**АЛЬТЕРНАТИВНЫЙ ЭКЗАМЕН КИТГ-2022**

Тема:

**«Сопровождение объекта по видео»**

Выполнили:

Морик Арина гр.1308

Михайлова Мария гр.1308

Черникова Полина гр. 1308

2022 г.

# Содержание

[Вводные дан](#_gar8fspuc955)ные

[Введение](#_m679jagy2egy)

[1 способ: SORT и YOLOv3](#_rxv7lurpzmy)

[Результат работы на SORT и YOLOv3](#_4rhxcnyxw4lc)

[2 способ: SORT и YOLOv5](#_wtu5ii4kckr4)

[3 способ: DeepSORT и YOLOv5](#_irjrzc2s7j5p)

[Результат работы DeepSORT и YOLOv5](#_egibhyuugo9x)

[Сравнение результатов](#_22cm2krwtmjk)

[Видео-пример работы программы](#_47ntgecqlomx)

[Что не получилось реализовать](#_dekqiamljap)

[Вывод](#_xwkzgw8h6nnv)

[Теория](#_wqpa1gu8p8i6)

[Справка по IOU](#_brbetu1a395r)

[Справка по Венгерскому алгоритму](#_hp6nhq5xuk3o)

[Использование фильтра Калмана](#_n843r5kce1qs)

# Вводные данные:

| **Цель** | сопровождать объект на кадрах видео (осуществлять трекинг объекта) |
| --- | --- |
| **Проблема** | похожих целей может быть несколько, цель может пропадать из зоны видимости |
| **Объект** | цель на видео |
| **Предмет** | поиск и обнаружение обозначенного объекта на видео |

# Введение

Задача трекинга считается одной из самых сложных и обсуждаемых задач Computer Vision. Она включает в себя основополагающую задачу области – детекцию, но в отличие от нее ставит перед программистом цель не только обнаружить объект, но и сопровождать его.

Вместе с тем, трекинг является повсеместно необходимым. Он нужен для систем безопасности, аналитики посещаемости общественных мест, контроля за работниками производств и соблюдением ПДД, используется на фермах, улицах и многих, многих других местах. Более того, можно с уверенностью сказать, что с течением времени трекинг не только сохранит свою актуальность, но даже ее увеличит.

Несмотря на такую популярность, которая является соблазнительной для специалистов, крайне малое количество людей берутся за исполнение задачи, и среди них, как правило, те, кто посвящает всего себя Computer Vision и является довольно узкопрофильным программистом. Разумеется, в Интернете можно найти готовые решения, вот только, как правило, они не могут быть использованы в моменте, а нуждаются в малых и крупных доработках, дообучении модели и в прочих телодвижениях, которые требуют знаний и навыков не только в программировании, но и в высшей математике.

Таким образом, задача трекинга стала для нас вызовом, интересным и немного пугающим, а также продолжением темы нашего альтернативного экзамена в прошлом семестре, когда мы занимались реализацией системы детекции объектов по кадрам видео. Перед собой мы поставили цель изучить существующие решения, выбрать наиболее для нас подходящее, разобраться с программной реализацией решений на Python, как наиболее часто выбираемом для такой задачи языке программирования, и создать работающую программу с как можно более высоким процентом правильного отслеживания, при этом не отойдя от изначально обозначенных нам научным руководителем входных данных и условий.

# 1 способ: SORT и YOLOv3

После изучения теоретической части вопроса первым способом программной реализации стал алгоритм SORT с детектором [YOLOv3](#kix.zibn94rrh3tl).

**SORT** — это подход к отслеживанию объектов, в котором элементарные подходы, такие как [фильтр Калмана](#kix.e0kvmrdr3kqp) и [венгерский алгоритм](#kix.scu9yzctuc7), используются для отслеживания объектов и работают лучше, чем многие онлайн-трекеры.

**Обнаружение**

На каждом кадре видео работает детектор объектов. Обнаруженные объекты передаются на следующий шаг работы алгоритма SORT. Для пробной версии программы мы выбрали детектор YOLOv3-tiny и датасет coco.

**COCO -** Набор данных Microsoft Common Objects in Context ([COCO](https://cocodataset.org/)) является самым популярным на данный момент набором данных для обнаружения объектов.

Репозиторий на GitHub, на котором мы основывались: [**https://github.com/abewley/sort**](https://github.com/abewley/sort)

**YOLOv3**

YOLO или You Only Look Once — это очень популярная на текущий момент архитектура CNN, которая используется для распознавания множественных объектов на изображении.

Главная особенность этой архитектуры по сравнению с другими состоит в том, что большинство систем применяют CNN несколько раз к разным регионам изображения, в YOLO CNN применяется один раз ко всему изображению сразу.

YOLOv3-tiny — обрезанная версия архитектуры YOLOv3, состоит из меньшего количества слоев (выходных слоя всего 2). Она хуже предсказывает мелкие объекты и предназначена для небольших датасетов. Но, из-за урезанного строения, веса сети занимают небольшой объем памяти (~35 Мб). Этот вариант архитектуры детектора был выбран для пробной версии программы.

Использованный нами репозиторий на GitHub: [**https://github.com/ultralytics/yolov3**](https://github.com/ultralytics/yolov3)

# Результат работы на SORT и YOLOv3

На момент начала работы над данным проектом наша осведомленность, касательно устарелости данной пары алгоритмов, была низкой, что позволило нам начать изучение с более интуитивно понятного, хотя и не актуального, решения.

Пример работы программы при использовании SORT и YOLOv3:



На кадре видно, что обнаружение происходило неверно, хоть и было близко к объекту. Сам трекинг при таком подходе работает прерывисто: bbox’ы часто “слетают” с объекта, цвет bbox’а меняется на одном и том же объекте по несколько раз.

Очевидно, результат нас не устроил.

Полагая, что смена поколения алгоритма детектирования исправит ситуацию, мы решили заменить алгоритм YOLOv3 на более новый (2020 год) YOLOv5 (существуют более новые версии. Так, например, осенью этого года вышел YOLOv7, но существенно для нашего проекта разницы при просмотре обоих алгоритмов мы не увидели, поэтому остановили свой выбор на YOLOv5).

# 2 способ: SORT и YOLOv5

Была протестирована связка YOLOv5+SORT, но эта попытка дала не самые успешные результаты: в процессе трекинга один объект чаще распознавался как разные, выводились ложные обнаружения отсутствующих объектов.



За время изучения материала в сети мы не раз натыкались на пару алгоритмов YOLOv5 и DeepSORT. Поэтому, не добившись очевидных положительных результатов с SORT, мы решили перейти на DeepSORT.

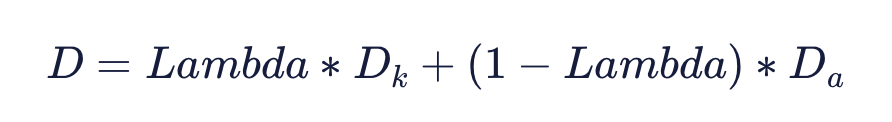
Использованный нам репозиторий YOLOv5 на GitHub: [**https://github.com/ultralytics/yolov5**](https://github.com/ultralytics/yolov5)

# 3 способ: DeepSORT и YOLOv5

**DeepSORT**

Нами уже был описан алгоритм SORT, в котором используется фильтр Калмана. Для работы DeepSORT используются фильтр Калмана и расстояние Махаланобиса.

Технология DeepSORT позволяет улучшить отслеживание объектов. В отличие от оригинального SORT, DeepSORT опирается не только на динамику объекта, но и на его внешний вид, что помогает избежать определения объекта как нового при его исчезновении из зоны видимости и последующего нового появления. В итоге расстояние от определенного YOLO объекта до предсказанного равно:



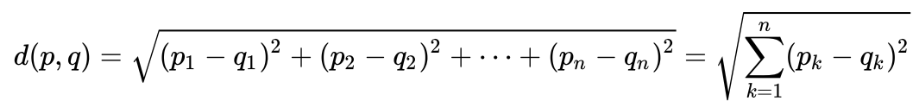
где Da — это дистанция по внешней схожести, а Dk — расстояние Махаланобиса.

На Хабр существует [статья о DeepSORT](https://habr.com/ru/post/514450/), в которой доступно объяснена суть расстояния Махаланобиса. Далее будут представлены выдержки из данной статьи (включая изображения).

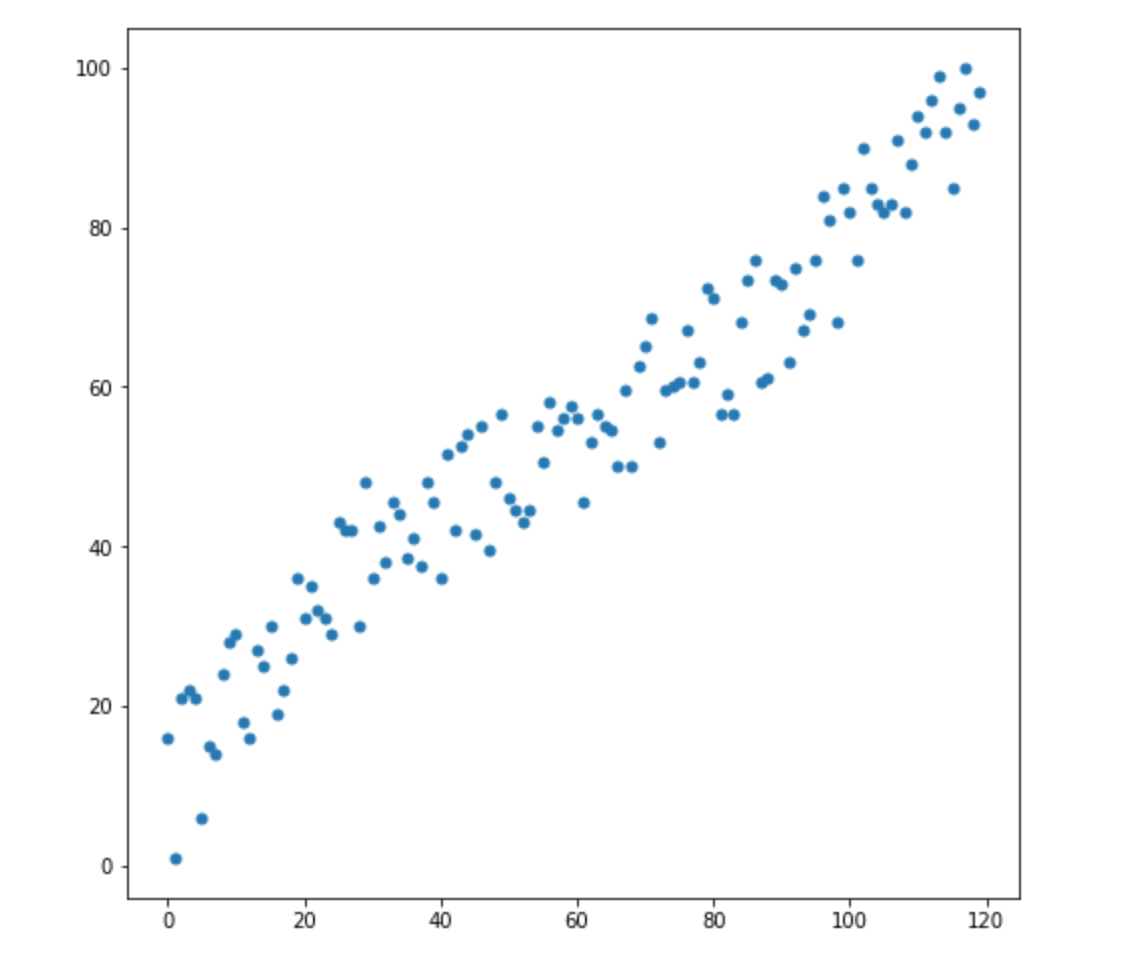
Репозиторий на GitHub, который мы использовали в работе: [**https://github.com/ConstantSun/Yolov5\_DeepSort\_Pytorch**](https://github.com/ConstantSun/Yolov5_DeepSort_Pytorch)

**Расстояние Махаланобиса**

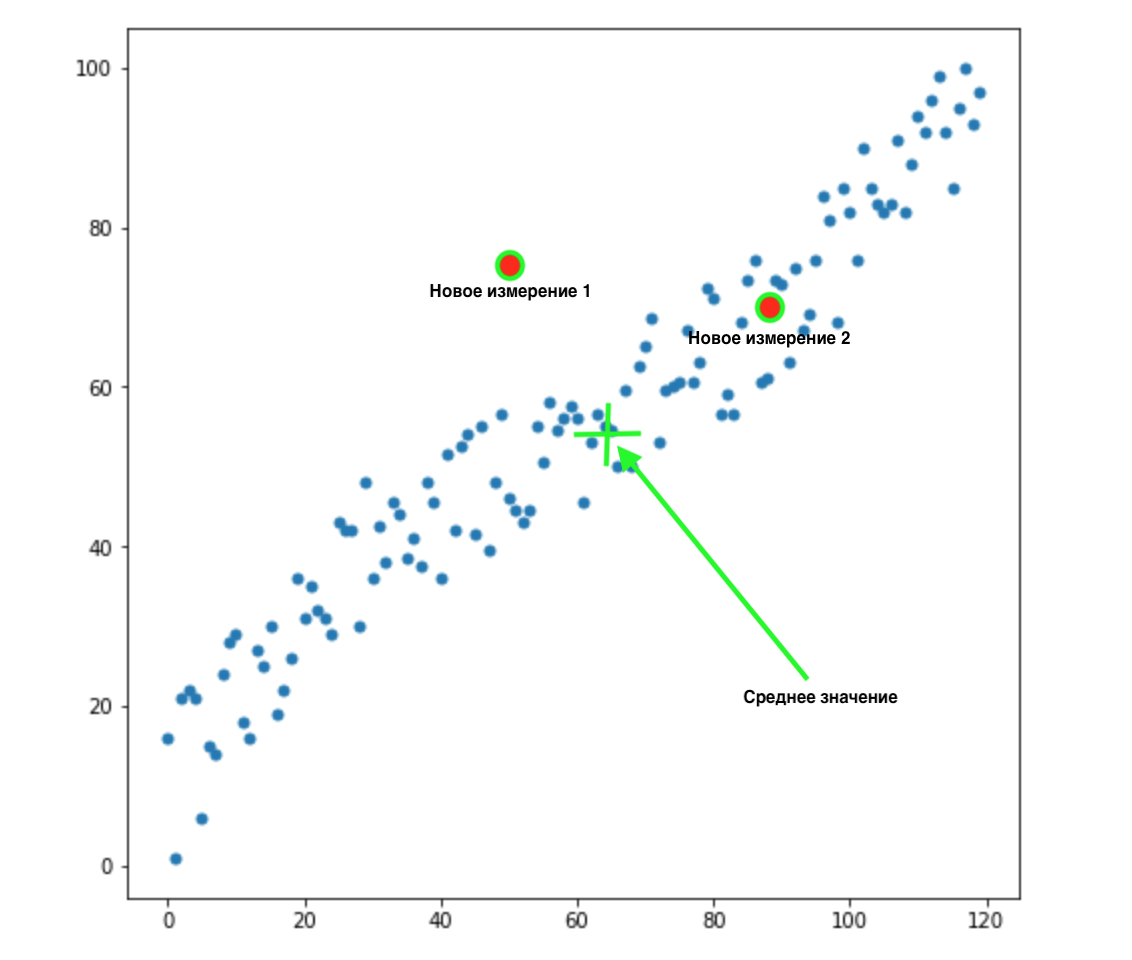
Рассмотрим очень простой пример, чтобы интуитивно понять, что такое расстояние Махаланобиса и зачем оно нужно. Многим, наверное, известно, что такое евклидово расстояние. Обычно, это расстояние от одной точки до другой в евклидовом пространстве:



Допустим, у нас есть две переменные — X1 и X2. Для каждой из них у нас есть много измерений.



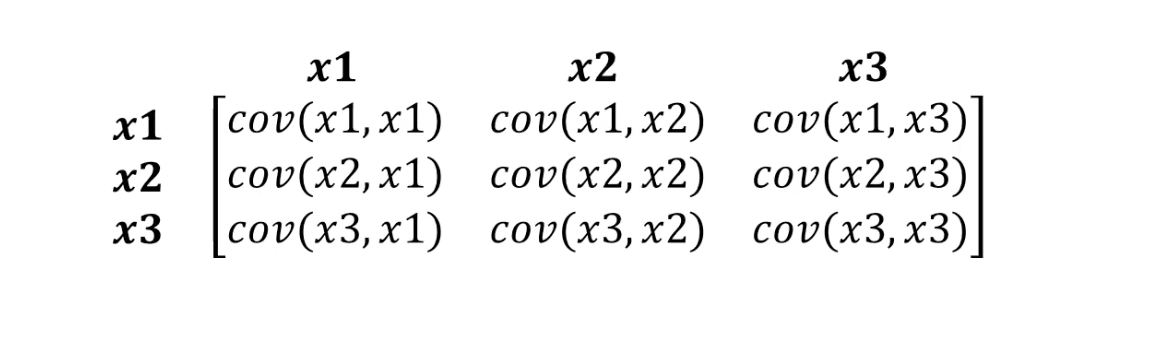
Теперь, допустим, у нас появилось 2 новых измерения:



Как понять, какое из этих двух значений наиболее подходит для нашего распределения? На глаз все очевидно — точка 2 нам подходит. Но вот евклидово расстояние до среднего значения у обоих точек одинаково. Соответственно, простое евклидово расстояние до среднего значения нам не подойдет.

Как мы видим из картинки выше, переменные между собой коррелируют, и довольно сильно. Если бы они не коррелировали между собой, или коррелировали намного меньше, мы могли бы закрыть глаза и применить евклидово расстояние для определенных задач, но здесь нам нужно сделать поправку на корреляцию и принять ее во внимание.

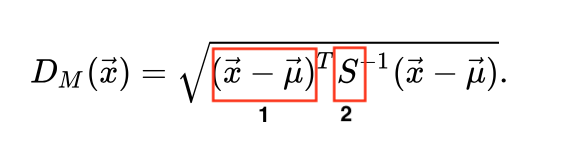
С этим как раз справляется расстояние Махаланобиса. Поскольку в обычно датасетах переменных больше чем двух, вместо корреляции мы будем использовать ковариационную матрицу:



Что на самом деле делает расстояние Махаланобиса:

* Избавляется от ковариации переменных
* Делает дисперсию (variance) переменных равной 1
* После этого использует обычное евклидово расстояние для трансформированных данных

Посмотрим на формулу, как вычисляется расстояние Махаланобиса:



Составляющие формулы:

1. Эта разница — разница между новой точкой и средними значениями для каждой переменной
2. S — это ковариационная матрица

Из формулы можно понять очень важную вещь. Мы по факту умножаем на перевернутую ковариационную матрицу. В этом случае, чем выше корреляция между переменными, тем скорее всего мы сократим дистанцию, так как будем домножать на обратное большему — то есть меньшее число (если простыми словами).

# Результат работы DeepSORT и YOLOv5

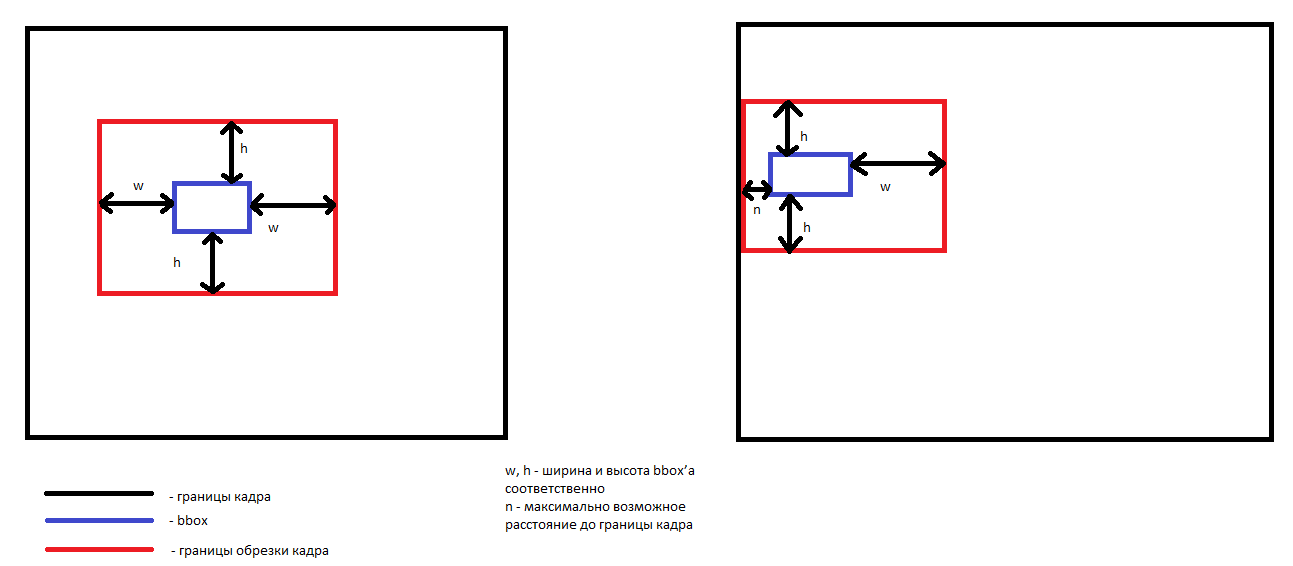
В результате реализации программы с DeepSORT и YOLOv5 мы получили трекер, наиболее устраивающий нас по результатам работы. Подходя к концу выполнения нашего проекта, мы решили улучшить работу программы, добавив обнаружение объекта по обрезанному кадру.

Предполагается, что примерное расположение объекта задано изначально, поэтому для выбранном для тестирования программы видео изначальные координаты обрезки кадра заданы так, что в кадре остается только необходимый объект.

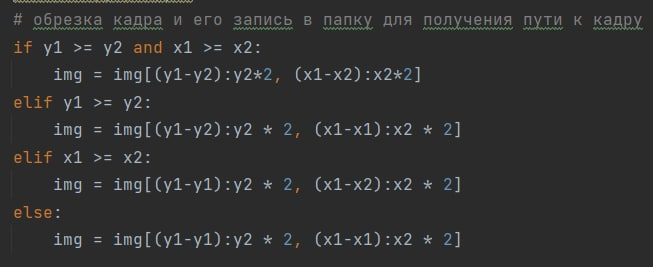


*рис. обрезка первого кадра видео*

При обрезке используются координаты, полученные от bbox’а, при этом обрезка всегда больше самого bbox’а. В нашем случае в каждые стороны откладывается расстояние, равное ширине/высоте bbox’а, или максимально возможное расстояние (если расстояние от bbox’а до края кадра меньше расстояния, на которое планировалось увеличение).



*рис. обрезка кадра*

**

*рис. программная реализация обрезки*

# Сравнение результатов

| Обработка видео | Результат нахождения объекта | Время обработки одного кадра без выбросов, с |
| --- | --- | --- |
| Без обрезки | 53% (для видео с 1464 кадрами, где объект ни разу не пропадал из кадра) | 0.700-0.850 |
| С обрезкой | 91% | 0.450-0.550 |

Процентное соотношение в столбце “Результат нахождения объекта” было рассчитано следующим образом: количество кадров с верно определенным отслеживаемым объектом было разделено на общее количество кадров.

# Видео-пример работы программы

<https://drive.google.com/file/d/1WOhWJSIinKsmy6DNgUOwL1dCH8iwa2Qo/view?usp=share_link>

# Что не получилось реализовать

Долгое время у нас хранилась идея запуска кода на GPU. Однако, спустя несколько неудачных попыток, было принято решение, что тратить на это время, которого и так не очень много, нецелесообразно. Очевидно причиной неудачи в данном аспекте является нехватка знаний и понимания в этом вопросе, что привело нас к немалому количеству тупиков в попытках наладить работу кода на GPU. В дальнейшем со временем мы надеемся изучить этот вопрос более тщательно.

# Вывод

В ходе выполнения задания были рассмотрены несколько подходов к решению данной задачи. Несмотря на обширное количество различных алгоритмов трекинга, для работы мы выбрали алгоритмы SORT/DeepSORT с детектором YOLOv5 после сравнительных экспериментов.

Результат является на наш взгляд весьма неплохим. Мы глубже погрузились в сложную математическую часть программирования, узнали новые для нас методы решения, а также изучили уже существующие работы на интересующую нас тему, что позволило нам расширить понимание современного положения дел в сфере, где мы потенциально можем начать работать.

Тем не менее, нынешний результат бесспорно нуждается в доработке: его можно сделать универсальнее и быстрее. Например, подключить GPU, использовать новейшие версии YOLO при наличии более высоких, нежели у нас, вычислительных мощностей компьютера.

Таким образом, мы и подготовили отличную основу для дальнейших исследований, и сделали готовый продукт, с которым уже можно работать.

# Теория

Изображение делится на сетку с ячейками размером S x S. Каждая ячейка может содержать несколько разных объектов для распознавания.

Во-первых, каждая ячейка отвечает за прогнозирование количества bounding box’ов. Также, каждая ячейка прогнозирует доверительное значение (*confidence value*) для каждой области, ограниченной bounding box’ом. Иными словами, это значение определяет вероятность нахождения того или иного объекта в данной области. То есть в случае, если какая-то ячейка сетки не имеет определенного объекта, важно, чтобы доверительное значение для этой области было низким.

Во-вторых, каждая ячейка отвечает за предсказание вероятностей классов - вероятность нахождения объекта. Допустим, если ячейка предсказывает автомобиль, это не гарантирует, что автомобиль в действительности присутствует в ней. Это говорит лишь о том, что если присутствует объект, то этот объект скорее всего автомобиль.

В YOLO используются anchor boxes (*якорные рамки / фиксированные рамки*) для прогнозирования bounding box’ов. Идея anchor box’ов сводится к предварительному определению двух различных форм. И таким образом, мы можем объединить два предсказания с двумя anchor box’ами. Эти якоря были рассчитаны с помощью датасета [COCO](https://cocodataset.org/#home) (*Common Objects in Context*) и кластеризации [k-средних](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_k-%D1%81%D1%80%D0%B5%D0%B4%D0%BD%D0%B8%D1%85) (*K-means clustering*).

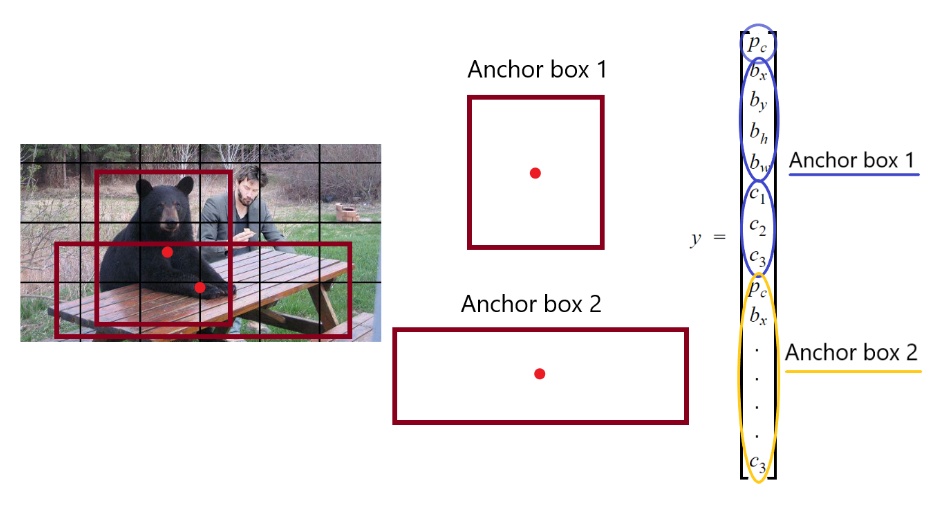


рис. Пример использования [anchor boxes](https://towardsdatascience.com/anchor-boxes-the-key-to-quality-object-detection-ddf9d612d4f9).

Таким образом, у нас есть сетка, где каждая ячейка предсказывает:

* Для каждого bounding box'а:
  + 4 координаты (tx , ty , tw , th)
  + 1 objectness error (*ошибка объектности*), которая является показателем уверенности в присутствии того или иного объекта
* Некоторое количество вероятностей классов

В общей сложности нейронная сеть содержит

S x S x [B \* (4+1+C)] выводов,

где B – это количество bounding box'ов, которое может предсказать ячейка на карте объектов, C – это количество классов, 4 – для bounding box'ов, 1 – для objectness prediction (*прогнозирование объектности*). За один проход мы можем пройти от входного изображения к выходному тензору, который соответствует обнаруженным объектам на картинке.

Теперь, если мы возьмем вероятность и умножим их на доверительные значения, мы получим все bounding box'ы, взвешенные по вероятности содержания этого объекта.

Простое нахождение порогового значения избавит нас от прогнозов с низким доверительным значением. Для следующего шага важно определить метрику [IoU](#kix.dqhdr0unmbaq) (*Intersection over Union / Пересечение над объединением*). Эта метрика равняется соотношению площади пересекающихся областей к площади областей объединенных.

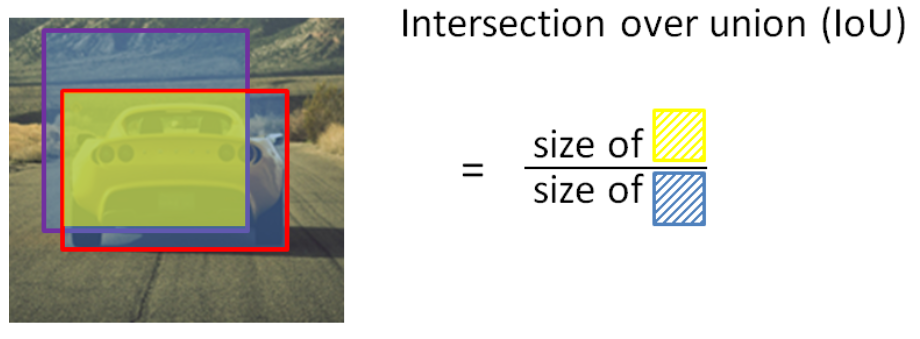


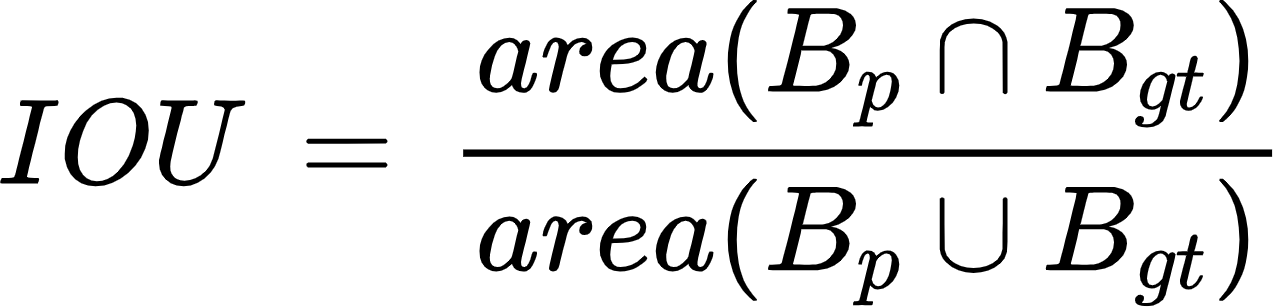
рис. [IoU](http://datahacker.rs/deep-learning-intersection-over-union/)

# Справка по IOU

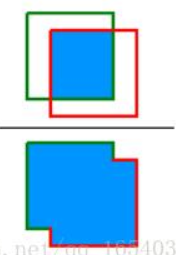
**(она же индекс Жаккарда/коэффициент подобия Жаккарда):**

IOU – метрика степени пересечения между двумя ограничивающими рамками. Данная метрика используется в процессе тестирования, чтобы понять, нуждается ли модель в дальнейшем обучении, увеличении/уменьшении тренировочной выборки, etc.

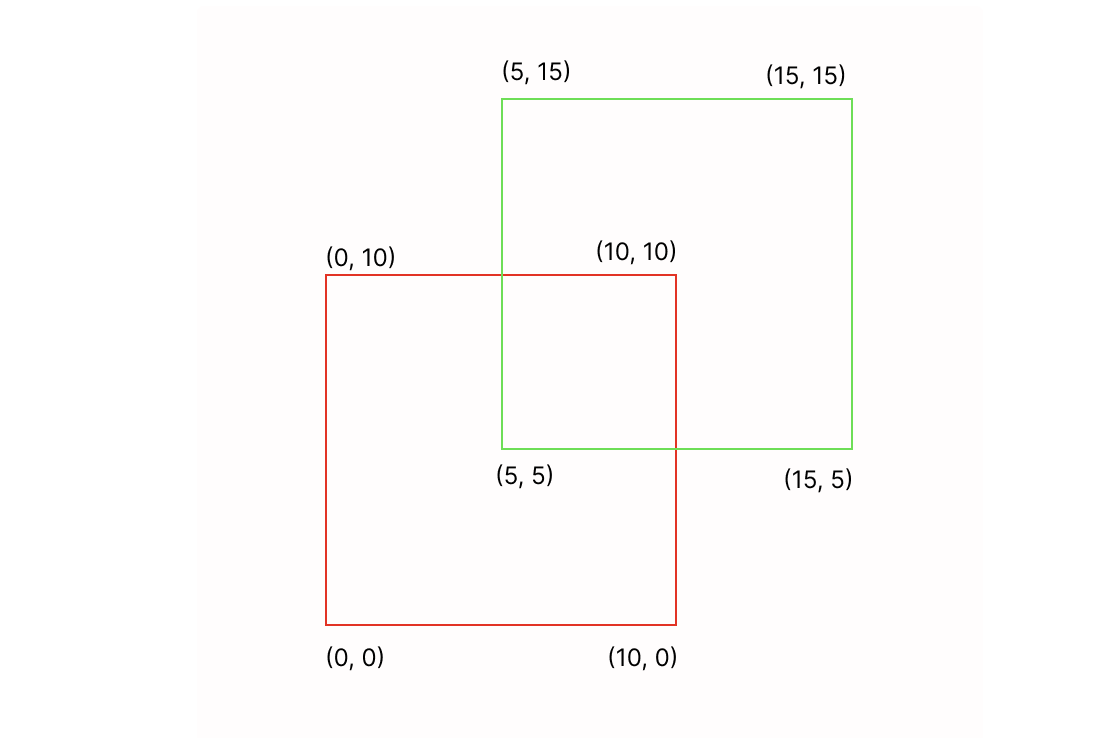
IOU – область пересечения между предсказанным положением объекта и его истинным значением, где положение = площадь ограничивающего прямоугольника, разделенная на объединение этих областей.

, где p – предсказанный, а gt – истинный прямоугольник.

Данная формула визуализируется следующим образом:



Полученное с помощью упомянутой формулы значение легко интерпретируется: чем больше перекрытие, тем выше оценка, тем лучше результат. Наилучшее возможное значение равно 1. Тем не менее, при реальных экспериментах единицы достичь крайне тяжело.

Пример наглядной работы алгоритма:

Площадь пересечения: 5 \* 5 = 25;

Площадь объединения: Область красного ограничивающего прямоугольника + Область зеленого ограничивающего прямоугольника - Площадь пересечения = 10\*10 + 10\*10 - 25 = 175

Тогда IOU: Площадь перекрытия / Площадь объединения= 25/175 ~ 0,14

# Справка по Венгерскому алгоритму:

В качестве первого шага алгоритма слежения необходимо установить соответствие между сегментами, найденными в текущем кадре, (т.е. найденными предположительными объектами) и отслеживаемыми объектами. С этой целью между каждым *i*-м объектом и каждым *j*-м сегментом вычисляется количественная мера сходства. В качестве такой меры можно использовать евклидово расстояние между прогнозируемыми координатами объекта  и центром сегмента , т.е.



Необходимо учитывать, что на изображении могут появляться новые объекты, а прослеживаемые в течение некоторого времени объекты могут покидать пределы кадра или временно заслоняться препятствиями. РАссмотрим три вида ситуаций:

а) найдено соответствие между объектом и сегментом;

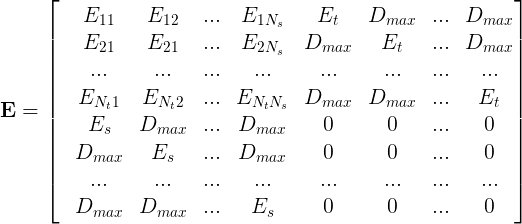
б) для данного объекта не найдено соответствия среди сегментов;

в) для данного сегмента не найдено соответствия среди объектов.

Евклидово расстояние можно рассматривать как стоимость принятия решения (а) о соответствии между *i*-м объектом и *j*-м сегментом. Введем величину *Et*, задающую стоимость решения (б), и величину *Es*, задающую стоимость решения (в).

В итоге приходим к решению следующей задачи: необходимо установить соответствие между сегментами и объектами или принять решение о невозможности такого сопоставления таким образом, чтобы суммарная стоимость всех решений была минимальной. Данная задача известна как задача о назначениях, для ее решения используется [венгерский](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%BD%D0%B3%D0%B5%D1%80%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) алгоритм.

Чтобы применить венгерский алгоритм, необходимо составить квадратную матрицу стоимости размера *NM*=*Nt*+*Ns*, где *Nt* – число отслеживаемых объектов, а *Ns* – число найденных сегментов. Матрица стоимостей имеет следующий вид:



где *Dmax* – достаточно большое число, такое что *Dmax* >> *Eij*. По строкам матрицы отсчитываются отслеживаемые объекты, по столбцам – найденные сегменты.

В результате выполнения венгерского алгоритма получим список пар *(t,s)k, k,t,s=1..NM*.

Если *t<=Nt* и *s<=Ns*, то между *t*-м объектом и *s*-м сегментом установлено соответствие [ситуация (а)].

Если *t<=Nt* и *s>Ns*, то для t-го объекта не найдено соответствующего ему сегмента [ситуация (б)].

Если *t>Nt* и *s<=Ns*, то для s-го сегмента не найдено подходящего объекта [ситуация (в)].

# Использование фильтра Калмана

Распространение обнаружения объекта от текущего кадра к следующему осуществляется с помощью фильтра Калмана, способного предсказывать будущее состояние динамической системы по имеющейся информации о ее состоянии в прошлом и настоящем, учитывая при этом некорректность, неполноту этой информации и ошибки при ее обработке. Использование данного алгоритма уменьшает время поиска и область поиска объекта на кадре, увеличивая в целом скорость трекинга.

В данном случае, имея расположение и масштаб рамки в текущем кадре необходимо рассчитать ее расположение и масштаб через промежуток времени на следующем кадре. Таким образоим, имеется дискретизированная по времени линейная динамическая система - параметры рамки, изменяющиеся от кадра к кадру за промежутки времени. Состояние системы описывается вектором состояния : x = {x,y,s,r,ẋ,ẏ,ṡ}

В каждый отрезок времени матрица эволюции процесса F, она же - матрица перехода с k-1-го шага на k-й, действует на вектор состояния x и переводит его в другой вектор состояния:



Для управляемых систем добавляется произведение матрицы управления В и вектора управляющих воздействий u. Однако, для имеющейся системы это слагаемое можно опустить,так как мы только наблюдаем за процессом и не можем никак задавать параметры наблюдаемого объекта в рамке.

Вектор w - вектор ошибки модели (вектор шума процесса) при случайных неотслеживаемых воздействиях на систему. В фильтре Калмана предполагается, что этот шум имеет нормальное распределение с нулевым математическим ожиданием, т.е. нулевым средним значением, которое не изменит новое состояние на шаге k, однако дисперсии ковариационной матрицы Q этого шума для более точного прогноза будут учитываться при расчете ошибки нового состояния.

Данное уравнение представлено для истинных состояний, которые не могут быть измерены непосредственно. Поэтому при действительном использовании фильтра для предсказываемых состояний используются их априорные (предшествующие) и апостериорные (последующие) оценки, обозначаемые с ~ или ^ сверху.

Также в каждый отрезок времени будет производиться измерение ( не всегда напрямую) наблюдаемых параметров системы (описывающейся вектором состояния x) с шумом v, вектором ошибки измерений с ковариационной матрицей R, использующейся при дальнейших расчетах.



Создадим модель состояния отслеживаемой системы, инициализируя начальные условия для работы фильтра.

1. Зададим вектор состояний x = {x,y,s,r,ẋ,ẏ,ṡ}: координаты центра рамки, ее площадь, соотношение сторон, движение ее координат и изменение площади и вектор измерений z = {x,y,s,r,}
2. Составим квадратную матрицу процесса/состояний (7x7)F: она задаёт систему линейных уравнений, описывающих законы изменения каждого из состояний {x,y,s,r,ẋ,ẏ,ṡ}
3. Определим матрицу измерений H(4x7): она связывает состояния модели с показаниями датчиков:

* столбцы содержат параметры рамки, заданной в централизованном виде: {x,y,s,r,};
* строки - упомянутые состояния системы;

1. Заполним квадратную ковариационную матрицу шума измерений R(4x4): она показывает, насколько можно доверять измеренным значениям датчиков;

* данные измерения независимы, поэтому элементы вне главной диагонали, соответствующие их ковариациям (зависимостям) будут иметь нулевые значения;
* на диагонали расположим значения дисперсий так, что малые величины будут соответствовать точным показаниям датчиков, большие - относительно неточным;

Координаты центра рамки, найденные относительно прямых измерений координат её углов, считаются показаниями более точными, чем косвенные - площадь и соотношение сторон, полученные в результате расчета;

1. Установим начальные значения квадратной ковариационной матрицы шума состояний P(7x7), а далее алгоритм самостоятельно будет обновлять их с каждым проходом фильтра: данная матрица показывает насколько предсказанные состояния соответствуют измерениям в действительности;

* состояния независимы, поэтому элементы вне главной диагонали, соответствующие ковариациям их неопределенностей будут иметь нулевые значения;
* малые значения дисперсии устанавливаются для показаний точного датчика, большие - для относительно неточного, чтобы позволить фильтру сравнительно быстро сходиться;

Для дисперсий движения координат центра рамки и изменения ее площади, рассчитываемых с учетом ошибок измерений постоянных величин, будут взяты значения значительно превосходящие остальные дисперсии.

1. Смоделируем квадратную ковариационную матрицу шума процесса Q(7x7): указывает, на какие переменные состояния будут в первую очередь влиять ошибки модели и неучтённые факторы;

* шум процесса прибавляется к шуму состояния в каждом предсказании приблизительно следующим образом:



т.е. малые значения в матрице Q не будут увеличивать неуверенность в оценке состояния P на каждом этапе; в таком случае, полагают, что построенная модель состояний будет точно описывать процесс и предсказаниям алгоритма можно доверять;

* поэтому обычно предполагается, что наибольшее значение в Q соответствует самому высокому порядку производной; однако это не является фиксированным правилом ее заполнения,и помимо использования моделей приближения непрерывного, либо кусочного белого шума, оптимальные значения для матрицы можно настроить для каждого отдельного случая экспериментально;
* состояния независимы, поэтому элементы вне главной диагонали, соответствующие ковариациям их неопределенностей будут иметь нулевые значения;
* на диагонали неопределенности ошибок процесса для координат центра рамки, ее площади, соотношения сторон устанавливаются наибольшими, для движения координат центра в 100 раз меньше, а для изменения площади в 10 000 раз меньше;

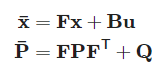
то есть в результате для параметров x,y,s,r алгоритм будет строить прогнозы больше доверяя данным измерениям, чем своим предсказаниям их состояний;

а для параметров ẋ,ẏ,ṡ наоборот, алгоритм будет строить прогнозы больше доверяя своим предсказаниям этих состояний (т.к. значения дисперсий для них в матрице Q сообщают о практически нулевых ошибках алгоритма при их определении), чем полученным расчетам, (т.к. значения дисперсий для них в матрице P сообщают неточности датчиков);

Теперь с помощью спроектированной модели состояния системы на этапе предсказания:

1.Используя матрицу процесса F мы можем получить новое состояние модели

2. Скорректировать неопределенность для следующего состояния по формулам:

,

где произведение управляющей матрицы B и управляющего входа u не рассматриваются в данной системе ввиду отсутствия фактора управления ей.

а x является состоянием на отрезке от k-1 до k:



На этапе обновления модели:

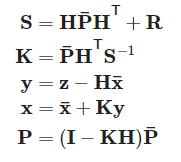
1. Мы имеем измерение (z) и степень его точности (H)

2. Вычисляем y: отклонение предсказанных показаний от показаний датчиков

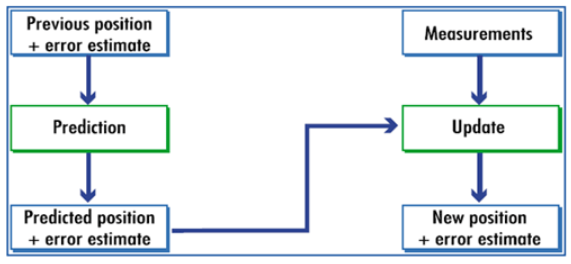
3. Ковариационную матрицу для отклонения,системную ошибку, включающую ошибку предсказанных показаний (PH) и ошибку измеренных показаний(R);

4. Устанавливаем усиление Калмана К: соотношение между прогнозом (P) и измерением(H) на основе коэффициента масштабирования S, что является более знакомым и лучше описывает данную динамическую систему(изменение рамки от кадра к кадру)

5. Обновляем состояния модели (х) и степень его точности (P), основанную на том, насколько мы уверены в измерении ( где I единичная матрица)



Схематично алгоритм работы фильтра Калмана можно изобразить следующим образом:



Фильтр работает постоянно, пока поступают измеренные значения. Он также может быть разомкнутым, поэтому сможет выполняться только шаг прогнозирования, если измерения недоступны. Тогда неопределенность состояния модели будет увеличиваться все больше и больше.