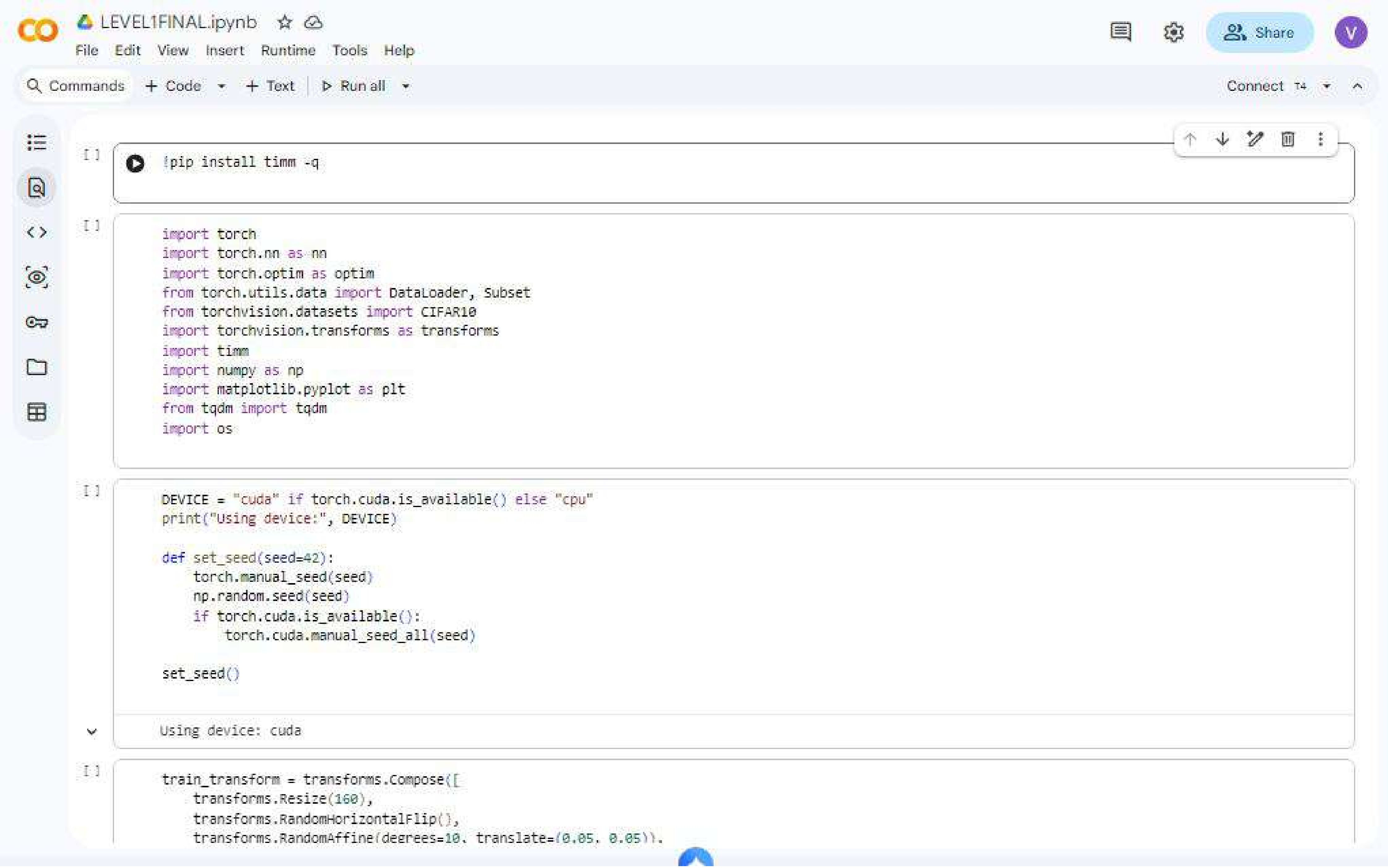


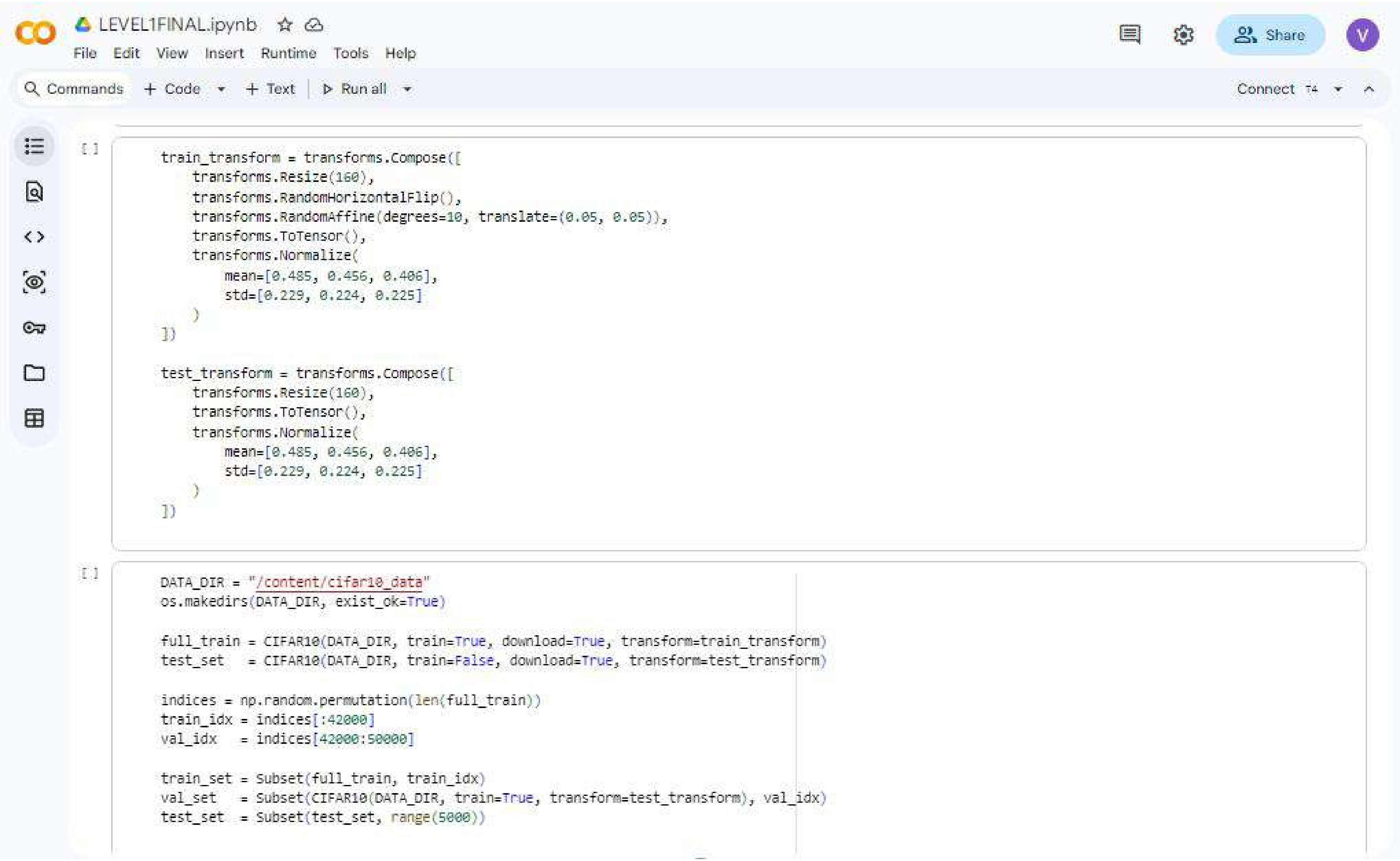
```
!pip install timm -q
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Subset
from torchvision.datasets import CIFAR10
import torchvision.transforms as transforms
import timm
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
import os
```

```
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
print("Using device:", DEVICE)

def set_seed(seed=42):
    torch.manual_seed(seed)
    np.random.seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)

set_seed()
```

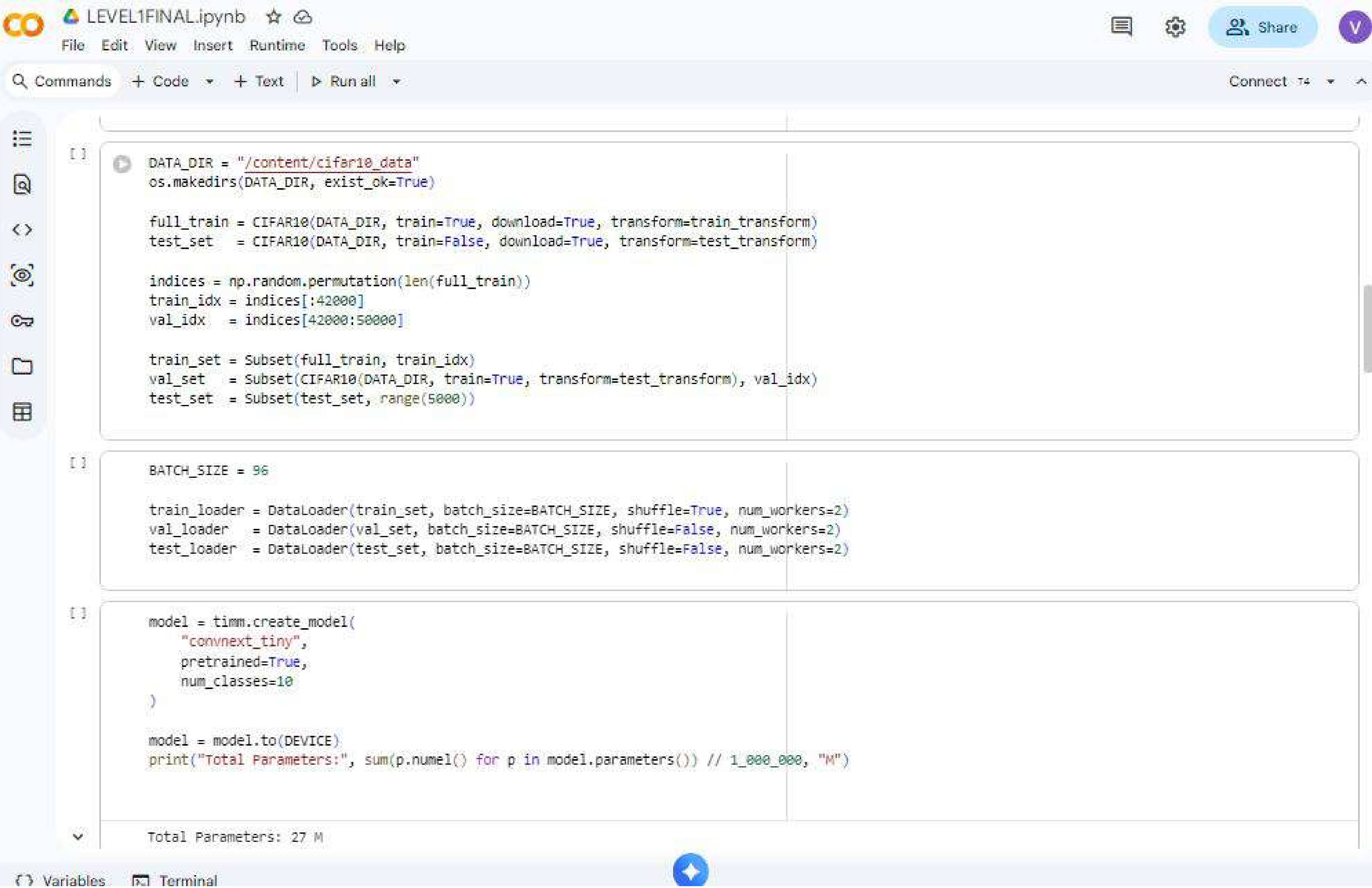
```
Using device: cuda
```

```
train_transform = transforms.Compose([
    transforms.Resize(160),
    transforms.RandomHorizontalFlip(),
    transforms.RandomAffine(degrees=10, translate=(0.05, 0.05)),
```



```
train_transform = transforms.Compose([
    transforms.Resize(160),
    transforms.RandomHorizontalFlip(),
    transforms.RandomAffine(degrees=10, translate=(0.05, 0.05)),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

test_transform = transforms.Compose([
    transforms.Resize(160),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])
```

```
DATA_DIR = "/content/cifar10_data"
os.makedirs(DATA_DIR, exist_ok=True)

full_train = CIFAR10(DATA_DIR, train=True, download=True, transform=train_transform)
test_set   = CIFAR10(DATA_DIR, train=False, download=True, transform=test_transform)

indices = np.random.permutation(len(full_train))
train_idx = indices[:42000]
val_idx   = indices[42000:50000]

train_set = Subset(full_train, train_idx)
val_set   = Subset(CIFAR10(DATA_DIR, train=True, transform=test_transform), val_idx)
test_set  = Subset(test_set, range(5000))
```

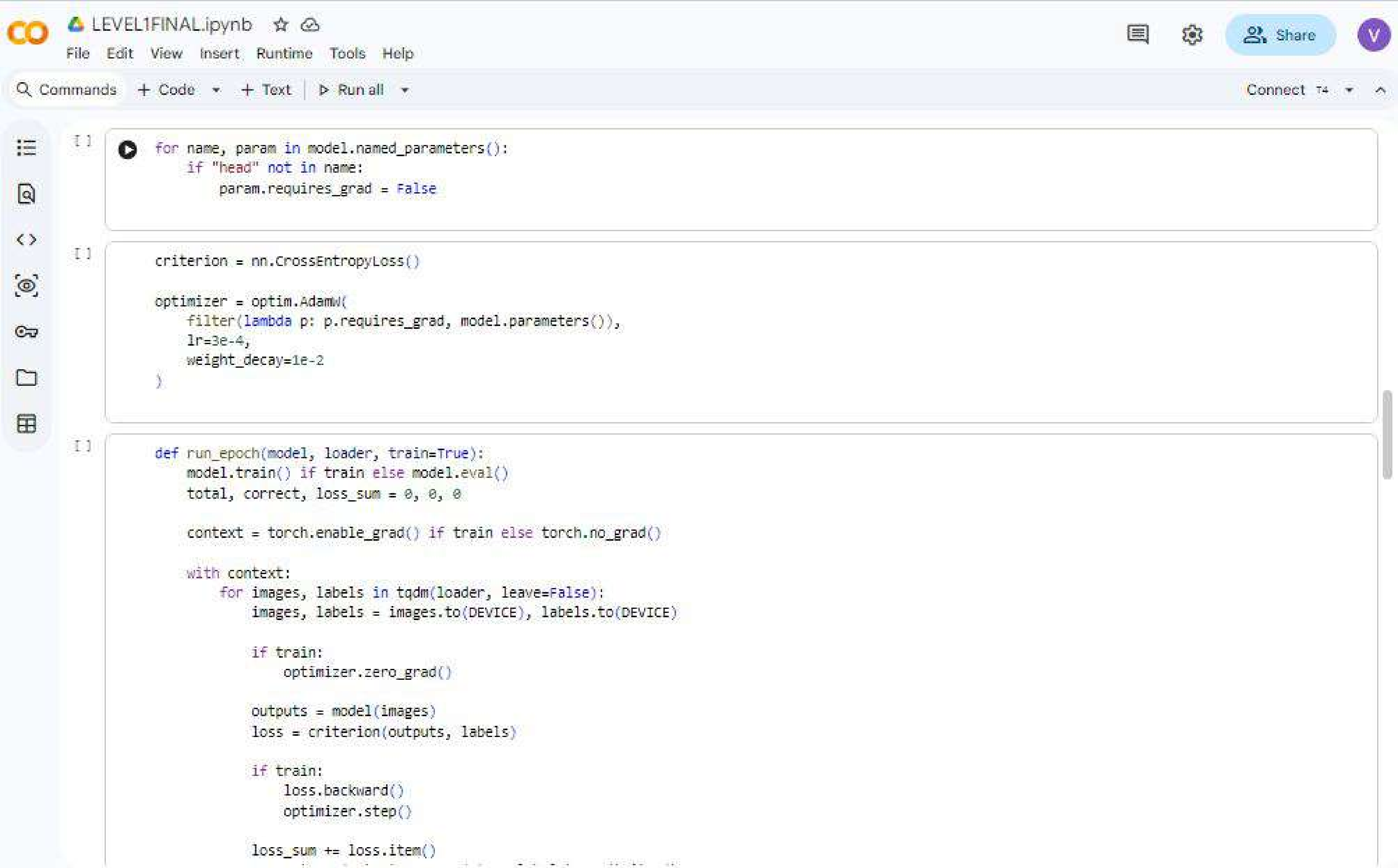

LEVEL1FINAL.ipynb
File Edit View Insert Runtime Tools Help
Share
V
Commands + Code + Text Run all
Connect T4
Variables Terminal

```
DATA_DIR = "/content/cifar10_data"
os.makedirs(DATA_DIR, exist_ok=True)

full_train = CIFAR10(DATA_DIR, train=True, download=True, transform=train_transform)
test_set   = CIFAR10(DATA_DIR, train=False, download=True, transform=test_transform)

indices = np.random.permutation(len(full_train))
train_idx = indices[:42000]
val_idx   = indices[42000:50000]

train_set = Subset(full_train, train_idx)
val_set   = Subset(CIFAR10(DATA_DIR, train=True, transform=test_transform), val_idx)
test_set  = Subset(test_set, range(5000))
```

```
BATCH_SIZE = 96

train_loader = DataLoader(train_set, batch_size=BATCH_SIZE, shuffle=True, num_workers=2)
val_loader   = DataLoader(val_set, batch_size=BATCH_SIZE, shuffle=False, num_workers=2)
test_loader  = DataLoader(test_set, batch_size=BATCH_SIZE, shuffle=False, num_workers=2)
```

```
model = timm.create_model(
    "convnext_tiny",
    pretrained=True,
    num_classes=10
)

model = model.to(DEVICE)
print("Total Parameters:", sum(p.numel() for p in model.parameters()) // 1_000_000, "M")
```

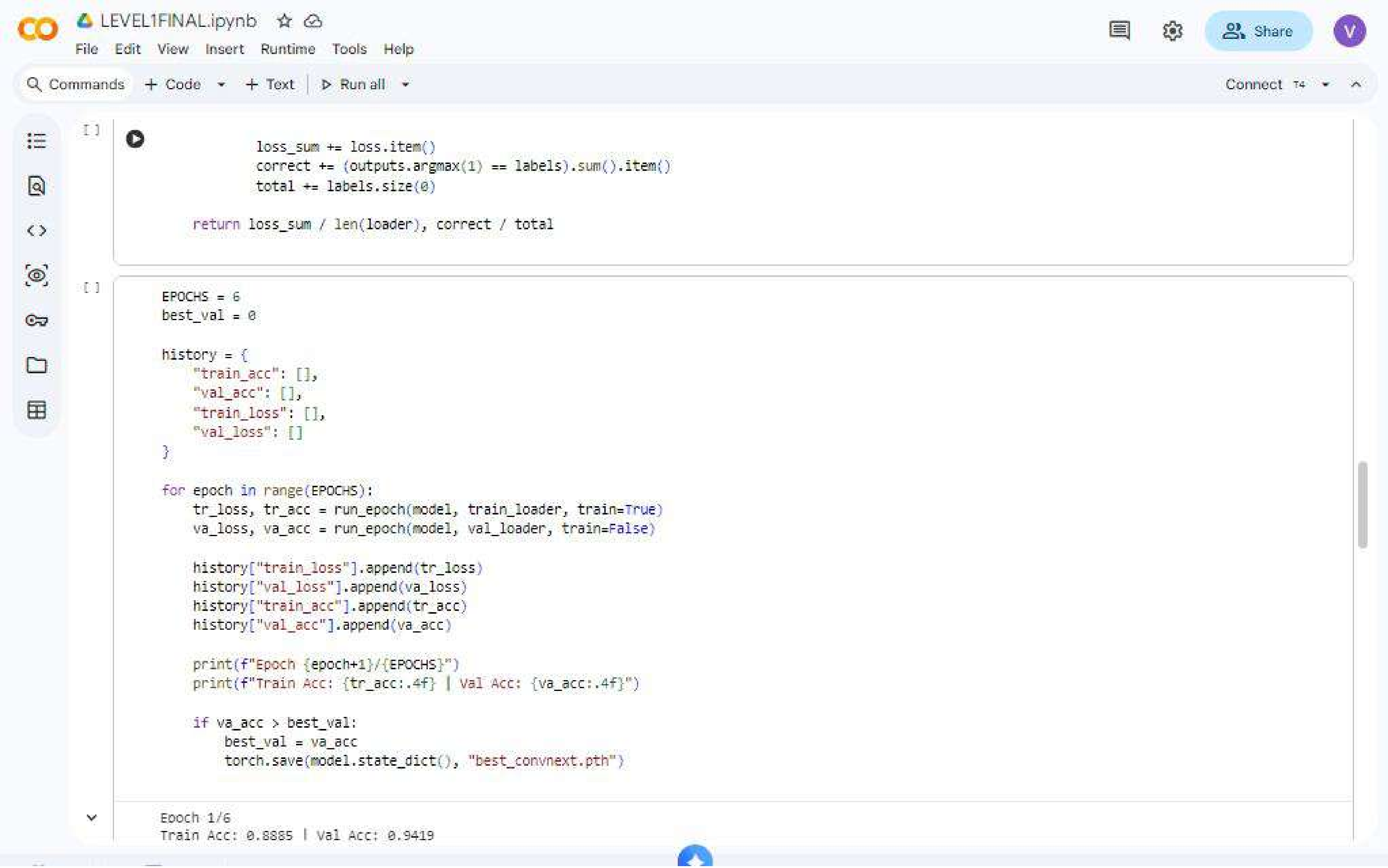
```
Total Parameters: 27 M
```



```
for name, param in model.named_parameters():
    if "head" not in name:
        param.requires_grad = False
```

```
criterion = nn.CrossEntropyLoss()

optimizer = optim.AdamW(
    filter(lambda p: p.requires_grad, model.parameters()),
    lr=3e-4,
    weight_decay=1e-2
)
```

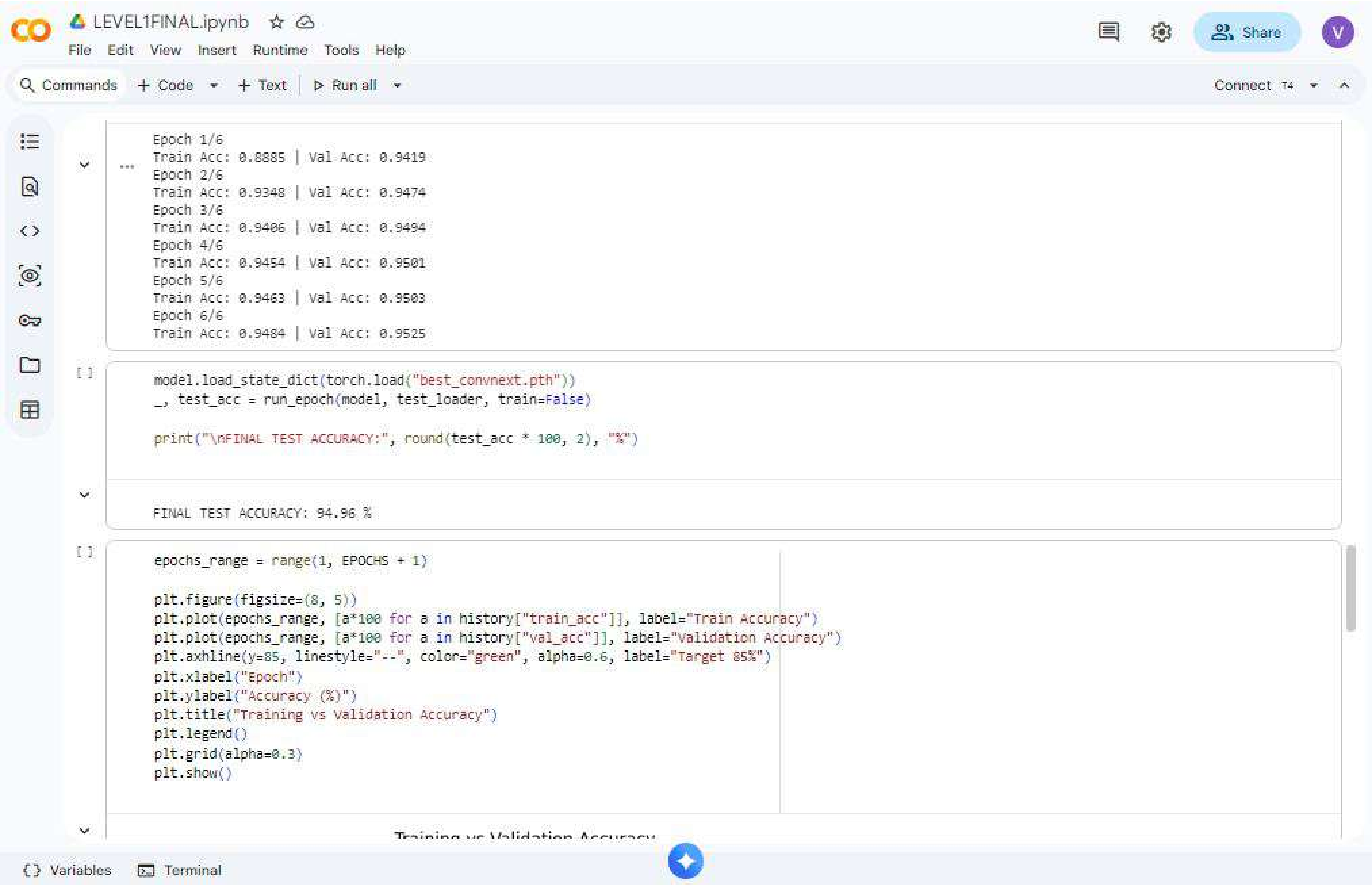
```
def run_epoch(model, loader, train=True):
    model.train() if train else model.eval()
    total, correct, loss_sum = 0, 0, 0

    context = torch.enable_grad() if train else torch.no_grad()

    with context:
        for images, labels in tqdm(loader, leave=False):
            images, labels = images.to(DEVICE), labels.to(DEVICE)

            if train:
                optimizer.zero_grad()

            outputs = model(images)
            loss = criterion(outputs, labels)

            if train:
                loss.backward()
                optimizer.step()

            loss_sum += loss.item()
```



```
            loss_sum += loss.item()
            correct += (outputs.argmax(1) == labels).sum().item()
            total += labels.size(0)

    return loss_sum / len(loader), correct / total
```

```
EPOCHS = 6
best_val = 0

history = {
    "train_acc": [],
    "val_acc": [],
    "train_loss": [],
    "val_loss": []
}

for epoch in range(EPOCHS):
    tr_loss, tr_acc = run_epoch(model, train_loader, train=True)
    va_loss, va_acc = run_epoch(model, val_loader, train=False)

    history["train_loss"].append(tr_loss)
    history["val_loss"].append(va_loss)
    history["train_acc"].append(tr_acc)
    history["val_acc"].append(va_acc)

    print(f"Epoch {epoch+1}/{EPOCHS}")
    print(f"Train Acc: {tr_acc:.4f} | Val Acc: {va_acc:.4f}")

    if va_acc > best_val:
        best_val = va_acc
        torch.save(model.state_dict(), "best_convnext.pth")
```

```
Epoch 1/6
Train Acc: 0.8885 | Val Acc: 0.9419
```

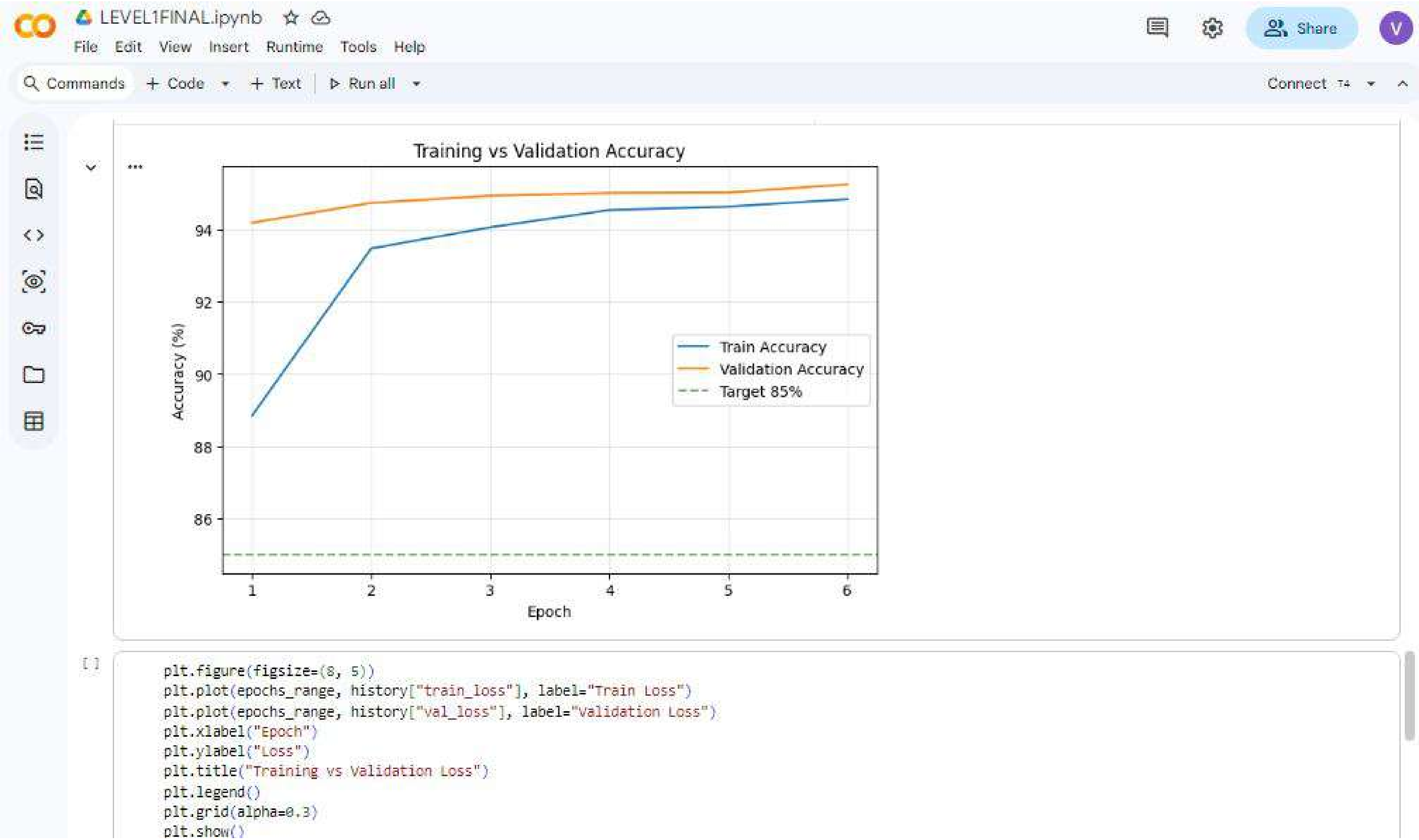


```
Epoch 1/6
Train Acc: 0.8885 | Val Acc: 0.9419
Epoch 2/6
Train Acc: 0.9348 | Val Acc: 0.9474
Epoch 3/6
Train Acc: 0.9406 | Val Acc: 0.9494
Epoch 4/6
Train Acc: 0.9454 | Val Acc: 0.9501
Epoch 5/6
Train Acc: 0.9463 | Val Acc: 0.9503
Epoch 6/6
Train Acc: 0.9484 | Val Acc: 0.9525
```

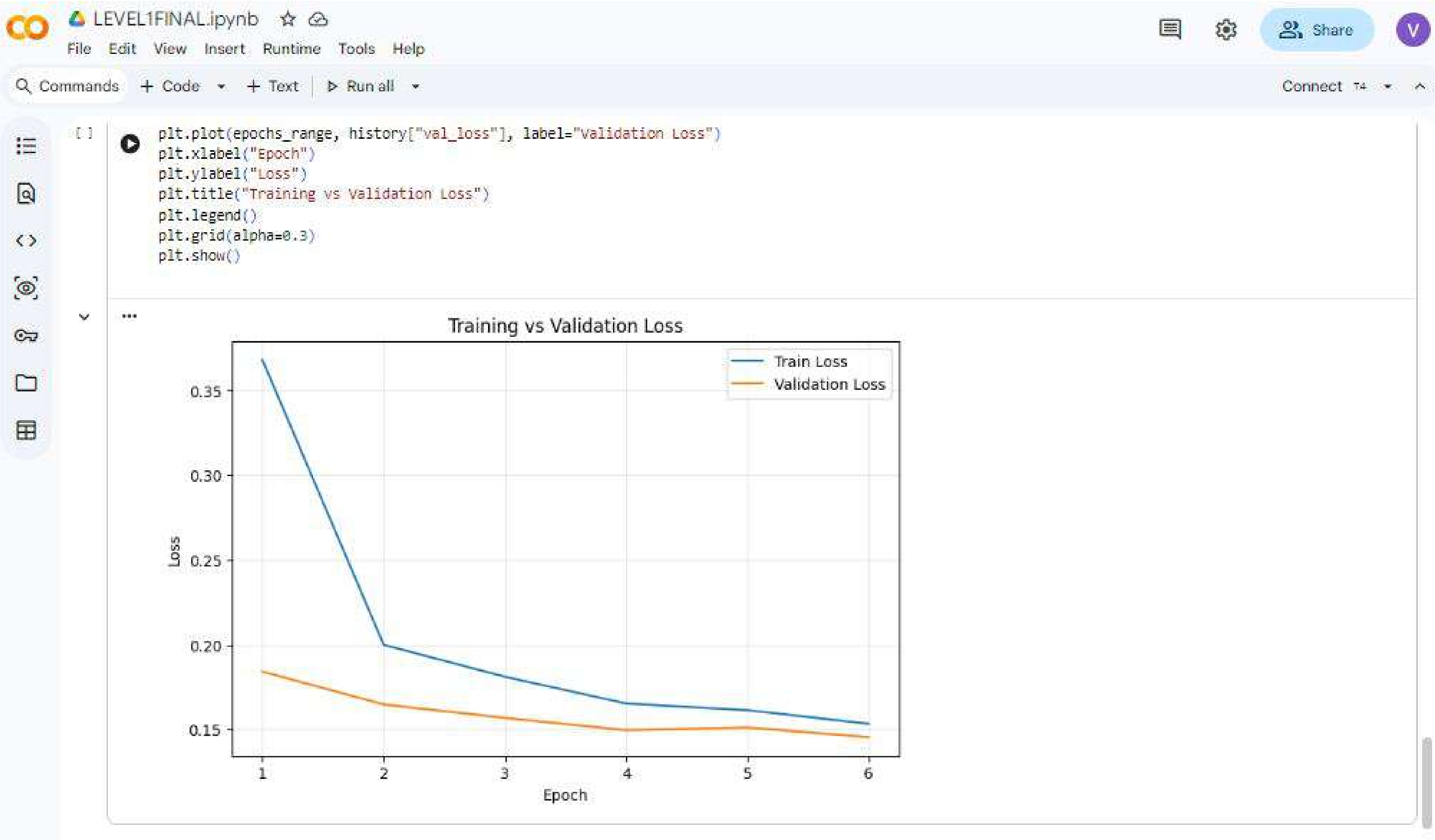
```
model.load_state_dict(torch.load("best_convnext.pth"))
_, test_acc = run_epoch(model, test_loader, train=False)

print("\nFINAL TEST ACCURACY:", round(test_acc * 100, 2), "%")
```

```
FINAL TEST ACCURACY: 94.96 %
```

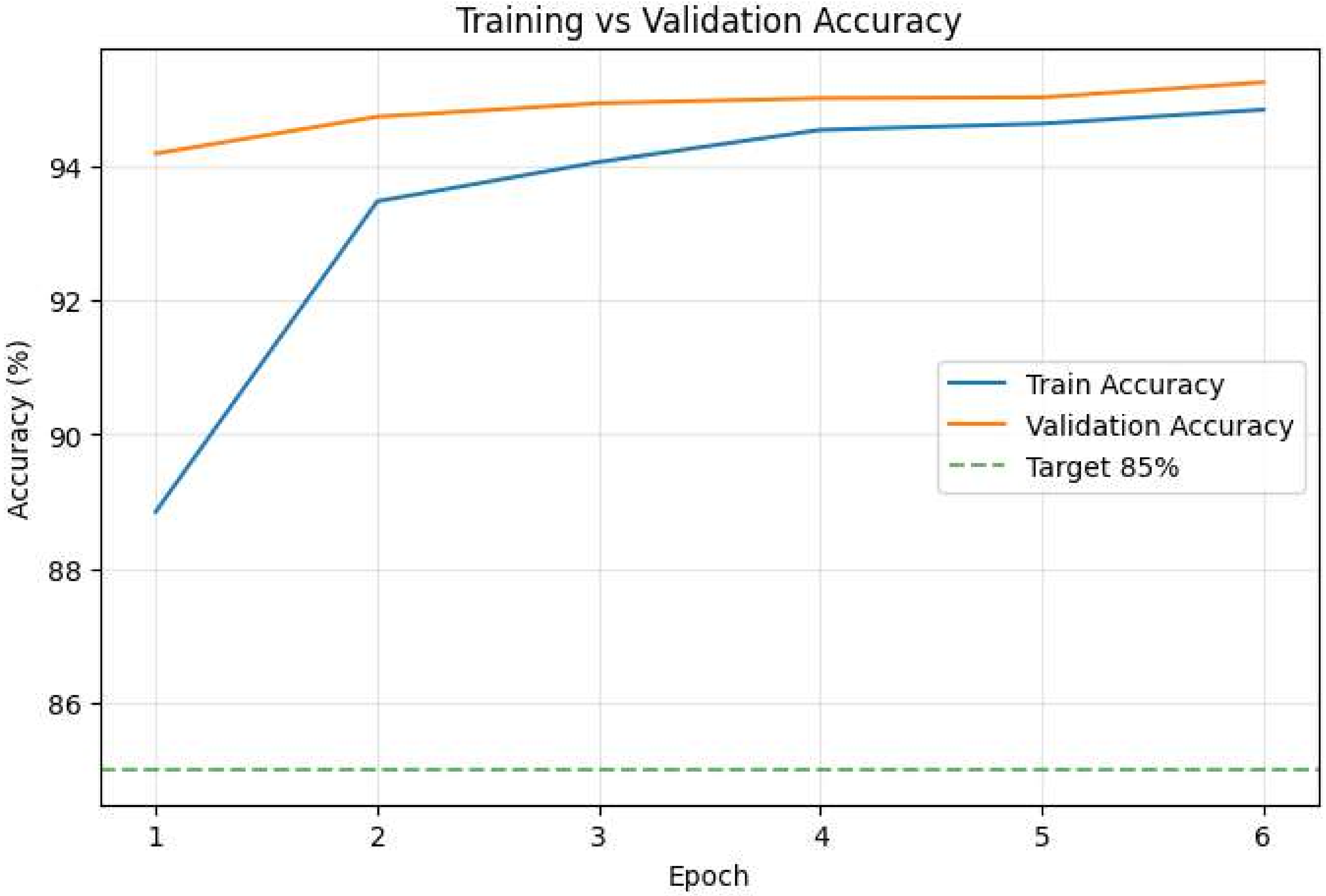
```
epochs_range = range(1, EPOCHS + 1)

plt.figure(figsize=(8, 5))
plt.plot(epochs_range, [a*100 for a in history["train_acc"]], label="Train Accuracy")
plt.plot(epochs_range, [a*100 for a in history["val_acc"]], label="Validation Accuracy")
plt.axhline(y=85, linestyle="--", color="green", alpha=0.6, label="Target 85%")
plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")
plt.title("Training vs Validation Accuracy")
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```

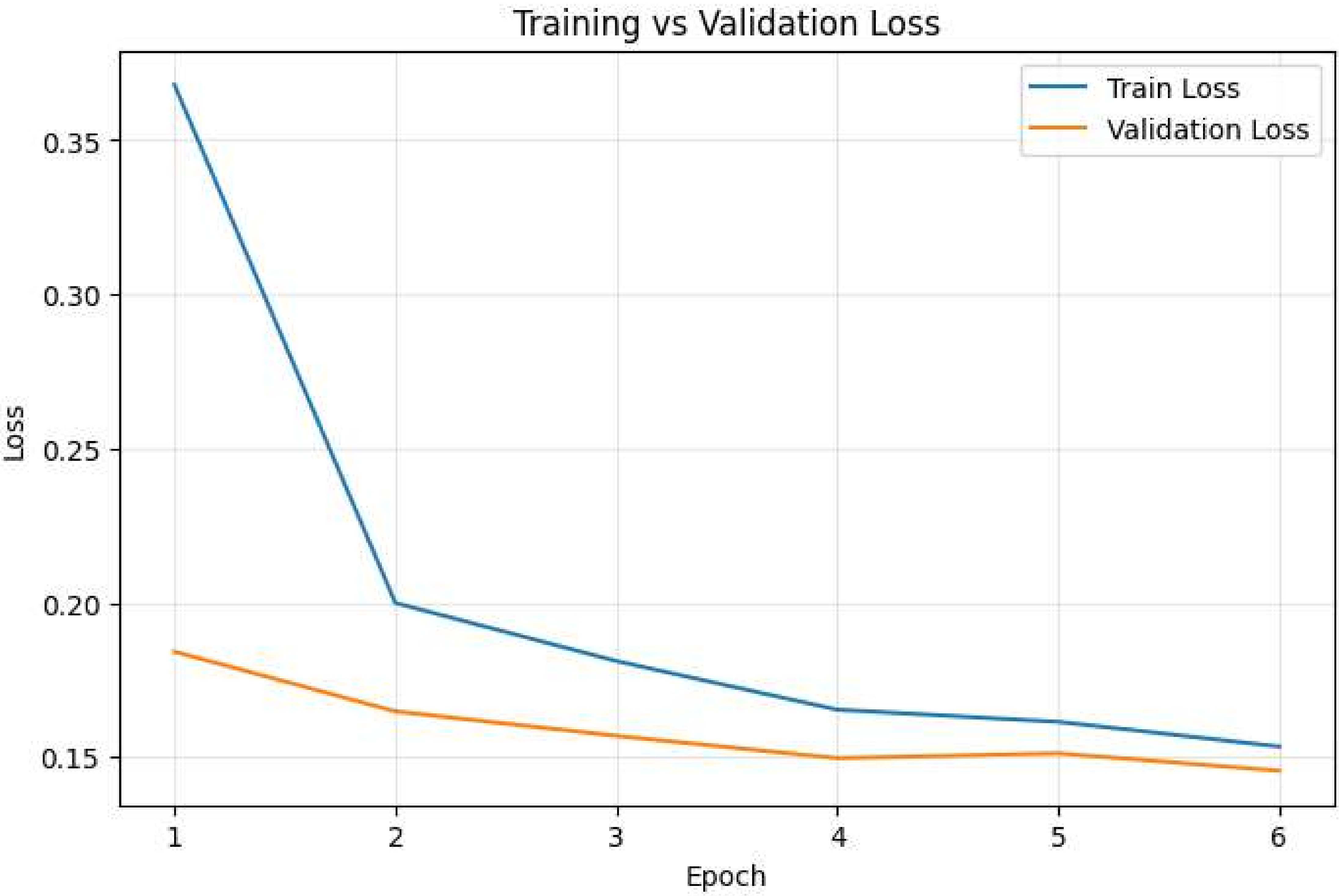


```
plt.figure(figsize=(8, 5))
plt.plot(epochs_range, history["train_loss"], label="Train Loss")
plt.plot(epochs_range, history["val_loss"], label="Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training vs Validation Loss")
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```



```
plt.plot(epochs_range, history["val_loss"], label="Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training vs Validation Loss")
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```


Training vs Validation Accuracy
Accuracy (%)
94
92
90
88
86
Epoch
1
2
3
4
5
6
Train Accuracy
Validation Accuracy
Target 85%


Training vs Validation Loss
Train Loss
Validation Loss
Loss
0.35
0.30
0.25
0.20
0.15
1
2
3
4
5
6
Epoch