Object Detection

Milan Straka

March 20, 2023

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Beyond Image Classification

Beyond Image Classification

- Object detection (including location)

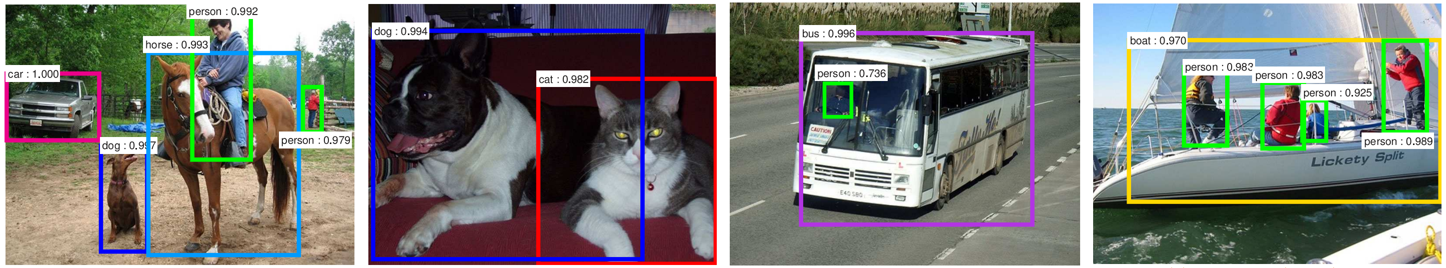


Figure 3 of "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", https://arxiv.org/abs/1506.01497

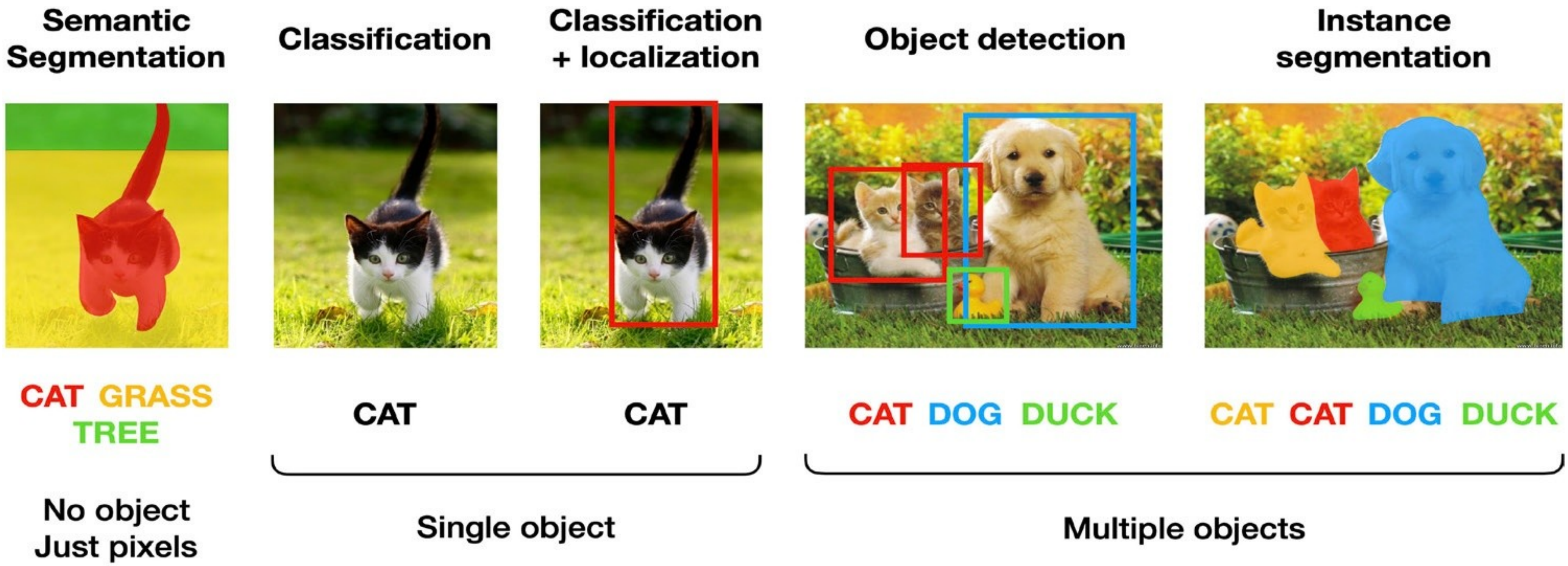
- Image segmentation

Figure 2 of "Mask R-CNN", https://arxiv.org/abs/1703.06870

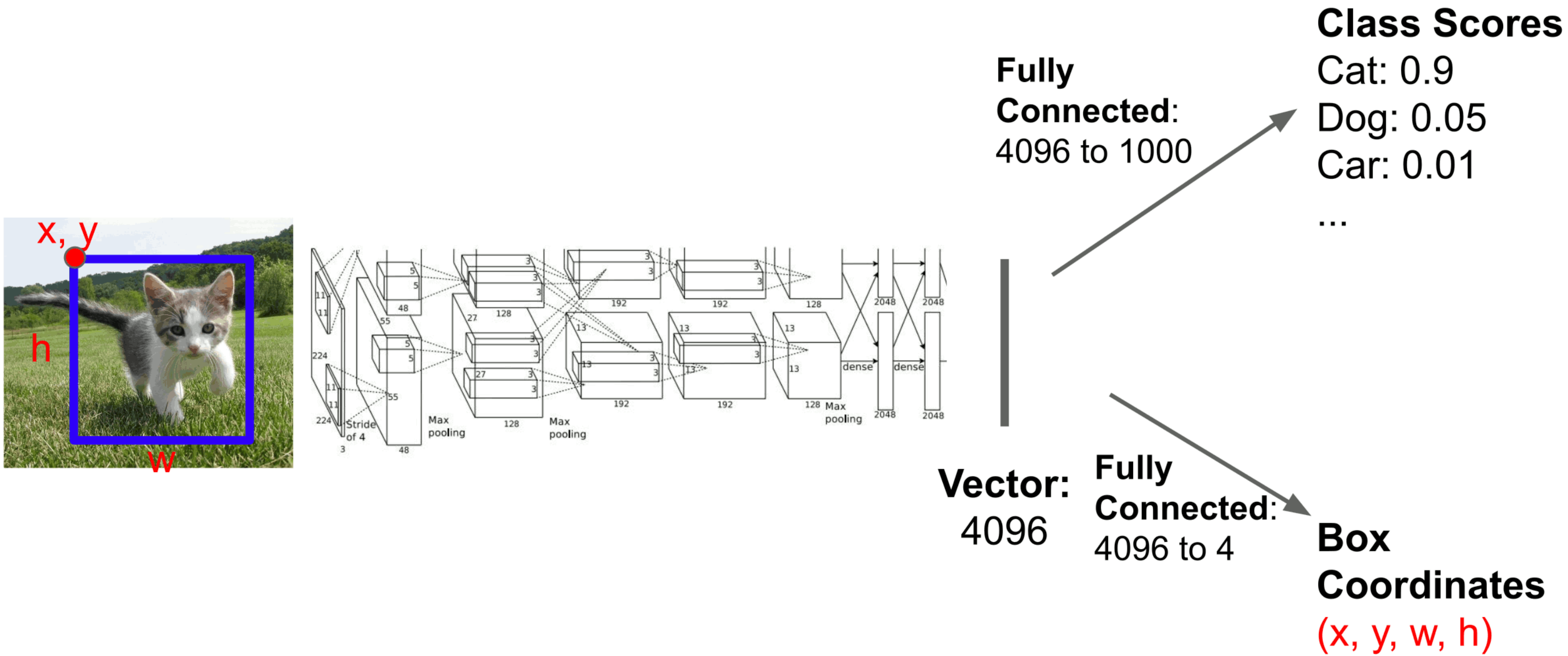
- Human pose estimation

Figure 7 of "Mask R-CNN", https://arxiv.org/abs/1703.06870

Beyond Image Classification



https://www.implantology.or.kr/articles/xml/RvNO/

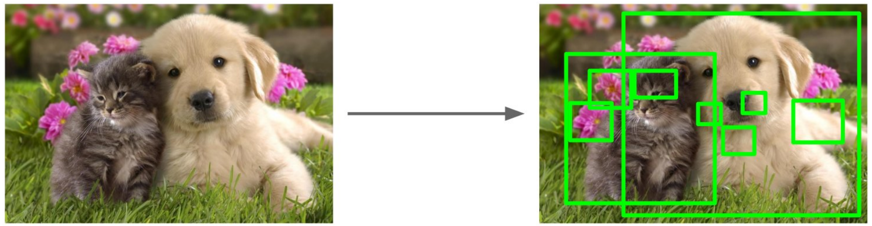


Slide 38 of http://cs231n.stanford.edu/slides/2021/lecture_15.pdf.

We can perform object localization by jointly predicting the bounding box coordinates using regression.

To be able to recognize and localize *several* objects, assume we were given multiple interesting regions of the image, called **regions of interest** (RoI). For each of them, we decide:

- whether it contains an object;
- the location of the object relative to the RoI.

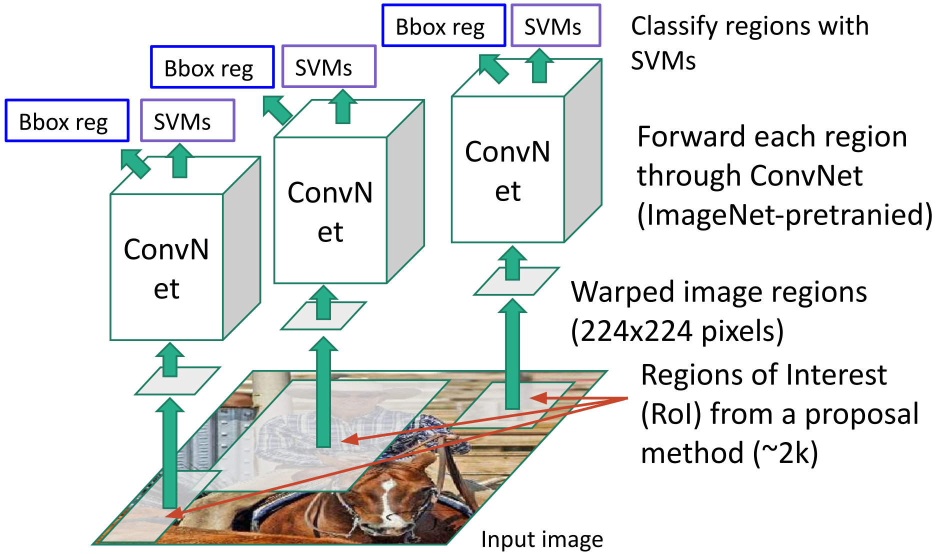


Slide 48 of http://cs231n.stanford.edu/slides/2021/lecture_15.pdf.

In R-CNN, we start with a network pre-trained on ImageNet (VGG-16 is used in the original paper), and we use it to process *every RoI*, rescaling every one of them to the size of 224×224.

For every RoI, two sibling heads are added:

- *classification head* predicts either *background* or one of K object types ($K+1$ in total),
- *bounding box regression head* predicts 4 bounding box parameters relative to RoI.



Slide 54 of http://cs231n.stanford.edu/slides/2021/lecture_15.pdf.

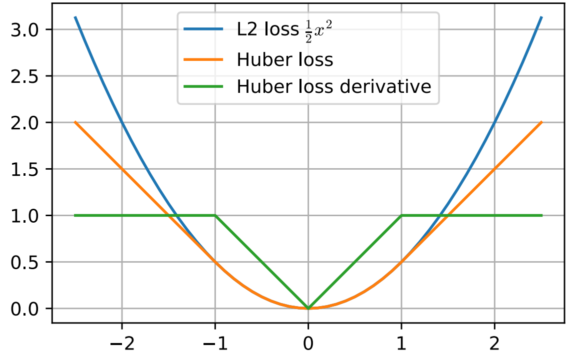
R-CNN – Bounding Boxes

A bounding box is parametrized as follows. Let x_r, y_r, w_r, h_r be center coordinates and width and height of the RoI respectively, and let x, y, w, h be parameters of the bounding box. We represent the bounding box relative to the RoI as follows:

$$\begin{aligned} t_x &= (x - x_r)/w_r, & t_y &= (y - y_r)/h_r, \\ t_w &= \log(w/w_r), & t_h &= \log(h/h_r). \end{aligned}$$

In Fast R-CNN, the $\operatorname{smooth}_{L_1}$ loss, or **Huber loss**, is employed for bounding box parameters:

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

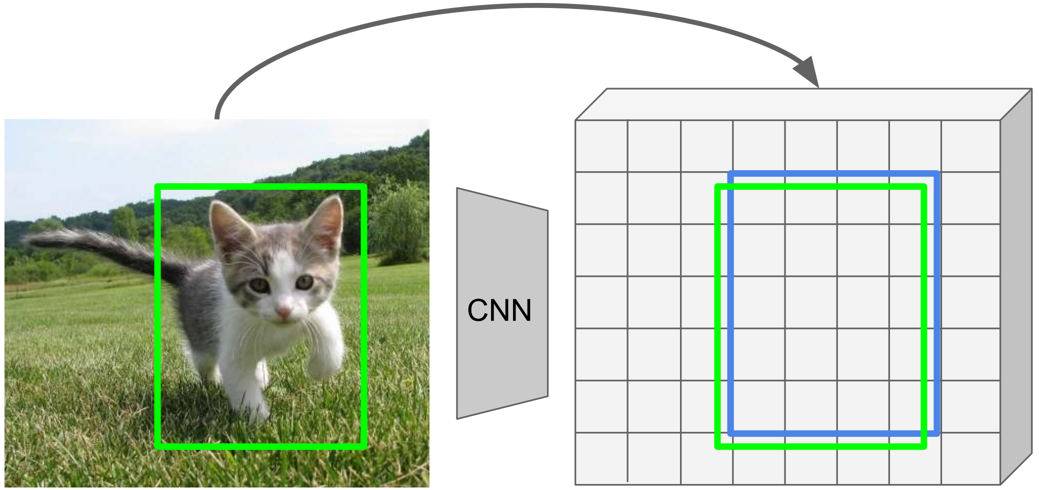


The complete loss is then ($\lambda = 1$ is used in the Fast R-CNN paper)

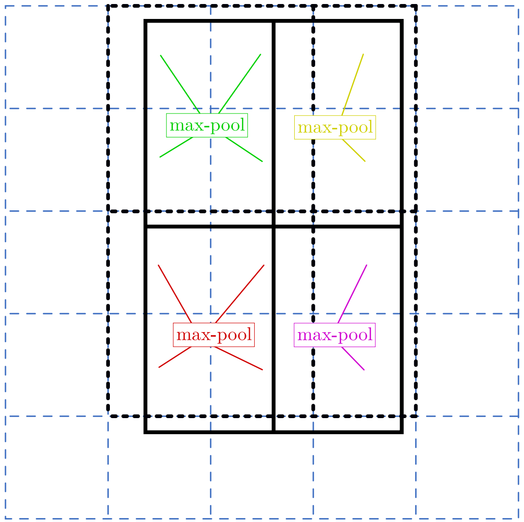
$$L(\hat{c}, \hat{t}, c, t) = L_\text{cls}(\hat{c}, c) + \lambda \cdot [c \geq 1] \cdot \sum\nolimits_{i \in \{\text{x,y,w,h}\}} \operatorname{smooth}_{L_1}(\hat{t}_i - t_i).$$

The R-CNN is slow, because it needs to process every RoI by the convolutional backbone. To speed it up, we might want to first process the whole image by the backbone and only then extract a fixed-size representation for every RoI.

We achieve that using **RoI pooling**, replacing the last max-pool $14 \times 14 \rightarrow 7 \times 7$ VGG layer.

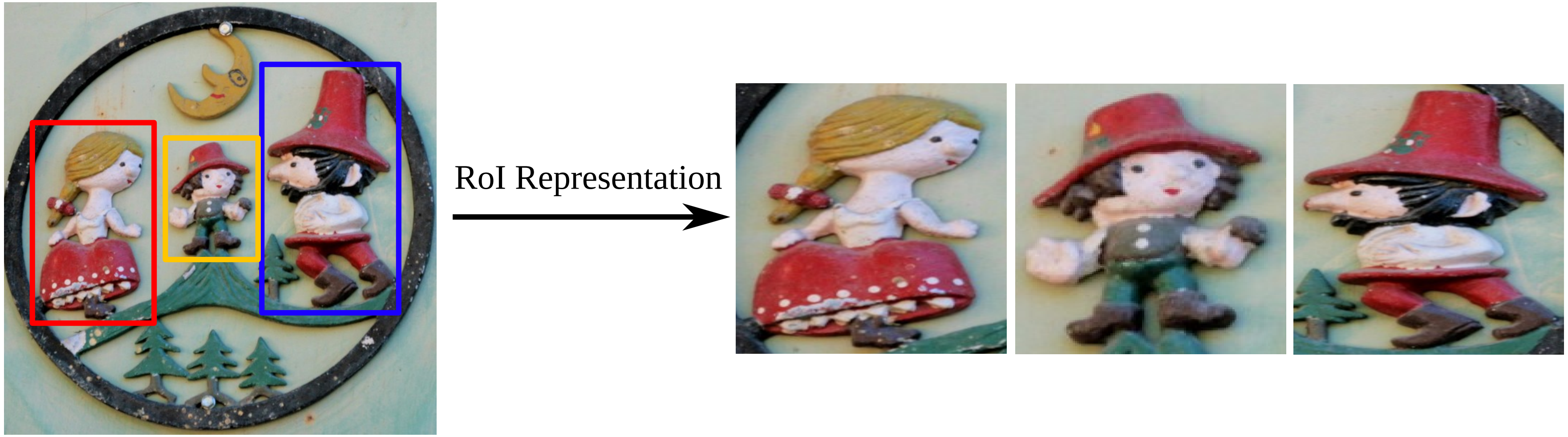


Slide 65 of http://cs231n.stanford.edu/slides/2021/lecture_15.pdf.

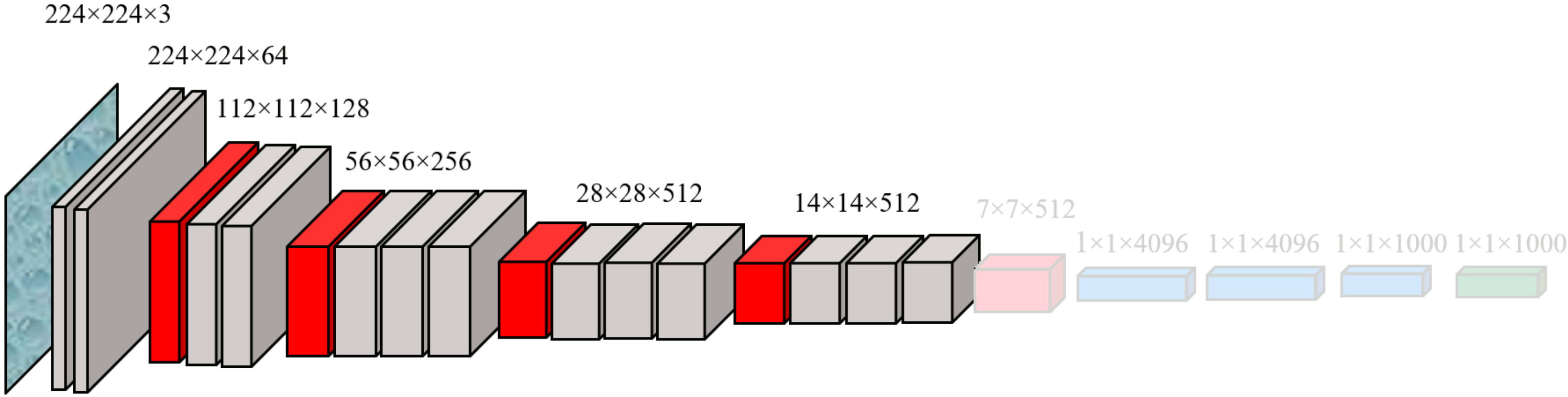


During RoI pooling, we obtain a 7×7 RoI representation by first projecting the RoI to the 14×14 resolution and then computing each of the 7×7 values by **max-pooling** the corresponding "pixels" of the convolutional image features.

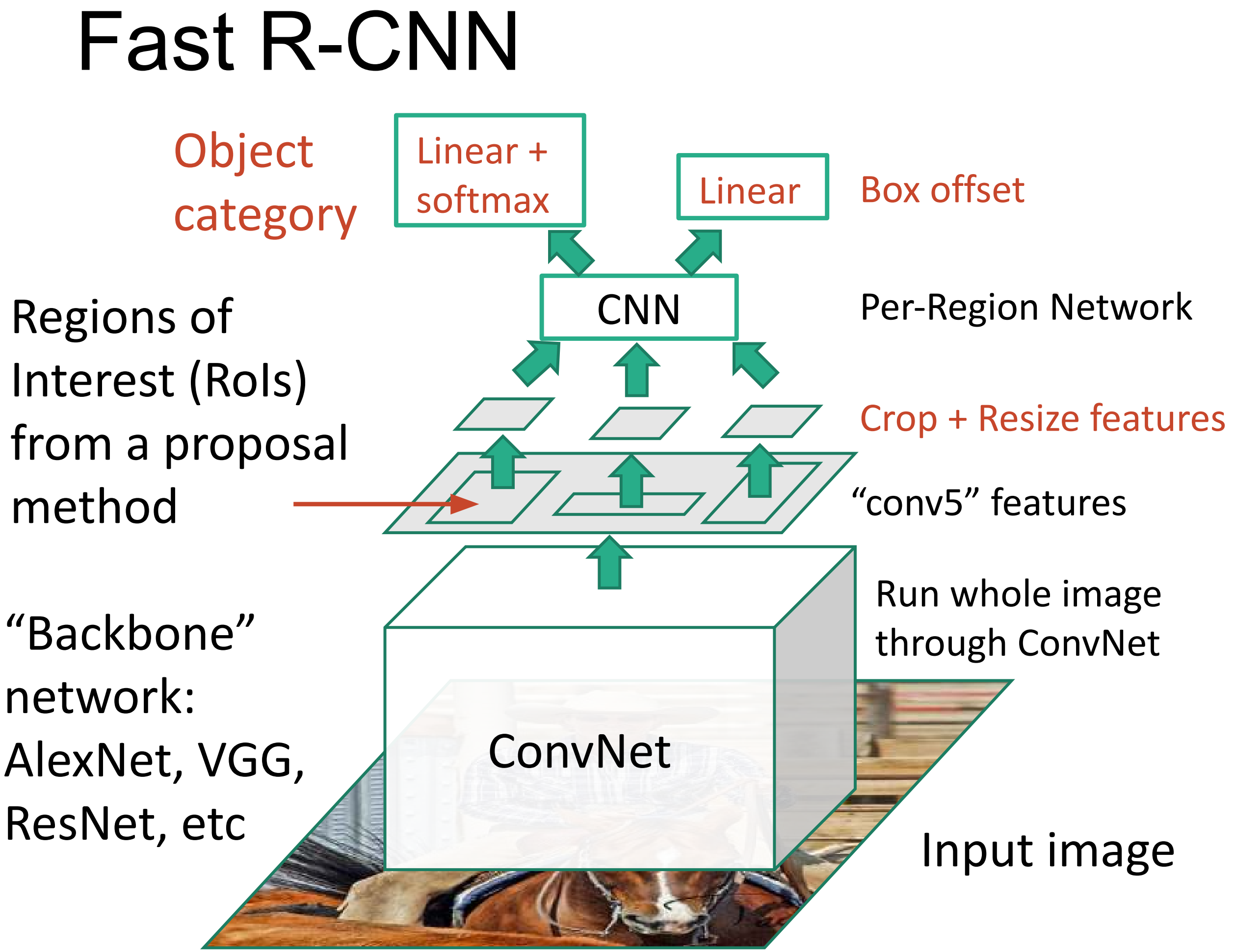
Fast R-CNN

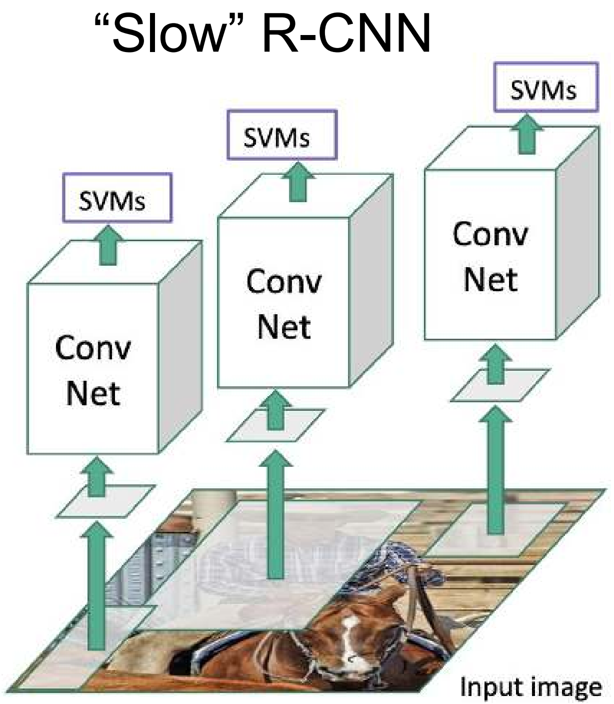


https://commons.wikimedia.org/wiki/File:Tišnov,_Hajánky,_garážová_ozdoba_(6597).jpg



https://en.wikipedia.org/wiki/File:VGG_neural_network.png





Slide 61 of http://cs231n.stanford.edu/slides/2021/lecture_15.pdf.

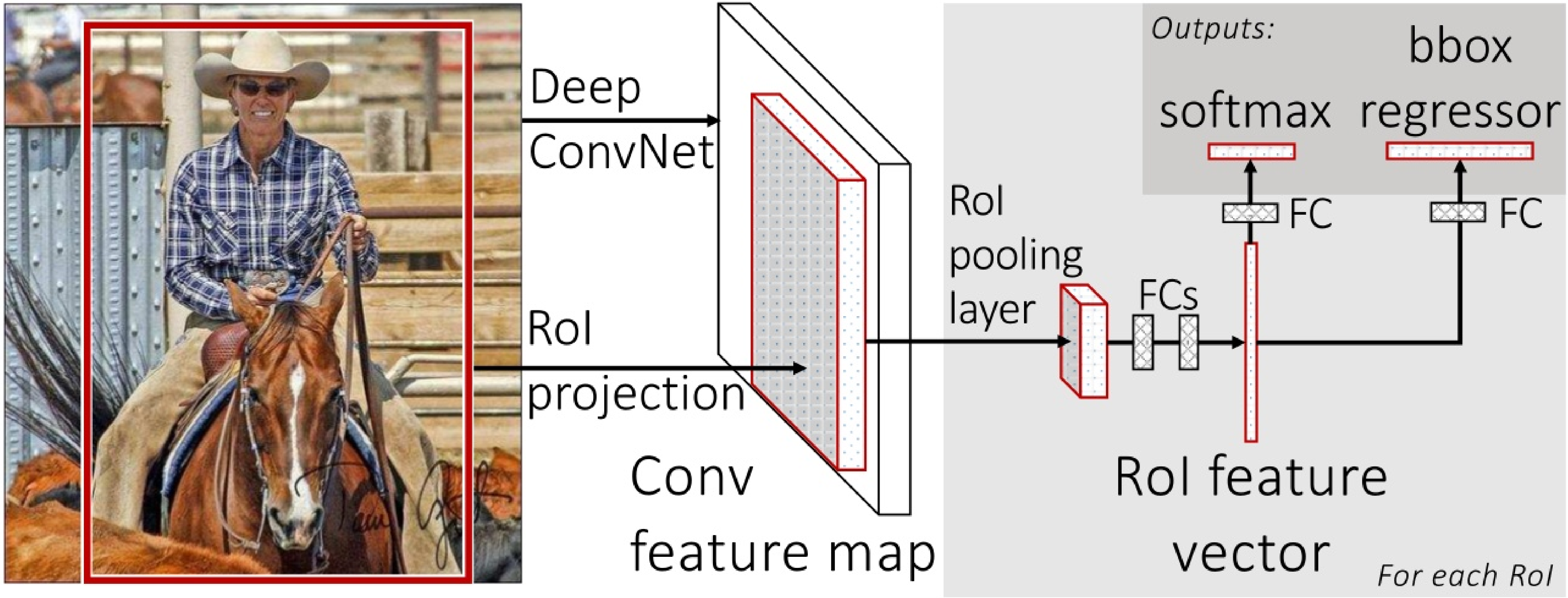


Figure 1 of "Fast R-CNN", https://arxiv.org/abs/1504.08083

Intersection over Union

For two bounding boxes (or two masks) the *intersection over union* (*IoU*) is a ratio of the intersection of the boxes (or masks) and the union of the boxes (or masks).

Choosing RoIs for Training

During training, we use 2 images with 64 RoIs each. The RoIs are selected so that 25% have intersection over union (IoU) overlap with ground-truth boxes at least 0.5; the others are chosen to have the IoU in range $[0.1, 0.5)$, the so-called *hard examples*.

Running Inference

During inference, we utilize all RoIs, but a single object can be found in several of them. To choose the most salient prediction, we perform **non-maximum suppression** – we ignore predictions which have an overlap with a higher scoring prediction of the *same class*, where the overlap is computed using IoU (0.3 threshold is used in the paper). Higher scoring predictions is the ones with higher probability from the *classification head*.

Average Precision

Evaluation is performed using *Average Precision* (AP or AP_{50}).

We assume all bounding boxes (or masks) produced by a system have confidence values which can be used to rank them. Then, for a single class, we take the boxes (or masks) in the order of the ranks and generate precision/recall curve, considering a bounding box correct if it has IoU at least 50% with any ground-truth box.

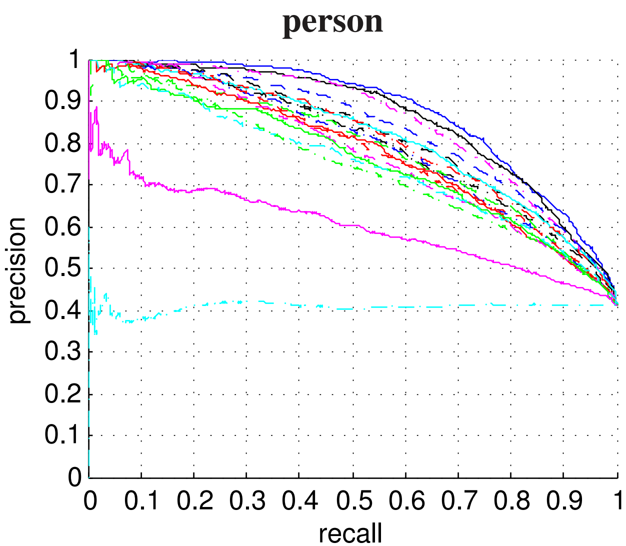


Figure 6 of "The PASCAL Visual Object Classes (VOC) Challenge", http://homepages.inf.ed.ac.uk/ckiw/postscript/ijcv_voc09.pdf

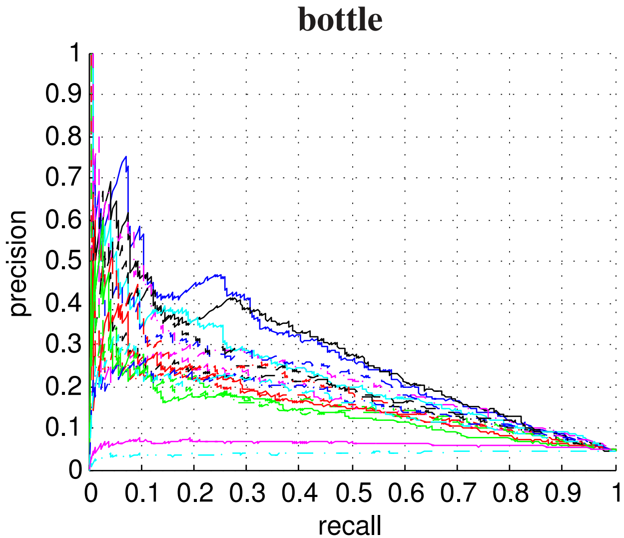
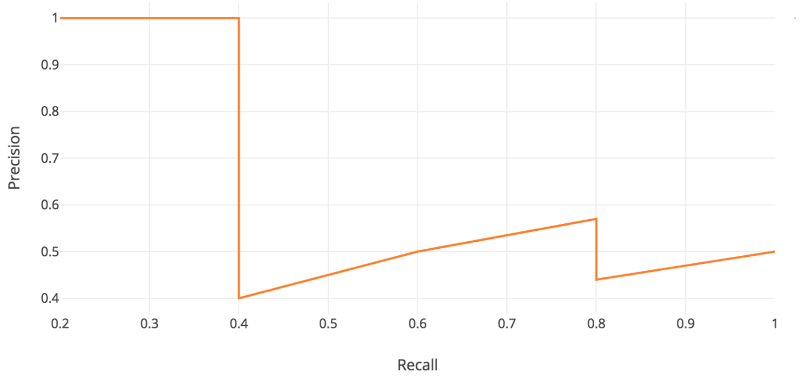
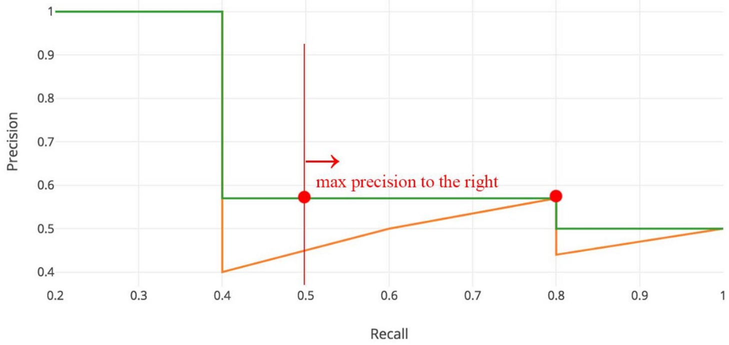


Figure 6 of "The PASCAL Visual Object Classes (VOC) Challenge", http://homepages.inf.ed.ac.uk/ckiw/postscript/ijcv_voc09.pdf

The general idea of AP is to compute the area under the precision/recall curve.



https://miro.medium.com/max/1400/1*VenTq4IgxjmIpOXWdFb-jg.png

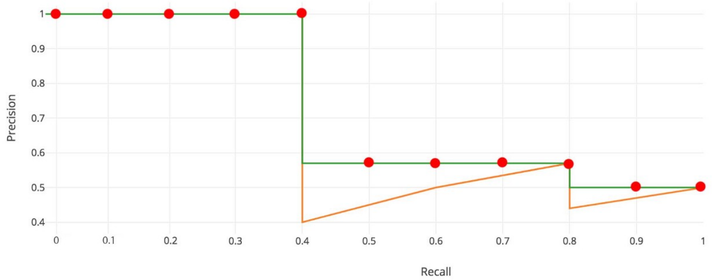


https://miro.medium.com/max/1400/1*pmSxeb4EfdGnzT6Xa68GEQ.jpeg

We start by interpolating the precision/recall curve, so that it is always nonincreasing.

Finally, the average precision for a single class is an average of precision at recall $0.0, 0.1, 0.2, \ldots, 1.0$.

The final AP is a mean of average precision of all classes.



https://miro.medium.com/max/1400/1*naz02wO-XMywlwAdFzF-GA.jpeg

Object Detection Evaluation – Average Precision

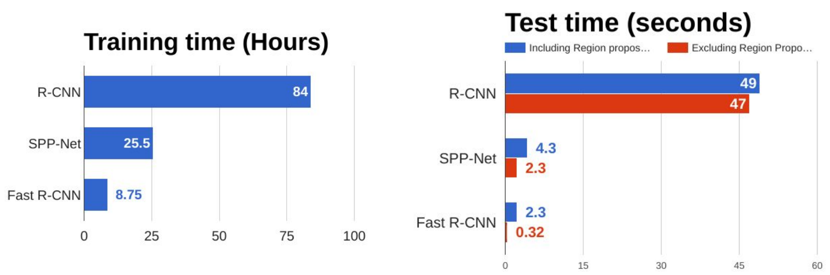
For the COCO dataset, the AP is computed slightly differently. First, it is an average over 101 recall points $0.00, 0.01, 0.02, \ldots, 1.00$.

In the original metric, IoU of 50% is enough to consider a prediction valid. We can generalize the definition to AP_t, where an object prediction is considered valid if IoU is at least t%.

The main COCO metric, denoted just AP, is the mean of $AP_{50}, AP_{55}, AP_{60}, \ldots, AP_{95}$.

Metric	Description
AP	Mean of $AP_{50}, AP_{55}, AP_{60}, AP_{65}, \ldots, AP_{95}$
AP_{50}	AP at IoU 50%
AP_{75}	AP at IoU 75%
AP_S	AP for small objects: $area < 32^2$
AP_M	AP for medium objects: $32^2 < area < 96^2$
AP_L	AP for large objects: $96^2 < area$

Even if Fast R-CNN is much faster then R-CNN, it can still be improved, considering that the most problematic and time consuming part is generating the RoIs.



Slide 76 of http://cs231n.stanford.edu/slides/2021/lecture_15.pdf.

Faster R-CNN extends Fast R-CNN by including a **region proposal network (RPN)**, whose goal is to generate the RoIs automatically.

The regional proposal networks produces the so-called **region proposals**, which then play the role of RoIs in the rest of the pipeline (i.e., the Fast R-CNN).

The region proposals are generated similarly to how predictions are generated in Fast R-CNN. We start with several **anchors** and from each anchor we generate either a single region proposal or nothing.

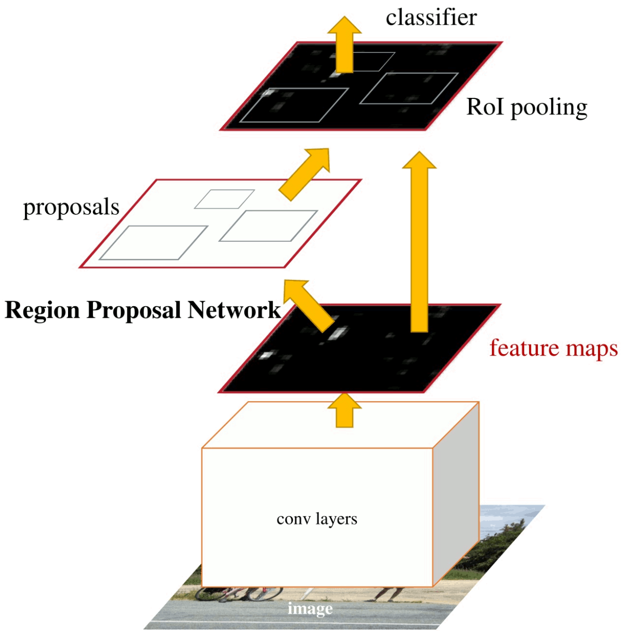
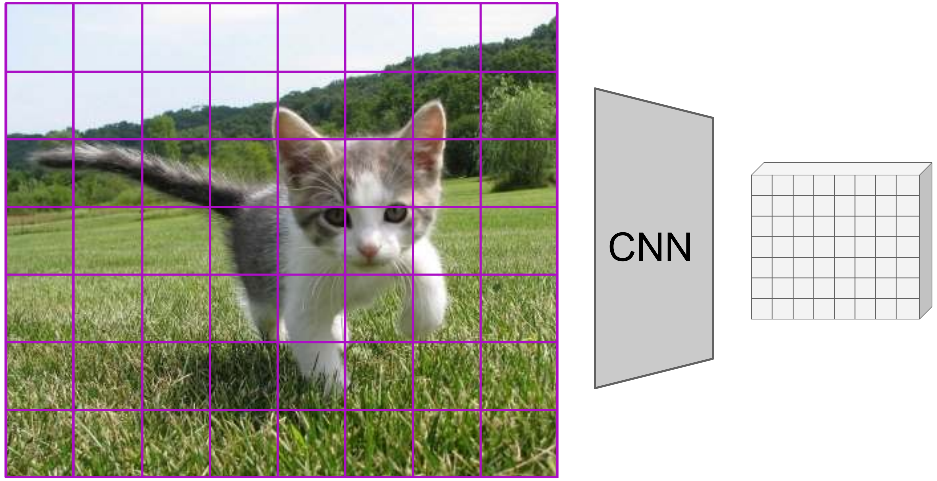


Figure 2 of "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", https://arxiv.org/abs/1506.01497

If we consider the 14×14 VGG backbone output, each "pixel" corresponds to a region of size 16×16 in the original image.



Adapted from slide 65 of http://cs231n.stanford.edu/slides/2021/lecture_15.pdf.

We can therefore interpret each value in the 14×14 output as a representation of a part of the image *centered* in the corresponding image region, and try predicting a region proposal from **every one** of them.

We call the dense grid of image regions from which we are predicting the proposals the **anchors**. They have fixed size, and in practice we use *several* anchors per position.

For every anchor, we classify it in two classes (background, object) and also predict the region proposal bounding box relatively to the anchor, exactly as in (Fast) R-CNN.

We perform the classification and the bounding box regression by first running a 3×3 convolution followed by ReLU on the 14×14 VGG output, and then attaching the two heads. Assuming there are A anchors on every position:

- the classification head generates $2A$ outputs, performing $\operatorname{softmax}$ on every 2 of them;
- the regression head generates $4A$ region proposal coordinates.

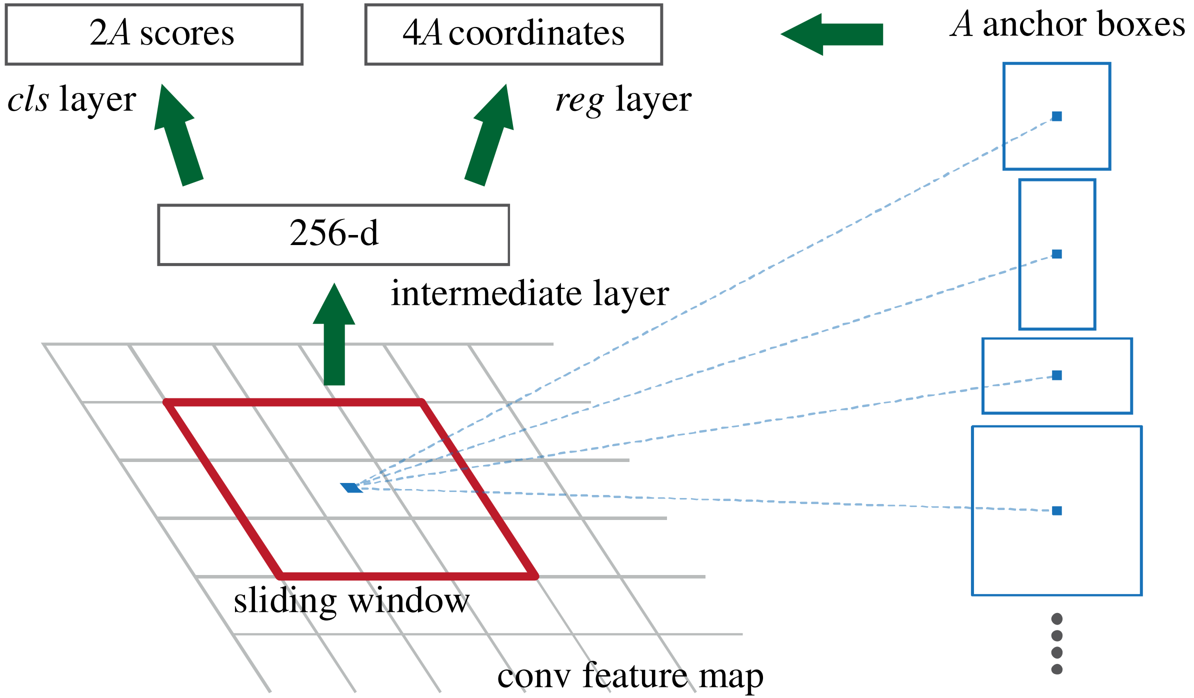


Figure 3 of "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", https://arxiv.org/abs/1506.01497

The authors consider 3 scales $(128^2, 256^2, 512^2)$ and 3 aspect ratios $(1:1, 1:2, 2:1)$.

During training, we generate

- positive training examples for every anchor that has the highest IoU with a ground-truth box;
- furthermore, a positive example is also any anchor with IoU at least 0.7 for any ground-truth box;
- negative training examples for every anchor that has IoU at most 0.3 with all ground-truth boxes;
- the positive and negative examples are generated with a ratio *up to* 1:1 (less, if there are not enough positive examples).

During inference, we consider all predicted non-background regions, run non-maximum suppression on them using a 0.7 IoU threshold, and then take N top-scored regions (i.e., the ones with the highest probability from the classification head) – the paper uses 300 proposals, compared to 2000 in the Fast R-CNN.

Faster R-CNN

Table 3: 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 2000. †: this number was reported in [2]; using the repository provided by this paper, this result is higher (68.1).

method	# proposals	data	mAP (%)
SS	2000	07	66.9†
SS	2000	07+12	70.0
RPN+VGG, unshared	300	07	68.5
RPN+VGG, shared	300	07	69.9
RPN+VGG, shared	300	07+12	**73.2**
RPN+VGG, shared	300	COCO+07+12	**78.8**

Table 4: 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 2000. †: http://host.robots.ox.ac.uk:8080/anonymous/HZJTQA.html. ‡: http://host.robots.ox.ac.uk:8080/anonymous/YNPLXB.html. §: http://host.robots.ox.ac.uk:8080/anonymous/XEDH10.html.

method	# proposals	data	mAP (%)
SS	2000	12	65.7
SS	2000	07++12	68.4
RPN+VGG, shared†	300	12	67.0
RPN+VGG, shared‡	300	07++12	**70.4**
RPN+VGG, shared§	300	COCO+07++12	**75.9**

Tables 3 and 4 of "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", https://arxiv.org/abs/1506.01497

Two-stage Detectors

The Faster R-CNN is a so-called **two-stage** detector, where the regions are refined twice – once in the region proposal network, and then in the final bounding box regressor.

Several **single-stage** detector architectures have been proposed, mainly because they are faster and smaller, but until circa 2017 the two-stage detectors achieved better results.

Mask R-CNN

Straightforward extension of Faster R-CNN able to produce image segmentation (i.e., masks for every object).

Figure 2 of "Mask R-CNN", https://arxiv.org/abs/1703.06870

Mask R-CNN – Architecture

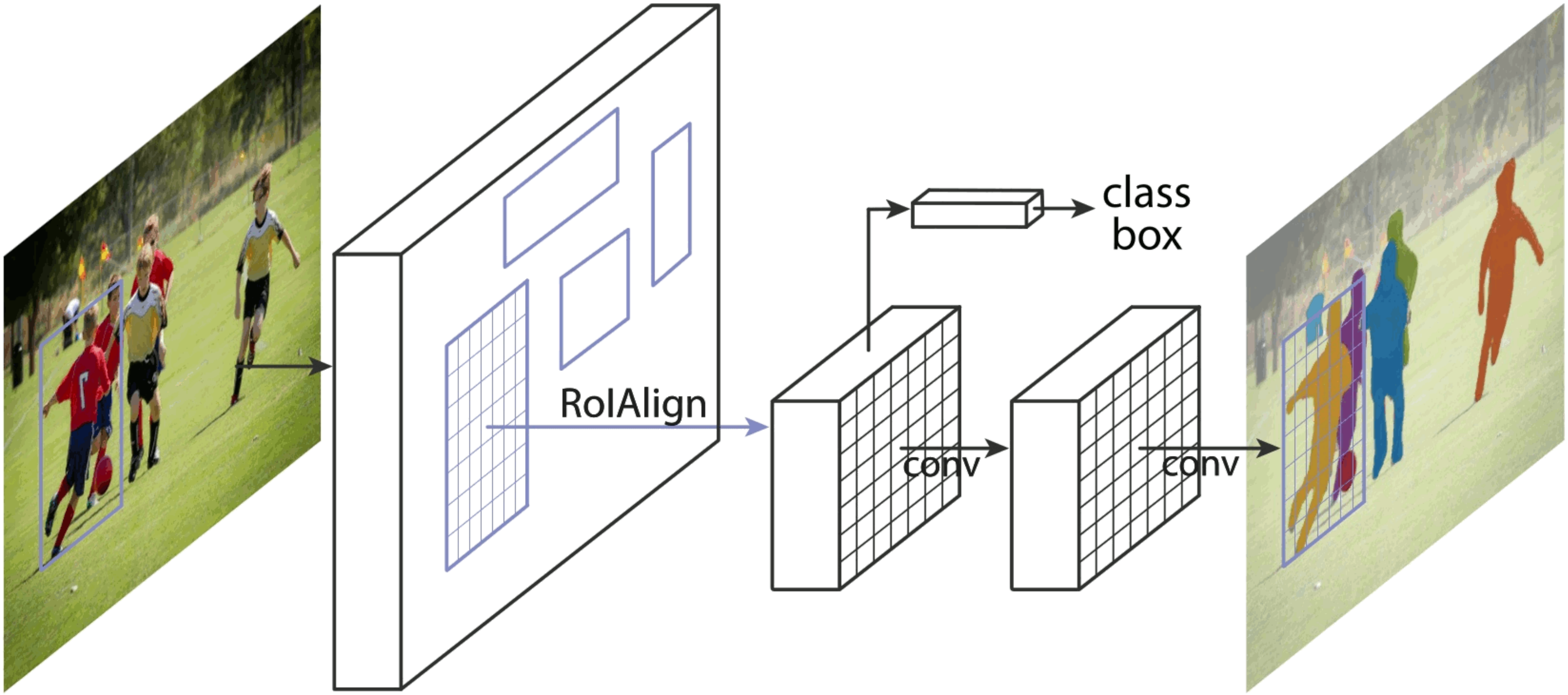
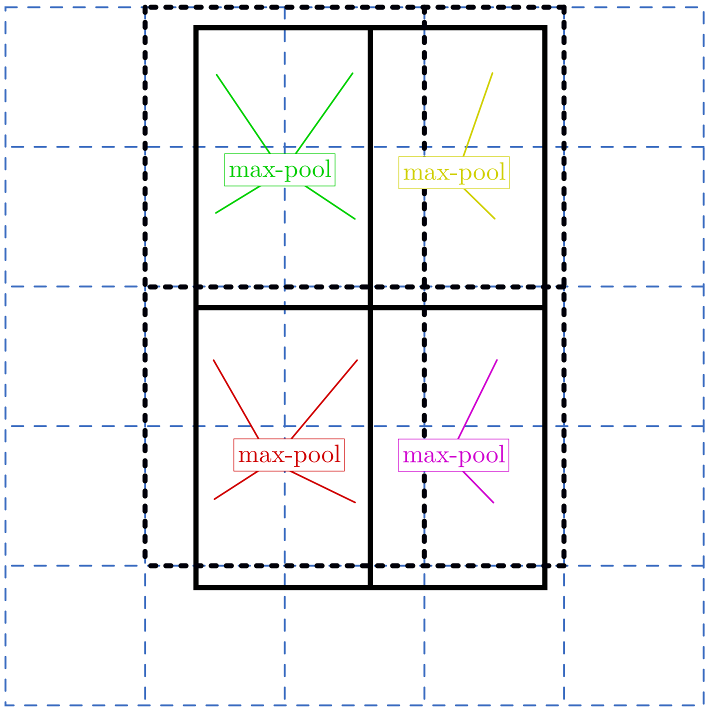


Figure 1 of "Mask R-CNN", https://arxiv.org/abs/1703.06870

Mask R-CNN – RoIAlign

More precise alignment is required for the RoI in order to predict the masks. Instead of quantization and max-pooling in RoI pooling, **RoIAlign** uses bilinear interpolation of features at four regularly sampled locations in each RoI bin and averages them.



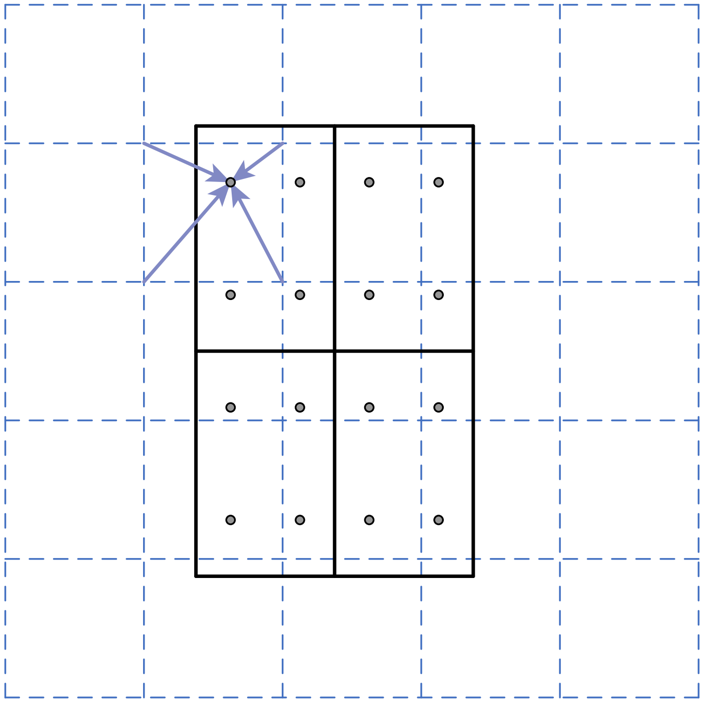


Figure 3 of "Mask R-CNN", https://arxiv.org/abs/1703.06870

TensorFlow provides `tf.image.crop_and_resize` capable of implementing RoIAlign.

Mask R-CNN

Masks are predicted in a third branch of the object detector.

- Higher resolution of the mask is usually needed (at least 14×14, or even more).
- The masks are predicted for each class separately.
- The masks are predicted using convolutions instead of fully connected layers (the upscaling convolutions are 2×2 with stride 2).

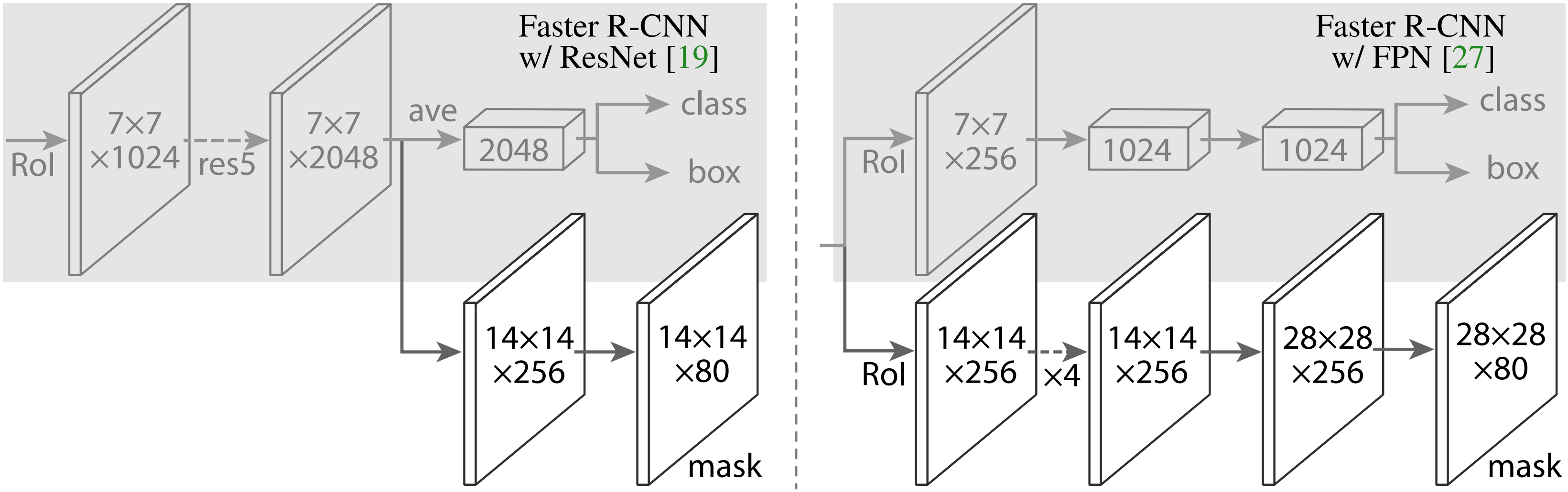


Figure 4 of "Mask R-CNN", https://arxiv.org/abs/1703.06870

Improvements from Nov 2021: all convs (except for the output layer) are followed by BN, the *class&bbox* head uses 4 convs instead of 2 MLPs, RPN contains two convs instead of one.

net-depth-features	AP	AP_{50}	AP_{75}
ResNet-50-C4	30.3	51.2	31.5
ResNet-101-C4	32.7	54.2	34.3
ResNet-50-FPN	33.6	55.2	35.3
ResNet-101-FPN	35.4	57.3	37.5
ResNeXt-101-FPN	**36.7**	**59.5**	**38.9**

(a) **Backbone Architecture**: Better backbones bring expected gains: deeper networks do better, FPN outperforms C4 features, and ResNeXt improves on ResNet.

	AP	AP_{50}	AP_{75}
softmax	24.8	44.1	25.1
sigmoid	**30.3**	**51.2**	**31.5**
	+5.5	*+7.1*	*+6.4*

(b) **Multinomial *vs.* Independent Masks** (ResNet-50-C4): *Decoupling* via per-class binary masks (sigmoid) gives large gains over multinomial masks (softmax).

	align?	bilinear?	agg.	AP	AP_{50}	AP_{75}
RoIPool [12]			max	26.9	48.8	26.4
RoIWarp [10]		✓	max	27.2	49.2	27.1
		✓	ave	27.1	48.9	27.1
RoIAlign	✓	✓	max	**30.2**	**51.0**	**31.8**
	✓	✓	ave	**30.3**	**51.2**	**31.5**

(c) **RoIAlign** (ResNet-50-C4): Mask results with various RoI layers. Our RoIAlign layer improves AP by ~3 points and AP_{75} by ~5 points. Using proper alignment is the only factor that contributes to the large gap between RoI layers.

	AP	AP_{50}	AP_{75}	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}
RoIPool	23.6	46.5	21.6	28.2	52.7	26.9
RoIAlign	**30.9**	**51.8**	**32.1**	**34.0**	**55.3**	**36.4**
	+7.3	*+ 5.3*	*+10.5*	*+5.8*	*+2.6*	*+9.5*

(d) **RoIAlign** (ResNet-50-**C5**, *stride 32*): Mask-level and box-level AP using *large-stride* features. Misalignments are more severe than with stride-16 features (Table 2c), resulting in big accuracy gaps.

	mask branch	AP	AP_{50}	AP_{75}
MLP	fc: $1024 \rightarrow 1024 \rightarrow 80 \cdot 28^2$	31.5	53.7	32.8
MLP	fc: $1024 \rightarrow 1024 \rightarrow 1024 \rightarrow 80 \cdot 28^2$	31.5	54.0	32.6
FCN	conv: $256 \rightarrow 256 \rightarrow 256 \rightarrow 256 \rightarrow 256 \rightarrow 80$	**33.6**	**55.2**	**35.3**

(e) **Mask Branch** (ResNet-50-FPN): Fully convolutional networks (FCN) *vs.* multi-layer perceptrons (MLP, fully-connected) for mask prediction. FCNs improve results as they take advantage of explicitly encoding spatial layout.

Table 2. **Ablations**. We train on `trainval35k`, test on `minival`, and report *mask* AP unless otherwise noted.

Table 2 of "Mask R-CNN", https://arxiv.org/abs/1703.06870

Figure 7 of "Mask R-CNN", https://arxiv.org/abs/1703.06870

- Testing applicability of Mask R-CNN architecture.
- Keypoints (e.g., left shoulder, right elbow, …) are detected as independent one-hot masks of size 56×56 with $\operatorname{softmax}$ output function.

	AP^{kp}	AP^{kp}_{50}	AP^{kp}_{75}	AP^{kp}_{M}	AP^{kp}_{L}
CMU-Pose+++ [6]	61.8	84.9	67.5	57.1	68.2
G-RMI [32]†	62.4	84.0	68.5	**59.1**	68.1
Mask R-CNN, keypoint-only	62.7	87.0	68.4	57.4	71.1
Mask R-CNN, keypoint & mask	**63.1**	**87.3**	**68.7**	57.8	**71.4**

Table 4 of "Mask R-CNN", https://arxiv.org/abs/1703.06870

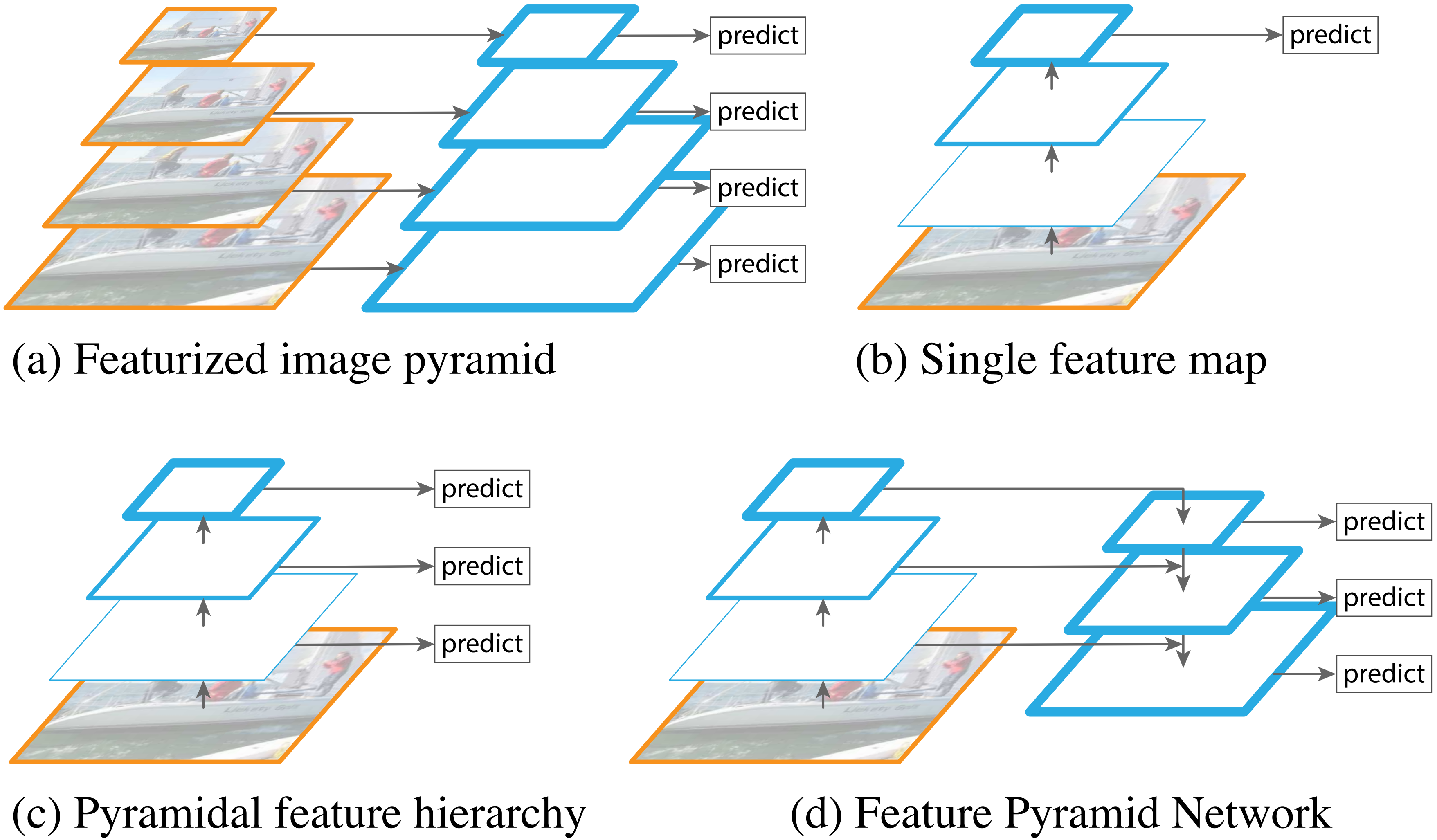


Figure 1 of "Feature Pyramid Networks for Object Detection", https://arxiv.org/abs/1612.03144

Feature Pyramid Networks

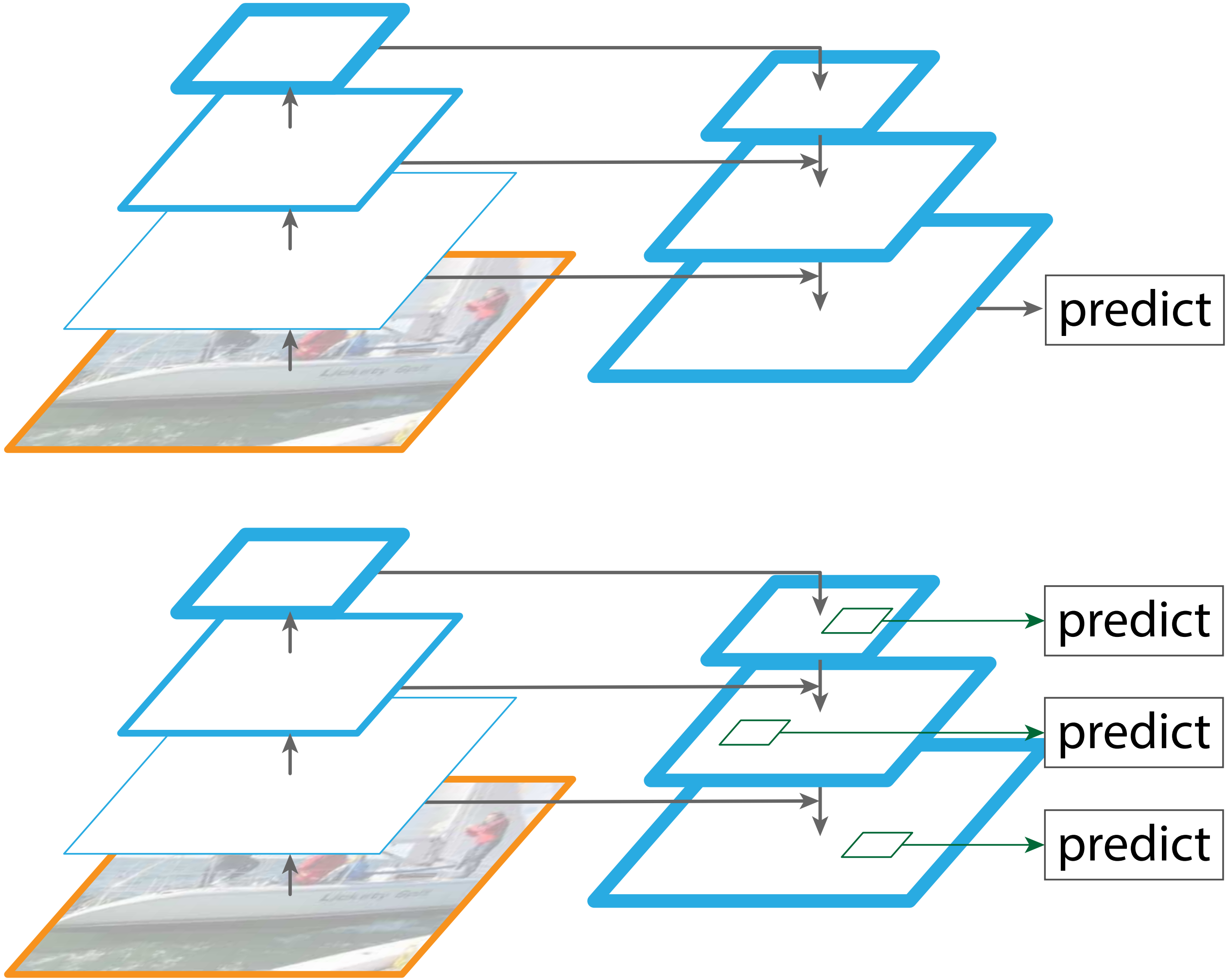


Figure 2 of "Feature Pyramid Networks for Object Detection", https://arxiv.org/abs/1612.03144

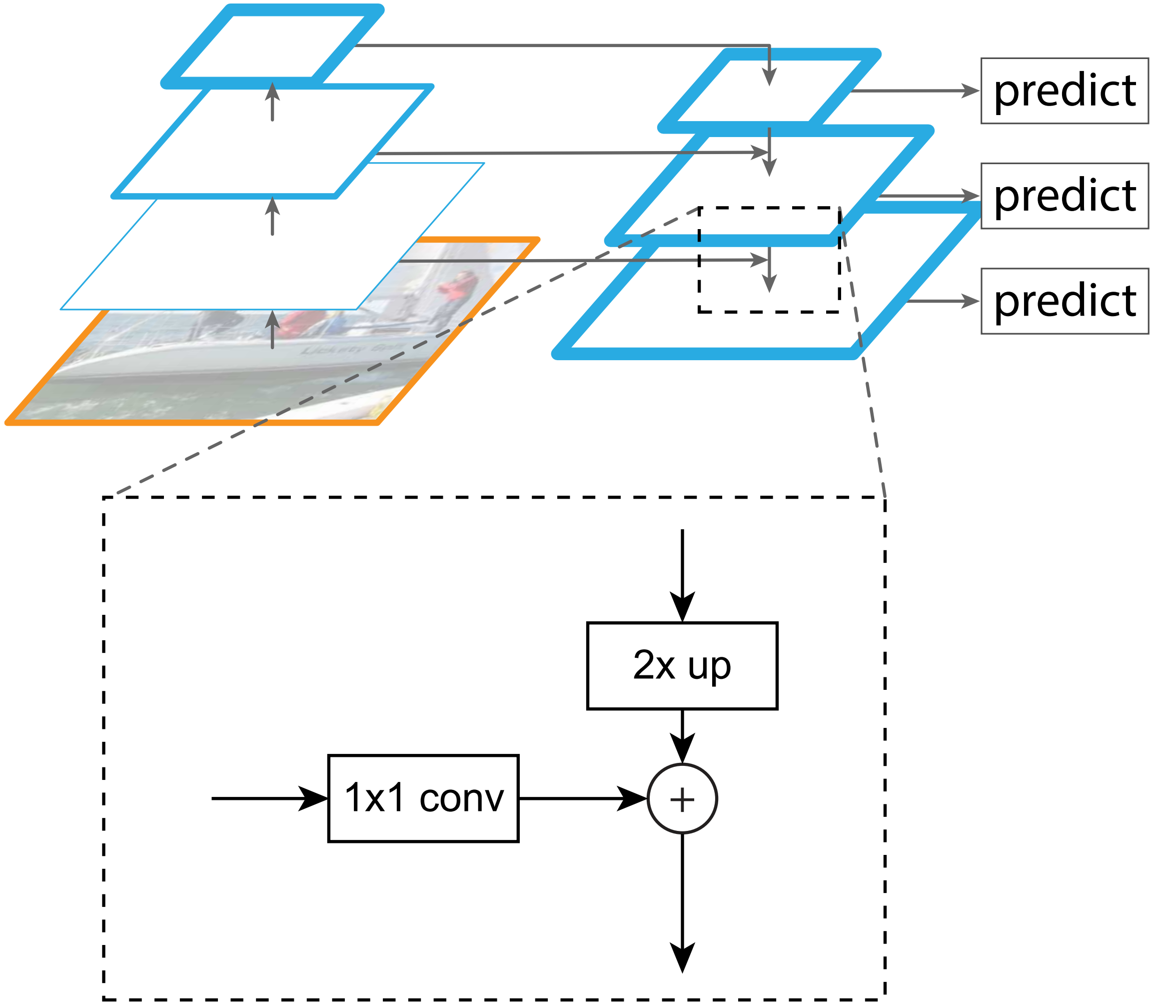


Figure 3 of "Feature Pyramid Networks for Object Detection", https://arxiv.org/abs/1612.03144

We employ FPN as a backbone in Faster R-CNN.

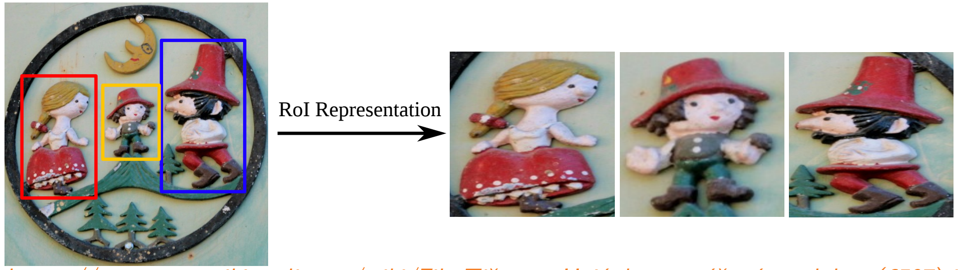
Assuming ResNet-like network with 224×224 input, we denote $C_2, C_3, \ldots, C_5$ the image features of the last convolutional layer of size $56 \times 56, 28 \times 28, \ldots, 7 \times 7$ (i.e., C_i indicates a downscaling of 2^i). The FPN representations incorporating the smaller resolution features are denoted as $P_2, \ldots, P_5$, each consisting of 256 channels; the classification heads are shared.

In both the RPN and the Fast R-CNN, authors utilize the $P_2, \ldots, P_5$ representations, considering single-size anchors for every P_i (of size $32^2, 64^2, 128^2, 256^2$, respectively). However, three aspect ratios $(1:1, 1:2, 2:1)$ are still used.

method	backbone	competition	image pyramid	test-dev					test-std				
				$AP_{@.5}$	AP	AP_s	AP_m	AP_l	$AP_{@.5}$	AP	AP_s	AP_m	AP_l
ours, Faster R-CNN on **FPN**	ResNet-101	-		**59.1**	**36.2**	**18.2**	**39.0**	48.2	**58.5**	**35.8**	**17.5**	**38.7**	47.8
*Competition-winning **single-model** results follow:*													
G-RMI†	Inception-ResNet	2016		-	34.7	-	-	-	-	-	-	-	-
AttractioNet‡ [10]	VGG16 + Wide ResNet§	2016	✓	53.4	35.7	15.6	38.0	**52.7**	52.9	35.3	14.7	37.6	**51.9**
Faster R-CNN +++ [16]	ResNet-101	2015	✓	55.7	34.9	15.6	38.7	50.9	-	-	-	-	-
Multipath [40] (on minival)	VGG-16	2015		49.6	31.5	-	-	-	-	-	-	-	-
ION‡ [2]	VGG-16	2015		53.4	31.2	12.8	32.9	45.2	52.9	30.7	11.8	32.8	44.8

Table 4 of "Feature Pyramid Networks for Object Detection", https://arxiv.org/abs/1612.03144

For single-stage object detection architectures, *class imbalance* has been identified as the main issue preventing obtaining performance comparable to two-stage detectors. In a single-stage detector, there can be tens of thousands of anchors, with only dozens of useful training examples.



https://commons.wikimedia.org/wiki/File:Tišnov,_Hajánky,_garážová_ozdoba_(6597).jpg

Cross-entropy loss is computed as

$$\mathcal{L}_\text{cross-entropy} = -\log p_\text{model}(y|x).$$

Focal-loss (loss focused on hard examples) is proposed as

$$\mathcal{L}_\text{focal-loss} = -(1 - p_\text{model}(y|x))^\gamma \cdot \log p_\text{model}(y|x).$$

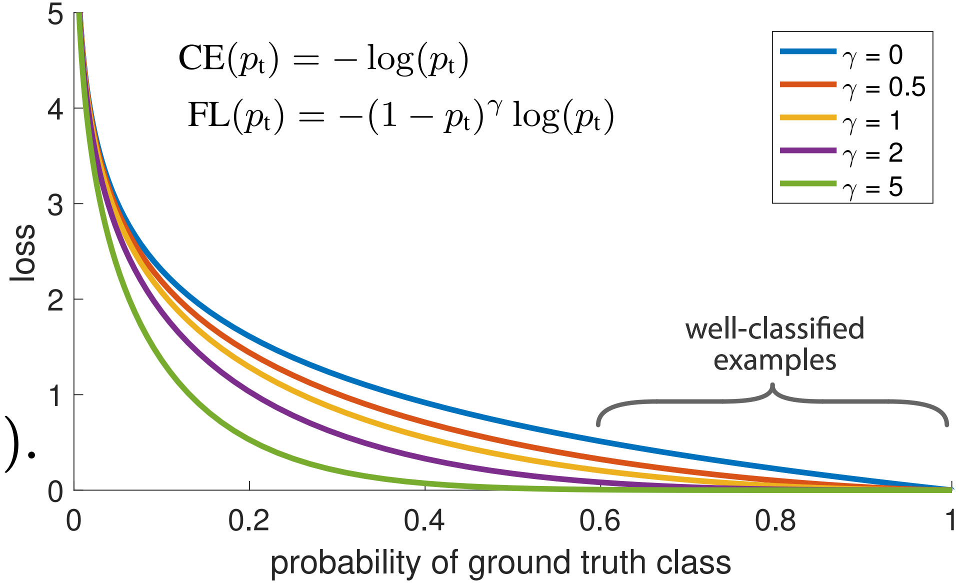


Figure 1 of "Focal Loss for Dense Object Detection", https://arxiv.org/abs/1708.02002

For $\gamma = 0$, focal loss is equal to cross-entropy loss.

Authors reported that $\gamma = 2$ worked best for them for training a single-stage detector.

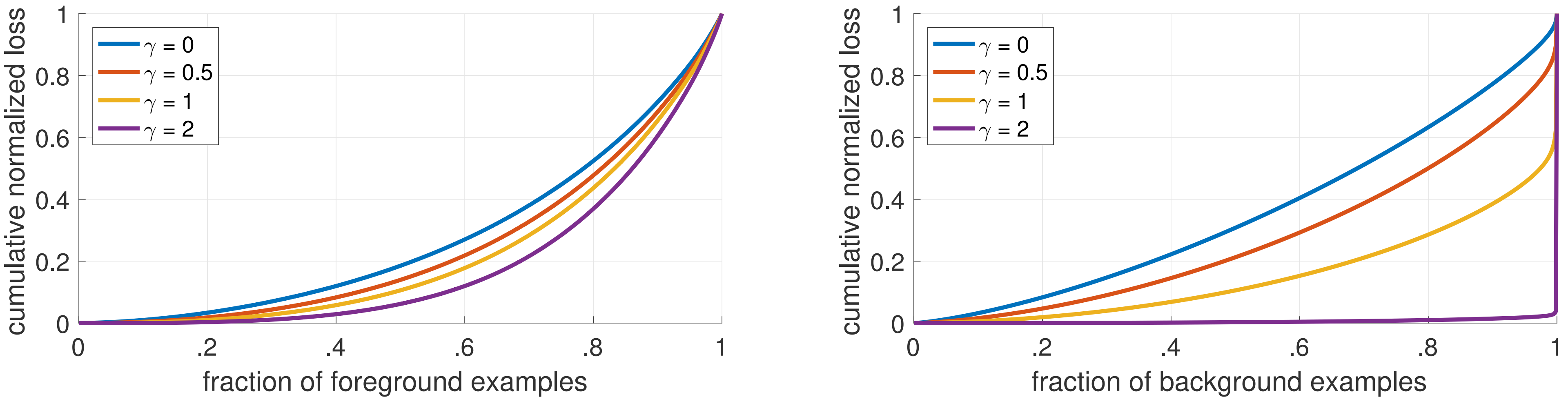


Figure 4. Cumulative distribution functions of the normalized loss for positive and negative samples for different values of γ for a *converged* model. The effect of changing γ on the distribution of the loss for positive examples is minor. For negatives, however, increasing γ heavily concentrates the loss on hard examples, focusing nearly all attention away from easy negatives.

Figure 4 of "Focal Loss for Dense Object Detection", https://arxiv.org/abs/1708.02002

Focal Loss and Class Imbalance

Focal loss is connected to another solution to class imbalance – we might introduce weighting factor $\alpha \in (0, 1)$ for one class and $1 - \alpha$ for the other class, arriving at

$$-\alpha_y \cdot \log p_\textrm{model}(y|x).$$

The weight α might be set to the inverse class frequency or treated as a hyperparameter.

Even if weighting focuses more on low-frequent class, it does not distinguish between easy and hard examples, contrary to focal loss.

In practice, the focal loss is usually used together with class weighting:

$$-\alpha_y \cdot (1 - p_\textrm{model}(y|x))^\gamma \cdot \log p_\textrm{model}(y|x).$$

For example, authors report that $\alpha = 0.25$ (weight of the rare class) works best with $\gamma = 2$.

RetinaNet is a single-stage detector, using feature pyramid network architecture. Built on top of ResNet architecture, the feature pyramid contains levels P_3 through P_7, with each P_l having 256 channels and resolution 2^l times lower than the input. On each pyramid level P_l, we consider 9 anchors for every position, with 3 different aspect ratios (1, $1:2$, $2:1$) and with 3 different sizes $(\{2^0, 2^{1/3}, 2^{2/3}\} \cdot 4 \cdot 2^l)^2$.

Note that ResNet provides only C_3 to C_5 features. C_6 is computed using a 3×3 convolution with stride 2 on C_5, and C_7 is obtained by applying ReLU followed by another 3×3 stride-2 convolution. The C_6 and C_7 are included to improve large object detection.

The classification head and the boundary regression heads are fully convolutional and do not share parameters (but classification heads are shared across levels, and so are the boundary regression heads), generating $anchors \cdot classes$ sigmoids and $anchors$ bounding boxes per position.

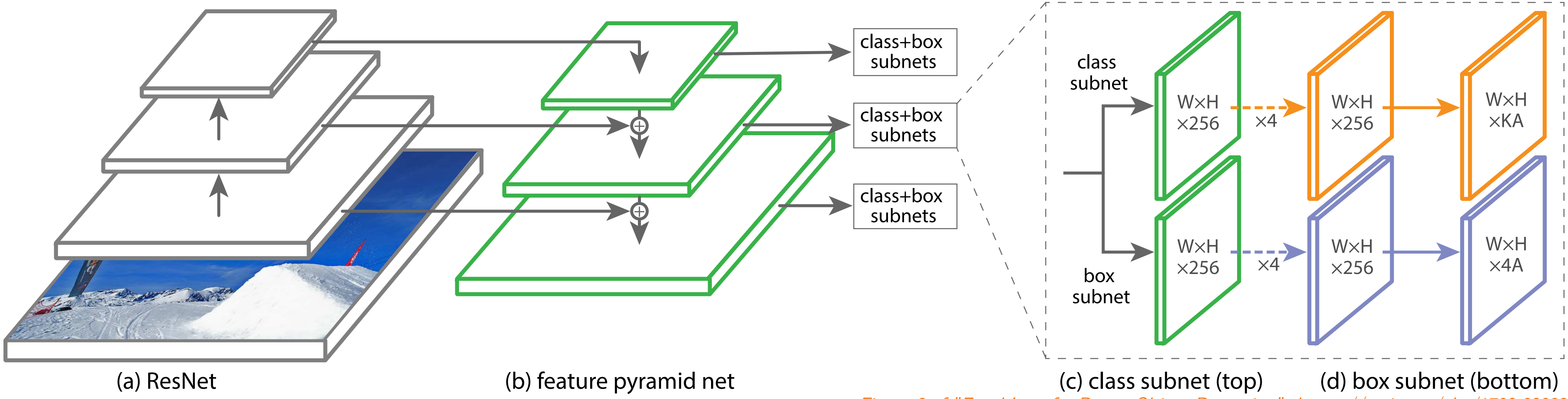


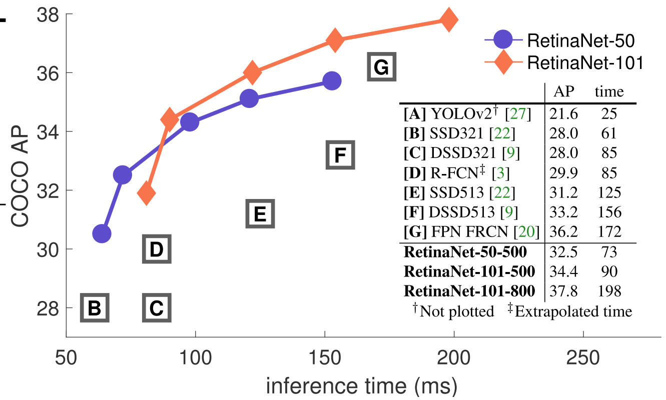
Figure 3 of "Focal Loss for Dense Object Detection", https://arxiv.org/abs/1708.02002

During training, anchors are assigned to ground-truth object boxes if IoU is at least 0.5; to background if IoU with any ground-truth region is at most 0.4 (the rest of anchors is ignored during training). The classification head is trained using focal loss with $\gamma = 2$ and $\alpha = 0.25$ (but according to the paper, all values of γ in $[0.5, 5]$ range work well); the boundary regression head is trained using smooth_{L_1} loss as in Fast(er) R-CNN.

During inference, at most 1000 objects with at least 5% probability from all pyramid levels are considered, and all of them are combined using non-maximum suppression with a threshold of 0.5. Fixed-size training and testing is used, with sizes 400, 500, …, 800 pixels.

	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 of "Focal Loss for Dense Object Detection", https://arxiv.org/abs/1708.02002



	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

Figure 2 of "Focal Loss for Dense Object Detection", https://arxiv.org/abs/1708.02002

Ablations use ResNet-50-FPN backbone trained and tested with 600-pixel images.

α	AP	AP_{50}	AP_{75}
.10	0.0	0.0	0.0
.25	10.8	16.0	11.7
.50	30.2	46.7	32.8
.75	31.1	49.4	33.0
.90	30.8	49.7	32.3
.99	28.7	47.4	29.9
.999	25.1	41.7	26.1

(a) **Varying α for CE loss** ($\gamma = 0$)

γ	α	AP	AP_{50}	AP_{75}
0	.75	31.1	49.4	33.0
0.1	.75	31.4	49.9	33.1
0.2	.75	31.9	50.7	33.4
0.5	.50	32.9	51.7	35.2
1.0	.25	33.7	52.0	36.2
2.0	.25	**34.0**	**52.5**	**36.5**
5.0	.25	32.2	49.6	34.8

(b) **Varying γ for FL** (w. optimal α)

#sc	#ar	AP	AP_{50}	AP_{75}
1	1	30.3	49.0	31.8
2	1	31.9	50.0	34.0
3	1	31.8	49.4	33.7
1	3	32.4	52.3	33.9
2	3	**34.2**	**53.1**	**36.5**
3	3	34.0	52.5	**36.5**
4	3	33.8	52.1	36.2

(c) **Varying anchor scales and aspects**

method	batch size	nms thr	AP	AP_{50}	AP_{75}
OHEM	128	.7	31.1	47.2	33.2
OHEM	256	.7	31.8	48.8	33.9
OHEM	512	.7	30.6	47.0	32.6
OHEM	128	.5	32.8	50.3	35.1
OHEM	256	.5	31.0	47.4	33.0
OHEM	512	.5	27.6	42.0	29.2
OHEM 1:3	128	.5	31.1	47.2	33.2
OHEM 1:3	256	.5	28.3	42.4	30.3
OHEM 1:3	512	.5	24.0	35.5	25.8
FL	n/a	n/a	**36.0**	**54.9**	**38.7**

(d) **FL *vs.* OHEM** baselines (with ResNet-101-FPN)

depth	scale	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	time
50	400	30.5	47.8	32.7	11.2	33.8	46.1	64
50	500	32.5	50.9	34.8	13.9	35.8	46.7	72
50	600	34.3	53.2	36.9	16.2	37.4	47.4	98
50	700	35.1	54.2	37.7	18.0	39.3	46.4	121
50	800	35.7	55.0	38.5	18.9	38.9	46.3	153
101	400	31.9	49.5	34.1	11.6	35.8	48.5	81
101	500	34.4	53.1	36.8	14.7	38.5	49.1	90
101	600	36.0	55.2	38.7	17.4	39.6	49.7	122
101	700	37.1	56.6	39.8	19.1	40.6	49.4	154
101	800	37.8	57.5	40.8	20.2	41.1	49.2	198

(e) **Accuracy/speed trade-off** RetinaNet (on `test-dev`)

Table 1 of "Focal Loss for Dense Object Detection", https://arxiv.org/abs/1708.02002

EfficientDet builds up on EfficientNet and delivered state-of-the-art performance in Nov 2019 with minimum time and space requirements (however, its performance has already been surpassed significantly). It is a single-scale detector similar to RetinaNet, which:

- uses EfficientNet as a backbone;
- employs compound scaling;
- uses a newly proposed BiFPN, "efficient bidirectional cross-scale connections and weighted feature fusion".

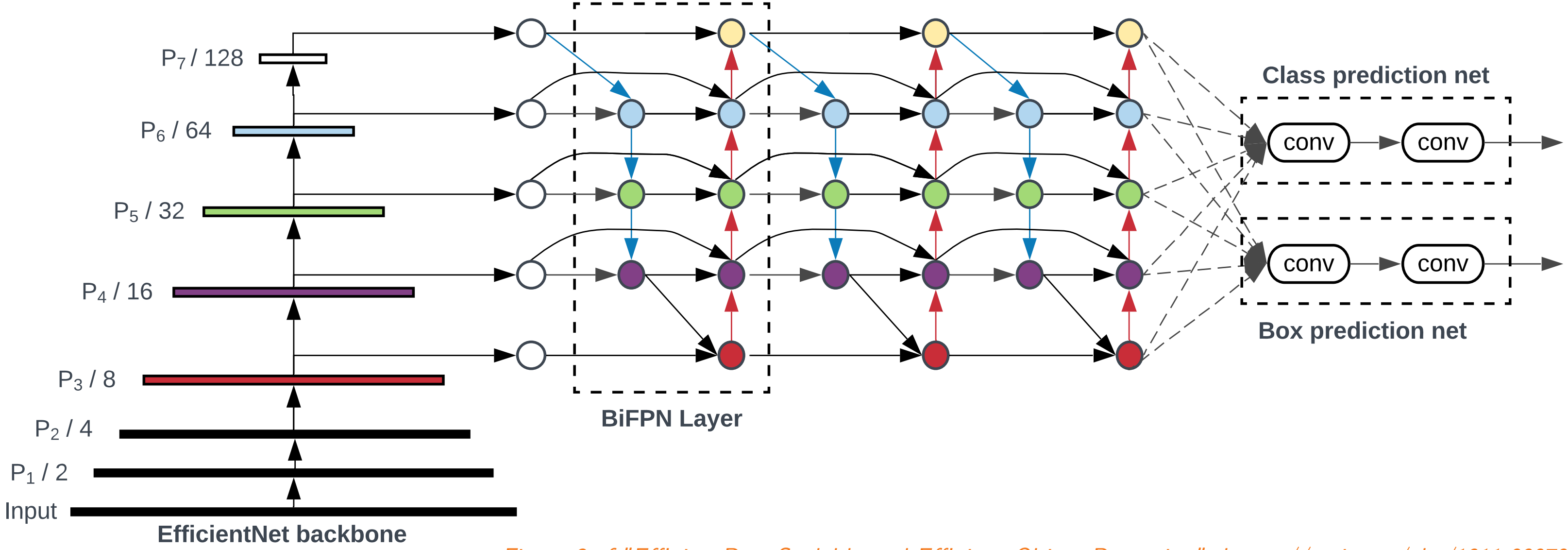


Figure 3 of "EfficientDet: Scalable and Efficient Object Detection", https://arxiv.org/abs/1911.09070

EfficientDet – BiFPN

In multi-scale fusion in FPN, information flows only from the pyramid levels with smaller resolution to the levels with higher resolution.

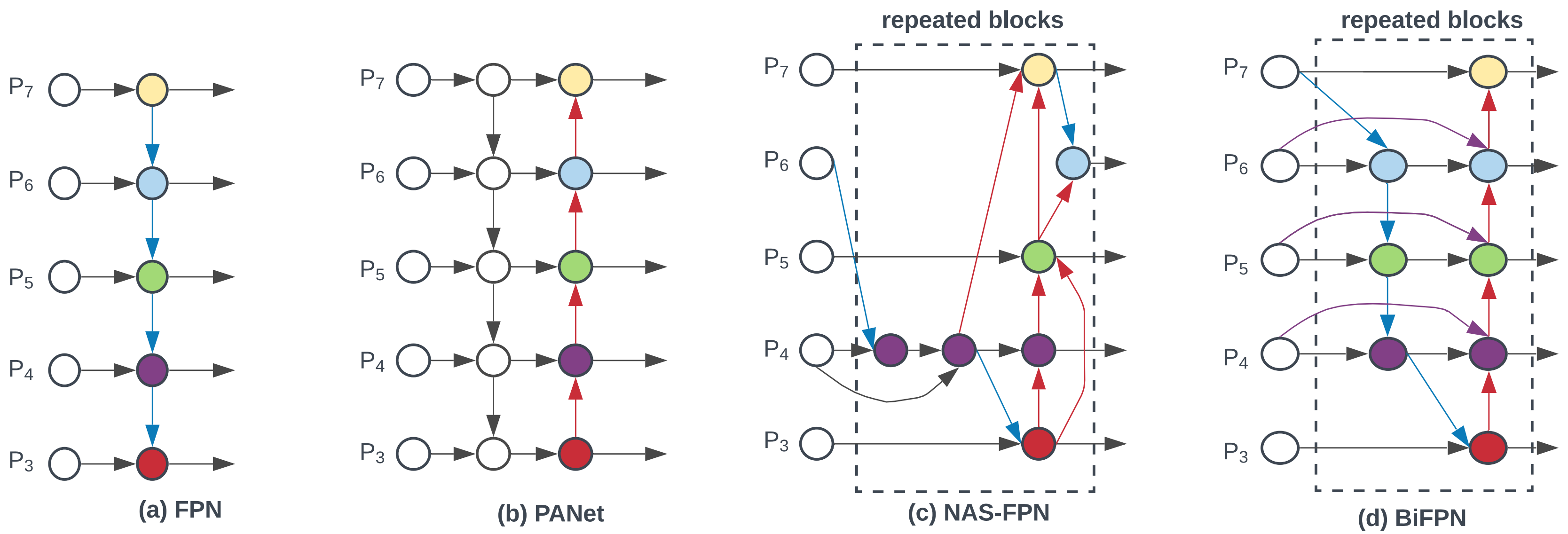


Figure 2 of "EfficientDet: Scalable and Efficient Object Detection", https://arxiv.org/abs/1911.09070

BiFPN consists of several rounds of bidirectional flows. Each bidirectional flow employs residual connections and does not include nodes that have only one input edge with no feature fusion. All operations are 3×3 separable convolutions with batch normalization and ReLU, upsampling is done by repeating rows and columns and downsampling by max-pooling.

EfficientDet – Weighted BiFPN

When combining features with different resolutions, it is common to resize them to the same resolution and sum them – therefore, all set of features are considered to be of the same importance. The authors however argue that features from different resolution contribute to the final result *unequally* and propose to combine them with trainable weighs.

- **Softmax-based fusion**: In each BiFPN node, we create a trainable weight w_i for every input I_i and the final combination (after resize, before a convolution) is

$$\sum_i \frac{e^{w_i}}{\sum_j e^{w_j}} \mathsf{I}_i.$$

- **Fast normalized fusion**: Authors propose a simpler alternative of weighting:

$$\sum_i \frac{\mathrm{ReLU}(w_i)}{\varepsilon + \sum_j \mathrm{ReLU}(w_j)} \mathsf{I}_i.$$

  It uses $\varepsilon = 0.0001$ for stability and is up to 30% faster on a GPU.

Similar to EfficientNet, authors propose to scale various dimensions of the network, using a single compound coefficient ϕ.

After performing a grid search:

- the width of BiFPN is scaled as $W_{BiFPN} = 64 \cdot 1.35^\phi$,
- the depth of BiFPN is scaled as $D_{BiFPN} = 3 + \phi$,
- the box/class predictor has the same width as BiFPN and depth $D_{class} = 3 + \lfloor \phi/3 \rfloor$,
- input image resolution increases according to $R_{image} = 512 + 128 \cdot \phi$.

	Input size R_{input}	Backbone Network	BiFPN #channels W_{bifpn}	BiFPN #layers D_{bifpn}	Box/class #layers D_{class}
D0 ($\phi = 0$)	512	B0	64	3	3
D1 ($\phi = 1$)	640	B1	88	4	3
D2 ($\phi = 2$)	768	B2	112	5	3
D3 ($\phi = 3$)	896	B3	160	6	4
D4 ($\phi = 4$)	1024	B4	224	7	4
D5 ($\phi = 5$)	1280	B5	288	7	4
D6 ($\phi = 6$)	1280	B6	384	8	5
D6 ($\phi = 7$)	1536	B6	384	8	5

Table 1 of "EfficientDet: Scalable and Efficient Object Detection", https://arxiv.org/abs/1911.09070

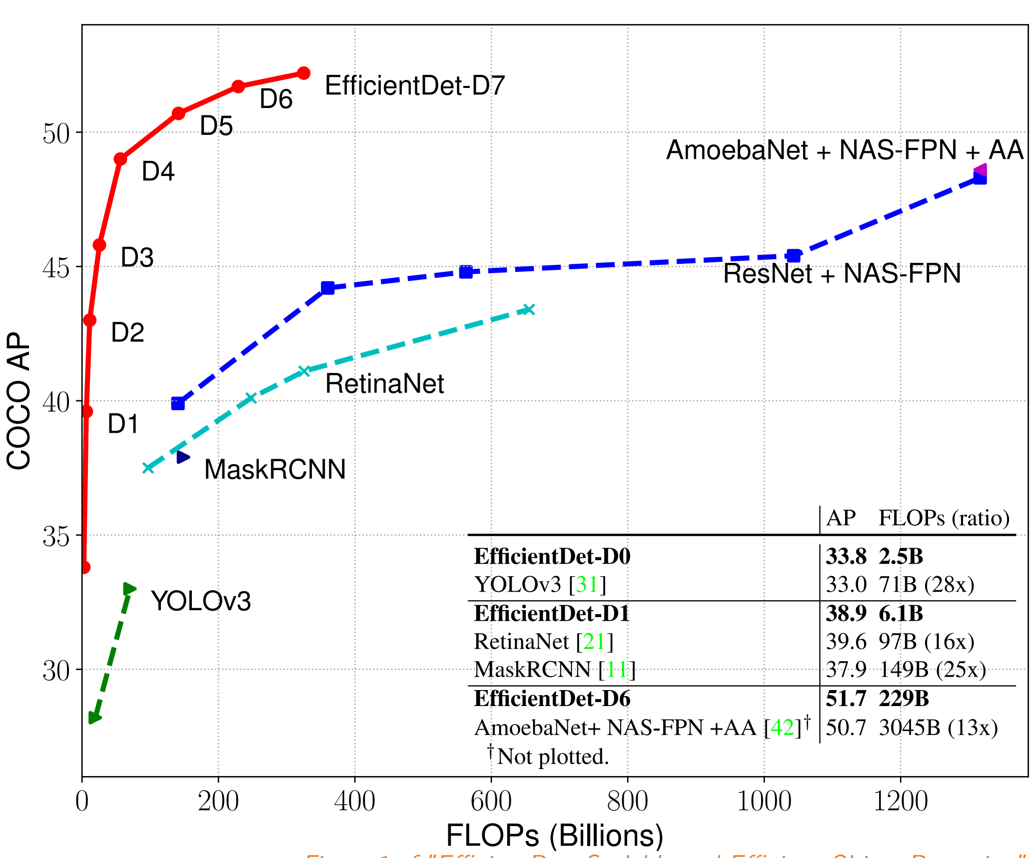


Figure 1 of "EfficientDet: Scalable and Efficient Object Detection", https://arxiv.org/abs/1911.09070

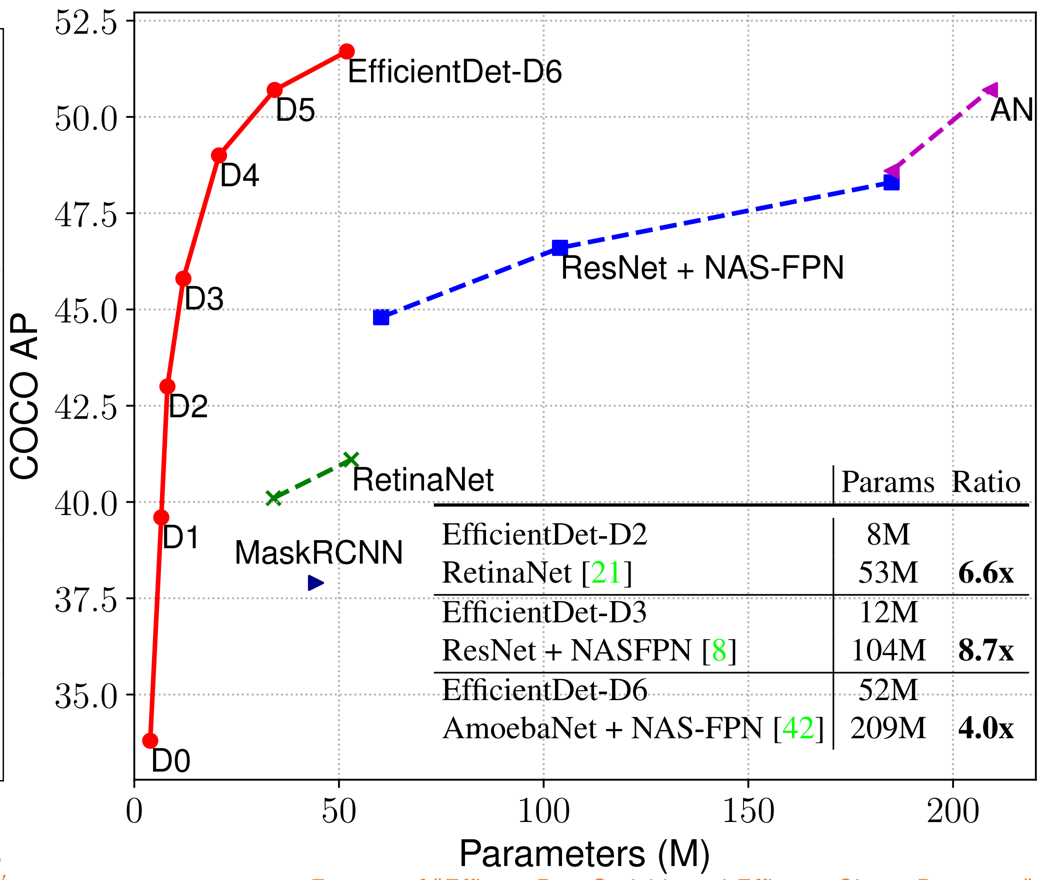


Figure 4 of "EfficientDet: Scalable and Efficient Object Detection", https://arxiv.org/abs/1911.09070

EfficientDet – Results

Model	tet-dev AP	tet-dev AP_{50}	tet-dev AP_{75}	val AP	Params	Ratio	FLOPs	Ratio	Latency GPU_{ms}	Latency CPU_s
EfficientDet-D0 (512)	**33.8**	**52.2**	**35.8**	**33.5**	**3.9M**	**1x**	**2.5B**	**1x**	**16**	**0.32**
YOLOv3 [31]	33.0	57.9	34.4	-	-	-	71B	28x	51†	-
EfficientDet-D1 (640)	**39.6**	**58.6**	**42.3**	**39.1**	**6.6M**	**1x**	**6.1B**	**1x**	**20**	**0.74**
RetinaNet-R50 (640) [21]	37.0	-	-	-	34M	6.7x	97B	16x	27	2.8
RetinaNet-R101 (640)[21]	37.9	-	-	-	53M	8.0x	127B	21x	34	3.6
EfficientDet-D2 (768)	**43.0**	**62.3**	**46.2**	**42.5**	**8.1M**	**1x**	**11B**	**1x**	**24**	**1.2**
RetinaNet-R50 (1024) [21]	40.1	-	-	-	34M	4.3x	248B	23x	51	7.5
RetinaNet-R101 (1024) [21]	41.1	-	-	-	53M	6.6x	326B	30x	65	9.7
ResNet-50 + NAS-FPN (640) [8]	39.9	-	-	-	60M	7.5x	141B	13x	41	4.1
EfficientDet-D3 (896)	**45.8**	**65.0**	**49.3**	**45.9**	**12M**	**1x**	**25B**	**1x**	**42**	**2.5**
ResNet-50 + NAS-FPN (1024) [8]	44.2	-	-	-	60M	5.1x	360B	15x	79	11
ResNet-50 + NAS-FPN (1280) [8]	44.8	-	-	-	60M	5.1x	563B	23x	119	17
ResNet-50 + NAS-FPN (1280@384)[8]	45.4	-	-	-	104M	8.7x	1043B	42x	173	27
EfficientDet-D4 (1024)	**49.4**	**69.0**	**53.4**	**49.0**	**21M**	**1x**	**55B**	**1x**	**74**	**4.8**
AmoebaNet+ NAS-FPN +AA(1280)[42]	-	-	-	48.6	185M	8.8x	1317B	24x	259	38
EfficientDet-D5 (1280)	**50.7**	**70.2**	**54.7**	**50.5**	**34M**	**1x**	**135B**	**1x**	**141**	**11**
EfficientDet-D6 (1280)	**51.7**	**71.2**	**56.0**	**51.3**	**52M**	**1x**	**226B**	**1x**	**190**	**16**
AmoebaNet+ NAS-FPN +AA(1536)[42]	-	-	-	50.7	209M	4.0x	3045B	13x	608	83
EfficientDet-D7 (1536)	**52.2**	**71.4**	**56.3**	**51.8**	**52M**	**1x**	**325B**	**1x**	**262**	**24**

We omit ensemble and test-time multi-scale results [27, 10].

†Latency marked with † are from papers, and others are measured on the same machine with Titan V GPU.

Table 2 of "EfficientDet: Scalable and Efficient Object Detection", https://arxiv.org/abs/1911.09070

EfficientDet – Inference Latencies

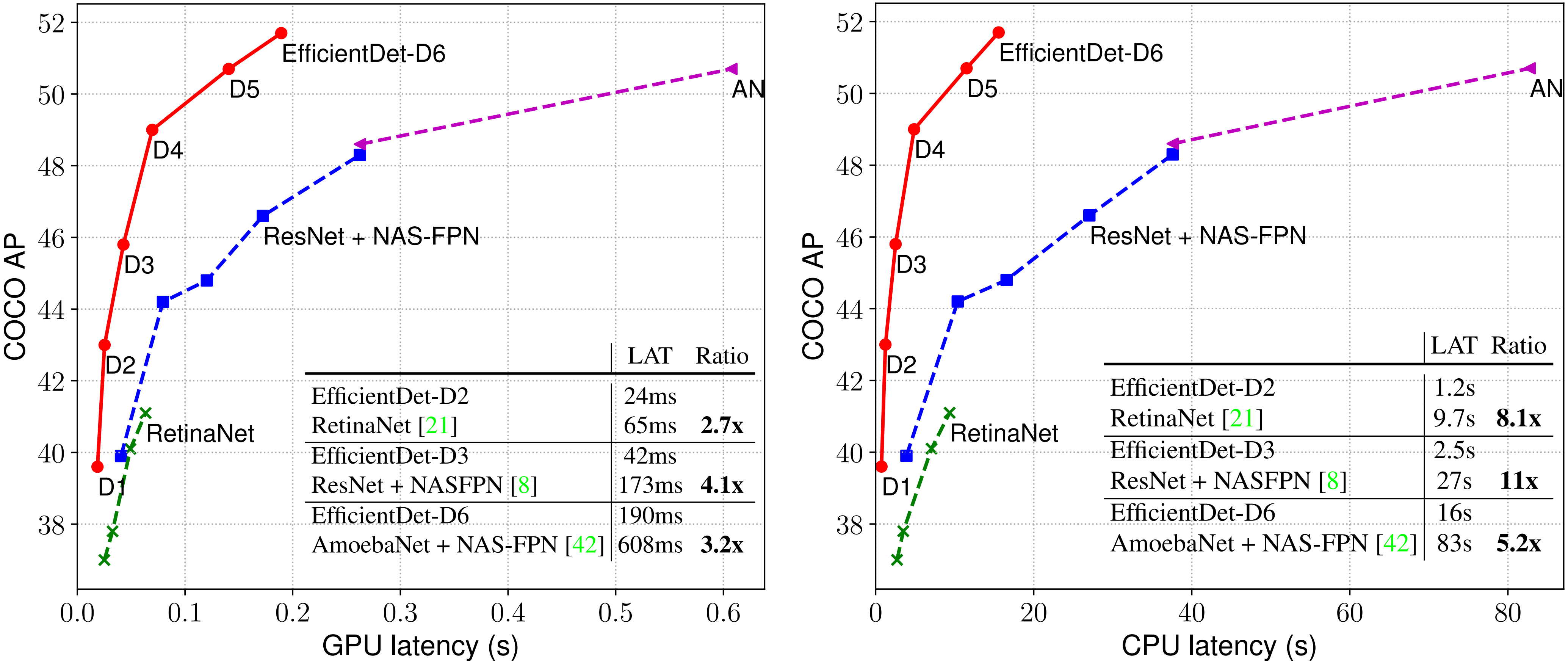


Figure 4 of "EfficientDet: Scalable and Efficient Object Detection", https://arxiv.org/abs/1911.09070

Given that EfficientDet employs both a powerful backbone and new BiFPN, authors quantify the improvement of the individual components.

	AP	Parameters	FLOPs
ResNet50 + FPN	37.0	34M	97B
EfficientNet-B3 + FPN	40.3	21M	75B
EfficientNet-B3 + **BiFPN**	44.4	12M	24B

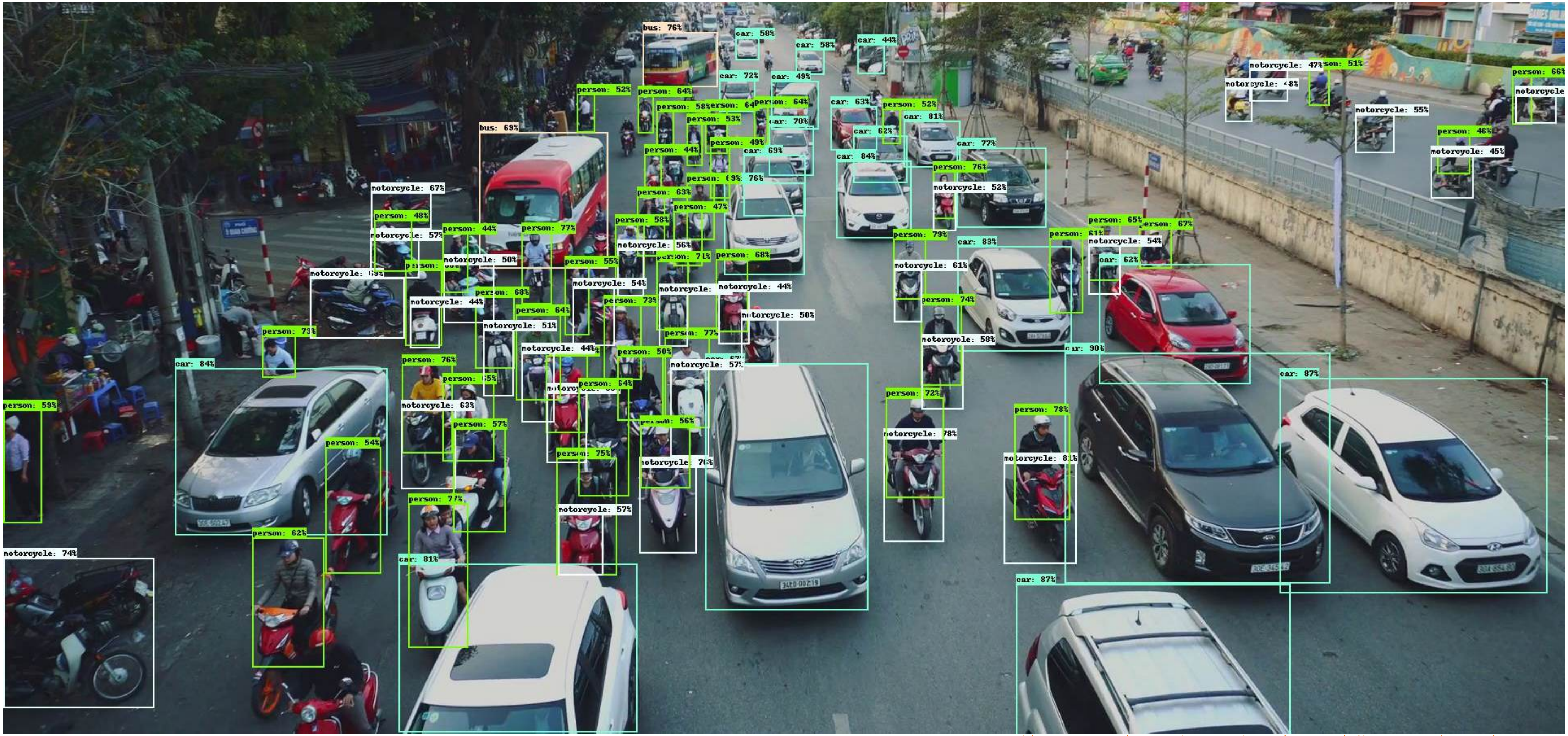
Table 4 of "EfficientDet: Scalable and Efficient Object Detection", https://arxiv.org/abs/1911.09070

The comparison with previously used cross-scale fusion architectures is also provided:

	AP	#Params ratio	#FLOPs ratio
Repeated top-down FPN	42.29	1.0x	1.0x
Repeated FPN+PANet	44.08	1.0x	1.0x
NAS-FPN	43.16	0.71x	0.72x
Fully-Connected FPN	43.06	1.24x	1.21x
BiFPN (w/o weighted)	**43.94**	**0.88x**	**0.67x**
BiFPN (w/ weighted)	**44.39**	**0.88x**	**0.68x**

Table 5 of "EfficientDet: Scalable and Efficient Object Detection", https://arxiv.org/abs/1911.09070

EfficientDet-D0 Example



https://github.com/google/automl/blob/master/efficientdet/g3doc/street.jpg

Batch Normalization

Neuron value is normalized across the minibatch, and in case of CNN also across all positions.

Layer Normalization

Neuron value is normalized across the layer.

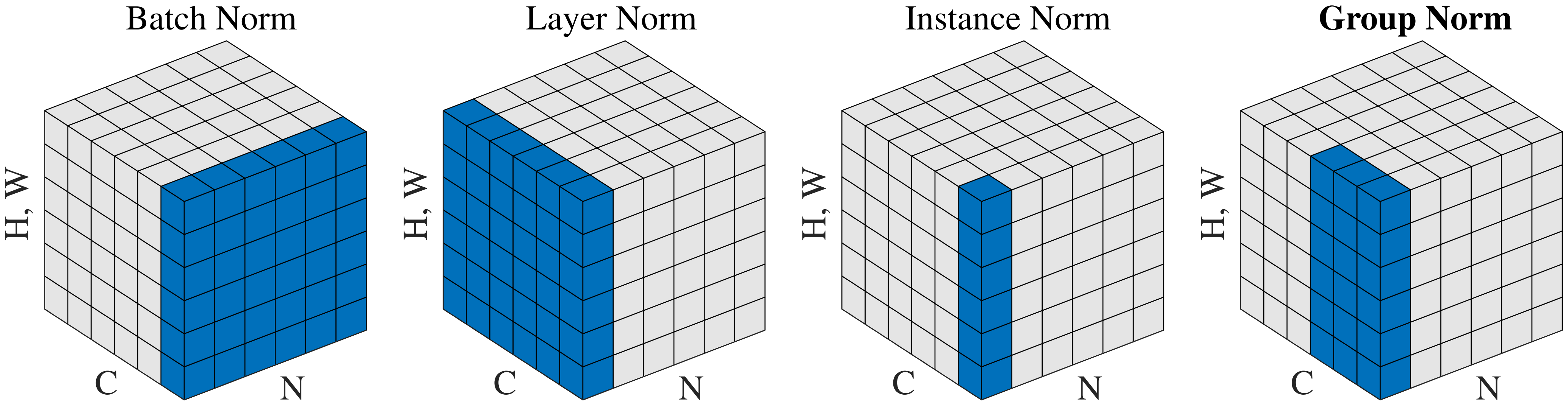


Figure 2 of "Group Normalization", https://arxiv.org/abs/1803.08494

Group Normalization is analogous to Layer normalization, but the channels are normalized in groups (by default, $G = 32$).

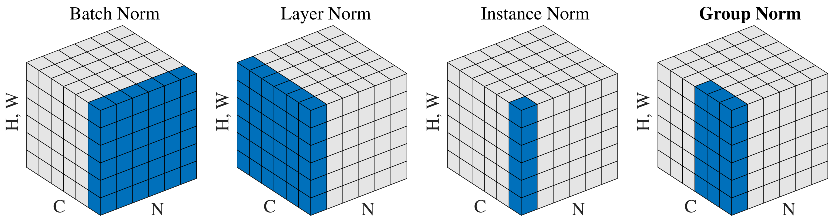


Figure 2 of "Group Normalization", https://arxiv.org/abs/1803.08494

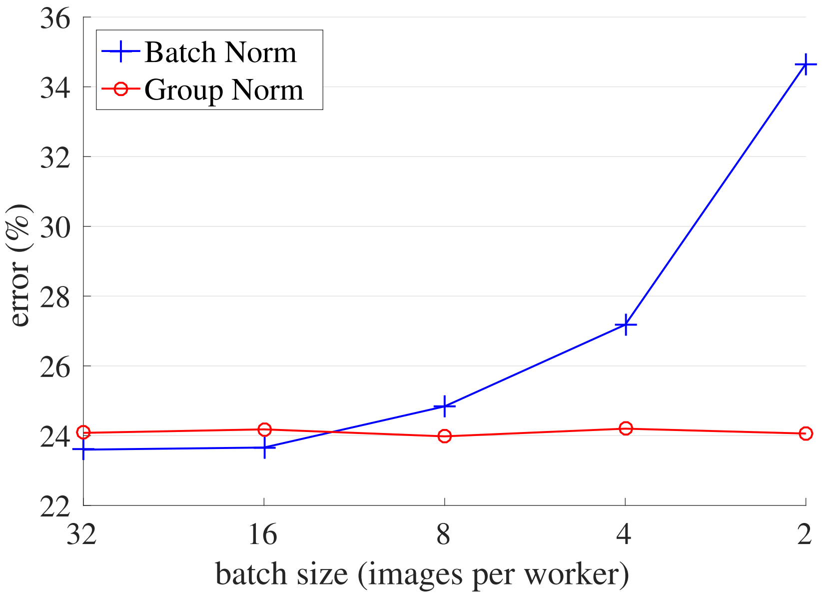


Figure 1 of "Group Normalization", https://arxiv.org/abs/1803.08494

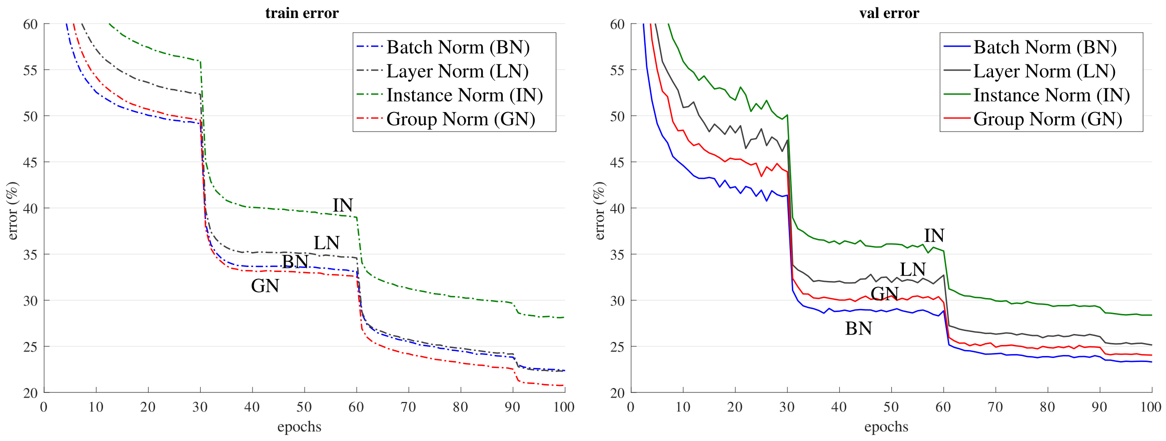


Figure 4. Comparison of error curves with a batch size of 32 images/GPU. We show the ImageNet training error (left) and validation error (right) vs. numbers of training epochs. The model is ResNet-50.

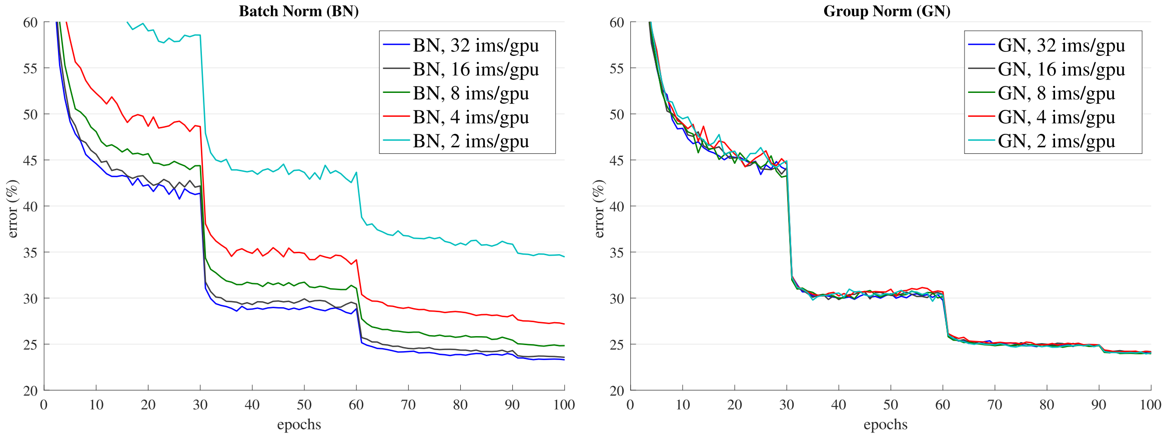


Figure 5. Sensitivity to batch sizes: ResNet-50's validation error of BN (left) and GN (right) trained with 32, 16, 8, 4, and 2 images/GPU.

Figures 4 and 5 of "Group Normalization", https://arxiv.org/abs/1803.08494

backbone	AP^{bbox}	AP^{bbox}_{50}	AP^{bbox}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
BN^*	37.7	57.9	40.9	32.8	54.3	34.7
GN	**38.8**	**59.2**	**42.2**	**33.6**	**55.9**	**35.4**

Table 4. **Detection and segmentation results in COCO**, using Mask R-CNN with **ResNet-50 C4**. BN^* means BN is frozen.

backbone	box head	AP^{bbox}	AP^{bbox}_{50}	AP^{bbox}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
BN^*	-	38.6	59.5	41.9	34.2	56.2	36.1
BN^*	GN	39.5	60.0	43.2	34.4	56.4	**36.3**
GN	GN	**40.0**	**61.0**	**43.3**	**34.8**	**57.3**	**36.3**

Table 5. **Detection and segmentation results in COCO**, using Mask R-CNN with **ResNet-50 FPN** and a 4conv1fc bounding box head. BN^* means BN is frozen.

Tables 4 and 5 of "Group Normalization", https://arxiv.org/abs/1803.08494