Using Machine Learning to Diagnose Pneumonia from Covid-19 in Patients from Chest X-Rays.

James Lamb

November 2020

1 Overview of Convolutional Neural Network

1.1 Elements of a CNN

Convolutions The prime advantage of a CNN over other neural networks is the use of convolution, a linear operation used for feature extraction. Formally, the convolutional formula can be given as,

$$G[i,j] = f * k[i,j] = \sum_m \sum_n k[m,n] \times f[i-m, j-n] \ .$$

$$f = Image$$
$$k = Kernal$$

Consider an input image of 16×16 pixels with RGB channels. In a typical neural network, to connect this input to a single neuron would require $16 \times 16 \times 3 = 768$ weighted connections. To reduce this number, we define a "kernal" (perhaps several) as an $n \times m$ matrix [1] to isolate local regions of the input image.

The kernal can be visualised as a kind of view-port sliding over the image. This means an assumption can be made in that for each local region we use a duplicate neuron, the weights applied are remain fixed and identical across each neighbouring neuron, thus reducing further the number of parameters. Further, by considering that the filters which are intended to be applied to the input image can be represented in matrix form then the kernal itself can be used as a filter. To highlight this, consider a 1-dimensional convolutional layer wherein each neuron can be described by the function,

$$\Omega(\sum_{i=0}^{n}(w_i x_i) + b) \ . \qquad i = 0, 1, 2, ...$$

$$b = bias$$
$$w = weight$$
$$\Omega = activation\ function$$

Now consider the weight matrix, W, which consists of each weight element, w_i. In the case of a regular ANN, shown in (1), every input pixel is connected to each neuron with a different weight. However within a convolution layer (2), many of the neurons are disconnected and therefore have zero value, and due to the multiple copies of neurons we have the same weight functions applied in

[1] typically $n = m$, unless specific features of the input image are known prior to processing for which $n \neq m$ would be computationally optimal.

differing positions.

$$\begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & & \\ \vdots & & \ddots & \\ w_{n,0} & & & w_{n,n} \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} w_0 & w_1 & 0 & 0 & \dots \\ 0 & w_0 & w_1 & 0 & \dots \\ 0 & 0 & w_0 & w_1 & \dots \\ 0 & 0 & 0 & w_0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{2}$$

Hence each output neuron only has an interaction with the input pixel within the convolutional layer's kernal, and those weight parameters are shared across each neighbouring layer vastly reducing the number of parameters requiring optimisation.

That the weights are shared means that the features of an image are then considered to be spatially invariant. This implies that convolution may not be appropriate for applications where feature positions in the input image are important.

Stride Simply put, the stride defines the pixel distance the kernal must translate to process each local region of the input image. By setting this value, and considering the chosen kernal size, the overlap of each region is also defined along with the size of the output.

Consider a $N \times N$ image with a $K \times K$ kernal. Choosing the stride to be S, then the overlap is given by $K - S$ and the size of the output matrix is given by,

$$O = \frac{N-K}{S} + 1 \ .$$

Note that if stride and kernal are chosen such that $\frac{N-K}{S} \notin \mathbb{Z}$, then the remainder of the image pixels are not parsed by the kernal and do not factor into the output, hence the output size is more accurately given by,

$$O = \lfloor \frac{N-K}{S} \rfloor + 1 \ .$$

This can be further extended to three dimensions such that the process is repeated for each channel separately and the resulting 2-dimensional outputs are collated into a single 3-dimensional tensor, ie.

$$O = [N, N, N_c] * [K, K, N_c] = [\lfloor \frac{N-K}{S} \rfloor + 1, \lfloor \frac{N-K}{S} \rfloor + 1, N_f] \ .$$

$N_c =$ Number of image channels

$N_f =$ Number of filters

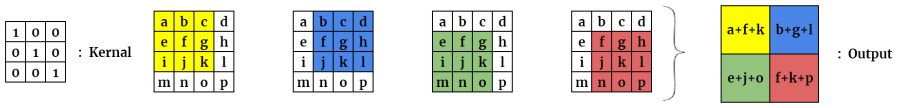

Figure 1: Kernal of order 3 acting on 4×4 image

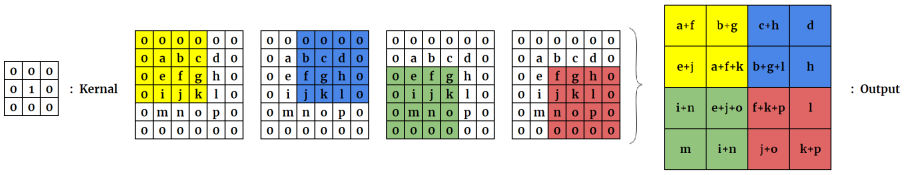

Figure 2: Kernal of order 3 acting on 4×4 image with zero-padding

A brief consideration of higher dimensions can be made. The fundamental amendment to the procedure here is in using a 3-dimensional tensor as a kernal. This acts on the input in a similar manner as before only it now slides in three dimensions, hence the output is also reduced in three dimensions.

Padding When a convolution layer is formed as discussed, the kernal captures information and collates it into a tensor, as in figure 1. This typically results in a loss of information at the border of the local region.

In order to retain this information, a method called zero-padding[2] is utilised wherein an artificial border of zero value is added to the input matrix. Using this method prevents the output size reducing with the depth of the convolution layers and inherently then allows the network to have any number of convolutional layers.

It also permits further management of the size of the output, with the modification to the formula for output size given by,

$$O = \lfloor \frac{N - K + 2P}{S} \rfloor + 1 \ .$$

Where P is the number of padding appended. Or the 3-D variant,

$$O = [N, N, N_c] * [K, K, N_c] = [\lfloor \frac{N - K + 2P}{S} \rfloor + 1, \lfloor \frac{N - K + 2P}{S} \rfloor + 1, N_f]$$

Pooling Pooling, much like valid padded convolution, is a process utilised to reduce the sampling rate of it's proceeding layers. Depending on whether max

[2]Commonly referred to as "same" padding. Conversely, "valid" padding refers to convolution without any padding appended to the input image.

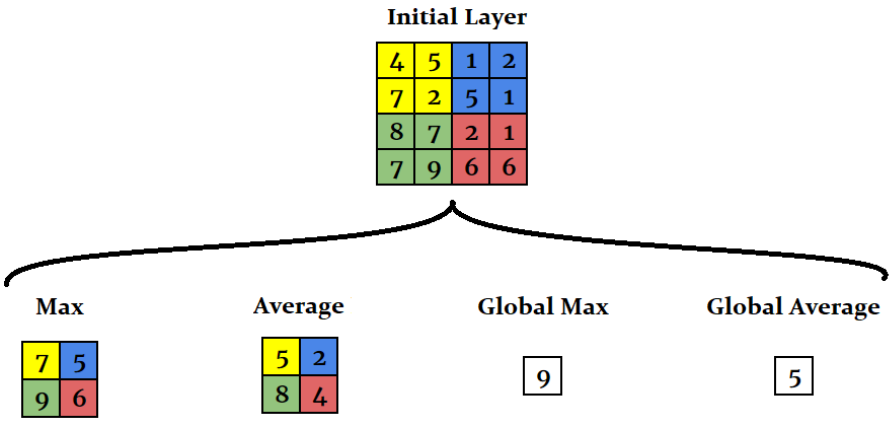

Figure 3: Depiction of different pooling outputs in regards to a sample initial layer

or average pooling is used this reduction is equivalent to either down-sampling or decimation, respectively. This acts to reduce noise in the input layers and secondly to speed up computation.

Pooling also has it's own stride parameter which, much like the stride parameter for the kernal, defines how many pixels across the tensor moves each time an element of the output tensor is evaluated and again inherently defines the output size and overlap. In contrast however, it is not typical to have an overlap in the pooling layer although it has been noted to provide more accurate results in the model.

In either case, much like the kernal, a tensor evaluates a set of sub-regions of a layer and returns a new tensor consisting of values representing each of those sub-regions.

The most commonly used pooling technique is max pooling. The sub-region is evaluated such that the maximum value within the region is returned and the remaining values are discarded.

ie. Consider the set of sub-regions of the layer, $\mathfrak{s}$, then

$$p_{i,j} = \max x_{p,q} \ . \qquad\qquad x_{p,q} \in \mathfrak{s}_{\mathrm{i,j}}$$

The second pooling technique to consider is average pooling. In this case the average of all of the values within each sub-region becomes the relative element of the pooled output layer.

Both of these pooling techniques can also be extended to what is called global pooling. Instead of returning a tensor, a single number is returned, taken from either the maximum or average of all the elements in the layer. This technique is more appropriate in applications such as natural language processing rather than purposes such as machine vision, however.

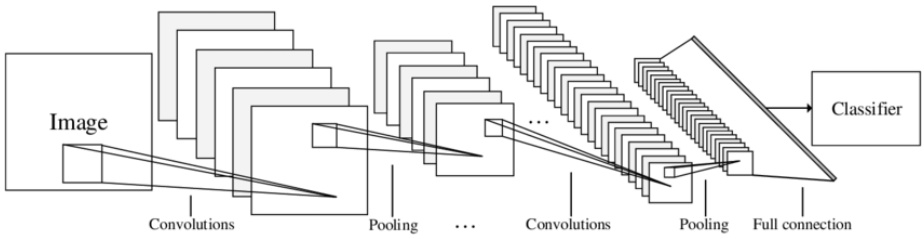

Figure 4: A basic architecture of a CNN

2 Results