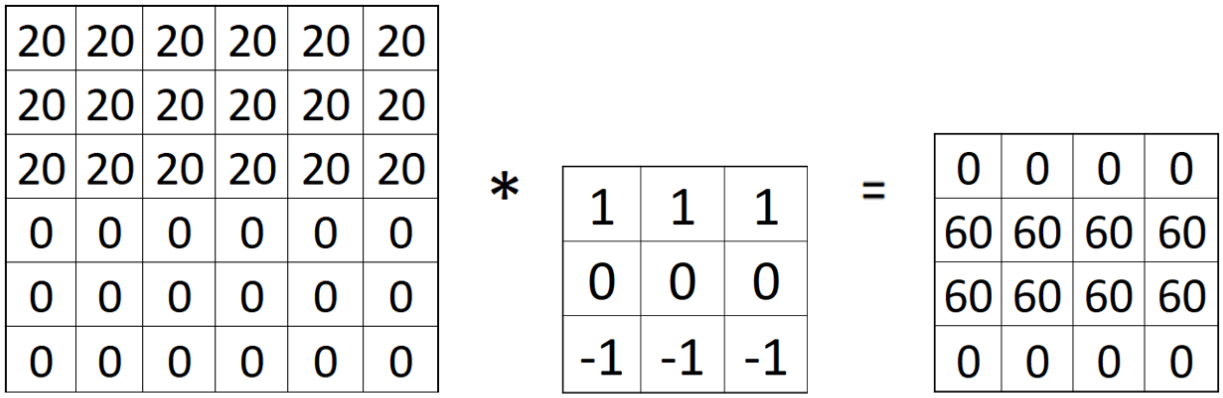
Convolution Output Image Dimensions

Let's look at the below example of a simplified image! The horizontal kernel/filter detects an edge in the input image.



A 6x6 Input Image is convolved with a 3x3 Filter and the Output Image is 4x4. To generalize this, if a **I x I** input image is convolved with a **f x f** filter, the output image is of size:

$$(I \times I) * (f \times f) = (I - f + 1) \times (I - f + 1)$$

Padding

There are two issues with the convolution operation:

1. If you perform multiple convolution operations consecutively, the final image might become vanishingly small because the output image shrinks with every convolution operation.
2. When a filter moves over original images, it slides over the edge of the image fewer times than the middle of the image. So, the corner/edge features of any image aren't used much in the output i.e., some information is lost.

Here, we introduce ***Padding*** which describes the addition of empty pixels around the edges of an image. The purpose of padding is to preserve the original size of an image when applying a convolutional filter and enable the filter to perform full convolutions on the edge pixels.

Below is an example of an input image with padding = 1.

	20	20	20	20	20	20	
	20	20	20	20	20	20	
	20	20	20	20	20	20	
	0	0	0	0	0	0	
	0	0	0	0	0	0	
	0	0	0	0	0	0	

The most common type of padding is ***Zero Padding***, where the image is padded with pixels of 0 value.

0	0	0	0	0	0	0	0
0	20	20	20	20	20	20	0
0	20	20	20	20	20	20	0
0	20	20	20	20	20	20	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Now, if we perform convolution after padding,

0	0	0	0	0	0	0	0
0	20	20	20	20	20	20	0
0	20	20	20	20	20	20	0
0	20	20	20	20	20	20	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Input ($I \times I$)

$*$

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

Filter ($f \times f$)

$=$

-40	-60	-60	-60	-60	-40
0	0	0	0	0	0
40	60	60	60	60	40
40	60	60	60	60	40
0	0	0	0	0	0
0	0	0	0	0	0

Output ($O \times O$)

A 6x6 Input Image with padding = 1 is convolved with a 3x3 Filter and the Output Image is 6x6. To generalize this, if a **I x I** input image with padding **p** is convolved with a **f x f** filter, the output image is of size:

$$(I \times I) * (f \times f) = (I + 2p - f + 1) \times (I + 2p - f + 1)$$

How do we discern how many pixels we need to pad an input image with to maintain its size after convolution? This depends entirely on the size of the filter. The most commonly used filter sizes are 3×3, 5×5, and 7×7. If your filter size is odd, you can calculate the pixels you need on each side i.e. padding, **p**:

$$p = (f - 1)/2$$

Stride

The stride simply describes the step size when sliding the convolutional filter over the input image. In the previous examples, we've always slid the filter by one pixel rightwards or downwards. We've used a stride of 1. With a stride of 2, we would slide the window by two pixels on each step.

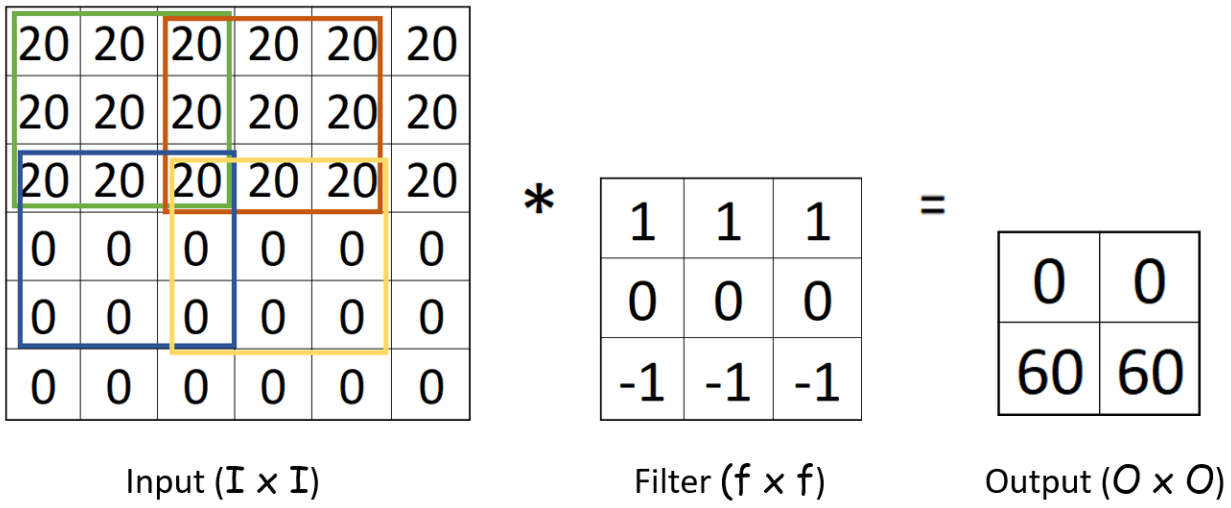
20	20	20	20	20	20
20	20	20	20	20	20
20	20	20	20	20	20
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

By taking larger steps, we will reach the end of the image in fewer steps. As a consequence, the resulting output image will be smaller i.e.. applying a larger stride has the effect of downsampling the image so that lower level details are obscured. It also helps reduce the

number of computations required. Therefore, if our neural network only requires an understanding of higher level features, we can make the learning process computationally more efficient by choosing a larger stride.

Note: These considerations stem from the early days of deep learning when computational power was a major obstacle to efficient neural network training. With the ability to train neural networks on large and extremely potent GPU clusters in the cloud, increasing the stride to improve computational efficiency has become largely unnecessary. In practice, many modern deep learning practitioners use a stride of 1.

Now, if we perform convolution with stride,



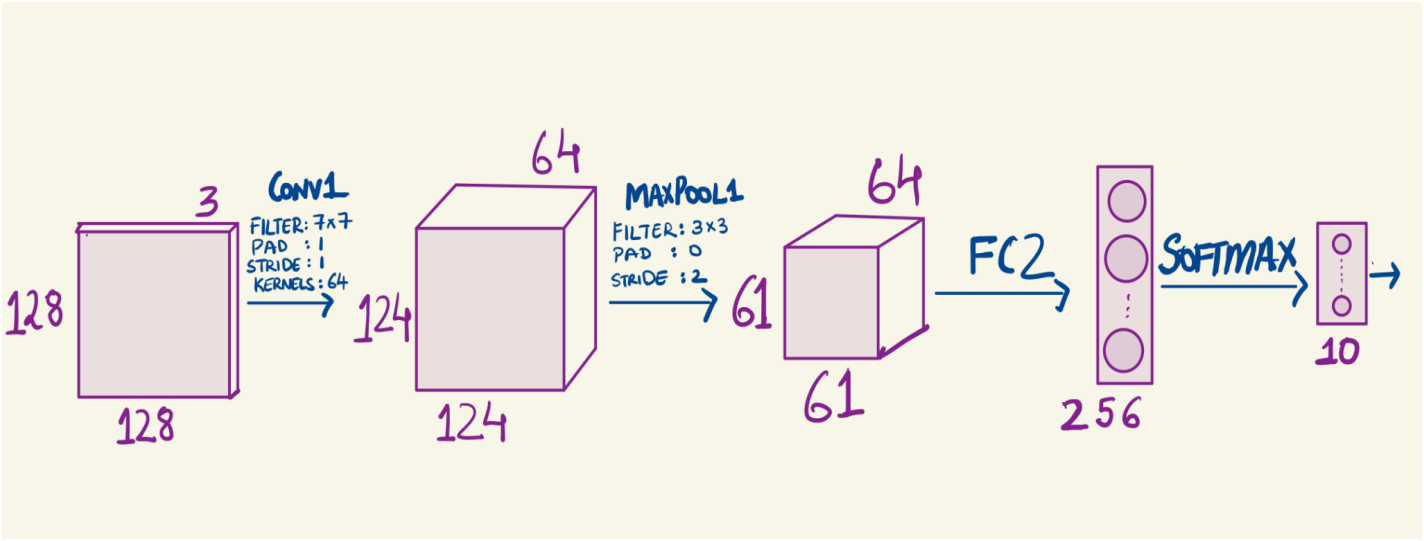
A 6x6 Input Image with no padding and stride = 2 is convolved with a 3x3 Filter and the Output Image is 2x2. To generalize this, if a **I x I** input image with padding **p** and stride **s** is convolved with a **f x f** filter, the output image is of size:

$$(I \times I) * (f \times f) = ((I + 2p - f)//s + 1) \times ((I + 2p - f)//s + 1)$$

Note: '//' represents floor division.

Number of Parameters Learnt

Consider the Convolutional Neural Network (CNN) below. A 128x128 RGB Image is first convoluted with a 7x7 Filter, followed by a 3x3 MaxPooling operation. Finally, we have a Dense Layer (Fully Connected) of 256 neurons, followed by an output Dense Layer of 10 neurons.



To understand the output dimensions of each output, refer to the section above. For example, the output after CONV1 = ((128 + 2*1 - 7)//1 + 1) X ((128 + 2*1 - 7)//1 + 1) = 124x124 with 64 kernels/channels as the third dimension.

Parameters are weights that are learnt during training. In CNNs, the parameters are the filters/kernels' values i.e., they are weight matrices that contribute to the model's predictive power, changed during back-propagation process.The below table summarizes the number of neurons and the number of parameters learnt at every layer of the example CNN.

Sl No.	Layer	Shape	# Neurons	# Parameters
1	Input	(128, 128, 3)	49152	0
2	CONV1 (filter: 7x7)	(124, 124, 64)	984,064	9472
3	POOL1	(61, 61, 64)	238,144	0
4	FC2	(256, 1)	256	60,965,120
5	Softmax	(10, 1)	10	2570

1. **Input Layer** - The input layer has no learnable parameters.

2. **CONV Layer** - Multiply the dimensions of the filter **mxn** by the number of filters/kernels/channels in the previous layer **d** and account for all such filters/kernels/channels **k** in the current layer. Don't forget the bias term for each of the filters by adding 1. Then the number of parameters in a CONV layer would be :

$$\#\ \text{parameters} = ((m * n * d) + 1) * k)$$

   Therefore, for the CONV1 layer in the above example, the number of parameters learnt = **((7 * 7 * 3) + 1) * 64 = 9472**

3. **POOL Layer** - The goal of pooling is to reduce the overall amount of information in an image, while maintaining the features that are detected as present. No parameters are learned.

4. **Fully Connected Layer (FC)** - Since these layers comprise connections between each and every neuron in the previous and current layer, the number of parameters depend on the number of neurons. Therefore, the number of parameters is the product of the number of neurons in the current layer **c** and the number of neurons in the previous layer **p** and as always, do not forget the bias term (represented by 1). Thus, the number of parameters here are:

$$\#\ \text{parameters} = ((c * p) + 1 * c)$$

   Therefore, for the FC2 layer in the above example, the number of parameters learnt = **(256 * 238144) + 1*256 = 60,965,120**

We see that the number of parameters learnt by dense layers is much larger than convolutional layers!