Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования «Московский политехнический университет»

**факультет информационных технологий**

**Кафедра СМАРТ-технологий**

Дисциплина: Нейронные сети в задачах технического зрения и управления

Отчёт по лабораторной работе №1

«Базовые принципы применения нейронных сетей для обработки изображений»

Работу выполнил\_и

студент\_ка 3 курса

очного отделения

<ФИО>

Научный руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<ФИО>

**Цель работы**

Ознакомиться с созданием проектов для платы, рассмотреть их

структуру; изучить принципы работы с портами ввода/вывода и организации их взаимодействия.

**Индивидуальное задание**

Разработать алгоритм обработки изображения с использованием простого однослойного перцептрона без обучения

**Ход работы**

Суть данной работы сводится к созданию простой нейронной сети, на вход которой подаётся массив пикселей из битового файла, чтобы, в результате сеть предсказала на каком изображении какая цифра изображена.



Для этого был подготовлен следующий набор изображений 2x5 пикселей:

Изображения загружаются в виде массива 2x5, значения 0-255 приводятся к виду 0-1, далее, массив «выпрямляется» в одномерный.

Выходной массив для обучения состоит из цифр от 0 до 9, разделённых на

9 для получения диапазона 0-1.

Затем генерируются начальные случайный веса:

synaptic\_weights = 2 \* np.random.random((10, 1)) - 1

После чего начинается процесс обучения сети, алгоритм которого выглядит так:

1. Вычисляются предсказанные значения, как скалярное произведение входного массива на веса, полученные значение, проходит через функцию активации – сигмоиду.

2. Вычисляется ошибка – разность между требуемыми значениями и предсказанными значениями

3. Вычисляются корректирующие значения, как скалярное произведение входных данных на ошибка ∗ (предсказанный выход ∗ (1 − предсказанный выход))

4. Полученные значения корректировки прибавляются к весам.

Данный алгоритм повторяется определённое количество итераций, пока не будет достигнут требуемый результат.

**Результат работы**

Training inputs:

[[1 1 1 1 1 1 1 1 1 1] [0 1 0 1 0 1 0 1 0 1] [1 1 0 1 1 1 1 0 1 1] [1 1 0 1 1 1 0 1 1 1] [1 1 1 1 1 1 0 1 0 1] [1 1 1 0 1 1 0 1 1 1] [1 1 1 0 1 1 1 1 1 1] [1 1 0 1 0 1 0 1 0 1] [1 1 1 1 0 0 1 1 1 1] [1 1 1 1 1 1 0 1 1 1]]

Training outputs: [[0.11111111] [0.77777778] [0.33333333] [0.22222222] [0.55555556] [0.44444444] [0.88888889] [0.66666667]

[1. ] [0. ]]

Synaptic weights: [[-0.16595599]

[ 0.44064899] [-0.99977125] [-0.39533485] [-0.70648822] [-0.81532281] [-0.62747958] [-0.30887855] [-0.20646505]

[ 0.07763347]]

Iteration 0 Predicted values: [2 6 4 5 3 3 2 6 4 3] Iteration 10 Predicted values: [1 5 1 1 1 2 2 5 6 1] Iteration 20 Predicted values: [5 7 3 3 5 5 7 8 9 3] Iteration 30 Predicted values: [6 8 3 3 5 6 8 8 9 3] Iteration 40 Predicted values: [5 8 3 2 4 6 8 8 9 2] Iteration 50 Predicted values: [5 8 2 2 4 6 8 8 9 2] Iteration 60 Predicted values: [4 7 2 2 4 6 8 8 8 2] Iteration 70 Predicted values: [4 7 3 2 5 6 8 8 8 2]

Iteration 80 Predicted values: [4 7 3 2 5 6 8 8 8 2] Iteration 90 Predicted values: [4 7 3 2 5 6 8 8 8 1] Iteration 100 Predicted values: [4 7 3 2 5 6 8 8 8 1]

. . .

Iteration 890 Predicted values: [2 7 4 2 5 5 8 7 9 0]

. . .

Iteration 2980 Predicted values: [1 7 3 2 5 4 8 6 9 0] Iteration 2990 Predicted values: [1 7 3 2 5 4 8 6 9 0]

------------- Predicted values:

[1 7 3 2 5 4 8 6 9 0]

Synaptic weights: [[-0.52748608]

[ 5.74840084] [-2.75424207] [-3.83529435] [ 2.24268189] [-7.62457707] [ 2.15922275] [ 1.56838326] [-4.24558363]

[ 5.38538532]]

-------------

Process finished with exit code 0

Как можно заметить, требуемый результат был достигнут менее чем за 3000 итераций.

**Исходный код**

import numpy as np import cv2

# Sigmoid activation function def sigmoid(x):

return 1 / (1 + np.exp(-x))

# Define training array training\_inputs = []

training\_outputs = [1, 7, 3, 2, 5, 4, 8, 6, 9, 0]

for i in range(0, 10):

train\_binary = cv2.bitwise\_not(cv2.imread('train\_values/' + str(i) + '.bmp', cv2.IMREAD\_UNCHANGED)).flatten()

train\_binary[train\_binary == 255] = 1 training\_inputs.append(train\_binary)

training\_inputs = np.array(training\_inputs)

training\_outputs = np.array([training\_outputs]) training\_outputs = training\_outputs / 9 training\_outputs = training\_outputs.T

print('Training inputs: ') print(training\_inputs) # 0 - 1 print()

print('Training outputs: ') print(training\_outputs) # 0 - 1 print()

# Define random weights np.random.seed(1)

synaptic\_weights = 2 \* np.random.random((10, 1)) - 1 print('Synaptic weights: ')

print(synaptic\_weights)

print()

# Predict value def think(inputs):

global synaptic\_weights

inputs = inputs.astype(float)

out\_value = sigmoid(np.dot(inputs, synaptic\_weights))

return out\_value

# Training function for i in range(3000):

output = think(training\_inputs)

error = training\_outputs - output

# Backpropagation

adjustments = np.dot(training\_inputs.T, error \* (output \* (1 - output)))

synaptic\_weights += adjustments

if i % 10 == 0:

print('Iteration ' + str(i) + ' Predicted values: ' + str((np.around(think(training\_inputs).flatten() \* 9))

.astype(int)))

print()

print('-------------') print('Predicted values: ') print((np.around(think(training\_inputs).flatten() \* 9)).astype(int)) print()

print('Synaptic weights: ') print(synaptic\_weights) print('-------------')