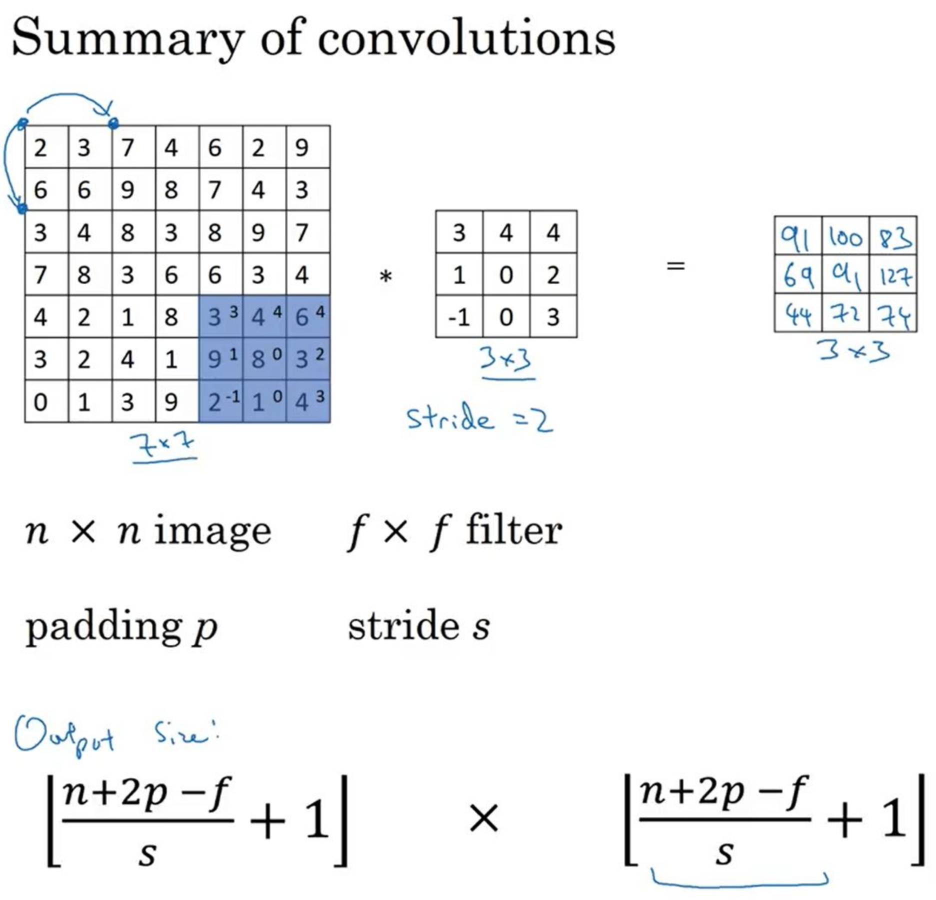
Convolutional neural network

- ➔ Application - Image processing, Computer vision, Motion detection and so on
- ➔ Convolutional operation (*) > it's **a dot product of filter sliding on the input image**
  - ◆ Filter (Edge detection)
    - Horizontal
    - Vertical
    - 45 Degree



Summary of convolutions

$$\underbrace{\begin{bmatrix} 2 & 3 & 7 & 4 & 6 & 2 & 9 \\ 6 & 6 & 9 & 8 & 7 & 4 & 3 \\ 3 & 4 & 8 & 3 & 8 & 9 & 7 \\ 7 & 8 & 3 & 6 & 6 & 3 & 4 \\ 4 & 2 & 1 & 8 & 3^{3} & 4^{4} & 6^{4} \\ 3 & 2 & 4 & 1 & 9^{1} & 8^{0} & 3^{2} \\ 0 & 1 & 3 & 9 & 2^{-1} & 1^{0} & 4^{3} \end{bmatrix}}_{7\times7} * \underbrace{\begin{bmatrix} 3 & 4 & 4 \\ 1 & 0 & 2 \\ -1 & 0 & 3 \end{bmatrix}}_{3\times3} = \underbrace{\begin{bmatrix} 91 & 100 & 83 \\ 69 & 91 & 127 \\ 44 & 72 & 74 \end{bmatrix}}_{3\times3}$$

Stride = 2

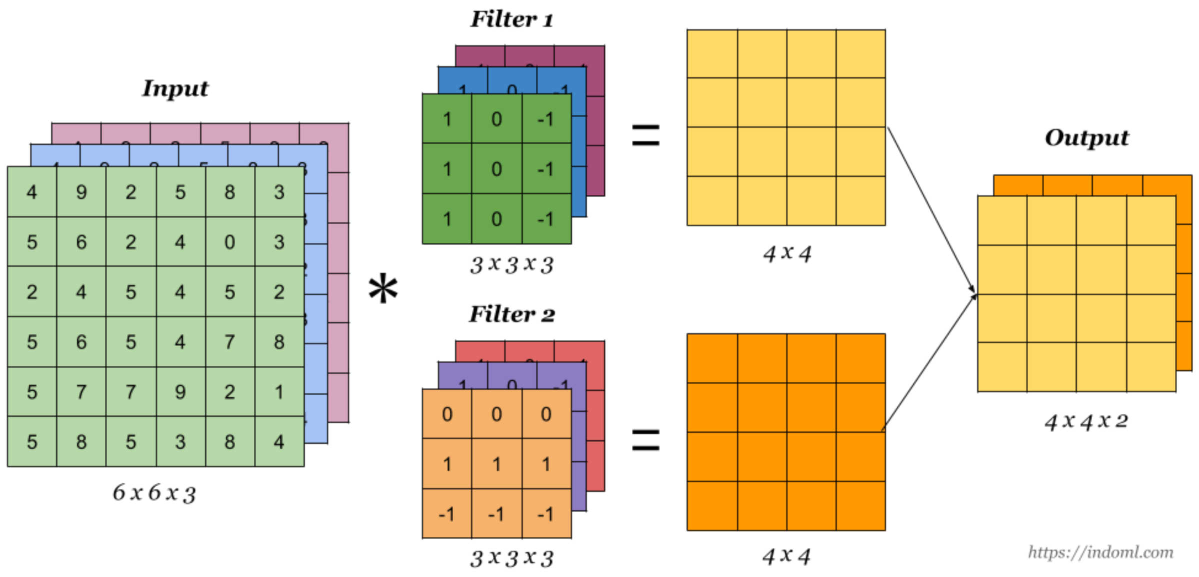
$n \times n$ image $\qquad f \times f$ filter

padding p $\qquad$ stride s

Output Size:

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \underbrace{\frac{n+2p-f}{s}} + 1 \right\rfloor$$

- ◆ Padding
  - Valid
  - Same
- ◆ Stride
- ◆ Convolution over volume
- ◆ Multiple filter



Input (6 x 6 x 3), front channel:

4	9	2	5	8	3
5	6	2	4	0	3
2	4	5	4	5	2
5	6	5	4	7	8
5	7	7	9	2	1
5	8	5	3	8	4

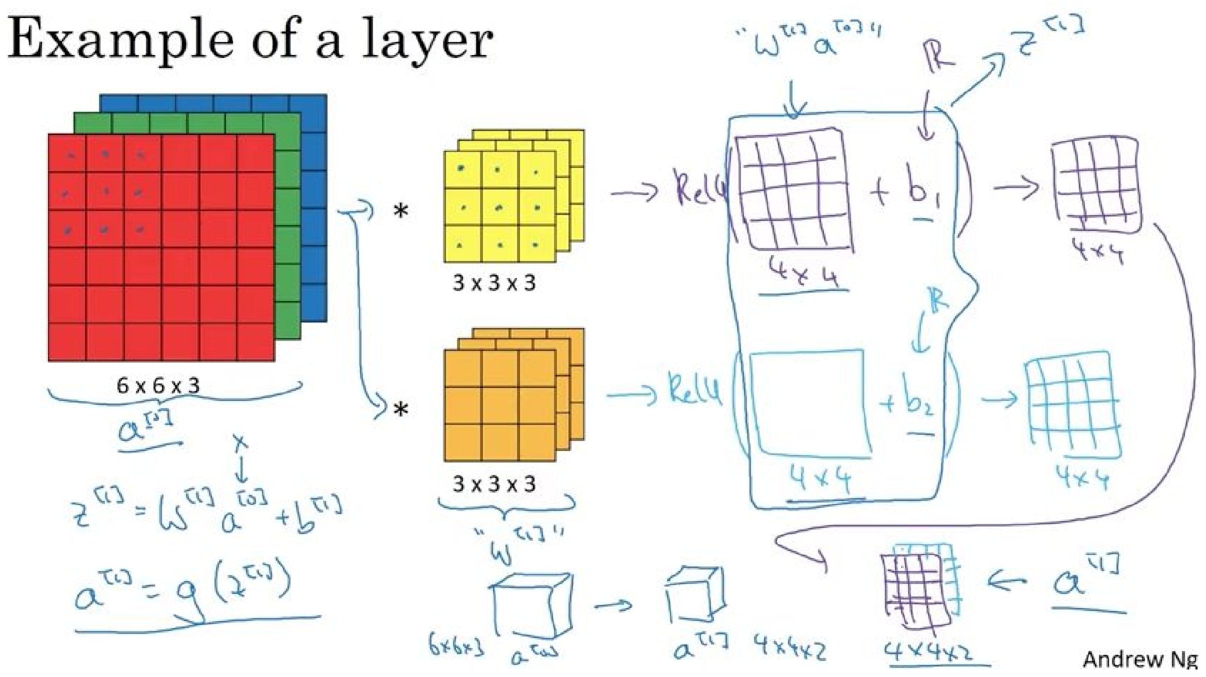
Filter 1 (3 x 3 x 3), front channel:

1	0	-1
1	0	-1
1	0	-1

Filter 2 (3 x 3 x 3), front channel:

0	0	0
1	1	1
-1	-1	-1

- ➔ One layer CNN



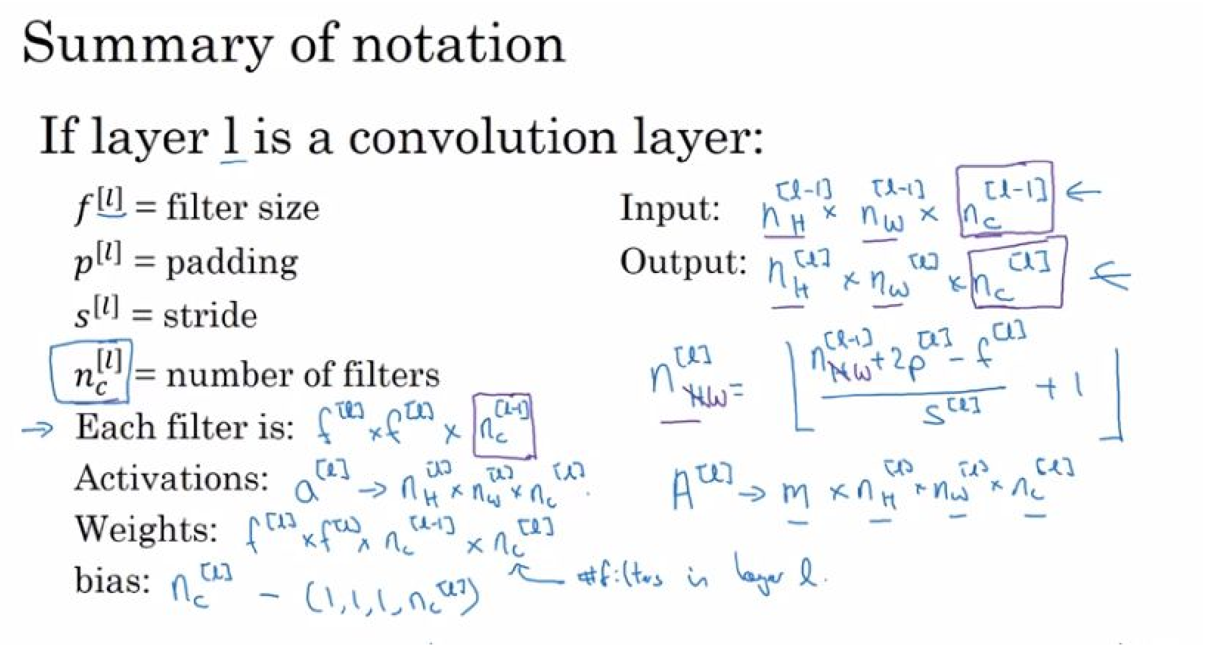
Example of a layer

$z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$

$a^{[1]} = g(z^{[1]})$

The filter is like a weight to find in the normal neural network and the result from those filters will be the input for the next layer.

The result from each filters will be the feature that you are extracting from the image. The parameters that you have to find is the number of each cell in the filter. For example, If you have 10 filter with 3x3x3 filter you will have 270 + 10 bias ~ 280 parameters to estimate the result.



Summary of notation

If layer l is a convolution layer:

$f^{[l]}$ = filter size

$p^{[l]}$ = padding

$s^{[l]}$ = stride

$\boxed{n_c^{[l]}}$ = number of filters

→ Each filter is: $f^{[l]} \times f^{[l]} \times \boxed{n_c^{[l-1]}}$

Activations: $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

Weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

bias: $n_c^{[l]}$ – $(1,1,1,n_c^{[l]})$ ← #filters in layer l.

Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times \boxed{n_c^{[l-1]}}$ ←

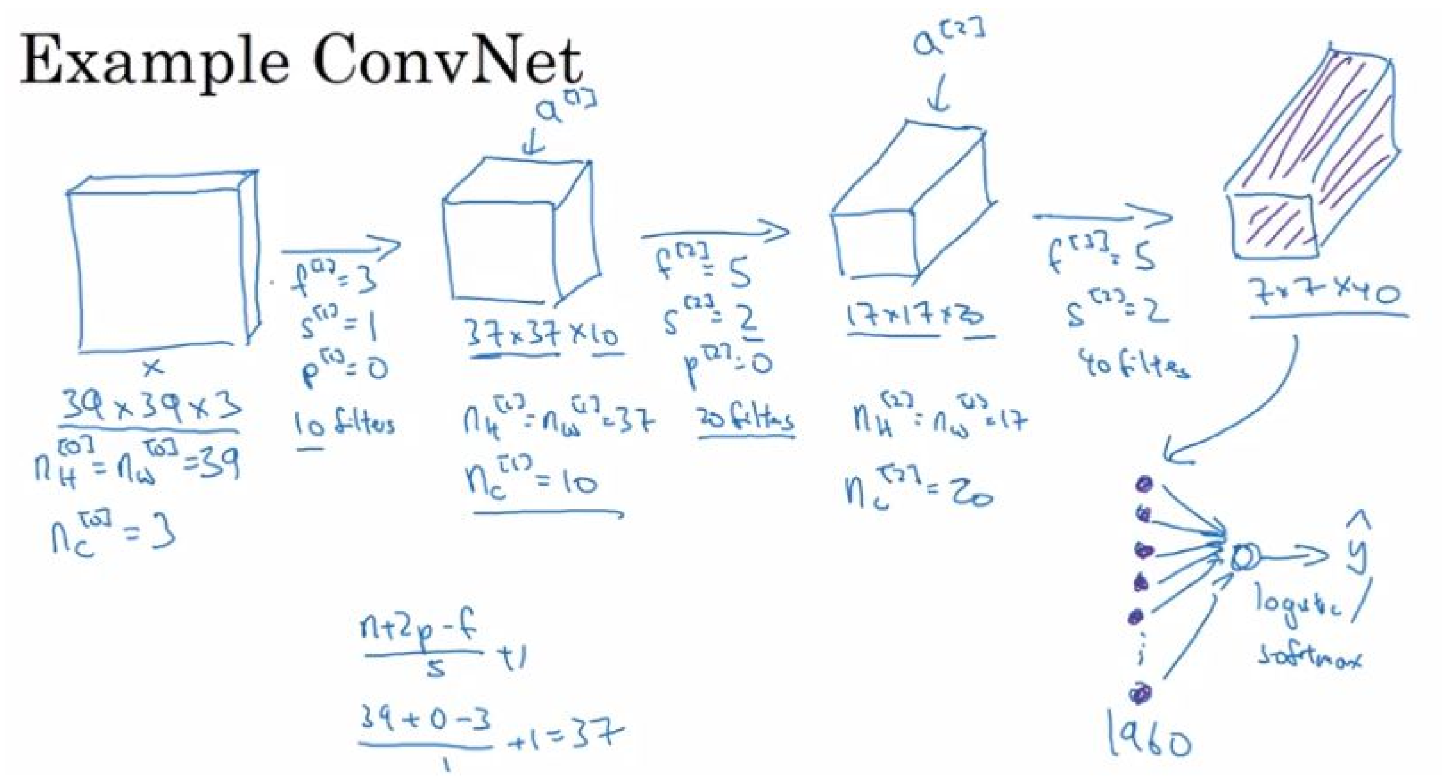
Output: $n_H^{[l]} \times n_W^{[l]} \times \boxed{n_c^{[l]}}$ ←

$$n_{H/W}^{[l]} = \left\lfloor \frac{n_{H/W}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

$A^{[l]} \rightarrow m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

The dimension of each notation in the CNN architecture. The position for number of channel in some papers or on github might be swop to the first position

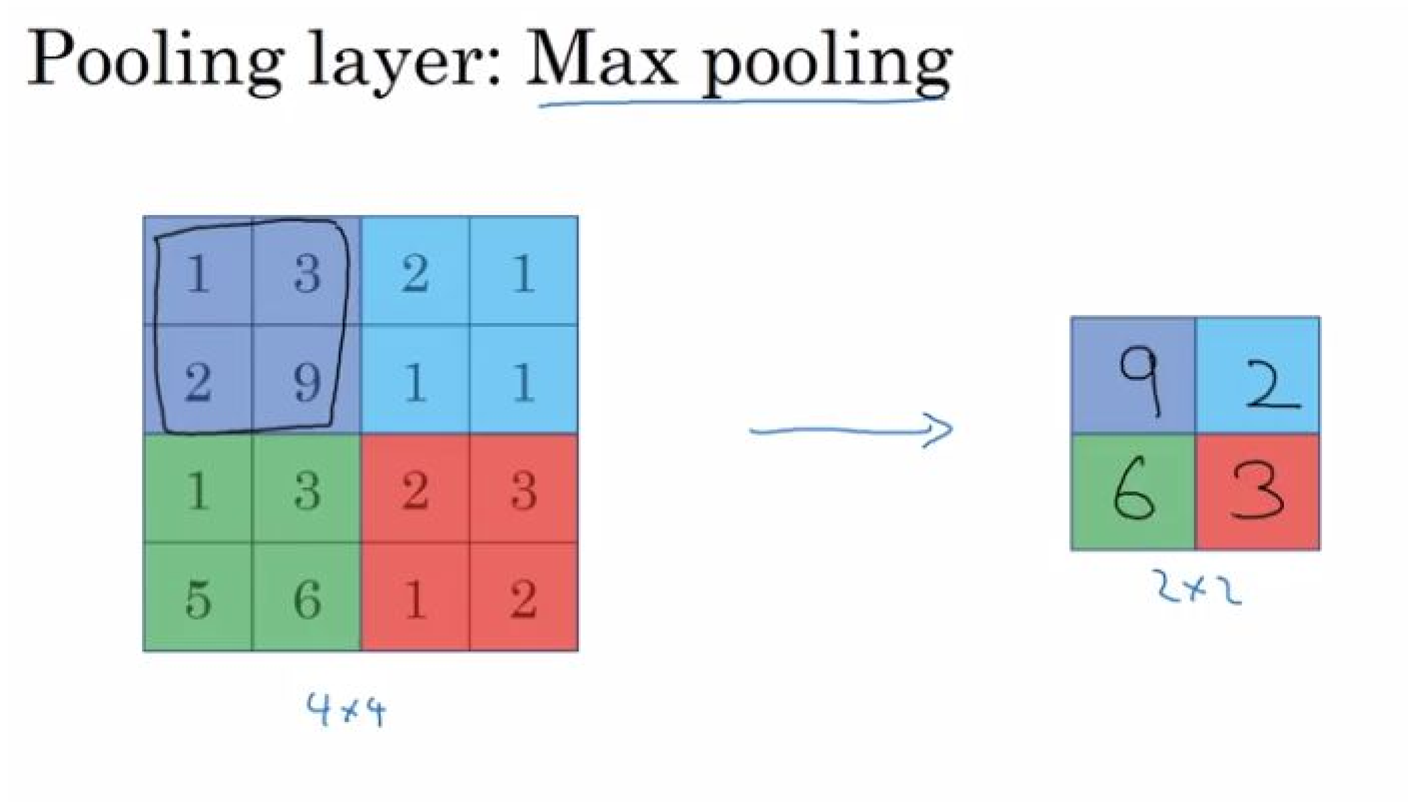
Simpler CNN architecture



Putting the image with RGB channel goes through filter 1 , 2 , 3 and then stacking them to the columns vector and pass to the logistic unit or softmax unit to get the predicted score.

Pooling layer

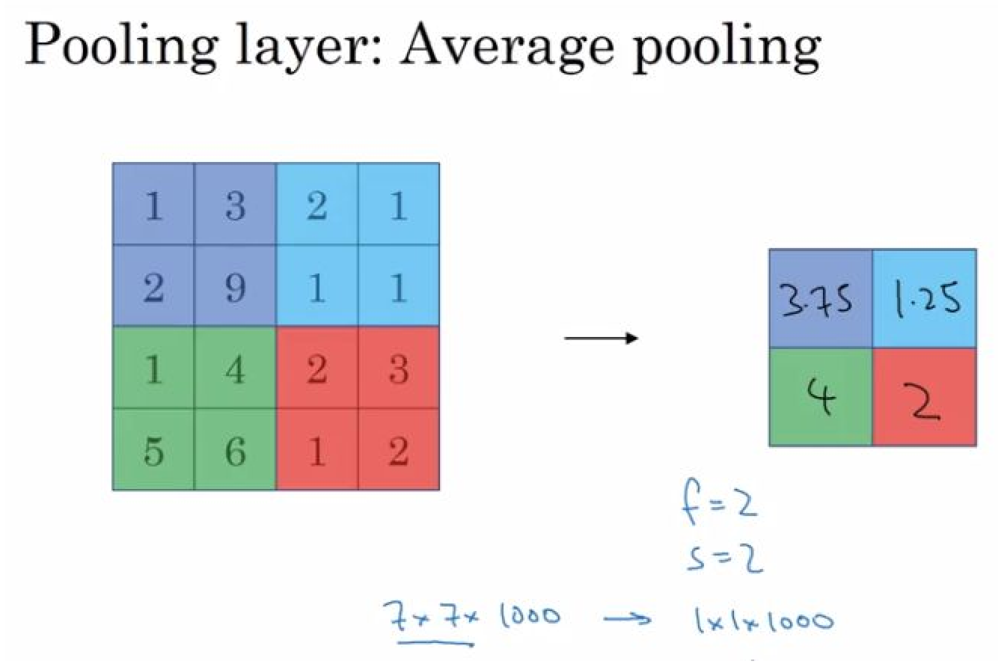
To reduce the size of the image to improve the speed of computation and to make the extracted features get more robust



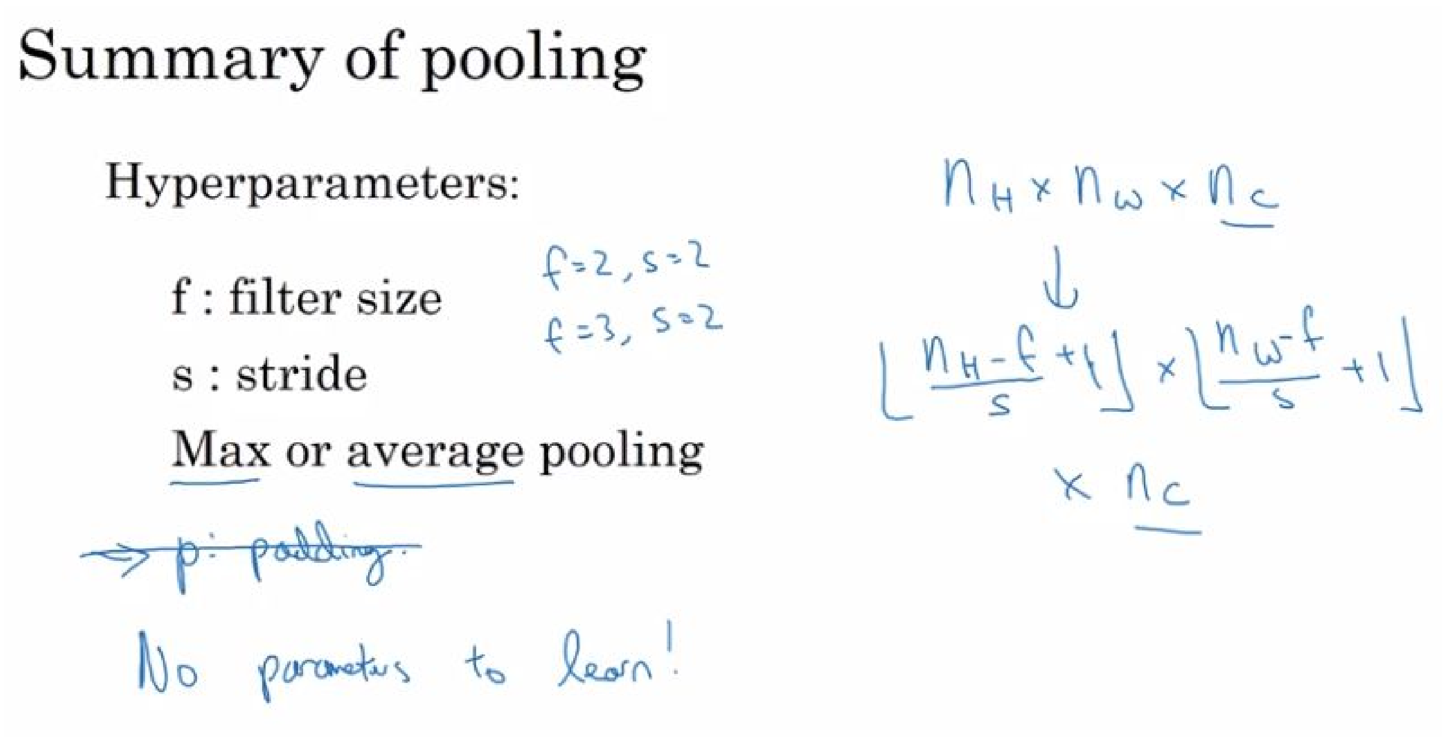
The parameter for the max pooling is F = 2 , S = 2 for the above figure. The reason to use the max pooling layer is to keep the highest number in each region that the max pooling layer overlay. For example the number 9 in the top left of the max pooling layer means that there might be some important points in that zone like edge, or some useful symbols (eyes, nose, eyebrow, etc.)

It has a parameter for the max pooling but it doesn't have a hyper parameter for the algorithms to find out.

The number of channel from the max pooling will be the same as the input dimension. For example, if the input is nh x nw x nc, then the output will be $\frac{n+2p-f}{s}+1$ x $\frac{n+2p-f}{s}+1$ x nc , where the nh is number of height and nw is number of width

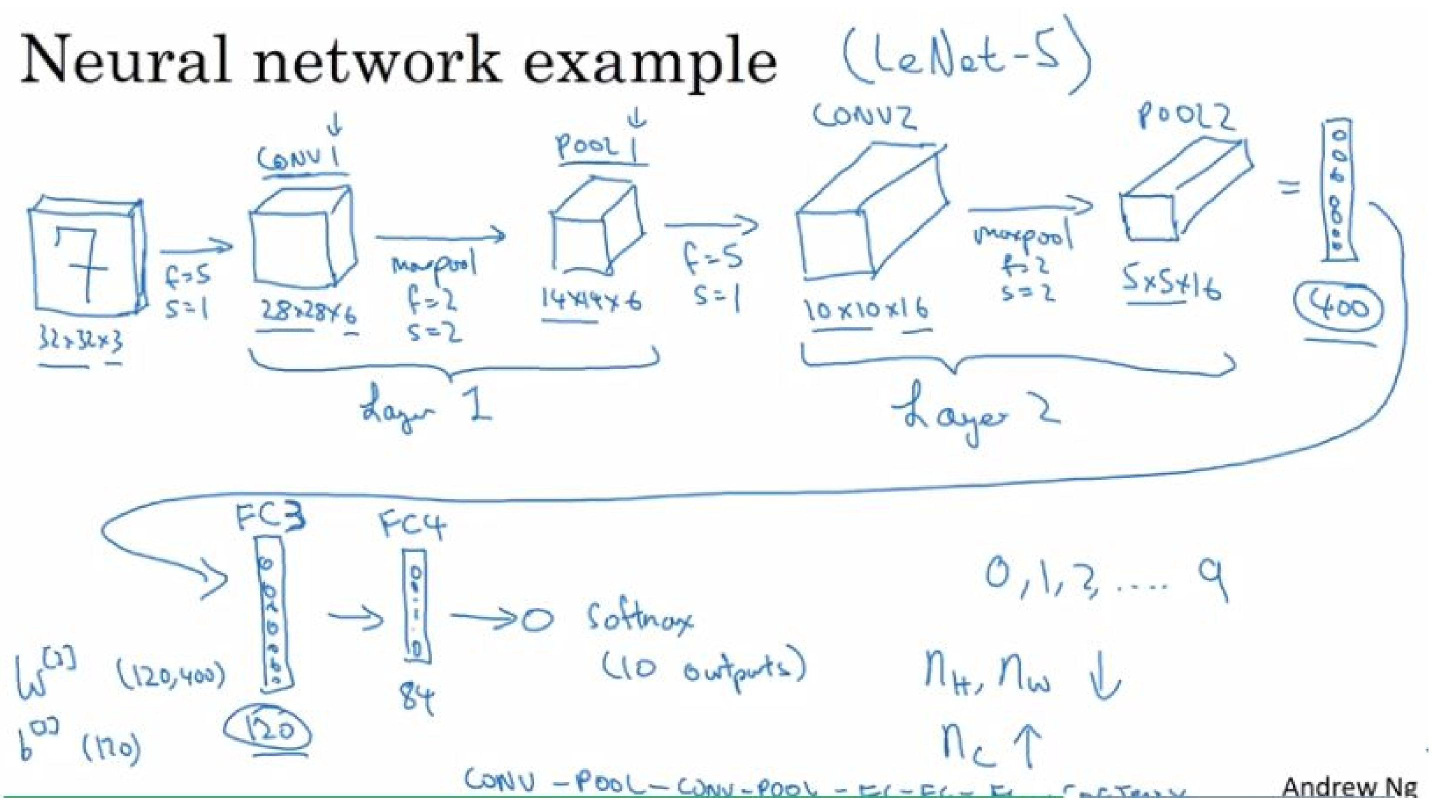


The max pooling is usually popular than the average pooling. You can also add padding to the pooling layer but it's rarely use in the real world. So the padding will be 0 for the pooling layer. So the output dimension will become $\frac{n-f}{s}+1$ x $\frac{n-f}{s}+1$ x nc .

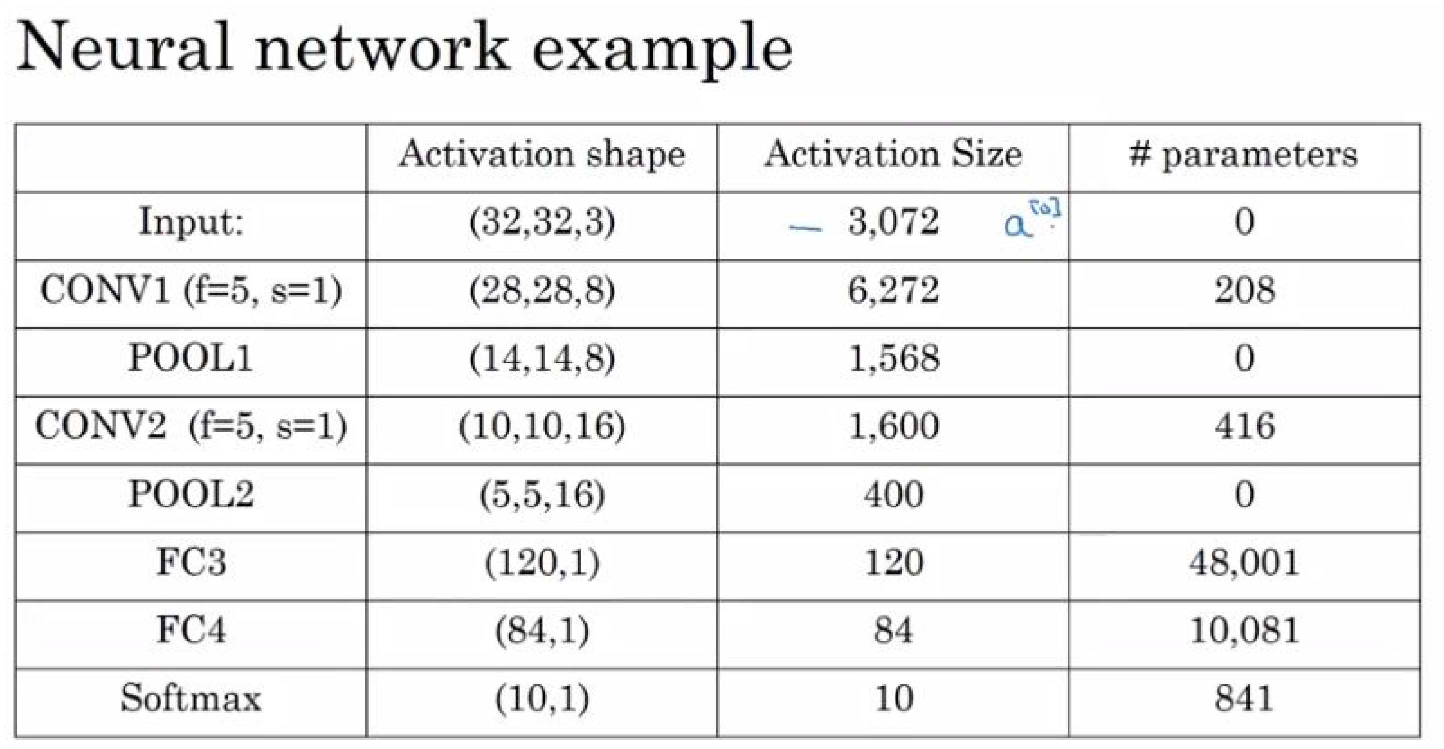


Fully connected layer
Usually when the people report the number of layer in the CNN they report only the layer that has a weight to estimate (so they will ignore the pooling layer because there is no weight to estimate)

The fully connected layer is the normal neural network layer that you use after the convolutional later so that the algorithms can learn from the feature you are extracting from the image.



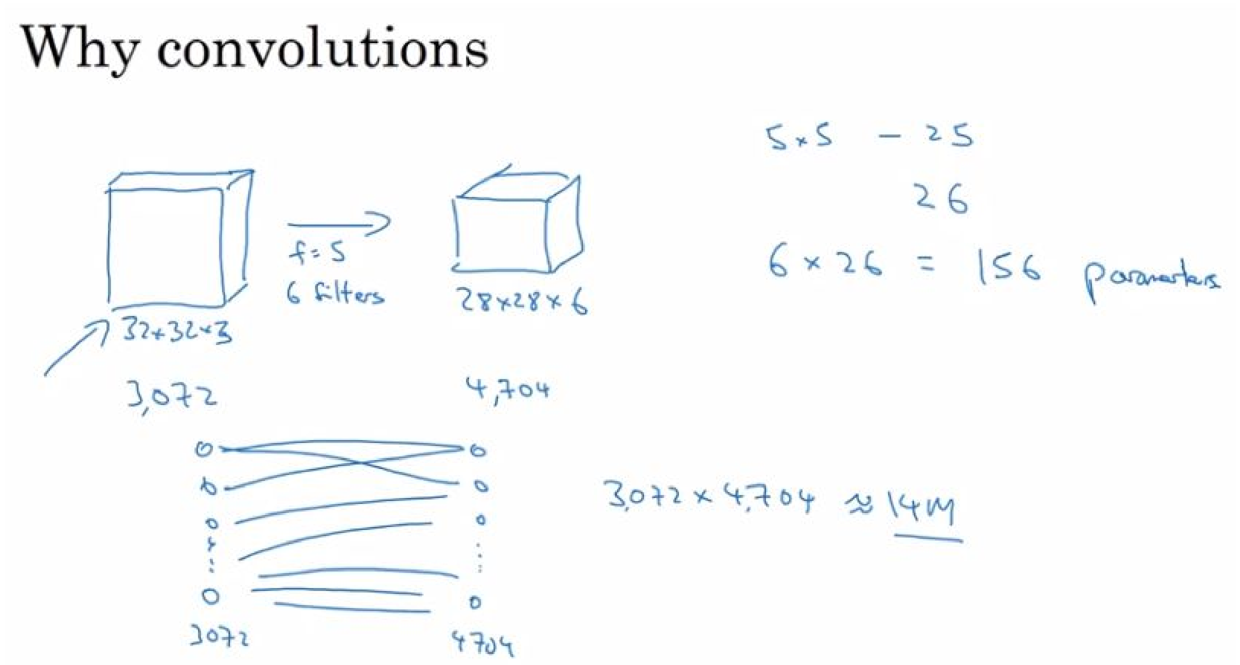
The choice for the user is to select the hyperparameter and architecture from the literature instead of trying it by yourself.


Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	– 3,072 $a^{[0]}$	0
CONV1 (f=5, s=1)	(28,28,8)	6,272	208
POOL1	(14,14,8)	1,568	0
CONV2 (f=5, s=1)	(10,10,16)	1,600	416
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	48,001
FC4	(84,1)	84	10,081
Softmax	(10,1)	10	841

The noticing are that the # or parameters is 0 for the pooling layer and fewer for the CONV layer and get so much larger in the fully connected layer. Secondly, the activation size is decreasing as well as the layer get deeper.

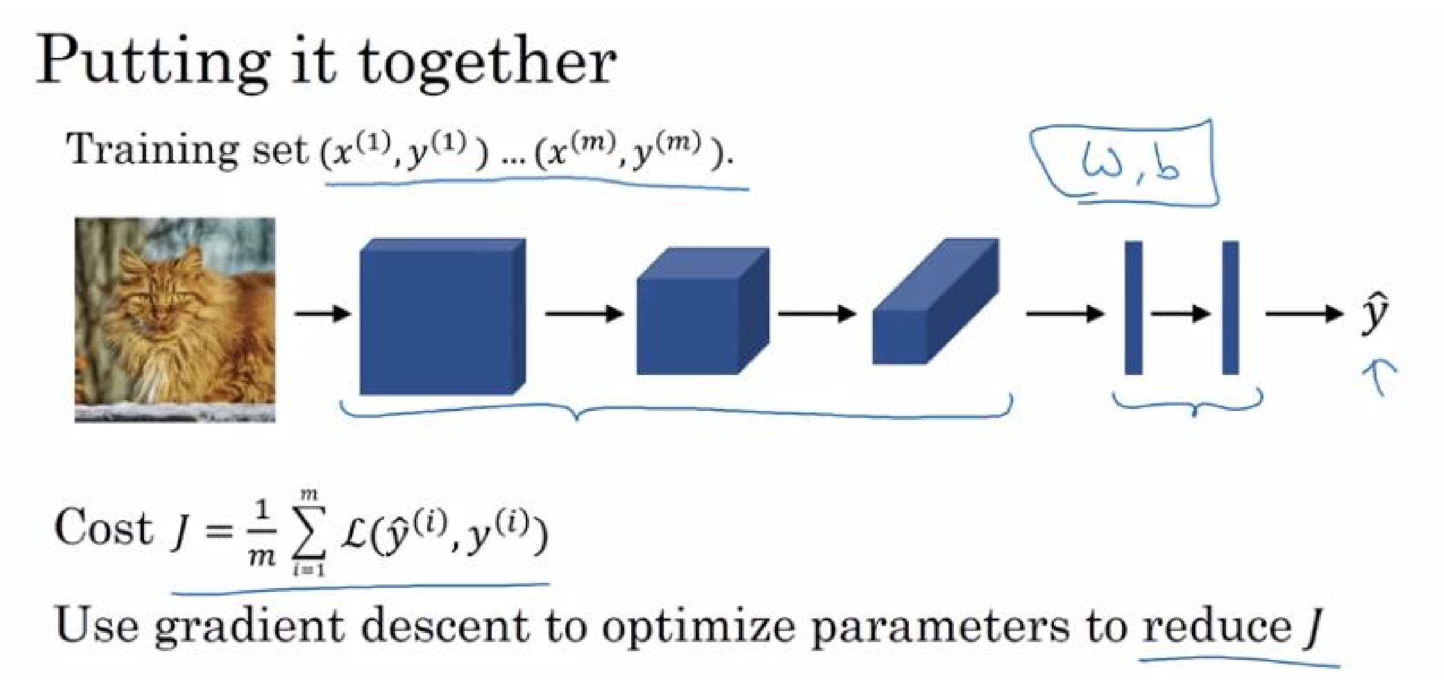
Why convolutions ?
The CNN will help reduce the parameter to estimate from the image for example in the below figure



The # of the weight to estimate in the fully connected layer is 14 million comparing to the 156 parameters in the CONV layer which is much smaller.

The reason of CNN using smaller number than fully connected layer is

1) **Parameter sharing** : a feature detector such as vertical, edge detector that's useful in one part of the image is probably useful in another part of the image
2) **Sparsity of connections** : In each layer, each output value depends only on a small number of inputs. For instance, the feature that we get from the conv network will have a value on the small or importance region of the input image and neglect the non important ones.


Putting it together
Training set $(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$.
W, b
$\hat{y}$
Cost $J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$
Use gradient descent to optimize parameters to reduce J

Case studies - CNN

The papers that are gathered the combination of basic building block of CNN together to make a complex one

1) LENET - 5
2) ALEXNET
3) VGG - 16

RESNET

To pass the activation value from the shallow network through the deeper network with a skip connection

Why RESNET

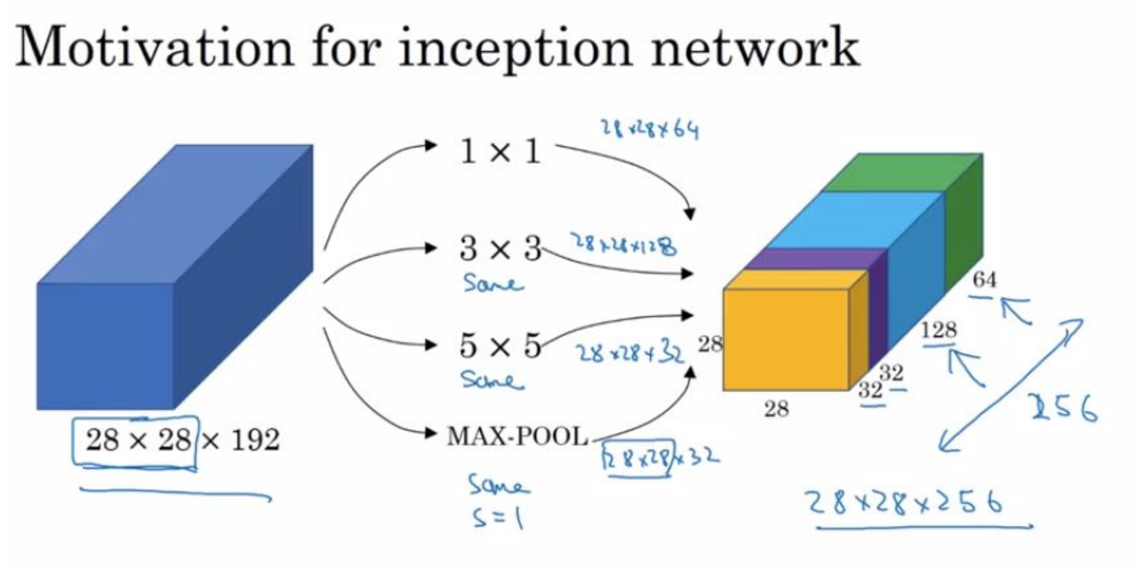
It helps you in training the much deeper neural network while preserving the result of activation function from the shallow layer to be not shiked too much in the deeper one (vanishing gradient)

Sometimes, with luck, you might get a better result by adding this architecture for the much deeper neural network architecture.

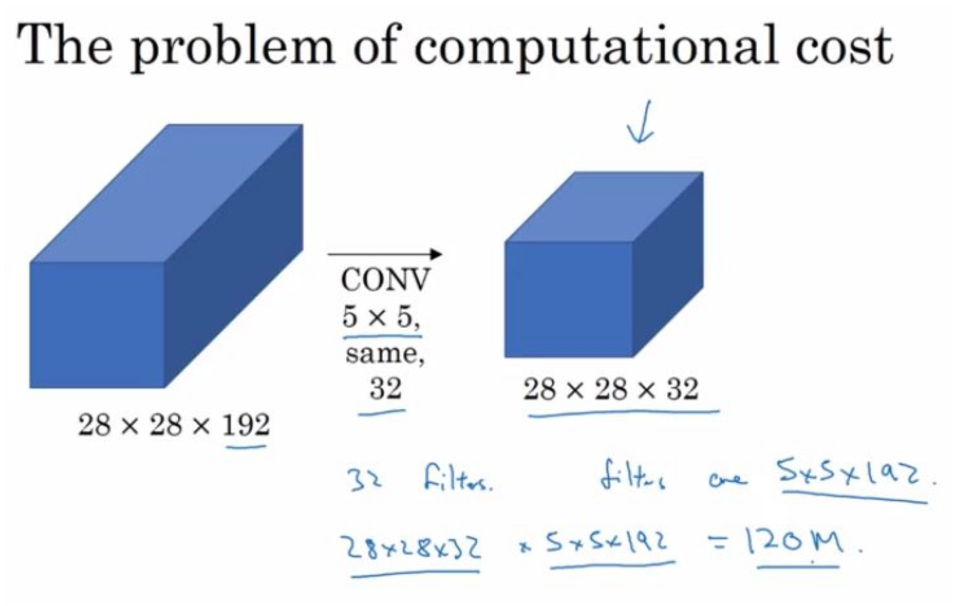
1x1 Convolutional

1) Used when you need to shrink the number of channel to be lower number of channel. (lower computational cost in some layers)
2) Also used in inception network as following section.

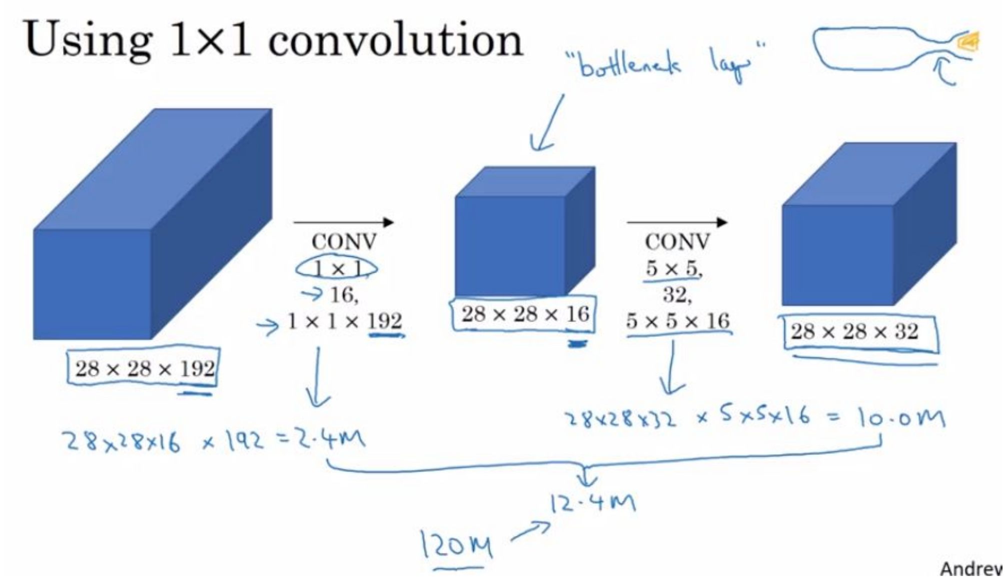
Inception network



The idea is when you would like to pass the image with the multiple filter with the different filter size you will end up like the above figure and then when you finding that how many parameters you have to estimate it's around 120M parameters which are cost so much in computation.



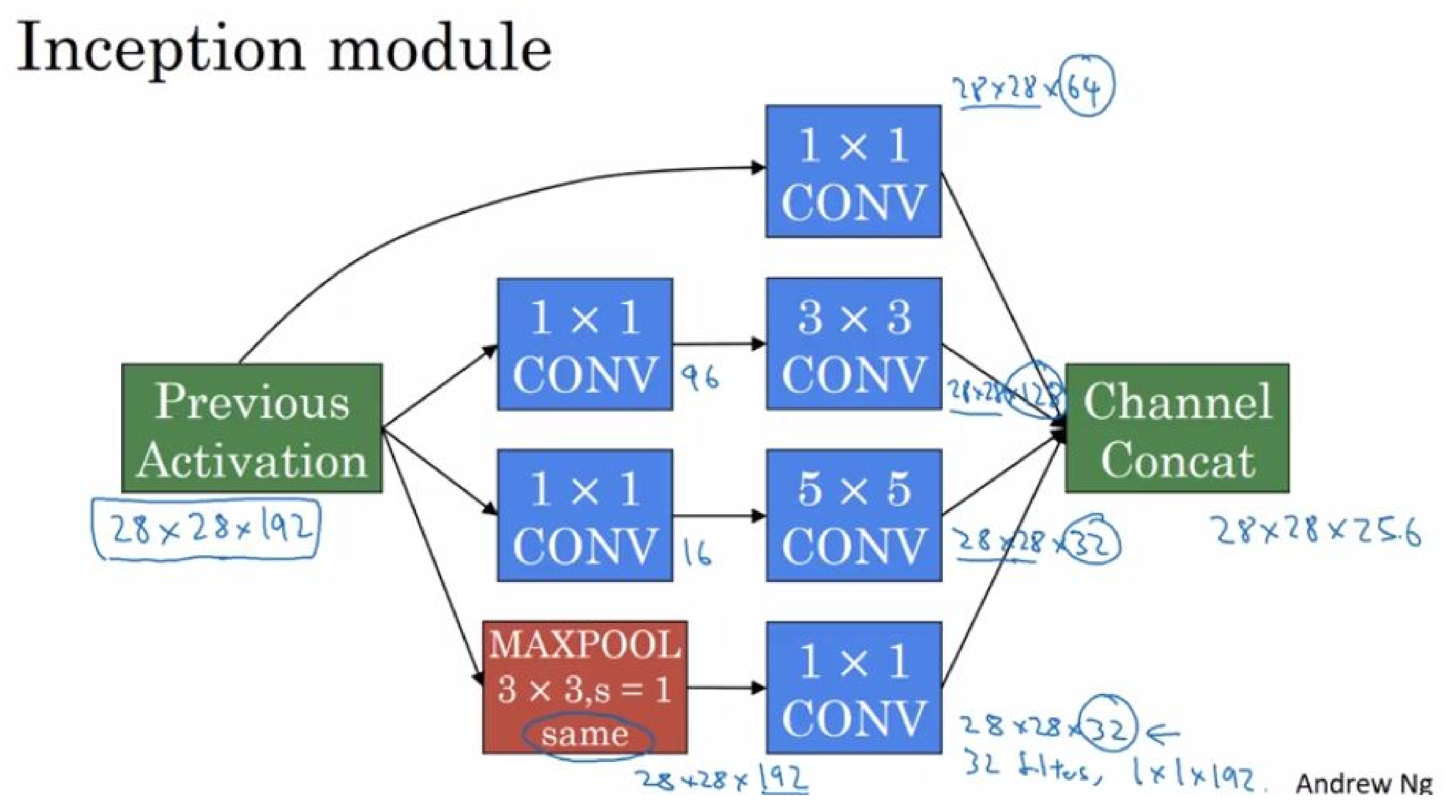
As you can see if you have 28 x 28 x 192 and would like to convolute it to 28, 28, 32 then you have around 120M parameters to estimate at that time.



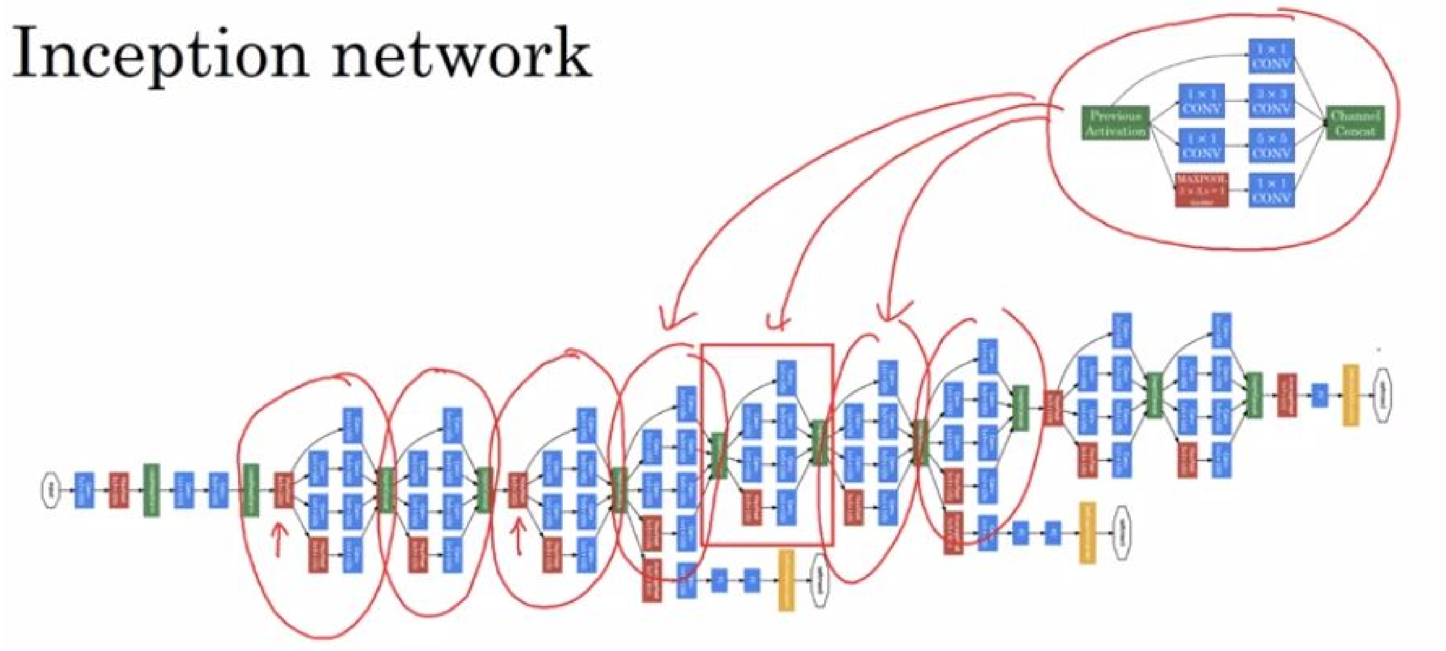
So the idea is you use 1x1 convolution first to shrink the number of channel and then just apply the convolution operation later. From the result, you will reduce the number of estimated parameters from 120 M to 12.4 M with just 1 layer of 1x1.

Why Inception network

The most impressive result is that the shuriken of the dimensional 1x1 doesn't hurt the performance of neural network that much. So it’s worth doing !



This is how you implement the inception model by adding 1x1 convolution in each part of filter



The yellow box is the fully connected layer with the softmax layer that will give you an output from the shallow layers so that you can ensure that the deep neural network is still worked well in the much deeper layer by seeing the result from softmax layer along the architecture.

Another name is “GOOGLENET”

PRACTICAL ADVICES IN CONVNET

There will have topics as following

OPEN SOURCE

There is a lot on GITHUB. You should start cloning it to make your project goes much faster

TRANSFER LEARNING

We can use the pretrained weight from the other people and then tuning the last layer based on your target to make a prediction. This kind of implementation is called transfer learning and it's much working well so many times in this field.

DATA AUGMENTATION

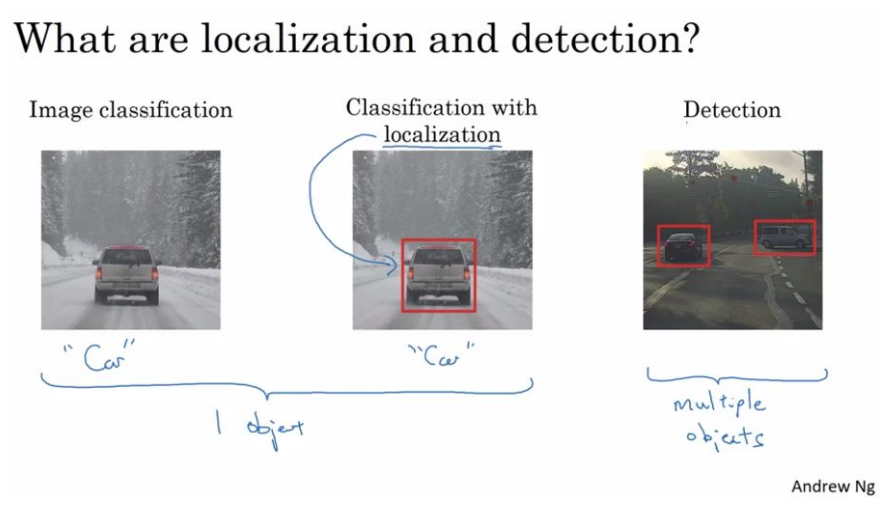
Sometimes, you can augment your data like mirroring, random cropping, and distorting the RGB channel to make an additional data set of your label

STATE OF COMPUTER VISION

It's like if we have a lots of data we can go with the prefered architecture and combination of them with the simpler CNN layer and also tuning the parameters. In the previous year, when the data was still small, the researcher tend to focus more on data labeling and hand-feature engineering but that's not nowadays.

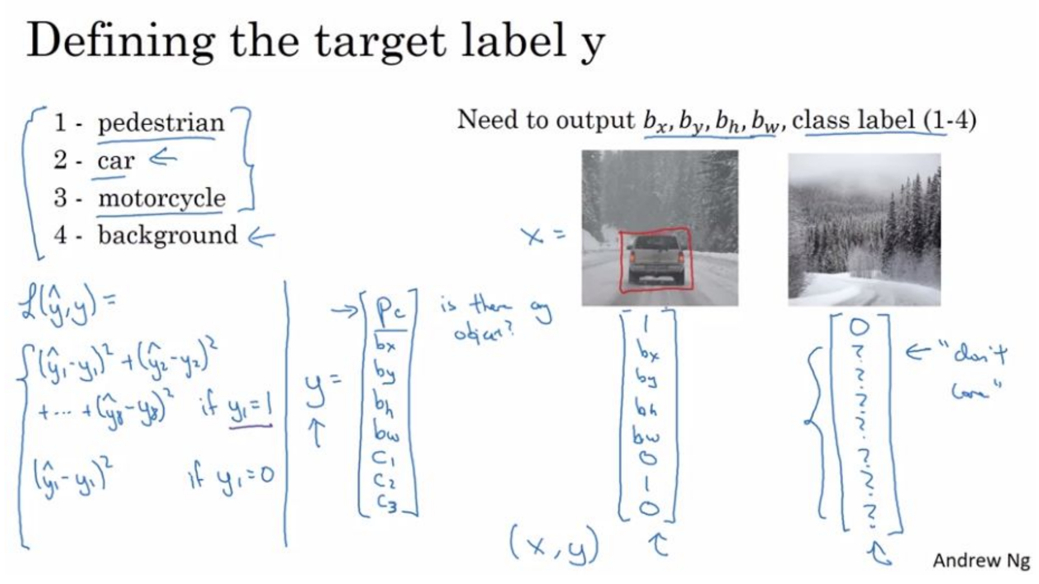
OBJECT DETECTION

Classification with localization



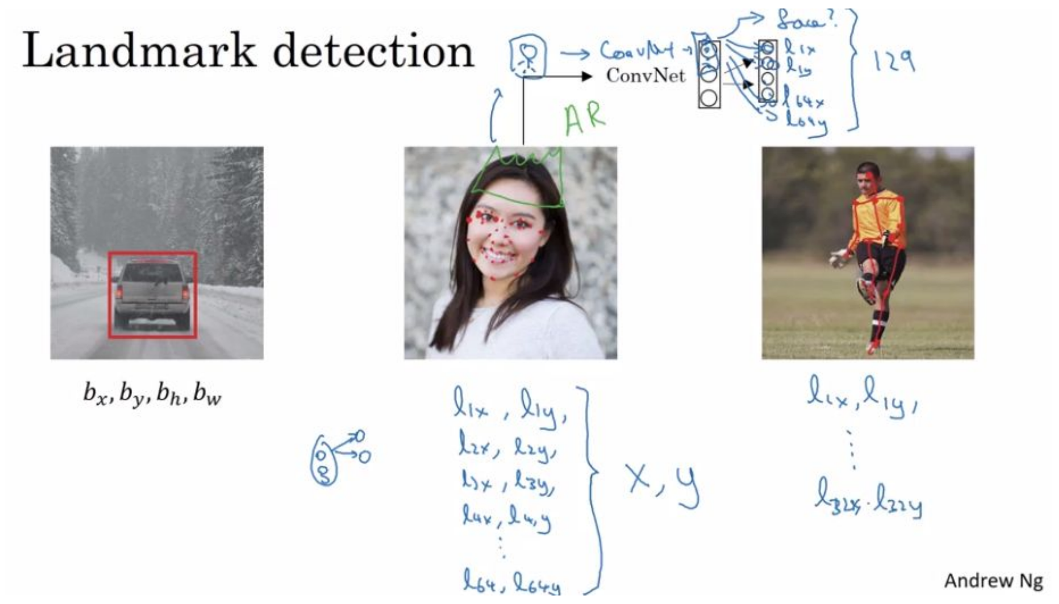
We will be using the softmax with multiclass like 1-pedestrian 2-car 3-motorcycle 4-background as well as concatenating the bounding box in (position x, position y, width, height) the output columns vector.

For output example, the below figure is the example of the classification localization problem output and the loss function to optimize while you are training the model


Defining the target label y
1 - pedestrian
2 - car
3 - motorcycle
4 - background
Need to output b_x, b_y, b_h, b_w, class label (1-4)
x =
$\mathcal{L}(\hat{y}, y) = \begin{cases} (\hat{y}_1-y_1)^2 + (\hat{y}_2-y_2)^2 + \dots + (\hat{y}_8-y_8)^2 & \text{if } y_1 = 1 \\ (\hat{y}_1-y_1)^2 & \text{if } y_1 = 0 \end{cases}$
$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$
is there any object?
$\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$
$\begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$
"don't care"
(x, y)
Andrew Ng

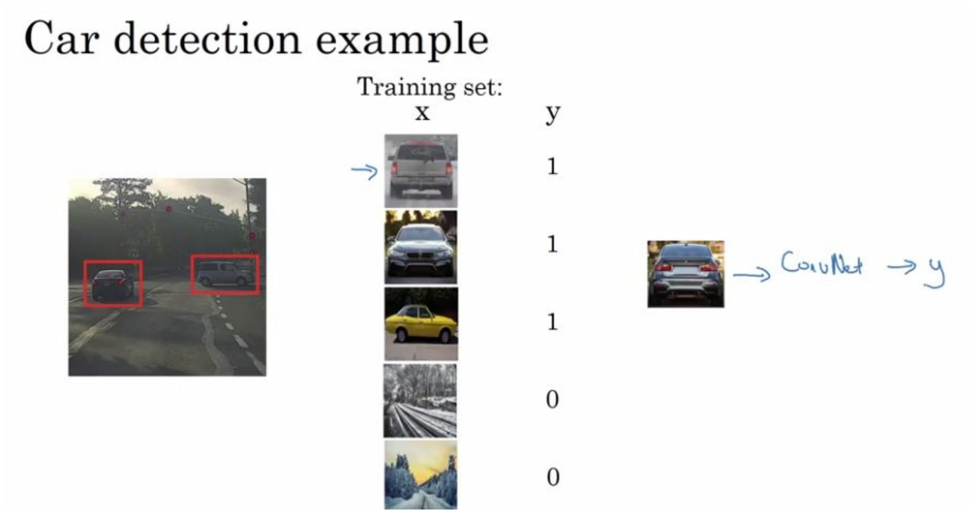
LANDMARK DETECTION

The detail (landmark) in each picture that you want to detect is called landmark (e.g. eyebrows, eyes, mouth, nose on your face). The output will be something like this

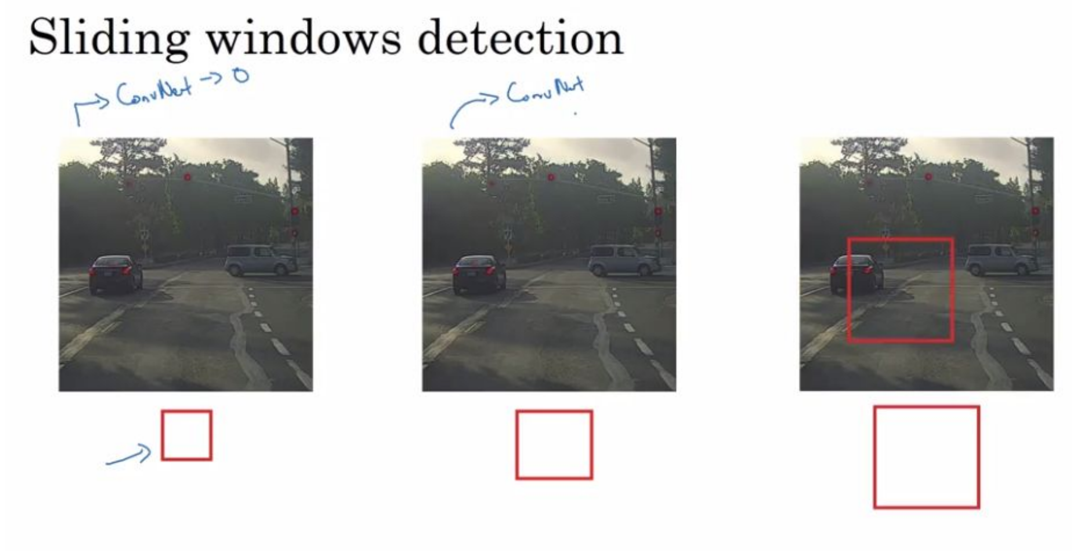


For 1 pictures, it will have is_face_arrived, landmark1 x, landmark1 y , … , landmark64 x, landmark64 y (so the total output parameters is 64*2 + 1 = 129 parameters)

OBJECT DETECTION IN PRACTICE

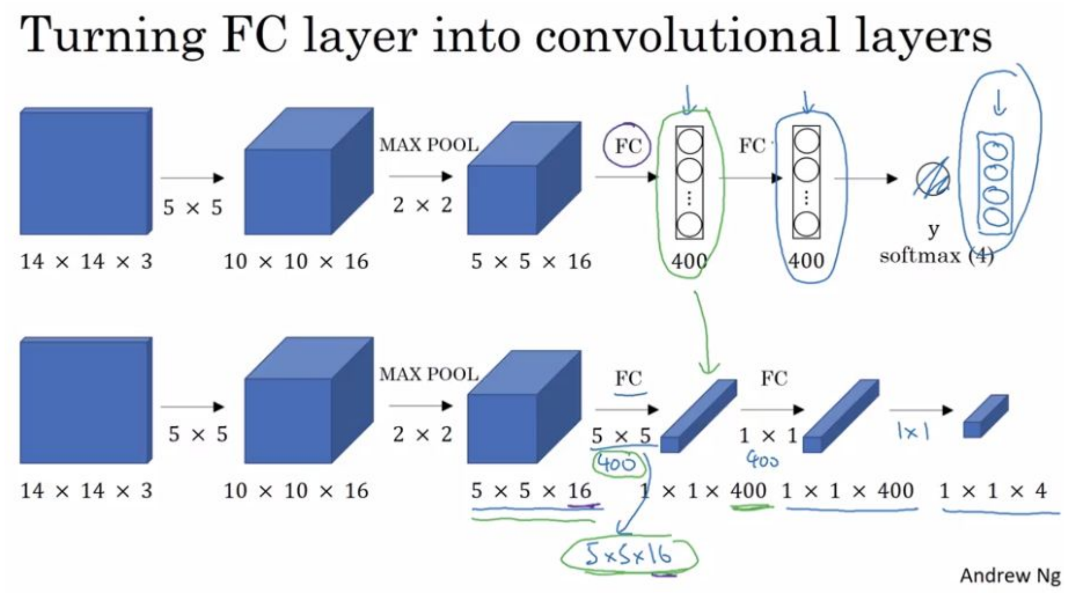


We can feed the ConvNet with this kind of training set and then using the sliding windows detection to get the prediction of each windows in the image. You can vary the size of windows and # of stride for your project.

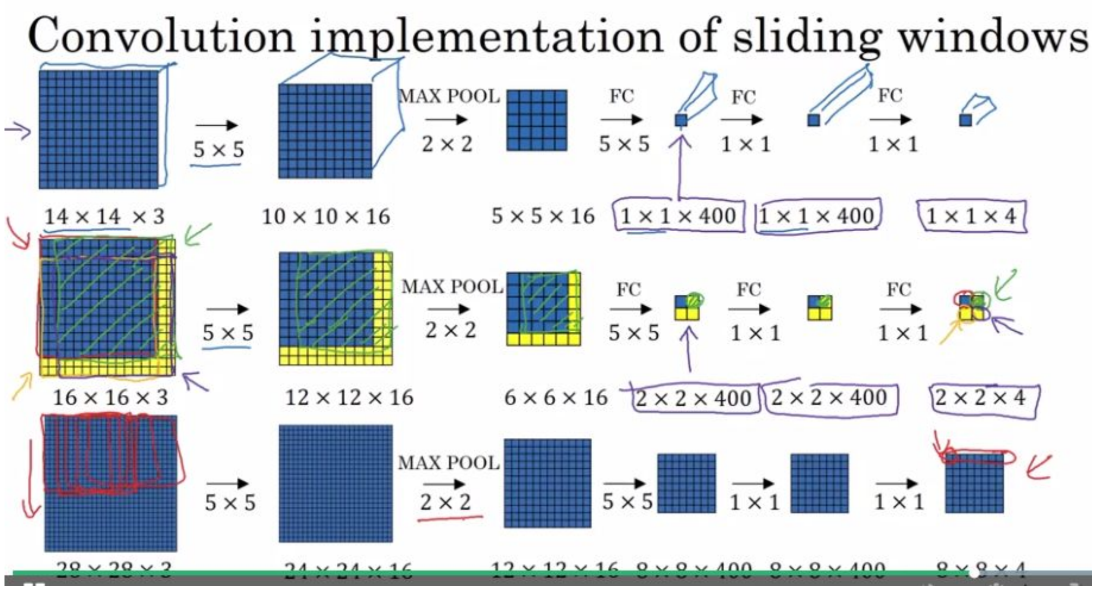


If the windows is too large it might be hurt the performance of the model. But with the fine grain windows it would be take so much computational cost > SO THEY FIX IT BY CONVOLUTIONAL SLIDING WINDOWS TECHNIQUES.

We can transform the fully connected layer to convolutional implementation by using the normal filter and 1x1 convolutional instead



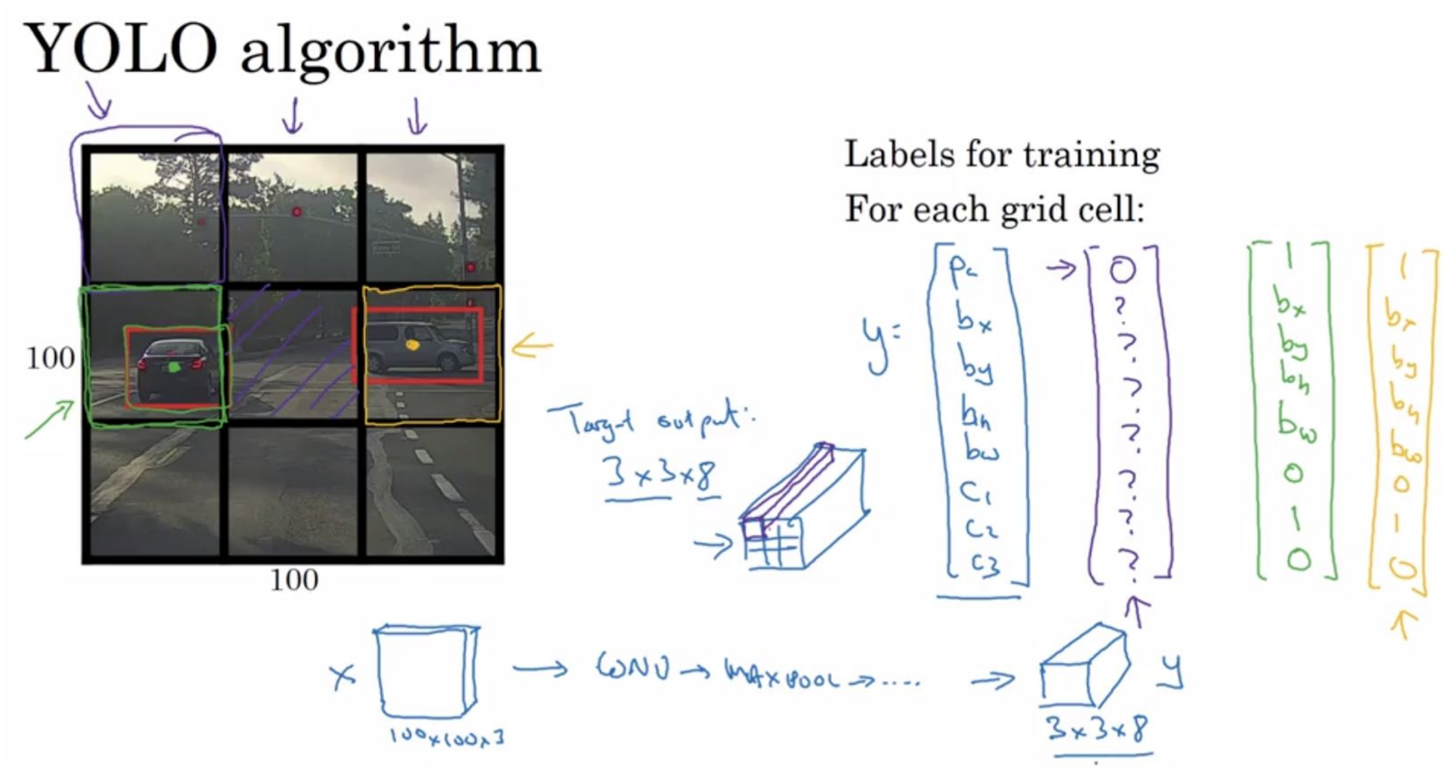
Then if you have some yellow pad then you got the all result of the sliding windows at once! The stride of max pooling is the same as the string of sliding window.



The Pros is that you got the result of all sliding windows in the one time. But there is still a problem with bounding box accuracy > this problem will be fixed by

BOUNDING PREDICTION

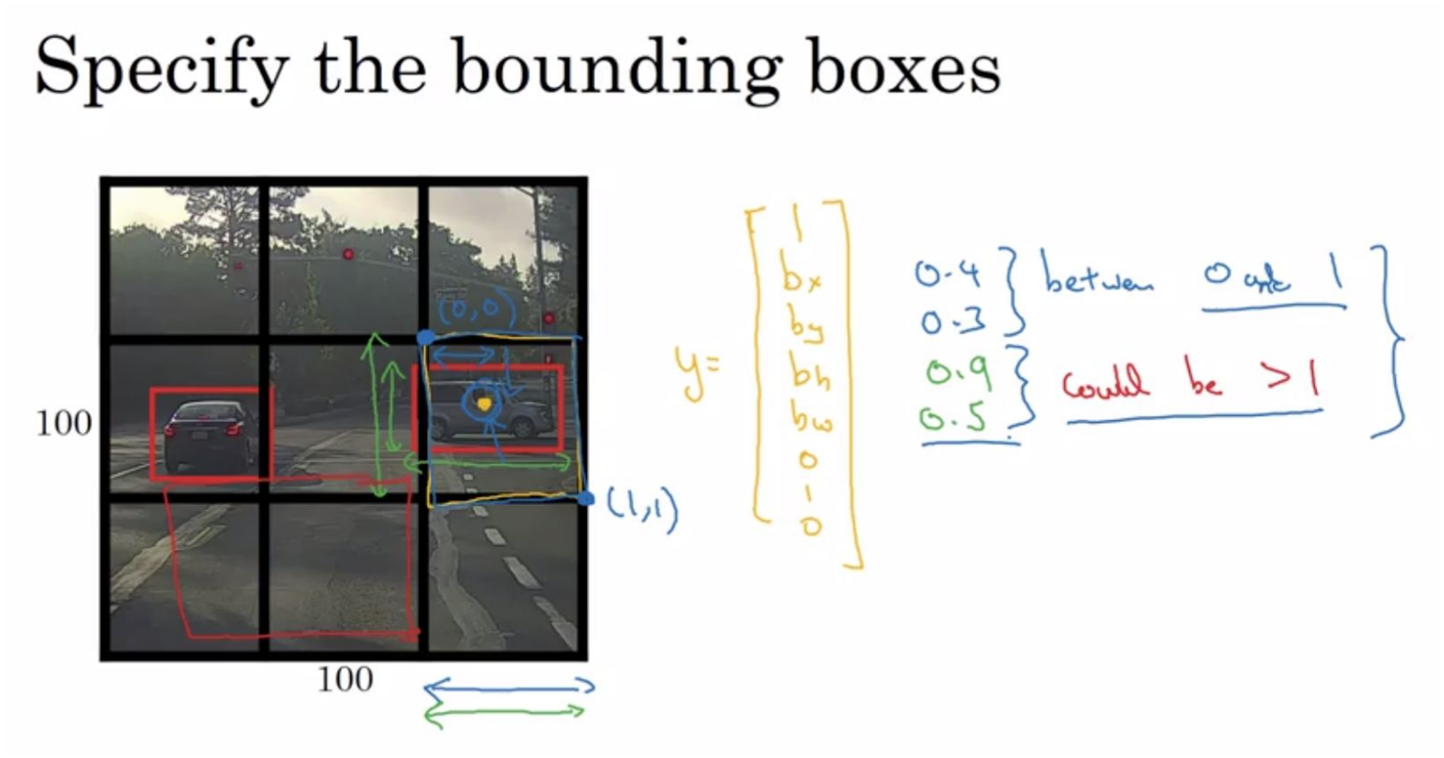
There are a YOLO (you only look once) algorithms that is made to solve this problem specifically. The idea is that there separates the image into n x n grid and then in each grid it will have a training label like the right side of the figure and the output of this algorithms will be n x n x 8 (8 comes from the dimension of the columns vector)



The advantage of this approach is we will got the more precise of bounding box because it's only assign to each subset grid cell instead of the whole picture. Secondly, it's convolutional implementation so it's much faster and can be used with real time objection detection

Be reminded each object will be assign to 1 grid cell based on the midpoint of that object. Also with the more fine grain gridcell the possibility that the object will be fall to only 1 grid cell is higher.

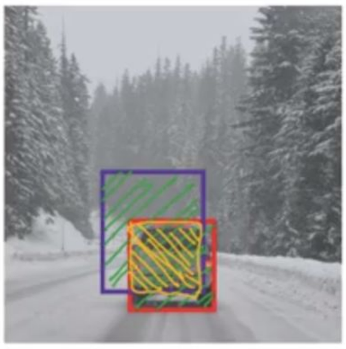
The logic for assigning the bounding box in YOLO is by referencing the bounding box grid cell origin and assign the the bx, by, bh, bw of that grid cell.



INTERSECTION OVER UNION

To help your algorithm even better, these are the measured of how good the bounding box is in the object detection.

Evaluating object localization



Intersection over Union (IoU)

$$= \frac{\text{size of } 🟨}{\text{size of } 🟩}$$

“Correct” if IoU $\geq \boxed{0.5}$ ←

0.6 ←

More generally, IoU is a measure of the overlap between two bounding boxes.

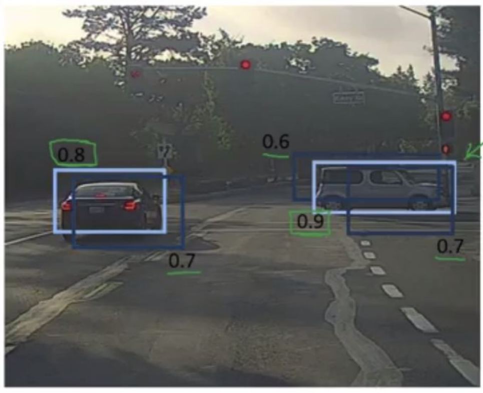
Andrew Ng

You can change the threshold whatever you want.

NON MAX SUPPRESSION

To ensure that your algorithms detects the object only one times we can use this method to help. The non max suppression will clean up multiple bounding box to get the right one by taking into account of the probability of the object appeared and **selected the one with highest probability and the highest IOU and suppress the other bounding boxes.**

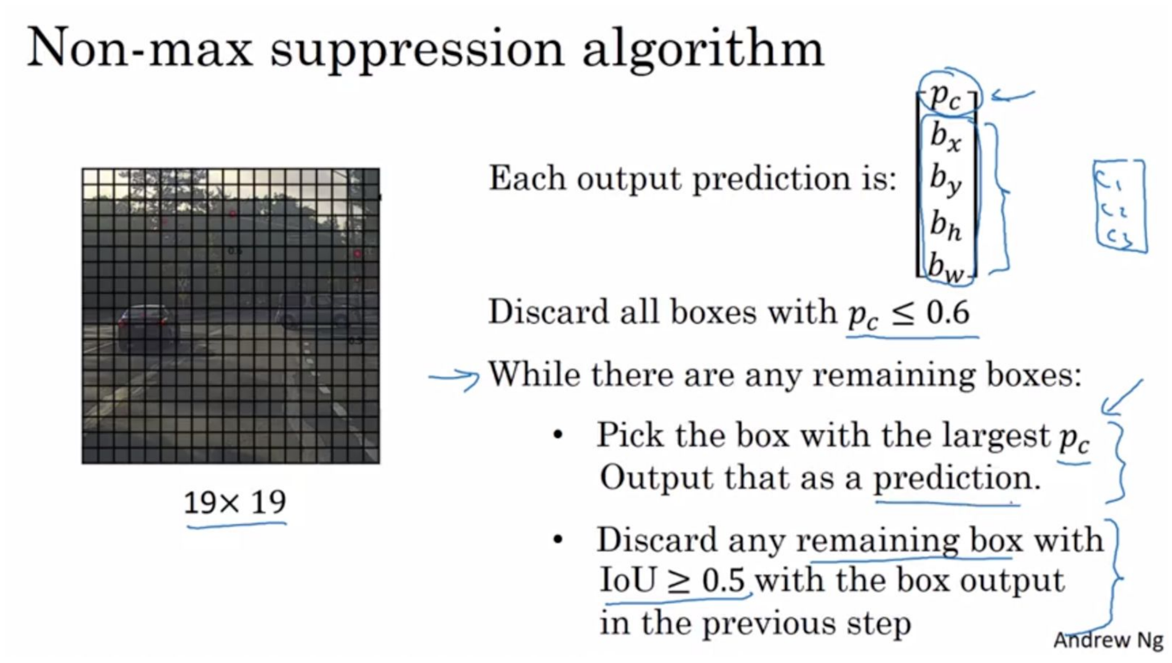
Non-max suppression example



P_c

Andrew Ng

For summary on the non max suppression, the output vectors can change if there have a multiple objects in the image by running non max suppression C times where C is the number of class object.



Non-max suppression algorithm

Each output prediction is: $\begin{bmatrix}p_c\\b_x\\b_y\\b_h\\b_w\end{bmatrix}$ $\begin{bmatrix}c_1\\c_2\\c_3\end{bmatrix}$

19× 19

Discard all boxes with $p_c \leq 0.6$

While there are any remaining boxes:

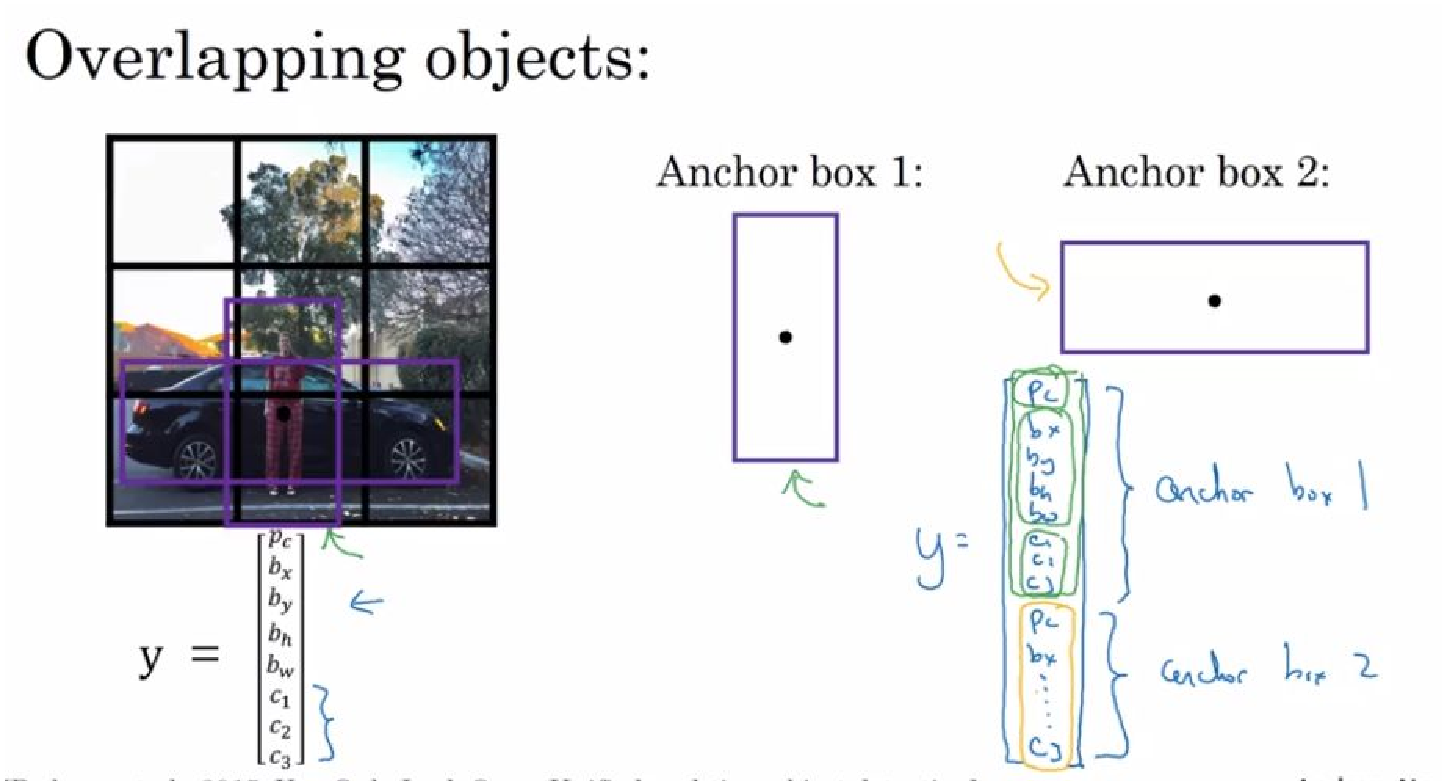
- Pick the box with the largest p_c Output that as a prediction.
- Discard any remaining box with IoU ≥ 0.5 with the box output in the previous step

Andrew Ng

ANCHOR BOXES (FOR MULTIPLE OBJECT)

If 1 grid cell is belong to more than 1 object then we will create the anchor boxes. For the anchor boxes each of boxes will be represented to the 1 object and then for that grid cell we will concatenating the result of 2 anchors boxes into the output of that gridcell.

Overlapping objects:

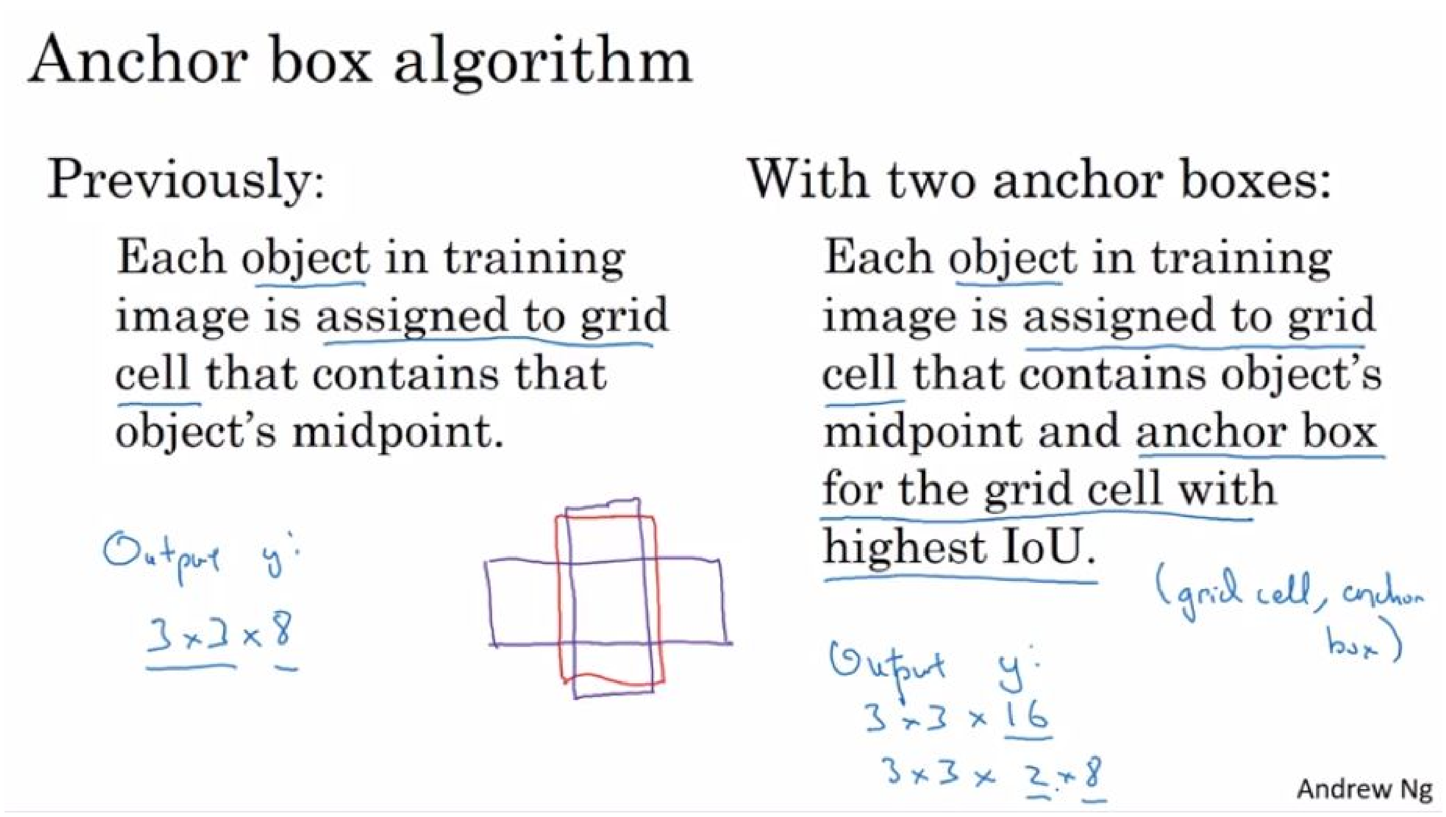


$$y = \begin{bmatrix}p_c\\b_x\\b_y\\b_h\\b_w\\c_1\\c_2\\c_3\end{bmatrix}$$

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

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

Andrew Ng



Anchor box algorithm

Previously:

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

Output y:

$3 \times 3 \times 8$

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.

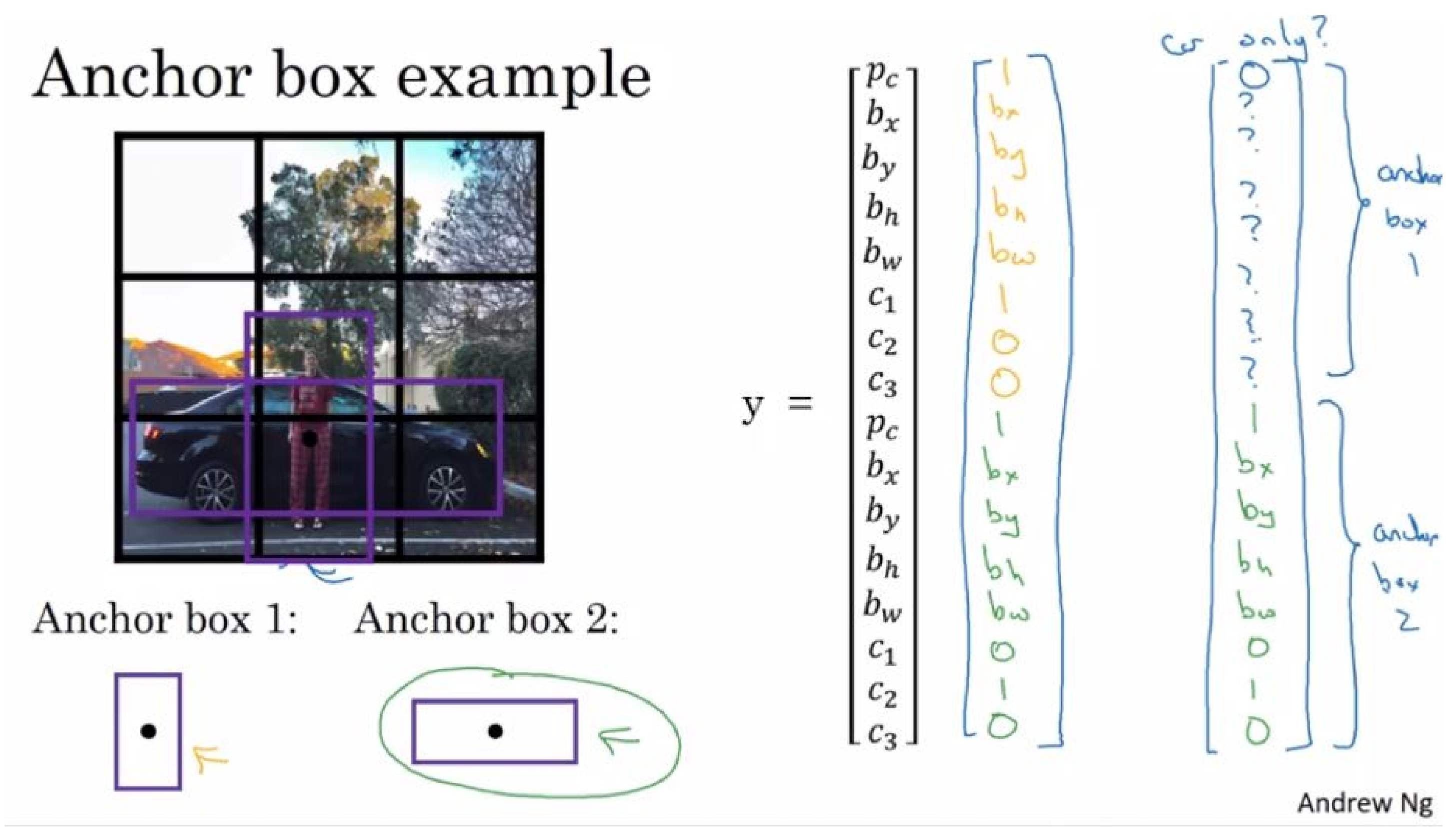
(grid cell, anchor box)

Output y:

$3 \times 3 \times 16$

$3 \times 3 \times 2 \times 8$

Andrew Ng



Anchor box example

Anchor box 1: Anchor box 2:

car only?

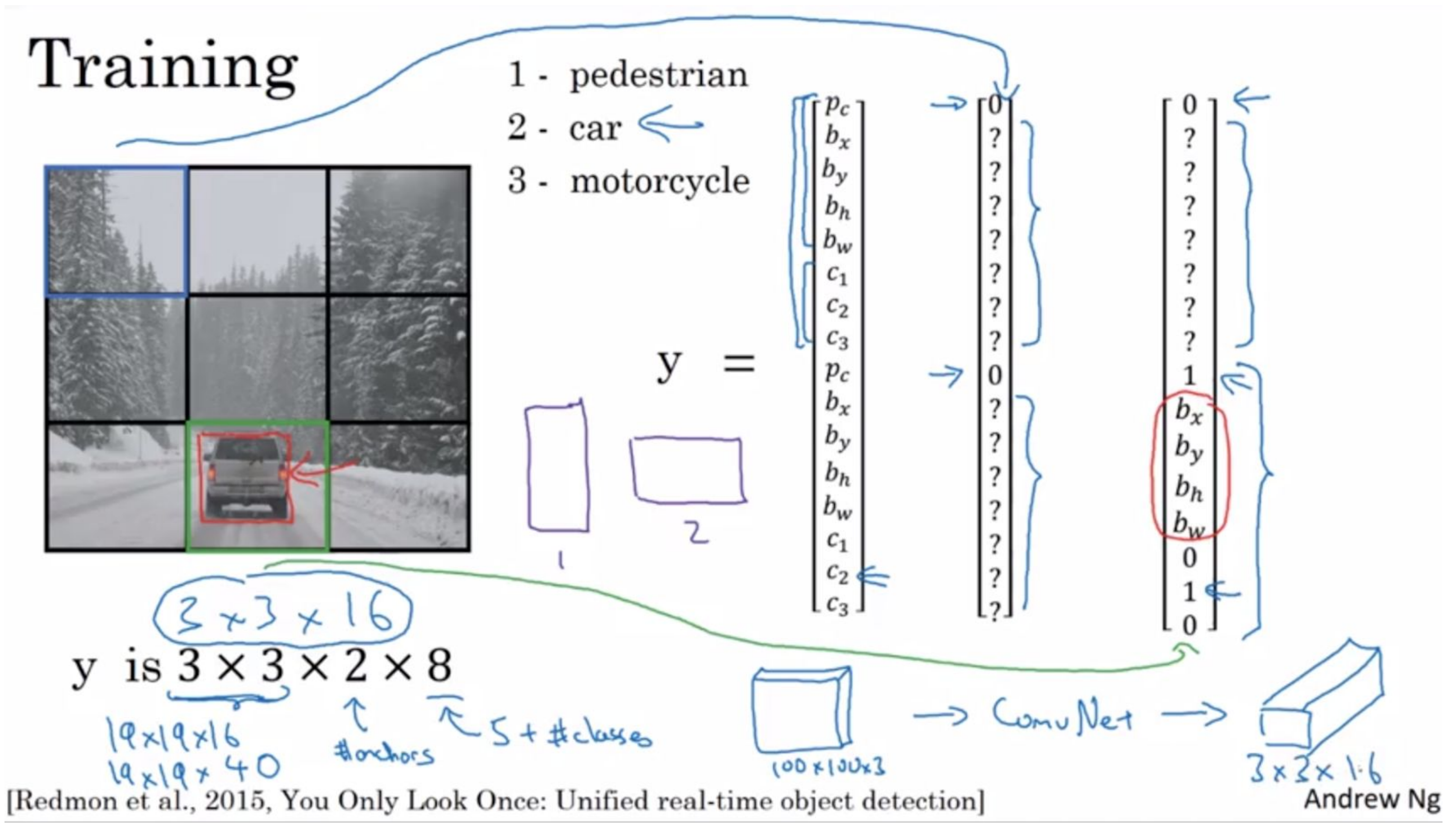
$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \\ 0 \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}1\\1\\1\\1\\1\\1\\1\\1\end{matrix}}\right\} \text{anchor box 1} \\ \left.\vphantom{\begin{matrix}1\\1\\1\\1\\1\\1\\1\\1\end{matrix}}\right\} \text{anchor box 2} \end{matrix}$$

Andrew Ng

This algorithm doesn't work well if there is 2 objects with the same kind of anchor box in the same grid cell.

The choice of anchor boxes shape is on your deciding. Usually 5-10 shapes of anchor boxes apply to the real world problem

PUT IT ALL TOGETHER IN YOLO



Training

1 - pedestrian
2 - car
3 - motorcycle

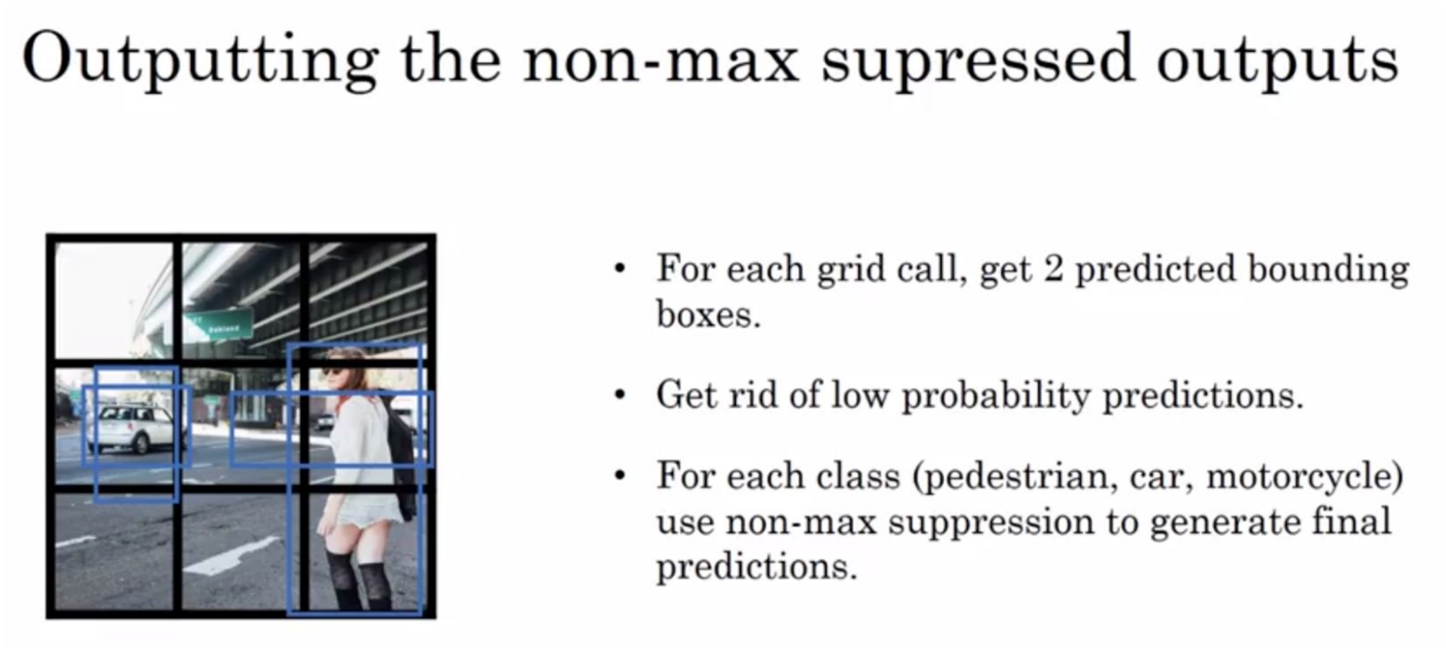
$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \quad \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ?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

y is $3 \times 3 \times 2 \times 8$

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

Andrew Ng

Because the car has the IOU with anchor box 2 > 1 then the result will put in the anchor box 2. Then run this result through the non max suppression

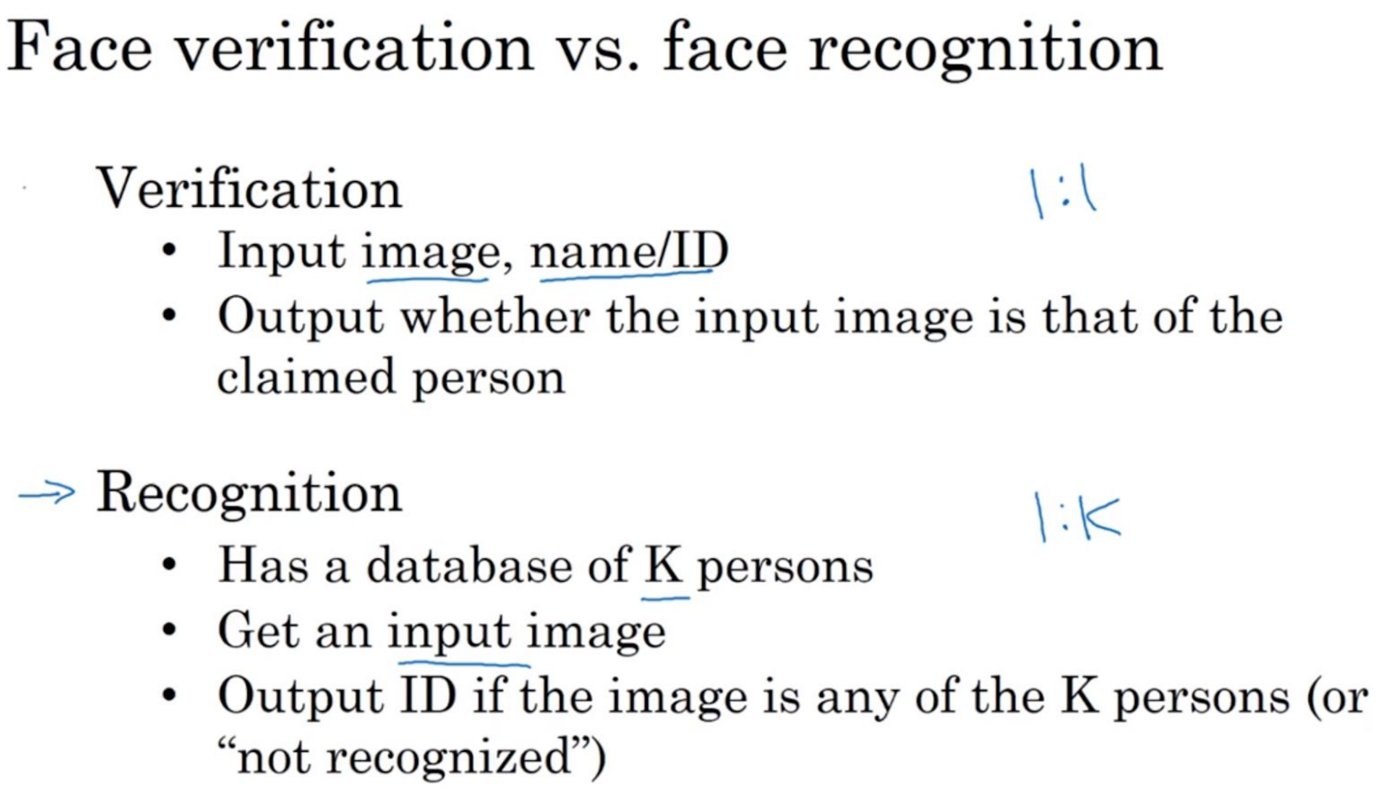


Outputting the non-max supressed outputs

- For each grid call, 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.

FACE RECOGNITION

Also liveness detection (how the system detecting that the object is live or non live)



Face verification vs. face recognition

Verification 1:1

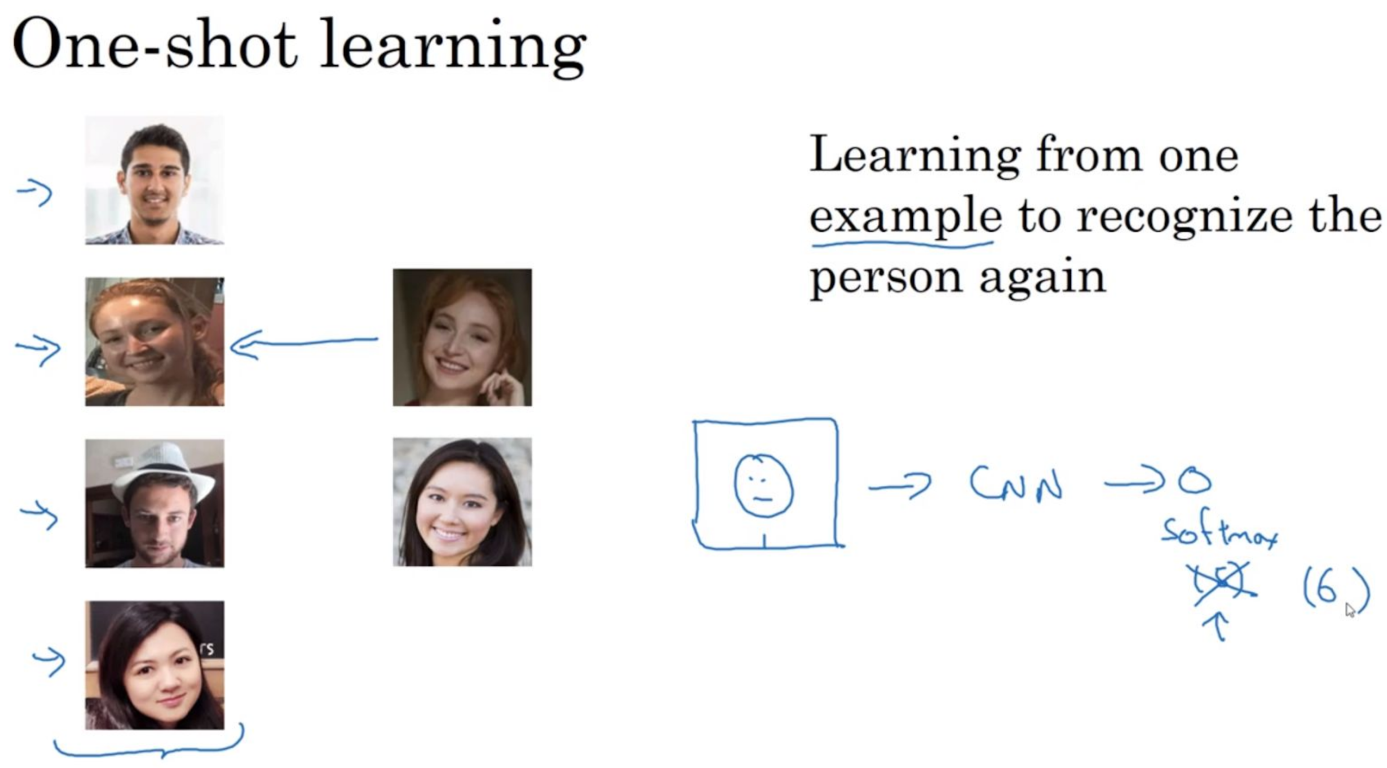
- Input image, name/ID
- Output whether the input image is that of the claimed person

→ Recognition 1:K

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

We have to make a good face verification first then put it in the Recognition system to make a better system.

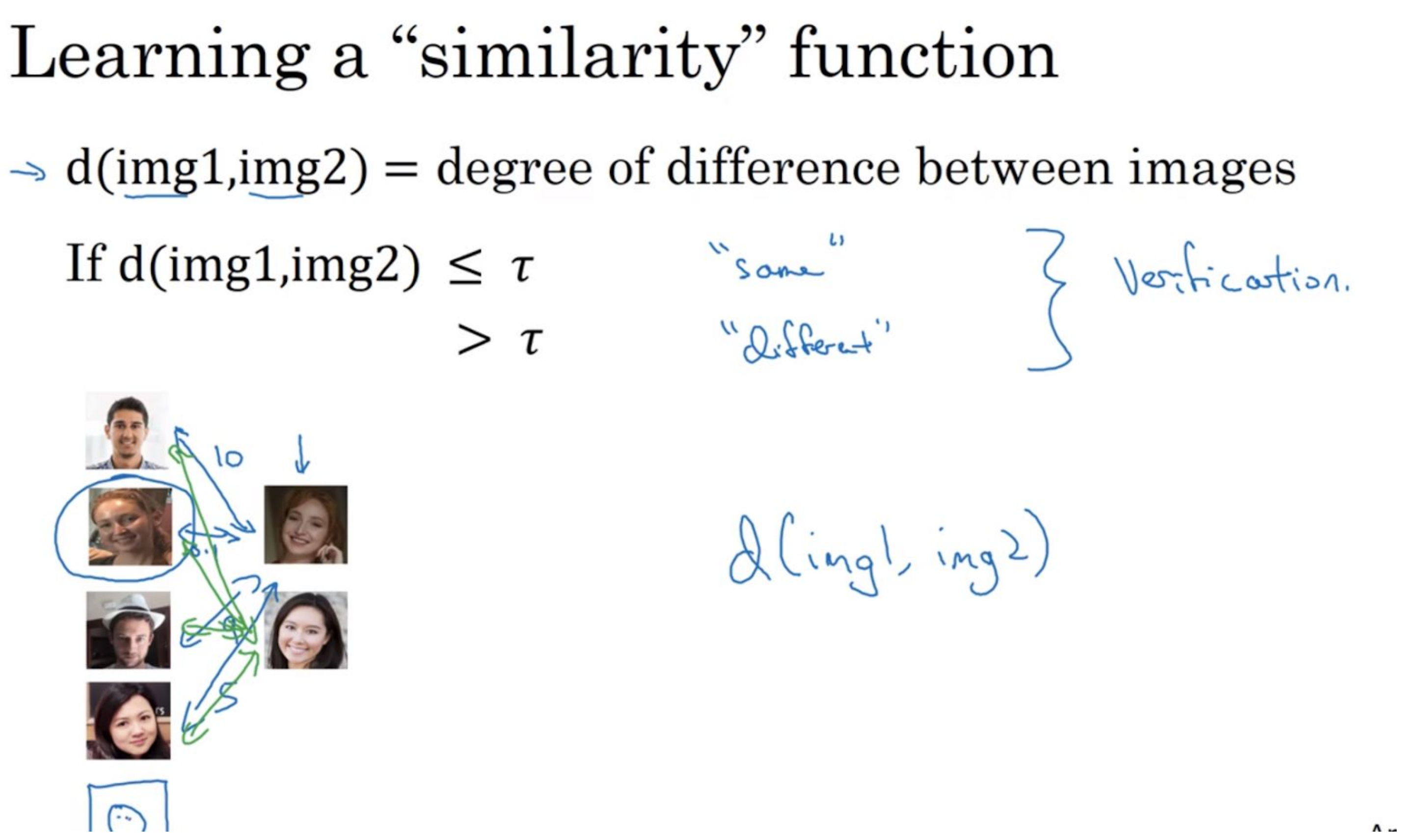
ONE SHOT LEARNING



One-shot learning

Learning from one example to recognize the person again

The drawback of one shot learning is that if the training example is too small and if you have a new person comes into the database you have to retrain the CONVNET model again every time. So to solve this problem we should try to learning the "similarity" of the image instead by



Learning a "similarity" function

→ d(img1,img2) = degree of difference between images

If d(img1,img2) $\leq \tau$ "same"

$> \tau$ "different"

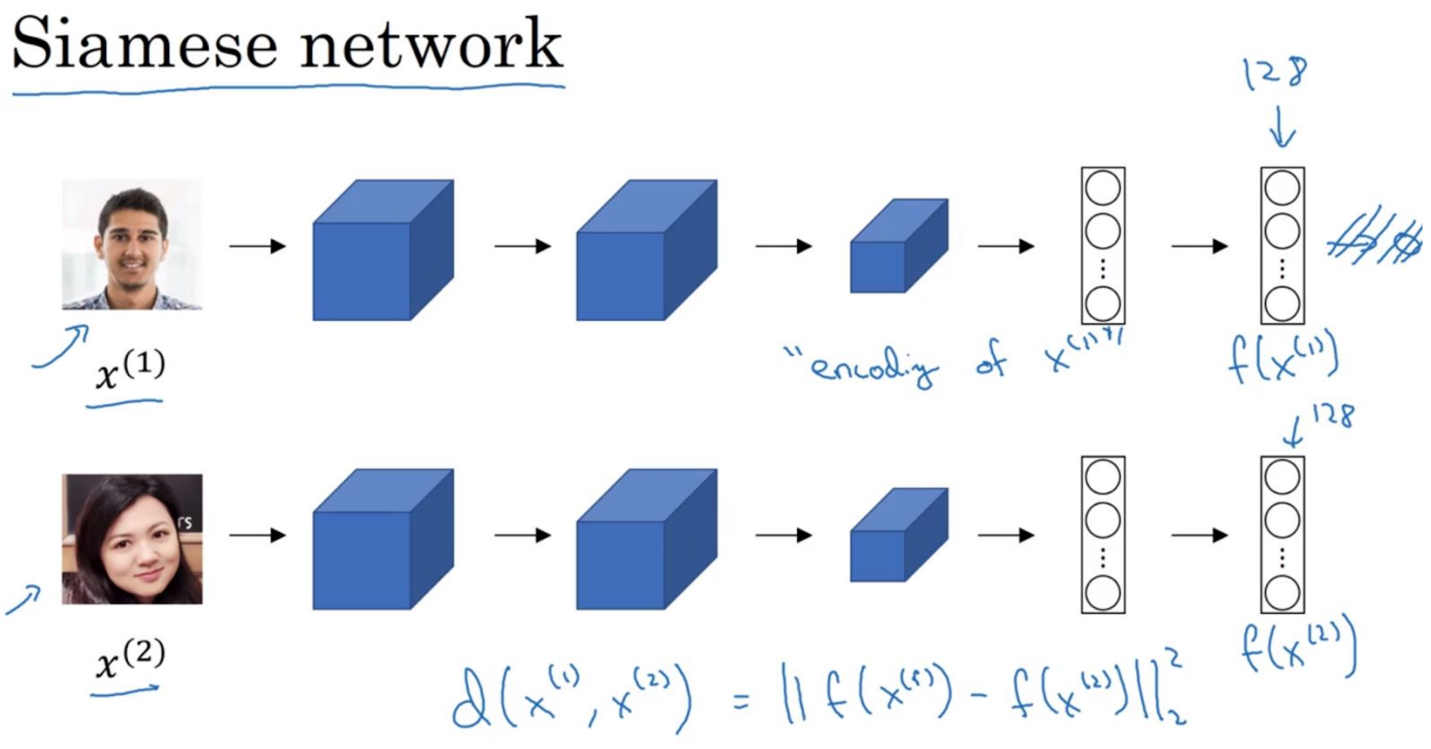
} Verification.

If the d(dissimilarity) is high then there tend to not be the same person. As well as you can just add the new image to the database without retrain the model.

SIAMESE NETWORK

The algorithme bring the last layer from fully connected layer and give it a name like encoding of training sample (1). Then make the same with the training sample (2) with the same NN and hyperparameters so you got encoding of training sample (2).

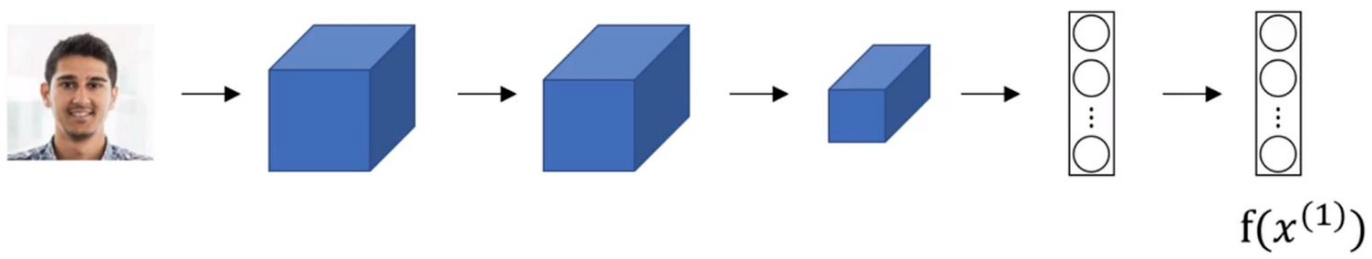
You find the similarity between these 2 encoding d(X(1), X(2)) = || f(X(1) - fX(2) || ^2 (norm of the encoding 1 and 2)



Siamese network

$$d(x^{(1)}, x^{(2)}) = \| f(x^{(1)}) - f(x^{(2)}) \|_2^2$$

Goal of learning



Parameters of NN define an encoding $f(x^{(i)})$ — 128

Learn parameters so that:

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

If $x^{(i)}, x^{(j)}$ are different persons, $\left\|f(x^{(i)}) - f(x^{(j)})\right\|^2$ is large.

You tune the parameters until there satisfy the above condition.

TRIPLET LOSS FUNCTION

The loss function to minimize the dissimilarity of the same person and maximize the dissimilarity the others way around

Learning Objective

A P $d(A,P) = 0.5$ $\nearrow 0.2$

A N $d(A,N) = \cancel{0.51}$ 0.7

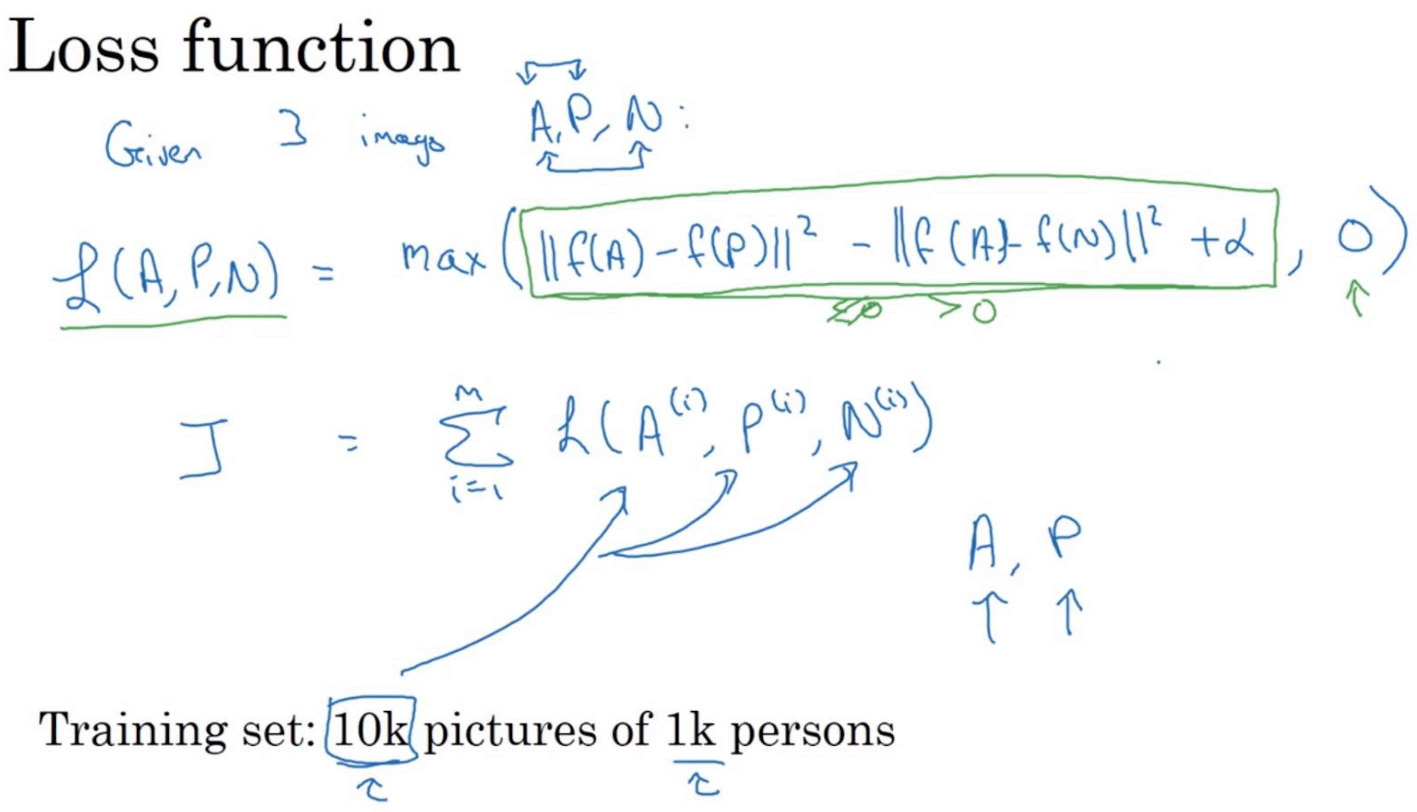
Want: $\underbrace{\| f(A) - f(P) \|^2}_{d(A,P)} + \alpha \le \underbrace{\| f(A) - f(N) \|^2}_{d(A,N)}$

$$\boxed{\underbrace{\|f(A)-f(P)\|^2}_{0} - \underbrace{\|f(A)-f(N)\|^2}_{0} + \underbrace{\alpha}_{\text{margin}} \le 0}$$ [illegible] $f(\text{img}) = \vec{0}$

The loss function will be the below figure. If the CONVNET can satisfy the criteria that the dissimilarity of same person compared to other people is lower than the alpha(margin) then the loss function of that example is 0 and if it’s not satisfy then the loss function of that example will be positive.

Also you have to had a lot of picture for 1 person in the training set so that we can make a comparison of the triplet loss function between ANCHOR, POSITIVE and NEGATIVE.

But when you deploy the model it’s not necessary to have a multiple picture of the target data to make a prediction.



Loss function

Given 3 images A, P, N:

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

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

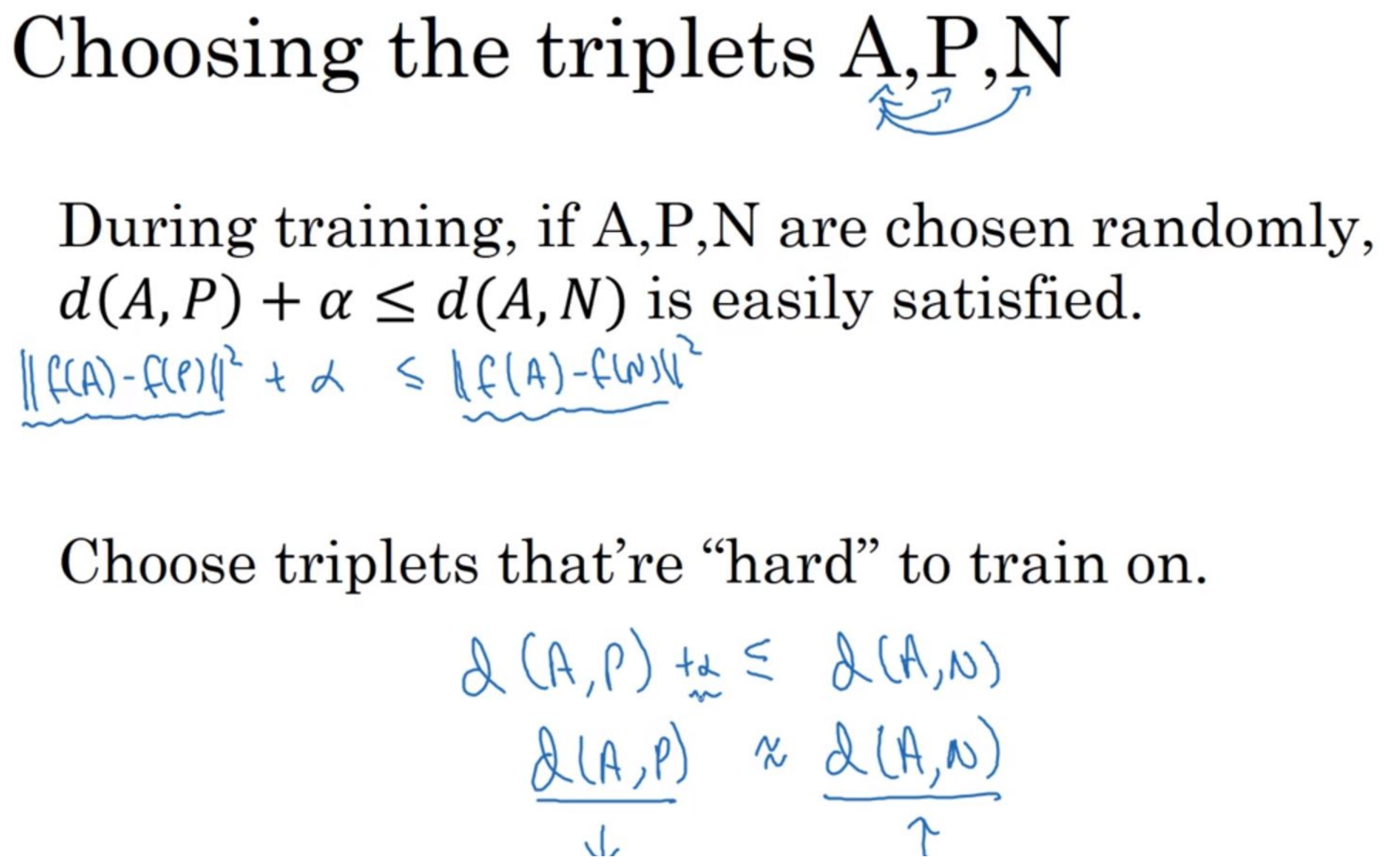
A, P

$\uparrow \ \uparrow$

Training set: $\boxed{10k}$ pictures of $\underline{1k}$ persons

Usually when you are select the training set if you randomly assign the A,P,N then the CONVNET won’t do so much in adjusting the weight because it’s always easier to satisfy the loss function (guessing that there are the different person from the random data).

As a best practice then you should use the triplets that are hard to train on so that the gradient descent have to work hard for adjusting the CONVNET weight. (by assigning d(A,P) ~ d(A,N)



Choosing the triplets A,P,N

During training, if A,P,N are chosen randomly, $d(A,P) + \alpha \leq d(A,N)$ is easily satisfied.

$$\|f(A)-f(P)\|^2 + \alpha \leq \|f(A)-f(N)\|^2$$

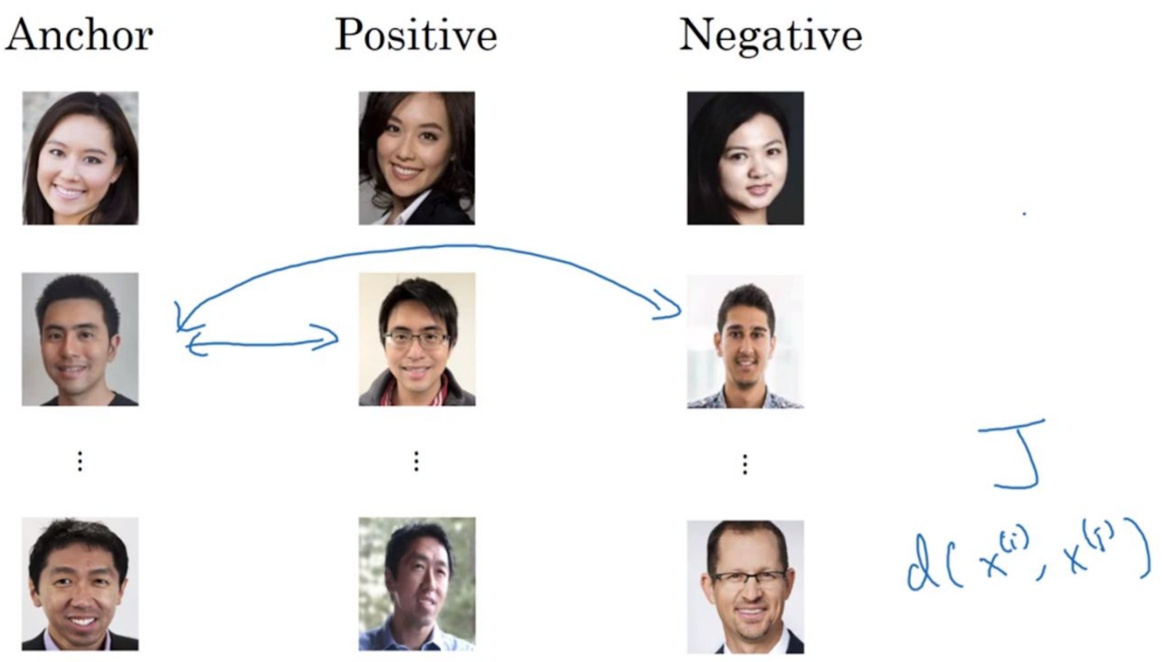
Choose triplets that’re “hard” to train on.

$$d(A,P) + \alpha \leq d(A,N)$$

$$\underline{d(A,P)} \approx \underline{d(A,N)}$$

$\downarrow \qquad \uparrow$

Training set using triplet loss



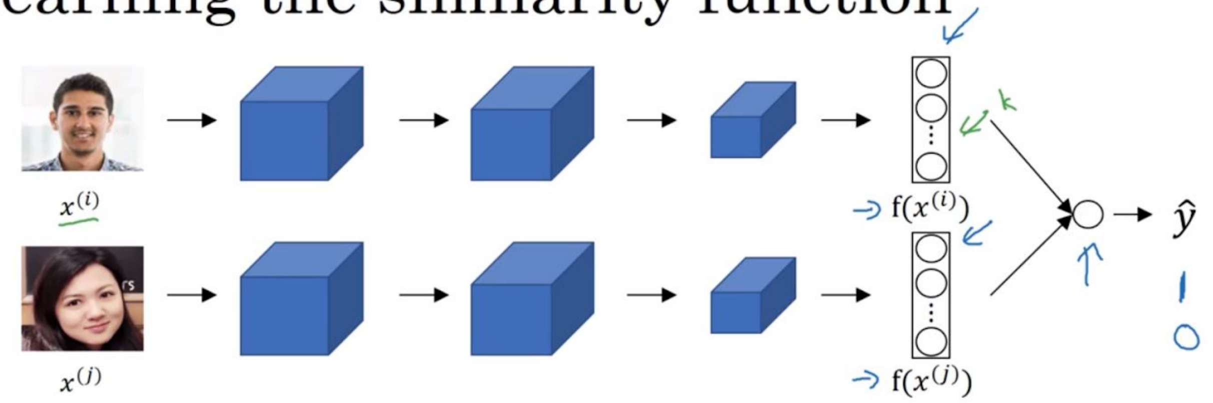
$d(x^{(i)}, x^{(j)})$

For the commercial package, the data set is around 10M data set.

FACE VERIFICATION AND BINARY CLASSIFICATION

To put the above encoding feature to the binary classification.

Learning the similarity function



$$\hat{y} = \sigma\left(\sum_{k=1}^{128} w_i \underbrace{\left|f(x^{(i)})_k - f(x^{(j)})_k\right|}_{\frac{(f(x^{(i)})_k - f(x^{(j)})_k)^2}{f(x^{(i)})_k + f(x^{(j)})_k}} + b\right)$$

χ^2

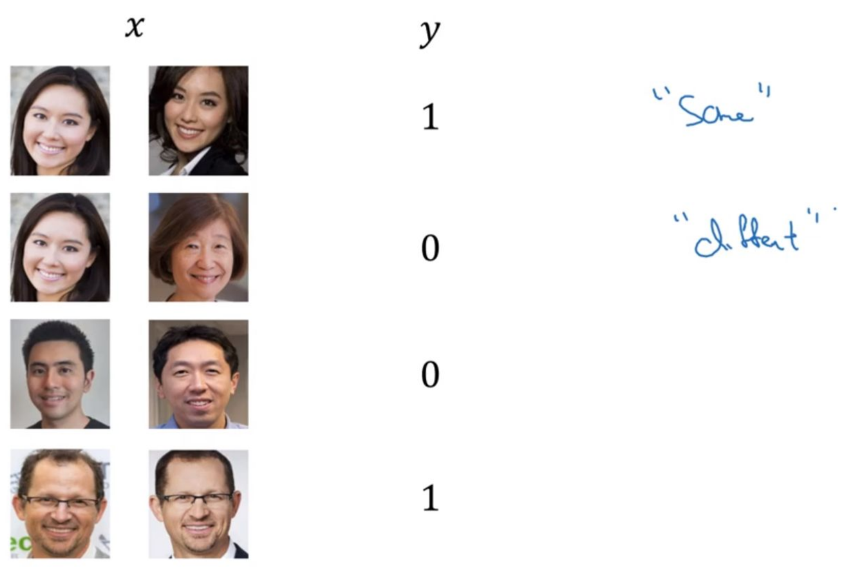
Taigman et. al., 2014. DeepFace closing the gap to human level performance]

Andrew Ng

We can use the element-wise difference between training example or use the chi-square similarity in order to get the weight in sigmoid function to predict whether there are the same person or not.

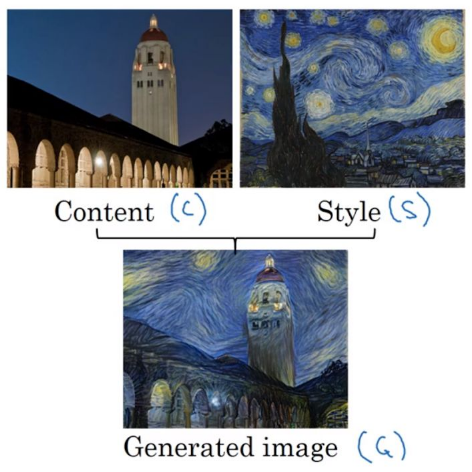
One production trick is that you can compute the encoding features of each face first then storing it in the database for comparison in the next time.

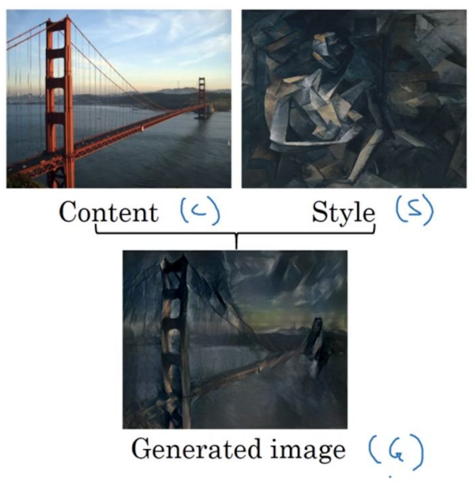
Face verification supervised learning



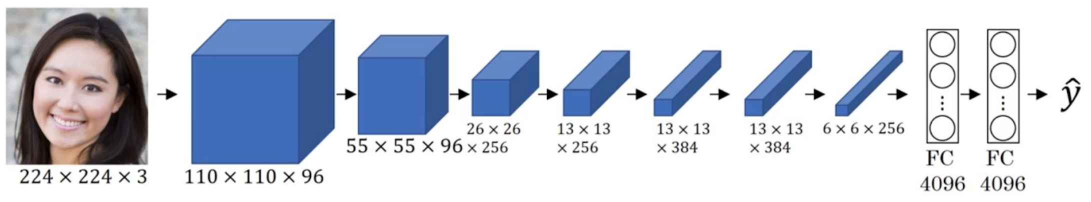
NEURAL STYLE TRANSFER

Neural style transfer



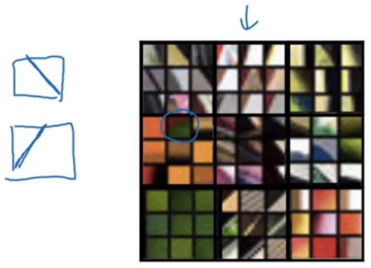


Visualizing what a deep network is learning



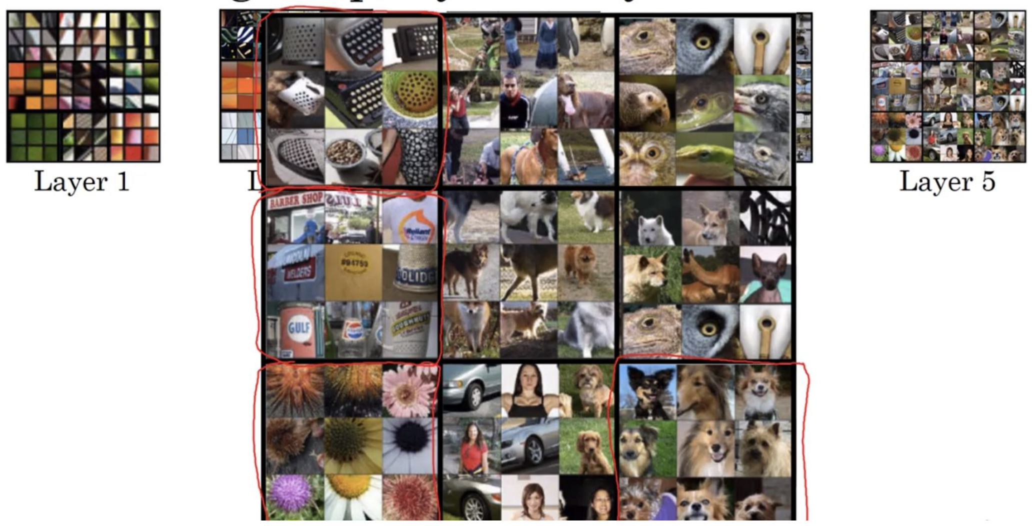
Pick a unit in layer 1. Find the nine image patches that maximize the unit's activation.

Repeat for other units.



The right below figure is the image patch that maximize the unit's activation.

Visualizing deep layers: Layer 5



COST FUNCTION OF NEURAL STYLE TRANSFER

Find the generated image G

1. Initiate G randomly

   G: 100 × 100 × 3

   RGB

2. Use gradient descent to minimize $J(G)$

$$G := G - \frac{d}{\partial G} J(G)$$

CONTENT COST FUNCTION AND STYLE COST FUNCTION

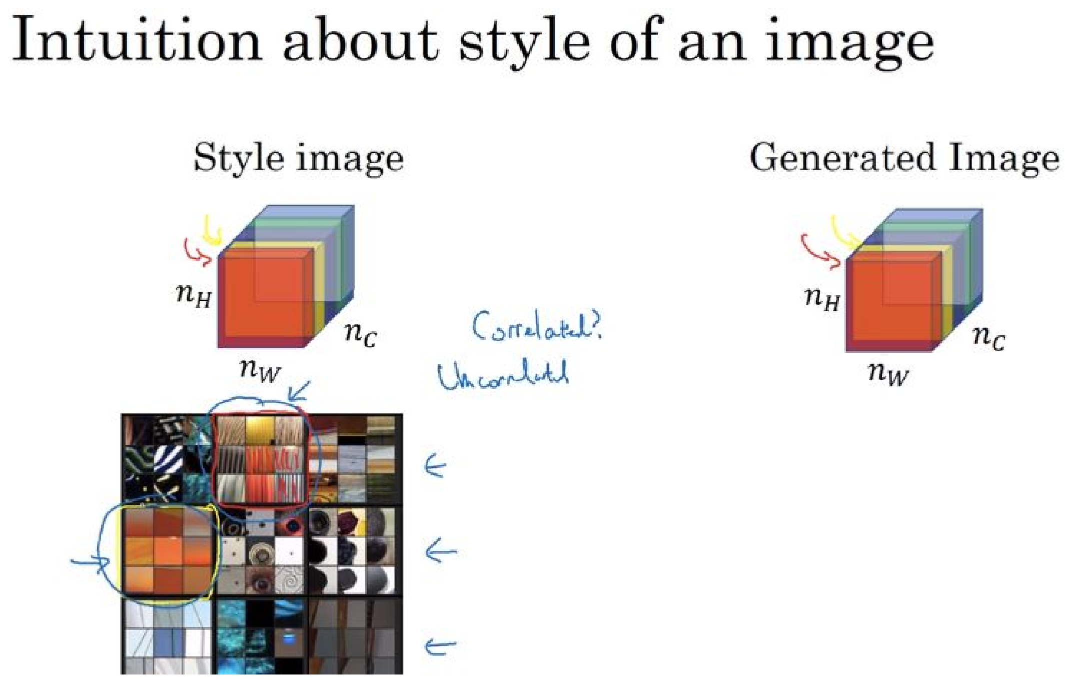
Content cost function

$$J(G) = \alpha \underline{J_{content}(C,G)} + \beta J_{style}(S,G)$$

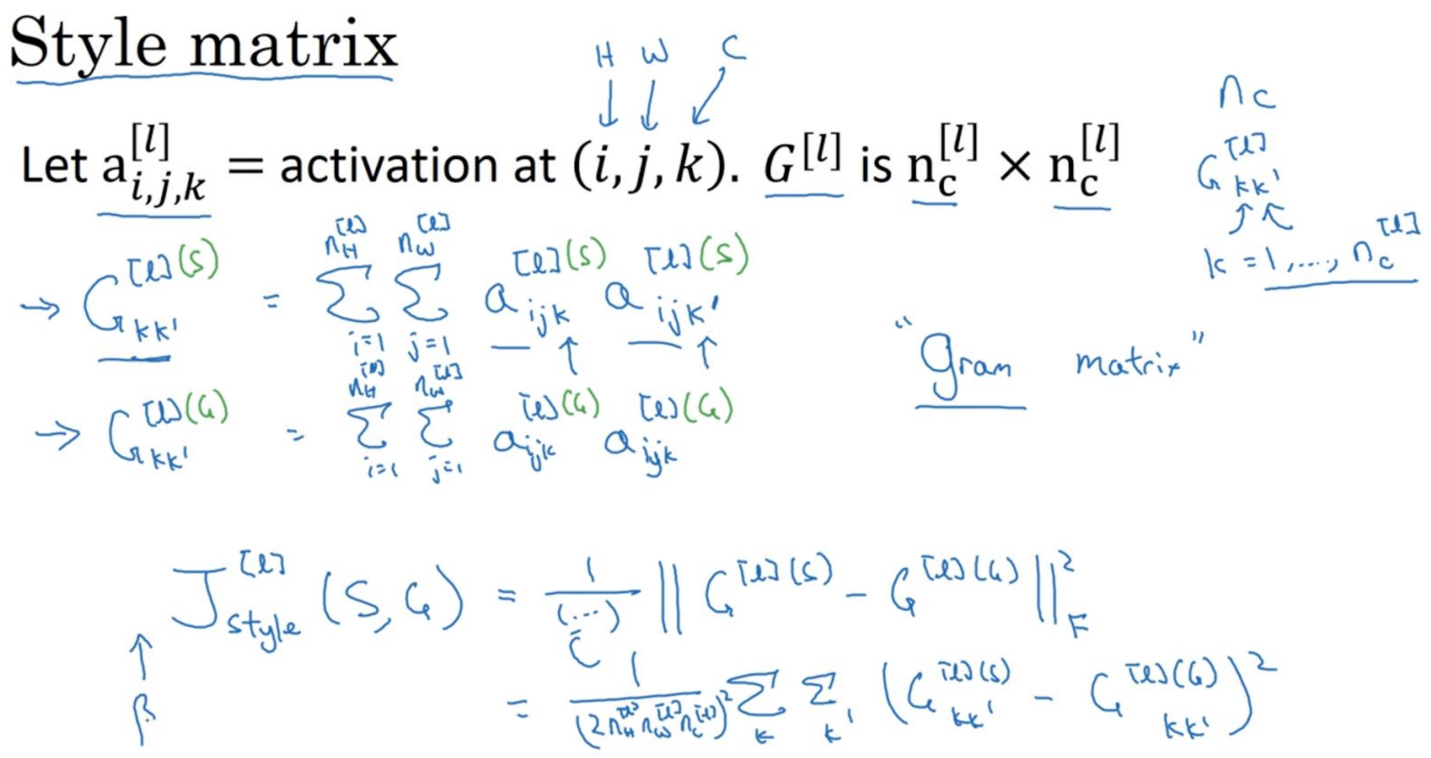
- Say you use hidden layer l to compute content cost.
- Use pre-trained ConvNet. (E.g., VGG network)
- Let $a^{[l](C)}$ and $a^{[l](G)}$ be the activation of layer l on the images
- If $a^{[l](C)}$ and $a^{[l](G)}$ are similar, both images have similar content

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

The style of the image is defined by the correlation in the hidden layer L across the channel. For example, if the correlation between the channel is high in the style image. It means that the image tend to have those features in the same time as well like have an orange with the vertical edge. Then we will use that kind of correlation to implement the cost function of the generated image as well.



We can formulate the concept by



Style matrix

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

H W C

n_c

$G^{[l]}_{kk'}$

$k = 1, \dots, n_c^{[l]}$

$$\rightarrow G^{[l](S)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l](S)}_{ijk} \, a^{[l](S)}_{ijk'}$$

"Gram matrix"

$$\rightarrow G^{[l](G)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l](G)}_{ijk} \, a^{[l](G)}_{ijk}$$

$$J^{[l]}_{style}(S,G) = \frac{1}{(\dots)} \left\| G^{[l](S)} - G^{[l](G)} \right\|_F^2$$

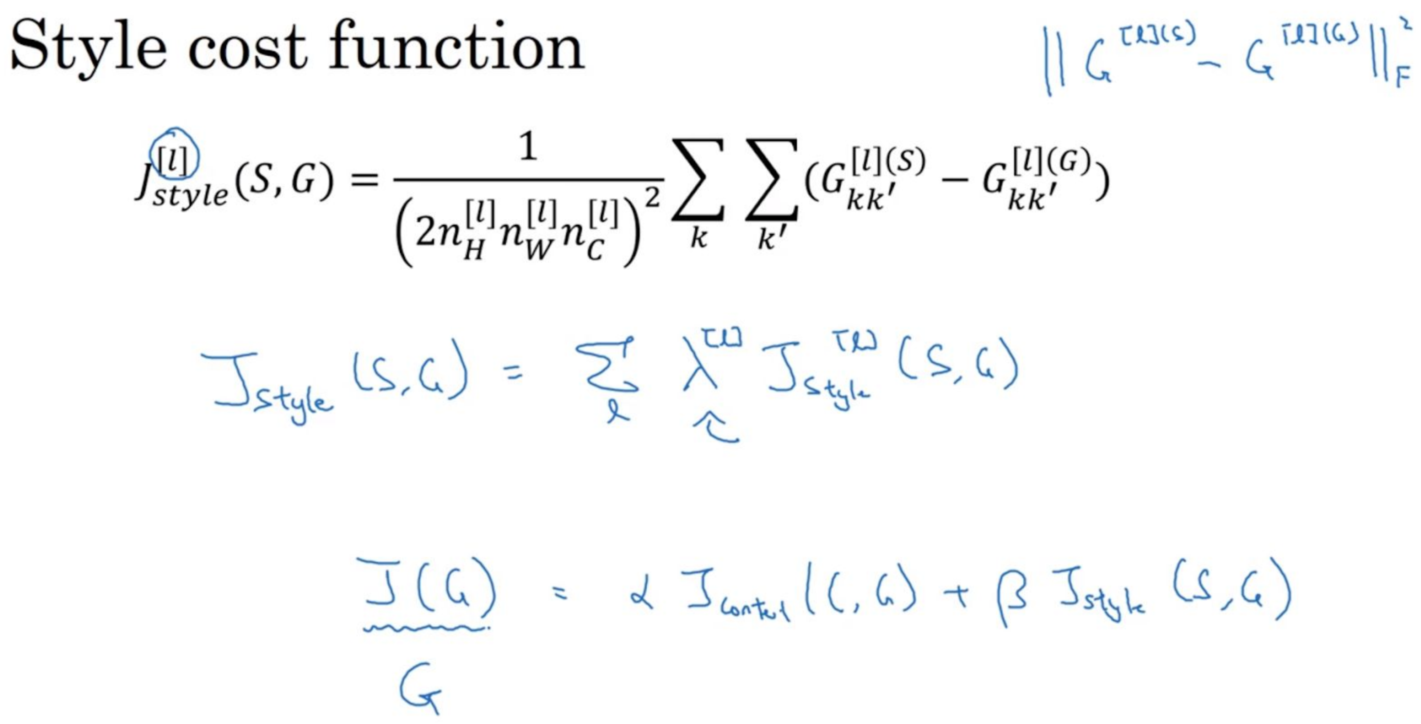
C β

$$= \frac{1}{(2 n_H^{[l]} n_W^{[l]} n_C^{[l]})^2} \sum_k \sum_{k'} \left(G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'} \right)^2$$

Where **i,j,k is the height, width, and channel of the hidden layer L** and we would like to find the sum of height and width of the product of activation unit at K and K prime.

We will do on both style image and generating image and use it in the cost function of style

For the above figure, we do this only on the Hidden layer L. So we expand this concept to several hidden layer and we will get the below figure.



Style cost function

$$\| G^{[l](S)} - G^{[l](G)} \|_F^2$$

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

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

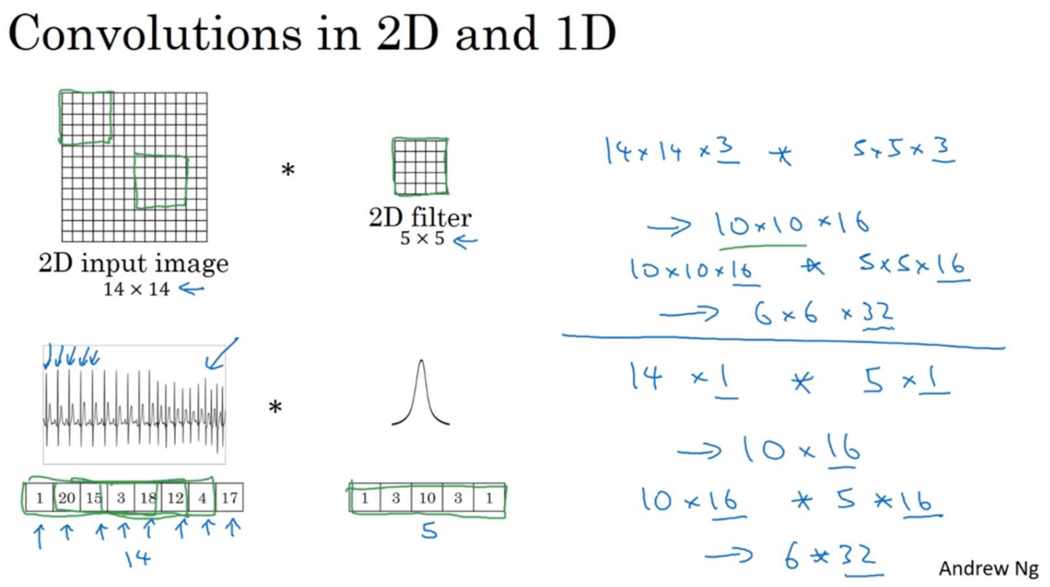
$$\underset{G}{\underbrace{J(G)}} = \alpha J_{content}(C,G) + \beta J_{style}(S,G)$$

The total style cost function is to sum the J style of each hidden layer L and product with the lambda [L] (another hyperparameter in this context)

Finally, we can combine the overall style transfer cost function and content function together

1D and 3D Generalizations of models

You can apply the convolution operation with the 1D data as well as 3D data



Convolutions in 2D and 1D

$14 \times 14 \times \underline{3} * 5 \times 5 \times \underline{3}$

$\rightarrow 10 \times 10 \times 16$

$10 \times 10 \times \underline{16} * 5 \times 5 \times \underline{16}$

$\rightarrow 6 \times 6 \times 32$

$14 \times \underline{1} * 5 \times \underline{1}$

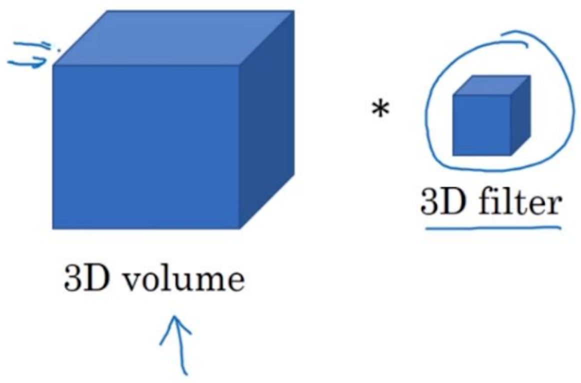
$\rightarrow 10 \times 16$

$10 \times \underline{16} * 5 * \underline{16}$

$\rightarrow 6 * 32$

For 1D data the convolutional operation is 1 of the option by the way there are other methods in the sequence model to tackle the 1D data as well. You will learn the pros and cons in that section too.

3D convolution



$$\overset{n_c}{}$$
$$14 \times 14 \times 14 \times 1$$
$$* \; 5 \times 5 \times 5 \times 1$$
16 filte.
$$\rightarrow 10 \times 10 \times 10 \times 16$$
$$* \; 5 \times 5 \times 5 \times 16$$
32 filtes
$$\rightarrow 6 \times 6 \times 6 \times 32$$

The example of 3D data such as the CT Scan or the movie data that you see the image changing over time period (height x width x time)