PadhAI Week 8: The Convolution Operation

by Manick Vennimalai

PadhAI Week 8: The Convolution Operation

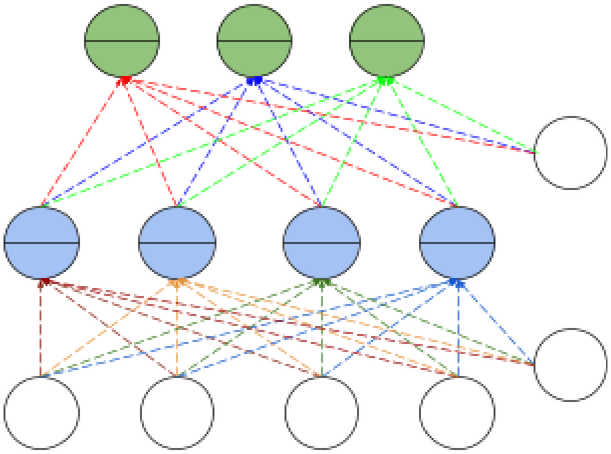
by Manick Vennimalai

8.1: The Convolution Operation

8.1.1: Setting the Context

Making sense of everything we have seen so far

1. So far, all the Neural Networks that we have studied about are called fully connected networks



2. **Fully Connected Neural Networks**: Any neuron in a given layer is fully connected to all the neurons in the previous layer
3. Let's look at at some of the pros and cons of Fully Connected Neural Networks (DNNs)

Pros	Cons
The Universal Approximation Theorem says that DNNs are power function approximators We can come up with a neural network of output $\hat{f}(x)$ which is very close to the true output $f(x)$	DNNs are prone to overfitting (too many parameters) Even a slight change in the training set could cause the model to arrive at very different weight configurations
Can be trained using backpropagation In PyTorch, backpropagation is automated.	Gradients can vanish due to long chains Vanishing gradient problem could occur in the case of saturated neurons

4. We aimed to mitigate these issues using
   a. Better Optimization Algorithms
   b. Better Activation Functions
   c. Better Initialisation Methods
   d. Better Regularization
5. Can we have DNNs which are complex (many non-linearities) but have fewer parameters and hence less prone to overfitting?

PadhAI Week 8: The Convolution Operation

by Manick Vennimalai

8.1.2: The 1D convolution operation

What does the convolution operation do?

1. Let's approach this with a real world example
2. Consider a flight from Chennai to Delhi
   a. We measure the distance of the flight from Chennai at regular intervals,
      i. $x_0 \; at \; t_0$
      ii. $x_1 \; at \; t_1$
      iii. $x_2 \; at \; t_2$
   b. In general, to calculate the overall speed, we would take the average speed at these measured points i.e $\frac{1}{3}(x_0 + x_1 + x_2)$.
   c. However, let us try giving the most importance to the current reading, and a progressively decreasing level of importance to every reading preceding the current one.
   d. Let's assign different weights to each of these reading points
      i. x_0 --> w_0 (0 indicates current reference point)
      ii. x_1 --> w_{-1} (1 reading before reference point)
      iii. x_2 --> w_{-2} (1 readings before reference point)
   e. So the new overall speed would be calculated by $w_{-2}x_0 + w_{-1}x_1 + w_0x_2$ where the weights are decreasing from w_0
3. The formula could be written as follows
   a. $s_t = \sum_{a=0}^{\infty} w_{-a}x_{t-a} = (x * w)_t$
   b. Where t refers to reference point
   c. a is the index of the weight, ranging from 0 for reference point to ∞
4. In practice, we wouldn't want to take the reading up till $-\infty$, thus we can simply say that those unwanted weights are all 0.
5. Consider the following table

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0					
W	0.01	0.01	0.02	0.02	0.04	0.04	0.05					
X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80					

6. In the above table, w_{-7} to $w_{-\infty}$ are all consider to be 0
7. Here, $s_6 = x_6w_0 + x_5w_{-1} + x_4w_{-2} + x_3w_{-3} + x_2w_{-4} + x_1w_{-5} + x_0w_{-6}$

PadhAI Week 8: The Convolution Operation

by Manick Vennimalai

8.1.3: The 2D convolution operation

What about 2D inputs? What are the neighbors that we consider?

1. In a nutshell, the convolution operation boils down to taking a given input and re-estimating it as a weighted average of all the inputs around it.
2. The above definition is easy to visualise in 1D, but what about 2D?
3. In 2D, we would consider neighbors along the rows and columns, using the following formula
4. $s_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i+a,\, j+b} * K_{a,b}$
    a. K refers to kernel or weights and I refers to the input. And * refers to the convolution operation

a = rows

K

K_{00}	K_{01}
K_{10}	K_{11}

b = cols

    b. Let a be the number of rows and b be the number of columns
    c. m & n specify the size of the matrix, in this case we consider them to be 2 each. So it's a 2x2 matrix. Therefore a & b range from 0-1 each.
    d. Now, to calculate the new value at a particular pixel $I_{i,j}$, we simply need to fill in the values ino the formula.
    e. $s_{ij} = I_{i+0,j+0}K_{0,0} + I_{i+0,j+1}K_{0,1} + I_{i+1,j+0}K_{1,0} + I_{i+1,j+1}K_{1,1}$
    f. Here is a pictorial representation

a	b	c	d
e	f	g	h
i	j	k	l

Input

w	x
y	z

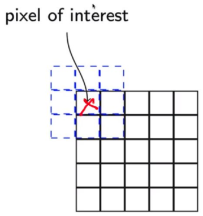
Kernel

Convolution ->

aw+bx+ey+fz	bw+cx+fy+gz	cw+dx+gy+hz
ew+fx+iy+jz	fw+gx+jy+kz	gw+hx+ky+lz

Output

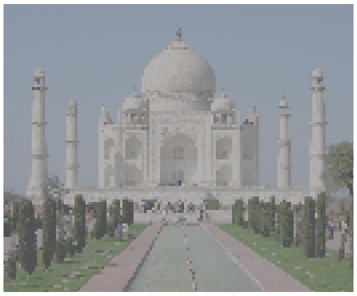
    g. This is how the convolutional operation looks like in 2D
    h. Instead of only choosing successive points, we must also consider previous points, on both sides of the reference pixel. $s_{ij} = (I * K)_{ij} = \sum_{a=[-\frac{m}{2}]}^{[\frac{m}{2}]} \sum_{b=[-\frac{n}{2}]}^{[\frac{n}{2}]} I_{i-a,\, j-b} * K_{\frac{m}{2}+a,\, \frac{n}{2}+b}$



8.1.4: Examples of 2D convolution

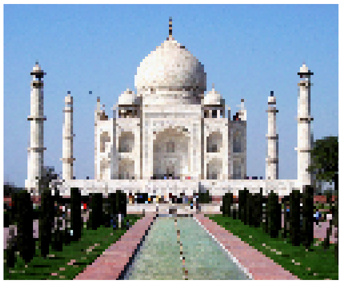
How is the convolution operation used in practice?

1. Let us consider a 3x3 kernel and run it over an image, pixel-by-pixel.
2. This is done to re-estimate every pixel in that 3x3 neighborhood

Input 30 x 30	conv	Kernel 3x3	Output 30x30 (blur)
	*	$\begin{vmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{vmatrix}$	

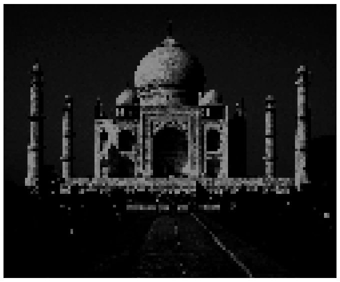
   a. Here, we can see that the kernel is essentially an average operation, so what it does is it converts the value of every pixel to $\frac{1}{9}^{th}$ of its original value.
   b. In any photo editing tool like GIMP or Photoshop, when we select an image blur, we are essentially performing a convolution operation using an average valued kernel.

3. Let's look at another convolution operation

Input 30 x 30	conv	Kernel 3x3	Output 30x30 (sharpens)
	*	$\begin{vmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{vmatrix}$	

   a. Here, the selected pixel is magnified by multiplying by 5 and then we subtract the 4 neighbors from it. This results in a sharper image, as it boosts the current pixel, thereby making it appear more prominent when compared to its neighbors.

4. Let's look at one more example

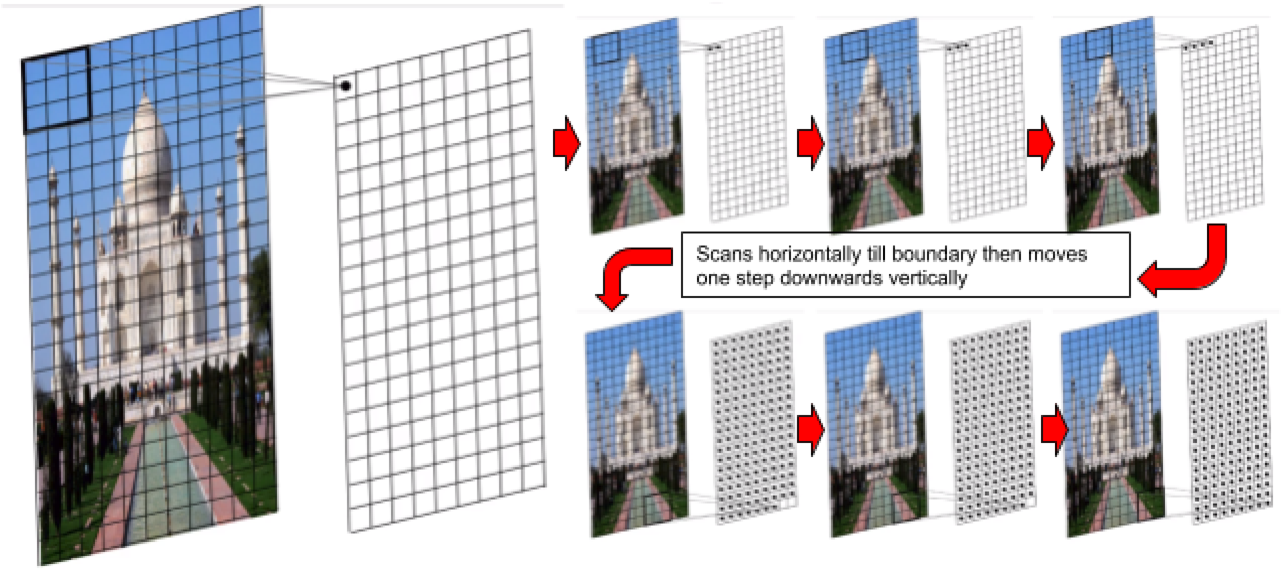
Input 30 x 30	conv	Kernel 3x3	Output 30x30 (Edge detection)
	*	$\begin{vmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{vmatrix}$	

   a. Here, pixels near others pixels of the same value are reduced to 0, leaving only the edges.

8.1.5: 2D convolution with a 3D filter

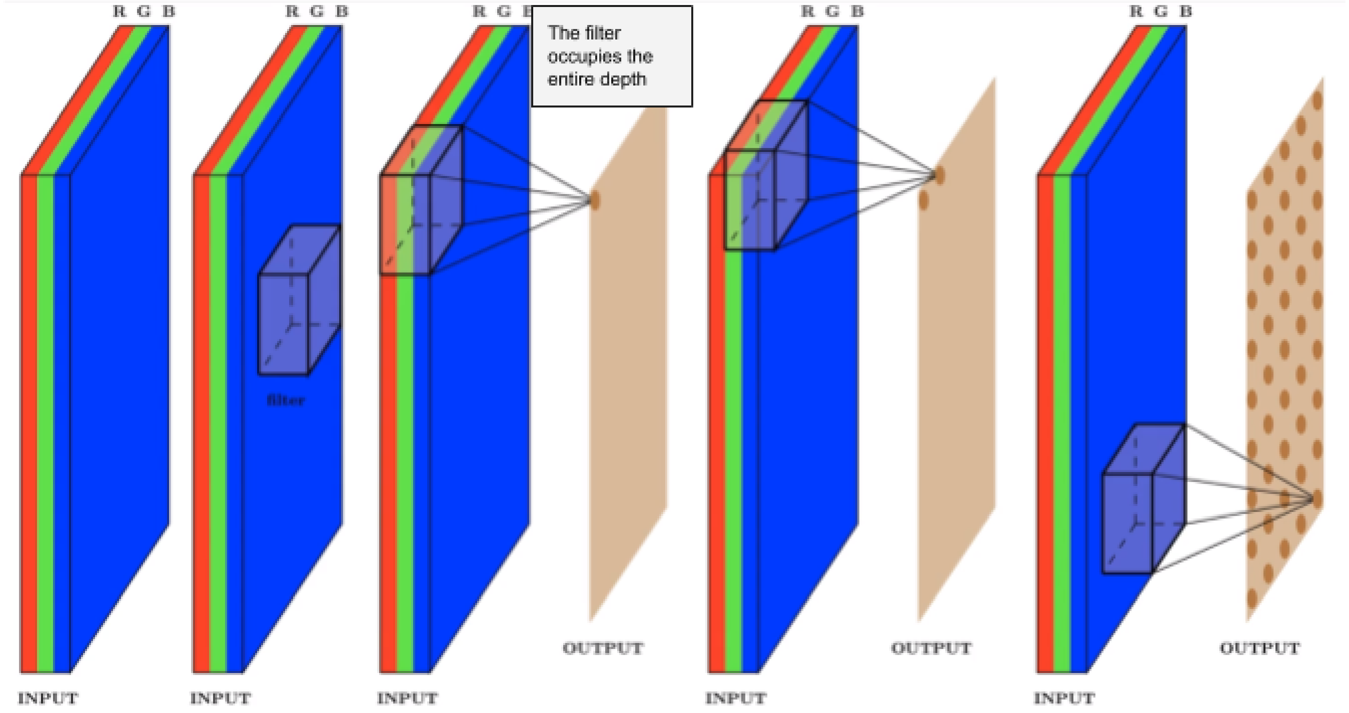
How is this operation performed over the entire image?

1. The following diagram illustrates how the kernel scans through the entire image and applies the transformation



Scans horizontally till boundary then moves one step downwards vertically

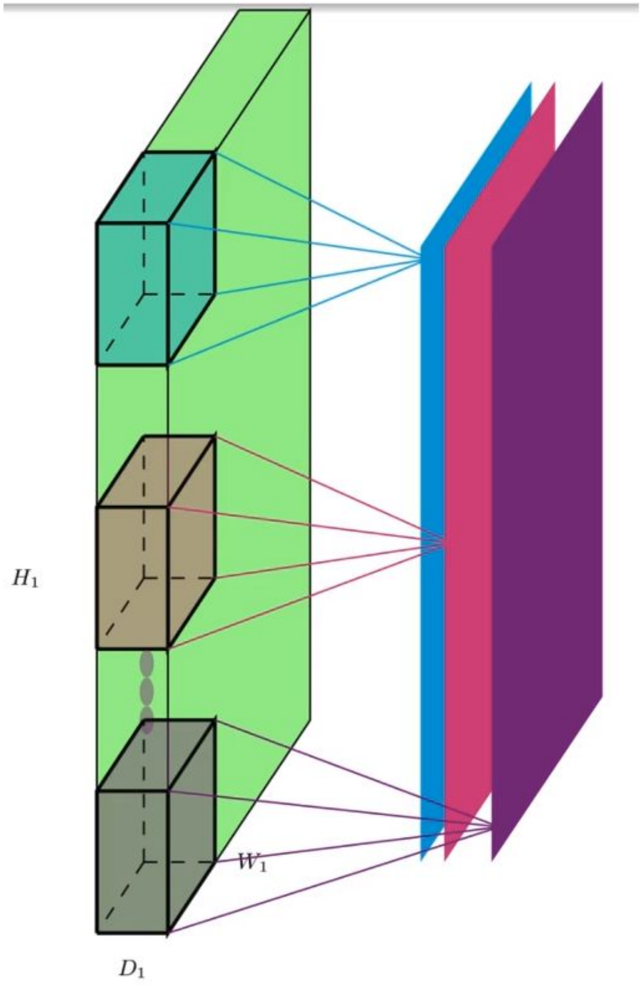
2. How do we do this in the case of a 3D input?
3. The fact is, all the images that we have been considering till now are all 3D inputs, i.e. each pixel is associated with 3 values (Red, Green and Blue). So let’s take a look at how the convolution operation takes place in 3D



The filter occupies the entire depth

4. Here, since the filter is of the same depth as the image, there is no movement along the depth axis, essentially it moves along the horizontal & vertical axes just as we have seen before.

5. Some important points about the 3D convolution Operation are:
   a. The input is 3D
   b. The filter is also 3D
   c. The Convolution operation that we perform is 2D
   d. We only slide vertically & horizontally and not along the depth
   e. This is because the depth of the filter is the same as the depth of the input.
6. We can also apply multiple filters to the same image

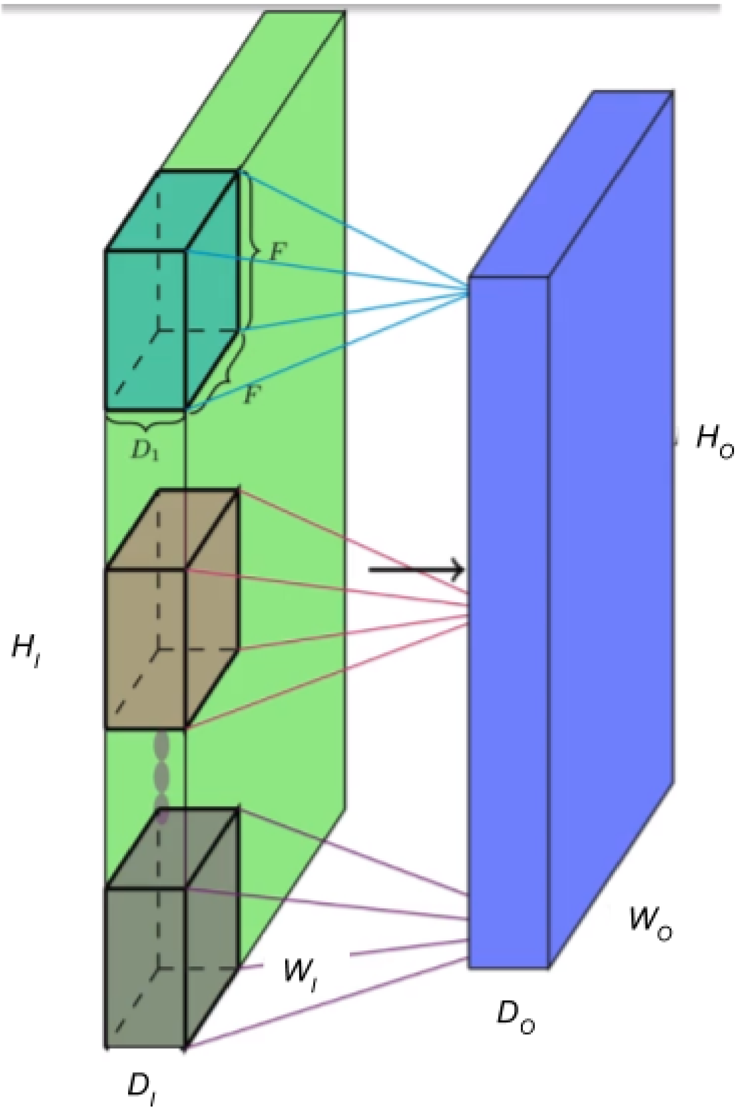


   a. Here, we can see how multiple filters are applied to the input volume(3D) to get an output area(2D).
   b. This is how it is commonly done in practice.
   c. It is called an input volume because it has 3 dimensions (width, height and depth)
   d. The output areas contain 2 dimensions (width and height)
   e. They can be stacked together to get an output volume,
   f. In this case, the depth of the output volume is 3, as we are stacking 3 output areas.

8.1.6: Terminology

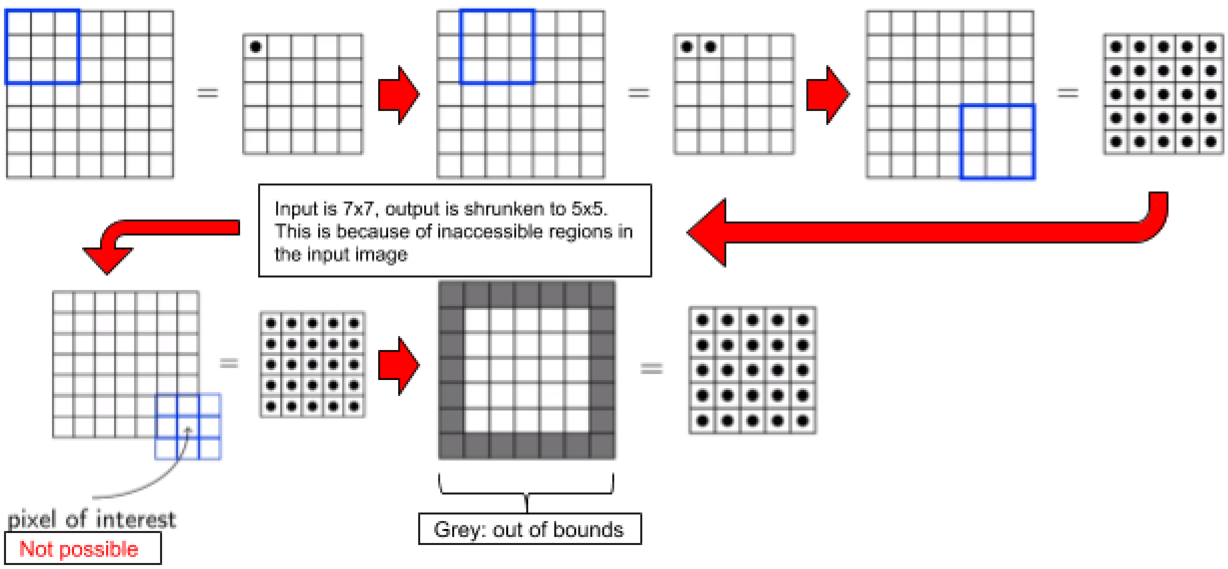
Let’s look at some terminology

1. Consider the following 3D convolution operation and look at the terminology associated with it



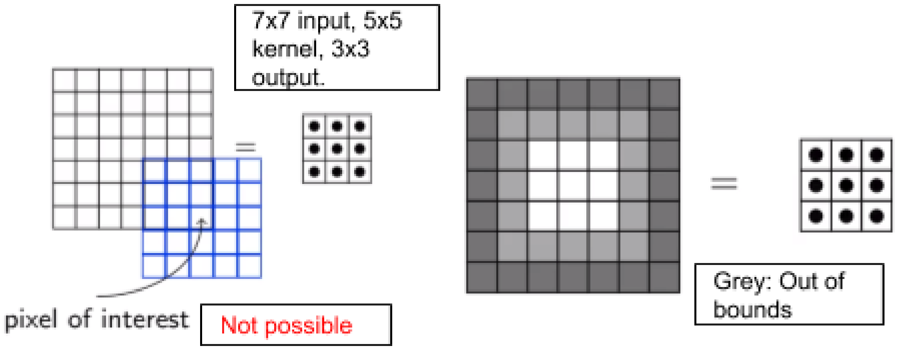
2. **Terminology**:
   a. Input Width (W_I), Height (H_I) and Depth (D_I)
   b. Output Width (W_0), Height (H_0) and Depth (D_0)
   c. The spatial extent of a filter (F), a single number to denote width and height as they are equal
   d. Filter depth is always the same as the Input Depth (D_I)
   e. The number of filters (K)
   f. Padding (P) and Stride (S)
3. **Question**: Given W_I, H_I, D_I, F, K, S and P how do you compute W_0, H_0, and D_0?

4. To answer that, let us look at a sample 3x3 kernel passing over a 7x7 image


Input is 7x7, output is shrunken to 5x5. This is because of inaccessible regions in the input image

   a. Here, we can see that by running the 3x3 kernel over a 7x7 image, we get a smaller 5x5 image.
   b. This is because we can’t place the kernel at the corners as it will cross the input boundaries
   c. This is true of all the shaded points.
   d. Hence the size of the output will be smaller than that of the input

5. Let’s see another example with a 5x5 kernel


7x7 input, 5x5 kernel, 3x3 output.

   a. Here, we can see that by running a 5x5 kernel over a 7x7 input, we get a smaller 3x3 image
   b. Here, the out-of-bounds regions are larger.
   c. Thus the output is much smaller.
6. We can see that the reduction in size can be given by the following equations
   a. $W_O = W_I - F + 1$
   b. $H = H_I - F + 1$
7. However in practice, we could still place the kernel on the boundary and take only the valid neighbors. This is roughly what is being done.

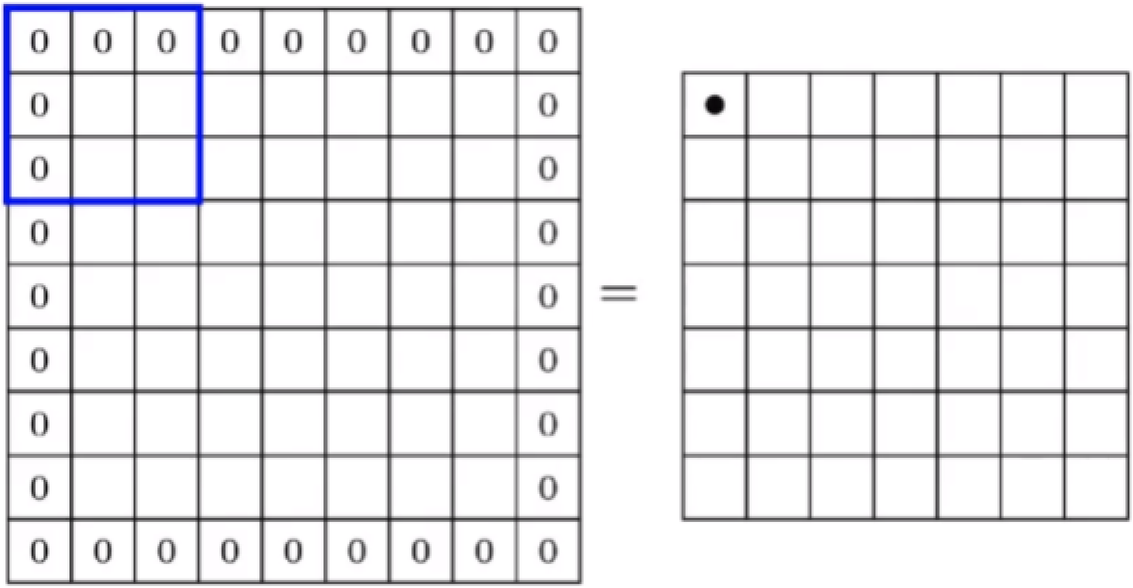
PadhAI Week 8: The Convolution Operation

by Manick Vennimalai

8.1.7: Padding and Stride

What if we want the output to be the same size as the input?

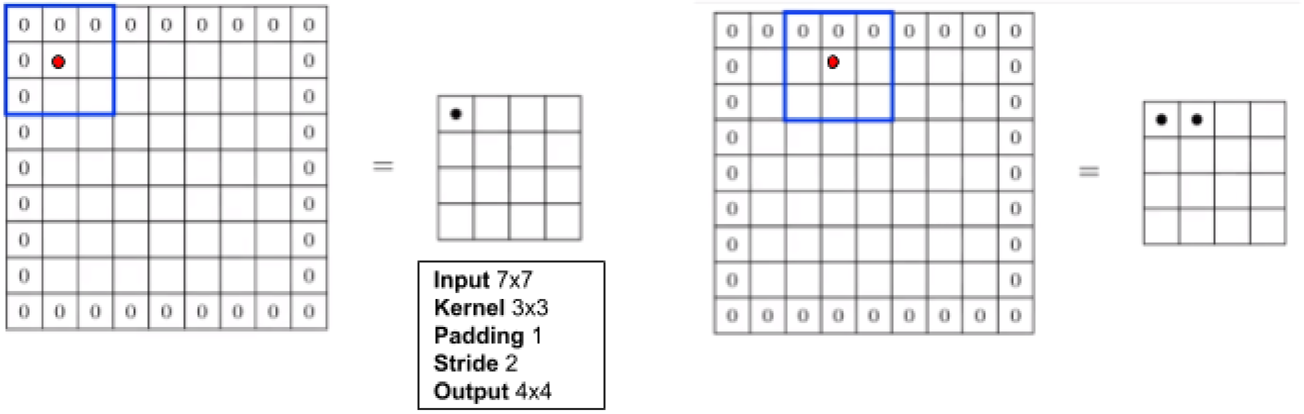
1. Let us consider adding extra rows+columns of zeros so that we can access all the image pixels



7x7 input, 3x3 kernel and padding of 1.
The zeros denote the padding layer of 1.

Here, output remains 7x7 when a 3x3 kernel is used. Need extra padding for 5x5 kernel.

   a. We can see that we must apply padding to preserve the output size
   b. The bigger the kernel size, the larger the padding required.
2. Thus, the formulae from the last section can be updates as follows
   a. $W_O = W_I - F + 2P + 1$
   b. $H = H_I - F + 2P + 1$
3. Another term that we use is called stride (S). It also affects the size of the output image.

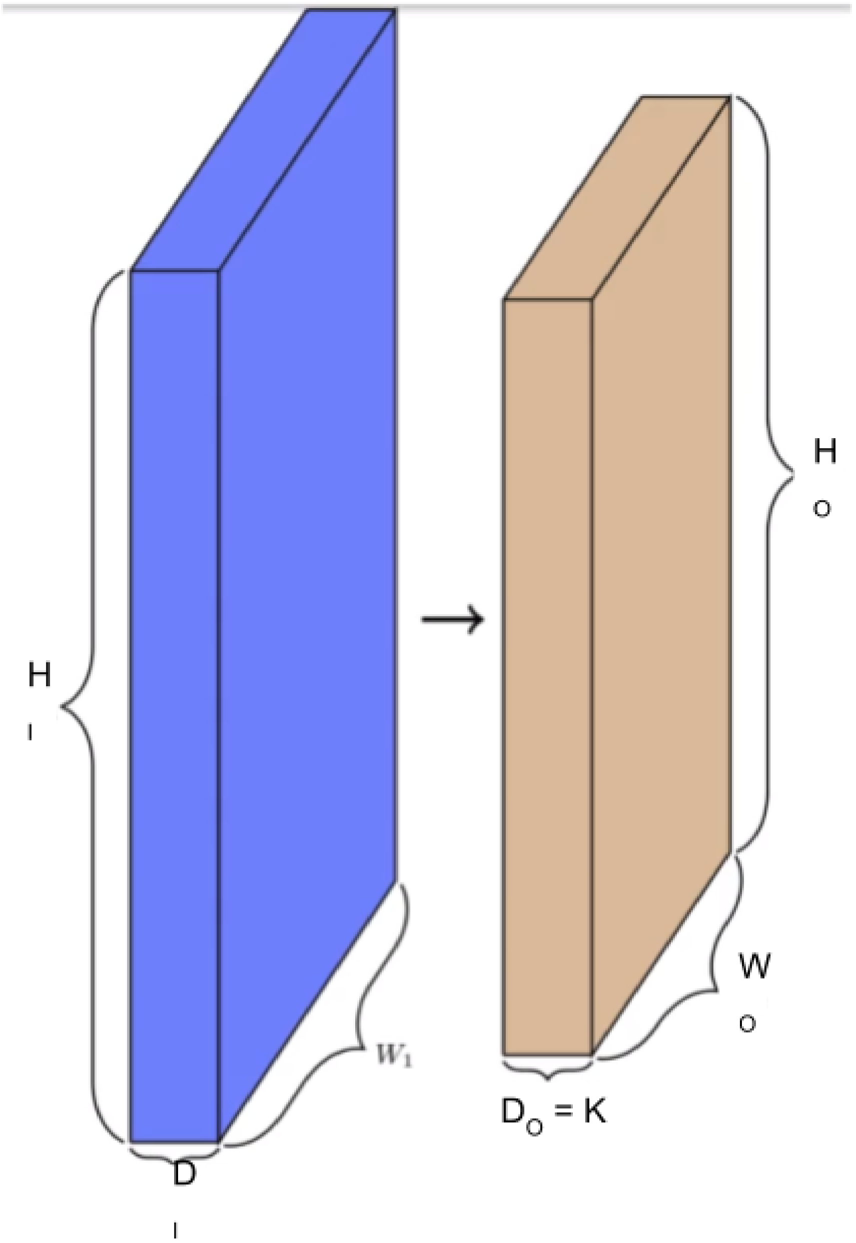


Input 7x7
Kernel 3x3
Padding 1
Stride 2
Output 4x4

   a. Stride defines the interval at which the filter is applied
   b. Higher the stride, the smaller the size of the output

4. We can see that the reduction in size can be given by the following equations
   a. $W_O = \frac{W_I - F + 2P}{S} + 1$
   b. $H_O = \frac{H_I - F + 2P}{S} + 1$

5. How do we compute the depth D of the output?
6. Consider the following image of a convolution operation



7. Each filter gives on 2D output
8. K filters will give K such 2D outputs
9. The depths of the output is the same as the number of filters
10. Thus, our final set of formulae are
    a. $W_O = \frac{W_I - F + 2P}{S} + 1$
    b. $H_O = \frac{H_I - F + 2P}{S} + 1$
    c. $D_O = K$