AIGS 538 Homework Assignment 2 20222421 심재윤 (JAEYOON SIM)

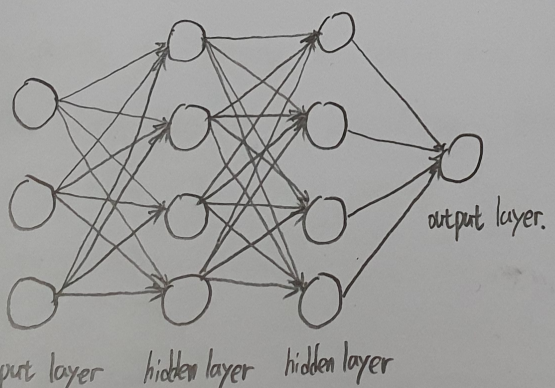
Title : Convolutional Neural Networks for Visual Recognition.

Convolutional Neural Networks (CNN) are very similar to ordinary Neural Networks (NN) and they are made up of some neurons that have learnable weights and biases. Most of the features of ConvNet are similar to an ordinary Neural Network, but the domain of input is limited to image. ConvNet architectures make the explicit assumption that the inputs are images. They can reduce the amount of parameters in the network and make the forward function more efficient.

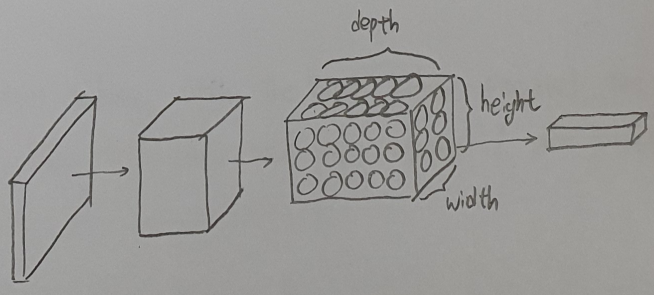
#1. Architecture Overview

In the case of regular Neural Networks they recieve an input and transform it to hidden layers which are made up of several neurons. Each neuron is fully connected to all neurons in previous layer. However, these regular Neural Networks don't scale well to full image. Beccuse they are made of fully connected neurons. This fully connected structure doesn't scale to large images. For example, if the images with 200 x 200 x 3 size, neuron in the input layer should have 200x200x3 weights. So, this full connectivity is wastefull and the huge number of parameteps will lead to overfitting.

However, Convolutional Neural Networks take lots of advantages beccuse neurons in these Networks are arranged in 3 dimensions (width, height, depth). The neurons in a layer will only be connected to a small region of the layer instead of all of the neurons in a fully connected manner. And the final output layer will have single vector because by the end of ConvNet architecture they will reduce the full image input into a single vector when the input dataset is CIFAR-10. Depending on the purpose, the image may be mapped as a low-dimensional vector. The followings are the visualization of regular Neural Networks and ConvNet.



Regular Neural Network (3-layers)



ConvNet

2022421 심재윤 (JAEYOON SIM)

#2. Layers used to build ConvNets

A ConvNet is a sequence of layers which transform one volume of activations to another through a differentiable function. Three main types of layers (convolutional layer, pooling layer, fully-connected layer) are used to build ConvNet architectures and they are exactly same in regular Neural Networks. Each layer may or maynot have parameters and additional parameters.

#2.1. Convolutional Layer.

The Conv layer is the core building block of a Convolutional Network. The Conv layer's parameters consist of a set of learnable filters and every filter is small spatially but extends through the full depth of input volume. During the forward pass, each filter slides across the width and height of input volume and compute dot products between the entries of filter and input. This process will produce 2-dimensional activation map that gives responses of that filter at every spatial position. The network will learn filters that activate when they see visual features like an edge.

It is impractical to connect neurons to all neurons when dealing with high dimensional inputs like images. So, the ConvNet connects each neuron to only local region of the input volume. The spatial extent of this connectivity is hyperparameter called receptive field of the neuron. The connections are local in space along width and height, but always full along the entire depth of the input volume. Three parameters control the size of the output volume and they are the depth, stride and zero-padding.

(1) The depth of the output volume is a hyperparameter and it is same as the number of filters we would like to use. These filters look for something different in same input and set of neurons that are looking at the same area is called depth column.

(2) The stride is about sliding the filter and it is the number of pixel passed when the filter moves at a time. When the stride increases, the output volume decreases spatially.

(3) The size of zero-padding is a hyperparameter and it will be convenient to pad the input volume with zeros around the border. Zero-padding will allow us to control the spatial size of the output volumes.

It is possible to compute the spatial size of the output volume with the input volume size (W), the receptive field size of the Conv Layer neurons (F), the stride (S), and the amount of zero padding (P). From these, The output volume size is computed as $(W-F+2P)/S+1$.

In general, setting zero padding to be $P=(F-1)/2$ when the stride is $S=1$ ensures that the input volume and output volume will have the same size spatially. And there are some constraints on strides. For example, some strides like 2 where $W=10$, $P=0$, $F=3$ is invalid since output

volume becomes 4.5 (= (10-3+0)/2+1) which cannot be done. Therefore, ConvNet library could throw an exception or zero pad the rest to make it fit, or crop the input to make it fit. Also, there is a concept of parameter sharing. Parameter sharing scheme is used in Convolutional Layers to control the number of parameters. If we do not use this parameter sharing scheme, input data with size 227x227x3 and filter with size 11x11x3 results in having 227x227x3 x(11x11x3+1) parameters which is really high. We can dramatically reduce the number of parameters by using parameter sharing scheme and this is constraing the neurons in each depth slice to use the same weights and bias. With this parameter sharing, we will only have 96x11x11x3+96 parameters in the same example. During backpropagation, every neuron in the volume will compute the gradient for its weights, but these gradients will be added up across each depth slice and update a single set of weights per slice. However, there are some cases that the parameter sharing may not make sense. When the input images to a ConvNet have some specific centered structure, that completely different features should be learned on one side of the image than another. In these cases, relaxing parameter sharing scheme and calling the layer a Locally-Connected Layer is common.

We can implement Conv layer with Numpy to understand more concrete. Suppose that the input X is input volume with shape X.shape: (11,11,4) and there are no zero padding (P=0), the filter size is 5 (F=5), the stride is 2 (S=2). Then, the output volume V will have spatial size (11-5)/2+1 = 4, giving a volume with width and height of 4.

```
V[0,0,0] = np.sum(X[:5,:5,:] * W0) + b0
⋮
V[3,0,0] = np.sum(X[6:11,:5,:] * W0) + b0
```

W0 : weight vector
b0 : bias
* : element wise multiplication

We are computing the dot product at each point and also using the same weight and bias due to parameter sharing. Like this, we can construct next activation map in the output volume. About the output depth, it is determined by the number of filters (K) which is another hyper parameter like F, S, and P.

The convolution operation performs dot products between the filters and local regions of input. The Conv layer formulates the forward pass of a convolutional layer as one big matrix multiply. The backpropagation of Conv layer is also convolution but with spatially-flipped filters. There is also 1x1 convolution. 1x1 convolution comes from signal processing background but signals are 2-dimensional so 1x1 convolutions do not make sense. However, in ConvNets this is not the case because we operate over 3-dimensional volumes, and that the filters always extend through the full depth of the input volume. Even though we only discussed Conv filters that are contiguous, there are also filters

that have spaces between each cell, called dilation. The dilated convolutions can be very useful in some settings to use in conjunction with 0-dilated filters because it allows to merge spatial information across the inputs much more agressively with fewer layers.

#2.2. Pooling Layer

Inserting a Pooling layer in-between successive Conv layers in a ConvNet architecture is common. Pooling operation is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, thus to also control overfitting. The pooling layer operates independently on every depth slice of the input and resizes it spatially. It usually use MAX operation among MAX, AVERAGE, and so on. More generally, it accepts a volume of size $W_1 \times H_1 \times D_1$ and requires 2 parameters which are their spatial extent (F) and the stride (S). It produces a volume of size $W_2 \times H_2 \times D_2$ where $W_2 = (W_1 - F)S + 1$, $H_2 = (H_1 - F)/S + 1$, and $D_2 = D_1$. For pooling layers, it is not common to pad the input using zero-padding.

The backpropagation of pooling layer is only routing the gradient to the input that had the highest value in the forward pass. Many people dislike the pooling operation and some papers propose to discard the pooling layer in favor of architecture that only consists of repeated Conv layer. Discarding pooling layers has also been found to be important in training good generative models, such as variational autoencoders (VAEs) or generative adversarial networks (GANs).

#2.3. Normalization Layer

Many types of normalization layers have been proposed for use in ConvNet architecture, but these layers are not used mostly because their contribution has been shown to be minimal.

#2.4. Fully-Connected Layer

Neurons in a fully connected layer have full connections to all activations like in regular Neural Networks.

#2.5. Converting FC layers to CONV layers.

The difference between FC and CONV layers is that the neurons in the CONV layer are connected only to a local region in the input, and many neurons in CONV volume share parameters. However, in both layers, the neurons still compute dot products so it is possible to convert between FC and CONV layers. Of these 2 conversions, the ability to convert an FC layer to a CONV layer is particularly useful in practice. This conversion allows us to "slide" the original ConvNet very efficiently across many spatial positions in a larger image, in a single forward pass.

20222421 심재윤 (JAEYOON SIM)

3. ConvNet Architecture

CONV, POOL, FC are commonly used to make Convolutional Networks and RELU activation function is used for non-linearity.

3.1. Layer Patterns

Most ConvNet architecture stacks a few CONV-RELU layers, follows them with POOL layers, and repeat this pattern until the image has been merged spatially to a small size as follow.

INPUT → [[CONV → RELU] *N → POOL] *M → [FC → RELU] *K → FC

This conventional paradigm of a linear list of layers has recently been challenged, in Google's Inception architectures and also in current Residual Networks from Microsoft Research Asia. In practice, we should look at whatever architecture currently works best on ImageNet, download a pretrained model and finetune it on our data.

3.2. Layer Sizing Patterns

The input layer should be invisible by 2 many times like 32, 64, 96, or 224, 384, and 512. The conv layer should be using small filters like 3x3 or 5x5 using a stride 1 and pad the input so that conv layer does not alter the spatial dimensions of the input. The pooling layers are in charge of downsampling the spatial dimensions of the input. They mostly use 2x2 max-pooling with stride 2. All of these are the rules of thumb for sizing the architectures.

3.3. Case Studies.

LeNet is the first successful applications of Convolutional Networks and it was used to read zip codes, digits, etc. AlexNet is the first work that popularized Convolutional Networks in Computer Vision. It had a very similar architecture to LeNet, but was deeper, bigger, and featured Convolutional Layers stacked on top of each other. ZFNet is the winner of ILSVRC 2013 and it was an improvement on AlexNet. GoogLeNet is dramatically reduced the number of parameters. VGGNet showed that the depth of network is crucial for good performance. ResNet features special skip connections and a heavy use of batch normalization. And it is also missing fully connected layers at the end of the network. As is common with Convolutional Networks, most of the memory is used in the early CONV layers, and that most of the parameters are in the last FC layers.

3.4. Computational Considerations

In ConvNet architectures, we should be awared of memory bottleneck. There are 3 major sources of memory to keep track of. From the intermediate volume size, from the parameter sizes, miscellaneous memory such as image data patches are those. If our network doesn't fit, a common heuristic to "make it fit" is to decrease the batch size, since most of the memory is usually consumed by the activations.