# ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ РАЗРАБОТКИ ПРОГРАММНО-АЛГОРИТМИЧЕСКОГО КОМПЛЕКСА АНАЛИЗА ОКТ-ИЗОБРАЖЕНИЙ СЕТЧАТКИ И ЭКСПЕРИМЕНТАЛЬНАЯ АПРОБАЦИЯ РАЗРАБОТКИ

## Выбор инструментов разработки

Задачу разработки программно-алгоритмического комплекса анализа ОКТ-изображений сетчатки возможно решить, используя различные наборы технологий. Однако, вдумчивый и обоснованные выбор инструментария способствует упрощению и ускорению разработки программно-алгоритмического комплекса.

### Обоснования выбора языка программирования

Основным языком программирования, использующимся для разработки систем обработки изображения, был выбран язык Python. Язык Python обладает лаконичным и легко читаемым синтаксисом, что позволяет легче вносить изменения в код проекта на всех этапах разработки. Также, Python обладает обширным набором специализированных библиотек, упрощающих обработки изображений и работу c проектами, использующими машинное обучение.

Также, использование языка Python позволяет обеспечить кроссплатформенность разрабатываемого программно-алгоритмического комплекса, так как Python работает на платформах Windows, Linux и macOS, суммарно охватывающих наибольшее количество пользователей ЭВМ.

### Выбор библиотек для обработки изображений

На данной этапе разработки программно-алгоритмического комплекса было решено использовать такие библиотеки, как:

* OpenCV – библиотека компьютерного зрения, содержащая более 2500 алгоритмов для работы и обработки изображений. Использование данной библиотеки позволяет упросить процесс разработки модулей предобработки, позволяя использовать уже готовые алгоритмы обработки изображений.
* Ultralytics – библиотека компьютерного зрения на основе искусственного интеллекта. Использование данной библиотеки позволяет упростить процесс обучения и использования моделей архитектуры YOLO.
* MatPlotLib – библиотека визуализации данных в виде статических графиков. Данная библиотека позволяет упросить визуализацию результатов работы каждого блока модуля, что требуется для контроля качества получаемых результатов обработки.

### Выбор инструментов разработки пользовательского интерфейса

В целях обеспечения кроссплатформенного использования разрабатываемого программно-алгоритмического комплекса пользовательский интерфейс было решено разработать в виде вэб-приложения. Данный подход позволяет использовать разрабатываемый программно-алгоритмический комплекс на любой платформе, имеющий какой-либо браузер.

Для разработки пользовательского интерфейса был использован язык TypeScript. TypeScript является расширенной версией JavaScript. TypeScript позволяет повысить надёжность, читаемость и поддерживаемость кода, снизить вероятность ошибок и облегчить масштабирование проекта.

### Выбор инструментов для разработки серверной инфраструктуры

Пользовательский интерфейс предоставляет визуальные инструменты взаимодействия с разрабатываемым программно-алгоритмическим комплексом. Вычислительная часть разрабатываемого комплекса расползается в отдельном логическом блок – сервере.

Для разработки серверной инфраструктуры была выбрана библиотека Flask. Flask является минималистичным веб-фреймворком. Он предоставляет простую и гибкую структуру, позволяющую быстро разработать как прототип, так и полнофункциональное веб-приложение.

## Разработки модуля предобработки ОКТ-изображений

В листинге 1 представлен пример программы, выполняющей предобработку обрабатываемого ОКТ-изображения.

Листинг 1 – Программа модуля предобработки

|  |
| --- |
| prep = Preprocessor()  image = cv2.imread(f"{dir}/{file}")      converted\_image = prep.convert\_to\_gray(image)      filtered\_image = prep.smooth\_image(converted\_image)      sobel\_image = prep.sobel(filtered\_image)      ret2, binary = prep.threshold(sobel\_image)      element = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (5, 5))      binary = prep.morph\_open(binary, element)      binary = prep.morph\_close(binary, element)      contours = prep.get\_contours(binary)      resized = prep.resize\_image(image, contours) |

Программа, представленная в листинге 1, является последовательностью алгоритмических преобразования обрабатываемого ОКТ-изображения сетчатки, представленных на рисунке [].

Каждое преобразование, выделенное на рисунке [] в отдельный блок, далее рассматривается отдельно.

### Реализация преобразования к чёрно-белому изображению

В листинге 2 представлен пример кода, преобразующего обрабатываемого ОКТ-изображение к чёрно-белому.

Листинг 2 – Преобразование ОКТ-изображения к чёрно-белому

|  |
| --- |
| def convert\_to\_gray(self, image):          return cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY) |

Для реализации преобразования ОКТ-изображения к чёрно-белому была использована функция библиотеки OpenCV cvtColor(image, cv2.COLOR\_RGB2GRAY).

Функция принимает на вход прочитанное изображение в формате RGB и флаг, определяющий тип преобразования. В данном случае указывается флаг cv2.COLOR\_RGB2GRAY, который позволяет преобразовать цветное изображение в оттенках RGB в одноканальное изображение в градациях серого.

Функция convert\_to\_gray возвращает изображение, преобразованное в градации серого.

### Реализация размытия изображения

В листинге 3 представлен пример кода, реализующего размытие изображения.

Листинг 3 – Реализация размытия изображения

|  |
| --- |
| def smooth\_image(self, image):          image\_medianBlur = cv2.medianBlur(image, 21)          image\_biFilter = cv2.bilateralFilter(image\_medianBlur, 11, 150, 150)          return image\_biFilter |

В реализации блока «размытие изображения» использовались две функции библиотеки OpenCV:

* medianBlur – функция, позволяющая применить к изображению медианный фильтр. Она принимает два аргумента: изображение и размер ядра фильтра (нечётное целое число).
* bilateralFilter – функция, выполняющая билинейную фильтрацию изображения, которая сглаживает изображение, одновременно сохраняя края. Она принимает четыре аргумента: входное изображение, диаметр окна фильтра, сигму цветового пространства и сигму координатного пространства.

В комбинации данные фильтры позволяют эффективно снизить уровень шумов и артефактов на изображении, одновременно сохранив важные структурные элементы.

Функция smooth\_image возвращает сглаженное изображение с сохранёнными границами.

### Реализация выделения краёв оператором Собеля

В листинге 4 представлен пример кода, реализующего выделение краёв изображения, используя оператор Собеля.

Листинг 4 – Реализация выделения краёв оператором Собеля

|  |
| --- |
| def sobel(self, image):          return cv2.Sobel(image, -1, 0, 1, ksize=5) |

Выделение краёв осуществляется при помощи функции cv2.Sobel() из библиотеки OpenCV, которая применяется для вычисления приближённой производной изображения, что позволяет выявить резкие изменения интенсивности – края объектов.

Функция cv2.Sobel принимает обрабатываемое изображение, глубину выходного изображения (значение -1 означает, что глубина результата совпадает с глубиной входного изображения), порядок производной по оси X, порядок производной по оси Y, размер ядра Собеля (должен быть нечётным и положительным).

Функция sobel возвращает изображение, на котором выделены края по вертикальному направлению, полученные в результате применения оператора Собеля.

### Реализация пороговой фильтрации

В листинге 5 приведен пример кода, реализующего пороговую фильтрацию изображения.

Листинг 5 – Реализация пороговой фильтрации изображения

|  |
| --- |
| def threshold(self, image):          return cv2.threshold(image, 100, 255, cv2.THRESH\_BINARY) |

Пороговая фильтрация изображения осуществляется при помощи функции cv2.threshold библиотеки OpenCV. Данная функция применяется для преобразования градационного (оттеночного) изображения в бинарное, то есть состоящее только из двух уровней яркости.

Функция cv2.threshold принимает обрабатываемое изображение, пороговое значение (все пиксели с яркостью выше этого значения будут установлены в заданное максимальное значение), максимальное значение (значение, присваиваемое пикселям, превышающим порог), тип пороговой фильтрации (в данном случае используется cv2.THRESH\_BINARY, при котором пиксели, превышающие порог, получают значение 255, остальные – 0).

Функция threshold возвращает кортеж, состоящий из применённого порогового значения и изображения, преобразованного в бинарный формат на основе заданного порога.

### Реализация морфологических преобразований

В листинге 6 представлен пример кода, реализующего морфологические преобразования изображения.

Листинг 6 – Реализация морфологических преобразований изображения

|  |
| --- |
| element = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (5, 5))  def morph\_open(self, image, element):          binary = cv2.morphologyEx(image, cv2.MORPH\_OPEN, element)          return binary        def morph\_close(self, image, element):          binary = cv2.morphologyEx(image, cv2.MORPH\_CLOSE, element)          return binary |

Морфологические преобразования изображения осуществляются при помощи функции cv2.morphologyEx библиотеки OpenCV. Эти преобразования применяются, как правило, к бинарным изображениям и служат для устранения шумов, разрывов и мелких артефактов, а также для выделения или сглаживания контуров объектов.

Функция cv2.morphologyEx принимает входное изображение, тип морфологической операции, структурный элемент (определяет форму и размер области, по которой применяется операция, и создаётся с помощью функции getStructuringElement).

Морфологическое открытие (MORPH\_OPEN) последовательно применяет операции эрозии и дилатации, эффективно удаляя мелкие шумы.

Морфологическое замыкание (MORPH\_CLOSE), наоборот, сначала выполняет дилатацию, а затем эрозию, что позволяет заполнять небольшие разрывы и пробелы внутри объектов.

В результате применения данных функций возвращается бинарное изображение, преобразованное с использованием операций морфологического открытия и замыкания. Эти преобразования позволяют устранить мелкие шумы и заполнить небольшие пробелы внутри объектов.

### Реализация объединений контуров

В листинге 7 приведен пример кода, реализующего объединение контуров изображения.

Листинг 7 – Реализация объединения контуров

|  |
| --- |
| def get\_contours(self, image):          contours, hierarchy = cv2.findContours(image, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)          approved\_cnts = []          for cnt in contours:              contourArea = cv2.contourArea(cnt)              if (contourArea > 250):                  x, y, w, h = cv2.boundingRect(cnt)                  if (w < image.shape[1] / 4):                      approved\_cnts.append(cnt)                  elif (not self.is\_line(cnt)) :                      approved\_cnts.append(cnt)          return approved\_cnts  def is\_line(self, contour):          [vx, vy, x0, y0] = cv2.fitLine(contour, cv2.DIST\_L2, 0, 0.01, 0.01)          errors = []          for point in contour:              x, y = point[0]              distance = abs(vy \* (x - x0) - vx \* (y - y0))              errors.append(distance)          return max(errors) < 13 # 12 |

Прежде, чем объединить контуры, необходимо найти их и провести фильтрацию с целью выявления контуров, относящихся к морфологической структуре сетчатки.

Для поиска контуров использовалась функция cv2.findContours библиотеки OpenCV. Функция принимает обрабатываемое изображение, тип извлечения и метод аппроксимации контуров. На выходе возвращается список контуров и их иерархия.

После получения всех контуров выполняется их фильтрация по следующим критериям:

* Площадь контура должна превышать 250 пикселей, чтобы отсеять мелкие шумы. Площадь контура высчитывается при помощи функции cv2.contourArea библиотеки OpenCV.
* Ширина ограничивающего прямоугольника должна быть меньше четверти ширины изображения. Ограничивающий прямоугольник высчитывается при помощи функции cv2.boundingRect библиотеки OpenCV.
* Контуры, имеющие форму прямой линии, определяются функцией is\_line и также отбрасываются.

Функция is\_line аппроксимирует контур прямой линией с помощью метода наименьших квадратов cv2.fitLine из библиотеки OpenCV и вычисляет максимальное отклонение точек от этой линии. Если отклонение невелико (менее 13), контур считается линейным и удаляется как неинформативный для анализа.

Функция get\_contours возвращает информативные контуры объектов, относящиеся к морфологической структуре сетчатки.

### Реализация выделения фрагмента изображения

В листинге 8 представлен пример кода, реализующего выделения фрагмента ОКТ-изображения.

Листинг 8 – Реализация выделения фрагмента изображения

|  |
| --- |
| def resize\_image(self, image, contours):          x, y, w, h = cv2.boundingRect(np.vstack(contours))          cropped = image[y:y+h + 30, x + 10:x+w]          return cropped |

Реализация выделения фрагмента изображения осуществляется при помощи использования функции cv2.boundingRect библиотеки OpenCV.

Функция cv2.boundingRect принимает массив точек (в данном случае – объединённый массив всех контуров с помощью np.vstack) и возвращает координаты верхнего левого угла прямоугольника (x, y) и его ширину и высоту (w, h). Далее, с помощью этих координат из исходного изображения вырезается соответствующий фрагмент. Для захвата всей интересующей области применяются небольшие смещения: область расширяется вниз на 30 пикселей и вправо на 10 пикселей, чтобы захватить возможные пограничные элементы.

Функция resize\_image возвращает выделенный фрагмент обрабатываемого ОКТ-изображения.

На рисунке 1 и 2 представлены ОКТ-изображения сетчатки до прохождения через модуль предобработки и после соответственно.

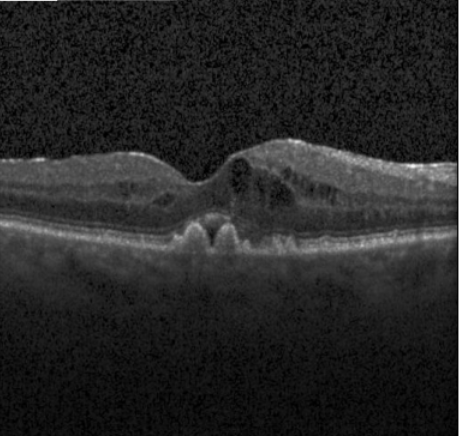


Рисунок 1 – ОКТ-изображение сетчатки до прохождения через модуль предобработки

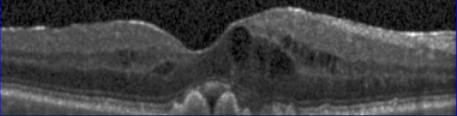


Рисунок 2 – ОКТ-изображения сетчатки после прохождения через модуль предобработки