AlexNet

week 3

(01.31)

1. Introduction

2. Dataset

3. Architecture

4. Reducing Overfitting

5. Details of learning

6. Results

7. Discussion

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 current-best 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.

Improving machine performance
- larger dataset
- powerful model
- better techniques for preventing overfitting

간단한 recognition task
~> 작은 Dataset에서 충분한 기능 학습.

실제 다양한 사물 존재
⇒ 더 큰 training set 필요

- LabelMe ⇒ hundreds of thousands image
- ImageNet
⇒ 22000 categories,
대략 1500만개의 고해상도 image

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.

방대한 양의 object recognition을 진행하는데에 CNNs 적합함

- depth와 breadth 조절해 capacity 조정
- 이미지의 특질에 강력한 추측 가능
- 비슷한 사이즈의 다른 neural network에 비해 학습하기 쉬움 (fewer connection, fewer parameter.
  - 이론상 약간 떨어진 성능.

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[1]. 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.

◦ CNN의 단점

- 상당한 양의 고해상도 image를 처리하는 데에 공간적 비용

⇒ 최적화된 2D-convolution 시행가능한 GPU 등장

- ImageNet 관련 ILSVRC 대회에 2010, 2012년 출전

→ 2012년 우승

• 현재 GPU 성능과 제한된 training time.

⇒ GPU 성능이 개선된다면 결과 향상 가능

1. Introduction

2. Dataset

3. Architecture

4. Reducing Overfitting

5. Details of learning

6. Results

7. Discussion

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.

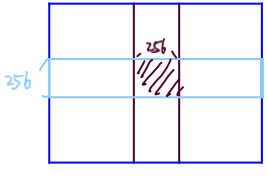
ImageNet consists of variable-resolution images, while our system requires a constant input dimensionality. Therefore, we down-sampled the images to a fixed resolution of 256×256. Given a rectangular image, we first rescaled the image such that the shorter side was of length 256, and then cropped out 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.

○ 2010년부터 ILSVRC 개최

- 1000개의 카테고리, 각각 1000개의 images
- 120만개의 training image,
- 5만개의 validation image
- 15만개의 testing image

○ 다양한 해상도의 사진

- 가로, 세로 중 짧은 쪽 길이 256



1. Introduction

2. Dataset

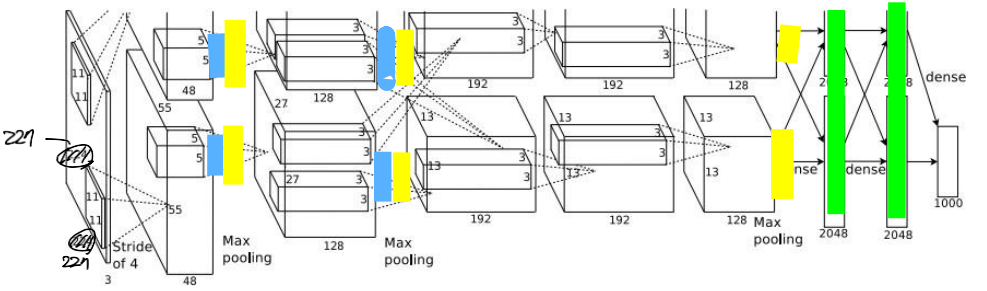
3. Architecture

4. Reducing Overfitting

5. Details of learning

6. Results

7. Discussion



: Local Response Normalization.

: Max-Pooling

Drop out

1st layer.

Input : 227 x 227 x 3

Kernel : 11 x 11 x 3

Padding : 0

stride : 4

Output : $\frac{227-11}{4}+1=55$

1st-Max-pooling

input : $55 \times 55 \times \frac{ab}{2}$

→ Kernel : 3 x 3 x 48

→ stride : 2

→ output : $\frac{55-3}{2}+1=27$

2nd - layer.

input : 27 x 27 x 48

Kernel : 5 x 5 x 48

stride : 1

padding : 2

output : $\frac{27-5+4}{2}+1=27$

2nd-Max-pooling

input : 27 x 27 x 128

Kernel : 3 x 3 x 128

Stride : 2

output : $\frac{27-3}{2}+1=13$

3rd-layer

input : 13 x 13 x 128

Kernel : 3 x 3 x 256

Stride : 1

padding : 1.

output : $\frac{13-3+2}{1}+1=13$

4th layer

input : 13 x 13 x 192

Kernel : 3 x 3 x 192

Stride : 1

padding : 1

output : $\frac{13-3+2}{1}+1=13$

5th layer

input : 13 x 13 x 192

Kernel : 3 x 3 x 192

Stride : 1

padding : 1

output : $\frac{13-3+2}{1}+1=13$

5th-Max-pooling

input : 13 x 13 x 128

Kernel : 3 x 3 x 128

Stride : 2

output : $\frac{13-3}{2}+1=6$

3.1 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.

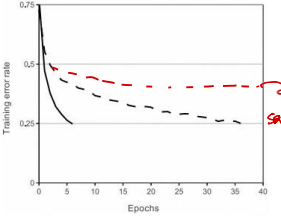


Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with $\tanh$ neurons **(dashed line)**. The learning rates for each network were chosen independently to make training as fast as possible. 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.

- ReLU
  - tanh, sigmoid ⇝ saturating nonlinearity
  - training에 소요되는 시간을 현저하게 감소시킴
- $f(x) = |\tanh|$ (대안 존재)

non-saturating

↓

Needs for AlexNet?

⇒ faster learning

3.2 Training on Multiple GPUs

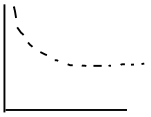
A single GTX 580 GPU has only 3GB of memory, which limits the maximum size of the networks that can be trained on it. It turns out that 1.2 million training examples are enough to train networks which are too big to fit on one GPU. Therefore we spread the net across two GPUs. Current GPUs are particularly 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 Cireşan 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].

- 한개의 GPU로는 무리, ⟿ 2개의 GPU 병렬 사용
  - 3rd layer에서만 communicate 가능 ⇒ 가능한 계산량까지 communicate 조절 가능
  - top-1, top-5 오차율 각각 1.7%, 1.2% 줄임

- 하나의 GPU로 각각 절반의 연산을 하는 것보다 시간적으로 효율적임

saturating?



- ReLU :) Normalization 없이 saturating 방지 가능
  - ReLU 값이 너무 커지면 주변 Neuron에 영향

3.3 Local Response Normalization

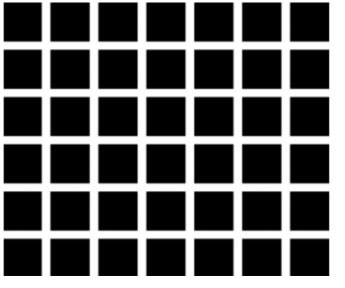
ReLUs have the desirable property that they do not require input normalization to prevent them from saturating. If at least some training examples produce a positive input to a ReLU, learning will happen in that neuron. However, we still find that the following local normalization scheme aids generalization. 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

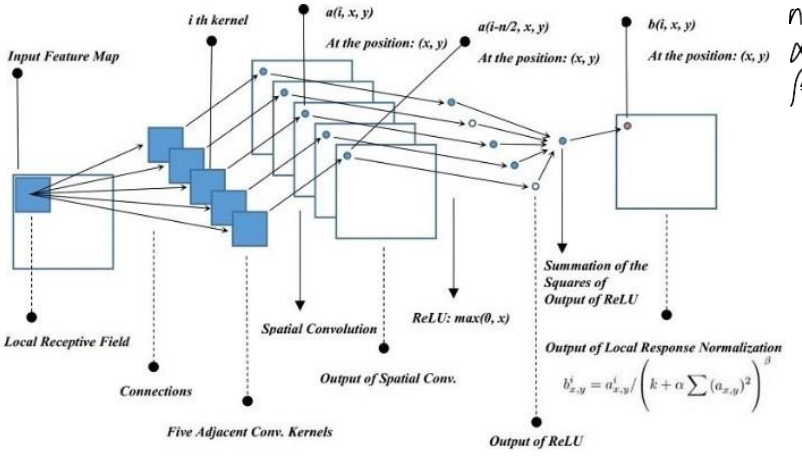
$$b^i_{x,y} = a^i_{x,y} \Big/ \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^\beta$$

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 $k = 2$, $n = 5$, $\alpha = 10^{-4}$, 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[3].

- 이용한 Normalization 존재
- 평균값 예제 없음
- top-1, top-5 오차율 각각 1.4%, 1.2% 줄임

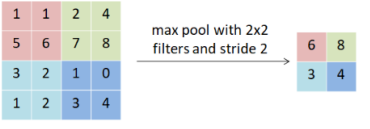




$k=2$
$n=5$
$\alpha = 10^{-4}$
$\beta = 0.75$

3.4 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.



o Reducing Overfitting

기존: stride = kernel ⇒ Overlapping 없음

Overlapping: stride 2 - kernel 3

- top-1, top-5 오차율 각각 0.4%, 0.3% 감소

⇝ Overfitting하기 더 어렵게 만들어줌

1. Introduction

2. Dataset

3. Architecture

4. Reducing Overfitting

5. Details of learning

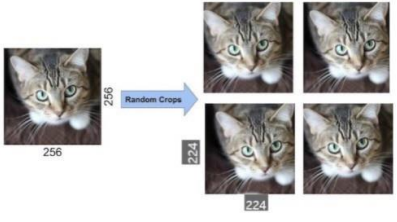
6. Results

7. Discussion

4.1 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 interdependent. 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.



- Reducing Overfitting
  - Data Augmentation
    - 적은 computation으로 transform → 계산적인 부담 X
    - CPU에서 이루어짐
  - Dropout

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 $I_{xy} = [I^R_{xy}, I^G_{xy}, I^B_{xy}]^T$ we add the following quantity:

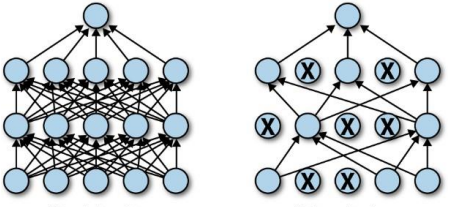
$$[\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T$$

where $\mathbf{p}_i$ 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%.

- PCA 진행
- 평균 0, std: 0.1 Gaussian 분포에서 Random값 추출 ⇒ 원래 Pixel의 RGB 값에 곱
- 원래 label 손상시키지 않는선에서 색상에 변형
- top-1 오차율 1% 감소

4.2 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.



(a) Standard Neural Net (b) After applying dropout

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.

1. Introduction

2. Dataset

3. Architecture

4. Reducing Overfitting

5. Details of learning

6. Results

7. Discussion

5 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

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

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

i : iteration index

v : momentum variable

ϵ : learning rate

$\left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i}$: i번째 batch에서 w_i를 사용해 구한 Gradient의 평균

- 초기화: std = 0.01, zero-mean Gaussian 분포
- neuron bias = 1
  - 2, 4, 5 Conv. layer, two FC layer

⇒ 초기 학습 가속화

- learning rate
  - validation error rate가 현상태에서 향상되지 않을 때 divided by 10
    - 0.01에서 시작, reduced 3 times

- 120만개의 image 90번 학습, 5-6일 소요

1. Introduction

2. Dataset

3. Architecture

4. Reducing Overfitting

5. Details of learning

6. Results

7. Discussion

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

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%	—
7 CNNs*	36.7%	15.4%	**15.3%**

Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ 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.

- 2개의 GPU 병렬 사용
- 각각의 GPU 특정 용어에 특화
  - color - agnostic
  - color - specified

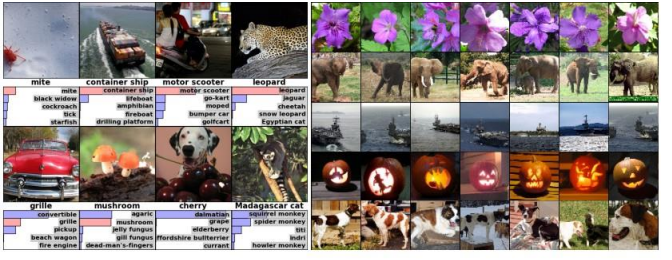


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010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

1. Introduction

2. Dataset

3. Architecture

4. Reducing Overfitting

5. Details of learning

6. Results

7. Discussion

7 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.

References

- 노루의 점핑일대기
  - [논문 리뷰]Alexnet- Image Classification with Deep Convolutional Neural Networks
- 딥러닝 공부방
  - [논문 리뷰] AlexNet(2012) 리뷰와 파이토치 구현
- Just my way
  - ImageNet Classification with Deep Convolutional Neural Networks
- 산 넘어 산 공부일지
  - [논문 리뷰] AlexNet(2012) 요약 및 리뷰
- Devmoon
  - [논문 리뷰] AlexNet / ImageNet Classification with Deep CNN