Summary Report

2019/08/09

Tianfangtong Zhang

Index

0 Project Background

0.1 Introduction

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. In the HTML5 file, there is a canvas which user can use mouse to draw. The Router then saves the handwritten image and requests the prediction from the MNIST Keras 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.

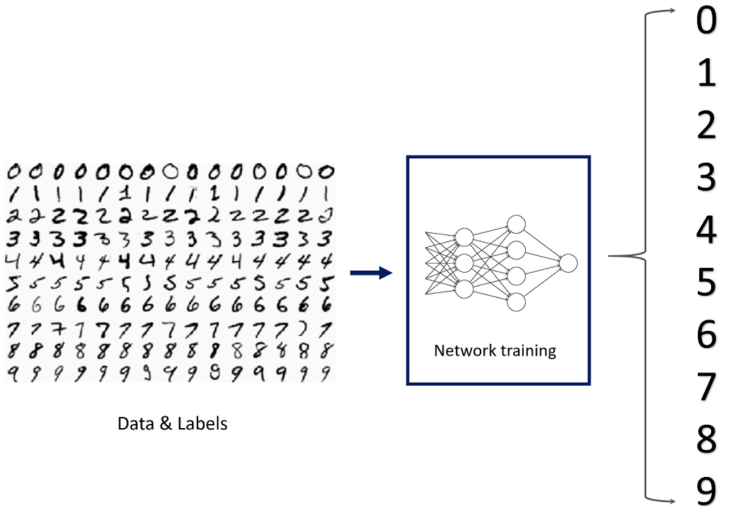
The main job for this project is to recognize the user's handwritten digit. User can draw the digit in the canvas and press "predict it" button to start to predict the image, and it will return the result on the right side of the page. User can also press "clear it" button to clear their drawing if he is not satisfy with it. Each time, the prediction and date will be recorded in the Cassandra database.

0.2 MNIST

The MNIST dataset consists of handwritten digit images (from 0 to 9) which include 60,000 examples of training set and 10,000 examples of testing set. It is a subset of a larger set available from NIST. However, the official training set of 60,000 examples is divided into an actual training set of 50,000 examples and 10,000 validation examples. Additionally all digit images are black and white and sized-normalized and with a fix size image of 28 x 28 pixels. In general, each digit image is represented by a value between 255 and 0, such that 255 is black and 0 is white and anything in between is a different degree of grey.

This dataset is combined by two of NIST's database: Special Database 1 and Special Database 3. These two databases includes digits hand-written by high school students and employees of the United States Census Bureau.

MNIST dataset is widely used in machine learning and image processing. On the other hand, this dataset is used by researchers to test and compare their research results with others.

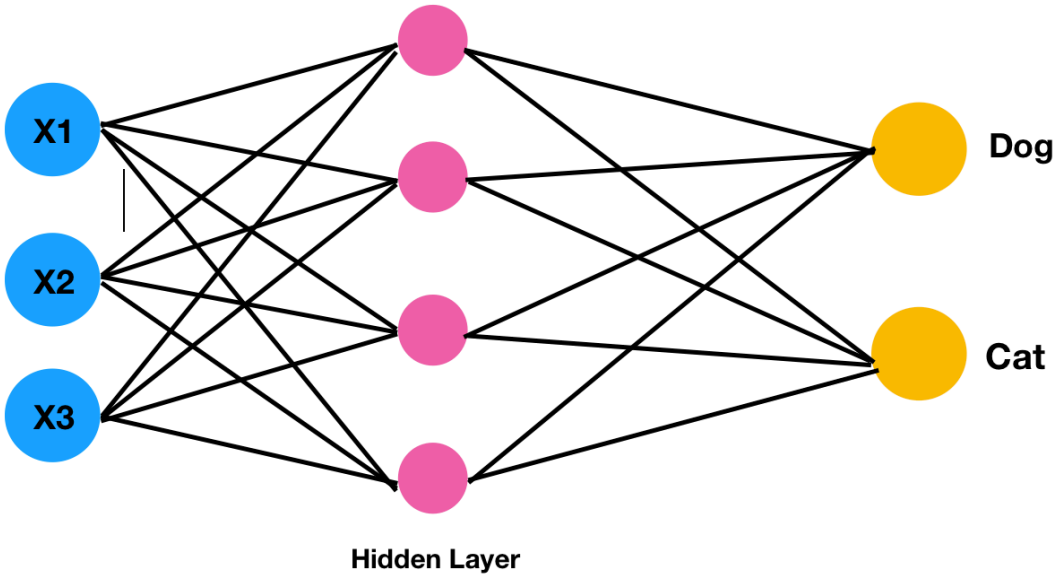


1 Project Knowledge Review

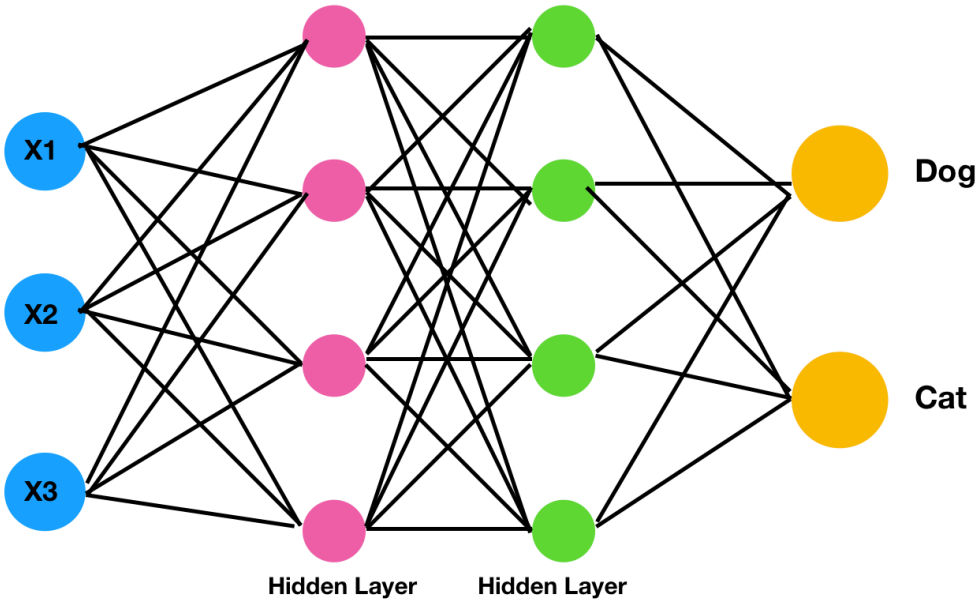
1. 1 Neural network

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.



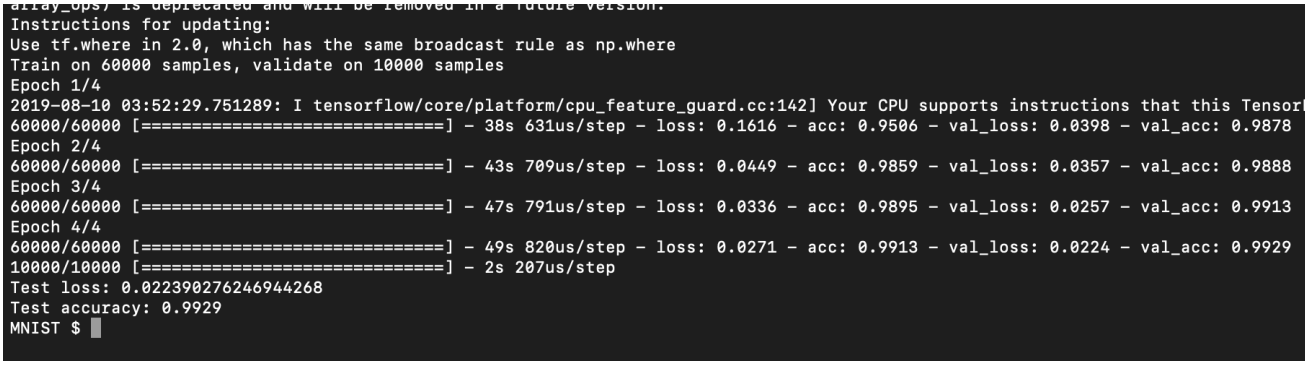
Then after these neurones fully connected all unique weights, each of those links (black lines) has a unique weight associated with it. Let's say at the final output layer, this output layer is almost certain to have just a sigmoid activation function and then we are going to say the cat is 0.81 and 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.

1.2 TensorFlow

“TensorFlow is an open-source machine learning library for research and production. TensorFlow offers APIs for beginners and experts to develop for desktop, mobile, web and cloud.”

In this project, Keras with backend TensorFlow to construct the MNIST model so that it can classify the user's handwritten digit.

This MNIST model gets to 99.29% test accuracy after 4 epochs.
(User can run mnist_train.py first to test the accuracy.)



1.3 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 —>

```
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'))
```

CLASSIFIER

```
compile(optimizer, loss=None, metrics=None, loss_weights=None,
        sample_weight_mode=None, weighted_metrics=None, target_tensors=None)
```

(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.

```
Test loss: 0.022390276246944268
Test accuracy: 0.9929
MNIST $
```

1.4 JSON

In this project, we need to save Model to JSON.

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for machines to parse and generate and say for human to read and write. It is built on two structures, a collection of name/value pairs (in other language it is realized as object, record, struct, dictionary, has table, keyed list, or associative array) and an ordered list of values (in other language it is realized as array, vector, list or sequence).

Focus on our project, we used JSON to save the MNIST model. Sicne Keras separates the concerns of saving model architecture and saving model weights. Thus, we saved MNIST model architecture to JSON and model weights to HDF5 format

```
model_json = model.to_json()
with open("model_mnist.json", "w") as json_file:
    json_file.write(model_json)

model.save_weights("model_mnist.h5")
```

Before MNIST model compiling, we need to open JSON file, read it and load the model.

```
file = open('model_mnist.json', 'r')
mnist_model = file.read()
file.close()
loaded_model = model_from_json(mnist_model)
loaded_model.load_weights("model_mnist.h5")
```

1.5 Flask

“Flask is lightweight WSGI web application framework.” Flaks is an excellent micro framework that really makes it enjoyable to work with this back-end web applications.

In our project, the Flask Router will link to the HTML5 file (index.html).

```
@mnist_web.route('/')
def index():
    """
    show html file.
    :return:
    """
    return render_template("index.html")
```

In the HTML5 file, there is a canvas which user can use mouse to draw.

```
<canvas id="canvas" width="280" height="280" style="border:8px solid;
float: left; margin: 140px; margin-top:360px;  border-radius: 5px; cursor: crosshair;"></canvas>
```

The Router then saves the handwritten image and requests the prediction from the MNIST Keras model. After the result returns, the Router forwards the result

```
@mnist_web.route('/predict/', methods=['GET', 'POST'])
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
```

to HTML5 and show it to user via browser and submits all two data to the Cassandra Database Container.

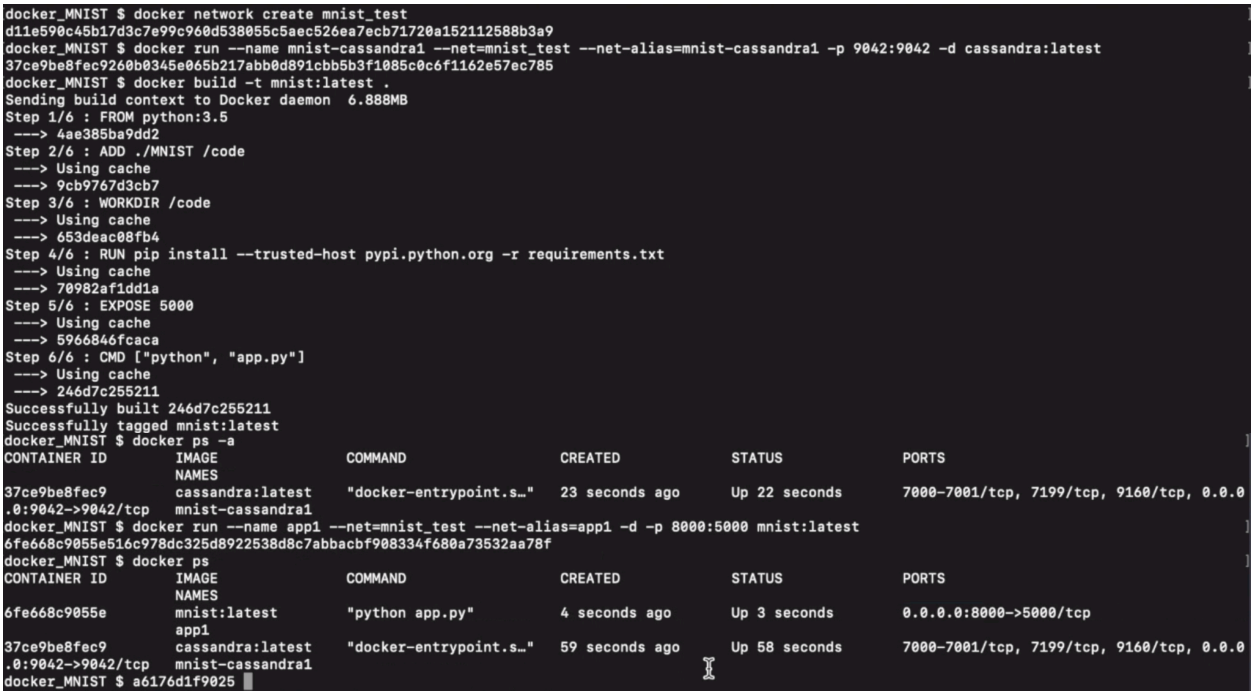
This will be the interface of our application. The predicted result will be showed to user on the right side of the page with string “I predicated the digit is:”

1.6 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.

The following image shows how we use Docker and Cassandra Database to deploy our application.



```
# construct docker network bridge.
docker network create mnist_test1
# constrict the docker container (for Cassandra)
docker run --name mnist_cassandra --net=mnist_test1 --net-alias=
mnist_cassandra -p 9042:9042 -d cassandra:latest
# build the image from Dockerfile
docker build --t mnist:latest .
# construct the docker container  (for our app)
docker run --name mn_app --net=mnist_test1 --net-alias=
mn_app -d -p 8000:5000 mnist:latest .
# list all runnning container.
docker ps
```

Here we will introduce docker cheat sheet. It includes all commands which we feel useful in this project.

COMMAND	DESCRIPTION
docker ps	List running containers
docker ps -a	List all containers
docker pull	Pull an image
docker images	List images
docker rm	Remove one or more container (only work for stopped containers)
docker —force rm	Force to remove one or more running container
docker start	Start one or more stopped containers
docker stop	Stop one or more running containers
docker —version	Show the docker version information
docker create network [network name]	Create a new network create
docker network rm	Remove the network
docker info	View details about docker installation
docker build —t=[image name]	Create a new docker image
docker run -p [host:port] [container name]	Publish a container's port to the host
docker run net=[network] -p [host:port] [container name]	Run the container with the network bridge and publish this container port to the host

1.7 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=['mnist_cassandra'], port=9042)
    # cluster = Cluster(contact_points=['127.0.0.1'], 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 table_name
( ColumnName1 type PRIMARY KEY,
  ColumnName2 type,
  ColumnName3 type);
```

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

```
log.info("Creating table...")
session.execute("""
    CREATE TABLE record_data(
        date timestamp,
        result text,
        PRIMARY KEY (date)
    )
    """)
```

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

First way

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

inserting data via terminal.

```python
def insertData(date, result):
    cluster = Cluster(contact_points=['mnist_cassandra1'], port=9042)
    # cluster = Cluster()
    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)
```

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

In our project, we use Cassandra-driver.

Then we can use the following command to call cqlsh, and take a look at our Cassandra.

```
~ $ docker exec -it mnist-cassandra1 cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.4 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> DESC KEYSPACES;

system_schema  system_auth  system  mnist11  system_distributed  system_traces

cqlsh> use mnist11;
cqlsh:mnist11> select*from record_data;

 date | result
------+--------

(0 rows)
cqlsh:mnist11> select*from record_data;

 date                            | result
---------------------------------+--------
 2019-08-08 23:41:36.000000+0000 |      3
```

In Cassandra, it will record two data (predicated) result and data into table record_data

Let’s introduce some helpful cql commands that we will use to view the data in the table which records our application’s data.

COMMAND	DESCRIPTION
USE KEYSPACE	Enter the KEYSPACE
select*from table_name	Show the table
DESC KEYSPCE	List all KEYSPACE

1.8 UI

We use HTML5 and CSS to design the UI of this project. Since it includes a canvas, we also use Javascript to catch the event of mouse on the canvas, so that user can draw on HTML5 canvas using a mouse.

The Canvas part is the most difficult part in UI design.

First we create a 280*280 Canvas in the HTML5 file index.html

```
<canvas id="canvas" width="280" height="280" style="border:8px solid;
float: left; margin: 140px; margin-top:360px;  border-radius: 5px; cursor: crosshair;"></canvas>
```

index.js includes all the events of the mouse movements. Including mouse press down, mouse press up, mouse move and mouse out of the canvas.
We set a boolean flag and assert it to be false at first so that we can make sure whether the mouse can be used to draw.

```
var flag=false
```

If mouse press up or out of the canvas, it will lose the ability to draw in the canvas.
For the event of mouse press down.

```
canvas.onmousedown=function(evt){

            var BeginX=evt.clientX-this.offsetLeft;
            var BeginY=evt.clientY-this.offsetTop;
            context.beginPath();
            context.moveTo(BeginX, BeginY);
            flag=true;

    }
```

For the event of mouse press up.

```
canvas.onmouseup=function(){
        flag=false;
    }
```

For the event of mouse move.

```
canvas.onmousemove=function(evt){
        if(flag){
            var endX=evt.clientX-this.offsetLeft;
            var endY=evt.clientY-this.offsetTop;
            context.lineTo(endX, endY);
            context.stroke();
        }
    }
```

For the event of mouse out of Canvas.

```
canvas.onmouseout=function(){
        flag=false;
    }
```

Also for the clean button feature, we used Javascript in the HTML5 file index.html.

```
<script type="text/javascript">

    $(".clButton").click(function(){
    var canvas = document.getElementById("canvas");
    var context = canvas.getContext( "2d" );
    context.clearRect( 0, 0, 280, 280 );
    context.fillStyle="white";
    context.fillRect(0,0,280,280);
    });

</script>
```

As well as, to show the predication result to user, we used Javascript in the HTML5 file index.html.

```
<script type="text/javascript">

    $(".prButton").click(function(){
        var g = document.getElementById("canvas");
        var img = g.toDataURL();
        var $SR = {{ request.script_root|tojson|safe }};
        $.ajax({
            type: "POST",
            url: $SR + "/predict/",
            data: img,
            success: function(data){
                $('#result').text('I predicate the digit is: '+ data);
            }
        });
    });

</script>
```

All the color style, font, size, and position are setting in the style.css.
For example, for the “clear it” button. In the style.css, it looks like this.

```
.clButton {
    background-color:#F0AF3F;
    display:inline-block;
    cursor:pointer;
    color:#ffffff;
    font-family:"Comic Sans MS", cursive, sans-serif;
    font-size:15px;
    padding:6px 17px;
    text-decoration:none;
    text-shadow:0px 1px 0px #5b8a3c;
}
```


```
'#72b352', endColorstr='#4DAF7C',GradientType=0);
```

It is designed to be orange in usual, but when mouse hover over this button, it will become green.

2 Summary

I'm so honoured to have the opportunity to participate this project. Unlike the usual projects or assignments I took in the university, this project is harder and more challenging. Most of the background of this project such as docker, Javascript, flask, mnist I have never seen before. I need to do lots of research so that I can understand and even use this new knowledge. We always do group work in the university, it is my first time to solo a whole project. I feel confidence and proud of myself I can finish this project.

Because of this project, I can have a interest in the machine learning. I can take a brief look of how neural network work. Even though, mnist is a patch in the machine learning field, I'm confident to learn more knowledge about machine learning in the following time.

At last, I would like to thank a lot to prof Zhang for teaching us the background and knowledge about this project and even more about big data, and giving me this opportunity to participate in this project and have access to machine learning.