# North East University Bangladesh (NEUB)

Telihaor, Sheikhghat, Sylhet-3100

Department of Computer Science & Engineering (CSE)

## Project Proposal Spring 2025

**Course Code: CSE-460**

**Course Title:** Deep Learning Lab

**Submitted by-**

**Name: Galib Asfak Tanzir**

**ID: 0562210005101018**

**Semester: 7th**

**Section: B**

**Project Name:** Handwritten Digit Recognition System

**1st** requires us to collect images of digits (from 0 to 9). The pyscreenshot package can be used to collect images. This package can be downloaded with pip, Python Package Installer. This pip can be installed automatically when Python is installed. To install pyscreenshot, open any terminal and enter the following command:

pip install pyscreenshot.

I'm also importing time since I want to pause my execution for a second so that I can draw the following image (digit). Everything is clear in the code below, but some may have difficulty locating the exact bbox coordinate value. This coordinate value specifies which portion of your screen will be captured. This coordinate was discovered through trial and error. You might also try using the same coordinate that I am using for your first run. However, because our laptop sizes vary, it is possible that it will be different for you and me. So, you have to utilize the trial and error method to identify the correct coordinate value.

```
 import pyscreenshot as ImageGrab
import time
images_folder="captured_images/0/"

for i in range(0,100):
   time.sleep(8)
   im=ImageGrab.grab(bbox=(60,170,400,550)) #x1,y1,x2,y2
   print("saved......",i)
   im.save(images_folder+str(i)+'.png')
   print("clear screen now and redraw now........")
```

Secondly requires us to create our dataset from the images we obtained in Part 1. To construct a dataset, we must first assign 1 to the drawn region and 0 to the background. That means our dataset will only include two values: 0 and 1. I'm sure you're aware that pixel values range from 0 to 255. In most cases, 0 symbolizes black and 255 represents white. I'm assigning 0 to pixel values ranging from 0 to 100, and 1 to pixel values ranging from 100 to 255. Our pixel values are now merely 0 and 1, rather than 0 and 255. In this manner, I am producing a dataset (CSV file).

The final step is to open the dataset, shuffle it i.e., change the position of each row of data, and display it.

```
#Generate dataset
import cv2
import csv
import glob

header  =["label"]
for i in range(0,784):
   header.append("pixel"+str(i))
with open('dataset.csv', 'a') as f:
   writer = csv.writer(f)
   writer.writerow(header)

for label in range(10):
   dirList = glob.glob("captured_images/"+str(label)+"/*.png")

   for img_path in dirList:
       im= cv2.imread(img_path)
       im_gray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
       im_gray = cv2.GaussianBlur(im_gray,(15,15), 0)
       roi= cv2.resize(im_gray,(28,28), interpolation=cv2.INTER_AREA)

       data=[]
       data.append(label)
       rows, cols = roi.shape

       ## Fill the data array with pixels one by one.
       for i in range(rows):
           for j in range(cols):
               k =roi[i,j]
               if k>100:
                   k=1
               else:
                   k=0
               data.append(k)
       with open('dataset.csv', 'a') as f:
           writer = csv.writer(f)
           writer.writerow(data)
```

## Load the dataset:

Then we have to open the dataset, shuffle it i.e., change the position of each row of data, and display it.

```
import pandas as pd #pip install pandas
from sklearn.utils import shuffle #pip install scikit-learn
#0,....,1.....,2.....
#5,3,1,0,2,5,.......

data = pd.read_csv('dataset.csv')
data = shuffle(data)
print(data)
```

Thirdly requires us to train our model and calculate its accuracy. To train our model, we can utilize scikit-learn. Run the following command to install scikit-learn:

pip install scikit-learn

## Separating dependent and independent variables:

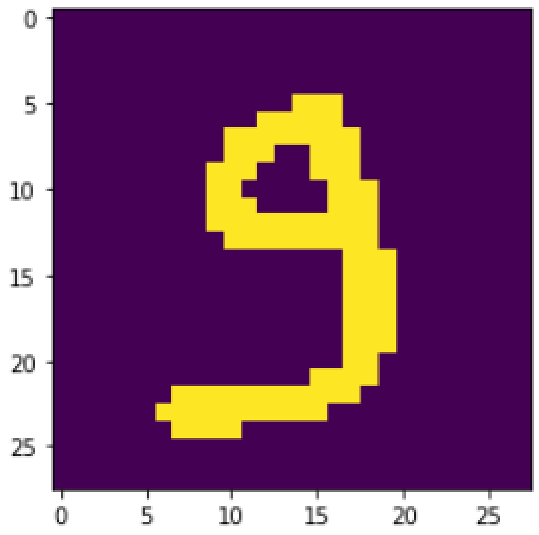
Then we have to separate the dependent (Y) and the independent variable (X). The pixel value (between 0 and 1) will be our independent variable. Keep in mind that each digit is represented by a massive amount of 0 and 1. Our digit (from 0 to 9) will be our dependent variable.

```
X = data.drop(["label"],axis=1)
Y= data["label"]
```

## Preview of one image using matplotlib:

```
%matplotlib inline
import matplotlib.pyplot as plt
import cv2
idx = 314
img = X.loc[idx].values.reshape(28,28)
print(Y[idx])
plt.imshow(img)
```

```
9
<matplotlib.image.AxesImage at 0x294f1119708>
```



## Train-Test split:

```
from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y = train_test_split(X,Y, test_size = 0.2)
```

## Fit the model using svc and also save the model using joblib:

```
import joblib
from sklearn.svm import SVC
classifier=SVC(kernel="linear", random_state=6)
classifier.fit(train_x,train_y)
joblib.dump(classifier, "model/digit_recognizer")
```

## Calculate accuracy:

```
from sklearn import metrics
prediction=classifier.predict(test_x)
print("Accuracy= ",metrics.accuracy_score(prediction, test_y))
```

I am giving 80% of my data to the training part and the remaining 20% to the testing part (as I have defined test_size = 0.2). Remember, training images are used to create our model, and testing images are used to calculate accuracy. In the training part, we train our model i.e., we give our model pixel value (bunch of 0 and 1) and also label (i.e. which digit is this). That means we teach our model. After this step, our model gets learned. Now, in the testing part, we only give pixel value (a bunch of 0 and 1) to our model, our model has to predict that digit. We count how much our model returns a true answer. This way, we calculate accuracy. If our accuracy is 0.94, that means out of 100 data, 94 are correctly predicted and 6 are wrong.

Finally  we have to predict digits drawn in paint. Our model is already trained, right? Now, we don't provide 20% images (testing images) to our model but we provide new images. We draw a digit in paint and at the same time, pass that digit to the model, our model has to predict that digit.

I am using an infinite loop so that I can test any number of inputs. If I press enter, it should close all the windows and stop my execution because I have passed cv2.waitkey(1)==13, and this 13 is the ASCII value of enter. But, I am using waitkey(10000) to display the same window for the long run. So, if I press enter, my execution does not halt. I have to press enter again and again (I think 4 or 5 times) and then only my execution gets stopped. I didn't get any solution regarding this. This is also nice, not. This is not a big deal to press enter 4 or 5 times but if you find any other solution, please let me know.

```
import joblib
import cv2
import numpy as np #pip install numpy
import time
import pyscreenshot as ImageGrab

model=joblib.load("model/digit_recognizer")
image_folder="img/"

while True:
   img=ImageGrab.grab(bbox=(60,170,400,500))

   img.save(images_folder+"img.png")
   im = cv2.imread(images_folder+"img.png")
   im_gray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
   im_gray  =cv2.GaussianBlur(im_gray, (15,15), 0)
   #Threshold the image
   ret, im_th = cv2.threshold(im_gray,100, 255, cv2.THRESH_BINARY)
   roi = cv2.resize(im_th, (28,28), interpolation  =cv2.INTER_AREA)

   rows,cols=roi.shape

   X = []

   ##  Fill the data array with pixels one by one.
   for i in range(rows):
       for j in range(cols):
           k = roi[i,j]
           if k>100:
               k=1
           else:
               k=0
           X.append(k)

   predictions  =model.predict([X])
   print("Prediction:",predictions[0])
   cv2.putText(im, "Prediction is: "+str(predictions[0]), (20,20), 0,
0.8,(0,255,0),2,cv2.LINE_AA)

   cv2.startWindowThread()
   cv2.namedWindow("Result")
   cv2.imshow("Result",im)
   cv2.waitKey(10000)
   if cv2.waitKey(1)==13: #27 is the ascii value of esc, 13 is the ascii value of
enter
       break
cv2.destroyAllWindows()
```