Fashion-MNIST prediction with neural networks

January 28, 2026

1 Classifying Fashion-MNIST

Now it's your turn to build and train a neural network. You'll be using the Fashion-MNIST dataset, a drop-in replacement for the MNIST dataset. MNIST is actually quite trivial with neural networks where you can easily achieve better than 97% accuracy. Fashion-MNIST is a set of 28x28 greyscale images of clothes. It's more complex than MNIST, so it's a better representation of the actual performance of your network, and a better representation of datasets you'll use in the real world.

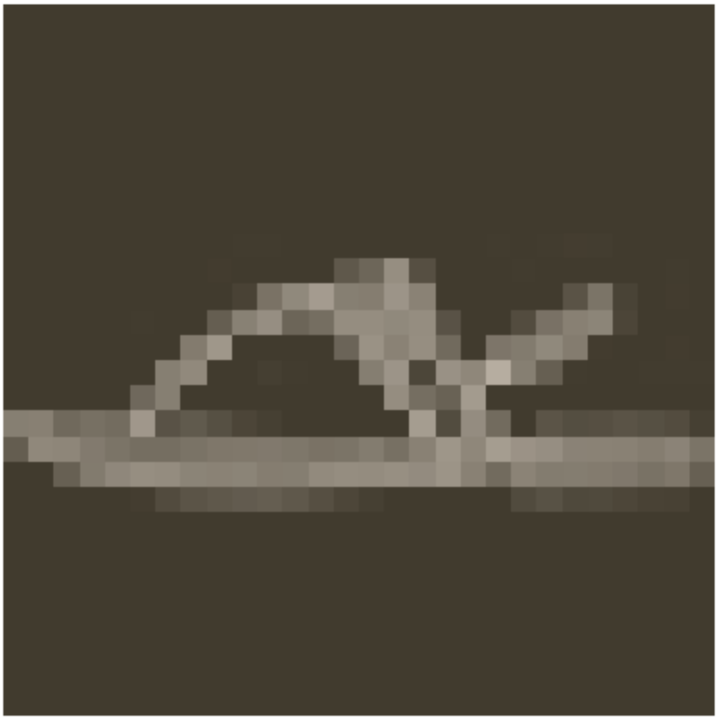
First off, let's load the dataset through torchvision.

[12]:
```
import torch
from torchvision import datasets, transforms
import helper

print(torch.__version__)
print("CUDA available:", torch.cuda.is_available())
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

# Define a transform to normalize the data
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5,), (0.5,))])
# Download and load the training data
trainset = datasets.FashionMNIST('~/.pytorch/F_MNIST_data/', download=True,␣
 ↪train=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)

# Download and load the test data
testset = datasets.FashionMNIST('~/.pytorch/F_MNIST_data/', download=True,␣
 ↪train=False, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=True)
```

```
2.4.0+cu124
CUDA available: True
Using device: cuda
```

Here we can see one of the images.

[13]:
```
image, label = next(iter(trainloader))
helper.imshow(image[0,:]);
```



1.1 Building the network

Here you should define your network. As with MNIST, each image is 28x28 which is a total of 784 pixels, and there are 10 classes. You should include at least one hidden layer. We suggest you use ReLU activations for the layers and to return the logits or log-softmax from the forward pass. It's up to you how many layers you add and the size of those layers.

[14]:
```
from torch import nn, optim
import torch.nn.functional as F
```

[15]:
```
# TODO: Define your network architecture here
class Classifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(784, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 64)
        self.fc4 = nn.Linear(64, 10)

    def forward(self, x):
```

```
        # make sure input tensor is flattened
        x = x.view(x.shape[0], -1)

        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.log_softmax(self.fc4(x), dim=1)

        return x
```

2 Train the network

Now you should create your network and train it. First you'll want to define the criterion (something like `nn.CrossEntropyLoss` or `nn.NLLLoss`) and the optimizer (typically `optim.SGD` or `optim.Adam`).

Then write the training code. Remember the training pass is a fairly straightforward process:

- Make a forward pass through the network to get the logits
- Use the logits to calculate the loss
- Perform a backward pass through the network with `loss.backward()` to calculate the gradients
- Take a step with the optimizer to update the weights

By adjusting the hyperparameters (hidden units, learning rate, etc), you should be able to get the training loss below 0.4.

[16]:
```
# TODO: Create the network, define the criterion and optimizer
model = Classifier()
criterion = nn.NLLLoss()
optimizer = optim.Adam(model.parameters(), lr=0.003)
```

[17]:
```
print(torch.cuda.is_available())
print(torch.cuda.get_device_name(0))
model.to(device)
print(next(model.parameters()).device)
```

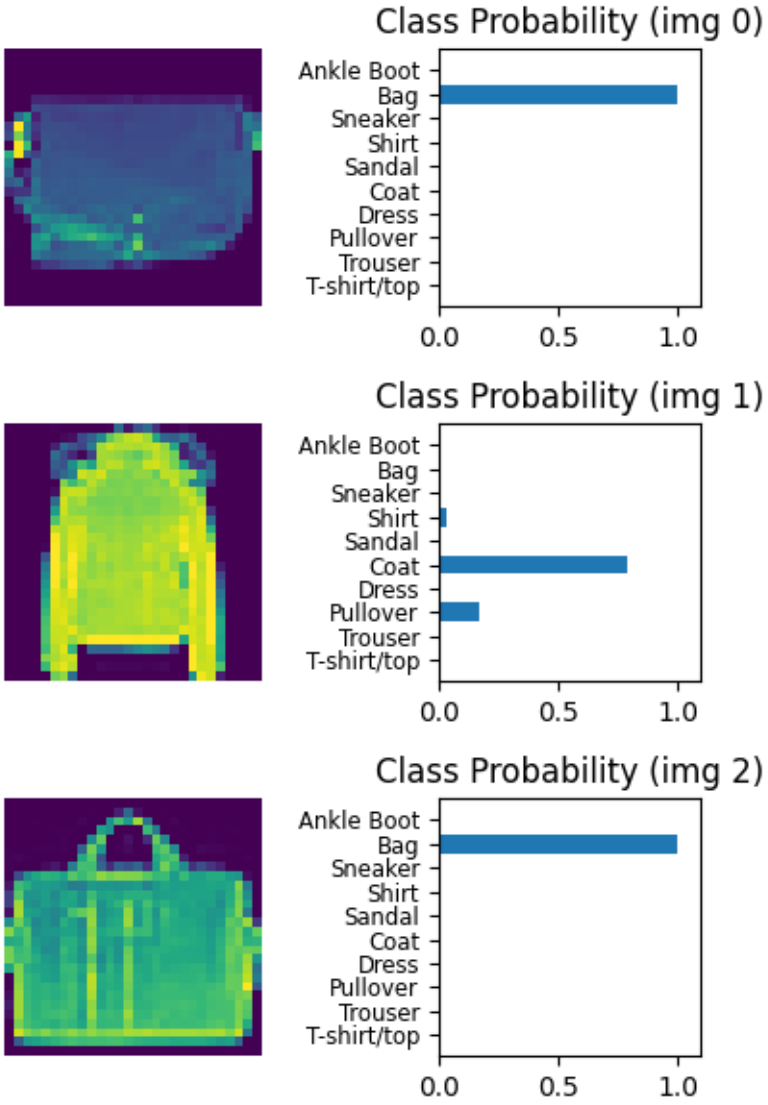
```
True
NVIDIA GeForce RTX 3050 Ti Laptop GPU
cuda:0
```

[18]:
```
# TODO: Train the network here
epochs = 5

for e in range(epochs):
    running_loss = 0
    for images, labels in trainloader:
        images = images.to(device)
        labels = labels.to(device)
```

```
        log_ps = model(images)
        loss = criterion(log_ps, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
    else:
        print(f"Training loss: {running_loss/len(trainloader)}")
```

```
Training loss: 0.5162554456989394
Training loss: 0.3902495345517771
Training loss: 0.35583574354235553
Training loss: 0.33392247879333586
Training loss: 0.3159903297657524
```

[22]:
```
import importlib
import helper
importlib.reload(helper)   # IMPORTANT după ce editezi helper.py

dataiter = iter(testloader)
images, labels = next(dataiter)

imgs = images[:3].to(device)  # <-- batch de 3 imagini

model.eval()
with torch.no_grad():
    ps = torch.exp(model(imgs))  # <-- batch output: [3, 10]

helper.view_classify_batch(imgs, ps, version='Fashion')
```


Class Probability (img 0)
Ankle Boot
Bag
Sneaker
Shirt
Sandal
Coat
Dress
Pullover
Trouser
T-shirt/top
0.0
0.5
1.0
Class Probability (img 1)
Ankle Boot
Bag
Sneaker
Shirt
Sandal
Coat
Dress
Pullover
Trouser
T-shirt/top
0.0
0.5
1.0
Class Probability (img 2)
Ankle Boot
Bag
Sneaker
Shirt
Sandal
Coat
Dress
Pullover
Trouser
T-shirt/top
0.0
0.5
1.0