Оглавление

[Введение 5](#_Toc59355795)

[1. Анализ технического задания 7](#_Toc59355796)

[1.1. Обзор предметной области 7](#_Toc59355797)

[1.2. Исходные данные к проекту 9](#_Toc59355798)

[1.3. Обзор аналогичных алгоритмов 10](#_Toc59355799)

[1.4. Требования к разрабатываемой информационной системе 13](#_Toc59355800)

[2. Разработка математической модели и алгоритмов 14](#_Toc59355801)

[2.1. Моделирование ИС 14](#_Toc59355802)

[2.2. Описания алгоритмов работы ИС 17](#_Toc59355803)

[3. Реализация ИС 23](#_Toc59355806)

[Заключение 32](#_Toc59355807)

[Список использованной литературы 33](#_Toc59355808)

[Приложение А 34](#_Toc59355809)

[Приложение В 36](#_Toc59355810)

## Введение

В рамках данного курсового проекта представлен процесс разработки информационной системы распознавания матричной маркировки на цифровых изображениях черных металлов. В настоящее время, в связи с необходимостью развития промышленности, требуется введение новых наукоемких технологий по контролю движения продукции. Реализация такого контроля возможна с помощью внедрения систем автоматической идентификации и распознавания. Для контроля движения продукции используются маркировки промышленных изделий. На новейших промышленных предприятиях контроль движения продукции осуществляется специализированными средствами - системами автоматической идентификации маркировки (САИМ). Основная сложность при использовании САИМ заключается в задаче локализации промышленной продукции, которую усложняют наличие неравномерной освещенности и резкости (засвечивание изображения искусственным и солнечным светом, неверная настройка фокуса), присутствие шумов и размытия, прочих лишних объектов. Автоматизация этой сферы деятельности является мало проработанной и, чаще всего, выполняется вручную. Этим обусловлена актуальность данной работы.

Цель данного курсового проекта – спроектировать информационную систему распознавания матричной маркировки на цифровых изображениях черных металлов.

Задачи данного курсового проекта:

a) провести анализ предметной области, проанализировать аналогичные алгоритмы, предъявить требования к разрабатываемой информационной системе;

b) произвести математическое моделирование разрабатываемой информационной системы;

c) проанализировать алгоритмы, используемые при разработке информационной системы, описать их;

d) реализовать информационную систему.

Таким образом, общая структура курсового проекта состоит из трех глав и выглядит следующим образом:

a) в первой главе производится анализ технического задания, а именно:

- обозревается предметная область;

- определяются исходные данные к проекту;

- кратко обозреваются некоторые аналогичные алгоритмы;

- выставляются требования к разрабатываемой информационной системе

b) во второй главе производится математическое моделирование разрабатываемой информационной системы, приводится описание используемых в разработке алгоритмов;

c) в третьей главе описывается процесс реализации разработанной информационной системы.

## Анализ технического задания

## 1.1. Обзор предметной области

Маркировка красками или специальными чернилами - один из старейших способов маркировки. Наибольшее распространение сегодня маркировка красками имеет в легкой, электронной и пищевой промышленности, где используется оборудование типа IMAJE и DOMINO. Эти маркеры оборудованы специальными головками с соплами, с помощью которых возможно нанесение любого вида маркировки. Контрастность изображения обеспечивается различным количеством точек на единицу площади изображения. Новейшее оборудование для маркировки красками позволяет создавать разноцветные обозначения очень высокого качества.

Однако клеймение горячего металла красками невозможно: лучшие образцы лакокрасочной продукции выдерживают температуру не более 1000ºС. Плюс к этому в условиях металлургических предприятий и горячих цехов трудно обеспечить необходимую чистоту и приемлемый температурный режим, требуемые для безотказной работы краскоструйных сопел.

Затрудняет использование краскоструйных маркеров и характер поверхности многих металлоизделий. Окалина, пыль, неровности, влага и т.п. делают маркировку красками фрагментарной и недолговечной.

Все перечисленные факторы делают невозможным или экономически невыгодным использование краскоструйных маркеров в горячих цехах и при машинах непрерывного литья заготовок.

Краски используют для маркировки труб горячекатаных (на выходе из прокатного стана, когда температура изделия становится приемлемой), маркировки слябов и блюмов и т.п.

Оборудование для маркировки краской состоит из следующих узлов:

- головка с 7 или 9 соплами;

- манипулятор для перемещения головки с соплами;

- специальная панель для подготовки краски;

- пульт управления;

- терминал для оператора, с помощью которого можно вручную задавать параметры маркировки;

- электронная система управления (программное обеспечение).

Краскосопла имеют увеличенный диаметр отверстий, что препятствует их засорению, и приспособлены для операции самоочищения. Минимальная высота наносимой маркировки - 19 миллиметров, что может служить ограничением в использовании такого оборудования для некоторых видов металлоизделий.

С помощью оборудования для маркировки краской можно наносить не только буквенно-цифровые коды, но и штрих-коды. Однако, на данном оборудовании можно нанести только бинарные коды (содержащие минимум информации о металлоизделии), для считывания которых нужна специальная аппаратура (промышленные сканеры).

Оборудование для нанесения маркировки краской имеет высокую степень надежности. Однако, при использовании маркировки в металлургии с помощью красок и специальных чернил необходимо учитывать, что для полного высыхания краски требуется определенное время, при этом чем горячее металлоизделие, тем больший срок требуется для высыхания краски. При высокой температуре маркируемых металлоизделий также повышаются эксплуатационные расходы на краскоструйное оборудование: увеличивается расход красок, требуется больше времени для технического обслуживания агрегата и т.п.

Основные характеристики:

- постоянные метки для идентификации;

- высокая степень разборчивости символов для обычного или автоматического считывания в условиях неповреждения метки;

- разборчивая маркировка на грубых и неровных поверхностях;

- высокая скорость маркировки [3].

Основное оборудование, необходимое для нанесения маркировки – это многопистолетные установки нанесения краски или чернил.

В ходе работы над разработкой информационной системы, распознающей матричные маркировки, будут использоваться искусственно сгенерированные изображения. Пример такого изображения представлен на рисунке 1.



Рисунок 1 – пример тестового изображения

## 1.2. Исходные данные к проекту

В качестве входных данных были определены следующие:

a) набор смоделированных исходных изображений для проверки правильности работы разрабатываемой информационной системы;

b) среда программирования Visual Studio Code для программной реализации алгоритма и проведения экспериментальных исследований работы алгоритма;

c) язык программирования Python, так как данный язык является языком высокого уровня, что значительно упрощает работу над разработкой информационной системы, и позволяет реализовать возможность создания графического интерфейса, а также поддерживает работу с библиотеками Tkinter, OpenCV, Numpy и Tensorflow;

d) библиотека Tkinter, при помощи которой будет реализован графический интерфейс программы;

d) библиотека OpenCV - это библиотека с открытым исходным кодом, разработанная на языке C++, которая используется разработчиками и исследователями при решении задач, связанных с компьютерным зрением;

e) библиотека Numpy – это также библиотека с открытым исходным кодом, которая позволяет работать с массивами и вычислять сложные арифметические операции;

f) библиотека Tensorflow - открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия.

Выходными данными в курсовом проекте будет являться текст распознанной матричной маркировки.

## 1.3. Обзор аналогичных алгоритмов

Задача распознавания матричной маркировки схожа с задачей распознавания почтовых индексов. В связи с этим, за основу был взят статистический метод распознавания рукописных цифр на почтовых индексах.

В соответствии с современными нормами оформления почтовых конвертов индексы могут записываться в произвольной форме, но в рамках заранее заданного поля определенного размера (отдельные ячейки для каждой цифры). Для автоматизации процесса сортировки писем почтовые индексы необходимо распознать. Поле индекса на конверте является единственным ограничением, но именно оно может быть взято за основу для унификации распознаваемых цифр. В случае с матричной маркировкой, ограничения по площади появляются после сегментации изображения (один сегмент – один символ).

Шрифтозависимые алгоритмы основываются на сопоставлении идентичных элементов в известных шрифтах и распознаваемых символах [7].

Первыми пунктами данного алгоритма являются выравнивание и масштабирование символов, но так как в случае с текущей темой все символы типизированы, этот пункт можно пропустить.

Далее производится обработка унифицированных изображений цифр имеющихся образцов почерков для получения их индивидуальных характеристик. С этой целью подсчитывается количество черных пикселей в каждом из столбцов *xi* (*i = 1…n*) и строк *yj* (*j = 1…m*) изображения. Полученные значения во многом зависят от толщины контура цифр. Для исключения влияния этого параметра выполняется нормирование с помощью общего количества черных пикселей для изображения рассматриваемой цифры и соответствующего почерка:

(1)

(2)

при этом, очевидно, что:

, (3)

и, соответственно, , .

Векторы и , элементами которых являются нормированные величины и , выступают в качестве характеристик соответствующих цифр. Верхний индекс *p* = 0,1,2,…,9 отмечает принадлежность вектора к соответствующей цифре, а индекс *l* = 1,2,… – к соответствующему почерку.

Выработка эталона для распознавания (обучение системы) производится путем усреднения элементов векторов и для каждой цифры при заданном количестве почерков [7]. Можно предположить, что если нанесение матричных маркировок стандартизировано, то для распознавания данного типа меток достаточно выработать эталон для распознавания на нескольких возможных типах изображений:

- чистое изображение;

- изображение со слабыми шумами;

- изображение со средним уровнем шумов;

- изображение с сильными шумами;

- изображение со световыми бликами.

Так, например, элемент эталонного (усредненного) вектора для цифры «0» по пяти почеркам определяется из соотношения

Аналогично вычисляется .

Распознавание цифры производится в следующем порядке:

1) считывание цифры;

2) бинаризация изображения;

3) нормирование изображения;

4) получение индивидуальной характеристики цифры (векторы и );

5) поочередное сравнение индивидуальной характеристики цифры с эталонными характеристиками всех цифр [7].

Поиск максимального совпадения производится путем нахождения рассогласований между индивидуальной характеристикой распознаваемой цифры и эталонными характеристиками всех цифр и выбора минимального из них. Определяются рассогласования отдельно для векторов и и, в качестве третьего параметра, вычисляется их сумма:

 (4)

 (5)

 (6)

Для облегчения анализа результатов при тестировании все полученные для какой-либо распознаваемой цифры величины нормируются с помощью , – с помощью , а – с помощью . В этом случае минимальные значения нормированных ,  и  равны единице и могут быть легко выделены. Рассматриваемая цифра считается распознанной при выполнении одного из логических выражений к величинам (4), (5) и (6) эталонной характеристики какой-либо цифры:

 (7)

 (8)

На основании выполненной работы автором статьи были получены следующие результаты:

- предлагаемый метод позволил с вероятностью ~ 0.97 распознать рукописные цифры;

- в 2,5 % однозначно определено, что цифра не распознана, не выполнено ни одно из логических выражений (7) к величинам (4), (5) и (6). Это всегда относилось к цифрам «2» и «8», что сужает круг дополнительного поиска и облегчает использование других методов распознавания, ориентированных именно на эти цифры;

- только в 0,5 % случаев происходит ошибочное распознавание. Это позволяет рекомендовать предложенный метод для распознавания рукописных цифр, в том числе в почтовых индексах [5].

В задаче распознавания матричной маркировки необходимо задействовать большее количество алгоритмов и предусмотреть корректное удаление дефектов изображения (предобработка изображения). Более подробно этот процесс описывается во второй главе пояснительной записки.

## 1.4. Требования к разрабатываемой информационной системе

Требования к разрабатываемой информационной системе следующие:

- разрабатываемая система должна оперировать со всеми входными данными и получать в результате текст распознанной матричной маркировки;

- каждый алгоритм разрабатываемой информационной системы должен состоять из отдельных элементарных шагов, работающих системно;

- последовательность шагов каждого алгоритма должна быть детерминирована, т.е. после каждого шага должно указываться, какой шаг следует выполнять дальше, либо указываться, когда следует работу алгоритма считать законченной;

- каждый алгоритм должен обладать результативностью, т.е. останавливаться после конечного числа шагов (зависящего от исходных данных) с выдачей результата.

## 2. Разработка математической модели и алгоритмов

## 2.1. Моделирование ИС

Разработка проекта информационной системы была начата с разработки модели информационной системы, которая была проведена в несколько этапов:

a) модель «черного ящика»;

b) модель состава;

c) структурная модель;

d) функциональная модель.

Модель «черного ящика» показывает следующие параметры: входные данные (изображение с матричной маркировкой), выходные данные (распознанная маркировка), сам «черный ящик» (процесс распознавания маркировки) и внешние факторы, влияющие на распознавание (шумы, световые блики, отсутствие резкости на изображении). Модель «черного ящика» представлена на рисунке 2.

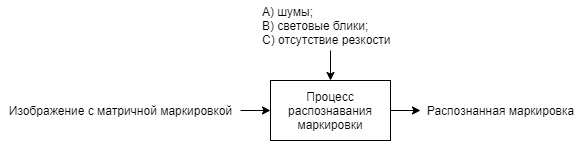


Рисунок 2 – модель «черного ящика»

Следующий этап – разработка модели состава. По сути, модель состава – это то, что находится внутри «черного ящика», но уже более подробно описанное – в виде отдельных блоков. В модели состава, соответствующей данной информационной системе, должно присутствовать 4 следующих блока:

a) блок предобработки изображения – на данном этапе происходит фильтрация шума на изображении, удаление световых бликов, повышение контраста изображения (это необходимо для того, чтобы распознавание маркировки на последнем этапе произошло более качественно) [4];

b) блок локализации маркировки – в данный блок включены алгоритмы, позволяющие определить координаты месторасположения маркировки на изображении, это нужно для ускорения работы программы, чтобы не приходилось задействовать всю область изображения [2];

c) блок сегментации маркировки – локализованная маркировка, обрабатываясь алгоритмами этого блока, разделяется на отдельные символы, что позволит в дальнейшем качественно произвести распознавание [1];

d) блок распознавания маркировки – в данном блоке происходит следующее: каждый поступающий на вход сегментированный элемент маркировки поступает в нейронную сеть, обученную при помощи датасета из 1000 элементов (цифры от 0 до 9), на выход поступает текст распознанной матричной маркировки [2].

Модель состава разработанной информационной системы представлена на рисунке 3.



Рисунок 3 – модель состава

Структурная модель представляет собой модель состава, дополненную стрелками, показывающими ход работы разрабатываемой информационной системы. Таким образом, на рисунке 4 видно, что на вход информационной системе проступает изображение с матричной маркировкой, где далее оно поступает в блок предобработки изображения, после чего маркировка локализуется, сегментируется на отдельные символы и наконец распознается. На выход поступает текст распознанной маркировки.

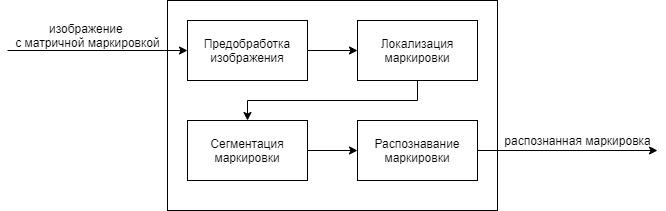


Рисунок 4 – структурная модель

Последним этапом в моделировании является построение функциональной схемы – схемы, которая подробно описывает принцип работы информационной системы. Каждый блок, описанный в модели состава, на функциональной схеме представлен в виде нескольких алгоритмов, которые последовательно обрабатывают изображение в целях эффективного получения выходных данных [2].

Таким образом, процесс распознавания маркировки был разделен на 7 алгоритмов, последовательно идущих друг за другом:

a) предобработка:

- повышение контраста изображения;

- удаление световых бликов;

- удаление шума;

b) локализация: выделение области с числами;

c) сегментация: получение массива чисел;

d) распознавание:

- распознавание цифр;

- формирование цифрового значения маркировки.

Более наглядно работа информационной сети представлена на рисунке 5 – функциональной модели разрабатываемой ИС. Загруженное оператором изображение поступает в систему, обрабатывается вышеперечисленными алгоритмами, и на выход поступает текст распознанной матричной маркировки.

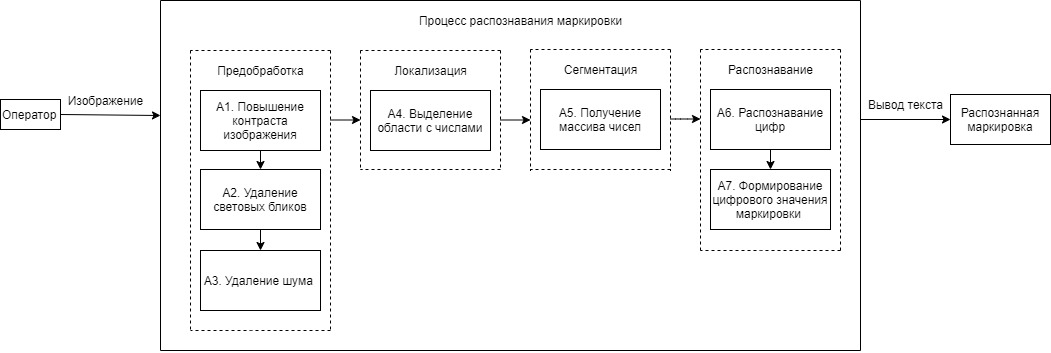


Рисунок 5 – функциональная модель

Для того, чтобы осуществить работу предпоследнего алгоритма, необходимо дополнительно разработать небольшую, обученную датасетом с эталонными изображениями, нейронную систему. На рисунке 6 изображена общая схема процесса распознавания маркировки.

Х – это характеристики, поступающие на вход для обучения информационной системы (база эталонных изображений), и реальные характеристики, получаемые с изображения после сегментации [7]. Обе совокупности этих характеристик начинают свою работу в системе сравнения, организуемой алгоритмом А6 (алгоритм обучения системы описан в пункте 2.2.6).

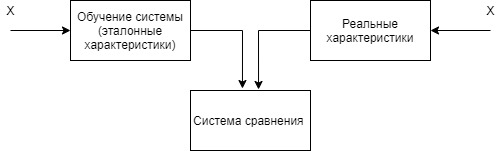


Рисунок 6 – общая схема процесса распознавания

## 2.2. Описания алгоритмов работы ИС

В данном пункте пояснительной записки приведены краткие описания используемых в данной информационной системе алгоритмов. В программной реализации будут использоваться методы библиотеки OpenCV, поэтому алгоритмы базовых операций над изображениями (таких как перевод изображения в градации серого, бинаризация, дилатация, поиск контуров и т.д.) не описываются подробно.

## 2.2.1. Блок предобработки. Алгоритм А1. Повышение контраста

Повышение контраста изображения будет производиться следующим образом. На первом шаге нужно будет найти среднее арифметическое всего количества пикселей на черно-белом изображении и прибавить к ним 100, так как это пороговое значение позволит более качественно отделить «полезные» пиксели от фоновых [6].

Затем нужно рассчитать новые значения яркостей пикселей по следующей формуле:

 (9)

где *AVG* – это значение, полученное на предыдущем шаге, *c* – это коэффициент контраста (по умолчанию устанавливается равным 2), *I*0 – текущая яркость пикселя.

Если яркость пикселя выше 255, его значение нужно поставить равным 255. Если яркость пикселя получилась меньше 0, то значение его яркости устанавливается равным 0. Иначе яркость устанавливается равной вычисленному значению.

2.2.2. Блок предобработки. Алгоритм А2. Удаление световых бликов

Световые блики занимают довольно большую часть изображения. В случаях, когда они своим расположением не пересекаются с маркировкой, их можно легко локализовать и удалить при помощи следующего алгоритма:

a) бинаризация изображения [8];

b) применение медианной фильтрации с ядром 3 на 3;

c) дилатация – это свертка изображения ядром 7 на 7;

d) поиск контуров изображения

e) определение площади контуров;

f) установление порогового значения равным 5000 пикселей (если площадь найденных контуров будет больше, чем 5000 пикселей, то часть изображения внутри этого контура будет закрашена фоновым цветом, так как эта область будет распознана как блик);

g) заполнение найденной области фоновым цветом [3].

2.2.3. Блок предобработки. Алгоритм А3. Удаление шума

Алгоритм удаления шума состоит из следующих шагов:

a) поиск контуров;

b) определение площади контуров;

c) определение количества найденных контуров - если площадь контура больше 0, то счетчик количества контуров увеличивается на 1;

d) для всех найденных контуров, кроме самого большого по площади (контура маркировки), яркость пикселей площадей устанавливается равной яркости фона.

Таким образом происходит удаление шума и размытых дефектов на изображении (например, мелкого мусора на изображении металла) [5].

## 2.2.4. Блок локализации. Алгоритм А4. Выделение области с числами

На этапе локализации происходит отделение области с маркировкой от остального изображения [2]. Это достигается при помощи алгоритма, состоящего из нижеперечисленных шагов:

a) дилатация изображения ядром 21 на 13 пикселей;

b) поиск контуров;

c) подсчет количества контуров;

d) отрисовка прямоугольной области вокруг каждого найденного контура;

e) получение 4-х точек по краям прямоугольника;

f) определение координат точек (*x*1;*y*1), (*x*2;*y*2);

g) поиск угла поворота локализованной маркировки по формуле 10:

 (10)

где *x*1, *x*2, *y*1, *y*2 – это координаты локализованной маркировки.

h) выравнивание угла поворота изображения относительно центра маркировки;

i) обрезка изображения согласно координатам локализованной маркировки и нормирование изображения таким образом, чтобы по координате y было 170 пикселей, а по координате x – 100n, где n – это количество цифр в локализуемой маркировке.

2.2.5. Блок сегментации. Алгоритм А5. Получение массива чисел

Сегментация маркировки происходит следующим образом. Зная, что размер каждой цифры в маркировке и в датасете равен 100 на 170 пикселей, нужно разделить общую ширину локализованной маркировки на 100 и провести вертикальные линии раздела в местах, где заканчивается каждая цифра в маркировке. Таким образом составляется массив из сегментированных изображений цифр маркировки [1].

2.2.6. Блок распознавания. Алгоритм А6. Распознавание цифр

Данный алгоритм производит сравнение сегментированных цифр с эталонными посредством пропуска реальных изображений через нейронную сеть, обученную при помощи датасета с эталонными изображениями [7].

Нейронная сеть состоит из двух слоев: входного (17000 нейронов по числу количества пикселей в каждом изображении датасета) и выходного (10 классов – числа от 0 до 9).

Работа нейросети состоит из двух частей: обучения и распознавания. Процесс обучения происходит 1 раз, затем полученный результат сохраняется в файл формата h5. Это необходимо для того, чтобы не приходилось обучать нейронную сеть каждый раз заново при запуске программы.

Обучение нейросети происходит согласно следующему алгоритму:

a) загрузка тренировочного датасета;

b) загрузка одного любого изображения из тренировочного датасета в качестве тестового изображения;

c) масштабирование значений яркости пикселей изображений датасета и тестового изображения от 0 до 1 по следующей формуле:

. (11)

где *I*0 – это текущая яркость пикселя;

d) настройка входного слоя (17000 нейронов) с присвоением ему линейной функции активации:

 (12)

e) настройка выходного слоя (10 классов) с присвоением ему функции активации Softmax, которая применяется в машинном обучении для задач классификации, когда количество возможных классов больше двух:

 (13)

где *K* – это размерность вектора (количество классов объектов), а *z* – это вектор-столбец, рассчитываемый следующим образом:

 (14)

где *x* — вектор-столбец признаков объекта размерности ;

 — транспонированная матрица весовых коэффициентов признаков, имеющая размерность  (*M* — количество признаков объектов);

 — вектор-столбец с пороговыми значениями размерности ;

f) компиляция модели путем установки таких параметров как функция потерь *categorical\_crossentropy* (измеряет точность модели во время обучения), оптимизатора *SGD* (показывает, каким образом обновляется модель на основе входных данных и функции потерь) и метрики *accuracy*, равной доле правильно классифицированных изображений;

g) запуск процесса обучения модели на выборке из 200 случайно выбранных данных тренировочного датасета в течение 40 эпох;

h) сохранение полученных весов (результата обучения) в файл с расширением h5.

Распознавание происходит следующим образом:

a) загрузка в нейросеть тестового изображения;

b) сравнение нейросетью тестового изображения с эталонным изображениями из датасета;

c) вывод результата распознавания распознавания.

Полученный результат поступает на работу следующего алгоритма.

2.2.7. Блок распознавания. Алгоритм А7. Формирование цифрового значения маркировки

Работа этого алгоритма заключается в том, чтобы объединить все распознанные цифры на изображении в одно число и вывести полученное значение.

После того, как работа всех алгоритмов была описана, можно приступить к программной реализации информационной системы.

## 3. Реализация ИС

В соответствии с пунктом «Исходные данные к проекту», изложенном в анализе технического задания, разработка информационной системы ведется на языке программирования Python. Также, для разработки потребуются библиотеки Numpy, OpenCV и Tensorflow.

Перед тем, как начать работать с изображением в программе, его необходимо открыть. Для этого была написана функция uploadImage(self, path):

def uploadImage(self, path):

self.\_\_image = None

f = open(path, "rb")

chunk = f.read()

chunk\_arr = np.frombuffer(chunk, dtype=np.uint8)

self.\_\_image = cv2.imdecode(chunk\_arr, 0)

return False if self.\_\_image is None else True

Данная функция считывает путь изображения и открывает его в программе в виде двумерного массива.

Для отслеживания изменений изображения была написана функция \_\_debagShow(self,image):

def \_\_debagShow(self,image):

if self.\_\_debug:

cv2.imshow('debug', cv2.resize(image, (960, 540)))

cv2.waitKey()

Эта функция необходима лишь на этапе разработки (для того, чтобы фиксировать изменения на каждом этапе в пояснительной записке), в конечном варианте debug будет равен False, то есть промежуточные результаты отображаться не будут.

Для работы данной функции необходимо проинициализировать переменную debug:

def \_\_init\_\_(self, debug=False):

self.\_\_debug = debug

Для отображения текущего состояния изображения используется функция showCurrentStateImage(self):

def showCurrentStateImage(self):

if self.\_\_image is not None:

cv2.imshow('image', cv2.resize(self.\_\_image, (960, 540)))

cv2.waitKey()

return True

else:

return False

Следующим этапом, согласно структурной модели, представленной на рисунке 4, идет предобработка изображения – этап, необходимый для подготовки изображения к дальнейшей работе, на котором минимизируется действие внешних факторов (таких как шумы, размытие изображения и световые блики).

Согласно функциональной модели, первый алгоритм – алгоритм повышения контраста изображения, описанный в пункте 2.2.1. пояснительной записки. В данной программе это происходит при помощи функции \_\_contrast(self, coefficient):

def \_\_contrast(self, coefficient):

avg = self.\_\_image.mean() + 100

self.\_\_image = avg + coefficient \* (self.\_\_image - avg)

self.\_\_image[self.\_\_image>255] = 255

self.\_\_image[self.\_\_image<0] = 0

self.\_\_image = self.\_\_image.astype(np.uint8)

self.\_\_debagShow(self.\_\_image)

Работа над очисткой изображения от шума и световых бликов осуществляется при помощи использования маски фильтра, который проходит по изображению и удаляет дефекты.

Функция \_\_masking(self, image, mask) выглядит следующим образом:

def \_\_masking(self, image, mask):

mask[mask>0] = 1

return cv2.bitwise\_and(image, image, mask=mask)

Световые блики занимают гораздо больше места на изображении, чем импульсный шум и какие-либо неровности в окраске металла, которые при прокате листа металла по конвейеру и фотографировании в это время, могут размываться. Исходя из этого было написано три функции, соответствующие описанным в пунктах 2.2.2. и 2.2.3. пояснительной записки алгоритмах: \_\_eraseBigAreas(self, mask), \_\_eraseSmallAreas(self, mask) и \_\_eraseNoise(self). Вторая функция была написана для того, чтобы можно было ее использовать в работе третьей функции следующим образом:

def \_\_eraseNoise(self):

\_, mask = cv2.threshold(self.\_\_image, 0, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

mask = cv2.medianBlur(mask, 3)

mask = cv2.dilate(mask, cv2.getStructuringElement(cv2.MORPH\_DILATE,(7,7)), iterations = 1)

mask = self.\_\_eraseBigAreas(mask)

self.\_\_image = self.\_\_masking(self.\_\_image, mask) # Big areas erased

\_, mask = cv2.threshold(self.\_\_image, 0, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

mask = cv2.dilate(mask, cv2.getStructuringElement(cv2.MORPH\_DILATE,(9,9)), iterations = 3)

self.\_\_debagShow(mask)

mask = self.\_\_eraseSmallAreas(mask)

self.\_\_image = self.\_\_masking(self.\_\_image, mask)

self.\_\_debagShow(self.\_\_image)

Код поиска световых бликов на изображении и удаления их выглядит следующим образом:

def \_\_eraseBigAreas(self, mask):

cnts = cv2.findContours(mask, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

cnts = cnts[0] if len(cnts) == 2 else cnts[1]

max\_area = 5000

for c in cnts:

area = cv2.contourArea(c)

if area > max\_area:

cv2.drawContours(mask, [c], -1, (0,0,0), -1)

return mask

На рисунке 7 приведен промежуточный результат работы программы после повышения контраста и удаления шума и светового блика (в качестве тестового изображения взят рисунок 1):



Рисунок 7 - изображение с повышенным контрастом и удаленным бликом

Код функции поиска зашумленных областей изображения приведен ниже:

def \_\_eraseSmallAreas(self, mask):

cnts = cv2.findContours(mask, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

cnts = cnts[0] if len(cnts) == 2 else cnts[1]

max\_area = 0

for c in cnts:

area = cv2.contourArea(c)

if area > max\_area:

max\_area\_c = c

max\_area = area

mask = np.zeros\_like(mask)

cv2.drawContours(mask, [max\_area\_c], -1, (255,255,255), -1)

return mask

На рисунке 8 изображен результат применения маски на изображении с целью поиска мелких дефектов, которые не получилось удалить при помощи функций повышения контраста, удаления шума и большого светового блика.

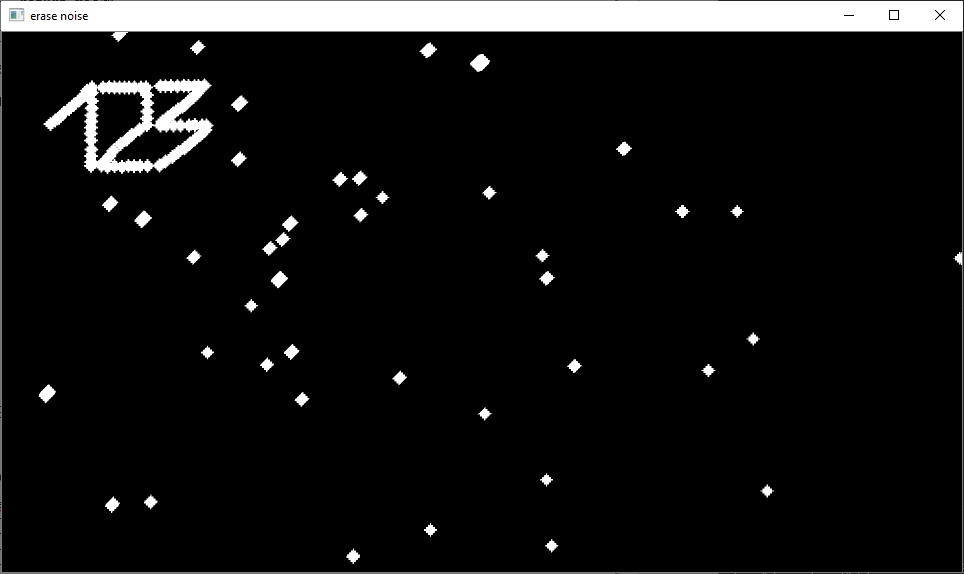


Рисунок 8 - выделение маской мелких дефектов изображения

На рисунке 9 представлен конечный результат работы алгоритмов блока предобработки изображения – чистое бинаризованное изображение с ярко выделенной маркировкой.

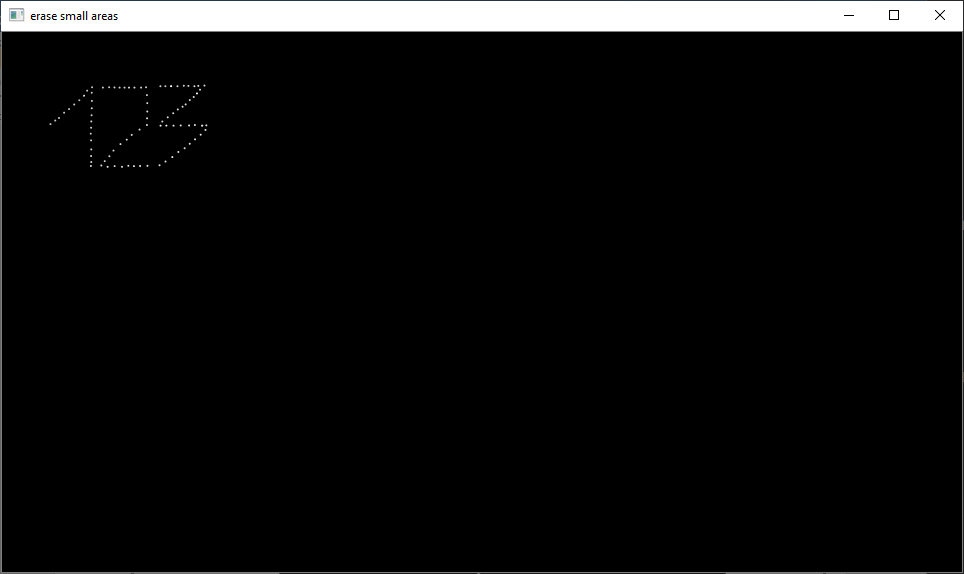


Рисунок 9 - результат работы алгоритмов блока предобработки

После предобработки изображения нужно локализовать маркировку и обрезать оставшуюся часть изображения. Это необходимо для ускорения работы программы. Работа данного алгоритма представлена в пункте 2.2.4. пояснительной записки. В программе локализация реализована в методе \_\_getOnlyNumbers(self):

def \_\_getOnlyNumbers(self):

new\_image = cv2.dilate(self.\_\_image, cv2.getStructuringElement(cv2.MORPH\_OPEN,(21,13)), iterations = 1)

contours = cv2.findContours(new\_image, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

contours = contours[0] if len(contours) == 2 else contours[1]

contour = contours[0]

new\_image = np.zeros\_like(self.\_\_image)

rect = cv2.minAreaRect(contour)

box = cv2.boxPoints(rect)

box = np.int0(box)

distanse = self.\_\_getDistanse(box[0], box[1])

right = box[0]

for i in range(1,4):

if box[i][0] > right[0]:

right = box[i]

left = box[0]

for i in range(1,4):

if box[i][0] < left[0]:

left = box[i]

top = box[0]

for i in range(1,4):

if box[i][1] < top[1]:

top = box[i]

bottom = box[0]

for i in range(1,4):

if box[i][1] > bottom[1]:

bottom = box[i]

deg = math.asin((box[0][0]-box[1][0])/distanse) / math.pi \* 180

self.\_\_image = self.subimage(

self.\_\_image,

((right[0] - left[0])/2 + left[0], (bottom[1] - top[1])/2 + top[1]),

deg,

int(self.\_\_getDistanse(box[1], box[2])),

int(self.\_\_getDistanse(box[0], box[1]))

)

Данный алгоритм включает в себя две функции. Первая - \_\_getDistanse(self, point1, point2), которая нужна для того, чтобы определить координаты линий, по которым будет производиться разделение изображения на часть изображения с маркировкой и фоновую часть. Вторая - subimage(self, image, center, theta, width, height) – это функция, которая производит проведение этих линий раздела и возвращает часть изображения с маркировкой.

def \_\_getDistanse(self, point1, point2):

x1, y1 = point1

x2, y2 = point2

return ((x2-x1)\*\*2 + (y2-y1)\*\*2)\*\*0.5

def subimage(self, image, center, theta, width, height):

shape = ( image.shape[1], image.shape[0] )

matrix = cv2.getRotationMatrix2D( center=center, angle=theta, scale=1 )

image = cv2.warpAffine( src=image, M=matrix, dsize=shape )

x = int( center[0] - width/2 )

y = int( center[1] - height/2 )

image = image[ y:y+height, x:x+width ]

return image

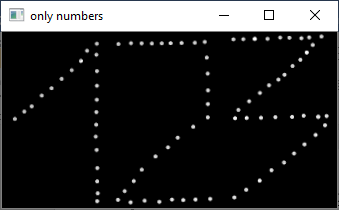


Рисунок 10 - локализованная маркировка на изображении

Далее идет работа алгоритма блока сегментации, а именно получение массива чисел из локализованной маркировки. В данной разрабатываемой информационной системе это реализовано при помощи функции \_\_getNumbersArray(self). Подробно работа этого алгоритма раскрыта в пункте 2.2.5. пояснительной записки.

def \_\_getNumbersArray(self):

path = os.path.dirname(os.path.realpath(\_\_file\_\_)) + "\\numbersDataSet\\images\_of\_0"

os.chdir(path)

number = self.\_\_getimage(path+'\\'+os.listdir()[-1])

y = self.\_\_image.shape[0]

yi, xi = number.shape[:2]

dy = yi/y

x = int(xi/dy)

countnumbers = int(self.\_\_image.shape[1]/x)

blackspace = int((self.\_\_image.shape[1] - countnumbers \* x) / countnumbers) + 1

images = []

for xc in range(0,

self.\_\_image.shape[1],

x + blackspace):

images.append(self.\_\_image[:, xc:xc+x])

cv2.imshow(str(xc), images[-1])

cv2.waitKey(0)

return images

Код работы функции \_\_getimage(self, path) выглядит следующим образом:

def \_\_getimage(self, path):

f = open(path, "rb")

chunk = f.read()

chunk\_arr = np.frombuffer(chunk, dtype=np.uint8)

return cv2.imdecode(chunk\_arr, 0)

На рисунках 11, 12 и 13 представлены результаты сегментации маркировки.

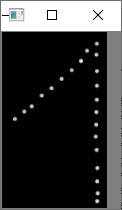


Рисунок 11 - сегментированная цифра 1

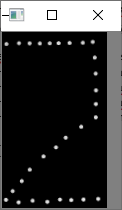


Рисунок 12 - сегментированная цифра 2

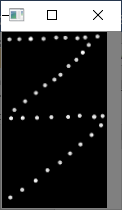


Рисунок 13 - сегментированная цифра 3

После того, как маркировка разбилась на отдельные элементы (изображения отдельных цифр), можно приступать к распознаванию Для реализации этой части программы была написана нейросеть, которая была обучена при помощи датасета, содержащего 1000 изображений матричных цифр (по типу почтовых индексов). Работа нейросети подробно описана в пункте 2.2.6. пояснительной записки. Полный код нейросети представлен в приложении В (модуль tensorflowmodel.py).

Для того, чтобы не приходилось каждый раз при запуске программы заново обучать нейросеть, результаты ее обучения были сохранены в файл number\_model.h5. Подключение этого файла реализовано в модуле II.py, его код выглядит следующим образом:

from tensorflow import keras

import numpy as np

model = keras.models.load\_model('number\_model.h5')

def recognize(image):

image = image.reshape(1, 17000) / 255

prediction = model.predict(image)

return str(np.argmax(prediction[0]))

Вышеприведенные функции в конечном итоге образуют функцию \_\_recognitionValue(self):

def \_\_recognitionValue(self):

value = None

self.\_\_contrast(2)

self.\_\_eraseNoise()

self.\_\_getOnlyNumbers()

numbers = self.\_\_getNumbersArray()

value = ''

for number in numbers:

value += II.recognize(cv2.resize(number, (100, 170)))

return value

В конечном итоге производится формирование текста распознанной маркировки (работа алгоритма А7, описанного в пункте 2.2.7. пояснительной записки):

def getValueFromImage(self):

if self.\_\_image is not None:

return self.\_\_recognitionValue()

else:

return None

На рисунке 14 представлен результат распознавания матричной маркировки, выведенный в текстовом виде в окно для вывода.

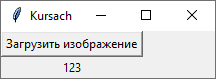


Рисунок 14 - вывод результата распознавания

Код графического интерфейса программы приведен ниже:

import tkinter

from tkinter import filedialog as fd

from MyFilters import MyFilters

def onclick():

path = fd.askopenfilename()

MF = MyFilters()

if MF.uploadImage(path):

lbl\_answer['text'] = MF.getValueFromImage()

# MF.showCurrentStateImage()

else:

lbl\_answer['text'] = ('Невозможно загрузить изображение')

root = tkinter.Tk()

root.title('Kursach')

btn = tkinter.Button(root, text='Загрузить изображение', command=onclick)

btn.grid(row=1, column=1)

lbl\_answer = tkinter.Label()

lbl\_answer.grid(row=2, column=1)

root.mainloop()

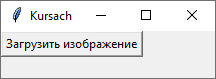


Рисунок 15 - графический интерфейс программы

Таким образом, работа всех алгоритмов, приведенных во второй главе пояснительной записки была реализована.

## Заключение

В процессе курсового проектирования была разработана информационная система, которая позволяет распознать матричную маркировку на цифровом изображении. Были соблюдены требования к работе алгоритма:

- он оперирует со всеми входными данными;

- состоит из детерминированной последовательности отдельных элементарных шагов, работающих системно;

- получает в распознанную матричную маркировку в текстовом виде.

В первую очередь был проведен анализ предметной области, проанализированы аналогичные алгоритмы, предъявлены требования к разрабатываемой информационной системе.

Затем было произведено математическое моделирование разрабатываемой информационной системы, а также приводится описание используемых в разработке алгоритмов.

В завершение представлен процесс разработки информационной системы и приведен программный код реализованной информационной системы.

Таким образом, задачи данного курсового проекта были выполнены в полном объеме.

## Список использованной литературы

1. Автоматический выбор порогов для сегментации изображений на основе градиентных структурных тензоров [электронный ресурс] / Грузман // Доклады Академии наук высшей школы Российской Федерации. - 2013. - №2. - с.67-75. - Режим доступа: https://rucont.ru/efd/236309.

2. Гонсалес, Р., Вудс, Р. Цифровая обработка изображений: монография. - Москва: Техносфера, 2005. - 1072 с.

3. Грузман, И.С. Цифровая обработка изображений в информационных системах: учеб. пособие. – Новосибирск: Изд-во НГТУ, 2000. – 168 с.

4. Кравцова, Т.А. Сравнительное исследование методов адаптивной бинаризации в задаче автоматизированного анализа изображений клеток в иммуноцитохимии. Молодежный научно-технический вестник, 2015.

5. Приоров, А.Л. Цифровая обработка изображений: учеб. пособие / Ярославский государственный университет. – Ярославль: ЯрГУ, 2007. – 235 с.

6. Федоров, А. Бинаризация черно-белых изображений: состояние и перспективы развития [электронный ресурс]. Режим доступа: http://itclaim.ru/Library/Books/ITS/wwwbook/ist4b/its4/fyodorov.htm/ (дата обращения: 30.11.2020).

7. Фисенко, В.Т., Фисенко, Т.Ю. Компьютерная обработка и распознавание изображений: учеб. пособие. – СПб.: СПбГУ ИТМО, 2008. – 192 с.

8. Янковский, А.А., Бугрий, А.Н. Критерии выбора метода бинаризации при обработке изображений лабораторных анализов // АСУ и приборы автоматики [электронный ресурс], 2010. № 153. Режим доступа: http://cyberleninka.ru/article/n/kriterii-vybora-metoda-binarizatsiipri-obrabotke-izobrazheniy-laboratornyh-analizov/ (дата обращения: 30.11.2020).

## Приложение А

Исходные данные к проекту



Рисунок А1 – Исходное изображение 1



Рисунок А2 – Исходное изображение 2

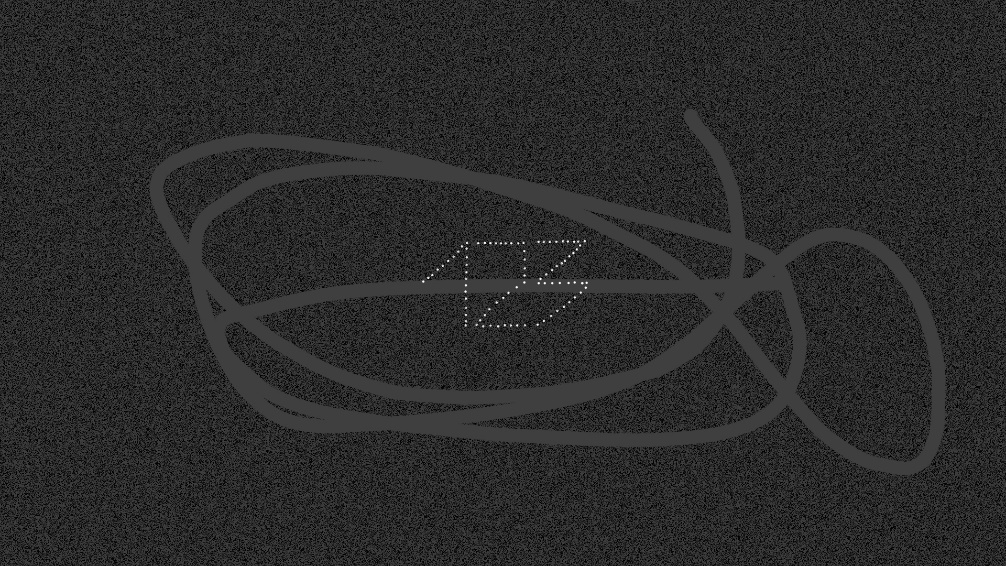


Рисунок А3 – Исходное изображение 3



Рисунок А4 – Исходное изображение 4



Рисунок А5 – Исходное изображение 5

## Приложение В

Листинг

Модуль DotMatrixRecognition.py:

import tkinter

from tkinter import filedialog as fd

from MyFilters import MyFilters

def onclick():

path = fd.askopenfilename()

MF = MyFilters(debug=True)

if MF.uploadImage(path):

lbl\_answer['text'] = MF.getValueFromImage()

# MF.showCurrentStateImage()

else:

print('Невозможно загрузить изображение')

root = tkinter.Tk()

root.title('Kursach')

btn = tkinter.Button(root, text='Загрузить изображение', command=onclick)

btn.grid(row=1, column=1)

lbl\_answer = tkinter.Label()

lbl\_answer.grid(row=2, column=1)

root.mainloop()

Модуль MyFilters.py:

import II

from cv2 import cv2

import numpy as np

import math

import os

class MyFilters():

\_\_image = None

\_\_debug = False

def \_\_init\_\_(self, debug=False):

self.\_\_debug = debug

def \_\_debagShow(self, name, image):

if self.\_\_debug:

if image.shape[0] > 540 or image.shape[1] > 960:

cv2.imshow(name, cv2.resize(image, (960, 540)))

else:

cv2.imshow(name, image)

cv2.waitKey()

def \_\_masking(self, image, mask):

mask[mask>0] = 1

return cv2.bitwise\_and(image, image, mask=mask)

def \_\_eraseBigAreas(self, mask):

cnts = cv2.findContours(mask, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

cnts = cnts[0] if len(cnts) == 2 else cnts[1]

max\_area = 5000

for c in cnts:

area = cv2.contourArea(c)

if area > max\_area:

cv2.drawContours(mask, [c], -1, (0,0,0), -1)

return mask

def \_\_eraseSmallAreas(self, mask):

cnts = cv2.findContours(mask, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

cnts = cnts[0] if len(cnts) == 2 else cnts[1]

max\_area = 0

max\_area\_c = cnts[0]

for c in cnts:

area = cv2.contourArea(c)

if area > max\_area:

max\_area\_c = c

max\_area = area

mask = np.zeros\_like(mask)

cv2.drawContours(mask, [max\_area\_c], -1, (255,255,255), -1)

return mask

def \_\_contrast(self, coefficient):

avg = self.\_\_image.mean() + 100

self.\_\_image = avg + coefficient \* (self.\_\_image - avg)

self.\_\_image[self.\_\_image>255] = 255

self.\_\_image[self.\_\_image<0] = 0

self.\_\_image = self.\_\_image.astype(np.uint8)

self.\_\_debagShow('contrast', self.\_\_image)

def \_\_eraseNoise(self):

\_, mask = cv2.threshold(self.\_\_image,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

mask = cv2.medianBlur(mask, 3)

mask = cv2.dilate(mask, cv2.getStructuringElement(cv2.MORPH\_DILATE,(7,7)), iterations = 1)

mask = self.\_\_eraseBigAreas(mask)

self.\_\_image = self.\_\_masking(self.\_\_image, mask) # Big areas erased

\_, mask = cv2.threshold(self.\_\_image, 0, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

mask = cv2.dilate(mask, cv2.getStructuringElement(cv2.MORPH\_DILATE,(9,9)), iterations = 3)

self.\_\_debagShow('erase noise', mask)

mask = self.\_\_eraseSmallAreas(mask)

self.\_\_image = self.\_\_masking(self.\_\_image, mask)

self.\_\_debagShow('erase small areas', self.\_\_image)

def \_\_getDistanse(self, point1, point2):

x1, y1 = point1

x2, y2 = point2

return ((x2-x1)\*\*2 + (y2-y1)\*\*2)\*\*0.5

def subimage(self, image, center, theta, width, height):

shape = ( image.shape[1], image.shape[0] )

matrix = cv2.getRotationMatrix2D( center=center, angle=theta, scale=1 )

image = cv2.warpAffine( src=image, M=matrix, dsize=shape )

x = int( center[0] - width/2 )

y = int( center[1] - height/2 )

image = image[ y:y+height, x:x+width ]

return image

def \_\_getOnlyNumbers(self):

new\_image = cv2.dilate(self.\_\_image, cv2.getStructuringElement(cv2.MORPH\_OPEN,(21,13)), iterations = 1)

contours = cv2.findContours(new\_image, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

contours = contours[0] if len(contours) == 2 else contours[1]

contour = contours[0]

new\_image = np.zeros\_like(self.\_\_image)

rect = cv2.minAreaRect(contour)

box = cv2.boxPoints(rect)

box = np.int0(box)

distanse = self.\_\_getDistanse(box[0], box[1])

right = box[0]

for i in range(1,4):

if box[i][0] > right[0]:

right = box[i]

left = box[0]

for i in range(1,4):

if box[i][0] < left[0]:

left = box[i]

top = box[0]

for i in range(1,4):

if box[i][1] < top[1]:

top = box[i]

bottom = box[0]

for i in range(1,4):

if box[i][1] > bottom[1]:

bottom = box[i]

deg = math.asin((box[0][0]-box[1][0])/distanse) / math.pi \* 180

self.\_\_image = self.subimage(

self.\_\_image,

((right[0] - left[0])/2 + left[0], (bottom[1] - top[1])/2 + top[1]),

deg,

int(self.\_\_getDistanse(box[1], box[2])),

int(self.\_\_getDistanse(box[0], box[1]))

)

self.\_\_debagShow('only numbers', self.\_\_image)

def \_\_getimage(self, path):

f = open(path, "rb")

chunk = f.read()

chunk\_arr = np.frombuffer(chunk, dtype=np.uint8)

return cv2.imdecode(chunk\_arr, 0)

def \_\_getNumbersArray(self):

path = os.path.dirname(os.path.realpath(\_\_file\_\_)) + "\\numbersDataSet\\images\_of\_0"

os.chdir(path)

number = self.\_\_getimage(path+'\\'+os.listdir()[-1])

y = self.\_\_image.shape[0]

yi, xi = number.shape[:2]

dy = yi/y

x = int(xi/dy)

countnumbers = int(self.\_\_image.shape[1]/x)

blackspace = int((self.\_\_image.shape[1] - countnumbers \* x) / countnumbers) + 1

images = []

for xc in range(0,

self.\_\_image.shape[1],

x + blackspace):

images.append(self.\_\_image[:, xc:xc+x])

self.\_\_debagShow(str(xc+1), images[-1])

return images

def \_\_recognitionValue(self):

value = None

self.\_\_contrast(2)

self.\_\_eraseNoise()

self.\_\_getOnlyNumbers()

numbers = self.\_\_getNumbersArray()

value = ''

for number in numbers:

value += II.recognize(cv2.resize(number, (100, 170)))

return value

def uploadImage(self, path):

self.\_\_image = None

f = open(path, "rb")

chunk = f.read()

chunk\_arr = np.frombuffer(chunk, dtype=np.uint8)

self.\_\_image = cv2.imdecode(chunk\_arr, 0)

return False if self.\_\_image is None else True

def showCurrentStateImage(self):

if self.\_\_image is not None:

cv2.imshow('image', cv2.resize(self.\_\_image, (960, 540)))

cv2.waitKey()

return True

else:

return False

if self.\_\_image is not None:

return self.\_\_recognitionValue()

else:

return None

def getValueFromImage(self):

if self.\_\_image is not None:

return self.\_\_recognitionValue()

else:

return None

Модуль tensorflowmodel.py:

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras import utils

from tensorflow.keras.preprocessing import image

import numpy as np

from cv2 import cv2

import os

def uploadImage(path):

f = open(path, "rb")

chunk = f.read()

chunk\_arr = np.frombuffer(chunk, dtype=np.uint8)

image = cv2.imdecode(chunk\_arr, 0)

image = cv2.resize(image, (100, 170))

return False if image is None else image

path = os.path.dirname(os.path.realpath(\_\_file\_\_)) + "\\numbersDataSet"

os.chdir(path)

x\_train = []

y\_train = []

for i in range(10):

os.chdir(path + f'\\images\_of\_{i}')

for img in os.listdir():

x\_train.append(uploadImage(img))

y\_train.append(i)

x\_train = np.array(x\_train).reshape(1000, 17000)

os.chdir(path + f'\\images\_of\_{4}')

x\_test = uploadImage('64.jpg').reshape(1, 17000) / 255

x\_train = x\_train / 255

print(y\_train[0])

y\_train = utils.to\_categorical(y\_train, 10)

print(y\_train[0])

model = Sequential()

model.add(Dense(17000, input\_dim=17000, activation="relu"))

model.add(Dense(10, activation="softmax"))

model.compile(loss="categorical\_crossentropy", optimizer="SGD", metrics=["accuracy"])

print(model.summary())

history = model.fit(x\_train, y\_train,

batch\_size=200,

epochs=40,

verbose=1)

os.chdir(os.path.dirname(os.path.realpath(\_\_file\_\_)))

model.save('number\_model.h5')

predictions = model.predict(x\_test)

n = 0

print(predictions[n])

print(np.argmax(predictions[n]))

Модуль II.py:

from tensorflow import keras

import numpy as np

model = keras.models.load\_model('number\_model.h5')

def recognize(image):

image = image.reshape(1, 17000) / 255

prediction = model.predict(image)

return str(np.argmax(prediction[0]))