

□ - per grid of  $7 \times 7$   
predict:  $(x, y, w, h)$ , obj or not,  
class prob, confidence

Critical thinking: ask non-trivial questions  
why predict 2 boxes per grid?

## You Only Look Once: Unified, Real-Time Object Detection

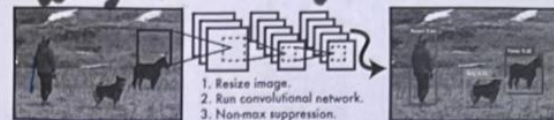
Joseph Redmon\*, Santosh Divvala<sup>†</sup>, Ross Girshick<sup>‡</sup>, Ali Farhadi<sup>†</sup>

University of Washington\*, Allen Institute for AI<sup>†</sup>, Facebook AI Research<sup>‡</sup>

<http://pjreddie.com/yolo/>

object det. requiring higher  
resol.

Why 448? Obj. det. re



**Figure 1: The YOLO Detection System.** Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to  $448 \times 448$ , (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

$7 \times 7 \times 30$

$7 \times 7$  (image divided) - intuition, no valid reason

methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding boxes, eliminate duplicate detections, and rescore the boxes based on other objects in the scene [13]. These complex pipelines are slow and hard to optimize because each individual component must be trained separately.

We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using our system, you only look once (YOLO) at an image to predict what objects are present and where they are.

YOLO is refreshingly simple: see Figure 1. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection.

First, YOLO is extremely fast. Since we frame detection as a regression problem we don't need a complex pipeline. We simply run our neural network on a new image at test time to predict detections. Our base network runs at 45 frames per second with no batch processing on a Titan X GPU and a fast version runs at more than 150 fps. This means we can process streaming video in real-time with less than 25 milliseconds of latency. Furthermore, YOLO achieves more than twice the mean average precision of other real-time systems. For a demo of our system running in real-time on a webcam please see our project webpage: <http://pjreddie.com/yolo/>.

Second, YOLO reasons globally about the image when

### Abstract

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is less likely to predict false positives on background. Finally, YOLO learns very general representations of objects. It outperforms other detection methods, including DPM and R-CNN, when generalizing from natural images to other domains like artwork.

### 1. Introduction

Humans glance at an image and instantly know what objects are in the image, where they are, and how they interact. The human visual system is fast and accurate, allowing us to perform complex tasks like driving with little conscious thought. Fast, accurate algorithms for object detection would allow computers to drive cars without specialized sensors, enable assistive devices to convey real-time scene information to human users, and unlock the potential for general purpose, responsive robotic systems.

Current detection systems repurpose classifiers to perform detection. To detect an object, these systems take a classifier for that object and evaluate it at various locations and scales in a test image. Systems like deformable parts models (DPM) use a sliding window approach where the classifier is run at evenly spaced locations over the entire image [10].

More recent approaches like R-CNN use region proposal

put object as 1 only the central grid (not all of them) if we detected object.

YOLO1 - predicts real coordinates

YOLO3 - predicts offset



making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance. Fast R-CNN, a top detection method [14], mistakes background patches in an image for objects because it can't see the larger context. YOLO makes less than half the number of background errors compared to Fast R-CNN.

Third, YOLO learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected inputs.

YOLO still lags behind state-of-the-art detection systems in accuracy. While it can quickly identify objects in images it struggles to precisely localize some objects, especially small ones. We examine these tradeoffs further in our experiments.

All of our training and testing code is open source. A variety of pretrained models are also available to download.

## 2. Unified Detection

We unify the separate components of object detection into a single neural network. Our network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously. This means our network reasons globally about the full image and all the objects in the image. The YOLO design enables end-to-end training and real-time speeds while maintaining high average precision.

Our system divides the input image into an  $S \times S$  grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.

Each grid cell predicts  $B$  bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. Formally we define confidence as  $\Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}}$ . If no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth.

Each bounding box consists of 5 predictions:  $x, y, w, h$ , and confidence. The  $(x, y)$  coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box.

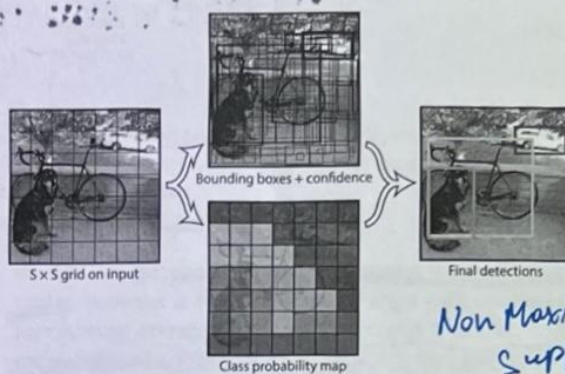
Each grid cell also predicts  $C$  conditional class probabilities,  $\Pr(\text{Class}_i | \text{Object})$ . These probabilities are conditioned on the grid cell containing an object. We only predict

one set of class probabilities per grid cell, regardless of the number of boxes  $B$ .

At test time we multiply the conditional class probabilities and the individual box confidence predictions,

$$\Pr(\text{Class}_i | \text{Object}) * \Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}} \quad (1)$$

which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.



**Figure 2: The Model.** Our system models detection as a regression problem. It divides the image into an  $S \times S$  grid and for each grid cell predicts  $B$  bounding boxes, confidence for those boxes, and  $C$  class probabilities. These predictions are encoded as an  $S \times S \times (B * 5 + C)$  tensor.

For evaluating YOLO on PASCAL VOC, we use  $S = 7$ ,  $B = 2$ . PASCAL VOC has 20 labelled classes so  $C = 20$ . Our final prediction is a  $7 \times 7 \times 30$  tensor.

### 2.1. Network Design

We implement this model as a convolutional neural network and evaluate it on the PASCAL VOC detection dataset [9]. The initial convolutional layers of the network extract features from the image while the fully connected layers predict the output probabilities and coordinates.

Our network architecture is inspired by the GoogLeNet model for image classification [34]. Our network has 24 convolutional layers followed by 2 fully connected layers. Instead of the inception modules used by GoogLeNet, we simply use  $1 \times 1$  reduction layers followed by  $3 \times 3$  convolutional layers, similar to Lin et al [22]. The full network is shown in Figure 3.

We also train a fast version of YOLO designed to push the boundaries of fast object detection. Fast YOLO uses a neural network with fewer convolutional layers (9 instead of 24) and fewer filters in those layers. Other than the size of the network, all training and testing parameters are the same between YOLO and Fast YOLO.

last layer  
reshape

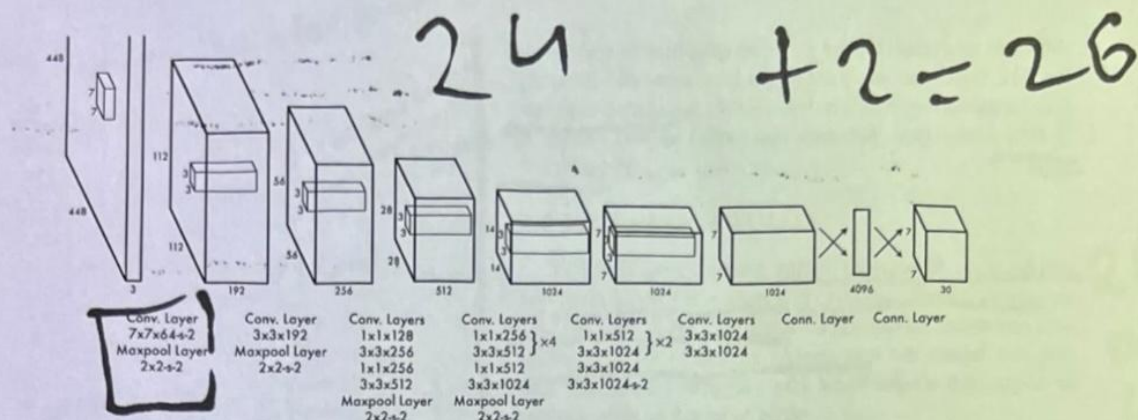
1x1  
3x3

john

9

code





**Figure 3: The Architecture.** Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating  $1 \times 1$  convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution ( $224 \times 224$  input image) and then double the resolution for detection.

The final output of our network is the  $7 \times 7 \times 30$  tensor of predictions.

## 2.2. Training

We pretrain our convolutional layers on the ImageNet 1000-class competition dataset [30]. For pretraining we use the first 20 convolutional layers from Figure 3 followed by a average-pooling layer and a fully connected layer. We train this network for approximately a week and achieve a single crop top-5 accuracy of 88% on the ImageNet 2012 validation set, comparable to the GoogLeNet models in Caffe's Model Zoo [24]. We use the Darknet framework for all training and inference [26].

We then convert the model to perform detection. Ren et al. show that adding both convolutional and connected layers to pretrained networks can improve performance [29]. Following their example, we add four convolutional layers and two fully connected layers with randomly initialized weights. Detection often requires fine-grained visual information so we increase the input resolution of the network from  $224 \times 224$  to  $448 \times 448$ .

Our final layer predicts both class probabilities and bounding box coordinates. We normalize the bounding box width and height by the image width and height so that they fall between 0 and 1. We parameterize the bounding box  $x$  and  $y$  coordinates to be offsets of a particular grid cell location so they are also bounded between 0 and 1.

We use a linear activation function for the final layer and all other layers use the following leaky rectified linear activation:

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases} \quad (2)$$

We optimize for sum-squared error in the output of our

model. We use sum-squared error because it is easy to optimize, however it does not perfectly align with our goal of maximizing average precision. It weights localization error equally with classification error which may not be ideal. Also, in every image many grid cells do not contain any object. This pushes the "confidence" scores of those cells towards zero, often overpowering the gradient from cells that do contain objects. This can lead to model instability, causing training to diverge early on. *class imbalance*

To remedy this, we increase the loss from bounding box coordinate predictions and decrease the loss from confidence predictions for boxes that don't contain objects. We use two parameters,  $\lambda_{\text{coord}}$  and  $\lambda_{\text{noobj}}$  to accomplish this. We set  $\lambda_{\text{coord}} = 5$  and  $\lambda_{\text{noobj}} = .5$ . **5 and 0.5**

Sum-squared error also equally weights errors in large boxes and small boxes. Our error metric should reflect that small deviations in large boxes matter less than in small boxes. To partially address this we predict the square root of the bounding box width and height instead of the width and height directly.

YOLO predicts multiple bounding boxes per grid cell. At training time we only want one bounding box predictor to be responsible for each object. We assign one predictor to be "responsible" for predicting an object based on which prediction has the highest current IOU with the ground truth. This leads to specialization between the bounding box predictors. Each predictor gets better at predicting certain sizes, aspect ratios, or classes of object, improving overall recall.

During training we optimize the following, multi-part



loss function:

$$\begin{aligned} \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{I}_{ij}^{obj} & \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{I}_{ij}^{obj} & \left[ (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\ + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{I}_{ij}^{obj} & (C_i - \hat{C}_i)^2 \quad \text{- Cost} \\ + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{I}_{ij}^{noobj} & (C_i - \hat{C}_i)^2 \\ \text{background} + \sum_{i=0}^{S^2} \mathbb{I}_i^{obj} \sum_{c \in \text{classes}} & (p_i(c) - \hat{p}_i(c))^2 \quad (3) \end{aligned}$$

where  $\mathbb{I}_i^{obj}$  denotes if object appears in cell  $i$  and  $\mathbb{I}_{ij}^{obj}$  denotes that the  $j$ th bounding box predictor in cell  $i$  is "responsible" for that prediction.

Note that the loss function only penalizes classification error if an object is present in that grid cell (hence the conditional class probability discussed earlier). It also only penalizes bounding box coordinate error if that predictor is "responsible" for the ground truth box (i.e. has the highest IOU of any predictor in that grid cell).

We train the network for about 135 epochs on the training and validation data sets from PASCAL VOC 2007 and 2012. When testing on 2012 we also include the VOC 2007 test data for training. Throughout training we use a batch size of 64, a momentum of 0.9 and a decay of 0.0005.

Our learning rate schedule is as follows: For the first epochs we slowly raise the learning rate from  $10^{-3}$  to  $10^{-2}$ . If we start at a high learning rate our model often diverges due to unstable gradients. We continue training with  $10^{-2}$  for 75 epochs, then  $10^{-3}$  for 30 epochs, and finally  $10^{-4}$  for 30 epochs.

To avoid overfitting we use dropout and extensive data augmentation. A dropout layer with rate = 0.5 after the first connected layer prevents co-adaptation between layers [18]. For data augmentation we introduce random scaling and translations of up to 20% of the original image size. We also randomly adjust the exposure and saturation of the image by up to a factor of 1.5 in the HSV color space.

### 2.3. Inference

Just like in training, predicting detections for a test image only requires one network evaluation. On PASCAL VOC the network predicts 98 bounding boxes per image and class probabilities for each box. YOLO is extremely fast at test time since it only requires a single network evaluation, unlike classifier-based methods.

The grid design enforces spatial diversity in the bounding box predictions. Often it is clear which grid cell an object falls in to and the network only predicts one box for each object. However, some large objects or objects near

the border of multiple cells can be well localized by multiple cells. Non-maximal suppression can be used to fix these multiple detections. While not critical to performance as it is for R-CNN or DPM, non-maximal suppression adds 2-3% in mAP.

### 2.4. Limitations of YOLO

YOLO imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class. This spatial constraint limits the number of nearby objects that our model can predict. Our model struggles with small objects that appear in groups, such as flocks of birds.

Since our model learns to predict bounding boxes from data, it struggles to generalize to objects in new or unusual aspect ratios or configurations. Our model also uses relatively coarse features for predicting bounding boxes since our architecture has multiple downsampling layers from the input image.

Finally, while we train on a loss function that approximates detection performance, our loss function treats errors the same in small bounding boxes versus large bounding boxes. A small error in a large box is generally benign but a small error in a small box has a much greater effect on IOU. Our main source of error is incorrect localizations.

### 3. Comparison to Other Detection Systems

Object detection is a core problem in computer vision. Detection pipelines generally start by extracting a set of robust features from input images (Haar [25], SIFT [23], HOG [4], convolutional features [6]). Then, classifiers [36, 21, 13, 10] or localizers [1, 32] are used to identify objects in the feature space. These classifiers or localizers are run either in sliding window fashion over the whole image or on some subset of regions in the image [35, 15, 39]. We compare the YOLO detection system to several top detection frameworks, highlighting key similarities and differences.

**Deformable parts models.** Deformable parts models (DPM) use a sliding window approach to object detection [10]. DPM uses a disjoint pipeline to extract static features, classify regions, predict bounding boxes for high scoring regions, etc. Our system replaces all of these disparate parts with a single convolutional neural network. The network performs feature extraction, bounding box prediction, non-maximal suppression, and contextual reasoning all concurrently. Instead of static features, the network trains the features in-line and optimizes them for the detection task. Our unified architecture leads to a faster, more accurate model than DPM.

**R-CNN.** R-CNN and its variants use region proposals instead of sliding windows to find objects in images. Selective

Proposed  
0.5

2 box  
and  
1 class

because it  
uses real  
coordinates,  
not offsets.

Weight decay  $\lambda \theta^2$  - ridge regularization



Search [35] generates potential bounding boxes, a convolutional network extracts features, an SVM scores the boxes, a linear model adjusts the bounding boxes, and non-max suppression eliminates duplicate detections. Each stage of this complex pipeline must be precisely tuned independently and the resulting system is very slow, taking more than 40 seconds per image at test time [14].

YOLO shares some similarities with R-CNN. Each grid cell proposes potential bounding boxes and scores those boxes using convolutional features. However, our system puts spatial constraints on the grid cell proposals which helps mitigate multiple detections of the same object. Our system also proposes far fewer bounding boxes, only 98 per image compared to about 2000 from Selective Search. Finally, our system combines these individual components into a single, jointly optimized model.

**Other Fast Detectors** Fast and Faster R-CNN focus on speeding up the R-CNN framework by sharing computation and using neural networks to propose regions instead of Selective Search [14] [28]. While they offer speed and accuracy improvements over R-CNN, both still fall short of real-time performance.

Many research efforts focus on speeding up the DPM pipeline [31] [38] [5]. They speed up HOG computation, use cascades, and push computation to GPUs. However, only 30Hz DPM [31] actually runs in real-time.

Instead of trying to optimize individual components of a large detection pipeline, YOLO throws out the pipeline entirely and is fast by design.

Detectors for single classes like faces or people can be highly optimized since they have to deal with much less variation [37]. YOLO is a general purpose detector that learns to detect a variety of objects simultaneously.

**Deep MultiBox.** Unlike R-CNN, Szegedy et al. train a convolutional neural network to predict regions of interest [8] instead of using Selective Search. MultiBox can also perform single object detection by replacing the confidence prediction with a single class prediction. However, MultiBox cannot perform general object detection and is still just a piece in a larger detection pipeline, requiring further image patch classification. Both YOLO and MultiBox use a convolutional network to predict bounding boxes in an image but YOLO is a complete detection system.

**OverFeat.** Sermanet et al. train a convolutional neural network to perform localization and adapt that localizer to perform detection [32]. OverFeat efficiently performs sliding window detection but it is still a disjoint system. OverFeat optimizes for localization, not detection performance. Like DPM, the localizer only sees local information when making a prediction. OverFeat cannot reason about global context and thus requires significant post-processing to produce coherent detections.

**MultiGrasp.** Our work is similar in design to work on

grasp detection by Redmon et al [27]. Our grid approach to bounding box prediction is based on the MultiGrasp system for regression to grasps. However, grasp detection is a much simpler task than object detection. MultiGrasp only needs to predict a single graspable region for an image containing one object. It doesn't have to estimate the size, location, or boundaries of the object or predict it's class, only find a region suitable for grasping. YOLO predicts both bounding boxes and class probabilities for multiple objects of multiple classes in an image.

## 4. Experiments

First we compare YOLO with other real-time detection systems on PASCAL VOC 2007. To understand the differences between YOLO and R-CNN variants we explore the errors on VOC 2007 made by YOLO and Fast R-CNN, one of the highest performing versions of R-CNN [14]. Based on the different error profiles we show that YOLO can be used to rescore Fast R-CNN detections and reduce the errors from background false positives, giving a significant performance boost. We also present VOC 2012 results and compare mAP to current state-of-the-art methods. Finally, we show that YOLO generalizes to new domains better than other detectors on two artwork datasets.

### 4.1. Comparison to Other Real-Time Systems

Many research efforts in object detection focus on making standard detection pipelines fast. [5] [38] [31] [14] [17] [28] However, only Sadeghi et al. actually produce a detection system that runs in real-time (30 frames per second or better) [31]. We compare YOLO to their GPU implementation of DPM which runs either at 30Hz or 100Hz. While the other efforts don't reach the real-time milestone we also compare their relative mAP and speed to examine the accuracy-performance tradeoffs available in object detection systems.

Fast YOLO is the fastest object detection method on PASCAL; as far as we know, it is the fastest extant object detector. With 52.7% mAP, it is more than twice as accurate as prior work on real-time detection. YOLO pushes mAP to 63.4% while still maintaining real-time performance.

We also train YOLO using VGG-16. This model is more accurate but also significantly slower than YOLO. It is useful for comparison to other detection systems that rely on VGG-16 but since it is slower than real-time the rest of the paper focuses on our faster models.

Fastest DPM effectively speeds up DPM without sacrificing much mAP but it still misses real-time performance by a factor of 2 [38]. It also is limited by DPM's relatively low accuracy on detection compared to neural network approaches.

R-CNN minus R replaces Selective Search with static bounding box proposals [20]. While it is much faster than

VGG



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	<b>155</b>
YOLO	2007+2012	<b>63.4</b>	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

**Table 1: Real-Time Systems on PASCAL VOC 2007.** Comparing the performance and speed of fast detectors. Fast YOLO is the fastest detector on record for PASCAL VOC detection and is still twice as accurate as any other real-time detector. YOLO is 10 mAP more accurate than the fast version while still well above real-time in speed.

R-CNN, it still falls short of real-time and takes a significant accuracy hit from not having good proposals.

Fast R-CNN speeds up the classification stage of R-CNN but it still relies on selective search which can take around 2 seconds per image to generate bounding box proposals. Thus it has high mAP but at 0.5 fps it is still far from real-time.

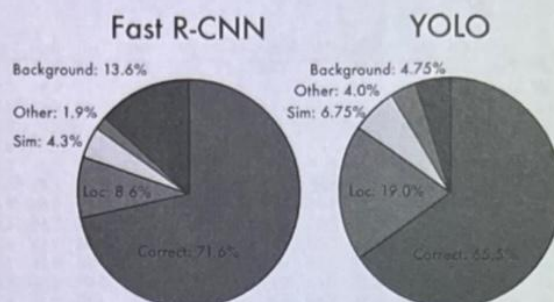
The recent Faster R-CNN replaces selective search with a neural network to propose bounding boxes, similar to Szegedy et al. [8] In our tests, their most accurate model achieves 7 fps while a smaller, less accurate one runs at 18 fps. The VGG-16 version of Faster R-CNN is 10 mAP higher but is also 6 times slower than YOLO. The Zeiler-Fergus Faster R-CNN is only 2.5 times slower than YOLO but is also less accurate.

#### 4.2. VOC 2007 Error Analysis

To further examine the differences between YOLO and state-of-the-art detectors, we look at a detailed breakdown of results on VOC 2007. We compare YOLO to Fast R-CNN since Fast R-CNN is one of the highest performing detectors on PASCAL and its detections are publicly available.

We use the methodology and tools of Hoiem et al. [19] For each category at test time we look at the top N predictions for that category. Each prediction is either correct or it is classified based on the type of error:

- Correct: correct class and  $\text{IOU} > .5$
- Localization: correct class,  $.1 < \text{IOU} < .5$
- Similar: class is similar,  $\text{IOU} > .1$



**Figure 4: Error Analysis: Fast R-CNN vs. YOLO** These charts show the percentage of localization and background errors in the top N detections for various categories ( $N = \#$  objects in that category).

- Other: class is wrong,  $\text{IOU} > .1$
- Background:  $\text{IOU} < .1$  for any object

Figure 4 shows the breakdown of each error type averaged across all 20 classes.

YOLO struggles to localize objects correctly. Localization errors account for more of YOLO's errors than all other sources combined. Fast R-CNN makes much fewer localization errors but far more background errors. 13.6% of its top detections are false positives that don't contain any objects. Fast R-CNN is almost 3x more likely to predict background detections than YOLO.

#### 4.3. Combining Fast R-CNN and YOLO

YOLO makes far fewer background mistakes than Fast R-CNN. By using YOLO to eliminate background detections from Fast R-CNN we get a significant boost in performance. For every bounding box that R-CNN predicts we check to see if YOLO predicts a similar box. If it does, we give that prediction a boost based on the probability predicted by YOLO and the overlap between the two boxes.

The best Fast R-CNN model achieves a mAP of 71.8% on the VOC 2007 test set. When combined with YOLO, its

	mAP	Combined	Gain
Fast R-CNN	71.8	-	-
Fast R-CNN (2007 data)	<b>66.9</b>	72.4	.6
Fast R-CNN (VGG-M)	59.2	72.4	.6
Fast R-CNN (CaffeNet)	57.1	72.1	.3
YOLO	63.4	<b>75.0</b>	<b>3.2</b>

**Table 2: Model combination experiments on VOC 2007.** We examine the effect of combining various models with the best version of Fast R-CNN. Other versions of Fast R-CNN provide only a small benefit while YOLO provides a significant performance boost.



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_JENS_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

**Table 3: PASCAL VOC 2012 Leaderboard.** YOLO compared with the full comp4 (outside data allowed) public leaderboard as of November 6th, 2015. Mean average precision and per-class average precision are shown for a variety of detection methods. YOLO is the only real-time detector. Fast R-CNN + YOLO is the forth highest scoring method, with a 2.3% boost over Fast R-CNN.

mAP increases by 3.2% to 75.0%. We also tried combining the top Fast R-CNN model with several other versions of Fast R-CNN. Those ensembles produced small increases in mAP between .3 and .6%, see Table 2 for details.

The boost from YOLO is not simply a byproduct of model ensembling since there is little benefit from combining different versions of Fast R-CNN. Rather, it is precisely because YOLO makes different kinds of mistakes at test time that it is so effective at boosting Fast R-CNN's performance.

Unfortunately, this combination doesn't benefit from the speed of YOLO since we run each model separately and then combine the results. However, since YOLO is so fast it doesn't add any significant computational time compared to Fast R-CNN.

#### 4.4. VOC 2012 Results

On the VOC 2012 test set, YOLO scores 57.9% mAP. This is lower than the current state of the art, closer to the original R-CNN using VGG-16, see Table 3. Our system struggles with small objects compared to its closest competitors. On categories like bottle, sheep, and tv/monitor YOLO scores 8-10% lower than R-CNN or Feature Edit. However, on other categories like cat and train YOLO achieves higher performance.

Our combined Fast R-CNN + YOLO model is one of the highest performing detection methods. Fast R-CNN gets a 2.3% improvement from the combination with YOLO, boosting it 5 spots up on the public leaderboard.

#### 4.5. Generalizability: Person Detection in Artwork

Academic datasets for object detection draw the training and testing data from the same distribution. In real-world applications it is hard to predict all possible use cases and

the test data can diverge from what the system has seen before [3]. We compare YOLO to other detection systems on the Picasso Dataset [12] and the People-Art Dataset [3], two datasets for testing person detection on artwork.

Figure 5 shows comparative performance between YOLO and other detection methods. For reference, we give VOC 2007 detection AP on person where all models are trained only on VOC 2007 data. On Picasso models are trained on VOC 2012 while on People-Art they are trained on VOC 2010.

R-CNN has high AP on VOC 2007. However, R-CNN drops off considerably when applied to artwork. R-CNN uses Selective Search for bounding box proposals which is tuned for natural images. The classifier step in R-CNN only sees small regions and needs good proposals.

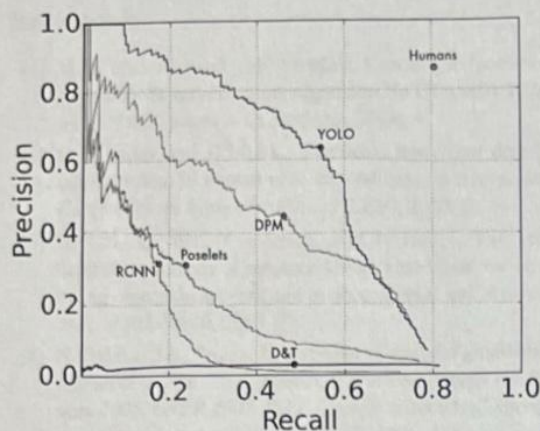
DPM maintains its AP well when applied to artwork. Prior work theorizes that DPM performs well because it has strong spatial models of the shape and layout of objects. Though DPM doesn't degrade as much as R-CNN, it starts from a lower AP.

YOLO has good performance on VOC 2007 and its AP degrades less than other methods when applied to artwork. Like DPM, YOLO models the size and shape of objects, as well as relationships between objects and where objects commonly appear. Artwork and natural images are very different on a pixel level but they are similar in terms of the size and shape of objects, thus YOLO can still predict good bounding boxes and detections.

#### 5. Real-Time Detection In The Wild

YOLO is a fast, accurate object detector, making it ideal for computer vision applications. We connect YOLO to a webcam and verify that it maintains real-time performance,





(a) Picasso Dataset precision-recall curves.

*Why? Yolo takes all entire image. big global perspective.*

	VOC 2007 AP	Picasso AP Best $F_1$	People-Art AP
<b>YOLO</b>	<b>59.2</b>	<b>53.3 0.590</b>	<b>45</b>
R-CNN	54.2	10.4 0.226	26
DPM	43.2	37.8 0.458	32
Poselets [2]	36.5	17.8 0.271	
D&T [4]	-	1.9 0.051	

(b) Quantitative results on the VOC 2007, Picasso, and People-Art Datasets. The Picasso Dataset evaluates on both AP and best  $F_1$  score.

Figure 5: Generalization results on Picasso and People-Art datasets.

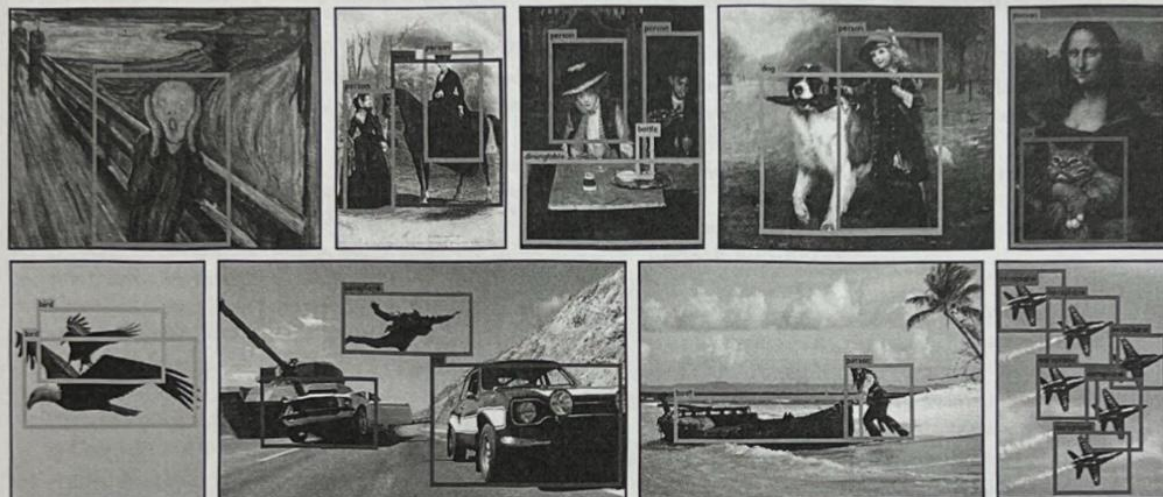


Figure 6: Qualitative Results. YOLO running on sample artwork and natural images from the internet. It is mostly accurate although it does think one person is an airplane.

including the time to fetch images from the camera and display the detections.

The resulting system is interactive and engaging. While YOLO processes images individually, when attached to a webcam it functions like a tracking system, detecting objects as they move around and change in appearance. A demo of the system and the source code can be found on our project website: <http://pjreddie.com/yolo/>.

## 6. Conclusion

We introduce YOLO, a unified model for object detection. Our model is simple to construct and can be trained

directly on full images. Unlike classifier-based approaches, YOLO is trained on a loss function that directly corresponds to detection performance and the entire model is trained jointly.

Fast YOLO is the fastest general-purpose object detector in the literature and YOLO pushes the state-of-the-art in real-time object detection. YOLO also generalizes well to new domains making it ideal for applications that rely on fast, robust object detection.

**Acknowledgements:** This work is partially supported by ONR N00014-13-1-0720, NSF IIS-1338054, and The Allen Distinguished Investigator Award.



# YOLOv3: An Incremental Improvement

Joseph Redmon   Ali Farhadi  
University of Washington

## Abstract

We present some updates to YOLO! We made a bunch of little design changes to make it better. We also trained this new network that's pretty swell. It's a little bigger than last time but more accurate. It's still fast though, don't worry. At  $320 \times 320$  YOLOv3 runs in 22 ms at 28.2 mAP, as accurate as SSD but three times faster. When we look at the old .5 IOU mAP detection metric YOLOv3 is quite good. It achieves 57.9 AP<sub>50</sub> in 51 ms on a Titan X, compared to 57.5 AP<sub>50</sub> in 198 ms by RetinaNet, similar performance but 3.8× faster. As always, all the code is online at <https://pjreddie.com/yolo/>.

## 1. Introduction

Sometimes you just kinda phone it in for a year, you know? I didn't do a whole lot of research this year. Spent a lot of time on Twitter. Played around with GANs a little. I had a little momentum left over from last year [12] [1]; I managed to make some improvements to YOLO. But, honestly, nothing like super interesting, just a bunch of small changes that make it better. I also helped out with other people's research a little.

Actually, that's what brings us here today. We have a camera-ready deadline [4] and we need to cite some of the random updates I made to YOLO but we don't have a source. So get ready for a TECH REPORT!

The great thing about tech reports is that they don't need intros, y'all know why we're here. So the end of this introduction will signpost for the rest of the paper. First we'll tell you what the deal is with YOLOv3. Then we'll tell you how we do. We'll also tell you about some things we tried that didn't work. Finally we'll contemplate what this all means.

## 2. The Deal

So here's the deal with YOLOv3: We mostly took good ideas from other people. We also trained a new classifier network that's better than the other ones. We'll just take you through the whole system from scratch so you can understand it all.

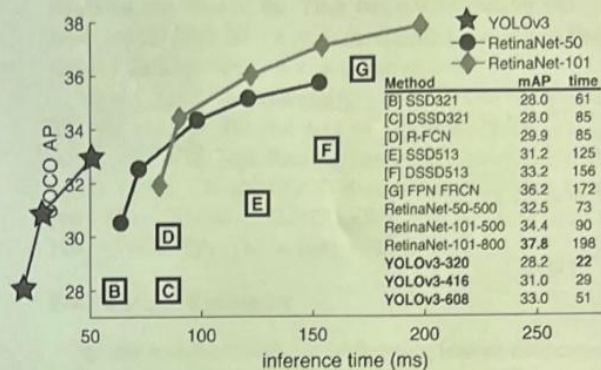


Figure 1. We adapt this figure from the Focal Loss paper [9]. YOLOv3 runs significantly faster than other detection methods with comparable performance. Times from either an M40 or Titan X, they are basically the same GPU.

## 2.1. Bounding Box Prediction

Following YOLO9000 our system predicts bounding boxes using dimension clusters as anchor boxes [15]. The network predicts 4 coordinates for each bounding box,  $t_x$ ,  $t_y$ ,  $t_w$ ,  $t_h$ . If the cell is offset from the top left corner of the image by  $(c_x, c_y)$  and the bounding box prior has width and height  $p_w, p_h$ , then the predictions correspond to:

$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$

During training we use sum of squared error loss. If the ground truth for some coordinate prediction is  $\hat{t}_*$  our gradient is the ground truth value (computed from the ground truth box) minus our prediction:  $\hat{t}_* - t_*$ . This ground truth value can be easily computed by inverting the equations above.

YOLOv3 predicts an objectness score for each bounding box using logistic regression. This should be 1 if the bounding box prior overlaps a ground truth object by more than any other bounding box prior. If the bounding box prior



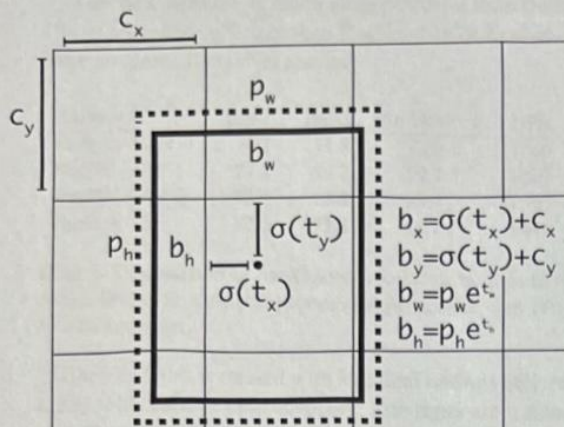


Figure 2. Bounding boxes with dimension priors and location prediction. We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function. This figure blatantly self-plagiarized from [15].

is not the best but does overlap a ground truth object by more than some threshold we ignore the prediction, following [17]. We use the threshold of .5. Unlike [17] our system only assigns one bounding box prior for each ground truth object. If a bounding box prior is not assigned to a ground truth object it incurs no loss for coordinate or class predictions, only objectness.

## 2.2. Class Prediction

Each box predicts the classes the bounding box may contain using multilabel classification. We do not use a softmax as we have found it is unnecessary for good performance, instead we simply use independent logistic classifiers. During training we use binary cross-entropy loss for the class predictions.

This formulation helps when we move to more complex domains like the Open Images Dataset [7]. In this dataset there are many overlapping labels (i.e. Woman and Person). Using a softmax imposes the assumption that each box has exactly one class which is often not the case. A multilabel approach better models the data.

## 2.3. Predictions Across Scales

YOLOv3 predicts boxes at 3 different scales. Our system extracts features from those scales using a similar concept to feature pyramid networks [8]. From our base feature extractor we add several convolutional layers. The last of these predicts a 3-d tensor encoding bounding box, objectness, and class predictions. In our experiments with COCO [10] we predict 3 boxes at each scale so the tensor is  $N \times N \times [3 * (4 + 1 + 80)]$  for the 4 bounding box offsets, 1 objectness prediction, and 80 class predictions.

Next we take the feature map from 2 layers previous and upsample it by  $2 \times$ . We also take a feature map from earlier in the network and merge it with our upsampled features using concatenation. This method allows us to get more meaningful semantic information from the upsampled features and finer-grained information from the earlier feature map. We then add a few more convolutional layers to process this combined feature map, and eventually predict a similar tensor, although now twice the size.

We perform the same design one more time to predict boxes for the final scale. Thus our predictions for the 3rd scale benefit from all the prior computation as well as fine-grained features from early on in the network.

We still use k-means clustering to determine our bounding box priors. We just sort of chose 9 clusters and 3 scales arbitrarily and then divide up the clusters evenly across scales. On the COCO dataset the 9 clusters were:  $(10 \times 13)$ ,  $(16 \times 30)$ ,  $(33 \times 23)$ ,  $(30 \times 61)$ ,  $(62 \times 45)$ ,  $(59 \times 119)$ ,  $(116 \times 90)$ ,  $(156 \times 198)$ ,  $(373 \times 326)$ .

## 2.4. Feature Extractor

We use a new network for performing feature extraction. Our new network is a hybrid approach between the network used in YOLOv2, Darknet-19, and that newfangled residual network stuff. Our network uses successive  $3 \times 3$  and  $1 \times 1$  convolutional layers but now has some shortcut connections as well and is significantly larger. It has 53 convolutional layers so we call it.... wait for it.... Darknet-53!

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

Table 1. Darknet-53.



This new network is much more powerful than Darknet-19 but still more efficient than ResNet-101 or ResNet-152. Here are some ImageNet results:

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

Table 2. Comparison of backbones. Accuracy, billions of operations, billion floating point operations per second, and FPS for various networks.

Each network is trained with identical settings and tested at  $256 \times 256$ , single crop accuracy. Run times are measured on a Titan X at  $256 \times 256$ . Thus Darknet-53 performs on par with state-of-the-art classifiers but with fewer floating point operations and more speed. Darknet-53 is better than ResNet-101 and  $1.5 \times$  faster. Darknet-53 has similar performance to ResNet-152 and is  $2 \times$  faster.

Darknet-53 also achieves the highest measured floating point operations per second. This means the network structure better utilizes the GPU, making it more efficient to evaluate and thus faster. That's mostly because ResNets have just way too many layers and aren't very efficient.

## 2.5. Training

We still train on full images with no hard negative mining or any of that stuff. We use multi-scale training, lots of data augmentation, batch normalization, all the standard stuff. We use the Darknet neural network framework for training and testing [14].

## 3. How We Do

YOLOv3 is pretty good! See table 3. In terms of COCOs weird average mean AP metric it is on par with the SSD variants but is  $3 \times$  faster. It is still quite a bit behind other

models like RetinaNet in this metric though.

However, when we look at the "old" detection metric of mAP at IOU = .5 (or AP<sub>50</sub> in the chart) YOLOv3 is very strong. It is almost on par with RetinaNet and far above the SSD variants. This indicates that YOLOv3 is a very strong detector that excels at producing decent boxes for objects. However, performance drops significantly as the IOU threshold increases indicating YOLOv3 struggles to get the boxes perfectly aligned with the object.

In the past YOLO struggled with small objects. However, now we see a reversal in that trend. With the new multi-scale predictions we see YOLOv3 has relatively high AP<sub>S</sub> performance. However, it has comparatively worse performance on medium and larger size objects. More investigation is needed to get to the bottom of this.

When we plot accuracy vs speed on the AP<sub>50</sub> metric (see figure 5) we see YOLOv3 has significant benefits over other detection systems. Namely, it's faster and better.

## 4. Things We Tried That Didn't Work

We tried lots of stuff while we were working on YOLOv3. A lot of it didn't work. Here's the stuff we can remember.

**Anchor box  $x, y$  offset predictions.** We tried using the normal anchor box prediction mechanism where you predict the  $x, y$  offset as a multiple of the box width or height using a linear activation. We found this formulation decreased model stability and didn't work very well.

**Linear  $x, y$  predictions instead of logistic.** We tried using a linear activation to directly predict the  $x, y$  offset instead of the logistic activation. This led to a couple point drop in mAP.

**Focal loss.** We tried using focal loss. It dropped our mAP about 2 points. YOLOv3 may already be robust to the problem focal loss is trying to solve because it has separate objectness predictions and conditional class predictions. Thus for most examples there is no loss from the class predictions? Or something? We aren't totally sure.

*focal loss delays of cross-entropy with dealing class imb.*

	backbone	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>
<i>Two-stage methods</i>							
Faster R-CNN++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	52.1
<i>One-stage methods</i>							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	40.8	61.1	44.1	24.1	44.2	51.2
YOLOv3 608 $\times$ 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Table 3. I'm seriously just stealing all these tables from [9] they take soooo long to make from scratch. Ok, YOLOv3 is doing alright. Keep in mind that RetinaNet has like  $3.8 \times$  longer to process an image. YOLOv3 is much better than SSD variants and comparable to state-of-the-art models on the AP<sub>50</sub> metric.



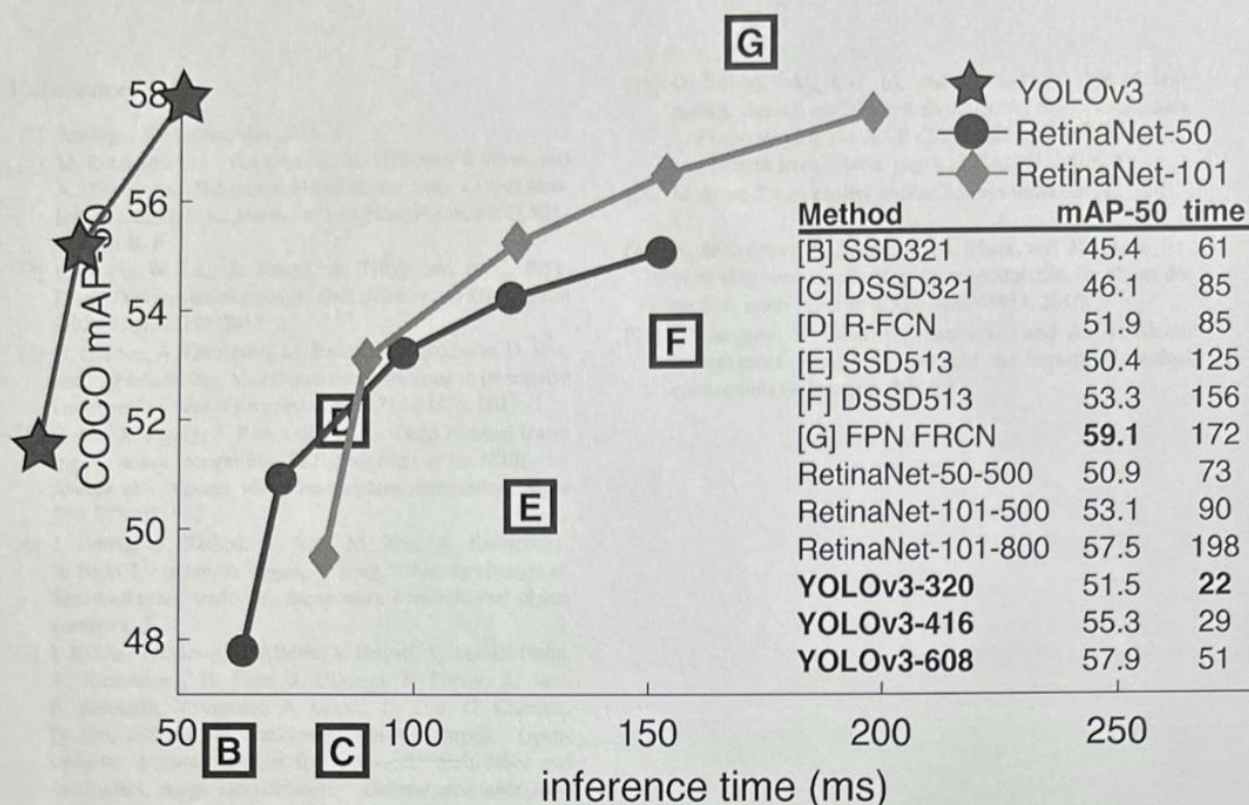


Figure 3. Again adapted from the [9], this time displaying speed/accuracy tradeoff on the mAP at .5 IOU metric. You can tell YOLOv3 is good because it's very high and far to the left. Can you cite your own paper? Guess who's going to try, this guy → [16]. Oh, I forgot, we also fix a data loading bug in YOLOv2, that helped by like 2 mAP. Just sneaking this in here to not throw off layout.

**Dual IOU thresholds and truth assignment.** Faster R-CNN uses two IOU thresholds during training. If a prediction overlaps the ground truth by .7 it is as a positive example, by [.3-.7] it is ignored, less than .3 for all ground truth objects it is a negative example. We tried a similar strategy but couldn't get good results.

We quite like our current formulation, it seems to be at a local optima at least. It is possible that some of these techniques could eventually produce good results, perhaps they just need some tuning to stabilize the training.

## 5. What This All Means

YOLOv3 is a good detector. It's fast, it's accurate. It's not as great on the COCO average AP between .5 and .95 IOU metric. But it's very good on the old detection metric of .5 IOU.

Why did we switch metrics anyway? The original COCO paper just has this cryptic sentence: "A full discussion of evaluation metrics will be added once the evaluation server is complete". Russakovsky et al report that that humans have a hard time distinguishing an IOU of .3 from .5! "Training humans to visually inspect a bounding box with IOU of 0.3 and distinguish it from one with IOU 0.5 is sur-

prisingly difficult." [18] If humans have a hard time telling the difference, how much does it matter?

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....<sup>1</sup>

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).

<sup>1</sup>The author is funded by the Office of Naval Research and Google.