Lab5 Report

Wu Han

January 1st, 2025

1 Experiment Overview

This experiment introduces fundamental knowledge related to PyTorch, a powerful machine learning library, and how neural networks are built and trained. Neural networks are the backbone of deep learning, which is an essential technology in the modern era of artificial intelligence.

In this experiment, we will be constructing neural networks to perform two key tasks: train a simple model and image classification. We will also explore the concept of convolutional neural networks (CNNs) and apply them for image feature extraction.

1.1 PyTorch and Neural Network Construction

A neural network can be simply understood as a mathematical model designed to approximate complex functions through layers of interconnected neurons. The goal of machine learning is to find the function f that best describes the relationship between input data and expected output. In a neural network, the structure is fixed, but the model's weights between nodes are the trainable parameters.

For the tasks in this experiment, we will:

1. Try to train a model based on CIFAR-10 dataset
2. Explore the basic steps to train a model and how to evaluate a model

1.2 Using Convolutional Neural Networks for Image Feature Extraction and Image Search

Early image retrieval systems were based on text descriptions, known as text-based image retrieval (TBIR), where images were indexed using textual annotations.

However, as the field of artificial intelligence progressed, content-based image retrieval (CBIR) emerged, which allows users to search for images based on their content rather than descriptions. This system has become ubiquitous on many online platforms.

The core task in CBIR is to find the most similar images to a query image from a large image database. Deep learning models, especially convolutional neural networks (CNNs), have revolutionized this task by learning to extract discriminative features from images automatically.

CNNs are a class of deep neural networks that include convolutional layers, which are particularly well-suited for image data. These networks learn hierarchical representations of image features, such as edges, textures, and higher-level structures, enabling accurate classification and retrieval.

In this experiment, we will introduce a method for image search by extracting the features of the image using CNN and then search the image by compare the similarties between the features.

2 Task 1: Simple Model Training Based On CIFAR-10

Although the teaching assistant has already written the framework of the code in exp2.py, I have already done some deep learning tasks myself and had my own coding style, and I want to plot the curves of the entire process, so instead I write the whole python script myself.

2.1 Code Explanation

This function adjusts the learning rate dynamically based on the current epoch during training. As training progresses, the learning rate decreases to allow the model to converge more accurately:

$$\text{If epoch} = 5, \text{ set learning rate to } 0.01$$

$$\text{If epoch} = 10, \text{ set learning rate to } 0.001$$

```
    def adjust_learning_rate(optimizer, epoch):
        if epoch == 5:
            for param_group in optimizer.param_groups:
                param_group['lr'] = 0.01
            print('Learning rate adjusted to 0.01')
        elif epoch == 10:
            for param_group in optimizer.param_groups:
                param_group['lr'] = 0.001
            print('Learning rate adjusted to 0.001')
```

This function evaluates the model's performance at the end of each epoch. The model is tested on the test dataset to calculate the test loss and accuracy: - First, the model is set to evaluation mode (`model.eval()`), disabling dropout and other regularization strategies to ensure stable inference. - The function uses the `torch.no_grad()` context manager to disable gradient calculation, saving memory. - It then loops through the test dataset, performs predictions for each batch, calculates the loss, and computes accuracy. Finally, the function prints the average loss and accuracy for that epoch:

$$\text{Test Loss} = \frac{\sum(\text{loss}) \times (\text{batch size})}{\text{total samples}}$$

$$\text{Test Accuracy} = \frac{\text{correct predictions}}{\text{total samples}} \times 100$$

```
# Testing function
def test(epoch):
    print('==> Testing...')
    model.eval()
    test_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for batch_idx, (inputs, targets) in enumerate(
            testloader):
            inputs, targets = inputs.to(device), targets.to(
                device)
            outputs = model(inputs)
            loss = criterion(outputs, targets)

            test_loss += loss.item() * inputs.size(0)  #
                Accumulate loss
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()

    avg_loss = test_loss / total
    acc = 100. * correct / total
    print('Epoch [%d] Test Loss: %.3f | Test Acc: %.3f%% (%d/%
        d)'
          % (epoch, avg_loss, acc, correct, total))
```

This part of the code uses `matplotlib` to plot the loss and accuracy trends during training: - The left subplot shows the training and test loss over time. - The right subplot shows the training and test accuracy over time. The function uses `plt.tight_layout()` to automatically adjust the spacing between subplots to prevent overlap. Finally, the figure is saved as a PNG file, and the file path is printed.

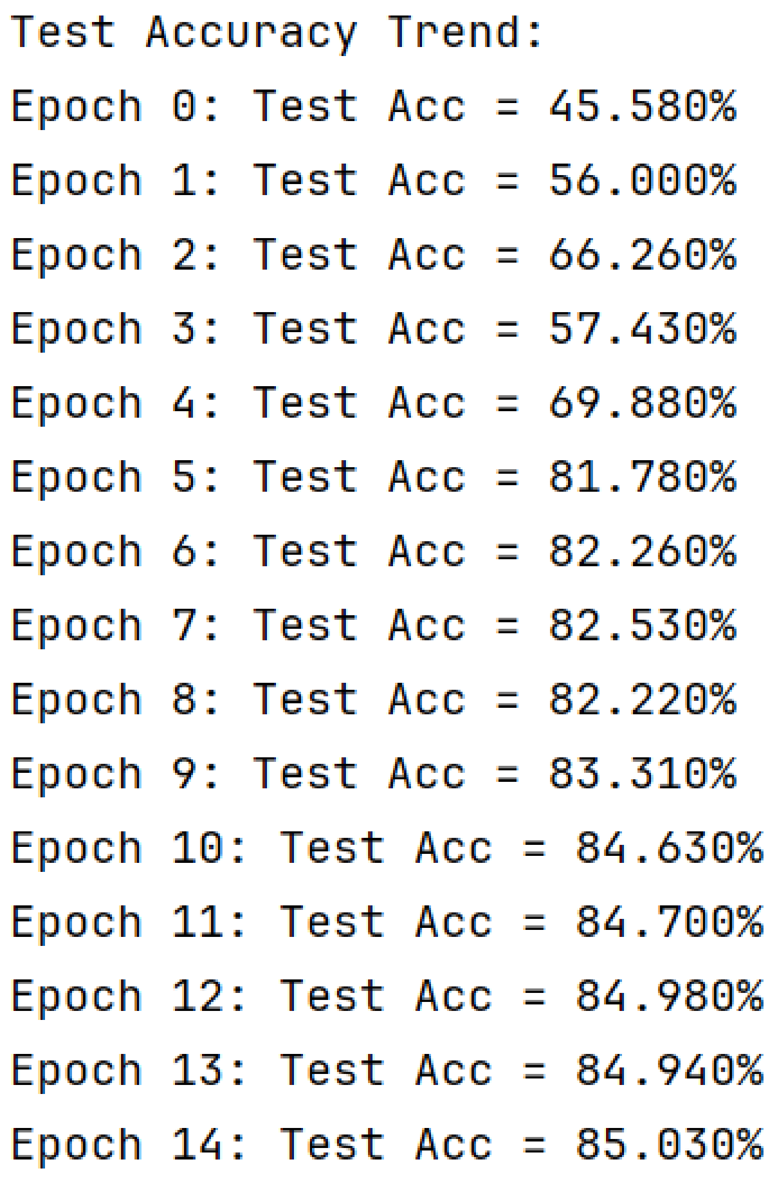
```
    plt.figure(figsize=(12, 5))

    # Plot Loss
    plt.subplot(1, 2, 1)
    plt.plot(epochs, train_losses, 'bo-', label='Training Loss')
    plt.plot(epochs, test_losses, 'ro-', label='Test Loss')
    plt.title('Training and Test Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    # Plot Accuracy
    plt.subplot(1, 2, 2)
    plt.plot(epochs, train_accuracies, 'bo-', label='Training
       Accuracy')
    plt.plot(epochs, test_accuracies, 'ro-', label='Test Accuracy'
       )
    plt.title('Training and Test Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy (%)')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plot_path = './training_curves.png'
    plt.savefig(plot_path)
    print(f'Training curves saved to {plot_path}')
    plt.show()

    # Print test accuracy trend
    print('\nTest Accuracy Trend:')
    for epoch, acc in enumerate(test_accuracies, start=start_epoch
       ):
        print('Epoch %d: Test Acc = %.3f%%' % (epoch, acc))
```

2.2 Training Results Analysis

The matching effect is relatively good.

The specific values of testing accuracy are shown in Figure1:

The training curve are shown in Figure2:



```
Test Accuracy Trend:
Epoch 0: Test Acc = 45.580%
Epoch 1: Test Acc = 56.000%
Epoch 2: Test Acc = 66.260%
Epoch 3: Test Acc = 57.430%
Epoch 4: Test Acc = 69.880%
Epoch 5: Test Acc = 81.780%
Epoch 6: Test Acc = 82.260%
Epoch 7: Test Acc = 82.530%
Epoch 8: Test Acc = 82.220%
Epoch 9: Test Acc = 83.310%
Epoch 10: Test Acc = 84.630%
Epoch 11: Test Acc = 84.700%
Epoch 12: Test Acc = 84.980%
Epoch 13: Test Acc = 84.940%
Epoch 14: Test Acc = 85.030%
```

Figure 1: Test Accuracy

2.3 Problem Reflection

2.3.1 Test Acc and Train Acc

It should be emphasized that in neural network training, the training and testing sets should be completely separated. Otherwise, the neural network may overfit

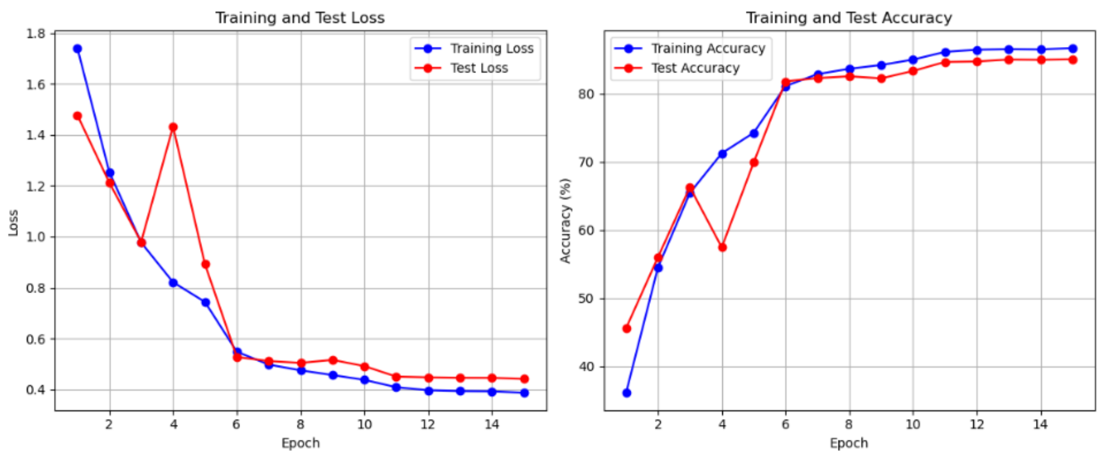


Figure 2: Training Curve

the training set and fail to generalize. Therefore, the test accuracy (Test Acc) is generally lower than the training accuracy (Train Acc), but the test accuracy is more objective.

2.3.2 Effect of Smaller Learning Rate

After decreasing the learning rate, the accuracy change rate decreases compared to the previous group, but the overall trend remains upward, resulting in a better final accuracy.

3 Task 2: Using CNN for Image Feature Extraction

3.1 Code Explanation

Since most of the code in the second task is easy to understand,I will only explain a small part of it.

This part of the code calculates the pairwise Euclidean distances between all pairs of feature vectors in the dataset.

$$\text{Distance Matrix} = \|\mathbf{X}_i - \mathbf{X}_j\|_2$$

where $\mathbf{X}_i$ and $\mathbf{X}_j$ are two feature vectors, and the $\|\cdot\|_2$ denotes the Euclidean norm (distance) between them.

- The code first prints a message indicating that the pairwise Euclidean distances are being computed.
- `distance_matrix = np.linalg.norm(features_array[:, np.newaxis] - features_array, axis=2)` computes the distance between each pair of feature vectors using `np.linalg.norm`, which calculates the Euclidean distance along the specified axis.
- The nested `for` loop iterates over all pairs of image indices and prints the Euclidean distance between each pair of images.

```
    # Compute pairwise Euclidean distances
    print('Computing pairwise Euclidean distances...')
    distance_matrix = np.linalg.norm(features_array[:, np.newaxis]
        - features_array, axis=2)

    # Display the results
    print('\nPairwise Euclidean Distances:')
    for (i, j) in combinations(range(len(valid_image_names)), 2):
        print(f'{valid_image_names[i]} <-> {valid_image_names[j]}:
            {distance_matrix[i, j]:.4f}')
```

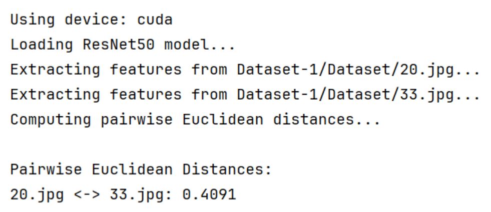
In this part of the code, the feature vectors are normalized, and the Euclidean distances to a query feature are computed.

- `features_norm = np.linalg.norm(features, axis=1, keepdims=True)` computes the L2-norm (Euclidean norm) of each feature vector.
- `features_norm[features_norm == 0] = 1` handles potential division by zero issues by setting norms that are zero to 1.
- The features are normalized by dividing each feature vector by its corresponding norm, ensuring that all feature vectors lie on the unit sphere.
- `distances = np.linalg.norm(features_normalized - query_feature_normalized, axis=1)` computes the Euclidean distance between the normalized feature vectors and the query feature.
- The indices of the top-K closest images are sorted, and the top-K distances and image names are selected for further processing.

```
    features_norm = np.linalg.norm(features, axis=1, keepdims=True
        )
    features_norm[features_norm == 0] = 1  # Prevent division by
        zero
    features_normalized = features / features_norm

    # Compute Euclidean distances
    distances = np.linalg.norm(features_normalized -
        query_feature_normalized, axis=1)

    # Get the top-K closest images
    sorted_indices = np.argsort(distances)
    top_indices = sorted_indices[:top_k]
    top_distances = distances[top_indices]
    top_image_names = image_names[top_indices]
```

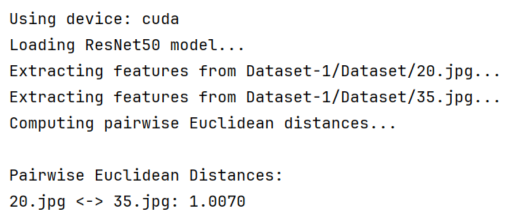
3.2 Corresponding Results

For similarity comparsion.py, I compare the similarity between image20 image33 and image35. Image20 and Image33 are a picture of a man, but image35 is a picture of an animal. And the result is shown in Figure3. You can see that the Euclidean Distance between 20.jpg and 33.jpg is smaller than that between 20.jpg and 35.jpg.



```
Using device: cuda
Loading ResNet50 model...
Extracting features from Dataset-1/Dataset/20.jpg...
Extracting features from Dataset-1/Dataset/33.jpg...
Computing pairwise Euclidean distances...

Pairwise Euclidean Distances:
20.jpg <-> 33.jpg: 0.4091
```

(a) Comparison between 20 and 33

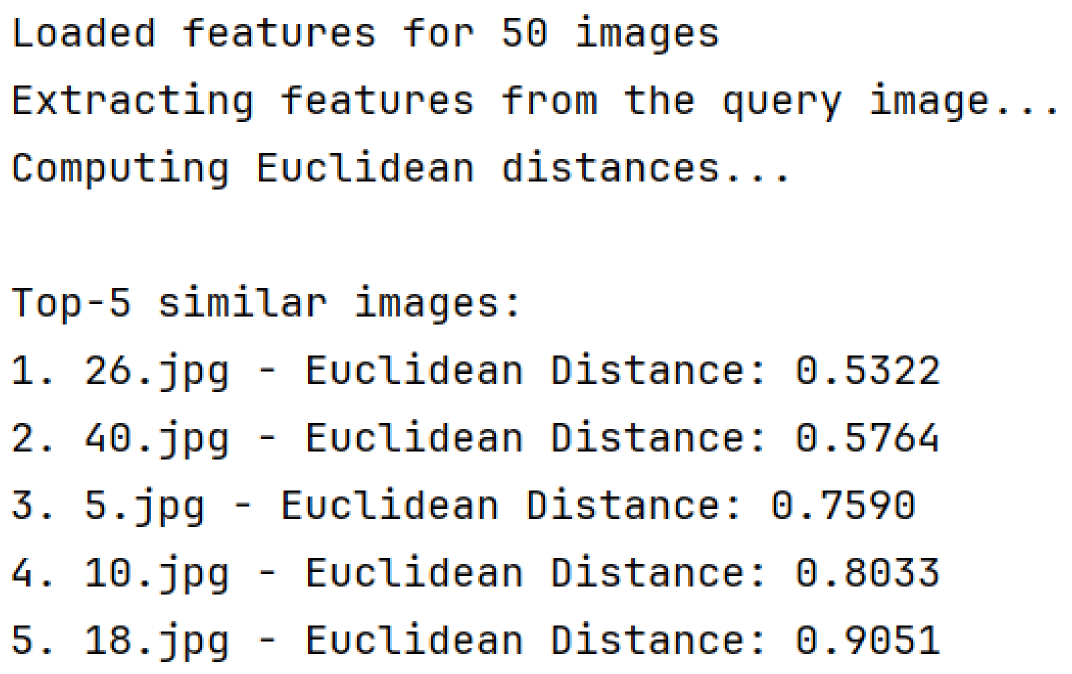


```
Using device: cuda
Loading ResNet50 model...
Extracting features from Dataset-1/Dataset/20.jpg...
Extracting features from Dataset-1/Dataset/35.jpg...
Computing pairwise Euclidean distances...

Pairwise Euclidean Distances:
20.jpg <-> 35.jpg: 1.0070
```

(b) Comparsion between 20 and 35

Figure 3: Overall caption for both images

For image search.py, I use leopard.png as the target image, and the result are shown in Figure4. Among the top5 imilar images, image 26, 40, 5 ,10 are all pictures of leopards, and image18 is rich in yellow which is also similar to the color in the target image.This suggests that this method is reliable. If you use other images, you can find the similar results.



```
Loaded features for 50 images
Extracting features from the query image...
Computing Euclidean distances...

Top-5 similar images:
1. 26.jpg - Euclidean Distance: 0.5322
2. 40.jpg - Euclidean Distance: 0.5764
3. 5.jpg - Euclidean Distance: 0.7590
4. 10.jpg - Euclidean Distance: 0.8033
5. 18.jpg - Euclidean Distance: 0.9051
```

Figure 4: Search Results

3.3 Problem Reflection

3.3.1 Other Available Image Feature Extraction Networks

MobileNet is a lightweight deep learning model designed for efficient image classification tasks, especially on mobile and embedded devices with limited computational resources. MobileNet introduces the concept of depthwise separable convolutions, which significantly reduce the number of parameters and computational cost compared to standard convolutions.

4 Conclusion

This experiment provided valuable insights into the process of training neural networks using PyTorch, with a focus on both theoretical understanding and practical implementation. Throughout the tasks, I gained a deeper appreciation for how models are trained, the challenges involved, and how various parameters (such as learning rate and training epochs) can significantly impact the model's performance.

The training of neural networks is a highly iterative process, where selecting the right parameters, such as learning rate and the number of epochs, is crucial to achieving optimal results.

Writing this report also significantly improved my ability to document and present code-related work. I focused on clear explanations of the code and the rationale behind the decisions made during the experiment. Learning how to structure reports to include not only the code itself but also the insights and reflections on the experiment was a valuable skill.

The field of artificial intelligence, particularly deep learning, holds immense potential for transforming industries across the globe. As demonstrated in this experiment, AI models such as convolutional neural networks can already perform impressive tasks like image classification and feature extraction with relatively simple implementations. However, the real power of AI lies in its ability to scale and improve with larger datasets and more sophisticated architectures.

Overall, this experiment has given me a glimpse into the vast potential of AI, and I am excited about the opportunities it presents in shaping the future of technology. As I continue to learn and experiment with AI, I look forward to contributing to the development of innovative solutions that can make a meaningful impact on society.