

Departamento de Eletrónica, Telecomunicações e
Informática

Complements of Machine Learning

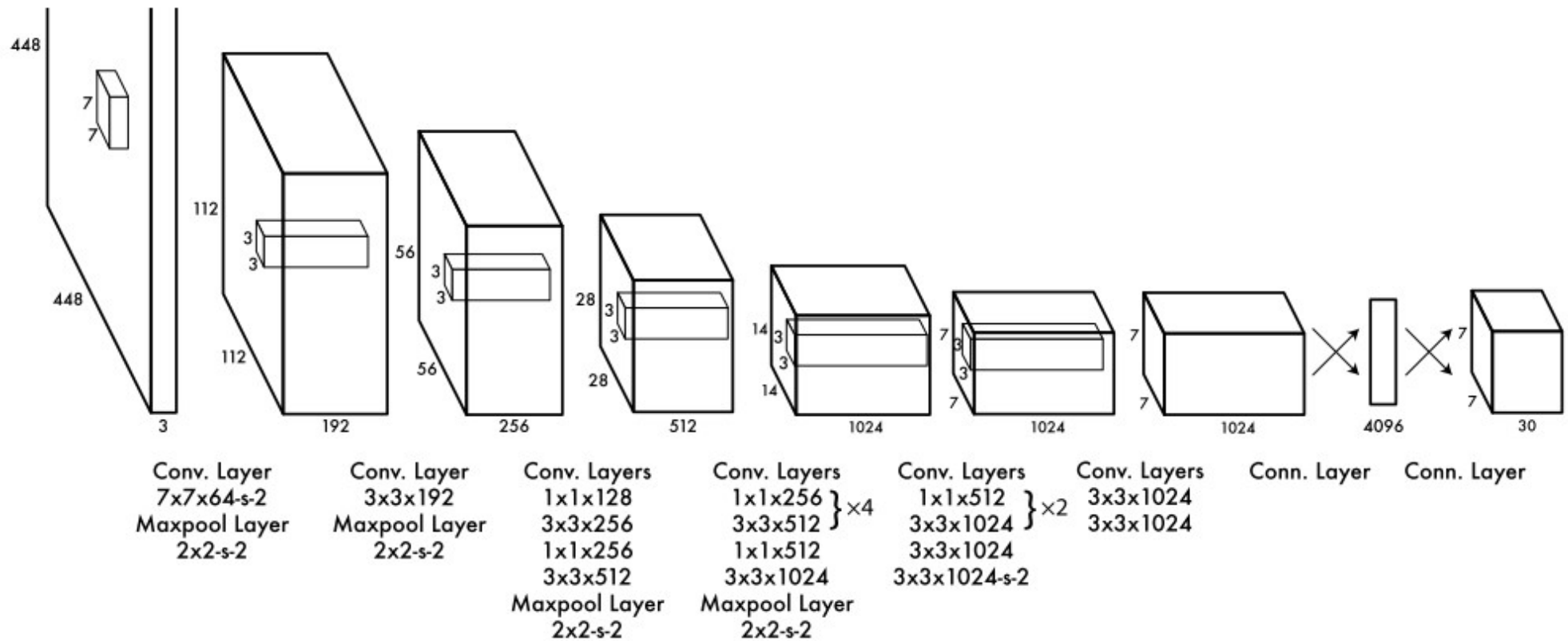
LECTURE 6 : DEEP NN - YOLO, U-NET

Petia Georgieva
(petia@ua.pt)

Outline

- **YOLO (You Only Look Once) algorithm**
- **Other CNNs: Region proposal R-CNN**
- **Image Segmentation (U-Net)**

YOLO (You Only Look Once)



YOLO - CNN network for both classification and localising the object using bounding boxes.

24 convolutional layers + 2 fully connected layers.

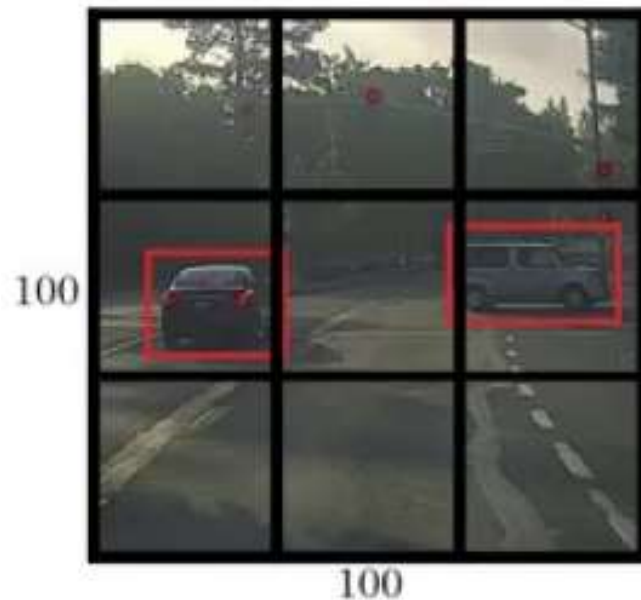
Conv layers pretrained on ImageNet dataset.

*Redmon et al, 2015, "You Only Look Once: Unified, Real-Time Object Detection"

(<https://arxiv.org/abs/1506.02640>)

Redmon & Farhadi, 2016 (<https://arxiv.org/abs/1612.08242>).

YOLO (You Only Look Once) algorithm



\leq cell label: $[0, ?, ?, ?, ? . ?, ?; ?]$
? – “don’t care”

\leq cell label: $[1, b_x, b_y, b_h, b_w, 0, 0, 1]$

\leq cell label: $[0, ?, ?, ?, ? . ?, ?; ?]$
? – “don’t care”

Let we have 100x100 pixels input image. We place a grid on this image.
In the original paper the grid is 19x19, here for illustration is 3x3 grid (9 cells).
Apply object classification and localization to each cell.

How to define the labels used for training:

For each cell, the label is 8 dimensional vector (if we have 3 classes)

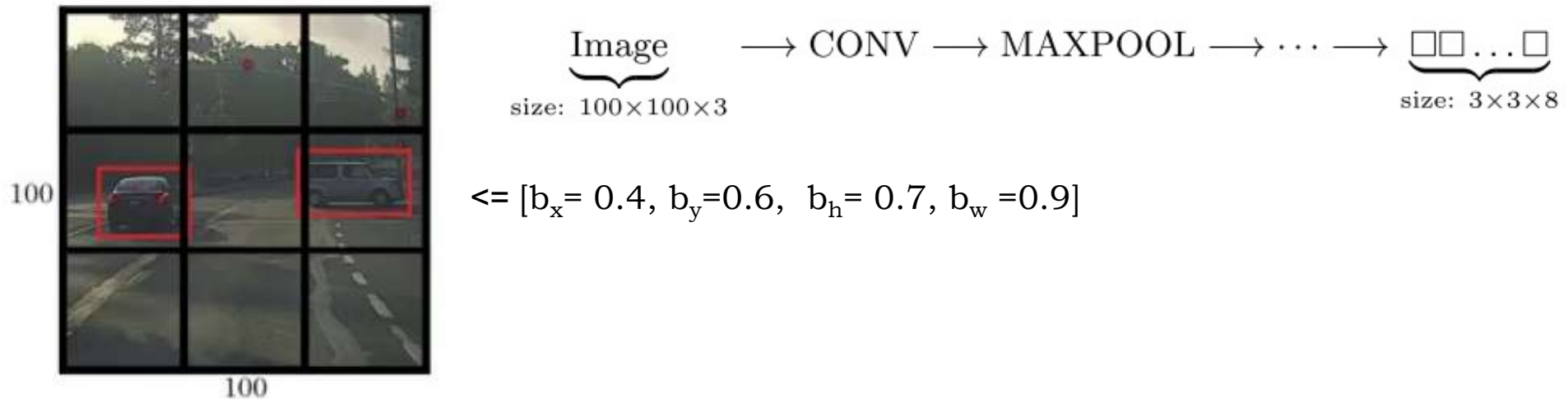
$y = [p_c, b_x, b_y, b_h, b_w, c1, c2, c3]$, where

- p_c (0 or 1) specifies if there is or not an object in that cell.
- $[b_x, b_y, b_h, b_w]$ specify the bounding box if there is an object in that cell.
- $c1; c2; c3$ (0 or 1) - to assign the class (e.g. person, bicycle, car)

YOLO algorithm assigns the object only to the cell containing the midpoint of the object.

Since we have 3x3 grid, the total volume of the target output Y is 3x3x8.

YOLO (You Only Look Once) algorithm



To train YOLO, input e.g. 100x100x3 original image (X).

The network has the usual conv layers, max pool layers, and so on.

It has to end up with 3x3x8 output volume that is compared with the target labels Y (also 3x3x8).

If the grid is much finer (e.g. 19x19), it reduces the chance of multiple objects assigned to the same grid cell.

How to specify the bounding boxes $[b_x, b_y, b_h, b_w]$?

YOLO assumes the upper left point of the cell is (0,0), the lower right point is (1,1).

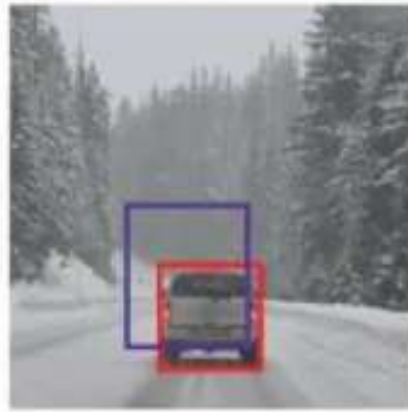
The bounding boxes are specified relative to the grid cell.

$[b_x, b_y]$ represent the midpoint of the object, have to be <1 and >0 .

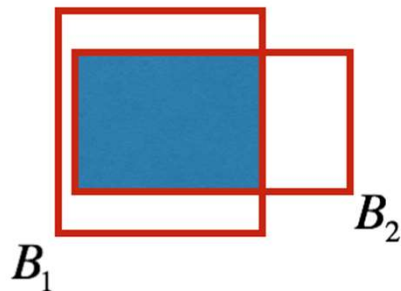
$[b_h, b_w]$ represent the height and width of the bounding box, they could be >1 if the object is bigger than the cell.

There are other ways to specify the bounding boxes.

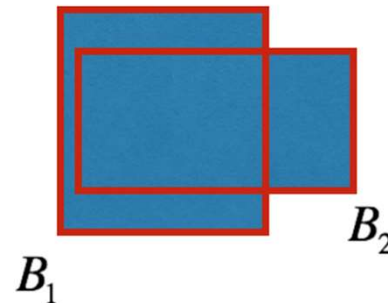
Intersection Over Union



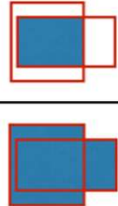
Intersection



Union



Intersection over Union

$$IoU = \frac{B_1 \cap B_2}{B_1 \cup B_2} = \frac{\text{Intersection}}{\text{Union}}$$


How to tell if the object detection algorithm is working well?

We use the function called intersection over union (IoU) .

IoU computes the intersection over union of the two bounding boxes (red is the ground true, purple is the YOLO prediction).

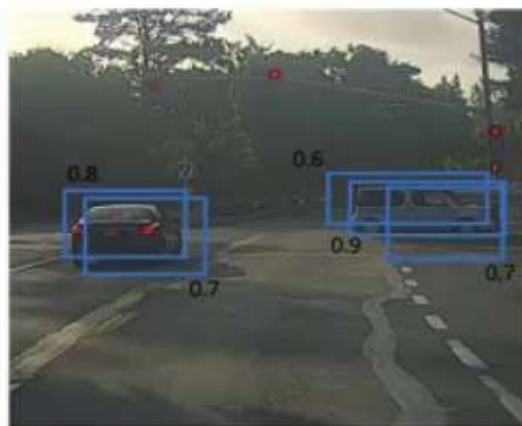
IoU is a measure of the overlap between two bounding boxes (measure of similarity).

Higher IoU, better prediction:

IoU=1 => perfect detection

IoU >=0.5 => “correct” (by convention)

Non-max Suppression



Common object detection problem: the algorithm finds multiple detections of same object =>

Non-max suppression is a way to guarantee that each object is detected only once.

Let's say we want to detect 3 classes - persons, bikes, cars.

We placed 19x19 grid. Technically each car has just one midpoint, so it should be assigned just to one grid cell.

In practice, we run object classification & localization algorithm for every cell.

It's possible that many neighbour cells claim they found a car.

Non-max suppression cleans up these detections independently for each class (persons, bikes, cars).

- 1) Discard all boxes with low probability (let say $p_c \leq 0.6$)
- 2) Pick the box with the largest p_c . Output that as a prediction.
- 3) Discard any remaining box with high overlap (e.g. $\text{IoU} \geq 0.5$) with the box picked in step 2).

Anchor boxes



Previously, each object in training image is assigned to grid cell that contains that object's midpoint. The label vector is (8 elements)

$$y = [p_c, b_x, b_y, b_h, b_w, c1, c2, c3]$$

Now with two anchor boxes, each object in training image is assigned to grid cell that contains object's midpoint and anchor box for the grid cell with the highest IoU. The label vector is

Anchor box 1: Anchor box 2:



$$y = [\underbrace{p_c; b_x; b_y; b_h; b_w; c_1; c_2; c_3}_{\text{anchor box 1}}; \underbrace{p_c; b_x; b_y; b_h; b_w; c_1; c_2; c_3}_{\text{anchor box 2}}]$$

Object detection problem: Each grid cell can detect only one object. What if a grid cell has multiple objects?

One idea is to use anchor boxes. In the image above, the midpoints of the person and the car are very close and fall into the same grid cell => the algorithm has to pick one of the two detections to output.

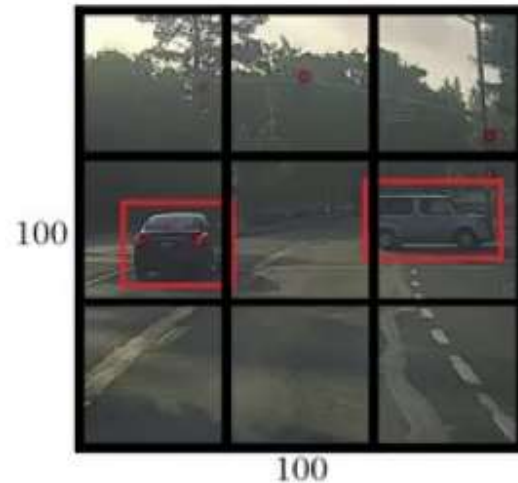
The idea of anchor boxes is to pre-define two different shapes called, anchor boxes and associate each prediction with one of the anchor boxes.

For example, anchor box for human may be a vertical rectangle while that for car may be a horizontal rectangle.

e.g. The label for the lower middle grid cell of the image above has 16 elements:

$$y = [1, b1_x, b1_y, b1_h, b1_w, 1, 0, 0, \quad 1, b2_x, b2_y, b2_h, b2_w, 0, 0, 1]$$

YOLO Training



Get labelled data: set of $X_p \times Y_p \times 3$ images;

Label dimension (**tensor**): (grid of cells) \times (# anchors) \times ($p_c + 4$ box coordinates + #classes)

In general, we may use multiple anchor boxes (e.g. 5 in the lab).

Train CNN with the same output dimension as data labels.

Ex. Label for the upper right cell of the image above (assuming 2 anchors (tall, wide),
3 classes)

$y = [0, ?, ?, ?, ?, ?, ?, ?, 0, ?, ?, ?, ?, ?, ?, ?, ?]$ (? Don't care)

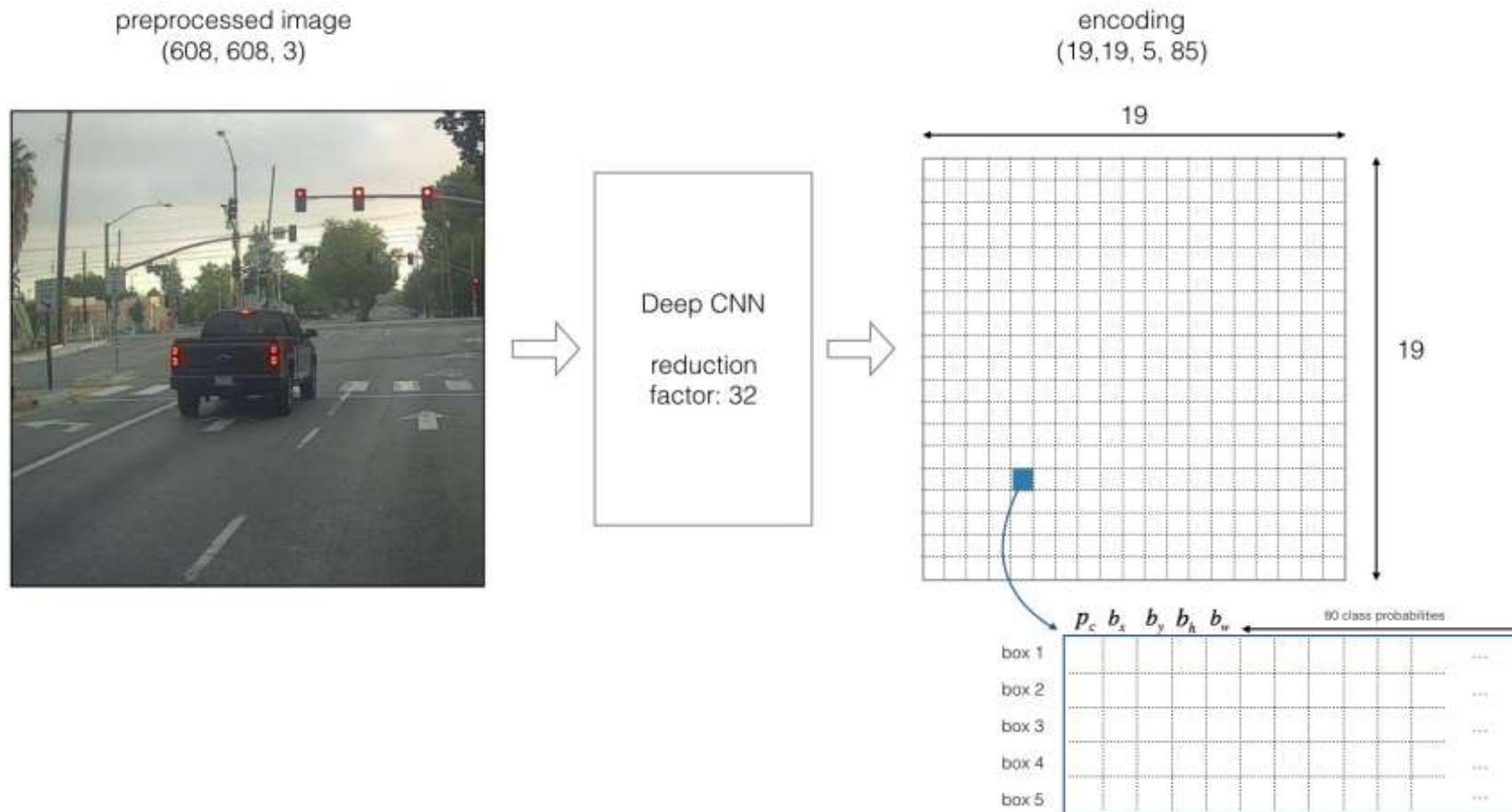
Label for the right middle cell of the image above:

$y = [0, ?, ?, ?, ?, ?, ?, ?, 1, b2_x, b2_y, b2_h, b2_w, 0, 0, 1]$

Ex. Input (100x100x3 image) \Rightarrow CNN \Rightarrow Output (3x3x2x8)

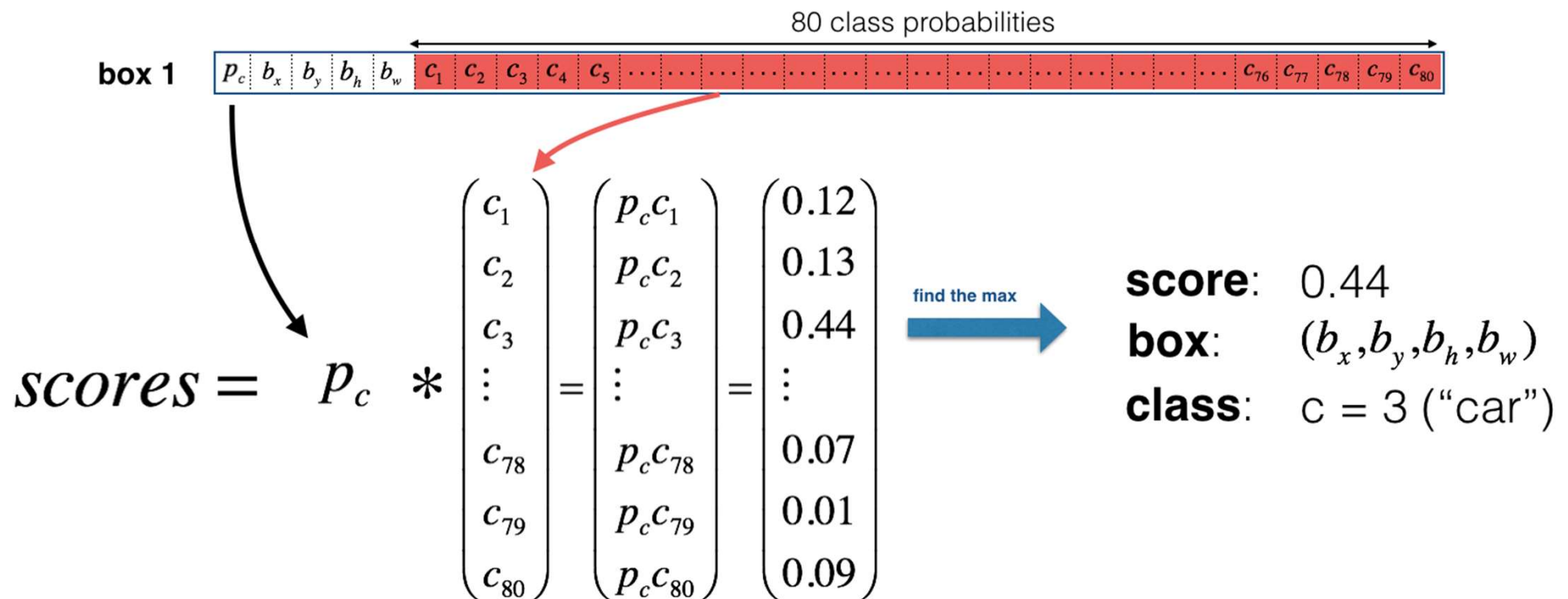
CNN outputs always numbers, cannot output don't care.

Image encoding (19x19 cell grid, 80 classes, 5 anchor boxes)



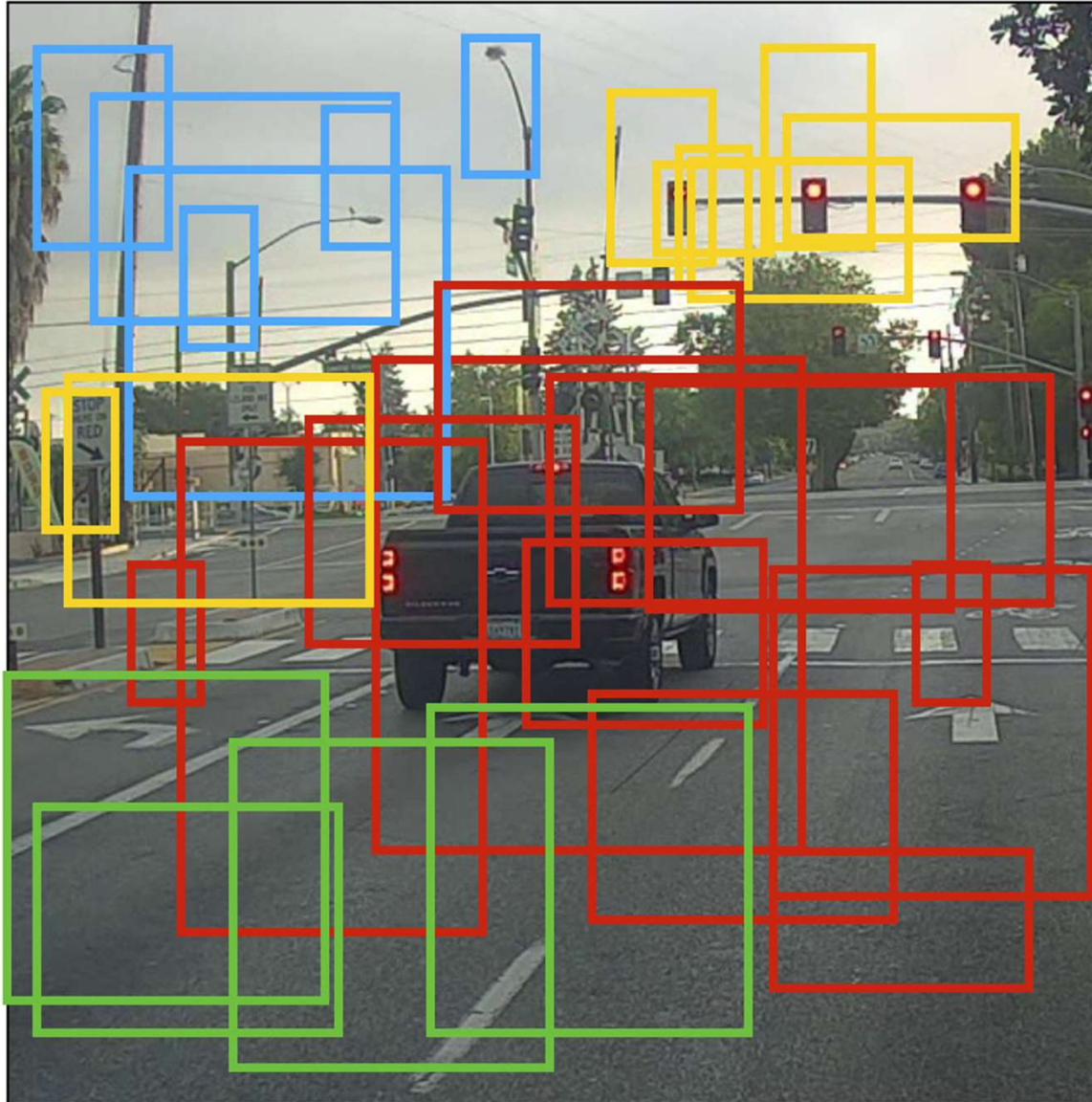
YOLO runs fast and is useful for real time object detection, because it is one pass convolutional implementation. It is not implemented sequentially for each cell (19x19 grid).

CNN output processing & filtering



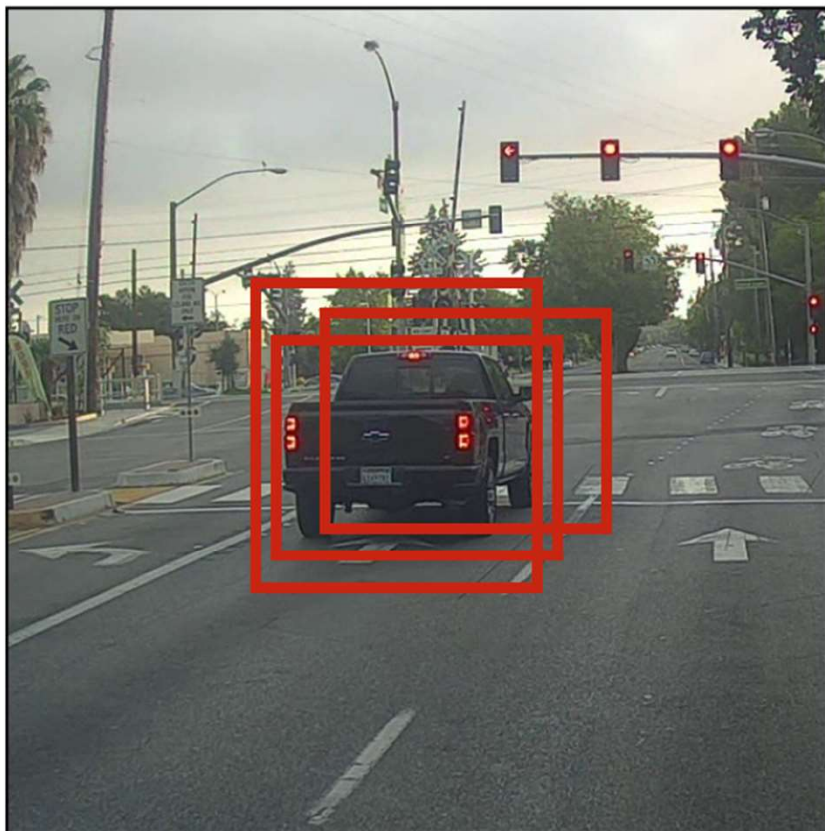
the box (b_x, b_y, b_h, b_w) has detected $c = 3$ ("car") with probability score: 0.44

Remove boxes with low probability (scores)

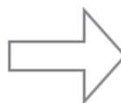


Non Max Supression (NMS)

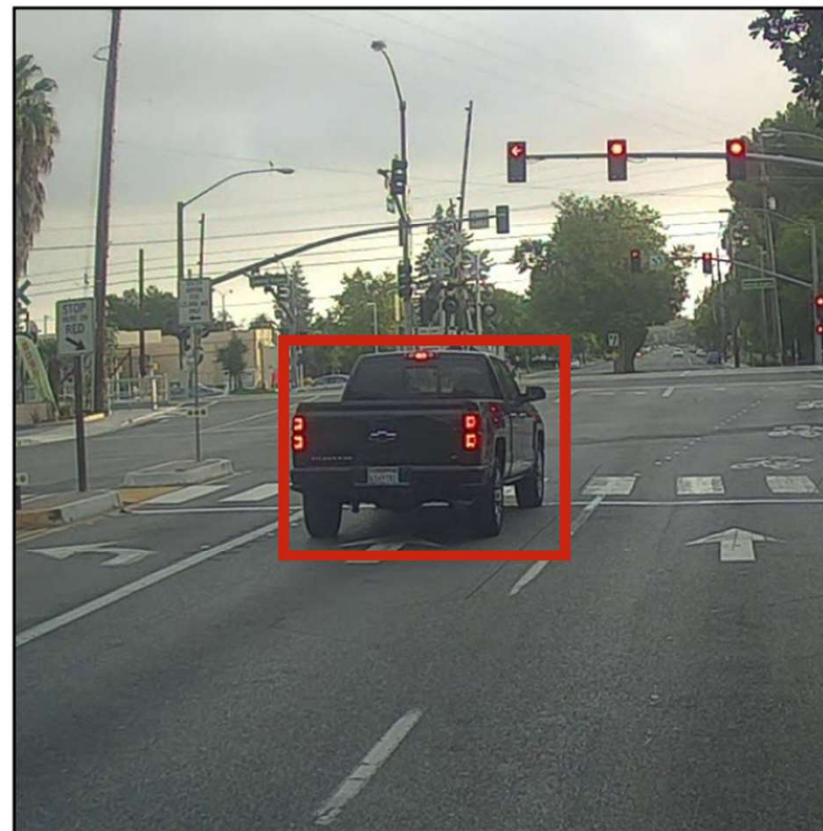
Before non-max suppression



Non-Max
Suppression



After non-max suppression



Object Detection with Region proposal CNN



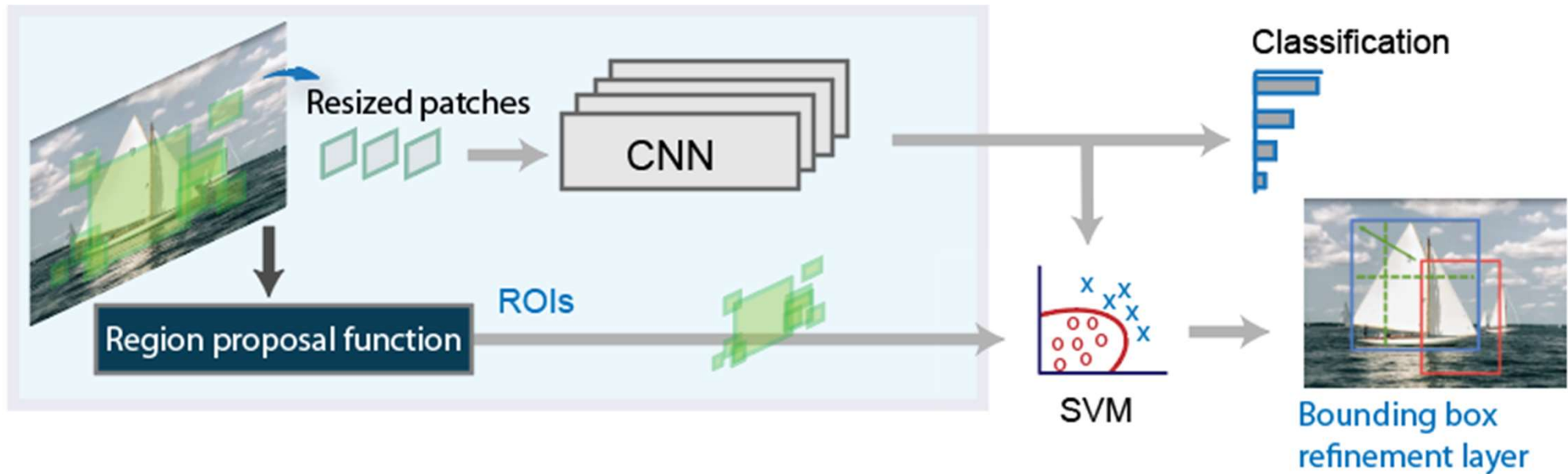
YOLO algorithm classifies a lot of empty regions.
Region proposal CNNs select just a few regions (windows).

First run a segmentation algorithm, find interesting blobs (e.g. 2000), place bounding boxes around them and run CNN classifier.

Object detection using regions with CNNs goes over the following steps:

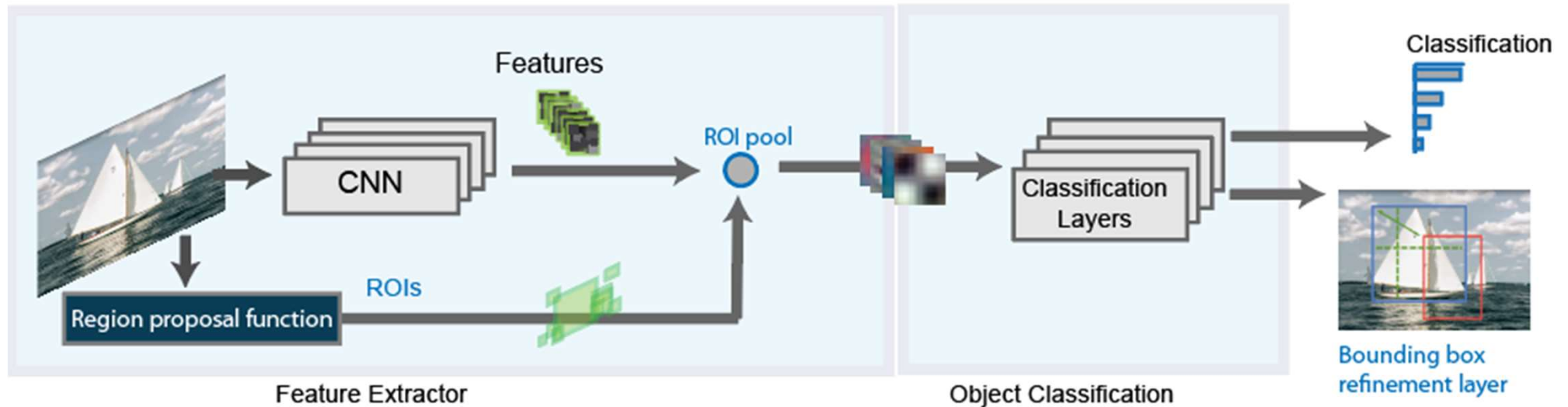
- 1) Find regions in the image that might contain an object. These regions are called *region proposals*.
- 2) Extract CNN features from the region proposals.
- 3) Classify the objects using the extracted features.

R-CNN model



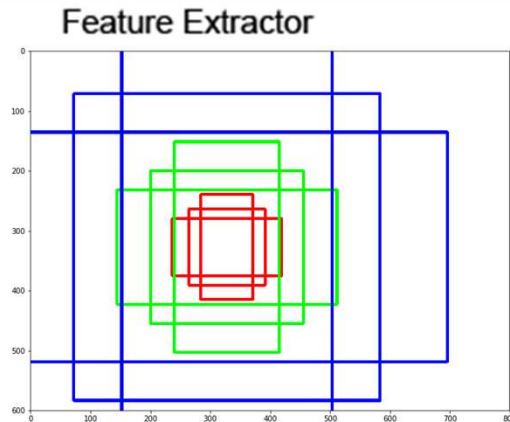
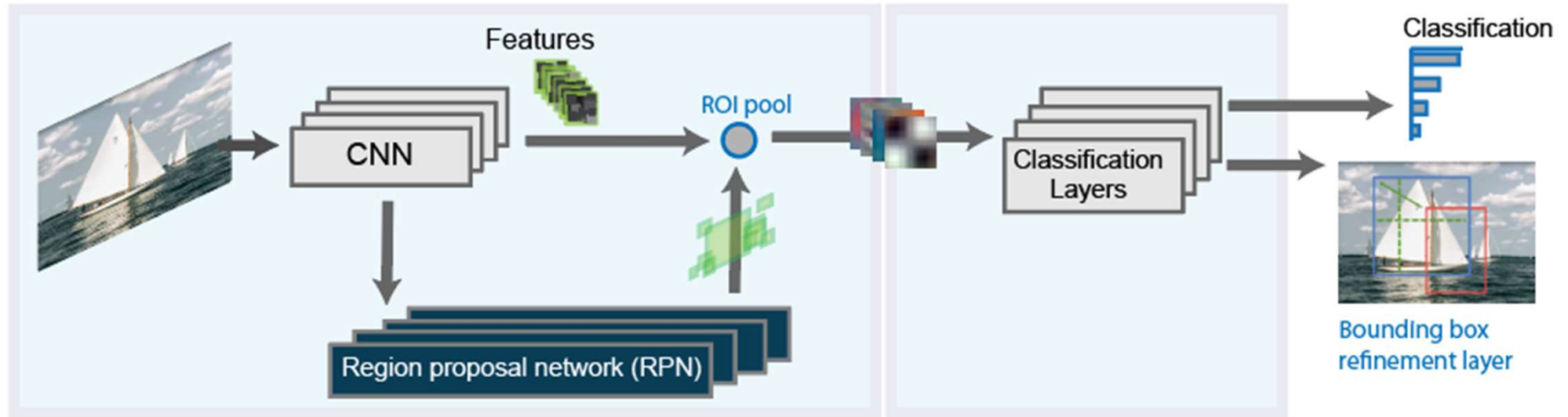
R-CNN detector first generates region proposals (i.e. segmentation) using an algorithm such as Edge Boxes. The proposal regions are cropped out of the image and resized. Then, the CNN classifies the cropped and resized regions. Finally, the region proposal bounding boxes are refined by a support vector machine (SVM) that is trained using CNN features.

Fast R-CNN model



As in the R-CNN, the Fast R-CNN detector also uses an algorithm like Edge Boxes to generate region proposals (segmentation). Unlike the R-CNN, 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 it classifies all regions simultaneously.

Faster R-CNN model



Typical anchor boxes shapes
scales or sizes: 128x128, 256x256, 512x512.
height width ratios 1:1, 1:2 and 2:1

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. The RPN uses Anchor Boxes for Object Detection. Generating region proposals in the network is faster and better tuned to your data.

Summary: R-CNN/Fast R-CNN/Faster R-CNN



R-CNN*: Apply segmentation algorithms to propose regions.
Classify proposed regions one at a time. Output label + bounding box.

Fast R-CNN**: Apply segmentation algorithms to propose regions.
Use convolution implementation of sliding windows to classify all the proposed regions simultaneously.

Faster R-CNN***: Use CNN to propose regions (instead of traditional segmentation alg.).
Use convolutional implementation of sliding windows to classify the regions simultaneously.

* Girshik et. al, 2013. Rich feature hierarchies for accurate object detection and semantic segmentation.

** Girshik, 2015. Fast R-CNN

*** Ren et. al, 2016. Faster R-CNN: Towards real-time object detection with region proposal networks.



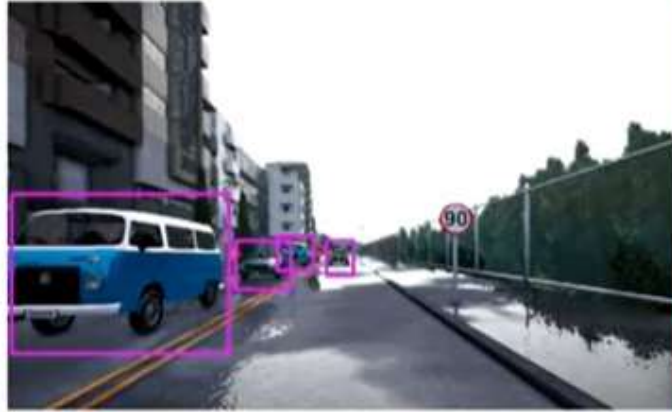
PART 2 Image Segmentation

1. **Object detection vs Image Segmentation**
2. **Semantic Segmentation**
3. **Transpose Convolutions**
4. **U-net architecture (lab work)**

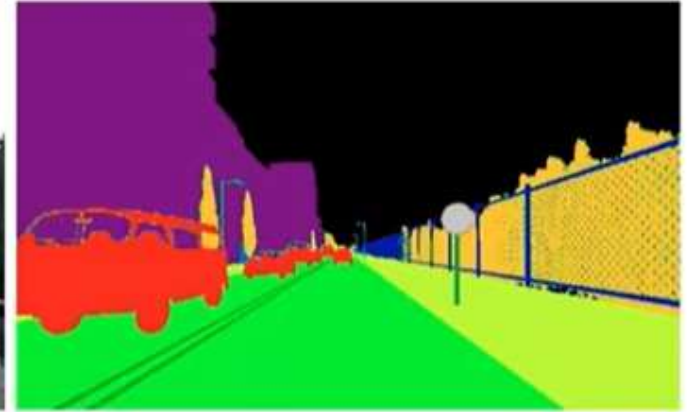
Object detection vs Semantic Segmentation



Input image



Object Detection



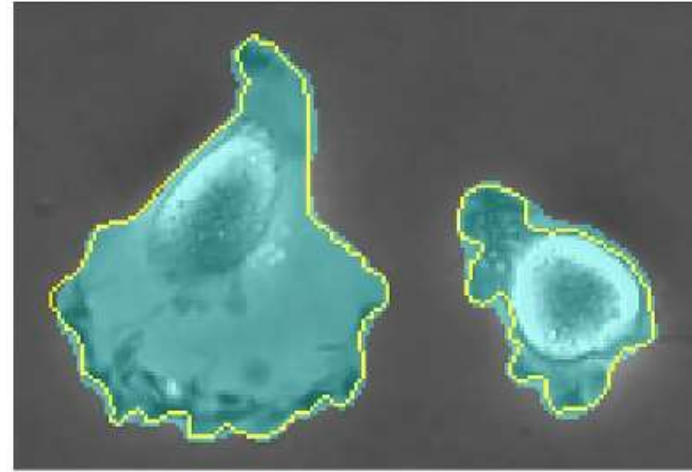
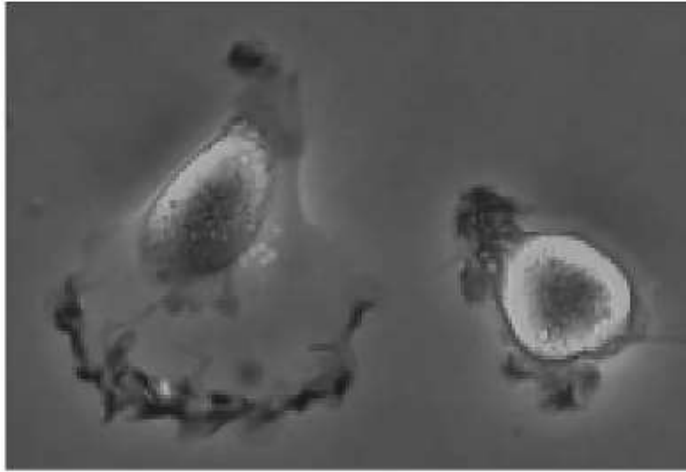
Semantic Segmentation

Object detection - the goal is to put a bounding box around a found object.

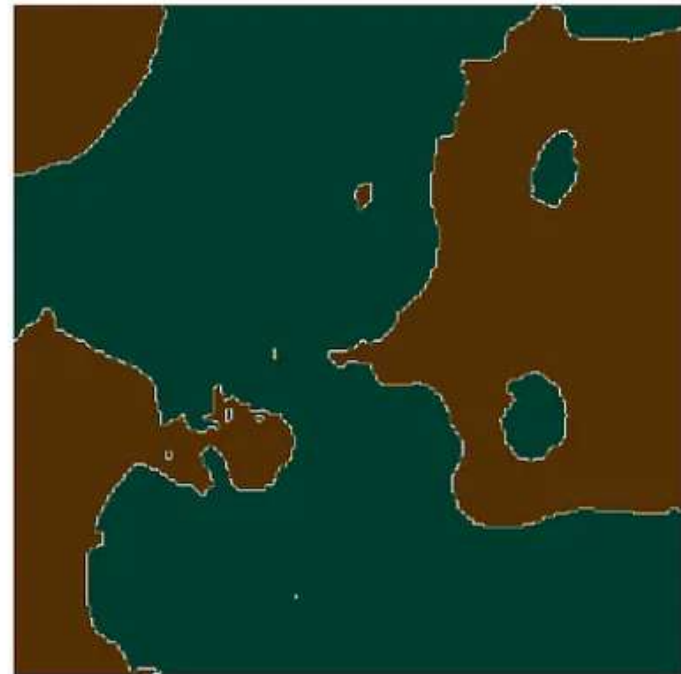
Semantic segmentation - more sophisticated learning algorithm, the goal is to know exactly which pixels belong to the object and which pixels don't, to know what is every single pixel in the image.

Particularly useful in medical imaging, satellite images.

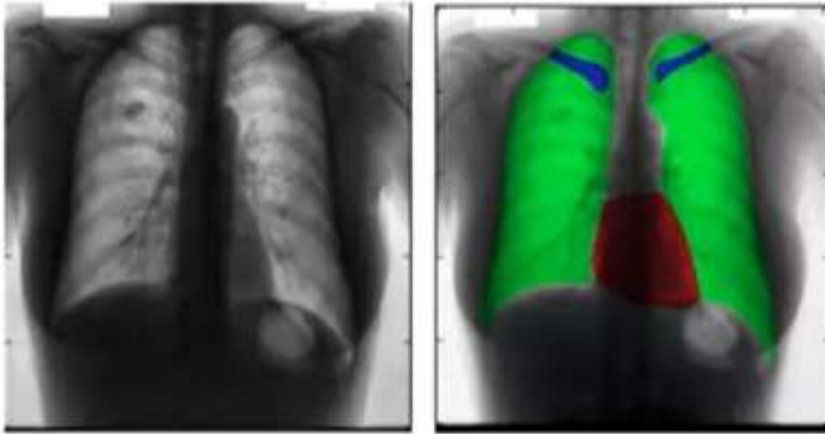
Semantic Segmentation



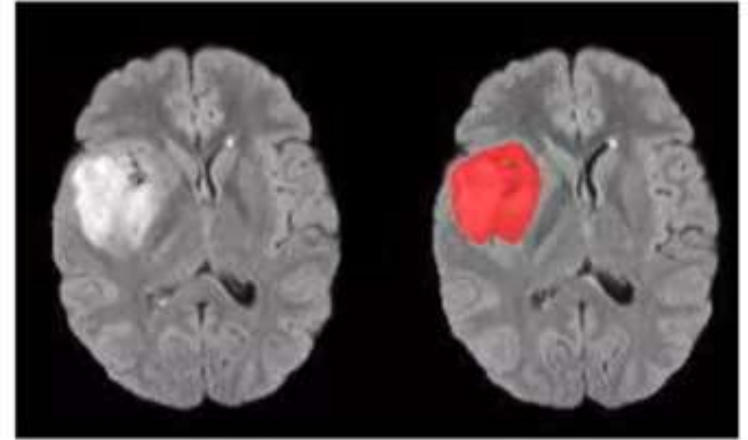
Segment the image into cell and non-cell



Semantic Segmentation



Chest X-Ray

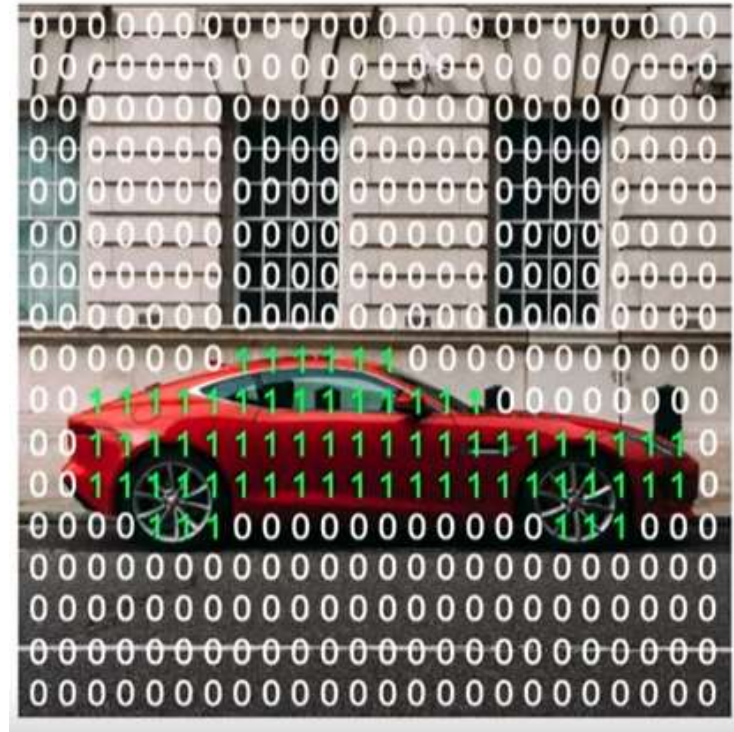


Brain MRI

Left image: lungs, heart, clavicle are segmented with different colors. With the segmentation is easier to spot irregularities, diagnose diseases, help surgeons when planning surgeries.

Right image: brain MRI scans are used for brain tumor detection. Manually segmenting the tumor is very time-consuming and laborious. If a learning algorithm can segment out the tumor automatically; this saves radiologist's time. It is also useful for surgical planning.

Per-pixel class label – example 1

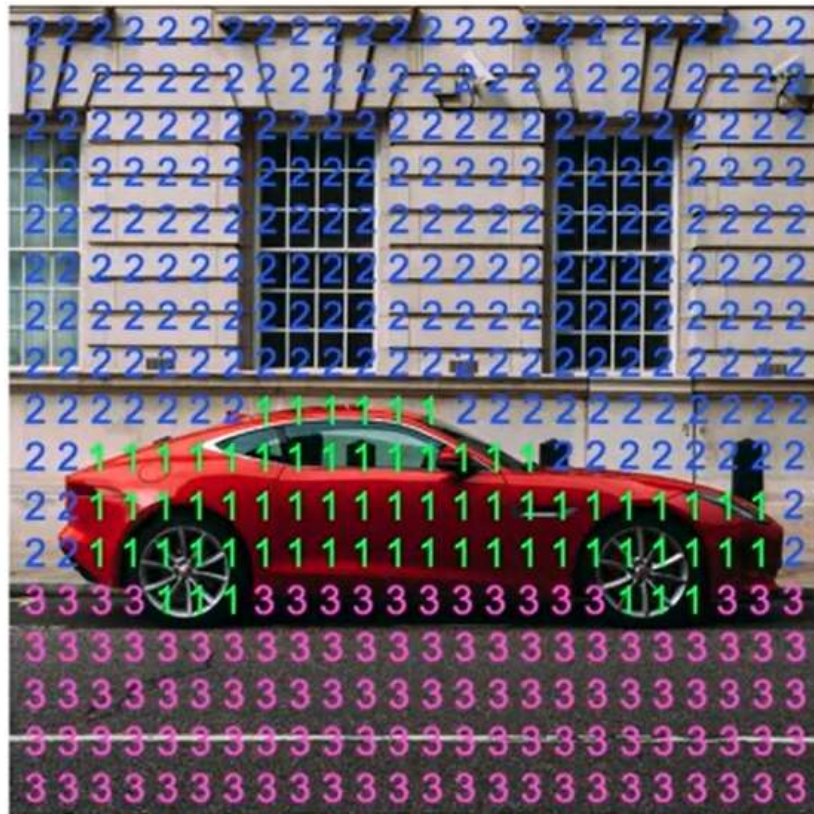


The task here is segmenting the car from the background.

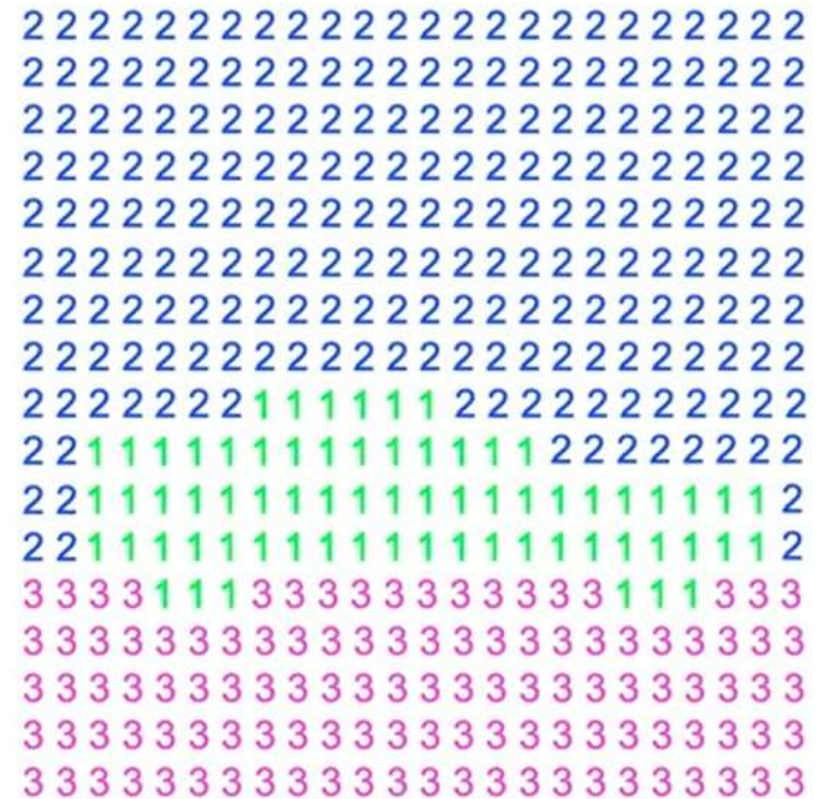
Segmentation algorithm outputs, **1 or 0 for every pixel** in the image.

Pixel should be labeled 1, if it is part of the car; 0 if it's not part of the car.

Per-pixel class label- example 2



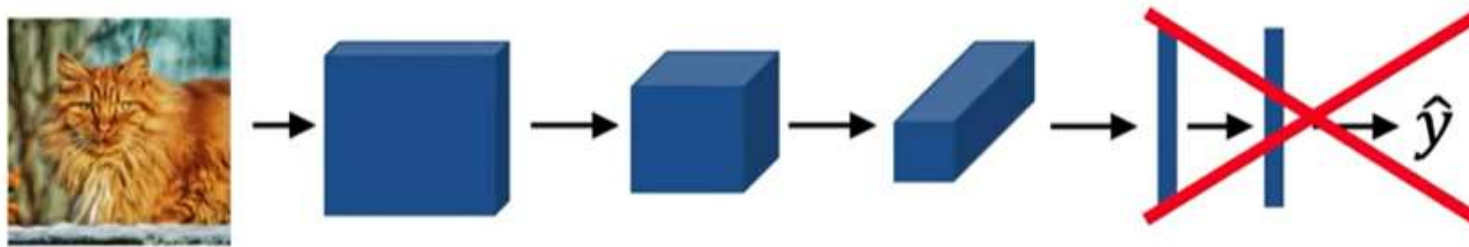
1. Car
2. Building
3. Road



Segmentation Map

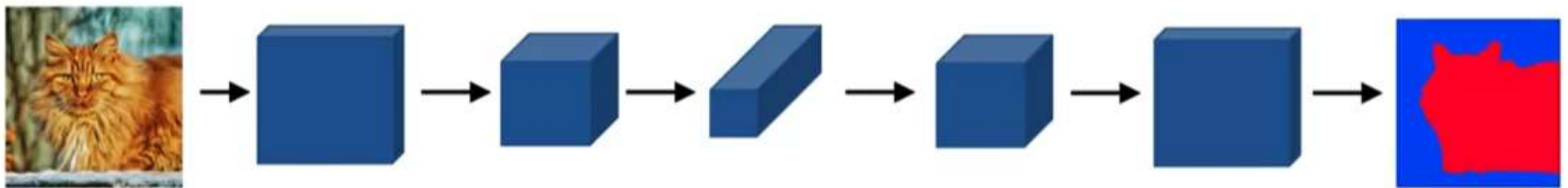
NN architecture suitable for such task is U-net

U-net architecture intuition



Standard CNN architecture: the input is an image, it goes through multiple layers in order to generate a class label.

Main trend in conv layers: height and width decrease, number of channels increase

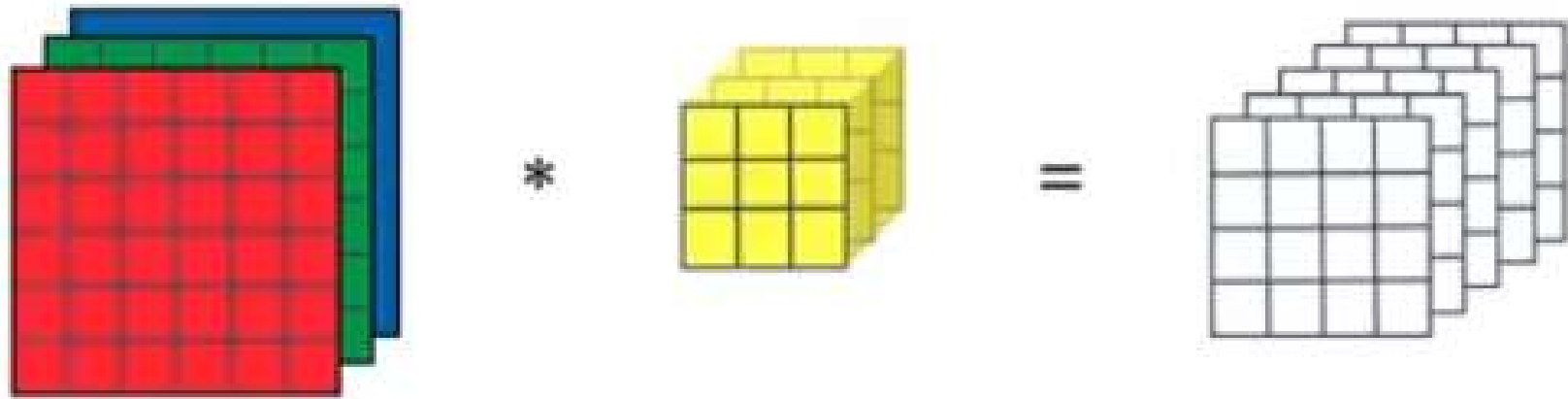


Semantic Segmentation CNN architecture: eliminate the last few (fully connected) layers.

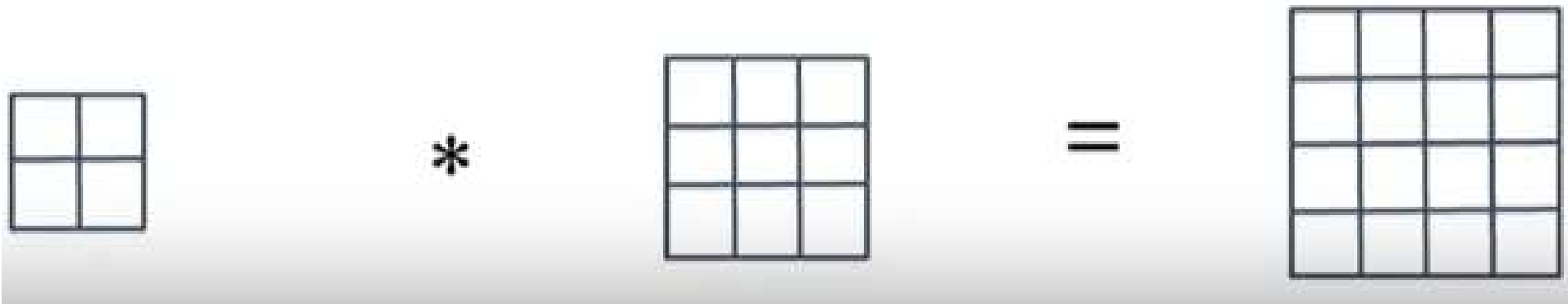
Key semantic segmentation step: First the image size is getting smaller as it goes from left to right, then it gradually gets back to its full-size at the network output. We get the segmentation map of the input image.

Normal vs Transpose Convolution

Normal Convolution



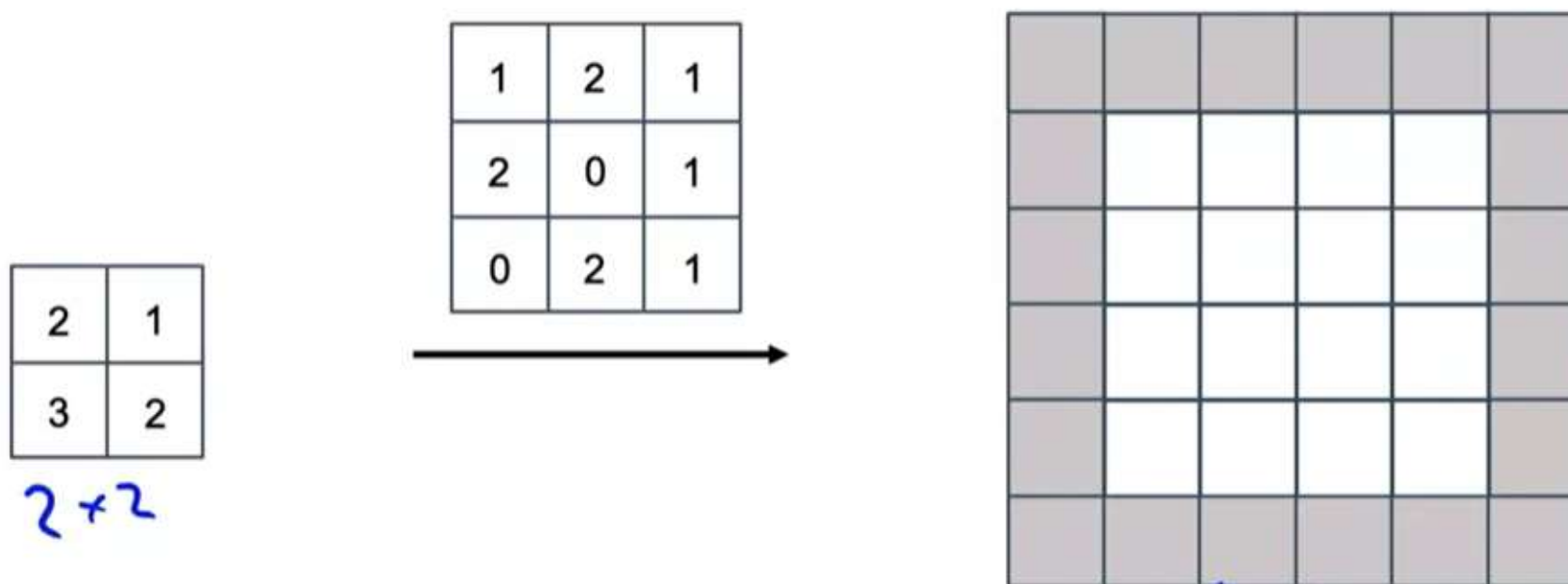
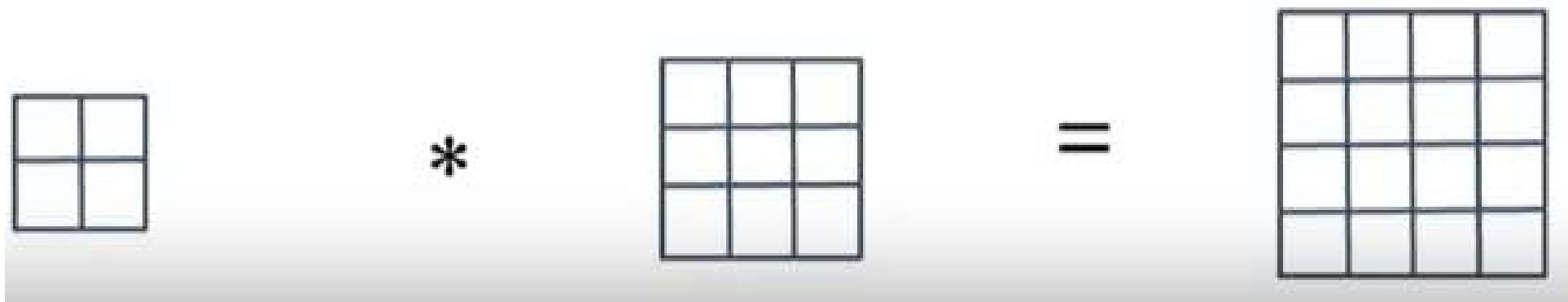
Transpose Convolution



Transpose Convolutions

From 2x2 feature map, with 3x3 filter, get 4x4 feature map,
padding $p=1$, stride =2

Other names: deconvolution, upconvolution, backward strided convolution



Transpose Convolutions

Each value of the input matrix multiplies each value of the filter.

Here 1st (value 2) and 2nd (value 1) elements of the input matrix are weights of the filter.

2	1
3	2

2x2

1 ²	2 ²	1 ²
2 ²	0 ²	1 ²
0 ²	2 ²	1 ²

2	1
3	2

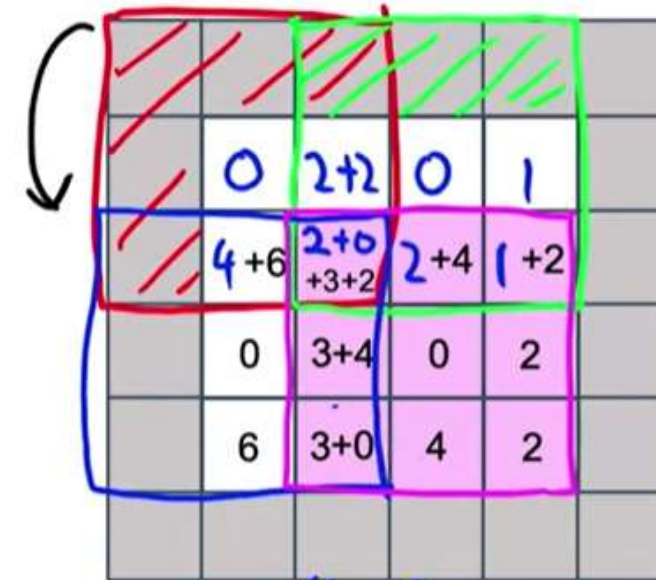
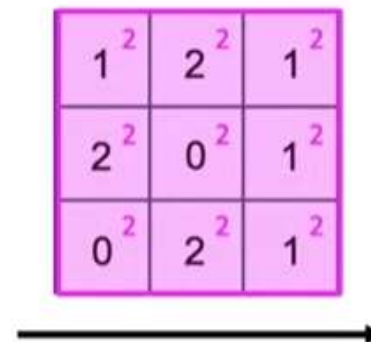
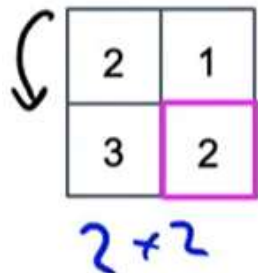
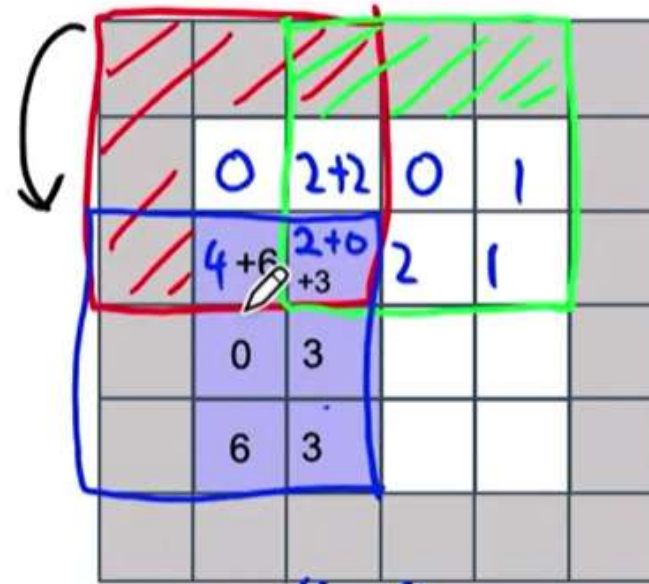
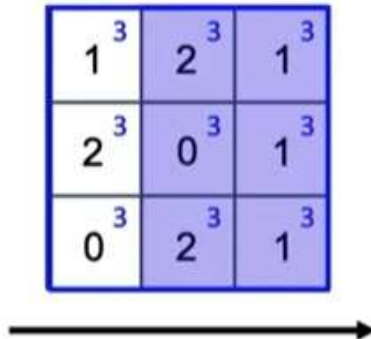
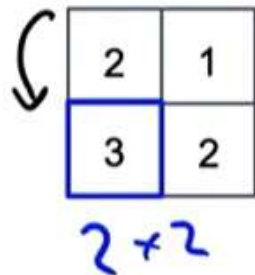
2x2

1 ¹	2 ¹	1 ¹
2 ¹	0 ¹	1 ¹
0 ¹	2 ¹	1 ¹

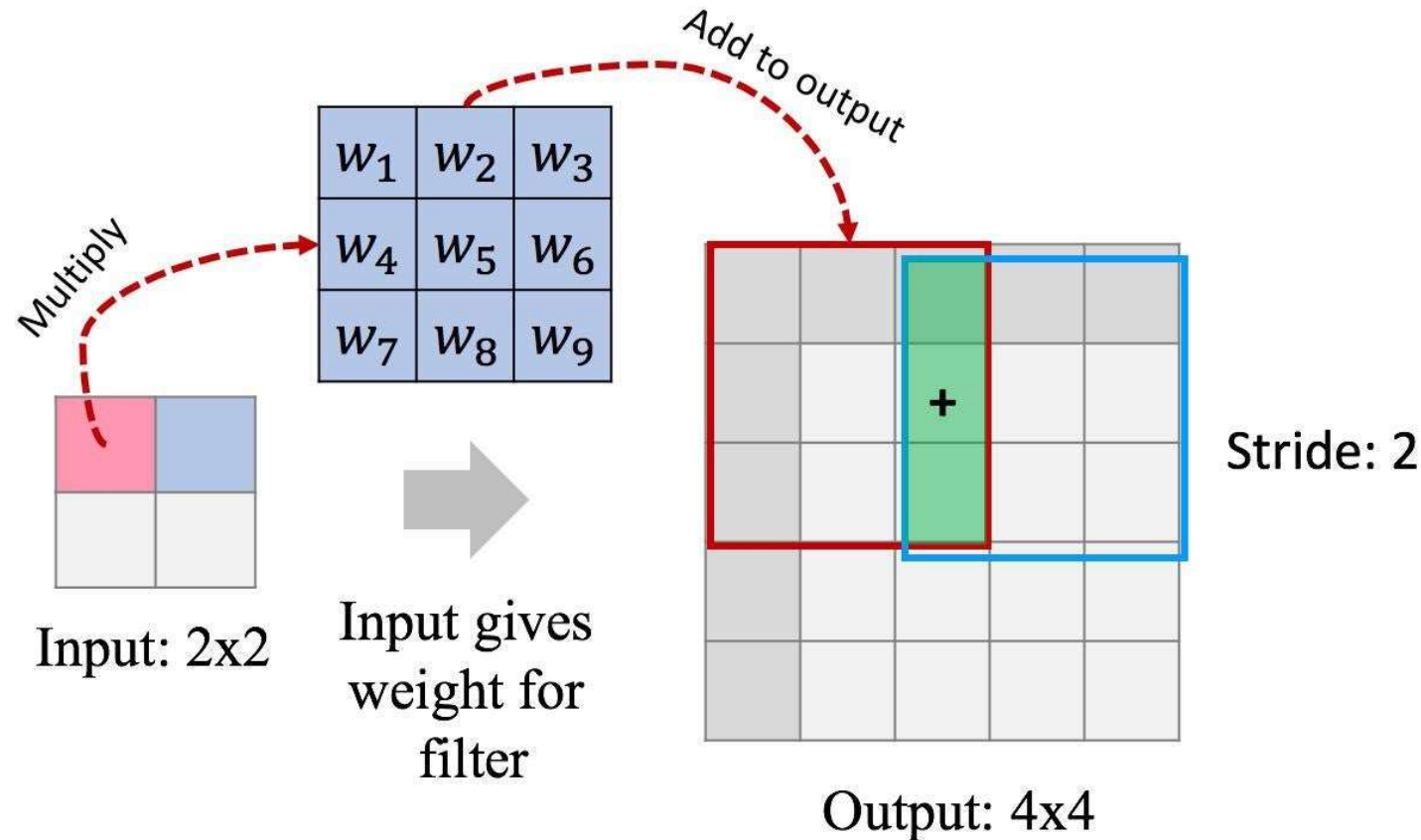
<= Sum
where output
overlaps,
stride =2

Transpose Convolutions

Here 3rd (value 3) and 4th (value 2) elements of the input matrix are weights of the filter, stride = 2.



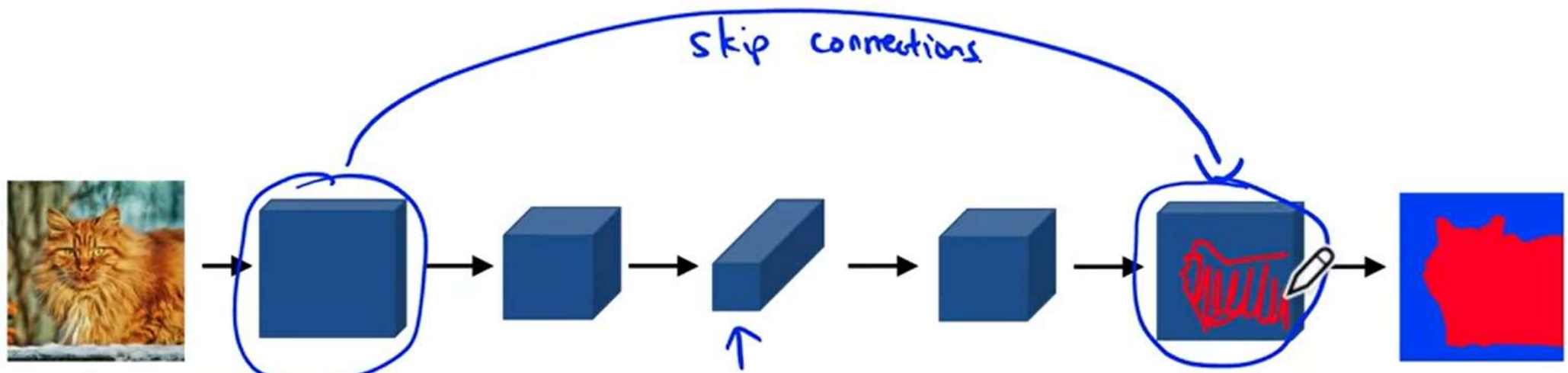
Transpose Convolutions



Ex. Take a 2-by-2 inputs and blow it up into a 4- by-4-dimensional output

There are multiple possible ways to take small inputs and turn them into bigger outputs. Transpose convolution is one of them and when the filter parameters are learned well it gives good results in the context of the U-net.

U-net Architecture Intuition



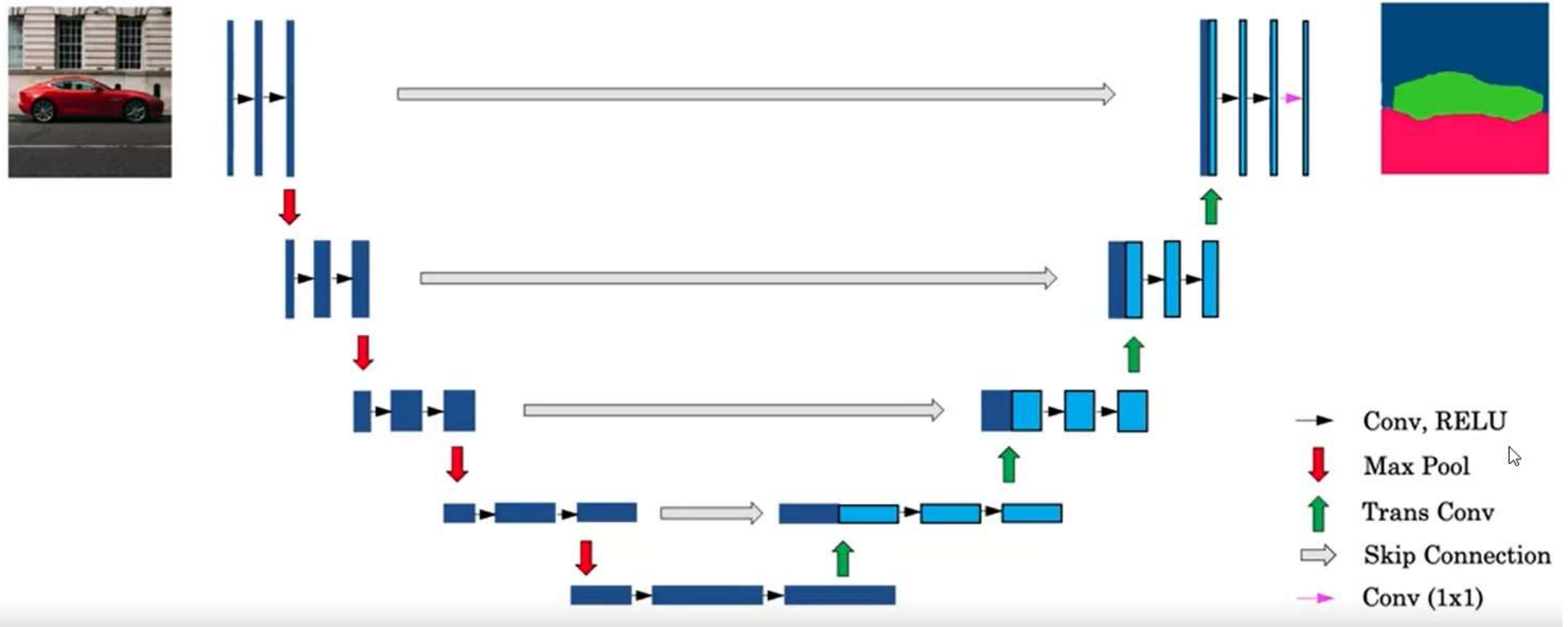
Left half (Encoder) of U-net uses normal convolutions. The image will be compressed. From a large image go to volume with small height and width (detailed spatial information is lost) but it's much deeper.

Right half (Decoder) uses the transpose convolution to blow the representation size up back to the size of the original input image.

With **skip connections** from earlier layers to later layers two types of info are got:

- i) High level spatial /contextual information but with lower resolution from the previous layer;
- ii) Low level feature information but with high resolution from the early layer.

U-Net



Encoder: sequence of normal feedforward (FF) conv layers (3D) + Relu act. functions & Max Pool to reduce height and width – each block has 3 conv layers.

Decoder: sequence of transpose convolution + skip connection+ normal conv layers. Increase the height and width, until it gets back to original height and width

Input image: height x width x 3

Output volume: height x width x Number of classes.