Introduction to Convolutional Neural Network

Zhihan Hu

March 2025

1 Introduction

In this document, I will introduce a widely used deep learning architect named Convolutional Neural Network(CNN). In the current ML industry, CNN presents to be the most powerful tool to handle computer vision tasks.

2 Convolution

Before talking about CNN, it is important to have a basic background of Convolution operation.

Convolution is an operation where we apply a filter (or kernel) to a matrix A (an image or data grid) by sliding the filter over A and computing weighted sums. Convolution works like this shown below:
Positioning the Filter:

- Place the filter on top of A such that it covers a small region of A.

Element-wise Multiplication and Sum up:

- Multiply each element of the filter with the corresponding element in the covered region of A.
- Sum all the multiplied values to get a single number. This result is stored in the output matrix at the corresponding position.

Sliding the Filter:

- Move the filter one step to the right (one column over).
- Repeat steps 2

Handling the Edge:

- If the filter reaches the right edge of A, move it one row down and reset to the first column.

Repeat the process until the filter has scanned the entire matrix.

Now let's see an numerical example to further understand Convolution

$$A = \begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{bmatrix} \text{ and filter} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

We use * to denote the convolution operation (Note that * may also mean mat mul, so I will try my best to make de notation clear).
A * filter will be computed as follows:

- Filter is put onto the upper left 3 x 3 region of A. Doing the element-wise product and summation, we have -5
- Filter then move one step to the right, the new region will be $\begin{bmatrix} 0 & 1 & 2 \\ 5 & 8 & 9 \\ 7 & 2 & 5 \end{bmatrix}$, and the result is -4
- ...
- Filters are at the upper right of A, we then move filter one step down, and reset to the left, the region on A will be $\begin{bmatrix} 1 & 5 & 8 \\ 2 & 7 & 2 \\ 0 & 1 & 3 \end{bmatrix}$, and the result is -10

The final result is given here for you to check your math: $\begin{bmatrix} -5 & -4 & 0 & 8 \\ -10 & -2 & 2 & 3 \\ 0 & -2 & -4 & -7 \\ -3 & -2 & -3 & -16 \end{bmatrix}$

3 Padding and Stride

One of the issues convolution has is that elements of A at the corner will be applied much less compared to elements in the middle area of A. In order to fix this issue, a common method is to pad the matrix. Suppose A is in dimension m by n, and we apply a padding of zeros p on A, the padded A will have dimension m + 2p by n + 2p.

We can also arbitrarily define the movement of the filter, i.e, how much steps the filter move. In the last example above, we move filter by 1 step each time, but we can also move filter 2 steps each time. We call this stride and denoted as s.

Suppose A has a dimension of n by n, and filter has f by f, we have padding p and stride s, we can then get the dimension of the output

$$\lfloor(\frac{n+2p-f}{s}+1)\rfloor \times \lfloor(\frac{n+2p-f}{s}+1) \tag{1}$$

4 Convolution on a tensor

For most of the image data currently, we use tensor instead of matrix to represent, since for each pixels, we have RGB channels. And thus an image is often represented as (height, width, 3). The following graph shows how convolution on tensor works We need a stack of filters, the size of the stack is the same as

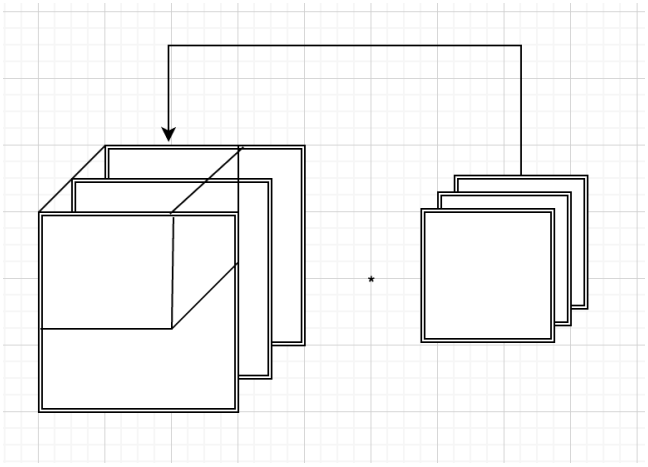

Figure 1: convolution on tensors

the input tensor, which is 3 in this case. We then apply each sub-filter in the stack correspondingly to the matrix of the tensor. Suppose the filter is (f, f, 3), then we will do element-wise product of the **volume** on A with filter, and then sum them together. **Note that, the result of convolution on tensor is a matrix**.

We simply stack the output matrix of applying each filter if we want to multiple numerous filters on the input tensor, the workflow is shown in Figure 2. As you can see, the output becomes a tensor whose last dimension equals to the number of filters.

5 One layer of Conv Layer

We now have known what will happen when convoluting filters on a tensor, now we are ready to get started with what the layer will be for CNN.

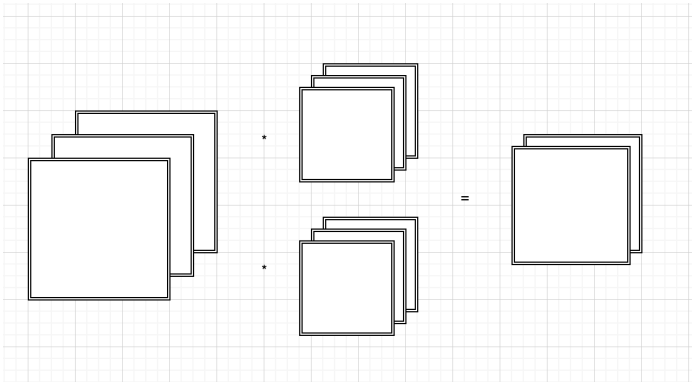

Figure 2: multi-filters convolution

For each conv layer, we have several filters, and will be applied independently and in parallel to the input tensor to generate same number of output matrices as the number of filters. Then we apply non-linearity for each output matrix, and add a bias to each element. **Note that, one output matrix has one bias**. The following graph shows the architect of a CNN.

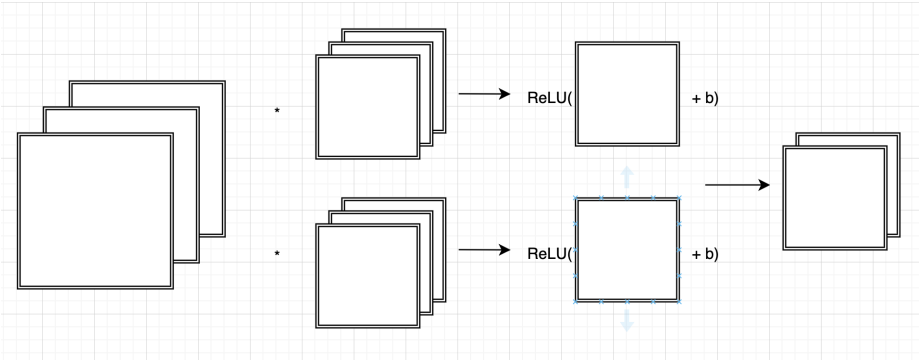

Figure 3: One conv layer

Here comes the complicated part, the notation of a CNN layer.

- $f^{[l]}$ = filter size on layer l
- $p^{[l]}$ = padding size on layer l
- $s^{[l]}$ = stride on layer l
- $n_c^{[l]}$: number of filters on layer l

- Input $A^{[l-1]} \in R^{m \times n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}}$
- Output $Z^{[l]} \in R^{m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}}$, where

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

$$n_W^{[l]} = \lfloor \frac{n_W^{[l-1]} + 2p - f}{s} + 1 \rfloor \tag{3}$$

- filters $\in R^{f^{[l]} \times f^{[l]} \times n_c^{[l-1]}}$, note that, in each stack of the filters, the size will be the number of previous filter number. This is the part easy to make mistakes.
- Activations $A^{[l]} \in R^{m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}}$
- Total number of weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$
- Total bias: $n_c^{[l]}$

Admittedly, CNN has a much more complex math setup compared to DNN. My suggestion is that if you need to play with these notation, please refer to these formulas and explicitly write out each variables of your model, e.g, $n_H^{[l]} = 3$, etc.

6 Pooling

Usually, a convolution layer will also contain a Pooling operation after the convolution. The pooling operation is much simpler than convolution, however, its behavior is a little bit different from convolution and may make you confused. So, please be sure to clearly understand how convolution and pooling work.

We commonly have 2 pooling filter, max and average pooling filter.

- Max pooling filter will select the max value within the region of the input matrix covered by the filter.
- Average pooling filter will compute the average value within the region of the input matrix covered by the filter.

The difference is that, pooling filter applies to the tensor matrix-wisely. i.e, there is no volume, instead, the filter will apply on the 2D matrices of the tensor one by one, and generate the result that has same $n_c^{[l]}$ as the third dimension.

For the pooling operation, we no longer have learnable parameters, instead, we need to tune some hyperparameters listed below:

- f: pooling filter size
- s: stride

- Max or average

One more thing to note is that input tensor does not need to be padded when it is applied a pooling filter, therefore, the output dimension will be

$$n_H^{[l]} = \lfloor \frac{n_H^{[l]} - f}{s} + 1 \rfloor \tag{4}$$

$$n_W^{[l]} = \lfloor \frac{n_W^{[l]} - f}{s} + 1 \rfloor \tag{5}$$

7 A complete example of CNN model

The following CNN architect is a classic architect that was designed by Yann LeCun. We will go through the dimension changes to make the above notations clear. This graph shows a 2 conv layer fully-connected (FC) CNN model, where

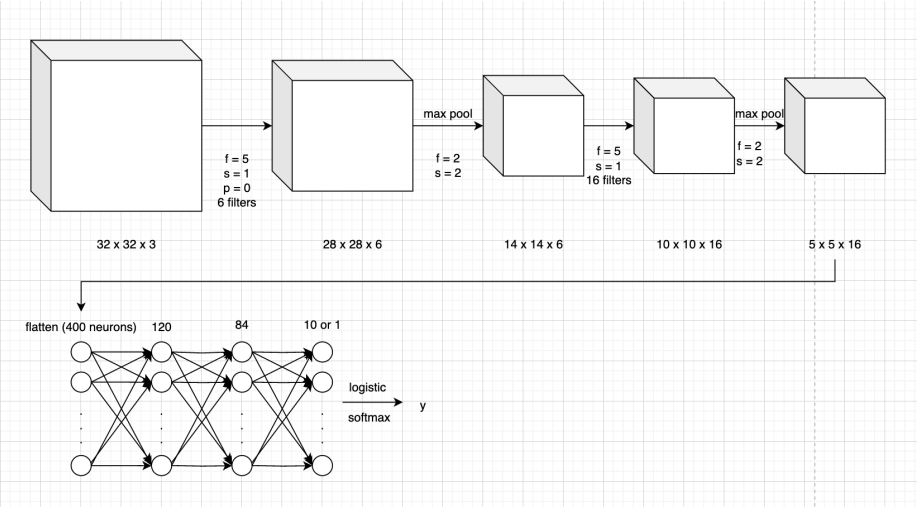

Figure 4: Yann LeCun CNN model

in each conv layer, one convolution and one pooling operation are presented.

Layer 1:

- Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]} = 32$ by 32 by 3
- $f^{[1]} = 5$, $s^{[1]} = 1$, $p^{[1]} = 0$, $n_c^{[1]} = 6$
- Thus the output is $\frac{32-5}{1} + 1$ by $\frac{32-5}{1} + 1$ by $6 = 28$ by 28 by 6
- We have 6 filters, and each filter contains 3 stacks as $n_c^{[0]} = 3$, thus the total number of weights is 5 * 5 * 3 * 6 = 450

- We also have 6 biases
- Thus the entire parameters for layer 1 is 456
- The pooling operation will not change $n_c^{[1]}$
- The output matrix for each 2D matrix of the output from conv is $\frac{28-2}{2}+1$, $\frac{28-2}{2}+1$, $6 = 14$ by 14 by 6

You can do similar things for layer 2.

After 2 conv layers, we flatten the tensor into a 1d vector, and feed them into a fully connected DNN. This is the same as what we have already learned, and we can do all the optimization algorithm to fasten the training process.

8 Why Convolution

The reason to use CNN instead of DNN in computer vision is intuitively due to that CNN use much less learnable parameters, and filters can be applied in parallel which maximizes the parallelism.

For example, if we are to use DNN, if the input is 32 by 32 by 3 and the output for layer 1 is 28 by 28 by 6, we then need around 14 million parameters for this layer, which is impractical.

Theoretically, CNN fits for the properties of image data.

- Parameter sharing: In an image, a lot of features can be detected in the same way from this subparts to another subparts. So we can create detectors(filters) for different features and apply them on the entire graph
- Sparse connection: In most of the cases, one part of the image has really weak or simply no connection to another part. So we don't need to use a fully connected DNN to connect each pixels to the output.

9 Aside: How CNN is developed

CNN is admittedly unintuitive, i.e, you can understand how it works, but you just don't know why it is designed like this. In this section, I will talk about the inspiration of CNN, how such unintuitive model was designed from 0 to 1.

The earliest inspiration can be traced back to 1960s, when Hubel & Wiesel published their astonishing results on human visions. In their publications, they found that the vision cortex on human brain detects features like edges and textures first, and then aggregate the simple features to more complex

and structured features. This led to an idea of processing images layer by layer, where preceding layers detects simple features(conv layers), and proceeding layers aggregate the detected features into more complex and structured features(FC DNN).

How to capture simple features? This problem is inspired by signal processing, where signal is processed using a filter. In fact, what conv layers do is the same as what signal processing does.

And based on these two inspiration(there are other inspiration I didn't mention), CNN is developed.