ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное бюджетное образовательное учреждение высшего образования "Санкт-Петербургский государственный университет телекоммуникаций им. проф. М.А. Бонч-Бруевича"

Факультет «Инфокоммуникационных сетей и систем»

Кафедра «Программной инженерии и вычислительной техники»

Отчет по лабораторной работе №4

**Тема: «Нейронные сети для распознавания образов»**

*Дисциплина: «Системы искусственных интеллектов»*

Выполнил студент гр. ИКПИ-84

Василец Павел

Приняла: Белая Т.И.

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Санкт-Петербург**

**2021 г.**

**Задание:**

1. Изучить устройство и работу однослойной, двухслойной нейронных сетей, сети встречного распространения.
2. Выбрать исходный алфавит классов (цифры, буквы русского алфавита, буквы английского алфавита, буквы греческого алфавита, знаки препинания и арифметические символы, радиоэлементы, геометрические фигуры, узоры, дорожные знаки, топологические элементы, картографические обозначения) по согласованию с преподавателем.
3. Составить программу, реализующую однослойную, двухслойную нейронные сети и сеть встречного распространения для задачи распознавания. Обучить нейронные сети.
4. Программное обеспечение должно позволять просматривать эталоны (классы) изображений, а также распознаваемые изображения; записывать эталоны изображений в библиотеку на диск, записывать входной образ на диск; позволять редактировать входной образ, сохранять веса нейронов при обучении сети.
5. Испытать программное обеспечение для различных входных данных.
6. Произвести оценку качества распознавания для различных случаев, систематизировав полученные результаты в таблицы. Построить графики выявленных зависимостей, сделать выводы.
7. Результаты работы оформить в виде отчета в текстовом редакторе.

**Ход работы:**

Выбранным исходным алфавитом классов являются прописные цифры. В качестве обучающих данных был выбран датасет MNIST, состоящий из 60 000 изображений для обучения и 10 000 изображений для тестирования.



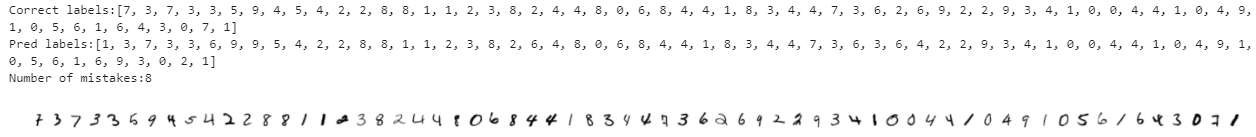
*Рис 1. Иллюстрация датасета MNIST*

Реализуемые нейронные сети:

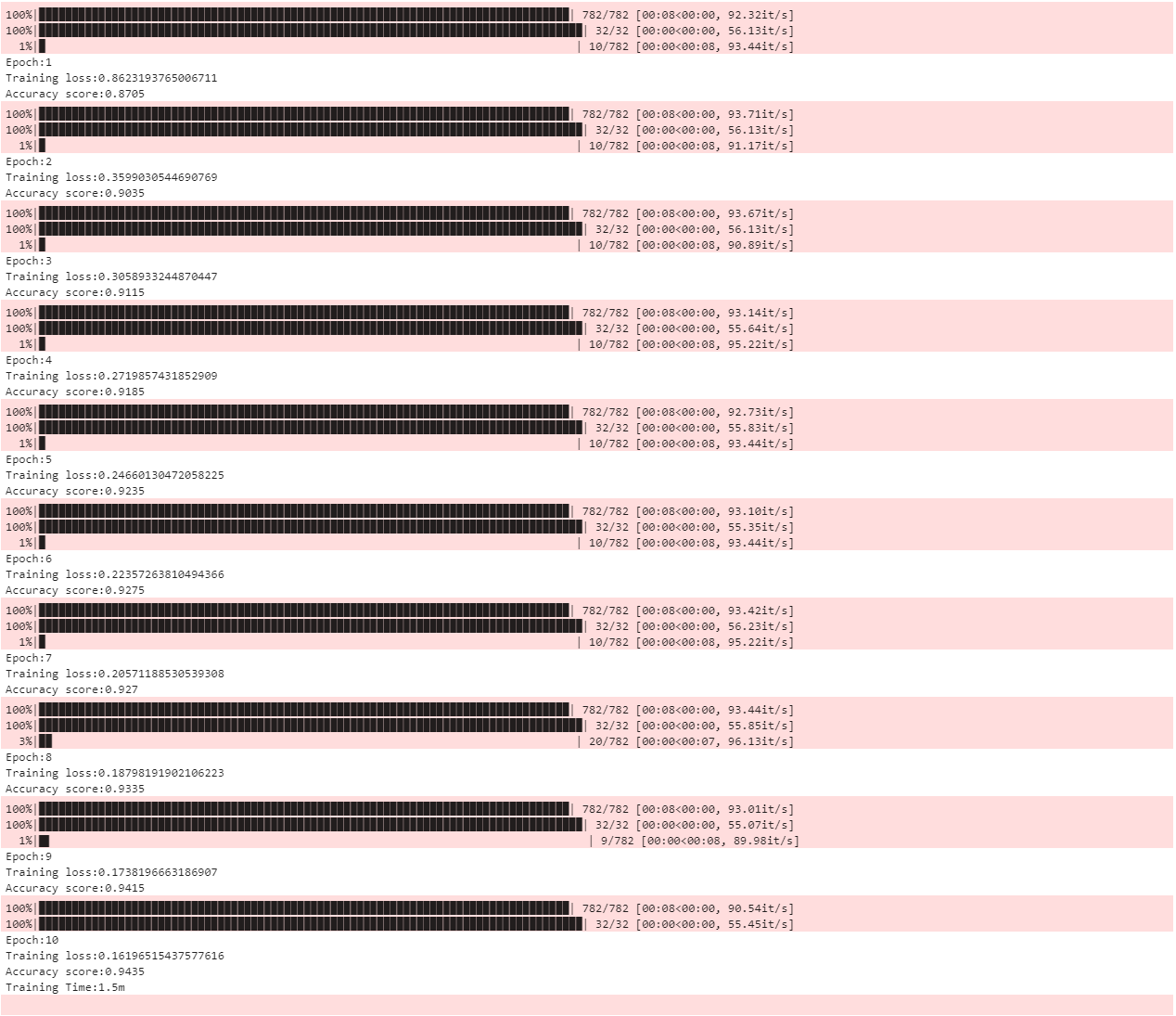
* Линейный классификатор (однослойный перцептрон);
* 5-уровневая нейронная сеть;
* Сверточная нейронная сеть;



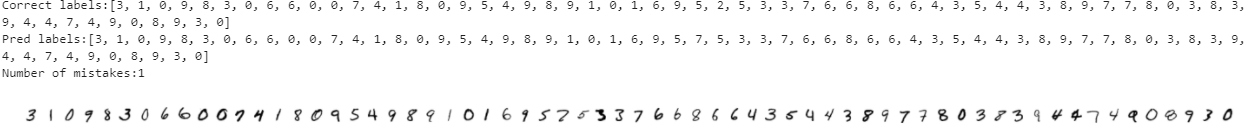
*Рис. 2 – Процесс обучения линейного классификатора*

**

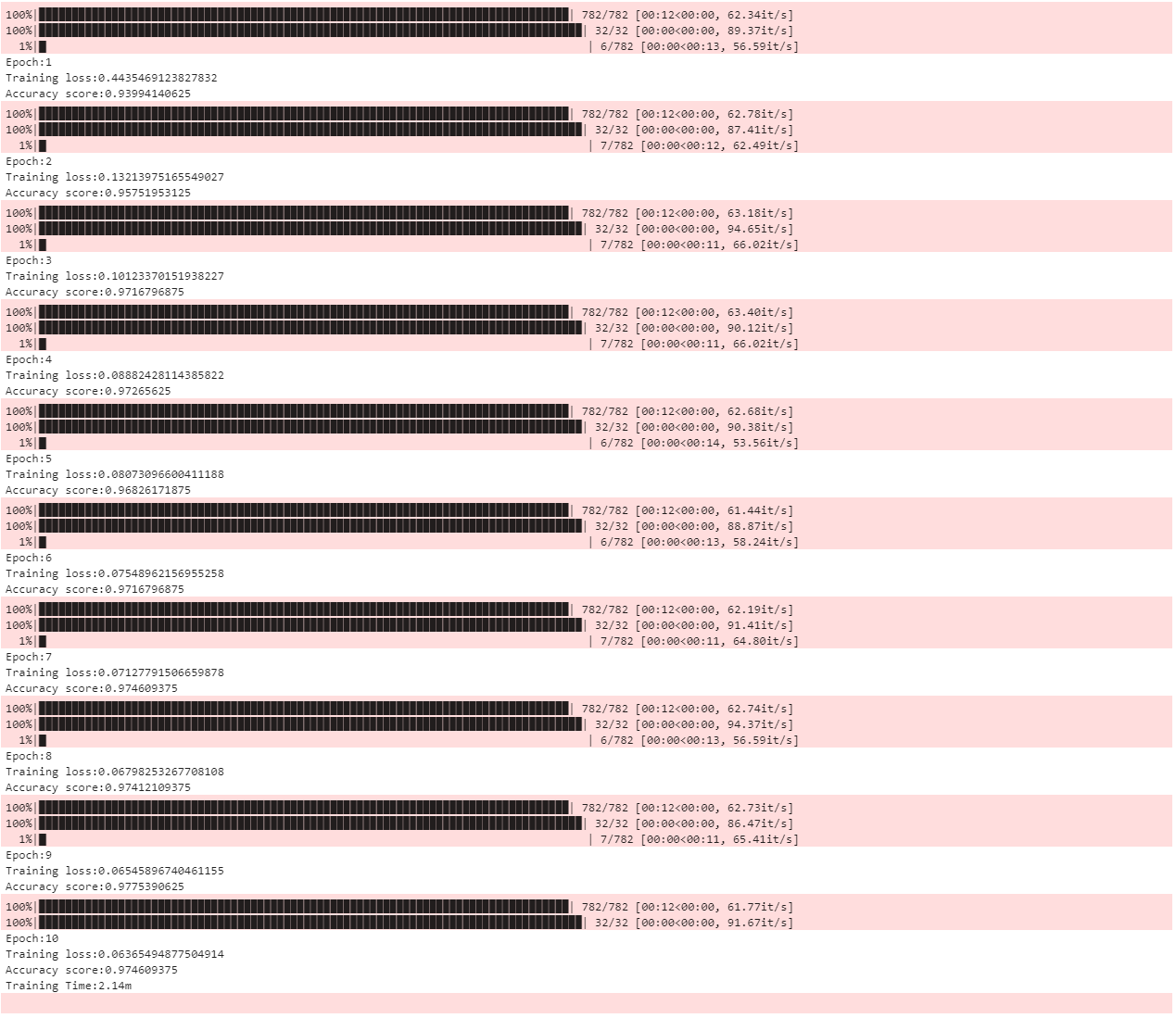
*Рис. 3 – Тестирование линейного классификатора*

**

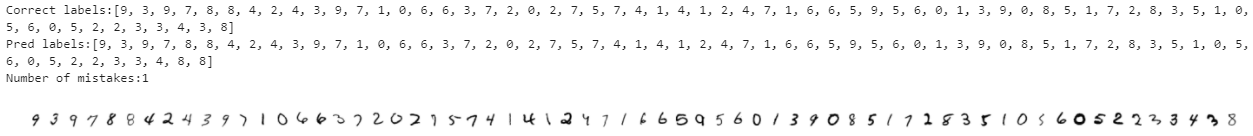
*Рис. 4– Процесс обучения 5-уровневой нейронной сети*

**

*Рис. 5 – Тестирование 5-уровневой нейронной сети*

**

*Рис. 6 – Процесс обучения сверточной нейронной сети*

**

*Рис. 7 – Тестирование сверточной нейронной сети*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Изображение входного образа | Правильный ответ | Нейронная сеть | | |
| Линейный классификатор | 5-уровневая нейронная сеть | Сверточная нейронная сеть |
|  | 3 | 3 | 3 | 3 |
|  | 4 | 4 | 4 | 4 |
|  | 6 | 6 | 6 | 6 |
|  | 8 | 3 | 1 | 8 |
|  | 5 | 5 | 5 | 5 |
|  | 9 | 9 | 9 | 9 |
|  | 5 | 5 | 5 | 5 |
|  | 4 | 9 | 4 | 4 |
|  | 2 | 2 | 2 | 3 |
|  | 3 | 3 | 3 | 3 |
|  | 9 | 9 | 9 | 9 |
|  | 7 | 7 | 7 | 7 |
|  | 4 | 9 | 9 | 4 |
|  | 5 | 3 | 5 | 5 |
|  | 6 | 6 | 6 | 6 |
|  | 5 | 0 | 5 | 5 |
| *Табл. 1 – Тестирование нейронных сетей* | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Изображение входного образа | Изображение измененного входного образа | Верный ответ | Нейронная сеть | | |
| Линейный классификатор | 5-уровневая нейронная сеть | Сверточная нейронная сеть |
|  |  | 3 | 3 | 5 | 3 |
|  |  | 8 | 8 | 8 | 8 |
|  |  | 1 | 1 | 1 | 1 |
|  |  | 9 | 4 | 4 | 9 |
|  |  | 9 | 3 | 9 | 4 |
|  |  | 9 | 9 | 8 | 8 |
|  |  | 1 | 2 | 6 | 1 |
|  |  | 7 | 7 | 7 | 4 |
|  |  | 4 | 4 | 4 | 4 |
|  |  | 1 | 6 | 6 | 1 |
|  |  | 0 | 0 | 0 | 0 |
|  |  | 6 | 5 | 6 | 6 |

*Табл. 2 – Эксперименты с нейронными сетями*

**Листинг:**

**import libs**

import numpy as np

import os.path

import torch

import torchvision

import matplotlib.pyplot as plt

from time import time

from torchvision import datasets, transforms

from torch import nn, optim

from ipywidgets import IntProgress

from os import path

from tqdm import tqdm

from random import randrange

from sklearn.metrics import accuracy\_score

## Loading and processing data

transform = transforms.Compose([transforms.ToTensor(),

                              transforms.Normalize((0.5,), (0.5,)),

                              ]) # преобразование изображений в тензор и уменьшение значений до [0;1]

if not path.exists('dataset\_mnist'):

    dataset = datasets.MNIST('dataset\_mnist', download=True, train=True, transform=transform)

    #val\_data = datasets.MNIST('dataset\_mnist', download=True, train=False, transform=transform)

else:

    dataset = datasets.MNIST('dataset\_mnist', download=False, train=True, transform=transform)

train\_data, val\_data, test\_data = torch.utils.data.random\_split(dataset, [50000, 2000, 8000])

train\_data = torch.utils.data.DataLoader(train\_data, batch\_size=64, shuffle=True)

val\_data = torch.utils.data.DataLoader(val\_data, batch\_size=64, shuffle=True)

test\_data = torch.utils.data.DataLoader(test\_data, batch\_size=64, shuffle=True)

## Check data

dataiter = iter(train\_data)

images, labels = dataiter.next()

print(images.shape)

print(labels.shape)

plt.rcParams['figure.dpi'] = 300

figure = plt.figure()

num\_of\_images = 64

for index in range(0, num\_of\_images):

    plt.subplot(1, 64, index + 1)

    plt.axis('off')

    plt.imshow(images[index].numpy().squeeze(), cmap='gray\_r')

print(labels.tolist())

## Multilayer neural network

input\_dimension = 28\*28

hidden\_dimension = [128, 64]

output\_dimension = 10

class NumberRecognizer(nn.Module):

    def \_\_init\_\_(self, input\_dim, hidden\_dim\_in, hidden\_dim\_out, out\_dim):

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

        self.fc1 = nn.Linear(input\_dimension, hidden\_dim\_in)

        self.fc2 = nn.ReLU()

        self.fc3 = nn.Linear(hidden\_dim\_in, hidden\_dim\_out)

        self.fc4 = nn.ReLU()

        self.fc5 = nn.Linear(hidden\_dim\_out, output\_dimension)

        self.fc6 = nn.LogSoftmax(dim=1)

    def forward(self, x\_in):

        x1 = self.fc2(self.fc1(x\_in))

        x2 = self.fc4(self.fc3(x1))

        logits = self.fc6(self.fc5(x2))

        return logits

## Training

learning\_rate = 1e-4

epochs = 10

numberRecognizer = NumberRecognizer(input\_dimension, hidden\_dimension[0], hidden\_dimension[1], output\_dimension)

loss\_fn = nn.NLLLoss()

optimizer = optim.Adam(numberRecognizer.parameters(), lr=learning\_rate)

start\_time= time()

for epoch in range(epochs):

    numberRecognizer.train()

    running\_loss = 0

    for images, labels in tqdm(train\_data):

        images = images.view(images.shape[0], -1) # преобразование в вектор [784]

        optimizer.zero\_grad()

        logits = numberRecognizer(images)

        loss = loss\_fn(logits, labels)

        loss.backward()

        optimizer.step()

        running\_loss += loss.item()

    numberRecognizer.eval()

    correct\_preds\_count, all\_preds\_count = 0, 0

    for images, labels in tqdm(val\_data):

        for i in range(len(labels)):

            image = images[i].view(1, 784)

            logit = numberRecognizer(image)

            prob = torch.exp(logit)

            prob = list(prob.detach().numpy()[0])

            pred\_label = prob.index(max(prob))

            true\_label = labels.numpy()[i]

            if(true\_label == pred\_label):

                correct\_preds\_count += 1

            all\_preds\_count += 1

    print(f'Epoch:{epoch + 1}\nTraining loss:{running\_loss/len(train\_data)}\nAccuracy score:{correct\_preds\_count/all\_preds\_count}')

print(f'Training Time:{round((time() - start\_time) / 60, 2)}m')

## Save model

torch.save(numberRecognizer.state\_dict(), 'models/NumberRecognizer\_params.pth')

## Load model

numberRecognizer.load\_state\_dict(torch.load('models/NumberRecognizer\_params.pth'))

## Test

dataiter = iter(test\_data)

images, labels = dataiter.next()

pred\_labels = []

mistakes\_count = 0

for i in range(len(labels)):

    image = images[i].view(1, 784)

    with torch.no\_grad():

        logit = numberRecognizer(image)

    prob = torch.exp(logit)

    prob = list(prob.detach().numpy()[0])

    pred\_label = prob.index(max(prob))

    true\_label = labels.numpy()[i]

    if pred\_label != true\_label:

        mistakes\_count += 1

    pred\_labels.append(pred\_label)

plt.rcParams['figure.dpi'] = 300

figure = plt.figure()

num\_of\_images = 64

for index in range(0, num\_of\_images):

    plt.subplot(1, 64, index + 1)

    plt.axis('off')

    plt.imshow(images[index].numpy().squeeze(), cmap='gray\_r')

print(f'Correct labels:{labels.tolist()}\nPred labels:{pred\_labels}\nNumber of mistakes:{mistakes\_count}')

## Single layer neural network

input\_dimension = 28\*28

output\_dimension = 10

class LinearClassifier(nn.Module):

    def \_\_init\_\_(self, in\_dim, out\_dim):

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

        self.fc1 = nn.Linear(in\_dim, out\_dim)

    def forward(self, x\_in):

        return torch.tanh(self.fc1(x\_in))

## Training

learning\_rate = 1e-3

epochs = 10

linearClassifier = LinearClassifier(input\_dimension, output\_dimension)

loss\_fn = nn.CrossEntropyLoss()

optimizer = optim.Adam(linearClassifier.parameters(), lr=learning\_rate)

start\_time= time()

for epoch in range(epochs):

    linearClassifier.train()

    running\_loss = 0

    for images, labels in tqdm(train\_data):

        images = images.view(images.shape[0], -1) # преобразование в вектор [784]

        optimizer.zero\_grad()

        logits = linearClassifier(images)

        loss = loss\_fn(logits, labels)

        loss.backward()

        optimizer.step()

        running\_loss += loss.item()

    linearClassifier.eval()

    correct\_preds\_count, all\_preds\_count = 0, 0

    for images, labels in tqdm(val\_data):

        for i in range(len(labels)):

            image = images[i].view(1, 784)

            logit = linearClassifier(image)

            prob = torch.exp(logit)

            prob = list(prob.detach().numpy()[0])

            pred\_label = prob.index(max(prob))

            true\_label = labels.numpy()[i]

            if(true\_label == pred\_label):

                correct\_preds\_count += 1

            all\_preds\_count += 1

    print(f'Epoch:{epoch + 1}\nTraining loss:{running\_loss/len(train\_data)}\nAccuracy score:{correct\_preds\_count/all\_preds\_count}')

print(f'Training Time:{round((time() - start\_time) / 60, 2)}m')

## Convolutional neural network

class ConvNumberRecognizer(nn.Module):

    def \_\_init\_\_(self):

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

        self.conv2d1 = nn.Conv2d(1, 4, kernel\_size=3, stride=1, padding=1)

        self.batchnorm2d1 = nn.BatchNorm2d(4)

        self.relu1 = nn.ReLU(inplace=True)

        self.maxpool2d1 = nn.MaxPool2d(kernel\_size=2, stride=2)

        self.conv2d2 = nn.Conv2d(4, 4, kernel\_size=3, stride=1, padding=1)

        self.batchnorm2d2 = nn.BatchNorm2d(4)

        self.relu2 = nn.ReLU(inplace=True)

        self.maxpool2d2 = nn.MaxPool2d(kernel\_size=2, stride=2)

        self.linear = nn.Linear(7 \* 7 \* 4, 10)

    def forward(self, x\_in):

        x = self.relu1(self.batchnorm2d1(self.conv2d1(x\_in)))

        x = self.maxpool2d1(x)

        x = self.relu2(self.batchnorm2d2(self.conv2d2(x)))

        x = self.maxpool2d2(x)

        x = x.view(x.size(0), -1)

        logits = self.linear(x)

        return logits

## Training

learning\_rate = 1e-3

epochs = 10

convNumberRecognizer = ConvNumberRecognizer()

loss\_fn = nn.CrossEntropyLoss()

optimizer = optim.Adam(convNumberRecognizer.parameters(), lr=learning\_rate)

start\_time = time()

for epoch in range(epochs):

    convNumberRecognizer.train()

    running\_loss = 0

    for images, labels in tqdm(train\_data):

        optimizer.zero\_grad()

        images, labels = torch.autograd.Variable(images), torch.autograd.Variable(labels)

        logits = convNumberRecognizer(images)

        loss = loss\_fn(logits, labels)

        loss.backward(retain\_graph=True)

        optimizer.step()

        running\_loss += loss.item()

    convNumberRecognizer.eval()

    acc\_score = 0

    all\_preds\_count = 0

    for images, labels in tqdm(val\_data):

        preds = convNumberRecognizer(images)

        preds = torch.exp(preds)

        preds = list(preds.detach().numpy())

        preds = np.argmax(preds, axis=1)

        acc\_score += accuracy\_score(labels, preds)

        all\_preds\_count += 1

    print(f'Epoch:{epoch + 1}\nTraining loss:{running\_loss/len(train\_data)}\nAccuracy score:{acc\_score / all\_preds\_count}')

print(f'Training Time:{round((time() - start\_time) / 60, 2)}m')

## Test

dataiter = iter(test\_data)

images, labels = dataiter.next()

pred\_labels = []

mistakes\_count = 0

for i in range(len(labels)):

    image = images[i]

    image = image.reshape(1, 1, 28, 28)

    with torch.no\_grad():

        logit = convNumberRecognizer(image)

    prob = torch.exp(logit)

    prob = list(prob.detach().numpy()[0])

    pred\_label = prob.index(max(prob))

    true\_label = labels.numpy()[i]

    if pred\_label != true\_label:

        mistakes\_count += 1

    pred\_labels.append(pred\_label)

plt.rcParams['figure.dpi'] = 300

figure = plt.figure()

num\_of\_images = 64

for index in range(0, num\_of\_images):

    plt.subplot(1, 64, index + 1)

    plt.axis('off')

    plt.imshow(images[index].numpy().squeeze(), cmap='gray\_r')

print(f'Correct labels:{labels.tolist()}\nPred labels:{pred\_labels}\nNumber of mistakes:{mistakes\_count}')

## Save model

torch.save(convNumberRecognizer.state\_dict(), 'models/ConvNumberRecognizer\_params.pth')

## Load model

convNumberRecognizer.load\_state\_dict(torch.load('models/ConvNumberRecognizer\_params.pth'))

## Experiments

#### *Secondary functions*

def convert\_logit\_to\_prediction(logit, labels):

    prob = torch.exp(logit)

    prob = list(prob.detach().numpy()[0])

    pred\_label = prob.index(max(prob))

    return prob.index(max(prob))

def replace\_pixel(image, in\_range, out\_color):

    for i in range(image.numpy().squeeze().shape[0]):

        for j in range(image.numpy().squeeze().shape[1]):

            if image.numpy()[0][i][j] > in\_range:

                print(f'replaced pixel index:[{i}, {j}]')

                image.numpy()[0][i][j] = out\_color

                return image

def add\_row(image, out\_color):

    row\_index = randrange(28)

    for i in range(image.numpy().squeeze().shape[1]):

        image.numpy()[0][row\_index][i] = out\_color

    return image

def add\_column(image, out\_color):

    column\_index = randrange(28)

    for i in range(image.numpy().squeeze().shape[1]):

        image.numpy()[0][i][column\_index] = out\_color

    return image

#### *Delete pixel(s) and noise*

dataiter = iter(test\_data)

images, labels = dataiter.next()

plt.rcParams['figure.dpi'] = 100

plt.imshow(images[0].numpy().squeeze(), cmap='gray\_r')

print(f'Correct label:{labels[0].item()}\nMultylayer NN:{convert\_logit\_to\_prediction(numberRecognizer(images[0].view(images[0].size(0), -1)), labels)}\nSingle layer NN:{convert\_logit\_to\_prediction(linearClassifier(images[0].view(images[0].size(0), -1)), labels)}\nConvolutioan NN:{convert\_logit\_to\_prediction(convNumberRecognizer(images[0].reshape(1, 1, 28, 28)), labels)}')

image = replace\_pixel(images[0], 0.8, -1.0)

plt.imshow(image.numpy().squeeze(), cmap='gray\_r')

print(f'Correct label:{labels[0].item()}\nMultylayer NN:{convert\_logit\_to\_prediction(numberRecognizer(image.view(images[0].size(0), -1)), labels)}\nSingle layer NN:{convert\_logit\_to\_prediction(linearClassifier(images[0].view(image.size(0), -1)), labels)}\nConvolutioan NN:{convert\_logit\_to\_prediction(convNumberRecognizer(image.reshape(1, 1, 28, 28)), labels)}')

dataiter = iter(test\_data)

images, labels = dataiter.next()

plt.rcParams['figure.dpi'] = 100

plt.imshow(images[0].numpy().squeeze(), cmap='gray\_r')

print(f'Correct label:{labels[0].item()}\nMultylayer NN:{convert\_logit\_to\_prediction(numberRecognizer(images[0].view(images[0].size(0), -1)), labels)}\nSingle layer NN:{convert\_logit\_to\_prediction(linearClassifier(images[0].view(images[0].size(0), -1)), labels)}\nConvolutioan NN:{convert\_logit\_to\_prediction(convNumberRecognizer(images[0].reshape(1, 1, 28, 28)), labels)}')

pixels\_count = 30

for i in range(pixels\_count):

    image = replace\_pixel(images[0], 0.8, -1.0)

plt.imshow(image.numpy().squeeze(), cmap='gray\_r')

print(f'Correct label:{labels[0].item()}\nMultylayer NN:{convert\_logit\_to\_prediction(numberRecognizer(image.view(images[0].size(0), -1)), labels)}\nSingle layer NN:{convert\_logit\_to\_prediction(linearClassifier(images[0].view(image.size(0), -1)), labels)}\nConvolutioan NN:{convert\_logit\_to\_prediction(convNumberRecognizer(image.reshape(1, 1, 28, 28)), labels)}')

#### *Add a row or column*

dataiter = iter(test\_data)

images, labels = dataiter.next()

plt.rcParams['figure.dpi'] = 100

plt.imshow(images[0].numpy().squeeze(), cmap='gray\_r')

print(f'Correct label:{labels[0].item()}\nMultylayer NN:{convert\_logit\_to\_prediction(numberRecognizer(images[0].view(images[0].size(0), -1)), labels)}\nSingle layer NN:{convert\_logit\_to\_prediction(linearClassifier(images[0].view(images[0].size(0), -1)), labels)}\nConvolutioan NN:{convert\_logit\_to\_prediction(convNumberRecognizer(images[0].reshape(1, 1, 28, 28)), labels)}')

image = add\_row(images[0], 1.0)

plt.imshow(image.numpy().squeeze(), cmap='gray\_r')

print(f'Correct label:{labels[0].item()}\nMultylayer NN:{convert\_logit\_to\_prediction(numberRecognizer(image.view(images[0].size(0), -1)), labels)}\nSingle layer NN:{convert\_logit\_to\_prediction(linearClassifier(images[0].view(image.size(0), -1)), labels)}\nConvolutioan NN:{convert\_logit\_to\_prediction(convNumberRecognizer(image.reshape(1, 1, 28, 28)), labels)}')

dataiter = iter(test\_data)

images, labels = dataiter.next()

plt.rcParams['figure.dpi'] = 100

plt.imshow(images[0].numpy().squeeze(), cmap='gray\_r')

print(f'Correct label:{labels[0].item()}\nMultylayer NN:{convert\_logit\_to\_prediction(numberRecognizer(images[0].view(images[0].size(0), -1)), labels)}\nSingle layer NN:{convert\_logit\_to\_prediction(linearClassifier(images[0].view(images[0].size(0), -1)), labels)}\nConvolutioan NN:{convert\_logit\_to\_prediction(convNumberRecognizer(images[0].reshape(1, 1, 28, 28)), labels)}')

image = add\_column(images[0], 1.0)

plt.imshow(image.numpy().squeeze(), cmap='gray\_r')

print(f'Correct label:{labels[0].item()}\nMultylayer NN:{convert\_logit\_to\_prediction(numberRecognizer(image.view(images[0].size(0), -1)), labels)}\nSingle layer NN:{convert\_logit\_to\_prediction(linearClassifier(images[0].view(image.size(0), -1)), labels)}\nConvolutioan NN:{convert\_logit\_to\_prediction(convNumberRecognizer(image.reshape(1, 1, 28, 28)), labels)}')