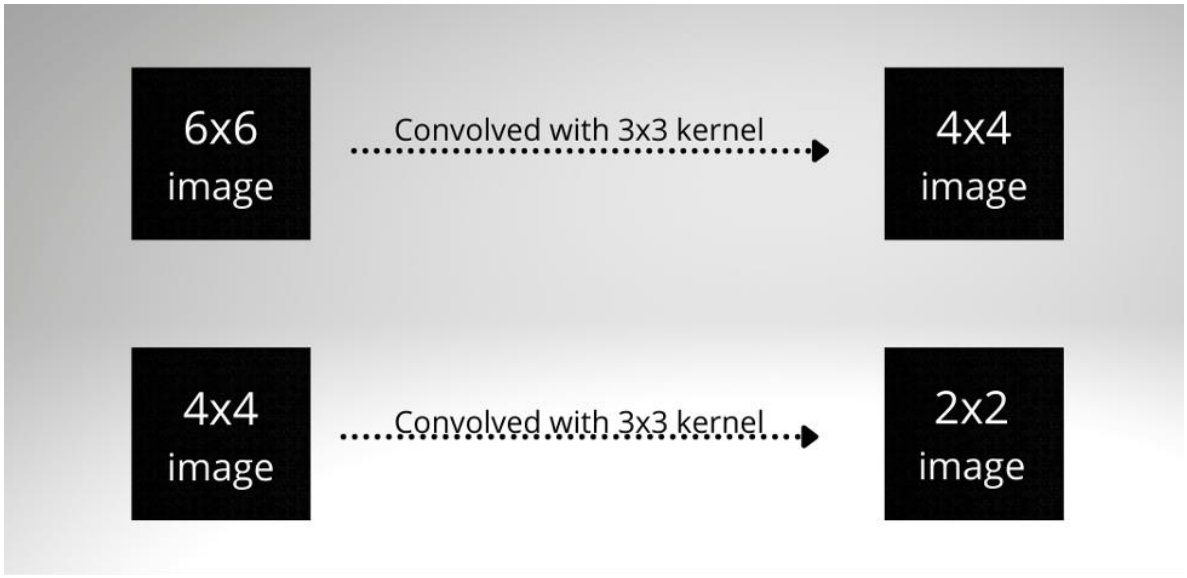
Module 4

As we know, while building a neural network, you should perform convolution to extract features. This is done with the help of kernels with respect to the current dataset, which is a vital practice to make your network learn while convoluting.

For example, if you want to train your neural network to classify whether it is a dog or cat then while applying convolution, kernels are going to extract features from the images like the ear of a dog & the tail of a dog to differentiate between dog and cat features. This is how convolution works. To get a better understanding of this concept, read our blog "Difference between Channels and Kernels in Deep Learning".



As you can see from the above image if you convolute 3 X 3 kernels over 6 x 6 input image size, the size of the image will be reduced by 2 as kernel size is 3 with output as 4 x 4. Further applying 3 x 3 on 4 x 4 size will be reduced to 2 x 2 as seen in the image above. However, if you apply 3 x 3, you cannot convolute further as the output image size is 2 x 2 now and mathematically you will get a negative dimension, which is not possible.

If you keenly observe, it may happen that your network does not consider some of the important features because you will end up with just 2 convolutions. This will make it difficult to extract all the features to differentiate between two objects that you are training the neural network with.

Now, the question arises - Why did image size reduce by 2 only?

We have a formula to calculate the output image size, which is:

$O = n - k + 1$

Module 4

In the above formula, **O stands for output image size, n is the input image size, and k is the kernel size.**

If you look at the above example, we have input image size as 6 x 6 with kernel size 3 x 3.

So, the output size after applying the first convolution will be **O** = 6 - 3 + 1 **O** = 4 x 4

We are sure you have a lot of questions, like "**Why do we always use odd kernels like 3 x 3 or 5 x 5 instead of even kernels like 4 x 4 kernels?**" or "**What if you want the same output image size as your input image size after convolution?**"

Here is where Padding comes into the picture

Why do we need padding?

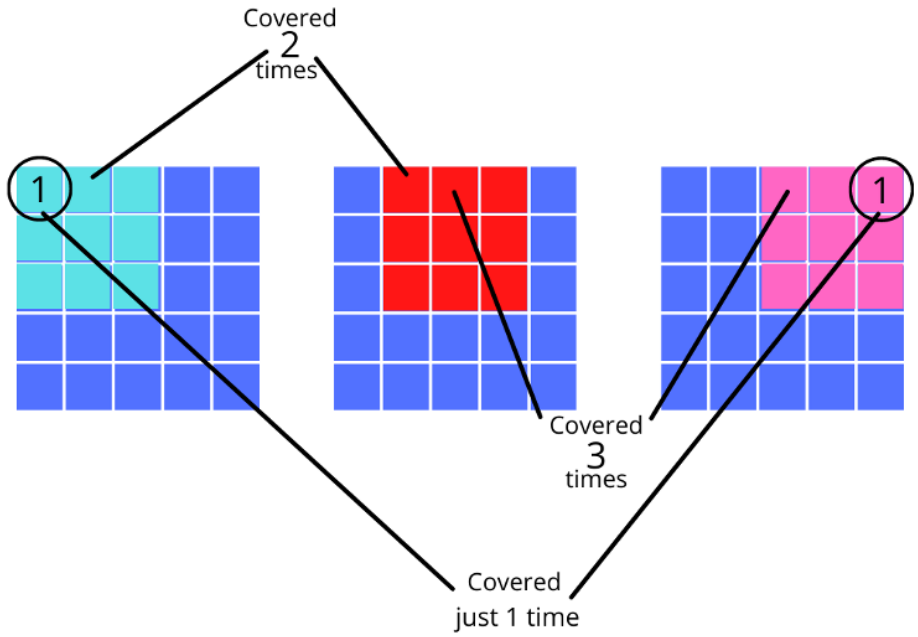
The main aim of the network is to find the important feature in the image with the help of convolutional layers. However, it may happen that some features are at the corner of the image which the kernel (feature extractor) visits very less number of times. Hence, there could be a possibility of missing out on some of the important information.

Padding is the solution where extra pixels are around the 4 corners of the image which increases the image size by 2. However, it should be done as neutrally as possible which means you should not alter the original image features in any way to make it complicated for the network to learn further. **Also, you can add more layers as now you have a larger image.**

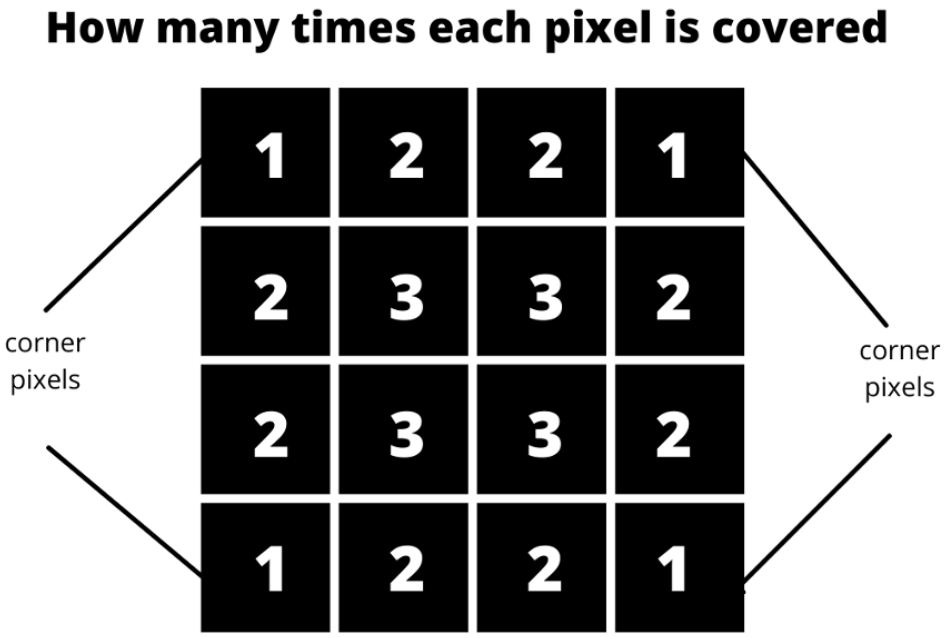
In simple terms, it can be said that padding helps the kernel (feature extractor) to visit pixels of the image around the corners more times to extract important features for better learning.

If you observe the image below, you will notice how a kernel visits the pixels of the image. It is important that a kernel visits each pixel at least more than once to extract the important features.

Module 4

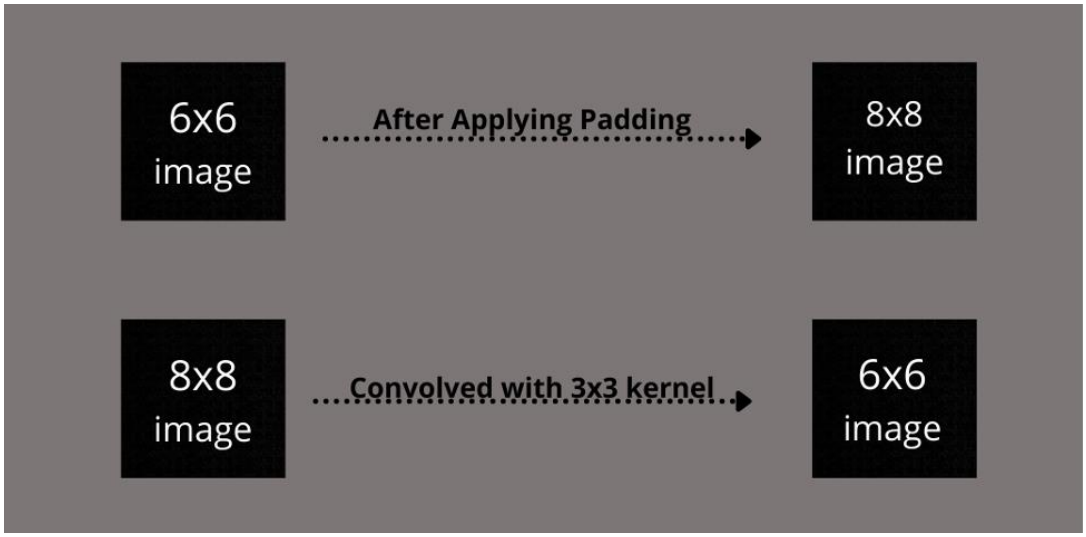


You can also notice in the above illustration that while convoluting the image, the pixels around the corners of the image are visited lesser number of times as compared to the pixel in the middle part.



Now, let's have a look at the image below while applying padding.

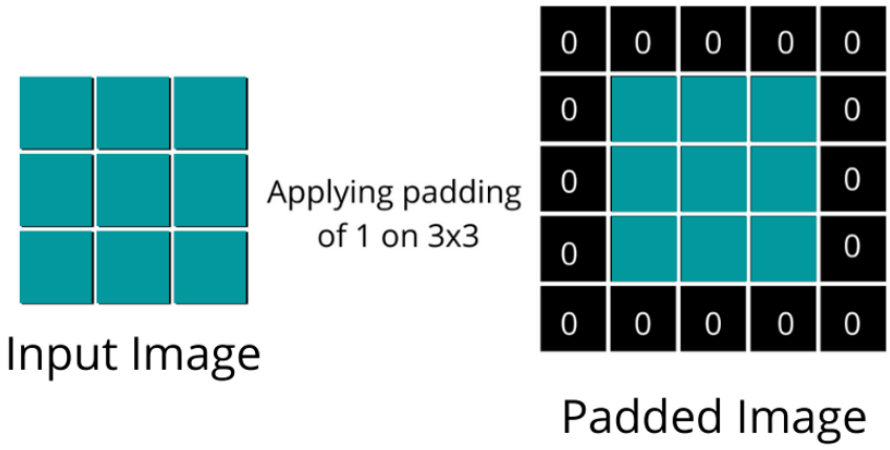
Module 4



If you can observe from the above picture, we have added the padding of zero around the 4 corners of the image and the input image size has increased from 6 x 6 to 8 x 8. After applying the first convolution, the output image size would be 6 x 6, which is what we want as per the question above.

After applying padding, you will observe that there is the possibility of convoluting one more time compared to without padding. This helps the network to extract more features for better learning.

The illustration below showcases what a padded image looks like.



There are two types of padding:

- Valid padding, where we do not apply padding at all and the value of P is 0.

- Same padding, where we apply padding and the output image size should be equal to the input image size.

As mentioned earlier, the formula for calculating output image is:

O = n - k + 1

Now, we are adding padding over the 4 sides of an image - down, up, right & left side adding zero padding. (NOTE: here we are not considering strides)

Let's look at a new formula with padding O = n - k + 1 +2p

For **same padding O = n (Output image size should be equal to input image size)**

So, O = O - k + 1 + 2p.

And from the above equation, we can simply derive the following formula: **P = (k - 1)/2**

Why do we always use ODD kernels and not EVEN kernels?

For instance, if you want to use a 4 x 4 kernel, then what is the padding that you will have to use?

You can use the above formula **p =(k - 1)/2**

We have kernel = 4 x 4 P = 4 -1/2 = 3/2 = 1.5

This means you have to add **1.5** pixels around 4 corners of the image to apply padding, which is not possible because we can only add pixels in whole numbers around the corners of the image.

This is one of the reasons **why kernels are always an odd number**, so that we can add a whole number of pixels around the image to extract more features.

Why do we always use padding = 1 and what do we mean by saying that we are using a padding value of 1?

In all the standard networks, you will notice that the padding used is always 1. This is because, when you perform padding, you add some information by yourself and it is not a good idea to interfere with the network. The best features learnt by the network are self-learnt. Furthermore, interference increases the input size a lot and also makes the model heavy.

Module 4

Why "zero-padding"?

So far, we have covered how padding works and why it is important for a neural network.

However, one question always comes to our mind - why only 0 padding? Why can't we use 1-padding or any other number?

When we normalize an image, the pixel range goes from 0 to 1, and if it hasn't been normalized then the pixel range will be from 0 to 255.

If we consider both cases, we can see 0 is the minimum number. So, using 0 becomes a more generic way of doing this. Also, it makes sense computationally since we are considering the minimum number among those pixels.

However, it is not always 0,

Consider a case where we have values ranging from -0.3 to 0.3. In this case, the minimum number is -0.3, and if we use the padding of 0 around it, it would be gray instead of black.

The same case comes when we have an activation function like tanh, the value of which ranges from -1 to 1. In this case, the minimum value is -1., so we would be using (-1) padding instead of 0 padding, for adding black pixels around it. If we use 0 padding, then it would be a gray color boundary instead of black color.

However, here is the problem; it is not always possible to calculate what will happen in activation values, so we look for a neutral value that can satisfy most of the cases without problem, and this is where 0 padding saves us.

If we even consider the above-mentioned two ranges, we can see, the use of 0 padding would give a gray boundary instead of black, which is not a huge loss.

We can also call these cases, "The Minimum Padding Case", where we are considering the minimum value among activation maps.

Most of the time, we use relu in between layers. So, in the case of relu, 0 becomes the minimum value and that is why it is so popular and generic.

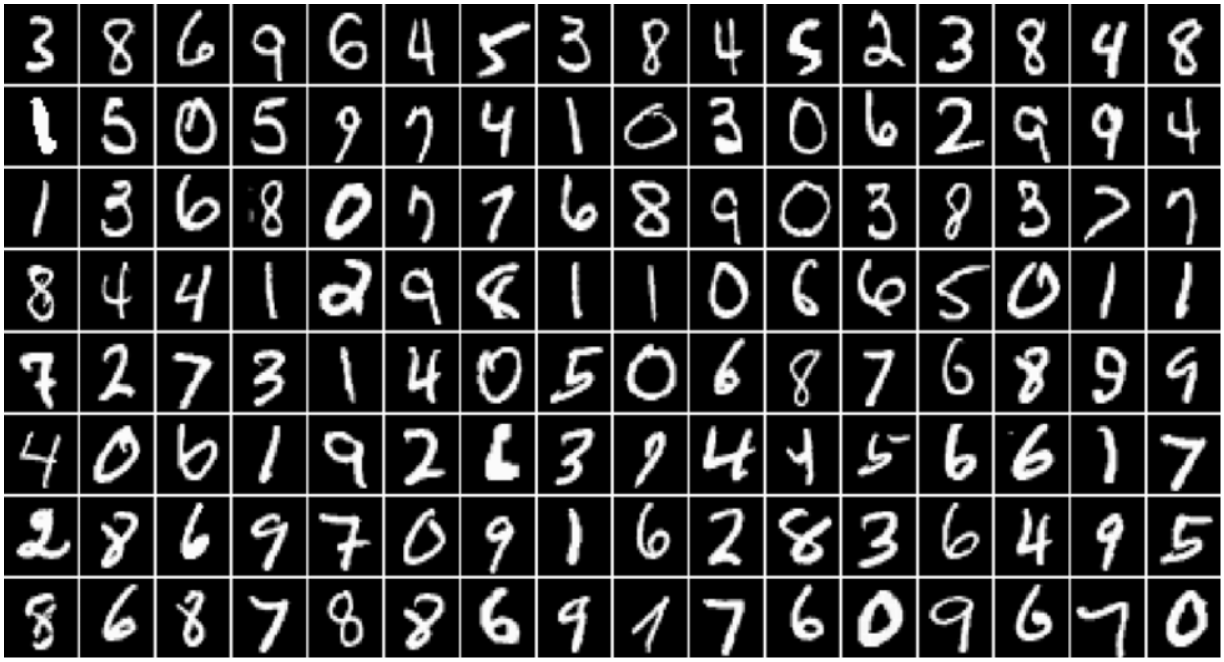
However, if we go deeper, the term "Minimum padding" would make more sense.

In the end, we should remember that using the minimum value from the activation maps is the best choice, and because most of the time it is relu, people prefer "0-padding".

Do we need to add padding in every layer?

Let us understand this with the help of an example. Consider the MNIST digit recognizer dataset

Module 4



Here are some sample images from the MNIST dataset. In the blog earlier, we discussed the two main reasons to use padding:

1. Shrinking output after convolution, so we can add more layers to extract more features.
2. To save the loss of information around the edges.

You can see that the numbers in the dataset are not around the edges, they are mainly at the center. This means there may be very little information around the edges. So, the main reason to add padding in this case is to add more layers. However, MNIST does not require that many layers. You may add padding in initial layers but it won't be of much use to add it in all the layers. It would only result in the model becoming heavier.