



In-depth understanding of the GoogLeNet structure (original)



Zhang Lei

Computer Vision, Deep Learning, Neural Networks

Follow him

Agree415



share

415 people agreed with this article

Inception (also known as GoogLeNet) is a new deep learning structure proposed by Christian Szegedy in 2014. Before that, structures such as AlexNet and VGG achieved better training results by increasing the depth (number of layers) of the network, but the increase in the number of layers will bring many negative effects, such as overfit, gradient disappearance, gradient explosion, etc. Inception improves training results from another perspective: it can use computing resources more efficiently and extract more features with the same amount of computing, thereby improving training results.

original link : ...

There are a lot of articles about Inception on the Internet, most of which are plagiarized from each other. The few original articles are limited to describing the structure, or just mechanically translating the paper, without telling you how the structure came about. After reading them, you are still confused. This article focuses on the principles behind Inception, and some of them are my own understanding. Please correct me if I have any misunderstandings.

1. Core Idea

The basic structure of the inception module is shown in the figure below. The entire inception structure is composed of multiple such inception modules connected in series. The main contributions of the inception structure are twofold: one is the use of 1x1 convolution to increase and decrease dimensions; the other is the simultaneous convolution and aggregation of multiple dimensions.

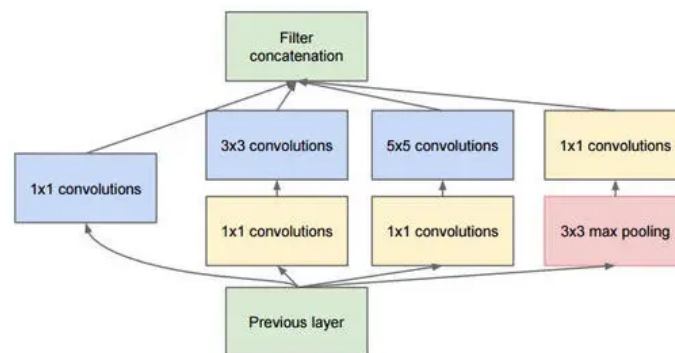


Figure 1: Inception module

1. 1x1 convolution

You can see that there are multiple yellow 1x1 convolution modules in Figure 1. What is the use of such convolution?

Function 1: Stacking more convolutions in the same size of receptive field can extract richer features. This view comes from Network in Network (NIN) arxiv.org/pdf/1312.4400 \ the three

Agree415



32 Comments



share



like



collect



Apply for reprint





Figure 2: Comparison of linear convolution and NIN structures

The left side of Figure 2 shows the traditional convolutional layer structure (linear convolution), which has only one convolution at one scale; the right side shows the Network in Network structure (NIN structure), which first performs a normal convolution (such as 3x3) and then a 1x1 convolution. For a certain pixel, a 1x1 convolution is equivalent to a fully connected calculation of all features of the pixel, so the 1x1 convolution on the right side is drawn in the form of a fully connected layer. It should be noted that in the NIN structure, whether it is the first 3x3 convolution or the newly added 1x1 convolution, it is followed by an activation function (such as relu). By connecting two convolutions in series, more nonlinear features can be combined. For example, assuming that the first 3x3 convolution + activation function is approximately $f_1(x)=ax^2+bx+c$, and the second 1x1 convolution + activation function is approximately $f_2(x)=mx^2+nx+q$, which one is more nonlinear and can better simulate nonlinear features, $f_1(x)$ or $f_2(f_1(x))$? The answer is obvious. The structure of NIN is somewhat similar to the multi-layer structure of traditional neural networks. The latter's multi-layers span receptive fields of different sizes (by adding pool layers between layers), thereby extracting features at a higher scale; the NIN structure is a multi-layer at the same scale (without a pool layer in the middle), so that stronger nonlinearity can be extracted within the same receptive field range.

Function 2: Using 1x1 convolution to reduce dimension reduces computational complexity. The 3x3 convolution in the middle of Figure 2 and the 1x1 convolution before the 5x5 convolution both play this role. When the number of features of a convolution layer input is large, performing convolution operations on this input will generate a huge amount of computation; if the input is first reduced in dimension, the number of features is reduced before performing convolution operations, the amount of computation will be significantly reduced. The following figure compares the number of multiplications of the two schemes before and after optimization. The same is true for a set of data with 192 features, 32x32 size, and 256 sets of features. The first picture is directly implemented with 3x3 convolution, which requires $192 \times 256 \times 3 \times 3 \times 32 \times 32 = 452984832$ multiplications; the second picture first uses 1x1 convolution to reduce to 96 features, and then uses 3x3 convolution to restore 256 sets of features, which requires $192 \times 96 \times 1 \times 1 \times 32 \times 32 + 96 \times 256 \times 3 \times 3 \times 32 \times 32 = 245366784$ multiplications. The 1x1 convolution dimensionality reduction method saves half of the computational effort. Some people may ask, after using 1x1 convolution to reduce to 96 features, the number of features is reduced, will it affect the final training effect? The answer is no, as long as the number of features output at the end remains unchanged (256 groups), the intermediate dimensionality reduction is similar to the effect of compression and does not affect the final training results.



Figure 3: Adding 1x1 convolution reduces the amount of computation

2 Convolution and aggregation on multiple scales

What are the benefits of this new structure? Szegedy explained it from multiple perspectives:

Explanation 1: Intuitively, performing convolution at multiple scales simultaneously can extract features at different scales. Richer features also mean more accurate final classification. ▲

Explanation 2: Use the principle of decomposing sparse matrices into dense matrix calculations to speed up convergence. For example, the left side of the figure below is a sparse matrix (many elements are 0, unevenly distributed in the matrix), and convolving it with a 2x2 matrix requires calculating every element in the sparse matrix; if the sparse matrix is decomposed into two sub-dense matrices as shown on the right, and then convolved with the 2x2 matrix, the area with more 0s in the sparse matrix does not need to be calculated, and the amount of calculation is greatly reduced. **This principle is applied to inception, which is to decompose in the feature dimension!** The input data of the traditional convolution layer is convolved with a convolution kernel of only one scale (such as 3x3), and the output data is fixed dimension (such as 256 features). All 256 output features are basically evenly distributed in the 3x3 scale range, which can be understood as outputting a sparsely distributed feature set; while the inception module extracts features at multiple scales (such as 1x1, 3x3, 5x5), and the 256 output features are no longer evenly distributed, but the features with strong correlation are clustered together (such as 96 features of 1x1, 96 features of 3x3, and 64 features of 5x5), which can be understood as multiple densely distributed sub-feature sets. In such a feature set, because the features with strong correlation are clustered together, the irrelevant and non-critical features are weakened. Although the same 256 features are output, the features output by the inception method have less "redundant" information. Using such a "pure" feature set to pass layer by layer and finally as the input of the reverse calculation will naturally converge faster.

$$\begin{bmatrix} 2 & 3 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 4 & 1 \\ 3 & 2 \end{bmatrix} \Leftrightarrow \begin{bmatrix} 2 & 3 \\ 0 & 2 \end{bmatrix} \otimes \begin{bmatrix} 4 & 1 \\ 3 & 2 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 0 \\ 2 & 0 & 3 \\ 0 & 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 4 & 1 \\ 3 & 2 \end{bmatrix}$$

Figure 4: Decomposing a sparse matrix into sub-dense matrices for computation

Explanation 3: Hebbin's principle. The Hebbin's principle is a theory in neuroscience that explains the changes that occur in neurons in the brain during the learning process. It can be summarized in one sentence: *fire together, wire together*. Hebbin believes that "if two neurons or neuron systems are always excited at the same time, they will form a 'combination', and the excitement of one neuron will promote the excitement of the other." For example, a dog will drool when it sees meat. After repeated stimulation, the neurons in the brain that recognize meat will promote each other and "entangle" with the neurons that control saliva secretion. When seeing meat in the future, it will drool faster. Used in the inception structure is to gather together features with strong correlation. This is a bit similar to the above explanation 2, separating the 1x1, 3x3, and 5x5 features. Because the ultimate goal of training convergence is to extract independent features, gathering highly correlated features in advance can accelerate convergence.

There is a branch in the inception module that uses max pooling. The author believes that pooling can also play a role in extracting features, so it is also added to the module. Note that the stride of this pooling is 1, and the size of the data is not reduced after pooling.

mimicking biological systems, this would also have the advantage of firmer theoretical underpinnings due to the ground-breaking work of Arora et al. [2]. Their main result states that if the probability distribution of the data-set is representable by a large, very sparse deep neural network, then the optimal network topology can be constructed layer by layer by analyzing the correlation statistics of the activations of the last layer and clustering neurons with highly correlated outputs.

The author proposes that the fully connected structure needs to be transformed into a sparsely connected structure. There are two methods of sparse connection. One is spatial sparse connection, which is the traditional CNN convolution structure: only a certain patch of the input image is convolved instead of the entire image. Shared parameters reduce the number of total parameters and the amount of calculation; the other method is sparse connection in the feature dimension, which is the convolution and aggregation at multiple sizes mentioned in the previous section. Features with strong correlation are grouped together. Each size of convolution only outputs a part of the 256 features. This is also a sparse connection. The author mentioned that the theoretical basis of this method comes from the paper Provable bounds for learning some deep representations by Arora et al. (The mathematical requirements of this paper by Arora et al are too high, and I didn't understand it).

Page3: On the downside, today's computing infrastructures are very inefficient when it comes to numerical calculation on non-uniform sparse data structures. Even if the number of arithmetic operations is reduced by 100x, the overhead of lookups and cache misses is so dominant that switching to sparse matrices would not pay off. The gap is widened even further by the use of steadily improving, highly tuned, numerical libraries that allow for extremely fast dense matrix multiplication, exploiting the minute details of the underlying CPU or GPU hardware [16, 9]. Also, non-uniform sparse models require more sophisticated engineering and computing infrastructure. Most current vision oriented machine learning systems utilize sparsity in the spatial domain just by the virtue of employing convolutions. However, convolutions are implemented as collections of dense connections to the patches in the earlier layer. ConvNets have traditionally used random and sparse connection tables in the feature dimensions since [11] in order to break the symmetry and improve learning, the trend changed back to full connections with [9] in order to better optimize parallel computing. The uniformity of the structure and a large number of filters and greater batch size allow for utilizing efficient dense computation.

The author mentioned that today's computers are very inefficient in calculating sparse data, and even using sparse matrix algorithms is not worth the effort (see the calculation method described in Figure 4. Note that the sparse matrix on the left side of Figure 4 is stored in the form of element value + row and column value inside the computer, and only non-zero elements are stored). Although using sparse matrix algorithms to calculate will greatly reduce the amount of calculation, it will increase the intermediate cache (please study the calculation method of sparse matrices for the specific reasons).

The most common way to exploit data sparsity today is to calculate local patches through convolution (CNN method, which is the aforementioned method of exploiting sparsity in spatial). Another way to exploit data sparsity is to exploit it in the feature dimension, such as the ConvNets structure, which uses a feature connection table to determine which convolution outputs are accumulated together (the ordinary structure uses a convolution kernel to convolve all input features, then accumulates all the results together and outputs a feature; while ConvNets selectively accumulates certain convolution results). ConvNets' method of exploiting sparsity is rarely used now, because only full connection in the feature dimension can more efficiently utilize the parallel computing capabilities of the GPU, otherwise you have to design a

theory, but exploits our current hardware by utilizing computations on dense matrices. The vast literature on sparse matrix computations (eg [3]) suggests that clustering sparse matrices into relatively dense submatrices tends to give state of the art practical performance for sparse matrix multiplication.

As mentioned earlier, methods like ConvNets that exploit sparsity are rarely used nowadays. So, are there any other methods that can exploit sparsity in the feature dimension? This leads to the focus of this paper: bringing together highly correlated features, which is the convolution and aggregation at multiple scales mentioned in the previous chapter.

Page 6: The use of average pooling before the classifier is based on [12], although our implementation differs in that we use an extra linear layer.

Network in Network (NIN, [arxiv.org/pdf/1312.4400 ...](https://arxiv.org/pdf/1312.4400)) first proposed the method of using the Global Average Pooling (GAP) layer to replace the fully connected layer. The specific method is to average all the points on each feature. If there are n features, n average values are output as the final softmax input. Its advantages are: 1. Regularize the data on the entire feature to prevent overfitting; 2. No longer need a fully connected layer, reducing the number of parameters of the entire structure (generally the fully connected layer is the layer with the most parameters in the entire structure), and the possibility of overfitting is reduced; 3. No need to pay attention to the size of the input image, because no matter what the input is, the averaging method is the same. The traditional fully connected layer must choose the number of parameters according to the size, which is not universal.

Page 6: By adding auxiliary classifiers connected to these intermediate layers, we would expect to encourage discrimination in the lower stages in the classifier, increase the gradient signal that gets propagated back, and provide additional regularization.

The inception structure adds branch classifiers at certain levels, and the output loss is multiplied by a coefficient and added to the total loss. The author believes that this can prevent the gradient vanishing problem (in fact, this treatment has little effect at lower levels, and the author clarified this in the subsequent inception v3 paper).

Posted on 2018-01-06 20:27

[Computer Vision](#) [Neural Networks](#) [Deep Learning](#)



Speak rationally and interact in a friendly manner

32 Comments

default up to
date



HarukiCat

It's so helpful. It's hard to describe the original text.

2018-03-25

reply 11



Bridge over the River

It's a good explanation. The network structure is easy to understand, but few people explain why it is designed this way. Thank you very much.



2020-01-01

reply 4



Sunny day every day

I followed two blogs before I came to the original article. It is so well written! It is like a sudden enlightenment, a sudden realization, and a wake-up call...

2021-02-06

reply 2



Flying fish

This answered my question about the sparseness of the original text. I couldn't understand this part of the original text after reading it several times.

2019-06-18

reply 2



Allenash

Hello, the original text of 1x1 convolution has two parts: while ensuring that the number of final output feature maps remains unchanged, if the dimensionality reduction in the middle part can reduce the amount of calculation without affecting the final training effect, then why not simply reduce the number of feature maps generated in the middle from 96 to 1?

2018-01-09

reply 1



Weixiao

Compression algorithms are also divided into lossy compression and lossless compression.

2020-02-19

reply 6



Go to bed early and get up early

Is Figure 4 just an example? I don't understand how the calculations on the left and right sides are equal.😓

2021-11-19

reply like



Wudaokou Xiao Bajie

That sentence "The inception module extracts features at multiple scales (such as 1x1, 3x3, 5x5), and the output 256 features are no longer evenly distributed, but features with strong correlation are clustered together (for example, 96 features of 1x1 are clustered together, 96 features of 3x3 are clustered together, and 64 features of 5x5 are clustered together)"~~~~ Please ask, why does "uneven distribution" become "features with strong correlation are clustered together"?




2020-03-12

reply like

features are different. Previously, there were 256 3x3 kernels, and now there are 96 3x3 kernels. The dimension of the feature channel extracted at the same scale is reduced. Can it be said that a smaller number of feature channels gathers features with stronger correlation? Similarly, the more feature channels there are, the more uniform the distribution of the features is, that is, only a small part of the features are allocated to specific channels. On the basis of keeping the total number of output feature channels unchanged, reducing the channel dimension of the features at each scale is equivalent to making the features more compact. Finally, because the features of other scales are catted, the feature representation ability is stronger under the same number of channels. Therefore, at the same scale, simply increasing the number of filter kernels to make the network wider will increase the redundancy of the features obtained.

2022-04-13

reply 3


 **you guess** ...

Because the feature maps output by convolutional layers of different sizes are concatenated in the channel dimension

2020-10-07

reply 1


Expand 1 more reply >

 **Meet the whales** ...

What does "256 output features are basically evenly distributed in the 3x3 scale range" mean?🤔

2020-03-01


reply like

 **The deer of the immortal** ...


I think that because we used 256 different 3x3 convolution kernels to perform convolution, the features learned by different convolution kernels must be different, so the learned features are evenly distributed (depending on the design of the convolution kernel). In addition, my personal understanding is that since we are talking about gathering strongly related features together, and Inception only allocates 96 3x3 convolution kernels, it means that the previous 256 3x3 convolution kernels are quite redundant, that is, the features they learned are important and unimportant, and we only need to learn the most important ones. As for how the hyperparameter 96 comes from, I don't know.

2020-08-16

reply 1

 **Image encryption** ▶ **Meet the whales** ...

For the case where 256 3x3s are redundant, my understanding is as follows: because the 256 3x3s extract features of 3x3 scale, resulting in 3x3 scale features being evenly distributed on 256 feature channels, the increase in the number of channels inevitably leads to redundant features. In the case where the total number of output feature channels remains unchanged, reducing the 256 kernels of 3x3 to 96 is equivalent to enhancing the cohesion of the features. In addition, other kernels can be allocated to extract features of other scales. Therefore, the reduction in the number of channels of different scales means that redundancy is reduced and

 Speak rationally and interact in a friendly manner

recommended reading



Google, so rich...

Wharton senior Annie



Google, so rich...



[SOSP'03] The Google File System

The latest Google customer development methods in...

Outline of this article: ♦ Using Google my business to develop customers ♦ Using Google Maps to develop customers ♦ Using Google+ to develop customers (including actual practice of...

知乎

