Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

# Отчет по лабораторной работе №8
# «Распознавание лиц методом Виолы-Джонса»
# по дисциплине «Техническое зрение»

Выполнил: студент гр. R3338,
Кирбаба Д.Д.
Преподаватель: Шаветов С.В.,
канд. техн. наук, доцент ФСУ и Р

Санкт-Петербург, 2022

## Цель работы

Освоение метода Виолы-Джонса для распознавания лиц и частей тела на изображении.

## Теоретическое обоснование применяемых методов

Метод Виолы-Джонса для распознавания лиц основан на следующих концептах:

1. Хаар-подобные особенности как «слабые» классификаторы.
2. Представления изображения в интегральном виде для быстрого вычисления Хаар-подобных особенностей.
3. Метод *AdaBoost* для преобразования «слабых» классификаторов в «сильные».
4. Объединение «сильных» классификаторов в каскад классификаторов.

### Хаар-подобные особенности

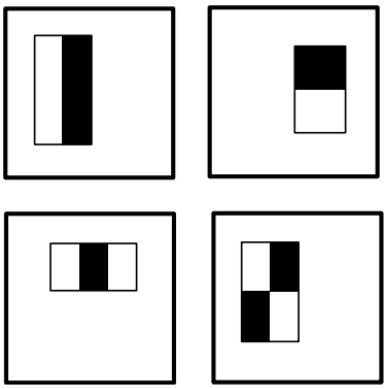

*Figure 1: Хаар-подобные особенности*

Хаар-подобные особенности можно считать, как «слабые» классификаторы. Они могут быть определены как разность суммы пикселей областей внутри прямоугольника, которые могут находиться в любой точке и масштабе в пределах исходного изображения.

Очевидно, что для прямого вычисления суммы значений пикселей в прямоугольнике потребуется количество слагаемых, равное количеству пикселей минус один.

Для ускорения вычисления признаков используется **интегральное представление изображения**. В этом представлении каждый пиксель хранит сумму всех значений пикселей, расположенных слева и сверху от текущего пикселя.

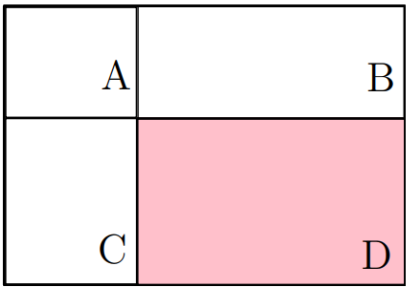


*Figure 2: Вычисление суммы в прямоугольнике с интегральным изображением.*

Для вычисления суммы значений интенсивности пикселей в произвольном прямоугольнике нам необходимо получить доступ к четырем пикселям интегрального изображения, которые расположены по углам прямоугольника.

$$sum = D - B - C + A$$

Набор Хаар-подобных признаков (которые являются «слабыми» классификаторами) может быть объединен в виде взвешенной суммы значений для формирования «сильного» классификатора.

Алгоритм обучения называется **AdaBoost**. Он состоит из нескольких стадий бустинга, и каждая стадия бустинга — это выбор лучшего «слабого» Хаар-подобного признака для классификации обучающего множества с учетом ошибок классификации предыдущих раундов.

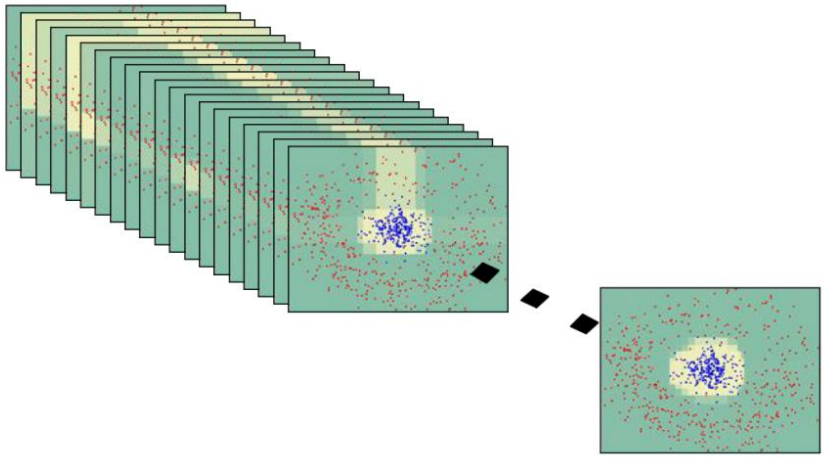

*Figure 3: объединение «слабых» классификаторов в «сильный».*

Основные шаги алгоритма AdaBoost:

1. На входе мы имеем тренировочное множество $T = \{(x_i, y_i)|x_i \in X, y_i \in \{-1,1\}\}$ и множество всех возможных «слабых» классификаторов $\{h\}$.

2. Присвоим одинаковые значения весам для тренировочного множества, такие что их сумма равна 1. $D_1(i) = \frac{1}{m}$, где $m$ – количество элементов в тренировочном множестве.

3. Сделаем $K$ итераций:
   a. Выберем $h_k$ из множества «слабых» классификаторов $\{H\}$, так, что вероятность взвешенной ошибки классификации минимальна (вероятность неверной классификации при выборе весов):
   $$\epsilon_k = Pr_{i \ D_k}[h_k(x_i) \neq y_i]$$
   b. Рассчитаем веча текущего «слабого» классификатора, основываясь на его вероятности ошибки классификатора:
   $$\alpha_k = \frac{1}{2}\ln\left(\frac{1-\epsilon_k}{\epsilon_k}\right)$$
   c. Перераспределим веча для тренировочного множества:
   $$D_{k+1}(i) = \frac{D_k(i)}{Z_k} \cdot \begin{cases} e^{-\alpha_k}, h_k(x_i) = y_i \\ e^{\alpha_k}, h_k(x_i) \neq y_i \end{cases}$$

4. После завершения $K$ итераций сформируем «сильный» классификатор как взвешенную сумму «слабых» классификаторов, которые были выбраны во время стадий бустинга:

$$H(x) = sign(\sum_{k=1}^{K} \alpha_k h_k(x))$$

**Каскадные классификаторы**

Сильный классификатор, для достижения требуемой точности, может потребовать вычисления слишком большого количества слабых классификаторов, что замедлит скорость обнаружения, учитывая, что большинство сканируемых окон не содержат лиц. Для ускорения скорости обнаружения набор классификаторов с возрастающей сложностью и коэффициентом обнаружения организуется в каскад классификаторов.

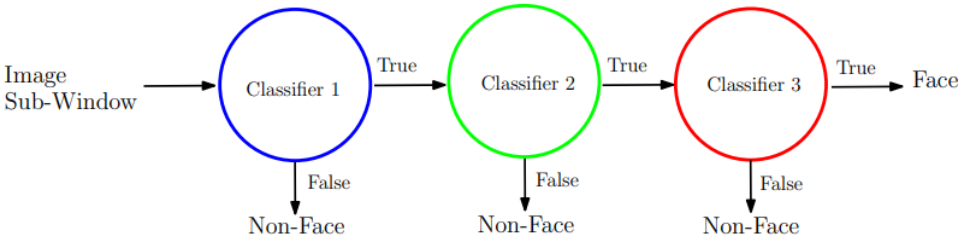


*Figure 4: каскадный классификатор.*

Чтобы быть классифицированным положительно, скользящее окно должно пройти все каскадные этапы. В случае, если какой-либо классификатор отвергает окно, оно немедленно отвергается, и детектор переходит к следующему окну. В результате большинство неверных окон быстро отвергаются первыми быстрыми классификаторами в каскаде.

Коэффициент обнаружения ($true\ positive\ rate, TP$) каскадного классификатора представляет собой умножение коэффициентов обнаружения всех классификаторов в каскаде:

$$TP = \prod_i TP_i$$

Коэффициент ложных срабатываний ($FP$) также представляет собой умножение числа ложных срабатываний каскадных классификаторов:

$$FP = \prod_i FP_i$$

В результате, чтобы построить классификатор с коэффициентами $TP = 0.9, FP = 10^{-6}$ , каждый классификатор в каскаде должен удовлетворять требованию $TP = 0.99, FP = 0.3$.

Каждый классификатор каскада обучается с помощью схемы обучения AdaBoost с требованием максимизации коэффициента обнаружения истинных положительных результатов при сохранении ложных положительных результатов в заданном диапазоне. Обучающий набор изменяется между стадиями бустинга, чтобы увеличить сложность каждого шага каскада.

https://github.com/Itseez/opencv/tree/master/data/haarcascades - готовые модели для распознавания различных частей тела.

## Ход выполнения работы

### 1. Распознавание лиц

Исходные изображения:

*Figure 5: изображение №1.*


*Figure 6: изображение №2.*

*Figure 7: изображение №3.*

Найдем лица на всех фотографиях и посчитаем количество найденных лиц, используя метод Виолы-Джонса.

*Listing 1. Поиск и подсчет лиц на изображении методом Виолы-Джонса, используя OpenCV и Python.*

```
import cv2 as cv


# Reading an image
img_path = "../images/"
img_name = "faces.jpg"
img = cv.imread(img_path + img_name, cv.IMREAD_COLOR)


# Create and load a cascade classifier faces
detector = cv.CascadeClassifier()
cascade_fn = cv.samples.findFile("haarcascade_frontalface_default.xml")
detector.load(cascade_fn)


# Execute a cascade classifier for faces using scale factor 1.07 and 3 minimum
required number of matches
faces = detector.detectMultiScale(img, scaleFactor=1.07, minNeighbors=3)
```

```
# Count number of faces
faces_num = len(faces)


# Display found faces
img_out = img.copy()
for (x, y, w, h) in faces:
    img_out = cv.rectangle(img_out, (x, y, w, h), (255, 0, 0), 1)
cv.imwrite(img_path + img_name.rpartition('.')[0] + "_faces.jpg", img_out)
```

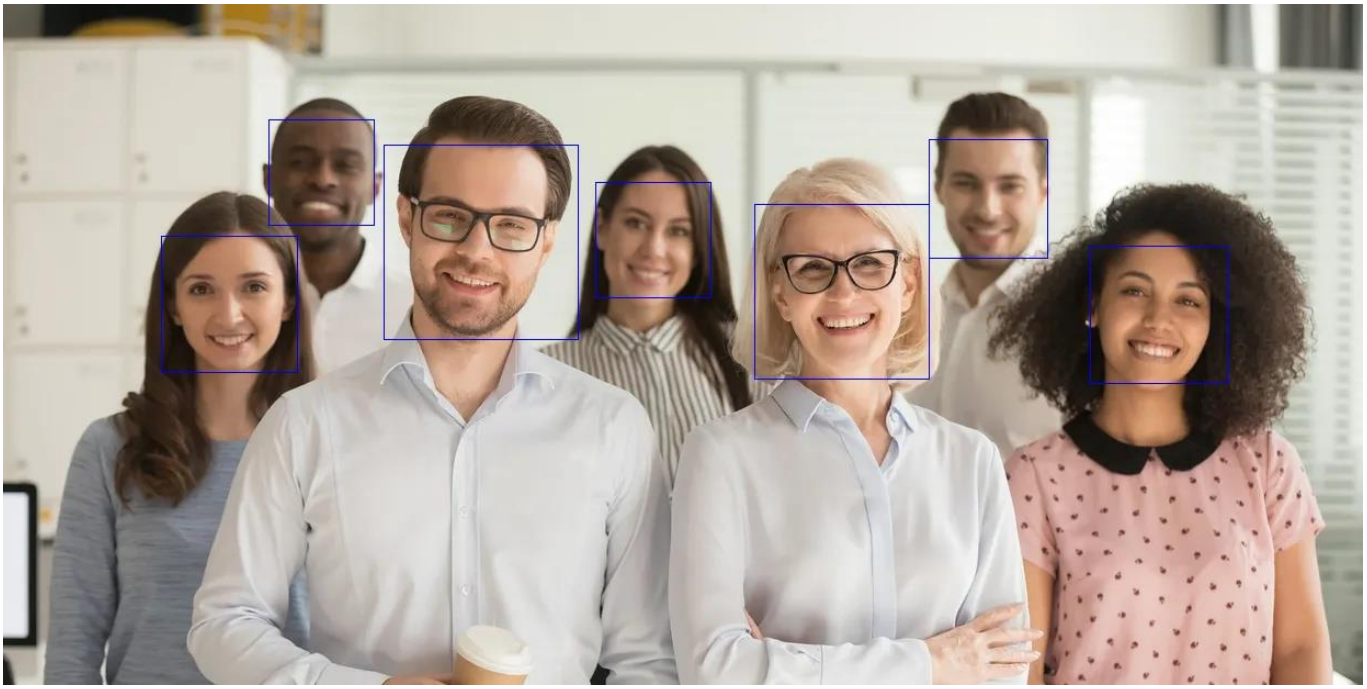

*Figure 8: определение лиц на изображении №1.*

Количество найденных лиц $= 7$.

Значения параметров: $scaleFactor = 1.07,\ minNeighbors = 13$.


*Figure 9: определение лиц на изображении №2.*

Количество найденных лиц $= 10$.

Значения параметров: $scaleFactor = 1.07,\ minNeighbors = 8$.

Будем уменьшать значение параметра $minNeighbors$ до тех пор, пока не определим все лица на изображении.


*Figure 10: определение лиц на изображении №2.*

Удалось определить все лица при параметре $minNeighbor = 4$ ($num_{faces} = 14$), однако также выделились объекты, не являющимися лицами.

*Figure 11: определение лиц на изображении № 3.*

Количество найденных лиц $= 23$.

Значения параметров: $scaleFactor = 1.07,\ minNeighbors = 6$.

Многие лица не были найдены, так как они повернуты, а использованный классификатор не натренирован для нахождения таких типов лиц.

## 2. Определение частей лица

Исходные изображения:

*Figure 12: изображение №1.*

*Figure 13: изображение №2.*

*Figure 14: изображение №3.*

В данном пункте будем определять лица на фотографии, затем используя *ROI* будем определять глаза, нос и рот на лице человека.

*Listing 2. Определение лица и частей лица на изображении, используя OpenCV и Python.*

```
import cv2 as cv


# Reading an image
img_path = "../images/"
img_name = "faces_13.jpg"
img = cv.imread(img_path + img_name, cv.IMREAD_COLOR)


# Create and load a cascade classifiers
face_detector = cv.CascadeClassifier()
cascade_face = cv.samples.findFile("haarcascade_frontalface_default.xml")
face_detector.load(cascade_face)

eyes_detector = cv.CascadeClassifier()
cascade_eyes = cv.samples.findFile("haarcascade_eye.xml")
eyes_detector.load(cascade_eyes)

nose_detector = cv.CascadeClassifier()
cascade_nose = cv.samples.findFile("haarcascade_nose.xml")
nose_detector.load(cascade_nose)
```

```
mouth_detector = cv.CascadeClassifier()
cascade_mouth = cv.samples.findFile("haarcascade_mouth.xml")
mouth_detector.load(cascade_mouth)


img_all = img.copy()


# Execute a cascade classifier for faces
faces = face_detector.detectMultiScale(img, scaleFactor=1.07, minNeighbors=15)
# Count number of faces
faces_num = len(faces)
print(faces_num)
# Display found faces
img_faces = img.copy()
for (x, y, w, h) in faces:
    img_faces = cv.rectangle(img_faces, (x, y, w, h), (255, 0, 0), 2)
    img_all = cv.rectangle(img_all, (x, y, w, h), (255, 0, 0), 2)
cv.imwrite(img_path + img_name.rpartition('.')[0] + "_faces.jpg", img_faces)


# Detect and display eyes with taking face areas into an account
# For each face use it as a ROI and detect eyes
eyes_num = 0
img_eyes = img.copy()
for (x, y, w, h) in faces:
    # Define ROI and top 2/3 of the face image
    img_face_top_2_3 = img[y:y+h*2//3, x:x+w]
    eyes = eyes_detector.detectMultiScale(img_face_top_2_3, scaleFactor=1.05,
minNeighbors=50)
    eyes_num += len(eyes)
    for (x2, y2, w2, h2) in eyes:
        img_eyes = cv.rectangle(img_eyes, (x+x2, y+y2, w2, h2), color=(0, 255, 255))
        img_all = cv.rectangle(img_all, (x+x2, y+y2, w2, h2), color=(0, 255, 255))
cv.imwrite(img_path + img_name.rpartition('.')[0] + "_eyes.jpg", img_eyes)
print(eyes_num)


# Detect and display noses with taking face areas into an account
# For each face use it as a ROI and detect noses
noses_num = 0
img_noses = img.copy()
for (x, y, w, h) in faces:
    # Define ROI and bot 3/4 of the face image
    img_face_bot_3_4 = img[y+h//4:y+h, x:x+w]
    noses = nose_detector.detectMultiScale(img_face_bot_3_4, scaleFactor=1.05,
minNeighbors=10)
    noses_num += len(noses)
    for (x2, y2, w2, h2) in noses:
        img_noses = cv.rectangle(img_noses, (x+x2, y+h//4+y2, w2, h2), color=(0, 0,
255))
        img_all = cv.rectangle(img_all, (x+x2, y+h//4+y2, w2, h2), color=(0, 0,
255))
cv.imwrite(img_path + img_name.rpartition('.')[0] + "_noses.jpg", img_noses)
print(noses_num)


# Detect and display mouths with taking face areas into an account
# For each face use it as a ROI and detect mouths
```

```
mouths_num = 0
img_mouths = img.copy()
for (x, y, w, h) in faces:
    # Define ROI and bot 1/2 of the face image
    img_face_bot_1_2 = img[y+h//2:y+h, x:x+w]
    mouths = mouth_detector.detectMultiScale(img_face_bot_1_2, scaleFactor=1.05,
minNeighbors=100)
    mouths_num += len(mouths)
    for (x2, y2, w2, h2) in mouths:
        img_mouths = cv.rectangle(img_mouths, (x+x2, y+h//2+y2, w2, h2), color=(0,
255, 0))
        img_all = cv.rectangle(img_all, (x+x2, y+h//2+y2, w2, h2), color=(0, 255,
0))
cv.imwrite(img_path + img_name.rpartition('.')[0] + "_mouths.jpg", img_mouths)
print(mouths_num)


cv.imwrite(img_path + img_name.rpartition('.')[0] + "_all.jpg", img_all)
```

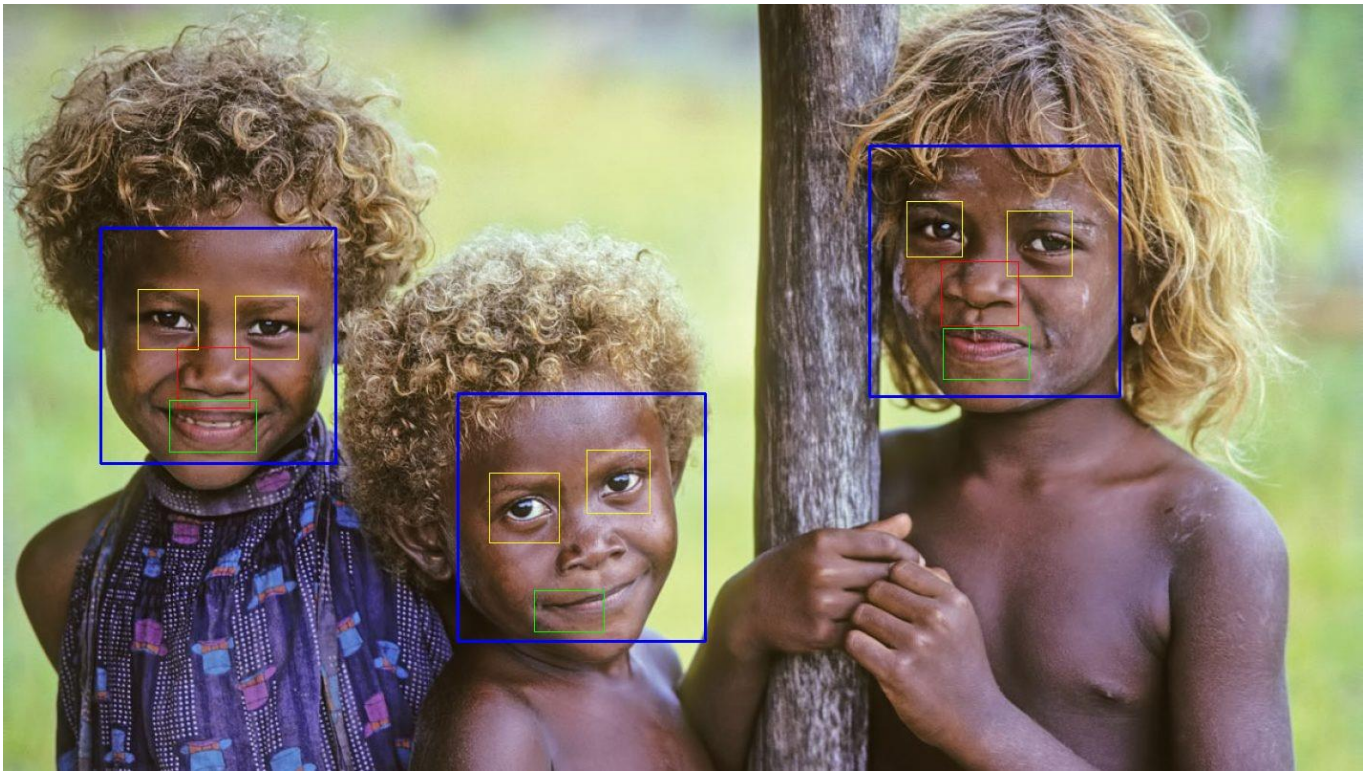

*Figure 15: результат поиска лица, глаз, носа и рта на изображении №1.*

Количество лиц = 3 ($scaleFactor = 1.07, minNeighbors = 15$).

Количество глаз = 6 ($scaleFactor = 1.05, minNeighbors = 50, ROI = top\ \frac{2}{3}$).

Количество носов = 2 $\left(scaleFactor = 1.05, minNeighbors = 10, ROI = bot\frac{3}{4}\right)$.

Количество ртов = 3 $\left(scaleFactor = 1.05, minNeighors = 100, ROI = bot\frac{1}{2}\right)$.

Результат определения частей лица достаточно хороший, не удалось лишь найти нос у ребенка посередине, из-за того, что он расположен под углом к горизонтали, а классификатор не обучался на таких примерах.

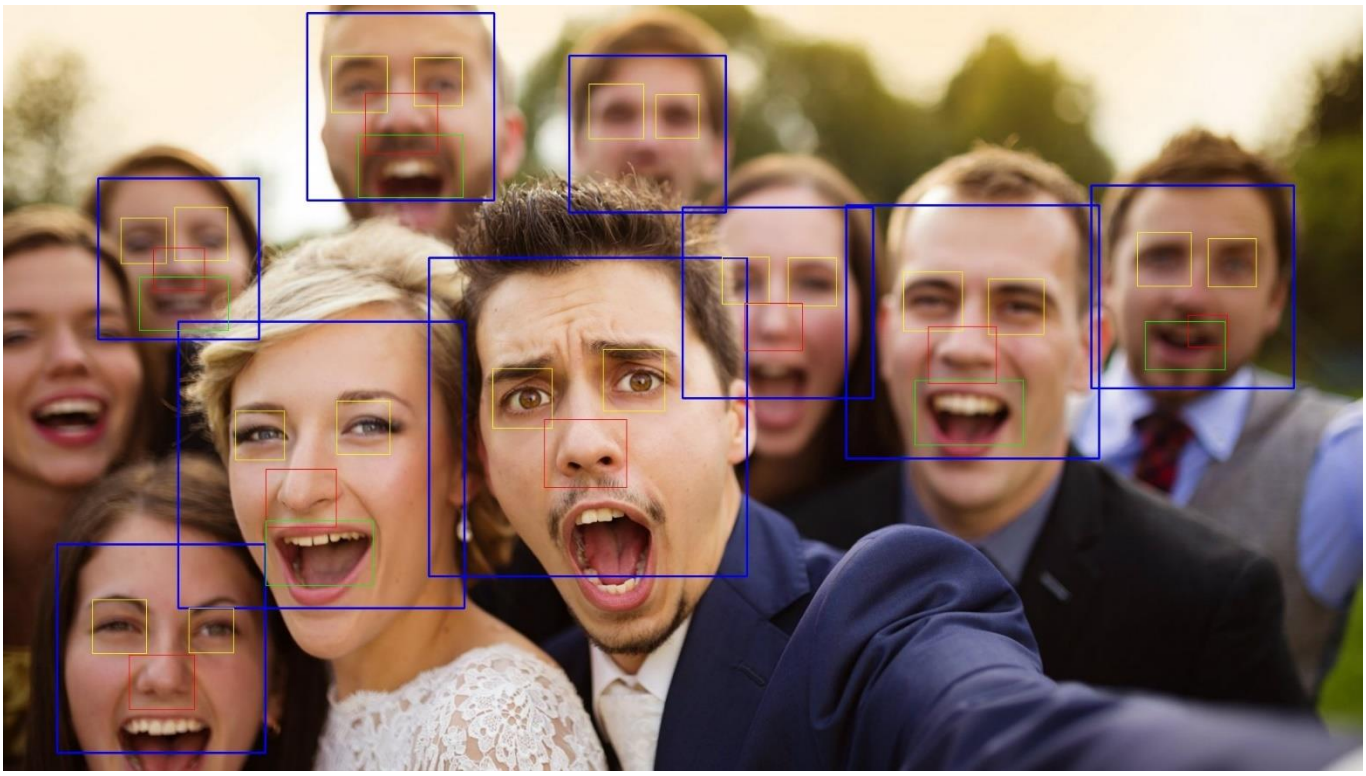

*Figure 16: результат поиска лица, глаз, носа и рта на изображении №2.*

Количество лиц $= 9$ $(scaleFactor = 1.07, minNeighbors = 10)$.

Количество глаз $= 18$ $(scaleFactor = 1.05, minNeighbors = 30, ROI = top\ \frac{2}{3})$.

Количество носов $= 8$ $\left(scaleFactor = 1.05, minNeighbors = 6, ROI = y{:}\, bot\frac{3}{4},\ x{:}\, middle\frac{5}{7}\right)$.

Количество ртов $= 5$ $\left(scaleFactor = 1.05, minNeighors = 10,\ minSize = (img.shape[1]//\right.$
$\left.20, img.shape[0]//20), ROI = bot\frac{1}{2}\right)$.

Результат неплохой. Лицо девушки слева не удалось распознать, так как оно обрезано, а проблема распознавания ртов объясняется тем, что они слишком открыты и классификатор не умеет работать со ртами такого типа.

*Figure 17: поиск лица, глаз, носа и рта на изображении №3.*

Количество лиц $= 5$ $(scaleFactor = 1.07, minNeighbors = 5)$.

Количество глаз $= 10$ $(scaleFactor = 1.05, minNeighbors = 30, ROI = top\ \frac{2}{3})$.

Количество носов $= 5$ $\left(scaleFactor = 1.05, minNeighbors = 20, ROI = y{:}\, bot\frac{3}{4},\ x{:}\, middle\frac{5}{7}\right)$.

Количество ртов $= 4$ $\left(scaleFactor = 1.05, minNeighors = 10,\ minSize = (img.shape[1]//20, img.shape[0]//20), ROI = bot\frac{1}{2}\right)$.

### 3. Определение лица на видео

В данном пункте предполагается выполнить определение лица на заранее записанном видео.

Для этого будем использовать класс $VideoCapture()$ библиотеки $OpenCV$, который предназначен для захвата видео из файлов, последовательностей изображений или камер.

Для сохранения результирующего видео будем использовать класс $VideoWriter()$.

На каждом фрейме видео будем применять детектор лиц Виолы-Джонса и отображать найденный прямоугольник в реальном времени.

*Listing 3. Определение лица на заранее записанном видео, используя OpenCV и Python.*

```
import cv2 as cv

# Read video
video_path = "../videos/"
video_name = "video.mp4"
video = cv.VideoCapture(video_path + video_name)

# Create and load a cascade classifiers
face_detector = cv.CascadeClassifier()
cascade_face = cv.samples.findFile("haarcascade_frontalface_default.xml")
face_detector.load(cascade_face)

# Check if video opened successfully
if not video.isOpened():
  print("Error opening video file")

# Get resolutions and fps of video
video_width = int(video.get(3))
video_height = int(video.get(4))
size = (video_width, video_height)
frame_rate = int(video.get(cv.CAP_PROP_FPS))

# Create VideoWriter object
writer = cv.VideoWriter(video_path + video_name.rpartition('.')[0] +
"_face_detection.avi", cv.VideoWriter_fourcc('M','J','P','G'), frame_rate, size)

# Read until video is completed
while video.isOpened():
    # Take a single frame of video
    # Method read() grabs, decodes and returns the next video frame
    retval, image = video.read()

    # Check if video is ended
    if not retval:
        break

    # Convert frame to grayscale
    image_gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

    # Detect faces
    # Execute a cascade classifier for faces
    faces = face_detector.detectMultiScale(image_gray, scaleFactor=1.1,
minNeighbors=7)

    # Display found faces
    for (x, y, w, h) in faces:
        cv.rectangle(image, (x, y, w, h), (255, 0, 0), 2)
    cv.imshow("Video", image)

    # Write the frame
    writer.write(image)

    # Display frame for 1 ms and exit if 'q' is pressed
    if cv.waitKey(1) & 0xFF == ord('q'):
        break
```

```
# Closing video file and writer
video.release()
writer.release()
cv.destroyAllWindows()
```

Определение лица на первом видео.

Определение лиц на втором видео.

Определение лиц на третьем видео.

Определение лиц на четвертом видео.

В итоге самая простая реализация определения лиц на видео сходится к покадровому определению лиц.

## 4. Определение лица в реальном времени

Теперь немного преобразуем программу, для того чтобы изменить поток ввода с видео файла в памяти на видео, получаемой с веб-камеры в реальном времени.

*Listing 4. Определение лица на веб-камере в реальном времени, используя OpenCV и Python.*

```
import cv2 as cv

# Set path
video_path = "../videos/"

# Set the video source to the default webcam
camera = cv.VideoCapture(0)

# Create and load a cascade classifiers
face_detector = cv.CascadeClassifier()
cascade_face = cv.samples.findFile("haarcascade_frontalface_default.xml")
face_detector.load(cascade_face)

# Check if camera opened successfully
if not camera.isOpened():
  print("Error opening video stream.")

# Get resolutions and fps of camera
video_width = int(camera.get(3))
video_height = int(camera.get(4))
size = (video_width, video_height)
frame_rate = int(camera.get(cv.CAP_PROP_FPS))

# Create VideoWriter object
writer = cv.VideoWriter(video_path + "webcam_face_detection.avi",
cv.VideoWriter_fourcc('M','J','P','G'), frame_rate, size)

# Main cycle
while True:
    # Take a single frame of video
    # Method read() grabs, decodes and returns the next video frame
```

```
    retval, image = camera.read()

    # Check if video is ended
    if not retval:
        break

    # Convert frame to grayscale
    image_gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

    # Detect faces
    # Execute a cascade classifier for faces
    faces = face_detector.detectMultiScale(image_gray, scaleFactor=1.1,
minNeighbors=5)

    # Display found faces
    for (x, y, w, h) in faces:
        cv.rectangle(image, (x, y, w, h), (255, 0, 0), 2)
    cv.imshow("Video", image)

    # Write the frame
    writer.write(image)

    # Display frame for 1 ms and exit if 'q' is pressed
    if cv.waitKey(1) & 0xFF == ord('q'):
        break

# Closing video file and writer
camera.release()
writer.release()
cv.destroyAllWindows()
```

Результат работы программы.

## Выводы

В данной работы был исследован и применен на практике алгоритм определения лиц на изображении методом Виолы-Джонса.

В первой части работы были найдены лица людей на изображениях с достаточно хорошей точностью, однако при каждом новом изображении требовалась настройки параметров для успешного распознавания.

Во второй части работы метод Виолы=Джонса был применен не только для поиска лиц, но и для других частей лица. Результаты данного поиска тоже хорошие и также была проведена индивидуальная настройка параметров для каждого изображения и каждой искомой части лица.

В двух заключительных частях велась работа с распознаванием лиц на видео (в заранее записанных и при записи с видеокамеры в реальном времени). Результаты в данных случаях плохие, так как используемая каскадная модель была натренирована для переднего плана лиц, без поворота, тогда как на видео таких условий сложно соблюсти. Также проблема была в скорости работы алгоритма, время работы программы на определение лиц на видео длиной 2 минуты, 60fps, full hd составляло порядка 6–7 минут (то есть в 3 раза дольше, чем длина видео).

Итого, метод Виолы-Джонса можно применять успешно для распознавания объектов как на изображениях, так и на видео (даже в реальном времени, при оптимизации алгоритмов и использовании хорошо обученных под требуемую задачу моделей).