目 录

写在前面

Life, thin and light-off time and time again
Frivolous tireless
生命，一次又一次轻薄过
轻狂不知疲倦

ImageNet Classification with Deep Convolutional Neural Networks

0.1 Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry

0.2 Introduction

Current approaches to object recognition make essential use of machine learning methods. To improve their performance, we can collect larger datasets, learn more powerful models, and use better techniques for preventing overfitting. Until recently, datasets of labeled images were relatively small — on the order of tens of thousands of images (e.g., NORB [16], Caltech-101/256 [8, 9], and CIFAR-10/100 [12]). Simple recognition tasks can be solved quite well with datasets of this size, especially if they are augmented with label-preserving transformations. For example, the currentbest error rate on the MNIST digit-recognition task (<0.3%) approaches human performance [4]. But objects in realistic settings exhibit considerable variability, so to learn to recognize them it is necessary to use much larger training sets. And indeed, the shortcomings of small image datasets have been widely recognized (e.g., Pinto et al. [21]), but it has only recently become possible to collect labeled datasets with millions of images. The new larger datasets include LabelMe [23], which consists of hundreds of thousands of fully-segmented images, and ImageNet [6], which consists of over 15 million labeled high-resolution images in over 22,000 categories.

To learn about thousands of objects from millions of images, we need a model with a large learning capacity. However, the immense complexity of the object recognition task means that this problem cannot be specified even by a dataset as large as ImageNet, so our model should also have lots of prior knowledge to compensate for all the data we don't have. Convolutional neural networks (CNNs) constitute one such class of models [16, 11, 13, 18,

15, 22, 26]. Their capacity can be controlled by varying their depth and breadth, and they also make strong and mostly correct assumptions about the nature of images (namely, stationarity of statistics and locality of pixel dependencies). Thus, compared to standard feedforward neural networks with similarly-sized layers, CNNs have much fewer connections and parameters and so they are easier to train, while their theoretically-best performance is likely to be only slightly worse.

Despite the attractive qualities of CNNs, and despite the relative efficiency of their local architecture, they have still been prohibitively expensive to apply in large scale to high-resolution images. Luckily, current GPUs, paired with a highly-optimized implementation of 2D convolution, are powerful enough to facilitate the training of interestingly-large CNNs, and recent datasets such as ImageNet contain enough labeled examples to train such models without severe overfitting.

The specific contributions of this paper are as follows: we trained one of the largest convolutional neural networks to date on the subsets of ImageNet used in the ILSVRC-2010 and ILSVRC-2012 competitions [2] and achieved by far the best results ever reported on these datasets. We wrote a highly-optimized GPU implementation of 2D convolution and all the other operations inherent in training convolutional neural networks, which we make available publicly. Our network contains a number of new and unusual features which improve its performance and reduce its training time, which are detailed in Section 3. The size of our network made overfitting a significant problem, even with 1.2 million labeled training examples, so we used several effective techniques for preventing overfitting, which are described in Section 4. Our final network contains five convolutional and three fully-connected layers, and this depth seems to be important: we found that removing any convolutional layer (each of which contains no more than 1% of the model's parameters) resulted in inferior performance.

In the end, the network's size is limited mainly by the amount of memory available on current GPUs and by the amount of training time that we are willing to tolerate. Our network takes between five and six days to train on two GTX 580 3GB GPUs. All of our experiments suggest that our results can be improved simply by waiting for faster GPUs and bigger datasets to become available.

0.3 The Dataset

ImageNet is a dataset of over 15 million labeled high-resolution images belonging to roughly 22,000 categories. The images were collected from the web and labeled by human labelers using Amazon's Mechanical Turk crowd-sourcing tool. Starting in 2010, as part of the Pascal Visual Object Challenge, an annual competition called the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) has been held. ILSVRC uses a subset of ImageNet

with roughly 1000 images in each of 1000 categories. In all, there are roughly 1.2 million training images, 50,000 validation images, and 150,000 testing images.

ILSVRC-2010 is the only version of ILSVRC for which the test set labels are available, so this is the version on which we performed most of our experiments. Since we also entered our model in the ILSVRC-2012 competition, in Section 6 we report our results on this version of the dataset as well, for which test set labels are unavailable. On ImageNet, it is customary to report two error rates: top-1 and top-5, where the top-5 error rate is the fraction of test images for which the correct label is not among the five labels considered most probable by the model.

ImageNet consists of variable-resolution images, while our system requires a constant input dimensionality. Therefore, we down-sampled the images to a fixed resolution of 256 × 256. Given a rectangular image, we first rescaled the image such that the shorter side was of length 256, and then cropped out the central 256×256 patch from the resulting image. We did not pre-process the images in any other way, except for subtracting the mean activity over the training set from each pixel. So we trained our network on the (centered) raw RGB values of the pixels.

0.4 The Architecture

The architecture of our network is summarized in Figure 2. It contains eight learned layers — five convolutional and three fully-connected. Below, we describe some of the novel or unusual features of our network's architecture. Sections 3.1-3.4 are sorted according to our estimation of their importance, with the most important first.

0.4.1 ReLU Nonlinearity

tandard way to model a neuron's output f as a function of its input x is with $f(x) = tanh(x)$ or $f(x) = (1 + e^x)^1$. In terms of training time with gradient descent, these saturating nonlinearities are much slower than the non-saturating nonlinearity $f(x) = max(0, x)$. Following Nair and Hinton [20], we refer to neurons with this nonlinearity as Rectified Linear Units (ReLUs). Deep convolutional neural networks with ReLUs train several times faster than their equivalents with tanh units. This is demonstrated in Figure 1, which shows the number of iterations required to reach 25% training error on the CIFAR-10 dataset for a particular four-layer convolutional network. This plot shows that we would not have been able to experiment with such large neural networks for this work if we had used traditional saturating neuron models.

We are not the first to consider alternatives to traditional neuron models in CNNs. For example, Jarrett et al. [11] claim that the nonlinearity $f(x) = |tanh(x)|$ works particularly well

with their type of contrast normalization followed by local average pooling on the Caltech-101 dataset. However, on this dataset the primary concern is preventing overfitting, so the effect they are observing is different from the accelerated ability to fit the training set which we report when using ReLUs. Faster learning has a great influence on the performance of large models trained on large datasets.

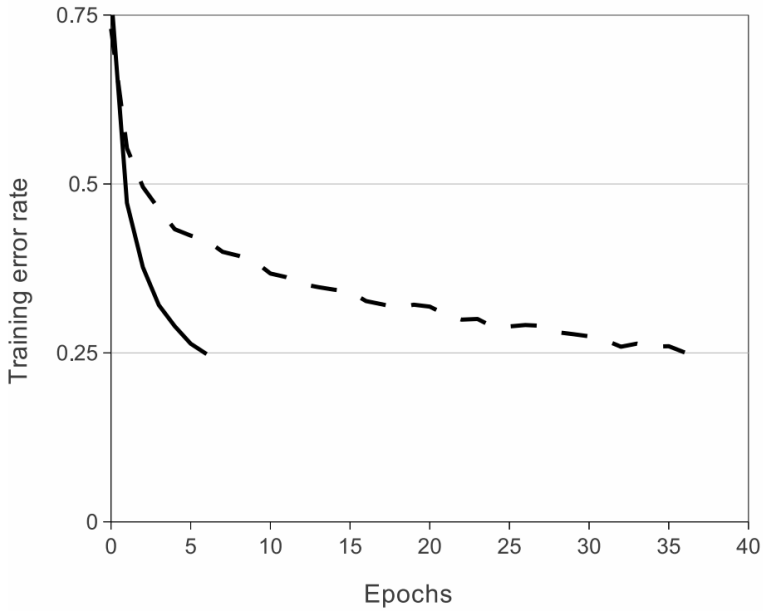


Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons **(dashed line)**. The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.

0.4.2 Training on Multiple GPUs

A single GTX 580 GPU has only 3GB of memory, which limits the maximum size of the networks that can be trained on it. It turns out that 1.2 million training examples are enough to train networks which are too big to fit on one GPU. Therefore we spread the net across two GPUs. Current GPUs are particularly well-suited to cross-GPU parallelization, as they are able to read from and write to one another's memory directly, without going through host machine memory. The parallelization scheme that we employ essentially puts half of the kernels (or neurons) on each GPU, with one additional trick: the GPUs communicate only in certain layers. This means that, for example, the kernels of layer 3 take input from all kernel maps in layer 2. However, kernels in layer 4 take input only from those kernel maps

in layer 3 which reside on the same GPU. Choosing the pattern of connectivity is a problem for cross-validation, but this allows us to precisely tune the amount of communication until it is an acceptable fraction of the amount of computation.

The resultant architecture is somewhat similar to that of the "columnar" CNN employed by Cire¸san et al. [5], except that our columns are not independent (see Figure 2). This scheme reduces our top-1 and top-5 error rates by 1.7% and 1.2%, respectively, as compared with a net with half as many kernels in each convolutional layer trained on one GPU. The two-GPU net takes slightly less time to train than the one-GPU net^2.

0.4.3 Local Response Normalization

ReLUs have the desirable property that they do not require input normalization to prevent them from saturating. If at least some training examples produce a positive input to a ReLU, learning will happen in that neuron. However, we still find that the following local normalization scheme aids generalization. Denoting by $a^i_{x,y}$ the activity of a neuron computed by applying kernel i at position (x, y) and then applying the ReLU nonlinearity, the response-normalized activity $b^i_{x,y}$ is given by the expression

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

where the sum runs over n "adjacent" kernel maps at the same spatial position, and N is the total number of kernels in the layer. The ordering of the kernel maps is of course arbitrary and determined before training begins. This sort of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels. The constants k, n, α, and β are hyper-parameters whose values are determined using a validation set; we used $k = 2$, $n = 5$, $\alpha = 104$, and $\beta = 0.75$. We applied this normalization after applying the ReLU nonlinearity in certain layers (see Section 3.5).

This scheme bears some resemblance to the local contrast normalization scheme of Jarrett et al. [11], but ours would be more correctly termed "brightness normalization", since we do not subtract the mean activity. Response normalization reduces our top-1 and top-5 error rates by 1.4% and 1.2%, respectively. We also verified the effectiveness of this scheme on the CIFAR-10 dataset: a four-layer CNN achieved a 13% test error rate without normalization and 11% with $normalization^3$.

0.4.4 Overlapping Pooling

Pooling layers in CNNs summarize the outputs of neighboring groups of neurons in the same kernel map. Traditionally, the neighborhoods summarized by adjacent pooling units do not overlap (e.g., [17, 11, 4]). To be more precise, a pooling layer can be thought of as consisting of a grid of pooling units spaced s pixels apart, each summarizing a neighborhood of size $z \times z$ centered at the location of the pooling unit. If we set $s = z$, we obtain traditional local pooling as commonly employed in CNNs. If we set $s < z$, we obtain overlapping pooling. This is what we use throughout our network, with $s = 2$ and $z = 3$. This scheme reduces the top-1 and top-5 error rates by 0.4% and 0.3%, respectively, as compared with the non-overlapping scheme $s = 2$, $z = 2$, which produces output of equivalent dimensions. We generally observe during training that models with overlapping pooling find it slightly more difficult to overfit.

0.4.5 Overall Architecture

Now we are ready to describe the overall architecture of our CNN. As depicted in Figure 2, the net contains eight layers with weights; the first five are convolutional and the remaining three are fully-connected. The output of the last fully-connected layer is fed to a 1000-way softmax which produces a distribution over the 1000 class labels. Our network maximizes the multinomial logistic regression objective, which is equivalent to maximizing the average across training cases of the log-probability of the correct label under the prediction distribution.

The kernels of the second, fourth, and fifth convolutional layers are connected only to those kernel maps in the previous layer which reside on the same GPU (see Figure 2). The kernels of the third convolutional layer are connected to all kernel maps in the second layer. The neurons in the fully connected layers are connected to all neurons in the previous layer. Response-normalization layers follow the first and second convolutional layers. Max-pooling layers, of the kind described in Section 3.4, follow both response-normalization layers as well as the fifth convolutional layer. The ReLU non-linearity is applied to the output of every convolutional and fully-connected layer.

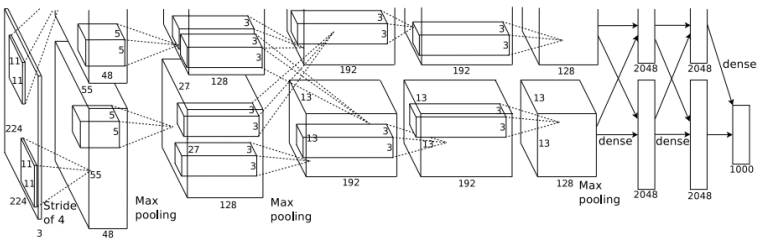


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delin-

eation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

The first convolutional layer filters the 224×224×3 input image with 96 kernels of size 11×11×3 with a stride of 4 pixels (this is the distance between the receptive field centers of neighboring neurons in a kernel map). The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with 256 kernels of size $5 \times 5 \times 48$. The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3 \times 256$ connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 192$, and the fifth convolutional layer has 256 kernels of size $3 \times 3 \times 192$. The fully-connected layers have 4096 neurons each.

0.5 Reducing Overfitting

Our neural network architecture has 60 million parameters. Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting. Below, we describe the two primary ways in which we combat overfitting.

0.5.1 Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random 224×224 patches (and their horizontal reflections) from the 256×256 images and training our network on these extracted $patches^4$. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly inter dependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At

test time, the network makes a prediction by extracting five 224 × 224 patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network's softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components, with magnitudes proportional to the corresponding eigenvalues times a random variable drawn from a Gaussian with mean zero and standard deviation 0.1. Therefore to each RGB image pixel $I_{xy} = \begin{bmatrix} I_{xy}^R & I_{xy}^G & I_{xy}^B \end{bmatrix}^T$ we add the following quantity:

$$\begin{bmatrix} P_1 & P_2 & P_3 \end{bmatrix} \begin{bmatrix} \alpha_1\lambda_1 & \alpha_2\lambda_2 & \alpha_3\lambda_3 \end{bmatrix}^T$$

where P_i and λ_i are ith eigenvector and eigenvalue of the 3 × 3 covariance matrix of RGB pixel values, respectively, and α_i is the aforementioned random variable. Each α_i is drawn only once for all the pixels of a particular training image until that image is used for training again, at which point it is re-drawn. This scheme approximately captures an important property of natural images, namely, that object identity is invariant to changes in the intensity and color of the illumination. This scheme reduces the top-1 error rate by over 1%.

0.5.2 Dropout

Combining the predictions of many different models is a very successful way to reduce test errors [1, 3], but it appears to be too expensive for big neural networks that already take several days to train. There is, however, a very efficient version of model combination that only costs about a factor of two during training. The recently-introduced technique, called "dropout" [10], consists of setting to zero the output of each hidden neuron with probability 0.5. The neurons which are "dropped out" in this way do not contribute to the forward pass and do not participate in back-propagation. So every time an input is presented, the neural network samples a different architecture, but all these architectures share weights. This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons. At test time, we use all the neurons but multiply their outputs by 0.5, which is a reasonable approximation to taking the geometric mean of the predictive distributions produced by the exponentially-many dropout networks.

We use dropout in the first two fully-connected layers of Figure 2. Without dropout, our network exhibits substantial overfitting. Dropout roughly doubles the number of iterations

required to converge.

0.6 Details of learning

We trained our models using stochastic gradient descent with a batch size of 128 examples, momentum of 0.9, and weight decay of 0.0005. We found that this small amount of weight decay was important for the model to learn. In other words, weight decay here is not merely a regularizer: it reduces the model's training error. The update rule for weight w was

$$v_{i+1} := 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w} \mid_{w_i} \right\rangle_{D_i}$$

$$w_{i+1} := w_i + v_{i+1}$$

where i is the iteration index, v is the momentum variable, ϵ is the learning rate, and $\left\langle \frac{\partial L}{\partial w} \mid_{w_i} \right\rangle_{D_i}$ is the average over the ith batch D_i of the derivative of the objective with respect to w, evaluated at w_i.

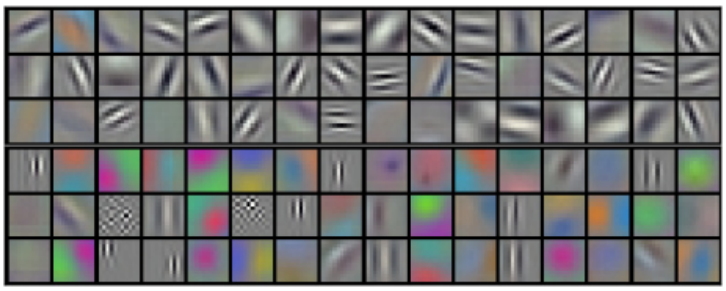


Figure 3: 96 convolutional kernels of size 11×11×3 learned by the first convolutional layer on the 224×224×3 input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2. See Section 6.1 for details.

We initialized the weights in each layer from a zero-mean Gaussian distribution with standard deviation 0.01. We initialized the neuron biases in the second, fourth, and fifth convolutional layers, as well as in the fully-connected hidden layers, with the constant 1. This initialization accelerates the early stages of learning by providing the ReLUs with positive inputs. We initialized the neuron biases in the remaining layers with the constant 0.

We used an equal learning rate for all layers, which we adjusted manually throughout training. The heuristic which we followed was to divide the learning rate by 10 when the validation error rate stopped improving with the current learning rate. The learning rate was initialized at 0.01 and reduced three times prior to termination. We trained the network for roughly 90 cycles through the training set of 1.2 million images, which took five to six days on two NVIDIA GTX 580 3GB GPUs.

0.7 Results

Our results on ILSVRC-2010 are summarized in Table 1. Our network achieves top-1 and top-5 test set error rates of 37.5% and 17.0. The best performance achieved during the ILSVRC-2010 competition was 47.1% and 28.2% with an approach that averages the predictions produced from six sparse-coding models trained on different features [2], and since then the best published results are 45.7% and 25.7% with an approach that averages the predictions of two classifiers trained on Fisher Vectors (FVs) computed from two types of densely-sampled features [24].

Model	**Top-1**	**Top-5**
Sparse coding [2]	47.1%	28.2%
SIFT + FVs [24]	45.7%	25.7%
CNN	**37.5**	**17.0%**

Table 1: Comparison of results on ILSVRC-2010 test set. In italics are best results achieved by others.

We also entered our model in the ILSVRC-2012 competition and report our results in Table 2. Since the ILSVRC-2012 test set labels are not publicly available, we cannot report test error rates for all the models that we tried. In the remainder of this paragraph, we use validation and test error rates interchangeably because in our experience they do not differ by more than 0.1% (see Table 2). The CNN described in this paper achieves a top-5 error rate of 18.2%. Averaging the predictions of five similar CNNs gives an error rate of 16.4%. Training one CNN, with an extra sixth convolutional layer over the last pooling layer, to classify the entire ImageNet Fall 2011 release (15M images, 22K categories), and then "fine-tuning" it on ILSVRC-2012 gives an error rate of 16.6%. Averaging the predictions of two CNNs that were pre-trained on the entire Fall 2011 release with the aforementioned five CNNs gives an error rate of 15.3%. The second-best contest entry achieved an error rate of 26.2% with an approach that averages the predictions of several classifiers trained on FVs computed from different types of densely-sampled features [7].

Model	**Top-1(val)**	**Top-5(val)**	**Top-5(test)**
SIFT + FVs [7]	-	-	26.2%
1 CNN	40.7%	18.2%	-
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	-

Model	Top-1(val)	Top-5(val)	Top-5(test)
7 CNNs*	36.7%	15.4%	**15.3%**

Table 2: Comparison of error rates on ILSVRC-2012 validation and test sets. In italics are best results achieved by others. Models with an asterisk* were "pre-trained" to classify the entire ImageNet 2011 Fall release. See Section 6 for details.

Finally, we also report our error rates on the Fall 2009 version of ImageNet with 10,184 categories and 8.9 million images. On this dataset we follow the convention in the literature of using half of the images for training and half for testing. Since there is no established test set, our split necessarily differs from the splits used by previous authors, but this does not affect the results appreciably. Our top-1 and top-5 error rates on this dataset are 67.4% and 40.9%, attained by the net described above but with an additional, sixth convolutional layer over the last pooling layer. The best published results on this dataset are 78.1% and 60.9% [19].

0.7.1 Qualitative Evaluations

Figure 3 shows the convolutional kernels learned by the network's two data-connected layers. The network has learned a variety of frequency- and orientation-selective kernels, as well as various colored blobs. Notice the specialization exhibited by the two GPUs, a result of the restricted connectivity described in Section 3.5. The kernels on GPU 1 are largely color-agnostic, while the kernels on on GPU 2 are largely color-specific. This kind of specialization occurs during every run and is independent of any particular random weight initialization (modulo a renumbering of the GPUs).

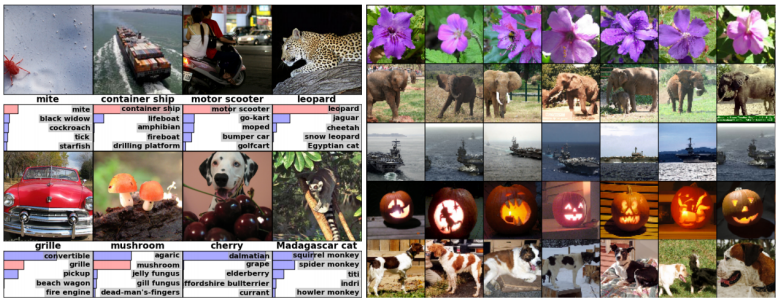


Figure 4: (Left) Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). (Right) Five ILSVRC-2010 test images in the first column. The remaining columns show

the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

In the left panel of Figure 4 we qualitatively assess what the network has learned by computing its top-5 predictions on eight test images. Notice that even off-center objects, such as the mite in the top-left, can be recognized by the net. Most of the top-5 labels appear reasonable. For example, only other types of cat are considered plausible labels for the leopard. In some cases (grille, cherry) there is genuine ambiguity about the intended focus of the photograph.

Another way to probe the network's visual knowledge is to consider the feature activations induced by an image at the last, 4096-dimensional hidden layer. If two images produce feature activation vectors with a small Euclidean separation, we can say that the higher levels of the neural network consider them to be similar. Figure 4 shows five images from the test set and the six images from the training set that are most similar to each of them according to this measure. Notice that at the pixel level, the retrieved training images are generally not close in L2 to the query images in the first column. For example, the retrieved dogs and elephants appear in a variety of poses. We present the results for many more test images in the supplementary material.

Computing similarity by using Euclidean distance between two 4096-dimensional, real-valued vectors is inefficient, but it could be made efficient by training an auto-encoder to compress these vectors to short binary codes. This should produce a much better image retrieval method than applying auto-encoders to the raw pixels [14], which does not make use of image labels and hence has a tendency to retrieve images with similar patterns of edges, whether or not they are semantically similar

0.8 Discussion

Our results show that a large, deep convolutional neural network is capable of achieving record-breaking results on a highly challenging dataset using purely supervised learning. It is notable that our network's performance degrades if a single convolutional layer is removed. For example, removing any of the middle layers results in a loss of about 2% for the top-1 performance of the network. So the depth really is important for achieving our results.

To simplify our experiments, we did not use any unsupervised pre-training even though we expect that it will help, especially if we obtain enough computational power to significantly increase the size of the network without obtaining a corresponding increase in the amount of labeled data. Thus far, our results have improved as we have made our network larger and trained it longer but we still have many orders of magnitude to go in order to match the infero-temporal pathway of the human visual system. Ultimately we would like to

use very large and deep convolutional nets on video sequences where the temporal structure provides very helpful information that is missing or far less obvious in static images.

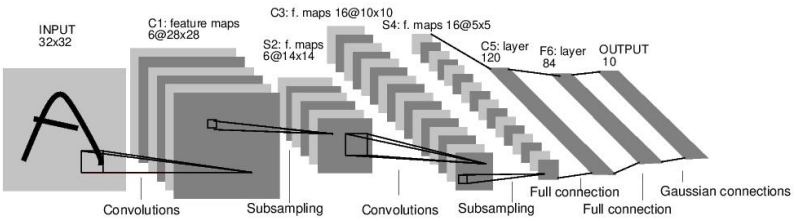
经典深度学习网络模型

0.9 LeNet-5

0.9.1 模型介绍

LeNet-5 是由 $LeCun$ 提出的一种用于识别手写数字和机器印刷字符的卷积神经网络（Convolutional Neural Network，CNN）[1]，其命名来源于作者 $LeCun$ 的名字，5 则是其研究成果的代号，在 LeNet-5 之前还有 LeNet-4 和 LeNet-1 鲜为人知。LeNet-5 阐述了图像中像素特征之间的相关性能够由参数共享的卷积操作所提取，同时使用卷积、下采样（池化）和非线性映射这样的组合结构，是当前流行的大多数深度图像识别网络的基础。

0.9.2 模型结构



LeNet-5 一共包含 7 层（输入层不作为网络结构），分别由 2 个卷积层、2 个下采样层和 3 个全连接层组成，网络的参数配置如下表所示，其中下采样层和全连接层的核尺寸分别代表采样范围和连接矩阵的尺寸（如卷积核尺寸中的 $5 \times 5 \times 1/1,6$ 表示核大小为 $5 \times 5 \times 1$、步长为 1 且核个数为 6 的卷积核）。

LeNet-5 网络参数配置:

网络层	输入尺寸	核尺寸	输出尺寸	可训练参数量
卷积层 C_1	$32 \times 32 \times 1$	$5 \times 5 \times 1/1,6$	$28 \times 28 \times 6$	$(5 \times 5 \times 1 + 1) \times 6$
下采样层 S_2	$28 \times 28 \times 6$	$2 \times 2/2$	$14 \times 14 \times 6$	$(1 + 1) \times 6$ *
卷积层 C_3	$14 \times 14 \times 6$	$5 \times 5 \times 6/1,16$	$10 \times 10 \times 16$	1516*
下采样层 S_4	$10 \times 10 \times 16$	$2 \times 2/2$	$5 \times 5 \times 16$	$(1 + 1) \times 16$
卷积层 C_5*	$5 \times 5 \times 16$	$5 \times 5 \times 16/1,120$	$1 \times 1 \times 120$	$(5 \times 5 \times 16 + 1) \times 120$
全连接层 F_6	$1 \times 1 \times 120$	120×84	$1 \times 1 \times 84$	$(120 + 1) \times 84$

网络层	输入尺寸	核尺寸	输出尺寸	可训练参数量
输出层	$1 \times 1 \times 84$	84×10	$1 \times 1 \times 10$	$(84+1) \times 10$

* 在 LeNet 中，下采样操作和池化操作类似，但是在得到采样结果后会乘以一个系数和加上一个偏置项，所以下采样的参数个数是 $(1+1) \times 6$ 而不是零。

* C_3 卷积层可训练参数并未直接连接 S_2 中所有的特征图（Feature Map），而是采用如下图所示的采样特征方式进行连接（稀疏连接），生成的 16 个通道特征图中分别按照相邻 3 个特征图、相邻 4 个特征图、非相邻 4 个特征图和全部 6 个特征图进行映射，得到的参数个数计算公式为 $6\times(25\times3+1)+6\times(25\times4+1)+3\times(25\times4+1)+1\times(25\times6+1) = 1516$，在原论文中解释了使用这种采样方式原因包含两点：限制了连接数不至于过大（当年的计算能力比较弱）；强制限定不同特征图的组合可以使映射得到的特征图学习到不同的特征模式。

S_2 与 C_3 之间的特征图稀疏连接

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

* C_5 卷积层显示为全连接层，原论文中解释这里实际采用的是卷积操作，只是刚好在 5×5 卷积后尺寸被压缩为 1×1，输出结果看起来和全连接很相似。

0.9.3 模型特性

- 卷积网络使用一个 3 层的序列组合：卷积、下采样（池化）、非线性映射（LeNet-5 最重要的特性，奠定了目前深层卷积网络的基础）
- 使用卷积提取空间特征
- 使用映射的空间均值进行下采样
- 使用 $tanh$ 或 $sigmoid$ 进行非线性映射
- 多层神经网络（MLP）作为最终的分类器
- 层间的稀疏连接矩阵以避免巨大的计算开销

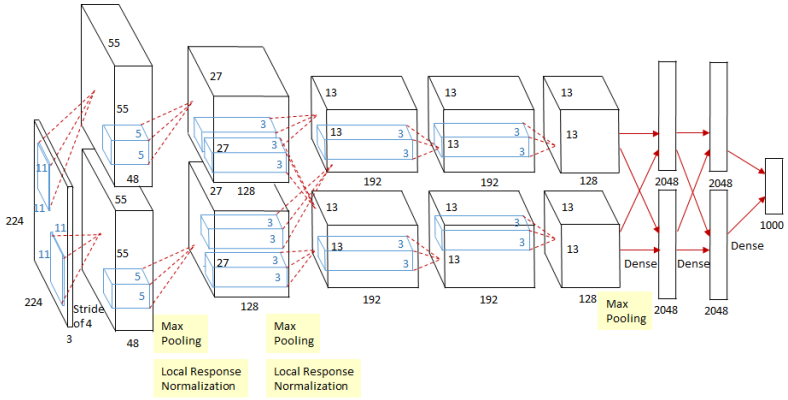
0.10 AlexNet

0.10.1 模型介绍

AlexNet 是由 _Alex Krizhevsky_ 提出的首个应用于图像分类的深层卷积神经网络，该网络在 2012 年 ILSVRC（ImageNet Large Scale Visual Recognition Competition）图像

分类竞赛中以 15.3% 的 top-5 测试错误率赢得第一名。AlexNet 使用 GPU 代替 CPU 进行运算，使得在可接受的时间范围内模型结构能够更加复杂，它的出现证明了深层卷积神经网络在复杂模型下的有效性，使 CNN 在计算机视觉中流行开来，直接或间接地引发了深度学习的热潮。

0.10.2 模型结构



除去下采样（池化层）和局部响应规范化操作（Local Responsible Normalization, LRN），AlexNet 一共包含 8 层，前 5 层由卷积层组成，而剩下的 3 层为全连接层。网络结构分为上下两层，分别对应两个 GPU 的操作过程，除了中间某些层（C_3 卷积层和 F_{6-8} 全连接层会有 GPU 间的交互），其他层两个 GPU 分别计算结果。最后一层全连接层的输出作为 $softmax$ 的输入，得到 1000 个图像分类标签对应的概率值。除去 GPU 并行结构的设计，AlexNet 网络结构与 LeNet 十分相似，其网络的参数配置如表所示。

网络层	输入尺寸	核尺寸	输出尺寸	可训练参数量
卷积层 C_1 *	$224 \times 224 \times 3$	$11 \times 11 \times 3/4, 48(\times 2_{GPU})$	$55 \times 55 \times 48(\times 2_{GPU})$	$(11 \times 11 \times 3 + 1) \times 48 \times 2$
下采样层 S_{max}*	$55 \times 55 \times 48(\times 2_{GPU})$	$3 \times 3/2(\times 2_{GPU})$	$27 \times 27 \times 48(\times 2_{GPU})$	0
卷积层 C_2	$27 \times 27 \times 48(\times 2_{GPU})$	$5 \times 5 \times 48/1, 128(\times 2_{GPU})$	$27 \times 27 \times 128(\times 2_{GPU})$	$(5 \times 5 \times 48 + 1) \times 128 \times 2$
下采样层 S_{max}	$27 \times 27 \times 128(\times 2_{GPU})$	$3 \times 3/2(\times 2_{GPU})$	$13 \times 13 \times 128(\times 2_{GPU})$	0
卷积层 C_3 *	$13 \times 13 \times 128 \times 2_{GPU}$	$3 \times 3 \times 256/1, 192(\times 2_{GPU})$	$13 \times 13 \times 192(\times 2_{GPU})$	$(3 \times 3 \times 256 + 1) \times 192 \times 2$
卷积层 C_4	$13 \times 13 \times 192(\times 2_{GPU})$	$3 \times 3 \times 192/1, 192(\times 2_{GPU})$	$13 \times 13 \times 192(\times 2_{GPU})$	$(3 \times 3 \times 192 + 1) \times 192 \times 2$

网络层	输入尺寸	核尺寸	输出尺寸	可训练参数量
卷积层 C_5	$13 \times 13 \times 192(\times 2_{GPU})$	$3 \times 3 \times 192/1,128(\times 2_{GPU})$	$13 \times 13 \times 128(\times 2_{GPU})$	$(3 \times 3 \times 192 + 1) \times 128 \times 2$
下采样层 S_{max}	$13 \times 13 \times 128(\times 2_{GPU})$	$3 \times 3/2(\times 2_{GPU})$	$6 \times 6 \times 128(\times 2_{GPU})$	0
全连接层 F_6 *	$6 \times 6 \times 128 \times 2_{GPU}$	$9216 \times 2048(\times 2_{GPU})$	$1 \times 1 \times 2048(\times 2_{GPU})$	$(9216 + 1) \times 2048 \times 2$
全连接层 F_7	$1 \times 1 \times 2048 \times 2_{GPU}$	$4096 \times 2048(\times 2_{GPU})$	$1 \times 1 \times 2048(\times 2_{GPU})$	$(4096 + 1) \times 2048 \times 2$
全连接层 F_8	$1 \times 1 \times 2048 \times 2_{GPU}$	4096×1000	$1 \times 1 \times 1000$	$(4096 + 1) \times 1000 \times 2$

- 卷积层 C_1 输入为 $224 \times 224 \times 3$ 的图片数据，分别在两个 GPU 中经过核为 $11 \times 11 \times 3$、步长（stride）为 4 的卷积卷积后，分别得到两条独立的 $55 \times 55 \times 48$ 的输出数据。

- 下采样层 S_{max} 实际上是嵌套在卷积中的最大池化操作，但是为了区分没有采用最大池化的卷积层单独列出来。在 C_{1-2} 卷积层中的池化操作之后（ReLU 激活操作之前），还有一个 LRN 操作，用作对相邻特征点的归一化处理。

- 卷积层 C_3 的输入与其他卷积层不同，$13 \times 13 \times 192 \times 2_{GPU}$ 表示汇聚了上一层网络在两个 GPU 上的输出结果作为输入，所以在进行卷积操作时通道上的卷积核维度为 384。

- 全连接层 F_{6-8} 中输入数据尺寸也和 C_3 类似，都是融合了两个 GPU 流向的输出结果作为输入。

0.10.3 模型特性

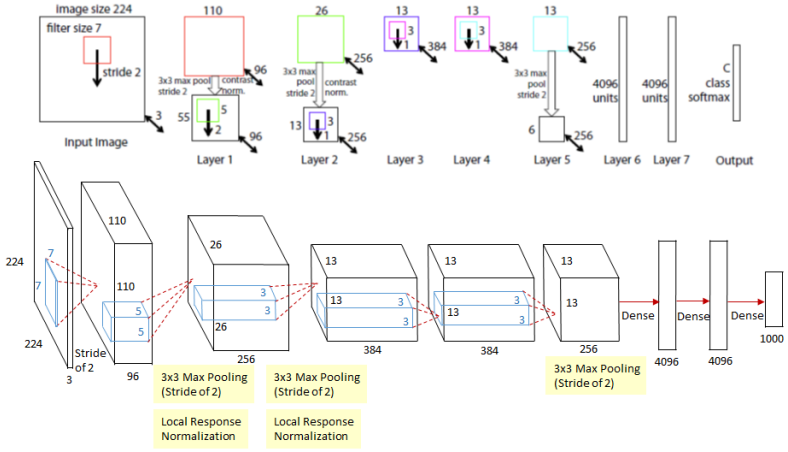
- 所有卷积层都使用 ReLU 作为非线性映射函数，使模型收敛速度更快
- 在多个 GPU 上进行模型的训练，不但可以提高模型的训练速度，还能提升数据的使用规模
- 使用 LRN 对局部的特征进行归一化，结果作为 ReLU 激活函数的输入能有效降低错误率
- 重叠最大池化（overlapping max pooling），即池化范围 z 与步长 s 存在关系 $z > s$（如 S_{max} 中核尺度为 $3 \times 3/2$），避免平均池化（average pooling）的平均效应
- 使用随机丢弃技术（dropout）选择性地忽略训练中的单个神经元，避免模型的过拟合

0.11 ZFNet

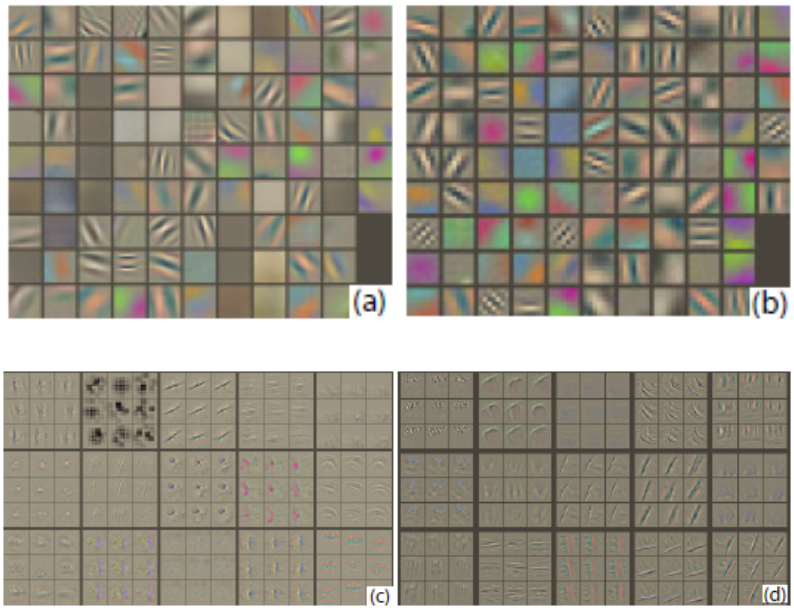
0.11.1 模型介绍

ZFNet 是由 $Matthew\ D.Zeiler$ 和 $Rob\ Fergus$ 在 AlexNet 基础上提出的大型卷积网络，在 2013 年 ILSVRC 图像分类竞赛中以 11.19% 的错误率获得冠军（实际上原 ZFNet 所在的队伍并不是真正的冠军，原 ZFNet 以 13.51% 错误率排在第 8，真正的冠军是 $Clarifai$ 这个队伍，而 $Clarifai$ 这个队伍所对应的一家初创公司的 CEO 又是 $Zeiler$，而且 $Clarifai$ 对 ZFNet 的改动比较小，所以通常认为是 ZFNet 获得了冠军）[3−4]。ZFNet 实际上是微调（fine-tuning）了的 AlexNet，并通过反卷积（Deconvolution）的方式可视化各层的输出特征图，进一步解释了卷积操作在大型网络中效果显著的原因。

0.11.2 模型结构



如图所示，ZFNet 与 AlexNet 类似，都是由 8 层网络组成的卷积神经网络，其中包含 5 层卷积层和 3 层全连接层。两个网络结构最大的不同在于，ZFNet 第一层卷积采用了 $7 \times 7 \times 3/2$ 的卷积核替代了 AlexNet 中第一层卷积核 $11 \times 11 \times 3/4$ 的卷积核。ZFNet 相比于 AlexNet 在第一层输出的特征图中包含更多中间频率的信息，而 AlexNet 第一层输出的特征图大多是低频或高频的信息，对中间频率特征的缺失导致后续网络层次能够学习到的特征不够细致，而导致这个问题的根本原因在于 AlexNet 在第一层中采用的卷积核和步长过大。



（a）ZFNet 第一层输出的特征图（b）AlexNet 第一层输出的特征图（c）AlexNet 第二层输出的特征图（d）ZFNet 第二层输出的特征图

网络层	输入尺寸	核尺寸	输出尺寸	可训练参数量
卷积层 C_1 *	$224 \times 224 \times 3$	$7 \times 7 \times 3/2, 96$	$110 \times 110 \times 96$	$(7 \times 7 \times 3 + 1) \times 96$
下采样层 S_{max}	$110 \times 110 \times 96$	$3 \times 3/2$	$55 \times 55 \times 96$	0
卷积层 C_2 *	$55 \times 55 \times 96$	$5 \times 5 \times 96/2, 256$	$26 \times 26 \times 256$	$(5 \times 5 \times 96 + 1) \times 256$
下采样层 S_{max}	$26 \times 26 \times 256$	$3 \times 3/2$	$13 \times 13 \times 256$	0
卷积层 C_3	$13 \times 13 \times 256$	$3 \times 3 \times 256/1, 384$	$13 \times 13 \times 384$	$(3 \times 3 \times 256 + 1) \times 384$
卷积层 C_4	$13 \times 13 \times 384$	$3 \times 3 \times 384/1, 384$	$13 \times 13 \times 384$	$(3 \times 3 \times 384 + 1) \times 384$
卷积层 C_5	$13 \times 13 \times 384$	$3 \times 3 \times 384/1, 256$	$13 \times 13 \times 256$	$(3 \times 3 \times 384 + 1) \times 256$
下采样层 S_{max}	$13 \times 13 \times 256$	$3 \times 3/2$	$6 \times 6 \times 256$	0
全连接层 F_6	$6 \times 6 \times 256$	9216×4096	$1 \times 1 \times 4096$	$(9216 + 1) \times 4096$

网络层	输入尺寸	核尺寸	输出尺寸	可训练参数量
全连接层 F_7	$1 \times 1 \times 4096$	4096×4096	$1 \times 1 \times 4096$	$(4096+1) \times 4096$
全连接层 F_8	$1 \times 1 \times 4096$	4096×1000	$1 \times 1 \times 1000$	$(4096+1) \times 1000$

- 卷积层 C_1 与 AlexNet 中的 C_1 有所不同，采用 $7 \times 7 \times 3/2$ 的卷积核代替 $11 \times 11 \times 3/4$，使第一层卷积输出的结果可以包含更多的中频率特征，对后续网络层中多样化的特征组合提供更多选择，有利于捕捉更细致的特征。

- 卷积层 C_2 采用了步长 2 的卷积核，区别于 AlexNet 中 C_2 的卷积核步长，所以输出的维度有所差异。

0.11.3 模型特性

ZFNet 与 AlexNet 在结构上几乎相同，此部分虽属于模型特性，但准确地说应该是 ZFNet 原论文中可视化技术的贡献。

- 可视化技术揭露了激发模型中每层单独的特征图。
- 可视化技术允许观察在训练阶段特征的演变过程且诊断出模型的潜在问题。
- 可视化技术用到了多层解卷积网络，即由特征激活返回到输入像素空间。
- 可视化技术进行了分类器输出的敏感性分析，即通过阻止部分输入图像来揭示那部分对于分类是重要的。
- 可视化技术提供了一个非参数的不变性来展示来自训练集的哪一块激活哪个特征图，不仅需要裁剪输入图片，而且自上而下的投影来揭露来自每块的结构激活一个特征图。
- 可视化技术依赖于解卷积操作，即卷积操作的逆过程，将特征映射到像素上。

0.12 Network in Network

0.12.1 模型介绍

Network In Network (NIN) 是由 $MinLin$ 等人提出，在 CIFAR-10 和 CIFAR-100 分类任务中达到当时的最好水平，因其网络结构是由三个多层感知机堆叠而被成为 NIN[5]。NIN 以一种全新的角度审视了卷积神经网络中的卷积核设计，通过引入子网络结构代替纯卷积中的线性映射部分，这种形式的网络结构激发了更复杂的卷积神经网络的结构设计，其中下一节中介绍的 GoogLeNet 的 Inception 结构就是来源于这个思想。

0.12.2 模型结构

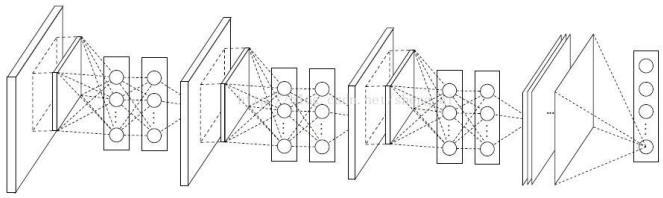


Figure 2: The overall structure of Network In Network. In this paper the NINs include the stacking of three mlpconv layers and one global average pooling layer.

NIN 由三层的多层感知卷积层（MLPConv Layer）构成，每一层多层感知卷积层内部由若干层的局部全连接层和非线性激活函数组成，代替了传统卷积层中采用的线性卷积核。在网络推理（inference）时，这个多层感知器会对输入特征图的局部特征进行划窗计算，并且每个划窗的局部特征图对应的乘积的权重是共享的，这两点是和传统卷积操作完全一致的，最大的不同在于多层感知器对局部特征进行了非线性的映射，而传统卷积的方式是线性的。NIN 的网络参数配置表 4.4 所示（原论文并未给出网络参数，表中参数为编者结合网络结构图和 CIFAR-100 数据集以 3×3 卷积为例给出）。

网络层	输入尺寸	核尺寸	输出尺寸	参数个数
局部全连接层 L_{11} *	$32 \times 32 \times 3$	$(3 \times 3) \times 16/1$	$30 \times 30 \times 16$	$(3 \times 3 \times 3 + 1) \times 16$
全连接层 L_{12} *	$30 \times 30 \times 16$	16×16	$30 \times 30 \times 16$	$((16 + 1) \times 16)$
局部全连接层 L_{21}	$30 \times 30 \times 16$	$(3 \times 3) \times 64/1$	$28 \times 28 \times 64$	$(3 \times 3 \times 16 + 1) \times 64$
全连接层 L_{22}	$28 \times 28 \times 64$	64×64	$28 \times 28 \times 64$	$((64 + 1) \times 64)$
局部全连接层 L_{31}	$28 \times 28 \times 64$	$(3 \times 3) \times 100/1$	$26 \times 26 \times 100$	$(3 \times 3 \times 64 + 1) \times 100$
全连接层 L_{32}	$26 \times 26 \times 100$	100×100	$26 \times 26 \times 100$	$((100 + 1) \times 100)$
全局平均采样 GAP *	$26 \times 26 \times 100$	$26 \times 26 \times 100/1$	$1 \times 1 \times 100$	0

- 局部全连接层 L_{11} 实际上是对原始输入图像进行划窗式的全连接操作，因此划窗得到的输出特征尺寸为 30×30（$\frac{32-3_k+1}{1_{stride}} = 30$）
- 全连接层 L_{12} 是紧跟 L_{11} 后的全连接操作，输入的特征是划窗后经过激活的局部响应特征，因此仅需连接 L_{11} 和 L_{12} 的节点即可，而每个局部全连接层和紧接的

全连接层构成代替卷积操作的多层感知卷积层（MLPConv）。

- 全局平均采样层或全局平均池化层 GAP（Global Average Pooling）将 L_{32} 输出的每一个特征图进行全局的平均池化操作，直接得到最后的类别数，可以有效地减少参数量。

0.12.3 模型特点

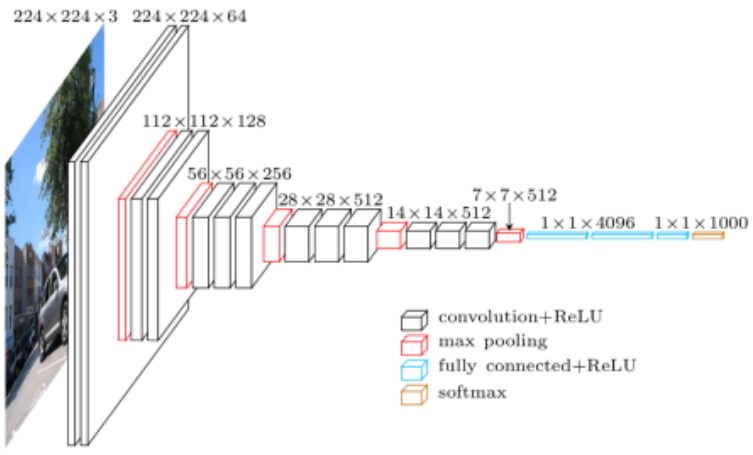
- 使用多层感知机结构来代替卷积的滤波操作，不但有效减少卷积核数过多而导致的参数量暴涨问题，还能通过引入非线性的映射来提高模型对特征的抽象能力。
- 使用全局平均池化来代替最后一个全连接层，能够有效地减少参数量（没有可训练参数），同时池化用到了整个特征图的信息，对空间信息的转换更加鲁棒，最后得到的输出结果可直接作为对应类别的置信度。

0.13 VGGNet

0.13.1 模型介绍

VGGNet 是由牛津大学视觉几何小组（Visual Geometry Group, VGG）提出的一种深层卷积网络结构，他们以 7.32% 的错误率赢得了 2014 年 ILSVRC 分类任务的亚军（冠军由 GoogLeNet 以 6.65% 的错误率夺得）和 25.32% 的错误率夺得定位任务（Localization）的第一名（GoogLeNet 错误率为 26.44%）[5]，网络名称 VGGNet 取自该小组名缩写。VGGNet 是首批把图像分类的错误率降低到 10% 以内模型，同时该网络所采用的 3×3 卷积核的思想是后来许多模型的基础，该模型发表在 2015 年国际学习表征会议（International Conference On Learning Representations, ICLR）后至今被引用的次数已经超过 1 万 4 千余次。

0.13.2 模型结构



在原论文中的 VGGNet 包含了 6 个版本的演进，分别对应 VGG11、VGG11-LRN、

VGG13、VGG16-1、VGG16-3 和 VGG19，不同的后缀数值表示不同的网络层数（VGG11-LRN 表示在第一层中采用了 LRN 的 VGG11，VGG16-1 表示后三组卷积块中最后一层卷积采用卷积核尺寸为 1×1，相应的 VGG16-3 表示卷积核尺寸为 3×3），本节介绍的 VGG16 为 VGG16-3。上图中的 VGG16 体现了 VGGNet 的核心思路，使用 3×3 的卷积组合代替大尺寸的卷积（2 个 3×3 5×5 卷积拥有相同的感受视野），网络参数设置如下表所示。

网络层	输入尺寸	核尺寸	输出尺寸	参数个数
卷积层 C_{11}	$224 \times 224 \times 3$	$3 \times 3 \times 64/1$	$224 \times 224 \times 64$	$(3 \times 3 \times 3 + 1) \times 64$
卷积层 C_{12}	$224 \times 224 \times 64$	$3 \times 3 \times 64/1$	$224 \times 224 \times 64$	$(3 \times 3 \times 64 + 1) \times 64$
下采样层 S_{max1}	$224 \times 224 \times 64$	$2 \times 2/2$	$112 \times 112 \times 64$	0
卷积层 C_{21}	$112 \times 112 \times 64$	$3 \times 3 \times 128/1$	$112 \times 112 \times 128$	$(3 \times 3 \times 64 + 1) \times 128$
卷积层 C_{22}	$112 \times 112 \times 128$	$3 \times 3 \times 128/1$	$112 \times 112 \times 128$	$(3 \times 3 \times 128 + 1) \times 128$
下采样层 S_{max2}	$112 \times 112 \times 128$	$2 \times 2/2$	$56 \times 56 \times 128$	0
卷积层 C_{31}	$56 \times 56 \times 128$	$3 \times 3 \times 256/1$	$56 \times 56 \times 256$	$(3 \times 3 \times 128 + 1) \times 256$
卷积层 C_{32}	$56 \times 56 \times 256$	$3 \times 3 \times 256/1$	$56 \times 56 \times 256$	$(3 \times 3 \times 256 + 1) \times 256$
卷积层 C_{33}	$56 \times 56 \times 256$	$26 \times 26 \times 256/1$	$56 \times 56 \times 256$	$(3 \times 3 \times 256 + 1) \times 256$
下采样层 S_{max3}	$56 \times 56 \times 256$	$2 \times 2/2$	$28 \times 28 \times 256$	0
卷积层 C_{41}	$28 \times 28 \times 256$	$3 \times 3 \times 512/1$	$28 \times 28 \times 512$	$(3 \times 3 \times 256 + 1) \times 512$
卷积层 C_{42}	$28 \times 28 \times 512$	$3 \times 3 \times 512/1$	$28 \times 28 \times 512$	$(3 \times 3 \times 512 + 1) \times 512$
卷积层 C_{43}	$28 \times 28 \times 512$	$3 \times 3 \times 512/1$	$28 \times 28 \times 512$	$(3 \times 3 \times 512 + 1) \times 512$
下采样层 S_{max4}	$28 \times 28 \times 512$	$2 \times 2/2$	$14 \times 14 \times 512$	0

网络层	输入尺寸	核尺寸	输出尺寸	参数个数
卷积层 C_{51}	$14 \times 14 \times 512$	$3 \times 3 \times 512/1$	$14 \times 14 \times 512$	$(3 \times 3 \times 512 + 1) \times 512$
卷积层 C_{52}	$14 \times 14 \times 512$	$3 \times 3 \times 512/1$	$14 \times 14 \times 512$	$(3 \times 3 \times 512 + 1) \times 512$
卷积层 C_{53}	$14 \times 14 \times 512$	$3 \times 3 \times 512/1$	$14 \times 14 \times 512$	$(3 \times 3 \times 512 + 1) \times 512$
下采样层 S_{max5}	$14 \times 14 \times 512$	$2 \times 2/2$	$7 \times 7 \times 512$	0
全连接层 FC_1	$7 \times 7 \times 512$	$(7 \times 7 \times 512) \times 4096$	1×4096	$(7 \times 7 \times 512 + 1) \times 4096$
全连接层 FC_2	1×4096	4096×4096	1×4096	$(4096 + 1) \times 4096$
全连接层 FC_3	1×4096	4096×1000	1×1000	$(4096 + 1) \times 1000$

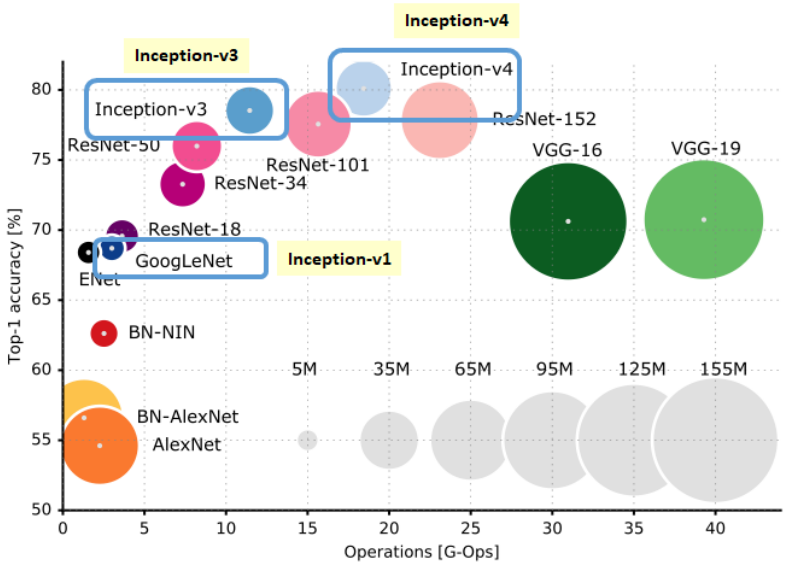
0.13.3 模型特性

- 整个网络都使用了同样大小的卷积核尺寸 3×3 和最大池化尺寸 2×2。
- 1×1 卷积的意义主要在于线性变换，而输入通道数和输出通道数不变，没有发生降维。
- 两个 3×3 的卷积层串联相当于 1 个 5×5 的卷积层，感受野大小为 5×5。同样地，3 个 3×3 的卷积层串联的效果则相当于 1 个 7×7 的卷积层。这样的连接方式使得网络参数量更小，而且多层的激活函数令网络对特征的学习能力更强。
- VGGNet 在训练时有一个小技巧，先训练浅层的的简单网络 VGG11，再复用 VGG11 的权重来初始化 VGG13，如此反复训练并初始化 VGG19，能够使训练时收敛的速度更快。
- 在训练过程中使用多尺度的变换对原始数据做数据增强，使得模型不易过拟合。

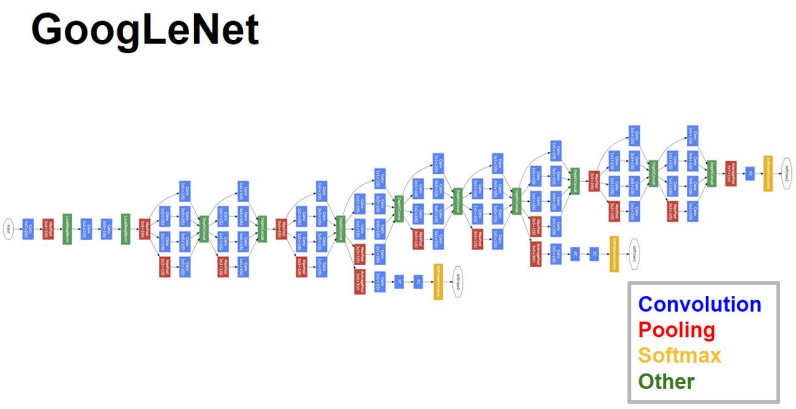
0.14 GoogLeNet

0.14.1 模型介绍

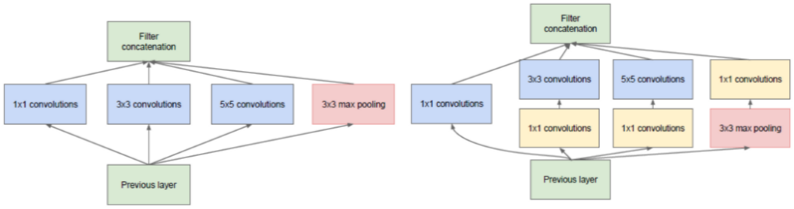
GoogLeNet 作为 2014 年 ILSVRC 在分类任务上的冠军，以 6.65% 的错误率力压 VGGNet 等模型，在分类的准确率上面相比过去两届冠军 ZFNet 和 AlexNet 都有很大的提升。从名字 **GoogLe**Net 可以知道这是来自谷歌工程师所设计的网络结构，而名字中 Goog**LeNet** 更是致敬了 LeNet[0]。GoogLeNet 中最核心的部分是其内部子网络结构 Inception，该结构灵感来源于 NIN，至今已经经历了四次版本迭代（Inception$_{v1-4}$）。

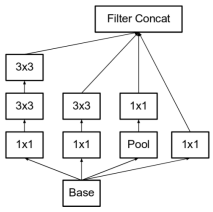


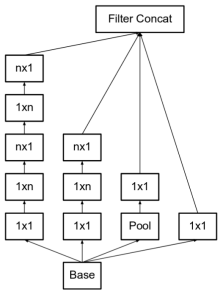
0.14.2 4.6.2 模型结构

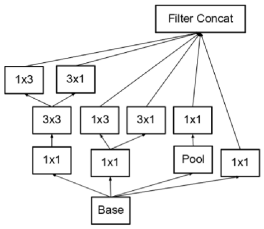


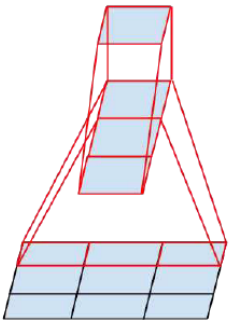
GoogLeNet 相比于以前的卷积神经网络结构，除了在深度上进行了延伸，还对网络的宽度进行了扩展，整个网络由许多块状子网络的堆叠而成，这个子网络构成了 Inception 结构。上图 Inception 的四个版本：$Inception_{v1}$ 在同一层中采用不同的卷积核，并对卷积结果进行合并;$Inception_{v2}$ 组合不同卷积核的堆叠形式，并对卷积结果进行合并;$Inception_{v3}$ 则在 v_2 基础上进行深度组合的尝试;$Inception_{v4}$ 结构相比于前面的版本更加复杂，子网络中嵌套着子网络。

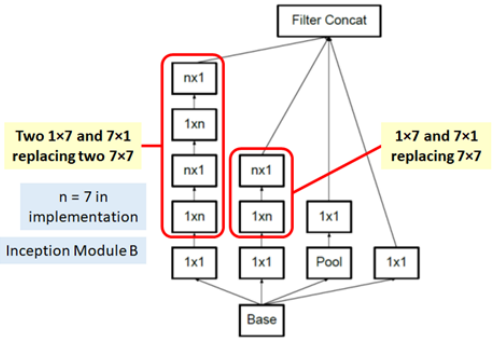


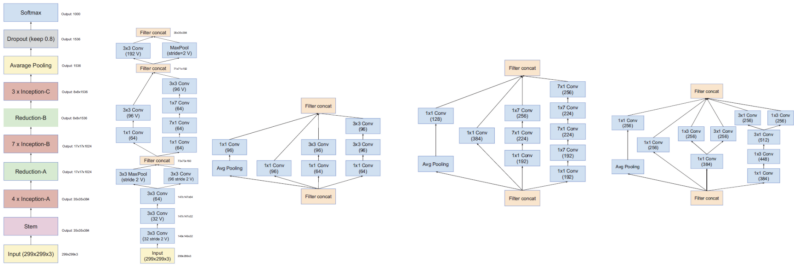












网络层	输入尺寸	核尺寸	输出尺寸	参数个数
卷积层 C_{11}	$H \times W \times C_1$	$1 \times 1 \times C_2/2$	$\frac{H}{2} \times \frac{W}{2} \times C_2$	$(1 \times 1 \times C_1 + 1) \times C_2$
卷积层 C_{21}	$H \times W \times C_2$	$1 \times 1 \times C_2/2$	$\frac{H}{2} \times \frac{W}{2} \times C_2$	$(1 \times 1 \times C_2 + 1) \times C_2$
卷积层 C_{22}	$H \times W \times C_2$	$3 \times 3 \times C_2/1$	$H \times W \times C_2/1$	$(3 \times 3 \times C_2 + 1) \times C_2$
卷积层 C_{31}	$H \times W \times C_1$	$1 \times 1 \times C_2/2$	$\frac{H}{2} \times \frac{W}{2} \times C_2$	$(1 \times 1 \times C_1 + 1) \times C_2$
卷积层 C_{32}	$H \times W \times C_2$	$5 \times 5 \times C_2/1$	$H \times W \times C_2/1$	$(5 \times 5 \times C_2 + 1) \times C_2$
下采样层 S_{41}	$H \times W \times C_1$	$3 \times 3/2$	$\frac{H}{2} \times \frac{W}{2} \times C_2$	0
卷积层 C_{42}	$\frac{H}{2} \times \frac{W}{2} \times C_2$	$1 \times 1 \times C_2/1$	$\frac{H}{2} \times \frac{W}{2} \times C_2$	$(3 \times 3 \times C_2 + 1) \times C_2$

网络层	输入尺寸	核尺寸	输出尺寸	参数个数
合并层 M	$\frac{H}{2} \times \frac{W}{2} \times C_2(\times 4)$	拼接	$\frac{H}{2} \times \frac{W}{2} \times (C_2 \times 4)$	0

0.14.3 模型特性

- 采用不同大小的卷积核意味着不同大小的感受野，最后拼接意味着不同尺度特征的融合；
- 之所以卷积核大小采用 1、3 和 5，主要是为了方便对齐。设定卷积步长 stride=1 之后，只要分别设定 pad=0、1、2，那么卷积之后便可以得到相同维度的特征，然后这些特征就可以直接拼接在一起了；
- 网络越到后面，特征越抽象，而且每个特征所涉及的感受野也更大了，因此随着层数的增加，3x3 和 5x5 卷积的比例也要增加。但是，使用 5x5 的卷积核仍然会带来巨大的计算量。为此，文章借鉴 NIN2，采用 1x1 卷积核来进行降维。

0.15 为什么现在的 CNN 模型都是在 GoogleNet、VGGNet 或者 AlexNet 上调整的？

- 评测对比：为了让自己的结果更有说服力，在发表自己成果的时候会同一个标准的 baseline 及在 baseline 上改进而进行比较，常见的比如各种检测分割的问题都会基于 VGG 或者 Resnet101 这样的基础网络。
- 时间和精力有限：在科研压力和工作压力中，时间和精力只允许大家在有限的范围探索。
- 模型创新难度大：进行基本模型的改进需要大量的实验和尝试，并且需要大量的实验积累和强大灵感，很有可能投入产出比比较小。
- 资源限制：创造一个新的模型需要大量的时间和计算资源，往往在学校和小型商业团队不可行。
- 在实际的应用场景中，其实是有大量的非标准模型的配置。