Оглавление

[**ВВЕДЕНИЕ** 2](#_Toc123711279)

[**1.** **Теория** 3](#_Toc123711280)

[**1.1** **Формальная постановка задачи** 3](#_Toc123711281)

[**1.2.** **Составление алгоритма** 4](#_Toc123711282)

[**1.3.** **Выбор технологий для реализации.** 5](#_Toc123711283)

[**2.** **Практика** 6](#_Toc123711284)

[**2.1.** **Преобразование изображения в двоичный массив** 6](#_Toc123711285)

[**2.2.** **Создание эталонных образов** 8](#_Toc123711286)

[**2.3.** **Сравнение матрицы со множеством эталонных образов** 9](#_Toc123711287)

[**2.4.** **Распознавание рукописной буквы** 11](#_Toc123711288)

[**ЗАКЛЮЧЕНИЕ** 13](#_Toc123711289)

[**Список литературы** 14](#_Toc123711290)

# **ВВЕДЕНИЕ**

В интеллектуальных системах управления одним из базовых тем является распознавание образов. Образы могут быть визуальными, аудиальными, информационными и другими. Чаще других бывает необходимость в распознавании именно визуальных/зрительных образов. Для ИСУ проще даются задачи распознавания символов. При этом они сейчас довольно полезны и используются во многих сервисах. Современные переводчики умеют отличать текст при наведении на него камеры и сразу же отрисовать перевод.

Для оптического распознавания образов можно применить метод перебора вида объекта под различными углами, масштабами, смещениями и т. д. Для букв нужно перебирать шрифт, свойства шрифта и т. д. Второй подход — найти контур объекта и исследовать его свойства (связность, наличие углов и т. д.) Ещё один подход — использовать искусственные нейронные сети. Этот метод требует либо большого количества примеров задачи распознавания (с правильными ответами), либо специальной структуры нейронной сети, учитывающей специфику данной задачи.

# **Теория**

## **Формальная постановка задачи**

Распознавание образов — это отнесение исходных данных к определённому классу с помощью выделения существенных признаков, характеризующих эти данные, из общей массы данных.

При постановке задач распознавания стараются пользоваться математическим языком, стремясь — в отличие от теории искусственных нейронных сетей, где основой является получение результата путём эксперимента, — заменить эксперимент логическими рассуждениями и математическими доказательствами.

Классическая постановка задачи распознавания образов: Дано множество объектов. Относительно них необходимо провести классификацию. Множество представлено подмножествами, которые называются классами. Заданы: информация о классах, описание всего множества и описание информации об объекте, принадлежность которого к определённому классу неизвестна. Требуется по имеющейся информации о классах и описании объекта установить — к какому классу относится этот объект.

Наиболее часто в задачах распознавания образов рассматриваются монохромные изображения, что дает возможность рассматривать изображение как функцию на плоскости. Если рассмотреть точечное множество на плоскости T, где функция f(x,y) выражает в каждой точке изображения его характеристику — яркость, прозрачность, оптическую плотность, то такая функция есть формальная запись изображения.

Множество же всех возможных функций f(x,y) на плоскости T есть модель множества всех изображений X. Вводя понятие сходства между образами можно поставить задачу распознавания. Конкретный вид такой постановки сильно зависит от последующих этапов при распознавании в соответствии с тем или иным подходом.

* 1. **Составление алгоритма**

Алгоритм, который мы напишем, будет основан на вычислении эвклидовой близости. Близость будет по отношению к эталонным образам, их обозначим множеством A. Данные как в теории, так и в программном коде, будут представлять собой матрицу n×m.

(1)

После вычисления определяется тот элемент из всех сравнений, который дал наименьшую норму . Этот элемент, показавший минимум разницы с распознаваемым объектом X, будет являться наиболее похожим (близким) среди всех из множества эталонных образов A.

## **Выбор технологий для реализации.**

Python – язык программирования, удобный для написания скриптов, а также нейронных сетей. Инструмент имеет лёгкий синтаксис, позволяющий удобно работать со списками. На языке много готовых библиотек, особенно графической направленности, чем мы и будем пользоваться.

PIL (Python Imaging Library) – библиотека для работы с изображением. Мы же будем использовать его только чтобы открывать картинки, а также генерировать их.

TKinter – включен в стандартную сборку Python, нам нужен исключительно для интерфейса (GUI).

Skimage – данная библиотека позволит нам открывать и представлять растровые изображения в виде матрицы пикселей RGB. Также имеет функции применения различных фильтров на эти изображения, что позволит нам избавиться от лишних шумов и разных пятен на не чёрно-белых фотографиях, сделанных на камеру.

# **Практика**

## **Преобразование изображения в двоичный массив**

Изображения в цифровом виде как правило представляют собой матрицу из пикселей, которые возможно представить в виде RGB. Так, черный пиксель будет принимать значения (0, 0, 0), а белый в свою очередь являясь смесью всех трёх основных цветов – (255, 255, 255). Для нас эти два цвета будут двумя состояниями матрицы: 0 или 1. Разбиваем сетку на n×m части, далее смотрим на пиксели внутри каждой ячейки данной сетки: если буква касается текущей сетки (имеет достаточное количество черных пикселей), то ячейка в матрице получит значение 1, если нет – 0.

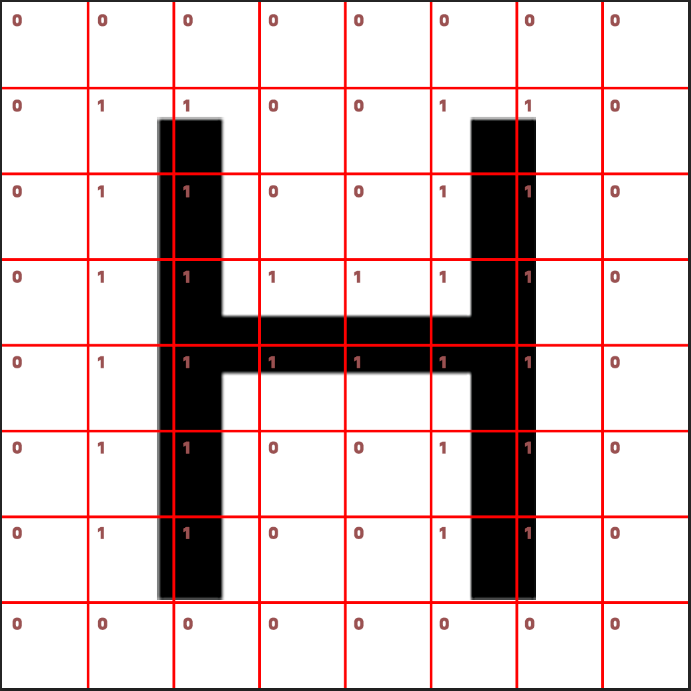


Рис. 1. Разделение буквы Н на двоичную матрицу.

В программном коде это реализовано в модуле arrayhelper.py вызовом функции parse\_image(image, div, pixel) – возвращает двумерную матрицу единиц и нулей, а параметрами являются: сканируемое изображение, количество сегментов на которое нужно его разделить и порог значений RGB, от которого можно считать пиксель условно-белым (пустым).

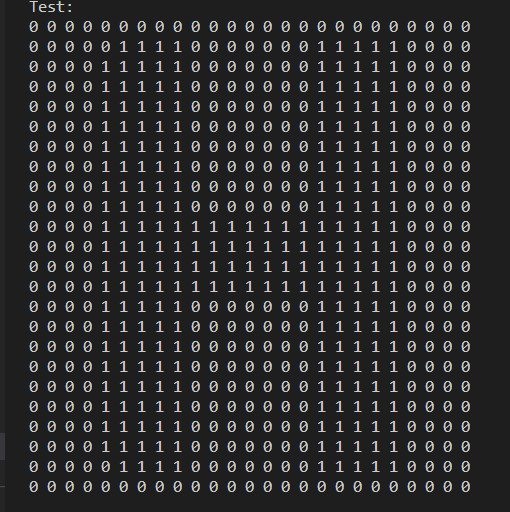
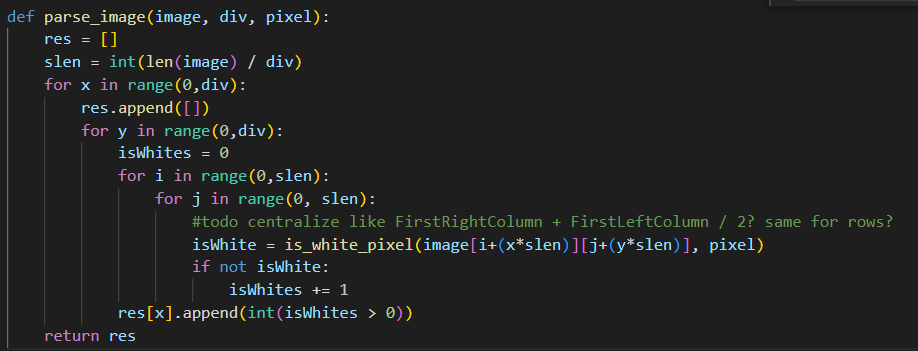
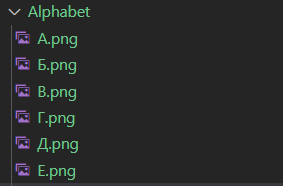


Рис. 2. Работа программы по преобразованию картинки буквы Н в двоичную матрицу.



## **Создание эталонных образов**

Было решено генерировать их автоматически. Написанный программный код содержится в модуле generator.py, там перечислены все буквы русского алфавита. На каждую букву создается 100x100 изображение в локальной папке «Alphabet/», шрифт calibri.



Сгенерированные таким образом изображения будут нашей базой данных, который так же будет считываться функцией parse\_image.

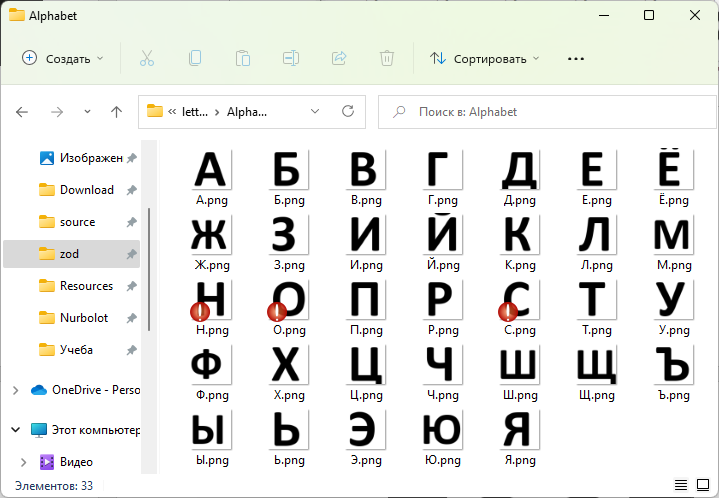


Рис. 3. Сгенерированная база эталонных образов.

## **Сравнение матрицы со множеством эталонных образов**

Запуск алгоритма, реализующего формулу (1), происходит в модуле tester.py. Программа просит ввести путь к файлу с изображением буквы.

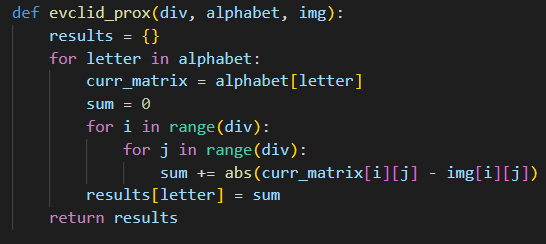


Рис. 2.4. Формула (1) в виде алгоритма на Python.

Можно заметить, что вместо квадрата разницы, мы использовали функцию модуль. Дело в том, что т.к. значения матрицы двоичные, нет разницы между возведением в чётную степень и использованием простой функции модули. Если выйдет 1 и 0, то значение останется неизменным, если же -1, то становится просто 1.

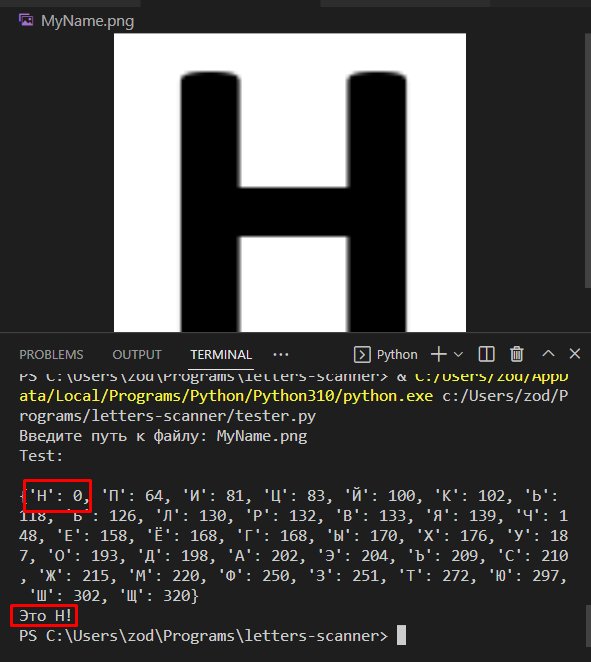
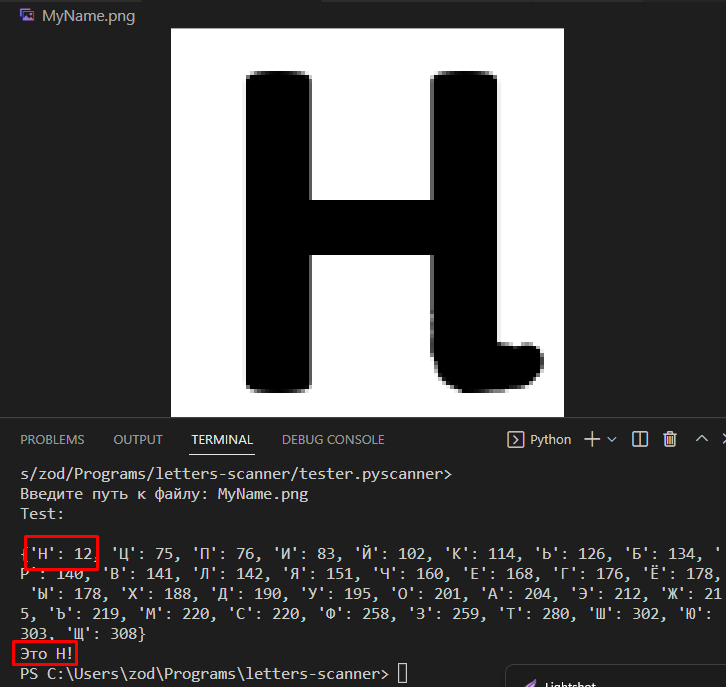


Рис. 4. Сгенерированная база эталонных образов.

Как показано на рис. 4, будет буква Н, что алгоритм нам и сообщает. С буквой «С» то же самое.

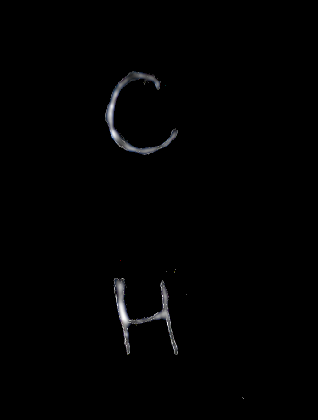
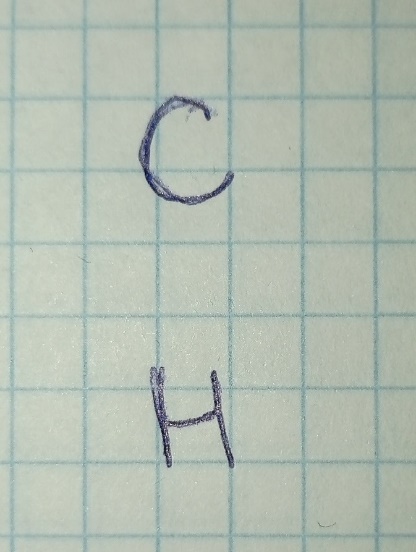




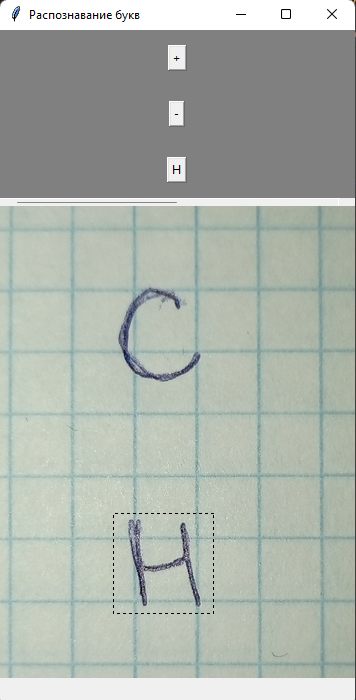
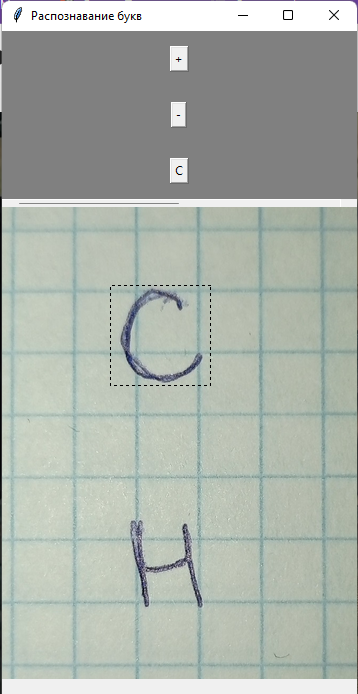
Когда мы добавили хвостик к букве Н, программа дала понять, что есть некоторое отхождение от эталонной модели, но изображение все ещё ближе всего к ней.

## **Распознавание рукописной буквы**

Этот пункт дается алгоритму не совсем удачно, однако примитивные буквы кое-как программа распознает. Для удаления лишнего фона на картинке (например, клетки бумаги) было использовано наложение фильтров.



Процессом занимается модуль main.py, запускающий GUI. Изображение можно отдалять и приближать (+/-). Надо навести мышкой на букву и нажать на третью кнопку, по умолчанию на нем написано “Here!”, тогда текст этой кнопки поменяется на ответ программы.



Немного про настраиваемые переменные. В модулях присутствуют переменные, которые можно изменять словно параметры системы. Такие параметры следовало упаковать в один конфиг, но из-за расхождения в контекстах они объявлены как обычные переменные. Вот некоторые из них:

* alphabet.py – div – integer. По умолчанию 25. Отвечает за то, насколько сегментов будет поделена картинка. Таким образом рисунки разных разрешений будут иметь одинаковые матрицы.
* mainy.py – DOFILTER – boolean. Фильтры нужны и применяются только тогда, когда изображение не на белом фоне чёрной краской. Таким образом, при true система будет применять их, а при false не станет. И значение при котором пиксель можно считать фоном будет меняться с 0 на 255 (белый фон или чёрный).
* main.py – sizee, extension, path. Sizee будет определять расстояние для сканирования (при наведении мышкой). Extension – расширение открываемого файла, дело в том что модуль использует промежуточное изображение, соответственно нужно знать его расширение, иначе ошибка. Path – имя открываемого файла.

# **ЗАКЛЮЧЕНИЕ**

В данной работе мы написали скрипт на языке программирования Python, который распознает символы (в данном случае заглавные буквы моего ФИО), сравнивая эвклидовы близости со всеми эталонными образами в базе данных. Использовались различные библиотеки,

Созданную программу и сам метод не стоит рассматривать как полноценное решение вопроса распознавания символов. Всё из-за погрешностей и допущений. Дело в том, что данный способ является примитивнейшим решением. Однако тема хорошо подходит в качестве обучающего материала. Реализуя программу, можно понять некоторую логику интеллектуальных систем управления, работу алгоритмов, работу графических библиотек. Также есть плюсы с точки зрения программирования. Программный код, алгоритм лёгкие и понятные, что полезно для изучения разработки ПО.

# **Список литературы**

* <https://stackoverflow.com/>
* <https://docs.python.org/3/>
* <https://docs.python.org/3/library/tkinter.html>
* <https://scikit-image.org/docs/stable/>
* <https://ru.wikipedia.org/wiki/Теория_распознавания_образов>
* Курс лекций по ИСУ