Digits And Characters Recognition Using Convolutional Neural Network

Yiyue Zhang

May 10, 2017

1 Abstract

Image Classification, as one of the fundamental topics in Machine Learning, have been developed and its classification accuracy have been boosted by using Convolutional Neural Networks. Today, with a relative high performance in digits classification, people want to take advantage of it in real life. For example, one famous laptop application, MyScript MathPad, can convert your handwritten mathematical expressions into their digital equivalents for easy sharing [4]. This is a breakthrough for paperless writing on iPads and smart phones.

In my project, a Convolutional Neural Network is constructed in C language on DSP board to perform handwritten digits and characters conversions in a relative short time. It is aimed to work for applications, like conversions from handwritten documents to their printed forms. To be specific, users are required to write some characters or digits (by users' choices) on a white paper and within gridlines on the monitor. Their writings will be captured by camera, and then be displayed in their printed forms on a monitor. This project has three major parts: pre-training the Neural Network with data on handwritten digits and characters (The Chars74K Dataset and collected data from classmates), constructing the network using C language, and testing the performance with camera inputs from users' handwritten characters and digits.

2 Description of the system

This project contains three lab equipments–a camera, a monitor, and a DSP board, as shown in Figure 1. The system is not restricted to certain types of cameras and monitors, and could be functional with any user-specified camera and monitor. In addition, white papers and pens on black and blue colors are provided as basic supply. Technically, the system is designed to work for writings in any pen colors. However, lab equipments only supplied pen in black and blue colors.

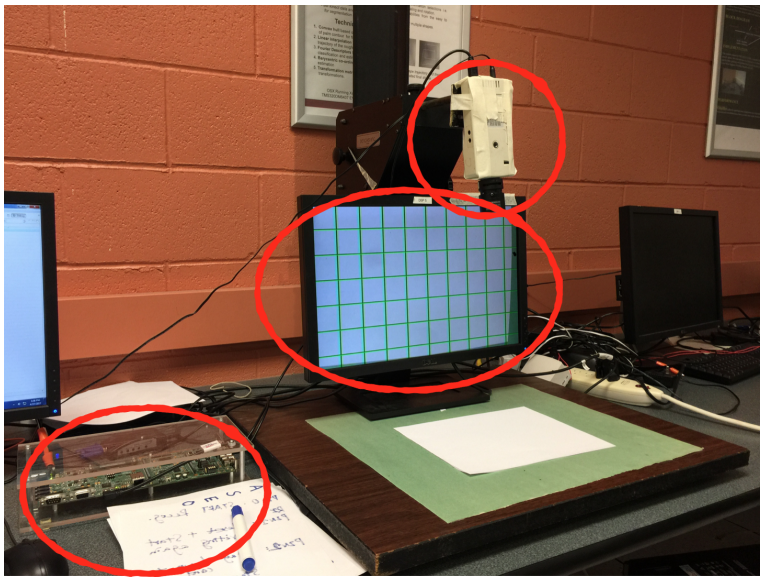


Figure 1: System Setup

Users are required to write some characters or digits (by users' choices) on a white paper and within gridlines on the monitor. One sample of the operating system is shown in Figure 2. Users are also required to signal the system the end of their writings. The camera will be place on top of the white paper and be used to capture users' handwritings. Images from the camera will be used as inputs for the Convolutional Neural Network. After processing by the network, each character should be outputted on the monitor in same physical positions as they are on the white paper. The DSP board will be used to process the Neural Network.

The design of this system is to simulate the process of a real-life writing process. In real life, people write on white papers or touch-screen devices. Their writings should be captured either by a camera or the screen of the device. Therefore, in my system, a camera is placed above the white paper to capture users' writings and to minimize the influence from image dilation and rotation. For real world application, the input to the network in my system can be replaced with any images, like cell photos, screenshots, or scanned documents. And the DSP board can be replaced with other media processors, like Cisco Processor on iPhone and Intel Media Processor on laptops.

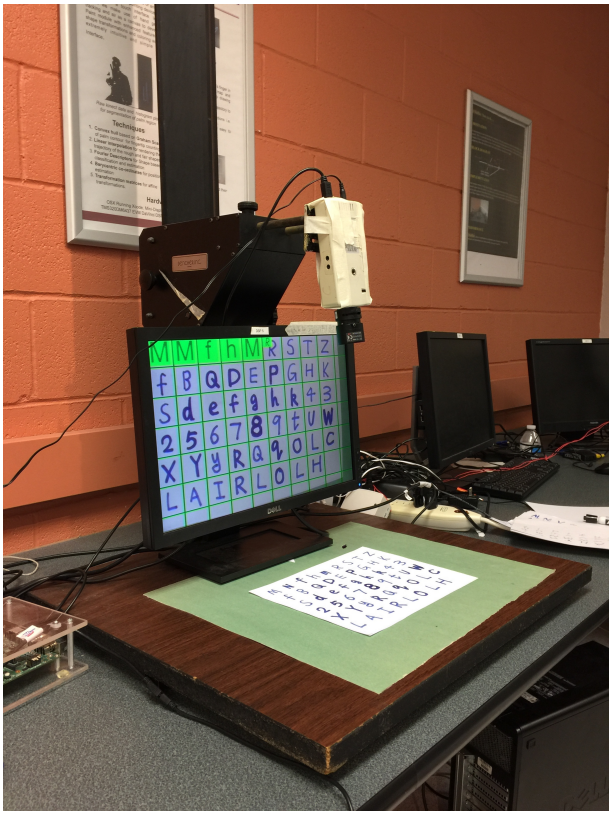


Figure 2: Sample of the operating system

3 Description of Possible Algorithms

This project is developed based on the Convolutional net LeNet-5 and Tensorflow tutorials about training networks with MNIST Dataset. MNIST is a simple computer vision dataset and consists of images on handwritten digits (0-9), shown in Figure 3 [2]. My project is similar with the training for MNIST. It trains a network of 63 classes on handwritten characters and digits (A-Z, a-z, 0-9, white) with images similar with the MNIST Dataset, shown in Figure 4 [7]. Thus, networks for MNIST are practical to be used in this project with minor modifications. There are three major algorithms used/developed in this project. They are the Softmax model, LeNet-5 model, and the Deep MNIST model from Tensorflow tutorials.

Figure 3: Samples from MNIST

Figure 4: Samples from The Chars74k Dataset

For the Softmax model, it applies the Softmax Regressions, which can assign probabilities to an object being one of several different things. This algorithm has two major steps. The first step is to add up all the evidence for the input to be in certain classes. And, the second step is to convert evidence into probabilities. Figure 5 shows the detailed mathematical expressions of this algorithm. Evidence for each class i is calculated by matrix multiplication of the Weight matrix and the input image, and then followed by a matrix summation of the bias, where W_i is the weights and b_i is the bias for class i, and j is an index for summing over the pixels in our input image [2]. After getting the evidence for each class, the next step is to convert the evidence to probability by calling the softmax(x) function, shown in Figure 5.

$$\text{evidence}_i = \sum_j W_{i,\,j} x_j + b_i$$

$$y = \text{softmax}(\text{evidence})$$

$$\text{softmax}(x) = \text{normalize}(\exp(x))$$

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

Figure 5: The Softmax Regression

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

Figure 6: The Softmax Regression in Matrix Form

The second algorithm is the Convolutional net LeNet-5, which is a special kind of multi-layer neural network and designed to recognize visual patterns directly from pixel images with minimal preprocessing [5]. LeNet-5 is the latest convolutional network designed by Yann LeCun for handwritten and machine-printed character recognition. LeNet-5 consists of 5 layers– two convolutional layers, two fully connection layers, and one gaussian connection layer, shown in Figure 7. One sample test of LeNet-5 is shown in Figure 8. By passing a input image to the network, it can extract features of the image and classified it to its corresponding class. This model can get a 93 percent accuracy in classifying handwritten digits.

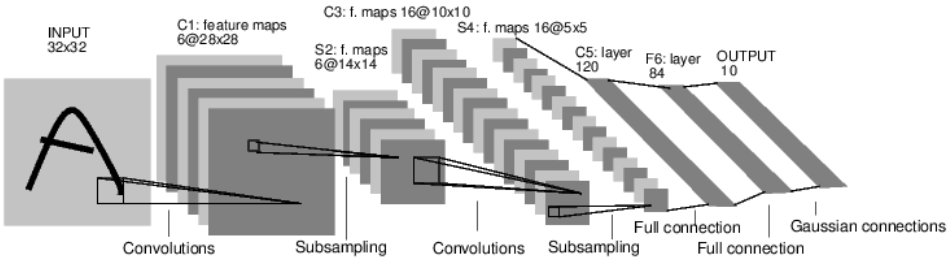


Figure 7: Lenet-5 Network Structure

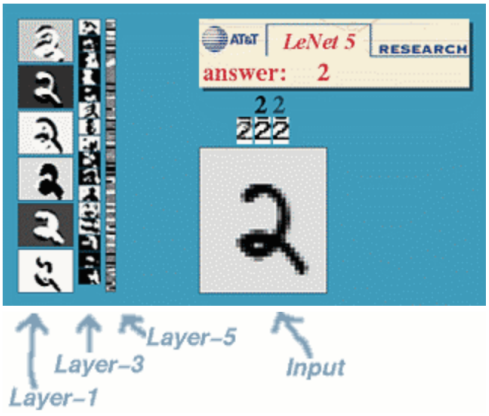


Figure 8: One sample from Lenet-5

The third algorithm is the Deep MNIST model from Tensorflow tutorial. This model follows the same structure as

LeNet-5, and adds dropouts to prevent overfitting in network training. They add dropout between two full connected layers. With the help from dropout, their network can achieve a 99 precent accuracy on MNIST dataset [1].

4 Complexity Analysis

In this project, the complexities come from four different aspects.

The first aspect is the memory size of the Neural Network. In order to apply the network on DSP board, pre-trained weights and biases need to be saved and loaded to the DSP board. Therefore, the total size of all weights and biases files should fit to the memory space of the DSP board. This leads to a limitation on the size of my network. Using less layers inside the Neural Network can reduce the memory requirement on DSP board, but can also lead to a lower accuracy.

The second aspect is from the training of the Neutral Network using TensorFlow. This project is based on applying a pre-trained Neural Network on DSP board. To do this, I need to train a network with a total of 63 classes (A-Z, a-z, 0-9, white) and on a small amount of training samples. Choosing right algorithms for the Neural Network are essential to achieve a high enough accuracy for the actual application.

The third complexity is from the testing speed. For a good user experience, a relative high speed in recognition is required. In this project, because we are using neutral network, huge number of computations are required. From the time that the user's writing is captured by the camera to the time that results are displayed on the monitor, the wait time need to be minimized, considering of real life applications.

The last complexity is the data collection. In the MNIST database of handwritten digits, there are 60,000 training samples and 10,000 test samples [6]. However, in my project, there are only 55 samples per class from the Chars74k. Therefore, collecting more data is required to boost the accuracy of the system.

5 Major Challenges

Basing on Complexity Analysis, the challenges are coming from similar aspects. They will be addressed in this section respectively.

First, to solve memory over-loading problem for weights, biases and other temporary variables, I add them to ".user" directory, instead of using external memory, shown in Figure 9. This solves memory over-loading problem. I also reduce the size of fully connected layers to lower the use of memory space. The fully connected layers initially are of size 2048*512 and 512*63. I reduced them to 2048*128 and 128*63. This change makes the memory requirement 4 times smaller.

```
#pragma DATA_SECTION(buffer_out,".ddr2")
#pragma DATA_SECTION(buffer_in,".ddr2")
#pragma DATA_SECTION(Buffer_input, ".user")
#pragma DATA_SECTION(display_chars, ".user")
#pragma DATA_SECTION(tmp_char_display, ".user")
#pragma DATA_SECTION(test_data_dsp, ".user")
#pragma DATA_SECTION(conv1_weights, ".user")
#pragma DATA_SECTION(conv1_bias, ".user")
#pragma DATA_SECTION(conv2_weights, ".user")
#pragma DATA_SECTION(conv2_bias, ".user")
#pragma DATA_SECTION(fc1_weights, ".user")
#pragma DATA_SECTION(fc1_bias, ".user")
#pragma DATA_SECTION(fc2_weights, ".user")
#pragma DATA_SECTION(fc2_bias, ".user")
#pragma DATA_SECTION(y, ".user")
#pragma DATA_SECTION(output1, ".user")
#pragma DATA_SECTION(temp_array1, ".user")
#pragma DATA_SECTION(evidence1, ".user")
#pragma DATA_SECTION(output2, ".user")
#pragma DATA_SECTION(temp_array2, ".user")
#pragma DATA_SECTION(evidence2, ".user")
#pragma DATA_SECTION(reshape_img, ".user")
#pragma DATA_SECTION(input_padded1, ".user")
#pragma DATA_SECTION(input_padded2, ".user")
#pragma DATA_SECTION(temp_img, ".user")
```

Figure 9: Data Saving Location on DSP Board

Second, because my network has 63 classes and with much less training samples, choosing right network are essential. I decide to follow LeNet-5, but add dropout after each layer to reduce overfit. And instead of following the Tensorflow tutorial to use the Gradient Descent Optimizer, I choose to use the Adam Optimizer to make the training process converging faster. These modifications leads to a near 88 percent accuracy for my system. Detailed network structure will be introduced in the Training Section. Moreover, I also added a voting system to boost the accuracy further.

Third, as I mentioned, a relative fast speed of recognition is required to improve user's experience. I optimized my functions to make them run faster by using less for loops and combining multiple calculations to one. Also, adding the voting system slows down the recognition process because one same image needs to go through the network three times with different black-white thresholds. Although, the voting system can increase the accuracy, but the slow down effect leads to a less-satisfied user's experience. I need to decide if to add the voting system to the final demo.

To solve the last challenge on the number of training samples, I collected data from camera inputs by asking classmates to writes characters for me. Their writings give a diversity for network training, because letters are written with different styles, at different positions inside the gridlines, and at different angles. With these variation, when I pre-process the collected data, I do not need to dilate or rotate the images. I collected 54 samples per class from my classmates. And I converted them to black and white images with multiple thresholds to increase the training set.

6 Modeling Tasks

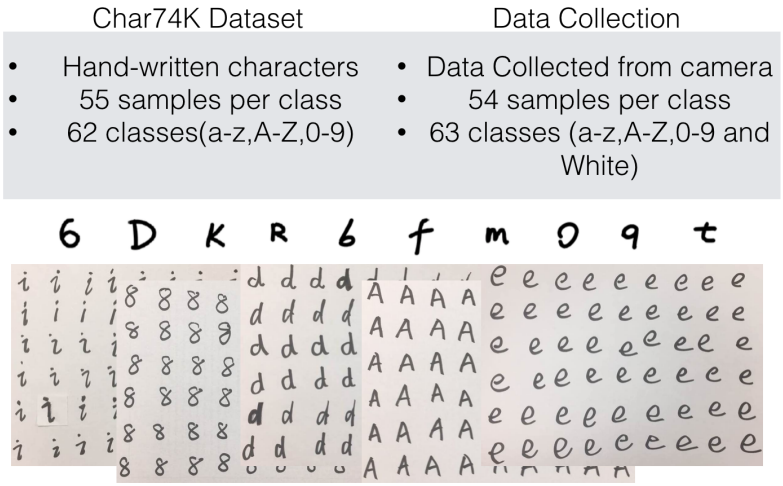


Figure 10: Dataset and Collected Data for the Training of Neural Network

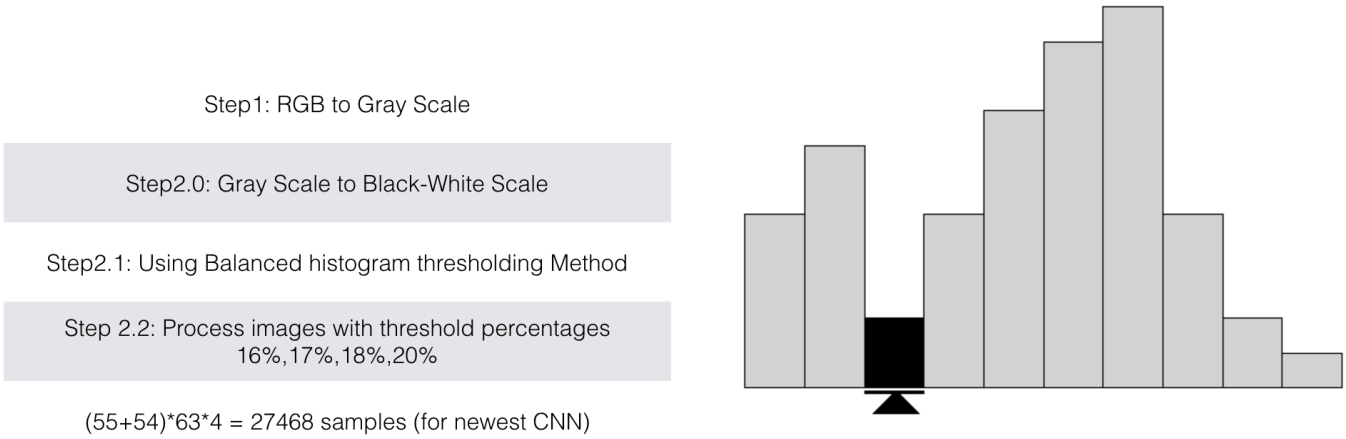


Figure 11: Data Processing and The Balanced Histogram Thresholding Method

Dataset Chars74k, which is mainly used for on-line handwritten character recognition methods, and my collected data from lab camera are used in this project to train and test the Neural Network, shown in Figure 10. The Chars74k dataset consists of 62 classes (26 UpperCase + 26 LowerCase + 10 Digits) with 55 samples per class. My collected data consists of 63 classes (26 UpperCase + 26 LowerCase + 10 Digits + white) with 54 samples per class.

For data processing, I first convert them from RGB to gray scale, and then to black-white images. I choose to use the Balanced Histogram Thresholding Method for conversion [3]. This method counts the number of pixels in each value between 0-255 and set certain percentage of the pixels to be 0 and others to be 255, shown in Figure 11. I processed the data by set threshold percentages to be 16 percent, 17 percent, 18 percent, 20 percent (basing on normal lab environment). After data processing, I get 436 samples per class and a total of 63 classes.

7 Human Factors

There are three major human factors in the recognition process on the DSP board.

First of all, this project requires users to provide their handwriting characters and digits on a given white paper with gridlines on monitor. Therefore, the size of characters that the user writes is important. If the handwriting is too small, it will make the recognition process harder due to the effect from camera's resolution. To resolve this issue, I can put a bounding box around each character to remove its size dependency. However, time was not enough for me to implement this functionality.

Second, for a good detection on handwriting characters, users must write inside gridlines. If they write outside or on gridlines, my network will not be able to recognize the character. To solve this problem, I should remove all gridlines on monitor and implement a auto-detection ability to locate characters on white paper, rather than force user to writes inside specific locations.

Third, the quality of handwriting can influence the performance as well, for example, care writing can lead to a high accuracy, and poor writing will result more confusions. To remove this human factor, I should collect more data for each class. This can help to further increase the diversity of training samples, and therefore, leads to a higher accuracy on DSP testing.

8 Training

This project involves the training of a small Neutral Network using TensorFlow. To do this, I need to train a network with a total of 63 classes (A-Z, a-z, 0-9, white) and on a small amount of training samples. I decide to follow LeNet-5, but add dropout after each layer to reduce overfit, as the dataset I used is small. Also, instead of following the Tensorflow tutorial to use the Gradient Descent Optimizer, I choose to use the Adam Optimizer to make the training process converging faster. By using this network, the recognition accuracy can reach 88 percent. Comparing to the 99 percent accuracy on MNIST, 88 percent on this network is satisfactory because this project involves training of 63 classes and with much less training samples than MNIST.

Milestones	Details
TensorFlow Network (Training + Testing)	88%
Converting TensorFlow Model to C	68%, but right code
Recognition on DSP	50%—65%
Recognition Speed without Voting System	About 3 seconds per Character
Recognition Speed with Voting System	About 9 seconds per Character

Figure 12: Network Performance

The detailed Network structure is shown in Figure 13.

Image (Size 32x32)	Weight	Bias	Input Size	Output Size
Convolutional Layer	5x5x1x16	1x16	32x32x1	32x32x16
Relu Layer				
Max Pooling Layer	NONE	NONE	32x32x16	16x16x16
Convolutional Layer	5x5x16x32	1x32	16x16x16	16x16x32
Relu Layer				
Max Pooling Layer	NONE	NONE	16x16x32	8x8x32
Fully Connected Layer	2048x128	1x128	1x2048	1x128
Fully Connected Layer	128x63	1x63	1x128	1x63
Softmax Layer	NONE	NONE	1x63	probability

Figure 13: Convolutional Neural Network Structure

There are four major functions for my neural network, shown in Figure 14. They are the conv2d-relu(), max-pool(), fc(), and softmax(). For the conv2d-relu() function, it is a void function and takes several parameters–the function input in array form, the after-padding input, the function output array, the weight matrix, the bias matrix, the filter size (F), the output depth (K), and the stride (S). This function assumes the output size (height and width) should be the same as the input. This function also includes the Relu layer by assigning all negative elements on the output array to 0. For the max-pool() function, it reduces its input dimension to half and takes in the input array and its dimensions as parameters (dim-h–helight, dim-w–width, dim-d–depth.) Function fc() stands for the fully connected layer. It takes input (x[]) , output (evidence[]), the weight matrix, the bias, the input dimension (input-num), and the output dimension (middle-layer.) The softmax() function takes in the evidence from fully connected layer and returns the recognized class number.

Functions	Detail
Convolutional Layer + Relu Layer	`void conv2d_relu( float input[],float input_padded[], float output_conv[], float weights[], float biases[], int K, int F, int S, int input_h, int input_w, int input_D)`
Max Pooling Layer	`void max_pool(float input[], float max_pool[], int dim_w, int dim_h, int dim_d)`
Fully Connected Layer	`void fc(float w[], float b[],float evidence[],float x[],int middle_layer, int input_num)`
Softmax Layer	`int softmax(float evidence_s[],int class_num)`

Figure 14: Convolutional Neural Network Functions

For my network, there exists two future works. First, my network can get as high as 88 percent in Tensorflow (training and testing). However, when I load saved weights and biases to the same network in Tensorflow and my C network for testing. The accuracy drops to 68 percent. This indicates that the C model I constructed is indeed right, because it gives same accuracy as that of in Tensorflow testing model. I think this problem may come from weights and biases saving and loading. There might be a precision lose when I saved weights and biases.

The second future work is on the accuracy for specific classes, shown in Figure 15. I notice for some classes the accuracy can get more than 90 percent. While for some other classes, the accuracy can drop below 10 percent. I think this problem might come from the data processing section. I automatically segmented my collected data to 64x64. There might be misalignment for some classes, shown in Figure 16. To solve this problem, instead of automatic segmentation, I should crop characters by putting bounding boxes around them.

Classes: X, V, M, N, W, E, F, R, T, P, Y, H, K, k, S, A, Q, Z, L	High Accuracy
Classes: D, C, B, i, j, o, r, 1, 2, 8, 0, I, l, e	Low Accuracy
Other Classes	Moderate Accuracy
Observation: Problem might come from collecting data segmentation	

Figure 15: Accuracy for Specific Classes

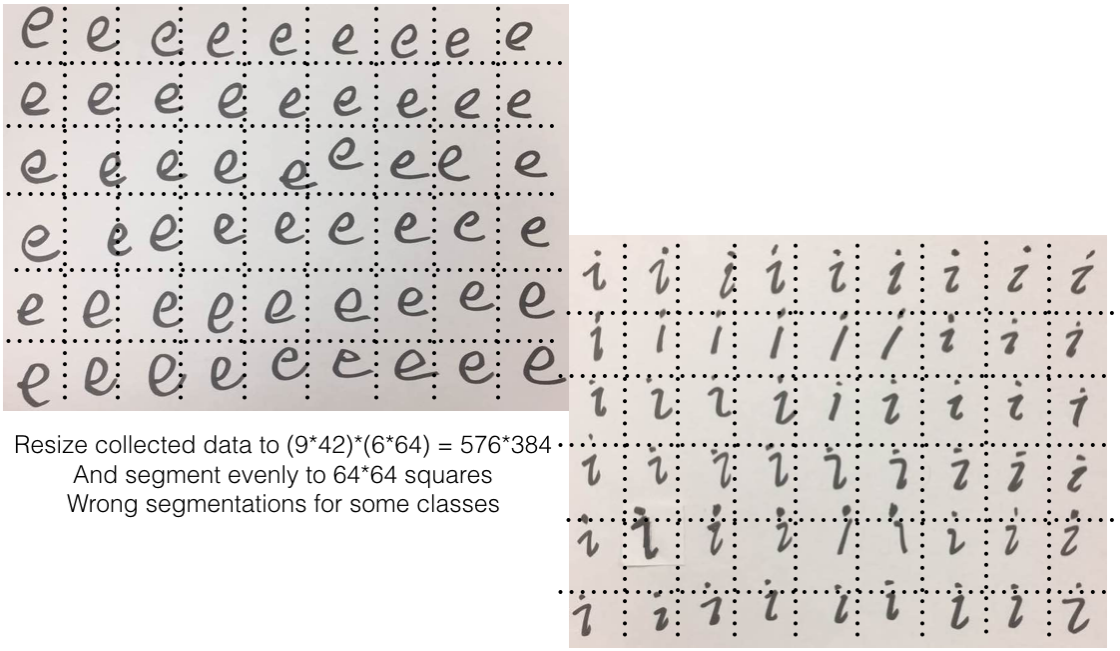


Figure 16: Accuracy for Specific Classes

9 Rough schedule

This section reports the rough schedule of my work for the project though out the semester, shown in Figure 17.

10 Final Test Set-Up

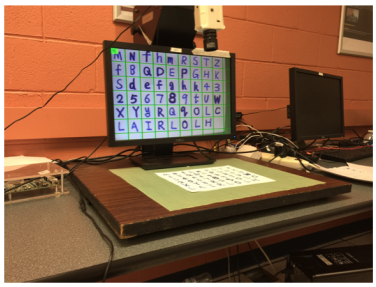
The final test set-up is shown in Figure 18. As mentioned in section 2, a camera and a monitor are connected to the DSP board as input and output respectively. The camera is placed above the write paper to capture user's handwriting. And gridlines are printed on the monitor to guide user's writing. During testing, user can write any characters (Uppercase or Lowercase) or digits on a white paper. When user finishes their writings, they should press PIN1 to start the recognition process. It will first display the black-white image of the character on the corresponding position, and then display the printed form. This process takes about 3 seconds without the voting system and 9 seconds with voting system. If the user wants to reset the system, PIN2 should be pressed down. If the user wants to pause the system, PIN3 should be pressed. And, pressing PIN2 and PIN3 at the same time will allow user to start a new writing process. The accuracy of my system is around 60 precent on DSP board, and can get as high as 72 precent.

11 Board

In this project, Texas Instruments DaVinci TMS320FM6437 board is used and with connection to a video camera and a monitor. Additional software, like TensorFlow, MATLAB and xCode, are used to perform training and testing of the Convlolutional Neural Network.

What I did	Tranning Samples	Test Samples	Problems	Accuracy
Step1: Training with Softmax	All samples Char74k	10 samples per class	Overlapping Train and Test	Above 90%
Step2: Training with 1 Convolutional Layer	All samples Char74k	10 samples per class	Overlapping Train and Test	Above 80%
Step3: Separate Training and Testing samples	45 samples per class	10 samples per class	Low accuracy	Below 60%
Step4: Using 2 layers of Conv, MaxPool, and Fully connected, and 1 layer softmax	45 samples per class	10 samples per class		Around 70%
Step5: Converting TensorFlow functions to C function (Conv2d, Maxpool, FC, and Softmax)	45 samples per class	10 samples per class	Inaccurate on DSP	
Step6: Data Collection	No preprocessing		Depends on environment Low Accuracy	
Step7: Balanced histogram thresholding Method	Char74k and collected data		Better than before	72% TF
Step8: Train the network in Step 4 again	Char74k and collected data		Weights and Bias saving	88% TF
Step9: DSP testing	Direct Camera Input		Ramdom Output	Low Accuracy on
Step10: Debug network on C	Direct Camera Input		Improved	Lower than expected
Step10: Compare network in C with TF	Char74k and Collected Data		Cannot reach 88% accuracy	
Step11: Focusing on Final Test		Direct Camera Input And Pins (1-3)		
Overall: The performance on DSP can get around 60% accuracy. Some classes with nearly 90% accuracy. Some with less than 10% accuracy.				

Figure 17: Rough Schedule



Write inside grids on Monitor

PIN1: Start the Recognition Process
PIN2: Reset the System
PIN3: Start/Pause the Recognition Process
PIN2+3: Start Writing Process

Accuracy: 39/54

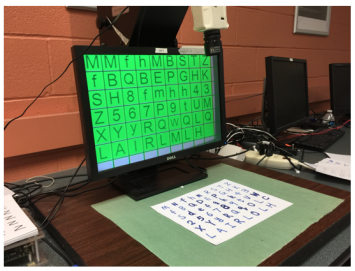


Figure 18: Final Test Setup

12 References

References

[1] Deep mnist for experts. 2013.

[2] Mnist for ml beginners. 2013.

[3] Balanced histogram thresholding. 2017.

[4] Myscript mathpad - handwriting latex generator on the app store. April 26, 2017.

[5] Yann LeCun. Lenet-5, convolutional neural networks.

[6] Yann LeCun. The mnist database of handwritten digits.

[7] T.deCampos. The chars74k dataset. 2012.