HANDWRITTEN DIGIT RECOGNITION USING PYTHON

IN THE FULFILMENT OF THE PROJECT by

(BATCH 27)

- *P.VENKATA NAGESWARA RAO* *622237*

Under the guidance of

DR.V.PRAKASH SINGH

DEPARTMENT OF

ELECTRONIC AND COMMUNICATION ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY ANDHRA PRADESH,

TADEPALLIGUDEM-534102, INDIA

APRIL – 2024

ABSTRACT

The main objective is to represent a neural network-based approach for handwritten digit classification and we also test this project by giving some handwritten digit images as input. Handwritten digit recognition is a fundamental problem in the field of machine learning and computer vision with widespread applications. The model uses Convolution Neural Network (CNN) architecture, which is effective for simpler image classification tasks like recognizing handwritten digits, Here we'll trace characters of digits. The project involves collecting a dataset of handwritten digits, preprocessing the images, designing and training a neural network, validating the model's performance, evaluating its accuracy, and deploying the trained model for practical use. The outcomes of this project are, firstly, we are going to learn how we can train our MNIST dataset, and whatever the training model we have made we can implement that in real-life scenarios.

1. INTRODUCTION

In the realm of digital image processing and machine learning, the recognition of handwritten digits stands as a benchmark challenge that has garnered significant attention. Our project aims to address this challenge by employing a Convolutional Neural Network (CNN), a class of deep neural networks renowned for their prowess in analyzing visual imagery. Utilizing a Convolutional Neural Network (CNN), our project aims to accurately classify handwritten digits from the MNIST dataset.

In addition to processing this pre-defined dataset, our model extends its capabilities to recognize digits drawn in Microsoft Paint. This feature simulates a real-world scenario where the model interprets and classifies user-generated content, thereby demonstrating its adaptability and potential for practical applications. Through rigorous training and optimization, our CNN model endeavors to achieve high accuracy in digit recognition, paving the way for further research and development

2. PROBLEM STATEMENT

Handwritten digit recognition is a pivotal task in pattern recognition and machine learning. Its objective is to accurately identify and classify handwritten digits (0-9) from input images. This project aims to develop a convolutional neural network (CNN) model capable of achieving high accuracy in recognizing handwritten digits.

The scope of this project encompasses the recognition of digits from grayscale images using a dataset consisting of handwritten digit images, such as the widely used MNIST dataset. The CNN model will be trained to classify each digit image into one of the ten classes (0-9).

Key deliverables include a trained CNN model, comprehensive documentation detailing the model architecture, training process, hyperparameters, and evaluation results, as well as an evaluation of metrics such as accuracy, precision, recall, and F1-score. Additionally, a user-friendly interface will be developed to allow users to input handwritten digit images and obtain predicted digit outputs.

Challenges in this project include handling variations in handwriting styles and quality, optimizing the CNN architecture and hyperparameters to achieve high accuracy, dealing with overfitting and underfitting during model training, and developing efficient preprocessing techniques to enhance the quality of input images.

Success criteria for this project include achieving a classification accuracy above 95% on the test dataset and consistently accurate predictions on a variety of handwritten digit images.

3. METHODOLOGY

1. DATA COLLECTION

The MNIST dataset is a widely used benchmark dataset for handwritten digit recognition tasks. The dataset contains 60,000 training images and 10,000 test images, making it a standard dataset for training and evaluating machine learning models, especially neural networks. Due to its simplicity and accessibility, the MNIST dataset has become a benchmark for evaluating the performance of new machine learning algorithms and neural network architectures.

2. DATA PROCESSING

Preprocess the images to standardize them, including resizing, normalization, and possibly augmentation techniques to increase the dataset size.

- Normalize pixel values to the range [0, 1].
- Split the dataset into training, validation, and test sets.

3. MODEL ARCHITECTURE DESIGN

Design a CNN architecture suitable for image classification. Typically, it consists of convolutional layers, pooling layers, and fully connected layers. Experiment with different architectures to find the most effective one.

Common layers include: Convolutional layers (with filters/kernels)

Pooling layers (e.g., max-pooling)

Fully connected (dense) layers

Dropout layers (to prevent overfitting).

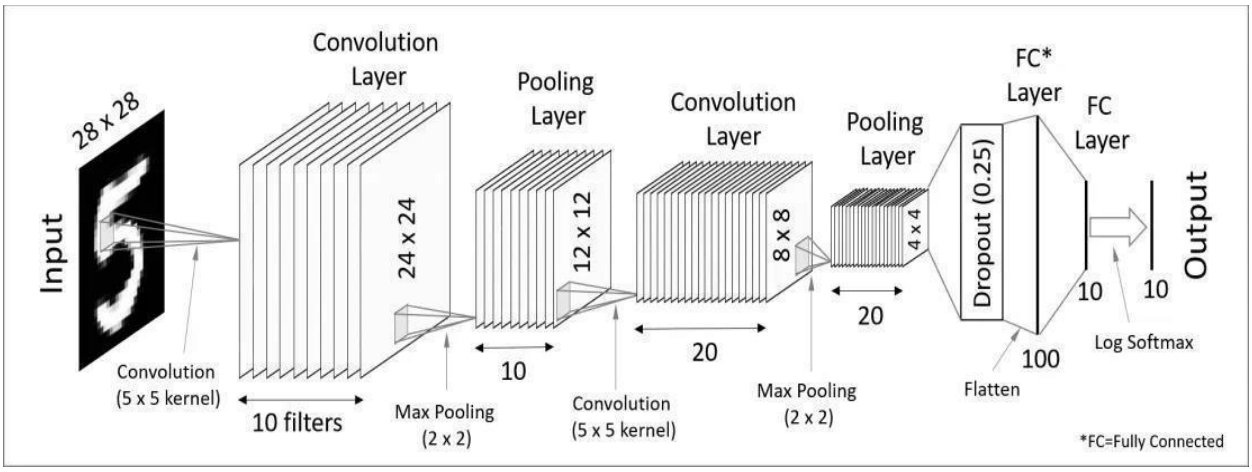


Figure1: CNN Architecture in MNIST Dataset

4. TRAINING

Train the CNN on your dataset using techniques like Adam optimization. Adjust hyperparameters such as learning rate, batch size, and number of epochs to optimize performance**.**

5. VADILATION

Evaluate the model on the validation set. Calculate metrics (e.g., accuracy, precision, recall, F1-score).

6. FINE-TUNING AND OPTIMIZATION

Hyperparameter Tuning: Fine-tune hyperparameters (e.g., learning rate, dropout rate) using techniques like grid search or random search.

Regularization Techniques: Apply L2 regularization to prevent overfitting. Experiment with different dropout rates.

Learning Rate Scheduling: Implement learning rate schedules (e.g., reduce-on-plateau) to adaptively adjust learning rates during training.

4. PLATFORM

In the context of a project on handwritten digit recognition using convolutional neural networks (CNNs), the "platform" refers to the software environment or framework utilized to develop and train the neural network model. Common platforms for such deep learning projects include TensorFlow, and Keras. These platforms offer a range of tools and functionalities tailored for building, training, and deploying neural network models. The choice of platform often depends on factors such as familiarity with the framework, community support, and specific features required for the project.

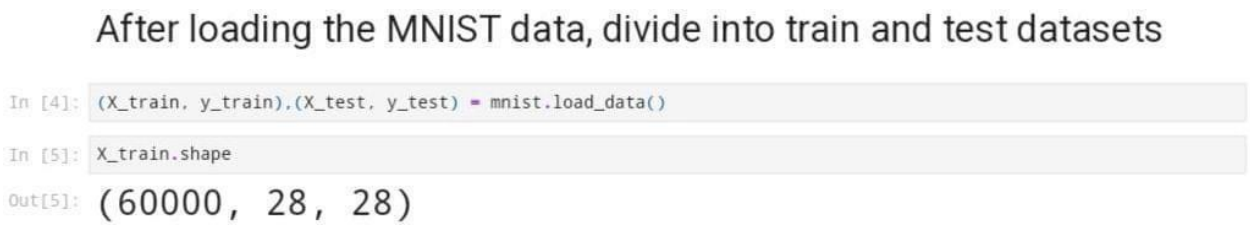
5. CONFIGURATION OF PLATFORM

It pertains to the specific setup and parameters employed within the chosen platform. This encompasses various aspects, including the architecture of the CNN model, hyperparameters, and optimization techniques utilized during training. For instance, the configuration may involve specifying the number of layers in the CNN, the type of convolutional and pooling layers used, activation functions, learning rate, batch size, and regularization techniques such as dropout or weight decay. Additionally, the platform configuration may include the choice of loss function and optimization algorithm, such as categorical cross-entropy loss and Adam optimizer, respectively. Fine-tuning these configurations is crucial for achieving optimal performance and accuracy in the handwritten digit recognition task. Overall, the platform configuration plays a vital role in shaping the effectiveness and efficiency of the CNN model in recognizing handwritten digits.

6. IMPLEMENTATION

1. First, Import Tensor flow as “tf” and load the MNIST data set.

2. Training and Testing the data set

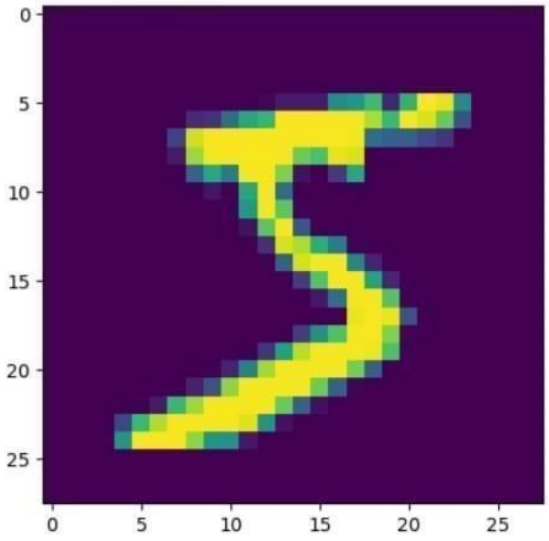


3. check the image of the number is in binary or not, if the image is in color than process it into binary image

4. Normalizing the data in gray level (0 to 255) to increase the accuracy to recognize the digit. It is data-reprocessing step.

5. Resizing the image to make suitable for applying the convolution operation.

In [6]:
```
## just check the graph,how data looks like
import matplotlib.pyplot as plt
plt.imshow(X_train[0])
plt.show()## in order to execute the graph
## however we don't know whether its colour image or binary images
## so inorder to plot it change the configuration
plt.imshow(X_train[0], cmap = plt.cm.binary)
```



Out[6]: <matplotlib.image.AxesImage at 0x2004d4218d0>

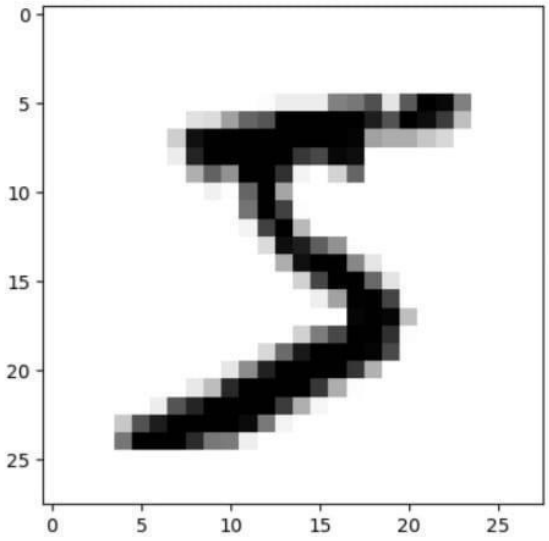


Figure2: converting color image to Binary

```
In [8]: X_train = tf.keras.utils.normalize(X_train, axis = 1)
        X_test = tf.keras.utils.normalize(X_test, axis=1)
        plt.imshow(X_train[0], cmap= plt.cm.binary)
```

```
Out[8]: <matplotlib.image.AxesImage at 0x2004d4903d0>
```

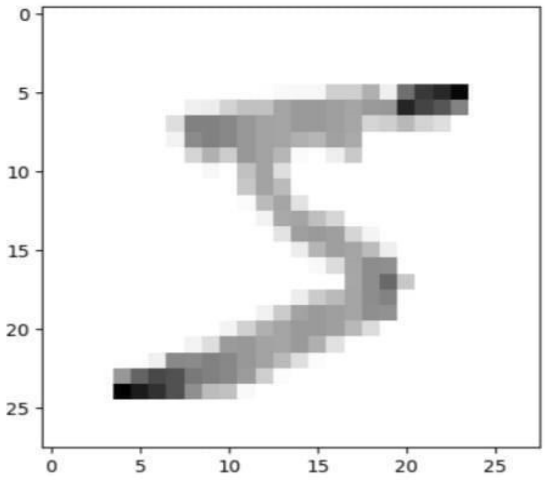


Figure3: Normalization of image

```
In [11]: import numpy as np
         IMG_SIZE = 28
         X_train = np.array(X_train).reshape(-1, IMG_SIZE, IMG_SIZE, 1)## increasing one dimension for kernel operation
         X_test =  np.array(X_test).reshape(-1, IMG_SIZE, IMG_SIZE, 1)##increasing one dimension for kernel operation
         print("training samples dimension",X_train.shape)
         print("testing samples dimension",X_test.shape)
```

```
training samples dimension (60000, 28, 28, 1)
testing samples dimension (10000, 28, 28, 1)
```

Figure4: Resizing of the image

6. Creating the Convolution Neural Network and Evaluating on testing data set to find the accuracy

In [12]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout, Activation, Flatten, Conv2D, MaxPooling2D
```

In [13]:
```
## creating a neural network
model = Sequential()

##First convolution layer (60,000,28,28,1)
model.add(Conv2D(64, (3,3), input_shape = X_train.shape[1:]))##only for first convolution layer to mention inp
model.add(Activation("relu"))## activation function to make non-linear
model.add(MaxPooling2D(pool_size=(2,2)))##Maxpooling single maximum value of 2x2

##2nd convolution layer
model.add(Conv2D(64, (3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

## 3rd 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())## before using fully connected layer, need to be flatten so that 2D to 1D
model.add(Dense(64))
model.add(Activation("relu"))

##Fully connected layer # 2
model.add(Dense(32))
model.add(Activation("relu"))

##Last Fully connected layer, output must be equal to number of classes, 10(0-9)
model.add(Dense(10))## this last dense layer must be equal to 10
model.add(Activation('softmax'))
```

Figure5: Creating Convolution neural network

In [18]:
```
##Evaluating on testing data set MNIST
test_loss, test_acc = model.evaluate(X_test, y_test)
print("test loss on 10,000 test samples", test_loss)
print("validation accuracy on 10,000 test samples", test_acc)
```

```
313/313 [==============================] - 5s 16ms/ste
p - loss: 0.0556 - accuracy: 0.9835
test loss on 10,000 test samples 0.05556328594684601
validation accuracy on 10,000 test samples 0.983500003
8146973
```

Figure6: Evaluating the accuracy

7. Now we can check the confusion matrix for the predictions

```
In [30]: cm = tf.math.confusion_matrix(labels=y_test,predictions=y_predicted_labels)
         cm

Out[30]: <tf.Tensor: shape=(10, 10), dtype=int32, numpy=
         array([[ 966,    1,    1,    0,    8,    0,    1,    1,    1,
         1,    1],
                [   0, 1130,    1,    1,    0,    1,    0,    0,    2,
         0,    0],
                [   1,    1, 1010,    1,    2,    0,    2,   14,    1,
         0,    0],
                [   0,    2,    5,  992,    0,    6,    0,    4,    1,
         0,    0],
                [   0,    0,    1,    0,  978,    0,    1,    0,    0,
         2,    0],
                [   2,    0,    0,    6,    0,  880,    2,    1,    1,
         0,    0],
                [   2,    3,    1,    0,    4,    3,  944,    0,    1,
         0,    0],
                [   0,    5,   10,    1,    0,    0,    0, 1009,    2,
         1,    0],
                [   2,    1,    2,    1,    6,    2,    0,    1,  956,
         3,    0],
                [   2,    2,    0,    0,   13,    7,    0,    5,    2,
         978]])>

In [31]: import seaborn as sn
         plt.figure(figsize = (10,7))
         sn.heatmap(cm,annot=True,fmt='d')
         plt.xlabel('predicted')
         plt.ylabel('truth')

Out[31]: Text(95.72222222222221, 0.5, 'truth')
```

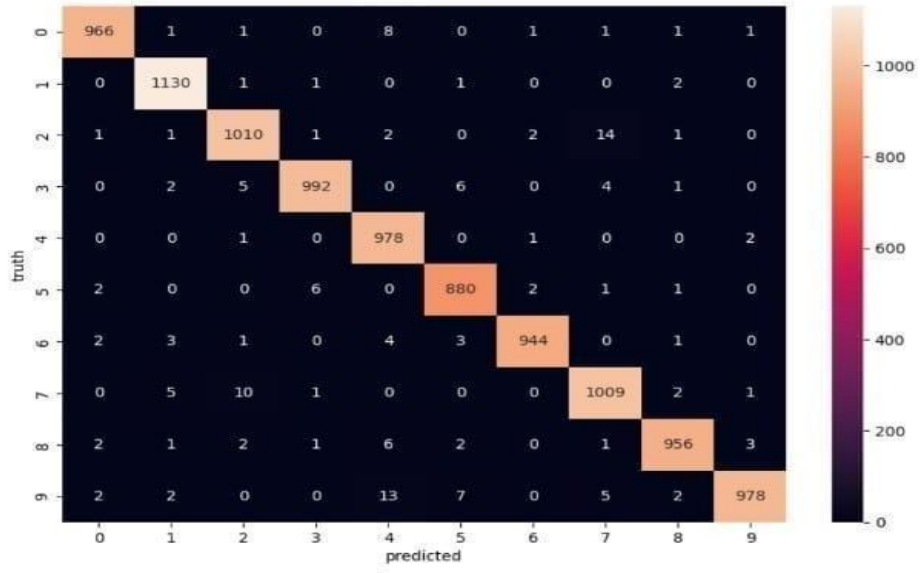


Figure7: confusion matrix for predictions

7. Giving a image of a number as input by defining its location and the output can be obtain by following the above process.

INPUT:

```
In [25]: import numpy
         import cv2

In [26]: img = cv2.imread("C:\\Users\\shray\\Pictures\\Screenshots\\three.png")

In [27]: cv2.imshow('Image',img)

In [28]: img.shape
Out[28]: (643, 365, 3)

In [29]: gray= cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

In [30]: gray.shape
Out[30]: (643, 365)

In [31]: resized = cv2.resize(gray, (28,28), interpolation = cv2.INTER_AREA)

In [32]: resized.shape
Out[32]: (28, 28)

In [33]: newimg = tf.keras.utils.normalize(resized, axis=1) ## 0 to 1 scaling

In [34]: newimg = np.array(newimg).reshape(-1, IMG_SIZE, IMG_SIZE,1) ##kernel operation of convolution layer,

In [35]: newimg.shape
Out[35]: (1, 28, 28, 1)

In [36]: predictions = model.predict(newimg)
         1/1 [==============================] - 0s 204ms/step

In [37]: print(np.argmax(predictions))
         3
```

Figure8: process of recognition the image of number “3”

7. RESULT

After training the convolutional neural network (CNN) on the MNIST dataset for handwritten digit recognition, the model achieved a test accuracy of approximately 98%. This means that the model is able to correctly classify handwritten digits with a high level of accuracy.

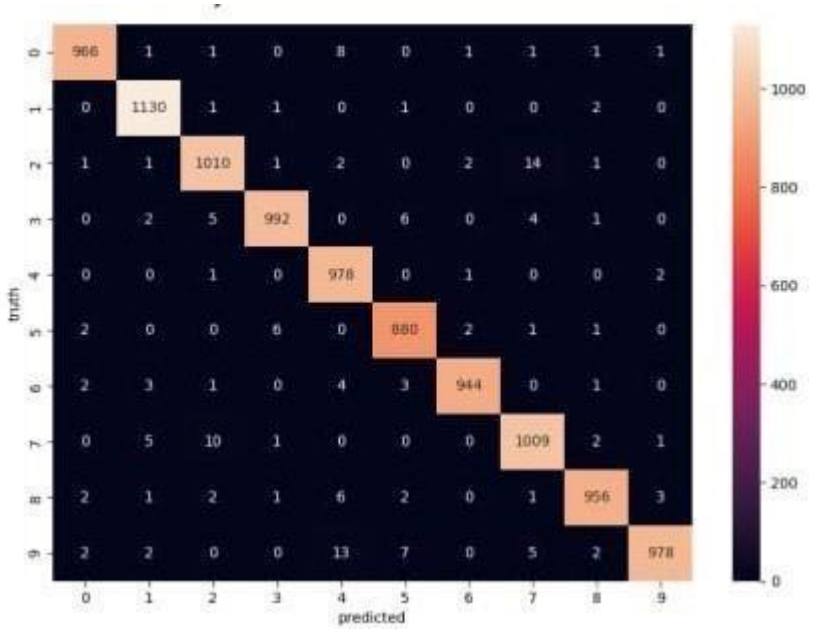


fig9: confusion matrix for predictions

8. CONCLUSION

The project successfully demonstrated the effectiveness of using a CNN for handwritten digit recognition. By leveraging deep learning techniques and the TensorFlow library, we were able to build and train a model that can accurately classify handwritten digits with high accuracy. This model could be further improved by fine-tuning hyperparameters, experimenting with different architectures, or using more advanced techniques such as data augmentation. Overall, the project showcases the power of deep learning in solving image classification tasks and highlights the potential applications of CNNs in various real-world scenarios, such as digit recognition in postal services, bank checks processing, and more.

9. REFERENCES

1. **https://youtu.be/lZPMDvovvl0?si=pLL93qols1Afvsy8**
2. **https://www.geeksforgeeks.org/handwritten-digit-recognition-using-neural-network/amp/**
3. **https://ieeexplore.ieee.org/document/9823806**