Convolutional Neural Network

# Introduction

If you had to pick one deep learning technique for computer vision from the plethora of options out there, which one would you go for? For a lot of folks, including myself, convolutional neural network is the default answer.

But what is convolutional neural network and why has it suddenly become so popular? Well, that’s what we’ll find out in this article! CNNs have become the go-to method for solving any image data challenge. Their use is being extended to video analytics as well, but we’ll keep the scope to image processing for now. Any data that has spatial relationships is ripe for applying CNN – let’s just keep that in mind for now.

# Computer Vision

Some of the computer vision problems which we will be solving in this article are:

1. Image Classification
2. Object detection
3. Neural Style Transfer

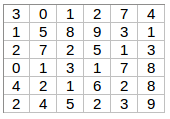
One major problem with computer vison problems is that the input data can get really big. Suppose an image is of the size 68 \* 68 \* 3. The input feature dimension then becomes 12288. This will be even bigger if we have larger images (say, of size 720 \* 720 \* 3). Now if we pass such a big input to a neural network, the number of parameters will swell up to a HUGE number (depending on the number of hidden layers and hidden units). This will result in more computational and memory requirements --- not something most of us can deal with.

# Edge Detection Example

Several people standing on a bridge

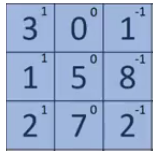
Description automatically generatedWe saw that the early layers of a neural network detect edges from an image. Deep layers might be able to detect the cause of the objects and even more deeper layers might detect the cause of complete objects (like a person’s face).

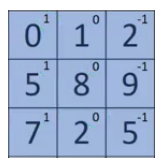
As you can see, there are many vertical and horizontal edges in the image. The first thing to do is to detect these edges:

But how do we detect these edges? To illustrate this, let’s take a 6\*6 grayscale image (i.e., only one channel):

A grid with numbers and a pixelated image

Description automatically generatedNext, we convolve this 6\*6 matrix with a 3\*3 filter:

After the convolution, we will get a 4\*4 image. The first element of the 4\*4 matrix will be calculated as:

So, we take the first 3\*3 matrix from the 6\*6 image and multiply it with the filter. Now, the first element of the 4\*4 output will be the sum of the element-wise product of these values, i.e., 3\*1 + 0\*1 +1\*-1 + 1\*1 + 5\*0 + 8\*-1 + 2\*1 + 7\*0 + 2\*-1 = -5. To calculate the second element of the 4\*4 output, we will shift our filter one step towards the right and again get the sum of the element-wise product:

A grid of numbers with black text

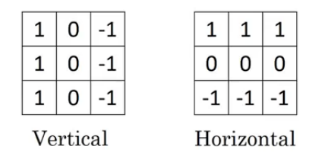
Description automatically generatedSimilarly, we will convolve over the entire image and get a 4\*4 output:

A number grid with numbers and symbols

Description automatically generated with medium confidenceSo, convolving a 6\*6 input with a 3\*3 filter gave us an output of 4\*4. Consider one more example:

Note: Higher pixel values represent the brighter portion of the image, and the lower pixel values represent the darker portions. This is how we can detect a vertical edge in an image.

# More Edge Detection

The type of filter that we choose helps to detect the vertical or horizontal edges. We can use the following filters to detect different edges:

A screenshot of a grid

Description automatically generatedSome of the commonly used filters are:

The Sober filter puts a little bit more weight on the central pixels. Instead of using these filters, we can create our own as well and treat them as a parameter which the model will learn using backpropagation.

# Padding

We have seen that convolving an input 6\*6 dimension with a 3\*3 filter results in 4\*4 output. We can generalize it and say that if the input is n X n and the filter size is f X f, then the output size will be (n-f+1) \* (n-f+1)

There are primarily two disadvantages here:

1. Every time we apply convolutional operation, the size of the image shrinks.
2. Pixels present in the corner of the image are used only a few numbers of times during convolution as compared to the central pixels. Hence, we do not focus too much on the corners since that can lead to information loss.

To overcome these issues, we can pad the image with an additional border, i.e., we add one pixel all around the edges. This means that the input will be an 8\*8 matrix (instead of a 6\*6 matrix). Applying convolution of 3\*3 on it will result in a 6\*6 matrix which is the original shape of the image. This is where padding comes to the force:

* Input: n \* n
* Padding: p
* Filter size: f \* f
* Output: (n+2p-f+1) \* (n+2p-f+1)

There are 2 common choices for padding:

1. **Valid**: It means no padding. If we are using valid padding, the output will be (n-f+1) \* (n-f+1)
2. **Same**: Here, we apply padding so that the output size is the same as the input size, i.e.,

n+2p-f+1 = n

So, p = (f-1)/2

# Strided Convolutions

Suppose we choose a stride of 2. So, while convoluting through the image, we will take 2 steps --- both in the horizontal and vertical directions separately. The dimensions for stride s will be:

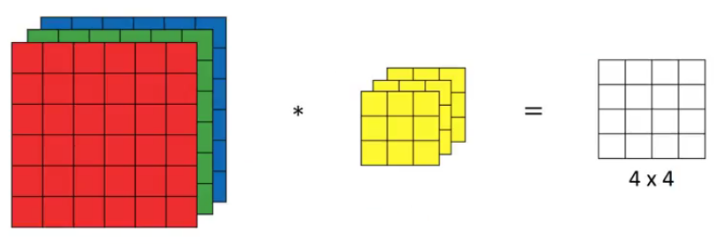
* **Input** --- n \* n
* **Padding** --- p
* **Stride** --- s
* **Filter** **size** --- f \* f
* **Output** --- [(n+2p-f)/s+1] \* [(n+2p-f)/s+1]

Stride helps to reduce the size of the image, a particularly useful feature.

# Convolutions Over Volume

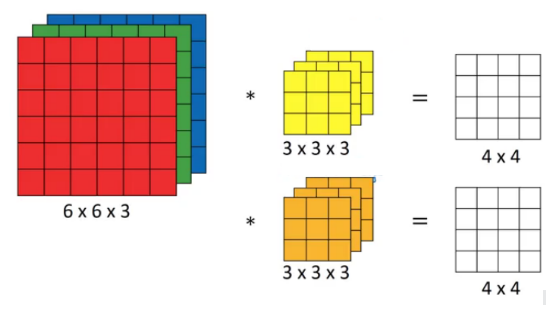
Suppose instead of a 2-D image, we have a 3-D input image of shape 6 \* 6 \* 3. How will we apply convolution to this image? We will use a 3 \* 3 \* 3 filter instead of a 3 \* 3 filter. Let’s look at an example:

* Input: 6\*6\*3
* Filter: 3\*3\*3

The dimensions above represent the height, width and channels in the input and filter. Keep in mind that the number of channels in the input and filter should be the same. This will result in an output of 4 \* 4. Let’s understand it visually:

Since there are 3 channels in the input, the filter will consequently also have 3 channels. After convolution, the output shape is a 4\*4 matrix. So, the first element of the output is the sum of the element-wise production of the first 27 values from the input (9values from each channel) and the 27 values from the filter. After that we convolve over the entire image.

Instead of using just a single filter, we can multiple filters as well. How do we do that?

Let’s say that first filter will detect vertical edges and the second filter will detect horizontal edges from the image. If we use multiple filters, the output dimension will change. So, instead of having 4\*4 output, we would have a 4\*4\*2 output (if we have used 2 filters)

Generalized dimensions can be given as:

* Input --- n \* n \* nc
* Filter --- f \* f \* nc
* Padding --- p
* Stride --- s
* Output --- [(n+2p-f)/s+1]\*[(n+2p-f)/s+1]\*nc’

Here, nc is the number of channelsin the input and filter, while nc’ is the number of filters.

# One Layer of a Convolutional Network

Once we get an output after convolving over the entire image using a filter, we add a bias term to those outputs and finally apply an activation function to generate activations. This is one layer of a convolutional network. Recall that equation for one forward pass is given by:

In our case, input (6\*6\*3) is a[0] and filters (3\*3\*3) are the weights w[1]. These activations from layer 1 act as the input for layer 2, and so on. Clearly, the number of parameters in the case of convolutional neural networks is independent of the size of the image. It essentially depends on the filter size. Suppose we have 10 filters, each shape 3\*3\*3. What will be the number of parameters in that layer? Let’s try to solve this:

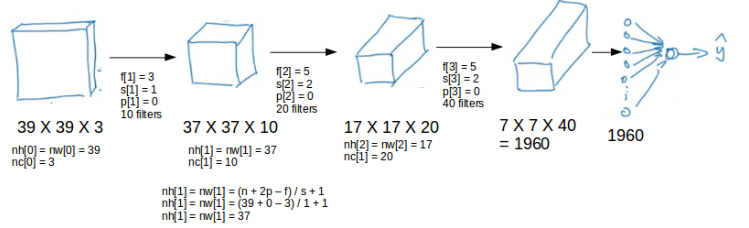
* Number of parameters for each filter = 3\*3\*3 = 27
* There will be a bias term for each filter, so total parameters per filter = 28
* As there are 10 filters, the total parameters for that layer = 28\*10 = 280

No matter how big the image is, the parameters only depend on the filter size. Awesome, isn’t it? Let’s have a look at the summary of notations for a convolution layer:

* F[l] = filter size
* P[1] = padding size
* S[l] = stride
* N[c][l] = number of fitlers

Let’s combine all the concepts we have learned so far and look at a convolutional network example.

# Simple Convolutional Network Example

This is how a typical convolutional network looks like:

We take an input image (size = 39\*39\*3 in our case), convolve it with 10 filters of size 3 \* 3, and take the stride as 1 and no padding. This will give us an output of 37 \* 37 \* 10. We convolve this output further and get and output of 7 \* 7 \* 40 as shown above. Finally, we take all these numbers (7 \* 7 \* 40 = 1960), unroll them into a larger vector, and pass them to a classifier that will make predictions. This is a microcosm of how a convolutional network works.

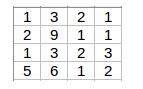
There are a number of hyperparameters that we can tweak while building a convolutional network. These include the number of filters, size of filters, stride to be used, padding, etc. We will look at each of these in detail later in this article. Just keep in mind that as we go deeper into the network, the size of the image shrinks whereas the number of channels usually increases.

In a convolutional network there are basically 3 types of layers:

1. Convolution layer
2. Pooling layer
3. Fully connected layer

Let’s understand the pooling layer in the next section.

# Pooling Layers

Pooling layers are generally used to reduce the size of the inputs and hence speed up the computation. Consider a 4 \* 4 as shown below:

A red and blue squares with black arrows

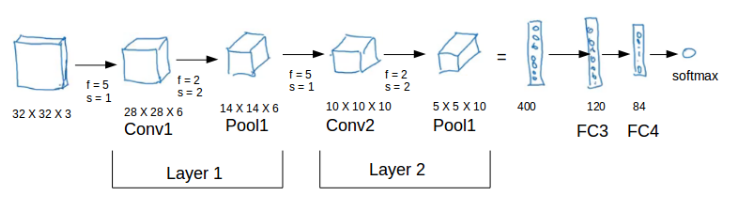
Description automatically generatedApplying max pooling on this matrix will result in a 2 \* 2 output:

For every consecutive 2\*2 block, we take the max number. Here, we have applied a filter of size 2 and a stride of 2. These are the hyperparameters for the pooling layer. Apart from max pooling, we can also apply average pooling where, instead of taking the max of the numbers, we take their average. In summary, the hyperparameters for a pooling layer are:

1. Filter Size
2. Stride
3. Max or Average Pooling

If the input of the pooling layer is nh \* nw \* nc , then the output will be

# CNN Example

We’ll take things up a notch now. Let’s look at how a convolution neural network with convolutional and pooling layer works. Suppose we have an input of shape 32 \* 32 \* 3:

There are a combination of convolutional and pooling layers at the beginning, a few fully connected layers at the end and finally a softmax classifier to classify the input into various categories. There are a lot of hyperparameters in this network which we have to specify as well.

Generally, we take the set of hyperparameters which have been used in proven research and they end up doing well. As seen in the above example, the height and width of the input shrinks as we go deeper into the network (from 32 \* 32 to 5 \* 5) and the number of channels increases (from 3 to 10).

All of these concepts and techniques bring up a very fundamental question --- why convolutions? Why not something else?

# Why Convolution?

There are primarily 2 major advantages of using convolutional layers over using just fully connect layers:

1. Parameter Sharing
2. Sparsity of connections

A drawing of a cube with numbers and a black arrow

Description automatically generatedConsider the below example:

If we would have used just the fully connected layer, the number of parameters would be 32 \* 32 \* 3 \* 28 \*28 \* 6, which is nearly equal to 14 million! Make no sense, right?

If we see the number of parameters in case of a convolutional layer, it will be = (5\*5+10\*6 (if there are 6 filters), which is equal to 156. Convolutional layers reduce the number of parameters and speed up the training of the model significantly.

In convolutions, we share the parameters while convolving through the input. The intuition behind that is that a feature detector, which is helpful in one part of the image, is probably also useful in another part of the image. So, a single filter is convolved over the entire input and hence the parameters are shared.