Digit Recognition using CNN

Ashish, G V V Sharma*

Contents

***Abstract*—This manual provides a brief description on how to implement a Convolutional Neural Network from scratch and use it to recognise a hand written digit.**

1. Objective

Our objective is to implement and train a CNN on the standard MNIST database and use those trained parameters to recognise our own hand written digits.

2. Training Dataset

The MNIST database used for training this CNN is available at the following link. Download all the four files in the folder where you want to train the CNN

http://yann.lecun.com/exdb/mnist/

*The authors are with the Department of Electrical Engineering, Indian Institute of Technology, Hyderabad 502285 India e-mail: gadepall@iith.ac.in.

3. About CNN

A CNN is a combination of convolutional filters and fully connected Multi Layer Perceptron (MLP) with backpropagation implemented to update the parameters during the training phase.

There are four main operations in the ConvNet: 1)Convolution 2)Non Linearity (ReLU) 3)Pooling or Sub Sampling 4)Fully connected MLP for Classification

A. Architecture Used

The detailed CNN architecture is used classification of handwritten digits is presented in Fig.1. The first block Image represents training image from MNIST database. All the images in the MNIST database are of the dimension 28*28 pixels.

In brief, the architecture is:

INPUT - CONV1 - RELU - CONV2 - RELU-MAXPOOL - FC1 - OUT

B. Hyperparameters Initialisation

CNN is initialised with the following hyperparameters:

NumOutput=10, LearningRate=0.01, ImgWidth=28, ImgDepth=1, FilterSize=5, NumFilt1=8, NumFilt2=8, BatchSize=20, NumEpochs=2, MU=0.95

Filter size represents the dimension of convolutional filters. Here, the dimension of convolutional filters in both the layers is 5*5. The number of filters in each layer is 8. Epochs represent the number of times the network is allowed to see an image in the training set. The hyperparameter Mu represents momentum in the momentum gradient descent algorithm implemented to update the parameters.

C. Parameters Initialisation

The parameters of the CNN are: Filter weights of Convolutional Layer 1, Layer 2 and their

corresponding biases, Weights of edges in fully connected layer and the biases of each node in output layer. All the biases are initialised to zero (for both convolutional filters and fully connected layer). The filter weights are initialised using lecun normal distribution. The weights of edges in fully connected layer are initialised using uniform distribution.

1) Lecun Normal Distribution: It draws samples from a truncated normal distribution centered on 0 with stddev=sqrt(1 /fanIn) where fanIn is the number of input units in the weight tensor.

D. Back Propagation

Back Propagation algorithm is implemented to update each of the parameters above during the training phase after passing each batch of images. To know more about Back Propagation Algorithm, refer the following tutorial:

https://pdfs.semanticscholar.org/5d79/11c93ddcb34cac088d99bd0cae9124e5dcd1.pdf

4. Training

The CNN initialised above is trained on the available MNIST database. This database also has a separate set for testing the trained network. We can use this set to compute the accuracy of the trained network. The code for training the database can be obtained by cloning the all the files in following github link in the folder where MNIST database is stored.

https://github.com/EE15BTECH11025/cnn_implementation

The network can be trained and accuracy can be computed by running the train.py file. The number of images from the database to be used for training (upto 50000) and testing (upto 10000) can be changed by changing the corrosponding variables in the code and accuracy check can be performed. Note that generally the performance of the network increases as we increase the number of training images as the network becomes more 'trained' to take an accurate decision. The parameters of the trained network are stored in pickle file denoted by PickleFile variable in the code.

5. Testing

Our task now is to recognise our own hand written digits using the parameters of the trained network. For this, we require the images of our own hand written digits to be given as input. One important note to make here is that the input image given to the network should be of size 28*28 pixels as the parameters trained corrospond to MNIST database in which each image is of size 28*28 pixels.

The pickle file trained.pickle contains the parameters when the network is trained on all the available images in the MNIST Database. We can use this pickle file or the file in which we stored parameters after training the network by changing the variable PickleFile in test.py

The test.py file can be used for testing the network. As an example, I have stored resized (28*28 pixels) images of my handwritten digits in 28pix folder and giving them as input to test the acurracy of network (using trained.pickle as pickle file). The prediction for each image and their corrosponding probabilities are presented in Fig.2.

6. Figures

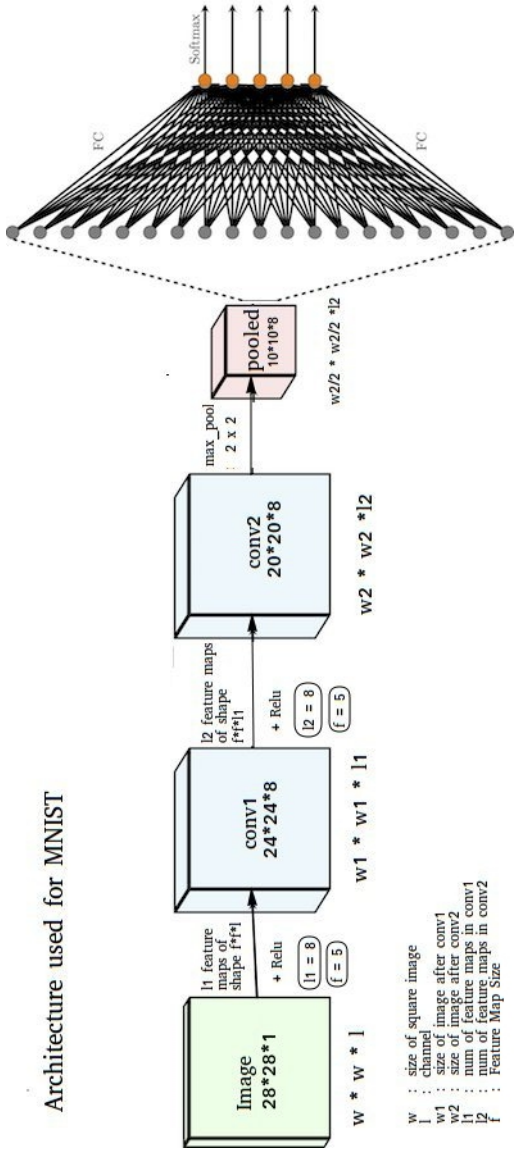

Fig. 1

```
Correct Digit: 0 Predicted Digit: 0 Probability: 0.948116004802174
Correct Digit: 1 Predicted Digit: 1 Probability: 0.9907775138896765
Correct Digit: 2 Predicted Digit: 2 Probability: 0.9948587809924908
Correct Digit: 3 Predicted Digit: 3 Probability: 0.9972565587135374
Correct Digit: 4 Predicted Digit: 4 Probability: 0.9469926119940879
Correct Digit: 5 Predicted Digit: 5 Probability: 0.9982415225544128
Correct Digit: 6 Predicted Digit: 6 Probability: 0.9383764224036766
Correct Digit: 7 Predicted Digit: 7 Probability: 0.999889207524486
Correct Digit: 8 Predicted Digit: 8 Probability: 0.9972504329499281
Correct Digit: 9 Predicted Digit: 9 Probability: 0.978998082580528
```

Fig. 2