CMPT412: Computer Vision

Project - 1 Report

This project concerns with the implementation of Convolutional Neural Network (CNN). The project flows as follow:

1. Forward Propagation

1.1 Fully Connected Layer / Inner Product Layer (IP)

Here, for given input x, the output can be calculated by f(x) = Wx + b. In my code I used transpose of 'param.w' for W, 'input.data' for x, and transpose of 'param.b' for bias(b). The output image is shown below (fig 1).

1.2 Pooling Layer

Here, the input data is divided into smaller kernels from which the maximum data is extracted with the stride of 2. Two for loops are used, one for output.height, i.e row of the selected kernel and the second for output.width, i.e, column of the selected kernel. Then the maximum value is extracted from that kernel so that the size of feature map is reduced. The output image is shown below (fig 2).

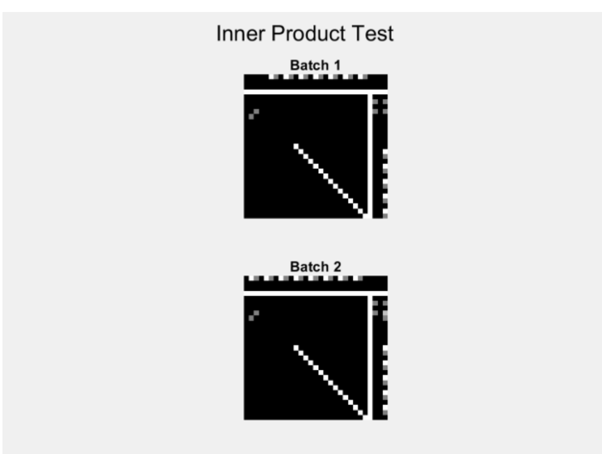


fig 1(Input Product)

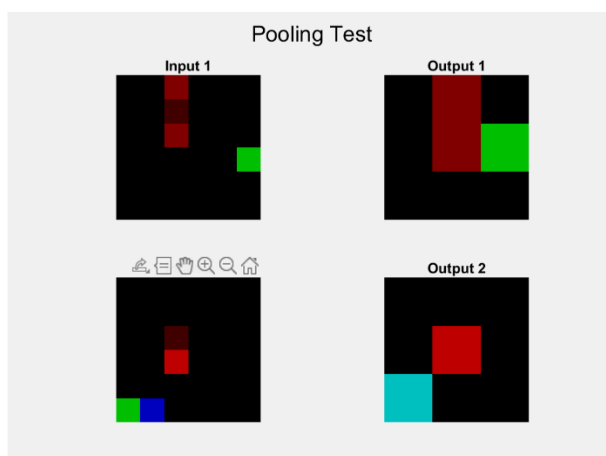
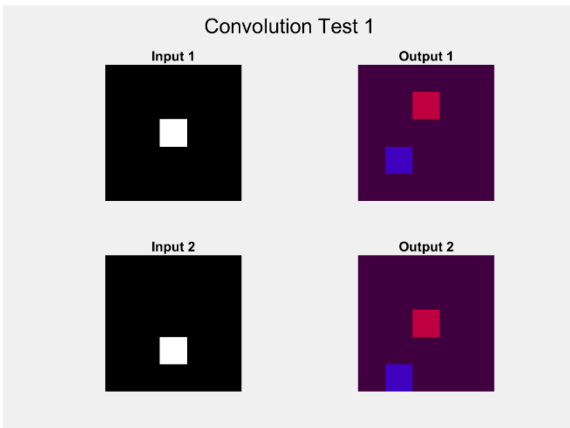


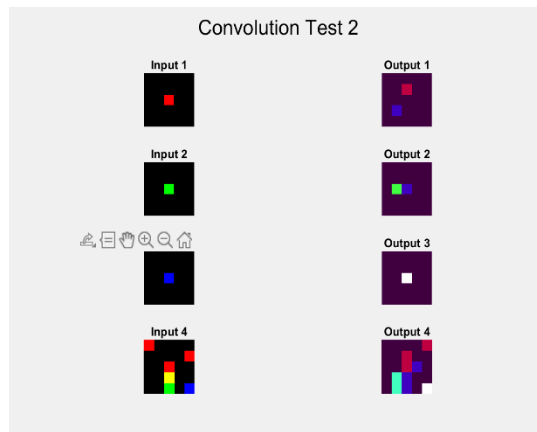
fig 2 (Pooling layer)

1.3 Convolution Layer

In this layer, the loop is iterated on batch_size so that the matrix calculations can be done. For the calculations the helper function `im2col_conv(input_n, layer, h_out, w_out);` is used and then the matrix is reshaped so that it can be used as an input x. Here using the formula f(x) = Wx + b where transpose of param.w is used as W, reshaped matrix is used as x and transpose of param.b is used as bias b.

The output images for convolution Layer are:





1.4 ReLU

It is the simplest function where the negative values are replaced by 0. For output the function, f(x) = max(x,0) is used which ensures that only positive values are in the matrix and the negative values get replaced by 0.

2. <u>Back Propagation</u>

2.1 ReLU

Here, the same concept as above is used where only the positive values are kept in the matrix and the negative values are replaced by 0 via `input_od = input.data > 0;` (I tried using the same function max(x,0) here as well but due to this my accuracy was only 81% for part 3 so I used another approach to achieve the same output). After that the element-wise multiplication is done on the differentiation output.diff and the about input_od and the result is stored in input_od.

2.2 Inner Product Layer

Here, the matrix transpose of output.diff multiplied by input.data is used to calculate the param_grad.w. The sum of transpose of output.diff is used to calculate the param_grad.b and the matrix multiplication of param.w and output.diff is used to calculate the input_od. The differentiation output.diff is actually calculated the same way as discussed in the description of the project.

3. <u>Training</u>

3.1 Training

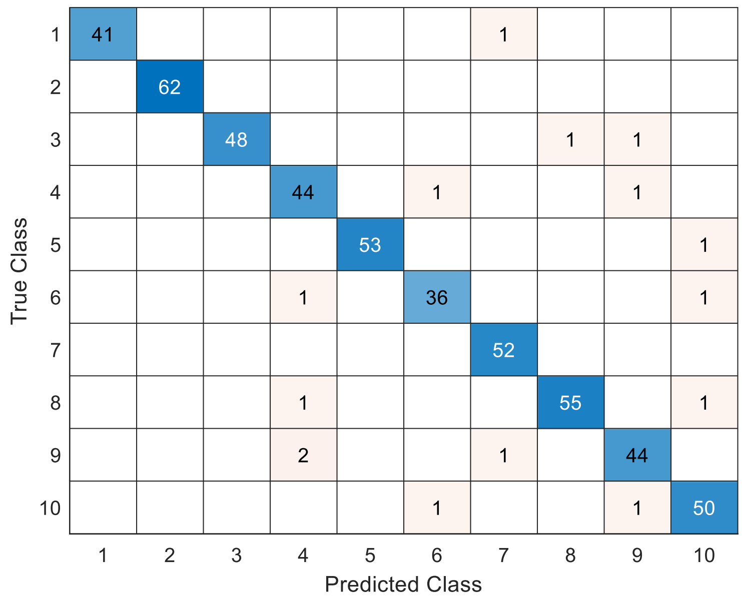
The accuracy of the data is calculated using train_lenet.m, and the accuracy is 97% as shown below:

```
Command Window
>> train_lenet
cost = 0.273491 training_percent = 0.910000
cost = 0.279565 training_percent = 0.910000
cost = 0.176619 training_percent = 0.920000
cost = 0.127344 training_percent = 0.950000
cost = 0.191895 training_percent = 0.960000
test accuracy: 0.944000

cost = 0.192910 training_percent = 0.930000
cost = 0.131836 training_percent = 0.970000
cost = 0.115812 training_percent = 0.970000
cost = 0.103636 training_percent = 0.970000
cost = 0.124224 training_percent = 0.980000
test accuracy: 0.960000

cost = 0.111115 training_percent = 0.960000
cost = 0.113216 training_percent = 0.940000
cost = 0.134874 training_percent = 0.960000
cost = 0.067548 training_percent = 0.990000
cost = 0.095426 training_percent = 0.980000
test accuracy: 0.966000
cost = 0.086685 training_percent = 0.980000
cost = 0.106186 training_percent = 0.950000
cost = 0.034245 training_percent = 1.000000
cost = 0.048397 training_percent = 1.000000
cost = 0.060728 training_percent = 0.970000
test accuracy: 0.968000

cost = 0.069977 training_percent = 1.000000
cost = 0.068312 training_percent = 0.980000
cost = 0.063643 training_percent = 0.980000
cost = 0.084625 training_percent = 0.960000
cost = 0.083214 training_percent = 0.980000
test accuracy: 0.970000

cost = 0.083081 training_percent = 0.970000
cost = 0.026531 training_percent = 1.000000
cost = 0.044653 training_percent = 0.980000
cost = 0.056298 training_percent = 0.980000
cost = 0.049833 training_percent = 0.990000
fx test accuracy: 0.970000
```

3.2 Test the network

Here the confusion matrix of size 10*10 is calculated in test_network.m using convnet_forward function.

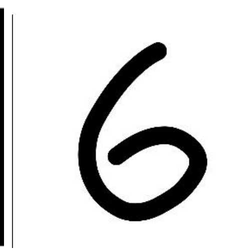
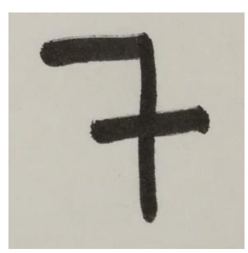
The confusion chart is:



3.3 Real-world testing

Here 5 images are downloaded from the internet MNIST dataset, which are:




A new MATLAB script called real_images.m is created where the predictions are done based on the above 5 images.

The actual numbers are: 2, 4, 6, 7, 8

The predicted numbers are: 2, 4, 3, 8, 8, which is a good prediction.

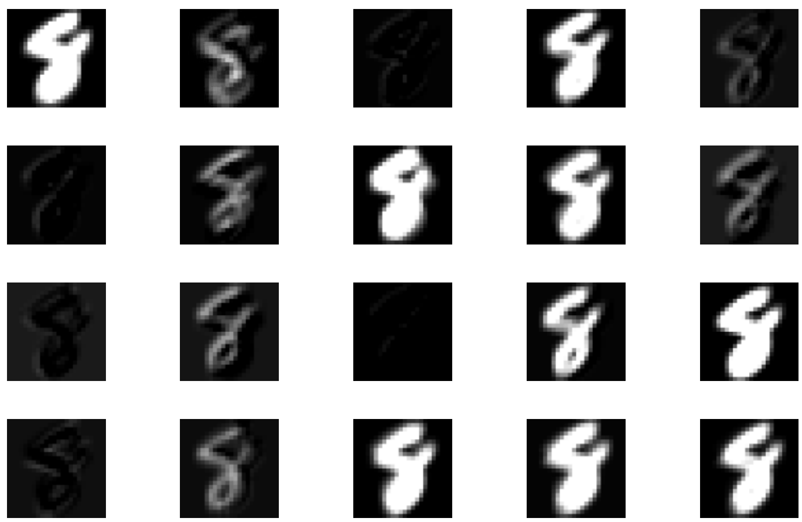
4. Visualization

4.1

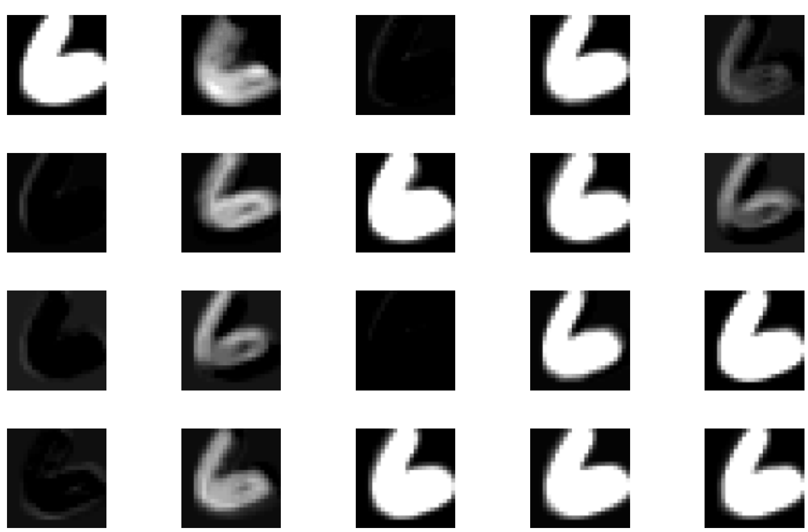
Here, in vis_data.m, the plots of output_2 and output_3 are plotted where output_2 calculates the CONV layer and output_3 calculates the RELU layer.

The images are as below:

The CONV layer:



The ReLU layer:



4.2

The CONV layer and ReLU layer are plotted as above. It can be noted that when these filters are applied to the image, both layers give different features highlighted depending on the weights. The CONV layer shows the different onstrasts of number 8. The ReLU layer focus on increasing the non-linearity in the image with number 6, which have a quite dark contrast which is because ReLU replaces the negative values with 0.

5. Image Classification

Here, firstly the image is converted to grayscale using graythresh. The other parts of ec.m, I didn’t quite understand how to do so I just left it blank.