Цель работы

Обучить нейронную сеть Хопфилда распознавать образ.

Вариант 11

В данном варианте образами для распознавания выступали три дорожных знака. Были выбраны следующие знаки: знак парковки, знак "Прочие опасности" и знак "Въезд запрещён".

Ход работы

Был произведён импорт всех необходимых библиотек. Результат представлен на рисунке 1.

```
[ ]  import matplotlib.pyplot as plt
     import numpy as np
     from random import random, randint, randrange, choice
```

Рисунок 1 - Импорт всех библиотек

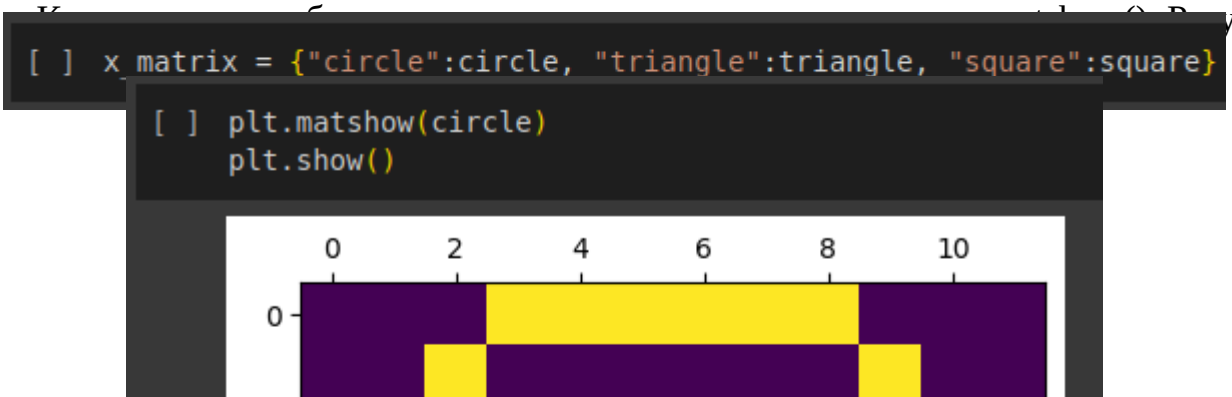
Были сгенерированы 3 матрицы с эталонными образами. Был использован метод np.array(). Результат представлен на рисунке 2.

```
[ ]  circle = np.array([
       [-1, -1, -1, 1, 1, 1, 1, 1, 1, -1, -1, -1],
       [-1, -1, 1, -1, -1, -1, -1, -1, -1, 1, -1, -1],
       [-1, 1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1],
       [1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1],
       [1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1],
       [1, -1, 1, 1, 1, 1, 1, 1, 1, 1, -1, 1],
       [1, -1, 1, 1, 1, 1, 1, 1, 1, 1, -1, 1],
       [1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1],
       [1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1],
       [-1, 1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1],
       [-1, -1, 1, -1, -1, -1, -1, -1, -1, 1, -1, -1],
       [-1, -1, -1, 1, 1, 1, 1, 1, 1, -1, -1, -1]
     ])
```

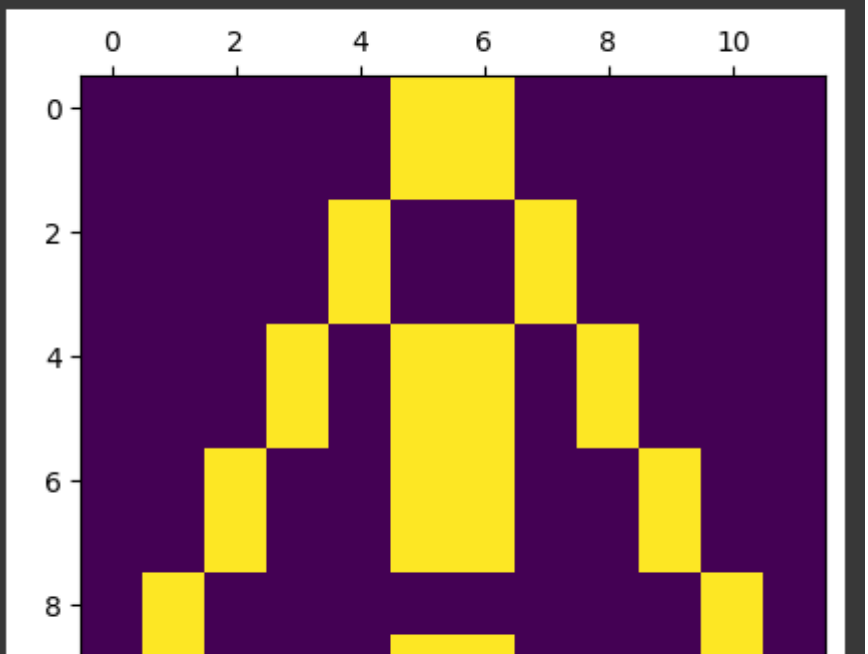
```
[ ]  triangle = np.array([
       [-1, -1, -1, -1, -1, 1, 1, -1, -1, -1, -1, -1],
       [-1, -1, -1, -1, -1, 1, 1, -1, -1, -1, -1, -1],
       [-1, -1, -1, -1, 1, -1, -1, 1, -1, -1, -1, -1],
       [-1, -1, -1, -1, 1, -1, -1, 1, -1, -1, -1, -1],
       [-1, -1, -1, 1, -1, 1, 1, -1, 1, -1, -1, -1],
       [-1, -1, -1, 1, -1, 1, 1, -1, 1, -1, -1, -1],
       [-1, -1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1],
       [-1, -1, 1, -1, -1, 1, 1, -1, -1, 1, -1, -1],
```

Эталонные образы представляют из себя изображения дорожных знаков 12/12 пикселей. В матричной форме данные образы представляют из себя вложенные списки с значениями 1 (наличие пикселя) или -1 (отсутствие пикселя).

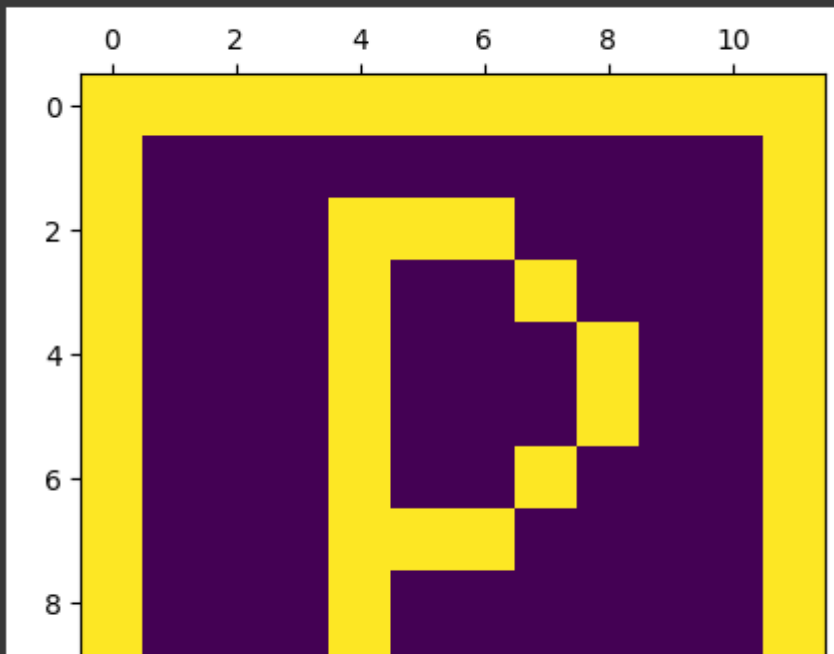
Для удобства матрицы были записаны в словарь, где ключи это названия знаков, а значения – матрицы. Результат представлен на рисунке 3.



```
[ ]  plt.matshow(triangle)
     plt.show()
```



```
[ ]  plt.matshow(square)
     plt.show()
```



Жёлтым цветом отображаются значения +1, а фиолетовым -1 из исходной матрицы.

Матрицы были переписаны в векторы. Результат представлен на рисунке 7.

```
def from_matrix_to_arrow(matrix):
  return np.copy(matrix).flatten()

circle_arrow = from_matrix_to_arrow(circle)
triangle_arrow = from_matrix_to_arrow(triangle)
square_arrow = from_matrix_to_arrow(square)
```

Рисунок 7 - Эталоны в виде векторов

Векторы, как и матрицы, для удобства были записаны в словарь. Результат представлен на рисунке 8.

```
x_arrows = {"circle":circle_arrow, "triangle":triangle_arrow, "square":square_arrow}
```

Рисунок 8 - Словарь векторов

Была создана вспомогательная функция для поиска количества пикселей, которые следует зашумить. На вход она принимает матрицу образа и процент зашумления. Результат представлен на рисунке 9.

```
def find_number_of_pixels_to_make_noisy(image, percent):
  total = 0
  for i in image:
    total += len(i)
  total *= percent/100
  return int(total)
```

Рисунок 9 - Функция нахождения количества зашумлённых пикселей

Была создана функция, которая получает на вход матрицу образа и процент зашумления. В процессе она обращается к функции find_number_of_pixels_to_make_noisy(), для поиска количества пикселей для изменения. Дальше функция случайным образом выбирает строчку и столбец матрицы и меняет значение на противоположное. Если значение уже было изменено, то выбирается новое случайное значение. Это сделано затем, чтобы более наглядно работало зашумление с разным процентом. Результат представлен на рисунке 10.

```
def make_some_noise(image, percent):
  exceptions = []
  noisy_image = np.copy(image)
  noisy_pixels = find_number_of_pixels_to_make_noisy(image, percent)
  for i in range(noisy_pixels):
    j = randrange(len(image))
    k = randrange(len(image[j]))
    while [j, k] in exceptions:
      j = randrange(len(image))
      k = randrange(len(image[j]))
    noisy_image[j][k] *= -1
    exceptions.append([j , k])
  noisy_arrow_image = noisy_image.flatten()
  return noisy_arrow_image
```

Рисунок 10 - Функция зашумления

Была создана функция, которая получает на вход образ и количество генерируемых изображений с шумами и дальше, используя предыдущие функции, создаёт список векторов с шумами. Результат представлен на рисунке 11.

```
def make_more_noise(x_arrows = x_arrows, x_matrix = x_matrix, N = 100):
  noisy_arrows = []
  for i in range(1, N+1):
    random_number = i
    if N != 100:
      random_number = randint(1, 100)
    noisy_image_label = choice(list(x_arrows.keys()))
    noisy_arrows.append({"Вектор с шумами":make_some_noise(x_matrix[noisy_image_label], random_number),"Вектор после предсказания":'', "Изображение без шумов":noisy_image_label, "Предсказанное изображение":'', "Пр
  return noisy_arrows
```

Рисунок 11 - Функция массового зашумления

Был создан набор данных с шумами. Он состоит из 100 элементов. Результат представлен на рисунке 12.

```
[ ]  noisy_arrows = make_more_noise()

[ ]  len(noisy_arrows)

     100
```

Была создана функция перевода вектора в матрицу с возможностью отрисовки результата. Результат представлен на рисунке 13.

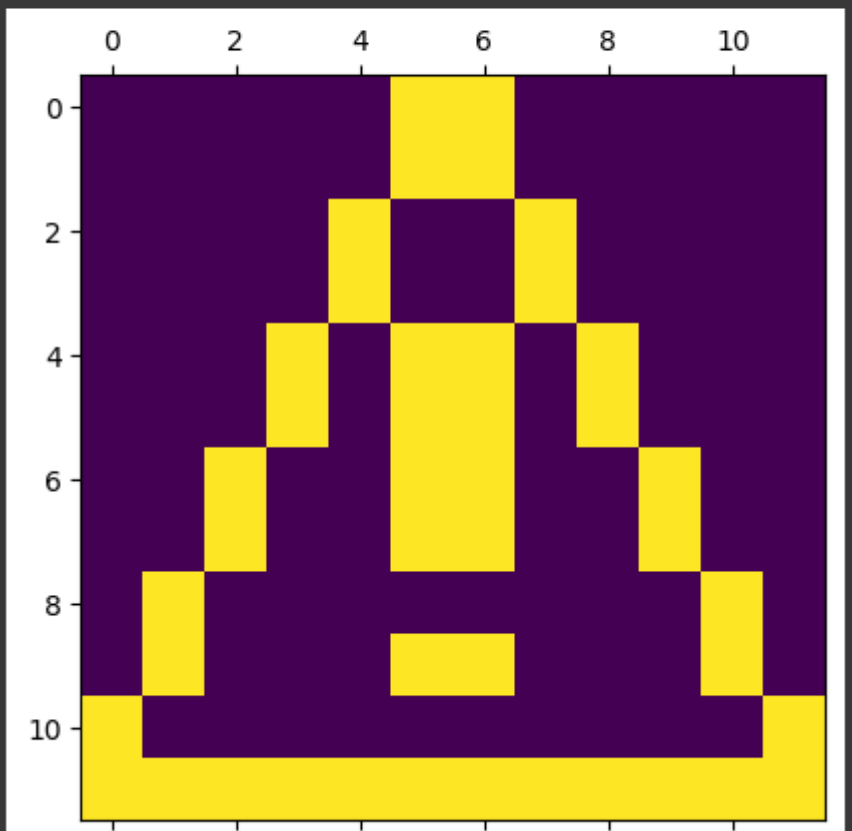
```
def from_arrow_to_matrix(arrow, draw = False):
  matrix = np.copy(arrow).reshape(-1, 12)
  if draw:
    draw_matrix(matrix)
    return
  return matrix
```

Рисунок 13 - Функция перевода вектора в матрицу

Была создана вспомогательная функция отрисовки образа, а также отрисованы эталонный и зашумлённый образы. Результат представлен на рисунках 14 - 15.

```
[ ]  def draw_matrix(matrix):
       plt.matshow(matrix)
       plt.show()
```

```
[ ]  from_arrow_to_matrix(x_arrows['triangle'], True)
```



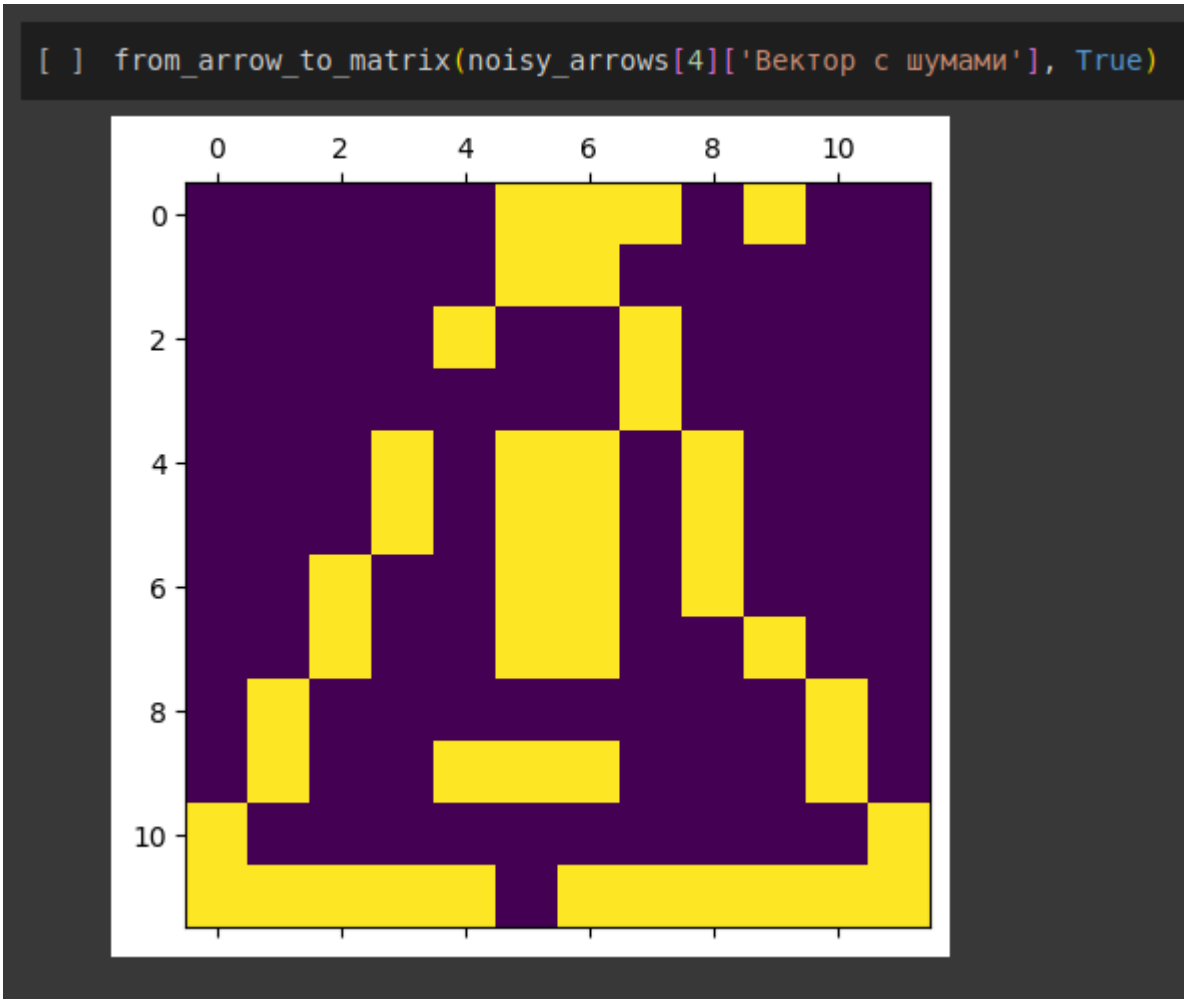


Рисунок 15 - Зашумлённый образ

Как видно из примера выше, образ с шумами отличается от эталонного образа.

Была создана функция для нахождения матрицы весовых коэффициентов. Данная функция создаёт копию эталонной матрицы и создаёт её транспонированную версию. Функция возвращает произведение полученных матриц. Результат представлен на рисунке 16.

```
[ ]  def find_value_matrix(image):
       array = np.copy(image)
       array_T = np.transpose(array)
       return np.outer(array, array_T)
```

Рисунок 16 - Функция нахождения матрицы весовых коэффициентов

Каждый эталонный вектор был преобразован в матрицу весовых коэффициентов. В данном случае их три - по количеству образов. Результат представлен на рисунке 17.

```
circle_dot = find_value_matrix(circle_arrow)
square_dot = find_value_matrix(square_arrow)
triangle_dot = find_value_matrix(triangle_arrow)
```

Рисунок 17 - Матрицы весовых коэффициентов

Матрицы были просуммированы. Была найдена длина вектора. Результат представлен на рисунке 18.

```
W = circle_dot + square_dot + triangle_dot

N = len(circle_arrow)
```

Рисунок 18 - Суммирование матриц и нахождение длины вектора

Матрица была поделена на длину вектора. Результат представлен на рисунке 19.

```
[ ]  W = W / N
```

Рисунок 19 - Суммарная матрица, поделённая на длину вектора

Была создана функция для обнуления диагональных элементов. Данная функция создаёт копию входной матрицы и заменяет диагональные ячейки на ноль. Результат представлен на рисунке 20.

```
def make_diagonal_zero(value_matrix):
  zero_matrix = np.copy(value_matrix)
  for i in range(len(value_matrix)):
      zero_matrix[i][i] = 0
  return zero_matrix
```

Рисунок 20 - Функция обнуления диагональных ячеек

Произошло обнуление диагональных элементов суммарной матрицы. Результат представлен на рисунке 21.

```
W = make_diagonal_zero(W)
```

Рисунок 21 - Обнуление диагональных ячеек

Сеть была обучена.

Была создана функция, которая меняет отрицательные числа на -1, а неотрицатльные на 1. Результат представлен на рисунке 22.

```
def make_plus_or_minus(array):
  for i in range(len(array)):
    if array[i] < 0:
      array[i] = -1
    elif array[i] >= 0:
      array[i] = 1
```

Рисунок 22 - Функция замены чисел

Была создана функция, которая находит и вносит словарь зашумлённых изображений значение предсказанного образа. Если совпадение, то будет написан незашумлённый образ, а если ошибка, то фраза "false prediction". Результат представлен на рисунке 23.

```
def get_prediction_image_origin(x_arrows = x_arrows, noisy_arrows = noisy_arrows):
  for j in noisy_arrows:
    for i in x_arrows:
      if np.array_equal(j['Вектор после предсказания'], x_arrows[i]):
        j['Предсказанное изображение'] = i
        break
      else:
        j['Предсказанное изображение'] = 'false prediction'
```

Рисунок 23 - Функция нахождения исходного образа

Была создана функция предсказания, используя предыдущие функции. Функция проходит по всем зашумлённым векторам и умножает их на суммарную матрицу W, затем заменяет значения меньше нуля на -1, а неотрицательные - на 1. Результат представлен на рисунке 24.

```
def make_prediction(noisy_arrows = noisy_arrows, W = W, x_arrows = x_arrows):
  for i in noisy_arrows:
    y = i['Вектор с шумами']
    y = np.dot(W, y)
    make_plus_or_minus(y)
    i['Вектор после предсказания'] = y
  get_prediction_image_origin(x_arrows, noisy_arrows)
  return
```

Рисунок 24 - Функция предсказания

Было произведено предсказание для всех зашумлённых векторов. Результат представлен на рисунке 25.

```
[ ]  make_prediction()
```

Рисунок 25 -
Предсказание

Для примера был выведен один зашумлённый образ. Результат представлен на рисунке 26.

```
[ ]  noisy_arrows[3]

    {'Вектор с шумами': array([ 1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1, -1, -1, -1, -1,
           -1, -1, -1, -1, -1, -1,  1,  1, -1, -1, -1,  1,  1,  1, -1, -1, -1,
           -1,  1,  1,  1, -1, -1,  1, -1, -1,  1, -1, -1, -1,  1,  1, -1, -1,
           -1,  1, -1, -1, -1,  1, -1, -1, -1,  1, -1, -1, -1,  1, -1, -1,  1,
            1, -1, -1,  1, -1, -1, -1, -1,  1, -1, -1,  1, -1, -1, -1,  1,  1,
           -1, -1, -1,  1,  1,  1, -1, -1, -1, -1,  1,  1, -1, -1, -1,  1, -1,
           -1, -1, -1, -1, -1,  1,  1, -1, -1, -1,  1, -1, -1, -1, -1, -1, -1,
            1,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  1,  1,  1,  1,  1,
            1,  1,  1,  1,  1,  1,  1,  1]),
     'Вектор после предсказания': array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
           -1., -1., -1., -1., -1., -1., -1., -1., -1., -1.,  1.,  1., -1.,
           -1., -1.,  1.,  1.,  1., -1., -1., -1., -1.,  1.,  1., -1., -1.,
           -1.,  1., -1., -1.,  1., -1., -1., -1.,  1.,  1., -1., -1., -1.,
            1., -1., -1., -1.,  1., -1., -1.,  1.,  1., -1., -1., -1.,  1.,
           -1., -1., -1.,  1., -1., -1.,  1.,  1., -1., -1., -1.,  1., -1.,
           -1.,  1., -1., -1., -1.,  1.,  1., -1., -1., -1.,  1.,  1.,  1.,
           -1., -1., -1., -1.,  1.,  1., -1., -1., -1.,  1., -1., -1., -1.,
           -1., -1., -1.,  1.,  1., -1., -1., -1.,  1., -1., -1., -1., -1.,
           -1., -1.,  1.,  1., -1., -1., -1., -1., -1., -1., -1., -1., -1.,
           -1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
            1.]),
     'Изображение без шумов': 'square',
     'Предсказанное изображение': 'square',
     'Процент зашумления': 4,
     'Ошибка': ''}
```

Рисунок 26 - Пример работы модели

Можно заметить, что все записи отображаются корректно. В данном случае модель верно предсказала образ, потому что "Изображение без шумов" совпадает с "Предсказанное изображение". В поле "Ошибка" отсутствует значение, потому что предсказание верно и ошибка равна нулю.

Была создана функция для нахождения количества отличающихся пикселей между предсказанным образом и эталонным. Результат представлен на рисунке 27.

```
def find_difference_between_prediction_and_reference(noisy_dict, x_arrows = x_arrows):
  counter = 0
  for i in range(len(noisy_dict['Вектор с шумами'])):
    if x_arrows[noisy_dict['Изображение без шумов']][i] != noisy_dict['Вектор после предсказания'][i]:
      counter += 1
  return counter
```

Рисунок 27 - Функция нахождения разницы предсказания и эталона

Было выведено эталонное изображение. Результат представлен на рисунке 28.

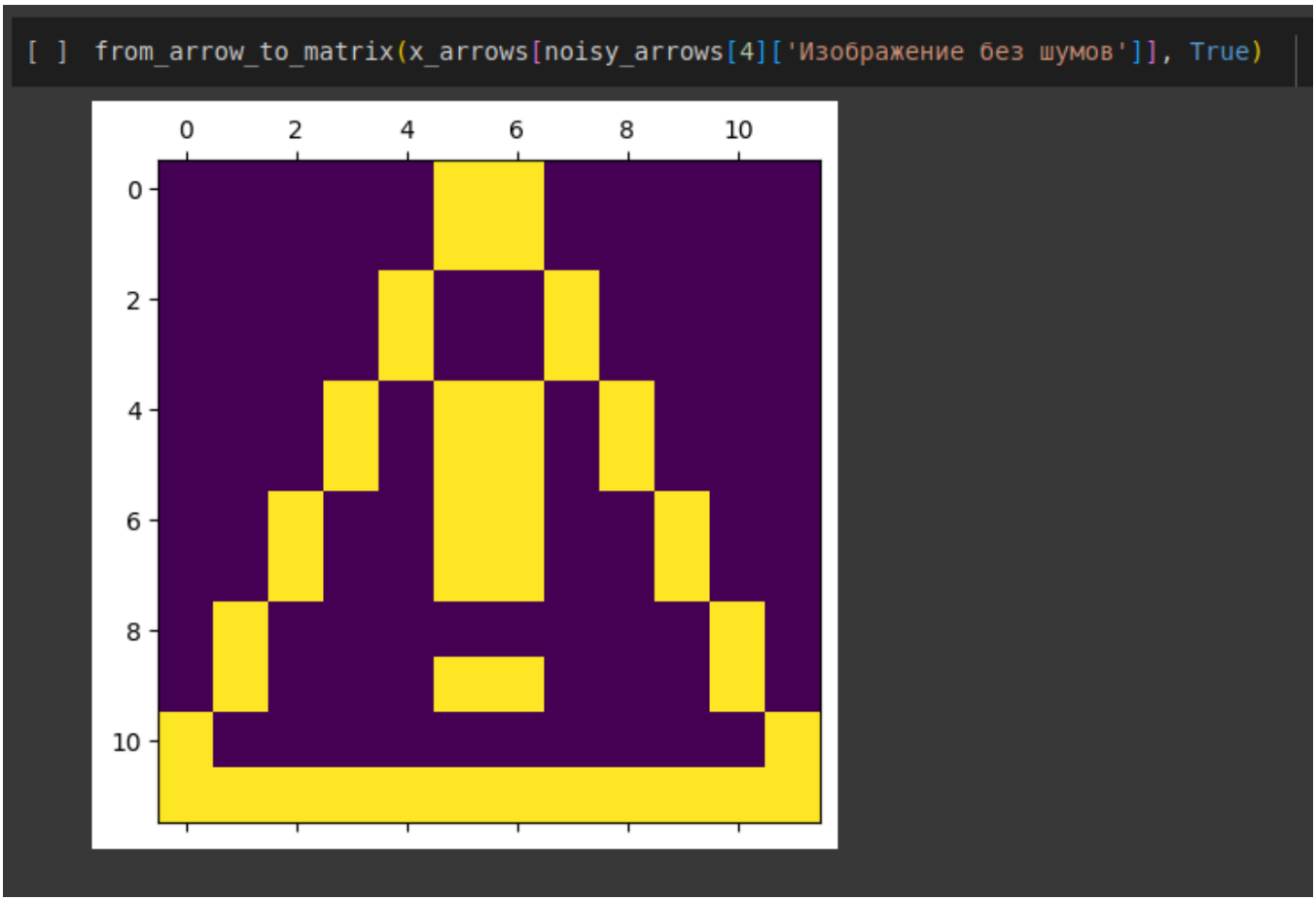
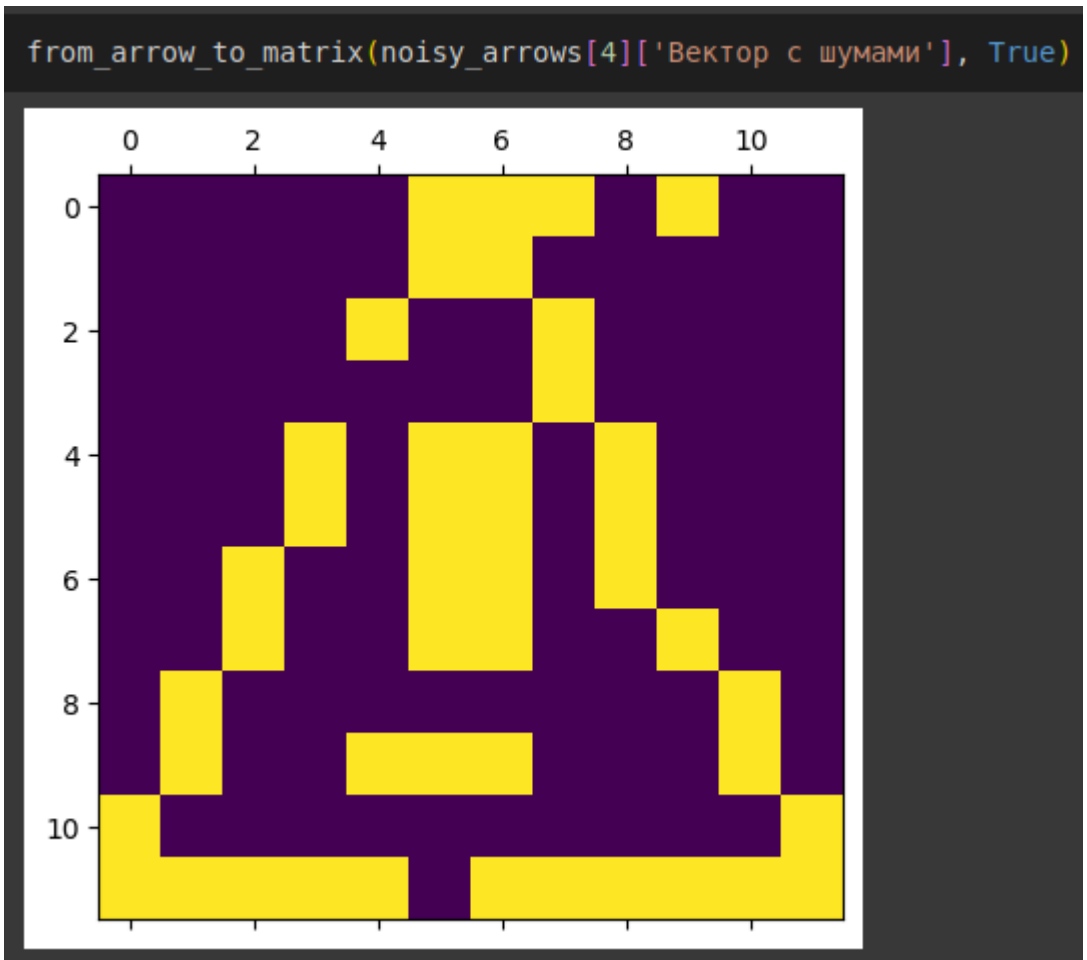


Рисунок 28 - Эталонное изображение

Было выведено зашумлённое изображение. Результат представлен на рисунке 29.



Было выведено предсказанное изображение. Результат представлен на рисунке 30.

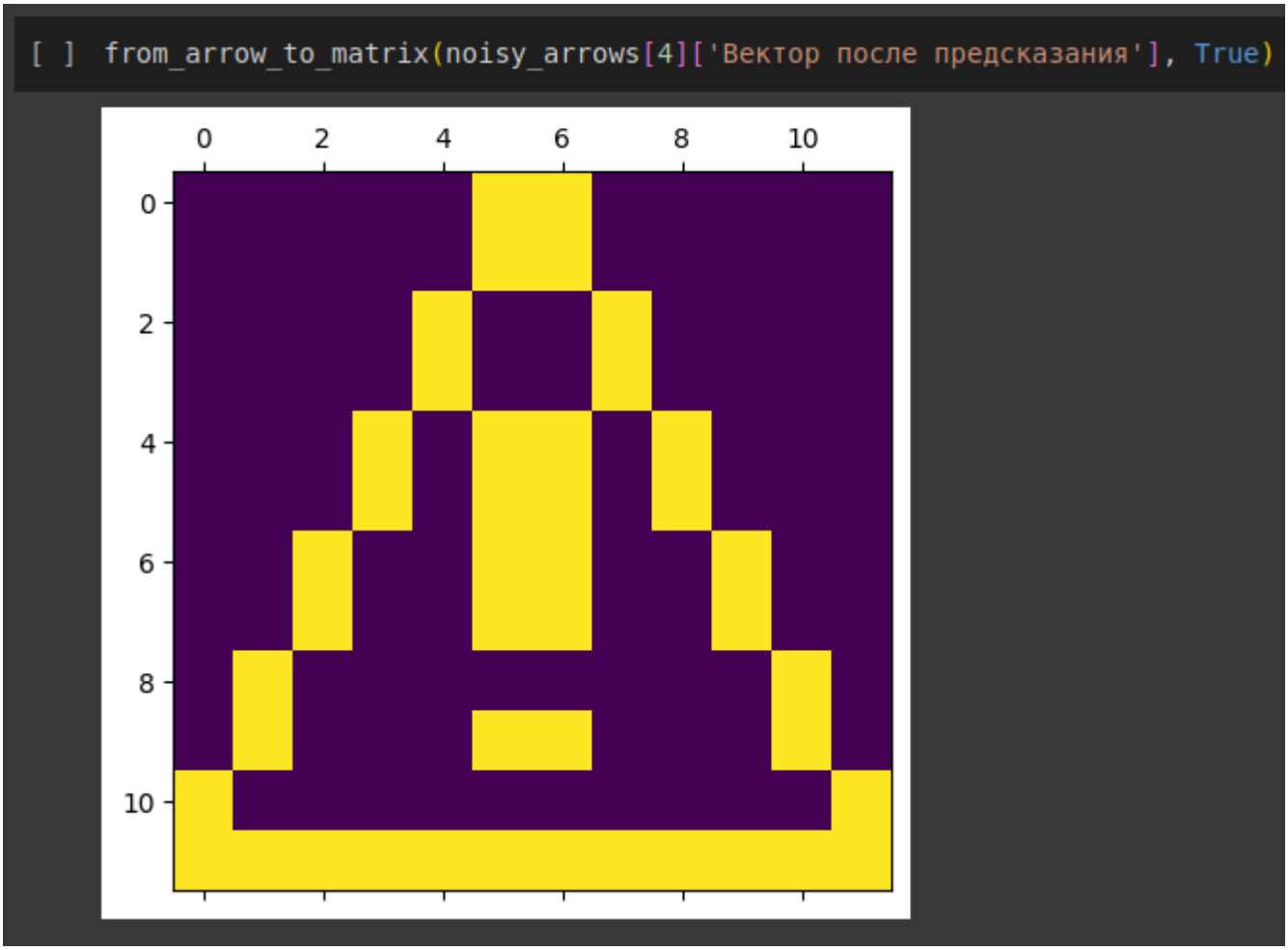


Рисунок 30 - Предсказанное изображение

Было посчитано количество отличающихся пикселей между результатом и эталонным изображением. Результат представлен на рисунке 31.

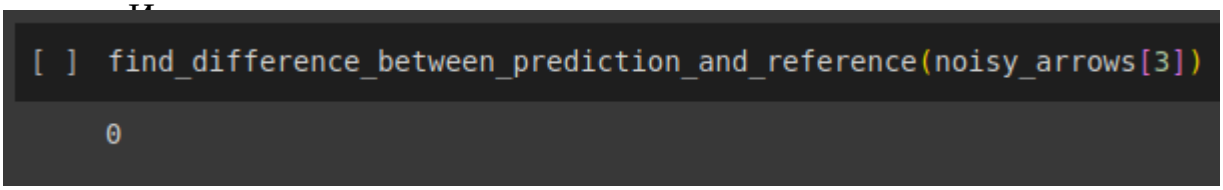


Рисунок 31 - Разница между предсказанием и эталоном

предсказание и точно угадала эталонное изображение.

Неправильно угаданные изображения были записаны в список. Результат представлен на рисунке 32.

```
[ ]  false_predict_array = []

[ ]  def print_false_predict(noisy_arrows = noisy_arrows, false_predict_array = false_predict_array, printer = True):
       for i in noisy_arrows:
         if i['Предсказанное изображение'] == 'false prediction':
           false_predict_array.append(i)
       if printer:
         for j in false_predict_array:
           from_arrow_to_matrix(j['Вектор с шумами'], True)
           print('Изображение до зашумления =',j['Изображение без шумов'])
           print('Процент зашумления =',j['Процент зашумления'])
           print('-' * 50)
```

Рисунок 32 - Функция наполнения списка ложных предсказаний и их вывода

Список неверно распознанных изображений был выведен на экран. Результат представлен на рисунке 33.

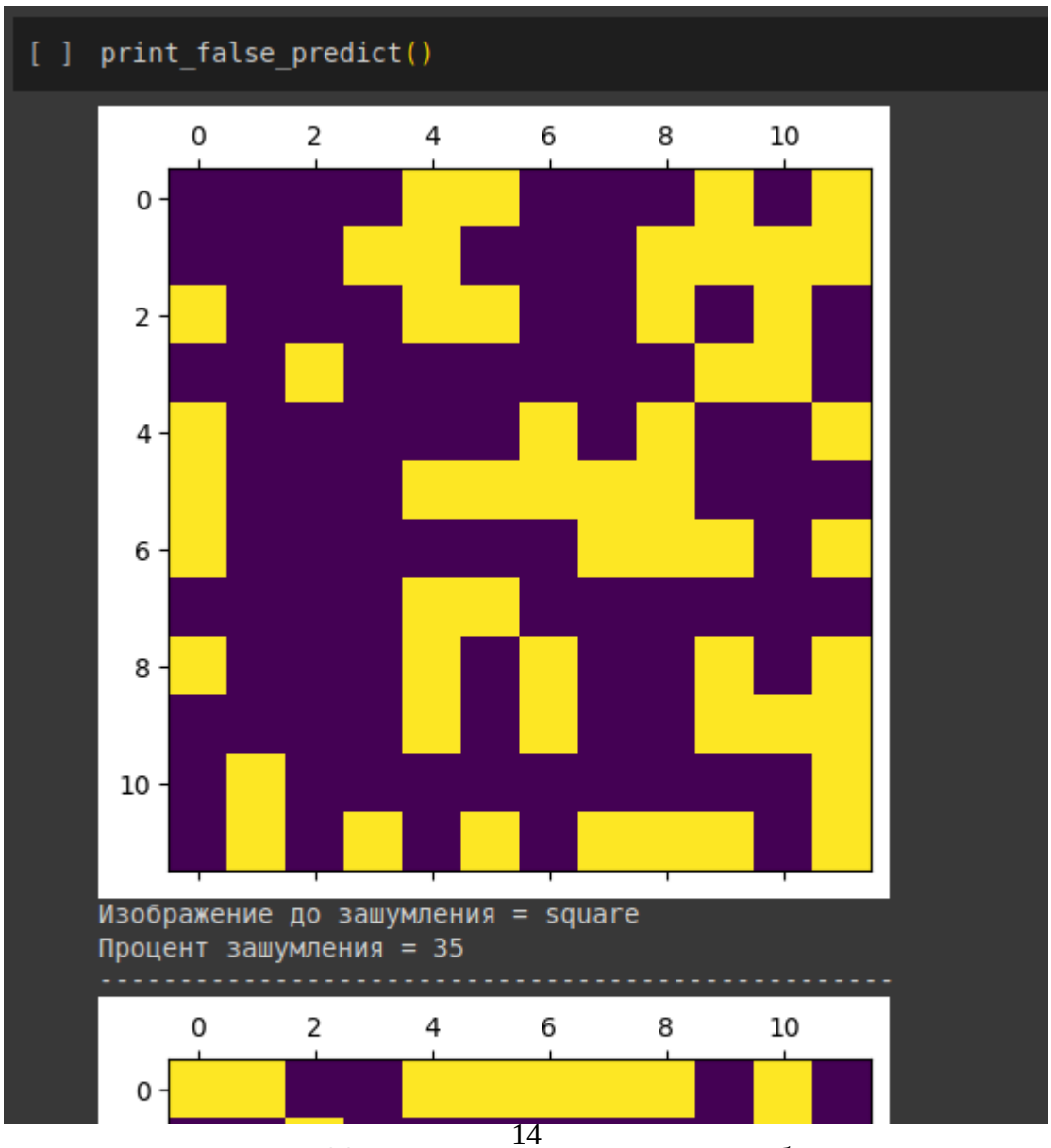


Рисунок 33 - Неверно предсказанные образы

Было выведено количество неверно предсказанных образов. Результат представлен на рисунке 34.

```
[ ]  len(false_predict_array )

     57
```

Рисунок 34 - Количество неверно предсказанных образов

Отношение ложных предсказаний к правильным при 100 генерациях чаще всего равно примерно 50%. В данном примере количество равно 57.

Была создана функция для нахождения разницы в пикселях между зашумлённым образом и эталонным образом. Результат представлен на рисунке 35.

```
[ ]  def find_difference_between_test_and_reference(noisy_dict, x_arrows = x_arrows):
       counter = 0
       for i in range(len(noisy_dict['Вектор с шумами'])):
         if x_arrows[noisy_dict['Изображение без шумов']][i] != noisy_dict['Вектор с шумами'][i]:
           counter += 1
       return counter
```

Рисунок 35 - Функция нахождения разницы зашумлённого образа и предсказанного

Для примера была вычислена ошибка восстановления у случайно выбранного предсказанного изображения. Результат представлен на рисунке 36.

```
difference_between_prediction_and_reference = find_difference_between_prediction_and_reference(noisy_arrows[61])
difference_between_test_and_reference = find_difference_between_test_and_reference(noisy_arrows[61])

difference_between_prediction_and_reference / difference_between_test_and_reference

1.6179775280898876
```

Рисунок 36 - Пример ошибки восстановления

Значение ошибки восстановление не равно нулю. Из этого можно сделать вывод, что пердсказание неверно и предсказанное значение отличается от эталонного.

Была создана функция для добавления ошибки восстановления как значение в словарях предсказанных векторов. Результат представлен на рисунке 37.

```
def find_and_add_errors(noisy_arrows = noisy_arrows, x_arrows = x_arrows):
  for i in noisy_arrows:
    i['Ошибка'] = find_difference_between_prediction_and_reference(i, x_arrows) / find_difference_between_test_and_reference(i, x_arrows)
```

Рисунок 37 - Функция добавления ошибки восстановления

Функция была применена. Затем было создано три списка. Список errors содержит ошибки предсказанных векторов, список percent - процент зашумления, список error_and_noise_percent объединяет их под один список. Результат представлен на рисунке 38.

```
error_and_noise_percent = []
errors = []
percent = []
for i in false_predict_array:
  errors.append(i['Ошибка'])
  percent.append(i['Процент зашумления'])
error_and_noise_percent.append(errors)
error_and_noise_percent.append(percent)
```

Рисунок 38 - Запись ошибки восстановления в список

Был выведен график зависимости ошибки восстановления от процента зашумления. Результат представлен на рисунке 39.

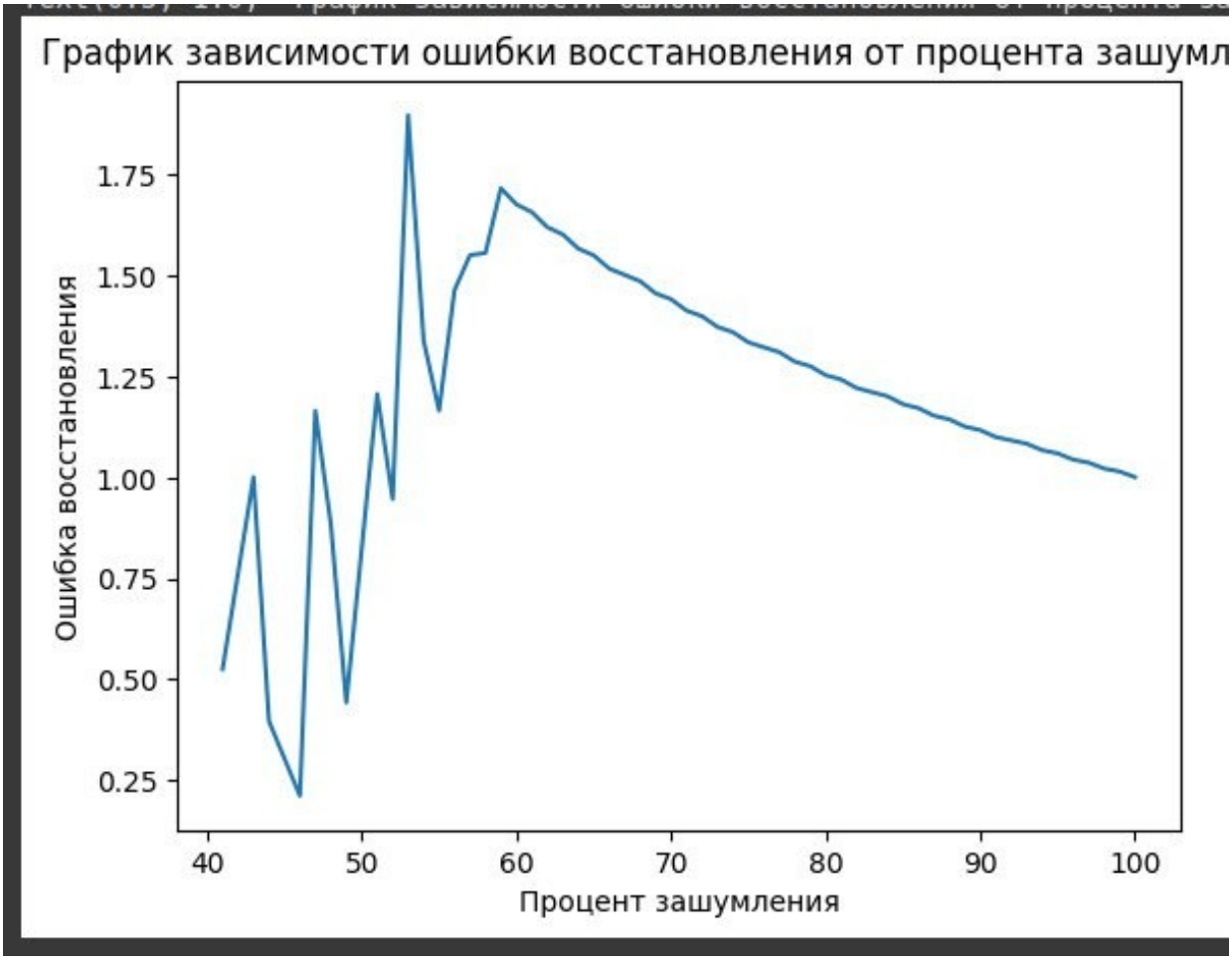


Рисунок 39 - График ошибки восстановления от процента шума

увеличивается и знаменатель дроби в ошибке восстановления. Этим и объясняется падение ошибки после 60 процентов. Числитель же ведёт себя непредсказуемо, поэтому можно заметить скачкообразное изменение графика. В среднем можно сказать, что чем меньше процент зашумления, тем меньше отличие предсказанной матрицы от эталонной и тем меньше ошибка восстановления

Ссылка на Google Colab

Ссылка

Вывод

В ходе выполнения лабораторной работы о распознавании образов с помощью нейронной сети Хопфилда были изучены основные принципы работы данной модели ассоциативной памяти. Были проведены эксперименты по обучению нейронной сети Хопфилда на наборе образов и последующему восстановлению их при предъявлении зашумленных или искаженных входных данных.

В результате работы было установлено, что нейронная сеть Хопфилда успешно справляется с задачей ассоциации и восстановления образов, даже при наличии шума или искажений. Однако были выявлены ограничения данной модели, такие как постепенное снижение верных предсказаний с увеличением процента зашумления, а также возможность предсказывать только изображения бинарных цветов.

Была графически отображена зависимость ошибки восстановления от процента шума. По графику можно заметить, что при увеличении процента шума, увеличивается и знаменатель дроби в ошибке восстановления. Этим и объясняется падение ошибки после 60 процентов. Числитель же ведёт себя непредсказуемо, поэтому можно заметить скачкообразное изменение графика. В среднем можно сказать, что чем меньше процент зашумления, тем меньше отличие предсказанной матрицы от эталонной и тем меньше ошибка восстановления

Таким образом, можно сделать вывод о том, что нейронная сеть Хопфилда является эффективным инструментом для решения задач распознавания образов и ассоциации, однако для более сложных задач может потребоваться применение

более сложных моделей нейронных сетей. В целом, лабораторная работа позволила более глубоко понять принципы работы нейронной сети Хопфилда и ее применение в практических задачах распознавания образов.

Сложностей во время выполнения работы не возникло.