Project 1: Digit Recognition With Convolutional Neural Networks

1. Part1 : Forward pass

   1) Inner Product Layer

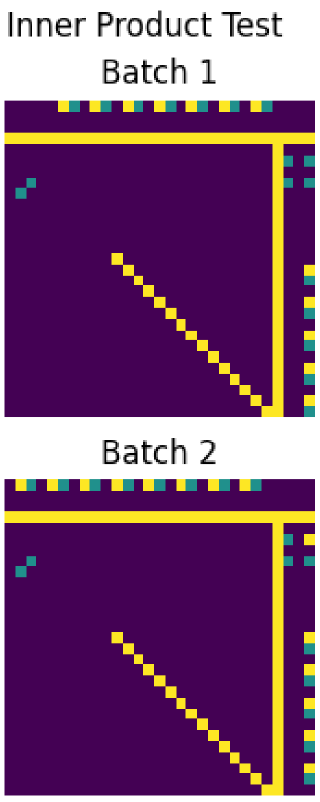


Figure 1

The implementation of Inner Product Layer forward pass gets the transpose of the weight matrix and adds the bias vector. The backward pass is obtained by the chain rule, which gets the gradient for weights via outer product of input data and error differentials, the gradient for bias via the sum of error differentials. The error differential for input data is obtained by multiplying the weight matrix with error differentials.

   2) Pooling Layer

   Specifically implemented the max_pooling in the forward pass, where the maximum value is taken over a certain window of the feature map.

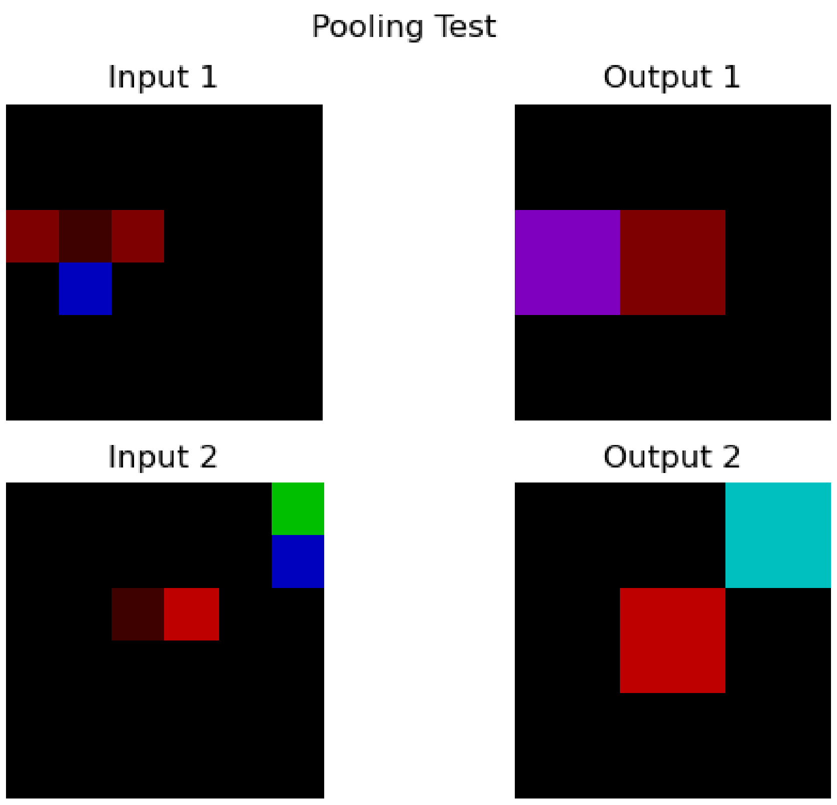


Figure 2

3) Convolution Layer

In the forward pass, I used the im2col approach to trade off increased memory usage for faster computation. Reshape the convolutional filters into a 2D matrix (filters) with one row for each filter. Each row will be the flattened version of the 3D filter (height, width, channel).

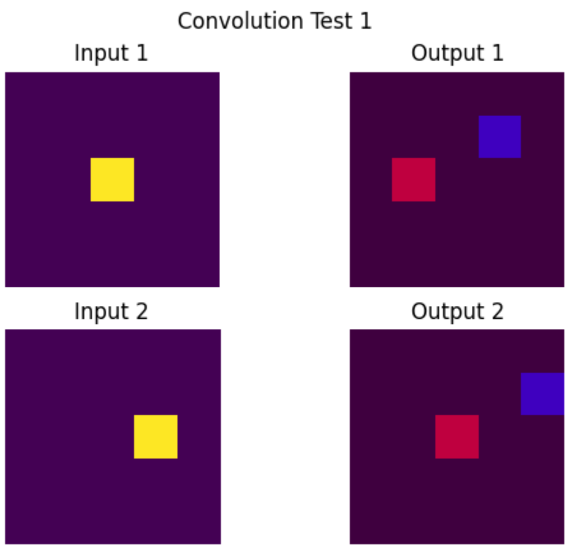


Figure 3

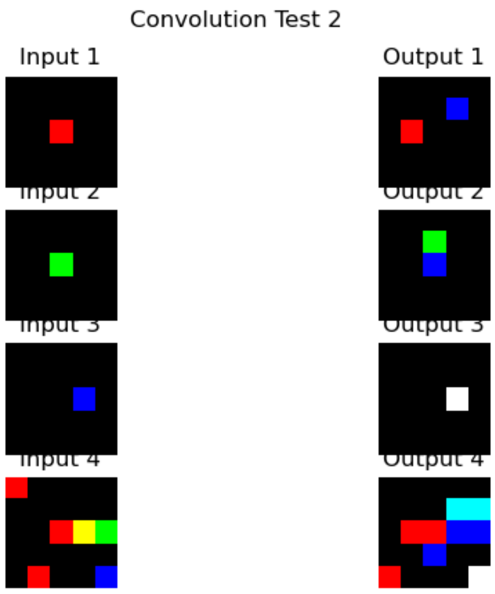


Figure 4

4) ReLU

Basically implemented the ReLU with its derivative approach, which is 1 for input greater than 0 and 0 for otherwise values.

2. Part2 : Back propagation

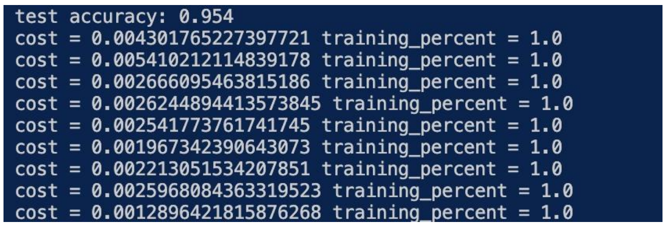
1) ReLU

At the backward pass of the ReLU, I implemented the gradient propagation, which is multiplying the incoming gradient with respect to the output of the ReLU, with the gradient of the loss with respect to the input.

2) The conv_net.py

3. Part3: Training

1) The accuracy of the training is around 0.954, as shown in Figure 5.



```
test accuracy: 0.954
cost = 0.004301765227397721 training_percent = 1.0
cost = 0.005410212114839178 training_percent = 1.0
cost = 0.002666095463815186 training_percent = 1.0
cost = 0.0026244894413573845 training_percent = 1.0
cost = 0.002541773761741745 training_percent = 1.0
cost = 0.001967342390643073 training_percent = 1.0
cost = 0.002213051534207851 training_percent = 1.0
cost = 0.0025968084363319523 training_percent = 1.0
cost = 0.0012896421815876268 training_percent = 1.0
```

Figure 5

2) Test the network

Interpretation of the matrix:

As shown in Figure 6, the diagonal represents the number of correct predictions for each class. Specifically, for class '0', 43 instances were correctly classified, for class '1', 61 instances were correctly classified, and so on.

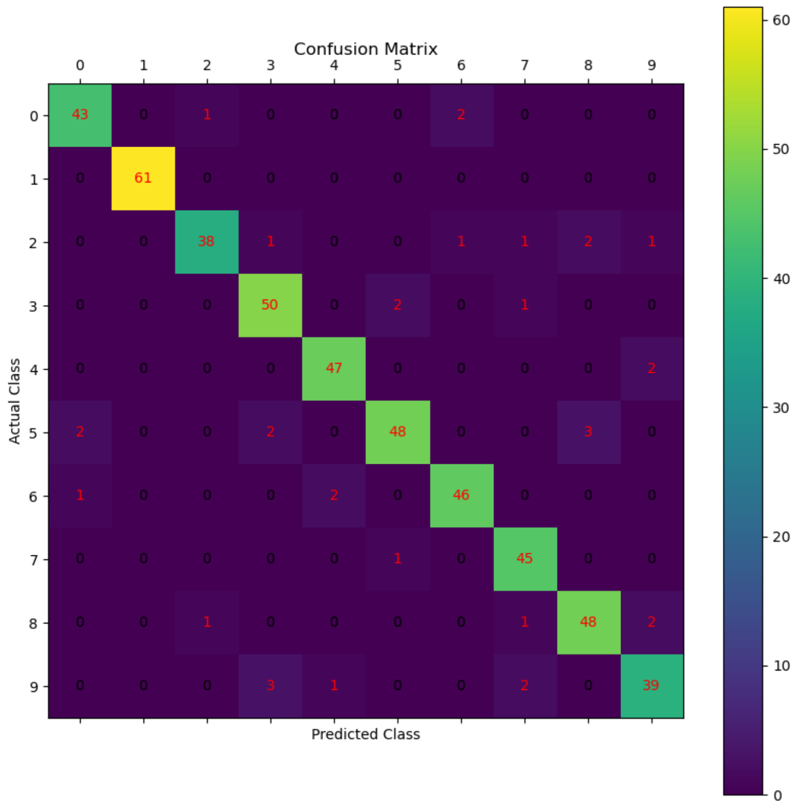


Figure 6

After normalizing the confusion matrix, we can get:

The most confused pair is (9, 3) with a normalized total confusion of 0.0667.
The second most confused pair is (5, 8) with a normalized total confusion of 0.0545.

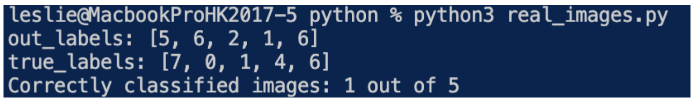
- ❖ The confusion between '9' and '3' can be due to the structural similarities in the way we write these numbers. Both '9' and '3' can have a rounded structure at the top. Especially in hand-written digits, if the bottom loop of '9' isn't very pronounced, it can look a lot like '3'. This can be a potential reason for the model mistaking '9' for '3'.

- ❖ ‘5’ and ‘8’ both have curves. While '5' has a semicircle on its bottom part, '8' comprises two circles on top of each other. If the top line of '5' is not clearly discernible or if it is written with a curve, it might closely resemble an '8', leading to potential misclassification.

- ➢ Ambiguous Handwriting: In hand-written digit datasets, people might write numbers in various styles and levels of clarity. For instance, a hastily written '5' could look like an '8', especially if the gap between its two halves isn't clear.
- ➢ Overlapping Features: In a CNN, certain filters might activate for rounded shapes. Given that both '5' and '8' possess rounded features, some layers might produce similar activations for both numbers, leading to confusion in the deeper layers or in the classification layer of the model.

3) Real-world Testing

I have collected several real world images, and randomly selected 5 images for testing at each time. Each image includes one digit on it.
After label mapping, images are then converted to grayscale to align with the network's expectations. The pixel intensities are normalized to the [0, 1] range, ensuring consistency with the training dataset. Moreover, images are resized to a resolution of 28x28 pixels, matching the input size the CNN has been trained on. The images are further transposed and flattened.



```
leslie@MacbookProHK2017-5 python % python3 real_images.py
out_labels: [5, 6, 2, 1, 6]
true_labels: [7, 0, 1, 4, 6]
Correctly classified images: 1 out of 5
```

Figure 7

The average correctly classified images is 1 out of 5, as shown in the print statement in Figure 7.

4. Part4: Visualization

Sample Image

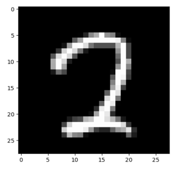


Figure 8

Conv Layer *ReLU Layer*

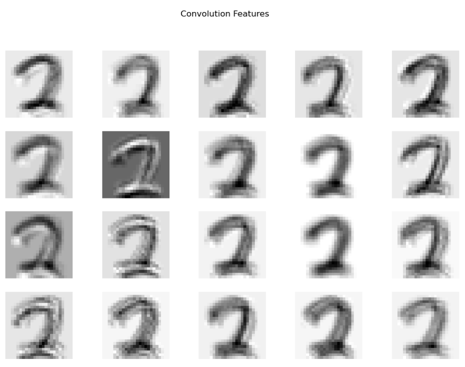


Figure 9

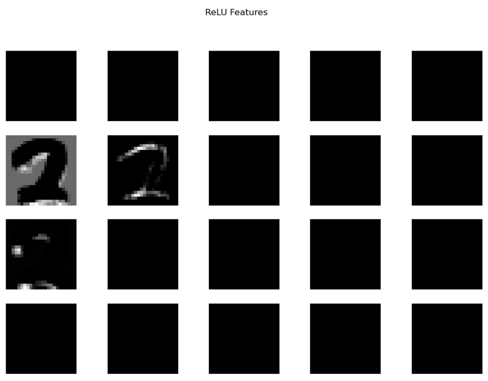


Figure 10

Comparison:

Convolution Layer :

Differences:

Some feature maps highlight particular regions of the image. This indicates that the corresponding filters are sensitive to features present in those regions. Some feature maps that highlight the background or appear less clear might correspond to filters that didn't converge to any significant feature during training or are identifying broader patterns/regions rather than specific details.

Explanations:

Convolutional layers are responsible for scanning an input image with a set of filters (or kernels) to detect specific features, such as edges, corners, and textures. Convolution layers aim to highlight specific features. Compared with their outputs to the original image, certain parts of the image (like edges, corners, or specific patterns) are enhanced or made more visible. The output feature maps of a convolution layer can show various aspects of the image, depending on the learned filters. Some filters might detect vertical edges, while others might detect horizontal edges or specific textures.

ReLU Layer:

Differences:

There are three visible images having dim edges and corners, which indicate only those feature maps had positive values at those positions, suggesting those particular filters detected meaningful features in the input image. Many of the feature maps after the ReLU activation are entirely black, which suggests that a lot of the values in the convolution layer's output were negative for that particular input image. Only the positive values pass through unchanged, while the negatives are set to zero (black in the visualization).

Explanations:
The primary purpose of the ReLU activation function is to introduce non-linearity into the model. By setting all negative values to zero, it ensures that the model can learn from error and make adjustments, which is essential for learning complex patterns. ReLU layers don't necessarily give us insight into the image's content, but more into the network's behavior. If many feature maps after ReLU are black, it could mean that the network has invalidated many convolutional layer outputs of the image, which means some outputs are not necessary or useless for a given task.

Potential Reasons & Improvements:

As we can see, the performance of training is not satisfactory. I have listed several potential reasons and future improvements for my training.

Normalization: If the data is not properly normalized, the convolutions might produce largely negative values, resulting in mostly black images post-ReLU. Therefore, it might be helpful if we can ensure that the input data is preprocessed correctly, such as being normalized to a certain range. What's more, batch normalization after the convolution and before activation can sometimes help in making the activations more balanced across the network.

Choice of Parameters: The choice of parameters like the kernel size, number of filters, stride, etc., in your convolutional layer affects the kind of features that are learned and hence the visualizations that you observe.

Network Architecture: The architecture of the network might not be conducive to learning meaningful hierarchical representations of the data, leading to poor feature visualizations.

For further improvements, we might consider retraining the model with a different initialization method or a different architecture to get a more diverse set of feature maps after ReLU.

5. Part5: Image Classification

Approach:

- Pre-processing and Binarization:

Each image was converted to greyscale to reduce computational complexity.
Depending on the content of the images, I used two distinct methods for binarization:

For images image1 and image2, I applied Adaptive Gaussian Thresholding. This method dynamically calculates the threshold for small regions of the image, which yields different thresholds for different regions and accounts for variations in lighting conditions across the image.
For image3, I used a manual threshold of 110 after noticing this value gives better segmentation for the digits in the image.
For image4, a manual threshold of 40 was applied. The choice was due to the particular characteristics of this image.

- Digit Segmentation:

To segment the handwritten digits, I used the connectedComponentsWithStats function from OpenCV. This function identifies each connected component in the image (i.e., each digit).
Each detected component was then enclosed with a bounding box, as suggested in the project instruction.

- Digit Pre-processing for Neural Network:

Every segmented digit was resized to a 28x28 resolution to match the input size expected by the neural network. To ensure the aspect ratio was maintained, padding was applied as necessary.
The images were then normalized, ensuring pixel values were in the range [0, 1].

- Digit Classification:

The pre-processed digit image was passed through the neural network to obtain the predicted label.
The prediction was then annotated on the original image for visual validation.

- Grid Search for Optimal Preprocessing:

To further fine-tune the preprocessing parameters for optimum image before feeding them into the network, I have been trying to get better parameters with iterative grid search, which includes dynamic image thresholding, dilation kernel adjustment, gaussian blur configurations, and also accuracy metrics & parameter retention.

Results:

For each image, I plotted the thresholded image and the output image with the predicted label on each digit.

For image1.JPG:

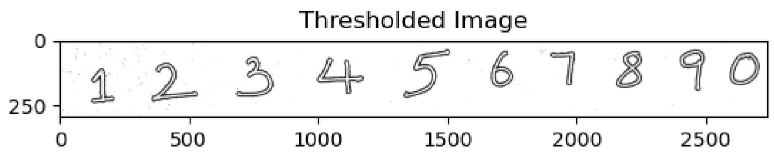


Figure 11

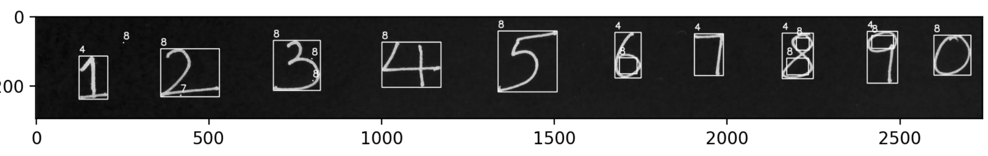


Figure 12

The accuracy for image1 is 20.00%.

For image2.JPG

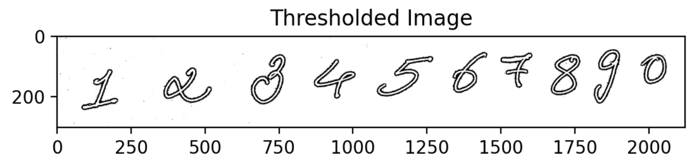


Figure 13

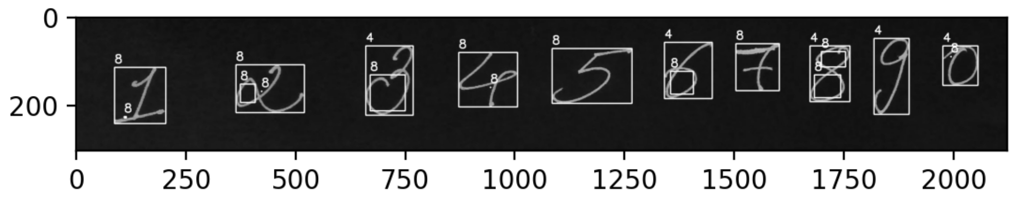


Figure 14

The accuracy for image2 is 10.00%.

For image3.png:

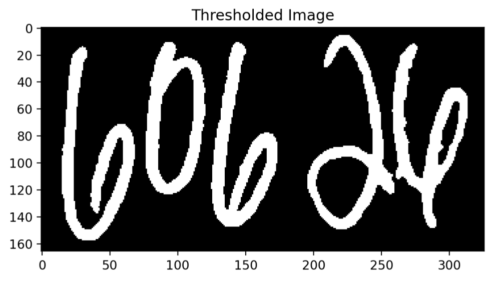


Figure 15

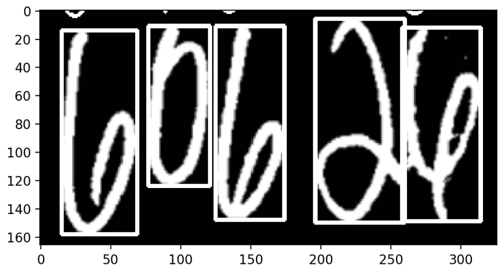


Figure 16

The accuracy for image3 is 20.00%.

For image4.jpg:

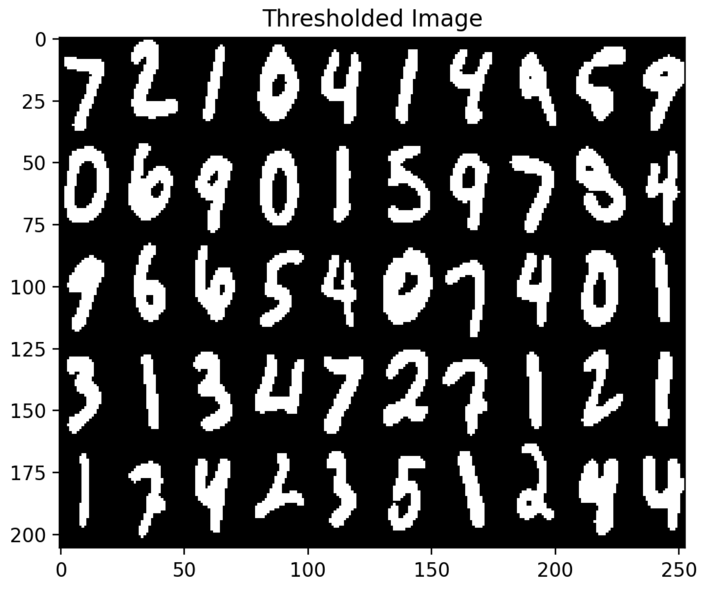


Figure 17

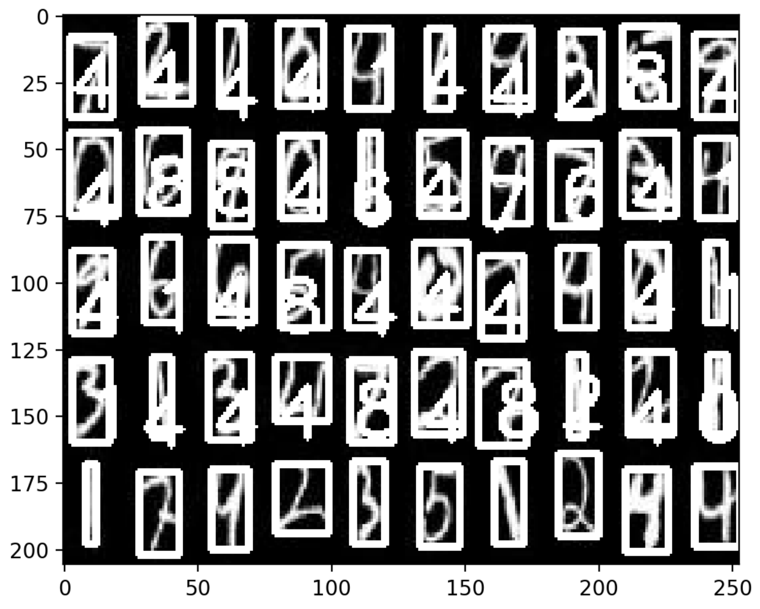
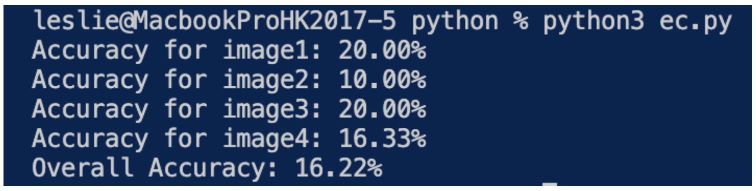


Figure 18

The accuracy for image4 is 16.33%.

Using the above approach, the following accuracies were observed:



```
leslie@MacbookProHK2017-5 python % python3 ec.py
Accuracy for image1: 20.00%
Accuracy for image2: 10.00%
Accuracy for image3: 20.00%
Accuracy for image4: 16.33%
Overall Accuracy: 16.22%
```

Figure 19

Overall Accuracy: 16.22% (Based on the total digits across all images).

Conclusion

In this endeavor, we embarked on a journey to understand and harness the potential of Convolutional Neural Networks (CNNs) for digit recognition.

Our forward pass methods showcased the inner workings of layers that define a CNN. After understanding and implementing the inner products, max pooling, convolution, I obtained the foundational concept of CNN. One critical takeaway was recognizing the indispensable role of activation functions. These functions, by adding non-linearity, give our network the ability to capture complex patterns. This made me realize that how we process and extract features from different layers has a significant impact, especially when it comes to what the activation layers pick up. Furthermore, a successful model training may also include data normalization, parameter tunings such as changing filter size and step size, and also architecture review with trying different setups.

The neural network showed a promising performance in recognizing handwritten digits from real-world images. What's more, fine-tuning, especially in pre-processing stages like thresholding or digit extraction, is a must if we want to apply our model to real-world images effectively.

Time constraints and current model limitations did pose challenges. In the future, I will be delving deeper into more advanced techniques, like transfer learning, to enhance our model's accuracy further. I also aim to explore other neural network architectures and expand the dataset, considering variations in handwriting and lighting conditions. I believe with iterative improvements and continuous learning, our model can achieve even greater heights in digit recognition.