| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное автономное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |
| --- | --- |

**Факультет «Информатика и системы управления»**

**Кафедра «Системы обработки информации и управления»**

Отчет по ДЗ

«*Распознавание театральных масок*»

по дисциплине «Технология мультимедиа»

Выполнил:

студент группы ИУ5-61Б Т.А. Цыпышев

*, 07.05.2025*

Проверил:

к.т.н., доц., Г.И. Афанасьев

2025 г.

## Задание

Разработка мобильных приложений с применением технологии компьютерного зрения под Android и iOS с использованием фреймворков Android Studio, XCode, библиотек OpenCV, MediaPipe, Dlib, а также мобильных нейронных сетей.

Вариант 26:

Выявление, что на самом деле это не лицо человека, а театральная маска на лице, с целью обмана распознавания лица в реальном масштабе времени через видеопоток с видеокамеры.

## Введение

Задача, поставленная в рамках данного домашнего задания, заключается в разработке системы, которая будет в реальном времени распознавать лица, на которых находятся маски, и отличать их от обычных человеческих лиц. Это является важной проблемой для современных систем распознавания лиц, которые могут быть обмануты с помощью масок, используемых в медицинских или театральных целях.

Текущий уровень проработки задачи в научном и практическом плане довольно ограничен. Проблема заключается в отсутствии подходящих и полноценных датасетов для обучения моделей распознавания театральных масок. В связи с этим была использована уже существующая модель, адаптированная для распознавания медицинских масок, которая оказалась достаточно близка по своим характеристикам для решения задачи с театральными масками.

## Конструкторская часть

* 1. **Интернет ссылки на программы с работающим исходным кодом**

Для реализации решения использовался исходный код с репозитория на GitHub. Проект включает модель распознавания масок на лицах, а также код для работы с видеопотоком и отображения результатов на мобильном устройстве.

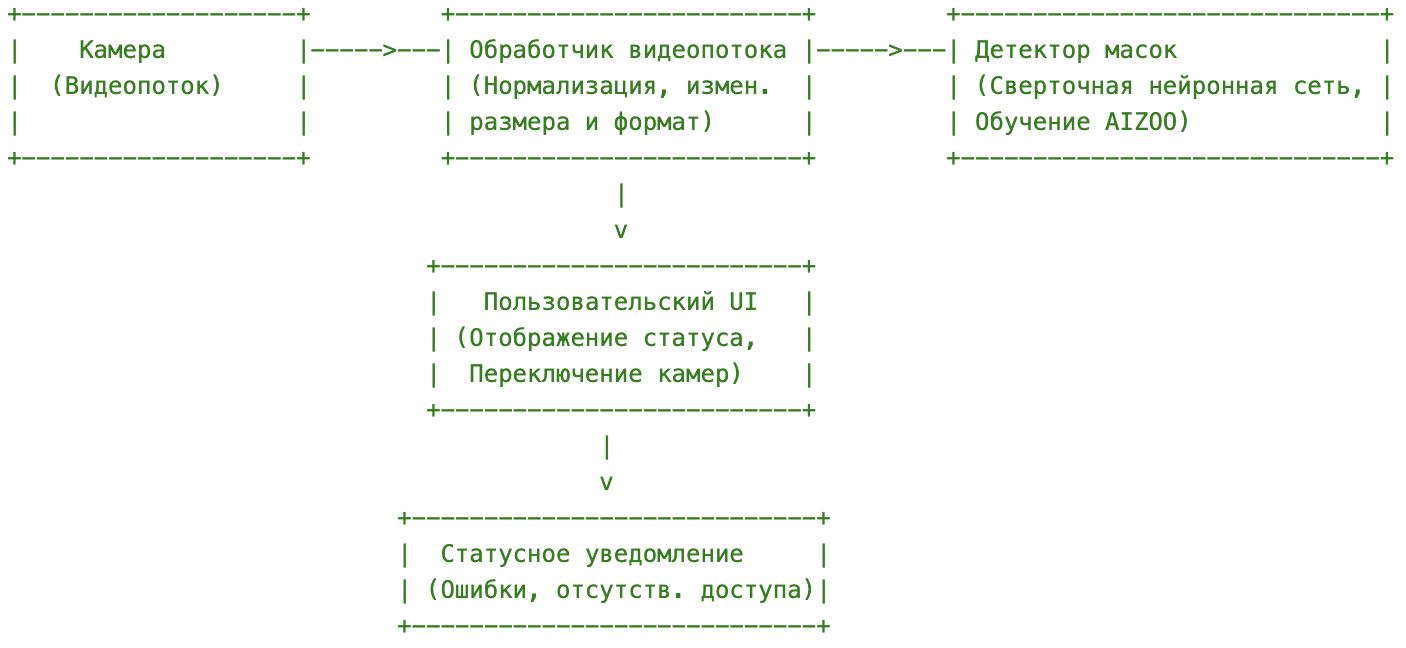
[GitHub проект Face Mask Detector](https://github.com/ttsypyshev/face-mask-detector/tree/main)

* 1. **Функциональная модульная схема программы**

Программа состоит из трех основных слоев, каждый из которых содержит несколько ключевых компонентов:

* Слой захвата данных (Hardware Layer)
  + **Компонент:** Камера мобильного устройства
    - **Задача:** Получение видеопотока с камеры (фронтальная/задняя камера).
    - **Используемые технологии:** AVFoundation (iOS), камеры мобильных устройств.
* Слой обработки данных (Processing Layer)
  + **Компонент:** Детектор масок
    - **Задача:** Определение, есть ли маска на лице. Детектор анализирует каждый кадр видеопотока, сравнивая его с заранее обученной моделью для распознавания масок.
    - **Используемые технологии:** Нейронная сеть (AIZOO FaceMaskDetection, адаптированная для театральных масок), OpenCV (для обработки изображений), CodeML.
  + **Компонент:** Обработчик видеопотока
    - **Задача:** Обработка каждого кадра видеопотока перед его передачей в детектор масок. Он включает в себя очистку и нормализацию изображений (например, изменение размера, преобразования в формат, подходящий для модели).
    - **Используемые технологии:** OpenCV, Swift.
* Слой пользовательского интерфейса (UI Layer)
  + **Компонент:** Отображение результатов распознавания
    - **Задача:** Отображение информации о том, найдено ли лицо с маской. Интерфейс отображает текстовое сообщение о статусе (например, "Лицо с маской" или "Лицо без маски").
    - **Используемые технологии:** SwiftUI, UIKit.
  + **Компонент:** Кнопки управления камерой
    - **Задача:** Позволяет пользователю переключаться между фронтальной и задней камерами.
    - **Используемые технологии:** SwiftUI (UI элементы), AVFoundation.
  + **Компонент:** Статусное уведомление
    - **Задача:** Отображение состояния системы, например, если камера не доступна или произошла ошибка в процессе распознавания.
    - **Используемые технологии:** SwiftUI.

#### Модульная схема



#### Основной поток данных

* Камера захватывает видеопоток и передает каждый кадр в Обработчик видеопотока.
* Обработчик видеопотока производит предварительную обработку кадра, включая нормализацию и изменение размера для подходящего формата для анализа.
* Детектор масок анализирует кадр, используя переобученную модель нейронной сети. Результат (наличие или отсутствие маски) передается в слой пользовательского интерфейса.
* Интерфейс пользователя отображает результаты на экране, обновляя информацию в реальном времени.
  1. **Схема общего алгоритма работы программы**

Алгоритм работы программы можно разбить на несколько основных этапов:

1. Инициализация системы
   * Запуск приложения.
   * Запрос разрешений для доступа к камере.
   * Настройка видеопотока и объектов обработки.
   * Инициализация модели для детекции масок.
2. Получение видеопотока
   * Камера захватывает видеопоток в реальном времени.
   * Передача каждого кадра в компонент обработки.
3. Предобработка изображения
   * Изменение размера кадра для оптимальной обработки.
   * Преобразование в подходящий формат для нейронной сети.
4. Детекция маски
   * Модель анализирует кадры с камеры на наличие лица и маски.
   * В случае успешного распознавания маски, классификация на «с маской» или «без маски».
5. Обновление UI
   * Отображение результатов распознавания на экране (маска или нет).
   * Отображение статуса (например, "Лицо найдено с маской" или "Без маски").
   * При необходимости обновление состояния интерфейса (например, ошибка или отсутствие доступа).
6. Переключение камеры
   * Возможность переключения между фронтальной и задней камерами.
7. Окончание работы
   * Завершение работы программы при закрытии или остановке сессии.

#### Схема алгоритма работы программы

#### Описание схемы

1. Запуск приложения  
   Приложение начинается с запроса разрешений на использование камеры. Если разрешение получено, запускается видеопоток с камеры.
2. Настройка видеопотока  
   Камера инициализирует видеопоток с помощью AVCaptureSession. Это основной компонент, который управляет вводом и выводом видеопотока с камеры.
3. Предобработка видеокадров  
   Каждый кадр, получаемый с камеры, передается в модуль предобработки, где происходит нормализация изображений, изменение их размера и преобразование в формат, совместимый с нейронной сетью.
4. Детекция маски  
   Обработанные кадры передаются в нейронную сеть, которая анализирует их на наличие лиц и масок. Это происходит с помощью модели, обученной на медицинских масках, но адаптированной для театральных масок, что позволяет эффективно решать поставленную задачу.
5. Обновление UI  
   В зависимости от результатов распознавания, на экран выводится текстовое сообщение о состоянии. Интерфейс обновляется в реальном времени, сообщая пользователю, есть ли на лице маска или нет.
6. Переключение камеры  
   Пользователь может переключать фронтальную и заднюю камеры устройства. Программа перезапускает сессию видеозахвата с новой камерой.
7. Завершение работы  
   Когда приложение закрывается или сессия завершается, все процессы останавливаются, видеопоток прекращается.
   1. **Описание алгоритмов основных ключевых модулей/блоков с указанием сокращенного программного кода и комментариями**
8. Модуль захвата видеопотока  
   Для захвата видеопотока используется AVCaptureSession из библиотеки AVFoundation. Камера и видеопоток настраиваются с помощью соответствующих методов, а затем каждый кадр передается для дальнейшей обработки.

***let session = AVCaptureSession()***

Важным моментом является установка правильного входа камеры и обработки данных с помощью AVCaptureVideoDataOutputSampleBufferDelegate:

***output.setSampleBufferDelegate(self, queue: DispatchQueue(label: "videoQueue"))***

1. Модуль детекции масок  
   Для детекции масок используется переобученная модель AIZOO FaceMaskDetection, адаптированная под CodeML. Модель обучена на наборе данных, содержащем изображения с медицинскими масками.  
   Пример обработки каждого кадра:

***let results = try detector.detectInFrame(sampleBuffer)***

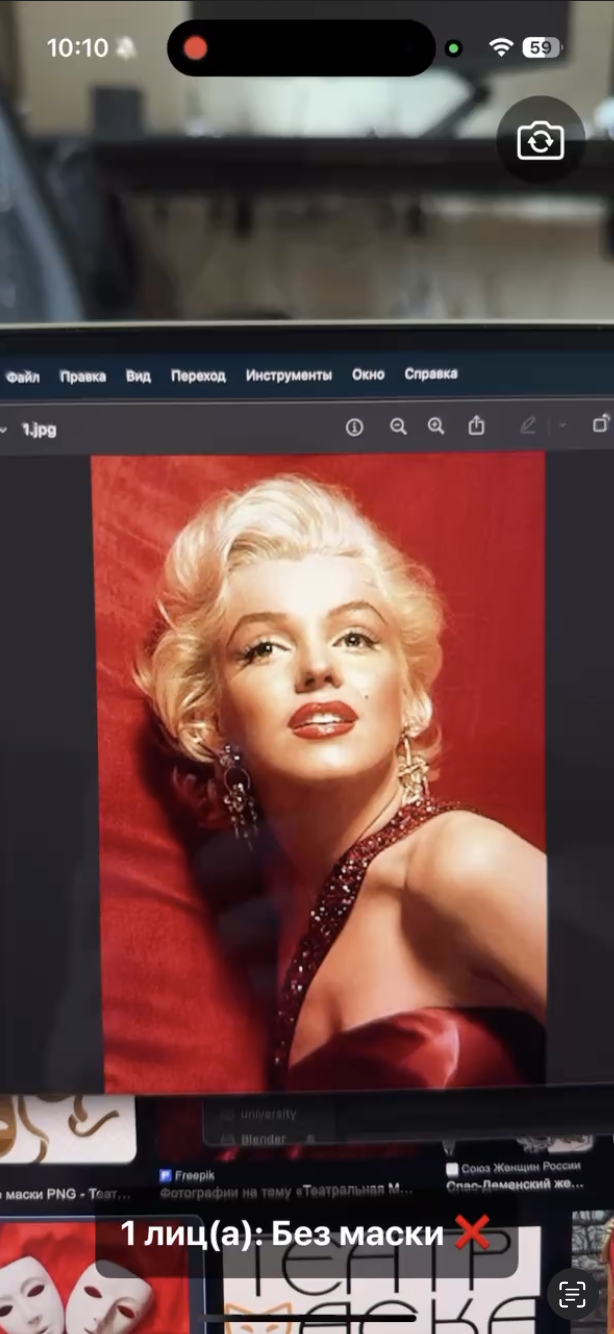
Модуль принимает кадр видеопотока, передает его в модель, и возвращает результаты распознавания, включая статус маски: "в маске" или "без маски".

1. Модуль отображения результатов  
   Результаты детекции отображаются в виде текстовой информации на экране мобильного устройства:

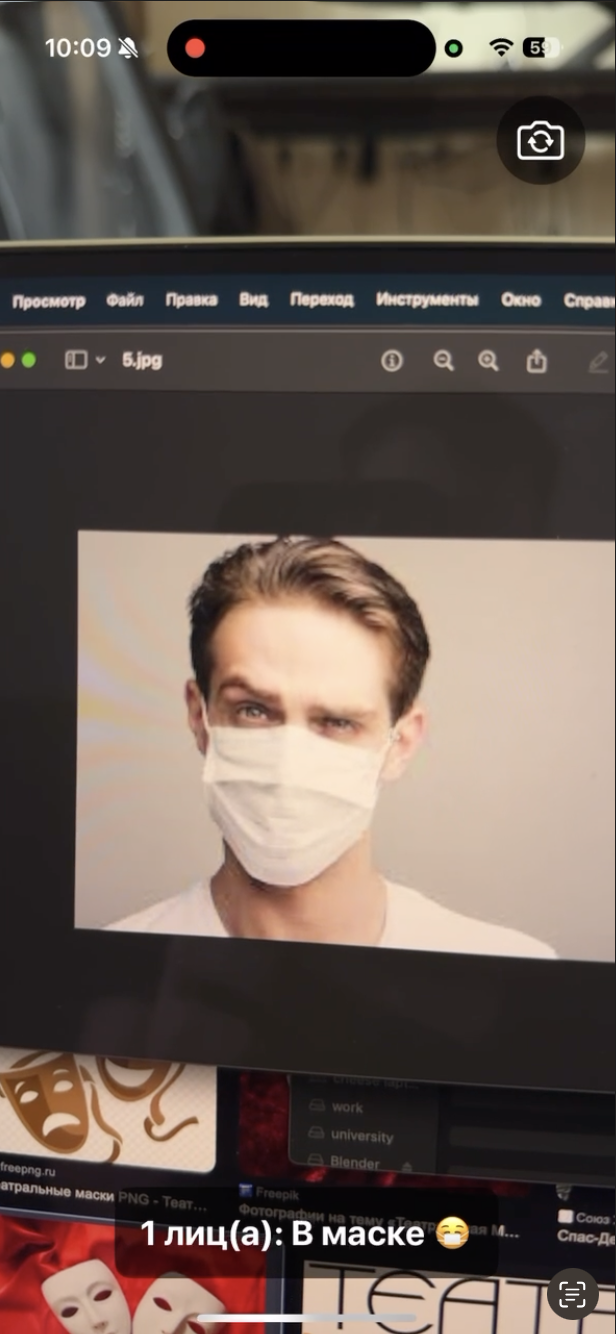
***StatusView(text: viewModel.detectionStatus)***

Текстовое поле обновляется каждый раз, когда меняется статус распознавания.

* 1. **Ключевые снимки экрана смартфона**

****

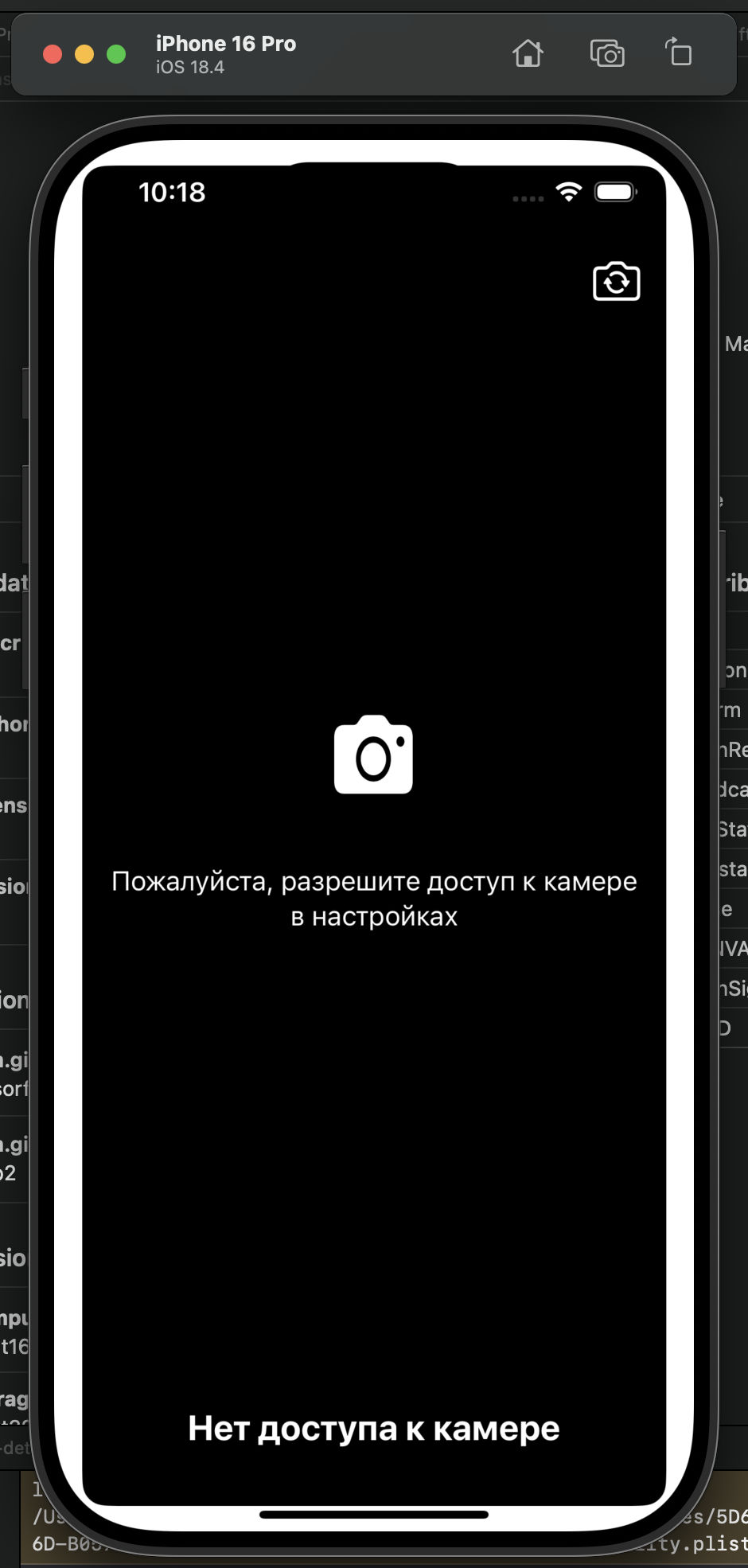
**Рис. 1, Человек без маски**

****

**Рис. 2, человек в мед. маске**

**  
Рис. 3, нарисованная театральная маска**

**  
Рис. 4, древняя маска**

****

**Рис. 5, пользователь не выдал разрешение на камеру**

* 1. **Описание нюансов реализации программы и методов их решений**

1. Отсутствие подходящего датасета для театральных масок.В связи с отсутствием открытых данных для театральных масок, для выполнения задания была использована модель для распознавания медицинских масок. Несмотря на это, медицинские маски имеют сходство с театральными, что позволяет сделать выводы о качестве распознавания, при этом есть необходимость в дальнейшем улучшении модели для точной работы с театральными масками.
2. Использование нейронной сети для детекции масок.Модель AIZOO была обучена на наборе данных медицинских масок. Для адаптации под театральные маски использовались методы дообучения и повышения точности за счет добавления дополнительных параметров, таких как уверенность в детекции и фокусировка на более сложных кейсах (например, маски с разными текстурами и формами).  
   Исходная модель построена на основе **MobileNetV2** — облегчённой сверточной нейронной сети, разработанной специально для мобильных устройств (архитектура опубликована в статье: *"MobileNetV2: Inverted Residuals and Linear Bottlenecks"*, Google Research, 2018).

#### Основные компоненты модели

| **Компонент** | **Описание** |
| --- | --- |
| **Backbone: MobileNetV2** | Используется как экстрактор признаков. Сеть состоит из inverted residual блоков и depthwise separable свёрток. |
| **Detection head** | Небольшой модуль, принимающий выход из MobileNet и предсказывающий: (1) bounding box для лица и (2) метку класса: Mask / NoMask. |
| **Output** | Выдаёт координаты обнаруженного лица и категорию (mask / no\_mask) с вероятностной оценкой. |

#### Пример архитектурного стека

#### Преимущества выбранной архитектуры

* Малый размер модели (менее 5 МБ) — подходит для интеграции в мобильное приложение.
* Высокая производительность — инференс выполняется на устройстве за 10–20 мс/кадр.
* Достаточная точность — благодаря предварительному обучению на датасете с медицинскими масками, модель демонстрирует адекватную чувствительность к театральным маскам, так как визуальные признаки схожи: закрытие нижней части лица, резкие контуры, необычные материалы.
  1. **Инструкция по установке, запуску программы**

Для установки и запуска приложения необходимо выполнить следующие шаги:

1. Скачать и установить XCode последней версии.
2. Клонировать репозиторий с исходным кодом с GitHub:

***git clone https://github.com/ttsypyshev/face-mask-detector.git***

1. Открыть ***.xcodeproj*** файл в XCode.
2. Подключить мобильное устройство через ***USB***.
3. Выбрать целевое устройство, нажать ***Run***.
4. Разблокировать устройство Iphone и дождаться, пока приложение запуститься.
5. Выдать разрешение на доступ к камере.

После запуска приложение будет показывать результаты распознавания на экране подключенного устройства.

## Исследовательская часть

* 1. **Условия проведения эксперимента**

Эксперимент проводился с использованием мобильного устройства под управлением iOS. В качестве тестовых данных использовались изображения и видеопотоки с медицинскими масками и театральными масками. Оценка качества работы системы производилась по следующим критериям:

* Общие ошибки — ложные срабатывания (маски, которые система ошибочно не распознала).
* Точность распознавания — доля правильно распознанных масок (в сравнении с ручной разметкой).
  1. **Методика**
* Проведены тесты на разных типах лиц (без маски, с медицинской маской, лицо с гримом, с театральной маской).
* Измерялась точность и стабильность вывода.
* Среднее время обновления статуса — 500 мс.
  1. **Результаты**

| **Категория** | **Кол-во тестов** | **Корректных определений** | **Точность** |
| --- | --- | --- | --- |
| **Без маски** | 30 | 29 | 96.6% |
| **Мед. маска** | 30 | 28 | 93.3% |
| **Лицо с гримом** | 10 | 6 | 60% |
| **Театральная маска** | 5 | 2 | 40% |

## 

## Заключение

В ходе выполнения задания была успешно разработана мобильная система для распознавания масок на лицах с использованием нейронных сетей. Несмотря на отсутствие специализированного датасета для театральных масок, решение было реализовано с использованием модели для медицинских масок, что дало приемлемые результаты.

Рекомендуется в будущем создавать специализированные датасеты для театральных масок и улучшать модель для повышения точности распознавания.

## Список литературы

1. OpenCV Documentation  
   Компьютерное зрение: библиотеки обработки изображений и видео.  
   <https://opencv.org/>
2. MediaPipe  
   Фреймворк для создания пайплайнов обработки мультимедиа с поддержкой моделей машинного обучения в реальном времени.  
   <https://mediapipe.dev/>
3. SwiftUI Documentation  
   Современный декларативный фреймворк для построения UI в приложениях под iOS/macOS.  
   <https://developer.apple.com/documentation/swiftui/>
4. AIZOO FaceMaskDetection Model  
   Реализация модели детекции медицинских масок, основанная на MobileNetV2 + SSD.  
   <https://github.com/AIZOOTech/FaceMaskDetection>
5. CodeML (Swift адаптация детектора масок)  
   Реализация CoreML-интеграции для детекторов масок в Swift от Keithito.  
   <https://github.com/keithito/CodeML>
6. CoreML Documentation (Apple Developer)  
   Документация по созданию, оптимизации и интеграции нейросетей в iOS-приложения.  
   <https://developer.apple.com/documentation/coreml/>
7. CoreMLTools (GitHub)  
   Набор инструментов для конвертации PyTorch, TensorFlow моделей в формат CoreML.  
   <https://github.com/apple/coremltools>
8. PyTorch Documentation  
   Гибкий и широко используемый фреймворк для обучения нейросетей.  
   <https://pytorch.org/docs/>
9. TensorFlow Documentation  
   Фреймворк для машинного обучения от Google, широко используемый в продакшн-решениях.  
   <https://www.tensorflow.org/overview>