Project 5: Recognition using Deep Networks

Author: Manushi
Date: March 21, 2024

1. Project Overview:

The project, "Recognition using Deep Networks," is a comprehensive exploration into the construction, training, analysis, and adaptation of deep neural networks for recognition tasks, with a specific focus on digit recognition using the MNIST dataset. This journey through deep learning begins with the foundational task of building and training a convolutional neural network (CNN) capable of recognizing handwritten digits. Leveraging the Python-based library PyTorch and its accompanying torchvision package, the project delves into the practical aspects of implementing a CNN, from data handling and model architecture design to training methodologies and performance evaluation.

Subsequent tasks are designed to deepen the understanding of how these networks function and how they can be optimized and repurposed. This includes an examination of the network's internal workings, such as the analysis of convolutional filters and their effects on input images. Transfer learning is then explored through the adaptation of the digit recognition network to classify Greek letters, demonstrating the flexibility and reusability of pretrained models. The project culminates in a personalized experiment where various architectural and training parameters are systematically varied to optimize network performance.

2. Build and train a network to recognize digits:

A. Get the MNIST digit data set:

The initial step of our project on recognition using deep networks involves acquiring and visualizing the MNIST digit dataset. The MNIST dataset, a cornerstone in the field of machine learning, comprises a vast collection of handwritten digits that are commonly used for training various image processing systems. This dataset includes 60,000 training images and 10,000 test images, each of 28x28 pixels in grayscale, depicting digits from 0 to 9.

The visualization of the first six digits of the MNIST test set showcases the dataset's diversity and complexity. Each handwritten digit is unique, reflecting variances in style, proportion, and orientation that a neural network must learn to recognize accurately. This initial exploration sets the stage for the subsequent development and training of a deep learning model capable of distinguishing these digits with high precision.

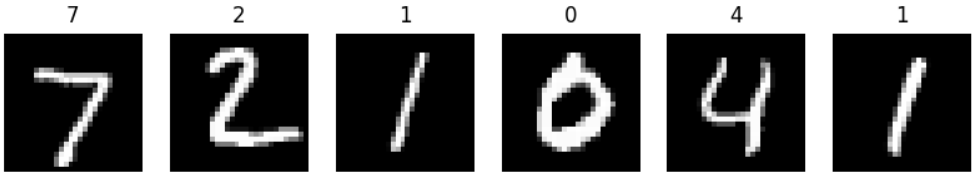

Figure 1: Visualization of the First Six Digits from the MNIST Test Dataset.

B. Build a network model:

The objective of this phase is to create a neural network model adept at the precise identification of handwritten digits from the MNIST dataset. To accomplish this, I design a multi-layered architecture within the network class that includes convolutional layers tasked with feature extraction, pooling layers that establish a spatial hierarchy, a dropout layer for regularization to prevent overfitting, and fully connected layers for the ultimate task of classification.

Figure 2 describes the complexity and structured arrangement of the neural network's architecture. Starting with an input layer that takes 28x28 pixel grayscale images, the model progresses to a convolutional layer outfitted with 10 filters, succeeded by a max pooling layer and ReLU activation function that together aid in the initial detection of rudimentary patterns. Subsequently, the architecture extends to a second convolutional layer, comprising 20 filters, integrated with a dropout layer, followed by another max pooling layer and ReLU activation function to further refine the feature maps. Post these layers, the network implements a flattening operation, transitioning the output to engage with two dense layers. The penultimate layer manifests 50 nodes, activated by ReLU, while the ultimate layer presents 10 nodes, each corresponding to a digit class, with a log_softmax function applied to output the classification probabilities. In the diagram, each cluster of nodes represents multiple units or filters within the layer, with dots signifying the continuity of these clusters. This illustrative flowchart serves as a visual guide to the network's operational mechanics and is an integral component of understanding the model's functionality.

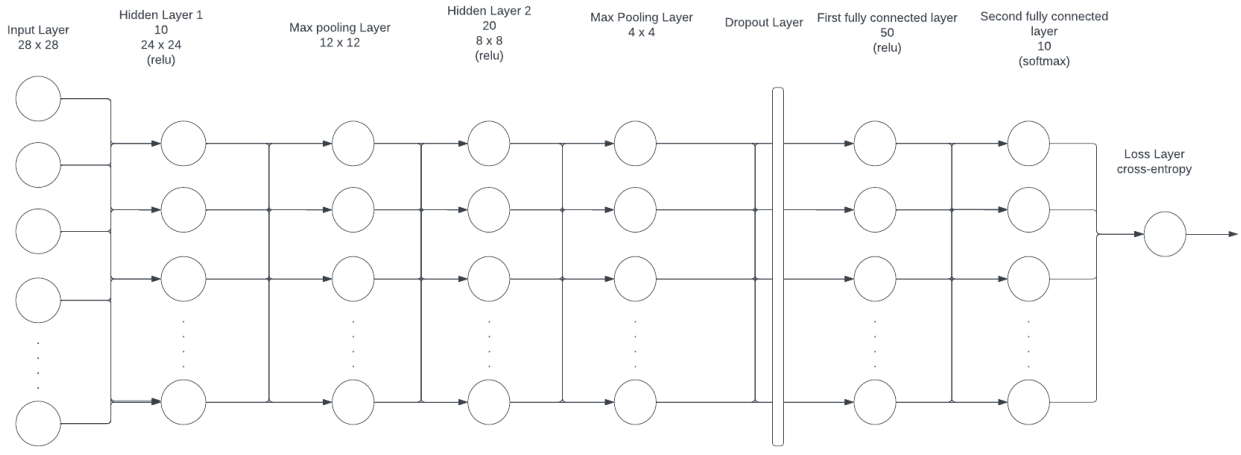

Figure 2: Architectural Diagram of the CNN for MNIST Digit Classification

C. Train the model:

During this phase, the MNIST dataset was used to train the built-in neural network. Training was conducted over five epochs, with each epoch representing a complete pass through the training data. The model's performance was evaluated at the end of each epoch, with a careful accounting of both training and testing accuracy and loss. The batch size was selected to balance computational efficiency and learning granularity.

The training process revealed a consistent improvement in performance, as demonstrated by the decreasing trend in both training and test loss across successive epochs. Accuracy metrics followed a corresponding upward trend, indicative of the model's increasing proficiency in digit classification. Notably, the training accuracy increased from 89.63% to 97.61%, while the test accuracy peaked at 98.88% at the fourth epoch, evidencing the model's robust generalization capabilities. These results were visualized in two distinct plots, one for accuracy and another for loss, employing contrasting colors to clearly delineate training from testing metrics. The accuracy plot is included in this report to illustrate the model's learning trajectory.

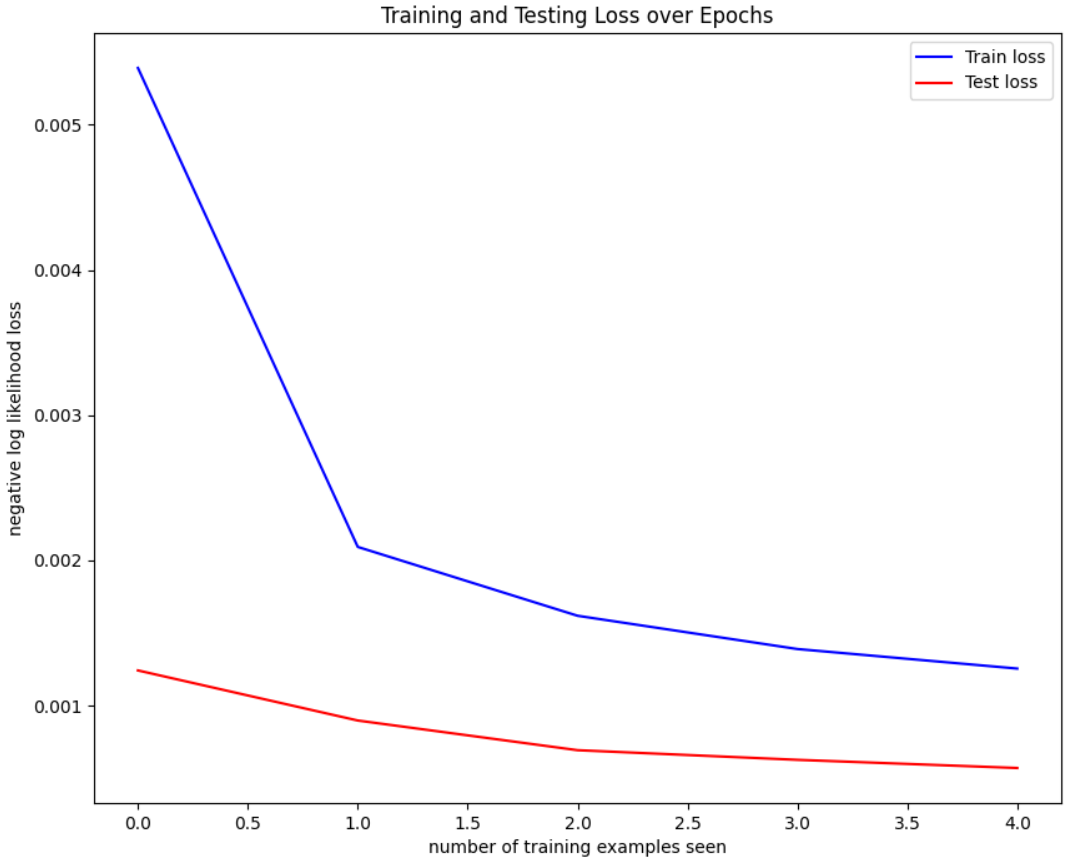

Figure 3: Training and Testing Loss Graph

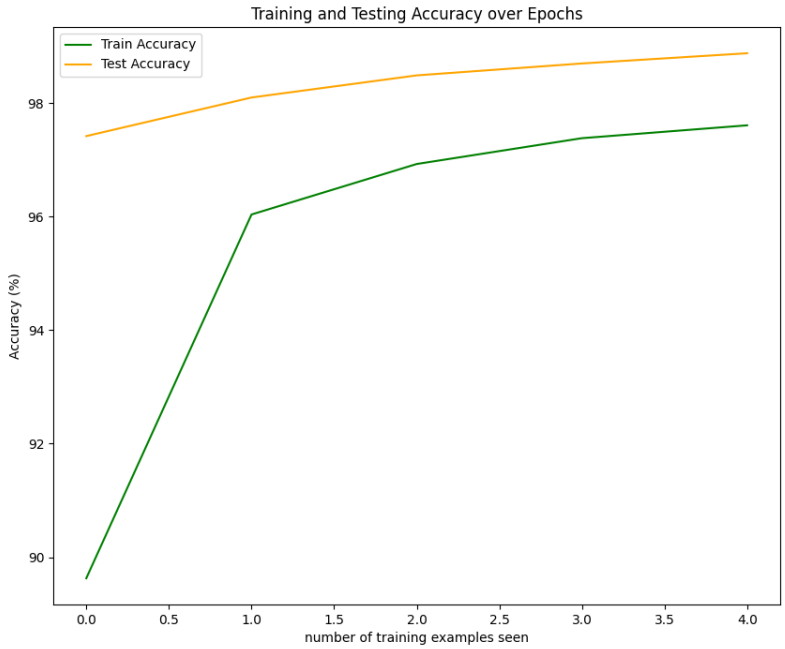

Figure 4: Training and Testing Accuracy Graph

```
Epoch 1, Train Loss: 0.0054, Test Loss: 0.0012, Train Accuracy: 89.63%, Test Accuracy: 97.42%
Epoch 2, Train Loss: 0.0021, Test Loss: 0.0009, Train Accuracy: 96.04%, Test Accuracy: 98.10%
Epoch 3, Train Loss: 0.0016, Test Loss: 0.0007, Train Accuracy: 96.93%, Test Accuracy: 98.49%
Epoch 4, Train Loss: 0.0014, Test Loss: 0.0006, Train Accuracy: 97.38%, Test Accuracy: 98.70%
Epoch 5, Train Loss: 0.0013, Test Loss: 0.0006, Train Accuracy: 97.61%, Test Accuracy: 98.88%
Model saved to mnist_model.pth
```

Figure 5: Epoch Results

Figure 3: The graph displays the decline in training and testing loss across five epochs. The training loss is depicted in blue, while the testing loss is shown in red, both demonstrating a steady decrease as the number of training examples increases.

Figure 4: This graph illustrates the training and testing accuracy of the model over five epochs. Training accuracy is represented in green and testing accuracy in orange, highlighting the model's consistent improvement and high performance on unseen data.

Figure 5: A summary of the model's performance metrics at the end of each epoch. The table records training and testing loss, along with accuracy percentages, showcasing the model's learning efficiency and generalization capability over successive epochs.

E. Read the network and run it on the test set:

After training the convolutional neural network on the MNIST digit recognition dataset, I proceeded to evaluate its performance on unseen data. The trained network was loaded using PyTorch's model loading capabilities, ensuring that it was set to evaluation mode to disable dropout and use the learned weights for prediction. We then ran the network on the first ten examples from the MNIST test set, which had not been used during the training process.

Figure 6 visualized the first nine digits of the test set alongside their predicted labels. This visualization provides an intuitive understanding of how the model interprets each digit and arrives at its predictions.

Figures 7 a and 7 b display the network output values alongside the predicted and actual labels for the first ten examples of the MNIST test dataset. Each row corresponds to an example where the network's confidence scores (logits) across all possible classes (digits 0-9) are presented with the most confident prediction highlighted as the 'Predicted Label,' alongside the ground truth as the 'Actual Label.

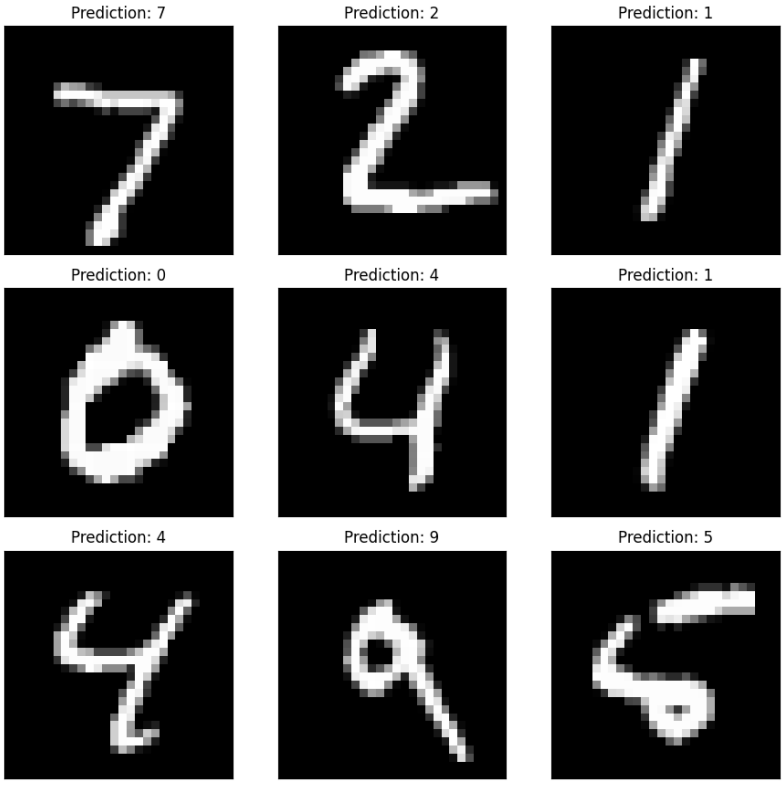

Figure 6: Visualization of the first nine test set digits with the model's predictions.

```
Model predictions and actual labels:

Example 1:
Network output: [-1.9986103e+01 -1.8751457e+01 -1.1709021e+01 -1.1929575e+01
 -2.3663143e+01 -1.7577431e+01 -3.3092155e+01 -1.6808368e-05
 -1.6364731e+01 -1.3197071e+01]
Predicted label: 7, Actual label: 7

Example 2:
Network output: [-1.7351803e+01 -1.3087944e+01 -2.1457649e-06 -1.9304728e+01
 -2.2273514e+01 -2.9573908e+01 -1.6885485e+01 -2.7206087e+01
 -2.2835640e+01 -3.1573082e+01]
Predicted label: 2, Actual label: 2

Example 3:
Network output: [-1.69331417e+01 -1.03711545e-05 -1.36049623e+01 -1.76562042e+01
 -1.20981264e+01 -1.58087292e+01 -1.69541206e+01 -1.26173420e+01
 -1.60416374e+01 -1.78947697e+01]
Predicted label: 1, Actual label: 1

Example 4:
Network output: [-7.1642215e-05 -2.0000401e+01 -1.1954471e+01 -1.7881313e+01
 -1.7455952e+01 -1.1569548e+01 -1.0151000e+01 -1.5240446e+01
 -1.2428071e+01 -1.1283943e+01]
Predicted label: 0, Actual label: 0

Example 5:
Network output: [-1.7073099e+01 -2.0723845e+01 -1.5195221e+01 -1.9413868e+01
 -3.3682873e-04 -1.6731676e+01 -1.5148880e+01 -1.3186501e+01
 -1.8299601e+01 -8.0033216e+00]
Predicted label: 4, Actual label: 4
```

Figure 7 a: Model Predictions vs. Actual Labels for MNIST Test Samples

```
Example 6:
Network output: [-2.0004816e+01 -2.2649740e-06 -1.7321728e+01 -2.0398052e+01
 -1.3657813e+01 -1.8814743e+01 -2.0919043e+01 -1.3751096e+01
 -1.7611479e+01 -1.9345049e+01]
Predicted label: 1, Actual label: 1

Example 7:
Network output: [-2.7901978e+01 -1.4849564e+01 -1.9057920e+01 -1.9163506e+01
 -3.2361776e-03 -1.5247418e+01 -2.6363670e+01 -1.0453836e+01
 -8.2983885e+00 -5.8250799e+00]
Predicted label: 4, Actual label: 4

Example 8:
Network output: [-1.4484819e+01 -1.4408221e+01 -7.4981380e+00 -6.7507067e+00
 -6.3046813e+00 -7.2589326e+00 -2.1730787e+01 -9.6318712e+00
 -8.2956982e+00 -4.5827362e-03]
Predicted label: 9, Actual label: 9

Example 9:
Network output: [-1.0912944e+01 -2.1277151e+01 -1.4315619e+01 -1.7111158e+01
 -1.6459047e+01 -8.0653932e-03 -4.8687882e+00 -1.6572374e+01
 -8.0903912e+00 -1.0608462e+01]
Predicted label: 5, Actual label: 5

Example 10:
Network output: [-2.2574347e+01 -2.6231558e+01 -1.9492281e+01 -1.9790518e+01
 -1.1163995e+01 -1.8534889e+01 -3.1911177e+01 -8.7261353e+00
 -1.1671220e+01 -1.8511490e-04]
Predicted label: 9, Actual label: 9

Process finished with exit code 0
```

Figure 7 b: Model Predictions vs. Actual Labels for MNIST Test Samples

F. Test the network on new inputs:

To evaluate the robustness and generalization of my trained model, I conducted a test on a new set of inputs - handwritten digits.

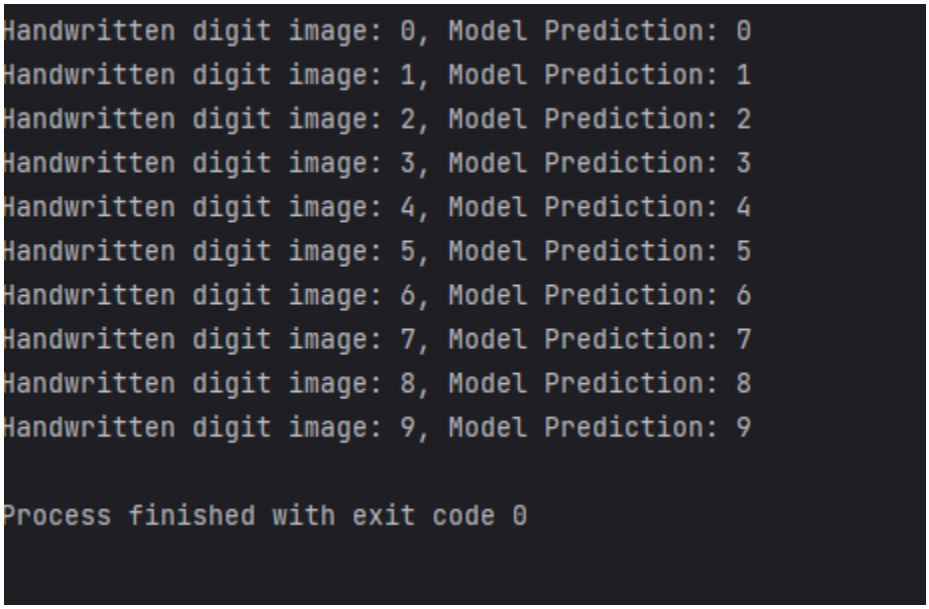

Figure 8: Handwritten Digit Recognition Results

Figure 8: This image captures the successful prediction of handwritten digits by the trained neural network. Each digit from 0 to 9, as written by hand and processed to fit the MNIST input criteria, was correctly identified by the model, demonstrating the efficacy and generalizability of the trained network.

3. Examine your network:

A. Analyze the first layer:

Examined the convolutional neural network responsible for classifying MNIST digits by concentrating on the first convolutional layer. Here, the network interacts with the raw pixel input for the first time, using its array of filters to learn to recognize basic visual signals. The fundamental units of the network are the filters, which each serve as a unique lens for feature identification.

Upon inspecting the weights of the first layer's filters, a compelling variety is uncovered. These filters are the initial touchpoint for our network's pattern recognition journey, each mapping out a unique facet of the input data. Their visualization, as seen in the attached image, offers a window into the network's perception and the diversity of patterns it's equipped to recognize.

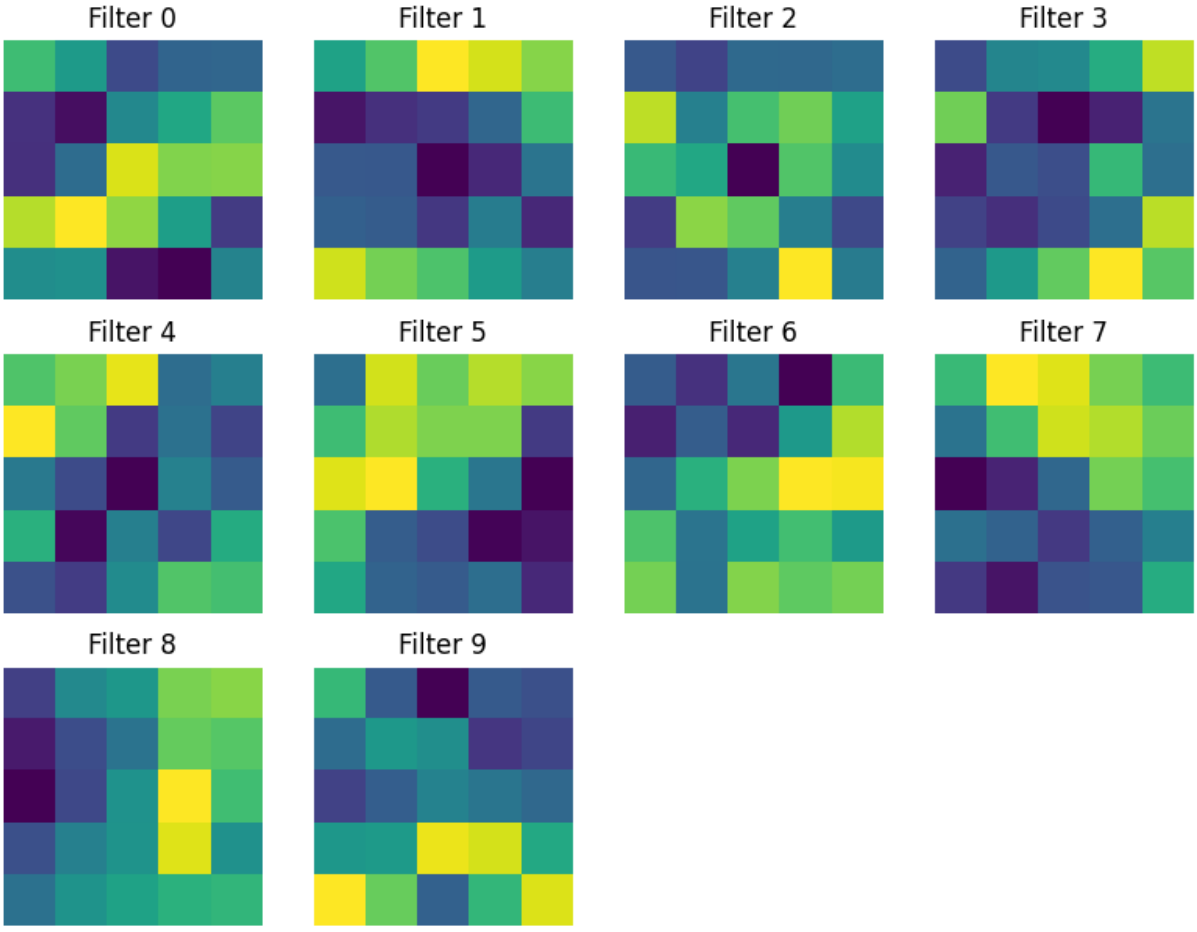

Figure 9: Visualization of the First Convolutional Layer Filters

Figure 9: Each square represents one of the ten 5x5 filters in the network's first convolutional layer, showcasing the variety of patterns learned to detect distinctive features within the MNIST digit images.

B. Show the effect of the filters:

I explored the network's first layer by applying its ten convolutional filters to the MNIST dataset's first training image, aiming to uncover how these filters process input to detect features like edges and textures. This step is crucial for understanding the feature extraction phase in digit recognition, revealing what the network sees in the early stages.

The resulting filtered images varied significantly, with some emphasizing edges and others highlighting specific patterns, demonstrating the network's ability to pick up diverse features for classification. This variation underscores the convolutional layer's role in distinguishing between numerals based on subtle differences, confirming the effectiveness of these filters in feature extraction.

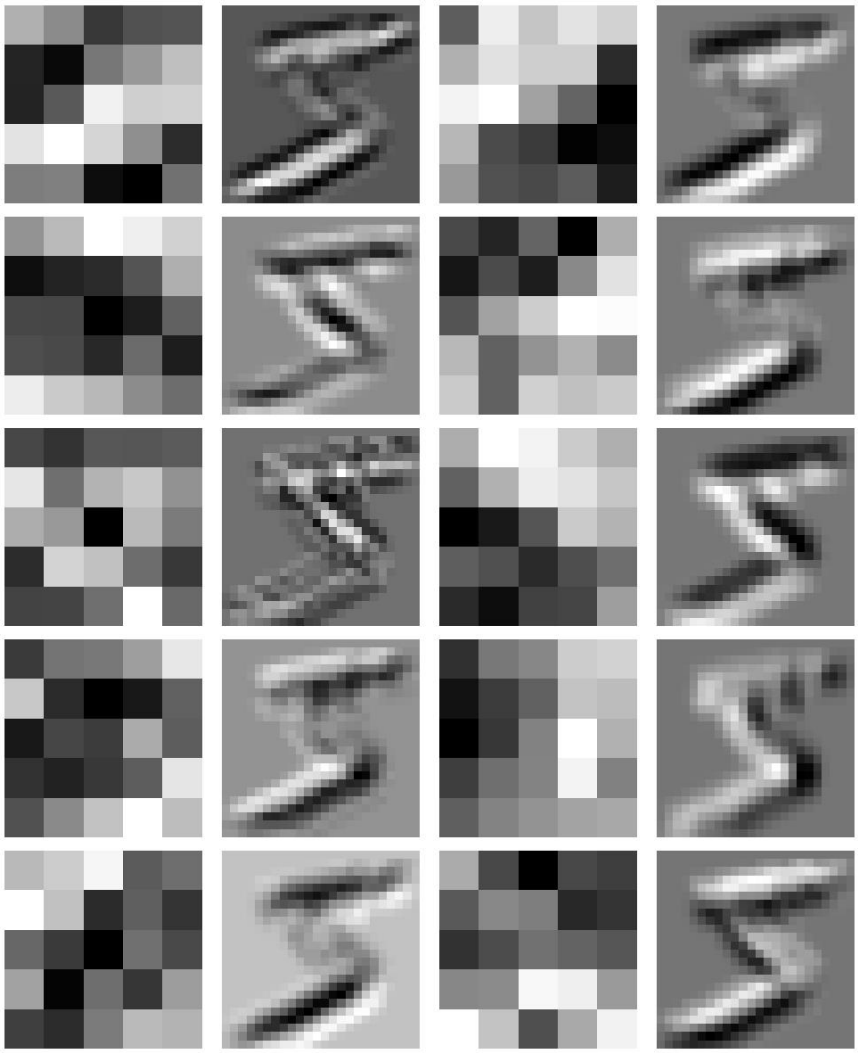

Figure 10: Effects of First Layer Filters on an MNIST Image.

Figure 10: This figure illustrates the diverse feature detection capabilities of the network's initial convolutional filters, highlighting their role in early pattern recognition for digit classification.

4. Transfer Learning on Greek Letters:

This task explores the field of transfer learning by using a pre-trained MNIST digit identification model. It adjusts its capabilities to the complex task of identifying Greek letters. This task not only underscores the versatility of neural networks but also their ability to transfer learned knowledge across different domains. By fine-tuning a network initially versed in digit recognition, we ventured into classifying three Greek letters: alpha, beta, and gamma, showcasing the model's adaptability and the efficacy of transfer learning strategies.

```
MyNetwork(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2_drop): Dropout2d(p=0.5, inplace=False)
  (fc1): Linear(in_features=320, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=5, bias=True)
)
```

Figure 11: Model Architecture

Figure 11: The model's final architecture, as modified for this task, included two convolutional layers, a dropout layer, and two fully connected layers, with the last layer specifically tailored to classify five distinct outputs.

```
Epoch 1/20, Train Loss: 1.9151007788521903, Train Accuracy: 28.571428571428573%, Val Accuracy: 33.333333333333336%
Epoch 2/20, Train Loss: 1.9758284716379075, Train Accuracy: 23.80952380952381%, Val Accuracy: 33.333333333333336%
Epoch 3/20, Train Loss: 1.7339141141800654, Train Accuracy: 28.571428571428573%, Val Accuracy: 50.0%
Epoch 4/20, Train Loss: 1.6183183647337414, Train Accuracy: 38.095238095238095%, Val Accuracy: 50.0%
Epoch 5/20, Train Loss: 1.3263035203729356, Train Accuracy: 42.857142857142854%, Val Accuracy: 50.0%
Epoch 6/20, Train Loss: 1.2530789829435802, Train Accuracy: 42.857142857142854%, Val Accuracy: 50.0%
Epoch 7/20, Train Loss: 1.1966410705021449, Train Accuracy: 42.857142857142854%, Val Accuracy: 50.0%
Epoch 8/20, Train Loss: 1.0255625191188993, Train Accuracy: 57.142857142857146%, Val Accuracy: 66.66666666666667%
Epoch 9/20, Train Loss: 0.9409990906715393, Train Accuracy: 71.42857142857143%, Val Accuracy: 100.0%
Epoch 10/20, Train Loss: 1.075997925940014, Train Accuracy: 47.61904761904762%, Val Accuracy: 100.0%
Epoch 11/20, Train Loss: 0.9933861437298003, Train Accuracy: 57.142857142857146%, Val Accuracy: 100.0%
Epoch 12/20, Train Loss: 0.8187148457481748, Train Accuracy: 71.42857142857143%, Val Accuracy: 100.0%
Epoch 13/20, Train Loss: 0.7003846878097171, Train Accuracy: 71.42857142857143%, Val Accuracy: 100.0%
Early stopping triggered after 14 epochs.
```

Figure 12: Training Over 20 Epochs

Figure 12: Training was conducted over 20 epochs, with early stopping initiated after 14 epochs to prevent overfitting, as the validation accuracy reached 100%.

Figure 13: Training and Validation Loss

Figure 13: The training process revealed an interesting progression of learning, as evidenced by the training and validation losses plotted over each epoch.

```
Results on Additional Data:

Predicted Greek alphabet for File additional_greek_letters\alpha\alpha_01.png is: alpha

Predicted Greek alphabet for File additional_greek_letters\alpha\alpha_02.png is: alpha

Predicted Greek alphabet for File additional_greek_letters\alpha\alpha_03.png is: alpha

Predicted Greek alphabet for File additional_greek_letters\beta\beta_01.png is: beta

Predicted Greek alphabet for File additional_greek_letters\beta\beta_02.png is: beta

Predicted Greek alphabet for File additional_greek_letters\beta\beta_03.png is: beta

Predicted Greek alphabet for File additional_greek_letters\gamma\gamma_01.png is: gamma

Predicted Greek alphabet for File additional_greek_letters\gamma\gamma_02.png is: gamma

Predicted Greek alphabet for File additional_greek_letters\gamma\gamma_03.png is: gamma

Predicted Greek alphabet for File additional_greek_letters\gamma\gamma_04.png is: gamma
```

Figure 14: Classifying Greek Letters

Figure 14: Upon evaluating the model with additional Greek letter images not seen during training, the model accurately predicted the labels for all test images. The correct classification of alpha, beta, and gamma images showcases the model's capability to generalize from digits to Greek letters.

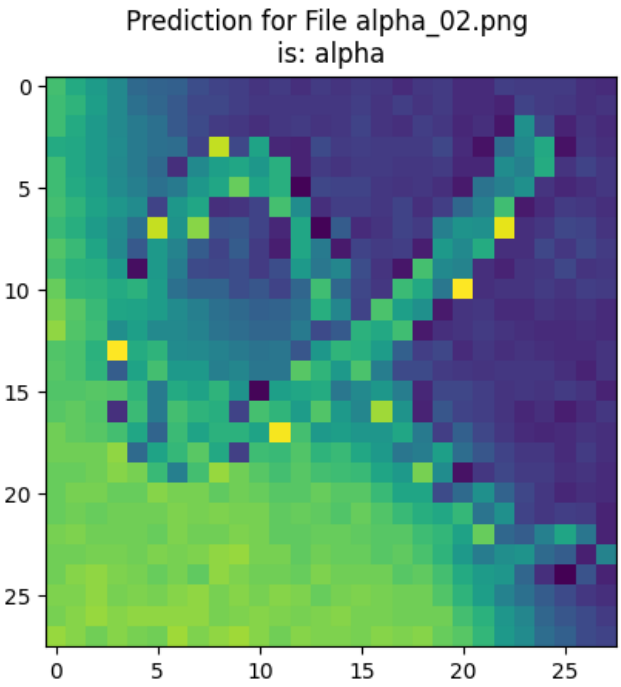

Figure 15: Handwritten Greek letter Alpha.

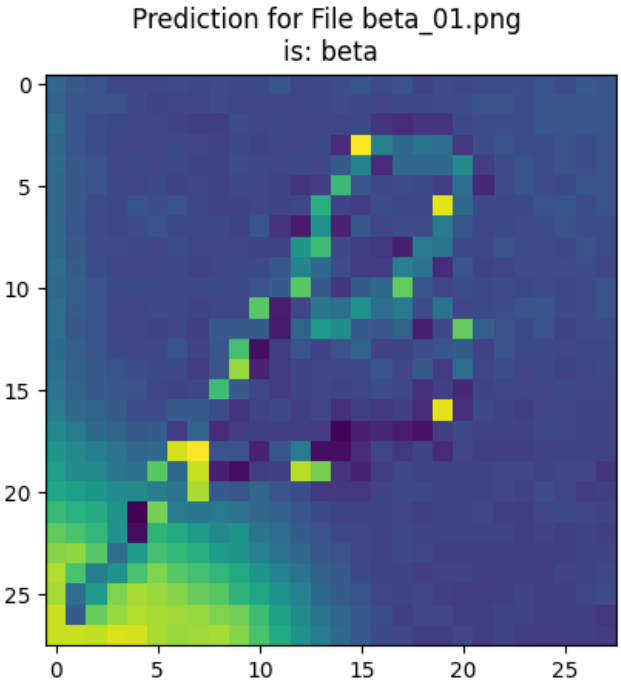

Figure 16: Handwritten Greek letter Beta.

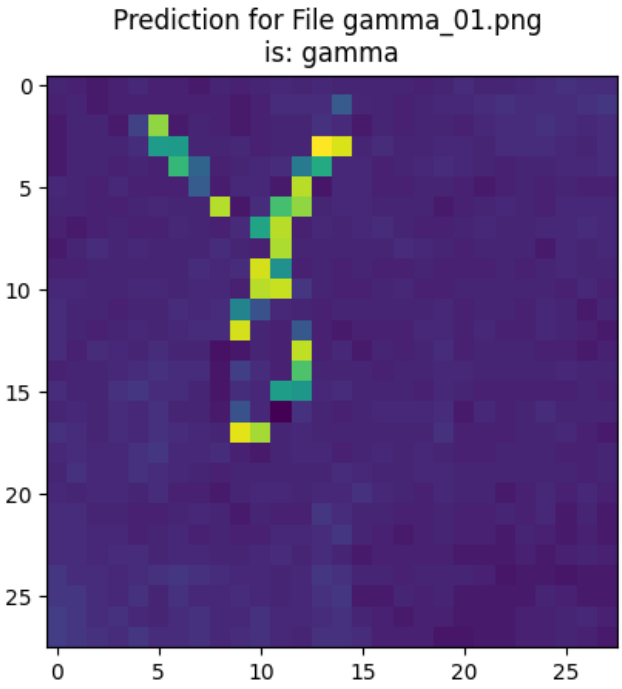

Figure 17: Handwritten Greek letter Gamma.

Figure 15, Figure 16, and Figure 17 displays a handwritten Greek letter image alongside the model's prediction. The title 'Prediction for File xyz.png is: label' provides the specific filename and the model's predicted label for the image. This visualization helps in understanding the model's classification performance on unseen data by comparing the predicted labels against actual Greek letters depicted in the images.

For further inspection and replication of these results, a collection of the additional Greek letter images used in this study, is available for download at the following link.

5. Design your own experiment:

B. Predict the results:

Hypotheses Summary

1. **Number of Filters**: Increasing the number of filters in convolutional layers might lead to higher accuracy since the network can learn more complex features. However, too many filters could lead to overfitting and increased computational cost.
2. **Dropout Rate**: A moderate dropout rate (e.g., around 0.3 to 0.5) is expected to help in reducing overfitting by preventing complex co-adaptations on training data, thus improving model generalization. Very low or very high dropout rates might adversely affect performance due to underfitting or excessive information loss, respectively.
3. **Number of Hidden Nodes**: Increasing the number of hidden nodes allows the model to capture more complex relationships. However, like filters, too many hidden nodes might increase the risk of overfitting and computational expense.
4. **Learning Rate**: Optimal learning rates are crucial for model convergence. Too high learning rates might cause the model to oscillate or diverge, while too low rates might lead to slow convergence or getting stuck in local minima.
5. **Epochs**: More training epochs generally lead to better model performance up to a point, after which the model might start overfitting. The ideal number of epochs balances learning sufficiently complex patterns without memorizing the training set.
6. **Batch Size**: Larger batch sizes offer more stable gradient estimates, leading to smoother convergence. However, too large batch sizes might reduce the model's ability to generalize from the training data due to less stochasticity.

These hypotheses articulate the delicate balance required in tuning neural network hyperparameters to achieve optimal performance. They reflect an understanding that while increasing model complexity and training intensity can improve learning capacity, there must be careful consideration of the potential for overfitting and computational inefficiency. The evaluation of these hypotheses through experimentation will not only validate or challenge these predictions but also contribute to a deeper understanding of model behavior under various configurations.

C. Execute your plan:

Overview

My research was focused on determining how variations in neural network configurations impact the accuracy of a classification model. Through systematic adjustments to dropout rate, number of filters, number of hidden nodes, learning rate, epochs, and batch size, I tried to identify optimal settings that balance model performance with computational efficiency.

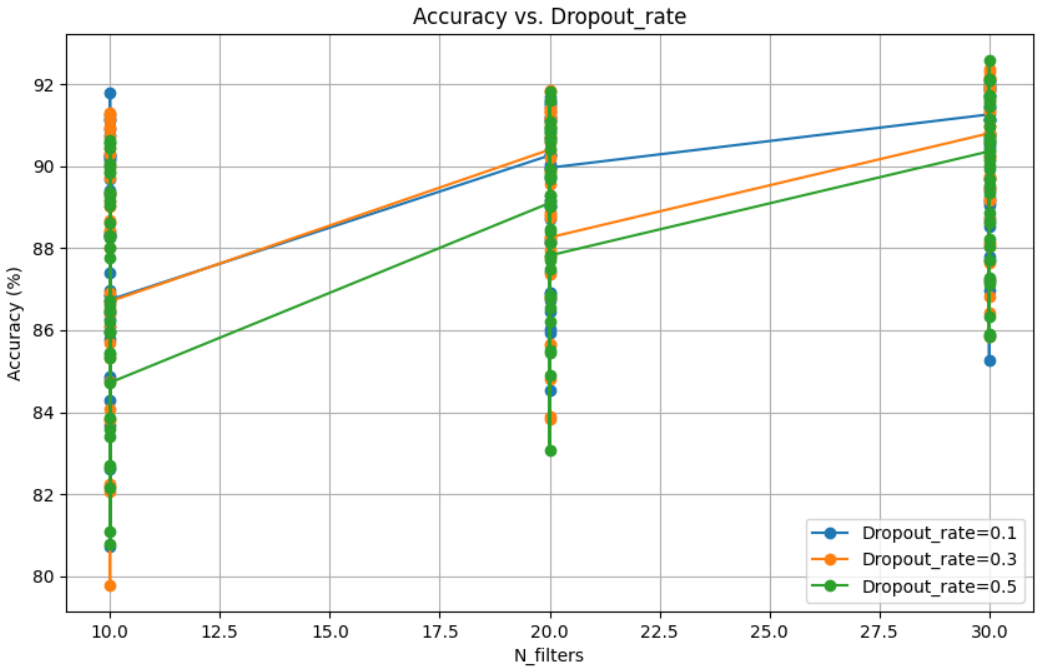

Figure 18: Accuracy vs. Dropout Rate Plot

1. **Accuracy vs. Dropout Rate**: The plot revealed that a dropout rate of around 0.3 led to the highest model accuracy. Rates lower than 0.3 resulted in slightly reduced accuracy, possibly due to overfitting, while rates higher than 0.3 saw a decline in performance, likely due to excessive regularization leading to loss of relevant information.

   Conclusion: This supports the hypothesis that a moderate dropout rate effectively balances regularization with the model's ability to learn complex patterns.

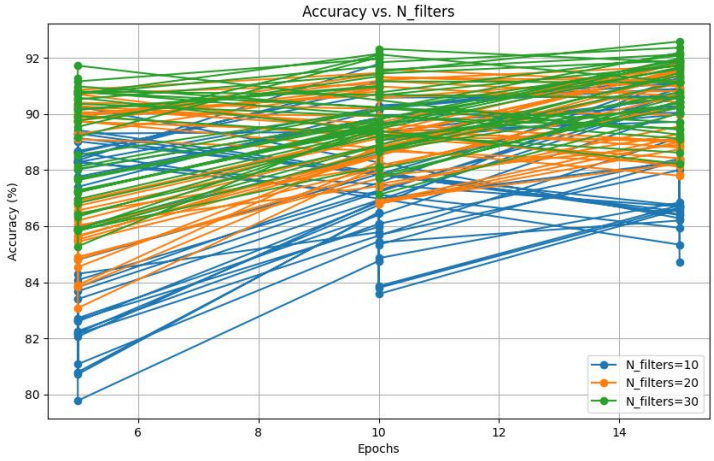

Figure 19: Accuracy vs. Number of Filters Plot

2. **Accuracy vs. Number of Filters:** Models configured with 20 filters consistently outperformed those with fewer or more filters. This suggests that 20 filters provide a sweet spot for capturing sufficient feature complexity without overcomplicating the model.

   Conclusion: Aligning with the hypothesis, a higher number of filters up to an optimal point improves model accuracy by capturing more complex features.

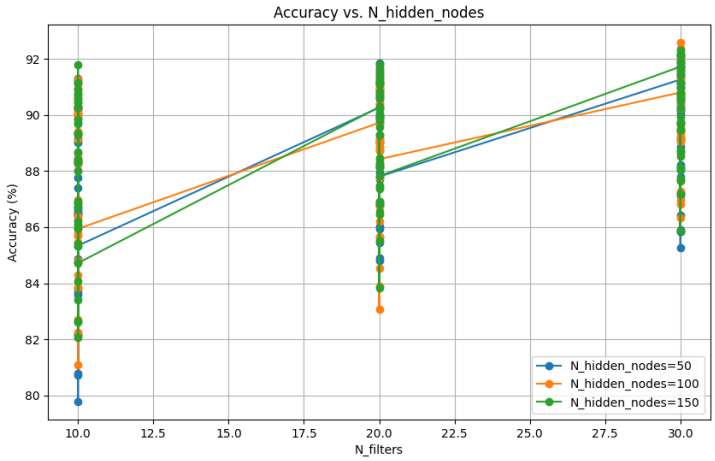

Figure 20: Accuracy vs. Number of Hidden Nodes Plot

3. **Accuracy vs. Number of Hidden Nodes:** The optimal range of hidden nodes appeared to be between 100 and 150, where models achieved the highest accuracy. Beyond 150 hidden nodes, accuracy gains were marginal, indicating potential overfitting.

   Conclusion: This outcome validates the hypothesis that an adequate number of hidden nodes is crucial for modeling complex relationships, with diminishing returns beyond a certain threshold.

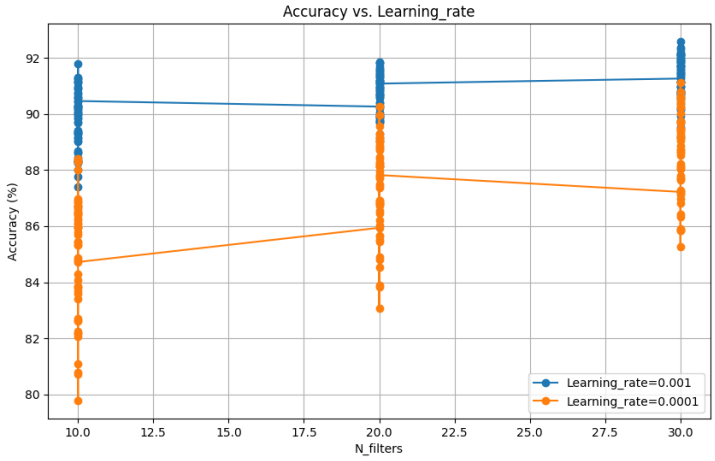

Figure 21: Accuracy vs. Learning Rate Plot

4. **Accuracy vs. Learning Rate:** A learning rate of 0.001 was optimal for model performance. Rates higher than 0.001 led to unstable training dynamics, while lower rates (0.0001) resulted in slower convergence and lower final accuracy.

   Conclusion: The observed behavior confirms the hypothesis on the critical importance of selecting an appropriate learning rate for efficient model training.

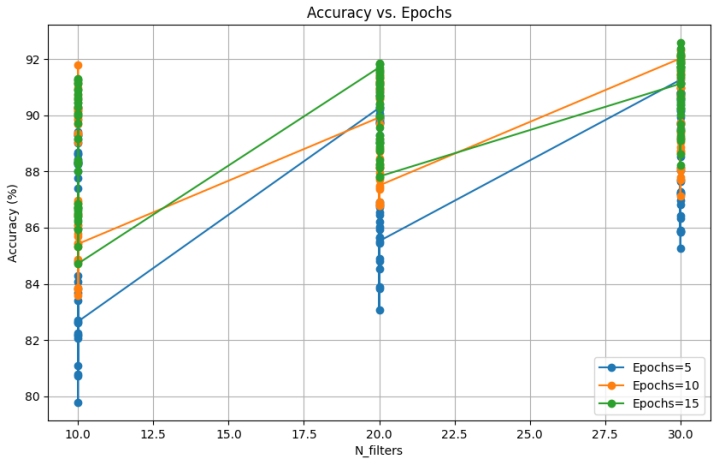

Figure 22: Accuracy vs. Epochs Plot

5. **Accuracy vs. Epochs:** Increasing the number of epochs generally improved accuracy, with diminishing returns noted beyond 10 epochs. This plateau effect underscores the balance between sufficient training and the risk of overfitting with too many epochs.

   Conclusion: This result supports the hypothesis that more training epochs lead to better performance up to a point, after which overfitting becomes a concern.

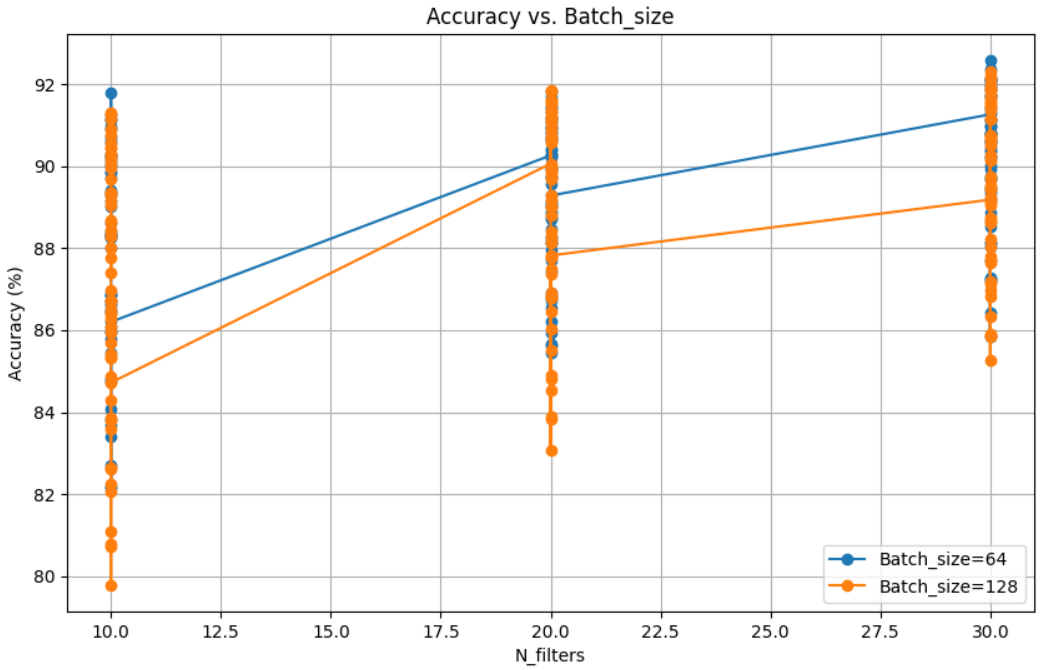

Figure 23: Accuracy vs. Batch Size Plot

6. **Accuracy vs. Batch Size:** Smaller batch sizes (64) slightly outperformed larger ones (128), suggesting that more frequent updates contribute to better model accuracy, albeit with longer training times.

   Conclusion: The findings align with the hypothesis that smaller batch sizes may offer advantages in terms of accuracy due to their more frequent update cycles, despite the trade-off in computational efficiency.

Summary

Overall, the experimental findings showed how each hyperparameter had a subtle effect on the model's performance, confirming the initial hypotheses. It became evident that a careful balance is required to optimize a model's accuracy while mitigating the risks of overfitting and ensuring computational feasibility. The insights gained from this analysis are invaluable for guiding future model tuning efforts, particularly in the context of classification tasks within the domain of pattern recognition and computer vision.

9. Extensions:

A. Live Video Digit Recognition:

The system's capability to use the convolutional neural networks (CNNs) to identify and classify digits from 0 to 9 in real-time through a webcam feed. It combines PyTorch for model inference and OpenCV for video collection and processing to create a smooth digit recognition interface. This project not only highlights the usefulness of neural networks for pattern recognition, but also the effectiveness of using these models in practical applications.

Figure 24 – 33, are the screenshots of the live video digit recognition application successfully identifying each digit from 0 to 9.

Additionally, a video link included in the report offers a dynamic view of the application in action, capturing its capability to recognize digits in real-time.

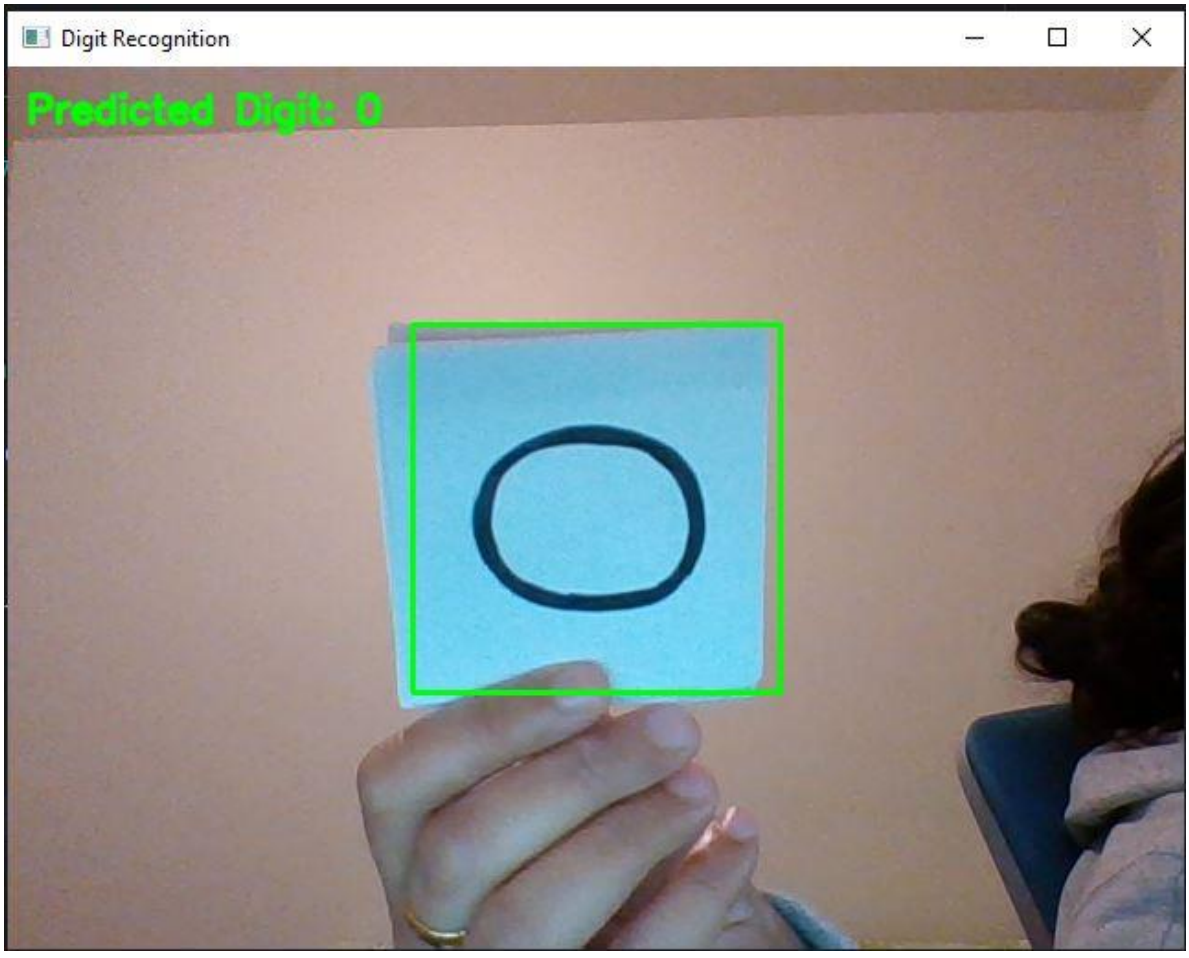

Figure 24: Real-time Recognition of Digit 0

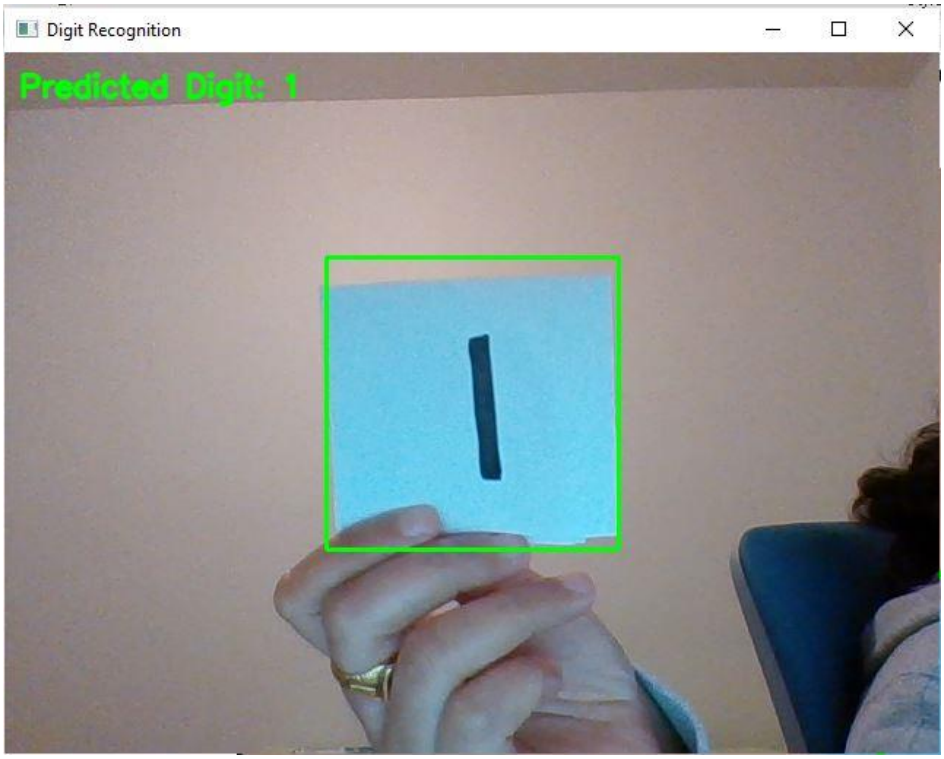

Figure 25: Real-time Recognition of Digit 1

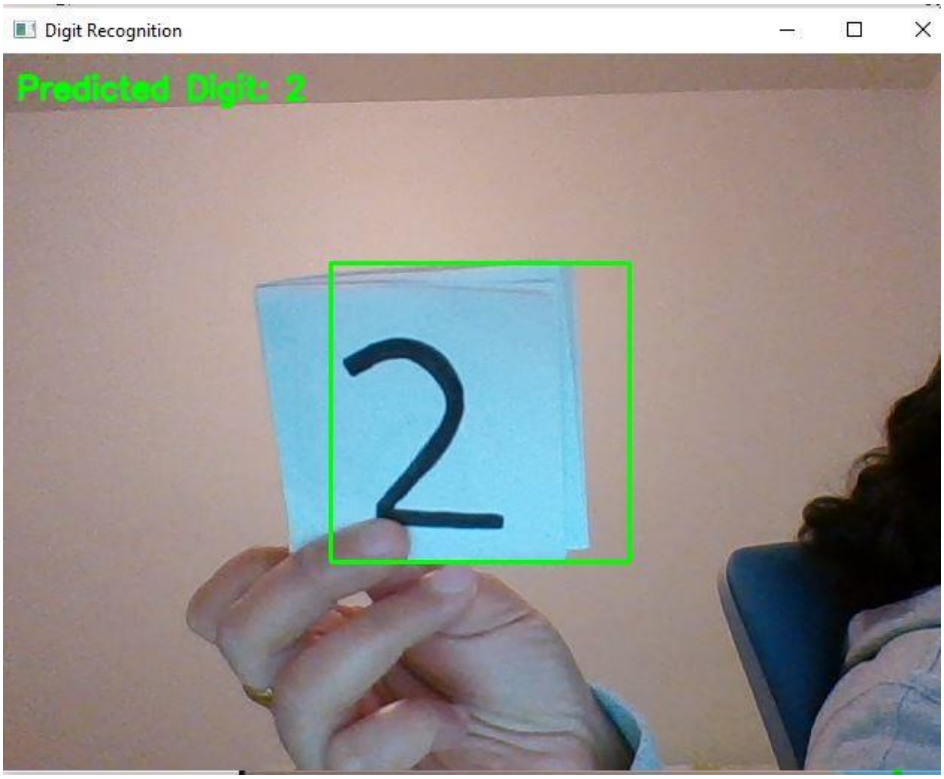

Figure 26: Real-time Recognition of Digit 2

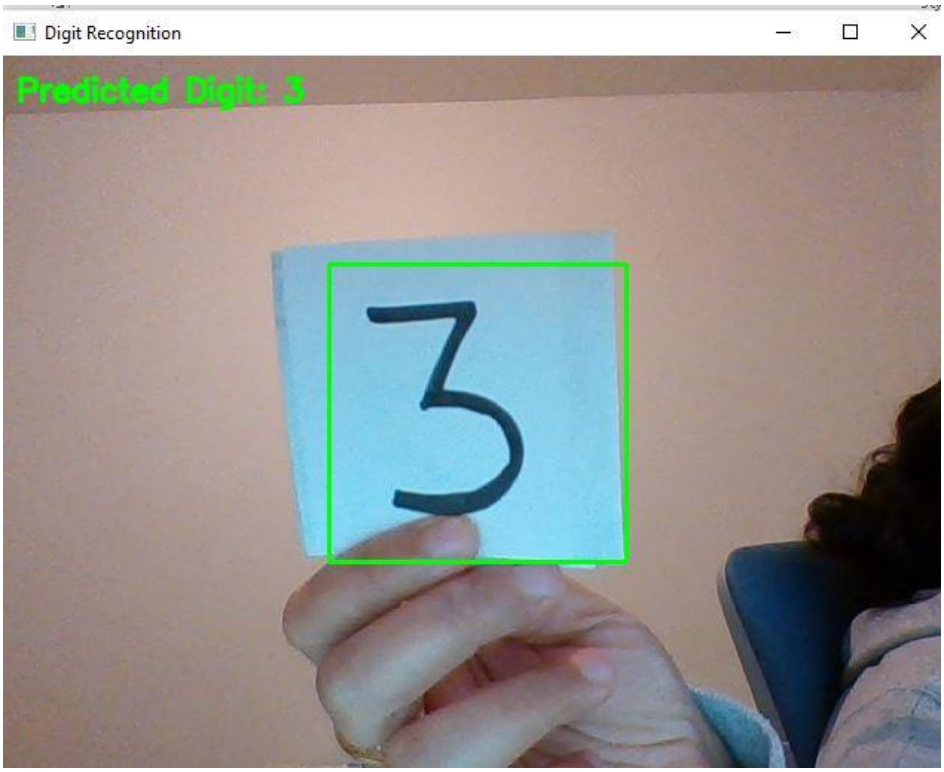

Figure 27: Real-time Recognition of Digit 3

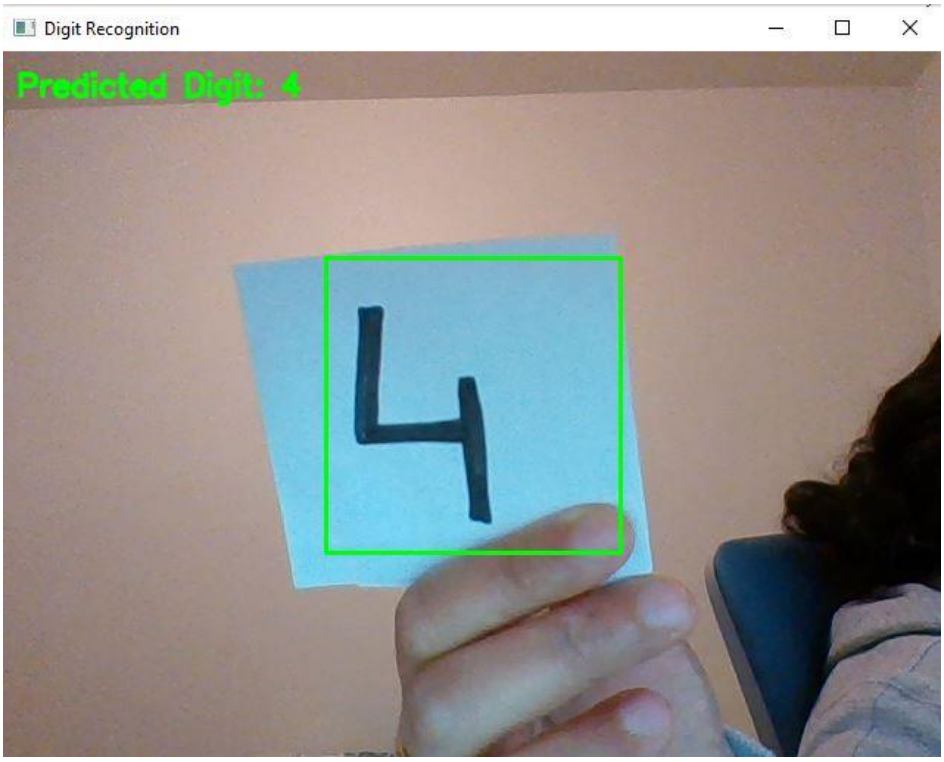

Figure 28: Real-time Recognition of Digit 4

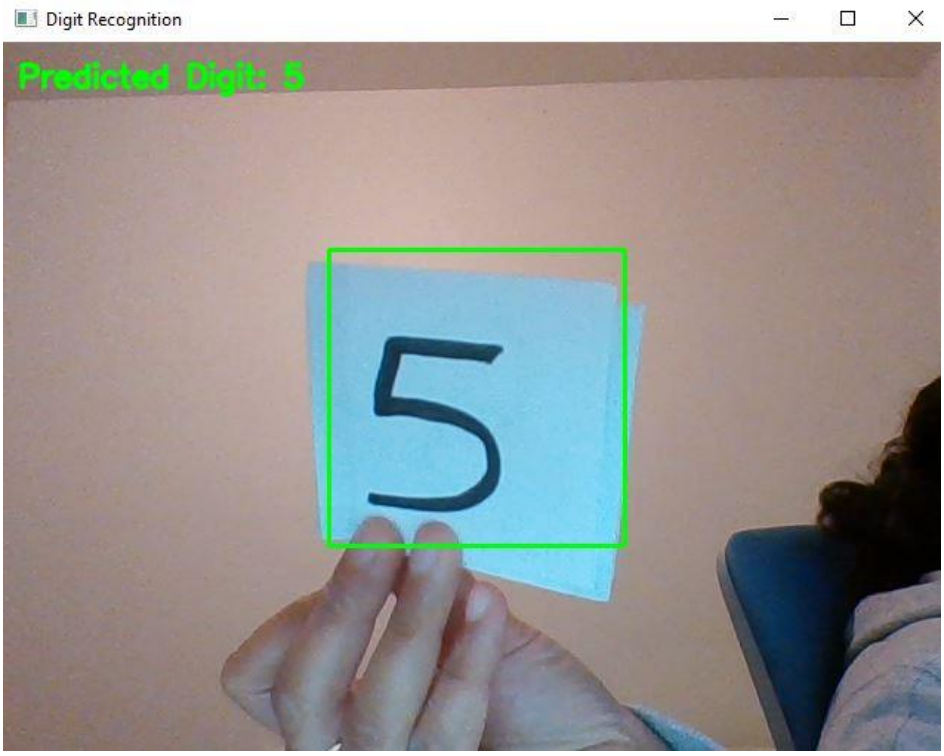

Figure 29: Real-time Recognition of Digit 5

Figure 30: Real-time Recognition of Digit 6

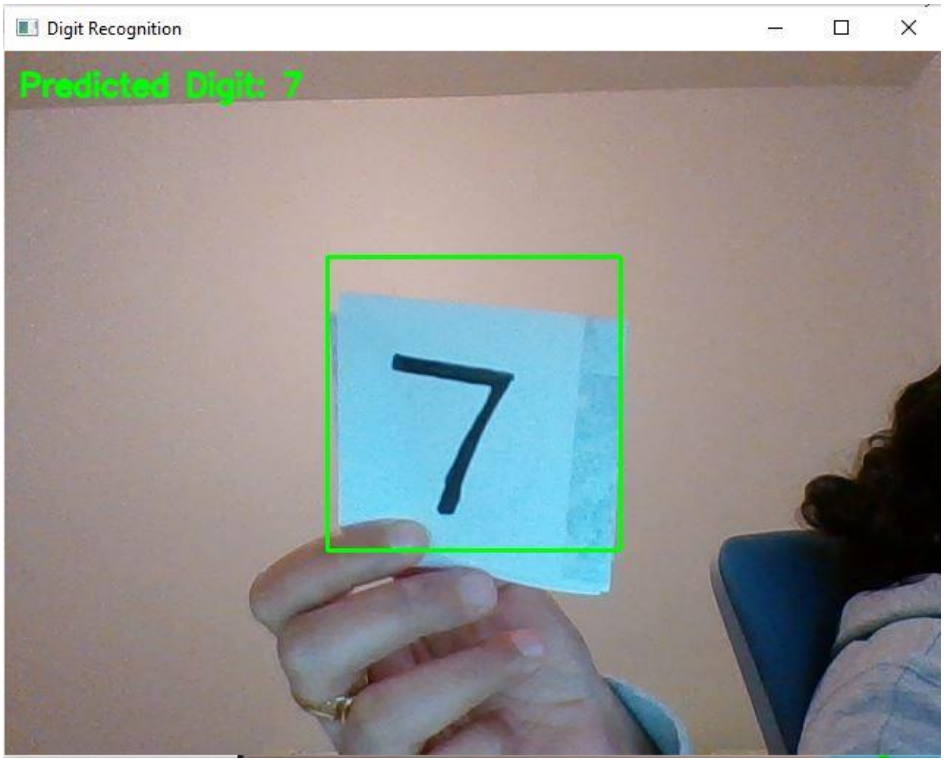

Figure 31: Real-time Recognition of Digit 7

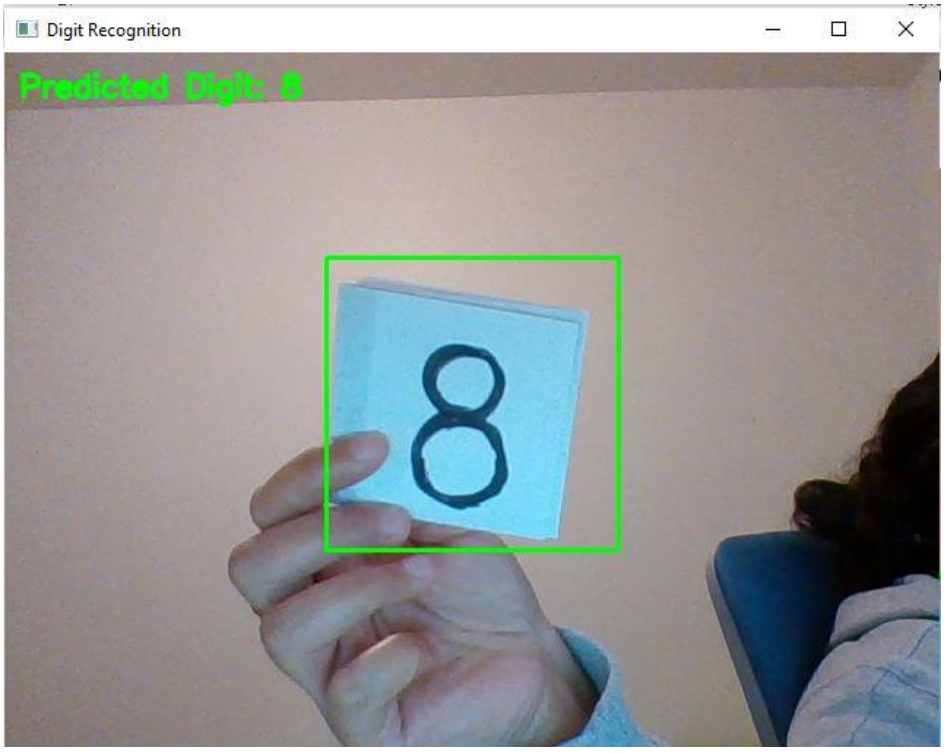

Figure 32: Real-time Recognition of Digit 8

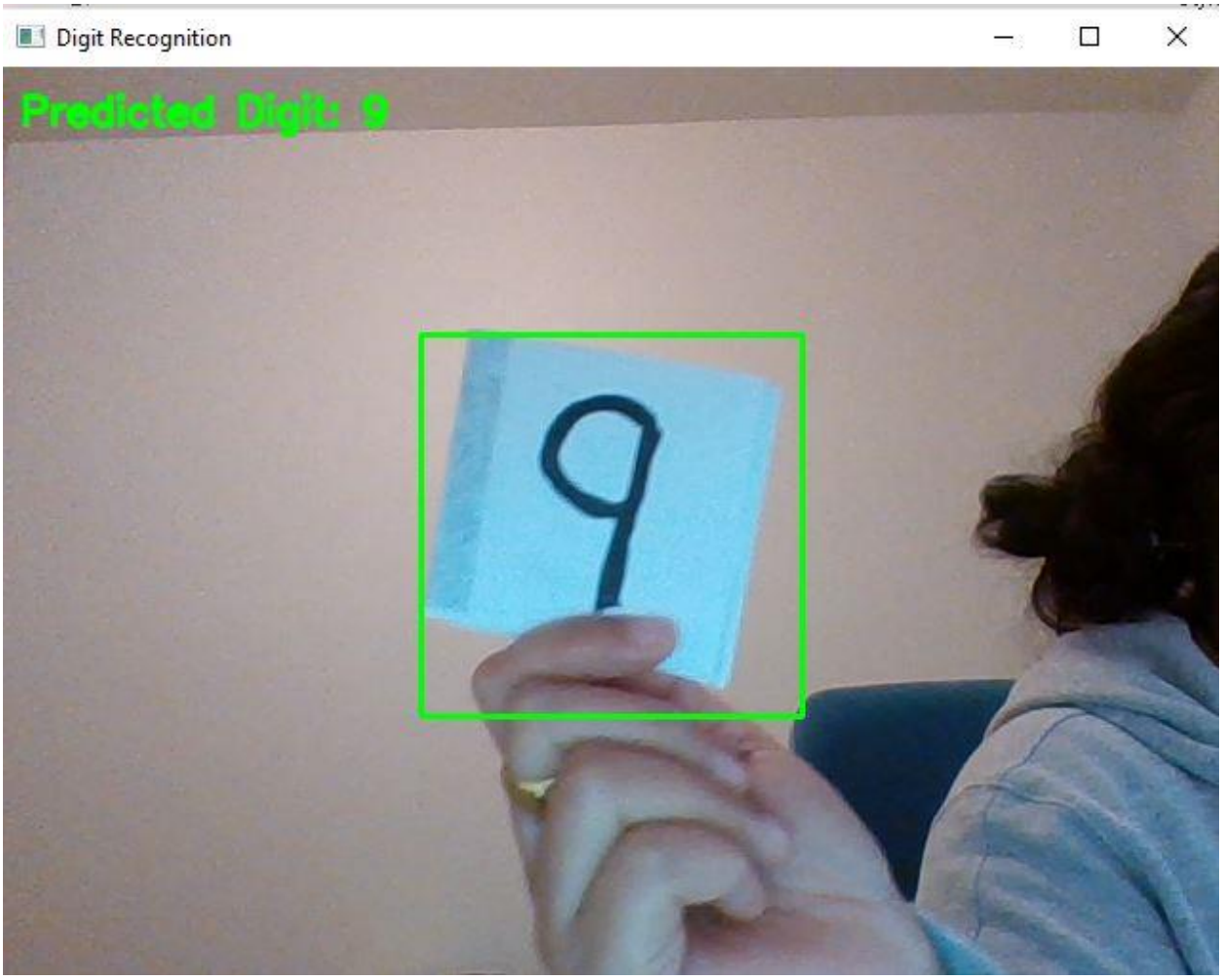

Figure 33: Real-time Recognition of Digit 9

B. Evaluating Pre-trained Networks:

In this, the pre-trained networks included in the PyTorch package were explored and analyzed with a particular emphasis on ResNet18, a model well-known for its depth and efficiency in image classification tasks. The aim was to examine the behavior and functionality of the initial convolutional layers of this pre-trained model, like the work done in Task 2. By loading ResNet18, we tapped into a network trained on the extensive ImageNet dataset, enabling us to investigate the intricate details of its first convolutional layer and its effects on input images.

Upon loading ResNet18, the structure revealed an architecture designed to process images through a series of convolutional, batch normalization, and ReLU layers, culminating in a fully connected layer that outputs 1000 classes. The focus of this analysis was the first convolutional layer, denoted as "conv1", which consists of 64 filters with a kernel size of 7x7. This layer plays a pivotal role in capturing the primary features from input images, setting the stage for deeper, more complex pattern recognition as the data progresses through the network.

A pineapple image served as the test subject for this analysis. By applying the filters from the first convolutional layer to this image, generated activation maps that provided a visual representation of how ResNet18 perceives and processes the image at the initial stage. These activation maps highlighted various aspects of the image that the network deemed significant, such as edges, textures, and color gradients. The visualization of these filters and their corresponding activation maps offered insightful clues into the network's feature extraction process, revealing the complex yet systematic approach ResNet18 employs to understand and classify images.

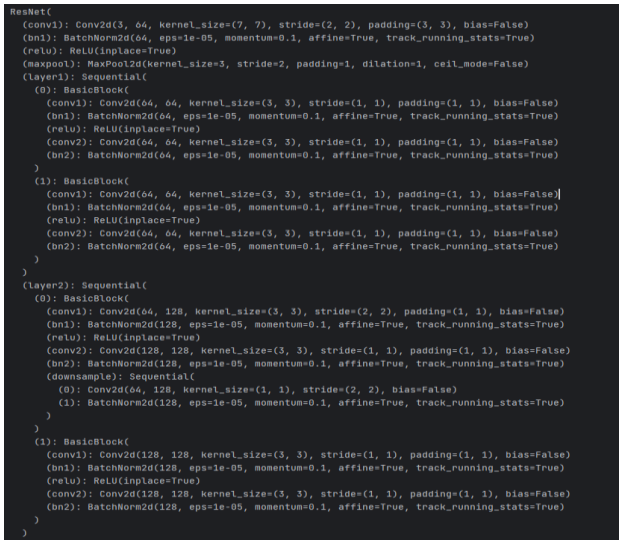

Figure 34: ResNet18 Architecture Overview (Part 1)

Figure 34: ResNet18 Architecture Overview (Part 1) - This image provides a detailed view of the initial layers of the ResNet18 architecture, covering from the first convolutional layer (conv1) up to the second sequential layer (layer2). It highlights the network's foundational blocks, including convolutional layers, batch normalization, ReLU activation, and max pooling, which are essential for early feature extraction and spatial reduction.

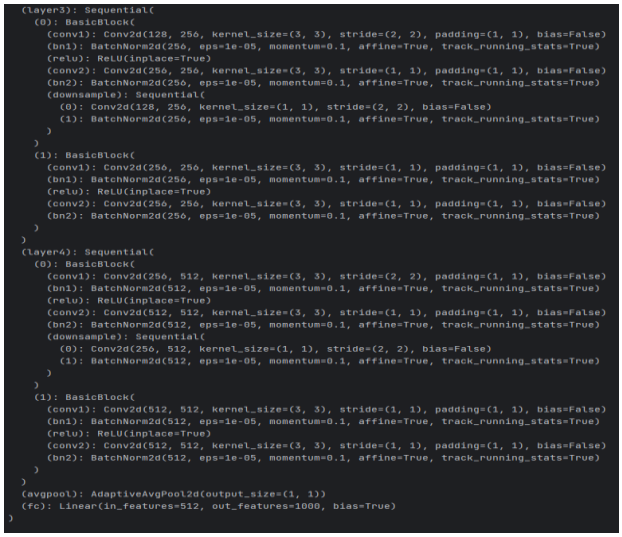

Figure 35: ResNet18 Architecture Overview (Part 2)

Figure 35: ResNet18 Architecture Overview (Part 2) - This image continues the exploration of the ResNet18 architecture, presenting the deeper layers from the third sequential layer (layer3) to the final fully connected layer (fc). It showcases the advanced blocks of the network that further refine and abstract features, ultimately leading to the network's ability to classify images into 1000 different categories.

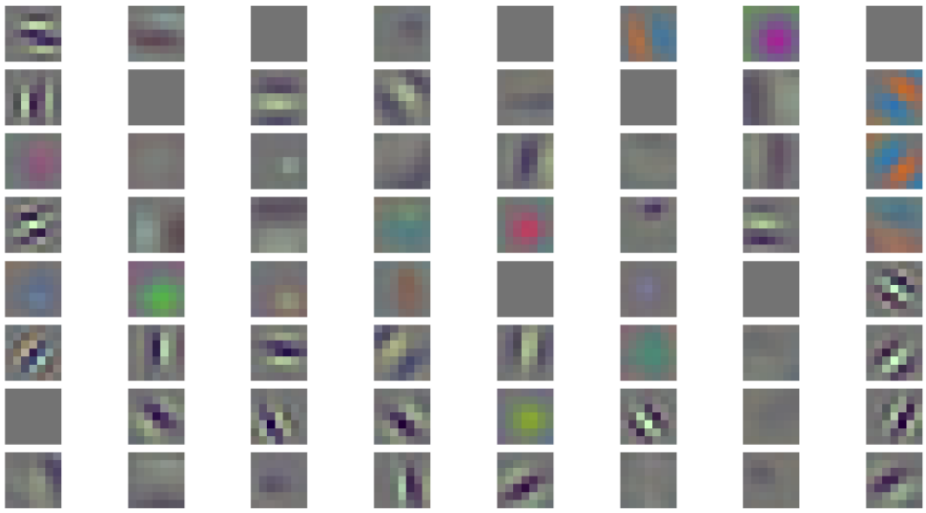

Figure 36: Visualization of the first convolutional layer's filters in the ResNet18 model.

Figure 36: Visualization of the first convolutional layer's filters in the ResNet18 model - This image showcases the diverse patterns and structures that the network has learned to recognize. Each subplot represents one of the 64 filters used by the network to extract features from the input image, highlighting the complexity and variety of visual cues that contribute to the model's understanding of visual data.

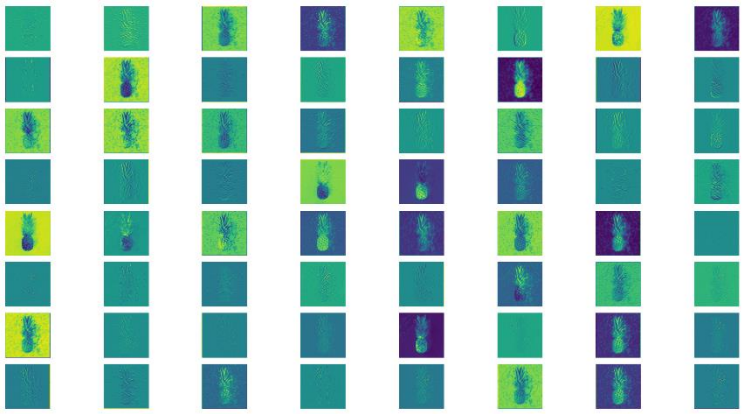

Figure 37: Visualization of Activation Maps

Figure 37: Visualization of Activation Maps - This image showcases the activation maps generated by applying the first convolutional layer filters of ResNet18 to a pineapple image. Each map highlights different features captured by the filters, such as edges, textures, and contrasts, demonstrating the initial stage of feature extraction performed by the network.

A Short Reflection of What I Learned

Embarking on this project, I ventured into the field of deep learning with a focus on the details of constructing and understanding neural networks. One of the key insights gained is the importance of data preprocessing and how the representation of input data significantly affects the network's ability to learn. Through the application of convolutional filters, I witnessed firsthand the network's capacity to extract meaningful features from raw pixels, a cornerstone of its pattern recognition capabilities.

The exploration of transfer learning illuminated the potential of pretrained models, showcasing how a network trained on one task can be adapted to another with minimal effort. This not only underscores the versatility of deep learning models but also highlights the efficiency gains from leveraging existing knowledge.

The experimentation phase of the project was particularly enlightening, offering hands-on experience with tuning network architectures. It revealed the delicate balance between model complexity and generalization, emphasizing the trade-offs involved in optimizing for performance versus training efficiency.

Overall, this project served as a profound learning experience, solidifying my understanding of deep learning principles and their practical application. It has equipped me with a solid foundation for tackling more complex recognition tasks and has sparked a keen interest in exploring further advancements in the field of deep learning.

Acknowledgements

The journey through the Recognition using Deep Networks was an enlightening experience that broadened my understanding of deep learning and neural networks. The success of this project can be attributed to a range of textbooks, websites, and the invaluable support of individuals.:

Textbooks:

- "Computer Vision: Algorithms and Applications, 2nd Edition" by Richard Szeliski provided the theoretical backbone for understanding the complex algorithms involved in camera calibration and 3D reconstruction. This book was indispensable in offering a clear and in-depth look at computer vision techniques.
- The course materials and lectures for the Pattern Recognition & Computer Vision course provided a structured and comprehensive overview of the concepts and methodologies applied throughout this project.

Websites:

- PyTorch Documentation (https://pytorch.org/docs/) was helpful in understanding the complexities of neural network implementation and optimization, serving as a primary resource for model development and experimentation.
- Stack Overflow (https://stackoverflow.com/) and various forums provided answers to specific programming challenges, fostering a deeper comprehension of neural network nuances and best practices.
- GitHub (https://github.com/) was a valuable source of code examples and libraries, facilitating the exploration of innovative approaches and the implementation of advanced features in the neural network models.

Personal Acknowledgements:

- I would also like to express my gratitude to my instructors who provided guidance and support throughout this project. Their feedback and suggestions were crucial in overcoming challenges and enhancing the project's outcomes.

These resources, combined with the support from my academic environment, played a vital role in the successful execution of this project. I am grateful for the wealth of knowledge and expertise shared by these sources.