1

Object Detection

Wei-Ta Chu

R-CNN

Wei-Ta Chu

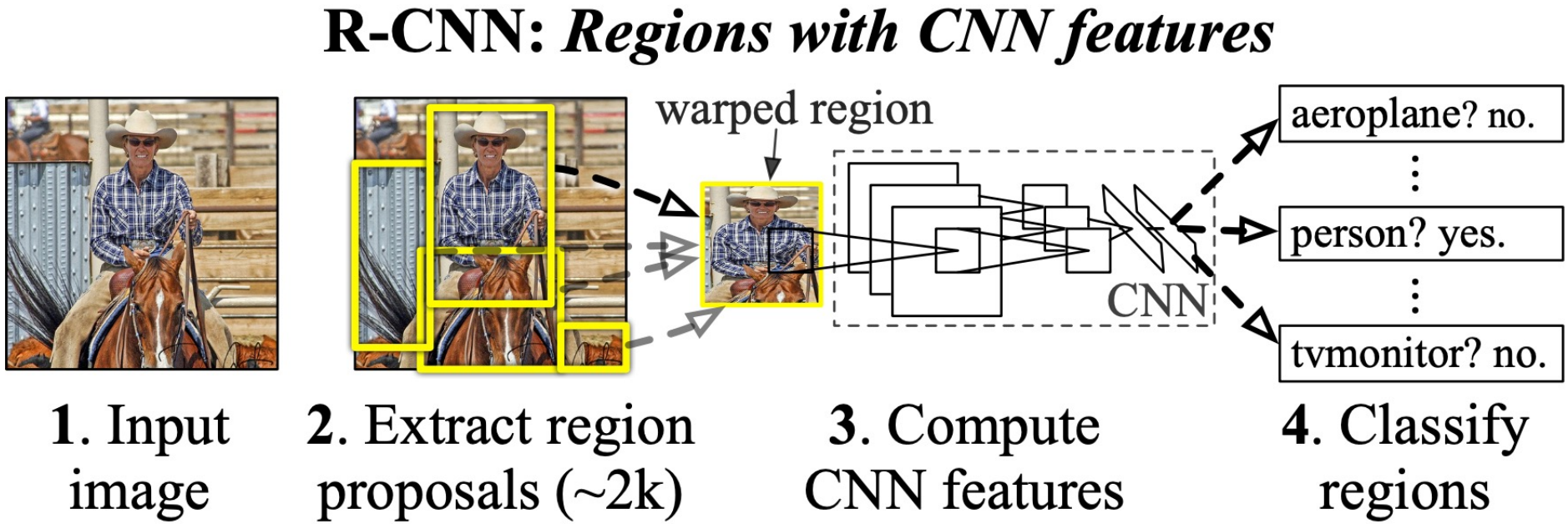
R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” CVPR, 2014.

Introduction

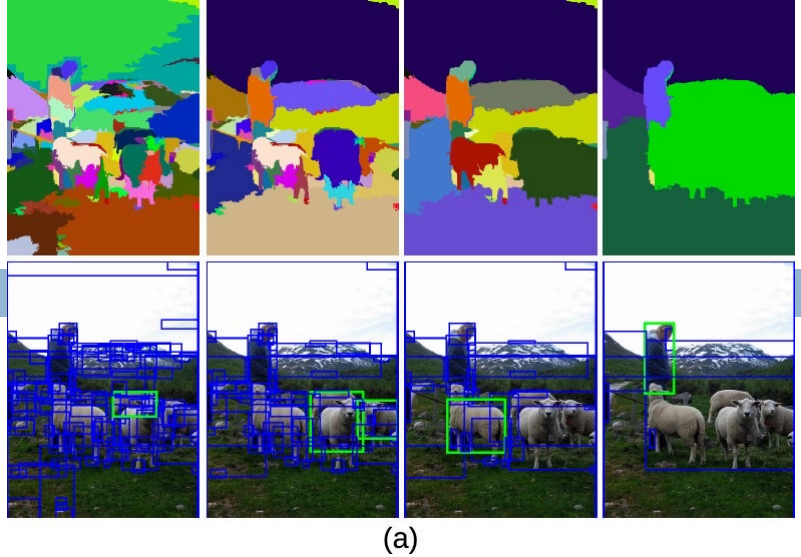
- First paper to show a CNN can lead much higher object detection performance on PASCAL VOC.
- Two problems:
  - Localizing objects
  - Training a high-capacity model with small data
- Ideas:
  - Generate category-independent region proposals
  - Extracts features from each proposal using a CNN
  - Classify each region with linear SVMs

Introduction

- R-CNN



Region Proposals



- Selective search
  - Create initial regions.
  - Use a greedy algorithm to iteratively group regions together until the whole image becomes a single region.
- Diversification Strategies
  - using a variety of color spaces
  - using different similarity measures
  - varying starting regions

J. Uijlings, K.van de Sande, T. Gevers, and A. Smeulders, “Selective search for object recognition,” IJCV, 2013.

Feature Extraction

- 4096-dimensional feature vector extracted by AlexNet
- Warp all pixels in a tight bounding box around it to the required size (227 x 227)

A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In NIPS, 2012.

Test-Time Detection

- Run selective search on the test image to extract around 2000 region proposals
- Feed to the CNN to extract features
- For each class, score each extracted feature vector using the SVM trained for that class
- Given all scored regions, we apply a greedy non-maximum suppression that rejects a region if it has an intersection-over-union (IoU) overlap with a higher scoring selected region

Experiments

- PASCAL VOC 2010

VOC 2010 test	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
DPM v5 [17]†	49.2	53.8	13.1	15.3	35.5	53.4	49.7	27.0	17.2	28.8	14.7	17.8	46.4	51.2	47.7	10.8	34.2	20.7	43.8	38.3	33.4
UVA [32]	56.2	42.4	15.3	12.6	21.8	49.3	36.8	46.1	12.9	32.1	30.0	36.5	43.5	52.9	32.9	15.3	41.1	31.8	47.0	44.8	35.1
Regionlets [35]	65.0	48.9	25.9	24.6	24.5	56.1	54.5	51.2	17.0	28.9	30.2	35.8	40.2	55.7	43.5	14.3	43.9	32.6	54.0	45.9	39.7
SegDPM [15]†	61.4	53.4	25.6	25.2	35.5	51.7	50.6	50.8	19.3	33.8	26.8	40.4	48.3	54.4	47.1	14.8	38.7	35.0	52.8	43.1	40.4
R-CNN	67.1	64.1	46.7	32.0	30.5	56.4	57.2	65.9	27.0	47.3	40.9	66.6	57.8	65.9	53.6	26.7	56.5	38.1	52.8	50.2	50.2
R-CNN BB	**71.8**	**65.8**	**53.0**	**36.8**	**35.9**	**59.7**	**60.0**	**69.9**	**27.9**	**50.6**	**41.4**	**70.0**	**62.0**	**69.0**	**58.1**	**29.5**	**59.4**	**39.3**	**61.2**	**52.4**	**53.7**

Table 1: Detection average precision (%) on VOC 2010 test. R-CNN is most directly comparable to UVA and Regionlets since all methods use selective search region proposals. Bounding box regression (BB) is described in Section 3.4. At publication time, SegDPM was the top-performer on the PASCAL VOC leaderboard. †DPM and SegDPM use context rescoring not used by the other methods.

Experiments

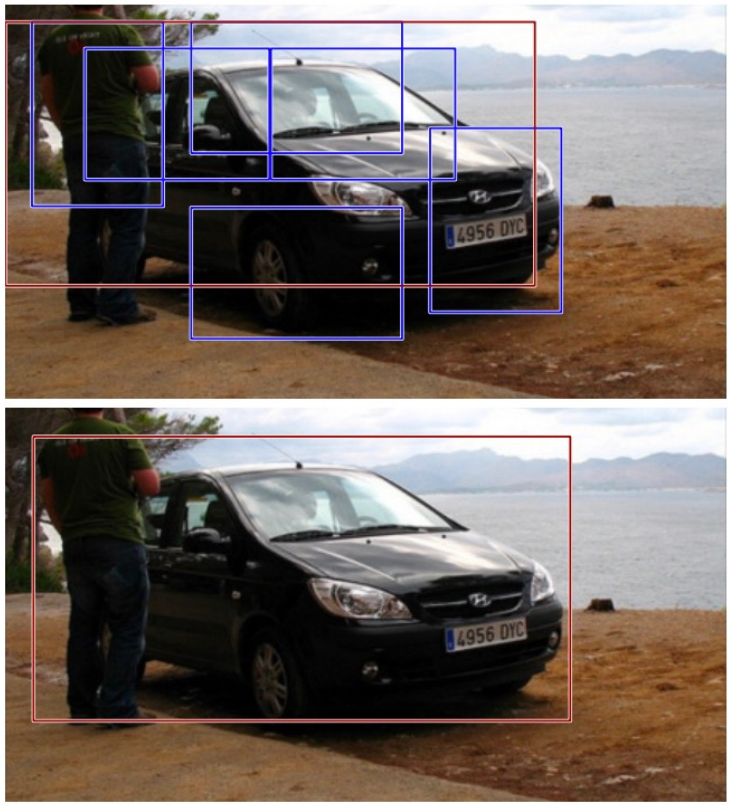
- PASCAL VOC 2007 – Ablation Study

VOC 2007 test	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
R-CNN $pool_5$	51.8	60.2	36.4	27.8	23.2	52.8	60.6	49.2	18.3	47.8	44.3	40.8	56.6	58.7	42.4	23.4	46.1	36.7	51.3	55.7	44.2
R-CNN fc_6	59.3	61.8	43.1	34.0	25.1	53.1	60.6	52.8	21.7	47.8	42.7	47.8	52.5	58.5	44.6	25.6	48.3	34.0	53.1	58.0	46.2
R-CNN fc_7	57.6	57.9	38.5	31.8	23.7	51.2	58.9	51.4	20.0	50.5	40.9	46.0	51.6	55.9	43.3	23.3	48.1	35.3	51.0	57.4	44.7
R-CNN FT $pool_5$	58.2	63.3	37.9	27.6	26.1	54.1	66.9	51.4	26.7	55.5	43.4	43.1	57.7	59.0	45.8	28.1	50.8	40.6	53.1	56.4	47.3
R-CNN FT fc_6	63.5	66.0	47.9	37.7	29.9	62.5	70.2	60.2	32.0	57.9	47.0	53.5	60.1	64.2	52.2	31.3	55.0	50.0	57.7	63.0	53.1
R-CNN FT fc_7	64.2	69.7	50.0	41.9	32.0	62.6	71.0	60.7	32.7	58.5	46.5	56.1	60.6	66.8	54.2	31.5	52.8	48.9	57.9	64.7	54.2
R-CNN FT fc_7 BB	**68.1**	**72.8**	**56.8**	**43.0**	**36.8**	**66.3**	**74.2**	**67.6**	**34.4**	**63.5**	**54.5**	**61.2**	**69.1**	**68.6**	**58.7**	**33.4**	**62.9**	**51.1**	**62.5**	**64.8**	**58.5**
DPM v5 [17]	33.2	60.3	10.2	16.1	27.3	54.3	58.2	23.0	20.0	24.1	26.7	12.7	58.1	48.2	43.2	12.0	21.1	36.1	46.0	43.5	33.7
DPM ST [25]	23.8	58.2	10.5	8.5	27.1	50.4	52.0	7.3	19.2	22.8	18.1	8.0	55.9	44.8	32.4	13.3	15.9	22.8	46.2	44.9	29.1
DPM HSC [27]	32.2	58.3	11.5	16.3	30.6	49.9	54.8	23.5	21.5	27.7	34.0	13.7	58.1	51.6	39.9	12.4	23.5	34.4	47.4	45.2	34.3

Table 2: Detection average precision (%) on VOC 2007 test. Rows 1-3 show R-CNN performance without fine-tuning. Rows 4-6 show results for the CNN pre-trained on ILSVRC 2012 and then fine-tuned (FT) on VOC 2007 trainval. Row 7 includes a simple bounding box regression (BB) stage that reduces localization errors (Section 3.4). Rows 8-10 present DPM methods as a strong baseline. The first uses only HOG, while the next two use different feature learning approaches to augment or replace HOG.

Experiments

- Bounding box regression
  - Inspired by the bounding box regression employed in DPM, we train a linear regression model to predict a new detection window given the pool5 features for a selective search region proposal.
  - This simple approach fixes a large number of mislocalized detections, boosting mAP by 3 to 4 points.



Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester and Deva Ramanan, "Object Detection with Discriminatively Trained Part Based Models," IEEE Trans. on PAMI, 2010.

Fast R-CNN

Wei-Ta Chu

R. Girshick, “Fast R-CNN,” ICCV, 2015.

Introduction

- Drawbacks of R-CNN
  - Training is a multi-stage pipeline.
  - Training is expensive in space and time.
  - Object detection is slow.
- R-CNN is slow because it performs a ConvNet forward pass for each object proposal, without sharing computation.

Fast R-CNN Architecture

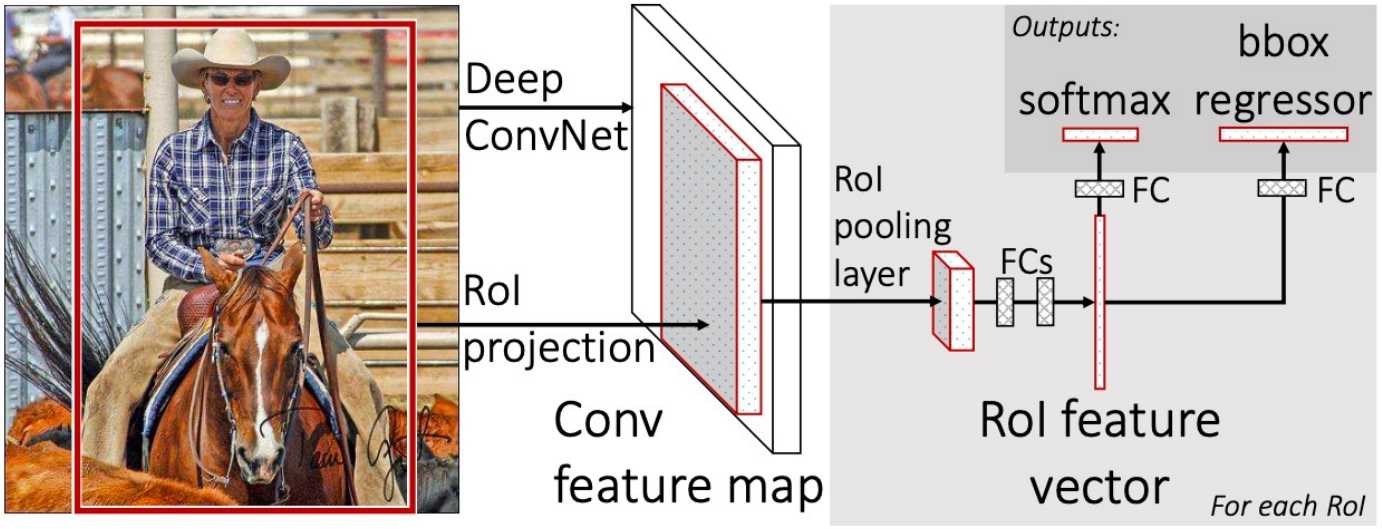


Figure 1. Fast R-CNN architecture. An input image and multiple regions of interest (RoIs) are input into a fully convolutional network. Each RoI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per RoI: softmax probabilities and per-class bounding-box regression offsets. The architecture is trained end-to-end with a multi-task loss.

Fast R-CNN Architecture

- Input: an entire image and a set of object proposals.
- First processes the whole image with several convolutional (conv) and max pooling layers to produce a conv feature map.
- For each object proposal a region of interest (RoI) pooling layer extracts a fixed-length feature vector from the feature map.
- Each feature vector is fed into a sequence of fully connected (fc) layers that finally branch into two sibling output layers.

The ROI Pooling Layer

- The RoI pooling layer uses max pooling to convert the features inside any valid region of interest into a small feature map with a fixed spatial extent of H × W
- RoI max pooling works by dividing the h × w RoI window into an H × W grid of sub-windows of approximate size h/H × w/W and then max-pooling the values in each sub-window into the corresponding output grid cell.

Multi-Task Loss

- Two branches: object classification and location regression
- Loss: $L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda[u \geq 1] L_{\text{loc}}(t^u, v),$

$L_{\text{cls}}(p, u) = -\log p_u$ is log loss for true class u.

$$L_{\text{loc}}(t^u, v) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t_i^u - v_i),$$

in which

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

Fast R-CNN Detection

- Once a Fast R-CNN is fine-tuned, detection amounts to little more than running a forward pass (assuming object proposals are pre-computed).
- For each test RoI *r*, the forward pass outputs a class posterior probability distribution *p* and a set of predicted bounding-box offsets relative to *r*.
- non-maximum suppression

Fast R-CNN Detection

- For detection the number of RoIs to process is large and nearly half of the forward pass time is spent computing the fully connected layers
- Factorize the weight matrix W using SVD

$$W \approx U\Sigma_t V^T$$

- To compress a network, the single fully connected layer corresponding to W is replaced by two fully connected layers. The first of these layers uses the weight matrix $\Sigma_t V^T$ and the second uses U.

Experiments

- Detection accuracy in terms of mAP

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
SPPnet BB [11]†	07 \ diff	73.9	72.3	62.5	51.5	44.4	74.4	73.0	74.4	42.3	73.6	57.7	70.3	74.6	74.3	54.2	34.0	56.4	56.4	67.9	73.5	63.1
R-CNN BB [10]	07	73.4	77.0	63.4	45.4	**44.6**	75.1	78.1	79.8	40.5	73.7	62.2	79.4	78.1	73.1	64.2	**35.6**	66.8	67.2	70.4	**71.1**	66.0
FRCN [ours]	07	74.5	78.3	69.2	53.2	36.6	77.3	78.2	82.0	40.7	72.7	67.9	79.6	79.2	73.0	69.0	30.1	65.4	70.2	75.8	65.8	66.9
FRCN [ours]	07 \ diff	74.6	**79.0**	68.6	57.0	39.3	79.5	**78.6**	81.9	**48.0**	74.0	67.4	80.5	80.7	74.1	69.6	31.8	67.1	68.4	75.3	65.5	68.1
FRCN [ours]	07+12	**77.0**	78.1	**69.3**	**59.4**	38.3	**81.6**	**78.6**	**86.7**	42.8	**78.8**	**68.9**	**84.7**	**82.0**	**76.6**	**69.9**	31.8	**70.1**	**74.8**	**80.4**	70.4	**70.0**

Table 1. **VOC 2007 test** detection average precision (%). All methods use VGG16. Training set key: **07**: VOC07 trainval, **07 \ diff**: **07** without "difficult" examples, **07+12**: union of **07** and VOC12 trainval. †SPPnet results were prepared by the authors of [11].

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
BabyLearning	Prop.	77.7	73.8	62.3	48.8	45.4	67.3	67.0	80.3	41.3	70.8	49.7	79.5	74.7	78.6	64.5	36.0	69.9	55.7	70.4	61.7	63.8
R-CNN BB [10]	12	79.3	72.4	63.1	44.0	44.4	64.6	66.3	84.9	38.8	67.3	48.4	82.3	75.0	76.7	65.7	35.8	66.2	54.8	69.1	58.8	62.9
SegDeepM	12+seg	**82.3**	75.2	67.1	50.7	**49.8**	71.1	69.6	88.2	42.5	71.2	50.0	85.7	76.6	81.8	69.3	**41.5**	**71.9**	62.2	73.2	**64.6**	67.2
FRCN [ours]	12	80.1	74.4	67.7	49.4	41.4	74.2	68.8	87.8	41.9	70.1	50.2	86.1	77.3	81.1	70.4	33.3	67.0	63.3	77.2	60.0	66.1
FRCN [ours]	07++12	82.0	**77.8**	**71.6**	**55.3**	42.4	**77.3**	**71.7**	**89.3**	**44.5**	**72.1**	**53.7**	**87.7**	**80.0**	**82.5**	**72.7**	36.6	68.7	**65.4**	**81.1**	62.7	**68.8**

Table 2. **VOC 2010 test** detection average precision (%). BabyLearning uses a network based on [17]. All other methods use VGG16. Training set key: **12**: VOC12 trainval, **Prop.**: proprietary dataset, **12+seg**: **12** with segmentation annotations, **07++12**: union of VOC07 trainval, VOC07 test, and VOC12 trainval.

Experiments

- Detection accuracy in terms of mAP

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
BabyLearning	Prop.	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	63.2
NUS_NIN_c2000	Unk.	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	63.8
R-CNN BB [10]	12	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	62.4
FRCN [ours]	12	80.3	74.7	66.9	46.9	37.7	73.9	68.6	87.7	41.7	71.1	51.1	86.0	77.8	79.8	69.8	32.1	65.5	63.8	76.4	61.7	65.7
FRCN [ours]	07++12	**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**	**68.4**

Table 3. **VOC 2012 test** detection average precision (%). BabyLearning and NUS_NIN_c2000 use networks based on [17]. All other methods use VGG16. Training set key: see Table 2, **Unk.**: unknown.

Experiments

- Training and testing time

S: AlexNet
M: VGG_M
L: VGG16

	Fast R-CNN			R-CNN			SPPnet
	S	**M**	**L**	**S**	**M**	**L**	†**L**
train time (h)	**1.2**	2.0	9.5	22	28	84	25
train speedup	**18.3**×	14.0×	8.8×	1×	1×	1×	3.4×
test rate (s/im)	0.10	0.15	0.32	9.8	12.1	47.0	2.3
▷ with SVD	**0.06**	0.08	0.22	-	-	-	-
test speedup	98×	80×	146×	1×	1×	1×	20×
▷ with SVD	169×	150×	**213**×	-	-	-	-
VOC07 mAP	57.1	59.2	**66.9**	58.5	60.2	66.0	63.1
▷ with SVD	56.5	58.7	66.6	-	-	-	-

Table 4. Runtime comparison between the same models in Fast R-CNN, R-CNN, and SPPnet. Fast R-CNN uses single-scale mode. SPPnet uses the five scales specified in [11]. †Timing provided by the authors of [11]. Times were measured on an Nvidia K40 GPU.

Experiments

- Truncated SVD can reduce detection time by more than 30% with only a small (0.3 percent- age point) drop in mAP

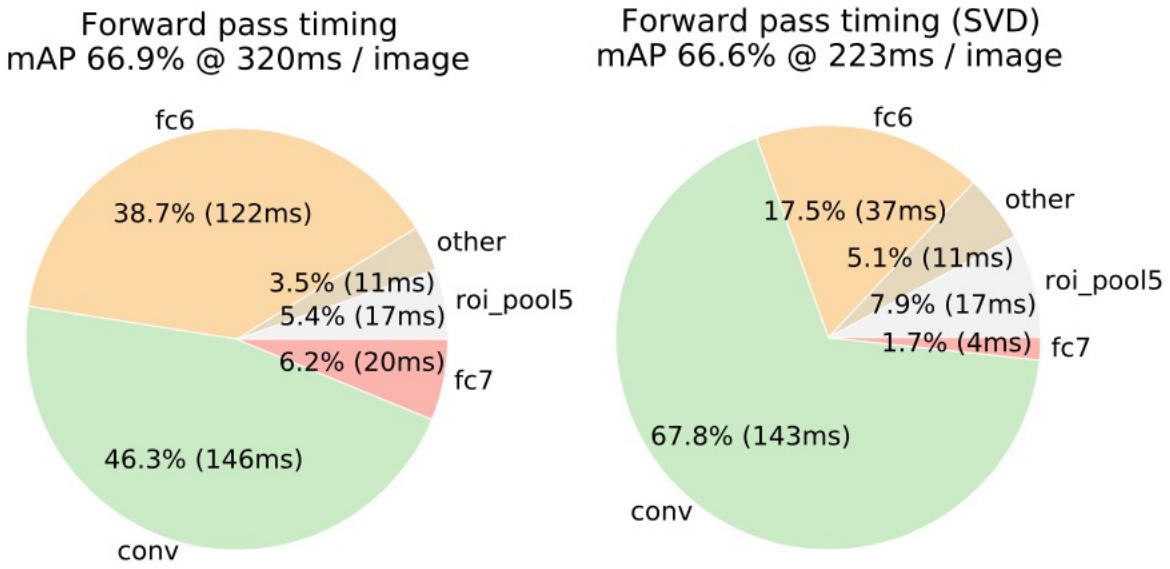


Figure 2. Timing for VGG16 before and after truncated SVD. Before SVD, fully connected layers fc6 and fc7 take 45% of the time.

Experiments

- Does multi-task training help?

	S				M				L			
multi-task training?		✓		✓		✓		✓		✓		✓
stage-wise training?			✓				✓				✓	
test-time bbox reg?			✓	✓			✓	✓			✓	✓
VOC07 mAP	52.2	53.3	54.6	**57.1**	54.7	55.5	56.6	**59.2**	62.6	63.4	64.0	**66.9**

Table 6. Multi-task training (forth column per group) improves mAP over piecewise training (third column per group).

- Do we need more training data?
  - We augment the VOC07 trainval set with the VOC12 trainval set, roughly tripling the number of images to 16.5k. Enlarging the training set improves mAP on VOC07 test from 66.9% to 70.0% (Table 1).

Faster R-CNN

Wei-Ta Chu

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” NIPS, 2015.

Introduction

- Proposals are the computational bottleneck of Fast R-CNN.
- Selective Search achieves 2s per image in a CPU implementation. EdgeBoxes provides the best tradeoff between proposal quality and speed, at 0.2s per image.
- Fast region-based CNNs take advantage of GPUs, while the region proposal methods are implemented on the CPU, making such runtime comparisons inequitable.

Introduction

- Contribution:
  - A novel Region Proposal Networks (RPNs) that share convolutional layers with state-of-the-art object detection networks.
  - By sharing convolutions at test-time, the marginal cost for computing proposals is small (e.g., 10ms per image).
- Even using VGG, our detection method still has a frame rate of 5fps (including all steps) on a GPU. (73.2% mAP on PASCAL VOC 2007 and 70.4% mAP on 2012).

Introduction

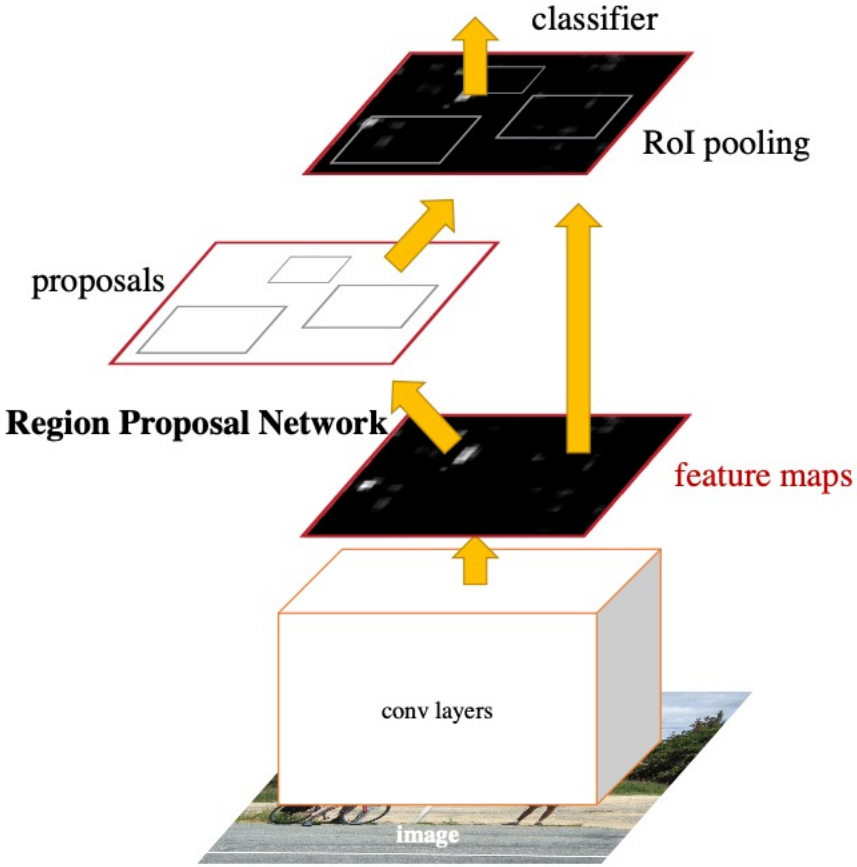


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

Region Proposal Network

- A Region Proposal Network (RPN) takes an image (of any size) as input and outputs a set of rectangular object proposals, each with an objectness score.
- To generate region proposals, we slide a small network over the conv feature map output by the last shared conv layer.

Region Proposal Network

- Multiple scales and sizes

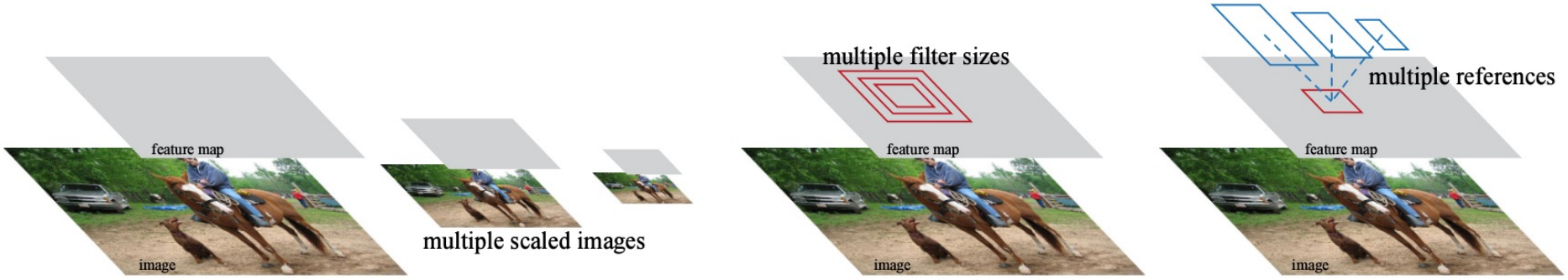
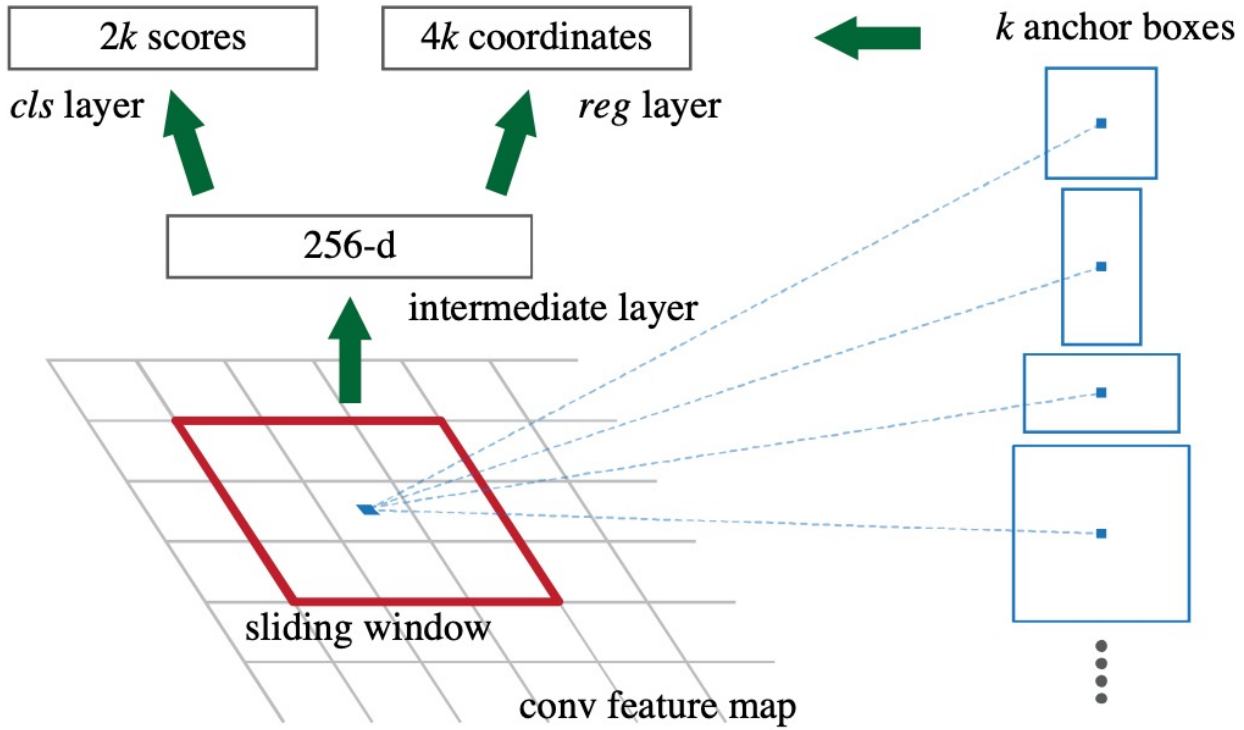


Figure 1: Different schemes for addressing multiple scales and sizes. (a) Pyramids of images and feature maps are built, and the classifier is run at all scales. (b) Pyramids of filters with multiple scales/sizes are run on the feature map. (c) We use pyramids of reference boxes in the regression functions.

Region Proposal Network

- Each sliding window is mapped to a lower-dimensional vector, which is fed into two sibling fully-connected layers—a box-regression layer (reg) and a box-classification layer (cls).



Region Proposal Network

- Translation-Invariant Anchors
- At each sliding-window location, we simultaneously predict k region proposals, so the *reg* layer has 4k outputs encoding the coordinates of k boxes.
- The *cls* layer outputs 2k scores that estimate probability of object / not-object for each proposal.

- The k proposals are parameterized *relative to* k reference boxes, called *anchors*. Each anchor is centered at the sliding window in question, and is associated with a scale and aspect ratio. We use 3 scales and 3 aspect ratios, yielding k = 9 anchors at each sliding position.

Region Proposal Network

- Loss function
- For training RPNs, we assign a binary class label (of being an object or not) to each anchor.

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*)$$

Log loss over two classes

Smooth L1

For regression, we adopt the parameterizations of the 4 coordinates following [6]:

$$t_{\mathrm{x}} = (x - x_{\mathrm{a}})/w_{\mathrm{a}}, \quad t_{\mathrm{y}} = (y - y_{\mathrm{a}})/h_{\mathrm{a}}, \quad t_{\mathrm{w}} = \log(w/w_{\mathrm{a}}), \quad t_{\mathrm{h}} = \log(h/h_{\mathrm{a}}),$$
$$t_{\mathrm{x}}^* = (x^* - x_{\mathrm{a}})/w_{\mathrm{a}}, \quad t_{\mathrm{y}}^* = (y^* - y_{\mathrm{a}})/h_{\mathrm{a}}, \quad t_{\mathrm{w}}^* = \log(w^*/w_{\mathrm{a}}), \quad t_{\mathrm{h}}^* = \log(h^*/h_{\mathrm{a}}),$$

Shared Conv. Features

- For the detection network, we adopt Fast R-CNN.
- Alternating optimization
  - (1) train the RPN
  - (2) train a separate detection network by Fast R-CNN using the proposals generated by the step-1 RPN.
  - (3) use the detector network to initialize RPN training, but we fix the shared conv layers and only fine-tune the layers unique to RPN
  - (4) keeping the shared conv layers fixed, we fine-tune the fc layers of the Fast R-CNN

Experiments

- Trained and tested using various region proposal methods

Table 1: Detection results on **PASCAL VOC 2007 test set** (trained on VOC 2007 trainval). The detectors are Fast R-CNN with ZF, but using various proposal methods for training and testing.

train-time region proposals		**test-time region proposals**		
method	# boxes	method	# proposals	mAP (%)
SS	2k	SS	2k	58.7
EB	2k	EB	2k	58.6
RPN+ZF, shared	2k	RPN+ZF, shared	300	**59.9**
ablation experiments follow below				
RPN+ZF, unshared	2k	RPN+ZF, unshared	300	58.7
SS	2k	RPN+ZF	100	55.1
SS	2k	RPN+ZF	300	56.8
SS	2k	RPN+ZF	1k	56.3
SS	2k	RPN+ZF (no NMS)	6k	55.2
SS	2k	RPN+ZF (no *cls*)	100	44.6
SS	2k	RPN+ZF (no *cls*)	300	51.4
SS	2k	RPN+ZF (no *cls*)	1k	55.8
SS	2k	RPN+ZF (no *reg*)	300	52.1
SS	2k	RPN+ZF (no *reg*)	1k	51.3
SS	2k	RPN+VGG	300	59.2

Experiments

- The effect of sharing conv layers between the RPN and Fast R-CNN detection network
  - We stop after the second step in the 4-step training process. Using separate networks reduces the result slightly to 58.7% (RPN+ZF, unshared, Table 1).
- Disentangle the RPN's influence
  - We train a Fast R-CNN model by using the 2k SS proposals and ZF net. We fix this detector and evaluate the detection mAP by changing the proposal regions used at test-time.
  - Replacing SS with 300 RPN proposals at test-time leads to an mAP of 56.8%. The loss in mAP is because of the inconsistency between the training/testing proposals.

Experiments

- The roles of RPN's *cls* and *reg* outputs
  - When the *cls* layer is removed at test-time, we randomly sample N proposals from the unscored regions. The mAP is nearly unchanged with N = 1k (55.8%), but degrades considerably to 44.6% when N = 100. This shows that the *cls* scores account for the accuracy of the highest ranked proposals.
  - When the *reg* layer is removed at test-time (so the proposals become anchor boxes), the mAP drops to 52.1%. This suggests that the high-quality proposals are mainly due to regressed positions. The anchor boxes alone are not sufficient for accurate detection.

Experiments

- Detection Accuracy and Running Time of VGG-16.

Table 2: Detection results on **PASCAL VOC 2007 test set**. The detector is Fast R-CNN and VGG-16. Training data: "07": VOC 2007 trainval, "07+12": union set of VOC 2007 trainval and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2k. †: this was reported in [5]; using the repository provided by this paper, this number is higher (68.0±0.3 in six runs).

method	# proposals	data	mAP (%)	time (ms)
SS	2k	07	66.9†	1830
SS	2k	07+12	70.0	1830
RPN+VGG, unshared	300	07	68.5	342
RPN+VGG, shared	300	07	69.9	**198**
RPN+VGG, shared	300	07+12	**73.2**	**198**

Experiments

- Detection Accuracy and Running Time of VGG-16.

Table 3: Detection results on **PASCAL VOC 2012 test set**. The detector is Fast R-CNN and VGG-16. Training data: "07": VOC 2007 trainval, "07++12": union set of VOC 2007 trainval+test and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2k. †: `http://host.robots.ox.ac.uk:8080/anonymous/HZJTQA.html`. ‡: `http://host.robots.ox.ac.uk:8080/anonymous/YNPLXB.html`

method	# proposals	data	mAP (%)
SS	2k	12	65.7
SS	2k	07++12	68.4
RPN+VGG, shared†	300	12	67.0
RPN+VGG, shared‡	300	07++12	**70.4**

Experiments

- Detection Accuracy and Running Time of VGG-16.

Table 4: **Timing** (ms) on a K40 GPU, except SS proposal is evaluated in a CPU. "Region-wise" includes NMS, pooling, fc, and softmax. See our released code for the profiling of running time.

model	system	conv	proposal	region-wise	total	rate
VGG	SS + Fast R-CNN	146	1510	174	1830	0.5 fps
VGG	RPN + Fast R-CNN	141	**10**	47	**198**	**5 fps**
ZF	RPN + Fast R-CNN	31	**3**	25	**59**	**17 fps**

YOLO

Wei-Ta Chu

Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," CVPR, 2016.

Introduction

- We frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation.
- The base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second.

Introduction

- A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes.

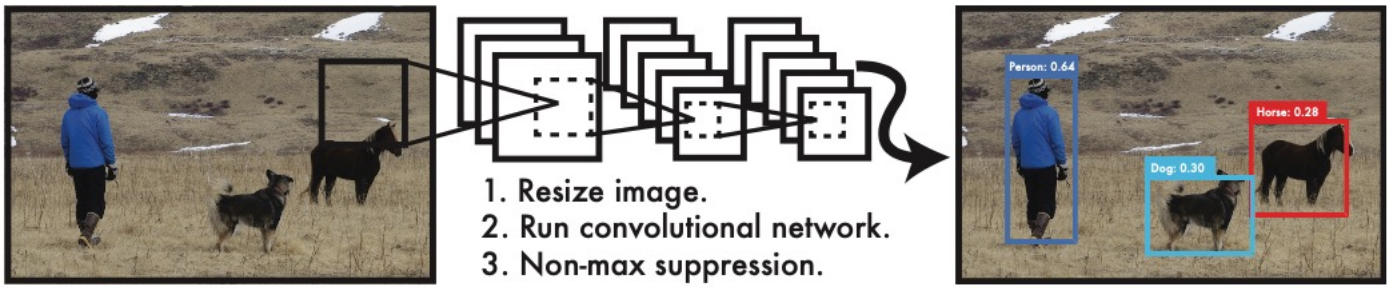


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

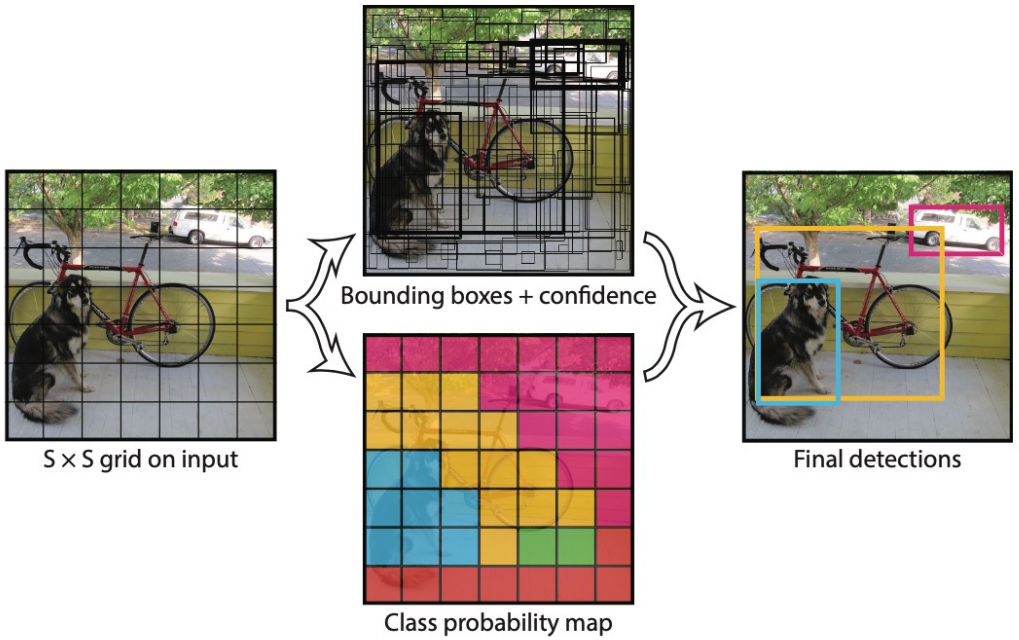
Introduction

- YOLO is extremely fast.
- YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance.
  - YOLO makes less than half the number of background errors compared to Fast R-CNN.
- YOLO learns generalizable representations of objects.

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

Unified Detection

- Divide the input image into an S × S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.
- Each grid cell predicts *B* bounding boxes and confidence scores for those boxes.



Unified Detection

- Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell.
- The width and height are predicted relative to the whole image. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box.
- Each grid cell also predicts C conditional class probabilities, $\Pr(\text{Class}_i \,|\text{Object})$.

Network Design

- 24 convolutional layers followed by 2 fully connected layers

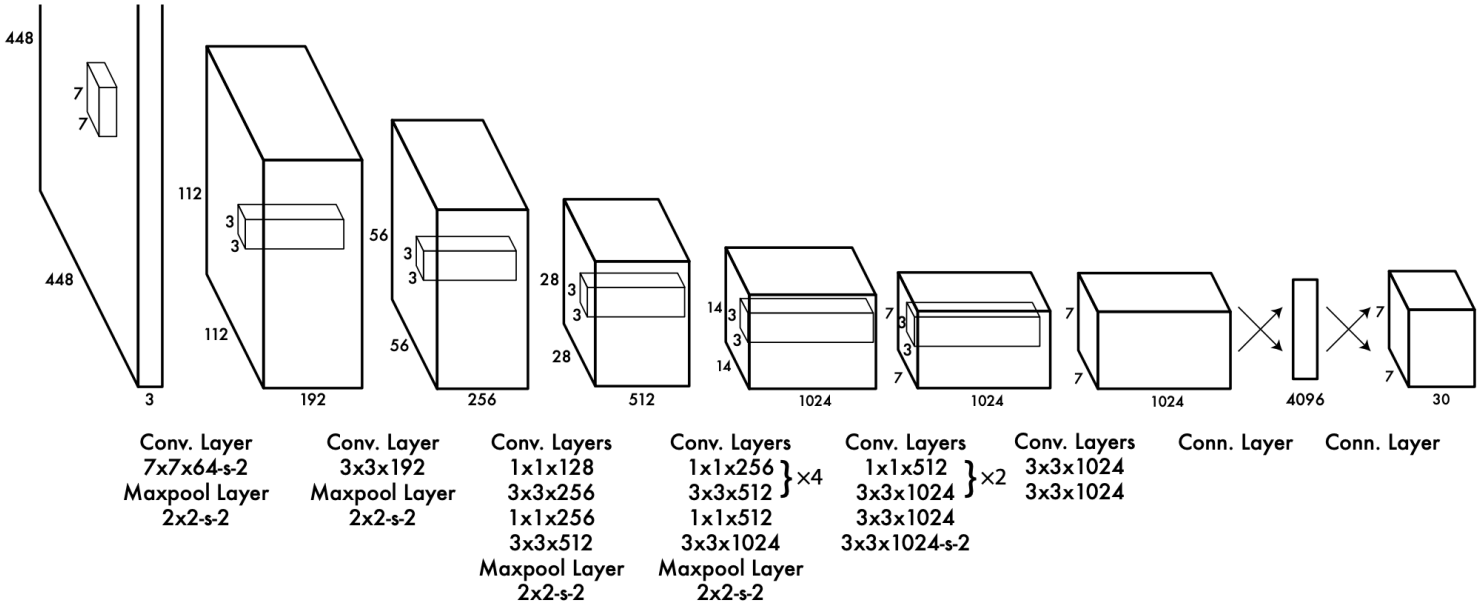


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224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

Training

- Pretrain convolutional layers on the ImageNet 1000-class competition dataset.
- Our final layer predicts both class probabilities and bounding box coordinates.
- We optimize for sum-squared error in the output of our model. We increase the loss from bounding box coordinate predictions and decrease the loss from confidence predictions for boxes that don't contain objects.

Limitations

- YOLO imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class.
- It struggles to generalize to objects in new or unusual aspect ratios or configurations.
- Our loss function treats errors the same in small bounding boxes versus large bounding boxes.

Experiments

- Few systems really can run in real-time.
- Fast R-CNN speeds up the classification stage of R-CNN but it still relies on selective search which can take around 2 seconds per image.
- The VGG-16 version of Faster R-CNN is 10 mAP higher but is also 6 times slower than YOLO.

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

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

Experiments

- VOC 2007 error analysis
- Correct: correct class and IOU > .5
- Localization: correct class, .1 < IOU
- Similar: class is similar, IOU > .1
- Other: class is wrong, IOU > .1
- Background: IOU < .1 for any object

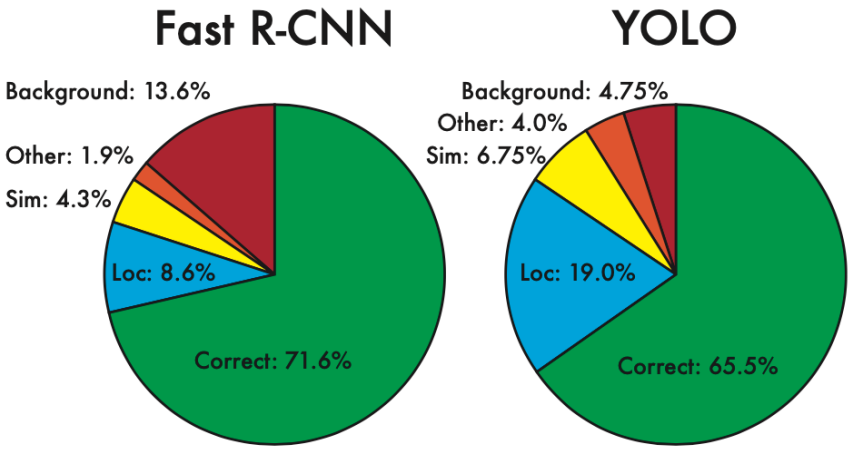


Figure 4: Error Analysis: Fast R-CNN vs. YOLO These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

- YOLO struggles to localize objects correctly.
- Fast R-CNN is almost 3x more likely to predict background detections than YOLO.

Experiments

- YOLO makes far fewer background mistakes than Fast R-CNN. By using YOLO to eliminate background detections from Fast R-CNN we get a significant boost in performance.

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

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

Experiments

- VOC 2012

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.

YOLOv2

Wei-Ta Chu

Joseph Redmon and Ali Farhadi, "YOLO9000: Better, Faster, Stronger," CVPR, 2017.

Introduction

- YOLOv2 model can run at varying sizes, offering an easy tradeoff between speed and accuracy. At 67 fps, YOLOv2 gets 76.8 mAP on VOC 2007. At 40 fps, YOLOv2 gets 78.6 mAP, outperforming state-of-the-art methods.
- Can detect over 9000 object categories

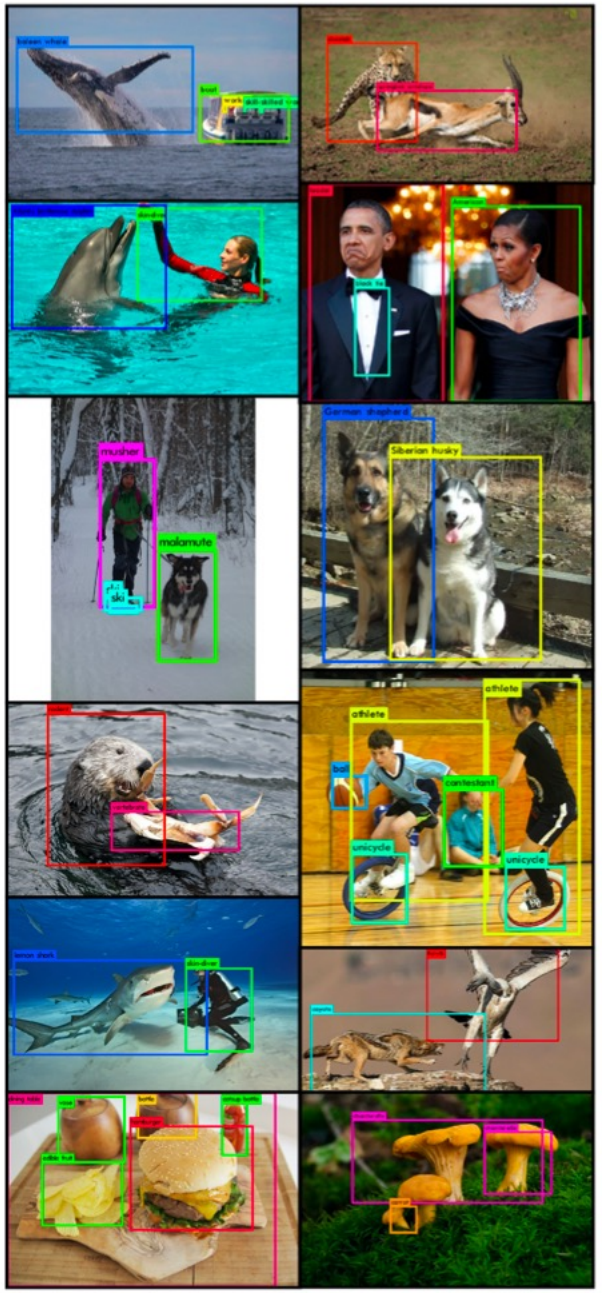


Figure 1: YOLO9000. YOLO9000 can detect a wide variety of object classes in real-time.

Better

- YOLO makes a significant number of localization errors.
- **Batch normalization**
  - Batch normalization leads to significant improvements in convergence while eliminating the need for other forms of regularization.
  - With batch normalization we can remove dropout from the model without overfitting.
- **High resolution classifier**
  - The original YOLO trains the classifier network at 224 × 224 and increases the resolution to 448 for detection.
  - For YOLOv2 we first fine tune the classification network at the full 448 × 448 resolution for 10 epochs on ImageNet. We then fine tune the resulting network on detection.

Better

- **Convolutional With anchor boxes**
  - We remove the fully connected layers from YOLO and use anchor boxes to predict bounding boxes.
  - In Faster R-CNN, RPN predicts these offsets at every location in a feature map. Predicting offsets instead of coordinates simplifies the problem and makes it easier for the network to learn.
  - When we move to anchor boxes we also decouple the class prediction mechanism from the spatial location and instead predict class and objectness for every anchor box.

Better

- **Dimension clusters**
  - In YOLO, the box dimensions are hand picked.
  - We run k-means clustering on the training set bounding boxes to automatically find good priors.
  - We choose k = 5 as a good tradeoff between model complexity and high recall.

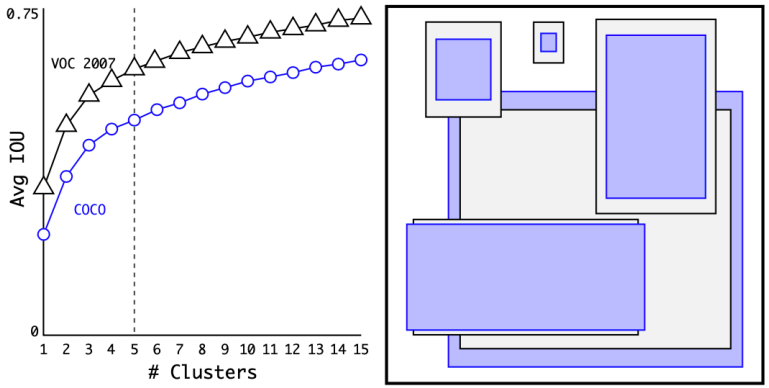


Figure 2: Clustering box dimensions on VOC and COCO. We run k-means clustering on the dimensions of bounding boxes to get good priors for our model. The left image shows the average IOU we get with various choices for k. $k = 5$ gives a good tradeoff for recall vs. complexity of the model. The right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

Better

- **Direct location prediction**
  - When using anchor boxes with YOLO we encounter model instability. Most of the instability comes from predicting the (x, y) locations for the box.
  - A prediction of $t_x = 1$ would shift the box to the right by the width of the anchor box, a prediction of $t_x = -1$ would shift it to the left by the same amount. Anchor box can end up at any point in the image, regardless of what location predicted the box.
  - We follow the approach of YOLO and predict location coordinates relative to the location of the grid cell. This bounds the ground truth to fall between 0 and 1.

Better

- **Fine-Grained Features**
  - Design a passthrough layer that concatenates the higher resolution features with the low resolution features by stacking adjacent features into different channels instead of spatial locations.
- **Multi-Scale Training**
  - Instead of fixing the input image size we change the network every few iterations. Every 10 batches our network randomly chooses a new image dimension size.

Better

	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%.

Better

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

Table 3: Detection frameworks on PASCAL VOC 2007. YOLOv2 is faster and more accurate than prior detection methods. It can also run at different resolutions for an easy tradeoff between speed and accuracy. Each YOLOv2 entry is actually the same trained model with the same weights, just evaluated at a different size. All timing information is on a Geforce GTX Titan X (original, not Pascal model).

Better

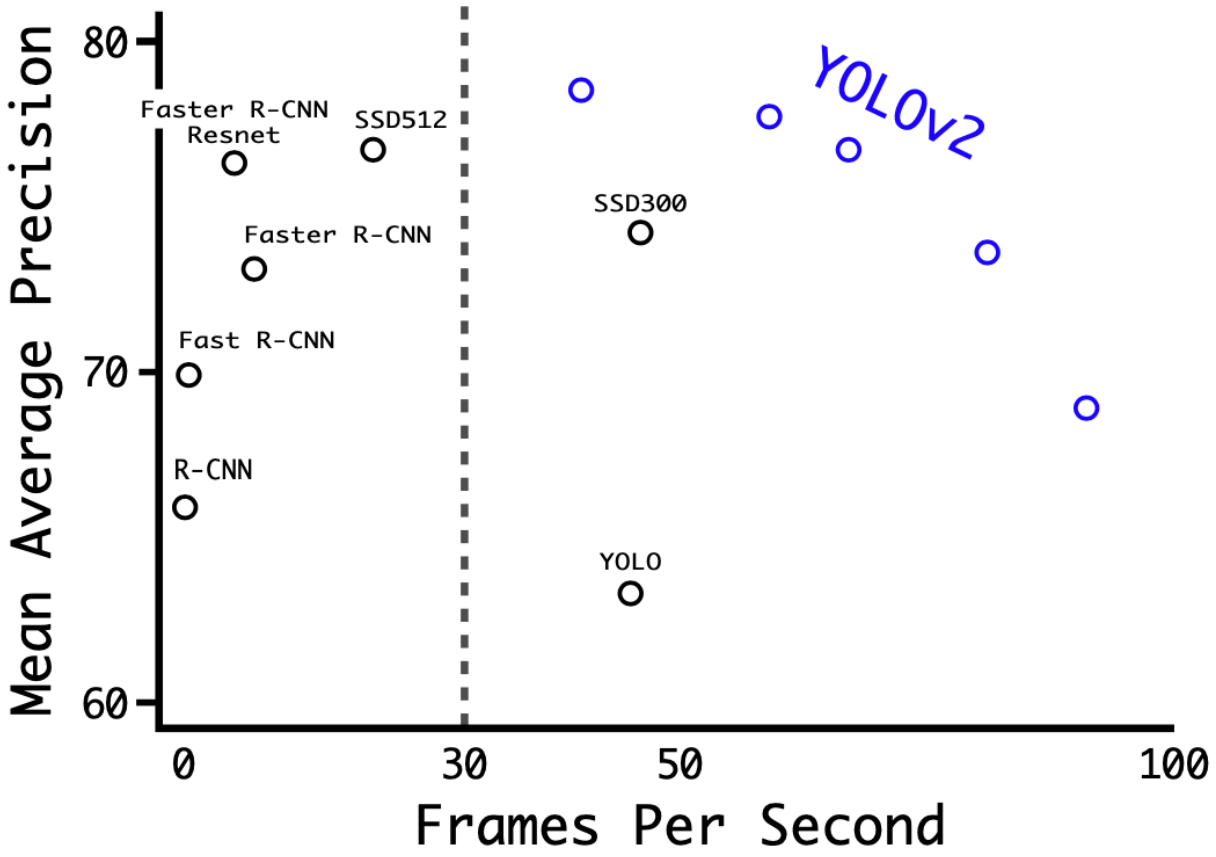


Figure 4: Accuracy and speed on VOC 2007.

Faster

- **Darknet-19**
  - Similar to the VGG models we use mostly 3 × 3 filters and double the number of channels after every pooling step.
  - Following the work on Network in Network (NIN) we use global average pooling to make predictions as well as 1 × 1 filters to compress the feature representation between 3 × 3 convolutions.

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

Stronger

- Most approaches to classification use a softmax layer across all the possible categories. Using a softmax assumes the classes are mutually exclusive.
- Hierarchical classification
  - Construct a WordTree, a hierarchical model of visual concepts
  - If we want to compute the absolute probability for a particular node we simply follow the path through the tree to the root node and multiply to conditional probabilities.

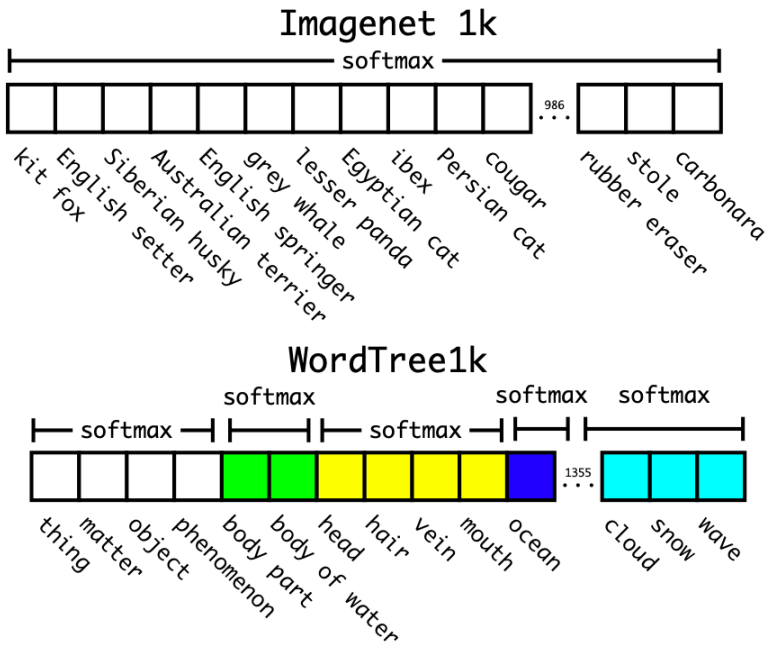


Figure 5: Prediction on ImageNet vs WordTree. Most ImageNet models use one large softmax to predict a probability distribution. Using WordTree we perform multiple softmax operations over co-hyponyms.

Stronger

- Dataset combination with WordTree
- We create a combined dataset using the COCO detection dataset and the top 9000 classes from the full ImageNet release

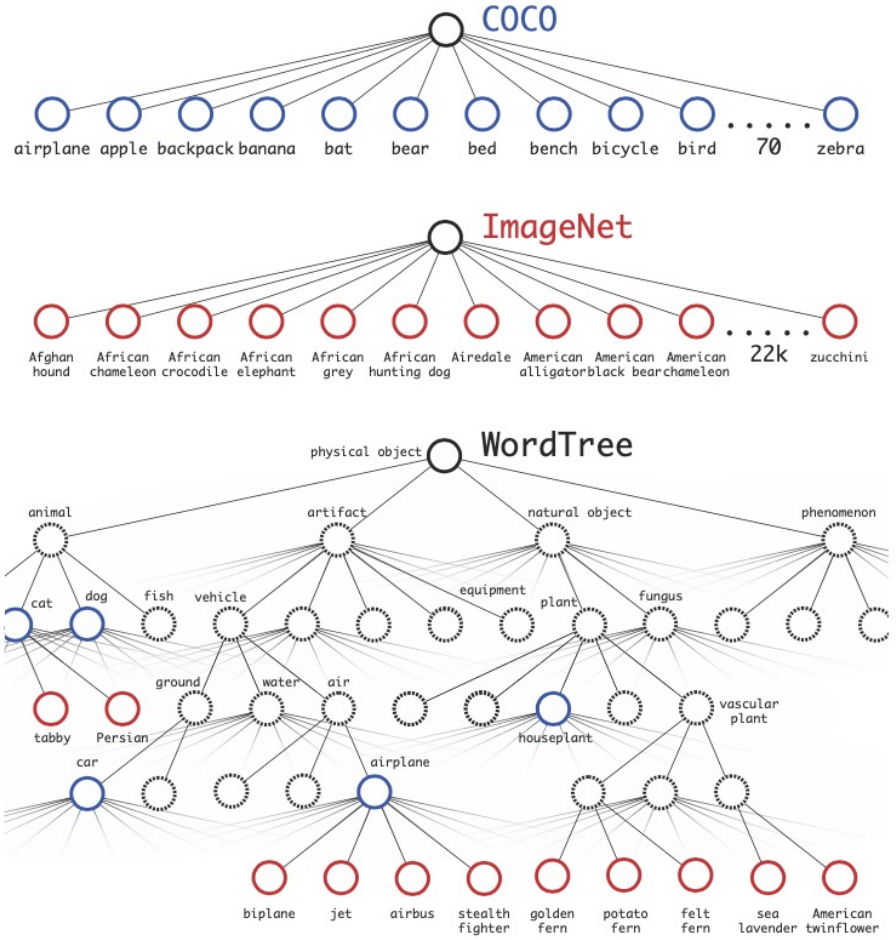


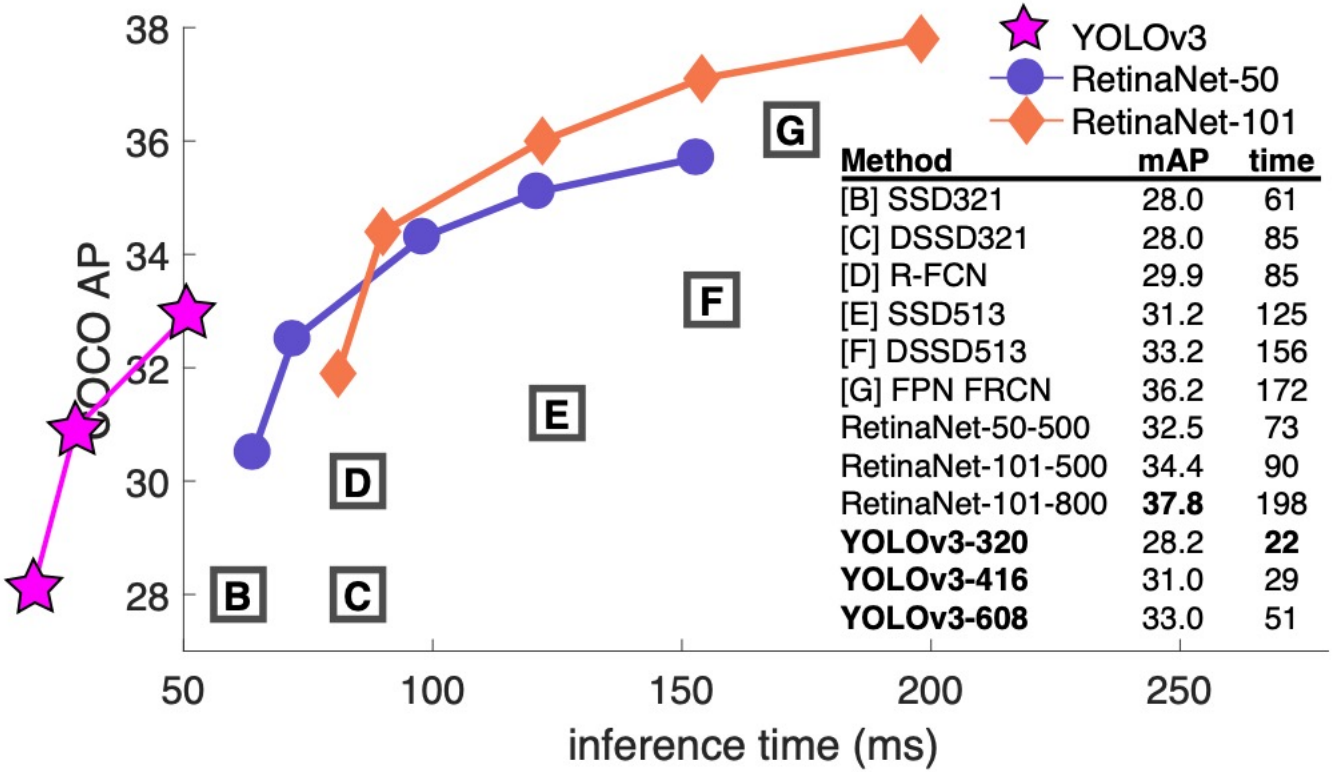
Figure 6: Combining datasets using WordTree hierarchy. Using the WordNet concept graph we build a hierarchical tree of visual concepts. Then we can merge datasets together by mapping the classes in the dataset to synsets in the tree. This is a simplified view of WordTree for illustration purposes.

YOLOv3

Wei-Ta Chu

Joseph Redmon and Ali Farhadi, "YOLOv3: An Incremental Improvement," https://arxiv.org/abs/1804.02767, 2018.

Introduction



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.

Bounding Box Prediction

- YOLOv2 predicts 4 coordinates for each bounding box, t_x, t_y, t_w, t_h. If the cell is offset from the top left corner of the image by (c_x , c_y) and the bounding box prior has width and height p_w, p_h, then the predictions correspond to:

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

  - During training we use sum of squared error loss.

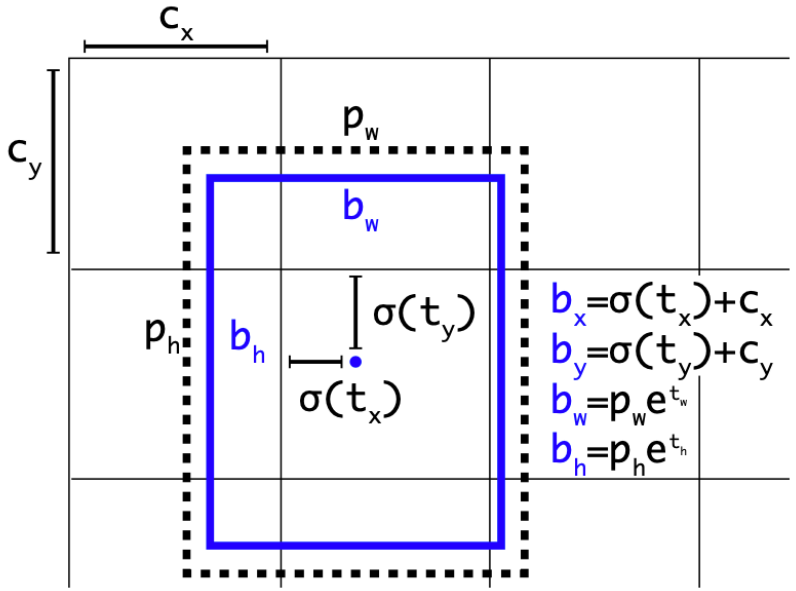


Figure 2. **Bounding boxes with dimension priors and location prediction.** We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function. This figure blatantly self-plagiarized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

Bounding Box Prediction

- YOLOv3 predicts an objectness score for each bounding box using logistic regression.
- This should be 1 if the bounding box prior overlaps a ground truth object by more than any other bounding box prior. If the bounding box prior is not the best but does overlap a ground truth object by more than some threshold we ignore the prediction, following Faster R-CNN.

Other Techniques

- **Class prediction**
  - Each box predicts the classes the bounding box may contain using multilabel classification.
  - We do not use a softmax but instead simply use independent logistic classifiers. During training we use binary cross-entropy loss for the class predictions.
- **Predictions across scales**
  - YOLOv3 predicts boxes at 3 different scales.
- **Feature extractor**
  - Our new network is a hybrid approach between the network used in YOLOv2, Darknet-19, and that newfangled residual network stuff. This new network is much more powerful than Darknet-19 but still more efficient than ResNet-101 or ResNet-152.

Experiments

	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

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

What This All Means

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

- I have a lot of hope that most of the people using computer vision are just doing happy, good stuff with it.
- But computer vision is already being put to questionable use …

YOLOv4

Wei-Ta Chu

Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," https://arxiv.org/abs/2004.10934, 2020.

Introduction

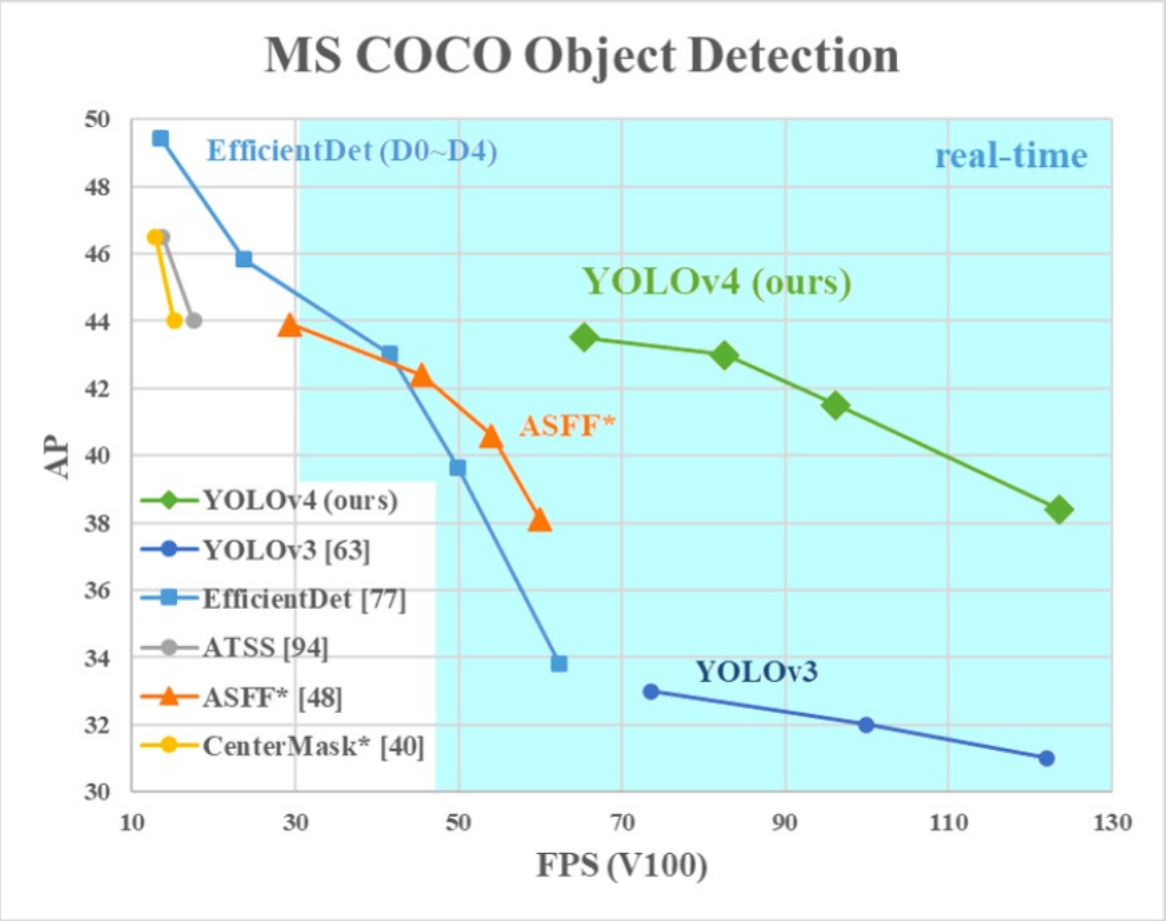


Figure 1: Comparison of the proposed YOLOv4 and other state-of-the-art object detectors. YOLOv4 runs twice faster than EfficientDet with comparable performance. Improves YOLOv3's AP and FPS by 10% and 12%, respectively.

YOLOv4 Consists of …

- A modern detector is usually composed of two parts, a *backbone* which is pre-trained on ImageNet and a *head* which is used to predict classes and bounding boxes of objects.
- Object detectors developed in recent years often insert some layers (called *neck*) between backbone and head, and these layers are usually used to collect feature maps from different stages.
- Backbone: CSPDarknet53 [81]
- Neck: SPP [25], PAN [49]
- Head: YOLOv3 [63]

YOLOv4 Uses …

- Bag of Freebies (BoF) for backbone: CutMix and Mosaic data augmentation, DropBlock regularization, Class label smoothing.
  - Researchers train the object detector offline, and develop **better training methods which can make the object detector receive better accuracy without increasing the inference cost.**
  - We call these methods that only change the training strategy or only increase the training cost as “bag of freebies.”

YOLOv4 Uses …

- Bag of Specials (BoS) for backbone: Mish activation, Cross-stage partial connections (CSP), Multi-input weighted residual connections (MiWRC)
  - For those **plugin modules and post-processing methods that only increase the inference cost by a small amount** but can significantly improve the accuracy of object detection, we call them "bag of specials".

YOLOv4 Uses …

- Bag of Freebies (BoF) for detector: CIoU-loss, CmBN, DropBlock regularization, Mosaic data augmentation, Self-Adversarial Training, Eliminate grid sensitivity, Using multiple anchors for a single ground truth, Cosine annealing scheduler [52], Optimal hyper-parameters, Random training shapes
- Bag of Specials (BoS) for detector: Mish activation, SPP-block, SAM-block, PAN path-aggregation block, DIoU-NMS

Influence of different features on Classifier training

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%**

Influence of different features on Detector training

Table 4: Ablation Studies of Bag-of-Freebies. (CSPResNeXt50-PANet-SPP, 512x512).

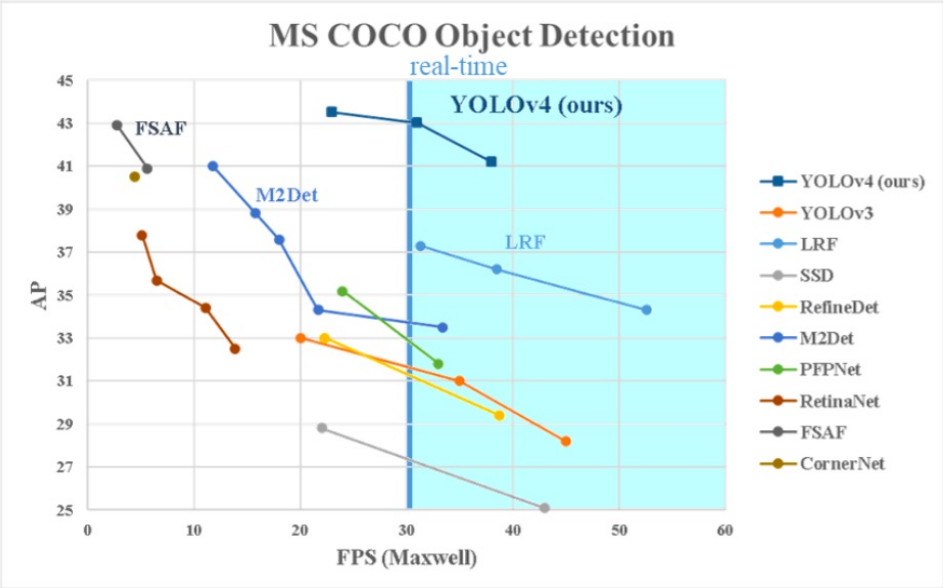
S	M	IT	GA	LS	CBN	CA	DM	OA	loss	AP	AP_{50}	AP_{75}
									MSE	38.0%	60.0%	40.8%
✓									MSE	37.7%	59.9%	40.5%
	✓								MSE	**39.1%**	**61.8%**	**42.0%**
		✓							MSE	36.9%	59.7%	39.4%
			✓						MSE	**38.9%**	**61.7%**	**41.9%**
				✓					MSE	33.0%	55.4%	35.4%
					✓				MSE	**38.4%**	**60.7%**	**41.3%**
						✓			MSE	**38.7%**	**60.7%**	**41.9%**
							✓		MSE	35.3%	57.2%	38.0%
✓									GIoU	**39.4%**	59.4%	**42.5%**
✓									DIoU	**39.1%**	58.8%	**42.1%**
✓									CIoU	**39.6%**	59.2%	**42.6%**
✓	✓	✓	✓						CIoU	**41.5%**	**64.0%**	**44.8%**
	✓		✓					✓	CIoU	36.1%	56.5%	38.4%
✓	✓	✓	✓					✓	MSE	**40.3%**	**64.0%**	**43.1%**
✓	✓	✓	✓					✓	GIoU	**42.4%**	**64.4%**	**45.9%**
✓	✓	✓	✓					✓	CIoU	**42.4%**	**64.4%**	**45.9%**

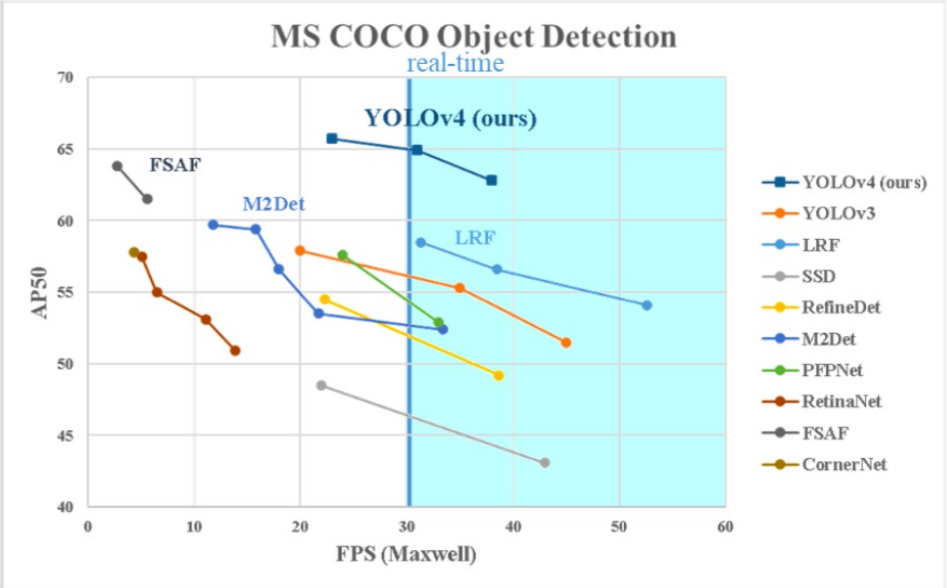
Influence of different backbones and pre-trained weightings on Detector training

Table 6: Using different classifier pre-trained weightings for detector training (all other training parameters are similar in all models) .

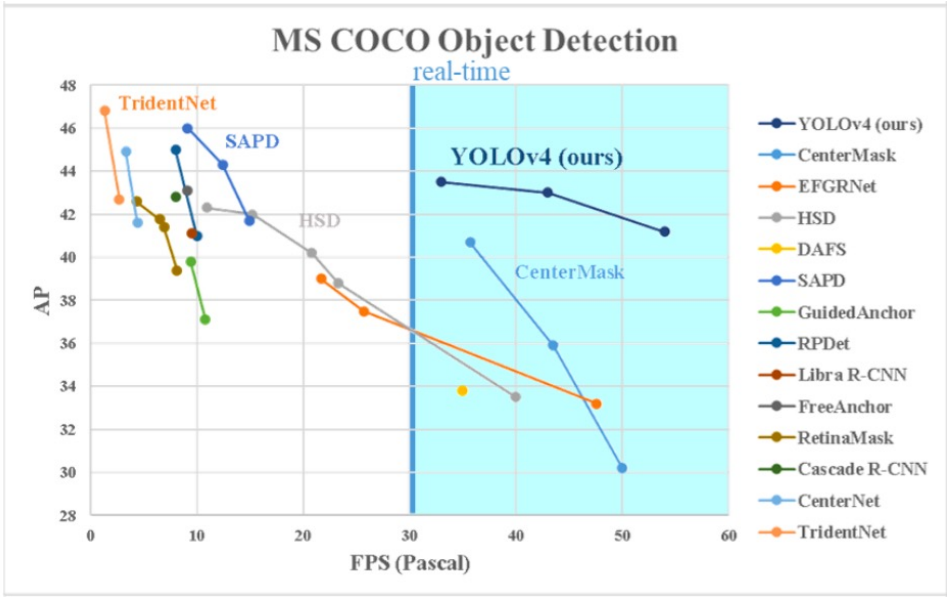
Model (with optimal setting)	Size	AP	AP_{50}	AP_{75}
CSPResNeXt50-PANet-SPP	512x512	42.4	64.4	45.9
CSPResNeXt50-PANet-SPP (BoF-backbone)	512x512	42.3	64.3	45.7
CSPResNeXt50-PANet-SPP (BoF-backbone + Mish)	512x512	42.3	64.2	45.8
CSPDarknet53-PANet-SPP (BoF-backbone)	512x512	42.4	64.5	46.0
CSPDarknet53-PANet-SPP (BoF-backbone + Mish)	512x512	43.0	64.9	46.5

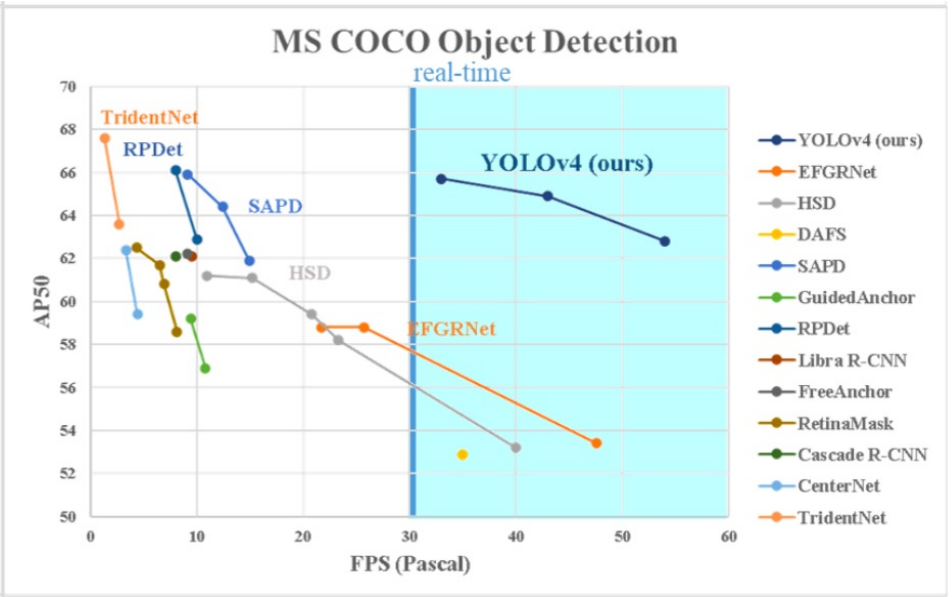
Performance Comparison



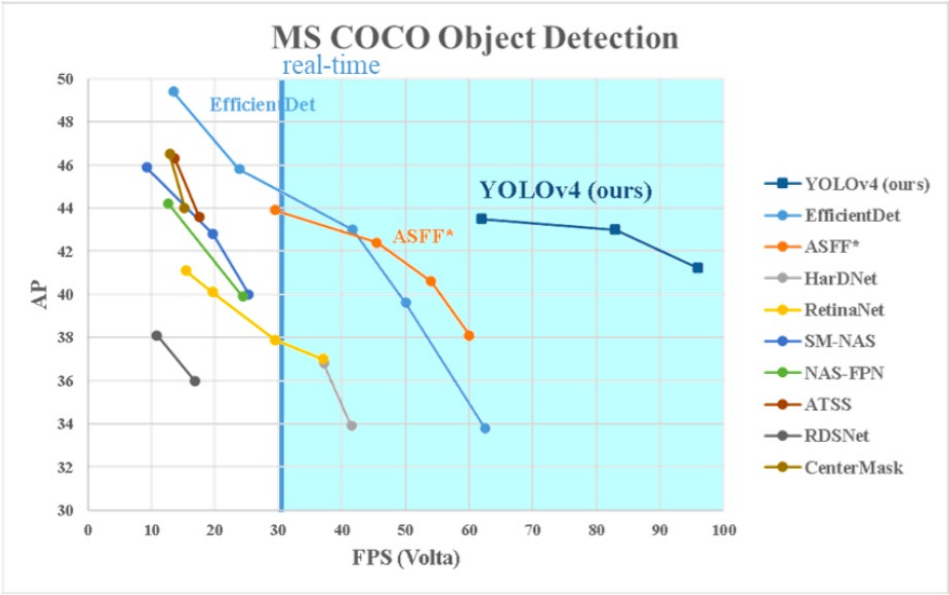


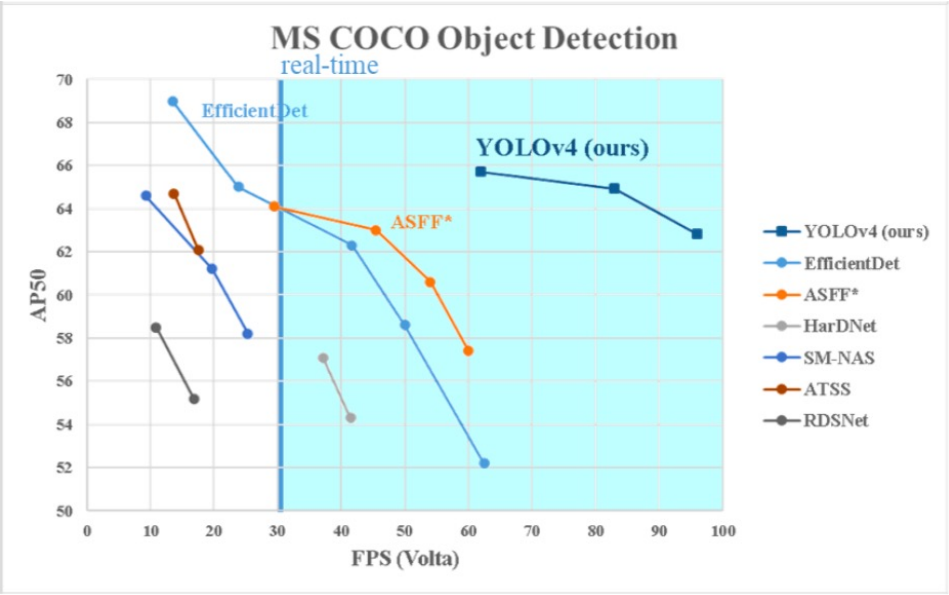
Performance Comparison





Performance Comparison



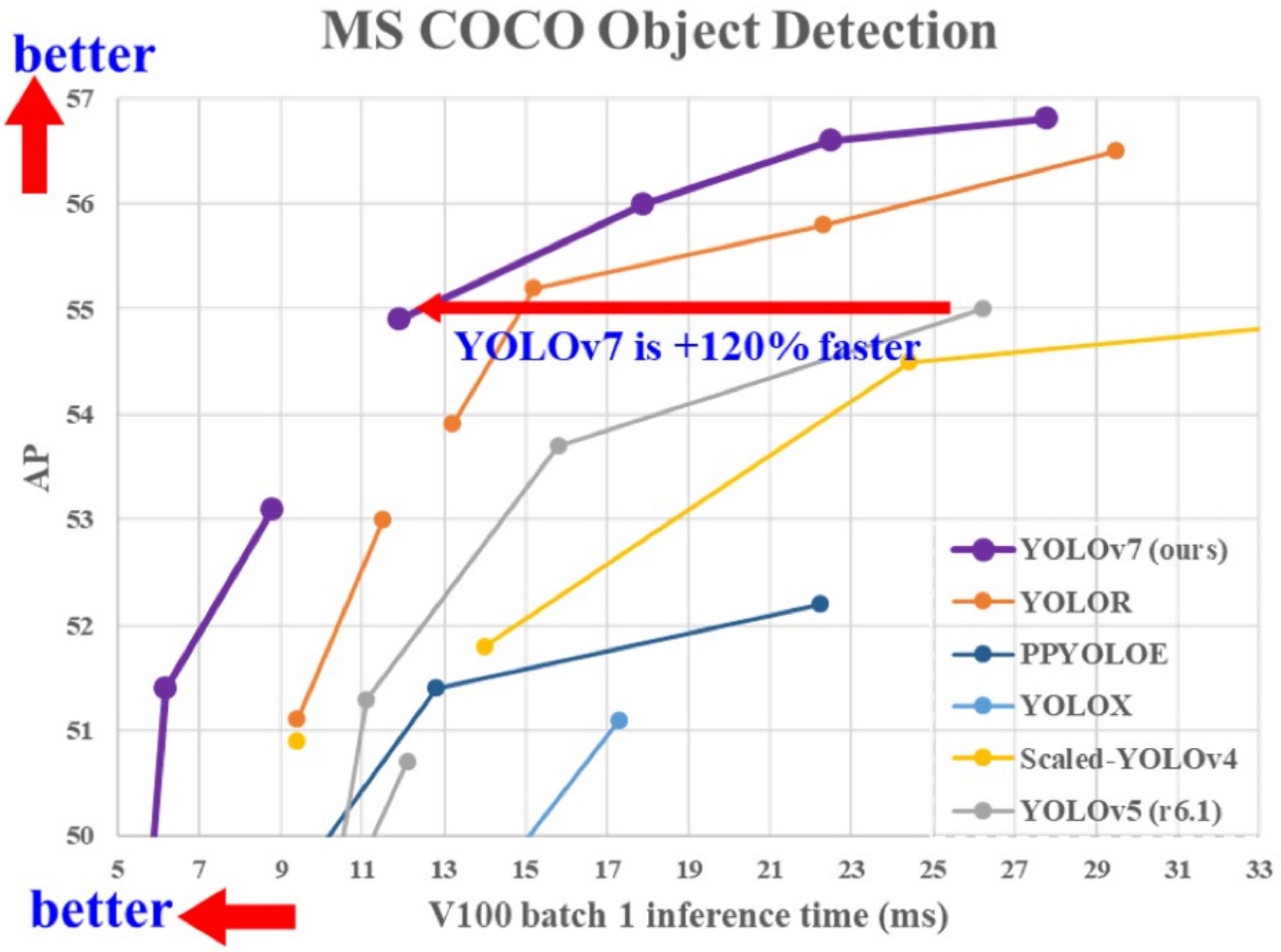


YOLOv7

Wei-Ta Chu

Chien-Yao Wang, Alexey Bochkovskiy, Hong-Yuan Mark Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," https://arxiv.org/abs/2207.02696, 2022.

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)

DETR

Wei-Ta Chu

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko, “End-to-End Object Detection with Transformers,” ECCV 2020.

Summary of DEtection TRansformer

- View object detection as a direct set prediction problem
- Removing the need of non-maximum suppression and anchor generation
- A set-based global loss that forces unique predictions via bi-partite matching, and a transformer encoder-decoder architecture.

- DETR demonstrates accuracy and run-time performance on par with Faster R-CNN on the COCO object detection dataset.
- Can be easily generalized to panoptic segmentation

Summary of DEtection TRansformer

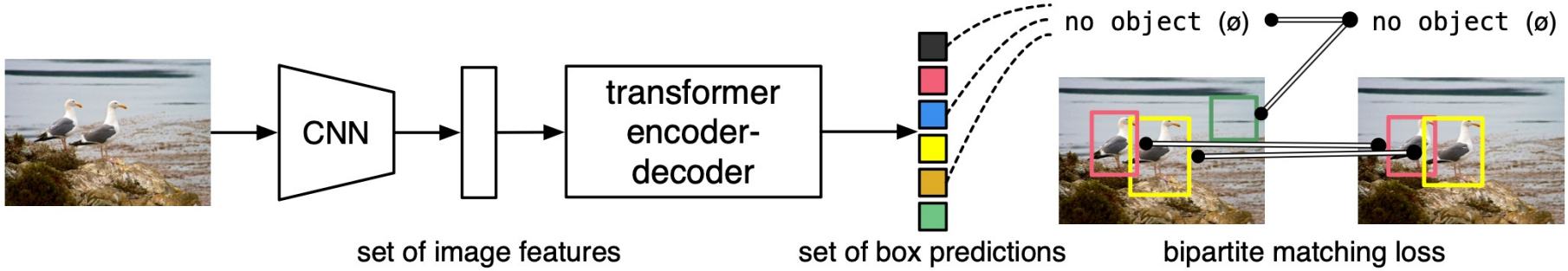


Fig. 1: DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes. Prediction with no match should yield a "`no object`" ($\varnothing$) class prediction.

DEtection TRansformer

- DETR infers a fixed-size set of *N* predictions, where *N* is significantly larger than the typical number of objects
- Use Hungarian Algorithm to find a bipartite matching with the lowest matching cost:

$$\hat{\sigma} = \arg\min_{\sigma \in \mathfrak{S}_N} \sum_{i}^{N} \mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)})$$

we consider *y* also as a set of size *N* padded with ∅
we search for a permutation of *N* elements σ with the lowest cost

DEtection TRansformer

- Matching cost accounts for class probability and the predicted box:

$$\hat{\sigma} = \arg\min_{\sigma \in \mathfrak{S}_N} \sum_{i}^{N} \mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)})$$

$$\mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)}) = -\mathbb{1}_{\{c_i \neq \varnothing\}} \hat{p}_{\sigma(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\sigma(i)})$$

$y_i = (c_i, b_i)$ where c_i is the target class label (which may be $\varnothing$) and $b_i \in [0,1]^4$ is a vector that defines ground truth box

DEtection TRansformer

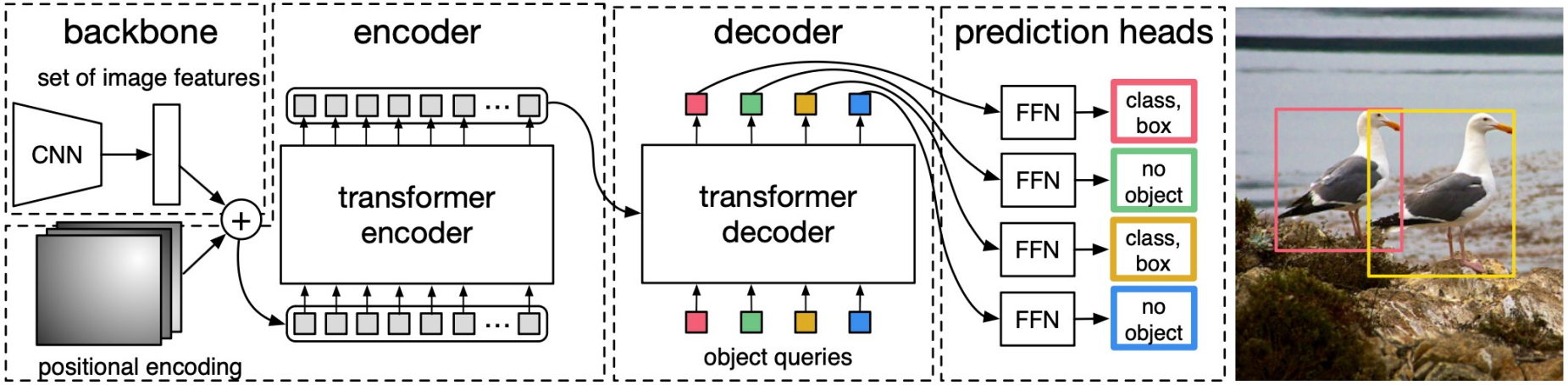
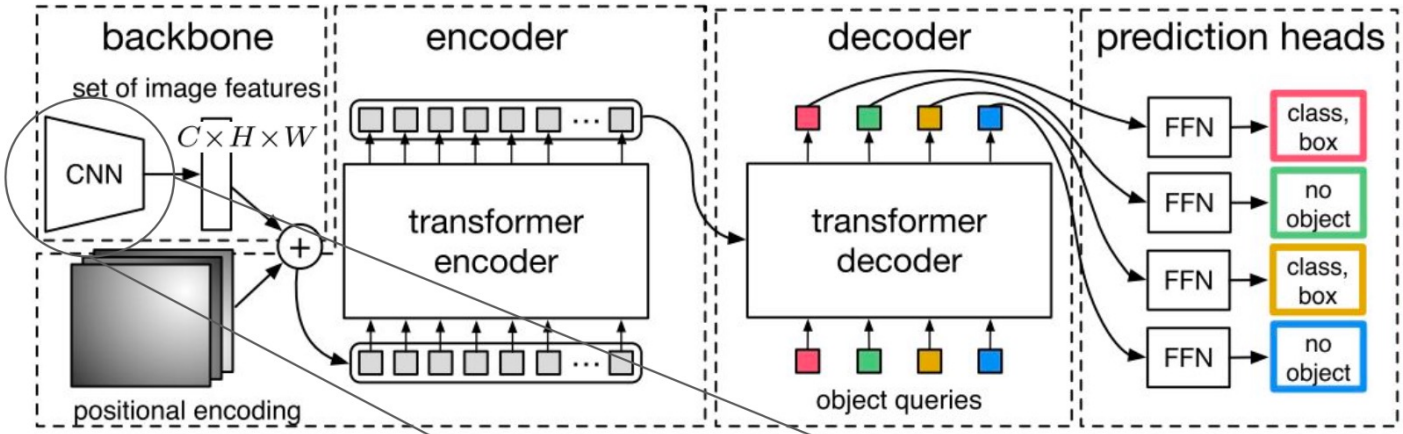
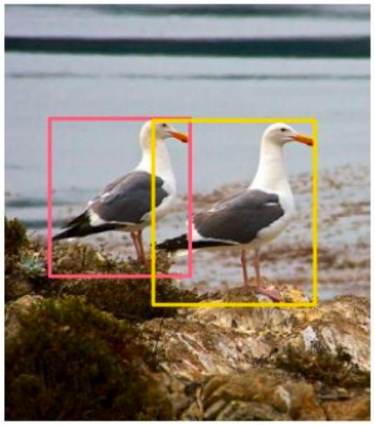


Fig. 2: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small fixed number of learned positional embeddings, which we call *object queries*, and additionally attends to the encoder output. We pass each output embedding of the decoder to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a "`no object`" class.

DEtection TRansformer

https://reurl.cc/va6kZk





- CNN Backbone:

$$f \in \mathbb{R}^{C \times H \times W}$$

Activation Map $C \times H \times W$

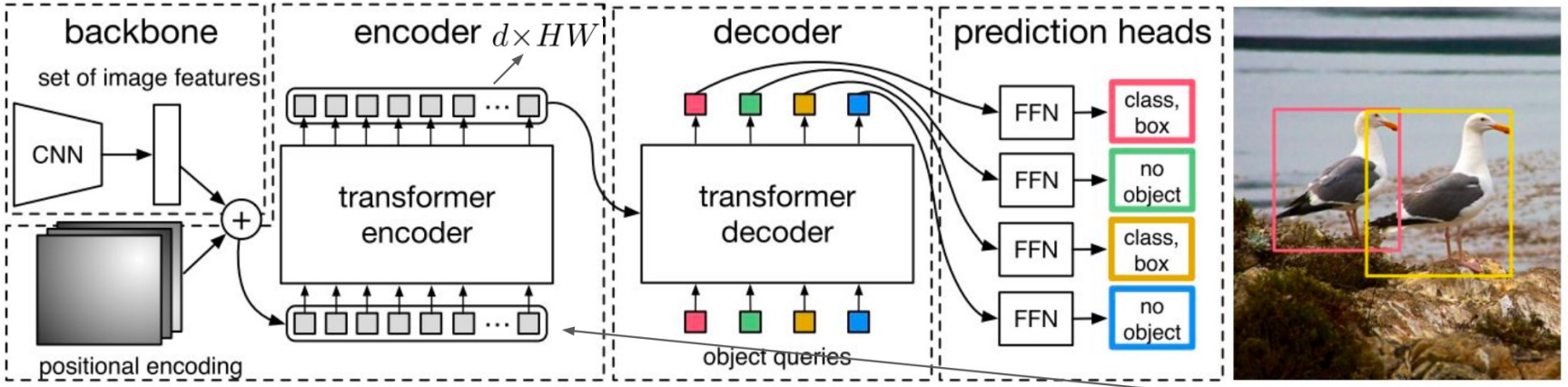
$$x_{\text{img}} \in \mathbb{R}^{3 \times H_0 \times W_0}$$

Typically:

$$C = 2048 \quad H, W = \frac{H_0}{32}, \frac{W_0}{32}$$

DEtection TRansformer

https://reurl.cc/va6kZk



- Transformer Encoder:
  - Reduce channel dimension (1x1 Convolution)
  - Flatten features into a sequential feature map
  - Add positional encodings to input of each attention layer
  - A multi-head self-attention module and a feed forward network

$$C \times H \times W \longrightarrow z_0 \in \mathbb{R}^{d \times H \times W}$$

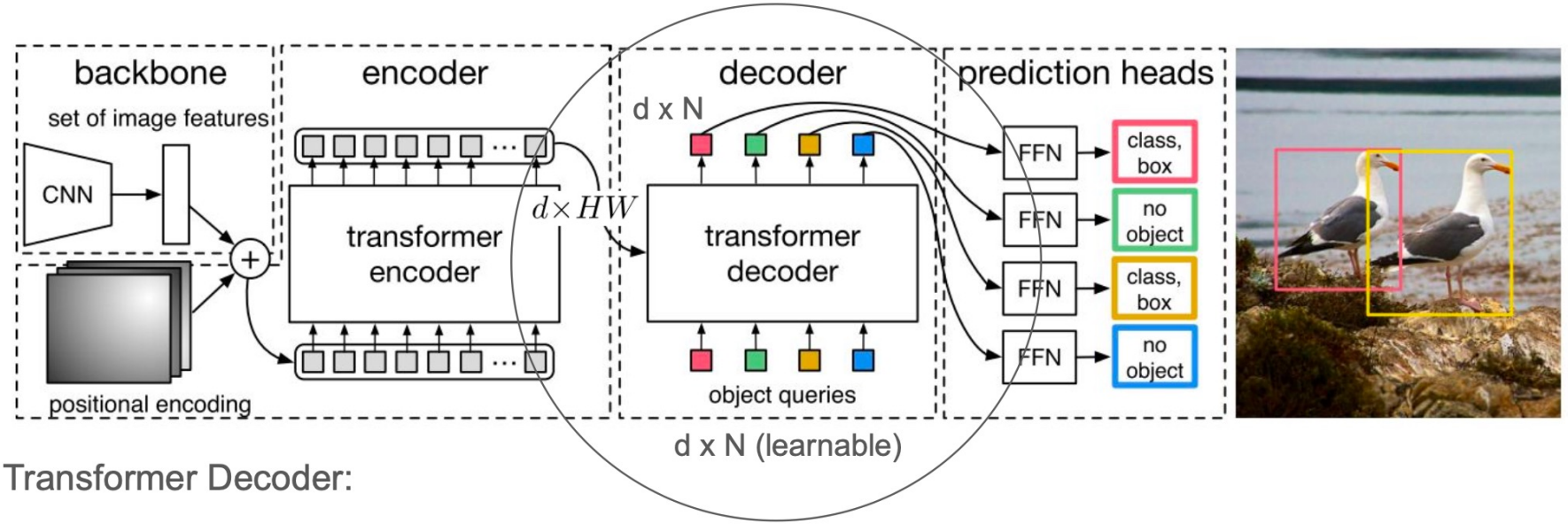
$$d \times HW$$

$$+$$

Positional Encoding

DEtection TRansformer

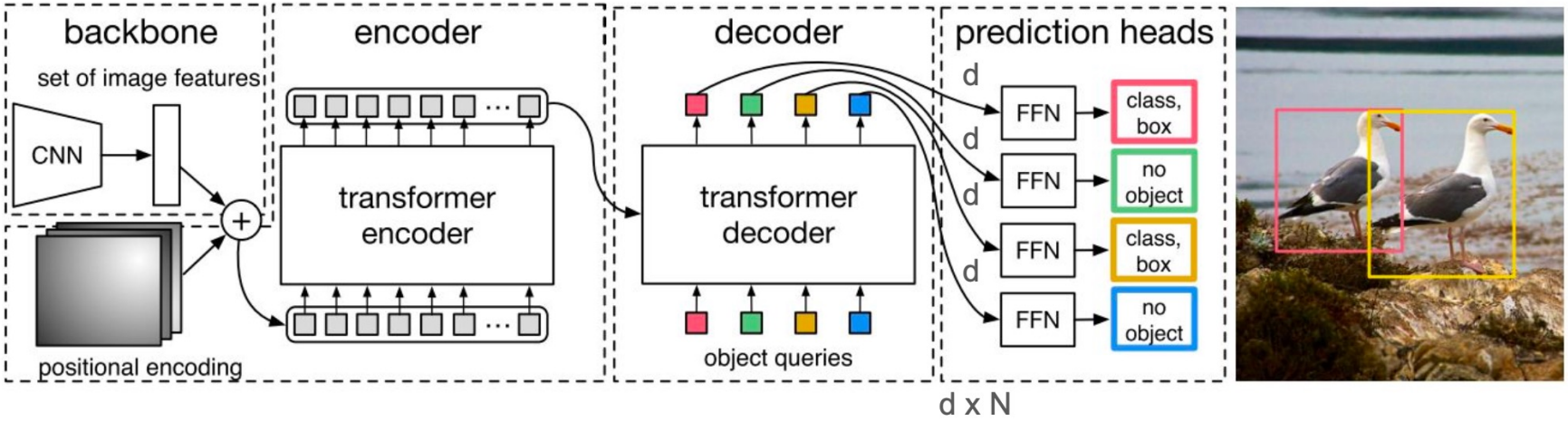
https://reurl.cc/va6kZk



- Transformer Decoder:
  - Use encoded representations, d x HW embeddings, from encoder as key and value
  - Add N learnt positional encodings (object queries) to input of each attention layer
  - Transforms N object queries into N output embeddings in parallel (non-autoregressive)
  - Trained with auxiliary decoding loss to improve training

DEtection TRansformer

https://reurl.cc/va6kZk



- Prediction Feed-forward networks (FFNs):
  - For normalized center coordinates: 3-layer perceptron with ReLU activation, hidden dim d
  - For class prediction: a linear projection layer with softmax
  - Class prediction also predicts “no object”

Experiments

- DETR can be competitive with Faster R-CNN with the same number of parameters, having lower AP_S but greatly improved AP_L.

Model	GFLOPS/FPS	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
RetinaNet	205/18	38M	38.7	58.0	41.5	23.3	42.3	50.3
Faster RCNN-DC5	320/16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	180/26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	246/20	60M	42.0	62.5	45.9	25.2	45.6	54.6
RetinaNet+	205/18	38M	41.1	60.4	43.7	25.6	44.8	53.6
Faster RCNN-DC5+	320/16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster RCNN-FPN+	180/26	42M	42.0	62.1	45.5	26.6	45.4	53.4
Faster RCNN-R101-FPN+	246/20	60M	44.0	63.9	**47.8**	**27.2**	48.1	56.0
DETR	86/28	41M	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5	187/12	41M	43.3	63.1	45.9	22.5	47.3	61.1
DETR-R101	152/20	60M	43.5	63.8	46.4	21.9	48.0	61.8
DETR-DC5-R101	253/10	60M	**44.9**	**64.7**	47.7	23.7	**49.5**	**62.3**

Experiments

- More encoder layers improve AP overall
  - Without encoder layers, AP drops by 3.9 with a significant drop of 6.0 AP in large objects

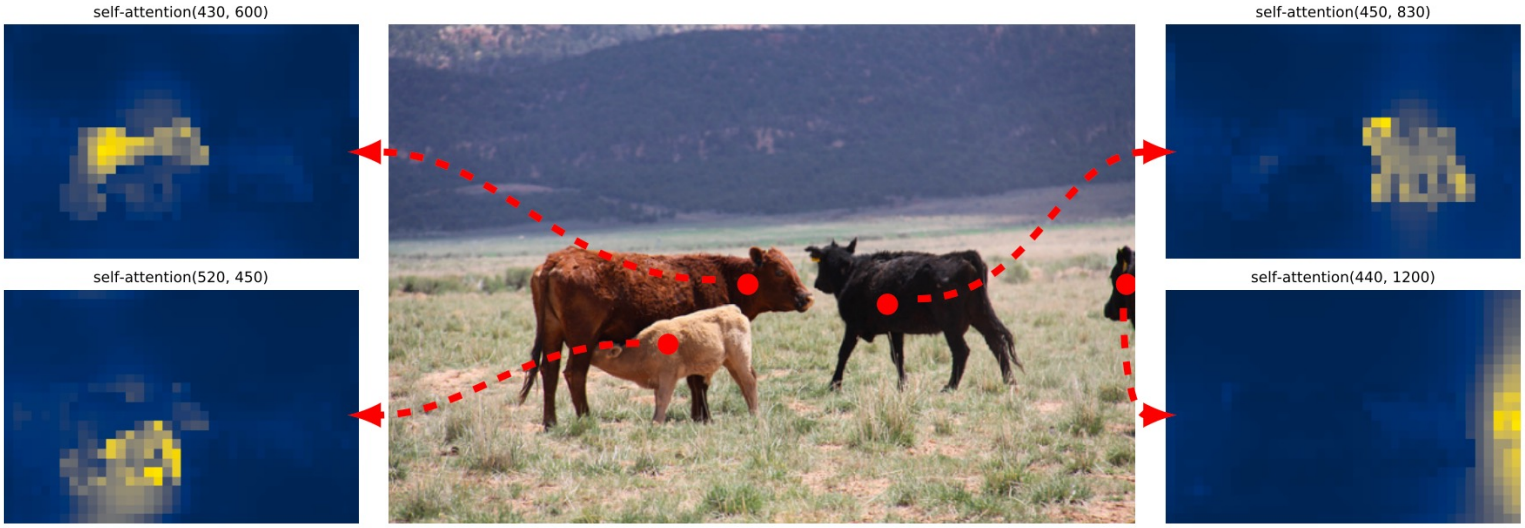


Fig. 3: Encoder self-attention for a set of reference points. The encoder is able to separate individual instances. Prediction made with baseline DETR on a validation image.

Experiments

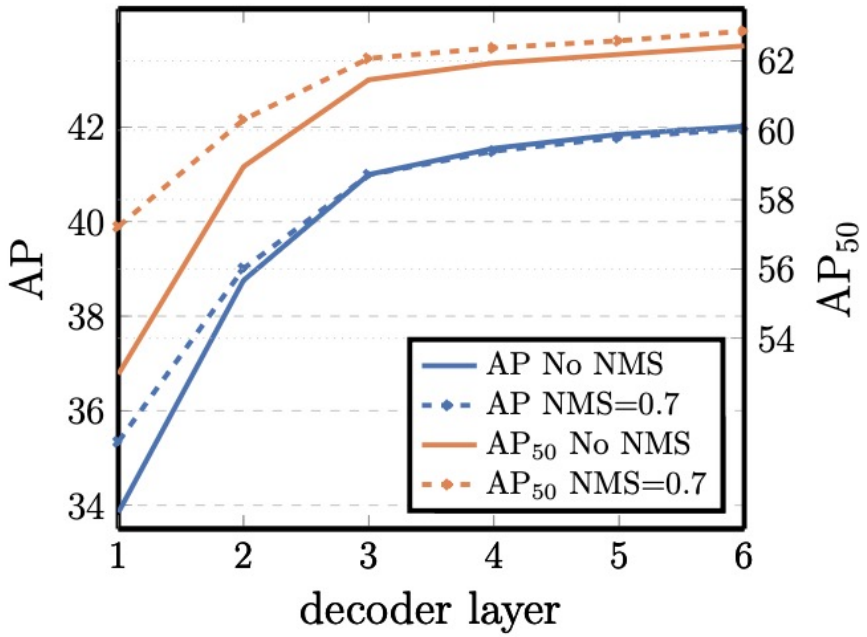


Fig. 4: AP and AP_{50} performance after each decoder layer in a long schedule baseline model. DETR does not need NMS by design, which is validated by this figure. NMS lowers AP in the final layers, removing TP predictions, but improves it in the first layers, where DETR does not have the capability to remove double predictions.

Experiments

- Decoder attention is fairly local. It mostly attends to object extremities such as heads or legs.

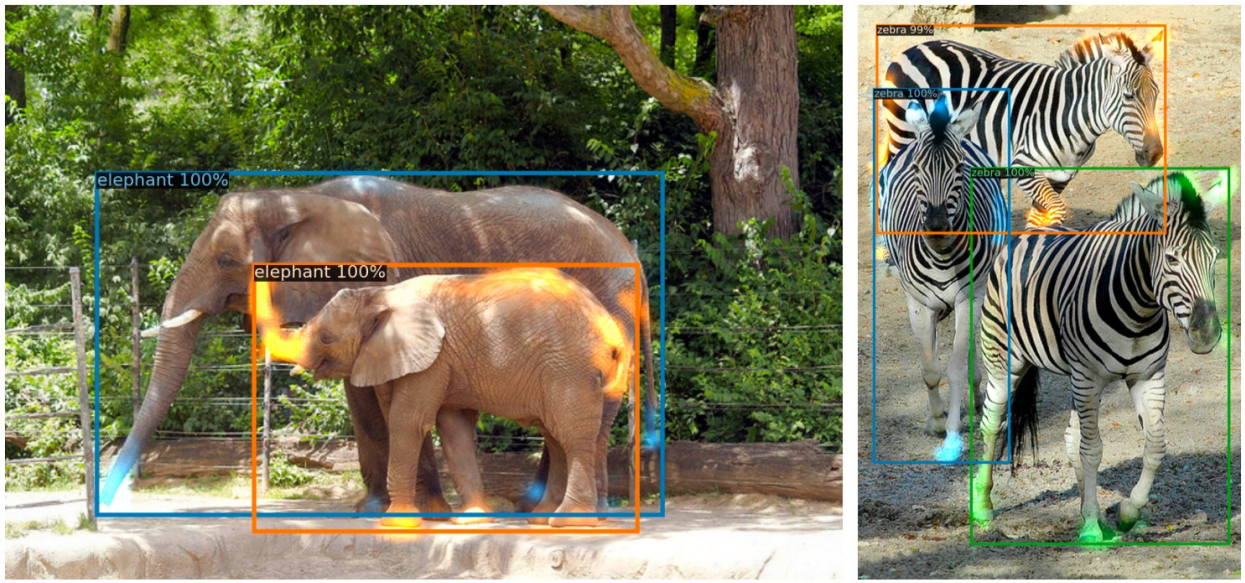
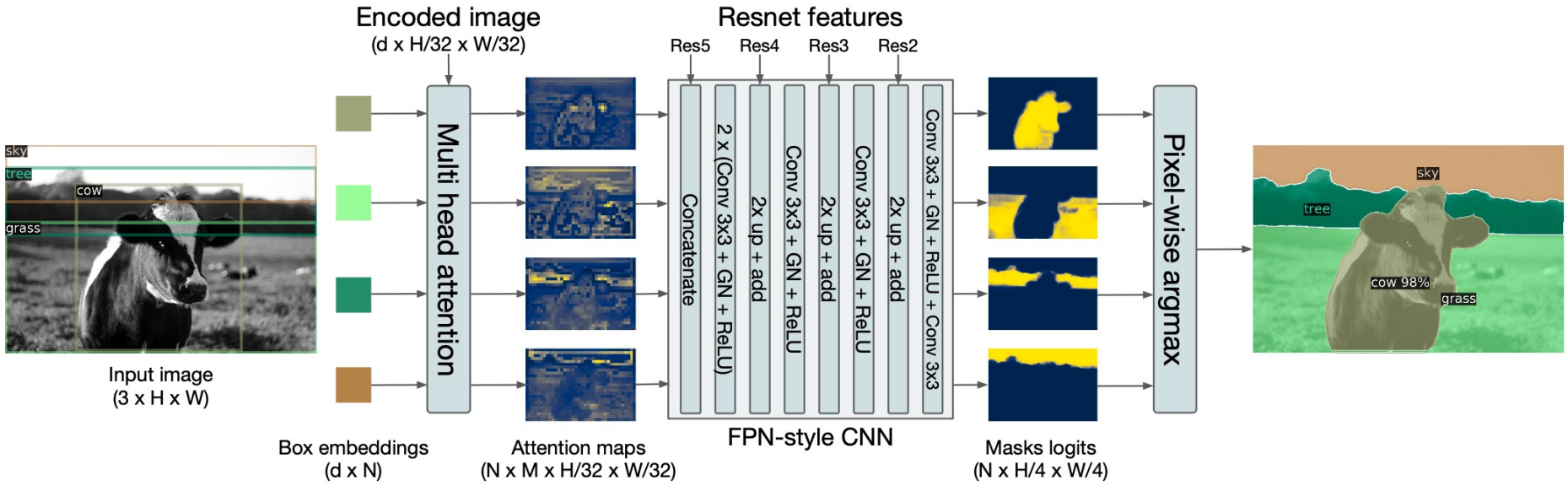


Fig. 6: Visualizing decoder attention for every predicted object (images from COCO `val` set). Predictions are made with DETR-DC5 model. Decoder typically attends to object extremities, such as legs and heads.

DETR for panoptic segmentation

- DETR can be naturally extended by adding a mask head on top of the decoder outputs.



DETR for panoptic segmentation

- DETR can be naturally extended by adding a mask head on top of the decoder outputs.

Fig. 8: Qualitative results for panoptic segmentation generated by DETR-R101. DETR produces aligned mask predictions in a unified manner for things and stuff.