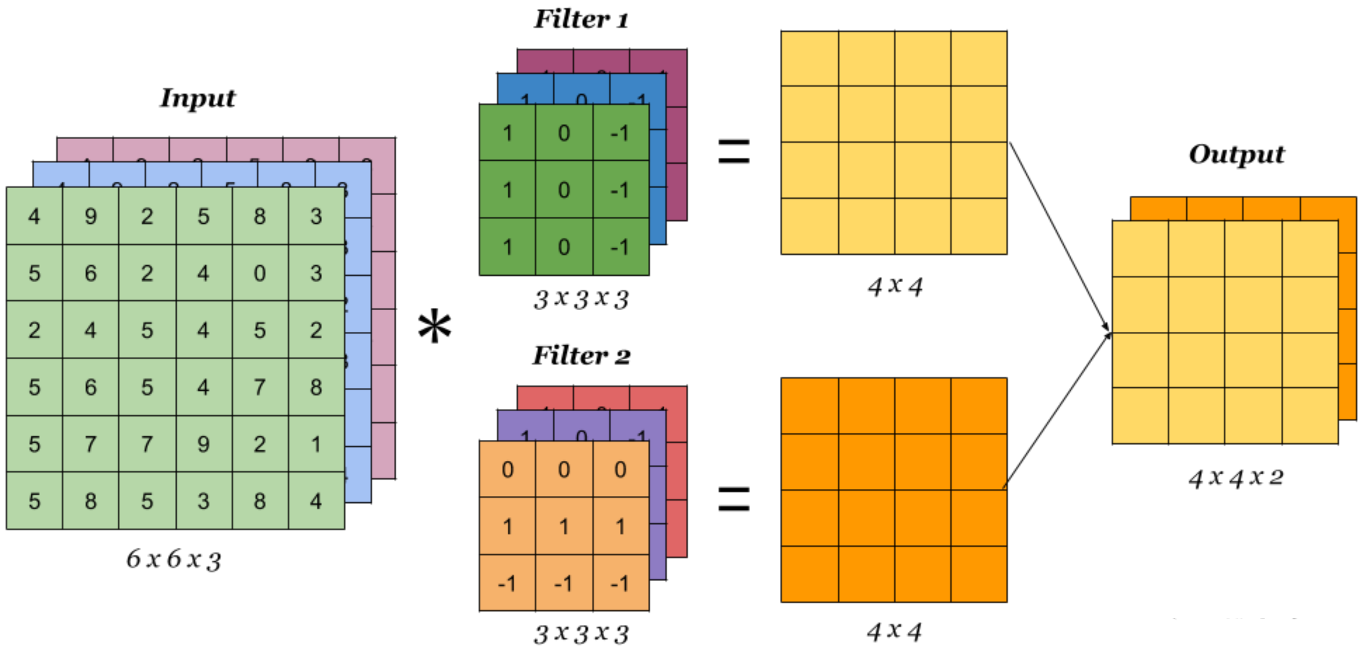
CNN

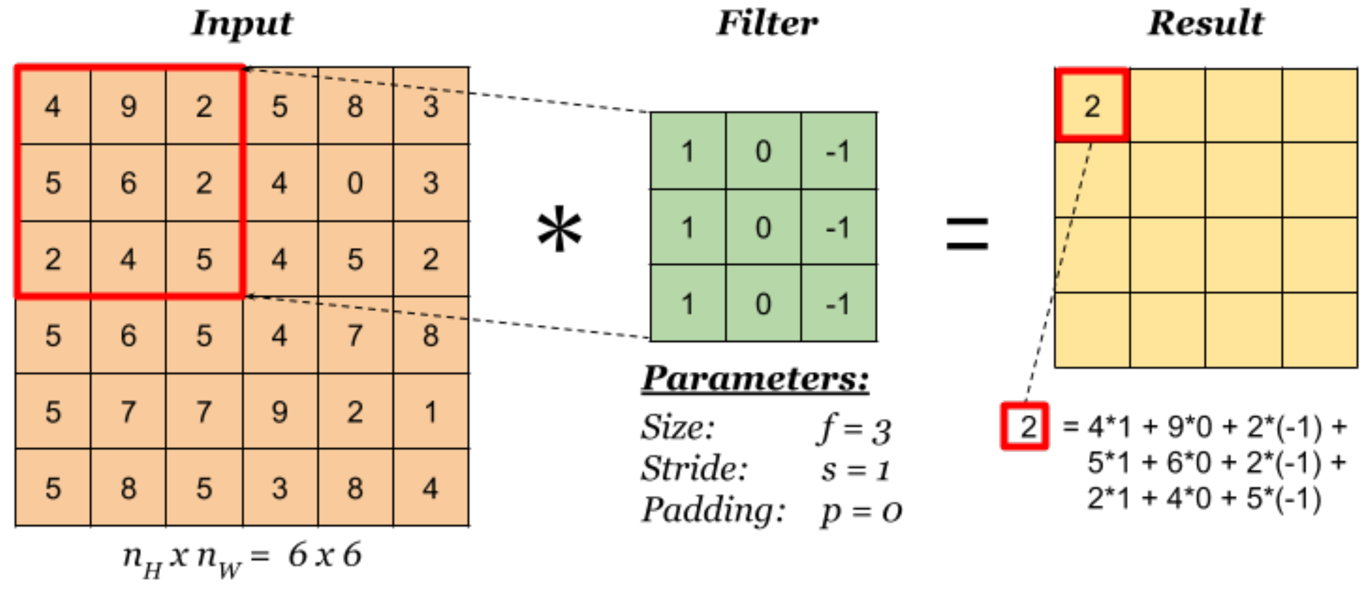
Notes


Input
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
6 x 6 x 3
*
Filter 1
1 0 -1
1 0 -1
1 0 -1
3 x 3 x 3
=
4 x 4
Filter 2
0 0 0
1 1 1
-1 -1 -1
3 x 3 x 3
=
4 x 4
Output
4 x 4 x 2

Convolution Operation

Basic Convolution Operation

Step 1: overlay the filter to the input, perform element wise multiplication, and add the result.



Input

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

$n_H \, x \, n_W = 6 \, x \, 6$

*

Filter

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

=

Result

2			

Parameters:

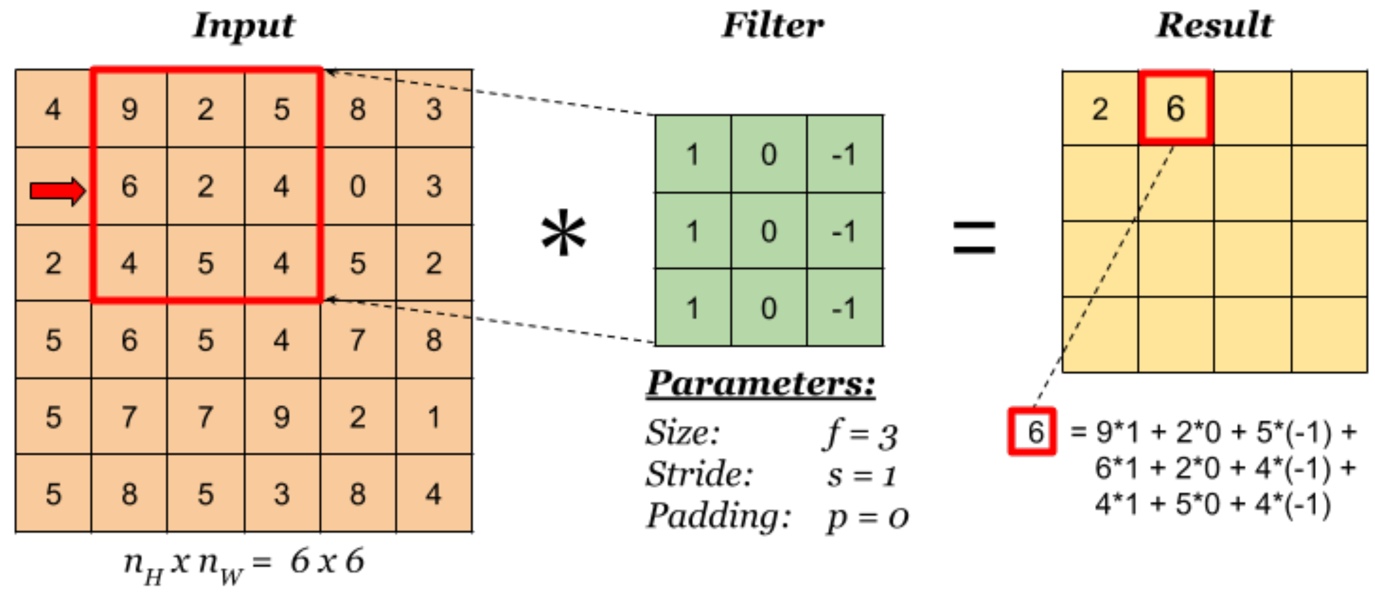
Size: $f = 3$

Stride: $s = 1$

Padding: $p = 0$

2 = 4*1 + 9*0 + 2*(-1) +
5*1 + 6*0 + 2*(-1) +
2*1 + 4*0 + 5*(-1)

Step 2: move the overlay right one position (or according to the **stride** setting), and do the same calculation above to get the next result. And so on.



Input

4	9	2	5	8	3
	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

$n_H \, x \, n_W = 6 \, x \, 6$

*

Filter

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

=

Result

2	6		

Parameters:

Size: $f = 3$

Stride: $s = 1$

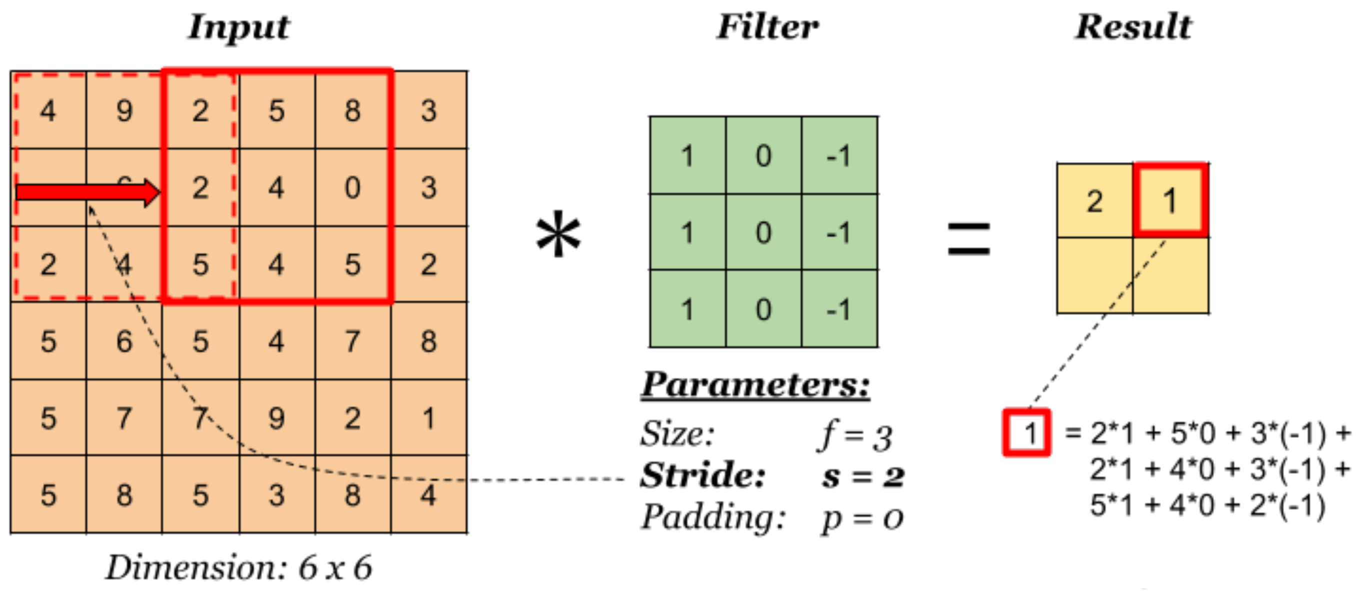
Padding: $p = 0$

6 = 9*1 + 2*0 + 5*(-1) +
6*1 + 2*0 + 4*(-1) +
4*1 + 5*0 + 4*(-1)

The total number of multiplications to calculate the result above is (4 x 4) x (3 x 3) = 144.

Stride

Stride governs how many cells the filter is moved in the input to calculate the next cell in the result.



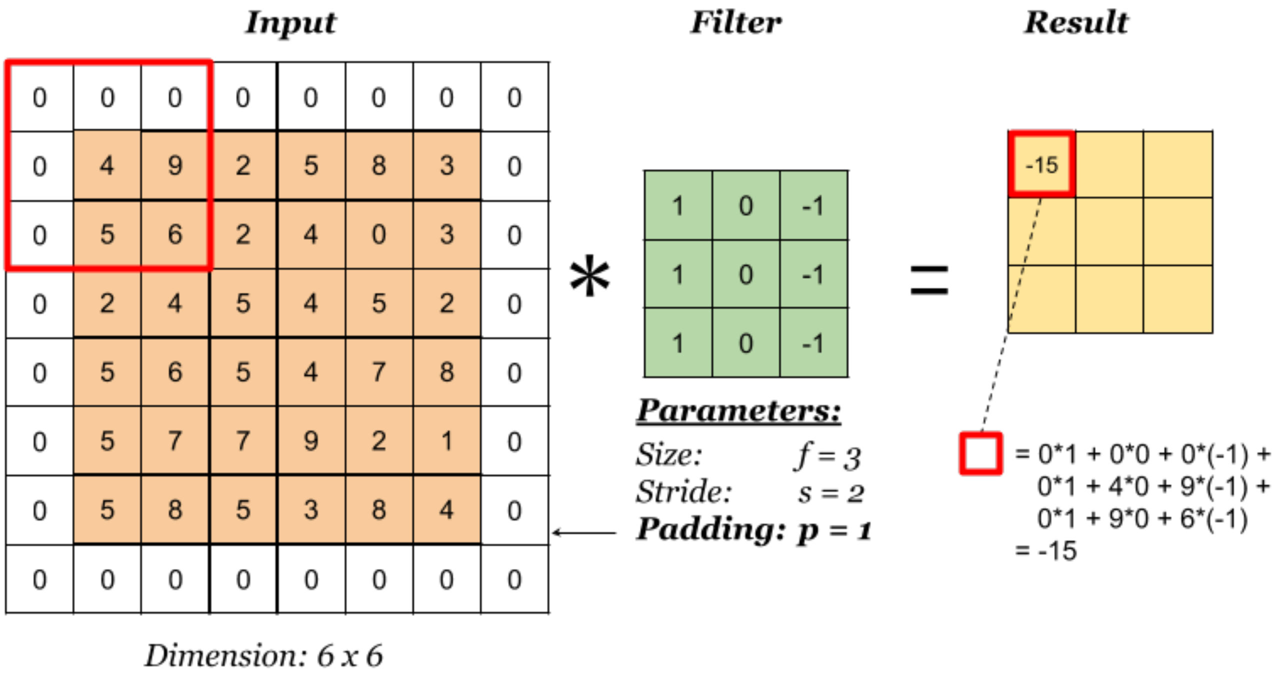
Filter with stride (s) = 2

The total number of multiplications to calculate the result above is (2 x 2) x (3 x 3) = 36.

Padding

Padding has the following benefits:

1. It allows us to use a CONV layer without necessarily shrinking the height and width of the volumes. This is important for building deeper networks, since otherwise the height/width would shrink as we go to deeper layers.
2. It helps us keep more of the information at the border of an image. Without padding, very few values at the next layer would be affected by pixels as the edges of an image.



Notice the the dimension of the result has changed due to padding. See the following section on how to calculate output dimension.

Some padding terminologies:

- "**valid**" padding: no padding
- "**same**" padding: padding so that the output dimension is the same as the input

Calculating the Output Dimension

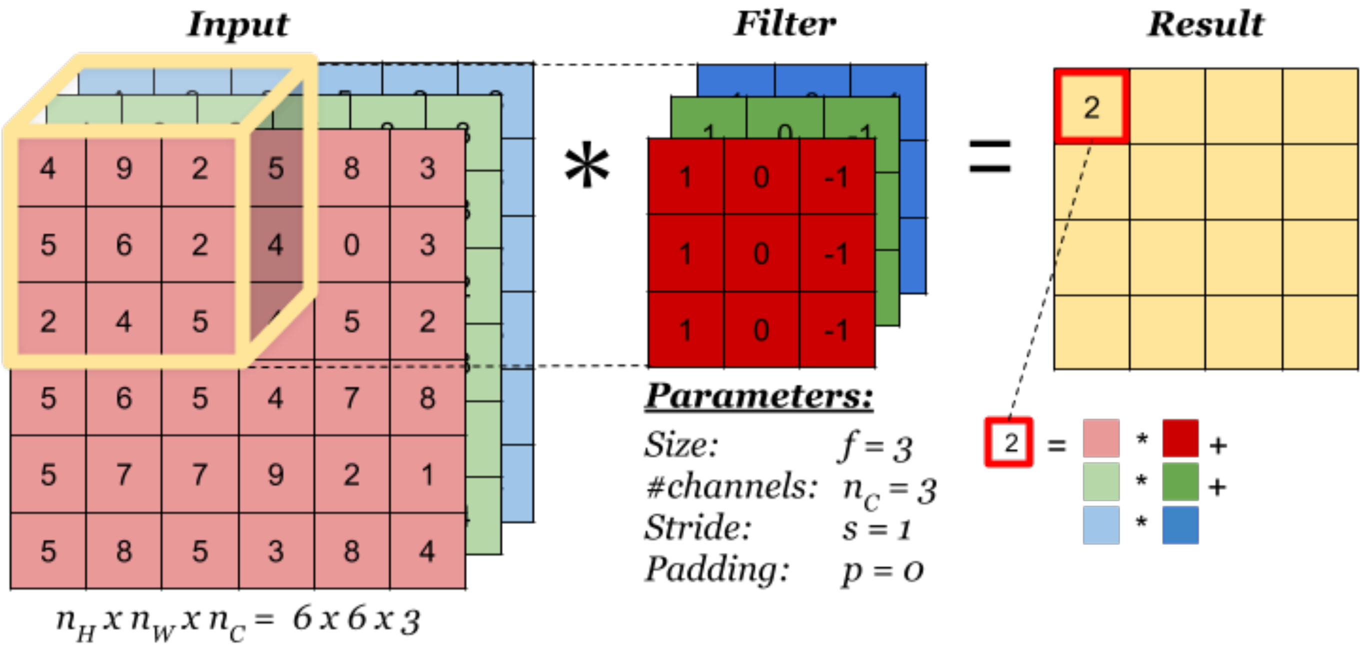
The output dimension is calculated with the following formula:

$$n^{[l]} = \lfloor \frac{n^{[l-1]}+2p^{[l-1]}-f^{[l]}}{s^{[l]}} + 1 \rfloor$$

where the $\lfloor \ \rfloor$ symbols denote *math.floor()* operation.

Convolution Operation on Volume

When the input has more than one channels (e.g. an RGB image), the filter should have matching number of channels. To calculate one output cell, perform convolution on each matching channel, then add the result together.



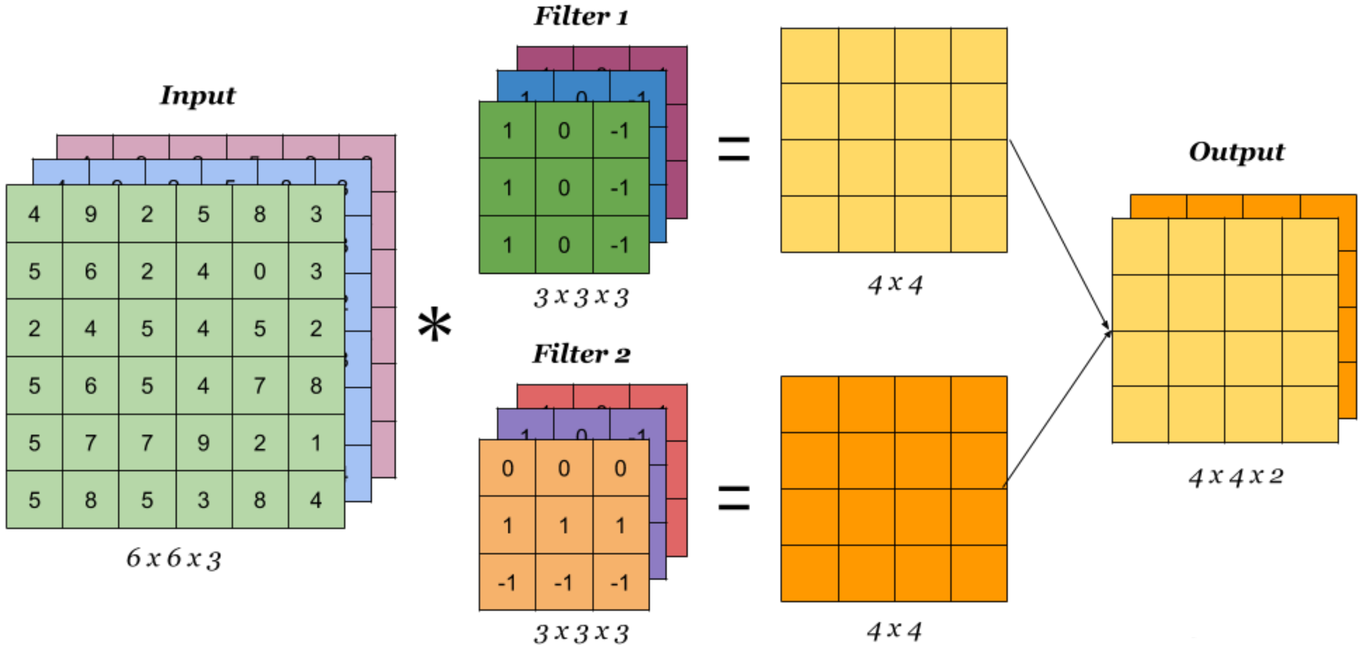
4	9	2	5	8	3
5	6	2	4	0	3
2	4	5	4	5	2
5	6	5	4	7	8
5	7	7	9	2	1
5	8	5	3	8	4

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

The total number of multiplications to calculate the result is (4 x 4) x (3 x 3 x 3) = 432.

Convolution Operation with Multiple Filters

Multiple filters can be used in a convolution layer to detect multiple features. The output of the layer then will have the same number of channels as the number of filters in the layer.



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

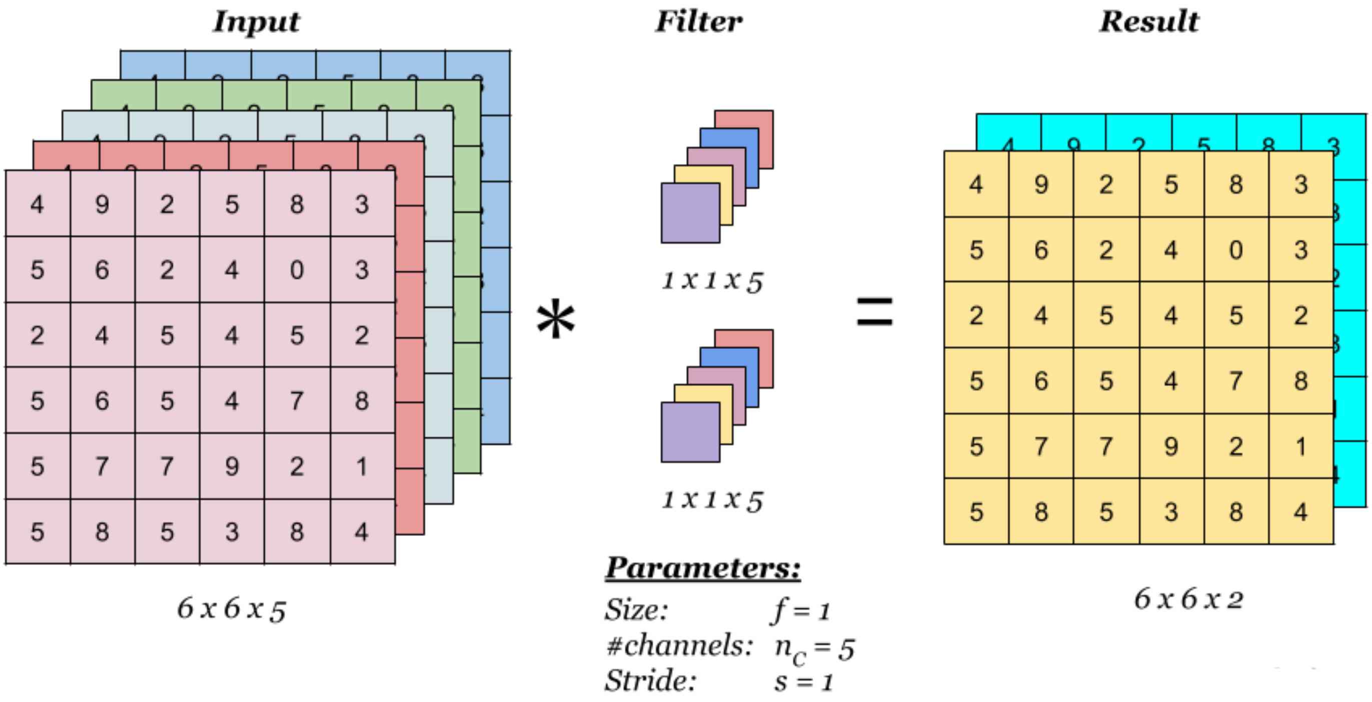
1	0	-1
1	0	-1
1	0	-1

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

The total number of multiplications to calculate the result is (4 x 4 x 2) x (3 x 3 x 3) = 864.

1 x 1 Convolution

This is convolution with 1 x 1 filter. The effect is to flatten or “merge” channels together, which can save computations later in the network:



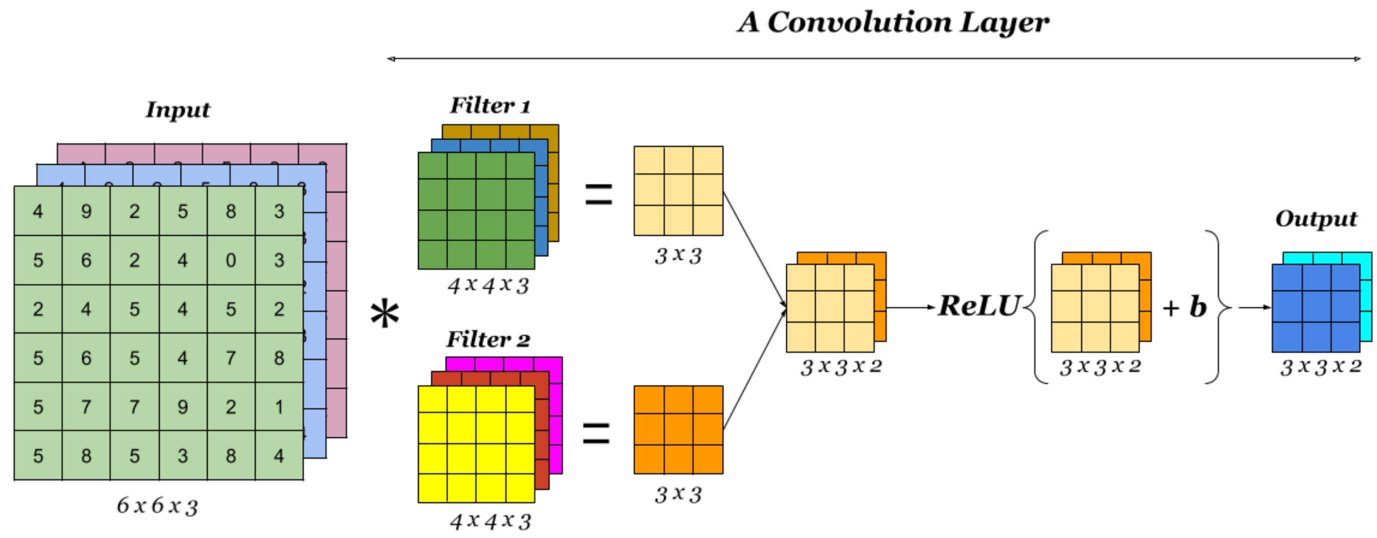
4	9	2	5	8	3
5	6	2	4	0	3
2	4	5	4	5	2
5	6	5	4	7	8
5	7	7	9	2	1
5	8	5	3	8	4

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

~

One Convolution Layer

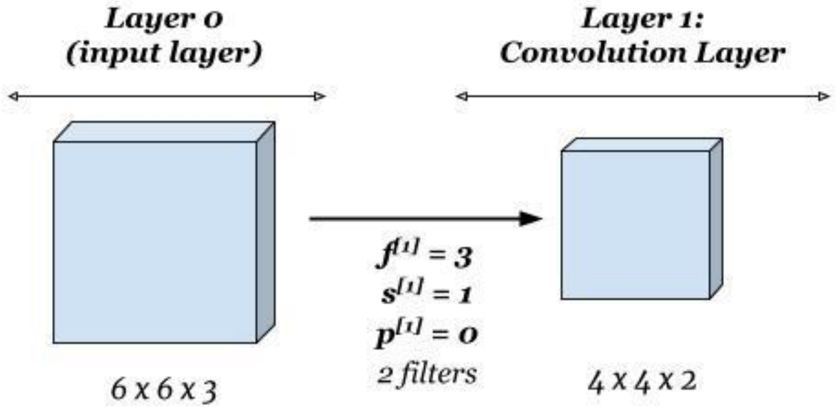
Finally to make up a convolution layer, a bias (ϵ R) is added and an activation function such as **ReLU** or **tanh** is applied.



~

Shorthand Representation

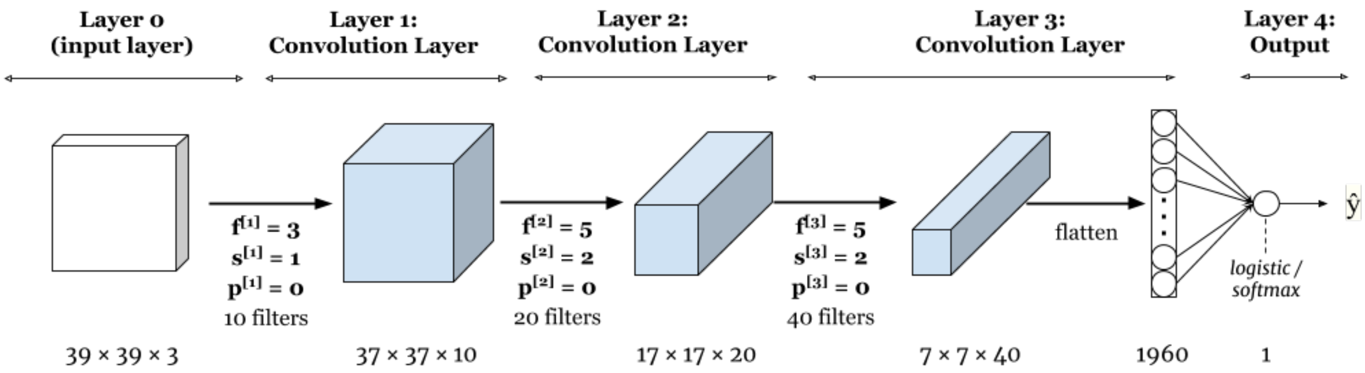
This simpler representation will be used from now on to represent one convolutional layer:



~

Sample Complete Network

This is a sample network with three convolution layers. At the end of the network, the output of the convolution layer is flattened and is connected to a logistic regression or a softmax output layer.

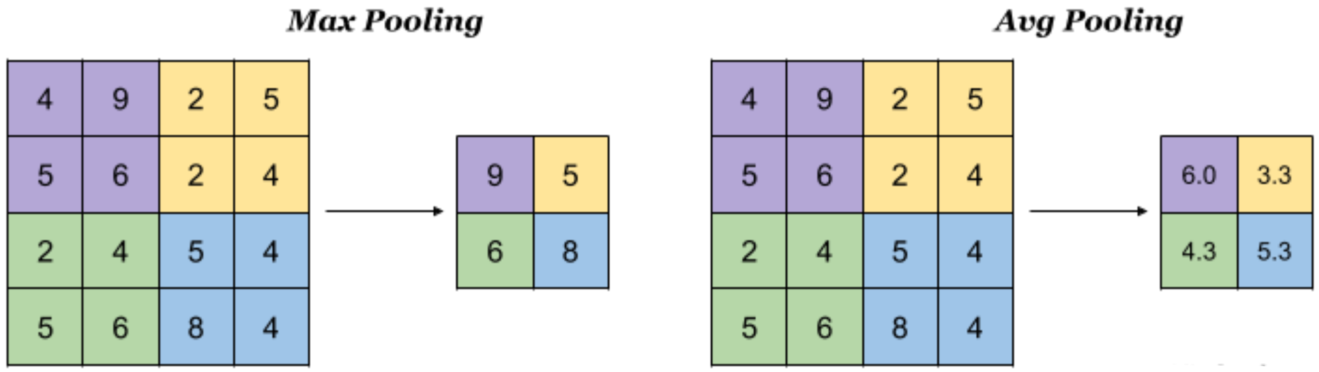


~

Pooling Layer

Pooling layer is used to reduce the size of the representations and to speed up calculations, as well as to make some of the features it detects a bit more robust.

Sample types of pooling are **max pooling** and **avg pooling**, but these days max pooling is more common.



4	9	2	5
5	6	2	4
2	4	5	4
5	6	8	4

9	5
6	8

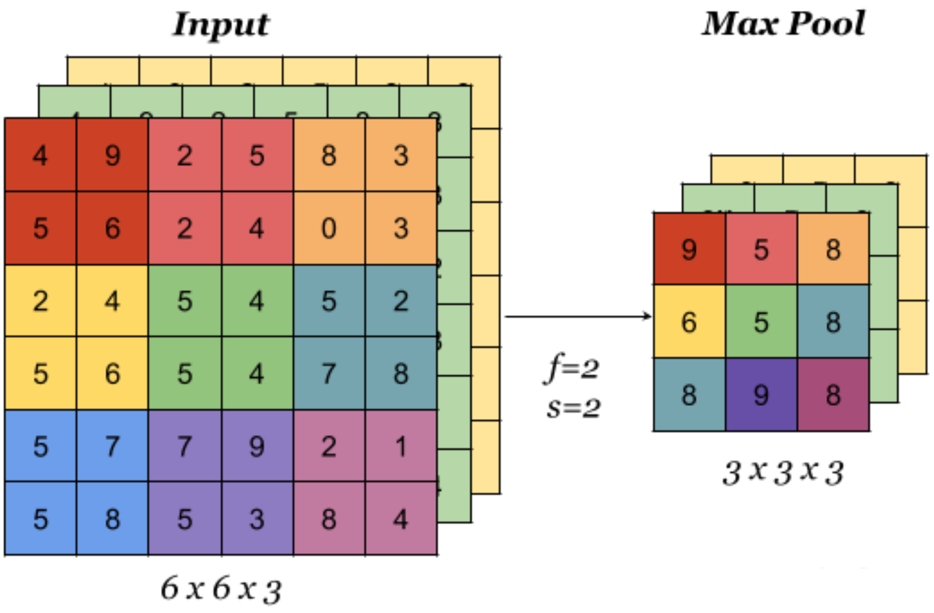
4	9	2	5
5	6	2	4
2	4	5	4
5	6	8	4

6.0	3.3
4.3	5.3

Interesting properties of pooling layer:

- it has hyper-parameters:
  - **size** (*f*)
  - **stride** (*s*)
  - **type** (max or avg)
- but it doesn't have parameter; there's nothing for gradient descent to learn

When done on input with multiple channels, pooling reduces the height and width (nW and nH) but keeps nC unchanged:

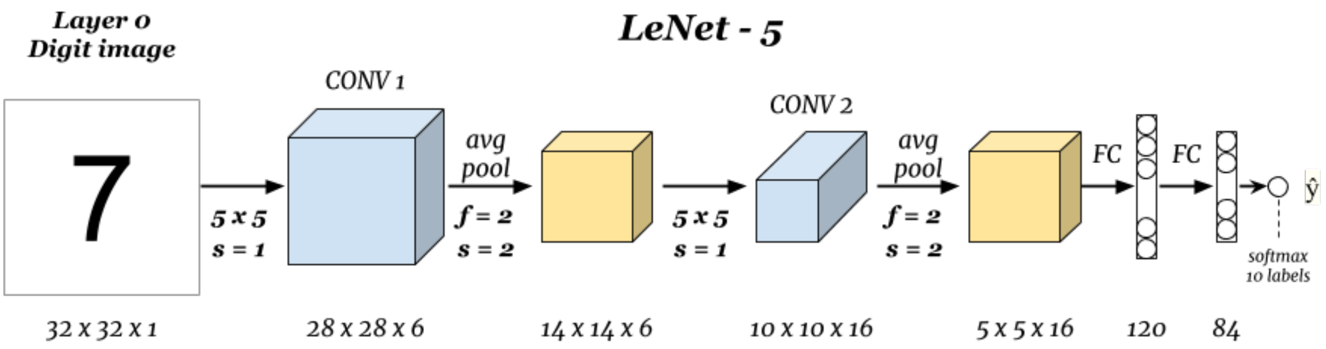


~

Well Known Architectures

Classic Network: LeNet – 5

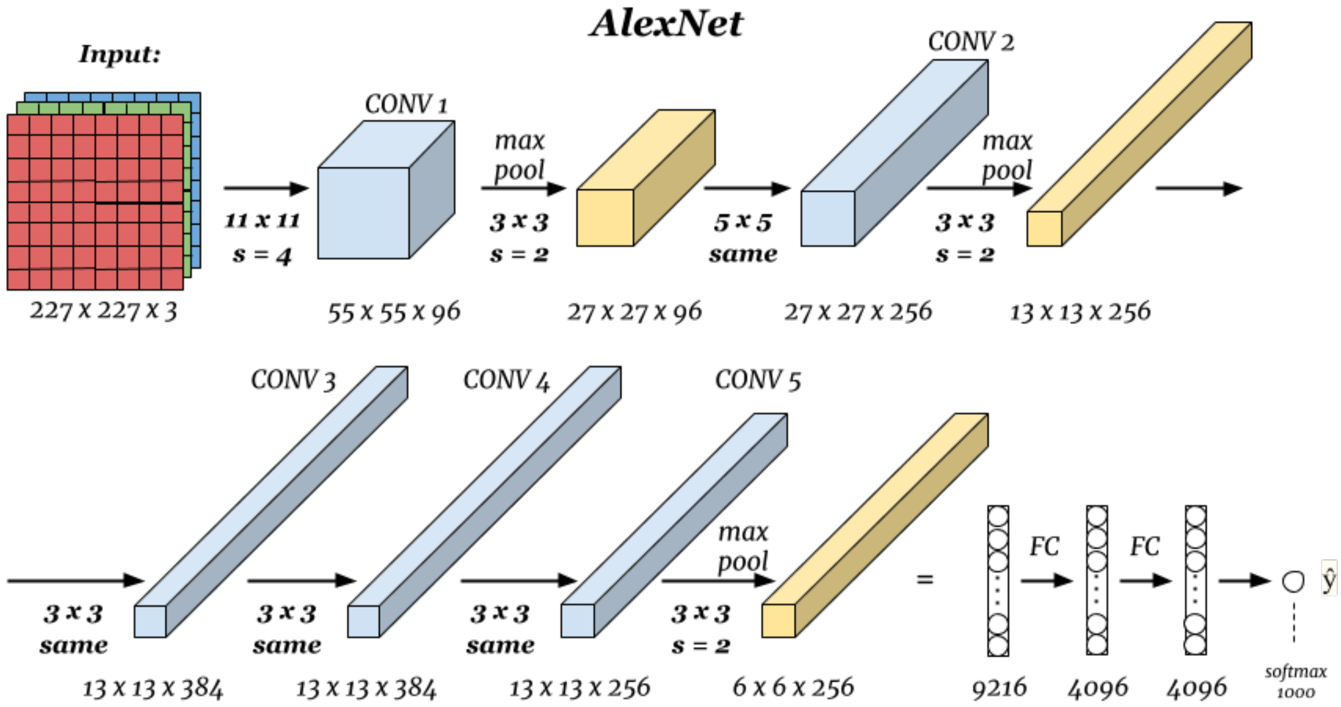
One example of classic networks is LeNet-5, from *Gradient-Based Learning Applied to Document Recognition* paper by Y. Lecun, L. Bottou, Y. Bengio and P. Haffner (1998):



- Number of parameters: ~ 60 thousands.

Classic Network: AlexNet

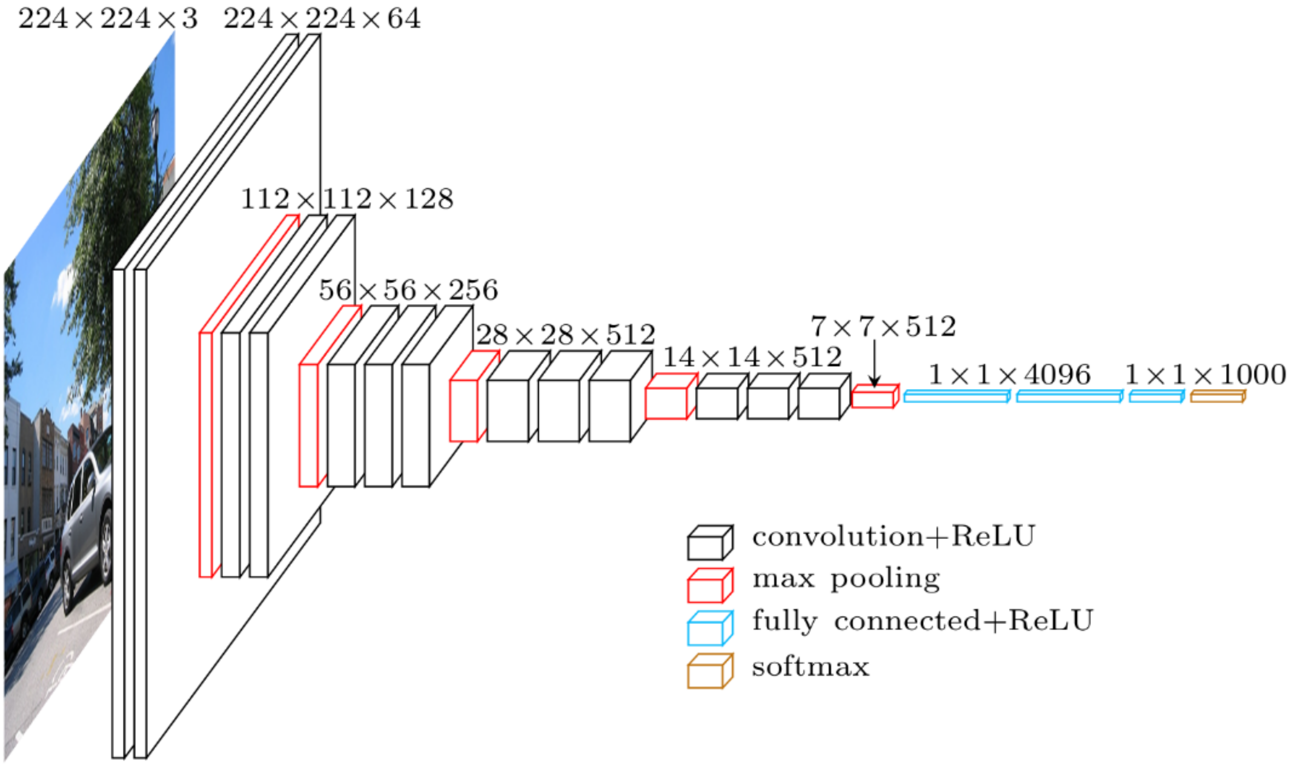
AlexNet is another classic CNN architecture from *ImageNet Classification with Deep Convolutional Neural Networks* paper by Alex Krizhevsky, Geoffrey Hinton, and Ilya Sutskever (2012).



- Number of parameters: ~ 60 millions.

Classic Network: VGG-16

VGG-16 from *Very Deep Convolutional Networks for Large-Scale Image Recognition* paper by Karen Simonyan and Andrew Zisserman (2014). The number 16 refers to the fact that the network has 16 trainable layers (i.e. layers that have weights).

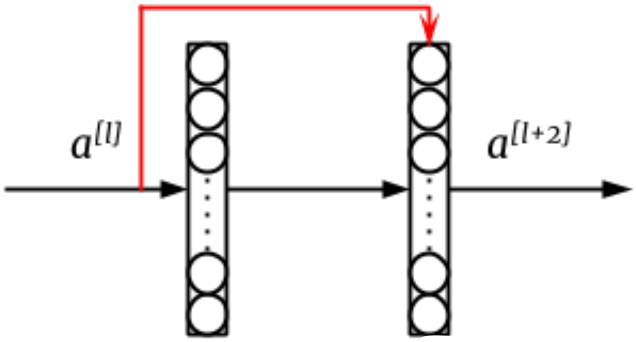


(image from blog.heuritech.com)

- Number of parameters: ~ 138 millions.
- The strength is in the simplicity: the dimension is halved and the depth is increased on every step (or stack of layers)

ResNet

The problem with deeper neural networks are they are harder to train and once the number of layers reach certain number, the training error starts to raise again. Deep networks are also harder to train due to exploding and vanishing gradients problem. ResNet (Residual Network), proposed by He at all in *Deep Residual Learning for Image Recognition paper* (2015), solves these problems by implementing skip connection where output from one layer is fed to layer deeper in the network:



In the image above, the skip connection is depicted by the red line. The activation $a^{[l+2]}$ is then calculated as:

$$z^{[l+2]} = W^{[l+2]}\, a^{[l+1]} + b^{[l+2]}$$

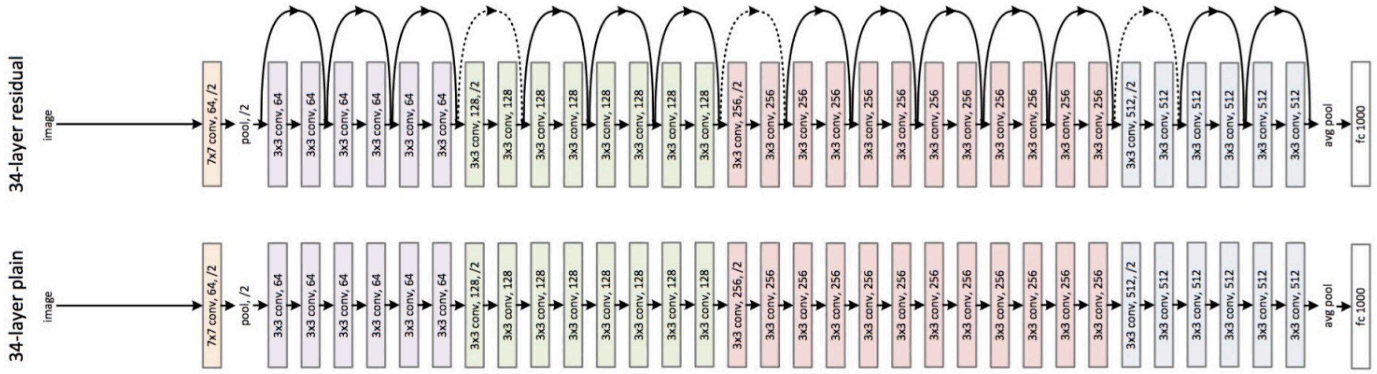
$$a^{[l+2]} = g^{[l+2]}(z^{[l+2]} + a^{[l]})$$

The advantages of ResNets are:

- performance doesn't degrade with very deep network
- cheaper to compute
- ability to train very very deep network

ResNet works because:

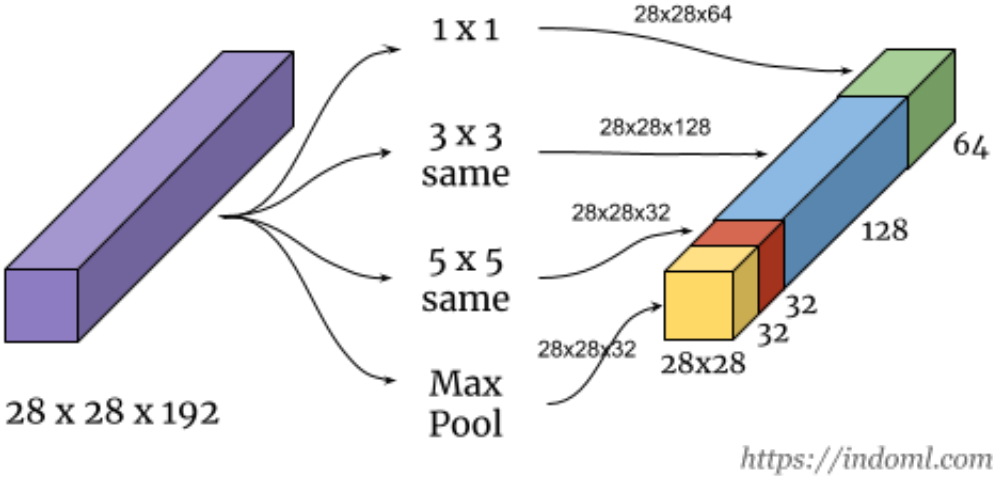
- identify function is easy for residual block to learn
- using a skip-connection helps the gradient to back-propagate and thus helps you to train deeper networks



Comparison of 34 layers ResNet with plain network (image from euler.stat.yale.edu)

Inception

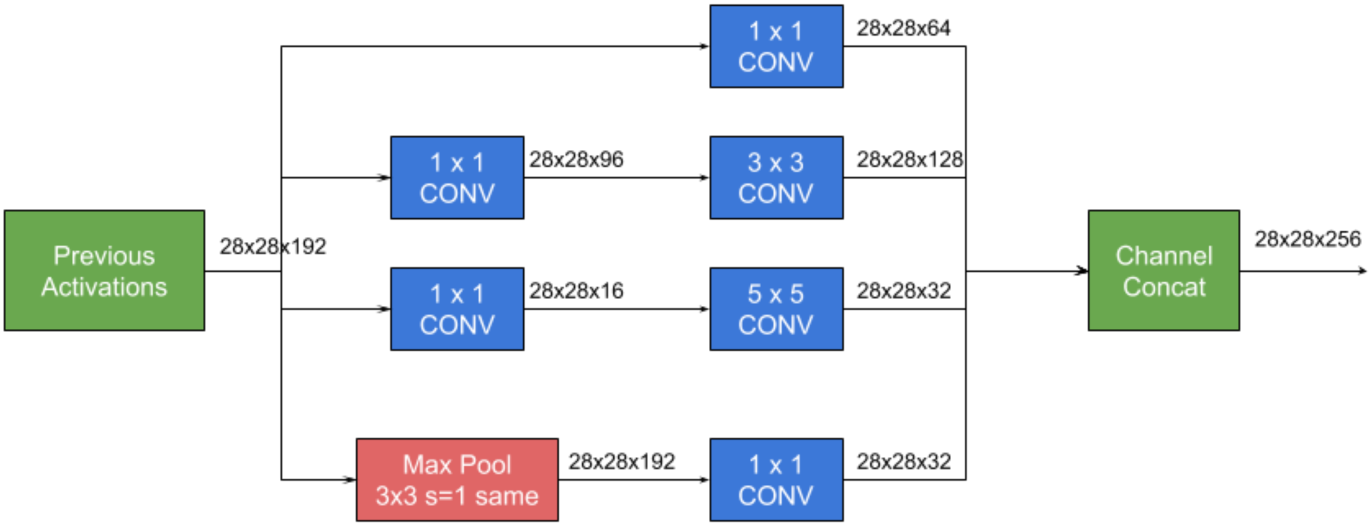
The motivation of the inception network is, rather than requiring us to pick the filter size manually, let the network decide what is best to put in a layer. We give it choices and hopefully it will pick up what is best to use in that layer:



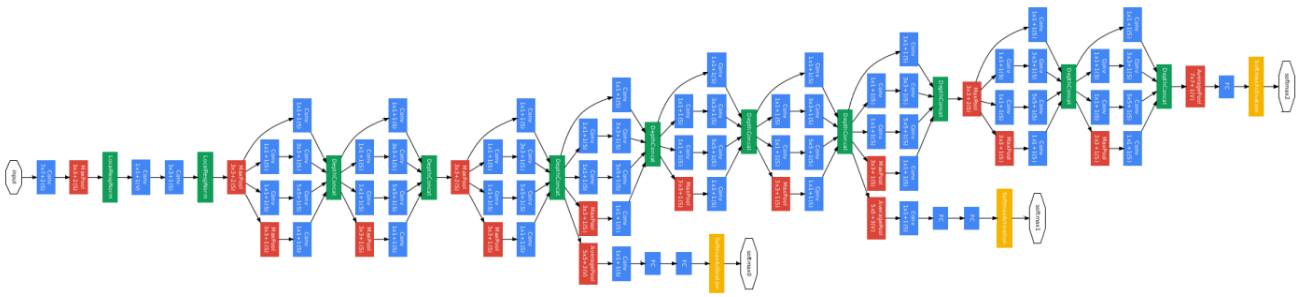
https://indoml.com

The problem with the above network is computation cost (e.g. for the 5 x 5 filter only, the computation cost is (28 x 28 x 32) x (5 x 5 x 192) = ~ 120 millions).

Using 1 x 1 convolution will reduce the computation to about 1/10 of that. With this idea, an inception module will look like this:



Below is an inception network called **GoogLeNet**, described in *Going Deeper with Convolutions paper* by Szegedy et all (2014), which has 9 inception modules:



GoogLeNet architecture (image source)

Data Augmentation

- Mirroring
- Random cropping
- Less common (perhaps due to their complexity):
  - Rotation
  - Shearing
  - Local warping
- Color shifting

Tips on Doing Well on Benchmarks/ Winning Competitions

- Ensemble
  - train several networks independently and average their outputs
  - can increase performance by 0.5 – 2%. Too little and too expensive.
- Multi-crop at test time
  - run classifier on multiple versions (say 10) of test images and average the results