IBM-EPBL / IBM-Project-31681-1660204101

SOLUTION ARCHITECTURE: A Novel Method for Handwritten Digit Recognition System

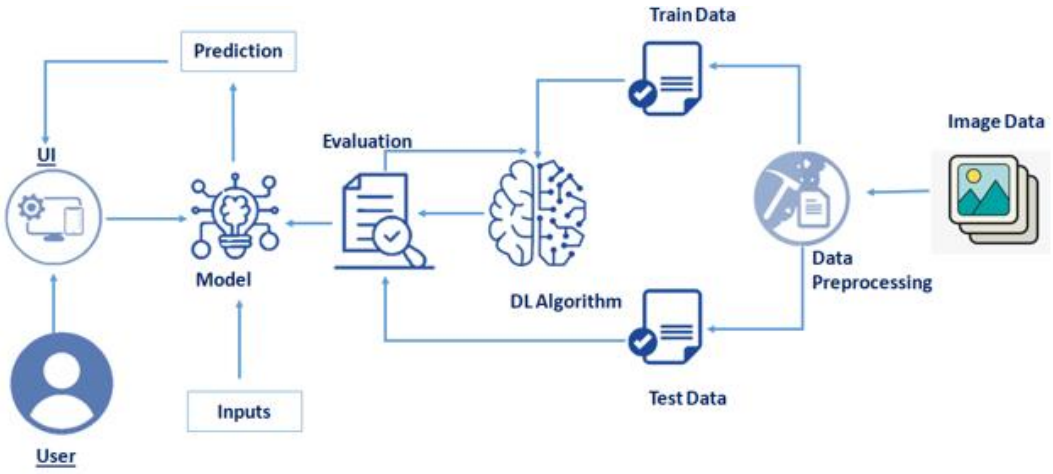
Team members:

1. Hariprakash S
2. Abu Backer Siddiq
3. Sivamanibala S
4. Zekkin Thanraj S

A Novel Method for Handwritten Digit Recognition System

Handwriting recognition is one of the compelling research works going on because every individual in this world has their own style of writing. It is the capability of the computer to identify and understand handwritten digits or characters automatically. Because of the progress in the field of science and technology, everything is being digitalized to reduce human effort. Hence, there comes a need for handwritten digit recognition in many real-time applications. MNIST data set is widely used for this recognition process and it has 70000 handwritten digits. We use Artificial neural networks to train these images and build a deep learning model. Web application is created where the user can upload an image of a handwritten digit. this image is analysed by the model and the detected result is returned on to UI

Technical Architecture:



Project Flow

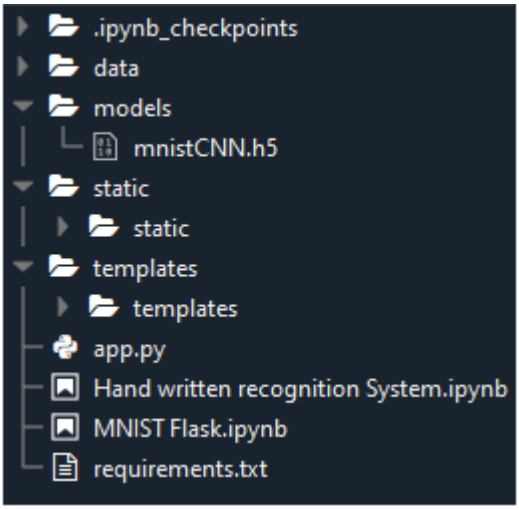
- The user interacts with the UI (User Interface) to upload the image as input
- The uploaded image is analysed by the model which is integrated
- Once the model analyses the uploaded image, the prediction is showcased on the UI

To accomplish this, we have to complete all the activities and tasks listed below

- Understanding the data.
  - Importing the required libraries
  - Loading the data
  - Analysing the data
  - Reshaping the data.
  - Applying One Hot Encoding
- Model Building
  - Creating the model and adding the input, hidden and output layers to it
  - Compiling the model
  - Training the model
  - Predicting the result
  - Testing the model by taking image inputs
  - Saving the model
- Application Building
  - Create an HTML file
  - Build Python Code

Project Structure

Create a Project folder which contains files as shown below



- We are building a Flask Application which needs HTML pages stored in the templates folder and a python script app.py for server-side scripting.
- The model is built in the notebook Hand written recognition system.ipynb
- We need the model which is saved and the saved model in this content is mnistCNN.h5
- The static folder will contain js and css files.
- The templates mainly used here are main.html and index6.html for showcasing the UI

Prerequisites

To complete this project, you should have the following software and packages

Anaconda Navigator:

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS. Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupiter notebook and Spyder

To install Anaconda navigator and to know how to use Jupyter Notebook a Spyder using Anaconda watch the video

To build Deep learning models you must require the following packages

Tensor flow: TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers can easily build and deploy ML powered applications.

Keras: Keras leverages various optimization techniques to make high level neural network API easier and more performant. It supports the following features:

- Consistent, simple and extensible API.
- Minimal structure - easy to achieve the result without any frills.
- It supports multiple platforms and backends.
- It is user friendly framework which runs on both CPU and GPU.
- Highly scalability of computation.

Flask: Web frame work used for building Web applications

Prior Knowledge

One should have knowledge on the following Concepts:

- **Supervised and unsupervised learning**
- **Regression Classification and Clustering**
- **Artificial Neural Networks**
- **Convolution Neural Networks**
- **Flask**

Understanding The Data

ML depends heavily on data, without data, it is impossible for a machine to learn. It is the most crucial aspect that makes algorithm training possible. In Machine Learning projects, we need a training data set. It is the actual data set used to train the model for performing various actions. TensorFlow already has MNist Data set so there is no need to explicitly download or create Dataset
The MNSIT dataset contains ten classes: Digits from 0-9. Each digit is taken as a class.

Importing The Required Libraries

Let’s first import the libraries

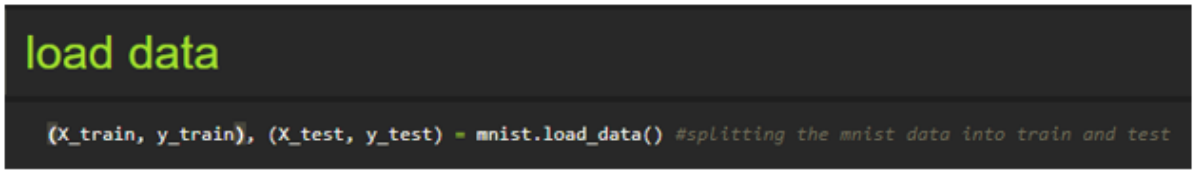
Importing the required libraries which are required for the model to run. The dataset for this model is imported from the Keras module.

The dataset contains ten classes: Digits from 0-9. Each digit is taken as a class

For a detail point of view on Keras and TensorFlow refer to the **link here:**

Loading The Data

The dataset for this model is imported from the Keras module.



We split the data into train and test. Using the training dataset, we train the model and the testing dataset is used to predict the results.

```
print(X_train.shape)#shape is used for give the dimension values #60000-rows 28x28-pixels
print(X_test.shape)
```

```
(60000, 28, 28)
(10000, 28, 28)
```

We are finding out the shape of X_train and x_test for better understanding. It lists out the dimensions of the data present in it.
in trainset, we have 60000 images, and in the test set we have 10000 images

Analysing The Data

Let's see the Information of an image lying inside the x_train variable

Understanding the data

```
X_train[0]#printing the first image
```

```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
         18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170,
        253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253,
        253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  18, 219, 253, 253, 253, 253,
        253, 198, 182, 247, 241,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  80, 156, 107, 253, 253,
        205,  11,   0,  43, 154,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  14,   1, 154, 253,
```

Basically, the pixel values range from 0-255. Here we are printing the first image pixel value which is index [0] of the training data. As you see it is displayed in the output.

With respect to this image, the label of this image will be stored in y_train let's see what is the label of this image by grabbing it from the y_train variable

```
y_train[0]#printing lable of first image

5
```

As we saw in the previous screenshot, we get to know that the pixel values are printed. Now here we are finding to which image the pixel values belong to. From the output displayed we get to know that the image is ‘5’.

Lets Plot the image on a graph using the Matplot library

```
import matplotlib.pyplot as plt #used for data visualization
plt.imshow(X_train[0]) #ploting the index=0 image

<matplotlib.image.AxesImage at 0x1c397af32e0>
```

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. By using the Matplotlib library we are displaying the number ‘5’ in the form of an image for proper understanding.

Note: You can see the results by replacing the index number till 59999 as the train set has 60K images

Reshaping The Data

As we are using Deep learning neural network, the input for this network to get trained on should be of higher dimensional. Our dataset is having three-dimensional images so we have to reshape them too higher dimensions

Reshaping Dataset ¶

```
# Reshaping to format which CNN expects (batch, height, width, channels)
X_train = X_train.reshape(60000, 28, 28, 1).astype('float32')
X_test = X_test.reshape(10000, 28, 28, 1).astype('float32')
```

We are reshaping the dataset because we are building the model using CNN. As CNN needs four attributes batch, height, width, and channels we reshape the data.

Applying One Hot Encoding

If you see our y_train variable contains Labels representing the images containing in x_train. AS these are numbers usually, they can be considered as numerical or continuous data, but with respect to this project these Numbers are representing a set of class so these are to be represented as categorical data, and we need to binaries these categorical data that's why we are applying One Hot encoding for y_train set

One-Hot Encoding

```
# one hot encode
number_of_classes = 10 #storing the no. classes in a variable
y_train = np_utils.to_categorical(y_train, number_of_classes) #converts the output in binary format
y_test = np_utils.to_categorical(y_test, number_of_classes)
```

One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. We apply One-Hot Encoding in order to convert the values into 0's and 1's. For a detailed point of view, look at this link

Now let's see how our label 5 is index 0 of y_train is converted

```
y_train[0] #printing the new label
```

```
array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)
```

As we see the new the label is printed in the form of 0's and 1's and is of type float.

Model Building

This activity includes the following steps

- Initializing the model
- Adding CNN Layers
- Training and testing the model
- Saving the model

Add CNN Layers

Creating the model and adding the input, hidden, and output layers to it

Creating the Model

```
# create model
model = Sequential()
# adding model layer
model.add(Conv2D(64, (3, 3), input_shape=(28, 28, 1), activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu'))
#model.add(Conv2D(32, (3, 3), activation='relu'))
#flatten the dimension of the image
model.add(Flatten())
#output layer with 10 neurons
model.add(Dense(number_of_classes, activation='softmax'))
```

The Sequential model is a linear stack of layers. You can create a Sequential model by passing a list of layer instances to the
constructor:

To know more about layers, watch the below video

Compiling The Model

With both the training data defined and model defined, it's time to configure the learning process. This is accomplished with a call to the compile () method of the Sequential model class. Compilation requires 3 arguments: an optimizer, a loss function, and a list of metrics.

Compiling the model

```
# Compile model
model.compile(loss='categorical_crossentropy', optimizer="Adam", metrics=['accuracy'])
```

Note: In our project, we have 2 classes in the output, so the loss is binary_crossentropy.
If you have more than two classes in output put "loss = categorical_cross entropy".

Train The Model

Now, let us train our model with our image dataset.
Fit: functions used to train a deep learning neural network

Fitting the model

```
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=5, batch_size=32)
```

```
Epoch 1/5
1875/1875 [==============================] - 184s 98ms/step - loss: 0.2451 - accuracy: 0.9497 - val_loss: 0.0966 - val_accuracy: 0.9
715
Epoch 2/5
1875/1875 [==============================] - 183s 98ms/step - loss: 0.0694 - accuracy: 0.9785 - val_loss: 0.0971 - val_accuracy: 0.9
714
Epoch 3/5
1875/1875 [==============================] - 183s 98ms/step - loss: 0.0487 - accuracy: 0.9850 - val_loss: 0.0829 - val_accuracy: 0.9
782
Epoch 4/5
1875/1875 [==============================] - 177s 94ms/step - loss: 0.0382 - accuracy: 0.9877 - val_loss: 0.0881 - val_accuracy: 0.9
769
```

Arguments:
steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your dataset divided by the batch size.

Epochs: an integer and number of epochs we want to train our model for.

Validation_data:

- an inputs and targets list
- a generator
- inputs, targets, and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.

validation_steps: only if the validation_data is a generator then only this argument can be used. It specifies the total number of steps taken from the generator before it is stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size.

Observing The Metrics

We here are printing the metrics which lists out the Test loss and Test accuracy

- Loss value implies how poorly or well a model behaves after each iteration of optimization.
- An accuracy metric is used to measure the algorithm's performance in an interpretable way.

Test The Model

Firstly, we are slicing the x_test data until the first four images. In the next step we the printing the predicted output.

Predicting the output

```
prediction=model.predict(X_test[:4])
print(prediction)
```

```
[[5.50544734e-15 7.41999492e-20 5.00876077e-12 1.26642463e-09
  3.52252804e-21 1.54133163e-17 3.15550259e-21 1.00000000e+00
  1.32678888e-13 6.44072333e-14]
 [1.51885260e-08 8.02883537e-09 1.00000000e+00 6.44802788e-13
  6.37117113e-16 3.40490114e-15 2.15804121e-08 2.18907611e-19
  3.38496564e-10 2.07915498e-20]
 [3.14093924e-08 9.99941349e-01 2.01593957e-06 1.45100779e-10
  5.25237965e-06 1.59223120e-07 3.15299786e-08 1.53995302e-07
  5.09846941e-05 1.14552066e-07]
 [1.00000000e+00 1.35018288e-14 2.28308122e-10 1.79766094e-16
  1.28767550e-14 7.12401882e-12 2.92727509e-11 3.52439052e-13
  2.56207252e-12 2.32345068e-12]]
```

```
import numpy as np
print(np.argmax(prediction,axis=1)) #printing our labels from first 4 images
print(y_test[:4]) #printing the actual labels
```

```
[7 2 1 0]
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

As we already predicted the input from the x_test. According to that by using argmax function here we are printing the labels with high prediction values

Observing the metrics

```
# Final evaluation of the model
metrics = model.evaluate(X_test, y_test, verbose=0)
print("Metrics(Test loss & Test Accuracy): ")
print(metrics)
```

```
Metrics(Test loss & Test Accuracy):
[0.1097492054104805, 0.9753000140190125]
```

Test The Model

Firstly, we are slicing the x_test data until the first four images. In the next step we the printing the predicted output.

Predicting the output

```
prediction=model.predict(X_test[:4])
print(prediction)
```

```
[[5.50544734e-15 7.41999492e-20 5.00876077e-12 1.26642463e-09
  3.52252804e-21 1.54133163e-17 3.15550259e-21 1.00000000e+00
  1.32678888e-13 6.44072333e-14]
 [1.51885260e-08 8.02883537e-09 1.00000000e+00 6.44802788e-13
  6.37117113e-16 3.40490114e-15 2.15804121e-08 2.18907611e-19
  3.38496564e-10 2.07915498e-20]
 [3.14093924e-08 9.99941349e-01 2.01593957e-06 1.45100779e-10
  5.25237965e-06 1.59223120e-07 3.15299786e-08 1.53995302e-07
  5.09846941e-05 1.14552066e-07]
 [1.00000000e+00 1.35018288e-14 2.28308122e-10 1.79766094e-16
  1.28767550e-14 7.12401882e-12 2.92727509e-11 3.52439052e-13
  2.56207252e-12 2.32345068e-12]]
```

```
import numpy as np
print(np.argmax(prediction,axis=1)) #printing our labels from first 4 images
print(y_test[:4]) #printing the actual labels
```

```
[7 2 1 0]
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

As we already predicted the input from the x_test. According to that by using argmax function here we are printing the labels with high prediction values

Save The Model

Your model is to be saved for future purposes. This saved model can also be integrated with an android application or web application in order to predict something.

Saving the model

```
# Save the model
model.save('models/mnistCNN.h5')
```

The model is saved with .h5 extension as follows:
An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

Test With Saved Model

Now open another jupyter file and write the below code

Taking images as input and checking results

```
# Importing the Keras libraries and packages
from tensorflow.keras.models import load_model
model = load_model(r'C:/Users/DELL/Hand written recognition System/models/mnistCNN.h5')
from PIL import Image#used for manipulating image uploaded by the user.
import numpy as np#used for numerical analysis
for index in range(4):
    img = Image.open('data/' + str(index) + '.png').convert("L")# convert image to monochrome
    img = img.resize((28,28))# resizing of input image
    im2arr = np.array(img) #converting to image
    im2arr = im2arr.reshape(1,28,28,1) #reshaping according to our requirement
    # Predicting the Test set results
    y_pred = model.predict(im2arr) #predicting the results
    print(y_pred)
```

```
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]]
[[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]]
```

Firstly, we are loading the model which was built. Then we are applying for a loop for the first four images and converting the image to the required format. Then we are resizing the input image, converting the image as per the CNN model and we are reshaping it according to the requirement. At last, we are predicting the result.

You can use predict_classes for just predicting the class of an image

Application Building

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has uploaded an image. The uploaded image is given to the saved model and prediction is showcased on the UI.
This section has the following tasks

- Building HTML Pages
- Building server-side script

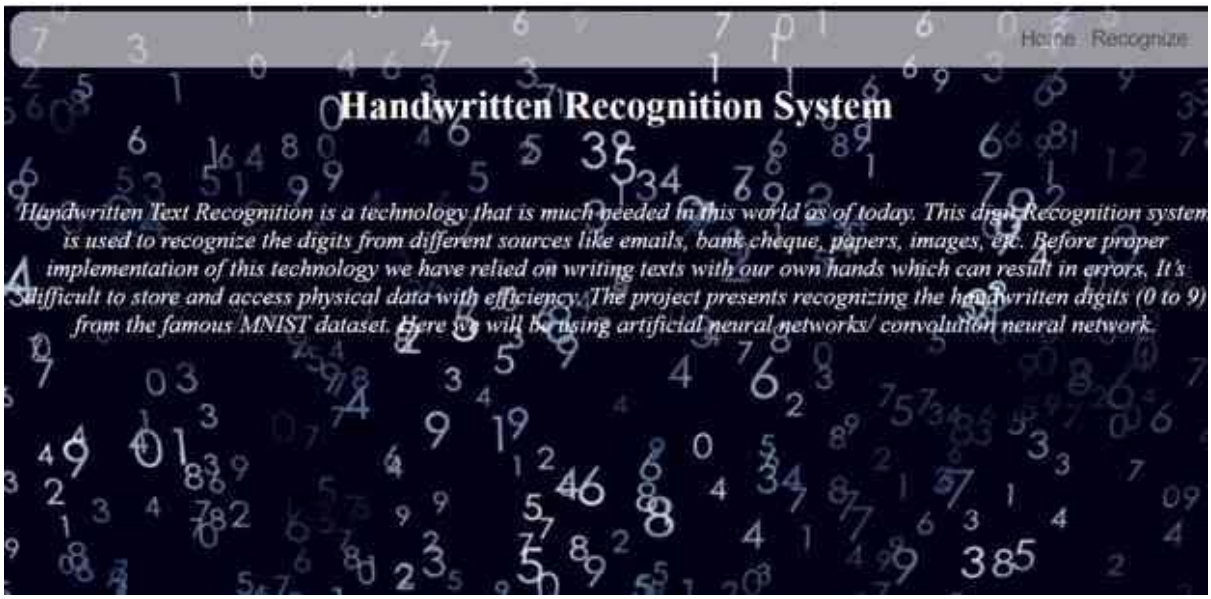
Create An HTML File

- We use HTML to create the front-end part of the web page.
- Here, we created 2 html pages- index.html, web.html.
- index.html displays home page.
- web.html accepts the values from the input and displays the prediction.
- For more information regarding HTML refer the link below

Please refer to the link for HTML code files

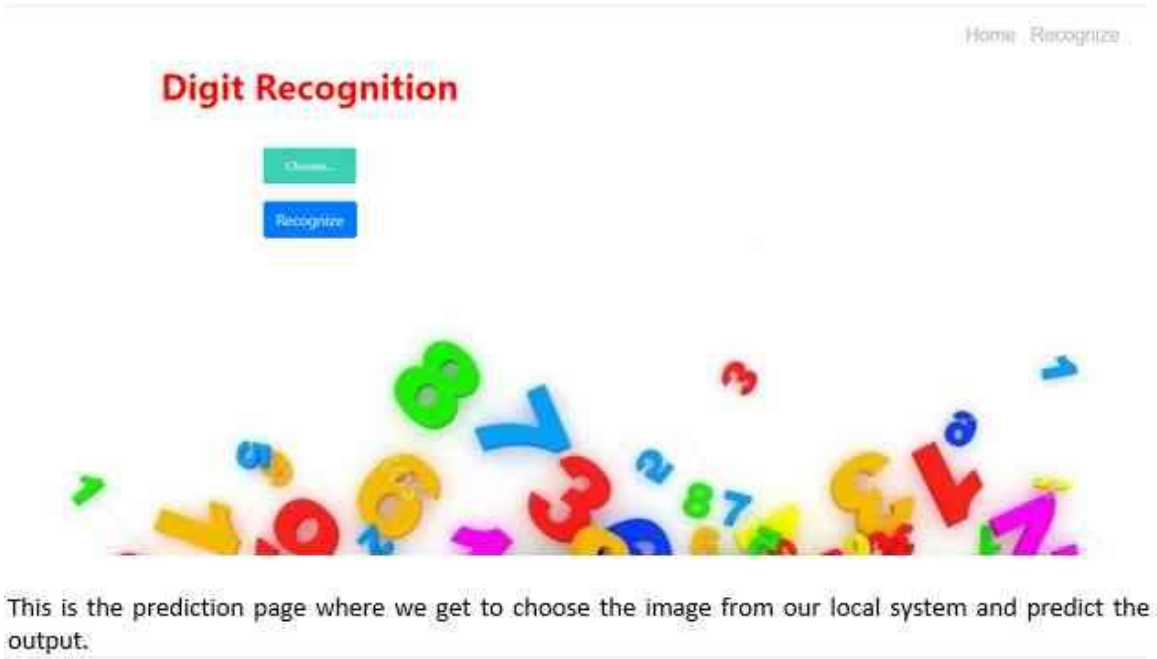
Let's see how our index.html file looks like

This is the main page which describes about the project and summarizes it.



Let's see how our web.html page looks like

This is the prediction page where we get to choose the image from our local system and predict the output.



Build Python Code (Part 1)

Let us build the flask file 'app.py' which is a web framework written in python for server-side scripting. Let's see step by step procedure for building the backend application.

- App starts running when the "__name__" constructor is called in main.
- render_template is used to return HTML file.
- "GET" method is used to take input from the user.
- "POST" method is used to display the output to the user.

Import Libraries:

```
from flask import Flask, render_template, request# Flask-It is our framework which we are going to use to
run/serve our application.
#request-for accessing file which was uploaded by the user on our application.

from PIL import Image #used for manipulating image uploaded by the user.
import numpy as np #used for numerical analysis
from tensorflow.keras.models import load_model#to load our model trained with MNIST data
import tensorflow as tf#to run our model.
```

Libraries required for the app to run are to be imported.

Routing to the html Page

```
@app.route('/') #default route
def upload_file():
    return render_template('main.html') #rendering html page
@app.route('/about') #Main page route
def upload_file1():
    return render_template('main.html') #rendering html page
@app.route('/upload') #main page route
def upload_file2():
    return render_template('index6.html')
```

We are routing the app to the HTML templates which we want to render.
Firstly, we are rendering the main.html template and from there we are navigating to our prediction page that is index6.html

Returning the prediction on UI:

```
@app.route('/predict', methods = ['POST']) #route for our prediction
def upload_image_file():
    if request.method == 'POST':
        img = Image.open(request.files['file'].stream).convert("L")# convert image to monochrome
        img = img.resize((28,28))# resizing of input image
        im2arr = np.array(img)#converting to image
        im2arr = im2arr.reshape(1,28,28,1) #reshaping according to our requirement
        y_pred = model.predict_classes(im2arr) #predicting the results
        print(y_pred) #printing our result in prompt
        #return 'Predicted Number: ' + str(y_pred) #returning our output
```

Build Python Code (Part 2)

Here the route for prediction is given and necessary steps are performed in order to get the predicted output.

```
    if(y_pred == 0) :
      return render_template("0.html",showcase =  str(y_pred))
    elif(y_pred == 1) :
      return render_template("1.html",showcase =  str(y_pred))
    elif(y_pred == 2) :
      return render_template("2.html",showcase =  str(y_pred))
    elif(y_pred == 3) :
      return render_template("3.html",showcase =  str(y_pred))
    elif(y_pred == 4) :
      return render_template("4.html",showcase =  str(y_pred))
    elif(y_pred == 5) :
      return render_template("5.html",showcase =  str(y_pred))
    elif(y_pred == 6) :
      return render_template("6.html",showcase =  str(y_pred))
    elif(y_pred == 7) :
      return render_template("7.html",showcase =  str(y_pred))
    elif(y_pred == 8) :
      return render_template("8.html",showcase =  str(y_pred))
    else :
      return render_template("9.html",showcase =  str(y_pred))
else:
      return None
```

Necessary conditions are given according to the input classes and the app will be returning the templates according to that.

Main Function:
This function runs your app in a web browser

Lastly, we run our app on the localhost. Here we are running it on localhost:8000

```
    else:
        return None
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000,debug=True)
    #app.run(debug = True) #running our flask app
```

Run The Application

- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command

```
(base) C:\Users\DELL>cd C:\Users\DELL\Hand_Written_Final

(base) C:\Users\DELL\Hand_Written_Final>python app.py
```

Navigate to the localhost where you can view your web page
Upload an image and see the predicted output on UI your page and output look like: