Object Detection III: YOLO, YOLOv2 / YOLO9000, YOLOv3

Антон Витвицкий

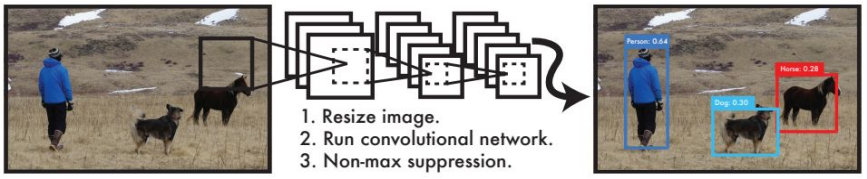
Otus, 2020

YOLO (You Only Look Once): Unified, Real-Time Object Detection, 2015
tags: *Object Detection, Real-Time Detectors*
arXiv, link2, link3, code

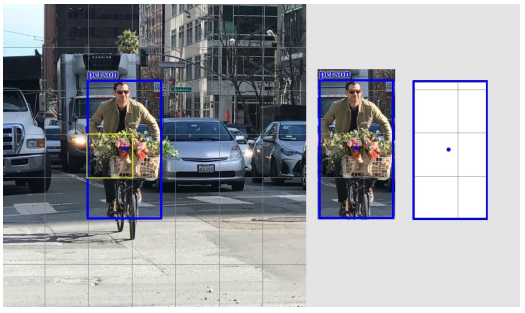
Авторы предложили новый подход к *Object Detection*: вместо того, чтобы использовать классификаторы для задач детекции (как в *DPM, R-CNN*), они определяют *Object Detection* как задачу регрессии к боксам (*bounding boxes*) и связанными с ними вероятностями классов (*class probabilities*). Модель *YOLO* состоит из одной *CNN*, которая делает предсказания за один проход, и может быть обучена end-to-end. Базовая модель может работать real-time на скорости *45* fps (на Titan X GPU), а версия *Fast YOLO* - *155* fps, обходя другие real-time детекторы по метрике *mAP*. В сравнении с state-of-the-art детекторами (на 2015 это *DPM*, *R-CNN*), *YOLO* менее точен в локализации объектов, однако выдает меньше ложных срабатываний (*false positives*).

1. Unified Detection

Авторы объединяют все компоненты детектора в единую сеть, что позволяет: 1) обучать модель end-to-end; 2) делать предсказания за один проход и работать в real-time; 3) использовать контекстную информацию со всего изображения для предсказания боксов. На *puc.1* представлена схема (pipeline) работы YOLO.



Puc. 1



Puc. 2

1.1 Grid Cell

YOLO разбивает входное изображение на сетку размером *S×S* (*puc.2*). Если центр объекта попадает в одну из ячеек сетки, то эта ячейка будет ответственна за детекцию этого объекта.

Каждая ячейка предсказывает фиксированное число *B* боксов (*bounding boxes)* и оценок (*box confidence scores)* для них (*puc. 3*). При этом, независимо от числа *B*, в каждой ячейка может детектится одновременно только один объект. Кроме того, каждая ячейка предсказывает *C* условных вероятностей для класса объекта (*conditional class probabilities*).

Каждый бокс состоит из пятерки: [*x, y, w, h*, *box confidence score]*. Координаты (*x,y*) представляют собой центр бокса относительно границ ячейки (т.е. дают смещение), а ширина и высота (*w,h*) нормализованы относительно всего изображения. Все координаты находятся в диапазоне [*0;1*].

Box confidence score отражает уверенность модели в том, что бокс содержит объект и как точно этот объект локализован.

Conditional class probabilities описывает вероятности того к какому классу принадлежит объект (для каждой клетки одна вероятность на каждый класс независимо от числа боксов *B*).

Class confidence score вычисляется на этапе теста для каждого бокса и дает итоговую оценку отражающую точность классификации объекта и точность локализации бокса для него.

Вычисление этих оценок показано на *puc.4*. Здесь *Pr(object)* - вероятность, что бокс содержит объект (*1* если есть, и *0* - если нет); *IoU* - *intersection under union* между предсказанным боксом и ground truth; *Pr(class_i)* - вероятность того, что объект принадлежит классу *i*; *Pr(class_i | object)* - вероятность того, что объект принадлежит классу *i*, если объект был обнаружен.

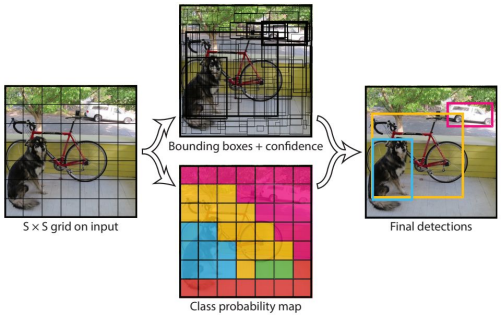


Рис. 3

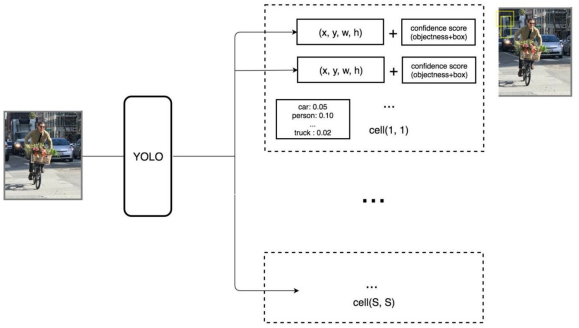


Рис. 5

$$\begin{aligned}\text{box confidence score} &\equiv P_r(object) \cdot IoU \\ \text{conditional class probability} &\equiv P_r(class_i|object) \\ \text{class confidence score} &\equiv P_r(class_i) \cdot IoU \\ &= \text{box confidence score} \times \text{conditional class probability}\end{aligned}$$

Рис. 4

Итоговый *pipeline* представлен на *рис.5*. Для *PASCAL VOC* авторы используют параметры *S=7, B=2, C=20*, что приводит к размеру выходного тензора: *(S, S, B×5 + C) = (7, 7, 2×5 + 20) = (7, 7, 30)*.

1.2 Network Design

В качестве модели *YOLO* использует *CNN* состоящую из *24* conv слоев и *2* полносвязных (*рис. 6*). Архитектура сети основана на *GoogLeNet*, где *inception* модуль заменен на чередующиеся слои: *1×1* conv (для понижения размерности *feature map*) с последующим *3×3* conv. Вход сети имеет размер *448×448*, а выход представлен тензором размера *7×7×30*. Кроме того, авторы представляют более быструю версию модели - *Fast YOLO*, которая состоит из *9* conv слоев вместо *24*, а также имеет меньшую глубину сверток.

В обоих случаях, сеть предобучается на *ImageNet 2012*, а в качестве функций активации используется *leaky ReLU*. Во время обучения используется *dropout* и аугментация данных (сдвиги/масштабирование до 20%, а также изменения цвета и насыщенности в пространстве *HSV*).

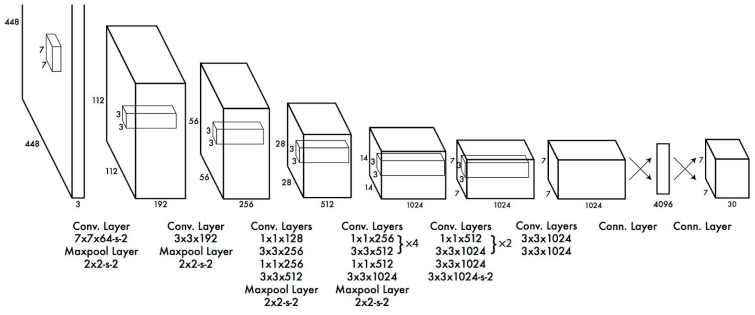


Рис. 6

1.3 Loss Function

YOLO предсказывает несколько боксов для каждой ячейки, т.е. имеет несколько *предикторов* (*predictors*) в этой ячейки, но во время обучения, только один *предиктор* отвечает за детекцию каждого объекта (и сл-но вносит вклад в *loss*). Для этого, “ответственным” назначается тот предиктор, чей бокс имеет максимальный *IoU* с ground truth. Это приводит к специализации этих предикторов: каждый предиктор обучается лучше предсказывать определенные размеры, пропорции и классы объектов, улучшая итоговую точность детекции.

YOLO использует *Sum of Squared Errors (SSE)* в качестве loss. Полный loss модели состоит из: 1) *Classification loss*; 2) *Localization loss*; 3) *Confidence loss*.

1.3.1 Classification loss

Отвечает за классификацию задетекченного объекта:

$$\sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c)\right)^2$$

, где $\mathbb{1}_i^{obj}$ - равен *1*, если объект присутствует в ячейке *i*, иначе *0*; $\hat{p}_i(c)$ - условная вероятность что объект класса *c* присутствует в ячейки *i*.

1.3.2 Localization loss

Измеряет ошибку в предсказании координат и размеров боксов. Вклад в loss вносят только “ответственные” предикторы:

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2\right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2\right]$$

, где $\mathbb{1}_{ij}^{obj}$ - равен *1* если предиктор *j* в ячейке i “ответственен” за детекцию этого объекта, иначе *0*; λ_{coord} - нормировочный коэффициент (равен *5*).

Корень для высоты (*w*) и ширины (*h*) используется для того, чтобы усилить разницу в ошибках локализации больших боксов и маленьких (иначе ошибка, например, в 2 пикселя для большого бокса равноценна ошибке в 2 пикселя для маленького бокса, а это не корректно).

1.3.3 Confidence loss

Измеряет ошибку *объектности* (*objectness*), т.е. что в предсказанном боксе действительно находится объект. Применяется только к “ответственному” предиктору:

$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i\right)^2$$

, где $\hat{C}_i$ - *box confidence score* предиктора *j* в ячейке *i*.

Для тек предикторов, которые не содержат объект, применяется:

$$\lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i\right)^2$$

, где $\mathbb{1}_{ij}^{noobj}$ - равен *1* если предиктор *j* в ячейке *i* не содержит объект; λ_{noobj} - нормировочный коэффициент (равен *0.5*).

Большинство боксов в ячейках не содержат объектов, поэтому это приводит к дисбалансу классов во время обучения (*background* детектится намного чаще объектов). Чтобы этого избежать, нормировочный коэффициент λ_{noobj} снижает вес вклада в loss предикторов без объектов.

1.3.4 Full loss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2\right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2\right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i\right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i\right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c)\right)^2$$

Рис. 7

1.4 Training

YOLO обучается в два этапа: 1) сеть предобучается на задаче классификации (на *ImageNet*) со входом *224×224*; 2) затем полносвязные слои заменяются conv слоями, вход сети увеличивается до *448×448,* и сеть дотюнивается на задаче детекции (на *VOC* или *COCO*).

1.5 Inference: Non-maximum suppression

YOLO может выдавать несколько детекций для одного и того же объекта. Поэтому к итоговым детекциям применяется *non-maximum suppression* (*NMS, рис.12*), что дает прирост на *2-3%* к *mAP*.

1.6 Limitations of YOLO

Из-за своей архитектуры (*grid cell*) YOLO имеет ограничения на детекцию близких друг к другу небольших объектов (например, стая птиц, или толпа людей, см. *рис. 7*).

2. Main Results

На *рис.8* представлена таблица сравнения real-time детекторов (и близких к real-time) на *Pascal VOC 2007*. На *рис.9* представлена диаграмма сравнения типа ошибок в *Fast R-CNN* и *YOLO*: *Correct* - класс предсказан верно и *IoU>0.5; Localization* - класс верен и *0.1<IoU<0.5*; *Similar* - класс похож и *IoU>0.1*; *Other* - класс не верен и *IoU>0.1*; *Background* - *IoU<0.1* для любых объектов.

На *рис.10* представлены результаты *VOC 2012 leaderboard* на 06.2015. YOLO единственный real-time детектор в таблице. *Fast R-CNN + YOLO* - детекции из R-CNN проверяются на YOLO.

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Рис. 8

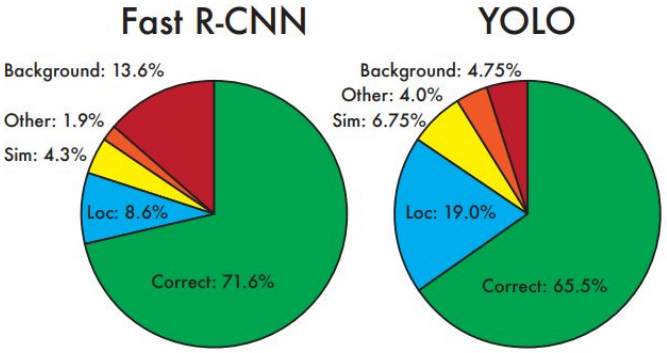


Рис. 9

VOC 2012 test	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
MR_CNN_MORE_DATA [11]	**73.9**	**85.5**	**82.9**	**76.6**	**57.8**	**62.7**	**79.4**	77.2	86.6	**55.0**	**79.1**	**62.2**	87.0	**83.4**	**84.7**	78.9	45.3	73.4	65.8	80.3	74.0
HyperNet_VGG	71.4	84.2	78.5	73.6	55.6	53.7	78.7	**79.8**	87.7	49.6	74.9	52.1	86.0	81.7	83.3	**81.8**	**48.6**	**73.5**	59.4	79.9	65.7
HyperNet_SP	71.3	84.1	78.3	73.3	55.5	53.6	78.6	79.6	87.5	49.5	74.9	52.1	85.6	81.6	83.2	81.6	48.4	73.2	59.3	79.7	65.6
Fast R-CNN + YOLO	70.7	83.4	78.5	73.5	55.8	43.4	79.1	73.1	**89.4**	49.4	75.5	57.0	**87.5**	80.9	81.0	74.7	41.8	71.5	68.5	**82.1**	67.2
MR_CNN_S_CNN [11]	70.7	85.0	79.6	71.5	55.3	57.7	76.0	73.9	84.6	50.5	74.3	61.7	85.5	79.9	81.7	76.4	41.0	69.0	61.2	77.7	72.1
Faster R-CNN [28]	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
DEEP_ENS_COCO	70.1	84.0	79.4	71.6	51.9	51.1	74.1	72.1	88.6	48.3	73.4	57.8	86.1	80.0	80.7	70.4	46.6	69.6	**68.8**	75.9	71.4
NoC [29]	68.8	82.8	79.0	71.6	52.3	53.7	74.1	69.0	84.9	46.9	74.3	53.1	85.0	81.3	79.5	72.2	38.9	72.4	59.5	76.7	68.1
Fast R-CNN [14]	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	**87.5**	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
UMICH_FGS_STRUCT	66.4	82.9	76.1	64.1	44.6	49.4	70.3	71.2	84.6	42.7	68.6	55.8	82.7	77.1	79.9	68.7	41.4	69.0	60.0	72.0	66.2
NUS_NIN_C2000 [7]	63.8	80.2	73.8	61.9	43.7	43.0	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	38.6	68.3	58.0	68.7	63.3
BabyLearning [7]	63.2	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6
NUS_NIN	62.4	77.9	73.1	62.6	39.5	43.3	69.1	66.4	78.9	39.1	68.1	50.0	77.2	71.3	76.1	64.7	38.4	66.9	56.2	66.9	62.7
R-CNN VGG BB [13]	62.4	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3
R-CNN VGG [13]	59.2	76.8	70.9	56.6	37.5	36.9	62.9	63.6	81.1	35.7	64.3	43.9	80.4	71.6	74.0	60.0	30.8	63.4	52.0	63.5	58.7
YOLO	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
Feature Edit [33]	56.3	74.6	69.1	54.4	39.1	33.1	65.2	62.7	69.7	30.8	56.0	44.6	70.0	64.4	71.1	60.2	33.3	61.3	46.4	61.7	57.8
R-CNN BB [13]	53.3	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1
SDS [16]	50.7	69.7	58.4	48.5	28.3	28.8	61.3	57.5	70.8	24.1	50.7	35.9	64.9	59.1	65.8	57.1	26.0	58.8	38.6	58.9	50.7
R-CNN [13]	49.6	68.1	63.8	46.1	29.4	27.9	56.6	57.0	65.9	26.5	48.7	39.5	66.2	57.3	65.4	53.2	26.2	54.5	38.1	50.6	51.6

Рис. 10

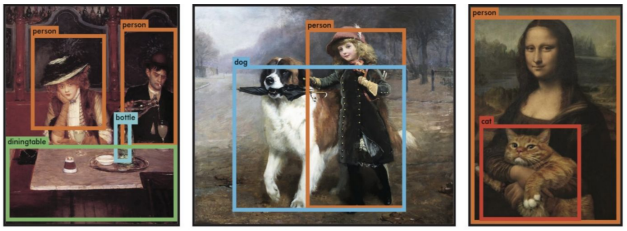


Рис. 11

3. Conclusion

Преимущества модели *YOLO*:

1) Очень быстрая детекция, т.к. задача решается за один проход через сеть, и не использует *sliding window* или *region proposals.*
2) Модель может быть обучена end-to-end.
3) Используется контекст объектов, т.к. сеть обрабатывает изображение целиком.
4) Хорошее обобщение на объекты из других доменов (например, рисунки, см *рис.11*), также дает меньше *false positive* срабатываний на *background*.

Рис. 12

YOLOv2 / YOLO9000, 2016

tags: *Object Detection, Real-Time Detectors*

arXiv, link2, link3, code

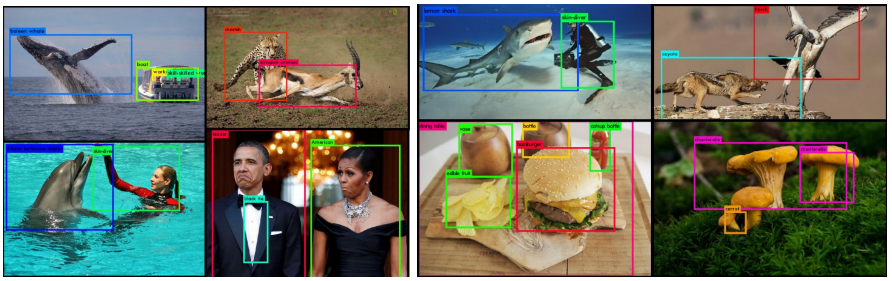


Рис.1

1. Accuracy improvements

Анализ показал, что YOLO имеет более низкую точность локализации в сравнении с Fast R-CNN, а также более низкий *recall* в сравнении с *region proposals* подходами. Поэтому авторы сфокусировались на улучшении *recall* и точности локализации, при сохранении хорошей точности классификации и высокой скорости детекции.

1.1 Batch normalization

Авторы добавляют *batch normalization* во все conv слои, что усиливает регуляризацию сети, устраняет необходимость в использовании *dropouts* и увеличивает *mAP* на 2%.

1.2 High-resolution classifier

Вспомним, что YOLO обучался задаче классификации на *ImageNet* со входом *224×224,* а затем вход увеличивался до *448×448*, и модель дотюнивался на задаче детекции (на *VOC* или *COCO)*.

YOLOv2 начинает обучение аналогично со входом *224×224,* но затем вход увеличивается до *448×448* и модель обучается еще несколько эпох на задаче классификации. После этого модель дотюнивается на задаче детекции (на *VOC* или *COCO)* и сходится быстрее. Такая схема обучения увеличивает *mAP* на 4%.

1.3 Convolutional with Anchor Boxes

На ранних эпохах обучения YOLO страдает от нестабильности градиента, т.к. пытается предсказывать боксы произвольной формы. Эти предсказания могут быть точными для одних объектов и неточными для других, что и приводит к крутым изменениям в градиенте. Кроме того, на ранних этапах обучения предикторы соревнуются друг с другом за специализацию на определенных формах боксов (см. *рис.2*). В реальных же доменах объекты не так разнообразны: машины имеют очень схожую форму, а пешеходы примерно одинаковое отношение сторон *0.41* (см. *рис.3*). Поэтому логично предположить, что обучение будет более стабильным, если модель будет использовать несколько наиболее общих форм боксов (*рис. 4*).

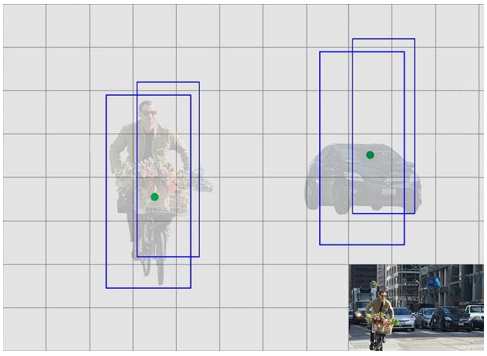


Рис.2

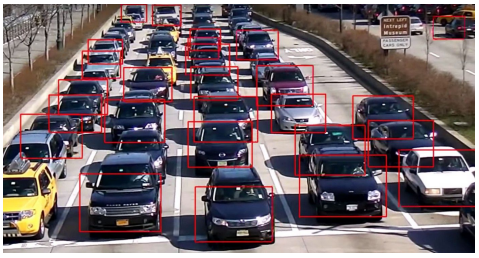


Рис.3

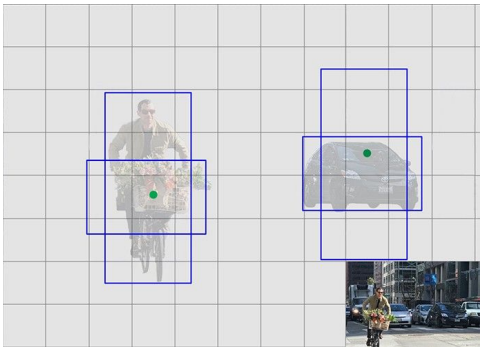


Рис.4

YOLOv2 заимствует идею из Faster R-CNN и использует *якорные боксы* (*anchors,* см. *рис.5*). Вместо предсказания боксов произвольных форм YOLO предсказывает смещения для этих якорных боксов (для *VOC* берется *5*). И если ограничить диапазоны этих смещений, то можно сохранить разнообразие форм, и заставить каждый якорный бокс специализироваться на конкретной форме, что в итоге приводит к более стабильному обучению модели на ранних этапах.

Для этих целей авторы вносят следующие изменения в архитектуру модели:

1) Удаляются все полносвязные слои ответственные за предсказания боксов (*рис.6*).
2) Предсказание класса объекта выносится с уровня *ячейки*, и предсказывается на уровне бокса (*рис.7*). Каждое предсказание включает: *4* координаты бокса, оценку объектности (*confidence score*) и *20* вероятностей для классов объектов (для *VOC*). Таким образом, каждая ячейка предсказывает *5×(4+1+20)=125* значений, а выходной тензор имеет форму *7×7×125* (*рис.8*).
3) Размер входа сети изменяется с *448×448* на *416×416* для того, чтобы иметь нечетное число ячеек (*7×7* вместо *8×8*). Так как центр изображения часто содержит большой объект, нечетное число ячеек помогает упростить его детекция (в противном случае мы имеем не *1*, а *4* ячейки которые соответствуют центру изображения). См. *рис.9*.
4) Выходной размер сети увеличивается с *7×7* до *13×13* путем удаления одного *pooling* слоя.

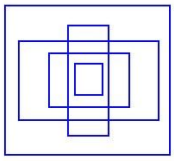


Рис.5

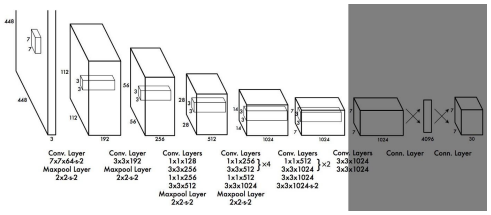


Рис.6

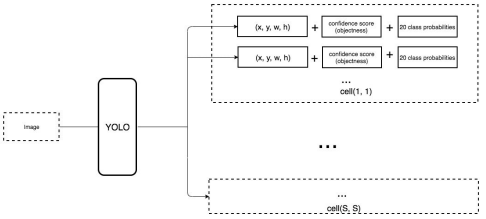


Рис.7

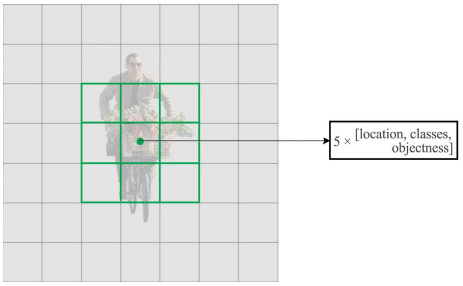


Рис.8

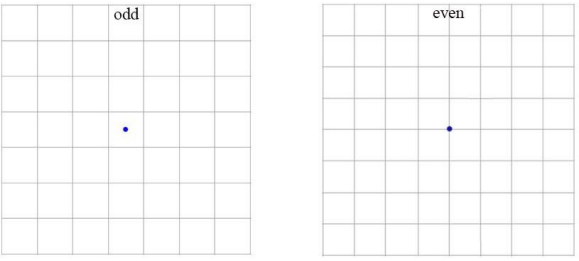


Рис.9

В итоге, использование якорных боксов слегка снижает mAP (с *69.5* до *69.2*), но при этом повышает recall с *81*% до *88*%, что уменьшает вероятность пропуска объектов.

1.4 Dimension Clusters

? Как выбрать эти 5 якорных точек*?* Чем лучше будет этот выбор, тем сеть будет лучше сходится. Для того, чтобы найти топ-5 якорных боксов, которые лучше всего покрывают датасет, авторы используют *k-means* кластеризацию и находят центроиды для топ-k кластеров (*рис.10*). Поскольку кластеризация идет на боксах, а не на точках, обычное евклидово расстояние не очень подходит для вычисления расстояния между боксами и центрами кластеров. Поэтому авторы используют *IoU*: $d(\text{box}, \text{centroid}) = 1 - \text{IOU}(\text{box}, \text{centroid})$.

На *рис.11* слева представлен график: среднее *IoU* между якорными боксами и ground truth в зависимости от выбранного числа кластеров (якорных боксов). Видно, что с повышением числа кластеров (якорных боксов) точность возрастает, однако вырастает и вычислительная сложность детектора. Поэтому авторы останавливаются на 5 кластерах. На *рис.11* справа представлены 5 якорных боксов для *COCO* (сиреневым цветом) и 5 для *VOC2007* (белым). В обоих случаях якорные боксы очень похожи, что говорит о том, что в реальных данных объекты действительно имеют скорее фиксированные, а не произвольные формы.

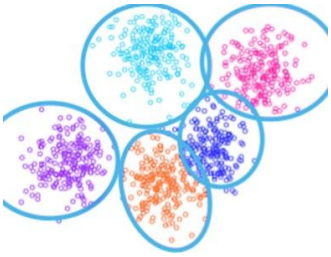


Рис.10

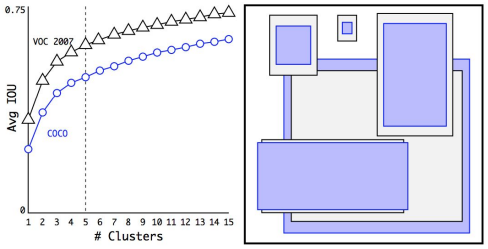


Рис.11

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$
$$Pr(\text{object}) * IOU(b, \text{object}) = \sigma(t_o)$$

Рис.12

1.5 Direct location prediction

Вторая проблема с которой столкнулись авторы при использовании якорных точек это нестабильность модели на начальных итерациях обучения. Эта нестабильность идет в основном из предсказания (*x,y*) координаты бокса. Вспомним, что *Region Proposals Network* (*RPN*) предсказывает значения *tx* и *ty*, из которых уже и выводятся координаты (*x,y*): $x = (t_x * w_a) - x_a$, $y = (t_y * h_a) - y_a$. Поэтому мы вновь сталкиваемся с некоторой рандомизацией. Однако, если ограничить эти значения, сеть будет более стабильна.

Поэтому YOLOv2 предсказывает пятерку (*tx,ty,tw,th,to*) и применяет *сигмоид* для ограничения диапазона их значений (см. *рис.12).* Здесь t_x, t_y, t_w, t_h - предсказания из YOLOv2; c_x, c_y - левый верхний угол клетки якорного бокса; p_w, p_h - ширина/высота якорного бокса; c_x, c_y, p_w, p_h - нормализованы к размеру изображения и имеют значения в диапазоне [*0;1*); b_x, b_y, b_w, b_h - вычисленные координаты предсказанного бокса; $\sigma(t_o)$ - *box confidence score*. На *рис.13* представлена визуализация: голубым обозначен предсказанный бокс, а пунктиром - якорный бокс.

Используя якорные точки, полученные через кластеризацию, а также прямое предсказание координат, YOLOv2 получает прирост к *mAP* на *5*%.

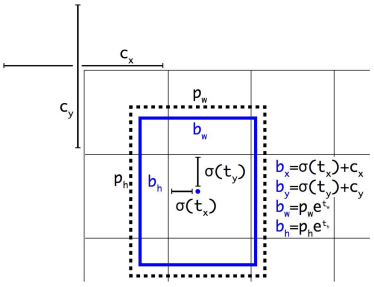


Рис.13

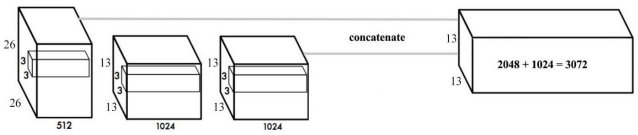


Рис.14

1.6 Fine-Grained Features

Сверточные слои постепенно снижают разрешение *feature maps*, поэтому на низких разрешениях становится сложнее детектить небольшие объекты. Разные детектора по разному подходят к решению этой проблемы. Например, *SSD* ищет объекты на *feature maps* разных слоев, поэтому каждый слой начинает специализироваться на разных масштабах объектов. *Faster R-CNN* использует пирамиду изображений для *RPN*. Авторы YOLOv2 применяют подход, основанный на использовании *passthrough* слоя. Этот слой берет фичи из более ранних слоев, решейпит их к форме выходного слоя, а затем конкатенирует их фичи. В итоге, YOLOv2 берет слой *26×26×512* и решейпит его *13×13×2048*, после чего конкатенирует с последним conv слоем *13×13×1024*. Таким образом, *feature map* выходного слоя имеет форму 13×13×3072 (*рис.14*), а детектор, работающий на этой *feature map* получает доступ к более *мелкозернистым* фичам. Это повышает *mAP* на *1*%.

1.7 Multi-Scale Training

После удаления всех полносвязных слоев сеть YOLOv2 становится *Fully Convolutional Network* (*FCN*), поэтому может работать на различных входных разрешениях. Например, если входное разрешение удваивается, то на выходе мы получаем в 4 раза больше ячеек, а значит и в 4 раза

больше предсказаний. Поскольку в YOLOv2 сети *коэффициент downsampling* равен *32*, то допускаются только входные разрешения делящиеся на *32*.

Во время обучения, вместо использования одного фиксированного размера, YOLOv2 случайно выбирает входное разрешение из списка {*320, 352, ..., 608*} (с шагом *32*) каждые *10* батчей. Это действует как аугментация и заставляет сеть лучше делать предсказания на разных масштабах входных изображений. На низких разрешениях модель работает быстрей, но менее точно. Благодаря этому, появляется возможность выбора между точностью и скоростью работы детектора.

На самом высоком разрешении YOLOv2 имеет *mAP=78.6* на *VOC 2007*, а на самом низком - *288×288* - работает на скорости *90* FPS, при этом выдает *mAP* почти равный *Fast R-CNN (рис.15)*.

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	**78.6**	40

Рис.15

	YOLO							YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓
anchor boxes?				✓	✓			
new network?					✓	✓	✓	✓
dimension priors?					✓	✓	✓	✓
location prediction?					✓	✓	✓	✓
passthrough?						✓	✓	✓
multi-scale?							✓	✓
hi-res detector?								✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Рис.16

1.8 Accuracy

На *рис.16* представлена таблица показывающая влияние вышеназванных техник на итоговую точность детектора. *На рис.17* показано сравнение YOLOv2 с другими детекторами на *VOC 2007*.

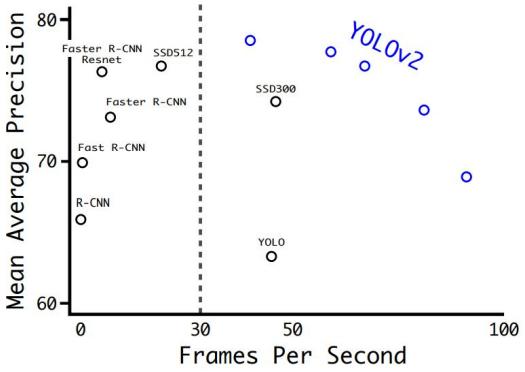


Рис.17

Type	Filters	Size/Stride	Output
Convolutional	32	3 × 3	224 × 224
Maxpool		2 × 2/2	112 × 112
Convolutional	64	3 × 3	112 × 112
Maxpool		2 × 2/2	56 × 56
Convolutional	128	3 × 3	56 × 56
Convolutional	64	1 × 1	56 × 56
Convolutional	128	3 × 3	56 × 56
Maxpool		2 × 2/2	28 × 28
Convolutional	256	3 × 3	28 × 28
Convolutional	128	1 × 1	28 × 28
Convolutional	256	3 × 3	28 × 28
Maxpool		2 × 2/2	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Maxpool		2 × 2/2	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	1000	1 × 1	7 × 7
Avgpool		Global	1000
Softmax			

Рис.18

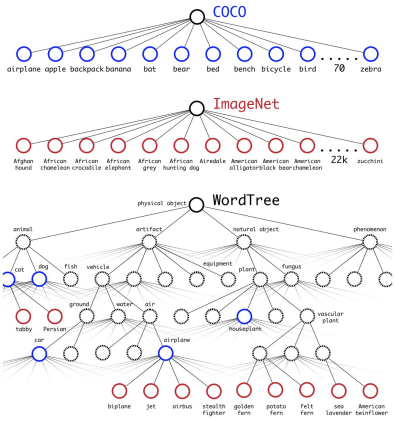


Рис.20

2. Speed improvement

Первый YOLO использует кастомную архитектуру основанную на *GoogLeNet* в качестве *backbone*. Эта сеть быстрее *VGG16* (которую использует *R-CNN*), но менее точна и выдает на *ImageNet* точность *88%* (top-5) против *90%* у *VGG16*.

2.1 DarkNet

В YOLOv2 используется собственная архитектура *DarkNet-19*, которая имеет *5.58* млрд. параметров (против *8.52* млрд у YOLO) и достигает точности *91.2%* на *ImageNet* (top-5). На *рис.18* представлена архитектура *DarkNet-19* (для задач классификации): в основном используются свертки 3×3, и 1×1 для понижения размерности фич, а также используется слой *average pooling*.

Для задач детекции последний conv слой (на *рис.18* зачеркнутая секция) заменяется на три conv слоя с шейпом *7×7×1024*, а также добавляется *1×1* слой чтобы преобразовать фичи из *7×7×1024* в шейп *7×7×125* (*5* боксов с *4* координатами + *confidence* score + *20 conditional class probabilities*). На *рис.19* показана архитектура *YOLOv2 + DarkNet-19*.

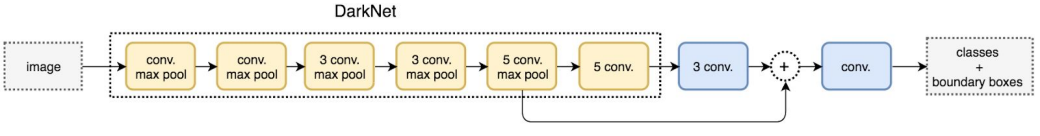


Рис.19

2.2 Training

1) YOLOv2 предобучается на *ImageNet 1000-классов* на разрешении *224×244* (*160* эпох).
2) После входное разрешение поднимается до *448×448* и обучается еще *10* эпох.
3) Далее полносвязные слои и последний conv удаляются, добавляются conv 3x3 и 1x1, а также *passthrough* слой, и модель обучается задаче детекции 160 эпох (*COCO* или *VOC*).

3. Classification

Датасеты для детекции объектов имеют гораздо меньше классов чем датасеты для классификации. Чтобы увеличить число классов которые YOLOv2 может детектить, авторы используют во время обучения оба типа датасетов. Когда встречается изображение из классификации, то градиент идет только через *loss* классификатора, а в качестве “ответственного” бокса выбирается тот, который выдал больший *class probability*. *И*спользуется несколько *softmax*.

? Однако встает проблема, как слить метки классов (*labels*) из разных датасетов? Для этого авторы объединяют все классы в иерархическую структуру *WordTree* (*рис. 20*).

3.1 YOLO9000

YOLO9000 это расширение модели YOLOv2 на *9000* классов. В качестве датасетов используется *COCO* (*200* классов) и топ *9000* классов из *ImageNet* (*44* класса пересекаются). Во время evaluation, *YOLO9000* способен детектировать объекты для всех классов (т.е. включая *156* классов для которых нет разметки детекции), и выдает *19.7 mAP* для всех классов, и *16.0 mAP* для *156* классов. Причем, *YOLO9000* хорошо обобщается на новые виды животных, т.к. некоторые их виды есть в *COCO*, но плохо - на категории одежды, т.к. в *COCO* одежды нет совсем.

YOLOv3: An Incremental Improvement, 2018

tags: *Object Detection, Real-Time Detectors*

arXiv, link2, link3, code, link4, link5

Авторы провели ряд экспериментов с использованием идей позаимствованных из других детекторов, моделей и архитектур, и написали tech report, назвав новую модель *YOLOv3*. YOLOv3 все также обладает достаточной точностью сравнимой со state-of-the-art (SOTA) моделями, при этом работает гораздо быстрей своих конкурентов.

1.1 Class Prediction

Для предсказания класса объекта предыдущие версии YOLO решали задачу *multi-class classification,* и использовали *softmax** и взаимоисключающие метки классов (в YOLO), или их иерархию (в YOLOv2). YOLOv3 вместо этого решает задачу *multi-label classification*, подразумевая что метки классов (*labels*) не взаимоисключающие (например, ребенок в кадре это и “ребенок” и “пешеход”). Таким образом, YOLOv3 заменяет *softmax* на независимые *логистические классификаторы* (*logistic classifiers***), и во время обучения для каждой метки класса, независимо от других, применяется *binary cross-entropy loss****. Такой подход помогает более эффективно обучаться на таких датасетах как Open Images Dataset, которые содержат много не взаимоисключающих меток, т.к. более точно моделирует их распределение, в отличие от *softmax*, который требует чтобы метка класса каждого объекта была эксклюзивна.

**Softmax* использует *softmax function*: $\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$. ***Logistic* использует *logistic function*: $f(x) = \frac{1}{1+e^{-x}}$.

****Cross-entropy*: $-log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)$, *binary cross-entropy*: $-t_1 log(f(s_1)) - (1-t_1) log(1-f(s_1))$.

1.2 Bounding Box Prediction & Objectness Score

Предсказание координат бокса выполняется аналогично YOLOv2 (*рис.1*). Но в отличие от YOLO, где для предсказания *objectness score* использовалась *Sum of Squared Errors (SSE),* YOLOv3 использует *логистическую регрессию* следующим способом:

- *objectness=1* только для предсказаний, чей бокс имеет максимальное *IoU* с *ground truth* (т.е. это *ответственный предиктор*).
- все остальные предсказания, чей *IoU>0.5,* не вносят вклад в *loss* (взято из *Faster R-CNN*).
- для каждого *ground truth* назначается только одно предсказание (*ответственный предиктор;* это отличается от *Faster R-CNN*).
- если для предсказания не назначено *ground truth*, то оно не вносит вклад в *classification loss* и в *localization loss*, а только в *objectness confidence loss*.

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

Рис.1

1.3 Predictions Across Scales

YOLOv3 делает три предсказания в каждой ячейке. Каждое предсказание состоит из: *bounding box offset*, *objectness score* и *class probability* для *N* классов. Таким образом, для *COCO* выходной тензор будет иметь форму $N \times N \times [3 * (4 + 1 + 80)]$, где *N×N* - число ячеек в *grid cell*, *3* предсказания на ячейку, *4* смещения (*offsets*) для бокса, *1* objectness score, и *80* классов объектов.

Для решения проблемы много масштабности YOLOv3 делает предсказания на *3* масштабах (аналогично *Feature Pyramid Network*):

1) На *feature map* последнего conv слоя
2) Затем на двух предпоследних слоях делает *upsampling* x2 и их *feature maps* конкатенируются с *feature map* более раннего слоя (размер которого в 2 раза больше). На итоговой *feature map* применяются свертки и получаем второй набор предсказаний.
3) Действие 2 повторяется еще раз, что позволяет переиспользовать вычисленные фичи и получить больше пространственной информации о положении объектов.

YOLOv3 (аналогично YOLOv2) используется *k-means* кластеризацию для получения якорных боксов (*anchors*). Для *COCO* выбирается *9* кластеров: (10×13),(16×30),(33×23),(30×61),(62×45),(59×119),(116 × 90),(156 × 198),(373 × 326). Далее эти кластеры разбиваются на *3* группы согласно их масштабу, и каждая группа используется для соответствующей *feature map* как описано выше.

1.4 Features Extractor

В качестве экстрактора фич YOLOv3 использует новую архитектуру *DarkNet-53* (*рис.2*) вместо *DarkNet-19* (как в YOLOv2). *DarkNet-53* состоит из чередующихся сверток 3×3 и 1×1 как у *DarkNet-19*, но слоев стало больше, а также добавляются *skip (shortcut) connections* (как у *ResNet*). Эта архитектура почти так же эффективна как *ResNet-152*, но имеет меньше параметров и работает в *2* раза быстрее. На *рис.3* представлено сравнение архитектур на *ImageNet* на Titan X GPU (точность, billions of operations, billion floating point operations per second и frames per second).

2 Main Results

На *рис.4* представлено сравнение YOLOv3 с другими SOTA детекторами. YOLOv3 показывает точность схожую с *SSD*, но при этом работает в 3 раза быстрее. Также авторам удалось решить проблему точной локализации небольших объектов (как это было в предыдущих версиях YOLO; см. *APsmall* колонку на *рис.4*). Однако YOLOv3 все еще позади *RetinaNet* по *AP*. Причем наибольшее отставание идет в метрике AP@IoU=.75, что говорит о том что YOLOv3 все также испытывает проблемы с точной локализацией объектов.

Авторы высказывают негодование в связи с тем, что в *COCO* вычисление *mAP* происходит на разных порогах, где YOLOv3 выдает не очень хорошие результаты (*рис.5*). Однако по “старому” способу расчета AP@IoU=.5 YOLOv3 выдает гораздо лучшие результаты (*рис.6*). Авторы ссылаются на исследования Ольги Руссоковской, где было показано что даже людей сложно натренировать замечать разницу в боксах между *IoU=0.3* и *IoU=0.5*.

3 Further fate of YOLO

Joseph Redmon в данном тех репорте (и в Twitter) выражает свое возмущение по поводу тому, как технологии компьютерного зрения применяются военными и корпорациями (см. *рис. 7*), и после этого перестал заниматься развитием YOLO.

Дальнейшее развитие YOLO было следующим (спасибо Станиславу Кускову за инфу):

Основным продолжателем стал один из контрибьютеров (и некоторые другие) проекта AlexeyAB. Он и создал версию yolo 4 (backbone был заменен, а часть yolo осталась прежней). 4 версия работает примерно так же по скорости, но при этом гораздо точнее (по моим субъективным ощущениям). Yolo4-tiny (облегченной версии) пока еще нет, но AlexeyAB сказал что она в разработке.

Также у AlexeyAB есть репозиторий, в котором подробная инструкция по использованию, ссылки на различные имплементации и многое другое. Я бы рекомендовал прочитать весь readme (https://github.com/AlexeyAB/darknet).

Также yolo3 поддерживается из deepstream 4.0 и выше для инференса на встраиваемых решениях (jetson). Очень хорошо оптимизирована, работает гораздо быстрее чем на обычной gpu (там вроде как немного измененная реализация).

Yolo5 - разработка (доработка) от ultralitics, ранее сделавшей порт yolo3 на pytorch. В качестве преимуществ указывается высокая скорость обучения.

Вот тут некоторые мысли по поводу yolo5:

https://github.com/opendatacam/opendatacam/issues/221 https://youtu.be/ptDTHIa2U3o

Также AlexeyAB говорит:

"YOLOv4 создавалась при поддержке Academia Sinica и MoST (Ministry of Science and Technology) Taiwan в рамках проекта, целью которого изначально было не создать, а выбрать лучшую нейронную сеть по заказу коммерческих корпораций. Поэтому они достаточно тщательно проверяли наши результаты. Собственно Tsung-Yi Lin автор MSCOCO, RetinaNet, FPN,… выходец из Academia Sinica тоже в курсе этих разработок, как и многие другие значимые люди."

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Рис. 2

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Рис.3 (верх)

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [3]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [6]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [4]	Inception-ResNet-v2 [19]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [18]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [13]	DarkNet-19 [13]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [9, 2]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [2]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [7]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [7]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Рис.4 (низ)

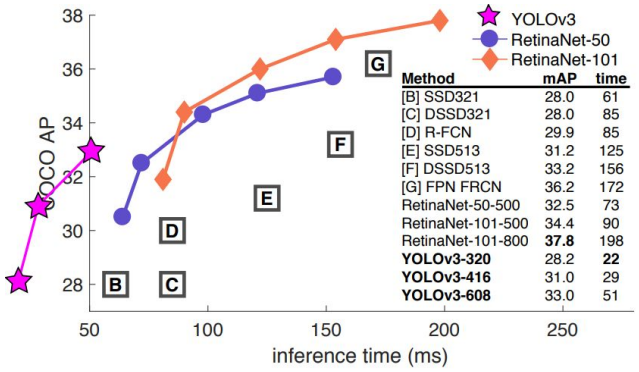


Рис.5

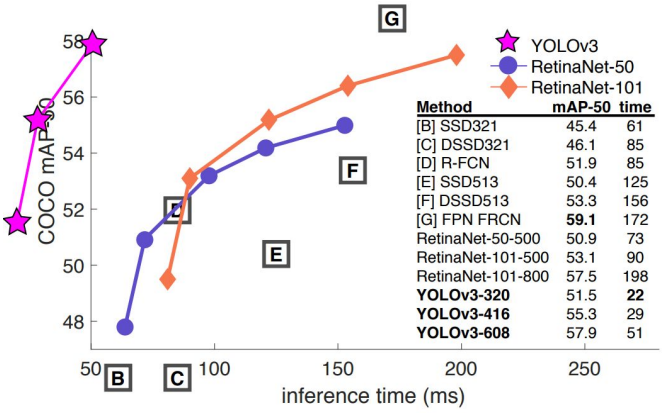


Рис.6

But maybe a better question is: "What are we going to do with these detectors now that we have them?" A lot of the people doing this research are at Google and Facebook. I guess at least we know the technology is in good hands and definitely won't be used to harvest your personal information and sell it to.... wait, you're saying that's exactly what it will be used for?? Oh.

Well the other people heavily funding vision research are the military and they've never done anything horrible like killing lots of people with new technology oh wait.....[1]

I have a lot of hope that most of the people using computer vision are just doing happy, good stuff with it, like counting the number of zebras in a national park [13], or tracking their cat as it wanders around their house [19]. But computer vision is already being put to questionable use and as researchers we have a responsibility to at least consider the harm our work might be doing and think of ways to mitigate it. We owe the world that much.

In closing, do not @ me. (Because I finally quit Twitter).

Рис. 7

YOLOv3 demo: https://youtu.be/MPU2HistivI