Recognition using Deep Networks

Chirag Dhoka Jain, Keval Visaria
Project 5 Submission
course - ***EECE5330***
{dhokajain.c, visaria.k}@northeastern.edu
April 2nd, 2024
Northeastern University

I. Introduction

In this project, we embark on an exciting journey into the realm of deep learning and computer vision. Our primary objective is to build and train a neural network capable of recognizing handwritten digits from the renowned MNIST dataset.

The project is divided into several distinct phases. Initially, we will construct and train a convolutional neural network (CNN) using the powerful PyTorch framework. This network will be meticulously designed with multiple layers, including convolutional layers, pooling layers, and fully connected layers, to effectively extract features and classify the digit images. Once the network is trained and optimized, we will save it to a file for future use. Subsequently, we will test the network's performance on a subset of the MNIST test set, evaluating its accuracy and visualizing the predictions. To further challenge the network, we will introduce handwritten digits created by ourselves, assessing its ability to generalize to new, unseen data.

In the next phase, we will explore the concept of transfer learning by adapting the trained MNIST network to recognize Greek letters. This process will involve freezing the pre-trained weights and fine-tuning the network with a new dataset containing images of alpha, beta, and gamma characters. Finally, we will delve into experimentation and optimization, systematically evaluating the impact of various architectural changes on the network's performance. This will involve modifying parameters such as the number of layers, filter sizes, dropout rates, and activation functions, among others. Through this process, we aim to gain insights into the intricate interplay between network design and performance, ultimately striving to achieve optimal results.

A. *Task 1: Build and train a network to recognize digits*

In this task, the goal was to build and train a network capable of recognizing digits from images, specifically using the **MNIST** dataset. This dataset is a collection of 70,000 images of handwritten digits (0 through 9), split into a training set of 60,000 images and a test set of 10,000 images. Each image is a 28x28 pixel grayscale representation of a digit.

To achieve this, we utilized PyTorch, a powerful library for building neural networks. The core of our approach involved creating a `convolutional` Neural Network (CNN), a type of deep learning model highly effective for image recognition tasks.

Our CNN was designed with the following components:

1. A `convolutional` layer with 10 filters of size 5x5, designed to extract basic visual features from the input images.
2. A max pooling layer with a 2x2 window, followed by a **ReLU** (Rectified Linear Unit) activation function. This layer reduces the spatial dimensions of the feature maps while introducing non-linearity to the model.
3. Another `convolutional` layer, this time with 20 filters of size 5x5, for further feature extraction.
4. A dropout layer with a 50'%' dropout rate, introduced to reduce `overfitting` by randomly omitting units from the network during training.
5. A second max pooling layer with a 2x2 window, followed again by a **ReLU** activation function.
6. A flattening operation that converts the 2D feature maps into a 1D feature vector, making it suitable for processing by fully connected layers.
7. A fully connected layer with 50 nodes, followed by a **ReLU** activation function, for learning non-linear combinations of the high-level features extracted by the `convolutional` layers.
8. A final fully connected layer with 10 nodes, each corresponding to one of the ten digit classes. The output of this layer is passed through a `log_softmax` function to produce a probability distribution over the classes.

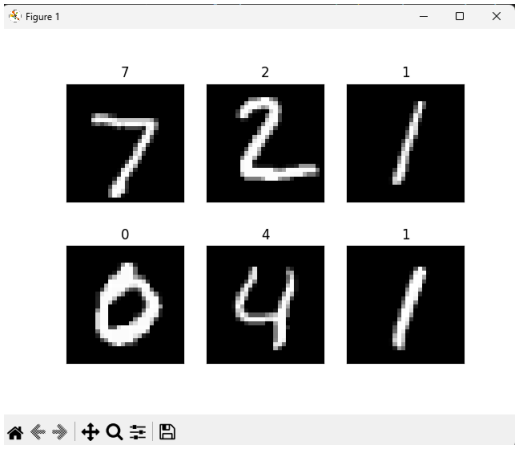

Fig. 1. Resulted output image of digits

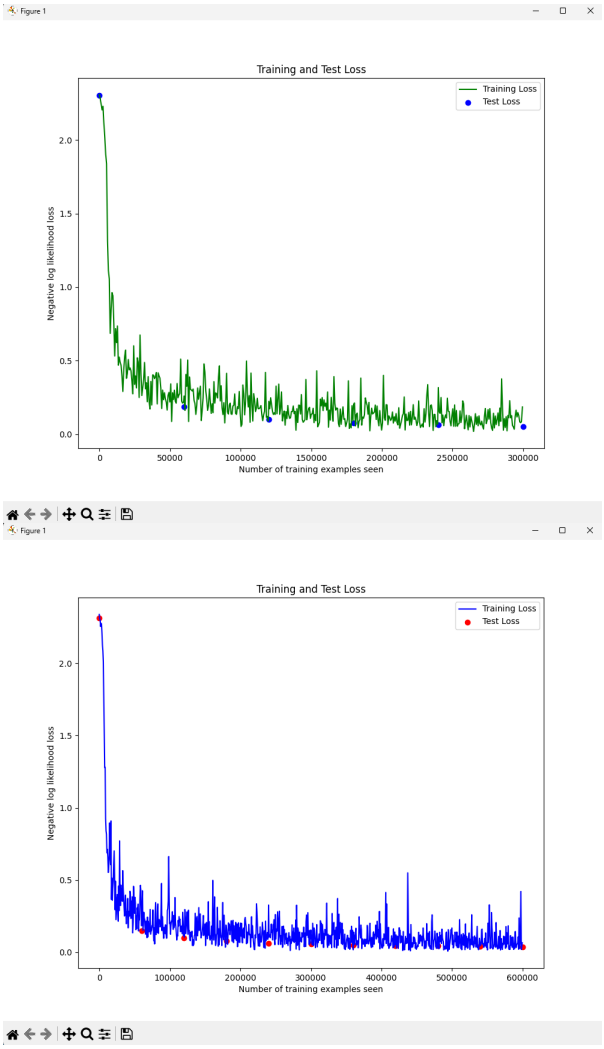

Fig. 2. Output graphs with epochs 5 and epochs 10 value

After defining the model to recognize handwritten digits using the MNIST dataset, refining its abilities over several passes through the data. With each sweep (epoch), it got better at discerning the digits. We trained it in batches of 64 images, adjusting its settings to minimize mistakes in classifying the digits. After each epoch, we tested its accuracy on both the training and test sets, watching its performance improve over time as shown in above fig 2,3. Once it reached a satisfactory level of proficiency, we saved its knowledge to a file, giving it the ability to recognize digits without needing to go through the training process again.

Here's an hand drawn network model:

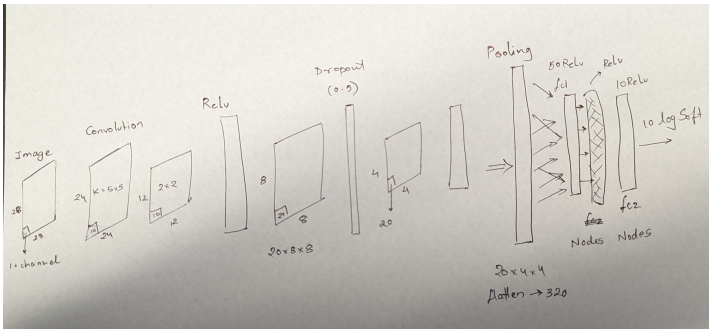

Fig. 3. Output of network model

E: Test Set

In this task, We tests a neural network on the first 10 examples of the MNIST test set, which includes handwritten digits. The network, designed for image classification, is set to evaluation mode. This mode ensures consistent behavior, as opposed to training mode, where techniques like dropout might lead to variable outputs. The process involves running each test example through the model, printing the network's output, predicted label, and the correct label. The model is expected to accurately classify most examples, with a possibility of one error. Additionally, the first 9 digits are displayed in a 3x3 grid, with each image's predicted label shown above it. This visual representation helps to quickly assess the model's performance. The task demonstrates the model's ability to recognize handwritten digits, emphasizing its reliability and effectiveness in image classification.

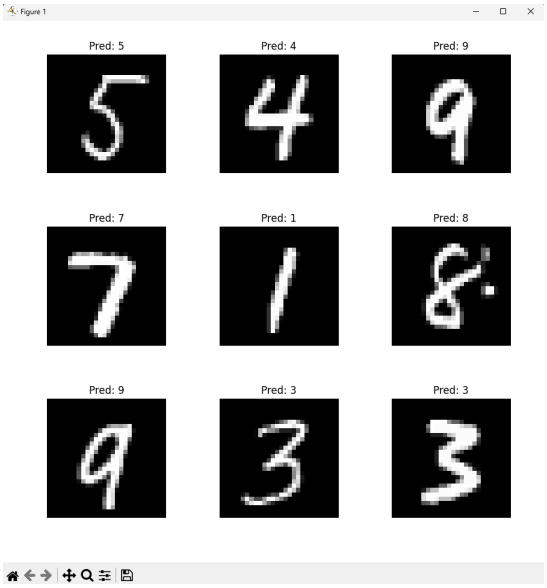

Fig. 4. Output of digits predicted and their values

```
PS C:\Users\visar\Desktop\OneDrive - Northeastern University\PRCV\RDN> & "c:/Users/visar/D
" "c:/Users/visar/Desktop/OneDrive - Northeastern University/PRCV/RDN/Task1E.py"
Example 0:
Network output: [-5.13 -5.84 -5.46 -4.19 -6.9  -0.24 -4.53 -7.95 -2.26 -2.68]
Predicted label: 5, Correct label: 5

Example 1:
Network output: [-1.145e+01 -8.710e+00 -7.170e+00 -8.620e+00 -1.000e-02 -7.390e+00
 -6.330e+00 -6.260e+00 -7.070e+00 -6.280e+00]
Predicted label: 4, Correct label: 4

Example 2:
Network output: [-4.84 -6.78 -4.99 -3.24 -3.33 -5.16 -8.07 -3.06 -2.98 -0.22]
Predicted label: 9, Correct label: 9

Example 3:
Network output: [ -6.61  -4.83  -3.73  -2.61  -9.53  -8.03 -15.56  -0.13  -4.45  -5.31]
Predicted label: 7, Correct label: 7

Example 4:
Network output: [-6.23 -0.11 -4.38 -5.2  -3.92 -6.22 -6.41 -3.3  -3.99 -4.84]
Predicted label: 1, Correct label: 1

Example 5:
Network output: [-4.69 -6.86 -4.06 -4.41 -6.28 -4.6  -3.72 -7.09 -0.08 -5.49]
Predicted label: 8, Correct label: 8

Example 6:
Network output: [-3.17 -5.2  -3.79 -3.14 -3.03 -4.25 -6.39 -2.08 -2.33 -0.51]
Predicted label: 9, Correct label: 9

Example 7:
Network output: [ -8.63  -6.92  -5.62  -0.04  -8.03  -5.88 -10.57  -5.09  -3.89  -5.59]
Predicted label: 3, Correct label: 3

Example 8:
Network output: [-10.08  -6.46  -8.05  -0.02 -10.54  -5.03 -12.39  -7.48  -5.15  -7.74]
Predicted label: 3, Correct label: 3

Example 9:
Network output: [-4.52 -6.41 -3.07 -4.53 -5.36 -5.18 -5.68 -4.67 -0.11 -4.56]
Predicted label: 8, Correct label: 8
```

Fig. 5. Output of digits predicted and their values

F: New Inputs

In this task it involves testing a trained neural network's ability to classify handwritten digits, created by the user, against the MNIST dataset's learned patterns. To do this, digits [0-9] are handwritten, captured via photograph, and each digit is isolated into its own image. These images are then converted to grayscale, resized to 28x28 pixels, and their color intensities possibly inverted to match the MNIST format (typically, the dataset has white digits on a black background).

The provided code has several critical steps:
1.`Image Preprocessing`: Each digit image undergoes conversion to grayscale, inversion (if necessary), resizing, and normalization. This ensures the input matches the network's training data format.
2. `Classification`: The model, set to evaluation mode to ensure consistent output, predicts the digit represented in each image.

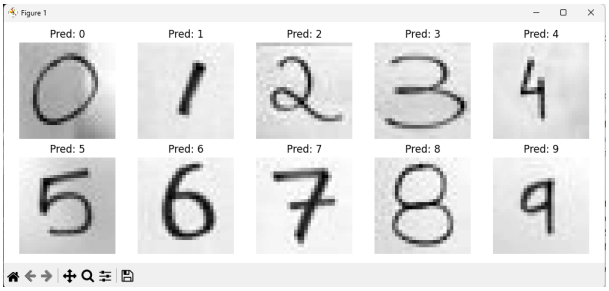

Fig. 6. Output of digits predicted with new inputs

B. Task 2: Examine your network

In this task, we focuses on exploring a pre-trained convolutional neural network (CNN) and to understand how it interprets handwritten digits from the MNIST dataset. The process involves two main steps:
1. `Examining the Network's First Layer`: The first layer's weights, or filters, are extracted and visualized. These filters are essentially the 'eyes' of the network, determining what features (edges, curves, etc.) it pays attention to in the initial stage of processing an image. The network's structure reveals that it contains ten 5x5 filters in the first layer, each designed to capture different aspects of the input images.
2. `Applying Filters to an Image`: The effects of these filters are demonstrated by applying them to a sample MNIST digit image. This step uses `openCV`, a new library introduced for this task, particularly its `filter2D` function, to apply each filter to the image. The resulting images showcase how each filter emphasizes different features of the digit, such as edges or specific patterns.

Key tips:
Layer Examination:

- Reveals the foundational structure the network uses to analyze images.

Filter Visualization:

- Offers insight into the diverse features the network can recognize.

Effect Demonstration:

- Illustrates the practical impact of filters on image perception, highlighting the network's feature extraction process.

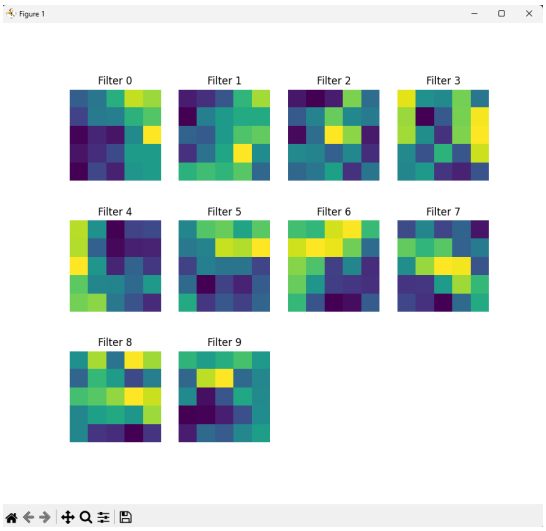

Fig. 7. Grid plot and effect on filters output

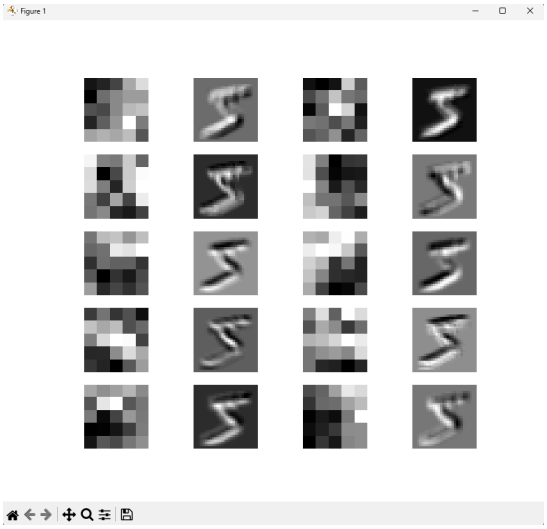

Fig. 8. Grid plot and effect on filters output

C. *Task 3: Transfer Learning on Greek Letters*

In this task it involves adapting a pre-trained neural network, originally designed for recognizing handwritten digits (0-9) from the MNIST dataset, to classify three Greek letters (alpha, beta, and gamma). The process, known as transfer learning, leverages the knowledge the network has gained from its initial training to tackle a related but distinct task.

Process Overview:

1. `Network Adaptation`: The original MNIST recognition network is modified for the new task by freezing the pre-existing learned weights. This ensures that the nuanced features learned from digits are retained. Then, the final layer of the network is replaced with a new linear layer, having three nodes corresponding to the three Greek letters. This modification tailors the network's output to the new classification task.
2. `Data Preparation`: The Greek letter images, which are colored and larger than MNIST images, undergo several preprocessing steps to fit the network's input expectations. This involves converting color images to grayscale, resizing, cropping to a 28x28 pixel size, and inverting the colors.
3. `Training and Evaluation`: With the network adapted and data prepared, the model is trained on the Greek letters dataset. The training involves adjusting only the parameters of the new final layer to classify the letters correctly. The effectiveness of this approach is measured by testing the network's accuracy on both the Greek letters dataset and additional handwritten samples provided by the user.

Transfer Learning Efficiency:

- This approach is notably efficient, requiring significantly less computational resources and training data compared to training a model from scratch.

Preprocessing Necessity:

- Adjusting the new dataset to fit the pre-trained model's input requirements is crucial for effective transfer learning. This includes image resizing, color inversion, and normalization.

Model Adaptability:

- The success of the network in recognizing Greek letters after being trained on digits underscores the adaptability of neural networks through transfer learning.

Training on Limited Data:

- Despite the small dataset of 27 Greek letter images, transfer learning allows for relatively high accuracy, showcasing the method's effectiveness in data-scarce scenarios.

Fig. 9. Training loss output graph

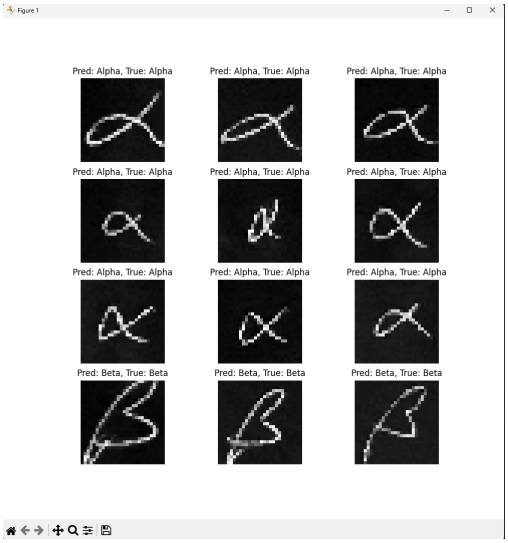

Fig. 10. Output of modified network prediction

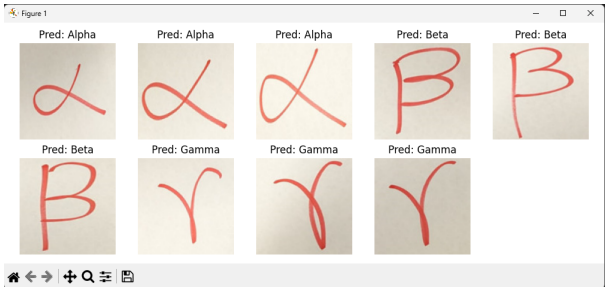

Fig. 11. Output of additional dataset prediction

D. Task 4: Calculate Current Position of the Camera

In this task, we are experimenting the impact of modifying various parameters of a convolutional neural network (CNN) architecture on its performance and training time, specifically targeting the 'Fashion MNIST' dataset. The Fashion MNIST dataset, which comprises images of clothing items, provides a more challenging classification problem than the traditional MNIST dataset of handwritten digits.

1. `Design and Execution`:
   - **Variables Explored**: Three key aspects of the network architecture were varied: the number of convolutional layers, the number of filters in these layers, and the dropout rates after the convolutional layers.
   - **Methodology**: The experiment systematically tested combinations of these variables to identify their impact on the network's accuracy and training time. A total of 64 (4 options for convolution layers * 4 options for filters * 4 options for dropout rates) configurations were evaluated.
   - **Automation**: The process was automated using a script that iterated over all possible combinations of the selected parameters, training a new model for each combination and recording its performance and training time.
2. `Key Observations`:
   - **Impact of Convolutional Layers**: Increasing the number of convolutional layers generally improved accuracy, as more complex features could be extracted. However, this also led to longer training times.
   - **Filter Numbers**: A higher number of filters in the convolutional layers allowed the network to capture more detailed features, improving accuracy up to a point. Beyond a certain threshold, the improvements plateaued, suggesting a point of diminishing returns.
   - **Dropout Rates**: Dropout rates had a nuanced effect on performance. Moderate dropout rates helped prevent overfitting, enhancing model generalizability. However, too high dropout rates hindered the model's learning capacity.
3. `Findings`:
   - **Optimization vs. Complexity**: The best-performing model struck a balance between complexity (enabling high accuracy) and training efficiency. It suggests that while adding more layers and filters can improve accuracy, there is a trade-off in terms of increased computational cost and training time.
   - **Dropout Rate Sweet Spot**: There exists an optimal dropout rate that maximizes accuracy while minimizing overfitting, highlighting the importance of tuning regularization techniques according to the specific dataset and model architecture.

4. `Conclusions and Predictions`:

   - **Predictions**: It was hypothesized that increasing the network's complexity would improve accuracy, which was largely supported. However, the impact of dropout rates was less predictable and underscored the importance of empirical testing.
   - **Best Configuration**: The experiment identified an optimal set of parameters that offered the best trade-off between accuracy and training time. This configuration can serve as a starting point for further fine-tuning or as a benchmark for comparing alternative architectures or datasets.

Below plot helps in understanding how well the model is learning from the training data and generalizing to unseen validation data, I had just plotted to get some figure that it

Fig. 12. Plot

The best result after 64 iterations: Best Configuration: Num Conv: 2, Num Filters: 128, Dropout Rate: 0.5, **Accuracy: 0.9242**

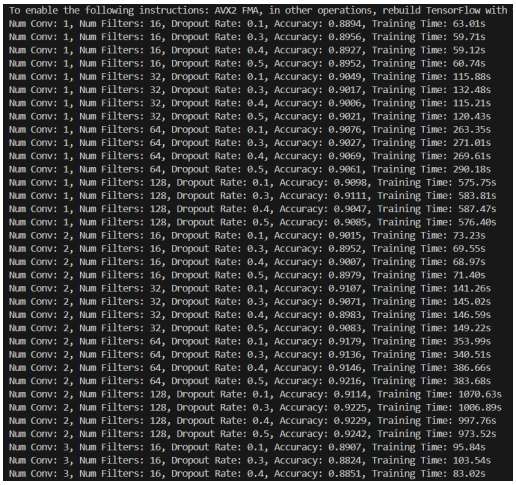

Fig. 13. Output of iterations

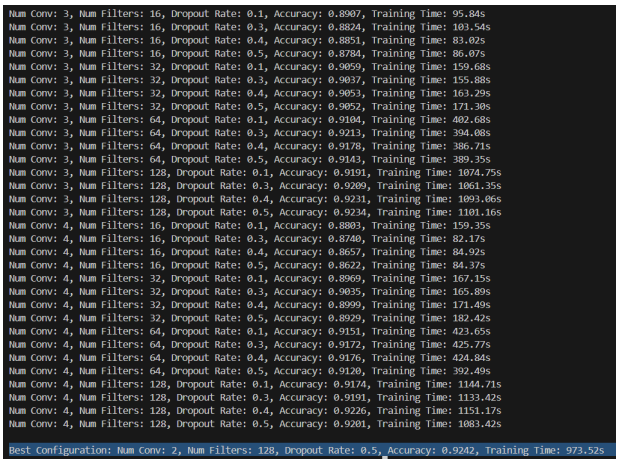

Fig. 14. Output of iterations

For each configuration, the model is trained, and its accuracy is measured. The training process involves adjusting the weights within the network to minimize the loss function, which measures the difference between the predicted and actual values. The dropout acts during training by randomly zeroing out the outputs of some neurons, forcing the network to spread out the learning and potentially reducing over-reliance on any specific neuron.

Ultimately, the ideal model configuration is the one that achieves the highest accuracy on the validation or test set, indicating that it has learned well from the training data and can generalize well to new, unseen data. This configuration is then chosen as the "best" one, although in practice, one might also consider the training time and the complexity of the model before making a final decision.

E. Extensions

1) VGG16: Pre-trained networks

In this task, we're using a pre-trained VGG16 model, a renowned convolutional neural network known for its prowess in image classification tasks. By focusing on the model's initial convolutional layers, we gain insights into how VGG16 processes and interprets image data at the foundational level.
`Approach`:
1. `Model Preparation`: We start by loading the VGG16 model pre-trained on 'ImageNet', setting it to evaluation mode to ensure consistent behavior during inference.
2. `Filter Visualization`: The weights of the first few convolutional layers are visualized. These weights, or filters, are crucial in detecting various features in the input images, such as edges and textures. Visualizing these filters helps us understand the types of patterns that the network is sensitive to in its early stages.
3. `Image Transformation and Filtering`: To demonstrate the filters' effects, an image from the CIFAR10 dataset is resized to fit VGG16's input requirements and then processed through one of the first layer's filters. This process highlights how the selected filter modifies the image, emphasizing certain features while de-emphasizing others.
Observations: The visualized filters reveal diverse patterns, each designed to capture specific aspects of the image data. The filtered image showcases the practical application of these filters, illustrating how they isolate and enhance features within the image.

This exploration underscores the complexity and effectiveness of VGG16's convolutional layers in feature extraction. Through filter visualization and practical application, we gain a clearer understanding of the initial steps in the network's image processing pipeline, illuminating the foundational mechanisms driving its classification capabilities.

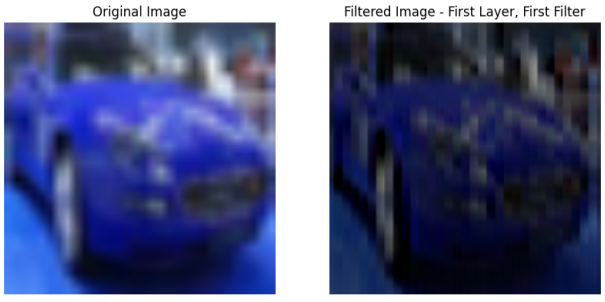

Fig. 15. Output images using VGG16 network filters

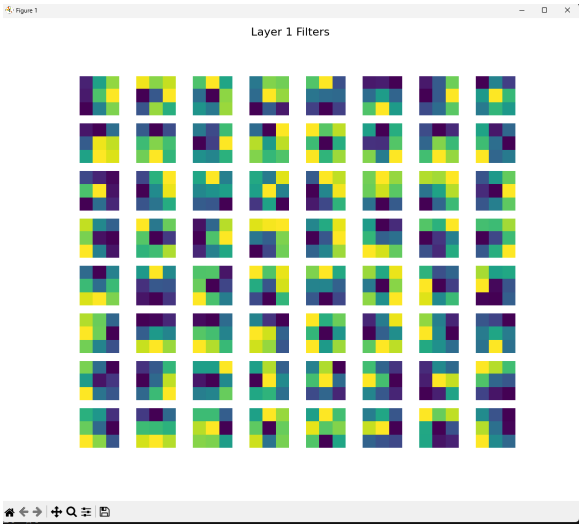

Fig. 16. Output images using VGG16 network filters

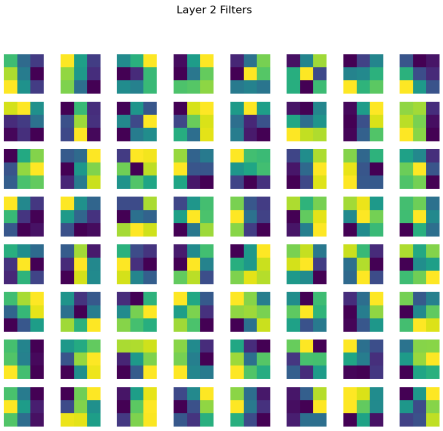

Fig. 17. Output images using VGG16 network filters

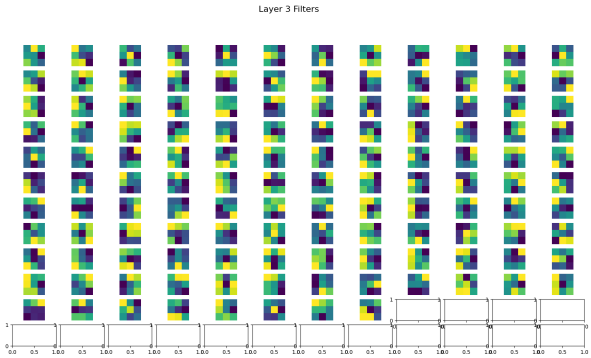

Fig. 18. Output images using VGG16 network filters

2) *More greek letters*

In this task, we aimed to extend a neural network, originally trained to recognize MNIST digits, to classify additional Greek letters beyond the initial set of alpha, beta, and gamma. By integrating letters like delta, eta, and theta, the task complexity increased due to the expanded classification scope.

`Implementation`:

1. `Model Adaptation`: We began by selecting an existing CNN model, either the original or a modified version, based on user input. The model's final layer was adjusted to output six categories, corresponding to the six Greek letters, to accommodate the new classification task.
2. `Dataset Preparation`: A custom transformation pipeline was implemented for the Greek letters dataset, which involved converting images to grayscale, resizing, and cropping to match the MNIST format, and inverting colors to ensure consistency with the training data.
3. `Training and Evaluation`: With the model and dataset prepared, the network underwent training on the extended Greek letters dataset. After training, the model's ability to accurately classify both the extended set of Greek letters and additional handwritten samples was evaluated.

Outcomes:

- The adapted model demonstrated effective classification across the broader set of Greek letters, highlighting the flexibility and applicability of CNNs in accommodating new, related tasks through minor adjustments and targeted training.
- Visual inspection of the model's predictions on test images and handwritten samples provided tangible evidence of its generalization capabilities, underscoring the practical value of neural network adaptability in real-world applications.

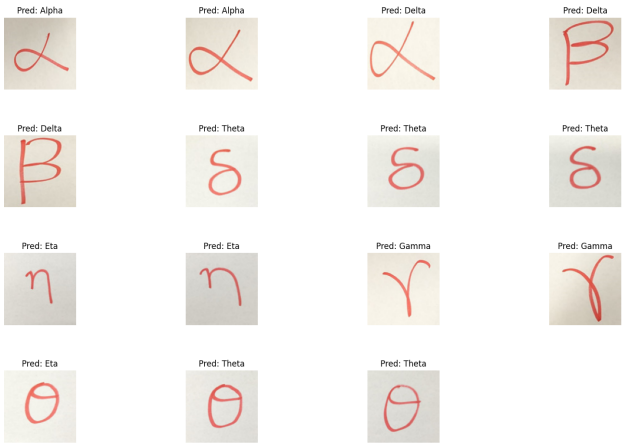

Fig. 19. Output image of predicted greek letters

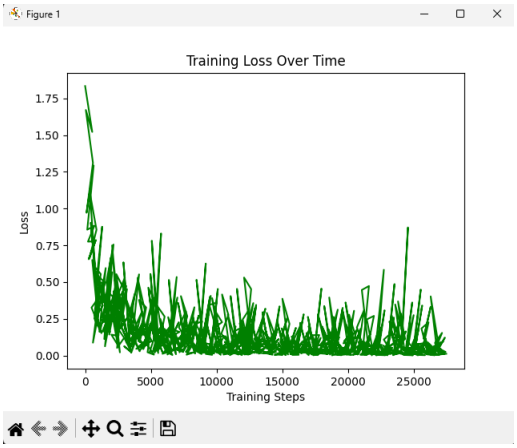

Fig. 20. Output image training loss plot

*3) **Live video digit recognition***

In this task, It enables real-time digit recognition via a camera using OpenCV for video capture and processing, and a convolutional neural network (CNN) for digit identification. It allows choosing from different CNN models, notably a Gabor-filter-based CNN for advanced feature detection. Video frames captured are first turned to grayscale and then binary thresholded to highlight digits. The script identifies and isolates the largest contour, presumed to be a digit, using a mask. This isolated digit is resized to 28x28 pixels, color-inverted to match training data standards, and transformed into a tensor for the CNN.

The CNN, here exemplified by a Gabor-filter-based model for its efficiency in feature extraction, receives the pre-processed image to predict the digit. The prediction is then displayed on the live video feed. The process repeats, providing continuous digit recognition, until the program is exited with a "q" press. This seamless integration of image processing and deep learning facilitates the real-time digit recognition from video streams

- Live demo video 1 of working system
  Live demo video 2 of working system

*4) **Replace the first layer: Gabor filter***

In this task, we use Gabor filters into the first layer of a convolutional neural network (CNN) designed for digit recognition on the MNIST dataset. By incorporating these filters, the network aims to enhance its feature extraction capabilities, especially for varying orientations and frequencies of digit strokes.

The process begins by generating a set of Gabor filters, each with a distinct orientation, to replace the standard kernels in the first convolutional layer of the network. This initial layer is then frozen, ensuring that these specially designed filters remain unchanged during training, allowing the network to learn higher-level features based on this optimized input. The rest of the network comprises additional convolutional, dropout, and fully connected layers, tasked with classifying the digits based on the features extracted through the Gabor-enhanced first layer.

Training involves feeding the network batches of MNIST digit images, pre-processed to suit the model's input requirements. The model learns to classify digits over multiple epochs, adjusting its parameters (excluding the Gabor filters) to minimize classification errors. The goal is to assess whether the introduction of Gabor filters improves the network's ability to recognize digits compared to a standard CNN setup. This approach combines traditional image processing techniques with modern deep learning to potentially achieve more accurate and robust digit recognition.

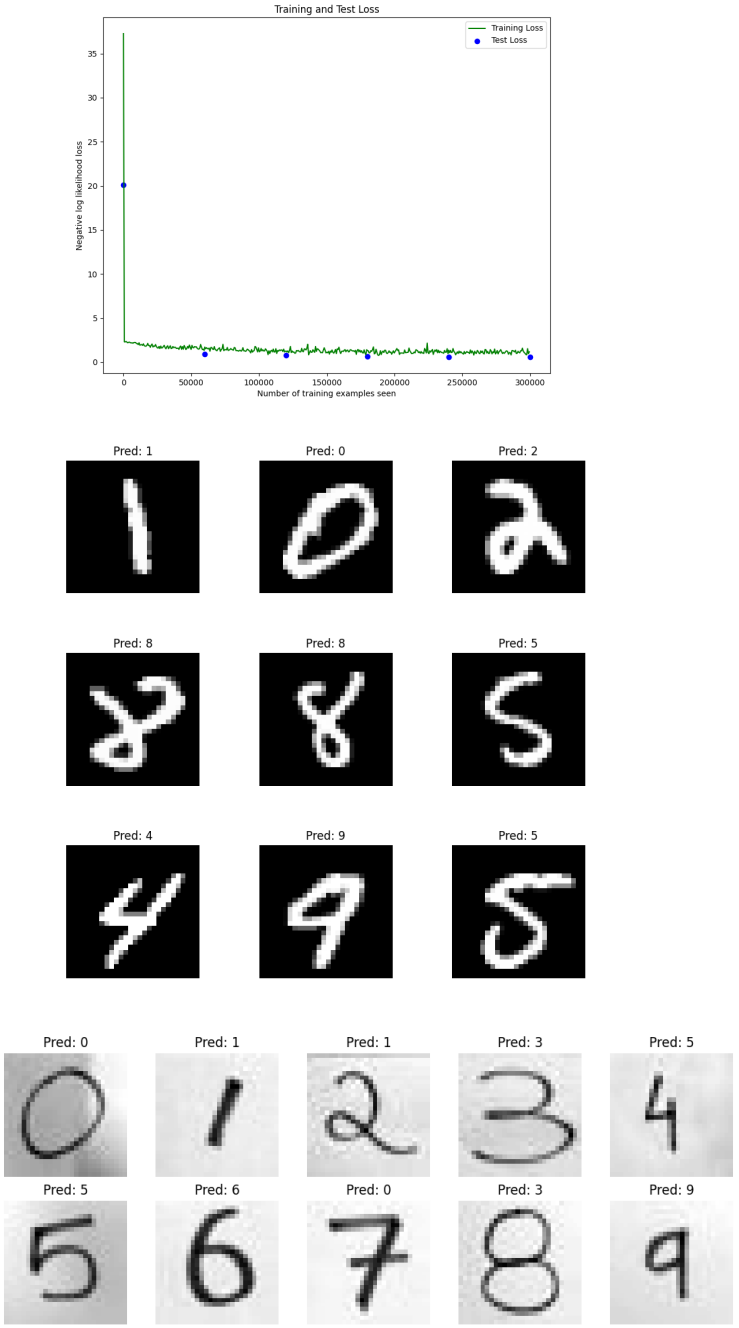

Fig. 21. Output images replacing the first layer with gabor

II. WHAT WE LEARNED!!!

- Deep Learning Fundamentals: Delved into the principles of deep learning, understanding how neural networks function, and the significance of architecture design in achieving accurate predictions.
- Model Training and Optimization: Acquired hands-on experience in training convolutional neural networks (CNNs) using PyTorch, focusing on optimizing hyper-parameters and fine-tuning to enhance performance.
- Transfer Learning Application: Explored the concept of transfer learning by adapting a pre-trained CNN from recognizing MNIST digits to classifying Greek letters, demonstrating the versatility of neural networks in tackling diverse tasks with minor adjustments.
- Architectural Experimentation: Conducted systematic experimentation on various architectural parameters, such as layer configurations, filter sizes, and activation functions, to understand their impact on network performance and gain insights into optimizing model design for specific tasks.

III. ACKNOWLEDGEMENT

Acknowledging the wealth of knowledge available online, particularly in the 'OpenCV' documentation and tutorials, played a crucial role. This project's success is a testament to the collective effort of the educational community, emphasizing the shared wealth of expertise. Interactions with ChatGPT proved invaluable for exploring concepts and troubleshooting, enhancing the overall learning experience of the project. This acknowledgment is a tribute to the collaborative and supportive environment that significantly facilitated the learning journey in this project.