Chapter 16-17

Segmentation Detection – U-Net

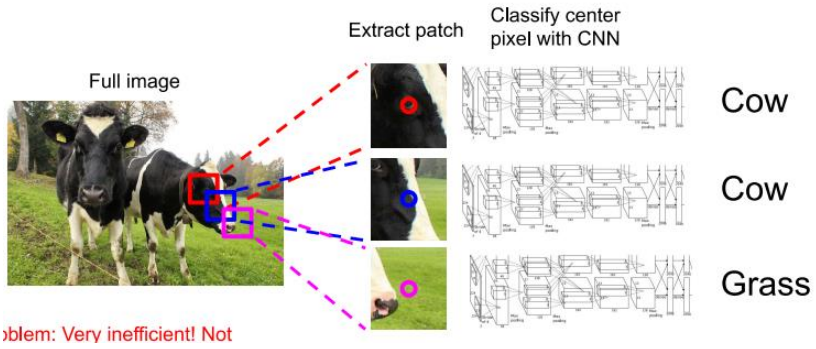
Author: Gianmarco Scarano

gianmarcoscarano@gmail.com

1. Segmentation Detection

As we saw in previous chapters, we have multiple Computer Vision tasks such as *Classification, Semantic Segmentation, Object Detection and Instance Segmentation*, but we'll focus right now on the Semantic Segmentation part.

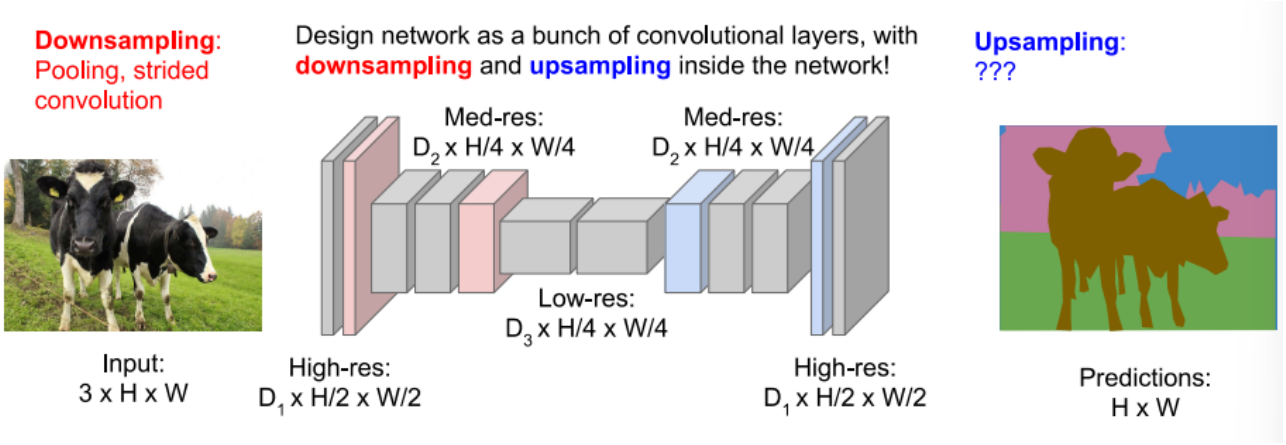
The main problem in this task is that, while in training each pixel is labeled, during test time we have to classify each pixel of a new image never-seen-before.



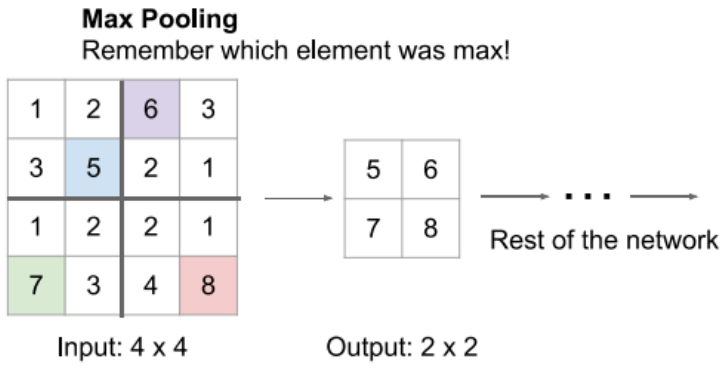
The algorithm behind this (well, a possible idea) relies on the use of a sliding window which slides all over the image and classifies that pixel as something (Cow, Grass, Sky, etc.). This, though, is very inefficient because we are not reusing shared features between overlapping patches of the images.

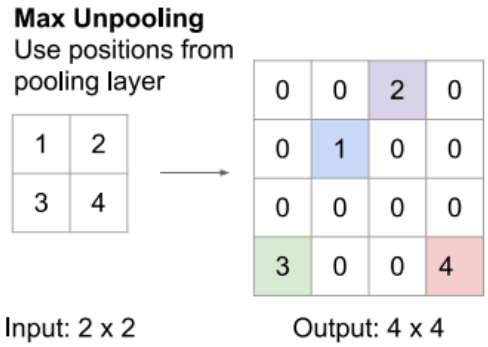
So, one of the proposed solution was to design a network with only Convolutional Layers without any downsampling operators (this is done because semantic segmentation requires the output shape to be equal as the input shape), since in CNNs we shrink the image in order to go deeper. This, anyways, would bring a lot of computational operations at the first Convolutions, since we have to convolve the image at high resolution.

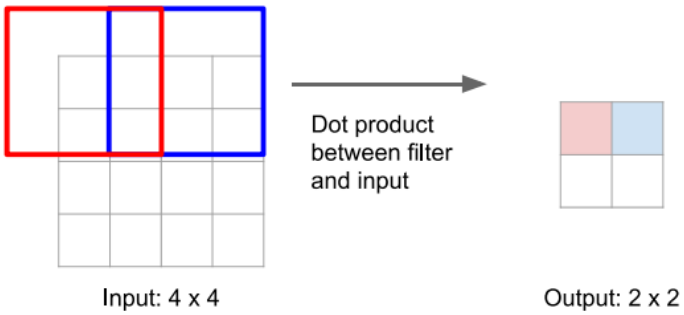
The final and working solution, then, is to design a network with bunch of convolutional layers, along with downsampling layers at the beginning and upsampling layers at the end, as we can see down below.



We know how downsample works (through Pooling or Convolutional Layers with Strides, etc), but how about the upsampling? The technique is relatively simple, as we apply a Max Unpooling, using positions from the Pooling Layer applied at the downsampling phase.

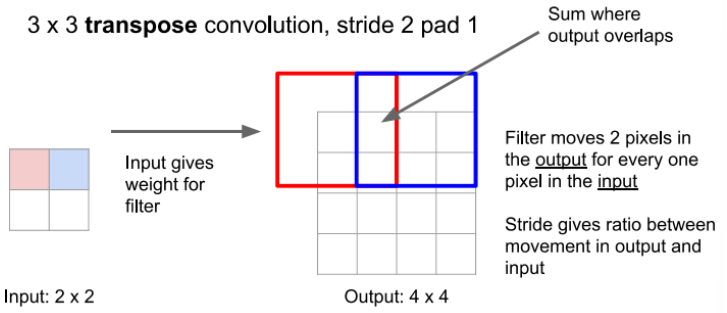




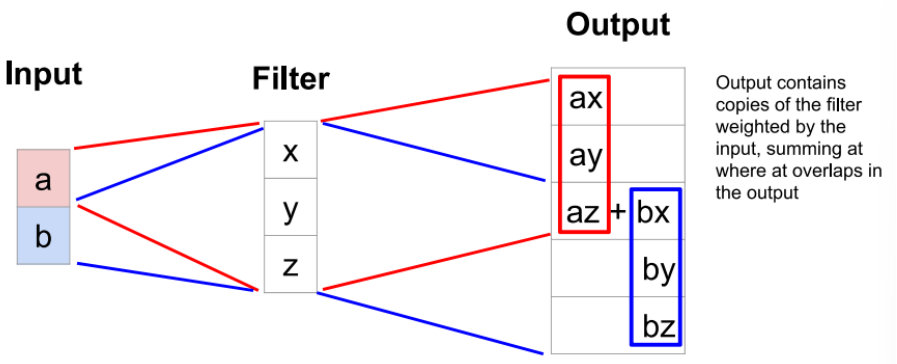


Otherwise, we could use a Learnable Upsampling technique through a transposed convolution. Here on the left we can see a normal convolution 3x3 with stride 2.

In the bottom-left part, instead, we can see how the input itself gives the weight for the Convolutional Filter. Clearly written, also, that "Filter moves 2 pixels in the output for every one pixel in the input".



A really self-explanatory scheme of this **Fractionally Strided Convolution** (or Learnable Upsampling) technique can be found down here.



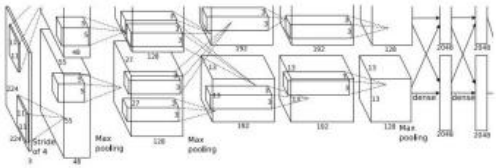
In Semantic Segmentation, we don't differentiate instances, but we only classify pixels.

The instance differentiation is a task of **Instance Segmentation**.

2. Object Segmentation

In Object Segmentation, we don't want to classify the single pixel, but we want to classify multiple objects (Dog, Cat, Cow, etc.) in an image.

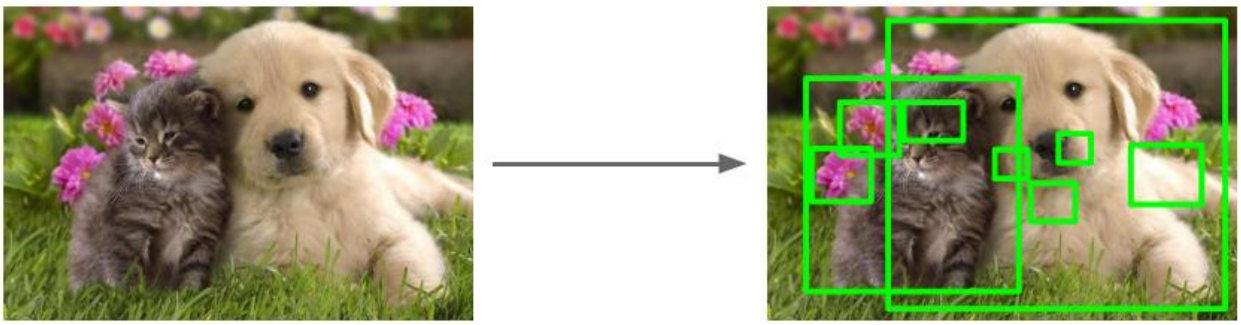
In case of multiple objects, we can apply a CNN to many different crops of the image such that it classifies each crop as object or background.



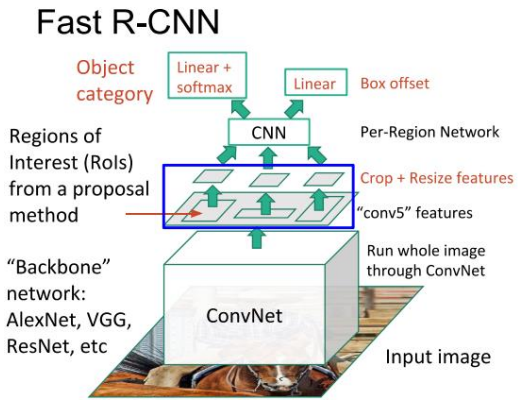
Dog? YES
Cat? NO
Background? NO

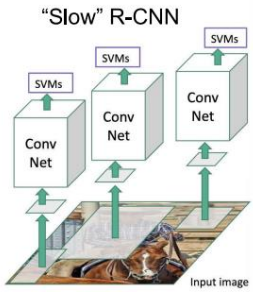
The main problem here is that we need to apply CNN to huge number of locations, scales and aspect ratios, which is very computationally expensive!

So, the solution comes with "Region Proposal", where we find blobby image regions that are most-likely to contain objects. (Fast: 2000 region proposals in few seconds on CPU).



2.1 Fast R-CNN

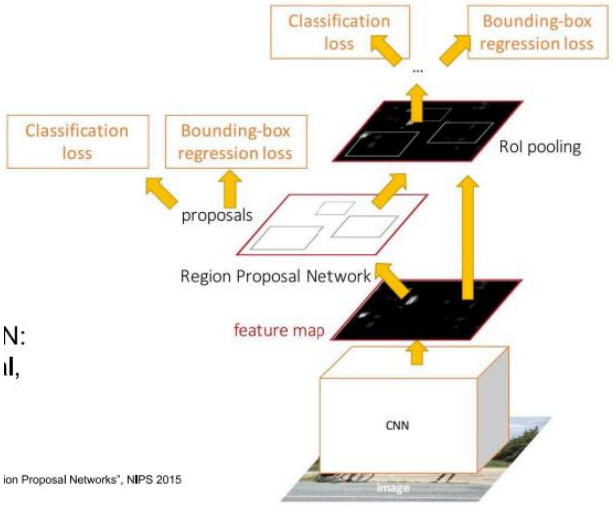




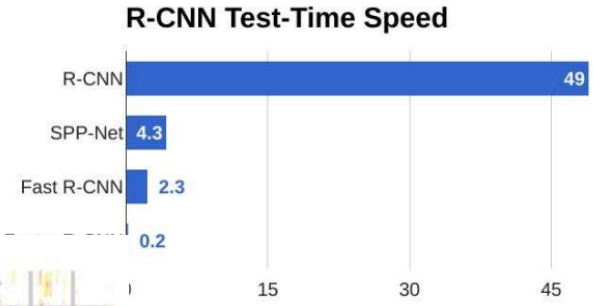
Fast R-CNN is a method in order to exploit the object segmentation problem. Fast R-CNN, also, solves the problem of the "Slow" R-CNN. We can see, in fact, on the right part that in the slower architecture, we are passing the regions to a Convolutional Neural Network before applying the SVM classification, which is really slow, as it has to compute 2k independent forward passes for each image!

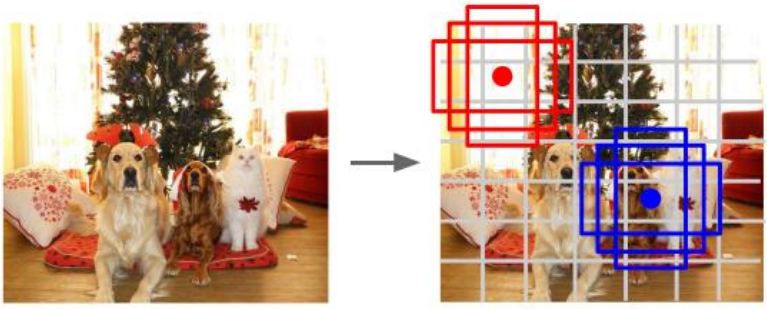
RoIs (Regions of interest) are coming from the fusion of 2000 independent boxes.

The fix for this was, indeed, the Fast R-CNN architecture, where instead of passing the single images crop into the classifier, we are passing the whole image in the CNN and then cropping the Convolutional features (output of the Conv Layer) instead. This results in 8 hours training time instead of 84h (standard R-CNN).



A further optimization showed how it was possible to achieve better results and faster test-time speed if we make the CNN do proposals, meaning that they are predicted from features through CNN. For doing so, we insert the Region Proposal Network.





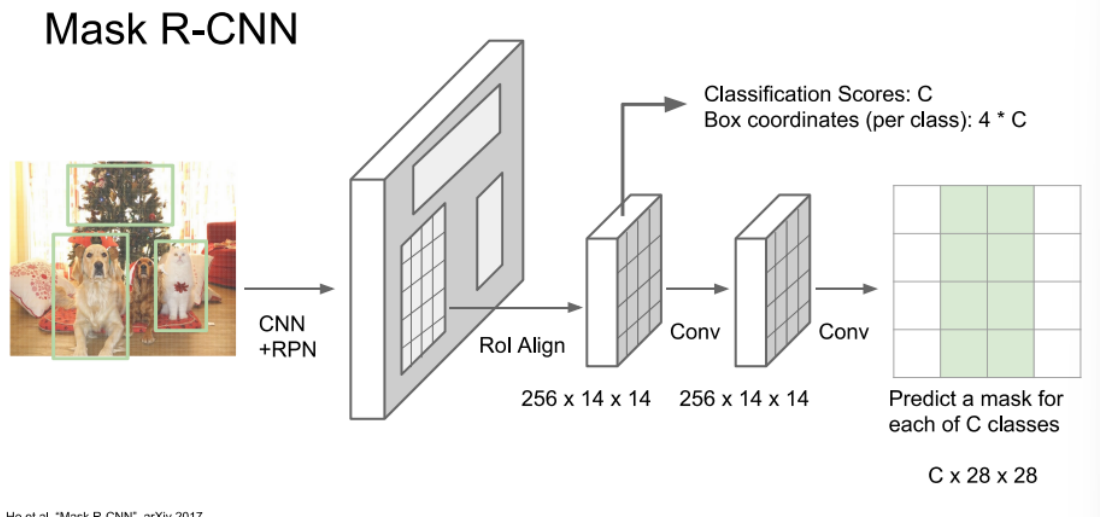
Input image
3 x H x W

Divide image into grid
7 x 7

We also have to give props to the Single-State object Detectors, which predict scores for each cell through a bounding box technique + classification.

2.2 Mask R-CNN

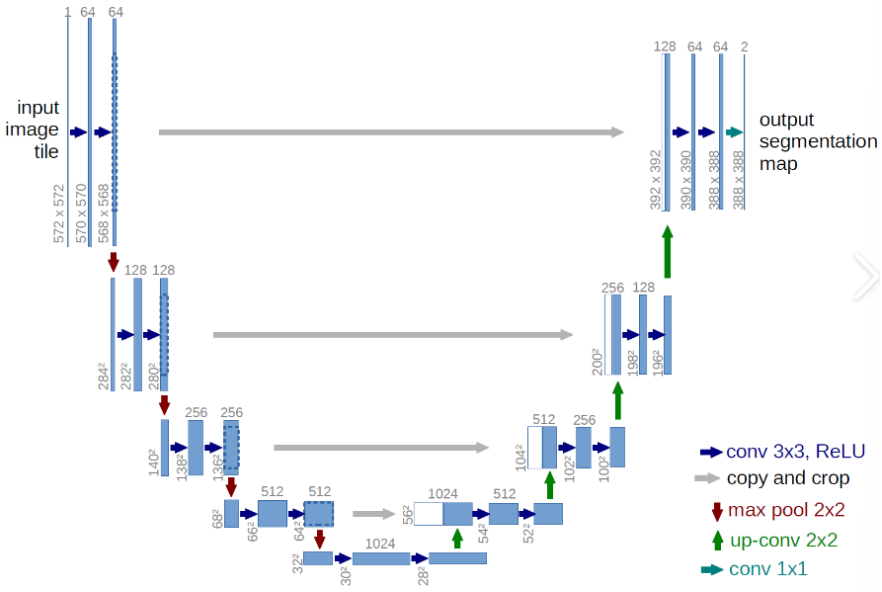
We can see Mask R-CNN as a R-CNN which has a mask prediction network that operates on each RoI (Region of Interest) and predicts a 28x28 binary mask.



Concluding this chapter, worth a note is the 3D object detection (which also need to take track of depth ---> parameters increase a lot) with normal images or with monocular camera, the Mesh R-CNN (3D shape prediction) and other things.

3. U-Net

U-Net is a Convolutional Network for Segmentation. Given an image, it outputs a segmentation map by a very special architecture which we'll explain:



Our image goes through a Convolution phase, where we also increase the spatial size of it (losing spatial information).

After that (it's called the Contraction Phase), we have the Expansion Phase, in which we expand our image by upscaling it once again through Unpooling. In this way we'll recover object details and also the dimension itself.

In between the two phases, we concatenate high-resolution feature maps from the Contraction phase back to the Expansion phase, such that we will recover the "where" information of the objects.