GALGOTIAS UNIVERSITY

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

CNN: IMPLEMENTATION OF HANDWRITTEN DIGIT RECOGNITION SYSTEM

A Report for the Evaluation 3 of Project 2

Submitted by

AKSHIT GAMBHIR

(1613101096/16SCSE101510)

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

DR. ARVIND KUMAR

Associate 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 trouble for recognition by a human being. After all, some digits 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 into 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 way of writing digits which can either be understood or can even make you confuse, 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 perform. 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 a 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 several scanned document datasets 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.

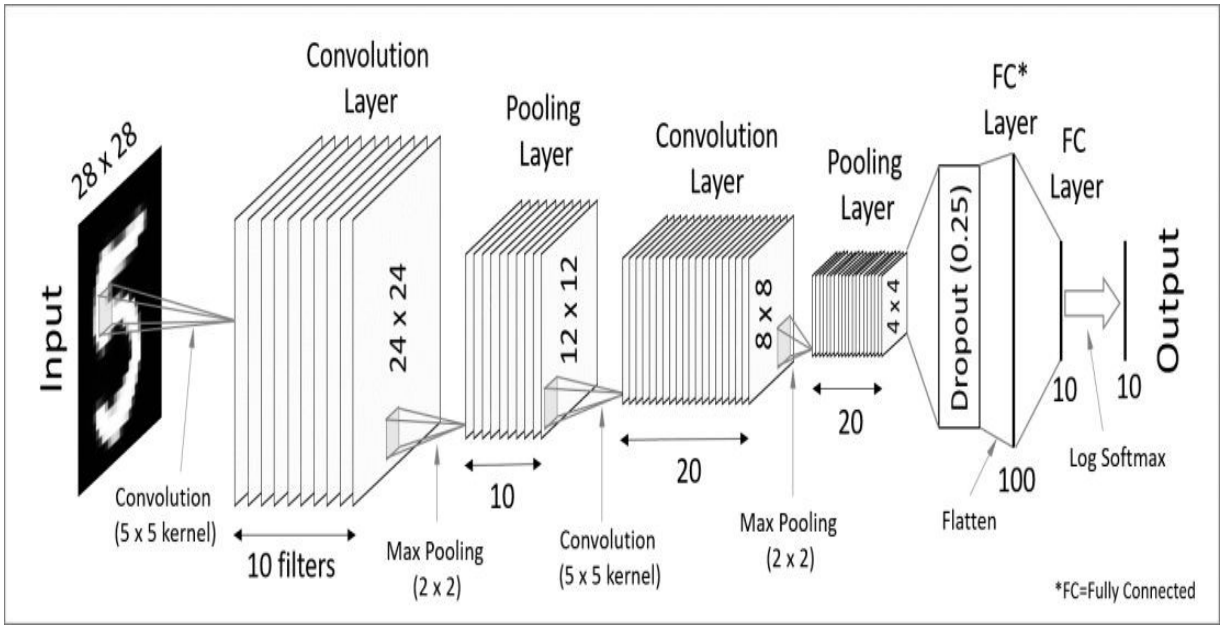
Images of digits were taken from a variety of scanned documents, normalized in size and centered. This makes it an excellent dataset for evaluating models, allowing the developer to focus on machine learning with very little data cleaning or preparation required.

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 means 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 input data. This layer performs convolution operation to small localized areas by convolving a filter with the previous layer. Also, 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 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. To reduce overfitting, the dropout regularization method is used at fully connected layer 1.

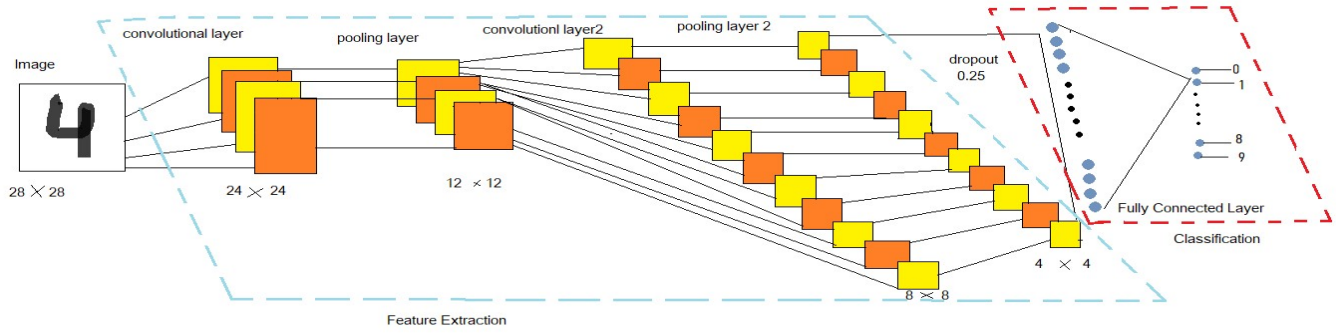
Issues with Existing Model

The performance of the previous models can be increased by tuning the parameters and the proposed model works on the same and excels over the previous models [1-3] in metrics like accuracy and loss by hyperparameter tuning, the accuracy has now been significantly increased in comparison to previous model [2] and now the accuracy comes to about 99.3 % and loss is calculated as 0.0235 which is also low.

Proposed System

CNN is a neural network that consists of Convolution layers, max-pooling layers, and fully connected layers of a neuron. Each neuron consists of some input weights known as connections, biases, and applied activation functions and these connections help the network to determine which neuron is to be givens more preference and in which layer. In general, a neural net involves two processes the feed-forward process that involves feeding the inputs to the model and the backpropagation process that involves backpropagation of the loss but the CNN can be divided into two parts the first one being the feature extraction and the second one is the classification part. The feature extraction is done by combining the two layers the Convolutional layer and the max or min pooling layer whereas the classification part is performed by the dense layers i.e. the fully connected layers followed by the softmax activation at the output layer.

.



1. Convolution layer – For a pixel, this layer is responsible for taking the weighted average of the neighboring pixels, and hence determining and learning new features, unlike the fully connected layers it only depends on the neighboring pixels and not all of them. This layer can have one or more filters the more the number of filters the more diverse our image feature can become, major advantages of the convolution layer over fully connected layers is the significant reduction in the number of parameters due to weight sharing i.e. a common kernel for all the neurons. Apart from that, it learns the areas of interest without human intervention.

2. Pooling layer - this layer is applied to down-sample the inputs for reducing the computational complexity of the model and for avoiding the overfitting problem, the common techniques used are max pooling, average max pooling, min pooling.

3. Dense layer – Also Known as the Fully Connected Layer, after the features have been extracted those features are flattened and fed to the Dense Layer and hence involves the maximum number of parameters to be trained.

Implementation Details

SOFTWARE REQUIREMENT SPECIFICATIONS :

Python Libraries Required:

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

API Required:

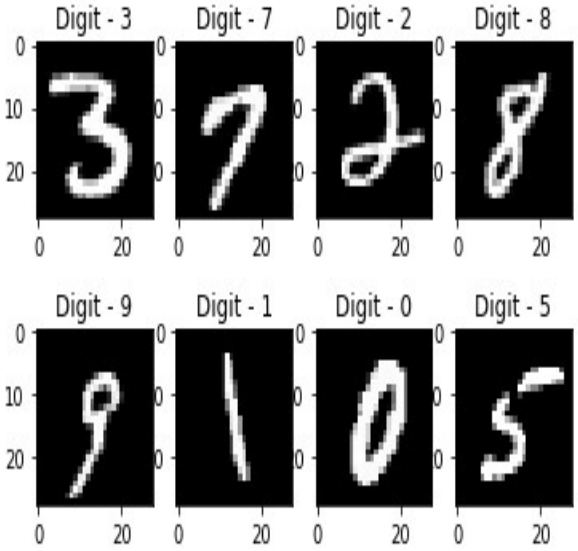
- Image Recognition API

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

DATASET USED

The proposed system uses the MNIST data set [7]. It has 70,000 images that can be used to train and evaluate the system. The train set has 60,000 images and the test set has 10,000 images [9]. It is the subset of the NIST dataset (National institute of standards and technology), having 28 x 28 size input images and 10 class labels from (0-9).



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

All the digits are normalized and also centered. Each image in the referenced data set is represented as an array of 28x28. The array has 784 pixels whose values range from 0 to 255. If the pixel value is '0' it indicates that the background is black and if it is '1' the background is white.

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

SOURCE CODE FOR MODEL IMPLEMENTATION-

Importing Libraries

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Activation
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
from keras.layers.normalization import BatchNormalization
import matplotlib.pyplot as plt
import numpy as np
import os
import cv2
from tqdm import tqdm
import pickle
```

Connecting TensorFlow to the GPU

```
%tensorflow_version 2.x
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

Load Train and Test data from MNIST

```
# input image dimensions
img_rows, img_cols = 28, 28

# the data, shuffled and split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols
, 1)
x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1
)
input_shape = (img_rows, img_cols, 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')
```

Model building and Training

```
batch_size = 64
num_classes = 10
epochs = 15

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()

model.add(Conv2D(10, (5, 5), input_shape=(28,28,1)))
model.add(Activation('relu'))


"""BatchNormalization(axis=-1)
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))"""

model.add(MaxPooling2D(pool_size=(2,2)))
BatchNormalization(axis=-1)

model.add(Conv2D(20,(5, 5)))
model.add(Activation('relu'))

"""BatchNormalization(axis=-1)

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))"""
```

```
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Dropout(0.25))
model.add(Flatten())
# Fully connected layer
BatchNormalization()

model.add(Dense(100))
model.add(Activation('relu'))
BatchNormalization()


model.add(Dense(10))
model.add(Activation('softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=2,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=1)

print('Test loss:', score[0])
print('Test accuracy:', score[1])
model.save('recog_te1')
```

Creating Custom Test Data and using pickle to save it

```
test_data = []
path = "/content/attachements3"
IMG_SIZE = 28

def create_test():
    for img in tqdm(os.listdir(path)):
      plt.figure(figsize=(10,10))
      try:
          img_array = cv2.imread(os.path.join(path, img), cv2.
IMREAD_GRAYSCALE)  # convert to array

          plt.imshow(img_array)
          new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZ
E))  # resize to normalize data size
          test_data.append([new_array])  # add this to our tra
ining_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
            plt.subplot(10/10, 10, x + 1)
```

```
            plt.imshow(img_array)
            new_array = cv2.resize(img_array, (IMG_SIZE, IMG_S
IZE))  # resize to normalize data size
            test_data.append([new_array])  # add this to our t
raining_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()
```

Testing the Model

```
pickle_in = open("Xtest.pickle","rb")
X = pickle.load(pickle_in)
print(X.shape)
X = X/255.0

new_model = tf.keras.models.load_model('recog_te1')
predictions = new_model.predict(X)

for x in range(len(predictions)):
    print(np.argmax(predictions[x]))
```

Model Training

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/15
 - 6s - loss: 0.2416 - accuracy: 0.9247 - val_loss: 0.0601 - val_accuracy: 0.9803
Epoch 2/15
 - 6s - loss: 0.0772 - accuracy: 0.9765 - val_loss: 0.0400 - val_accuracy: 0.9866
Epoch 3/15
 - 6s - loss: 0.0599 - accuracy: 0.9809 - val_loss: 0.0386 - val_accuracy: 0.9872
Epoch 4/15
 - 6s - loss: 0.0492 - accuracy: 0.9849 - val_loss: 0.0272 - val_accuracy: 0.9900
Epoch 5/15
 - 6s - loss: 0.0428 - accuracy: 0.9862 - val_loss: 0.0305 - val_accuracy: 0.9902
Epoch 6/15
 - 6s - loss: 0.0376 - accuracy: 0.9883 - val_loss: 0.0247 - val_accuracy: 0.9922
Epoch 7/15
 - 6s - loss: 0.0360 - accuracy: 0.9886 - val_loss: 0.0226 - val_accuracy: 0.9925
Epoch 8/15
 - 6s - loss: 0.0325 - accuracy: 0.9900 - val_loss: 0.0236 - val_accuracy: 0.9926
Epoch 9/15
 - 6s - loss: 0.0291 - accuracy: 0.9906 - val_loss: 0.0250 - val_accuracy: 0.9916
Epoch 10/15
 - 6s - loss: 0.0269 - accuracy: 0.9914 - val_loss: 0.0241 - val_accuracy: 0.9927
Epoch 11/15
 - 7s - loss: 0.0243 - accuracy: 0.9923 - val_loss: 0.0244 - val_accuracy: 0.9914
Epoch 12/15
 - 7s - loss: 0.0238 - accuracy: 0.9923 - val_loss: 0.0271 - val_accuracy: 0.9917
Epoch 13/15
 - 6s - loss: 0.0214 - accuracy: 0.9931 - val_loss: 0.0220 - val_accuracy: 0.9929
Epoch 14/15
 - 6s - loss: 0.0211 - accuracy: 0.9931 - val_loss: 0.0263 - val_accuracy: 0.9918
Epoch 15/15
 - 6s - loss: 0.0195 - accuracy: 0.9937 - val_loss: 0.0235 - val_accuracy: 0.9922
10000/10000 [==============================] - 1s 96us/step
Test loss: 0.023492475468767225
Test accuracy: 0.9922000169754028
```

Model Summary

The Model involves two Convolution Layers with two max-pooling layers and two fully connected layers,

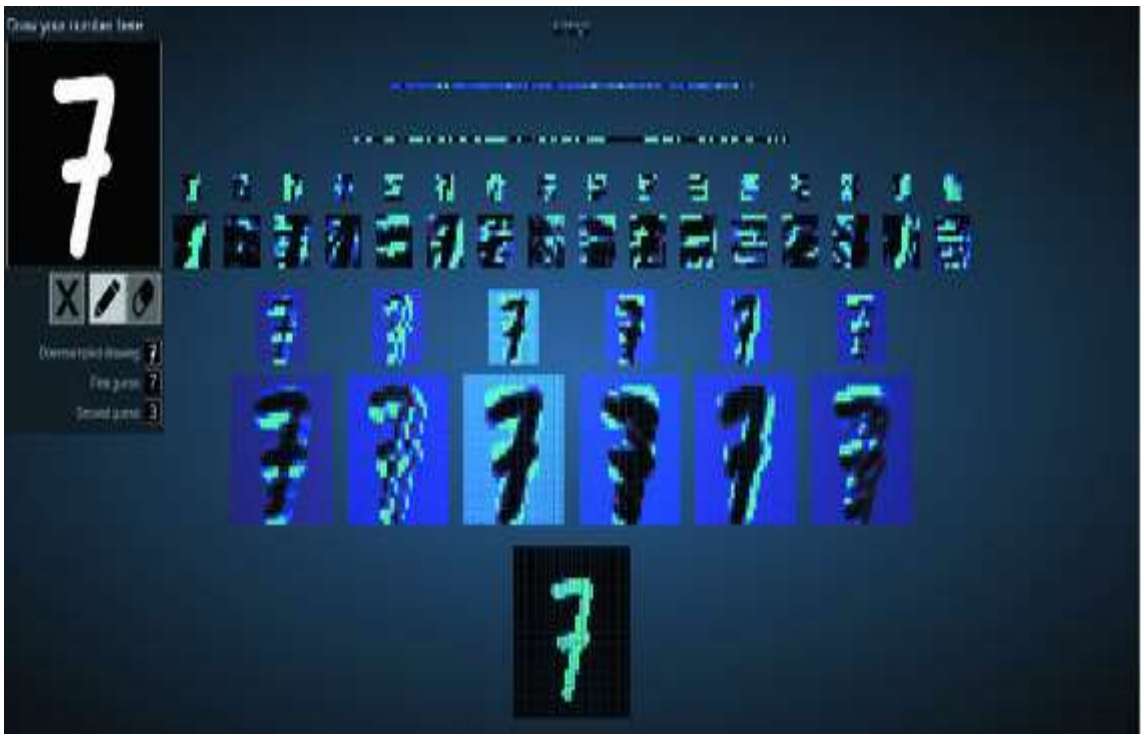
The total number of parameters to be trained is 38390 and the time is taken to train the model has been significantly reduced and is approximately 91 seconds for the 15 epochs by utilizing the GPU of the machine.

The Layered Structure of the Model is presented below:-

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
activation_1 (Activation)    (None, 26, 26, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 24, 24, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 10, 10, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 64)          36928
_________________________________________________________________
activation_4 (Activation)    (None, 8, 8, 64)          0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
activation_5 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_6 (Activation)    (None, 10)                0
=================================================================
```

Visualization of Image transformation representations undergoing a Convolutional model

The below image tries to visualize what happens to an image as it undergoes the convolution model and how some of the image features are extracted by each filter : -



Result

After Testing the model the accuracy comes out to be very high approximately 99.28 % and the loss is minimized to 0.0245 which is also considerably low. Also, the Training time of the model is reduced drastically by making use of the GPU of the device

For the custom Inputs the result is displayed by the image:-

INPUT

OUTPUT

```
4
1
8
9
5
6
7
3
2
8
```

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. Hand Written Digit Recognition using Convolutional Neural Network (CNN) Nimisha Jain, Kumar Rahul, Ipshita Khamaru, Anish Kumar Jha, Anupam Ghosh IJIACS ISSN 2347 – 8616 Volume 6, Issue 5 May 2017.
2. A Comprehensive Data Analysis on Handwritten Digit Recognition using Machine Learning Approach Meer Zohra, D.Rajeswara RaoSeewald, A. K. (2011). On the brittleness of handwritten digit recognition models. ISRN Machine Vision, 2012.
3. Plamondon, R., & Srihari, S. N. Online and off-line handwriting recognition: a comprehensive survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(1), 63-84.
4. Yang, Xuan Jiang. “MDig: Multi-digit Recognition using Convolutional Neural Network on Mobile.” (2015).
5. F. Ertam and G. Aydın, "Data classification with deep learning using Tensorflow," 2017 International Conference on Computer Science and Engineering (UBMK), Antalya, 2017, pp. 755-758.
6. Li Deng, "The MNIST Database of Handwritten Digits images for Machine Learning Research", MIT Press, November 2012.
7. Visualizing and Understanding Convolution Networks 2013 by Matthew D Zeiler, Rob Fergus.

8. Image Preprocessing Loading in your own data - Deep Learning basics with Python, Sentdex YouTube This video helped in the preprocessing of the image which was taken as input from the user.
   Source:
   https://www.youtube.com/watch?v=j-3vuBynnOE&list=PLQVvvaa0QuDfhTox0AjmQ6tvTgMBZBEXN&index=2
9. The MNIST DATABASE by Yann, LeCun, Corinna Cortes, Christopher J.C. Burges http://yann.lecun.com/exdb/mnist