Привет всем читателям habr.com! Мы студенты НГТУ им. Р.Е. Алексеева, и хотим рассказать о своем опыте работы с библиотекой компьютерного зрения OpenCV. Данный набор инструментов был использован для распознавания, классификации и отслеживания объектов.

Для начала давайте познакомимся. Мы - студенты 1 курса ИРИТ, кафедры «Информатика и системы управления» – Чекалин Дмитрий, Какушкина Ольга, Солодун Максим, Перегон Артем.

Наше знакомство с OpenCV произошло на первом курсе. Мы решили использовать этот инструмент для обработки видеопотока, выделения объектов (людей, автомобилей), отслеживания и распознавания признаков.

Что такое OpenCV?

OpenCV — это открытая библиотека компьютерного зрения, которая предназначена для обработки изображений и видео, для идентификации и отслеживания объектов, установления их характерных признаков. В настоящее время библиотека играет очень важную роль, т.к. позволяет разработчику работать в режиме реального времени.

Изначально она была написана на C++, но адаптирована для использования на Python, C, Java и MATLAB. OpenCV адаптирована для использования на различных платформах, в частности — на Windows, Linux и MacOS.

Какие преимущества даёт пользователю OpenCV?

Основные преимущества продукта:

- Бесплатная и открытая библиотека.
- Написана на C/C++ и в сравнении с другими библиотеками работает быстрее.
- Не требует большого количества оперативной памяти на устройстве.
- Поддерживаема большинством ОС.

Установка библиотеки

В терминале VS Code выполним команду: pip install opencv-python

Теперь в новом файле Python пишем команду: import cv2

```
1  # подключение библиотеки компьютерного зрения
2  import cv2
```

Убедимся, что все работает правильно: запустим скрипт, если ошибок нет – двигаемся дальше.

Получение изображения с камеры

По умолчанию библиотека работает с веб-камерой устройства, которая, как правило, установлена первой (зачастую и единственной). Следует обратиться к нулевому устройству видеозахвата (нумерация в программировании начинается с нуля).

Теперь пропишем вариант, когда по какой-то причине камера не работает и не получает изображение, в таком случае будем прекращать выполнение программы. Также добавим выход из программы при нажатии ESC.

Получаем следующий код:

```
import cv2

def analiz():

    video = cv2.VideoCapture(0)

    while True:

        ret, frame = video.read()

        if not ret:

            break

        cv2.imshow("Face Detection", frame)

        if cv2.waitKey(1) & 0xFF == 27:

                break

analiz()
```

Подключение моделей

Для того чтобы библиотека могла определять лица и устанавливать их характерные признаки (пол и возраст) ее нужно этому обучить. В рамках нашей работы были использованы обученные модели, ниже приведен код с моделями и некоторыми переменными, которые будут нужны дальше.

Модели заранее были помещены в отдельную папку внутри проекта (рис. 1).

```
facePhoto = "model/opencv_face_detector.pbtxt"

faceModel = "model/opencv_face_detector_uint8.pb"

ageProto = "model/age_deploy.prototxt"

ageModel = "model/age_net.caffemodel"

genderPhoto = "model/gender_deploy.prototxt"

genderModel = "model/gender_net.caffemodel"

faceNet = cv2.dnn.readNet(faceModel, facePhoto)

ageNet = cv2.dnn.readNet(ageModel, ageProto)

genderNet = cv2.dnn.readNet(genderModel, genderPhoto)

MODEL_MEAN_VALUES = (78.4263377603, 87.7689143744, 114.895847746)

ageList = ['(0-2)', '(4-6)', '(8-15)', '(16-25)', '(25-32)', '(38-43)', '(48-
53)', '(60-100)']

genderList = ['Male', 'Female']
```

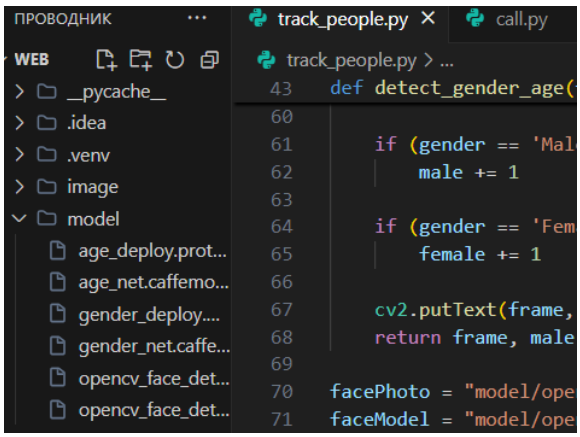

Рисунок 1 - Расположение моделей в проекте.

Распознавание лиц

Затем прописываем функцию распознавания лица человека:

Наша функция принимает два аргумента. Модель faceNet и текущее изображение с веб-камеры:

```
def faceBox(faceNet, frame)
```

Затем в переменные frameHeight и frameWidth занесем высоту и ширину полученного изображения:

```
frameHeight = frame.shape[0]
```

```
frameWidth = frame.shape[1]
```

Переходим к созданию блоба из полученного фрейма. Блоб – область изображения, у которой ряд свойств считаются постоянными или приблизительно постоянными, все точки в блобе можно считать в некотором смысле подобными друг другу.

```
blob = cv2.dnn.blobFromImage(frame, 1.0, (227, 227), [104, 117, 123], swapRB=False)
```

Параметры функции blobFromImage:

- frame – изображение, полученное с камеры.
- (227, 227) - размер блоба.
- [104, 117, 123] - значения смещения для каналов RGB.
- swapRB=False - не меняет местами каналы RGB.

Установим блоб в качестве входных данных для модели обнаружения лиц:

```
faceNet.setInput(blob)
```

Запустим модель обнаружения лиц на входном блобе и получения результатов.

```
detection = faceNet.forward()
```

Инициализируем пустой список для хранения координат обнаруженных лиц.

```
bboxs = []
```

Дальше в цикле пройдем по результатам обнаружения (каждое обнаруженное лицо), в переменную `confidence` получим вероятность обнаружения лица. Затем проверим вероятность, что она выше установленного порога (в нашем случае 0.7) и вычислим координаты лица из результатов модели, при этом результат вычисления координат преобразуем к целому числу.

В переменные `center_x, center_y` занесем результат вычисления координат центра лица. Также добавим координаты центра в наш список bboxs.

```
for i in range(detection.shape[2]):
        confidence = detection[0, 0, i, 2]
        if confidence > 0.7:
            x1 = int(detection[0, 0, i, 3] * frameWidth)
            y1 = int(detection[0, 0, i, 4] * frameHeight)
            x2 = int(detection[0, 0, i, 5] * frameWidth)
            y2 = int(detection[0, 0, i, 6] * frameHeight)
            center_x = (x1 + x2) // 2
            center_y = (y1 + y2) // 2
            bboxs.append([x1, y1, x2, y2, center_x, center_y])
```

После выполнения функция вернем в нашу основную функцию analiz изображение с камеры и список обнаруженных лиц.

Таким образом, функция `faceBox` приняла на вход изображение, полученное с камеры устройства, обнаружила на нем лица (при помощи предварительно обученной модели) и вернула изображение и список координат обнаруженных лиц.

Определение пола и возраста

Далее реализуем функцию определения пола и возраста человека.

Наша функция принимает следующие параметры:

- frame - изображение с камеры, на котором будут определены пол и возраст
- x1, y1, x2, y2 - координаты ограниченного прямоугольника, в котором находится лицо
- male, female - количество мужчин и женщин (эти переменные передаются из основной функции analiz, их опишем далее):

```
def detect_gender_age(frame, x1, y1, x2, y2, male, female)
```

Аналогично прошлой функции добавим в код следующие команды:

```
blob = cv2.dnn.blobFromImage(frame, 1.0, (227, 227), MODEL_MEAN_VALUES, swapRB=False)
genderNet.setInput(blob)
genderPreds = genderNet.forward()
```

Далее добавим следующую команду:

```
gender = genderList[genderPreds[0].argmax()]
```

Данная команда определяет пол человека на основе выходных данных нейронной сети для определения пола. Она находит индекс категории пола с наибольшей вероятностью в выходном векторе нейронной сети, а затем использует этот индекс для получения соответствующего текстового значения пола (например, 'Male' или 'Female') из предварительно определенного списка категорий полов.

Аналогично поступим и для определения возраста:

```
ageNet.setInput(blob)

agePreds = ageNet.forward()

age_index = agePreds[0].argmax()
```

Проверим находится ли значение age_index (индекс предсказанного возраста) в пределах длины списка agelist. Этот список содержит категории возрастов, которые были определены раннее.

Если условие выполнилось, то значение возраста берется из списка ageList по индексу age_index. Таким образом, получено текстовое значение возраста, соответствующее предсказанному возрасту.

Если условие не выполнилось, значит, предсказанный возраст не соответствует ни одной из заранее определенных категорий. В этом случае возрасту присваивается значение “Unknown”.

```
if age_index < len(ageList):

      age = ageList[age_index]

else:

     age = "Unknown"
```

Данные строки необходимы для того, чтобы корректно обрабатывать ситуации, когда модель обнаружения возраста предсказывает значение возраста, которое не соответствует ни одной из категорий из списка ageList. В таком случае устанавливается значение “Unknown”.

Затем создадим текстовую метку, которая будет отображаться на экране и содержать значение пола и возраста человека.

Для удобства воспользуемся f-строкой, она позволит нам вставлять значение gender, age прямо в момент выполнения программы.

```
label = f"{gender}, {age}"
```

Далее установим обновление счетчиков мужчин и женщин. Эта информация нам пригодится далее в проекте.

```
if (gender == 'Male'):

      male += 1


if (gender == 'Female'):

      female += 1
```

Далее применим команду:

```
cv2.putText(frame, label, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,
255, 255), 2, cv2.LINE_AA)
```

- frame- текущий кадр
- label – текстовая метка, содержащая пол и возраст человека

- (x1, y1 – 10) – координаты, где будет расположен левый верхний угол метки
- cv2.FONT_HERSHEY_SIMPLEX – шрифт, используемый для отображения текста (в OpenCv предусмотрено несколько шрифтов)
- 0.8 – масштаб текста (его размер)
- (0, 255, 255) – цвет текста в формате RGB (в данном случае желтый цвет)
- 2 – толщина линий текста
- cv2.LINE_AA – метод, придающий тексту более гладкий и плавный вид

Вернем из нашей функции текущий фрейм, счетчик мужчин и женщин.

```
return frame, male, female
```

Затем переходим к основной функции analiz, в которой используются две вышеупомянутые функции.

Основная функция analiz()

Вернемся к раннее написанной функции analiz.

Добавим переменные:

```
unique_people_count = 0

prev_people_positions = {}

track_id = 0

frame_count = 0

frame_skip = 3

male_count = 0

female_count = 0
```

Вызовем функцию faceBox, которая распознает лица на изображении, полученном с камеры, увеличим счетчик фреймов на каждой иттерации цикла while, инициализируем пустой список center_point.

```
frame, bboxs = faceBox(faceNet, frame)

frame_count += 1

center_point = []
```

Добавим условие, чтобы пропускать некоторые кадры, чтобы оптимизировать выполнение программы, frame_skip – переменная, которая определяет какие кадры отбрасываются.

```
if frame_count %frame_skip == 0:
```

Сделаем цикл, который проходит по всем обнаруженным прямоугольникам на текущем кадре. Распакуем координаты ограничивающего прямоугольника (x1, y1, x2, y2) и координаты центра (center_x, center_y) лица из объекта bbox.

Отрисуем зеленый прямоугольник вокруг обнаруженного лица, добавим координаты центра лица в список `center_point` (он будет использоваться для трекинга людей), вызовем функцию определения возраста и пола.

```
if frame_count %frame_skip == 0:

            for bbox in bboxs:

                x1, y1, x2, y2, center_x, center_y = bbox
```

```
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
                center_point.append((center_x, center_y))
                frame, male_count, female_count = detect_gender_age(frame,
x1, y1, x2, y2, male_count, female_count)
```

Трекинг людей

Трекинг – одна из задач сопровождения объекта, от распознавания отличается тем, что необходимо следить, где находится объект на следующем кадре.

Далее определим является ли найденный человек новым или он уже был на видео.

Проходим циклом for по списку `center_point`, установим флаг is_new в True, тем самым предполагая, что обнаруженный человек является новым.

Добавим вложенный цикл for track_id, prev_position in prev_people_positions.items():, который будет проходить по словарю prev_people_positions (ключом будет уникальный идентификатор человека – track_id, а значением его предыдущая позиция).

Вычислим Евклидово расстояние между текущей позицией человека и его предыдущей позицией. Если расстояние между текущей и предыдущей позицией человека меньше 220 пикселей, то считается, что это тот же самый человек. В этом случае флаг is_new устанавливается в False, и позиция человека обновляется в словаре prev_people_positions. Как только найден человек, совпадающий с текущим, вложенный цикл прерывается.

Если флаг is_new остался True, то это означает, что человек является новым. Новая позиция человека добавляется в словарь prev_people_positions с уникальным идентификатором track_id и увеличивается счетчик уникальных людей. Идентификатор трека (track_id) увеличивается на 1, чтобы быть готовым к следующему новому человеку.

```
for i, current_position in enumerate(center_point):
                is_new = True
           for track_id, prev_position in prev_people_positions.items():
               distance = math.hypot(prev_position[0] - current_position[0],
prev_position[1] - current_position[1])


               if distance < 220: #120 было
                 is_new = False
                 prev_people_positions[track_id] = current_position
                 break
             if is_new:
               prev_people_positions[track_id] = current_position
               unique_people_count += 1
               track_id += 1
```

В конце цикла добавим вывод количества уникальных людей на экран:

```
cv2.putText(frame, f"Unique People: {unique_people_count}", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)
```

Функция будет возвращать количество мужчин, женщин и уникальных людей.

```
return male_count, female_count, unique_people_count
```

Проверим выполнение функции (рис. 2).

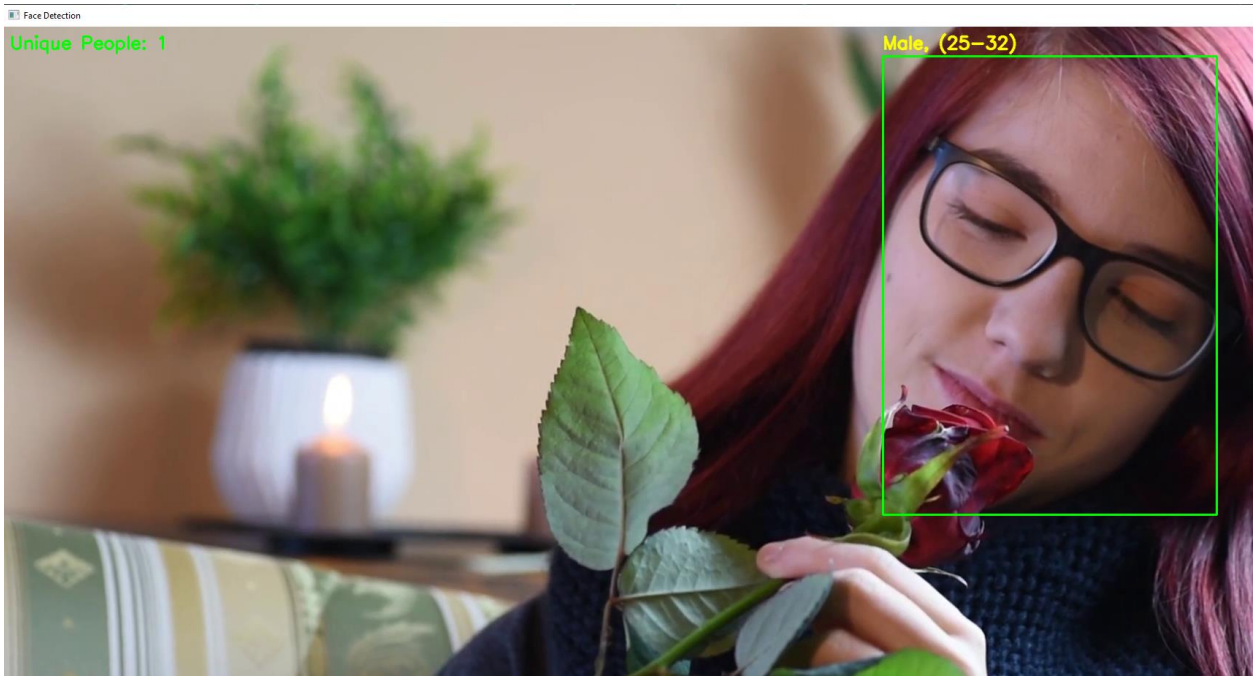

Рисунок 2 - Результат работы программы.

Распознавание машин

Наша функция будет принимать один параметр: путь к видео, которое будет анализироваться (это пригодится в дальнейшем в рамках проекта).

```
def analiz_car(file_path)
```

Для начала необходимо загрузить модель, которая будет распознавать машины. Для этой задачи была выбрана модель YOLOv5. Модель выбрана ввиду следующих преимуществ:

- легковесная модель.
- высокое качество предсказания.
- может быть использована для распознавания объектов в реальном времени.

Модель можно скачать на официальном сайте GitHub Релизы · Ультралитики/Йолов5 (github.com).

На сайте есть несколько версий модели, была выбрана yolov5n (small), она не требует больших вычислительных мощностей, для поставленной задачи обнаружить машину ее хватит. Также можно попробовать взять версии «потяжелее», они требуют больше вычислительной мощности, но дают более точной результат. Помещаем модель в папку проекта.

Введем начальные переменные, которые далее будут использоваться:

Создадим экземпляр модели YOLO версии 5, используя предварительно обученные веса из файла.

```
model = YOLO("yolov5n.pt")
```

Чтобы использовать модель и определять Эвклидово расстояние, необходимо добавить следующие команды:

```
import cv2

from ultralytics import YOLO

import math
```

Объявим переменные-счетчики для подсчета количества грузовых и легковых автомобилей. Аналогично прошлой функции определения людей добавим словарь для хранения информации об объектах, список для хранения центров обнаруженных объектов, счетчик уникальных идентификаторов объекта, счетчик фреймов и переменную для пропуска некоторых кадров.

```
detected_cars = 0

detected_trucks = 0

tracking_objects = {}

center_point_prev_frame = []

track_id = 0

frame_count = 0

frame_skip = 5
```

Вынесем в отдельный вспомогательный метод определение центра координат.

```
def get_center(x1, y1, x2, y2):

    x = int((x1 + x2) // 2)

    y = int((y1 + y2) // 2)

    return (x, y)
```

Создадим переменную для захвата видео из файла, раннее переданного в основную функцию в качестве аргумента.

```
video = cv2.VideoCapture(file_path)
```

Внутри функции аналогично прошлой создадим бесконечный цикл, в котором будем захватывать изображение с видео, пропишем условие выхода из цикла, создадим вспомогательные переменные.

```
while True:

        ret, frame = video.read()

        if not ret:

            break

        frame_count += 1

        center_point = []

if cv2.waitKey(1) & 0xFF == 27:

            break

    video.release()

    cv2.destroyAllWindows()
```

Пропишем условие обработки кадра, далее необходимо с помощью встроенного метода resize изменить размер изображения, чтобы наша модель корректно могла его обработать. Затем передадим в модель текущий фрейм, после обработки модель вернет фрейм с результатами

обработки, которые сохранятся в переменной results. Такими действиями мы повысим точность и сбалансируем производительность обнаружение объектов.

```
if frame_count % frame_skip == 0:

            frame = cv2.resize(frame, (1920, 1080))

            results = model(frame)
```

Затем в цикле пройдем по всем обнаруженным объектам аналогично прошлой функции и в текстовую метку поместим номер класса, к которому относится объект.

```
for result in results[0].boxes.data:

                x1, y1, x2, y2, confidence, class_id = result

                label = model.names[int(class_id)]
```

Сделаем проверку: если обнаруженный объект автомобиль или грузовик и эта вероятность выше 0.5, то нарисуем зеленый прямоугольник вокруг объекта в кадре и добавим текстовую метку объекта. Внутри условия сделаем проверки на то, к какому классу относится автомобиль, и увеличим соответствующие счетчики. Также найдем координаты середины обнаруженного объекта и добавим их в список.

```
if (label == "car" or label == "truck") and confidence >= 0.5:

                    cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), 
int(y2)), (0, 255, 0), 1)

                    cv2.putText(frame, f"{label} ({confidence:.2f})", 
(int(x1), int(y1) - 10), cv2.FONT_HERSHEY_SIMPLEX,

                                0.5, (36, 255, 12), 1)


                    if (label == "car"):

                       detected_cars += 1


                    if (label == "truck"):

                        detected_trucks += 1


                    center_x, center_y = get_center(x1, y1, x2, y2)

                    center_point.append((center_x, center_y))
```

Трекинг машин

В трекинге машин будем применять похожий принцип, как и с людьми. Нам не нужно знать все координаты центров объектов, достаточно знать положение объекта на прошлом кадре и на текущем. Сложнее подобрать пороговое расстояние, при котором объект считается уникальным. Усложнять задачу может разное качество камеры, из-за которого будет отличаться количество кадров, соответственно объекты будут перемещаться более резко.

Но машины проще отслеживать, чем людей, потому что их траектории более предсказуемы.

Напишем код отслеживания автомобилей для первых двух кадров. Сравним центры объектов на текущем кадре и на прошлом. Если расстояние меньше 150, то это тот же объект. Расстояние 150 получено опытным путем, в зависимости от задачи его можно менять.

```
if frame_count <= 2:
      for pt in center_point:
            for pt2 in center_point_prev_frame:
                  distance = math.hypot(pt2[0] - pt[0], pt2[1] - pt[1])
                  if distance < 150:
                        tracking_objects[track_id] = pt
                        track_id += 1
```

В следующем фрагменте кода мы должны обязательно сравнить предыдущий объект с текущим и обновить положение идентификатора. У того же объекта остается один идентификатор на протяжении его движения, в случае если объект исчезает, он теряет идентификатор.

```
else:

                tracking_objects_copy = tracking_objects.copy()
                center_point_copy = center_point.copy()


                for object_id, pt2 in tracking_objects_copy.items():
                    objects_exists = False
                    for pt in center_point_copy:
                        distance = math.hypot(pt2[0] - pt[0], pt2[1] - pt[1])


                        if distance < 150:
                            tracking_objects[object_id] = pt
                            objects_exists = True
                            if pt in center_point:
                                center_point.remove(pt)
                            continue


                    # Remove id
                    if not objects_exists:
                        tracking_objects.pop(object_id)
```

Если идентифицирован новый объект, список точек также должен быть обновлен. В конце прошлого фрагмента необходимо дописать изменения, которые позволяют удалять старые и добавлять точки новых идентифицированных автомобилей.

```
for pt in center_point:
      tracking_objects[track_id] = pt
      track_id += 1
```

Наконец, отобразим зеленые круги в центрах всех обнаруженных автомобилей и их идентификаторы.

```
for object_id, pt in tracking_objects.items():
      cv2.circle(frame, pt, 5, (0, 255, 0), -1)
      cv2.putText(frame, str(object_id), (pt[0], pt[1] - 7), 0, 1, (0, 0,
255), 1)
```

Отобразим количество уникальных автомобилей в кадре, а также сохраним копию списка центров обнаруженных объектов на текущем кадре, чтобы на следующем кадре мы могли отслеживать машины.

```
cv2.putText(frame, f"Unique cars: {track_id}", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)
            cv2.imshow("Object Detection", frame)
center_point_prev_frame = center_point.copy()
```

Функция analiz_car возвращает количество грузовиков и легковых машин.

```
return detected_cars, detected_trucks
```

Проверим выполнение нашей функции и выведем результаты работы программы в консоль. Для этого напишем следующий фрагмент.

```
file = "video/d.mp4"
car, truck = analiz_car(file)
print("vars =", car)
print("trucks =", truck)
```

Результаты работы программы на рисунках 3 и 4.

Рисунок 3 - Отображение работы программы в окне.

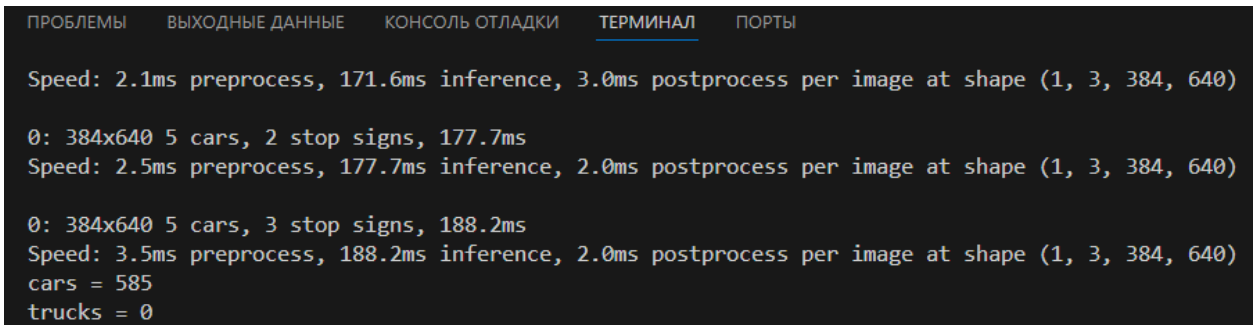

Рисунок 4 - Результаты работы программы в консоли.

Построение гистограммы

Для отображения результатов работы программы мы сделаем гистограмму.

В начале подключим необходимые библиотеки.

```
import matplotlib.pyplot as plt

import os
```

Функция принимает два аргумента, которые будут использованы для отображения статистики. Затем с помощью переменной output_dir, которая содержит путь к папке, где сохранен график, проверяется есть ли папка с таким названием. Если такой папки нет, она создается.

Переменные variable1, variable2 принимают значение аргументов, переданных в функцию. С помощью функции subplots создаем фигуру и ось построения графика. На графике строятся столбчатые диаграммы для значений variable1 и variable2. На оси X устанавливаются метки «cars», «trucks». На оси Y устанавливается метка «Values». set_ylim устанавливает то, что значение на Y должны начаться с нуля. Метод title устанавливает заголовок графика «Stats».

Затем график сохраняется в папке с названием “stats_car.png”. С помощью метода show отобразим наш график. Результат работы программы рисунок 5.

```
def stats_car(a, b):
    # Создаем папку "output", если она не существует
    output_dir = "results"
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    variable1 = a
    variable2 = b
    fig, ax = plt.subplots()
    ax.bar([variable1, variable2], [variable1,variable2])
    ax.set_xticks([variable1, variable2])
    ax.set_xticklabels(["cars", "trucks"])
    ax.set_ylabel('Values')
    ax.set_ylim(bottom=0)
    plt.title('Stats')
```

```
    plt.savefig(os.path.join(output_dir, "stats_car.png"))
    plt.show()
```

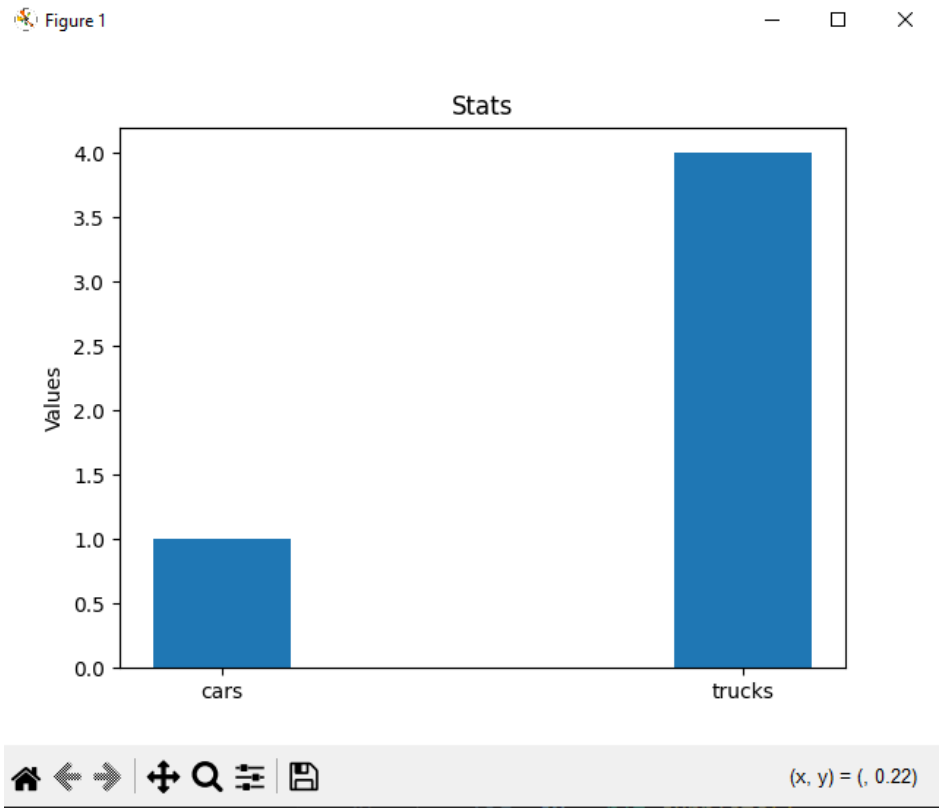

Рисунок 5 - График соотношения грузовых и легковых машин.

Аналогично сделаем функцию для построения графика людей.

```
def stats_people(a, b):
    output_dir = "results"
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    variable1 = a
    variable2 = b
    fig, ax = plt.subplots()
    ax.bar([variable1, variable2], [variable1,variable2])
    ax.set_xticks([variable1, variable2])
    ax.set_xticklabels(["male", "female"])
    ax.set_ylabel('Values')
    ax.set_ylim(bottom=0)
    plt.title('Stats')
    plt.savefig(os.path.join(output_dir, "stats_people.png"))
    plt.show()
```

Структура проекта

Наложим логику работы нашей программы на веб-разработку для создание простого интерфейса.

Для сборки конечного варианта нашего проекта мы будем применять Eel.

EEL – библиотека, позволяющая разработчику создавать простые электронные приложения с автономным графическим интерфейсом пользователя (GUI) на основе HTML и JavaScript с полным доступом к возможностям и библиотекам Python.

Начнем формирование структуры проекта:

Создадим несколько файлов python и создадим папку src, внутри которой создадим файл html и css (рисунок 6).

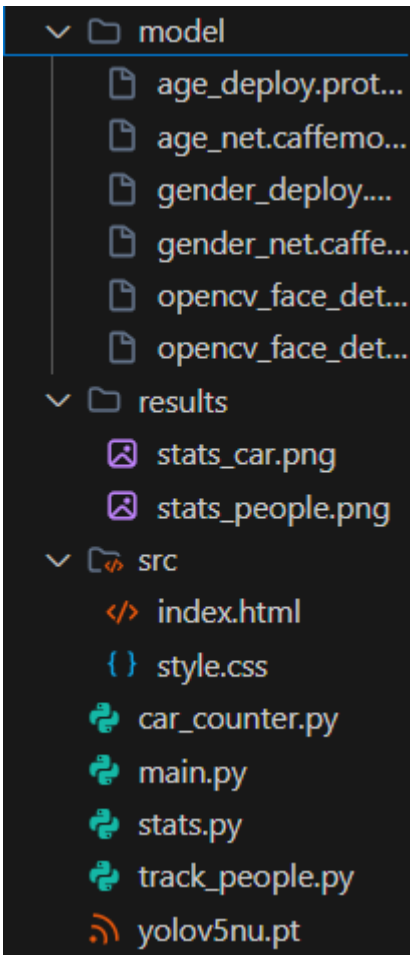

Рисунок 6 – Структура всего проекта.

В соответствующие файлы python вставим код (например, в файл car_counter код анализа видеопотока для машин).

Формирование файла main

Из файла main мы будем вызывать все остальные методы из других файлов, поэтому подключим необходимые зависимости. А сам файл main выступает в роли связующего звена между кодом на python и его вызовом через JS на HTML страницу.

```
import eel
```

```
from track_people import *

from car_counter import*

from stats import *

from PIL import Image
```

Чтобы мы могли вызвать любую функцию python в JS нам необходимо перед ее объявлением написать @eel.expose.А также необходимо прописать:

```
eel.init('src')

eel.start("index.html", size=(1920, 1080), mode="chrome")
```

Где eel.init это указание библиотеке eel, где искать HTML и JS файлы (в нашем случае JS код мы вставим внутрь файла HTML при помощи тэга script).

eel.start здесь мы указываем название HTML файла, размер окна 1920 на 1080 пикселей, Chrome – указание на то, что программа должна запускаться в браузере Chrome.

Переходим к формированию вспомогательных функций, которые мы вызовем через JS.

Сделаем функцию для запуска анализа людей. Для этого сохраним результат вызова функции в некоторые переменные и передадим количество мужчин и женщин в функцию отрисовки графика.

```
@eel.expose

def call_analiz():

    man, woman, unique_people_count = analiz()

    stats_people(man, woman)
```

Аналогично поступим и для машин, только будем передавать в функцию для графика количество легковых и грузовых авто.

```
@eel.expose

def call_analiz_car(file):

    det_cars, det_trucks = analiz_car(file)

    stats_car(det_cars, det_trucks)
```

Теперь напишем функцию для отображения графиков. Она принимает путь к изображению, с помощью метода open из библиотеки Pillow открывает его. Затем с помощью метода show изображение показывается.

```
@eel.expose

def show_stats(image_path):

    im = Image.open(image_path)

    im.show()
```

Работа с HTML файлом

Для использования библиотеки Eel в HTML файле подключим в тэг head:

```
<script type="text/javascript" src="/eel.js"></script>
```

Кнопкам присвоим атрибкт onclick, который активирует событие onclick в момент нажатия на кнопку.

```
<button class="submit" type="people_analiz"
onclick="people_analiz()"><p>Анализ людей</p></button>

<button class="analysis" type="car_analiz" onclick="openModal()"><p>Анализ
машин</p></button>

<button class="close" onclick="handleFilePathInput()">Анализировать</button>

<button type="submit" onclick="statistic_people()"><p>Статистика
людей</p></button>

<button type="analysis" onclick="statistic_car()"><p>Статистика
машин</p></button>
```

Теперь пропишем внутри тэга script асинхронные функции, которые будут вызываться в момент нажатия на кнопку.

Например, для анализа людей функция выглядит следующим образом:

```
async function people_analiz(){

              alert('Вы запустили выполнение программы, сверните окно и
перейдите в программу видеоанализа');

              await eel.call_analiz();

        };
```

people_analiz – это значение атрибута onclick у кнопки «Анализ людей»

alert – выдаст модальное окно с информацией для пользователя.

Аналогично поступим и для других функций:

```
async function car_track(file) {

            alert('Вы запустили выполнение программы, сверните окно и
перейдите в программу видеоанализа');

            await eel.call_analiz_car(file);

        }


        async function statistic_people() {

            const imagePath = "results/stats_people.png";
```

```
    await eel.show_stats(imagePath)
}

async function statistic_car() {
    const imagePath = "results/stats_car.png";
    await eel.show_stats(imagePath)
}
```

Был разработан простой и интуитивно понятный интерфейс. Итоговый вид приложения, рисунок 7.

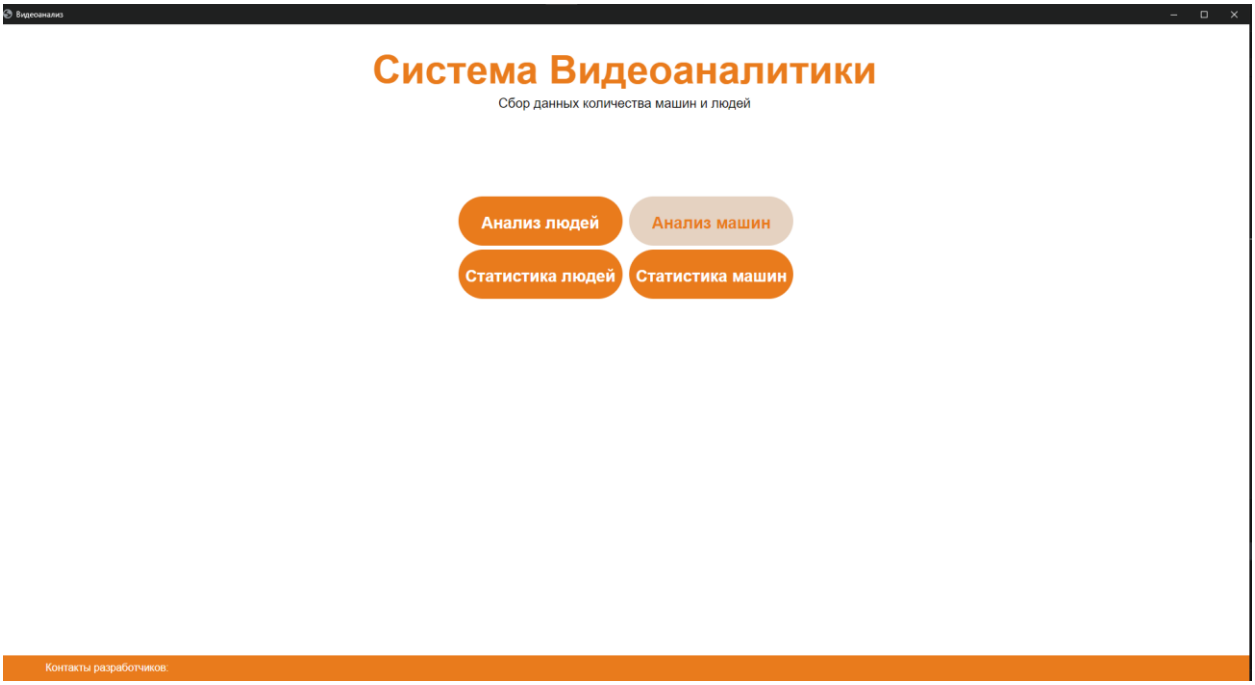

Рисунок 7 - Дизайн приложения.

Также для анализа машин было прописано модально окно, в котором пользователь вводит путь к файлу, рисунок 8.

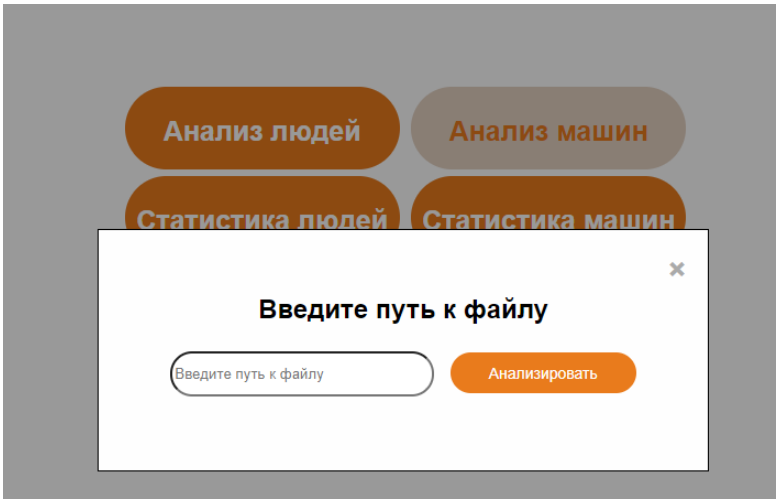

Рисунок 8 – Модальное окно

Выводы:

В рамках данной статьи мы изучили и рассмотрели на практике задачи, связанные с распознаванием, классификацией и трекингом объектов. При этом использовали возможности библиотеки OpenCV, модель YOLOv5. Продемонстрировали работу программы при создании веб приложения при помощи библиотеки Eel.

OpenCV (Open Source Computer Vision Library) - мощная библиотека компьютерного зрения с открытым исходным кодом, широко используемая в разработке приложений, связанных с обработкой и анализом изображений и видео. Она предоставляет множество алгоритмов и функций, которые позволяют выполнять задачи, такие как обнаружение объектов, отслеживание движения, распознавание лиц и многое другое.

YOLOv5 (You Only Look Once version 5) - одна из наиболее популярных моделей для обнаружения объектов в режиме реального времени. YOLOv5 отличается высокой точностью, скоростью обработки и относительной простотой использования по сравнению с другими моделями обнаружения объектов.

Результаты можно считать положительными. Программа работала, обрабатывая видео и выдавая ожидаемые результаты.

Но все же трудности возникали. Например, при трекинге машин было тяжело подобрать пороговое расстояние и при видео, на которых разные скорости машин и разное качество камер результаты будут с погрешностью.

Несмотря на некоторые ограничения, мы приобрели важные навыки:

- Научились работать в команде, распределять задачи между собой.
- Освоили базовые инструменты библиотеки OpenCV.
- Научились использовать возможности модели YOLOv5.
- Применяли свое креативное мышление и знания для разрешения возникающих трудностей.
- Развили в себе навык самообучения.
- Научились применять свои знания в прикладных отраслях.