Implementation of Handwritten Digit Recognizer using CNN

B M Vinjit, Mohit Kumar Bhojak, Sujit Kumar and Gitanjali Nikam

National Institute of Technology, Kurukshetra Haryana, India

Abstract

In this paper, we have presented an implementation of Handwritten Digit Recognition. HCR or Handwritten Character Recognition is one of the most challenging tasks in machine learning as the handwriting of every individual on the Earth is unique. So it's quite challenging to train a model that can predict handwritten text with high accuracy. We have developed a CNN (Convolutional Neural Network) model that can recognize digits from images with a 99.15% accuracy. This model is useful in converting handwritten numbers to digital form and our purpose to build this model is to develop a system for automatically inserting marks awarded to students on answer sheets into a database. Nowadays, everything is getting digitized and the proposed application is capable of reducing the effort taken and mistakes done while manually inserting numbers into the database.

Keywords

Machine Learning, Artificial Intelligence, Convolutional Neural Network, Optical Character Recognition, Offline Recognition, Handwritten character recognition, Image Processing

1. Introduction

Handwritten Digit Recognition is a classic problem of image classification. In this, we have to classify handwritten digits into labels i.e. 0-9. Neural Network models are very powerful and efficient classification methods to perform this task. Human beings are intelligent and can read and recognize different handwritten characters and digits written by other fellow humans. We want to inculcate the same features in a machine using Artificial Intelligence and Machine Learning.

Covid-19 global pandemic made us realize the significance of digitalization in educational and business organizations. It makes all the processes faster, efficient, and accessible. Features like Auto Classification, Text Reader, Automatic Data Extraction, etc. helps in the identification of the documents and indexes accordingly. In remote teaching and work from home scenarios, these features can be pivotal. The conversion of an image of text (printed or handwritten) to machine-encoded format is Optical Character Recognition (OCR).

This technology is widely used in form processing and data entry applications. Various stages of OCR are shown in Fig. 1 and we go through these four-stage process while implementing an OCR model. Training a model for all the different handwritings in this world is impossible as handwriting is unique to every individual. So we can train a model using a large dataset of handwritten digits like the MNIST dataset and test them on other handwritings. OCR is challenging but very useful for quick processing of data records like bank statements, emails, passport documents, invoices, mark sheets, etc. OCR helps in digitizing handwritten and printed text and hence making it easy to apply functions like searching, sorting, and editing easier.

Accuracy is the most important parameter in our proposed system - to automate the process of manual entry of numeric data (marks, roll number, subject code, etc.), as even one wrongly recognized digit can have serious consequences. The accuracy and efficiency of the system bank upon the methodology and dataset used. In this implementation, we have used the Convolutional Neural Network (CNN). CNN has a couple of key characteristics. The patterns that they learn are translation invariant [1]. After learning a certain pattern convolution neural network can recognize it anywhere. They can learn spatial hierarchies of the pattern. In the first

ACI'21: Workshop on Advances in Computational Intelligence at ISIC 2021, February 25–27, 2021, Delhi, India
EMAIL: bm_51810062@nitkkr.ac.in (B M Vinjit);
ORCID: 0000-0003-4242-1885 (B M Vinjit);

convolution layer, we learn small local patterns such as edges. In the second convolutional layer, we learn larger patterns made from features from the first layer, and so on. This allows us to efficiently learn increasingly complex and abstract visual concepts [2].

The paper is structured as: Section 2 contains steps involved and other details of the implementation of the model. The pseudocode of our implemented system is written in Section 3. Section 4 contains a comparison with the traditional model for character recognition from images and Section 5 contains the result and some further scopes and improvements.

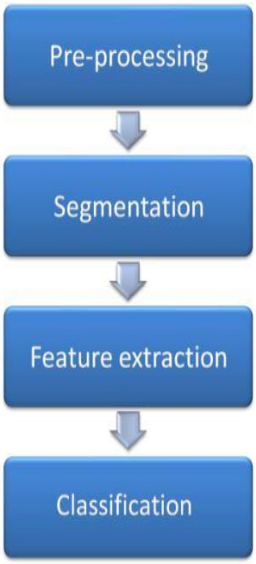

Figure 1: Stages of Optical Character Recognition [3]

2. Implementation

In this section, we will discuss the segmentation process and implementation of the CNN model we created using Tensor Flow for digit recognition from the image.

Our first task will be to convert a numeric string into single digits. We used the OpenCV library for the segmentation of a string. We converted a colored image into grayscale using the BGR2GRAY function in OpenCV. We are recognizing black pixels in the white background for the segmentation process. We make rectangular boxes around the digits in a string and extract them. We are using the matplotlib library to represent it. After getting the digits from the string, we pass them into a CNN model.

To develop the CNN model we download and pre-process MNIST handwritten digital mission data set (Refer Fig. 2). We reshape each image into a 4D tensor. This is done to satisfy the input requirements of the CNNs. So, the reshaping is done with a reshape command. Both train images and test images tensors are reshaped into 4D tensors. Now we have prepared data for training. To build a model we will first create the convolution base. We are using the equation model. We add a convolution layer to the model. The convolution layer is added using layers.Conv2D command.

The convolution operation involves a filter that captures a local pattern and applies it to the image as shown in Fig. 3. The filter is a 3D tensor of a specific width, height, and depth. Each entry in the filter is a real number. The entries in the filter are learned during the CNN transition. The filter slides across the width, height, and depth stopping at all possible positions to extract a local 3D patch of the surrounding feature. We take the filter and position it at different places in the image. If we slide the filter we get a new position of the filter. We keep doing this across the length and breadth of the image till the final positioning of the filter. Since the input is a grayscale image we have depth = 1.

Apart from convolution, there is a second important operation - pooling. Pooling aggressively downsamples the output of the convolution. Strides while sliding across the image, provides a way to calculate the next position along each axis to position the filter starting from the current position. We have taken stride = 1 which is the most common choice.

Such a stridden convolution tends to downsample the input by the factor proportional to the stride. It helps us to calculate the next position of the filter. The filter got shifted by one column to the right. Once it exhausts all the columns we start shifting it downwards by rows. This is how we slide the filter across the image and try to match the pattern in the image. So, let us say we have a 28 x 28 x 1 image and we have a filter of size 3 x 3 x 1 using which we will be able to position the filter at 26 possible positions along the width as well as on the height. So, the final position of the filter will be at position 26. So, this is how we get 26 x 26 x 1 output of the convolution. In a convolution layer, we

typically used k different filters. We define all these filters with the Conv2D layer in a Keras command. In a tf.keras API, we use Conv2D. We specify the number of filters, the size of the patch, the activation, and the input shape. Here, we have 32 filters; each filter is of size 3 x 3. Then we specify the activation that we want to use after a linear combination of weight of the filter with the values of the pixel in the image and finally, we specify the shape of the input. The size of the filter and the stride (which is 1 by default) is applicable across all the k filters. After applying the convolution of k filters we get a 3D tensor with the same number of rows and columns for each filter.

All the outputs are combined. So, we get all the channels to be 32; each having 26 x 26 output. So, concretely for our MNIST example, we get a 3D tensor as output with 26 rows 26 columns, and 32 channels, i.e. 1 for each filter. The total number of parameters for this filter will be 320 because we have 10 parameters per filter as shown in Fig. 4.

Pooling is usually done with a window of size 2 x 2 with a stride of 2. We apply the pooling policy on the first 2x2 square box and select a number based on that policy. We use either max pooling or average pooling as the pooling policies. The second important point is we apply pooling operation on each channel separately.

If the output is 26 x 26 x 1 and if we apply a max-pooling of 2 x 2 we get the output of 13 x 13 x 1. So, we can see that there is downsampling happening from the output of the convolution when we apply max pooling on it as depicted in Fig. 5. Note that max-pooling does not have any parameters. In practice, we set up a series of convolution and pooling layers in CNNs. The number of convolution and pooling layers is a configurable parameter and is set by the designer of the network. In the current example, we use two convolutions and one more convolution layer.

In the current example, we use two convolution pooling layers and one additional convolution layer at the end. We use 32 filters in the first convolution layer and 64 filters each in the second and the third layer. Each filter is 3 x 3 in size and we use a stride of 1. We have not used any padding in any of the convolution layers. We used max-pooling for downsampling with a window of 2 x 2 with a stride of 2.

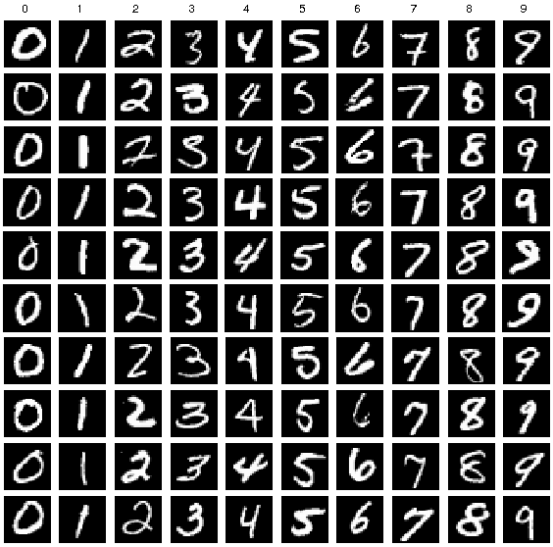

Figure 2: MNIST Dataset [4]

The number of parameters in the convolution layer depends only on the filter size. It does not depend on the height and width of the input. It can be observed that the width and height dimensions tend to shrink as we go deeper into the network. We started with a height and width of 28 each and after a couple of convolutional pooling followed by a single convolution operation, we got a height and width of 3.

So, what is happening here is, we take an image, we apply a bunch of convolution pooling operations that gave us a representation that will feed into a feed-forward neural network which will give us the label corresponding to the digit written in the image.

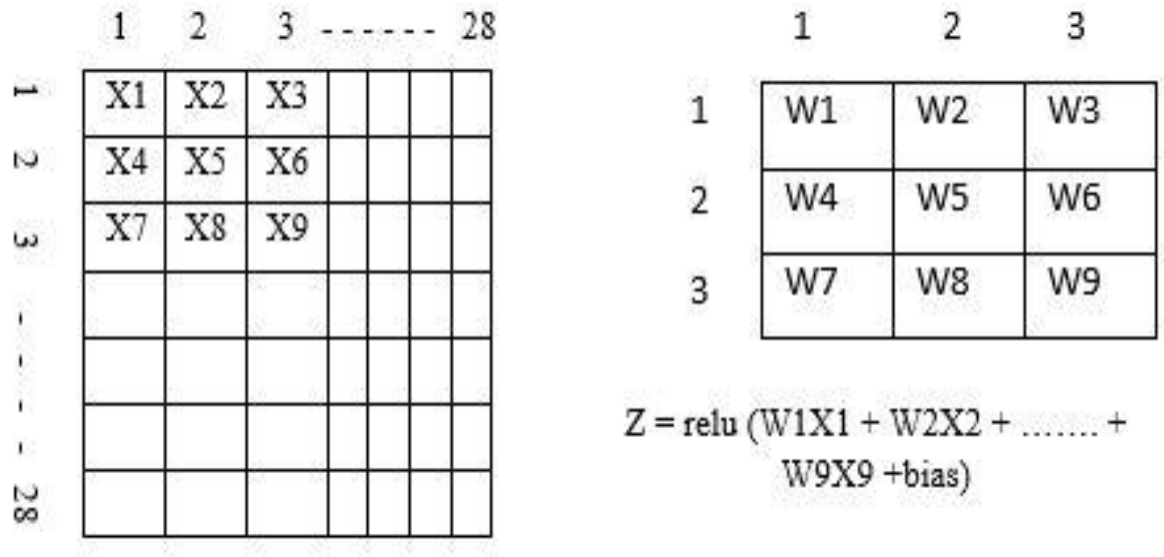

Figure *3*: 28x28 pixel image (left) and 3x3 filter (right) with activation function (Z)

So, here the input is (3, 3, 64). We pass it to the Flatten layer which gives us 576 numbers that are fed into a dense layer whose output is fed into another dense layer; flatten has no parameters it outputs 576 numbers which are input to each of the 64 units in the dense layer over here. So, each of the units in the dense layer has 576 parameters + 1 bias which makes it to 577 parameters per unit and we have 64 such units making it, 36928 parameters. So, this produces 64 values one corresponding to each unit. So, the final dense layer has 10 units; each unit receives 64 values from the previous layer adding 1 bias parameter to it makes it 65 values per unit. So, in total, we have 650 parameters for the final layer. So, if we sum across the CNNs and fully connected top layer which gives a total of 93,322 parameters. For training the model we use *sparse_categoricalcrossentropy* loss with *Adam* optimizer. We train the model for 5 epochs with training images and training labels.

So, we can see that in the case of CNN we are defining patches and we are taking a patch and performing a convolution operation with the filter. We perform a linear combination of each position of the image with each parameter in the filter. We perform linear combination followed by non-linear activation; while in the case of feed-forward neural network we take the entire image, you flatten it so that we get a single array. In this case, since we have a 20 x 20 image we get an array of 576 numbers which we are passing to a hidden layer with 128 units followed by a dense layer of 10 units to get the output. If we come up with the equivalent flattened representation, we have these 9 values and we have a node. These 9 values are connected to this particular node which is a neuron or a unit in the neural network which performs linear combination followed by activation. So, we are capturing local patterns in CNN.

So, CNN procedures by capturing local patterns; whereas, in a feed-forward neural network, a global pattern involving all the pixels are captured.

3. Pseudocode

In this section, we are giving pseudocode of the segmentation of image containing a string of numeric digits and CNN model recognize each digit in the string and store it.

segmentation()
img=image of string
image_size=height_of_image*width_of_image

preprocess image using MSER in opencv library

{Convert the image to grayscale}

gray_image = BGR2GRAY (img , image_size)

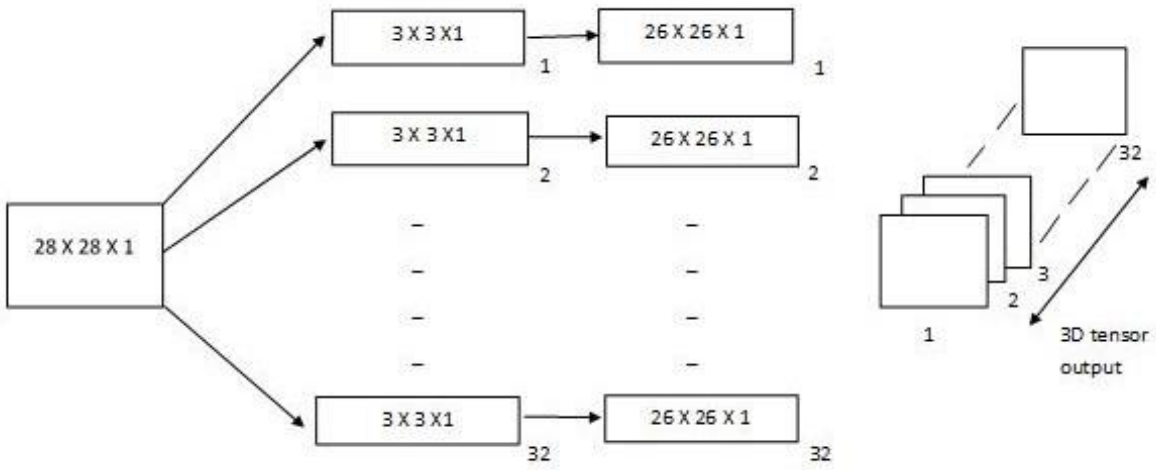

Figure *4*: 32 3x3 patch applied on the image giving 3D tensor of 26x26x32

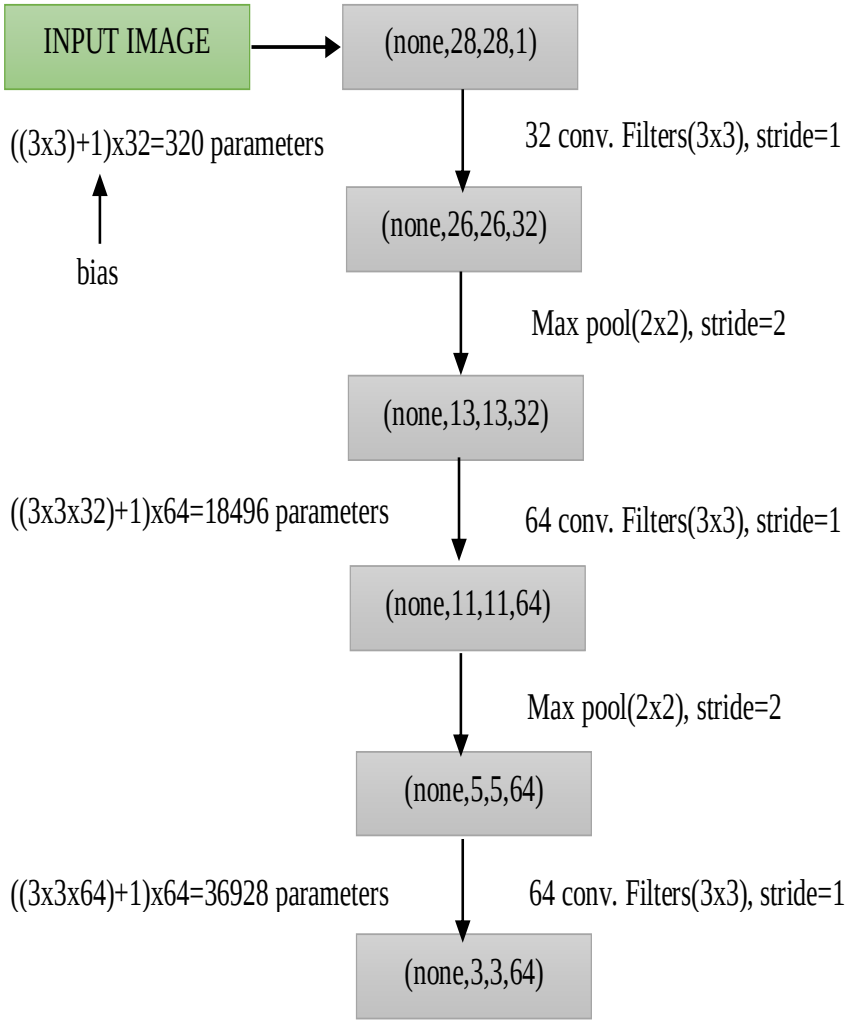

Figure 5: Convolutional Model Building process and parameter calculation

{Using inbuilt opencv function to detect regions}

```
x,y=detectRegions(binary_values_of_gray_
image)

Makerectagles(x,y,x+height_of_image,
y+width_of_image)
```

{inbuilt function in matplotlib library}

```
plot(gray_image)
return gray_image
```

character_recognition()

```
load (MNIST_dataset)
```

{convert 3D tensor to 4D tensor}

```
reshape(train_images,4)
reshape(test_image,4)
```

{Normalize pixel values to be between 0 and 1}

```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

{Make a sequential model by adding convolutional layers with relu activation}

```
add_conv_layer(activation='relu')
add_MaxPooling()
add_conv_layer(activation='relu')
add_MaxPooling()
add_conv_layer(activation='relu')
print(model.summary())
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 64)	36928
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 64)	36928
dense_1 (Dense)	(None, 10)	650

Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0

Figure 6: Summary of Model and total parameters

To complete our model, we will feed the last output tensor from the convolutional base (of shape (3, 3, 64)) into one or more dense layers to perform classification. Dense layers take vectors as input (which are 1D), while the current output in a 3D tensor. First, we will fatten (or unroll) the 3D output to 1D, then add one or more Dense layers on top as shown in the model summary i.e. Fig.6. MNIST has 10 output classes, so we use a final dense layer with 10 outputs and a softmax activation [5].

```
flatten_layers(model)
add_dense_layer(64,activation='relu')
add_dense_layer(10,activation='softmax')
print(model.summary())
```

{Compiling model with adam optimizer, sparse categorical crossentropy loss and metric we are interested in i.e accuracy}

```
compile(model,optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics='accuracy')
```

{train model for 5 epochs/iterations}

```
train(model,train_images,epochs=5)
test_accuracy=evalute(model,test_images)
print(test_accuracy)
gray_image=segmentation()
```

{Recognise the segmented image one by one and print it}

```
for(all the images in rectangles in gray_image)
    test_predict = model.predict(image)
    max=max_value(test_predict)
    number=index_of(max)
    print(number)
```

4. Comparison with the Traditional Method

We did a detailed review and comparative study of various significant handwritten character recognition methods and techniques in our paper – "A Review on Handwritten Character Recognition Methods and Techniques" published in the International Conference on Communication and Signal Processing (ICCSP), Chennai, India, 2020 [6].

In traditional machine learning flow given an image, we used to first perform feature engineering using computer vision libraries. A feature is fed into any machine learning classifier which after training will give us the output. Now, the feature engineering part in traditional machine learning is getting replaced by CNN. So, we can think of CNNs as a way of generating features automatically for a given image. The beauty of this approach is that the right representation is learned during the model training freeing us from expensive and tedious feature engineering tasks.

We see that Feed Forward Neural Network (FFNN) has more than 100k parameters as compared against 93k parameters that our CNN has and despite that classification with FFNN has less accuracy for training and test data as we can see in Fig. 7.

5. CONCLUSION

We can see from Fig. 8 that in the output, the digits in the string are in rectangular boxes and can be extracted by using the crop function. As shown in Fig. 10 our CNN model has a training accuracy of 99.36% and a testing accuracy of 99.15%. When we segment the string and pass the digits into our model, they are being correctly recognized as we see in Fig. 9 where the grayscale image of digit '8' is correctly predicted by our model. Similarly, we can pass all the digits one by one from the segmented string to obtain the string/number in digital format.

So this model can be used for building a proposed system to automate the process of storing marks and other details like roll number and subject code in a database by just taking a photograph. It will nearly remove the manual process which is hectic, tedious, and error-prone

Further scope involves improving this model to recognize alphabets/character string by training it on a suitable database so that the usability of this system can be extended to other domains of HCR as well.

```
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 5s 76us/sample - loss: 0.2990 - accuracy: 0.9124
Epoch 2/5
60000/60000 [==============================] - 5s 76us/sample - loss: 0.1412 - accuracy: 0.9584
Epoch 3/5
60000/60000 [==============================] - 4s 73us/sample - loss: 0.1062 - accuracy: 0.9675
Epoch 4/5
60000/60000 [==============================] - 4s 72us/sample - loss: 0.0859 - accuracy: 0.9737
Epoch 5/5
60000/60000 [==============================] - 4s 74us/sample - loss: 0.0750 - accuracy: 0.9762

Test on 10000 samples

10000/10000 [==============================] - 1s 66us/sample - loss: 0.0799 - accuracy: 0.9750
97.50000238418579 %
```

Figure 7: Training and Testing accuracy using Feed Forward Neural Network

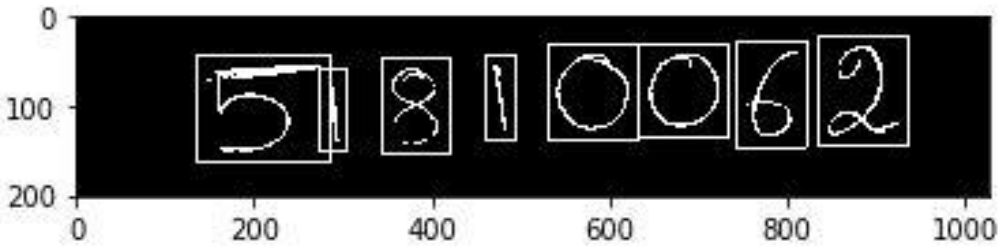

Figure 8: Output of segmentation

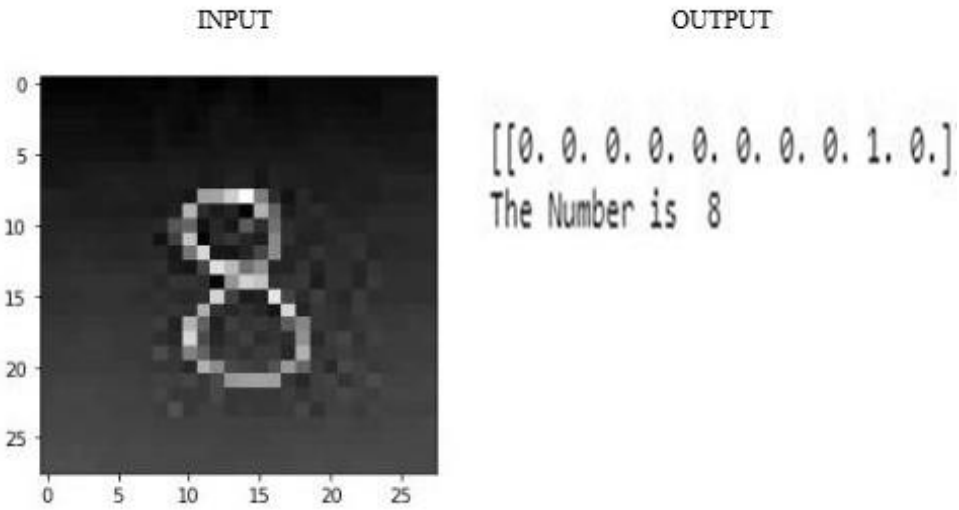

Figure *9*: Grayscale input and predicted output by our model

```
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 12s 201us/sample - loss: 0.1476 - accuracy: 0.9534
Epoch 2/5
60000/60000 [==============================] - 6s 97us/sample - loss: 0.0490 - accuracy: 0.9848
Epoch 3/5
60000/60000 [==============================] - 6s 98us/sample - loss: 0.0344 - accuracy: 0.9893
Epoch 4/5
60000/60000 [==============================] - 6s 98us/sample - loss: 0.0261 - accuracy: 0.9917
Epoch 5/5
60000/60000 [==============================] - 6s 98us/sample - loss: 0.0205 - accuracy: 0.9936

Test on 10000 samples

10000/10000 [==============================] - 1s 81us/sample - loss: 0.0269 - accuracy: 0.9915
99.15000200271606 %
```

Figure 10: Training and Testing accuracy of CNN model

6. References

[1] Donald J. Norris. "Chapter 6 CNN demonstrations", Springer Science and Business Media LLC, 2020

[2] José-Sergio Ruiz-Castilla, Juan-José RangelCortes, Farid García-Lamont, Adrián TruebaEspinosa. "Chapter 54 CNN and Metadata for Classification of Benign and Malignant Melanomas", Springer Science and Business Media LLC, 2019.

[3] RS, S.N., and Afseena, S., 2015. Handwritten Character Recognition–A Review. International Journal of Scientific and Research Publications.

[4] MNIST image-Lim, Seung-Hwan & Young, Steven & Patton, Robert. (2016). An analysis of image storage systems for scalable training of deep neural networks.

[5] Internet Source - www.tensorflow.org

[6] B. M. Vinjit, M. K. Bhojak, S. Kumar and G. Chalak, "A Review on Handwritten Character Recognition Methods and Techniques," 2020 International Conference on Communication and Signal Processing (ICCSP), Chennai, India, 2020, pp. 1224-1228, DOI: 10.1109/ICCSP48568.2020.9182129.