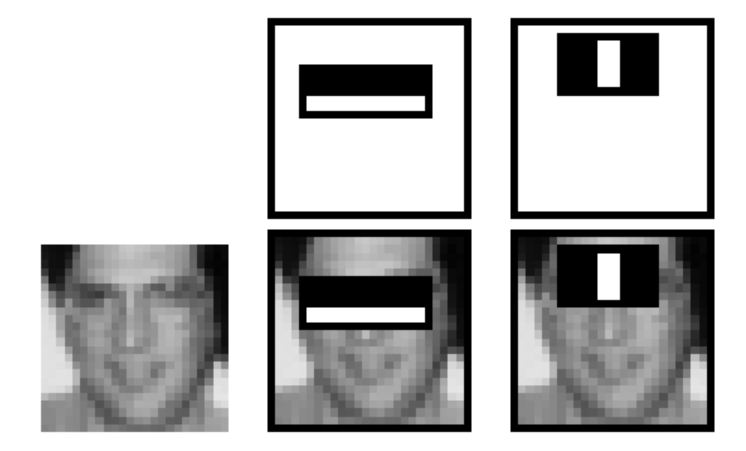
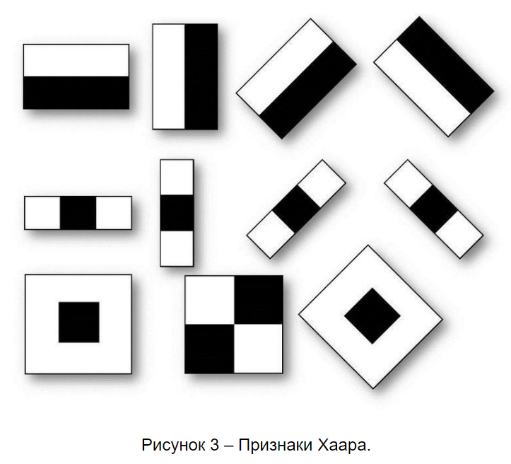
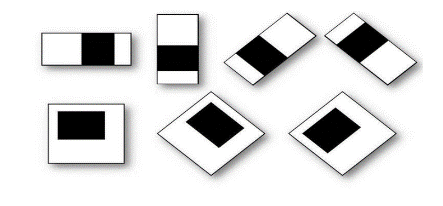
Рассмотрим некоторые существующие детекторы лиц:

* **Haar Cascade Face Detector in OpenCV**

Признаки Хаара — признаки цифрового изображения, используемые в распознавании образов. Признак Хаара состоит из смежных прямоугольных областей. Они позиционируются на изображении, далее суммируются интенсивности пикселей в областях, после чего вычисляется разность между суммами. Эта разность и будет значением определенного признака, определенного размера, определенным образом спозиционированного на изображении. Для всех изображений человеческих лиц общим является то, что область в районе глаз темнее, чем область в районе щек. Следовательно, общим признаком Хаара для лиц является 2 смежных прямоугольных региона, лежащих на глазах и щеках.



Пример вызова:

faceCascade = cv2.CascadeClassifier('./haarcascade\_frontalface\_default.xml')

faces = faceCascade.detectMultiScale(frameGray)

for face in faces:

    x1, y1, w, h = face

    x2 = x1 + w

    y2 = y1 + h

Приведенный выше фрагмент кода загружает файл модели Haar Cascade и применяет его к изображению в градациях серого. Выводом является список, содержащий обнаруженные лица. Каждый элемент списка снова является списком с 4 элементами, указывающими (x, y) координаты верхнего левого угла, а также ширину и высоту обнаруженного лица.

Преимущества:

- Работает практически в режиме реального времени на процессоре

- Простая архитектура

- Обнаруживает лица в разных масштабах

Недостатки:

- Основным недостатком этого метода является то, что он дает много ложных предсказаний.

- Не работает на нефронтальных изображениях

- Не работает при окклюзии (Окклюзия — это зрительное восприятие объекта, который находится перед другим объектом или позади него, что даёт информацию о порядке уровней текстуры)

* **DNN Face Detector in OpenCV**

Эта модель была включена в OpenCV с версии 3.3. Он основан на детекторе Single-Shot-Multibox и использует архитектуру ResNet-10 в качестве магистрали. Модель была обучена с использованием изображений, доступных в Интернете, но источник не разглашается.

Метод Single Shot Multibox Detector (SSD) был опубликован сравнительно недавно. Термин SSD используется для описания архитектур, в которых используется одна сверточная нейронная сеть (feedforward convolutional network) для непосредственного предсказания расположения областей и их классов, без применения второго этапа классификации. В этом методе на выходе нейронной сети формируются несколько тысяч прогнозов для возможных регионов расположения объектов разной формы на разных масштабах, затем с помощью подавления немаксимумов (Non-Maximum Suppression) происходит выбор нескольких наиболее вероятных областей.

OpenCV предоставляет 2 модели для этого детектора лица:

- Версия с плавающей запятой 16 оригинальной реализации caffe (5,4 МБ)

DNN = "TF"

if DNN == "CAFFE":

    modelFile = "res10\_300x300\_ssd\_iter\_140000\_fp16.caffemodel"

    configFile = "deploy.prototxt"

    net = cv2.dnn.readNetFromCaffe(configFile, modelFile)

else:

    modelFile = "opencv\_face\_detector\_uint8.pb"

    configFile = "opencv\_face\_detector.pbtxt"

    net = cv2.dnn.readNetFromTensorflow(modelFile, configFile)

- 8-битная квантованная версия с использованием Tensorflow (2,7 МБ)

lob = cv2.dnn.blobFromImage(frameOpencvDnn, 1.0, (300, 300), [104, 117, 123], False, False)

net.setInput(blob)

detections = net.forward()

bboxes = []

for i in range(detections.shape[2]):

    confidence = detections[0, 0, i, 2]

    if confidence > conf\_threshold:

        x1 = int(detections[0, 0, i, 3] \* frameWidth)

        y1 = int(detections[0, 0, i, 4] \* frameHeight)

        x2 = int(detections[0, 0, i, 5] \* frameWidth)

        y2 = int(detections[0, 0, i, 6] \* frameHeight)

В приведенном выше коде изображение преобразуется в большой двоичный объект и передается по сети с помощью функции forward (). Выход обнаружений представляет собой 4-D матрицу, где:

- The 3rd dimension соответсвует обнаруженным лицам. (i - итератор по количеству граней)

- The fourth dimension содержат информацию о ограничительной рамке и счет для каждого лица. Например, обнаружения [0,0,0,2] дают оценку достоверности для первого лица, а обнаружения [0,0,0,3:6] дают ограничивающую рамку.

Выходные координаты ограничительной рамки нормированы между [0,1]. Таким образом, координаты должны быть умножены на высоту и ширину исходного изображения, чтобы получить правильную ограничивающую рамку на изображении.

Преимущества:

* Наиболее точный из четырех методов

- Работает в режиме реального времени на процессоре

- Работает для различных ориентаций лица - вверх, вниз, влево, вправо, на лицевой стороне и т. д.

- Работает даже при существенной окклюзии

- Обнаруживает лица в разных масштабах (обнаруживает как большие, так и крошечные лица)

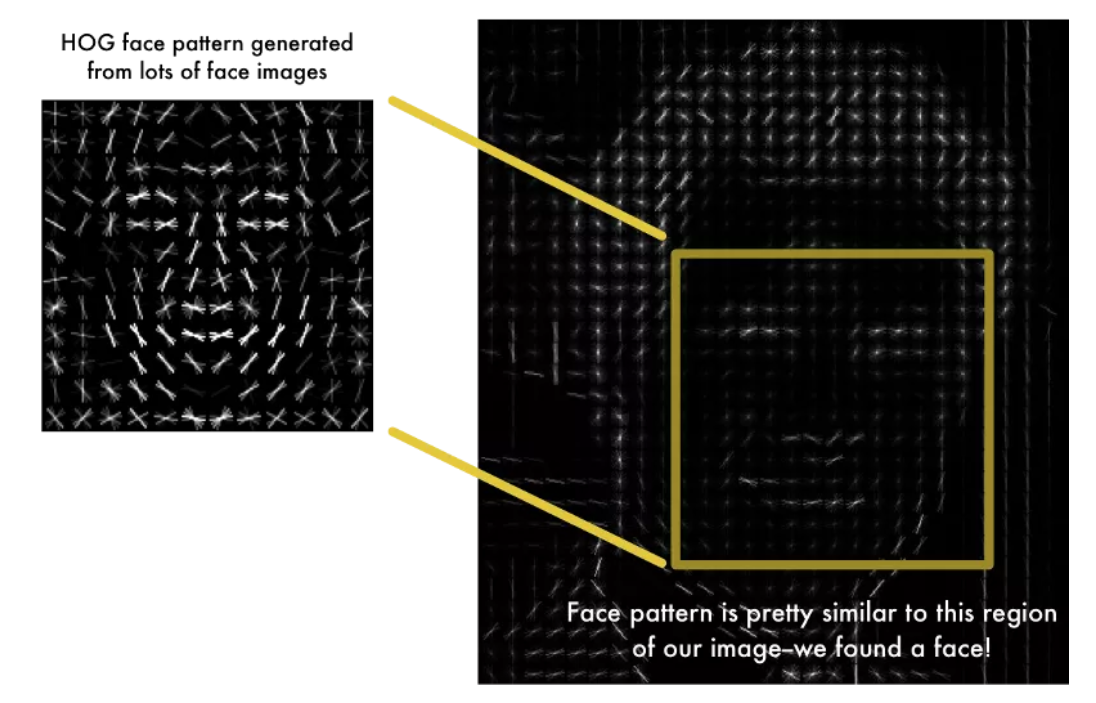
Детектор на основе DNN преодолевает все недостатки детектора на основе Haar Cascade без ущерба для любых преимуществ, предоставляемых Haar. Мы не могли видеть никакого существенного недостатка для этого метода, за исключением того, что он медленнее, чем детектор лица на основе Dlib HoG, который обсуждается далее.

* **HoG Face Detector in Dlib**

Это широко используемая модель обнаружения лиц, основанная на функциях HoG и SVM. Модель построена из 5 фильтров HOG - вид спереди, вид слева, вид справа, вид спереди, но повернутый влево, и вид спереди, но повернутый вправо. Модель встроена в сам заголовочный файл.

Набор данных, используемый для обучения, состоит из 2825 изображений, которые получены из набора данных LFW и аннотированы вручную Дэвисом Кингом, автором Dlib.

Общая схема вычисления HOG выглядит следующим образом. Картинка разделяется на части (ячейки), для каждой ячейки строим гистограмму направлений градиента яркости, далее гистограммы ячеек нормируются по контрасту и объединяются. Чтобы найти лицо на HOG-изображении, все, что нужно сделать, – это найти часть изображения, которая наиболее похожа на известный рисунок HOG, полученный из множества других лиц в ходе обучения:



hogFaceDetector = dlib.get\_frontal\_face\_detector()

faceRects = hogFaceDetector(frameDlibHogSmall, 0)

for faceRect in faceRects:

    x1 = faceRect.left()

    y1 = faceRect.top()

    x2 = faceRect.right()

    y2 = faceRect.bottom()

В приведенном выше коде сначала загружается детектор лица. Затем передается ему изображение через детектор. Второй аргумент - это количество раз, когда мы хотим увеличить масштаб изображение. Чем выше масштаб, тем больше шансов обнаружить меньшие лица. Однако масштабирование изображения будет иметь существенное влияние на скорость вычислений. Выходные данные представлены в виде списка граней с координатами (x, y) диагональных углов.

Преимущества:

- Самый быстрый метод на процессоре

- Работает очень хорошо для фронтальных и слегка не фронтальных граней

- Облегченная модель по сравнению с остальными тремя.

- Работает при небольшой окклюзии

Недостатки:

- Основным недостатком является то, что он не обнаруживает маленькие лица, поскольку он обучен для минимального размера лица 80 × 80. Тем не менее, можно обучить свой собственный детектор лица для лиц меньшего размера.

- Ограничительная рамка часто исключает часть лба и даже часть подбородка.

- Не очень хорошо работает при значительной окклюзии

- Не работает для боковой грани и экстремальных не лобных граней, таких как взгляд вниз или вверх.

* **CNN Face Detector in Dlib**

В этом методе используется Maximum-Margin Object Detector (MMOD) с функциями на основе CNN.

Max-Margin Object Detection улучшает обнаружение объектов путем разработки оптимизатора производительности системы обнаружения объектов с точки зрения количества пропущенных обнаружений и ложных срабатываний в конечном выводе системы. Сверточная нейронная сеть (CNN) используется в качестве двоичного классификатора и используется для классификации каждого из подокон.

Он использует набор данных, помеченный его автором Дэвисом Кингом вручную, состоящий из изображений из различных наборов данных, таких как ImageNet, PASCAL VOC, VGG, WIDER, Face Scrub. Он содержит 7220 изображений.

dnnFaceDetector = dlib.cnn\_face\_detection\_model\_v1("./mmod\_human\_face\_detector.dat")

faceRects = dnnFaceDetector(frameDlibHogSmall, 0)

for faceRect in faceRects:

    x1 = faceRect.rect.left()

    y1 = faceRect.rect.top()

    x2 = faceRect.rect.right()

    y2 = faceRect.rect.bottom()

Код похож на детектор HoG за исключением того, что в этом случае мы загружаем модель обнаружения лиц CNN. Кроме того, координаты присутствуют внутри прямоугольного объекта.

Преимущества:

- Работает для разных ориентаций лица

- Устойчив к окклюзии

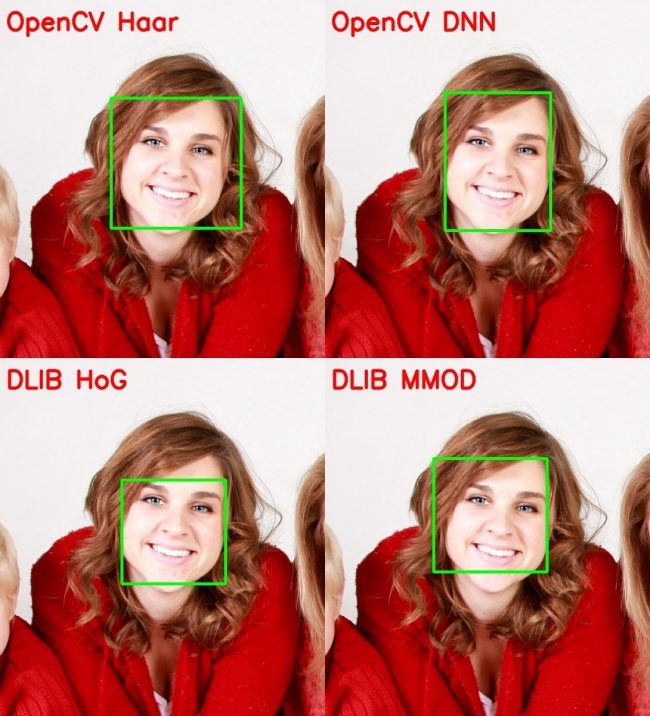
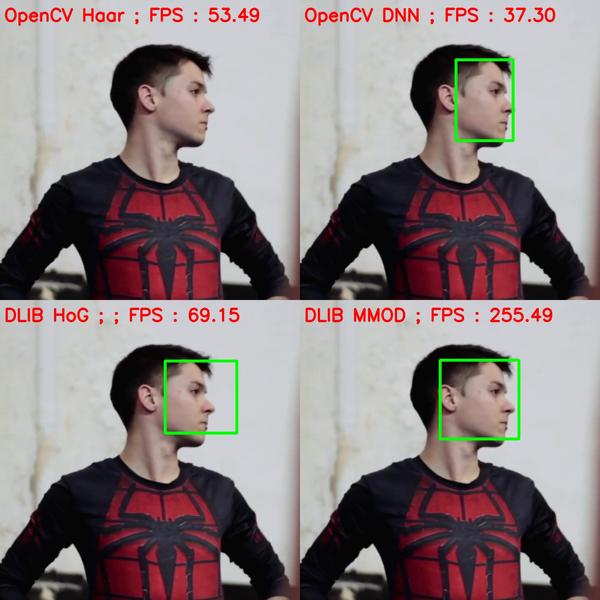
- Очень быстро работает на GPU

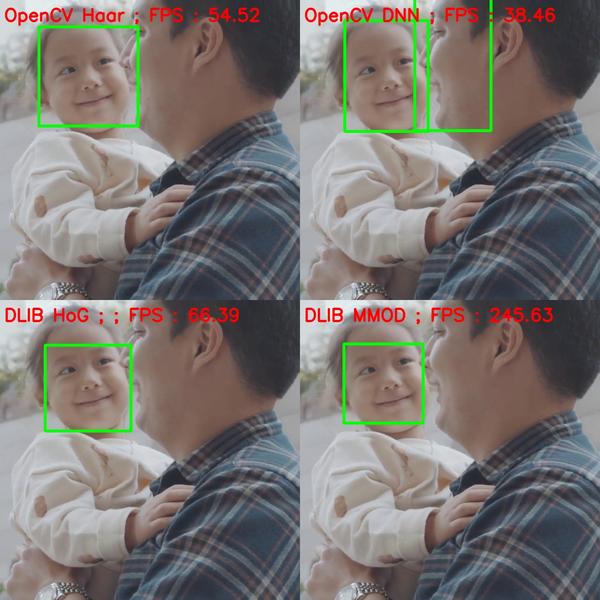
- Очень легкий тренировочный процесс

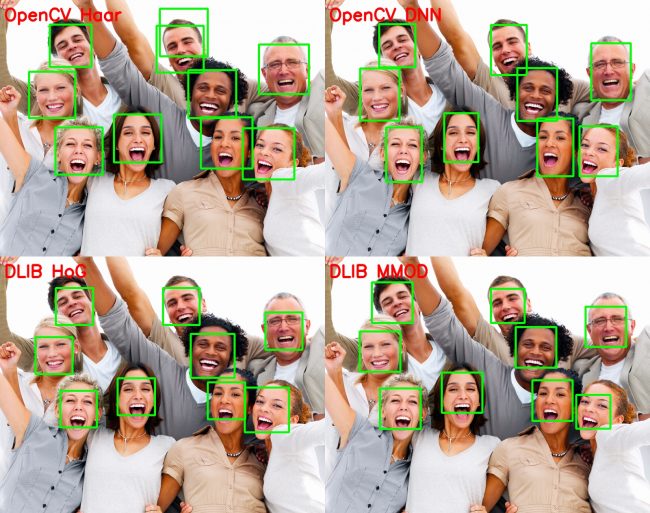
Недостатки:

- Очень медленно на процессоре

- Не обнаруживает маленькие лица, так как тренируется для минимального размера лица 80 × 80. Таким образом, вы должны убедиться, что размер лица должен быть больше, чем в вашем приложении. Тем не менее, вы можете обучить свой собственный детектор лица для лиц меньшего размера.

- Ограничительная рамка даже меньше детектора HoG.





Также стоит уделить внимание и рассмотреть такое понятие как генеративно-состязательная сеть. Это может быть подходом, с помощью которого осуществлять детекцию лиц.

**GAN – генеративно-состязательная нейросеть (Generative adversarial network, GAN) – один из алгоритмов классического машинного обучения, обучения без учителя. Суть идеи в комбинации двух нейросетей, при которой одновременно работает два алгоритма “генератор” и “дискриминатор”. Задача генератора – генерировать образы заданной категории. Задача дискриминатора – пытаться распознать созданный образ.**

Рассмотрим, как работают алгоритмы дикриминатора и генератора.

Дискриминатор

Дискриминационные алгоритмы пытаются классифицировать входные данные. Учитывая особенности полученных данных, они стараются определить категорию, к которой они относятся.

К примеру, пробегая все слова в письме дискриминационный алгоритм может предсказать, является сообщение спамом или не спамом. Спам — это категория, а пакет слов, собранный из электронной почты — образы, которые составляют входные данные. Математически категории обозначают y, а образы обозначают x. Запись p(y|x) используется для обозначения «вероятности y при заданном x», которая обозначает «вероятность того, что электронное письмо является спамом при имеющемся наборе слов».

Итак, дискриминационные функции сопоставляют образы с категорией. Они заняты только этой корреляцией.

Генератор

Генеративные алгоритмы заняты обратным. Вместо того, чтобы предсказывать категорию по имеющимся образам, они пытаются подобрать образы к данной категории.

В то время как дискриминационные алгоритмы волнует взаимосвязь между y и x, генеративные алгоритмы волнует “откуда берутся x”. Они позволяют находить p(x|y), вероятность x при данном y или вероятность образов при данном классе (генеративные алгоритмы также могут использоваться в качестве классификаторов. Они могут делать больше, чем классифицировать входные данные.)

Еще одно представление о работе генеративных алгоритмов можно получить, разделяя дискриминационные модели от генеративных таким образом:

Дискриминационные модели изучают границу между классами;

Генеративные модели моделируют распределение отдельных классов.