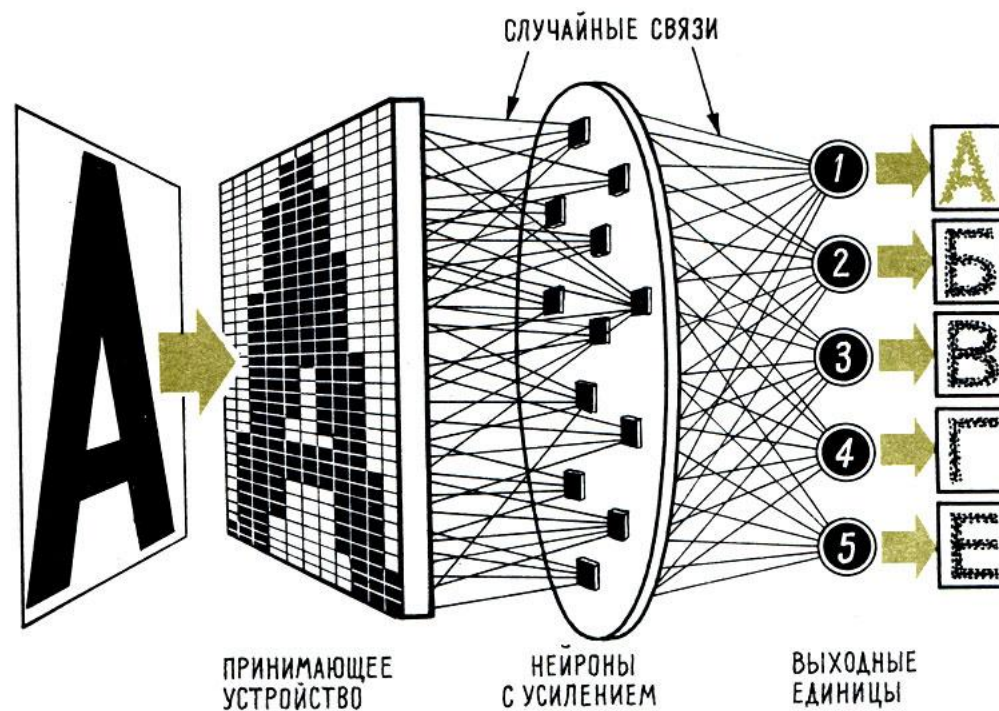




Проект #24: [Распознавание образов]



Техническое задание

Разработать проект, включающий в себя:

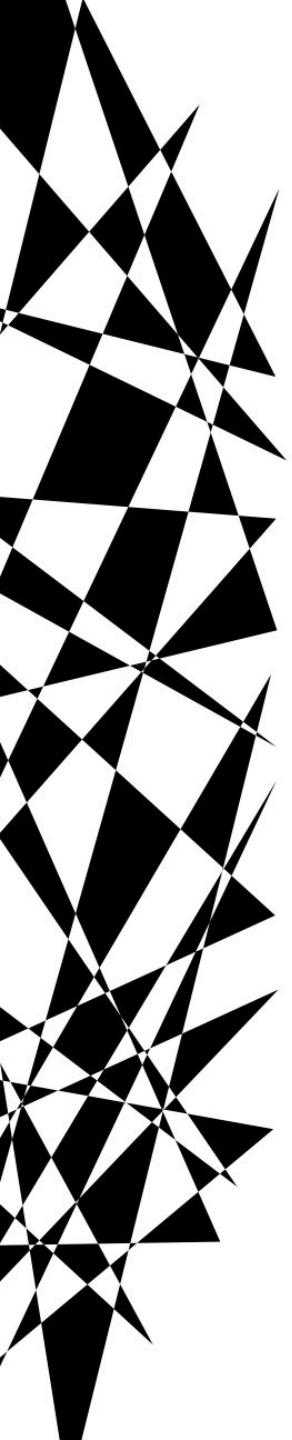
- Распознавание поданных образов
- Распознавание искаженных образов



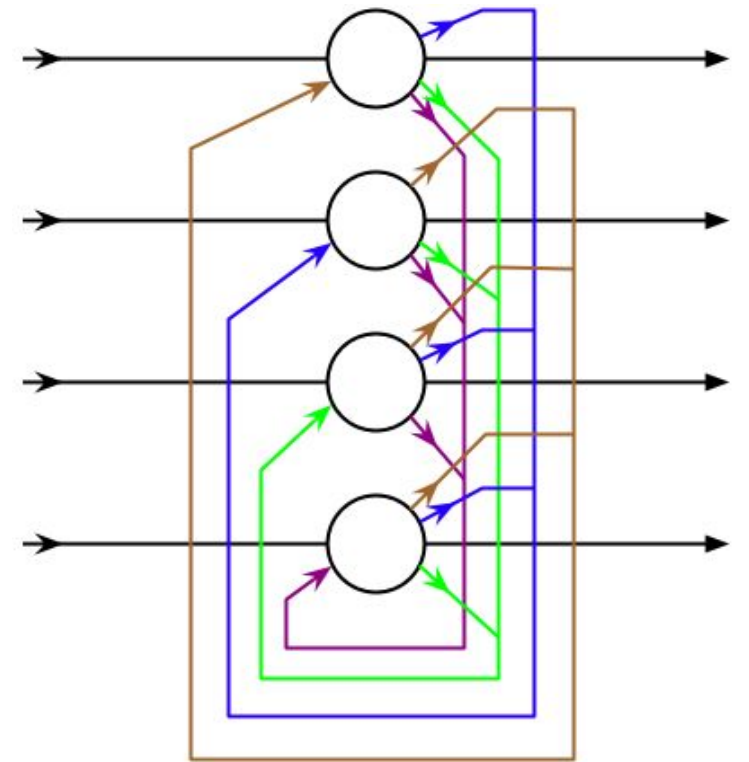
Используемые фреймворки и библиотеки

- numpy <http://www.numpy.org>
- functools <https://docs.python.org/2/library/functools.html>
- os <https://docs.python.org/3/library/os.html>
- Pillow <https://pillow.readthedocs.io/en/5.3.x>



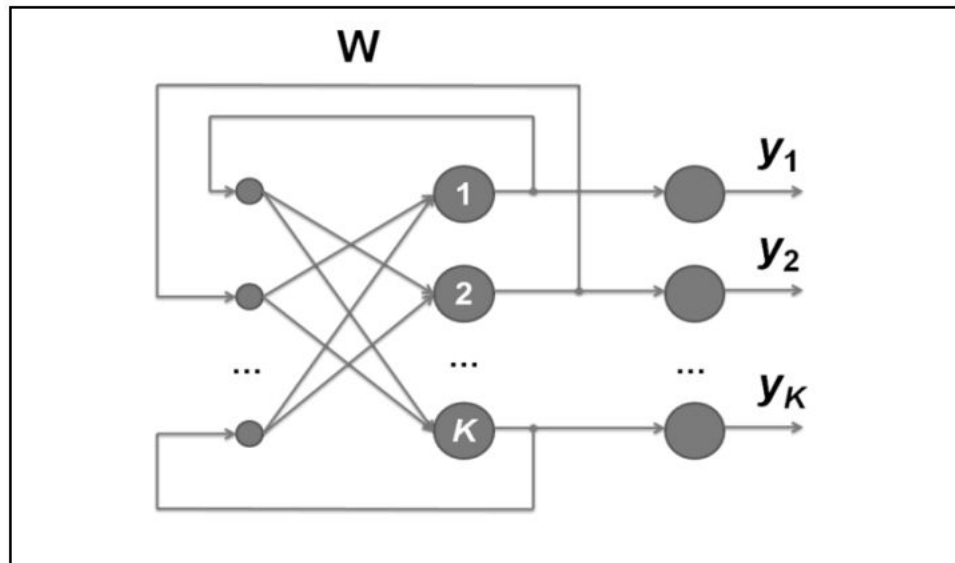


Теория



Теория

Нейронная сети Хопфилда на примере задачи распознавания образов



Рекуррентная нейронная сеть Хопфилда является автоассоциативной памятью



Теория



В синхронном режиме каждая эпоха с номером $n=1,2,\dots$ включает в себя следующие вычисления:

$$\text{net}_k^{(n)} = \sum_{\substack{j=1 \\ (j \neq k)}}^K w_{jk} y_j^{(n-1)}, \quad y_k^{(n)} = f(\text{net}_k^{(n)}), \quad k=1,2,\dots,K. \quad (7.1)$$

Здесь функция активации каждого нейрона

$$f(\text{net}_k^{(n)}) = \begin{cases} 1, & \text{net}_k^{(n)} > 0, \\ f(\text{net}_k^{(n-1)}), & \text{net}_k^{(n)} = 0, \\ -1, & \text{net}_k^{(n)} < 0. \end{cases}$$

В асинхронном режиме вместо (Н.1) имеем

$$\text{net}_k^{(n)} = \sum_{j=1}^{k-1} w_{jk} y_j^{(n)} + \sum_{j=k+1}^K w_{jk} y_j^{(n-1)}, \quad y_k^{(n)} = f(\text{net}_k^{(n)}), \quad k=1,2,\dots,K. \quad (7.2)$$



Рассмотрим пример



Собственно пример

Нарисуем входные матрицы-«числа»

«1»

-1	1	-1
1	1	-1
-1	1	-1
-1	1	-1
1	1	1

«2»

1	1	1
-1	-1	1
1	1	1
1	-1	-1
1	1	1

«3»

1	1	1
-1	-1	1
1	1	1
-1	-1	1
1	1	1

Запишем их в виде векторов

$$X^{(1)} = (-1, 1, -1, -1, 1, 1, 1, 1, 1, 1, -1, -1, -1, -1, 1),$$

$$X^{(2)} = (1, -1, 1, 1, 1, 1, -1, 1, -1, 1, 1, 1, 1, -1, 1),$$

$$X^{(3)} = (1, -1, 1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 1, 1).$$



Пример

В соответствии с формулой

$$W = \left(\sum_{l=1}^3 X^{(l)T} X^{(l)} \right)',$$

Рассчитаем матрицу весов:

$$W = \begin{pmatrix} 0 & -3 & 3 & 1 & 1 & 1 & -3 & 1 & -3 & 1 & 3 & 3 & 3 & 1 & 1 \\ -3 & 0 & -3 & -1 & -1 & -1 & 3 & -1 & 3 & -1 & -3 & -3 & -3 & -1 & -1 \\ 3 & -3 & 0 & 1 & 1 & 1 & -3 & 1 & -3 & 1 & 3 & 3 & 3 & 1 & 1 \\ 1 & -1 & 1 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 & 0 & 3 & -1 & 3 & -1 & 3 & 1 & 1 & 1 & -1 & 3 \\ 1 & -1 & 1 & -1 & 3 & 0 & -1 & 3 & -1 & 3 & 1 & 1 & 1 & -1 & 3 \\ -3 & 3 & -3 & -1 & -1 & -1 & 0 & -1 & 3 & -1 & -3 & -3 & -3 & -1 & -1 \\ 1 & -1 & 1 & -1 & 3 & 3 & -1 & 0 & -1 & 3 & 1 & 1 & 1 & -1 & 3 \\ -3 & 3 & -3 & -1 & -1 & -1 & 3 & -1 & 0 & -1 & -3 & -3 & -3 & -1 & -1 \\ 1 & -1 & 1 & -1 & 3 & 3 & -1 & 3 & -1 & 0 & 1 & 1 & 1 & -1 & 3 \\ 3 & -3 & 3 & 1 & 1 & 1 & -3 & 1 & -3 & 1 & 0 & 3 & 3 & 1 & 1 \\ 3 & -3 & 3 & 1 & 1 & 1 & -3 & 1 & -3 & 1 & 3 & 0 & 3 & 1 & 1 \\ 3 & -3 & 3 & 1 & 1 & 1 & -3 & 1 & -3 & 1 & 3 & 3 & 0 & 1 & 1 \\ 1 & -1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 0 & -1 \\ 1 & -1 & 1 & -1 & 3 & 3 & -1 & 3 & -1 & 3 & 1 & 1 & 1 & -1 & 0 \end{pmatrix}$$



' – операция
обнуления главной
диагонали

Итог



Поочередно подадим на вход все вектора:

$$f(X^{(l)}W) = X^{(l)}, \quad l = 1, 2, 3.$$

Убеждаемся что НС распознает знакомые ей образы

Подадим на вход искаженный образ «1»

$$\tilde{X}^{(1)} = (-1, -1, 1, -1, 1, 1, 1, 1, 1, 1, -1, -1, -1, -1, 1).$$

НС верно определила исходный образ

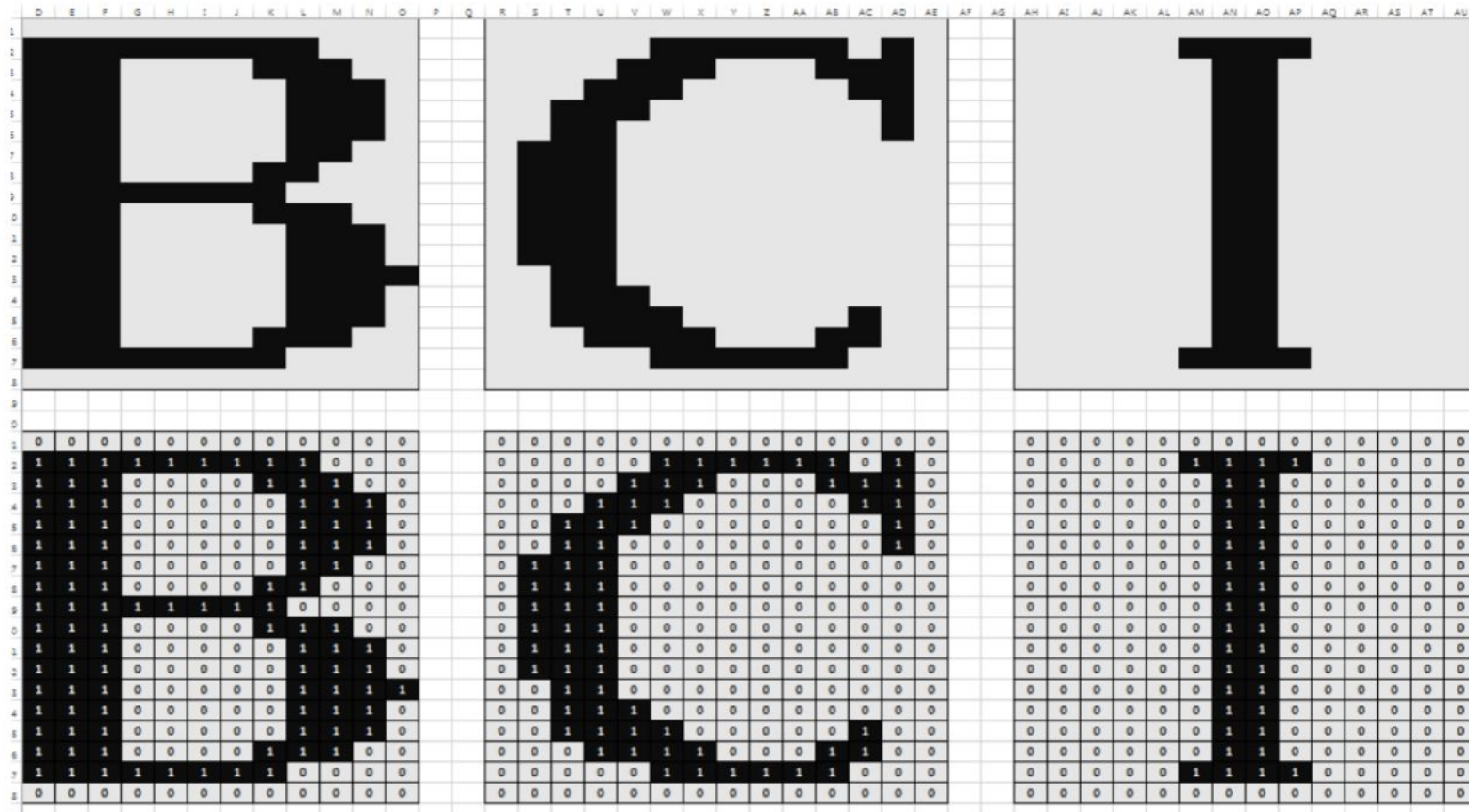
$$f(\tilde{X}^{(1)}W) = X^{(1)}.$$

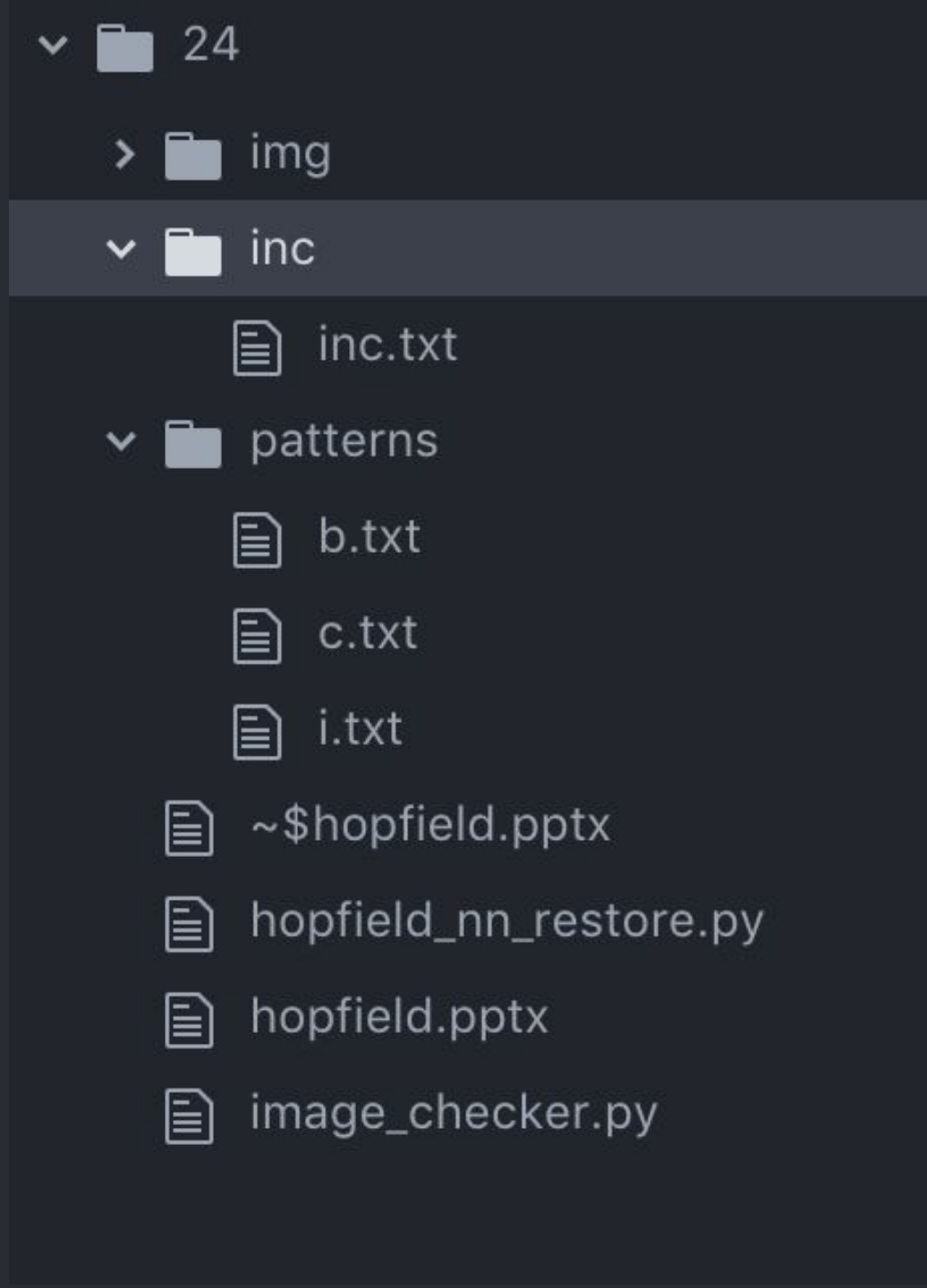


Создадим свою Нейросеть



Входные данные





Структура проекта



Код программы



```
if __name__ == '__main__':
```

```
    import numpy as np
    from functools import reduce
    from os import listdir
    from os.path import isfile, join
```

Go!

```
    # Достаем все файлы из папки patterns
    names = [f for f in listdir('patterns') if isfile(join('patterns', f))]
    print(f'\n* Список файлов в директории patterns: \n{names}\n')
```

Go!

```
    # Создаем массив входных векторов
    input_vectors, len_line = read_in_file(names, 'patterns/')
    print_pattern(input_vectors, names, len_line)
```

Go!

```
    # Создаем матрицу весовых коэффициентов
    W = calc_W(input_vectors)
    print(f'\n* Матрица весовых коэффициентов: {W}')
```

Go!

```
    # Подадим искаженный образ
    inc_name = [f for f in listdir('inc') if isfile(join('inc', f))]
    inc_vector, len_line = read_in_file(inc_name, 'inc/')
    print_pattern(inc_vector, inc_name, len_line)
```

Go!

```
    # Распознаем образ
    testing(inc_vector[0], W, input_vectors, len_line)
```



Вывод на экран

```
# Достаем все файлы из папки patterns  
names = [f for f in listdir('patterns') if isfile(join('patterns', f))]  
print(f'\n* Список файлов в директории patterns: \n{names}\n')
```



```
* Список файлов в директории patterns:  
['i.txt', 'c.txt', 'b.txt']
```



Home




```
def read_in_file(names, dir):
```

```
# Создаем пустой массив
```

```
input_vector = []
```

```
for name in names:
```

```
# Создаем массив строк из файла до переноса
```

```
path = dir + name
```

```
vector = list()
```

```
with open(path) as file_object:
```

```
    for line in file_object:
```

```
        line = line.rstrip('\n')
```

```
        vector.append(line)
```

```
# Превращаем полученный массив в строку
```

```
new_vector = reduce((lambda x, y: x + y), vector)
```

```
# Создаем массив нулей размера входной матрицы и заполняем
```

```
# в соответствии с заданными условиями
```

```
v = np.empty(len(new_vector))
```

```
for i in range(len(new_vector)):
```

```
    if new_vector[i] == '0':
```

```
        v[i] = -1
```

```
    else:
```

```
        v[i] = 1
```

```
input_vector.append(v)
```

```
# Добавляем значения каждой матрицы в общий массив
```

```
input_vector = np.asarray(input_vector)
```

```
return input_vector, len(line)
```

Код программы



Код программы

```
# Функция для красивого вывода образ на экран
def print_pattern(pattern, name, ll):

    for i, elem in enumerate(pattern):
        print(f"\npattern: {name[i].replace('.txt', '').upper()}")
        img = ''
        for num, j in enumerate(elem):
            if num % ll == 0:
                img += '\n'
            if j == -1.:
                img += '0'
            elif j == 1.:
                img += '1'
        print(img)
```



Вывод на экран

```
# Создаем массив входных веткоров
input_vectors, len_line = read_in_file(names, 'patterns/')
print_pattern(input_vectors, names, len_line)
```



Home



pattern: I

```
00000000000000
00000111100000
00000011000000
00000011000000
00000011000000
00000011000000
00000011000000
00000011000000
00000011000000
00000011000000
00000011000000
00000011000000
00000011000000
00000011000000
00000011000000
00000011000000
00000011000000
00000111100000
00000000000000
```

pattern: C

```
00000000000000
0000011111010
00001110001110
00011100000110
00111000000110
00111000000010
00110000000010
01110000000000
01110000000000
01110000000000
01110000000000
01110000000000
01110000000000
01110000000000
01110000000000
00110000000000
00111000000000
00111100000100
00011110001100
0000011111000
00000000000000
```

pattern: B

```
00000000000000
11111111111000
00111000011100
00111000001110
00111000001110
00111000001110
00111000001100
00111000011000
00111111110000
00111000011100
00111000001110
00111000001110
00111000001111
00111000001110
00111000001110
00111000011100
00111000011100
11111111110000
00000000000000
```



Код программы

```
def calc_W(input_vector):  
  
    n = len(input_vector[0])  
    m = len(input_vector)  
    W = np.zeros((n, n))  
  
    for l in range(m):  
        W += input_vector[l]*transpose(input_vector[l])  
  
    for i in range(n):  
        for j in range(n):  
            if i == j:  
                W[i, j] = 0  
  
    return W
```



Вывод на экран

```
# Создаем матрицесовых коэффициентов
W = calc_W(input_vectors)
print(f'\n* Матрица весовых коэффициентов: {W}')
```



```
* Матрица весовых коэффициентов: [[0. 3. 3. ... 3. 3. 3.]
[3. 0. 3. ... 3. 3. 3.]
[3. 3. 0. ... 3. 3. 3.]
...
[3. 3. 3. ... 0. 3. 3.]
[3. 3. 3. ... 3. 0. 3.]
[3. 3. 3. ... 3. 3. 0.]]
```



[Home](#)



Вывод на экран

```
# Подадим искаженный образ  
inc_name = [f for f in listdir('inc') if isfile(join('inc', f))]  
inc_vector, len_line = read_in_file(inc_name, 'inc/')  
print_pattern(inc_vector, inc_name, len_line)
```



pattern: INC

```
00000000000000  
00111100011000  
00111000011100  
00001000001110  
10111000001000  
00111000001110  
00111011001100  
00111000011000  
00000001110000  
00111000011100  
00111011001100  
00111110001110  
00111000001111  
00111011001000  
00111000001110  
00111000011100  
11111001110000  
00000000000000
```



[Home](#)



```
def testing(inc_vector, W, input_vector, ll):
```

```
    n = len(inc_vector)
    net = np.empty(n)
```

```
    for k in range(n):
        for j in range(n):
            net[k] += W[j, k]*inc_vector[j]
```

```
        if net[k] > 0: net[k] = 1
        if net[k] < 0: net[k] = -1
        if net[k] == 0: net[k] = inc_vector[k]
```

```
    # Тестирование на соответствие известным векторам
```

```
    m = len(input_vector)
    num = 0
    for i in range(m):
        if np.array_equal(input_vector[i], net) == True:
            num = i + 1
```

```
    if num == 1: word = 'I'
    elif num == 2: word = 'C'
    elif num == 3: word = 'B'
```

```
    if num != 0:
        print_pattern([net], ['restored'], ll)
        print('\nВосстановленному образцу соответствует образец: "', word, "'\n')
    else:
        print('\nВосстановить образ не удалось!')
```

Код программы



Вывод на экран

Распознаем образ

```
testing(inc_vector[0], W, input_vectors, len_line)
```



```
pattern: RESTORED
```

```
0000000000000000
111111111111000
00111000011100
00111000001110
00111000001110
00111000001110
00111000001100
00111000011000
00111111110000
00111000011100
00111000001110
00111000001110
00111000001111
00111000001110
00111000001110
00111000001110
00111000011100
11111111110000
0000000000000000
```

Восстановленному образцу соответствует образец: " В "



[Home](#)





Библиотека Pillow



Код программы

```
from os import listdir
from os.path import isfile, join
from PIL import Image
from hopfield_nn_restore import read_in_file

names = [f for f in listdir('patterns') if isfile(join('patterns', f))]

input_vector = []
for name in names:

    path = 'patterns/' + name
    vector = list()
    with open(path) as file_object:
        for line in file_object:
            line = line.rstrip('\n')
            vector.append(line)
    input_vector.append(vector)

for i, elem in enumerate(input_vector):
    background = Image.new('RGBA', (len(elem[0]), len(vector)), (255, 255, 255, 255))
    width, height = background.size

    for x in range(height):
        for y in range(width):

            if elem[x][y] == '1':
                img = Image.new('RGB', (1, 1), color = (0, 0, 0))
                background.paste(img, (y, x))
            if elem[x][y] == '0':
                img = Image.new('RGB', (1, 1), color = (255, 255, 255))
                background.paste(img, (y, x))

background.save(f'img/img_{name[i]}.png')
```





Результаты выполнения



С

И

В

И

Потому что
18x14 px
~\ (ツ) /~



9 | шакалов из 10



Код и презентация
доступны по ссылке:

<https://github.com/SITIS-project/24>



Спасибо за внимание!