**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **ІКНІ**

Кафедра **СШІ**

**ЗВІТ**

До лабораторної роботи №3

**З дисципліни:**

**«Машинне навчання»**

**Виконав:**

ст. гр. КН-310

Шиманський П.С.

**Прийняла:**

Якимишин Х.М.

Львів - 2020

**Мета: ​** засвоїти основні відомості про роботу з згортковими нейронними мережами. Навчитись використовувати їх для класифікації зображень.

Варіант 6

**Завдання:** ​

Реалізувати згорткову нейронну мережу використовуючи PyTorch та натренувати її на заданому наборі даних, а саме:

1. Розділити дані на тренувальні(60 000) і тестові(10 000)

2. Реалізувати згорткову нейронну мережу на PyTorch.

3. Реалізувати тренувальний цикл.

4. Реалізувати тестувальний цикл.

5. Натренувати мережу до заданої точності.

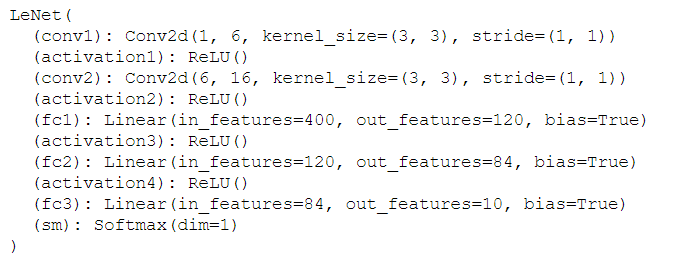
6. Звізуалізувати найгірші 10 передбачень, порівняти їх з вдалими передбаченнями цього ж класу. Спробувати пояснити причини поганих передбачень.





**Хід роботи:**

1. За допомогою PyTorch створив відповідну нейронну мережу

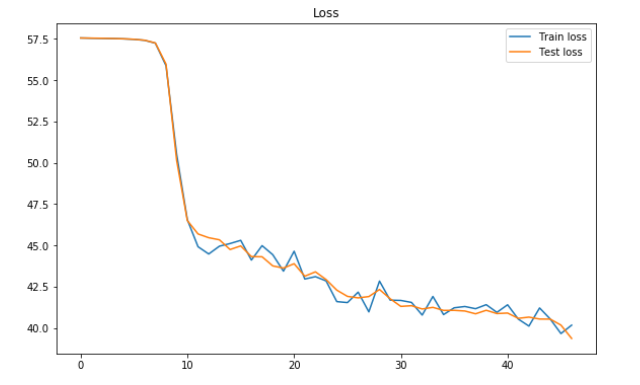
****

**Рис. 1.** Результат створення нейронної мережі

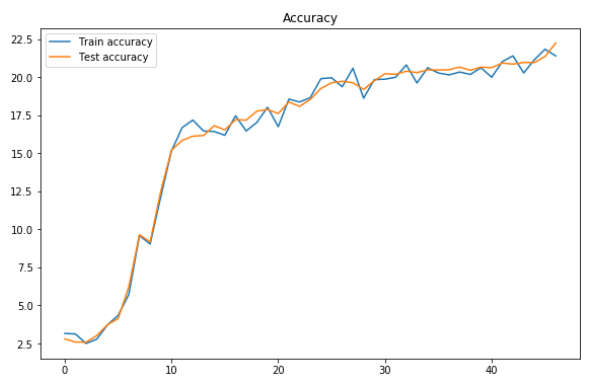
1. За функцію втрат було взято крос-ентропію:

L(x, class)= -x[class] + log(

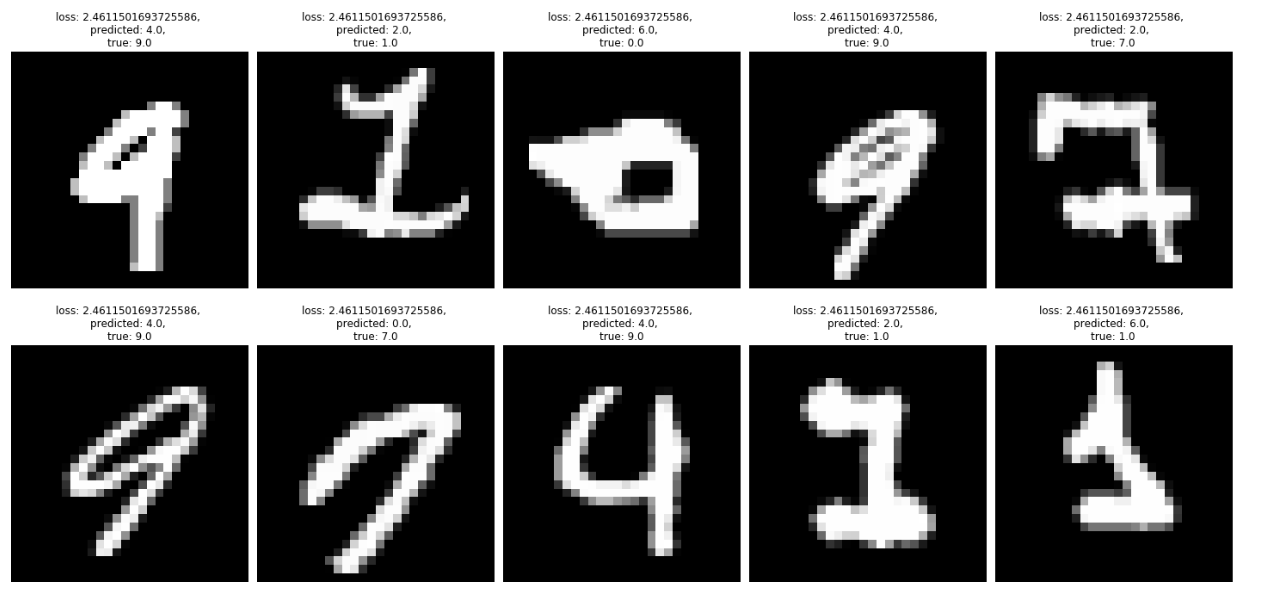
1. За умовою завдання, точність – 90%, нейронна мережа тренерувалася завдяки стохастичному градієнтному спуску з начальним кроком 0.001 та розміром батча 32. Тренування було закінчено на 1200 ітерацій.



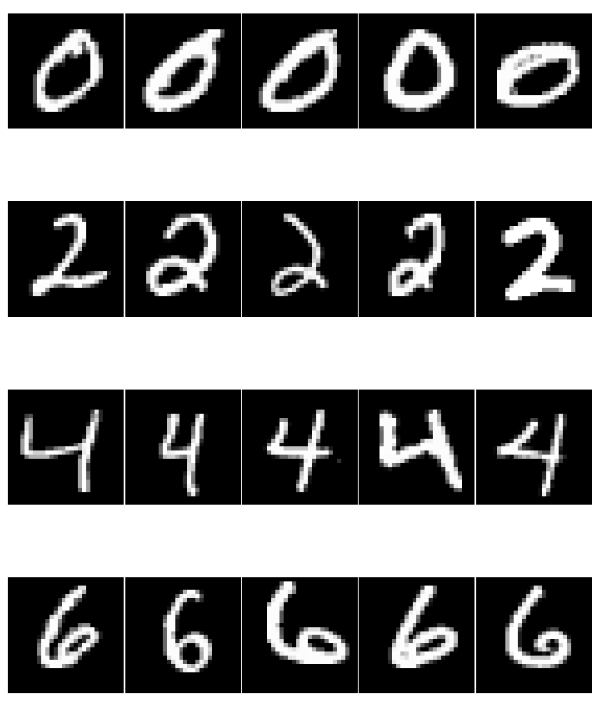
**Рис. 2.** Результати ходу тренувань



**Рис. 3.** Результати ходу тренувань



**Рис. 4.** Зображення з найгіршими передбаченнями



**Рис. 5.** Зображення найкращих передбачень

Отже, судячи по фотографіях можна зрозуміти, що у деяких випадках правильне передбачення класу певних об’єктів без перетренування під них є неможливим.

Проте їхню якість все ще можна покращити, а саме якщо продовжити тренування, оскільки є випадки, де проведення коректної класифікації є можливим.

**Висновок:**

На лабораторній роботі я навчився використовувати основні відомості про роботу із згортковими нейронними мережами. Також навчився застосовувати їх для класифікації зображень.

**Код програми:**

import torch

import torch.nn as nn

import torch.optim as optim

import torch.nn.functional as F

import matplotlib.pyplot as plt

import torchvision

import numpy as np

from IPython.display import clear\_output

from collections import defaultdict

class LeNet(nn.Module):

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

super(LeNet, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(1, 6, 3)

self.activation1 = nn.ReLU(inplace=False)

self.conv2 = nn.Conv2d(6, 16, 3)

self.activation2 = nn.ReLU(inplace=False)

self.fc1 = nn.Linear(5\*5\*16, 120)

self.activation3 = nn.ReLU(inplace=False)

self.fc2 = nn.Linear(120, 84)

self.activation4 = nn.ReLU(inplace=False)

self.fc3 = nn.Linear(84, 10)

self.sm = nn.Softmax(1)

def forward(self, x):

x = F.avg\_pool2d(self.activation1(self.conv1(x)), (2, 2))

x = F.avg\_pool2d(self.activation2(self.conv2(x)), 2)

x = x.view(-1, self.num\_flat\_features(x))

x = self.activation3(self.fc1(x))

x = self.activation4(self.fc2(x))

x = self.fc3(x)

x = self.sm(x)

return x

def num\_flat\_features(self, x): # функція, яка повертає числа особливостей шару згортки

size = x.size()[1:]

num\_features = 1

for s in size:

num\_features \*= s

return num\_features

cuda = torch.device('cuda')

net = LeNet().to(cuda) # створює модель

print(net)

criterion = nn.CrossEntropyLoss() # створює функцію втрат

optimizer = optim.SGD(net.parameters(), lr=0.08, momentum=0.7) # створює оптимайзер

epochs = 1000

train\_loader = torch.utils.data.DataLoader( # скачуємо дані

torchvision.datasets.MNIST('./data/mnist', train=True, download=True,

transform=torchvision.transforms.Compose([

torchvision.transforms.Resize((28, 28), interpolation=2), # зміна розміру зображень орінгалу у формі (32, 32), коли модель має вхід (28, 28)

torchvision.transforms.ToTensor(), # перетворення зображення з нумерованого масиву в тензор

torchvision.transforms.Normalize( # нормалізація зображень

(0.1307,), (0.3081,))

])),

batch\_size=32, shuffle=False)

test\_loader = torch.utils.data.DataLoader(

torchvision.datasets.MNIST('./data/mnist', train=False,

transform=torchvision.transforms.Compose([

torchvision.transforms.Resize((28, 28), interpolation=2),

torchvision.transforms.ToTensor(),

torchvision.transforms.Normalize(

(0.1307,), (0.3081,))

])),

batch\_size=10000, shuffle=False)

test\_inputs, test\_labels = iter(test\_loader).next() # отримуємо всі дані

test\_inputs = test\_inputs.to(cuda)

test\_labels = test\_labels.to(cuda)

%%time

finished = False # змінна, яка представляє, чи досягнута модель визначеної точності

# списки, в яких зберігається історія навчання

train\_loss\_history = []

test\_loss\_history = []

train\_accuracy\_history = []

test\_accuracy\_history = []

for epoch in range(epochs):

running\_train\_loss = 0.0

running\_train\_accuracy = 0.0

running\_test\_loss = 0.0

running\_test\_accuracy = 0.0

if not finished:

for i, data in enumerate(train\_loader, 0):

inputs, labels = data # отримати входи; дані - це список [входів, міток]

inputs = inputs.to(cuda)

labels = labels.to(cuda)

optimizer.zero\_grad() # нуль градієнтів параметра

outputs = net(inputs)

test\_outputs = net(test\_inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

test\_loss = criterion(test\_outputs, test\_labels) # розрахунок тестових втрат

# точність обчислення

train\_accuracy = (outputs.argmax(1) == labels).sum().item() / outputs.shape[0]

test\_accuracy = (test\_outputs.argmax(1) == test\_labels).sum().item() / test\_outputs.shape[0]

running\_train\_accuracy += train\_accuracy

running\_train\_loss += loss.item()

running\_test\_accuracy += test\_accuracy

running\_test\_loss += test\_loss.item()

# вивести статистику

if i % 25 == 24: # друкувати кожні 25 міні-партій

print('[%d, %5d] \nTrain: \n\tloss: %.3f, accuracy: %.3f\nTest: \n\tloss: %.3f, accuracy: %.3f' %

(epoch + 1, i + 1, running\_train\_loss / 25, running\_train\_accuracy / 25,

running\_test\_loss / 25, running\_test\_accuracy / 25))

if running\_test\_accuracy / 25 > 0.90 and running\_train\_accuracy / 25 > 0.90: # перевірити, чи досягли ми встановленої точності

finished = True

break

# додавання статистики до логів

train\_loss\_history.append(running\_train\_loss)

train\_accuracy\_history.append(running\_train\_accuracy)

test\_loss\_history.append(running\_test\_loss)

test\_accuracy\_history.append(running\_test\_accuracy)

running\_train\_loss = 0.0

running\_train\_accuracy = 0.0

running\_test\_loss = 0.0

running\_test\_accuracy = 0.0

print('Finished Training')

outputs[:5]

(outputs.argmax(1) == labels).sum().item() / outputs.shape[0]

f, ax = plt.subplots(1, 2, figsize=(21, 6))

# n = len(train\_loss\_history)

ax[0].plot(train\_loss\_history, label='Train loss')

ax[0].plot(test\_loss\_history, label='Test loss')

ax[0].set\_title('Loss')

ax[0].legend()

ax[1].plot(train\_accuracy\_history, label='Train accuracy')

ax[1].plot(test\_accuracy\_history, label='Test accuracy')

ax[1].set\_title('Accuracy')

ax[1].legend()

f.savefig('res.png')

# створення іншого завантажувача даних розміром партії 1, щоб отримати втрати окремо для кожного елемента

train\_loader = torch.utils.data.DataLoader(

torchvision.datasets.MNIST('./data/mnist', train=True,

transform=torchvision.transforms.Compose([

torchvision.transforms.Resize((28, 28), interpolation=2),

torchvision.transforms.ToTensor(),

torchvision.transforms.Normalize(

(0.1307,), (0.3081,))

])),

batch\_size=1, shuffle=False)

test\_loader = torch.utils.data.DataLoader(

torchvision.datasets.MNIST('./data/mnist', train=False,

transform=torchvision.transforms.Compose([

torchvision.transforms.Resize((28, 28), interpolation=2),

torchvision.transforms.ToTensor(),

torchvision.transforms.Normalize(

(0.1307,), (0.3081,))

])),

batch\_size=1, shuffle=False)

res = np.zeros((70000, 4)) # тут буде зберігатися (індекс зображення, втрата зображення, передбачувана мітка, справжня мітка)

for i, item in enumerate(train\_loader, 0):

img, labels\_true = item

img = img.to(cuda)

labels\_true = labels\_true.to(cuda)

labels\_pred = net(img)

loss = criterion(labels\_pred, labels\_true)

res[i] = torch.tensor([i, loss.item(), labels\_pred.argmax(1).item(), labels\_true.item()])

for i, item in enumerate(test\_loader, 60000):

img, labels\_true = item

img = img.to(cuda)

labels\_true = labels\_true.to(cuda)

labels\_pred = net(img)

loss = criterion(labels\_pred, labels\_true)

res[i] = torch.tensor([i, loss.item(), labels\_pred.argmax().item(), labels\_true.item()])

indxs\_worst = res[:, 1].argsort()[-10:][::-1] # індекси 10 зображень з найбільшою втратою

worst = res[np.sort(indxs\_worst)]

unique = np.unique(worst[:, 2]) # всі унікальні класи в найгірших образах

# тут ми отримуємо 5 знімків із найменшими втратами в класах, де було одне чи більше зображень із найбільшими втратами

# і зберігати індекси зображень у словнику d - {index: class}

d = defaultdict(int)

for class\_ in unique:

class\_res = res[res[:, 2] == class\_]

class\_indxs = class\_res[:, 1].argsort()[:5]

indxs = class\_res[class\_indxs][:, 0]

for indx in indxs:

d[indx] = class\_

imgs\_worst = [] # Зображення з найбільшою втратою зберігатимуться тут

imgs\_best = defaultdict(list) # зберігає зображення з найменшими втратами для конкретного класу - {class: список із 5 зображень}

# У наступних рядках ми повторюємо набір даних та зберігаємо відповідні зображення

for i, item in enumerate(train\_loader, 0):

if i in indxs\_worst:

imgs\_worst.append(item[0])

if i in d.keys():

imgs\_best[d[i]].append(item[0])

for i, item in enumerate(test\_loader, 60000):

if i in indxs\_worst:

imgs\_worst.append(item[0])

if i in d.keys():

imgs\_best[d[i]].append(item[0])

# візуалізація зображень

f, ax = plt.subplots(2, 5, figsize=(20, 10))

for i in range(10):

ax[i//5, i%5].imshow(imgs\_worst[i].reshape((28, 28)), cmap='gray')

ax[i//5, i%5].axis('off')

ax[i//5, i%5].set\_title('loss: {}, \npredicted: {}, \ntrue: {}'.format(\*worst[i, 1:]))

plt.tight\_layout()

f.savefig('worst.png')

f, ax = plt.subplots(unique.shape[0], 5, figsize=(11.25, 15))

for i, class\_ in enumerate(unique):

for j, img in enumerate(imgs\_best[class\_]):

ax[i, j].imshow(img.reshape((28, 28)), cmap='gray')

ax[i, j].axis('off')

plt.tight\_layout(pad=5.5, w\_pad=0.2, h\_pad=0.1)

fig = plt.gcf();

fig.suptitle("Best examples in classes", fontsize=28);

f.savefig('best.png')