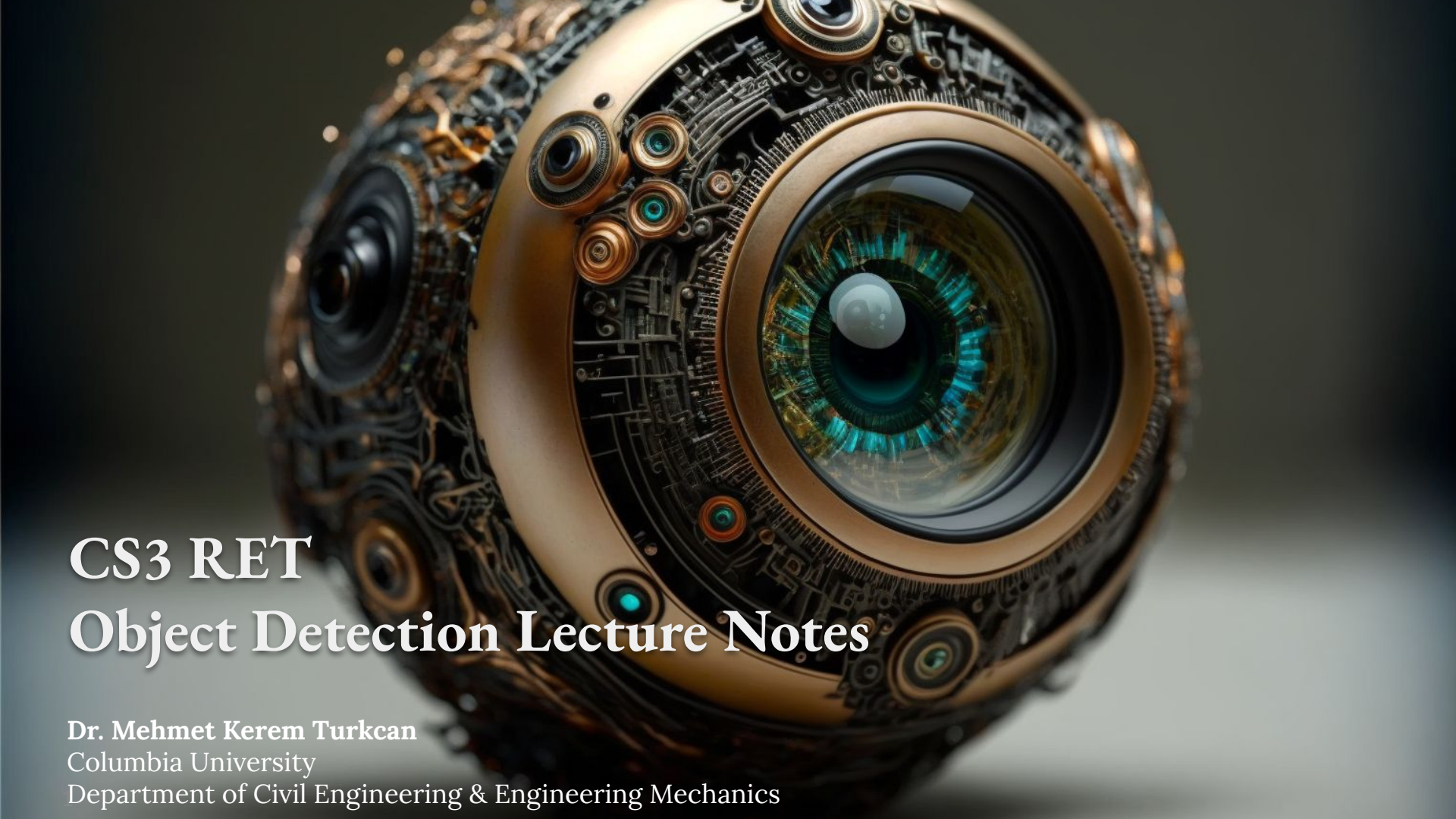


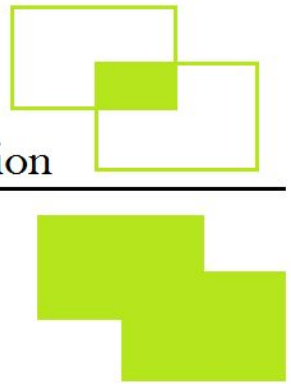
CS3 RET
Object Detection Lecture Notes

Dr. Mehmet Kerem Turkcan
Columbia University
Department of Civil Engineering & Engineering Mechanics


Object Detection
Review I

Intersection over Union (IoU)

$$IOU = \frac{\text{Area of Intersection}}{\text{Area of Union}}$$



Mean Average Precision (mAP)

mAP determines how well the model performs at accurately detecting objects for all classes. It is the most common metric used for evaluating object detection:

$$mAP = \frac{1}{N} \sum_{i}^{N} AP_i$$

Average Precision (AP)

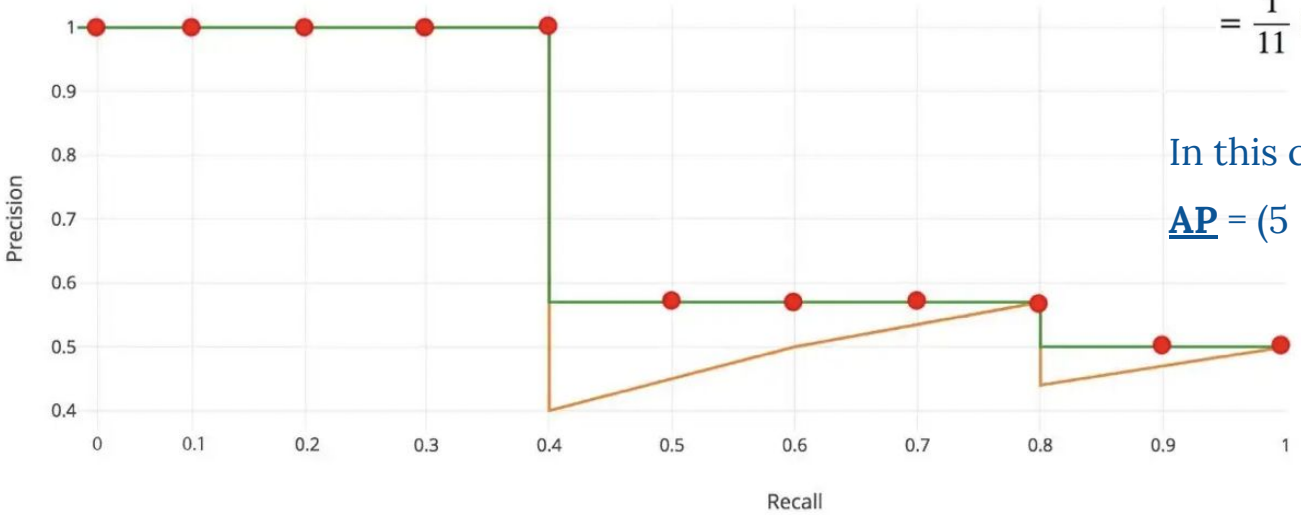
Before 2008 Pascal VOC,

interpolate the PR curve (with e.g. 11 points).

$$AP = \frac{1}{11} \sum_{r \in \{0.0, \ldots, 1.0\}} AP_r$$

$$= \frac{1}{11} \sum_{r \in \{0.0, \ldots, 1.0\}} p_{interp}(r)$$

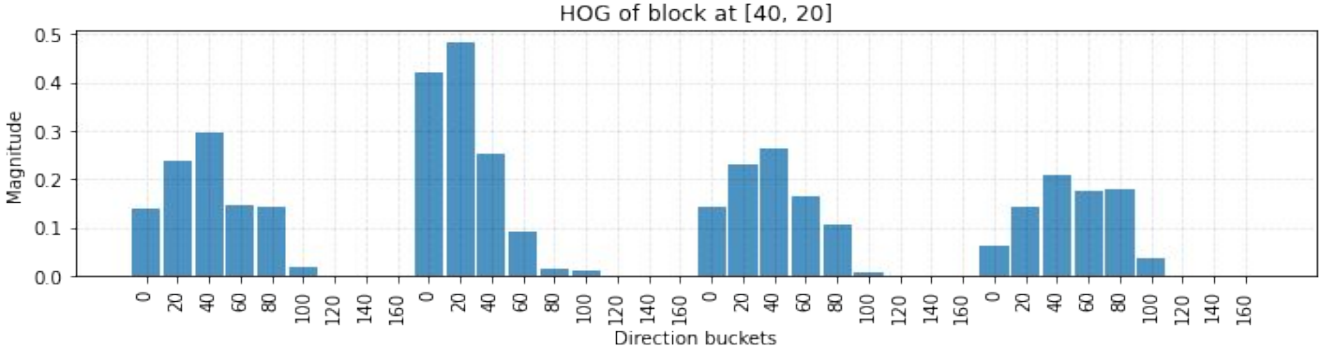
$$= \frac{1}{11} \times \left(AP_r(0) + AP_r(0.1) + \ldots + AP_r(1.0) \right)$$

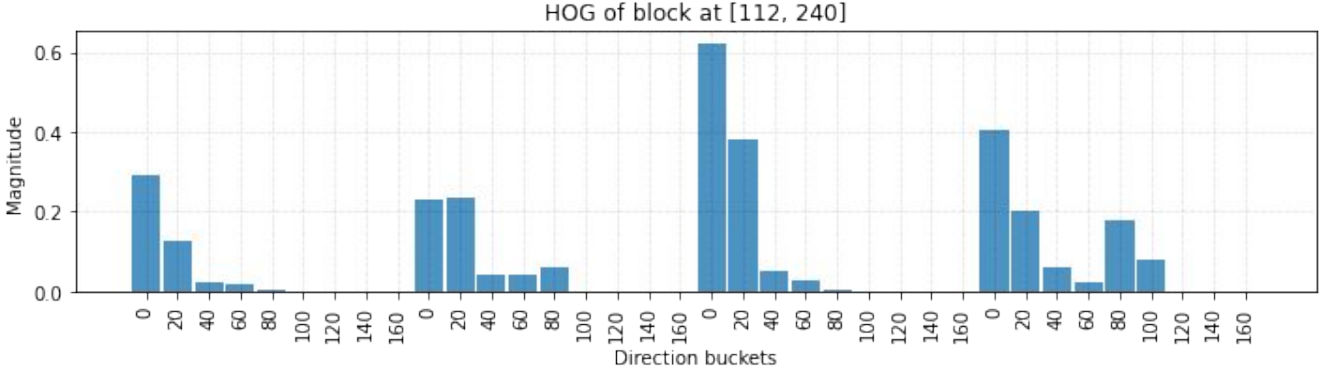


In this case,

AP = (5 × 1.0 + 4 × 0.57 + 2 × 0.5)/11

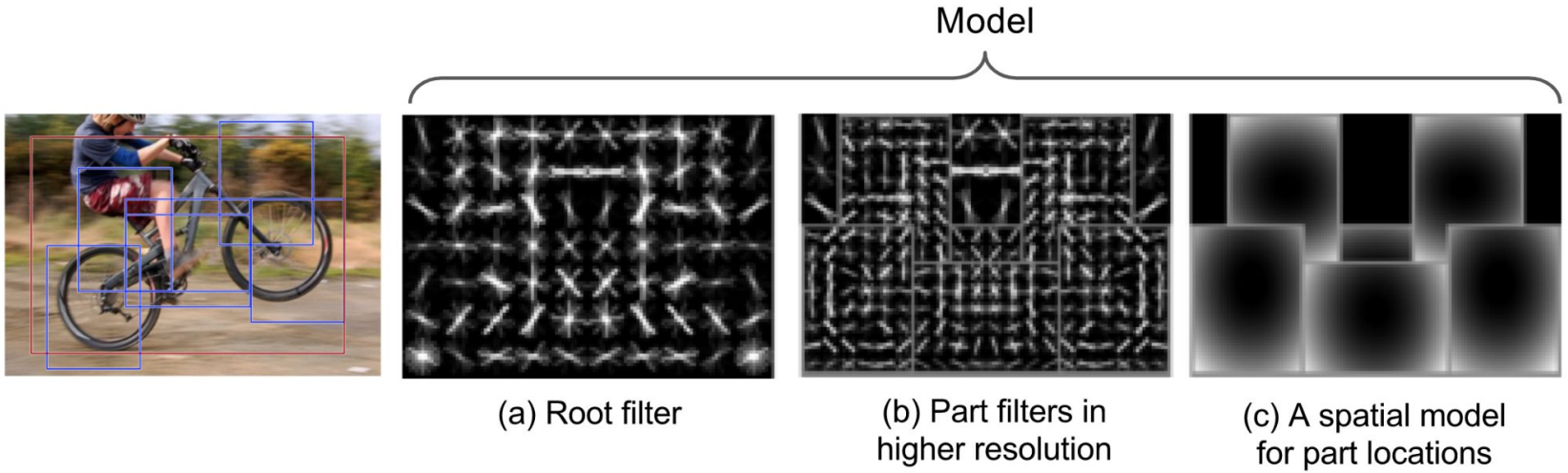
HOG


HOG of block at [40, 20]
Magnitude
0.0
0.1
0.2
0.3
0.4
0.5
0 20 40 60 80 100 120 140 160 0 20 40 60 80 100 120 140 160 0 20 40 60 80 100 120 140 160 0 20 40 60 80 100 120 140 160
Direction buckets


HOG of block at [112, 240]
Magnitude
0.0
0.2
0.4
0.6
0 20 40 60 80 100 120 140 160 0 20 40 60 80 100 120 140 160 0 20 40 60 80 100 120 140 160 0 20 40 60 80 100 120 140 160
Direction buckets

Deformable Parts Model (DPM)

- DPM has 3 major components:
  - A coarse root filter defines a detection window that approximately covers an entire object
  - Multiple part filters covering parts of the object, calculated at double resolution
  - A spatial model for scoring the locations of part filters relative to the root, provided with design



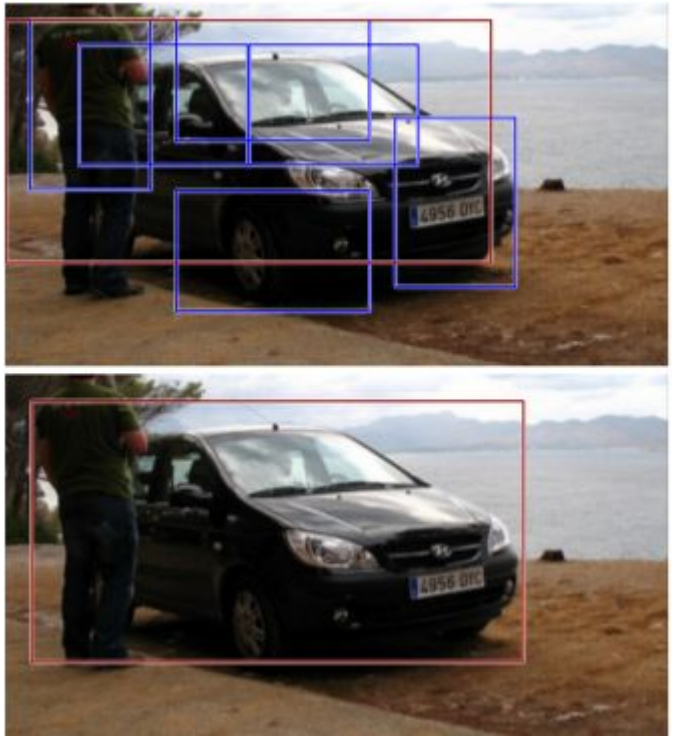
Non-Max Suppression

Non-Maximum Suppression finds the optimal bounding box, and suppresses all other bounding boxes.

Input: $B = \{b_1, .., b_N\}, S = \{s_1, .., s_N\}, Th$
B:list of detection boxes
S: corresponding detection scores
Th: NMS threshold
Begin:
  $D \leftarrow \emptyset$
  while $B \neq \emptyset$ **do**
    $m \leftarrow argmax\ S$
    $M \leftarrow b_m$
    $D \leftarrow D \cup M,\ \ B \leftarrow B - M$
    for b_i *in* B **do**
      if $iou(M, b_i) \geq Th$ **then**
        $B \leftarrow B - b_i;\ \ S \leftarrow S - s_i$
      end
    end
  end
  return D, S
end

Remove redundant boxes

NMS Algorithm Pseudocode



Overfeat: Almost-Free Sliding Window Implementation

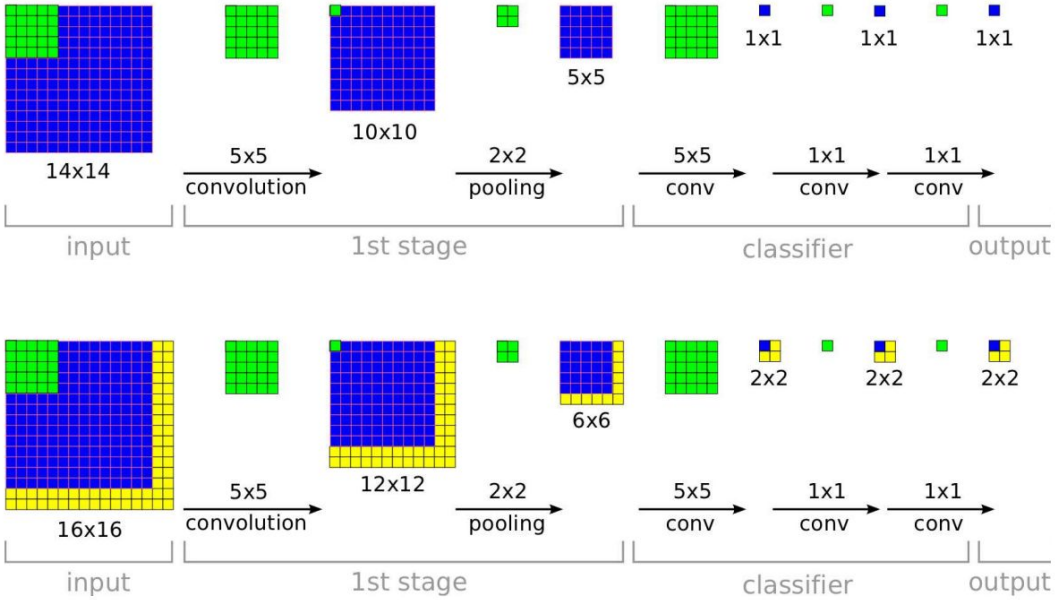
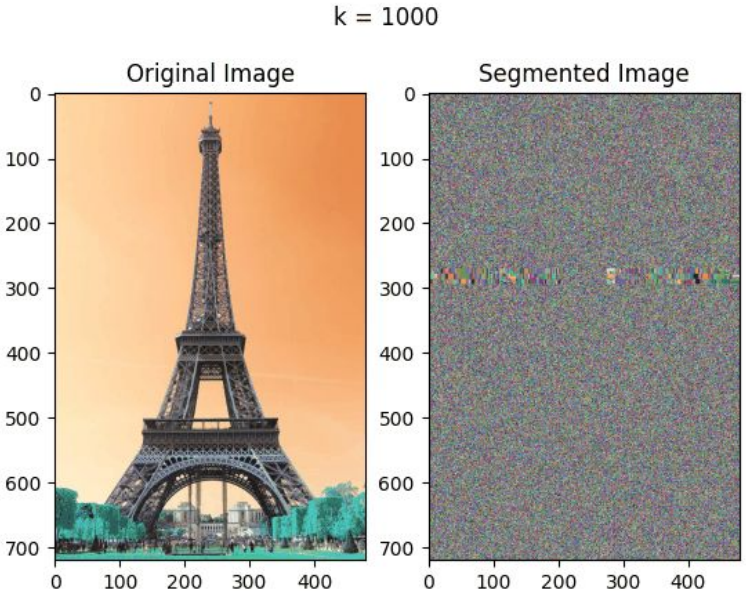


Figure 5: **The efficiency of ConvNets for detection.** During training, a ConvNet produces only a single spatial output (top). But when applied at test time over a larger image, it produces a spatial output map, e.g. 2x2 (bottom). Since all layers are applied convolutionally, the extra computation required for the larger image is limited to the yellow regions. This diagram omits the feature dimension for simplicity.

Segmentation: Evaluation Methods

- Pixel-wise Accuracy
- Mean Accuracy
- Mean IoU (Mean of class-wise region intersection over union)
- Frequency Weighted IU
- BF measure (Boundary F1-measure)

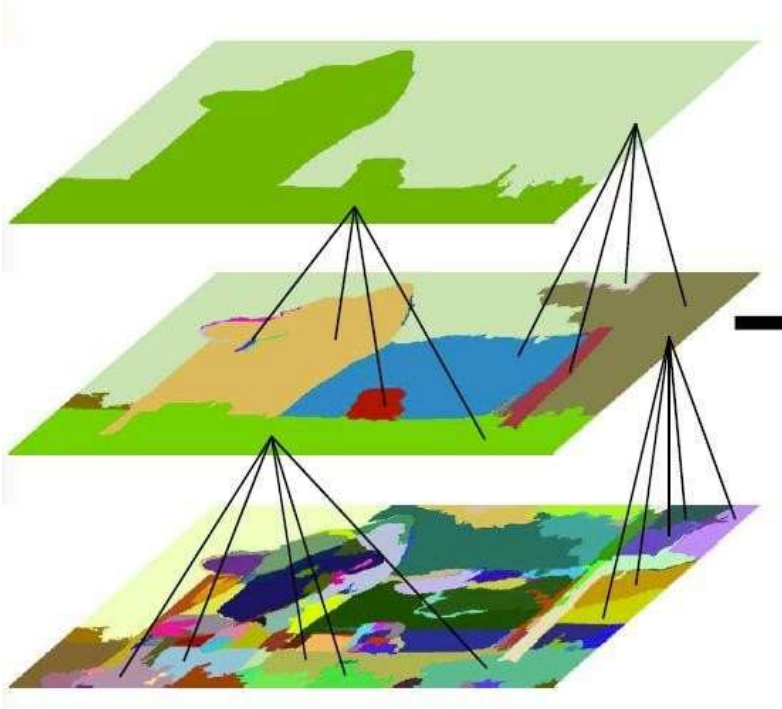
Image Segmentation with Felzenszwalb's Algorithm



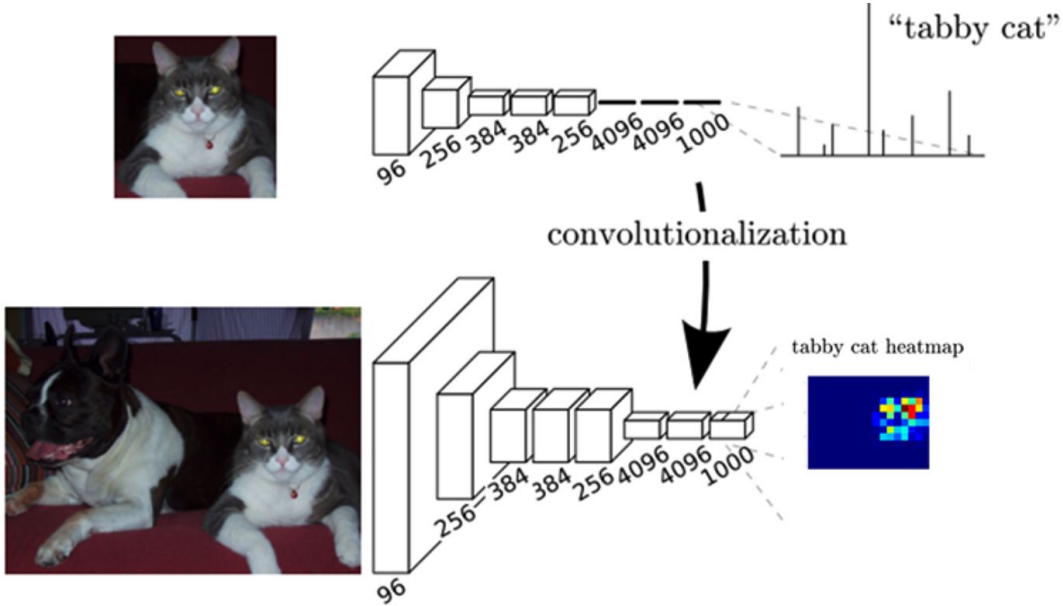
Selective Search

Starting with an initial segmentation (bottom),

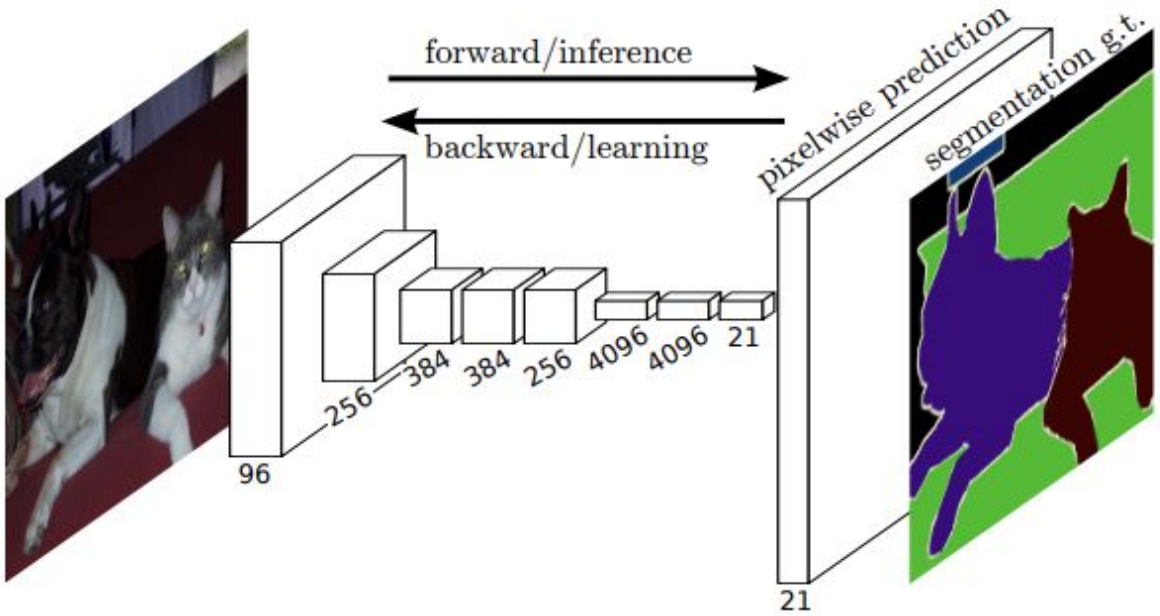
1. Merge two most similar regions based on similarities S.
   a. Color Similarity
   b. Texture Similarity
   c. Size Similarity
   d. Fill Similarity
2. Update similarities between the new region and its neighbors.
3. Go back to step 1. until the whole image is a single region.



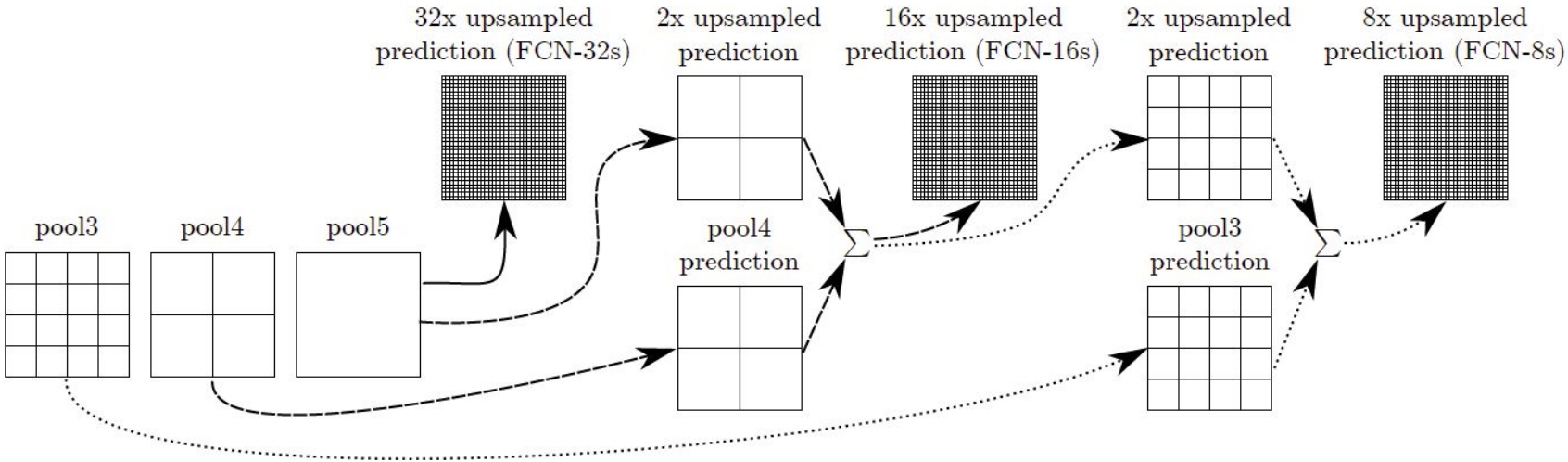
Fully Convolutional Networks (FCNs)



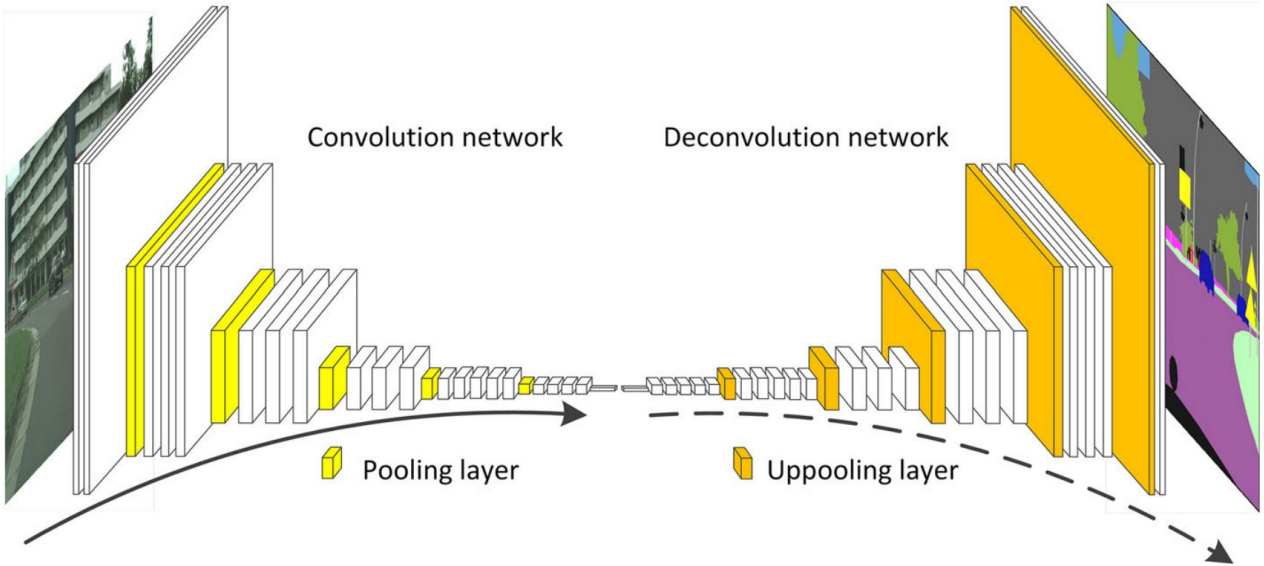
Fully Convolutional Networks (FCNs)



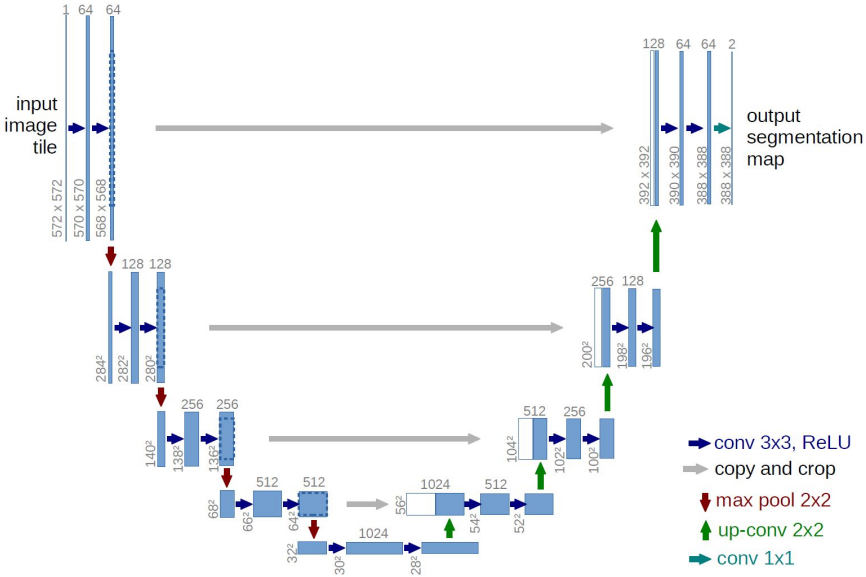
Fully Convolutional Networks (FCNs)



Deconvolution Networks



U-net


input image tile
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

UNet++

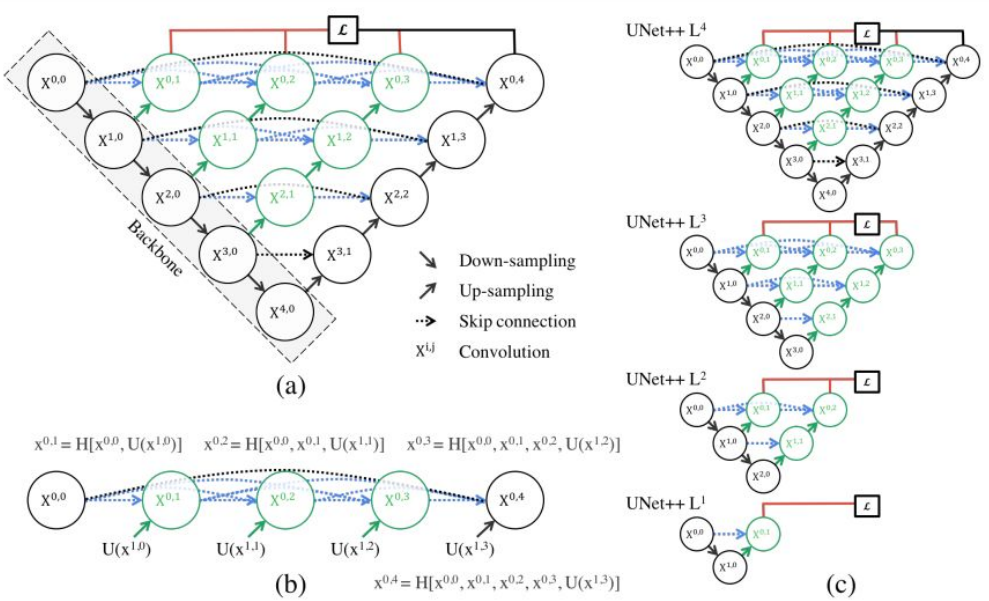
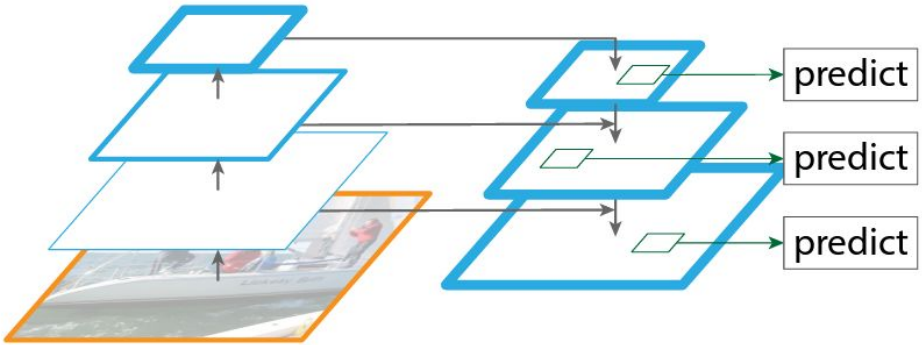


Fig. 1: (a) UNet++ consists of an encoder and decoder that are connected through a series of nested dense convolutional blocks. The main idea behind UNet++ is to bridge the semantic gap between the feature maps of the encoder and decoder prior to fusion. For example, the semantic gap between ($X^{0,0}$,$X^{1,3}$) is bridged using a dense convolution block with three convolution layers. In the graphical abstract, black indicates the original U-Net, green and blue show dense convolution blocks on the skip pathways, and red indicates deep supervision. Red, green, and blue components distinguish UNet++ from U-Net. (b) Detailed analysis of the first skip pathway of UNet++. (c) UNet++ can be pruned at inference time, if trained with deep supervision.

Feature Pyramid Networks



ReInspect

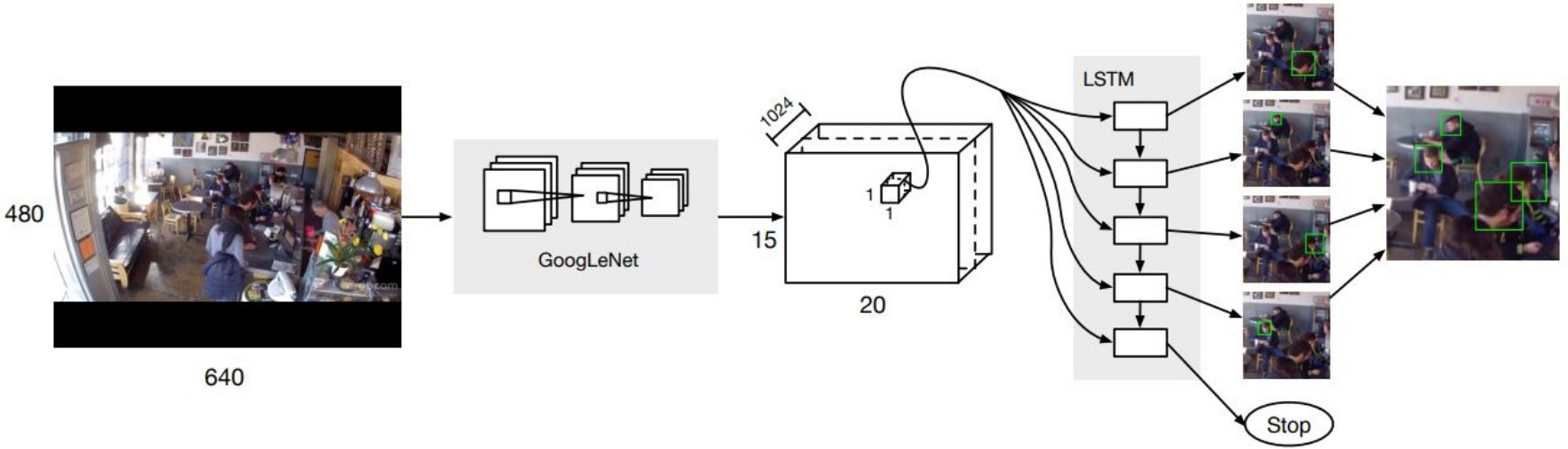
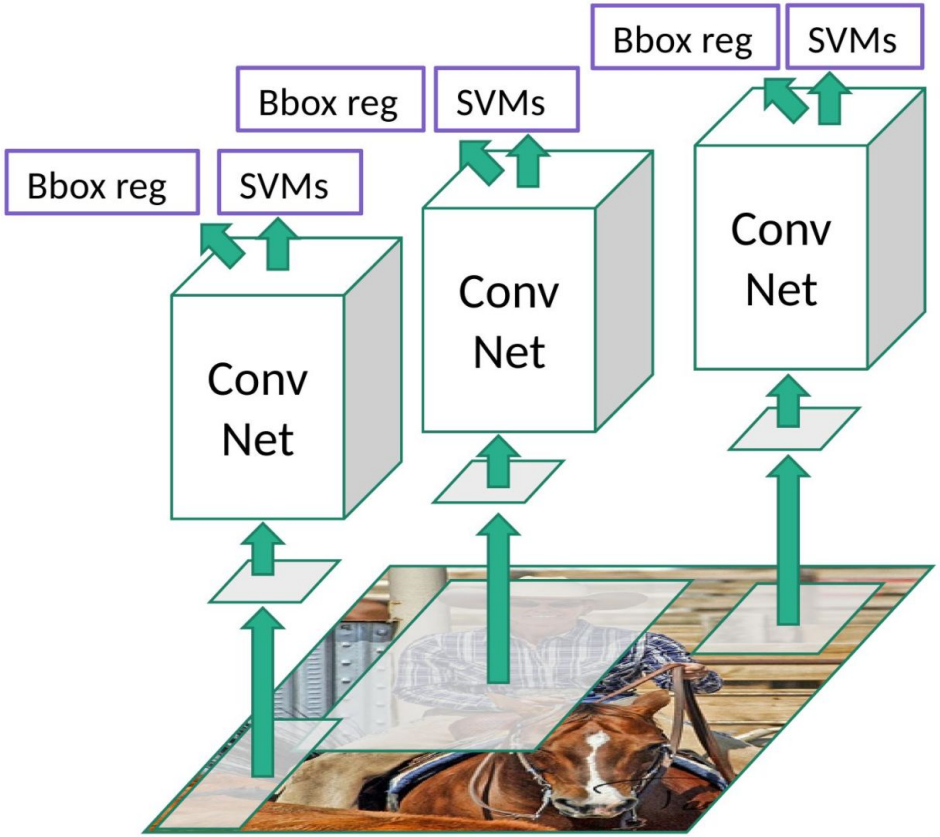
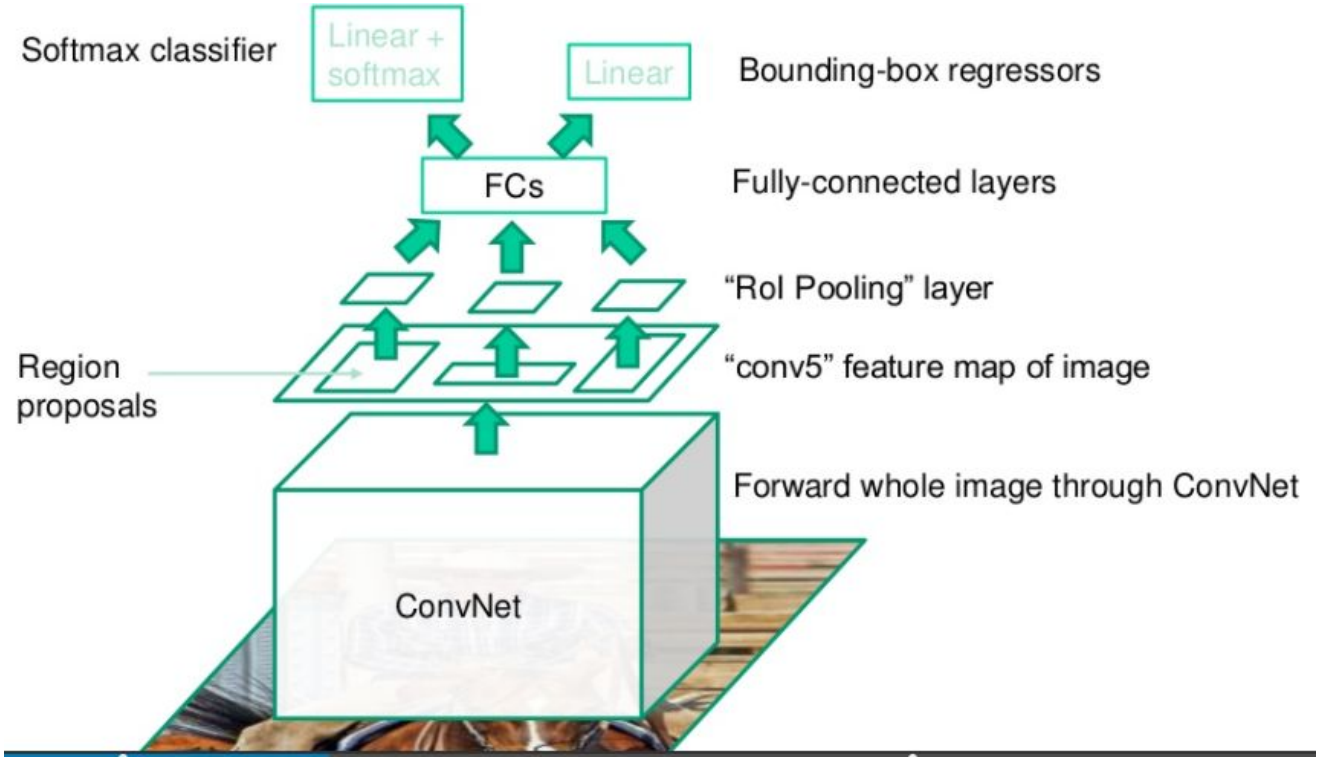


Figure 2: Our system first encodes an image into a block of high level features. An LSTM then acts as a controller, decoding this information into a set of detections.

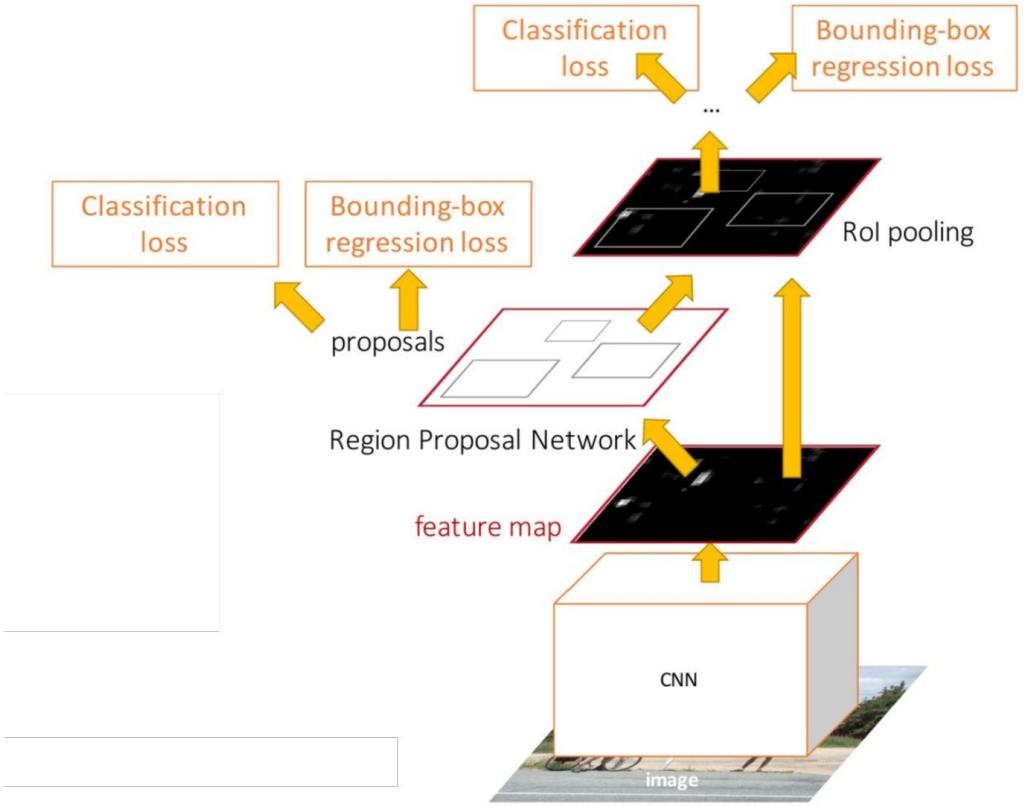
R-CNN Model


Bbox reg
SVMs
Bbox reg
SVMs
Bbox reg
SVMs
Conv Net
Conv Net
Conv Net

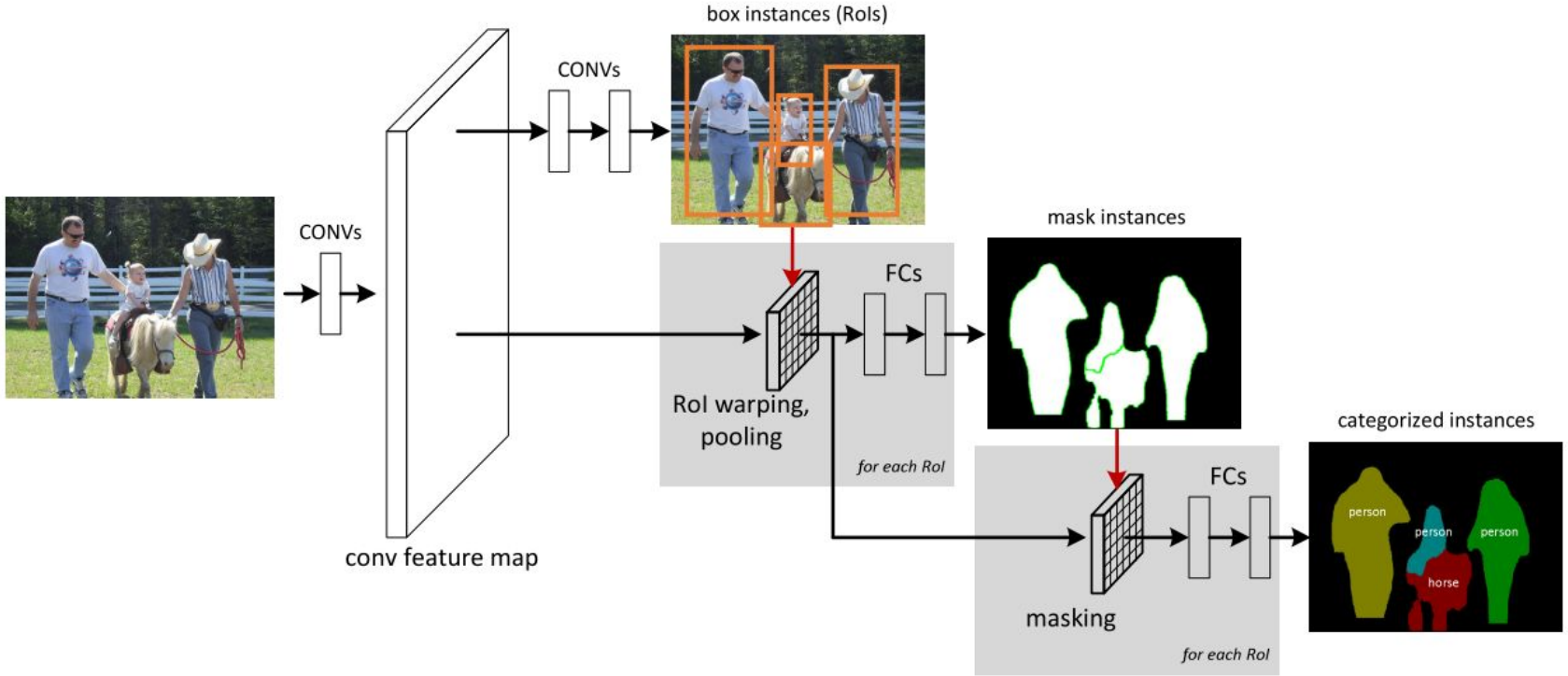
Fast R-CNN


Softmax classifier
Linear + softmax
Linear
Bounding-box regressors
FCs
Fully-connected layers
"RoI Pooling" layer
Region proposals
"conv5" feature map of image
Forward whole image through ConvNet
ConvNet

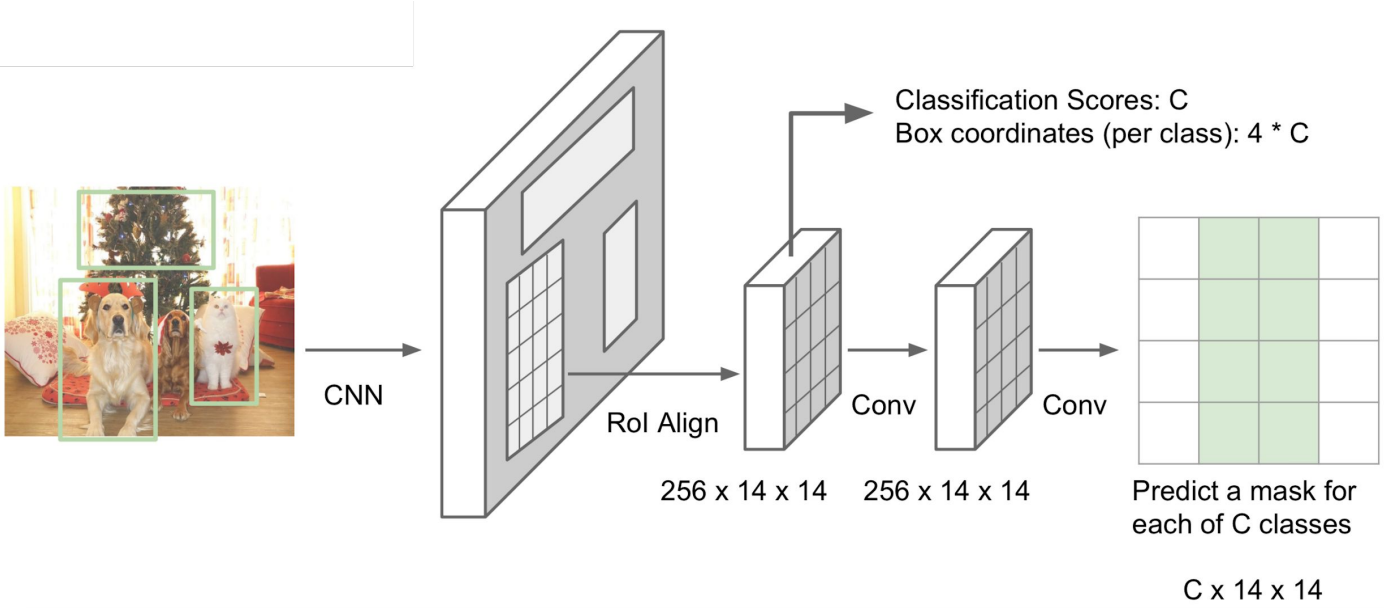
Faster R-CNN


Classification loss
Bounding-box regression loss
...
RoI pooling
Classification loss
Bounding-box regression loss
proposals
Region Proposal Network
feature map
CNN
image

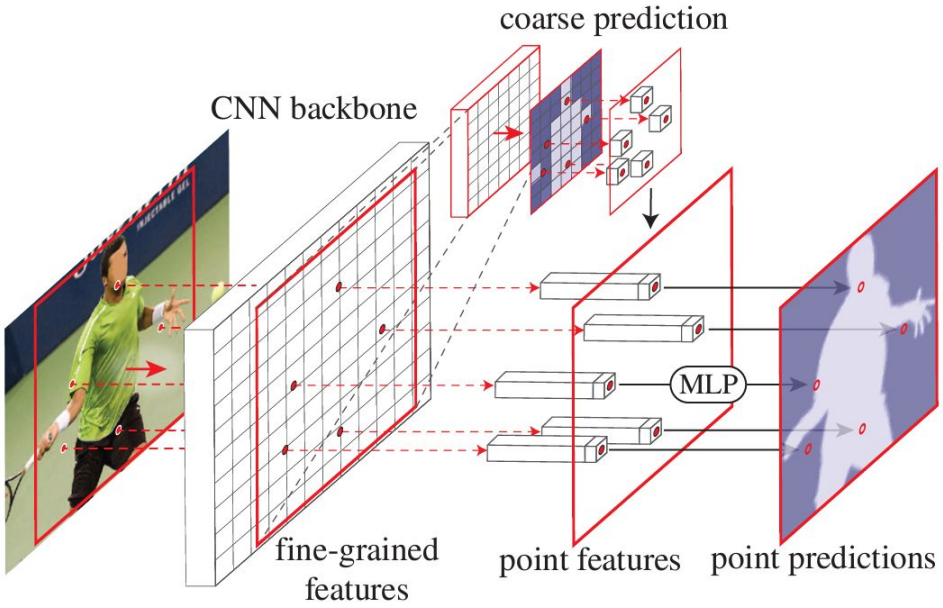
Multi-task Network Cascades for Instance-Aware Segmentation



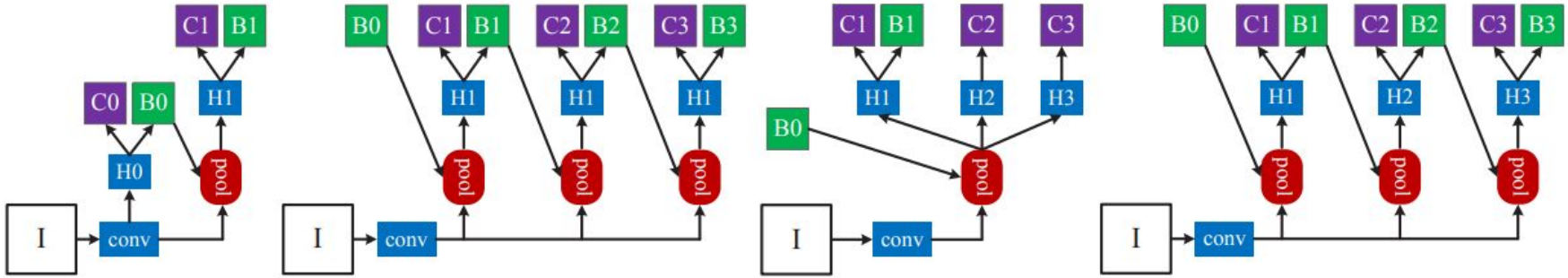
Mask R-CNN


Classification Scores: C
Box coordinates (per class): 4 * C
CNN
RoI Align
Conv
Conv
256 x 14 x 14
256 x 14 x 14
Predict a mask for each of C classes
C x 14 x 14

PointRend


coarse prediction
CNN backbone
MLP
fine-grained features
point features
point predictions

Cascade R-CNN



(a) Faster R-CNN (b) Iterative BBox at inference (c) Integral Loss (d) Cascade R-CNN

Figure 3. The architectures of different frameworks. "I" is input image, "conv" backbone convolutions, "pool" region-wise feature extraction, "H" network head, "B" bounding box, and "C" classification. "B0" is proposals in all architectures.

Object Detection

Single Shot MultiBox Detector (SSD)

Object Detection Outline

Fundamentals of Object Detection

Evaluation Criteria for Object Detection

Conventional Object Detection Methods

- Histogram of Gradients (HOG)
- Deformable Parts Model (DPM)
- Overfeat

Segmentation

- Semantic Image Segmentation
- Instance Segmentation

Evaluation Criteria for Semantic Segmentation

Early Segmentation Approaches

Fully Convolutional Networks (FCNs)

UNet Segmentation

Region Based Convolutional Neural Networks (R-CNN)

Fast R-CNN

Faster R-CNN

Multi-Task Network Cascades

Mask R-CNN Instance Segmentation

Single Shot MultiBox Detector (SSD) Single-Stage Detector

YOLO (You Only Look Once) Single Stage Detector

RetinaNet

Single Shot Detector (SSD)

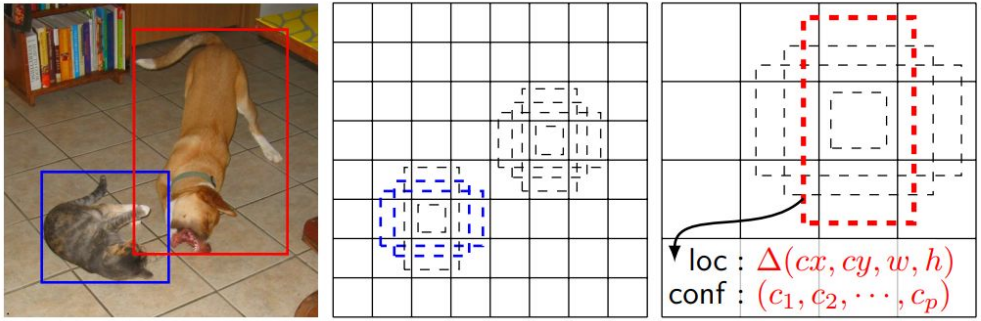
SSD is a **single stage detector:**

- There is **no** bounding box proposal stage
- Contains multi-scale prediction(s) which use predictors (filters) at different points in the network

This results in:

- **Significant speed improvement** over two stage detectors, since there is no proposal stage:
  - Achieves **59 FPS** on VOC 2007 dataset
- Accuracy on VOC2007: **74.3% mAP** (compared to 63.4% mAP for YOLO at 45 FPS)

SSD: Single Shot MultiBox Detector



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Fig. 1: **SSD framework.** (a) SSD only needs an input image and ground truth boxes for each object during training. In a convolutional fashion, we evaluate a small set (e.g. 4) of default boxes of different aspect ratios at each location in several feature maps with different scales (e.g. 8×8 and 4×4 in (b) and (c)). For each default box, we predict both the shape offsets and the confidences for all object categories ($(c_1, c_2, \cdots, c_p)$). At training time, we first match these default boxes to the ground truth boxes. For example, we have matched two default boxes with the cat and one with the dog, which are treated as positives and the rest as negatives. The model loss is a weighted sum between localization loss (e.g. Smooth L1 [6]) and confidence loss (e.g. Softmax).

SSD: Single Shot MultiBox Detector

Set of default boxes over different **scales**:

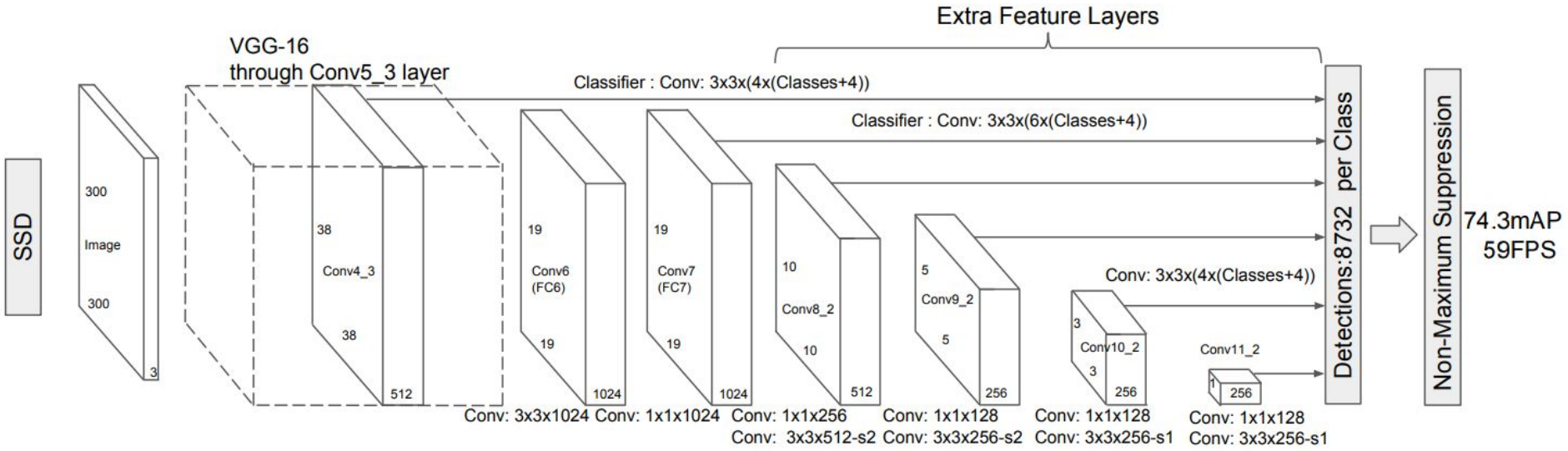
- We have different feature maps corresponding to each different scale;
- We combine predictions resulting from feature maps of multiple different resolutions

SSD Components

The core of SSD is predicting category scores and box offsets for a fixed set of default bounding boxes, using small convolutional filters applied to feature maps:

- To achieve high detection accuracy they produce predictions of different scales from feature maps of different scales, and explicitly separate predictions by aspect ratios
- SSD can achieve high accuracy on low resolution images, thereby reducing the computation cost and increasing the fps

SSD Architecture

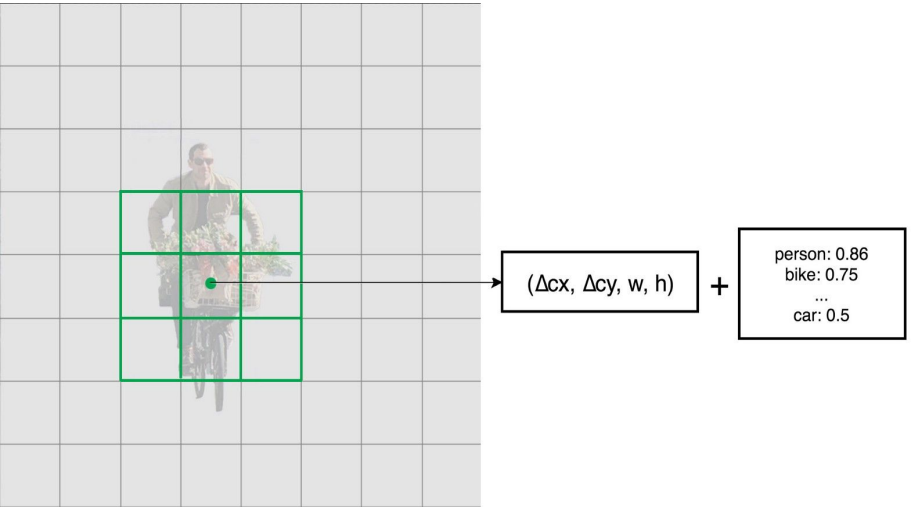


SSD Multi-Scale Feature Maps for Detection

- Convolutional feature layers are added to the end of the truncated base network
- These layers decrease the feature map size progressively and allow predictions of detections at multiple scales

SSD Convolutional Predictors for Detection

- At each convolution layer, a convolution filter is applied to get the predictions
- For a $m \times n \times p$ feature map, a $3 \times 3 \times p \times (k(c+4))$ convolutional filter is applied
- The output size for a $m \times n$ feature map is $(c+4)kmn$.



$$(38 \times 38 \times 512) \xrightarrow{(4\times3\times3\times512\times(21+4))} (38 \times 38 \times 4 \times (21+4))$$

SSD Training

Matching strategy:

For each ground truth box, default boxes that vary over location, aspect ratio, and scale are selected:

- This is done by first matching each ground truth box to that default box which has the best Jaccard overlap (IoU) with the ground truth box;
- Then, default boxes are matched to any ground truth with Jaccard overlap (IOU) higher than a threshold (0.5).

SSD Training Objective (Loss)

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^k \text{smooth}_{\text{L1}}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) \qquad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right)$$

$\mathbf{x}_{ij}^k$: Indicator for matching the i-th default box to the j-th ground truth box of category p

The confidence loss is the softmax loss over multiple classes confidences (c).

$$L_{conf}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^p log(\hat{c}_i^p) - \sum_{i \in Neg} log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

and the weight term α is set to 1 by cross validation.

SSD: Choosing Scales and Aspect Ratios of Default Boxes

$$s_k = s_{\min} + \frac{s_{\max} - s_{\min}}{m - 1}(k - 1), \quad k \in [1, m]$$

$$w = scale \cdot \sqrt{\text{aspect ratio}}$$

$$h = \frac{scale}{\sqrt{\text{aspect ratio}}}$$

Then SSD adds an extra default box with scale:

$$scale = \sqrt{\text{scale} \cdot \text{scale at next level}}$$

and aspect ratio = 1.

SSD Hard Negative Mining

- After the matching step, most of the default boxes are negatives, especially when the number of possible default boxes is large
- This introduces a significant imbalance between the positive and negative training examples
- Instead of using all negative examples, the negative samples are sorted using the highest confidence loss for each default box, and the top ones are picked so that the ratio between the negatives and positives is at most 3:1
- This leads to faster optimization and a more stable training

Datasets for Comparison

Name	Type	Categories	Image size	Images per category	Objects per image
PASCAL VOC (2007, 2012)	Common objects	20	470×380	303~4087	2.4
MS COCO	Common objects	91	640×480	-	7.3
ImageNet (ILSVRC)	Common objects	21	500×400	-	1.5
KITTI	Driving scene	8	1242×375	-*	-*
Cityscape	Driving scene	8	2048×1024	-*	-*
...	...	...			...

For driving scene datasets, they are continuous frames. And they don't provide stats of images per category and objects per image.

SSD vs. YOLO and R-CNN Results

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ 1000 × 600
Fast YOLO	52.7	155	1	98	448 × 448
YOLO (VGG16)	66.4	21	1	98	448 × 448
SSD300	74.3	46	1	8732	300 × 300
SSD512	76.8	19	1	24564	512 × 512
SSD300	74.3	59	8	8732	300 × 300
SSD512	76.8	22	8	24564	512 × 512

Results

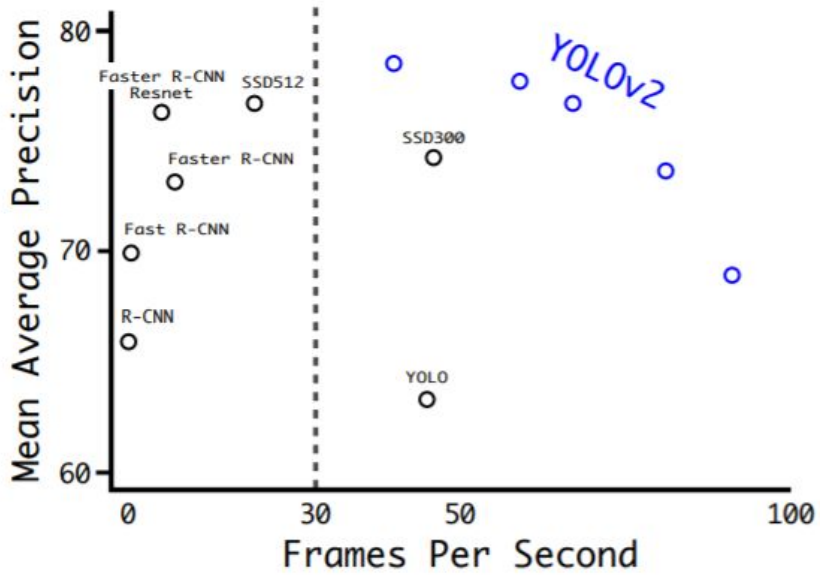


Figure 4: Accuracy and speed on VOC 2007.

Object Detection Chapter 19:

YOLO (You Only Look Once) Single Stage Detector

Object Detection Outline

Fundamentals of Object Detection

Evaluation Criteria for Object Detection

Conventional Object Detection Methods

- Histogram of Gradients (HOG)
- Deformable Parts Model (DPM)
- Overfeat

Segmentation

- Semantic Image Segmentation
- Instance Segmentation

Evaluation Criteria for Semantic Segmentation

Early Segmentation Approaches

Fully Convolutional Networks (FCNs)

UNet Segmentation

Region Based Convolutional Neural Networks (R-CNN)

Fast R-CNN

Faster R-CNN

Multi-Task Network Cascades

Mask R-CNN Instance Segmentation

Single Shot MultiBox Detector (SSD) Single-Stage Detector

YOLO (You Only Look Once) Single Stage Detector

RetinaNet

YOLOv1 Salient Features

YOLO is a **Single Stage Detector:**

- A single neural network predicts bounding boxes and class probabilities from full images in one evaluation step
- Very fast - reaching up to 45 fps
- Makes more localization errors, but is less prone to false positives on background
- Since it sees the entire image, it implicitly encodes the contextual information

Introduction to YOLO: Unified, Real-Time Object Detection

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.

Key steps:

- resize images, run neural network, perform non-max suppression.

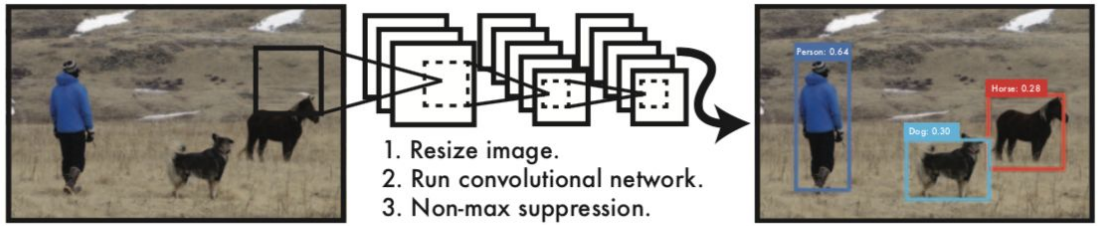
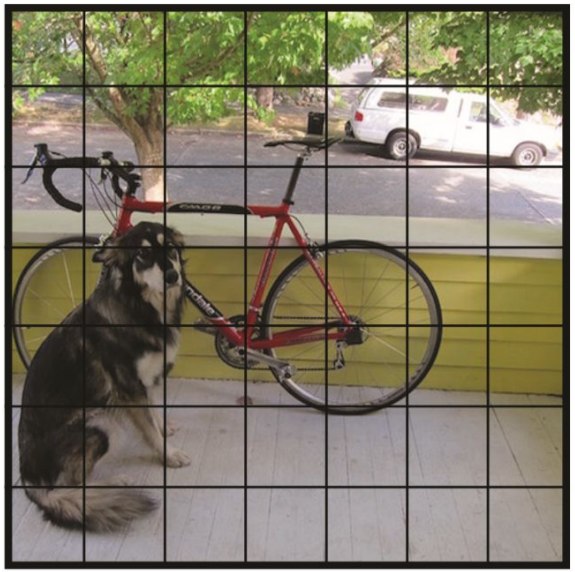


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448×448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

YOLO Unified Detection

- Divide the image into a S x S grid, for example 7 x 7 cells for evaluating PASCAL VOC which has 20 classes.



S × S grid on input

YOLO Unified detection

Each grid produce B (2 in this experiment) bounding boxes.

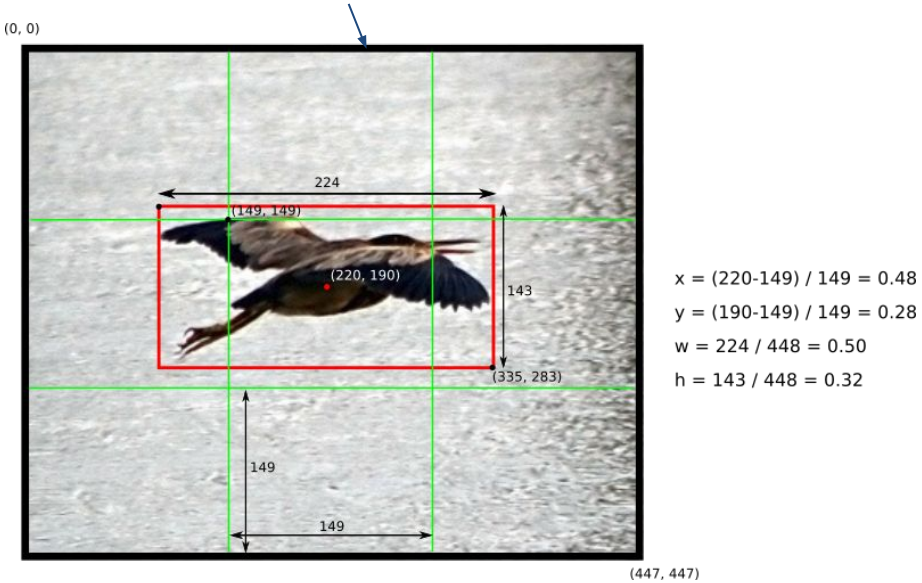
Each bounding box has 5 components (x, y, w, h, c)

where c is confidence $\Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}}$.

1 for object, 0 for no object

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.

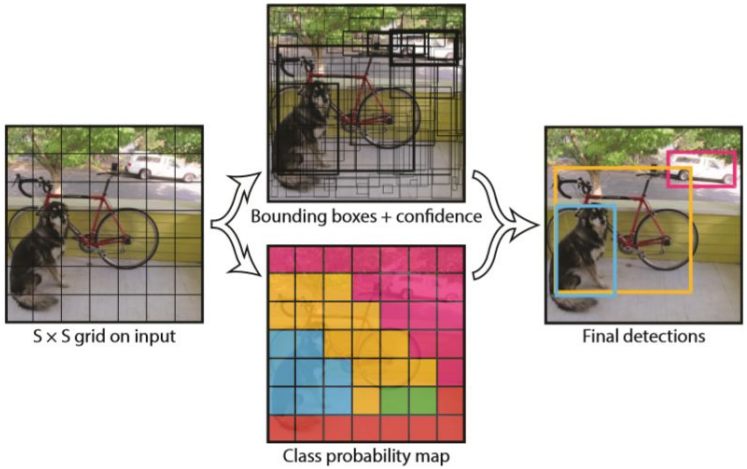
Grid cells are green, bounding box is red



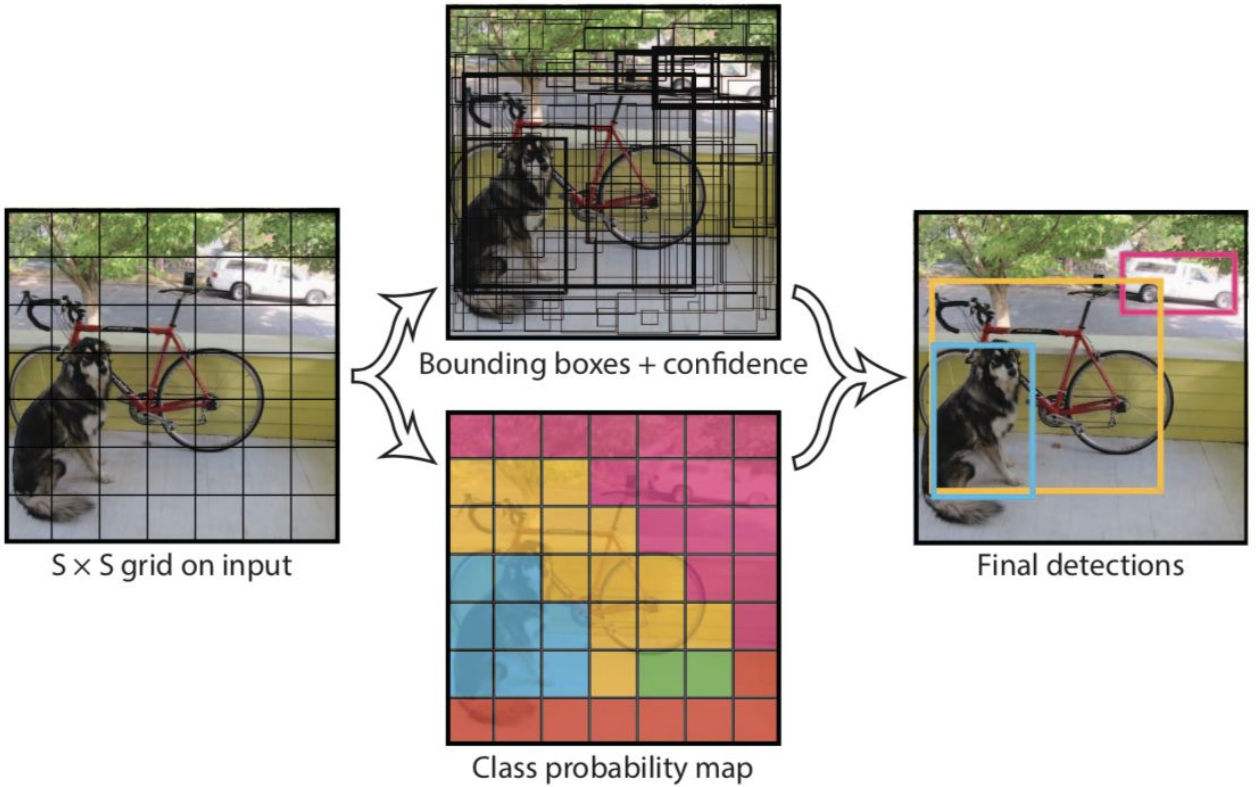
https://hackernoon.com/understanding-yolo-f5a74bbc7967

YOLO Unified Detection

- Divide the image into a grid of 7 x 7 cells, where each cell predicts 2 bounding boxes and 20 class probabilities. Each bounding box has 5 descriptive components.
- 7x7x(2x5+20) = 7x7x30 = 1470 is the output size of CNN before non-max suppression.



YOLO Processing Steps



YOLO Processing Steps

- Divide the input image into a grid with S x S cells
  - Each grid cell predicts B bounding boxes and confidence scores for boxes
  - Scores = $\Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}}$
- **Each bounding box** consists of 5 predictions: x, y, w, h and confidence
- **Each cell** predicts C conditional class probabilities
  - $\Pr(\text{Class}_i|\text{Object})$
- We can then multiply the conditional class probabilities and the individual box confidence predictions to predict class probability:
  - $\Pr(\text{Class}_i|\text{Object}) * \Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}}$
- All predictions are thus encoded with a tensor of size
  - $S \times S \times (B * 5 + C)$

Confidence Score

In context of YOLO, the Confidence Score is formally defined as:

- *Pr(Object) * IoU*
  - **If no object exists in that cell**, we want this score to be 0
  - **Otherwise,** we want it to be equal to the *IoU* between the predicted box and the groundtruth

What Does the YOLO Network Predict?

- Each bounding box contains 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 $(0 < x, y < 1)$
  - The width (w) and height (h) are predicted relative to the entire image $(0 < w, h < 1)$
  - The confidence represents the IOU between the predicted box and the ground truth box
  - Each grid cell, in addition to previous, predicts C conditional class probabilities, Pr(Classi|Object)

Important: the model predicts only one set of class probabilities per grid cell regardless of the number of boxes B

YOLO Network design

- Inspired by GoogLeNet

YOLO Network Architecture

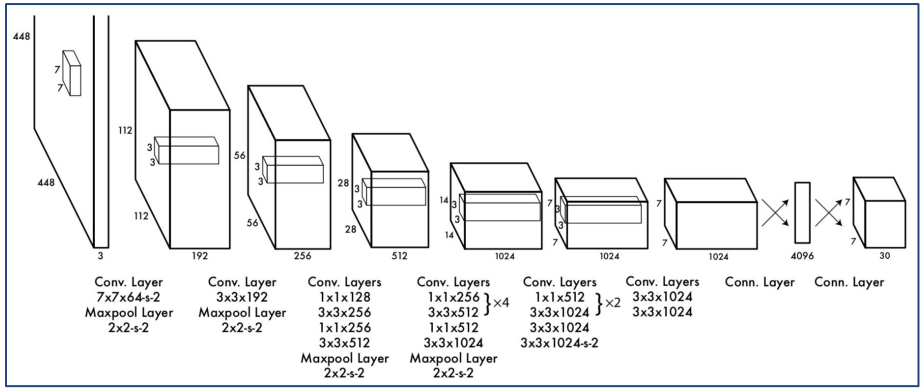
Architecture inspired by GoogLeNet.

Detection requires fine-grained information, so increase resolution from 224*x*224 to 448*x*448 pixels.

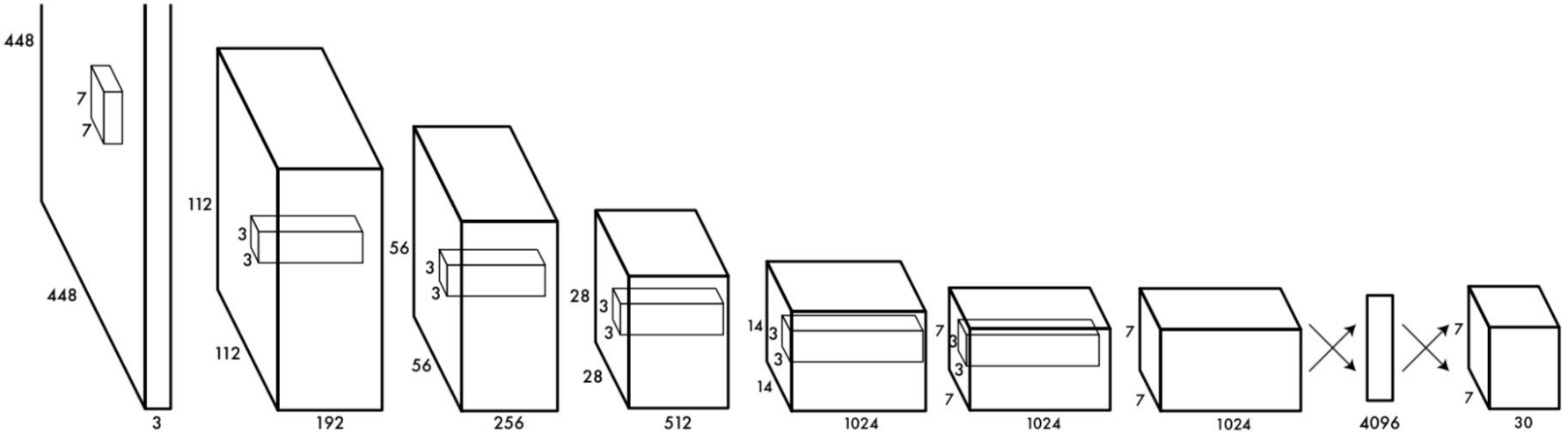
Consists of 24 convolutional layers followed by 2 fully connected layers.

Does not use inception modules but 1 *x* 1 reduction layers followed by 3 *x* 3 convolutional layers.

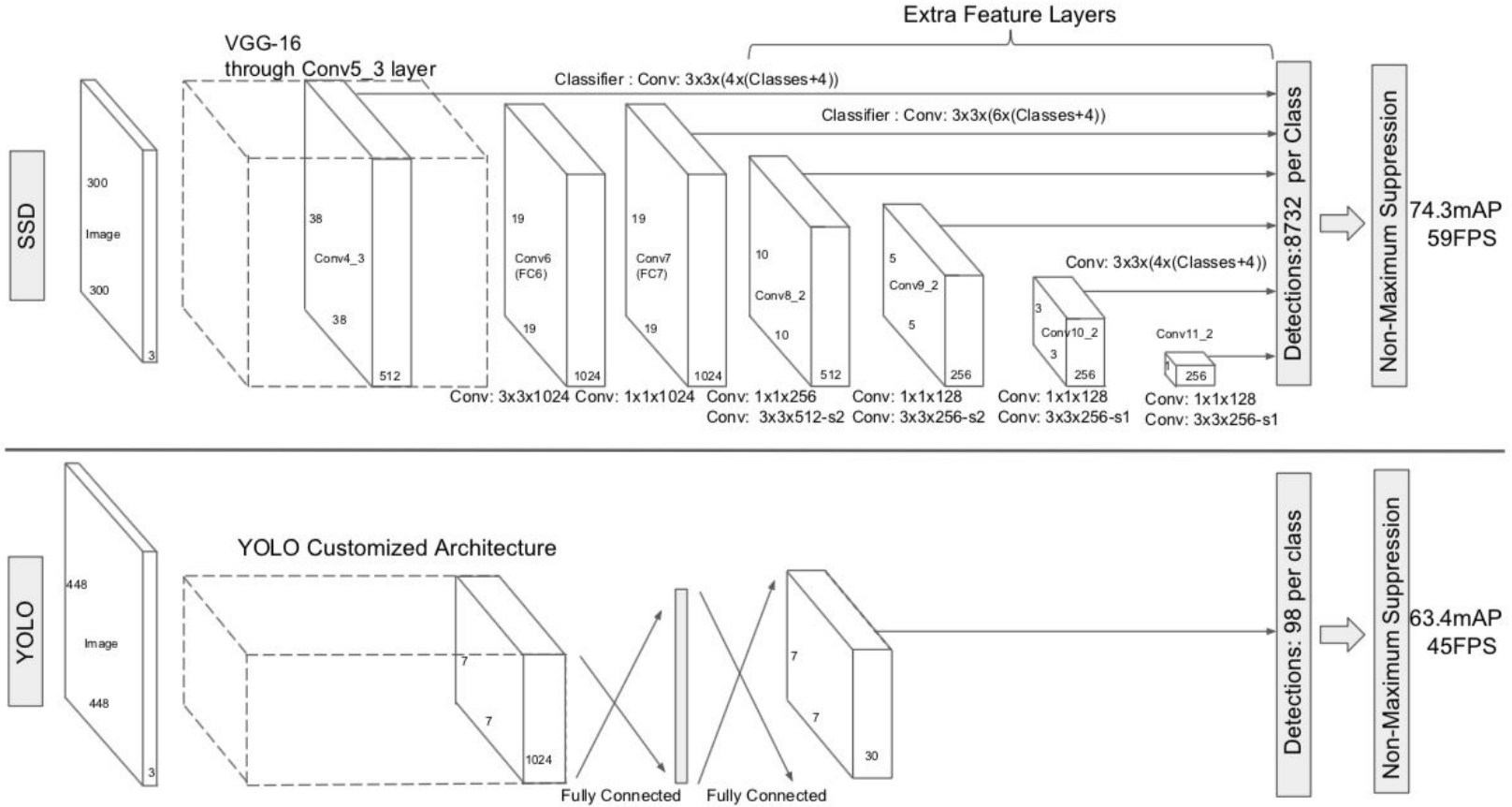
For S=7, B=2 and C=20: the output size is 7 *x* 7 *x* 30 (30=5*x*2+20).



YOLO Network Architecture



SSD Network vs. YOLO Network



YOLO Training

- **First, pre-train** the convolutional layers on the ImageNet 1000-class dataset
  - input size 224*x*224
  - First 20 layers use of the original network
  - Average pooling layer
  - Fully connected layer
  - ImageNet 1000-class competition dataset
- **Next, train the network to perform detection:** increase input to 448*x*448, add 4 convolutional layers and 2 fully connected layers with randomly initialized weights, train for detection
- The final layer predicts both class probabilities and bounding box coordinates
- Linear activation for final layer, and **leaky ReLu** for other layers:

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

YOLO Training - Complete Loss Function

coordinates predictions

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

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

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

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

box confidence prediction

class prediction

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

We assign one predictor to be "responsible" for predicting an object based on which prediction has the highest current intersection over union (IOU) with the ground truth.

YOLO Loss Functions

- **Design Goal:** Optimize the sum-squared error in the output.
  - It is **easy to optimize**
  - However, it is an imperfect auxiliary loss for maximizing average precision
    - It gives an equal amount of importance to localization error and classification error
    - Many cells do not contain objects ⇒ this pushes the confidence scores towards zero, which would overpower the gradients from cells that do contain objects

All these factors can lead to model instability!

YOLO - How do We Ensure a Stable Model?

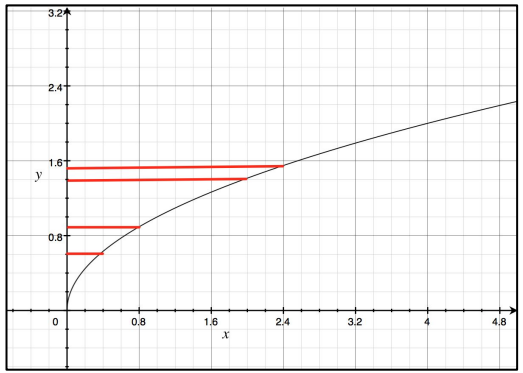
- **Increase** the loss contribution from bounding box coordinate predictions:

$$\lambda_{coord} = 5$$

- **Decrease** the loss contribution from confidence predictions that DO NOT contain objects:

$$\lambda_{noobj} = 0.5$$

- Sum-squared error also **equally weighs** errors from large and small bounding boxes.
  - To partially address this, the square root of the bounding box *width* and *height* is used in the MSE calculation



YOLO Loss Components

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

where $\mathbb{1}_i^{\text{obj}}$ denotes if object appears in cell i and $\mathbb{1}_{ij}^{\text{obj}}$ denotes that the jth bounding box predictor in cell i is "responsible" for that prediction.

If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.

YOLO Loss Components

YOLO predicts **multiple bounding boxes per grid cell**.

Specialization of the bounding box predictor

- At training time we only want one bounding box predictor to be responsible for each object.
- Assign one predictor to be “responsible” for predicting an object, based on which prediction has the highest current intersection over union (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 objects, improving the overall recall.

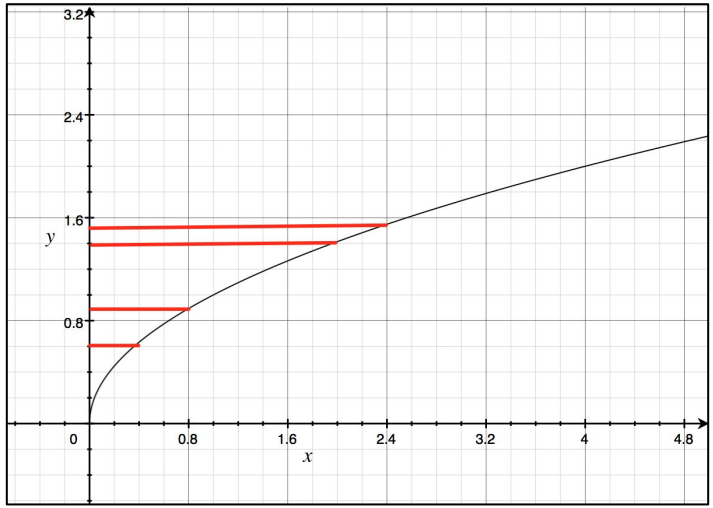
YOLO Loss Components

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.

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



YOLO Loss Components: Confidence

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

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

YOLO Loss Components: *P(Class$_i$|object)*

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

YOLO Inference

General procedure:

- 98 bounding boxes per image
- Discard low confidence bounding boxes
- Use non-max suppression to combine predictions

YOLO Advantages

Extremely fast:

- 45 FPS on Titan X
- 155 FPS on fast version (9 layers instead of 24, fewer filters/layer)

YOLO reasons globally about the image:

- Sees the entire image during training and test time.

Learns generalizable representations of objects.

YOLO Drawbacks

YOLO imposes strong spatial constraints on bounding boxes:

- Each grid cell predicts only B boxes (default choice B = 2)
- Each grid cell can have only one class
  - **Problem:** There could be multiple objects in the same cell, like a man standing in front of a car

Struggles to generalize to objects in new or unusual aspect ratios or configurations:

- Bounding boxes have weak priors (no anchors)
- Coarse features due to many downsampling layers

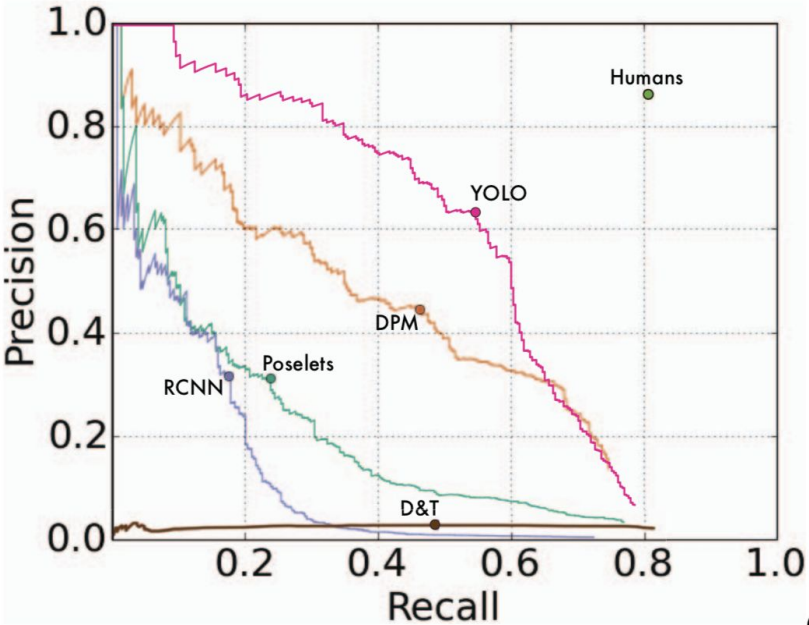
Important: The model struggles with small objects that appear in groups.

Incorrect localizations, especially for small objects.

Alternatives to YOLO - Baseline Comparison

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [30]	2007	16.0	100
30Hz DPM [30]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [37]	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[27]	2007+2012	73.2	7
Faster R-CNN ZF [27]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

YOLO - Baseline Comparisons



(a) Picasso Dataset precision-recall curves.

	VOC 2007	Picasso		People-Art
	AP	AP	Best F_1	AP
YOLO	**59.2**	**53.3**	**0.590**	**45**
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.

YOLO Examples

Comparisons

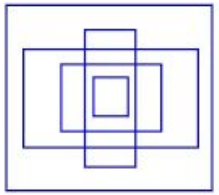
VOC 2012 test	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
MR_CNN_MORE_DATA [11]	**73.9**	**85.5**	**82.9**	**76.6**	**57.8**	**62.7**	**79.4**	77.2	86.6	**55.0**	**79.1**	**62.2**	87.0	**83.4**	**84.7**	78.9	45.3	73.4	65.8	80.3	74.0
HyperNet_VGG	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

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.

Object Detection Chapter 20:

YOLO9000

YOLO9000: Improving YOLOv1



YOLOv1 has high localization error (wrong bbox) and low recall (cannot detect all objects in the image)

Goals: Improve recall, localization, and classification accuracy

Some Improvements:

- **Apply Batch Normalization**
  - Adding batch normalization to all convolutional layers in YOLO increases mAP by 2%
  - This also regularizes the model, and therefore, they remove “dropout”
- **Use High Resolution Classifier**
  - Fine tune the classification network at the full 448x448 resolution for 10 epochs
- **Use Convolution With Anchor Boxes**
  - Remove the fully connected layers in YOLOv1 and use anchor boxes to predict bounding boxes
  - Remove one pooling layer, make the output of the convolutional layer higher resolution
  - Shrink the network to operate on images of size 416x416 instead of 448x448

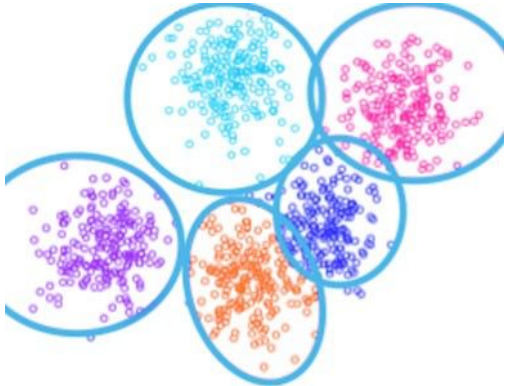
Improving YOLOv1

- **Fine-Grained Features**
  - A pass through layer is added that brings features from an earlier layer at 26 X 26 resolution
  - These are concatenated with the low res 13 X 13 features by stacking adjacent features into different channels (26x26x512 to 13x13x2048)

- **Multi-Scale Training**
  - Since the model uses only convolutional layers, it can be resized “on the fly”
  - Input image size is varied constantly. Every 10 batches, input is resized
  - Since the model down samples by 32, input size is varied from {320, 352, … , 608}

Improving YOLOv1

- **Anchor Boxes: Clusters**
  - The problem with anchor boxes is that their dimensions are hand picked
  - Run *k-means* clustering on the training set bounding boxes to automatically find good priors



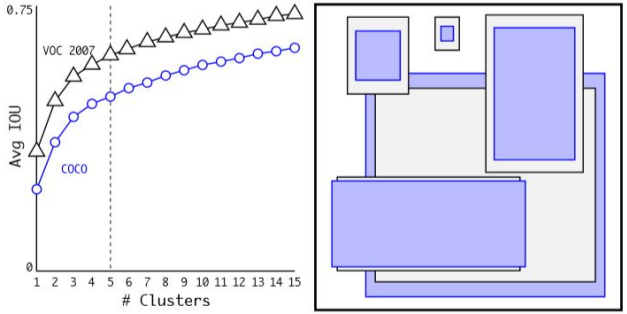
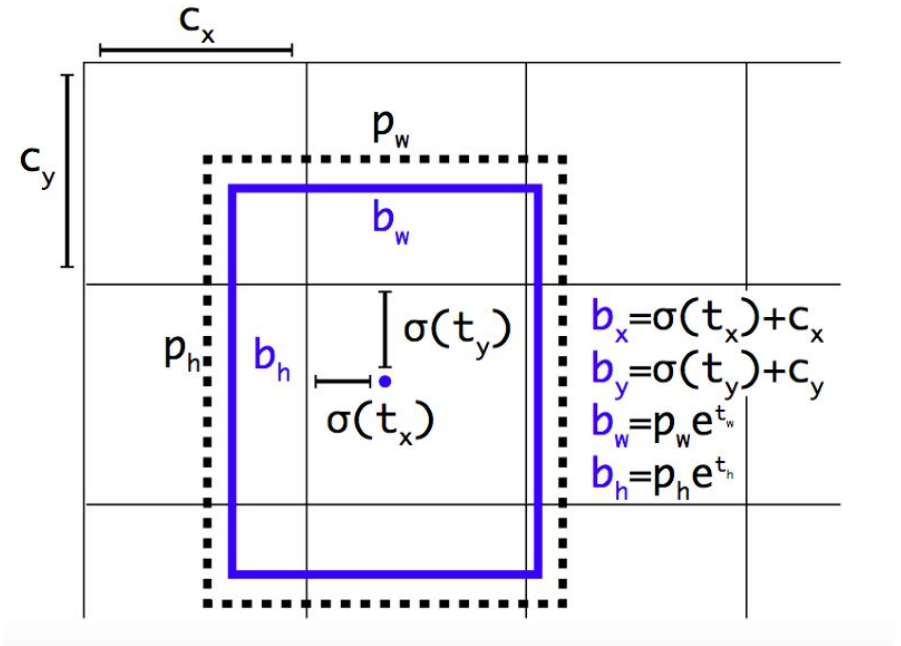


Figure 2: Clustering box dimensions on VOC and COCO. We run k-means clustering on the dimensions of bounding boxes to get good priors for our model. The left image shows the average IOU we get with various choices for k. We find that $k = 5$ gives a good tradeoff for recall vs. complexity of the model. The right image shows the relative centroids for VOC and COCO. Both sets of priors favor thinner, taller boxes while COCO has greater variation in size than VOC.

Improving YOLOv1

- **Anchor Boxes: Clusters**
  - The problem with anchor boxes is that their dimensions are hand picked
  - Run *k-means* clustering on the training set bounding boxes to automatically find good priors
- **Direct Location Prediction**
  - They parameterize the anchor box coordinate prediction in order to improve model stability



https://github.com/datlife/yolov2/blob/0bbb30fd7a59dabec92a1b08c81332d01ace0df4/yolov2/core/custom_layers.py#L92

Anchor Boxes

- Location prediction
- t_x, t_y, t_w, t_h, t_o lead to 5% increase

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

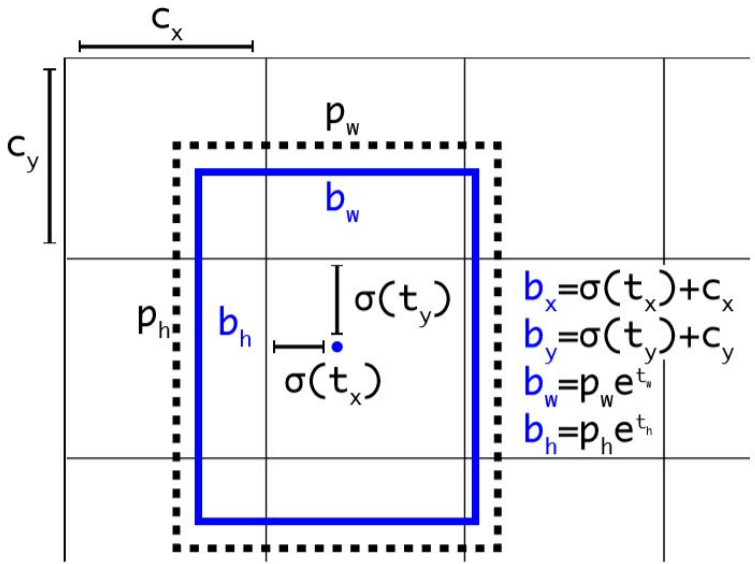
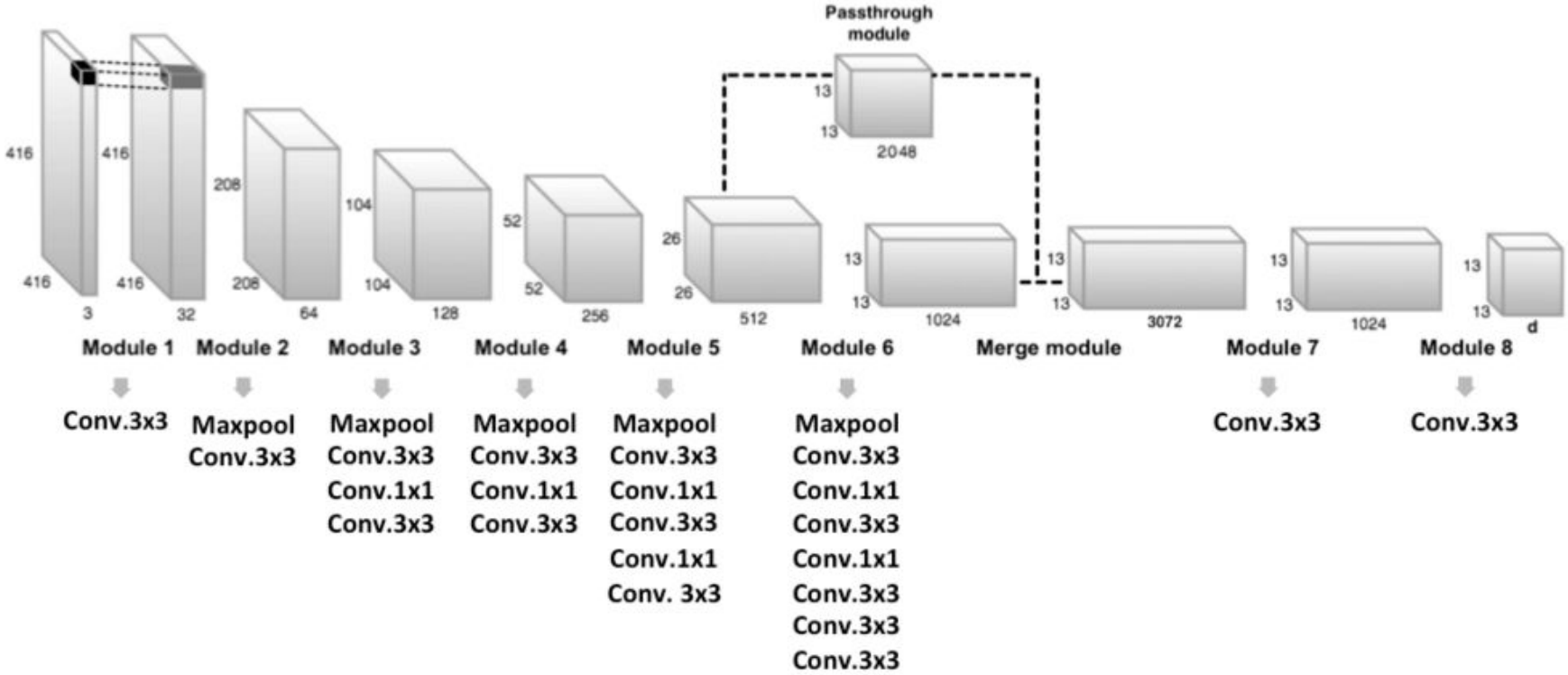


Figure 3: Bounding boxes with dimension priors and location prediction. We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function.

YOLO v2 is Better than YOLO v1

- Remove the last FC layer, use conv layer for prediction
- Resize input into 416*416 , 416/32=13, **one center (objects tend to be at the center)**
- Predict Pr(class|object) and confidence for every bbox(anchor box)

Yolo V2 Architecture



From YOLOv1 to YOLOv2

	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**

Table 2: The path from YOLO to YOLOv2. Most of the listed design decisions lead to significant increases in mAP. Two exceptions are switching to a fully convolutional network with anchor boxes and using the new network. Switching to the anchor box style approach increased recall without changing mAP while using the new network cut computation by 33%.

Darknet-19
Backbone for YoloV2

- Fewer conv layers than Yolo

- 72.9% Top-1 accuracy and
- 91.2% Top-5 accuracy on ImageNet.

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	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		$2 \times 2/2$	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			

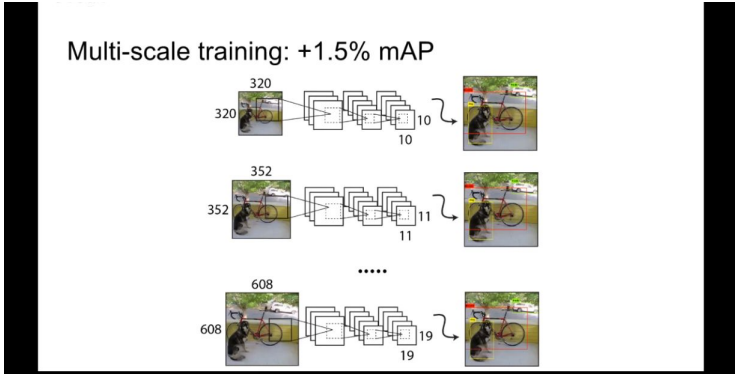
YOLOv2 Training

The training can be divided into 2 steps:

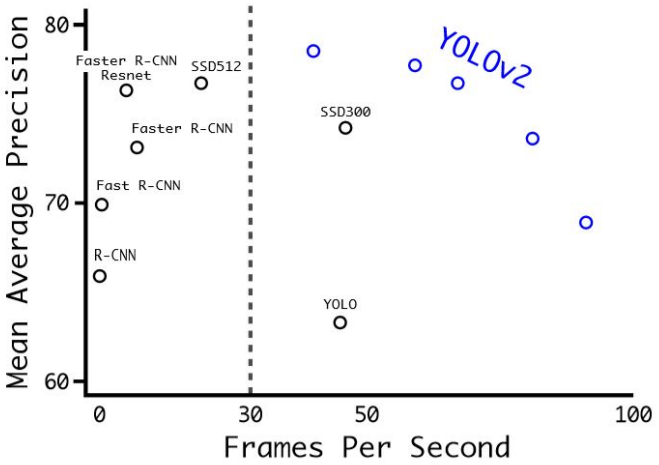
- **Training for Classification**
  - Trained on ImageNet 1000 class dataset
  - For 160 epochs using **darknet** framework
  - SGD 0.1, polynomial rate decay with a power of 4, weight decay of 0.0005 and momentum of 0.9
  - Standard data augmentation tricks including random crops, rotations, and hue, saturation, and exposure shifts.
  - after our initial training on images at 224×224 we fine tune our network at a larger size, 448.
  - After initial training on 224 X 224, fine tuned for 10 epochs on size 448 X 448
- **Training for Detection**
  - The above network is modified by removing the last convolutional layer and adding on three 3 X 3 convolutional layers with 1024 filters followed by a final 1 X 1 convolutional layer
  - A pass through layer is also added (does not exist for classification training)

YOLOv2 Multi Scale Training

- Train with different input sizes every 10 batches
- 1.5% improvement
- 320-608(multiples of 32)



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



YOLOv2 Training Results

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast R-CNN [5]	07++12	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
Faster R-CNN [15]	07++12	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
YOLO [14]	07++12	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
SSD300 [11]	07++12	72.4	85.6	80.1	70.5	57.6	46.2	79.4	76.1	89.2	53.0	77.0	60.8	87.0	83.1	82.3	79.4	45.9	75.9	69.5	81.9	67.5
SSD512 [11]	07++12	74.9	87.4	82.3	75.8	59.0	52.6	81.7	81.5	90.0	55.4	79.0	59.8	88.4	84.3	84.7	83.3	50.2	78.0	66.3	86.3	72.0
ResNet [6]	07++12	73.8	86.5	81.6	77.2	58.0	51.0	78.6	76.6	93.2	48.6	80.4	59.0	92.1	85.3	84.8	80.7	48.1	77.3	66.5	84.7	65.6
YOLOv2 544	07++12	73.4	86.3	82.0	74.8	59.2	51.8	79.8	76.5	90.6	52.1	78.2	58.5	89.3	82.5	83.4	81.3	49.1	77.2	62.4	83.8	68.7

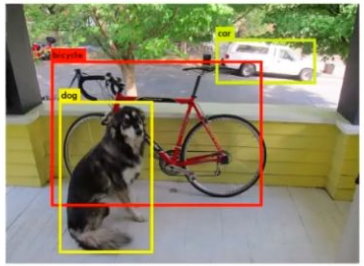
Table 4: PASCAL VOC2012 `test` detection results. YOLOv2 performs on par with state-of-the-art detectors like Faster R-CNN with ResNet and SSD512 and is $2 - 10\times$ faster.

Yolo V2 -> Yolo 9000

- YOLO9000 is YOLOv2 version for a large number of classes - 10000
  - In comparison, YOLO with COCO has only **80** classes

COCO
Common Objects in Context

- 100k images
- 80 classes
- Detection labels



IMAGENET

- 14 million images
- 22k classes
- Classification labels

YOLOv2 -> YOLO9000: Training

Joint Training for Detection and Classification:

- Images labelled for detection are used to learn detection-specific information like bounding box coordinates prediction and objectness
- Images with only class labels are used to to expand the number of categories that can be detected

YOLO9000 Challenges

- Detection datasets have only a few objects and generic labels
- Classification datasets have a much wider and deeper range of labels

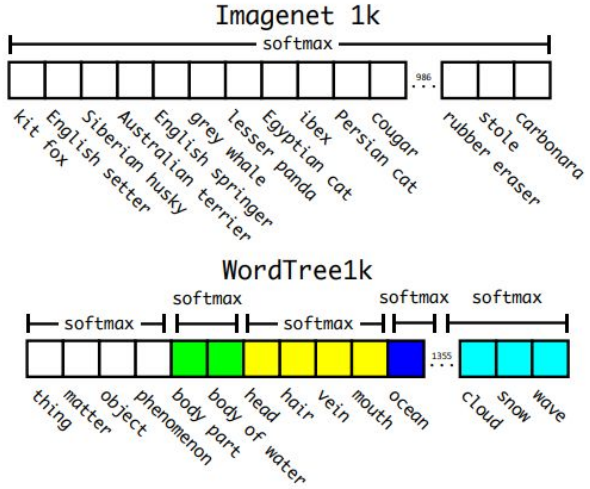
Dog

Norfolk Terrier

Jointly Train Classification and Detection

- Backpropagate full loss for detection data.
- Backpropagate classification loss for classification data.
- Two datasets have different labels, how to combine them?
  - Dog in detection and “Norfolk terrier”, “Yorkshire terrier” “Bedlington terrier” in classification. Cannot use softmax directly, not mutually exclusive.
  - **Approach:** Build a multi label model, that is, give one object **multiple labels.**

$$Pr(\text{Norfolk terrier}) = Pr(\text{Norfolk terrier}|\text{terrier}) * Pr(\text{terrier}|\text{hunting dog}) * \ldots * Pr(\text{mammal}|Pr(\text{animal}) * Pr(\text{animal}|\text{physical object})$$



Jointly Train Classification and Detection

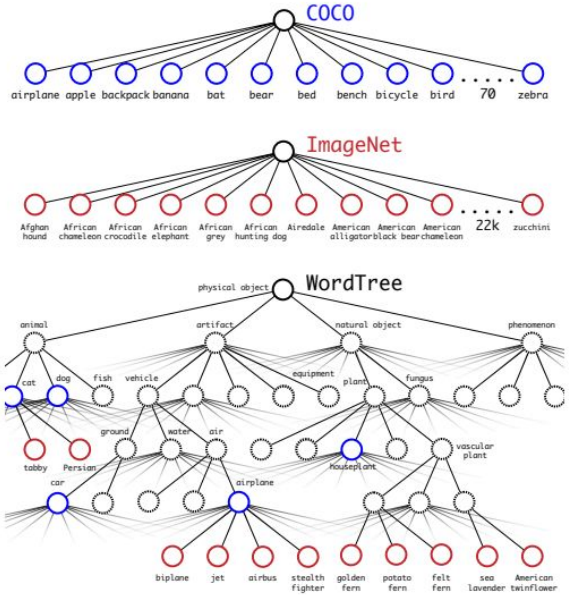


Figure 6: Combining datasets using WordTree hierarchy. Using the WordNet concept graph we build a hierarchical tree of visual concepts. Then we can merge datasets together by mapping the classes in the dataset to synsets in the tree. This is a simplified view of WordTree for illustration purposes.

YOLOv2 Results

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

YOLOv2

	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**

Table 2: The path from YOLO to YOLOv2. Most of the listed design decisions lead to significant increases in mAP. Two exceptions are switching to a fully convolutional network with anchor boxes and using the new network. Switching to the anchor box style approach increased recall without changing mAP while using the new network cut computation by 33%.

YOLOv2

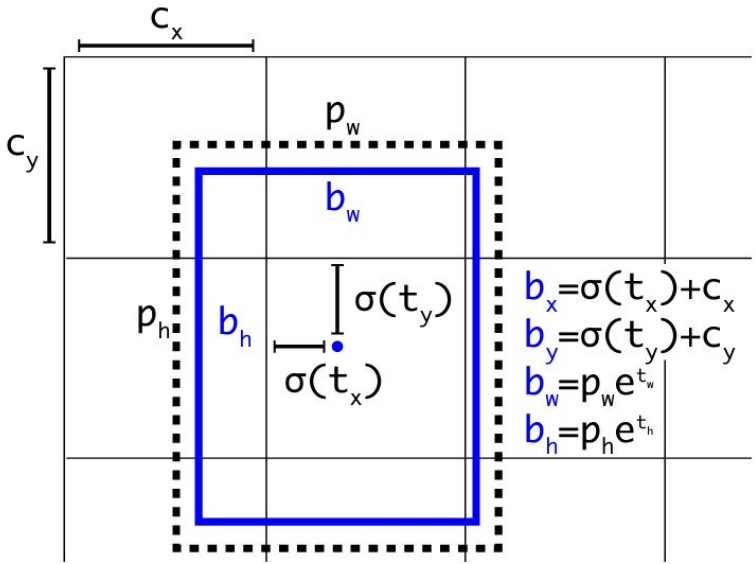


Figure 3: Bounding boxes with dimension priors and location prediction. We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function.

The network predicts 5 bounding boxes at each cell in the output feature map. The network predicts 5 coordinates for each bounding box, t_x, t_y, t_w, t_h, and t_o. 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:

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

YOLOv2: Some Details

- Bounding box classes are no longer per-cell, they are per-bounding box
- We predict 5 bounding boxes per cell


Object Detection Chapter 21:
YOLOv3

YOLOv3: An Incremental Improvement

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.

YOLOv3: An Incremental Improvement

A bunch of little design changes to make YOLO better.

- At 320 × 320 YOLOv3 runs in 22 ms at 28.2 mAP,
- As accurate as SSD and three times faster
- Good Performance: Achieves 57.9 AP50 in 51 ms on a Titan X, compared to 57.5 AP50 in 198 ms by RetinaNet, similar performance but 3.8× faster.
- The code is online at https://pjreddie.com/darknet/yolo/
- Youtube: https://www.youtube.com/watch?v=MPU2HistivI

YOLOv3: Significant Changes

- Backbone is now more ResNet-like (Now called darknet53).
- Each box predicts the classes the bounding box may contain using multilabel classification.
- Predictions are made at 3 different scales.

YOLOv3: An Incremental Improvement From Yolo 9000 and YOLOv2

Class Prediction

- Each box predicts the classes the bounding box may contain using **multi-label** classification. We do not use a softmax as we have found it is unnecessary for good performance, instead we simply use independent logistic classifiers.

YOLOv3: An Incremental Improvement from Yolo 9000 and YOLOv2

Predictions Across Scales

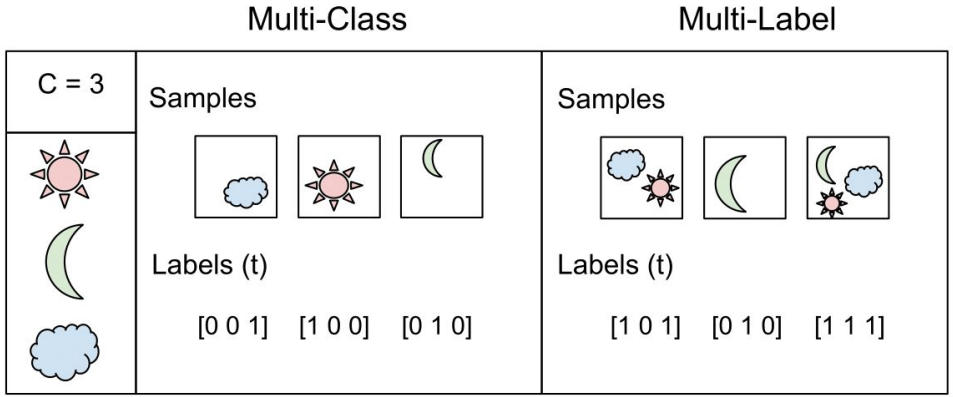
- YOLOv3 predicts boxes at 3 different scales. Extracts features from those scales using a similar concept to feature pyramid networks.

Feature Extractor

- A new network for performing feature extraction, a hybrid approach between the network used in YOLOv2, Darknet-19, with added residual connections. Uses successive 3 × 3 and 1 × 1 convolutional layers but has some shortcut connections as well and is significantly larger.
- It has 53 convolutional layers -> Call it Darknet-53!
- https://pjreddie.com/media/files/papers/YOLOv3.pdf
- https://pjreddie.com/

Class Prediction (Multi-Label Prediction)

- No softmax, instead we use the sigmoid function for each class output
  - Binary classification for each class (Yes/No question)
  - Suitable for multi-label tasks



https://gombru.github.io/2018/05/23/cross_entropy_loss/

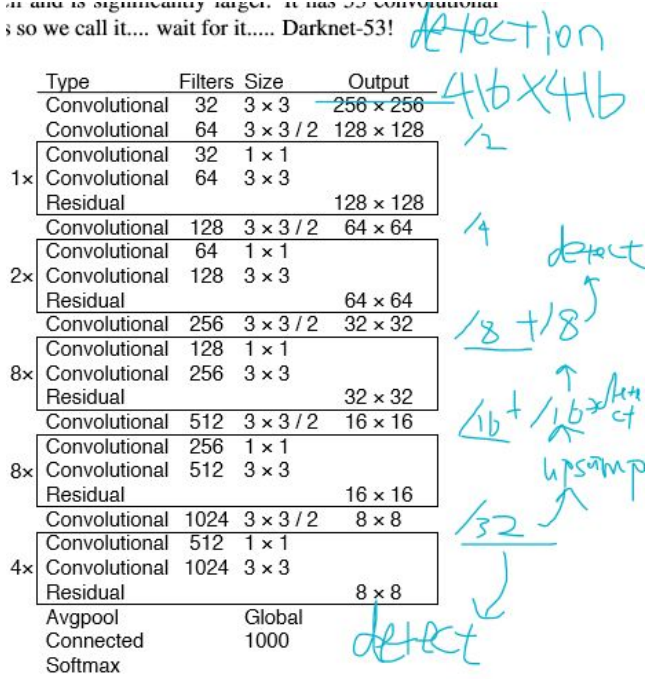
Predictions Across Scales

- Produces feature maps in 3 resolutions
- Input is 416×416, scaled down by 32
- 13×13, 26×26, 52×52, all have 255 (85×3) channels, higher resolution for smaller object
- Each resolution has 3 anchor boxes
- In total 9 anchor boxes for 3 resolutions

jectness, 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.

Feature Extractor

- First, the model is trained at 256×256 resolution for classification.
- Then, we change the input to 416×416 for classification (and fine tune)
- Then change to detection
- COCO Dataset (80 classes), 3 bounding boxes/cell, 80 classes, N by N spatial resolution:
  - **N×N×[3×(4+1+80)]** total outputs per image



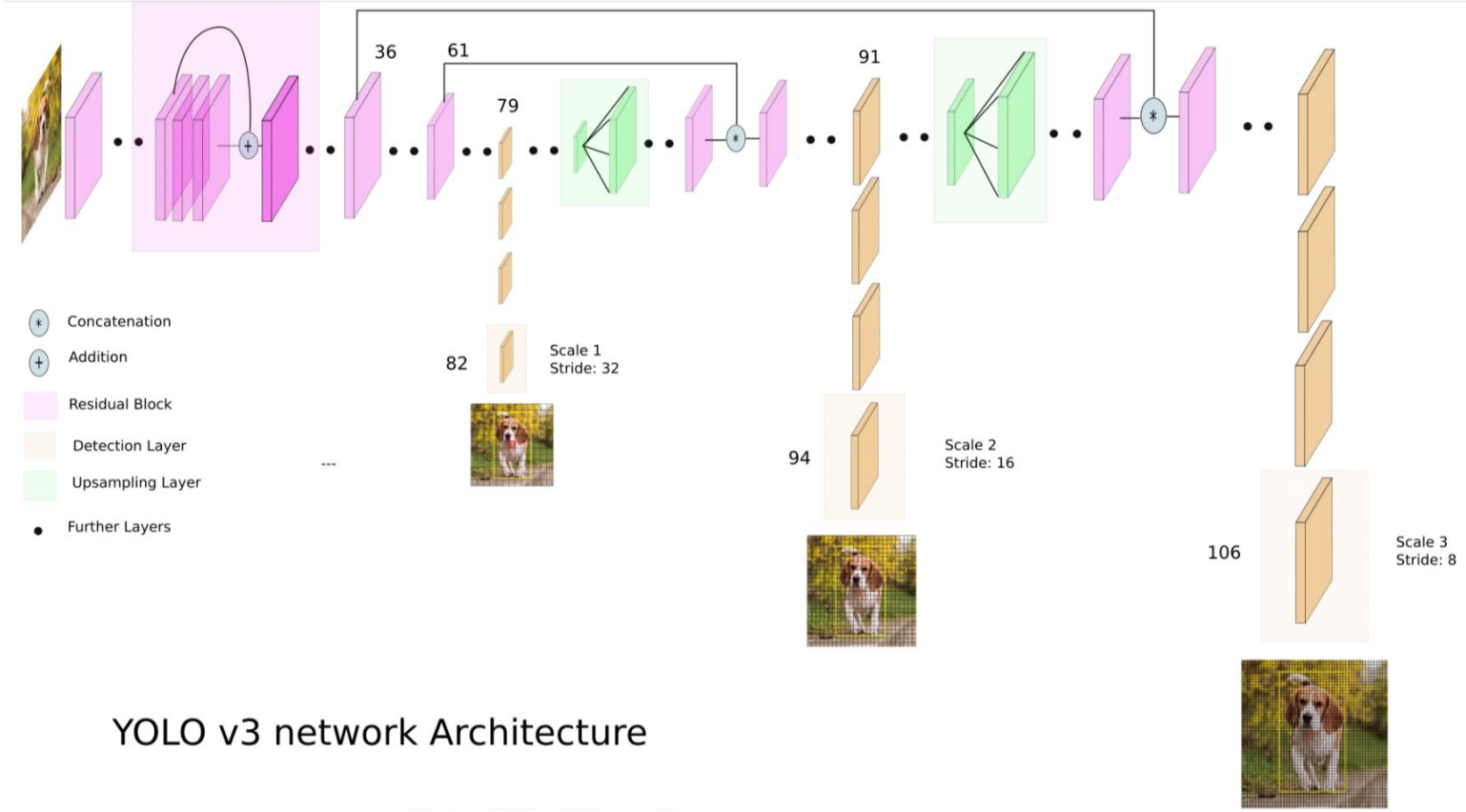
	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 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			

Table 1. **Darknet-53.**

Predictions Across Scales

- We make predictions on three different convolutional layers:
  - The final output, on layer 82, is scaled down by 32
    - Input is 416×416, output is 13×13
    - This 13×13 output is used for large object detection
  - Then, we upsample to 26×26 and concatenate layer 61
    - This scale is used for medium object detection
  - Then, we upsample to 52×52 and concatenate layer 36
    - This scale is designed for small object detection
- Model cfg file can be found in https://github.com/pjreddie/darknet/blob/master/cfg/yolov3.cfg

Predictions Across Scales

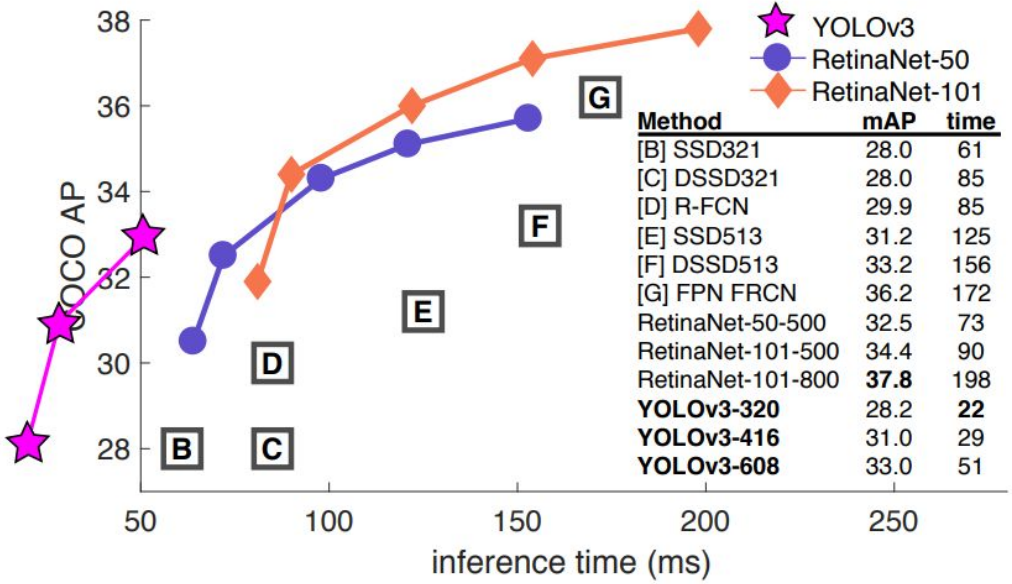


YOLO v3 network Architecture

YOLO v3 Classification 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

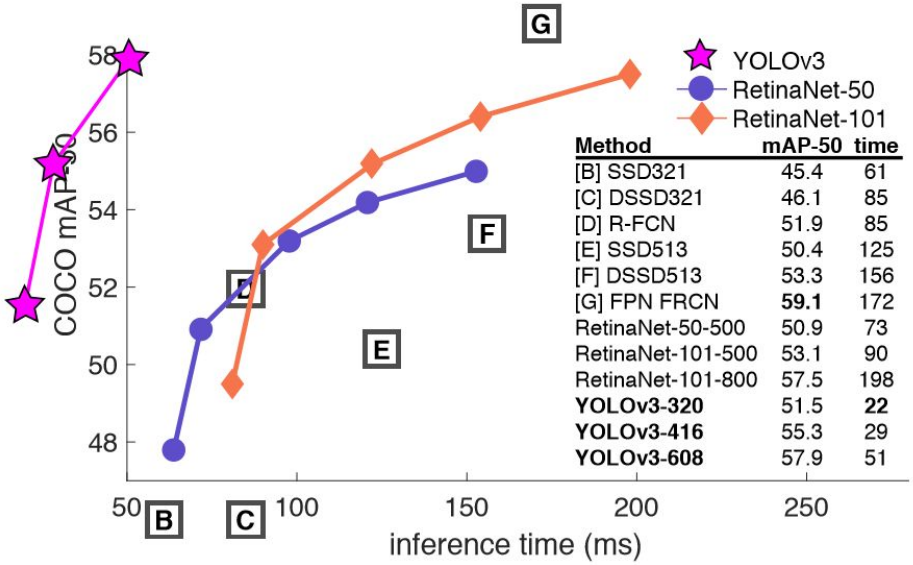
YOLOv3 Detection Results
An Incremental Improvement



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

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.

YOLOv3 Detection Results



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

YOLOv3 Detection Results

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
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**
One-stage methods							
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 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

YOLOv3: What Didn't Work

- **Bounding Box Offsets:** Anchor box prediction mechanism x, y offset is predicted as a multiple of the box width or height using a linear activation decreased model accuracy and stability. Trying out a linear activation by itself also didn't work.
- **Dual IOU Thresholds:** In this formulation, if a prediction overlaps the ground truth by .7 it is as a positive example, by [.3−.7] it is ignored, less than .3 is a negative example. This assignment system doesn't work for YOLOv3.

Object Detection Chapter 21:

YOLOv4

YOLOv4

Bringing several ideas to enhance the performance, with core idea as:

Creating a CNN that operates

A. in real-time on a conventional GPU,
B. training requires only one conventional GPU (by optimization for parallel computations rather than the low computation volume theoretical indicator - BFLOP)

Gain on 416px | v3::35 FPS (Titan X)::AP@50:55.3% | v4:: 38FPS (Titan X)::AP@50::62.8%

https://arxiv.org/pdf/2004.10934.pdf
https://github.com/AlexeyAB/darknet

YOLOv4

Anatomy an object detector:

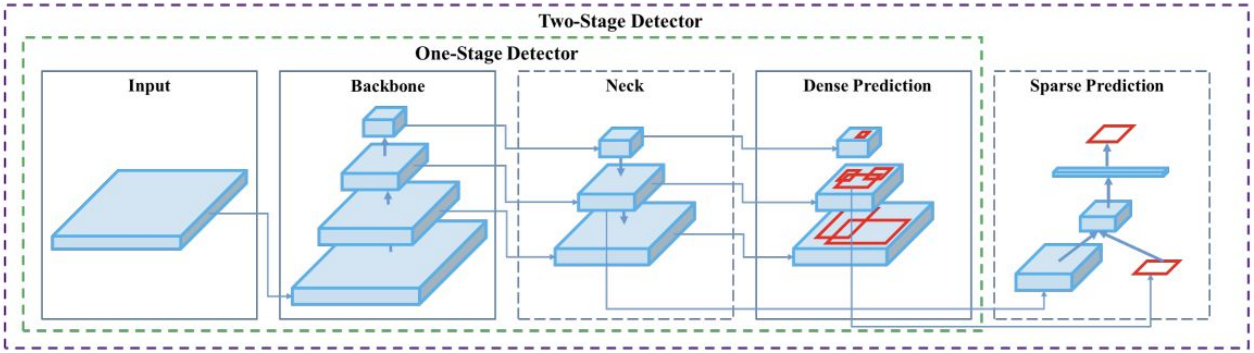


Figure 2: Object detector.

YOLOv4

YOLOv4 consists of:

- Backbone: CSPDarknet53
- Neck: SPP, PAN
- Head: YOLOv3

YOLOv4: Backbone

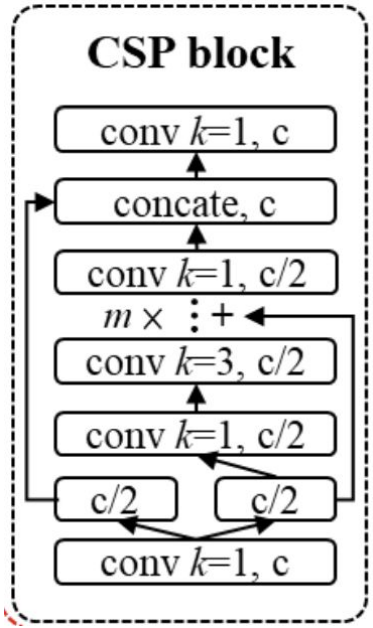
CSPDarknet53: CSPNet strategy to partition the feature map of the base layer into 2 parts and then merges them through a cross-stage hierarchy

Split and merge strategy allows for more gradient flow

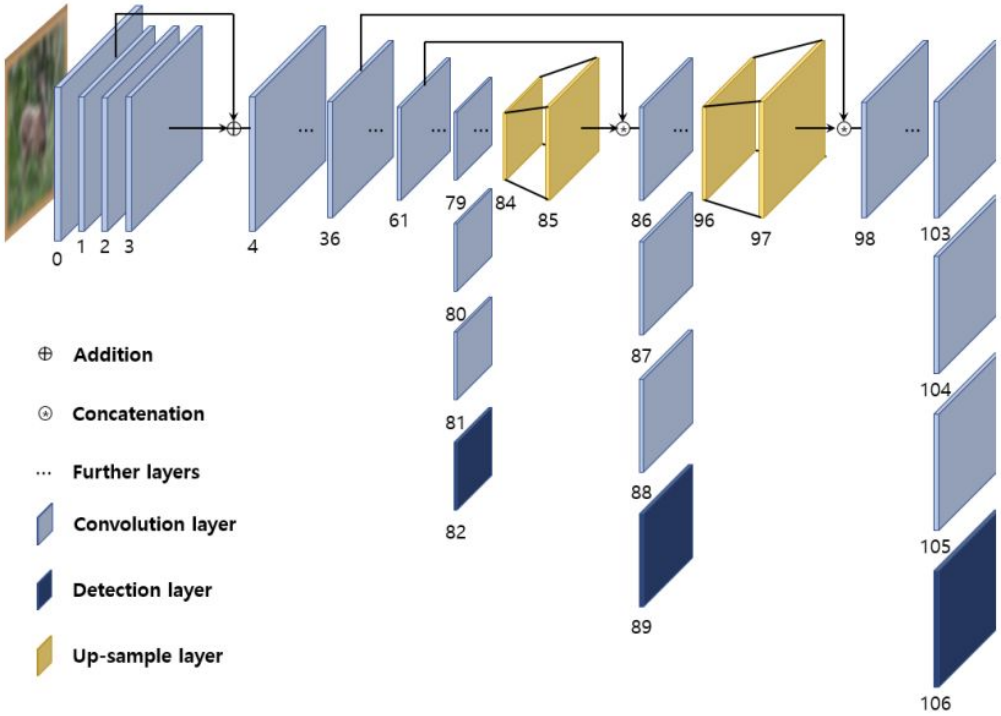
CSP saves on computation for various CNN networks

Table 1: Parameters of neural networks for image classification.

Backbone model	Input network resolution	Receptive field size	Parameters	Average size of layer output (WxHxC)	BFLOPs (512x512 network resolution)	FPS (GPU RTX 2070)
CSPResNext50	512x512	425x425	20.6 M	**1058 K**	31 (15.5 FMA)	62
CSPDarknet53	512x512	725x725	**27.6 M**	950 K	52 (26.0 FMA)	**66**
EfficientNet-B3 (ours)	512x512	**1311x1311**	12.0 M	668 K	11 (5.5 FMA)	26

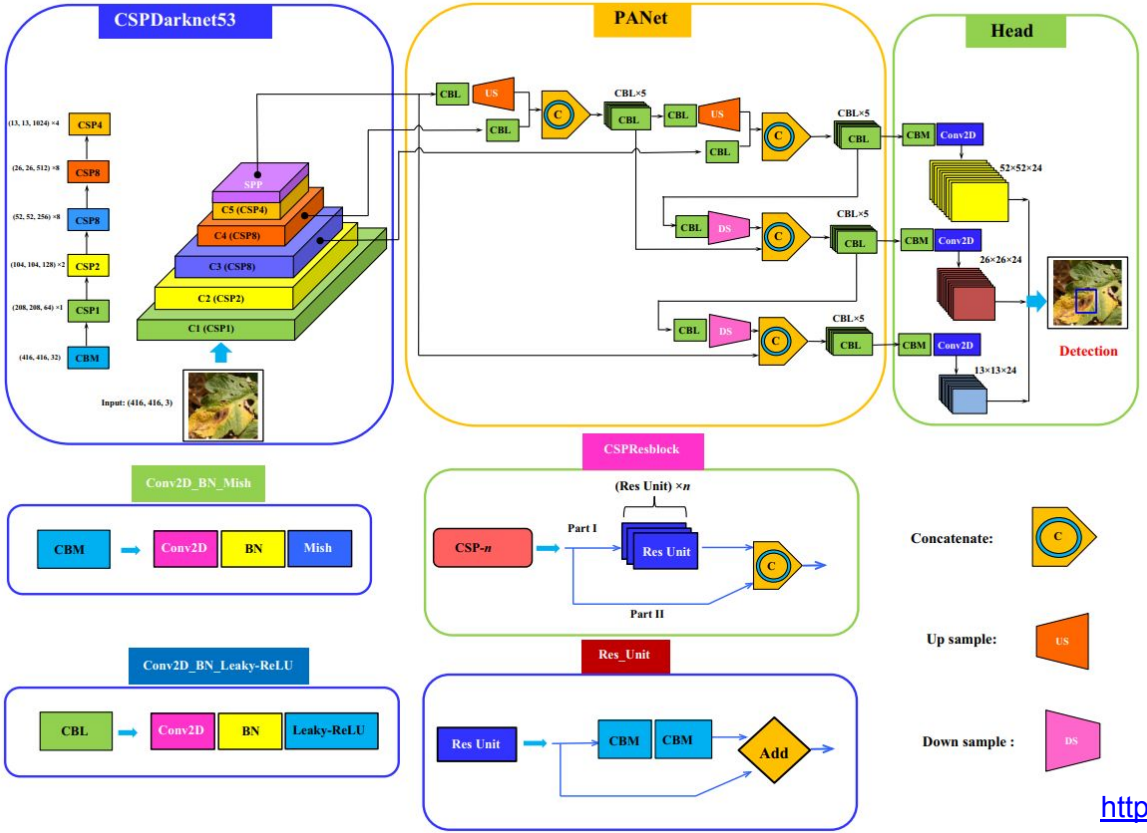


YOLOv3 Architecture



(a)

YOLOv4 vs YOLOv3



https://arxiv.org/abs/2111.00298

CSPBlock

```
def dense_block_cspnet(inps, partition, filter, times, id):
    shape = inps.shape
    features = shape[3] - partition
    part1 = inps
    part1 = inps[:, :, :, 0:features]
    part2 = inps[:, :, :, features:]
    for time in range(0, times):
        part2 = Conv2D(filter[0], 1, padding="same")(part2)
        part2 = mish(part2)
        part2 = Conv2D(filter[1], 3, padding="same")(part2)
        part2 = mish(part2)

    part2 = Conv2D(filter[2], 1, padding="same")(part2)
    part2 = mish(part2)
    inps = Concatenate()([part1, part2])
    return inps
```

YOLOv4: SPP: Enhancing Receptive Field

Spatial Pyramid Pooling:

Problem: CNN output depends on input size.

Solution: SPP tries to create fixed size feature vector for arbitrarily sized image input.

This is helpful in both classification and detection.

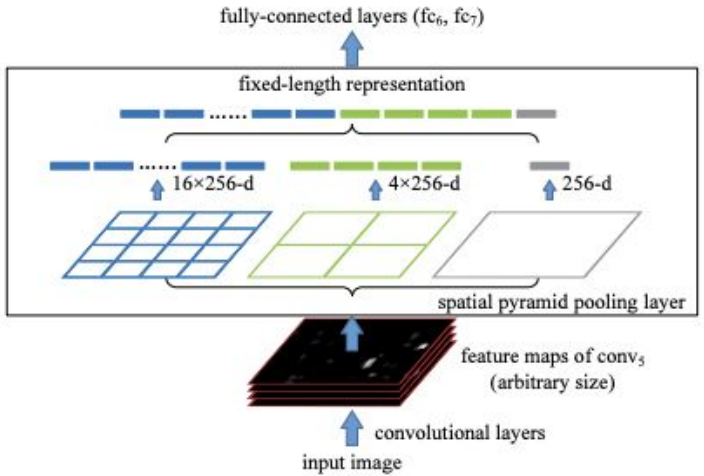
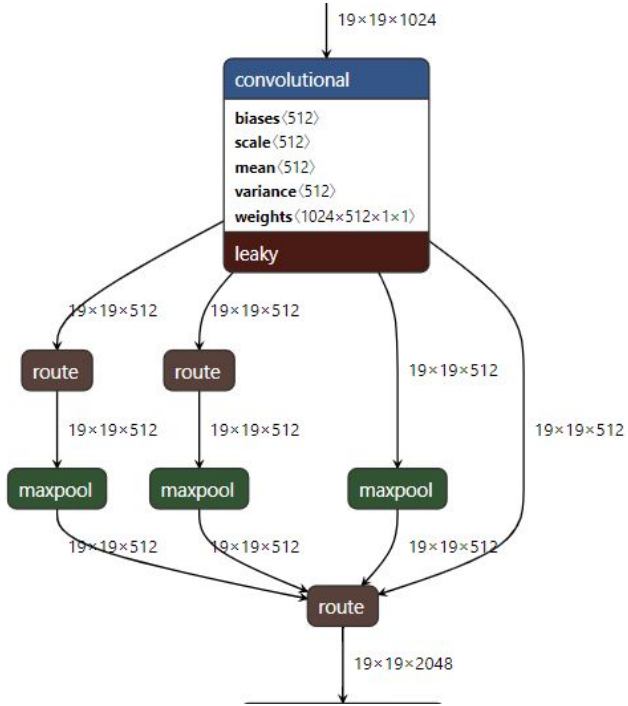


Figure 3: A network structure with a **spatial pyramid pooling layer**. Here 256 is the filter number of the $conv_5$ layer, and $conv_5$ is the last convolutional layer.

YOLOv4: SPP: Enhancing Receptive Field

Spatial Pyramid Pooling:

- performs max-pooling over the 19x19x512 feature map, without pooling
- with different kernel sizes k = { 5, 9, 13} and ‘same’ padding (to keep the same spatial size)



https://towardsdatascience.com/yolo-v4-optimal-speed-accuracy-for-object-detection-79896ed47b50

YOLOv4 - PANet

Path Aggregation Network:

A method for parameter aggregation from different backbone levels for different detector levels

YOLOv3 used FPN instead of PAN:

FPN:
https://arxiv.org/pdf/1612.03144.pdf
PAN:
https://openaccess.thecvf.com/content_cvpr_2018/papers/Liu_Path_Aggregation_Network_CVPR_2018_paper.pdf

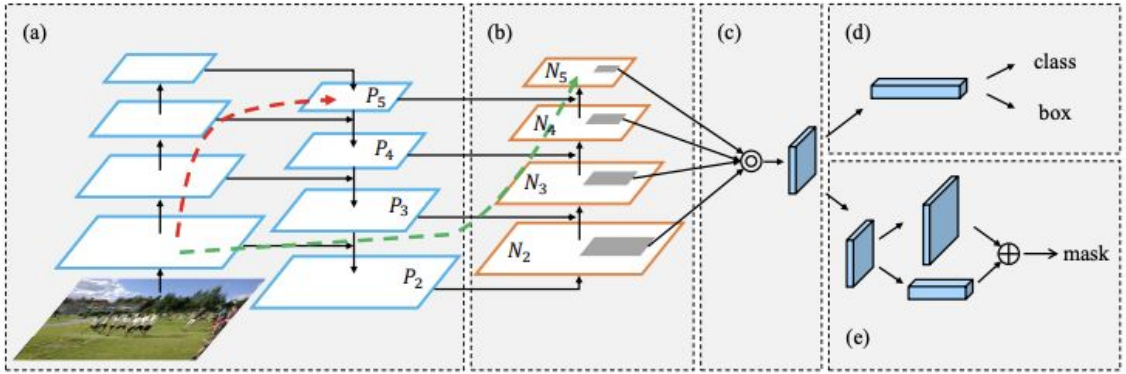


Figure 1. Illustration of our framework. (a) FPN backbone. (b) Bottom-up path augmentation. (c) Adaptive feature pooling. (d) Box branch. (e) Fully-connected fusion. Note that we omit channel dimension of feature maps in (a) and (b) for brevity.

YOLOv4: PANet

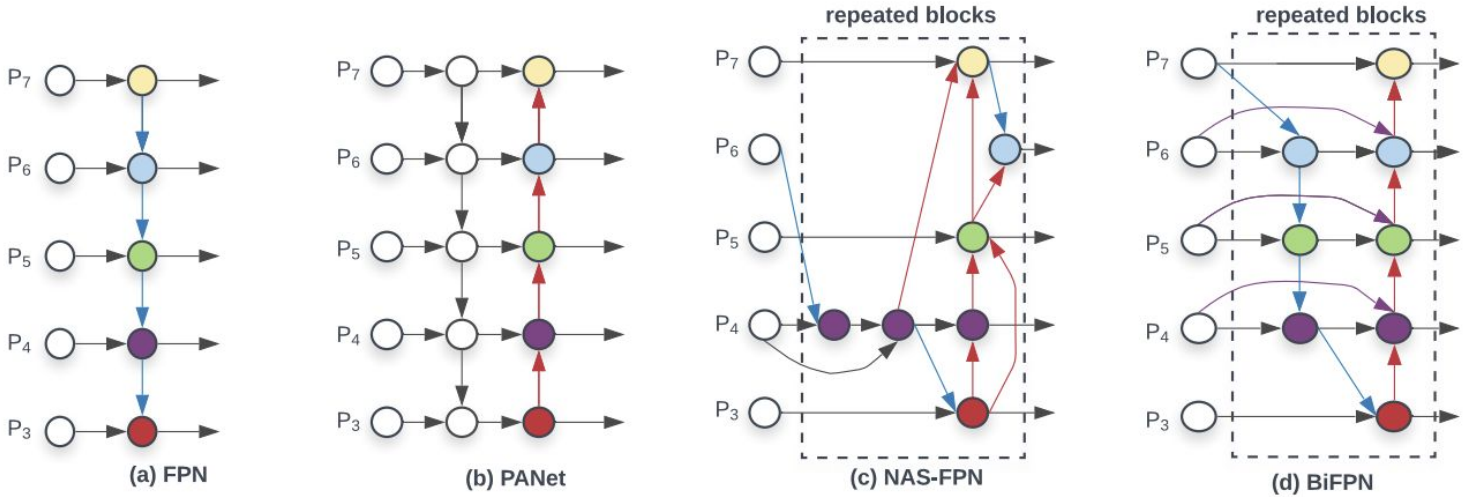


Figure 2: **Feature network design –** (a) FPN [23] introduces a top-down pathway to fuse multi-scale features from level 3 to 7 (P_3 - P_7); (b) PANet [26] adds an additional bottom-up pathway on top of FPN; (c) NAS-FPN [10] use neural architecture search to find an irregular feature network topology and then repeatedly apply the same block; (d) is our BiFPN with better accuracy and efficiency trade-offs.

https://arxiv.org/pdf/1911.09070.pdf
EfficientDet

YOLOv4: Detection Head

Detection still anchor-based as it was in v3.

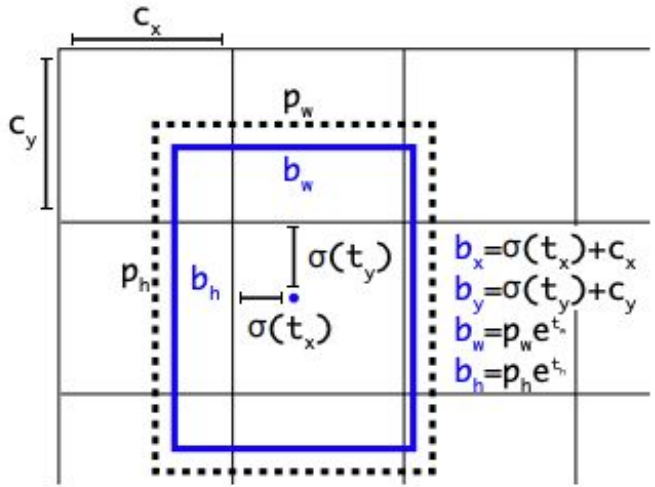
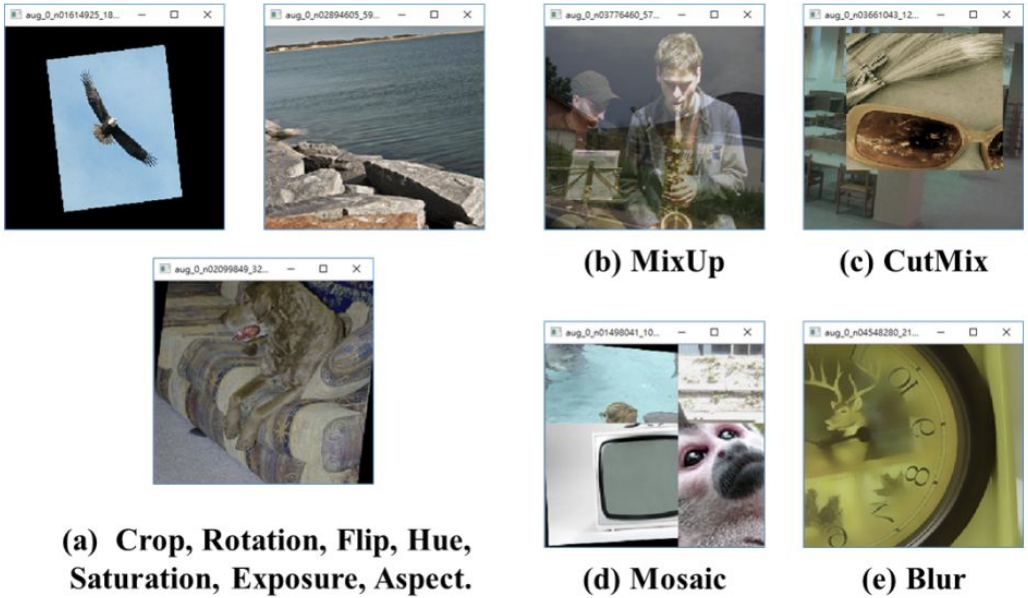


Figure 3: Bounding boxes with dimension priors and location prediction. We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function.

YOLOv4: Augmentation



(a) Crop, Rotation, Flip, Hue, Saturation, Exposure, Aspect.

(b) MixUp

(c) CutMix

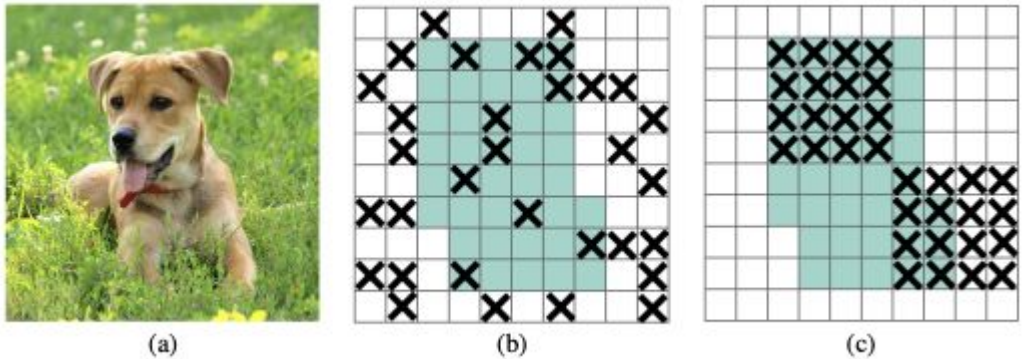
(d) Mosaic

(e) Blur

YOLOv4: DropBlock

DropBlock is more effective than Dropout for CNNs.

We apply DropBlock to input layer(s).



https://arxiv.org/pdf/1810.12890.pdf

YOLOv4: Label Smoothing

Given a parameter α, the ground truth labels are updated with:

$$y\_ls = (1 - \alpha) * y\_hot + \alpha / K$$

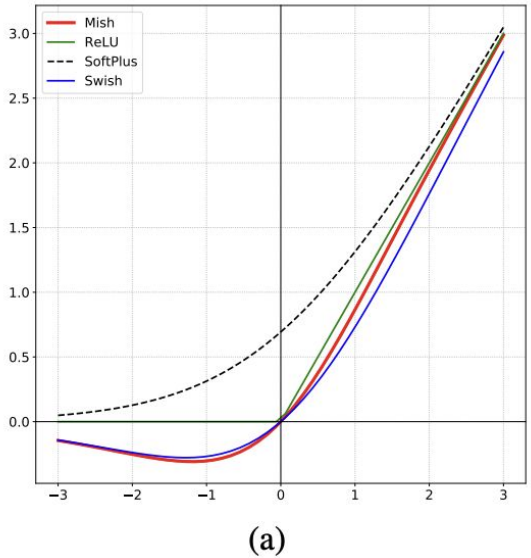
Example:

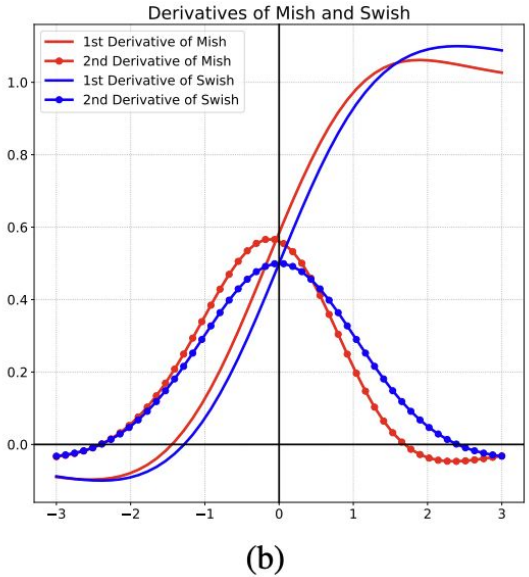
- Applying softmax to the vector [10, 0, 0] gives [0.9999, 0, 0] rounded to 4 decimal places.
- If we use label smoothing with α = 0.1, the smoothed label vector becomes ≈ [0.9333, 0.0333, 0.0333].

https://towardsdatascience.com/what-is-label-smoothing-108debd7ef06

YOLOv4: Mish Activation Function:

Non-monotonous activation function that helps preserve small negative weights.





https://arxiv.org/pdf/1908.08681.pdf

$$\text{mish}(x) = x\tanh(\text{Softplus}(x))$$

$$\text{Softplus}(x) = \frac{1}{\beta} * \log(1 + \exp(\beta * x))$$

YOLOv4: Mish Activation Function:

Table 2: Influence of BoF and Mish on the CSPResNeXt-50 classifier accuracy.

MixUp	CutMix	Mosaic	Bluring	Label Smoothing	Swish	Mish	Top-1	Top-5
							77.9%	94.0%
✓							77.2%	**94.0%**
	✓						**78.0%**	**94.3%**
		✓					**78.1%**	**94.5%**
			✓				77.5%	93.8%
				✓			**78.1%**	**94.4%**
					✓		64.5%	86.0%
						✓	**78.9%**	**94.5%**
	✓	✓		✓			**78.5%**	**94.8%**
	✓	✓		✓		✓	**79.8%**	**95.2%**

YOLOv4: Complete IoU (CIoU)

CIoU loss [4], simultaneously considers three metrics: overlapping area, the distance between center points, and the aspect ratio. It is an extension to DIoU loss.

$$\mathcal{L}_{CIoU} = 1 - IoU + \frac{\rho^2(b, b^{gt})}{c^2} + \alpha v$$

CIoU loss

where b and b^{gt} denote the central points of B and B^{gt}, ρ(·) is the Euclidean distance, c is the diagonal length of the smallest enclosing box covering the two boxes, α is a positive trade-off parameter, and v measures the consistency of aspect ratio.

The α is a positive trade off parameter where overlaps are given higher priority over non-overlap cases and v gives information about consistency of aspect ratio.

$$v = \frac{4}{\pi^2}(arctan\frac{w^{gt}}{h^{gt}} - arctan\frac{w}{h})^2$$

$$\alpha = \frac{v}{(1 - IoU) + v}$$

where h is the height of the bounding box and w is the width.

https://wikidocs.net/167839

YOLOv4: Self-Adversarial Training (SAT)

SAT aims to find the portion of the image that the network most relies on during training, and then edits the image to obscure this reliance, forcing the network to generalize to new features that can help it with detection.

Two-stage Operation:

1. Neural network alters the original image instead of the network weights. In this way the neural network executes an adversarial attack on itself, altering the original image to create the deception that there is no desired object on the image.
2. NN is trained to detect an object on this modified image in the normal way.

YOLOv4: Cross mini-batch Normalization (CmBN)

This collects statistics only between mini-batches within a single batch, and only updates bias and scale parameters at the end of the batch.

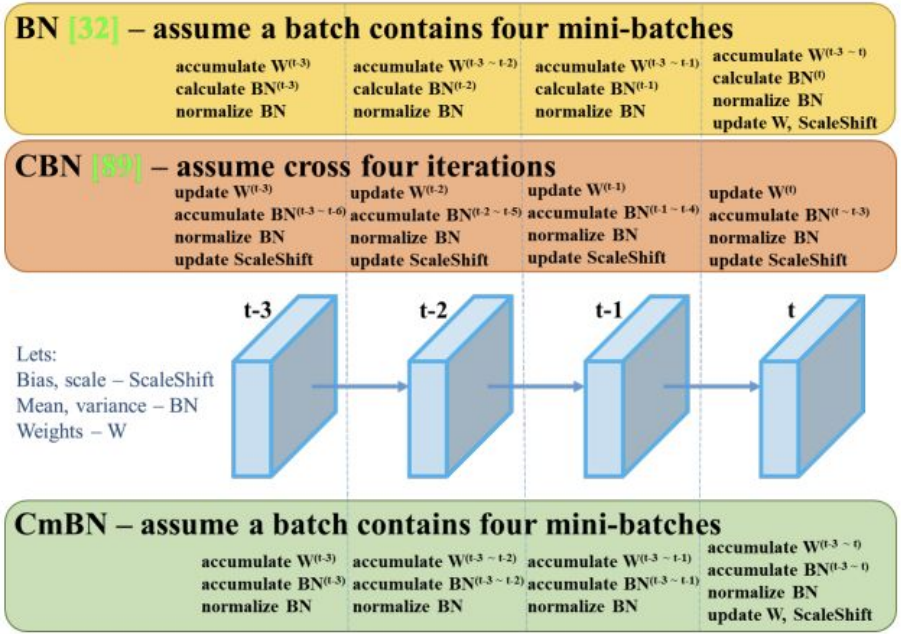
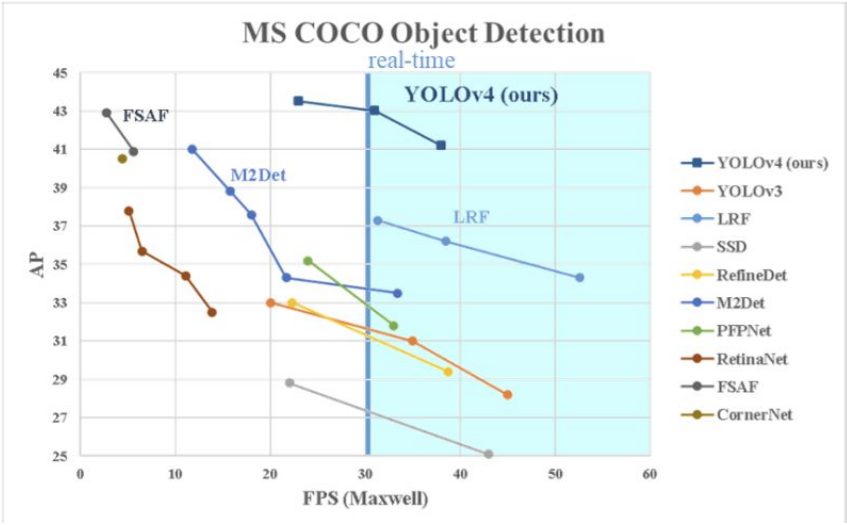
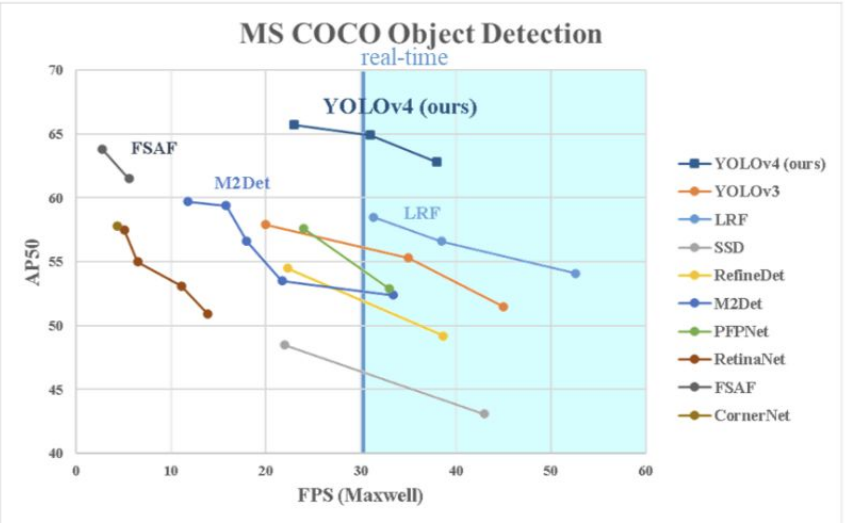


Figure 4: Cross mini-Batch Normalization.

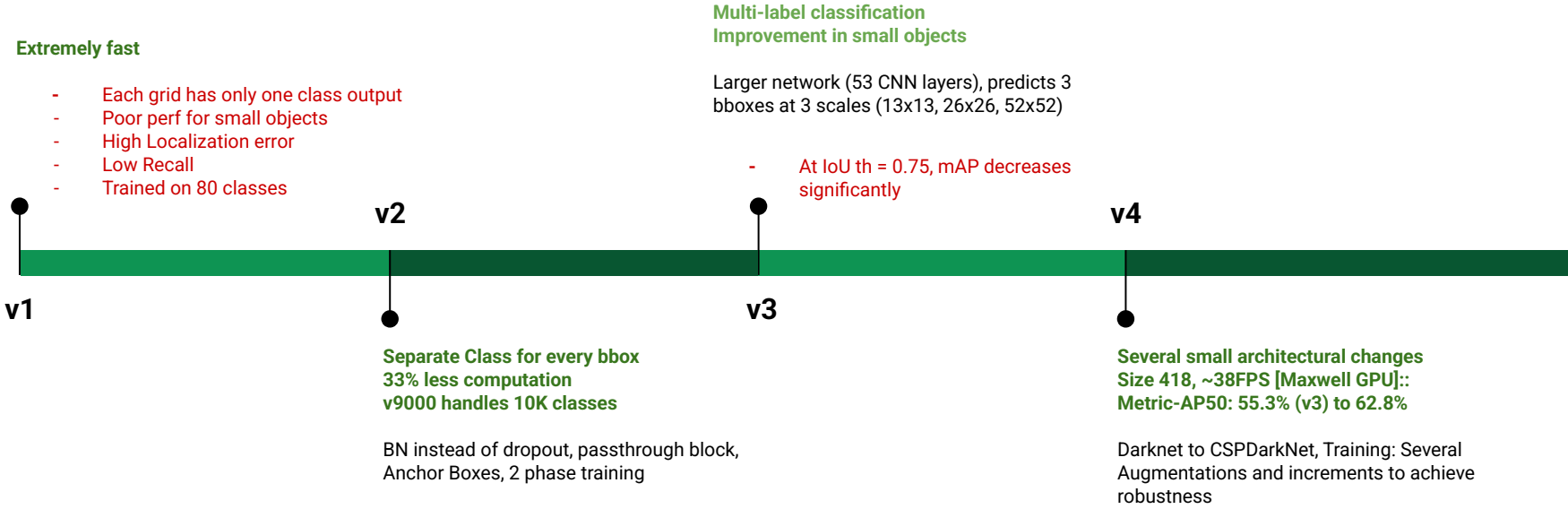
https://arxiv.org/pdf/1908.08681.pdf

YOLOv4: Performance





Evolution of YOLO Models



Ref: https://blog.roboflow.com/guide-to-yolo-models/
Paper Yolo v4: https://arxiv.org/pdf/2004.10934.pdf

Object Detection Chapter 24:

RetinaNet

Why do One-Stage Detectors Trail in Accuracy?

We encounter an extreme foreground-background class imbalance in object detection tasks.

Two-stage Detectors' Approach:

The proposal stage rapidly narrows down #candidate object locations to a small number (e.g., 1-2k), filtering out most background samples

In the classification stage, fix foreground-to-background ratio to 1:3, or online hard example mining (OHEM).

We encounter an extreme foreground-background class imbalance in object detection tasks.

One-stage Detectors' Approach:

Have to process a much larger set of candidate object locations regularly sampled across an image, which amounts to enumerating ~100k locations that densely cover spatial positions, scales, and aspect ratios.

RetinaNet - Focal Loss

The central cause of lower performance of single-stage detectors is the extreme foreground-background class imbalance encountered during training of dense detectors.

New Approach called Focal Loss: Addresses this class imbalance by reshaping the standard cross entropy loss such that it down-weights the loss assigned to well-classified examples.

Focal Loss for Dense Object Detection, Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár, https://arxiv.org/abs/1708.02002

Object Detection Background/Foreground Imbalance

An important issue for object detection model training is the extreme imbalance between background which contains no object and foreground which holds objects of interest.

Focal loss is designed to assign more weights on hard, easily misclassified examples (i.e. background with noisy texture or partial object) and to down-weight easy examples (i.e. obviously empty background).

Object Detection Background/Foreground Imbalance

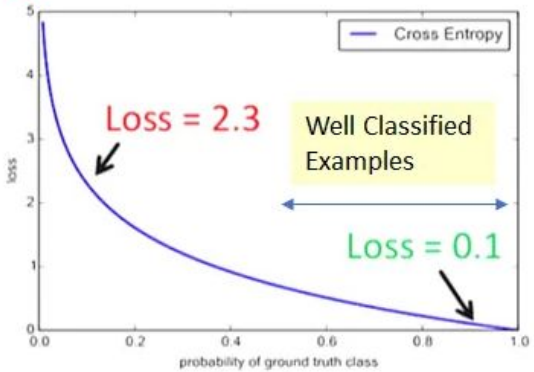
Let's start with the **cross-entropy loss** for binary classification:

$$\mathrm{CE}(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise.} \end{cases}$$

where $y \in \{-1,1\}$ is a ground truth binary label, indicating whether a bounding box contains a object, and $p \in [0,1]$ is the predicted probability for the class with label 1.

Behaviour of Cross-Entropy Loss for Unbalanced Data

- 100000 easy : 100 hard examples
- **40x bigger loss** from easy examples



- The loss from easy (background) examples = 100,000 x 0.1 = 10,000
- The loss from hard (foreground) examples = 100 x 2.3 = 230
- 10,000 / 230 = 43
- The loss from easy examples is 40x bigger

One Approach: α-Balanced CE Loss

For notational convenience:

$$\text{let } p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise} \end{cases}, \text{then } \mathrm{CE}(p, y) = \mathrm{CE}(p_t) = -\log p_t$$

To address the class imbalance, one method is to add a weighting factor α for class 1 and 1 - α for class -1:

$$\mathrm{CE}(p_t) = -\alpha_t \log(p_t)$$

α may be set by **inverse class frequency** or treated as **a hyperparameter** to set by cross validation.

As seen at the two-stage detectors, α is implicitly implemented by selecting the foreground-to-background ratio of 1:3.

Focal Loss Expression

For notational convenience

$$\text{let } p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{otherwise} \end{cases}, \text{then } \mathrm{CE}(p, y) = \mathrm{CE}(p_t) = -\log p_t$$

- Easily classified examples with large $p_t \gg 0.5$, that is, when p is very close to 0 (*when* y=0) *or* 1 (*when* y=1), can incur a loss with non-trivial magnitude.

Focal loss explicitly adds a weighting factor $(1-p_t)^\gamma$, $\gamma \geq 0$ to each term in cross entropy so that the weight is small when p_t is large and therefore easy examples are down-weighted:

$$\mathrm{FL}(p_t) = -(1 - p_t)^\gamma \log p_t$$

Focal Loss Expression with Hyperparameter α

For better control, we introduce **a hyperparameter α**

$$\mathrm{FL}(p_t) = -\alpha_t (1 - p_t)^{\gamma} \log(p_t)$$

$$\mathrm{FL}(\mathrm{p}, \mathrm{y}) = \begin{cases} -\alpha(1-\mathrm{p})^{\gamma}\log(\mathrm{p}) & \text{if} \quad \mathrm{y} = 1 \\ -(1-\alpha)\mathrm{p}^{\gamma}\log(1-\mathrm{p}) & \text{otherwise,} \end{cases}$$

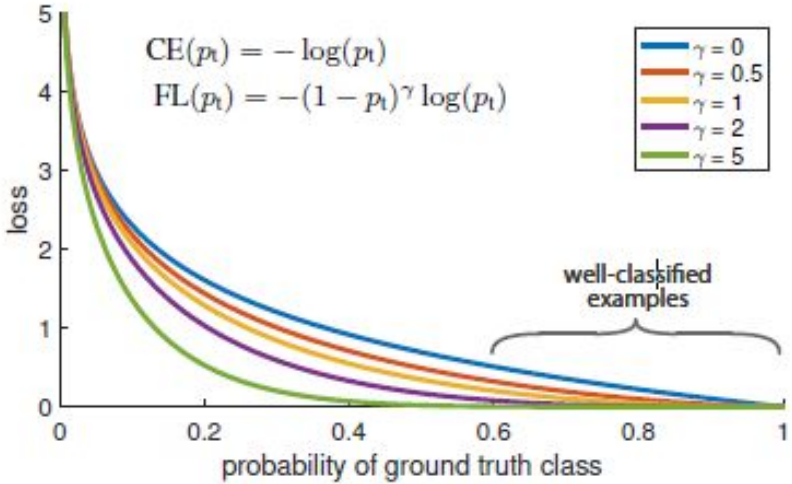
RetinaNet uses an α-balanced variant of the focal loss, where α=0.25, γ=2 works the best.

- While α balances the importance of positive/negative examples **it does not differentiate between** easy/hard examples.

Focal Loss vs. Conventional Cross-Entropy Loss

Setting γ**>0** reduces the relative loss for well-qualified examples with p_t**>0.5**, putting more focus on hard, misclassified examples.

Focal loss facilitates the training of highly accurate dense object detectors in the presence of vast number of easy background examples.



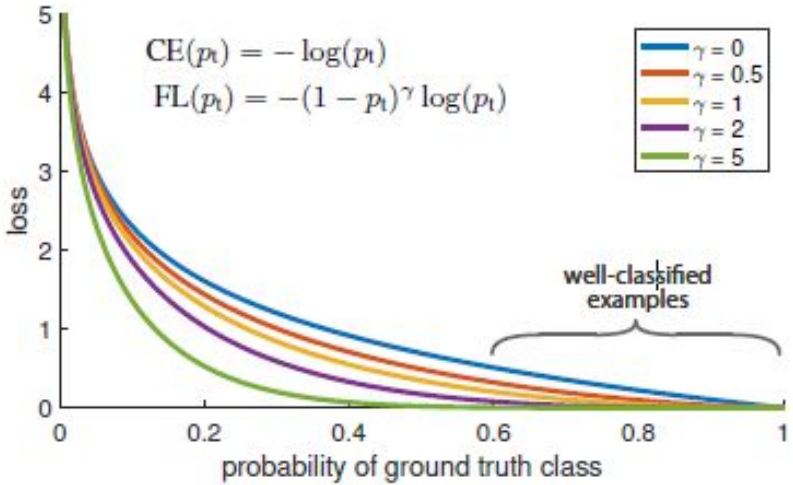
https://arxiv.org/abs/1708.02002

Focal Loss Interpretation

Every sample is weighted according to its error!

Focusing parameter γ smoothly adjusts the rate at which easy examples are down-weighted.

- For misclassified, p_t is small, modulating factor is near 1, and the loss is unaffected
- As $p_t \rightarrow 0.5$, the factor goes to 0 and the loss for well-qualified examples is down-weighted
- With $\gamma=2$, the example classified with $p_t=0.9$ would have 100 times lower loss compared to CE and with $p_t=0.968$ it would have 1000 times lower loss. This in turn increases the importance of correcting misclassified examples



Focal Loss -> RetinaNet

The Focal Loss focuses training on a sparse set of hard examples and prevents the vast number of easy negatives from overwhelming the detector during training:

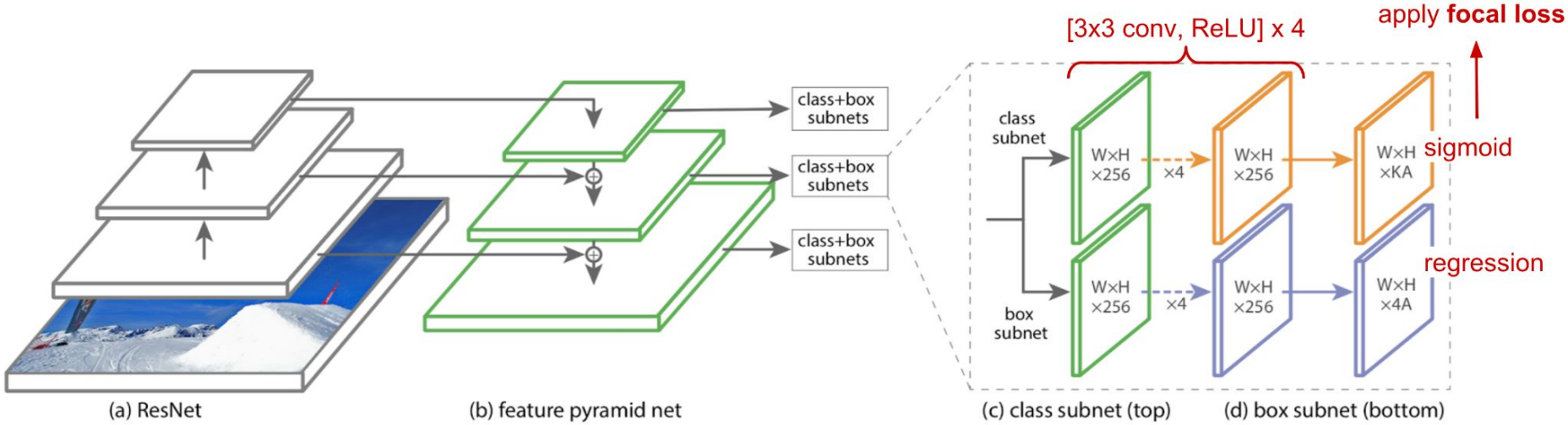
- To evaluate the effectiveness of the focal loss, a new design/training method: a simple dense detector called **RetinaNet**.

The RetinaNet is a one-stage dense object detector:

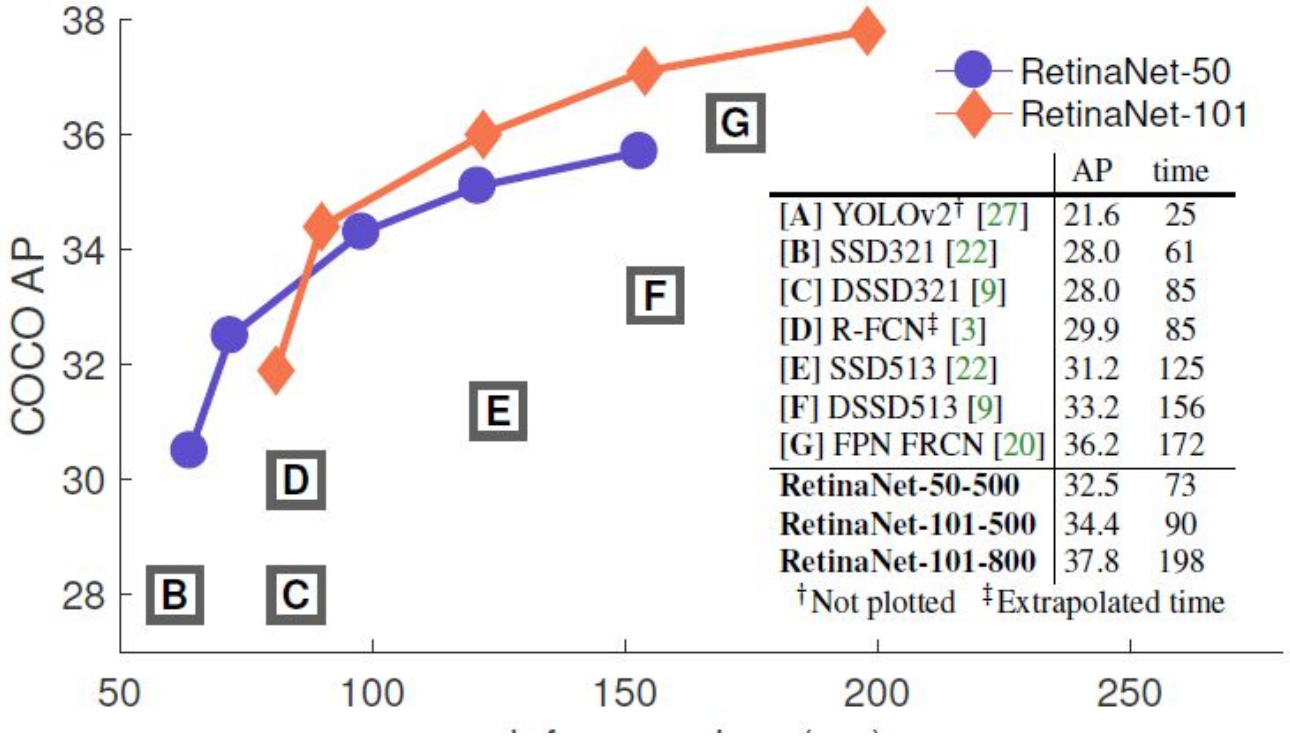
- Two crucial building blocks are
  - the FPN, and
  - the use of focal loss.
- The results show that when trained with the focal loss, RetinaNet is able to match the speed of previous one-stage detectors while surpassing the accuracy of all existing state-of-the-art two-stage detectors.

Code is at: https://github.com/facebookresearch/detectron2

RetinaNet Model Architecture



RetinaNet Performance

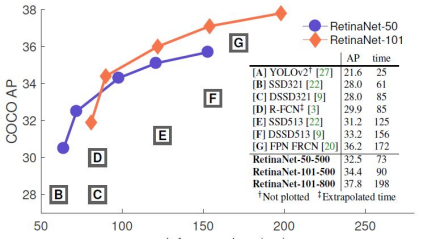


	AP	time
[A] YOLOv2† [27]	21.6	25
[B] SSD321 [22]	28.0	61
[C] DSSD321 [9]	28.0	85
[D] R-FCN‡ [3]	29.9	85
[E] SSD513 [22]	31.2	125
[F] DSSD513 [9]	33.2	156
[G] FPN FRCN [20]	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	37.8	198

†Not plotted ‡Extrapolated time

RetinaNet Performance

Speed (ms) versus accuracy (AP) on COCO test-dev.



	AP	time
[A] YOLOv2† [27]	21.6	25
[B] SSD321 [22]	28.0	61
[C] DSSD321 [9]	28.0	85
[D] R-FCN† [3]	29.9	85
[E] SSD513 [22]	31.2	125
[F] DSSD513 [9]	33.2	156
[G] FPN FRCN [20]	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	37.8	198

†Not plotted ‡Extrapolated time

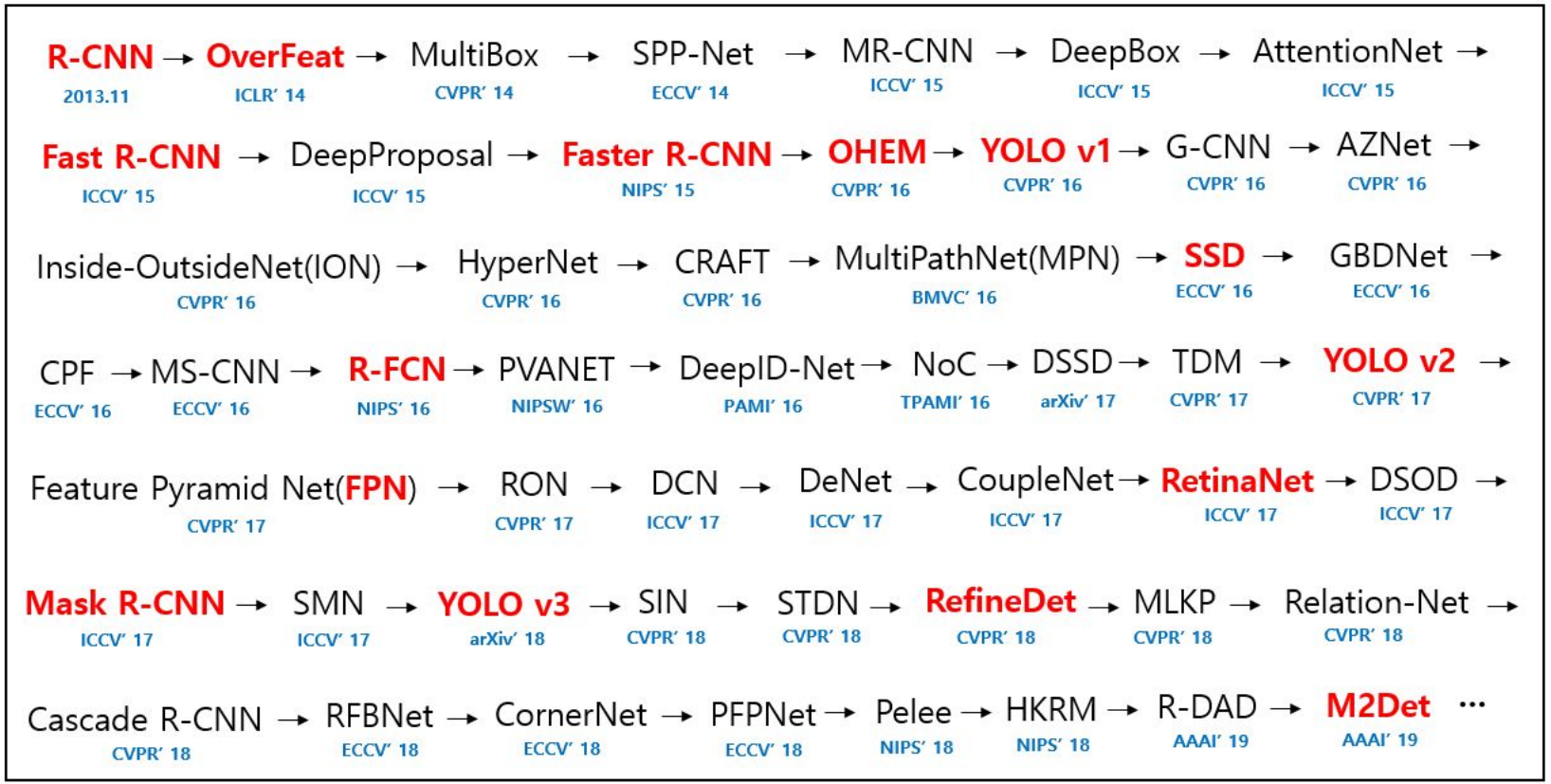
- Enabled by the focal loss, the simple one-stage RetinaNet detector outperforms all previous one-stage and two-stage detectors, including the best reported Faster R-CNN.
- See variants of RetinaNet with ResNet-50-FPN (blue circles) and ResNet-101-FPN (orange diamonds) at five scales (400-800 pixels).
- Ignoring the low-accuracy regime (AP<25), RetinaNet forms an upper envelope of all current detectors, and an improved variant (not shown) achieves 40.8 AP.

RetinaNet Prediction Pipeline - Comparison

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [16]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [20]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [17]	Inception-ResNet-v2 [34]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [32]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [27]	DarkNet-19 [27]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [22, 9]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [9]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet (ours)	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet (ours)	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2

Table 2. **Object detection** *single-model* results (bounding box AP), *vs.* state-of-the-art on COCO `test-dev`. We show results for ou RetinaNet-101-800 model, trained with scale jitter and for 1.5× longer than the same model from Table 1e. Our model achieves top results outperforming both one-stage and two-stage models. For a detailed breakdown of speed versus accuracy see Table 1e and Figure 2.

Snapshot of Object Detection Methods (till 2019)

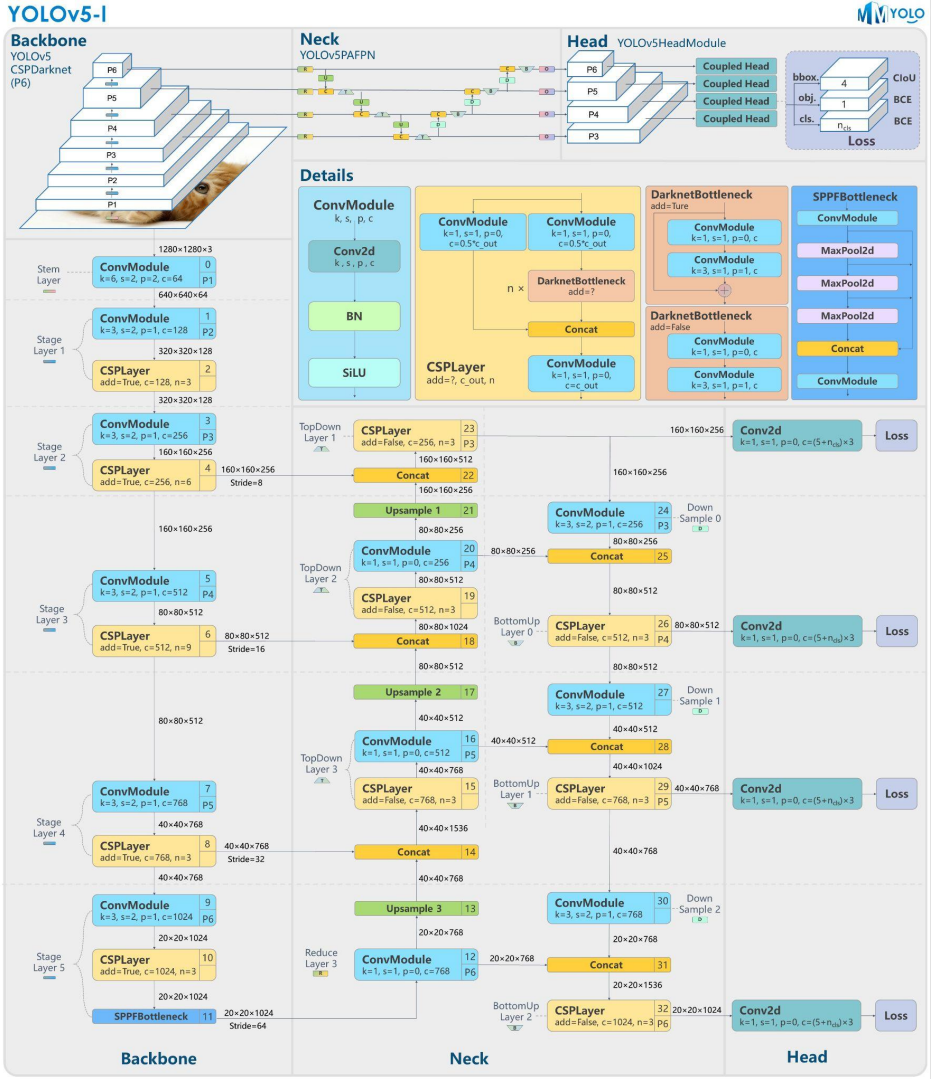


https://github.com/hoya012/deep_learning_object_detection

Object Detection Chapter 25:

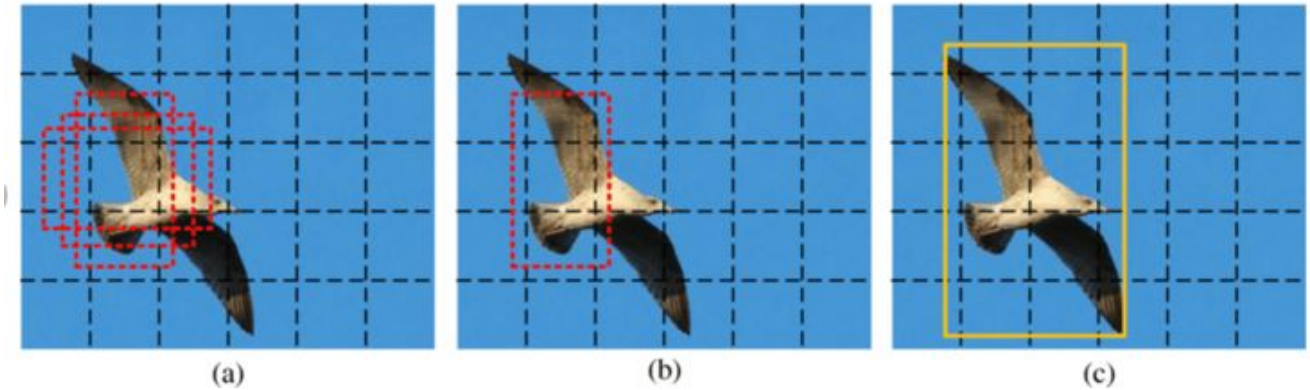
YOLOv5-YOLOv8

YOLOv5



https://user-images.githubusercontent.com/27466624/211143533-1725c1b2-6189-4c3a-a046-ad968e03cb9d.jpg

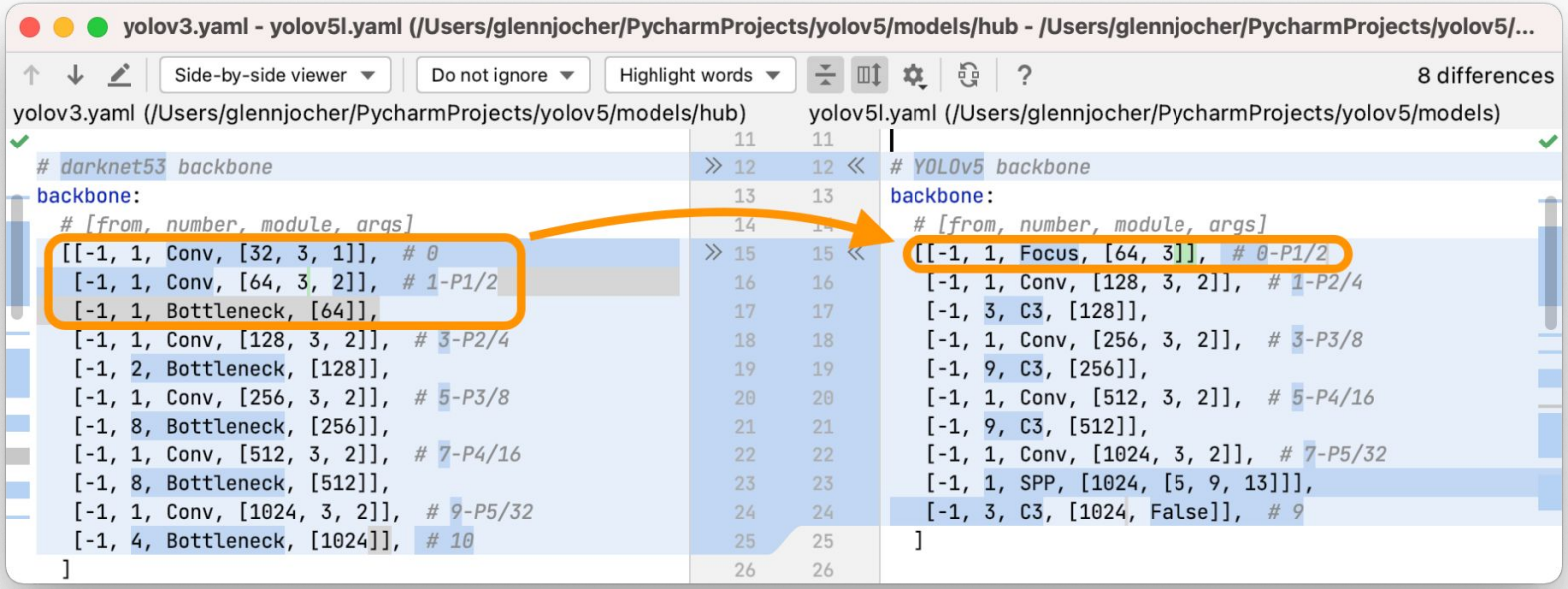
YOLOv5: Adaptive Anchor Box



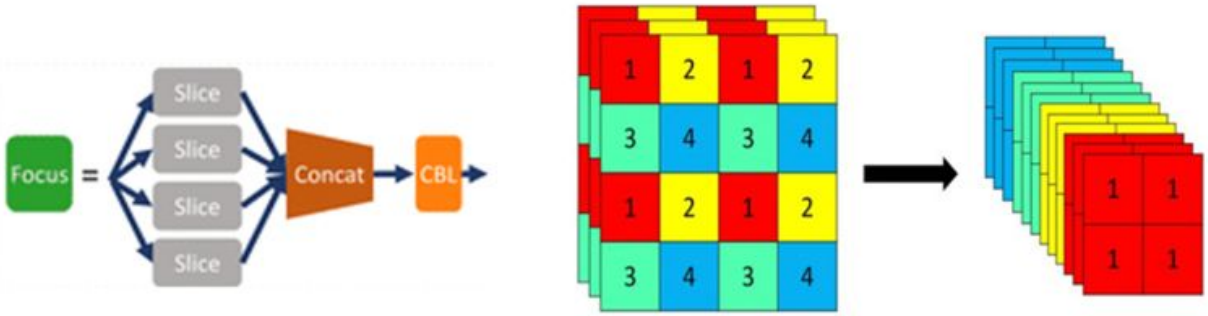
- The optimal anchor box value in different training sets is adaptively calculated during each training.

YOLOv5: Focus Structure

- The YOLOv5 Focus layer replaces the first 3 YOLOv3 layers with a single layer:

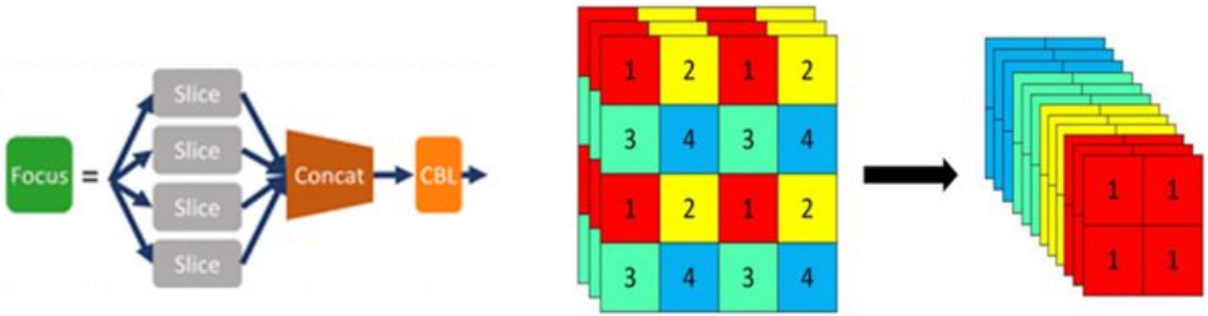


YOLOv5: Focus Structure



- Focus structure slices high-resolution images and splits them into multiple low resolution images.

YOLOv5: Focus Structure




```
class Focus(nn.Module):
    # Focus wh information into c-space
    def __init__(self, c1, c2, k=1, s=1, p=None, g=1, act=True):  # ch_in, ch_out, kernel, stride, padding, groups
        super(Focus, self).__init__()
        self.conv = Conv(c1 * 4, c2, k, s, p, g, act)
        # self.contract = Contract(gain=2)

    def forward(self, x):  # x(b,c,w,h) -> y(b,4c,w/2,h/2)
        return self.conv(torch.cat([x[..., ::2, ::2], x[..., 1::2, ::2], x[..., ::2, 1::2], x[..., 1::2, 1::2]], 1))
```

YOLOv5: C3 Module

```
class C3(nn.Module):
    # CSP Bottleneck with 3 convolutions
    def __init__(self, c1, c2, n=1, shortcut=True, g=1, e=0.5):  # ch_in, ch_out, number, shortcut, groups, expansion
        super().__init__()
        c_ = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = Conv(c1, c_, 1, 1)
        self.cv3 = Conv(2 * c_, c2, 1)  # optional act=FReLU(c2)
        self.m = nn.Sequential(*(Bottleneck(c_, c_, shortcut, g, e=1.0) for _ in range(n)))

    def forward(self, x):
        return self.cv3(torch.cat((self.m(self.cv1(x)), self.cv2(x)), 1))
```

https://github.com/ultralytics/yolov5/blob/3c0a6e664bc3847ab9cca3df66195de6acfeb012/models/common.py#L156

YOLOv5: Bottleneck Module

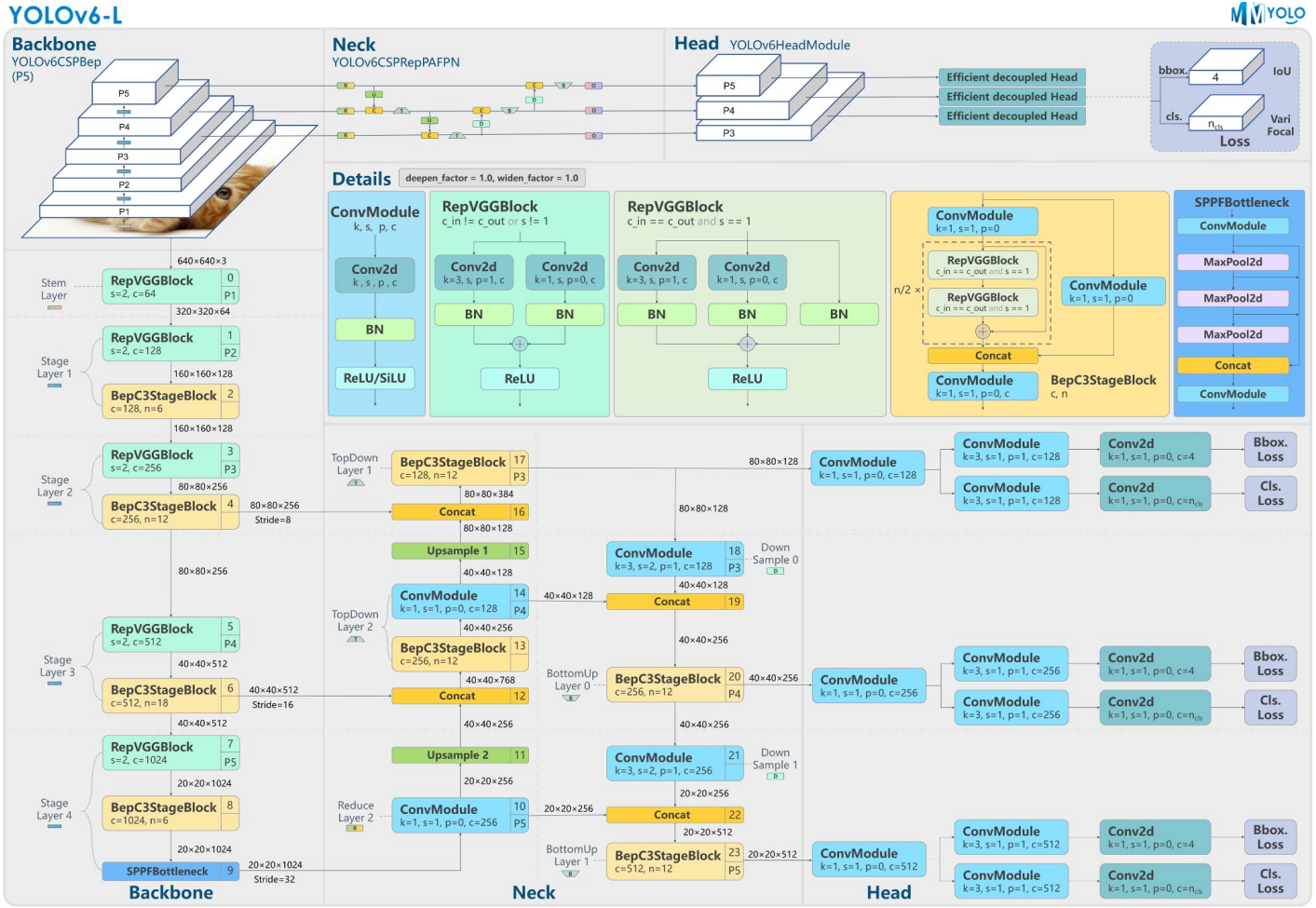
```
class Bottleneck(nn.Module):
    # Standard bottleneck
    def __init__(self, c1, c2, shortcut=True, g=1, k=(3, 3), e=0.5):  # ch_in, ch_out, shortcut, groups, kernels, expand
        super().__init__()
        c_ = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, k[0], 1)
        self.cv2 = Conv(c_, c2, k[1], 1, g=g)
        self.add = shortcut and c1 == c2

    def forward(self, x):
        return x + self.cv2(self.cv1(x)) if self.add else self.cv2(self.cv1(x))
```


https://github.com/ultralytics/yolov5/blob/3c0a6e664bc3847ab9cca3df66195de6acfeb012/models/common.py#L156

YOLOv5: Focal Loss

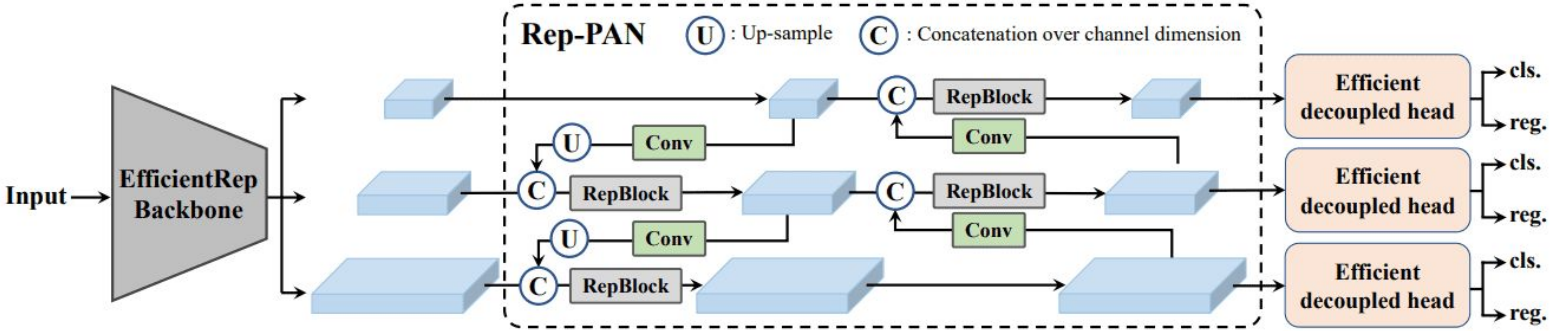
- YOLOv5 optionally allows training via focal loss.

YOLOv6



https://user-images.githubusercontent.com/58845482/209787949-d57691c0-a2ea-4a0a-829f-e8a64ac29c7e.png

YOLOv6: Changes



- **Backbone:** VGG used are used during training, which are replaced with simple 3x3 convolutions during inference.
- **Neck:** PAN that concatenates features from various reparameterized blocks, and is as such called Rep-PAN.
- **Head:** An efficient decoupled head is used: Classification and detection branches do not share parameters.
- **Classification Loss:** Varifocal loss.

YOLOv6: RepVGG

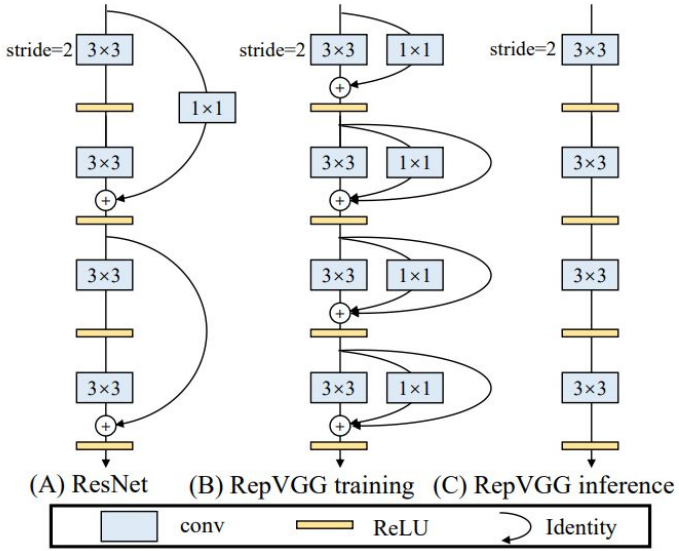
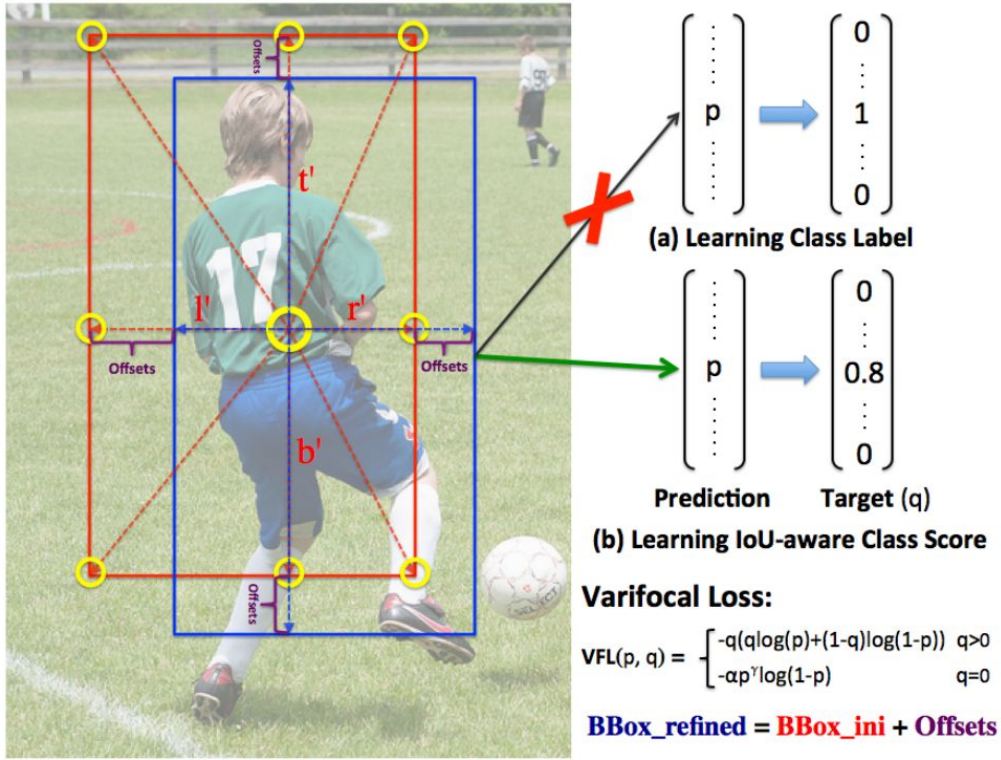


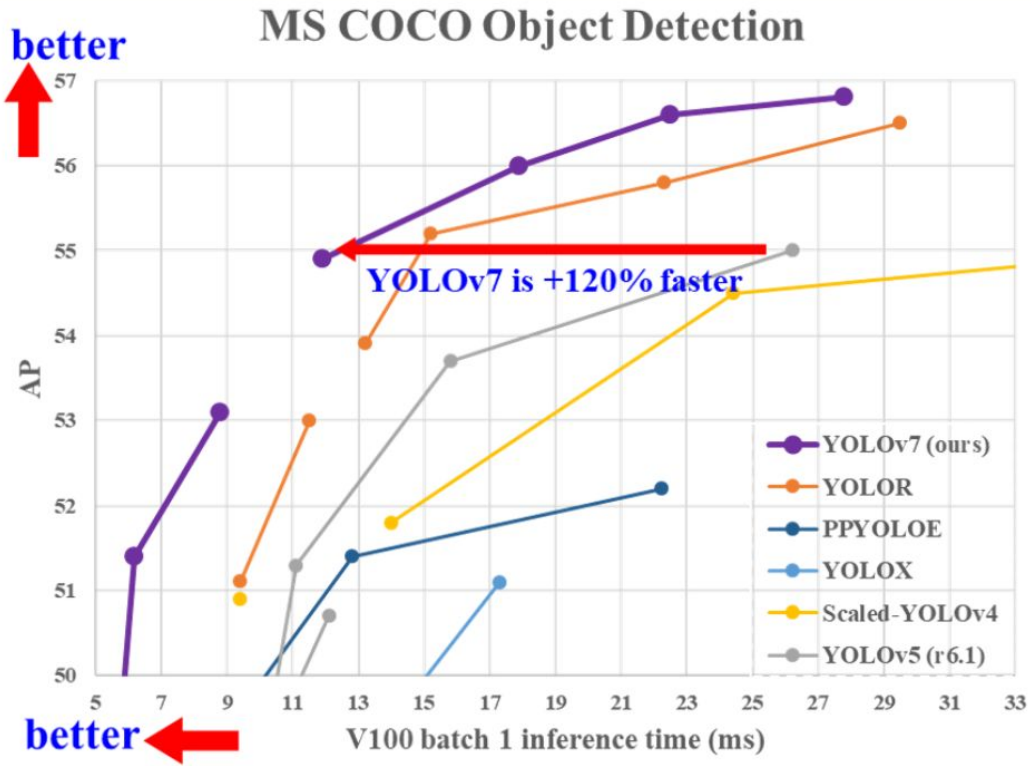
Figure 2: Sketch of RepVGG architecture. RepVGG has 5 stages and conducts down-sampling via stride-2 convolution at the beginning of a stage. Here we only show the first 4 layers of a specific stage. As inspired by ResNet [12], we also use identity and 1×1 branches, but only for training.

https://arxiv.org/abs/2101.03697

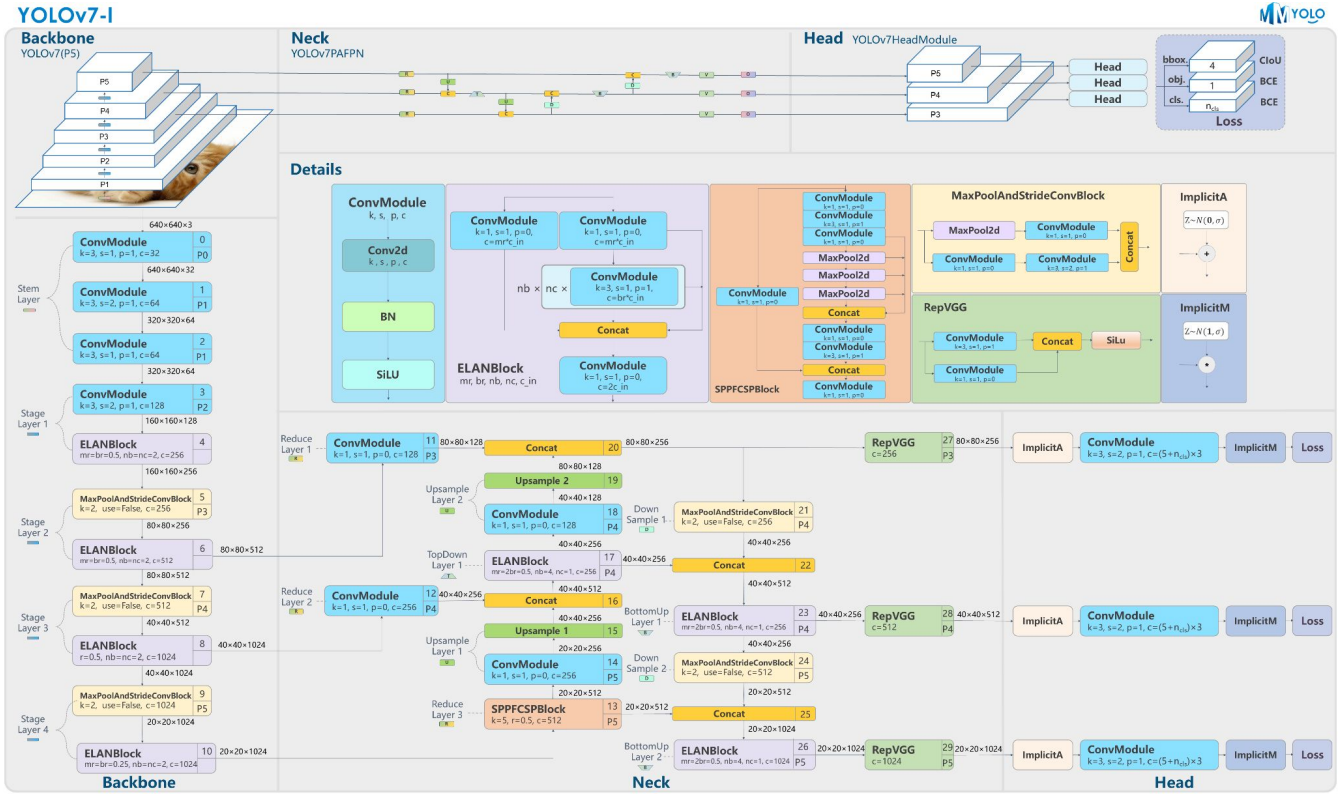
YOLOv6: Varifocal Loss



YOLOv7


MS COCO Object Detection
better
AP
57
56
55
54
53
52
51
50
YOLOv7 is +120% faster
YOLOv7 (ours)
YOLOR
PPYOLOE
YOLOX
Scaled-YOLOv4
YOLOv5 (r6.1)
5
7
9
11
13
15
17
19
21
23
25
27
29
31
33
better
V100 batch 1 inference time (ms)

YOLOv7



https://user-images.githubusercontent.com/68552295/216335336-963bd03a-71f3-4556-97af-18b20d69e065.png

YOLOv7: Efficient Layer Aggregation Networks (ELAN)

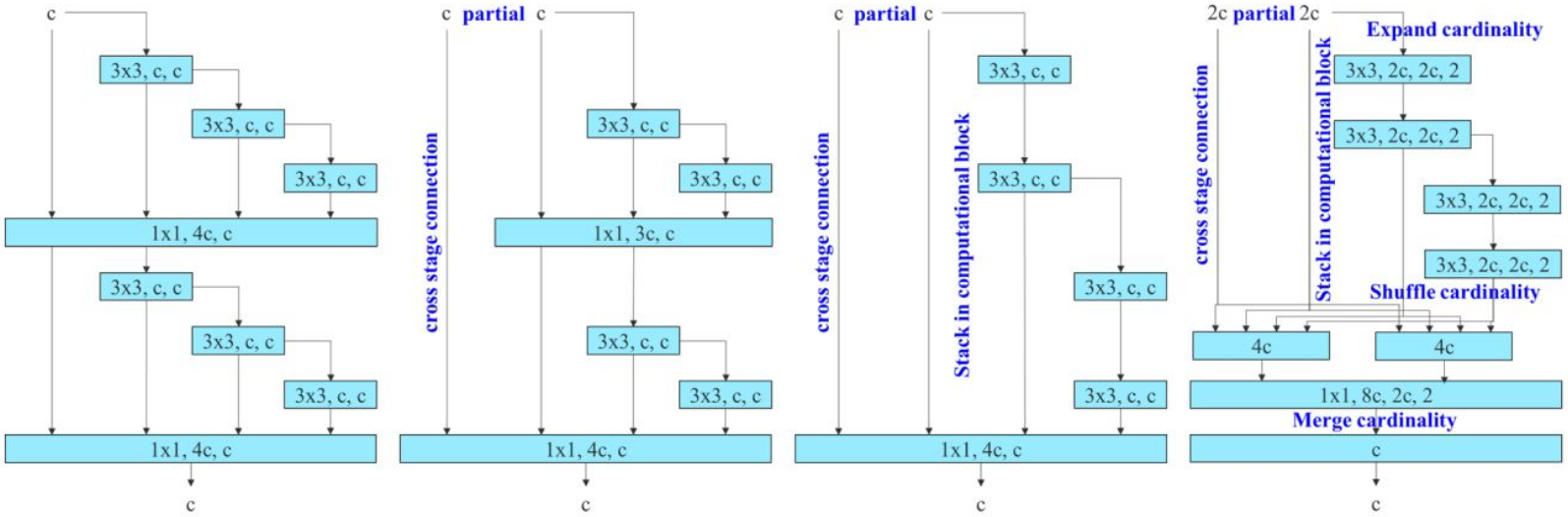
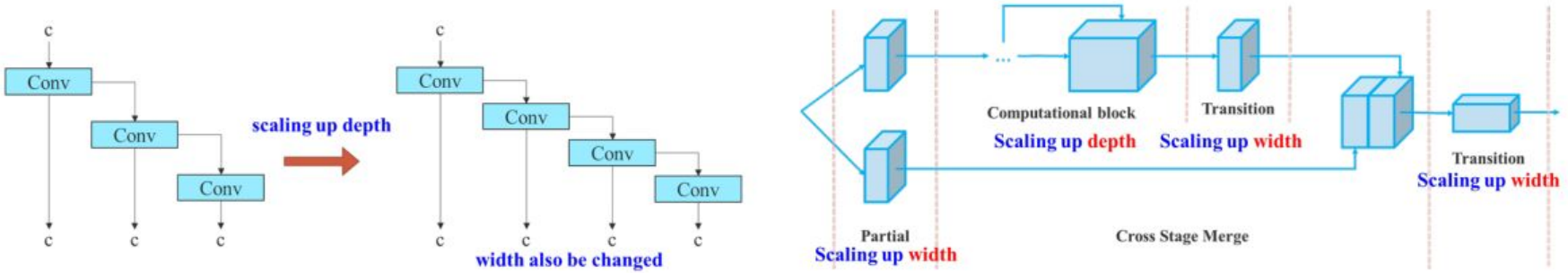


Figure 2: Extended efficient layer aggregation networks. The proposed extended ELAN (E-ELAN) does not change the gradient transmission path of the original architecture at all, but use group convolution to increase the cardinality of the added features, and combine the features of different groups in a shuffle and merge cardinality manner. This way of operation can enhance the features learned by different feature maps and improve the use of parameters and calculations.

https://github.com/WongKinYiu/yolov7/issues/762

YOLOv7: Model Scaling



(a) concatenation-based model (b) scaled-up concatenation-based model (c) compound scaling up depth and width for concatenation-based model

Figure 3: Model scaling for concatenation-based models. From (a) to (b), we observe that when depth scaling is performed on concatenation-based models, the output width of a computational block also increases. This phenomenon will cause the input width of the subsequent transmission layer to increase. Therefore, we propose (c), that is, when performing model scaling on concatenation-based models, only the depth in a computational block needs to be scaled, and the remaining of transmission layer is performed with corresponding width scaling.

YOLOv7: Deep Supervision

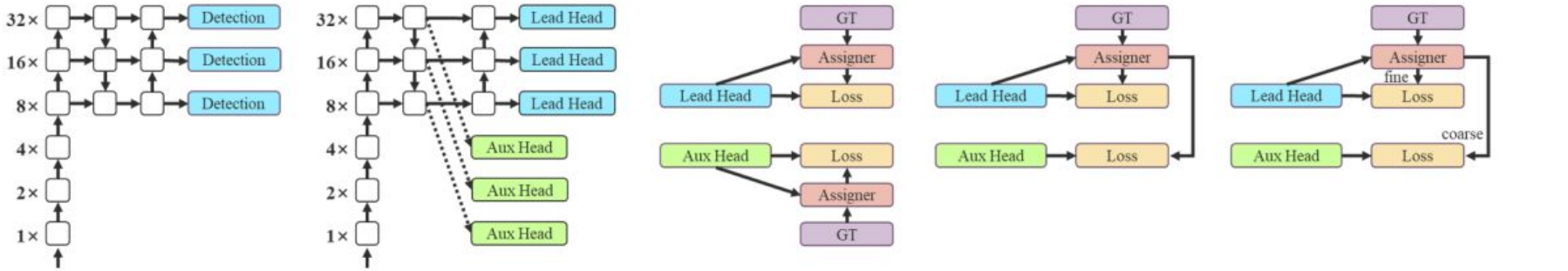
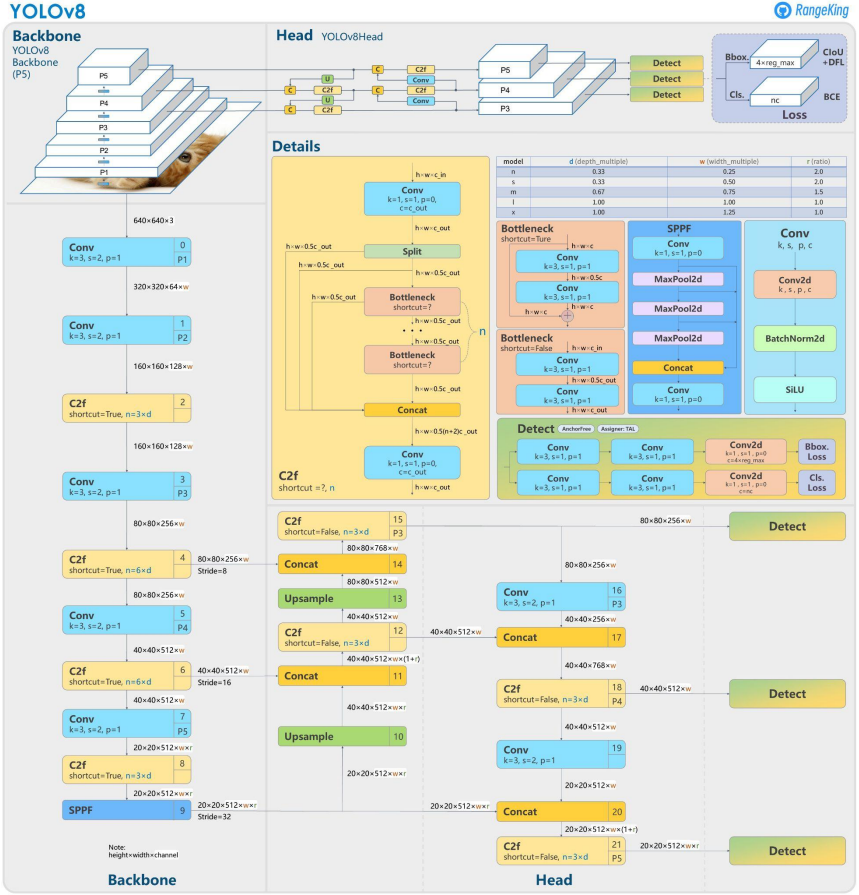


Figure 5: Coarse for auxiliary and fine for lead head label assigner. Compare with normal model (a), the schema in (b) has auxiliary head. Different from the usual independent label assigner (c), we propose (d) lead head guided label assigner and (e) coarse-to-fine lead head guided label assigner. The proposed label assigner is optimized by lead head prediction and the ground truth to get the labels of training lead head and auxiliary head at the same time. The detailed coarse-to-fine implementation method and constraint design details will be elaborated in Apendix.

YOLOv8



https://user-images.githubusercontent.com/27466624/222874205-3873bdac-7135-4ecc-8ab2-ca18b8e13fdf.jpg

YOLOv8 vs YOLOv5

- C3 module (3 layer CSP) is replaced with C2f.
- The first 6x6 Conv is replaced with a 3x3 Conv in the backbone.
- Two Convs (No.10 and No.14 in the YOLOv5 config) are removed.
- The first 1x1 Conv is replaced with a 3x3 Conv in the Bottleneck.
- A decoupled head is used and the objectness branch is deleted.

YOLOv8: C2f Module

```python
class C2f(nn.Module):
    # CSP Bottleneck with 2 convolutions
    def __init__(self, c1, c2, n=1, shortcut=False, g=1, e=0.5):  # ch_in, ch_out, number, shortcut, groups, expansion
        super().__init__()
        self.c = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, 2 * self.c, 1, 1)
        self.cv2 = Conv((2 + n) * self.c, c2, 1)  # optional act=FReLU(c2)
        self.m = nn.ModuleList(Bottleneck(self.c, self.c, shortcut, g, k=((3, 3), (3, 3)), e=1.0) for _ in range(n))

    def forward(self, x):
        y = list(self.cv1(x).chunk(2, 1))
        y.extend(m(y[-1]) for m in self.m)
        return self.cv2(torch.cat(y, 1))

    def forward_split(self, x):
        y = list(self.cv1(x).split((self.c, self.c), 1))
        y.extend(m(y[-1]) for m in self.m)
        return self.cv2(torch.cat(y, 1))
```

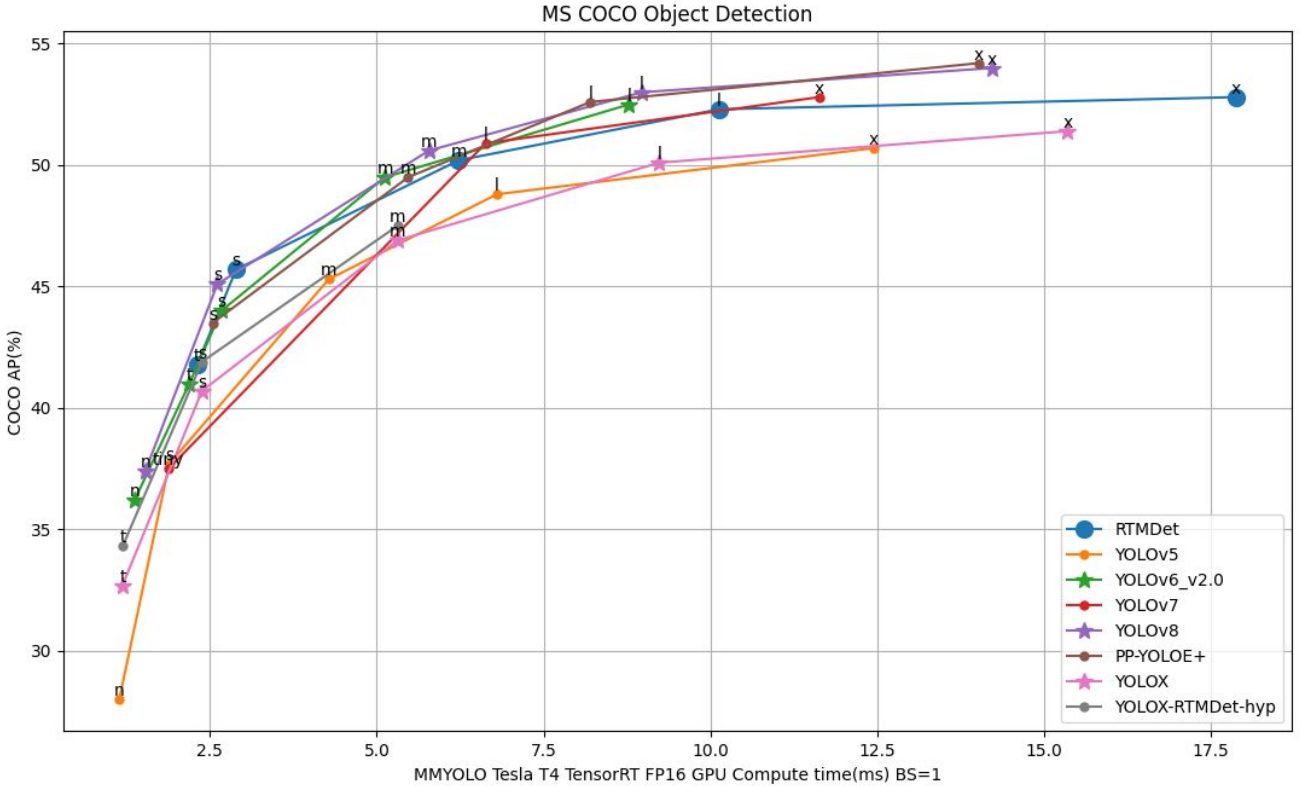
https://github.com/ultralytics/ultralytics/blob/main/ultralytics/nn/modules.py

YOLOv5: C3 Module

```
class C3(nn.Module):
    # CSP Bottleneck with 3 convolutions
    def __init__(self, c1, c2, n=1, shortcut=True, g=1, e=0.5):  # ch_in, ch_out, number, shortcut, groups, expansion
        super().__init__()
        c_ = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = Conv(c1, c_, 1, 1)
        self.cv3 = Conv(2 * c_, c2, 1)  # optional act=FReLU(c2)
        self.m = nn.Sequential(*(Bottleneck(c_, c_, shortcut, g, e=1.0) for _ in range(n)))

    def forward(self, x):
        return self.cv3(torch.cat((self.m(self.cv1(x)), self.cv2(x)), 1))
```

https://github.com/ultralytics/yolov5/blob/3c0a6e664bc3847ab9cca3df66195de6acfeb012/models/common.py#L156

YOLOv8



https://github.com/open-mmlab/mmyolo