|  |  |
| --- | --- |
| **СОГЛАСОВАНО**  Доцент кафедры  ИАНИ ННГУ, к.ф.-м.н.  \_\_\_\_\_\_\_\_\_\_\_\_\_ Д.А. Яшунин  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г. | **УТВЕРЖДАЮ**  Профессор кафедры  ИАНИ ННГУ, д.т.н.  \_\_\_\_\_\_\_\_\_\_\_\_\_ Н.В. Старостин  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г. |

**Пояснительная записка № 1**

**«Исследования существующих методов нахождения и распознавания лиц»**

**Этап 1. Исследование предметной области и разработка способа решения задачи**

**НИР «Разработка и реализация программного обеспечения   
для распознавания лиц на групповых фотографиях»**

**(Шифр ПО «FaceRecognition»)**

Ответственный исполнитель

\_\_\_\_\_\_\_\_\_\_\_\_ A.C. Филатова

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г.

**Н. Новгород 2019**

**Аннотация**

В пояснительной записке «Исследования существующих методов нахождения и распознавания лиц» в рамках этапа исследования предметной области и разработка способа решения задачи представлены обзоры существующих методов детекции лиц и объектов на изображениях, методов выравнивания лиц, методов распознавания лиц. Также представлены краткие описания инструментов – фреймворков и библиотек – с помощью которых можно решать задачу распознавания лиц.

В заключении обозначены выбранные методы и инструменты для дальнейшей работы.

**Оглавление**

[1 Обзор решений задачи детектирования лиц/объектов на фотографии 4](#_Toc4955471)

[1.1 Haar Cascade Face Detector in OpenCV 4](#_Toc4955472)

[1.2 DNN Face Detector in OpenCV 5](#_Toc4955473)

[1.3 HoG Face Detector in Dlib 6](#_Toc4955474)

[1.4 CNN Face Detector in Dlib 8](#_Toc4955475)

[1.5 R-CNN, Fast R-CNN, Faster R-CNN 9](#_Toc4955476)

[1.6 Single Shot MultiBox Detector (SSD) 10](#_Toc4955477)

[1.7 You Only Look Once (YOLO) 11](#_Toc4955478)

[1.8 OpenVino Objects Detection Models 12](#_Toc4955479)

[2 Обзор решений задачи поворота (выравнивания) лица 12](#_Toc4955480)

[2.1 OpenCv 12](#_Toc4955481)

[2.2 OpenBR 13](#_Toc4955482)

[2.3 OpenFace 13](#_Toc4955483)

[3 Обзор методов решений задачи распознавания лиц 13](#_Toc4955484)

[3.1 FaceNet 13](#_Toc4955485)

[3.2 InsightFace 15](#_Toc4955486)

[3.3 SphereFace 15](#_Toc4955487)

[4 Обзор фреймфорков и технологий, реализующих выбранное решение задачи распознавания лиц 16](#_Toc4955488)

[4.1 Torch 16](#_Toc4955489)

[4.2 PyTorch 16](#_Toc4955490)

[4.3 Theano 17](#_Toc4955491)

[4.4 Keras 18](#_Toc4955492)

[4.5 Deeplearning4j 19](#_Toc4955493)

[4.6 OpenVINO 19](#_Toc4955494)

[4.7 OpenCV 19](#_Toc4955495)

[4.8 TensorFlow 20](#_Toc4955496)

[Заключение 21](#_Toc4955497)

[Список используемых источников 22](#_Toc4955498)

# Обзор решений задачи детектирования лиц/объектов на фотографии

## Haar Cascade Face Detector in OpenCV

Признаки Хаара [1] — признаки цифрового изображения, используемые в распознавании образов. Признак Хаара состоит из смежных прямоугольных областей. 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.

Маски накладываются на разные части кадра, и с помощью этого наложения программа определяет, есть ли лицо. Наложение маски на определенную часть кадра дает числовое значение - результат свертки маски с кадром: программа складывает яркость всех пикселей изображения, попавших при наложении в белую часть маски, а также яркость всех пикселей, попавших в черную часть маски, затем вычисляет разность этих значений. Результат свертки сравнивается с пороговой величиной. Если это значение выше пороговой величины, то в кадре присутствует лицо. Наряду с положительной обучающей выборкой, содержащей изображения лиц, создается отрицательная выборка, на фотографиях которой лиц нет. На этой выборке тоже проходит обучение классификатора, но в этом случае свертка должна выдавать значение меньше пороговой величины. Если на какой-либо фотографии выборки одна маска ошиблась, то вес (важность) этой фотографии для других масок увеличивается.

По результатам свертки с положительной и отрицательной выборкой маска попадает в каскад классификатора с некоторым коэффициентом, учитывающим погрешность детектирования маской лиц и вес фотографий выборки, на которых маска не ошиблась. Модуль обнаружения лиц сравнивает сумму сверток всех масок каскада классификатора с учетом их коэффициентов погрешности с пороговой величиной. Если сумма больше порога, детектор выдает сигнал о присутствии лица в кадре.

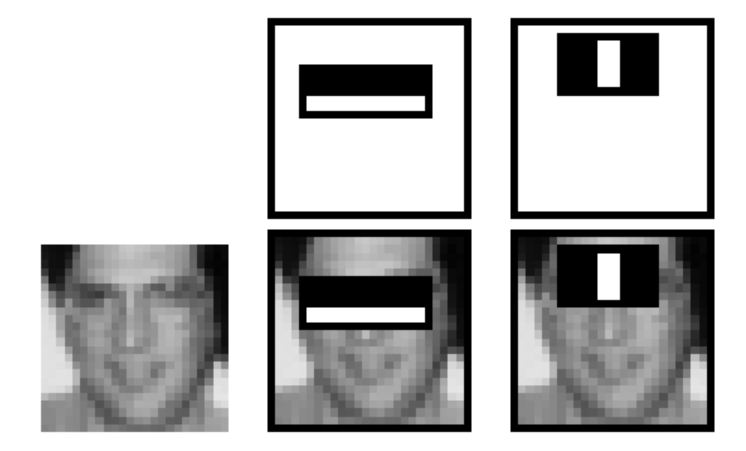
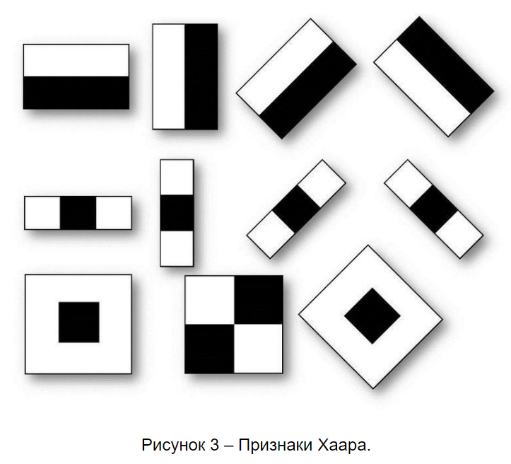
Чаще всего выборка содержит кадры с изображениями лиц анфас, поэтому и детектируются лучше всего лица при таком расположении. Однако классификатор можно обучить детектировать лица в разных положениях, создавая соответствующие выборки.

Рис. 1 Признаки Хаара

Рис. 2 Признаки Хаара при наложении на лицо

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 [2] 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 [3]. Модель была обучена с использованием изображений, доступных в Интернете, но источник не разглашается.

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. Он медленнее, чем детектор лица на основе Dlib HoG, который обсуждается далее.

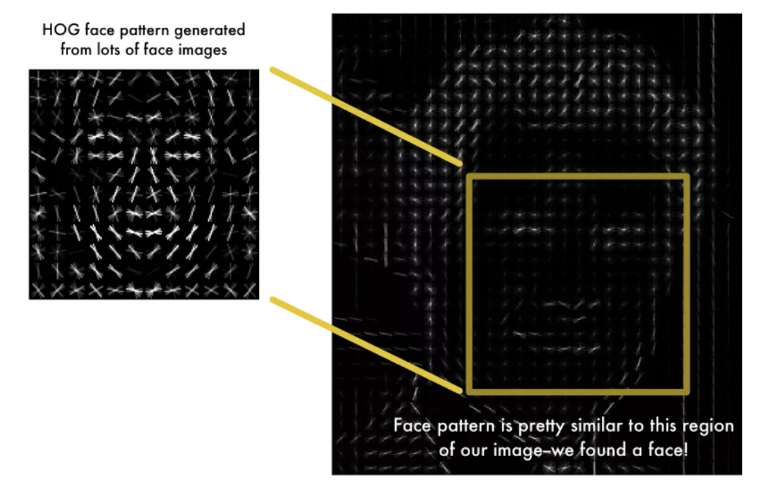
## HoG Face Detector in Dlib

Это широко используемая модель обнаружения лиц, основанная на функциях HoG и SVM [5, 6]. 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 строится гистограмма 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.

Для каждого отдельного пикселя необходимо посмотреть на пиксели, которые непосредственно его окружают: цель в том, насколько темный текущий пиксель по сравнению с окружающими пикселями. Затем рисуется стрелка, показывающая, в каком направлении изображение становится темнее. Этот процесс повторяется для каждого пикселя изображения. Каждый пиксель заменяется стрелкой. Эти стрелки называются градиентами. Градиенты показывают поток света по всему изображению:



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()

Рис. 3 HOG-шаблон и HOG-изображение

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) [7] 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.

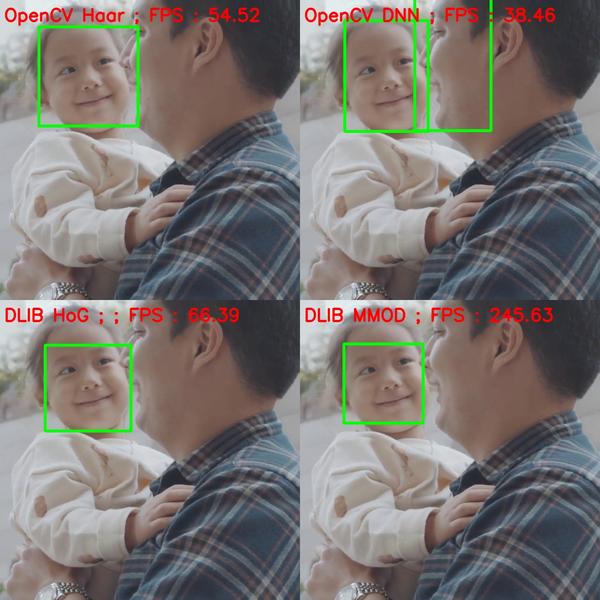
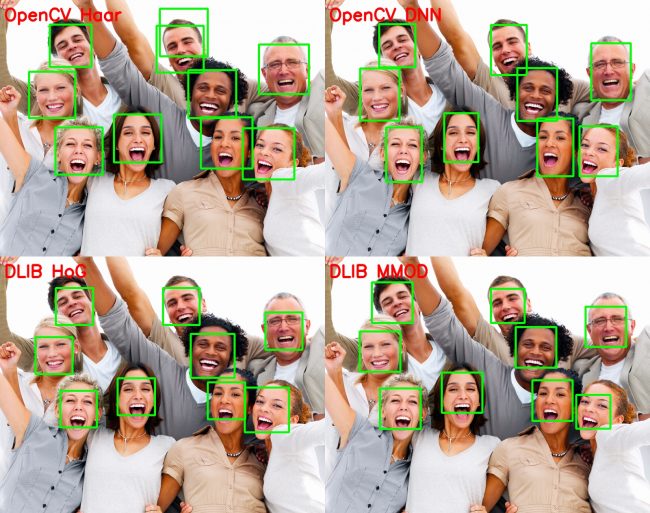
- Ограничительная рамка даже меньше детектора HoG. Ниже (рис. 4, 5) приведены примеры сравнения результатов применения некоторых их вышеописанных методов:

Рис. 4, 5 Примеры результатов детекции лиц методами OpenCV Haar, OpenCV DNN, DLIB HOG, DLIB MMOD

## R-CNN, Fast R-CNN, Faster R-CNN

**R-CNN** [12] создает ограничивающие рамки или предложения регионов, используя процесс, называемый Selective Search [13]. На высоком уровне Selective Search просматривает изображение через окна разных размеров и для каждого размера пытается сгруппировать смежные пиксели по текстуре, цвету или интенсивности для идентификации объектов.

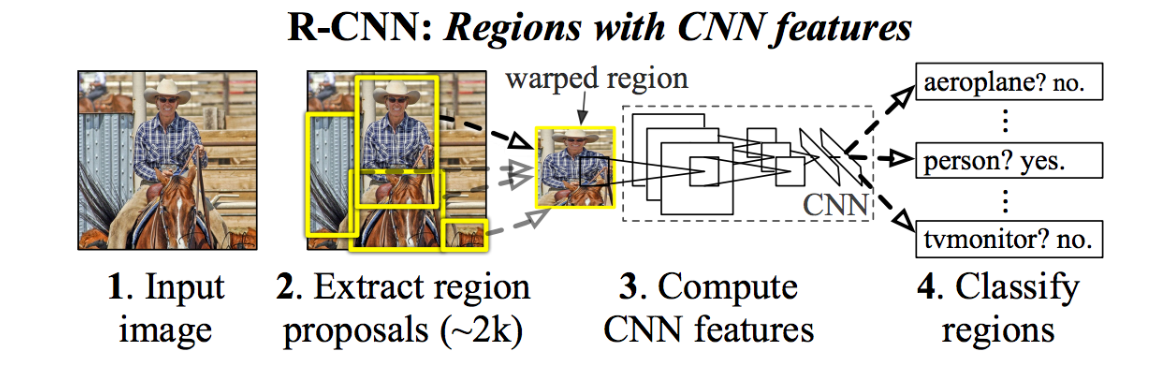


Рис. 6 Структура R-CNN

Создается набор предложений для ограничивающих рамок. Затем запускается изображения в ограничивающих прямоугольниках через предварительно обученную AlexNet и, наконец, SVM, чтобы увидеть, что представляет собой изображение в этом прямоугольнике. Пропустите блок через модель линейной регрессии, чтобы вывести более точные координаты для блока после классификации объекта.

Сеть **Faster R-CNN** [9, 10] состоит из двух модулей. Первый модуль представляет собой глубокую полностью сверточную сеть, которая предлагает регионы (это замена процедуры генерации претендентов selective search в других моделях R-CNN, занимало достаточно много времени), а второй модуль представляет собой **Fast R-CNN** [11] детектор, который использует предложенные регионы. Вся система представляет собой единую унифицированную сеть для обнаружения объектов.

1. Сверточная сетка выдает feature map
2. В рамках RPN по извлечённым CNN признакам скользят «мини-нейросетью» с небольшим (3х3) окном. Полученные с её помощью значения передаются в два параллельных полносвязанных слоя: box-regression layer (reg) и box-classification layer (cls).
3. Первый слой rpn\_cls\_score выдает k пар - вероятности наличия или отсутствия объекта в соответствующем региона

Слой rpn\_bbox\_pred выдает k четверок - поправки для координат центра и размеров соответствующего региона претендента. Авторы называют четвёрку: две координаты центра, масштаб и отношение сторон - анкер (anchor). Выходные данные RPN представляют собой набор предложений, которые будут проанализированы классификатором и регрессором, чтобы в конечном итоге проверить наличие объектов. Т.е. RPN предсказывает возможность анкера быть фоном ил передним планом и уточняет анкер.

## Single Shot MultiBox Detector (SSD)

Обнаружение объектов SSD [14] состоит из 2 частей:

Извлечь Feature maps и применить фильтры свертки, чтобы обнаружить объекты.

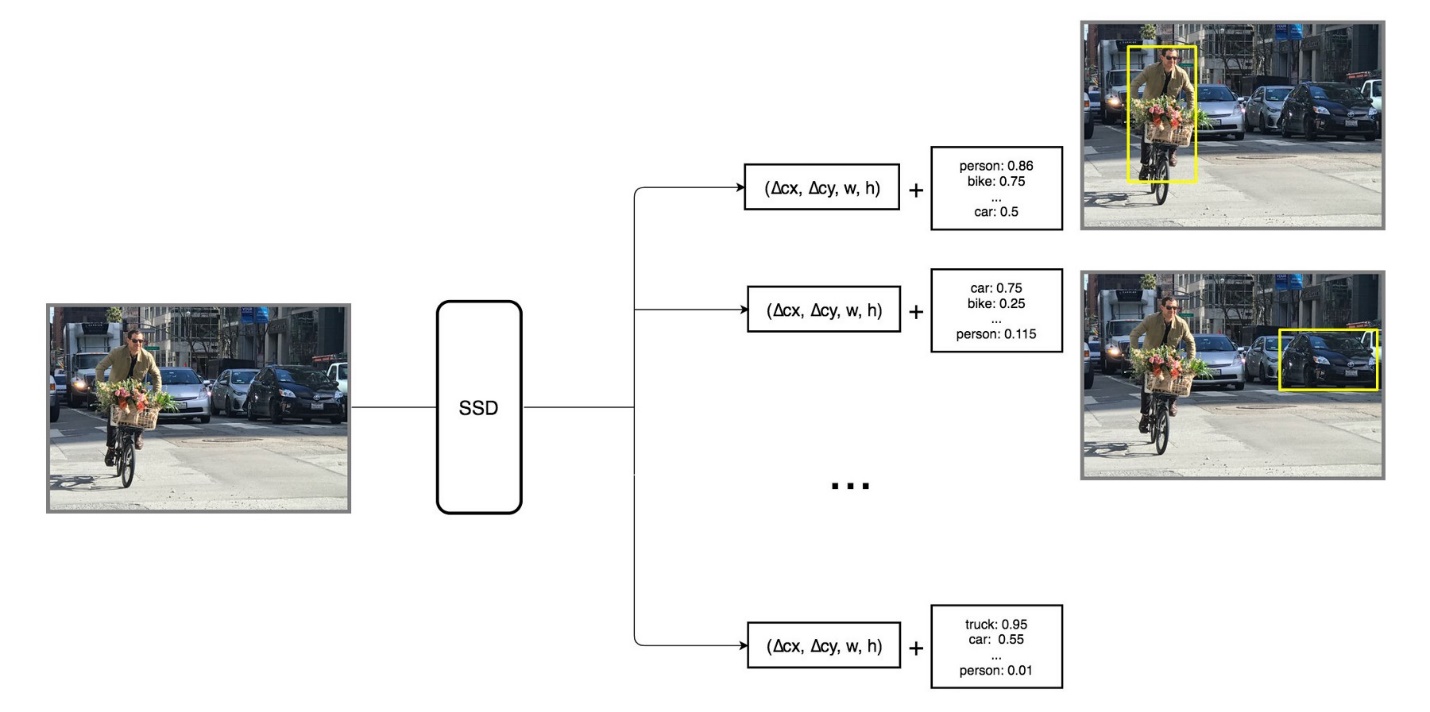


Рис. 7 Детекция с помощью SSD

SSD использует VGG16 для извлечения карт объектов. Затем он обнаруживает объекты, используя слой Conv4\_3. Для иллюстрации мы нарисуем Conv4\_3 пространственно 8 × 8 (38 × 38). Для каждой ячейки (также называемой местоположением) она делает 4 предсказания объекта.

Каждое предсказание состоит из граничного блока и 21 оценки для каждого класса (один дополнительный класс для объекта), и мы выбираем наивысшую оценку в качестве класса для ограниченного объекта. Conv4\_3 делает в общей сложности 38 × 38 × 4 прогнозов: четыре прогноза на ячейку независимо от глубины карт объектов. Как и ожидалось, многие прогнозы не содержат объекта. SSD резервирует класс «0», чтобы указать, что у него нет объектов.

SSD использует предварительно обучению сеть для извлечения карт объектов. Затем он обнаруживает объекты, используя слой Conv4\_3. Для иллюстрации мы нарисуем Conv4\_3 пространственно 8 × 8 (38 × 38). Для каждой ячейки (также называемой местоположением) она делает 4 предсказания объекта.

Каждое предсказание состоит из граничного блока и 21 оценки для каждого класса (один дополнительный класс для объекта), и мы выбираем наивысшую оценку в качестве класса для ограниченного объекта.

## You Only Look Once (YOLO)

YOLO [15, 16] использует совершенно другой подход.

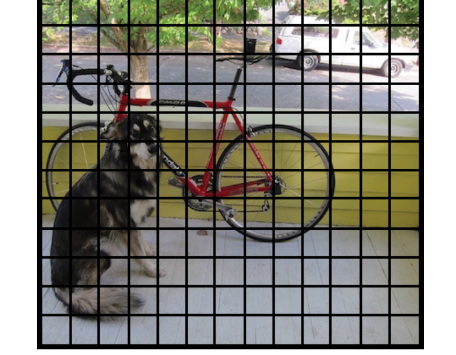
YOLO на самом деле смотрит на изображение только один раз (отсюда и его название: «Вы смотрите только один раз»), но умным способом. YOLO делит изображение на сетку из 13 на 13 ячеек. Каждая из этих ячеек отвечает за предсказание 5 ограничивающих рамок.

Рис. 8 Этапы детекции с помощью YOLO

YOLO также выводит показатель достоверности, который говорит нам, насколько достоверным является то, что предсказанная ограничивающая рамка на самом деле охватывает некоторый объект.Предсказанные ограничивающие рамки могут выглядеть примерно следующим образом (чем выше показатель достоверности, тем толще будет нарисовано поле). Для каждого ограничивающего прямоугольника ячейка также предсказывает класс. Это работает так же, как классификатор: оно дает распределение вероятностей по всем возможным классам.

Оценка достоверности для ограничивающего прямоугольника и прогнозирования класса объединяются в один окончательный показатель, который говорит нам о вероятности того, что этот ограничивающий прямоугольник содержит объект определенного типа.

Поскольку существует 13 × 13 = 169 ячеек сетки, и каждая ячейка предсказывает 5 ограничивающих рамок, в итоге мы получаем 845 ограничивающих рамок.

Оказывается, что большинство из этих блоков будут иметь очень низкие показатели достоверности, поэтому мы сохраняем только поля, окончательный результат которых составляет 30% или более (вы можете изменить этот порог в зависимости от того, насколько точным вы хотите, чтобы детектор был).

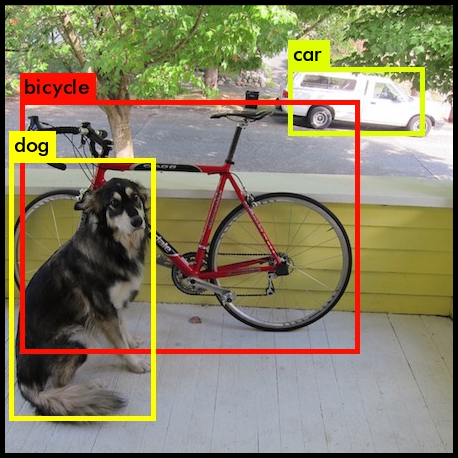
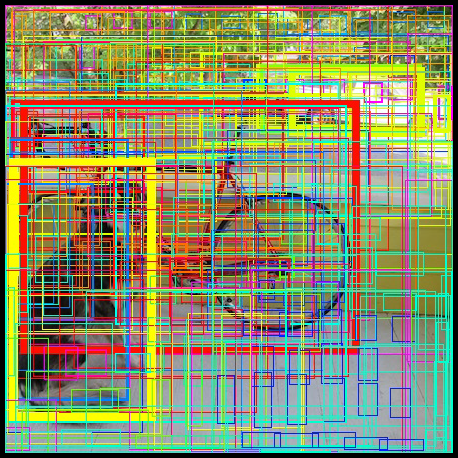
Из 845 общих ограничительных рамок мы оставили только эти три, потому что они дали лучшие результаты. Но обратите внимание, что, хотя было 845 отдельных прогнозов, все они были сделаны одновременно - нейронная сеть просто запускалась один раз. И именно поэтому YOLO такой мощный и быстрый.

Рис. 9 Результаты детекции объектов с помощью YOLO

## OpenVino Objects Detection Models

OpenVino предлагает несколько вариантов сеток, которые можно использовать для детекции лиц.

Примеры таких сеток можно увидеть в [17].

# Обзор решений задачи поворота (выравнивания) лица

## OpenCv

OpenCv [18] является самой мощной библиотекой компьютерного зрения среди BR и Face. OpenCv не только связан с распознаванием изображений, он может быть использован для создания других интересных вещей, связанных с компьютерным зрением. Имеется возможность построить ряд проектов, используя OpenCV, начиная от применения фильтра к фотографиям, OCR, обнаружения объектов из livestream видео кадры и т.д. Также можно создавать модули распознавания лиц, обучая свои собственные наборы данных. Но OpenCv не полностью совместим с Nueral networks. Содержит более 2000 алгоритмов.

## OpenBR

OpenBr [19] -это система биометрии. Как предполагает официальный сайт, его можно использовать для оценки возраста, сравнения лиц и выявления пола. Так что это ограничивается только распознаванием лица в целом. Даже OpenCv имеет возможность сделать это. Но OpenCv требует много работы, чтобы сделать, чтобы построить передовые модели. OpenBr предоставляет прямое решение для распознавания лиц. OpenBR доступен только для нескольких языков, таких как C и c++. Но OpenCv имеет поддержку многих языков.

## OpenFace

OpenFace [20] использует глубокие сети Nueral для реализации распознавания лиц. OpenFace-это в основном реализация python. Много pertained моделей имеющиеся для пользы. OpenFace очень проста в использовании. Сервер распознавания изображений можно создать в течение часа.

# Обзор методов решений задачи распознавания лиц

## FaceNet

FaceNet [21, 22] — нейронная сеть, которая учится преобразовывать изображения лица в компактное евклидово пространство (128D), где дистанция соответствует мере схожести лиц. Проще говоря, чем более похожи лица, тем они ближе.

Сеть состоит из пакетного входного слоя и глубокой CNN с последующей нормализацией L2, что приводит к встраиванию граней. Затем следует потеря триплета во время тренировки.

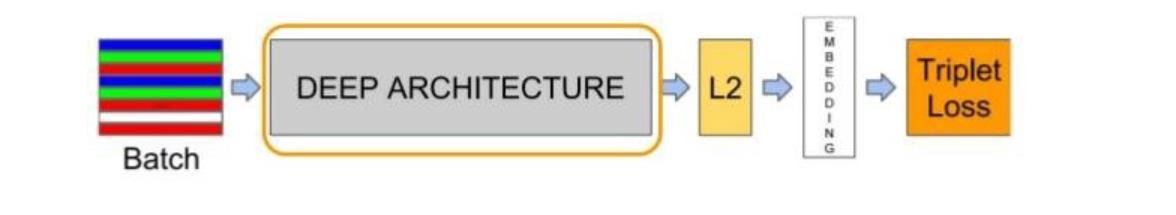
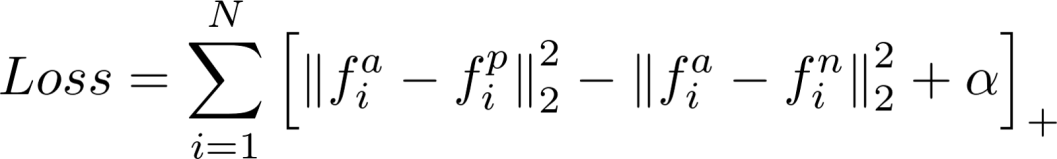


Рис. 10 FaceNet

FaceNet использует особую функцию потерь, называемую TripletLoss. Она минимизирует дистанцию между якорем и изображениями, которые содержат похожую внешность, и максимизирует дистанцую между разными.



* f(a) это энкодинг анкора
* f(p) это энкодинг похожих лиц (positive)
* f(n) это энкодинг непохожих лиц (negative)
* Альфа — это константа, которая позволяет быть уверенным, что сеть не будет пытаться оптимизировать напрямую f(a) — f(p) = f(a) — f(n) = 0
* […]+ экиввалентено max(0, sum)

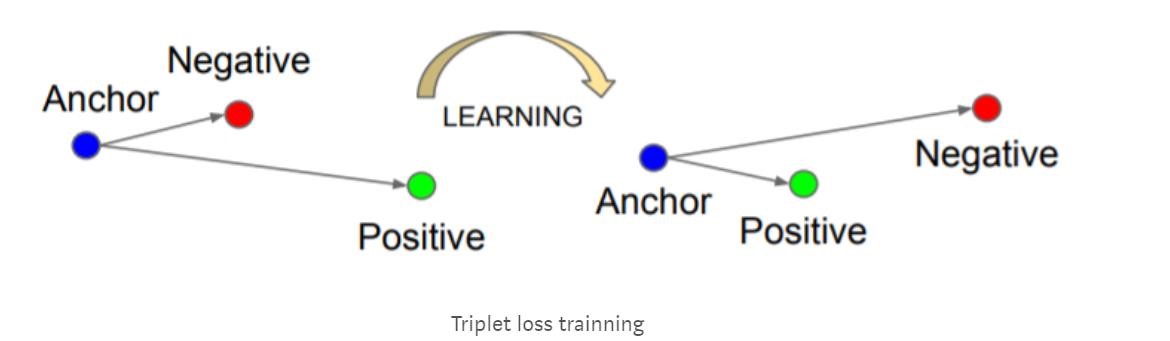


Рис. 11 Тренировка Triplet Loss

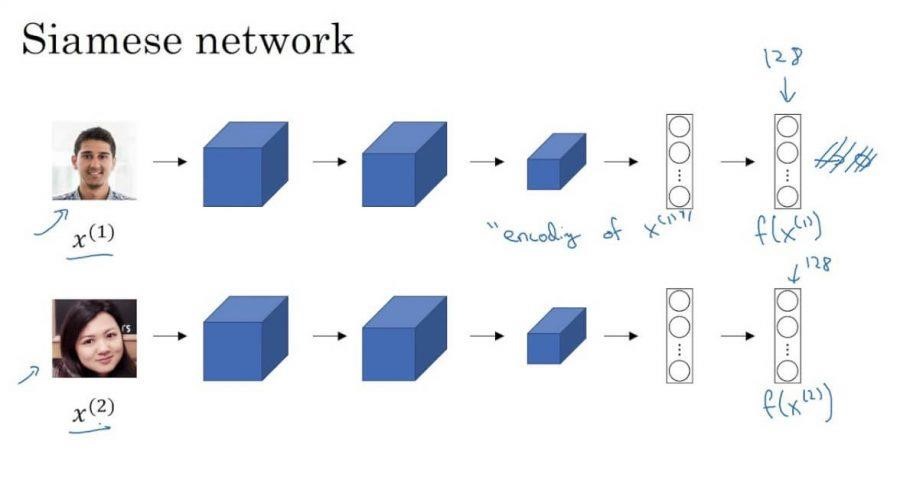
FaceNet — сиамская сеть. Сиамская сеть — тип архитектуры нейросети, который обучается диффиренцированию входных данных. То есть, позволяет научиться понимать какие изображения похожи, а какие нет.

Рис. 12 Описание сиамской сети

Сиамские сети состоят из двух идентичных нейронных сетей, каждая из которых имеет одинаковые точные веса. Во-первых, каждая сеть принимает одно из двух входных изображений в качестве входных данных. Затем выходы последних слоев каждой сети отправляются в функцию, которая определяет, содержат ли изображения одинаковые идентификаторы.

В FaceNet это делается путем вычисления расстояния между двумя выходами.

Обучив модель FaceNet, можно создать вложение для лица, введя его в модель. Чтобы сравнить два изображения, необходимо создать вложение для обоих изображений, пропуская модель по отдельности. Затем можно использовать приведенную выше формулу, чтобы найти расстояние, которое будет иметь меньшее значение для похожих граней и более высокое значение для разных граней.

## InsightFace

Обучающие данные включают в себя нормализованные наборы данных MS1M, VGG2 и CASIAWebface, которые уже были упакованы в двоичный формат MXNet. Сетевые магистрали включают в себя ResNet, MobilefaceNet, MobileNet, InceptionResNet\_v2, DenseNet, DPN. Функции потерь включают Softmax, SphereFace, CosineFace, ArcFace и Triplet (Euclidean / Angular) Loss [23, 24]. Авторы предлагают новую функцию потерь, аддитивный угловой запас (ArcFace), чтобы изучить отличительные особенности для надежного распознавания лиц.

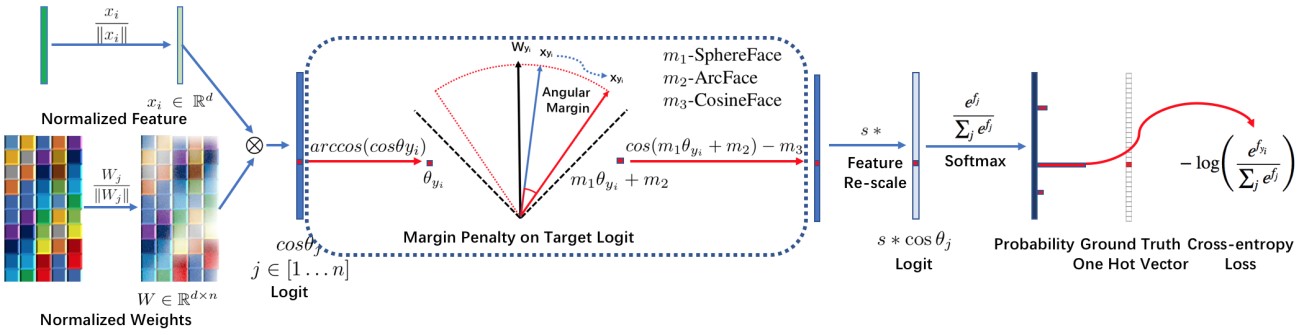


Рис. 13 Описание InsightFace

## SphereFace

SphereFace [25, 26] – фреймворк для глубокого распознавания лиц. В общем случае пайплайн содержит три этапа - обнаружение, выравнивание и распознавание лиц. Для нашей дальнейшей работы нас будет интересовать только этап распознавания лиц.

Данный метод основан на сопоставлении вектора в пространстве гиперсферы объекту (в нашем случае – изображению лица).

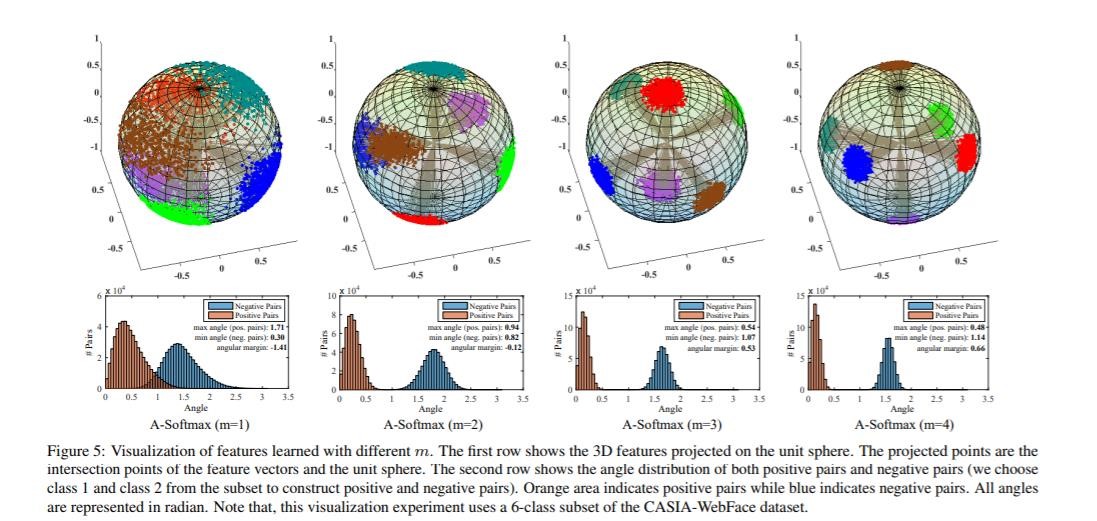
Этот метод предлагает функцию A-Softmax (angular softmax), которая позволяет сверточным нейронным сетям исследовать угловые отличительные особенности. Геометрически функция A-Softmax может рассматриваться как наложение дискриминационных ограничений на гиперсферное многообразие, которое по сути совпадает с тем, что изображения также лежат на многообразии.

Рис. 14 Интерпретация SphereFace

# Обзор фреймфорков и технологий, реализующих выбранное решение задачи распознавания лиц

## Torch

Torch [27] — библиотека для научных вычислений с широкой поддержкой алгоритмов машинного обучения. Библиотека реализована на языке Lua с использованием C и CUDA. Быстрый скриптовый язык Lua в совокупности с технологиями SSE, OpenMP, CUDA позволяют Torch показывать неплохую скорость по сравнению с другими библиотеками. На данный момент поддерживаются операционные системы Linux, FreeBSD, Mac OS X. Основные модули также работают и на Windows. В зависимостях Torch находятся пакеты imagemagick, gnuplot, nodejs, npm и другие.

Библиотека состоит из набора модулей, каждый из которых отвечает за различные стадии работы с нейросетями. Так, например, модуль nn обеспечивает конфигурирование нейросети (определению слоев, и их параметров), модуль optim содержит реализации различных методов оптимизации, применяемых для обучения, а gnuplot предоставляет возможность визуализации данных (построение графиков, показ изображений и т.д.). Установка дополнительных модулей позволяет расширить функционал библиотеки.

Torch позволяет создавать сложные нейросети с помощью механизма контейнеров. Контейнер — это класс, объединяющий объявленные компоненты нейросети в одну общую конфигурацию, которая в дальнейшем может быть передана в процедуру обучения. Компонентом нейросети могут быть не только полносвязные или сверточные слои, но и функции активации или ошибки, а также готовые контейнеры.

## PyTorch

PyTorch [28] - это пакет машинного обучения Python на основе Torch, представляющий собой пакет машинного обучения с открытым исходным кодом, основанный на языке программирования Lua. PyTorch имеет две основные функции:

- Тензорные вычисления (например, NumPy) с сильным ускорением GPU

- Автоматическое дифференцирование для построения и обучения нейронных сетей

В отличие от других библиотек, таких как TensorFlow, где нужно сначала определить весь вычислительный граф, прежде чем запустить свою модель, PyTorch позволяет определять свой граф динамически.

Тензоры PyTorch очень похожи на массивы NumPy с тем дополнением, что они могут работать на GPU. Это важно, потому что это помогает ускорить численные вычисления, которые могут увеличить скорость нейронных сетей в 50 и более раз .PyTorch глубоко интегрирован с кодом C++, таким образом, пользователи могут программировать на C/C ++ с помощью API расширения на основе c FFI для Python. Функциональность PyTorch построена как классы Python, поэтому весь его код может легко интегрироваться с пакетами и модулями Python. Он позволяет связывать высокоуровневые нейронные сетевые модули, потому что он поддерживает Keras-like API в своем пакете torch.nn.

PyTorch в основном используется для быстрого и эффективного обучения моделей глубокого обучения, поэтому он является основой выбора для большого числа исследователей.

У этого есть некоторые существенные преимущества:

(+) Процесс моделирования прост и прозрачен благодаря архитектурному стилю фреймворка;

(+) Режим определения по умолчанию больше похож на традиционное программирование, и вы можете использовать общие средства отладки, такие как отладчик pdb, ipdb или PyCharm;

(+) Имеет декларативный параллелизм данных;

(+) Легко писать свои собственные типы слоев и работать на GPU

(+) Имеет много предварительно обученных моделей и модульных частей, которые легко объединяются;

Но некоторые недостатки, конечно, неизбежны:

(-) Не хватает модельной порции

(-) В нем отсутствуют интерфейсы для мониторинга и визуализации, такие как Tensorboard, хотя вы можете подключиться к Tensorboard извне.

(-) Нет коммерческой поддержки

(-) Неясная документация

## Theano

Theano [29] — это библиотека Python и оптимизирующий компилятор, которые позволяют определять, оптимизировать и вычислять математические выражения эффективно используя многомерные массивы  
Возможности библиотеки:

* тесная интеграция с NumPy;
* прозрачное использование GPU;
* эффективное дифференцирование переменных;
* быстрая и стабильная оптимизация;
* динамическая генерация кода на C;
* расширенные возможности юнит-тестирования и самопроверок;

Плюсы и минусы

(+) Python + Numpy

(+) Вычислительный граф - хорошая абстракция

(+) RNNs хорошо вписываются в вычислительный граф

(-) Сырой Theano несколько низкоуровневый

(+) Обертки высокого уровня (Keras, Lasagne) облегчают разработку

(-) Сообщения об ошибках могут быть бесполезны

(-) Большие модели могут иметь длительное время компиляции

(-) гораздо «толще», чем Torch

(-) Пятнистая поддержка для предварительно обученных моделей

(-) Багги на AWS

(-) Одиночный графический процессор

## Keras

Keras - минималистичная библиотека [30] на основе Python, которую можно запускать поверх TensorFlow, Theano или CNTK. Он поддерживает широкий спектр слоев нейронных сетей, таких как сверточные слои, рекуррентные слои или плотные слои.

Преимущества:

(+) Прототипирование действительно быстро и просто;

(+) легкий с точки зрения построения моделей DL с большим количеством слоев;

(+) имеет полностью настраиваемые модули;

(+) упрощенный и интуитивный интерфейс

(+) имеет встроенную поддержку для обучения на нескольких графических процессорах;

(+) Его можно превратить в оценщики Tensorflow и обучить на кластерах графических процессоров в Google Cloud;

(+) может быть запущен на Spark;

(+) Он поддерживает графические процессоры NVIDIA, Google TPU и графические процессоры с поддержкой Open-CL, такие как AMD.

Недостатки:

(-) не всегда легко настроить;

(-) ограничен бэкэндами Tensorflow, CNTK и Theano, Deeplearning4j.

(-) не обеспечивает столько функций, как TensorFlow, и обеспечивает меньший контроль над сетью, поэтому это может быть серьезным ограничением, если вы планируете построить специальный тип модели DL.

## Deeplearning4j

Коммерческая среда с открытым исходным кодом [31], написанная в основном для Java и Scala, предлагающая мощную поддержку для различных типов нейронных сетей (таких как CNN, RNN, RNTN или LTSM).

Преимущества:

- надежный, гибкий и эффективный;

- Он может обрабатывать огромные объемы данных, не жертвуя скоростью;

- Он работает с Apache Hadoop и Spark, поверх распределенных процессоров или графических процессоров;

- Документация ясная;

- Он имеет версию для сообщества и корпоративную версию.

Поскольку Java не очень популярна среди проектов машинного обучения, сама структура не может полагаться на растущие кодовые базы.

## OpenVINO

OpenVINO [32] - набора инструментов Intel, основанный на сверточных нейронных сетях (CNN).

Для традиционного компьютерного зрения использует под собой OpenCV и OpenVX.

Для deep learning-а имеет внутри множество натренированных моделей и поддерживает модели из других популярных фреймворков: TensorFlow, Caffe, MXNet.

Продукт идеально подходит для реализации приложений, использующих Deep Learning и Computer Vision для решения задач. К примеру, производительность продукта при вычислении сетей на платформах Intel в разы выше по сравнению с популярными фреймворками. Также значительно ниже требования по используемой памяти, что актуально для ряда приложений. Тренировка сетей в продукт не включена, только запуск.

OpenVINO предоставляет OpenVINO Model Server. Предоставляя Inference сервис через интерфейс gRPC, механизм исполнения становится доступным для приложений, написанных на большинстве языков (C #, C ++, Java, Golang, JavaScript, Python и т. Д.), что делает интеграцию более легкой для разработчиков.

Плюсы: производительность, семплы, возможность запуска необходимых сетей.

Минусы: отсутствие тренировки, отсутствие прямой поддержки JavaScript (?).

## OpenCV

Загрузка и получение результатов (предсказаний) с помощью моделей, созданных в любом из трех популярных фреймворков (Caffe, TensorFlow, Torch), быстрая работа на CPU, поддержка основных слоев нейронных сетей, кроссплатформенность, открытость исходного кода и поддержка. Примеры использования доступны на C++ и Python (даже найден на JavaScript).

Тренировка сетей в продукт не включена.

Плюсы: часть обширной библиотеки КЗ, хорошая производительность на CPU, семплы, возможность запуска необходимых сетей.

Минусы: отсутствие тренировки.

## TensorFlow

TensorFlow [33]  — открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия. Применяется как для исследований, так и для разработки собственных продуктов Google. Основной API для работы с библиотекой реализован для Python, также существуют реализации для C++, Haskell, Java, Go и Swift.

Работа c TF строится вокруг построения и выполнения графа вычислений. Граф вычислений — это конструкция, которая описывает то, каким образом будут проводиться вычисления. В классическом императивном программировании мы пишем код, который выполняется построчно. В TF привычный императивный подход к программированию необходим только для каких-то вспомогательных целей. Основа TF — это создание структуры, задающей порядок вычислений. Программы естественным образом структурируются на две части — составление графа вычислений и выполнение вычислений в созданных структурах.

Плюсы: построение и тренировка сетей, JavaScript версия фреймворка + статьи на тему решения задач глубокого обучения в браузере.

Минусы: низкоуровневая.

# Заключение

Были проведены исследования методов детекции, методов выравнивания и методов распознавания лиц для решения поставленной задачи. Также приведены обзоры инструментов, с помощью которых можно будет решать задачу.

Выбранные решения:

Детекция - Faster R-CNN / сети, входящие в OpenVINO Toolkit (будет выбрано в зависимости от результатов точности работы)

Выравнивание – OpenCV

Распознавание - SphereFace

В качестве инструментов для работы выбраны OpenVINO и PyTorch (только для тренировки сетки, так как данная возможность отсутствует в OpenVINO)

# Список используемых источников

1. Viola and Jones, «Rapid object detection using a boosted cascade of simple features», Computer Vision and Pattern Recognition, 2001
2. https://docs.opencv.org/3.4.3/d7/d8b/tutorial\_py\_face\_detection.html
3. https://github.com/opencv/opencv/tree/master/samples/dnn/face\_detector
4. https://www.learnopencv.com/face-detection-opencv-dlib-and-deep-learning-c-python/
5. Histograms of Oriented Gradients for Human Detection - Navneet Dalal and Bill Triggs
6. https://towardsdatascience.com/vehicles-tracking-with-hog-and-linear-svm-c9f27eaf521a
7. Max-Margin Object Detection - Davis E. King
8. https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
9. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks - Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun
10. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks - Shaoqing Ren; Kaiming He ; Ross Girshick; Jian Sun
11. Fast R-CNN - Ross Girshick, Microsoft Research
12. Rich feature hierarchies for accurate object detection and semantic segmentation - Ross Girshick Jeff Donahue Trevor Darrell Jitendra Malik
13. Selective Search for Object Recognition - J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers and A.W.M. Smeulders
14. https://medium.com/@smallfishbigsea/understand-ssd-and-implement-your-own-caa3232cd6ad
15. https://medium.com/@jonathan\_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088
16. You Only Look Once: Unified, Real-Time Object Detection - Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi
17. https://docs.openvinotoolkit.org/2018\_R5/usergroup1.html
18. https://opencv.org/
19. http://openbiometrics.org/
20. https://github.com/cmusatyalab/openface
21. https://github.com/davidsandberg/facenet
22. FaceNet: A Unified Embedding for Face Recognition and Clustering - Florian Schroff, Dmitry Kalenichenko, James Philbin
23. https://github.com/deepinsight/insightface
24. ArcFace: Additive Angular Margin Loss for Deep Face Recognition - Jiankang Deng, Jia Guo, Niannan Xue, Stefanos Zafeiriou
25. SphereFace: Deep Hypersphere Embedding for Face Recognition - Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, Le Song
26. https://github.com/wy1iu/sphereface
27. http://torch.ch/
28. https://pytorch.org/
29. http://www.deeplearning.net/software/theano/
30. https://keras.io/
31. https://deeplearning4j.org/
32. https://software.intel.com/en-us/openvino-toolkit
33. https://www.tensorflow.org/