Layers of a Convolutional Neural Network

Convolutional neural networks are built by concatenating individual blocks that achieve different tasks. These building blocks are often referred to as the layers in a convolutional neural network. In this section, some of the most common types of these layers will be explained in terms of their structure, functionality, benefits and drawbacks.

1. Convolutional Layer
2. Non-Linearity Layer
3. Rectification Layer
4. Rectified Linear Units (ReLU)
5. Pooling Layer
6. Fully Connected Layer
7. Literature
8. Weblinks

**This section is an excerpt from Convolutional Neural Networks**

**Convolutional Layer**

The main task of the convolutional layer is to detect local conjunctions of features from the previous layer and mapping their appearance to a feature map. As a result of convolution in neuronal networks, the image is split into perceptrons, creating local receptive fields and finally compressing the perceptrons in feature maps of size $m_2 \times m_3$. Thus, this map stores the information where the feature occurs in the image and how well it corresponds to the filter. Hence, each filter is trained spatial in regard to the position in the volume it is applied to.

In each layer, there is a bank of $m_1$ filters. The number of how many filters are applied in one stage is equivalent to the depth of the volume of output feature maps. Each filter detects a particular feature at every location on the input. The output $Y_{i}^{(l)}$ of layer $l$ consists of $m_1^{(l)}$ feature maps of size $m_2^{(l)} \times m_3^{(l)}$. The $i^{th}$ feature map, denoted $Y_{i}^{(l)}$, is computed as

$$Y_{i}^{(l)} = B_{i}^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} K_{ij}^{(l)} \ast Y_{j}^{(l-1)}$$

(1)

where $B_{i}^{(l)}$ is a bias matrix and $K_{ij}^{(l)}$ is the filter of size $2h_1^{(l)} + 1 \times 2h_2^{(l)} + 1$ connecting the $j^{th}$ feature map in layer $(l-1)$ with $i^{th}$ feature map in layer $l$.

The result of staging these convolutional layers in conjunction with the following layers is that the information of the image is classified like in vision. That means that the pixels are assembled into edglets, edglets into motifs, motifs into parts, parts into objects, and objects into scenes.

---

**Non-Linearity Layer**

A non-linearity layer in a convolutional neural network consists of an activation function that takes the feature map generated by the convolutional layer and creates the activation map as its output. The activation function is an element-wise operation over the input volume and therefore the dimensions of the input and the output are identical.

In other words; let layer $l$ be a non-linearity layer, it takes the feature volume $Y_{i}^{(l-1)}$ from a convolutional layer $(l-1)$ and generates the activation volume $Y_{i}^{(l)}$. 

Original version of the page created by Simon Pöcheim can be found here.
\[ Y_i^{(l)} = f(Y^{(l-1)}_i) \]

with,

\[ Y^{(l)}_i \in \mathbb{R}^{m_{1}^{(l)} \times m_{2}^{(l)} \times m_{3}^{(l)}} \]
\[ Y^{(l-1)}_i \in \mathbb{R}^{m_{1}^{(l-1)} \times m_{2}^{(l-1)} \times m_{3}^{(l-1)}} \]
\[ m_{1}^{(l)} = m_{1}^{(l-1)} \land m_{2}^{(l)} = m_{2}^{(l-1)} \land m_{3}^{(l)} = m_{3}^{(l-1)} \]

This can also be interpreted as \( m_{1}^{(l-1)} \) number of 2-dimensional feature maps (generated by \( m_{1}^{(l-1)} \) number of filters in convolutional layer \((l-1)\)) each with size \( m_{2}^{(l-1)} \times m_{3}^{(l-1)} \).

In some publications\(^2\) a gain coefficient is added to the activation function in order to cope up with the vanishing gradient problem:

\[ Y_i^{(l)} = g_i f(Y^{(l-1)}_i) \]

Similar to multilayer perceptrons, the activation function is generally implemented as logistic (sigmoid) or hyperbolic tangent functions. However, more recent research suggests rectified linear units (ReLUs) are advantageous over the traditional activation functions particularly in convolutional neural networks \(^3\).

It is noteworthy that, although this wiki separates the non-linearity layer from convolutional layer, it is not uncommon to see a combination of the non-linearity layer with the convolutional layer in the literature. Some of the works use different notations for their layer and architecture description. In some publications\(^2\) of Yann LeCun for example, the combined layer is named a filter bank layer and represented as \( F_{CSC} \) denoting a convolutional layer \((C)\) with a hyperbolic tangent activation function \((S)\) and gain coefficients \((G)\).

**Rectification Layer**

A rectification layer in a convolutional neural network performs element-wise absolute value operation on the input volume (generally the activation volume). Let layer \( l \) be a rectification layer, it takes the activation volume \( Y^{(l-1)}_i \) from a non-linearity layer \((l-1)\) and generates the rectified activation volume \( Y^{(l)}_i \):

\[ Y_i^{(l)} = |Y_i^{(l-1)}| \]

Similar to the non-linearity layer, the element-wise operation properties do not change the size of the input volume and therefore, these operations can be (and in many cases\(^7\) including AlexNet\(^4\) and GoogLeNet\(^5\) are) merged into a single layer:

\[ Y_i^{(l)} = |f(Y_i^{(l-1)})| \]

Regardless of the general simplicity of the operation, it plays a key role in the performance of the convolutional neural network by eliminating cancellation effects in subsequent layers. Particularly when an average pooling method is utilized, the negative values within the activation volume are prone to cancel out the positive activations, degrading the accuracy of the network significantly. Therefore, the rectification is named as a “crucial component” \(^2\).

**Rectified Linear Units (ReLU)**
The rectified linear units (ReLUs) are a special implementation that combines non-linearity and rectification layers in convolutional neural networks. A rectified linear unit (i.e. thresholding at zero) is a piecewise linear function defined as:

$$Y_i^{(l)} = \max(0, Y_i^{(l-1)})$$

The rectified linear units come with three significant advantages in convolutional neural networks compared to the traditional logistic or hyperbolic tangent activation functions:

- Rectified linear units propagate the gradient efficiently and therefore reduce the likelihood of a vanishing gradient problem that is common in deep neural architectures.
- Rectified linear units threshold negative values to zero, and therefore solve the cancellation problem as well as result in a much more sparse activation volume at its output. The sparsity is useful for multiple reasons but mainly provides robustness to small changes in input such as noise.
- Rectified linear units consist of only simple operations in terms of computation (mainly comparisons) and therefore much more efficient to implement in convolutional neural networks.

As a result of its advantages and performance, most of the recent architectures of convolutional neural networks utilize only rectified linear unit layers (or its derivatives such as noisy or leaky ReLUs) as their non-linearity layers instead of traditional non-linearity and rectification layers.

In works such as AlexNet\(^{(4)}\) rectified linear units are shown to operate six times faster then hyperbolic tangent non-linearities while reaching 25% error rate on CIFAR-10 dataset\(^{(2)}\). More recently, utilization of an advanced derivative Parametric Rectified Linear Units (PReLU) allowed convolutional networks to surpass human-level performance in ImageNet database\(^{(9)}\). More information on the advantages of the rectified linear units in convolutional neural networks can be found in\(^{(3)}\).

### Pooling Layer

The pooling or downsampling layer is responsible for reducing the spacial size of the activation maps. In general, they are used after multiple stages of other layers (i.e. convolutional and non-linearity layers) in order to reduce the computational requirements progressively through the network as well as minimizing the likelihood of overfitting.

The pooling layer \(l\) has two hyperparameters, the spatial extent of the filter \(F^{(l)}\) and the stride \(S^{(l)}\). It takes an input volume of size \(m_1^{(l-1)} \times m_2^{(l-1)} \times m_3^{(l-1)}\) and provides an output volume of size \(m_1^{(l)} \times m_2^{(l)} \times m_3^{(l)}\) where:

\[
\begin{align*}
m_1^{(l)} &= m_1^{(l-1)} \\
m_2^{(l)} &= (m_2^{(l-1)} - F^{(l)}) S^{(l)} + 1 \\
m_3^{(l)} &= (m_3^{(l-1)} - F^{(l)}) S^{(l)} + 1
\end{align*}
\]

The key concept of the pooling layer is to provide translational invariance since particularly in image recognition tasks, the feature detection is more important compared to the feature’s exact location. Therefore the pooling operation aims to preserve the detected features in a smaller representation and does so, by discarding less significant data at the cost of spatial resolution.

The pooling layer operates by defining a window of size \(F^{(l)} \times F^{(l)}\) and reducing the data within this window to a single value. The window is moved by \(S^{(l)}\) positions after each operation similarly to the convolutional layer and the reduction is repeated at each position of the window until the entire activation volume is spatially reduced.
It is noteworthy that the window for pooling layers does not have to be a square and can be parametrised with \( F_1 \) and \( F_2 \) resulting in a rectangle of size \( F_1 \times F_2 \). However, this is extremely uncommon and is therefore left out of the notation in most of the publications, including this wiki.

The most common methods for reduction are max pooling and average pooling. Max pooling operates by finding the highest value within the window region and discarding the rest of the values. Average pooling on the other hand uses the mean of the values within the region instead.

Max pooling has demonstrated faster convergence and better performance in comparison to the average pooling and other variants such as \( l^2 \)-norm pooling \(^9\). Thus, recent work generally trends towards max pooling or similar variants.

Apart from the reduction method, the hyperparameter selections determine whether the pooling windows overlap or not. In case \( F > S \) the pooling windows overlap on top of each other, filtering some of the data points multiple times. Depending on this condition, a pooling layer is named overlapping or non-overlapping pooling.

Even though the overlapping window method has been used with success to some extent and some of the research \(^4\) suggests that it can reduce the chance of overfitting on certain datasets, the larger spatial size \( F \) in pooling layers is generally considered destructive \(^13\). Therefore, the most commonly used pooling parameters are \( F = 2, S = 2 \) which is non-overlapping.

The selection of \( F = 2, S = 2 \) is the smallest feasible integer sized filter, however, it still discards 75\% of the data. This aggressive reduction executed by pooling layers can limit the depth of a network and ultimately limit the performance. This problem has pushed the research to find other methods to improve or replace the pooling layer.

One approach is to use smaller "fractional" filters \(^10\) instead of common ones. Another option is to remove pooling layers completely and simply perform the reduction by increasing the stride in convolutional layers \(^11\). These solutions still remain as active research topics.

It is important to note that some of the cutting edge architectures such as GoogLeNet’s Inception Modules also feature \( 1 \times 1 \) convolutions to reduce the dimensions of the activation volume before a \( 3 \times 3 \) or \( 5 \times 5 \) convolutional layer.

These are not considered as pooling layers since the reduction is not happening strictly spatially but along the depth axis. However, the architecture of the GoogLeNet still features max and average pooling layers between the inception modules apart from its \( 1 \times 1 \) convolutions. \(^5\)

---

*GoogLeNet’s Inception Module architecture depicting the underlying \( 1 \times 1 \) convolutions. (Image source \(^5\))*

**Fully Connected Layer**
The fully connected layers in a convolutional network are practically a multilayer perceptron (generally a two or three layer MLP) that aims to map the $m^{(i-1)}_1 \times m^{(i-1)}_2 \times m^{(i-1)}_3$ activation volume from the combination of previous different layers into a class probability distribution. Thus, the output layer of the multilayer perceptron will have $m^{(i-1)}_1$ outputs, i.e. output neurons where $i$ denotes the number of layers in the multilayer perceptron.

The key difference from a standard multilayer perceptron is the input layer where instead of a vector, an activation volume is taken as the input. As a result the fully connected layer is defined as:

If $l = 1$ is a fully connected layer;

$$y^{(l)}_i = f(z^{(l)}_i) \quad \text{with} \quad z^{(l)}_i = \sum_{j=1}^{m^{(l-1)}_1} w^{(l)}_{ij} y^{(l-1)}_j$$

otherwise;

$$y^{(l)}_i = f(z^{(l)}_i) \quad \text{with} \quad z^{(l)}_i = \sum_{j=1}^{m^{(l-1)}_1} \sum_{k=1}^{m^{(l-1)}_2} \sum_{l=1}^{m^{(l-1)}_3} w^{(l)}_{ijkl} y^{(l-1)}_j$$

The goal of the complete fully connected structure is to tune the weight parameters $w^{(l)}_{ij}$ or $w^{(l)}_{ijkl}$ to create a stochastic likelihood representation of each class based on the activation maps generated by the concatenation of convolutional, non-linearity, rectification and pooling layers. Individual fully connected layers operate identically to the layers of the multilayer perceptron with the only exception being the input layer.

It is noteworthy that the function $f$ once again represents the non-linearity, however, in a fully connected structure the non-linearity is built within the neurons and is not a separate layer.

As a contradiction, according to Yann LeCun, there are no fully connected layers in a convolutional neural network and fully connected layers are in fact convolutional layers with a $1 \times 1$ convolution kernels. This is indeed true and a fully connected structure can be realized with convolutional layers which is becoming the rising trend in the research.

As an example; the AlexNet generates an activation volume of $512 \times 7 \times 7$ prior to its fully connected layers dimensioned with $4096$, $4096$, $1000$ neurons respectively. The first layer can be replaced with a convolutional layer consisting of $4096$ filters, each with a size of $m_1 \times m_2 \times m_3 = 512 \times 7 \times 7$ resulting in a $4096 \times 1 \times 1$ output, which in fact is only a 1-dimensional vector of size $4096$.

The architecture of AlexNet also depicting its dimensions including the fully connected structure as its last three layers.

Subsequently, the second layer can be replaced with a convolutional layer consisting of $4096$ filters again, each with a size of $m_1 \times m_2 \times m_3 = 4096 \times 1 \times 1$ resulting in a $4096 \times 1 \times 1$ output once again. Ultimately, the output layer can be replaced with a convolutional layer consisting of $1000$ filters, each with a size of $m_1 \times m_2 \times m_3 = 4096 \times 1 \times 1$ resulting in a $1000 \times 1 \times 1$ output, which yields the classification result of the image among $1000$ classes.

**Literature**

[1] Convolutional networks and applications in vision (2010, Yann LeCun, Koray Kavukcuoglu and Clement Farabet)


Weblinks


