Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №4**

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: «Сверточные сети»

**Выполнил:**

Студент 3 курса

Группы ИИ-21

Карагодин Д. Л.

**Проверил:**

Туз И. С.

Брест 2024

**Цель:** необходимо реализовать и обучить свёрточную нейронную сеть для классификации изображений.

**Ход работы**

**Вариант 1**

1. [mnist digits](https://paperswithcode.com/dataset/mnist)

**Код программы:**

import time

import json

from sklearn.model\_selection import train\_test\_split

from image\_reader import get\_labels, read\_idxfile

from models import \*

class CnnFromScratch:

    def \_\_init\_\_(self, load\_weights=False):

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

        self.conv2 = Conv2d(1, 1, 2, 2, padding=1)

        self.max\_pool = Maxpool2d(2, 2, padding=1)

        self.fc1 = Linear(64, 30)

        self.fc2 = Linear(30, 10)

        self.flatten = Flatten()

        self.relu = ReLU()

        self.sigmoid1 = Sigmoid()

        self.sigmoid2 = Sigmoid()

        self.softmax = Softmax()

        if load\_weights:

            with open('weights.json') as file:

                data = json.load(file)

                self.fc1.fc\_w = np.array(data['fc1']['w'])

                self.fc1.fc\_b = np.array(data['fc1']['b'])

                self.fc2.fc\_w = np.array(data['fc2']['w'])

                self.fc2.fc\_b = np.array(data['fc2']['b'])

    def \_\_call\_\_(self, x):

        x = self.conv1(x)

        x = self.relu(x)

        x = self.max\_pool(x)

        x = self.conv2(x)

        x = self.sigmoid1(x)

        x = self.flatten.matrices2vector(x)

        x = self.fc1(x)

        x = self.sigmoid2(x)

        x = self.fc2(x)

        x = self.softmax(x)

        return x

    def backprop(self, x, lr=0.01):

        x = self.softmax.backprop(x)

        x = self.fc2.backprop(x, lr)

        x = self.sigmoid2.backprop(x)

        x = self.fc1.backprop(x, lr)

        x = self.flatten.vector2matrices(x)

        x = self.sigmoid1.backprop(x)

        x = self.conv2.backprop(x, lr)

        x = self.max\_pool.backprop(x)

        x = self.relu.backprop(x)

        x = self.conv1.backprop(x, lr)

    def change\_weights\_file(self):

        with open("weights.json", 'w') as file:

            data = {

                'fc1': {

                    'w': self.fc1.fc\_w.tolist(),

                    'b': self.fc1.fc\_b.tolist()

                },

                'fc2': {

                    'w': self.fc2.fc\_w.tolist(),

                    'b': self.fc2.fc\_b.tolist()

                },

            }

            json.dump(data, file)

import cv2

def train\_loop(images, labels, model, criterion, print\_log\_freq, lr, change\_weights\_file):

    loss\_log = []

    acc\_log = []

    start\_time = time.time()

    for idx in range(len(images)):

        image = images[idx]

        target = labels[idx]

        # print(target.argmax())

        # cv2.imshow('', image)

        # cv2.waitKey(0)

        pred = model([image])

        loss = criterion(target, pred)

        x = criterion.backprop(target, pred)

        model.backprop(x, lr=lr)

        loss\_log.append(loss.sum())

        acc\_log.append(pred.argmax() == target.argmax())

        if idx % print\_log\_freq == 0:

            if change\_weights\_file:

                model.change\_weights\_file()

            loss\_avg = sum(loss\_log[-print\_log\_freq:])/print\_log\_freq

            acc\_avg = sum(acc\_log[-print\_log\_freq:])/print\_log\_freq

            loop\_time = time.time() - start\_time

            start\_time = time.time()

            print(f'Train step {idx}, Loss: {loss\_avg:.5f}, '

                  f'Acc: {acc\_avg:.4f}, time: {loop\_time:.1f}')

def train(model, change\_weights):

    imgs = read\_idxfile("train-images.idx3-ubyte") # [[0, 255, 34, ... 784], ...]

    lbls = read\_idxfile("train-labels.idx1-ubyte") # [[0, 0, 1, 0 ... 9], ...]

    # Поиск картинок с 1, 3, 4, 0

    poss = []

    for i in range(len(lbls)):

        if lbls[i] in (1,3,4,0):

            poss.append(i)

    images = []

    labels = []

    for i in poss[:1000]:

        images.append(imgs[i])

        labels.append(lbls[i])

    images = np.array(images)

    labels = get\_labels(np.array(labels))

    train\_images, test\_images, train\_labels, test\_labels = train\_test\_split(images, labels, test\_size=0.2)

    criterion = CrossEntropyLoss()

    for epoch in range(1):

        train\_loop(train\_images, train\_labels, model, criterion, print\_log\_freq=100, lr=0.04, change\_weights\_file=change\_weights)

if \_\_name\_\_ == '\_\_main\_\_':

    model = CnnFromScratch(load\_weights=True)

    train(model, change\_weights=False)

**Результат программы:**

Train step 0, Loss: 0.08196, Acc: 0.0000, time: 0.1

Train step 100, Loss: 1.61358, Acc: 0.3700, time: 10.4

Train step 200, Loss: 1.32112, Acc: 0.4100, time: 10.6

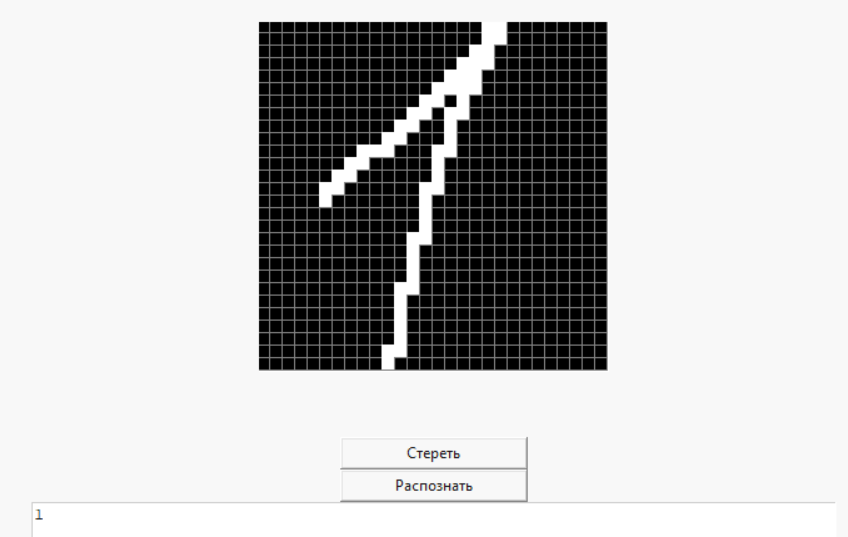
Train step 300, Loss: 1.22529, Acc: 0.4700, time: 10.4

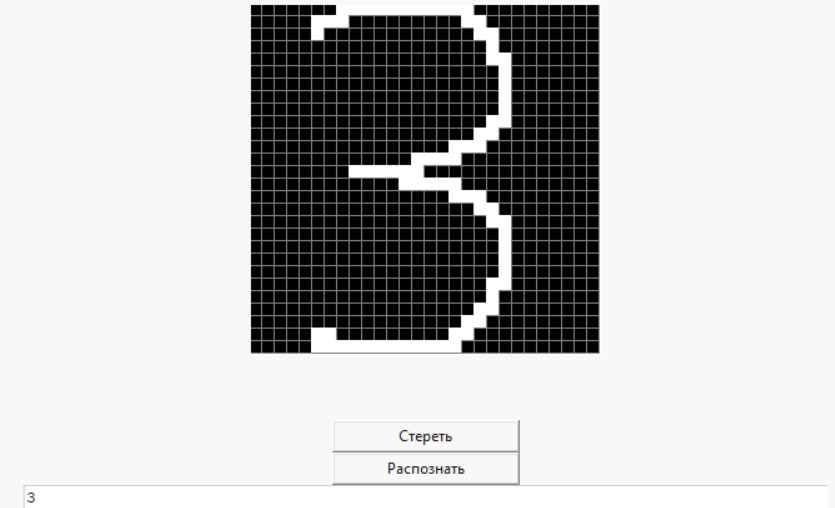
Train step 400, Loss: 0.98444, Acc: 0.5900, time: 10.4

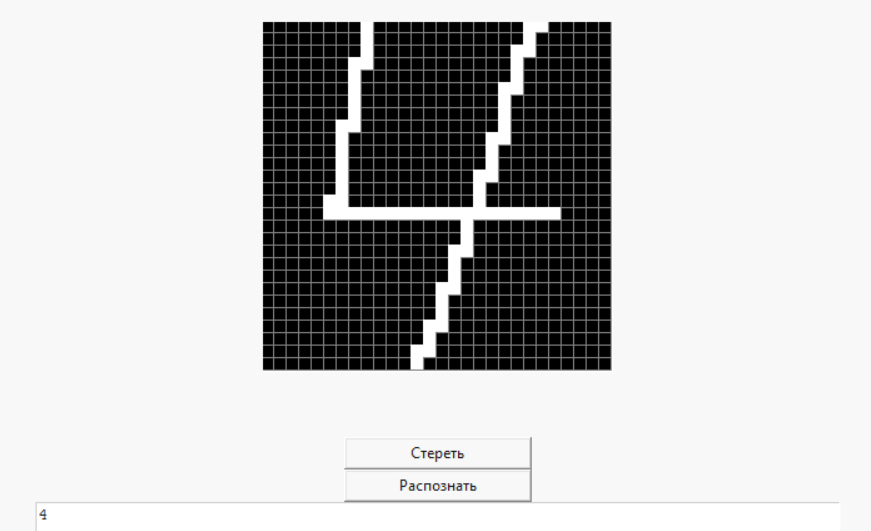
Train step 500, Loss: 0.80045, Acc: 0.7100, time: 10.5

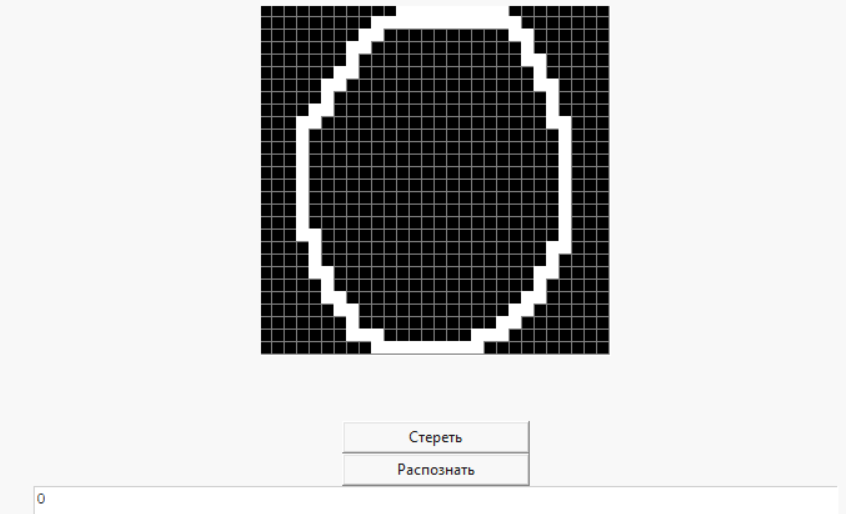
Train step 600, Loss: 0.65442, Acc: 0.7600, time: 10.5

Train step 700, Loss: 0.81128, Acc: 0.6900, time: 10.

****

****

****

****

**Цель:** реализовал однослойную нейронную сеть для решения задачи классификации с использованием пороговой функции активации.