```
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))
```

```
                                      images.to
                                       <built-in method to of Tensor object at 0x7bd6972d9400>
```

```
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
                                     images.to

# Load pretrained ResNet50            <built-in method to of Tensor object at 0x7bd6972d9400>
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 'pretrained' is deprecated since 0.13 and may be removed in the future, pl
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /root/.cache/torch/hub/checkpoints/resnet50-0676ba61.pth
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 = [], []
```

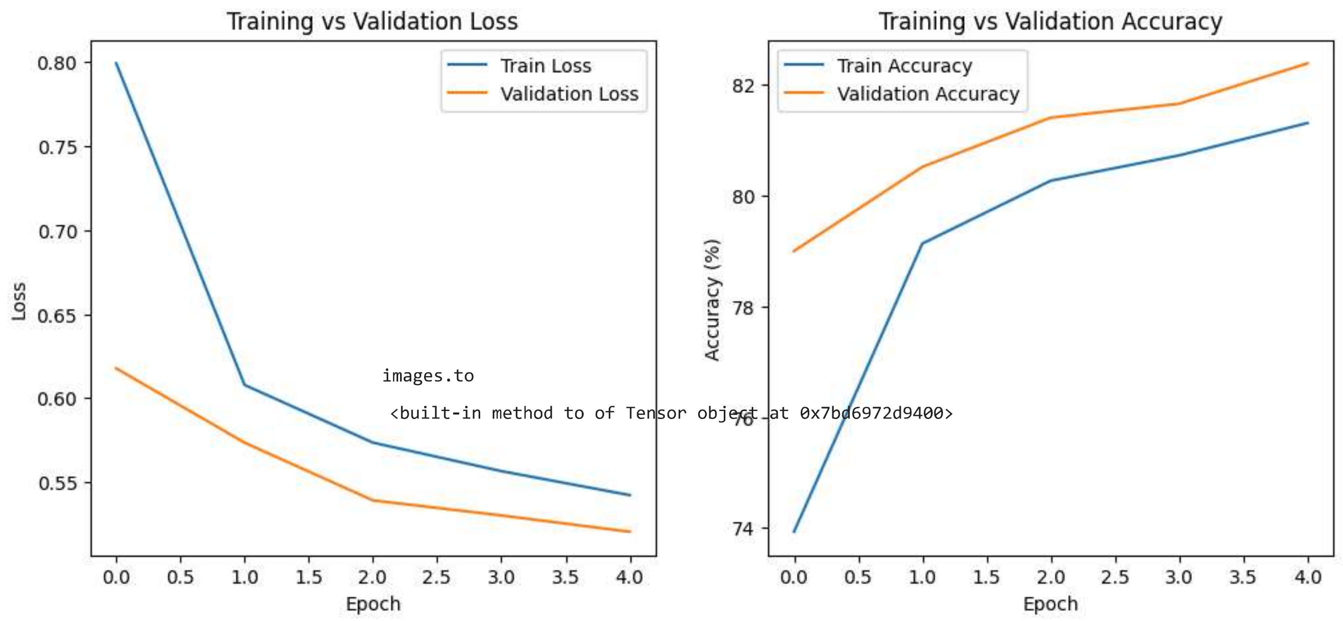
```
images.to

<built-in method to of Tensor object at 0x7bd6972d9400>
```

```
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()
```

```
images.to
 <built-in method to of Tensor object at 0x7bd6972d9400>
```



```
images.to

<built-in method to of Tensor object at 0x7bd6972d9400>
```

```
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
)
```

```
images.to
 <built-in method to of Tensor object at 0x7bd6972d9400>
```

```
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()              images.to
        optimizer.step()
                                      <built-in method to of Tensor object at 0x7bd6972d9400>
        _, 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%
```

```
images.to
```

```
<built-in method to of Tensor object at 0x7bd6972d9400>
```