CMPT 762 X100, Fall 2024, Computer Vision

Project 1: Digit recognition with convolutional neural networks

Name: Wenhe Wang
Student Number: 301586596
Email: wwa118@sfu.ca
Discussed with: Gouttham N, gna23

1. Forward Pass

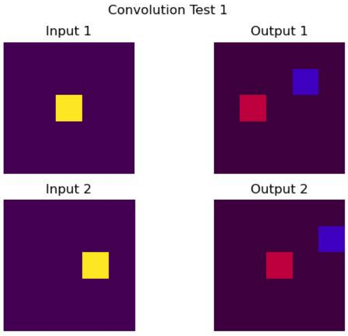
Inner Product: This part of the implementation is achieved by multiplying the transposed weight matrix with each image in the batch, followed by adding the bias term to the result.

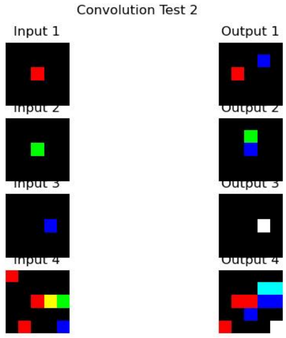
Max Pooling: The kernel moves across the pixels with the specified stride to downsample the feature maps.

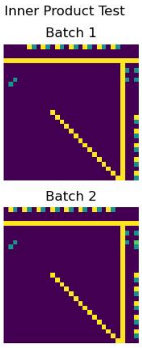
Convolution Layer: The im2col_conv_batch function converts the image in the batch into a vector of pixels. This vector is then multiplied by the weight matrix to perform the convolution. This approach is significantly faster since matrix multiplication is highly optimized, leading to faster training times.

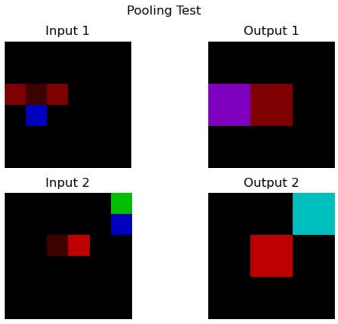
ReLU Layer: In this layer, each element in the input matrix is passed through the ReLU function, which sets any negative value to zero, i.e., element = max(element, 0).

test_components.py results:









2. Back Propagation

For Output, we have:

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

For the gradients w.r.t. the activation function, we have:

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

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

For Relu, we have:

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

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

And the gradient for Relu is 1 if x > 0, else 0.

For Innet Product layer, we have:

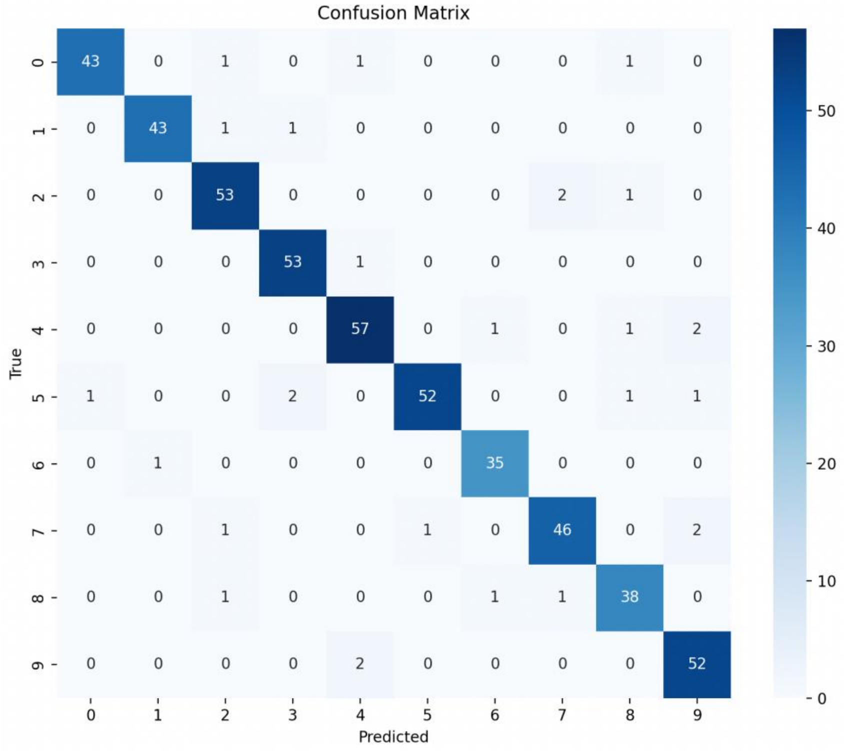
$$h_i = wx + b$$
$$\frac{dl}{dw} = \frac{dl}{d_i}\frac{dh_i}{dw_i} = output.diff \times input.data$$
$$\frac{dl}{dh_{i-1}} = \frac{dl}{dh_i}\frac{dh_i}{dh_{i-1}} = output.diff \times weights$$
$$\frac{dl}{db} = \frac{dl}{dh_i}\frac{dh_i}{db} = output.diff \times 1$$

3. Training

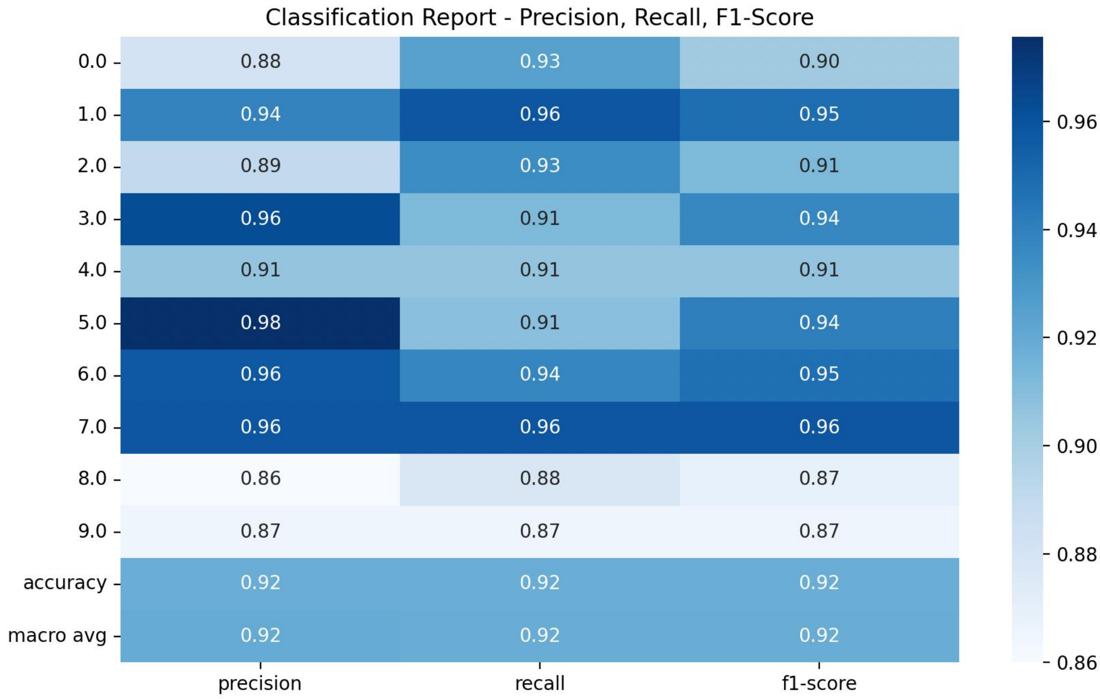
Training process:

```
(cv_proj1) wenhe@Wenhes-MacBook-Air-2 python % python train_lenet.py
cost = 2.305296955167195 training_percent = 0.12
iter: 0
test accuracy: 0.104
cost = 0.8678104903829282 training_percent = 0.74
cost = 0.3737822689966489 training_percent = 0.9
cost = 0.2992355040308167 training_percent = 0.89
cost = 0.19080101320868784 training_percent = 0.93
cost = 0.20644959880388997 training_percent = 0.94
cost = 0.1316775490306894 training_percent = 0.95
cost = 0.10750126821783451 training_percent = 0.97
cost = 0.025157921344850273 training_percent = 1.0
cost = 0.048283259153948 training_percent = 0.99
cost = 0.14003165702098508 training_percent = 0.96
iter: 500
test accuracy: 0.916
cost = 0.05409842806003138 training_percent = 0.97
cost = 0.08970770217055925 training_percent = 0.98
cost = 0.021029781818570112 training_percent = 0.99
cost = 0.025518487991197762 training_percent = 1.0
cost = 0.026006208744183693 training_percent = 0.99
cost = 0.004594530733134321 training_percent = 1.0
cost = 0.009770060048904973 training_percent = 1.0
cost = 0.009373706464501454 training_percent = 1.0
cost = 0.01497565527379685 training_percent = 1.0
cost = 0.007278696780447309 training_percent = 1.0
iter: 1000
test accuracy: 0.916
cost = 0.002224787810899817 training_percent = 1.0
cost = 0.00541639649158786 training_percent = 1.0
cost = 0.0033392156754192036 training_percent = 1.0
cost = 0.00229068818037975 training_percent = 1.0
cost = 0.003066818222135854 training_percent = 1.0
cost = 0.003059440421472122 training_percent = 1.0
cost = 0.0011870536527473195 training_percent = 1.0
cost = 0.002657053925608126 training_percent = 1.0
cost = 0.0015115170601482748 training_percent = 1.0
cost = 0.003336577131047255 training_percent = 1.0
iter: 1500
test accuracy: 0.924
cost = 0.0011464161156371734 training_percent = 1.0
cost = 0.0026870973761157226 training_percent = 1.0
cost = 0.001534198974025787 training_percent = 1.0
cost = 0.0012573842993870692 training_percent = 1.0
cost = 0.003312213587797182 training_percent = 1.0
cost = 0.0014009809429186887 training_percent = 1.0
cost = 0.002213974685958105 training_percent = 1.0
cost = 0.0024475123755748583 training_percent = 1.0
cost = 0.0013070971928505722 training_percent = 1.0
```

Confusion matrix:



Classification report:



```
(cv_proj1) wenhe@Wenhes-MacBook-Air-2 python % python test_network.py
              precision    recall  f1-score   support

           0       0.88      0.93      0.90        40
           1       0.94      0.96      0.95        48
           2       0.89      0.93      0.91        61
           3       0.96      0.91      0.94        57
           4       0.91      0.91      0.91        53
           5       0.98      0.91      0.94        44
           6       0.96      0.94      0.95        48
           7       0.96      0.96      0.96        48
           8       0.86      0.88      0.87        49
           9       0.87      0.87      0.87        52

    accuracy                           0.92       500
   macro avg       0.92      0.92      0.92       500
weighted avg       0.92      0.92      0.92       500
```

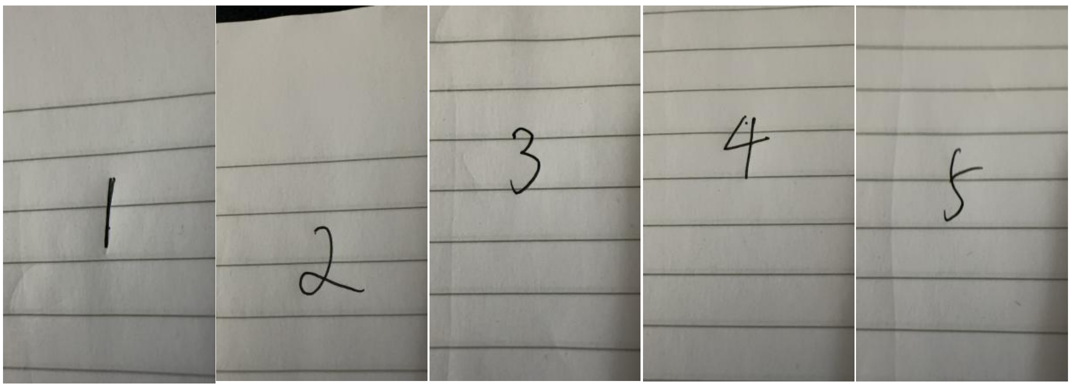
Top two confused pairs of classes:

```
(cv_proj1) wenhe@Wenhes-MacBook-Air-2 python % python test_network.py
Top two confused pairs: [(9, 4), (4, 9)]
```

I think the confusion between classes 4 and 9 in my trained model likely occurs due to the visual similarities between these 2 digits in the training set. A handwritten '4' can sometimes look like a '9' depending on how the stroke is drawn. If the lower part of '4' is curved or incomplete, the model may confuse it with '9', which also has a circular component.

The convolutional layers in my model extract features such as edges and curves. If the model hasn’t learned to distinguish subtle differences between '4' and '9', it could lead to confusion in these specific cases.

Real world testing:



Predictions:

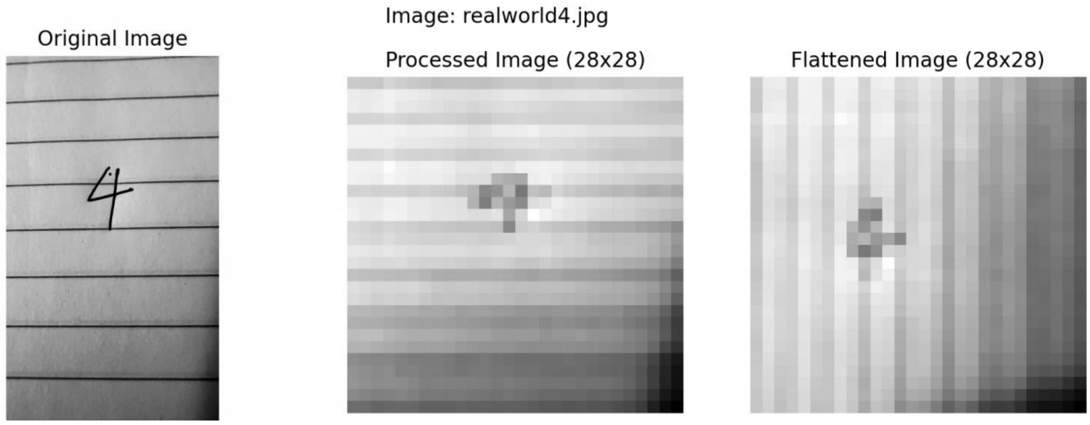


Image: realworld4.jpg
Original Image
Processed Image (28x28)
Flattened Image (28x28)

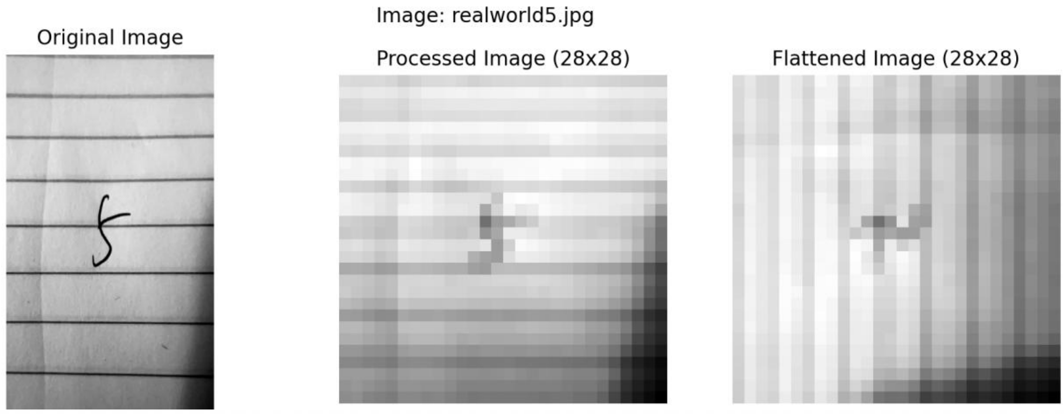


Image: realworld5.jpg
Original Image
Processed Image (28x28)
Flattened Image (28x28)

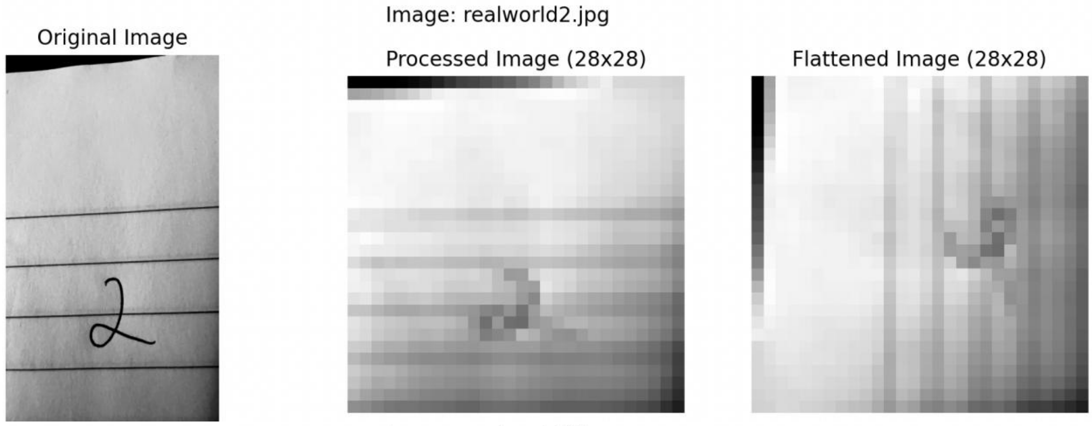


Image: realworld2.jpg
Original Image
Processed Image (28x28)
Flattened Image (28x28)

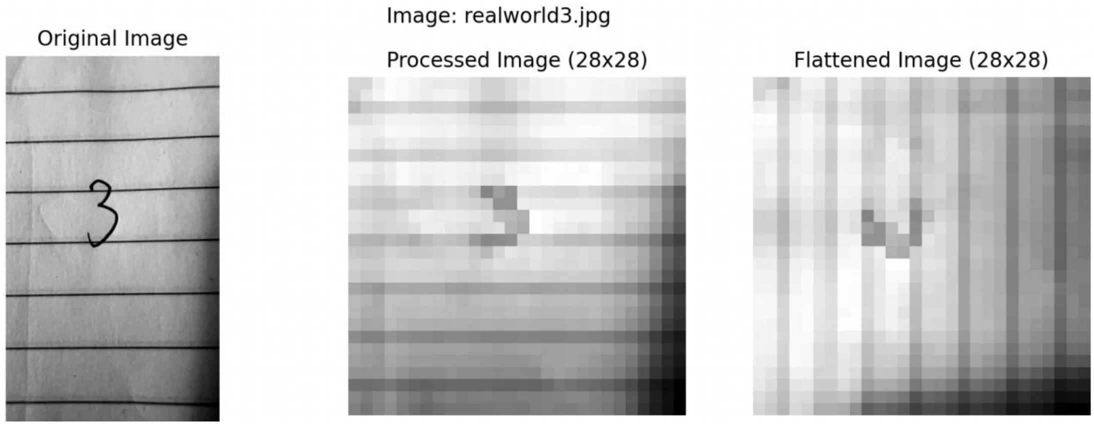
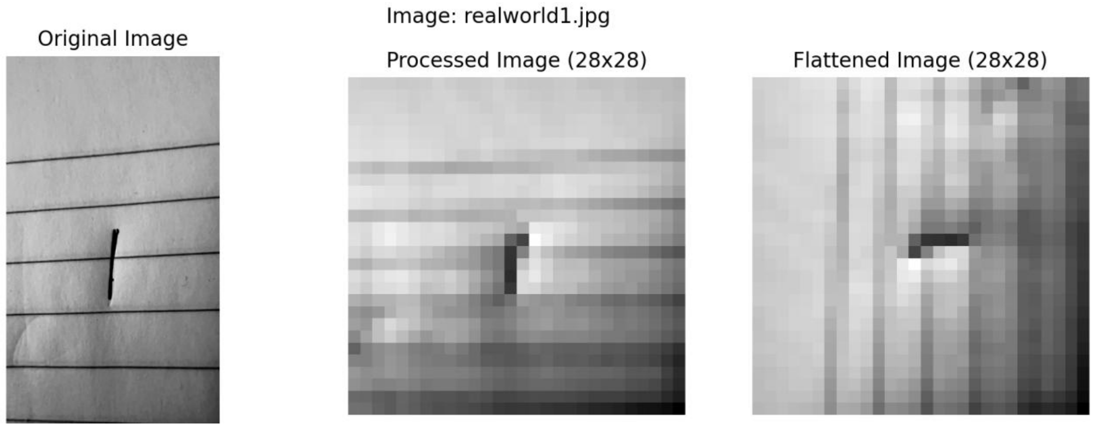


Image: realworld3.jpg
Original Image
Processed Image (28x28)
Flattened Image (28x28)

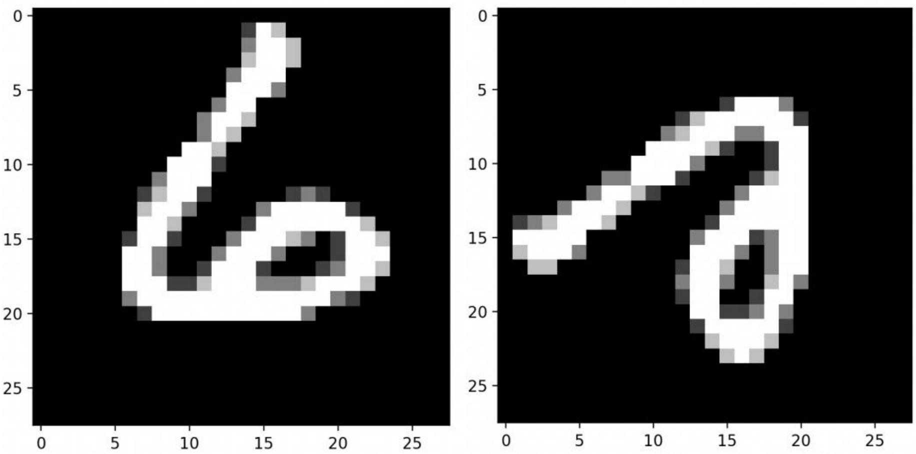


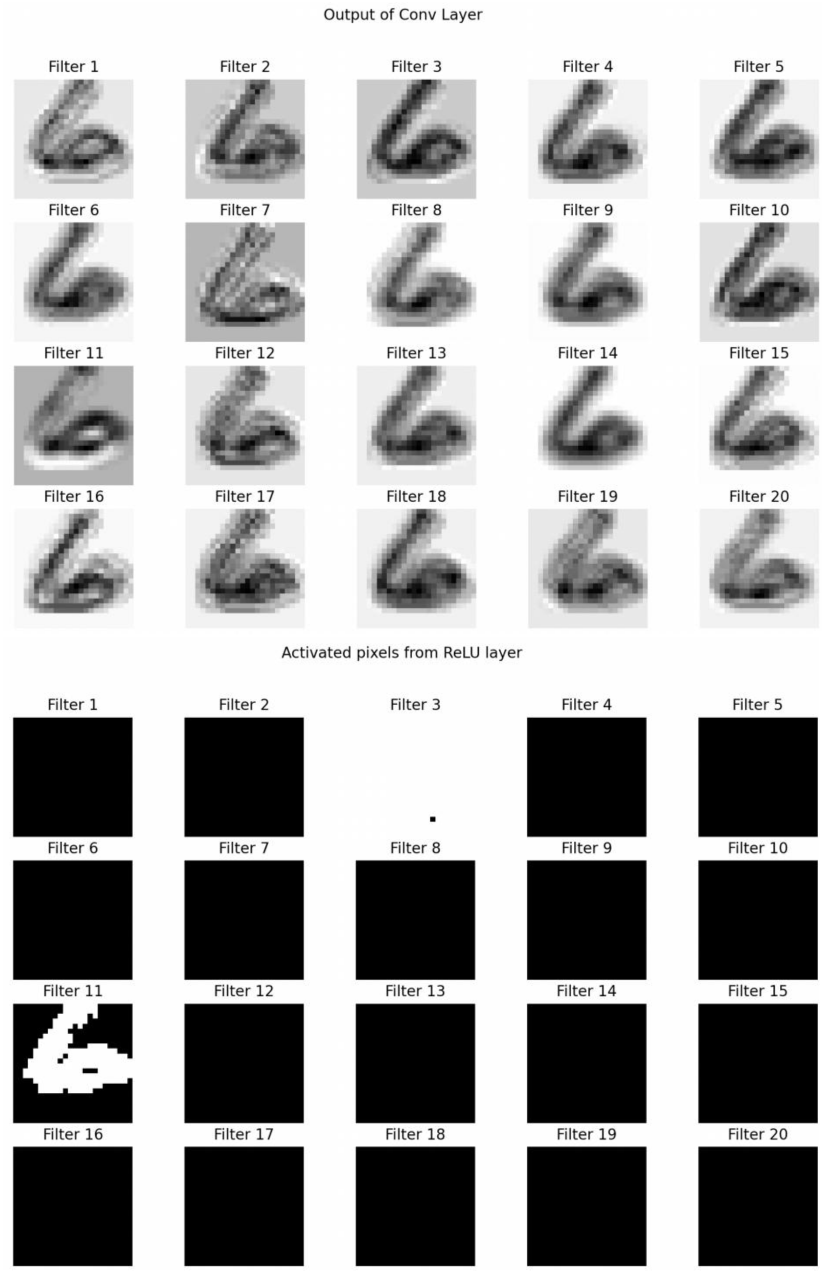

```
(cv_proj1) wenhe@Wenhes-MacBook-Air-2 python % python test_real_world.py
Press Enter to continue to the next image...
Press Enter to continue to the next image...
Press Enter to continue to the next image...
Press Enter to continue to the next image...
Press Enter to continue to the next image...
Image: realworld4.jpg, Predicted Label: 1
Image: realworld5.jpg, Predicted Label: 1
Image: realworld2.jpg, Predicted Label: 1
Image: realworld3.jpg, Predicted Label: 1
Image: realworld1.jpg, Predicted Label: 1
```

The network predicted all the labels as “1”, one reason could be that our model is trained on 28*28 images, but our 5 real world examples are all 318 * 1024. After manually conpressing these images with the resize() function, the quality of the digits are heavily reduced. Also, our background for the examples are not aligned with our training data, which has a lot of “pass-through” clear lines, becasue these were taken from my practise paper which leads to more “noises” to small resolution images trained model like lenet.

4. Visualization

Input image:





From the second Conv layer output, we can see that the network has learned how to find the edges and the outline of the digit (the dark area in the middle of the image), but some filters are weaker at finding digits even with this clear “6” example (their visuals are more blurry in general).

From the third Relu layer output, we can see that only one filter is activated and showing the affected pixels correctly, which means our model may not robbust enough to unseen data based on current observation, and it seems heavily rely on few filters.

5. OCR

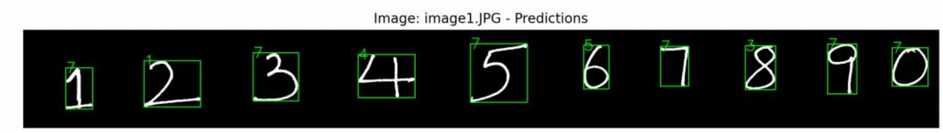


Image: image1.JPG - Predictions

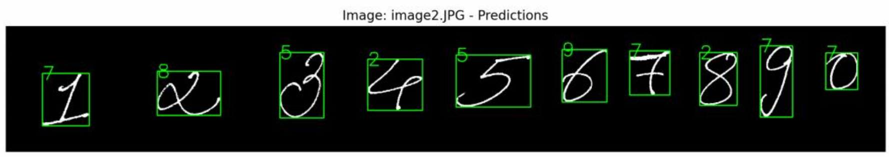


Image: image2.JPG - Predictions

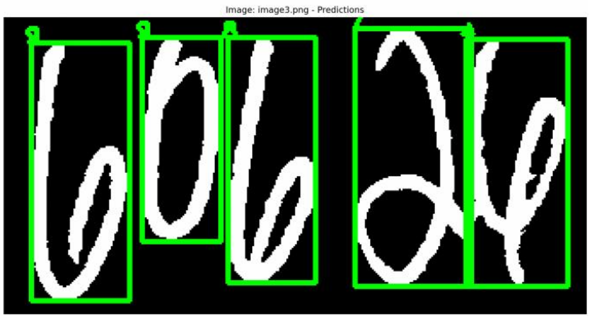


Image: image3.png - Predictions

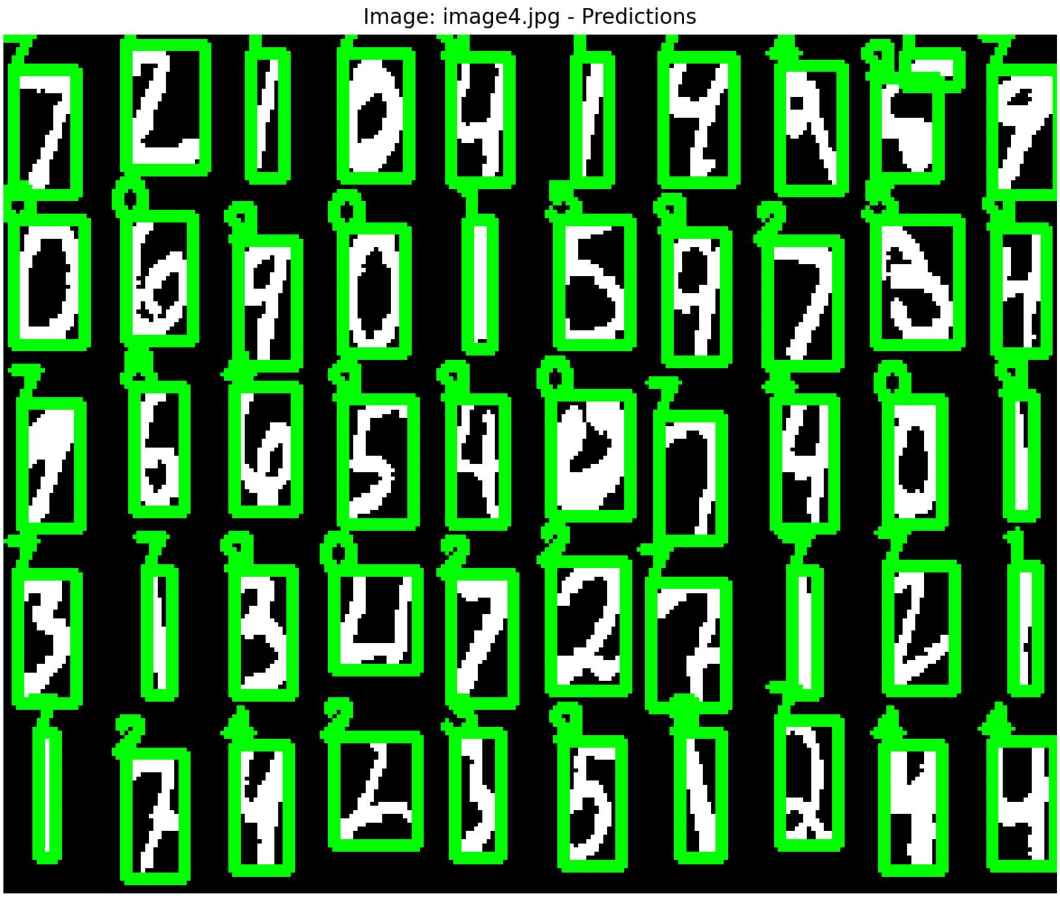


Image: image4.jpg - Predictions