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

Кирилович А. А.

**Проверил:**

Туз И. С.

Брест 2024

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

**Ход работы**

import numpy as np

from scipy import signal

class Layer:

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

self.input = None

self.output = None

def \_\_call\_\_(self, input):

# TODO: return output

pass

def backward(self, output\_gradient, learning\_rate):

# TODO: update parameters and return input gradient

pass

class Conv2D(Layer):

def \_\_init\_\_(self, input\_shape, kernel\_size, depth):

super().\_\_init\_\_()

input\_depth, input\_height, input\_width = input\_shape

self.depth = depth

self.input\_shape = input\_shape

self.input\_depth = input\_depth

self.output\_shape = (depth, input\_height - kernel\_size + 1, input\_width - kernel\_size + 1)

self.kernels\_shape = (depth, input\_depth, kernel\_size, kernel\_size)

# self.kernels = np.random.randn(\*self.kernels\_shape)

# self.biases = np.random.randn(\*self.output\_shape)

self.kernels = np.ones(self.kernels\_shape)

self.biases = np.ones(self.output\_shape)

def \_\_call\_\_(self, input):

self.input = input

self.output = np.copy(self.biases)

for i in range(self.depth):

for j in range(self.input\_depth):

self.output[i] += signal.correlate2d(self.input[j], self.kernels[i, j], "valid")

return self.output

def backward(self, output\_gradient, learning\_rate):

kernels\_gradient = np.zeros(self.kernels\_shape)

input\_gradient = np.zeros(self.input\_shape)

for i in range(self.depth):

for j in range(self.input\_depth):

kernels\_gradient[i, j] = signal.correlate2d(self.input[j], output\_gradient[i], "valid")

input\_gradient[j] += signal.convolve2d(output\_gradient[i], self.kernels[i, j], "full")

self.kernels -= learning\_rate \* kernels\_gradient

self.biases -= learning\_rate \* output\_gradient

return input\_gradient

class MeanSquaredErrorLoss:

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

self.predict = None

self.target = None

def \_\_call\_\_(self, target, predict):

self.target = target

self.predict = predict

return np.mean((self.predict - self.target)\*\*2) / 2

def backward(self):

return (self.predict - self.target) / len(self.target)

class ReLU:

def \_\_call\_\_(self, x):

x = np.array(x, dtype=np.float32)

y = np.where(x > 0, x, 0)

self.y = y

return y

def backward(self, dEdy):

dydx = np.where(self.y <= 0, self.y, 1)

dEdx = dEdy \* dydx

return dEdx

**Вывод:** в ходе лабораторной работы я реализовал сверточную нейронную сеть для классификации изображений.