Санкт-Петербургский государственный университет
Кафедра технологий программирования

Коробов Дмитрий Владимирович

Выпускная квалификационная работа бакалавра

Опыт распознавания изображений в специальной области

Направление 010400
Прикладная математика и информатика

Научный руководитель,
старший преподаватель
Андреев Д.В.

Санкт-Петербург
2016

Содержание

Введение

В настоящий момент развитие цифровых технологий позволяет людям передавать и принимать информацию с большой скоростью. Доступность и развитость современной техники сделало особо распространенным такой вид данных, как цифровые изображения. Например, в последние годы среди пользователей интернета и социальных сетей прослеживается тенденция к передаче и обмену информацией в форме картинок и фотографий.

В связи с этим большим спросом пользуются программы, позволяющие среднестатистическому пользователю получать и работать с данными, полученными с изображения. Эти задачи можно решить с помощью методов цифровой обработки.

Первоначально эти методы разрабатывались и исследовались специалистами, работающими в области прикладной математики. Сейчас создано несколько общедоступных библиотек, позволяющих вести работу с цифровыми данными, однако их использование все еще требует определенного технического образования. Такие библиотеки компьютерного зрения не содержат готовых решений и являются лишь инструментом для создания рабочих алгоритмов.

Описанные технологии доступны массовому пользователю лишь посредствам различных программных продуктов, в том числе мобильных приложений, рынок которых стремительно растет. В процессе работы над мобильным приложением, перед разработчиками стоит задача создания механизма упрощенного взаимодействия пользователя с библиотеками компьютерного зрения.

Перенасыщенность рынка мобильных приложений для массового пользователя сподвигло разработчиков на расширение диапазона сфер их использования. Это привело к созданию приложений, направленных на применение в узких областях человеческой деятельности. В свою очередь, доступность и повсеместность гаджетов сделало их незаменимым инструмен-

том в работе многих специалистов. Поэтому в нынешних условиях для разработчика важно умение находить решения еще нерешенных или мало изученных задач.

Одной из таких задач является распознавание шрифта текста с изображения. Например, с ней часто сталкиваются многие web дизайнеры. Красивый шрифт добавит сайту оригинальности, и он будет выгодно выглядеть на фоне конкурентов. Конечно, уже существуют инструменты, которые решают такую задачу, но в подавляющем большинстве они представляют собой специализированные сайты. Как быть, если нужно будучи вне дома, срочно распознать шрифт с какой-нибудь вывески или рекламного плаката, используя только возможности мобильного телефона? На момент написания данной работы в магазине мобильных приложений "App Store" находилось все лишь несколько программ, реализующих такой функционал, и то только частично. В частности, из минусов можно выделить работу только с включенным интернетом, так как часть приложений несет в себе функциональность отправки изображения на специализированный сайт и не решает задачу сама.

Исходя из всего вышесказанного, целью данной работы является проектирование и разработка приложения для мобильной платформы iOS с функционалом распознавания шрифта текста с изображения. Распознавание должно происходить на самом устройстве, без использования сторонних ресурсов. На момент написания работы не было найдено готовых решений с открытым кодом, все существующие программы были коммерческими. Также в литературе и сети не было информации о конкретном алгоритме решения.

Краткий обзор литературы

В качестве основных источников для изучения библиотеки OpenCV в данной работе были использованы такие книги, как Gloria Bueno Garcia "Learning Image Processing with OpenCV" [1] и Gary Bradski, Adrian Kaehler "Learning OpenCV: Computer Vision with the OpenCV Library" [2]. В данных учебниках авторы подробно описывают возможности библиотеки, приводят примеры работы различных функций. Практические советы по реализации алгоритмов и подходов представлены в книгах David A. Forsyth, Jean Ponce "Computer Vision: A Modern Approach" [3] и Kenneth Dawson-Howe "A Practical Introduction to Computer Vision with OpenCV" [4].

Существует также официальный сайт "docs.opencv.org" [10], на котором присутствует документация обо всех функциях библиотеки: описание входных параметров к ним, принципы работы. Для многих функций описан математический фундамент или алгоритм, лежащие в их основе.

При подготовке к выполнению данной работы наибольшее внимание было уделено статьям Serge Belongie, Jitendra Malik, Jan Puzicha "Shape Matching and Object Recognition Using Shape Contexts" [5], Suzuki, S., Abe, K., "Topological Structural Analysis of Digitized Binary Images by Border Following" [6] и Hu "Visual Pattern Recognition by Moment Invariants, IRE Transactions on Information Theory" [7]. Многие подходы и концепции, описанные в приведенных выше статьях были использованы в данной работе.

В связи с тем, что работа с OpenCV ведется на языке программирования C++, для его изучения использовалась книга Роберта Лафоре "Объектно-ориентированное программирование в C++" [8]. В ней хорошо описан синтаксис языка, основные понятия, отлично подходит для начинающих.

Для изучения программирования для платформы iOS были выбраны следующие издания: Apple Inc. "The Swift Programming Language (Swift 2.2)" [11] – официальная книга от компании Apple по языку программирования Swift, на котором ведется основная разработка приложения, а так же

Аарон Хиллегас "Objective-C. Программирование для iOS и MacOS" [9]. В данных книгах описываются основные возможности, концепции и синтаксис языков программирования.

Дополнительная информация по проектированию и разработке мобильных приложений под iOS была взята с официального сайта компании Apple "developer.apple.com" [12], на котором находится наиболее полное описание всех функций и библиотек, а так же шаблонов программирования.

Изучение библиотеки Tesseract велось по официальной документации компании Google представленной в интернете.

Глава 1. Постановка задачи

Целью данной работы является разработка мобильного приложения на операционной системе iOS для распознавания шрифта текста методами компьютерного зрения. В частности был использованы алгоритмы контурного анализа объектов на изображении. Для выполнения данной задачи были выбраны несколько ведущих технологий. Рассмотрим их более подробно.

1.1. Обзор технологий

Основными используемыми технологиями являются: Xcode, OpenCV, Tesseract.

Xcode – интегрированная среда разработки программного обеспечения под iOS и OS X разработанная компанией Apple. В этой среде ведется вся работа по созданию мобильного приложения. Он включает в себя большую часть документации разработчика от Apple и Interface Builder — приложение, использующееся для создания графических интерфейсов. Является самой популярной средой разработки для платформы iOS и OS X. Оперативно обновляется и дополняется при выходе новых версий операционных систем.

Tesseract – библиотека для распознавания текста, разрабатывалась компанией Hewlett-Packard с середины 1980-х по середину 1990-х годов. В 2006 году компания Google купила данную технологию и открыла исходный код. Является очень популярным и мощным инструментом. Поддерживает большое количество языков для распознавания, в том числе и русский. Имеет на выбор несколько внутренних методов, один из которых основан на нейронных сетях. Присутствует алгоритм обучение новым языкам или распознаванию текста с экзотическим шрифтом. Из минусов можно выделить слабую документацию, ограничивающуюся небольшой статьей от компании Google. Существует не официальная версия библиотеки с открытым кодом, написанная на языке программирования Objective-C и оптимизиро-

ванная для платформы iOS. Именно такая версия библиотеки используется в данной работе.

OpenCV - библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом. Может свободно использоваться в академических и коммерческих целях — распространяется в условиях лицензии BSD. Мощнейший и самый популярный инструмент для работы с изображениями и видеопотоками на сегодняшний день. Реализованна на C/C++, но также ведется разработка для Python, Ruby, Java и других языков программирования. Содержит в себе огромное количество функций, имеет прекрасную документацию, постоянно обновляется и дополняется. В данной работе используется C++ версия библиотеки.

Среда разработки Xcode поддерживает установку сторонних библиотек, таким образом не составляет труда подключить к проекту OpenCV и Tesseract. Так же Xcode поддерживает работу с несколькими видами языков программирования. В данной работе используются три языка: Swift, Objective C, C++. Для подключения библиотек использовалась специальная сторонняя программа CocoaPods - менеджер зависимостей для Swift и Objective-C проектов. Данное средство позволяет подключать сторонние библиотеки в проект, а так же следить за их обновлениями или изменениями.

Swift - открытый мультипарадигменный объектно-ориентированный язык программирования общего назначения. Создан компанией Apple в первую очередь для разработчиков iOS и OS X. Первая версия была выпущена в 2014 году. Пришел на замену устаревающему Objective-C. Очень быстрый и легкий в освоении язык программирования. Вся работа с пользовательским интерфейсом, логикой и внутренней структурой написана с использованием Swift.

Objective-C – компилируемый объектно-ориентированный язык программирования, используемый корпорацией Apple. Построен на основе языка C и парадигм Smalltalk. До недавнего времени был основным языком

для разработки программных продуктов для iOS и OS X, пока Apple не выпустила Swift. Swift может использовать рантайм (среда выполнения) Objective-C, что делает возможным использование обоих языков в рамках одной программы. В приложении Objective-C используется для работы с библиотекой Tesseract, а так же является мостом (прослойкой) между Swift и C++.

C++ — компилируемый, статически типизированный язык программирования общего назначения, широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. В приложении C++ используется для работы с библиотекой OpenCV.

Необходимость применения Objective-C в качестве моста обуславливается тем, что не существует прямого контакта из Swift в C++, нет возможности вызова функций напрямую. К счастью, Objective-C такую возможность имеет. Получаем следующую цепочку вызовов функций.

$$func(Swift) \mapsto func(Objective - C) \mapsto func(C++)$$

Таким образом осуществляется связь между всеми технологиями в рамках одного проекта.

1.2. База шрифтов

Для реализации данной работы требовалось создать базу эталонных изображений каждого символа (буквы и цифры) в верхнем и нижнем регистрах в различных шрифтах. Создание такой базы было одной из самых больших трудностей, так как готовые базы не подходили по ряду причин. Первая из них это ограниченность в объеме. Вторая - сложный для переработки формат изображений. В силу вышеперечисленных причин была создана подпрограмма, которая создает такие эталонные изображения и сохраняет их отдельно, но до конца процесс автоматизировать не удалось, многое делалось вручную. Из-за этого в данной работе присутствует база только из 7 различных шрифтов, но в дальнейшем ее можно увеличить

до любого необходимого предела. В силу структуры алгоритма на входное изображение налагается некоторые условия. Символы текста не должны соприкасаться, чтобы можно было выделить отдельную букву. Существует целый кластер шрифтов, которые не попадают под это условие. Перечислим какие шрифты присуствуют в базе.

1. Arial
2. Calibri
3. Times New Roman
4. Cambria
5. Candara
6. Javanese Text
7. Mongolian Baiti

1.3. Основные понятия

Рассмотрим основные понятия и определения, которые используются в данной работе.

1. Растовое изображение - изображение, представляющее собой набор пикселей.
2. Пиксель - наименьший логический элемент двумерного цифрового изображения, характеризуемый определённым цветом, яркостью и, возможно, прозрачностью.
3. Размер изображения в пикселях - количество пикселей по ширине и высоте.
4. Цветовое пространство - модель представления цвета, основанная на использовании цветовых координат. В данной работе используются изображения в RGB.

5. RGB - трёхмерное цветовое пространство, где каждый цвет описан набором из трёх координат — каждая из них отвечает компоненте цвета в разложении на красный, зелёный и синий цвета.

6. Цифровой шум — дефект изображения, заметен на изображении в виде наложенной маски из пикселей случайного цвета и яркости.

7. Маска - набор весовых коэффициентов, заданных во всех точках окрестности, симметрично окружающих рабочую точку изображения. Распространенным видом окрестности, часто применяемым на практике, является квадрат размером n на n (n - нечетное) с рабочим элементом в центре.

Глава 2. Описание алгоритма

Приложение, реализованное в ходе данной работы, в качестве начальных данных получает изображение с камеры или из памяти устройства. Основной идеей, как распознать шрифт текста, является контурный анализ. Контуры символов с входного изображения будут сравниваться в контурами букв эталонных изображений в разных шрифтах. Перед выделением контуров следует подготовить изображение.

2.1. Предобработка изображения

Опишем алгоритм предварительной обработки изображения.

Шаг 1. Выделение текста. На входном изображении пользователь выбирает область с текстом, шрифт которого нужно распознать. Выбранный текст должен быть одного шрифта. Программа обрезает исходное изображение по контуру выбранной области. Выделение текста реализовано функциями стандартной библиотеки UIKit, встроенной в Xcode.

Шаг 2. Перевод из UIImage в Mat. Для работы с изображениями при программировании под iOS используется стандартный класс UIImage. UIImage позволяет хранить в себе изображения различных форматов (jpeg, png и др.) и содержит в себе большое количество функций для работы с ними. Входное изображение с камеры или из памяти устройства хранится в виде объекта класса UIImage. Однако, так как распознавание шрифта в дальнейшем ведется с помощью сторонней библиотеки OpenCV, следует перевести исходное изображение в формат Mat, который является основным контейнером для хранения и обработки изображений в OpenCV. Класс Mat представляет объект содержащий два вида информации: заголовок матрицы (содержит размерность матрицы, метод используемый для хранения, адрес хранения и т.д.), а также саму матрицу, в которой содержатся значения пикселей.

Шаг 3. Перевод изображения в градации серого. С помощью функции cvtColor() из библиотеки OpenCV переводим изображение из RGB в

градации серого. Данный перевод происходит по формуле:

$$RGB[A]\ to\ Grey: \quad Y \leftarrow 0.299 \cdot R + 0.589 \cdot G + 0.114 \cdot B,$$

где A – матрица изображения, Y – выходное значение пикселя, R – значение интенсивности красного канала, G – значение интенсивности зеленого канала, B – значение интенсивности синего канала

Шаг 4. Фильтрация шумов. Входное изображение может содержать больше количество шума. Для избавления от него используется медианный фильтр. Принцип его действия прост: он заменяет значение пикселя на медиану значений соседних от него пикселей. Тем самым происходит усреднение значений всех пикселей, то есть избавление от шумов (рис. 1). В OpenCV медианный фильтр реализован в виде функции medianBlur(). Матрица окрестности имеет размерность 5 на 5.

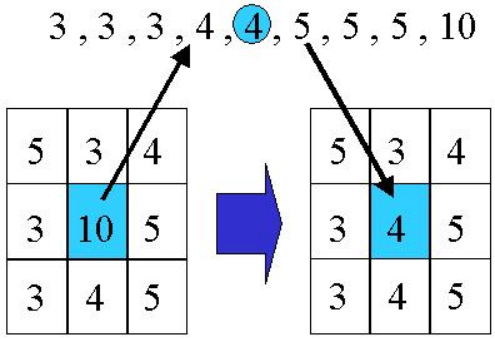

Рис. 1: Процесс усреднения значений пикселей

Удаление шумов позволит в дальнейшем получить лучшие результаты при нахождении контуров.

Шаг 5. Бинаризация изображения. После фильтрации произведем бинаризацию изображения с помощью функции thresholdMat(). Каждый пиксель имеет значение интенсивности от 0 до 255. Возьмем пороговое значение 127. Если значение интенсивности пикселя меньше 127, тогда присваиваем ему значение 0, иначе 255. На выходе мы получаем черно-белое изображение. Бинаризация улучшит результаты работы детектора Кэнни и уменьшит вычислительные затраты.

На данном этапе программа заканчивает предварительную обработку изображения.

2.2. Выделение контуров

Дальнейшая задача состоит в том, чтобы отдельно выделить каждую букву на изображении. Для этого используется следующие шаги.

Шаг 6. Выделение границ детектором Кэнни. Детектор Кэнни (оператор Кэнни, алгоритм Кэнни) — это алгоритм обнаружения границ объектов на изображении. Был создан в 1986 году Джоном Кэнни, и до сих пор трудно найти другой детектор, кроме частных случаев, который работал бы качественно лучше. Из-за этого имеет широчайшее распространение и применяется повсеместно. В библиотеке OpenCV детектор Кэнни реализован в функции Canny(). Кратко рассмотрим этапы работы алгоритма.

1. К изображению применяется фильтр Гаусса с маской K. Происходит размытие изображения, тем самым удаляется оставшийся шум.

$$K = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}.$$

2. Для каждого пикселя изображения применяется оператор Собеля, который вычисляет приближенное значение градиента яркости изображения. Используются следующие маски:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \quad G_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}.$$

Далее вычисляется значение и направление градиента:

$$G = \sqrt{G_x^2 + G_y^2}\,, \quad \theta = Arctan(\frac{G_y}{G_y}).$$

3. Подавление немаксимумов. Пикселями границ объявляются пиксели,

в которых достигается локальный максимум градиента в направлении вектора градиента. Значение направления должно быть кратно 45°.

4. Двойная пороговая фильтрация. Применяется, чтобы определить находится или нет граница в данной точке изображения. Чем меньше порог, тем больше границ будет находиться, но тем более восприимчивым к шуму станет результат, выделяя лишние данные изображения. Высокий порог может проигнорировать слабые края или получить границу фрагментами. Выделение границ Кэнни использует два порога фильтрации: если значение пикселя выше верхней границы – он принимает максимальное значение (граница считается достоверной), если ниже – пиксель подавляется. Точки со значением, попадающим в диапазон между порогов, принимают фиксированное среднее значение. В работе пороговые значения были найдены экспериментально.

Шаг 7. Нахождение контуров. После получения изображения с выделенными границами детектором Кэнни используется функция findContours() из библиотеки OpenCV, чтобы найти контуры объектов. На выходе функции мы получаем набор контуров и матрицу иерархий. Каждый контур представляет собой массив точек с координатами X и Y, принадлежащий этому контуру. Матрица иерархий содержит значения позволяющие определить какой контур вложен, а какой нет. В основе работы findContours() лежит алгоритм Suzuki-Abe [6].

Шаг 8. Отрисовка контуров. Функция drawContours() позволяет отобразить на изображении найденные ранее контуры.

Шаг 9. Выделение объектов по контурам. Имея набор точек для каждого контура можно определить наименьший прямоугольник, который будет содержать в себе область внутри контура. Найти такой прямоугольник можно с помощью функции boundingRect(). Прямоугольник будет опять же представлен в виде набора точек его контура. Используя эту информацию, выделяем на изображении контур каждого прямоугольника и сохраняем внутреннюю область в виде нового изображения.

2.3. Получение набора изображений символов

После проделанной работы мы имеем начальное изображение текста и отдельные вырезанные изображения объектов с изображения.

Для распознавания текста на начальном изображении используется библиотека Tesseract. С ее помощью распознаем текст и сопоставляем каждому символу изображения объектов, отсортированных по местоположению на начальном изображении слева на право. После этого вся информация выводится на отдельный экран приложения и пользователь может оценить правильность сопоставления. Если появилась ошибка, есть возможность поменять значение буквы и удалить изображение, на котором объект не является буквой или цифрой. Наглядный пример представлен в главе 3.

2.4. Сравнение контуров

После выделения символов из общей массы объектов, контуры которых были выделены на начальном изображении, программа выбирает для каждого символа набор его изображений в разных шрифтах.

В итерационном порядке происходят шаги $1 - 7$ для всех эталонных изображений символов каждого шрифта и входных изображений символов. Таким образом формируется массив контуров для каждой буквы или цифры в разном шрифте. Контур входного символа сравнивается с каждым элементом этого массива. Стоит отметить, что между 5 и 6 шагами дополнительно происходит масштабирование эталонного изображения символа в каком-либо шрифте с помощью функции resize(). Эталонное изображение сводится к размеру входного изображения. Размеры варьируются в зависимости от символа. Данную процедуру необходимо проводить для лучшей работы контурного анализа. В библиотеке OpenCV есть две функции которые реализуют сравнение двух контуров: matchShapes() и computeDistance(). Кратко рассмотрим работу обеих функций.

Функция matchShapes() сравнивает два контура A и B и принимает

значение от 0 до бесконечности. Чем выше ее значение, тем больше различие между двумя контурами. Работа функции основана на нахождении Хью моментов контура. Рассмотрим кратко алгоритм работы. Существует три формулы для подсчета выходного значения (в работе используется формула 3).

$$I_1(A,B) = \sum_{i=1\ldots7} |\frac{1}{m_i^A} - \frac{1}{m_i^B}|,$$

$$I_2(A,B) = \sum_{i=1\ldots7} |m_i^A - m_i^B|,$$

$$I_3(A,B) = \max_{i=1\ldots7} \frac{|m_i^A - m_i^B|}{|m_i^A|},$$

где m_i вычисляются следующим образом:

$$m_i^A = sing(h_i^A) \cdot \log h_i^A,$$

$$m_i^B = sing(h_i^B) \cdot \log h_i^B.$$

Здесь h_i – Хью моменты, вычисляемые по формулам:

$$h_1 = \eta_{20} + \eta_{02},$$
$$h_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2,$$
$$h_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2,$$
$$h_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2,$$
$$h_5 = (\eta_{30} - \eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] +$$
$$+(3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{03} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2],$$
$$h_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] +$$
$$+4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}),$$
$$h_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] +$$
$$+(\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{03} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2],$$

где

$$\eta_{ij} = \frac{\mu_{ij}}{\alpha_{ij}^{\frac{i+j}{2}+1}}$$

– нормированные центральные моменты в случае растрового изображения. Для их вычисления сначало необходимо найти пространственные моменты α_{ij} и центральные моменты μ_{ij}:

$$\alpha_{ij} = \sum_{x,y}(array(x,y)\ x^j y^i),$$
$$\mu_{ij} = \sum_{x,y}(array(x,y)(x-\bar{x})^j(y-\bar{y})^i),$$

где x, y – координаты пикселя, $\bar{x}$ и $\bar{y}$ – центр масс:

$$\bar{x} = \frac{\alpha_{10}}{\alpha_{00}}, \quad \bar{y} = \frac{\alpha_{01}}{\alpha_{00}}.$$

На практике использование моментов изображения не всегда дает точный результат при сравнении контуров, так как теряется информация о точной форме объекта.

Принцип работы функции computeDistance() схож с matchShapes(). На вход подается два контура, а на выходе численное значение. Чем больше число, тем меньше похожи контура. В основе алгоритма лежит сравнение точек контуров между собой (рис. 2) и (рис. 3). Отыскиваются похожие точки и строится гистограмма различий, по которой считается расстояние (рис. 4). Подробная реализация алгоритма, заложенного в основе функция computeDistance() описана в статье Belongie S., Malik J., Puzicha J. "Shape matching and object recognition using shape contexts" [5].

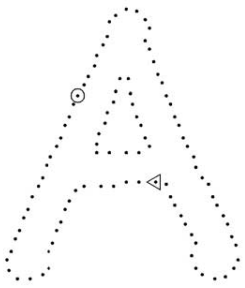

Рис. 2: Первый контур

Рис. 3: Второй контур

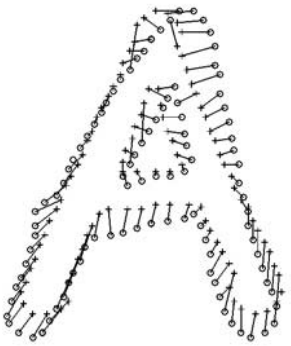

Рис. 4: Положения точек

Обе функции были реализованы в приложении, в частности, для сравнения их работы, и зачастую дают довольно близкие результаты, но предпочтение отдается computeDistance(), как более современному и точному подходу.

Общий результат вычисляется с помощью принципа "голосования". Для каждого входного символа мы получаем набор из семи числовых значений, характеризующих принадлежность к тому или иному шрифту. Выбираем из них наименьшее число и "голосуем" тремя баллами за тот шрифт, при сравнении с которым было получено данное значение. Затем переходим к следующему числу по возрастанию, и "голосуем" за соответствующий шрифт двумя баллами. Далее выбираем еще одно число по возрастанию и "голосуем" за соответствующий шрифт одним баллом. Такая процедура происходит для всех входных символов. На выходе проверяется количество баллов, отданных за каждый шрифт, и выбираются три из них с наилучшими результатами. Затем в порядке убывания баллов показываются названия шрифтов на экране, тем самым показывая не только наиболее вероятный шрифт текста на входном изображении, но и ближайшие к нему два других.

Глава 3. Работа приложения

Для удобства использования приложения пользователем и демонстрации наглядной работы был спроектирован интерфейс. Все взаимодействие с приложением осуществляется с помощью нажатий на экран устройства.

3.1. Структура программы

Работу с приложением можно разбить на несколько этапов. Рассмотрим их подробнее.

Этап 1. При включении приложения пользователь видит начальное окно с названием "Image" (рис. 5). Единственным доступным элементом интерфейса является активная кнопка "Snap/Upload". При ее нажатии открывается меню выбора изображения из памяти устройства или создание снимка на камеру. (рис. 6).

Рис. 5: Стартовое окно

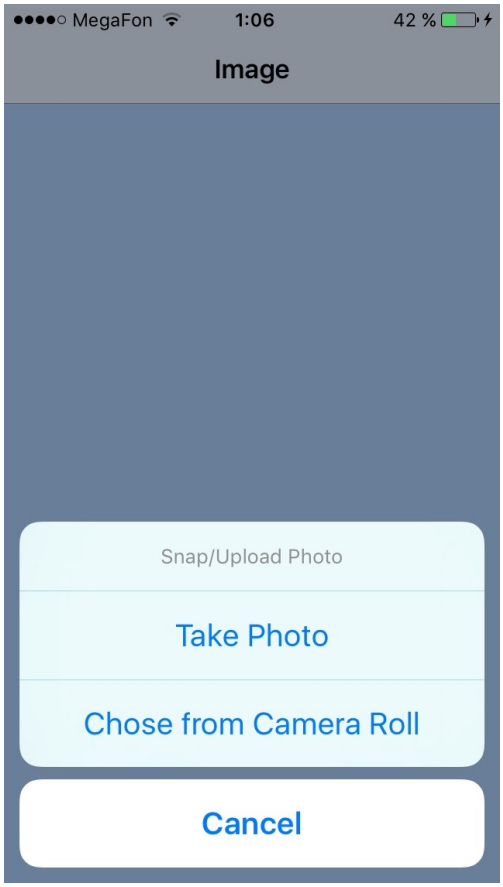

Рис. 6: Выбор изображения

При нажатии на "Take Photo" открывается стандартный интерфейс камеры, где можно сделать фотографию. При нажатии "Chose from Camera Roll" открывается фотогалерея устройства, откуда пользователь может выбрать изображение, сделанное заранее.

После выбора изображения оно появляется на экране и становится активна кнопка "Cut Image"(рис. 7). При нажатии на нее откроется новое окно, где можно с помощью касаний менять масштаб рамки для выделения нужной части изображения (рис. 8) После нажатия на кнопку "Done" в правом верхнем углу переходим на начальный экран. Кнопка "Recognize"становится активной (рис. 9).

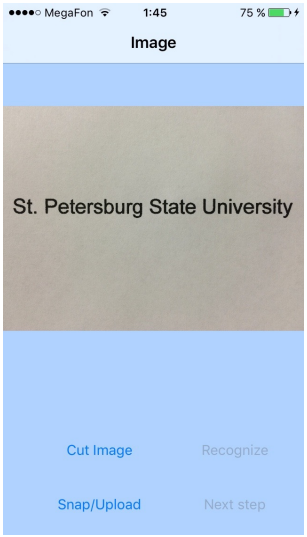

Рис. 7: Окно с изображением

Рис. 8: Выделение текста

Рис. 9: Окно с текстом

После нажатия на кнопку Recognize внутри программы происходит основная работа по выделению символов и распознаванию текста, описанная в разделах 1-3 второй главы. Изображение проходит предобработку, выделяются контуры объектов (красные линии), а так же области, которые их содержат (синие прямоугольники). Так же изображение передается для работы в библиотеку Tesseract, где на нем выделется текст. Результат распознавания текста отображается на экране. Все эти данные наносятся на начальное изображение и показываются пользователю. Кнопка "Next step"становится активной (рис. 10).

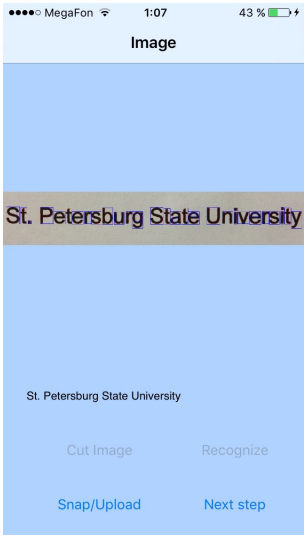

Рис. 10: Выделение контуров

Рис. 11: Список объектов

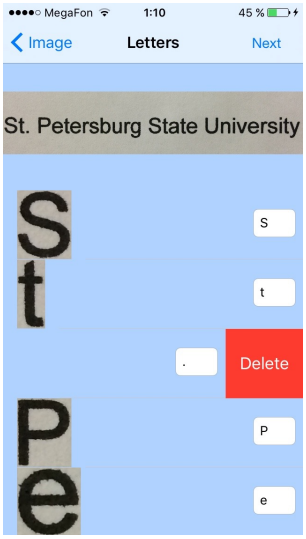

Рис. 12: Удаление объекта

При нажатии на кнопку "Next step"осуществляется переход на новое окно с названием "Letters". (рис. 11) На нем в верхней части находится начальное изображение, а чуть ниже список из изображений отдельных символов и других обьектов, контуры которых получилось распознать. В данном окне пользователь имеет возможность проверить правильность определения букв и цифр на изображении, их символьные значения и изменить их в случае ошибки распознавания. Если буква была определена неправильно, то по нажатию на белую область справа от буквы можно ввести с клавиатуры нужный символ. Если же был определен контур не нужного обьекта, можно полностью удалить его из списка движением по экрану справа на лево. Например, был выделен контур точки после букв "St". (рис. 12) Этот контур не является буквой или цифрой, следовательно он подвергается удалению. После проверки пользователем всего списка, можно нажать на кнопку “Next” в правом верхнем углу. Произойдет переход на новое окно "Fonts"(рис. 13). На данном окне отображается начальное изображение и список из трех названий максимально похожих шрифтов. Шрифт под номером один - максимально вероятный шрифт текста входного изображения. Два последующих - ближайшие по сходству шрифты.

Рис. 13: Вывод результата

Заключение

В ходе данной работы было создано приложение для платформы iOS для распознавания шрифта текста на изображении. Также был получен колоссальный опыт в самой разработке мобильных приложений под iOS и опыт в использовании библиотек OpenCV и Tesseract.

Программный код, реализованный в результате данной работы, является хорошей базой для дальнейшего развития и улучшения программы. Алгоритм сравнения контуров не зависит от размера базы шрифтов, следовательно имеется возможность расширения базы до нужного предела. При дальнейшем развитии продукта это позволит использовать программу массовому пользователю. Возможно будет создана коммерческая версия для магазина iOS приложений App Store.

Подходы контурного анализа, рассмотренные в данной работе можно использовать и для других задач, связанных с обработкой изображений, сравнений объектов на снимках.

Список литературы

[1] Garcia G. B., Suarez O. D., Aranda J. L. E. Learning Image Processing with OpenCV. ISBN-13: 978-1-78328-765-9. Birmingham: Packt Publishing, 2015. 319 p.

[2] Bradski G. Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly. 2008. 543 p.

[3] Forsyth D., Ponce J. Computer vision: A modern approach // Prentice Hall. 2002

[4] Dawson-Howe K. A Practical Introduction to Computer Vision with OpenCV. John Wiley&Sons Ltd, 2014. 235 p.

[5] Belongie S., Malik J., Puzicha J. Shape matching and object recognition using shape contexts // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2002. Vol. 24. No 4. P. 509 – 522.

[6] Suzuki S, Abe K. Topological Structural Analysis of Digitized Binary Images by Border Following // CVGIP. 1985. Vol. 1. P. 32-46.

[7] Hu M.K. Visual pattern recognition by moment invariants // IRE Transactions on Information Theory. 1962. Vol. 8. - P. 179-187.

[8] Лафоре Р. Объектно-ориентированное программирование в C++. СПб.: Питер, 2004. 923 с.

[9] Хиллегасс А. Objective-C Программирование для iOS и MacOS. СПб: Питер, 2012. 304 с.

[10] OpenCV documentation. htt://docs.opencv.org/.

[11] The Swift Programming Language Reference: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/SwiftProgram

[12] iOS Developer Library. https://developer.apple.com/.

Приложение А.

Код перевода из UIImage в Mat и обратно.

```
//Перевод из UIImage в Mat
+ (cv::Mat)cvMatFromUIImage:(UIImage *)image {
   CGColorSpaceRef colorSpace = CGImageGetColorSpace(image.CGImage);
   CGFloat cols = image.size.width;
   CGFloat rows = image.size.height;
   cv::Mat cvMat(rows, cols, CV_8UC4);
   CGContextRef contextRef = CGBitmapContextCreate(cvMat.data,
                             cols, rows, 8, cvMat.step[0],
                             colorSpace, kCGImageAlphaNoneSkipLast|
                             kCGBitmapByteOrderDefault);

   CGContextDrawImage(contextRef, CGRectMake(0, 0, cols, rows),
                      image.CGImage);
   CGContextRelease(contextRef);
   return cvMat;
}

//Перевод из Mat в UIIMage
+ (UIImage *)UIImageFromCVMat:(cv::Mat)cvMat {
   NSData *data = [NSData dataWithBytes:cvMat.data length:
                  cvMat.elemSize()*cvMat.total()];
   CGColorSpaceRef colorSpace;
   if (cvMat.elemSize() == 1) {
      colorSpace = CGColorSpaceCreateDeviceGray();
   } else {
      colorSpace = CGColorSpaceCreateDeviceRGB();
   }
   CGDataProviderRef provider = CGDataProviderCreateWithCFData(
```

```
                                    (__bridge CFDataRef)data);
    CGImageRef imageRef = CGImageCreate(cvMat.cols,
                                        cvMat.rows,
                                        8,
                                        8 * cvMat.elemSize(),
                                        cvMat.step[0],
                                        colorSpace,
                                        kCGImageAlphaNone|
                                        kCGBitmapByteOrderDefault,
                                        provider,
                                        NULL,
                                        false,
                                        kCGRenderingIntentDefault);
    UIImage *finalImage = [UIImage imageWithCGImage:imageRef];
    CGImageRelease(imageRef);
    CGDataProviderRelease(provider);
    CGColorSpaceRelease(colorSpace);
    return finalImage;
}
```

Приложение В.

Код выделения символов на входном изображении.

```
//Структура для сортировки контуров слево - направо
struct contour_sorter_left
{
   bool operator ()( std::vector<cv::Point>& a,
                     std::vector<cv::Point> & b )
   {
      cv::Rect ra(boundingRect(a));
      cv::Rect rb(boundingRect(b));
      return ra.x<rb.x;
   }
};

//Нарезка букв с входного изображения
+ (NSMutableArray*) cutCharacters: (UIImage*) myImage {

   //Массив для нарезанных символов
   NSMutableArray* charactersImageArray = [[NSMutableArray alloc]
   init];

   //Перевод UIImage в cv::mat
   cv::Mat cvMat = [self cvMatFromUIImage:myImage];
   cv::Mat cvMatWithContours = [self cvMatFromUIImage:myImage];

   //Cv::mat хранящие в себе изображение
   cv::Mat grayMat, blurMat, thresholdMat, cannyMat, contoursMat;

   //Перевод в оттенки серого
   cv::cvtColor(cvMat, grayMat, CV_BGR2GRAY);
```

```
//Медианный фильтр шумов
cv::medianBlur(grayMat, blurMat, 5);

//Бинаризация изображения
cv::threshold(blurMat, thresholdMat, 127, 255,
CV_THRESH_BINARY_INV);

//Выделение контуров детектором границ Canny
cv::Canny(thresholdMat, cannyMat, 10, 200);

//Вектор контуров
std::vector < std::vector<cv::Point> > contours;

//Вектор иерархии контуров
std::vector<cv::Vec4i> hierarchy;

//Выделение границ
findContours(cannyMat  , contours, hierarchy, CV_RETR_TREE,
             CV_CHAIN_APPROX_SIMPLE, cv::Point(1,1));

//Сортировка контуров слева направо
std::sort(contours.begin(), contours.end(),
contour_sorter_left());

//Красный цвет для контуров
cv::Scalar contourColor(255, 0, 0);

//Синий цвет для прямоугольников
cv::Scalar rectangleColor(0,0,255);
```

```
//Вектор прямоугольников
std::vector<cv::Rect> boundRect(contours.size());

for( int i = 0; i < contours.size(); i++ ) {

   //Построение прямоугольников по новым контурам
   boundRect[i] = boundingRect( cv::Mat(contours[i]) );
   boundRect[i].x -=4;
   boundRect[i].y -=4;
   boundRect[i].height +=8;
   boundRect[i].width +=8;
}

//Группируем прямоугольники (удаляем лишние)
cv::groupRectangles(boundRect, 1);

for( int i = 0; i< contours.size(); i++ ) {

   //Отрисовка контуров
   cv::drawContours(cvMatWithContours, contours, i,
                    contourColor, 1, 8, hierarchy);

   //Отрисовка прямоугольников
   rectangle(cvMatWithContours, boundRect[i].tl(),
             boundRect[i].br(), rectangleColor,
             2, 8, 0 );
}

//Начальное изображение
[charactersImageArray addObject:
   [self UIImageFromCVMat:cvMat]];
```

```
    //Конечное изображение с контурами и прямоугольниками
    [charactersImageArray addObject:
        [self UIImageFromCVMat:cvMatWithContours]];

    //Бинарное изображение
    [charactersImageArray addObject:
        [self UIImageFromCVMat:thresholdMat]];

    //Нарезка символов по контурам
    for( int i = 0; i < boundRect.size(); i++ ) {
        cv::Mat croppedImage;
        croppedImage = cvMat(boundRect[i]).clone();
        [charactersImageArray addObject:
            [self UIImageFromCVMat:croppedImage]];
    }
    return charactersImageArray;
}
```

Приложение С.

Код сравнения контуров.

```
+ (NSMutableArray*) compareImages: (NSMutableArray*) myImages {

   //Выходной массив данных по сравнению
   NSMutableArray* outputArray = [[NSMutableArray alloc] init];

   for (int i = 0; i<=myImages.count-8; i += 8) {

      //Матрицы для фильтрации изображения
      cv::Mat mainMat, mainGrayMat, mainBlurMat,
              mainThresholtMat, mainCannyMat;

      cv::Ptr <cv::ShapeContextDistanceExtractor> mySCDE =
               cv::createShapeContextDistanceExtractor();

      //Изображение распознаваемой буквы
      UIImage* mainImage = myImages[i];

      //Перевод UIImage в cv::Mat
      mainMat = [self cvMatFromUIImage:mainImage];

      //Перевод изображения в оттенки серого
      cv::cvtColor(mainMat, mainGrayMat, CV_BGRA2GRAY);

      //Медианный фильтр
      cv::medianBlur(mainGrayMat, mainBlurMat, 11);

      //Перевод изображения в черно-белое
      cv::threshold(mainBlurMat, mainThresholtMat, 127,
```

```
                255, CV_THRESH_BINARY);

//Вектор контуров изображения
std::vector < std::vector<cv::Point> > mainContours;

//Вектор иерархии контуров
std::vector<cv::Vec4i> mainHierarchy;

//Нахождение контуров изображения
findContours(mainMorphologyMat, mainContours,
             mainHierarchy,
             CV_RETR_LIST, CV_CHAIN_APPROX_SIMPLE,
             cv::Point(0,0));

for (int j = 1; j<=7; j++) {

   //Проверка верного индекса
   if ((i!=0) && (i%8)-(i/8) == 0) {
      j++;
   }

   //Матрицы для фильтрации изображения
   cv::Mat cvMat, grayMat, blurMat, thresholtMat;

   //Изображение для сравнения
   UIImage* image = myImages[i+j];

   //Перевод UIImage в cv::mat
   cvMat = [self cvMatFromUIImage:image];

   //Изменение размера изображения для сравнения
```

```
cv::resize(cvMat, afterResizeMat, mainMat.size());

//Перевод изображения в оттенки серого
cv::cvtColor(afterResizeMat, grayMat, CV_BGRA2GRAY);

//Медианный фильтр
cv::medianBlur(grayMat, blurMat, 11);

//Перевод изображения в черно-белое
cv::threshold(blurMat, thresholtMat, 127, 255,
              CV_THRESH_BINARY);

//Вектор контуров изображения
std::vector < std::vector<cv::Point> > contours;

//Вектор иерархии контуров
std::vector<cv::Vec4i> hierarchy;

//Нахождение контуров
findContours(morphologyMat, contours, hierarchy,
             CV_RETR_LIST, CV_CHAIN_APPROX_SIMPLE,
             cv::Point(0,0));

//Сравнение контуров
float distanceComputeDistance = mySCDE->
                  computeDistance(
                  mainContours[mainContours.size()-2],
                  contours[contours.size()-2]);
float distanceMatchShapes = cv::matchShapes(
                  mainContours[mainContours.size()-2],
                  contours[contours.size()-2],
```

```
                                CV_CONTOURS_MATCH_I1, 1);

            [outputArray addObject:[NSNumber numberWithFloat:
                                    distanceComputeDistance]];
        }
    }
    return outputArray;
}
```