Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №8**

**Дисциплина: Алгоритмы цифровой обработки мультимедиа**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.А.Мазницкий

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А. А. Крамаренко

**Тема работы:** Обнаружение объектов на изображении.

**Ход работы:**

Задание 1 (Самостоятельно). С помощью каскадов Хаара обработать видео с камеры в реальном времени и обнаружить ВСЕ лица на видео.

Задание 2 (Самостоятельно). Построить нейронную сеть на основе YOLO для обнаружения лиц на видео в реальном времени.

Задание 3 Сделать сводную таблицу сравнений результатов задания 1 и

задания 2.

Задание 4 Используя дообученные (информацию по обучению каскадов

Хаара можно найти тут https://habr.com/ru/articles/208092/) каскады Хаара обнаружить оружие на видео.

В лабораторной работе необходимо было решить задачу обнаружения объектов на изображении.

В первом задании необходимо было с помощью каскадов Хаара распознать лица. Определим, что это такое. Каскады Хаара (Haar cascades) представляют собой эффективный метод обнаружения объектов на изображениях. Этот метод был предложен Виолой и Джонсом в 2001 году и стал широко используемым в области компьютерного зрения и распознавания образов. Каскады Хаара являются основной частью алгоритма Viola-Jones для обнаружения лиц.

Вот основные шаги создания и использования каскадов Хаара:

1. **Собрание позитивных и негативных образцов:**
   * *Позитивные образцы:* Это изображения объектов, которые вы хотите обнаружить (например, лица).
   * *Негативные образцы:* Изображения без объектов, которые вы хотите обнаружить.
2. **Преобразование изображений в интегральное изображение:**
   * Это шаг, который ускоряет вычисления при работе с окнами различных размеров.
3. **Выбор признаков Хаара:**
   * Признаки Хаара представляют собой прямоугольные области с различными значениями пикселей.
   * Применяются фильтры, представляющие собой комбинации чередующихся черных и белых прямоугольников.
4. **Обучение классификатора:**
   * Используется метод Adaboost для комбинирования нескольких слабых классификаторов в сильный.
   * Каждый классификатор обучается на небольшом подмножестве данных.
5. **Создание каскада:**
   * Каскад представляет собой серию классификаторов, каждый из которых обучен на различных уровнях сложности.
   * Каскад имеет структуру, при которой объект считается обнаруженным только в том случае, если прошел через все этапы.
6. **Тестирование на новых данных:**
   * Каскад применяется к новым изображениям, и объекты обнаруживаются, если они соответствуют заданным характеристикам.

Преимущества каскадов Хаара включают высокую скорость обнаружения и возможность эффективного отбора регионов интереса. Однако, они могут иметь некоторые ограничения, такие как неспособность точно обрабатывать повороты и изменения масштаба объектов.

Для разработки использовался язык программирования Python. В интернете была найдена предварительно обученная модель для обнаружения лиц, после этого был разработан скрипт, для работы с этой моделью. На рисунке 1 представлен пример работы каскадов Хаара.

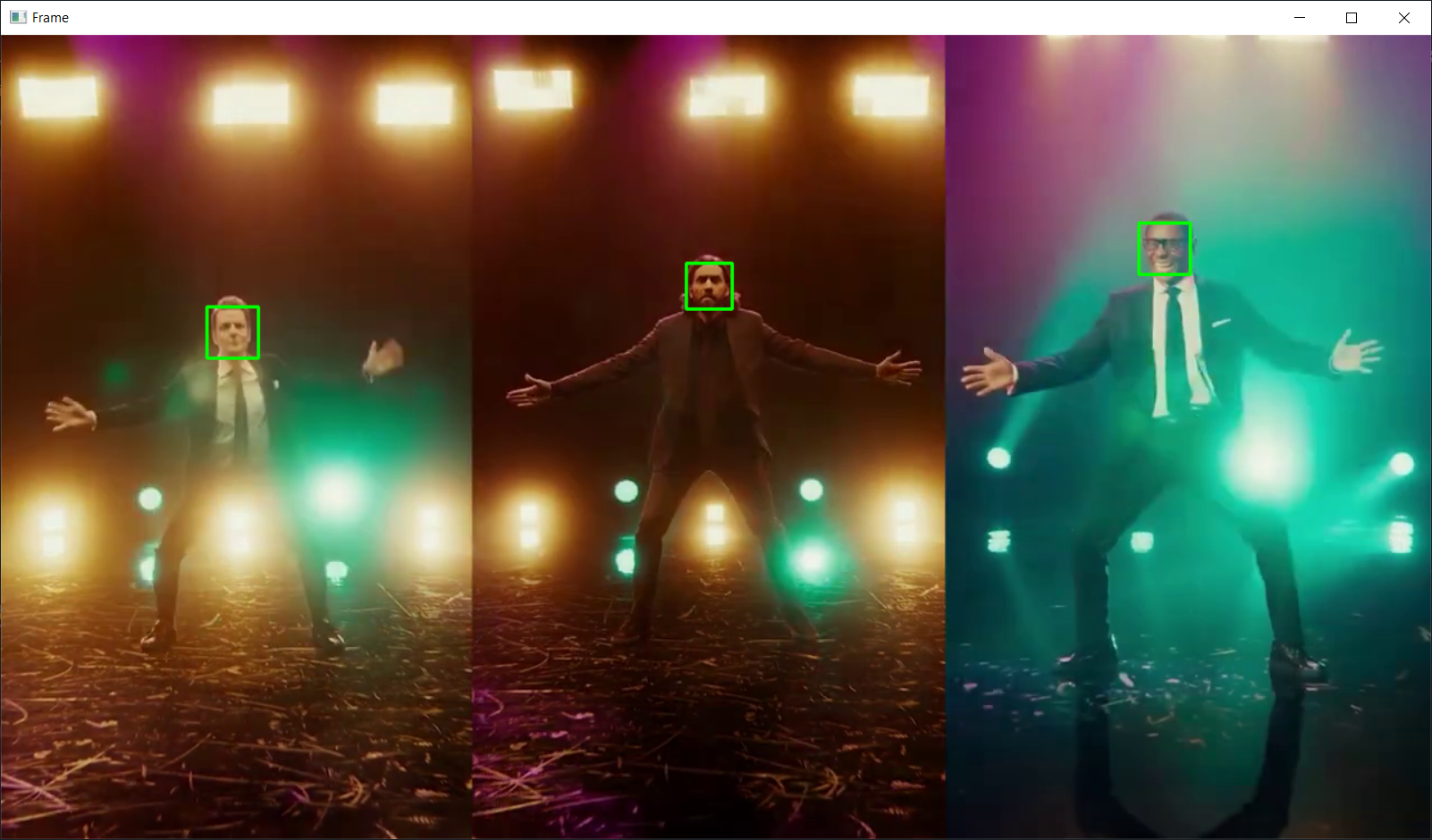


Рисунок 1 – Работа каскадов Хаара.

Во втором задании была поставлена задача сделать тоже самое, только с использованием YOLO. Это задание не представляется сложным, так как YOLO идеально обучена для обнаружения лиц. Пример работы YOLO представлен на рисунке 2.

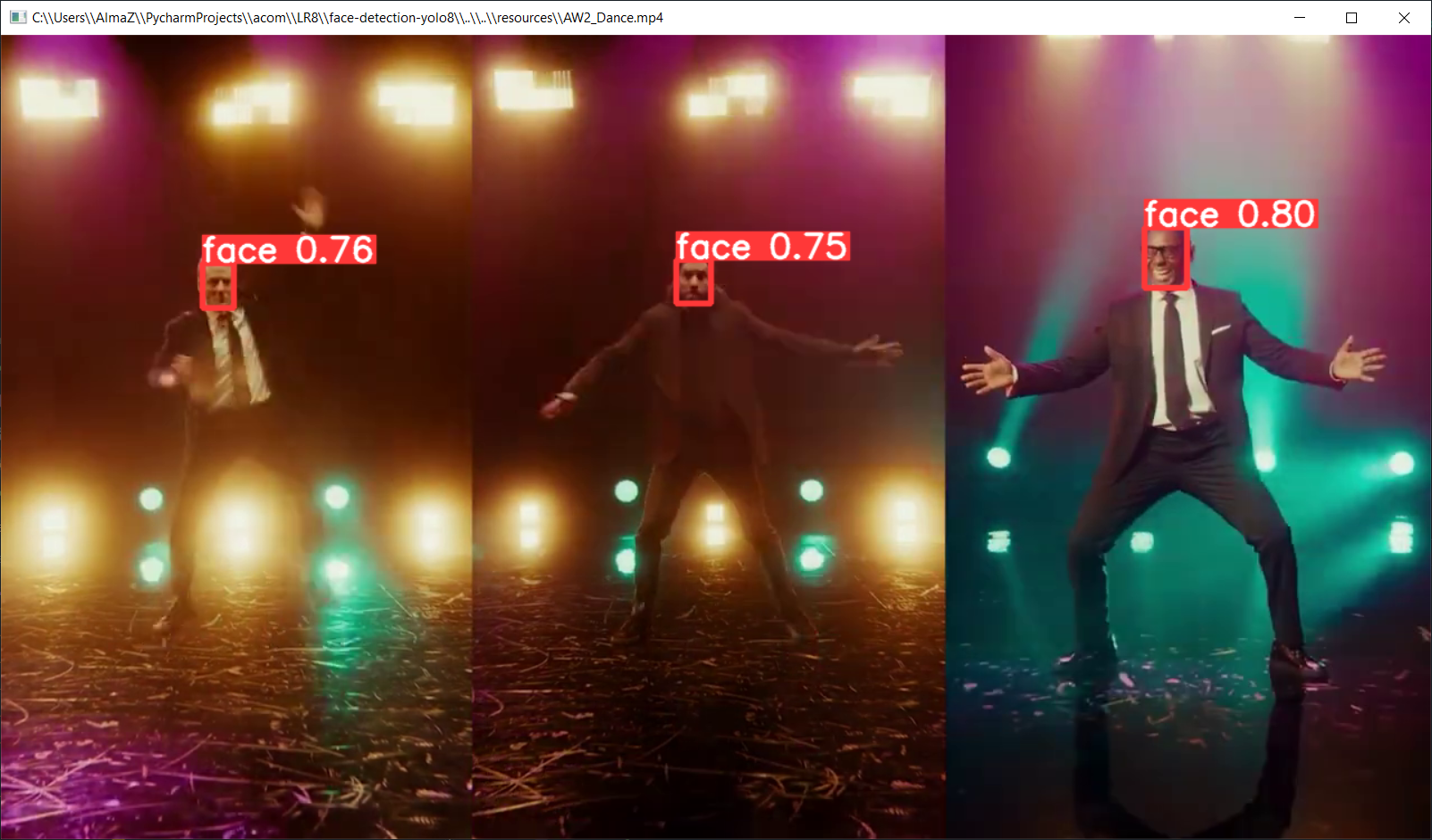


Рисунок 2 – Работа YOLO.

В следующем задании была поставлена задача сравнения этих способов обнаружения лиц с использованием сводной таблицы. В коде есть строчки кода, позволяющие следить за основными параметрами для сравнения этих алгоритмов. Результаты сравнения представлены на рисунке 3.

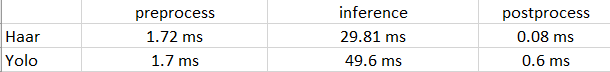


Рисунок 3 – Результаты сравнения YOLO и Каскадов Хаара.

В последнем задании была поставлена цель самостоятельного дообучения каскадов Хаара с целью обнаружения оружия в кадре в реально времени. Задача была несложной, так как тема каскадов Хаара уже давно довольно подробно изучена. Самым сложным оказалось создать датасет из положительных и отрицательных предметов данной предметной области и последующее её обучение. В результате обучения получились каскады Хаара, способные распознавать оружие в реальном времени, результат работы представлен на рисунке 4.

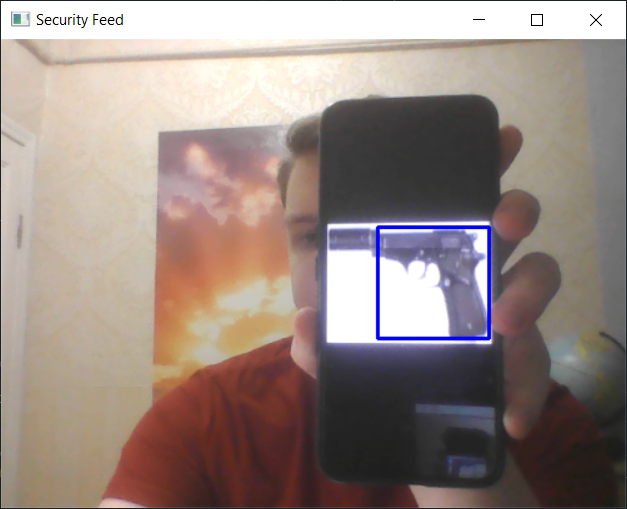


Рисунок 4 – Обнаружение оружия.

**Листинг программы**

Файл Haar\_Face\_Detection.py

*from* imutils.video *import* VideoStream  
*import* argparse  
*import* imutils  
*import* cv2  
*import* time  
*import* os  
  
*# construct the argument parser and parse the arguments*ap = argparse.ArgumentParser()  
ap.add\_argument("-c", "--cascade", type=*str*,  
 default="haarcascade\_frontalface\_default.xml",  
 help="path to haar cascade face detector")  
args = *vars*(ap.parse\_args())  
  
*# load the haar cascade face detector from*detector = cv2.CascadeClassifier(args["cascade"])  
  
*# time.sleep(2.0)*vid = cv2.VideoCapture(r'../resources/AW2\_Dance.mp4')  
  
  
*def* video\_rec():  
 preprocess\_times = []  
 inference\_times = []  
 postprocess\_times = []  
  
 *while True*:  
 start\_time = time.time()  
 ret, frame = vid.read()  
 *if not* ret:  
 *print*("Не могу прочитать кадр :(")  
 *break* gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 preprocess\_time = time.time() - start\_time  
 preprocess\_times.append(preprocess\_time)  
  
 start\_time = time.time()  
  
 rects = detector.detectMultiScale(gray, scaleFactor=1.15,  
 minNeighbors=8, minSize=(30, 30),  
 flags=cv2.CASCADE\_SCALE\_IMAGE)  
  
 inference\_time = time.time() - start\_time  
 inference\_times.append(inference\_time)  
  
 *print*(rects)  
  
 start\_time = time.time()  
  
 *for* (x, y, w, h) *in* rects:  
 cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)  
  
 postprocess\_time = time.time() - start\_time  
 postprocess\_times.append(postprocess\_time)  
  
 speed = {  
 'preprocess': *sum*(preprocess\_times) / *len*(preprocess\_times) *if* preprocess\_times *else* 0,  
 'inference': *sum*(inference\_times) / *len*(inference\_times),  
 'postprocess': *sum*(postprocess\_times) / *len*(postprocess\_times) *if* postprocess\_times *else* 0  
 }  
  
 *print*("speed:", speed)  
 cv2.imshow("Frame", frame)  
 *if* cv2.waitKey(1) & 0xFF == 27:  
 *break* vid.release()  
 cv2.destroyAllWindows()  
  
 total\_preprocess\_time = *sum*(preprocess\_times)  
 total\_inference\_time = *sum*(inference\_times)  
 total\_postprocess\_time = *sum*(postprocess\_times)  
  
 *print*("speed:")  
 *print*("total preprocess time:", total\_preprocess\_time)  
 *print*("total inference time:", total\_inference\_time)  
 *print*("total postprocess time:", total\_postprocess\_time)  
  
  
*def* camera\_detect():  
 vs = VideoStream(src=0).start()  
 *while True*:  
 frame = vs.read()  
 frame = imutils.resize(frame, width=500)  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 rects = detector.detectMultiScale(gray, scaleFactor=1.05,  
 minNeighbors=5, minSize=(30, 30),  
 flags=cv2.CASCADE\_SCALE\_IMAGE)  
  
 *print*(rects)  
 *for* (x, y, w, h) *in* rects:  
 cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)  
  
 cv2.imshow("Frame", frame)  
 *if* cv2.waitKey(1) & 0xFF == 27:  
 *break* cv2.destroyAllWindows()  
 vs.stop()  
  
  
*# camera\_detect()*video\_rec()