**PRACTICAL: 3**

# a. Loading and Preprocessing the Image Data

**import** torch *# PyTorch package* **import** torchvision *# load datasets* **import** torchvision.transforms **as** transforms *# transform data* **import** torch.nn **as** nn *# basic building block for neural networks* **import** torch.nn.functional **as** F *# import convolution functions like ReLU* **import** torch.optim **as** optim *# optimizer* **import** numpy **as** np *# for numpy operations* **import** matplotlib.pyplot **as** plt *# for plotting* **import** time *# for timing the training process*

device = torch.device("cuda" **if** torch.cuda.is\_available() **else** "cpu")

transform = transforms.Compose([

transforms.ToTensor(), *# convert to tensor object*

transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)) *# mean = 0.5, std = 0.5* ])

**import** torchvision

**import** torchvision.transforms **as** transforms **import** torch

*# Set batch size and number of workers* batch\_size = 4 num\_workers = 2

*# Define a transform for the dataset* transform = transforms.Compose([ transforms.ToTensor(),

transforms.Normalize((0.5,), (0.5,)) *# Normalizing for grayscale images* ])

*# Load training data*

trainset = torchvision.datasets.FashionMNIST(root='./data', train=True,

*# Load test data*

testset = torchvision.datasets.FashionMNIST(root='./data', train=False, download=True, transform=transfor m)

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**def** imshow(img):

*'''Function to show image'''* img = img /

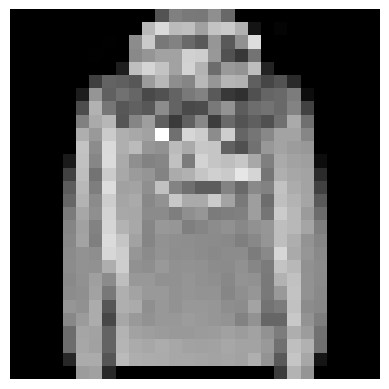
plt.imshow(npimg[0], cmap='gray') *# Display the single channel image* plt.axis('off') *# Hide axis* plt.show()

= iter(trainloader)

images, labels = next(dataiter)

*# Show the first image in the batch* imshow(images[0])

print(f'Label: {classes[labels[0]]}')



Label: Pullover

# b. Defining the Model’s Architecture

**import** torch **import** torch.nn **as** nn **import** torch.nn.functional **as** F **import** torchvision

*# Assuming you've already defined the trainloader and imshow function*

*# Iterate through the training data* dataiter = iter(trainloader) images, labels = next(dataiter)

*# Call function on our images*

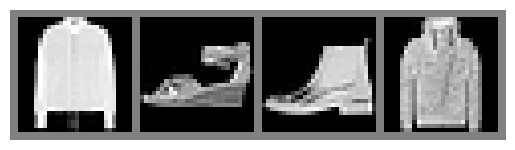
imshow(torchvision.utils.make\_grid(images))

*# Print the class of the images* print(' '.join('%s' % classes[labels[j]] **for** j **in** range(batch\_size)))

*'''Initialize the network'''* super(Net, self).\_\_init\_\_()

*\*

*# You can now proceed with the rest of your training code*



Coat Sandal Ankle boot Coat

# c. Training the Model

net = Net().to(device) *# Move the model to the selected device* print(net)

*# Loss and optimizer* criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

*# Timing the training process* start\_time = time.time()

**for** epoch **in** range(2): *# Loop over the dataset multiple times* running\_loss = 0.0 **for** i, data **in** enumerate(trainloader, 0):

*# Get the inputs; data is a list of [inputs, labels]* inputs, labels = data

# d. Estimating the Model’s Performance

end\_time = time.time() print('Finished Training')

print('Training time (seconds):', end\_time - start\_time)

*# Test the network* dataiter = iter(testloader)

images, labels = next(dataiter) *# Corrected this line*

*# Move test images to the selected device* images, labels = images.to(device), labels.to(device)

Finished Training

Training time (seconds): 113.67387127876282