Object Localization

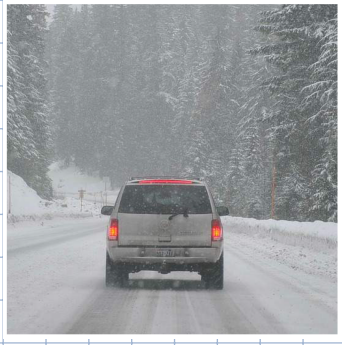
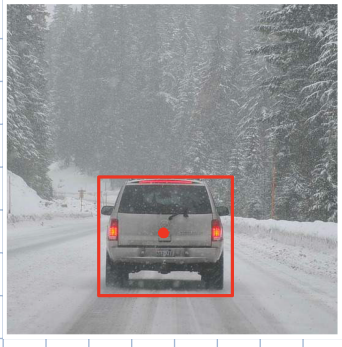


Image Classification: Given an image, label if it's a car or non-car.

Only one object is in the input image.



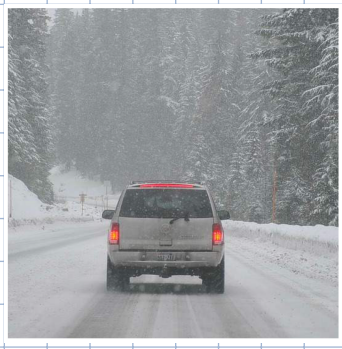
Classification with Localization: Given an image, label if it's a car or non-car and find where the car is (Draw a bounding box).

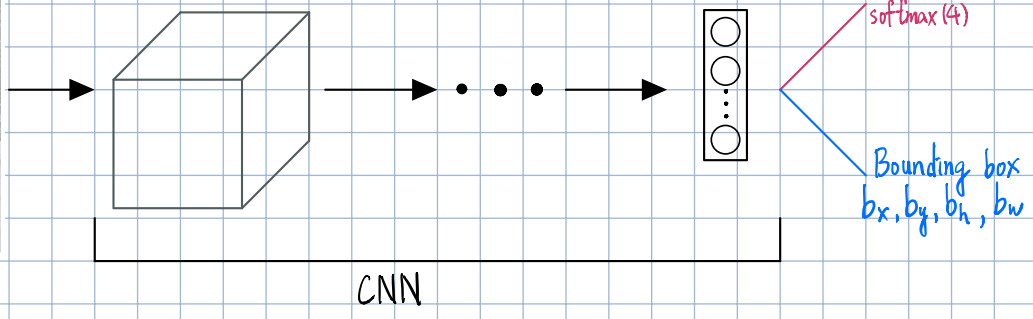
Only one object is in the input image.

Detection: Detect and localize all objects in an image.

Multiple objects with different categories are in the input image.

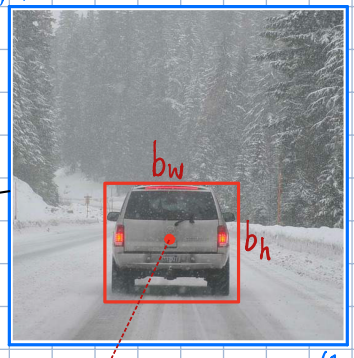
Classification with Localization





Label:
1. Pedestrian
2. Car
3. Motorcycle
4. Background / None of the above

E.g.: If this is an input image X,



Let's define the target label y.

$$y = \begin{cases} P_c & \text{: Is there any object? Probability?} \\ b_x \\ b_y \\ b_h \\ b_w & \text{bounding box} \\ c_1 \\ c_2 \\ c_3 & \text{which object it is. Probability?} \end{cases}$$

then $y = \begin{bmatrix} 1 \\ 0.5 \\ 0.7 \\ 0.4 \\ 0.3 \\ 0 \\ 1 \\ 0 \end{bmatrix} \begin{matrix} \leftarrow y_1 \\ \leftarrow y_2 \\ \\ \\ \\ \\ \\ \leftarrow y_8 \end{matrix}$

E.g.: If there is only background (no objects) in the input image X, then $y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$ → no objects

?: This value doesn't matter. I don't need to care about it.

And the loss function $\mathcal{L}(\hat{y}, y)$:

if $y_1 = 1$, $\mathcal{L}(\hat{y}, y) = (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \cdots (\hat{y}_8 - y_8)^2$

if $y_1 = 0$, $\mathcal{L}(\hat{y}, y) = (\hat{y}_1 - y_1)^2$

Note: The squared error is used in $\mathcal{L}(\hat{y}, y)$ for the sake of simplicity.

In practice, the following should be used.

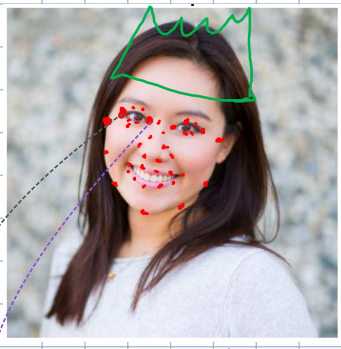
$$\begin{cases} P_c & \text{: logistic regression loss} \\ b_x \\ b_y \\ b_h \\ b_w & \text{squared error} \\ c_1 \\ c_2 \\ c_3 & \text{log likelyhood loss} \end{cases}$$

Landmark Detection

Landmark Detection: Neural networks can output coordinates of important points/landmarks in images.

4 landmarks

b_x
b_y
b_h
b_w



64 landmarks

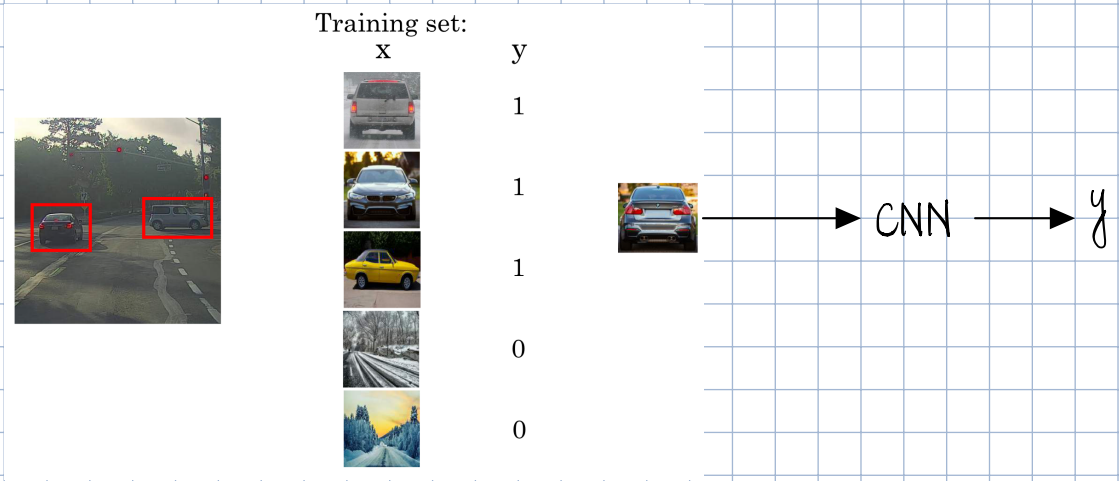
l_{1x}, l_{1y}
l_{2x}, l_{2y}
l_{64x}, l_{64y}

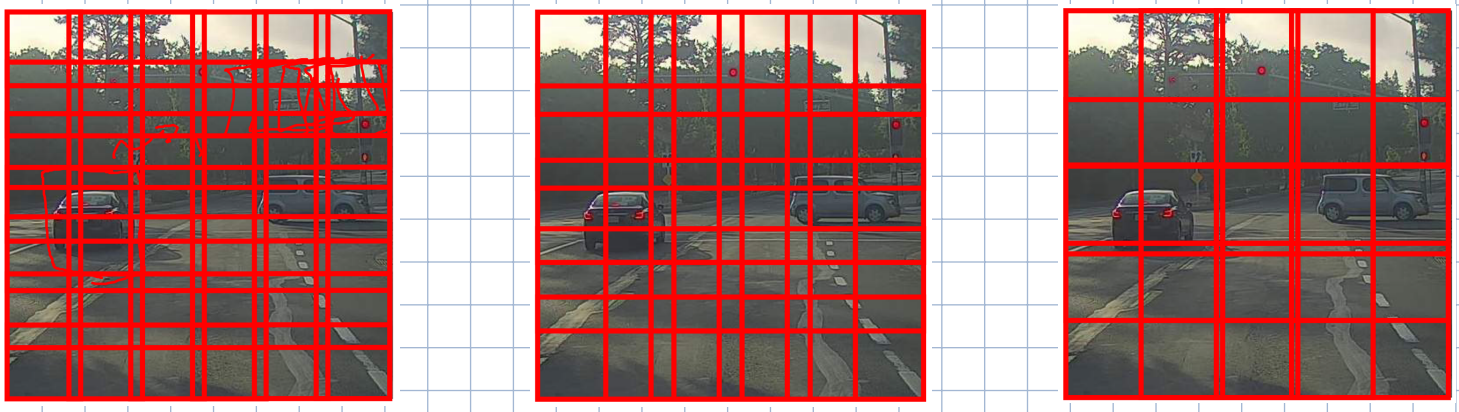
Human Pose Estimation

It turns out that landmarks don't have to be bounding box, they can be anything as long as they are meaningful features, such as, eyes, noses, chins, etc. Or positions of heads, hands, foot, etc.

Note that the identity (feature) of landmarks must be consistent among training images. E.g., if the first landmark in the first image is the left eye, then the first landmark in the second image must also be the left eye.

Object Detection using Sliding Window Detection





small medium big

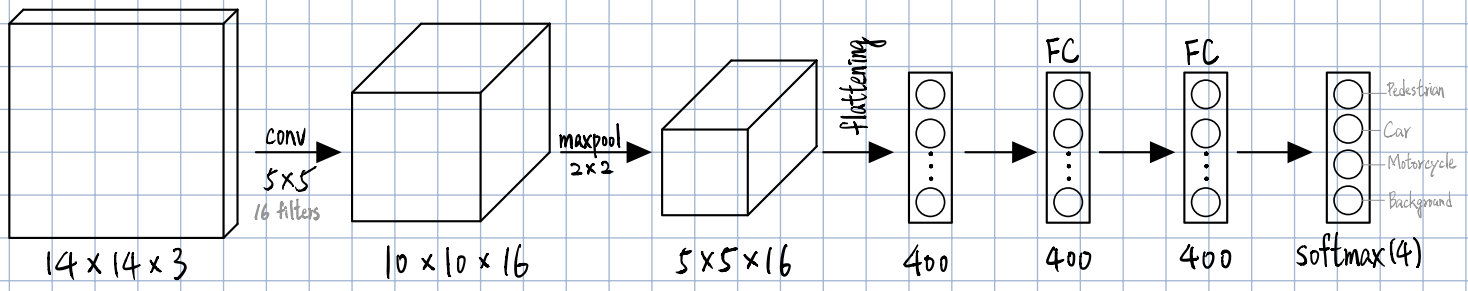
Using windows with different sizes and sliding windows with different strides.

The computation cost is too HIGH for CNN in inference.

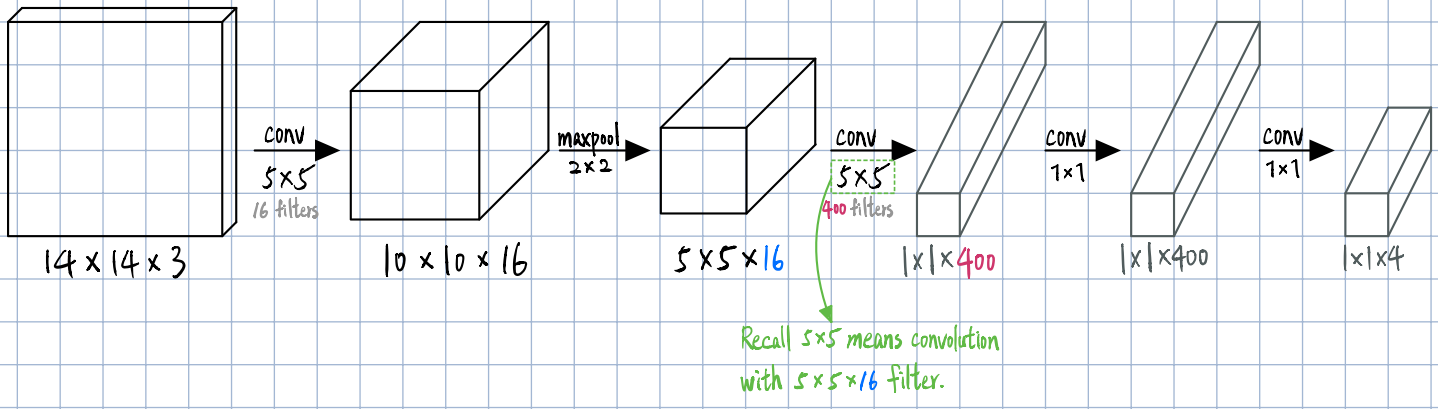
And the strides affect the accuracy.

Convolutional Implementation of Sliding Windows

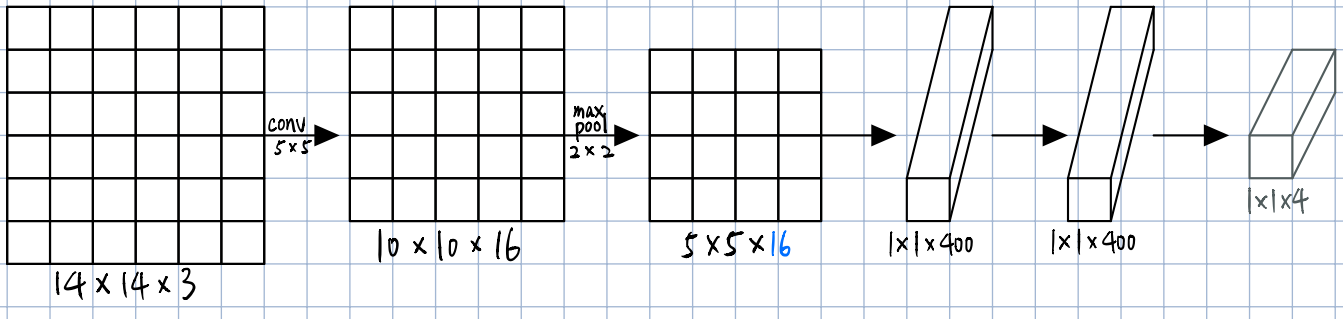
A normal CNN with Fully Connected (FC) layers.

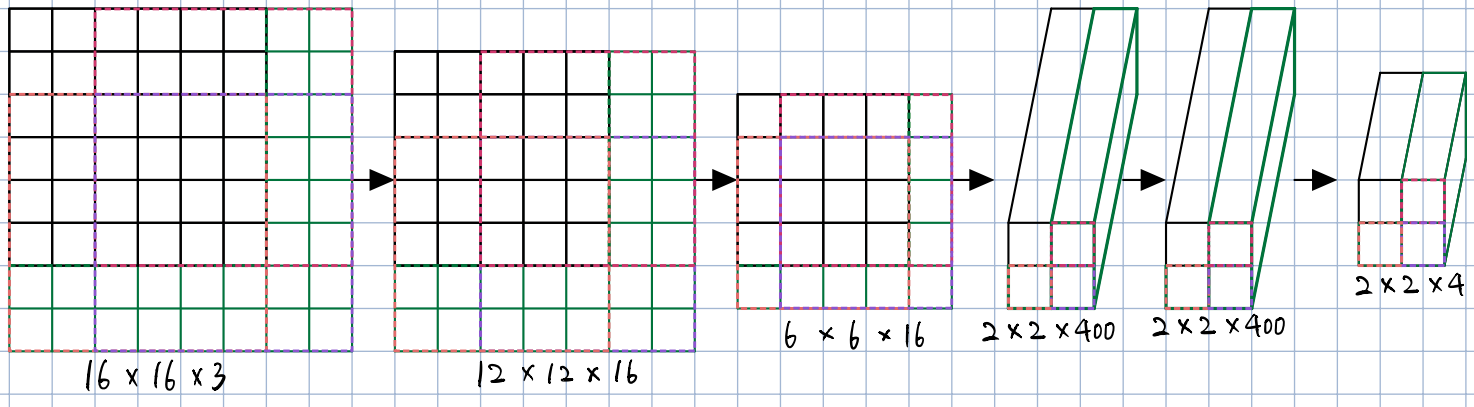


Turning FC Layer into Convolutional Layers:

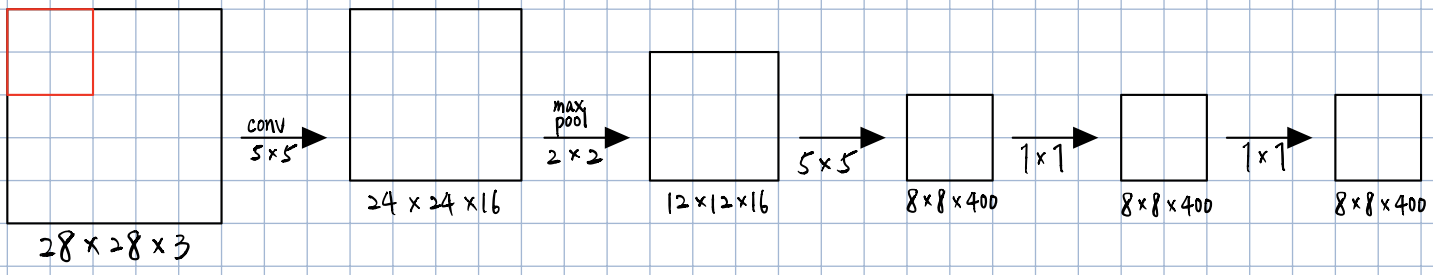


Convolutional Implementation of Sliding Windows





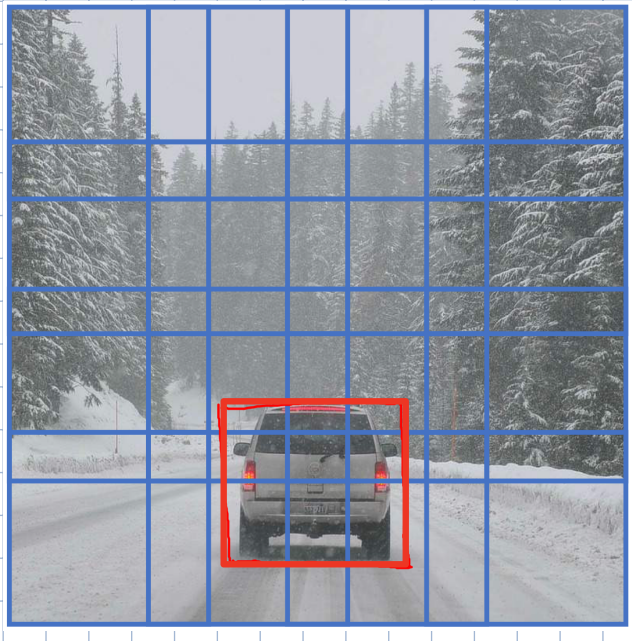
Another example: Given an $28 \times 28 \times 3$ image, use 14×14 filter and the stride 2 sliding window.



Read "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks"

More Accurate Bounding Box Predictions

Using convolutional implementation of sliding window can be computational efficient, but it still can not generate the most accurate bounding box.



The left image shows that

1: None of the boxes really match up perfectly with the position of the car.

2: The perfect bounding box is not even square, but all sliding windows are square.

Andrew says he had a hard time figuring out YOLO paper.

You Only Look Once (YOLO) Read "You Only Look Once: Unified, Real time Object Detection"

100

100

Two objects in an image.

Split into Grids →

For the purpose of illustration, 3×3 grid is used. In practice, finer grids are used. Eg: 19×19

Only one object in a grid cell.

The basic idea is that applying the image classification and localization algorithm to each of the nine grid cells in this image. (Convolutional Sliding Window)

Labels for training

For each of the nine grid cells:

$$y = \begin{bmatrix} P_c & \text{: Is there any object?} \\ b_x & \\ b_y & \text{bounding box} \\ b_h & \\ b_w & \\ c_1 & \\ c_2 & \text{which object it is.} \\ c_3 & \end{bmatrix}$$

Eg:

The 1st cell:

$$y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$

The 4th cell:

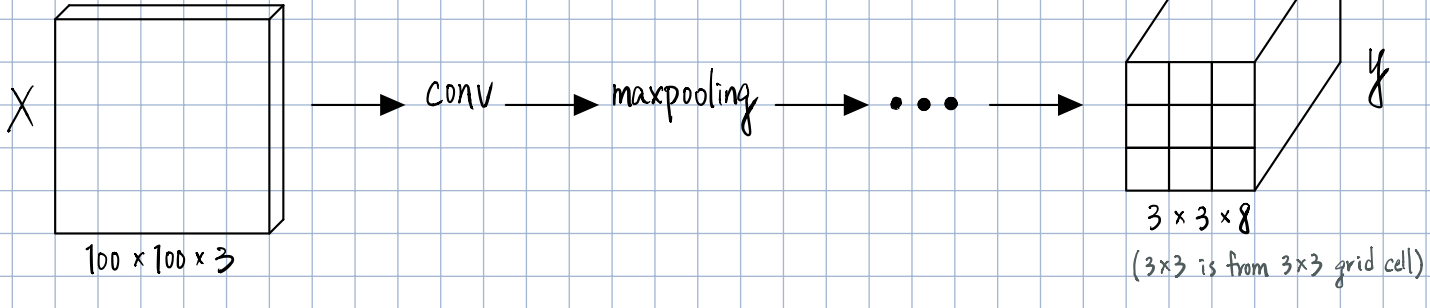
$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

The 6th cell:

$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

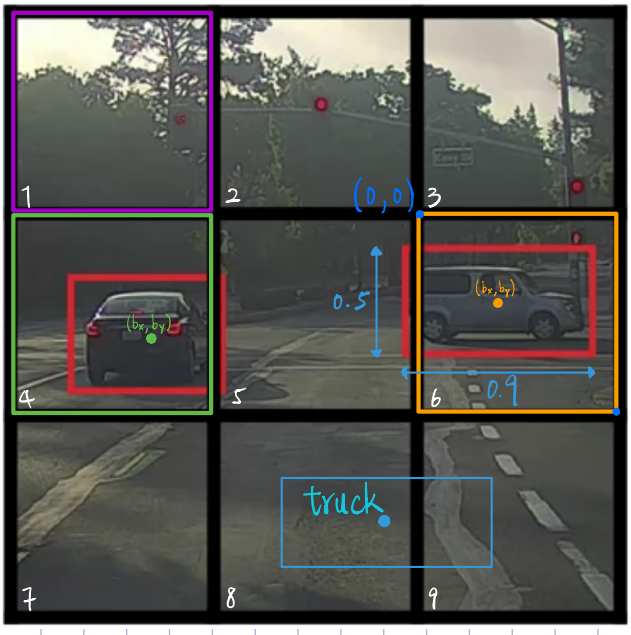
The target output is 3×3×8 matrix.
Each cell (1×1×8) is explained by the example on the right.

Thus, the flow of training is the following:



The YOLO method described above should be good enough as long as there is only one object in a grid cell. In practice, a finer grid cell should be used, which reduces the chance that there are multiple objects assigned to the same grid cell. An object is assigned to the grid cell which contains the midpoint of that object. In another word, each object, even an object spanning multiple grid cells, is assigned to only one grid cell.

Specifying the bounding box:



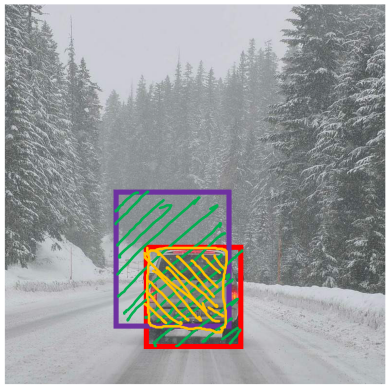
The 6th cell:

$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{matrix} \\ 0.4 \\ 0.3 \\ 0.5 \\ 0.9 \\ \\ \\ \end{matrix}$$

b_x, b_y: Must be between 0 and 1

b_h, b_w: Could be greater than 1.

Intersection Over Union (IoU)



Intersection Over Union (IoU)

$$= \frac{\text{size of } \square}{\text{size of } \square} \qquad 0 < IoU < 1$$

The covention: "Correct" if $IoU \geq 0.9$

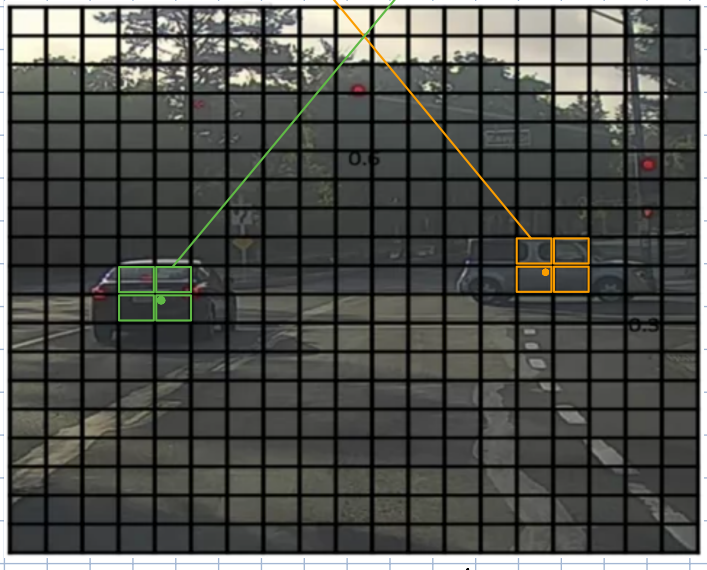
Detect an Object only Once with Non-Max Suppression

Goal: Detect two cars.

100

100

First, the image is splitted into 19×19 grid cells. Now, the problem is that many cells would claim they detect cars. but there is only one car. Eg, four cells here and here all think they detect a car.

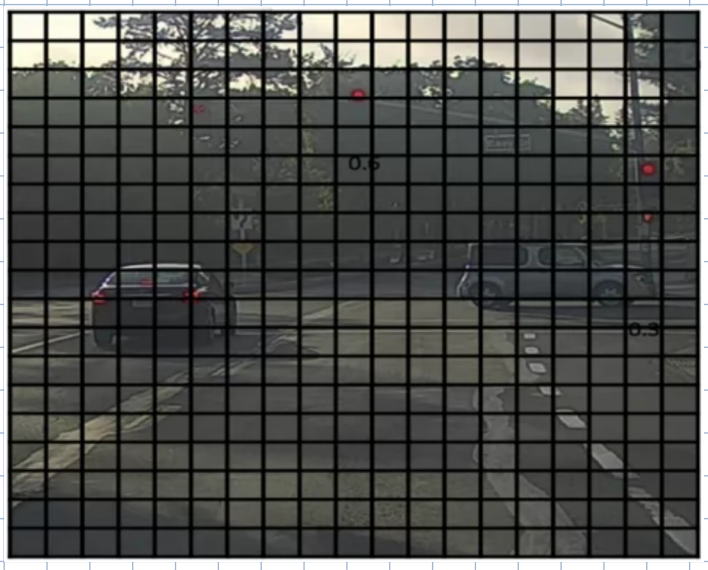


19 × 19 grids

Non-Max Suppression algorithm

For the sake of simplicity, each output prediction is: $\begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$ P_c: Probability of an object presented; b_x, b_y, b_h, b_w: bounding box

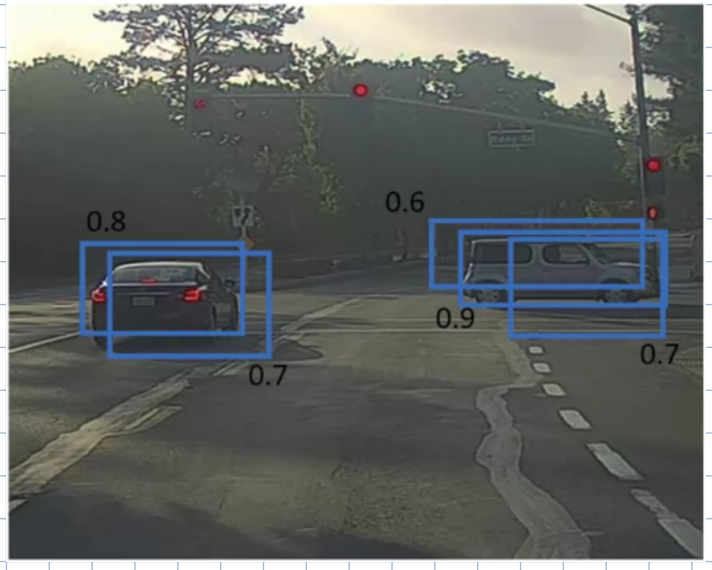
19 × 19 grid cells



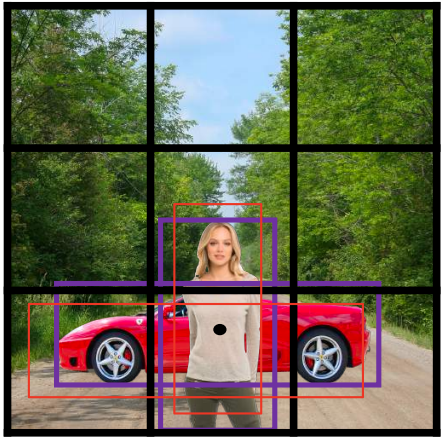
Step 1: Discard all bounding boxes with $P_c \leq 0.6$

Step 2: While there are any remaining boxes:

- Pick the box with the largest P_c and output that as a prediction.
- Discard any remaining box with IoU ≥ 0.5 with the box output in the previous step.



Overlapping objects: Anchor Boxes Algorithm

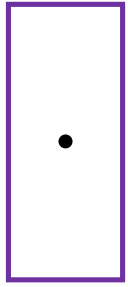


$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

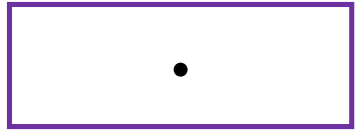
Previously: Each object in training image is assigned to grid cell that contains that object's midpoint.

Output y: $3 \times 3 \times 8$

Anchor box 1:



Anchor box 2:



$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

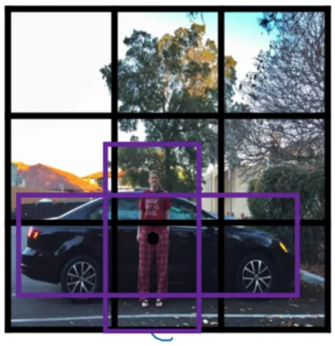
Now, with two Anchor Boxes: Each object in training image is assigned to grid cell that contains object's midpoint and anchor box for the grid cell with highest IoU.

Output y: $3 \times 3 \times (2 \times 8) = 3 \times 3 \times 16$

Two Anchors

[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

Anchor box example



Anchor box 1: Anchor box 2:

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

The training label y

		With human and car	With car
Anchor Box 1	P_c	1	0
	b_x	b_x	?
	b_y	b_y	?
	b_h	b_h	?
	b_w	b_w	?
	c_1	1	?
	c_2	0	?
	c_3	0	?
Anchor Box 2	P_c	1	1
	b_x	b_x	b_x
	b_y	b_y	b_y
	b_h	b_h	b_h
	b_w	b_w	b_w
	c_1	0	0
	c_2	1	1
	c_3	0	0

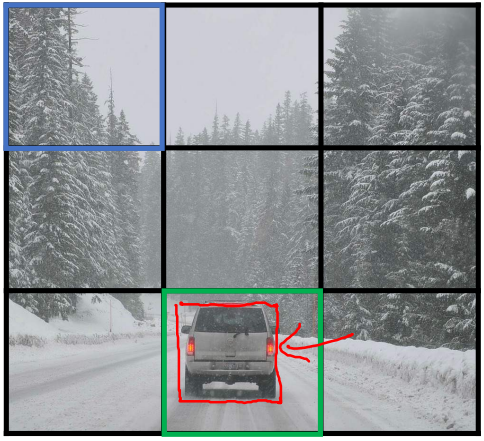
deeplearning.ai

Object Detection

Putting it together: YOLO algorithm

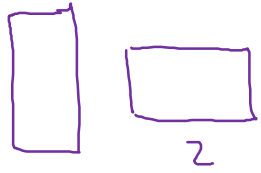
Training

1 - pedestrian
2 - car
3 - motorcycle



$y =$

$$\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \quad \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \quad \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$



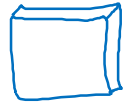
3 × 3 × 16

y is 3×3×2×8

19×19×16
19×19×40

#anchors

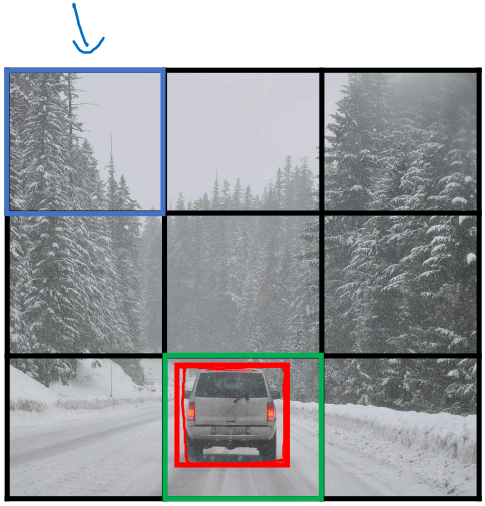
5 + #classes

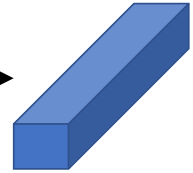


→ ConvNet →

[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

Making predictions

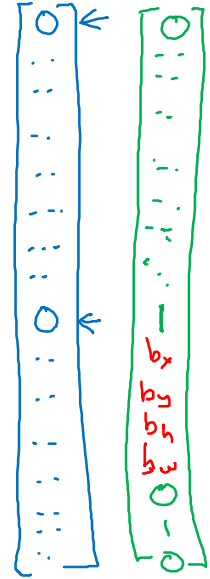




3×3×2×8

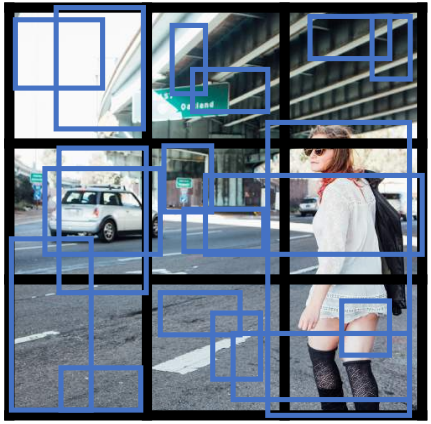
$y =$

$$\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$



Andrew Ng

Outputting the non-max supressed outputs



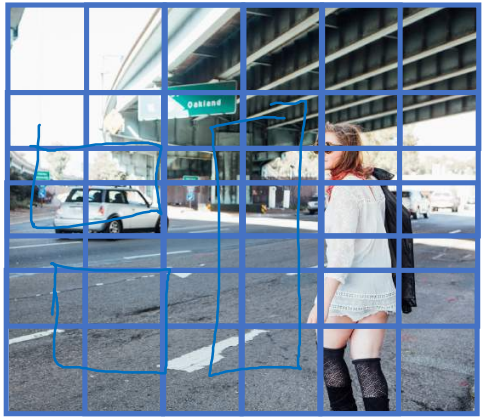
- For each grid ~~call~~ cell, get 2 predicted bounding boxes.
- Get rid of low probability predictions.
- For each class (pedestrian, car, motorcycle) use non-max suppression to generate final predictions.

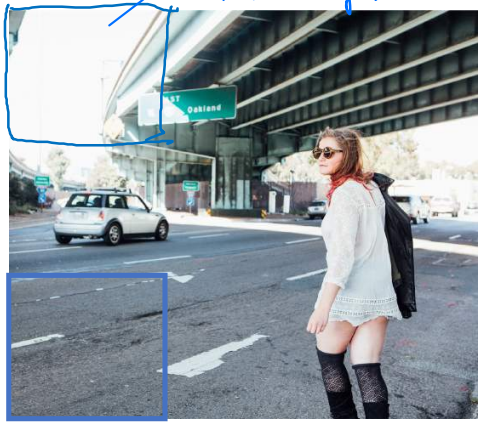
deeplearning.ai

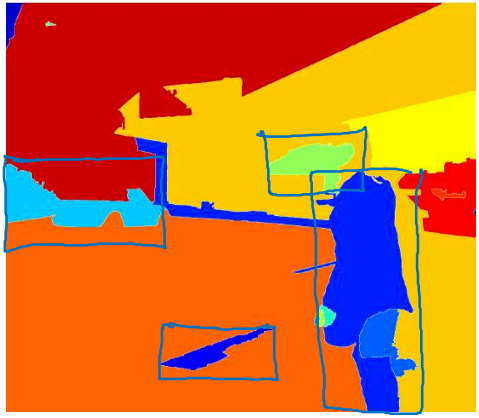
Object Detection

Region proposals (Optional)

Region proposal: R-CNN







Segmentation algorithm

~2,000

Running CNN over the whole image is kind of wasteful of computation resource. We can use Segmentation to find out interesting regions first, then run CNN over interesting regions. I feel this Region-based CNN is not very robust.

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

Faster algorithms

R-CNN: Propose regions. Classify proposed regions one at a time. Output label + bounding box.

Fast R-CNN: Propose regions. Use convolution implementation of sliding windows to classify all the proposed regions.

Faster R-CNN: Use convolutional network to propose regions.

[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]

deeplearning.ai

Convolutional Neural Networks

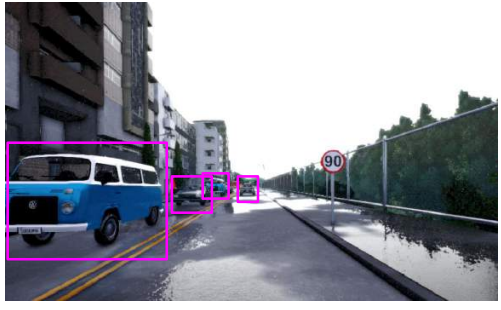
Semantic segmentation with U-Net

Object Detection vs. Semantic Segmentation

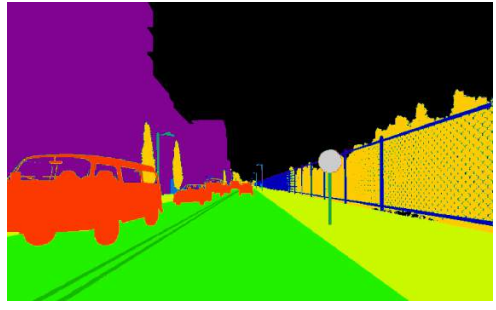
Find bounding box

To know the class of every pixel

Input image

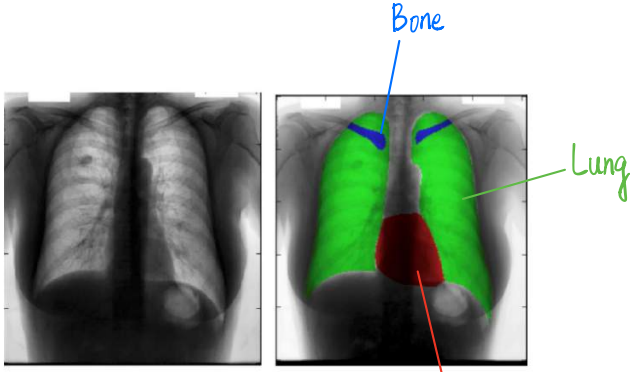


Object Detection

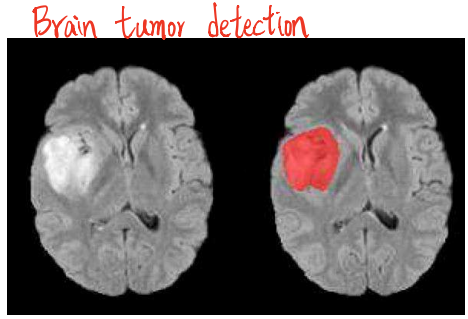


Semantic Segmentation

Motivation for U-Net



Chest X-Ray



Brain MRI

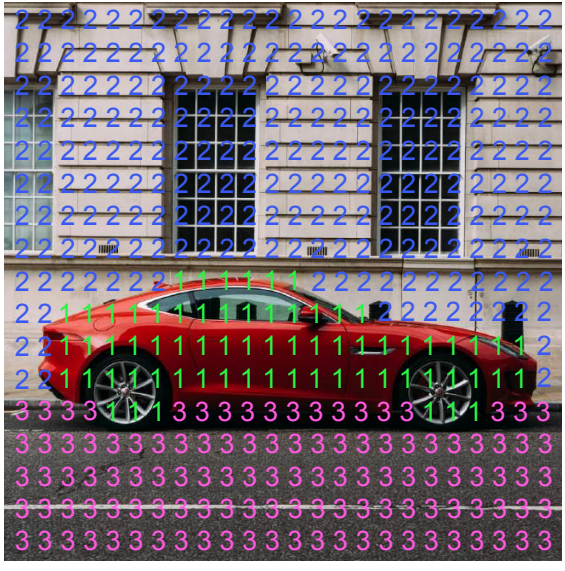
[Novikov et al., 2017, Fully Convolutional Architectures for Multi-Class Segmentation in Chest Radiographs]
[Dong et al., 2017, Automatic Brain Tumor Detection and Segmentation Using U-Net Based Fully Convolutional Networks]

Per-pixel class labels

1. Car
0. Not Car

Per-pixel class labels

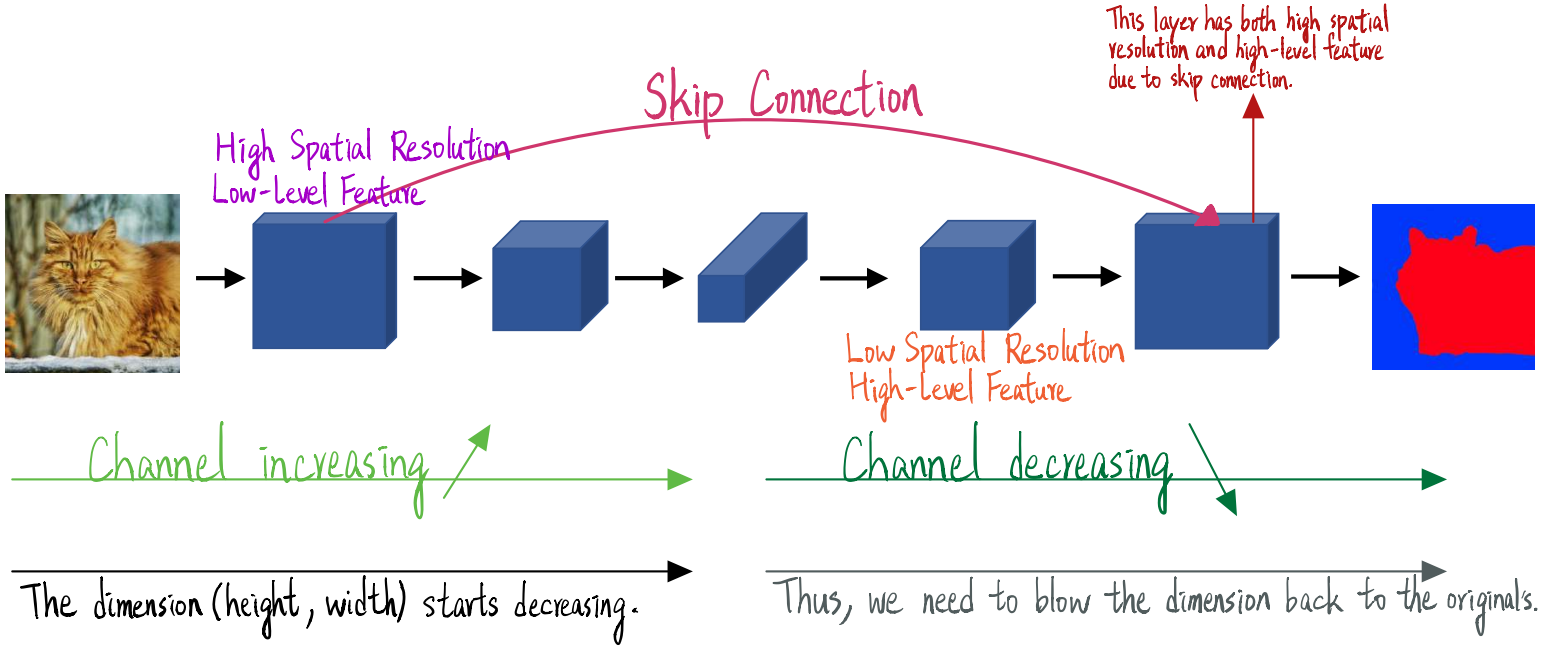
This is quite a huge output matrix which consumes lots of memory.



1. Car
2. Building
3. Road

Segmentation Map

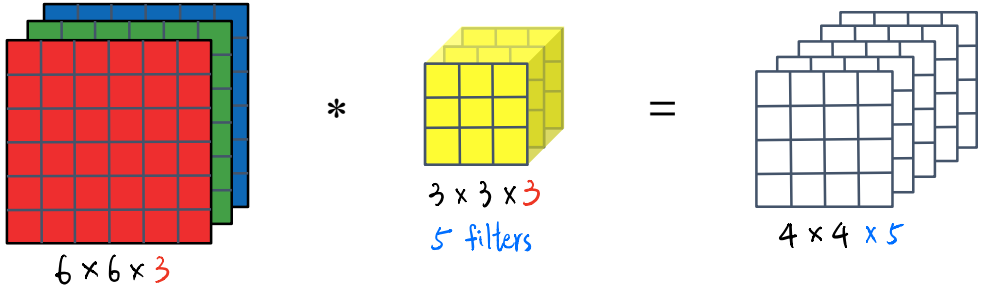
Deep Learning for Semantic Segmentation



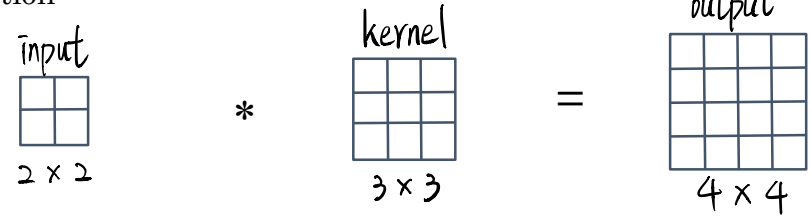
Andrew Ng

Transpose Convolution

Normal Convolution



Transpose Convolution



Andrew Ng

Transposed Convolution

Input

2	1
3	2

2 × 2

*

Filter

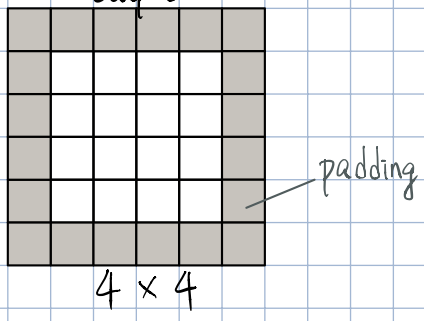
1	2	1
2	0	1
0	2	1

3 × 3

padding = 1

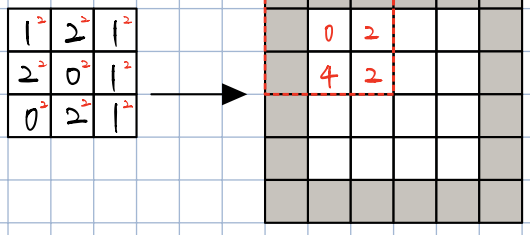
stride = 2

Output

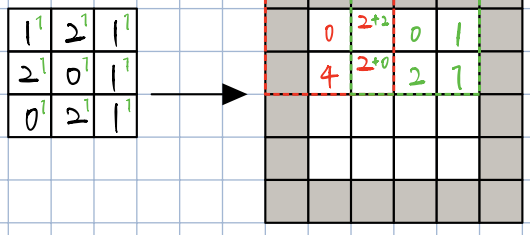


4 × 4

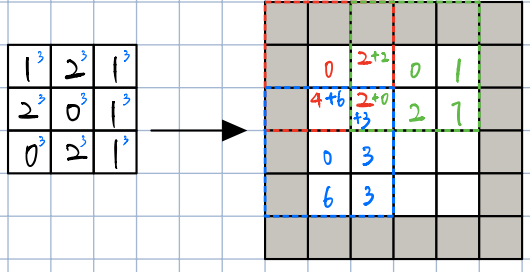
Step 1: The (1,1) input pixel:



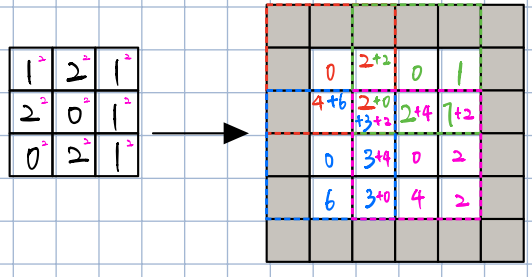
Step 2: The (1,2) input pixel:



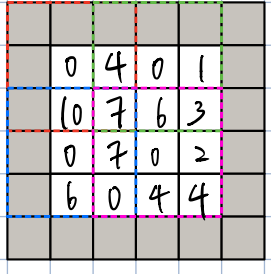
Step 3: The (2,1) input pixel:

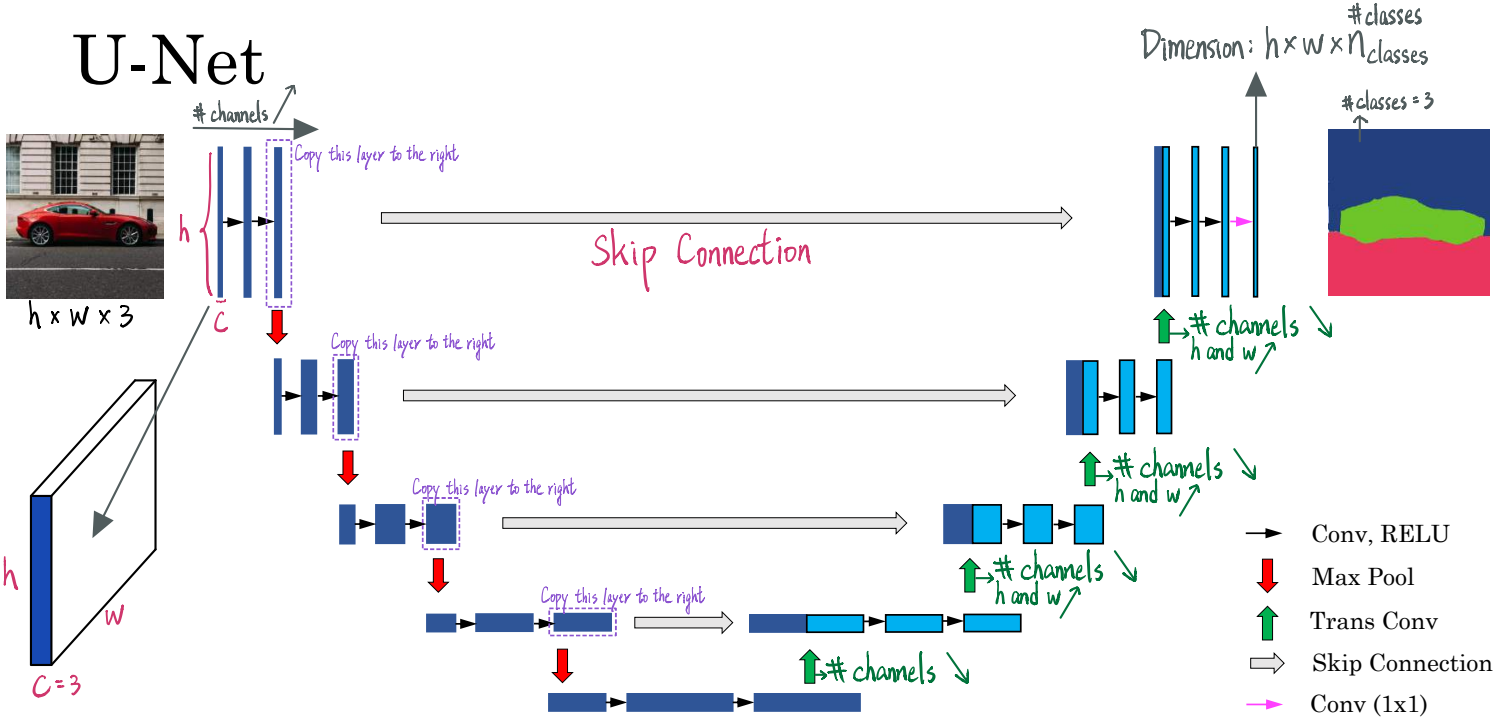


Step 3: The (2,2) input pixel:



Step 5: Sum up





[Ronneberger et al., 2015, U-Net: Convolutional Networks for Biomedical Image Segmentation]