**МИНИСТЕРСТВО**

**ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ  
ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ  
 УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4

по дисциплине: «**Базы знаний и поддержка принятия решений в САПР»**

на тему: «Нейронная сеть Хопфилда»

Выполнил: студент гр. ИТП-31

Расшивалов Н.И.  
 Принял: профессор

Мурашко И.А.

Гомель 2022

**Цель работы:** Изучение топологии, алгоритма функционирования сети Хопфилда.

**Ход работы**

На рисунке 1 представлен пример исходного изображения.

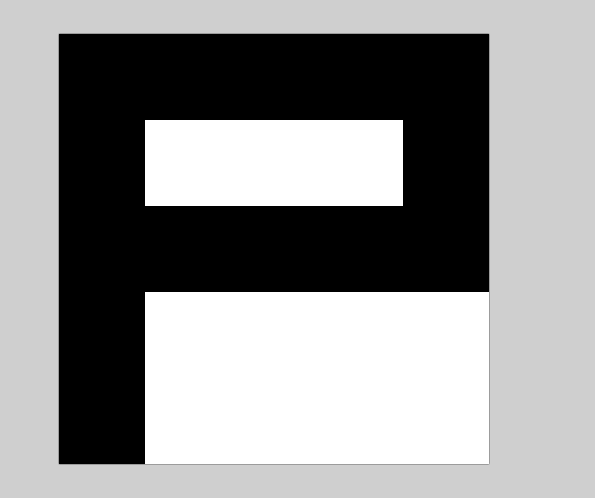


Рисунок 1 – Пример исходного изображения.

На рисунке 2 представлен результат обработки буквы Р при 10% зашумления.



Рисунок 2 – Результат обработки буквы Р при 10% зашумления

На рисунке 3 представлен результат обработки буквы Д при 40% зашумления.



Рисунок 3 – Результат обработки буквы Д при 40% зашумления

На рисунке 4 представлен результат обработки буквы О при 40% зашумления.



Рисунок 4 – Результат обработки буквы О при 40% зашумления

Листинг программы представлен в приложении А.

**Вывод:** в ходе выполнения лабораторной работы, была изучена сеть Хопфилда.

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

import numpy as np

import cv2

import random

def activation(x):

items = []

for item in x:

if item >= 0:

items.append(1)

else:

items.append(-1)

return np.array(items)

def to\_binary\_matrix(img):

bin\_img = []

for iheight in range(img.shape[0]):

for iwidth in range(img.shape[1]):

val = np.average(img[iheight][iwidth])

if val == 0:

val = 1

else:

val = -1

bin\_img.append(val)

return np.array(bin\_img)

class SampleImage:

def \_\_init\_\_(self, letter = '', noise = 0, image = []):

self.letter = letter

self.noise = noise

self.image = image

class HopfieldNetwork:

def \_\_init\_\_(self, train\_samples) -> None:

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

self.count\_neurons = train\_samples[0].shape[0]

self.train\_samples = train\_samples

self.weights = np.zeros(shape=(self.count\_neurons, self.count\_neurons))

def train(self):

for sample in self.train\_samples:

self.weights += sample \* sample.reshape(len(sample), 1)

self.weights /= self.count\_neurons

self.weights \*= 1 - np.identity(self.count\_neurons)

def fit(self, sample):

result = activation(self.weights.dot(sample.reshape(len(sample), 1)) / 2) # f(W\*X \* 1/2)

return result.flatten()

def add\_noise(image, amount):

indexes = []

for i in range(len(image)):

indexes.append(i)

noise\_count = int(amount\*len(image))

for i in range(noise\_count):

next = random.choice(indexes)

indexes.remove(next)

if image[next] == 1:

image[next] = -1

else:

image[next] = 1

return image

def str\_array\_img(array\_img):

\_str = '|'

for i in range(array\_img.shape[0]):

if array\_img[i] <= 0:

\_str += ' '

else:

\_str += '█'

if (i + 1) % 10 == 0:

\_str += '|\n|'

return \_str

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

base\_path = "D:\\university\\3\_course\\2semester\\sapr\\lab9\\samples\\"

paths = ["O.png","D.png","R.png"]

sampleImages = []

for path in paths:

letter = path.split('.')[0]

image = cv2.imread(base\_path+path)

sampleImages.append(SampleImage(letter,0\*100, add\_noise(to\_binary\_matrix(image), 0)))

for i in range(1, 11):

for j in range(10):

sampleImages.append(SampleImage(letter, i/10.0\*100, add\_noise(to\_binary\_matrix(image), i/10.0)))

print("All images")

for img in sampleImages:

print("----------------------------")

print(str\_array\_img(img.image))

np.random.shuffle(sampleImages)

training, test = sampleImages[:int(0.8\*len(sampleImages))], sampleImages[int(0.8\*len(sampleImages)):]

trainSamples = []

for img in training:

trainSamples.append(img.image)

hn = HopfieldNetwork(trainSamples)

hn.train()

for image in test:

result = hn.fit(image.image)

print(f"{image.letter} {image.noise}%")

print("before----------------------------")

print(str\_array\_img(image.image))

print("after----------------------------")

print(str\_array\_img(result))