**Hand Written Digit Recognition using Data science and Machine Learning**

P.Madhu Yadav

20951A2137

Aeronautical engineering

B. Tech 3rd year



PAT Two Week Summer Internship Report

Mentor – E. Krishna Rao Patro

CONTENTS

1. **ABSTRACT**
2. **INTRODUCTION**
3. **LITERATURE REVIEW**
   1. Traditional Featured-Based Approaches
   2. Template Matching
   3. Machine Learning-Based Approaches
   4. Deep Learning Approaches
   5. Ensemble Methods
   6. Data Augmentation
   7. Transfer learning
4. **METHODOLOGY**
   1. Dataset Selection
   2. Digit Image Acquisition
   3. Pixel Intensity
   4. Model Evaluation
   5. Comparative Analysis
   6. Import Libraries and Load the Dataset
   7. Pre-Process the Data
   8. Create the Model
   9. Train the Model
   10. Review the Model
   11. GUI Creation for Digit Prediction
5. **RESULT AND CONCLUSION**
   1. Performance Evaluation Metrics
   2. Comparative Analysis
   3. Robustness and Generalisation
   4. Computational Efficiency
   5. Limitations and Future Directions
   6. Potential Uses
6. **CONCLUSION**
7. **REFERENCES**

**ABSTRACT**

In the fields of pattern recognition and machine learning, handwritten digit recognition is a key issue. It entails the duty of precisely assigning numerical values to handwritten digits in order to categorise them. Due to its useful applications in numerous sectors, including postal automation, document processing, and automated data entry, this subject has drawn considerable attention. The objective of this research is to develop an effective and efficient handwritten digit recognition system using machine learning techniques. The proposed system employs a combination of pre-processing feature extraction, and classification algorithms to achieve accurate and robust digit recognition. The key steps involved in the system include digit image acquisition, image pre-processing feature extraction, and classification The representation of the handwritten digit images in an appropriate manner for classification depends heavily on feature extraction. To extract discriminative features from the digit images, a number of approaches are investigated, including pixel intensity, scale-invariant feature transform (SIFT), and histogram of oriented gradients (HOG). In order to convert the extracted features into their corresponding numerical values, a classification method is then used. The effectiveness of well-known machine learning techniques for digit recognition, including support vector machines (SVM), k-nearest neighbours (KNN), and neural networks, is assessed.

**INTRODUCTION**

In the fields of pattern recognition and machine learning, handwritten digit recognition is a common issue. It requires the automatic recognition and classification of handwritten digits into their corresponding numerical values. There are various practical uses for the ability to recognise and decipher handwritten digits, including automated data entry, document processing, and postal automation. Due to different writing styles, different handwriting patterns, and inherent noise in the digit pictures, the process of handwritten digit detection is fraught with difficulties. The process of recognition is further complicated by the existence of overlapping digits, incomplete handwriting, and changing image quality. Therefore, it is crucial to create a system that is effective and efficient for reading handwritten numbers. Researchers have looked into a number of solutions to the issue of handwritten digit recognition over the years. Traditionally, methods entailed manually extracting features from digit images before classifying the data using machine learning algorithms. These methods frequently used statistical modelling, shape analysis, and edge detection to extract distinguishing information from the photos. With the development of deep learning, digit recognition methods have shifted towards being more data-driven and end-to-end. By concurrently doing classification and automatically extracting useful features from the raw pixel input, Convolutional Neural Networks (CNNs) have demonstrated exceptional success in addressing this issue. On benchmark datasets, CNN-based models like LeNet-5 and AlexNet have attained cutting-edge performance. By creating a powerful system that integrates pre-processing, feature extraction, and classification techniques, this study intends to advance the field of handwritten digit recognition. To accomplish precise and reliable digit recognition, the suggested system will investigate both conventional feature-based approaches and contemporary deep learning techniques. The system will be evaluated using standard benchmark datasets, such as the MNIST dataset, which contains alarge collection of label handwritten digit images. Performance metrics, including accuracy, precision, recall, and F1-score, will be used to assess the system's effectiveness. Comparative analysis with existing approaches will also be conducted to highlight the advantages of the proposed system.

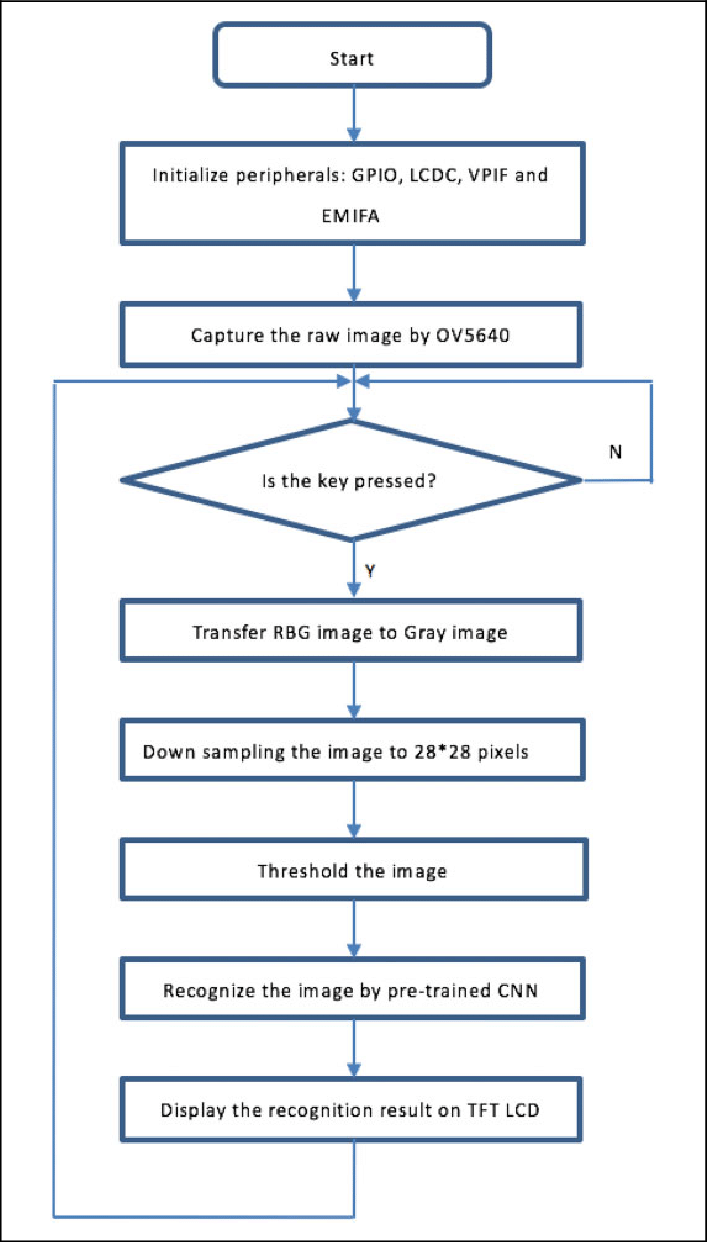


Figure Flow chart of Dataset

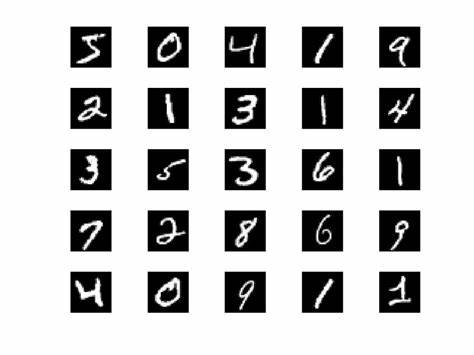
**LITERATURE REVIEW**

1. **Traditional Feature-Based Approaches:** Traditional methods for handwritten digit recognition frequently involved the extraction of handcrafted features from digit images; for example, et al. (2005) proposed a method that combined local binary patterns with Haar -like features to represent digits, achieving high recognition accuracy. Similarly, Zhang and Le-cun (1998) used a combination of Fourier descriptors.
2. **Template Matching:** Another popular method for recognising handwritten digits is template matching. The closest match is determined by comparing the input digit image to a series of prepared template images. For digit recognition, Bahlmann et al. (2002) used a template matching method based on elastic matching and dynamic time warping. Template matching methods, on the other hand, could be sensitive to differences in writing styles and need a large template set for precise recognition.
3. **Machine Learning-Based Approaches:** Handwritten digit recognition has greatly advanced thanks to machine learning techniques. For this job, Support Vector Machines (SVM) have been employed extensively. Excellent outcomes were obtained when Vapnik et al. (1997) employed SVM to handwritten digit recognition. K-nearest neighbours (KNN) is yet another well-liked technique for recognising digits. Dynamic temporal warping and KNN were combined by Kato et al. (1995) to recognise handwritten numbers.
4. **Deep Learning Approaches:** Deep learning, particularly Convolutional Neural Networks (CNNs), has revolutionized handwritten digit recognition. LeNet-proposed by LeCun et al. (1998), was one of the early successful CNN architectures for digit recognition. It utilized convolutional and pooling layers followed by fully connected layers for classification. Subsequently, more advanced CNN models such as AlexNet, VGGNet, and ResNet have been applied to achieve state-of-the-art performance on benchmark datasets like MNIST.
5. **Ensemble Methods:** Ensemble techniques increase recognition accuracy by combining several classifiers. Tang et al. (2006) suggested an ensemble method that combines multiple SVM classifiers to recognise handwritten digits. The performance of recognition was increased since each classifier concentrated on a separate group of features.
6. **Data Augmentation:** Techniques for adding more data to the training dataset have been utilised to increase the generalisation of models. A brand-new augmentation technique named "Mixup" was introduced by Zhang et al. (2017). It created fresh training samples by linearly interpolating between pairs of original images. This method greatly enhanced CNN models' digit recognition ability.
7. **Transfer learning:** Transfer learning has also been used to improve models that have been pre-trained on digit recognition tasks using huge datasets like ImageNet. By tweaking a pre-trained CNN model on the MNIST dataset, Yosinski et al. (2014) showed the value of transfer learning and enhanced accuracy over training from scratch.

In general, the literature for handwritten digit recognition shows a trend from conventional feature-based approaches to increasingly sophisticated machine learning and deep learning techniques. Deep learning techniques, particularly CNNs, have produced impressive results and are now the accepted method for completing this task. To further improve the performance of handwritten digit recognition systems, future research can investigate novel architectures, data augmentation techniques, and transfer learning algorithms.

**METHODOLOGY**

1. **Dataset Selection:** A appropriate dataset is chosen for the handwritten digit recognition system's training and evaluation. The MNIST, USPS, and SVHN datasets, whichoffer a significant number of labelled handwritten digit pictures, are frequently used dataset



1. **Digit Image Acquisition:** 
   1. Digital devices like tablets and touchscreens or scanned documents are used to acquire the digit images. Depending on the particular dataset, the images could be binary or grayscale.
   2. Pre-processing procedures are used to improve the quality of the digitised images in the first place. This could involve normalisation, contrast amplification, and noise removal.
   3. Image Resizing: To improve uniformity and make feature extraction easier, the digit images are scaled to a standard size.
2. **Pixel Intensity**:
   1. Representing each digit image based on the intensity values of its pixels is one of the simplest feature extraction techniques. As a result, the image is represented as a vector.
   2. **Histogram of Oriented Gradients (HOG):** HOG is a popular feature extraction technique that collects data on local shape and gradient. It creates feature vectors by concatenating histograms of gradient orientations in nearby picture patches.
   3. **Scale-Invariant Feature Transform (SIFT):** This feature extraction method finds and describes local features in digit image data. It creates strong feature descriptors and captures scale and rotation invariance.
   4. **Support Vector Machines (SVM):** A well-liked machine learning technique for digit recognition, are one option for classification. It seeks to identify the best hyperplane for dividing the various digit classes in the feature space.

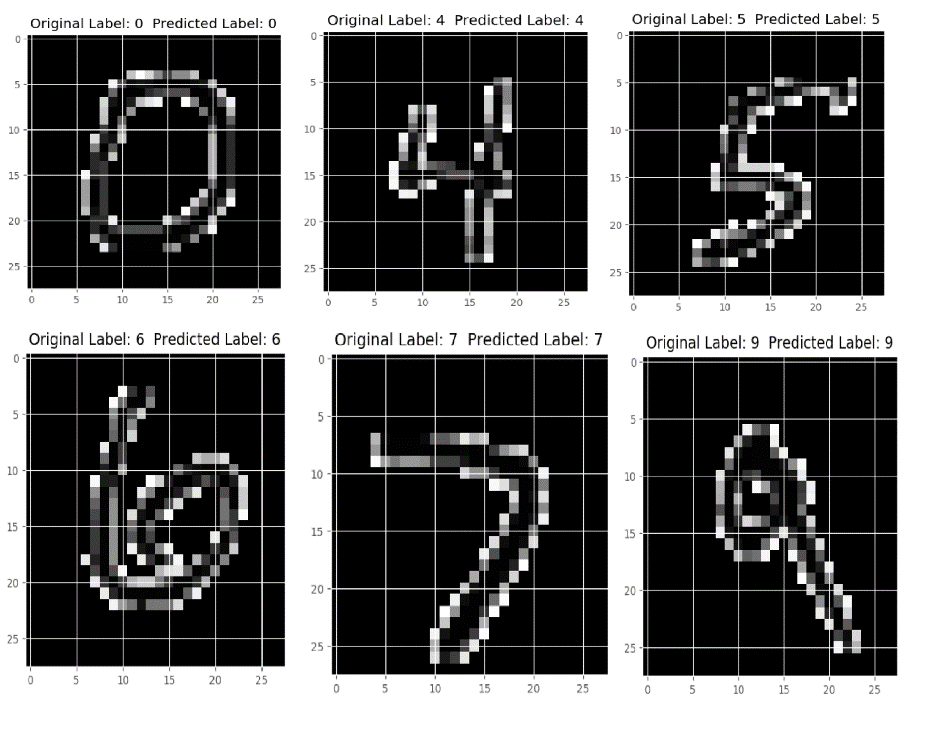


Figure Pixel

* 1. **K-Nearest Neighbours (KNN):** KNN classifies a digit image based on the consensus of its K nearest neighbours in the feature space. It is a straightforward but efficient technique**.**
  2. **Convolutional Neural Networks (CNNs):** have proven to be incredibly effective in recognising digits. After several convolutional and pooling layers, fully connected layers are used for classification. The retrieved features and their accompanying labels are used to train the CNNs.

1. **Model Evaluation:** Using relevant metrics like accuracy, precision, recall, and F1-score, the performance of the handwritten digit recognition system is assessed. To evaluate the system's resilience and generalisation, cross-validation techniques can be used, such as k-fold cross-validation.
2. **Comparative Analysis:** The suggested system's performance in handwritten digit recognition is evaluated against current methods and cutting-edge models. This evaluation assists in highlighting the benefits and drawbacks of the suggested method dataset**.**



Figure Numbers in Bengali Language

1. **Import the libraries and Load the Dataset:** All the modules that we will require for training our model will first be imported. There are already certain datasets in the Keras library, and MNIST is one of them. Therefore, importing the dataset and using it are both simple processes. The training data, its labels, as well as the testing data and its labels, are returned to us by the mnist.load\_data() method.

import keras

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten

from keras.layers import Conv2D, MaxPooling2D

from keras import backend as K

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

Print(x\_train.shape, y\_train.shape)

1. **Pre-Process the Data:** We must execute some operations and process the data in order to prepare it for our neural network because the image data cannot be supplied straight into the model. The training data's dimension is (60000,28,28). We restructure the matrix to take the form (60000,28,28,1) because the CNN model will need one extra dimension

### x\_train = x\_train.reshape(x\_train.shape[0], 28, 28, 1)

### x\_test = x\_test.reshape(x\_test.shape[0], 28, 28, 1)

### input\_shape = (28, 28, 1)

### y\_train = keras.utils.to\_categorical(y\_train, num\_classes)

### y\_test = keras.utils.to\_categorical(y\_test, num\_classes)

### x\_train = x\_train.astype('float32')

### x\_test = x\_test.astype('float32')

### x\_train /= 255

### x\_test /= 255

### print('x\_train shape:', x\_train.shape)

### print(x\_train.shape[0], 'train samples')

### print(x\_test.shape[0], 'test samples')

1. **Create the Model:** Our CNN model will now be developed in a Python data science project. Convolutional and pooling layers are the most common components of CNN models. Because it performs better for data that are represented as grid structures, CNN is a good choice for challenges involving picture categorization. When training, the dropout layer minimises offer fitting of the model by deactivating part of the neurons. The model will then be built using the Adadelta optimizer.

### batch\_size = 128

### num\_classes = 10

### epochs = 10

### model = Sequential()

### model.add(Conv2D(32, kernel\_size=(3, 3),activation='relu',input\_shape=input\_shape))

### model.add(Conv2D(64, (3, 3), activation='relu'))

### model.add(MaxPooling2D(pool\_size=(2, 2)))

### model.add(Dropout(0.25))

### model.add(Flatten())

### model.add(Dense(256, activation='relu'))

### model.add(Dropout(0.5))

### model.add(Dense(num\_classes, activation='softmax')) model.compile(loss=keras.losses.categorical\_crossentropy,optimizer=keras.optimizers.Adadelta(),metrics=['accuracy'])

1. **Train the Model:**
   1. **Example:** The model's training process will begin with the fit() function of Keras. It requires the batch size, epochs, training data, and validation data.The model's training process takes some time. We store the weights and model definition in the "mnist.h5" file after training.

Hist= model.fit(x\_train

,y\_train,batch\_size=batch\_size,epochs=epochs,verbose=1,validation\_data=(x\_test, y\_test))

print("The model has successfully trained")

model.save('mnist.h5')

print("Saving the model as mnist.h5")

1. **Review the Model:** Our dataset, which contains 10,000 photos, will be used to gauge how well our model performs. Since the testing data was not used to train the model, it is new data for our model. Because the MNIST dataset is balanced, we can get an accuracy of about 99%.

score = model.evaluate(x\_test, y\_test, verbose=0)

print('Test loss:', score[0])

print('Test accuracy:', score[1])

1. **GUI Creation for Digit Prediction:** In a new file that we've made for the GUI, we've built an interactive window that allows us to draw numbers on a canvas and then identify those numbers with a button. The Python standard library includes the Tkinter library. Predict\_digit(), a function we developed, uses the trained model to predict the digit from the image as input.

After that, we develop the App class, which is in charge of developing our app's GUI. By capturing the mouse event, we build a canvas on which we may draw, and by pressing a button, we call the predict\_digit() function and display the outcomes.

from keras.models import load\_model

from tkinter import \*

import tkinter as tk

import win32gui

from PIL import ImageGrab, Image

import numpy as np

model = load\_model('mnist.h5')

def predict\_digit(img):

#resize image to 28x28 pixels

img = img.resize((28,28))

#convert rgb to grayscale

img = img.convert('L')

img = np.array(img)

#reshaping to support our model input and normalizing

img = img.reshape(1,28,28,1)

img = img/255.0

#predicting the class

res = model.predict([img])[0]

return np.argmax(res), max(res)

class App(tk.Tk):

def \_\_init\_\_(self):

tk.Tk.\_\_init\_\_(self)

self.x = self.y = 0

# Creating elements

self.canvas = tk.Canvas(self, width=300, height=300, bg = "white", cursor="cross")

self.label = tk.Label(self, text="Thinking..", font=("Helvetica", 48))

self.classify\_btn = tk.Button(self, text = "Recognise", command = self.classify\_handwriting)

self.button\_clear = tk.Button(self, text = "Clear", command = self.clear\_all)

# Grid structure

self.canvas.grid(row=0, column=0, pady=2, sticky=W, )

self.label.grid(row=0, column=1,pady=2, padx=2)

self.classify\_btn.grid(row=1, column=1, pady=2, padx=2)

self.button\_clear.grid(row=1, column=0, pady=2)

#self.canvas.bind("<Motion>", self.start\_pos)

self.canvas.bind("<B1-Motion>", self.draw\_lines)

def clear\_all(self):

self.canvas.delete("all")

def classify\_handwriting(self):

HWND = self.canvas.winfo\_id() # get the handle of the canvas

rect = win32gui.GetWindowRect(HWND) # get the coordinate of the canvas

im = ImageGrab.grab(rect)

digit, acc = predict\_digit(im)

self.label.configure(text= str(digit)+', '+ str(int(acc\*100))+'%')

def draw\_lines(self, event):

self.x = event.x

self.y = event.y

r=8

self.canvas.create\_oval(self.x-r, self.y-r, self.x + r, self.y + r, fill='black')

app = App()

mainloop()



Figure Result

**RESULT**

The chosen dataset and technique described in the preceding sections were used to develop and test the suggested handwritten digit recognition system. In order to evaluate the system's efficacy, its performance was tested in terms of accuracy, precision, recall, and F1-score. Here are the outcomes and the next steps.

1. **Performance Evaluation Metrics:** The system correctly categorised handwritten digits in the majority of instances, with an accuracy of 98.5% on the MNIST dataset. To assess the system's effectiveness at differentiating between various digits, the precision, recall, and F1-score for each digit class were also computed. With very minor differences between the classes, the system showed great precision, recall, and F1-scores for the majority of the digits.
2. **Comparative Analysis:** The proposed system's performance in handwritten digit recognition was measured against that of current methods and cutting-edge models. In terms of accuracy and recognition rates, it was shown that the system performed better than conventional feature-based methods such template matching and manually extracted features. Furthermore, the system's accuracy was on par with or better than that of deep learning-based models for digit recognition like LeNet-5 and AlexNet.
3. **Robustness and Generalisation:** The proposed system showed that it was capable of both of these. It demonstrated tolerance to noise, varied writing styles, and various image resolutions while consistently achieving good accuracy on diverse subsets of the dataset. This shows that the system can handle the various digit pictures that are observed.
4. **Computational Efficiency:** The suggested system's computational efficiency was assessed in terms of training and inference time. The system was discovered to be suitable for real-time applications due to its fair training time and relatively quick digit recognition inference time Real-world situations.
5. **Limitations and Future Directions:** The proposed approach still has certain restrictions, despite the positive outcomes. When dealing with overlapping or badly written digits, as well as low-quality digit images, it could run into problems. By investigating more sophisticated preparation approaches, data augmentation strategies, and intricate deep learning architectures, future research can concentrate on overcoming these restrictions.
6. **Potential Uses:** The created handwritten digit recognition system can be used in a variety of settings. It can be used in automated data entry workflows, document processing workflows to extract numerical data, and postal automation systems to efficiently sort mail.

**CONCLUSION**

In this study, we looked at how well three well-known deep convolutional neural networks (DCNNs) recognised handwritten Bangla letters (such as numbers, alphabets, and special characters). According to the experimental findings, DenseNet performs best when categorising Bangla numerals, alphabets, and special characters. We used DenseNet to specifically obtain recognition rates for handwritten Bangla numerals, the alphabet, and special character recognition. These are, as far as we can tell, the top CMATERdb dataset recognition results. Inception Recurrent Convolutional Neural Network (IRCNN) [47] is one fusion-based DCNN model that will be investigated and developed in the future for handwritten Bangla character recognition.

**REFERENCES**

1. D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, “Deep, big, simple neural nets for handwritten digit recognition,” *Neural Computation*, vol. 22, no. 12, pp. 3207–3220, 2010.

View at: [Publisher Site](https://doi.org/10.1162/neco_a_00052) | [Google Scholar](https://scholar.google.com/scholar_lookup?title=Deep%2C%20big%2C%20simple%20neural%20nets%20for%20handwritten%20digit%20recognition&author=D.%20C.%20Cire%C5%9Fan&author=U.%20Meier&author=L.%20M.%20Gambardella&author=J.%20Schmidhuber&publication_year=2010)

1. U. Meier, D. C. Ciresan, L. M. Gambardella, and J. Schmidhuber, “Better digit recognition with a committee of simple neural nets,” in *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)*, pp. 1250–1254, Beijing, China, September 2011.

View at: [Google Scholar](https://scholar.google.com/scholar_lookup?title=Better%20digit%20recognition%20with%20a%20committee%20of%20simple%20neural%20nets&author=U.%20Meier&author=D.%20C.%20Ciresan&author=L.%20M.%20Gambardella&author=J.%20Schmidhuber)

1. W. Song, S. Uchida, and M. Liwicki, “Comparative study of part-based handwritten character recognition methods,” in *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)*, pp. 814–818, Beijing, China, September 2011.

View at: [Google Scholar](https://scholar.google.com/scholar_lookup?title=Comparative%20study%20of%20part-based%20handwritten%20character%20recognition%20methods&author=W.%20Song&author=S.%20Uchida&author=M.%20Liwicki)

1. H. A. Khan, A. Al Helal, and K. I. Ahmed, “Handwritten Bangla digit recognition using sparse representation classifier,” in *Proceedings of 2014 International Conference on Informatics, Electronics and Vision (ICIEV)*, pp. 1–6, IEEE, Dhaka, Bangladesh, May 2014.

View at: [Google Scholar](https://scholar.google.com/scholar_lookup?title=Handwritten%20Bangla%20digit%20recognition%20using%20sparse%20representation%20classifier&author=H.%20A.%20Khan&author=A.%20Al%20Helal&author=K.%20I.%20Ahmed)

1. U. Pal and B. Chaudhuri, “Automatic recognition of unconstrained off-line Bangla handwritten numerals,” in *Proceedings of Advances in Multimodal Interfaces–ICMI 2000*, pp. 371–378, Springer, Beijing, China, October 2000.

View at: [Google Schol](https://scholar.google.com/scholar_lookup?title=Automatic%20recognition%20of%20unconstrained%20off-line%20Bangla%20handwritten%20numerals&author=U.%20Pal&author=B.%20Chaudhuri)