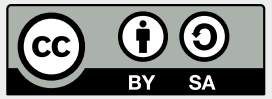
Convolutional Neural Networks III, Recurrent Neural Networks

Milan Straka

April 08, 2019

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



Fast R-CNN

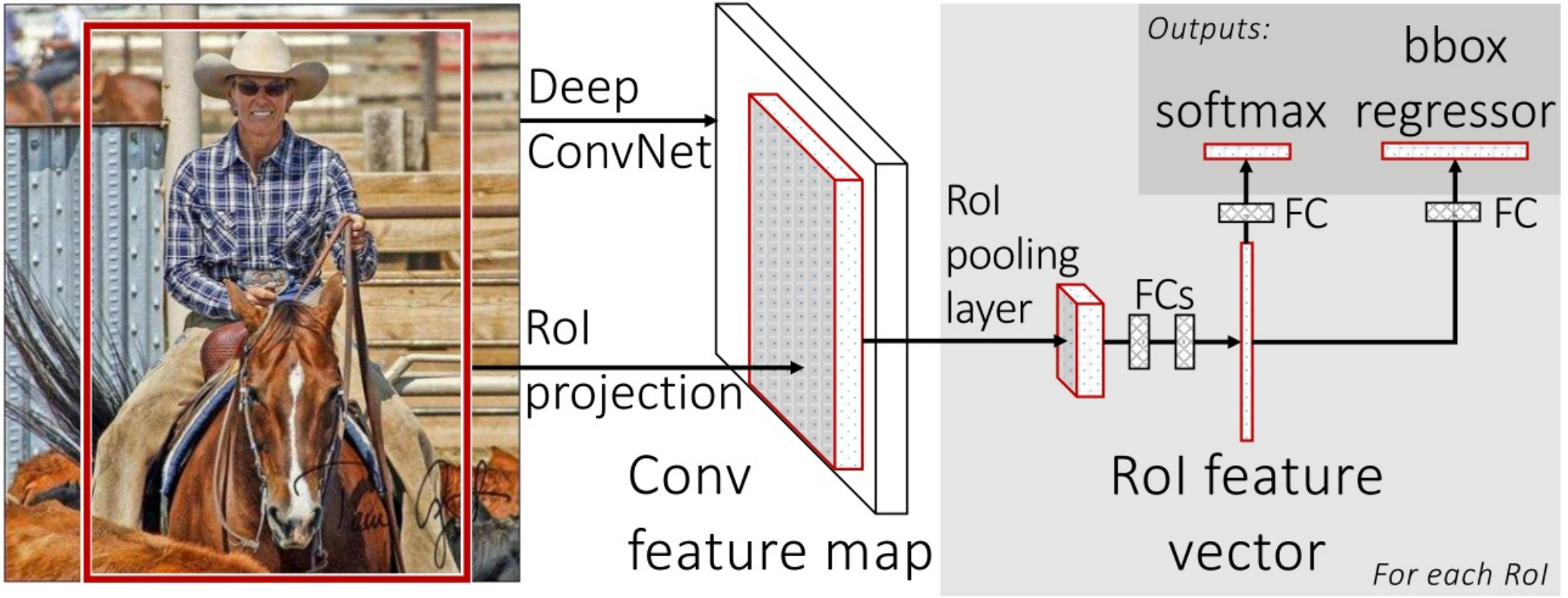


Figure 1 of paper "Fast R-CNN", https://arxiv.org/abs/1504.08083.

The 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, and let x, y, w, h be parameters of the bounding box. We represent them as follows:

$$t_x = (x - x_r)/w_r, \quad t_y = (y - y_r)/h_r$$
$$t_w = \log(w/w_r), \quad t_h = \log(h/h_r)$$

Usually a smooth_{L_1} loss, or *Huber loss*, is employed for bounding box parameters

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

The complete loss is then

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

Intersection over union

For two bounding boxes (or two masks) the *intersection over union* (*IoU*) is a ration 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 $[0.1, 0.5)$.

Choosing RoIs during inference

Single object can be found in multiple RoIs. To choose the most salient one, we perform *non-maximum suppression* -- we ignore RoIs which have an overlap with a higher scoring RoI of the same type, where the IoU is larger than a given threshold (usually, 0.3 is used). Higher scoring RoI is the one with higher probability from the classification head.

Average Precision

Evaluation is performed using *Average Precision* (*AP*).

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 0.5 with any ground-truth box. We define *AP* as an average of precisions for recall levels $0, 0.1, 0.2, \ldots, 1$.

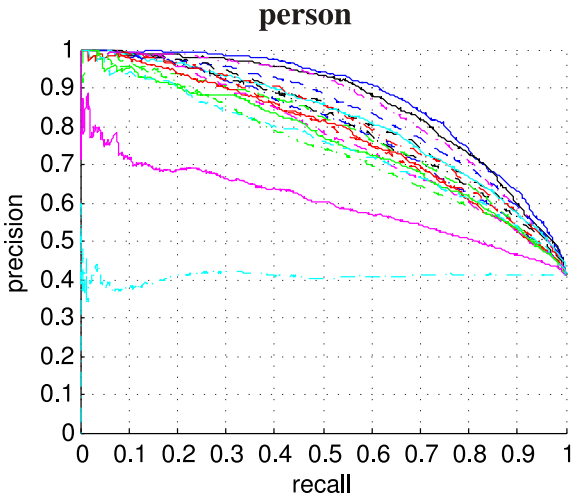


Figure 6 of paper "The PASCAL Visual Object Classes (VOC) Challenge", http://homepages.inf.ed.ac.uk/ckiw/postscript/ijcv_voc09.pdf.

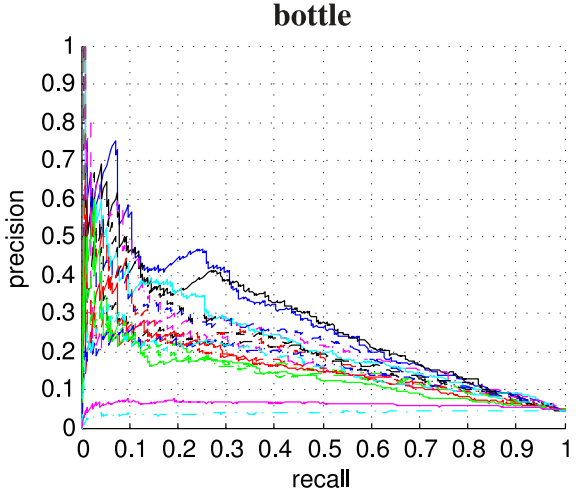


Figure 6 of paper "The PASCAL Visual Object Classes (VOC) Challenge", http://homepages.inf.ed.ac.uk/ckiw/postscript/ijcv_voc09.pdf.

Faster R-CNN

For Fast R-CNN, the most time consuming part is generating the RoIs.

Therefore, Faster R-CNN jointly generates *regions of interest* using a *region proposal network* and performs object detection.

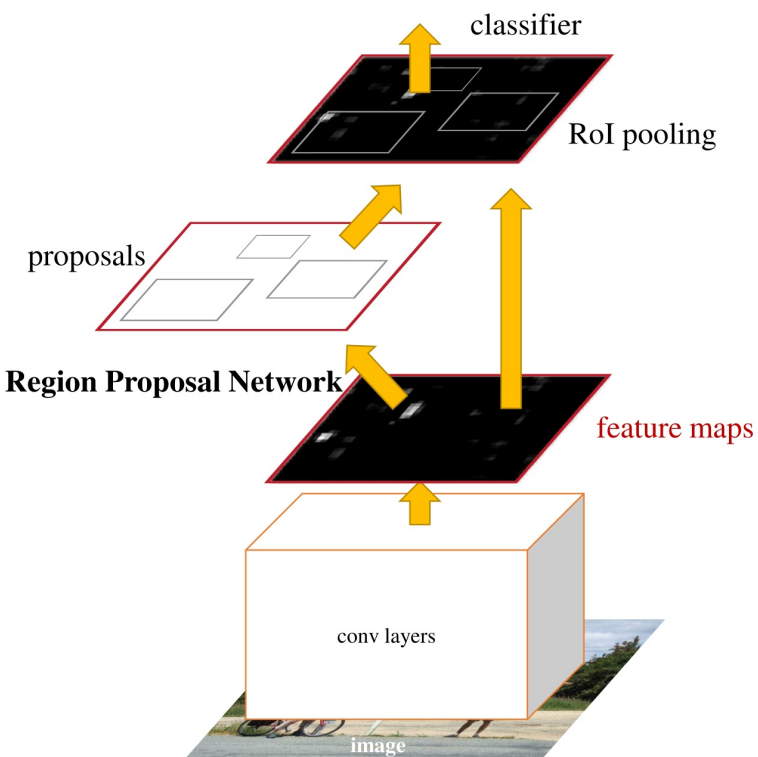


Figure 2 of paper "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", https://arxiv.org/abs/1506.01497

Mask R-CNN

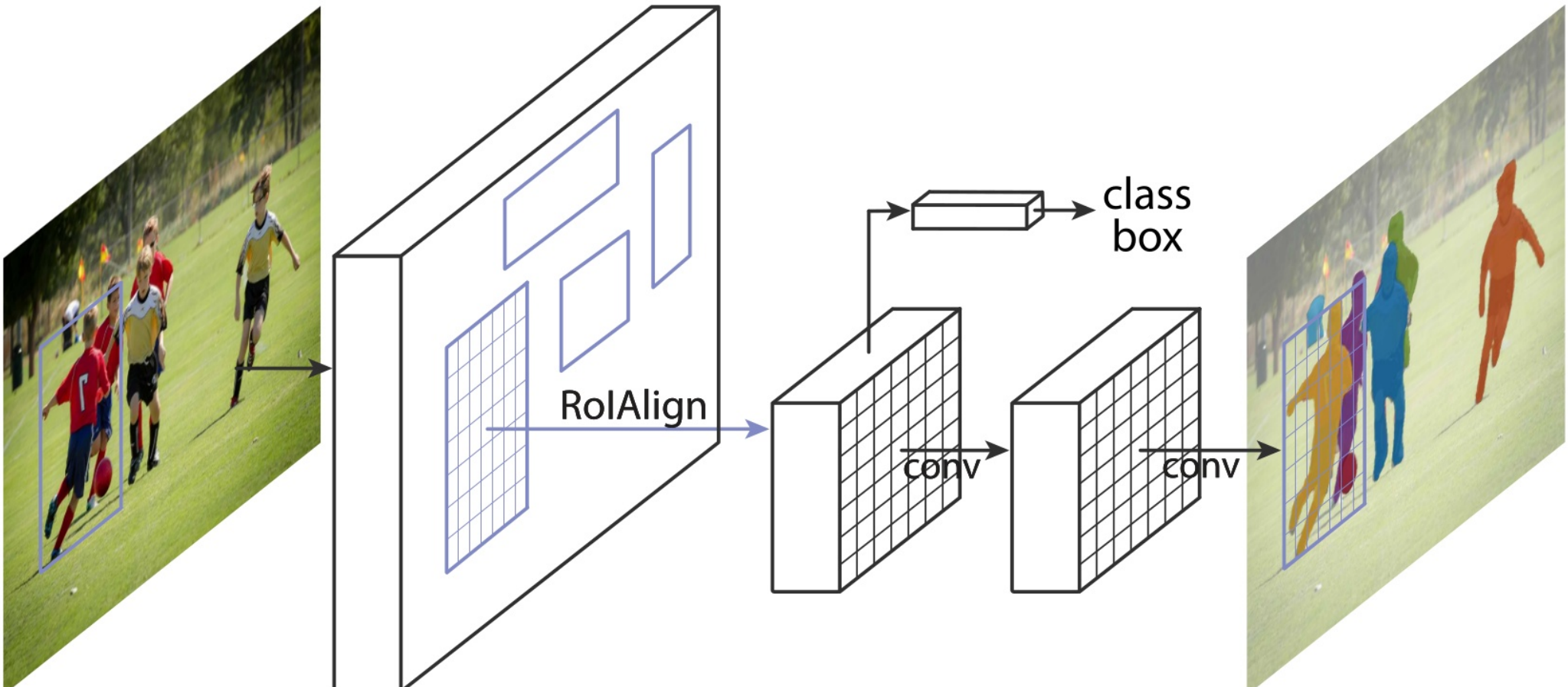


Figure 1 of paper "Mask R-CNN", https://arxiv.org/abs/1703.06870.

RoIAlign

More precise alignment is required for the RoI in order to predict the masks. Therefore, instead of max-pooling used in the RoI pooling, RoIAlign with bilinear interpolation is used.

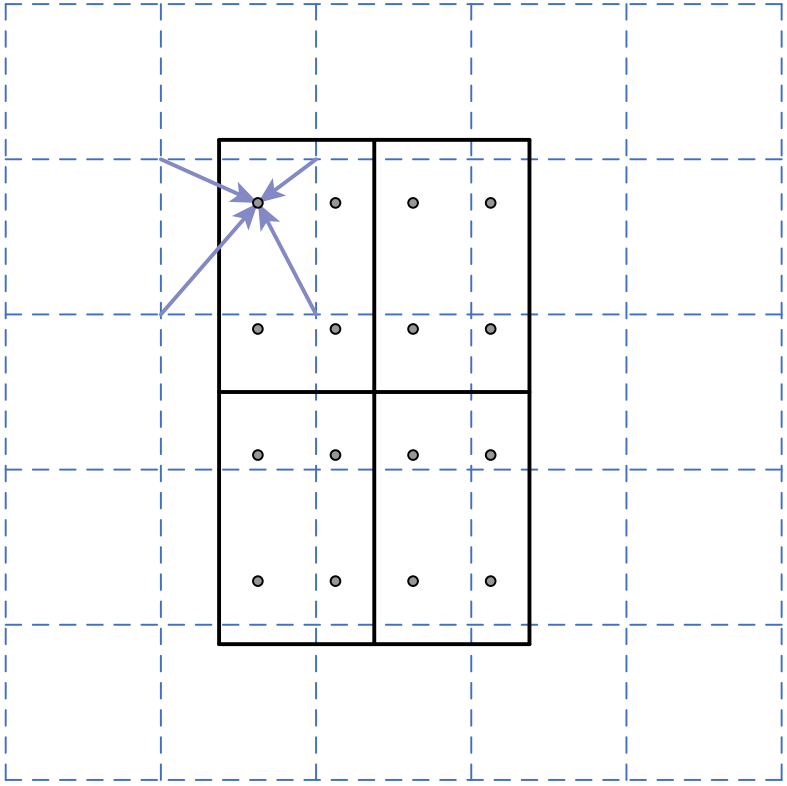


Figure 3 of paper "Mask R-CNN", https://arxiv.org/abs/1703.06870.

Mask R-CNN

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

- Usually higher resolution is needed (14×14 instead of 7×7).
- The masks are predicted for each class separately.
- The masks are predicted using convolutions instead of fully connected layers.

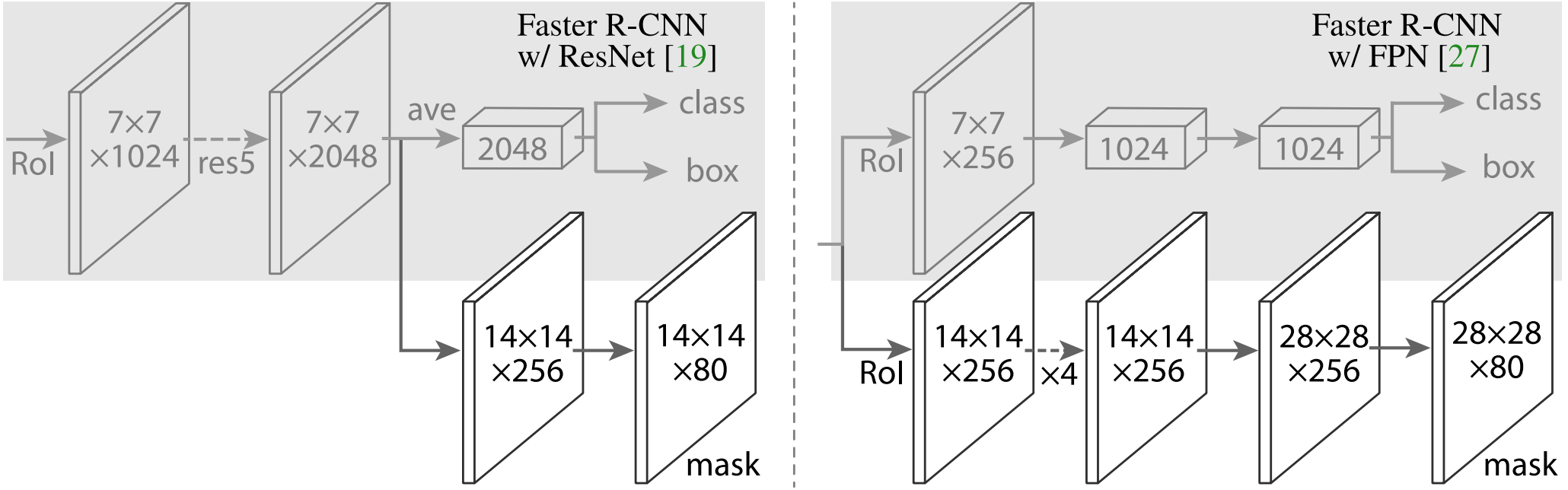


Figure 4 of paper "Mask R-CNN", https://arxiv.org/abs/1703.06870.

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: $1024 \to 1024 \to 80 \cdot 28^2$	31.5	53.7	32.8
MLP	fc: $1024 \to 1024 \to 1024 \to 80 \cdot 28^2$	31.5	54.0	32.6
FCN	conv: $256 \to 256 \to 256 \to 256 \to 256 \to 80$	**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 paper "Mask R-CNN", https://arxiv.org/abs/1703.06870.

Mask R-CNN – Human Pose Estimation

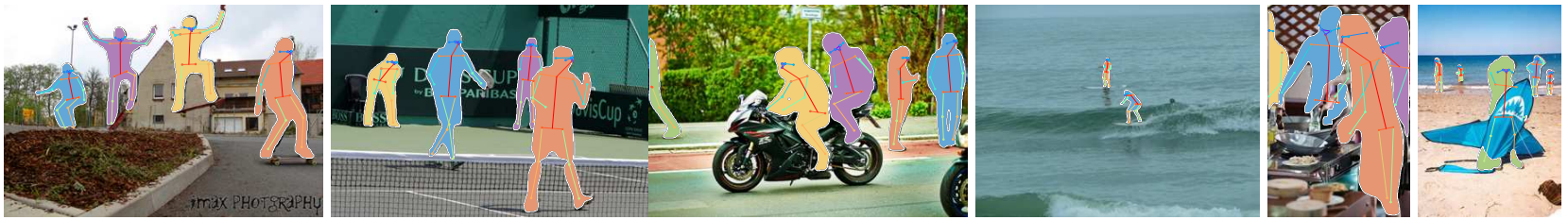


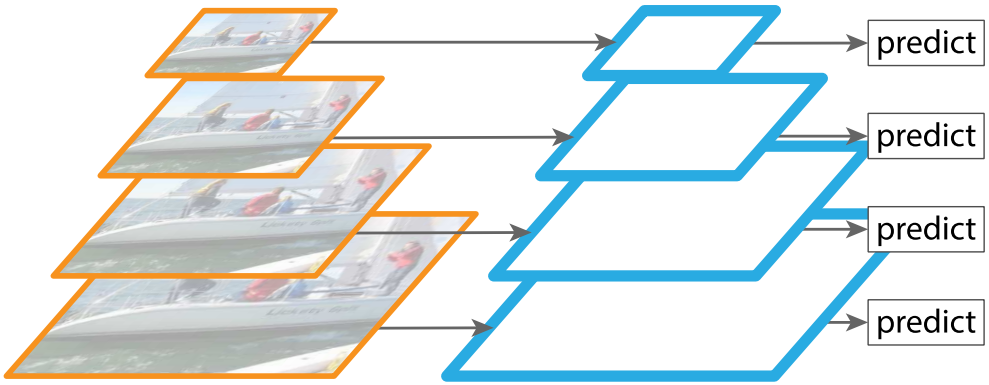
Figure 7 of paper "Mask R-CNN", https://arxiv.org/abs/1703.06870.

- 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 $\mathrm{softmax}$ output function.

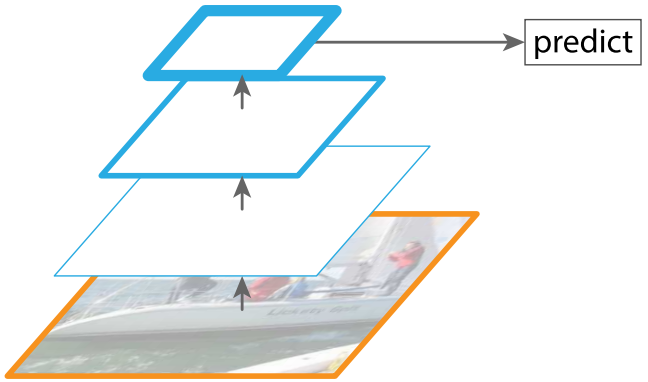
	AP^{kp}	AP^{kp}_{50}	AP^{kp}_{75}	AP^{kp}_{M}	AP^{kp}_{L}
CMU-Pose+++ [6]	61.8	84.9	67.5	57.1	68.2
G-RMI [32]†	62.4	84.0	68.5	**59.1**	68.1
Mask R-CNN, keypoint-only	62.7	87.0	68.4	57.4	71.1
Mask R-CNN, keypoint & mask	**63.1**	**87.3**	**68.7**	57.8	**71.4**

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

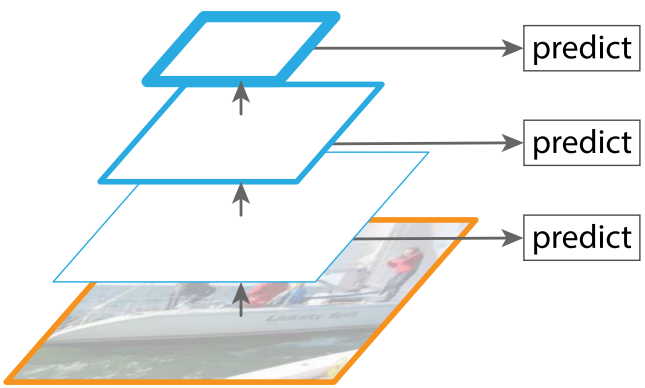
Feature Pyramid Networks



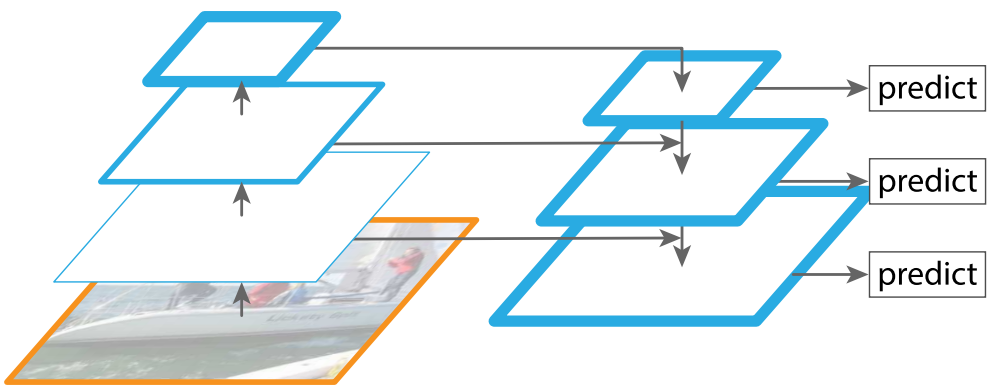
(a) Featurized image pyramid



(b) Single feature map



(c) Pyramidal feature hierarchy



(d) Feature Pyramid Network

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

Feature Pyramid Networks

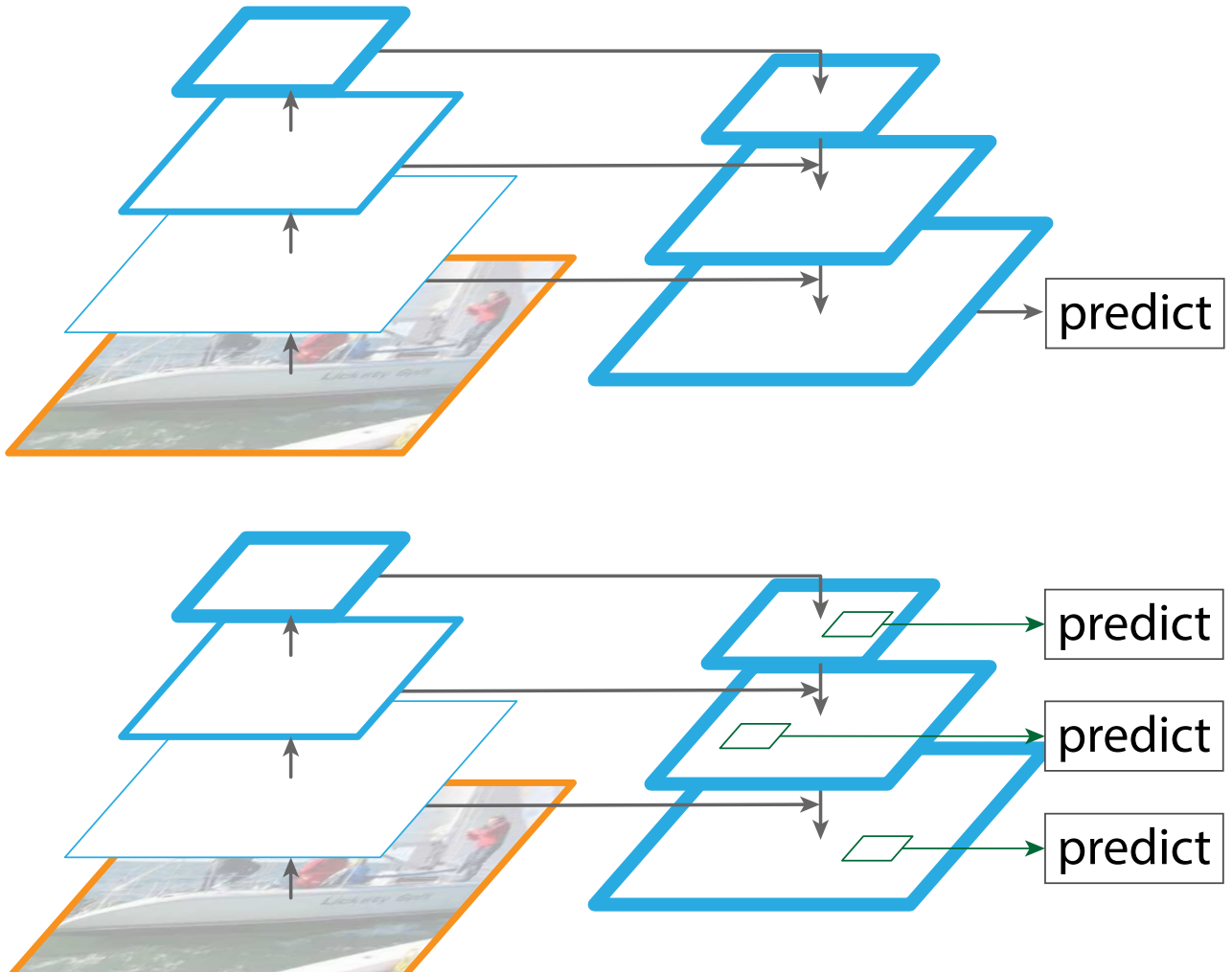


Figure 2 of paper "Feature Pyramid Networks for Object Detection", https://arxiv.org/abs/1612.03144.

Feature Pyramid Networks

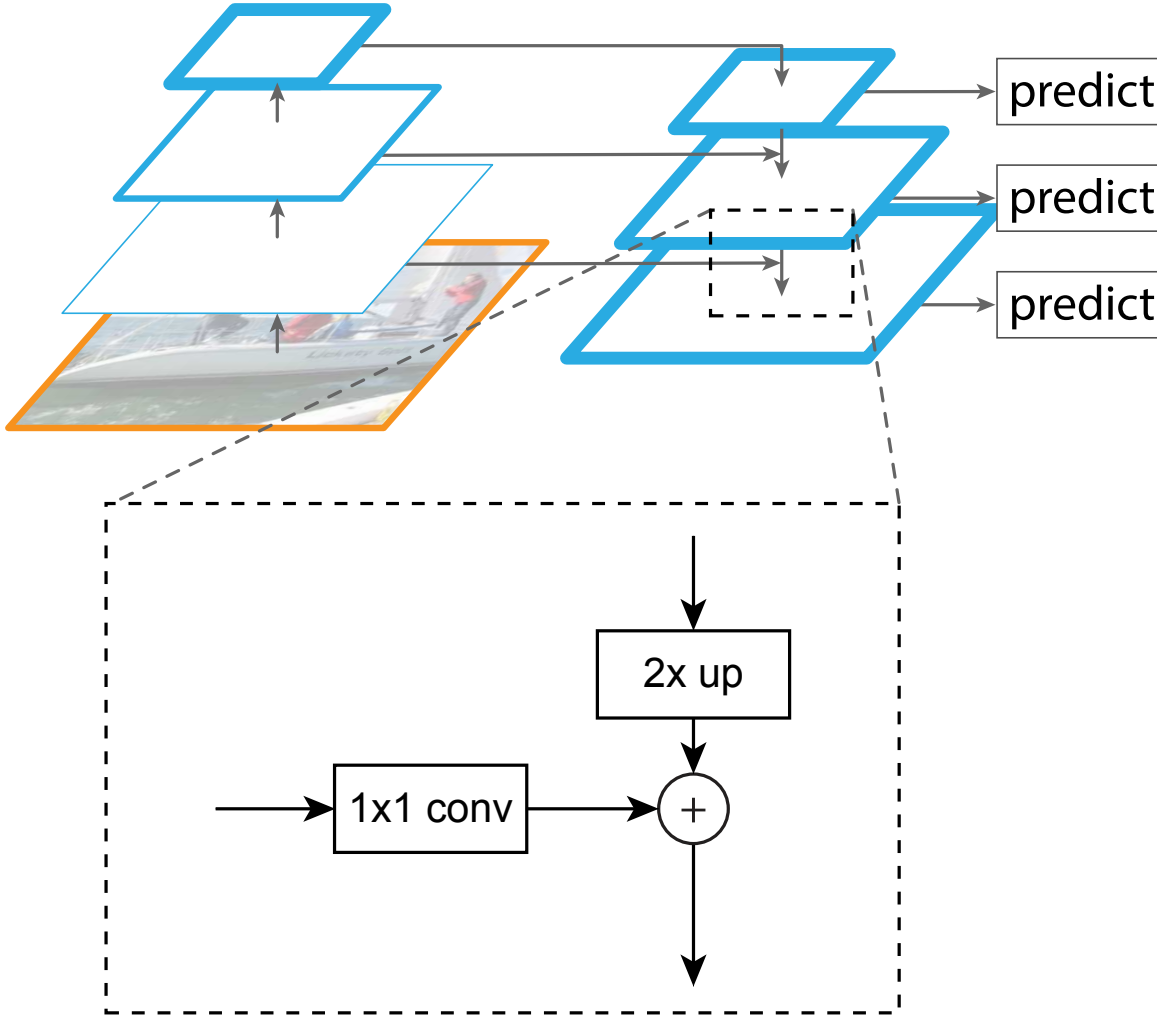


Figure 3 of paper "Feature Pyramid Networks for Object Detection", https://arxiv.org/abs/1612.03144.

Feature Pyramid Networks

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

Focal Loss

For single-stage object detection architectures, *class imbalance* has been identified as the main issue preventing to obtain 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.

Cross-entropy loss is computed as

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

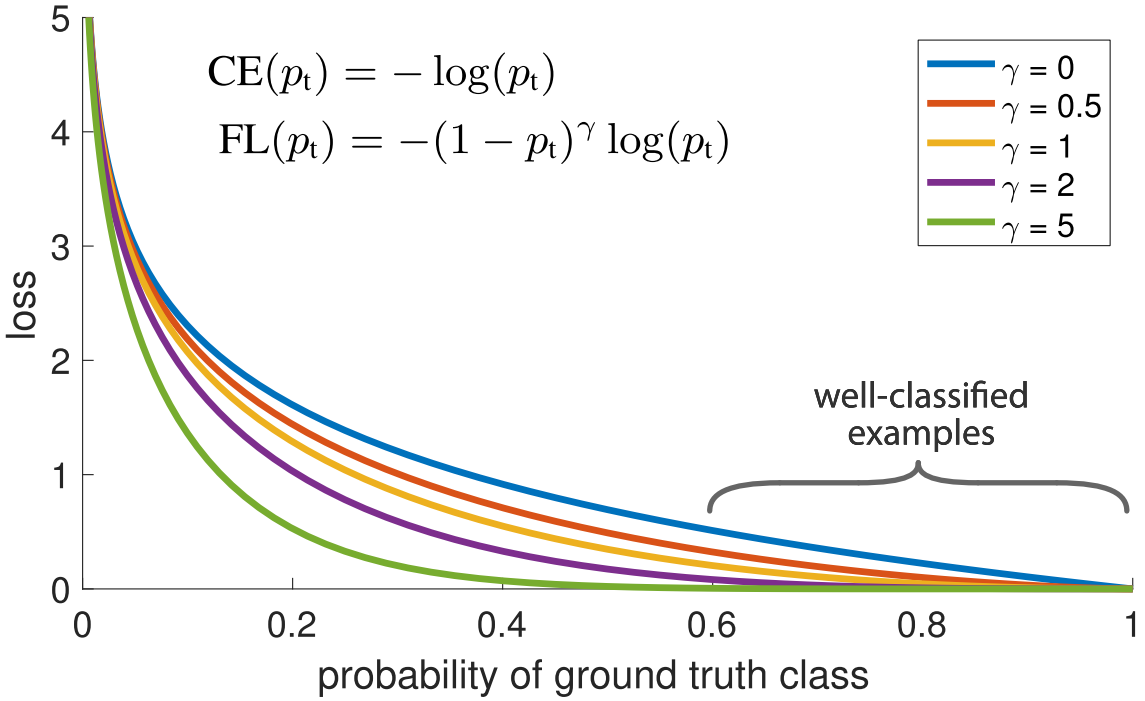


Figure 1 of paper "Focal Loss for Dense Object Detection", https://arxiv.org/abs/1708.02002.

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

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

Focal Loss

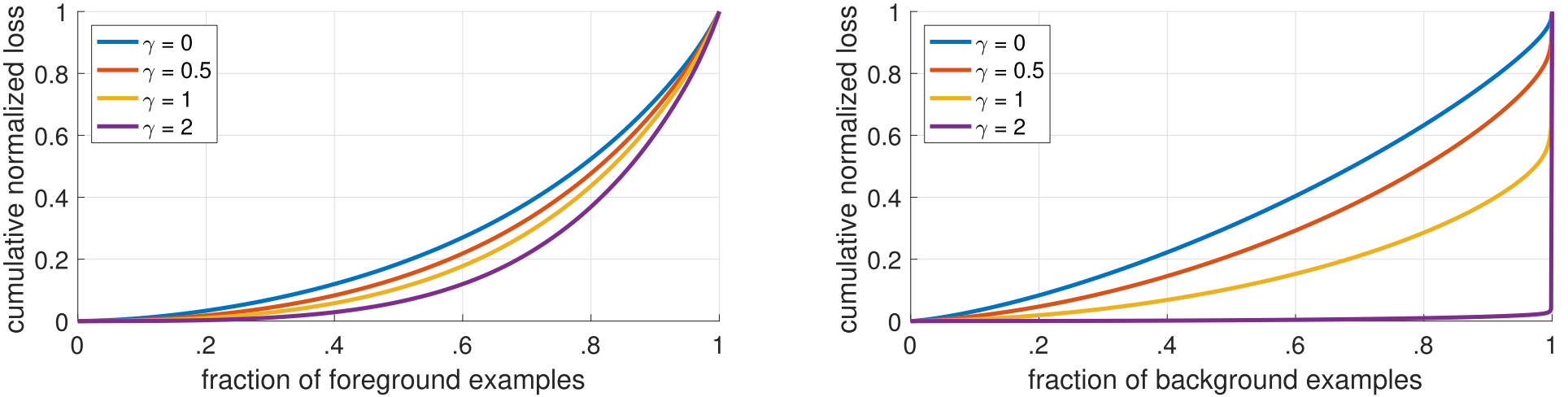


Figure 4. Cumulative distribution functions of the normalized loss for positive and negative samples for different values of γ for a *converged* model. The effect of changing γ on the distribution of the loss for positive examples is minor. For negatives, however, increasing γ heavily concentrates the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

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

RetinaNet

RetineNet 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 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 $(\{2, 2^{1/3}, 2^{2/3}\} \cdot 4 \cdot 2^l)$. The classification and boundary regression heads do not share parameters and are fully convolutional, generating $anchors \cdot classes$ sigmoids and $anchors$ bounding boxes per position.

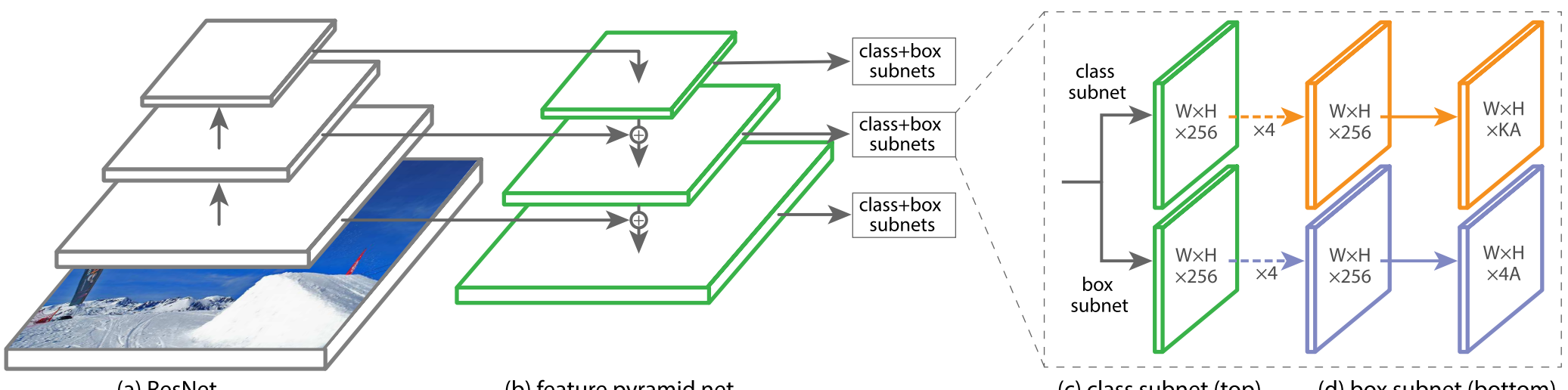


Figure 3 of paper "Focal Loss for Dense Object Detection", https://arxiv.org/abs/1708.02002.

RetinaNet

During training we assign anchors 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 are ignored during training). The classification head is trained using focal loss with $\gamma = 2$ (but according to the paper, all values in $[0.5, 5]$ range works well); the boundary regression head is trained using smooth_{L_1} loss as in Fast(er) R-CNN.

During inference, we consider at most 1000 objects with at least 0.05 probability from every pyramid level, merging the top predictions from all levels using non-maximum suppression with 0.5 threshold.

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

Transfer Learning

In many situations, we would like to utilize a model trained on a different dataset – generally, this cross-dataset usage is called *transfer learning*.

In image processing, models trained on ImageNet are frequently used as general **feature extraction models**.

The easiest scenario is to take a ImageNet model, drop the last classification layer, and use the result of the global average pooling as image features. The ImageNet model is not modified during training.

For efficiency, we may precompute the image features **once** and reuse it later many times.

Transfer Learning – Finetuning

After we have successfully trained a network employing an ImageNet model, we may improve performance further by *finetuning* – training the full network including the ImageNet model, allowing the feature extraction to adapt to the current dataset.

- The laters after the ImageNet models **must** be already trained to convergence.
- Usually a smaller learning rate is necessary (for example one tenth of the original one, i.e., 0.0001 for Adam).
- We have to think about batch normalization, data augmentation or other regularization techniques.

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.

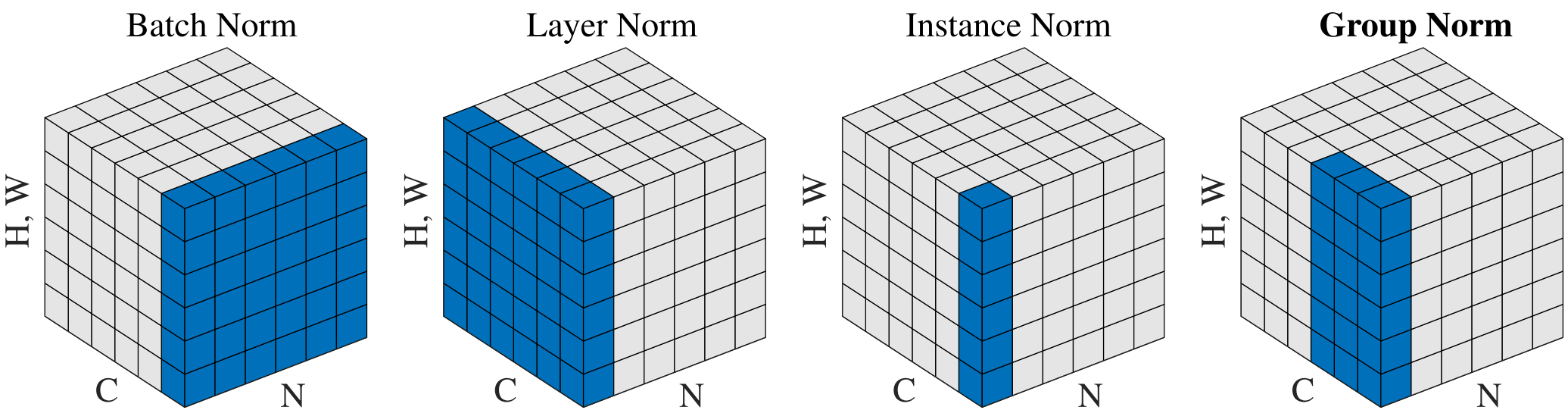


Figure 2 of paper "Group Normalization", https://arxiv.org/abs/1803.08494.

Group Normalization

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

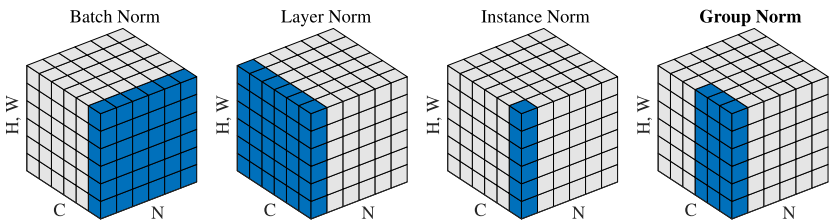


Figure 2 of paper "Group Normalization", https://arxiv.org/abs/1803.08494.

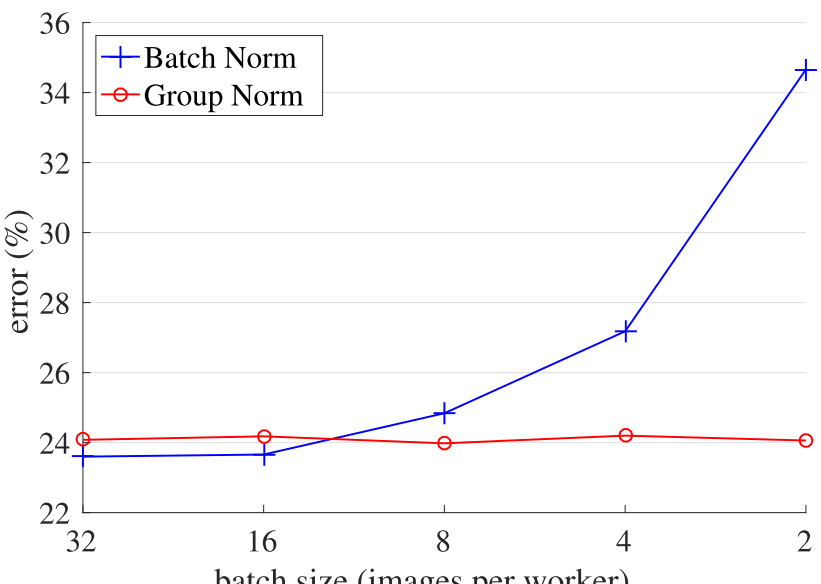


Figure 1 of paper "Group Normalization", https://arxiv.org/abs/1803.08494.

Group Normalization

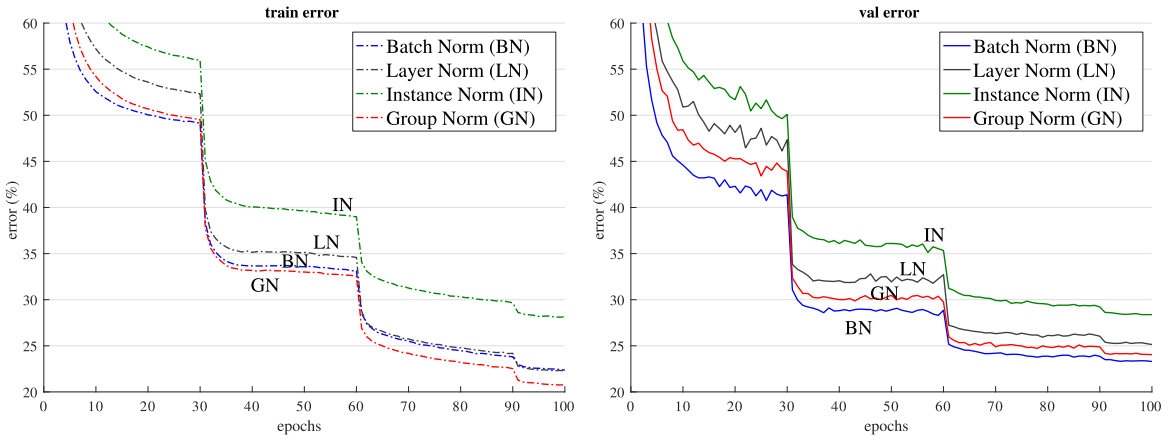


Figure 4. Comparison of error curves with a batch size of 32 images/GPU. We show the ImageNet training error (left) and validation error (right) vs. numbers of training epochs. The model is ResNet-50.

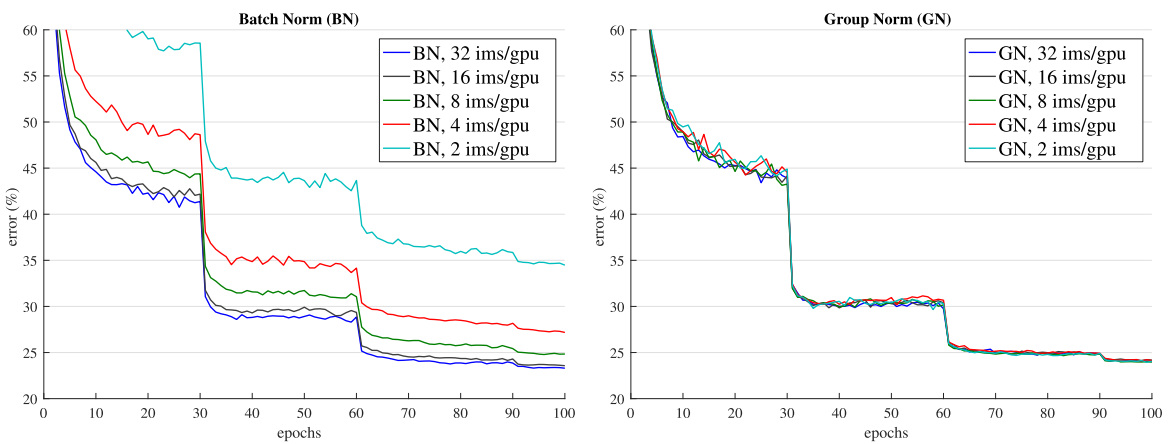


Figure 5. Sensitivity to batch sizes: ResNet-50's validation error of BN (left) and GN (right) trained with 32, 16, 8, 4, and 2 images/GPU.

Figures 4 and 5 of paper "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.

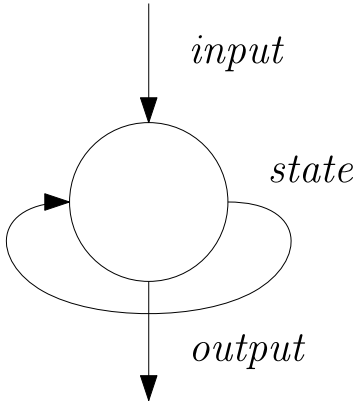
backbone	box head	AP^{bbox}	AP^{bbox}_{50}	AP^{bbox}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
BN^*	-	38.6	59.5	41.9	34.2	56.2	36.1
BN^*	GN	39.5	60.0	43.2	34.4	56.4	**36.3**
GN	GN	**40.0**	**61.0**	**43.3**	**34.8**	**57.3**	**36.3**

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.

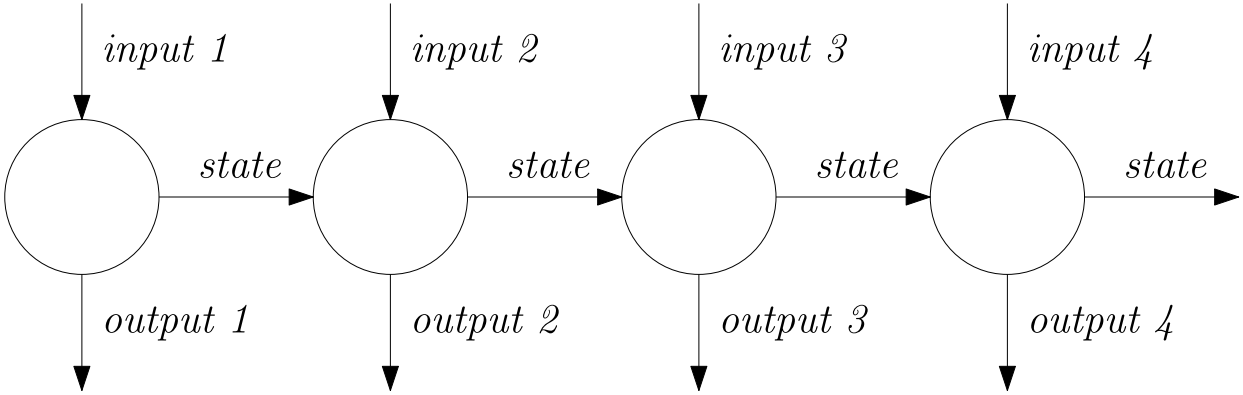
Tables 4 and 5 of paper "Group Normalization", https://arxiv.org/abs/1803.08494.

Recurrent Neural Networks

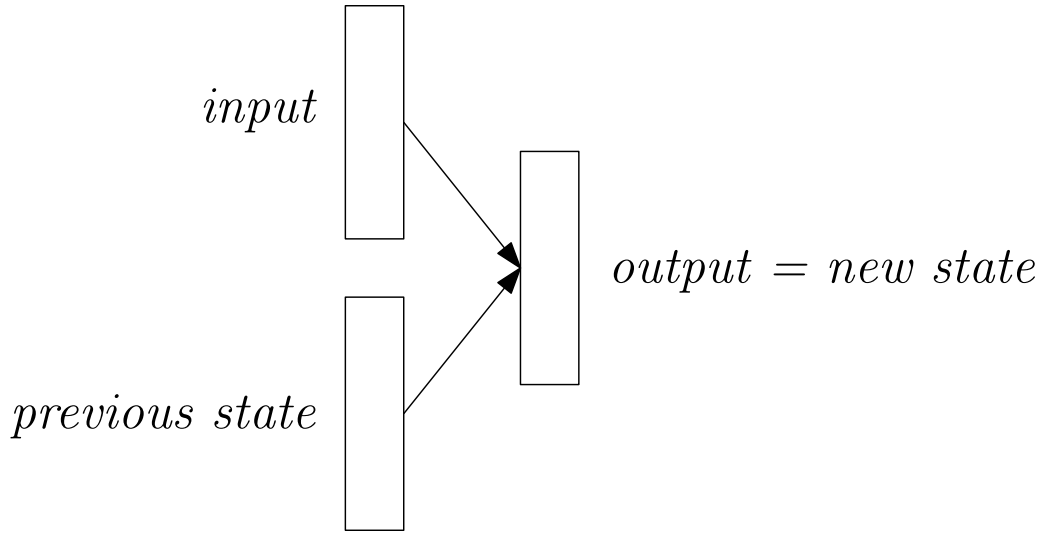
Single RNN cell



Unrolled RNN cells



Basic RNN Cell



Given an input $\boldsymbol{x}^{(t)}$ and previous state $\boldsymbol{s}^{(t-1)}$, the new state is computed as

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}).$$

One of the simplest possibilities is

$$\boldsymbol{s}^{(t)} = \tanh(\boldsymbol{U}\boldsymbol{s}^{(t-1)} + \boldsymbol{V}\boldsymbol{x}^{(t)} + \boldsymbol{b}).$$

Basic RNN Cell

Basic RNN cells suffer a lot from vanishing/exploding gradients (*the challenge of long-term dependencies*).

If we simplify the recurrence of states to

$$\boldsymbol{s}^{(t)} = \boldsymbol{U}\boldsymbol{s}^{(t-1)},$$

we get

$$\boldsymbol{s}^{(t)} = \boldsymbol{U}^t\boldsymbol{s}^{(0)}.$$

If U has eigenvalue decomposition of $\boldsymbol{U} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1}$, we get

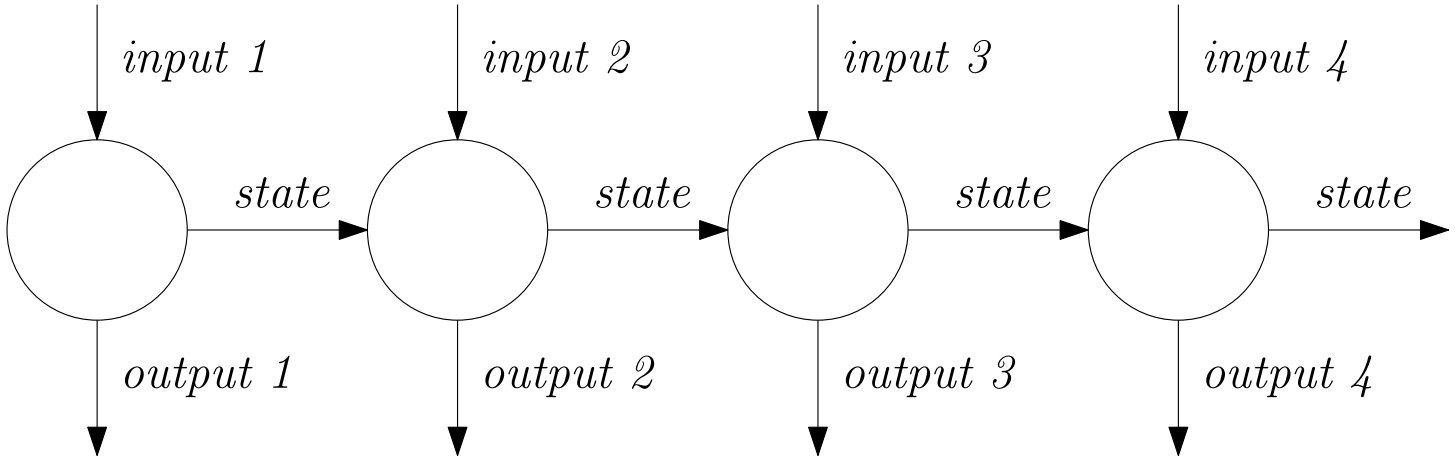
$$\boldsymbol{s}^{(t)} = \boldsymbol{Q}\boldsymbol{\Lambda}^t\boldsymbol{Q}^{-1}\boldsymbol{s}^{(0)}.$$

The main problem is that the *same* function is iteratively applied many times.

Several more complex RNN cell variants have been proposed, which alleviate this issue to some degree, namely **LSTM** and **GRU**.

Sequence Element Classification

Use outputs for individual elements.

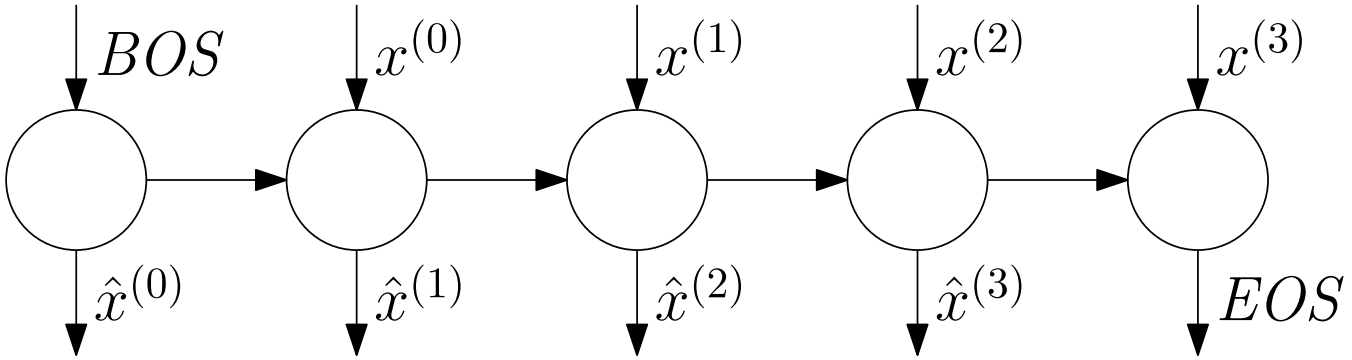


Sequence Representation

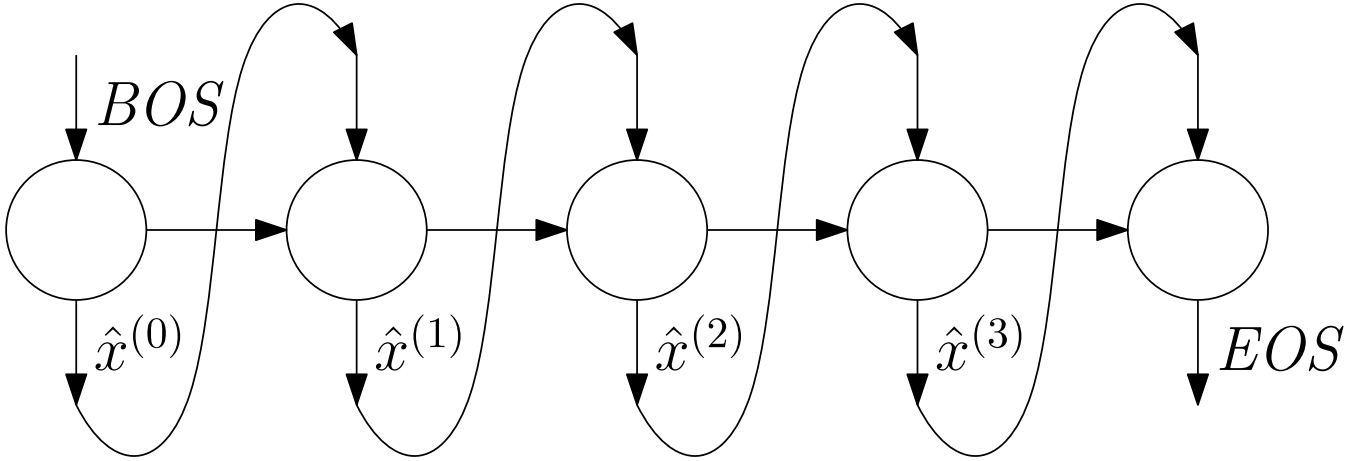
Use state after processing the whole sequence (alternatively, take output of the last element).

Sequence Prediction

During training, predict next sequence element.



During inference, use predicted elements as further inputs.

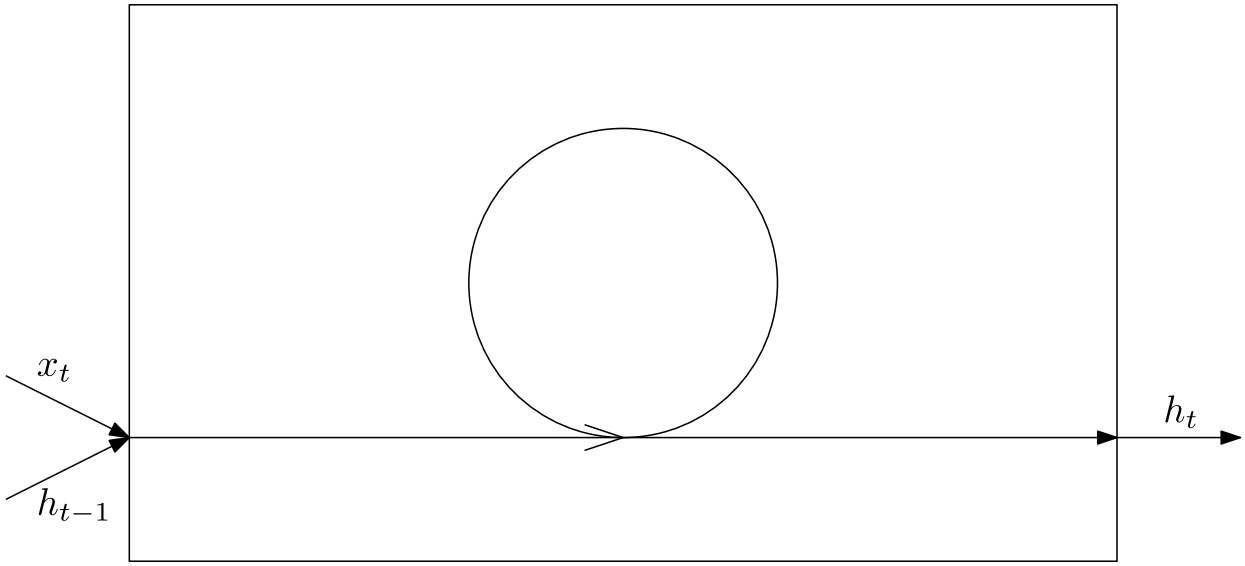


Long Short-Term Memory

Hochreiter & Schmidhuber (1997) suggested that to enforce *constant error flow*, we would like

$$f' = \mathbf{1}.$$

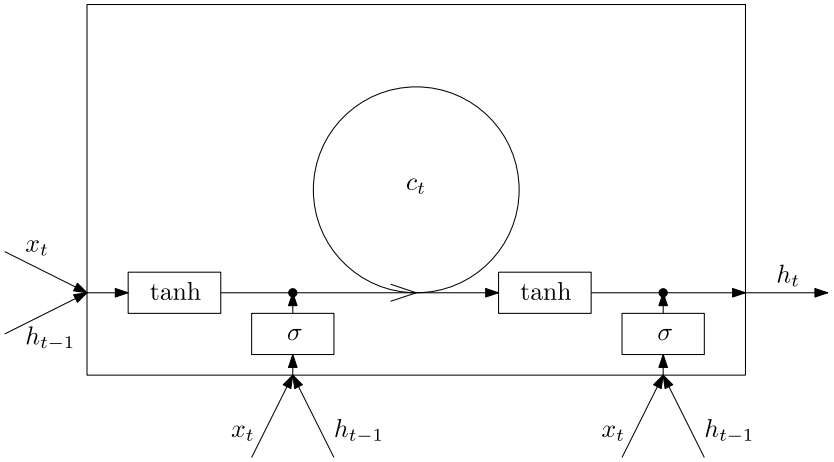
They propose to achieve that by a *constant error carrousel*.



Long Short-Term Memory

They also propose an *input* and *output* gates which control the flow of information into and out of the carrousel (*memory cell* $\boldsymbol{c}_t$).

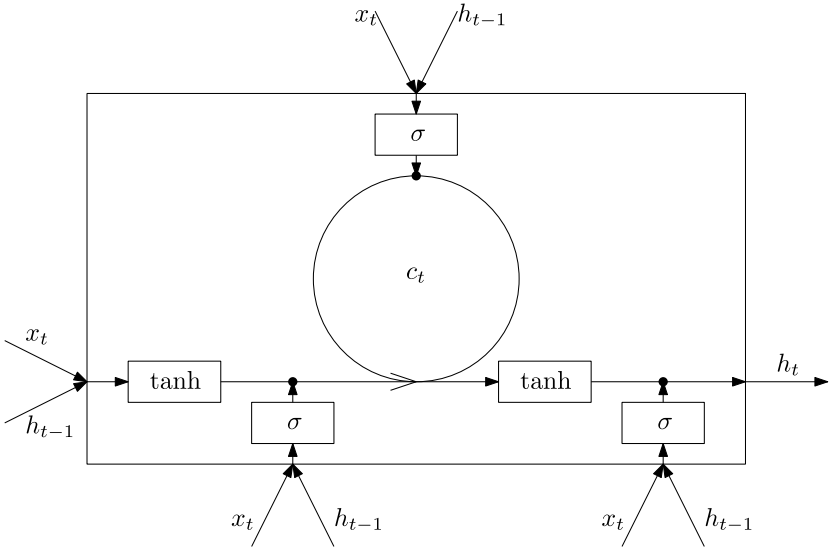
$$\boldsymbol{i}_t \leftarrow \sigma(\boldsymbol{W}^i \boldsymbol{x}_t + \boldsymbol{V}^i \boldsymbol{h}_{t-1} + \boldsymbol{b}^i)$$
$$\boldsymbol{o}_t \leftarrow \sigma(\boldsymbol{W}^o \boldsymbol{x}_t + \boldsymbol{V}^o \boldsymbol{h}_{t-1} + \boldsymbol{b}^o)$$
$$\boldsymbol{c}_t \leftarrow \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \cdot \tanh(\boldsymbol{W}^y \boldsymbol{x}_t + \boldsymbol{V}^y \boldsymbol{h}_{t-1} + \boldsymbol{b}^y)$$
$$\boldsymbol{h}_t \leftarrow \boldsymbol{o}_t \cdot \tanh(\boldsymbol{c}_t)$$



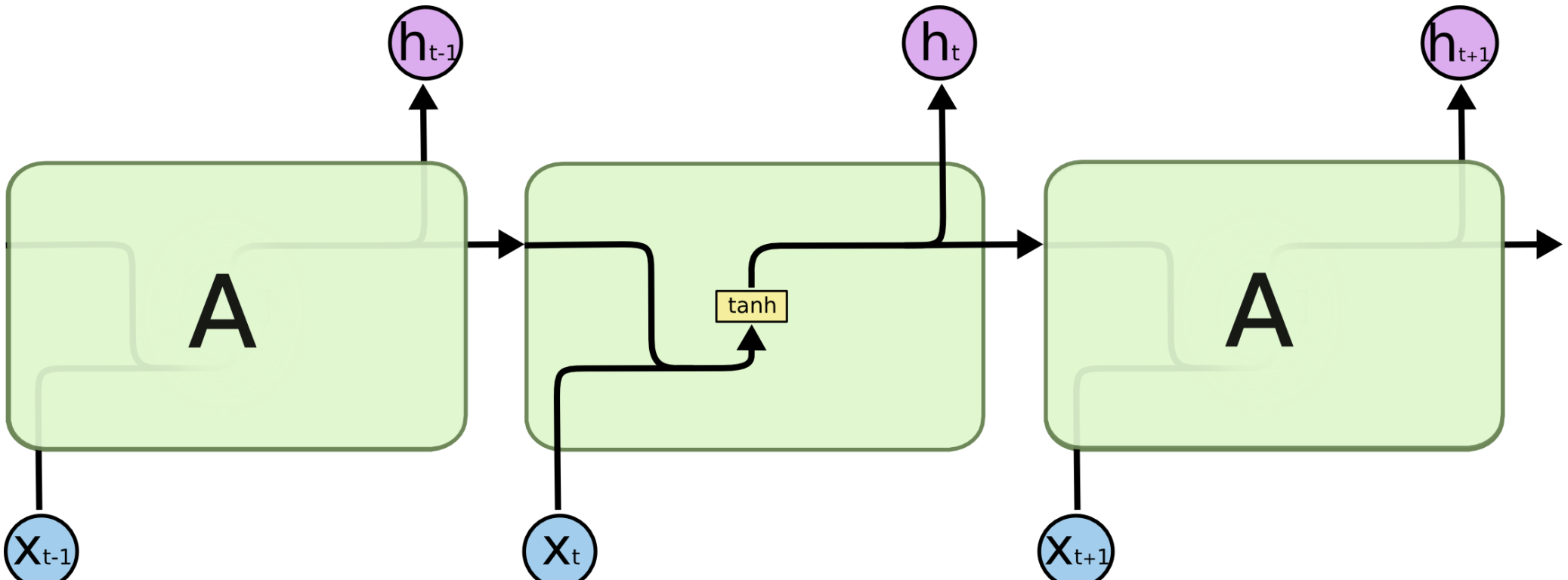
Long Short-Term Memory

Later in Gers, Schmidhuber & Cummins (1999) a possibility to *forget* information from memory cell $\boldsymbol{c}_t$ was added.

$$\begin{aligned}
\boldsymbol{i}_t &\leftarrow \sigma(\boldsymbol{W}^i \boldsymbol{x}_t + \boldsymbol{V}^i \boldsymbol{h}_{t-1} + \boldsymbol{b}^i) \\
\boldsymbol{f}_t &\leftarrow \sigma(\boldsymbol{W}^f \boldsymbol{x}_t + \boldsymbol{V}^f \boldsymbol{h}_{t-1} + \boldsymbol{b}^f) \\
\boldsymbol{o}_t &\leftarrow \sigma(\boldsymbol{W}^o \boldsymbol{x}_t + \boldsymbol{V}^o \boldsymbol{h}_{t-1} + \boldsymbol{b}^o) \\
\boldsymbol{c}_t &\leftarrow \boldsymbol{f}_t \cdot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \cdot \tanh(\boldsymbol{W}^y \boldsymbol{x}_t + \boldsymbol{V}^y \boldsymbol{h}_{t-1} + \boldsymbol{b}^y) \\
\boldsymbol{h}_t &\leftarrow \boldsymbol{o}_t \cdot \tanh(\boldsymbol{c}_t)
\end{aligned}$$

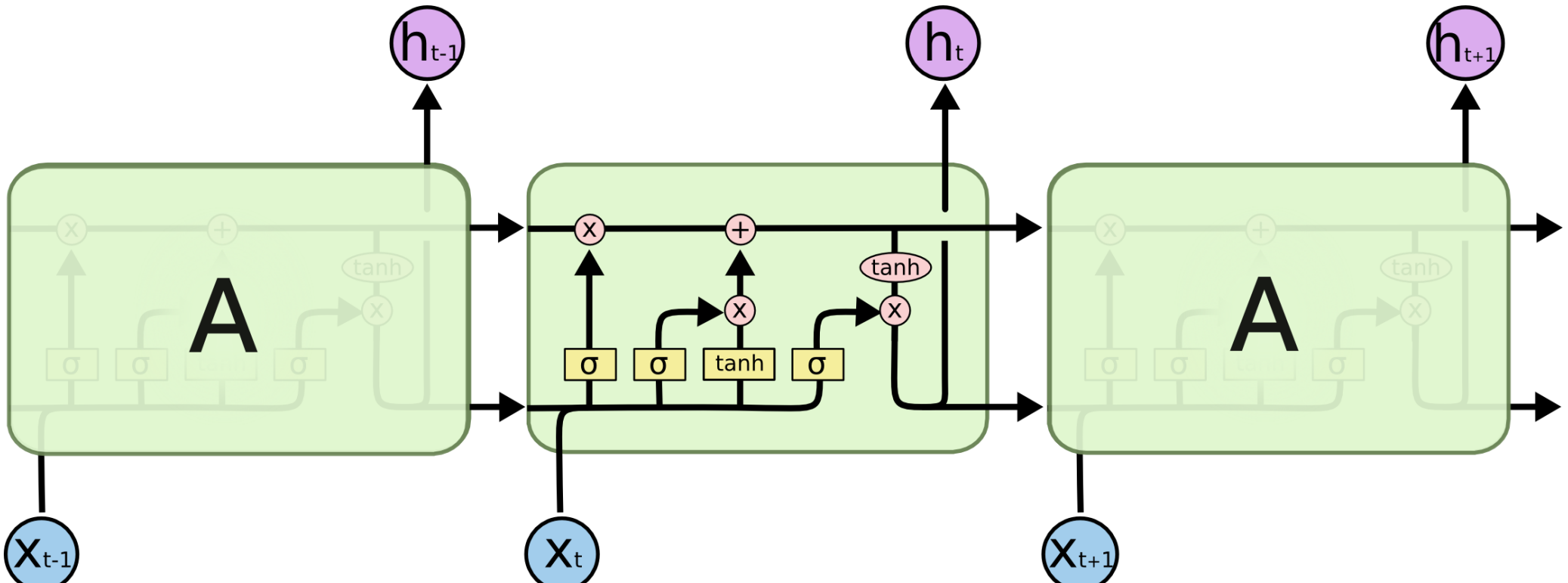


Long Short-Term Memory



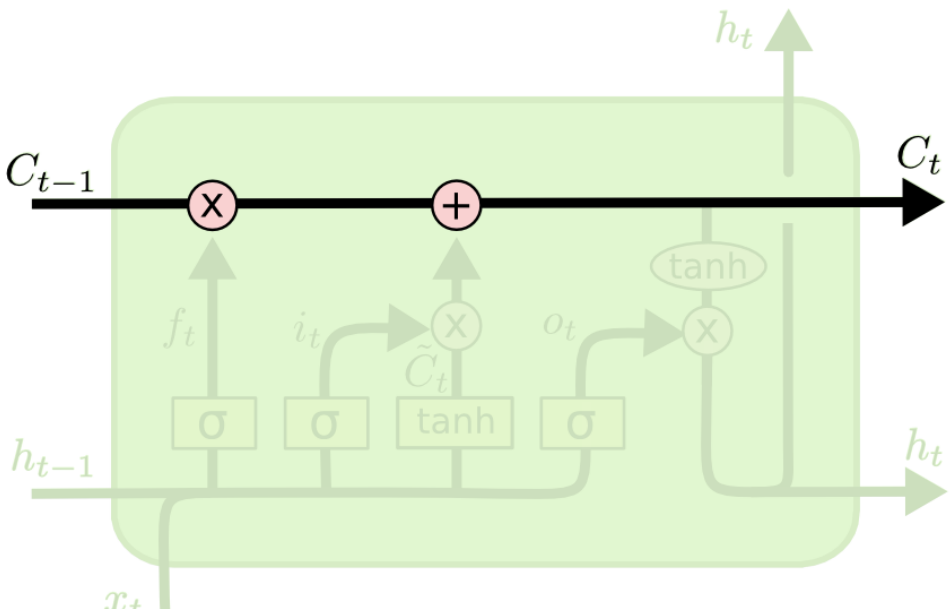
http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-SimpleRNN.png

Long Short-Term Memory



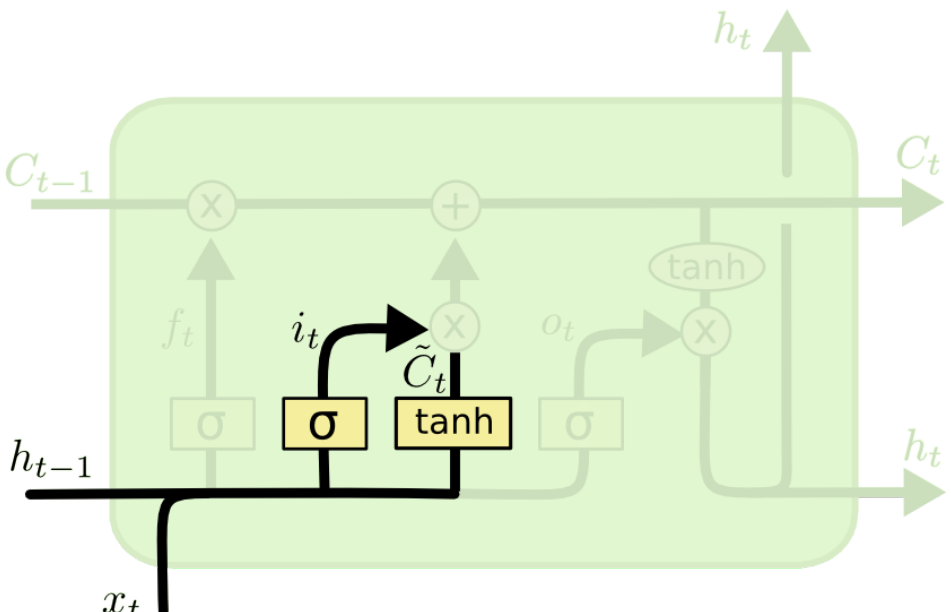
http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-chain.png

Long Short-Term Memory



http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-C-line.png

Long Short-Term Memory

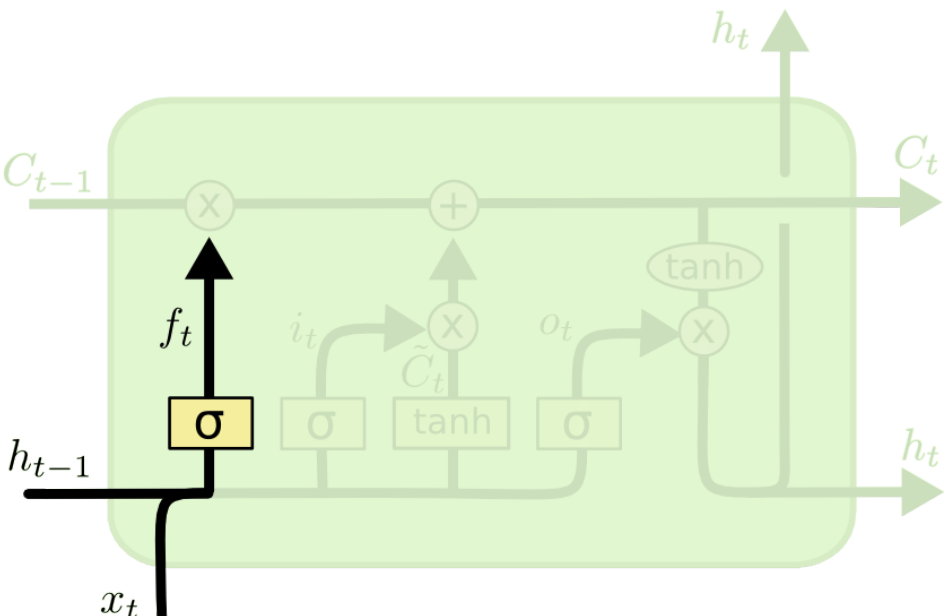


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-focus-i.png

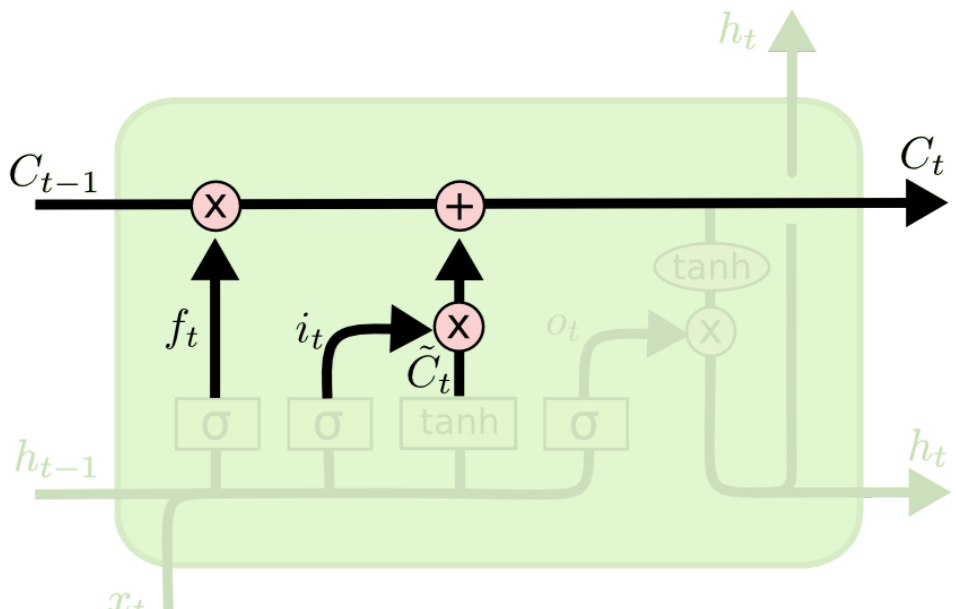
Long Short-Term Memory



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-focus-f.png

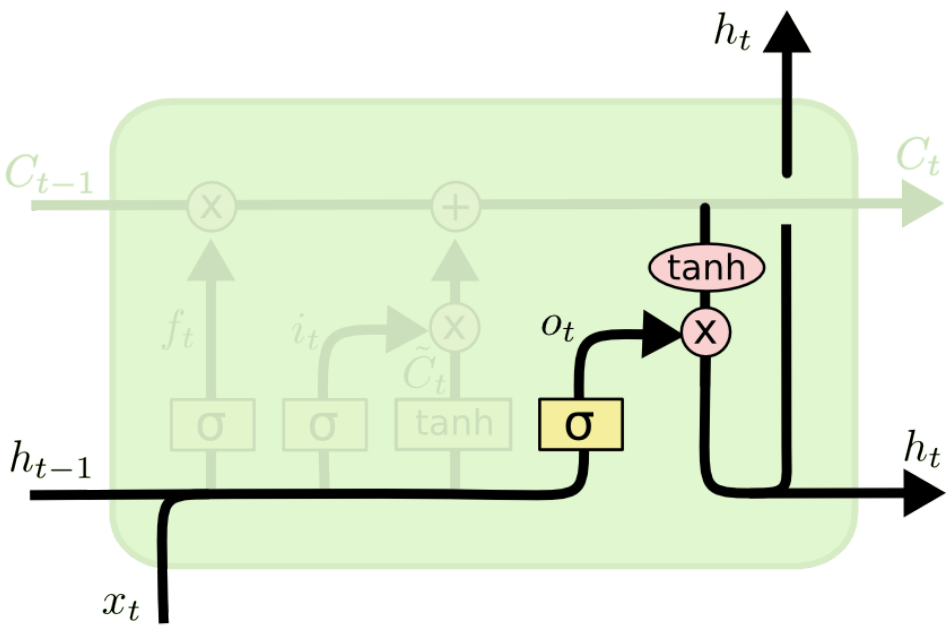
Long Short-Term Memory



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-focus-C.png

Long Short-Term Memory



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-focus-o.png