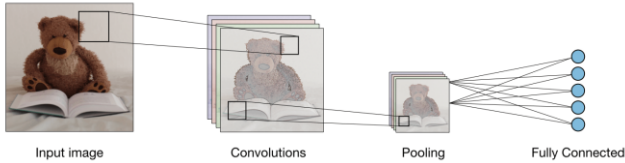


1 ARCHITECTURE OF A TRADITIONAL CNN

Convolutional Neural Networks (CNNs) are specific types of neural networks commonly used for tasks like image classification. They consist of several layers designed to capture hierarchical features from input data.



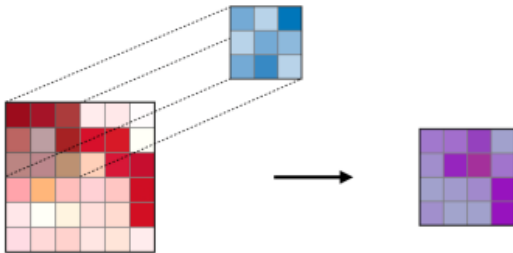
1.1 Layers in a CNN

A typical CNN architecture includes the following layers:

- **Convolutional Layer (CONV)**
- **Pooling Layer (POOL)**
- **Fully Connected Layer (FC)**

1.1.1 Convolutional Layer (CONV)

The convolutional layer uses filters to scan the input data, performing convolution operations with respect to its dimensions. The filter size F and stride S are key hyperparameters. The resulting output is an activation map.



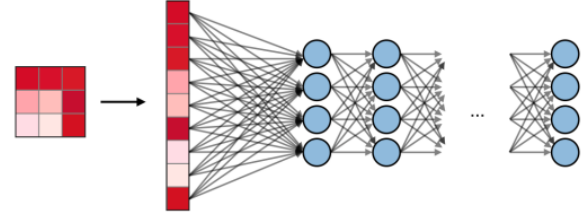
1.1.2 Pooling Layer (POOL)

The pooling layer downsamples the data, often after a convolutional layer, to achieve spatial invariance. Max pooling selects the maximum value, while average pooling calculates the average value within a region.

	Max pooling	Average pooling
Purpose	Each pooling operation selects the maximum value of the current view	Each pooling operation averages the values of the current view
Illustration		
Comments	- Preserves detected features - Most commonly used	- Downsamples feature map - Used in LeNet

1.1.3 Fully Connected Layer (FC)

The fully connected layer operates on flattened input, connecting each input to all neurons. FC layers, usually at the end of CNNs, optimize objectives such as class scores.



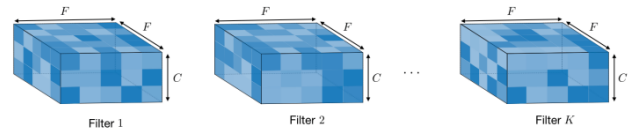
Type	Description
Convolutional (CONV)	Performs convolution operations using filters on input data, producing activation maps.
Pooling (POOL)	Downsampling operation, often applied after convolution, preserving important features.
Fully Connected (FC)	Operates on flattened input, optimizing objectives like class scores.

1.2 Filter Hyperparameters

The convolutional layer utilizes several hyperparameters that affect its behavior:

1.2.1 Filter Dimensions:

A filter of size $F \times F$ applied to an input containing C channels is a $F \times F \times C$ volume that performs convolutions on an input of size $I \times I \times C$ and produces an output feature map (also called an activation map) of size $O \times O \times 1$. The application of K filters of size $F \times F$ results in an output feature map of size $O \times O \times K$.



1.2.2 Stride:

For a convolutional or a pooling operation, the stride S denotes the number of pixels by which the window moves after each operation.



1.2.3 Padding:

Padding involves adding extra pixels around the input data to preserve its spatial dimensions. Zero padding is common to control the output size after convolution.

2 LEARNING CAPACITY , COMPUTE COMPLEXITY

In order to assess the complexity of a model, it is often useful to determine the number of parameters that its architecture will have. Let x denotes input to the layer, y denotes output obtained from the layer, w denotes trainable parameters or weights, and b denotes the bias. In a given layer of a convolutional neural network, it is done as follows:

For $y = \text{Conv2D}(x)$:

$$\begin{aligned} x &\in \mathbb{R}^{C_{\text{in}} \times M \times N}, \\ y &\in \mathbb{R}^{C_{\text{out}} \times M' \times N'}, \\ w &\in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times h \times w}, \\ b &\in \mathbb{R}^{C_{\text{out}} \times 1}. \end{aligned}$$

For $y = \text{MaxPool2D}(x)$:

$$\begin{aligned} x &\in \mathbb{R}^{C_{\text{in}} \times M \times N}, \\ y &\in \mathbb{R}^{C_{\text{out}} \times M' \times N'}, \\ C_{\text{out}} &= C_{\text{in}}. \end{aligned}$$

For $y = \text{FC}(x)$:

$$\begin{aligned} x &\in \mathbb{R}^{N \times 1}, \\ y &\in \mathbb{R}^{M \times 1}, \\ w &\in \mathbb{R}^{M \times N}, \\ b &\in \mathbb{R}^{M \times 1}. \end{aligned}$$

For $y = \text{Sigmoid}(x)$:

$$\begin{aligned} x &\in \mathbb{R}^{N \times 1}, \\ y &\in \mathbb{R}^{M \times 1}, \\ M &= N. \end{aligned}$$

$$LC(\text{Conv2D}(\cdot)) = C_{\text{out}}(hwC_{\text{in}} + 1)$$

$$CC(\text{Conv2D}(\cdot)) = 2 \left(\frac{M-h}{s_h} + 1 \right) \left(\frac{N-w}{s_w} + 1 \right) \frac{hwC_{\text{in}}}{C_{\text{out}}}$$

$$LC(\text{MaxPool2D}(\cdot)) = 0$$

$$CC(\text{MaxPool2D}) = C_{\text{in}} \left(\frac{M-h}{s_h} + 1 \right) \left(\frac{N-w}{s_w} + 1 \right) (hw - 1)$$

$$LC(\text{FC}) = M(N + 1)$$

$$CC(\text{FC}) = 2MN$$

$$LC(\text{Sigmoid}) = 0$$

$$CC(\text{Sigmoid}) = 3N$$

Output Tensor Size :

For Convolution Layer (CONV)

For a convolution layer with input tensor of size $C_{\text{in}} \times H \times W$ and output tensor of size $C_{\text{out}} \times H' \times W'$, using a filter of size $F \times F$, stride S , and padding P :

$$H' = \frac{H + 2P - F}{S} + 1 \quad \text{and} \quad W' = \frac{W + 2P - F}{S} + 1$$

For Max-Pooling Layer (MAXPOOL)

For a max-pooling layer with input tensor of size $C_{\text{in}} \times H \times W$ and output tensor of size $C_{\text{out}} \times H' \times W'$, using a pooling window of size $P \times P$ and stride S :

$$H' = \frac{H - P}{S} + 1 \quad \text{and} \quad W' = \frac{W - P}{S} + 1$$

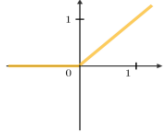
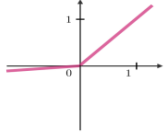
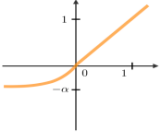
$$C_{\text{out}} = C_{\text{in}}$$

3 COMMONLY USED ACTIVATION FUNCTIONS

Activation functions introduce non-linearity to neural networks, allowing them to learn complex relationships in the data.

3.1 Rectified Linear Unit (ReLU)

The rectified linear unit layer (ReLU) is an activation function that is used on all elements of the volume. It aims at introducing non-linearities to the network. Its variants are summarized in the table below:

ReLU	Leaky ReLU	ELU
$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$	$g(z) = \max(\alpha(e^z - 1), z)$ with $\alpha \ll 1$
		
Non-linearity complexities biologically interpretable	Addresses dying ReLU issue for negative values	Differentiable everywhere

3.2 Softmax

The Softmax activation function is often used in the output layer of a neural network for multiclass classification problems. It converts a vector of real numbers into a probability distribution.

Formula:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

Where:

z_i : Input to the Softmax function for class i

K : Total number of classes

The Softmax function ensures that the sum of the probabilities of all classes equals 1, making it suitable for classification tasks.