0.1 Convolutional Neural Networks

- *Algorithm:* Convolutional Neural Networks (algo. ??)
- *Input:* Datas, Images, etc.
- *Complexity:* N/A
- *Data structure compatibility:* N/A
- *Common applications:* Image recognition, video analysis, etc.

Problem. Convolutional Neural Networks

Inspired by biological processes, convolutional neural networks(CNN, ConvNets) are a class of deep neural networks that employs a mathematical operation call convolution instead of general matric multiplication in their layers and have been widely used for computer vision.

Description

Basics A convolutional neural network is a feed-forward neural network, and it is designed for processing structured arrays of data like images. Like neural networks, ConvNets are also made up of neurons with learnable weights and biases but due to the convolutional layers inside, they do well in extracting patterns or features(lines, circles, etc.) from the input images. This is the reason why it has been widely used in fields like computer vision and has become the state of the art for many visual applications like image recognition, video analysis.

Network Architecture A convolutional neural networks is actually a multi-layered feed-forward neural network with input, output layers and many hidden layers stacked on top of each other but compared with an ordinary neural network, layers in ConvNets have their neurons arranged in 3 dimensions: width, height, and depth and each layer transforms an input 3D volume to an output 3D volume with some differentiable functions. Layers are including convolutional layers, pooling layers and so on. This sequential design allows ConvNets to learn hierarchical features. Below figure shows a simple ConvNet.

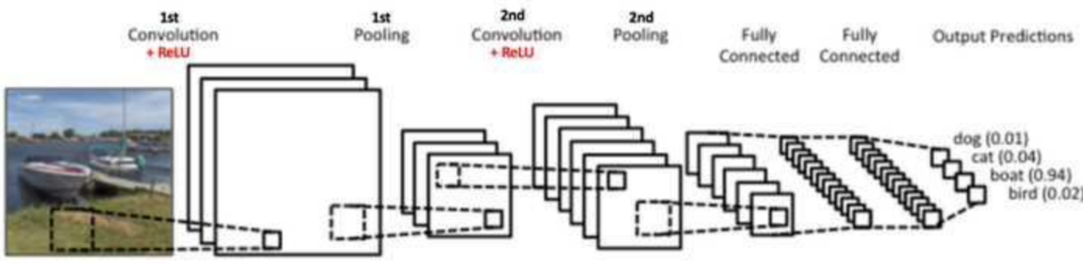

Figure 1: A simple ConvNets

The input of this net is an image. Convolutional layers extract and create feature maps that explain the input in different ways while the pooling layers reduce the spatial dimensions to control the amount of parameters. As going deeper, the representations are getting more and more abstract and finally getting flatten and fed into fully-connected layers to determine the likelihood of each target class. The detailed description of these layers are given below.

• **Convolutional Layers** Convolutional layers are the core of the ConvNets. Typically, a convolutional layer contains many learnable **kernels**(filters) which are small square templates. Kernels slide over the input feature map and do dot products between the input at any position and the entries of the kernels. After this, we will produce a 2-dimensional activation map that gives responses of that kernel and we stack these activation maps along the depth dimension to produce the output. When training, kernels from the first to later layers will all learn to become activate when they see some specific types of feature. Unlike a fully-connected neural network where neurons are all connected with each other between layers, in ConvNet, we do **local connectivity** that we just connect each neuron to a specific local region(receptive field) of the input and this is just the kernel size. Also, we adopt **parameter sharing** to control number of parameters. That is, we make each kernel responsible for a particular type of features on the input because if one feature is useful at some position, then it should be useful at any different position of the input.

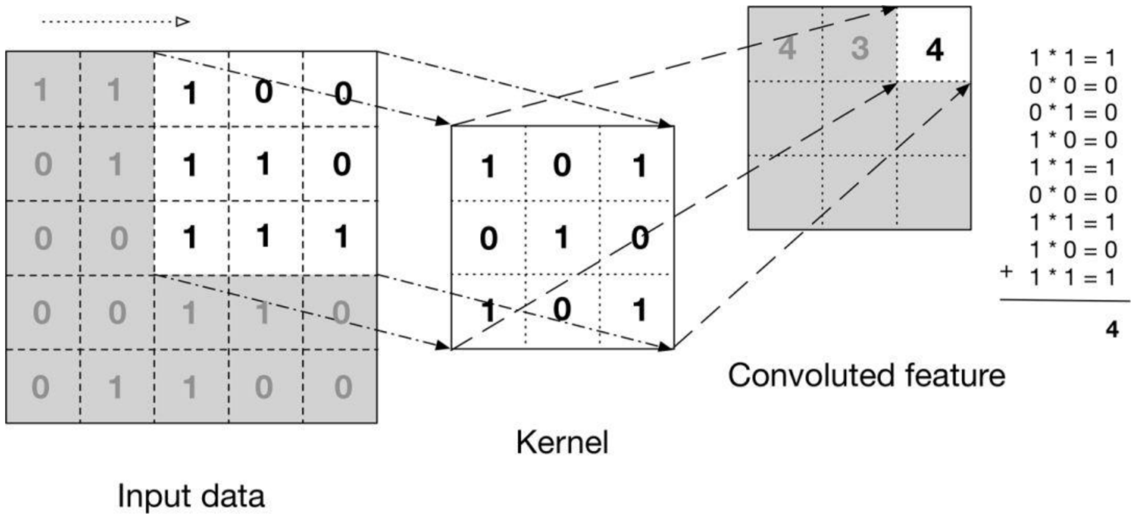

Figure 2: A sample of how convolutional layers work

There are some hyperparameters that decide the size of the output of one convolutional layer: **depth**, **stride**, and **zero-padding**. Depth is corresponding to the numbers of kernels we use in one convolutional layer. Stride decides how fast the kernels move over the input and 1 means that the kernel slide 1 pixel per time. Zero-padding is about padding the input with zeros around its border to control the output size.

With above hyperparameters, let's assume the size of the input is $(w \times w \times c)$ and kernels are $(f \times f \times c)$ with s as its stide. The amount of zero padding is p. Thus, the output size is calculated by

$$size = \frac{w - f + 2p}{s} + 1$$

Usually, set $p = \frac{f-1}{2}$ and $s = 1$ makes the input and the output have the same size. Also, sometimes $(w - f - 2p)$ cannot get divided by s and then we can just throw an exception or zero pad the rest to make it valid.

• **Pooling Layers** Pooling layers are another important types of layers in ConvNets and are usually get put between convolutional layers. They are also called subsampling or down-sampling. The main task of pooling layers is to reduce the parameters that needed to extract features in following layers by reducing the spatial dimensions of the input representation but retains its important informations. Pooling layers also contain filter with size and stride. However, it works independently on each depth slice of the input and combines the neuron clusters at each depth into a single neuron in the next layer. Given input of size $(w \times h \times d)$ and the filter of size f with stride s. Note that zero-paddings are not commonly used in pooling layers. Then, the output has size

$$(W = \frac{w - f + (2p)}{s} + 1,\ H = \frac{h - f + (2p)}{s} + 1,\ D = d)$$

There are some different types of pooling: Max pooling(most commonly used), Average pooling and Sum pooling. In max pooling, MAX operation just returns max value over each $f \times f$ numbers.

Actually, pooling with large receptive fields is destructive so 2×2 filters are commonly used. And in many ConvNets or other models like DCGANs, there are just conv layers but no pooling layers.

• **Fully-Connected Layers** Like an ordinary neural network, in fully-connected layers, neurons in one layer are connect to all the neurons in another layers. Fully-connected layers are often put near the end of ConvNets after several conv layers and pooling layers and the feature maps will get flatten as the input of fully-connected layers.

Other Techniques Besides the basic structures, there are also many useful techniques to prevent over-fiiting and give rise to better performance, more stable training for CNNs.

• **Data Augmentation** A intuitive way to increase the performance of the ConvNets is to increase the amount of the training data. There are many ways to pre-process the image data includinng random rotation, cropping, and adjusting its brightness and so on. GANs may also be used for data augmentation. This technique is actually very helpful and serves as a regularizer and reduce the over-fitting when training.

• **Dropout** Dropout is one kinds of techniques that reduce over-fitting in neural networks. Typically in ConvNets, fully-connected layers contains most amount of parameters which will often result in over-fitting. Using Dropout technique then at each stage of training, neurons are either 'dropped out' with probablity p or kept with probablity $1 - p$. Then, after this stage of training, the dropped neurons remain the same weights. By avoiding training all the neurons on all the training data, dropout decrease the over-fitting and increases the training speed significantly. What's more, this technique often helps the network to learn more robust features and make it performs better.

• **Batch Normalization** Deep networks are hard to train. Batch normalization(Batch norm), proposed by Sergey Ioffe et al., is a technique that has been widely used to train neural networks by standardizing('whitening') the inputs to a layer for each mini batch. It also accelerates training, results in more stable training process and can provide some regularization.

• **Weight Decay** Weight decay is a simple form of added regularizer and it adds an additional penalty term, proportional to the sum of weights(L1) or squared magnitude of the weight vector(L2), to loss function on the training set to reduce the complexity of the learned model and hence it reduces over-fitting and make the model perform better.

Well-known Models There are many important and famous models during the development of CNN. Starting from 1998, LeNet-5 was first introduced [need complement]

• **ResNet**