Atharva Nayak **002322653**

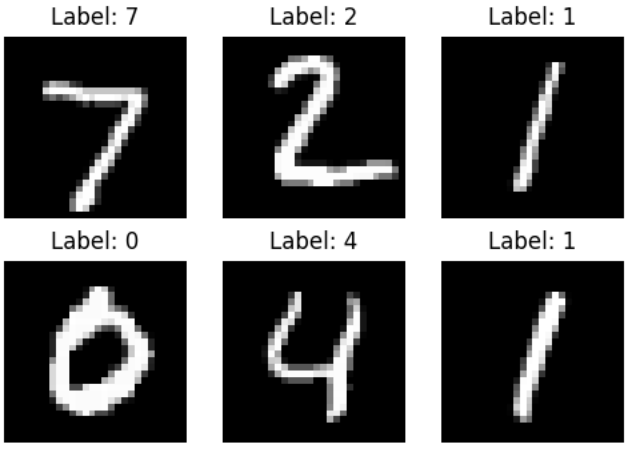
Project 5: Recognition using Deep Networks

In this project, we're taking a hands-on dive into deep learning by building, training, and exploring a neural network that can recognize handwritten digits. We'll work with the famous MNIST dataset using PyTorch to create a convolutional network that layers in convolutions, pooling, dropout, and fully connected nodes to classify the digits. Along the way, you'll learn everything from data preprocessing and model construction to training and visualizing both the network's performance and its inner workings. We don't stop at just digit recognition—we push further by adapting the network to identify Greek letters through transfer learning and by experimenting with different network configurations to see how tweaks can impact performance and training speed. Overall, this project is designed to help you build essential deep learning skills while also encouraging you to explore, experiment, and even brainstorm ideas for your own final project.

Task 1

A. Get the MNIST digit data set

The MNIST dataset, which contains 10,000 28x28 test images and 60,000 training images of handwritten numbers, was imported straight from the torchvision package for this purpose. To guarantee consistency, the test set was maintained in its original order while the dataset was imported into DataLoader objects. Using the subplot feature of matplotlib, the first six test set photographs were shown in a 2x3 grid with labels attached to each image. In addition to verifying that the data was loaded correctly, this visualization phase offers a preliminary understanding of the digit samples that were used for training and assessment.

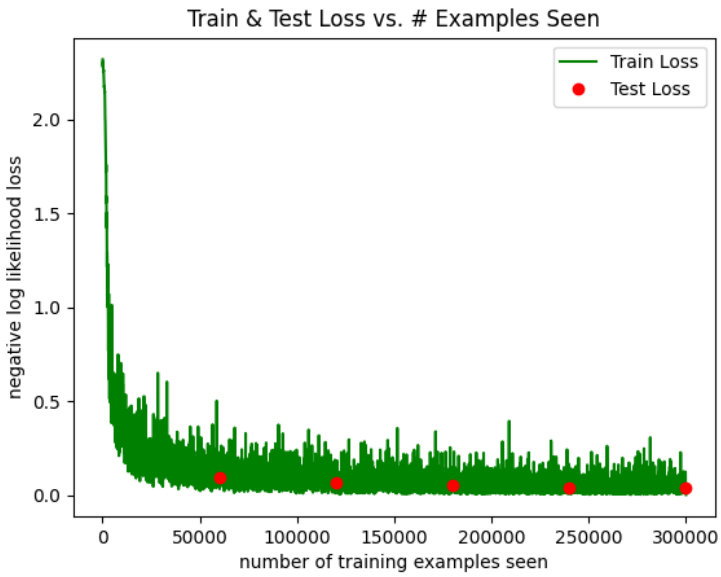


B. Build a network model

The network starts with a convolutional layer that uses 10 filters of size 5×5 to capture basic patterns in the input images, and then a max pooling layer with a 2×2 window and ReLU activation is applied to emphasize the most important features while reducing the image size. Next, a second convolutional layer with 20 filters further refines these features, followed by a dropout layer (with a chosen rate between 5% and 50%) to help prevent overfitting. Another max pooling layer with ReLU activation then reduces the dimensionality even more. The output is then flattened into a one-dimensional vector, which is passed through a fully connected layer with 50 neurons using ReLU activation to combine the features. Finally, a last fully connected layer produces 10 output values corresponding to each digit, with a log_softmax function applied to convert these into log-probabilities.

C. Train the model

In this task, the model was trained for five full epochs, where each epoch represents one complete pass through the entire training dataset. A batch size of 64 was used to efficiently process the data in smaller chunks. During training, the model's performance was tracked by recording the training loss after every batch and the test loss once at the end of each epoch. The training loss was calculated using the negative log likelihood loss function, and the Adam optimizer was used to update the model weights. After all five epochs, the recorded losses were plotted on a graph with the number of training examples seen as the x-axis. The graph shows the continuous drop in training loss along with test loss points for each epoch, helping visualize how well the model was learning over time. This analysis provides valuable insight into the model's convergence behavior and generalization to unseen data. The final trained model was saved for use in future tasks.

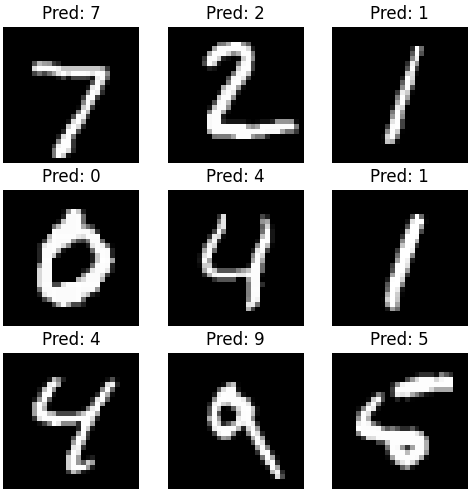


D. Save the Model

The model is successfully saved.

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

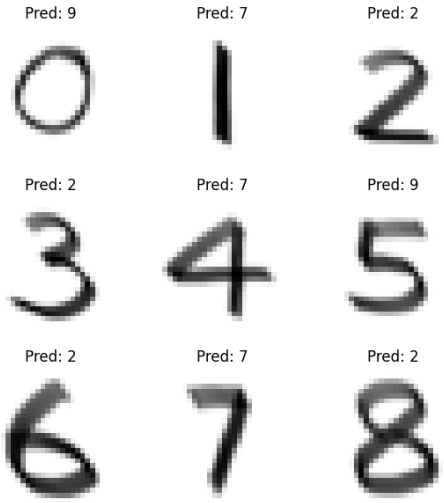
In this task, a separate Python script was used to evaluate the trained model on the first ten samples from the MNIST test set. Before making predictions, the model was switched to evaluation mode using model.eval(). This step is essential because it ensures the dropout layer behaves consistently, allowing for reliable and repeatable outputs. The model then processed the first ten test images, printing the predicted probabilities for each of the ten digit classes, rounded to two decimal places. Alongside the predicted class (i.e., the index of the highest probability), the actual label was also printed for comparison. In addition to the printed values, the first nine images were displayed in a 3×3 grid with their predicted digits labeled above each image. The model performed well, correctly classifying nearly all of the samples. This task confirmed the model's ability to generalize to unseen data and provided a visual and numerical validation of its accuracy.




```
PS C:\Users\nayak\PRCV_Projects\project_5> & "C:/Program Files/Python313/python.exe" c:/Users/nayak/PRCV_Projects/project_5/eval_MNIST.py
  Image 0: ['0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '1.00', '0.00', '0.00']  Pred=7  True=7
  Image 1: ['0.00', '0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']  Pred=2  True=2
  Image 2: ['0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']  Pred=1  True=1
  Image 3: ['1.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']  Pred=0  True=0
  Image 4: ['0.00', '0.00', '0.00', '0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00']  Pred=4  True=4
  Image 5: ['0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']  Pred=1  True=1
  Image 6: ['0.00', '0.00', '0.00', '0.00', '1.00', '0.00', '0.00', '0.00', '0.00', '0.00']  Pred=4  True=4
  Image 7: ['0.00', '0.00', '0.00', '0.00', '0.02', '0.00', '0.00', '0.00', '0.00', '0.98']  Pred=9  True=9
  Image 8: ['0.00', '0.00', '0.00', '0.00', '0.00', '0.99', '0.01', '0.00', '0.00', '0.00']  Pred=5  True=5
  Image 9: ['0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.02', '0.00', '0.98']  Pred=9  True=9
```

F. Test the network on new inputs

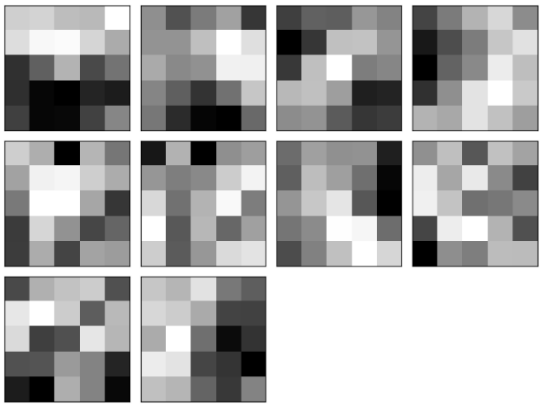
For this task, I tested the trained model on a custom set of handwritten digits created by writing the numbers 0 through 9 using thick strokes on a white background. Each digit was then individually cropped, converted to grayscale, resized to 28×28 pixels, and preprocessed to match the format of the MNIST dataset. This included inverting the pixel intensities, as MNIST digits are white on a black background, and applying the same normalization used during training. The images were then passed through the trained model in evaluation mode, and the predicted probabilities for each digit were printed alongside the true labels. A 3×3 grid of the first nine images with their predicted digits was also generated and saved. The predictions and the actual digits are given in the figure below.



Task 2

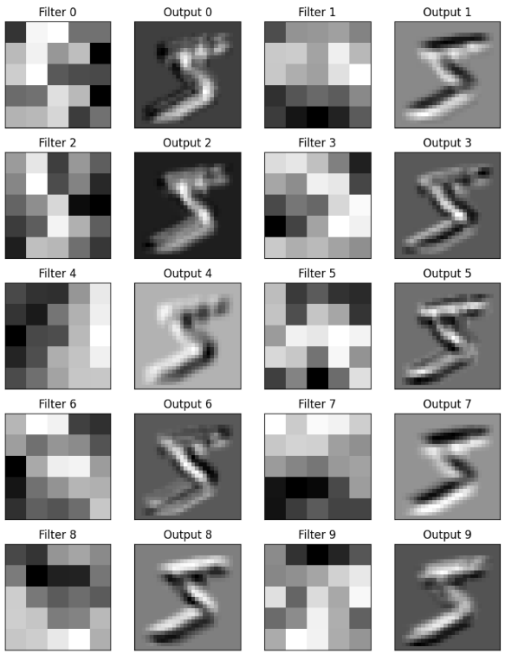
A. Analyze the first layer

In this task, the focus was on analyzing and visualizing the filters from the first convolutional layer of the trained network. The layer, named conv1, contains ten 5×5 filters, each designed to detect different low-level features from the input images. The filter weights were extracted and displayed in a 3×4 grid using matplotlib. Each filter revealed a unique pattern of values, which likely corresponds to detecting edges, textures, or specific strokes in handwritten digits. Some filters appear to highlight vertical or horizontal transitions similar to Sobel filters, while others may focus on corners or diagonal lines.



B. Show the effect of the filters

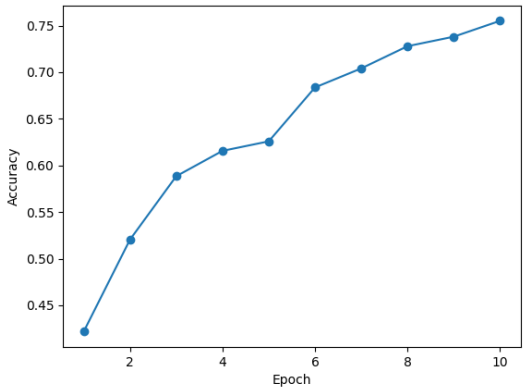
In this task, the filters from the first convolutional layer of the trained network were applied to an actual MNIST digit image to observe how each filter responds to different visual features. By running the image through each of the ten filters, we generated a set of transformed outputs that highlight different characteristics of the digit, such as edges, corners, and directional strokes. The resulting images were plotted in a grid to visually compare how each filter interprets the same input. These filtered outputs clearly show that different filters focus on different parts or shapes within the digit, which aligns with how convolutional neural networks learn to detect simple patterns in early layers.

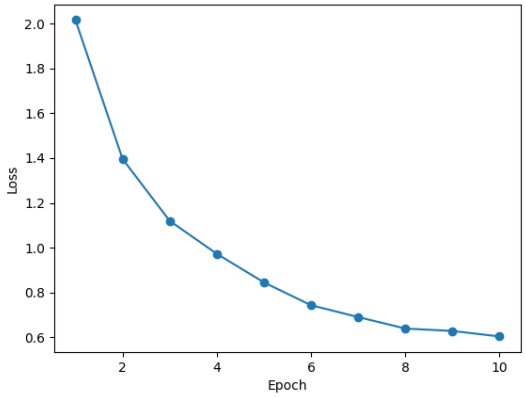


Task 3

Transfer Learning on Greek Letters

In this task, transfer learning was applied to adapt the pre-trained MNIST digit classifier to recognize three Greek letters: alpha, beta, and gamma. The existing neural network was reused by loading the pre-trained weights and freezing all layers except the final fully connected layer, which was replaced with a new layer tailored to classify three classes. A small dataset of Greek letters was preprocessed to match the MNIST format, including converting to grayscale, resizing to 28×28 pixels, and inverting pixel intensities. The model was trained over several epochs using this new data, and performance was tracked by plotting training loss and accuracy across epochs. As shown in the graphs, the loss consistently decreased while the accuracy improved, indicating the model successfully learned to distinguish between the Greek letters. This demonstrates the effectiveness of transfer learning, especially when working with limited data, and shows how a well-trained model can be repurposed for related classification tasks with minimal adjustments.





Task 4

Design your own experiment

In the final phase of the project, an extensive experiment was conducted to explore how different hyperparameters impact the performance and training time of a convolutional neural network on the FashionMNIST dataset. The goal was to evaluate how changes in architecture and training configurations such as the number of filters, dropout rate, batch size, and number of epochs affect accuracy and efficiency. A grid search approach was implemented to automate the process, varying one parameter at a time while holding others constant.

Before running the experiments, several hypotheses were formulated. It was expected that increasing the number of filters would improve accuracy due to better feature extraction but would also increase training time. Similarly, smaller batch sizes were hypothesized to yield slightly better accuracy due to more frequent weight updates, while larger batch sizes were expected to train faster. For dropout, moderate values were anticipated to improve generalization, but higher dropout rates might hinder learning. Finally, increasing the number of epochs was expected to improve accuracy at the cost of longer training times.

The experimental results confirmed many of these expectations. As shown in the analysis plots, increasing the number of filters generally improved accuracy, but also led to longer training durations. Accuracy improved with more epochs, while training time increased proportionally. Smaller batch sizes slightly improved performance but resulted in longer training times. Interestingly, dropout values around 0.25 achieved the best balance between regularization and learning efficiency, while too high a dropout rate negatively impacted accuracy. These insights provide valuable guidance on how to tune CNN hyperparameters for optimal performance, balancing accuracy and computational cost.

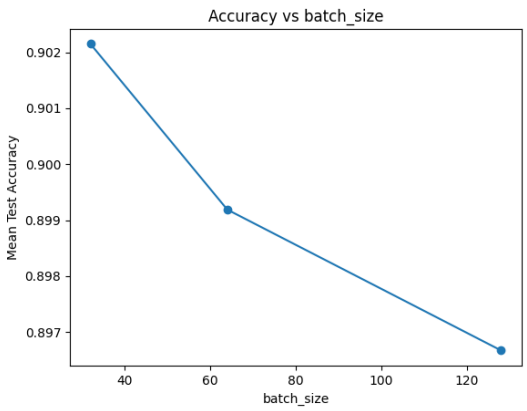


Fig: Accuracy when batch size is varied

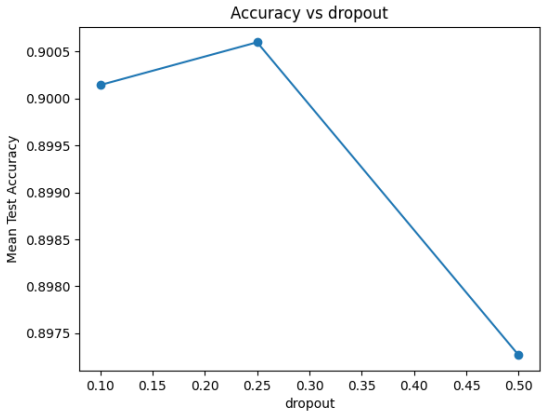


Fig: Accuracy when dropout rate is varied

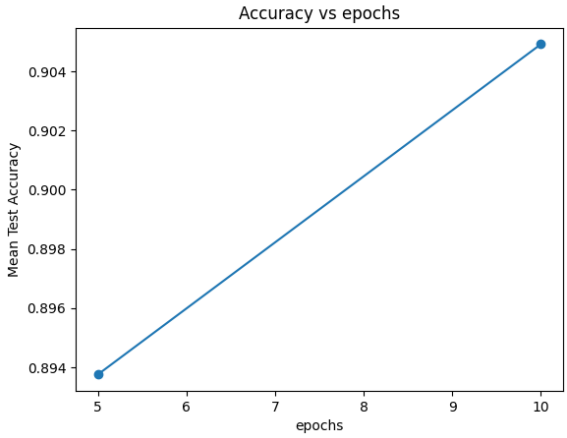


Fig: Accuracy when epochs are varied

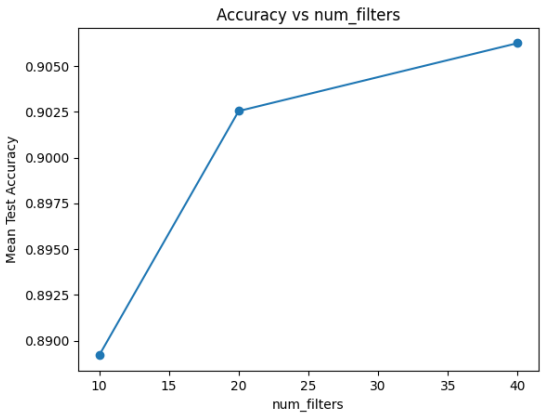


Fig: Accuracy when num of filters are varied

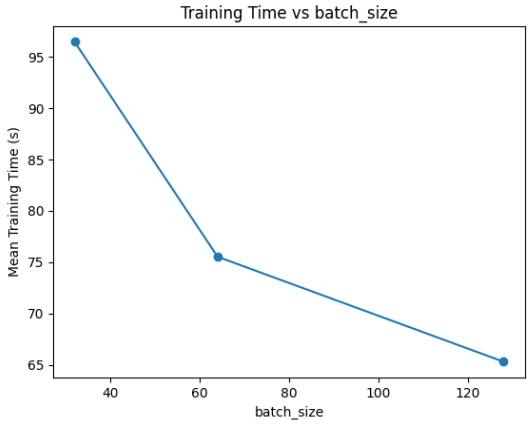


Fig: Training time when batch size is varied

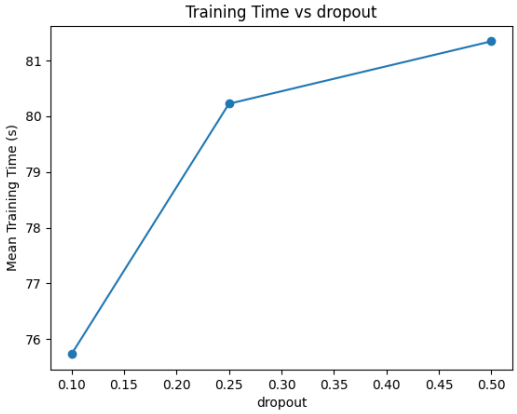


Fig: Training time when dropout rate is varied

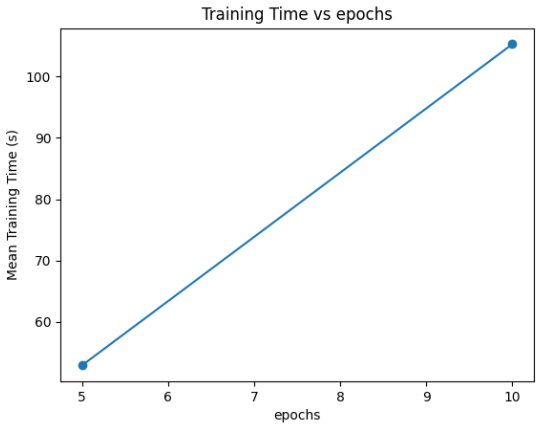


Fig: Training time when epochs are varied

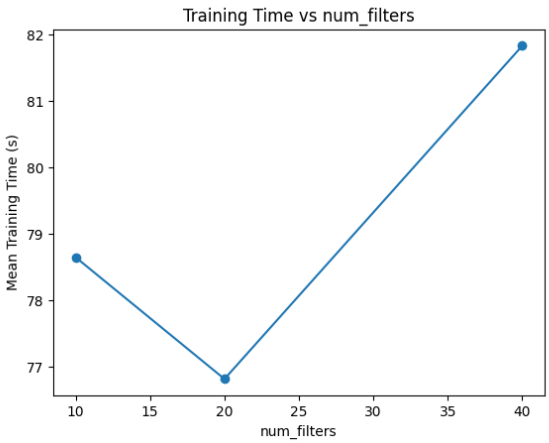


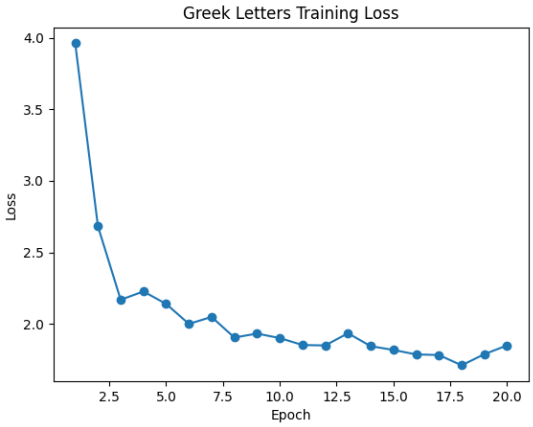
Fig: Training time when num of filters are varied

Extension

Trained the model for more Greek letters other than alpha, beta and gamma.

As an extension to the original transfer learning task, this experiment expanded the classification problem from three Greek letters (alpha, beta, gamma) to multiple additional Greek characters by adapting the pre-trained MNIST digit recognition model. Instead of training the entire network from scratch, only selected layers—specifically the second convolutional layer and the fully connected layers—were unfrozen and retrained. The model's final classification layer was replaced to accommodate the total number of detected Greek letter classes in the dataset. To improve generalization, data augmentation techniques such as random rotations and translations were applied during preprocessing.

The training results for the extended Greek letter classification task show a clear downward trend in the loss over 20 epochs, indicating that the model successfully minimized the error during training. At the same time, the accuracy gradually improved, reaching just above 30%. While this accuracy is relatively modest, it reflects the increased complexity of the task due to the larger number of classes and the limited dataset size. The upward trajectory of the accuracy curve, despite some fluctuation, suggests that the model is learning to differentiate between the expanded set of Greek letters. These results also highlight the value of transfer learning with partial fine-tuning, which allows the model to adapt previously learned features to a new classification task with minimal training and without starting from scratch.



Atharva Nayak **002322653**

Reflection

This project provided a hands-on experience through the core concepts of deep learning, starting from building and training a convolutional neural network for digit recognition to extending the model through transfer learning and experimentation. It helped me understand how each layer contributes to feature extraction and decision-making, and how small architectural or training changes can significantly affect performance. The transfer learning task was particularly insightful, as it showed the value of leveraging pre-trained models for new tasks with limited data. Overall, this project strengthened my confidence in using PyTorch, deepened my understanding of CNNs, and sparked curiosity to explore more advanced applications in computer vision.

Acknowledgements

I would like to express my sincere gratitude to my instructor for their guidance and support throughout this project. I also appreciate the valuable resources and tutorials provided, which helped me better understand the concepts of deep learning and PyTorch.