

GANFORGE Assignment Report

Task 1: Iris Dataset Classification

June 13, 2025

Insights

While working on this task, I learned how to build a complete machine learning pipeline using PyTorch for a classical dataset. Specifically:

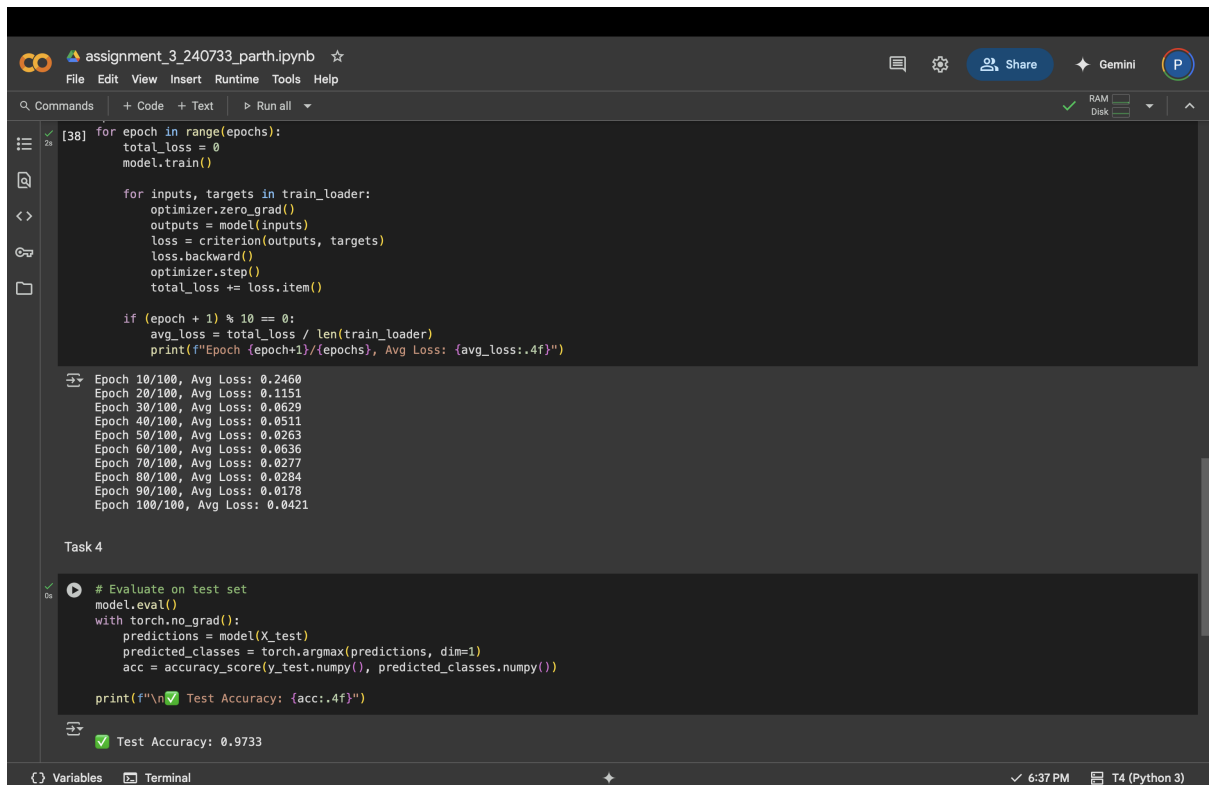
- How to load and preprocess datasets using Pandas, LabelEncoder, and StandardScaler.
- Building and training a multi-layer neural network using PyTorch's `nn.Module`.
- Implementing batch training with DataLoader.
- Evaluating model performance on unseen test data using `accuracy_score`.
- Understanding why small, clean datasets like Iris can lead to high accuracy (sometimes even 100%) when modeled with a well-structured neural network.

This assignment gave me valuable experience in end-to-end implementation and model evaluation in a low-data regime.

Console Screenshot

This screenshot shows the code run and printed accuracy

...



```
[38] for epoch in range(epochs):
    total_loss = 0
    model.train()

    for inputs, targets in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    if (epoch + 1) % 10 == 0:
        avg_loss = total_loss / len(train_loader)
        print(f"Epoch {epoch+1}/{epochs}, Avg Loss: {avg_loss:.4f}")

Epoch 10/100, Avg Loss: 0.2460
Epoch 20/100, Avg Loss: 0.1151
Epoch 30/100, Avg Loss: 0.0629
Epoch 40/100, Avg Loss: 0.0511
Epoch 50/100, Avg Loss: 0.0263
Epoch 60/100, Avg Loss: 0.0636
Epoch 70/100, Avg Loss: 0.0277
Epoch 80/100, Avg Loss: 0.0284
Epoch 90/100, Avg Loss: 0.0178
Epoch 100/100, Avg Loss: 0.0421

Task 4

# Evaluate on test set
model.eval()
with torch.no_grad():
    predictions = model(X_test)
    predicted_classes = torch.argmax(predictions, dim=1)
    acc = accuracy_score(y_test.numpy(), predicted_classes.numpy())

print(f"\n✅ Test Accuracy: {acc:.4f}")

✅ Test Accuracy: 0.9733
```

Figure 1: Screenshot showing training and accuracy

GitHub Repository

The full codebase has been uploaded to GitHub and can be accessed at the following link:
<https://github.com/Fighter-195/GanForge>

Task 2: Image Similarity Search using Pre-trained Model and Annoy

For this task, I built an image similarity search tool using the CIFAR-10 dataset. I used the ResNet18 model pre-trained on ImageNet and removed its final classification layer to extract feature embeddings of the images.

These embeddings represent each image in a 512-dimensional space. I then used Spotify's **Annoy** (Approximate Nearest Neighbors Oh Yeah) library to index these embeddings and retrieve similar images based on Euclidean distance.

I randomly picked an image from the dataset and queried its top 5 most similar images using the built index. The system successfully returned visually and semantically similar images from CIFAR-10.

My Understanding of Feature Detection:

In this task, the ResNet18 model acts as a feature detector. When an image is passed through the network (excluding the last layer), it outputs a compact vector that captures important visual patterns like edges, shapes, and textures. These vectors are what we call embeddings or feature representations.

These embeddings make it possible to compare two images meaningfully — not by pixel values, but by their learned content. So even if two images are not pixel-wise identical, if they contain similar objects or patterns, their embeddings will be close in the vector space. That’s how feature detection helps in building an image similarity system.

Screenshot Showing Code and Output:

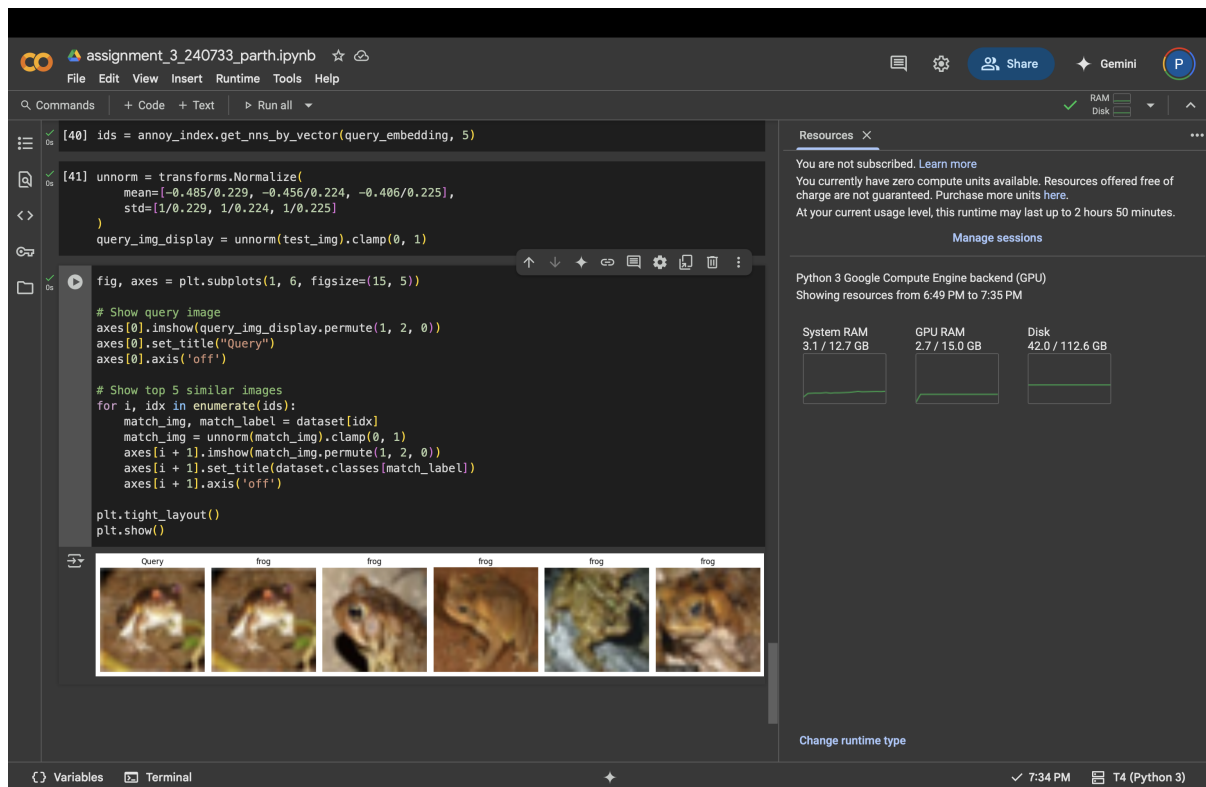


Figure 2: Console Output of Similar Image Retrieval

Output Image of Query and Similar Matches: