Neural Net and Deep Learning

© by Xia, Tingfeng

Tuesday 11th February, 2020

Preface

Contents

1 Convolutional Neural Net and Image Classification

1.1 Motivations for Convolution Layer

- Images are, usually, large and hence using a lot of fully connected layers would result in an insane amount of parameter to learn and makes calculation intractable.
- In images, there are usually *local* patterns or relationships. For example in the task of semantic segmentation, usually the pixel at bottom left has very little to do with the pixel at top right and using a fully connected layer where every pixel contributes to every output would be wasteful. Thus, we want to have an operation that focus on ***local*** patterns of the input image.
- Also, the same sorts of features that are useful in analyzing one part of the image will probably be useful for analyzing other parts of the image as well which motivates us in using a "filter" to "slide" across the input.

1.2 The Convolution Operator

1.2.1 1-D Signal Processing

Consider two arrays, a and b. The result of the convolution will be a new array, where

$$(a * b)_t = \sum_{\tau} a_\tau b_{t-\tau} \tag{1}$$

In order to make this indexing notation to work, we have to have the indices start at zero rather at one.

where the summation over τ is a lazy notation for saying summing over all combinations that makes sense. [1]

1.2.2 2-D Convolution

Consider two two dimensional arrays, A and B. The result of the convolution will be such that the slot at $(A * B)_{ij}$ is calculated as

$$(A * B)_{ij} = \sum_{s} \sum_{t} A_{st} B_{i-s,j-t} \tag{2}$$

Usually, we call the matrix/tensor that we convolve the original input with a "kernel" or "filter".

[1] We assume infinite zero padding here, more on this later.

1.2.3 Properties of Convolution

- Convolution is ***Commutative***, i.e.,

$$a * b = b * a \tag{3}$$

- Convolution is ***Linear***, i.e.,

$$a * (\lambda_1 b + \lambda_2 c) = \lambda_1 a * b + \lambda_2 a * c \tag{4}$$

1.3 Canonical Kernels

Some of these canonical kernels are not working as expected...

1.3.1 Blurring Kernel

$$\begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & 4 & 1 \\ \hline 0 & 1 & 0 \\ \hline\end{array} \tag{5}$$

1.3.2 Sharpening Kernel

$$\begin{array}{|c|c|c|}\hline 0 & -1 & 0 \\ \hline -1 & 5 & -1 \\ \hline 0 & -1 & 0 \\ \hline\end{array} \tag{6}$$

1.3.3 Edge Detector Kernel

$$\begin{array}{|c|c|c|}\hline 0 & -1 & 0 \\ \hline -1 & 4 & -1 \\ \hline 0 & -1 & 0 \\ \hline\end{array} \tag{7}$$

1.3.4 Vertical Edge Detector Kernel

$$\begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline\end{array} \tag{8}$$

1.4 Convolutional Networks

In a Convolutional Neural Net, of course there will be convolution layers, however there is another sort of layers that are common, called the pooling layer. Intuitively, pooling layers shrink the dimensions by taking a "local pool".

1.4.1 Pooling Layer

Most commonly, we use the max-pooling operation in the pooling layer, which computes the maximum of the units in a pooling group

$$y_i = j\text{in local pooling group} \overset{\max}{z_j} \tag{9}$$

where z represents the input and y is the output. Typically, we use a 2×2 max pooling unit, which outputs

$$\begin{array}{|c|c|}\hline \alpha & \beta \\ \hline \gamma & \theta \\ \hline \end{array} \quad \longrightarrow \quad \boxed{\max\{\alpha, \beta, \gamma, \theta\}} \tag{10}$$

Thanks to pooling layers, deeper layers' filters will cover a larger region of the input than equal-sized filters in the lower layers. We say that deeper pooling layers have larger receptive fields in terms of the original image.

1.4.2 Non-linearity in Convolutional Layers

After convolution operation, it is common to add a non-linear activation function to introduce non-linearity. We are doing this because convolution is a linear operation and stacking convolution layers together without non-linearity is no more powerful than a single linear layer (possibly a fully connected layer). For example, the order of the layers could be

$$\text{Image} \to \underbrace{\text{Convolution} \to \text{ReLU activation}}_{\text{Convolution Layer}} \to \text{Max Pooling} \to \text{Convolution} \to \cdots \tag{11}$$

1.4.3 Equivariance and Invariance

TODO: I have no idea...

We want our network's responses to be robust to translations of the input, which could mean the following two things

- Convolution layers are equivariant[2]: if you translate the inputs, the outputs are translate by the same amount.
- Network's predictions are invariant: if you translate the inputs, the prediction should not change. Pooling layers provide invariance to small translations.

1.4.4 Channels in Convolution Layers

Each layer is consist of several feature maps, or ***channels***, each of which is an array (and of the same size). In the case where input is an image, then usually we would have 3 channels for coloured input (RGB channels) and 1 channel if it is in greyscale. Each unit is connected to each unit within its receptive field in the previous layer. This includes *all* of the previous layer's feature maps.

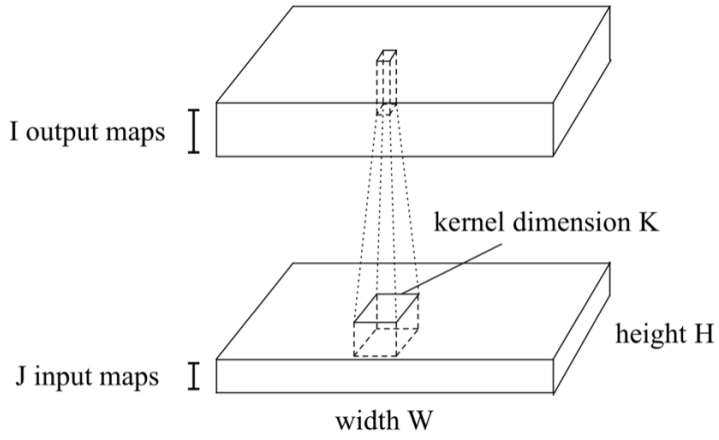
[2]Equivariant means roughly"unchanged in terms of distortion"

1.5 Size of Convolutional Neural Nets

There are several "measures of sizes" that are interested in, namely

- ***Number of units:*** measures the activations needed to be stored in memory during training for back propagation.
- ***Number of weights:*** since weights need to be stored (and updated at each iteration) in memory.
- ***Number of connections:*** measures the computation costs; approximately 3 add-multiply operations per connection. (1 for the forward pass, and 2 for backward pass.)

Below is the number of parameters in a fully connected multilayer perceptron layer along side with those in a convolutional layer. [1]



	fully connected layer	convolution layer
# output units	WHI	WHI
# weights	W^2H^2IJ	K^2IJ
# connections	W^2H^2IJ	WHK^2IJ

References

[1] Jimmy Ba. Csc421/2516 lecture 5: Convolutional neural network & image classification, 2020.