**Лущик Никита КС-44**

**Нейронная сеть Хопфилда**

**Входные данные**

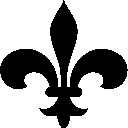
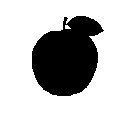
На вход программы в качестве эталонных образов необходимо подать несколько изображений произвольного цвета и размера – для ускорения обучения, в нашем случае это будут следующие 3 изображения, по 128х128 пикселей каждое:

0) 1) 2)



После обработки они преобразуются в монохромные черно-белые изображения, с глубиной цвета 1 бит:

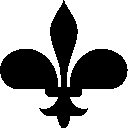
0) 1) 2)

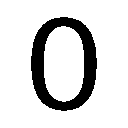
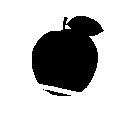


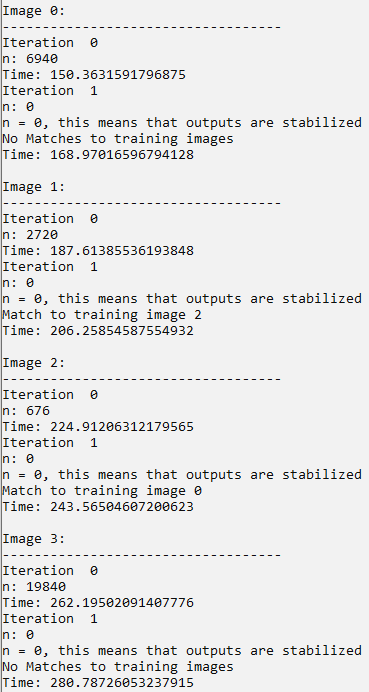
Эти изображения, в свою очередь, становятся массивами значений в диапазоне [-1; 1], где -1 – белый пиксель, 1 – черный, на которых и обучается нейронная сеть. Соотношение между количеством бинарных признаков, характеризующих образы, и количеством самих запоминаемых образов, выполняется:

**3 ≤ 2458**

В качестве тестовой выборки подадим уже обработанные и использовавшиеся при обучении образы, но зашумленные и в другом порядке, и 1 новый образ:

0) 1)  2) 3)



**Результат**

После обучения сеть успешно опознала зашумленные образы 1 и 2, и, как и ожидалось, не смогла распознать образы 0 и 3 – образа 3 не было в обучающей выборке, а образ 2 оказался слишком сильно зашумлен для распознавания.

Вывод программы

**Листинг AutoassociativeMemory.py**

import cv2

import os

import Common

import time

from numpy import array

from PIL import Image

def imgToBlackWhite(pathToImg, threshold):

imgGrey = cv2.imread(pathToImg, cv2.IMREAD\_GRAYSCALE)

cv2.imshow("Grayscale", imgGrey)

img\_binary = cv2.threshold(imgGrey, threshold, 255, cv2.THRESH\_BINARY)[1]

parsed\_filename = os.path.splitext(os.path.basename(pathToImg))[0]

new\_filename = parsed\_filename + 'B&W.bmp'

cv2.imwrite(new\_filename, img\_binary)

cv2.imshow("BlackWhite", img\_binary)

cv2.waitKey()

def imageRecognizer(filename):

img = Image.open(filename)

img = img.convert('1')

return list(array(img))

def binArrayParser(array):

returnArray = list()

for i in array:

for j in i:

if (j == False):

returnArray.append(1)

elif (j == True):

returnArray.append(-1)

return returnArray

def Hopfield(X, Xprac):

start = time.time()

f = open(f"log{int(time.time())}.txt", "w")

print(f"Start time {start}")

f.write(f"Start time {start}")

f.write("\nX(training images):\n")

Common.NumerizedFilePrint(X, f)

f.write("Y(noisy images):\n")

Common.NumerizedFilePrint(Xprac, f)

W = Common.multiply(Common.transpose(X), X)

W = Common.diagonalNullifier(W)

print("W calculated\nTime:", time.time() - start)

f.write("W:\n")

Common.NumerizedFilePrint(W, f)

for prac in range(len(Xprac)):

Y = [Xprac[prac]] # 2.1

n = 1.0

counter = 0

print(f"\nImage {prac}:\n-----------------------------------")

f.write(f"\n\nImage {prac}(noisy image): {Y}\n----------------------------")

while(n > 0.0):

print("Iteration ", counter)

f.write(f"\nIteration {counter}\n")

f.write("Y0(input image+calc data):\n")

Common.NumerizedFilePrint(Y, f)

temp = []

S = []

for i in range(len(W)): # 2.2

tmp = 0

for j in range(len(W[0])):

tmp += W[i][j]\*Y[counter][j]

S.append(tmp)

f.write(f"S(neuron state):\n {S}\n")

temp = []

for j in range(len(Y[0])): # 2.3

temp.append(Common.SingleJump(S[j], 0))

Y.append(temp)

f.write(f"Y1(output image):\n {Y[counter+1]}\n")

n = sum(Common.powList(Common.difference(Y[counter], Y[counter+1]))) # 2.4

print(f"n: {n}")

f.write(f"n: {n}")

if(n == 0):

print("n = 0, this means that outputs are stabilized")

f.write(f"\nn = 0, this means that outputs are stabilized")

switcher = False

for i in range(len(X)): # 2.2

if (X[i] == Y[counter+1]):

print("Match to training image", i)

f.write(f"\nMatch to training image {i}")

switcher = True

if (switcher == False):

print("No Matches to training images")

f.write("\nNo Match to training images")

if(counter > 1 and sum(Common.powList(Common.difference(Y[counter], Y[counter - 2]))) == 0 and sum(Common.powList(Common.difference(Y[counter - 1], Y[counter - 3]))) == 0):

print("This image not recognized")

f.write("\nThis image not recognized")

break

counter += 1

print("Time:", time.time() - start)

f.write(f"\nTotal time:{time.time() - start}")

print(f"\nTotal time:{time.time() - start}")

f.close()

def main():

inputArray = list()

PracticeArray = list()

imgToBlackWhite('img1.bmp', 230)

imgToBlackWhite('img2.bmp', 240)

imgToBlackWhite('img3.bmp', 230)

inputArray.append(binArrayParser(imageRecognizer('mono128x128/img1mono.bmp')))

inputArray.append(binArrayParser(imageRecognizer('mono128x128/img2mono.bmp')))

inputArray.append(binArrayParser(imageRecognizer('mono128x128/img3mono.bmp')))

PracticeArray.append(binArrayParser(imageRecognizer('mono128x128/img4mono.bmp')))

PracticeArray.append(binArrayParser(imageRecognizer('mono128x128/img5mono.bmp')))

PracticeArray.append(binArrayParser(imageRecognizer('mono128x128/img6mono.bmp')))

Hopfield(inputArray, PracticeArray)

main()

**Листинг Common.py**

def SingleJump(S, T):

if (S <= T):

return -1

elif (S > T):

return 1

def transpose(array):

transposed = list()

for i in range(len(array[0])):

row = list()

for sublist in array:

row.append(sublist[i])

transposed.append(row)

return transposed

def multiply(array0, array1):

multiplied = []

counter = len(array0)

for k in range(len(array0)):

stringsum = []

for i in range(len(array1[0])):

string = 0

for j in range(len(array0[0])):

string += array0[k][j]\*array1[j][i]

stringsum.append(string)

print(k, " from", counter)

multiplied.append(stringsum.copy())

return multiplied

def diagonalNullifier(array):

nullified = array

for i in range(len(nullified)):

for j in range(len(nullified[0])):

if(i == j):

nullified[i][j] = 0

return nullified

def difference(array0, array1):

diff = list()

for i in range(len(array0)):

diff.append(array0[i]- array1[i])

return diff

def powList(array: list) -> list:

powList = array

for i in range(len(powList)):

powList[i] = powList[i]\*\*2

return powList

def NumerizedPrint(array, end = "\n"):

for i in range(len(array)):

print("%3d) %s" % (i, array[i]), end=end)

def NumerizedFilePrint(array, f):

counter = 0

for line in array:

f.write(f"{counter}){line}\n")

counter += 1