PROJECT REPORT

A NOVEL METHOD FOR HANDWRITTEN DIGIT RECOGNITION

PNT2022TMID35588

PRAVEEN R	**2019115070**
HRITHIK VIKNESH B	**2019115040**
AVINASH KRISHNA B	**2019115025**
KALAISELVAN S	**2019115045**

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

Automatic Handwriting recognition is a domain in which extensive research is being done. It is a challenging task since each individual has their own style of handwriting. In today's world, every aspect of all organizations' infrastructure are being digitized in a manner that aims to reduce human labour. This gives rise to the need to develop a system that can automatically recognize the handwritten digits from a given source such as image or video, in real-time. The MNIST dataset is a benchmark dataset that can be used to train Deep Learning models that learn to recognize handwritten digits from images. Various image processing techniques can be used to pre-process the image, perform noise removal, thresholding etc before feeding to the model for prediction. Instead of recognizing individual digits, it will be more beneficial if the system is able to isolate individual digits from a sequence of handwritten digits and recognize them.

1.2 PURPOSE

- To reduce the manual workload involved in recognizing handwritten digits by automating the process.
- To enable better utilization of human resources and time by automating the monotonous task such as handwritten digit recognition, so that those resources can be focused on more important tasks.
- To enable efficient sorting of postal delivery packages by sorting them based on automatic detection of the handwritten zip code on the package.
- To reduce time consumption by automating the process of recognizing handwritten information from bank cheques.
- To implement a robust handwritten digit recognition that can be embedded into the traffic surveillance system which enables monitoring of vehicles violating the traffic regulations, through the surveillance footage involving real-time and noisy data.

CHAPTER 2

LITERATURE SURVEY

2.1 EXISTING PROBLEM

Based on the literature survey conducted, a number of observations were made.

- Though multiple solutions are proposed to perform recognition of individual handwritten digits, there is a lack of any solution that extends the functionality to a sequence of handwritten digits.

- The performance of the existing solutions can be improved by using Image processing techniques to perform various transformations on the input image such as normalization, rescaling, binarization, thresholding, morphological opening and closing etc.

2.2 REFERENCES

[1] Ali Abdullah Yahya, Jieqing Tan and Min Hu. A Novel Handwritten Digit Classification System Based on Convolutional Neural Network Approach. Sensors 21(18):6273- 2021.
[2] Savita Ahlawat, Amit Choudhary, Anand Nayyar, Saurabh Singh and Byungun Yoon. Improved Handwritten Digit Recognition Using Convolutional Neural Networks (CNN), Sensors, 20(12): 3344 – 2020.
[3] Chao Zhang, Zhiyao Zhou and Lan Lin. Handwritten Digit Recognition Based on Convolutional Neural Network. 2020 Chinese Automation Congress (CAC), 2020
[4] Savita Ahlawat and Amit Choudhary. Hybrid CNN-SVM Classifier for Handwritten Digit Recognition. International Conference on Computational Intelligence and Data Science (ICCIDS 2019), 167 :2554–2560. 2019.
[5] Connor Shorten and Taghi M. Khoshgoftaar. A survey on Image Data Augmentation for Deep Learning, Journal of Big Data volume 6: 60. 2019.
[6] Saqib Ali, Zeeshan Shaukat, Muhammad Azeem, Zareen Sakhawat, Tariq Mahmood and Khalil ur Rehman. An efficient and improved scheme for handwritten digit recognition based on convolutional neural network. SN Applied Sciences volume 1: 1125 - 2019.
[7] Md Anwar Hossain, M. Ali. Recognition of Handwritten Digit using Convolutional Neural Network (CNN). Computer Science Global Journal of Computer Science and Technology. 19: 27–33- 2019.
[8] Shruti R. Kulkarni and Bipin Rajendran. Spiking neural networks for handwritten digit recognition—Supervised learning and network optimization. Neural Networks, Volume 103: 118-127- July 2018.

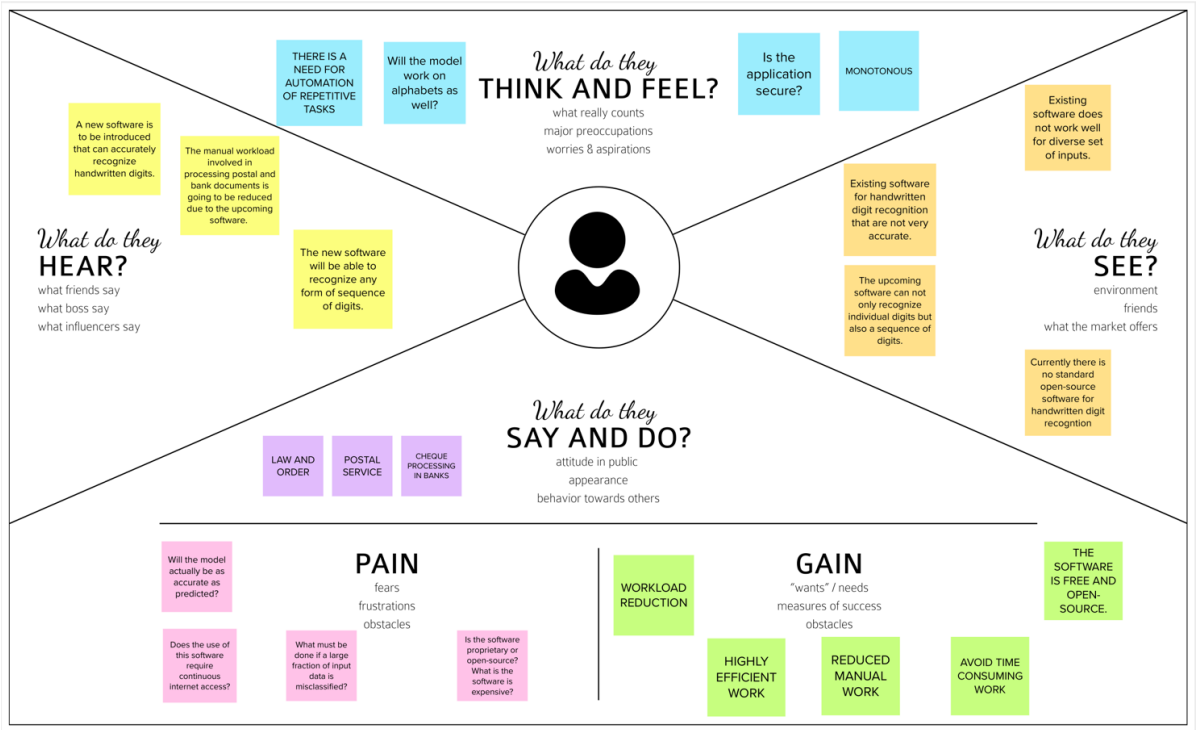
2.3 PROBLEM STATEMENT DEFINITION

Given an image containing a sequence of handwritten digits that has been uploaded by a user to the webpage, the aim is to identify the handwritten digit sequence by isolating and identifying the individual digits using Deep Learning and Image processing techniques.

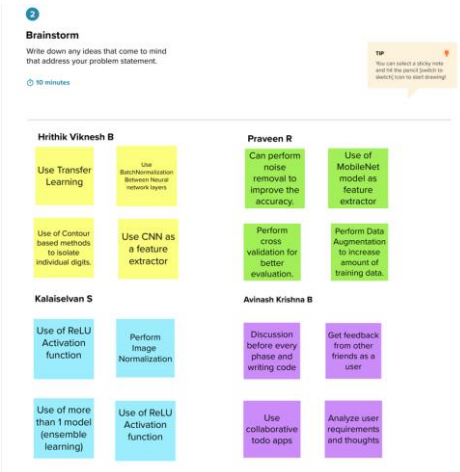
CHAPTER 3

IDEATION AND PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS



3.2 IDEATION AND BRAINSTORMING

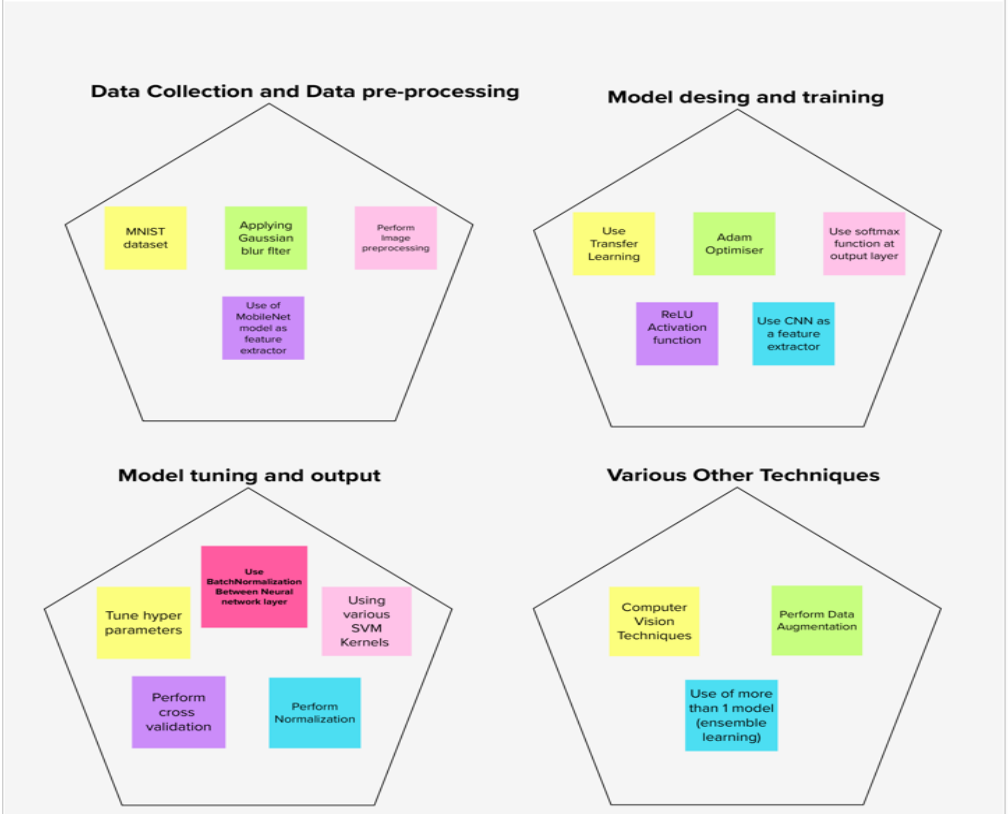


3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

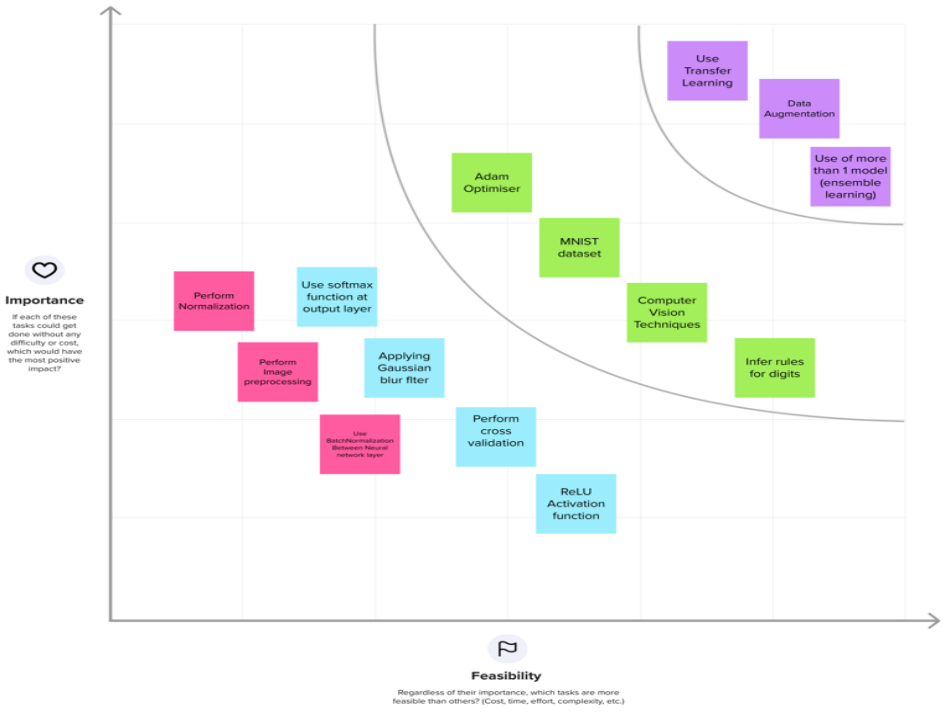
20 minutes



Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

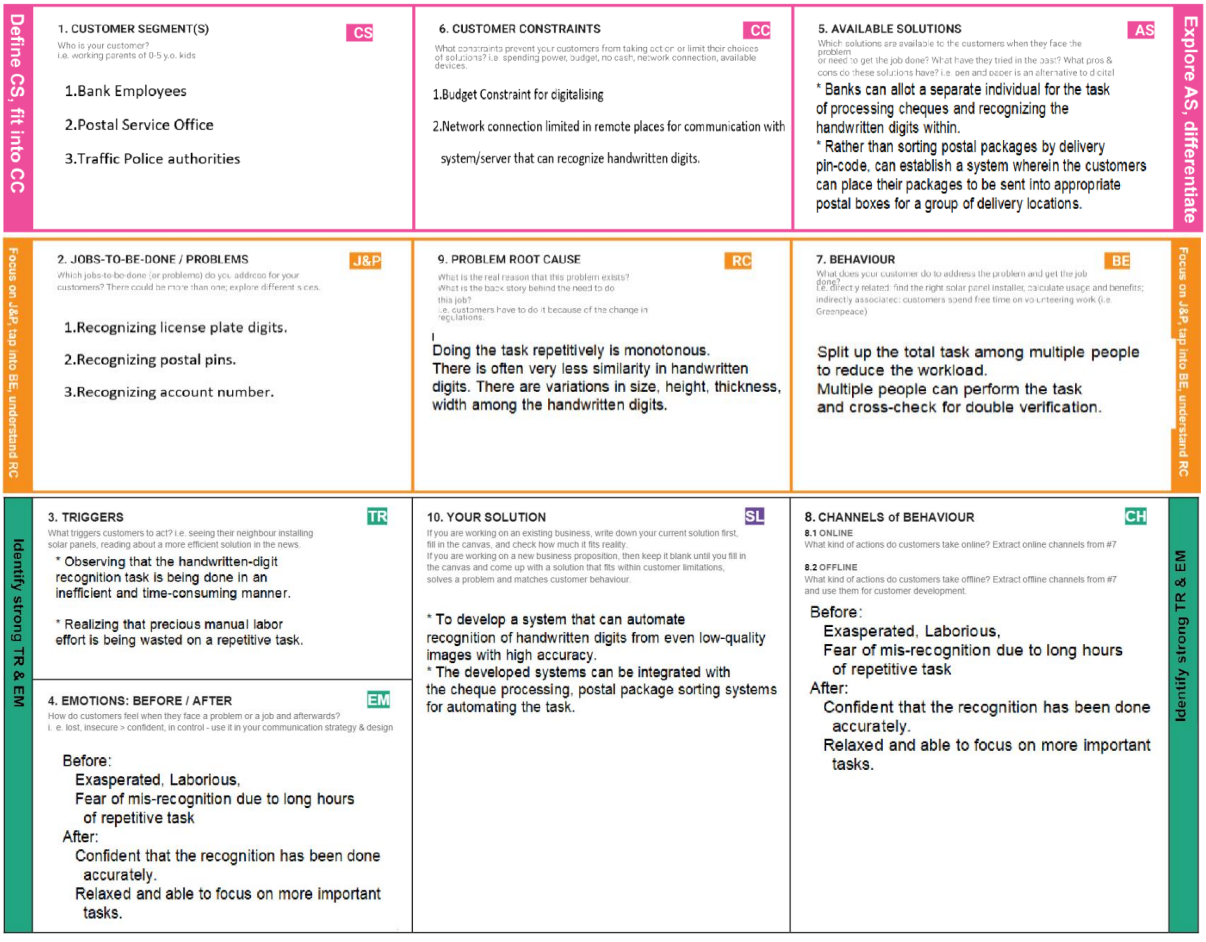


3.3 PROPOSED SOLUTION

S.No	Parameter	Description
1.	Problem Statement (Problem to be solved)	To develop a system that can automatically recognize a sequence of handwritten images from an image, using Computer Vision and Deep Learning techniques.
2.	Idea / Solution description	Using the benchmark MNIST dataset containing 60,000 images of handwritten digits, a Deep Learning model can be trained and tuned to a high level of accuracy. This model can be integrated with existing systems to make the business processes efficient.
3.	Novelty / Uniqueness	Besides recognizing individual digits from given images, our system will also be able to detect a sequence of handwritten digits, isolate the individual digits, and recognize each such digit. Instead of building and training a Deep Learning model from scratch, Transfer Learning can be used which enables the use of pre-trained, state of the art models which perform with a higher level of accuracy.
4.	Social Impact / Customer Satisfaction	The proposed system will provide a faster way to process cheques in banks wherein the handwritten digits can be recognized automatically. In postal offices, the delivery packages can be sorted based on the delivery address' zip code in an automatic manner by recognizing the handwritten zip code on the package, thus resulting in a reduction of manual workload.
5.	Business Model (Revenue Model)	In Banks, the developed system can be deployed as a software/web application which requires batches of cheque images to be uploaded and performs recognition of the handwritten digits within. In postal offices, the developed system can be integrated/embedded with a mechanical setup that can move/lift packages, thereby removing the need for human intervention.
6.	Scalability of the Solution	The proposed system can be developed cost-efficiently through various open-source software and standard benchmark datasets. It can also be made available in offline mode,

		thus eliminating the need for the system to be connected to the Internet. In the future, the system can be extended to handwritten characters as well, which would allow it to be used even in the Traffic Control and Monitoring system to monitor over-speeding or vehicles violating traffic rules, by identifying their License Plate Numbers through surveillance footage, thus resulting in a real-time monitoring system.

3.4 PROBLEM SOLUTION FIT



CHAPTER 4

REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENTS

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	Image upload	Upload image through web application Use of file system to store the uploaded image on server side
FR-2	Image validation & Preprocessing	Validating the give image format as in JPG and PNG. Preprocessing the image such as reshaping to standard size, Noise removal, Normalization.
FR-3	Sequence Detection & Digit Isolation	From entire area identify the area contain the sequence of digits From sequence isolation each digit
FR-4	Recognition	Classify the digit to one of the 10 digits use of deep learning technologies

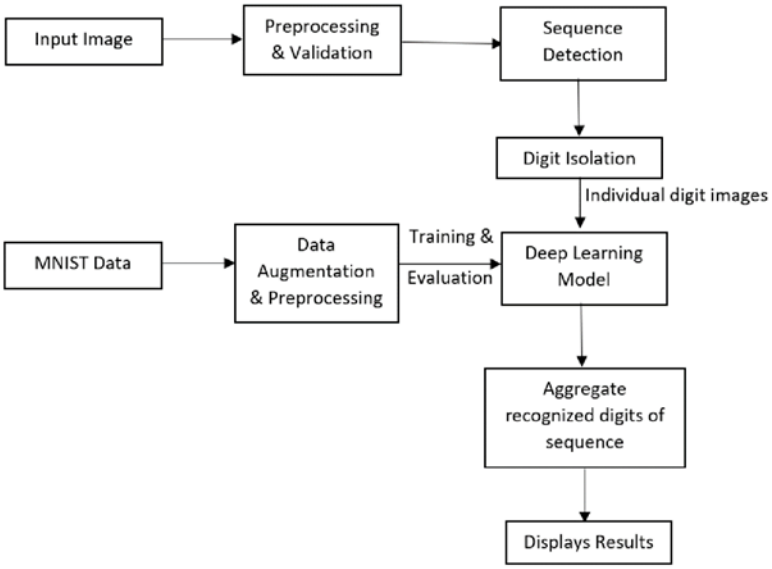
4.2 NON-FUNCTIONAL REQUIREMENTS

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Website is user friendly.
NFR-2	**Security**	The uploaded images are stored safely and is encrypted.
NFR-3	**Reliability**	It works for all kinds of inputs.
NFR-4	**Performance**	It can be also used for sequence of digits.
NFR-5	**Availability**	IBM cloud is used which ensures that the application is highly available across multiple regions through redundancy and maintaining replications.
NFR-6	**Scalability**	IBM cloud servers are used which can be scaled dynamically according to the demand, providing elasticity

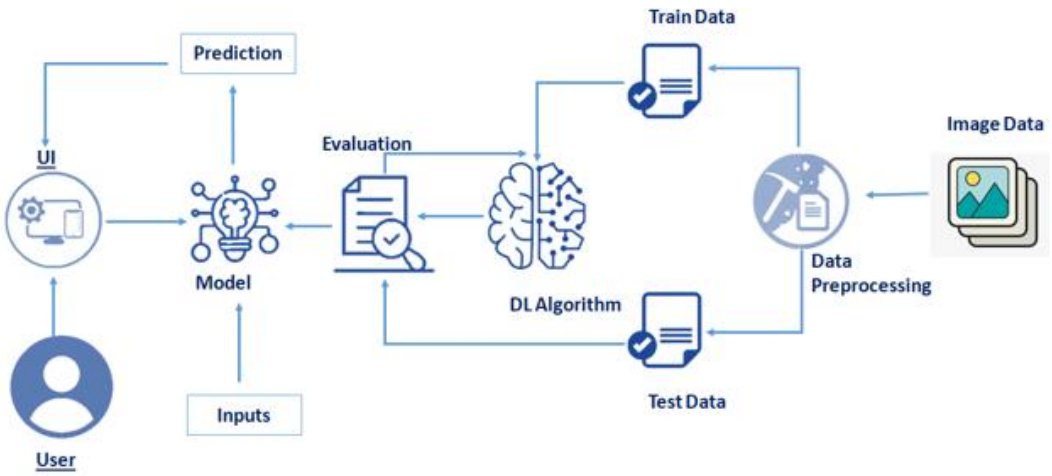
CHAPTER 5

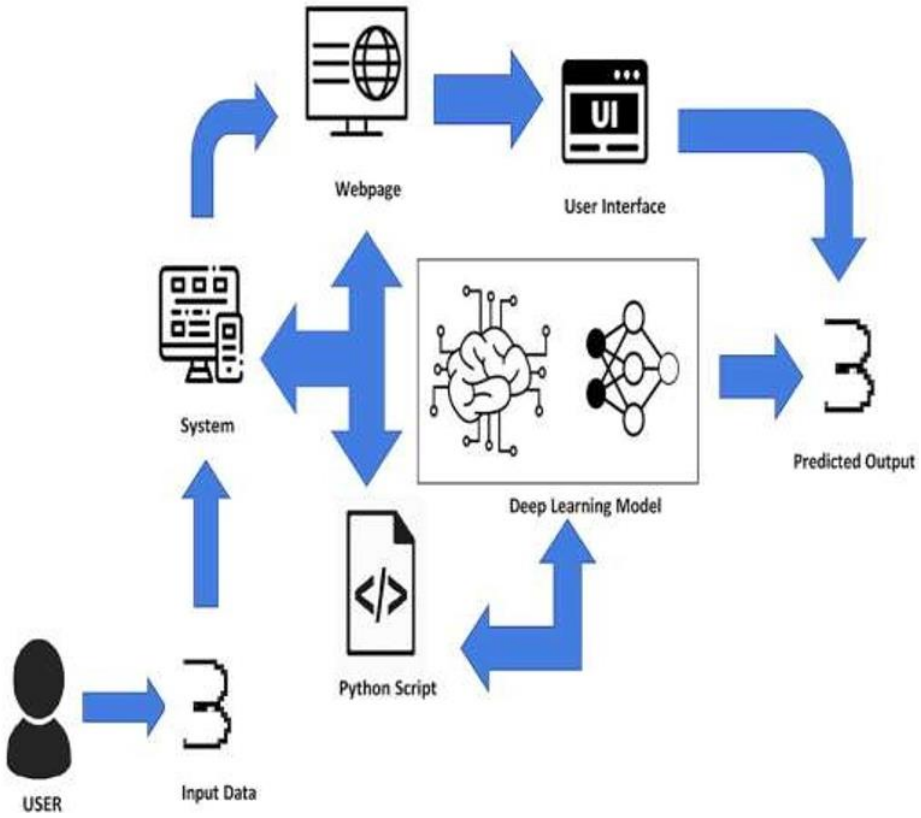
PROJECT DESIGN

5.1 DATA FLOW DIAGRAMS



5.2 SOLUTION AND TECHNICAL ARCHITECTURE





5.3 USER STORIES

User Type	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer	USN-1	I can use this Web App to do calculation for basic math like addition, subtraction etc.	I am getting the result	Medium	Sprint-1
	USN-2	I am a postman, I want to recognize the numbers in letters for delivery.	I can get the digital text to store it in computer memory	Medium	Sprint-1
	USN-3	I am bank employee,I want to recognize digits of cheque or challan and enter in computer	I can get the numbers from the cheque	Medium	Sprint-2
	USN-4	As a user, I can able to input the images of digital documents to the application	As a user, I can able to input the images of digital documents to the application	High	Sprint-2
	USN-5	As a user I can able to get the recognised digit as output from the images of digital documents or images	I can access the recognized digits from digital document or images	High	Sprint-3
	USN-6	As a user, I will train and test the input to get the maximum accuracy of output.	I can able to train and test the application until it gets maximum accuracy of the result.	Medium	Sprint-3
Customer (Web user)	USN-7	As a user, I can use the web application virtually anywhere.	I can use the application in any device with a browser	Medium	Sprint-4

CHAPTER 6

PROJECT PLANNING AND SCHEDULING

6.1 SPRINT PLANNING AND ESTIMATION

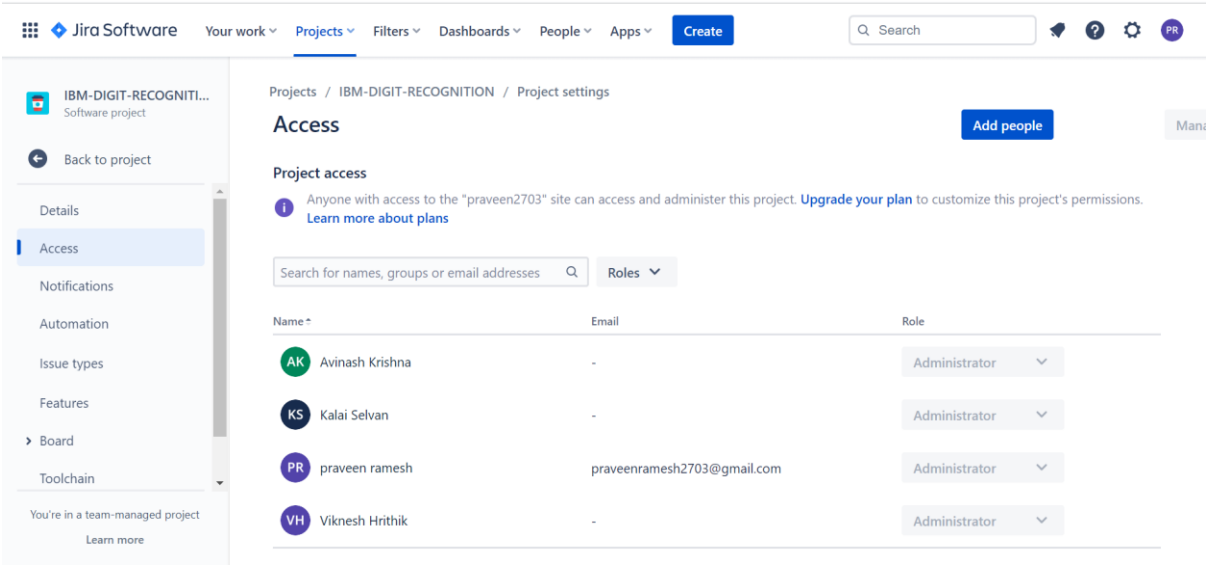
Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Data Collection	USN-1	Perform Data Collection from MNIST data of handwritten digits.	1	Medium	Praveen, Avinash Krishna
Sprint-1	Data Preprocessing	USN-2	Perform Data Preprocessing - Scaling, Noise Removal, Normalization, Data Augmentation.	2	High	Hrithik Viknesh, Kalaiselvan
Sprint-2	Model Building	USN-3	Build the model, Use Transfer Learning techniques.	2	High	Hrithik Viknesh, Praveen

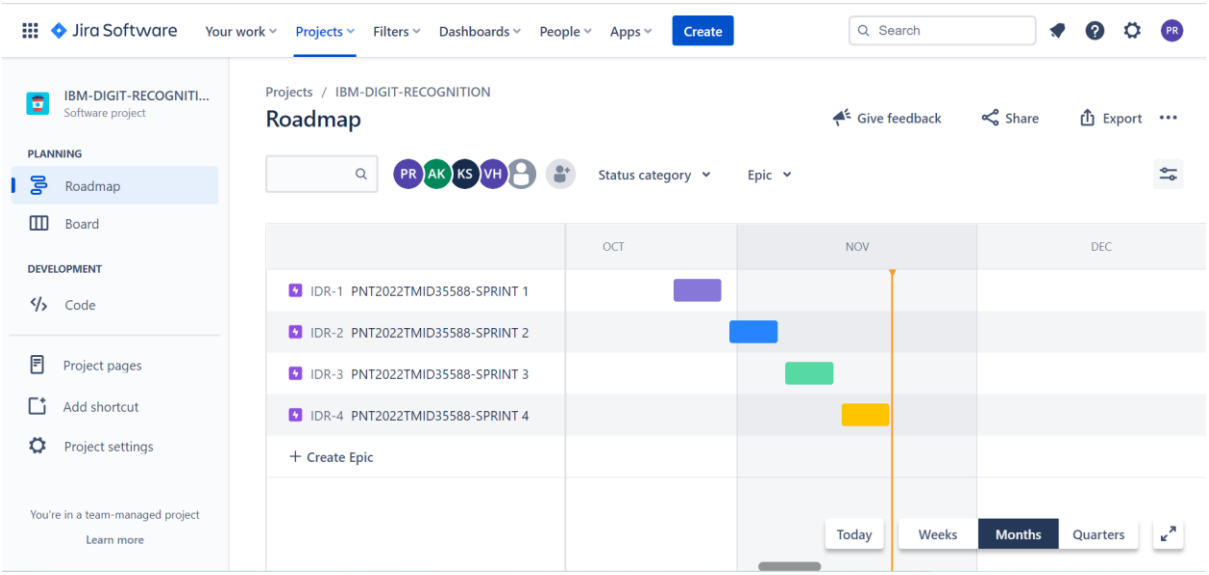
Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-2	Compiling Model	USN-4	Compile the model using appropriate loss function, metrics, optimizers and callbacks.	1	Medium	Kalaiselvan, Avinash Krishna
Sprint-2	Model Training & Validation	USN-5	Feed the data in batches for multiple epochs to the model, Save the model with best accuracy.	1	High	Praveen, Kalaiselvan
Sprint -3	Model Tuning	USN-6	Tune the model by either increasing or decreasing the model complexity, adding/removing one or more layers by observing the plots of loss and accuracy across epochs.	1	Medium	Hrithik Viknesh, Avinash Krishna
Sprint-3	Testing & Inference	USN-7	Evaluate model performance on test data, and perform classification of new data.	1	High	Kalaiselvan
Sprint -4	Implement for string of digits	USN-8	Implementation of Image Processing techniques to isolate individual digits from an image with a sequence of handwritten digits.	1	High	Hrithik Viknesh
Sprint -4	Build & Deploy the web app	USN-9	Deploy the web app in local/cloud environment, Implement Front-end and Back-end functionalities for the application.	2	Medium	Praveen, Kalaiselvan

6.2 SPRINT DELIVERY SCHEDULE

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	1,2	30 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	1,2	04 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	1	11 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	1,2	19 Nov 2022

6.3 REPORTS FROM JIRA





CHAPTER 7

CODING AND SOLUTIONING

7.1 PREPROCESSING AND NOISE REMOVAL

Image data, especially real-time data has a higher chance of containing noise within itself, which may not be visible to the naked eye. Better results can be achieved when performing prediction on the images, if the internal noise that exists is removed, and the image is smoothened and sharpened, along with other pre-processing steps. The input image is rescaled to the dimension (28,28) which is the the shape of the images in the benchmark dataset used and hence is the shape of the input image as expected by the Deep Learning model. In our system, various such pre-processing and morphological operations are performed such as

- Rescaling
- Normalization of pixel values
- Bitwise Inversion
- Applying Median Filter for noise removal
- Applying Gaussian Filter for noise removal.
- Image thresholding
- Morphological Opening (Erosion followed by dilation)

These pre-processing steps performed on the input image will lead to better accuracy of prediction.

```
f = request.files["image"]
filepath = (f.filename)
# Save the image
f.save(os.path.join(app.config['IMAGE_FOLDER'], filepath))

upload_img = os.path.join(IMAGE_FOLDER, filepath)

image = cv2.imread(upload_img)  # Read the image
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
gray = cv2.GaussianBlur(gray, (3, 3), 0)    # Noise Removal
edged = cv2.Canny(gray, 40, 120)    # Detect edges

# Isolate individual digits
cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2:]
cnts = grab_contours(cnts)

if cnts == ():                          # If no digits detected
    predictions = ['None']
    bbox_path = upload_img
else:
    cnts = sort_contours(cnts, method='left-to-right')[0]

    extracted_digits = []
    i = 0
    # Loop over the detected contours
    for c in cnts:
        # Compute bounding box of the contour
        (x, y, w, h) = cv2.boundingRect(c)
        cv2.rectangle(image, (x, y), (x+w, y+h), color=(0, 255, 0), thickness=2)
```

```
        # Filter extracted bounding boxes based on size criteria to avoid processing unwanted contours
        if (w >= 5 and w <= 150) and (h >= 15 and h <= 120):
            i += 1
            # Extract the character
            roi = gray[y:y + h, x:x + w]

            # Perform image inverting it to make the digit appear as "white" (foreground) on a "black" background
            thresh = cv2.bitwise_not(roi)

            # Resize the image to 28x28 pixels
            thresh = resize_image(thresh)

            # Noise removal before feeding to the model
            thresh = cv2.medianBlur(thresh, 3)
            thresh = cv2.GaussianBlur(thresh, (3,3), 0)
            thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)

            # Normalize pixel values between 0 and 1
            thresh = thresh / 255.0

            plt.imshow(thresh, cmap='gray')
            plt.savefig('static/images/digits/digit_' + str(i))

            # Add to list of extracted individual digits
            extracted_digits.append(thresh)

    # Plot uploaded image with bounding boxes around individual digits
    bbox_path = os.path.join(app.config['IMAGE_FOLDER'], 'bbox.png')
    plt.imshow(image, cmap = 'gray')
    plt.savefig(bbox_path)


    # Make predictions on the extracted digits and display the output
    predictions = []
    if len(extracted_digits) == 0:
        predictions = ['None']
        bbox_path = upload_img
    else:
        extracted_digits = np.array(extracted_digits)
        predictions = model.predict(extracted_digits)
        predictions = np.argmax(predictions, axis=1)
```

7.2 RECOGNIZING SEQUENCE OF HANDWRITTEN DIGITS

A system that recognizes a sequence of handwritten digits is far more beneficial than one that recognizes only a single digit. This requires that the sequence of digits be identified first and the individual digits be isolated and extracted. The user must upload the input image containing a sequence of handwritten digits through the application webpage. The extraction of individual digits is done using image processing techniques such as detecting the edges, observing the contours of the resultant image, identifying the contours of significant area, and extracting the image regions corresponding to those contours. The extracted digits are then fed to the trained Deep Learning model and the resultant predictions are aggregated. The aggregated digit predictions represent the handwritten digit sequence present in the input image. The predicted sequence is then displayed to the user through the webpage. Additionally, the input image with bounding boxes around the individual digits is also displayed to the user for better comprehension of the predictions**.**

```
# Building model
model4 = Sequential()

model4.add(Conv2D(128, kernel_size=3, activation='relu', input_shape=(28, 28, 1)))
model4.add(MaxPooling2D(pool_size=2))
model4.add(Conv2D(64, kernel_size=3, activation='relu'))
model4.add(MaxPooling2D(pool_size=2))
model4.add(Dropout(0.3))
model4.add(Flatten())
model4.add(Dense(256, activation="relu"))
model4.add(Dropout(0.3))
model4.add(Dense(10, activation='softmax'))

# Defining Callbacks
callbacks = [tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)]

# Compile the model
model4.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Fitting to the model
history4 = model4.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=15, callbacks=callbacks)
```

CHAPTER 8

TESTING

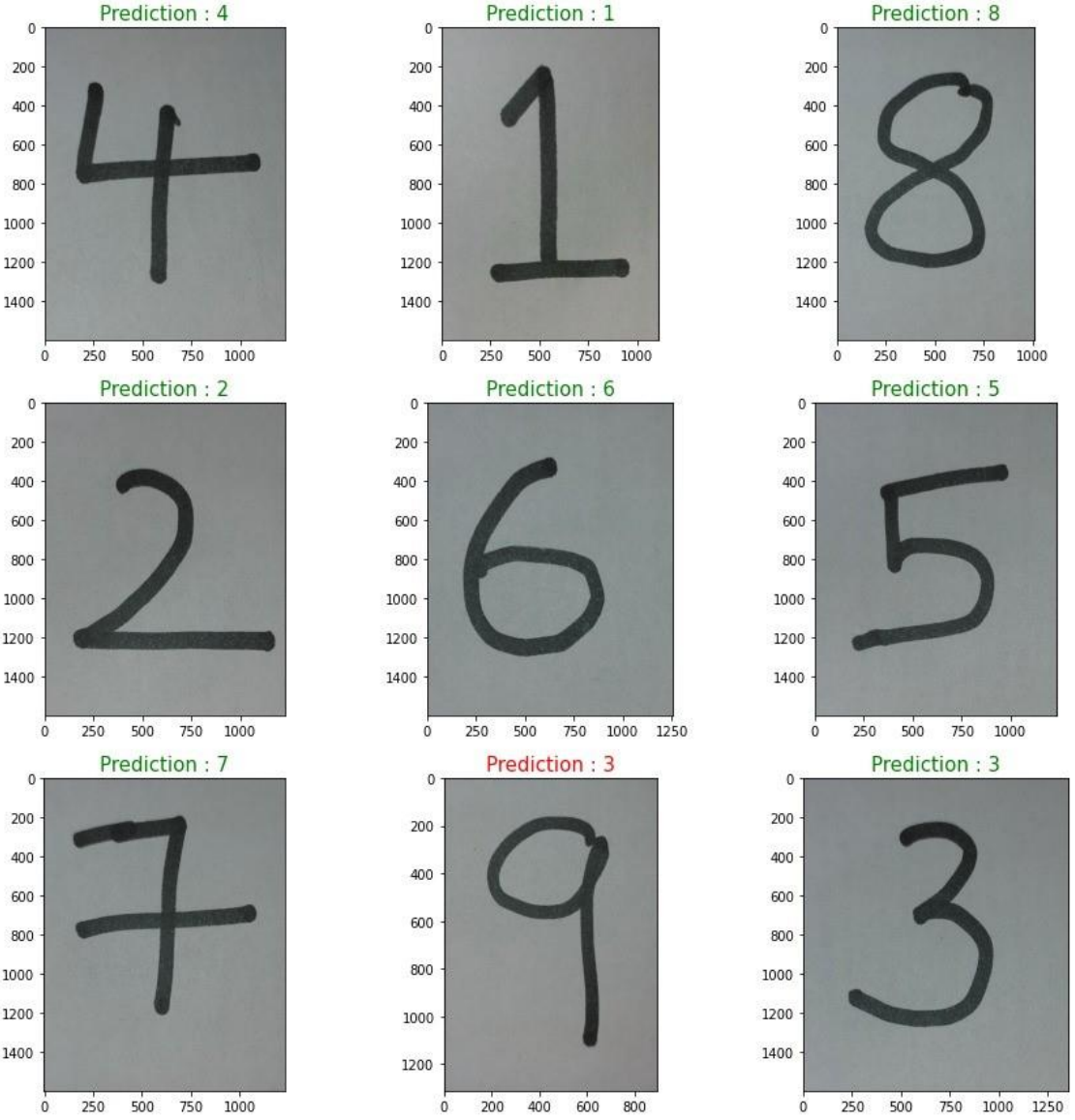
8.1 TEST CASES

Load the model

```
def predict_on_test_data(model_path, test_data_path):
  """ Uses the trained model to make predictions on test data and visualize them
  Args:
  model - path to the trained and saved model to be loaded
  test_data_path - path to folder containing test images to perform prediction on

  Returns:
  Plots the test images along with the predicted values which are returned
  """
  model = load_model(model_path)
  kernel = np.ones((1,1),np.uint8) # For morphological operations on the image

  images = []
  files = [test_data_path + '/' + f for f in os.listdir(test_data_path)]
  labels = [int(path.split('.')[0].split('_')[-1]) for path in files]

  for file in files:
      image = cv2.imread(file)
      # Convert to grayscale
      img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
      # Perform pre-processing and noise removal
      img = cv2.resize(img, (28, 28))
      img = cv2.bitwise_not(img)            # Convert foreground pixels to background pixels and vice-versa
      img = cv2.medianBlur(img, 3)
      img = cv2.GaussianBlur(img, (3,3), 0)
      img = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)       # Morphological Opening
      ret,img = cv2.threshold(img,127,255,cv2.THRESH_BINARY)    # Thresholding

      img = img / 255.0  # Normalize pixel values between 0 and 1
      images.append(img)

  # Convert list to batch of data
  images = np.array(images)
  # Use the model to make prediction
  preds = model.predict(images)
  pred = np.argmax(preds, axis=1)

  count = 1
  fig, axs = plt.subplots(3,3,figsize=(15,15))
  for i in range(3):
    for j in range(3):
      axs[i,j].imshow(cv2.imread(files[count-1]))
      if labels[count -1] == pred[count -1]:
        color = 'green'
      else:
        color = 'red'
      axs[i,j].set_title("Prediction : " +  str(pred[count-1]), color=color, fontsize=15)
      count += 1

  return labels, pred
```

```
MODEL_PATH = "/content/Digit_Recognition.h5"
TEST_DATA_PATH = "/content/test_cases/images"

labels, predictions = predict_on_test_data(MODEL_PATH, TEST_DATA_PATH)
```

```
1/1 [==============================] - 0s 115ms/step
```




```
[68] from sklearn.metrics import accuracy_score
```

```
print("Accuracy on the test cases : ", accuracy_score(labels, predictions))
```

```
Accuracy on the test cases :  0.8888888888888888
```

8.2 USER ACCEPTANCE TESTING

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	1	0	1	0	2
Duplicate	0	1	0	0	1
External	0	0	2	0	2
Fixed	4	1	0	0	5
Not Reproduced	0	0	1	1	2
Skipped	0	0	0	1	1
Won't Fix	1	0	0	0	1
Totals	6	2	4	2	14

Section	Total Cases	Not Tested	Fail	Pass
Client Application	9	0	1	8
Security	2	0	0	2
Exception Reporting	4	0	1	3
Performance	4	0	0	4

CHAPTER 9

RESULTS

9.1 PERFORMANCE METRICS

Performance Metrics

Accuracy, Precision, Recall Score are the appropriate tasks for the prediction task.

```
[11] test_predictions = np.argmax(model.predict(X_test), axis=1)

     print(len(test_predictions))
```

```
313/313 [==============================] - 1s 2ms/step
10000
```

```
[12] test_labels = np.argmax(y_test, axis = 1)
```

```
[18] import sklearn
     from sklearn.metrics import accuracy_score
     from sklearn.metrics import precision_score
     from sklearn.metrics import recall_score
     from sklearn.metrics import f1_score
     from sklearn.metrics import classification_report
     from sklearn.metrics import confusion_matrix
```

Accuracy

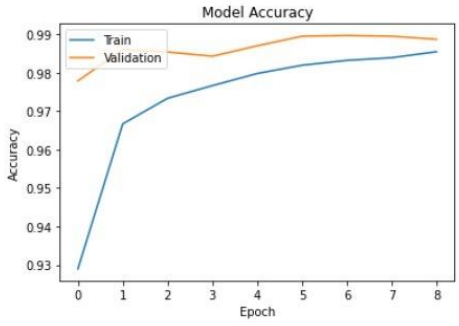
```
[ ] print(accuracy_score(test_labels, test_predictions))
```

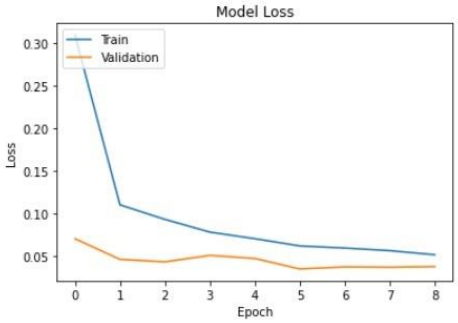
```
0.9906
```

Recall

```
[ ] print(recall_score(test_labels, test_predictions, average='weighted'))
```

```
0.9906
```





F1 score

```
[ ] print(f1_score(test_labels, test_predictions, average='weighted'))
```

```
0.9906029544489391
```

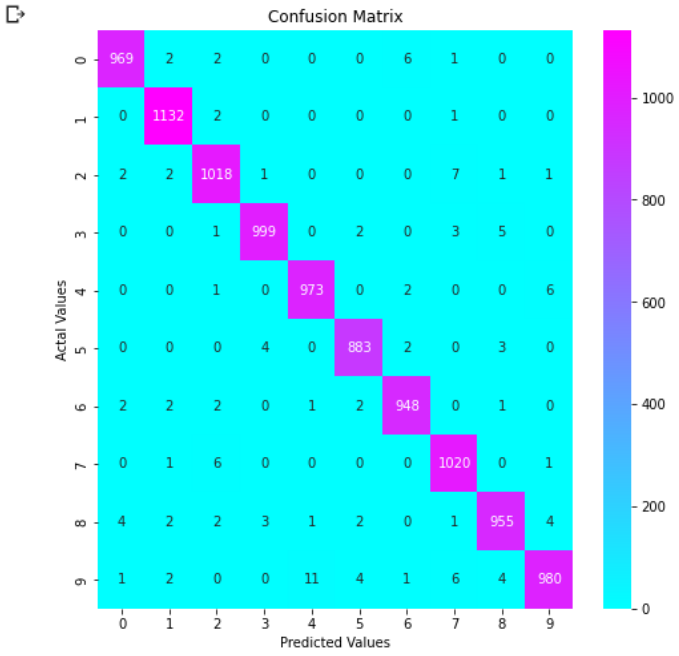
Classification Report

```
[ ] print(classification_report(test_labels, test_predictions))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       1.00      0.99      0.99      1135
           2       0.99      0.99      0.99      1032
           3       1.00      0.99      0.99      1010
           4       0.99      0.99      0.99       982
           5       0.99      0.99      0.99       892
           6       0.99      0.99      0.99       958
           7       0.98      0.99      0.99      1028
           8       0.98      1.00      0.99       974
           9       0.99      0.98      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

Plotting the confusion matrix

```
plt.figure(figsize=(8,8))
sns.heatmap(cm_df, annot=True, fmt='d', cmap='cool')
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



CHAPTER 10

ADVANTAGES AND DISADVANTAGES

ADVANTAGES

- Since data augmentation is performed on the available dataset, the application performs well on a diverse range of input data making the model more robust.
- Extensive noise removal and smoothing is performed before feeding the image to the model; hence the application performs well even on noisy data.
- Along with the predicted digits, bounding boxes are displayed around the individual digits as an image to the user.

DISDVANTAGES

- The application does not perform well on sequences containing partially/fully overlapping digits.
- Since the model has been trained on a standard dataset with images of a fixed shape, the application's performance might be affected when rescaling the image to the same fixed shape.

CHAPTER 11

CONCLUSION

In this project, developed and demonstrated a web application that uses Deep Learning and Image Processing techniques to recognize a sequence of handwritten digits. Technologies such as Flask, HTML, CSS, JavaScript, OpenCV, TensorFlow, Scikit-Learn were used in the development of the application. A Convolutional Neural Network model is trained and tuned to detect handwritten digits from images. Image Processing techniques are used to isolate the individual digits which are then fed to the Deep Learning model. During testing, the model achieved an accuracy of 99.57%. The proposed project is developed in a generic, robust and scalable manner and can handle a large number of users. Since the proposed system is developed as a web application, it is compatible with any device that has the capability to run a browser software. This project is highly beneficial in multiple real-time scenarios such as Efficient Sorting of postal delivery packages by automatically identifying the handwritten zip code, quick processing of bank cheques, monitoring vehicles violating traffic regulations by recognizing the vehicle identification numbers through the license plates from the traffic surveillance footage. There are multiple scopes for improving the developed system, which can be implemented as part of subsequent versions.

CHAPTER 12

FUTURE SCOPE

Currently, the application works for handwritten digit data. If this functionality is extended to handwritten mathematical operations and symbols, the application can be developed into a Mathematical Equation Solver that takes in an image of a handwritten mathematical equation as input and solves it. Moreover, the handwritten digit recognition system can be embedded into an IoT component that has access to a camera feed and performs automatic recognition of the handwritten digits from any source such as postal delivery packages, bank cheques, license plate images from traffic surveillance footage. Several other augmentations can be performed on the available data to make the model more robust.

CHAPTER 13

APPENDIX

SOURCE CODE

Model_Tuning_And_Performance_Metrics.ipynb

Import Libraries

```
import cv2
import pandas as pd
import numpy as np
import seaborn as sns
import tensorflow as tf
from keras.datasets import mnist
from keras.layers import Dense, Flatten, MaxPooling2D, Dropout
from keras.layers.convolutional import Conv2D
from keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import load_model
import matplotlib.pyplot as plt
```

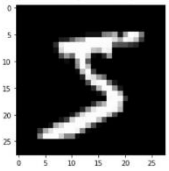
Loading data

```
[ ] (X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
```

```
[ ] plt.imshow(X_train[0], cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7fa2e68a21d0>
```



Reshaping data to feed to the model

```
[ ] X_train = X_train.reshape(60000, 28, 28, 1)
    X_test = X_test.reshape(10000, 28, 28, 1)
```

```
[ ] print ("Shape of X_train: {}".format(X_train.shape))
    print ("Shape of y_train: {}".format(y_train.shape))
    print ("Shape of X_test: {}".format(X_test.shape))
    print ("Shape of y_test: {}".format(y_test.shape))
```

```
Shape of X_train: (60000, 28, 28, 1)
Shape of y_train: (60000,)
Shape of X_test: (10000, 28, 28, 1)
Shape of y_test: (10000,)
```

One-Hot encoding of labels

```
[ ] y_train = to_categorical(y_train)
    y_test = to_categorical(y_test)
```

Model Building and Training

```
# Building model
model4 = Sequential()

model4.add(Conv2D(128, kernel_size=3, activation='relu', input_shape=(28, 28, 1)))
model4.add(MaxPooling2D(pool_size=2))
model4.add(Conv2D(64, kernel_size=3, activation='relu'))
model4.add(MaxPooling2D(pool_size=2))
model4.add(Dropout(0.3))
model4.add(Flatten())
model4.add(Dense(256, activation="relu"))
model4.add(Dropout(0.3))
model4.add(Dense(10, activation='softmax'))

# Defining Callbacks
callbacks = [tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)]

# Compile the model
model4.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Fitting to the model
history4 = model4.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=15, callbacks=callbacks)
```

```
Epoch 1/15
1875/1875 [==============================] - 16s 4ms/step - loss: 0.0854 - accuracy: 0.9274 - val_loss: 0.0640 - val_accuracy: 0.9785
Epoch 2/15
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1113 - accuracy: 0.9676 - val_loss: 0.0593 - val_accuracy: 0.9806
Epoch 3/15
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0910 - accuracy: 0.9726 - val_loss: 0.0403 - val_accuracy: 0.9880
Epoch 4/15
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0798 - accuracy: 0.9770 - val_loss: 0.0388 - val_accuracy: 0.9877
Epoch 5/15
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0716 - accuracy: 0.9788 - val_loss: 0.0393 - val_accuracy: 0.9873
Epoch 6/15
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0623 - accuracy: 0.9809 - val_loss: 0.0428 - val_accuracy: 0.9872
Epoch 7/15
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0599 - accuracy: 0.9822 - val_loss: 0.0395 - val_accuracy: 0.9877
```

This model achieves the highest validation accuracy. So we save and use this model for the prediction purpose.

```
[ ] model4.save("Digit_Recognition.h5")
```

Performance Metrics

Accuracy, Precision, Recall Score are the appropriate tasks for the prediction task.

```
[ ] test_predictions = np.argmax(model.predict(X_test), axis=1)

    print(len(test_predictions))
```

```
313/313 [==============================] - 1s 2ms/step
10000
```

```
[ ] test_labels = np.argmax(y_test, axis = 1)
```

```
[ ] import sklearn
    from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report,confusion_matrix
```

Accuracy

```
print(accuracy_score(test_labels, test_predictions))
```

```
0.9906
```

Recall

```
[ ] print(recall_score(test_labels, test_predictions, average='weighted'))
```

```
0.9906
```

F1 score

```
[ ] print(f1_score(test_labels, test_predictions, average='weighted'))
```

```
0.9906029544489391
```

Classification Report

```
[ ] print(classification_report(test_labels, test_predictions))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       1.00      0.99      0.99      1135
           2       0.99      0.99      0.99      1032
           3       1.00      0.99      0.99      1010
           4       0.99      0.99      0.99       982
           5       0.99      0.99      0.99       892
           6       0.99      0.99      0.99       958
           7       0.98      0.99      0.99      1028
           8       0.98      1.00      0.99       974
           9       0.99      0.98      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

Confusion Matrix

```
[ ] cm = confusion_matrix(test_labels, test_predictions)
```

```
[ ] cm_df = pd.DataFrame(cm, index = [i for i in range(10)], columns = [i for i in range(10)])
```

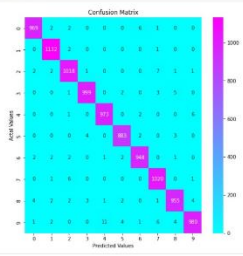
Confusion Matrix

```
[ ] cm = confusion_matrix(test_labels, test_predictions)
```

```
[ ] cm_df = pd.DataFrame(cm, index = [i for i in range(10)], columns = [i for i in range(10)])
```

Plotting the confusion matrix

```
[ ] plt.figure(figsize=(9,8))
    sns.heatmap(cm_df, annot=True, fmt='d', cmap='cool')
    plt.title('Confusion Matrix')
    plt.ylabel('Actual Values')
    plt.xlabel('Predicted Values')
    plt.show()
```



App.py

```python
import logging
import os
import pickle

import cv2
import sklearn
import matplotlib
import numpy as np
matplotlib.use('Agg')
import tensorflow
import tensorflow_hub as hub
from flask import Flask, render_template, request, send_from_directory, url_for
from PIL import Image
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from utils import grab_contours, sort_contours, label_contour, resize_image
import matplotlib.pyplot as plt

plt.axis("off")

# Kernel for OpenCV morphological operations
kernel = np.ones((1,1),np.uint8)

# Disable Tensorflow warnings
logging.disable(logging.WARNING)
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"

# Folder to store images
```

```python
IMAGE_FOLDER = 'static/images'

app = Flask(__name__)
app.config['IMAGE_FOLDER'] = IMAGE_FOLDER

# Set Tensorflow to use CPU
tensorflow.config.set_visible_devices([], 'GPU')

# Load the model
model = load_model("model/model.h5")

# Home Page
@app.route('/')
def index():
    return render_template('index.html')

# Prediction page
@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == "POST":
        f = request.files["image"]
        filepath = (f.filename)
        # Save the image
        f.save(os.path.join(app.config['IMAGE_FOLDER'], filepath))

        upload_img = os.path.join(IMAGE_FOLDER, filepath)

        image = cv2.imread(upload_img)  # Read the image
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
        gray = cv2.GaussianBlur(gray, (3, 3), 0)    # Noise Removal
```

```python
        edged = cv2.Canny(gray, 40, 120)    # Detect edges

        # Isolate individual digits
        cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2:]
        cnts = grab_contours(cnts)

        if cnts == ():                      # If no digits detected
            predictions = ['None']
            bbox_path = upload_img
        else:
            cnts = sort_contours(cnts, method='left-to-right')[0]

            extracted_digits = []
            i = 0
            # Loop over the detected contours
            for c in cnts:
                # Compute bounding box of the contour
                (x, y, w, h) = cv2.boundingRect(c)
                cv2.rectangle(image, (x, y), (x+w, y+h), color=(0, 255, 0), thickness=2)

                # Filter extracted bounding boxes based on size criteria to avoid processing unwanted contours
                if (w >= 5 and w <= 150) and (h >= 15 and h <= 120):
                    i += 1
                    # Extract the character
                    roi = gray[y:y + h, x:x + w]

                    # Perform image inverting it to make the digit appear as "white" (foreground) on a "black" background
                    thresh = cv2.bitwise_not(roi)

                    # Resize the image to 28x28 pixels
```

```python
                    thresh = resize_image(thresh)

                    # Noise removal before feeding to the model
                    thresh = cv2.medianBlur(thresh, 3)
                    thresh = cv2.GaussianBlur(thresh, (3,3), 0)
                    thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)

                    # Normalize pixel values between 0 and 1
                    thresh = thresh / 255.0

                    plt.imshow(thresh, cmap='gray')
                    plt.savefig('static/images/digits/digit_' + str(i))

                    # Add to list of extracted individual digits
                    extracted_digits.append(thresh)

            # Plot uploaded image with bounding boxes around individual digits
            bbox_path = os.path.join(app.config['IMAGE_FOLDER'], 'bbox.png')
            plt.imshow(image, cmap = 'gray')
            plt.savefig(bbox_path)


            # Make predictions on the extracted digits and display the output
            predictions = []
            if len(extracted_digits) == 0:
                predictions = ['None']
                bbox_path = upload_img
            else:
                extracted_digits = np.array(extracted_digits)
                predictions = model.predict(extracted_digits)
```

```
                predictions = np.argmax(predictions, axis=1)

        return render_template('predict.html', num=predictions, img_path = bbox_path)


if __name__ == '__main__':
    app.run(debug=True, threaded=False)
```

Index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <title>HandWritten Digit Recognition WebApp</title>

  <meta name="viewport" content="width=device-width">

  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" 
  <link rel="stylesheet" type= "text/css" href= "{{ url_for('static',filename='css/style.css') }}">
</style>

</head>

<script>
  function preview() {
    frame.src=URL.createObjectURL(event.target.files[0]);
}
    //When clear button clicked
    $(document).ready(function() {
         $('#clear_button').on('click', function() {
             $('#image').val('');
             $('#frame').attr('src','../static/images/preview.jpeg");
           });
       });

</script>

<body>
  <section align='center'>
  <h1 class="welcome">Handwritten Digit Recognition
  <div align="'center" id="team_id">TEAM ID :PNT2022TMID35588 </div>
  </h1>
  </section>


  <section id="title">
    <h4 class="heading">Handwritten Digit Recognition Web Application</h4>
    <br><br>
      <p>
        The website is designed to predict the handwritten digit.
      </p>
    <p>
      Handwriting recognition is one of the compelling research works going on because every individual in this world has their own
    </p>

      <br>

  </section>

  <section id="content" align='center'>

        <div class="leftside">
        <form action="/predict" method="POST" enctype="multipart/form-data">
        <label>Select a image:</label>
        <input label='image_input' id="image" type="file" name="image" accept="image/png, image/jpeg" onchange="preview()"><br><br>
          <img id="frame" src="../static/images/preview.jpeg" alt="Uploaded image" width="200px" height="200px"/>
          <div class="buttons_div">
            <button type="submit" class="btn btn-dark" id="predict_button">Predict Number</button>
            <button type="button" class="btn btn-dark" id="clear_button">  Clear  </button>
          </div>
        </form>
        </div>
  </section>

</body>

</html>
```

Predict.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Handwritten Digit Recognition</title>
</head>

<style>
    body{
     background-image: url('images/images.png');
     background-repeat: no-repeat;
     background-size: cover;
    }

    #rectangle{
     width:400px;
     height:150px;
     background-color: #5796a5;
     opacity: 0.8;
     border-radius: 25px;
     position:absolute;
     top:25%;
     left:50%;
     transform:translate(-50%,-50%);
    }

    #ans{
   text-align: center;
   font-size: 40px;
   margin: 0 auto;
   margin: 0 auto;
   padding: 3% 5%;
   padding-top: 9%;
   color: white;
    }

    span{
        color: white;
        font-size: 40px;
        text-align: center;
    }

</style>
<body>
    <div id="bbox_image_div" style="display:relative">
        <img id="bbox_image" src="{{ img_path }}" alt="Uploaded Image" height="300" width="300">
    </div>
    <div id="rectangle" style="display:relative">
        <h1 id="ans">Prediction : <br/>
        {% if num[0] == 'None' %}
            <span>No digits detected</span>
        {% else %}
            {%for digit in num%}
            <span><b>{{digit}} </b></span>
            {%endfor%}
        {% endif %}
        </h1>
    </div>
</body>
</html>
```

Utils.py

```python
import cv2
import numpy as np


def grab_contours(cnts):
    # if the length the contours tuple returned by cv2.findContours
    # is '2' then we are using either OpenCV v2.4, v4-beta, or
    # v4-official
    if len(cnts) == 2:
        cnts = cnts[0]

    # if the length of the contours tuple is '3' then we are using
    # either OpenCV v3, v4-pre, or v4-alpha
    elif len(cnts) == 3:
        cnts = cnts[1]

    else:
        raise Exception(("Contours tuple must have length 2 or 3"))

    # return the actual contours array
    return cnts


def sort_contours(cnts, method="left-to-right"):
    # initialize the reverse flag and sort index
    reverse = False
    i = 0

    # handle to sort in reverse
    if method == "right-to-left" or method == "bottom-to-top":
        reverse = True

    # handle for sorting against the y-coordinate rather than the x-coordinate of the bounding box
    if method == "top-to-bottom" or method == "bottom-to-top":
        i = 1

    # construct the list of bounding boxes and sort them from top to bottom
    boundingBoxes = [cv2.boundingRect(c) for c in cnts]
    (cnts, boundingBoxes) = zip(*sorted(zip(cnts, boundingBoxes),
                                        key=lambda b: b[1][i], reverse=reverse))

    # return the list of sorted contours and bounding boxes
    return cnts, boundingBoxes


def resize_image(img, size=(28,28)):

    h, w = img.shape[:2]
    c = img.shape[2] if len(img.shape)>2 else 1

    if h == w:
        return cv2.resize(img, size, cv2.INTER_AREA)

    dif = h if h > w else w

    interpolation = cv2.INTER_AREA if dif > (size[0]+size[1])//2 else cv2.INTER_CUBIC

    x_pos = (dif - w)//2
    y_pos = (dif - h)//2

    if len(img.shape) == 2:
        mask = np.zeros((dif, dif), dtype=img.dtype)
        mask[y_pos:y_pos+h, x_pos:x_pos+w] = img[:h, :w]
    else:
        mask = np.zeros((dif, dif, c), dtype=img.dtype)
        mask[y_pos:y_pos+h, x_pos:x_pos+w, :] = img[:h, :w, :]

    return cv2.resize(mask, size, interpolation)

def label_contour(image, conts, color=(0, 255, 0), thickness=2):
    # compute the center of the contour area and draw a circle
    # representing the center
    print(type(conts))
    print("------------------")
    print(conts)
    print("------------------")
    for i in range(len(conts)):
        c = conts[i]
        M = cv2.moments(c)
        cX = int(M["m10"] / M["m00"])
        cY = int(M["m01"] / M["m00"])

        # draw the contour and label number on the image
        cv2.drawContours(image, [c], -1, color, thickness)
        cv2.putText(image, " ", (cX - 20, cY), cv2.FONT_HERSHEY_SIMPLEX,
                    1.0, (255, 255, 255), 2)

    # return the image with the contour number drawn on it
    return image
```

GITHUB

https://github.com/IBM-EPBL/IBM-Project-10449-1659180179

PROJECT DEMO

https://drive.google.com/file/d/18t0gdWjJxrs5rDAGx1PjMQ22dYxAVwgE/view?usp=share_link