```
import torch
import torchvision
print(torch.__version__)
print(torchvision.__version__)
```

```
2.9.0+cu126
0.24.0+cu126
```

```
import torch

print("GPU available:", torch.cuda.is_available())
if torch.cuda.is_available():
    print("GPU name:", torch.cuda.get_device_name(0))
```

```
GPU available: True
GPU name: Tesla T4
```

```
from torchvision import datasets, transforms

# Transform images to fit ResNet50 input
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])

# Download and load CIFAR-10
train_data = datasets.CIFAR10(
    root="./data",
    train=True,
    download=True,
    transform=transform
)

test_data = datasets.CIFAR10(
```

```
    root="./data",
    train=False,
    download=True,
    transform=transform
)

print("Training samples:", len(train_data))
print("Test samples:", len(test_data))
```

```
100%|██████████| 170M/170M [00:05<00:00, 30.2MB/s]
Training samples: 50000
Test samples: 10000
```

```
from torch.utils.data import random_split

# 80% train, 20% temp
train_size = int(0.8 * len(train_data))
val_size = len(train_data) - train_size

train_dataset, val_dataset = random_split(
    train_data, [train_size, val_size]
)

print("Train samples:", len(train_dataset))
print("Validation samples:", len(val_dataset))
```

```
Train samples: 40000
Validation samples: 10000
```

```
import torch
import torch.nn as nn
from torchvision import models

# Load pretrained ResNet50
model = models.resnet50(pretrained=True)
```

```
# Freeze all layers (important for Level-1)
for param in model.parameters():
    param.requires_grad = False

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

# Move model to GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

print("Model loaded and moved to:", device)
```

```
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretraine
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a w
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /root/.cache/torch/hub/checkpoints/
100%|██████████| 97.8M/97.8M [00:00<00:00, 163MB/s]
Model loaded and moved to: cuda
```

```
from torch.utils.data import DataLoader

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

print("Train batches:", len(train_loader))
print("Validation batches:", len(val_loader))
print("Test batches:", len(test_loader))
```

```
Train batches: 625
Validation batches: 157
Test batches: 157
```

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()

# Train ONLY final layer (correct for Level-1)
optimizer = optim.Adam(model.fc.parameters(), lr=0.001)

print("Loss and optimizer defined")
```

```
Loss and optimizer defined
```

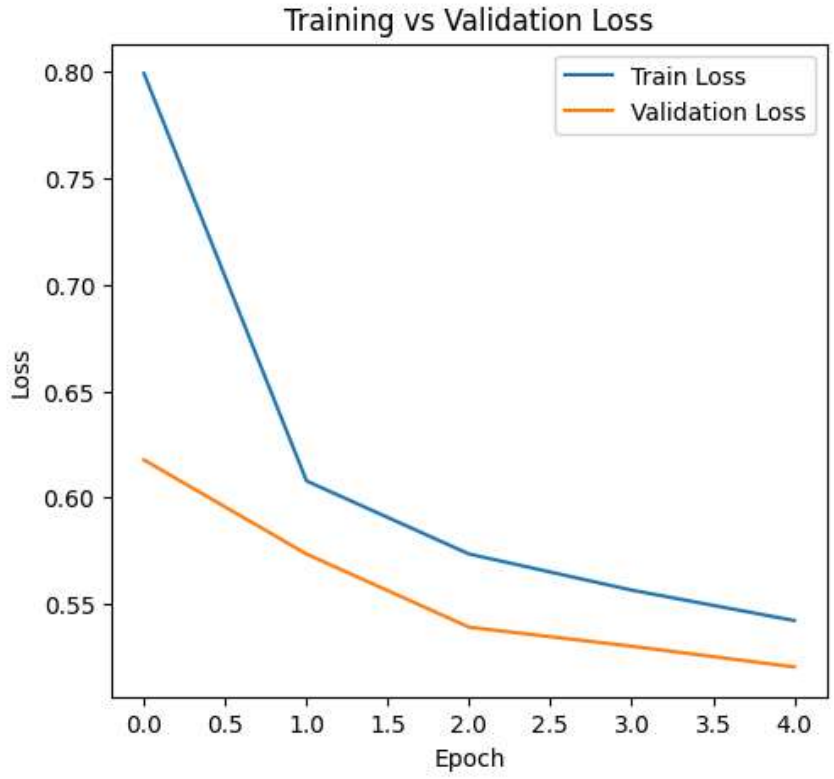
```
epochs = 5

train_losses, val_losses = [], []
train_accs, val_accs = [], []

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

        running_loss += loss.item()
        _, preds = outputs.max(1)
        total += labels.size(0)
        correct += preds.eq(labels).sum().item()

    train_loss = running_loss / len(train_loader)
    train_acc = 100 * correct / total
```

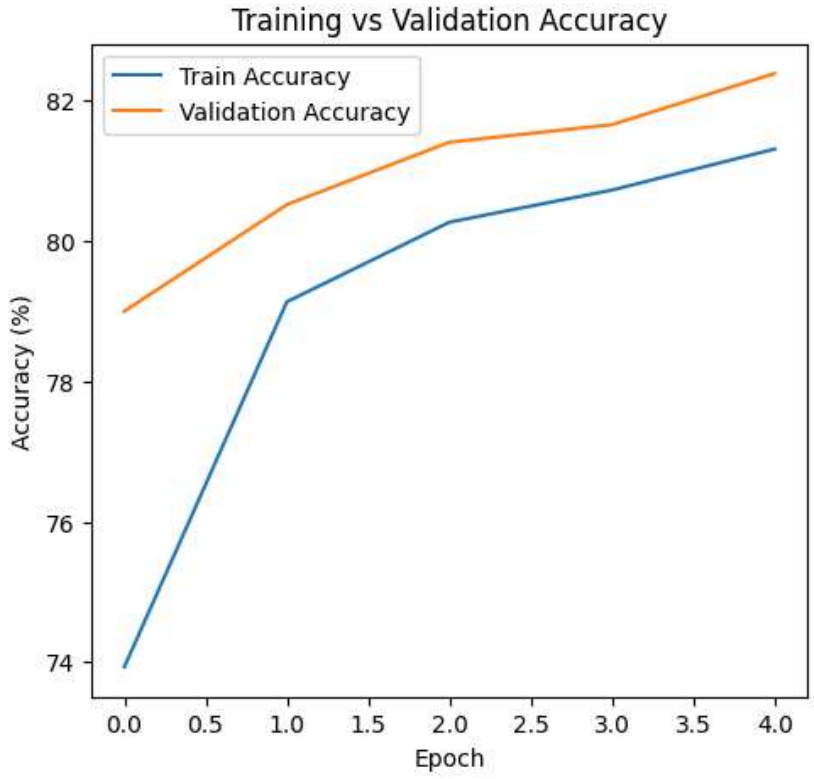
```
    # -------- Validation --------
    model.eval()
    correct, total, running_loss = 0, 0, 0.0

    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)

            running_loss += loss.item()
            _, preds = outputs.max(1)
            total += labels.size(0)
            correct += preds.eq(labels).sum().item()

    val_loss = running_loss / len(val_loader)
    val_acc = 100 * correct / total

    train_losses.append(train_loss)
    val_losses.append(val_loss)
    train_accs.append(train_acc)
    val_accs.append(val_acc)

    print(f"Epoch [{epoch+1}/{epochs}] | "
          f"Train Acc: {train_acc:.2f}% | Val Acc: {val_acc:.2f}%")
```

```
Epoch [1/5] | Train Acc: 73.93% | Val Acc: 79.00%
Epoch [2/5] | Train Acc: 79.14% | Val Acc: 80.52%
Epoch [3/5] | Train Acc: 80.27% | Val Acc: 81.41%
Epoch [4/5] | Train Acc: 80.73% | Val Acc: 81.66%
Epoch [5/5] | Train Acc: 81.31% | Val Acc: 82.39%
```

```
import matplotlib.pyplot as plt

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

```
    # Loss Curve
    plt.subplot(1,2,1)
    plt.plot(train_losses, label="Train Loss")
    plt.plot(val_losses, label="Validation Loss")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.title("Training vs Validation Loss")
    plt.legend()

    # Accuracy Curve
    plt.subplot(1,2,2)
    plt.plot(train_accs, label="Train Accuracy")
    plt.plot(val_accs, label="Validation Accuracy")
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy (%)")
    plt.title("Training vs Validation Accuracy")
    plt.legend()

    plt.show()
```






```
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)
        _, preds = outputs.max(1)
        total += labels.size(0)
        correct += preds.eq(labels).sum().item()

test_accuracy = 100 * correct / total
```

```
print(f"Final Test Accuracy: {test_accuracy:.2f}%")
```

```
Final Test Accuracy: 81.74%
```

```
from torchvision import transforms

# Data Augmentation for Training
train_transform_aug = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.ToTensor()
])

# No augmentation for validation & test
test_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])
```

```
from torchvision import datasets

train_data_aug = datasets.CIFAR10(
    root="./data",
    train=True,
    download=False,
    transform=train_transform_aug
)

test_data_aug = datasets.CIFAR10(
    root="./data",
    train=False,
    download=False,
    transform=test_transform
```

```
)
```

```
from torch.utils.data import random_split

train_size = int(0.8 * len(train_data_aug))
val_size = len(train_data_aug) - train_size

train_dataset_aug, val_dataset_aug = random_split(
    train_data_aug, [train_size, val_size]
)

print(len(train_dataset_aug), len(val_dataset_aug))
```

```
40000 10000
```

```
from torch.utils.data import DataLoader

train_loader_aug = DataLoader(train_dataset_aug, batch_size=64, shuffle=True)
val_loader_aug = DataLoader(val_dataset_aug, batch_size=64, shuffle=False)
test_loader_aug = DataLoader(test_data_aug, batch_size=64, shuffle=False)
```

```
from torchvision import models
import torch.nn as nn

model_aug = models.resnet50(pretrained=True)

for param in model_aug.parameters():
    param.requires_grad = False

model_aug.fc = nn.Linear(model_aug.fc.in_features, 10)
model_aug = model_aug.to(device)
```

```
    import torch.optim as optim

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model_aug.fc.parameters(), lr=0.001)

    epochs = 5

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

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

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

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

        train_acc = 100 * correct / total

        model_aug.eval()
        correct, total = 0, 0
        with torch.no_grad():
            for images, labels in val_loader_aug:
                images, labels = images.to(device), labels.to(device)
                outputs = model_aug(images)
                _, preds = outputs.max(1)
                total += labels.size(0)
                correct += preds.eq(labels).sum().item()

        val_acc = 100 * correct / total
```

```
    print(f"Epoch {epoch+1} | Train Acc: {train_acc:.2f}% | Val Acc: {val_acc:.2f}%")
```

```
Epoch 1 | Train Acc: 78.27% | Val Acc: 77.17%
Epoch 2 | Train Acc: 78.46% | Val Acc: 78.95%
Epoch 3 | Train Acc: 78.56% | Val Acc: 79.18%
Epoch 4 | Train Acc: 78.73% | Val Acc: 79.94%
Epoch 5 | Train Acc: 79.33% | Val Acc: 80.32%
```

```
model_aug.eval()
correct, total = 0, 0

with torch.no_grad():
    for images, labels in test_loader_aug:
        images, labels = images.to(device), labels.to(device)
        outputs = model_aug(images)
        _, preds = outputs.max(1)
        total += labels.size(0)
        correct += preds.eq(labels).sum().item()

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

```
Test Accuracy with Augmentation: 84.06%
```

```
# Unfreeze only the last ResNet block (layer4)
for name, param in model_aug.named_parameters():
    if "layer4" in name:
        param.requires_grad = True
```

```
import torch.optim as optim

optimizer_ft = optim.Adam(
    filter(lambda p: p.requires_grad, model_aug.parameters()),
    lr=1e-4   # small LR for fine-tuning
```

```
)
```

```
epochs = 3   # short training is enough for Level-3

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

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

        optimizer_ft.zero_grad()
        outputs = model_aug(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer_ft.step()

        running_loss += loss.item()
        _, preds = outputs.max(1)
        total += labels.size(0)
        correct += preds.eq(labels).sum().item()

    train_acc = 100 * correct / total

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

    val_acc = 100 * correct / total
```

```
    print(f"[Level-3] Epoch {epoch+1} | Train Acc: {train_acc:.2f}% | Val Acc: {val_acc:.2f}%")
```

```
[Level-3] Epoch 1 | Train Acc: 85.15% | Val Acc: 88.15%
[Level-3] Epoch 2 | Train Acc: 90.51% | Val Acc: 89.80%
[Level-3] Epoch 3 | Train Acc: 92.62% | Val Acc: 90.86%
```

```
model_aug.eval()
correct, total = 0, 0

with torch.no_grad():
    for images, labels in test_loader_aug:
        images, labels = images.to(device), labels.to(device)
        outputs = model_aug(images)
        _, preds = outputs.max(1)
        total += labels.size(0)
        correct += preds.eq(labels).sum().item()

test_acc_level3 = 100 * correct / total
print(f"Level-3 Test Accuracy (Fine-tuned): {test_acc_level3:.2f}%")
```

```
Level-3 Test Accuracy (Fine-tuned): 91.70%
```

```
pip install grad-cam
```

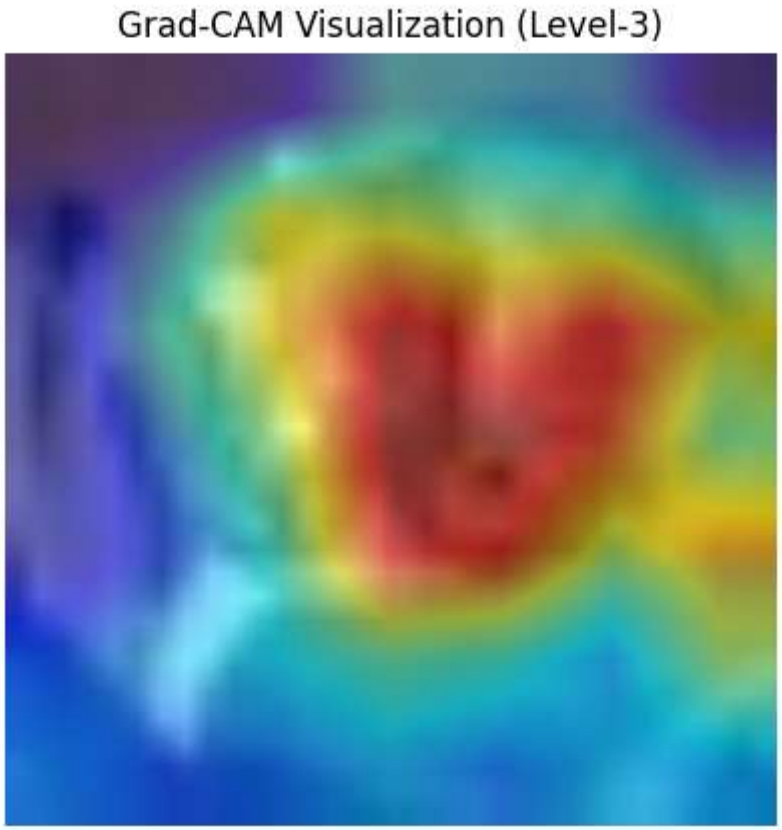
```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 78.9 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from grad-cam) (2.0.2)
Requirement already satisfied: Pillow in /usr/local/lib/python3.12/dist-packages (from grad-cam) (11.3.0)
Requirement already satisfied: torch>=1.7.1 in /usr/local/lib/python3.12/dist-packages (from grad-cam) (2.9.0+
Requirement already satisfied: torchvision>=0.8.2 in /usr/local/lib/python3.12/dist-packages (from grad-cam)
Collecting ttach (from grad-cam)
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.12/dist-packages (from grad-cam) (4.12
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (from grad-cam) (3.10.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (from grad-cam) (1.6.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from torch>=1.7.1->grad-ca
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.12/dist-packages (from torc
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch>=1.7.1->grad-
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch>=1.7.1->gr
Requirement already satisfied: networkx>=2.5.1 in /usr/local/lib/python3.12/dist-packages (from torch>=1.7.1->
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch>=1.7.1->grad-cam)
Requirement already satisfied: fsspec>=0.8.5 in /usr/local/lib/python3.12/dist-packages (from torch>=1.7.1->gr
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (fro
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packages (fro
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from t
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from t
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from to
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in /usr/local/lib/python3.12/dist-packages (from torch
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in /usr/local/lib/python3.12/dist-packages (from to
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torc
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from t
Requirement already satisfied: triton==3.5.0 in /usr/local/lib/python3.12/dist-packages (from torch>=1.7.1->gr
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib->g
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib->grad-
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib->
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib->
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib->gr
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib->g
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotli
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn->gra
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn->gr
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-le
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7-
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy>=1.13
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torch>
Downloading ttach-0.0.3-py3-none-any.whl (9.8 kB)
```

```
  Created wheel for grad-cam: filename=grad_cam-1.5.5-py3-none-any.whl size=44284 sha256=e40e18b00f9995fed291
  Stored in directory: /root/.cache/pip/wheels/fb/3b/09/2afc520f3d69bc26ae6bd87416759c820a3f7d05c1a077bbf6
Successfully built grad-cam
Installing collected packages: ttach, grad-cam
```

```
from pytorch_grad_cam import GradCAM
from pytorch_grad_cam.utils.model_targets import ClassifierOutputTarget
from pytorch_grad_cam.utils.image import show_cam_on_image
import matplotlib.pyplot as plt
import numpy as np
```

```
# Target last convolution layer
target_layers = [model_aug.layer4[-1]]
cam = GradCAM(model=model_aug, target_layers=target_layers)

# Take one test image
image, label = test_data[0]
input_tensor = image.unsqueeze(0).to(device)

targets = [ClassifierOutputTarget(label)]
grayscale_cam = cam(input_tensor=input_tensor, targets=targets)[0]

# Convert image for visualization
img = image.permute(1, 2, 0).numpy()
visualization = show_cam_on_image(img, grayscale_cam, use_rgb=True)

plt.figure(figsize=(5,5))
plt.imshow(visualization)
plt.title("Grad-CAM Visualization (Level-3)")
plt.axis("off")
plt.show()
```

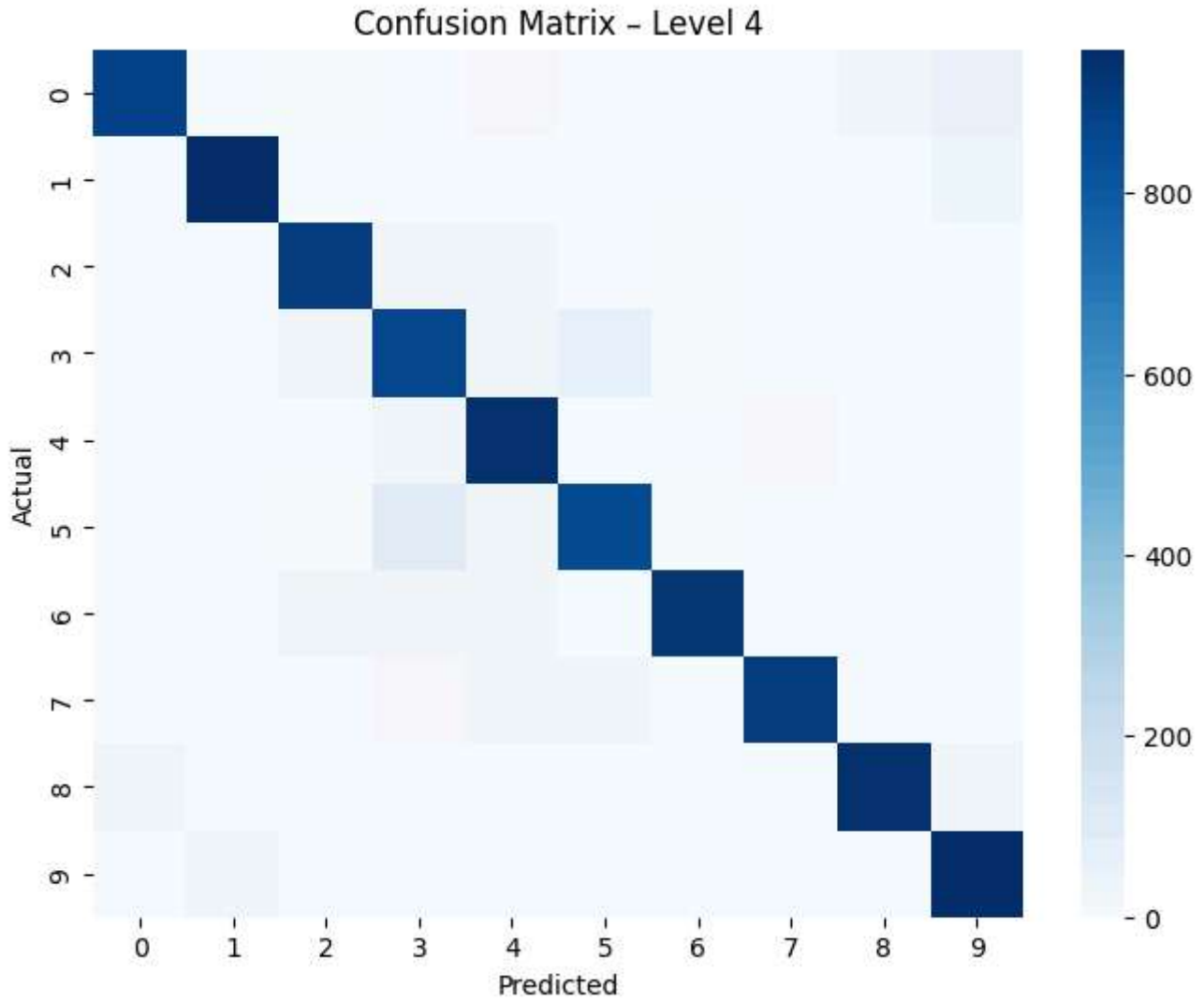



```
from torch.optim.lr_scheduler import StepLR

scheduler = StepLR(
    optimizer_ft,   # optimizer from Level 3
    step_size=2,
    gamma=0.1
)
```

```
scheduler.step()
```

```
/usr/local/lib/python3.12/dist-packages/torch/optim/lr_scheduler.py:192: UserWarning: Detected call of `lr_sche
  warnings.warn(
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

all_preds = []
all_labels = []

model_aug.eval()
with torch.no_grad():
    for images, labels in test_loader_aug:
        images, labels = images.to(device), labels.to(device)
        outputs = model_aug(images)
        _, preds = outputs.max(1)

        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

cm = confusion_matrix(all_labels, all_preds)

plt.figure(figsize=(8,6))
sns.heatmap(cm, cmap="Blues")
plt.title("Confusion Matrix – Level 4")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



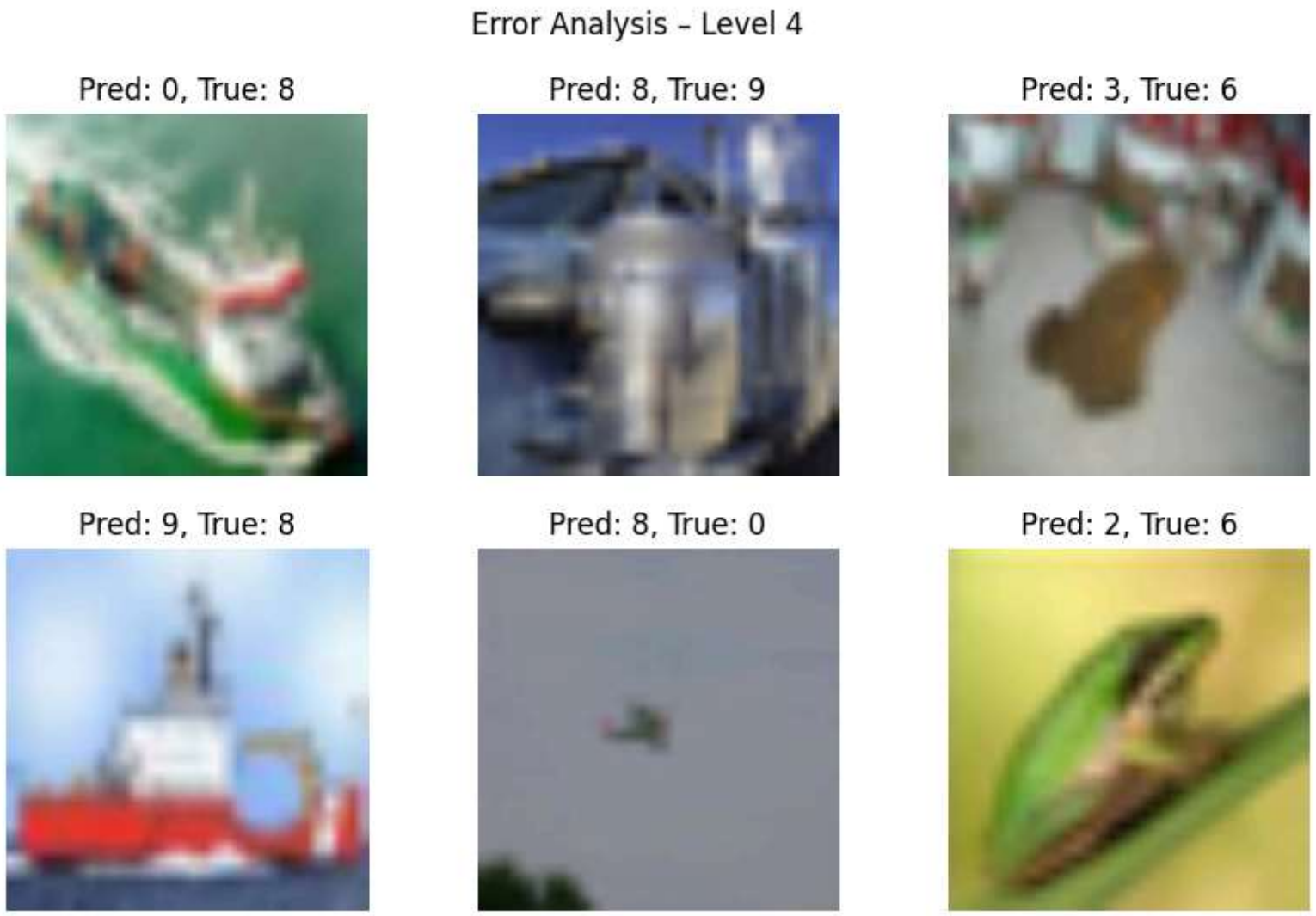

```
class_correct = [0]*10
class_total = [0]*10

for i in range(len(all_labels)):
    label = all_labels[i]
    pred = all_preds[i]
    if label == pred:
        class_correct[label] += 1
```

```
        class_total[label] += 1

for i in range(10):
    print(f"Class {i} Accuracy: {100 * class_correct[i] / class_total[i]:.2f}%")
```

```
Class 0 Accuracy: 88.40%
Class 1 Accuracy: 95.20%
Class 2 Accuracy: 91.30%
Class 3 Accuracy: 86.80%
Class 4 Accuracy: 94.70%
Class 5 Accuracy: 85.80%
Class 6 Accuracy: 92.70%
Class 7 Accuracy: 91.30%
Class 8 Accuracy: 94.90%
Class 9 Accuracy: 95.90%
```

```
count = 0
plt.figure(figsize=(10,6))

for images, labels in test_loader_aug:
    images, labels = images.to(device), labels.to(device)
    outputs = model_aug(images)
    _, preds = outputs.max(1)

    for i in range(len(labels)):
        if preds[i] != labels[i] and count < 6:
            img = images[i].cpu().permute(1,2,0)
            plt.subplot(2,3,count+1)
            plt.imshow(img)
            plt.title(f"Pred: {preds[i].item()}, True: {labels[i].item()}")
            plt.axis("off")
            count += 1

    if count >= 6:
        break

plt.suptitle("Error Analysis - Level 4")
```

```
plt.show()
```




```
# Save the trained model
torch.save(model_aug.state_dict(), "cifar10_resnet50_level5.pth")

print("Model saved successfully!")
```

```
Model saved successfully!
```

```
# Load model for inference
model_infer = models.resnet50(pretrained=False)
model_infer.fc = nn.Linear(model_infer.fc.in_features, 10)

model_infer.load_state_dict(
    torch.load("cifar10_resnet50_level5.pth", map_location=device)
)

model_infer = model_infer.to(device)
model_infer.eval()

print("Model loaded for inference!")
```

```
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretraine
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a w
  warnings.warn(msg)
Model loaded for inference!
```

```
import torch.nn.functional as F

classes = ['airplane','automobile','bird','cat','deer',
           'dog','frog','horse','ship','truck']
```