Convolutional Neural Networks

Mihai Berbec

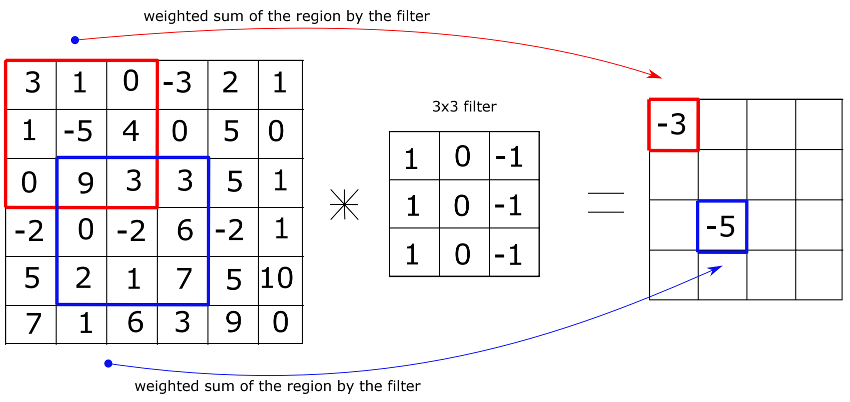
April 27, 2020

Chapter 1

Convolutional Neural Networks

1.1 Convolutions

Convolution



Dimensions: $(6, 6) * (3, 3) = (6 - 3 + 1, 6 - 3 + 1) = (4, 4)$, thus the image is shrinking with every convolution. Also, the information contained near the edges is not used.

If an (n, n) image is convoluted with an (f, f) filter, then the resulting image transforms to $(n - f + 1, n - f + 1)$.

Padding: add a border of p pixels around the edges (usually zeros). If the dimension of the original image is (n, n), then the dimension of the padded image becomes $(n + 2p, n + 2p)$.

To preserve the input dimension we have to take $p = (f - 1)/2$, where (f, f) is the dimension of the filter (f is usually odd).

Stride: how many pixels to move between two consecutive applications of the filter (in the figure above we have stride $= 1$).

In general, the result of a convolution $(n, n) * (f, f)$, with stride s and padding p is

$$\left(\frac{n + 2p - f}{s} + 1, \frac{n + 2p - f}{s} + 1 \right).$$

If the image has multiple channels, then the filter must have the same number of channels: $(n, n, 3) * (f, f, 3) = (n - f + 1, n - f + 1)$. Both the image region and the filter are cubes now, so the weighted sum is still possible.

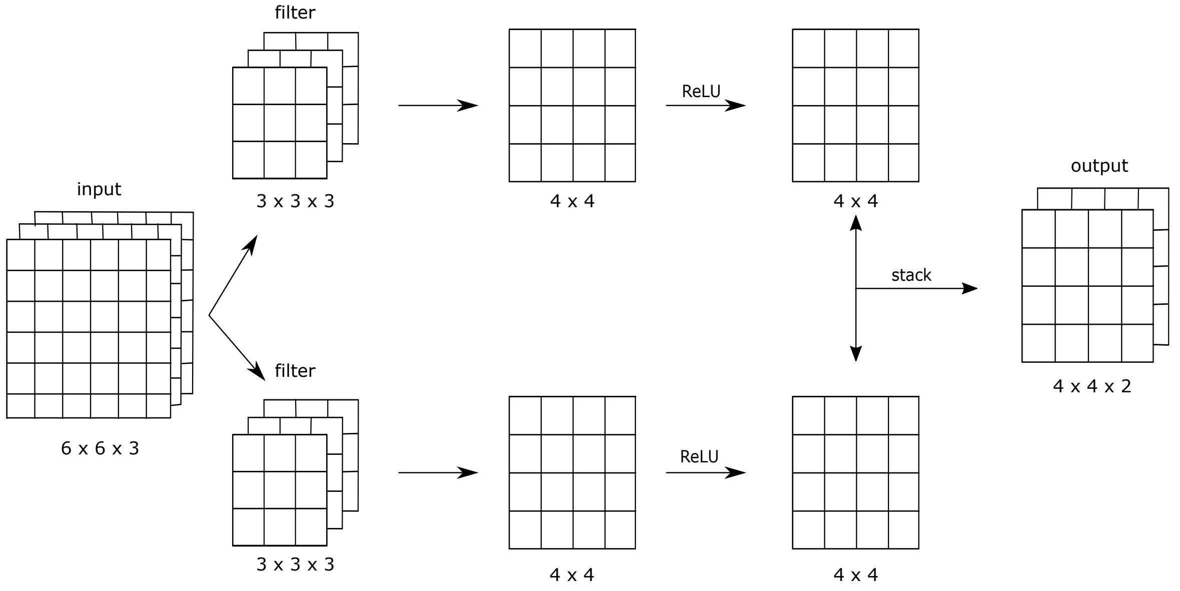
Most of the time we apply many filters with the same size on an image and stack the resulting images as channels of new image:

$$\left.\begin{array}{rcl}(n,n,3)*(f,f,3) & = & (n-f+1,n-f+1)\\(n,n,3)*(f,f,3) & = & (n-f+1,n-f+1)\end{array}\right\}\xrightarrow{\text{stack}}(n-f+1,n-f+1,2).$$

"Same" convolution: dimension of the input equals dimension of the output.

1.2 Convolutional layers

Example of convolutional layer:



The number of parameters of the layer is determined by the number and size of the filters, not by the size of the input image.

We fix the following notation:

- l = convolutional layer
- f^l = filter size
- p^l = padding size
- s^l = stride
- $n_H^{l-1} \times n_W^{l-1} \times n_C^{l-1}$ = height $\times$ width $\times$ channels of the input coming from layer $l-1$
- $n_H^l \times n_W^l \times n_C^l$ = height $\times$ width $\times$ channels of the output
- n_C^l = number of filters in the layer l

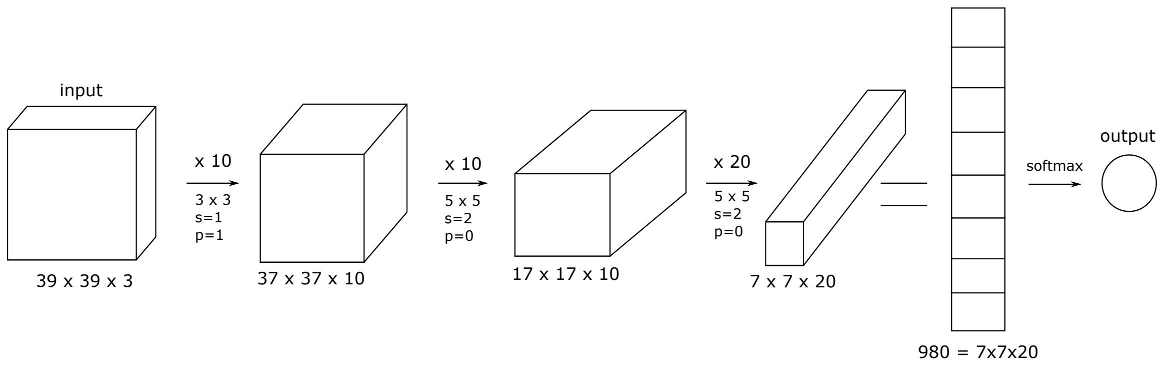
Then:

$$n^l_{H/W} = \left\lfloor \frac{n^{l-1}_{H/W} + 2p^l - f^l}{s^l} + 1 \right\rfloor,$$

and we have the following dimensions:

- filters $= f^l \times f^l \times n^{l-1}_C$
- activations $a^l = n^l_H \times n^l_W \times n^l_C$
- weights $= f^l \times f^l \times n^{l-1}_C \times n^l_C$
- biases $= 1 \times 1 \times 1 \times n^l_C$

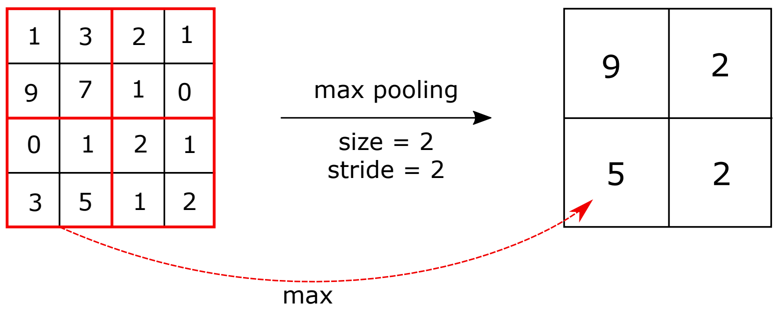
Example of convolutional neural network (CNN):



Why are convolutional networks good in practice ?

- the number of learnable parameters is relatively small as many parameters are shared by the input nodes
- sparsity of connections: outputs depend only on a small number of inputs (less prone to overfitting, can train on smaller datasets)

1.3 Pooling layers



Max pooling works very well in practice, but we can use min, average pooling, etc.

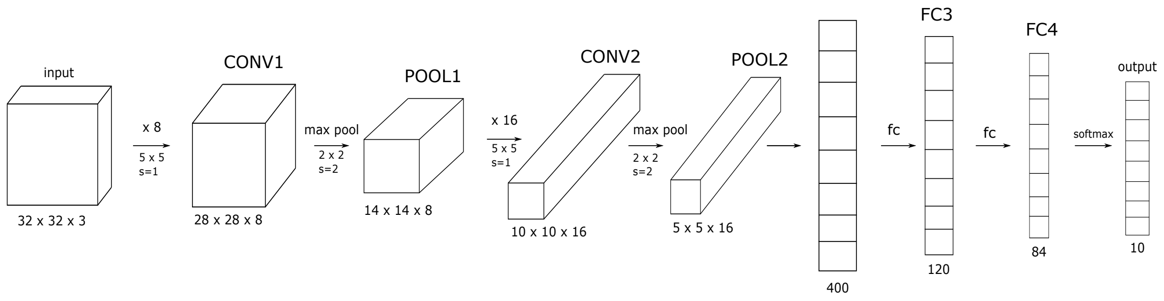
If the input image has multiple channels, pooling is done separately on each channel (thus it preserves the number of channels).

Pooling layers have no learnable parameters, they are used to reduce the image size.

1.4 Classical convolutional networks

LeNet-5

Used to classify hand-written digits from the MNIST dataset (approx. 60k parameters).

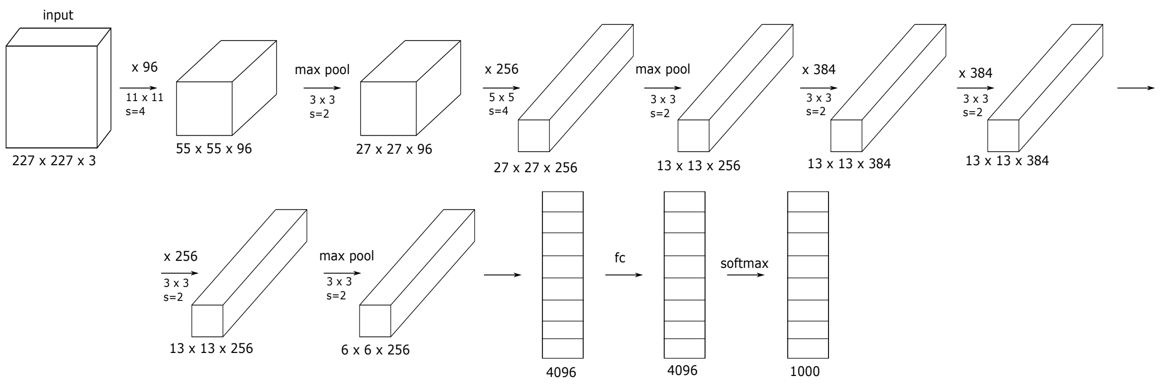


Layer	Shape	Size	Parameters
Input	(32,32,3)	3072	0
Conv1	(28,28,8)	6272	208
Pool1	(14,14,8)	1586	0
Conv2	(10,10,16)	1600	416
Pool2	(5, 5, 16)	400	0
FC3	(120, 1)	120	48001
FC4	(84, 1)	84	10081
Softmax	(10, 1)	10	841

- pooling layers have no learnable parameters
- convolutional layers have few parameters comparing to the fully connected layers
- activations size goes down gradually trough the network
- image size decreases, but the number of channels increases
- in the original paper, sigmoid/tanh are used as activation functions (modern implementations use ReLU)

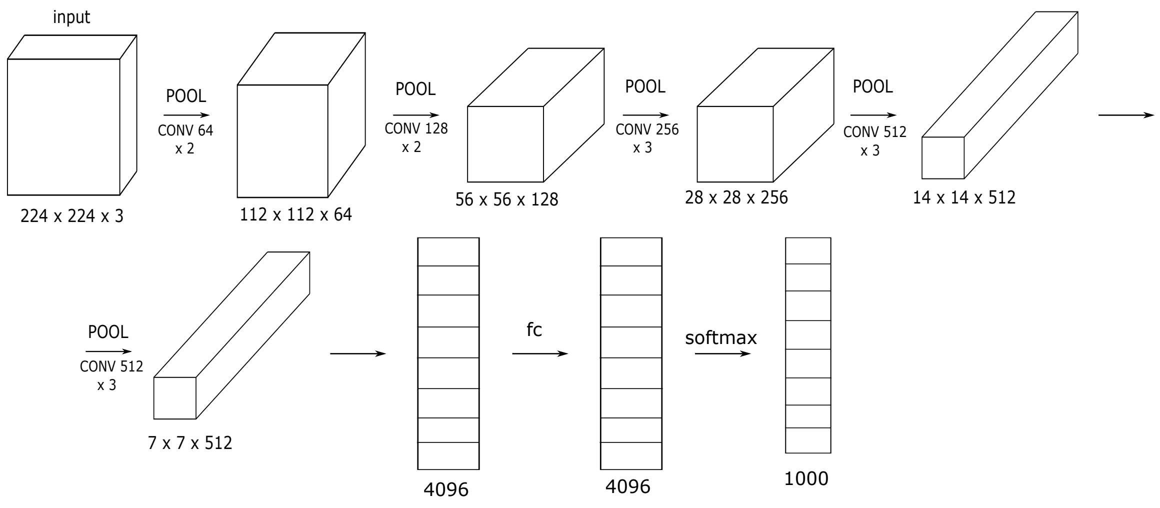
AlexNet

Has approx. 60 millions parameters and uses ReLU as activation function. First successful convolutional network for images, trained on the ImageNet dataset.



VGG-16

Has approx. 138 millions parameters. All convolutions are 3×3 with stride 1 and max pooling is 2×2 with stride 2.



ResNet - Residual Networks

Very deep networks have an issue: vanishing/exploding gradients.

Residual block

$$\begin{tikzcd}
{} \arrow[r, "\text{main path}"] & a^{[l]} \arrow[r] \arrow[rrrrr, bend right, "\text{skip connection}"'] & \text{linear} \arrow[r] & g \arrow[r] & a^{[l+1]} \arrow[r] & \text{linear} \arrow[r] & \oplus \arrow[r] & g \arrow[r] & a^{[l+2]}
\end{tikzcd}$$

On layer $l + 2$, we have that:

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]}),$$

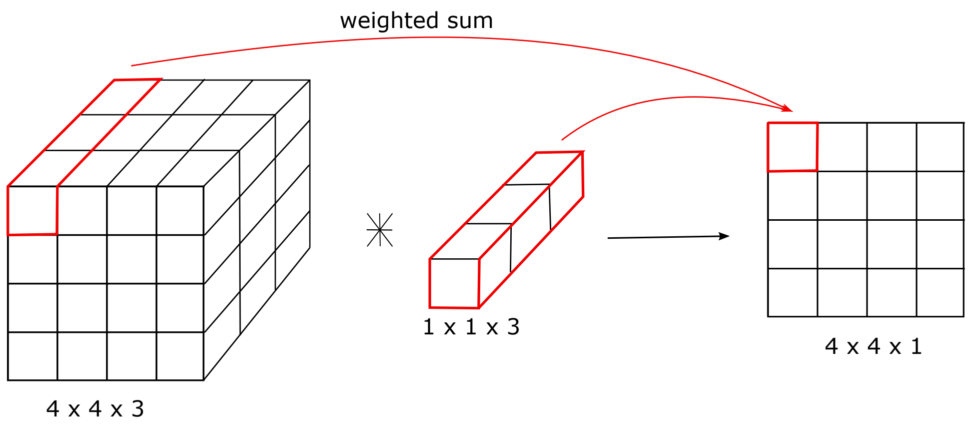
thus $z^{[l+2]}$ and $a^{[l]}$ must have the same dimension.

Residual networks are made of residual blocks. This particular architecture helps training very deep networks (e.g. ResNet: 34-layer residual network).

Inception networks

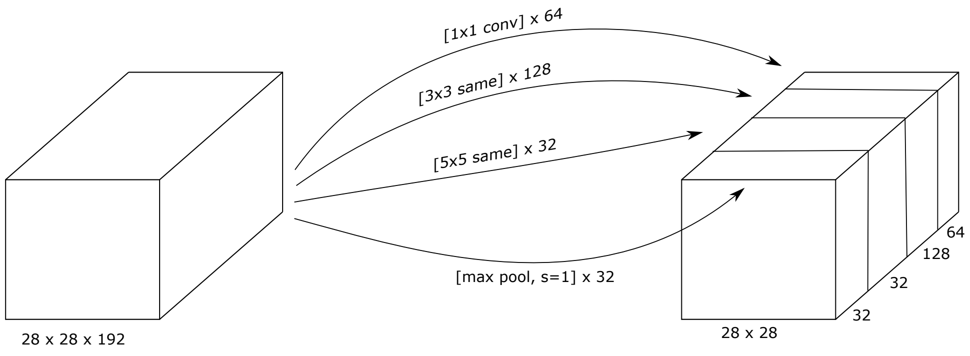
1×1 **convolutions** (networks in networks)

- reduce the number of channels
- add non-linearity if keeping the same number of channels

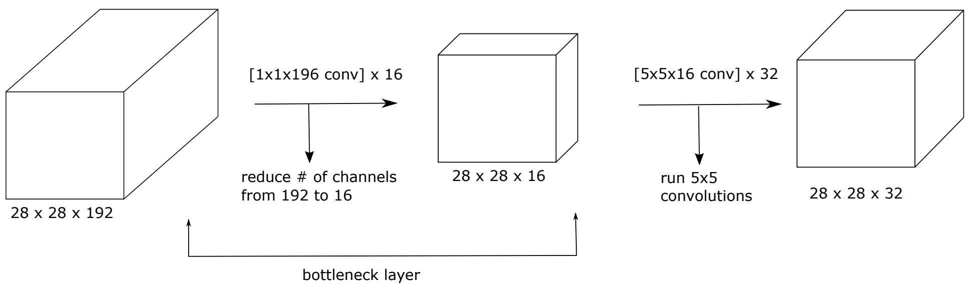


Inception networks

Assume we have to implement the following convolutional layer:

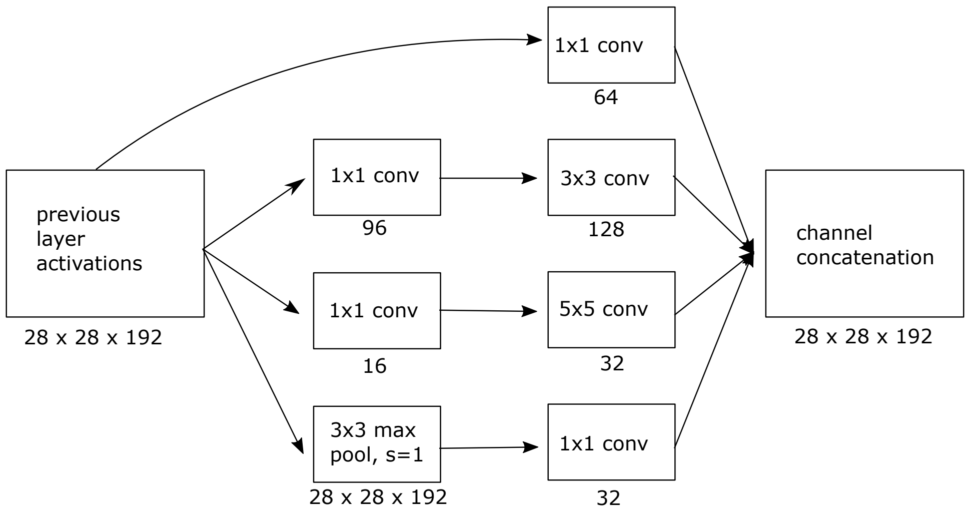


This layer is very expensive computationally: approx. 120 millions multiplications. Instead of 3×3 and 5×5 convolutions, we can use 1×1 convolutions to reduce the number of channels and then run the corresponding convolutions.



Compared to the layer above, this implementation has around 12.4 millions multiplications, almost 10 times less.

Using this kind of convolutions, we can implement an **inception module** as follows:



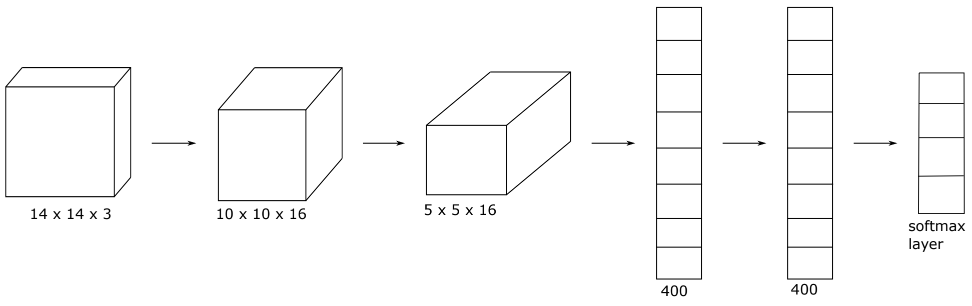
Inception network (**googLeNet**) - put together many inception modules (9 modules up to pooling), plus some additional side branches with softmax output to ensure that intermediate features are not too bad for prediction (this has a regularizing effect and prevents overfitting).

Data augmentation

- horizontal flip
- random cropping
- rotation, shearing, wrapping (not too often)
- color shifting (RGB)

1.5 Object Detection

Localization - classify the image and localize the main object (as a bounding box). We can use a simple classifier with a softmax layer with size equal to the number of classes.



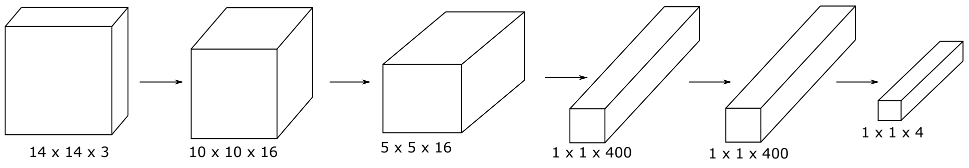
Detection - find all objects in the image and draw bounding boxes around them (many objects, different categories).

Sliding window detection:

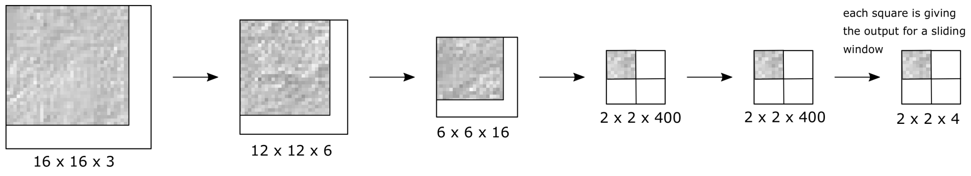
- slide different size windows though an image and use a classifier to find objects (this method has a huge computational cost)

- can be implemented using convolutions

To do this, we turn the fully connected layers of the previous classifiers into convolutional layers.



For larger images, we apply the same network on each sliding window and put together the outputs.

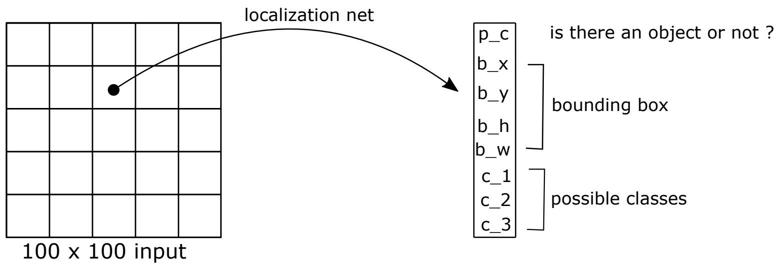


This still has an issue: the position of bounding boxes is not very accurate.

YOLO object detection algorithm

YOLO (You Only Look Once) predicts the bounding boxes of objects in the image.

Assume we start with an 100×100 image and we have 3 different object classes we want to detect. We add a finer grid on the image and apply an image localization network (as the previous classifier) on each grid cell.

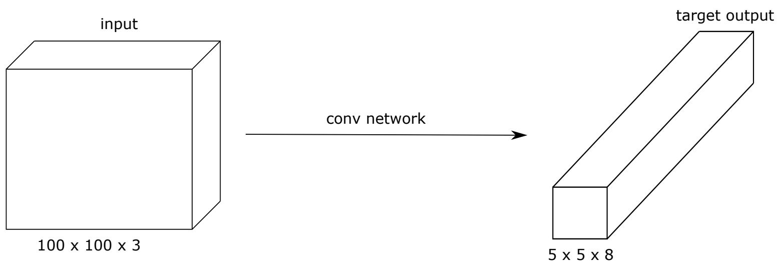


Thus for each cell of the input image we get an 8-dimensional vector $y = [p_c, b_x, b_y, b_h, b_w, c_1, c_2, c_3]$, where:

- p_c is 0 or 1, saying if there is an object or not in the cell,
- b_x, b_y, b_h, b_w is the bounding box of the object,
- c_1, c_2, c_3 is a vector indicating the class of the object (an entry is 1 and the rest are 0).

We use these 8-dimensional vectors as labels for training a convolutional network. If an object is spread over multiple cells, we assign the object to the cell that contains its center.

As in the previous figure, if we put a 5×5 grid on the input images, we get 25 output vectors arranged in a $5 \times 5 \times 8$ matrix:



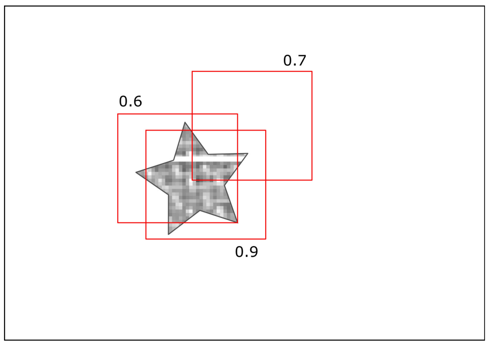
We use IOU (intersection over union) to measure detection accuracy.

- IOU measures the overlap of two bounding boxes,
- in our case, IOU equals the ratio between the area of the intersection and the area of the union of the ground truth and predicted bounding boxes,
- usually, if IOU ≥ 0.5, the prediction is assumed to be correct.

So far, this approach works better than the sliding window, but there are still issues: we can find multiple detections of the same object if the grid is fine.

Non max suppression

Assume we have a star-like object to detect in the input image:



As we see in the image, we have 3 bounding boxes containing the object ($p_c = 1$ and the same class). To get the correct bounding box we use the non max suppression principle.

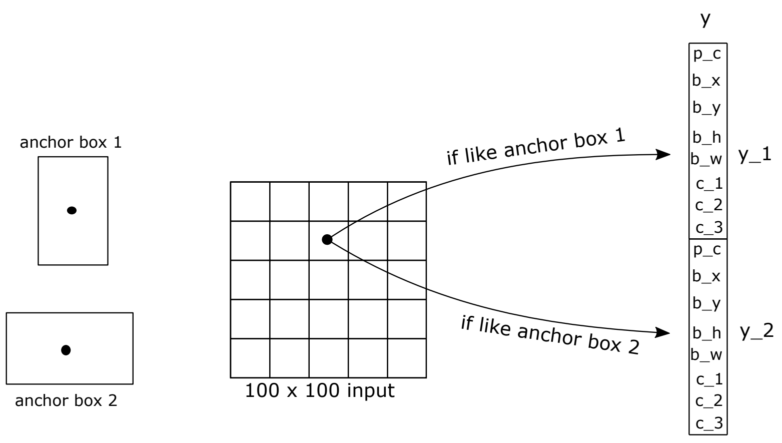
- take the most confident prediction (e.g. that one with IOU $= 0.9$) and mark it,
- look at the remaining bounding boxes and suppress those with high IOU,
- the final predictions will be the marked ones.

There is another issue here: each of the grid cells can detect only one object.

Anchor boxes

Pre-define different shapes called anchor boxes and allow the algorithm to specialize in detecting objects similar to these anchor boxes.

More specifically, for each grid cell we define a target output vector $y = [y_1, y_2, \dots, y_n]$, where y_i are similar to the output defined above, that can encode many objects in the same grid cell.

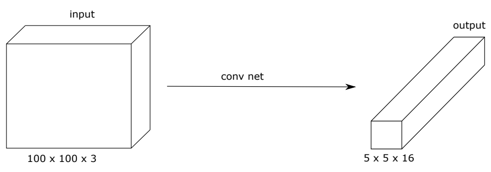


Now, each object in the training image is assigned to the cell that contains object's midpoint and anchor box with the highest IOU.

The YOLO algorithm

For simplicity, assume that we have 3 classes and 2 anchor boxes as before.

- start with an input image x and add a fine grid on it (e.g. 5×5 grid),
- for each grid cell, run an image localization algorithm to get the target output y (in our case, the size of y is $5 \times 5 \times 2 \times 8$ or $5 \times 5 \times 16$),
- train a convolutional network with input x and output y:



- run non-max suppression on all bounding boxes predicted for the input image and get the correct predictions.

Region Proposal (R-CNN)

R-CNN: if running a trained classifier across sliding windows, there are many windows containing no objects, so no need for detection. Instead, we can pick some regions where it makes sense to run the detector. To do this, we can use a segmentation algorithm based on pixel clustering (this can be very slow, as it may propose many regions and one region is classified at a time).

Fast R-CNN: use convolutional implementation of sliding window to classify all proposed regions at once. It is faster than R-CNN algorithm.

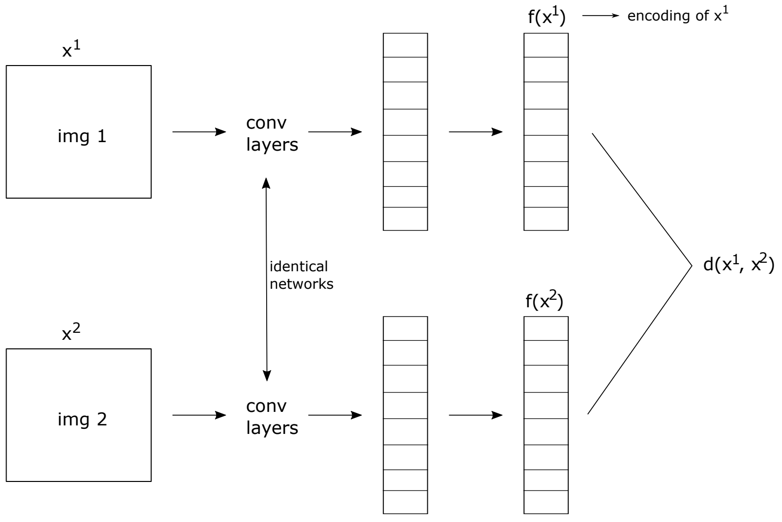
Faster R-CNN: use convolutional networks to propose the regions instead of the clustering segmentation algorithm (faster than R-CNN, but somehow slower than YOLO).

1.6 Face Recognition

One shot learning: there exists only one image for every person. We need to learn a similarity function which tells if two images are of the same person or not.

Siamese networks

Use two identical convolutional networks to extract feature encodings of the input images and put them together using a similarity function (which can be seen as a cost function).



The similarity distance d between the input images x^1 and x^2 is defined as the distance between the feature encodings $f(x^1)$ and $f(x^2)$:

$$d(x^1, x^2) = \left\| f(x^1) - f(x^2) \right\|_2^2 .$$

Learn network's parameters in such a way that:

- if x^i and x^j are the same person, then $\left\| f(x^i) - f(x^j) \right\|_2^2$ is small,
- if x^i and x^j are the different person, then $\left\| f(x^i) - f(x^j) \right\|_2^2$ is large.

Triplet loss

The triplet loss function requires a three images as input: anchor (A), positive (P), negative (N), where the anchor and the positive are similar and the anchor and the negative are different. The objective is to find parameters such that:

$$\left\| f(A) - f(P) \right\|_2^2 + \alpha \leq \left\| f(A) - f(N) \right\|_2^2 ,$$

where α is a positive number called margin.

For a triplet (A, P, N), the triplet loss function is defined as:

$$\mathcal{L}(A, P, N) = \max \left\{ \|f(A) - f(P)\|_2^2 - \|f(A) - f(N)\|_2^2 + \alpha, 0 \right\}.$$

If the triplets (A, P, N) are randomly chosen, then $d(A, P) + \alpha \leq d(A, N)$ is easily satisfied. Instead, choose triplets that are "hard" to train on, i.e. $d(A, P) \sim d(A, N)$ (see FaceNet, 2015).