**Министерство науки и высшего образования Российской Федерации**
**Федеральное государственное бюджетное 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 (ИУ)

КАФЕДРА Информационная безопасность (ИУ8)

# ОБНАРУЖЕНИЕ И РАСПОЗНАВАНИЕ СИГНАЛОВ

# ДОМАШНЕЕ ЗАДАНИЕ

**Вариант 2**

**Преподаватель:**

Ковынёв Н.В.

**Студент**:

Шапран А.В.

**Группа:**

ИУ8-84

Москва 2020

## Оглавление

# 1 Постановка задачи

Вариант 2. Реализовать нейронную сеть, добавляющую на фото фильтры snapchat.

На рисунке 1 представлен пример входных данных, а на рисунке 2 – пример выходных данных.

Рисунок 1 - Пример входных данных

Рисунок 2 - Пример выходных данных

# 2 Калибровка камеры

Выбранная для работы с изображениями библиотека OpenCV использует проекционную модель, известную как камера-обскура, что описано в документации OpenCV, 2013 [1]. В этой модели существует несколько неизвестных, которые необходимы для преобразования точек изображения на плоскости в точки объекта в реальном пространстве. На рисунке 3 изображена данная модель. В данном рисунке центр проекции обозначен буквой *O*, для упрощения принципиальная ось сделана параллельной оси *Z*. Плоскость изображения находится на расстоянии *f* от *O*, где *f* - фокусное расстояние.

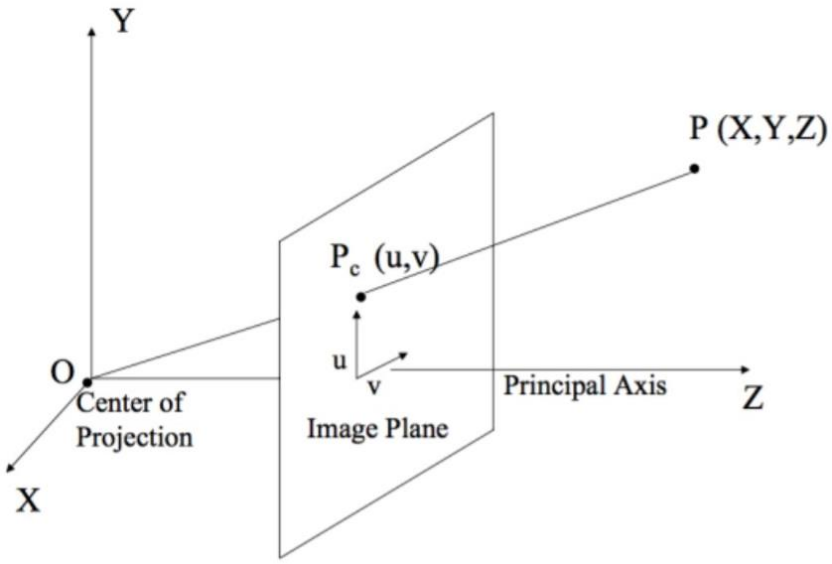


Рисунок 3 – Модель камеры-обскура (Joshi, 2015)

Координата плоскости камеры $P_C$ находится в положении *(u, v, w)*, где *w* является постоянной величиной. Координаты *(u, v, w)* вычисляются следующим образом:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

Поскольку плоскость изображения может не совпадать с местом, где ось $Z$ пересекает плоскость изображения, $P_C$ необходимо перевести в нужное начало с помощью смещений $(t_u, t_v)$. Новые однородные координаты следующие:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} f & 0 & t_u \\ 0 & f & t_v \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

Также необходимо учитывать масштаб пикселей. Если точка $P$ выражена в сантиметрах, а $P_C$ измеряется в пикселях, нужно включить значения $m_u$ и $m_v$, представляющие собой пиксели деленные на сантиметры. Тогда окончательное уравнение будет следующим:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} m_u f & 0 & m_u t_u \\ 0 & m_v f & m_v t_v \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} a_x & 0 & u_0 \\ 0 & a_y & v_0 \\ 0 & 0 & 1 \end{pmatrix} P = KP$$

*K* представляет собой коэффициент перекоса/наклона, если координатные оси *u* и *v* не ортогональны друг другу. Эта матрица называется матрицей

внутренних параметров камеры, как сказано в документации OpenCV, 2013 [1], и определяется следующим образом:

$$K = \begin{pmatrix} a_x & s & u_0 \\ 0 & a_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}$$

## 3 Каскады Хаара

У лица есть много особенностей, которые можно использовать для уникальной классификации людей. Большое хранилище данных изображений позволяет обучить каскад и классифицировать признаки Хаара в соответствии с полученным обучением. Признаки Хаара рассматривают только смежные пространственные прямоугольники в обнаруживаемой области, а затем задают разность суммы пикселей в белой области, определенной из черной области, чтобы дать относительный признак, как описано в статье Viola, P. & Jones, M., 2001. Rapid object detection using a boosted cascade of simple features [2]. На рисунке 4 представлены признаки Хаара, отмеченные на фронтальном изображении лица человека.

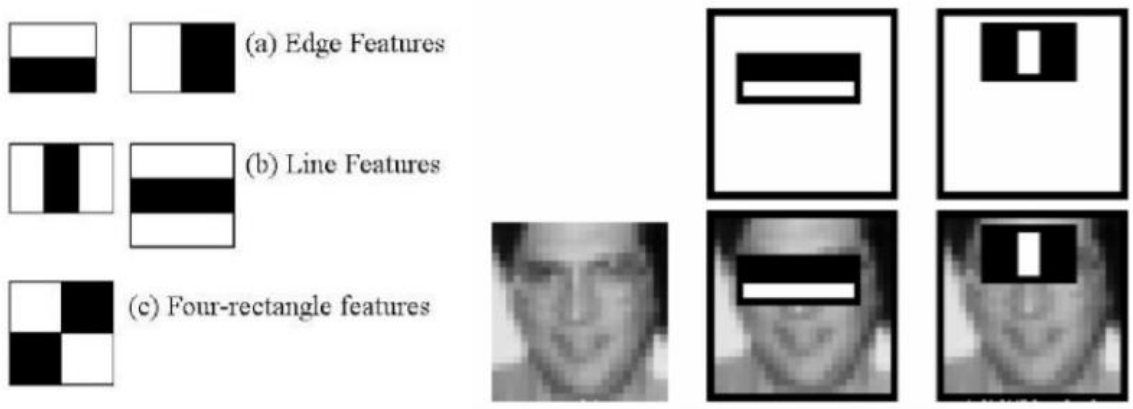


Рисунок 4 – Признаки Хаара, отмеченные на фронтальном изображении лица человека

На лицах есть общие черты, которые приводят к различным наборам ограничивающих рамок на лице, например, область вокруг глаз обычно темнее, чем область скул. Следовательно, две граничные особенности могут быть сгенерированы смежно. Данное заключение представлено в статье Papageorgiou, C. P., Oren, M. & Poggio, T., 1998. A General Framework for Object

Detection [3]. Это обнаружение генерирует сверточное ядро, которое работает от пикселей в самом изображении, чтобы получить сумму пикселей в прямоугольном элементе, что рассматривается в статье Viola, P. & Jones, M., 2001. Rapid object detection using a boosted cascade of simple features [2]. Затем наборы объектов могут быть наложены обратно на лицо, чтобы приблизительно показать местоположение отмеченных объектов, таких как лоб, подбородок, глаз, нос, челюсть, рот.

$$area = I(A) - I(B) + I(C) - I(D)$$

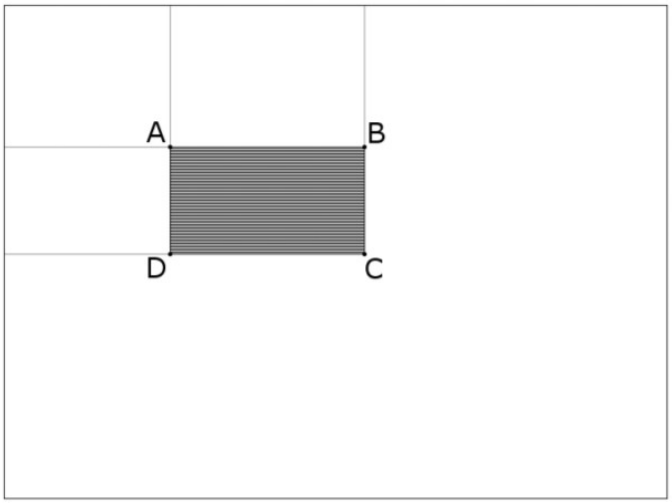


Рисунок 5 – Изображение суммы пикселей в прямоугольном элементе

## 4 Контуры лица. Особые точки

Каскады Хаара, как правило, требуют меньших вычислительных затрат по сравнению с другими методами, но они дают только приблизительную площадь объектов и выходят из строя при перекосах и поворотах изображения. Поэтому для точного определения лица на изображении, необходимо использовать более сложные ориентиры. Библиотека DLIB - это современный фреймворк для машинного обучения, написанный на C++. Она сочетает алгоритмы обнаружения и инструменты для решения реальных задач, как сказано в документации библиотеки DLIB C++ Library [4]. В инфраструктуру недавно было добавлено несколько новых функций, в том числе функция обнаружения лицевых контуров в реальном времени посредством выравнивания лица в одну миллисекунду с помощью множества деревьев

регрессии. Библиотека быстро добавляет аннотации 68 лицевых контуров даже при сильных наклонах. DLIB использует набор данных HELEN, состоящий из 2000 тренировочных изображений с различными освещениями, позами и окклюзиями. Описание того, как проводилось тестирование изображений, описано в статье Le, V. et al., 2012. Interactive Facial Feature Localization, Urbana: s.n [5].

Поскольку DLIB быстро находит новые ориентиры, необходимо использовать точки привязки (особые точки), чтобы сохранить некоторые константы при генерации размеров лица и позы. Здесь особые точки - это точки, которые не могут претерпеть большие искажения. Поскольку челюсть, рот и даже нос могут искажать форму и изменяться в зависимости от контекста, хорошим вариантом является использование ширины головы (точки 9 и 16) с центральной точкой на носе между глазами (точка 27) в качестве константы при создании контуров лица. Данные точки и пример построения ориентиров лица изображены на рисунке 7.

## 5 Архитектура нейронной сети

На рисунке 6 изображена архитектура нейронной сети.

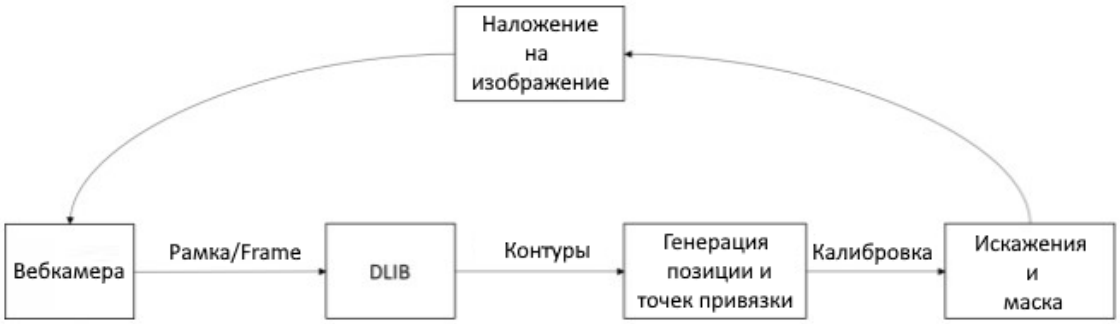


Рисунок 6 – Архитектура нейронной сети

Любое плоское наложение маски на изображение можно разделить на следующие шаги:

1. Создание представления лица с точки зрения контуров лица

2. Используя постоянные точки вокруг глаз, создание линии от одной стороны лица к другой

3. Расчет наклона этой линии и смещения от центральной точки между бровями

4. Применение этого искажения к наложению и преобразование в положение элемента

5. В случае усов создание соответствующего смещения от точки привязки в новое положение на грани для наложения на новую привязку

## 6 Обучение

Как упоминалось выше для обучения использовался набор данных HELEN в библиотеке DLIB. На рисунке 7 представлен пример изображений из обучающей выборки с предобработкой в виде построения контуров лица и поиска особых точек.

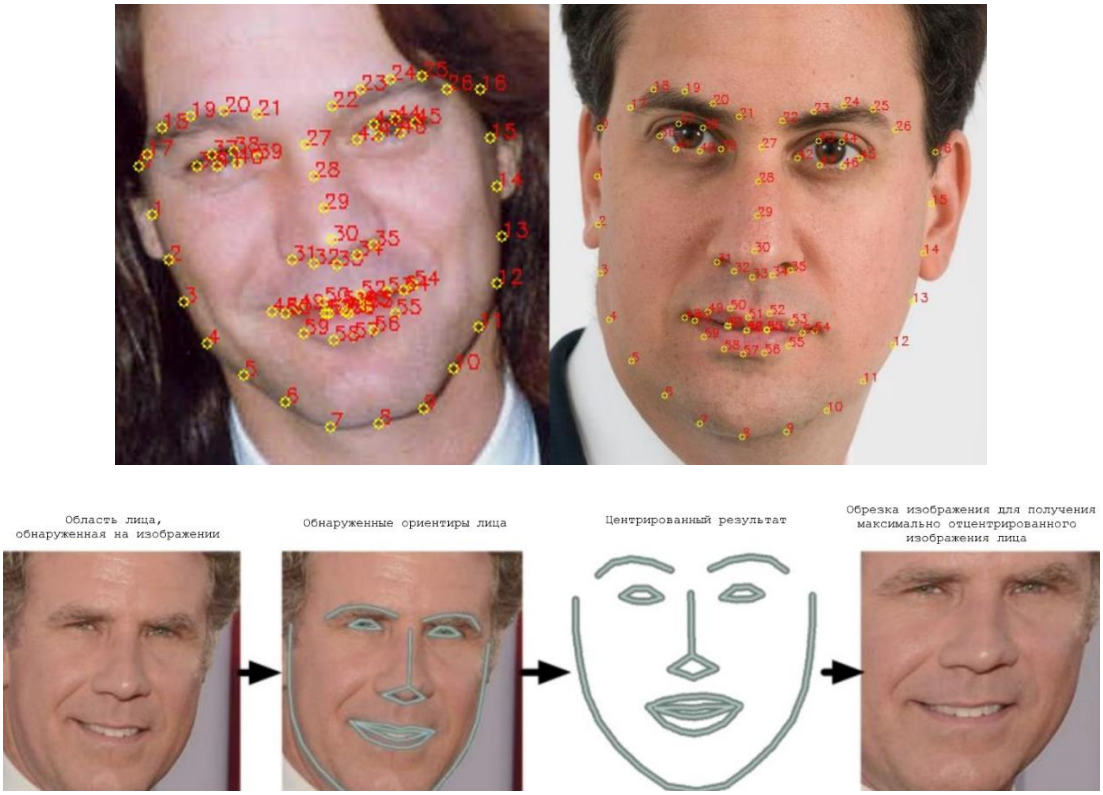


Рисунок 7 - Пример изображений из обучающей выборки с предобработкой в виде построения контуров лица и поиска особых точек

Полученная модель представлена на рисунке 8.

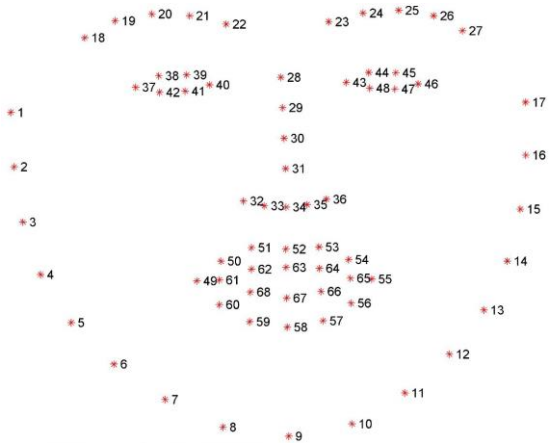

Рисунок 8 – Полученная модель

## 7 Реализация

Исходный код приложения, рекомендации по установке и настройке доступны по следующей ссылке: *https://github.com/alexshapran-as/Snapchat-Filter*.

В листинге 1 представлена функция *main()*, в которой согласно аргументам вызывается та или иная функция для наложения маски: очки или усы.

Листинг 1

```
def main():
    argparser = argparse.ArgumentParser()
    argparser.add_argument("--filter", type=str, default="glasses")
    argparser.add_argument("--footage", type=str, default=None)
    argparser.add_argument("--show-bounds", action="store_true")
    argparser.add_argument("--video-source", type=int, default=0, help="Video input
device number")

    args = argparser.parse_args()

    cam = create_capture(args.video_source)
    should_show_bounds = False

    footage = cv2.imread(args.footage, -1) # args.footage


    try:
        while True:
            with TimeProfiler(args.filter):
                if "glasses" == args.filter:
                    glasses_filter(cam, footage, args.show_bounds)
                elif "moustache" == args.filter:
                    moustache_filter(cam, footage, args.show_bounds)

            if 0xFF & cv2.waitKey(30) == 27:
                break

    except KeyboardInterrupt:
        pass

    cv2.destroyAllWindows()
```

В листинге 2 представлена функция *moustache_filter()*, выполняющая наложение маски «усы» на изображения, взятые с видео потока веб-камеры.

Листинг 2

```
def moustache_filter(cam, moustache, should_show_bounds=False):
    face = get_cam_frame(cam)
    landmarks = get_landmarks(face)
    # moustache.shape = (height, width, rgba channels)
    pts1 = np.float32([[0, 0], [moustache.shape[1], 0], [0, moustache.shape[0]],
[moustache.shape[1], moustache.shape[0]]])

    """
    MOUSTACHE ANCHOR POINTS
    centre anchor point is midway between 34 (top of philtrum) and 54 (bottom of
philtrum)
    width can be determined by the eyes as the mouth can move
    height also determined by the eyes as before
    generate as before and just modify multiplier coefficients & translate to anchor
point?
    ^^^ mouth and jaw can move, use eyes as anchor point initially then translate to
philtrum position
    """

    if type(landmarks) is not int:
        left_face_extreme = [landmarks[0, 0], landmarks[0, 1]]
        right_face_extreme = [landmarks[16, 0], landmarks[16, 1]]
        x_diff_face = right_face_extreme[0] - left_face_extreme[0]
        y_diff_face = right_face_extreme[1] - left_face_extreme[1]

        face_angle = math.degrees(math.atan2(y_diff_face, x_diff_face))

        # get hypotenuse
        face_width = math.sqrt((right_face_extreme[0] - left_face_extreme[0]) ** 2 +
                               (right_face_extreme[1] - right_face_extreme[1]) ** 2)
        moustache_width = face_width * 0.8

        # top and bottom of left eye
        eye_height = math.sqrt((landmarks[19, 0] - landmarks[28, 0]) ** 2 +
                               (landmarks[19, 1] - landmarks[28, 1]) ** 2)
        glasses_height = eye_height * 0.8

        # generate bounding box from the anchor points
        brow_anchor = [landmarks[27, 0], landmarks[27, 1]]
        tl = [int(brow_anchor[0] - (moustache_width / 2)), int(brow_anchor[1] -
(glasses_height / 2))]
        rot_tl = get_rotated_points(tl, brow_anchor, face_angle)

        tr = [int(brow_anchor[0] + (moustache_width / 2)), int(brow_anchor[1] -
(glasses_height / 2))]
        rot_tr = get_rotated_points(tr, brow_anchor, face_angle)

        bl = [int(brow_anchor[0] - (moustache_width / 2)), int(brow_anchor[1] +
(glasses_height / 2))]
        rot_bl = get_rotated_points(bl, brow_anchor, face_angle)

        br = [int(brow_anchor[0] + (moustache_width / 2)), int(brow_anchor[1] +
(glasses_height / 2))]
        rot_br = get_rotated_points(br, brow_anchor, face_angle)

        # locate new location for moustache on philtrum
        top_philtrum_point = [landmarks[33, 0], landmarks[33, 1]]
        bottom_philtrum_point = [landmarks[51, 0], landmarks[51, 1]]
        philtrum_anchor = [(top_philtrum_point[0] + bottom_philtrum_point[0]) / 2,
                           (top_philtrum_point[1] + bottom_philtrum_point[1]) / 2]

        # determine distance from old origin to new origin and translate
        anchor_distance = [int(philtrum_anchor[0] - brow_anchor[0]),
int(philtrum_anchor[1] - brow_anchor[1])]
```

```
        rot_tl[0] += anchor_distance[0]
        rot_tl[1] += anchor_distance[1]
        rot_tr[0] += anchor_distance[0]
        rot_tr[1] += anchor_distance[1]
        rot_bl[0] += anchor_distance[0]
        rot_bl[1] += anchor_distance[1]
        rot_br[0] += anchor_distance[0]
        rot_br[1] += anchor_distance[1]

        pts = np.float32([rot_tl, rot_tr, rot_bl, rot_br])
        m = cv2.getPerspectiveTransform(pts1, pts)

        rotated = cv2.warpPerspective(moustache, m, (face.shape[1], face.shape[0]))
        result_2 = blend_w_transparency(face, rotated)

        # annotate_landmarks(result_2, landmarks)

        if should_show_bounds:
            for p in pts:
                pos = (p[0], p[1])
                cv2.circle(result_2, pos, 2, (0, 0, 255), 2)
                cv2.putText(result_2, str(p), pos,
fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.4, color=(255, 0, 0))

        cv2.imshow("Moustache Filter", result_2)
```

В листинге 3 представлена функция *glasses_filter()*, выполняющая наложение маски «очки» на изображения, взятые с видео потока веб-камеры.

Листинг 3

```
def glasses_filter(cam, glasses, should_show_bounds=False):
    with TimeProfiler("image capture"):
        face = get_cam_frame(cam)

    with TimeProfiler("face pose prediction"):
        landmarks = get_landmarks(face)

    # glasses.shape = (height, width, rgba channels)
    pts1 = np.float32([[0, 0], [glasses.shape[1], 0], [0, glasses.shape[0]],
[glasses.shape[1], glasses.shape[0]]])

    if type(landmarks) is int:
        return

    with TimeProfiler("transformation"):
        """
        GLASSES ANCHOR POINTS:
        17 & 26 edges of left eye and right eye (left and right extrema)
        0 & 16 edges of face across eyes (other left and right extra, interpolate
between 0 & 17, 16 & 26 for half way points)
        19 & 24 top of left and right brows (top extreme)
        27 is centre of the eyes on the nose (centre of glasses)
        28 is the bottom threshold of glasses (perhaps interpolate between 27 & 28
if too low) (bottom extreme)
        """

        left_face_extreme = [landmarks[0, 0], landmarks[0, 1]]
        right_face_extreme = [landmarks[16, 0], landmarks[16, 1]]
        x_diff_face = right_face_extreme[0] - left_face_extreme[0]
        y_diff_face = right_face_extreme[1] - left_face_extreme[1]

        face_angle = math.degrees(math.atan2(y_diff_face, x_diff_face))

        # get hypotenuse
        face_width = math.sqrt((right_face_extreme[0] - left_face_extreme[0]) ** 2 +
                               (right_face_extreme[1] - right_face_extreme[1]) ** 2)
```

```
        glasses_width = face_width * 1.0

        # top and bottom of left eye
        eye_height = math.sqrt((landmarks[19, 0] - landmarks[28, 0]) ** 2 +
                               (landmarks[19, 1] - landmarks[28, 1]) ** 2)
        glasses_height = eye_height * 1.2

        # generate bounding box from the anchor points
        anchor_point = [landmarks[27, 0], landmarks[27, 1]]
        tl = [int(anchor_point[0] - (glasses_width / 2)), int(anchor_point[1] -
(glasses_height / 2))]
        rot_tl = get_rotated_points(tl, anchor_point, face_angle)

        tr = [int(anchor_point[0] + (glasses_width / 2)), int(anchor_point[1] -
(glasses_height / 2))]
        rot_tr = get_rotated_points(tr, anchor_point, face_angle)

        bl = [int(anchor_point[0] - (glasses_width / 2)), int(anchor_point[1] +
(glasses_height / 2))]
        rot_bl = get_rotated_points(bl, anchor_point, face_angle)

        br = [int(anchor_point[0] + (glasses_width / 2)), int(anchor_point[1] +
(glasses_height / 2))]
        rot_br = get_rotated_points(br, anchor_point, face_angle)

        pts = np.float32([rot_tl, rot_tr, rot_bl, rot_br])
        m = cv2.getPerspectiveTransform(pts1, pts)

        rotated = cv2.warpPerspective(glasses, m, (face.shape[1], face.shape[0]))
        result_2 = blend_w_transparency(face, rotated)

        if should_show_bounds:
            for p in pts:
                pos = (p[0], p[1])
                cv2.circle(result_2, pos, 2, (0, 0, 255), 2)
                cv2.putText(result_2, str(p), pos,
fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.4, color=(255, 0, 0))

        cv2.imshow("Glasses Filter", result_2)
```

На рисунке 9 представлен пример наложения маски «усы».


Рисунок 9 – Пример наложения маски «усы»

На рисунке 10 представлены примеры наложения маски «очки».

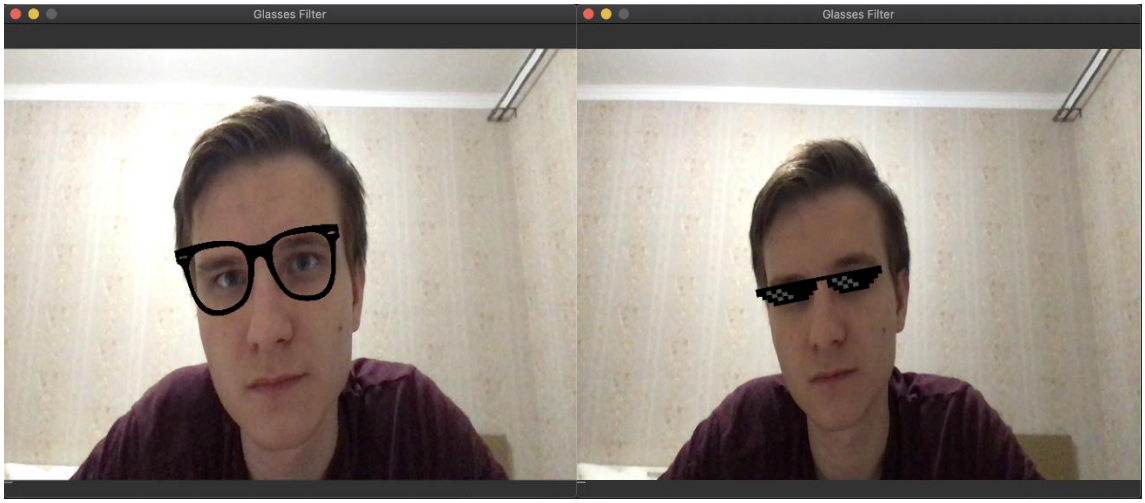


Рисунок 10 – Примеры наложения маски «очки»

## 8 Выводы

В данной работе были исследованы различные методы машинного обучения для обработки изображений и изучены библиотеки DLIB и OpenCV.

Была построена нейронная сеть для выделения лица на изображениях, полученных из кадров видео с видео потока веб-камеры, привязки особых точек к лицу и наложения двух различных масок – «усы» и «очки».

Результаты реализации показывают, что цель работы была достигнута: маски накладываются в нужное место на лице даже при искажениях, наклонах или поворотах.

## 9 Список литературы

1. Документация OpenCV, 2013. // Camera Calibration and 3D Reconstruction [Электронный ресурс]. – URL: https://docs.opencv.org/2.4.8/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html (дата обращения 25.05.2020).

2. Viola, P. & Jones, M., 2001. Rapid object detection using a boosted cascade of simple features, Seattle: IEEE [Электронный ресурс]. – URL: https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf (дата обращения 27.05.2020).

3. Papageorgiou, C. P., Oren, M. & Poggio, T., 1998. A General Framework for Object Detection, Cambridge: IEEE [Электронный ресурс]. – URL: https://www.researchgate.net/publication/3766402_General_framework_for_object_detection (дата обращения 30.05.2020).

4. Документация библиотеки DLIB C++ Library [Электронный ресурс]. – URL: http://dlib.net/ (дата обращения 33.05.2020).

5. Le, V. et al., 2012. Interactive Facial Feature Localization, Urbana: s.n [Электронный ресурс]. – URL: http://www.ifp.illinois.edu/~vuongle2/helen/eccv2012_helen_final.pdf (дата обращения 33.05.2020).