CMPT412 Project 1

Digit recognition with convolutional neural networks

Student: Sibei Zhou(301416917)
Computer ID: sibeiz@sfu.ca

Part 1: Forward Pass

Q 1.1 Inner Product Layer - 1 Pts

Implemented the function in inner_product_forward.m:

```
% Replace the following line with your implementation.
output.data = zeros([n, k]);
for i = 1 : k
   output.data(:,i) = input.data(:, i).' * param.w + param.b;
end
output.height = input.height;
output.width = input.width;
output.channel = input.channel;
output.batch_size = input.batch_size;
```

Q 1.2 Pooling Layer - 1 Pts

Implemented the function in pooling_layer_forward.m:

```
% Replace the following line with your implementation.
output.data = zeros([h_out*w_out*c, batch_size]);
each_img.data = zeros([h_out,w_out,c]);
each_img.height = h_out;
each_img.width = w_out;
each_img.channel = c;
for b = 1:batch_size
      img_data = reshape(input.data(:, b), input.height, input.width,
input.channel);
      img_data = padarray(img_data, [pad, pad]);
      s = size(img_data);
      for channel = 1:c
          h_i = 1;
          for h = 1: stride: s(1)
              w_i = 1;
              for w = 1: stride : s(2)
                   each_img.data(h_i, w_i, channel) = max(img_data(h: h+k-1,
w: w+k-1, channel),[], 'all');
                  w_i = w_i +1;
              end
              h_i = h_i +1;
          end
      end
      output.data(:, b) = reshape(each_img.data, [], h_out*w_out*c);
end
```

Q 1.3 Convolution Layer - 1 Pts

Implement the function in conv_layer_forward.m:

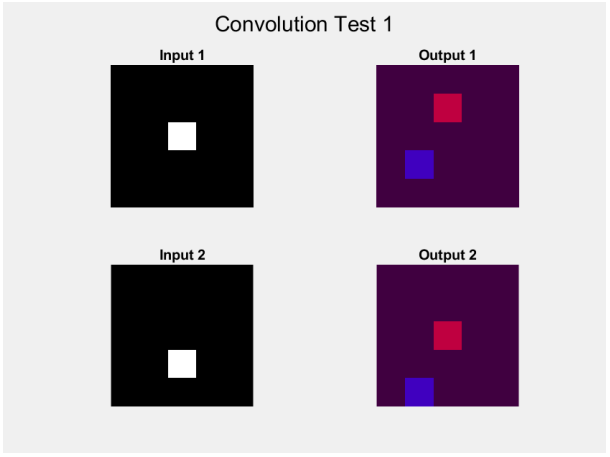
```
%% Fill in the code
% Iterate over the each image in the batch, compute response,
% Fill in the output data structure with data, and the shape.
output.height = h_out;
output.width = w_out;
output.channel = num;
output.batch_size = batch_size;
output.data = zeros([h_out*w_out*num, batch_size]);
% tmp = input;
for batch = 1:batch_size
   each_img.data = input.data(:, batch);
   each_img.height = h_in;
   each_img.width = w_in;
   each_img.channel = c;
   img = im2col_conv(each_img, layer, h_out, w_out);
   img = reshape(img, c*k*k, h_out*w_out);
   result_tmp = img.' * param.w + param.b;
   result = reshape(result_tmp, h_out, w_out, layer.num);
   output.data(:, batch) = reshape(result, [], 1);
end
```

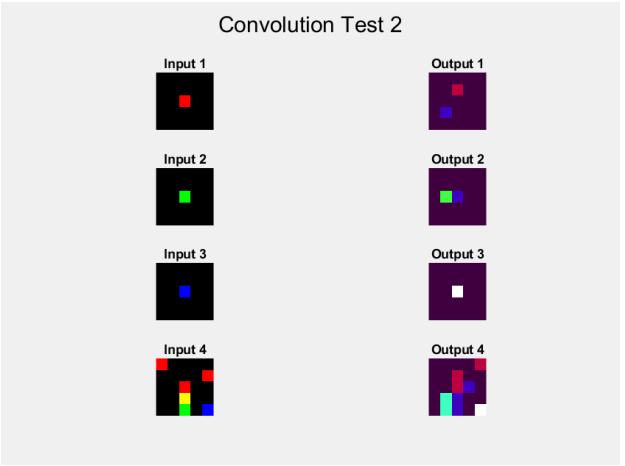
Q 1.4 ReLU - 1 Pts

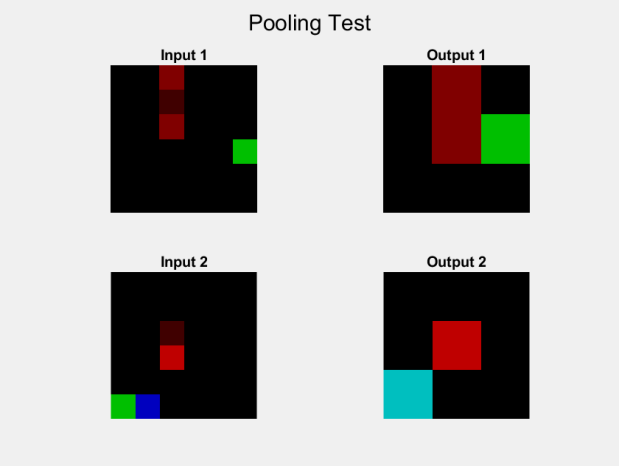
Implement the ReLU function:

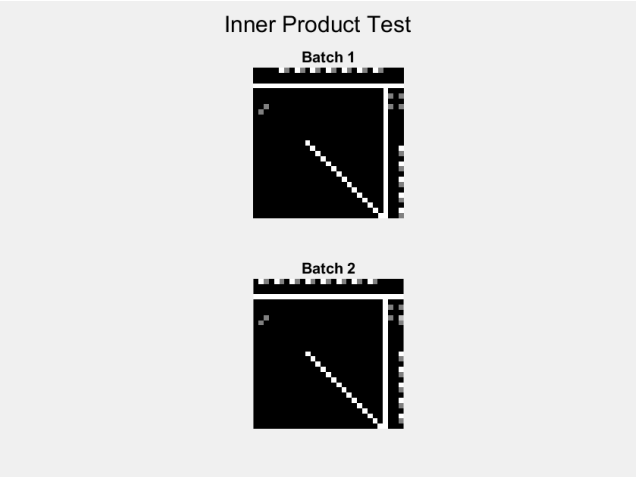
```
% Replace the following line with your implementation.
output.data = max(0, input.data);
```

Results of running the test_components.m:









Part 2 Back propagation

Q 2.1 ReLU - 1 Pts

Implement the backward pass for the Relu layer in relu_backward.m file:

```
% Replace the following line with your implementation.
equal_to = (max(input.data, 0) == input.data);
input_od = output.diff .* equal_to;
```

Q 2.2 Inner Product layer - 1 Pts

Implement the backward pass for the Inner product layer:

```
% Replace the following lines with your implementation.
s = size(output.diff);
param_grad.b = (output.diff * ones(s(2),1)).';
param_grad.w =  input.data * output.diff.';
input_od = param.w * output.diff;
```

Part 3 Training

Q 3.1 Training - 1 pts

Modified the parameter `max_iter` to 5000 from 3000 in in train_lenet.m at line 30 , then ran the test, I got the test accuracy **0.974** at last. And the full log is:

```
>> 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
test accuracy: 0.970000

cost = 0.030189 training_percent = 1.000000
cost = 0.041896 training_percent = 0.990000
cost = 0.039024 training_percent = 0.990000
cost = 0.055760 training_percent = 0.990000
cost = 0.029811 training_percent = 1.000000
test accuracy: 0.972000

cost = 0.029547 training_percent = 1.000000
cost = 0.023075 training_percent = 0.990000
cost = 0.021663 training_percent = 1.000000
cost = 0.034835 training_percent = 1.000000
cost = 0.045802 training_percent = 0.990000
test accuracy: 0.974000

cost = 0.017158 training_percent = 1.000000
cost = 0.022747 training_percent = 1.000000
cost = 0.015806 training_percent = 1.000000
cost = 0.014577 training_percent = 1.000000
cost = 0.020438 training_percent = 1.000000
test accuracy: 0.974000

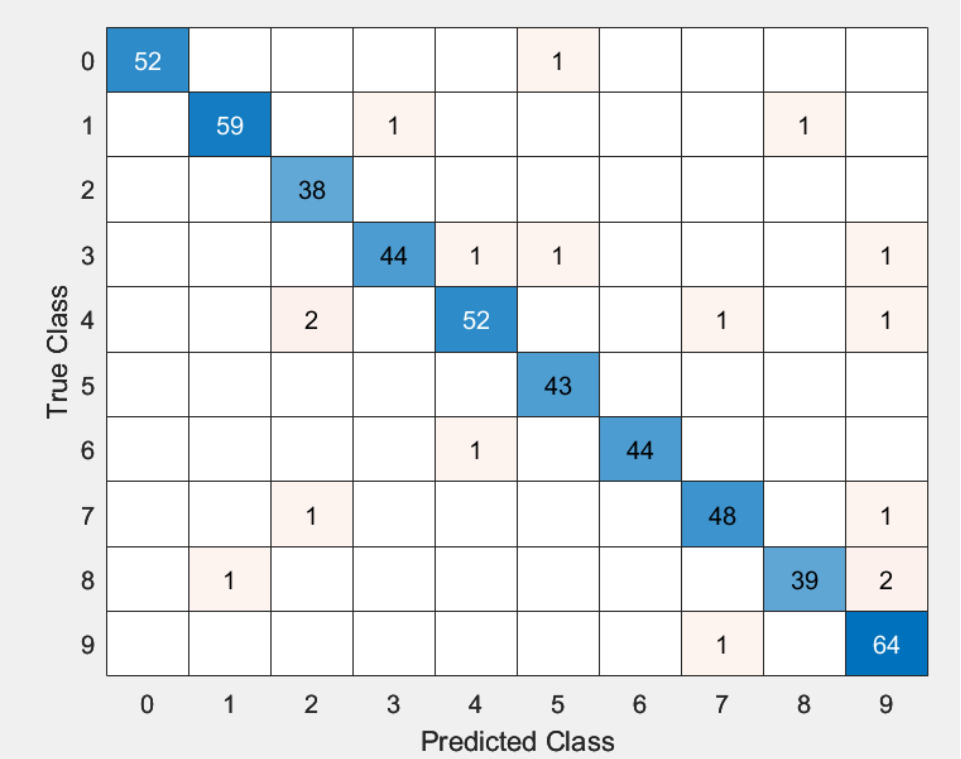
cost = 0.016835 training_percent = 1.000000
cost = 0.007583 training_percent = 1.000000
cost = 0.017550 training_percent = 1.000000
cost = 0.007484 training_percent = 1.000000
cost = 0.009981 training_percent = 1.000000
test accuracy: 0.974000

Q 3.2 Test the network - 1 Pts

Modify test_network.m to generate the confusion matrix:

```
% Modify the code to get the confusion matrix
prediction = zeros(1,size(xtest,2));
for i=1:100:size(xtest, 2)
    [output, P] = convnet_forward(params, layers, xtest(:, i:i+99));
    [probability, index_max] = max(P, [], 1);
    prediction(:, i:i+99) = index_max;
end
Compare = confusionmat(ytest, prediction);
confusionchart(Compare, [0:9])
```

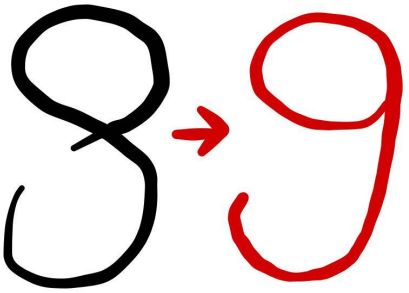
Running it to get the confusion matrix:



Comment on the top two confused pairs of classes:

The top two confused pairs are: (4, 2) and (8,9). The network wrongly predicts 4 as 2, I believe it is because the vertical strokes of 4 are too short to be ignored, and the oblique downward and horizontal strokes are misidentified because they are close to 2 like the picture 1 below; And for 8 being mistaken for 9, I think it is because the circle in the lower half of 8 is not closed, the system thinks it is the lower half of the curve of 9 like the picture 2 below.

picture 1



picture 2

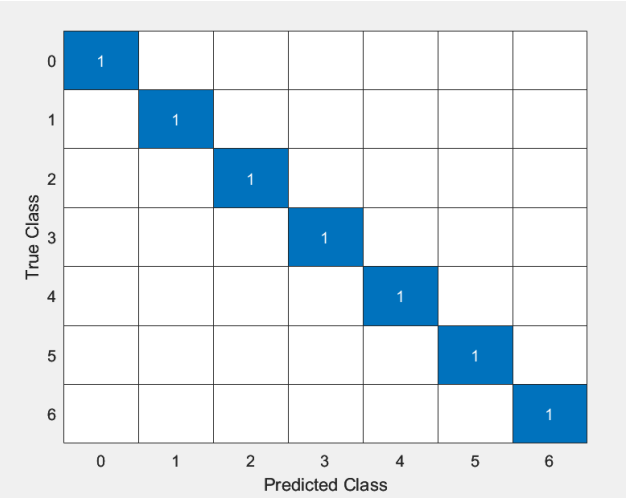
Q 3.3 Real-world testing - 1 Pts

Samples are in the `'../images/numbers/`, I scribble 7 numbers from 0-6:

The network predicted all their labels correctly, and the code for testing them I wrote was realworld_testing.m:

```
samples = zeros(784,7);
%% Loading data
for i = 0:6
   image_src = sprintf('../images/realworld_test/%d.jpg', i);
   image = rgb2gray(im2double(imread(image_src))).';
   img2col = reshape(image, [], 1);
   samples(:, i+1) = img2col;
end
% load the trained weights
load lenet.mat
%% Network defintion
layers = get_lenet();
layers{1,1}.batch_size = size(samples,2);
%% Testing the network
% Modify the code to get the confusion matrix
ytest = [0:6];
[output, P] = convnet_forward(params, layers, samples);
[probability, index_max] = max(P, [], 1);
C = confusionmat(ytest, index_max-1);
confusionchart(C, [0:6])
```

And the output confusion chart shows that network predicted all their labels correctly:



But one thing I found interesting in this part is, I only can get high accuracy when my self-scribbled pictures with white numbers on a black background. But when I change them to black numbers on the white background, the accuracy is very low and even all predictions are wrong. I think the reason is because the train data we got by `load_mnist(fullset)` is all white numbers on the black background. I choose some of the data in `xtest` and use `imshow()` to see how they look:

```
% Check how the training data picture look like
for j=1:10
   image_src_test = sprintf('../images/test/%d.jpg', j);
   some_test = xtrain(:,j);
   col2img = reshape(some_test, 28, 28);
   imwrite(col2img.', image_src_test);
end
```

The result is put into `'../images/test'` and they show below, which confirmed that my suspicion was probably right.

Part 4 Visualization

Q 4.1 - 1 Pts

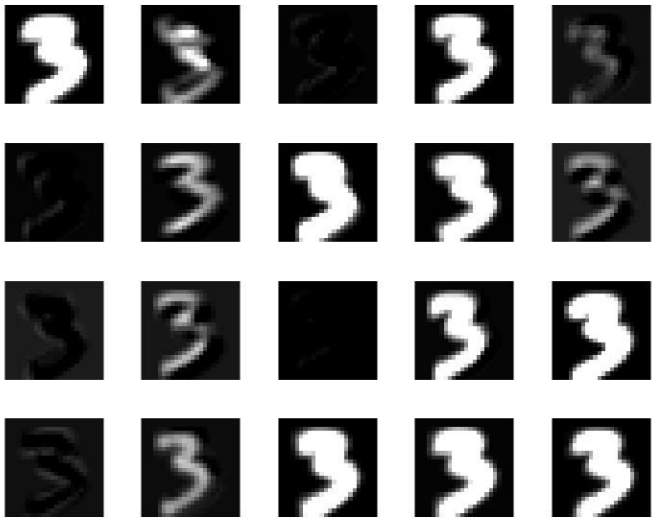
Write a script vis_data.m which can load a sample image from the data, visualize the output of the second and third layers (i.e., CONV layer and ReLU layer):

```
% Fill in your code here to plot the features.
% Figure 1
figure_data1 = reshape(output{1,2}.data,output{1,2}.height,
output{1,2}.width,output{1,2}.channel);
figure();
title('Feature maps of the second layer');
for fig = 1:20
   subplot(4,5,fig);
   h(fig) = imshow(figure_data1(:,:,fig).');
end
% Figure 2
figure_data2 = reshape(output{1,3}.data,output{1,3}.height,
output{1,3}.width,output{1,3}.channel);
figure();
title('Feature maps of the third layer');
for fig = 1:20
   subplot(4,5,fig);
   h(fig) = imshow(figure_data2(:,:,fig).');
end
```

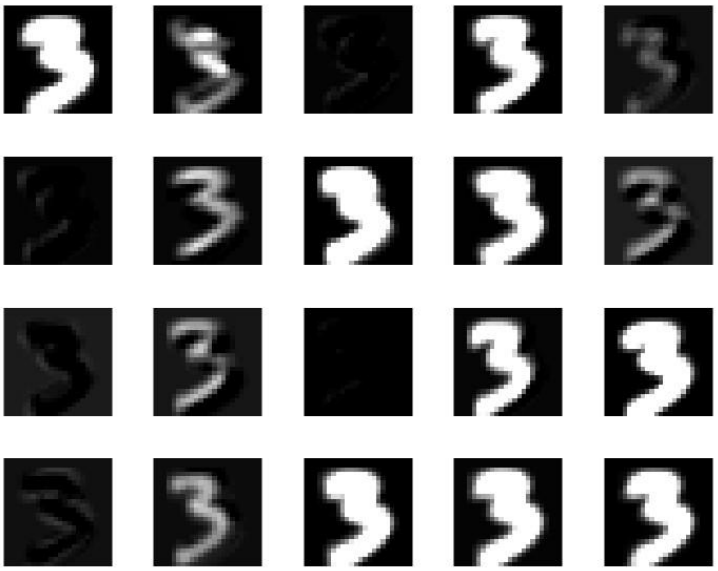
Show 20 images from each layer on a single figure file:

Input file:

Output figure of the second layer:



Output feature maps of the third layer:



The two output features are saved in `'../result'`, they look similar because the ReLU function only changes the data smaller than 0 to 0 and keeps the same with others, so the change is too small to be seen in our eyes.

Q 4.2 - 1 Pts

Compare the feature maps to the original image and explain the differences:

Compared with the original image, some of the feature maps returned from CONV and ReLU layers look more blurred and unclear, like the 1st, 4th, 7th, and 8th images. And some of them look completely dark, like the 3nd, 5th, and 6th pictures. The reason for this is that the convolution operation extracts different feature values from the input image (original image), and the first CONV layers can only extract some low-level features (such as edges, lines, and corners), so the visualized feature map looks like a blurred version of the original image.

Part 5 Image Classification - 2 Pts

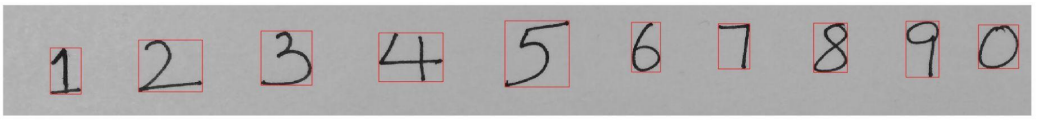
1. Classify each pixel as foreground or background pixel by performing simple operations like thresholding:

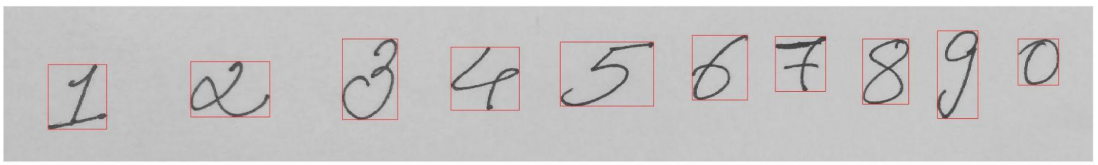
At this step, get the pixels of each image, and set different thresholds for each of them to separate them as foreground or background. Because of differences in brightness and contrast of the 4 images, they require different thresholds to distinguish foreground from background.

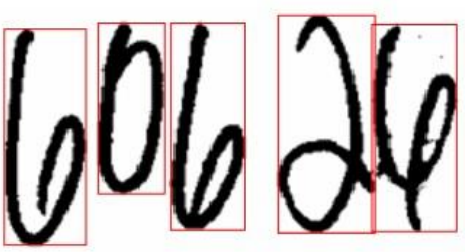
Transfer the image by im2double to make its pixels between 0 and 1, and reset the pixels which larger than threshold to pure white (1) and others to pure black (0), then get the inverse color image of the grayscale image, and use 1 to represent all stroke parts (foreground).

2. Find connected components and place a bounding box around each character:

Use `bwlabel()` to get the connected components, and use `regionprops()` to get the bounding box around each component. The results are shown below.







And they are also saved in `'../images/bound_box'`.

3. Take each bounding box, pad it if necessary and resize it to 28×28 and pass it through the network:

I first use padarray() to pad each bounding box to a square, then resize it to 28×28 using imresize(), then pass it to network and the output is:

>> ec
Testing image1.jpg, got accuracy at 9/10 (90%)
Testing image2.jpg, got accuracy at 7/10 (70%)
Testing image3.jpg, got accuracy at 5/5 (100%)
Testing image4.jpg, got accuracy at 40/50 (80%)

The code for this portion is the ec.m file in the matlab folder.