Computer Engineering and Intelligent Systems
ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online)
Vol.14, No.1, 2023

Handwritten Character Recognition Using Deep Learning (Convolutional Neural Network)

Altaf Ramjon[1] Asherl Bwatiramba[1*] Sivakumar Venkataraman[2]

[1] Department of Computer Science, University of East London, London, UK

[1*,2] Botho University, Botho Education Park, Gaborone, Botswana

*E-mail of the corresponding author: asherl.bwatiramba@bothouniversity.co.bw

Abstract

Handwriting recognition is the ability of a computer to receive and interpret intelligible handwritten input from sources such as paper documents, photographs, touch-screens and other devices. The purpose of the research is to design develops, construct, deploy, and test a convolutional neural network (CNN) for handwriting recognition. The CNN for handwriting recognition was developed with Python and MNIST dataset was used. CNN was evaluated together with Simple Neural Network (SNN) and it was found that a CNN operating on well-tuned hardware with GPU and adequate training data can recognize numbers with an accuracy of up to 98.7 percent. The accuracy and speed of the model can be improved by expanding the dataset, increasing the number of epoch runs, and executing it on parallel hardware. Using these strategies, an accuracy of up to 99.89 percent, can be achieved.

Keywords: CNN, SNN, Tensor Flow, Neural Network Architecture, Confusion matrix
DOI: 10.7176/CEIS/14-1-05
Publication date: February 28th 2023

1. Introduction

1.1 Problem Domain

Even though today we are in a digitized world, there are many day-to-day activities that still need information to be written on paper, and then inserted into their respective system. For example, when withdraw or deposit money or cheque into a bank account, we need to fill a form at the cashier with details of the transaction. The bank clerk will then insert these data into their system, which is time consuming and error-prone. Many similar situations exist elsewhere like insurance company, airport, and school and so on where data must be filled in manually and then inserted in a computer system.

For some time now, handwriting recognition systems have been developed to help people in their day-to-day data input activities. Biological neural networks, which enable people and animals to learn and model nonlinear and complicated interactions, can provide inspiration for handwriting recognition systems. The human brain can recognize many handwriting objects, including digits, letters, and characters. However, because people are prejudiced, they can read handwriting letters and numbers differently, (Vadapalli, 2021). Computerized systems, on the other hand, are impartial and capable of performing highly difficult tasks that may require humans to expend a great deal of effort and time. It is necessary to comprehend how humans interpret handwriting. Documentation analysis, mailing address interpretation, bank check processing, signature verification, and postal address verification are just a few of the practical issues that a handwriting recognition system may assist with in the field of information technology. Hence, handwriting recognition technologies are used to facilitate communication between humans and machines.

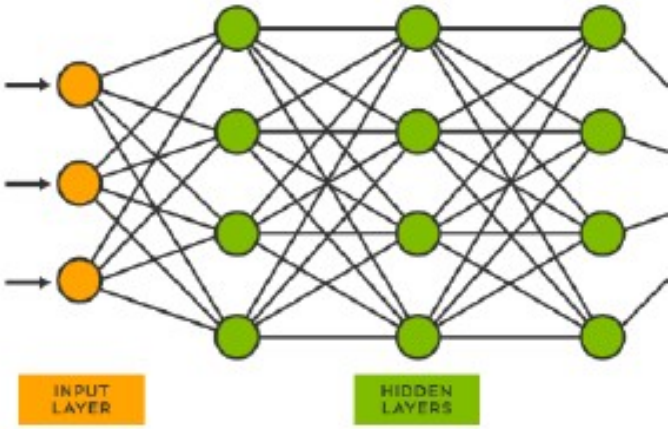
Based on its biological counterparts, artificial neural networks have been developed in computing to simulate some aspects of the work of the human brain. A neural network is the most suited technology for the suggested system because of its ability to draw meaning from complicated data and spot trends from data that are difficult to notice using other human techniques or by humans themselves. It is the primary goal of this study to construct models that will be utilized to interpret handwritten numbers, characters from an image. A general review of the linked works, theoretical basis, the architecture, methodology, experimental results, and conclusion will be provided in the following subsections, (IBM Education, 2020).

1.2 Research Objectives
The main purpose of this research is to design, implement and deploy a deep learning based custom image multi-class classifier for handwriting character recognition.

1.3 Research Questions
This research is aimed to answer the following questions:
What are the different techniques and methods used in handwriting character recognition?
How deep learning can improve handwriting character recognition system?

2. Theoretical Background
Artificial intelligence is the theory and development of computer systems able to perform tasks normally requiring human intelligence such as visual perception. Machine learning is a subfield of artificial intelligence (AI) that focuses on the use of data and algorithms to emulate how humans learn while gradually enhancing its accuracy. Deep learning, on the other hand, is a subfield of machine learning that consists of creating neural networks with three or more layers.

Handwriting recognition has been the subject of many studies, (Hamad & Kaya, 2016). A key feature of most recognition systems for handwriting is the use of neural networks (Figure 1.1



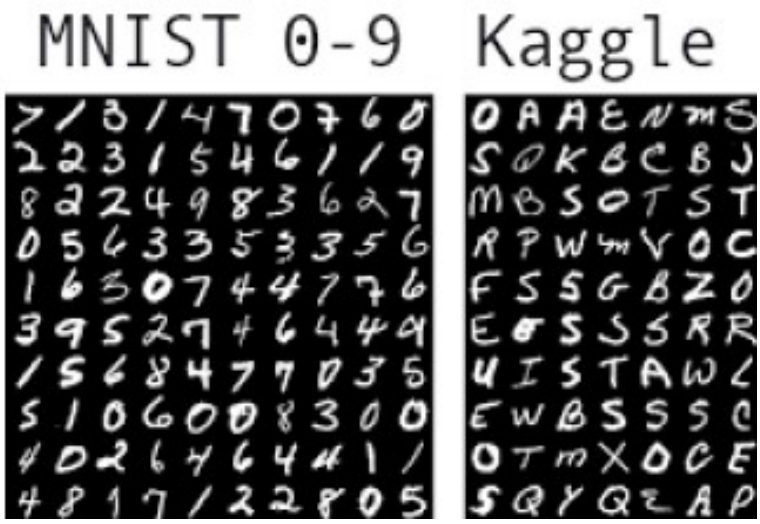
It consists of an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, is connected to another and has a weight and threshold associated with it. If the output of any particular node exceeds the threshold value, that node is activated and sends data to the subsequent network layer. Otherwise, no data is transmitted to the subsequent network layer.

High data uncertainty due to the various qualities of each person's handwriting is one of the challenges in handwriting recognition, (Surya Nath et al., 2015). The efficacy of handwriting recognition is dependent on the feature extraction and classification approach, as stated. Nowadays, this achieved by using deep learning, that is using neural network algorithms, but with more layers, (Techopedia, 2022). Convolutional Neural Network (CNN) is a form of Neural Network designed primarily for processing data using a grid-shaped architecture, (Goodfellow et al., 2016). CNN is said to be the finest model for dealing with object detection and recognition. In the case of particular datasets, digital picture recognition accuracy can rival that of humans, (Coates et al., 2011). (Trnovszky et al., 2017), when CNN was compared to other categorization algorithms, it produced the best results, with a 98 percent accuracy rate. It demonstrates that the CNN approach is particularly suitable for picture categorization. Our aim in this research is to construct a neural network first and then develop it into a CNN to do handwriting recognition

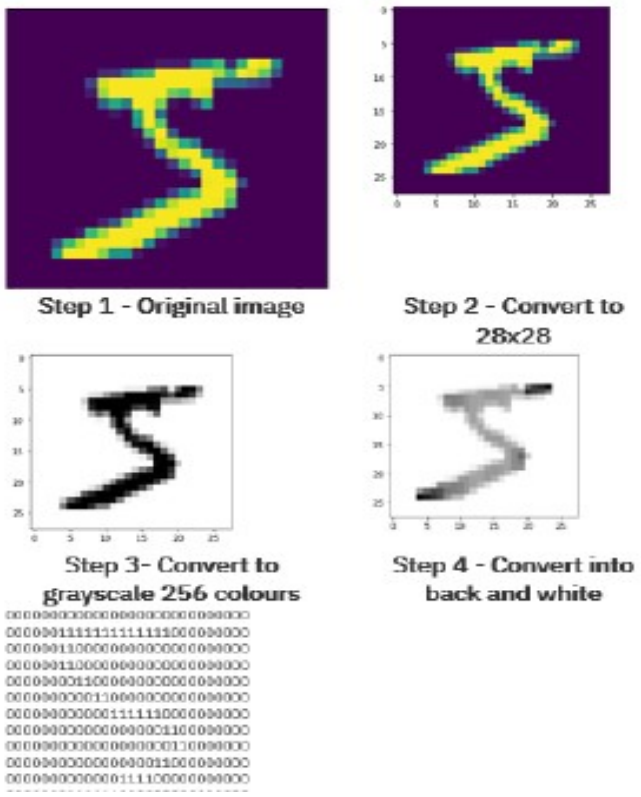
3. Methodology
This section explains the design and architecture of the proposed neural network- based handwritten character recognition system.

3.1 Dataset Selection & Image Pre- processing
The researchers chose the MINST Dataset for the model. It contains 60,000 training photos and 10,000 testing

images of handwritten digits from 0 to 9. The images of handwritten numerals are encoded as a 28x28 matrix with grayscale pixel values in each cell, (LeCun et al., 2022). Additionally, we will explore the Kaggle AZ dataset, which consists of 26 folders (A-Z) containing handwritten images in a 28x28 matrix with grayscale storage for each image. Figure 1.2 shows the Screenshot of the Dataset



Before doing recognition, all images from the dataset were resized and converted from grayscale to black and white. This is due to the fact that grayscale images contain a variety of shades of grey and permit a range of pixel values. Black-and-white graphics only include pixel values of 2 (0 and 1) and nothing in between. And also to check that all images are in a 28x28 pixel matrix. Figure 1.3 displayed the Image Pre-Processing.



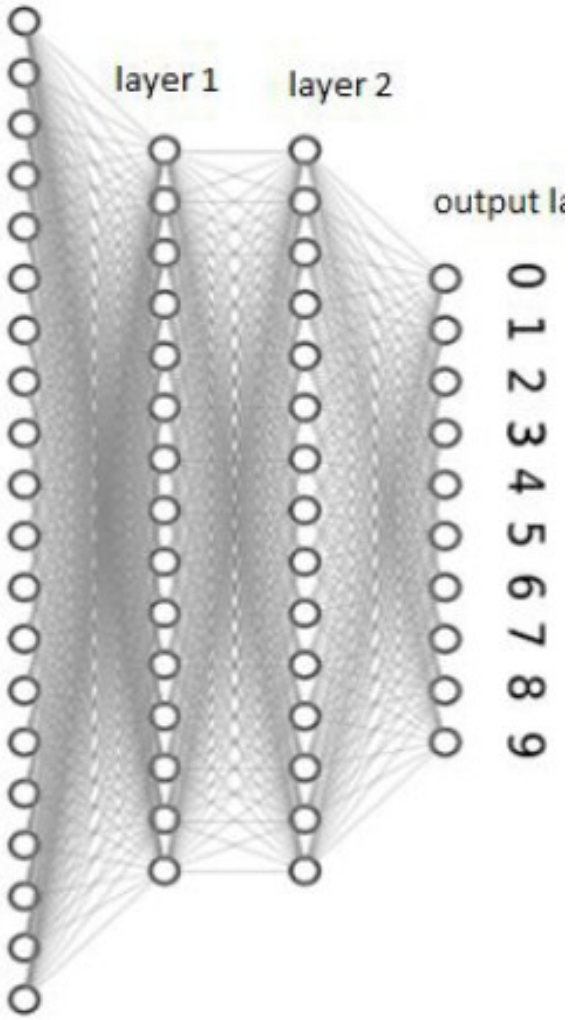
3.2 Neural Network Architecture

Firstly, the neural network-based handwriting recognition system was to be developed. Different neural network architectures can be used to generate distinct outcomes from handwriting image inputs, because designs are

based on certain parameters, data, and training periods. The size of a layer in a deep neural network depends on the predicted workload of the system.

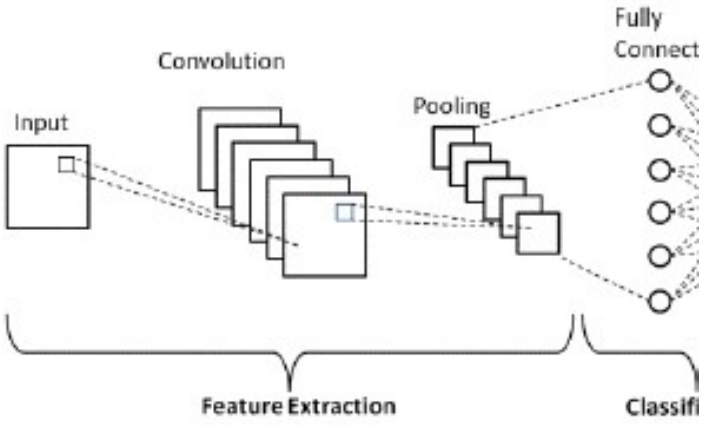
The **Input layer** will have 784 nodes (due to our images being of size 28x28 = 784 pixels)
The **layer 1 and layer 2** will have each 512 nodes
The **output layer** for digits will have 10 nodes (to recognize 0 to 9)

Figure 1.4 displays the Neural Network for Character Recognition Systems. This neural network was implemented and trained as well as tested with the MNIST dataset to check for accuracy and speed.



3.3 Convolutional Neural Network (CNN)

In the second stage, the CNN was used with the proposed aim of increasing the accuracy of the handwriting recognition system. The CNN was the mathematical construct that is typically composed of five types of layers (or building blocks): convolution, pooling, fully connected layers, ReLU correction, and dropout (Yamashita et al., 2018). The description of the CNN architecture is displayed in Figure 1.5.

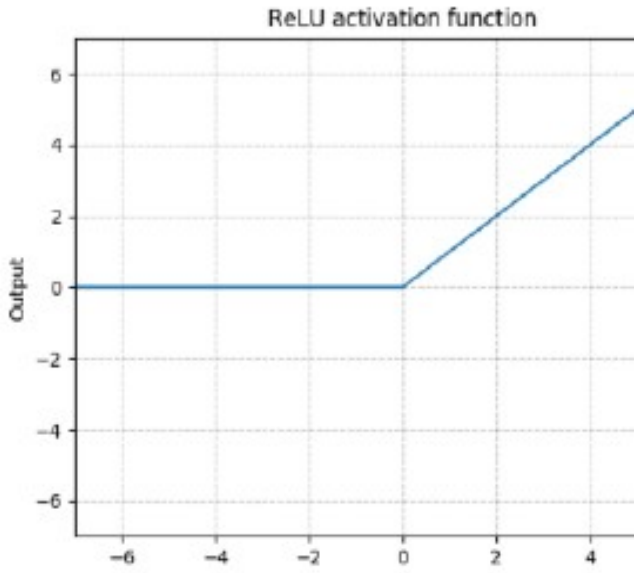


The convolution layer is where most of the computation occurs. It requires an input layer, a filter, and a feature map.

Pooling layer: The pooling layer substitutes network output at certain points by calculating a summary statistic of surrounding outputs. This aids in lowering the spatial dimension of the representation, which reduces the amount of computation and weights necessary, (Mishra, 2020).

Fully connected layer: This comprises of the neurons, as well as the weights and biases, and is used to link the neurons between two separate layers. These layers are often placed prior to the output layer and constitute the final few levels of a CNN design. The input picture from the preceding layers is flattened and supplied to the FC layer in this step. The flattened vector is then sent via a few additional FC levels, where the mathematical function operations are often performed. The categorization procedure begins at this point, (Gurucharan, 2020).

Activation layer: The activation function is a critical element in the CNN model. It is used to learn and estimate any type of continuous and complicated relationship between network variables. There are various popular activation functions, including the ReLU, Softmax, TanH, and Sigmoid functions. Each of these functions has a distinct purpose. For a binary classification CNN model, the sigmoid and softmax functions are favored, whereas softmax is typically employed for multi-class classification, (Gurucharan, 2020).



3.4 System Architecture

The system consists of five phases. The phases are loading & preparing dataset, image pre-processing, creation

of neural network, train the model, test & evaluate the model.

A. Loading & Preparing Dataset

This step involves acquiring the dataset and loading it into the model. The dataset is then divided into training and a test dataset. For the MNIST dataset, 60000 images are for training and 10000 images are for testing, representing digits.

B. Image Pre-Processing

After the dataset has been loaded, the images are converted into grayscale image from color images. Then normalize the data for training, which is converting the value ranges from [0-255] to [0-1]. The resulting data will then be flattened to be represented as matrices for faster processing.

C. Creation of Neural Network

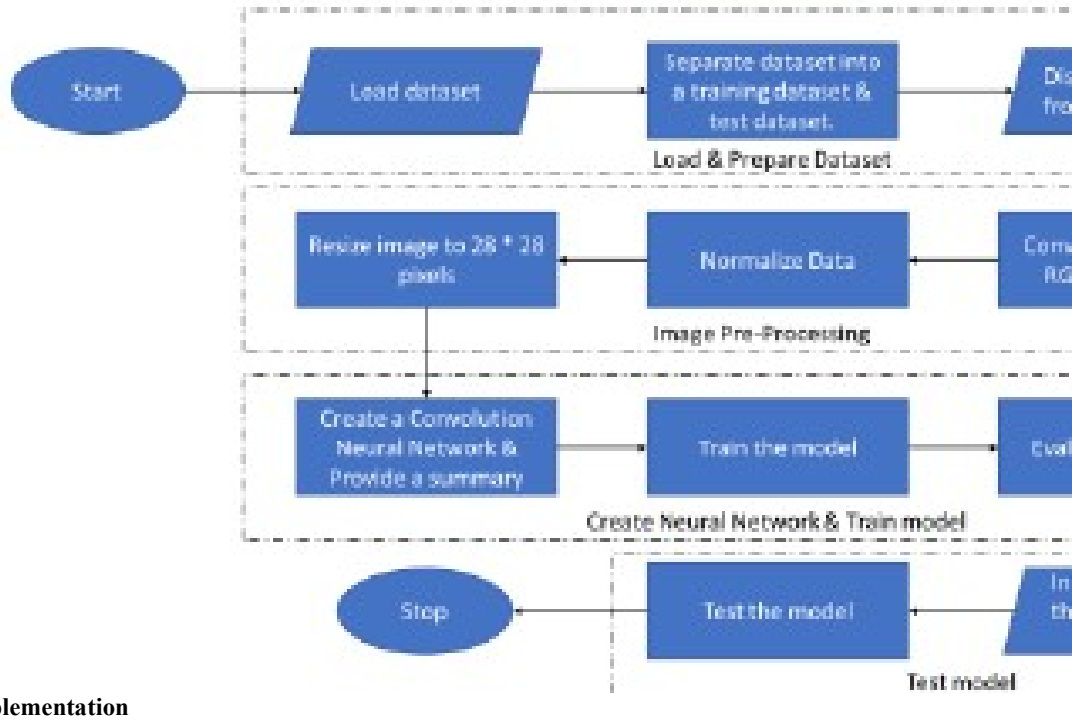
TensorFlow was used to build the neural network in the Python programming language, using Anaconda as the main development environment, (Vaughan, 2018). TensorFlow is an open-source AI library that enables the construction of data flow graphs for model creation. Keras, a Python-based API for deep learning that runs on top of TensorFlow was also utilized. In addition, the Matplotlib and Seaborn were used to plot the graphs and generate images from the datasets. The source code for all this is provided in Notes.

D. Training the model

After creating the neural network, the model was trained with the dataset that was uploaded earlier. From the training dataset that have been created, 30% of the training dataset was used for validation for up to 25 epochs (One epoch means all data processed one times). The aim was to increase accuracy and decrease processing times for each model. After training the model, we need to evaluate them.

E. Test the model

After training the model, both images that are in and outside the dataset are used to train the model. This is done in order to check if the model can accurately predict the character from any kind of image inserted. Figure 1.7 displays the flowchart for the architecture of the system



4. Implementation

Two different models were developed. The first one model was using Simple Neural Network and the second one using Convolution Neural Network. We are going to show the working of the models in this section.

4.1 Code for loading the data set

In Figure 4.1, the libraries for Numpy, Tensorflow and Keras were imported, and then the MNIST dataset was loaded from its source. The subsets that will be used for training and testing were also set.


```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load
len(x_train)
# 60000
len(x_test)
# 10000
# Finding the shape of individual sample
x_train[0].shape
```

4.2 Normalizing and flattening the dataset

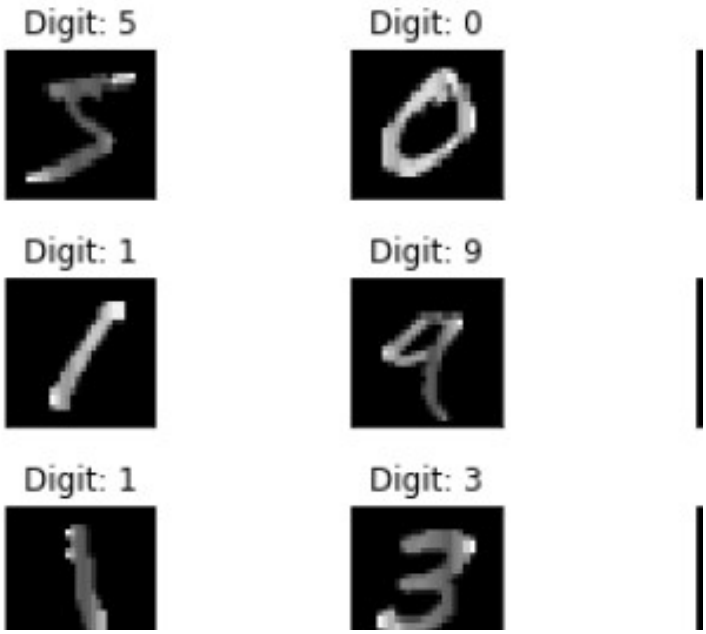
In Figure 4.2, the photos were split by 255 to turn them black and white (0 and 1), then transform the two-dimensional input data to a single-dimensional format for feeding into the model. This is accomplished by a technique known as flattening. The 28x28 grid picture is turned into a 784-dimensional single-dimensional array throughout this operation (28x28).

```
x_train.shape

# Scale the data so that the values are from 0 - 1
x_train = x_train / 255
x_test = x_test / 255
x_train[0]

# Flattening the train and test data
x_train_flattened = x_train.reshape(len(x_train), 28*28)
x_test_flattened = x_test.reshape(len(x_test), 28*28)
x_train_flattened.shape
```

4.3 Exploring the dataset

Figure 4.3, displays the code that was used to display the first 9 images in the dataset after converting them to black-and-white; the output is given in Figure 4.4

```
fig = plt.figure()
for i in range(9):
  plt.subplot(3,3,i+1)
  plt.tight_layout()
  plt.imshow(x_train[i], cmap='gray', interpolation='none')
  plt.title("Digit: {}".format(y_train[i]))
  plt.xticks([])
  plt.yticks([])
fig
```

The pixel distribution in each image could also be of interest to optimize our model further. The code for this is given in Figure 4.5 and the output as histogram for number 5 is given in Figure 4.6

```
fig = plt.figure()
plt.subplot(2,1,1)
plt.imshow(x_train[0], cmap='gray', interpolation='none')
plt.title("Digit: {}".format(y_train[0]))
plt.xticks([])
plt.yticks([])
plt.subplot(2,1,2)
plt.hist(x_train[0].reshape(784))
plt.title("Pixel Value Distribution")
```

Figure 4.5 - Code use to output a histogra

4.4 Build a Simple Neural network

The code provided in Figure 4.7 shows a simple neural network with 784 input nodes, one inner layer and out 10 output nodes. For evaluation purposes, we have added an optimization layer at a later stage.

```
# Sequential create a stack of layers
model = keras.Sequential([
    keras.layers.Dense(10, input_shape=(784,), activation='sigmo
])

# Optimizer will help in backproagation to reach better global o
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

Figure 4.7 – Simple Neural Netwo

The model summary for this simple neural network is provided in Figure 4.8


```
# Does the training
model.fit(x_train_flattened, y_train, epochs=25)
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              
=================================================================
 dense_6 (Dense)             (None, 10)                
=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
```

4.5 Netw

```
# Does the training
model.fit(x_train_flattened, y_train, epochs=25)
```

Figure 4.9 - Training the model

To 4.9 and output in Figure 4.10

```
Epoch 1/25
1875/1875 [==============================] - 4s 960us/step - loss: 0.4652 -
Epoch 2/25
1875/1875 [==============================] - 2s 1ms/step - loss: 0.3035 - a
Epoch 3/25
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2838 - a
Epoch 4/25
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2729 - a
Epoch 5/25
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2667 - a
```

In Figure 4.11, displays the code used to evaluate the model. An accuracy of 93% with the test data (Figure 4.12) was achieved.

```
model.evaluate(x_test_flattened, y_test)
```

Figure 4.11 Model Evaluation

```
313/313 [==============================] - 0s 1ms/step - loss:
acy: 0.9267

[0.27218154072761536, 0.9266999959945679]
```

4.6 Confusion matrix

This is an appropriate technique to evaluate the effectiveness of our models as it gives an idea of overall accuracy for various digits. In the Figure 4.13, we are showing the code used create the confusion matrix for validation and precision.

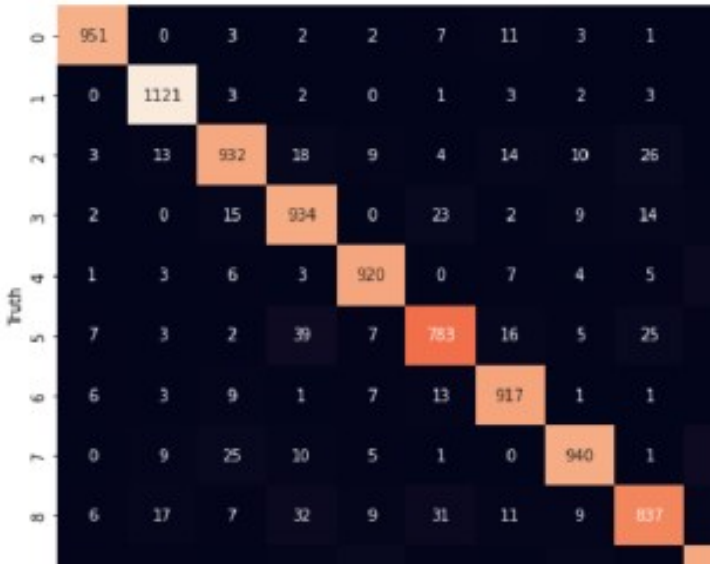
```
cm = tf.math.confusion_matrix(labels=y_test, predictions=y_predi
cm
```

The output is given in Figure 4.14, and it is further plotted graphically with code in Figure 4.15 and output in Figure 4.16.


```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 951,    0,    3,    2,    2,    7,   11,    3,
         [   0, 1121,    3,    2,    0,    1,    3,    2,
         [   3,   13,  932,   18,    9,    4,   14,   10,
         [   2,    0,   15,  934,    0,   23,    2,    9,
         [   1,    3,    6,    3,  920,    0,    7,    4,
         [   7,    3,    2,   39,    7,  783,   16,    5,
         [   6,    3,    9,    1,    7,   13,  917,    1,
         [   0,    9,   25,   10,    5,    1,    0,  940,
         [   6,   17,    7,   32,    9,   31,   11,    9,
```

```
import seaborn as sn
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Figure 4.15 - Code to plot Confusion



4.7 Convolution Neural Network

To build the second model, the CNN, the code is provided in Figure 4.17 and the model summary is in Figure 4.18.

.


```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D,
# Creating the network
model = Sequential()
### First Convolution Layer
# 64 -> number of filters, (3,3) -> size of each kernal,
model.add(Conv2D(64, (3,3), input_shape = x_train.shape[1:6]))
# For first layer we have to mention the size of input
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
### Second Convolution Layer
model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
### Third Convolution Layer
model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
### Fully connected layer 1
model.add(Flatten())
model.add(Dense(64))
model.add(Activation("relu"))
### Fully connected layer 2
model.add(Dense(32))
model.add(Activation("relu"))
### Fully connected layer 3, output layer must be equal to number of classes
model.add(Dense(10))
model.add(Activation("softmax"))
```

```
Training Samples dimension (60000, 28, 28, 1)
Testing Samples dimension (10000, 28, 28, 1)
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 64)

 activation (Activation)     (None, 26, 26, 64)

 max_pooling2d (MaxPooling2D  (None, 13, 13, 64)
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)

 activation_1 (Activation)   (None, 11, 11, 64)

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)
 2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 64)

 activation_2 (Activation)   (None, 3, 3, 64)

 max_pooling2d_2 (MaxPooling  (None, 1, 1, 64)
 2D)

 flatten (Flatten)           (None, 64)

 dense (Dense)               (None, 64)

 activation_3 (Activation)   (None, 64)

 dense_1 (Dense)             (None, 32)

 activation_4 (Activation)   (None, 32)
```

After being trained, the accuracy of the model is also provided in Figure 4.19. We can see that we have achieved a higher accuracy of 98% with the CNN model

```
Epoch 10/10
938/938 [==============================] - 21s 22ms/step - loss: 0.0226 -
0.9797
313/313 [==============================] - 1s 4ms/step - loss: 0.0702 - ac
Test Loss on 10,000 test samples 0.07016754150390625
Test Accuracy on 10,000 test samples 0.9829000234603882
```

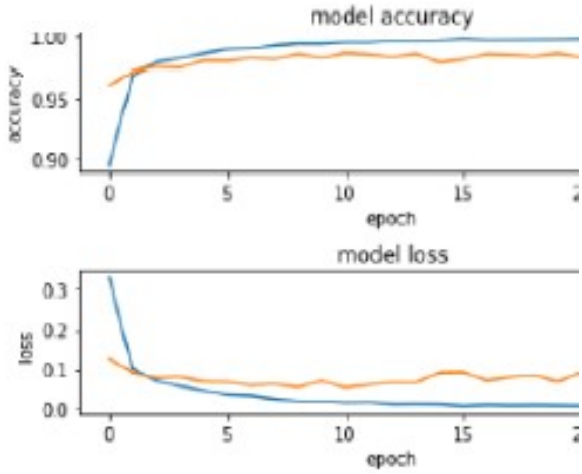
Fig 4.19 Accuracy of CNN Mod

4.8 Evaluating our model graphically

We have developed additional coding in Figure 4.20 to estimate the accuracy and loss in our models based on number of runs.

```
fig = plt.figure()
plt.subplot(2,1,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower right')

plt.subplot(2,1,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')

plt.tight_layout()
```

The output is given in Figure 4.21



4.9 Loading External to test with our model

Figure 4.22 provides the code which allows taking an externally scanned image and then testing it with our model. The results for these will be discussed in the testing section.

```
import cv2
img = cv2.imread('test1.jpg')
plt.imshow(img)
img.shape
# Converting to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray.shape
plt.imshow(gray)
# Resizing to a 28x28 image
# Please note my image was already in correct dimension
resized = cv2.resize(gray, (28,28), interpolation = cv2.
resized.shape
# 0-1 scaling
newimg = tf.keras.utils.normalize(resized, axis = 1)
# For kernal operations
newimg = np.array(newimg).reshape(-1, IMG_SIZE, IMG_SIZE
newimg.shape
#making predictions
predictions = model.predict(newimg)
print(np.argmax(predictions[0]))
```


5. Testing our model

In this section several tests were conducted for models with the aim of evaluating the effectiveness and accuracy.

5.1 Comparing the 2 models

Test Objectives: To compare the results obtained by running both models (simple neural network and convolutional neural network) and evaluate their effectiveness and accuracy.

Expected Results: A prediction that CNN shall perform better comparing it to Simple Neural Network (SNN) in handwriting recognition.

Actual Results: This is discussed below

The results, given in Figures 5.1 and 5.2 confirm that CNN is better than SNN, the former offering an accuracy of 98%, as compared with 92% for the latter. Also, the % loss is lower for CNN.

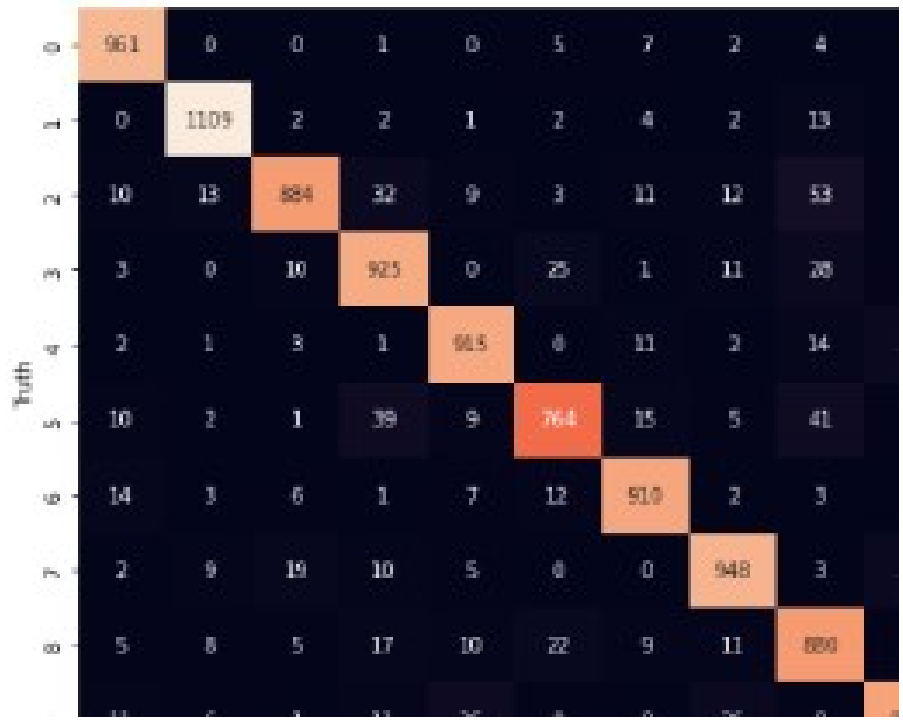
```
0s 616us/step - loss: 0.2839 - accuracy
```

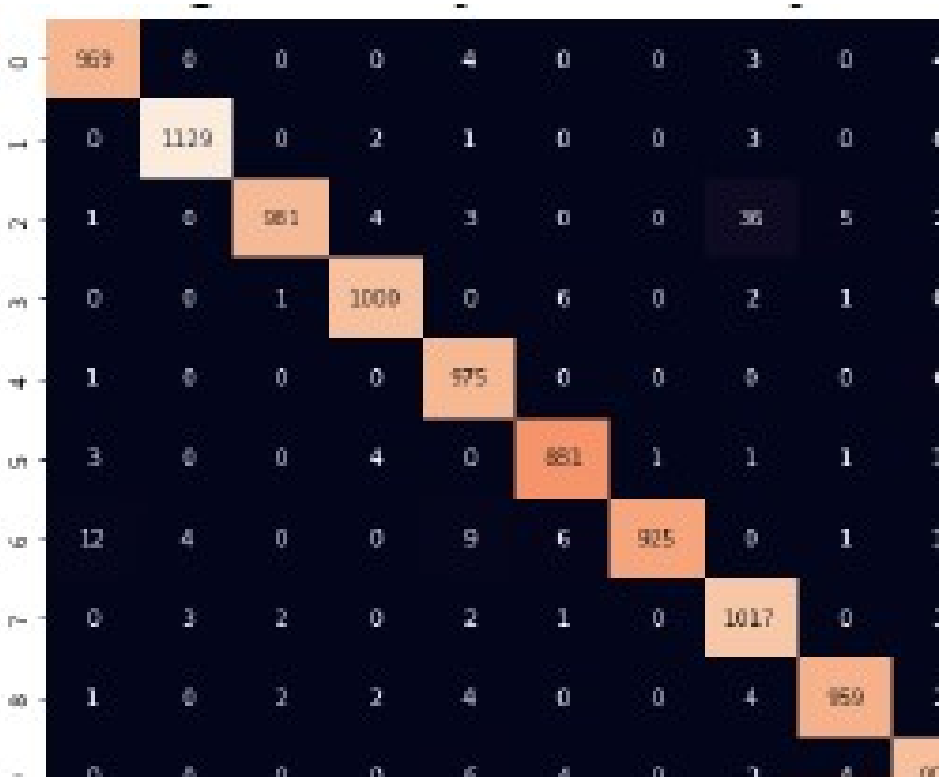
Fig 5.1 Accuracy results for SN

```
Test Loss on 10,000 test samples 0.070167541
Test Accuracy on 10,000 test samples 0.98290
```

Fig 5.2 Accuracy results for C

Furthermore, the confusing matrix for all digits [0-9] provided in Figure 5.3 and 5.4 also confirm the fact that CNN is much better than SNN. For example, the number "3" was correctly predicted 1000 times by CNN, but only 925 times by SNN. Also, the SNN wrongly predicted "3" to be "2" 10 times, whereas CNN did it only once. So, the overwhelming conclusion is to use CNN for handwriting recognition





5.2 Performance testing

Test Objectives: To train the CNN model with the CPU only and then with GPU enabled to compare the performance results to see how much time it takes in each case. We are going to set the model with 25 epochs

Expected Results: To see an improvement in the performance of the model with GPU enabled.

Actual Results: This is discussed below

With CPU only enabled on an AMD Ryzen, the model executed all 25 epochs in 750 seconds (12 minutes and 30 seconds) in Figure 5.5 and 5.6, with an average of 3ms per step.

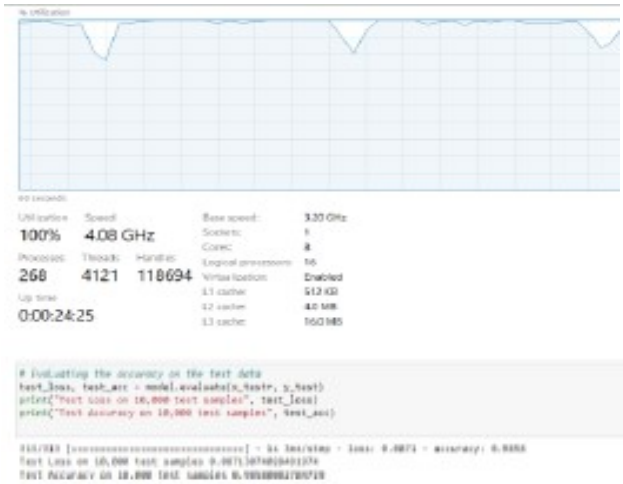
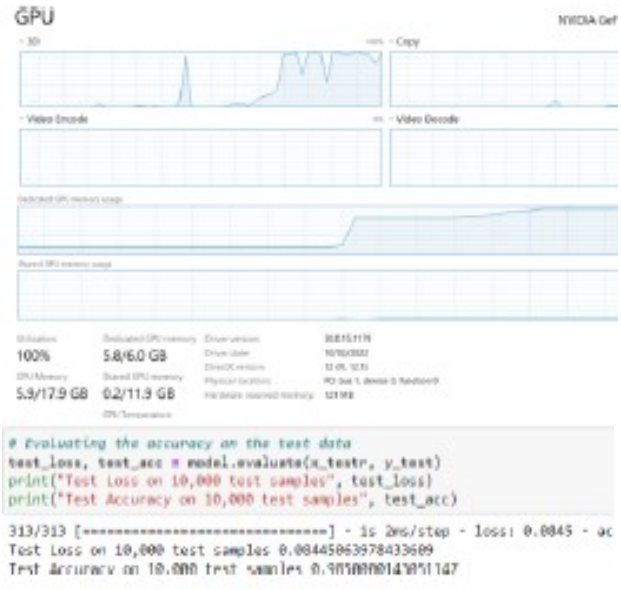


Figure 5.6 Model Evaluation for GP

With a less powerful CPU or we were going to train the model with more epochs or more images, it would have taken more time. As for accuracy, it is around 98.5%. Figures 5.7 and 5.8 confirm that with GPU enabled, our model to execute all 25 epochs took 108 minutes (1 minutes and 48 seconds), with an average of 2 ms per step. The accuracy is not changed at 98.5%. So, we can conclude that better hardware, such as addition of GPU or multiprocessing, is bound to improve the performance of a CNN.

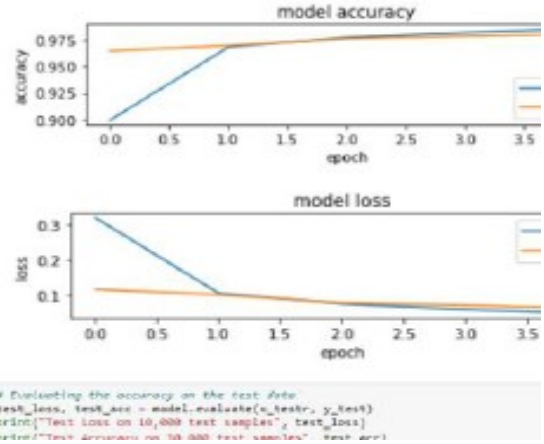


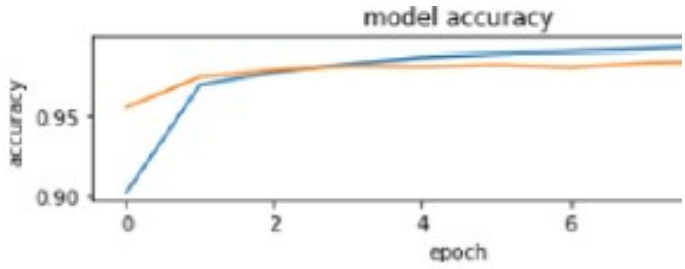
5.3 Training Epochs, Accuracy and Loss

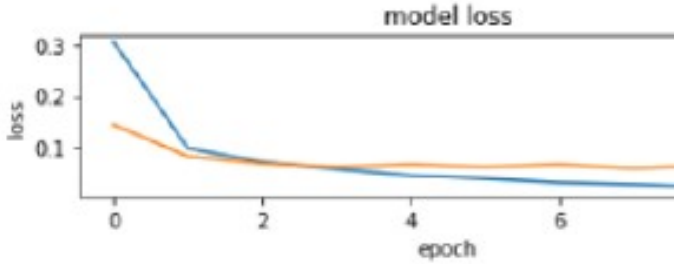
Test Objectives: To train the CNN model with 5, 10 and then 25 Epochs in order to investigate how this can affect the accuracy and loss in the model.

Expected Results: They should be an increase in the model accuracy from the beginning of the test, till the end.

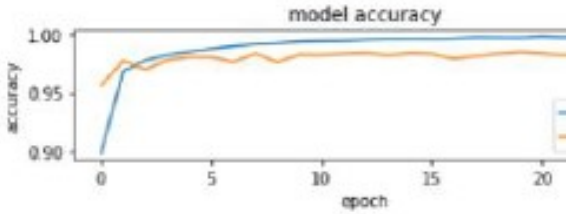
Actual Results: The results are displayed in Figure 5.9 to 5.11 and discussed.

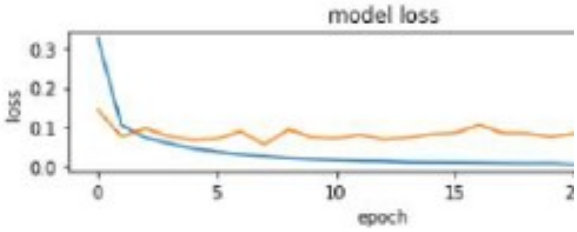







```
# Evaluating the accuracy on the test data
test_loss, test_acc = model.evaluate(x_testr, y_test)
print("Test Loss on 10,000 test samples", test_loss)
print("Test Accuracy on 10,000 test samples", test_acc)
```






```
# Evaluating the accuracy on the test data
test_loss, test_acc = model.evaluate(x_testr, y_test)
print("Test Loss on 10,000 test samples", test_loss)
print("Test Accuracy on 10,000 test samples", test_acc)
```

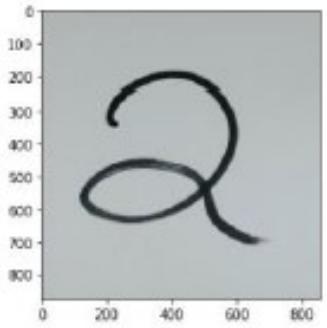
A summary of the results is presented in table 5.1. We can therefore conclude that the number of runs that must be conducted on a neural network model has an impact on its accuracy and hence its effectiveness to be used for business applications.

Epochs	5	10	
Accuracy	98.04%	98.46%	9
Loss	7.84%	5.39%	7.

5.4 Testing the model with external data

We have a scanned a number of handwritten numbers in order to load and test them with our CNN model by following the procedure shown (i) scanning of a handwritten text (Figure 5.12), transforming the format in Python and predicting with our model (Figure 5.13) and then loading in our model for prediction.

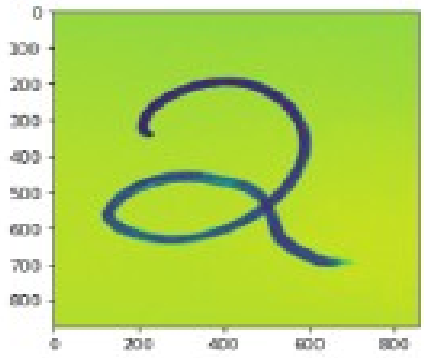
```
In [32]: import cv2
         img = cv2.imread('test1.jpg')
         plt.imshow(img)
Out[32]: <matplotlib.image.AxesImage at 0x196521
```




```
In [33]: img.shape
         # (28, 28, 3)
```

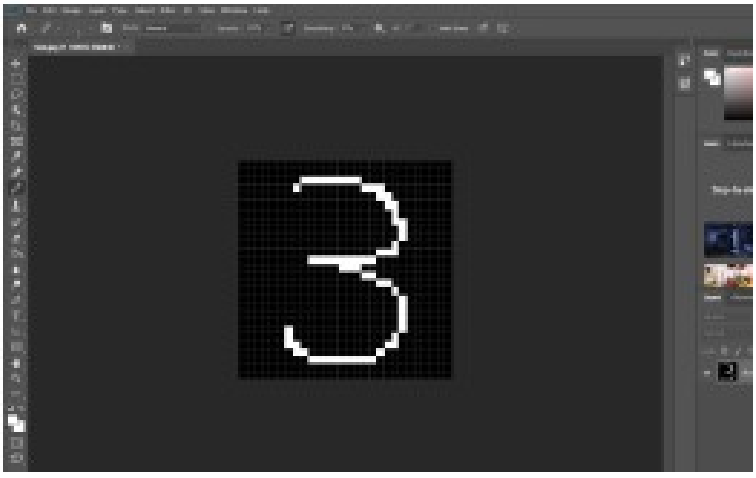


```
In [34]: # Converting to grayscale
         gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
         gray.shape
         plt.imshow(gray)

         # Resizing to a 28x28 image
         # Please note my image was already in correct dim
         resized = cv2.resize(gray, (28,28), interpolation
         resized.shape

Out[34]: (28, 28)
```



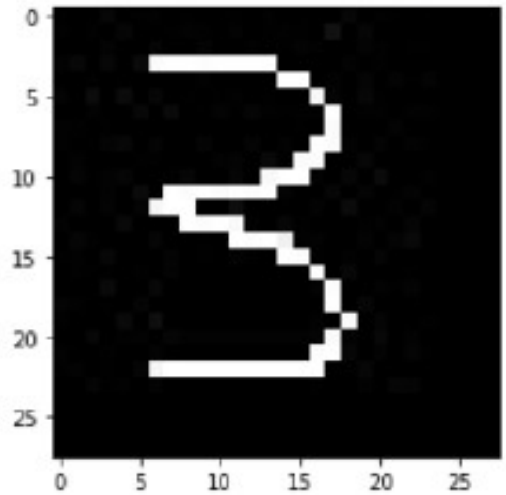

```
In [35]: # 0-1 scaling
         newimg = tf.keras.utils.normalize(resized, axis =

         # For kernal operations
         newimg = np.array(newimg).reshape(-1, IMG_SIZE, I
         newimg.shape

Out[35]: (1, 28, 28, 1)
```

We have also performed other types of tests like writing in Ms Paint, converting the data in Photoshop (Figure 5.14), loading and transforming the data to prepare (Figure 5.15) and predicting the result (Figure 5.16) and seen that the model is working fine



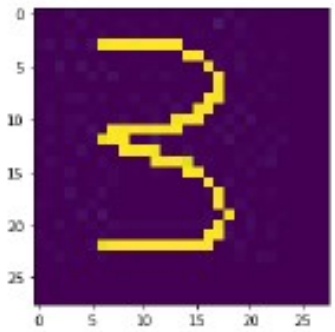

```
import cv2
img = cv2.imread('test.jpg')
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x20bae
```




```
# Converting to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray.shape
plt.imshow(gray)

# Resizing to a 28x28 image
# Please note my image was already in correct dimension
resized = cv2.resize(gray, (28,28), interpolation = cv2.INTER_
resized.shape
```

```
(28, 28)
```




```
# 0-1 scaling
newimg = tf.keras.utils.normalize(resized, axis = 1)

# For kernal operations
newimg = np.array(newimg).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
newimg.shape
```

6. Critical Analysis

One of the main drawbacks of this research paper is that it has focused essentially on the use of neural networks for handwriting recognition. It has not looked at other possibilities such as Hidden Markov Models or Support Vector Machines to achieve the same objectives.

Another aspect to consider is the accuracy we have obtained. 98.72% (in Table 5.1) seems a very good result. But is it good enough to be used in a banking environment, which is one of the examples of applications areas discussed in our introduction section? Whereas this accuracy could be good for general purpose applications such as object detection or recognition, it could be largely insufficient in banking environment. Imagine a cheque is drawn for Rs 2000 – our handwriting system based on CNN could interpret this as Rs 7000 in 36 out of 1000 cases (based on our results in Figure 5.4), with total potential loss of Rs up to Rs 180,000 for 36 bank customers. Clearly, this is not acceptable. Our handwriting recognitions require further refinement and validation to be used in commercial environments.

The other issue is also that our recognition system model works only on the MNIST dataset and was not implemented for the Kaggle dataset with Latin alphabets. But it must be pointed out that a similar approach could be used for letters as the one we did for numbers. In the same way, recognition systems for other languages like Arabic or Hindi could be implemented.

Furthermore, our recognition system works only for individually scanned numbers. We have not developed a system what will take a document, identify the numbers, break them down into individual images and then pass them to the model for recognition. But this could be a way forward to enhance the system by developing such functions.

Another drawback is that our model takes into account only the recognition of static images. With the use of mobile phones, tablets and lately touch screens on laptops, there will be an increase in dynamic recognition of handwriting where the system will also need to follow hand movements to identify the letter or number being written. This would be a future area of research.

The CNN model we have built could further having fine-tuned by using different types of activation functions, adding more internal layers or increasing the number of epochs in order to increase accuracy obtained. Also, we could have developed models based on other libraries such as Pytorch. The dataset could have been extended to contain much more than 60000 images of numbers, given the various ways in which human being write them.

Finally, our CNN model is an example of deep learning. The main criticism against the latter is that it is not sufficiently transparent (a black box). One might want to understand how a decision was picked, understand the extent to which there's bias, especially if it could lead to important decisions. Deep learning is not well integrated with prior knowledge. Furthermore, problems that require common sense are not yet within reach for deep learning. For example, if a customer is making a written bank transfer of Rs 2000 every month, the CNN will not find anything unusual if it wrongly recognizes a transfer of Rs 70000 for a specific month, whereas common sense would indicate otherwise.

7. Conclusion

In this research work, we have been able to study different learning models and have been able to design, develop, implement, deploy and test a convolutional neural network to recognize handwriting. We have used the MNIST dataset, explored various models and concluded that a CNN running on well-tuned hardware with GPU,with sufficient training data can deliver accuracy up to 98.7% to recognize numbers. The model can further be enhanced in terms of accuracy and speed if we increase the dataset, increase the number of epochs runs and execute it on parallel hardware. Using such techniques, some researchers have been able to achieve up to 99.89% of accuracy. But there is further scope for improvement.

8. References

1. Vadapalli, P. (2021,February 9). Biological Neural Network: Importance, Components & Comparison. https://www.upgrad.com/blog/biological- neural-network/
2. Education, I. C. (2020, August 17). What are Neural Networks? What Are Neural Networks?| IBM; www.ibm.com. https://www.ibm.com/cloud/learn/neural- networks/
3. Hamad, K., & Kaya, M. (2016). A Detailed Analysis of Optical Character Recognition Technology. international Journal of Applied Mathematics, Electronics and Computers, 4(Special Issue-1), 244–249. https://doi.org/10.18100/ijamec.270374/
4. Surya Nath, R. S., Afseena, S., & International Journal of Scientific and Research Publications. (2015). Handwritten Character Recognition – A Review. Handwritten Character Recognition – A Review, 5(3), 1–6. http://www.ijsrp.org/research-paper- 0315/ijsrp-p3996.pdf
5. Techopedia. (2022, February 23). What is Deep Learning? - Definition from Techopedia. Techopedia.Com; www.techopedia.com. https://www.techopedia.com/definition/30325/deep-learning/
6. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning (Adaptive

Computation and Machine Learning series) (Illustrated ed.). The MIT Press.

7. Coates, A., Lee, H., & Andrew, Y. N. (2011). An Analysis of Single-Layer Networks in Unsupervised Feature Learning. An Analysis of Single-Layer Networks in Unsupervised Feature Learning, 1–9. https://cs.stanford.edu/~acoates/papers/coate sleeng_aistats_2011.pdf
8. TRNOVSZKY, T., KAMENCAY, P., ORJESEK, R., BENCO, M., & SYKORA, P. (2017). Animal Recognition System Based on Convolutional Neural Network. DIGITAL IMAGE PROCESSING AND COMPUTER GRAPHICS, 15(3), 1–9. https://pdfs.semanticscholar.org/68ce/c9c6572abae2fddde2ed47785df24eba1713.pdf
9. LeCun, Y., Cortes, C., & J.C. Burges, C. (2022, 0 0). THE MNIST DATABASE of handwritten digits. THE MNIST DATABASE of Handwritten Digits; yann.lecun.com. http://yann.lecun.com/exdb/mnist/
10. Yamashita, R., Nishio, M., Gian Do, R. K., & Togashi, K. (2018,June 22). Convolutional neural networks: an overview and application in radiology - Insights into Imaging. SpringerLink; link.springer.com. https://link.springer.com/article/10.1007/s13244-018-0639-9
11. Mishra, M. (2020, September 2). Convolutional Neural Networks, Explained | by Mayank Mishra | Towards Data Science. Medium; towardsdatascience.com. https://towardsdatascience.com/convolution al-neural-networks-explained-9cc5188c4939
12. Gurucharan, M. K. (2020, December 7). Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network | upGrad blog. upGrad Blog; www.upgrad.com. https://www.upgrad.com/blog/basic-cnn- architecture/#3_Fully_Connected_Layer
13. Vaughan, J. (2018, February 1). What is TensorFlow? - Definition from https://www.techtarget.com/searchdatamana gement/definition/TensorFlow