Erdi KARA

1 Convolution Neural Networks(CNN)

In this section, we will introduce the basic principles of another neural network model, convolutional neural networks(CNN). CNN model are commonly applied for image analysis problems. In the previous section, we discussed DNN models where each neuron are essentially connected to each other in the architecture. This actually means we dont assume any meaningful connection between the neurons. A CNN models aims to explore possible hidden connections in the input data and by this way enables the architecture to learn with a more *regularized* way.

Introduction

Let's start with the formal definition of the convolution operation. Convolution is a special integral transform. Let f and b be two functions. " f convolution g " is a function defined as

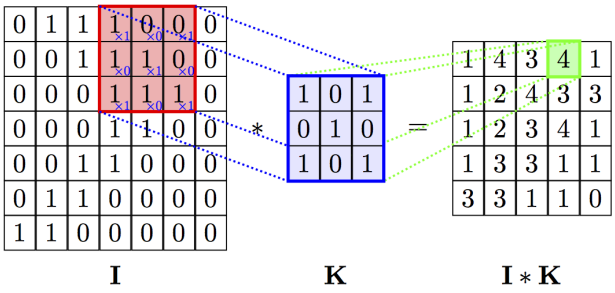
$$(f * g)(x) = \int_{-\infty}^{\infty} f(\tau)g(x-\tau)d\tau \tag{1}$$

Geometrically, we first reflect the function $g(\tau)$ to $g(-\tau)$ and add a time offset so that we obtain $g(t-\tau)$ .We then multiply $g(t-\tau)$ with $f(\tau)$, vary t from $-\infty$ to ∞ and compute the integral over all t's where both function are non-zero. In practical application, we can use define a discrete a convolution in 2D as follows:

$$(f * g)(x, y) = \sum_{i,j} w_{ij} f(\tau_i, \tau_j) g(x - \tau_i, y - \tau_j) \tag{2}$$

where w_{ij} are the integral weights.

Let's consider a concrete example to illustrate how (2) can be used in image processing. Let the function f represents an mxn pixel image. So $f(x_i, y_j)$ is the pixel value at (i, j) location. We will $g(x, y)$ as a *kernal* function. Assume that as a discrete function $g(x, y)$ has $3x3$ pixel support size. In this case, for a specific pixel location, the sum in (2) is performed over this $3x3$ region. Similar to the continuous case, we start with the upper left most pixel, apply 3x3 discrete convolution and slide g to the next pixel and perform the same operation. When we scan the entire pixels, the result is essentially another image. Here, the sliding amount is called the *stride* size while the $3x3$ is called the *filter size*. An illustrative figure can be seen below.



The convolution or filtering operation plays a central role in image processing. Discrete kernels are used for noise removal, edge detection, image enhancement and so many other image processing operations [1]. For example, following kernels can be used as edge detectors.

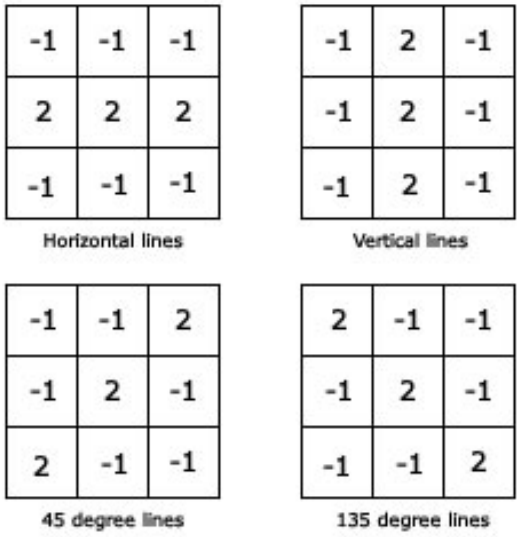


Figure 1: Typical image filters

There are some details in the implementation of image filters such as zero padding, wrapping, reflection etc but we will not go through the details. We should however mention the following formula concerning the output dimension after the convolution. For a square kernel of size kxk, and image of size WxW with padding p, stride s, the output image is of the size

$$W_{out} = \left\lceil \frac{W - k + 2p}{S} \right\rceil + 1$$

The kernels in (1) are specifically designed for edge detection. However we can claim that any type of filter applied to an image reveals *some pattern* even if it is qualified as a meaningful pattern.

CNN models use the kernels to explore the features in a picture in a more systematic way but the major difference is that the kernels are determined by the learnable parameters

what we call *shared weights* which don't have prescribed values but evolve over the course the training. Let's go through the details.

CNNs have three major components; local receptive fields, shared weights and a pooling layer. Shared weights are different type of kernels and the local receptive field is the portion of the image where we apply the kernel. Carefully note that even though we tend to think the image as mxn pixel size, we in fact have $m \cdot n$ input neurons to be passed to the architecture. In CNN models, convolution operation is usually followed by an *pooling* operation where we further condense the convoluted information. A visial structure of CNN can be seen in figure (2). (Source https://vinodsblog.com).

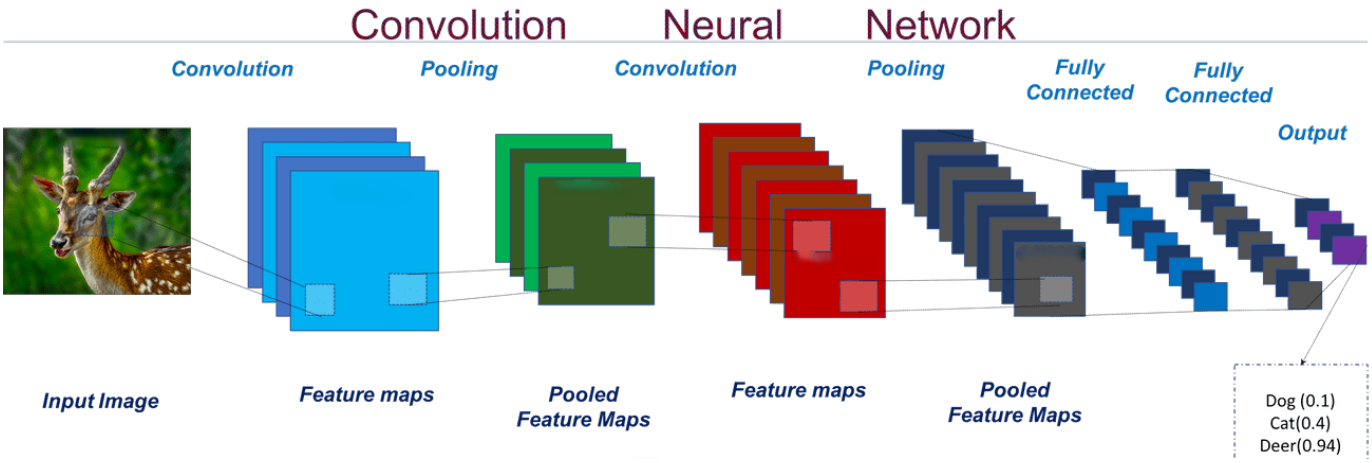


Figure 2: A generic CNN architecture

Let's formally define how to apply convolution operation in a CNN model. In convolution layers, we have convolution operation instead of matrix multiplication, which we can define as follows

$$z_{x,y}^{l+1} = w^{l+1} * \psi(z_{x,y}^{l}) + b_{x,y}^{l+1} = \sum_{a=1}^{k}\sum_{b=1}^{k} w_{a,b}^{l+1}\psi(z_{x-a,y-b}^{l}) + b_{x,y}^{l+1} \tag{3}$$

where w is the kxk kernel at the l^{th} layer, i.e, kxk matrix with entries $w_{a,b}^{l}$. The output image of the convolution operation is called a *feature map*. Note that the kernel w is a parameter which is same for the corresponding feature map but different for the others. As for the bias, we can use one bias per convolution filter (*tied bias*) or one bias for eaach kernel location(*untied bias*). We used untied bias here. Now assume that we have a gray scale image of size of size $50x50$. If we prescribe 7 feature maps with $3x3$ kernels then we have $7x(3x3+1) = 70$ parameters with tied bias. Note that for a color image with 3 color channels, we have $7(3x3x3+1) = 196$ parameters. The output size of each feature map with zero padding stride 1 becomes $\dfrac{50-3+0}{1} + 1 = 48$. Convolution is followed by a pooling operation which reduce the size of convolution output with an operation called pooling. The most common ones are max pooling and L2 pooling which we define as

$$z_{i,j} = max\{z_{i+n-1,j+n-1} : 1 \leq n \leq p\} \qquad \text{max pooling} \tag{4a}$$

$$z_{i,j} = \sqrt{\sum_{n=1}^{p}\sum_{n=1}^{p} z^2_{i+n-1,j+n-1}} \qquad \text{L2 pooling} \tag{4b}$$

where p is the pooling size. Max pooling outputs the maximum element in pxp pooling region while L2 pooling computes the L2 average. Output dimension of a pooling operation can be found as follows:

$$W_{out} = \frac{W - p}{S} + 1$$

where W is the input size, p is the kernel size and S is the stride. Complete picture can be seen in figure (3).

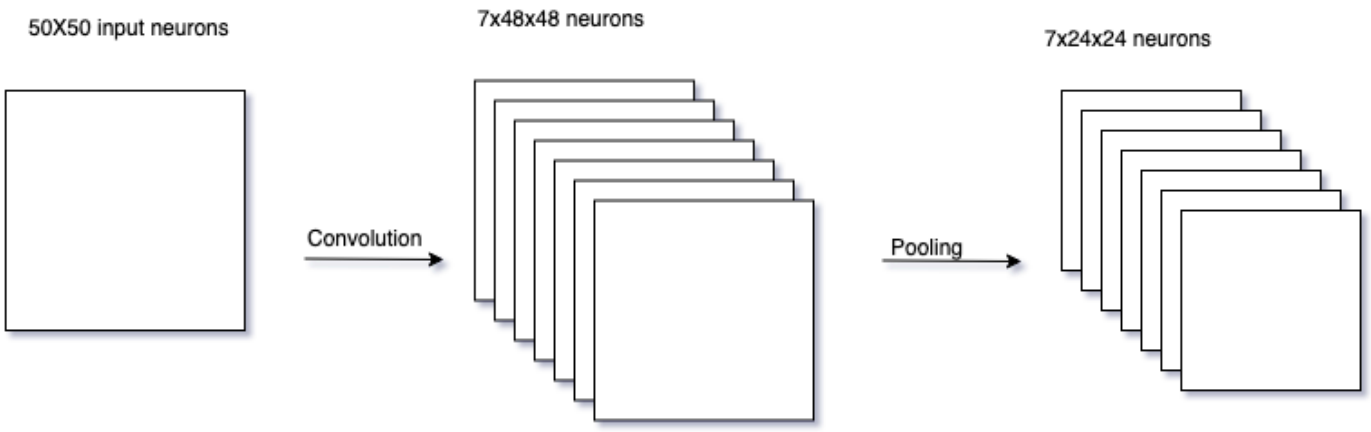


Figure 3: 7 3×3 filters applied to 50×50 image, followed by Max pooling 2×2 with with kernel size 2.

1.1 Back Propagation in CNN Models

In this section, we will derive the fundamental equations of backpropagation in CNN models. Let's define

$$\delta^l_{x,y} = \frac{\partial C}{\partial z^l_{x,y}} \tag{5}$$

for a generic cost function. Using the chain rule and (3), we can write

$$\delta^l_{x,y} = \frac{\partial C}{\partial z^{l+1}_{x',y'}} \frac{\partial z^{l+1}_{x',y'}}{\partial z^l_{x,y}} = \sum_{x'}\sum_{y'} \delta^{l+1}_{x',y'} \frac{\partial}{\partial z^l_{x,y}} \Big(\sum_a \sum_b w^{l+1}_{a,b} \psi(z^l_{x'-a,y'-b}) + b^{l+1}_{x',y'} \Big) \tag{6}$$

Note that the derivative term above is 0 expect the indexes satisfying $x' - a = x$ and $y' - b = y$, i.e., $a = x' - x, b = y' - y$. Using the definition of the convolution (3), we have

$$\delta^l_{x,y} = \sum_{x'}\sum_{y'} \delta^{l+1}_{x',y'} w^{l+1}_{x'-x,y'-y} \psi'(z^l_{x,y}) = \delta^{l+1} * w^{l+1}_{-x,-y} \psi'(z^l_{x,y}) \tag{7}$$

Sometimes, it is written $w_{-x,-y} = ROT180(w_{x,y})$ Here ROT180 represent the 180 degree rotation of the kernel. If start indexing w from the middle element entry, $w_{-x,-y}$ can be found by rotating the same kernel 180 degree about $(0,0)$ index. We then compute the partial derivatives using (7).

$$\frac{\partial C}{\partial w^l_{a,b}} = \sum_x \sum_y \frac{\partial C}{\partial z^l_{x,y}} \frac{\partial z^l_{x,y}}{\partial w^l_{a,b}} = \sum_x \sum_y \delta^l_{x,y} \frac{\partial}{\partial w^l_{a,b}} \Big(\sum_{a'} \sum_{b'} w^l_{a',b'} \psi(z^{l-1}_{x-a',y-b'}) + b^l_{x,y} \Big) =$$

$$\sum_x \sum_y \delta^l_{x,y} \psi(z^{l-1}_{x-a,y-b}) = \delta^l_{a,b} * \psi(z^{l-1}_{-a,-b})$$

So we have

$$\frac{\partial C}{\partial w^l_{a,b}} = \delta^l_{a,b} * \psi(z^{l-1}_{-a,-b}) \tag{8}$$

and similarly

$$\frac{\partial C}{\partial b^l_{a,b}} = \delta^l_{a,b} \tag{9}$$

We can close this section with an interesting note. As we see from the construction, we do not pre-assign any specific kernel type (Prewitt, Sobel, Gabor filters etc) to reveal some patters in the input images. However it is observed that as we train the architecture, shared weights are learning to reveal meaningful *patterns* . In particular early layers learns basic shapes such as vertical, horizontal lines while the next layers learn more compact patters such as cars, ships, mountains etc.

References

[1] Chris Solomon and Toby Breckon. *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab.* Wiley Publishing, 1st edition, 2011.