Imports and Data Loading

In [60]:
```
# Load data from CSV files
import pandas as pd
data_df = pd.read_csv('train.csv')

# Split data into features (X) and labels (y)
X = data_df.drop('label', axis=1).values
y = data_df['label'].values

# Split data into 70% training and 30% testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_st

# Convert data to PyTorch tensors
import torch
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.long)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.long)
```

Define Custom Dataset and Transformations

In [61]:
```
# Define a custom dataset class
from torch.utils.data import DataLoader, Dataset

class CustomDataset(Dataset):
    def __init__(self, data, labels):
        self.data = data
        self.labels = labels

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        sample = {'image': self.data[idx], 'label': self.labels[idx]}
        return sample

# Create datasets and dataloaders
batch_size = 64
train_dataset = CustomDataset(X_train_tensor, y_train_tensor)
test_dataset = CustomDataset(X_test_tensor, y_test_tensor)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size)
```

Define Neural Structure

In [78]:
```
# Define a simple neural network architecture
import torch.nn as nn

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 10)
```

```
    def forward(self, x):
        x = x.view(-1, 784)
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x

# Create an instance of the model
net = Net()

# Define loss function and optimizer
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

Training Loop with Progress Updates

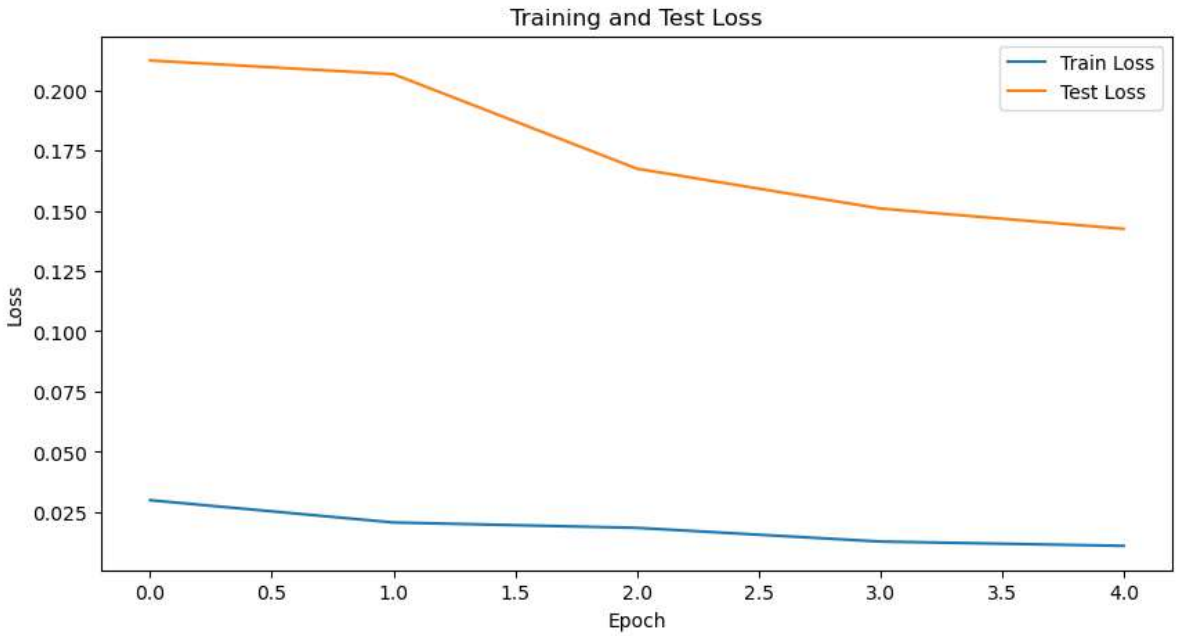
In [79]:

```
import matplotlib.pyplot as plt

# Training loop with progress updates and loss tracking
num_epochs = 5
train_losses = []
test_losses = []

for epoch in range(num_epochs):
    running_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        inputs, labels = data['image'], data['label']
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

        if i % 100 == 99:
            print(f"Epoch [{epoch+1}/{num_epochs}], Batch [{i+1}/{len(train_loader
            running_loss = 0.0

    # Calculate and store train loss for each epoch
    train_loss = running_loss / len(train_loader)
    train_losses.append(train_loss)

    # Calculate and store test loss for each epoch
    test_loss = 0.0
    total_samples = 0
    with torch.no_grad():
        for data in test_loader:
            inputs, labels = data['image'], data['label']
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            test_loss += loss.item() * labels.size(0)
            total_samples += labels.size(0)
        test_loss /= total_samples
        test_losses.append(test_loss)

print("Training finished.")
```

```
Epoch [1/5], Batch [100/460], Loss: 1.1128
Epoch [1/5], Batch [200/460], Loss: 0.3548
Epoch [1/5], Batch [300/460], Loss: 0.2653
Epoch [1/5], Batch [400/460], Loss: 0.2666
Epoch [2/5], Batch [100/460], Loss: 0.1682
Epoch [2/5], Batch [200/460], Loss: 0.1890
Epoch [2/5], Batch [300/460], Loss: 0.1651
Epoch [2/5], Batch [400/460], Loss: 0.1833
Epoch [3/5], Batch [100/460], Loss: 0.1326
Epoch [3/5], Batch [200/460], Loss: 0.1274
Epoch [3/5], Batch [300/460], Loss: 0.1393
Epoch [3/5], Batch [400/460], Loss: 0.1286
Epoch [4/5], Batch [100/460], Loss: 0.1061
Epoch [4/5], Batch [200/460], Loss: 0.0987
Epoch [4/5], Batch [300/460], Loss: 0.0923
Epoch [4/5], Batch [400/460], Loss: 0.1061
Epoch [5/5], Batch [100/460], Loss: 0.0709
Epoch [5/5], Batch [200/460], Loss: 0.0792
Epoch [5/5], Batch [300/460], Loss: 0.0751
Epoch [5/5], Batch [400/460], Loss: 0.0874
Training finished.
```

Graph Plot

In [80]:
```python
# Plot the loss values
plt.figure(figsize=(10, 5))
plt.plot(train_losses, label='Train Loss')
plt.plot(test_losses, label='Test Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Test Loss')
plt.legend()
plt.show()
```



Testing the Model

In [81]:
```python
# Evaluate the model
correct = 0
total = 0
with torch.no_grad():
```

```
    for data in test_loader:
        inputs, labels = data['image'], data['label']
        outputs = net(inputs)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"Accuracy on test data: {100 * correct / total:.2f}%")
```

```
Accuracy on test data: 96.15%
```