```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from sklearn.metrics import classification_report
import time
import matplotlib.pyplot as plt
```

```
def prepare_data():
    transform_train = transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.RandomCrop(32, padding=4),
        transforms.ToTensor(),
        transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
    ])
    transform_test = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
    ])
    train_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_train)
    test_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform_test)
    train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True, num_workers=2)
    test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False, num_workers=2)
    return train_loader, test_loader
```

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.classifier = nn.Sequential(
            nn.Linear(128 * 4 * 4, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, 10)
        )
    def forward(self, x):
```

```
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x
```

```
def train_model(model, train_loader, criterion, optimizer, device):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for inputs, targets in train_loader:
        inputs, targets = inputs.to(device), targets.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
        correct += predicted.eq(targets).sum().item()
    return running_loss / len(train_loader), 100. * correct / total

def evaluate_model(model, test_loader, criterion, device):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0
    all_targets = []
    all_predictions = []
    with torch.no_grad():
        for inputs, targets in test_loader:
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            running_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()
            all_targets.extend(targets.cpu().numpy())
            all_predictions.extend(predicted.cpu().numpy())
    return running_loss / len(test_loader), 100. * correct / total, classification_report(all_targets, all_predictions)
```

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
train_loader, test_loader = prepare_data()
model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
epochs = 10
for epoch in range(epochs):
```

```
    train_loss, train_acc = train_model(model, train_loader, criterion, optimizer, device)
    test_loss, test_acc, report = evaluate_model(model, test_loader, criterion, device)
    print(f"Epoch {epoch+1}/{epochs} -> Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}%, Test Loss: {test_loss:.4f}, Test Acc: {test_acc:.2f}%")
torch.save(model.state_dict(), "cnnwithlatency_model.pth")
print("Model saved successfully.")
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170M/170M [00:03<00:00, 55.9MB/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
Epoch 1/10 -> Train Loss: 1.6080, Train Acc: 40.54%, Test Loss: 1.1786, Test Acc: 57.55%
Epoch 2/10 -> Train Loss: 1.2299, Train Acc: 56.07%, Test Loss: 0.9890, Test Acc: 65.14%
Epoch 3/10 -> Train Loss: 1.0702, Train Acc: 62.25%, Test Loss: 0.9027, Test Acc: 68.17%
Epoch 4/10 -> Train Loss: 0.9681, Train Acc: 65.96%, Test Loss: 0.8197, Test Acc: 70.73%
Epoch 5/10 -> Train Loss: 0.9032, Train Acc: 68.48%, Test Loss: 0.7817, Test Acc: 72.75%
Epoch 6/10 -> Train Loss: 0.8547, Train Acc: 70.18%, Test Loss: 0.7391, Test Acc: 73.91%
Epoch 7/10 -> Train Loss: 0.8204, Train Acc: 71.48%, Test Loss: 0.7158, Test Acc: 75.30%
Epoch 8/10 -> Train Loss: 0.7858, Train Acc: 72.95%, Test Loss: 0.6719, Test Acc: 76.47%
Epoch 9/10 -> Train Loss: 0.7654, Train Acc: 73.67%, Test Loss: 0.6555, Test Acc: 77.42%
Epoch 10/10 -> Train Loss: 0.7371, Train Acc: 74.60%, Test Loss: 0.6412, Test Acc: 77.81%
Model saved successfully.
```

```
model = CNN().to(device)
model.load_state_dict(torch.load("cnnwithlatency_model.pth"))
model.eval()

def measure_latency(model, device):
    model = model.to(device)
    model.eval()
    dummy_input = torch.randn(1, 3, 32, 32).to(device)
    if device == 'cuda':
        start_event = torch.cuda.Event(enable_timing=True)
        end_event = torch.cuda.Event(enable_timing=True)
        start_event.record()
        model(dummy_input)
        end_event.record()
        torch.cuda.synchronize()
        latency = start_event.elapsed_time(end_event)
        print(f"GPU Latency: {latency:.2f} ms")
    else:
        start_time = time.time()
        model(dummy_input)
        end_time = time.time()
        latency = (end_time - start_time) * 1000
        print(f"CPU Latency: {latency:.2f} ms")

measure_latency(model, 'cpu')
if torch.cuda.is_available():
    measure_latency(model, 'cuda')
```

```
<ipython-input-6-f4e00528cf0c>:2: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicit
  model.load_state_dict(torch.load("cnnwithlatency_model.pth"))
CPU Latency: 56.53 ms
GPU Latency: 141.10 ms
```

```
!ls -lh
```

```
total 2.4M
-rw-r--r-- 1 root root 2.4M Dec 31 10:32 cnnwithlatency_model.pth
drwxr-xr-x 3 root root 4.0K Dec 31 10:28 data
drwxr-xr-x 1 root root 4.0K Dec 19 14:20 sample_data
```

```
!pip install onnx
```

```
Collecting onnx
  Downloading onnx-1.17.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (16 kB)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from onnx) (1.26.4)
Requirement already satisfied: protobuf>=3.20.2 in /usr/local/lib/python3.10/dist-packages (from onnx) (4.25.5)
Downloading onnx-1.17.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.0/16.0 MB 90.2 MB/s eta 0:00:00
Installing collected packages: onnx
Successfully installed onnx-1.17.0
```

```
dummy_input = torch.randn(1, 3, 32, 32).to(device)  # Example input for the model
torch.onnx.export(
    model,
    dummy_input,
    "cnnwithlatency_model.onnx",  # File name for ONNX
    input_names=["input"],  # Name of input layer
    output_names=["output"],  # Name of output layer
    opset_version=11  # ONNX opset version
)
print("ONNX model exported as 'cnnwithlatency_model.onnx'.")
```

```
ONNX model exported as 'cnnwithlatency_model.onnx'.
```

```
from google.colab import files

# Download the ONNX model
files.download("cnnwithlatency_model.onnx")
```

```
```

Start coding or generate with AI.