Computer Vision

Lecture 09: Advanced Vision Applications

Contents

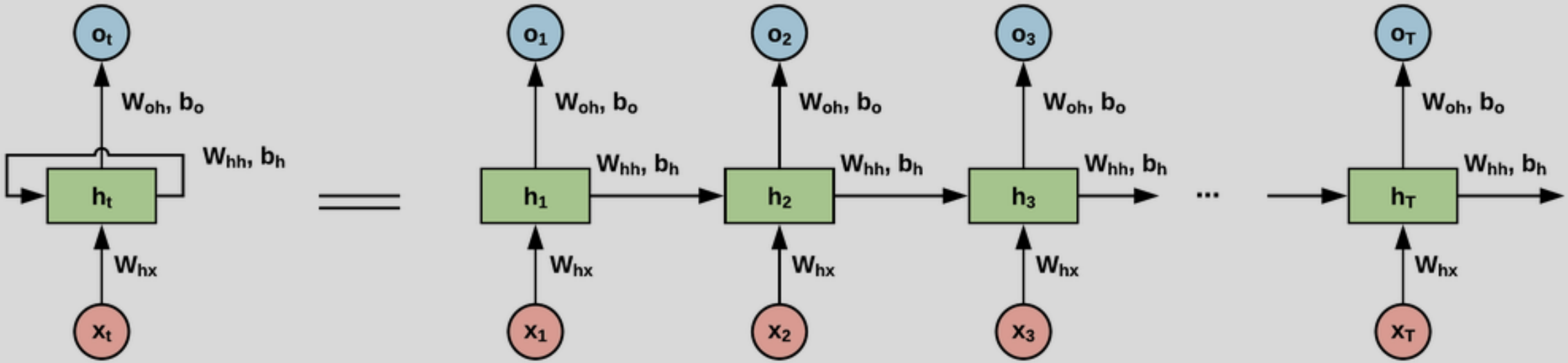
Transformer for vision

- RNN
- LSTM
- Transformer
- DETR

Advanced YOLO

- Yolo v2
- Yolo v3

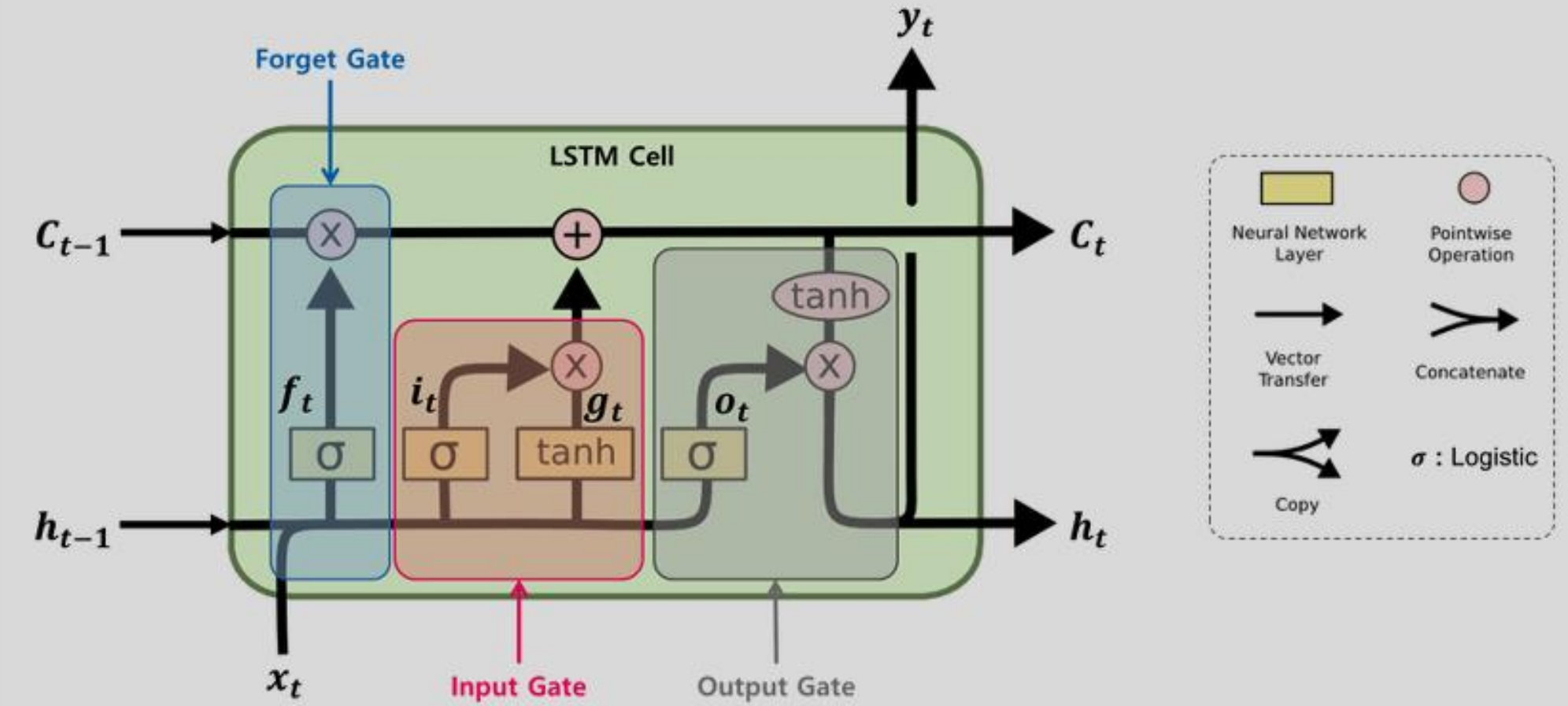
Recurrent Neural Network



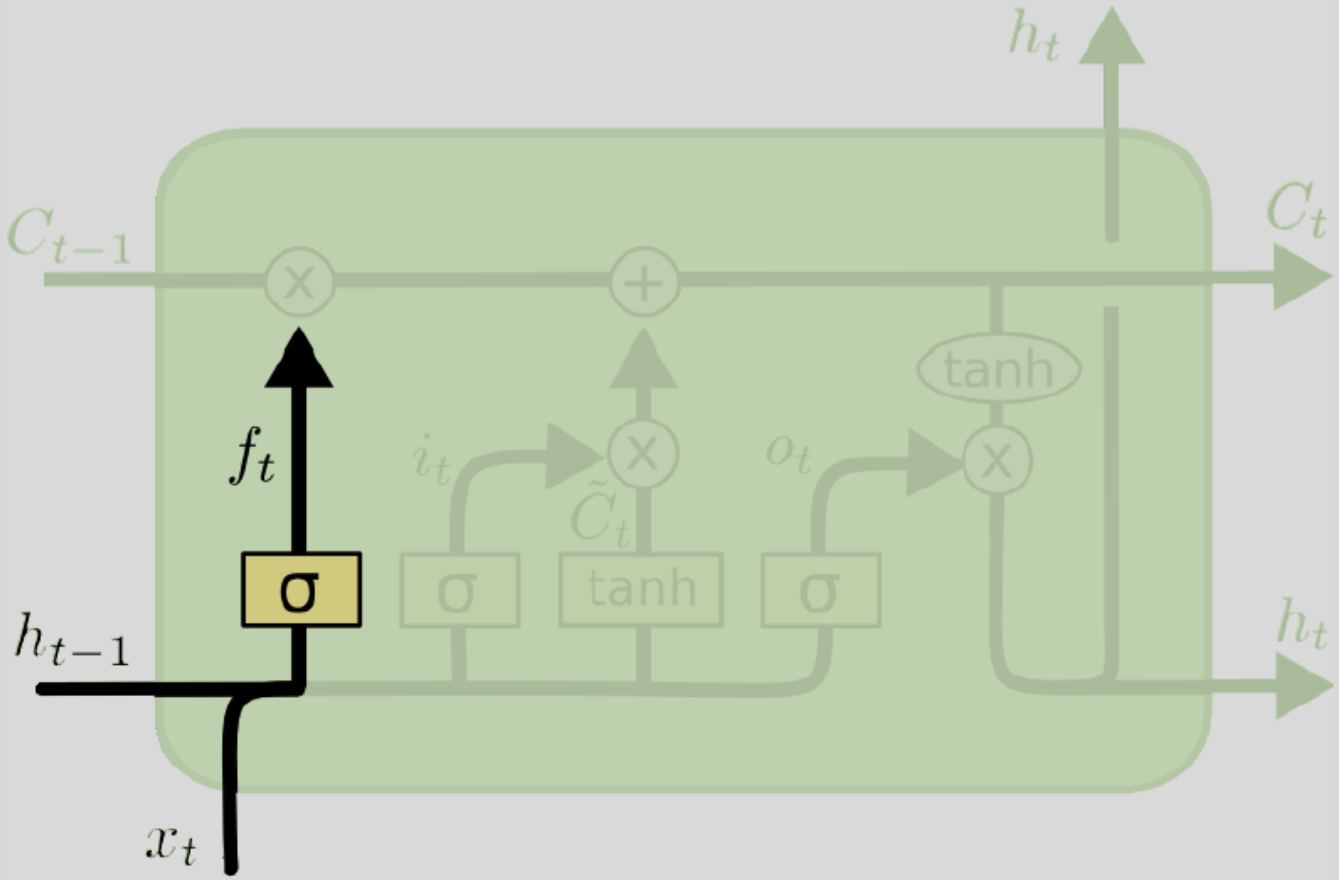
Limitations of RNN

- Non-trivial to parallelize
- Vanishing gradient

Long Term Short Memory (LSTM)

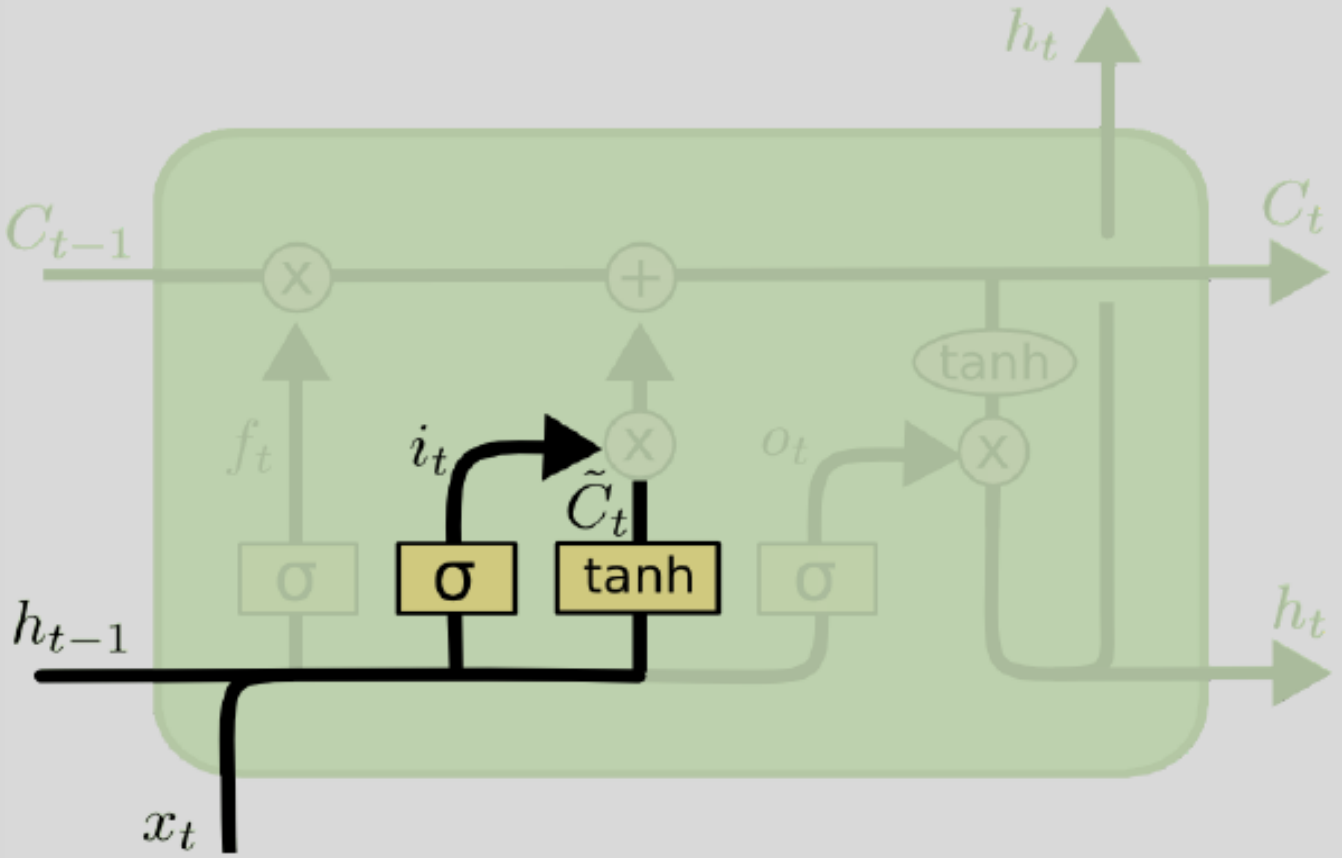


Long Term Short Memory (LSTM)



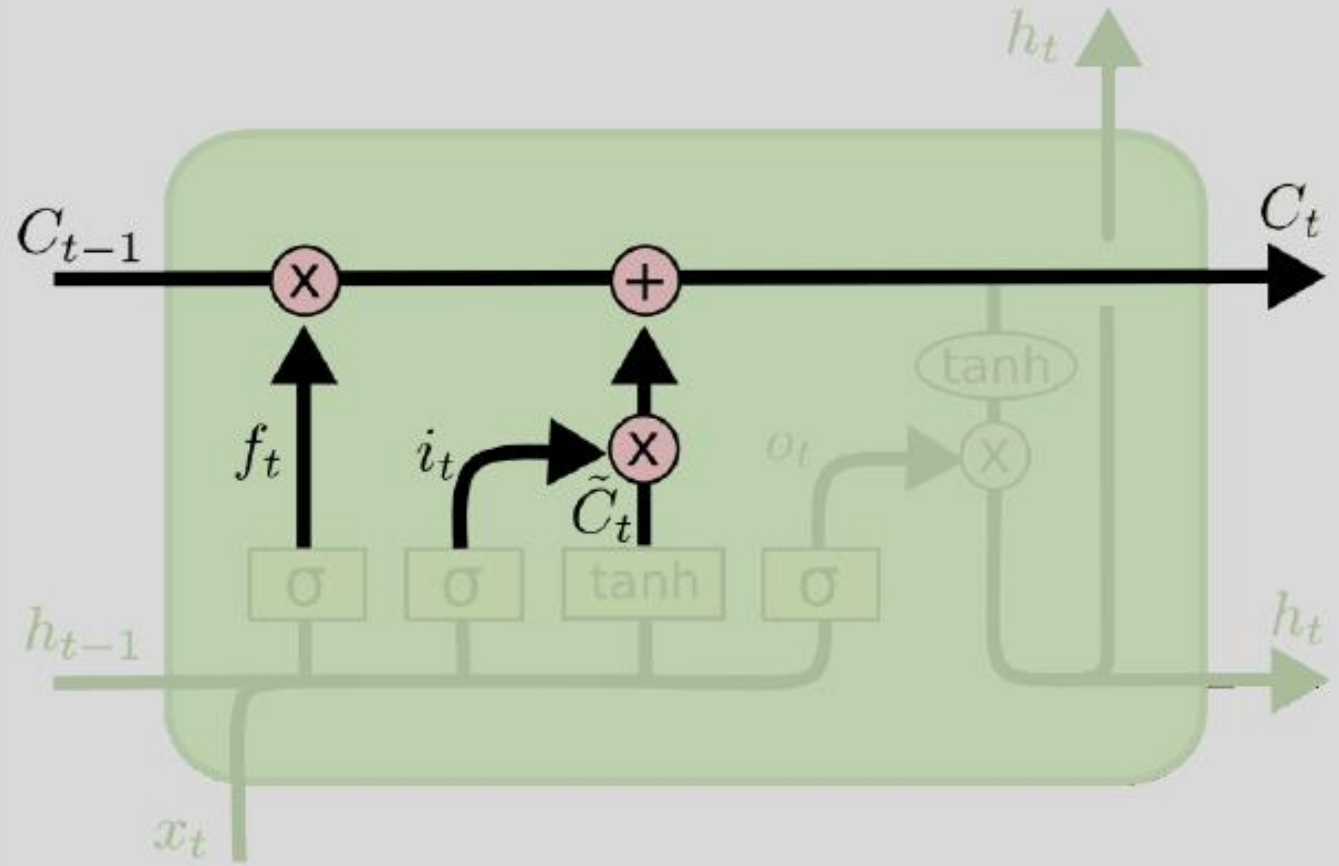
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Long Term Short Memory (LSTM)



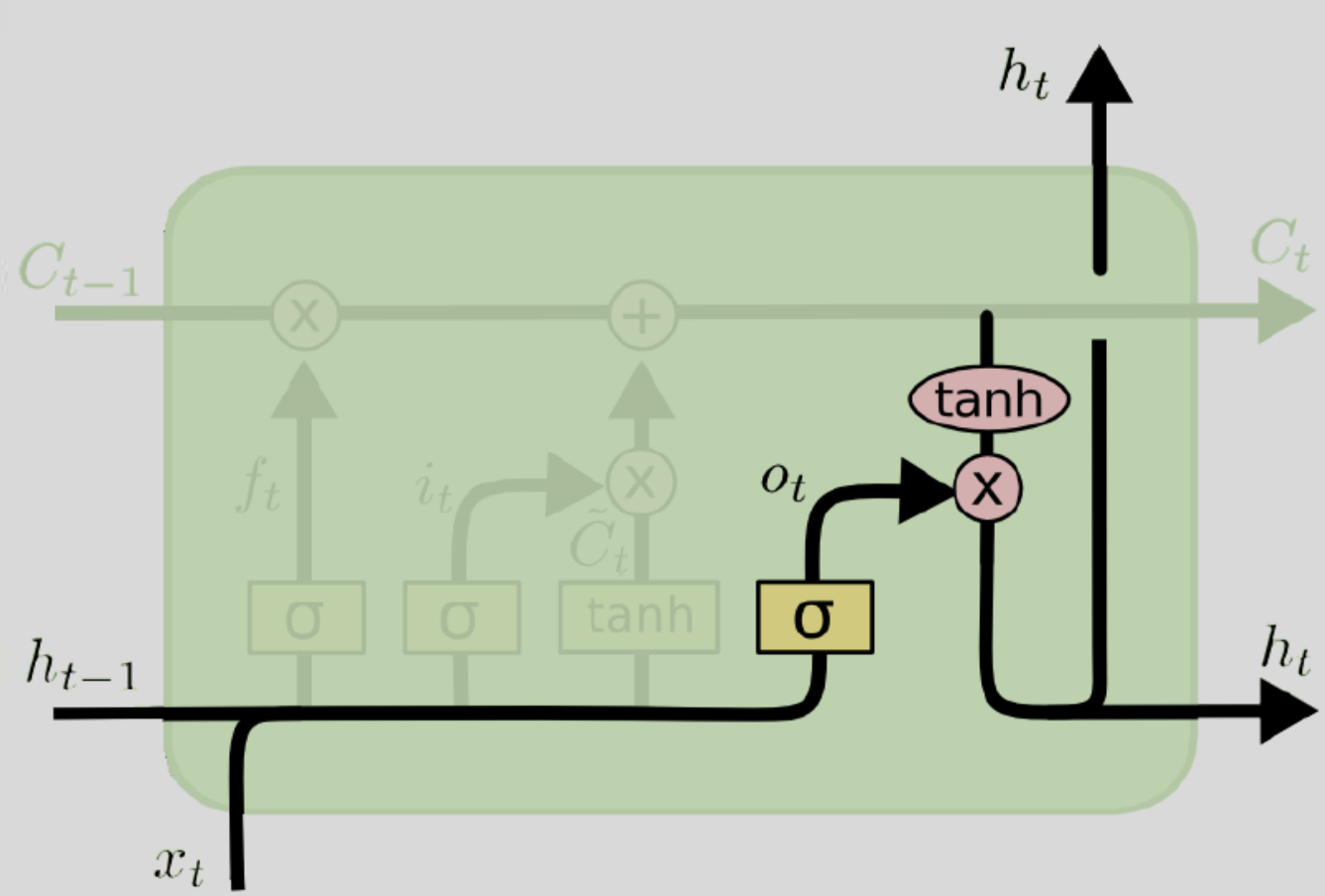
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C} = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Long Term Short Memory (LSTM)



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

Long Term Short Memory (LSTM)



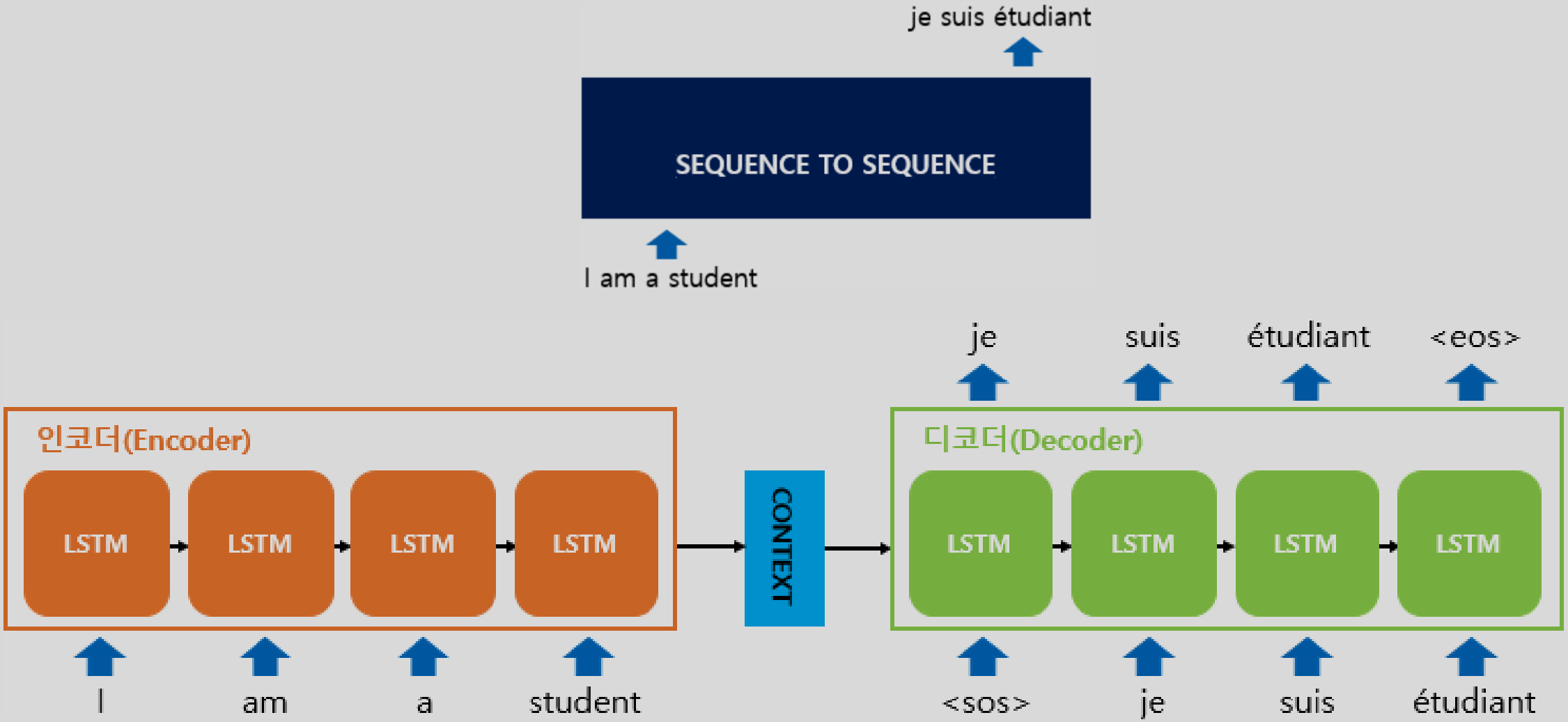
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * tanh(C_t)$$

LSTM

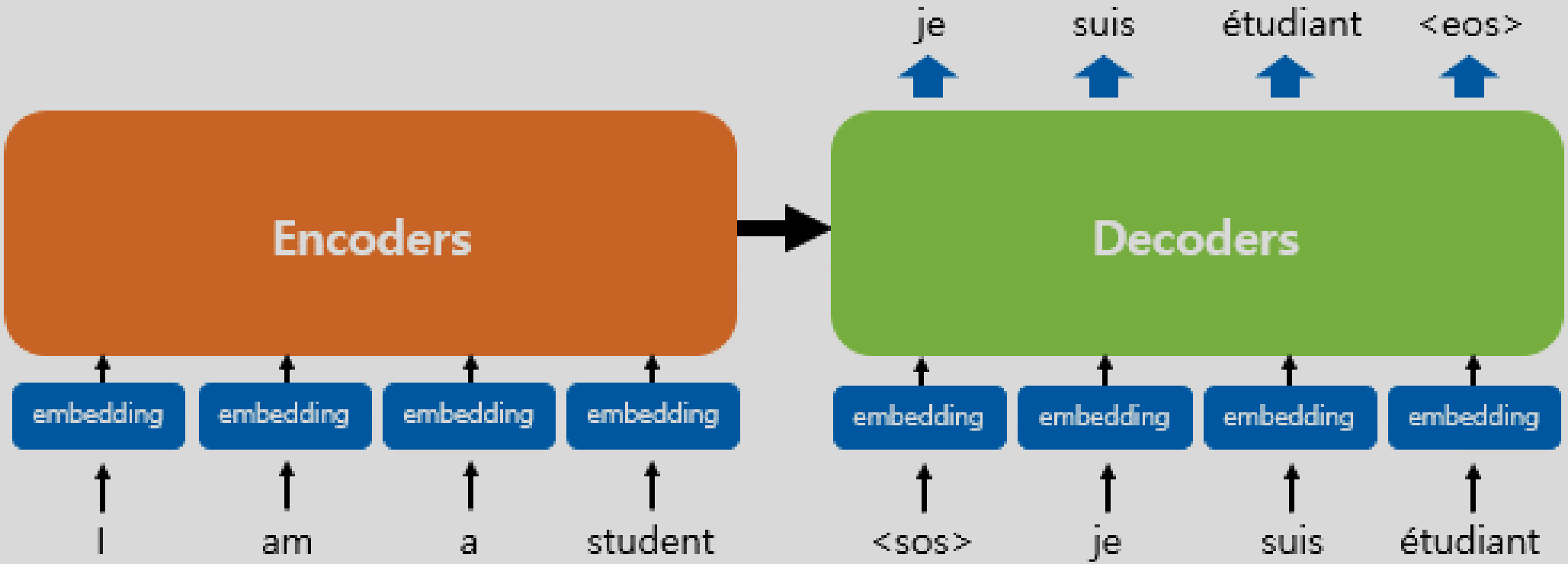
- Forget gate: How much retain past info.
- Input gate: How much use the new info.
- Output gate: How much use the new output.

- Become robust by learning how to forget and retain.

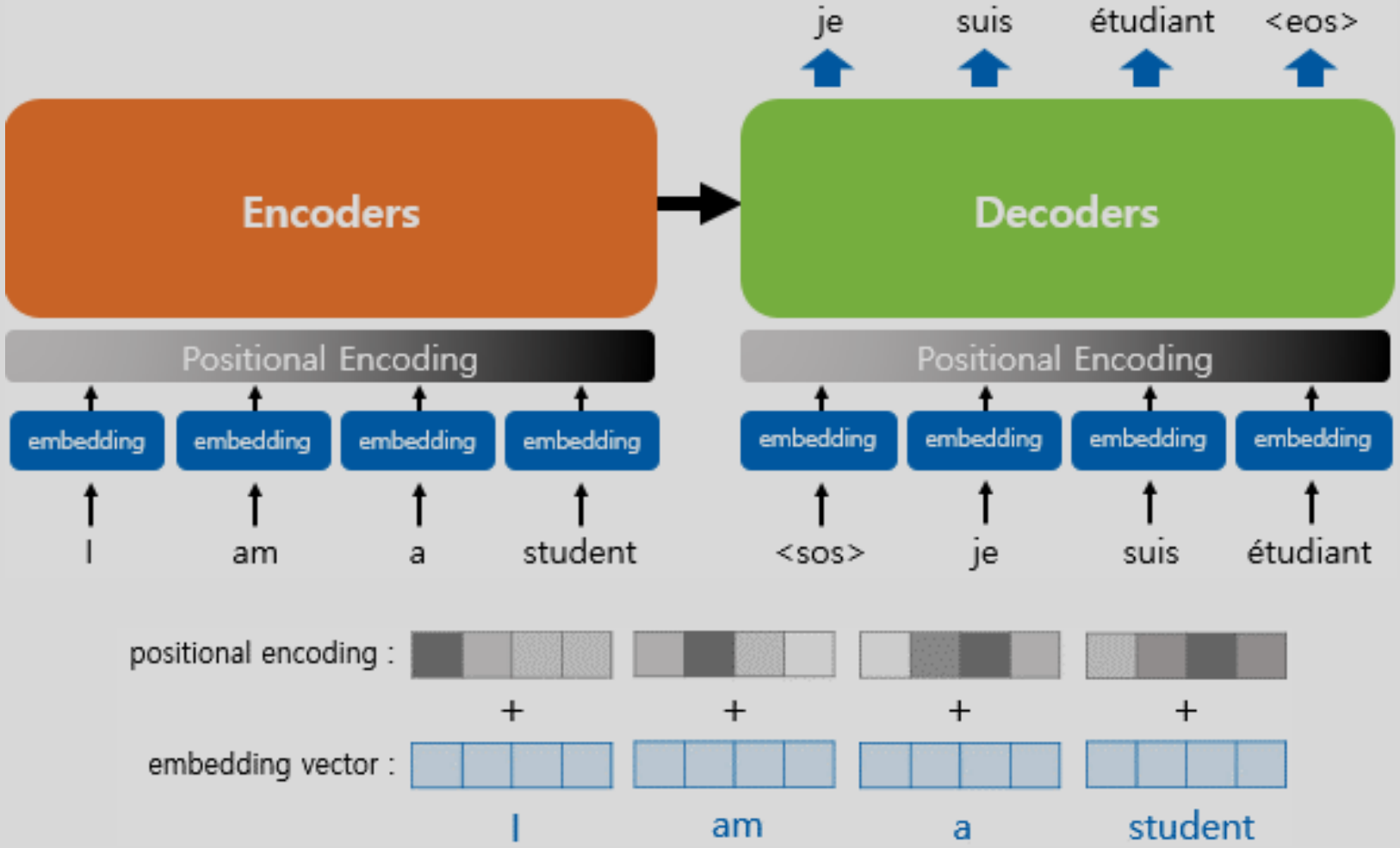
Seq2seq using LSTMs



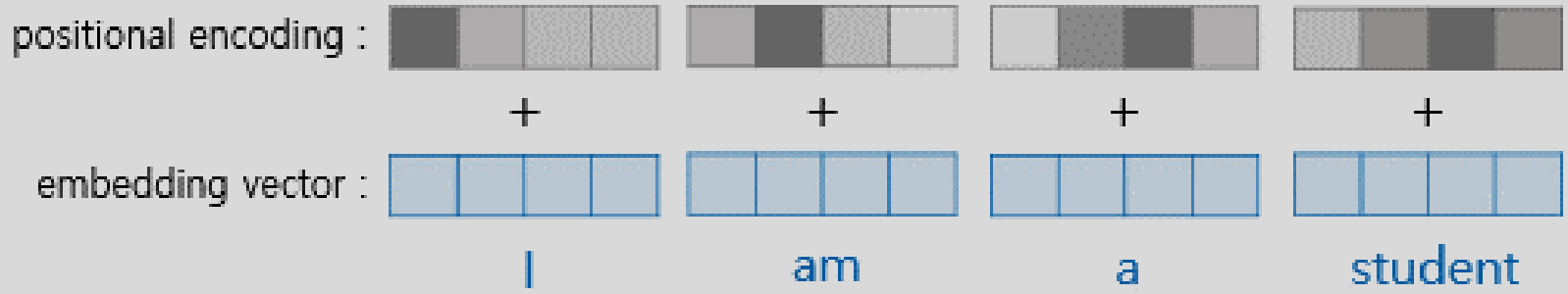
Transformer

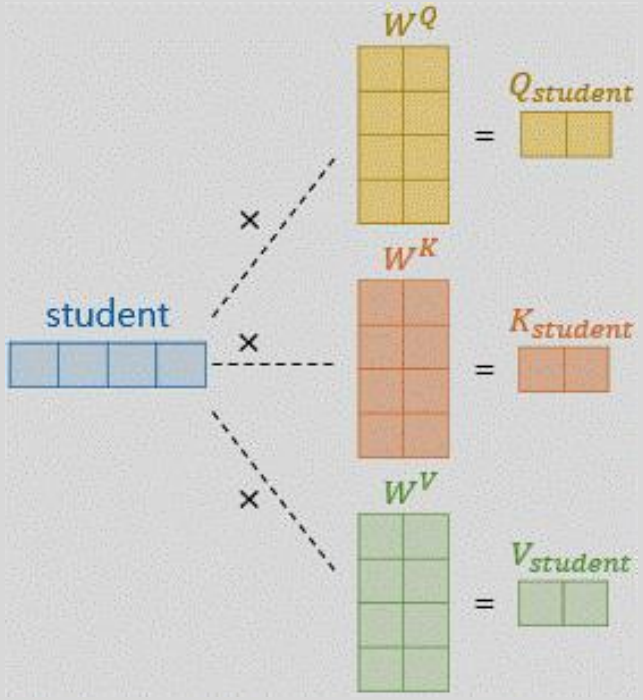


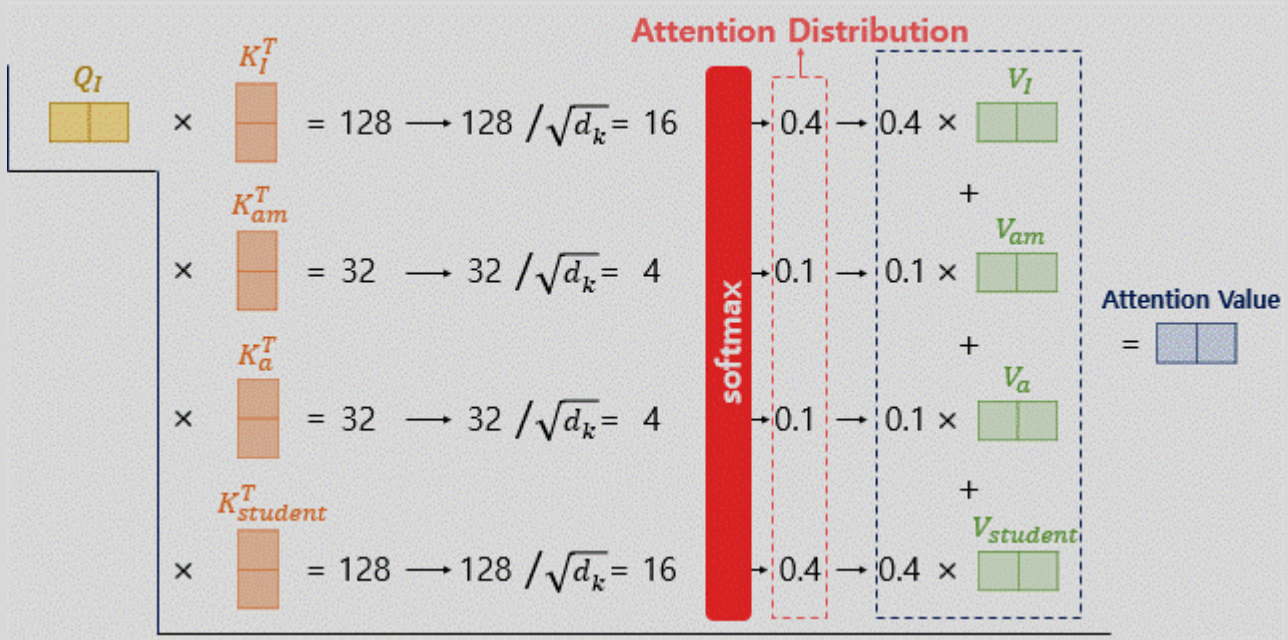
Transformer



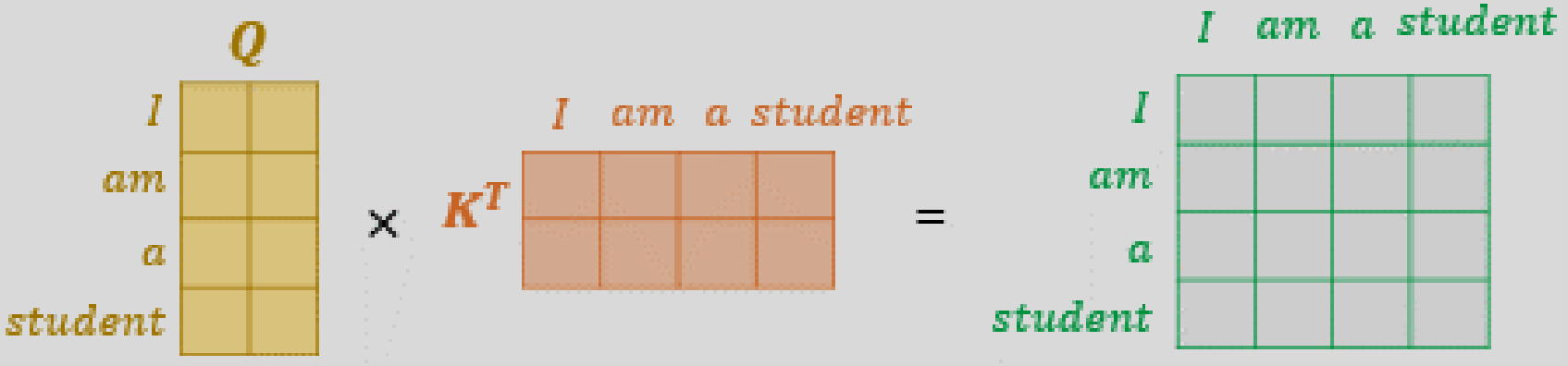
Transformer

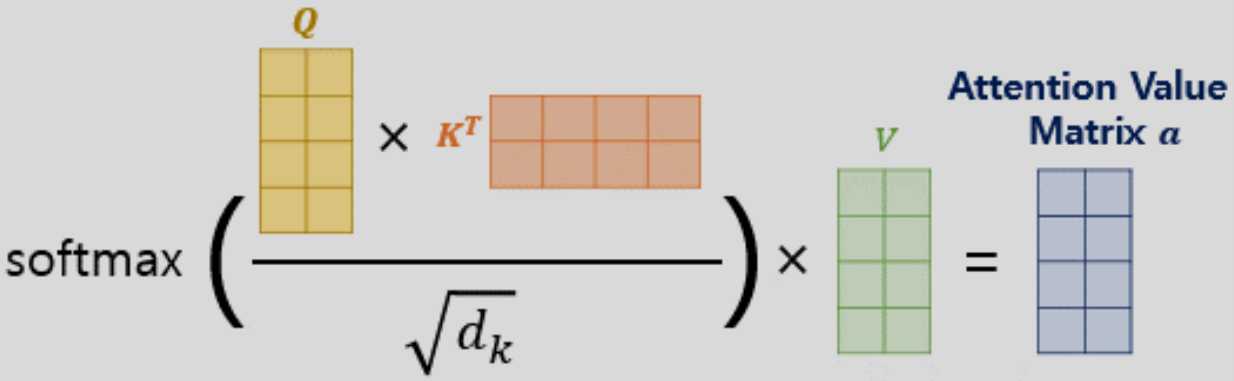






Transformer



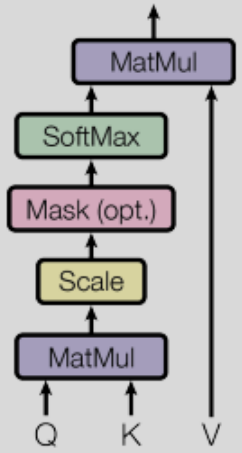


Transformer

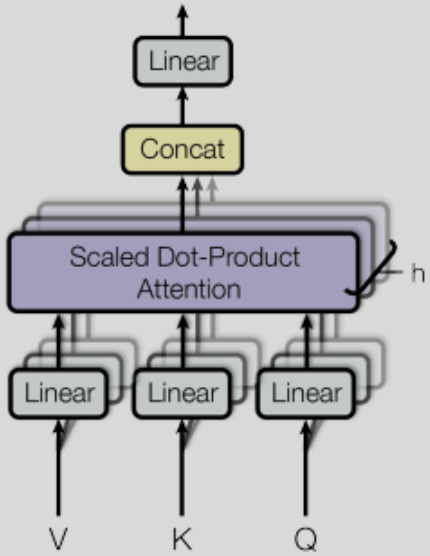
- Easy to parallelize.
- Not lose any information.
  - Dense attention.

Transformer

Scaled Dot-Product Attention



Multi-Head Attention



$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Transformer

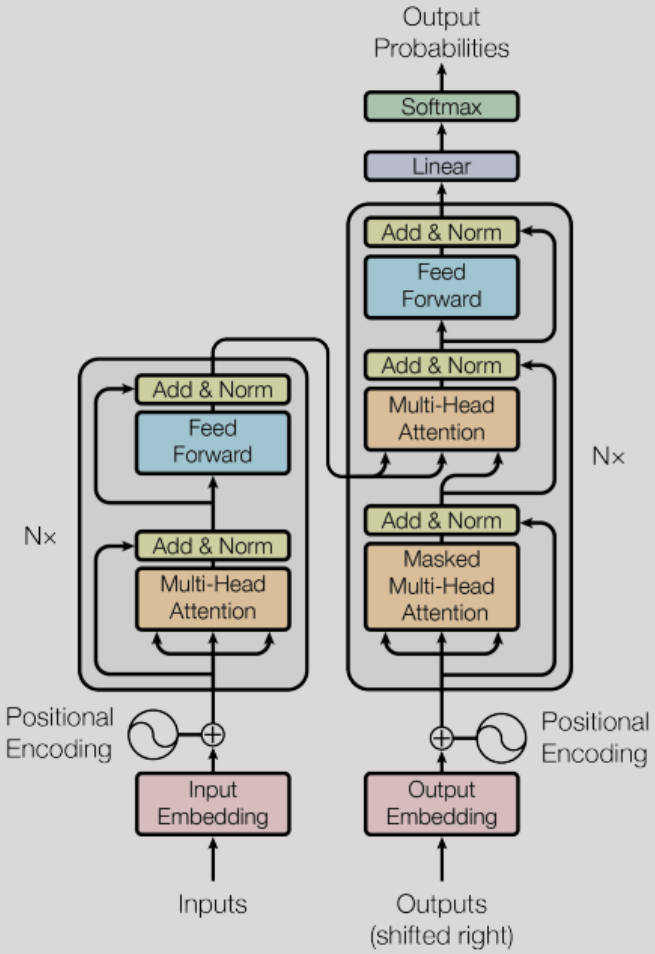


Figure 1: The Transformer - model architecture.

Attention is all you need, NIPS, 2017

DETR

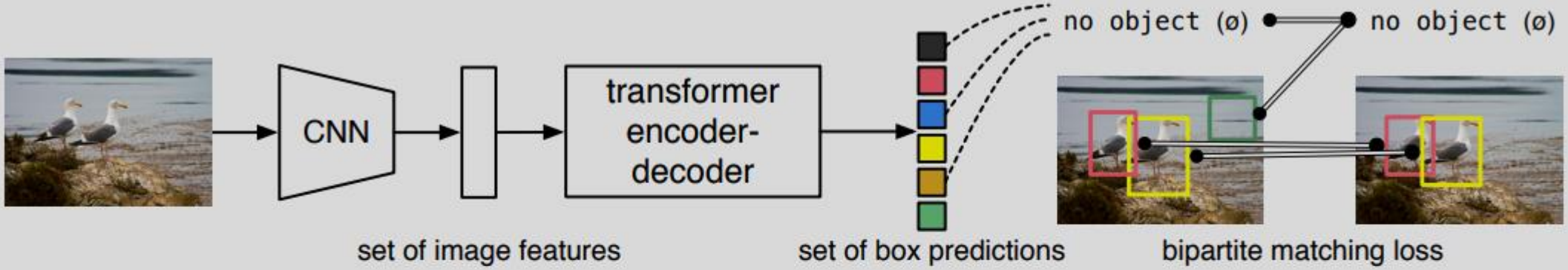


Fig. 1: DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes. Prediction with no match should yield a "`no object`" ($\varnothing$) class prediction.

DETR

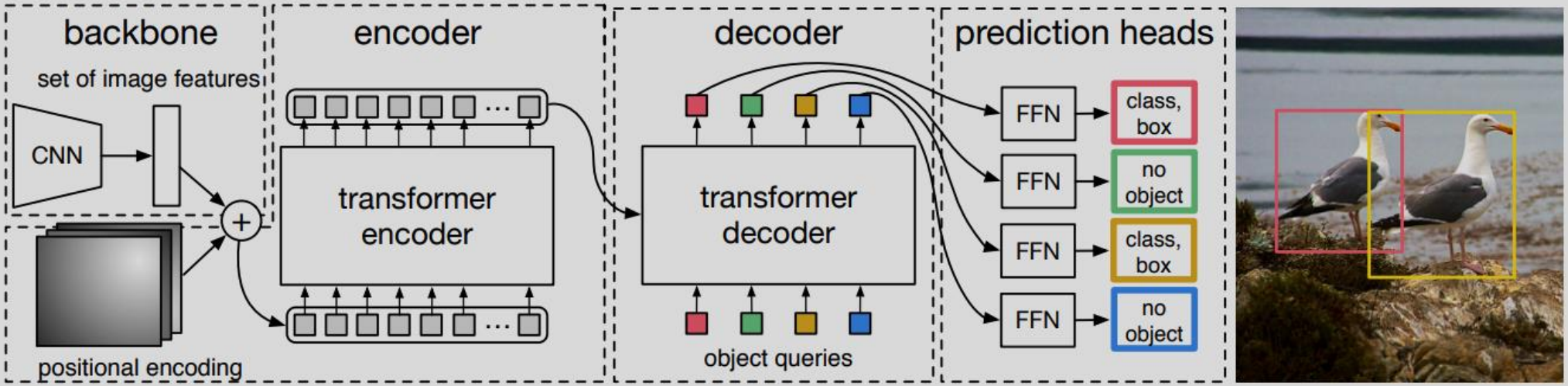
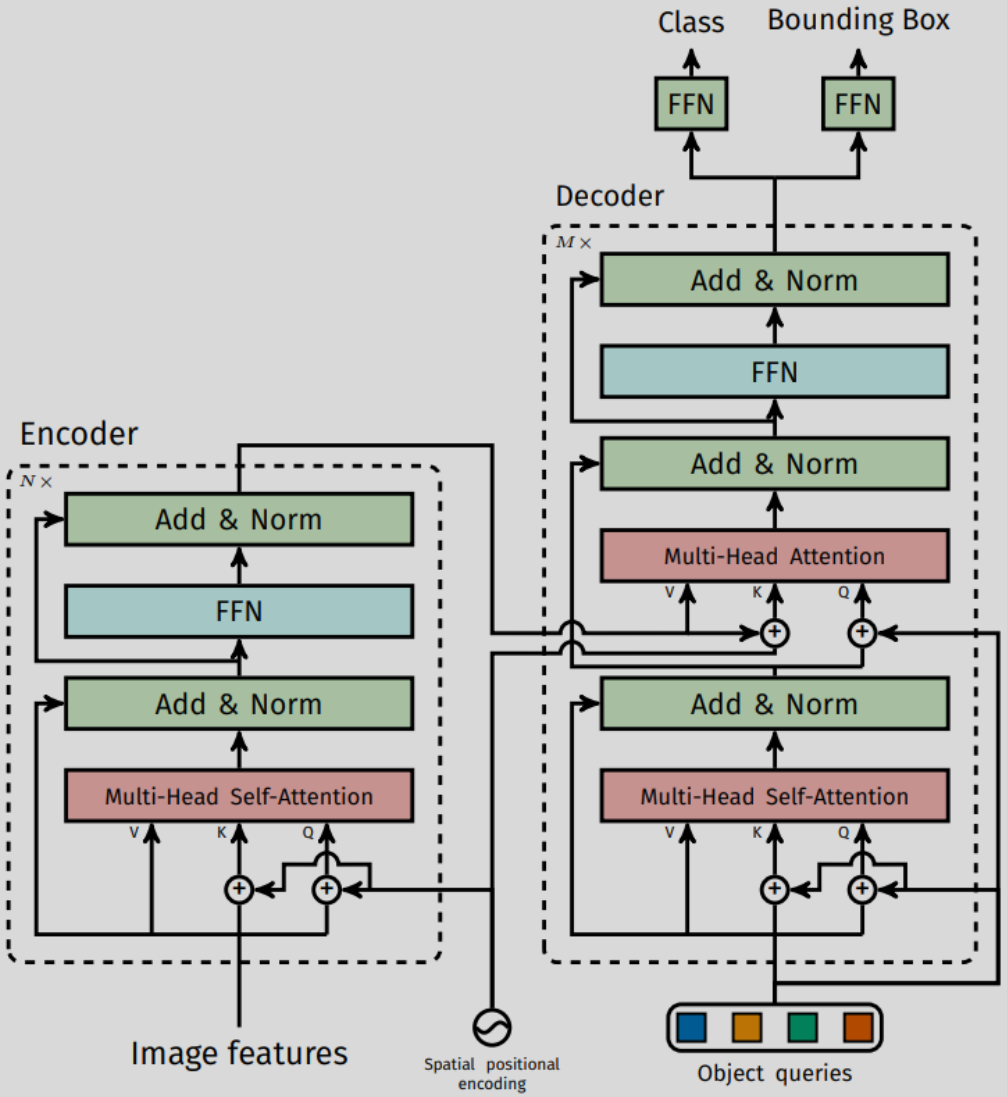


Fig. 2: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small fixed number of learned positional embeddings, 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.

End-to-End Object Detection with Transformers, ECCV'20

DETR



End-to-End Object Detection with Transformers, ECCV'20

DETR

1) Find the optimal prediction set:

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

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

2) Minimize the loss for the optimal prediction set:

$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \left[-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}}(i)) \right]$$

DETR

Table 1: Comparison with RetinaNet and Faster R-CNN with a ResNet-50 and ResNet-101 backbones on the COCO validation set. The top section shows results for models in Detectron2 [49], the middle section shows results for models with GIoU [37], random crops train-time augmentation, and the long `9x` training schedule. DETR models achieve comparable results to heavily tuned Faster R-CNN baselines, having lower AP_S but greatly improved AP_L. We use torchscript models to measure FLOPS and FPS. Results without R101 in the name correspond to ResNet-50.

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**

DETR

- Robust performance; while slow speed.

- Faster version:

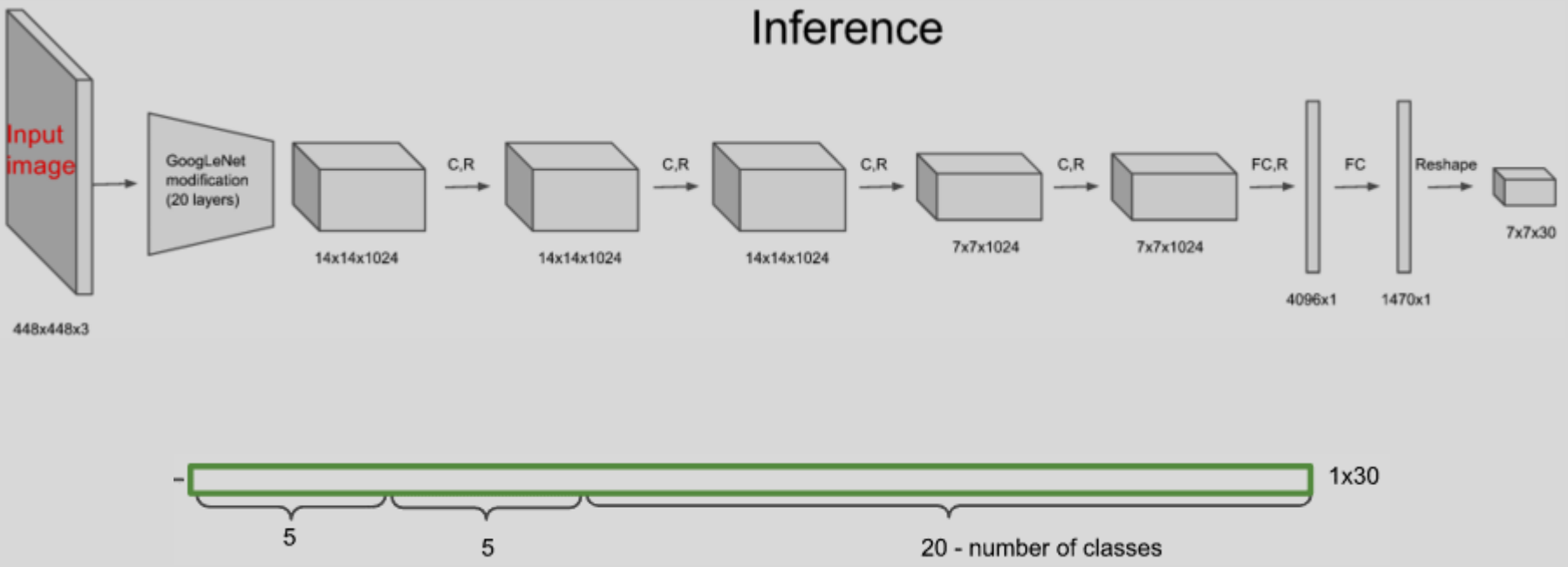
Deformable DETR: Deformable Transformers for End-to-End Object Detection, ICLR’21.

YOLO (You only look once, CVPR'16)

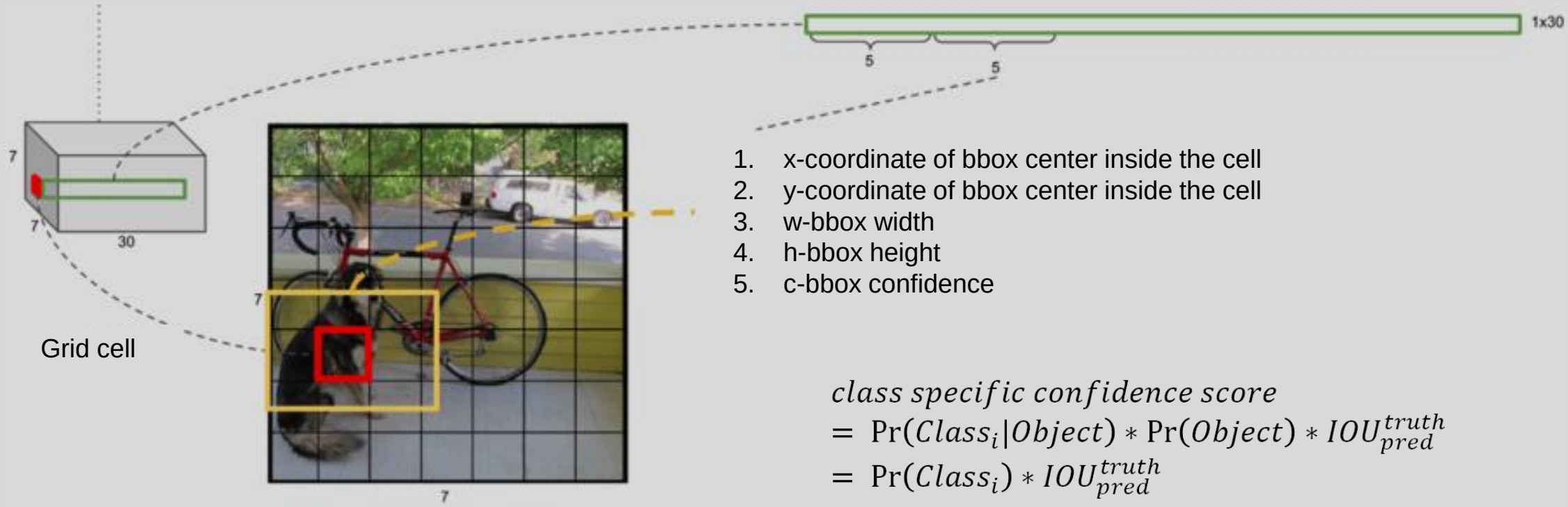
- 1 stage algorithm

YOLO (You only look once, CVPR'16)

- Two candidates for each grid, 20 classes:



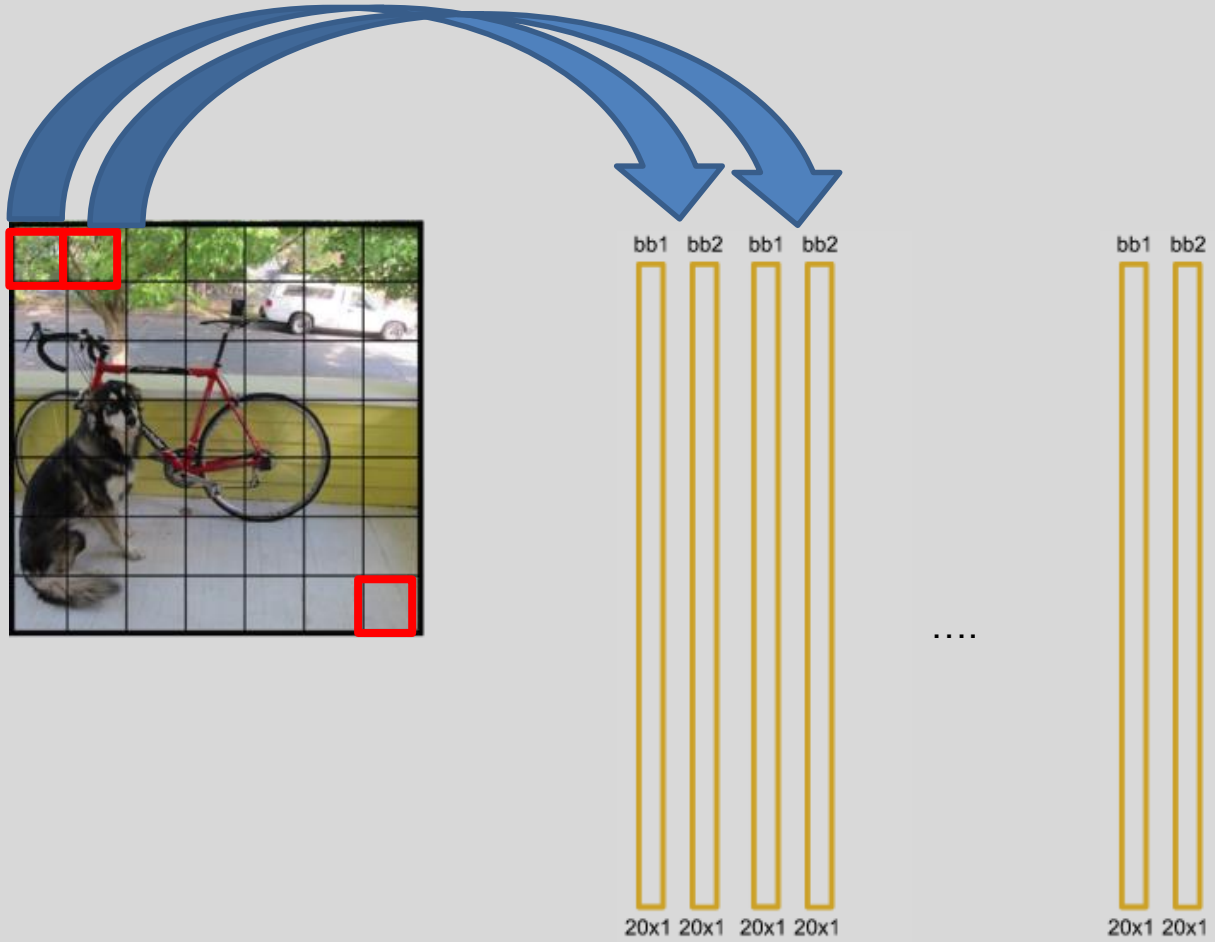
YOLO (You only look once, CVPR'16)



1. x-coordinate of bbox center inside the cell
2. y-coordinate of bbox center inside the cell
3. w-bbox width
4. h-bbox height
5. c-bbox confidence

$$class\ specific\ confidence\ score$$
$$= \Pr(Class_i|Object) * \Pr(Object) * IOU_{pred}^{truth}$$
$$= \Pr(Class_i) * IOU_{pred}^{truth}$$

YOLO (You only look once, CVPR'16)



7x7x2=98 boxes

Loss

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

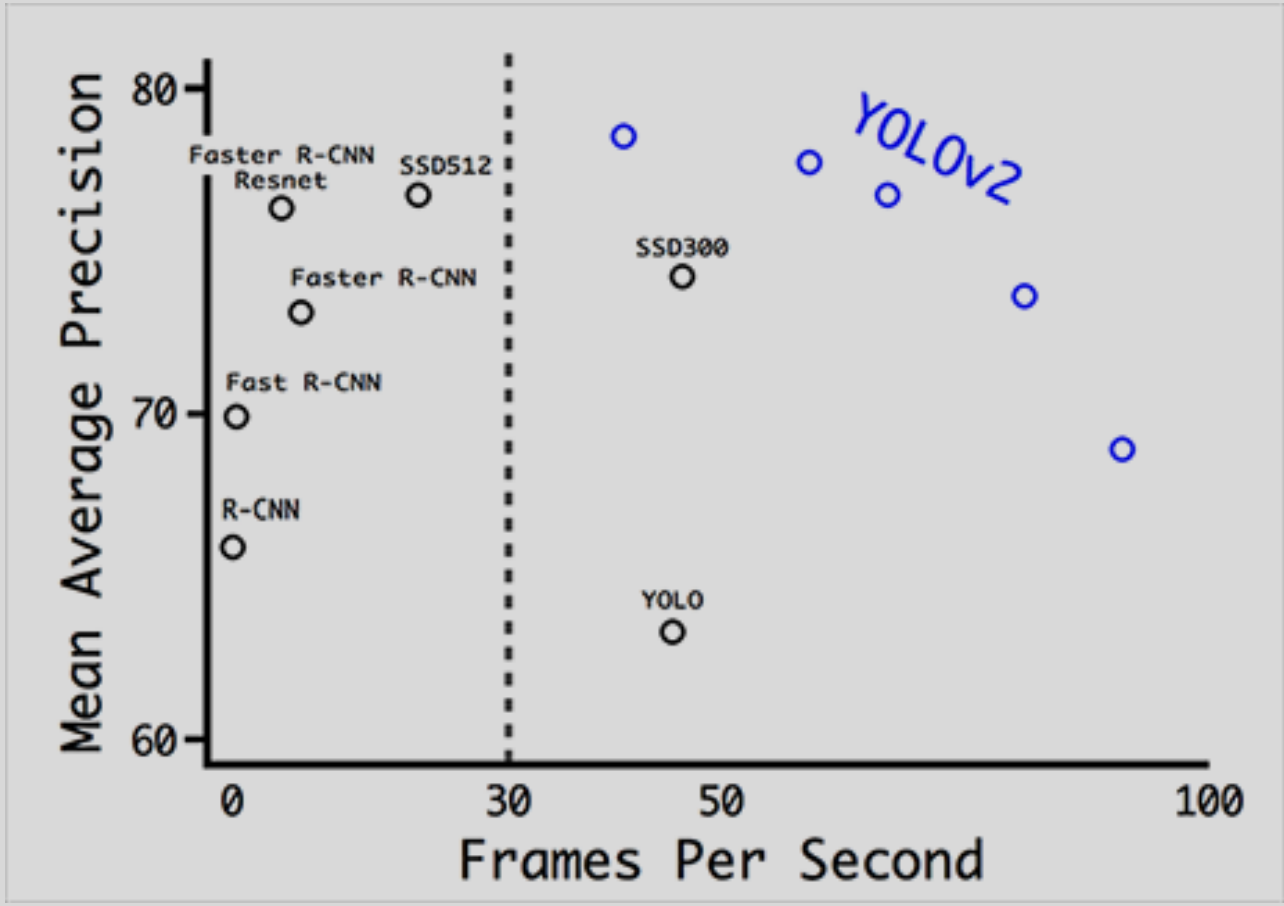
$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

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

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

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

YoloV2

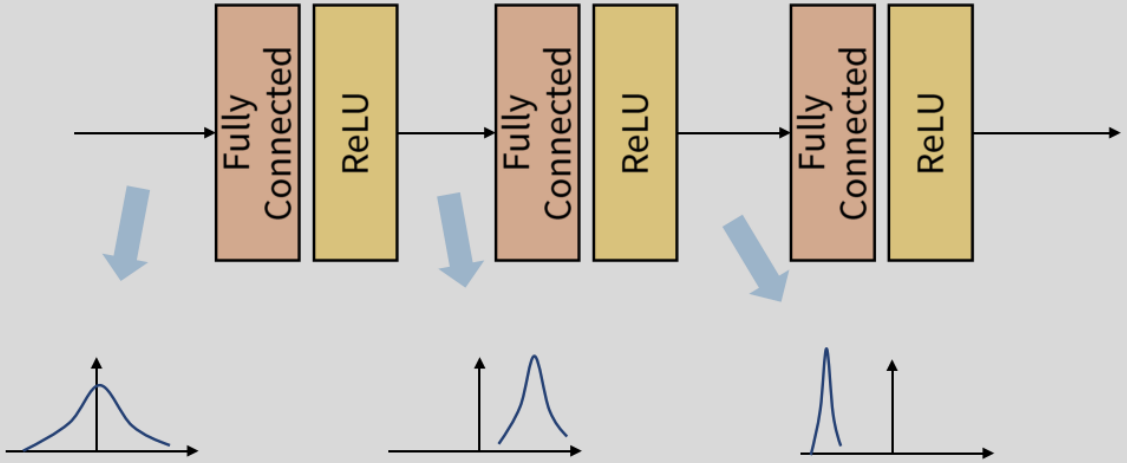

Mean Average Precision
80
70
60
0
30
50
100
Frames Per Second
Faster R-CNN
Resnet
SSD512
Faster R-CNN
Fast R-CNN
R-CNN
SSD300
YOLO
YOLOv2

YoloV2

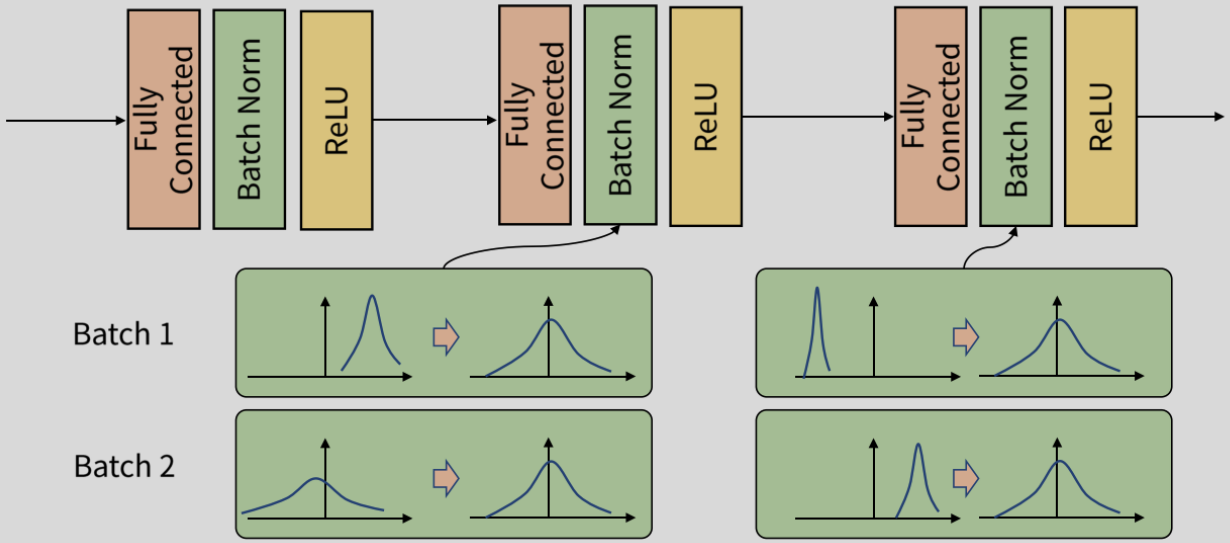
	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Batch normalization

Internal Covariate Shift:



Changes in the data distribution for each layer.



Applying the batch-normalization.

Batch normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

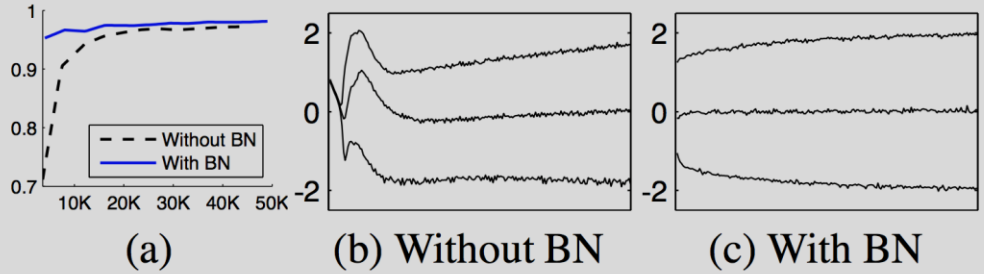
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.



(a) (b) Without BN (c) With BN

Figure 1: (a) *The test accuracy of the MNIST network trained with and without Batch Normalization, vs. the number of training steps. Batch Normalization helps the network train faster and achieve higher accuracy.* (b, c) *The evolution of input distributions to a typical sigmoid, over the course of training, shown as* $\{15, 50, 85\}$*th percentiles. Batch Normalization makes the distribution more stable and reduces the internal covariate shift.*

Hi-res classifier/detection

- YOLOv1 trains the classifier network (VGG16) at 224x224 and increases the resolution to 448 for detection.
- YOLOv2 fine-tune the classification network at the full 448x448 resolution for 10 epochs on ImageNet.

Hi-res classifier/detection

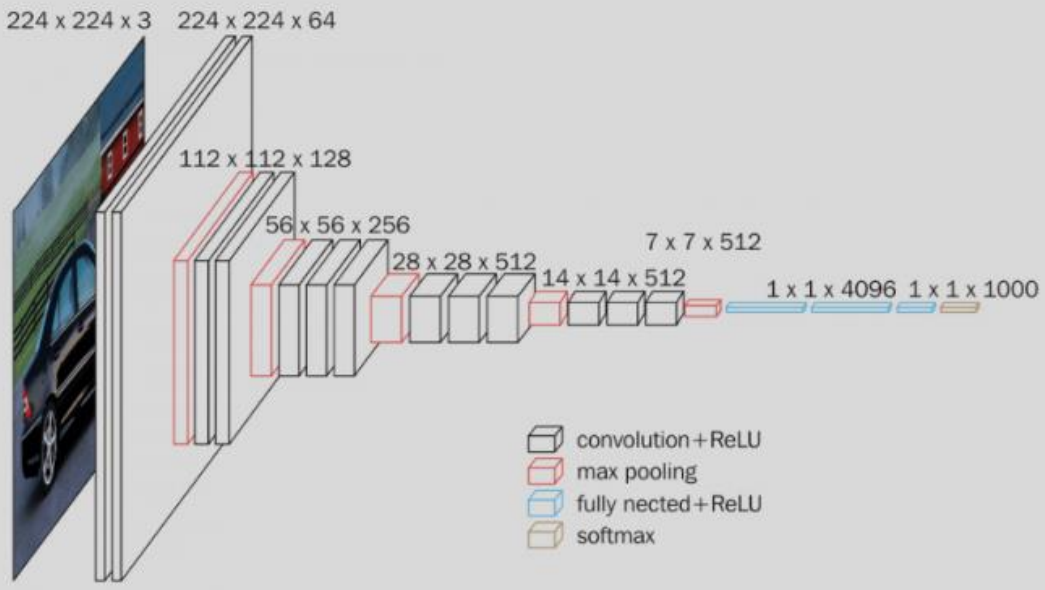
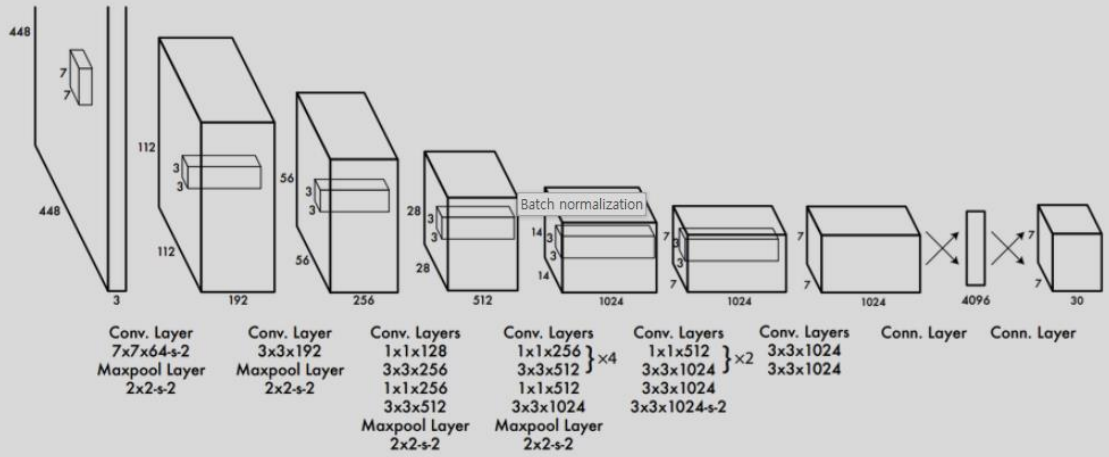


Image classification pre-trained on ImageNet (VGG-16) using 224x224 images.



VGG-16 with 448x448 input images for detection.

Architectures based on VGG16 for YOLO v1

Convolutional

- Remove FC layers from YOLO.
- Eliminate a pooling layer to make the output of the network's convolutional layers higher resolution.

New network

- Similar to VGG network, use 3x3 filters and double the channels after pooling step.
- Use batch normalization to stabilizing training.

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

Table 6: Darknet-19.

Dimension priors

- Run k-means clustering on training set bounding boxes to automatically find good priors.

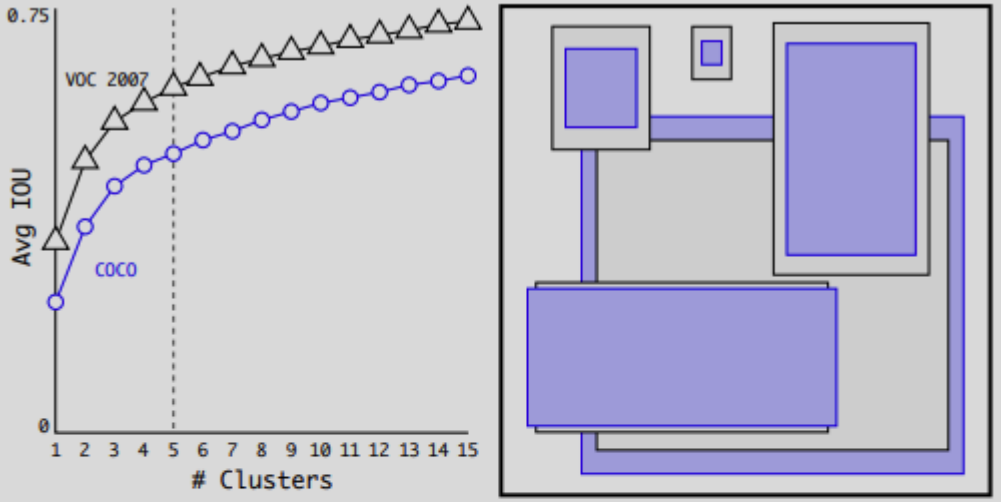


Figure 2: Clustering box dimensions on VOC and COCO. We run k-means clustering on the dimensions of bounding boxes to get good priors for our model. The left image shows the average IOU we get with various choices for k. We find that $k = 5$ gives a good tradeoff for recall vs. complexity of the model.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

Dimension priors

- Run k-means clustering on training set bounding boxes to automatically find good priors.

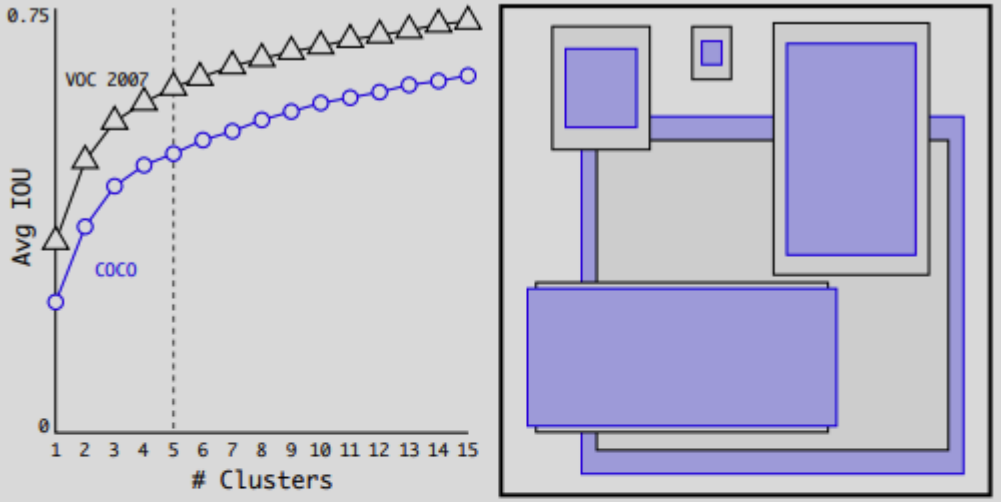
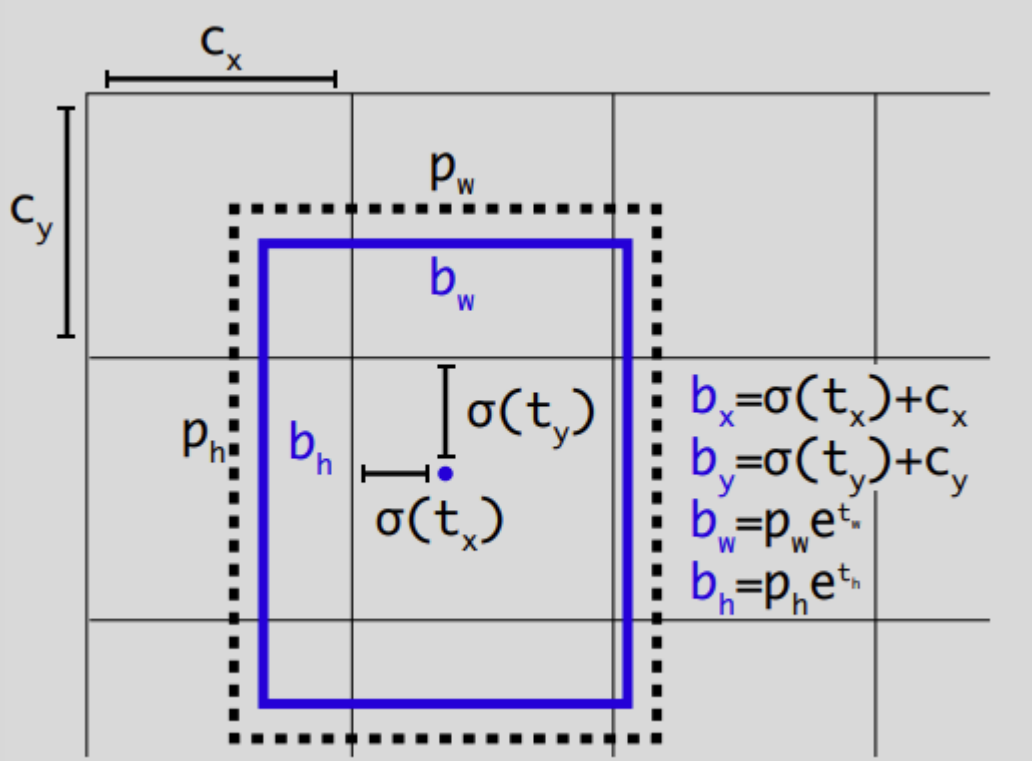


Figure 2: Clustering box dimensions on VOC and COCO. We run k-means clustering on the dimensions of bounding boxes to get good priors for our model. The left image shows the average IOU we get with various choices for k. We find that $k = 5$ gives a good tradeoff for recall vs. complexity of the model.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

Location



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

Multi-scale training

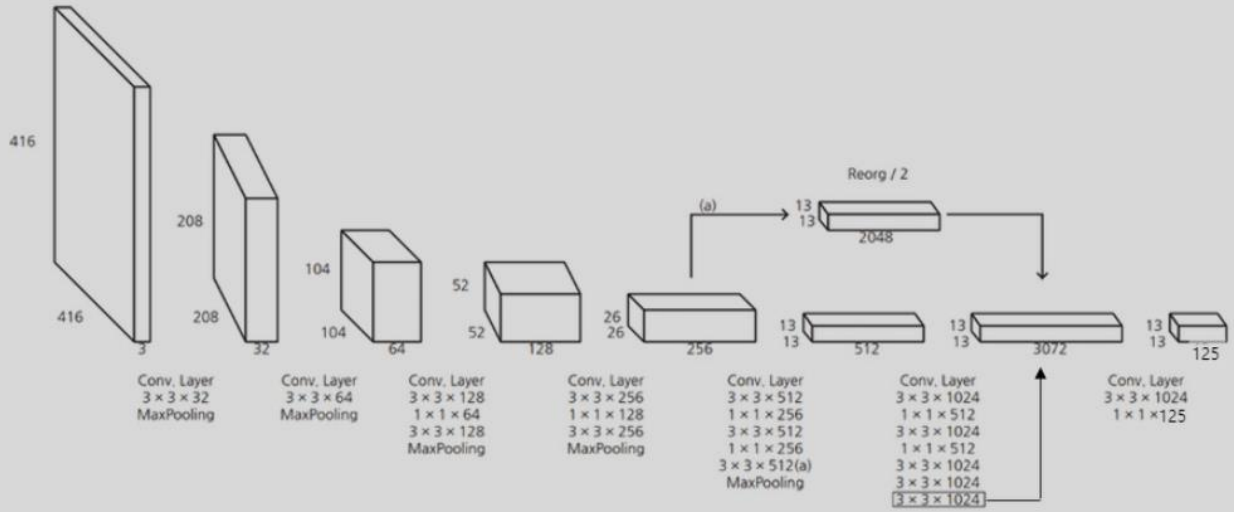
- The original YOLO uses an input resolution 448x448.
- YOLOv2 uses the resolution to 416x416.
- Instead of fixing input image size, we change the network input every 10 batches. The possible input size is: {320, 352, …, 608}.

Multi-scale training

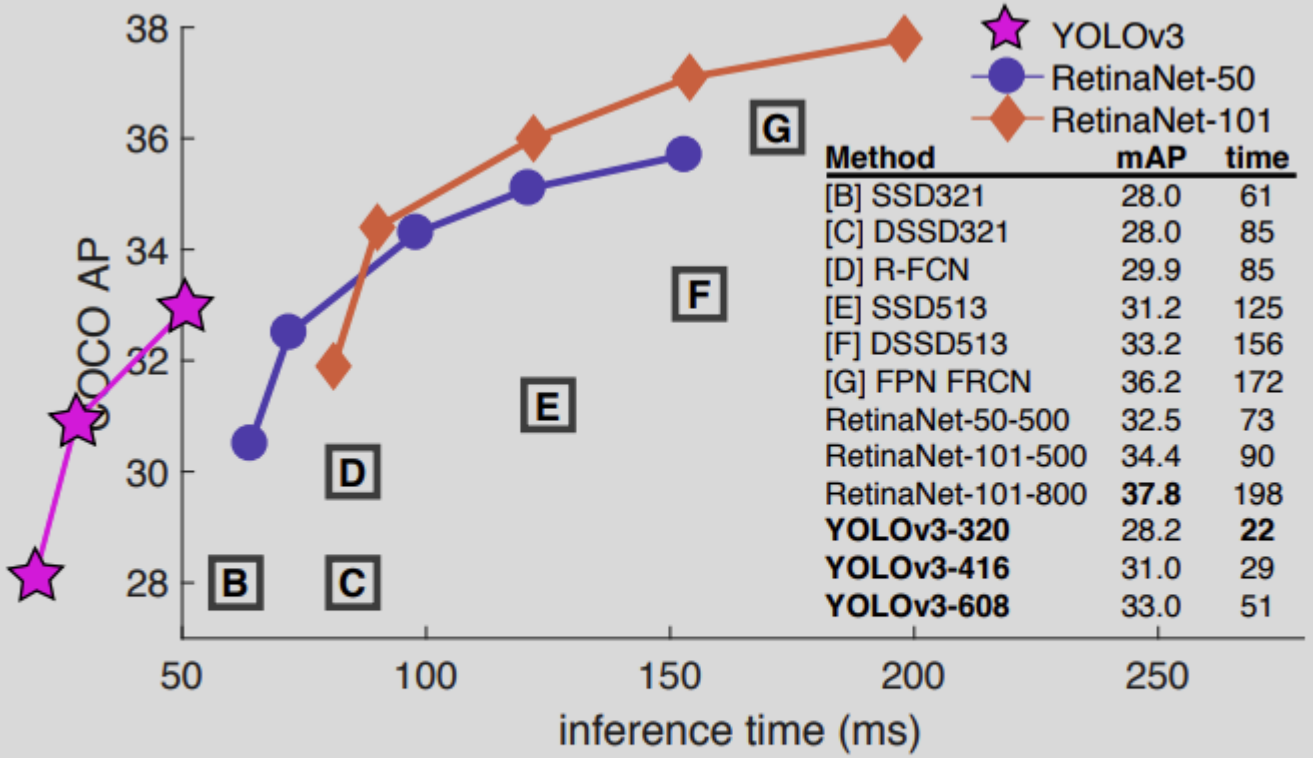
Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	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

Passthrough layer

- Passthrough layer concatenates the higher resolution features with the low resolution features.
- This simple scheme improves the 1% performance increase.



YOLOv3



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

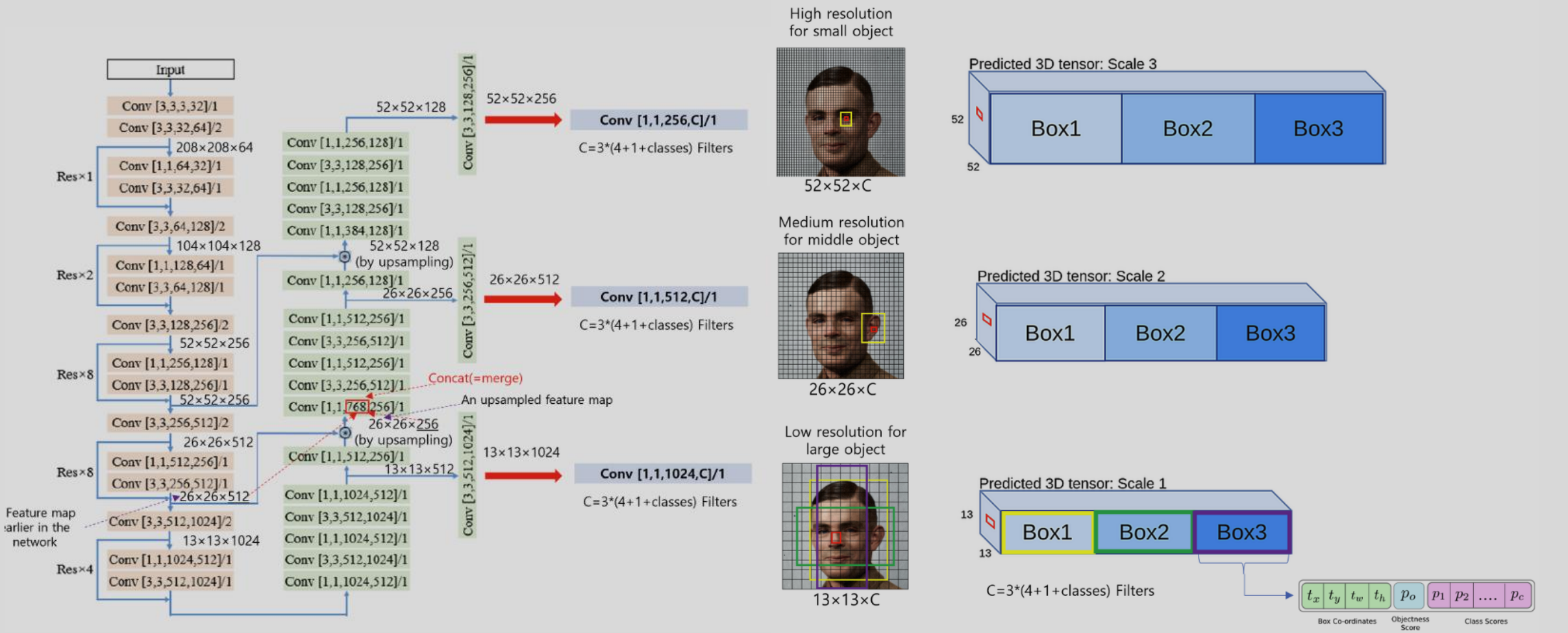
Figure 1. We adapt this figure from the Focal Loss paper [9]. YOLOv3 runs significantly faster than other detection methods with comparable performance. Times from either an M40 or Titan X, they are basically the same GPU.

Multi-scale

- YOLOv3 predicts boxes at 3 different scales:

- NxNx[3*(4+1+80)]-dimensional array is predicted for 4 bounding box offsets, 1 objectness prediction and 80 class predictions in 3 different scales.

Multi-scale


Input
Conv [3,3,3,32]/1
Conv [3,3,32,64]/2
208×208×64
Res×1
Conv [1,1,64,32]/1
Conv [3,3,32,64]/1
Conv [3,3,64,128]/2
104×104×128
Res×2
Conv [1,1,128,64]/1
Conv [3,3,64,128]/1
Conv [3,3,128,256]/2
52×52×256
Res×8
Conv [1,1,256,128]/1
Conv [3,3,128,256]/1
52×52×256
Conv [3,3,256,512]/2
26×26×512
Res×8
Conv [1,1,512,256]/1
Conv [3,3,256,512]/1
26×26×512
Feature map earlier in the network
Conv [3,3,512,1024]/2
13×13×1024
Res×4
Conv [1,1,1024,512]/1
Conv [3,3,512,1024]/1
52×52×128
Conv [1,1,256,128]/1
Conv [3,3,128,256]/1
Conv [1,1,256,128]/1
Conv [3,3,128,256]/1
Conv [1,1,384,128]/1
52×52×128
(by upsampling)
Conv [1,1,256,128]/1
26×26×256
Conv [1,1,512,256]/1
Conv [3,3,256,512]/1
Conv [1,1,512,256]/1
Conv [3,3,256,512]/1
Conv [1,1,768,256]/1
Concat(=merge)
An upsampled feature map
26×26×256
(by upsampling)
Conv [1,1,512,256]/1
13×13×512
Conv [1,1,1024,512]/1
Conv [3,3,512,1024]/1
Conv [1,1,1024,512]/1
Conv [3,3,512,1024]/1
Conv [1,1,1024,512]/1
Conv [3,3,128,256]/1
52×52×256
Conv [1,1,256,C]/1
C=3*(4+1+classes) Filters
Conv [3,3,256,512]/1
26×26×512
Conv [1,1,512,C]/1
C=3*(4+1+classes) Filters
Conv [3,3,512,1024]/1
13×13×1024
Conv [1,1,1024,C]/1
C=3*(4+1+classes) Filters
High resolution for small object
52×52×C
Medium resolution for middle object
26×26×C
Low resolution for large object
13×13×C
Predicted 3D tensor: Scale 3
52
52
Box1
Box2
Box3
Predicted 3D tensor: Scale 2
26
26
Box1
Box2
Box3
Predicted 3D tensor: Scale 1
13
13
Box1
Box2
Box3
C=3*(4+1+classes) Filters
t_x t_y t_w t_h
p_o
p_1 p_2 .... p_c
Box Co-ordinates
Objectness Score
Class Scores

DarkNet-53

- Powerful than Darknet-19 but still more efficient than ResNet-101 or ResNet-152.

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

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

Table 1. **Darknet-53.**


Thank you!
ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY
2 0 0 7