RA2311047010040

மரம் வளர்ப்போம்
மண் வளம் காப்போம்

Superior®
Note Books

Index

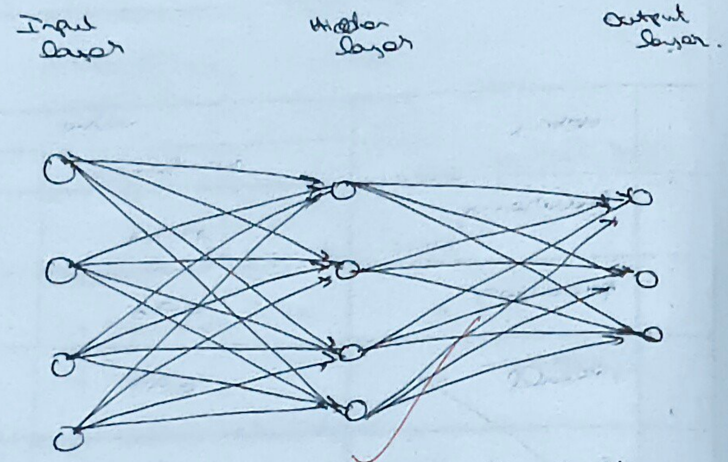
Name : Vishva S

Subject : DL

Std. : .... Sec. : .... Roll No. ....

School : ....

No.	Date	Title	Marks	Signature
1	24/7/25	Exploring the DL platforms.		[illegible]
2	31/7/25	Implement a classifier using open source dataset.		
3	7/8/25	Study the classifier with respect to . statistical parameters.		[illegible] 1/8/25
4	14/8/25	Build a simple feed forward neural network to Recognize Handwritten Characters		[illegible] 22/8/25
5	22/8/25	Study of Activation functions & Their role.		



feed forward neural network

14/8/25 4: Build a simple feed forward neural network to Recognize Handwritten characters (MNIST Dataset)

Aim:

To design & implement a simple feed forward neural network to classify handwritten digits from the MNIST dataset.

Objective:

→ To understand the architecture & working of a feed forward neural network.

→ To train the neural network using the MNIST dataset.

→ To evaluate the network's performance using accuracy as the primary metric

→ To demonstrate basic digit recognition using deep learning

Observation:

* Dataset used: MNIST Handwritten Digit Dataset (60,000 training images, 10,000 testing images, grayscale, size 28×28 pixels)

* Model Architecture:

→ Input layer: 784 neurons (flattened 28×28 image)

→ Hidden layer(s): one or more dense layers with activation function ReLU.

→ Output layer: 10 neurons (digits 0-9) with activation function Softmax

* Loss Function : Categorical crossentropy

* optimizer : Adam

* Evaluation Metric : Accuracy.

pseudocode:

```
BEGIN
  St1 : Load the MNIST dataset
  St2 : preprocess data
        * normalize all pixel values to range [0,1]
        * Flatten each 28x28 image into a 1D array of size 784
        * convert labels to onehot encoding format.
  St3 : Define the neural network architecture
        * Input layer : 784 neurons
        * Hidden layer : 128 neurons, ReLU activation function
        * output layer : 10 neurons, softmax activation function
  St4 : configure training parameters
        * loss function : categorical crossentropy
        * optimizer : Adam
        * Evaluation Metric : Accuracy.
  St5 : Train the neural network
        * use training data
        * Iterate for N epochs with a fixed batch size.
  St6 : Evaluate the model using the test dataset.
        * calculate accuracy on unseen data.
  St 7 : Display the evaluation results.
END.
```

MNIST dataset:

10 classes (0 to 9)

784 features (28x28)

Observation from the code:

```
Epoch 1/20 - Loss: 0.9036 - Train Accuracy : 87.65%.
Epoch 2/20 - Loss : 0.4213 - Train Accuracy : 89.75%.
Epoch 3/20 - Loss : 0.3527 - Train Accuracy : 90.77%.
  :
  :
Epoch 20/20 - Loss : 0.1250 - Train Accuracy : 98.08%.
```

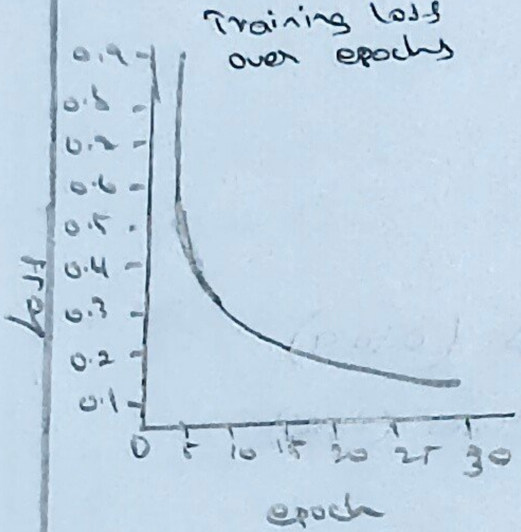
Test Accuracy : ~~96.06~~ 97.14 %

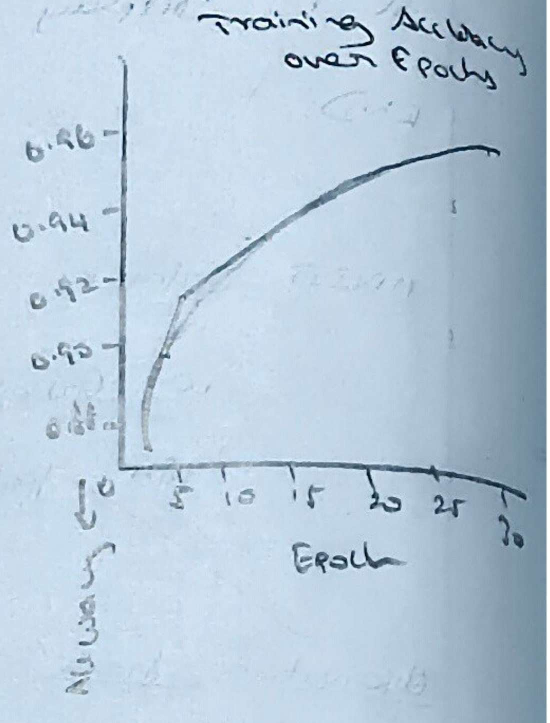
precision : ~~0.9606~~ 0.9722

Recall : ~~0.9606~~ 0.9721

F1 score : ~~0.9606~~ 0.9721

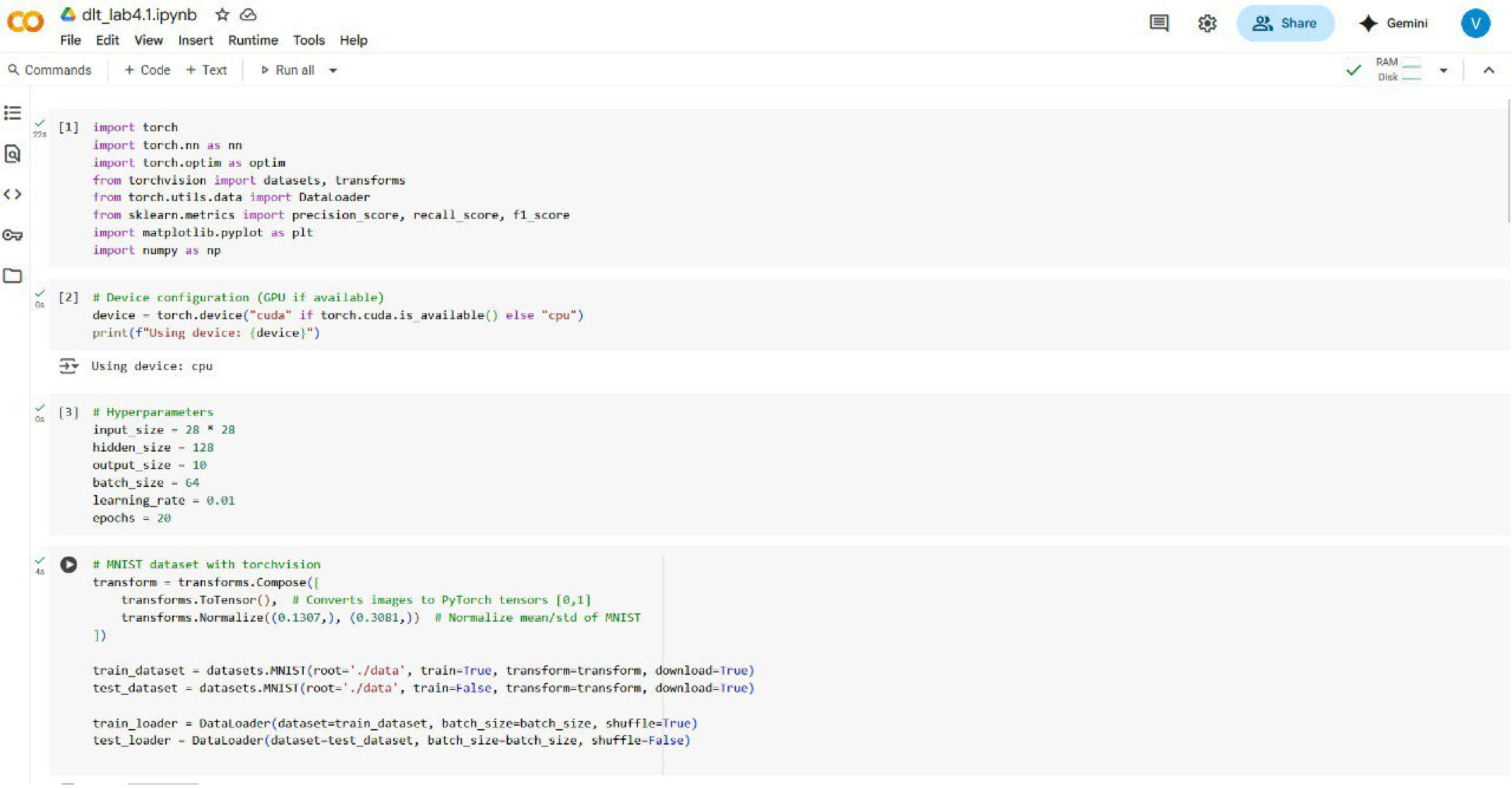
Graph :





Result :

Successfully implemented the feed forward neural network to recognize the handwritten ~~de~~ characters.



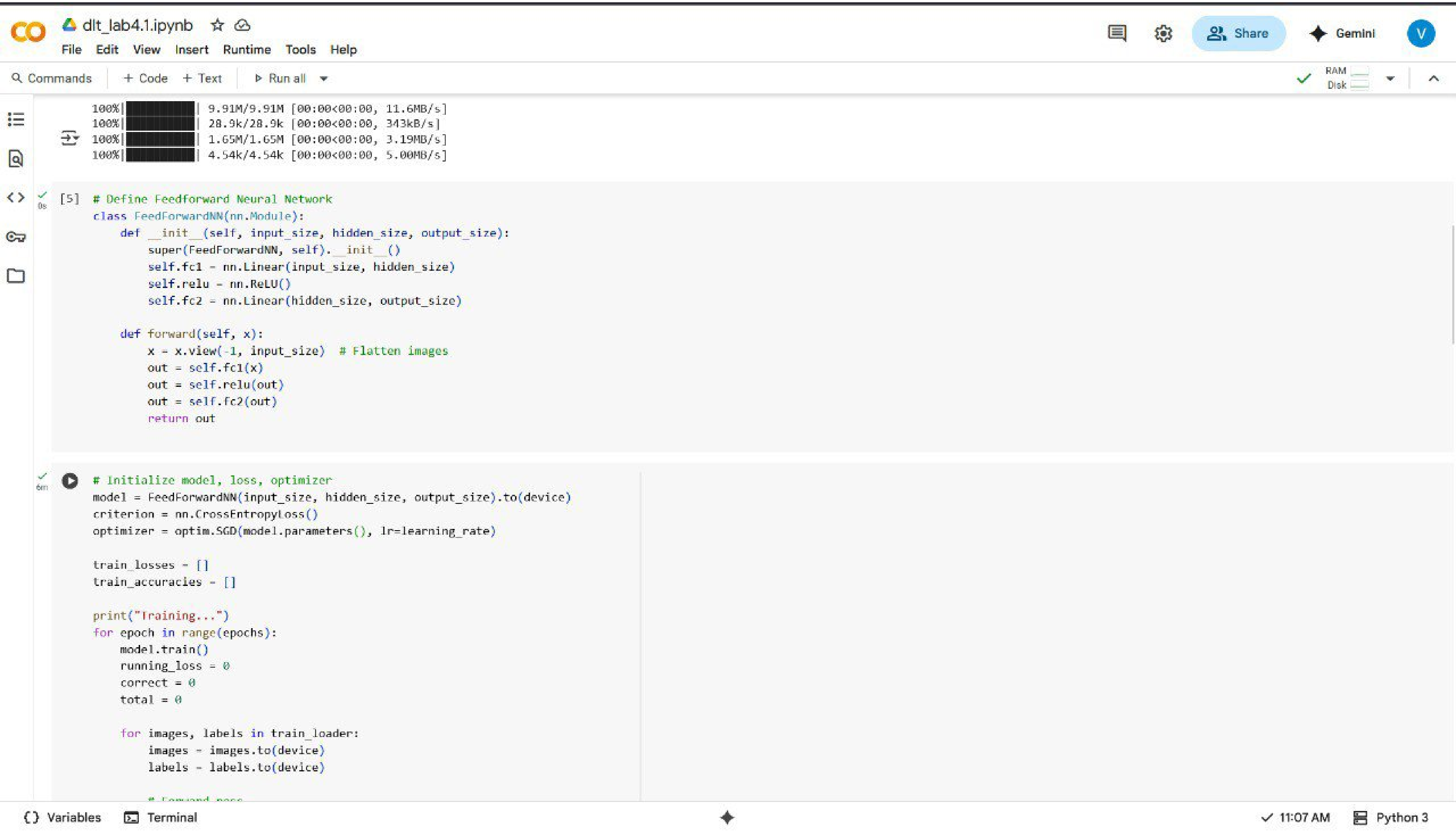
```
[1] import torch
    import torch.nn as nn
    import torch.optim as optim
    from torchvision import datasets, transforms
    from torch.utils.data import DataLoader
    from sklearn.metrics import precision_score, recall_score, f1_score
    import matplotlib.pyplot as plt
    import numpy as np
```

```
[2] # Device configuration (GPU if available)
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"Using device: {device}")
```

```
Using device: cpu
```

```
[3] # Hyperparameters
    input_size = 28 * 28
    hidden_size = 128
    output_size = 10
    batch_size = 64
    learning_rate = 0.01
    epochs = 20
```

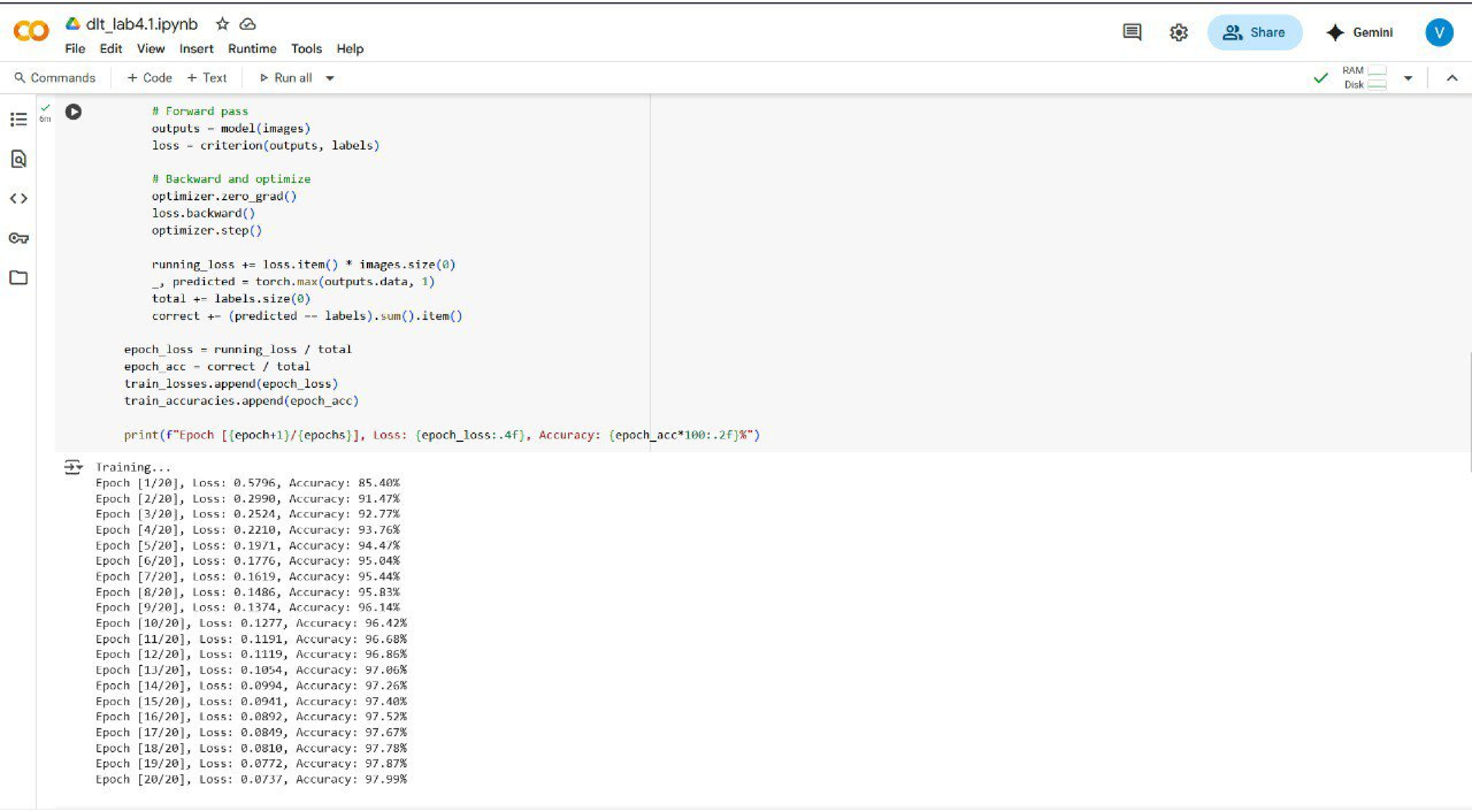
```
# MNIST dataset with torchvision
transform = transforms.Compose([
    transforms.ToTensor(),  # Converts images to PyTorch tensors [0,1]
    transforms.Normalize((0.1307,), (0.3081,))  # Normalize mean/std of MNIST
])

train_dataset = datasets.MNIST(root='./data', train=True, transform=transform, download=True)
test_dataset = datasets.MNIST(root='./data', train=False, transform=transform, download=True)

train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False)
```


dlt_lab4.1.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text Run all
RAM
Disk
Variables Terminal
11:07 AM
Python 3

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 11.6MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 343kB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 3.19MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 5.00MB/s]
```

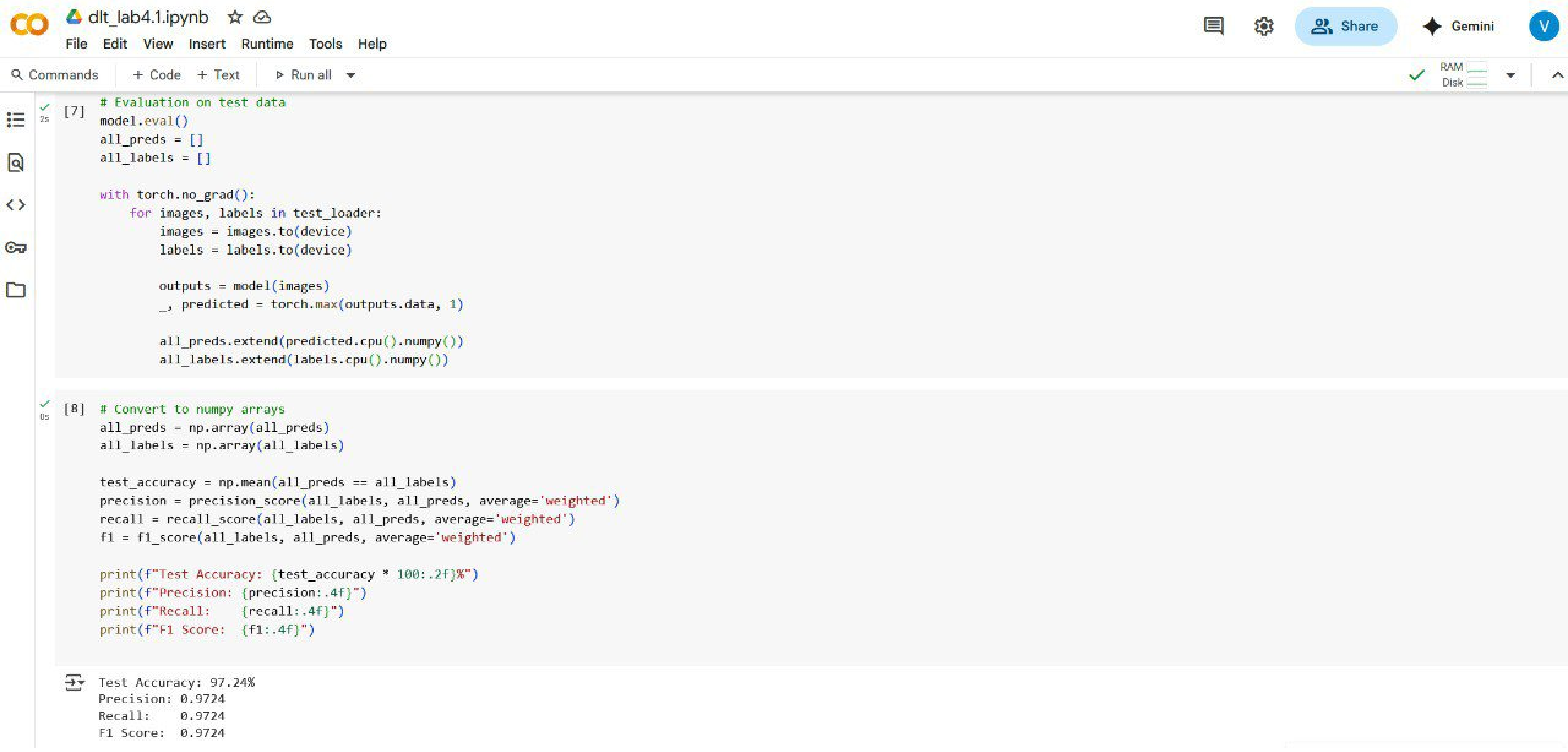
[5]
```
# Define Feedforward Neural Network
class FeedForwardNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(FeedForwardNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = x.view(-1, input_size)  # Flatten images
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

```
# Initialize model, loss, optimizer
model = FeedForwardNN(input_size, hidden_size, output_size).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=learning_rate)

train_losses = []
train_accuracies = []

print("Training...")
for epoch in range(epochs):
    model.train()
    running_loss = 0
    correct = 0
    total = 0

    for images, labels in train_loader:
        images = images.to(device)
        labels = labels.to(device)
```



```
            # Forward pass
            outputs = model(images)
            loss = criterion(outputs, labels)

            # Backward and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * images.size(0)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        epoch_loss = running_loss / total
        epoch_acc = correct / total
        train_losses.append(epoch_loss)
        train_accuracies.append(epoch_acc)

        print(f"Epoch [{epoch+1}/{epochs}], Loss: {epoch_loss:.4f}, Accuracy: {epoch_acc*100:.2f}%")
```

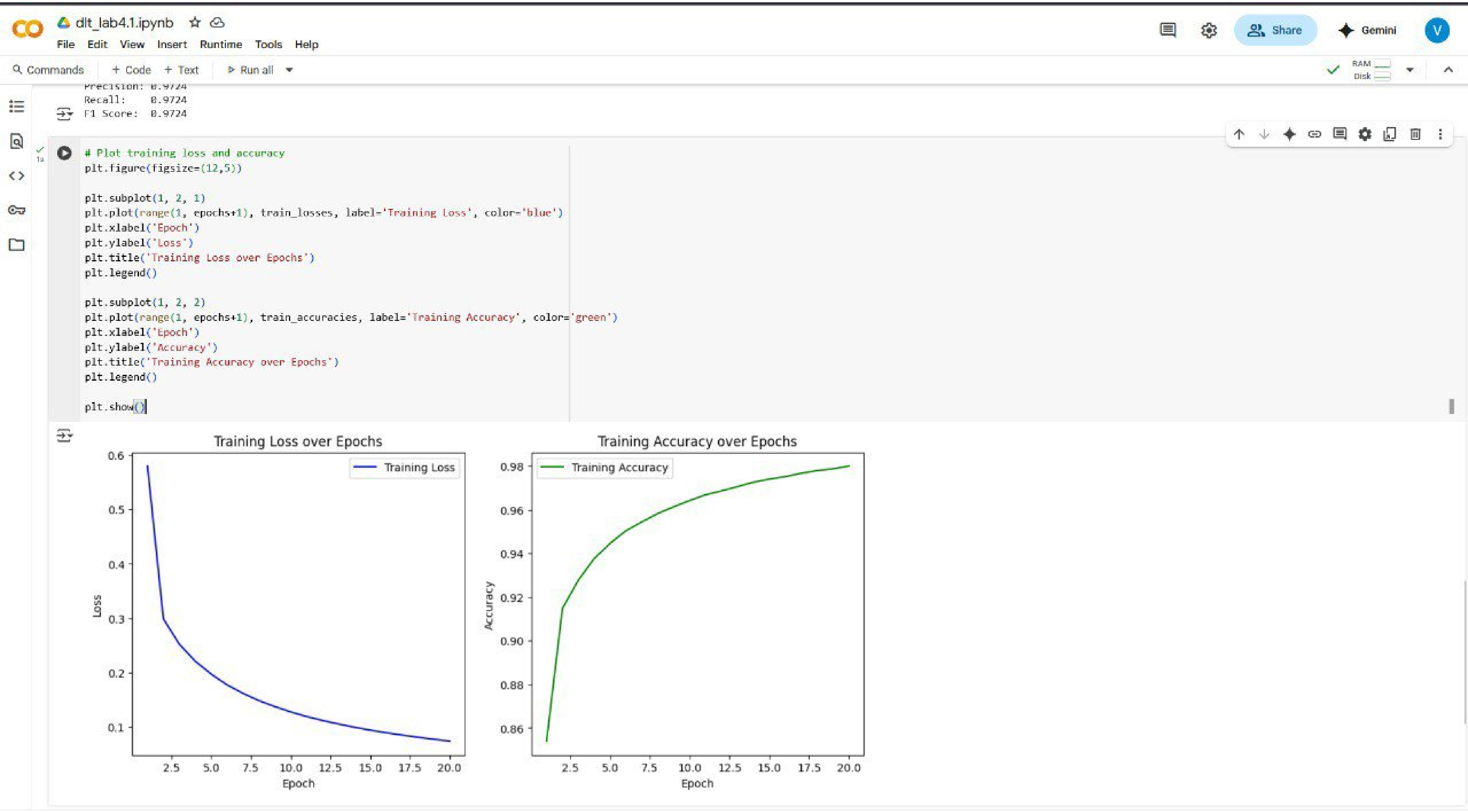
```
Training...
Epoch [1/20], Loss: 0.5796, Accuracy: 85.40%
Epoch [2/20], Loss: 0.2990, Accuracy: 91.47%
Epoch [3/20], Loss: 0.2524, Accuracy: 92.77%
Epoch [4/20], Loss: 0.2210, Accuracy: 93.76%
Epoch [5/20], Loss: 0.1971, Accuracy: 94.47%
Epoch [6/20], Loss: 0.1776, Accuracy: 95.04%
Epoch [7/20], Loss: 0.1619, Accuracy: 95.44%
Epoch [8/20], Loss: 0.1486, Accuracy: 95.83%
Epoch [9/20], Loss: 0.1374, Accuracy: 96.14%
Epoch [10/20], Loss: 0.1277, Accuracy: 96.42%
Epoch [11/20], Loss: 0.1191, Accuracy: 96.68%
Epoch [12/20], Loss: 0.1119, Accuracy: 96.86%
Epoch [13/20], Loss: 0.1054, Accuracy: 97.06%
Epoch [14/20], Loss: 0.0994, Accuracy: 97.26%
Epoch [15/20], Loss: 0.0941, Accuracy: 97.40%
Epoch [16/20], Loss: 0.0892, Accuracy: 97.52%
Epoch [17/20], Loss: 0.0849, Accuracy: 97.67%
Epoch [18/20], Loss: 0.0810, Accuracy: 97.78%
Epoch [19/20], Loss: 0.0772, Accuracy: 97.87%
Epoch [20/20], Loss: 0.0737, Accuracy: 97.99%
```


dlt_lab4.1.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Gemini
RAM
Disk

```
[7] # Evaluation on test data
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for images, labels in test_loader:
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)

            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
```

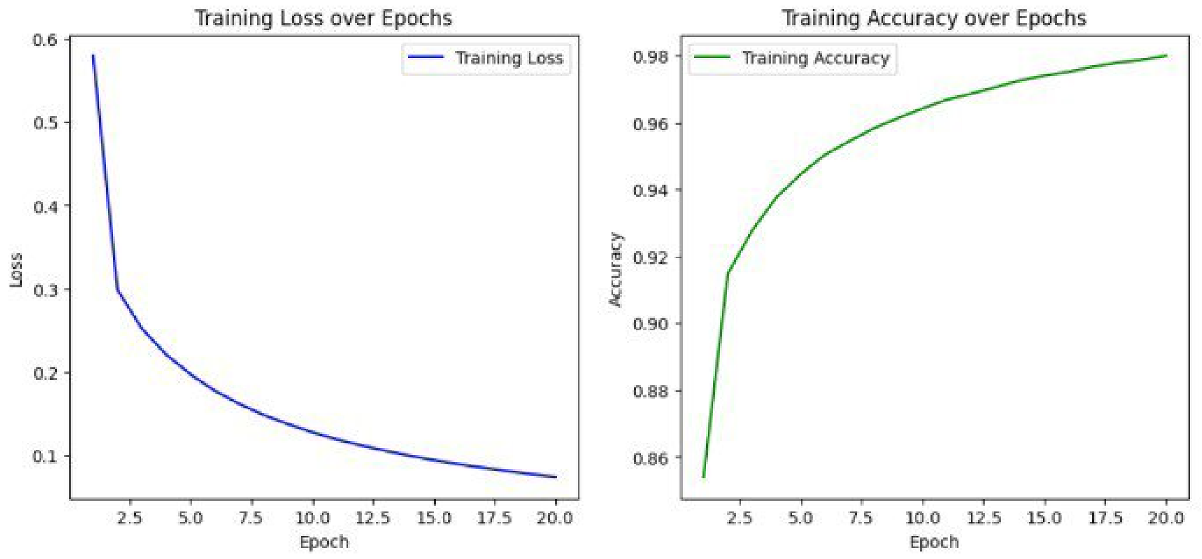
```
[8] # Convert to numpy arrays
    all_preds = np.array(all_preds)
    all_labels = np.array(all_labels)

    test_accuracy = np.mean(all_preds == all_labels)
    precision = precision_score(all_labels, all_preds, average='weighted')
    recall = recall_score(all_labels, all_preds, average='weighted')
    f1 = f1_score(all_labels, all_preds, average='weighted')

    print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
    print(f"Precision: {precision:.4f}")
    print(f"Recall:    {recall:.4f}")
    print(f"F1 Score:  {f1:.4f}")
```

```
Test Accuracy: 97.24%
Precision: 0.9724
Recall:    0.9724
F1 Score:  0.9724
```


dlt_lab4.1.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share Gemini
RAM Disk

```
Precision: 0.9724
Recall:    0.9724
F1 Score:  0.9724
```

```python
# Plot training loss and accuracy
plt.figure(figsize=(12,5))

plt.subplot(1, 2, 1)
plt.plot(range(1, epochs+1), train_losses, label='Training Loss', color='blue')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss over Epochs')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(range(1, epochs+1), train_accuracies, label='Training Accuracy', color='green')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training Accuracy over Epochs')
plt.legend()

plt.show()
```


Training Loss over Epochs
Training Loss
Loss
Epoch
Training Accuracy over Epochs
Training Accuracy
Accuracy
Epoch