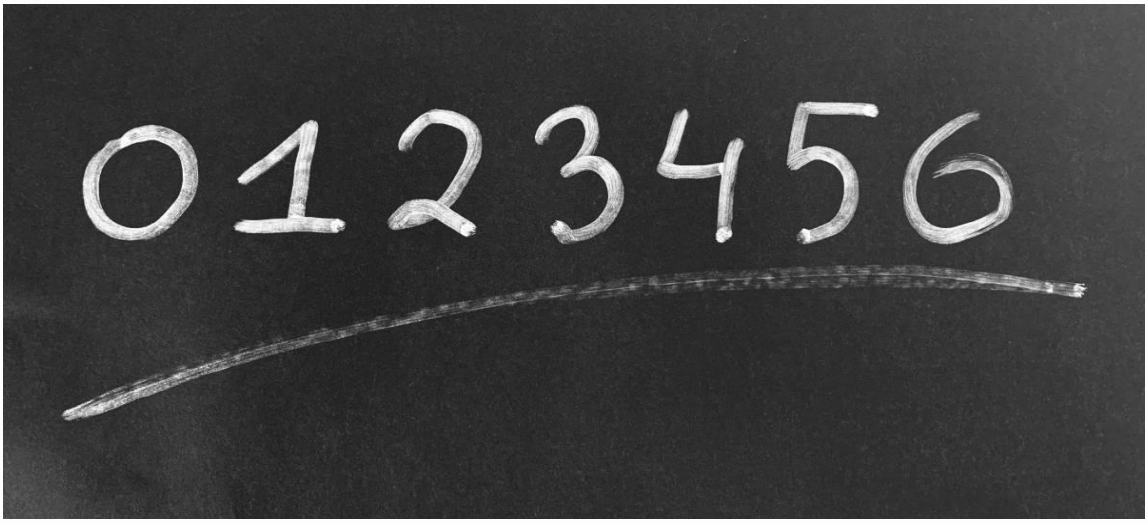
Handwritten Digit Recognition

DECEMBER 2021



By-

DEVENDRA SINGH CHAUHAN

ABSTRACT

Technology has advanced so quickly in recent years that we have become accustomed to it. Handwritten Digit Recognition has recently gained importance and many academics are interested in it because of its applications in Machine Learning. Furthermore, Digit Recognition is an impressive and vital topic. Multiple challenges must be considered in this project such as handwritten digits do not have the same thickness, size, location, or orientation. It may be used for a variety of things, including pre-programmed bank checks, postal location and legal paperwork.

All information was stored in written form before computers were invented. This is a very inefficient form of storage because paper information cannot be kept for very long periods of time and can be lost or destroyed. On the other hand, information on a computer is securely stored for a long time, and multiple copies of the same information can be made with ease. Thus, after the computers were invented, a lot of money was spent on manual labor to convert paper information to digital information rather than human intervention or manual labor. Machine Learning can be used to identify and convert this paper information into digital information.

The goal of our project is to develop a real-time application that uses CNN to recognize and classify handwritten digits of varied sizes and orientations. Machine Learning Algorithms like SVM, Random Forest Classifier, KNN, and CNN can also be used to execute this notion with varied degrees of accuracy.

Table of Contents

List of Figure

List of Tables

List of Abbreviations

CNN	Convolutional Neural Network
CCNN	Customized Convolutional Neural Network
FCL	Fully Connected Layer
KNN	K Nearest Neighbour
MNIST	Modified National Institute of Standards and Technology database
MLP	Multilayer perceptron
RFC	Random Forest Classifier
ReLU	Rectified Linear Unit
SVM	Supervised Vector Machine

1. INTRODUCTION

Image processing and analysis are simple tasks for the human brain. When the eye perceives a certain image, the brain can quickly segment it and understand its many components. The brain goes through that process automatically, which includes not only the examination of the visuals, but also the interpretation of those images. Handwriting recognition plays a vital part in information processing in today's digital world. On paper, there is a huge amount of information, and processing digital files is less expensive than processing hardcopy. A handwritten recognition software's goal is to convert handwritten characters into formats that are machine readable. Vehicle license plate identification, postal letter sorting, and historical document preservation in archaeological departments, old document automation in libraries and banks, and other applications are among the most common.

Machine Learning provides a variety of approaches for reducing human effort in detecting manually written digits. Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by doing. Human efforts in perceiving, learning, understanding, and a variety of other areas can be reduced with the use of deep learning algorithms. The computer learns to do classification tasks from images or the text of any document using deep learning. Deep Learning models can achieve state-of-the-art accuracy that is superior to human performance. To train and recognize handwritten digits from multiple sources, the digit recognition technology allows use of large datasets.

The goal of the proposed research is to obtain equivalent accuracy utilizing a pure CNN architecture for MNIST digit recognition by investigating the learning parameters in CNN architecture. Another goal is to look into the role of various hyper-parameters and fine-tune hyper-parameters that are important for boosting CNN architecture performance.

The main objective of our project is to apply the Convolutional Neural Network to recognize handwritten digits that have been trained on the MNIST dataset, which is a big collection containing handwritten images of digits 0-9 and finally build an interface for the user to use the notion as real time.

2. BACKGROUND STUDY

2.1 LITERATURE REVIEW

People are increasingly focusing on computer skills rather than developing exceptional handwriting skills. One explanation is because the internet and applications are getting smarter than they were previously. Anuj Dutt proved in his research that by applying Deep Learning systems, he was able to achieve extraordinarily high levels of accuracy. He achieved a 98.72 percent accuracy using the convolutional Neural Network with Keras and Theano as the backend. Furthermore, using Tensorflow to execute CNN yields a dramatically better result of 99.70 percent. Despite the fact that the technique and coding appear to be more complicated when compared to traditional Machine Learning algorithms, the precision he achieved is becoming increasingly apparent. Saeed AL-Mansoori released a paper in which he used a Multilayer Perceptron (MLP) Neural Network to recognise and forecast handwritten digits from 0 to 9. On a dataset obtained from MNIST, the suggested neural system was trained and tested.

Many studies on feature extraction and classifier techniques for handwritten digit recognition have been conducted. The majority of them had high recognition accuracy. Mane and Kulkarni (2018), for example, suggested a Customized Convolutional Neural Network (CNN) that can automatically learn features and predict numerical categories in a large data collection, such as Marathi, one of India's most widely spoken regional languages. Furthermore, utilising K-fold cross validation, the CCNN's performance averaged 94.93 percent accuracy. The suggested CCNN model does not set any limitations on the number of layers, but rather optimises it to meet the issue's requirement. The intermediate convolutional layer has also been subjected to the various filter sizes.

Handwritten Recognition from the MNIST dataset is well-known among scientists, as the error rate has been reduced by using different classifiers for various parameters, such as from a linear classifier (1-layer NN) with a 12 percent error rate to 0.23 percent by a board of 35 convolution neural systems. The goal is to build a Handwritten Digit Recognition framework and consider various classifiers and algorithms while focusing on getting as near to human performance as possible. The common issue faced in composing various numbers (0-9) for various persons would be the digit order issue and the closeness between the digits such as 1 and 7, 5 and 6, 3 and 8, 9 and 8, and so on. Additionally, because different persons write the same digit from different perspectives, the originality and variety in their handwriting has an impact on the development and presence of the digits.

2.2 Various Possible Algorithms

These are the various Machine Learning Algorithms that can also be used for the Handwritten Digit Recognition.

2.2.1 K-Nearest Neighbor

KNN is a learning method that is based on instances. The KNN method has two key advantages: it is resilient to noisy training data and it is particularly efficient when dealing with big amounts of data. This technique requires a collection of training datasets with precisely labelled data points in order to function wonderfully. KNN is also a non-parametric classifier. The algorithm considers new data points as its input and performs classification by calculating distance between new and labeled data points using the Euclidean or Hamming distance formulas. Distance is calculated by the Euclidean distance formula.

2.2.2 Random Forest Classifier

RFC is a method of supervised learning. It implies that there is a direct relationship between the total number of trees and the outcome: the more trees there are, the more precise the result will be. This classifier can be used for both classification and regression. If there are enough trees in RFC methods, the classifier will not overfit the model, instead avoiding overfitting concerns. The missing quantities are handled by this classifier. Following the training, predictions are obtained from each individual tree, and the average is determined.

2.2.3 Support Vector Machine

SVM is a supervised learning technique as well. It can also be used for classification and regression. There are data elements that are treated as points in an n-dimensional space in this type of method. By completing classifications between the two classes, this classifier discovers the hyper plane. One of the most important features of this algorithm is that it includes a regularisation parameter that prevents over-fitting.

2.2.4 Convolutional Neural Network

A convolutional neural network (CNN) is a form of artificial neural network that is specifically intended to process pixel input and is used in image recognition and processing. CNNs are image processing, artificial intelligence (AI) systems that employ deep learning to do both generative and descriptive tasks. They frequently use machine vision, which includes image and video recognition, as well as recommender systems and natural language processing (NLP).

CNN image classifications takes an input image, processes it, and categorises it into several groups. An input image is seen by computers as an array of pixels, with the number of pixels varying depending on the image resolution. It will see h x w x d(h = Height, w = Width, d = Dimension) based on the image

resolution. For example, a 6 × 6 x 3 array of RGB matrices (3 refers to RGB values) and a 4 x 4 x 1 array of grayscale matrix images.

To train and evaluate deep learning CNN models, each input image will be sent through a sequence of convolution layers with filters (Kernels), Pooling, fully connected layers (FC), and the Softmax function, which will categorize an item with probabilistic values between 0 and 1. The flow of CNN processing an input picture and classifying objects based on values is depicted in the diagram below.

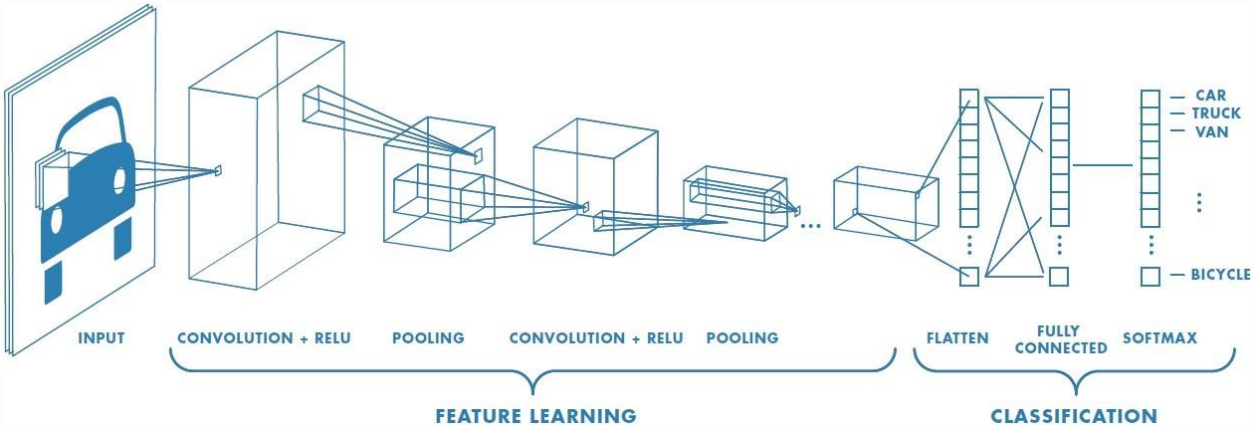


Figure 1.0

The higher performance of convolutional neural networks with picture, speech, or audio signal inputs sets them apart from other neural networks. They are divided into three sorts of layers:

- Convolutional Layer
- Pooling Layer
- Fully-connected(FC) Layer

A convolutional network's first layer is the convolutional layer. While further convolutional layers or pooling layers can be added after convolutional layers, the fully-connected layer is the last layer. The CNN becomes more complicated with each layer, detecting larger areas of the image. Earlier layers concentrate on basic elements like colours and borders. As the visual data travels through the CNN layers, it begins to distinguish larger elements or features of the item, eventually identifying the target object.

2.2.4.1 Convolutional Layer

The first layer to extract features from an input image is convolution. By learning visual attributes with tiny squares of input data, convolution retains the link between pixels. It's a mathematical process with two inputs: an image matrix and a filter or kernel.

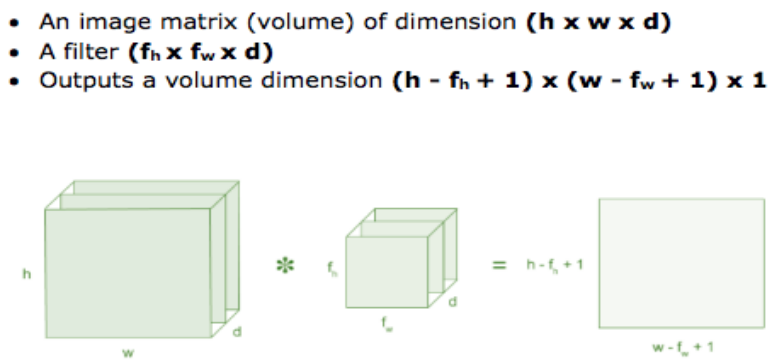


Figure 2.0

Consider a 5 x 5 whose image pixel values are 0/1 and filter matrix 3 x 3 as shown in below

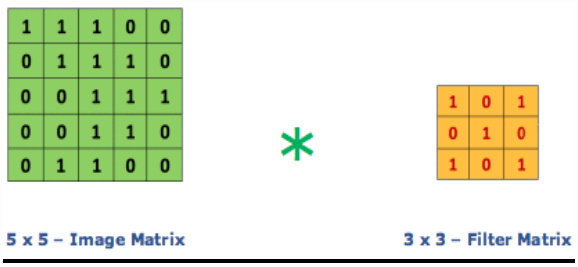


Figure 3.0

Then the convolution of 5 x 5 image matrix multiplied with 3 x 3 filter matrix which is called "Feature Map" as output shown in below

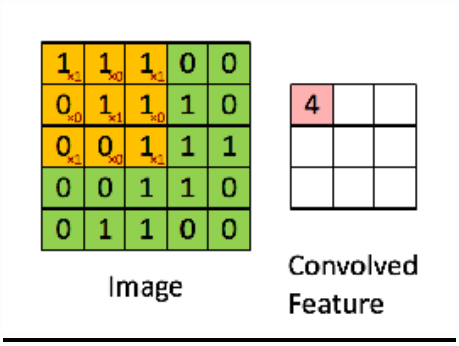


Figure 4.0

- **STRIDES**: The number of pixels shifted across the input matrix is referred to as the stride. When the stride is set to 1, the filters are moved one pixel at a time. We shift the filters two pixels at a time when the stride is two, and so on. Convolution would function with a stride of 2 as seen in the diagram below.

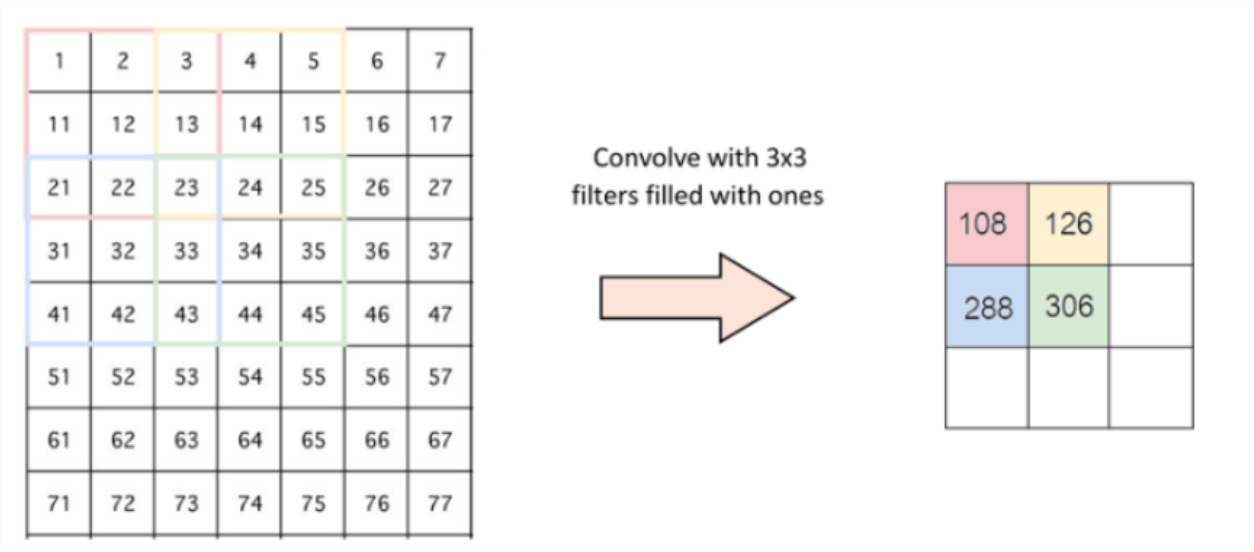


Figure 5.0

- **PADDING:** Filters do not always precisely fit the input image. There are two possibilities:
  - Pad the picture with zeros (zero-padding) so that it fits.
  - Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid parts of the image.

- **NON LINEARITY (ReLU):** ReLU stands for Rectified Linear Unit for a non-linear operation. The output is $f(x) = \max(0,x)$. ReLU's purpose is to introduce non-linearity in our ConvNet. Since, the real world data would want our ConvNet to learn would be non-negative linear values.

2.2.4.2 Pooling Layer

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling is also called subsampling or downsampling which reduces the dimensionality of each map but retains important information. Spatial pooling can be of different types:

- Max Pooling
- Average Pooling
- Sum Pooling

Max pooling takes the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map is called sum pooling.

2.2.4.3 Fully Connected Layer

We flattened our matrix into a vector and passed it into a fully connected layer, similar to a neural network, in the FC layer. In partly linked layers, the pixel values of the input picture are not directly connected to the output layer, as previously stated. Each node in the output layer, on the other hand, links directly to a node in the preceding layer in the fully-connected layer.

This layer performs classification tasks based on the characteristics retrieved by the preceding layers and their various filters. While convolutional and pooling layers often utilise ReLu functions to categorize inputs, FC layers typically use a softmax activation function to provide a probability from 0 to 1.

2.3 MNIST Dataset:

The MNIST (**Modified National Institute of Standards and Technology database**) dataset. This is probably one of the most popular datasets among machine learning and deep learning enthusiasts. The MNIST dataset contains 60,000 training images of handwritten digits from zero to nine and 10,000 images for testing. So, the MNIST dataset has 10 different classes. The handwritten digits images are represented as a 28×28 matrix where each cell contains gray-scale pixel value.

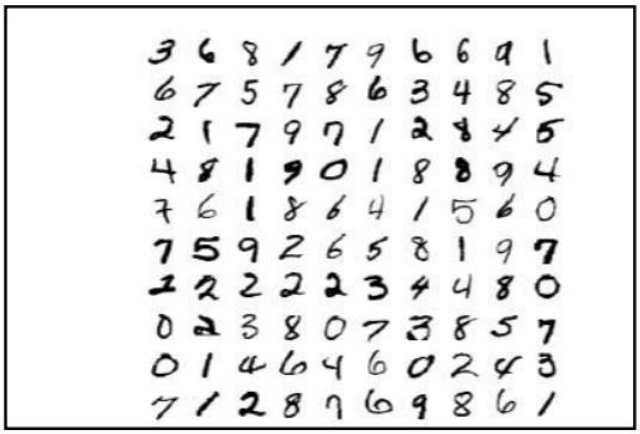


Figure 6.0 :- MNIST database sample

3. REQUIREMENT ANALYSIS

Handwriting has played an immeasurable role in human history. From manuscripts to textbooks , we have always been using handwriting. It is possible that if we do not preserve our handwritten articles, it will get lost in the ages. Hence we need digitalization to store our Information so that it lasts long and our future generations have access to it.

We need to convert written page information into digital information for storing information for a long period of time. Manually, it costs a lot of time and money to perform such an operation. Here comes an idea to build a handwritten digit recognition system with the help of machine learning. This project's main objective is to be able to read the images containing the handwritten digits and be able to identify those digits using basic image correlation techniques. These images are normally represented and read as matrices, in which every element portrays a pixel. This project will be mainly using matrices and heavy numerical computations, that is why it is very important to consider the tools that would provide us with a suitable environment for performing these computations**.**

3.1 SOFTWARE PLATFORM

Python 3.9: Python is broadly utilized universally and is a high-level programming language. It was primarily introduced for prominence on code, and its language structure enables software engineers to express ideas in fewer lines of code. Python is a programming language that gives you a chance to work rapidly and coordinate frameworks more effectively**.**

Tensor Flow: It is an amazing information stream in the machine learning library made by the Brain Team of Google and made open source in 2015. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of tensorflow.

Tkinter: It is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Numpy: NumPy is a Python library used for working with arrays. NumPy is short for "Numerical Python".

Pandas: Pandas is a Python library used to analyze data.

Keras: It is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible is key to doing good research.

3.2 HARDWARE PLATFORM

Web-Camera: A webcam is a video camera that feeds or streams an image or video in real time to or through a computer network, such as the Internet.

Graphic Card: A graphics card is an expansion card for your PC that is responsible for rendering images to the display.A graphics card is a type of display adapter or video card installed within most computing devices to display graphical data with high clarity, color, definition and overall appearance. A graphics card provides high-quality visual display by processing and executing graphical data using advanced graphical techniques, features and functions.

4. DETAILED DESIGN

The purpose of this document is to investigate the proposed system's design possibilities, such as architecture design, block diagram, sequence diagram, data flow diagram, and user interface design, in order to define steps such as pre-processing, feature extraction, segmentation, classification, and digit recognition.

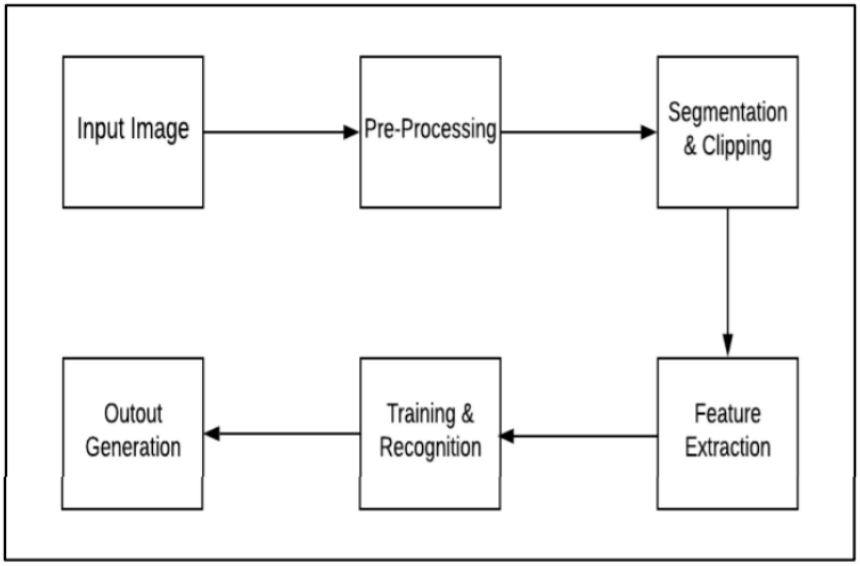


Figure 7.0 :- Architecture of the Proposed System

Figure 1 depicts the proposed system's architecture diagram. The proposed model comprises four phases for classifying and detecting the digits:

1) Pre-processing
2) Segmentation
3) Feature Extraction
4) Classification and Recognition

4.1 Pre-Processing: The pre-processing step's job is to execute a variety of activities on the supplied image. It essentially improves the image by making it segmentation-friendly. The first step in processing the training set photos is to threshold them into a binary image in order to reduce the data.

The image data cannot be fed directly into the model so we need to perform some operations and process the data to make it ready for our neural network. The dimension of the training data is (60000,28,28). The CNN model will require one more dimension so we reshape the matrix to shape (60000,28,28,1).

4.2 Segmentation: After the input photos have been pre-processed, the sequence of images is divided into sub-images of individual digits. Individual digits are assigned a number and pre-processed digit images are divided into a sub-image of individual digits. Each digit gets resized into pixels on its own. The dataset photos are segmented using an edge detection algorithm in this step.

4.3 Feature Extraction: After the pre-processing and segmentation stages are completed, the pre-processed images are represented in the form of a matrix that contains pixels from very large images. In this approach, representing the digits in the photos that carry the necessary information would be beneficial. Feature extraction is the term for this activity. Redundancy in the data is removed during the feature extraction stage.

4.4 Classification and Recognition: The retrieved feature vectors are used as individual inputs to each of the following classifiers in the classification and recognition stage. To demonstrate the working system model, the extracted characteristics are integrated and described using the convolution neural network CNN.

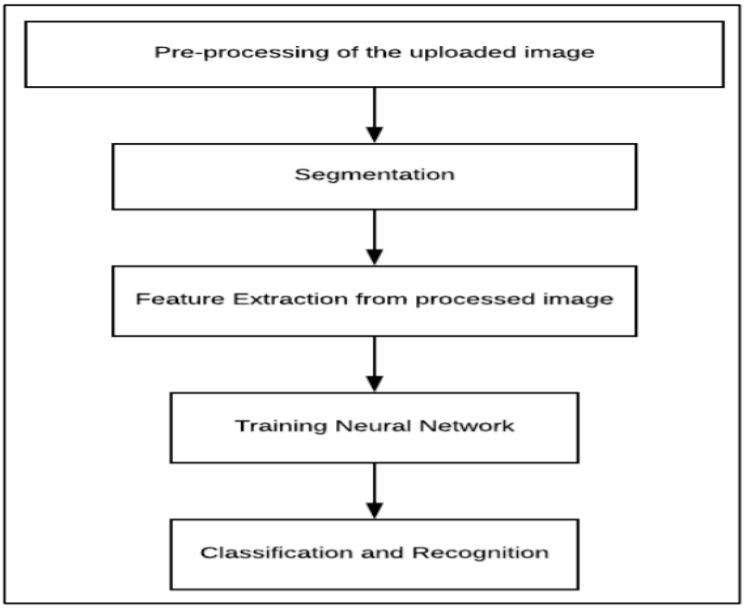


Figure 8.0 :- Block Diagram of proposed model

5. IMPLEMENTATION

Let's say, we want the computer to recognize the handwritten digit "9".The way computer look at this as a grid of numbers, here it is '-1' and '1'(In reality it is 0-255).The issue is that this is too much hard coded and there is variation since it is a handwritten digit, if we have little shift in the digit '9' then it doesn't matches with our original number grid and computer will not able to recognize this digit.

5.1 CNN Working On a Digit

In digit '9', there are these tiny features. Here, we use the concept of filter, in case of digit '9' we have three filters. These little edges come together to form a loopy circle pattern - kind of the head of this digit. In the middle, we have a vertical line. At the bottom, we have a diagonal line.We know that whenever there is a loopy circle pattern at the top, vertical line in the middle and diagonal at the bottom that means this is digit '9'.

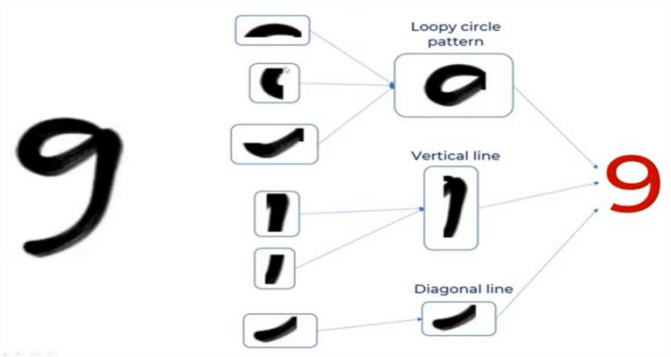


Figure 9.0 Filters

In the below figure, there are three filters:

- Loopy Circle Pattern at the head,
- Vertical line in the middle
- Diagonal line at the bottom

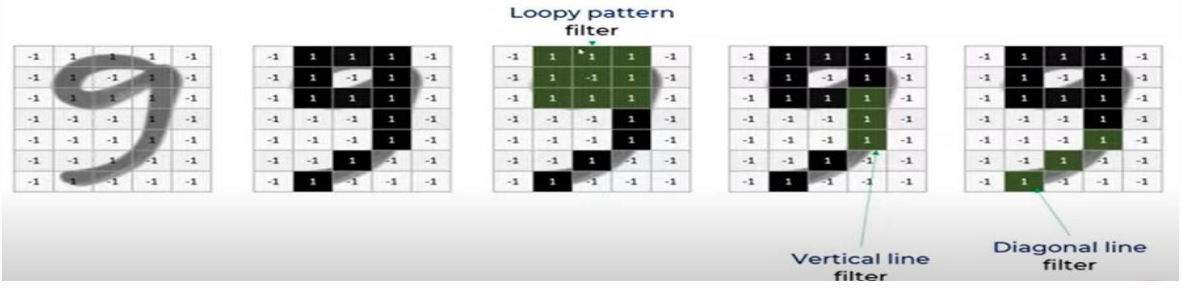


Figure 10.0

5.2 Filter operation for loopy pattern for digit '9'.

The way CNN works is taking a 3X3 grid from the original image and multiplying individual numbers with

this filter so this '-1' multiplied with '1 and so on. In the end, find the average which is divided by 9 because there are a total of 9 numbers and whatever we get the result, put it in the result matrix. This particular thing is a feature map.

Figure 11.0 Filter Operation

After the filter operation on each filter we get three features maps. In the figure below, First dimension represents the vertical line filter, the second represents the diagonal and the third represents the loopy pattern filter.

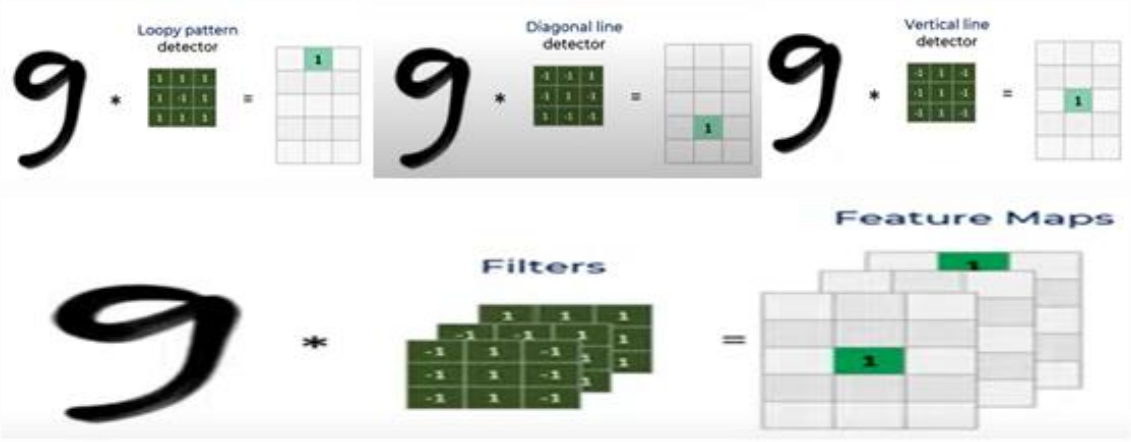


Figure 12.0 Feature Maps

5.3 Making the model Non Linear using ReLU

Relu helps with making the model non-linear. The ReLU function is f(x)=max(0,x).It replaces negative values with zero and does nothing with the positive values. One way ReLUs improve neural networks is by speeding up training.

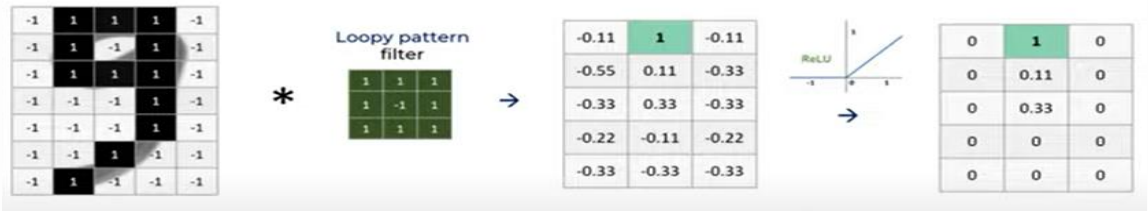


Figure 13.0 ReLU Operation

5.4 Applying Max Pooling

Max pooling is done To reduce the size(dimension) and computation.It returns the maximum value from the portion of the image covered by the filter.

Figure 14.0 Stride of One & Two

5.5 Complete CNN

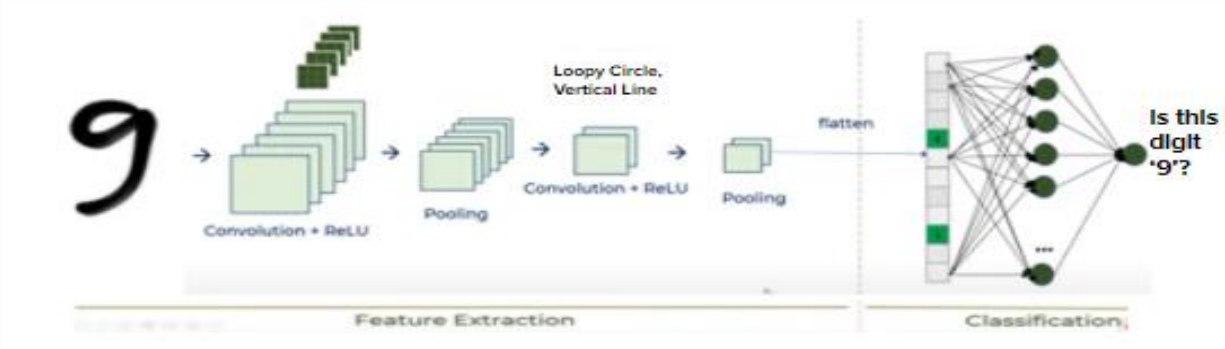


Figure 15.0 Complete CNN Model

5.6 CODE IMPLEMENTATION

5.6.1 Importing the required libraries:

```
#importing several required packages
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
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
from keras import utils as np_utils
from keras import optimizers
```

5.6.2 Loading the MNIST dataset from Keras and Splitting into training and testing dataset:

The Keras library already contains some datasets and MNIST is one of them. So we can easily import the dataset and start working with it. The mnist.load_data() method returns us the training data, its labels and also the testing data and its labels.

```
#Load and splitting the dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
print("Dataset successfully loaded!!!")
print(x_train.shape, y_train.shape)
```

5.6.3 Reshaping the training and testing image dataset and printing number of image data in training and testing respectively:

The image data cannot be fed directly into the model so we need to perform some operations and process the data to make it ready for our neural network. The dimension of the training data is (60000,28,28). The CNN model will require one more dimension so we reshape the matrix to shape (60000,28,28,1).

```
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)

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')
```

```
x_train shape: (60000, 28, 28, 1)
60000  - Train samples
10000  - Test samples
```

Figure 16.0 Training and Test Samples

5.6.4 Initializing batch size, epochs and num_classes:

```
batch_size = 128
```

```
num_classes = 10
epochs = 10
```

5.6.5 Building a CNN Model with accuracy on training and validation dataset:

A CNN model generally consists of convolutional and pooling layers. It works better for data that are represented as grid structures, this is the reason why CNN works well for image classification problems. The dropout layer is used to deactivate some of the neurons and while training, it reduces the overfitting of the model. We will then compile the model with the Adam optimizer.

The model.fit() function of Keras will start the training of the model. It takes the training data, validation data, epochs, and batch size. After training, we save the weights and model definition in the 'mnist.h5' file.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(5, 5),activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation=tf.nn.relu))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation=tf.nn.relu))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation=tf.nn.relu))
model.add(Dropout(0.2))model.add(Dense(64, activation=tf.nn.relu))
model.add(Dropout(0.3))
model.add(Dense(32, activation=tf.nn.relu))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation=tf.nn.softmax))
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
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")
```

```
2021-11-28 15:23:49.804671: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
Epoch 1/10
2021-11-28 15:23:51.696632: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8005
469/469 [==============================] - 10s 6ms/step - loss: 0.7050 - accuracy: 0.7691 - val_loss: 0.1048 - val_accuracy: 0.9739
Epoch 2/10
469/469 [==============================] - 3s 6ms/step - loss: 0.2150 - accuracy: 0.9438 - val_loss: 0.0770 - val_accuracy: 0.9820
Epoch 3/10
469/469 [==============================] - 3s 6ms/step - loss: 0.1527 - accuracy: 0.9605 - val_loss: 0.0560 - val_accuracy: 0.9879
Epoch 4/10
469/469 [==============================] - 3s 5ms/step - loss: 0.1239 - accuracy: 0.9675 - val_loss: 0.0643 - val_accuracy: 0.9859
Epoch 5/10
469/469 [==============================] - 3s 6ms/step - loss: 0.1063 - accuracy: 0.9720 - val_loss: 0.0577 - val_accuracy: 0.9884
Epoch 6/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0974 - accuracy: 0.9752 - val_loss: 0.0598 - val_accuracy: 0.9870
Epoch 7/10
469/469 [==============================] - 3s 5ms/step - loss: 0.0929 - accuracy: 0.9755 - val_loss: 0.0553 - val_accuracy: 0.9895
Epoch 8/10
469/469 [==============================] - 3s 5ms/step - loss: 0.0815 - accuracy: 0.9778 - val_loss: 0.0541 - val_accuracy: 0.9888
Epoch 9/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0719 - accuracy: 0.9798 - val_loss: 0.1008 - val_accuracy: 0.9807
Epoch 10/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0669 - accuracy: 0.9814 - val_loss: 0.0477 - val_accuracy: 0.9914
The model has successfully trained
```

Figure 17.0 CNN Model

5.6.6 Saving the CNN model:

```
model.save('model.h5')
print("Saving the CNN model as model.h5")
```

5.7 USER INTERFACE:

We have created a GUI Sketchpad using Tkinter for user input and also used Web-Cam for the interaction as another option. The GUI Sketchpad allows the user to draw the numbers, which the programme recognizes, while the camera allows the user to display the application a live image, which the application recognizes.

Figure 18.0 GUI Sketchpad

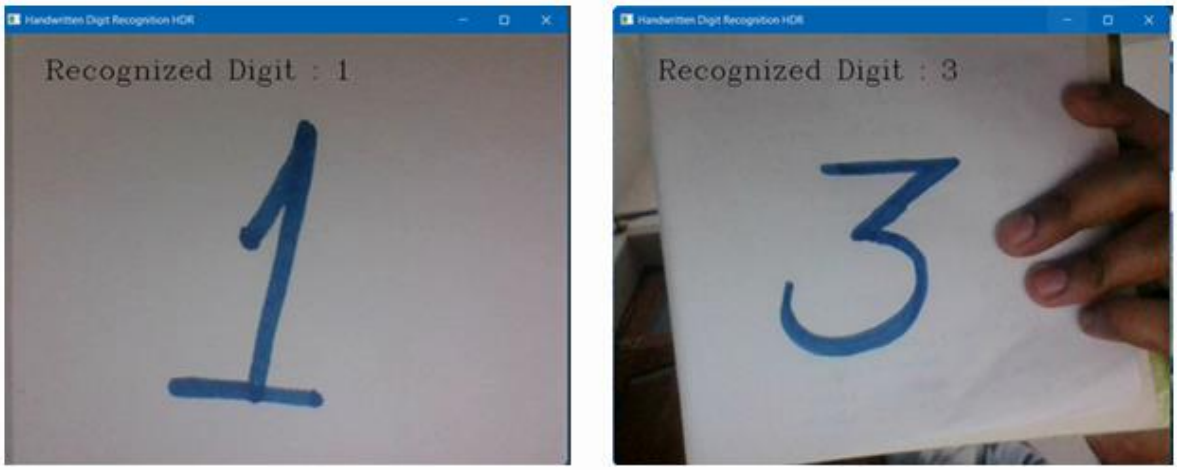


Figure 19.0 Web – Camera Recognition

6. EXPERIMENTAL RESULTS AND ANALYSIS

6.1 Evaluating the model

The training dataset is shuffled before being split, and the sample shuffling is done each time, ensuring that each model we evaluate has the same train and test datasets in each fold, allowing us to compare.

With a default batch size of 128, we'll train the baseline model for a modest 10 training epochs. The test set for each fold will be used to evaluate the model throughout each epoch of the training run as well as at the end of the run to estimate the model's performance. As a result, we'll maintain track of the history generated by each run, as well as the fold's categorization accuracy.

These behaviours are implemented in the evaluate model() function (score = model.evaluate(testX, testY, verbose=0) which takes the training dataset as an argument and returns a collection of accuracy scores and training histories.

The loss and any metrics defined while constructing the model are computed via model.evaluate(). In our scenario, the accuracy is calculated using the network weights provided by the saved model and 10,000 testing samples.

Training set size affects the accuracy and accuracy increases as the number of data increases. The more data in the training set, the smaller the impact of training error and test error, and ultimately the accuracy can be improved.

Accuracy can change depending on how training and testing data is split, and this can be improved even more if more training and testing data is provided. If the size of the data grows larger, there is always the possibility of improving accuracy.

Evaluating the model:

```
score = model.evaluate(x_test, y_test)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0477 - accuracy: 0.9914
Test loss: 0.04769239202141762
Test accuracy: 0.9914000034332275
```

Figure 20.0 Evaluation of Model

6.2 Comparison

ALGORITHM	ACCURACY
Random Forest	80%
KNN	90%
SVM	96.2%
CNN	99.2%

Table 1.0 Comparison

For handwritten digit recognition, we discovered that CNN provided the best accurate results. As a result, we may infer that CNN is the best choice for any form of prediction task involving picture data.

Following the model's construction, we compiled it using the Adam optimizer and a specific cross-entropy loss function, both of which are customary in the creation of a convolution neural network.

We can train the model using training data for 10 iterations after it has been successfully formed, but as the number of iterations increases, there is a risk of overfitting.

As we are using real-world data for prediction, we limit the training to 98 percent accuracy. Test data was utilised to evaluate the model.

7. CONCLUSION AND FUTURE SCOPE

7.1 Future Scope

It can later be expanded to include character recognition and real-time handwriting analysis. The recognition of handwritten digits is the initial step in the large field of Artificial Intelligence and Computer Vision. As may be observed from the outcomes of the experiment, CNN outperforms other classifiers. With more convolution layers and buried neurons, the findings can be made more accurate. It has the potential to fully eliminate the necessity for typing.

These algorithms can be used in hospitals for detailed medical diagnosis, treatment, and monitoring of patients, in surveillance systems to keep track of suspicious activity within the system, in fingerprint and retinal scanners, database filtering applications, equipment checking for national forces, and many other major and minor problems. By employing these algorithms in day-to-day applications, we can contribute to creating an environment of safety, awareness, and comfort.

7.2 Conclusion

In this project, to produce predictions of handwritten numbers from 0 to 9, a common dataset called MNIST was used. The data was cleaned, scaled, and shaped before being used. A CNN model was developed with TensorFlow and then trained on the training dataset. Finally, the trained model was used to make predictions.

Digit recognition is an excellent model issue for learning about neural networks and provides a solid foundation for developing more complex deep learning approaches.

A great level of accuracy can be achieved with these deep learning approaches. In comparison to other research approaches, this strategy focuses on which classifier performs best by enhancing classification model accuracy by above 99 percent. A CNN model with Keras as the backend and Tensorflow as the software can achieve 98.72 percent accuracy.

The only challenging part is the noise present in the real-world image, which needs to be looked after. The learning rate of the model is much dependent on the number of dense neurons and the cross-validation measure.

8. REFERENCES

Journal Article:

[1] A. Dutta and A. Dutta, Handwritten digit recognition using deep learning, International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), vol. 6, no. 7, Dec 2021.

[2] H. Jain and N. Sharma, "Handwritten digit recognition using CNN," *Lecture Notes in Networks and Systems*, pp. 631–637, Jan. 2021.

[3] M. A. Hossain and M. M. Ali, "Recognition of handwritten digit using convolutional neural network (CNN)," *Global Journal of Computer Science and Technology*, vol. 19, no. 2, pp. 27–33, 2019.

[4] *Irjet.net*.[Online].Available:https: //www.irjet.net/archives/V4/i7/IRJET-V4I7599.pdf. [Accessed: 01 - Dec -2021].

Online:

[5] "CNN for deep learning: Convolutional Neural Networks," *Analytics Vidhya*, 23-Jul-2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/. [Accessed: 01-Dec-2021].

Blog:

[6] J. Brownlee, "Handwritten Digit Recognition using Convolutional Neural Networks in Python with Keras," *Machine Learning Mastery*, 27-Jun-2016.