**Department of Electrical Engineering and   
Computer Science**

**Faculty Member:** Dr. Mohsin Kamal **Dated:** 16/11/2023

**Semester:** 7th **Section:** BEE 12C

**CS-477 Computer Vision**

Lab 9: Implementation of a simple CNN on PyTorch

**Group Members**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **PLO4 - CLO4** | | **PLO5 -CLO5** | **PLO8 -CLO6** | **PLO9 -CLO7** |
| **Name** | **Reg. No** | **Viva / Quiz / Lab Performance** | **Analysis of Data in Lab Report** | **Modern Tool Usage** | **Ethics and Safety** | **Individual and Teamwork** |
|  |  | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** |
| Muhammad Ahmed Mohsin | 333060 |  |  |  |  |  |
| Muhammad Umer | 345834 |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Table of Contents**

[2 Implementation of a simple CNN on PyTorch 3](#_Toc151406099)

[2.1 Introduction 3](#_Toc151406100)

[2.2 Objectives 3](#_Toc151406101)

[2.3 Software 3](#_Toc151406102)

[3 Lab Tasks 4](#_Toc151406103)

[3.1 Task 1 4](#_Toc151406104)

[3.2 Task 2 9](#_Toc151406105)

[4 Conclusion 14](#_Toc151406106)

# Implementation of a simple CNN on PyTorch

## Introduction

This laboratory report aims to provide a comprehensive introduction to the world of convolutional neural networks (CNNs) using the powerful PyTorch framework. Through a series of hands-on exercises, we will delve into the intricacies of CNNs, exploring their unique architecture and learning capabilities.

## Objectives

The objective of this lab is:

* Understand PyTorch’s Tensor library and neural networks at a high level.
* Introduction CNN
* Training a CNN classifier

## Software

Convolutional Neural Network (CNN): A CNN is a type of deep neural network designed to recognize and process visual data with a grid-like structure, such as images. CNNs are particularly effective in image recognition, object detection, and other computer vision tasks.

CNNs use convolutional layers to learn spatial hierarchies of features automatically and adaptively from input images. These layers contain filters that are convolved with the input image to extract features such as edges, textures, and more complex patterns. Pooling layers are often used to reduce the spatial dimensions of the data, and fully connected layers integrate the learned features for final classification or regression tasks.

In summary, CNNs are a powerful class of neural networks designed for processing grid-like data, making them especially effective for tasks involving images and spatial relationships within those images.



# Lab Tasks

## Task 1

Build a simple convolutional neural network in PyTorch and train it to recognize handwritten digits using the MNIST dataset.

*Note: Training a classifier on the MNIST dataset can be regarded as the hello world of image recognition.*

### TASK CODE STARTS HERE ###

import numpy as np

import matplotlib.pyplot as plt

import torch

import torchvision

from torchvision.transforms import ToTensor, Normalize, Compose

from torch.utils.data import DataLoader

import torch.nn as nn

from tqdm import tqdm

from colorama import Fore, Style

*# Use STIX font for matplotlib*

plt.rcParams["font.family"] = "STIXGeneral"

epochs = 3

batch\_size = 128

lr = 0.001

log\_interval = 100

random\_seed = 42

torch.manual\_seed(random\_seed)

*# Load the MNIST dataset*

train\_loader = DataLoader(

    torchvision.datasets.MNIST(

        "./",

*train*=True,

*download*=True,

*transform*=Compose([ToTensor(), Normalize((0.1307,), (0.3081,))]),

    ),

*batch\_size*=batch\_size,

*shuffle*=True,

)

test\_loader = DataLoader(

    torchvision.datasets.MNIST(

        "./",

*train*=False,

*download*=True,

*transform*=Compose([ToTensor(), Normalize((0.1307,), (0.3081,))]),

    ),

*batch\_size*=1000,

*shuffle*=True,

)

*# Visualize the dataset*

examples = enumerate(test\_loader)

batch\_idx, (example\_data, example\_targets) = next(examples)

fig = plt.figure()

for i in range(6):

    plt.subplot(2, 3, i + 1)

    plt.tight\_layout()

    plt.imshow(example\_data[i][0], *cmap*="gray", *interpolation*="none")

    plt.title("Ground Truth: {}".format(example\_targets[i]))

    plt.xticks([])

    plt.yticks([])

*# Define the model*

*class* CNN(*nn*.*Module*):

*def* \_\_init\_\_(*self*):

*super*(CNN, *self*).\_\_init\_\_()

*self*.conv1 = nn.Conv2d(*in\_channels*=1, *out\_channels*=32, *kernel\_size*=3)

*self*.conv2 = nn.Conv2d(*in\_channels*=32, *out\_channels*=64, *kernel\_size*=3)

*self*.conv2\_drop = nn.Dropout2d()

*self*.fc1 = nn.Linear(*in\_features*=1600, *out\_features*=128)

*self*.fc2 = nn.Linear(*in\_features*=128, *out\_features*=10)

*self*.relu = nn.ReLU()

*self*.max\_pool = nn.MaxPool2d(*kernel\_size*=2)

*def* forward(*self*, *x*):

        x = *self*.relu(*self*.max\_pool(*self*.conv1(x)))

        x = *self*.relu(*self*.max\_pool(*self*.conv2\_drop(*self*.conv2(x))))

        x = x.view(-1, 1600)

        x = *self*.relu(*self*.fc1(x))

        x = *self*.fc2(x)

        return x

*# Define the loss function and optimizer*

model = CNN()

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model.parameters(), *lr*=lr)

*# train the model and evaluate it on the test set*

for i in range(epochs):

    print(*f*"{Fore.CYAN}Epoch: {i + 1}/{epochs}{Style.RESET\_ALL}")

    for batch\_idx, (data, targets) in enumerate(train\_loader):

        optimizer.zero\_grad()

        scores = model(data)

        loss = criterion(scores, targets)

        loss.backward()

        optimizer.step()

        if batch\_idx % log\_interval == 0 or batch\_idx == len(train\_loader) - 1:

            print(

*f*">> Batch: {batch\_idx + 1}/{len(train\_loader)}\tLoss: {loss.item()*:.6f*}"

            )

            if batch\_idx == len(train\_loader) - 1:

                print()

*# Evaluate the model*

model.eval()

test\_loss = 0

correct = 0

total = 0

with torch.no\_grad():

    for data, targets in test\_loader:

        scores = model(data)

        loss = criterion(scores, targets)

        test\_loss += loss.item()

        \_, predicted = scores.max(1)

        total += targets.size(0)

        correct += predicted.eq(targets).sum().item()

    print(*f*"{Fore.GREEN}Test Loss: {test\_loss / len(test\_loader)*:.6f*}{Style.RESET\_ALL}")

    print(*f*"{Fore.GREEN}Test Accuracy: {correct / total \* 100*:.2f*}%{Style.RESET\_ALL}")

*# Visualize the predictions*

examples = enumerate(test\_loader)

batch\_idx, (example\_data, example\_targets) = next(examples)

with torch.no\_grad():

    scores = model(example\_data)

    \_, predicted = scores.max(1)

fig = plt.figure()

for i in range(6):

    plt.subplot(2, 3, i + 1)

    plt.tight\_layout()

    plt.imshow(example\_data[i][0], *cmap*="gray", *interpolation*="none")

    plt.title(

*f*"Predicted: {predicted[i]}, Actual: {example\_targets[i]}"

    )

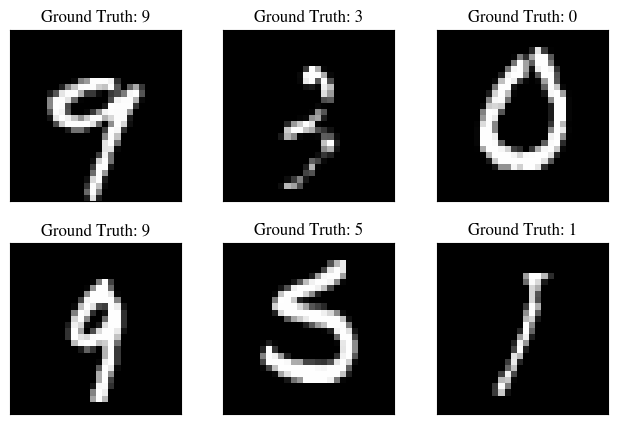
    plt.xticks([])

    plt.yticks([])

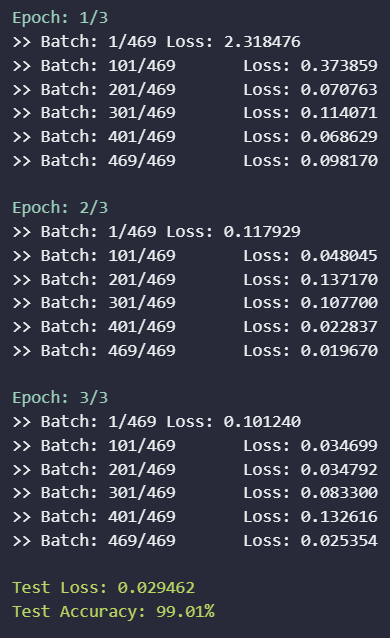
### TASK CODE ENDS HERE ###

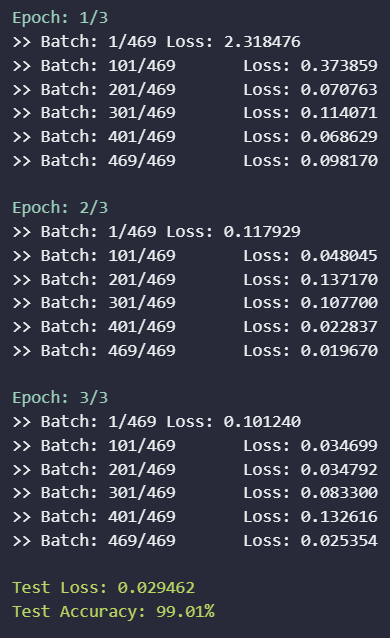
### TASK SCREENSHOT STARTS HERE ###

Dataset Visualization

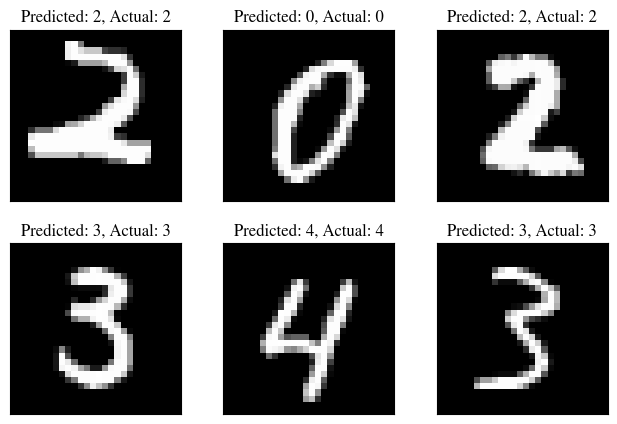


Training Logs





Predictions



### TASK SCREENSHOT ENDS HERE ###

### TASK Description

In this initial task, we ventured into the realm of convolutional neural networks (CNNs) by constructing a simple CNN architecture using PyTorch and employing it to classify handwritten digits from the MNIST dataset. This introductory exercise served as a stepping stone into the world of image recognition, providing a hands-on experience with the fundamental concepts and implementation of CNNs.

## Task 2

Build a simple convolutional neural network in PyTorch and train it to recognize the following fashion object using the fashion MNIST dataset.

10 classes (T-shirt, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot)

### TASK CODE STARTS HERE ###

import numpy as np

import matplotlib.pyplot as plt

import torch

import torchvision

from torchvision.transforms import ToTensor, Normalize, Compose

from torch.utils.data import DataLoader

import torch.nn as nn

from tqdm import tqdm

from colorama import Fore, Style

*# Use STIX font for matplotlib*

plt.rcParams["font.family"] = "STIXGeneral"

epochs = 10

batch\_size = 128

lr = 0.001

log\_interval = 100

random\_seed = 42

torch.manual\_seed(random\_seed)

class\_mapping = {

    0: "T-shirt/top",    1: "Trouser",    2: "Pullover",    3: "Dress",

    4: "Coat",    5: "Sandal",    6: "Shirt",    7: "Sneaker",

    8: "Bag",    9: "Ankle boot"}

*# Load the MNIST dataset*

train\_loader = DataLoader(

    torchvision.datasets.FashionMNIST(

        "./",

*train*=True,

*download*=True,

*transform*=Compose([ToTensor(), Normalize((0.1307,), (0.3081,))]),

    ),

*batch\_size*=batch\_size,

*shuffle*=True,

)

test\_loader = DataLoader(

    torchvision.datasets.FashionMNIST(

        "./",

*train*=False,

*download*=True,

*transform*=Compose([ToTensor(), Normalize((0.1307,), (0.3081,))]),

    ),

*batch\_size*=1000,

*shuffle*=True,

)

*# Visualize the dataset*

examples = enumerate(test\_loader)

batch\_idx, (example\_data, example\_targets) = next(examples)

fig = plt.figure()

for i in range(6):

    plt.subplot(2, 3, i + 1)

    plt.tight\_layout()

    plt.imshow(example\_data[i][0], *cmap*="gray", *interpolation*="none")

    plt.title("Ground Truth: {}".format(class\_mapping[*int*(example\_targets[i])]))

    plt.xticks([])

    plt.yticks([])

*# Define the model*

*class* CNN(*nn*.*Module*):

*def* \_\_init\_\_(*self*):

*super*(CNN, *self*).\_\_init\_\_()

*self*.conv1 = nn.Conv2d(*in\_channels*=1, *out\_channels*=32, *kernel\_size*=3)

*self*.conv2 = nn.Conv2d(*in\_channels*=32, *out\_channels*=64, *kernel\_size*=3)

*self*.conv2\_drop = nn.Dropout2d()

*self*.fc1 = nn.Linear(*in\_features*=1600, *out\_features*=128)

*self*.fc2 = nn.Linear(*in\_features*=128, *out\_features*=10)

*self*.relu = nn.ReLU()

*self*.max\_pool = nn.MaxPool2d(*kernel\_size*=2)

*def* forward(*self*, *x*):

        x = *self*.relu(*self*.max\_pool(*self*.conv1(x)))

        x = *self*.relu(*self*.max\_pool(*self*.conv2\_drop(*self*.conv2(x))))

        x = x.view(-1, 1600)

        x = *self*.relu(*self*.fc1(x))

        x = *self*.fc2(x)

        return x

*# Define the loss function and optimizer*

model = CNN()

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model.parameters(), *lr*=lr)

*# train the model and evaluate it on the test set*

for i in range(epochs):

    print(*f*"{Fore.CYAN}Epoch: {i + 1}/{epochs}{Style.RESET\_ALL}")

    for batch\_idx, (data, targets) in enumerate(train\_loader):

        optimizer.zero\_grad()

        scores = model(data)

        loss = criterion(scores, targets)

        loss.backward()

        optimizer.step()

        if batch\_idx % log\_interval == 0 or batch\_idx == len(train\_loader) - 1:

            print(

*f*">> Batch: {batch\_idx + 1}/{len(train\_loader)}\tLoss: {loss.item()*:.6f*}"

            )

            if batch\_idx == len(train\_loader) - 1:

                print()

*# Evaluate the model*

model.eval()

test\_loss = 0

correct = 0

total = 0

with torch.no\_grad():

    for data, targets in test\_loader:

        scores = model(data)

        loss = criterion(scores, targets)

        test\_loss += loss.item()

        \_, predicted = scores.max(1)

        total += targets.size(0)

        correct += predicted.eq(targets).sum().item()

    print(*f*"{Fore.GREEN}Test Loss: {test\_loss / len(test\_loader)*:.6f*}{Style.RESET\_ALL}")

    print(*f*"{Fore.GREEN}Test Accuracy: {correct / total \* 100*:.2f*}%{Style.RESET\_ALL}")

*# Visualize the predictions*

examples = enumerate(test\_loader)

batch\_idx, (example\_data, example\_targets) = next(examples)

with torch.no\_grad():

    scores = model(example\_data)

    \_, predicted = scores.max(1)

fig = plt.figure()

for i in range(4):

    plt.subplot(2, 2, i + 1)

    plt.tight\_layout()

    plt.imshow(example\_data[i][0], *cmap*="gray", *interpolation*="none")

    plt.title(

*f*"Predicted: {class\_mapping[*int*(predicted[i])]}, \

        Actual: {class\_mapping[*int*(example\_targets[i])]}"

    )

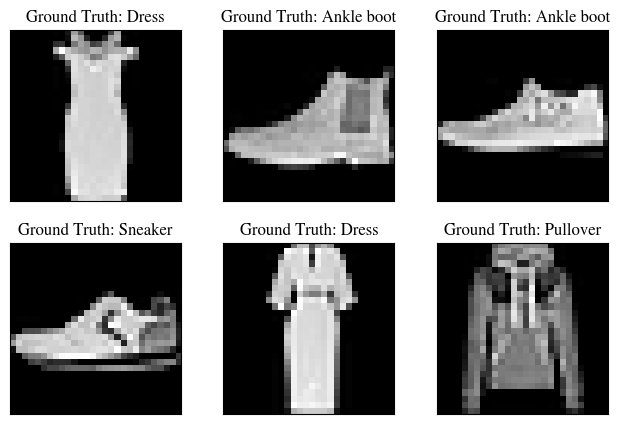
    plt.xticks([])

    plt.yticks([])

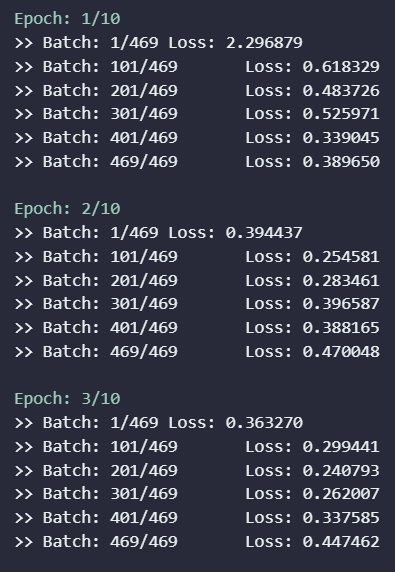
### TASK CODE ENDS HERE ###

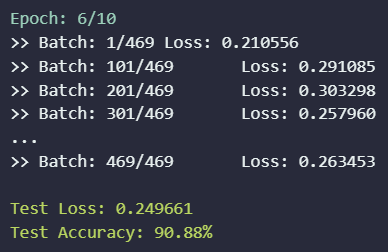
### TASK SCREENSHOT STARTS HERE ###

Dataset Visualization

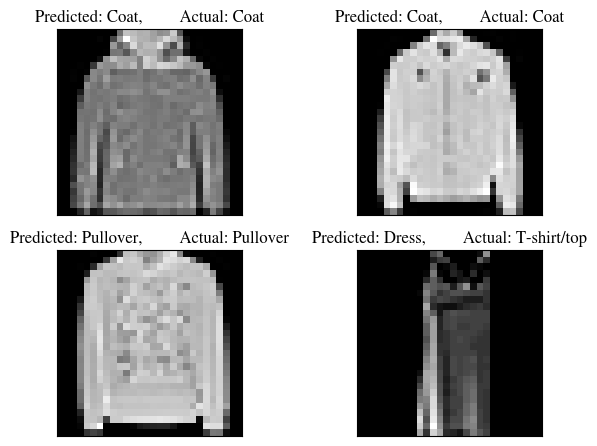


Training Logs





Predictions



### TASK SCREENSHOT ENDS HERE ###

### TASK Description

Building upon our foundation in CNNs from Task 1, we delved into the realm of fashion object recognition by constructing a CNN architecture and training it to identify ten distinct fashion items using the Fashion-MNIST dataset. This task expanded our understanding of CNNs and their applicability in classifying more complex visual patterns.

# Conclusion

In conclusion, our journey through the realm of convolutional neural networks (CNNs) culminated in the successful development and training of CNN classifiers for two distinct image classification tasks: handwritten digit recognition and fashion object recognition. Through these practical exercises, we gained a comprehensive understanding of CNNs, their architectural design, and their application in solving real-world image classification problems. Furthermore, we gained proficiency in utilizing PyTorch, a powerful deep learning framework, to construct, train, and evaluate CNN models.