```
from torchvision.datasets import CIFAR100
from torchvision import transforms
from torch.utils.data import DataLoader

# Transformaciones: Convertir imágenes a tensores y normalizar
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Normalización para 3 canales (RGB)
])

# Cargar CIFAR100 dataset usando torchvision
train_dataset = CIFAR100(root="./data", train=True, transform=transform, download=True)
test_dataset = CIFAR100(root="./data", train=False, transform=transform, download=True)

# Cargar los datos en batches
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

```
import torch.nn as nn
import torch

class AlexNet(nn.Module):
    def __init__(self, num_classes=100):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=2, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(64, 192, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((2, 2))
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 2 * 2, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x

model = AlexNet()
```

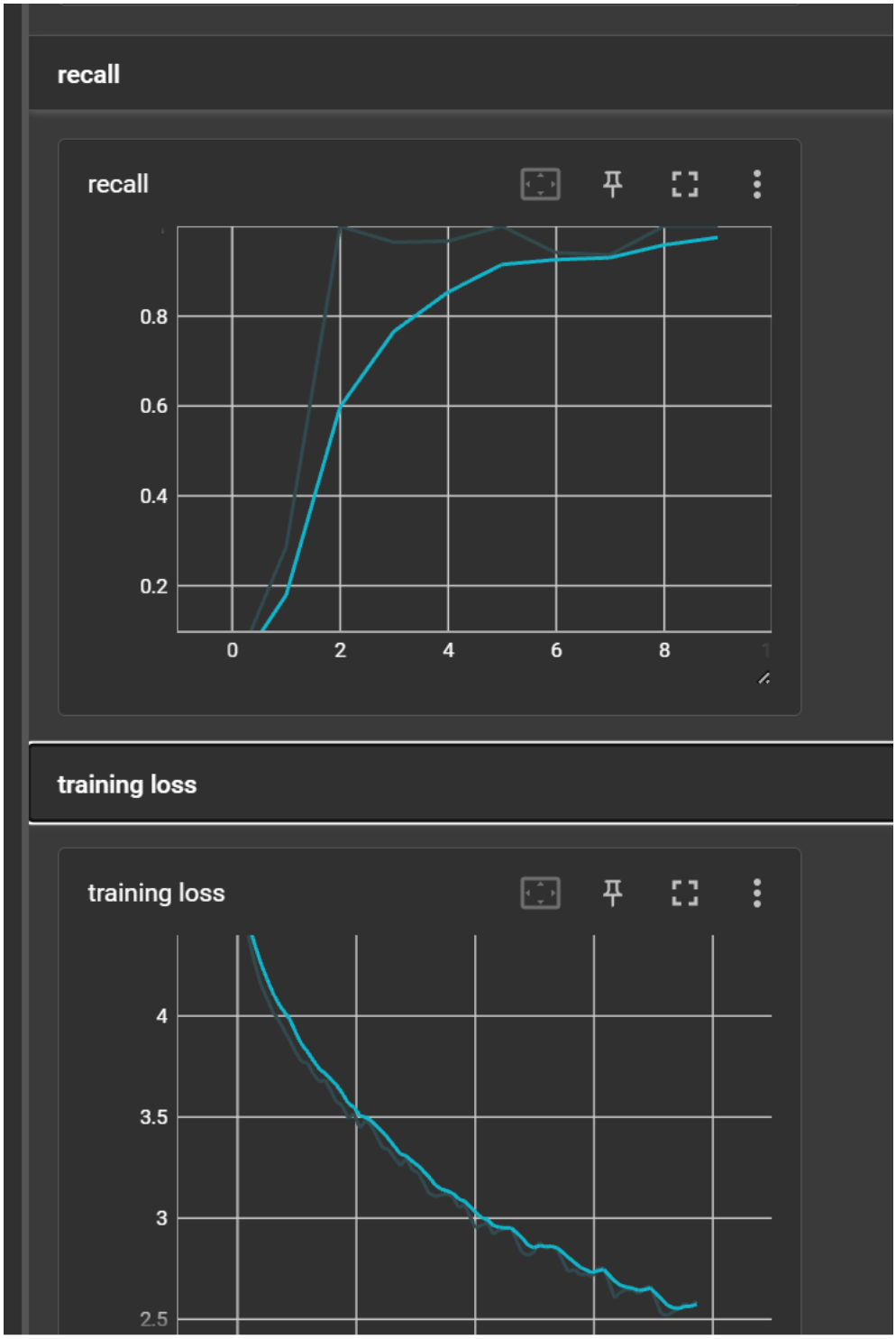
```
from torch.utils.tensorboard import SummaryWriter

writer = SummaryWriter('runs/alexnet_experiment_2')
```

```
import torch.optim as optim
import torch.nn.functional as F
import torch
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model.to(device)

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

# Añadir el modelo a TensorBoard
images, labels = next(iter(train_loader))

writer.add_graph(model, images.to(device))

num_epochs = 10

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for i, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)

        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        if i % 100 == 99:
            writer.add_scalar('training loss', running_loss / 100, epoch * len(train_loader) + i)
            running_loss = 0.0

    # Evaluación
    model.eval()
    total_correct = 0
    TP = 0
    TN = 0
    FP = 0
    FN = 0

    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            _, predicted = torch.max(output.data, 1)
            total_correct += (predicted == target).sum().item()

            # Calculo para clasificación binaria
            TP += ((predicted == 1) & (target == 1)).sum().item()
            TN += ((predicted == 0) & (target == 0)).sum().item()
            FP += ((predicted == 1) & (target == 0)).sum().item()
            FN += ((predicted == 0) & (target == 1)).sum().item()

    accuracy = total_correct / len(test_dataset)
    precision = TP / (TP + FP) if TP + FP != 0 else 0
    recall = TP / (TP + FN) if TP + FN != 0 else 0
    f1 = 2 * (precision * recall) / (precision + recall) if precision + recall != 0 else 0

    writer.add_scalar('accuracy', accuracy, epoch)
    writer.add_scalar('precision', precision, epoch)
    writer.add_scalar('recall', recall, epoch)
    writer.add_scalar('f1', f1, epoch)

    print(f"Epoch {epoch+1}/{num_epochs}, Accuracy: {accuracy:.4f}, Precision: {precision:.4f}, Recall: {recall:.4f}, F1-score: {f1:.4f}")

writer.close()
```

```
Epoch 1/10, Accuracy: 0.0772, Precision: 0.0000, Recall: 0.0000, F1-score: 0.0000
Epoch 2/10, Accuracy: 0.1322, Precision: 1.0000, Recall: 0.2857, F1-score: 0.4444
Epoch 3/10, Accuracy: 0.1803, Precision: 0.7500, Recall: 1.0000, F1-score: 0.8571
Epoch 4/10, Accuracy: 0.2183, Precision: 0.7714, Recall: 0.9643, F1-score: 0.8571
Epoch 5/10, Accuracy: 0.2446, Precision: 0.8056, Recall: 0.9667, F1-score: 0.8788
Epoch 6/10, Accuracy: 0.2680, Precision: 0.9677, Recall: 1.0000, F1-score: 0.9836
Epoch 7/10, Accuracy: 0.2805, Precision: 0.9143, Recall: 0.9412, F1-score: 0.9275
Epoch 8/10, Accuracy: 0.3020, Precision: 0.9565, Recall: 0.9362, F1-score: 0.9462
Epoch 9/10, Accuracy: 0.3053, Precision: 0.9250, Recall: 1.0000, F1-score: 0.9610
Epoch 10/10, Accuracy: 0.3140, Precision: 0.9074, Recall: 1.0000, F1-score: 0.9515
```

AlexNet Learning



Métrica de Desempeño:

Para problemas de clasificación, una métrica comúnmente utilizada es la precisión (accuracy). Esto mide el porcentaje de imágenes que el modelo clasifica correctamente. Es intuitiva y fácil de entender, y es apropiada para datasets como MNIST y CIFAR10 donde las clases están bastante equilibradas. Pero para AlexNet fue encesario usar el recall ya que el precision al ser un dataset desbalanceado no era muy representativo

Respuestas a las Preguntas:

a. Diferencia principal entre ambas arquitecturas:

LeNet-5 fue una de las primeras arquitecturas de redes neuronales convolucionales, diseñada principalmente para reconocer dígitos. Su diseño es más sencillo y tiene menos capas y parámetros. AlexNet, por otro lado, es mucho más profunda y fue diseñada para tratar con datasets de imágenes más complejos y de mayor resolución, como ImageNet. Utiliza más capas convolucionales, capas completamente conectadas más grandes y técnicas modernas como la activación ReLU y el dropout.

b. ¿Podría usarse LeNet-5 para un problema como el que resolvió usando AlexNet? ¿Y viceversa?

Sí, ambas redes pueden ser usadas para ambos datasets, pero la eficacia variará. Si usamos LeNet-5 en datasets más complicados como CIFAR10 o ImageNet, podría no tener el poder representacional suficiente para lograr un alto rendimiento debido a su simplicidad. AlexNet, aunque es más pesado, podría usarse para MNIST, pero es probable que sea excesivo y no tan eficiente en términos de recursos computacionales.

c. Qué le pareció más interesante de cada arquitectura:

LeNet-5: Es asombroso cómo esta arquitectura temprana, con su diseño sencillo, sentó las bases para las CNNs futuras. Su eficacia en la clasificación de dígitos a mano fue revolucionaria en su momento.
AlexNet: Lo que es destacable de AlexNet es cómo utilizó técnicas modernas y una arquitectura más profunda para lograr un rendimiento nunca antes visto en ImageNet, superando a otros enfoques tradicionales de procesamiento de imágenes. También es notable cómo popularizó las GPUs para entrenar redes neuronales, dado que fue entrenada usando GPUs.