Entrene un autoencoder para obtener una representación de baja dimensionalidad de las imágenes de MNIST. Use dichas representaciones para entrenar un perceptrón multicapa como clasificador. ¿Cuál es el tiempo de entrenamiento y la exactitud del clasificador obtenido cuando parte de la representación del autoencoder, en comparación con lo obtenido usando las imágenes originales?

En este caso se nos permitió usar Pytorch para la parte de código. La idea es definir la estructura general de la red y luego ir modificandola hasta lograr la mejor performance.

In [7]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
import torch.optim as optim
import matplotlib.pyplot as plt

# Elegir dispositivo
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device:", device)

# Transformación
transform = transforms.ToTensor()

# Dataset MNIST
train_dataset = datasets.MNIST(root="data", train=True, download=True, transform=transform)
test_dataset  = datasets.MNIST(root="data", train=False, download=True, transform=transform)

# minibatch
batch_size = 64
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader  = DataLoader(test_dataset, batch_size=len(test_dataset))
```

```
Device: cuda
```

Primero una función que entrena y grafica

In [8]:

```
def train_and_evaluate(model, train_loader, test_loader,
                       criterion, optimizer, device,
                       num_epochs=50, patience=5, min_delta=1e-4):

    model = model.to(device)
    best_loss = float("inf")
    epochs_no_improve = 0

    history = {"train_loss": [], "test_loss": [], "accuracy": []}

    total_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
    print(model)
    print(f"Parámetros entrenables: {total_params:,}")

    for epoch in range(num_epochs):
        model.train()
        train_loss = 0.0

        for x,y in train_loader:
            x, y = x.to(device), y.to(device)
            optimizer.zero_grad()
            outputs = model(x)
            loss = criterion(outputs, y)
            loss.backward()
```

```python
            optimizer.step()
            train_loss += loss.item()

        train_loss /= len(train_loader)
        history["train_loss"].append(train_loss)

        # Evaluación
        model.eval()
        test_loss = 0.0
        correct, total = 0, 0

        with torch.no_grad():
            for x,y in test_loader:
                x, y = x.to(device), y.to(device)
                outputs = model(x)
                test_loss += criterion(outputs, y).item()
                _, pred = outputs.max(1)
                total += y.size(0)
                correct += pred.eq(y).sum().item()

        test_loss /= len(test_loader)
        accuracy = 100 * correct / total

        history["test_loss"].append(test_loss)
        history["accuracy"].append(accuracy)

        print(f"Epoch {epoch+1} | Train Loss {train_loss:.4f} | "
              f"Test Loss {test_loss:.4f} | Acc {accuracy:.2f}%")

        # Early stopping
        if test_loss < best_loss - min_delta:
            best_loss = test_loss
            epochs_no_improve = 0
        else:
            epochs_no_improve += 1

        if epochs_no_improve >= patience:
            print("✔ Early stopping activado")
            break

    # Gráficos
    plt.figure(figsize=(10,4))
    plt.subplot(1,2,1)
    plt.plot(history["train_loss"], label="Train")
    plt.plot(history["test_loss"], label="Test")
    plt.title("Loss")
    plt.legend()

    plt.subplot(1,2,2)
    plt.plot(history["accuracy"], label="Accuracy")
    plt.title("Accuracy (%)")
    plt.legend()
    plt.show()

    return history
```

In [9]:
```python
def train_autoencoder(model, train_loader, test_loader, criterion, optimizer, device, num_epochs=10):
    train_losses = []
    test_losses = []
    model.to(device)
```

```
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        for x, _ in train_loader:
            x = x.to(device)
            x_flat = x.view(x.size(0), -1)     #  aplanar
            optimizer.zero_grad()
            outputs = model(x_flat)
            loss = criterion(outputs, x_flat)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

        train_losses.append(running_loss / len(train_loader))

        # evaluación
        model.eval()
        test_loss = 0.0
        with torch.no_grad():
            for x, _ in test_loader:
                x = x.to(device)
                x_flat = x.view(x.size(0), -1)
                outputs = model(x_flat)
                loss = criterion(outputs, x_flat)
                test_loss += loss.item()
        test_losses.append(test_loss / len(test_loader))

        print(f"Epoch [{epoch+1}/{num_epochs}] - Train Loss: {train_losses[-1]:.4f}, Test Loss: {test_losses[-1]:.4f}")

    return train_losses, test_losses
```

La primer estructura que quiero hacer es el autoencoder. Vo a elegir una arquitectura simétrica con una reducción del 80% de tamaño de dimensión de entrada, de acuerdo al principio de Pareto, que dice que el 80% de las cosas son causadas por el 20%. Esto lo extrapolo a energía y varianza y listo.

Entramos con $28 * 28$, luego a $28 * 28 * 0.5$y luego a $0.2 * 28 * 28$. De ahí sube de vuelta

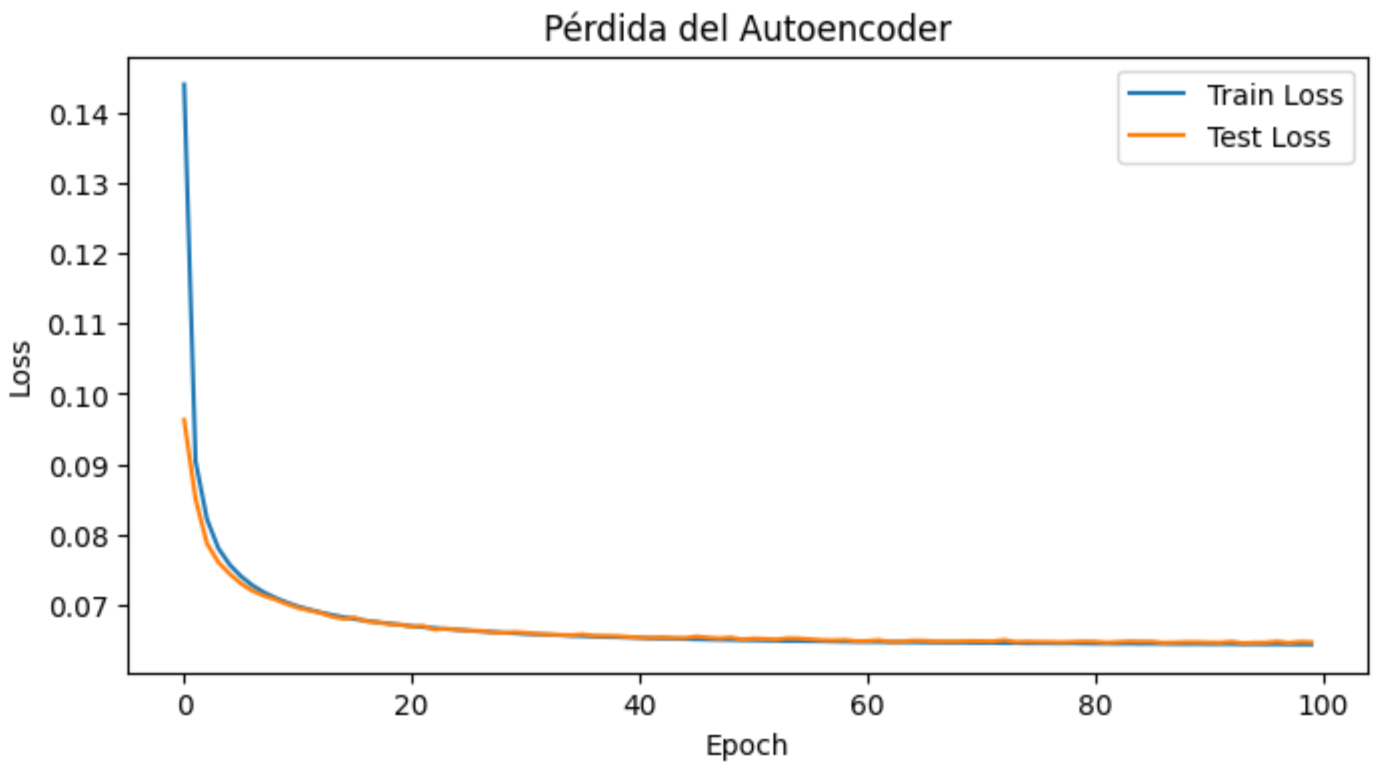
In [10]:
```
class Autoencoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(28*28, 392)
        self.fc2 = nn.Linear(392, 156)
        self.fc3 = nn.Linear(156, 392)
        self.fc4 = nn.Linear(392, 28*28)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = torch.sigmoid(self.fc4(x))
        return x

model_cnn = Autoencoder().to(device)
```

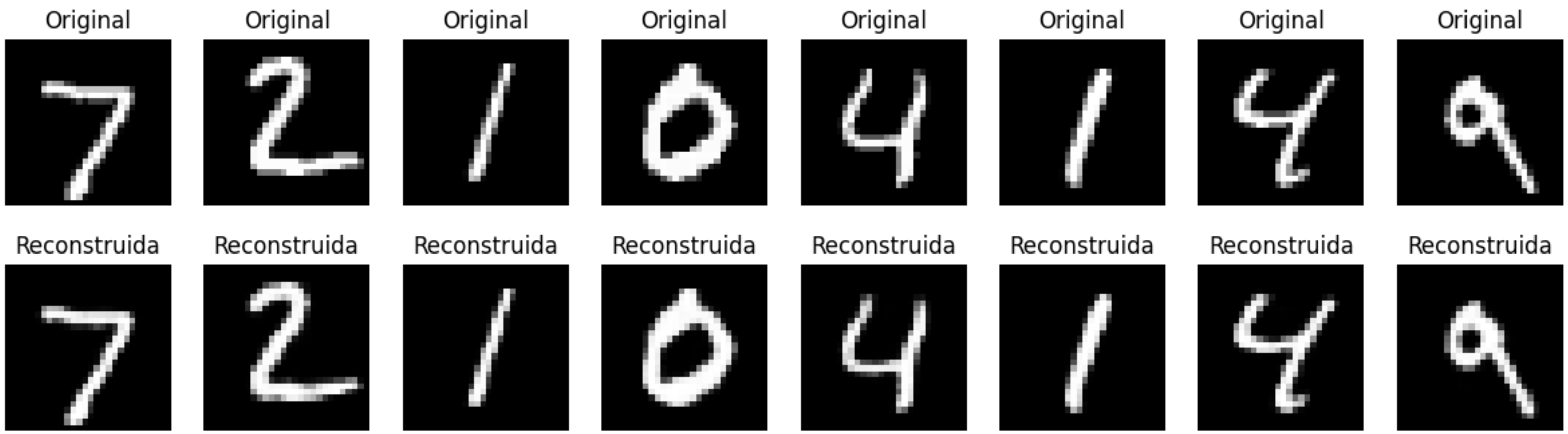
In [11]:
```
# -------- Entrenamiento --------
criterion = nn.BCELoss()
optimizer = optim.Adam(model_cnn.parameters(), lr=1e-3)
```

```
train_losses, test_losses = train_autoencoder(
    model_cnn, train_loader, test_loader,
    criterion, optimizer, device,
    num_epochs=100
)


# -------- Gráficos --------
plt.figure(figsize=(8,4))
plt.plot(train_losses, label='Train Loss')
plt.plot(test_losses, label='Test Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Pérdida del Autoencoder')
plt.show()


# -------- Visualizar reconstrucciones --------
model_cnn.eval()
with torch.no_grad():
    for x, _ in test_loader:
        x = x.to(device)
        x_flat = x.view(x.size(0), -1)
        outputs = model_cnn(x_flat)
        break  # solo un batch

# convertir a imágenes
x = x.cpu().view(-1, 1, 28, 28)
outputs = outputs.cpu().view(-1, 1, 28, 28)

# mostrar algunas imágenes
n = 8
plt.figure(figsize=(15,4))
for i in range(n):
    # original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x[i][0], cmap="gray")
    plt.title("Original")
    plt.axis("off")

    # reconstruida
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(outputs[i][0], cmap="gray")
    plt.title("Reconstruida")
    plt.axis("off")

plt.show()
```

```
Epoch [1/100] - Train Loss: 0.1440, Test Loss: 0.0963
Epoch [2/100] - Train Loss: 0.0904, Test Loss: 0.0851
Epoch [3/100] - Train Loss: 0.0822, Test Loss: 0.0788
Epoch [4/100] - Train Loss: 0.0781, Test Loss: 0.0761
Epoch [5/100] - Train Loss: 0.0757, Test Loss: 0.0744
Epoch [6/100] - Train Loss: 0.0741, Test Loss: 0.0730
Epoch [7/100] - Train Loss: 0.0728, Test Loss: 0.0720
Epoch [8/100] - Train Loss: 0.0718, Test Loss: 0.0713
Epoch [9/100] - Train Loss: 0.0710, Test Loss: 0.0708
Epoch [10/100] - Train Loss: 0.0704, Test Loss: 0.0701
Epoch [11/100] - Train Loss: 0.0698, Test Loss: 0.0696
Epoch [12/100] - Train Loss: 0.0694, Test Loss: 0.0692
Epoch [13/100] - Train Loss: 0.0689, Test Loss: 0.0689
Epoch [14/100] - Train Loss: 0.0686, Test Loss: 0.0683
Epoch [15/100] - Train Loss: 0.0683, Test Loss: 0.0680
Epoch [16/100] - Train Loss: 0.0680, Test Loss: 0.0682
Epoch [17/100] - Train Loss: 0.0678, Test Loss: 0.0677
Epoch [18/100] - Train Loss: 0.0675, Test Loss: 0.0675
Epoch [19/100] - Train Loss: 0.0674, Test Loss: 0.0673
Epoch [20/100] - Train Loss: 0.0672, Test Loss: 0.0672
Epoch [21/100] - Train Loss: 0.0670, Test Loss: 0.0669
Epoch [22/100] - Train Loss: 0.0669, Test Loss: 0.0670
Epoch [23/100] - Train Loss: 0.0667, Test Loss: 0.0665
Epoch [24/100] - Train Loss: 0.0666, Test Loss: 0.0667
Epoch [25/100] - Train Loss: 0.0665, Test Loss: 0.0665
Epoch [26/100] - Train Loss: 0.0664, Test Loss: 0.0663
Epoch [27/100] - Train Loss: 0.0663, Test Loss: 0.0663
Epoch [28/100] - Train Loss: 0.0662, Test Loss: 0.0661
Epoch [29/100] - Train Loss: 0.0661, Test Loss: 0.0660
Epoch [30/100] - Train Loss: 0.0660, Test Loss: 0.0662
Epoch [31/100] - Train Loss: 0.0659, Test Loss: 0.0661
Epoch [32/100] - Train Loss: 0.0658, Test Loss: 0.0659
Epoch [33/100] - Train Loss: 0.0658, Test Loss: 0.0658
Epoch [34/100] - Train Loss: 0.0657, Test Loss: 0.0658
Epoch [35/100] - Train Loss: 0.0656, Test Loss: 0.0657
Epoch [36/100] - Train Loss: 0.0656, Test Loss: 0.0658
Epoch [37/100] - Train Loss: 0.0655, Test Loss: 0.0656
Epoch [38/100] - Train Loss: 0.0655, Test Loss: 0.0656
Epoch [39/100] - Train Loss: 0.0654, Test Loss: 0.0656
Epoch [40/100] - Train Loss: 0.0654, Test Loss: 0.0655
Epoch [41/100] - Train Loss: 0.0653, Test Loss: 0.0654
Epoch [42/100] - Train Loss: 0.0653, Test Loss: 0.0653
Epoch [43/100] - Train Loss: 0.0652, Test Loss: 0.0654
Epoch [44/100] - Train Loss: 0.0652, Test Loss: 0.0653
Epoch [45/100] - Train Loss: 0.0652, Test Loss: 0.0653
Epoch [46/100] - Train Loss: 0.0651, Test Loss: 0.0655
Epoch [47/100] - Train Loss: 0.0651, Test Loss: 0.0654
Epoch [48/100] - Train Loss: 0.0651, Test Loss: 0.0653
Epoch [49/100] - Train Loss: 0.0651, Test Loss: 0.0654
Epoch [50/100] - Train Loss: 0.0650, Test Loss: 0.0651
Epoch [51/100] - Train Loss: 0.0650, Test Loss: 0.0652
Epoch [52/100] - Train Loss: 0.0650, Test Loss: 0.0652
Epoch [53/100] - Train Loss: 0.0649, Test Loss: 0.0651
Epoch [54/100] - Train Loss: 0.0649, Test Loss: 0.0653
Epoch [55/100] - Train Loss: 0.0649, Test Loss: 0.0652
Epoch [56/100] - Train Loss: 0.0649, Test Loss: 0.0651
Epoch [57/100] - Train Loss: 0.0649, Test Loss: 0.0650
Epoch [58/100] - Train Loss: 0.0648, Test Loss: 0.0650
Epoch [59/100] - Train Loss: 0.0648, Test Loss: 0.0650
Epoch [60/100] - Train Loss: 0.0648, Test Loss: 0.0649
```

```
Epoch [61/100] - Train Loss: 0.0648, Test Loss: 0.0649
Epoch [62/100] - Train Loss: 0.0648, Test Loss: 0.0650
Epoch [63/100] - Train Loss: 0.0647, Test Loss: 0.0648
Epoch [64/100] - Train Loss: 0.0647, Test Loss: 0.0648
Epoch [65/100] - Train Loss: 0.0647, Test Loss: 0.0650
Epoch [66/100] - Train Loss: 0.0647, Test Loss: 0.0649
Epoch [67/100] - Train Loss: 0.0647, Test Loss: 0.0648
Epoch [68/100] - Train Loss: 0.0647, Test Loss: 0.0648
Epoch [69/100] - Train Loss: 0.0647, Test Loss: 0.0648
Epoch [70/100] - Train Loss: 0.0646, Test Loss: 0.0649
Epoch [71/100] - Train Loss: 0.0646, Test Loss: 0.0649
Epoch [72/100] - Train Loss: 0.0646, Test Loss: 0.0648
Epoch [73/100] - Train Loss: 0.0646, Test Loss: 0.0650
Epoch [74/100] - Train Loss: 0.0646, Test Loss: 0.0647
Epoch [75/100] - Train Loss: 0.0646, Test Loss: 0.0648
Epoch [76/100] - Train Loss: 0.0646, Test Loss: 0.0648
Epoch [77/100] - Train Loss: 0.0646, Test Loss: 0.0648
Epoch [78/100] - Train Loss: 0.0646, Test Loss: 0.0647
Epoch [79/100] - Train Loss: 0.0646, Test Loss: 0.0648
Epoch [80/100] - Train Loss: 0.0645, Test Loss: 0.0648
Epoch [81/100] - Train Loss: 0.0645, Test Loss: 0.0648
Epoch [82/100] - Train Loss: 0.0645, Test Loss: 0.0647
Epoch [83/100] - Train Loss: 0.0645, Test Loss: 0.0647
Epoch [84/100] - Train Loss: 0.0645, Test Loss: 0.0648
Epoch [85/100] - Train Loss: 0.0645, Test Loss: 0.0648
Epoch [86/100] - Train Loss: 0.0645, Test Loss: 0.0648
Epoch [87/100] - Train Loss: 0.0645, Test Loss: 0.0646
Epoch [88/100] - Train Loss: 0.0645, Test Loss: 0.0647
Epoch [89/100] - Train Loss: 0.0645, Test Loss: 0.0647
Epoch [90/100] - Train Loss: 0.0645, Test Loss: 0.0647
Epoch [91/100] - Train Loss: 0.0645, Test Loss: 0.0647
Epoch [92/100] - Train Loss: 0.0645, Test Loss: 0.0647
Epoch [93/100] - Train Loss: 0.0645, Test Loss: 0.0648
Epoch [94/100] - Train Loss: 0.0644, Test Loss: 0.0646
Epoch [95/100] - Train Loss: 0.0644, Test Loss: 0.0646
Epoch [96/100] - Train Loss: 0.0644, Test Loss: 0.0647
Epoch [97/100] - Train Loss: 0.0644, Test Loss: 0.0648
Epoch [98/100] - Train Loss: 0.0644, Test Loss: 0.0646
Epoch [99/100] - Train Loss: 0.0644, Test Loss: 0.0648
Epoch [100/100] - Train Loss: 0.0644, Test Loss: 0.0647
```





1570 parámetros no congelados, la capa de 156 a 10 + bias

In [ ]:
```
class ClassifierFromEncoder(nn.Module):
    def __init__(self, autoencoder, num_classes=10):
        super().__init__()
        # acá tomamos las capas del otro modelo y las congelamos
        self.encoder_fc1 = autoencoder.fc1
        self.encoder_fc2 = autoencoder.fc2
        for param in self.encoder_fc1.parameters():
            param.requires_grad = False
        for param in self.encoder_fc2.parameters():
            param.requires_grad = False
        self.classifier = nn.Linear(156, num_classes) # lo más simple posible

    def forward(self, x):
        x = x.view(x.size(0), -1)
```

```
        with torch.no_grad():
            x = F.relu(self.encoder_fc1(x))
            encoded = F.relu(self.encoder_fc2(x))
        logits = self.classifier(encoded)
        return logits

classifier_model = ClassifierFromEncoder(model_cnn).to(device)
criterion_cls = nn.CrossEntropyLoss()
optimizer_cls = optim.Adam(classifier_model.classifier.parameters(), lr=1e-3)

# -----------------------
# Entrenamiento Clasificador
# -----------------------
num_epochs_cls = 100
for epoch in range(num_epochs_cls):
    classifier_model.train()
    running_loss = 0
    correct, total = 0, 0
    for x, y in train_loader:
        x, y = x.to(device), y.to(device)
        optimizer_cls.zero_grad()
        outputs = classifier_model(x)
        loss = criterion_cls(outputs, y)
        loss.backward()
        optimizer_cls.step()
        running_loss += loss.item()
        _, pred = outputs.max(1)
        correct += pred.eq(y).sum().item()
        total += y.size(0)
    train_acc = 100 * correct / total
    print(f"Epoch {epoch+1}/{num_epochs_cls} | Loss={running_loss/len(train_loader):.4f} | Acc={train_acc:.2f}%")

# -----------------------
# Evaluación en Test
# -----------------------
classifier_model.eval()
correct, total = 0, 0
with torch.no_grad():
    for x, y in test_loader:
        x, y = x.to(device), y.to(device)
        outputs = classifier_model(x)
        _, pred = outputs.max(1)
        correct += pred.eq(y).sum().item()
        total += y.size(0)
print(f"Test Accuracy: {100 * correct/total:.2f}%")



# Evaluación por dígito
classifier_model.eval()
confusion_matrix = torch.zeros(10, 10)
digit_totals = torch.zeros(10)
digit_correct = torch.zeros(10)
total_correct = 0
total_samples = 0

with torch.no_grad():
    for x, y in test_loader:
        x, y = x.to(device), y.to(device)
        outputs = classifier_model(x)
```

```
        _, pred = outputs.max(1)

        # Contadores globales
        total_correct += (pred == y).sum().item()
        total_samples += y.size(0)

        # Contadores por dígito
        for t, p in zip(y.view(-1), pred.view(-1)):
            confusion_matrix[t.long(), p.long()] += 1
            digit_totals[t.long()] += 1
            if t == p:
                digit_correct[t.long()] += 1

# Calcular ambas métricas
digit_accuracy = (digit_correct / digit_totals) * 100
average_accuracy = digit_accuracy.mean()
global_accuracy = (total_correct / total_samples) * 100

print("Accuracy por dígito:")
print("-" * 30)
for digit in range(10):
    print(f"Dígito {digit}: {digit_accuracy[digit]:.2f}% ({digit_totals[digit]:.0f} muestras)")
print("-" * 30)
print(f"Accuracy promedio (media de accuracies): {average_accuracy:.2f}%")
print(f"Accuracy global (total correctos/total muestras): {global_accuracy:.2f}%")
```

```
Epoch 1/100 | Loss=0.7307 | Acc=80.71%
Epoch 2/100 | Loss=0.3719 | Acc=89.63%
Epoch 3/100 | Loss=0.3321 | Acc=90.47%
Epoch 4/100 | Loss=0.3143 | Acc=90.88%
Epoch 5/100 | Loss=0.3054 | Acc=91.16%
Epoch 6/100 | Loss=0.2997 | Acc=91.23%
Epoch 7/100 | Loss=0.2950 | Acc=91.32%
Epoch 8/100 | Loss=0.2923 | Acc=91.53%
Epoch 9/100 | Loss=0.2911 | Acc=91.49%
Epoch 10/100 | Loss=0.2894 | Acc=91.58%
Epoch 11/100 | Loss=0.2887 | Acc=91.55%
Epoch 12/100 | Loss=0.2874 | Acc=91.56%
Epoch 13/100 | Loss=0.2870 | Acc=91.64%
Epoch 14/100 | Loss=0.2863 | Acc=91.69%
Epoch 15/100 | Loss=0.2859 | Acc=91.63%
Epoch 16/100 | Loss=0.2845 | Acc=91.66%
Epoch 17/100 | Loss=0.2842 | Acc=91.75%
Epoch 18/100 | Loss=0.2851 | Acc=91.71%
Epoch 19/100 | Loss=0.2838 | Acc=91.77%
Epoch 20/100 | Loss=0.2850 | Acc=91.65%
Epoch 21/100 | Loss=0.2838 | Acc=91.67%
Epoch 22/100 | Loss=0.2836 | Acc=91.68%
Epoch 23/100 | Loss=0.2834 | Acc=91.72%
Epoch 24/100 | Loss=0.2833 | Acc=91.73%
Epoch 25/100 | Loss=0.2832 | Acc=91.73%
Epoch 26/100 | Loss=0.2838 | Acc=91.70%
Epoch 27/100 | Loss=0.2829 | Acc=91.72%
Epoch 28/100 | Loss=0.2833 | Acc=91.70%
Epoch 29/100 | Loss=0.2836 | Acc=91.74%
Epoch 30/100 | Loss=0.2826 | Acc=91.77%
Epoch 31/100 | Loss=0.2827 | Acc=91.77%
Epoch 32/100 | Loss=0.2834 | Acc=91.71%
Epoch 33/100 | Loss=0.2830 | Acc=91.81%
Epoch 34/100 | Loss=0.2824 | Acc=91.83%
Epoch 35/100 | Loss=0.2829 | Acc=91.84%
Epoch 36/100 | Loss=0.2832 | Acc=91.73%
Epoch 37/100 | Loss=0.2830 | Acc=91.82%
Epoch 38/100 | Loss=0.2830 | Acc=91.86%
Epoch 39/100 | Loss=0.2823 | Acc=91.82%
Epoch 40/100 | Loss=0.2821 | Acc=91.83%
Epoch 41/100 | Loss=0.2832 | Acc=91.77%
Epoch 42/100 | Loss=0.2827 | Acc=91.77%
Epoch 43/100 | Loss=0.2826 | Acc=91.80%
Epoch 44/100 | Loss=0.2830 | Acc=91.83%
Epoch 45/100 | Loss=0.2825 | Acc=91.78%
Epoch 46/100 | Loss=0.2826 | Acc=91.78%
Epoch 47/100 | Loss=0.2832 | Acc=91.78%
Epoch 48/100 | Loss=0.2813 | Acc=91.83%
Epoch 49/100 | Loss=0.2827 | Acc=91.81%
Epoch 50/100 | Loss=0.2826 | Acc=91.77%
Epoch 51/100 | Loss=0.2821 | Acc=91.81%
Epoch 52/100 | Loss=0.2823 | Acc=91.74%
Epoch 53/100 | Loss=0.2823 | Acc=91.83%
Epoch 54/100 | Loss=0.2825 | Acc=91.84%
Epoch 55/100 | Loss=0.2827 | Acc=91.80%
Epoch 56/100 | Loss=0.2821 | Acc=91.84%
Epoch 57/100 | Loss=0.2835 | Acc=91.79%
Epoch 58/100 | Loss=0.2824 | Acc=91.81%
Epoch 59/100 | Loss=0.2829 | Acc=91.86%
Epoch 60/100 | Loss=0.2821 | Acc=91.83%
```

```
Epoch 61/100 | Loss=0.2822 | Acc=91.77%
Epoch 62/100 | Loss=0.2824 | Acc=91.85%
Epoch 63/100 | Loss=0.2825 | Acc=91.84%
Epoch 64/100 | Loss=0.2834 | Acc=91.77%
Epoch 65/100 | Loss=0.2822 | Acc=91.77%
Epoch 66/100 | Loss=0.2829 | Acc=91.71%
Epoch 67/100 | Loss=0.2834 | Acc=91.82%
Epoch 68/100 | Loss=0.2821 | Acc=91.72%
Epoch 69/100 | Loss=0.2823 | Acc=91.85%
Epoch 70/100 | Loss=0.2821 | Acc=91.78%
Epoch 71/100 | Loss=0.2820 | Acc=91.85%
Epoch 72/100 | Loss=0.2823 | Acc=91.81%
Epoch 73/100 | Loss=0.2820 | Acc=91.79%
Epoch 74/100 | Loss=0.2825 | Acc=91.75%
Epoch 75/100 | Loss=0.2826 | Acc=91.83%
Epoch 76/100 | Loss=0.2825 | Acc=91.81%
Epoch 77/100 | Loss=0.2816 | Acc=91.83%
Epoch 78/100 | Loss=0.2822 | Acc=91.79%
Epoch 79/100 | Loss=0.2830 | Acc=91.85%
Epoch 80/100 | Loss=0.2824 | Acc=91.78%
Epoch 81/100 | Loss=0.2826 | Acc=91.75%
Epoch 82/100 | Loss=0.2831 | Acc=91.76%
Epoch 83/100 | Loss=0.2821 | Acc=91.86%
Epoch 84/100 | Loss=0.2829 | Acc=91.78%
Epoch 85/100 | Loss=0.2819 | Acc=91.76%
Epoch 86/100 | Loss=0.2830 | Acc=91.84%
Epoch 87/100 | Loss=0.2812 | Acc=91.88%
Epoch 88/100 | Loss=0.2836 | Acc=91.81%
Epoch 89/100 | Loss=0.2828 | Acc=91.80%
Epoch 90/100 | Loss=0.2833 | Acc=91.80%
Epoch 91/100 | Loss=0.2825 | Acc=91.77%
Epoch 92/100 | Loss=0.2820 | Acc=91.82%
Epoch 93/100 | Loss=0.2833 | Acc=91.84%
Epoch 94/100 | Loss=0.2827 | Acc=91.78%
Epoch 95/100 | Loss=0.2823 | Acc=91.88%
Epoch 96/100 | Loss=0.2829 | Acc=91.84%
Epoch 97/100 | Loss=0.2825 | Acc=91.76%
Epoch 98/100 | Loss=0.2819 | Acc=91.88%
Epoch 99/100 | Loss=0.2826 | Acc=91.79%
Epoch 100/100 | Loss=0.2829 | Acc=91.80%
Test Accuracy: 92.09%
```

In [15]:

```python
class ClassifierSinEncoder(nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        self.classifier = nn.Linear(28*28, num_classes) # conexión directa de imagen a clases

    def forward(self, x):
        x = x.view(x.size(0), -1)  # aplanar la imagen
        logits = self.classifier(x)
        return logits

# Crear modelo y moverlo al dispositivo
classifier_model = ClassifierSinEncoder().to(device)
criterion_cls = nn.CrossEntropyLoss()
optimizer_cls = optim.Adam(classifier_model.parameters(), lr=1e-3)

# ------------------------
# Entrenamiento Clasificador
# ------------------------
```

```python
num_epochs_cls = 100
for epoch in range(num_epochs_cls):
    classifier_model.train()
    running_loss = 0
    correct, total = 0, 0
    for x, y in train_loader:
        x, y = x.to(device), y.to(device)
        optimizer_cls.zero_grad()
        outputs = classifier_model(x)
        loss = criterion_cls(outputs, y)
        loss.backward()
        optimizer_cls.step()
        running_loss += loss.item()
        _, pred = outputs.max(1)
        correct += pred.eq(y).sum().item()
        total += y.size(0)
    train_acc = 100 * correct / total
    print(f"Epoch {epoch+1}/{num_epochs_cls} | Loss={running_loss/len(train_loader):.4f} | Acc={train_acc:.2f}%")


# -----------------------
# Evaluación en Test
# -----------------------
classifier_model.eval()
correct, total = 0, 0
with torch.no_grad():
    for x, y in test_loader:
        x, y = x.to(device), y.to(device)
        outputs = classifier_model(x)
        _, pred = outputs.max(1)
        correct += pred.eq(y).sum().item()
        total += y.size(0)
print(f"Test Accuracy: {100 * correct/total:.2f}%")



# Evaluación por dígito
classifier_model.eval()
confusion_matrix = torch.zeros(10, 10)
digit_totals = torch.zeros(10)
digit_correct = torch.zeros(10)
total_correct = 0
total_samples = 0

with torch.no_grad():
    for x, y in test_loader:
        x, y = x.to(device), y.to(device)
        outputs = classifier_model(x)
        _, pred = outputs.max(1)

        # Contadores globales
        total_correct += (pred == y).sum().item()
        total_samples += y.size(0)

        # Contadores por dígito
        for t, p in zip(y.view(-1), pred.view(-1)):
            confusion_matrix[t.long(), p.long()] += 1
            digit_totals[t.long()] += 1
            if t == p:
                digit_correct[t.long()] += 1
```

```python
# Calcular ambas métricas
digit_accuracy = (digit_correct / digit_totals) * 100
average_accuracy = digit_accuracy.mean()
global_accuracy = (total_correct / total_samples) * 100

print("Accuracy por dígito:")
print("-" * 30)
for digit in range(10):
    print(f"Dígito {digit}: {digit_accuracy[digit]:.2f}% ({digit_totals[digit]:.0f} muestras)")
print("-" * 30)
print(f"Accuracy promedio (media de accuracies): {average_accuracy:.2f}%")
print(f"Accuracy global (total correctos/total muestras): {global_accuracy:.2f}%")
```

```
Epoch 1/100 | Loss=0.5398 | Acc=87.04%
Epoch 2/100 | Loss=0.3229 | Acc=91.07%
Epoch 3/100 | Loss=0.2953 | Acc=91.79%
Epoch 4/100 | Loss=0.2819 | Acc=92.12%
Epoch 5/100 | Loss=0.2736 | Acc=92.32%
Epoch 6/100 | Loss=0.2678 | Acc=92.57%
Epoch 7/100 | Loss=0.2634 | Acc=92.61%
Epoch 8/100 | Loss=0.2599 | Acc=92.80%
Epoch 9/100 | Loss=0.2570 | Acc=92.92%
Epoch 10/100 | Loss=0.2550 | Acc=92.92%
Epoch 11/100 | Loss=0.2523 | Acc=93.01%
Epoch 12/100 | Loss=0.2510 | Acc=92.99%
Epoch 13/100 | Loss=0.2492 | Acc=93.08%
Epoch 14/100 | Loss=0.2481 | Acc=93.16%
Epoch 15/100 | Loss=0.2464 | Acc=93.19%
Epoch 16/100 | Loss=0.2456 | Acc=93.18%
Epoch 17/100 | Loss=0.2445 | Acc=93.28%
Epoch 18/100 | Loss=0.2437 | Acc=93.29%
Epoch 19/100 | Loss=0.2422 | Acc=93.31%
Epoch 20/100 | Loss=0.2418 | Acc=93.33%
Epoch 21/100 | Loss=0.2409 | Acc=93.40%
Epoch 22/100 | Loss=0.2403 | Acc=93.37%
Epoch 23/100 | Loss=0.2396 | Acc=93.42%
Epoch 24/100 | Loss=0.2387 | Acc=93.45%
Epoch 25/100 | Loss=0.2382 | Acc=93.45%
Epoch 26/100 | Loss=0.2378 | Acc=93.45%
Epoch 27/100 | Loss=0.2375 | Acc=93.45%
Epoch 28/100 | Loss=0.2365 | Acc=93.52%
Epoch 29/100 | Loss=0.2364 | Acc=93.52%
Epoch 30/100 | Loss=0.2357 | Acc=93.48%
Epoch 31/100 | Loss=0.2353 | Acc=93.51%
Epoch 32/100 | Loss=0.2347 | Acc=93.57%
Epoch 33/100 | Loss=0.2342 | Acc=93.55%
Epoch 34/100 | Loss=0.2337 | Acc=93.55%
Epoch 35/100 | Loss=0.2336 | Acc=93.56%
Epoch 36/100 | Loss=0.2329 | Acc=93.58%
Epoch 37/100 | Loss=0.2326 | Acc=93.53%
Epoch 38/100 | Loss=0.2327 | Acc=93.63%
Epoch 39/100 | Loss=0.2318 | Acc=93.58%
Epoch 40/100 | Loss=0.2317 | Acc=93.58%
Epoch 41/100 | Loss=0.2314 | Acc=93.63%
Epoch 42/100 | Loss=0.2315 | Acc=93.64%
Epoch 43/100 | Loss=0.2311 | Acc=93.59%
Epoch 44/100 | Loss=0.2303 | Acc=93.69%
Epoch 45/100 | Loss=0.2302 | Acc=93.63%
Epoch 46/100 | Loss=0.2299 | Acc=93.67%
Epoch 47/100 | Loss=0.2296 | Acc=93.67%
Epoch 48/100 | Loss=0.2297 | Acc=93.65%
Epoch 49/100 | Loss=0.2288 | Acc=93.66%
Epoch 50/100 | Loss=0.2292 | Acc=93.69%
Epoch 51/100 | Loss=0.2287 | Acc=93.67%
Epoch 52/100 | Loss=0.2287 | Acc=93.69%
Epoch 53/100 | Loss=0.2284 | Acc=93.71%
Epoch 54/100 | Loss=0.2281 | Acc=93.70%
Epoch 55/100 | Loss=0.2278 | Acc=93.68%
Epoch 56/100 | Loss=0.2277 | Acc=93.70%
Epoch 57/100 | Loss=0.2275 | Acc=93.67%
Epoch 58/100 | Loss=0.2273 | Acc=93.66%
Epoch 59/100 | Loss=0.2273 | Acc=93.76%
Epoch 60/100 | Loss=0.2267 | Acc=93.69%
```

```
Epoch 61/100 | Loss=0.2268 | Acc=93.75%
Epoch 62/100 | Loss=0.2262 | Acc=93.75%
Epoch 63/100 | Loss=0.2267 | Acc=93.74%
Epoch 64/100 | Loss=0.2263 | Acc=93.77%
Epoch 65/100 | Loss=0.2261 | Acc=93.73%
Epoch 66/100 | Loss=0.2257 | Acc=93.74%
Epoch 67/100 | Loss=0.2257 | Acc=93.70%
Epoch 68/100 | Loss=0.2258 | Acc=93.69%
Epoch 69/100 | Loss=0.2252 | Acc=93.79%
Epoch 70/100 | Loss=0.2251 | Acc=93.73%
Epoch 71/100 | Loss=0.2247 | Acc=93.76%
Epoch 72/100 | Loss=0.2247 | Acc=93.81%
Epoch 73/100 | Loss=0.2252 | Acc=93.80%
Epoch 74/100 | Loss=0.2245 | Acc=93.78%
Epoch 75/100 | Loss=0.2246 | Acc=93.81%
Epoch 76/100 | Loss=0.2240 | Acc=93.85%
Epoch 77/100 | Loss=0.2243 | Acc=93.77%
Epoch 78/100 | Loss=0.2238 | Acc=93.80%
Epoch 79/100 | Loss=0.2242 | Acc=93.77%
Epoch 80/100 | Loss=0.2235 | Acc=93.80%
Epoch 81/100 | Loss=0.2235 | Acc=93.83%
Epoch 82/100 | Loss=0.2235 | Acc=93.81%
Epoch 83/100 | Loss=0.2238 | Acc=93.90%
Epoch 84/100 | Loss=0.2237 | Acc=93.86%
Epoch 85/100 | Loss=0.2228 | Acc=93.87%
Epoch 86/100 | Loss=0.2229 | Acc=93.85%
Epoch 87/100 | Loss=0.2230 | Acc=93.81%
Epoch 88/100 | Loss=0.2228 | Acc=93.80%
Epoch 89/100 | Loss=0.2227 | Acc=93.84%
Epoch 90/100 | Loss=0.2225 | Acc=93.84%
Epoch 91/100 | Loss=0.2230 | Acc=93.83%
Epoch 92/100 | Loss=0.2226 | Acc=93.80%
Epoch 93/100 | Loss=0.2218 | Acc=93.83%
Epoch 94/100 | Loss=0.2223 | Acc=93.81%
Epoch 95/100 | Loss=0.2219 | Acc=93.92%
Epoch 96/100 | Loss=0.2219 | Acc=93.84%
Epoch 97/100 | Loss=0.2218 | Acc=93.85%
Epoch 98/100 | Loss=0.2223 | Acc=93.79%
Epoch 99/100 | Loss=0.2216 | Acc=93.86%
Epoch 100/100 | Loss=0.2213 | Acc=93.86%
Test Accuracy: 92.73%
Accuracy por dígito:
------------------------------
Dígito 0: 96.94% (980 muestras)
Dígito 1: 97.53% (1135 muestras)
Dígito 2: 89.73% (1032 muestras)
Dígito 3: 90.10% (1010 muestras)
Dígito 4: 92.77% (982 muestras)
Dígito 5: 87.44% (892 muestras)
Dígito 6: 95.93% (958 muestras)
Dígito 7: 92.41% (1028 muestras)
Dígito 8: 90.97% (974 muestras)
Dígito 9: 92.57% (1009 muestras)
------------------------------
Accuracy promedio (media de accuracies): 92.64%
Accuracy global (total correctos/total muestras): 92.73%
```

In [16]:

```
class ClassifierFromEncoder(nn.Module):
    def __init__(self, autoencoder, num_classes=10):
        super().__init__()
```

```
        # Capas del encoder congeladas
        self.encoder_fc1 = autoencoder.fc1
        self.encoder_fc2 = autoencoder.fc2
        for param in self.encoder_fc1.parameters():
            param.requires_grad = False
        for param in self.encoder_fc2.parameters():
            param.requires_grad = False

        # Capas del clasificador
        self.classifier1 = nn.Linear(156, 156)  # Nueva capa oculta
        self.classifier2 = nn.Linear(156, num_classes)  # Capa de salida

    def forward(self, x):
        x = x.view(x.size(0), -1)
        # Encoder congelado
        with torch.no_grad():
            x = F.relu(self.encoder_fc1(x))
            encoded = F.relu(self.encoder_fc2(x))
        # Clasificador entrenable
        x = F.relu(self.classifier1(encoded))
        logits = self.classifier2(x)
        return logits

classifier_model = ClassifierFromEncoder(model_cnn).to(device)
criterion_cls = nn.CrossEntropyLoss()
# Optimizador solo con parámetros de las capas classifier
optimizer_cls = optim.Adam([
    *classifier_model.classifier1.parameters(),
    *classifier_model.classifier2.parameters()
], lr=1e-3)
# -----------------------
# Entrenamiento Clasificador
# -----------------------
num_epochs_cls = 100
for epoch in range(num_epochs_cls):
    classifier_model.train()
    running_loss = 0
    correct, total = 0, 0
    for x, y in train_loader:
        x, y = x.to(device), y.to(device)
        optimizer_cls.zero_grad()
        outputs = classifier_model(x)
        loss = criterion_cls(outputs, y)
        loss.backward()
        optimizer_cls.step()
        running_loss += loss.item()
        _, pred = outputs.max(1)
        correct += pred.eq(y).sum().item()
        total += y.size(0)
    train_acc = 100 * correct / total
    print(f"Epoch {epoch+1}/{num_epochs_cls} | Loss={running_loss/len(train_loader):.4f} | Acc={train_acc:.2f}%")

# -----------------------
# Evaluación en Test
# -----------------------
classifier_model.eval()
correct, total = 0, 0
with torch.no_grad():
    for x, y in test_loader:
        x, y = x.to(device), y.to(device)
```

```
        outputs = classifier_model(x)
        _, pred = outputs.max(1)
        correct += pred.eq(y).sum().item()
        total += y.size(0)
print(f"Test Accuracy: {100 * correct/total:.2f}%")



# Evaluación por dígito
classifier_model.eval()
confusion_matrix = torch.zeros(10, 10)
digit_totals = torch.zeros(10)
digit_correct = torch.zeros(10)
total_correct = 0
total_samples = 0

with torch.no_grad():
    for x, y in test_loader:
        x, y = x.to(device), y.to(device)
        outputs = classifier_model(x)
        _, pred = outputs.max(1)

        # Contadores globales
        total_correct += (pred == y).sum().item()
        total_samples += y.size(0)

        # Contadores por dígito
        for t, p in zip(y.view(-1), pred.view(-1)):
            confusion_matrix[t.long(), p.long()] += 1
            digit_totals[t.long()] += 1
            if t == p:
                digit_correct[t.long()] += 1

# Calcular ambas métricas
digit_accuracy = (digit_correct / digit_totals) * 100
average_accuracy = digit_accuracy.mean()
global_accuracy = (total_correct / total_samples) * 100

print("Accuracy por dígito:")
print("-" * 30)
for digit in range(10):
    print(f"Dígito {digit}: {digit_accuracy[digit]:.2f}% ({digit_totals[digit]:.0f} muestras)")
print("-" * 30)
print(f"Accuracy promedio (media de accuracies): {average_accuracy:.2f}%")
print(f"Accuracy global (total correctos/total muestras): {global_accuracy:.2f}%")
```

```
Epoch 1/100 | Loss=0.4646 | Acc=86.90%
Epoch 2/100 | Loss=0.2832 | Acc=91.65%
Epoch 3/100 | Loss=0.2215 | Acc=93.48%
Epoch 4/100 | Loss=0.1733 | Acc=94.85%
Epoch 5/100 | Loss=0.1412 | Acc=95.86%
Epoch 6/100 | Loss=0.1200 | Acc=96.42%
Epoch 7/100 | Loss=0.1059 | Acc=96.73%
Epoch 8/100 | Loss=0.0955 | Acc=97.11%
Epoch 9/100 | Loss=0.0846 | Acc=97.47%
Epoch 10/100 | Loss=0.0781 | Acc=97.62%
Epoch 11/100 | Loss=0.0730 | Acc=97.72%
Epoch 12/100 | Loss=0.0660 | Acc=97.93%
Epoch 13/100 | Loss=0.0634 | Acc=98.02%
Epoch 14/100 | Loss=0.0583 | Acc=98.12%
Epoch 15/100 | Loss=0.0547 | Acc=98.26%
Epoch 16/100 | Loss=0.0519 | Acc=98.33%
Epoch 17/100 | Loss=0.0479 | Acc=98.50%
Epoch 18/100 | Loss=0.0467 | Acc=98.52%
Epoch 19/100 | Loss=0.0432 | Acc=98.60%
Epoch 20/100 | Loss=0.0412 | Acc=98.59%
Epoch 21/100 | Loss=0.0390 | Acc=98.71%
Epoch 22/100 | Loss=0.0366 | Acc=98.80%
Epoch 23/100 | Loss=0.0379 | Acc=98.74%
Epoch 24/100 | Loss=0.0334 | Acc=98.92%
Epoch 25/100 | Loss=0.0335 | Acc=98.88%
Epoch 26/100 | Loss=0.0305 | Acc=99.00%
Epoch 27/100 | Loss=0.0295 | Acc=99.01%
Epoch 28/100 | Loss=0.0271 | Acc=99.08%
Epoch 29/100 | Loss=0.0265 | Acc=99.11%
Epoch 30/100 | Loss=0.0248 | Acc=99.17%
Epoch 31/100 | Loss=0.0247 | Acc=99.19%
Epoch 32/100 | Loss=0.0235 | Acc=99.26%
Epoch 33/100 | Loss=0.0224 | Acc=99.24%
Epoch 34/100 | Loss=0.0231 | Acc=99.22%
Epoch 35/100 | Loss=0.0202 | Acc=99.30%
Epoch 36/100 | Loss=0.0202 | Acc=99.30%
Epoch 37/100 | Loss=0.0165 | Acc=99.45%
Epoch 38/100 | Loss=0.0174 | Acc=99.38%
Epoch 39/100 | Loss=0.0195 | Acc=99.32%
Epoch 40/100 | Loss=0.0160 | Acc=99.46%
Epoch 41/100 | Loss=0.0146 | Acc=99.51%
Epoch 42/100 | Loss=0.0171 | Acc=99.44%
Epoch 43/100 | Loss=0.0168 | Acc=99.42%
Epoch 44/100 | Loss=0.0139 | Acc=99.52%
Epoch 45/100 | Loss=0.0164 | Acc=99.41%
Epoch 46/100 | Loss=0.0150 | Acc=99.52%
Epoch 47/100 | Loss=0.0127 | Acc=99.61%
Epoch 48/100 | Loss=0.0143 | Acc=99.52%
Epoch 49/100 | Loss=0.0138 | Acc=99.55%
Epoch 50/100 | Loss=0.0115 | Acc=99.62%
Epoch 51/100 | Loss=0.0134 | Acc=99.52%
Epoch 52/100 | Loss=0.0107 | Acc=99.64%
Epoch 53/100 | Loss=0.0125 | Acc=99.56%
Epoch 54/100 | Loss=0.0117 | Acc=99.61%
Epoch 55/100 | Loss=0.0139 | Acc=99.51%
Epoch 56/100 | Loss=0.0075 | Acc=99.77%
Epoch 57/100 | Loss=0.0127 | Acc=99.56%
Epoch 58/100 | Loss=0.0115 | Acc=99.60%
Epoch 59/100 | Loss=0.0109 | Acc=99.63%
Epoch 60/100 | Loss=0.0091 | Acc=99.67%
```

```
Epoch 61/100 | Loss=0.0141 | Acc=99.48%
Epoch 62/100 | Loss=0.0079 | Acc=99.72%
Epoch 63/100 | Loss=0.0070 | Acc=99.75%
Epoch 64/100 | Loss=0.0134 | Acc=99.52%
Epoch 65/100 | Loss=0.0083 | Acc=99.71%
Epoch 66/100 | Loss=0.0090 | Acc=99.69%
Epoch 67/100 | Loss=0.0111 | Acc=99.63%
Epoch 68/100 | Loss=0.0107 | Acc=99.59%
Epoch 69/100 | Loss=0.0075 | Acc=99.74%
Epoch 70/100 | Loss=0.0105 | Acc=99.62%
Epoch 71/100 | Loss=0.0073 | Acc=99.74%
Epoch 72/100 | Loss=0.0124 | Acc=99.57%
Epoch 73/100 | Loss=0.0088 | Acc=99.68%
Epoch 74/100 | Loss=0.0068 | Acc=99.76%
Epoch 75/100 | Loss=0.0079 | Acc=99.70%
Epoch 76/100 | Loss=0.0091 | Acc=99.70%
Epoch 77/100 | Loss=0.0096 | Acc=99.67%
Epoch 78/100 | Loss=0.0058 | Acc=99.79%
Epoch 79/100 | Loss=0.0081 | Acc=99.72%
Epoch 80/100 | Loss=0.0089 | Acc=99.69%
Epoch 81/100 | Loss=0.0094 | Acc=99.67%
Epoch 82/100 | Loss=0.0079 | Acc=99.72%
Epoch 83/100 | Loss=0.0067 | Acc=99.76%
Epoch 84/100 | Loss=0.0089 | Acc=99.69%
Epoch 85/100 | Loss=0.0066 | Acc=99.76%
Epoch 86/100 | Loss=0.0073 | Acc=99.75%
Epoch 87/100 | Loss=0.0050 | Acc=99.84%
Epoch 88/100 | Loss=0.0091 | Acc=99.66%
Epoch 89/100 | Loss=0.0093 | Acc=99.67%
Epoch 90/100 | Loss=0.0032 | Acc=99.90%
Epoch 91/100 | Loss=0.0135 | Acc=99.57%
Epoch 92/100 | Loss=0.0066 | Acc=99.78%
Epoch 93/100 | Loss=0.0064 | Acc=99.78%
Epoch 94/100 | Loss=0.0070 | Acc=99.77%
Epoch 95/100 | Loss=0.0063 | Acc=99.77%
Epoch 96/100 | Loss=0.0084 | Acc=99.69%
Epoch 97/100 | Loss=0.0013 | Acc=99.97%
Epoch 98/100 | Loss=0.0130 | Acc=99.58%
Epoch 99/100 | Loss=0.0070 | Acc=99.77%
Epoch 100/100 | Loss=0.0067 | Acc=99.77%
Test Accuracy: 97.94%
Accuracy por dígito:
------------------------------
Dígito 0: 99.08% (980 muestras)
Dígito 1: 99.65% (1135 muestras)
Dígito 2: 97.87% (1032 muestras)
Dígito 3: 98.71% (1010 muestras)
Dígito 4: 98.27% (982 muestras)
Dígito 5: 96.19% (892 muestras)
Dígito 6: 98.23% (958 muestras)
Dígito 7: 97.28% (1028 muestras)
Dígito 8: 97.64% (974 muestras)
Dígito 9: 96.13% (1009 muestras)
------------------------------
Accuracy promedio (media de accuracies): 97.90%
Accuracy global (total correctos/total muestras): 97.94%
```

In [17]:
```
class ClassifierFromEncoder(nn.Module):
    def __init__(self, autoencoder, num_classes=10):
        super().__init__()
```

```
        # Solo copiamos las capas, sin congelarlas
        self.encoder_fc1 = autoencoder.fc1
        self.encoder_fc2 = autoencoder.fc2
        self.classifier = nn.Linear(156, num_classes)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        # Sin torch.no_grad()
        x = F.relu(self.encoder_fc1(x))
        encoded = F.relu(self.encoder_fc2(x))
        logits = self.classifier(encoded)
        return logits

classifier_model = ClassifierFromEncoder(model_cnn).to(device)
criterion_cls = nn.CrossEntropyLoss()
# Optimizador con todos los parámetros
optimizer_cls = optim.Adam(classifier_model.parameters(), lr=1e-3)


# -----------------------
# Entrenamiento Clasificador
# -----------------------
num_epochs_cls = 100
for epoch in range(num_epochs_cls):
    classifier_model.train()
    running_loss = 0
    correct, total = 0, 0
    for x, y in train_loader:
        x, y = x.to(device), y.to(device)
        optimizer_cls.zero_grad()
        outputs = classifier_model(x)
        loss = criterion_cls(outputs, y)
        loss.backward()
        optimizer_cls.step()
        running_loss += loss.item()
        _, pred = outputs.max(1)
        correct += pred.eq(y).sum().item()
        total += y.size(0)
    train_acc = 100 * correct / total
    print(f"Epoch {epoch+1}/{num_epochs_cls} | Loss={running_loss/len(train_loader):.4f} | Acc={train_acc:.2f}%")

# -----------------------
# Evaluación en Test
# -----------------------
classifier_model.eval()
correct, total = 0, 0
with torch.no_grad():
    for x, y in test_loader:
        x, y = x.to(device), y.to(device)
        outputs = classifier_model(x)
        _, pred = outputs.max(1)
        correct += pred.eq(y).sum().item()
        total += y.size(0)
print(f"Test Accuracy: {100 * correct/total:.2f}%")



# Evaluación por dígito
classifier_model.eval()
confusion_matrix = torch.zeros(10, 10)
```

```python
digit_totals = torch.zeros(10)
digit_correct = torch.zeros(10)
total_correct = 0
total_samples = 0

with torch.no_grad():
    for x, y in test_loader:
        x, y = x.to(device), y.to(device)
        outputs = classifier_model(x)
        _, pred = outputs.max(1)

        # Contadores globales
        total_correct += (pred == y).sum().item()
        total_samples += y.size(0)

        # Contadores por dígito
        for t, p in zip(y.view(-1), pred.view(-1)):
            confusion_matrix[t.long(), p.long()] += 1
            digit_totals[t.long()] += 1
            if t == p:
                digit_correct[t.long()] += 1

# Calcular ambas métricas
digit_accuracy = (digit_correct / digit_totals) * 100
average_accuracy = digit_accuracy.mean()
global_accuracy = (total_correct / total_samples) * 100

print("Accuracy por dígito:")
print("-" * 30)
for digit in range(10):
    print(f"Dígito {digit}: {digit_accuracy[digit]:.2f}% ({digit_totals[digit]:.0f} muestras)")
print("-" * 30)
print(f"Accuracy promedio (media de accuracies): {average_accuracy:.2f}%")
print(f"Accuracy global (total correctos/total muestras): {global_accuracy:.2f}%")
```

```
Epoch 1/100 | Loss=0.7107 | Acc=81.59%
Epoch 2/100 | Loss=0.3689 | Acc=89.73%
Epoch 3/100 | Loss=0.3295 | Acc=90.53%
Epoch 4/100 | Loss=0.3135 | Acc=90.94%
Epoch 5/100 | Loss=0.3042 | Acc=91.09%
Epoch 6/100 | Loss=0.2988 | Acc=91.26%
Epoch 7/100 | Loss=0.2950 | Acc=91.42%
Epoch 8/100 | Loss=0.2922 | Acc=91.55%
Epoch 9/100 | Loss=0.2899 | Acc=91.47%
Epoch 10/100 | Loss=0.2894 | Acc=91.53%
Epoch 11/100 | Loss=0.2879 | Acc=91.56%
Epoch 12/100 | Loss=0.2875 | Acc=91.63%
Epoch 13/100 | Loss=0.2855 | Acc=91.65%
Epoch 14/100 | Loss=0.2854 | Acc=91.61%
Epoch 15/100 | Loss=0.2857 | Acc=91.71%
Epoch 16/100 | Loss=0.2855 | Acc=91.64%
Epoch 17/100 | Loss=0.2849 | Acc=91.71%
Epoch 18/100 | Loss=0.2841 | Acc=91.69%
Epoch 19/100 | Loss=0.2833 | Acc=91.80%
Epoch 20/100 | Loss=0.2836 | Acc=91.77%
Epoch 21/100 | Loss=0.2840 | Acc=91.77%
Epoch 22/100 | Loss=0.2831 | Acc=91.78%
Epoch 23/100 | Loss=0.2838 | Acc=91.70%
Epoch 24/100 | Loss=0.2834 | Acc=91.75%
Epoch 25/100 | Loss=0.2836 | Acc=91.71%
Epoch 26/100 | Loss=0.2834 | Acc=91.70%
Epoch 27/100 | Loss=0.2831 | Acc=91.82%
Epoch 28/100 | Loss=0.2824 | Acc=91.81%
Epoch 29/100 | Loss=0.2828 | Acc=91.69%
Epoch 30/100 | Loss=0.2824 | Acc=91.76%
Epoch 31/100 | Loss=0.2828 | Acc=91.74%
Epoch 32/100 | Loss=0.2828 | Acc=91.74%
Epoch 33/100 | Loss=0.2840 | Acc=91.75%
Epoch 34/100 | Loss=0.2836 | Acc=91.81%
Epoch 35/100 | Loss=0.2824 | Acc=91.83%
Epoch 36/100 | Loss=0.2825 | Acc=91.80%
Epoch 37/100 | Loss=0.2828 | Acc=91.82%
Epoch 38/100 | Loss=0.2840 | Acc=91.78%
Epoch 39/100 | Loss=0.2819 | Acc=91.83%
Epoch 40/100 | Loss=0.2819 | Acc=91.80%
Epoch 41/100 | Loss=0.2832 | Acc=91.78%
Epoch 42/100 | Loss=0.2819 | Acc=91.84%
Epoch 43/100 | Loss=0.2826 | Acc=91.78%
Epoch 44/100 | Loss=0.2833 | Acc=91.74%
Epoch 45/100 | Loss=0.2826 | Acc=91.72%
Epoch 46/100 | Loss=0.2824 | Acc=91.88%
Epoch 47/100 | Loss=0.2816 | Acc=91.84%
Epoch 48/100 | Loss=0.2827 | Acc=91.83%
Epoch 49/100 | Loss=0.2831 | Acc=91.75%
Epoch 50/100 | Loss=0.2818 | Acc=91.83%
Epoch 51/100 | Loss=0.2824 | Acc=91.81%
Epoch 52/100 | Loss=0.2831 | Acc=91.75%
Epoch 53/100 | Loss=0.2826 | Acc=91.81%
Epoch 54/100 | Loss=0.2822 | Acc=91.73%
Epoch 55/100 | Loss=0.2825 | Acc=91.78%
Epoch 56/100 | Loss=0.2821 | Acc=91.79%
Epoch 57/100 | Loss=0.2835 | Acc=91.74%
Epoch 58/100 | Loss=0.2826 | Acc=91.79%
Epoch 59/100 | Loss=0.2826 | Acc=91.81%
Epoch 60/100 | Loss=0.2824 | Acc=91.69%
```

```
Epoch 61/100 | Loss=0.2827 | Acc=91.82%
Epoch 62/100 | Loss=0.2829 | Acc=91.80%
Epoch 63/100 | Loss=0.2829 | Acc=91.84%
Epoch 64/100 | Loss=0.2825 | Acc=91.80%
Epoch 65/100 | Loss=0.2824 | Acc=91.86%
Epoch 66/100 | Loss=0.2831 | Acc=91.70%
Epoch 67/100 | Loss=0.2825 | Acc=91.70%
Epoch 68/100 | Loss=0.2822 | Acc=91.85%
Epoch 69/100 | Loss=0.2824 | Acc=91.84%
Epoch 70/100 | Loss=0.2827 | Acc=91.87%
Epoch 71/100 | Loss=0.2826 | Acc=91.83%
Epoch 72/100 | Loss=0.2816 | Acc=91.89%
Epoch 73/100 | Loss=0.2822 | Acc=91.77%
Epoch 74/100 | Loss=0.2826 | Acc=91.81%
Epoch 75/100 | Loss=0.2833 | Acc=91.82%
Epoch 76/100 | Loss=0.2829 | Acc=91.78%
Epoch 77/100 | Loss=0.2817 | Acc=91.81%
Epoch 78/100 | Loss=0.2819 | Acc=91.85%
Epoch 79/100 | Loss=0.2825 | Acc=91.75%
Epoch 80/100 | Loss=0.2820 | Acc=91.83%
Epoch 81/100 | Loss=0.2817 | Acc=91.80%
Epoch 82/100 | Loss=0.2818 | Acc=91.81%
Epoch 83/100 | Loss=0.2826 | Acc=91.84%
Epoch 84/100 | Loss=0.2828 | Acc=91.78%
Epoch 85/100 | Loss=0.2820 | Acc=91.81%
Epoch 86/100 | Loss=0.2820 | Acc=91.81%
Epoch 87/100 | Loss=0.2827 | Acc=91.82%
Epoch 88/100 | Loss=0.2822 | Acc=91.80%
Epoch 89/100 | Loss=0.2819 | Acc=91.82%
Epoch 90/100 | Loss=0.2825 | Acc=91.81%
Epoch 91/100 | Loss=0.2831 | Acc=91.80%
Epoch 92/100 | Loss=0.2826 | Acc=91.82%
Epoch 93/100 | Loss=0.2831 | Acc=91.79%
Epoch 94/100 | Loss=0.2819 | Acc=91.79%
Epoch 95/100 | Loss=0.2826 | Acc=91.73%
Epoch 96/100 | Loss=0.2819 | Acc=91.84%
Epoch 97/100 | Loss=0.2820 | Acc=91.80%
Epoch 98/100 | Loss=0.2820 | Acc=91.82%
Epoch 99/100 | Loss=0.2830 | Acc=91.75%
Epoch 100/100 | Loss=0.2818 | Acc=91.80%
Test Accuracy: 92.35%
Accuracy por dígito:
------------------------------
Dígito 0: 97.65% (980 muestras)
Dígito 1: 97.97% (1135 muestras)
Dígito 2: 90.41% (1032 muestras)
Dígito 3: 90.89% (1010 muestras)
Dígito 4: 93.28% (982 muestras)
Dígito 5: 85.87% (892 muestras)
Dígito 6: 95.09% (958 muestras)
Dígito 7: 91.54% (1028 muestras)
Dígito 8: 89.12% (974 muestras)
Dígito 9: 90.49% (1009 muestras)
------------------------------
Accuracy promedio (media de accuracies): 92.23%
Accuracy global (total correctos/total muestras): 92.35%
```