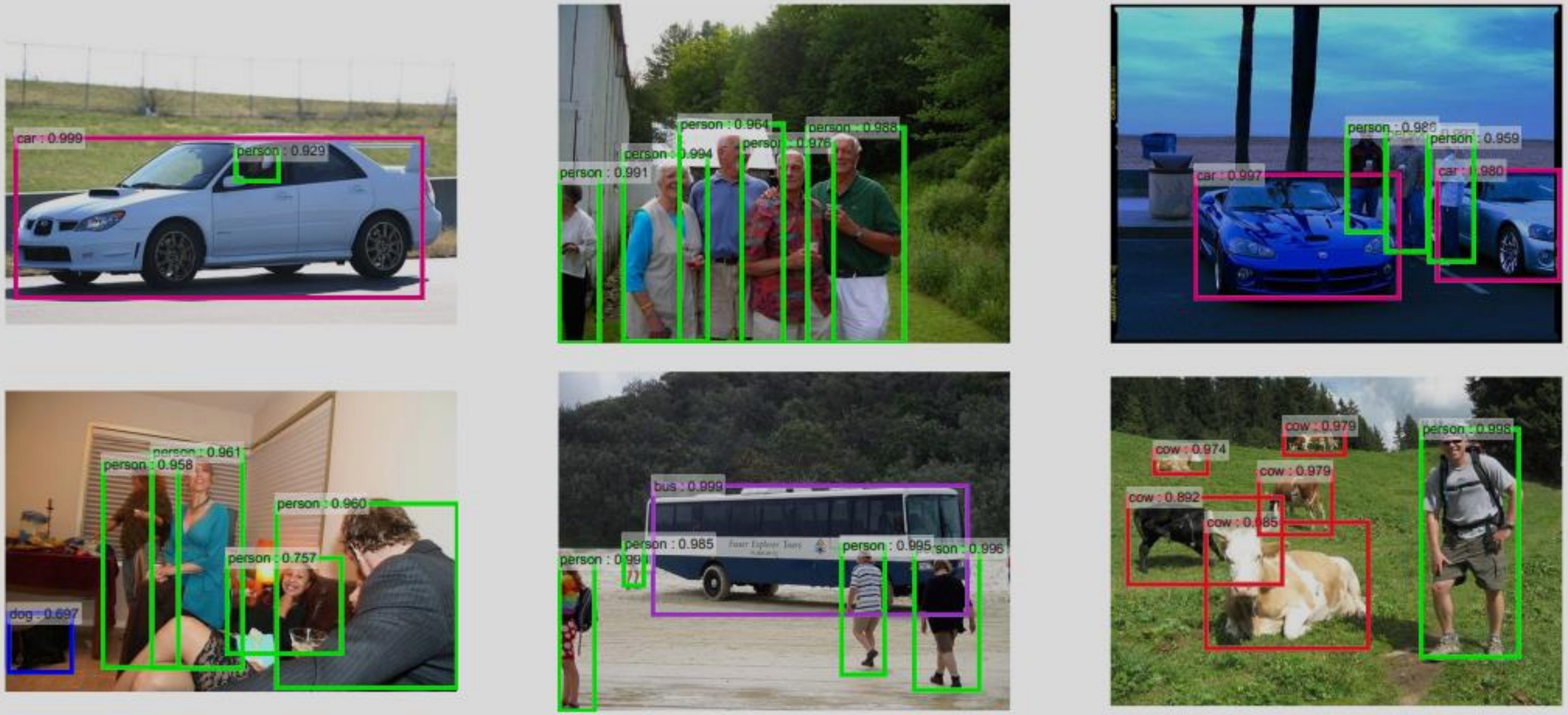
Computer Vision

Lecture 04: Object detection pipeline - 1

Computer vision applications

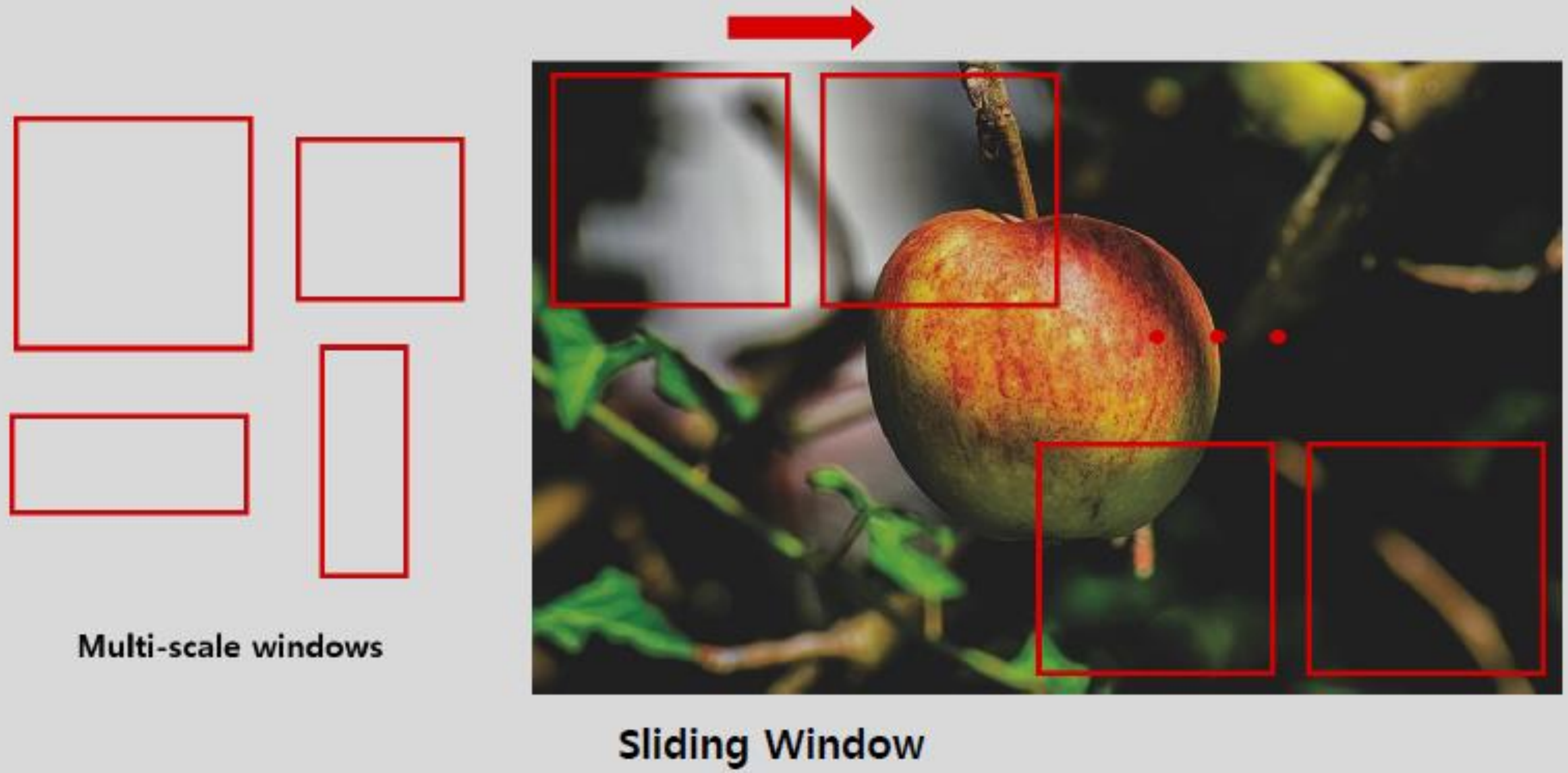


Detecting object locations. [Faster-RCNN NIPS’15]

Computer vision applications

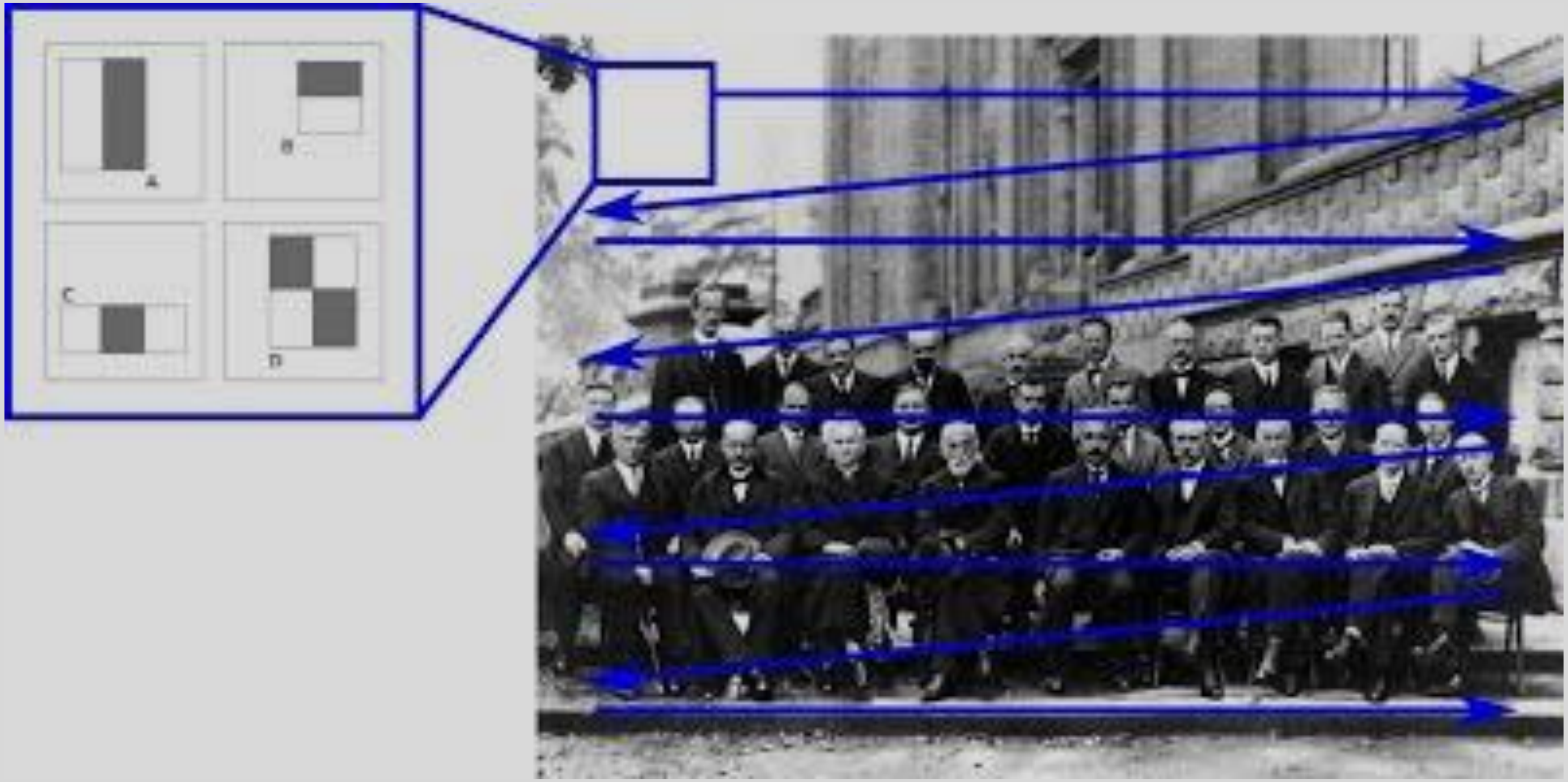
Detecting object locations and segmentation. [Mask RCNN ICCV’17]

Object detection



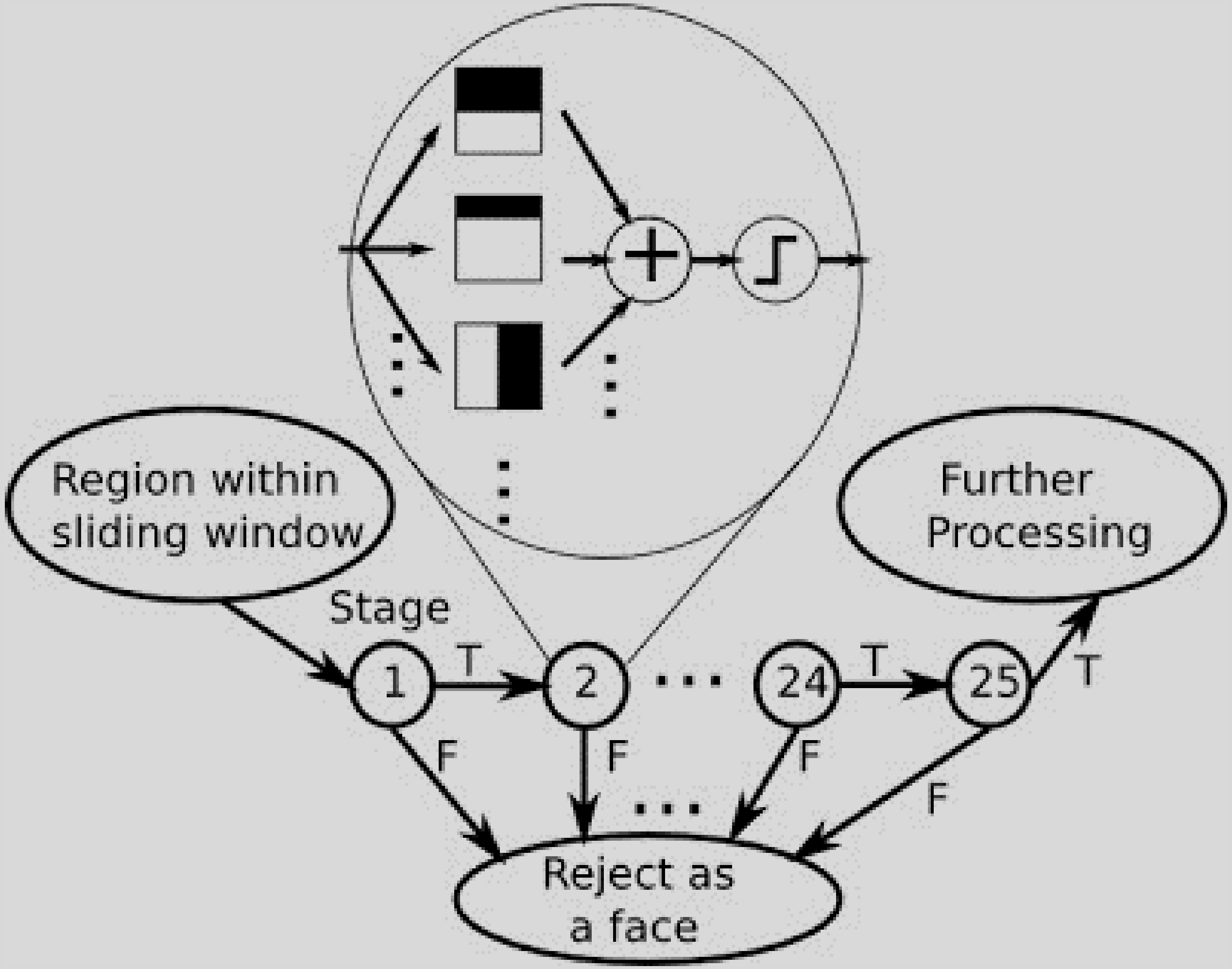
Considering the multi-scales, sliding window is too slow.

Face detection (2001)



Viola & Jones, CVPR'01

Face detection (2001)

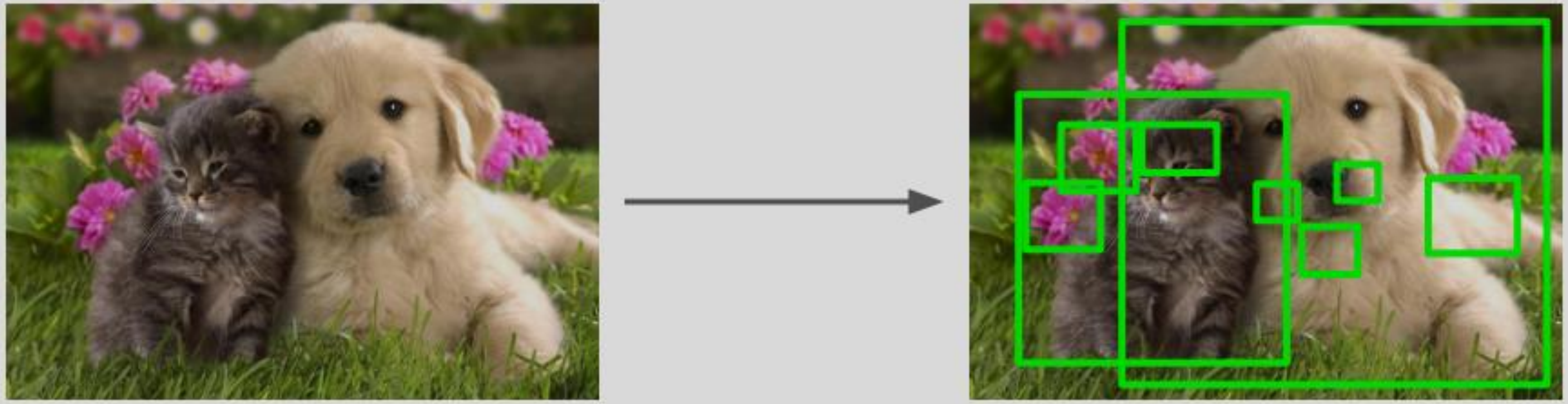


Viola & Jones, CVPR'01

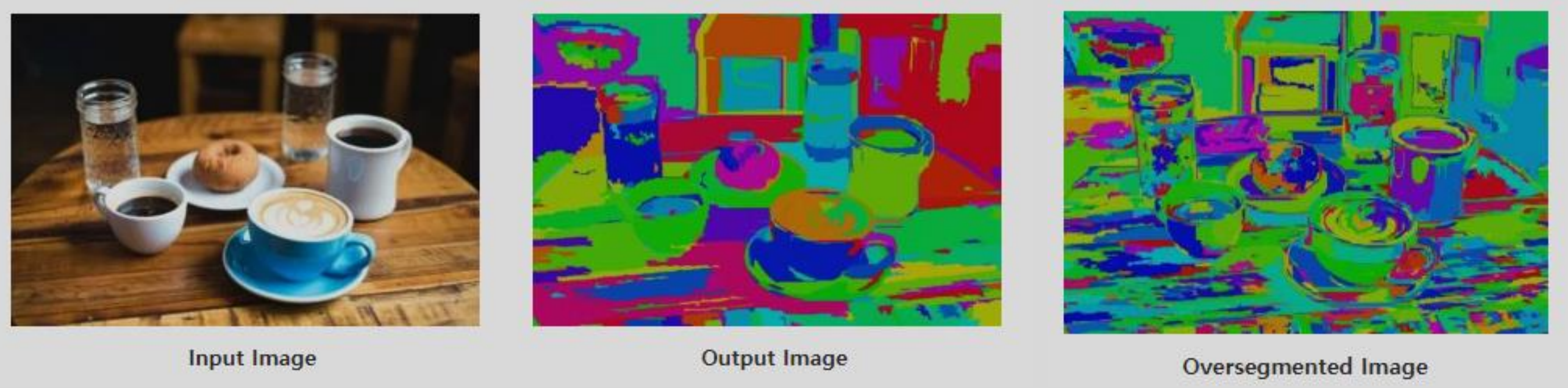
Object proposal

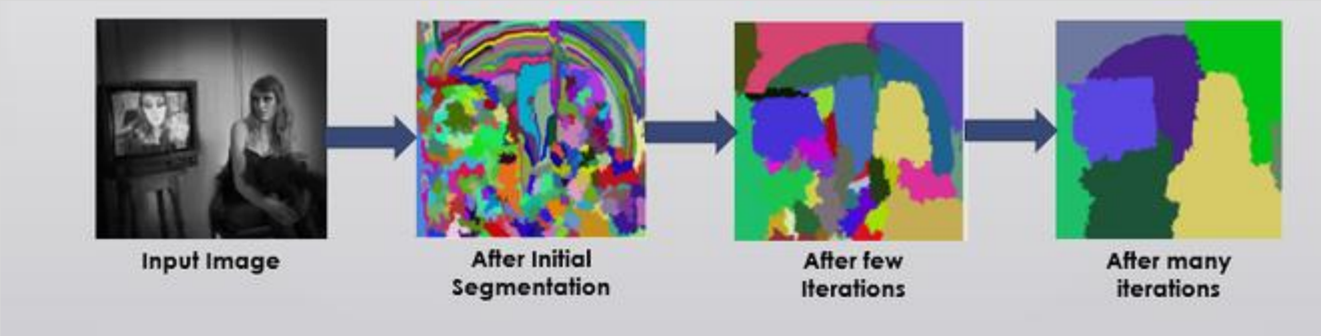
Find image regions that are likely to contain objects.

E.g. Selective search. (1000 regions in a few seconds on CPU).



Object proposal





Considering Color, Texture, Size, Shape similarities.

Object proposal

Algorithm 1: Hierarchical Grouping Algorithm

Input: (colour) image
Output: Set of object location hypotheses L

Obtain initial regions $R = \{r_1, \cdots, r_n\}$ using [13]
Initialise similarity set $S = \emptyset$
foreach *Neighbouring region pair* (r_i, r_j) **do**
  Calculate similarity $s(r_i, r_j)$
  $S = S \cup s(r_i, r_j)$

while $S \neq \emptyset$ **do**
  Get highest similarity $s(r_i, r_j) = \max(S)$
  Merge corresponding regions $r_t = r_i \cup r_j$
  Remove similarities regarding $r_i : S = S \setminus s(r_i, r_*)$
  Remove similarities regarding $r_j : S = S \setminus s(r_*, r_j)$
  Calculate similarity set S_t between r_t and its neighbours
  $S = S \cup S_t$
  $R = R \cup r_t$

Extract object location boxes L from all regions in R

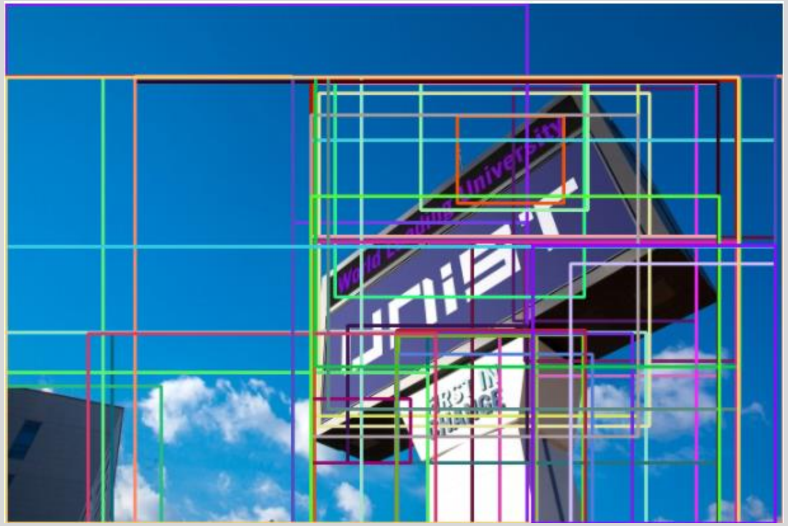
Object proposal

```
import cv2
from google.colab.patches import cv2_imshow
import random

image = cv2.imread("/content/unist.jpg")

ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
ss.setBaseImage(image)
ss.switchToSelectiveSearchFast()
rects = ss.process()

for i in range(0, len(rects), 100):
  output = image.copy()
  for (x, y, w, h) in rects[i:i + 100]:
    color = [random.randint(0, 255) for j in range(0, 3)]
    cv2.rectangle(output, (x, y), (x + w, y + h), color, 2)
cv2_imshow(output)
```

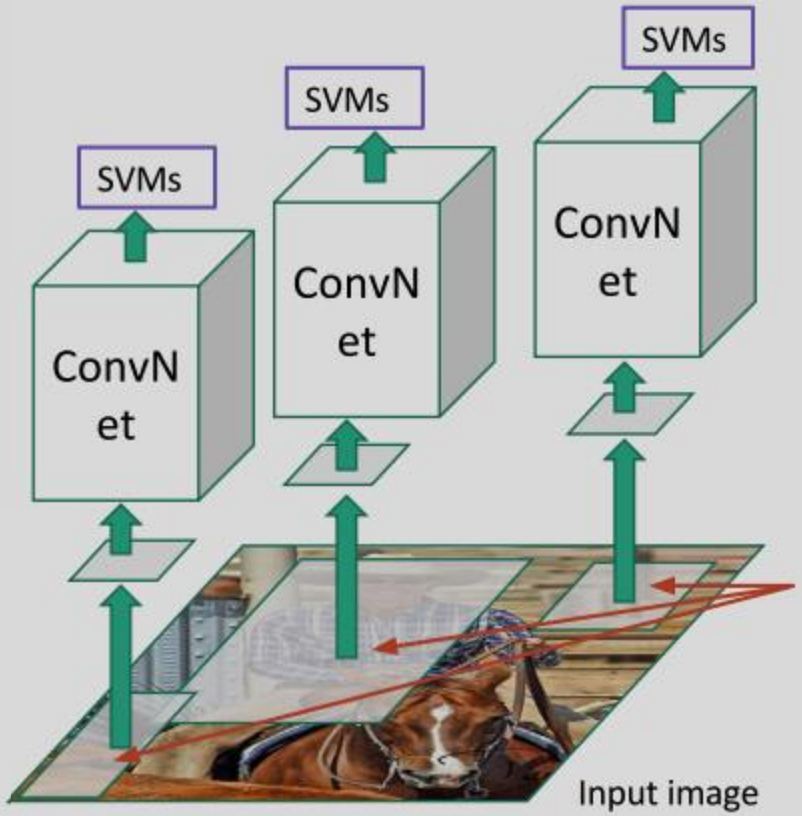


R-CNN (CVPR'13)

R-CNN (CVPR'13)

Regions of interest
From selective search (~2000).

R-CNN (CVPR'13)



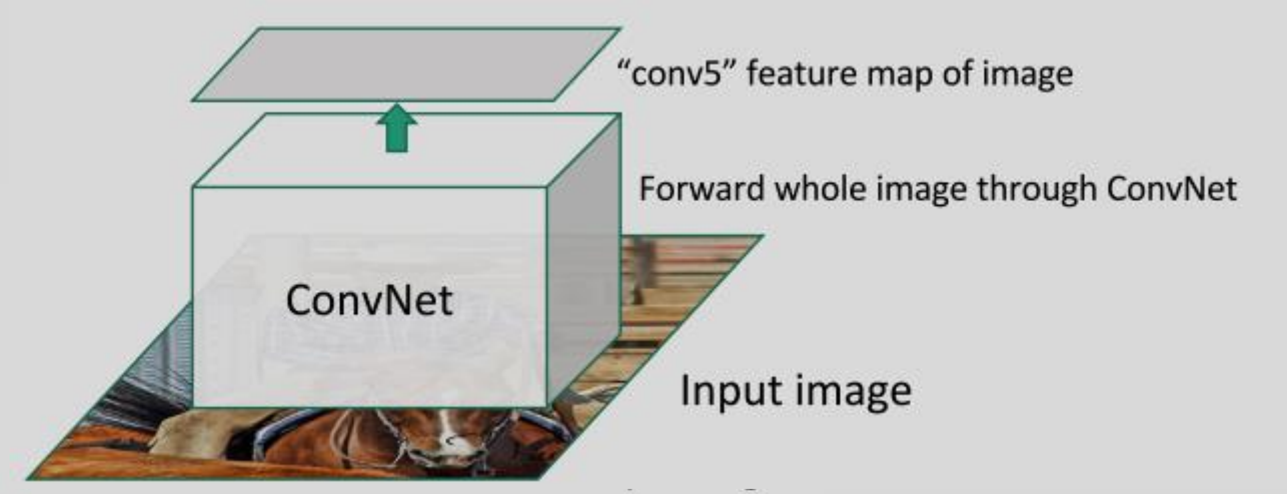
Classify each region with SVMs.

Forward with ImageNet-trained CNNs.

Regions of interest
From selective search (~2000).

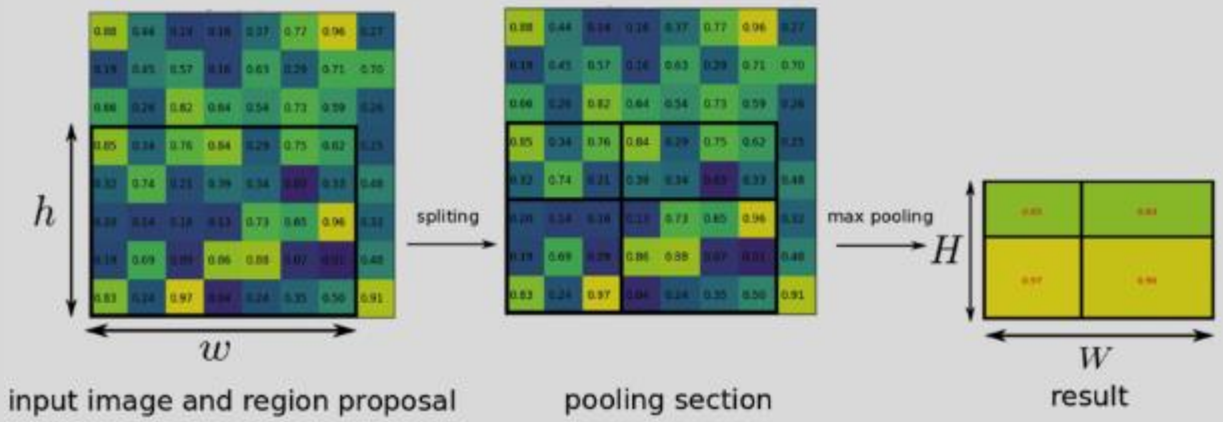
Limitations:
1) Not end-to-end trainable.
2) Still slow.

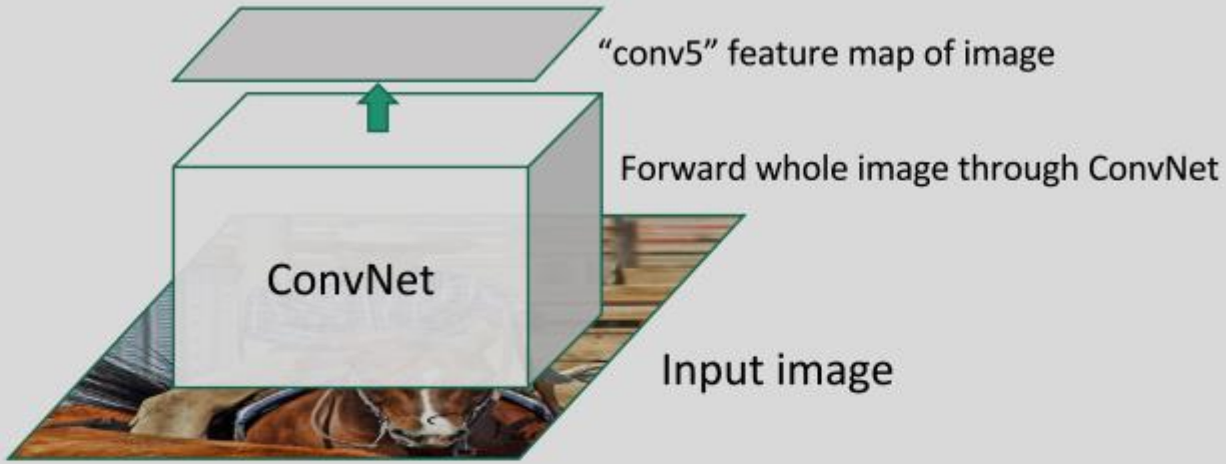
Fast R-CNN (ICCV'15)



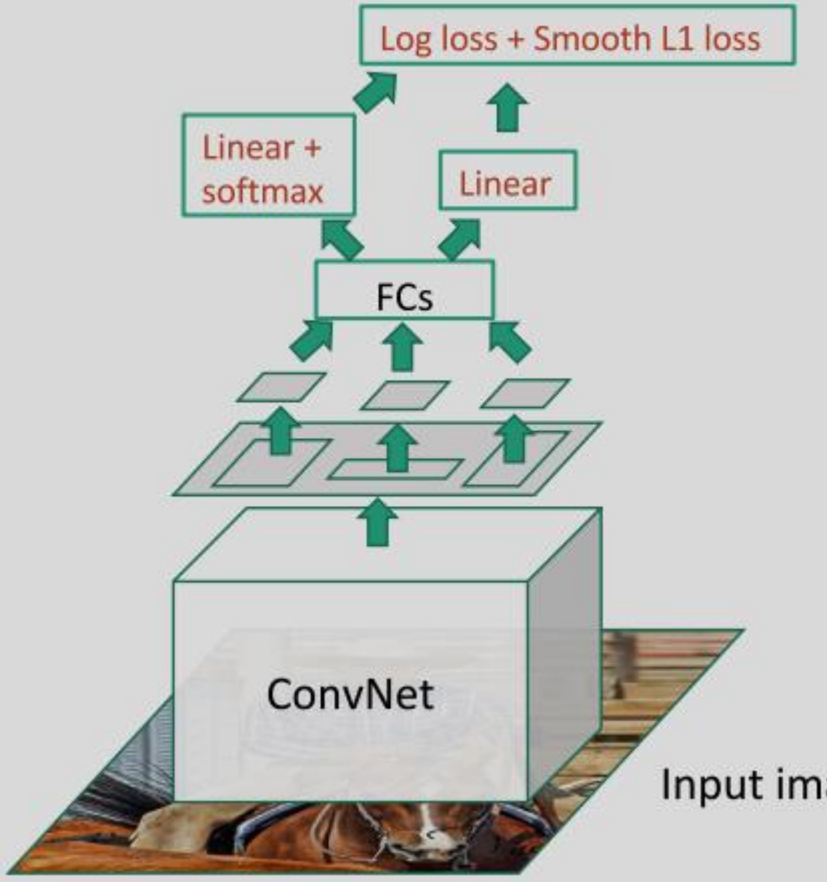
Fast R-CNN (ICCV’15)

RoI Pooling





Fast R-CNN (ICCV'15)



$$L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda[u \geq 1] L_{\text{loc}}(t^u, v),$$

$$p = (p_0, \ldots, p_K),$$

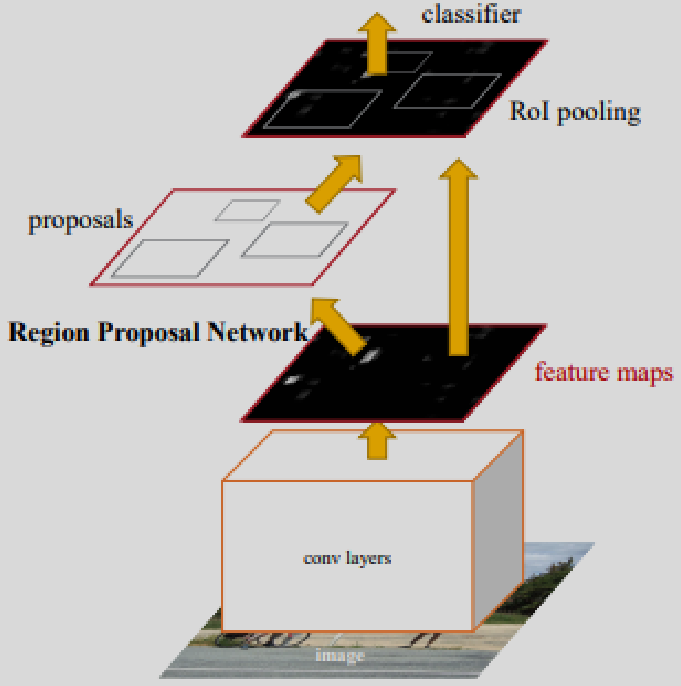
$$t^u = (t^u_{\text{x}}, t^u_{\text{y}}, t^u_{\text{w}}, t^u_{\text{h}})$$

$$L_{\text{cls}}(p, u) = -\log p_u$$

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

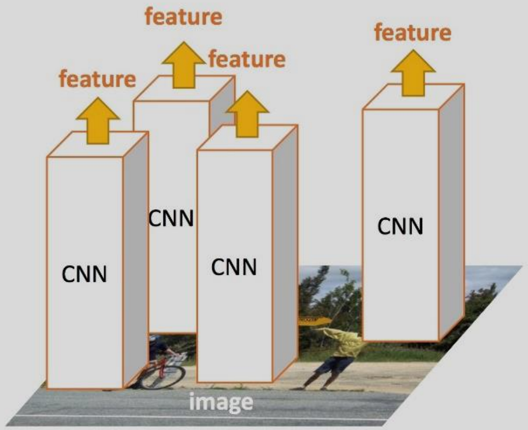
$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

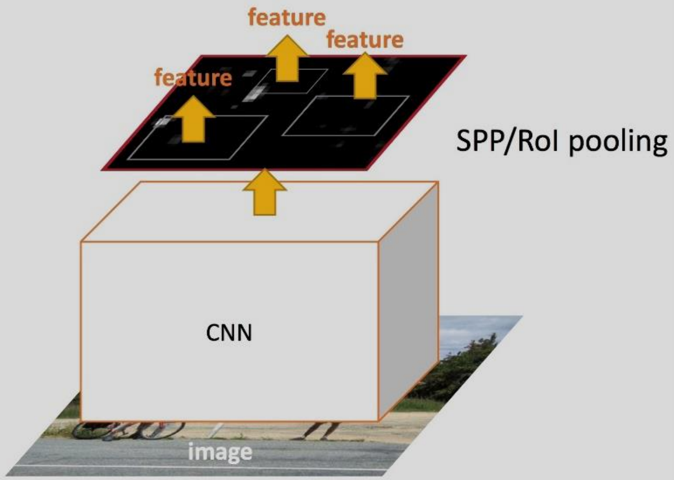
Faster R-CNN (NIPS'15)

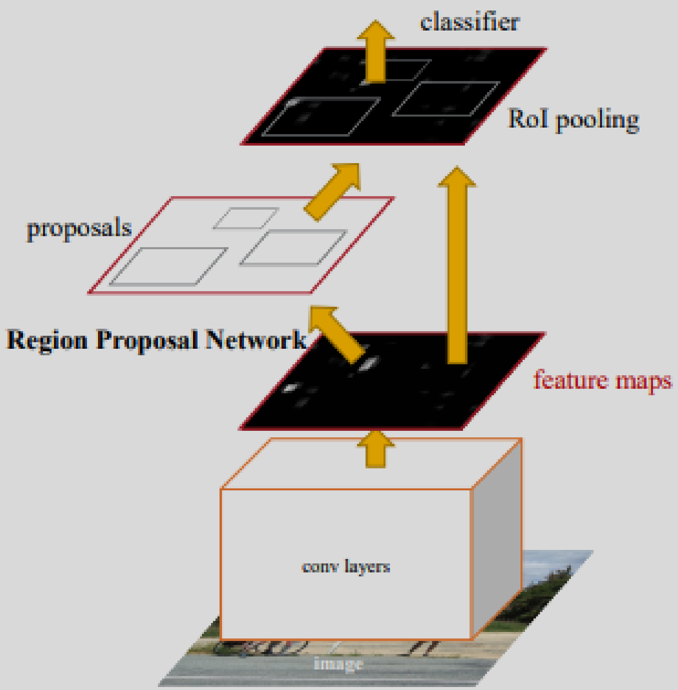


Solve the bottleneck in the region proposal of the Fast-RCNN

Comparisons







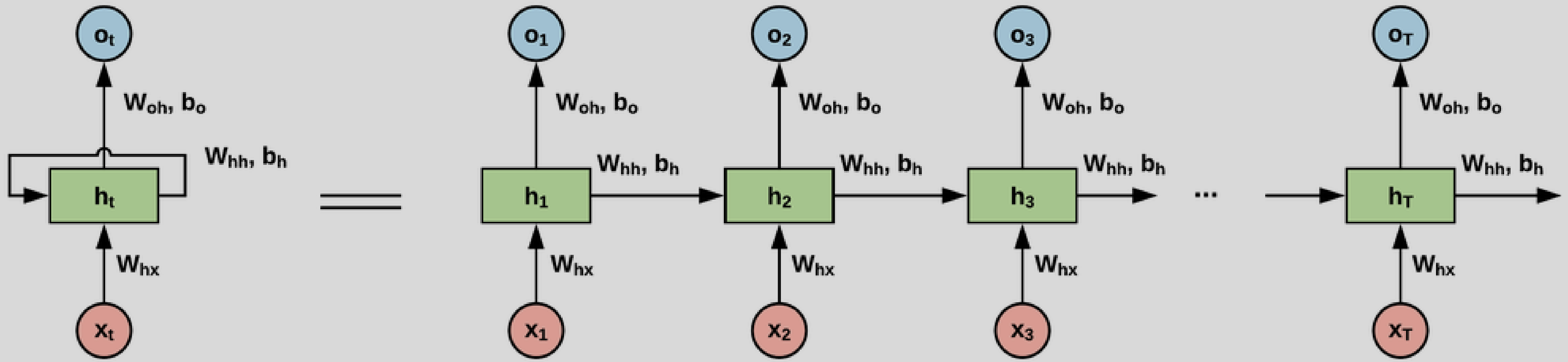
R-CNN
- Extract image regions
- 1 CNN per region (2000 CNNs)
- Classify region-based features

SPP-net & Fast R-CNN (the same forward pipeline)
- 1 CNN on the entire image
- Extract features from feature map regions
- Classify region-based features

System	Time	07 data	07 + 12 data
R-CNN	~ 50s	66.0	-
Fast R-CNN	~ 2s	66.9	70.0
Faster R-CNN	**~ 198ms**	**69.9**	**73.2**

Detection mAP on PASCAL VOC 2007 and 2012, with VGG-16 pre-trained on ImageNet Dataset

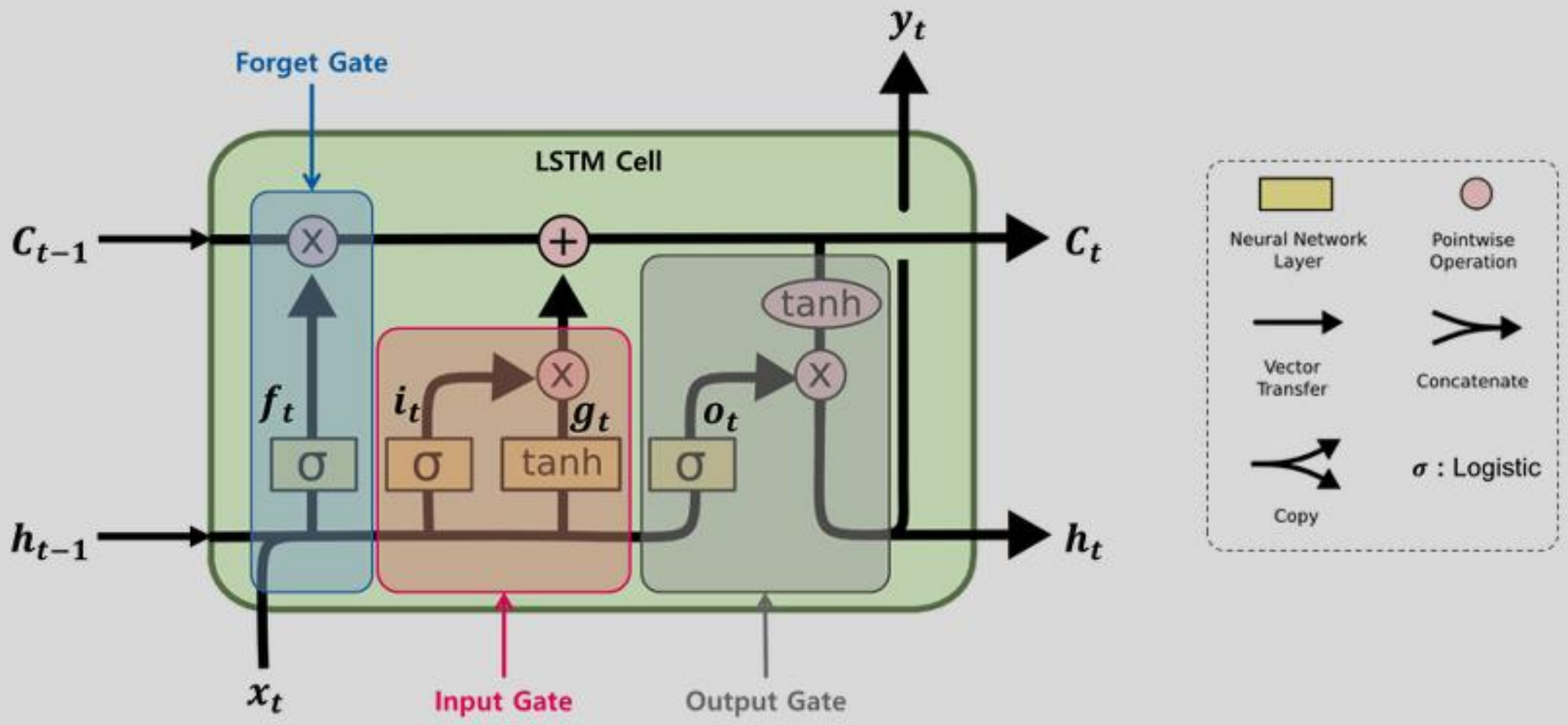
Recurrent Neural Network



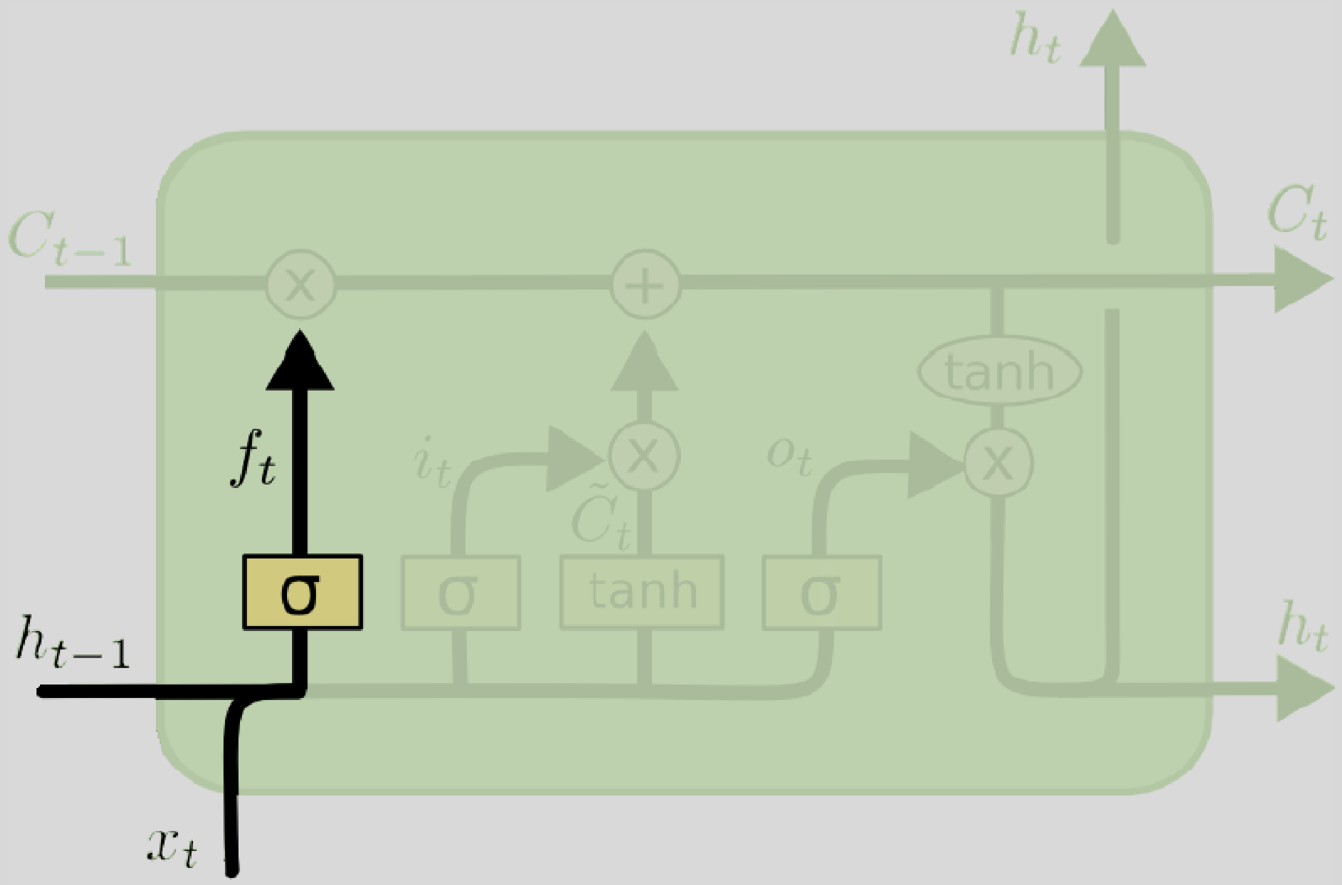
Limitations of RNN

- Non-trivial to parallelize
- Vanishing gradient

Long Term Short Memory (LSTM)

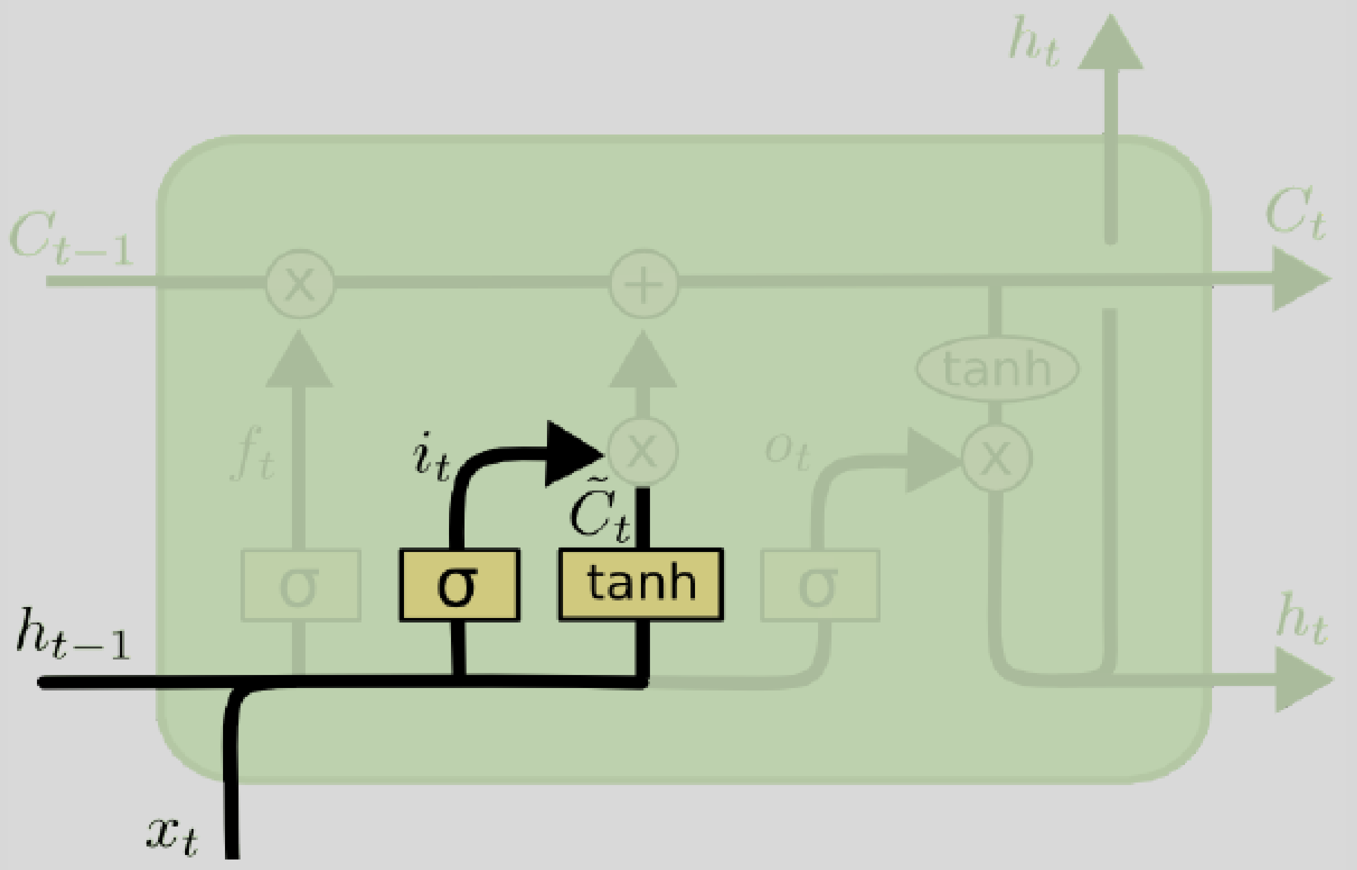


Long Term Short Memory (LSTM)



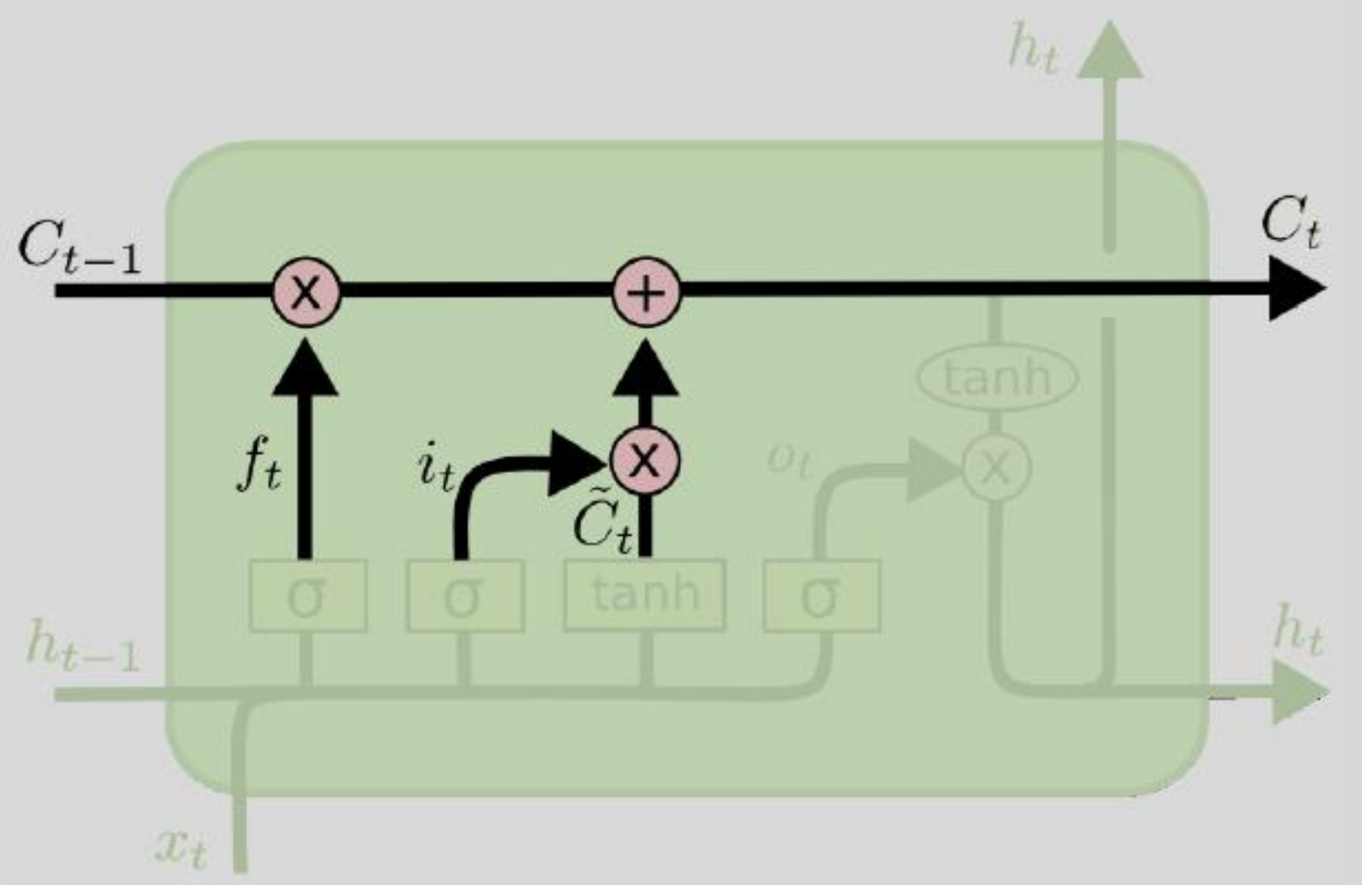
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Long Term Short Memory (LSTM)



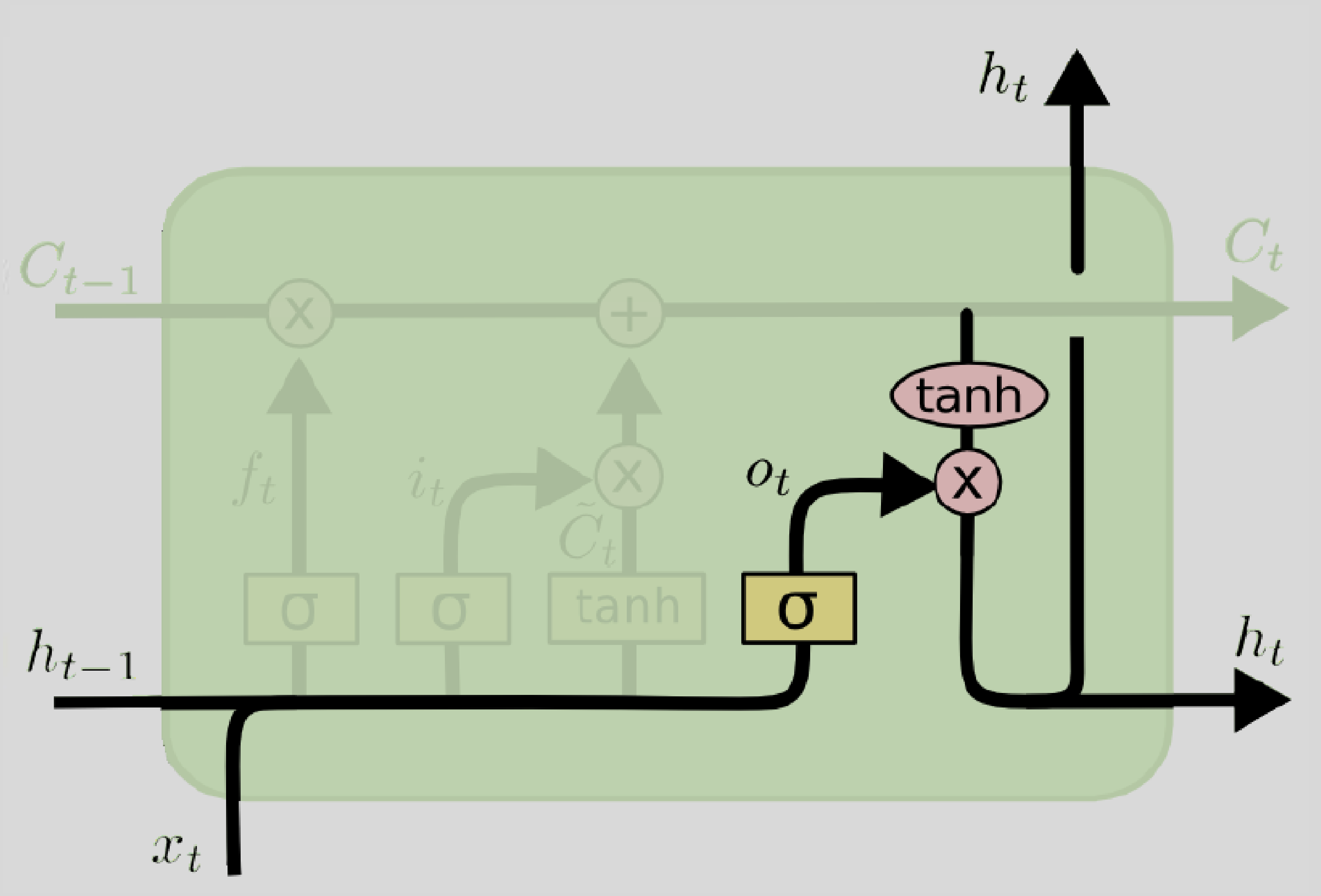
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C} = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Long Term Short Memory (LSTM)



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

Long Term Short Memory (LSTM)



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

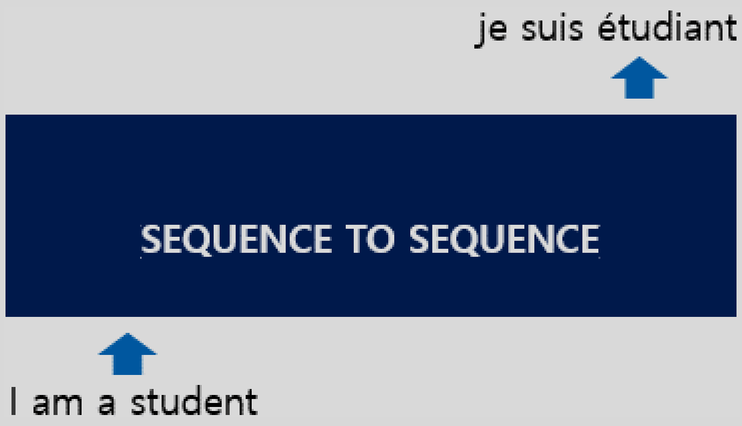
LSTM

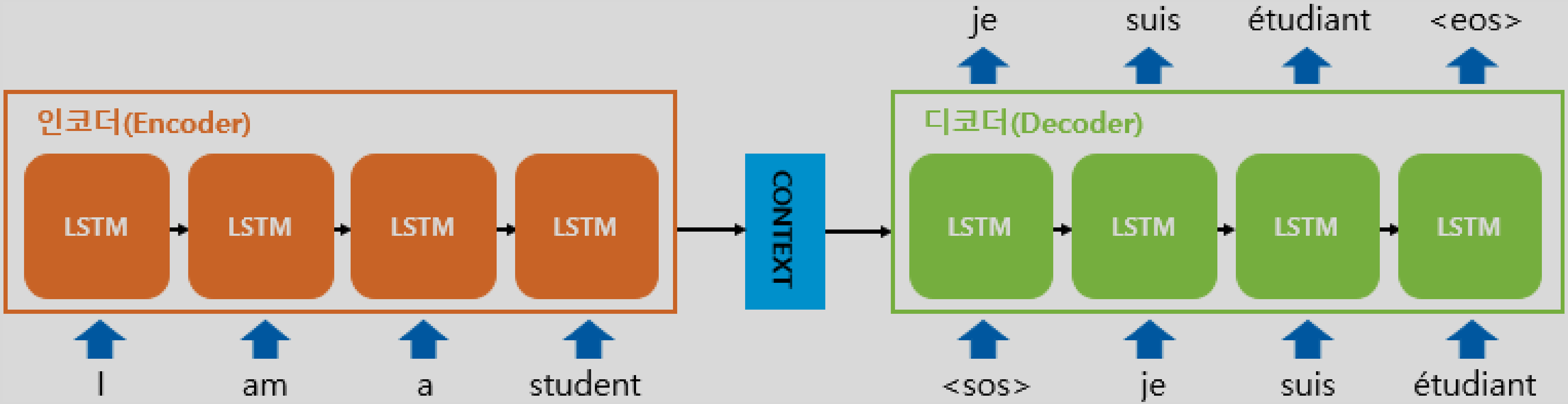
- Forget gate: How much retain past info.
- Input gate: How much use the new info.
- Output gate: How much use the new output.

- Become robust by learning how to forget and retain.

Seq2seq using LSTMs

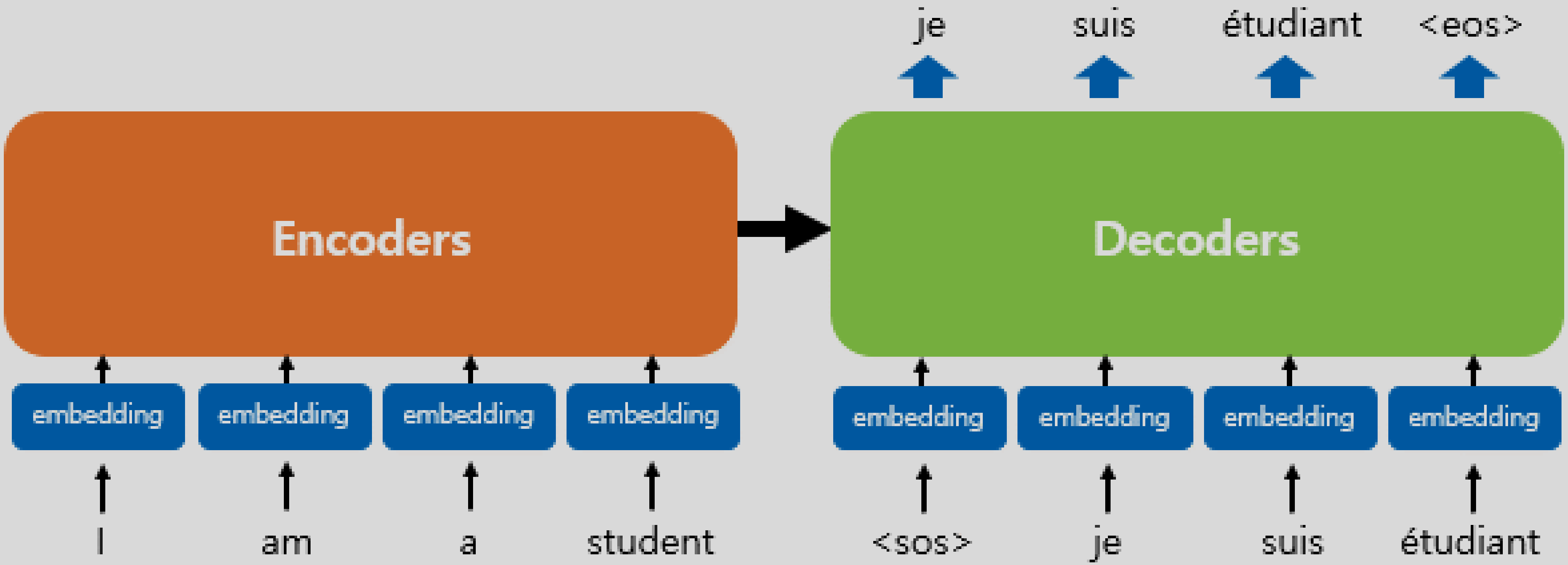
https://wikidocs.net/31379





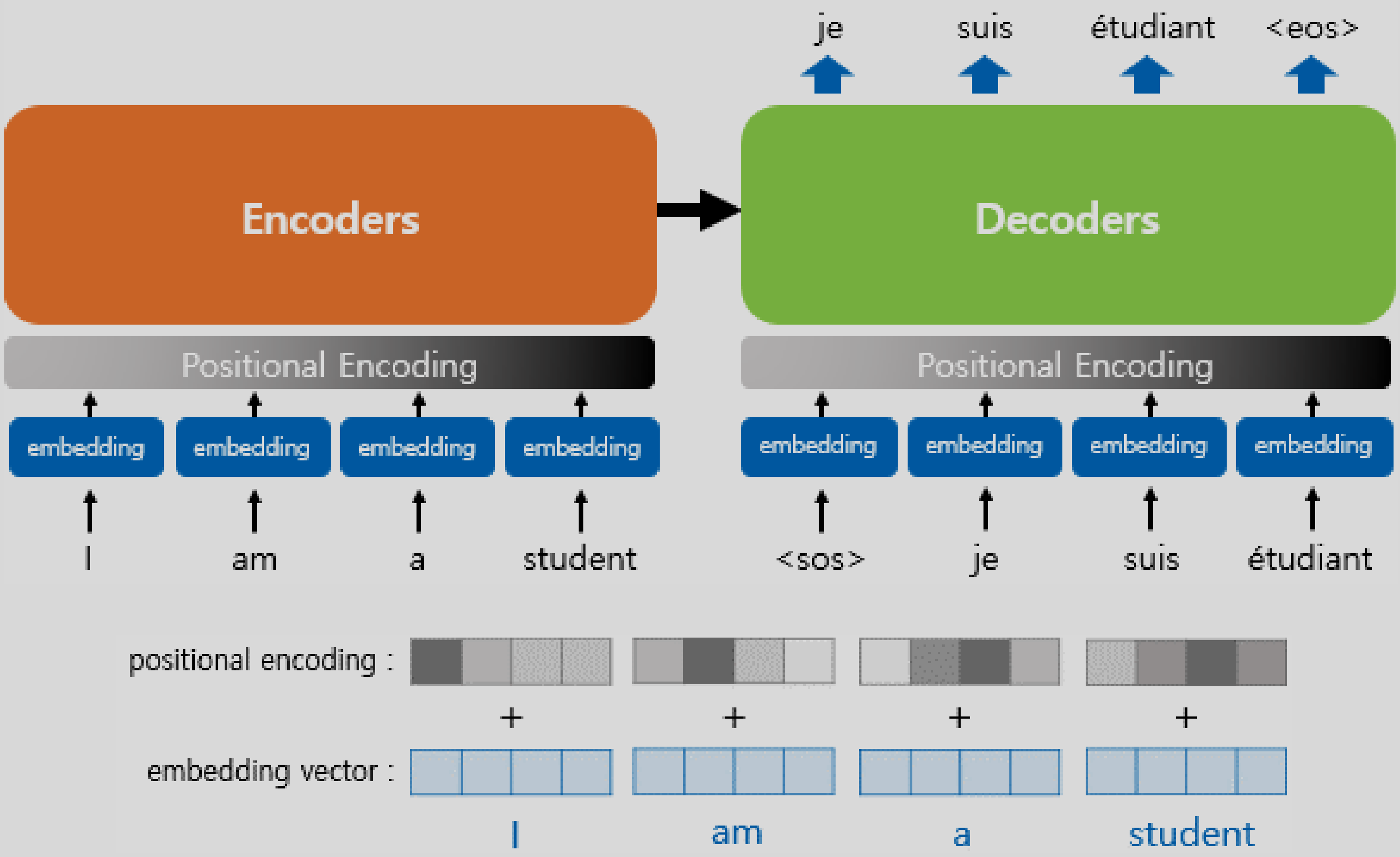
Transformer

https://wikidocs.net/31379



Transformer

https://wikidocs.net/31379



Transformer

https://wikidocs.net/31379

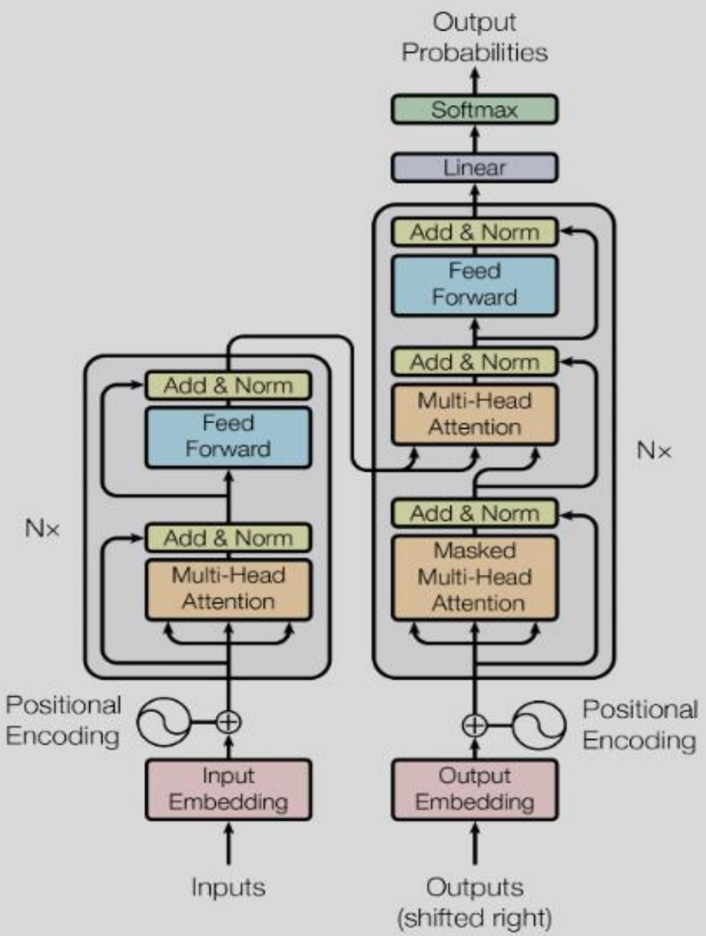
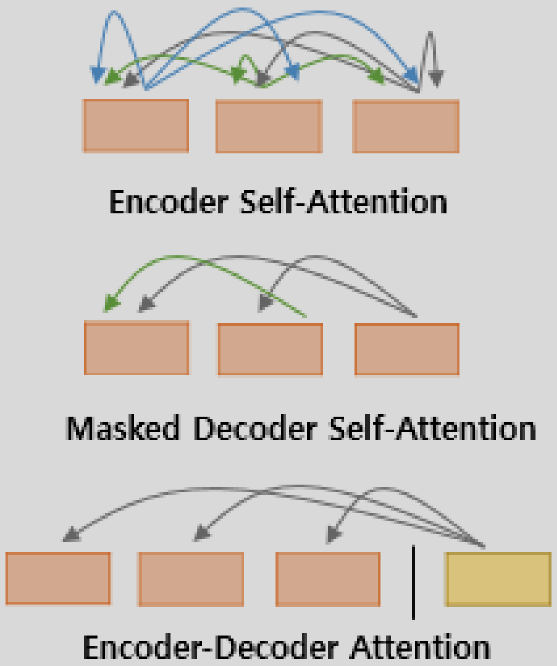


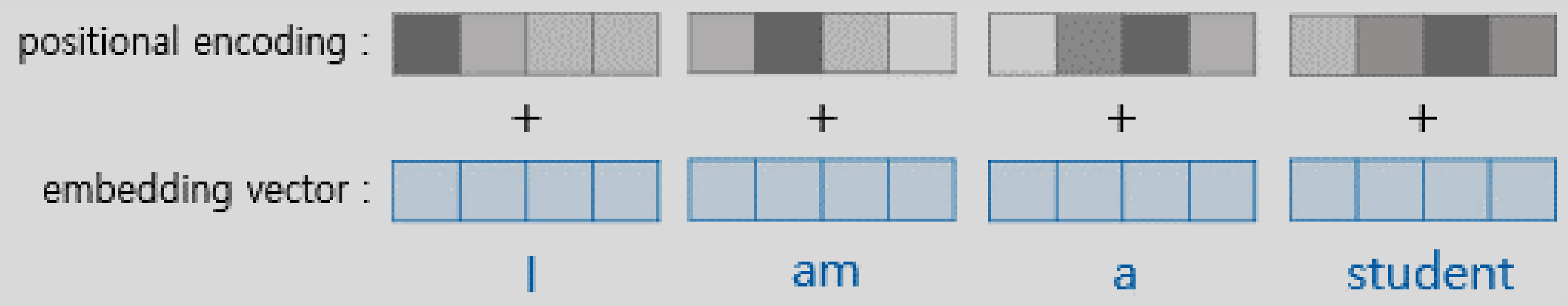
Figure 1: The Transformer - model architecture.

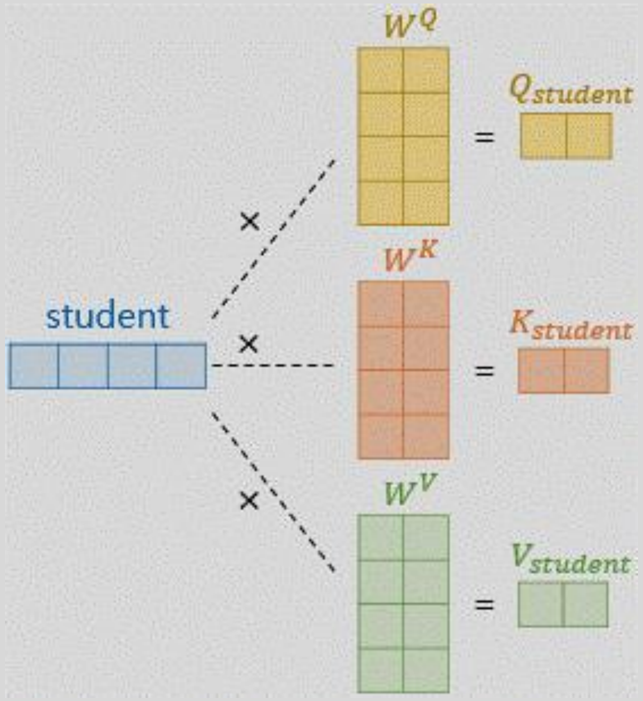


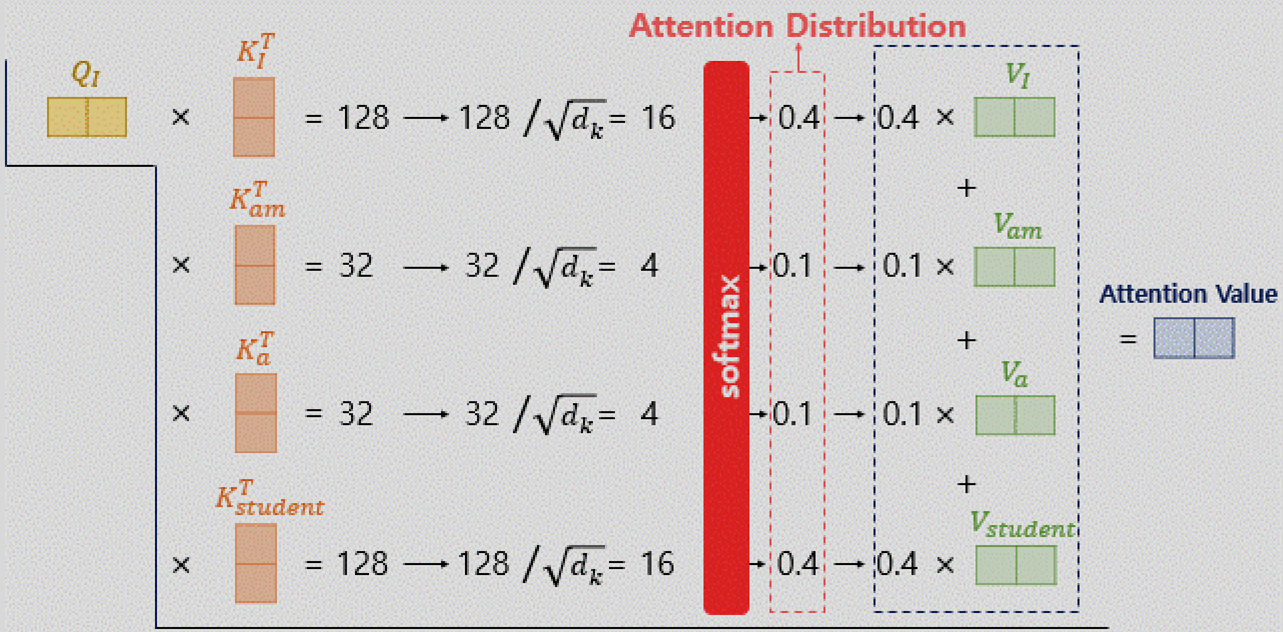
Attention is all you need, NIPS, 2017

Transformer

https://wikidocs.net/31379

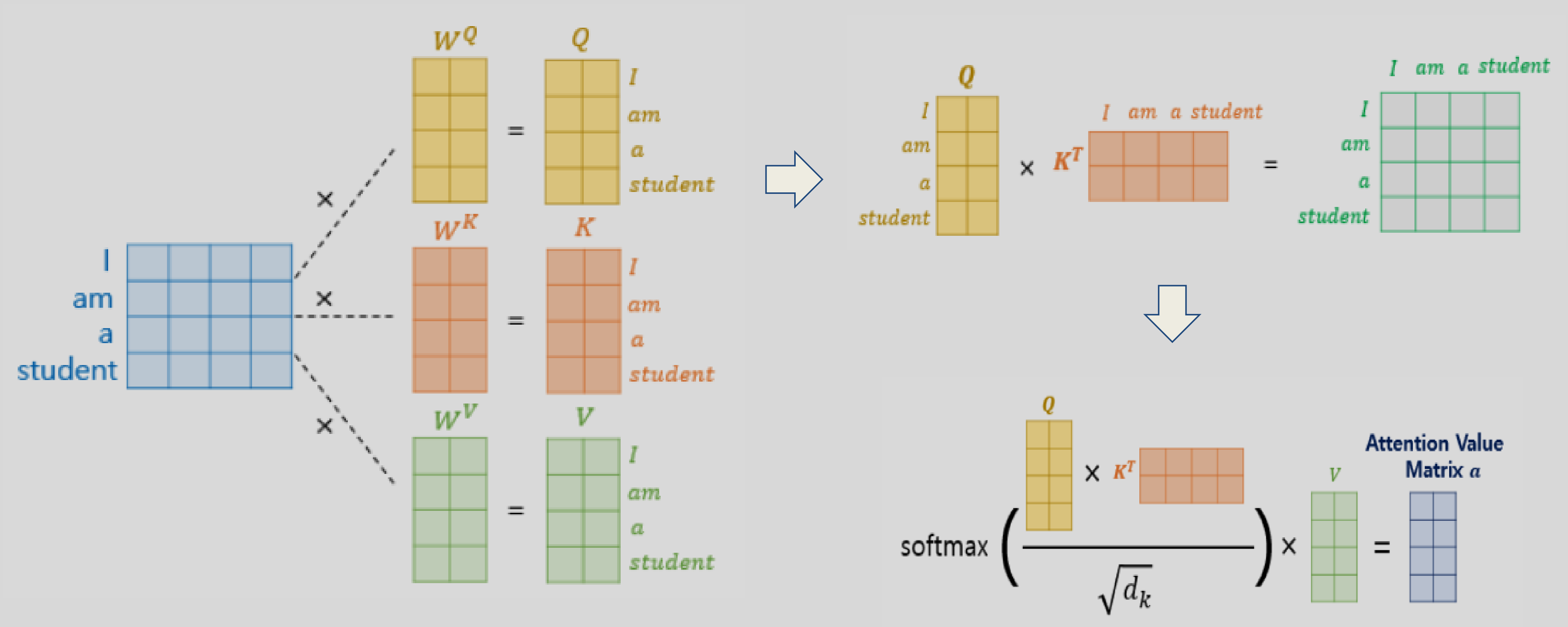






Self-attention

https://wikidocs.net/31379



Self-attention

https://wikidocs.net/31379

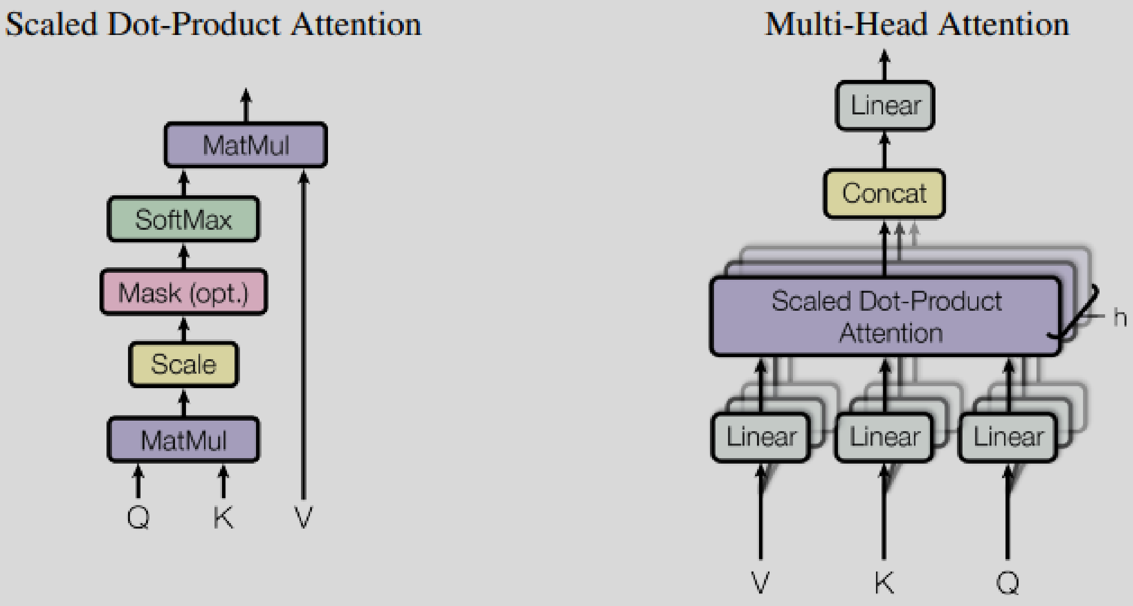


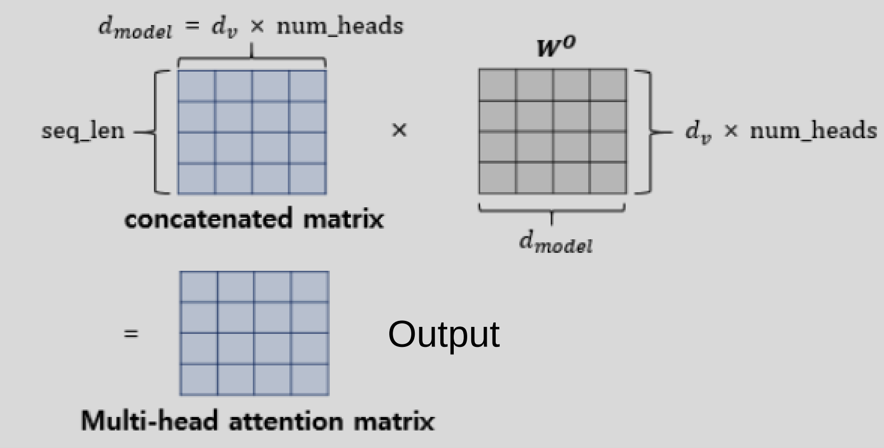
Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

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

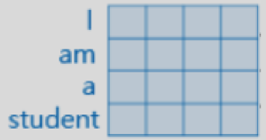
$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Self-attention

https://wikidocs.net/31379



Output



Input

- Multi-head results are concatenated and they are multipled with W^O to output the final attention matrix.

Masked attention

https://wikidocs.net/31379

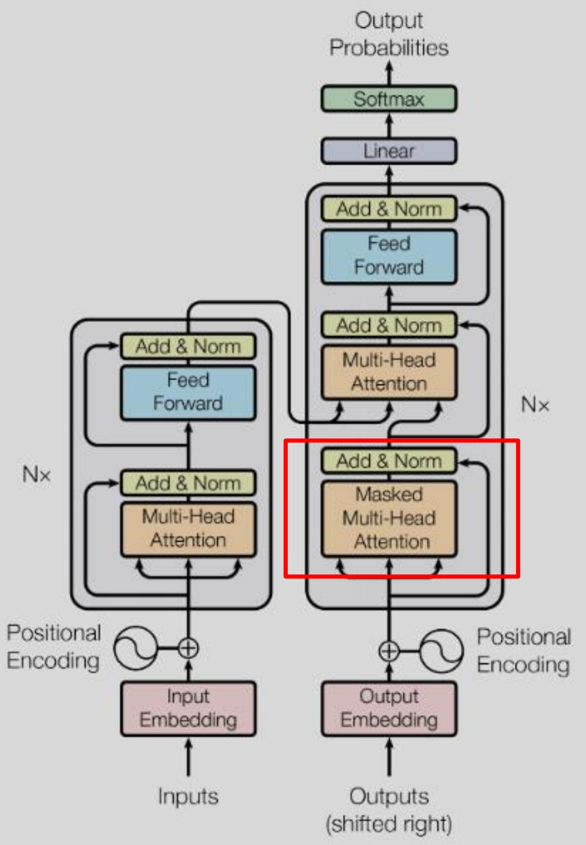
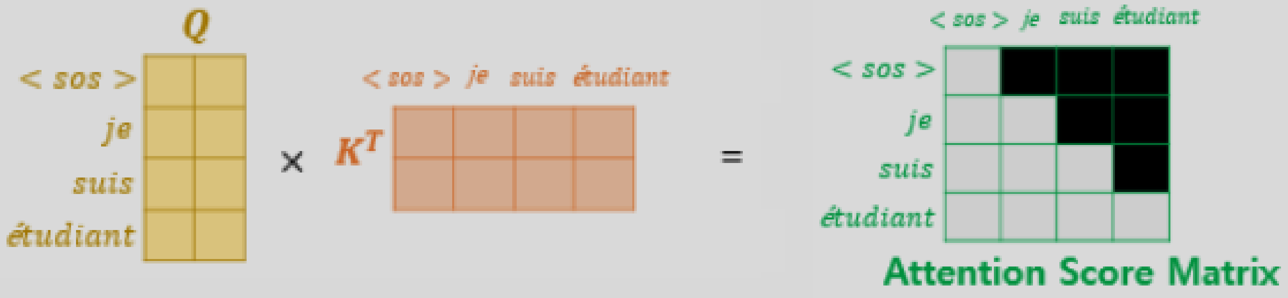
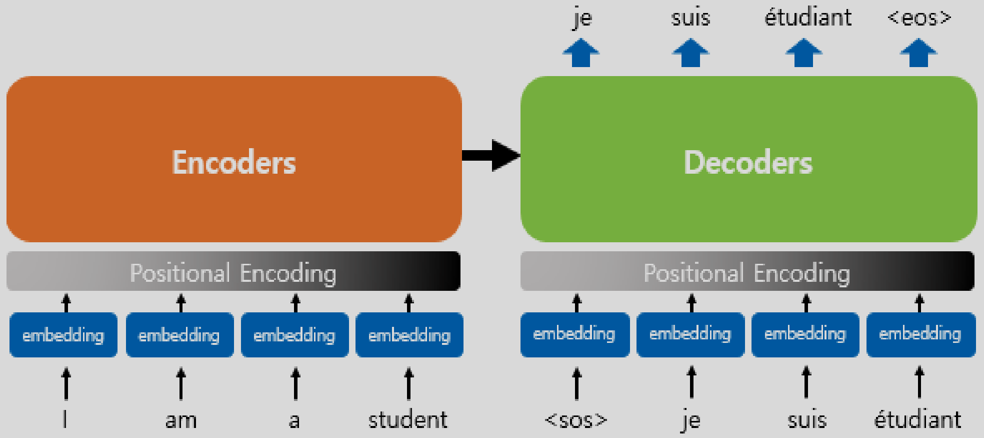


Figure 1: The Transformer - model architecture.





Cross attention

https://wikidocs.net/31379

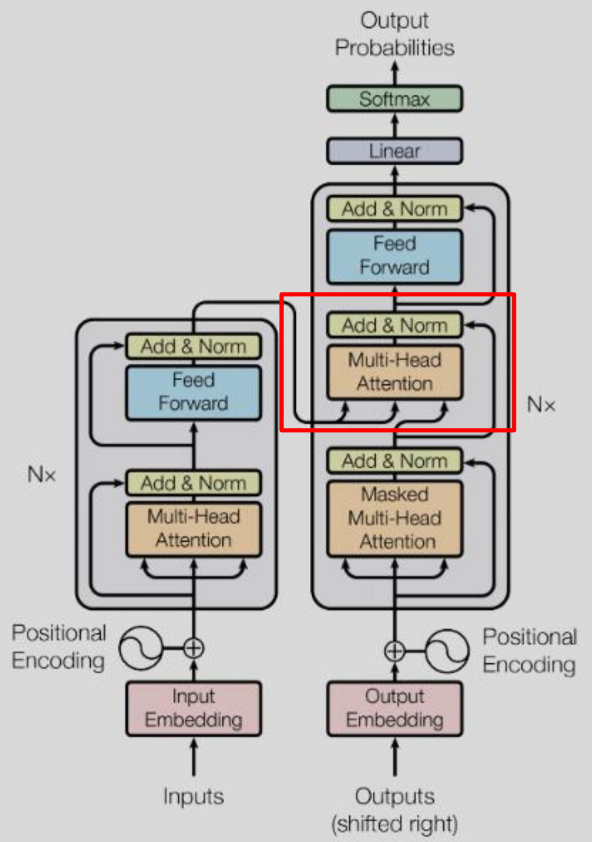
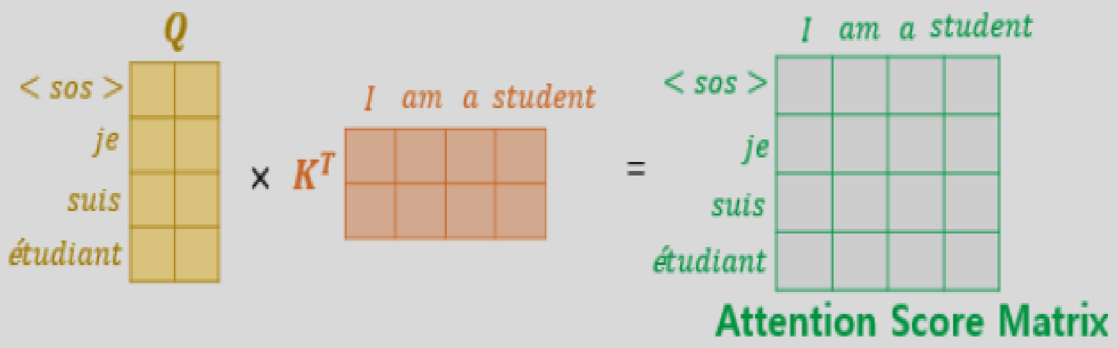
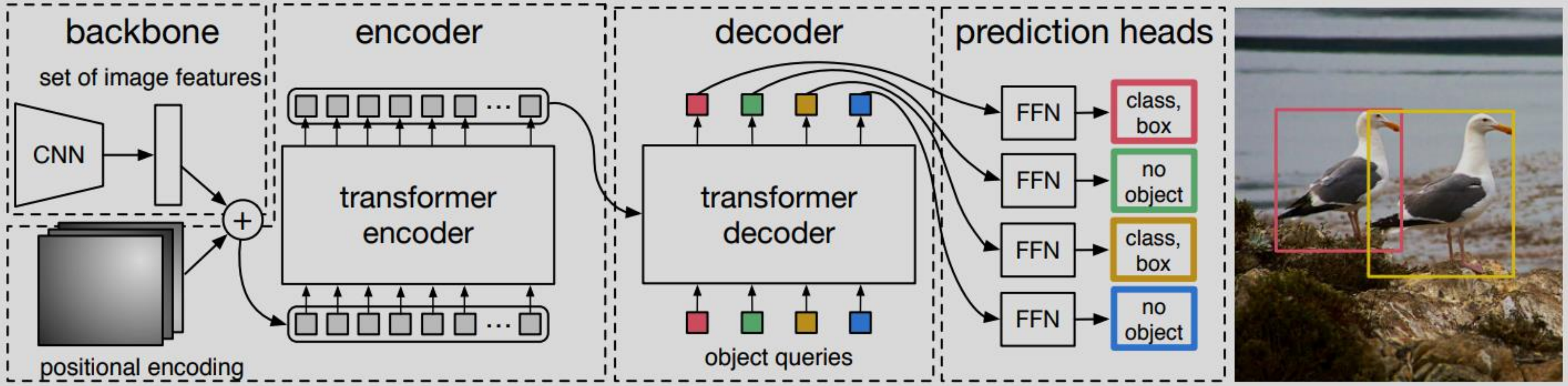


Figure 1: The Transformer - model architecture.



DETR

- Applying the Transformer for Object Detection Task.
- Better performance than Faster RCNN without any post-processing.



Input Image : $(3, H, W)$

Image feature : $(2048, H_0, W_0)$, $H_0, W_0 = H/32, W/32$

→ Flatten and projection (d, H_0W_0)

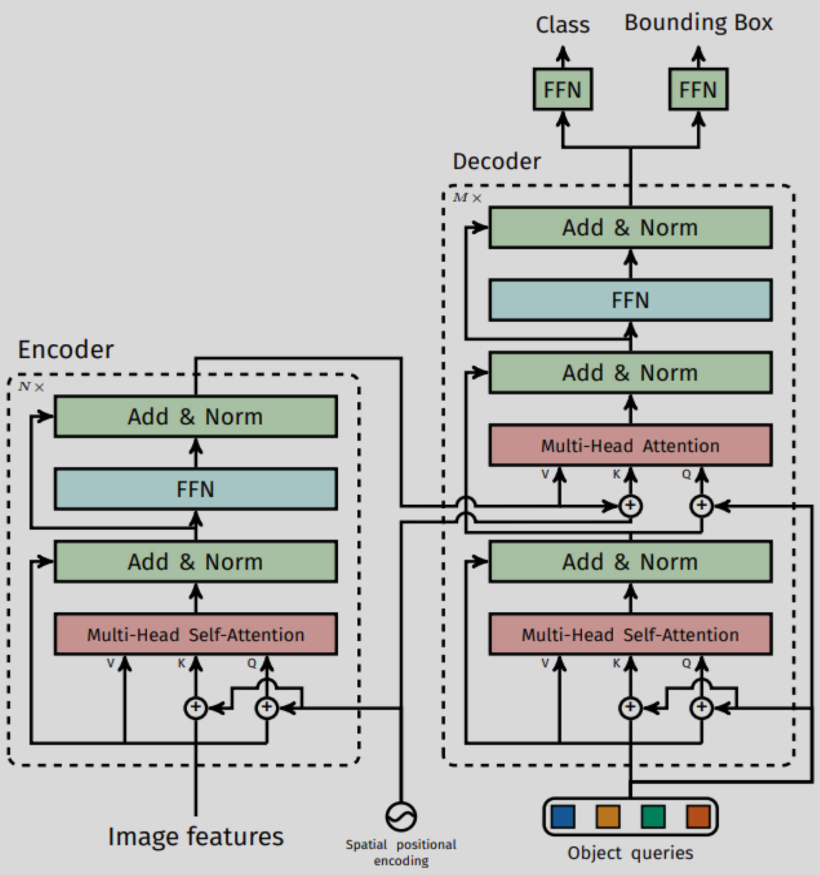
object queries : (d, N)

output : N sets of <class, bbox>

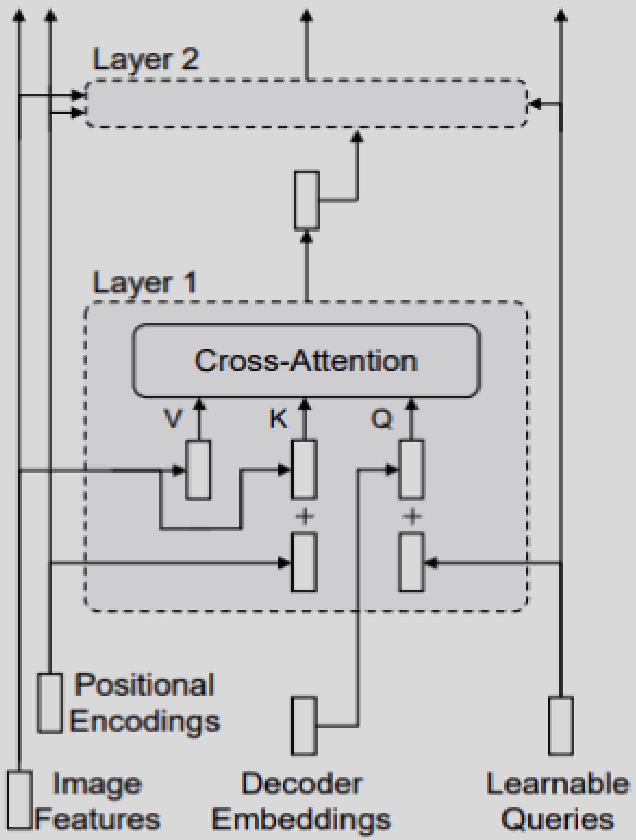
End-to-end object detection with Transformers, ECCV'20

DETR

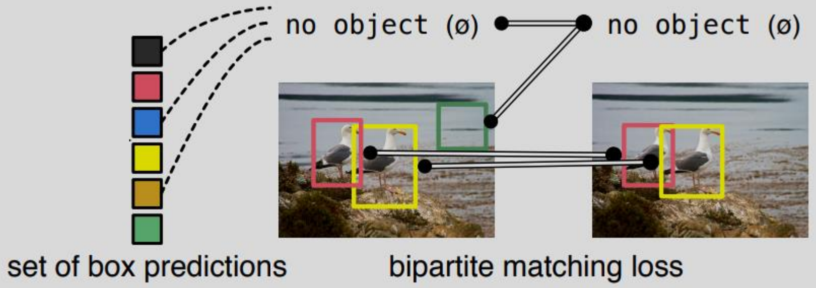
DETR Transformer

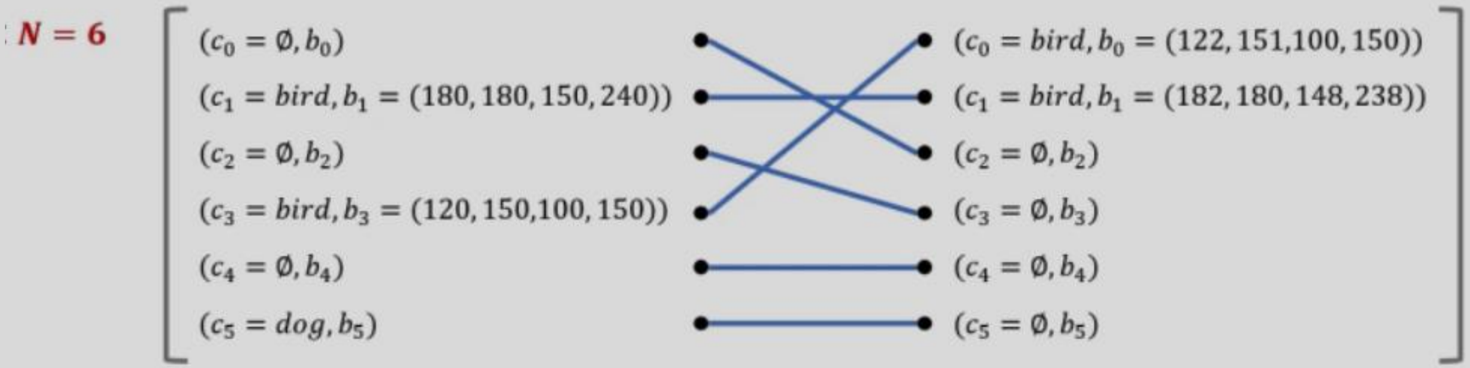


Decoder Structure



Bipartite matching





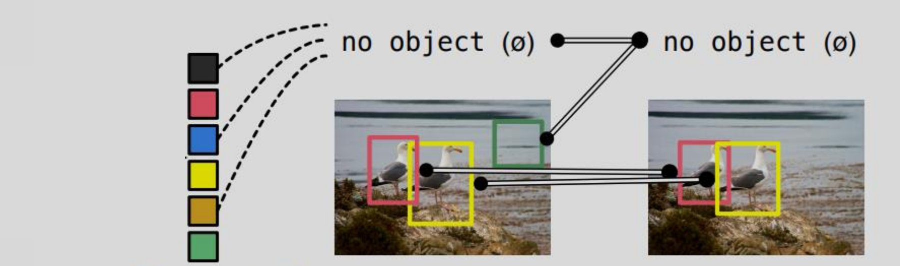
출처: https://www.youtube.com/watch?v=hCWUTvVrG7E

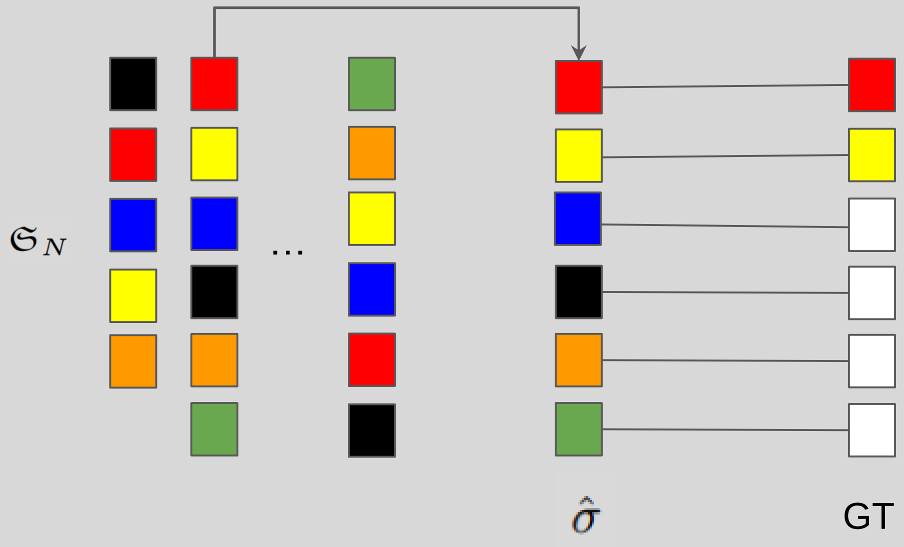
Bipartite matching

1) Find the optimal prediction set:

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

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





2) Minimize the loss for the optimal prediction set:

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

Result

Model	GFLOPS/FPS	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
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
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**

Thank you!