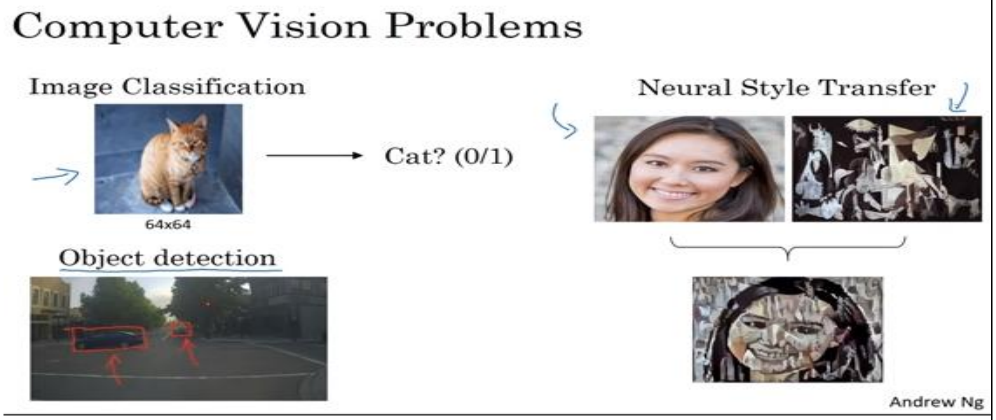
Course 4 Week 1

Computer Vision

Computer vision using deep learning techniques mainly relies on using convolutional neural networks which will be the main outline of this course especially week 1.

Computer vision is one of the rapidly advancing areas that helps in many fields as self-driving cars to figure out surrounding cars and pedestrians to avoid them. It also serve face recognition tasks as in phones, homes and work places to work better than before.

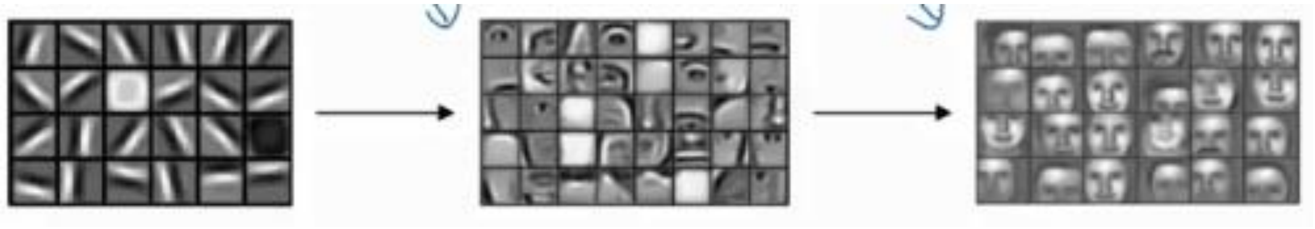
Computer vision problems: There are problems like image classification, object detection, neural style transfer, object segmentation, object localization, object tracking and much more.



Computer Vision Problems

One of the challenges facing computer vision is that inputs can get really big so for a 1 mega pixel image (which is actually pretty small comparing to what cameras output now) is actually a 1000 × 1000 × 3 = 3,000,000 (note that 3 here determines the 3 channels composing the image which red, green and blue) features for just 1 training example which is pretty large amount of data to be stored as a one input. This means that $w^{[1]}$ now is a matrix of $(n^{[1]}\times 3{,}000{,}000)$ matrix which means very large number of learning parameters that will make learning process harder on the computation requirements, memory requirements and also in getting much data to prevent overfitting with such large number of parameters. Here comes the convolution operations which is the fundamental element of computer vision.

The earlier layers of the neural networks usually focus on lines and edges and as we go further in the layer we are focusing on more complex shapes as curves then geometrical shapes as triangles, circles, rectangles, etc followed by more complex shapes. As an example if we are dealing with a face recognition problem so may be the early layers are dealing with edges of the face and its components and then intermediate layers are dealing with eyes, noses, ears, eye brows, mouths and so on then the latter layers of the network deals with the faces as awhole as shown in the example:



Let us focus on the earlier stages and see how convolution works on edge detection.

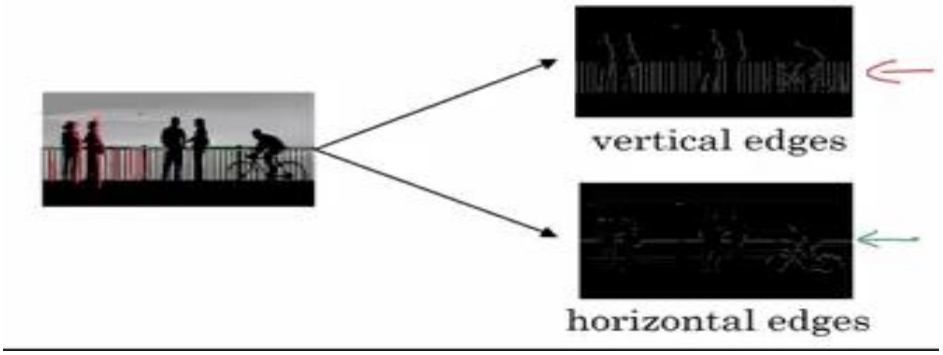
Convolutional operations: (edge detection as motivation example)

Assume having an example like this, so edge detection is usually done as:

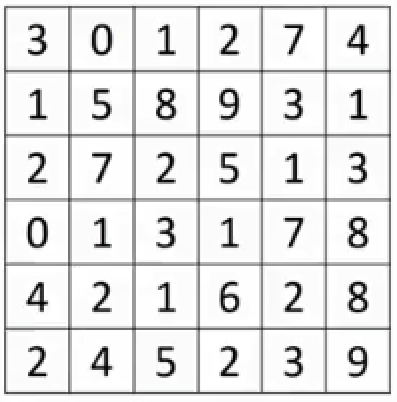
1) vertical edge detection

2)horizontal edge detection

So how would we detect edges in image like this?



Let's take a grey scale image (hence only 1 channel) as an example which is of size 6 × 6 and as said before each pixel of those 36 pixels will have a value between 0 (black) →255 (white).



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

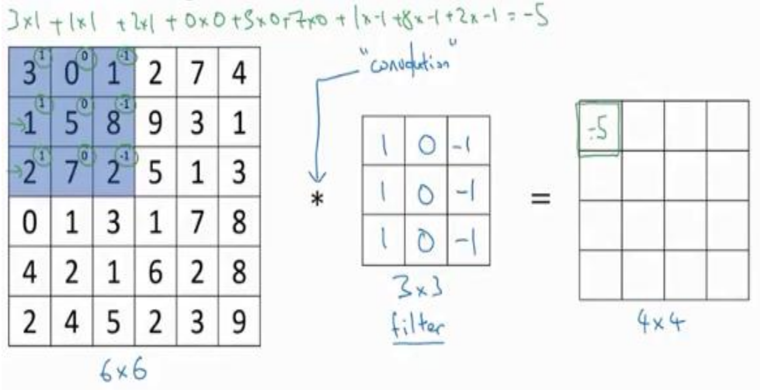
Let us do a vertical edge detection:

We need a matrix called filter or kernel which we will assume here as 3×3 filter having the following values: $\begin{bmatrix}1 & 0 & -1\\ 1 & 0 & -1\\ 1 & 0 & -1\end{bmatrix}$

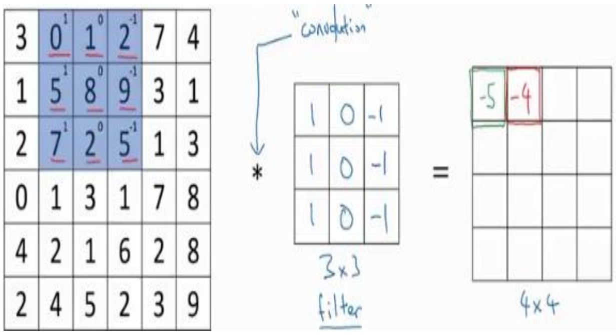
Then now we do convolutional operation between the image and this filter or kernel noting that convolutional operation is denoted by asterisk (*).

How we apply convolution in matrices?

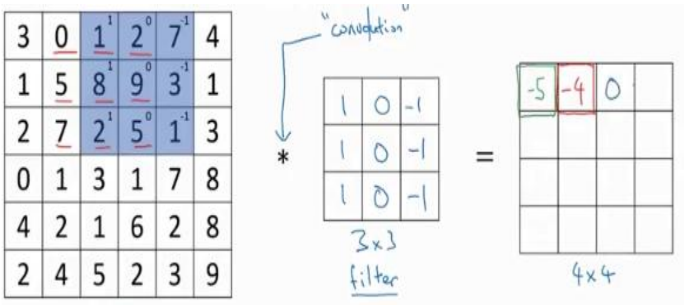
We just take the filter step by step from the up left corner and compute the summation of the element-wise multiplication between the filter and this up left corner and write the answer in the first place in the result and repeat this step by shifting the kernel to the left till you reach the end then repeat again by shifting downward and so on as shown below:



➔

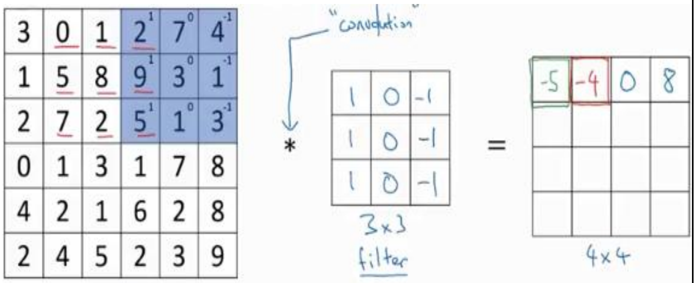


Then:



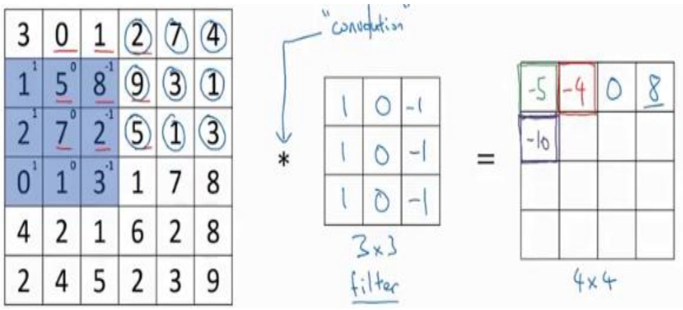
$$\begin{bmatrix}3&0&1&2&7&4\\1&5&8&9&3&1\\2&7&2&5&1&3\\0&1&3&1&7&8\\4&2&1&6&2&8\\2&4&5&2&3&9\end{bmatrix} * \begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix} = \begin{bmatrix}-5&-4&0&\\&&&\\&&&\\&&&\end{bmatrix}$$

➔



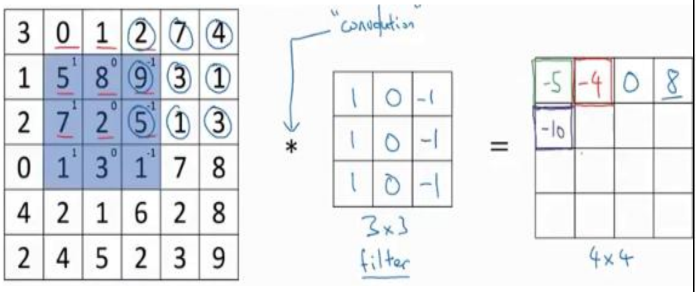
$$\begin{bmatrix}3&0&1&2&7&4\\1&5&8&9&3&1\\2&7&2&5&1&3\\0&1&3&1&7&8\\4&2&1&6&2&8\\2&4&5&2&3&9\end{bmatrix} * \begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix} = \begin{bmatrix}-5&-4&0&8\\&&&\\&&&\\&&&\end{bmatrix}$$

Then:



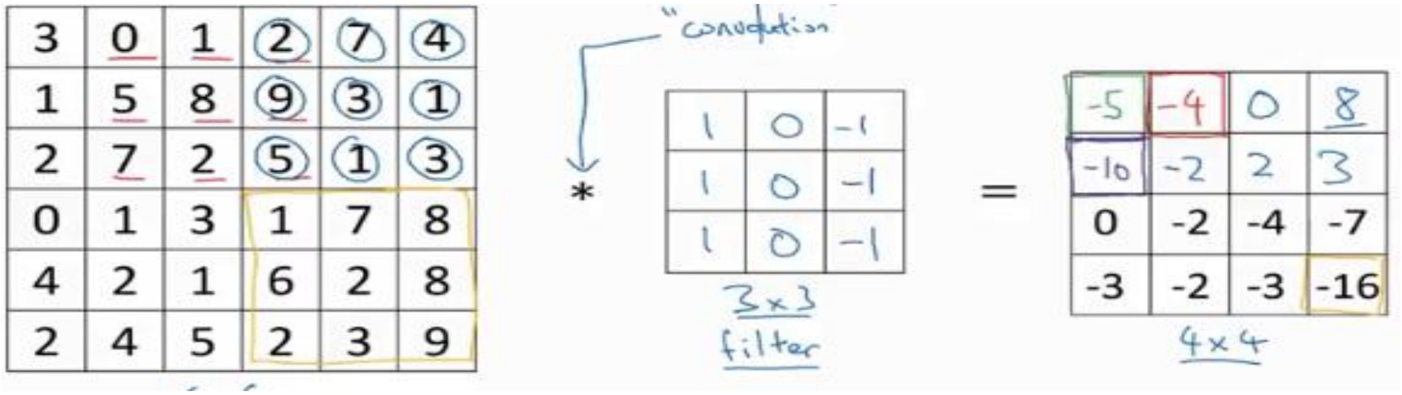
$$\begin{bmatrix}3&0&1&2&7&4\\1&5&8&9&3&1\\2&7&2&5&1&3\\0&1&3&1&7&8\\4&2&1&6&2&8\\2&4&5&2&3&9\end{bmatrix} * \begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix} = \begin{bmatrix}-5&-4&0&8\\-10&&&\\&&&\\&&&\end{bmatrix}$$

➔



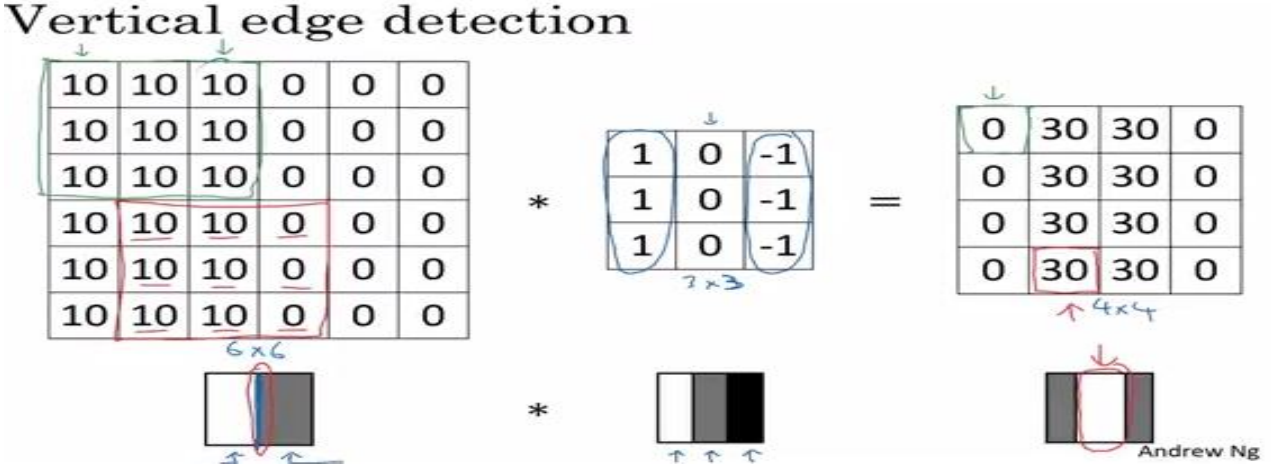
$$\begin{bmatrix}3&0&1&2&7&4\\1&5&8&9&3&1\\2&7&2&5&1&3\\0&1&3&1&7&8\\4&2&1&6&2&8\\2&4&5&2&3&9\end{bmatrix} * \begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix} = \begin{bmatrix}-5&-4&0&8\\-10&&&\\&&&\\&&&\end{bmatrix}$$

And so on till you reach the end of the image as shown:



$$\begin{bmatrix}3&0&1&2&7&4\\1&5&8&9&3&1\\2&7&2&5&1&3\\0&1&3&1&7&8\\4&2&1&6&2&8\\2&4&5&2&3&9\end{bmatrix} * \begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix} = \begin{bmatrix}-5&-4&0&8\\-10&-2&2&3\\0&-2&-4&-7\\-3&-2&-3&-16\end{bmatrix}$$

Another example

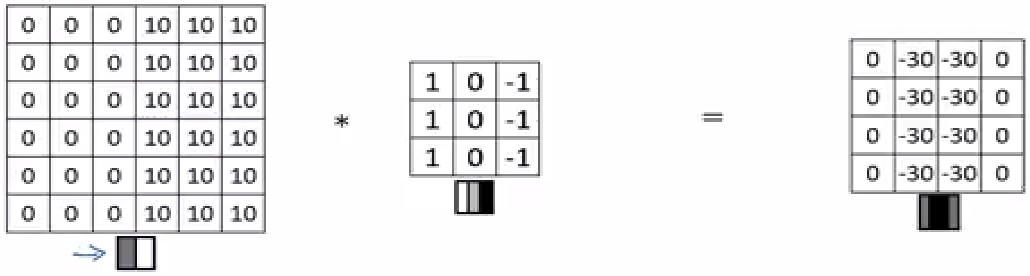


Vertical edge detection

$$\begin{bmatrix}10&10&10&0&0&0\\10&10&10&0&0&0\\10&10&10&0&0&0\\10&10&10&0&0&0\\10&10&10&0&0&0\\10&10&10&0&0&0\end{bmatrix} * \begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix} = \begin{bmatrix}0&30&30&0\\0&30&30&0\\0&30&30&0\\0&30&30&0\end{bmatrix}$$

You notice here that the vertical edge in the original image is translated into a brighter region between the 2 darker regions to show that there is a vertical edge existed here.

If we flipped the original image horizontally such that the dark is on the left and bright is on the right here will be the result :➔



$$\begin{bmatrix} 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \end{bmatrix}$$

The difference between the two images that the first one was light to dark transition while the second photo was a dark to light transition (moving from left to right (→)).

For horizontal edge detection

we here use a transposed filter of that used with vertical edge detection ➔ $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

Notes: (convolution properties)

1- Kernels can be any n×n kernel (usually odd number) not just 3×3 but the most common are 3×3, 5×5, 7×7 & 9×9.

2- If n×n matrix convolved with f×f matrix, the result will be n-f+1 × n-f+1.

3- Convolution has linear, commutative, associative & distributive properties.

What is the best set of numbers to use in the kernel to perform vertical and horizontal detection?

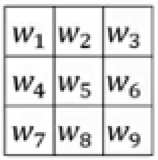
It turned out that Sobel filter is one of the best edge detecting kernels that puts more weight on the center which makes it more robust. ➔ vertical: $\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$ & horizontal: $\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

But computer vision researchers wanted to exaggerate more so they are now using Scharr filter:

Vertical: $\begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix}$ & Horizontal: $\begin{bmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix}$

Are these 2 filters really the best?

No, These were the best in the classical techniques but with the rise of deep learning era we found out that we don't need to handpick the numbers but in fact we need to make these filters get learned by learning algorithm to find more robust filters. ➔



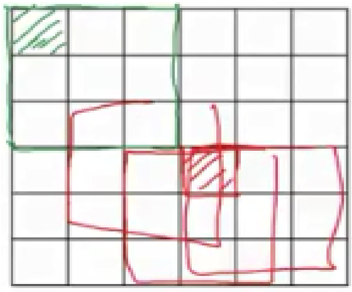
$$\begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix}$$

Note: Edge detection is not only for vertical or horizontal but also could be done with edge detection for edges with slope (like 45° edges)

Padding

Motivation behind using padding

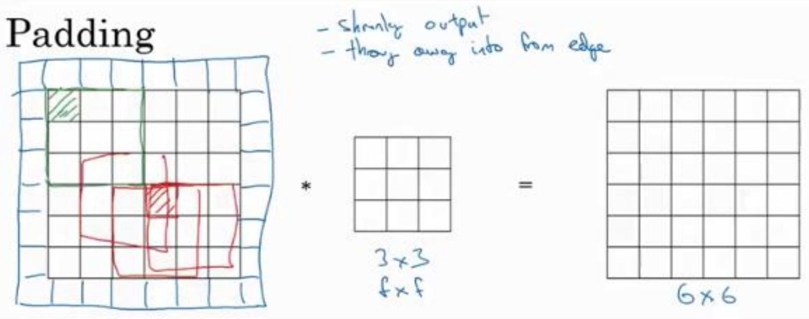
One down side of the convolution is that it will shrink the result so when entered a 6×6 image to convolve with a filter of 3×3 size we ended up with a 4×4 image which means that our image now shrunk from 6×6 to 4×4 which means that if we apply convolution several times on an image it will end up being so small. Another downside is that some pixels are involved in the convolution less than the others so for example the up left pixel in the image is just involved once in the convolution while for example the pixels in the middle of the image is involved more than once (some are twice, or 3 times or whatever) ➔



This means that we throw away some information near edges by only using them small number of times compared to the others.

Those problems are solved by padding.

Padding is adding a border of pixels around the image to increase the size of the image as shown:



As we can notice here that we entered 6×6 image and ended with 6×6 also so image is not shrunk in size and also now the edges of the image are included more often than before so we are not throwing this information.

Note: Padding is done with zeros.

NOW, if we have n×n matrix convolved with f×f filter and padded by P so the result will be n+2P-f+1 × n+2P-f+1.

Types of convolutions regarding padding

1)Valid padding: This means no padding. (P=0)

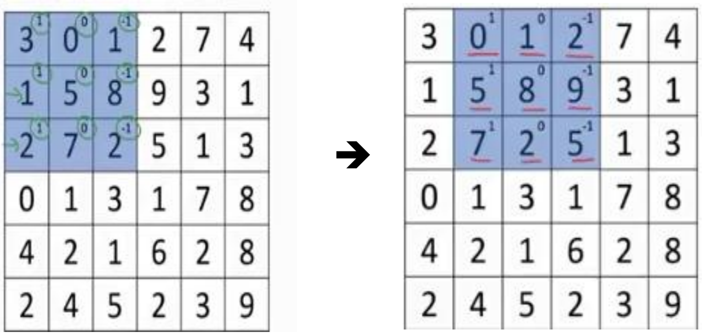
2)Same Padding: This means to pad such that input and output of convolution is of same size.(P= $\frac{f-1}{2}$)

Remember: Filters are usually using odd numbers.

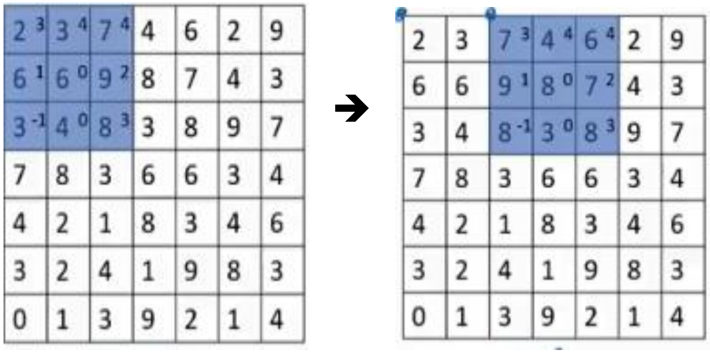
Stride

Stride means the step you take when convolving so usually we were taking stride = 1 as default so the filter was moving 1 step each time as shown:

If stride=1:

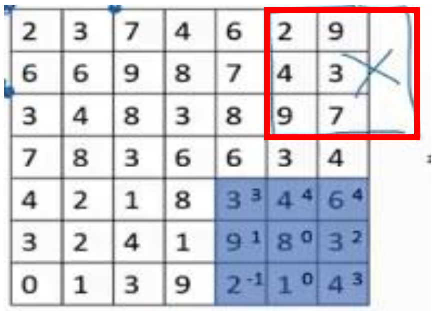


If stride=2:



As seen in the previous example that the filter is moving with 1 step on the left in case of stride=1 and 2 steps on the right in the case of stride =2.

Note: Take care that the stride is convenient with the image size in order not to make filter goes out the image due to this stride as shown ➔

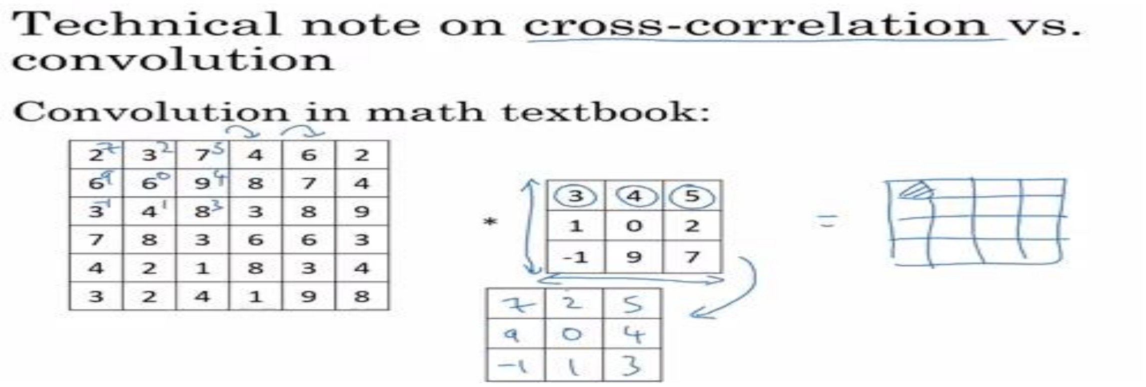


Convolution output size

For n×n input image using f×f filter with padding P and stride S so the output result will be:

$$\frac{n+2P-f}{S}+1 \times \frac{n+2P-f}{S}+1$$

Note: In the Math and signal processing textbooks convolution is not done as we did but they firstly mirror the filter before applying it and what we did here in math is called cross-correlation but in deep learning cross-correlation is called convolution and this is the convention deep learning researchers used to use.



Technical note on cross-correlation vs. convolution

Convolution in math textbook:

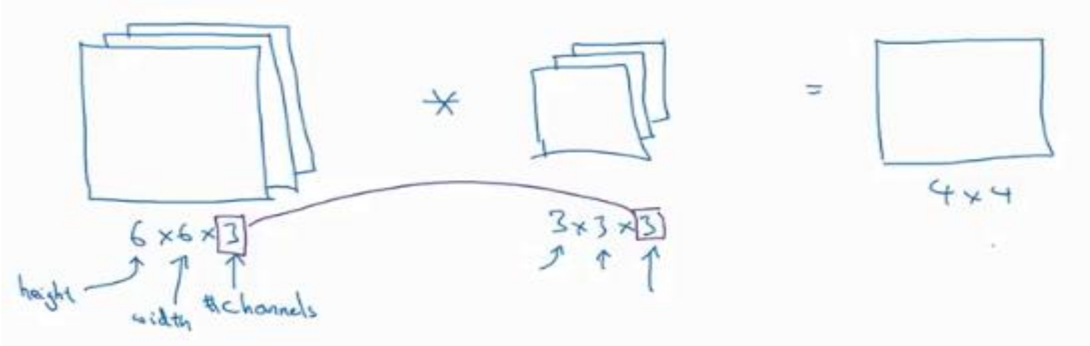
Convolution on RGB images

What we were doing was done on 2-D image or grey scale image but for RGB images which is more common we now has 3-D images so how convolution deal with them?

This will be done by stacking filters as shown:

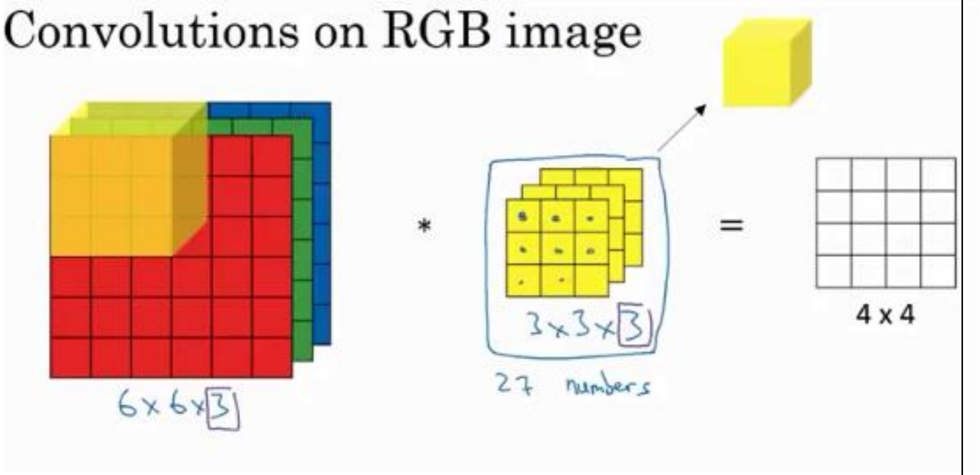
Note:

Number of channels in the input image should equal number of channels of the filter.



But How 6×6×3 matrix convolved with 3×3×3 filter ends with 4×4×1 matrix?

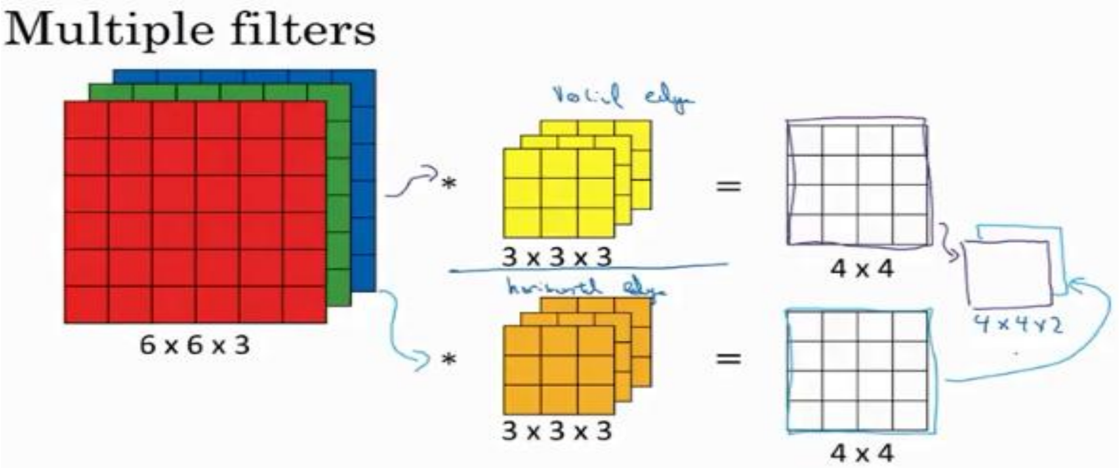
Take the filter 3×3×3 and apply it on the image such that each channel of image corresponds channel of filter and multiply each two numbers then add them all (the 27 numbers) and put them in the final result.



Multiple filters

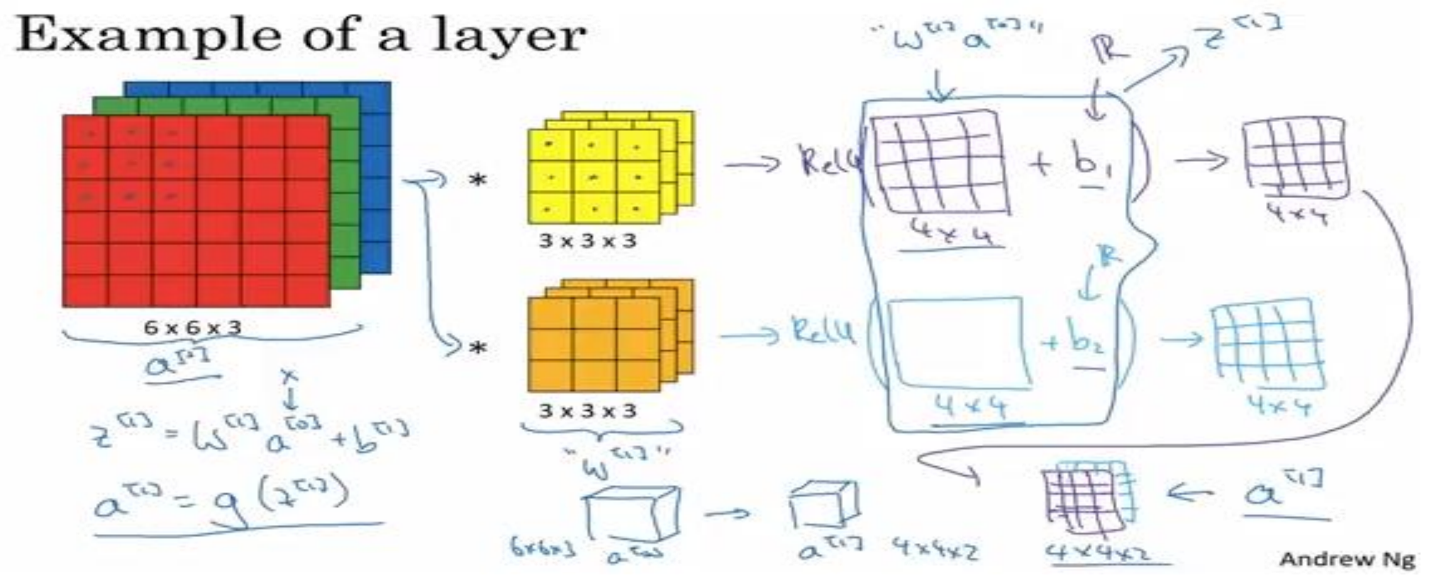
In fact we don't always want to only find vertical or horizontal but in fact we want them both in addition to a lot of filters that even find the e]inclined edges with different degrees that's how we need multiple filters to convolve with them to let each one has a specific task. So if we assume that we have an image and we want to get the horizontal and vertical edge detection so we will convolve the image once with vertical edge detector kernel and once with horizontla one then stack the output together. (output now is $\frac{n+2P-f}{S}+1 \times \frac{n+2P-f}{S}+1 \times n_c'$)

where n_c' is the number of filters used.



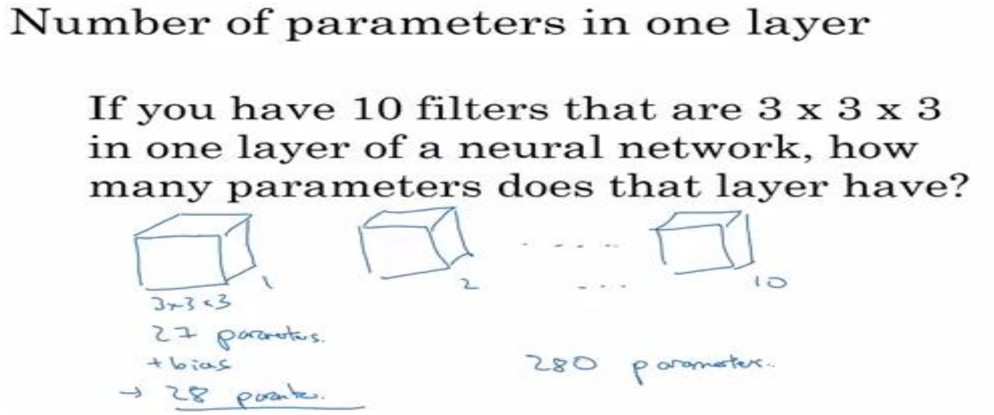
Convolutional neural network

We start with n×n×3 image ($a^{[0]}$) and convolve with a number of n_c' kernels of size f×f×3 ($w^{[1]}$) so the output now is $\frac{n+2P-f}{S}+1 \times \frac{n+2P-f}{S}+1 \times n_c'$ ($w^{[1]}.a^{[0]}$) then now add a bias term to each output of the n_c' outputs ($z^{[1]}$) and apply activation function on them ($a^{[1]}$) so now we have moved from $a^{[0]}$ to $a^{[1]}$ which is one layer of convolutional neural network. Check the example:



Example of a layer

Exercises

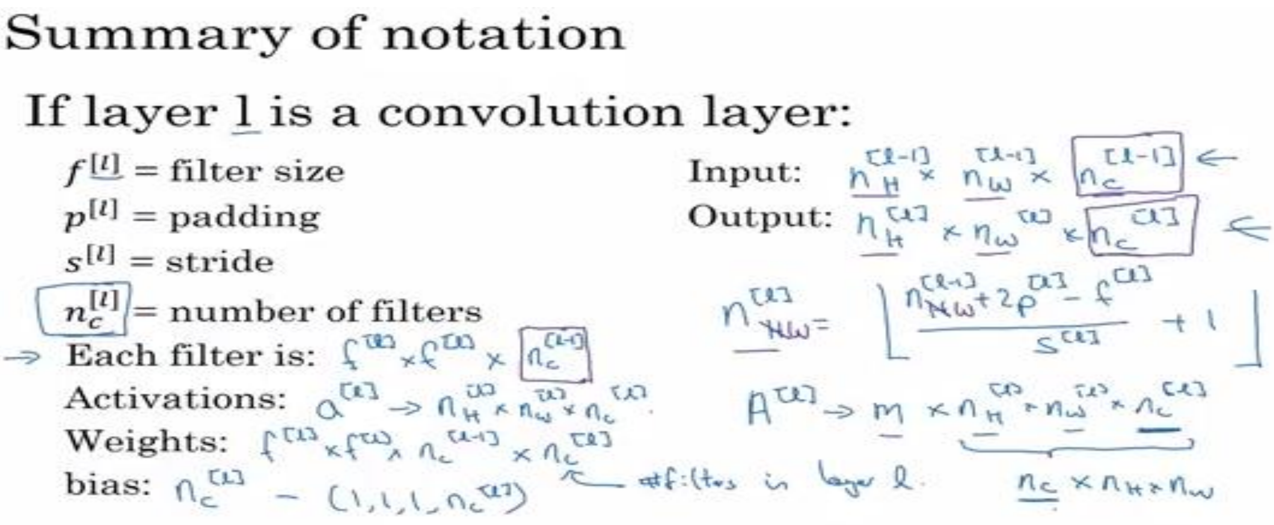


Number of parameters in one layer

If you have 10 filters that are 3 x 3 x 3 in one layer of a neural network, how many parameters does that layer have?

Note: The good thing here that no matters how big is the size of image, we still have the same number of parameters as long as we have the same filters' size and number.

Notations



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

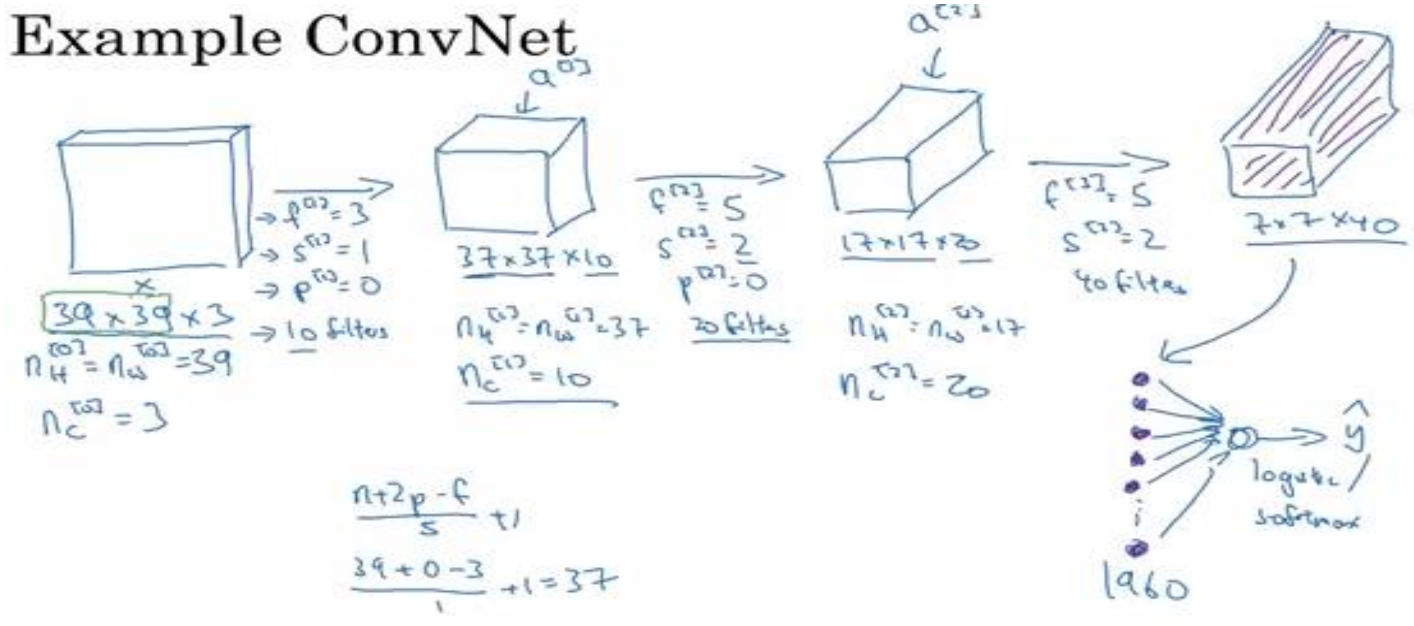
Output: $n_H^{[l]} \times n_W^{[l]} \times \boxed{n_c^{[l]}}$ ←

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

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

$n_c \times n_H \times n_W$

Simple convolutional neural network



Take care from how sizes differs from one layer to another:

From layer 1 → layer2: since we have 39×39×3 with kernel 3×3×3 and P=0 and S=1 and n_c'=10

Hence the output equals $\frac{39+2(0)-3}{1}+1 \times \frac{39+2(0)-3}{1}+1 \times 10$ = 37×37×10

From layer2 →layer3: we have 37×37×10 with kernel 5×5×10 and P=0 and S=2 and n_c'=20

Hence the output equals $\frac{37+2(0)-5}{2}+1 \times \frac{37+2(0)-5}{2}+1 \times 20$ = 17×17×20

From layer3→layer4: we have 17×17×20 with kernel 5×5×20 and P=0 and S=2 and n_c'=40

Hence the output equals $\frac{17+2(0)-5}{2}+1 \times \frac{17+2(0)-5}{2}+1 \times 40$ = 7×7×40

From layer4→layer5: This is just flattening as 7×7×40=1960 to be able to fed to softmax function to perform prediction of $\hat{y}$.

Note: As we go deeper in the network our image is getting diminished .

Types of layers in convolutional neural network

1) Convolution layer (CONV)

2)Pooling layer (POOL)

3)Fully connected layer (FC)

Any neural network is composed of a number of these layers to form our CNN.

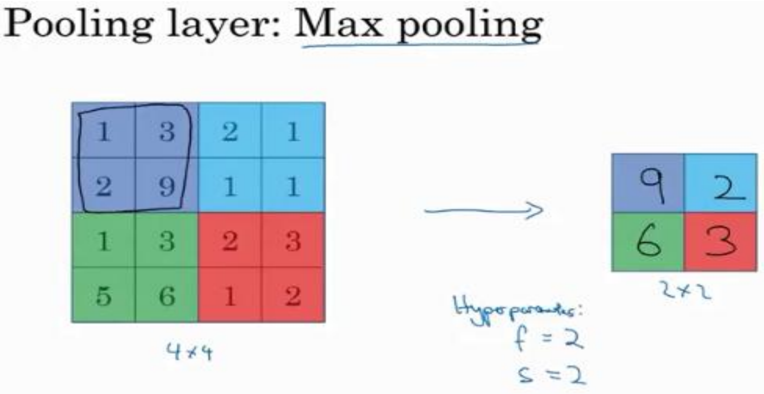
Pooling layers

Pooling layers is mainly used to reduce the size of the representation, speed computation and make the network more robust to the features.

Pooling has no learning parameters unlike convolution so gradient descent has nothing to learn.

Max pooling

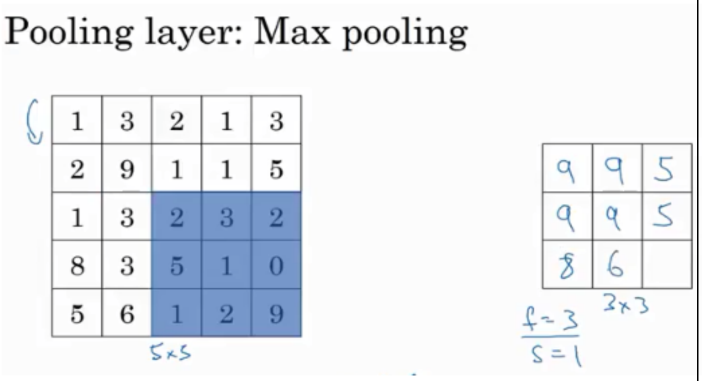
Max pooling is to take the highest value of a bunch of pixels in an image to represent this bunch by this value as shown in the figures



Pooling layer: Max pooling

1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

9	2
6	3

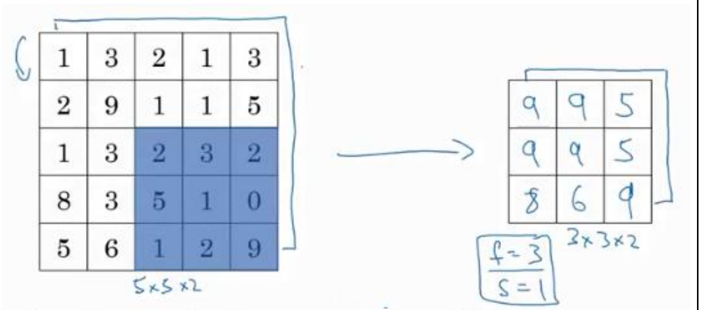


Pooling layer: Max pooling

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

9	9	5
9	9	5
8	6	

For having many channels, so the output will have the same number of channels

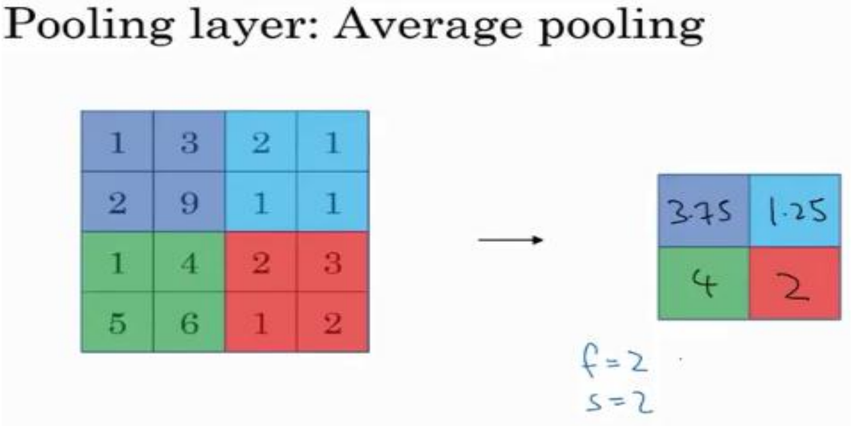


1	3	2	1	3
2	9	1	1	5
1	3	2	3	2
8	3	5	1	0
5	6	1	2	9

9	9	5
9	9	5
8	6	9

Average pooling

Similarly to max pooling but instead of taking the highest value we take the average of the pixels.



Pooling layer: Average pooling

1	3	2	1
2	9	1	1
1	4	2	3
5	6	1	2

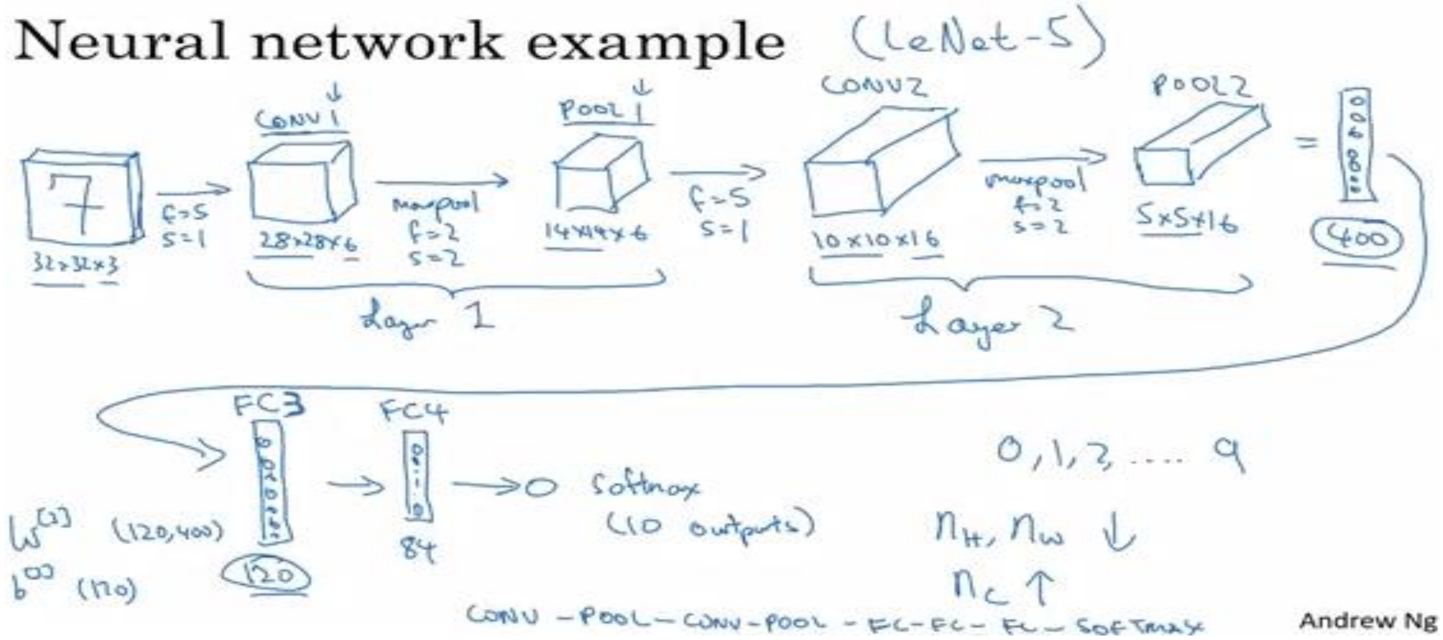
3.75	1.25
4	2

So, For n×n×n$_c$ image therefore the output of the max pooling layer will be:

$$\frac{n-f}{S}+1\ \times\ \frac{n-f}{S}+1\ \times n_c$$

Note: We don't usually use padding with pooling layer.

Neural network example (LeNet-5)



Neural network example

Andrew Ng

Notes:

1-This is the simplest deep neural network in the history.

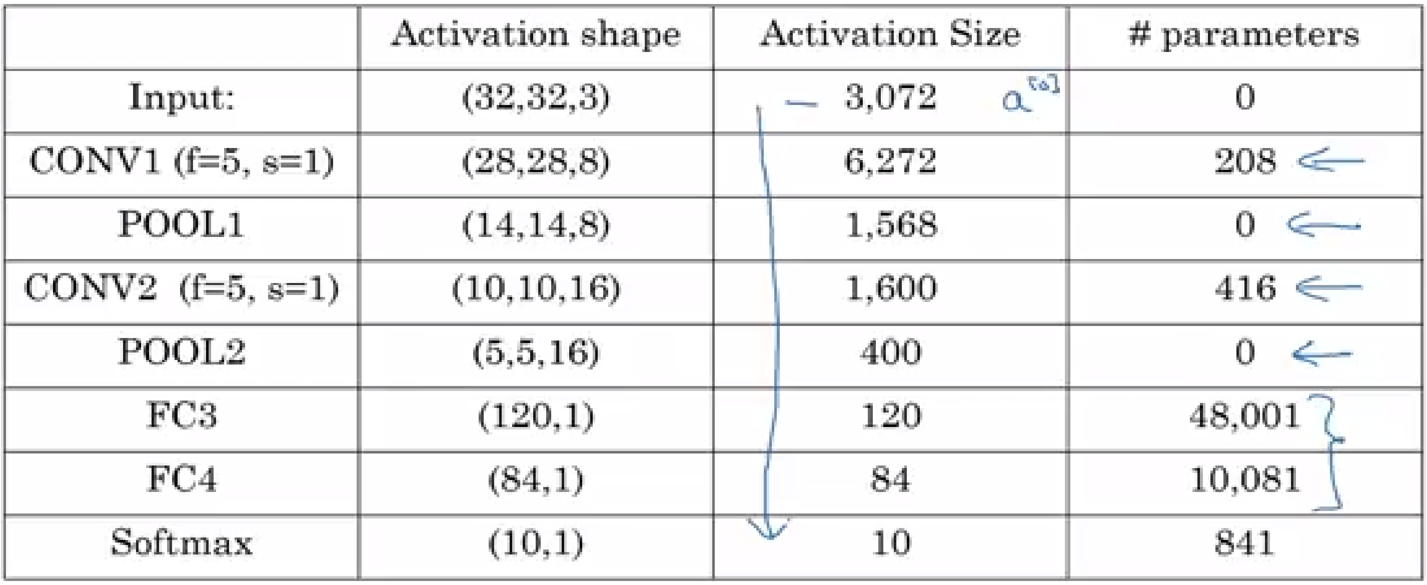
2-Each conv layer followed by pooling layer are together called a neural network layer.

3-Conv-pool-conv-pool-…-conv-pool-FC-FC-…-FC is a pretty common pattern used in most of neural networks.

4-Each neuron in the last pooling layer is connected to all neuron in the first FC layer that why it is of size (120,400).

5-Softmax has 10 outputs in this example as it was an example of hand-written digits recognition.

Notice this table that translates the previous 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