GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

HANDWRITTEN DIGIT RECOGNITION USING CNN

A Report for the Evaluation 3 of Project 2

Submitted by

AKRITI SINGH

(1613101087/16SCSE101363)

in partial fulfillment for the award of the degree

of

Bachelor of Technology

IN

Computer Science and Engineering

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

Under the Supervision of

MS.SUPRIYA KHAITAN

Assistant Professor

APRIL / MAY- 2020

TABLE OF CONTENTS

Abstract

Handwritten digit recognition is the working of a machine to train itself for recognizing the digits from various sources like emails, bank cheques, etc, in real-world scenarios. Our input consists of numerous images of digits, which are fed in the model, where they are preprocessed and converted in an array. That array is passed as input to our Working Model, i.e. Convolution Model. After repeated convolution and pooling, the convolution network predicts the output, based on factors like density and shape of the area under consideration.

In addition to that, we have added some of our own data that we have preprocessed and added to the data of MNIST. We have added those test cases that we found were missing in the given data set, and adding them will make sure that no kind of image remains uncovered by our model. It will help in improving the accuracy of our model, which it did as expected. We achieved an accuracy of 99.28%, and our loss percent is approximately 0.2. It makes our model stand out in terms of increased efficiency. Handwritten digit recognition is an important problem in today's world scenario because there are millions and millions of people across the globe with millions of different handwriting styles that could be a trouble for recognition by a human being because there are digits that are quite confusing .That is why I thought of working on this project.

Introduction

What is a Handwritten Digit Recognition System?

In this the machine trains itself to recognize human written digits from various sources like emails, bank cheques, etc. in real world scenarios .The inputs are then taken and fed in a model where they are processed and converted in an array. The main problem lies in developing an efficient algorithm for recognizing handwritten digits which are submitted by the users .There are more than millions of people in the world and each individual has their own way of writing digits which can either be understood or can even make you confuse, in order to reduce the complexity of understanding digits this system can be of help to people .It will not only save time but also increase the efficiency of the work that is to performed. There are various techniques for implementing this system like the machine learning algorithms-support vector machine, naïve bayes, bayes net etc. And various neural network approach like simple neural network, KNN, CNN, ANN etc. Here we will be focusing on the Convolutional neural network approach. The MNIST problem is a dataset developed by Yann LeCun, Corinna Cortes and Christopher Burges for evaluating machine learning models on the handwritten digit classification problem that I will be using to train my datasets.

The dataset was constructed from a number of scanned document dataset available from the National Institute of Standards and Technology (NIST). This is where the name for the dataset comes from, as the Modified NIST or MNIST dataset.

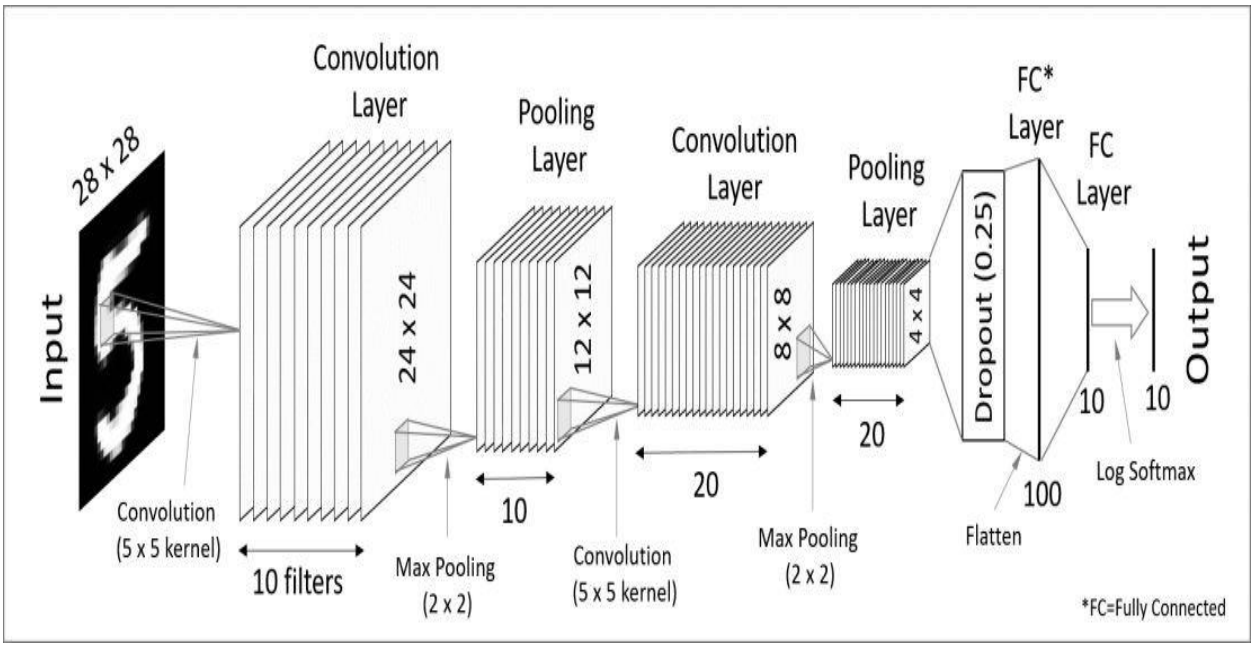
Images of digits were taken from a variety of scanned documents, normalized in size and centered. This makes it an excellent dataset for evaluating models, allowing the developer to focus on the machine learning with very little data cleaning or preparation required.

Each image is a 28 by 28 pixel square (784 pixels total). A standard split of the dataset is used to evaluate and compare models, where 60,000 images are used to train a model and a separate set of 10,000 images are used to test it.

It is a digit recognition task. As such there are 10 digits (0 to 9) or 10 classes to predict. Results are reported using prediction error, which is nothing more than the inverted classification accuracy.

Excellent results achieve a prediction error of less than 1%. State-of-the-art prediction error of approximately 0.2% can be achieved with large Convolutional Neural Networks.

EXISTING MODEL



1. First, an image is taken as an input.
2. Then that image is preprocessed and converted into an array and that array is passed as input to the convolution model.
3. we get the regions of interest to classify the objects in the image.
4. All these regions are then reshaped as per the input of the CNN, and each region is passed to the ConvNet.
5. CNN then extracts features for each region and uses a fully connected layer that uses softmax as an activation function to interpret the output as probabilities.
6. Finally, the maximum of the probability is classified as the output using np.argmax ().

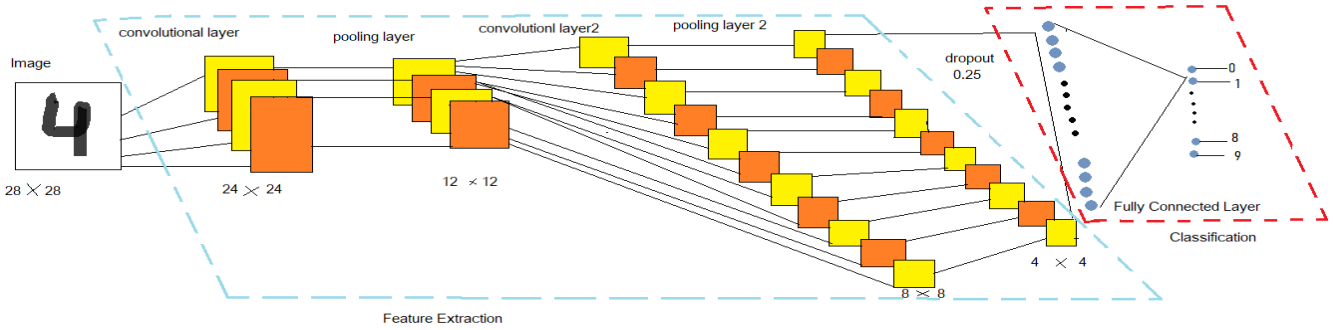
To recognize the handwritten digits, a seven-layered convolutional neural network with one input layer followed by five hidden layers and one output layer is designed. The input layer consists of 28 by 28 pixel images which mean that the network contains 784 neurons as input data. The input pixels are grayscale with a value 0 for a white pixel and 1 for a black pixel. Here, this model of CNN has five hidden layers. The first hidden layer is the convolution layer 1 which is responsible for feature extraction from an input data. This layer performs convolution operation tosmall localized areas by convolving a filter with the previous layer. In addition, it consists of multiple feature maps with learnable kernels and rectified linear units (ReLU). The kernel size determines the locality of the filters.

ReLU is used as an activation function at the end of each convolution layer as well as a fully connected layer to enhance the performance of the model. The next hidden layer is the pooling layer 1. It reduces the output information from the convolution layer and reduces the number of parameters and computational complexity of the model. The different types of pooling are max pooling, min pooling, average pooling, and L2 pooling. Here, max pooling is used to subsample the dimension of each feature map.

Convolution layer 2 and pooling layer 2 which has the same function as convolution layer 1 and pooling layer 1 and operates in the same way except for their feature maps and kernel size varies. A Flatten layer is used after the pooling layer which converts the 2D featured map matrix to a 1D feature vector and allows the output to get handled by the fully connected layers. A fully connected layer is another hidden layer also known as the dense layer. It is similar to the hidden layer of Artificial Neural Networks (ANNs) but here it is fully connected and connects every neuron from the previous layer to the next layer. In order to reduce overfitting, dropout regularization method is used at fully connected layer 1.

Proposed System

CNN is a neural network that consists of connected layers of neuron. A neuron consist of a number known as activation and the connections are known as weights which help the network to determine which neuron is to be activated in which layer and what should be the activation popularly known as the feed forward process. This process continues until the output neurons get activated. The whole CNN process can be divided into two parts the first one is the feature extraction part and the second one is the classification part. The feature extraction is done by combining the two layers; the Convolutional layer and the pooling layer whereas the classification part is performed by the dense layers the image prediction is handled by the input layer and the output layer focuses on the different classes that we are trying to predict.



1. **Convolution layer**- this layer basically combines a filter with the prior layer, and the number of filter combined can range from one to many because when the number of the filter increases in the layer the features of the image become more distinctive and you know what the best part is we don't even have to select which filter is to be used as that work is done by the neural network itself as they even learn the features themselves while training the data. And we select the kernel size which helps in determining the locality of the filter and choose the stride value which only determines how many pixels we need to advance while combining our filter to the layer.

2. **Pooling layer**-this layer is applied to down-sample the inputs for reducing the computational complexity of the model and for avoiding the over fitting problem, the most common technique used is max pooling.

3. **Dense layer**-this layer randomly exclude the number of neurons in the present layer in order to reduce the over fitting problem and then later converts the 2d matrix data to a vector called flatten.

Implementation Details

SOFTWARE REQUIREMENT SPECIFICATIONS :

Python Libraries Required:

- Matplotlib
- NumPy
- CV2
- TensorFlow v1.12 (TensorFlow-GPU)
- Keras
- tqdm

API Required:

- Image Recognition API

Dependencies for TensorFlow-GPU:

- CUDA Toolkit v9.0
- Nvidia graphic drivers associated with CUDA Toolkit v9.0
- CuDNN v7.0 associated with CUDA Toolkit v9.0

HANDWRITTEN DIGIT RECOGNITION DATA USED

the dataset used are the MNIST datasets where each image is a 28×28 pixel grey scale square image. There are 60,000 training datasets and 10,000 testing data sets. for example-

These are a few examples of the MNIST data sets used for the execution.

As seen the accuracy and the loss calculated of the handwritten digit recognition model using CNN is approx 99.28 and 0.02 simultaneously.

TRAINING DATASET

First we download the datasets from Keras by using from keras.datasets import mnist, and then we use the sequential model for the model building which is prebuilt in Keras and then the dense layers are imported which will be later used for predicting the labels and then dropout layer is added that will reduce the over fitting problem. After that the flatten layer is added that converts the 3d array to 1d.After doing all this we have finally imported all the layers the Convolutional, the pooling layer and the numpy.

Some class variables are assigned as we have 10 classes and also because of which the batch size becomes 128.After that we load the data and reshape it using the .reshape(60000,28,28,1) function where 60000 is the number of images data is being trained on and all of this is done because CNN accepts only 4-d vector. After all of this is done we prepare our test data to test upon the trained data.

SOURCE CODE-

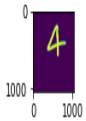
```
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from tqdm import tqdm
import pickle

test_data = []
path = "/home/akriti/Desktop/attachments3"
IMG_SIZE = 28

def create_test():
    for img in tqdm(os.listdir(path)):
        try:
            img_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)  # convert to array
            new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize to normalize data size
            test_data.append([new_array])  # add this to our training_data
        except Exception as e:pass


def create_test1():
    for x in range(10):
        path1 = '/content/attachements3/' + str(x) + '.png'
        plt.figure(figsize=(10,10))
        try:

            img_array = cv2.imread(path1, cv2.IMREAD_GRAYSCALE)  # convert to array
```

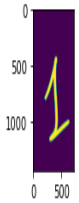

TESTING DATASET CODE

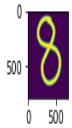
```
            img_array = cv2.imread(path1, cv2.IMREAD_GRAYSCALE)  # convert to array
            plt.subplot(10 / 10, 10, x + 1)
            plt.imshow(img_array)
            new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize to normalize data size
            test_data.append([new_array])  # add this to our training_data

        except Exception as e:pass


create_test1()

X = np.array(test_data).reshape(-1, IMG_SIZE, IMG_SIZE, 1)

pickle_out = open("Xtest.pickle","wb")
pickle.dump(X, pickle_out)
pickle_out.close()
```

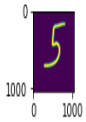
RESULT

After training the data the result is-









On testing the test data we get the correct output-

```
(10, 28, 28, 1)

4
1
8
9
5
```

THE ACCURACY RESULT-

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
 - 118s - loss: 0.1695 - accuracy: 0.9464 - val_loss: 0.0381 - val_accuracy: 0.9879
Epoch 2/10
 - 137s - loss: 0.0435 - accuracy: 0.9865 - val_loss: 0.0275 - val_accuracy: 0.9914
Epoch 3/10
 - 117s - loss: 0.0301 - accuracy: 0.9903 - val_loss: 0.0244 - val_accuracy: 0.9921
Epoch 4/10
 - 119s - loss: 0.0232 - accuracy: 0.9929 - val_loss: 0.0271 - val_accuracy: 0.9911
Epoch 5/10
 - 125s - loss: 0.0177 - accuracy: 0.9943 - val_loss: 0.0268 - val_accuracy: 0.9923
Epoch 6/10
 - 134s - loss: 0.0161 - accuracy: 0.9948 - val_loss: 0.0233 - val_accuracy: 0.9932
Epoch 7/10
 - 145s - loss: 0.0128 - accuracy: 0.9958 - val_loss: 0.0262 - val_accuracy: 0.9929
Epoch 8/10
 - 130s - loss: 0.0113 - accuracy: 0.9965 - val_loss: 0.0260 - val_accuracy: 0.9923
Epoch 9/10
 - 126s - loss: 0.0095 - accuracy: 0.9970 - val_loss: 0.0257 - val_accuracy: 0.9932
Epoch 10/10
 - 129s - loss: 0.0095 - accuracy: 0.9970 - val_loss: 0.0253 - val_accuracy: 0.9928


712/10000 [=====================>.......] - ETA: 1s
808/10000 [=====================>.......] - ETA: 1s
904/10000 [=====================>.......] - ETA: 1s
000/10000 [=====================>......] - ETA: 1s
096/10000 [=====================>......] - ETA: 1s
192/10000 [=====================>......] - ETA: 1s
288/10000 [=====================>......] - ETA: 1s
384/10000 [======================>.....] - ETA: 1s
480/10000 [======================>.....] - ETA: 0s
544/10000 [======================>.....] - ETA: 0s
640/10000 [======================>.....] - ETA: 0s
736/10000 [======================>.....] - ETA: 0s
832/10000 [======================>.....] - ETA: 0s
928/10000 [======================>.....] - ETA: 0s
024/10000 [======================>....] - ETA: 0s
120/10000 [======================>....] - ETA: 0s
248/10000 [======================>....] - ETA: 0s
344/10000 [======================>....] - ETA: 0s
440/10000 [======================>....] - ETA: 0s
536/10000 [======================>....] - ETA: 0s
632/10000 [======================>....] - ETA: 0s
728/10000 [======================>....] - ETA: 0s
824/10000 [======================>....] - ETA: 0s
920/10000 [======================>....] - ETA: 0s
000/10000 [==============================] - 6s 626us/step
st loss: 0.025287489897097476
st accuracy: 0.9927999973297119
```

Conclusion

Using CNN we not only get the correct output but also the accuracy is more as compared to others and even the loss is approximately 0.02 %.If the same procedure is done without using CNN then the error rate obtained is 2.68% which is more hence CNN helps in giving the optimum result then other methods with less error rate. And this is why because while using CNN we get more distinct image features for training our datasets using the Convolutional layer than we can obtain through any other method. That is why CNN should be preferred over other methods for the digit recognition

FUTURE ENHANCEMENTS

- As different people in the world have different kinds of writing style so this system will be very beneficial for identifying the handwritten digits even when it cannot be recognized by a human.
- This system can be used in banks for identifying the digits written on the cheques.
- It can be used for postal card code reading.
- License plate reading.

References

1. Wan Zhu, Classification of MNIST Handwritten Digit Database using Neural Network.
2. Norhidayu binti Abdul Hamid, Nilam Nur Binti Amir SjarifHandwritten Recognition Using SVM, KNN and Neural Network, Advance Informatics School Universiti Teknologi Malaysia.
3. LeCun et al., "Backpropagation applied to handwritten zip code recognition," Neural computation”
4. https://www.ijitee.org/wp-content/uploads/papers/v8i6/F3888048619.pdf
5. http://ijarcet.org/wp-content/uploads/IJARCET-VOL-6-ISSUE-7-990-997.pdf
6. https://www.youtube.com/watch?v=j-3vuBynnOE&list=PLQVvvaa0QuDfhTox0AjmQ6tvTgMBZBEXN&index=2