New text on deep neural networks for Image Analysis I

http://mrl.cs.vsb.cz//people/sojka/cnns.pdf

- It should give an idea what is happening in the area.
- Do do take it too much seriously, just try to get into the spirit.
- We have another course (Image Analysis II) that is fully focused into this area.
- The text is still under the development (i.e. unfinished at this moment).

Thanks for understanding and tolerance.

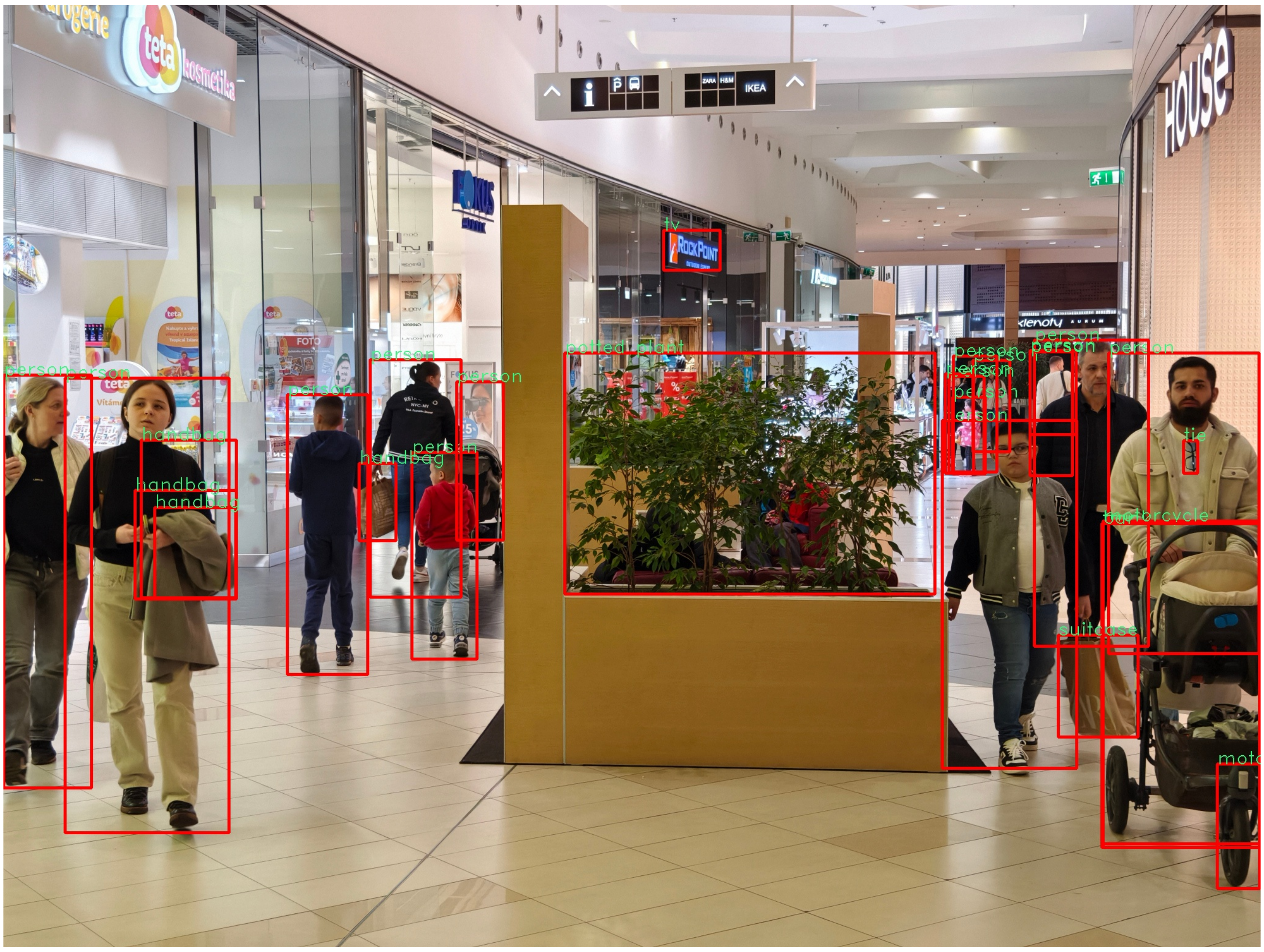
eduard

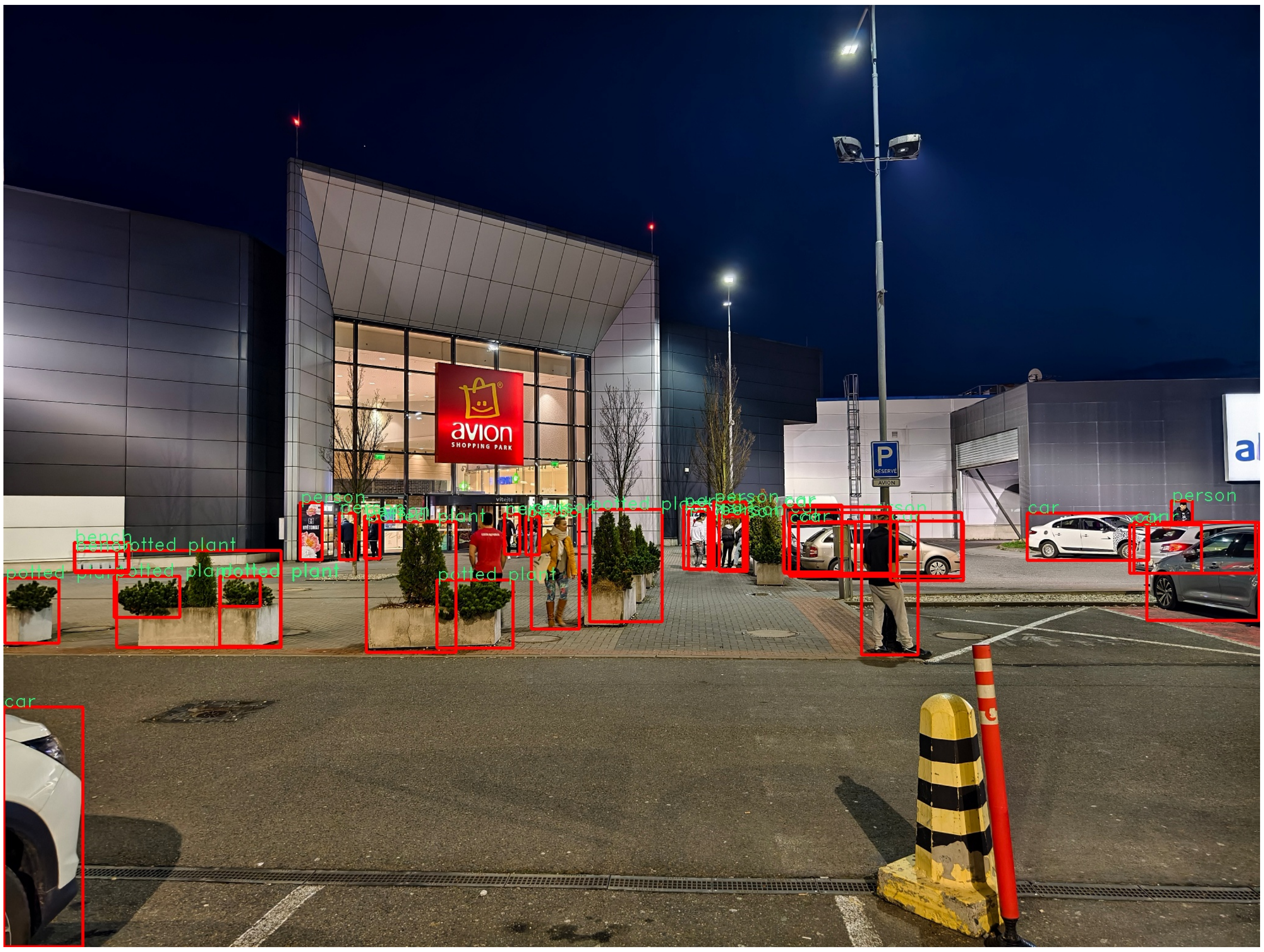
What can we do? “Hand-crafted“ recognition.

- Segmentation (similarity in color or brightness or texture, finding boundaries)
- Computing features (describing shape, color, brighness, texture)
- Classification

However, neither segmentation nor computing features are genarally easy. (segmention: consider a picture from a street, shopping centre etc.; features: consider the situation in which you are to recornise many types (classes) of objects). If segmentaion and the features are OK, the classification itself is not so complicated. („Only“ to find and separate the clusters in the space of features.)

What we can +- expect today (YOLO v8 was used in the folowing examples):


drogerie
teta kosmetika
ZARA H&M
IKEA
House
ROCKPOINT
tv
person
person
person
person
handbag
handbag
handbag
handbag
person
potted plant
person
person
person
tie
motorcycle
suitcase
motorcycle

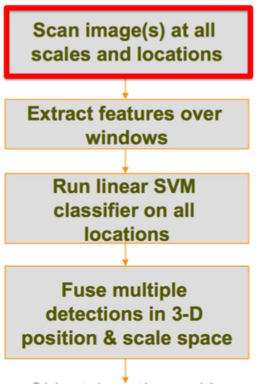

avion
SHOPPING PARK
P
RESERVE
AVION
person
bench
potted plant
potted plant
potted plant
potted plant
person
car
car
car
person
car

How we will continue in this course

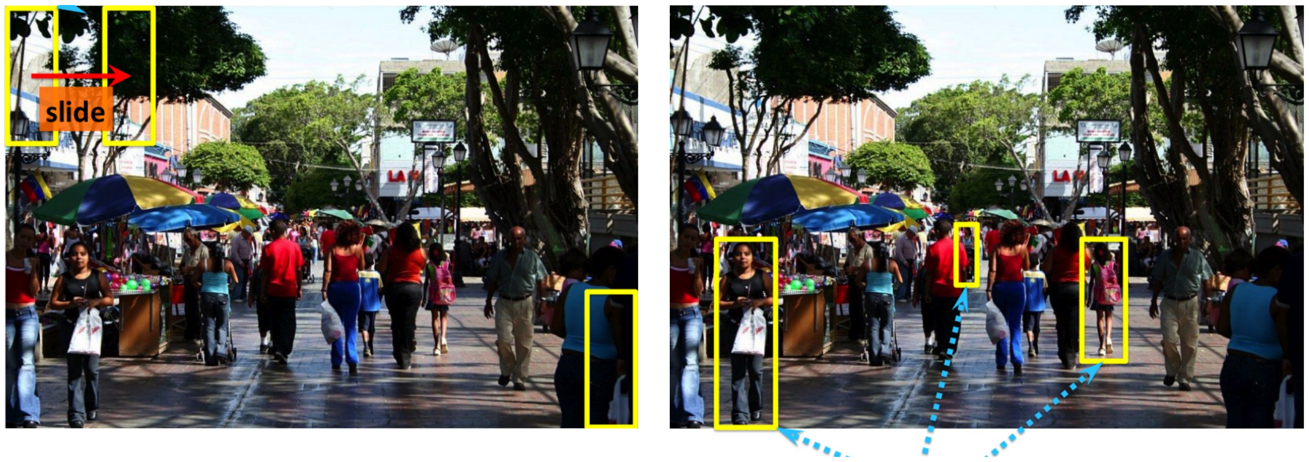
- Efficient methods before the boom of DNNs/CNNs (and also inspiring for DNNs/CNNs development): Dalal and Triggs (2005): Histograms of Oriented Gradients for Human Detection (HoG + SVM).
- CNNs/DNNs + sliding window, from LeNet (1998) to AlexNet(2012), …, and to ResNet (2015).
- R-CNN (2014), Fast R-CNN (2015), Faster-CNN (2015): The sliding window is replaced by automated proposals of windows potentially containing the objects (region proposal). The number of regions that are proposed is high (e.g. 2k); the proposals are often incorrect, the correctness is verified in the next step, which is classification. Together with classification, the position of object region may be recomputed with the goal to make it more precise.
- YOLO (You Only Look Once) V1 (2015) , … , YOLO v9 (2024): One network is responsible for everything (region proposal, classification, determining more precise region).
- Another interesting networks (e.g. with time): recurrent networks, LSTM, SelfAttention, …

I. Efficient and inspiring methods preceding the DNNs/CNNs era: Dalal and Triggs (2005): Histograms of Oriented Gradients for Human Detection (sliding window+HoG+SVM).

How to avoid segmentation: Instead of determining the pixels creating the object, only the window in which the object is possibly present is used. The features are computed from the whole area of this window. Sufficiently big training set will help the classifier to decide which features (from the whole window area) are important for recognising the object in window. The process of segmentation is thus replaced by using so called sliding window. Sliding window moves along the whole area of image by small steps. At each position, it is checked whether or no the object that is to be recognised is present at that position.

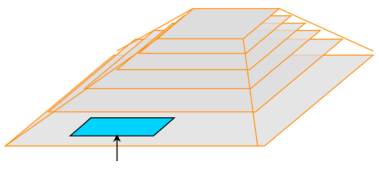

Scan image(s) at all scales and locations
Extract features over windows
Run linear SVM classifier on all locations
Fuse multiple detections in 3-D position & scale space

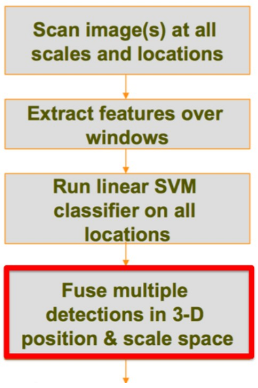
Sliding Window principle

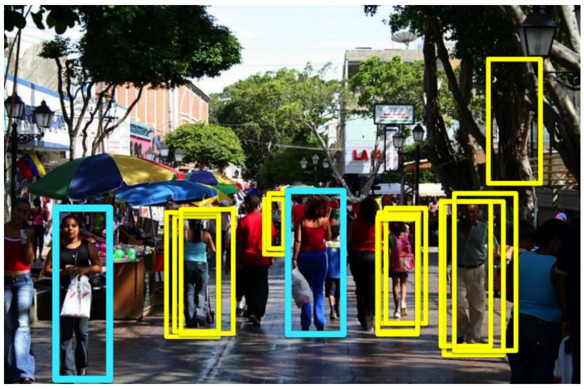


Say that we have a method that can recognise the human providing that the human fits into the yellow sliding window. Sliding window moves along the whole area of image by small steps. At each position, it is checked whether or no the object that is to be recognised is present at that position. As a result, we may obtain e.g. the probability that the human of the size corresponding to the size of window is present at that position.

The problem is that the people in the picture are of different size. The sliding window usually retains its size (since the size is connected with the subsequent steps). Instead the analysed picture is rescaled (reduced for big people, enlarged for small people). Many image scales are usually used (i.e. we have a pyramid of rescaled images as is depicted in the figure).

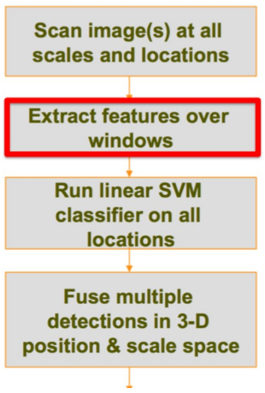


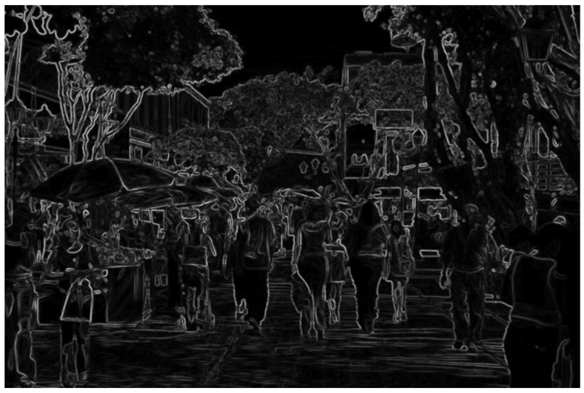

Scan image(s) at all scales and locations
Extract features over windows
Run linear SVM classifier on all locations
Fuse multiple detections in 3-D position & scale space

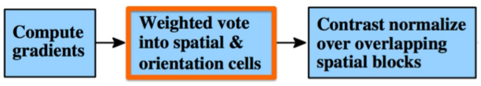


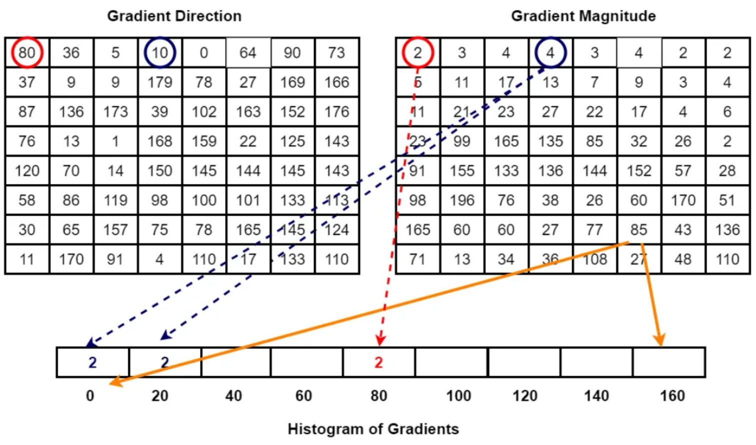
Finally, the results from all positions of sliding window and from all image scales are evaluated. 3D probability maps are created. Maximum probability values are taken into account, overlapping detections are removed.

HoG principle


Scan image(s) at all scales and locations
Extract features over windows
Run linear SVM classifier on all locations
Fuse multiple detections in 3-D position & scale space




Compute gradients
Weighted vote into spatial & orientation cells
Contrast normalize over overlapping spatial blocks



Gradient Direction

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Magnitude

2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	36	108	27	48	110

Histogram of Gradients

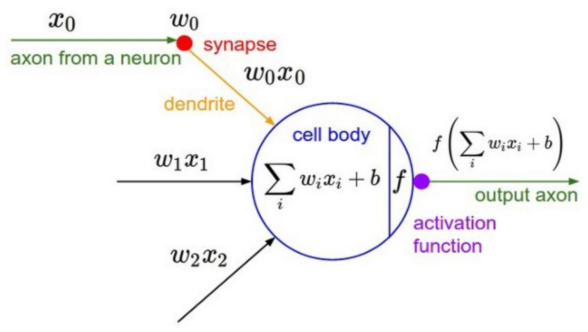
0	20	40	60	80	100	120	140	160
2	2			2				

Quite often blocks containing 8x8 pixels are used (as in the figure above). 2x2 blocks are usually joined together. Their histogram vectors are simply concatenated, which gives 36 values. This final vector is finally normalised to unit length. Dalal & Triggs used the sliding window of size 64x128. What was the size of the final feature vector for this window? (This is a homework. :-))

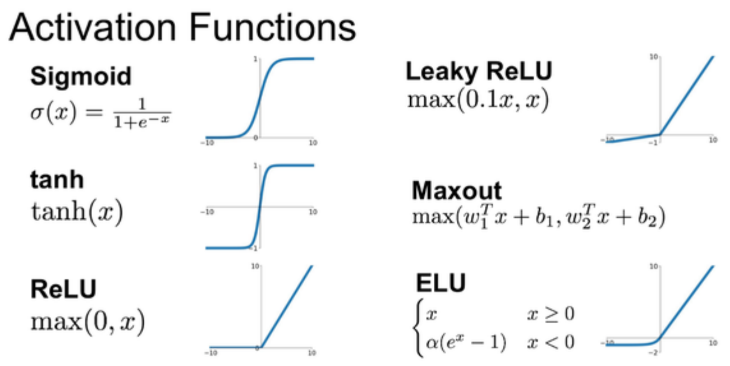
Once we have the feature vector for the whole sliding window, it can be sent to a classifier. Support vector machine (SVM) was used in the original approach. It would also be possible to use a shallow neural network.

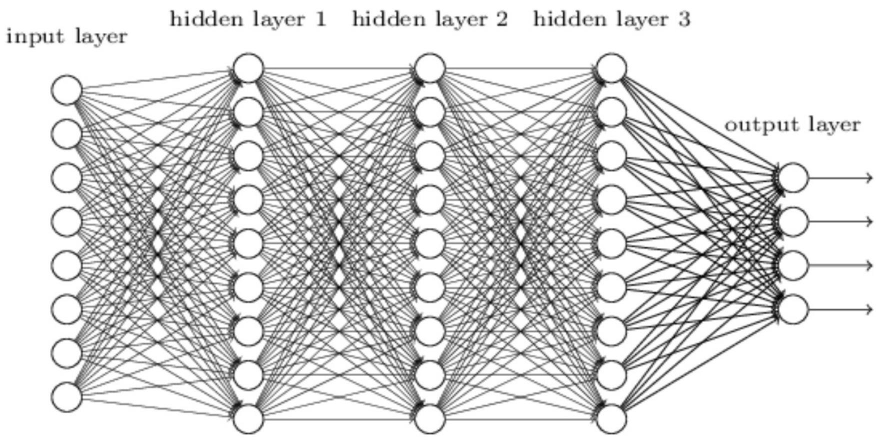
II. Sliding window + CNNs (DNNs)

Brief repetition of what we have already done (neural cell + fully connected layers)

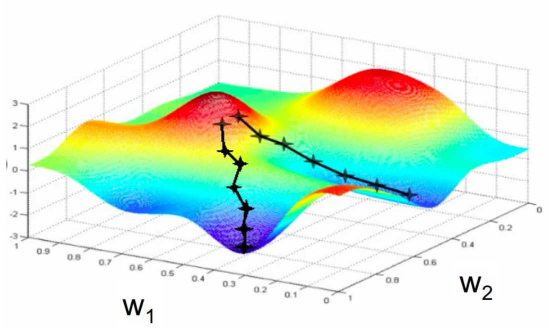


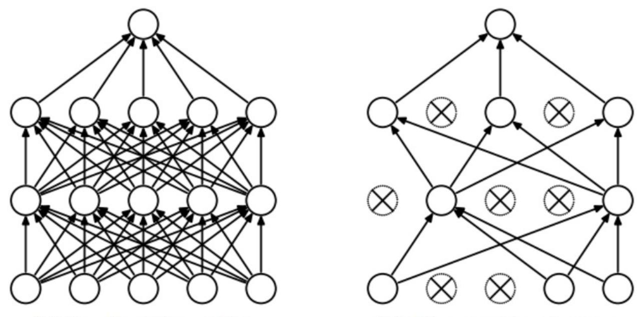
Activation functions: **Sigmoid, Tanh, ReLU, Leaky ReLU**





Update weights by **gradient descent:** $\mathbf{w} \leftarrow \mathbf{w} - \alpha \dfrac{\partial E}{\partial \mathbf{w}}$

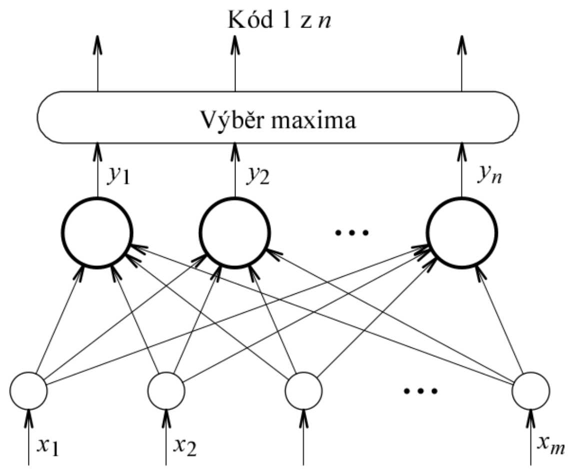




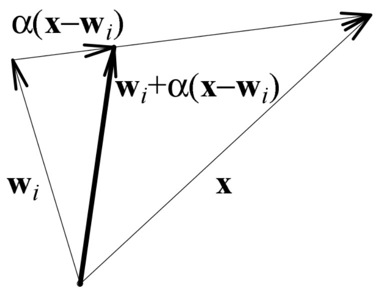
(a) Standard Neural Net (b) After applying dropout.

Gradient descent, stochastic gradient descent, (mini)batch gradient descent.

Another useful prerequisite: Networks with competition (Kohonen Maps, 1982)



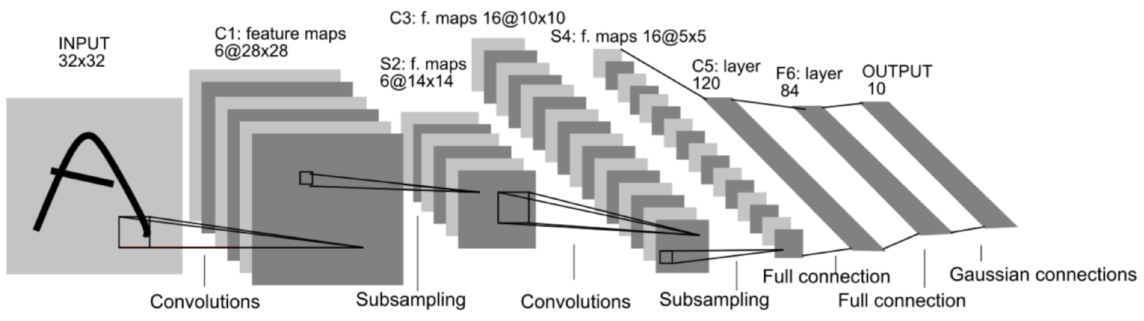
$$y_i = -\sqrt{\sum_{j=1}^{m}\left(x_j - w_{i,j}\right)^2} = -\left|\mathbf{x}-\mathbf{w}_i\right|$$


$\alpha(\mathbf{x}-\mathbf{w}_i)$
$\mathbf{w}_i+\alpha(\mathbf{x}-\mathbf{w}_i)$
$\mathbf{w}_i$
$\mathbf{x}$

Finally, we must be familiar with the convolution. We have discussed it in the previous course (Digital Image Processing). I will briefly repeat it during the lecture. It would not give a sense otherwise.

Why the convolution is used in the networks: It should extract what is important for recognition

The pioneering work LeNet (1998)



LeNet-5: Y. Lecun et al.: Gradient-based learning applied to document recognition (1998). Feature extraction by convolutional layers and pooling is followed by final classification, which is carried out in the fully connected layers. (You may notice, if you want, the term “Gaussian connections”. The last layer can be viewed as Kohonen map. We would probably use softmax today.)

In the past, we were interested what is exactly happening in the convolutional layers. It is illustrated by the following figure.

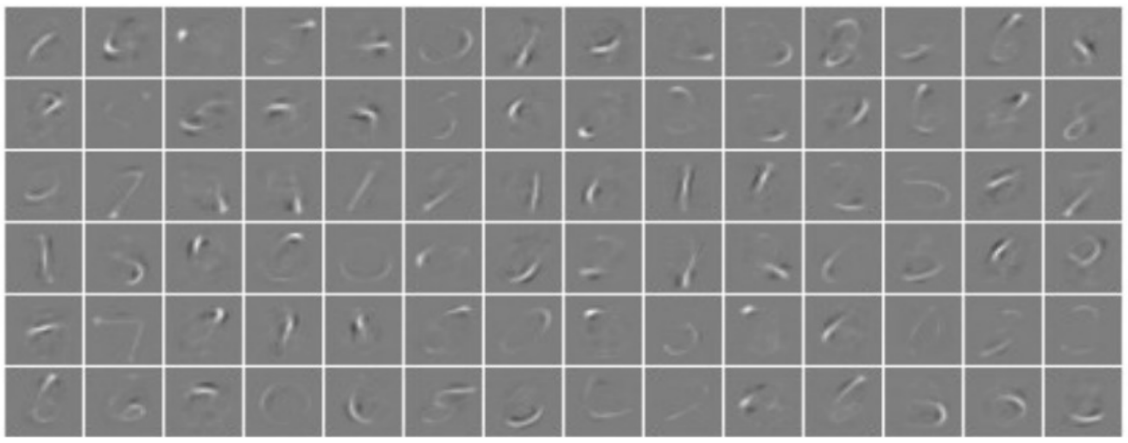


Fig. 3: Activations taken from the first convolutional layer of a simplistic deep CNN, after training on the MNIST database of handwritten digits. If you look carefully, you can see that the network has successfully picked up on characteristics unique to specific numeric digits.

A more general view on how the particular layers are put together:

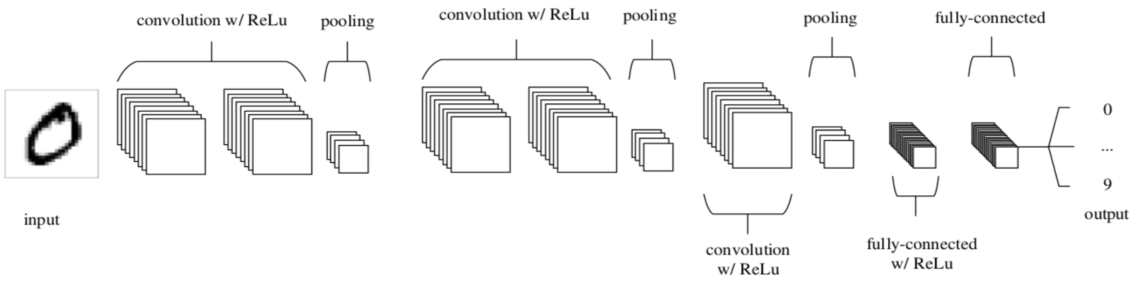
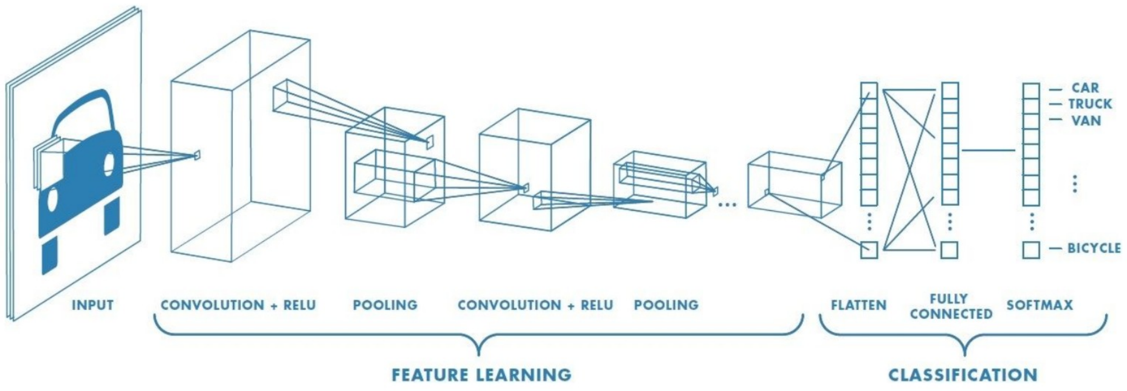
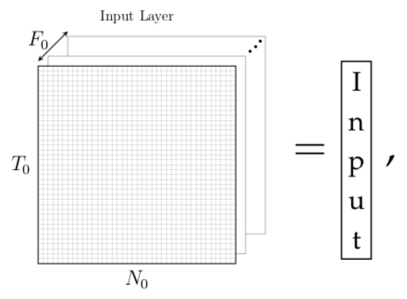
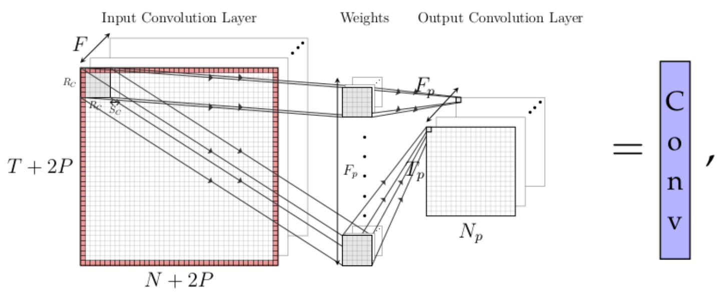


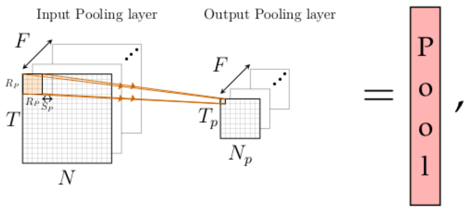
Fig. 5: A common form of CNN architecture in which convolutional layers are stacked between ReLus continuously before being passed through the pooling layer, before going between one or many fully connected ReLus.

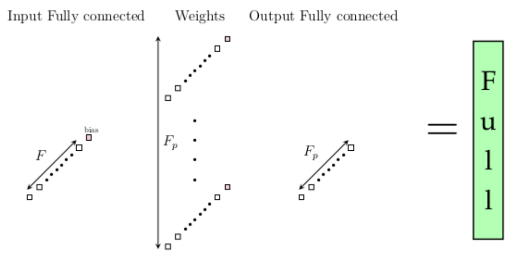


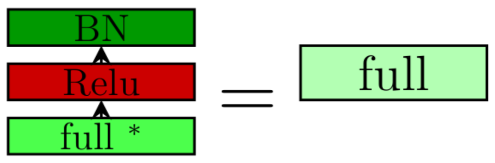
Elements of NNs and the key for reading the figures that follow

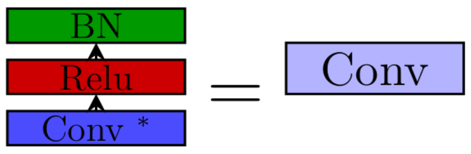












Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

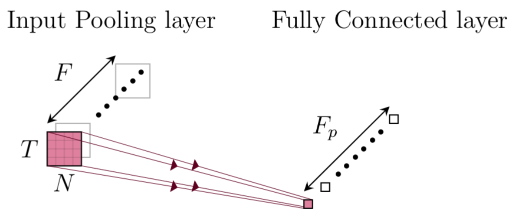
$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

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

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

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

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

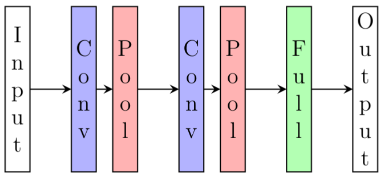


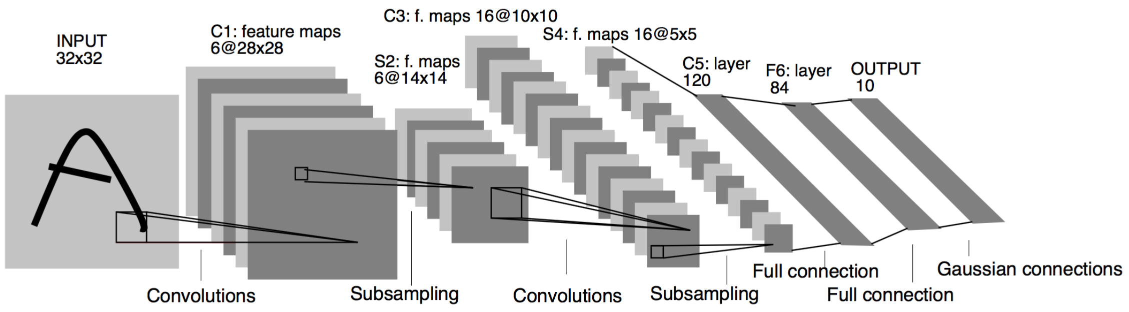
Softmax

- A special kind of activation layer, usually at the end of FC layer outputs
- Can be viewed as a fancy normalizer (a.k.a. Normalized exponential function)
- Produce a discrete probability distribution vector
- Very convenient when combined with cross-entropy loss

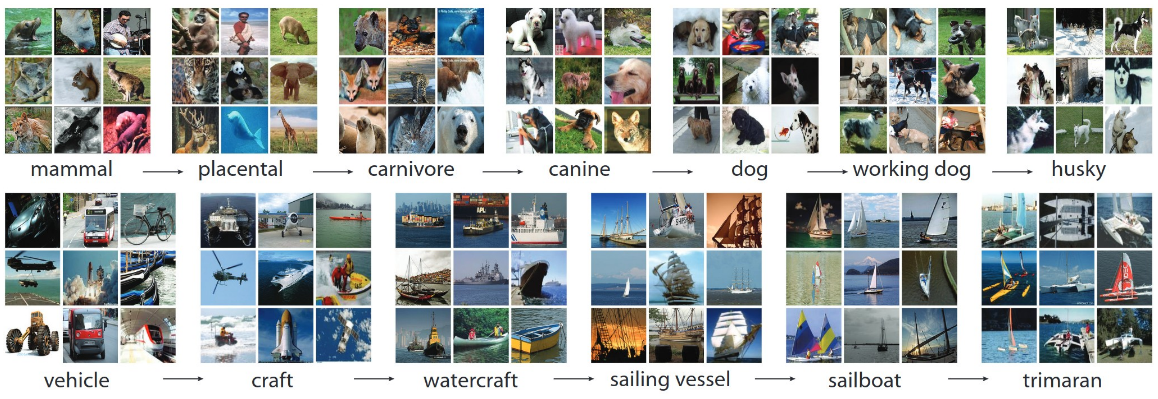
$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

Given sample vector input **x** and weight vectors {$\mathbf{w}_i$}, the predicted probability of y = j

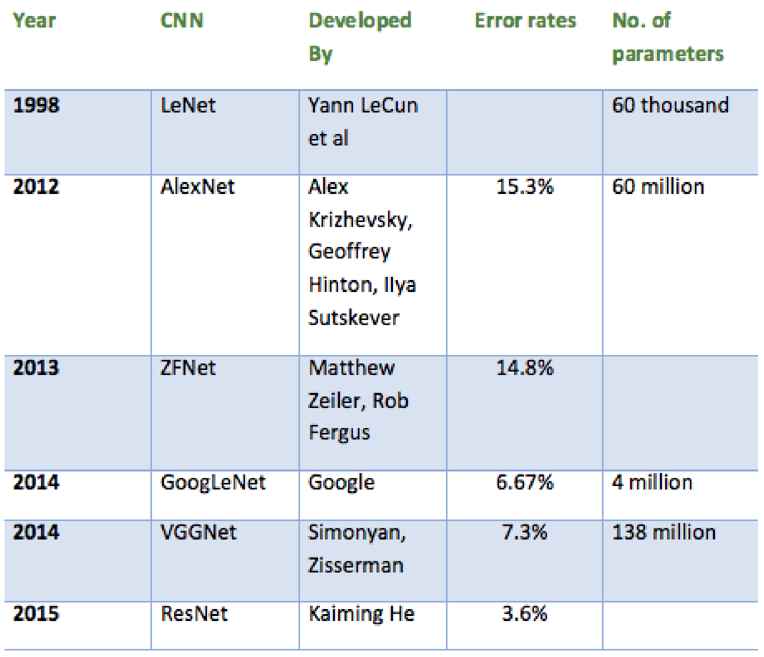




LeNet a co dál? ILSVRC (ImageNet Large Scale Visual Recognition Challenge) is an annual computer vision competition. It is done on a subset of a computer vision dataset called ImageNet https://www.image-net.org/challenges/LSVRC/.



V soutěži je více kategorií, prohlédněte si také výsledky pro jednotlivé roky, např. https://image-et.org/challenges/LSVRC/2015/results. Další sítě jsou vítězné sítě v této soutěži.



Year	CNN	Developed By	Error rates	No. of parameters
1998	LeNet	Yann LeCun et al		60 thousand
2012	AlexNet	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	15.3%	60 million
2013	ZFNet	Matthew Zeiler, Rob Fergus	14.8%	
2014	GoogLeNet	Google	6.67%	4 million
2014	VGGNet	Simonyan, Zisserman	7.3%	138 million
2015	ResNet	Kaiming He	3.6%	

AlexNet

AlexNet (*Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton:* ImageNet Classification with Deep Convolutional Neural Networks, 2012) was the winner of ImageNet ILSRVC 2012 (team name SuperVision).

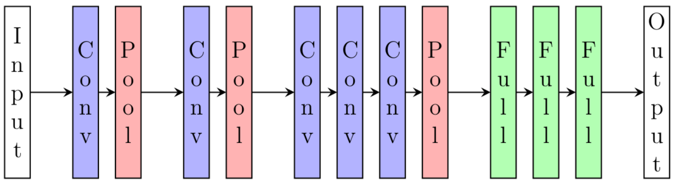
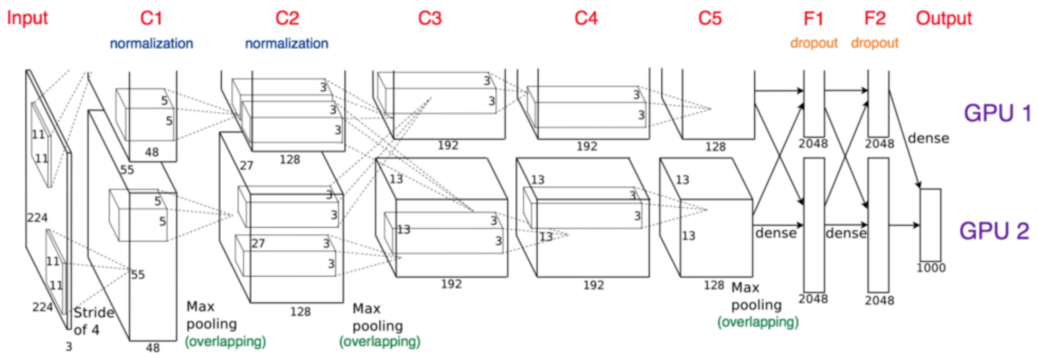
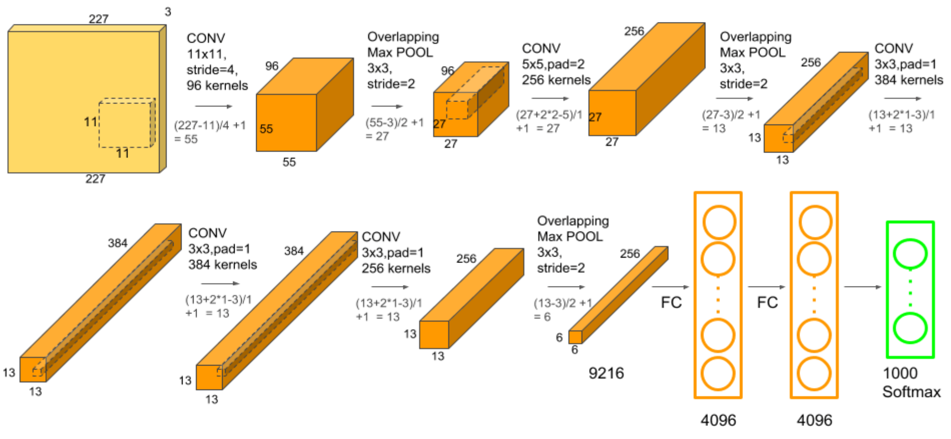


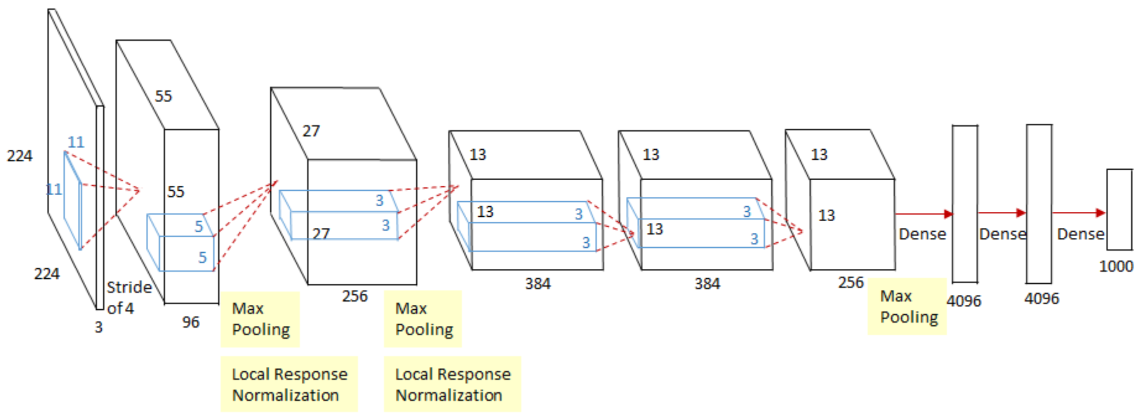
Figure 5.12: The AlexNet CNN

In the original paper:



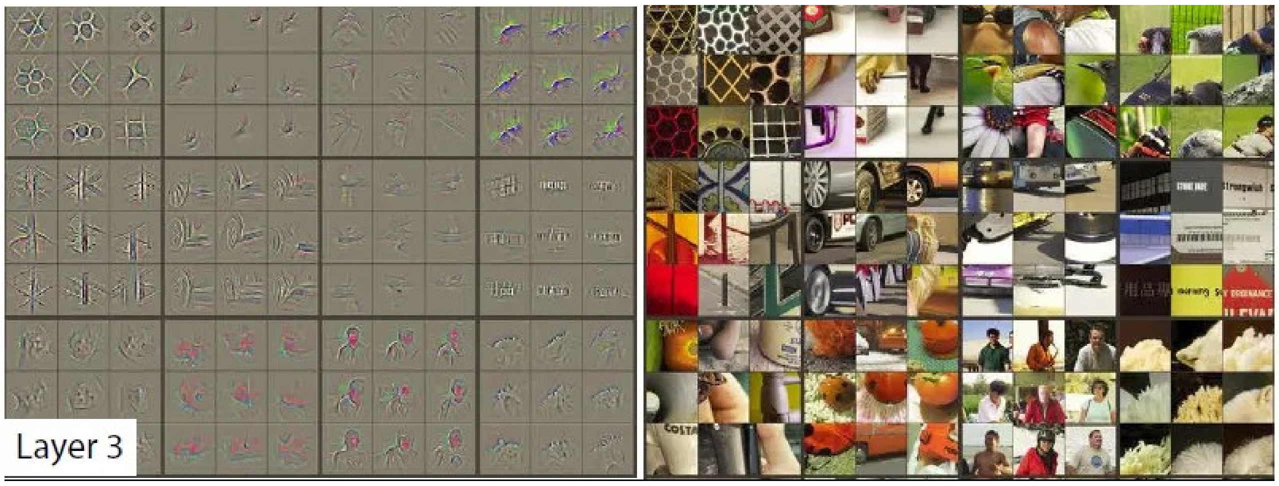
Caffe implementation (upper figure contains more details):

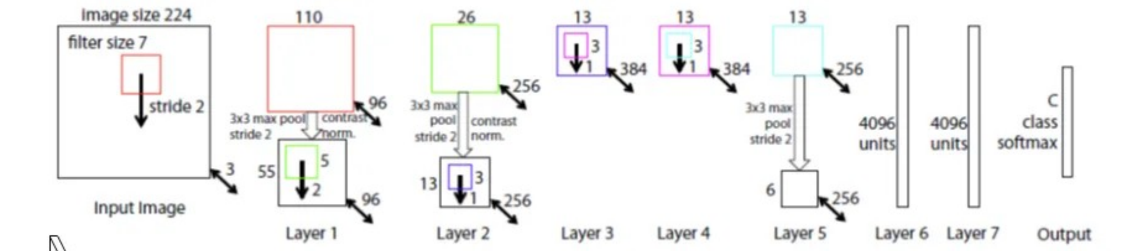

227
227
3
11
CONV 11x11, stride=4, 96 kernels
(227-11)/4 +1 = 55
96
55
Overlapping Max POOL 3x3, stride=2
(55-3)/2 +1 = 27
27
CONV 5x5,pad=2 256 kernels
(27+2*2-5)/1 +1 = 27
256
Overlapping Max POOL 3x3, stride=2
(27-3)/2 +1 = 13
13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
384
CONV 3x3,pad=1 256 kernels
Overlapping Max POOL 3x3, stride=2
(13-3)/2 +1 = 6
6
9216
FC
4096
4096
1000 Softmax

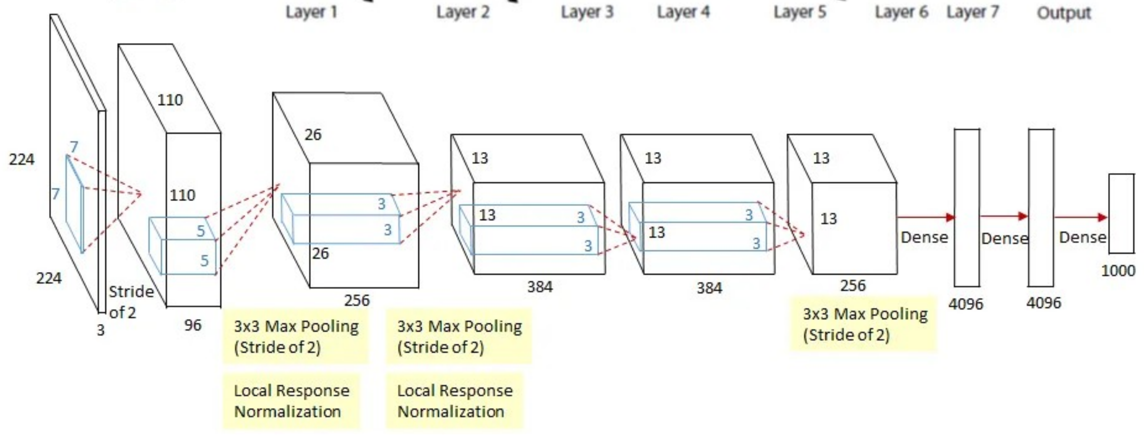


ZFNet

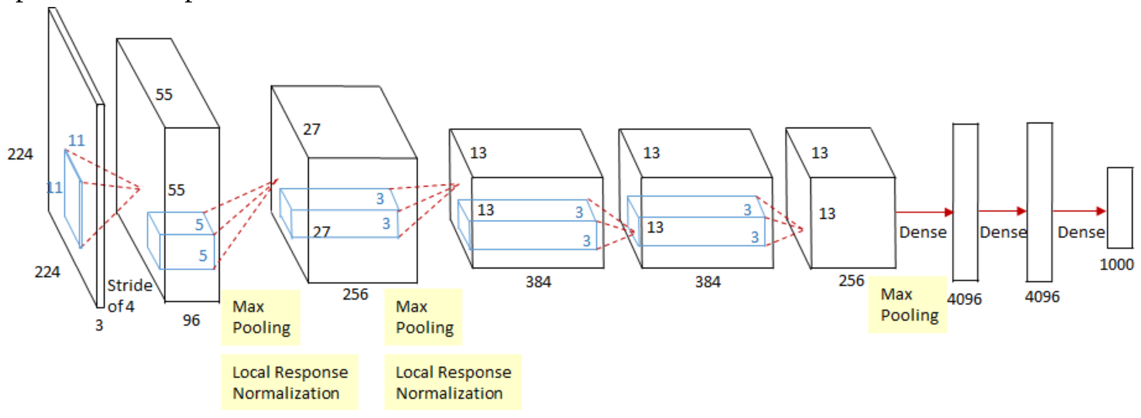
ZFNet: Winner of ILSVRC 2013 (Image Classification). Jedná se o tuning AlexNet. Zajímavé je, jak se přišlo na to, co se má změnit. Vizualizací toho, co se děje v jednotlivých vrstvách. Pro bližší vysvětlení se lze podívat do Matthew D. Zeiler and Rob Fergus, Visualizing and Understanding Convolutional Networks, 2014, https://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf.





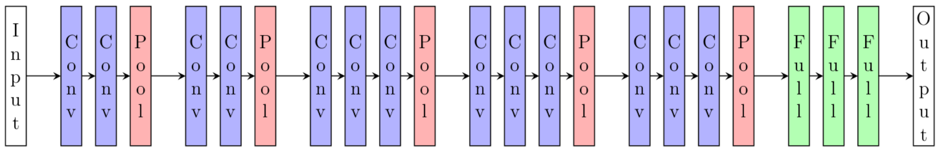


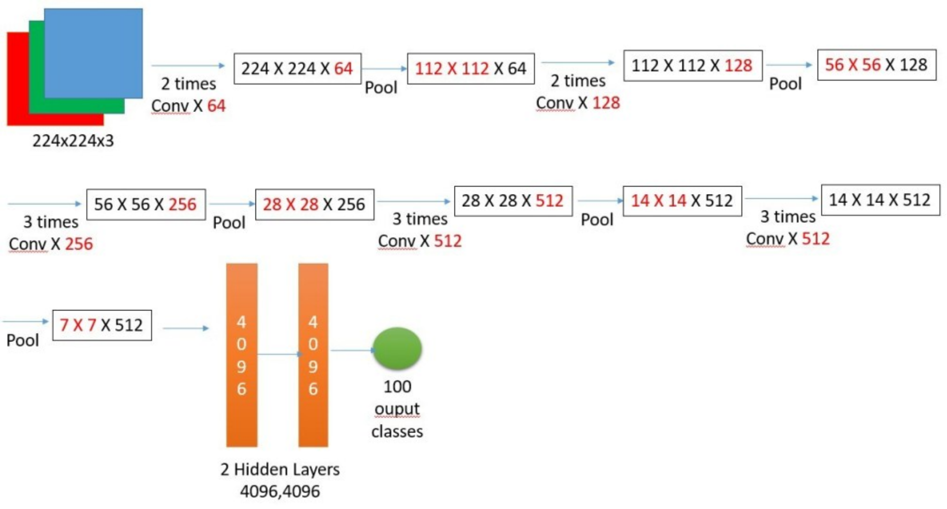
Níže je pro srovnání původní AlexNet

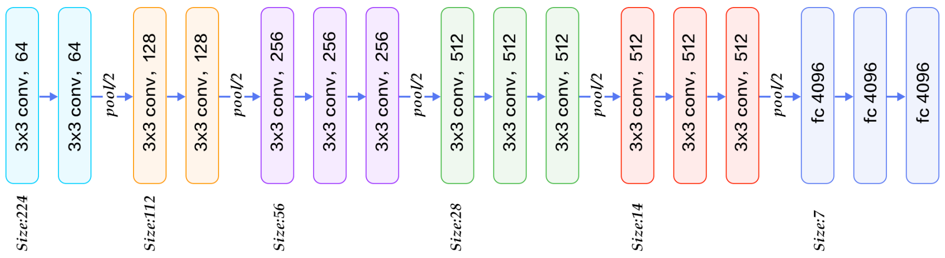


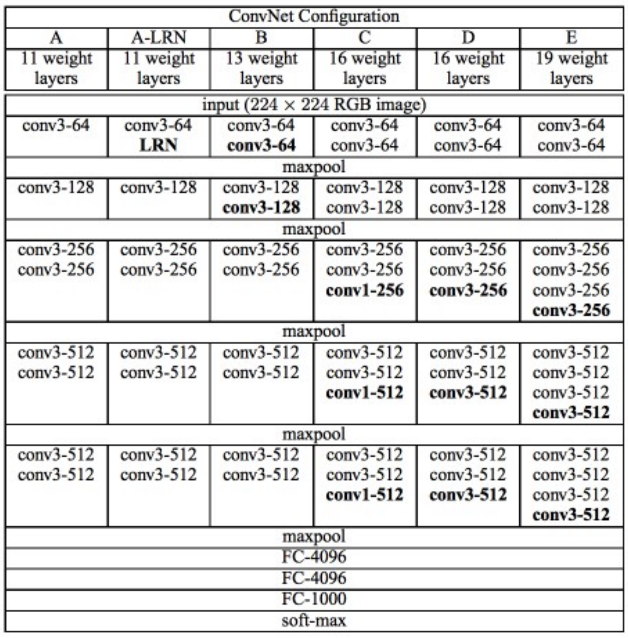
VGG: Oxford Visual Geometry Group

VGG: Karen Simonyan, Andrew Zisserman: Very Deep Convolutional Networks for Large-Scale Image Recognition (2015)









ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Above: Several types of visualisation, table of possible configurations. VGG is a famous network that has been also used as a building block in further networks.

GoogleNet CNN

C. Szegedy et al.: Going Deeper with Convolutions (2015)

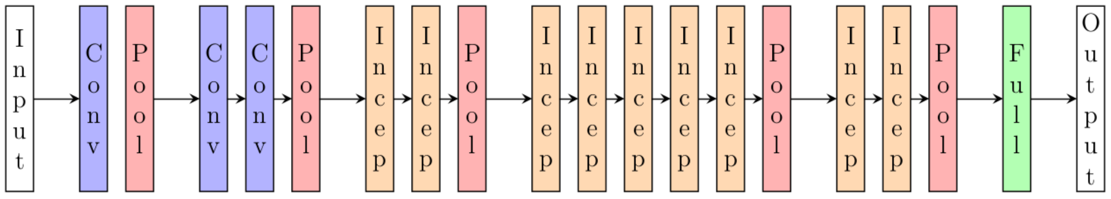
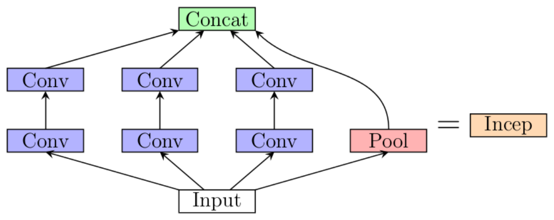


Figure 5.15: The GoogleNet CNN



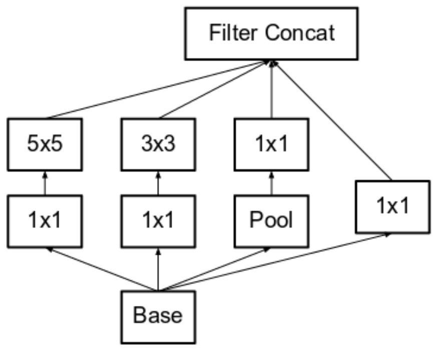
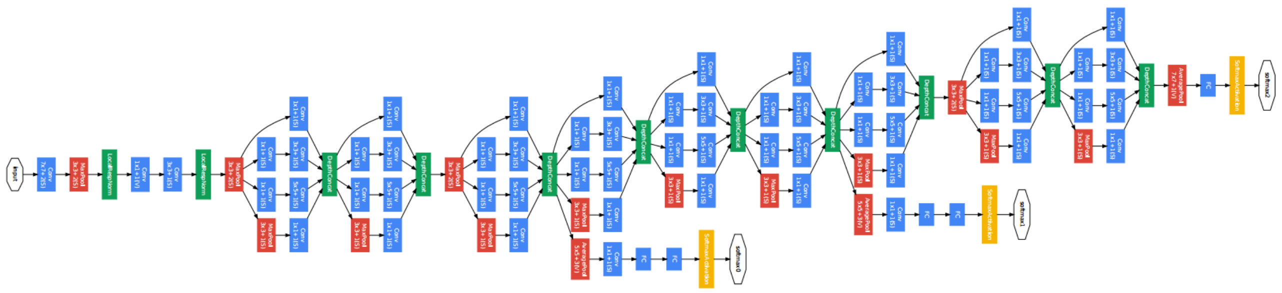
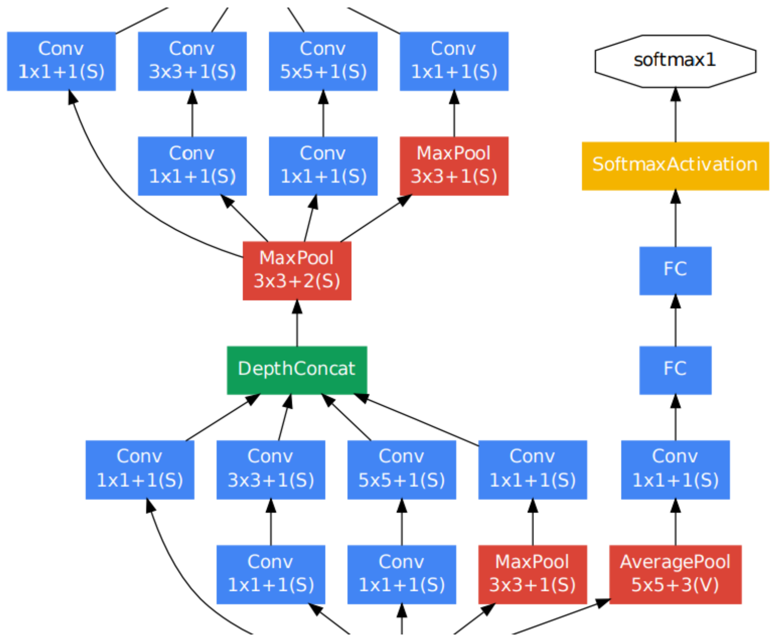
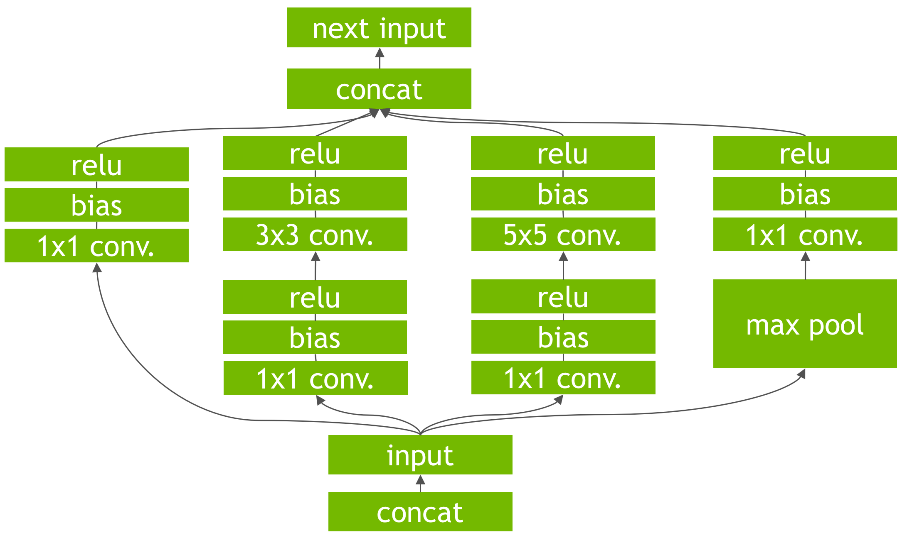


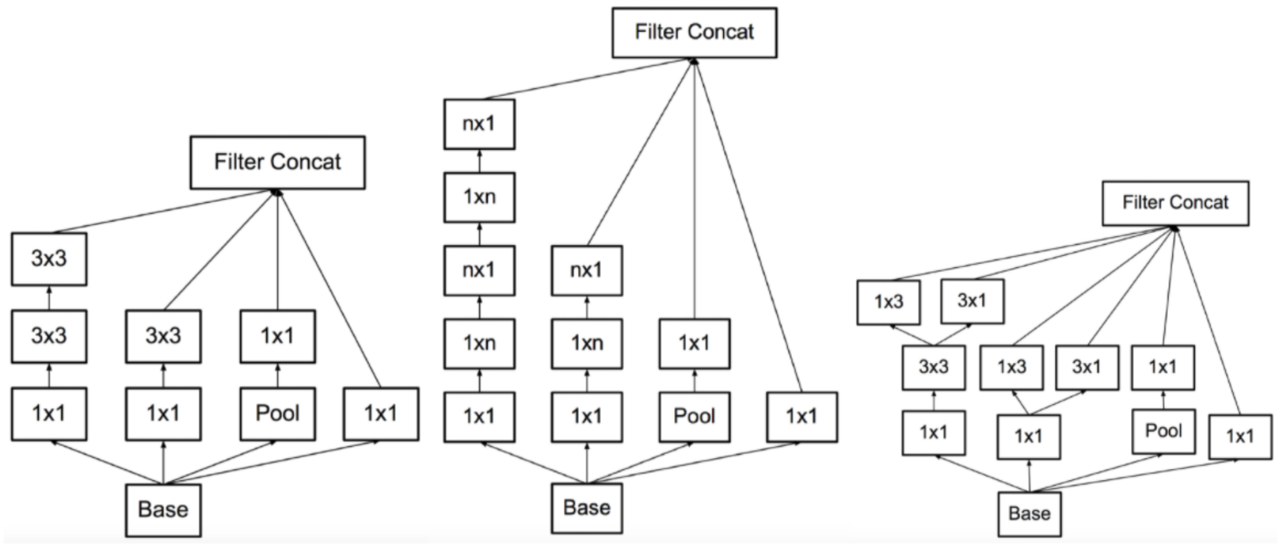
Figure 5.14: The Inception module







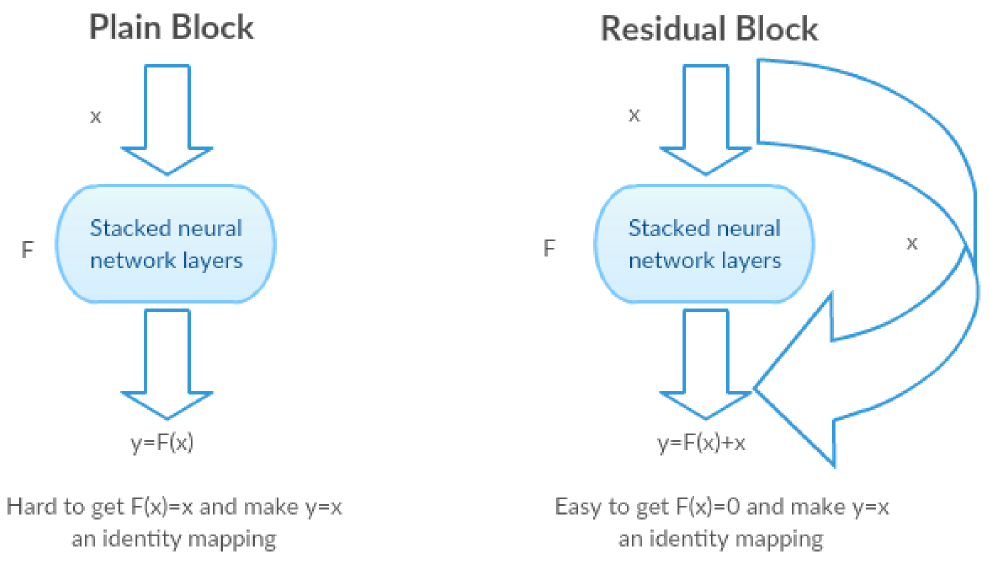
NVIDIA totéž



C. Szegedy et al.: Rethinking the Inception Architecture for Computer Vision (2015). Zlepšuje zde svůj předchozí návrh z téhož roku. Síť by měla mít méně parametrů. Odtud rychlejší učení i inference.

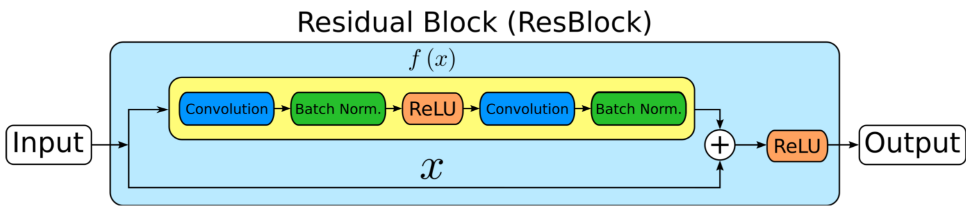
ResNet CNN

Kaiming He et al.: Deep Residual Learning for Image Recognition (2015)
Kaiming He et al.: Identity Mappings in Deep Residual Networks (2016)
Jie Hu et al.: Squeeze-and-Excitation Networks (2018)

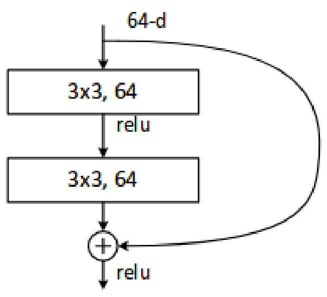


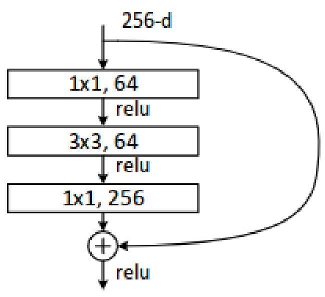
Plain Block
x
F
Stacked neural network layers
y=F(x)
Hard to get F(x)=x and make y=x an identity mapping
Residual Block
x
F
Stacked neural network layers
x
y=F(x)+x
Easy to get F(x)=0 and make y=x an identity mapping

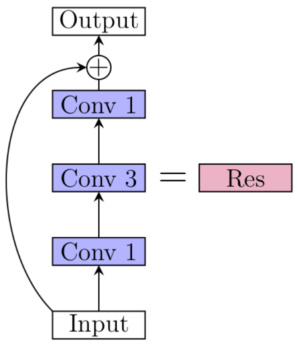
If the identity mapping is optimal, it is very easy to come up with a solution like F(x) =0 rather than F(x)=x using stack of non-linear CNN layers as function.

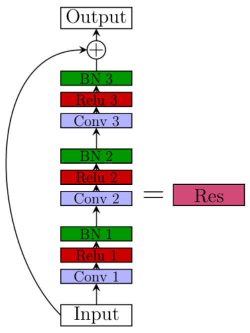


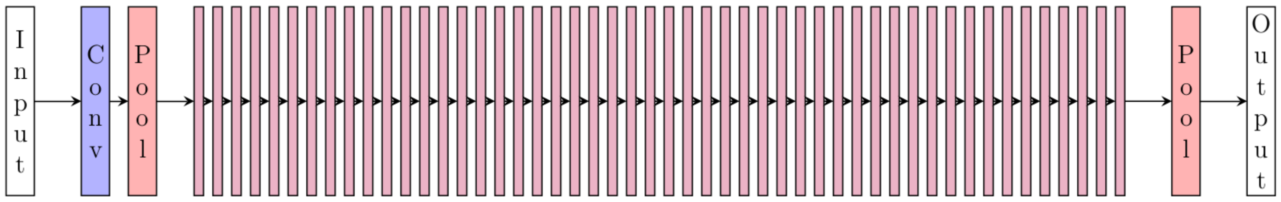
More variants are possible. Two examples follow.



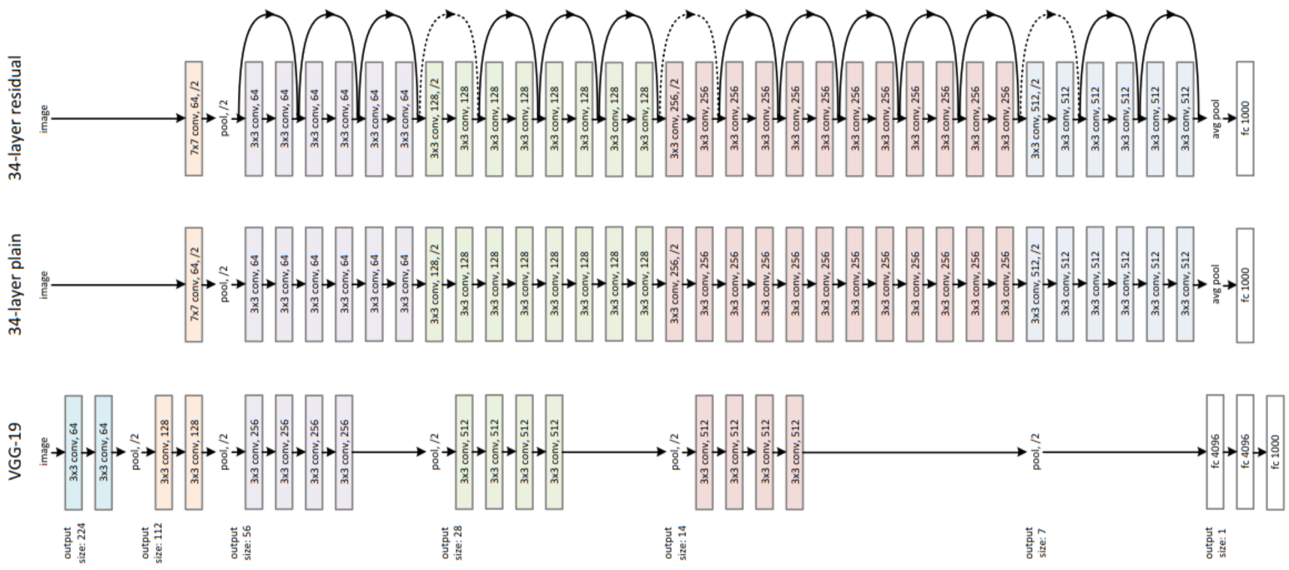




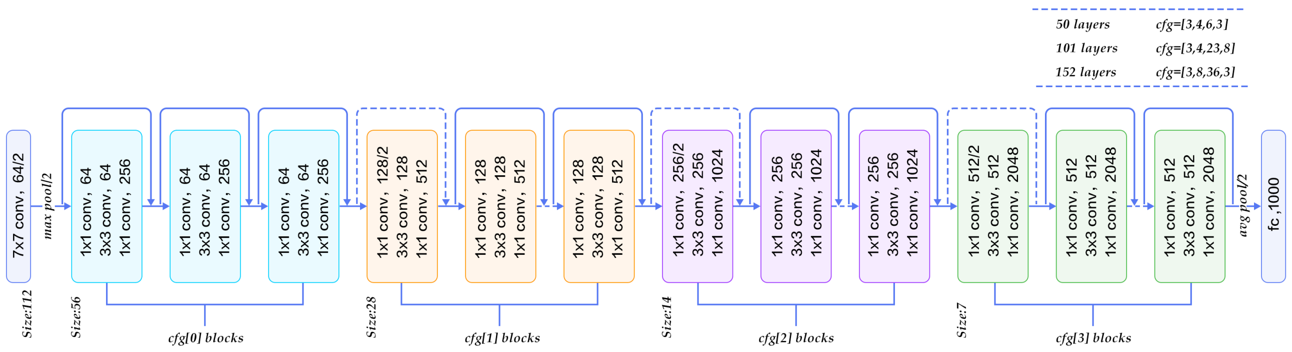




More explicitly:



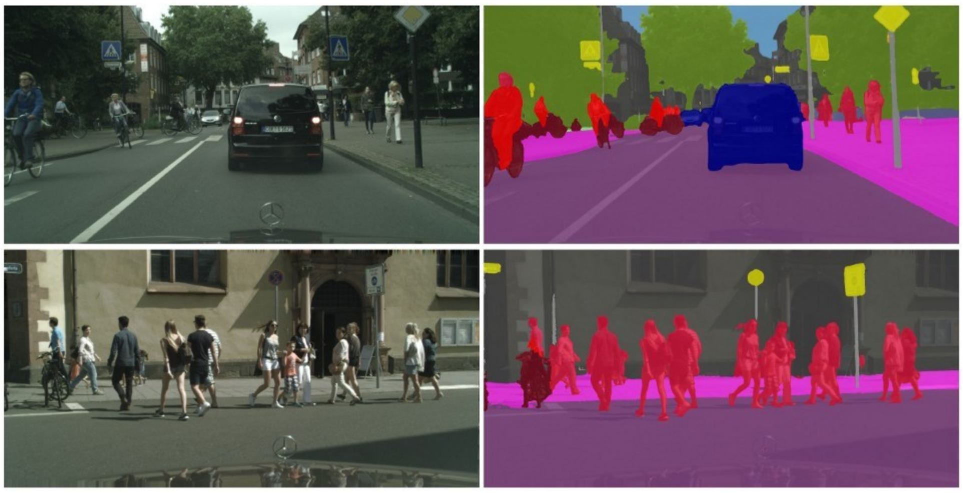
Another parametrised example:



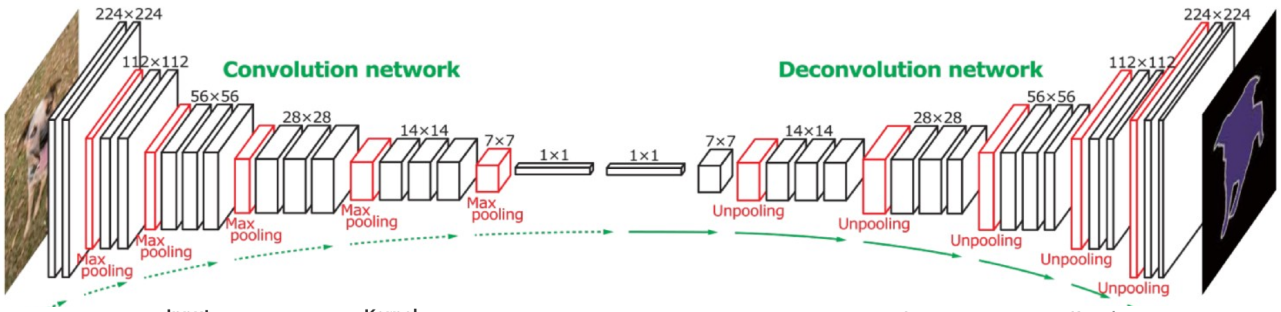
Solving the problem of segmentation by CNNs

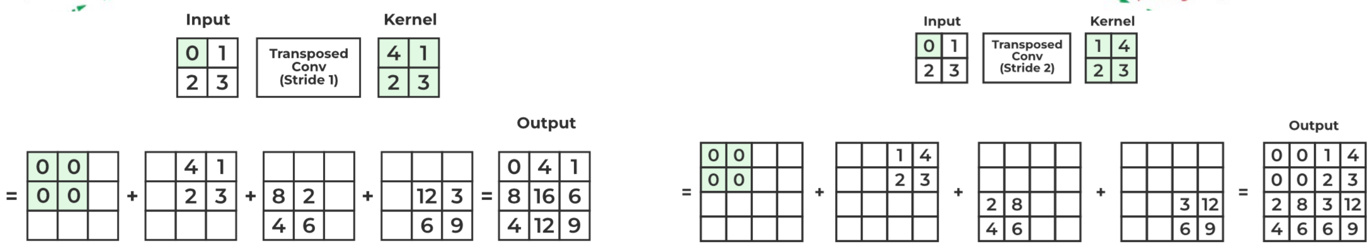
H. Noh, S. Hong and B. Han, "Learning Deconvolution Network for Semantic Segmentation," *2015 IEEE International Conference on Computer Vision (ICCV)*

Problem:

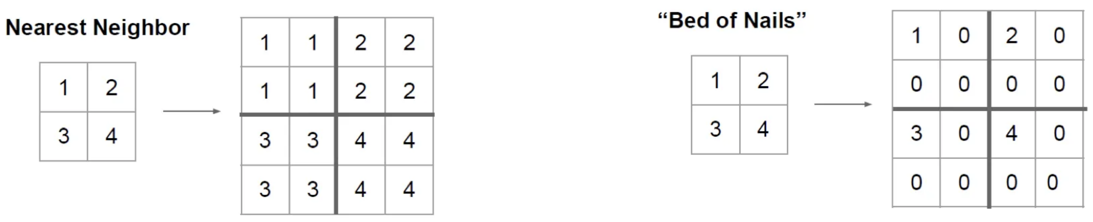


The network: VGG 16 is used as a convolution layer. The deconvolution network contains unpooling and transpose convolution layers, which are organised in the reverse order with respect to the input convolution network. The key idea is that the bottleneck (4096 numbers) between the convolution and deconvolution network describes all substantial information that is contained in the picture. The pairs like in the figure above must be prepared for training. The right images are the required answer of the deconvolution network.





Transpose convolution with stride 1 (left figure) and stride 2 (right figure)

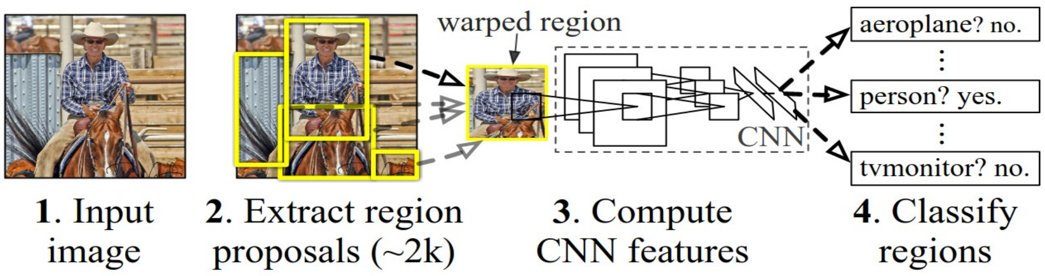


Unpooling

R-CNNs, Fast R-CNNs, Faster RCNNs (2014, 2015)

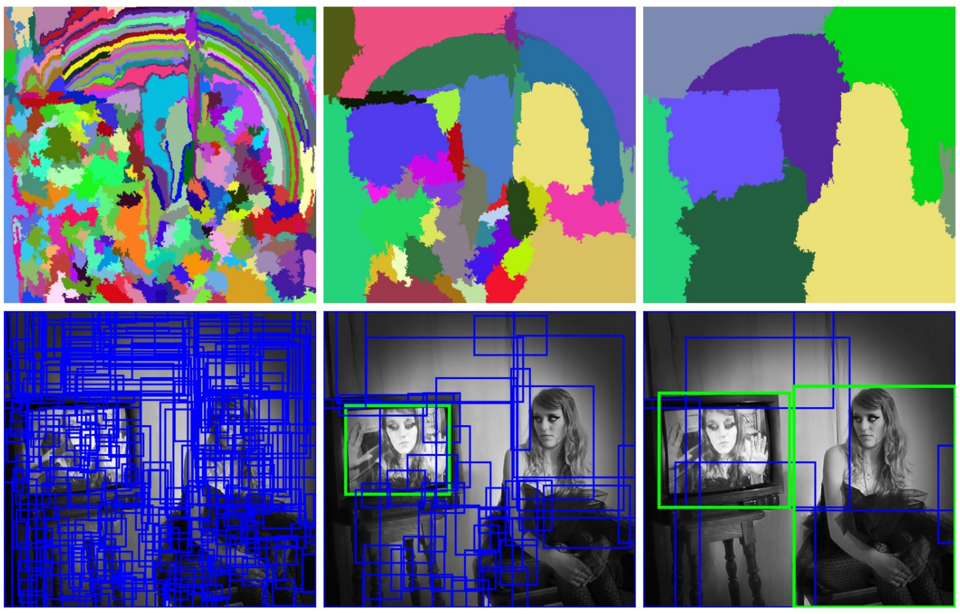
Girshick, R., J. Donahue, T. Darrell, and J. Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation." *CVPR '14 Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*. Pages 580-587. 2014

The R-CNN detector first generates region proposals using (approx 2000, selective search is used in the original paper). The proposal regions are cropped out of the image and resized and warped to window 227×227 pixels. The CNN features are then computed (Cafe implementation of the AlexNet, 4096 features). Finally, the region proposal bounding boxes are classified by a support vector machine (class-specific linear SVM) that is trained using CNN features.



The ideas of how the region proposals can be done

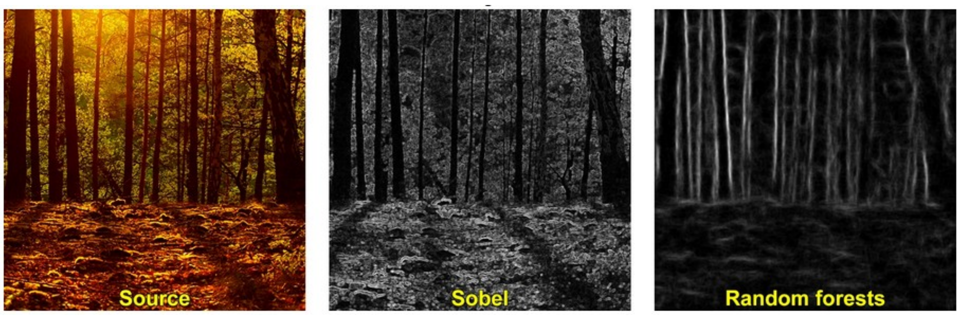
Selective search is inspired by the region growing method for image segmentation. In each iteration step two areas best fitting one to another (color, brightness, texture, shape) are connected together. Each new area may be an object., i.e. the corresponding region proposal is generated (see the figure below).

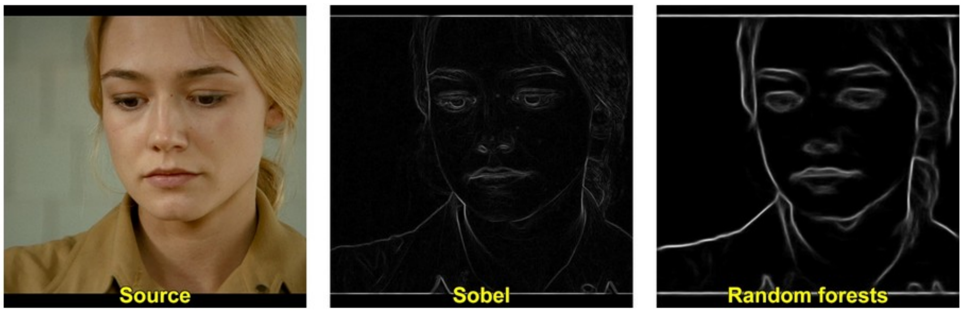


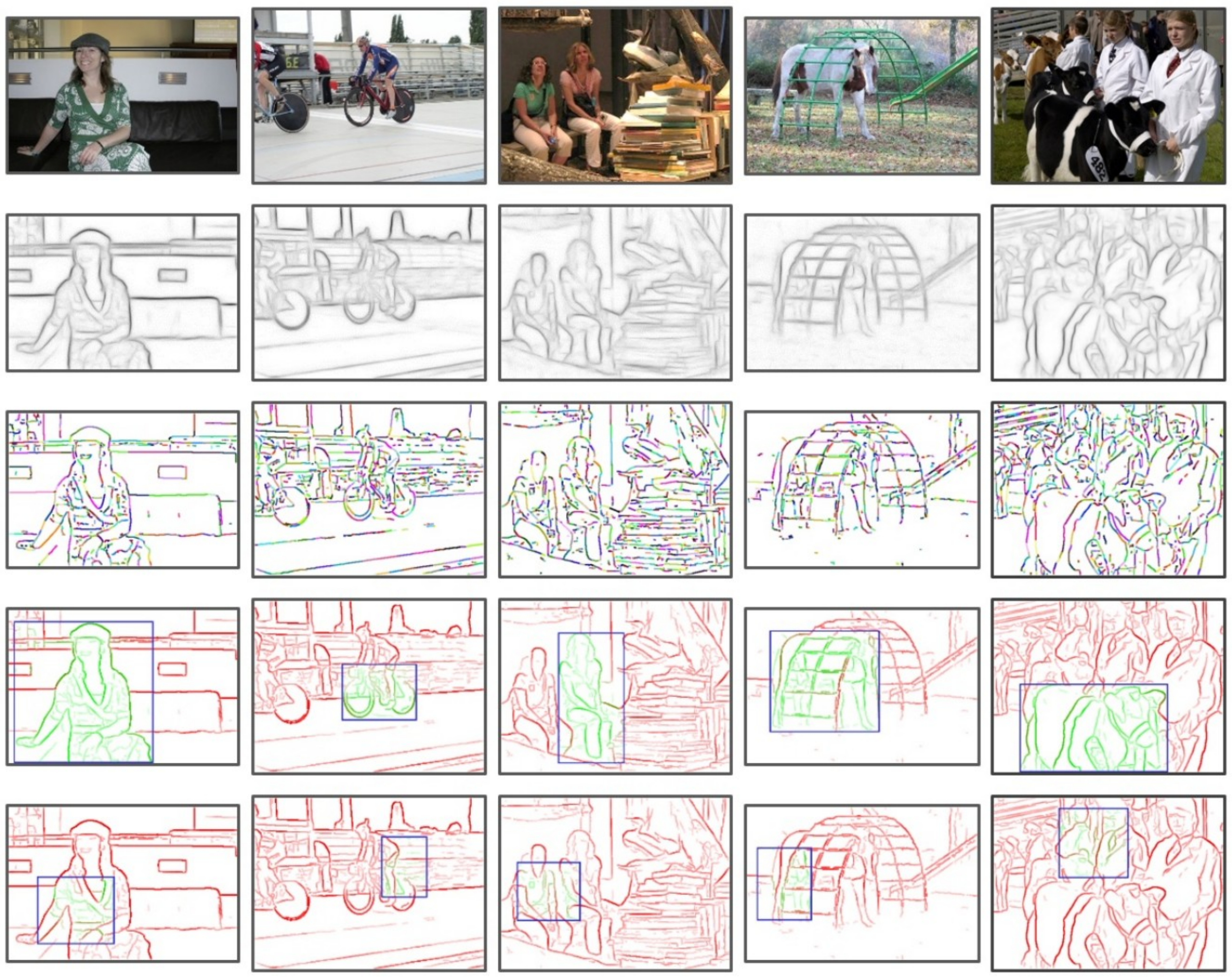
Edge bounding boxes: The candidate bounding boxes are sought for using a sliding window search over position, scale and aspect ratio. The length of contours that are fully enclosed by the box is evaluated, these contours increase the chance that an object is present in the box. On the contrary, the contours running across the boundary of the decrease this chance. Top-ranked region proposals are used.

In this context (edge bounding boxes), new approaches for detecting edges have been used (Structured Forests for Fast Edge Detection). Random forests were trained on the BSDS500 segmentation and NYU Depth datasets using 16x16 window (you can check it also in OpenCV). The difference between the results obtained by the usual algorithm and the algorithm using random forests can be seen in the figure below.

https://docs.opencv.org/3.4/d0/da5/tutorial_ximgproc_prediction.html



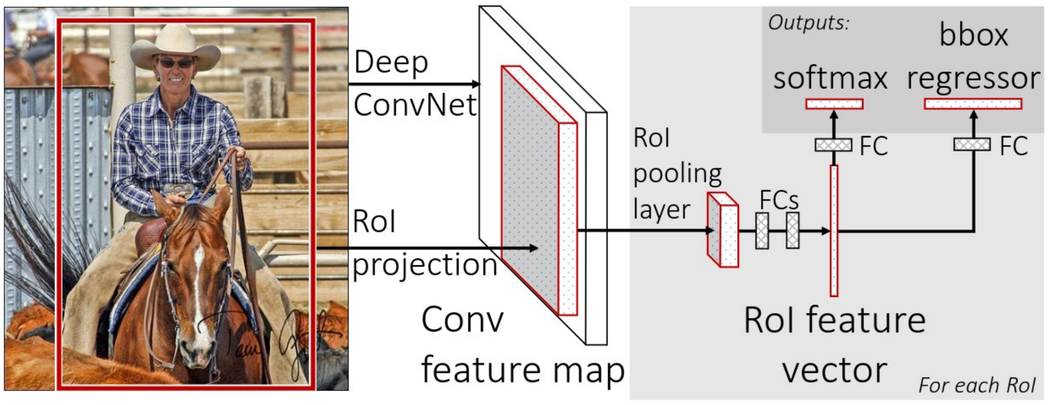



Illustrative examples showing from top to bottom (first row) original image, (second row) Structured Edges, (third row) edge groups, (fourth row) example correct bounding box and edge labeling, and (fifth row) example incorrect boxes and edge labeling. Green edges are predicted to be part of the object in the box while red edges are not.

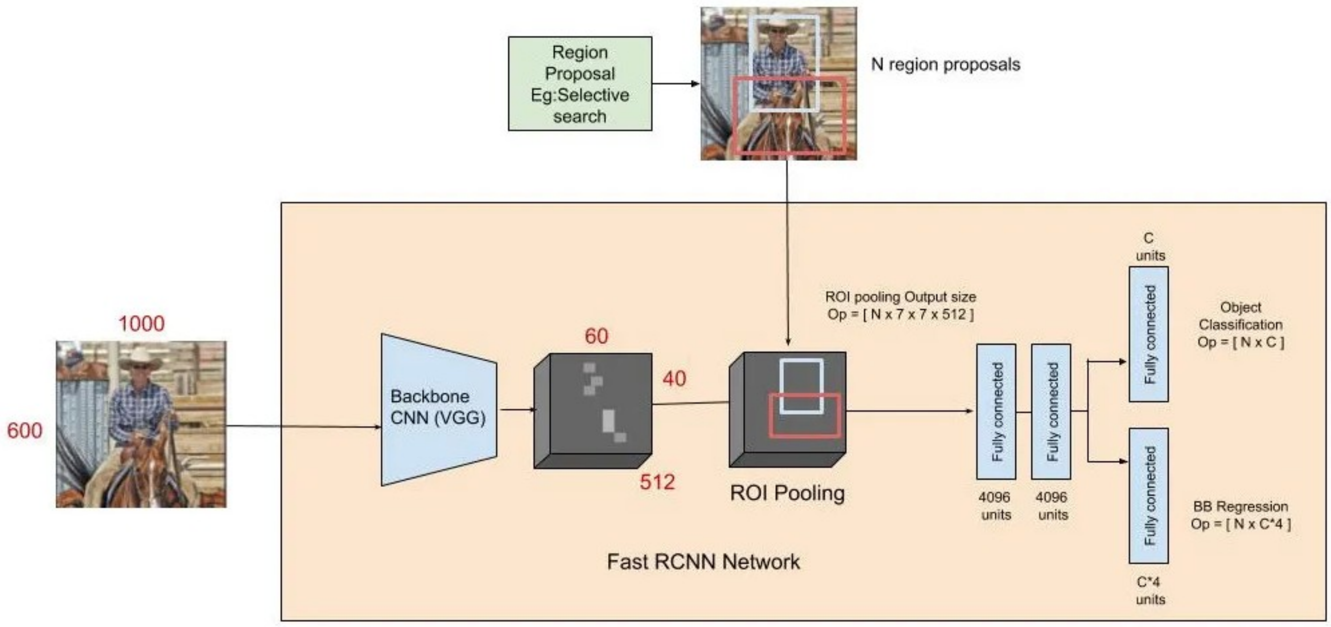
Fast R-CNN

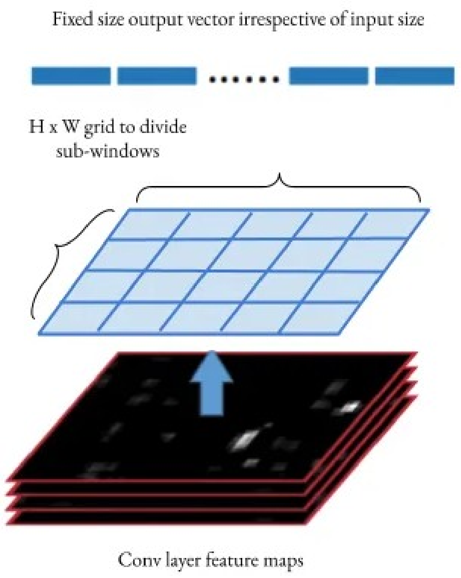
Girshick, Ross. "Fast r-cnn." *Proceedings of the IEEE International Conference on Computer Vision*. 2015

As in the R-CNN detector , the Fast R-CNN detector also uses an algorithm like Edge Boxes or Selective Search to generate region proposals. Unlike the R-CNN detector, which crops and resizes region proposals, the Fast R-CNN detector processes the entire image. Whereas an R-CNN detector must classify each region, Fast R-CNN pools CNN features corresponding to each region proposal. Fast R-CNN is more efficient than R-CNN, because in the Fast R-CNN detector, the computations for overlapping regions are shared.



Fast R-CNN architecture. The entire image (e.g. 1000×600) is fed into the backbone CNN (e.g. VGG) and the features from the last convolution layer are obtained (size 60×40×512, for example). Each RoI is pooled into a fixed-size (e.g. 7×7) feature map. RoI max pooling works by dividing the h × w RoI window into an H×W grid of sub-windows of approximate size h/H×w/W and then max-pooling the values in each sub-window into the corresponding output grid cell. Pooling is applied independently to each feature map channel, as in standard max pooling. The final feature vector is further processed in two fully connected layers. The network has two output vectors per RoI: softmax probabilities and per-class bounding-box regression offsets.




Fixed size output vector irrespective of input size
H x W grid to divide sub-windows
Conv layer feature maps

Pretrained backbone is used. The architecture is then trained end-to-end with a multi-task loss (classification and localisation error). Classification gives probabilities for every ROI over (K+1) categories (since the “background class” is considered too). The classification loss is given by -log(p_{true}) which is the log loss for the true class.

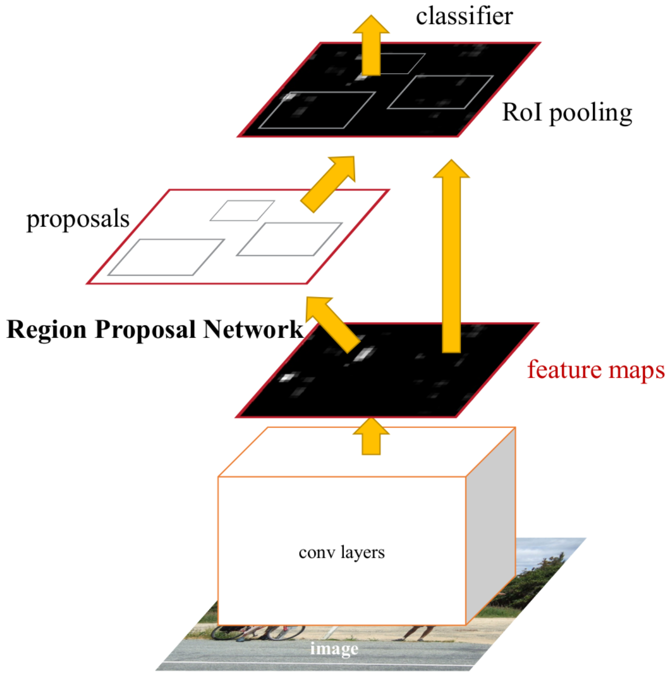
The regression branch produces 4 bounding box regression offsets (x,y,w,h), the localisation error is given by the sum of the smooth L1 between obtained and ground truth coordinates for x, y, w, h.

Faster R-CNN

Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." *Advances in Neural Information Processing Systems* . Vol. 28, 2015.

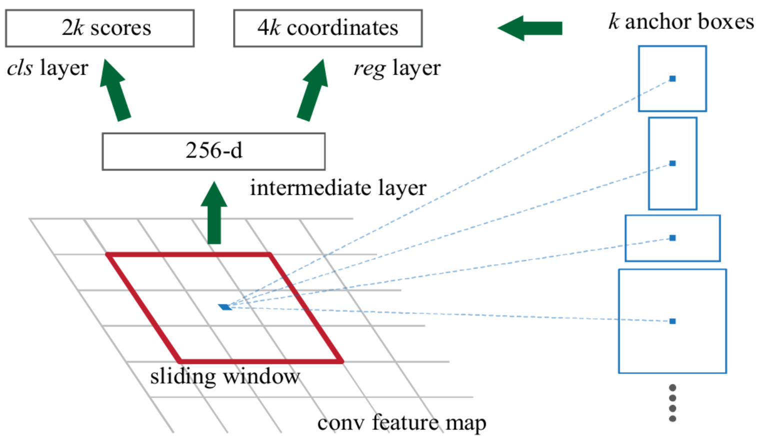
The Faster R-CNN detector adds a region proposal network (RPN) to generate region proposals directly in the network instead of using an external algorithm like Edge Boxes or Selective Search. RPN takes an input image (of any size) and outputs a set of rectangular object proposals, each with an objectness score. Generating region proposals in the network is faster than in Selective Search or Edge Boxes.

The RPN uses Anchor Boxes for object detection.



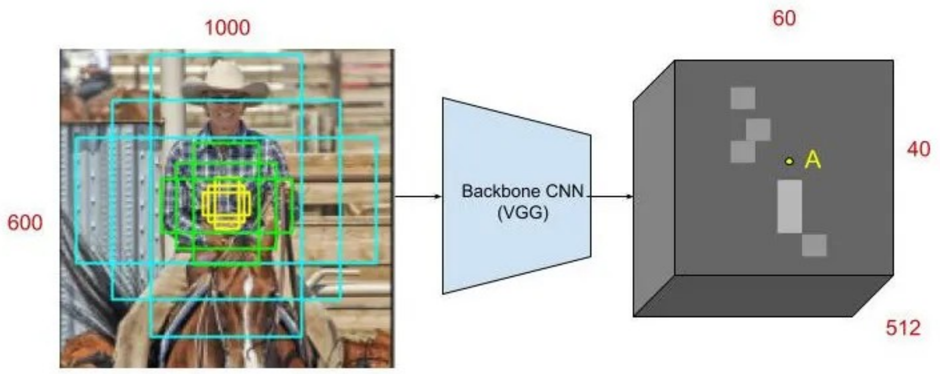
In the backbone convolutional neural network, a big image (say 1000×600 pixels) is transformed into the image containing features (size 60×40×512, say).

For every point in the output feature map, the RPN has to learn whether an object is present in the input image at its corresponding location and expected size. This is done by placing a set of “Anchors” on the input image for each location on the output feature map from the backbone network. The authors used 3 scales/sizes of box area 128, 256, 512, and 3 aspect ratios of 1:1, 1:2 and 2:1. For each position in the feature map, they thus have 9 anchor boxes.



Using 3×3 sliding window can also be understood as computing convolution with the 3×3×N (N=512, for example). It can also be easily seen that the anchors are translation invariant, as the authors note. Let k be the number of anchor boxes, the output of RPN is created by 2k scores (object/nonobject probability) and 4k coordinates (x, y, w, h for each anchor box).

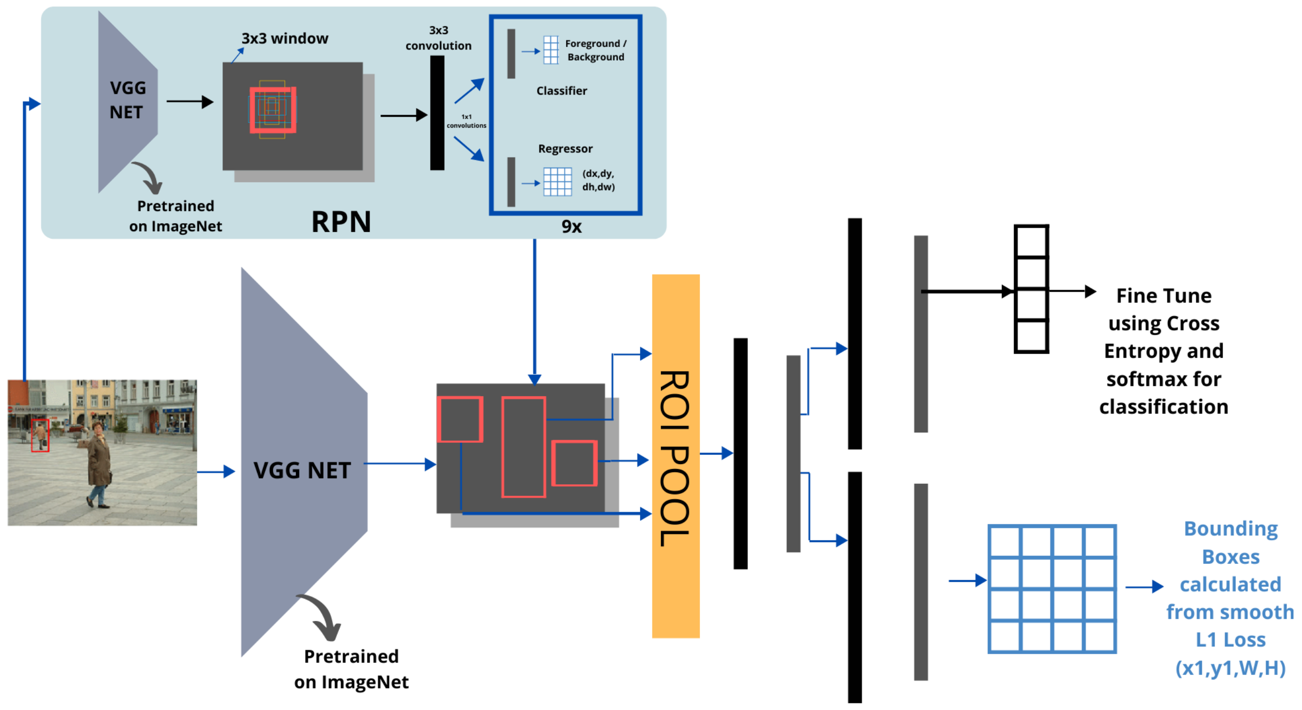
You can also see the positions that are considered for each pint in the feature map as is depicted in the image below.



The RPN is trained end-to-end by back-propagation and stochastic gradient descent (SGD). Each mini-batch arises from a single image that contains many positive and negative example anchors. It would be possible to optimize for the loss functions of all anchors, but this would bias towards negative samples as they are dominate. Instead, 256 anchors in an image are sampled randomly to compute the loss function of a mini-batch, where the sampled positive and negative anchors have a ratio of up to 1:1. If there are fewer than 128 positive samples in an image, the mini-batch is padded with negative ones. The training loss for the RPN is multi-task loss (erroneous object/nonobject classification as well as incorrect determining the box in the case of object are penalised).

Alternating training: The RPN is trained independently first. The backbone CNN for this task is initialized with weights from a network trained for an ImageNet classification task, and is then fine-tuned for the region proposal task. In the second step, a separate detection network is trained by Fast R-CNN using the proposals generated by RPN from the previous step. This detection network is also initialized by the ImageNet-pre-trained model. At this point the two networks do not share convolutional layers. In the third step, we use the detector network to initialize RPN training, but we fix the shared convolutional layers and only fine-tune the layers unique to RPN. Now the two networks share the convolutional layers. Finally, keeping the shared convolutional layers fixed, we fine-tune the unique layers of Fast R-CNN.

Overall structure of faster R-CNN.



If the alternating training is used, both VGG nets are replaced by only one VGG net.

YOLO v1 (2015)

J. Redmon, S. Divvala, R. Girshick and A. Farhadi, You Only Look Once: Unified, Real-Time Object Detection (2015), https://arxiv.org/pdf/1506.02640.pdf

A note: Although YOLO is now in v9 (February 2024), it seems appropriate to start from YOLO v1 since it is more readable than the newer versions.

Other detection methods like R-CNN and Fast(er) R-CNN are primarily image classifier networks which are used for object detection with the following steps.

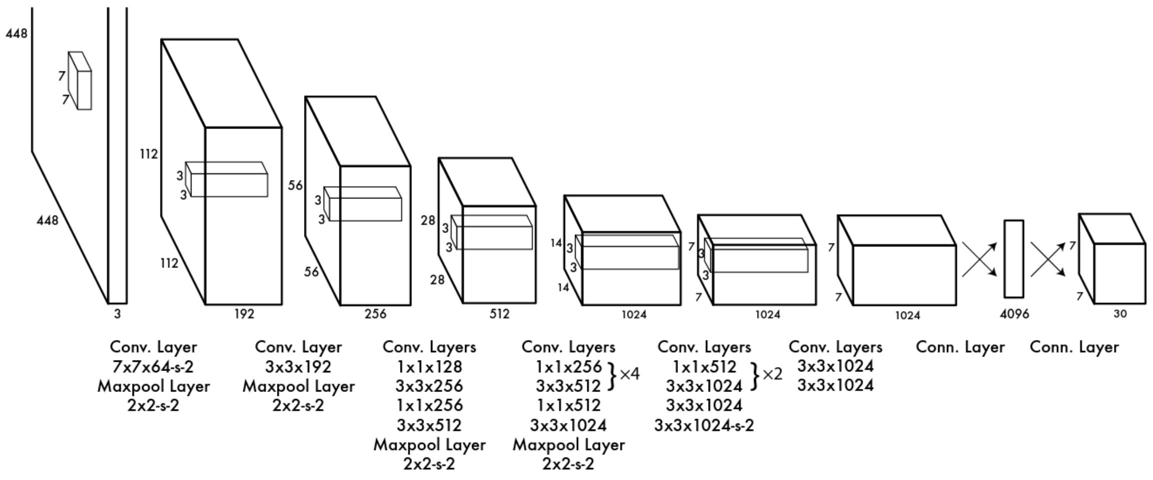
1. Use Region Proposal method to generate potential bounding boxes in an image
2. Run the classifier on these boxes
3. After classification, perform post processing to tighten the boundaries of the bounding boxes, remove duplicates

These pipelines prove to be complex and bulky and hard to optimize as each component needs to be trained separately. Also such a pipeline is often very slow during inference.

YOLO is different from all these methods as it treats the problem of image detection as a regression problem (YOLO v1) rather than a classification problem and supports a single convolutional neural network to perform all the above mentioned tasks. (Explanation: Regression predicts a continuous value, while classification predicts a categorical value.)

YOLO network uses features from the entire image to predict each bounding box. It predicts all bounding boxes across all classes for an image simultaneously. This means the YOLO network reasons globally about the full image and all the objects in the image.

YOLO v1 divides the input image into an $S \times S$ grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. Formally we define confidence as $\Pr(\text{Object}) * \text{IOU}^{\text{truth}}_{\text{pred}}$. (Intersection Over Union). If no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal IOU between the predicted box and the ground truth.
Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally, the confidence prediction represents the IOU between the predicted box and any ground truth box. Each grid cell also predicts C conditional class probabilities, $\Pr(\text{Class}_i|\text{Object})$. These probabilities are conditioned on the grid cell containing an object.



448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 }×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 }×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

For evaluating YOLO on PASCAL VOC, the authors used S=7, B=2. PASCAL VOC had 20 labelled classes so C=20. The final prediction is a 7×7×30 tensor.

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

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

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

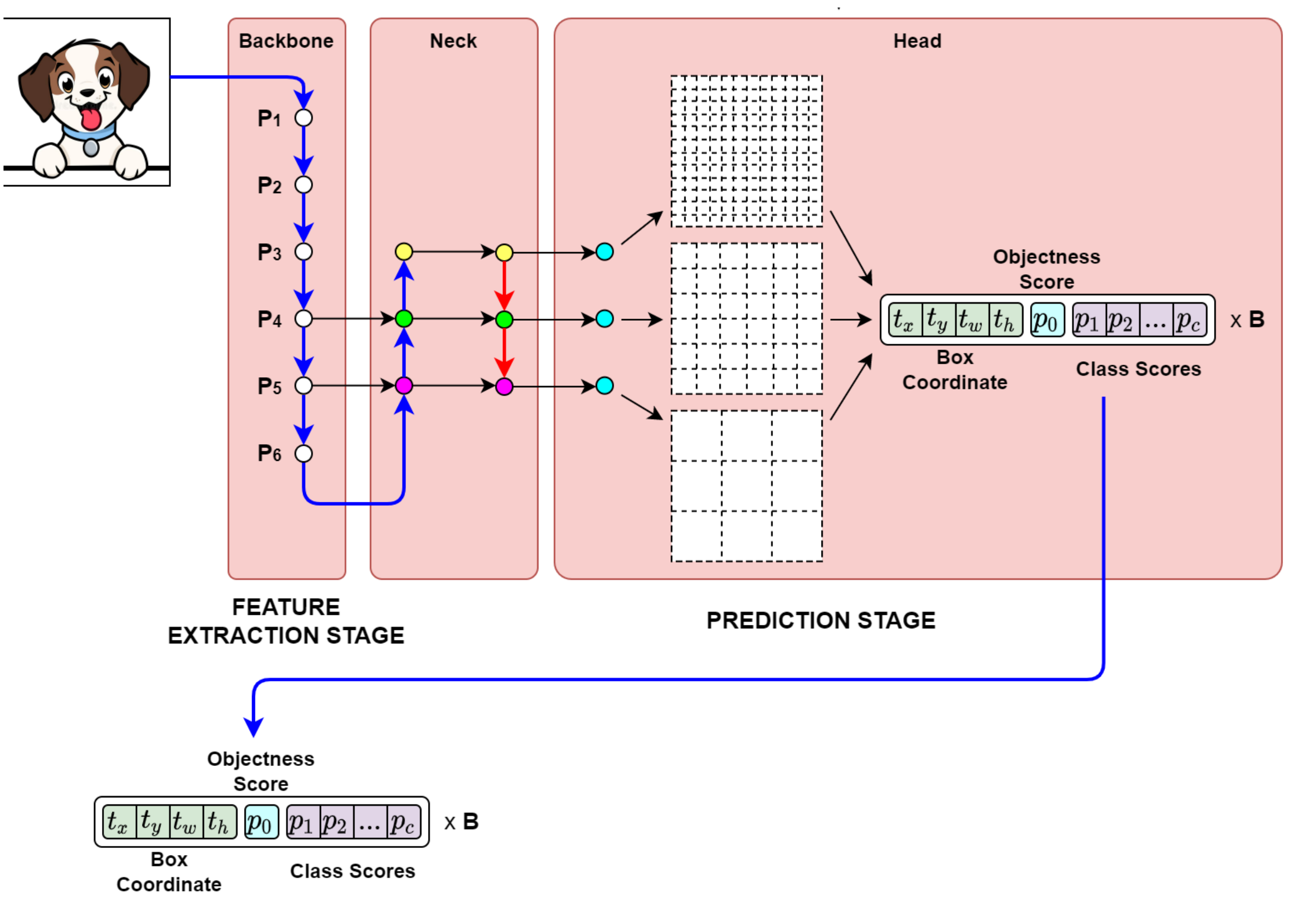
$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

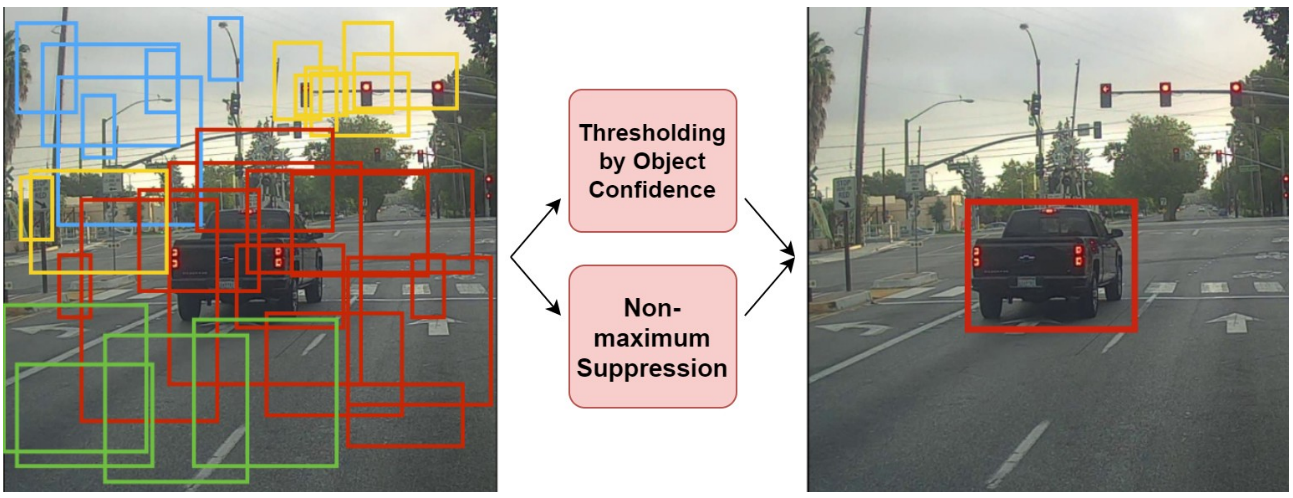
$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

1. This term penalize bad localization of center of cells
2. This term penalize the bounding box with inacurate height and width. The square root is present so that errors in small bounding boxes are more penalizing than errors in big bounding boxes.
3. This term tries to make the confidence score equal to the IOU between the object and the prediction when there is one object
4. This term tries to make confidence score close to 0 there is no object in the cell
5. This is a simple classification loss (not explained in the article)

YOLO v4

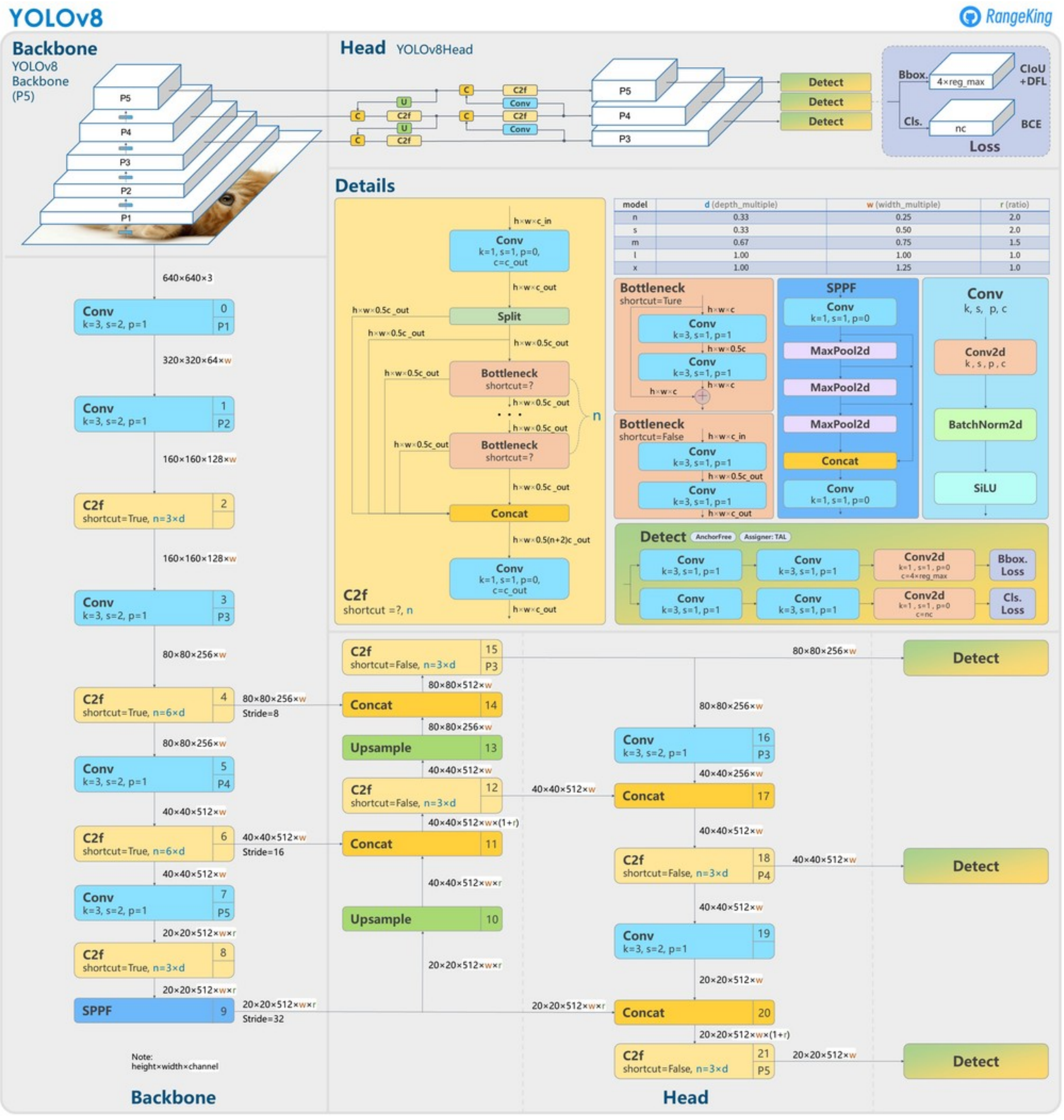
Although it is the YOLO v8 (or 9) which is in the center of our interest now (2024), I managed to get this explaining picture for YOLO v4. The basic backbone, neck, and head structure remains also in the higher YOLO versions. The detailed stucture however differs to certain extent. (For example, also the format of the result is now little bit different as will be shown later.)





YOLO v8

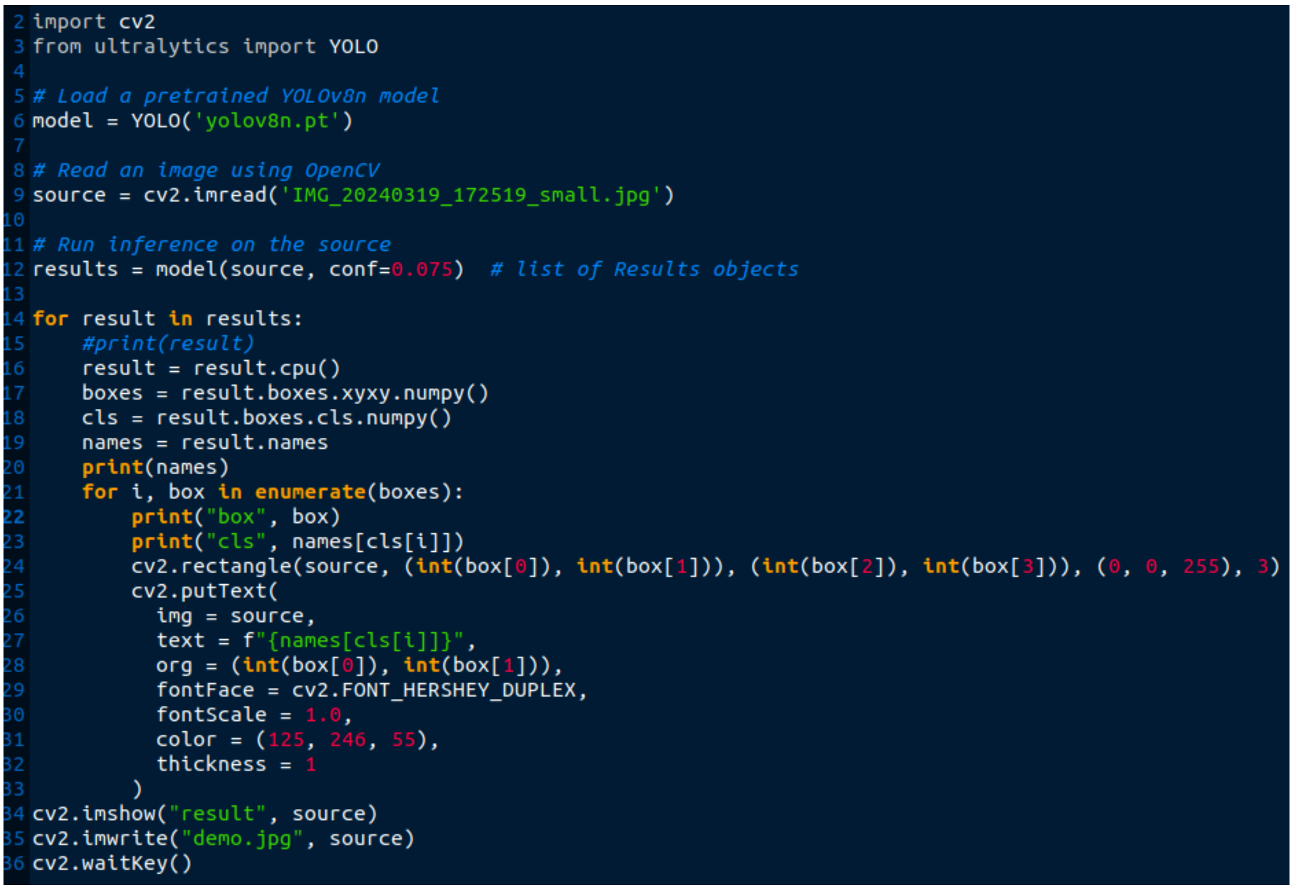
The architecture of YOLO v8 (the rough view in the top, a more detailed view to the left and in the bottom, the details of the particular blocks in the middle, see also the table containing parameters).



model	d (depth_multiple)	w (width_multiple)	r (ratio)
n	0.33	0.25	2.0
s	0.33	0.50	2.0
m	0.67	0.75	1.5
l	1.00	1.00	1.0
x	1.00	1.25	1.0

CIoU is an improved version of IoU loss function (1-IoU). DFL provides gradients that can guide the learning of boundary predicted features, thus helping to reduce the bbox loss. BCE is the binary cross-entropy loss.

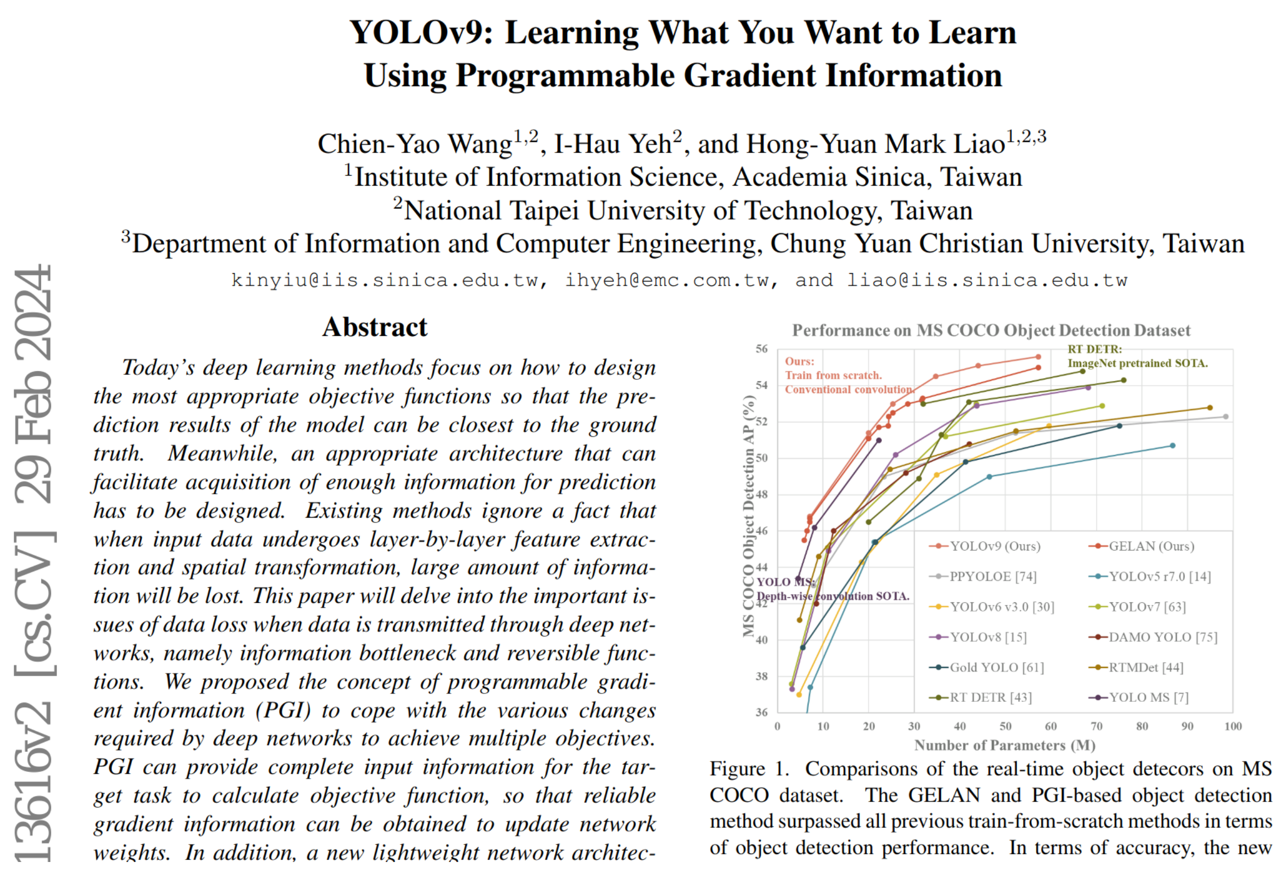
If you use the pretrained YOLO model, object classification may be quite easy. The sample images from the beginning of this text have been prepared with this program.



```python
import cv2
from ultralytics import YOLO

# Load a pretrained YOLOv8n model
model = YOLO('yolov8n.pt')

# Read an image using OpenCV
source = cv2.imread('IMG_20240319_172519_small.jpg')

# Run inference on the source
results = model(source, conf=0.075)  # list of Results objects

for result in results:
    #print(result)
    result = result.cpu()
    boxes = result.boxes.xyxy.numpy()
    cls = result.boxes.cls.numpy()
    names = result.names
    print(names)
    for i, box in enumerate(boxes):
        print("box", box)
        print("cls", names[cls[i]])
        cv2.rectangle(source, (int(box[0]), int(box[1])), (int(box[2]), int(box[3])), (0, 0, 255), 3)
        cv2.putText(
          img = source,
          text = f"{names[cls[i]]}",
          org = (int(box[0]), int(box[1])),
          fontFace = cv2.FONT_HERSHEY_DUPLEX,
          fontScale = 1.0,
          color = (125, 246, 55),
          thickness = 1
        )
cv2.imshow("result", source)
cv2.imwrite("demo.jpg", source)
cv2.waitKey()
```

A bigger example can be found here: http://mrl.cs.vsb.cz//people/sojka/zao.zip

A note on **YOLO v9**: February 2024, https://github.com/WongKinYiu/yolov9
https://arxiv.org/pdf/2402.13616.pdf



YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information

Chien-Yao Wang[1,2], I-Hau Yeh[2], and Hong-Yuan Mark Liao[1,2,3]
[1]Institute of Information Science, Academia Sinica, Taiwan
[2]National Taipei University of Technology, Taiwan
[3]Department of Information and Computer Engineering, Chung Yuan Christian University, Taiwan
kinyiu@iis.sinica.edu.tw, ihyeh@emc.com.tw, and liao@iis.sinica.edu.tw

Abstract

Today's deep learning methods focus on how to design the most appropriate objective functions so that the prediction results of the model can be closest to the ground truth. Meanwhile, an appropriate architecture that can facilitate acquisition of enough information for prediction has to be designed. Existing methods ignore a fact that when input data undergoes layer-by-layer feature extraction and spatial transformation, large amount of information will be lost. This paper will delve into the important issues of data loss when data is transmitted through deep networks, namely information bottleneck and reversible functions. We proposed the concept of programmable gradient information (PGI) to cope with the various changes required by deep networks to achieve multiple objectives. PGI can provide complete input information for the target task to calculate objective function, so that reliable gradient information can be obtained to update network weights. In addition, a new lightweight network architec-

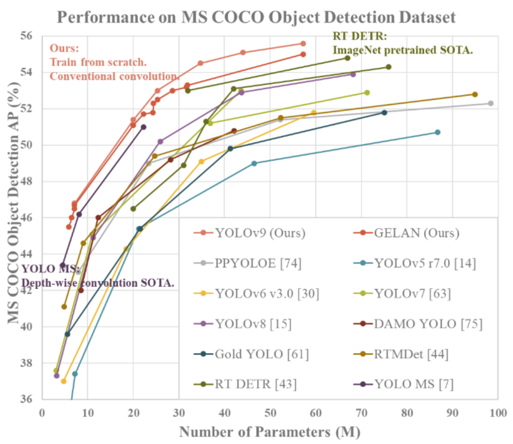
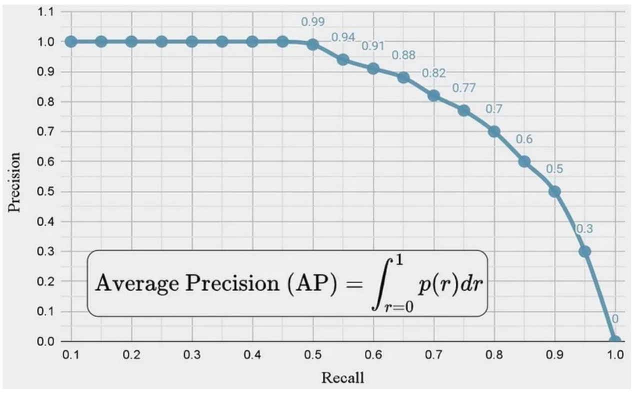


Figure 1. Comparisons of the real-time object detecors on MS COCO dataset. The GELAN and PGI-based object detection method surpassed all previous train-from-scratch methods in terms of object detection performance. In terms of accuracy, the new

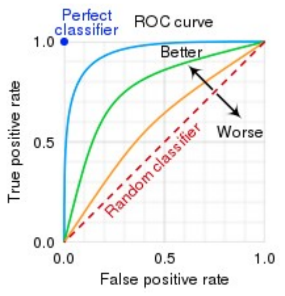
A note on measuring the results:

- *True Positive*: The model predicted that a bounding box exists at a certain position (positive) and it was correct (true).
- *False Positive*: The model predicted that a bounding box exists at a particular position (positive) but it was wrong (false).
- *False Negative*: The model did not predict a bounding box at a certain position (negative) and it was wrong (false) i.e. a ground truth bounding box existed at that position.
- *True Negative*: The model did not predict a bounding box (negative) and it was correct (true). This corresponds to the background, the area without bounding boxes, and is not used to calculate the final metrics.

Precision = TP/(TP+FP), Recall = TP/(TP+FN)



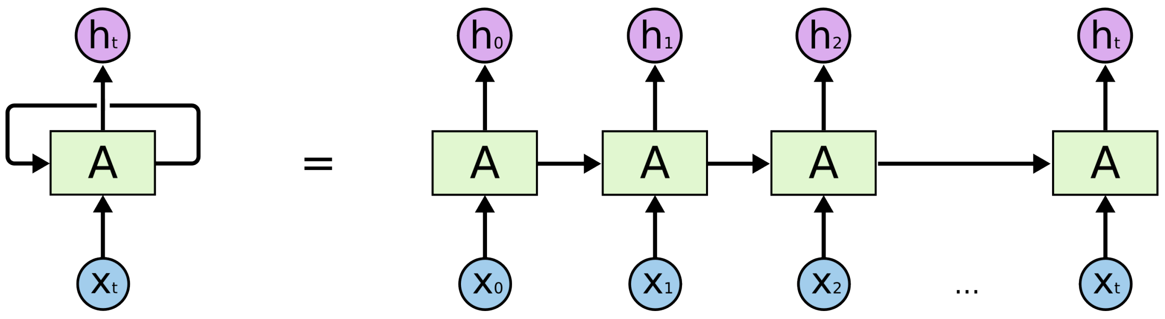
$$\text{Average Precision (AP)} = \int_{r=0}^{1} p(r)dr$$

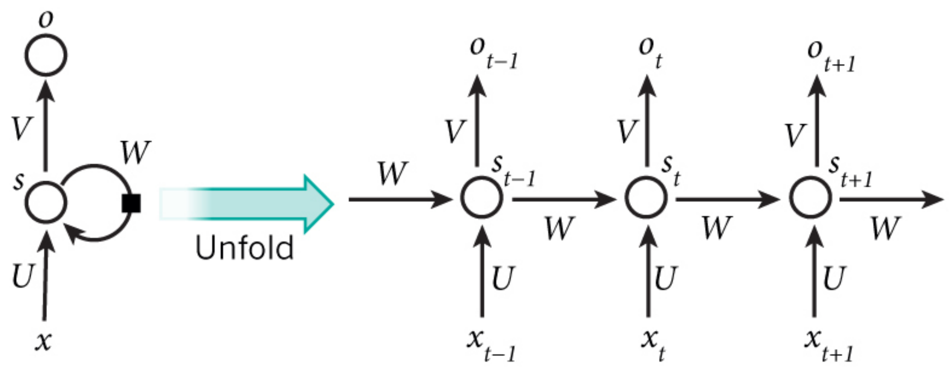


Average precision (AP) is the area under the PR curve above. (For completeness: Another possibility is to use ROC curve (right figure).

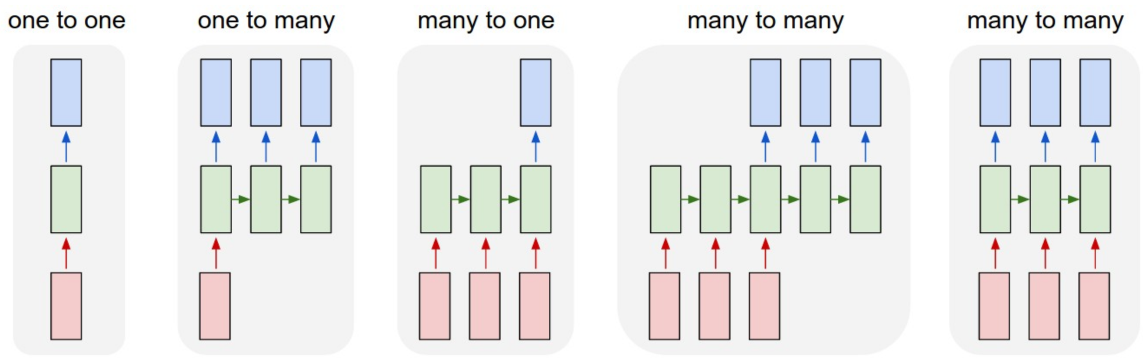
Another note: *Ablation study* means removing certain components to understand their contribution to the whole.

Recurrent Networks





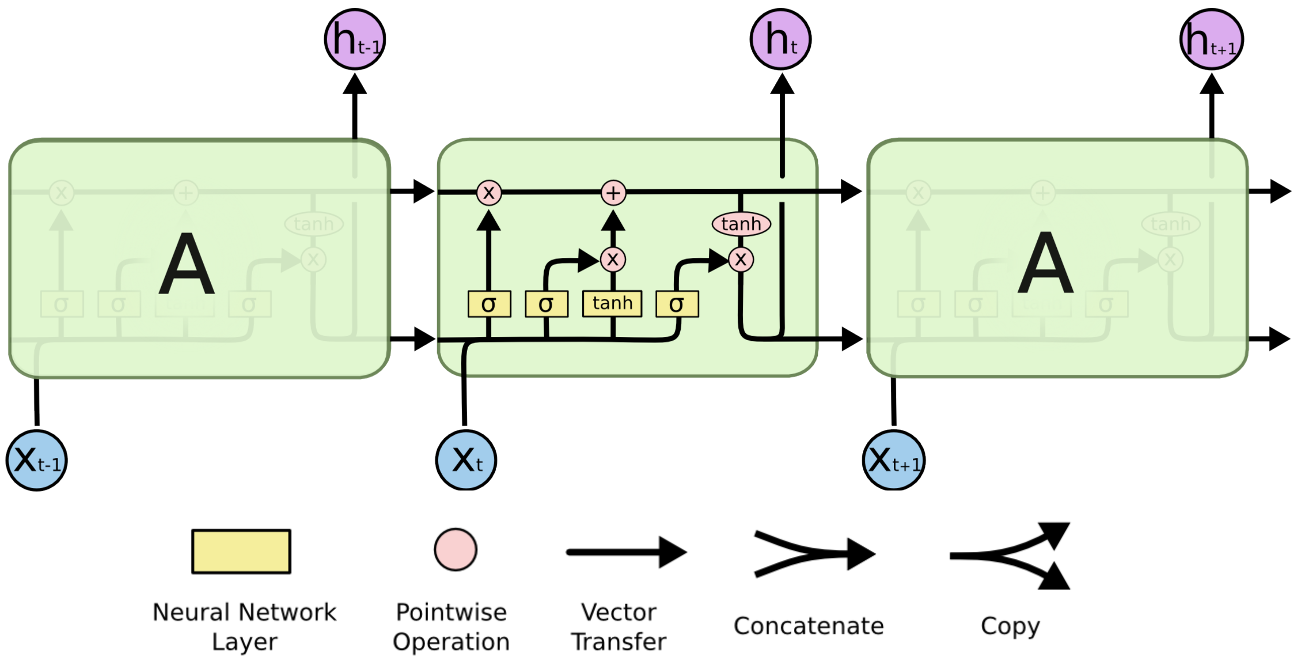
- x_t is the input at time step t. For example, x_1 could be a one-hot vector corresponding to the second word of a sentence.
- s_t is the hidden state at time step t. It's the "memory" of the network. s_t is calculated based on the previous hidden state and the input at the current step: $s_t = f(Ux_t + Ws_{t-1})$. The function f usually is a nonlinearity such as tanh or ReLU. s_{-1}, which is required to calculate the first hidden state, is typically initialized to all zeroes.
- o_t is the output at step t. For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary. $o_t = \mathrm{softmax}(Vs_t)$.

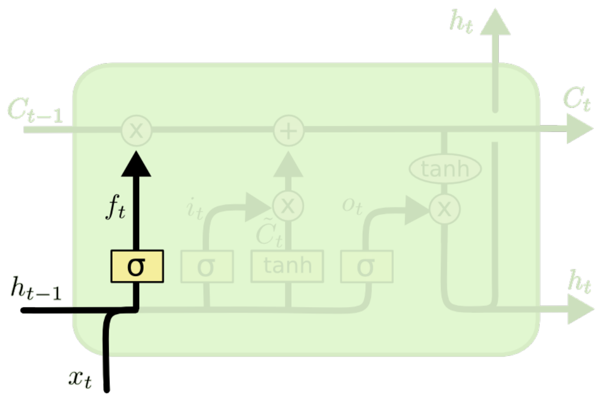


Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state. From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). **(2)** Sequence output (e.g. image captioning takes an image and outputs a sentence of words). **(3)** Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). **(4)** Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). **(5)** Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

Long Short-Term Memory Recurrent Neural Network

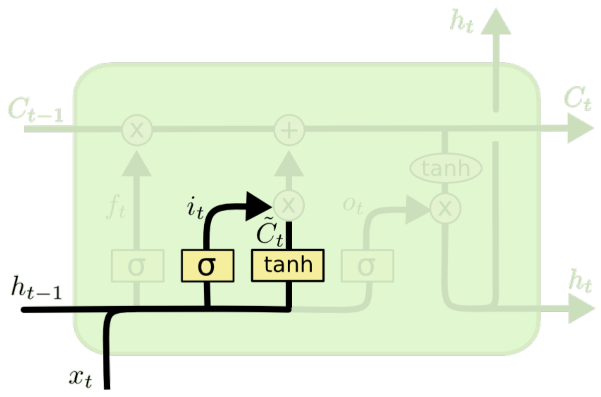
Hochreiter, S., & Schmidhuber, J"urgen. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780.
http://colah.github.io/posts/2015-08-Understanding-LSTMs/





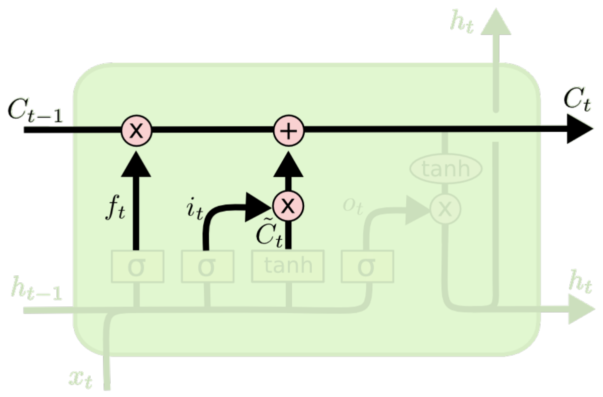
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \;+\; b_f\right)$$

The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at $ht-1$ and xt, and outputs a number between 0 and 1 for each number in the cell state $Ct-1$. A 1 represents "completely keep this" while a 0 represents "completely get rid of this."



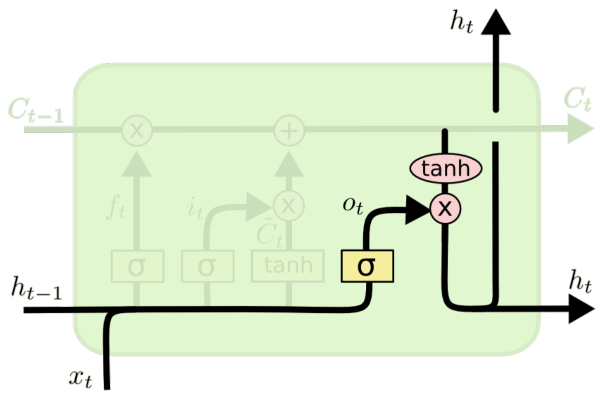
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \;+\; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \;+\; b_C)$$

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, $C\text{~}t$, that could be added to the state. In the next step, we'll combine these two to create an update to the state.



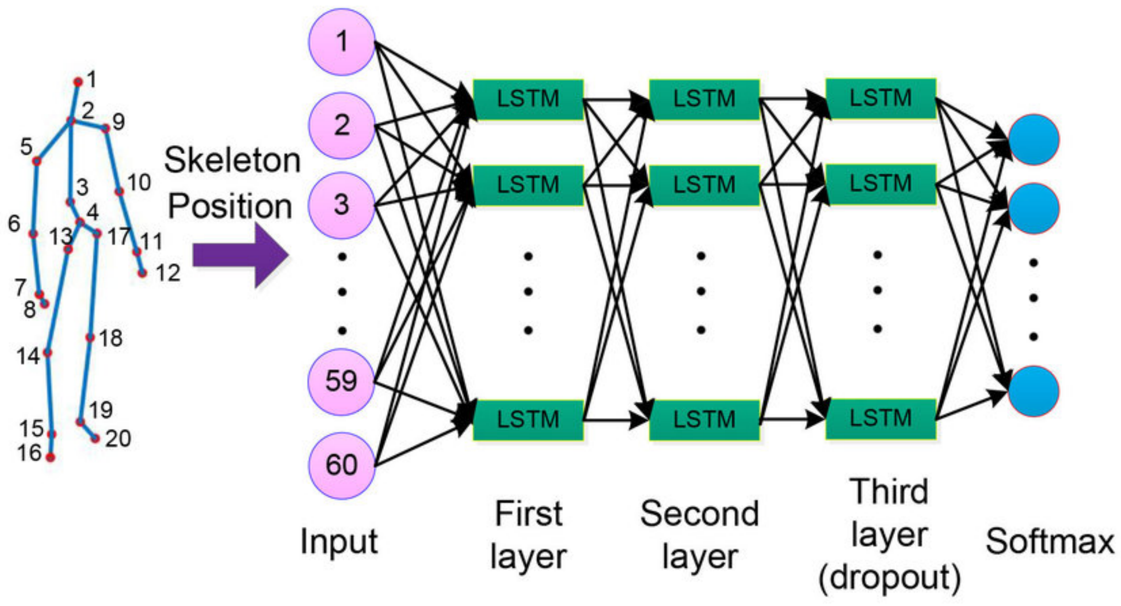
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Updating the old cell state, *Ct*−1, into the new cell state *Ct.* We multiply the old state by *ft* forgetting the things we decided to forget earlier. Then we add *it*∗*C*~*t*. This is the new candidate values, scaled by how much we decided to update each state value.



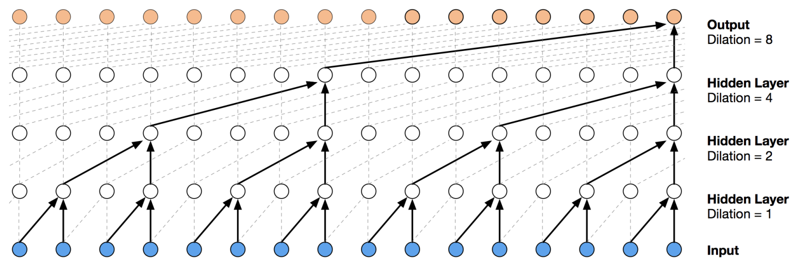
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

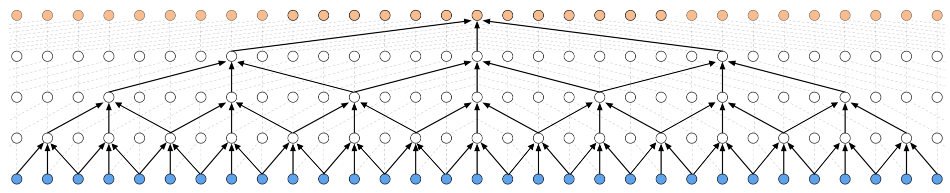
Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through `tanh` (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

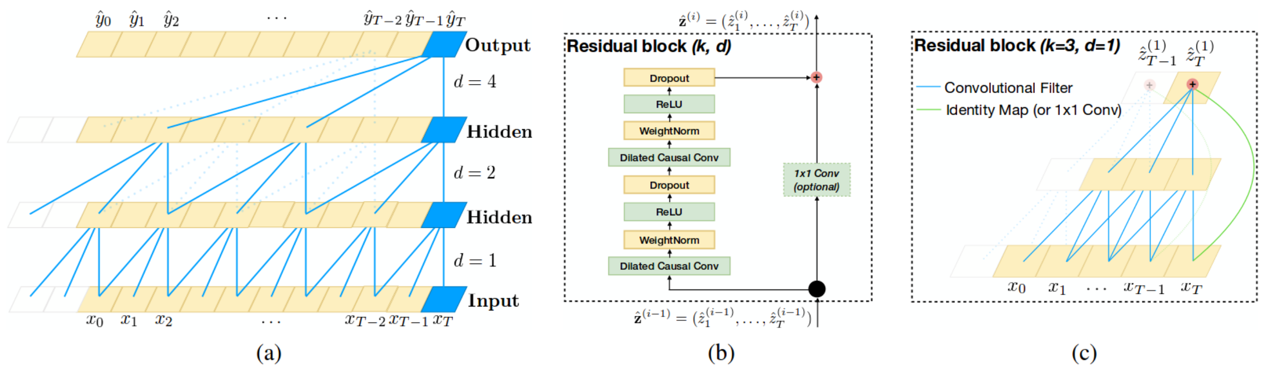


Temporal CNs

Shaojie Bai, J. Zico Kolter, Vladlen Koltun: An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling (2018)







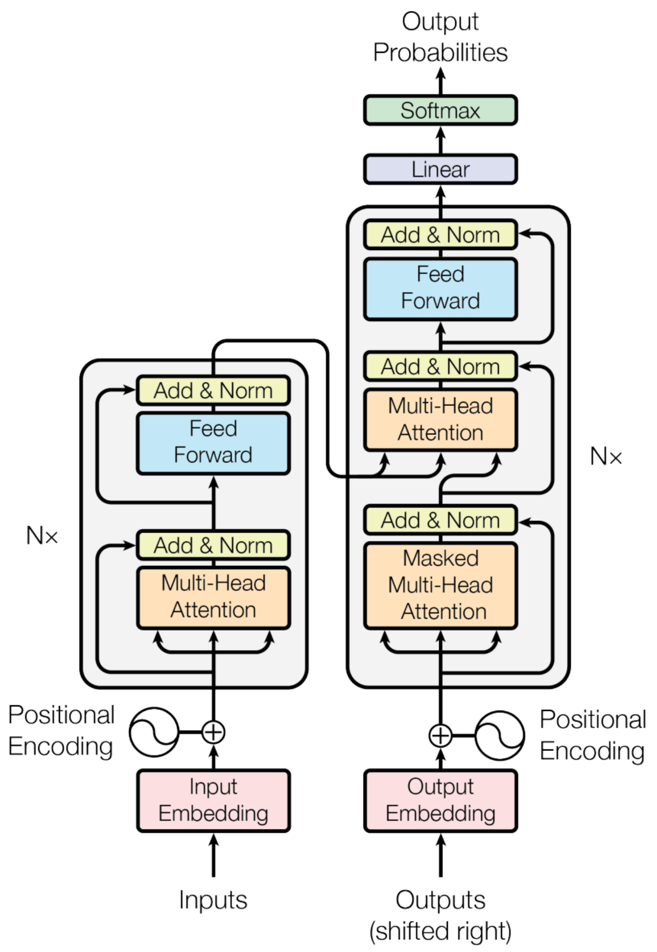
$\hat{y}_0$ $\hat{y}_1$ $\hat{y}_2$ ... $\hat{y}_{T-2}$ $\hat{y}_{T-1}$ $\hat{y}_T$
Output
$d = 4$
Hidden
$d = 2$
Hidden
$d = 1$
Input
x_0 x_1 x_2 ... x_{T-2} x_{T-1} x_T
$\hat{\mathbf{z}}^{(i)} = (\hat{z}_1^{(i)}, \dots, \hat{z}_T^{(i)})$
Residual block (k, d)
Dropout
ReLU
WeightNorm
Dilated Causal Conv
Dropout
ReLU
WeightNorm
Dilated Causal Conv
1x1 Conv (optional)
$\hat{\mathbf{z}}^{(i-1)} = (\hat{z}_1^{(i-1)}, \dots, \hat{z}_T^{(i-1)})$
Residual block (k=3, d=1)
$\hat{z}_{T-1}^{(1)}$ $\hat{z}_T^{(1)}$
Convolutional Filter
Identity Map (or 1x1 Conv)
x_0 x_1 ... x_{T-1} x_T
(a)
(b)
(c)

Figure 1. Architectural elements in a TCN. (a) A dilated causal convolution with dilation factors $d = 1, 2, 4$ and filter size $k = 3$. The receptive field is able to cover all values from the input sequence. (b) TCN residual block. An 1x1 convolution is added when residual input and output have different dimensions. (c) An example of residual connection in a TCN. The blue lines are filters in the residual function, and the green lines are identity mappings.

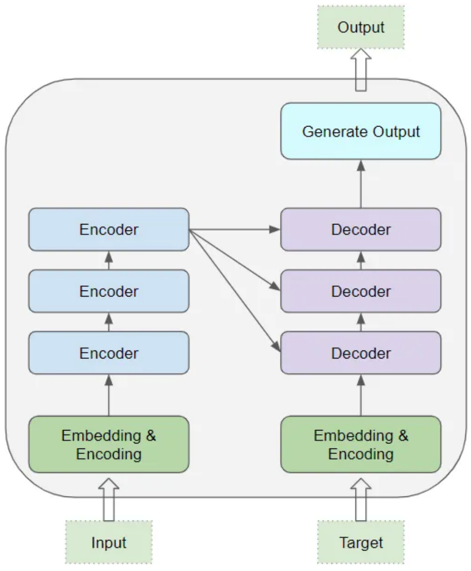
Vision Transformers & Self-Attention Networks

Creating the vision transformers was inspired by self-attention networks that are used in the areas like natural language processsing (NLP) and chatbots (the self-attention networks are alternatively also called the transformers). In NLP the goal is to generate a sequence of words, in vision, in contrast, the goal is only to classify objects. In vision, therefore, we will not use the whole architecture, but only a part of it.

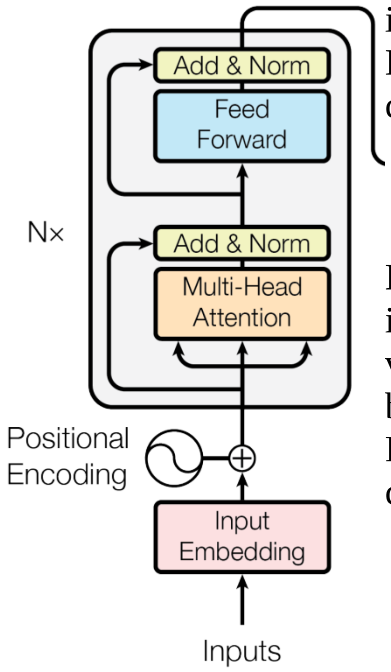
We start, however, from a brief view on the architecture for NLP as it was presented in Ashish Vaswani et al.: Attention Is All You Need (2017), https://arxiv.org/abs/1706.03762. The following figure is taken from the mentioned paper. We will focus mainly on the encoder part since it is also used in vision transformers.



How to understand N_x (here N_x=3)



Motivation: In NLP (e.g. during translation), one sentence in a certain language should be used for generating the equivalent sentence in an another language. Recurrent NN and LSTM networks were commonly used for this task before. The problem is that they process the input sequence sequentially. It causes that an important word that was pronounced longer before (e.g. at the beginning of the sentence, if the end of the sequence is being processed now) may be forgotten, which makes that the content of the sentence in the goal language is not expressed properly. Moreover, the sequential character of recurrent NNs, including LSTMs, brings also computational problems. Long sequentially processed sequences require long computational times.

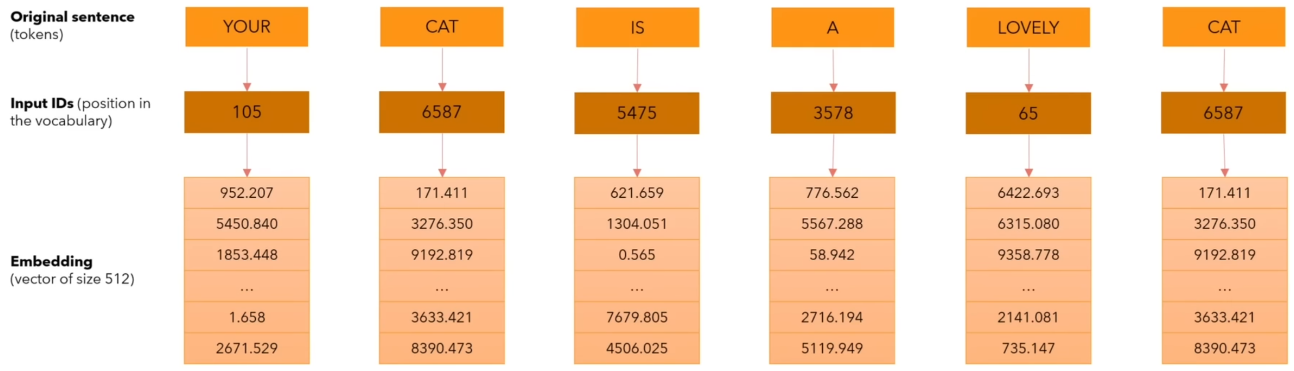


Let us first examine the encoder part, which is more important for us (from the point of view of application in vision). The most interesting multi head attention layer will be described later. Notes regarding the remaining structure: Please note the ResNET like shortcuts. *Feed forward* contains usual fully connected layers. *Add* is used due to the

ResNET approach. *Norm* is a batch normalisation that will be explained/illustrated later. The N_x value says that this structure is repeated N_x times. (In the paper, the authors used $N_x = 6$).

An example of how the encoder input is created. Each word is firstly replaced by a number (input ID) from a vocabulary, this representation is then mapped into a vector (512 real numbers in this case and in the original paper), which is called the embedding. The numbers changes during training, which reflects a specific meaning of the word.

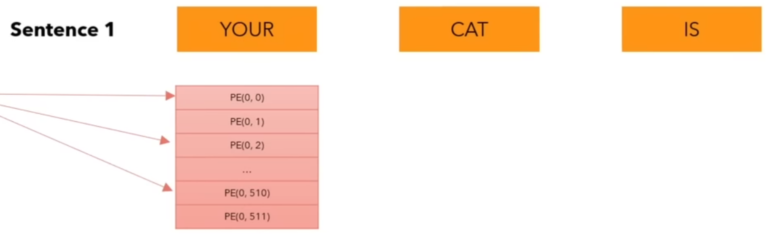
Note: Svereal following pictures presented here are taken from nice Umar Jamil’s tutorial that is available here https://www.youtube.com/watch?v=bCz4OMemCcA .



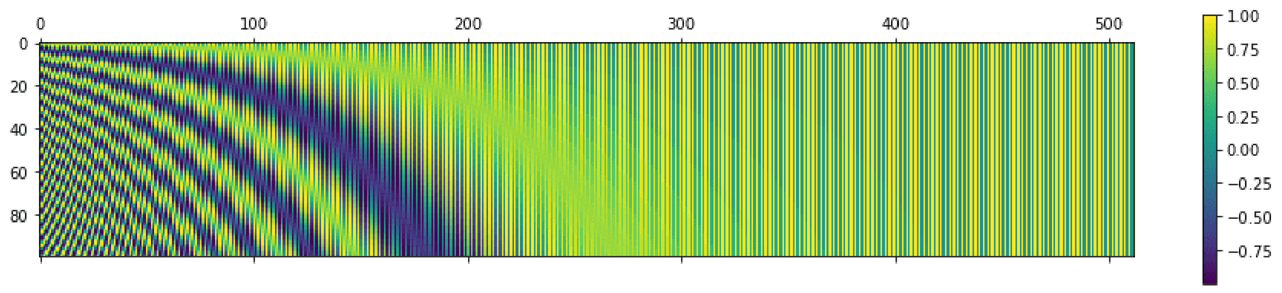
Original sentence (tokens)	YOUR	CAT	IS	A	LOVELY	CAT
Input IDs (position in the vocabulary)	105	6587	5475	3578	65	6587
Embedding (vector of size 512)	952.207	171.411	621.659	776.562	6422.693	171.411
	5450.840	3276.350	1304.051	5567.288	6315.080	3276.350
	1853.448	9192.819	0.565	58.942	9358.778	9192.819
	…	…	…	…	…	…
	1.658	3633.421	7679.805	2716.194	2141.081	3633.421
	2671.529	8390.473	4506.025	5119.949	735.147	8390.473

Positional encoding (which is the next step, see the figure above) reflects the position of the word in sentence. The authors proposed the following (see the figure). These values are simply added to the values above (in embedding), which gives the input to the encoder.

$$PE(pos, 2i) = \sin \frac{pos}{10000^{\frac{2i}{d_{model}}}}$$

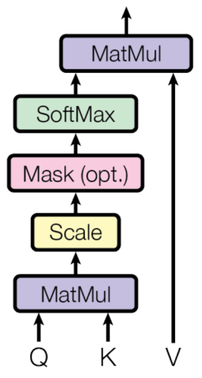


$$PE(pos, 2i+1) = \cos \frac{pos}{10000^{\frac{2i}{d_{model}}}}$$

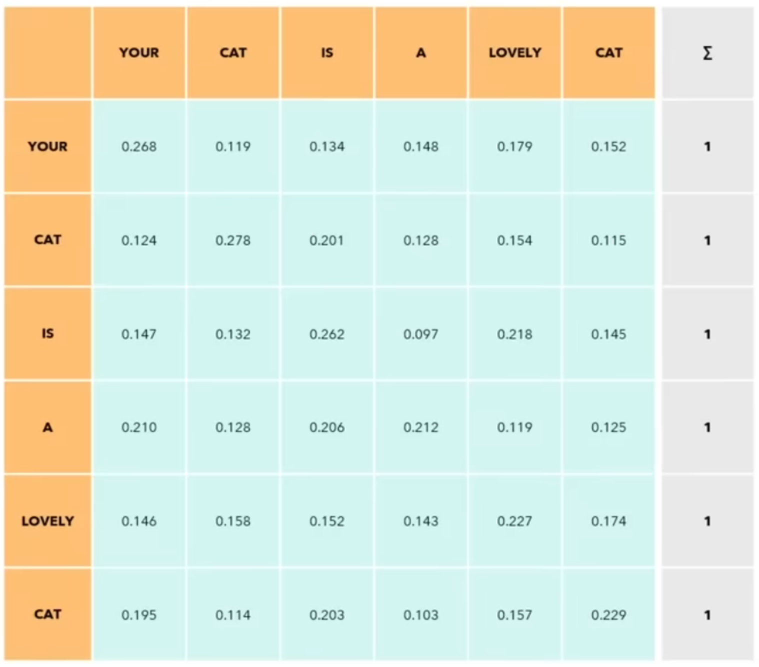


What is happening in the encoder, namely in its most interesting part? We start from the simple self attention, which should illustrate the nice idea that is hidden in attention. Consider for a while that *Q*, *K*, *V* (*query*, *key*, *value*) are three projections of the encoder input (i.e. obtained from the input by multiplying three matrices whose values are determined during training, simple copy is possible too). The projections may either retain the original size of embedding or may decrease the size. In this explanation, we understand *Q*, *K*, *V* as matrices whose size is 6×512 (6 is because we restrict ourselves here, for the purpose of explanation, to the sentences containing just 6 words, 512=d_k is the size of embedding of each word in the sentence, this size has been retained during projection). The attention is then defined as follows (you can also see it as a picture below):

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

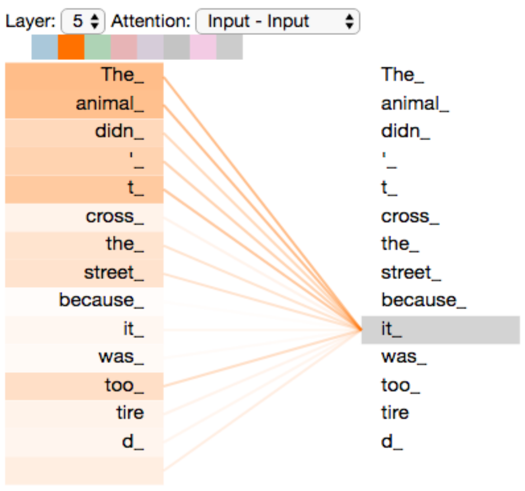


An important step is to understand what it exactly means. The result of softmax is illustrated in the figure below. The high values in the resulting matrix tells that the words of the corresponding row/column are tied together, i.e. are important one for another in the sentence, i.e. determine the context. Due to the softmax function, the values may be regarded as probabilities. From the way how the matrix was computed (dot product), we could also say that the values in the matrix express the “similarity” (or a “connection”) between the words.



	YOUR	CAT	IS	A	LOVELY	CAT	Σ
YOUR	0.268	0.119	0.134	0.148	0.179	0.152	1
CAT	0.124	0.278	0.201	0.128	0.154	0.115	1
IS	0.147	0.132	0.262	0.097	0.218	0.145	1
A	0.210	0.128	0.206	0.212	0.119	0.125	1
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174	1
CAT	0.195	0.114	0.203	0.103	0.157	0.229	1

Another illustration: Say the following sentence is an input sentence we want to translate: ”*The animal didn't cross the street because it was too tired.*” (Who was tired? The street or animal?) After training, the encoder reveals the following dependencies.



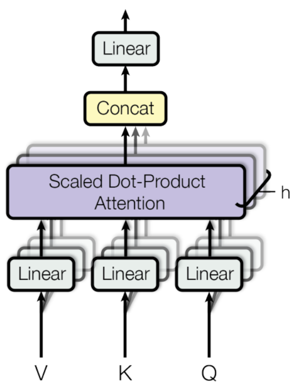
Layer: 5 Attention: Input - Input

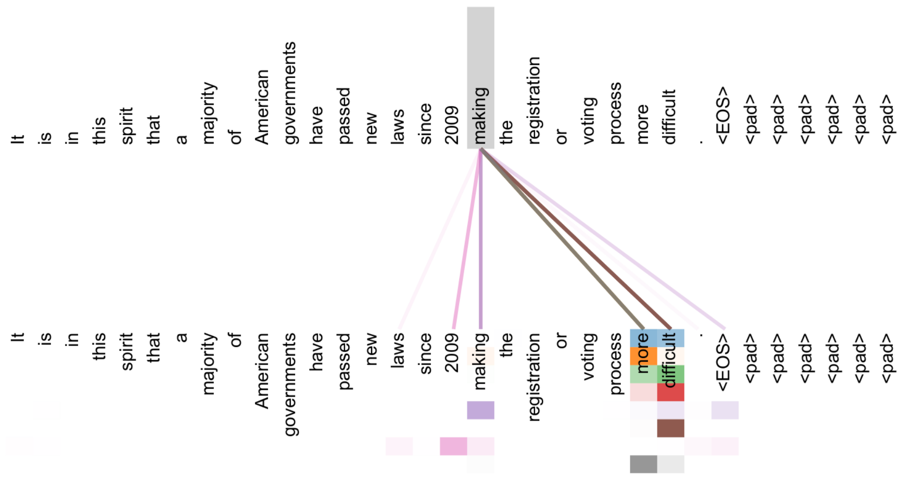
Now, please, try to imagine what is happening if the whole formula is computed, i.e. matrix multiplication by V is carried out. A new features for the particular word are computed which is done as a linear combination defined by similarities from the above matrix. It can be easily understood now that the word at the beginning of the sentence may have a big influence on the word at the end of the sequence. (Compare this with the recurrent networks, including LSTMs).

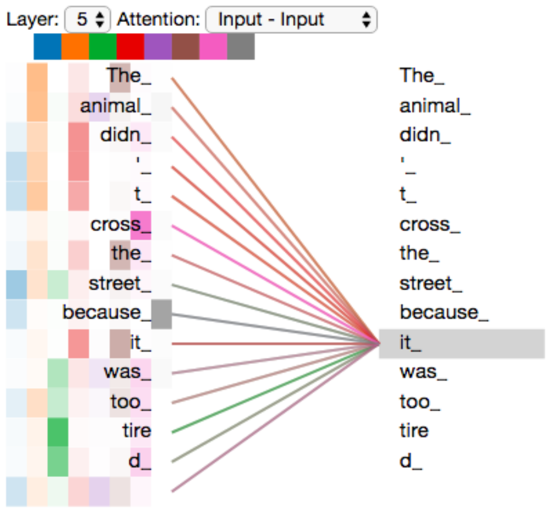
More exactly, the multihead attention is usually used. It is realised in such a way that the projection matrices W_i^Q, W_i^K, W_i^V are introduced whose values are determined by learning, i stands for i-th head.

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

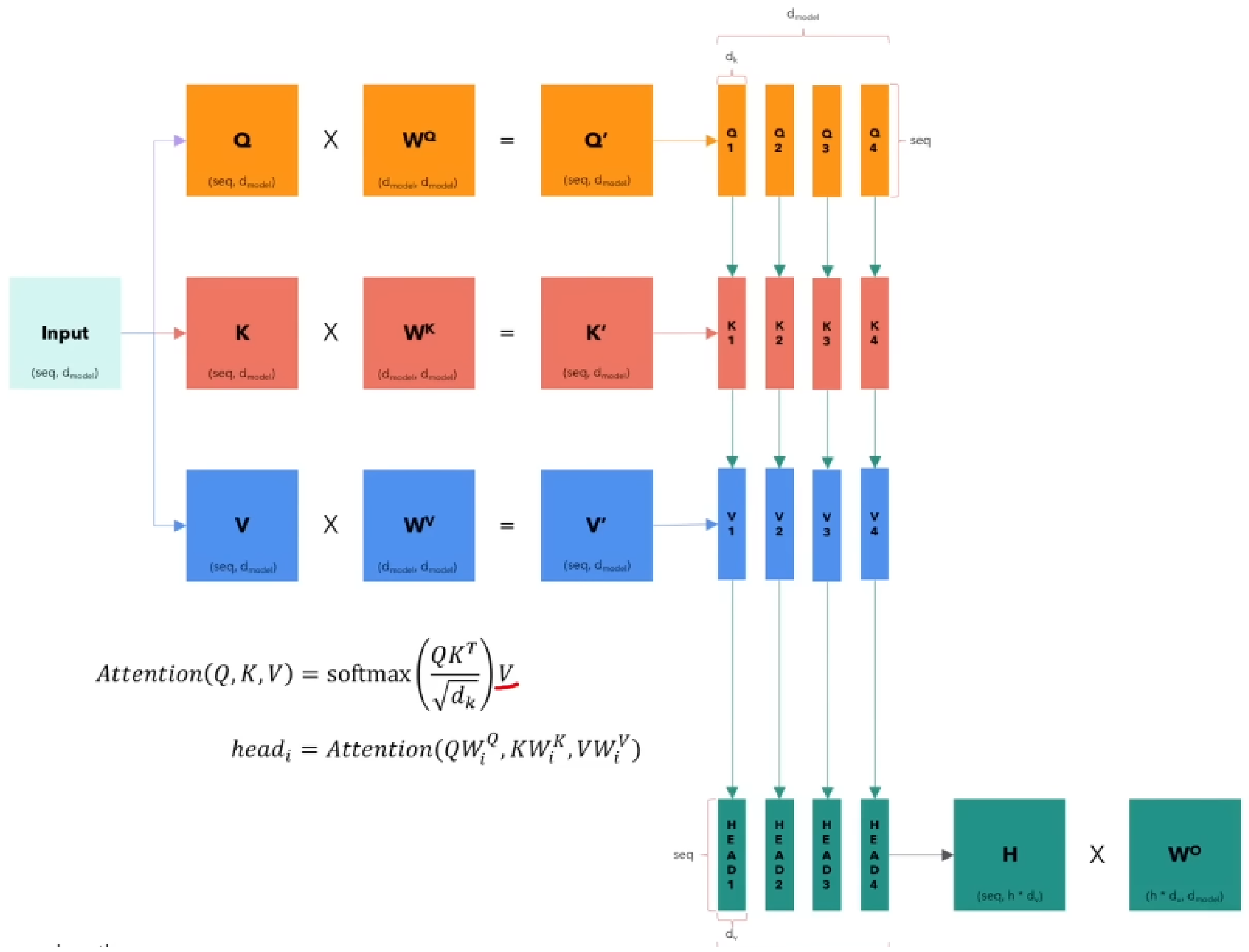
Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.



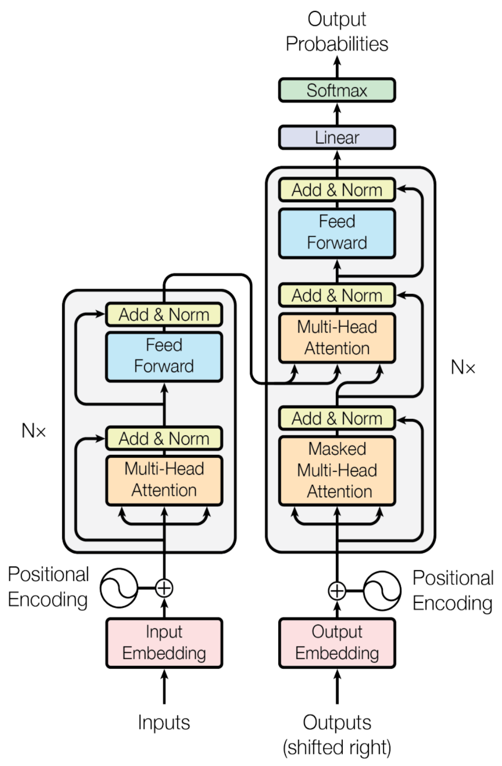

It is in this spirit that a majority of American governments have passed new laws since 2009 making the registration or voting process more difficult . <EOS> <pad> <pad> <pad> <pad> <pad> <pad>
It is in this spirit that a majority of American governments have passed new laws since 2009 making the registration or voting process more difficult . <EOS> <pad> <pad> <pad> <pad> <pad> <pad>

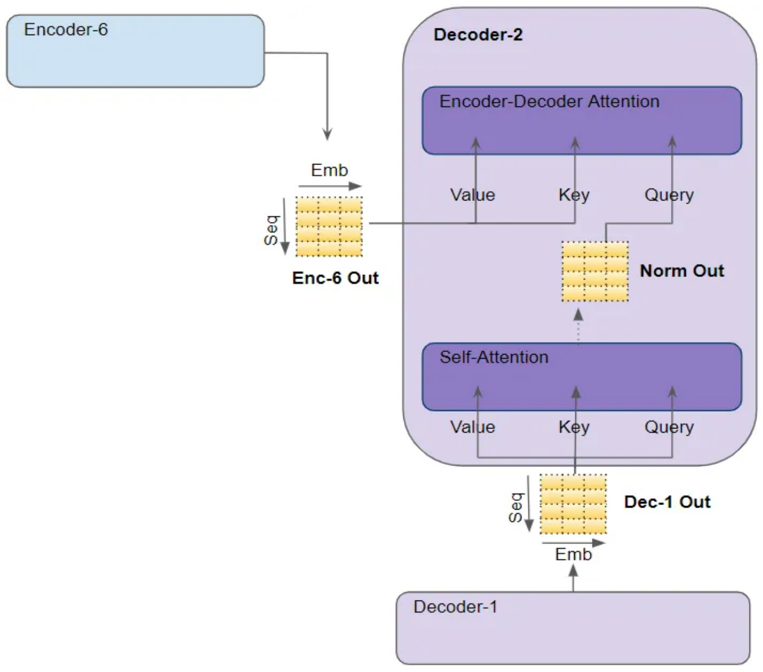

Layer: 5 Attention: Input - Input

If we imagine that the weight matrices for particular heads are put to one big matrix, the multihead processing may be illustrated by the following figure.

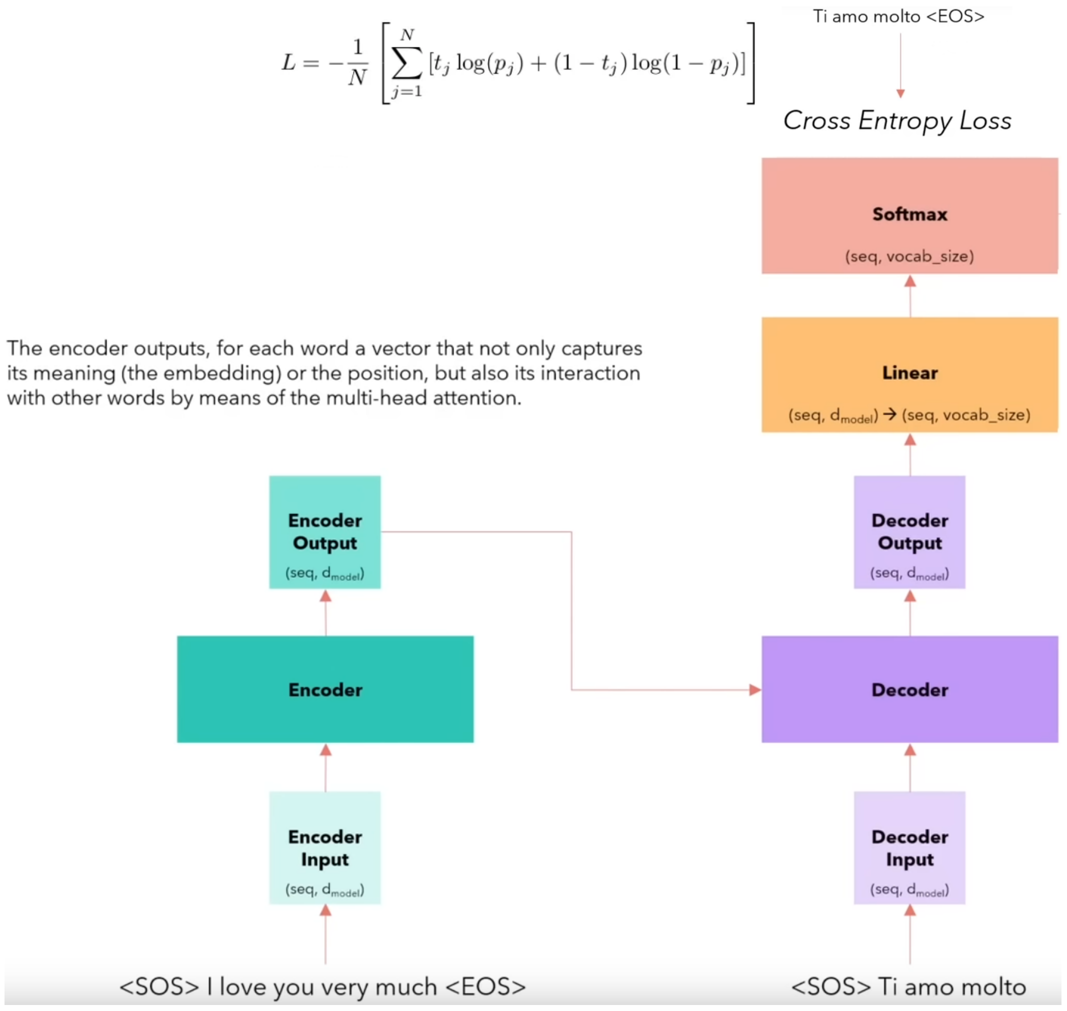


In the original paper, the authors had h=8, d_k=d_v=512/8=64.





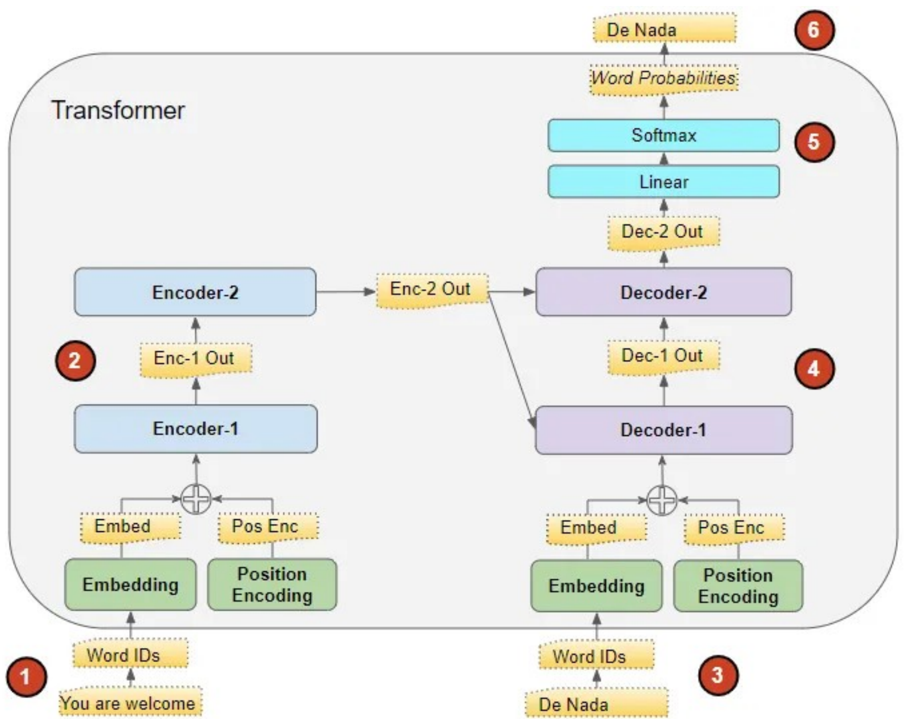
Training (in the context of translation if you are interested)

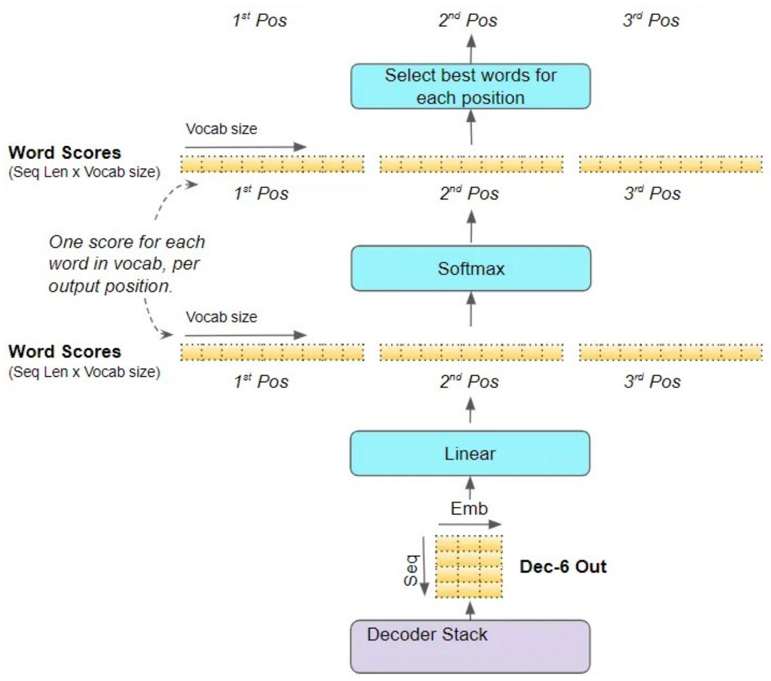


$$L = -\frac{1}{N}\left[\sum_{j=1}^{N}[t_j \log(p_j) + (1-t_j)\log(1-p_j)]\right]$$

The encoder outputs, for each word a vector that not only captures its meaning (the embedding) or the position, but also its interaction with other words by means of the multi-head attention.

Or with nice and more explicit pictures (taken from https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452)

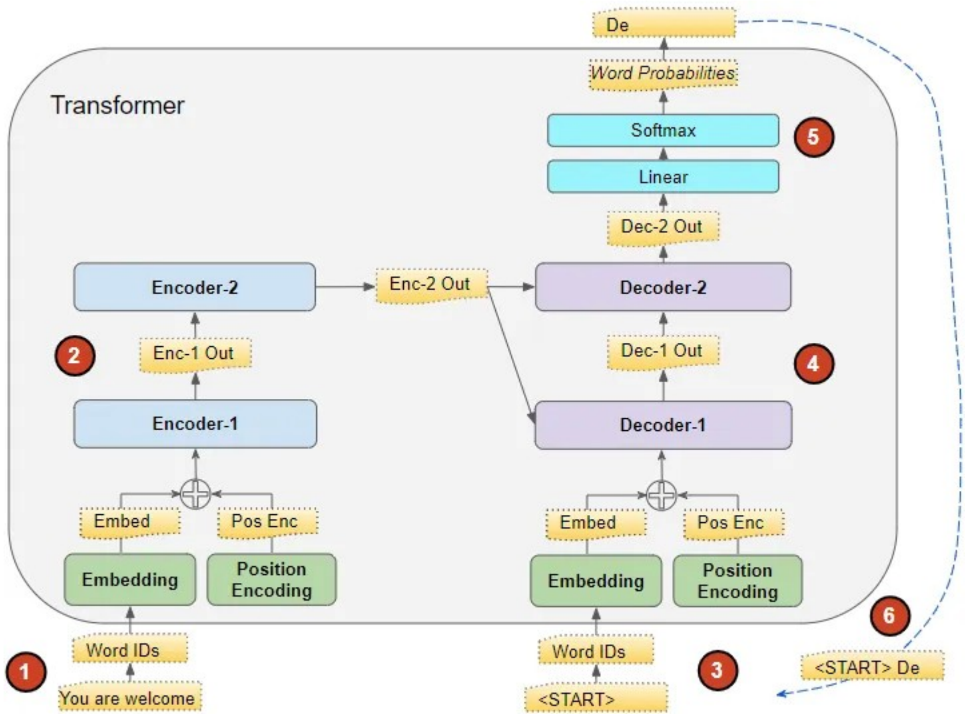




The Transformer processes the data like this:

1. The input sequence is converted into Embeddings (with Position Encoding) and fed to the Encoder.
2. The stack of Encoders processes this and produces an encoded representation of the input sequence.
3. The target sequence is prepended with a start-of-sentence token, converted into Embeddings (with Position Encoding), and fed to the Decoder.
4. The stack of Decoders processes this along with the Encoder stack's encoded representation to produce an encoded representation of the target sequence.
5. The Output layer converts it into word probabilities and the final output sequence.
6. The Transformer's Loss function compares this output sequence with the target sequence from the training data. This loss is used to generate gradients to train the Transformer during back-propagation.

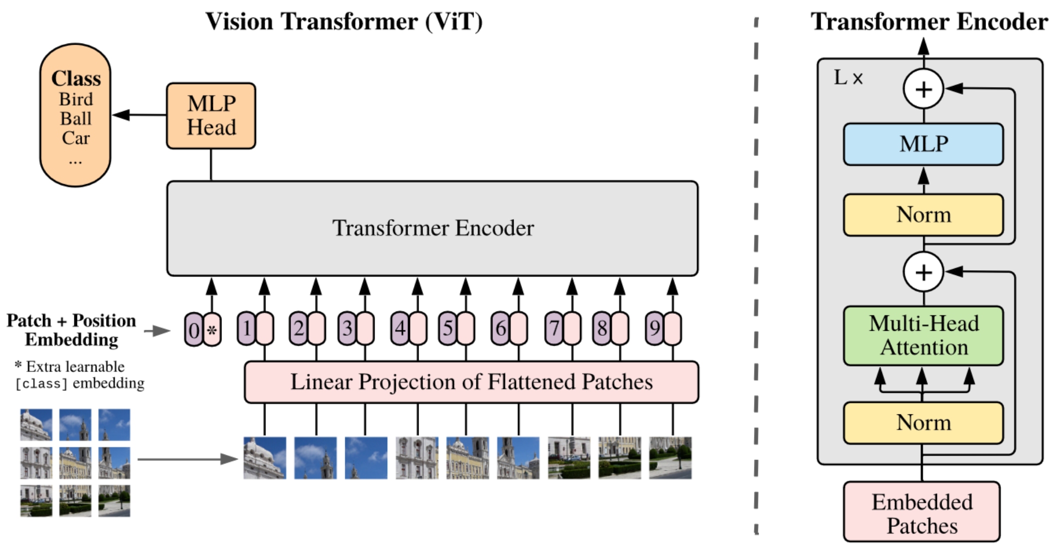
Inference (again from https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452)



The flow of data during Inference is:

1. The input sequence is converted into Embeddings (with Position Encoding) and fed to the Encoder.
2. The stack of Encoders processes this and produces an encoded representation of the input sequence.
3. Instead of the target sequence, we use an empty sequence with only a start-of-sentence token. This is converted into Embeddings (with Position Encoding) and fed to the Decoder.
4. The stack of Decoders processes this along with the Encoder stack's encoded representation to produce an encoded representation of the target sequence.
5. The Output layer converts it into word probabilities and produces an output sequence.
6. We take the last word of the output sequence as the predicted word. That word is now filled into the second position of our Decoder input sequence, which now contains a start-of-sentence token and the first word.
7. Go back to step #3. As before, feed the new Decoder sequence into the model. Then take the second word of the output and append it to the Decoder sequence. Repeat this until it predicts an end-of-sentence token. Note that since the Encoder sequence does not change for each iteration, we do not have to repeat steps #1 and #2 each time.

Now we can continue with the Vision Transformer (ViT) as it was introduced in: Alexey Dosovitskiy et al.: An Image Is Worth 16x16 Words: Transformers For Image Recognition at Scale, https://arxiv.org/abs/2010.11929. The authors uses only the encoder (left part) of the general transformer above. The figure below is taken from their paper.



The authors transformed the image recognition problem in such a way that they almost entirely copy the way of how the transformer is used in NLP. Image is divided into subimages. The subimages directly (or some projections) are used as embedding, which is done with the hope that in the process of encoding the more and more descriptive features will be computed in their place. However, the final values of particular tokens (features) are not interesting in this case (the classification should be done). Also, positional encoding is used as can be seen from the picture above. It can be simple in this case since the subimages extracted from the image have fixed positions.

For the purpose od classification, a special class token is used that is randomly initialized and prepended to the beginning of the input sequence. Since it is randomly initialized, it does not contain any useful information on its own. Token is able to accumulate information from the other tokens in the sequence the deeper and more layers have been processed. When the tansformer finally performs the final classification of the sequence, it uses an MLP head which only looks at data from the last layer class token and no other information. It can be perceived as a placeholder data structure that is used to store information that is extracted from other tokens in the sequence.

Unfortunately, this ViT is a sliding window approach. (However, the approaches using transformers aiming at detecting objects have appeared too.) The performance of this particular ViT was a bit questionable (i.e. below some CNNs, like ResNET, see the following picture). This was discussed in the following paper (I will comment on the details during the lecture).

Tokens-to-Token ViT: Training Vision Transformers from Scratch on ImageNet

Li Yuan[1]*, Yunpeng Chen[2], Tao Wang[1,3]*, Weihao Yu[1], Yujun Shi[1],
Zihang Jiang[1], Francis E.H. Tay[1], Jiashi Feng[1], Shuicheng Yan[1]

[1] National University of Singapore [2] YITU Technology [3] Institute of Data Science, National University of Singapore

`yuanli@u.nus.edu, yunpeng.chen@yitu-inc.com, shuicheng.yan@gmail.com`

Abstract

Transformers, which are popular for language modeling, have been explored for solving vision tasks recently, e.g., the Vision Transformer (ViT) for image classification. The ViT model splits each image into a sequence of tokens with fixed length and then applies multiple Transformer layers to model their global relation for classification. However, ViT achieves inferior performance to CNNs when trained from scratch on a midsize dataset like ImageNet. We find it is because: 1) the simple tokenization of input images fails to model the important local structure such as edges and lines among neighboring pixels, leading to low train-

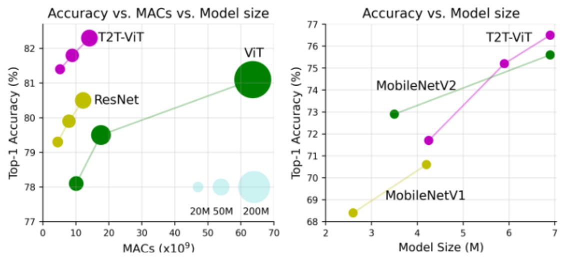
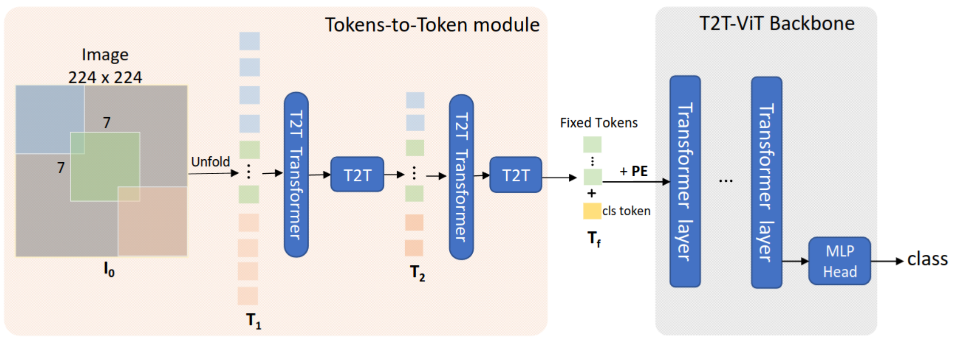
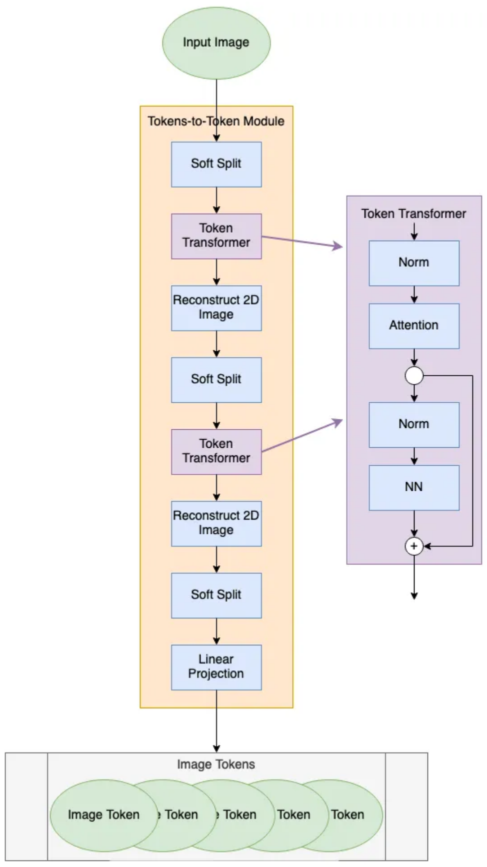
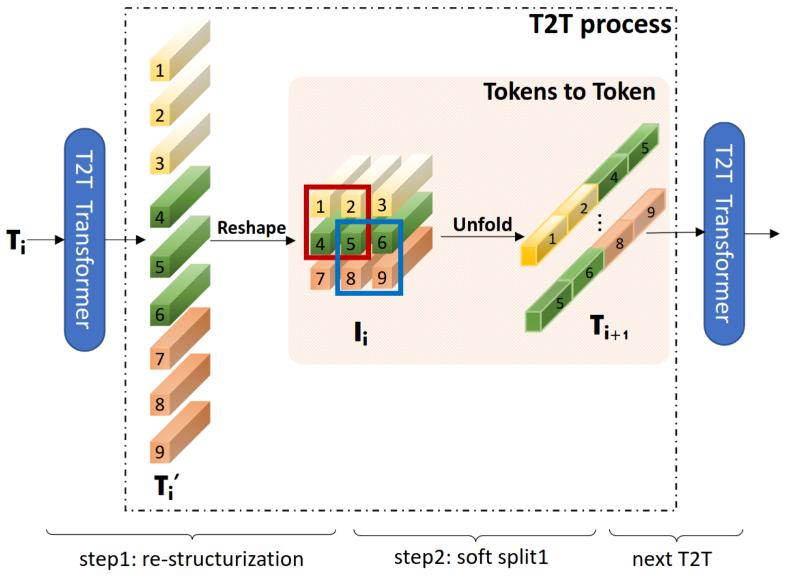


Figure 1. Comparison between T2T-ViT with ViT, ResNets and MobileNets when trained from scratch on ImageNet. Left: performance curve of MACs vs. top-1 accuracy. Right: performance curve of model size vs. top-1 accuracy.







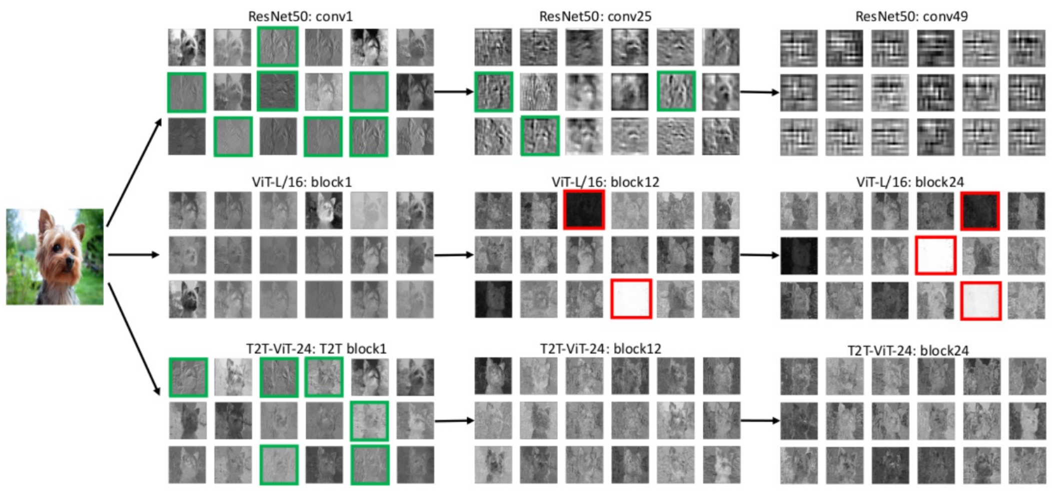
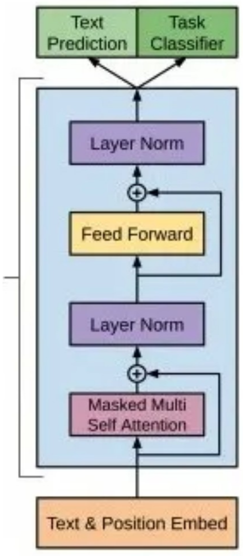
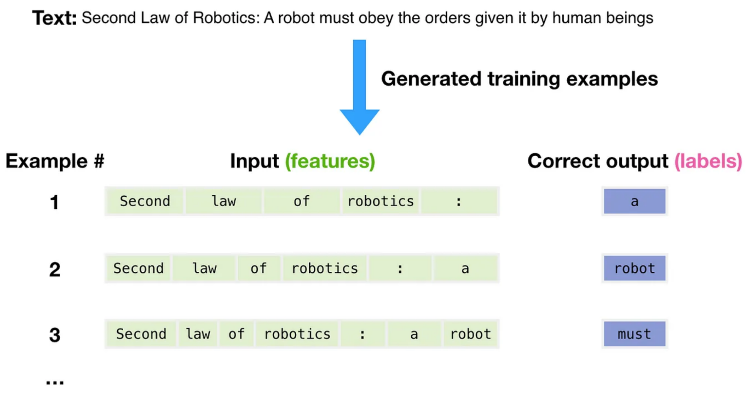


Figure 2. Feature visualization of ResNet50, ViT-L/16 [12] and our proposed T2T-ViT-24 trained on ImageNet. Green boxes highlight learned low-level structure features such as edges and lines; red boxes highlight invalid feature maps with zero or too large values. Note the feature maps visualized here for ViT and T2T-ViT are not attention maps, but image features reshaped from tokens. For better visualization, we scale the input image to size 1024×1024 or 2048×2048.

A note on GPT:

GPT uses an unmodified Transformer decoder, except that it lacks the encoder attention part. We can see this visually in the above diagrams. The GPT, GPT2, GPT 3 is built using transformer decoder blocks. GPT-3 was trained with huge Internet text datasets — 570GB in total. When it was released, it was the largest neural network with 175 billion parameters (100x GPT-2). GPT-3 has 96 attention blocks that each contain 96 attention heads




Text: Second Law of Robotics: A robot must obey the orders given it by human beings

Generated training examples

Example #	Input (features)	Correct output (labels)
1	Second law of robotics :	a
2	Second law of robotics : a	robot
3	Second law of robotics : a robot	must
...		