```
# Step 1: Install required libraries (Colab usually has these)
!pip install torch torchvision matplotlib --quiet
```

```
# Step 2: Import libraries
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader, random_split
import matplotlib.pyplot as plt
```

```
# Step 3: Data preprocessing
transform = transforms.Compose([
    transforms.Resize((224, 224)),  # ResNet expects 224x224 images
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5],
                         std=[0.5, 0.5, 0.5])
])

# Download dataset
dataset = datasets.CIFAR10(root="./data", train=True, download=True, transform=transform)
test_dataset = datasets.CIFAR10(root="./data", train=False, download=True, transform=transform)

# Train / Validation split (80/10)
train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

print("Dataset loaded successfully")
```

```
100%|██████████| 170M/170M [00:02<00:00, 57.6MB/s]
Dataset loaded successfully
```

```
# Step 4: Load pre-trained ResNet18
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = models.resnet18(pretrained=True)

# Replace final layer (because CIFAR-10 has 10 classes)
model.fc = nn.Linear(model.fc.in_features, 10)

model = model.to(device)

print("Model loaded on", device)
```

```
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:208: UserWarning: The paramet
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments c
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub
100%|██████████| 44.7M/44.7M [00:00<00:00, 184MB/s]
Model loaded on cuda
```

```
# Step 5: Loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)
```

```
# Step 6: Training loop
epochs = 5

train_acc, val_acc = [], []

for epoch in range(epochs):
    model.train()
    correct, total = 0, 0

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

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    train_accuracy = 100 * correct / total
    train_acc.append(train_accuracy)

    # Validation
    model.eval()
    correct, total = 0, 0
    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    val_accuracy = 100 * correct / total
    val_acc.append(val_accuracy)

    print(f"Epoch {epoch+1}/{epochs} | Train Acc: {train_accuracy:.2f}% | Val Acc: {val_accuracy:
```

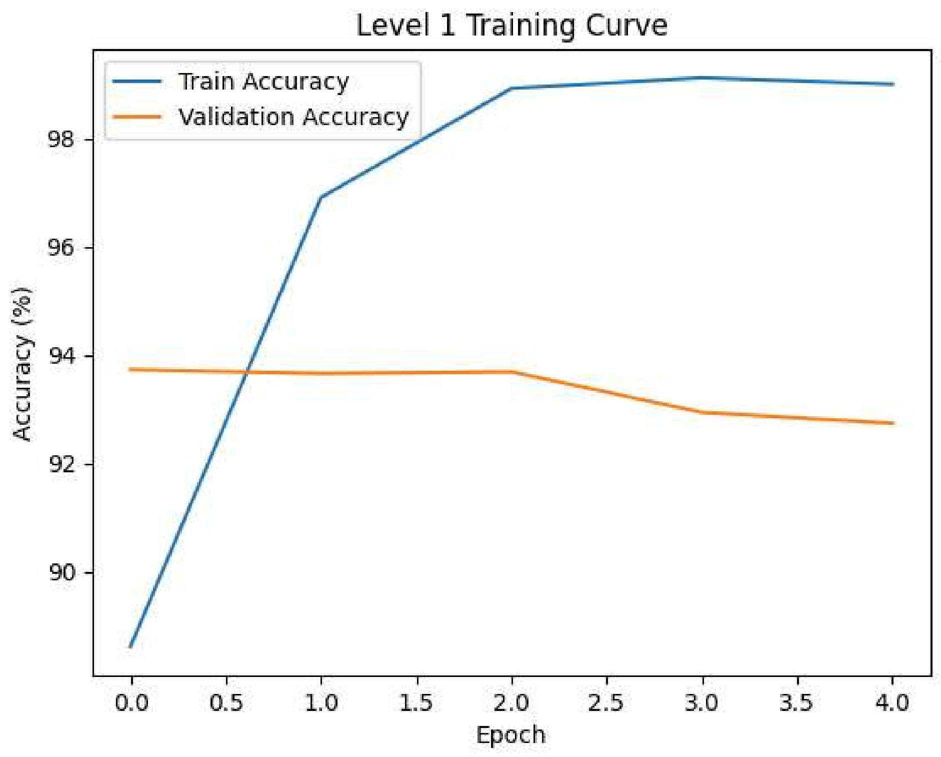
```
Epoch 1/5 | Train Acc: 88.62% | Val Acc: 93.73%
Epoch 2/5 | Train Acc: 96.91% | Val Acc: 93.66%
Epoch 3/5 | Train Acc: 98.92% | Val Acc: 93.69%
Epoch 4/5 | Train Acc: 99.12% | Val Acc: 92.94%
Epoch 5/5 | Train Acc: 99.00% | Val Acc: 92.74%
```

```
# Step 7: Test evaluation
model.eval()
correct, total = 0, 0

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

```
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    test_accuracy = 100 * correct / total
    print(f"Test Accuracy: {test_accuracy:.2f}%")
```

```
Test Accuracy: 92.16%
```

```
# Step 8: Plot training curve
plt.plot(train_acc, label="Train Accuracy")
plt.plot(val_acc, label="Validation Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")
plt.legend()
plt.title("Level 1 Training Curve")
plt.show()
```


Level 1 Training Curve
Train Accuracy
Validation Accuracy
Accuracy (%)
Epoch