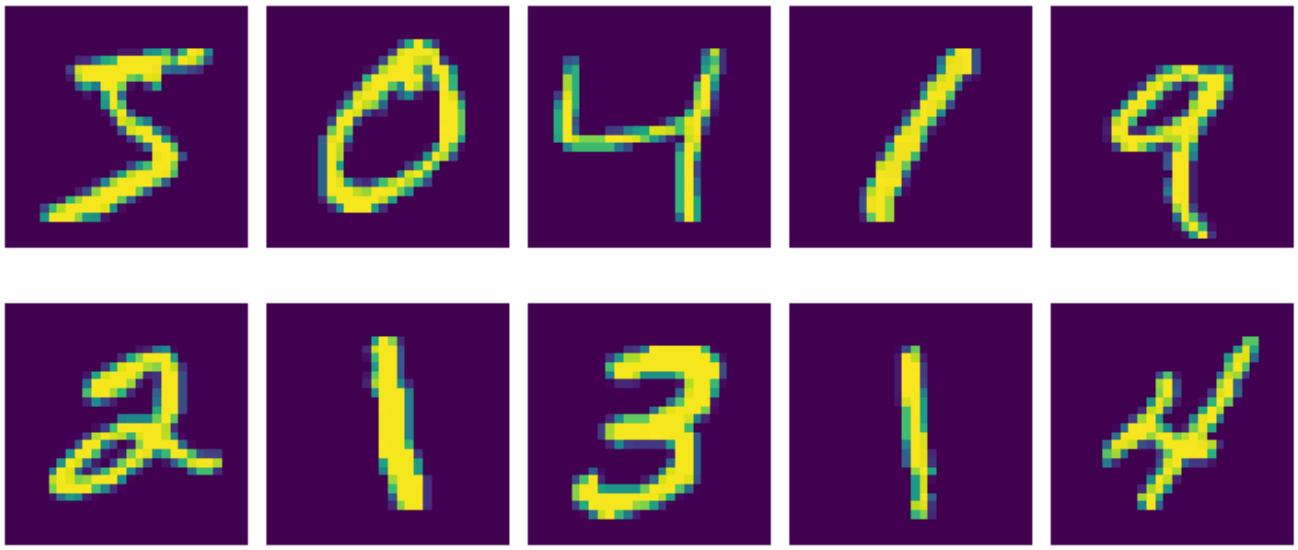
In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import torch
import torch.nn as nn
import torch.optim as optim
```

In [2]:
```
# Use CPU for this simple dataset
device = torch.device("cpu")
```

In [3]:
```
# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
```

In [4]:
```
# Display some images from the dataset
fig, axes = plt.subplots(2, 5, figsize=(10, 5))
for i, ax in enumerate(axes.flatten()):
    ax.imshow(x_train[i])
    #ax.set_title(f'Label: {y_train[i][0]}')
    ax.axis('off')

plt.tight_layout()
plt.show()
```



In [5]:
```
# Normalize images
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
```

In [6]:
```
# Flatten images
X_train = x_train.reshape(x_train.shape[0], -1)
X_test = x_test.reshape(x_test.shape[0], -1)
```

In [7]:
```
# Flatten labels
y_train = y_train.flatten()
y_test = y_test.flatten()
```

In [8]:
```
# Train a simple SVM classifier
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)
print("SVM Accuracy:", accuracy_score(y_test, y_pred))
```

```
SVM Accuracy: 0.9404
```

In [9]:
```
# Build a simple Neural Network model using PyTorch
class NeuralNetwork(nn.Module):
    def __init__(self, input_size, num_classes):
        super(NeuralNetwork, self).__init__()
        self.fc1 = nn.Linear(input_size, 64)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, num_classes)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.softmax(self.fc3(x))
        return x
```

In [10]:
```
# Initialize model, loss, and optimizer
model = NeuralNetwork(X_train.shape[1], 10)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

In [11]:
```
# Convert data to tensors
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)
```

In [12]:
```
# Track loss and accuracy
losses = []
accuracies = []
```

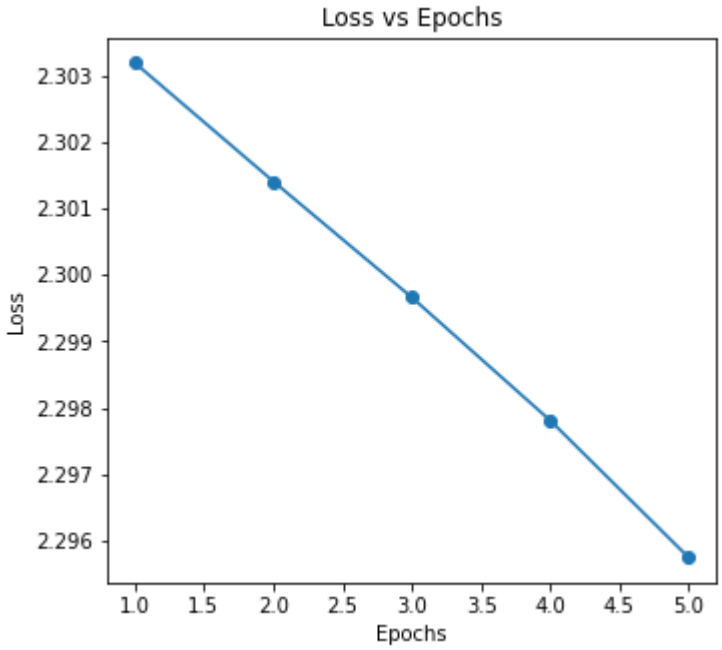
In [18]:
```
# Train the model
epochs = 5
for epoch in range(epochs):
    optimizer.zero_grad()
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)
    loss.backward()
    optimizer.step()

    # Track loss
```

```
    losses.append(loss.item())

    # Calculate accuracy
    _, predicted = torch.max(outputs, 1)
    accuracy = (predicted == y_train_tensor).sum().item() / y_train_tensor.size(0)
    accuracies.append(accuracy)

    print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}, Accuracy: {accuracy
```

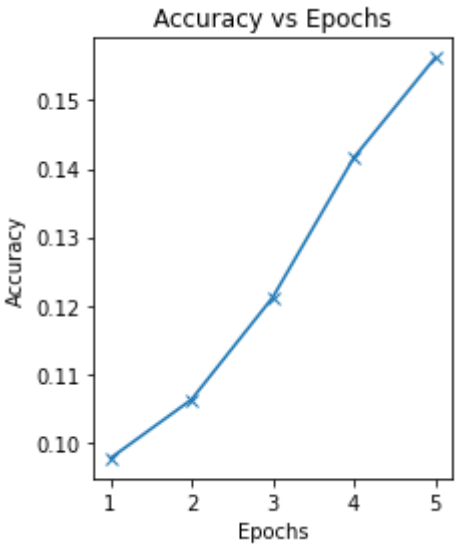
```
Epoch [1/5], Loss: 2.2934, Accuracy: 0.1646
Epoch [2/5], Loss: 2.2907, Accuracy: 0.1706
Epoch [3/5], Loss: 2.2876, Accuracy: 0.1811
Epoch [4/5], Loss: 2.2842, Accuracy: 0.2062
Epoch [5/5], Loss: 2.2802, Accuracy: 0.2436
```

In [14]:
```
# Plot Loss vs Epoch
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(1, epochs+1), losses, marker='o')
plt.title('Loss vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

Out [14]: `Text(0, 0.5, 'Loss')`



In [15]:
```
# Plot Accuracy vs Epoch
plt.subplot(1, 2, 2)
plt.plot(range(1, epochs+1), accuracies, marker='x')
plt.title('Accuracy vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')

plt.tight_layout()
plt.show()
```



In [16]:
```
# Evaluate the model
with torch.no_grad():
    outputs = model(X_test_tensor)
    _, predicted = torch.max(outputs, 1)
    accuracy = (predicted == y_test_tensor).sum().item() / y_test_tensor.size(0)
    print("Neural Network Accuracy:", accuracy)
```

```
Neural Network Accuracy: 0.1662
```

In [ ]: