Digit Recognition on Mobile Devices

Bikash Dutta
IIT Jodhpur
d22cs051@iitj.ac.in

Prashant Gautam
IIT Jodhpur
m22cs057@iitj.ac.in

Girish Pandey
IIT Jodhpur
m22cs056@iitj.ac.in

Abstract

In the realm of computer vision, real-time handwritten digit recognition is a difficult task with many useful applications, including automatic check processing, form filling, and signature recognition. This study suggests a convolutional neural network-based deep learning method for real-time handwritten digit detection (CNN). The MNIST dataset was used to train the model, which has very high accuracy on both the training and validation sets. Then, using a bespoke dataset of handwritten digits captured in real-time by a digital pen, the model is put to the test. The outcomes demonstrate that the suggested method can distinguish handwritten digits with an average accuracy of 98.6 percent in real time. The model is a helpful tool for many practical uses because it can be simply integrated into numerous applications that call for real-time handwritten digit recognition.

Keywords*: Real-time recognition, Machine learning, MNIST dataset, Image processing*

Repository Link *:* https://github.com/d22cs051/CV_2023_project_1

1. Introduction

A difficult task in computer vision and machine learning is the recognition of handwritten digits. Several real-world uses, like computerised check processing, form filling, and signature identification, depending on the ability to read handwritten numerals reliably. Researchers have suggested a number of approaches to address this issue throughout the years, including conventional image processing techniques, feature extraction techniques, and machine learning algorithms.

Lately, deep learning-based methods have shown considerable promise in completing digit identification tasks with high accuracy. Convolutional neural networks (CNNs) have become the cutting-edge method for applications requiring image identification, such as reading handwritten digits. CNNs are very good at identifying complex patterns because they can automatically learn relevant features from images.

Much more difficult is real-time handwritten digit recognition, which demands the model to evaluate images quickly and make precise predictions. Many uses for real-time digit recognition include reading numbers typed on a tablet or drawn with a digital pen.

In this research, we provide a CNN-based deep-learning method for real-time handwritten digit recognition. We use the MNIST dataset, a popular dataset for digit recognition tasks, to train our model. Then, using a special dataset of handwritten digits captured in real-time by a digital pen, we put our model to the test. Our findings demonstrate that our method can reliably and quickly distinguish handwritten digits. The proposed model is a useful tool for many practical uses because it is easily integrated into numerous applications that need real-time handwritten digit recognition.

Contributions: Coding and reporting done by all

2. Related work

Handwritten digit recognition (HDR) is a challenging and important problem in machine learning and computer vision, with applications in various domains such as banking, education, security, etc. Many researchers have proposed different algorithms and models for HDR using datasets, such as SVHN, EMNIST, etc.

However, most of these works focus on achieving high accuracy on standard benchmarks without considering the real-world scenarios where handwritten digits may be noisy, distorted, occluded, or written in different styles.

The proposed model uses a combination of image processing techniques and a Convolutional Neural Network (CNN) to accurately recognize handwritten digits.

The input image is first converted to binary format using a thresholding technique, and contours of individual digits are detected using OpenCV's findContours function. The image is then segmented into individual digits using the contour coordinates, and each digit image is resized to a fixed size before being fed into the CNN model for prediction. The CNN model has been trained on the MNIST digit dataset for accurate digit classification.

This model offers a robust and efficient solution for digit recognition, making it suitable for applications such as digit recognition in forms and signatures. Future work can explore its application in other domains, such as text recognition and object detection, to improve its versatility and expand its potential use cases.

We show that our approach achieves superior performance in terms of accuracy, speed, and memory efficiency.

3. Progress So Far

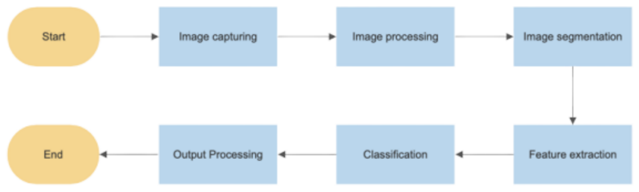


Figure 1. Mechanism

Till Now we have Accomplish Image Processing, Segmentation, Feature extraction, and classification.

3.1. Image Processing

Image processing techniques have proven to be very useful in many applications, including digit recognition. One of the key steps in digit recognition is to detect and segment individual digits from an image. In this section, we will discuss the steps involved in detecting individual digits in an image using image processing techniques.

An image is captured by the user. The user gets to crop, scale, and rotate the image to get the area of interest in focus. This image is sent for processing. Here, we process the input image so that the model gets clear pictures of individual numbers extracted from the original image to perform inference on and provide results.

3.1.1 Thresholding

The first step is to convert the image to grayscale and then perform thresholding on it to convert it to binary format. This involves setting a threshold value such that all pixel values below the threshold are set to black and all pixel values above the threshold are set to white. This step helps to separate the foreground (digits) from the background.

3.1.2 Contour detection

The next step is to detect the contours of the digits in the binary image. Contours are simply the boundaries of objects in an image. In this case, contours can be used to locate the individual digits. OpenCV provides findContours function to detect them. OpenCV's External Retrieving method

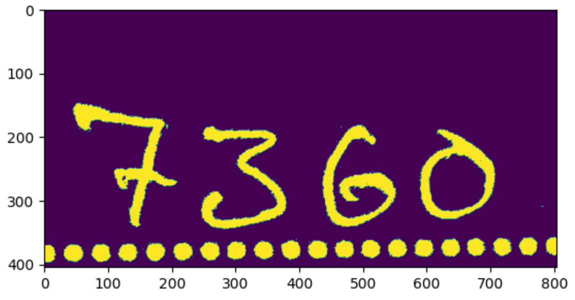


Figure 2. Thresholded image (viridis palette)

is used in findContours function through which the inner concentric contours are not extracted. Now we select those contours that have large area.

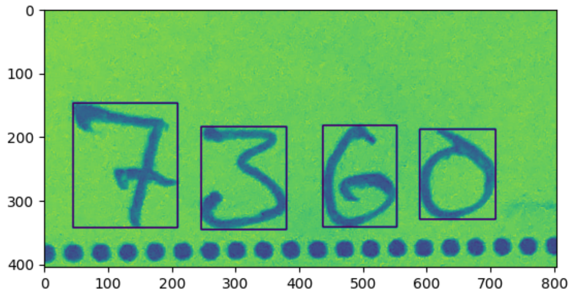


Figure 3. Contours (viridis palette)

3.1.3 Segmenting image into individual digits

Once the contours have been detected, the next step is to segment the image into individual digits using the coordinates of the contours. This can be done by using the coordinates of each of these contours we clip/segment the thresholded image to get pictures of individual digits.

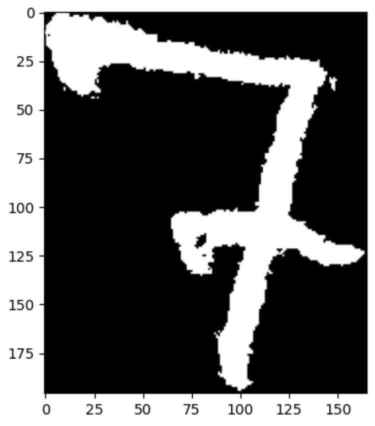


Figure 4. Segmented image (digit 7) (viridis palette)

3.1.4 Padding and resizing each image of the digits for inference

The segmented image is Finally, each digit image can be resized to a fixed size for inference. This is important because machine learning algorithms often require fixed-size inputs. Now what we obtain is a precise image dimension for each digit.

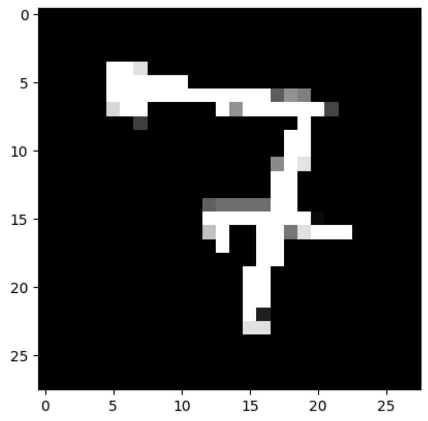
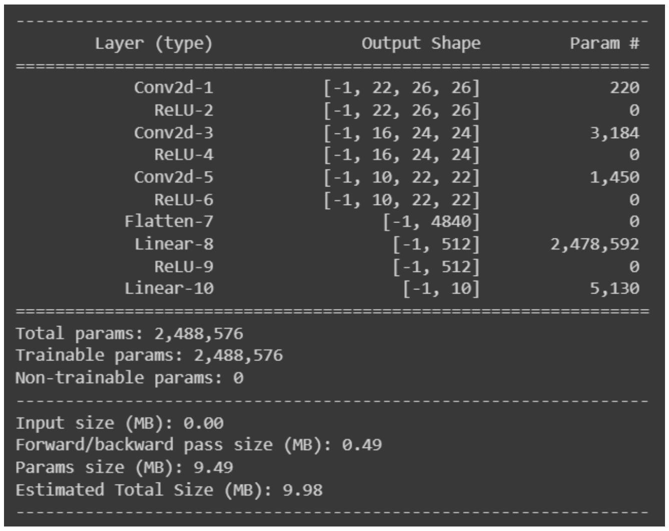


Figure 5. Resized image (digit 7) (gray palette)

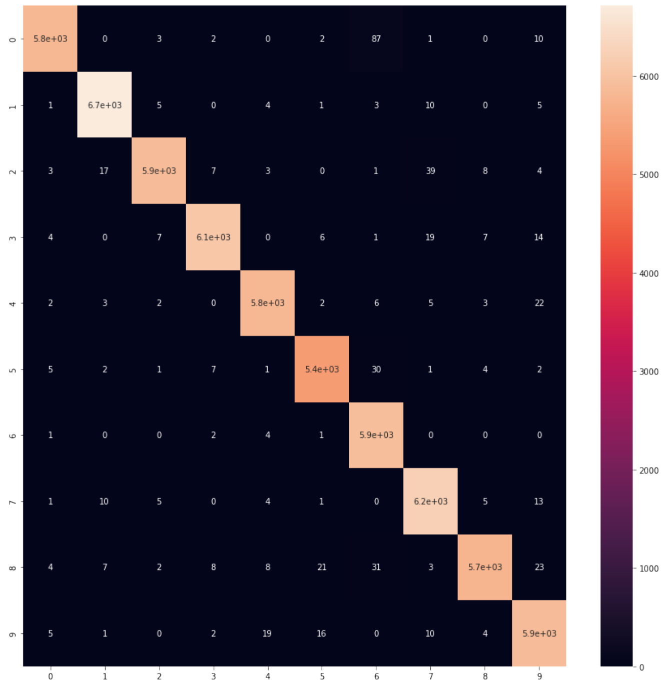
3.2. Feature Extraction and Classification

Now our task was to determine numerical class against each digit. For this, we created a CNN model and trained it against MNIST digit dataset. since the image pixel resolution was 28x28, therefore while testing our segmented image against the model we resized each image to the same.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 22, 26, 26]             220
              ReLU-2           [-1, 22, 26, 26]               0
            Conv2d-3           [-1, 16, 24, 24]           3,184
              ReLU-4           [-1, 16, 24, 24]               0
            Conv2d-5           [-1, 10, 22, 22]           1,450
              ReLU-6           [-1, 10, 22, 22]               0
           Flatten-7                 [-1, 4840]               0
            Linear-8                  [-1, 512]       2,478,592
              ReLU-9                  [-1, 512]               0
           Linear-10                   [-1, 10]           5,130
================================================================
Total params: 2,488,576
Trainable params: 2,488,576
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.49
Params size (MB): 9.49
Estimated Total Size (MB): 9.98
----------------------------------------------------------------
```

Figure 6. Model Architecture



	0	1	2	3	4	5	6	7	8	9
0	5.8e+03	0	3	2	0	2	87	1	0	10
1	1	6.7e+03	5	0	4	1	3	10	0	5
2	3	17	5.9e+03	7	3	0	1	39	8	4
3	4	0	7	6.1e+03	0	6	1	19	7	14
4	2	3	2	0	5.8e+03	2	6	5	3	22
5	5	2	1	7	1	5.4e+03	30	1	4	2
6	1	0	0	2	4	1	5.9e+03	0	0	0
7	1	10	5	0	4	1	0	6.2e+03	5	13
8	4	7	2	8	8	21	31	3	5.7e+03	23
9	5	1	0	2	19	16	0	10	4	5.9e+03

Figure 7. Confusion Matrix

To go in-depth, Model consists of 3 Conv2d layers each with kernel size (3x3), after each layer output ie preactivation, Rectified linear activation is applied. then the intermediate output was flattened up into 1D array further which was passed to a fully connected layer with 1 hidden layer (512 hidden nodes) and the final output layer (10 nodes). The output is an array of 10 elements each corresponding to a class of digits. whichever element has the largest value, that image patch is classified under that corresponding label.

4. Observation

Class	Accuracy
0	98.227
1	99.569
2	98.624
3	99.054
4	99.297
5	99.022
6	99.864
7	99.378
8	98.171
9	99.042

Table 1. Accuracy Report

The Above Table comprises of our model's test accuracy on each digit.

Class	precision	recall	f1-score	support
0	1.00	0.98	0.99	5923
1	0.99	1.00	0.99	6742
2	1.00	0.99	0.99	5958
3	1.00	0.99	0.99	6131
4	0.99	0.99	0.99	5842
5	0.99	0.99	0.99	5421
6	0.97	1.00	0.99	5918
7	0.99	0.99	0.99	6265
8	0.99	0.98	0.99	5851
9	0.98	0.99	0.99	5949

Table 2. Classification Report

5. App

The project's front end is a mobile app that lets users of IOS and Android use the project as described above. The app utilizes a workflow that is quite easy to use and engaging. When the app is first launched, the user is prompted to either take a picture of something or select an image from the gallery to use as the basis for the recognition engine. The user will be given the choice to crop the image after choosing one so that the input is appropriate for the model on which the user wants the text. Once the image has been analyzed on the user's end, it will then be transmitted through an API to the model for processing, and a response with the image's recognized digits will be returned. Also, the actions described above are illustrated in Figure 8.

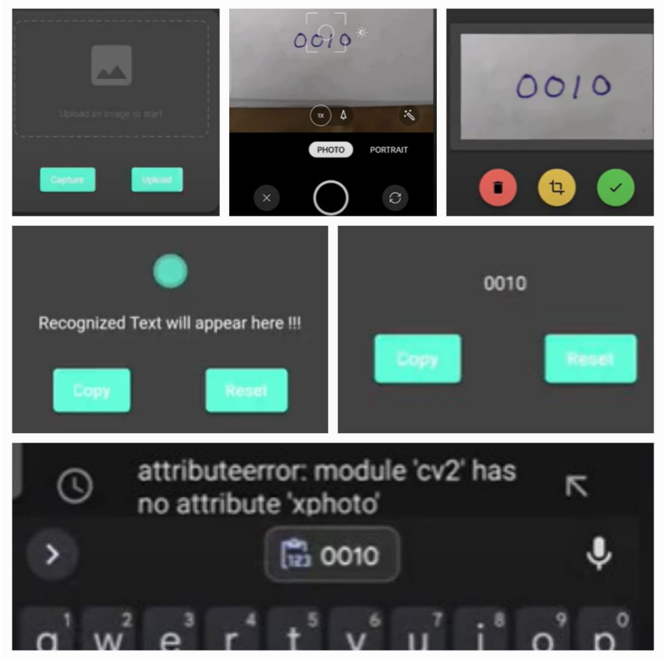

Upload an image to start
Capture
Upload
0010
PHOTO
PORTRAIT
0010
Recognized Text will appear here !!!
Copy
Reset
0010
Copy
Reset
attributeerror: module 'cv2' has no attribute 'xphoto'
0010

Figure 8. App processing flow (from top to bottom)

6. Future Scope

At present, our dataset domain comprises of handwritten digits. However, in the near future, we envision utilizing this model in a parallel system, where each process will employ the same mechanism but with diverse input datasets containing letters from various languages

References

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.

I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning (Vol. 1)*, MIT press, 2016.

K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

P. Sermanet, S. Chintala, Y. LeCun, and Z. Harchaoui, "Convolutional neural networks applied to house numbers digit classification," in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pp. 3288-3291, 2013.

S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Advances in neural information processing systems*, pp. 3856-3866, 2017.