imageprepare(image_path)
    init = tf.global_variables_initializer()
    saver = tf.train.Saver

    with tf.Session() as sess:
        sess.run(init)
        saver = tf.train.import_meta_graph('model.ckpt.meta')
        saver.restore(sess,'model.ckpt')
        graph=tf.get_default_graph()
        x = graph.get_tensor_by_name("x:")
        keep_prob = graph.get_tensor_by_name("keep_prob:0")
        y_conv = graph.get_tensor_by_name("y_conv:0")

        prediction=tf.argmax(y_conv,1)
        predint = prediction.eval(feed_dict = {x:[result],keep_prob:1.0},session=sess)
        return str(predint[0])
```

The function is to submit a image path and return the prediction of the image digit. In my project, I convert the image we draw into the raw data format of the image as below.

```
def predict():
    """
    whenever this function is called, we're going to convert the image we draw
    into the raw data format of the image.
    :return: string
    """
    img_data = request.get_data()
    convert_img(img_data)
    img = cv2.imread('digit.png', 0)
    img = cv2.resize(img, (28, 28))
    img = np.invert(img)  # change black to white, vice versa.
    img = img.reshape(1, 28, 28, 1)
    with graph.as_default():
        output = model.predict(img)
        response = np.array_str(np.argmax(output, axis=1))[1]

        # record data into cassandra with the Keyspace mnist_space, table recorder.
        ticks = int(round(time.time() * 1000))
        now = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(ticks / 1000))
        recorder.insertData(now, response)

        return response
```

(2)creating Cassandra KEYSPACE

```
import logging

log = logging.getLogger()
log.setLevel('INFO')
handler = logging.StreamHandler()
handler.setFormatter(logging.Formatter("%(asctime)s [%(levelname)s] %(name)s: %(message)s"))
log.addHandler(handler)
from cassandra.cluster import Cluster
from cassandra.query import SimpleStatement

KEYSPACE = "mnist_key"


def createKeySpace():
    cluster = Cluster(contact_points=['192.168.99.100'], port=9042)
    session = cluster.connect()
    log.info("Creating keyspace...")
    try:
        session.execute("""
            CREATE KEYSPACE %s
            WITH replication = { 'class': 'SimpleStrategy', 'replication_factor': '3' }
            """ % KEYSPACE)
```

（3）input correct docker command

1. construct the docker network bridge so that these to containers can communicate with each other

```
docker network create mnist_test
```

2. pull the Cassandra Database image from Docker Hub.

```
docker pull Cassandra
```

3. construct the docker container for Cassandra image.

```
docker run –network mnist_test -p 9042:9042 cassandra:latest
```

4. build the application image from the Dockerfile.

```
docker build -t mnist:latest .
```

5. run the container

```
docker run -p 5000:5000 mnist:latest
```

6. submit the Ports (it should be 192.168.99.100:8000) to any web browser.

7. call cqlsh shell

```
docker exec -it mnist_cassandra cqlsh
```

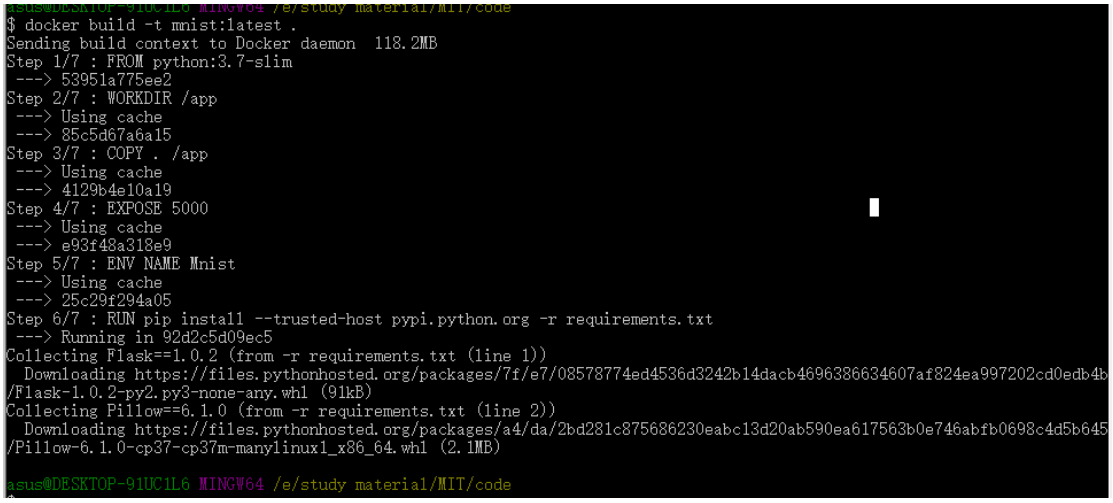
8. view the data recorded in Cassandra database.

```
USE mnist_key;
select*from record_data;
```

The Problems I Encountered

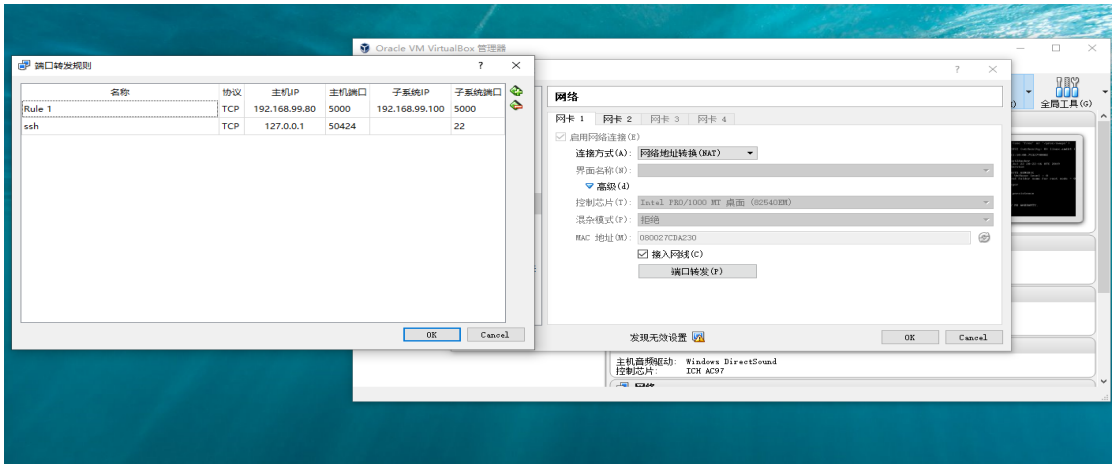
When deploying the program into docker and building the image, running procedure always stuck. Therefore the program cannot carry on and be underway. I even went to 网吧 and the building procedure still did not work. I think it should be caused by erroneous Internet.

I solved the problem when I went back to Korea.



Another problem I met is that I use Docker Toolbox to do the project and I need to set port mapping in Virtual Box as below.

这里的子系统IP是virtual box的默认IP，端口为Flask程序设置的5000端口



The Knowledge I Learned

I learned the frontier technology such as container, Cassandra , Hadoop and spark.etc in the curriculum series. It is a great experience for me to communicate and cooperate with brilliant classmates to overcame the difficulties I encountered in the progress. Besides, I watched online curriculum videos regarding with Flask and docker to gain a deeper insight of these technologies and try to look for the solutions I need during the period of implementing the project. Also, I read a book called 《Docker 技术入门与实践》 (Docker Primer) to study docker deeply and seeked inspiration of coding. Overall, I learned how to take fully use of the resources I have to solve a problem.

Acknowledgement

The current report is a stage report. And the final completed reported will be submitted by next weekend.

Thanks a lot for supervisor Dr.Zhang Fan's great leadership and patience.