DeepLearning.ai
https://www.coursera.org/specializations/deep-learning
course 4

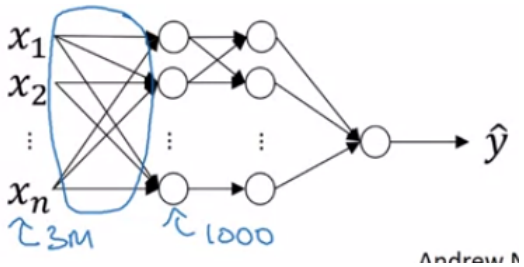
Bassalayev Maxim

Designations

- $m = m_{train}$−#train examples
- m_{test}− #test examples
- $n^{[l]}$−number of hidden units on layer l
- $g^{[l]}$− activation function on layer l
- $w^{[l]}, b^{[l]}$ - parameters on layer l
- $dW^{[l]}, dB^{[l]} \dots dA^{[l]}$ - derivatives on layer l
- $par^{[l](i)}$− the value of example i parameter on layer l
- α− learning rate
- $f^{[l]}$− filter size on layer l.
- $p^{[l]}$− padding on layer l.
- $s^{[l]}$− stride on layer l.
- $n_C^{[l]}$− number of filters on layer l.

Deep Learning on large images

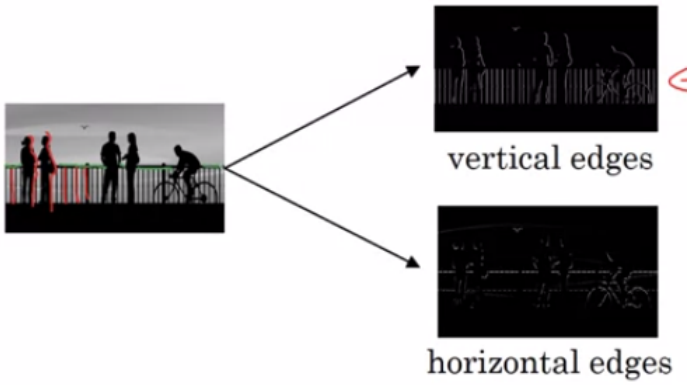
Let input X - cat picture $\underbrace{1000}_{\text{width}} \times \underbrace{1000}_{\text{height}} \times \underbrace{3}_{\text{RGB}} = 3.000.000$
Imagine that the first layer of the neural network contains 1000 hidden units.
$n^{[1]} = 1000, n^{[0]} = 3.000.000$



NN

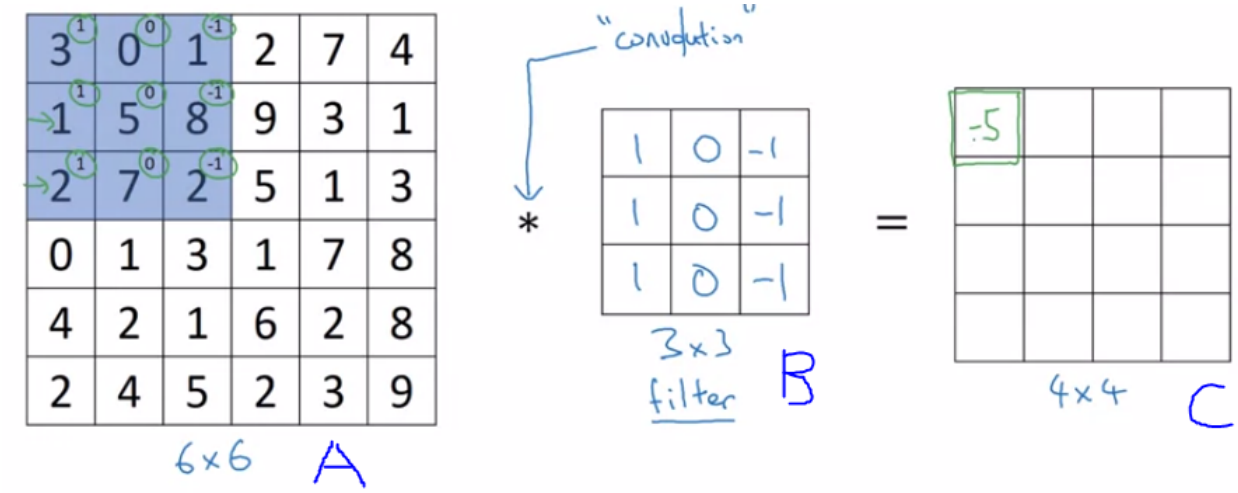
So, $w^{[1]}$ shape will be $(1000, 3.000.000)$. Totally 3 billion parameters. It's difficult to get enough data to prevent a neural network from overfitting.

Edge detection



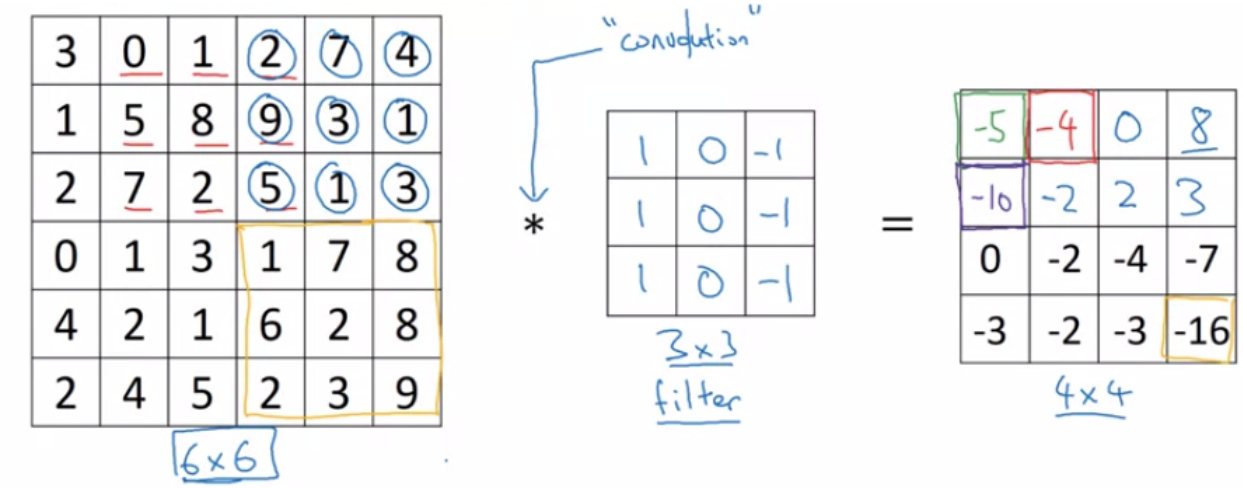
Vertical and horizontal edges

We want to detect vertical or horizontal edges using matrices.
Convolution operation :



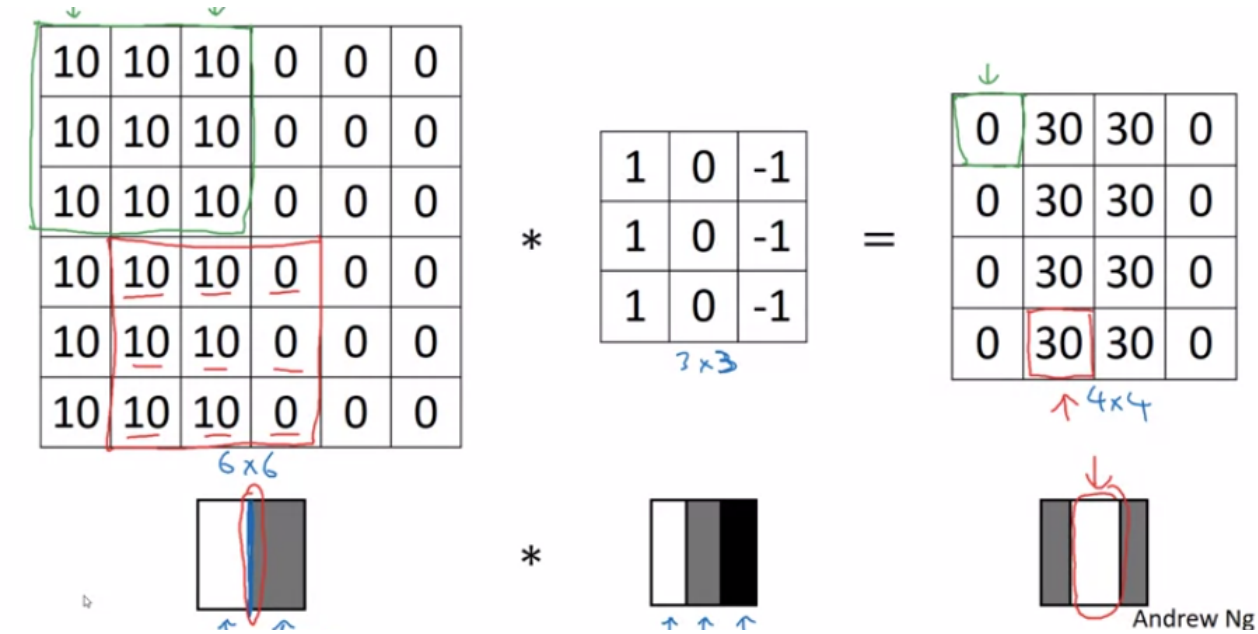
Convolution operation (1)

$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{12} + A_{13} \cdot B_{13} + A_{21} \cdot B_{21} + \ldots + A_{33} \cdot B_{33} = -5$
$\ldots$
$C_{ij} = A_{ij} \cdot B_{11} + A_{ij+1} \cdot B_{12} + A_{ij+2} \cdot B_{13} + A_{i+1j} \cdot B_{21} + \ldots + A_{i+2j+2} \cdot B_{33}$



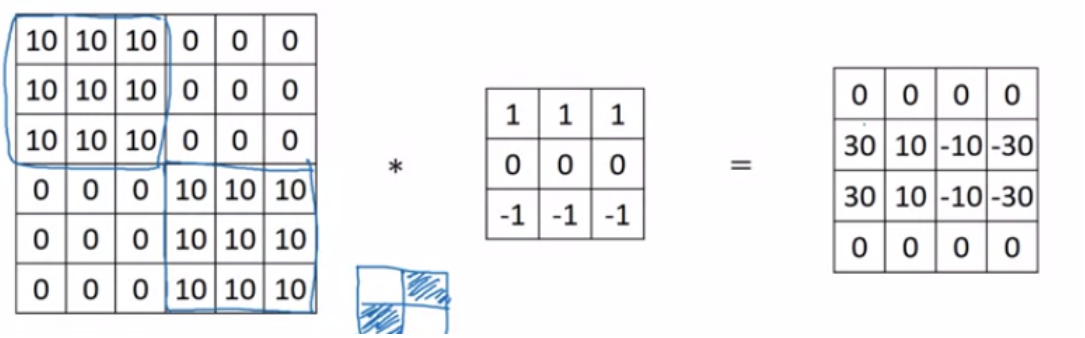
Convolution operation (2)

Vertical edge detection : Use convolution operation with the correct filter.



Vertical edge detection

Horizontal edge detection : Use convolution operation with the correct filter.



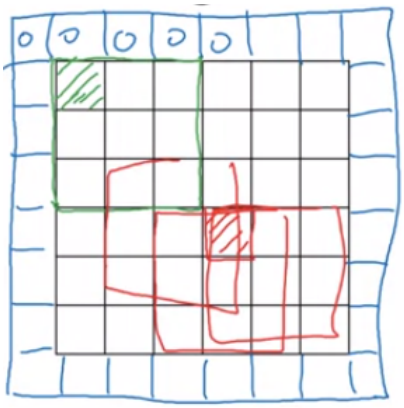
Horizontal edge detection

Filters :

$\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$ – Solber filter. $\begin{pmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{pmatrix}$ – Scharr filter.

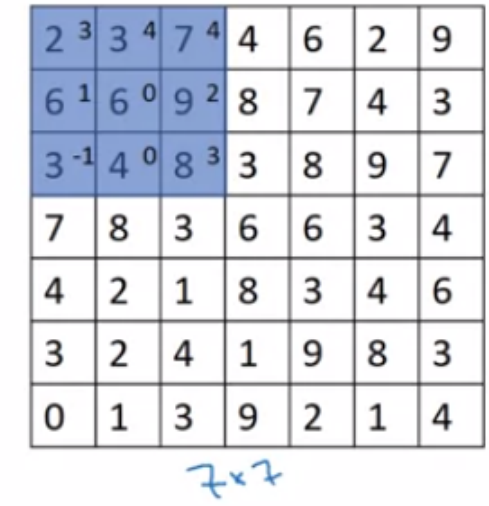
Convolution operation

Padding : Padding - additional border of pixels all around the edges.

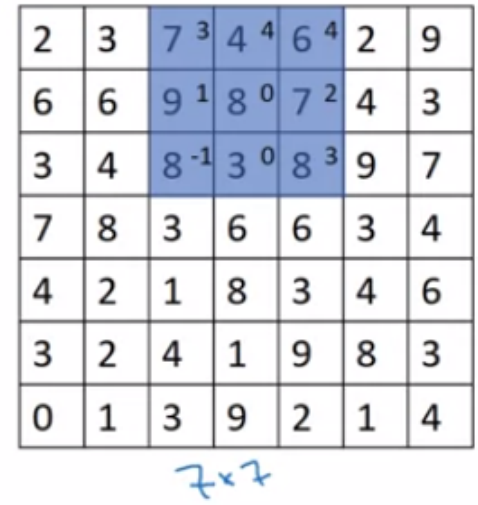


Pad the image with an additional one border

Input size $6 \times 6 \to 8 \times 8$.
Let, input size $n \times n$, padding size $= p$ and filter size $f \times f$.
So output size will be $n + 2p - f + 1 \times n + 2p - f + 1$
Valid convolutions :
$(n \times n) * (f \times f) \to (n - f + 1) \times (n - f + 1)$
No padding.
Same convolution :
$(n + 2p - f + 1) \times (n + 2p - f + 1)$
Pad so the output size is the same the input size.
$n + 2p - f + 1 = n \to p = \dfrac{f - 1}{2}$
Filter size is usually odd.
Strided Convolutions : Instead of sliding the window by 1, we will slide it by s positions right or down.



Strided Convolutions with s = 2 (1)



Strided Convolutions with s = 2 (2)

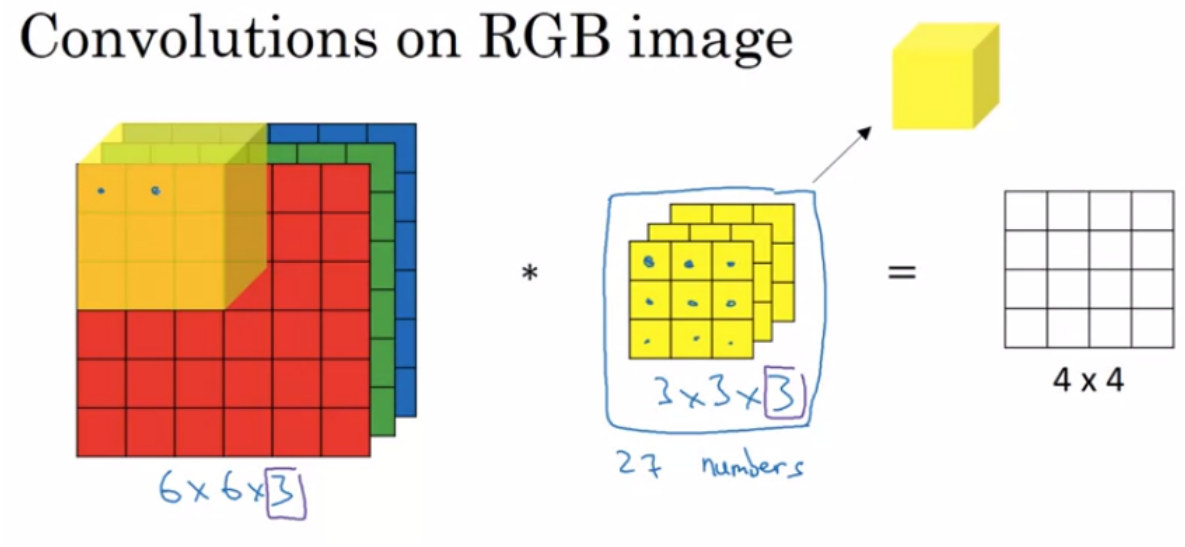
Summary of convolutions :

$$n \times n \text{ image } \ \mathrm{f} \times \mathrm{f} \text{ filter}$$

$$\text{padding } \mathrm{p} \ \ \text{stride } \mathrm{s}$$

$$\text{Output size } \left\lfloor \frac{\mathrm{n}+2\mathrm{p}-\mathrm{f}}{\mathrm{s}}+1) \right\rfloor \times \left\lfloor \frac{\mathrm{n}+2\mathrm{p}-\mathrm{f}}{\mathrm{s}}+1) \right\rfloor$$

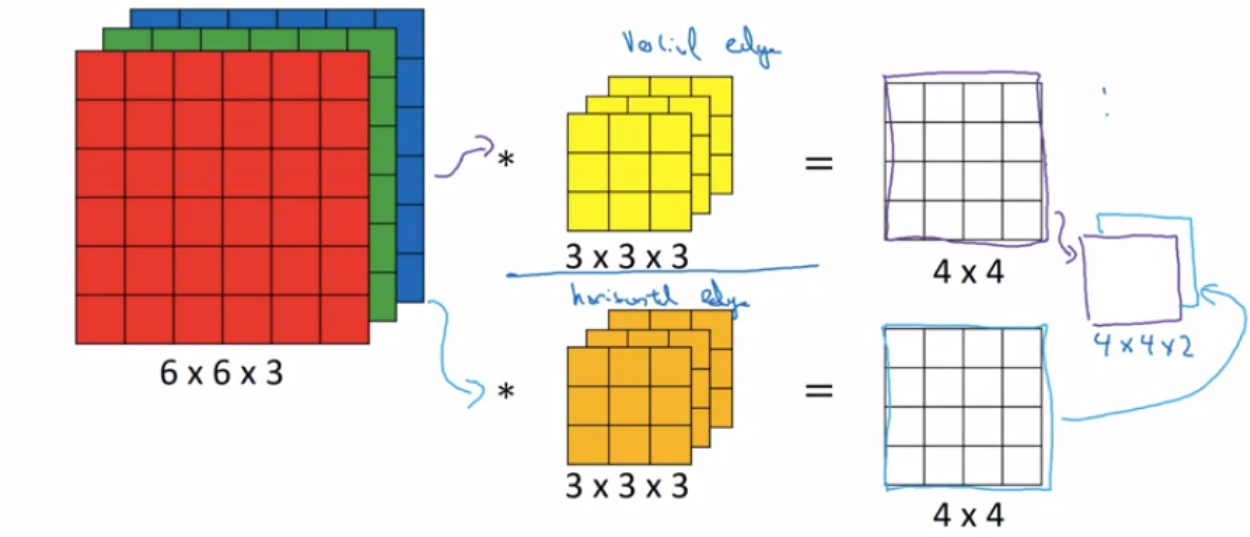
Convolutions Over Volume



Convolutions on RGB image

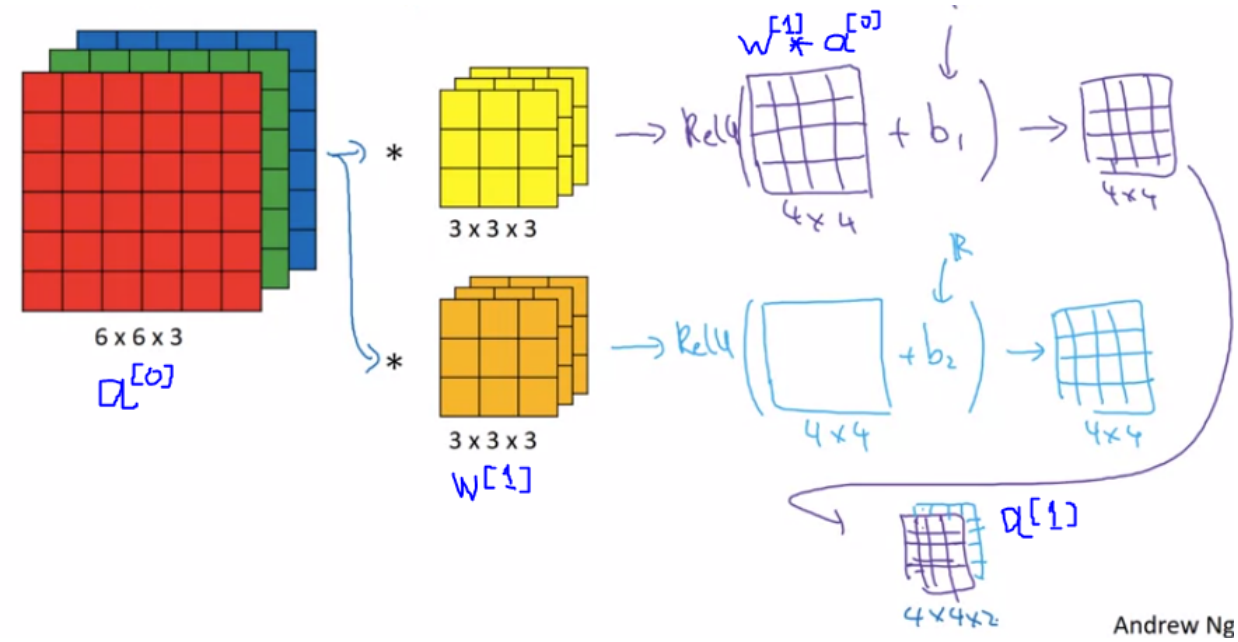
Input size = $\underbrace{6}_{\text{height}} \times \underbrace{6}_{\text{width}} \times \underbrace{3}_{\#\text{channels}}$

Multiple filters :



Convolutions with 2 filters

One Layer of a Convolutional Network

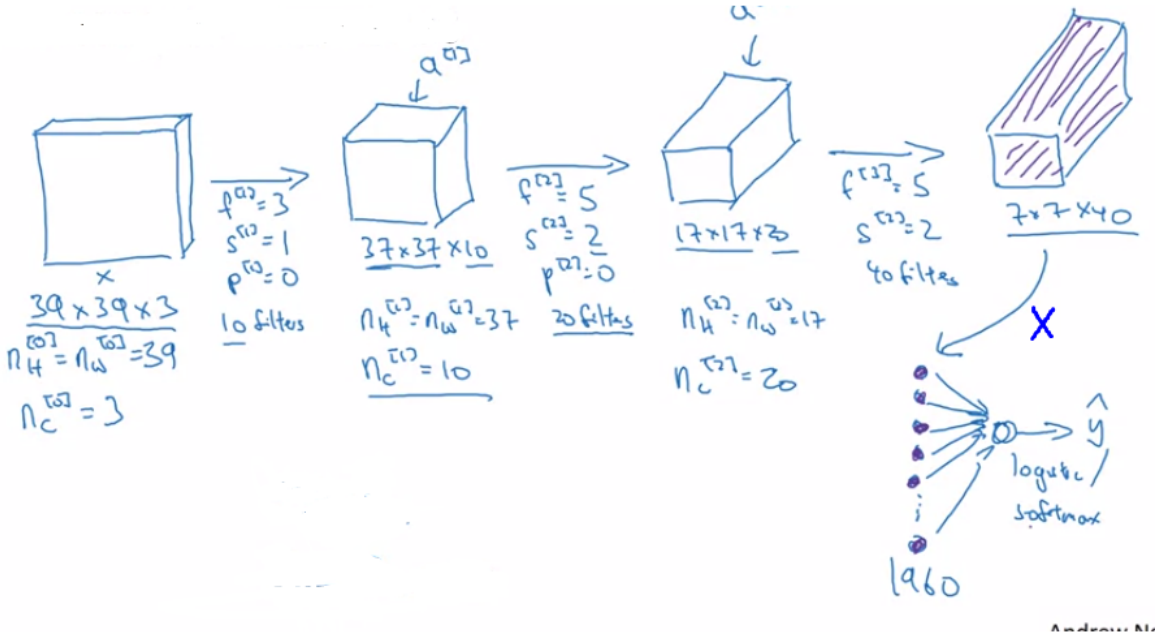


One Layer of a Convolutional Network

If layer l is a convolutional layer.

- $f^{[l]}$– filter size on layer l.
- $p^{[l]}$– padding on layer l.
- $s^{[l]}$– stride on layer l.
- $n_C^{[l]}$– number of filters on layer l.
- Each filter is : $f^{[l]} \times f^{[l]} \times n_C^{[l-1]}$.
- Activations : $a^{[l]} \to n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$.
- Weight : $f^{[l]} \times f^{[l]} \times n_C^{[l-1]} \times n_C^{[l]}$
- Bias : $1 \times 1 \times 1 \times n_C^{[L]}$
- Input : $n_H^{[l-1]} \times n_W^{[l-1]} \times n_C^{[l-1]}$
- Output : $n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$
- $n_H^{[l]} = \left\lfloor \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$
- $n_W^{[l]} = \left\lfloor \frac{n_W^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$
- $A^{[l]} \to m \times n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$

Simple Convolutional Network

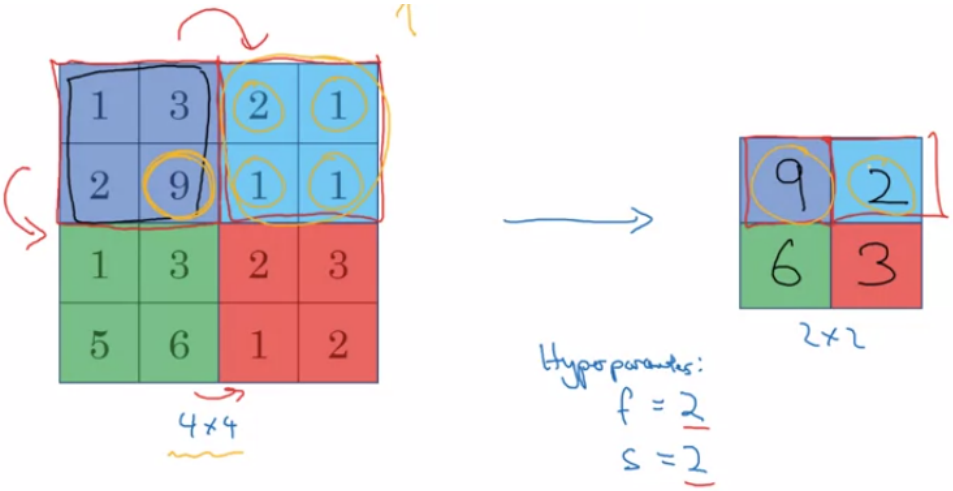


Simple Convolutional Network Example

At the X step we unrolling $7 \times 7 \times 40(1960)$ numbers into a very long vector. After that, we can use softmax/logistic regression.

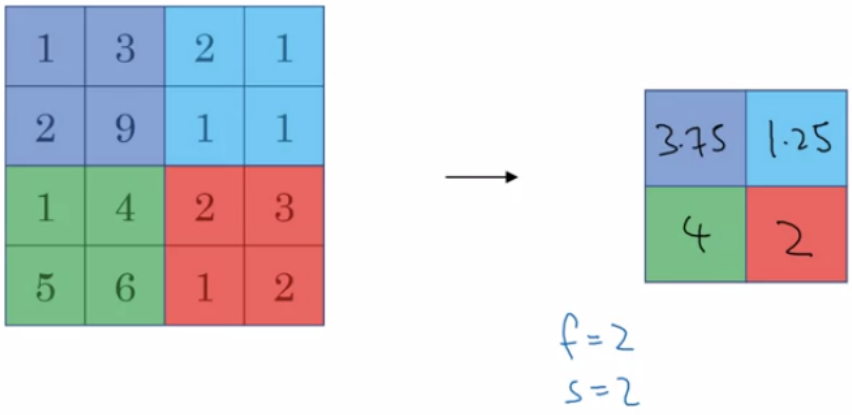
Pooling layers

Max pooling :



Max pooling

Average pooling :



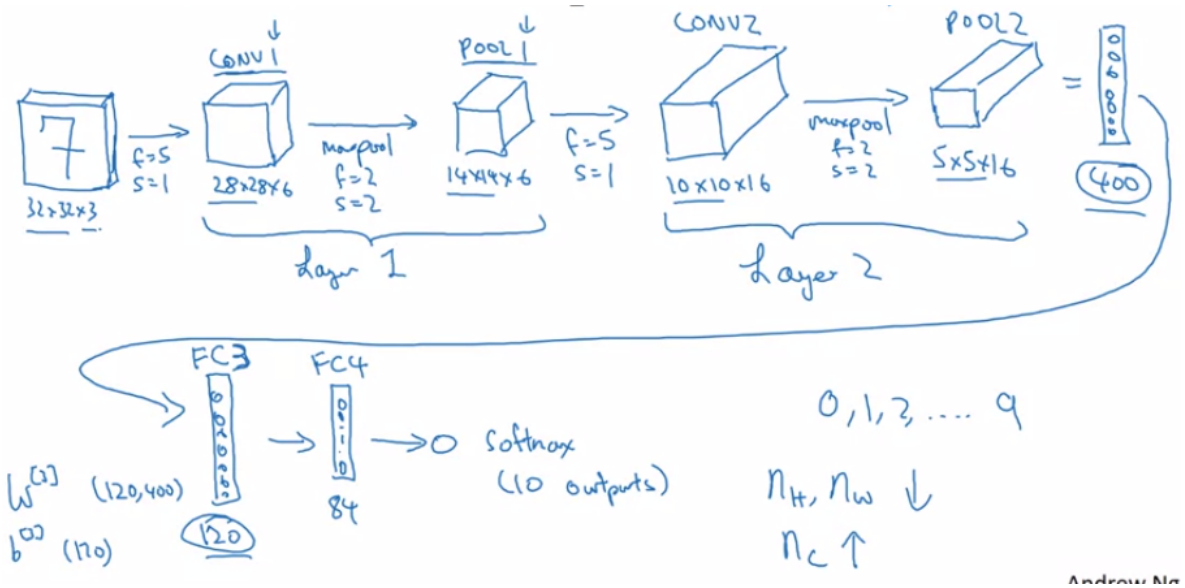
Average pooling

Hyperparameters :

- f : filter size
- s : stride
- p : padding
- Max or average pooling

No parameters to learn!

CNN

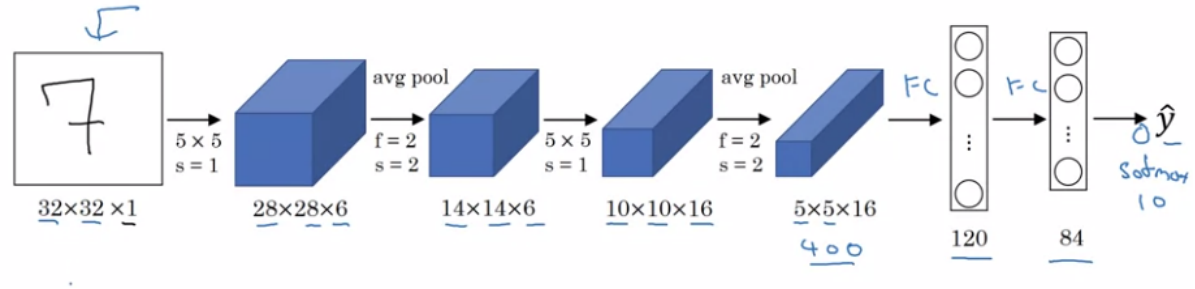


CNN example

	Activation shape	Activation size	# parameters
Input	(32, 32, 3)	3072	9
CONV1 (f = 5, s = 1)	(28, 28, 8)	6272	608
POOL1	(14, 14, 8)	1568	0
CONV2 (f = 5, s = 1)	(10, 10, 16)	1600	3216
POOL2	(5, 5, 16)	400	0
FC3	(120, 1)	120	0
FC3	(120, 1)	120	48120
FC4	(84, 1)	84	10164
Softmax	(10, 1)	10	850

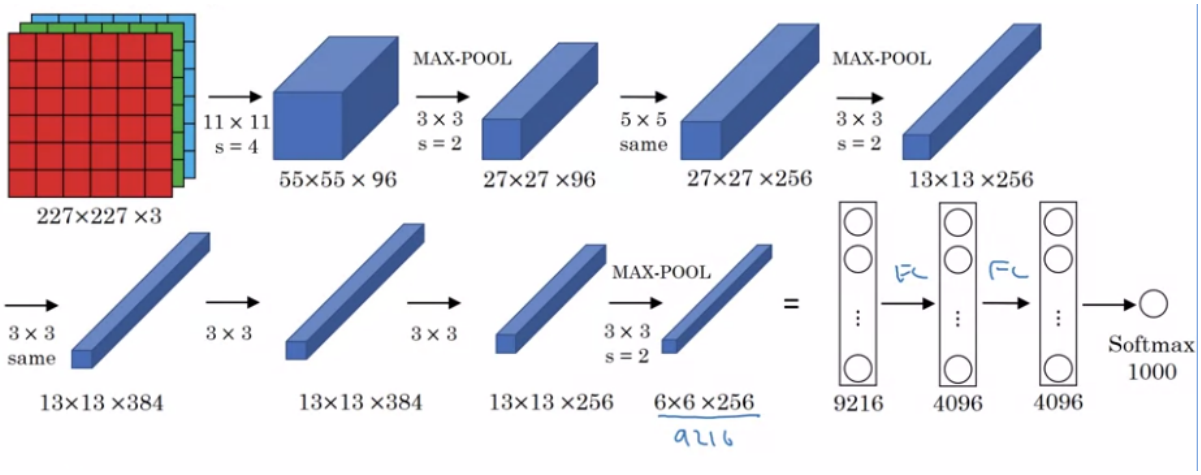
Classic networks

LeNet - 5 :



LeNet - 5 example

$\approx$ 60000 parameters
$n_H, n_W \downarrow$ $n_C \uparrow$
Input $\rightarrow$ CONV $\rightarrow$ POOL $\rightarrow$ CONV $\rightarrow$ POOL $\rightarrow$ FC $\rightarrow$ FC $\rightarrow$ Output
http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf
AlexNet :

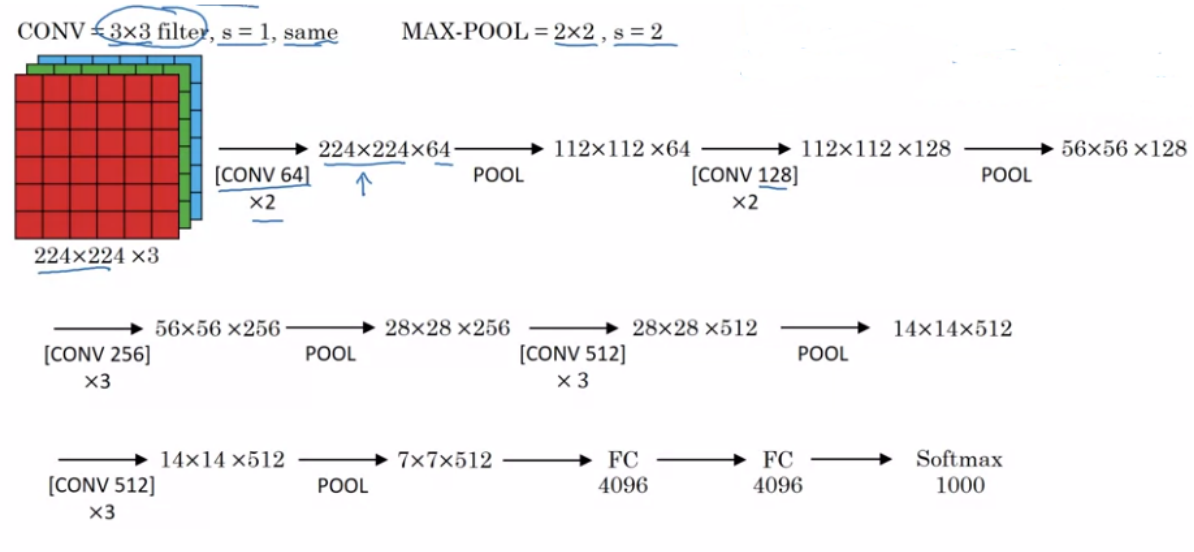


AlexNet example

$\approx 60M$ parameters
Similary to LeNet, but much bigger

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

VGG-16 :



VGG-16 example

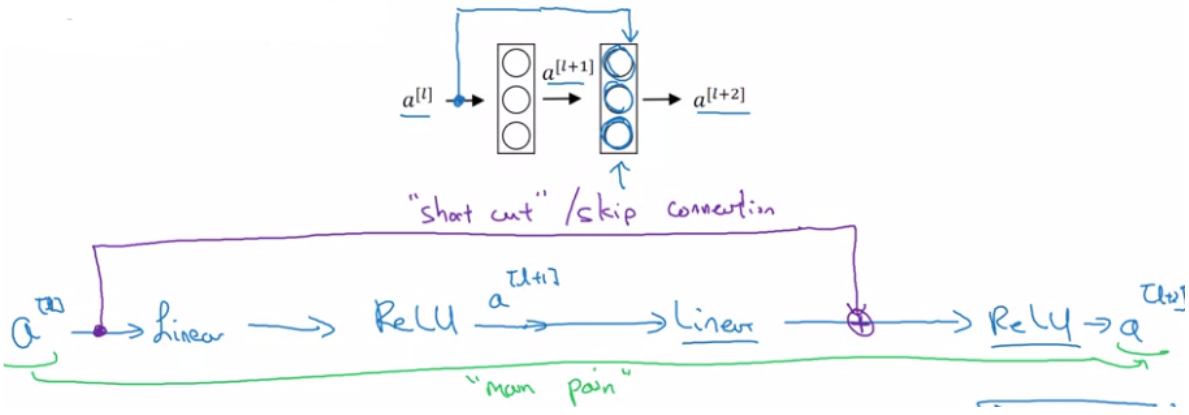
$\approx 138M$ parameters

https://arxiv.org/abs/1409.1556

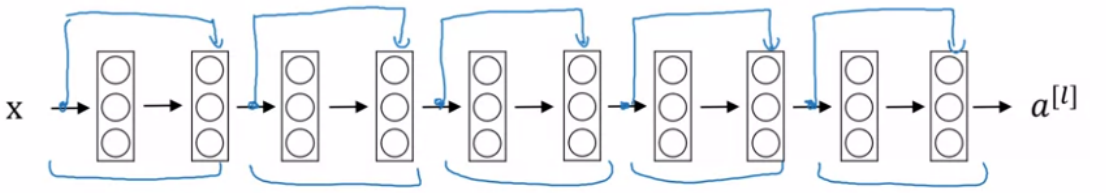
ResNet

Residual block :

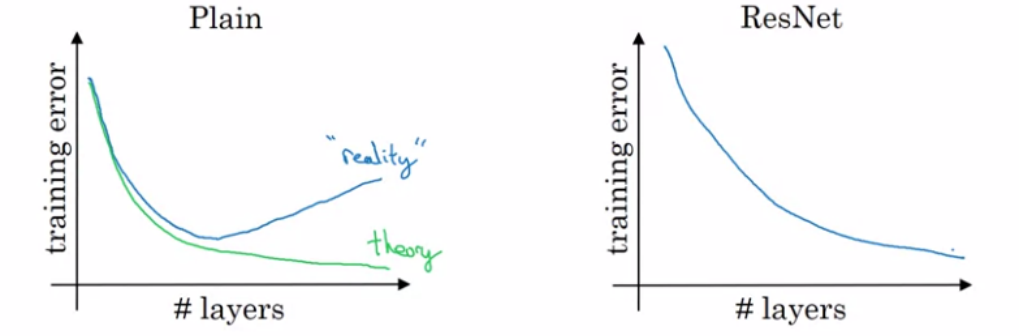
$$\text{Plain} \begin{cases} z^{[l+1]} = w^{[l+1]}a^{[l]} + b^{l+1} \\ a^{[l+1]} = g(z^{[l+1]}) \\ z^{[l+2]} = w^{[l+2]}a^{[l+1]} + b^{l+2} \\ a^{[l+2]} = g(z^{[l+2]}) \end{cases} \qquad \text{Residual} \begin{cases} a^{[l+2]} = g(z^{[l+2]} + a^{[l]}) \end{cases}$$



Residual block example



ResNet

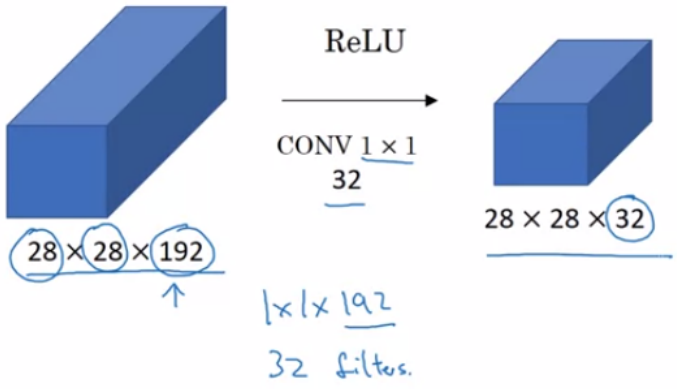


Training error in plain network and ResNet

https://arxiv.org/abs/1512.03385

Inception network

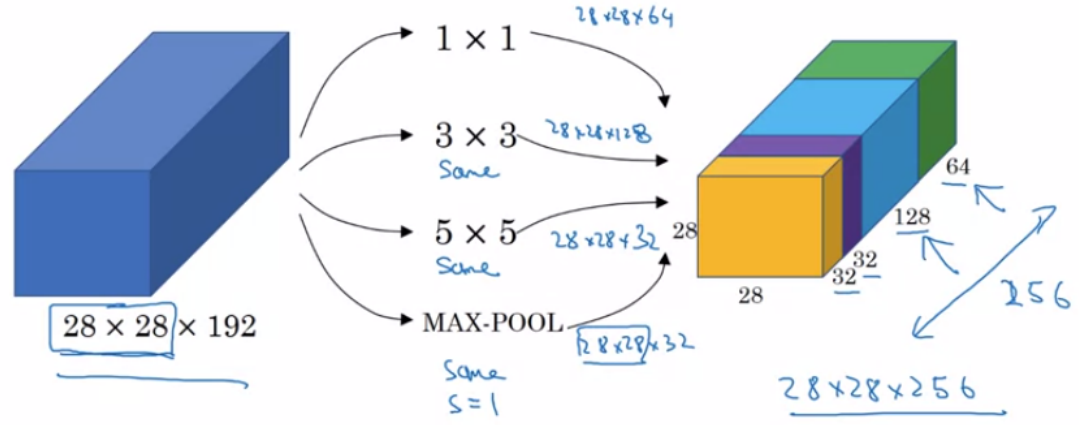
Using 1×1 convolutions :



Using 1×1 convolutions example

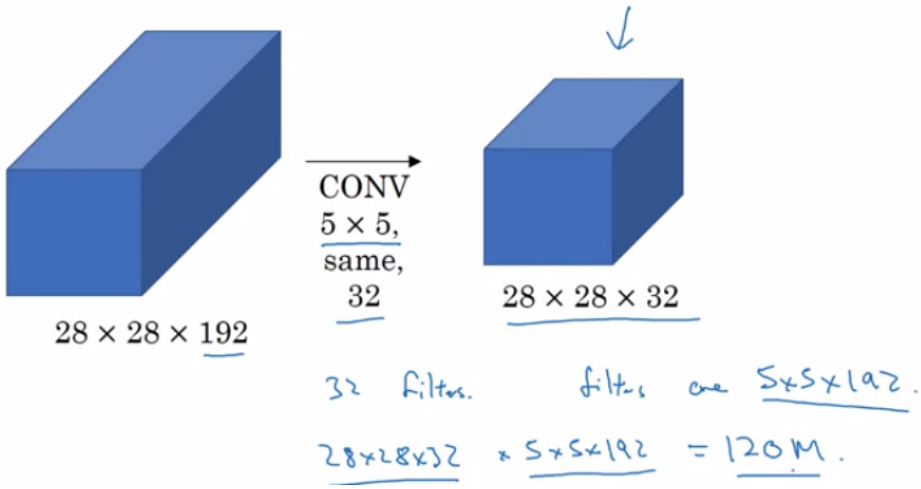
Using 1×1 convolutions is a good way to shrink n_C as well, whereas pulling layers, we can use just to shrink n_H and n_W.

Inception network motivation :



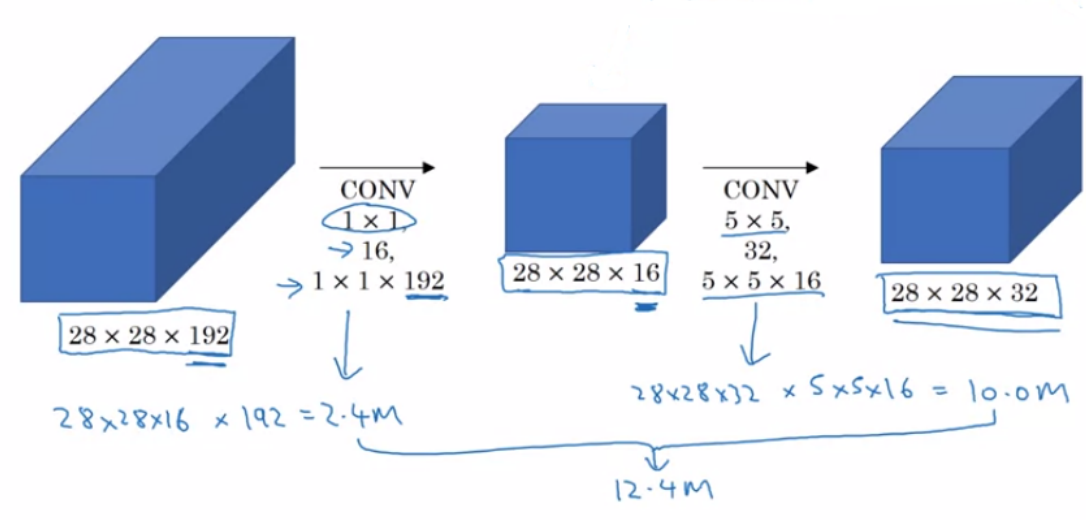
Inception network motivation

The basic idea is that instead of you needing to pick one of these filter sizes or pooling you want and committing to that, you can do them all and just concatenate all the outputs, and let the network learn whatever parameters it wants to use, whatever the combinations of these filter sizes it wants.
But this approach has a problem of computational cost.

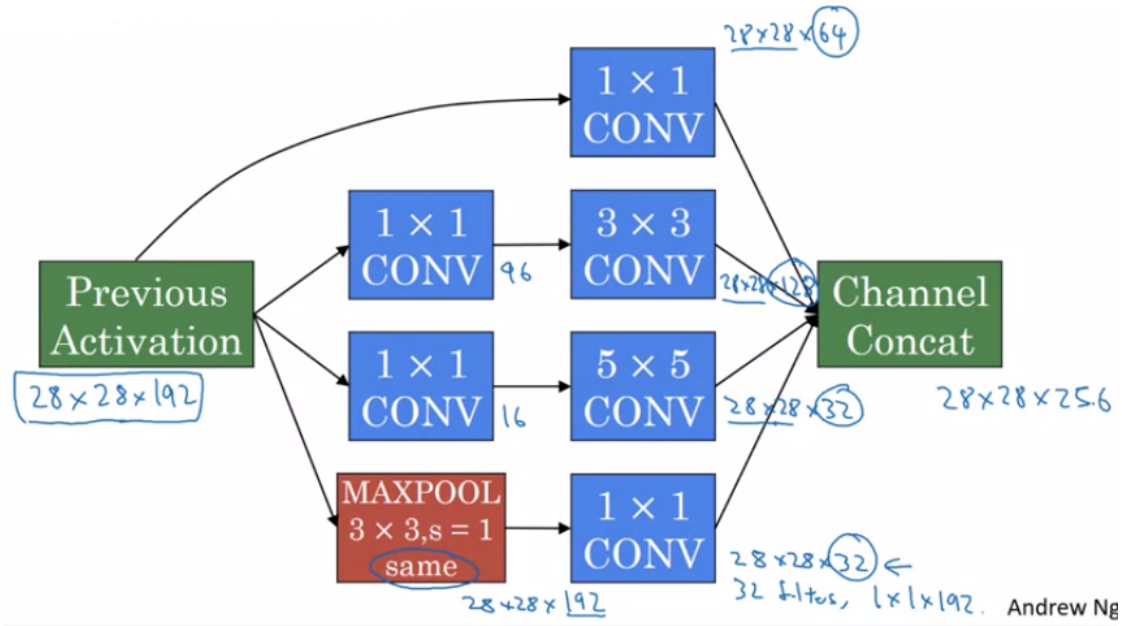


Problem of computational cost

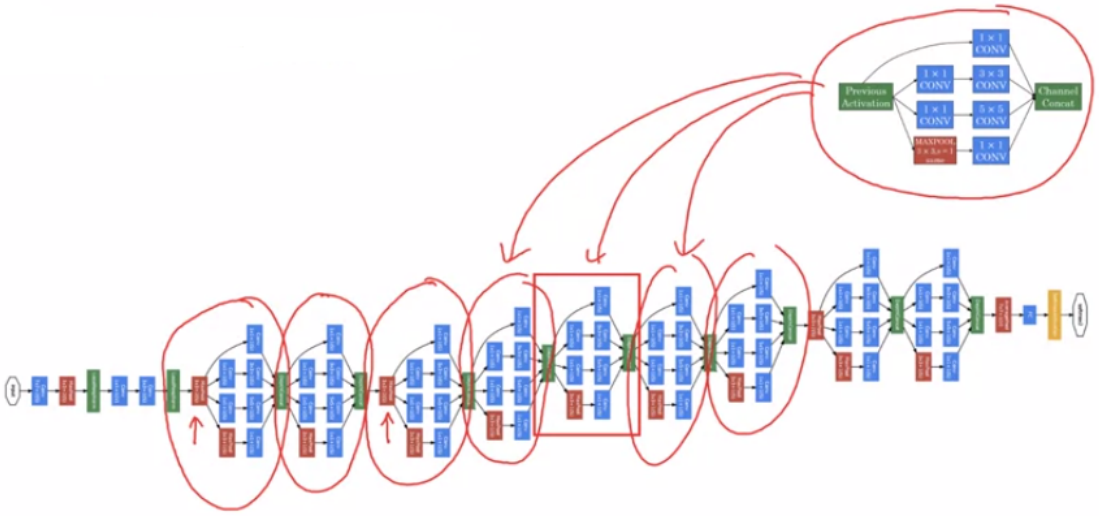
However we can interpret idea 1×1 convolutions to solve this issue.



Inception module :

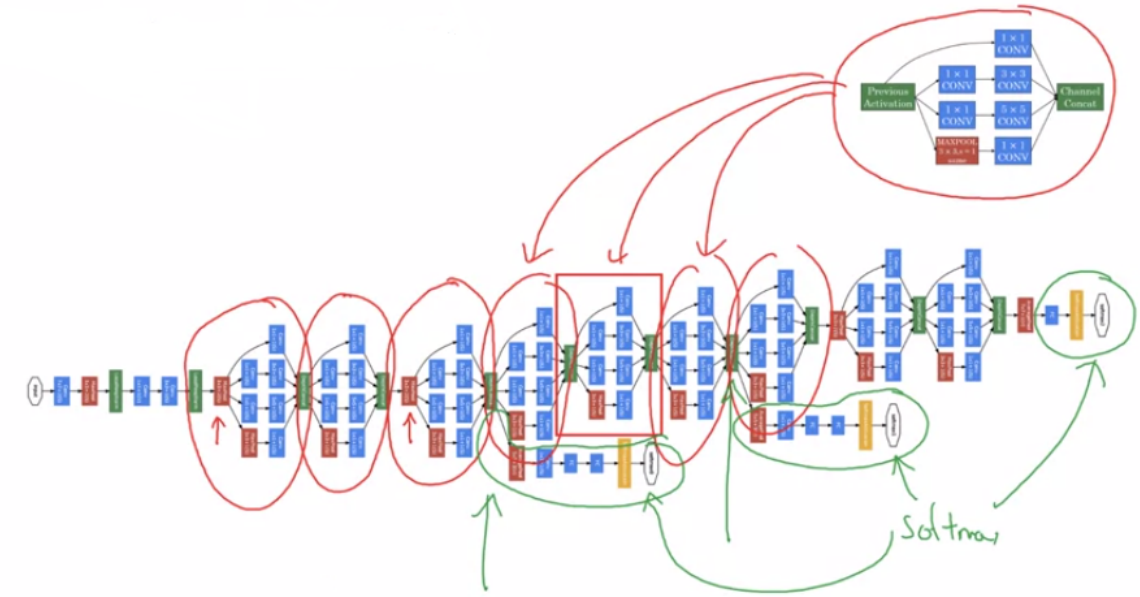


Inception module



Inception network

Also we can add more outputs.



Inception network with multiple outputs

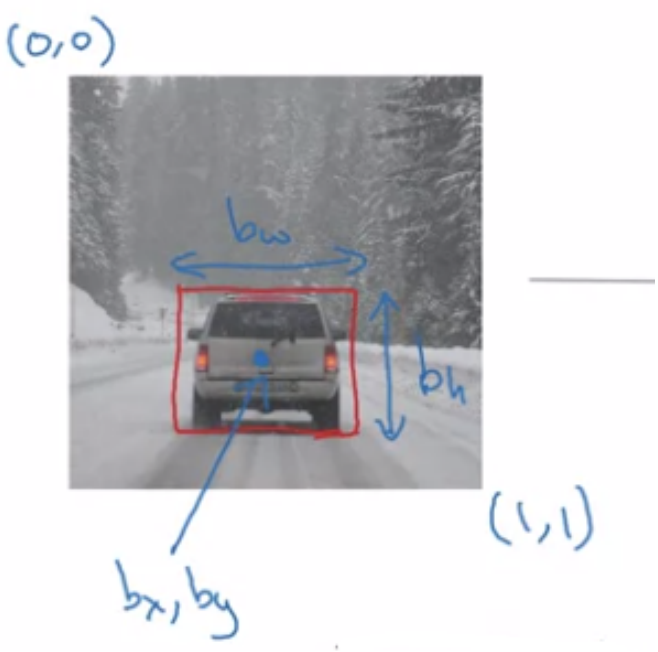
https://arxiv.org/abs/1409.4842

Object Localization

Classification with localization : We want to classify and localize object in the image as well.

$$\begin{cases} pedestrian - 1 \\ car - 2 \\ motorcycle - 3 \\ background - 4 \end{cases}$$

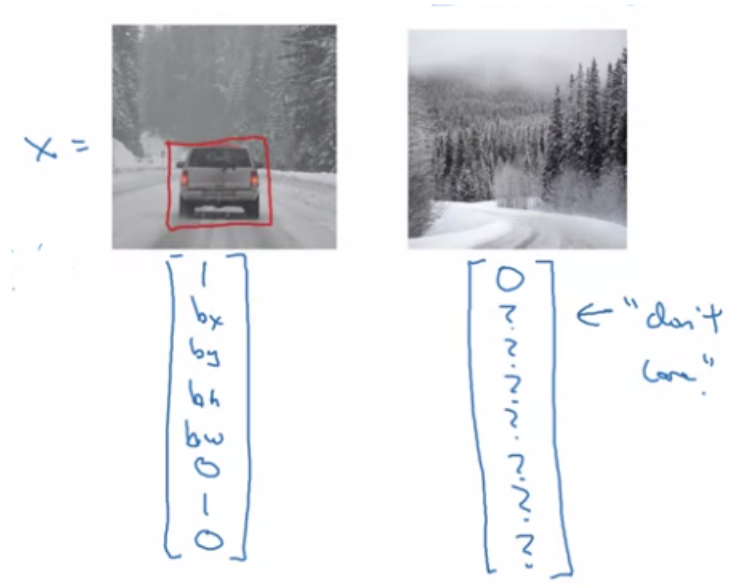
Image $\rightarrow$ ConvNet $\rightarrow$ (SoftMax(4) , (b_x, b_y, b_h, b_w))



Object detection

Need to output b_x, b_y, b_h, b_w, class label(1 - 4). $y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$,

p_c– is there an object? c_i–classes, b_x, b_y, b_h, b_w– bounding box parameters.



Output examples

Landmark detection :

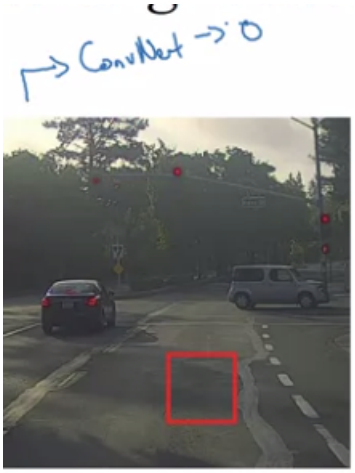
Landmark detection

Image → ConvNet → (face?, l_{1x}, l_{1y} , ... l_{64x}, l_{64y})

Object detection

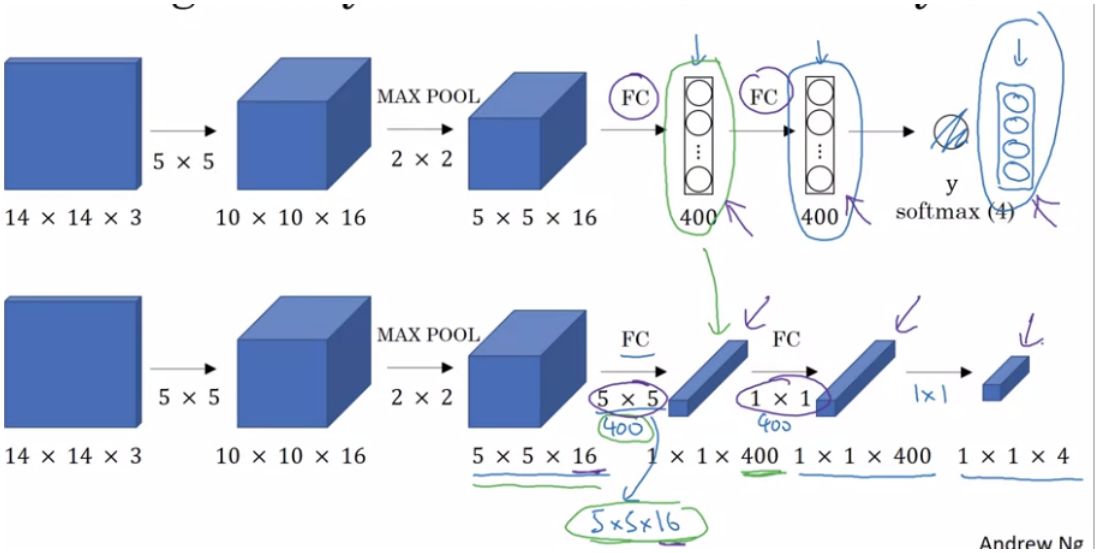
Sliding window detection : You're feeding just the region of the image in the red squares of the ConvNet and run the ConvNet again. And then you do that with a third image and so on. And you keep going until you've slid the window across every position in the image.

Sliding window (1)



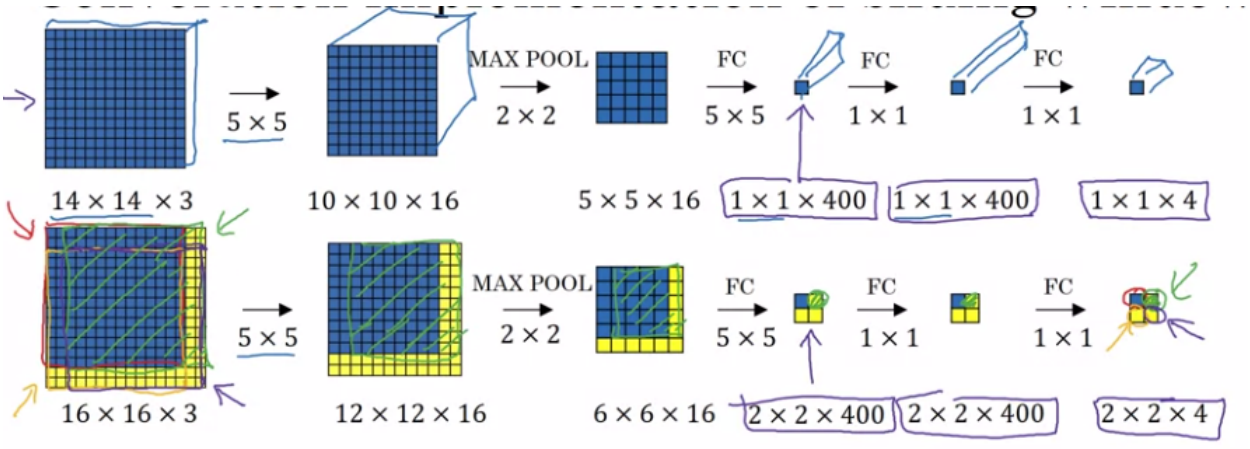
Sliding window (2)

Turning FC layer into convolutional layers :

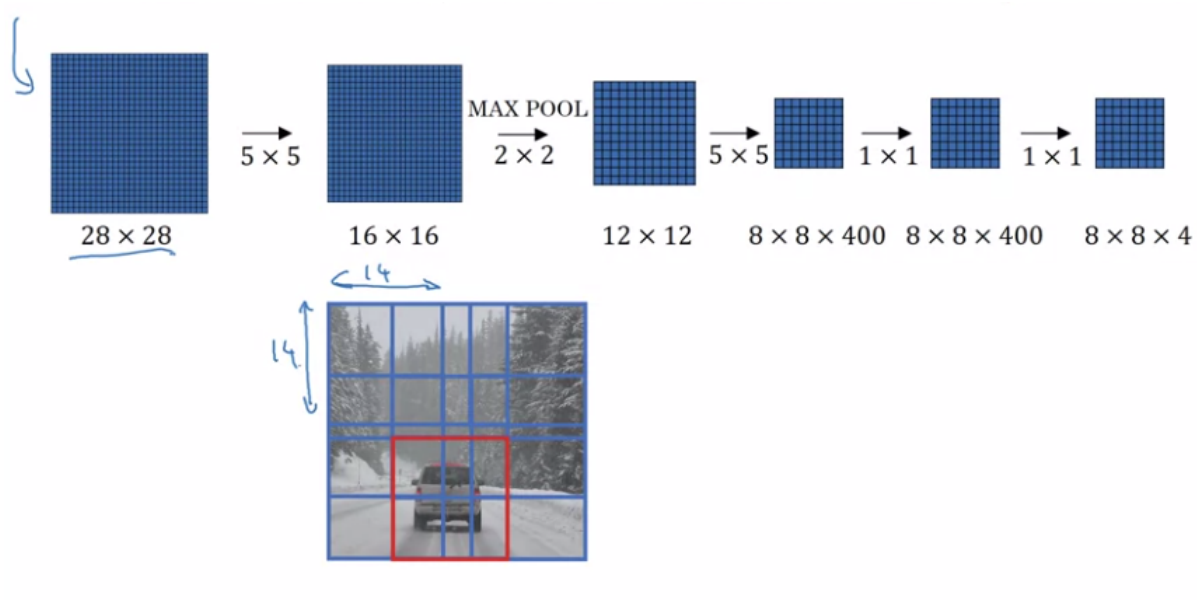


Turning FC layer

Convolution implementation of sliding windows : Instead of forcing you to run four propagation on four subsets of the input image independently, Instead, it combines all four into one form of computation and shares a lot of the computation in the regions of image that are common.



Sliding window to convolution (1)



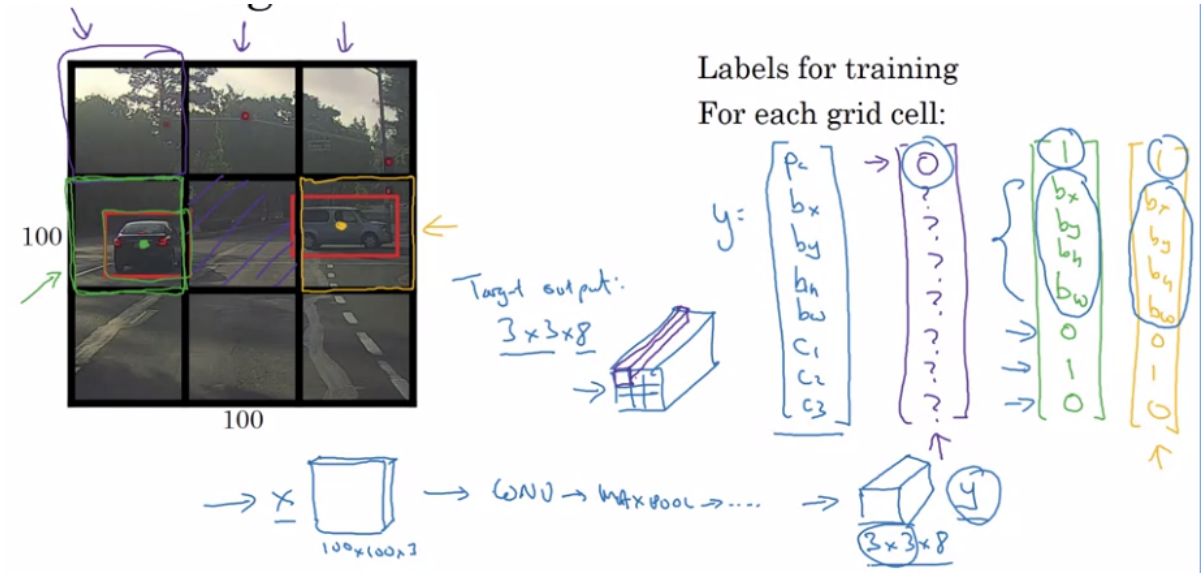
Sliding window to convolution (2)

We can implement the entire image, all maybe 28 by 28 and convolutionally make all the predictions at the same time by one forward pass through this big convnet and hopefully have it recognize the position of the car.
https://arxiv.org/abs/1312.6229

YOLO algorithm

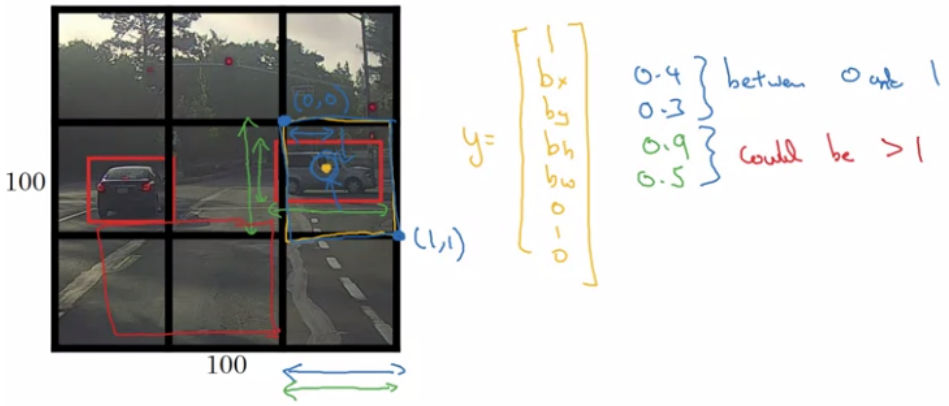
Bounding Box Predictions : We're going to place down a grid on this image. And for the purposes of illustration, We're going to use a 3 by 3 grid. And the basic idea is you're going to take the image classification and localization algorithm and apply that to each of the nine grid cells of this image. So for each of the nine grid cells, you specify a y, where the y is this eight dimensional vector, same as you saw previously.

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



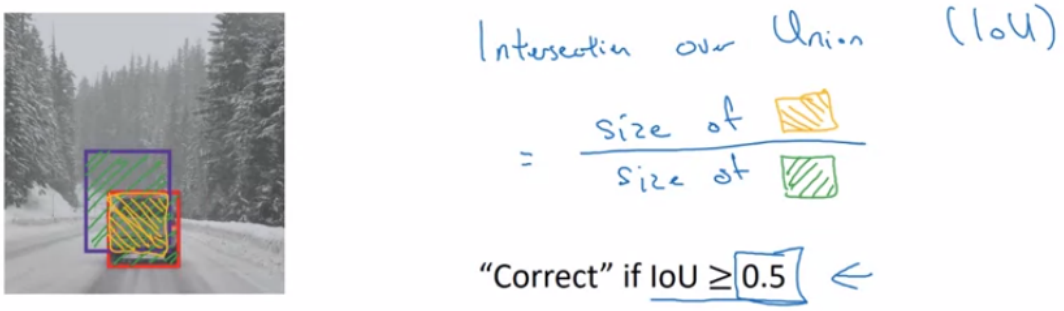
Bounding box prediction

In YOLO algorithm b_h, b_w could be $\geqslant 1$, but $b_x, b_y \in [0, 1]$



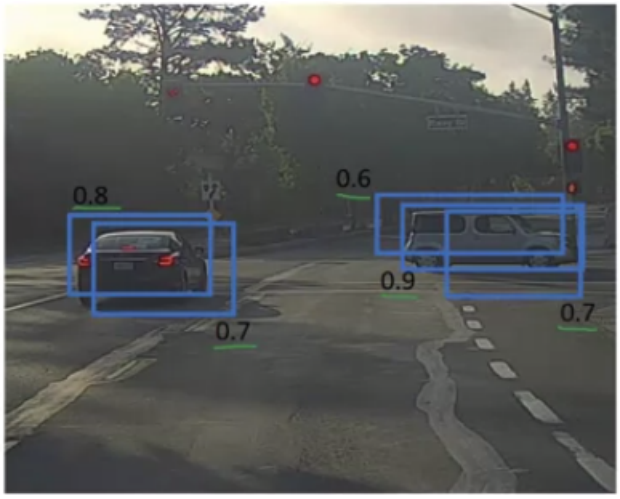
Specify the bounding boxes

Intersection over union : IoU is one way to map localization, to accuracy where you just count up the number of times an algorithm correctly detects and localizes an object where you could use a definition like this, of whether or not the object is correctly localized. But more generally, IoU is a measure of the overlap between two bounding boxes. Where if you have two boxes, you can compute the intersection, compute the union, and take the ratio of the two areas. And so this is also a way of measuring how similar two boxes are to each other.



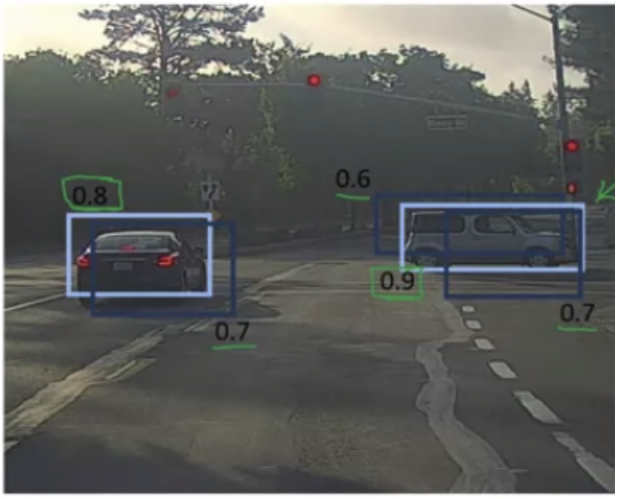
IoU

Non-max Suppression : One of the problems of Object Detection, is that your algorithm may find multiple detections of the same objects. Rather than detecting an object just once, it might detect it multiple times. Non-max suppression is a way for you to make sure that your algorithm detects each object only once.



Multiple detections

So concretely, what it does, is it first looks at the probabilities associated with each of these detections. And it first takes the largest one, which in this case is 0.9. Then looks at all of the remaining rectangles and all the ones with a high overlap, with a high IOU and discard them.

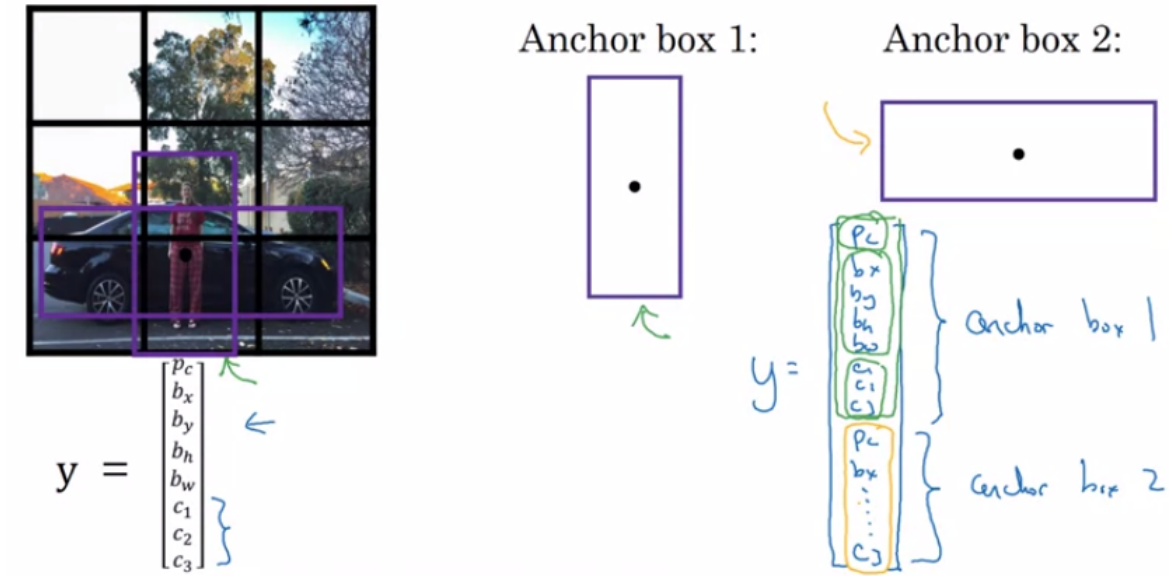


Remaining rectangles

Non-max suppression algorithm :
While there are remaining boxes :

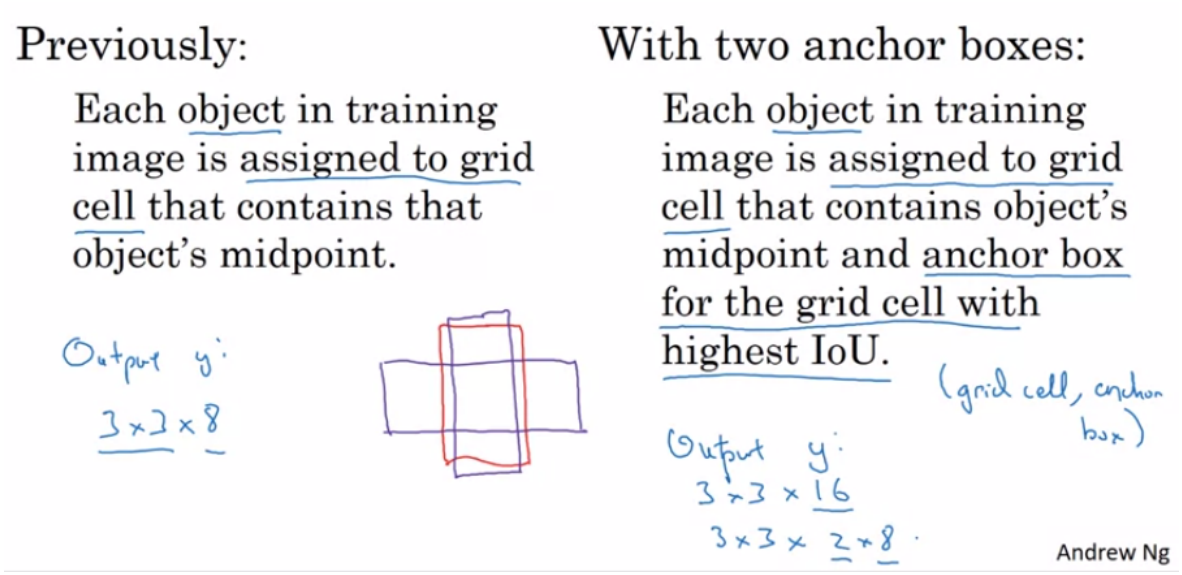
- Pick the box with the largest p_c(probability). Output that as a prediction.
- Discard any remaining box with IoU $\geqslant 0.5$ with the box output in the previous step.

Anchor boxes : One of the problems with object detection is that each of the grid cells can detect only one object. What if a grid cell wants to detect multiple objects? We can use the idea of anchor boxes. You are going to pre-define two different shapes called, anchor boxes or anchor box shapes. And what you are going to do is now, be able to associate two predictions with the two anchor boxes. And in general, you might use more anchor boxes.

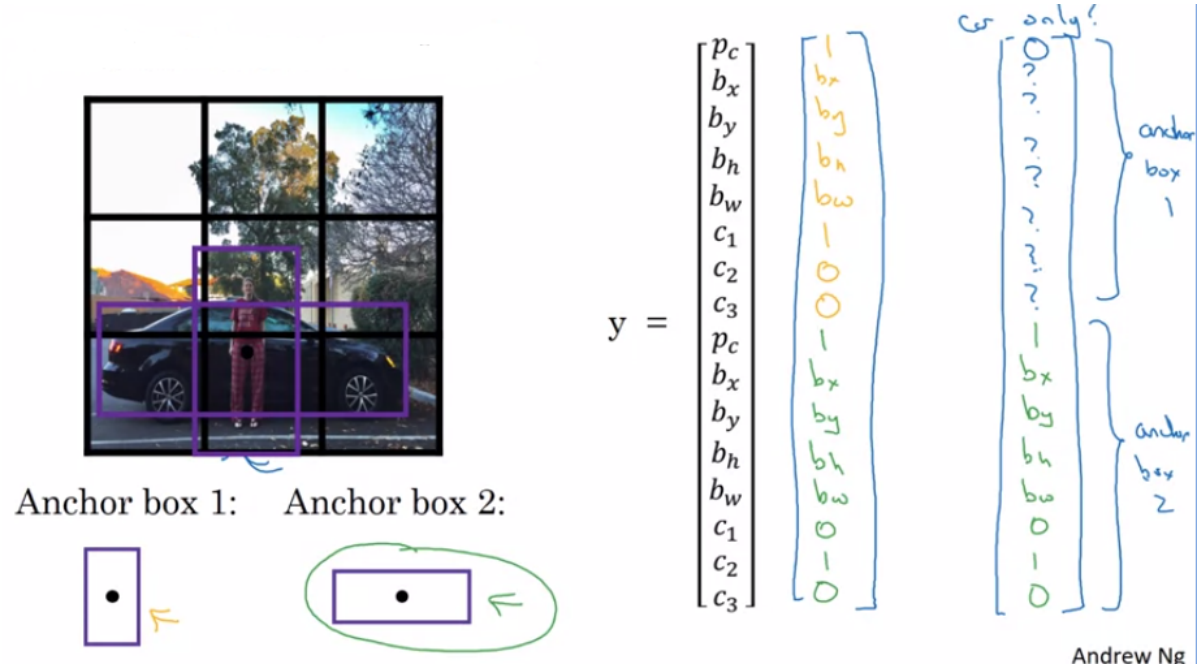


Anchor boxes example (1)

Anchor box algorithm :

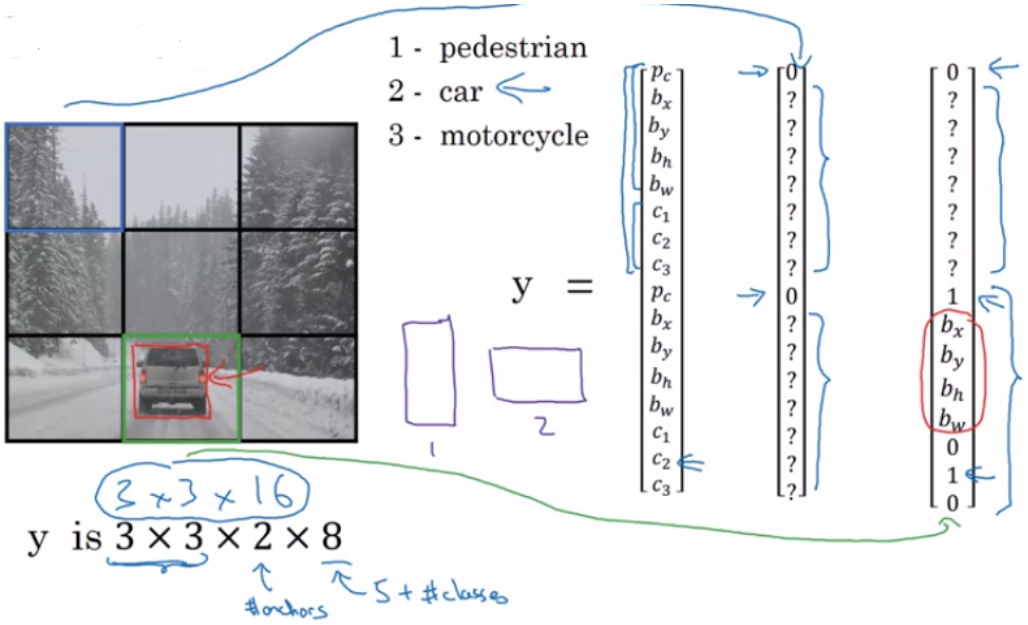


Anchor box algorithm



Anchor box example (2)

YOLO algorithm training :



Training

YOLO algorithm :

- For each grid call, get 2 predicted bounding boxes.
- Get rid of low probability predictions.
- For each class use non-max suppression to generate final predictions.

https://arxiv.org/abs/1506.02640

Face Recognition

Face verification and recognition :
Verification

- Input image, name/ID
- Output whether the input image is that of the claimed person

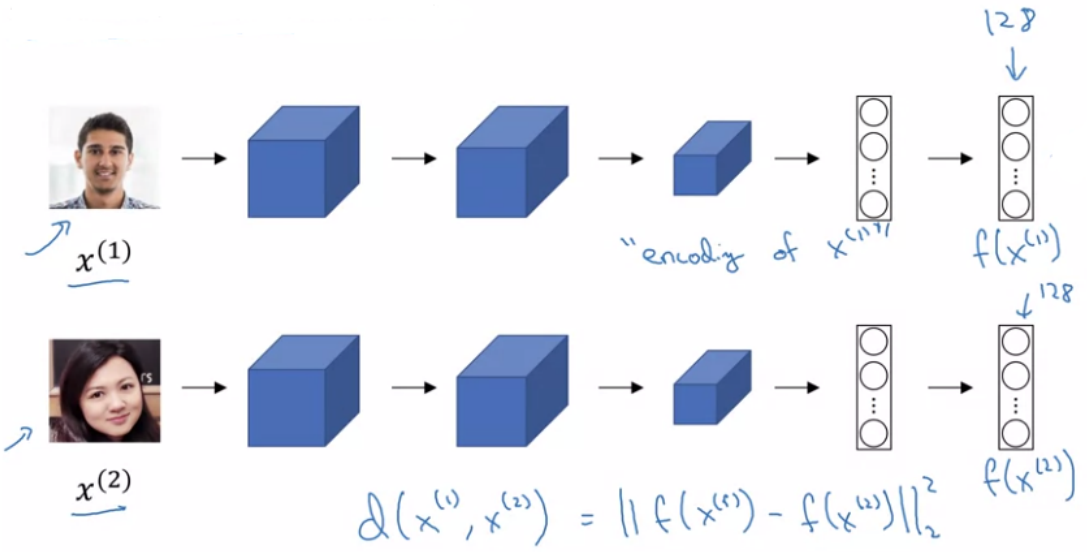
Recognition

- Has a database of K persons
- Get an input image
- Output ID if the image is any of the K persons(or "not recognized")

Similarity function :
$d(\text{img1}, \text{img2})$ = degree of difference between images.
If $d(\text{img1}, \text{img2}) \leqslant \tau \rightarrow$ "same"
If $d(\text{img1}, \text{img2}) > \tau \rightarrow$ "different"

Siamese Network : We're going to encode image in vector of let's say 128 numbers computed by some fully connected layer that is deeper in the network. So this idea of

running two identical, convolutional neural networks on two different inputs and then comparing them, sometimes that's called a Siamese neural network architecture.



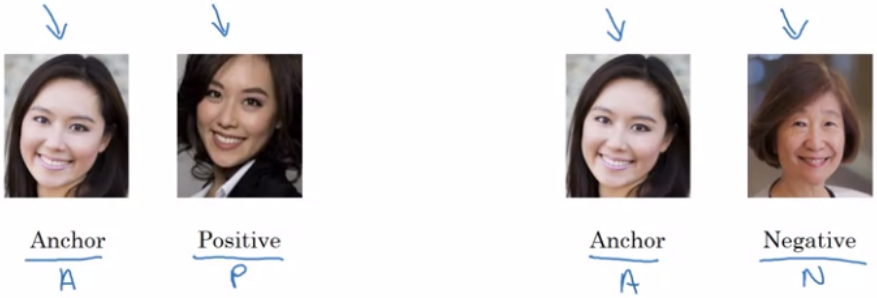
Picture encoding

$d(x^{(1)}, x^{(2)}) = \|f(x^{(1)}) - f(x^{(2)})\|_2^2$
Parameters of NN define and encoding $f(x^{(i)})$
Learn parameters so that :

- If, $x^{(i)}, x^{(j)}$ are the same person, $\|f(x^{(i)}) - f(x^{(j)})\|_2^2$ is small.
- If, $x^{(i)}, x^{(j)}$ are different person, $\|f(x^{(i)}) - f(x^{(j)})\|_2^2$ is large.

https://www.cs.toronto.edu/ ranzato/publications/taigman_cvpr14.pdf

Triplet Loss : We're going do is always look at one anchor image and then you want to distance between the anchor and the positive image, really a positive example, meaning as the same person to be similar. Whereas, you want the anchor when pairs are compared to the negative example for their distances to be much further apart.



Triplet

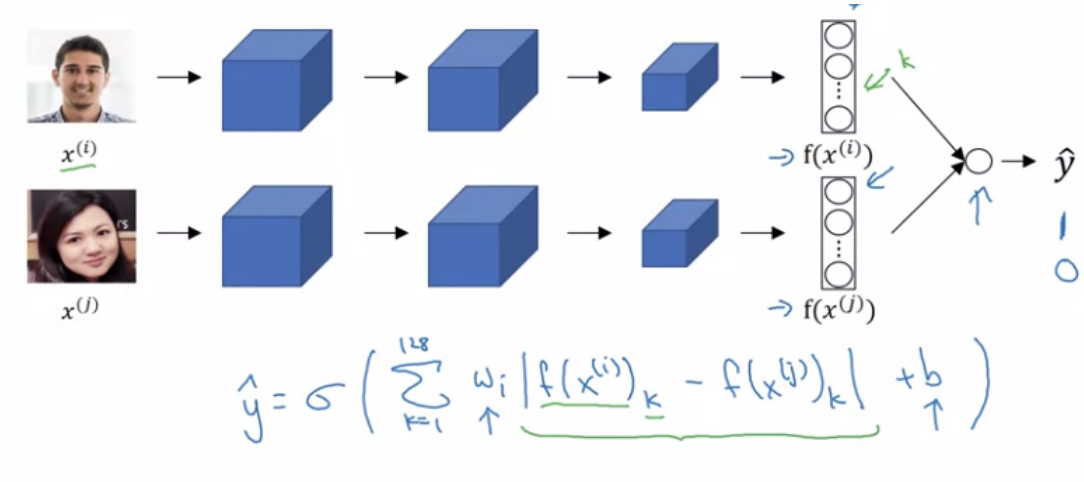
Want : $\underbrace{\|f(A) - f(P)\|}_{d(A,P)} + \alpha \leqslant \underbrace{\|f(A) - f(N)\|}_{d(A,N)}$

Loss function : $L(A, P, N) = \max(\|\text{f(A)} - \text{f(P)}\| - \|\text{f(A)} - \text{f(N)}\| + \alpha, 0)$

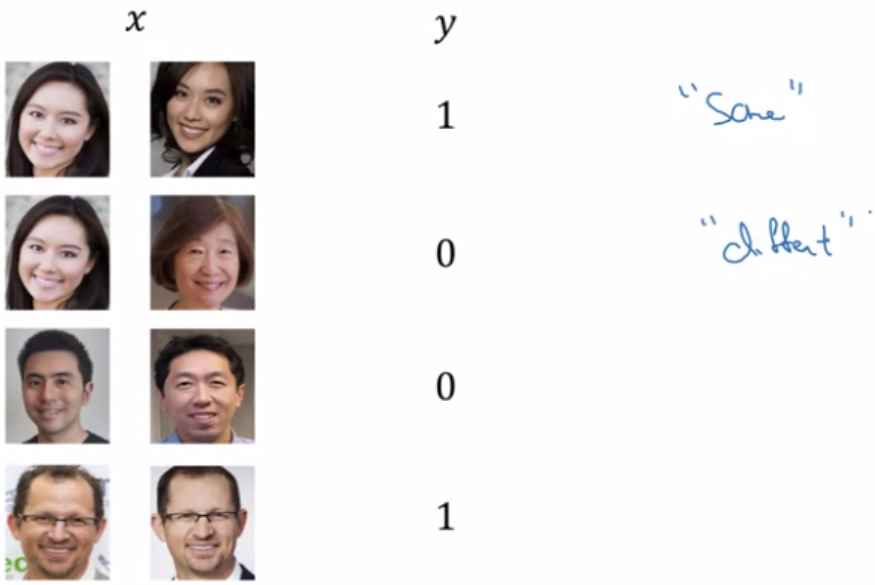
$J = \sum_{i=1}^{m} L(A^{(i)}, P^{(i)}, N^{(i)})$

During training, if A, P, N are chosen randomly, $d(A, P) + \alpha \leqslant d(A, N)$ is easily satisfied. So choose triplets that're "hard"to train on $d(A, P) \approx d(A, N)$.
https://arxiv.org/abs/1503.03832

Learning the similarity functuion :

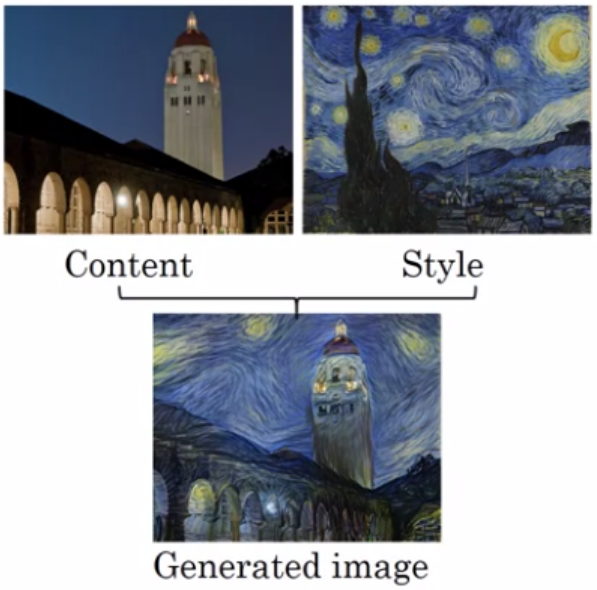


Learning the similarity function



Face verification supervised learning

Neural style transfer



Neural style transfer

Cost function :
G - generated image
C - content image
S - style image
$J(G) = \alpha J_{\text{content}}(C, G) + \beta J_{\text{style}}(S, G)$ https://arxiv.org/abs/1508.06576

Find the generated image :

- Initiate G randomly
- Use gradient descent to minimize $J(G)$. $J(G) = J(G) - \frac{\alpha}{\partial G} J(G)$

https://arxiv.org/abs/1508.06576

Content cost function :
$J(G) = \alpha J_{\text{content}}(C, G) + \beta J_{\text{style}}(S, G)$

- Say you use hidden layer l to compute content cost.
- Use pre-trained ConvNet (E.g., VGG network)
- Let $a^{[l](C)}, a^{[l](G)}$ be the activation of layer l on the images.
- If $a^{[l](C)}$ and $a^{[l](G)}$ are similar, both images have similar content.

$J_{\text{content}}(C, G) = \frac{1}{2} \| a^{[l](C)} - a^{[l](G)} \|^2$

Style cost function :
Let $a^{[l]}_{i,j,k}$ - activation at (i, j, k). $G^{[l]}$ is $n_c^{[l]} \times n_c^{[l]}$

$G^{[l](S)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l](S)}_{i,j,k} a^{[l](S)}_{i,j,k'}$ – Gram matrix.

$G^{[l](G)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l](G)}_{i,j,k} a^{[l](G)}_{i,j,k'}$ – Gram matrix.

$$J^{[l]}_{\text{style}} = \|G^{[l](S)} - G^{[l](G)}\|^2_F = \frac{1}{(2n_H^{[l]} n_W^{[l]} n_C^{[l]})^2} \sum_k \sum_{k'} (G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'})^2$$

$$J_{\text{style}} = \sum_l \lambda^{[l]} J^{[l]}_{\text{style}}(S, G)$$