# 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.

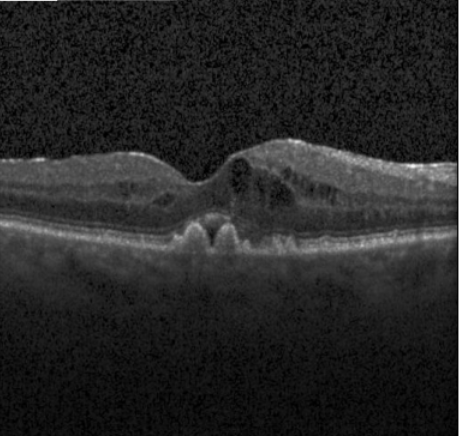


Рисунок 1 – ОКТ-изображение сетчатки до прохождения через модуль предобработки

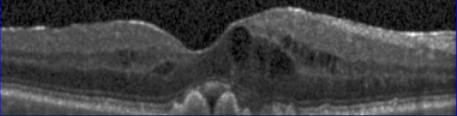


Рисунок 2 – ОКТ-изображения сетчатки после прохождения через модуль предобработки

## Разработка модуля обработки ОКТ-изображений на основе нейронной сети

Основой модуля обработки ОКТ-изображений является модель нейронной сети, сегментирующая и классифицирующая выбранные патологии морфологической структуры сетчатки.

### Выбор модели нейронной сети

В работе рассматривались такие модели нейронных сетей, как: SAM2, DeepLabV3+, YOLOv9, YOLOv11, YOLOv12.

На рисунке 3 приведен графики функции потерь для каждой из рассматриваемых моделей.

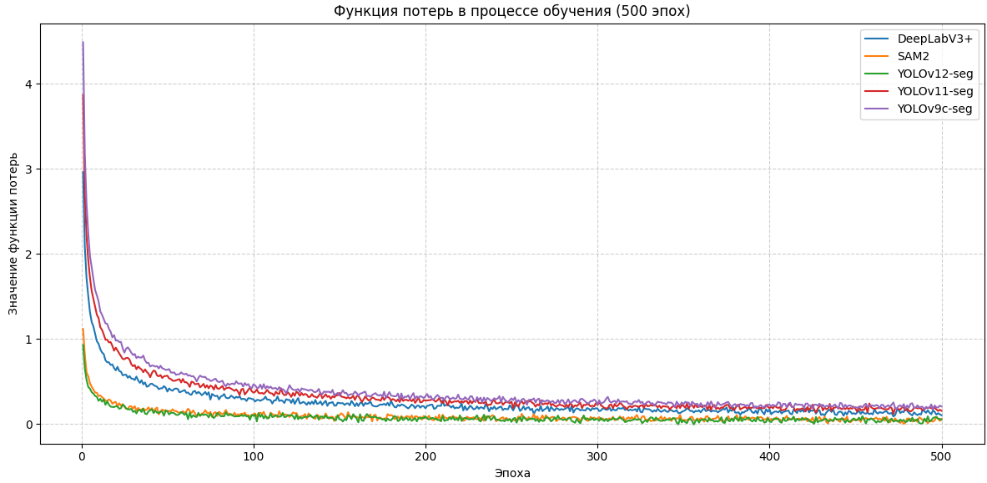


Рисунок 3 – График функции потерь для каждой из рассматриваемых моделей

Анализ графика на рисунке 3 позволяет сделать вывод, что наилучшую динамику сходимости функции потерь продемонстрировали модели YOLOv12-seg и SAM2, у которых значение функции потерь устойчиво снижалось и колебалось вблизи порога 0.5. Это свидетельствует о высокой стабильности и эффективности обучения. Остальные модели – DeepLabV3+, YOLOv11-seg и YOLOv9c-seg – показали удовлетворительные результаты: их функции потерь оставались на значительно более высоком уровне, не опускаясь ниже 1.5, что может указывать на менее качественное обучение или проблемы с адаптацией к специфике задачи.

Для комплексной оценки производительности моделей были рассчитаны следующие метрики:

* Dice Index – отражает степень совпадения предсказанной маски с эталонной и является важным показателем качества сегментации;
* Precision – показывает долю правильно предсказанных объектов среди всех предсказанных;
* Recall – показывает долю реальных объектов, корректно обнаруженных моделью;
* Mean Average Precision && IoU=0.5 ([mAP@0.5](mailto:mAP@0.5)) – средняя площадь под кривой Precision-Recall для всех классов при пороговом значении IoU 0.5, отражающая обобщённую точность модели в задаче обнаружения объектов.

На рисунке 4 представлены метрики оценки производительности каждой из моделей.

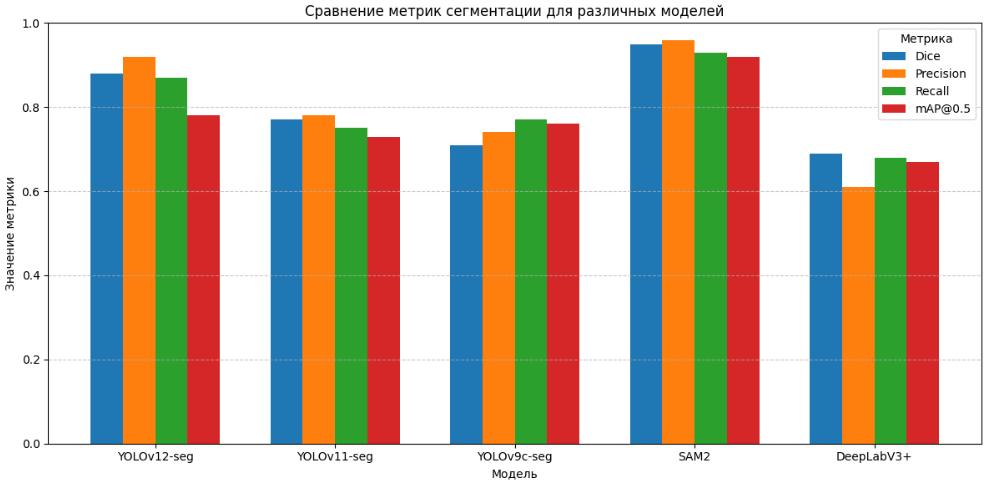


Рисунок 4 – Сравнение метрик сегментации моделей

Анализ графика на рисунке 4 показывает, что наилучший результат производительности модели при решении задачи сегментации патологий морфологической структуры показывает модель нейронной сети SAM2. SAM2 обладает сравнительно высокими показателями метрик Precision и Recall, что говорит о том, что модель в большинстве случаев правильно классифицирует найденную патологию и в большей степени не пропускает патологические объекты морфологической структуры.

Также модель, верно, сегментирует объект относительно структуры сетчатки. Это утверждение подтверждает довольно высокий показатель Dice.

Однако SAM2 является довольно тяжелой моделью и может требовать значительной производительности системы для обеспечения корректной и быстрой работы.

В связи со значительными системными требованиями, предъявляемыми к системе при использовании SAM2, моделью нейронной сети, которая составит основу модуля обработки было выбрана YOLOv12-seg. YOLOv12-seg обладает схожими метриками производительности, которые в перспективе могут быть улучшены путём использования более широкой обучающей выборки и увеличения количества обучающих эпох. Также, подстройка таких гиперпараметров, как: learning\_rate, batch\_size, optimizer – могу помочь улучшить метрики производительности и итоговую адекватность модели.

### Оценка качества выбранной модели

В таблице 1 приведены метрики сегментации каждого класса патологий для модели YOLOv12-seg, полученные при проверке модели на тестовой выборке.

Таблица 1 – Метрики сегментации

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Класс | Precision | Recall | mAP50 | Dice |
| intraretinal\_cyst | 0.925 | 0.943 | 0.956 | 0.938 |
| lamellar\_macular\_rupture | 0.897 | 0.910 | 0.923 | 0.941 |
| neuroepithelium\_detachment | 0.967 | 0.976 | 0.963 | 0.972 |
| retinal\_drusen | 0.883 | 0.924 | 0.941 | 0.952 |
| subretinal\_hyperreflective\_material | 0.949 | 0.963 | 0.952 | 0.963 |
| vitreomacular\_traction | 0.891 | 0.907 | 0.959 | 0.969 |
| vitreous\_detachment | 0.936 | 0.983 | 0.932 | 0.924 |

Анализ данных в таблице 1 показывает, что выбранная модель YOLOv12-seg хорошо справляется с задачей сегментации, однако некоторые классы патологий имеют метрику Precision, значение которой ниже 0.9. Это говорит о том, что модель некорректно классифицирует данные патологии. Негативное влияние на способность модели верно классифицировать патологии может оказывать недостаточная описательная способность обучающей выборки для этих классов патологий. Необходимо увеличить количество объектов обучающей выборки для данных классов патологий.

На рисунке 5, 6 представлены ОКТ-изображения сетчатки до и после прохождения через модуль обработки разрабатываемого программно-алгоритмического комплекса.

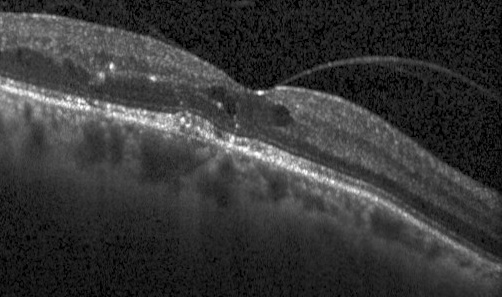


Рисунок 5 – ОКТ-изображения сетчатки до прохождения через модуль обработки

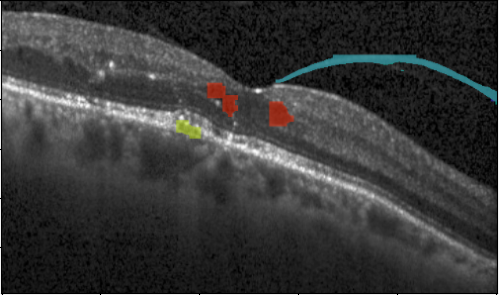


Рисунок 6 – ОКТ-изображение сетчатки после прохождения через модуль обработки

Выбранной модели нейронной сети удалось успешно сегментировать патологии морфологической структуры анализируемого изображения. В данном случае модель YOLOv12-seg сегментировала 3 объекта интраретинальных кист, 2 объекта друз и один объект отслойки стекловидного тела.

## Разработка модуля формирования заключения

Данные сегментации ОКТ-изображения, полученные модулем обработки, передаются на вход модуля формирования заключения.

В листинге 9 представлен пример данных сегментации, передаваемых модулю формирования заключения.

Листинг 9 – Пример результат сегментации

|  |
| --- |
| pred\_labels = { intraretinal\_cyst, retinal\_drusen, retinal\_drusen, intraretinal\_cyst, intraretinal\_cyst, vitreomacular\_traction }  pred\_scores = array([ 0.91497, 0.89903, 0.87451, 0.854, 0.83074, 0.79763], dtype=float32) |

Модуль формирования заключения получает на вход названия сегментированных классов (pred\_labels) и соответствующие им значения уверенности модели (pred\_scores). Позиции элементов в этих массивах согласованы: индекс каждого класса в pred\_labels соответствует индексу его вероятности в pred\_scores.

В листинге 10 приведен пример кода, формирующего заключение.

Листинг 10 – Реализация формирования заключения

|  |
| --- |
| def find\_matches(self, pred\_labels, pred\_scores):      result = set([])          for index, lable in enumerate(pred\_labels):              if (pred\_scores[index] > 0.5):                  for row in self.table:                      if lable == row[0]:                          result.add(row[1])            return result |

Пример кода в листинге 10 проходится по каждому из предсказанных классов патологий, проверяет уровень уверенности модели: если уровень уверенности модели меньше 0.5 – класс пропускается, находит соответствующий класс в референтной таблице и добавляет заключение в возвращаемый объект.

В приложении [] представлен полный код модуля формирования заключения.

## Разработка интерфейса программно-алгоритмического комплекса обработки ОКТ-изображений

Интерфейс программно-алгоритмического комплекса позволяет упростить взаимодействие с ним, сведя

На рисунке 7 представлен интерфейс программно-алгоритмического комплекса.

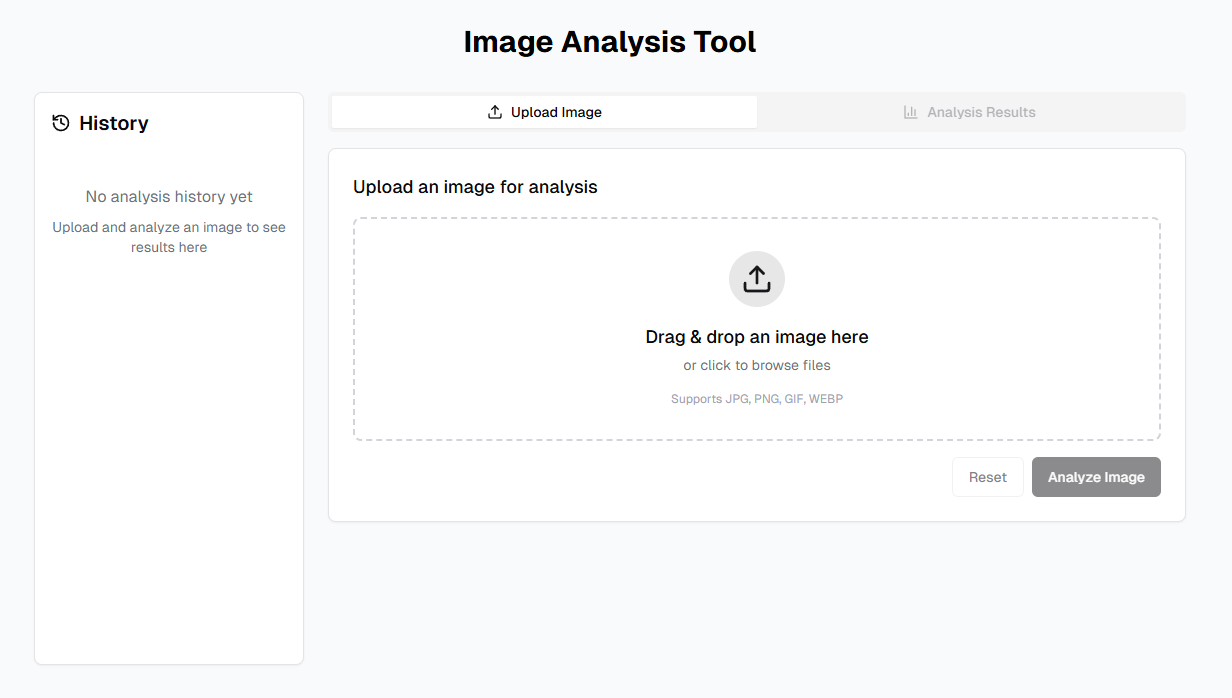


Рисунок 7 – Интерфейс программно-алгоритмического комплекса

Интерфейс состоит из трёх основных секций:

* История. В истории отображаются все изображения, обработанные за конкретный сеанс.
* Загрузка изображения. Загруженное изображение возможно просмотреть для подтверждения правильности загруженного изображения.
* Просмотр результатов обработки.

На рисунках 8, 9 представлены примеры взаимодействия с программно-алгоритмическим комплексом через визуальный интерфейс.

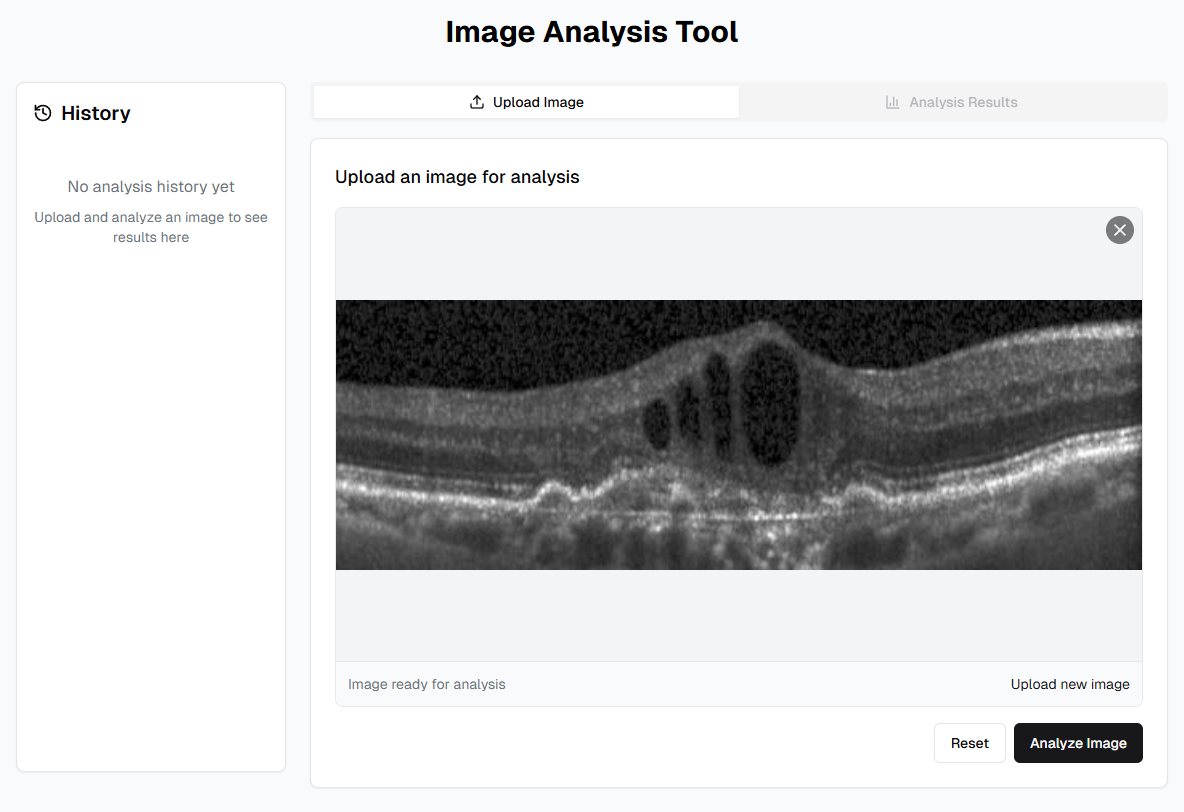


Рисунок 8 – Пример загрузки изображения

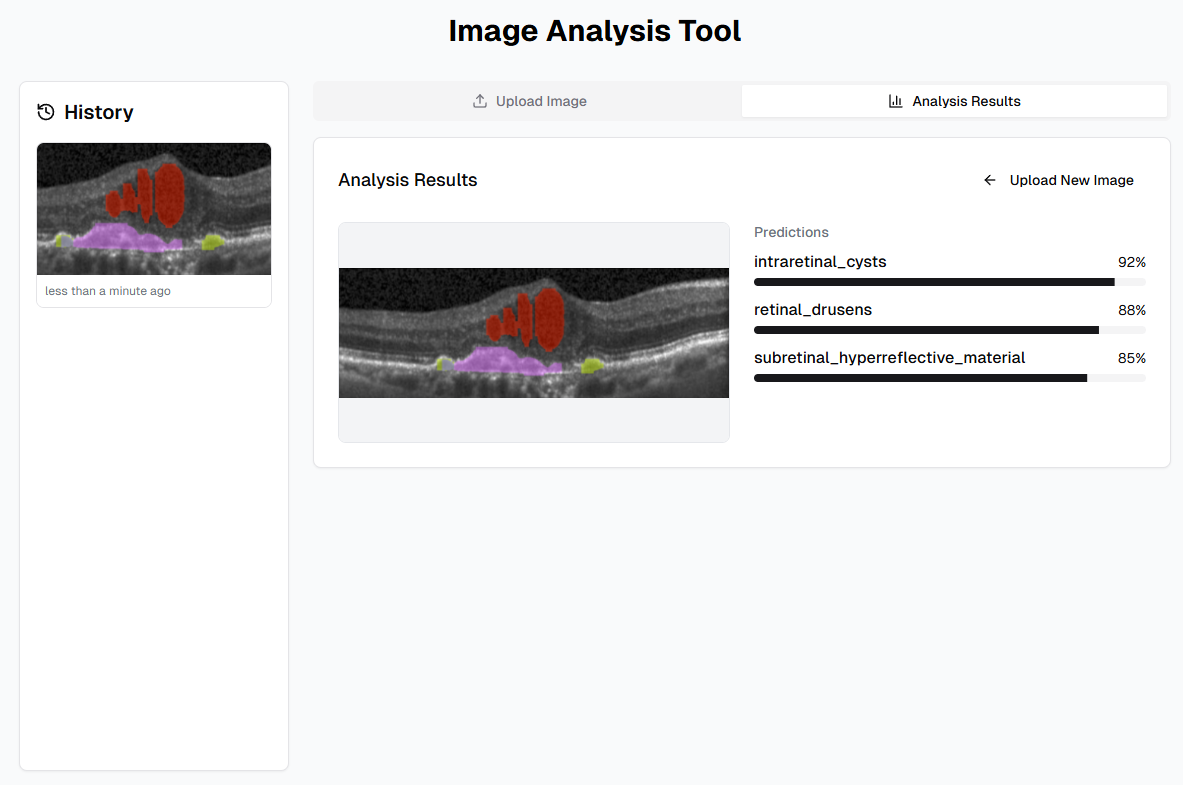


Рисунок 9 – Пример просмотра результатов обработки изображения

## Интеграция модулей программно-алгоритмического комплекса

Разработанные модули обработки ОКТ-изображения представляют собой отдельные логические элементы. Сущностью, объединяющей модули, является сервер, построенный при помощи мини-фреймворка Flask.

На рисунке 10 представлена схема взаимодействия разработанных модулей и пользовательского интерфейса.

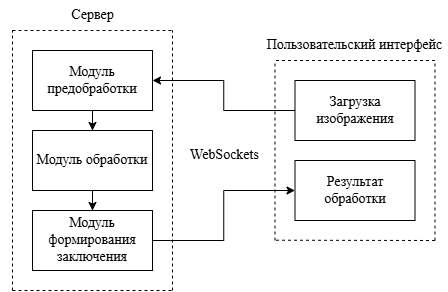


Рисунок 10 – Схема взаимодействия разработанных модулей с пользовательским интерфейсом

Серверная инфраструктура позволяет объединить разработанные модули и обращаться к ним в момент получения запроса на обработку ОКТ-изображения от пользователя.

Общение между сервером и интерфейсом осуществляется с помощью полнодуплексного интерфейса websokets, позволяющего пользователю получать результат обработки ОКТ-изображения в кратчайшие сроки.

## Вывод по главе

В данной главе были рассмотрены основные аспекты разработки программно-алгоритмического комплекса для анализа ОКТ-изображений сетчатки. Были обоснованы выбор использованных инструментов разработки, включая язык программирования Python и специализированные библиотеки, такие как OpenCV и Ultralytics. Подробно описаны этапы разработки модуля предобработки ОКТ-изображений, включая преобразование изображений, фильтрацию и выделение контуров.

Особое внимание уделено выбору и оценке моделей нейронных сетей для обработки ОКТ-изображений. В ходе работы были рассмотрены различные архитектуры, такие как SAM2, DeepLabV3+, YOLOv9, YOLOv11 и YOLOv12. Сравнительный анализ производительности этих моделей проводился на основе метрик, включающих Dice Index, Precision, Recall и Mean Average Precision (mAP@0.5).

Апробация моделей показала, что модель SAM2 демонстрирует наилучшую производительность с точки зрения точности сегментации и классификации патологий. Однако, SAM2 характеризуется высокими требованиями к вычислительным ресурсам. В связи с этим, для дальнейшей разработки была выбрана модель YOLOv12-seg, которая обеспечивает сопоставимые результаты при значительно меньших затратах ресурсов.

Метрики производительности выбранной модели YOLOv12-seg на тестовой выборке представлены в таблице 1. Анализ данных показывает, что модель успешно справляется с задачей сегментации, обеспечивая высокие значения метрик для большинства классов патологий. Тем не менее, для некоторых классов наблюдаются несколько более низкие значения Precision, что указывает на возможность улучшения классификации за счет расширения обучающей выборки и дополнительной настройки гиперпараметров модели.