Convolutional Neural Networks (CNNs / CovNets)

**Architecture Overview**

Convolutional Neural Networks (CNN) appear like other Artificial Neural Networks (ANN) as both are made up of neurons that learn weights and biases during the training. Each neuron receives the input usually in the form of vectors, performs dot operation and sometimes implements non-linearity as a follow up step. This input goes through multiple hidden layers for processing and applying further non-linearity. They still use a loss function like SVM/Softmax on the last fully connected layer(output-layer) to determine the Class probabilities / scores for the input pixel image.

CovNets are designed specifically to address image classification problems. This assumption allows the CNN feed-forward mechanism most efficient and greatly reduces the no. of parameters in the whole CNN network.

Regular Neural Nets don’t scale well to process and classify higher resolution images because of large no. of weights generated by fully-connection network.

CovNets take the advantage of the fact that input is an image and adjust the architecture in such a sensible way and layers of CovNet neurons are arranged in 3 dimensions like **width, height, depth.** Instead of fully-connected layers, the neurons in each layer are only be connected to small section of layer before it. Also, out layer has typically has just of no. of neurons equal to no. of image class categories.

Below is a simple comparison between 3-layered regular Neural Network on the left and CovNet on the right side. We see that CovNet arranges its neurons in three dimensions (width, height, depth). Each layer in CovNet transforms it’s input 3D volume to an output 3D volume of neuron activations. Input layer (red box) represents an image and this layer has the same dimensions of the image

|  |  |
| --- | --- |
| http://cs231n.github.io/assets/nn1/neural_net2.jpeg | http://cs231n.github.io/assets/cnn/cnn.jpeg |

**CovNets – Layers Overview**

There are three main layers that make up a CovNet architecture:  **Convolutional Layer (CONV)**, **Pooling Layer (POOL)**, and **Fully-Connected Layer (FC). These 3 layers are stacked to form a complete CovNet architecture. CovNets transform the input image layer by layer from original pixel values to the final class scores as shown in the below example CovNet, a tiny VGG Net, architecture.**



**Each layer in CovNet may or not may not have parameters. For example, CONV/FC layers have parameters while RELU/POOL layers don’t. Similarly, CONV/FC/POOL layers have hyper-parameters while RELU doesn’t have as such.**

**Convolutional Layer (CONV)**

**The Convolutional Layer is the core building block of CovNet as it does perform heavy computational tasks so tuning this later for efficiency can improve the overall performance of CovNet. The layer’s parameters are made up of set of learnable filters (aka kernels) that have small receptive field however they extend through the full depth of input volume. While performing forward pass, each filter is convolved across the width and height of the input volume and then computes the dot product between the filter entries and input to produce the filter activation map. With the convolution process, CovNet learns the filters that activate when it detects specific feature.**

**Activation maps for all filters along the depth dimension are stacked to form the full output volume of the convolution layer.**

**Example 1:**

**For example, let’s consider an in input image as volume that has size [32x32x3] ( ex: an RGB of CIFAR-10 image). If we are using filter of size 5x5 applied on the small receptive field of the original image, then each neuron in the convolution layer will have weights to a 5x5x3 region in the input volume for a total of 5\*5\*3 = 75 weights. We must note that each neuron has a depth of connectivity of 3 which is the depth of input volume. There are multiple neurons (here 5) along the depth and all of them are looking at the same region in the input.**

**Each neuron still functions still functions the same calculating the dot product of their weights with the input followed by some non-linearity function.**

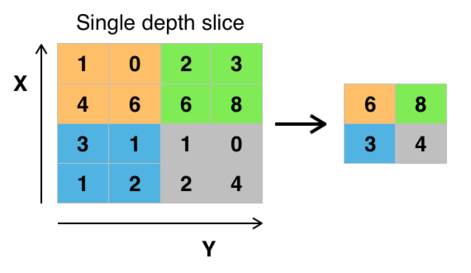
|  |  |
| --- | --- |
|  |  |

**Left:**

**Pooling Layer (POOL)**

**Pooling is one of important CNN concept specifically designed for non-linear down-sampling. There exist multiple non-linear functions to implement pooling and ‘max pooling’ happens to be the most popular choice. It divides the input image into a set of non-overlapping small sections and it outputs the maximum for each such section. The intuition behind this is that exact location of a feature has lesser importance than it’s rough location relative to other features. Pooling helps to reduce the spatial size of its input dimension so as to reduce the number of parameters and amount of computation. This also overcomes the overfitting problem.**

**Typically, pooling layers are inserted between successive convolution layers of a CNN architecture. Most common pooling layers apply filters of size 2x2 with a stride of 2 down-samples at every depth slice in the input by 2 along both width and height. This results in discarding of 75% of the activation but depth dimension remains unchanged. Current trend is to use smaller filter sizes due to aggressive reduction in the size of the representation.**

[](https://en.wikipedia.org/wiki/File:Max_pooling.png)

**Fig: Max pooling with a 2x2 filter and stride = 2**

**ReLU Layer**

**ReLu stands for ‘Rectified Linear Units’. This layer applies the non-saturating activation function f(x) = max(0, x). Without affecting the receptive fields of CovNets, this layer increases the non-linear properties of the decision function and of overall network.**

**We can use other functions also to increase the non-linearity and such examples include hyperbolic tangent f(x) = tanh(x) and the sigmoid function f(x)\_ = 1/ (1+ pow(e,-x)). ReLU is the often the most preferred function because of expedited training speed of the network without incurring much penalty for generalization.**

**Fully-Connected Layer (FC)**

After sequence of convolution and pooling layers, the final Fully-Connected (FC) layer forms a basis for high-level reasoning. In this layer, neurons connect fully to all the activations in the previous layer just like observed in typical regular neural networks. Hence Matrix multiplication with the addition of Bias offset is just enough to compute activations and implement this layer.

**Loss Layer**

This final layer determines how training should penalize the deviation between the predicted and true labels. We use different loss functions appropriate for the given task in this layer. Sofmax loss is used for predicting a single class of K mutually exclusive classes. Sigmoid cross-entropy loss is used for predicting K independent class probability values in [0,1] range. Similarly, Euclidian loss is used for regressing to real-values labels (-∞, ∞).

<http://cs231n.github.io/convolutional-networks/>

<https://en.wikipedia.org/wiki/Convolutional_neural_network>