Fashion-MNIST-Dataset-Hossam

April 2, 2025

[1]:

```
import pandas as pd
import numpy as np
import torch
from torch import optim as optim
from torch import nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt
```

[ ]:

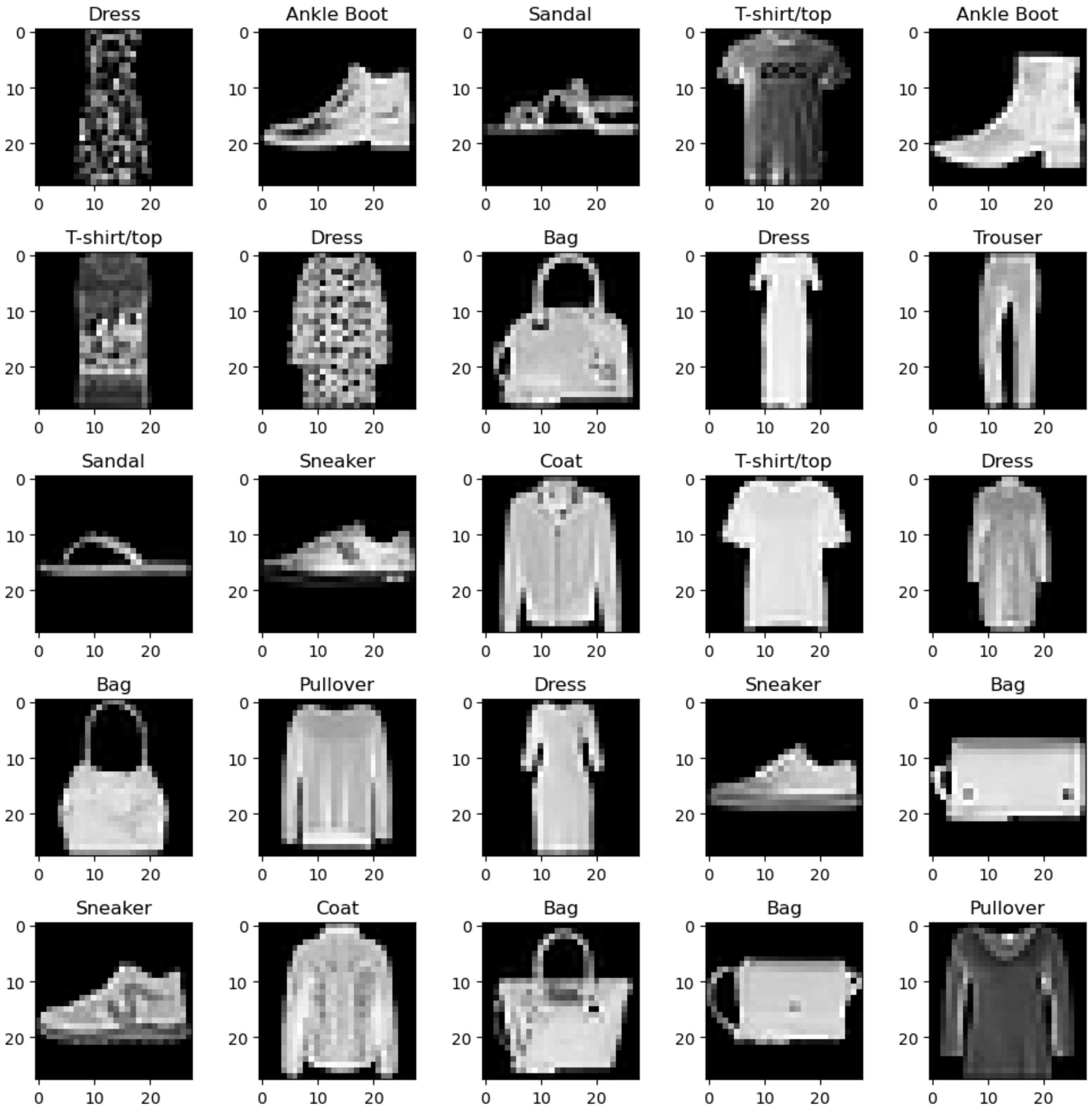
```
import torchvision
import torchvision.transforms as transforms
import random
np.random.seed(0)
torch.manual_seed(0)
random.seed(0)
```

[3]:

```
transform = transforms.ToTensor()

training_set = torchvision.datasets.FashionMNIST('./data', train=True,␣
 ↪transform=transform, download=True)
validation_set = torchvision.datasets.FashionMNIST('./data', train=False,␣
 ↪transform=transform, download=True)

training_loader = torch.utils.data.DataLoader(training_set, batch_size=32,␣
 ↪shuffle=True)
validation_loader = torch.utils.data.DataLoader(validation_set, batch_size=128,␣
 ↪shuffle=False)

classes = ('T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
        'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle Boot')
```

[4]:

```
batch = next(iter(training_loader))
plt.figure(figsize=(10, 10))
for i, (image, label) in enumerate(zip(*batch)):
    if i > 24:
        break
    plt.subplot(5, 5, i + 1)
    plt.imshow(image[0], cmap="gray")
    plt.title(classes[label])

plt.tight_layout()
```



1 Model 1

[79]:
```
class MLP1(torch.nn.Module):
    def __init__(self, input_shape, hidden_dim, output_shape):
        super(MLP1,self).__init__()
        self.fc1 = nn.Linear(input_shape, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, hidden_dim)
        self.fc3 = nn.Linear(hidden_dim, hidden_dim)
        self.output = nn.Linear(hidden_dim, output_shape)

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

1.0.1 Model 1 Architecture

[113]:

```
input_size = 28*28  # Flattened image
hidden_dim = 128
output_size = 10        # 10 classes
```

[114]:

```
model1 = MLP1(input_size, hidden_dim, output_size)
model = model1
model
```

[114]:

```
MLP1(
  (fc1): Linear(in_features=784, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=128, bias=True)
  (output): Linear(in_features=128, out_features=10, bias=True)
)
```

2 Loss

[115]:

```
loss_fn = nn.CrossEntropyLoss()
```

3 Optimizer

[116]:

```
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

[149]:

```
def train_model(model, optimizer, training_loader, criterion=loss_fn,␣
 ↪no_epochs=5):
    model.train()
    batches = []
    losses = []
    j = 0
    for epoch in range(no_epochs):
        running_loss = 0
        correct = 0
        total = 0
        for i, (images, labels) in enumerate(training_loader):
            images = images.view(-1, 28 * 28)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            if i % 100 == 99:
                losses.append(running_loss/100)
                j += i
                batches.append(j)
                print(f"Epoch: {epoch}, Batch: {i+1}, Loss: {running_loss/100:.
 ↪3f}, Accuracy: {100*correct/total:.2f}%")
                running_loss = 0

        if epoch % 2 == 0:
            print(f"Epoch {epoch+1} completed")

    return model, losses, batches
```

[118]:

```
def plot_loss(losses, batches):
    plt.plot(batches, losses)
    plt.xlabel('Batches')
    plt.ylabel('Loss')
    plt.title('Loss vs. Batches')
    plt.show()
```

[119]:

```
model1, losses, batches = train_model(model, optimizer, training_loader,␣
 ↪loss_fn, no_epochs=5)
```

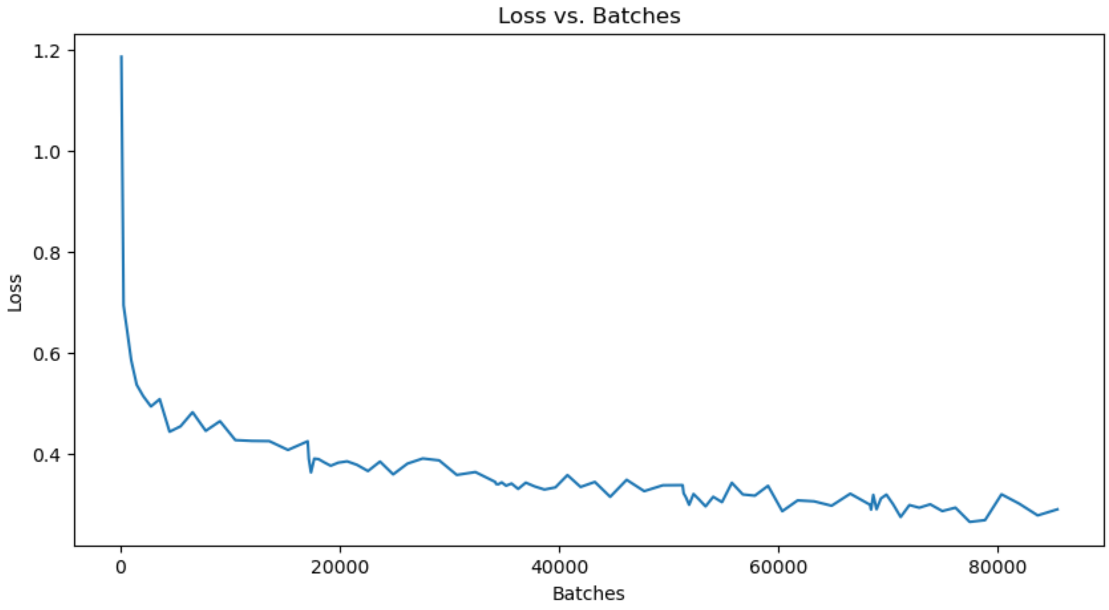
```
Epoch: 0, Batch: 99, Loss: 1.186, Accuracy: 55.34%
Epoch: 0, Batch: 199, Loss: 0.695, Accuracy: 64.31%
Epoch: 0, Batch: 299, Loss: 0.649, Accuracy: 67.81%
Epoch: 0, Batch: 399, Loss: 0.586, Accuracy: 70.77%
Epoch: 0, Batch: 499, Loss: 0.538, Accuracy: 72.81%
Epoch: 0, Batch: 599, Loss: 0.515, Accuracy: 74.26%
Epoch: 0, Batch: 699, Loss: 0.495, Accuracy: 75.39%
Epoch: 0, Batch: 799, Loss: 0.509, Accuracy: 76.13%
Epoch: 0, Batch: 899, Loss: 0.445, Accuracy: 76.98%
Epoch: 0, Batch: 999, Loss: 0.455, Accuracy: 77.62%
```

```
Epoch: 0, Batch: 1099, Loss: 0.483, Accuracy: 78.10%
Epoch: 0, Batch: 1199, Loss: 0.447, Accuracy: 78.54%
Epoch: 0, Batch: 1299, Loss: 0.466, Accuracy: 78.89%
Epoch: 0, Batch: 1399, Loss: 0.428, Accuracy: 79.29%
Epoch: 0, Batch: 1499, Loss: 0.427, Accuracy: 79.64%
Epoch: 0, Batch: 1599, Loss: 0.426, Accuracy: 79.93%
Epoch: 0, Batch: 1699, Loss: 0.409, Accuracy: 80.24%
Epoch: 0, Batch: 1799, Loss: 0.426, Accuracy: 80.49%
Epoch 1 completed
Epoch: 1, Batch: 99, Loss: 0.391, Accuracy: 85.72%
Epoch: 1, Batch: 199, Loss: 0.364, Accuracy: 86.16%
Epoch: 1, Batch: 299, Loss: 0.392, Accuracy: 85.91%
Epoch: 1, Batch: 399, Loss: 0.391, Accuracy: 85.73%
Epoch: 1, Batch: 499, Loss: 0.384, Accuracy: 85.75%
Epoch: 1, Batch: 599, Loss: 0.377, Accuracy: 85.83%
Epoch: 1, Batch: 699, Loss: 0.384, Accuracy: 85.92%
Epoch: 1, Batch: 799, Loss: 0.386, Accuracy: 85.90%
Epoch: 1, Batch: 899, Loss: 0.379, Accuracy: 85.95%
Epoch: 1, Batch: 999, Loss: 0.367, Accuracy: 85.97%
Epoch: 1, Batch: 1099, Loss: 0.386, Accuracy: 85.95%
Epoch: 1, Batch: 1199, Loss: 0.360, Accuracy: 86.01%
Epoch: 1, Batch: 1299, Loss: 0.382, Accuracy: 86.00%
Epoch: 1, Batch: 1399, Loss: 0.392, Accuracy: 85.94%
Epoch: 1, Batch: 1499, Loss: 0.388, Accuracy: 85.94%
Epoch: 1, Batch: 1599, Loss: 0.359, Accuracy: 86.00%
Epoch: 1, Batch: 1699, Loss: 0.365, Accuracy: 86.04%
Epoch: 1, Batch: 1799, Loss: 0.346, Accuracy: 86.05%
Epoch: 2, Batch: 99, Loss: 0.341, Accuracy: 87.69%
Epoch: 2, Batch: 199, Loss: 0.341, Accuracy: 87.36%
Epoch: 2, Batch: 299, Loss: 0.345, Accuracy: 87.41%
Epoch: 2, Batch: 399, Loss: 0.338, Accuracy: 87.23%
Epoch: 2, Batch: 499, Loss: 0.342, Accuracy: 87.15%
Epoch: 2, Batch: 599, Loss: 0.332, Accuracy: 87.24%
Epoch: 2, Batch: 699, Loss: 0.344, Accuracy: 87.23%
Epoch: 2, Batch: 799, Loss: 0.337, Accuracy: 87.32%
Epoch: 2, Batch: 899, Loss: 0.330, Accuracy: 87.37%
Epoch: 2, Batch: 999, Loss: 0.335, Accuracy: 87.41%
Epoch: 2, Batch: 1099, Loss: 0.359, Accuracy: 87.36%
Epoch: 2, Batch: 1199, Loss: 0.336, Accuracy: 87.36%
Epoch: 2, Batch: 1299, Loss: 0.346, Accuracy: 87.31%
Epoch: 2, Batch: 1399, Loss: 0.316, Accuracy: 87.35%
Epoch: 2, Batch: 1499, Loss: 0.350, Accuracy: 87.36%
Epoch: 2, Batch: 1599, Loss: 0.327, Accuracy: 87.40%
Epoch: 2, Batch: 1699, Loss: 0.339, Accuracy: 87.43%
Epoch: 2, Batch: 1799, Loss: 0.339, Accuracy: 87.42%
Epoch 3 completed
Epoch: 3, Batch: 99, Loss: 0.323, Accuracy: 88.31%
Epoch: 3, Batch: 199, Loss: 0.315, Accuracy: 88.19%
```

```
Epoch: 3, Batch: 299, Loss: 0.300, Accuracy: 88.25%
Epoch: 3, Batch: 399, Loss: 0.322, Accuracy: 88.26%
Epoch: 3, Batch: 499, Loss: 0.311, Accuracy: 88.31%
Epoch: 3, Batch: 599, Loss: 0.297, Accuracy: 88.39%
Epoch: 3, Batch: 699, Loss: 0.316, Accuracy: 88.44%
Epoch: 3, Batch: 799, Loss: 0.306, Accuracy: 88.47%
Epoch: 3, Batch: 899, Loss: 0.344, Accuracy: 88.39%
Epoch: 3, Batch: 999, Loss: 0.321, Accuracy: 88.42%
Epoch: 3, Batch: 1099, Loss: 0.318, Accuracy: 88.36%
Epoch: 3, Batch: 1199, Loss: 0.338, Accuracy: 88.26%
Epoch: 3, Batch: 1299, Loss: 0.288, Accuracy: 88.32%
Epoch: 3, Batch: 1399, Loss: 0.309, Accuracy: 88.35%
Epoch: 3, Batch: 1499, Loss: 0.307, Accuracy: 88.36%
Epoch: 3, Batch: 1599, Loss: 0.298, Accuracy: 88.41%
Epoch: 3, Batch: 1699, Loss: 0.322, Accuracy: 88.37%
Epoch: 3, Batch: 1799, Loss: 0.300, Accuracy: 88.40%
Epoch: 4, Batch: 99, Loss: 0.291, Accuracy: 88.75%
Epoch: 4, Batch: 199, Loss: 0.320, Accuracy: 88.20%
Epoch: 4, Batch: 299, Loss: 0.291, Accuracy: 88.59%
Epoch: 4, Batch: 399, Loss: 0.313, Accuracy: 88.64%
Epoch: 4, Batch: 499, Loss: 0.320, Accuracy: 88.49%
Epoch: 4, Batch: 599, Loss: 0.302, Accuracy: 88.59%
Epoch: 4, Batch: 699, Loss: 0.276, Accuracy: 88.76%
Epoch: 4, Batch: 799, Loss: 0.300, Accuracy: 88.78%
Epoch: 4, Batch: 899, Loss: 0.295, Accuracy: 88.76%
Epoch: 4, Batch: 999, Loss: 0.301, Accuracy: 88.79%
Epoch: 4, Batch: 1099, Loss: 0.288, Accuracy: 88.87%
Epoch: 4, Batch: 1199, Loss: 0.295, Accuracy: 88.86%
Epoch: 4, Batch: 1299, Loss: 0.266, Accuracy: 88.90%
Epoch: 4, Batch: 1399, Loss: 0.270, Accuracy: 89.00%
Epoch: 4, Batch: 1499, Loss: 0.321, Accuracy: 88.98%
Epoch: 4, Batch: 1599, Loss: 0.303, Accuracy: 88.96%
Epoch: 4, Batch: 1699, Loss: 0.279, Accuracy: 89.02%
Epoch: 4, Batch: 1799, Loss: 0.291, Accuracy: 89.03%
Epoch 5 completed
```

[120]:

```
plt.figure(figsize=(10, 5))
plot_loss(losses, batches)
```



[ ]:

```python
def evaluate_model(model, loader):
    model.eval()  # Set the model to evaluation mode
    correct = 0   # Counter for correctly classified samples
    total = 0


    with torch.no_grad():
        for images, labels in loader:
            images = images.view(-1, 28 * 28)  # Iterate through the DataLoader
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)

            # Update total sample count
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print(f"\nTest Accuracy: {100 * correct / total:.2f}%")
```

[122]:

```python
evaluate_model(model1, validation_loader)
```

```
Test Accuracy: 86.01%
```

4 Model 2

Let's change the layers and the neurons

[123]:

```
class MLP2(torch.nn.Module):
    def __init__(self, input_shape, hidden_dim, output_shape):
        super(MLP2,self).__init__()
        self.fc1 = nn.Linear(input_shape, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, hidden_dim*2)
        self.fc3 = nn.Linear(hidden_dim*2, hidden_dim)
        self.fc4 = nn.Linear(hidden_dim, hidden_dim//2)
        self.output = nn.Linear(hidden_dim//2, output_shape)

    def forward(self, x):
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc3(x)
        x = F.relu(x)
        x = self.fc4(x)
        x = F.relu(x)
        x = self.output(x)
        return x
```

4.0.1 Model 2 Architecture

[124]:

```
input_size = 28*28  # Flattened image
hidden_dim = 128
output_size = 10       # 10 classes

model2 = MLP2(input_size, hidden_dim, output_size)
model2
```

[124]:

```
MLP2(
  (fc1): Linear(in_features=784, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=128, bias=True)
  (fc4): Linear(in_features=128, out_features=64, bias=True)
  (output): Linear(in_features=64, out_features=10, bias=True)
)
```

[125]:

```
model = model2
optimizer = optim.Adam(model.parameters(), lr=0.001)

model2, losses, batches  = train_model(model, optimizer, training_loader,␣
 ↪loss_fn, no_epochs=5)
```

```
Epoch: 0, Batch: 99, Loss: 1.305, Accuracy: 49.88%
Epoch: 0, Batch: 199, Loss: 0.794, Accuracy: 59.78%
Epoch: 0, Batch: 299, Loss: 0.673, Accuracy: 64.76%
Epoch: 0, Batch: 399, Loss: 0.653, Accuracy: 67.70%
```

```
Epoch: 0, Batch: 499, Loss: 0.564, Accuracy: 70.22%
Epoch: 0, Batch: 599, Loss: 0.565, Accuracy: 71.86%
Epoch: 0, Batch: 699, Loss: 0.505, Accuracy: 73.31%
Epoch: 0, Batch: 799, Loss: 0.530, Accuracy: 74.25%
Epoch: 0, Batch: 899, Loss: 0.479, Accuracy: 75.20%
Epoch: 0, Batch: 999, Loss: 0.490, Accuracy: 75.95%
Epoch: 0, Batch: 1099, Loss: 0.503, Accuracy: 76.54%
Epoch: 0, Batch: 1199, Loss: 0.462, Accuracy: 77.09%
Epoch: 0, Batch: 1299, Loss: 0.458, Accuracy: 77.55%
Epoch: 0, Batch: 1399, Loss: 0.438, Accuracy: 77.98%
Epoch: 0, Batch: 1499, Loss: 0.460, Accuracy: 78.32%
Epoch: 0, Batch: 1599, Loss: 0.460, Accuracy: 78.60%
Epoch: 0, Batch: 1699, Loss: 0.419, Accuracy: 78.92%
Epoch: 0, Batch: 1799, Loss: 0.439, Accuracy: 79.17%
Epoch 1 completed
Epoch: 1, Batch: 99, Loss: 0.409, Accuracy: 84.81%
Epoch: 1, Batch: 199, Loss: 0.428, Accuracy: 84.59%
Epoch: 1, Batch: 299, Loss: 0.406, Accuracy: 84.76%
Epoch: 1, Batch: 399, Loss: 0.397, Accuracy: 85.00%
Epoch: 1, Batch: 499, Loss: 0.384, Accuracy: 85.14%
Epoch: 1, Batch: 599, Loss: 0.383, Accuracy: 85.20%
Epoch: 1, Batch: 699, Loss: 0.366, Accuracy: 85.43%
Epoch: 1, Batch: 799, Loss: 0.399, Accuracy: 85.41%
Epoch: 1, Batch: 899, Loss: 0.393, Accuracy: 85.48%
Epoch: 1, Batch: 999, Loss: 0.393, Accuracy: 85.51%
Epoch: 1, Batch: 1099, Loss: 0.385, Accuracy: 85.52%
Epoch: 1, Batch: 1199, Loss: 0.379, Accuracy: 85.54%
Epoch: 1, Batch: 1299, Loss: 0.367, Accuracy: 85.59%
Epoch: 1, Batch: 1399, Loss: 0.399, Accuracy: 85.56%
Epoch: 1, Batch: 1499, Loss: 0.386, Accuracy: 85.59%
Epoch: 1, Batch: 1599, Loss: 0.373, Accuracy: 85.64%
Epoch: 1, Batch: 1699, Loss: 0.369, Accuracy: 85.72%
Epoch: 1, Batch: 1799, Loss: 0.358, Accuracy: 85.76%
Epoch: 2, Batch: 99, Loss: 0.341, Accuracy: 86.69%
Epoch: 2, Batch: 199, Loss: 0.355, Accuracy: 86.81%
Epoch: 2, Batch: 299, Loss: 0.367, Accuracy: 86.59%
Epoch: 2, Batch: 399, Loss: 0.368, Accuracy: 86.58%
Epoch: 2, Batch: 499, Loss: 0.353, Accuracy: 86.75%
Epoch: 2, Batch: 599, Loss: 0.368, Accuracy: 86.78%
Epoch: 2, Batch: 699, Loss: 0.365, Accuracy: 86.78%
Epoch: 2, Batch: 799, Loss: 0.337, Accuracy: 86.84%
Epoch: 2, Batch: 899, Loss: 0.335, Accuracy: 86.98%
Epoch: 2, Batch: 999, Loss: 0.334, Accuracy: 87.08%
Epoch: 2, Batch: 1099, Loss: 0.340, Accuracy: 87.11%
Epoch: 2, Batch: 1199, Loss: 0.348, Accuracy: 87.14%
Epoch: 2, Batch: 1299, Loss: 0.337, Accuracy: 87.19%
Epoch: 2, Batch: 1399, Loss: 0.351, Accuracy: 87.20%
Epoch: 2, Batch: 1499, Loss: 0.347, Accuracy: 87.20%
```

```
Epoch: 2, Batch: 1599, Loss: 0.344, Accuracy: 87.21%
Epoch: 2, Batch: 1699, Loss: 0.327, Accuracy: 87.23%
Epoch: 2, Batch: 1799, Loss: 0.338, Accuracy: 87.25%
Epoch 3 completed
Epoch: 3, Batch: 99, Loss: 0.315, Accuracy: 88.25%
Epoch: 3, Batch: 199, Loss: 0.323, Accuracy: 88.42%
Epoch: 3, Batch: 299, Loss: 0.320, Accuracy: 88.33%
Epoch: 3, Batch: 399, Loss: 0.311, Accuracy: 88.42%
Epoch: 3, Batch: 499, Loss: 0.344, Accuracy: 88.12%
Epoch: 3, Batch: 599, Loss: 0.337, Accuracy: 88.03%
Epoch: 3, Batch: 699, Loss: 0.342, Accuracy: 88.02%
Epoch: 3, Batch: 799, Loss: 0.324, Accuracy: 88.08%
Epoch: 3, Batch: 899, Loss: 0.331, Accuracy: 88.07%
Epoch: 3, Batch: 999, Loss: 0.312, Accuracy: 88.10%
Epoch: 3, Batch: 1099, Loss: 0.312, Accuracy: 88.07%
Epoch: 3, Batch: 1199, Loss: 0.306, Accuracy: 88.14%
Epoch: 3, Batch: 1299, Loss: 0.314, Accuracy: 88.14%
Epoch: 3, Batch: 1399, Loss: 0.343, Accuracy: 88.11%
Epoch: 3, Batch: 1499, Loss: 0.333, Accuracy: 88.10%
Epoch: 3, Batch: 1599, Loss: 0.322, Accuracy: 88.14%
Epoch: 3, Batch: 1699, Loss: 0.321, Accuracy: 88.13%
Epoch: 3, Batch: 1799, Loss: 0.314, Accuracy: 88.12%
Epoch: 4, Batch: 99, Loss: 0.294, Accuracy: 89.31%
Epoch: 4, Batch: 199, Loss: 0.304, Accuracy: 88.89%
Epoch: 4, Batch: 299, Loss: 0.287, Accuracy: 89.22%
Epoch: 4, Batch: 399, Loss: 0.292, Accuracy: 89.14%
Epoch: 4, Batch: 499, Loss: 0.318, Accuracy: 88.91%
Epoch: 4, Batch: 599, Loss: 0.328, Accuracy: 88.70%
Epoch: 4, Batch: 699, Loss: 0.324, Accuracy: 88.61%
Epoch: 4, Batch: 799, Loss: 0.282, Accuracy: 88.62%
Epoch: 4, Batch: 899, Loss: 0.319, Accuracy: 88.53%
Epoch: 4, Batch: 999, Loss: 0.314, Accuracy: 88.55%
Epoch: 4, Batch: 1099, Loss: 0.320, Accuracy: 88.52%
Epoch: 4, Batch: 1199, Loss: 0.294, Accuracy: 88.53%
Epoch: 4, Batch: 1299, Loss: 0.294, Accuracy: 88.54%
Epoch: 4, Batch: 1399, Loss: 0.279, Accuracy: 88.62%
Epoch: 4, Batch: 1499, Loss: 0.310, Accuracy: 88.64%
Epoch: 4, Batch: 1599, Loss: 0.294, Accuracy: 88.69%
Epoch: 4, Batch: 1699, Loss: 0.301, Accuracy: 88.73%
Epoch: 4, Batch: 1799, Loss: 0.298, Accuracy: 88.74%
Epoch 5 completed
```

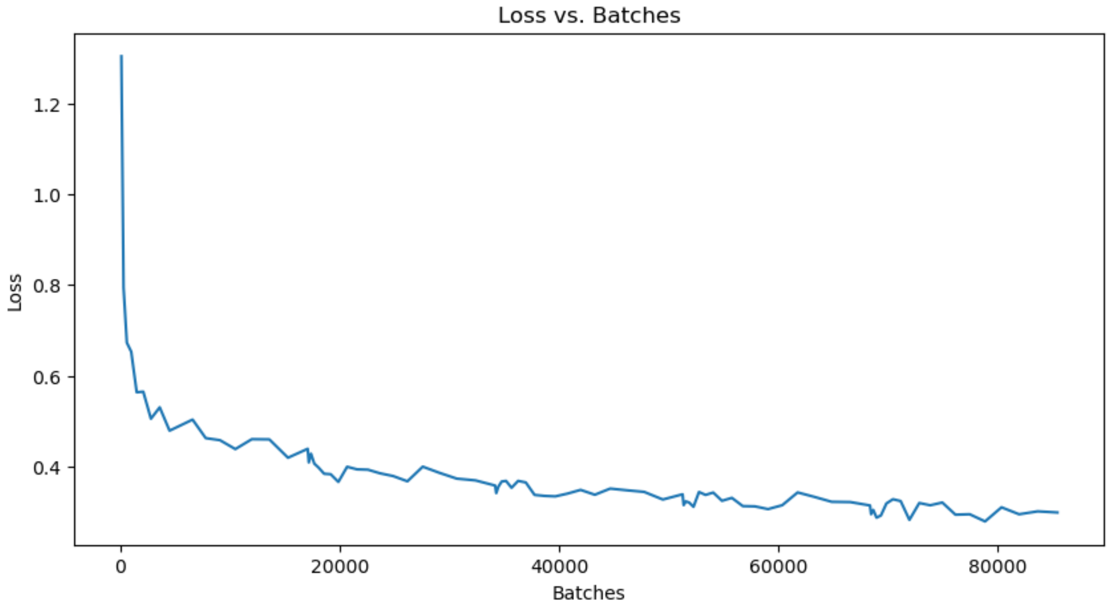
[126]:

```
evaluate_model(model2, validation_loader)
```

```
Test Accuracy: 87.49%
```

[127]:

```
plt.figure(figsize=(10, 5))
plot_loss(losses, batches)
```



5 Model 3

Let’s try the same architecture of model 1 but change the activation fn and the optimizer.

[156]:

```
class MLP3(torch.nn.Module):
    def __init__(self, input_shape, hidden_dim, output_shape):
        super(MLP3,self).__init__()
        self.fc1 = nn.Linear(input_shape, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, hidden_dim)
        self.fc3 = nn.Linear(hidden_dim, hidden_dim)
        self.output = nn.Linear(hidden_dim, output_shape)

    def forward(self, x):
        x = self.fc1(x)
        x = F.leaky_relu(x)
        x = self.fc2(x)
        x = F.leaky_relu(x)
        x = self.fc3(x)
        x = F.leaky_relu(x)
        x = self.output(x)
        return x
```

[157]:

```
input_size = 28*28  # Flattened image
hidden_dim = 128
output_size = 10
```

[158]:

```
model3 = MLP3(input_size, hidden_dim, output_size)
model3
```

[158]:

```
MLP3(
  (fc1): Linear(in_features=784, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=128, bias=True)
  (output): Linear(in_features=128, out_features=10, bias=True)
)
```

[159]:

```
model = model3
optimizer = optim.NAdam(model.parameters(), lr=0.001)

model3, losses, batches  = train_model(model, optimizer, training_loader,␣
 ↪loss_fn, no_epochs=5)
```

```
Epoch: 0, Batch: 100, Loss: 1.133, Accuracy: 57.12%
Epoch: 0, Batch: 200, Loss: 0.675, Accuracy: 65.86%
Epoch: 0, Batch: 300, Loss: 0.592, Accuracy: 70.12%
Epoch: 0, Batch: 400, Loss: 0.533, Accuracy: 72.68%
Epoch: 0, Batch: 500, Loss: 0.504, Accuracy: 74.59%
Epoch: 0, Batch: 600, Loss: 0.499, Accuracy: 75.98%
Epoch: 0, Batch: 700, Loss: 0.472, Accuracy: 76.96%
Epoch: 0, Batch: 800, Loss: 0.450, Accuracy: 77.86%
Epoch: 0, Batch: 900, Loss: 0.458, Accuracy: 78.53%
Epoch: 0, Batch: 1000, Loss: 0.434, Accuracy: 79.11%
Epoch: 0, Batch: 1100, Loss: 0.445, Accuracy: 79.52%
Epoch: 0, Batch: 1200, Loss: 0.433, Accuracy: 79.88%
Epoch: 0, Batch: 1300, Loss: 0.417, Accuracy: 80.26%
Epoch: 0, Batch: 1400, Loss: 0.427, Accuracy: 80.51%
Epoch: 0, Batch: 1500, Loss: 0.403, Accuracy: 80.80%
Epoch: 0, Batch: 1600, Loss: 0.385, Accuracy: 81.16%
Epoch: 0, Batch: 1700, Loss: 0.400, Accuracy: 81.40%
Epoch: 0, Batch: 1800, Loss: 0.388, Accuracy: 81.69%
Epoch 1 completed
Epoch: 1, Batch: 100, Loss: 0.341, Accuracy: 88.03%
Epoch: 1, Batch: 200, Loss: 0.383, Accuracy: 87.05%
Epoch: 1, Batch: 300, Loss: 0.343, Accuracy: 87.04%
Epoch: 1, Batch: 400, Loss: 0.368, Accuracy: 86.88%
Epoch: 1, Batch: 500, Loss: 0.363, Accuracy: 86.78%
Epoch: 1, Batch: 600, Loss: 0.380, Accuracy: 86.61%
Epoch: 1, Batch: 700, Loss: 0.352, Accuracy: 86.62%
Epoch: 1, Batch: 800, Loss: 0.359, Accuracy: 86.61%
Epoch: 1, Batch: 900, Loss: 0.347, Accuracy: 86.76%
```

```
Epoch: 1, Batch: 1000, Loss: 0.366, Accuracy: 86.75%
Epoch: 1, Batch: 1100, Loss: 0.340, Accuracy: 86.84%
Epoch: 1, Batch: 1200, Loss: 0.347, Accuracy: 86.87%
Epoch: 1, Batch: 1300, Loss: 0.350, Accuracy: 86.88%
Epoch: 1, Batch: 1400, Loss: 0.348, Accuracy: 86.89%
Epoch: 1, Batch: 1500, Loss: 0.347, Accuracy: 86.92%
Epoch: 1, Batch: 1600, Loss: 0.339, Accuracy: 86.98%
Epoch: 1, Batch: 1700, Loss: 0.361, Accuracy: 86.97%
Epoch: 1, Batch: 1800, Loss: 0.357, Accuracy: 86.93%
Epoch: 2, Batch: 100, Loss: 0.325, Accuracy: 87.47%
Epoch: 2, Batch: 200, Loss: 0.310, Accuracy: 87.72%
Epoch: 2, Batch: 300, Loss: 0.347, Accuracy: 87.69%
Epoch: 2, Batch: 400, Loss: 0.323, Accuracy: 87.77%
Epoch: 2, Batch: 500, Loss: 0.330, Accuracy: 87.75%
Epoch: 2, Batch: 600, Loss: 0.309, Accuracy: 87.94%
Epoch: 2, Batch: 700, Loss: 0.318, Accuracy: 88.02%
Epoch: 2, Batch: 800, Loss: 0.325, Accuracy: 88.02%
Epoch: 2, Batch: 900, Loss: 0.322, Accuracy: 88.00%
Epoch: 2, Batch: 1000, Loss: 0.311, Accuracy: 88.07%
Epoch: 2, Batch: 1100, Loss: 0.330, Accuracy: 88.08%
Epoch: 2, Batch: 1200, Loss: 0.320, Accuracy: 88.12%
Epoch: 2, Batch: 1300, Loss: 0.313, Accuracy: 88.17%
Epoch: 2, Batch: 1400, Loss: 0.325, Accuracy: 88.18%
Epoch: 2, Batch: 1500, Loss: 0.317, Accuracy: 88.15%
Epoch: 2, Batch: 1600, Loss: 0.324, Accuracy: 88.14%
Epoch: 2, Batch: 1700, Loss: 0.310, Accuracy: 88.16%
Epoch: 2, Batch: 1800, Loss: 0.312, Accuracy: 88.22%
Epoch 3 completed
Epoch: 3, Batch: 100, Loss: 0.304, Accuracy: 89.25%
Epoch: 3, Batch: 200, Loss: 0.275, Accuracy: 89.48%
Epoch: 3, Batch: 300, Loss: 0.299, Accuracy: 89.40%
Epoch: 3, Batch: 400, Loss: 0.293, Accuracy: 89.23%
Epoch: 3, Batch: 500, Loss: 0.302, Accuracy: 88.91%
Epoch: 3, Batch: 600, Loss: 0.315, Accuracy: 88.78%
Epoch: 3, Batch: 700, Loss: 0.284, Accuracy: 88.88%
Epoch: 3, Batch: 800, Loss: 0.304, Accuracy: 88.82%
Epoch: 3, Batch: 900, Loss: 0.318, Accuracy: 88.76%
Epoch: 3, Batch: 1000, Loss: 0.295, Accuracy: 88.80%
Epoch: 3, Batch: 1100, Loss: 0.310, Accuracy: 88.82%
Epoch: 3, Batch: 1200, Loss: 0.305, Accuracy: 88.84%
Epoch: 3, Batch: 1300, Loss: 0.306, Accuracy: 88.87%
Epoch: 3, Batch: 1400, Loss: 0.287, Accuracy: 88.91%
Epoch: 3, Batch: 1500, Loss: 0.292, Accuracy: 88.87%
Epoch: 3, Batch: 1600, Loss: 0.274, Accuracy: 88.92%
Epoch: 3, Batch: 1700, Loss: 0.307, Accuracy: 88.92%
Epoch: 3, Batch: 1800, Loss: 0.297, Accuracy: 88.94%
Epoch: 4, Batch: 100, Loss: 0.270, Accuracy: 89.56%
Epoch: 4, Batch: 200, Loss: 0.263, Accuracy: 89.86%
```

```
Epoch: 4, Batch: 300, Loss: 0.306, Accuracy: 89.58%
Epoch: 4, Batch: 400, Loss: 0.286, Accuracy: 89.48%
Epoch: 4, Batch: 500, Loss: 0.287, Accuracy: 89.40%
Epoch: 4, Batch: 600, Loss: 0.290, Accuracy: 89.32%
Epoch: 4, Batch: 700, Loss: 0.274, Accuracy: 89.41%
Epoch: 4, Batch: 800, Loss: 0.268, Accuracy: 89.39%
Epoch: 4, Batch: 900, Loss: 0.278, Accuracy: 89.37%
Epoch: 4, Batch: 1000, Loss: 0.287, Accuracy: 89.39%
Epoch: 4, Batch: 1100, Loss: 0.288, Accuracy: 89.37%
Epoch: 4, Batch: 1200, Loss: 0.260, Accuracy: 89.42%
Epoch: 4, Batch: 1300, Loss: 0.299, Accuracy: 89.38%
Epoch: 4, Batch: 1400, Loss: 0.258, Accuracy: 89.46%
Epoch: 4, Batch: 1500, Loss: 0.286, Accuracy: 89.46%
Epoch: 4, Batch: 1600, Loss: 0.294, Accuracy: 89.46%
Epoch: 4, Batch: 1700, Loss: 0.288, Accuracy: 89.47%
Epoch: 4, Batch: 1800, Loss: 0.275, Accuracy: 89.48%
Epoch 5 completed
```

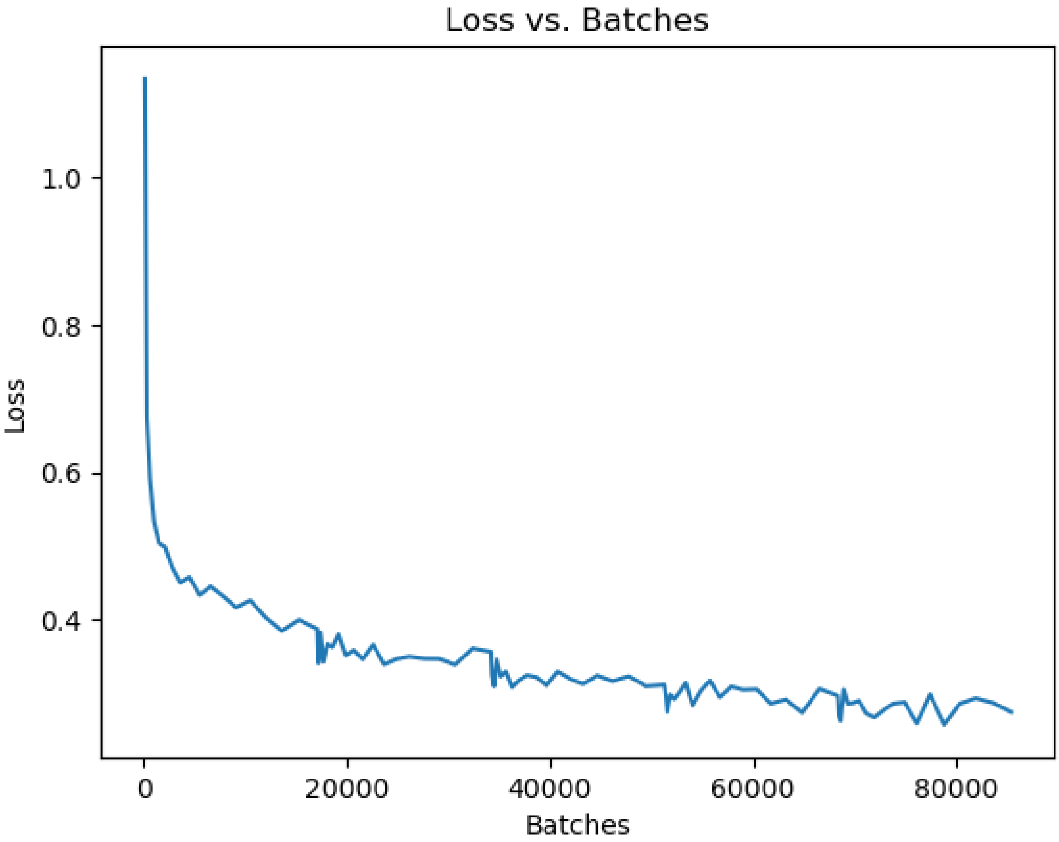
[160]:

```
evaluate_model(model3, validation_loader)
```

```
Test Accuracy: 87.76%
```

[161]:

```
plot_loss(losses, batches)
```



5.1 Notes of changes between (model1 and model2), (model1 and model3)

Changing the network depth and increasing the no. of perceptrons from model 1 to model 2 helped increase the test accuracy by 1.5 %, from 86% to 87.5%

While preserving the architecture of model 1 but changing the activation fn. to leaky_relu and optimizer to nesterov-Adam GD in model 3 helped increase the test accuracy from 86% to 87.76%