# Program for Constructing Convolution Neural Network (CNN) for Handwritten Digit Recognition & Consequent Piano Rendition

*By* Siddhartha Gupta, Sittal Aryal & Trey Smith

**Abstract:** Convolution neural networks (CNNs) as feed forward artificial neural networks can be applied for pattern recognition from images in an efficient manner. In this report we describe the construction of a CNN to use the MNIST handwritten digits database. We first train a CNN using the training and testing data from the MNIST database of handwritten digits as referred to in the problem description. The architecture of our CNN is described in this report along with the training and testing accuracies as well as the training time. Finally, we use the trained network to recognize hand written digits on a piece of paper. In one version of our application the user can choose to upload a picture of hand written characters for detection of digits and in another version the user can use their webcam or smartphone camera to generate a live image capture session where they can show handwritten digits on sheet of paper to the camera. Our program correctly identifies the digits using the trained CNN and then plays the keys of a piano starting from the middle C for each corresponding digit from 0 to 9. The digits are read left to right followed by top to bottom. The user has the option to add more digits to the sheet and continue playing. If an alternate sheet with a different combination of digits is shown a different tune is played for the new combination of notes.

**CNN architecture:**

Relu

Conv and batch normalization

28×28×32

Input Layer

Greyscale

Image:

28x28x1

Max pool

14×14×32

Preprocessing & resizing input images

Relu

Relu

Conv and batch normalization

14×14×32

Conv and batch normalization

7×7×64

Conv and batch normalization

7×7×64

Max pool

7×7×32

Softmax

Relu

Fully connected Layer

10×1

9

8

5

6

2

1

3

4

7

0

Max pool

3x3x64

Input Layer

7×32

64

Output Classification with size 10x1

**CNN layer summary:**

1 'imageinput' Image Input 28x28x1 images with 'zerocenter' normalization

2 'conv\_1' Convolution 32 3x3x1 convolutions with stride [1 1] and padding [1 1 1 1]

3 'batchnorm\_1' Batch Normalization Batch normalization with 32 channels

4 'relu\_1' ReLU ReLU

5 'maxpool\_1' Max Pooling 2x2 max pooling with stride [2 2] and padding [0 0 0 0]

6 'conv\_2' Convolution 32 3x3x32 convolutions with stride [1 1] and padding [1 1 1 1]

7 'batchnorm\_2' Batch Normalization Batch normalization with 32 channels

8 'relu\_2' ReLU ReLU

9 'maxpool\_2' Max Pooling 2x2 max pooling with stride [2 2] and padding [0 0 0 0]

10 'conv\_3' Convolution 64 3x3x32 convolutions with stride [1 1] and padding [1 1 1 1]

11 'batchnorm\_3' Batch Normalization Batch normalization with 64 channels

12 'relu\_3' ReLU ReLU

13 'conv\_4' Convolution 64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]

14 'batchnorm\_4' Batch Normalization Batch normalization with 64 channels

15 'relu\_4' ReLU ReLU

16 'maxpool\_3' Max Pooling 2x2 max pooling with stride [2 2] and padding [0 0 0 0]

17 'fc' Fully Connected 10 fully connected layer

18 'softmax' Softmax softmax

19 'classoutput' Classification Output crossentropyex with '0' and 9 other classes

**Network complexity in terms of learnable parameters:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Layers** | | Activation Shape | Activation Size | Learnable parameters |
| Input Layer | | (28×28) | 784 | 0 |
| **Layer 1** | CONV1 (f = 3, s=1, p=1), Nc = 32 | (28×28×32) | 25,088 | (3×3×32) +32 = 320 |
| BatchNorm1 | (28×28×32) | 25,088 | 32 + 32 = 64 |
| **Activation function for layer 1 = ReLU** | | | | |
| **Layer 2** | Max Pool 2 (f =2, s=2) | (14×14×32) | 6,272 | 0 |
| **Layer 3** | CONV3 (f = 3, s=1, p=1), Nc = 32 | (14×14×32) | 6,272 | (3×3×32) +32 = 320 |
|  | batchNorm3 | (14×14×32) | 6,272 | 32 + 32 = 64 |
| **Activation function for layer 3 = ReLU** | | | | |
| **Layer 4** | Max Pool 4 (f=2, s=2) | (7×7×32) | 1,568 | 0 |
| **Layer 5** | CONV5 (f = 3, s=1, p=1), Nc = 64 | (7×7×64) | 3,136 | (3×3×64) +64 = 640 |
|  | batchNorm5 | (7×7×64) | 3,136 | 64+64 = 128 |
| **Activation function for layer 5 = ReLU** | | | | |
| **Layer 6** | CONV6 (f = 3, s=1, p=1), Nc = 64 | (7×7×64) | 3,136 | (3×3×64) +64 = 640 |
| batchNorm6 | (7×7×64) | 3,136 | 64+64 = 128 |
| **Activation function for layer 6 = ReLU** | | | | |
| **Layer 7** | Max Pool 7 | (3×3×64) | 576 | 0 |
| **Layer 8** | Fully connected layer 8 | (10×1) | 10 | (576×10) +10 = 5,770 |
| **Soft max Layer** | | | | |
| **Classification Layer** | | | | |
| **Total Learnable Parameter** | | | | 8,074 |

**Hardware resource:**

The calculation was run on a multi core GPU enabled computer using MATLAB R2018a with built-in custom CNN function “trainNetwork”. Since training time depends on system hardware we provide them here for review:

1. OS Name Microsoft Windows 10 Home
2. Processor Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz, 2601 Mhz, 4 Core(s), 8 Logical Processor(s)
3. System Type x64-based PC
4. Installed Physical Memory (RAM) 16.0 GB

Graphics memory:

1. Name NVIDIA GeForce GTX 1060; Adapter RAM (1,048,576) bytes
2. Name Intel(R) HD Graphics 530; Adapter RAM 1.00 GB (1,073,741,824 bytes)

**Training time:**

Training on a single GPU with the MNIST data took **1 minute and 37 seconds**.

**Training accuracy and testing accuracy:**

Training accuracy was found to be 99.44 %.

Testing accuracy was found to be 99.07 %.

**Live demo description**

The Neural network code must be run first to train and save the CNN generated by using the MNIST data set. Finally, the image processing code must be run which uses the pre-trained CNN.

The code implemented allows for the ability to analyze an image that is: 1) found in the file directory, 2) taken via smartphone, or 3) captured by the onboard webcam. It should be noted that if the onboard webcam is used, the camera will continuously capture images. User is provided with a UI to choose between options.

The image to be analyzed will consist of the target area, a piece of paper bounded with a black box around the edges that will contain the numbers to be processed, as well as anything included in the background. The image must be preprocessed to find the target area, before being classified by the CNN. This is done by using thresholding and finding the bounding box present on the sheet of paper. This enables cleaner detection and avoiding extraneous objects beyond the area of interest. The thresholding is done via a median intensity value-based block filter and the objects are found using the “regionprops” command in MATLAB. The objects within the bounding box are filtered based on a minimum size criterion and arranged from left to right followed by top to bottom. For classification pre-trained CNN is used and the digits which are detected are surrounded by a bounding box with the text label of the digit displayed at the top left corner. The text label is provided to enable visual verification of the CNNs output.

**Demo component for extra credit: Playing the piano in C major**

The code will displays a figure of the image with the target area highlighted as well as the digits after the camera captures digits written on a sheet of paper with a bounding box. Another figure will be displayed afterward, which shows each digit that was classified using the neural net. The code will automatically sort detected digits from left to right and top to bottom much like the regular reading order of the English language. For each label from 0 to 9 that is written on the sheet of paper and detected by the code a tone will be played. The tones start from middle C of an 88 key piano with corresponding digit being 0 and consequently every digit until 9 corresponds to consecutive tones (including sharps and flats). After the piano music is played the user has the option to update the sheet such as by adding numbers or showing a new arrangement which is then played again.

Thus we have made a simple application for playing the piano using numeric keys.