CMPT 412: Project 1

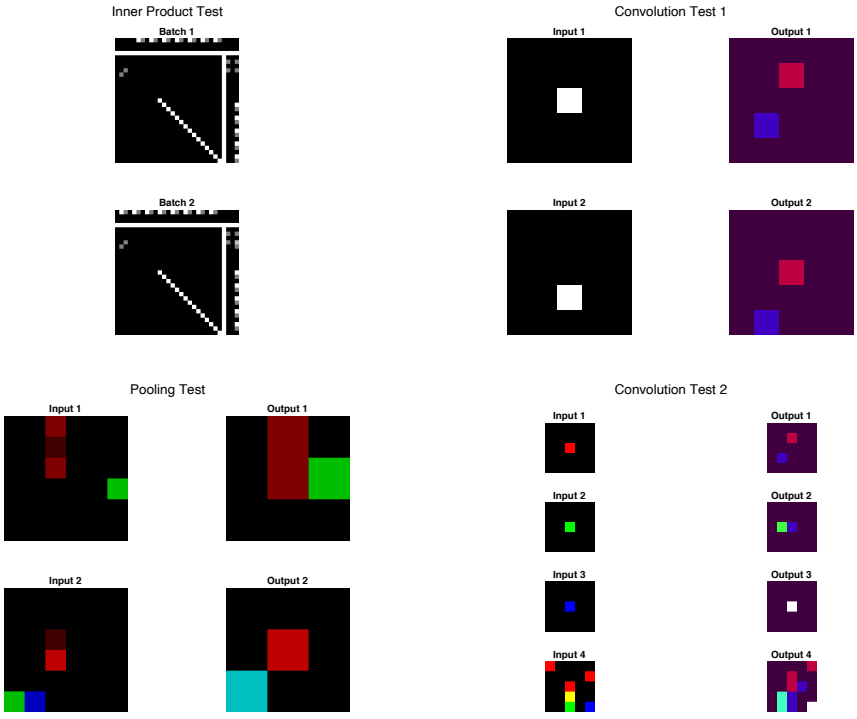
Digit recognition with convolutional neural networks

Name: Parsa Alamzadeh
Student #: 301316272
Computer ID: palamzad@sfu.ca
Discussed with: Eric Sund; 301284359

Part 1: Forward Pass

- Inner Product: this portion of the program was simply done by transpose of weights multiplied by each image in the batch and then the result is added by the bias
- Max Pooling: the kernel passes through the pixels with spacing of provided stride.
- Convolution Layer: there are two approaches for this layer, one is to aggregate the results from all channels and create the output channel for each filter number, or use the im2col function which creates a vector of pixels. The vectors multiplied by the weights produces the convolution of the image. This way is way faster since, matrix multiplication is optimized in matlab and decreases the training speed significantly.
- Relu layer: This layer is just the element = max(element,0) for each element in the input matrix.

Test components results:



Part 2: Back Propagation

Assuming that our output is the following:

$$h_i = f_i(w_i, h_{i-1})$$

We have the following for the derivative our loss with respect to our activation function:

$$\frac{\partial l}{\partial w_i} = \frac{\partial l}{\partial h_i}\frac{\partial h_i}{\partial w_i}$$

$$\frac{\partial l}{\partial h_{i-1}} = \frac{\partial l}{\partial h_i}\frac{\partial h_i}{\partial h_{i-1}}$$

- Relu layer: for the forward layer we have the followings:

$$relu(x) = \begin{pmatrix} x & x > 0 \\ x & x <= 0 \end{pmatrix}$$

$$\frac{dl}{dh_i} = input.data \; if \; x > 0, else \; 0$$

Therefore, the derivative of relu is 1 for x greater than 0 and 0 otherwise.

- Inner Product Layer: we have the following for our IP layer:

$$h_i = wx + b$$

$$\frac{dl}{dw} = \frac{dl}{d_i}\frac{dh_i}{dw_i} = output.diff \times input.data$$

$$\frac{dl}{dh_{i-1}} = \frac{dl}{dh_i}\frac{dh_i}{dh_{i-1}} = output.diff \times weights$$

$$\frac{dl}{db} = \frac{dl}{dh_i}\frac{dh_i}{db} = output.diff \times 1$$

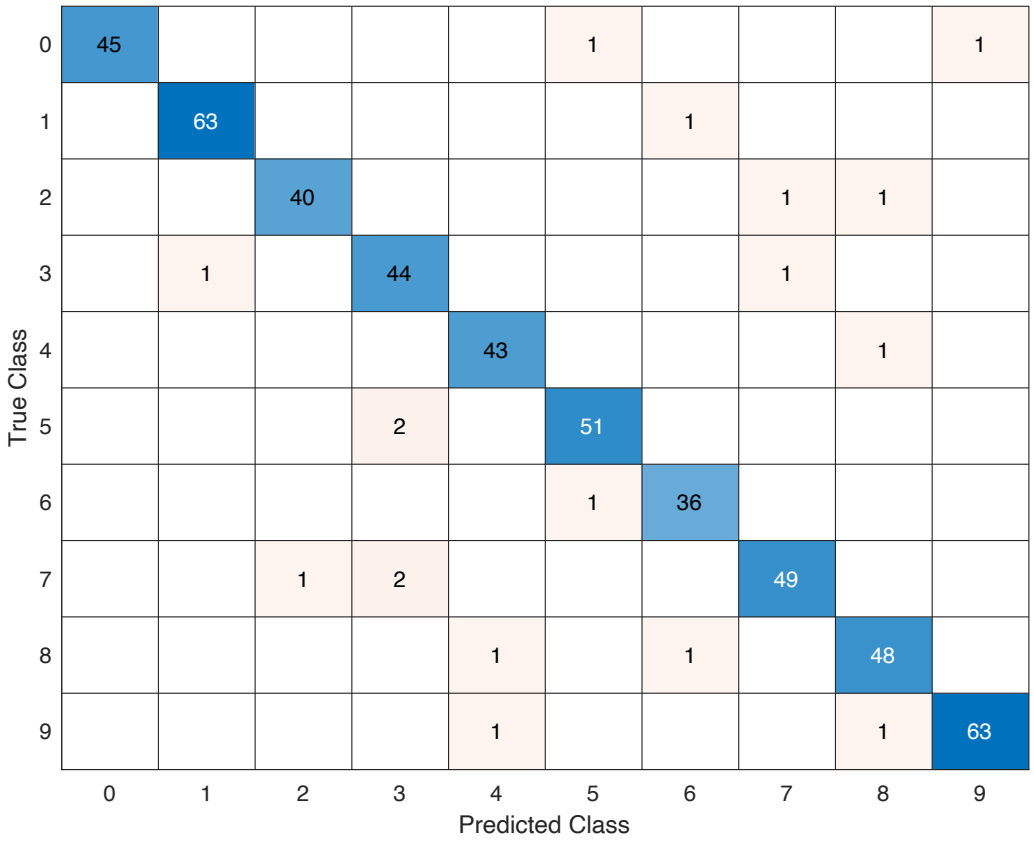
Part 3: Training

Here is the last two lines of train_lenet component:

cost = 0.049833 training_percent = 0.990000
test accuracy: 0.970000

for full log of the train_lenet please refer to res.txt in the main directory.

Confusion Matrix:



The inputs that the network had the most difficulties with are 5 and 7. For 5's I believe the reason might have been the top stroke of 5 was too close the semi-circle underneath it and the network assumed they were connected and that's why it predicted them as 3. As for the 7's I believe the reason the network failed was because the 7's were written with 2 horizontal strokes which threw off the network and that's most probably why it failed to predict them as 7's and predicted them as 3's.

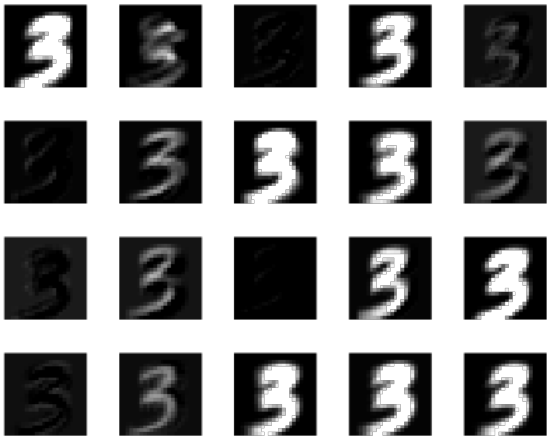
Real world testing:
Samples obtained:

The network predicted all the labels correctly, the code for this portion is available in *test_samples* component.

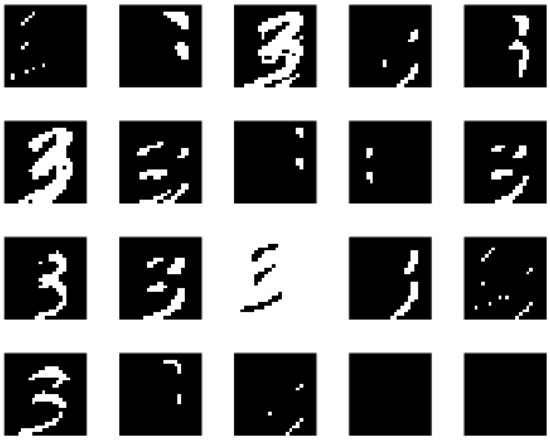
Part 4: Visualization

Input image:

Convolution Layer:



Affected pixels in Relu Layer:



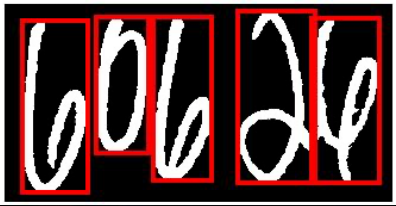
For visualization of relu layer, instead of just visualizing the output of relu function; I made some changes; because imshow function in matlab shows negative pixel values as black so if I were to visualize the output relu function it would have been identical to convlayer. Therefore, what I did was I set the value of all the pixels that would not have been affected by relu function (pixel values > 0) to 0 and pixel values that had values less than 0 to 1. The third figure, therefore, is showing the affected pixels.

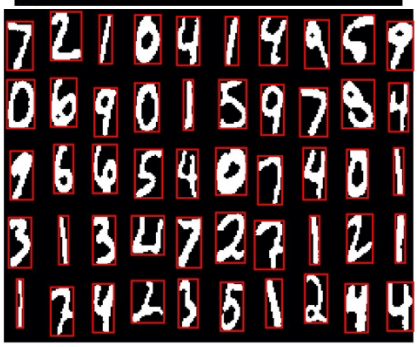
Looking at the convlayer, we see that the network has learned that to find the edges and outline of the digit, while some filters don't even get activated for some digits. For example, the third filter from top and left, seems to be totally black and when we look the modified relu

figure we see that pretty much the output image is black. Remember that black pixels in modified relu figure are untouched and white ones get set to 0.

Part 5: OCR

For this section, I greyscaled all images and used three different thresholds (the fourth image did not work well with the threshold set for the other ones and the third one would find a random pixel that would throw off the prediction) for binarizing the images. After that I dilated the images that had separate components, so that they become connected. After that I used bwlabel and regionprops functions to find connected components. Here are the results:





I then padded images based on the ratio of height and width and the results are as follows:

Testing on image 1, got 6 correct labels out of 10
Testing on image 2, got 6 correct labels out of 10
Testing on image 3, got 4 correct labels out of 5
Testing on image 4, got 40 correct labels out of 50

This means that the network got 56 labels correct out of 75 images. This gives us 74.67% accuracy for this task. The code for this portion is in the ec component, in the matlab folder.