Deep Learning Study Lecture Note

Lecture 5 : What is CNN?

Kim Tae Young

January 18, 2020

5.1 What is CNN?

CNN(Convolutional Neural Network) is another example of neural network, like MLP, which is largely used in problems related to computer vision. Unlike MLP, which uses fully connected layer as connection between nodes, here we use convolutional layer where the name CNN comes from.

Definition 5.1.1. *A convolution of function* $f, g : \mathbb{R} \to \mathbb{R}, f * g(x) = \int f(x - y)g(y)dy$. *This is the definition in analysis.*
A convolution of function $f, g : \mathbb{N} \to \mathbb{R}, f * g(x) = \sum_{b|n, b\in\mathbb{N}} f(b)g(\frac{n}{b})$. *This is the definition in number theory.*
A convolution layer is defined in terms of 'filter', where a filter is a n dimensional tensor. Here, I will simply show 1 dimensional convolution and 2 dimensional convolution.

$$\begin{pmatrix}1\\3\\1\\4\\2\\5\end{pmatrix} * \begin{pmatrix}-1\\2\\-1\end{pmatrix} = \begin{pmatrix}1\times-1+3\times2+1\times-1\\3\times-1+1\times2+4\times-1\\1\times-1+4\times2+2\times-1\\4\times-1+2\times2+5\times-1\end{pmatrix} = \begin{pmatrix}4\\-5\\5\\-5\end{pmatrix}$$

$$\begin{pmatrix}1&2&3&4\\2&3&4&5\\3&4&5&6\\4&5&6&7\\5&6&7&8\end{pmatrix} * \begin{pmatrix}3&-1\\-2&1\end{pmatrix} = \begin{pmatrix}0&1&2\\1&2&3\\2&3&4\\3&4&5\end{pmatrix}$$

Usually when using convolutional layer, the size of input tensor shrinks as seen above. However we usually don't want to have this kind of behavior, so we append padding to input layer. Usual one zero is padded, which is computed as following.

$$\begin{pmatrix}1\\3\\1\\4\\2\\5\end{pmatrix} * \begin{pmatrix}-1\\2\\-1\end{pmatrix} = \begin{pmatrix}0\times-1+3\times-1+1\times2\\1\times-1+3\times2+1\times-1\\3\times-1+1\times2+4\times-1\\1\times-1+4\times2+2\times-1\\4\times-1+2\times2+5\times-1\\2\times-1+5\times2+0\times-1\end{pmatrix} = \begin{pmatrix}-1\\4\\-5\\5\\-5\\8\end{pmatrix}$$

Also, we do not apply one filter for each connection. For example, if our input is a $28 \times 28 \times 3$ RGB pixel image, and want our output to be $28 \times 28 \times 16$ tensor, we use 48 filters of size 3×3.
We can use different size of filters, as well as change stride. Meaning, if we use stride 2 instead of 1 as in above, we get

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 6 \\ 4 & 5 & 6 & 7 \\ 5 & 6 & 7 & 8 \\ 6 & 7 & 8 & 9 \end{pmatrix} * \begin{pmatrix} 3 & -1 \\ -2 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 2 \\ 2 & 4 \\ 4 & 6 \end{pmatrix}$$

The other main layer we use in CNN is pooling layer. Since applying convolutional layer does not decrease size of input tensor, pooling layer's main role is decreasing the size of tensor. The main two pooling layer is max pooling layer and average pooling layer.

$$MaxPool(\begin{pmatrix} 1 & 2 & -1 & 0 \\ 3 & 2 & 0 & 4 \\ -3 & 5 & 2 & -1 \\ 0 & 2 & -1 & 4 \end{pmatrix}, 2, 2) = \begin{pmatrix} 3 & 4 \\ 5 & 4 \end{pmatrix}$$

$$AveragePool(\begin{pmatrix} 1 & 2 & -1 & 0 \\ 3 & 2 & 0 & 4 \\ -3 & 5 & 2 & -1 \\ 0 & 2 & -1 & 4 \end{pmatrix}, 2, 2) = \begin{pmatrix} 2 & 0.75 \\ 1 & 1 \end{pmatrix}$$

Again, backpropagation of neural network can be computed since convolution layer is just simple multiplication. However, you can see that backpropagation is also a convolution.
Since applying convolutional layer only outputs n dimensional tensor, not vector(which is 1 dimensional tensor), we need a final fully connected layer to output.
The main structure of CNN is as following.

$$IN \to [[CONV \to ACT]^N \to POOL]^M \to [FC \to ACT]^K \to FC \to OUT$$

Followings are instance of VGGnet, which was the runner up network at ILSVRC2014.

$$VGG11[64, M, 128, M, 256^2, M, 512^2, M, 512^2, M]$$
$$VGG13[64^2, M, 128^2, M, 256^2, M, 512^2, M, 512^2, M]$$
$$VGG16[64^2, M, 128^2, M, 256^3, M, 512^3, M, 512^3, M]$$
$$VGG19[64^2, M, 128^2, M, 256^4, M, 512^4, M, 512^4, M]$$

Here, number means we will have that number of output dimension. M means max pooling layer, and the exponent means the number of times the layer is applied. Every filter is 3×3 filter with 1 stride and zero padding, and every max pooling is of size2×2.
The neuroscientific narrative of CNN is the sensory system, especially the vision. As in CNN, in vision system, each neurons are locally connected, not fully connected as MLP. This explains why CNN is better, as we only require local information in each layer in vision problem, while MLP fully connects.

Also, filter is one of the main methods in computer vision in processing image, called kernel. The blur or shapening in your favorite photo editors are these kernels. I will present some well used kernels.

Edge detection $\begin{pmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$

Sharpen $\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$

Blur $\begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}, \begin{pmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{pmatrix}$

5.2 Methods in CNN

So in VGGnet, we only used 3×3 filters. Consider compressing 5×5 information to one scalar. If we use 5×5 filter, we require 25 parameters. However if we use two 3×3 filter, we only require 18 parameters while having equal input size. This allows us to increase performance of network with less number of parameters, meaning we need less time to train.
Similar to MLP, we can use both batch normalization and dropout in CNN. However, it is experimentally proven that normalization does not work in CNN, so normalization is not used well.

5.3 Inception

Inception is a model used in advanced CNN. It was first introduced in GoogLeNet, which won first place at ILSVRC2014. As we talked right before, using 3×3 filter was better then using larger filter, However, Inception uses both, as parallel layers. Inception is computed as following, where $\circ$ is concatenation.

$$
\begin{aligned}
NaiveInception(in) =& Conv(in, 1, 1, 128) \\
& \circ Conv(in, 3, 3, 192) \\
& \circ Conv(in, 5, 5, 96) \\
& \circ MaxPool(3, 3)
\end{aligned}
$$

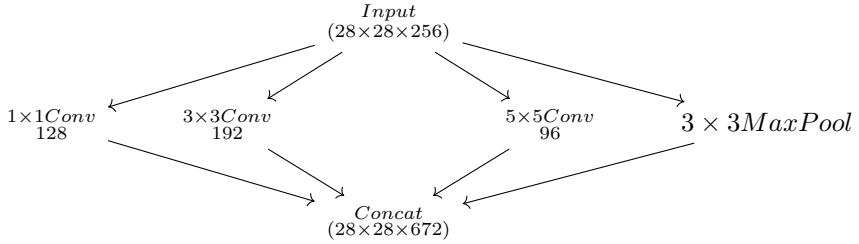

Figure 5.1: Naive Inception

Now consider the number of parameter here. Assume input is size $28 \times 28 \times 256$, and we have 128 1×1 filters, 192 3×3 filters, and 96 5×5 filters. We then have $28 \times 28 \times 256 \times (1 \times 1 \times 128 + 3 \times 3 \times 194 + 5 \times 5 \times 96)$ convolutional operation, which is approximately 854 million operations. So instead, we append 1×1 Conv to reduce depth of input.

$$
\begin{aligned}
Inception(in) =& Conv(in, 1, 1, 128) \\
& \circ Conv(Conv(in, 1, 1, 64), 3, 3, 192) \\
& \circ Conv(Conv(in, 1, 1, 64), 5, 5, 96) \\
& \circ Conv(MaxPool(in, 3, 3), 1, 1, 64)
\end{aligned}
$$

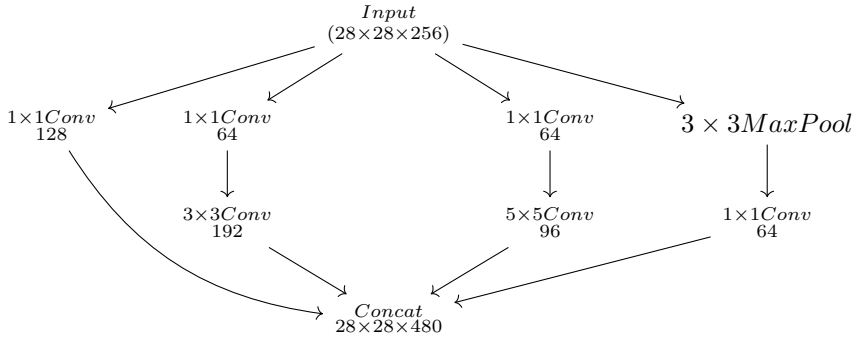

Figure 5.2: Inception

5.4 Auxiliary classifiers

Since GoogLeNet has large depth, gradient vanishing. is of main concern. The motivation here is that as shallow networks also work, middle of the GoogLeNet would also be discriminative. Thus by adding auxiliary classifiers in middle of the network, it will encourage discrimination in earlier stages, and increase gradient signal. The auxiliary classifiers are of the form of small convolutional networks, and their loss is added to total loss with discount weight.

5.5 Residual Connection

Residual Connection is another module used in advanced CNN. It was first introduced in ResNet, which won first place at ILSVRC2015, in all classification. The motivation of ResNet was simple: as we saw in MLP, they increased depth. However as deep networks have gradient vanishing, it is hard to train. To overcome such phenomena, ResNet used residual connection.

$$Res(in) = Conv(ReLU(Conv(in))) + in$$

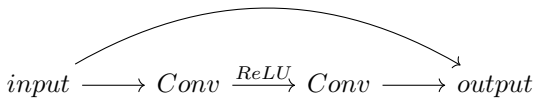

Figure 5.3: Residual Connection

Since the network tries to learn residual, not the full function, it is easier to train. In deeper networks, we use bottleneck layer, which is similar to residual connection.

$$Bottleneck(in) = Conv(Conv(Conv(in, 1, 1, 64), 3, 3, 64), 1, 1, 256) + in$$

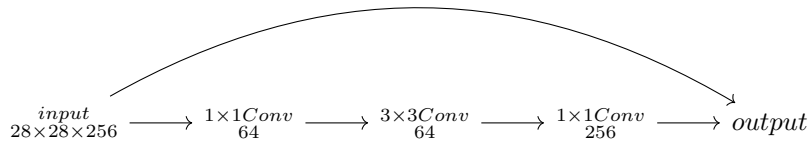

Figure 5.4: Bottleneck