

---

# ImageNet Classification with Deep Convolutional Neural Networks

---

**Alex Krizhevsky**  
University of Toronto  
kriz@cs.utoronto.ca

**Ilya Sutskever**  
University of Toronto  
ilya@cs.utoronto.ca

**Geoffrey E. Hinton**  
University of Toronto  
hinton@cs.utoronto.ca

## Abstract

## 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.

딥러닝에 GPU를 통한 연산을 사용  
Dropout 기법 사용

## Introduction

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.

많은 image를 학습하기 위해서는 모델의 capacity가 커야함

### CNN

- depth, breadth를 조절
- 더 적은 connection과 parameter를 가지고 있음 → 학습 하기 쉬움

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.

많은 연산량을 가진 CNN → GPU로 해결

## 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.

ImageNet에서 주는 이미지는 22000개 카테고리에 포함되는 15개의 이미지  
ILSVRC대회에서는 1000개의 카테고리에 포함되는 이미지들로 학습

## The Architecture

### - ReLU Nonlinearity

The standard 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.**

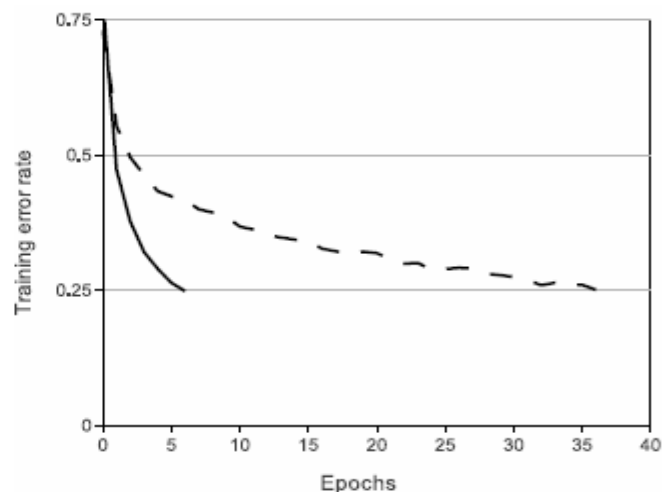


Figure 1: A four-layer convolutional neural network with ReLUs (solid line) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (dashed line). The learning rates for each network were chosen independently to make training as fast as possible. No 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.

ReLU가 tanh보다 activation function으로 사용했을 때 더 빠르게 학습됨  
빠른 학습은 많은 데이터 셋과 큰 모델에서 많은 영향을 줌



## The Architecture

### - 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.

GPU 2개 병렬  
Kernel 절반으로 나누어 계산  
특정 layer에서는 GPU끼리 communicate함

## The Architecture

### - Local Response Normalization

ReLU's 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  $a_{x,y}^i$  the activity of a neuron computed by applying kernel  $i$  at position  $(x, y)$  and then applying the ReLU nonlinearity, the response-normalized activity  $b_{x,y}^i$  is given by the expression

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

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  $k, n, \alpha$ , and  $\beta$  are hyper-parameters whose values are determined using a validation set; we used  $k = 2$ ,  $n = 5$ ,  $\alpha = 10^{-4}$ , 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<sup>3</sup>.

ReLU에 입력 값에 비례하여 출력 값이 증가함 (높은 하나의 픽셀 값이 주변 값에 영향을 미침)  
normalization을 해주어 성능 개선

## The Architecture

### - Overlapping Pooling

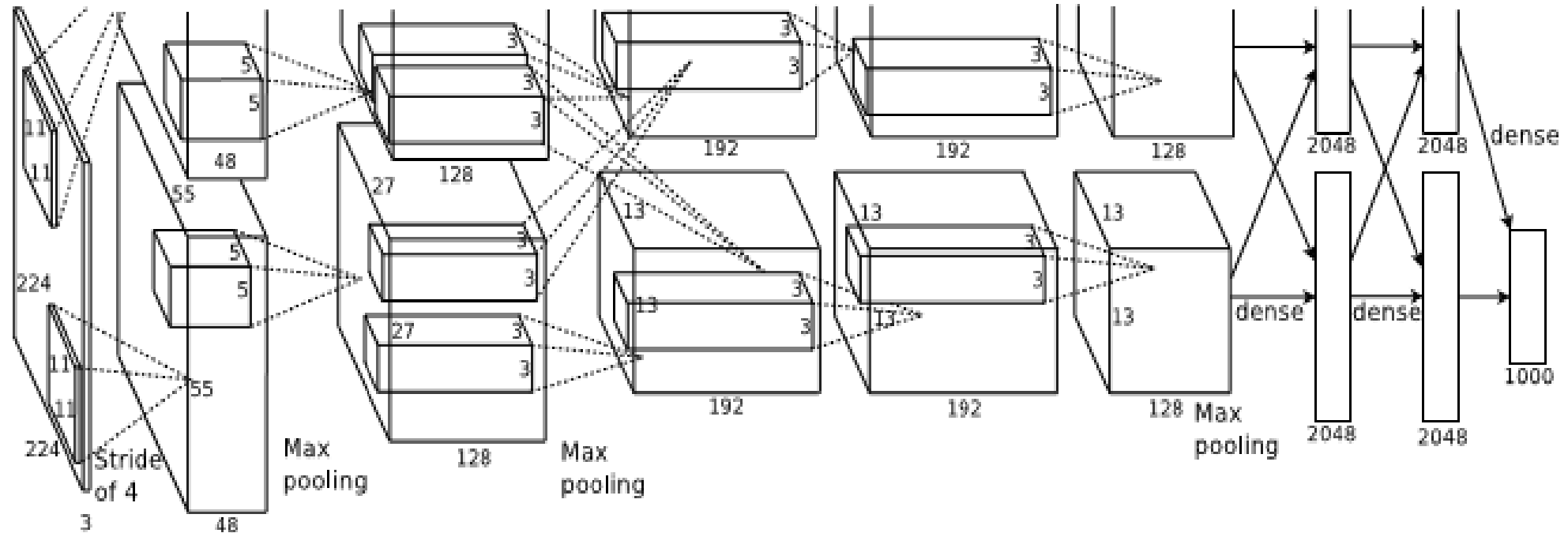
Pooling layers in CNNs summarize the outputs of neighboring groups of neurons in the same kernel map. Traditionally, the neighborhoods summarized by adjacent pooling units do not overlap (e.g., [17, 11, 4]). To be more precise, a pooling layer can be thought of as consisting of a grid of pooling units 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.

Overlapping 하여 pooling 함



# The Architecture

## - Overall Architecture



8 layers : 5개의 conv, 3개의 FC layer

마지막 FC layer output은 1000개 softmax로 들어가서 1000개의 클래스로 나누어짐

conv3에서는 모든 커널이 conv4에 연결 **왜?**

conv2,4,5는 다 같은 GPU

## Reducing Overfitting

parameter가 많은 모델은 overfitting 문제의 발생이 높음

### - Data Augmentation

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random  $224 \times 224$  patches (and their horizontal reflections) from the  $256 \times 256$  images and training our network on these extracted patches<sup>4</sup>. 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  $224 \times 224$  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,

data set을 늘리는 것

- translation, reflection 사용
- RGB의 intensity 변경

## Reducing Overfitting

### - 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.

각각 히든 뉴런 값을 50% 확률로 0으로 셋팅  
첫 두개의 FC layer에 적용

## Results

Model	Top-1	Top-5
<i>Sparse coding [2]</i>	<i>47.1%</i>	<i>28.2%</i>
<i>SIFT + FVs [24]</i>	<i>45.7%</i>	<i>25.7%</i>
CNN	<b>37.5%</b>	<b>17.0%</b>

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

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
<i>SIFT + FVs [7]</i>	—	—	26.2%
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	<b>16.4%</b>
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	<b>15.3%</b>

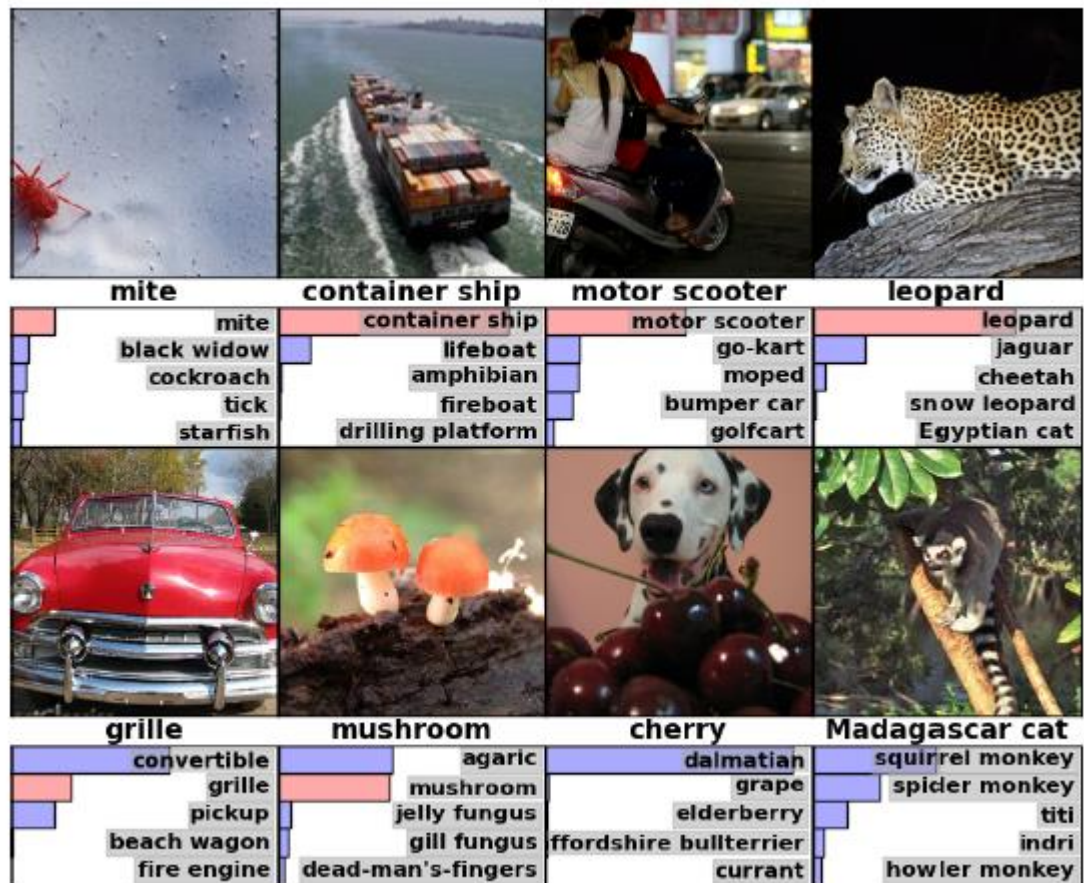
Table 2: Comparison of error rates on ILSVRC-2012 validation and test sets. In *italics* are best results achieved by others. Models with an asterisk\* were “pre-trained” to classify the entire ImageNet 2011 Fall release. See Section 6 for details.

CNN은 top-5 에러율 18.2%

5개의 CNN의 에러율 16.4%

## Results

## - Qualitative Evaluations



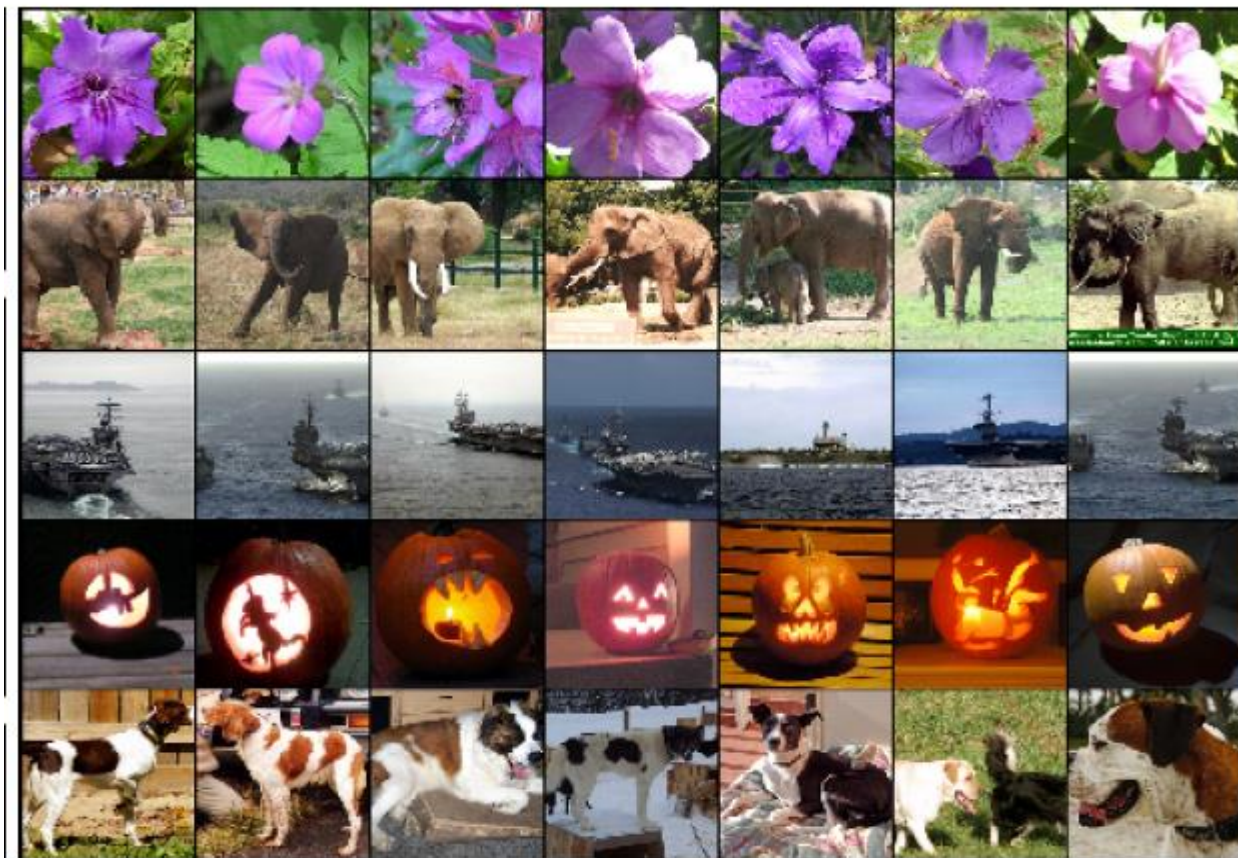
8개의 테스트 이미지 들에서 top-5 예측들을 계산함

top-5에서 정답을 맞춘 것들은 빨간색



## Results

### - Qualitative Evaluations



비주얼로 증명하는 방법

마지막 4096차원의 히든레이어에서 이미지로 부터 만들어진 feature activation를 고려

만약 두 이미지가 유클리디안 거리가 작은 feature activation를 가진다면, 그 두 이미지가 하이 레벨에서 매우 유사하다고 말할 수 있음.

첫번째 열 사진: 테스트 이미지  
나머지 열 사진: 첫번째 열과 비슷한 마지막 feature activation 벡터를 갖는 training 이미지들

## 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.

네트워크에서 Conv layer가 한개라도 제거되면 네트워크 성능이 저하됨

네트워크의 깊이가 중요