

Week 2

- Convolutional Neural Networks(CNNs)

2019.05.18
Solaris
(<http://solarisailab.com>)

Week1 복습 – Week1의 학습목표

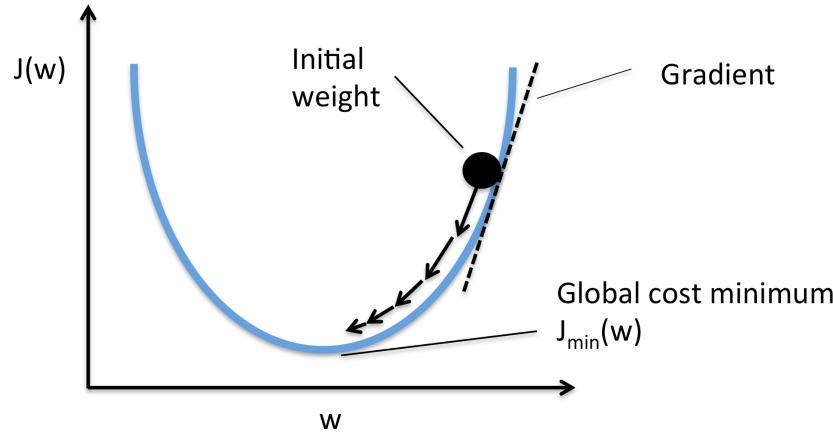
▣ 1강의 학습목표 :

1. 딥러닝의 등장배경, 머신러닝의 기초 이론, Gradient Descent, Softmax Regression의 개념을 이해한다.
2. ANN, Autoencoder의 구조와 동작과정을 이해한다.
3. TensorFlow를 이용한 MNIST 분류기를 구현해본다.

Week1 복습 - 머신러닝(Machine Learning)의 기본 프로세스

1. 학습하고자하는 가설(hypothesis) θ 을 수학적 표현식으로 나타낸다.
2. 가설의 성능을 측정할 수 있는 비용함수(cost function) $J(\theta)$ 을 정의한다.
3. cost function $J(\theta)$ 을 최소화할 수 있는 학습 알고리즘을 설계한다.

Week1 복습 – Gradient Descent



Gradient descent algorithm

repeat until convergence {
 $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ } (simultaneously update
 $j = 0$ and $j = 1$)
 learning rate derivative
 $\min_{\theta_1} J(\theta_1) \quad \theta_1 \in \mathbb{R}$.

Andrew Ng

Mini-batch gradient descent

- Batch gradient descent: Use all m examples in each iteration
- Stochastic gradient descent: Use 1 example in each iteration

Mini-batch gradient descent: Use b examples in each iteration

$b = \text{mini-batch size.}$ $b = 10. \quad \frac{2-100}{10}$

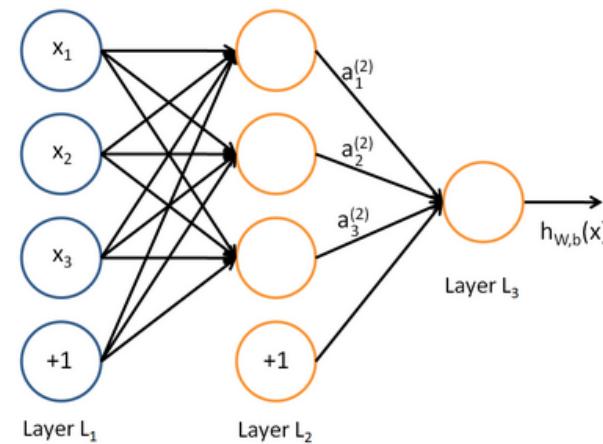
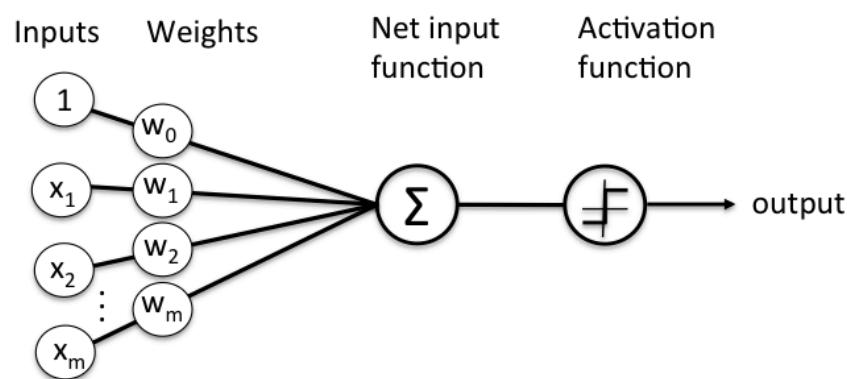
Get $b = 10$ examples $(x^{(i)}, y^{(i)}), \dots (x^{(i+9)}, y^{(i+9)})$

$\Rightarrow \theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) \cdot x_j^{(k)}$

$i := i + b$

Andrew Ng

Week1 복습 – Perceptron, Artificial Neural Networks(ANNs)



Schematic of Rosenblatt's perceptron.

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$

$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$

$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$

$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$

Week1 복습 – BackPropagation Algorithm

1. Perform a feedforward pass, computing the activations for layers L_2, L_3 , and so on up to the output layer L_{nr}

2. For each output unit i in layer n_l (the output layer), set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

3. For $l = n_l - 1, n_l - 2, n_l - 3, \dots, 2$

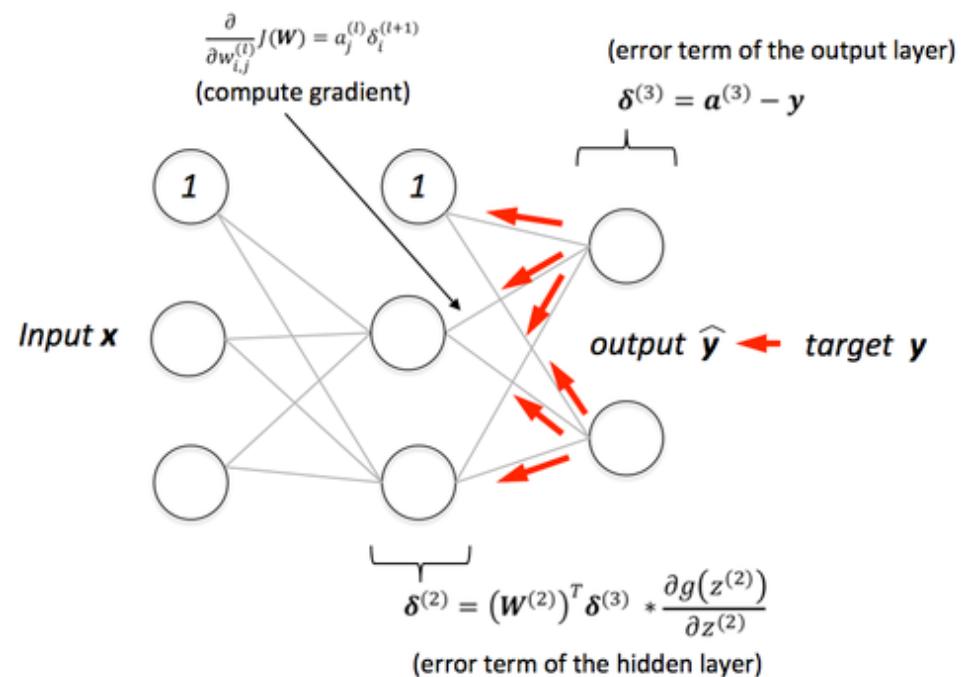
For each node i in layer l , set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

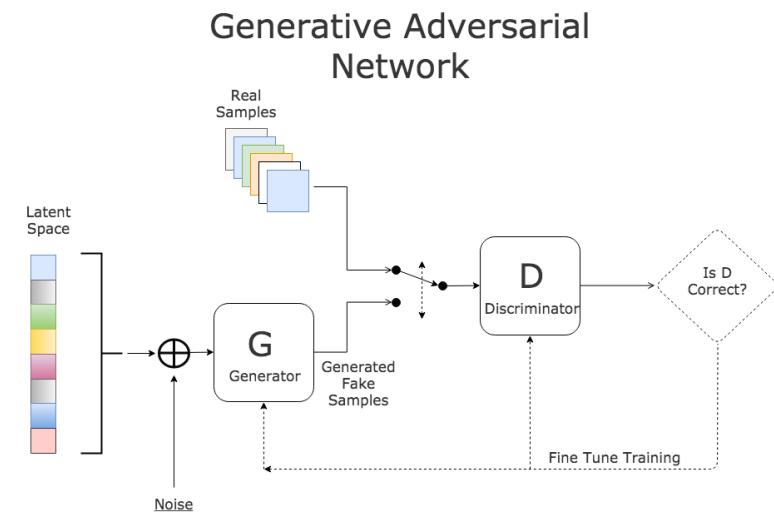
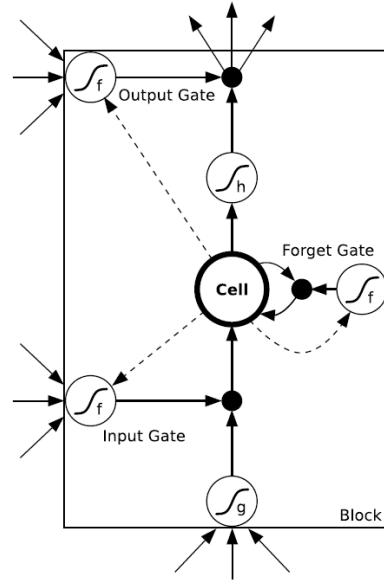
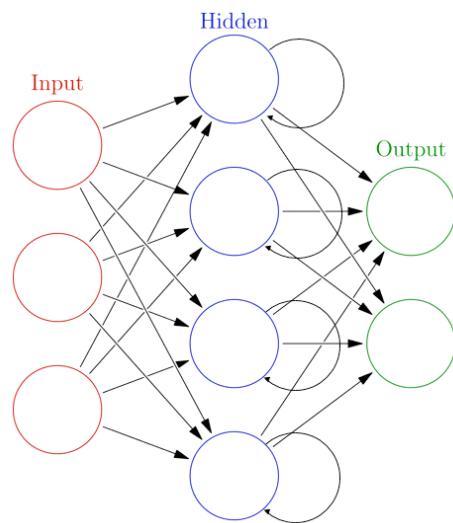
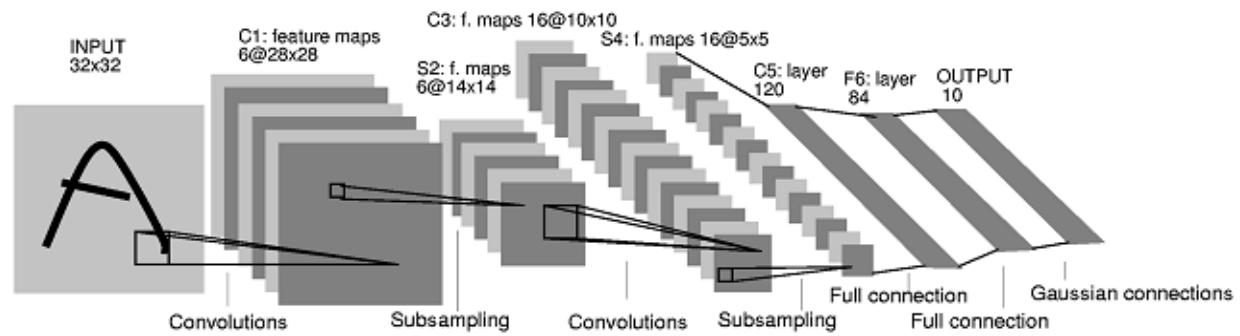
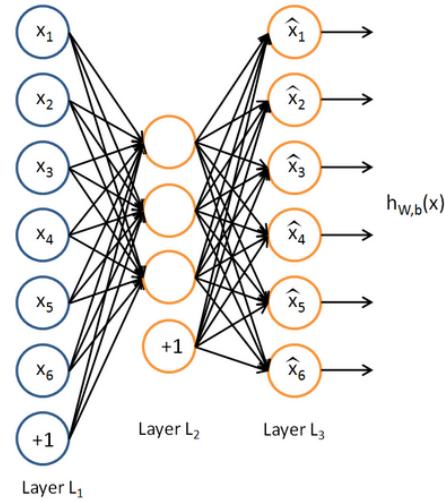
4. Compute the desired partial derivatives, which are given as:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)}$$

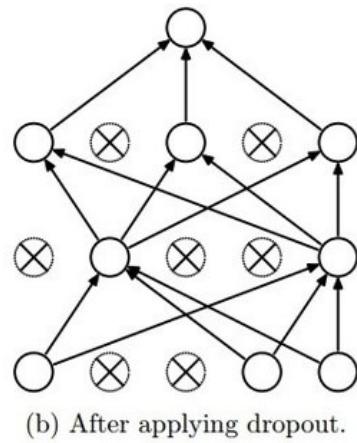
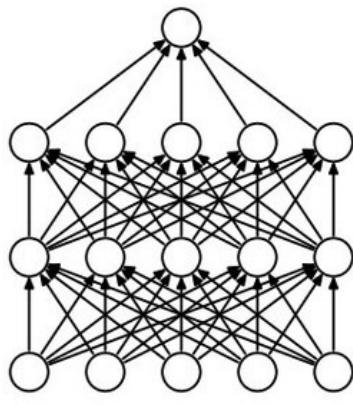
$$\frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)}.$$



Week1 복습 – 다양한 Deep Learning 모델들



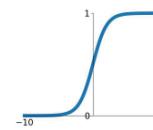
Week1 복습 – 효율적인 학습을 위한 테크닉들



Activation Functions

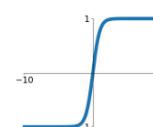
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



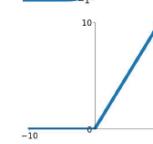
tanh

$$\tanh(x)$$



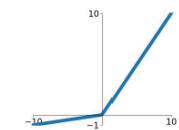
ReLU

$$\max(0, x)$$



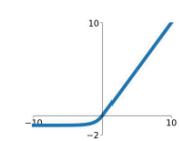
Leaky ReLU

$$\max(0.1x, x)$$



Maxout

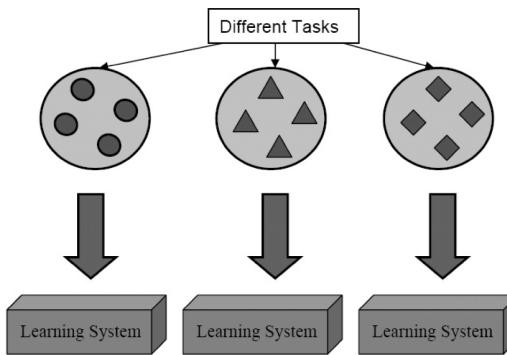
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$



ELU

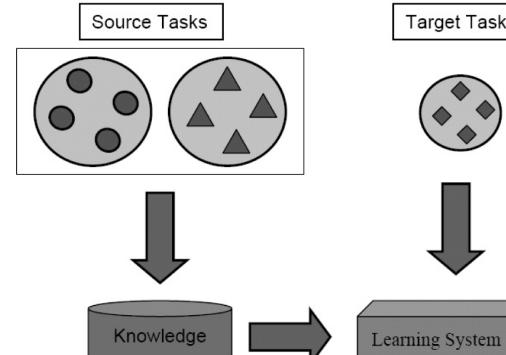
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

Learning Process of Traditional Machine Learning



(a)

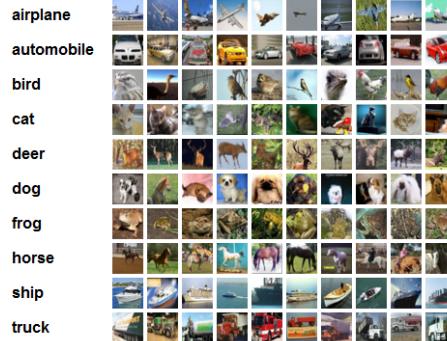
Learning Process of Transfer Learning



(b)

Week1 복습 – 수업에서 다룰 컴퓨터비전 문제들

0 0 0 0 0 0 0 0 0 0 0 0 0
 1 1 1 1 1 1 1 1 1 1 1 1 1
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

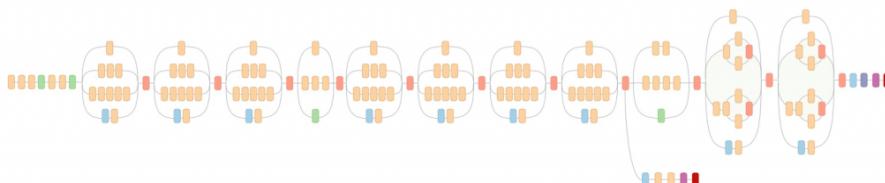


에스트라공: (무대 옆에서 몸을 비꼰다.) 앞으로!
 블라디미르: 나는 우릴 고맙다 (헐떡내 다친다) 이것 사실이오. (럭키에게) 하지만 생
 각을 할까? 무슨 소리를 터티봐
 블라디미르: 회연하는 성을 거칠게 좋니?
 에스트라공: 벌써 미칠태를 럭키의 주위를 지면줘요!
 블라디미르: 블리빨리! 이놈 좀 일으켜 세워요!
 그는 말을 엉죽고 블라디미르 모자를 높고 럭키의 모자를 고우지 않을 거들어 줘!
 블라디미르가 끈과 소년은 물러서며 이번 한데전 것도 없었나?
 에스트라공: 안 그래요!
 블라디미르: 조용히!
 침묵)
 에스트라공: 왜 왜 누구?
 블라디미르: 고도를 기다려야지.
 에스트라공: 참 그렇구 말야?
 블라디미르: 고도가 싫다니까. 벌써 시간이 흐르는 게 있으면

한국어 QA봇(질의응답봇)

bAbI 태스크를 위한 End-To-End Memory Network

Original works done by [Vinh Khuc](http://VinhKhuc.solarisailab.com)



Legend:
 Convolution
 Maxpool
 Dropout
 Fully connected
 Softmax

스토리

```
안톤 피곤하다
슈미트가 올 말아
제이슨은 피곤하다
안이 힘들로 갔다
```

질의(Question) ⓘ

왜 안은 힘들어 갔습니까?

응답(Answer)

정답(Answer) = '피곤한'
 신뢰 확률 = 99.91%
 맞았습니다!

텍스트	메모리 1 (Layer 1)	메모리 2 (Layer 2)	메모리 3 (Layer 3)
안톤 피곤하다	0.93	0.96	0.85
슈미트가 올 말아	0.04	0.00	0.00
제이슨은 피곤하다	0.02	0.04	0.15
안이 힘들로 갔다	0.00	0.00	0.00

Week1 복습 - 수업에서 다룰 컴퓨터비전 문제들

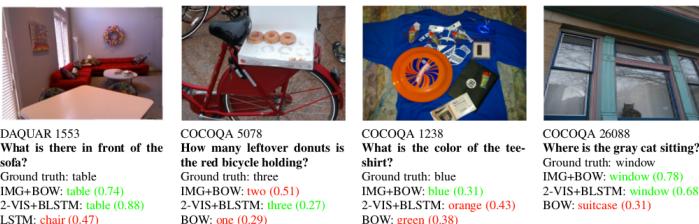
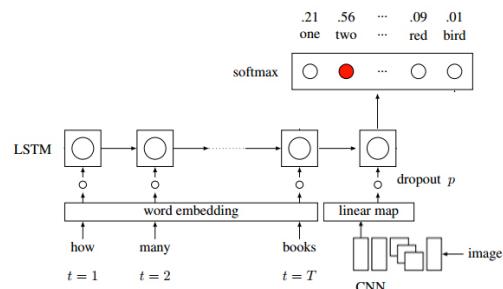
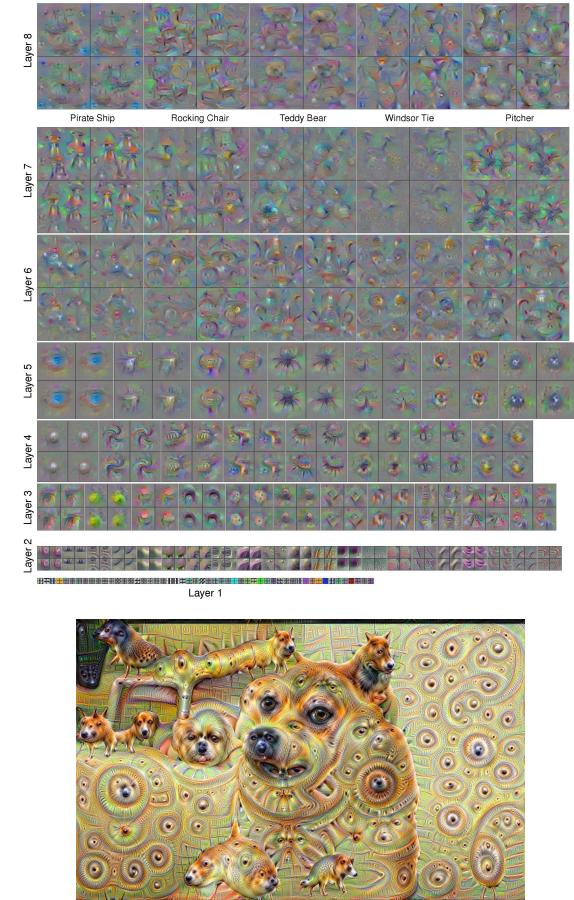
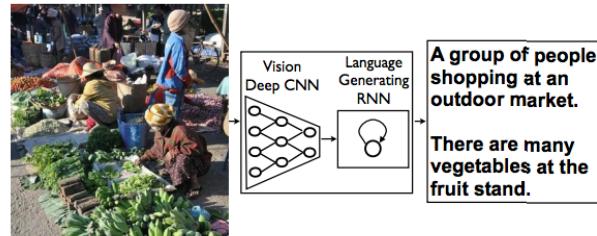
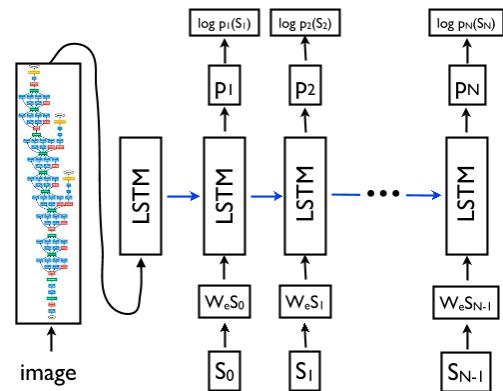


Figure 1: Sample questions and responses of a variety of models. Correct answers are in green and incorrect in red. The numbers in parentheses are the probabilities assigned to the top-ranked answer by the given model. The leftmost example is from the DAQUAR dataset, and the others are from our new COCO-QA dataset.



Week1 복습 – 수업에서 다룰 컴퓨터비전 문제들

Neural Style Transfer

Content Image



This image is licensed under CC-BY 3.0

Style Image



Starry Night by Van Gogh is in the public domain

Style Transfer!



This image copyright Justin Johnson, 2015. Reproduced with permission.

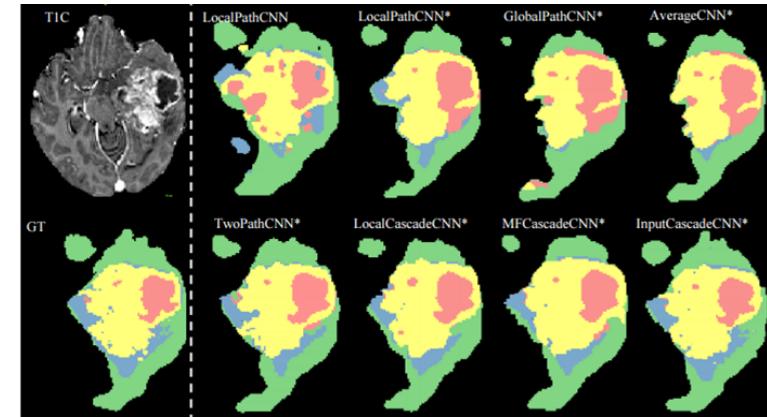
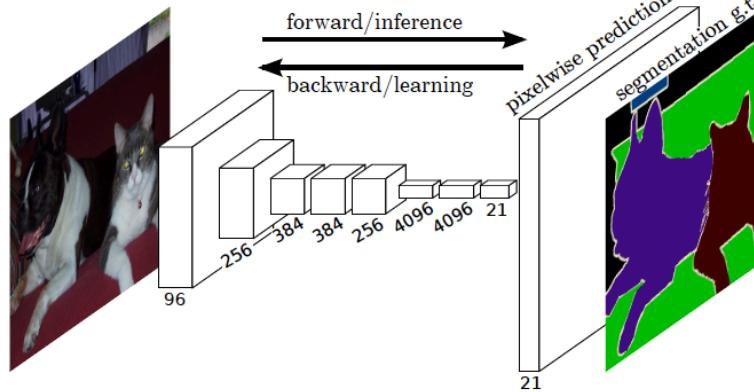
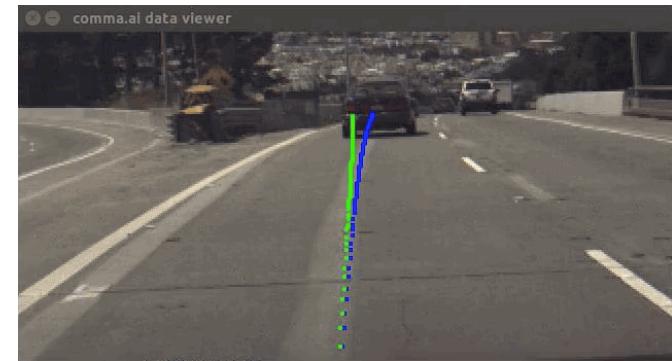
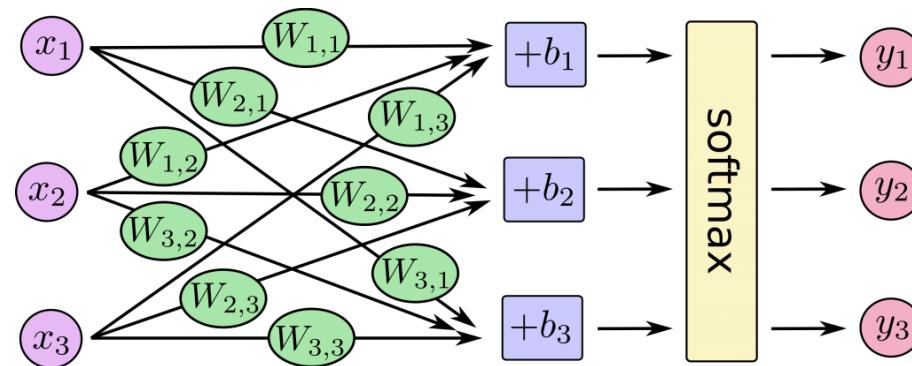


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

Week1 복습 – Softmax Regression

$$y = \text{softmax}(Wx + b)$$



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

Week1 복습 – Cost Functions

Squared-Error Cost Function

$$J(W, b; x, y) = \frac{1}{2} \|h_{W,b}(x) - y\|^2.$$

Cross-Entropy Cost function

$$H_{y'}(y) = - \sum_i y'_i \log(y_i)$$

Week1 - TensorFlow 실행과정

1. 그래프(Graph) 생성

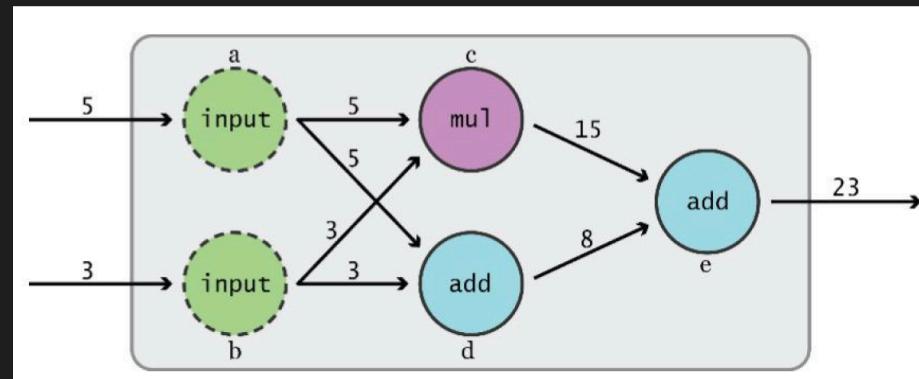
Slide credit: CS 20SI
(<https://goo.gl/Ez8wRq>)

2. Session을 이용한 그래프 상에 연산(operation) 실행

Data Flow Graphs

Phase 1: assemble a graph

Phase 2: use a session to execute operations in the graph.



Graph by TFFMI

Week2의 학습목표

▣ 2강의 학습목표 :

1. CNN의 기본 구성 요소-Convolution, Pooling(Subsampling)-의 개념을 이해한다.
2. 대표적인 CNN 모델-AlexNet, VGGNet, GoogLeNet-의 구조와 디자인철학을 이해한다.
3. TensorFlow를 이용한 CIFAR-10 분류기, Transfer Learning-Inception v3 Retraining-을 구현해본다.

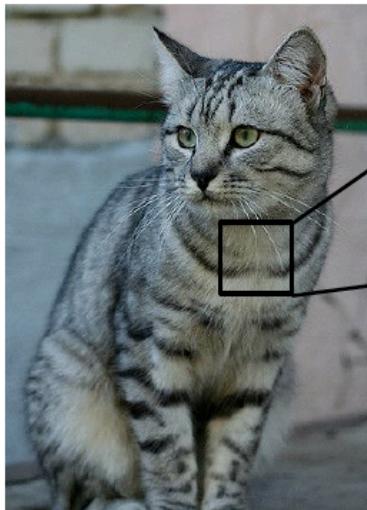
Outline

- ▣ 영상인식(Image Recognition)의 어려운 점들(Challenges)
- ▣ Convolutional Neural Networks(CNNs)
- ▣ Convolution, Pooling(Subsampling)
- ▣ CNNs에 대한 자세한 설명
- ▣ LeNet, AlexNet, VGGNet, GoogleLeNet(Inception v1) 소개
- ▣ CIFAR-10 Image Classification을 위한 CNNs
- ▣ Inception v3 Retraining을 통한 나만의 데이터셋에 대한 Image Classification

영상인식(Image Recognition)의 어려운 점들(Challenges) – 컴퓨터가 바라보는 영상(Image)

▣ 컴퓨터가 바라보는 화면

The Problem: Semantic Gap



```
[185 112 186 111 184 99 186 99 96 183 112 110 184 57 93 87]
[1 91 98 182 186 184 79 98 183 99 185 123 136 185 94 85]
[1 76 85 98 185 128 185 87 98 183 95 99 115 112 186 183 99 85]
[1 99 81 81 93 120 131 127 180 95 98 182 99 96 93 181 94]
[1 86 91 61 64 69 91 88 85 181 187 189 98 75 84 95 95]
[1 114 188 85 55 55 69 64 54 64 87 112 129 98 74 84 91]
[1 133 137 147 183 65 81 89 65 52 54 74 84 182 93 85 82]
[1 128 132 144 149 189 95 86 79 62 65 63 63 68 73 86 101]
[1 125 133 148 137 119 121 117 94 65 79 89 65 54 64 72 98]
[1 127 125 131 147 133 127 126 131 111 96 89 75 61 64 72 84]
[1 115 114 189 123 158 148 131 118 113 189 180 92 74 65 72 78]
[1 89 93 98 97 189 147 131 118 113 114 113 189 186 95 77 80]
[1 63 77 86 81 77 79 182 123 117 115 117 125 125 130 115 87]
[1 62 65 82 89 78 71 88 181 124 126 119 181 187 114 131 119]
[1 63 65 75 88 89 71 62 81 120 138 135 185 81 98 110 118]
[1 87 65 71 87 186 95 69 45 76 130 126 187 92 94 185 112]
[1 118 97 92 86 117 123 116 66 41 51 95 93 89 95 182 107]
[1 164 146 112 89 82 128 124 184 76 48 45 66 88 181 182 189]
[1 157 178 157 120 93 86 114 132 112 97 69 55 70 82 99 94]
[1 138 128 134 161 139 189 118 121 134 114 87 65 53 69 86]
[1 128 112 96 117 158 144 128 115 184 187 182 93 87 81 72 79]
[1 123 187 96 86 83 112 153 149 122 189 184 75 88 187 112 99]
[1 122 121 182 80 82 86 94 117 145 148 153 182 58 78 92 107]
[1 22 164 148 183 71 56 78 83 93 183 119 139 182 61 69 84])]
```

What the computer sees

An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

영상인식(Image Recognition)의 어려운 점들(Challenges) - 밝기변화

■ Illumination(밝기변화)

Challenges: Illumination



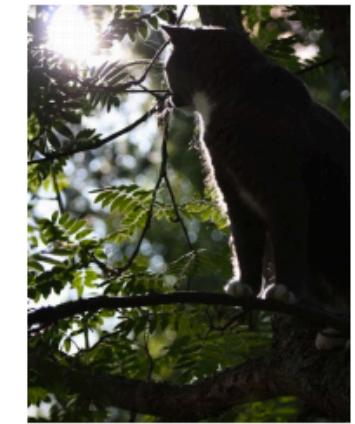
[This image is CC0 1.0 public domain](#)



[This image is CC0 1.0 public domain](#)



[This image is CC0 1.0 public domain](#)

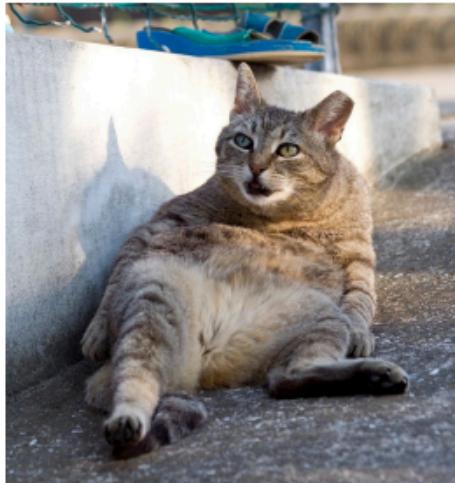


[This image is CC0 1.0 public domain](#)

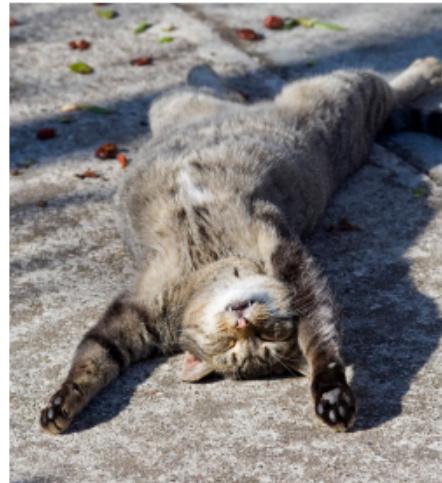
영상인식(Image Recognition)의 어려운 점들(Challenges) - 변형

▣ Deformation(변형)

Challenges: Deformation



This image by Umberto Salvagnin
is licensed under CC-BY 2.0



This image by Umberto Salvagnin
is licensed under CC-BY 2.0



This image by sare_bear is
licensed under CC-BY 2.0



This image by Tom Thai is
licensed under CC-BY 2.0

영상인식(Image Recognition)의 어려운 점들(Challenges) - 가림

▣ Occlusion(가림)

Challenges: Occlusion



[This image is CC0 1.0 public domain](#)



[This image is CC0 1.0 public domain](#)



[This image by Jonsson is licensed under CC-BY 2.0](#)

영상인식(Image Recognition)의 어려운 점들(Challenges) – 배경 장애

■ Background Clutter(배경 장애)

Challenges: Background Clutter



This image is CC0 1.0 public domain



This image is CC0 1.0 public domain

영상인식(Image Recognition)의 어려운 점들(Challenges) – Intraclass variation

- Intraclass variation(같은 클래스라도 다양한 형태로 보인다.)

Challenges: Intraclass variation



This image is CC0 1.0 public domain

기존의 Computer Vision Approach와 Deep Learning의 비교

■ Handcrafted Feature vs End-To-End Learning

■ CNNs의 핵심 아이디어 : Raw Image로부터로 특징들(Features)을 자동으로 학습하자!

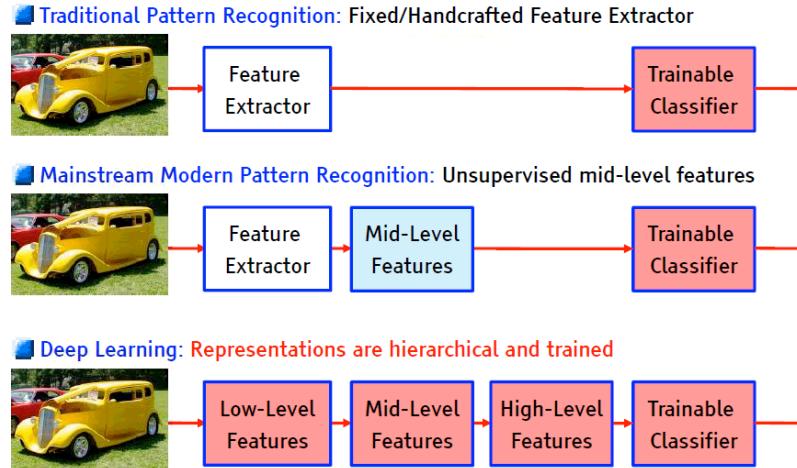
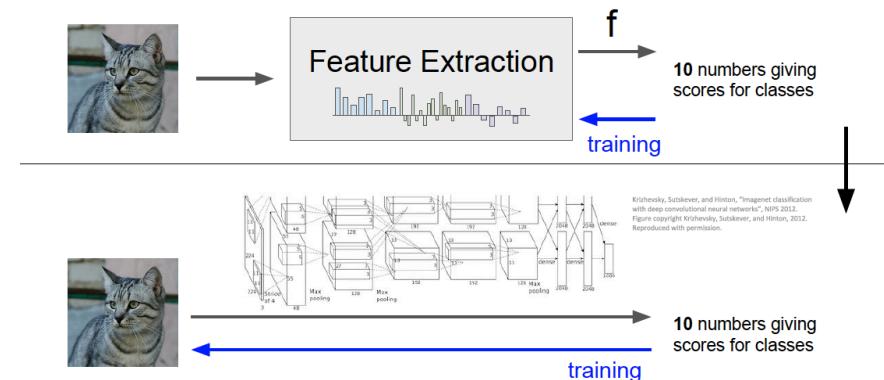


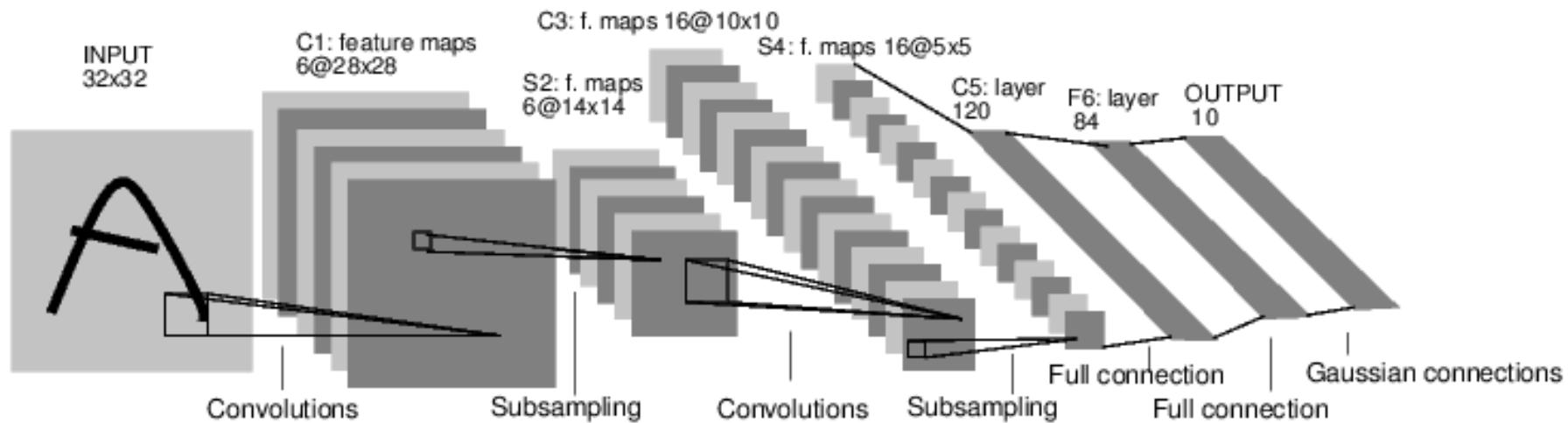
Image features vs ConvNets



Slide credit: Yann Lecun

CNNs의 시초 – LeNet (1989년)

- 1989년 Yann Lecun이 LeNet이라는 Convolutional Neural Networks(CNNs)의 구조를 제안

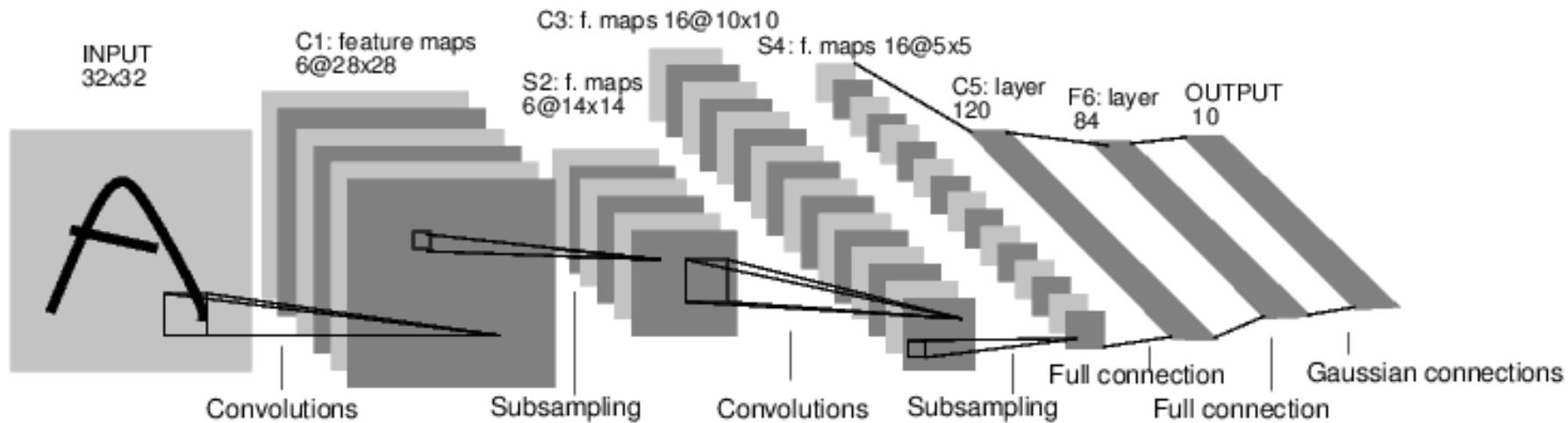


CNNs의 구성 요소 – Convolution, Pooling

▣ CNNs의 핵심 구조 :

1. Convolution

2. Pooling



Convolution

- Convolution을 이용한 효율적인 특징 추출 및 Dimension 축소

1 <small>×1</small>	1 <small>×0</small>	1 <small>×1</small>	0	0
0 <small>×0</small>	1 <small>×1</small>	1 <small>×0</small>	1	0
0 <small>×1</small>	0 <small>×0</small>	1 <small>×1</small>	1	1
0	0	1	1	0
0	1	1	0	0

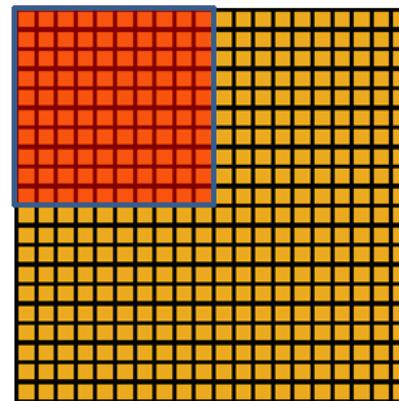
Image

4		

Convolved
Feature

Pooling(Subsampling)

- Max Pooling (가장 큰 값만 추출)-Mean pooling이나 Min Pooling도 가능-을 통해 특징을 더욱 더 압축적으로 표현

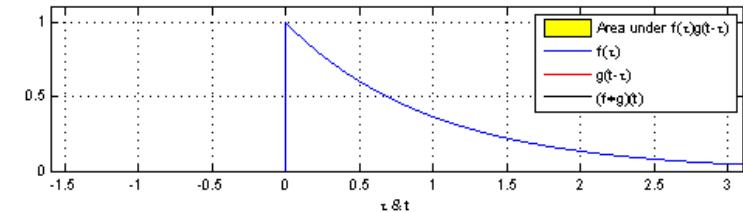


Convolved
feature Pooled
feature

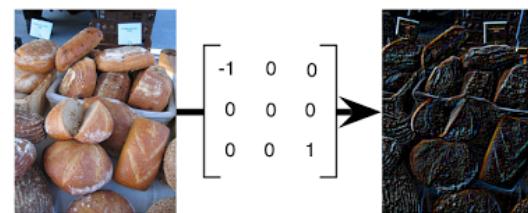
Convolution Operation

- 합성곱(Convolution)은 하나의 함수와 또 다른 함수를 반전 이동한 값을 곱한 다음, 구간에 대해 적분하여 새로운 함수를 구하는 수학 연산자이다.

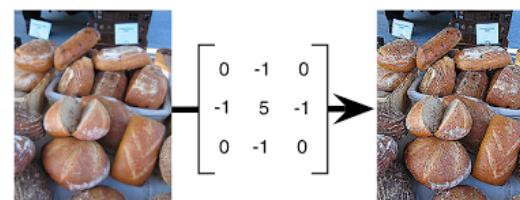
$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau) d\tau$$



- 컴퓨터 비전 분야에서 Convolution은 이미지에 특정한 Filter를 적용하여 이미지의 특징을 추출하는데 사용된다.



Edge Detection Kernel을 이용한 경우



Sharpness Kernel을 이용한 경우

다양한 필터들

Some basic kernels



input



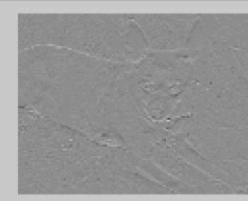
blur



sharpen



edge



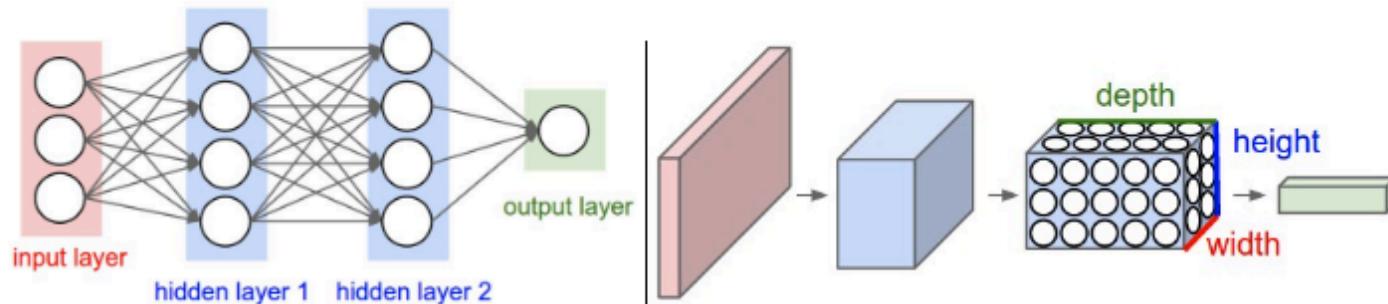
top sobel



emboss

CNNs – 3차원 데이터 표현

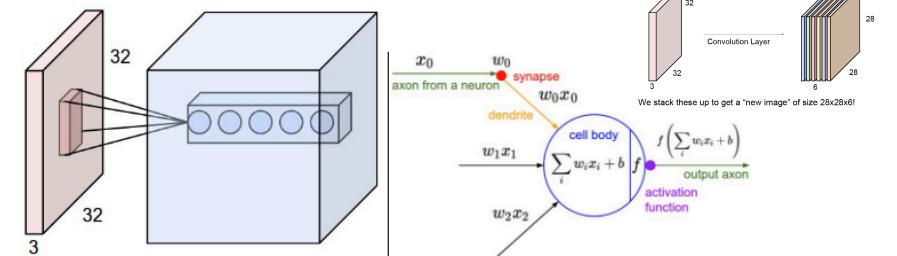
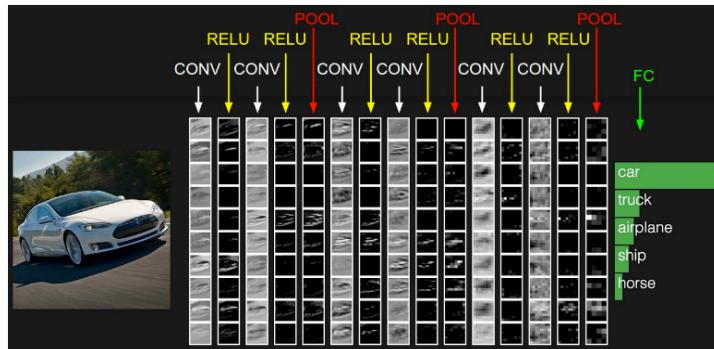
- 일반 신경망은 이미지를 다루기에 적절하지 않다. CIFAR-10 데이터의 경우 각 이미지가 $32 \times 32 \times 3$ (가로, 세로 32, 3개 컬러 채널)로 이뤄져 있어서 첫 번째 히든 레이어 내의 하나의 뉴런의 경우 $32 \times 32 \times 3 = 3072$ 개의 가중치가 필요하지만, 더 큰 이미지를 사용할 경우에는 같은 구조를 이용하는 것이 불가능하다. 예를 들어 $200 \times 200 \times 3$ 의 크기를 가진 이미지는 같은 뉴런에 대해 $200 \times 200 \times 3 = 120,000$ 개의 가중치를 필요로 하기 때문이다. 더욱이, 이런 뉴런이 레이어 내에 여러개 존재하므로 모수의 개수가 크게 증가하게 된다. 이와 같이 Fully-connectivity는 심한 낭비이며 많은 수의 모수는 곧 오버피팅(overfitting)으로 귀결된다.
- ConvNet은 입력이 이미지로 이뤄져 있다는 특징을 살려 좀 더 합리적인 방향으로 아키텍쳐를 구성할 수 있다. 특히 일반 신경망과 달리, ConvNet의 레이어들은 가로, 세로, 깊이의 3개 차원을 갖게 된다 (여기에서 말하는 깊이란 전체 신경망의 깊이가 아니라 액티베이션 볼륨 (activation volume)에서의 3번째 차원을 이야기 함). 예를 들어 CIFAR-10 이미지는 $32 \times 32 \times 3$ (가로, 세로, 깊이)의 차원을 갖는 입력 액티베이션 볼륨 (activation volume)이라고 볼 수 있다. 조만간 보겠지만, 하나의 레이어에 위치한 뉴런들은 일반 신경망과는 달리 앞 레이어의 전체 뉴런이 아닌 일부에만 연결이 되어 있다.



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

CNNs – Local Connectivity

- ▣ **개요 및 직관적인 설명.** CONV 레이어의 모수(parameter)들은 일련의 학습가능한 필터들로 이루어져 있다. 각 필터는 가로/세로 차원으로는 작지만 깊이 (depth) 차원으로는 전체 깊이를 아우른다. 포워드 패스 (forward pass) 때에는 각 필터를 입력 볼륨의 가로/세로 차원으로 슬라이딩 시키며 (정확히는 convolve 시키며) 2차원의 액티베이션 맵 (activation map)을 생성한다. 필터를 입력 위로 슬라이딩 시킬 때, 필터와 입력의 요소들 사이의 내적 연산 (dot product)이 이뤄진다. 직관적으로 설명하면, 이 신경망은 입력의 특정 위치의 특정 패턴에 대해 반응하는 (**activate**) 필터를 학습한다. 이런 액티베이션 맵 (activation map)을 깊이 (depth) 차원을 따라 쌓은 것이 곧 출력 볼륨이 된다. 그러므로 출력 볼륨의 각 요소들은 입력의 작은 영역만을 취급하고, 같은 액티베이션 맵 내의 뉴런들은 같은 모수들을 공유한다 (같은 필터를 적용한 결과이므로).
- ▣ **로컬 연결성 (Local connectivity).** 이미지와 같은 고차원 입력을 다룰 때에는, 현재 레이어의 한 뉴런을 이전 볼륨의 모든 뉴런들과 연결하는 것이 비 실용적이다. 대신에 우리는 레이어의 각 뉴런을 입력 볼륨의 로컬한 영역(local region)에만 연결할 것이다. 이 영역은 리셉티브 필드 (receptive field)라고 불리는 초모수 (hyperparameter)이다. 깊이 차원 측면에서는 항상 입력 볼륨의 총 깊이를 다룬다 (가로/세로는 작은 영역을 보지만 깊이는 전체를 본다는 뜻). 공간적 차원 (가로/세로)와 깊이 차원을 다루는 방식이 다르다는 걸 기억하자.
- ▣ **예제 1.** 예를 들어 입력 볼륨의 크기가 (CIFAR-10의 RGB 이미지와 같이) [32x32x3]이라고 하자. 만약 리셉티브 필드의 크기가 5x5라면, CONV 레이어의 각 뉴런은 입력 볼륨의 [5x5x3] 크기의 영역에 가중치 (weight)를 가하게 된다 (총 $5 \times 5 \times 3 = 75$ 개 가중치). 입력 볼륨 (RGB 이미지)의 깊이가 3이므로 마지막 숫자가 3이 된다는 것을 기억하자. **예제 2.** 입력 볼륨의 크기가 [16x16x20]이라고 하자. 3x3 크기의 리셉티브 필드를 사용하면 CONV 레이어의 각 뉴런은 입력 볼륨과 $3 \times 3 \times 20 = 180$ 개의 연결을 갖게 된다. 이번에도 입력 볼륨의 깊이가 20이므로 마지막 숫자가 20이 된다는 것을 기억하자.



Left: An example input volume in red (e.g. a 32x32x3 CIFAR-10 image), and an example volume of neurons in the first Convolutional layer. Each neuron in the convolutional layer is connected only to a local region in the input volume spatially, but to the full depth (i.e. all color channels). Note, there are multiple neurons (5 in this example) along the depth, all looking at the same region in the input - see discussion of depth columns in text below. Right: The neurons from the Neural Network chapter remain unchanged: They still compute a dot product of their weights with the input followed by a non-linearity, but their connectivity is now restricted to be local spatially.

CNNs의 볼륨 크기를 결정하는 요소 – 깊이, stride, 제로 패딩

- ▣ **공간적 배치.** 지금까지는 컨볼루션 레이어의 한 뉴런과 입력 볼륨의 연결에 대해 알아보았다. 그러나 아직 출력 볼륨에 얼마나 많은 뉴런들이 있는지, 그리고 그 뉴런들이 어떤식으로 배치되는지는 다루지 않았다. 3개의 hyperparameter들이 출력 볼륨의 크기를 결정하게 된다. 그 3개 요소는 바로 **깊이**, **stride**, 그리고 **제로 패딩 (zero-padding)** 이다. 이들에 대해 알아보자.
- ▣ 먼저, 출력 볼륨의 **깊이**는 우리가 결정할 수 있는 요소이다. 컨볼루션 레이어의 뉴런들 중 입력 볼륨 내 동일한 영역과 연결된 뉴런의 개수를 의미한다. 마치 일반 신경망에서 히든 레이어 내의 모든 뉴런들이 같은 입력값과 연결된 것과 비슷하다. 앞으로 살펴보겠지만, 이 뉴런들은 입력에 대해 서로 다른 특징 (feature)에 활성화된다 (activate). 예를 들어, 이미지를 입력으로 받는 첫 번째 컨볼루션 레이어의 경우, 깊이 축에 따른 각 뉴런들은 이미지의 서로 다른 엣지, 색깔, 블롭(blob) 등에 활성화된다. 앞으로는 인풋의 서로 같은 영역을 바라보는 뉴런들을 **깊이 컬럼 (depth column)**이라고 부르겠다.
- ▣ 두 번째로 어떤 간격 (가로/세로의 공간적 간격)으로 깊이 컬럼을 할당할지를 의미하는 **stride**를 결정해야 한다. 만약 stride가 1이라면, 깊이 컬럼을 1칸마다 할당하게 된다 (한 칸 간격으로 깊이 컬럼 할당). 이럴 경우 각 깊이 컬럼들은 receptive field 상 넓은 영역이 겹치게 되고, 출력 볼륨의 크기도 매우 커지게 된다. 반대로, 큰 stride를 사용한다면 receptive field끼리 좁은 영역만 겹치게 되고 출력 볼륨도 작아지게 된다 (깊이는 작아지지 않고 가로/세로만 작아지게 됨).
- ▣ 조만간 살펴보겠지만, 입력 볼륨의 가장자리를 0으로 패딩하는 것이 좋을 때가 있다. 이 **zero-padding**은 hyperparameter이다. zero-padding을 사용할 때의 장점은, 출력 볼륨의 공간적 크기(가로/세로)를 조절할 수 있다는 것이다. 특히 입력 볼륨의 공간적 크기를 유지하고 싶은 경우 (입력의 가로/세로 = 출력의 가로/세로) 사용하게 된다.
- ▣ 출력 볼륨의 공간적 크기 (가로/세로)는 입력 볼륨 크기 (W), CONV 레이어의 리셉티브 필드 크기(F)와 stride (S), 그리고 제로 패딩 (zero-padding) 사이즈 (P)의 합으로 계산할 수 있다. $(W-F+2P)/S+1$ 을 통해 알맞은 크기를 계산하면 된다. 만약 이 값이 정수가 아니라면 stride가 잘못 정해진 것이다. 이 경우 뉴런들이 대칭을 이루며 깔끔하게 배치되는 것이 불가능하다. 다음 예제를 보면 이 수식을 좀 더 직관적으로 이해할 수 있을 것이다:

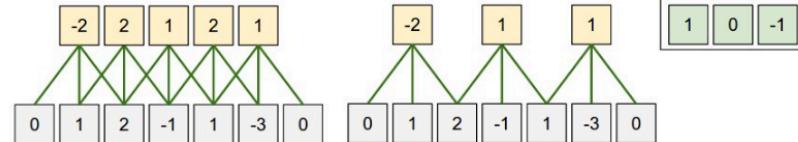


Illustration of spatial arrangement. In this example there is only one spatial dimension (x-axis), one neuron with a receptive field size of $F = 3$, the input size is $W = 5$, and there is zero padding of $P = 1$. Left: The neuron strided across the input in stride of $S = 1$, giving output of size $(5 - 3 + 2)/1 + 1 = 5$. Right: The neuron uses stride of $S = 2$, giving output of size $(5 - 3 + 2)/2 + 1 = 3$. Notice that stride $S = 3$ could not be used since it wouldn't fit neatly across the volume. In terms of the equation, this can be determined since $(5 - 3 + 2) = 4$ is not divisible by 3.
The neuron weights are in this example [1,0,-1] (shown on very right), and its bias is zero. These weights are shared across all yellow neurons (see parameter sharing below).

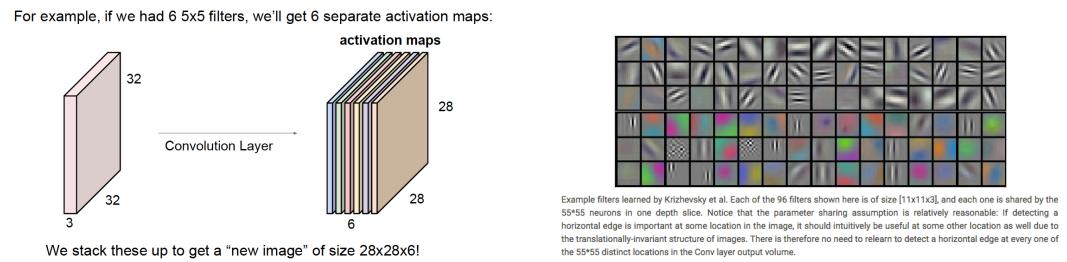
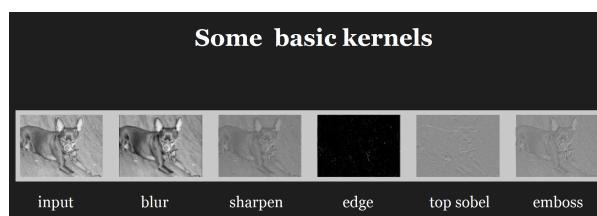
0	0	0	0	0	0		
0							
0							
0							
0							

출력 볼륨의 가로.세로 크기
 $(W-F+2P)/S + 1$

출력 볼륨의 깊이
K(커널 개수)

파라미터 공유(Parameter Sharing)

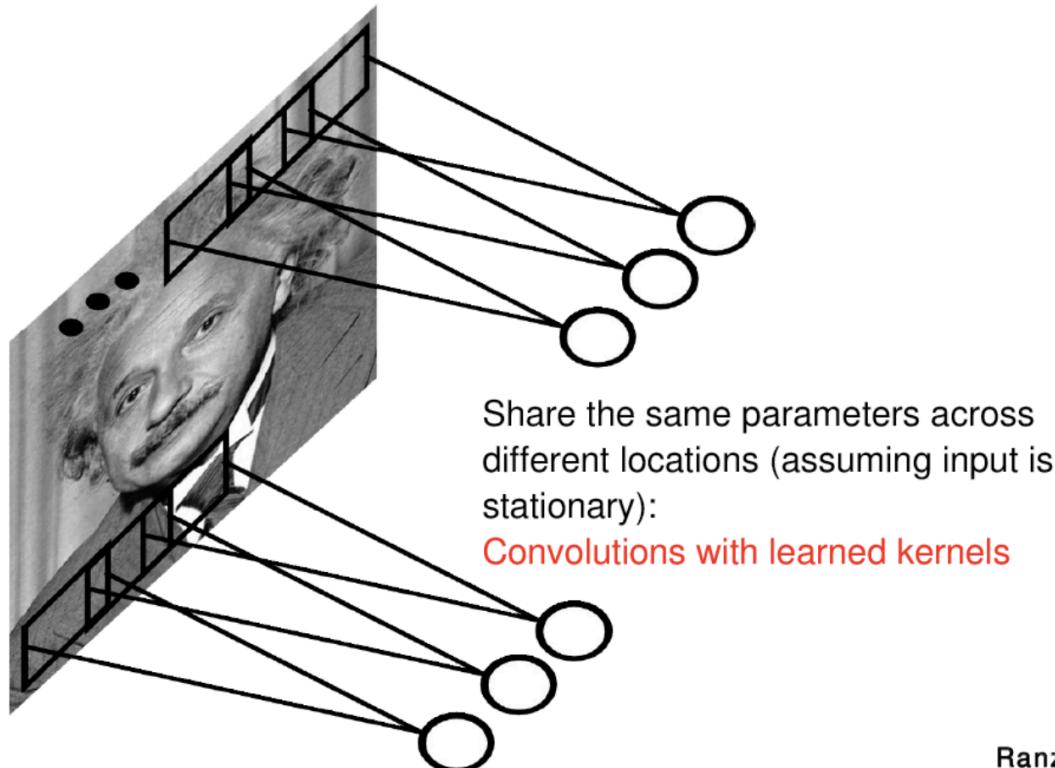
- **실제 예제.** 이미지넷 대회에서 우승한 Krizhevsky et al.(AlexNet)의 모델의 경우 $[227 \times 227 \times 3]$ 크기의 이미지를 입력으로 받는다. 첫 번째 컨볼루션 레이어에서는 리셉티브 필드 $F=11$, stride $S=4$ 를 사용했고 제로 패딩은 사용하지 않았다 $P=0$. $(227 - 11)/4 + 1 = 55$ 이고 컨볼루션 레이어의 깊이는 $K=96$ 이므로 이 컨볼루션 레이어의 크기는 $[55 \times 55 \times 96]$ 이 된다. 각각의 $55 \times 55 \times 96$ 개 뉴런들은 입력 볼륨의 $[11 \times 11 \times 3]$ 개 뉴런들과 연결되어 있다. 그리고 각 깊이의 모든 96개 뉴런들은 입력 볼륨의 같은 $[11 \times 11 \times 3]$ 영역에서 서로 다른 가중치를 가지고 연결된다.
- **파라미터 공유.** 파라미터 공유 기법은 컨볼루션 레이어의 파라미터 개수를 조절하기 위해 사용된다. 위의 실제 예제에서 보았듯, 첫 번째 컨볼루션 레이어에는 $55 \times 55 \times 96 = 290,400$ 개의 뉴런이 있고 각각의 뉴런은 $11 \times 11 \times 3 = 363$ 개의 가중치와 1개의 바이어스를 가진다. 첫 번째 컨볼루션 레이어만 따져도 총 파라미터 개수는 $290400 \times 363 = 105,705,600$ 개가 된다. 분명히 이 숫자는 너무 크다.
- 사실 적절한 가정을 통해 파라미터 개수를 크게 줄이는 것이 가능하다: **(x,y)에서 어떤 patch feature가 유용하게 사용되었다면, 이 feature는 다른 위치 (x2,y2)에서도 유용하게 사용될 수 있다.** 3차원 볼륨의 한 슬라이스 (깊이 차원으로 자른 2차원 슬라이스)를 **depth slice**라고 하자 ($[55 \times 55 \times 96]$ 사이즈의 볼륨은 각각 $[55 \times 55]$ 의 크기를 가진 96개의 depth slice임). 앞으로는 각 depth slice 내의 뉴런들이 같은 가중치와 바이어스를 가지도록 제한할 것이다. 이런 파라미터 공유 기법을 사용하면, 예제의 첫 번째 컨볼루션 레이어는 (depth slice 당) 96개의 고유한 가중치를 가져서 총 $96 \times 11 \times 11 \times 3 = 34,848$ 개의 고유한 가중치, 또는 바이어스를 합쳐서 34,944개의 파라미터를 갖게 된다. 또는 각 depth slice에 존재하는 55*55개의 뉴런들은 모두 같은 파라미터를 사용하게 된다.
- 한 depth slice내의 모든 뉴런들이 같은 가중치 벡터를 갖기 때문에 컨볼루션 레이어의 forward pass는 입력 볼륨과 가중치 간의 **컨볼루션**으로 계산될 수 있다 (컨볼루션 레이어라는 이름이 붙은 이유). 그러므로 컨볼루션 레이어의 가중치는 **필터(filter)** 또는 **커널(kernel)**이라고 부른다. 컨볼루션의 결과물은 **액티베이션 맵(activation map, [55x55] 사이즈)**이며 각 깊이에 해당하는 필터의 액티베이션 맵들을 쌓으면 최종 출력 볼륨 ($[55 \times 55 \times 96]$ 사이즈) 가 된다.



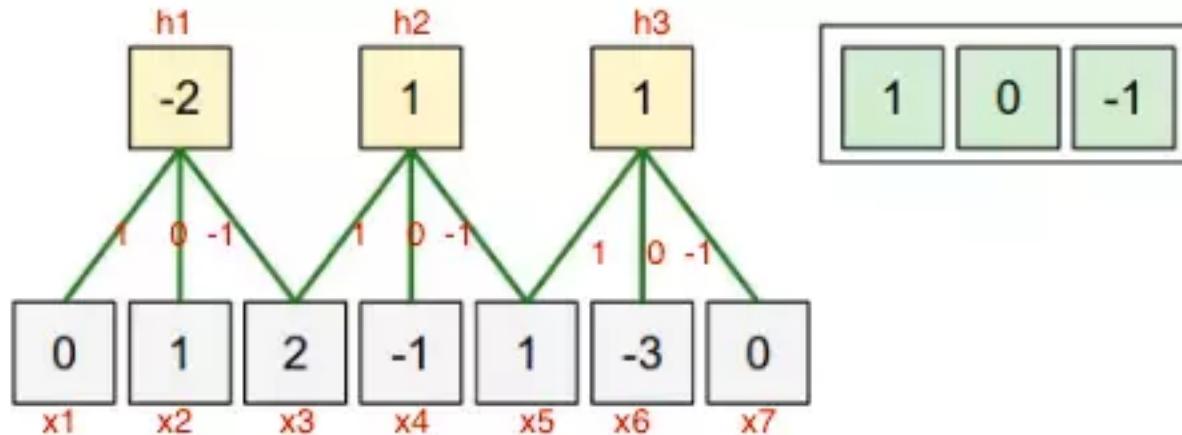
파라미터 공유(Parameter Sharing)

- ▣ 자연 이미지는 정적인(stationary) 특징이 있다. 이것이 의미하는 바는 이미지의 한 부분의 통계치가 다른 부분의 통계치와 비슷하다는 것이다. 이것은 우리가 이미지의 하나의 부분으로부터 학습한 특징들을 이미지의 다른 부분들에 대해서도 적용할 수 있다는 것을 의미한다. 그리고 우리는 같은 특징들을 전체 지역에 적용할 수 있다.

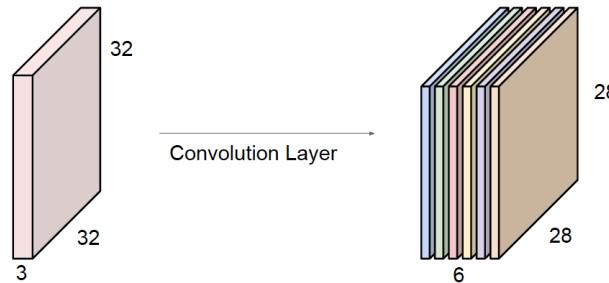
Weight sharing: Parameter saving



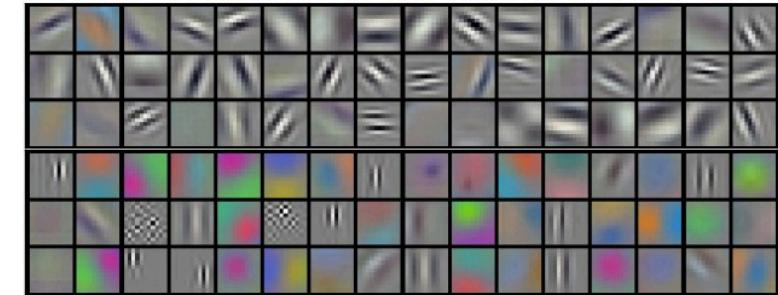
Parameter Sharing과 학습된 Filter(Kernel)들



For example, if we had 6 5×5 filters, we'll get 6 separate activation maps:



We stack these up to get a “new image” of size $28 \times 28 \times 6$!



Example filters learned by Krizhevsky et al. Each of the 96 filters shown here is of size $[11 \times 11 \times 3]$, and each one is shared by the 55×55 neurons in one depth slice. Notice that the parameter sharing assumption is relatively reasonable: If detecting a horizontal edge is important at some location in the image, it should intuitively be useful at some other location as well due to the translationally-invariant structure of images. There is therefore no need to relearn to detect a horizontal edge at every one of the 55×55 distinct locations in the Conv layer output volume.

Convolution Layer Summary

▣ $W_1 \times H_1 \times D$ 크기의 볼륨을 입력받는다.

▣ 4개의 hyperparameter가 필요하다:

1. 필터 개수 K

출력 볼륨의 가로.세로 크기

2. 필터의 가로/세로 Spatial 크기 F

$(W-F+2P)/S + 1$

3. Stride S

4. 제로 패딩 P .

▣ $W_2 \times H_2 \times D_2$ 크기의 출력 볼륨을 생성한다:

1. $W_2 = (W_1 - F + 2P) / S + 1$

2. $H_2 = (H_1 - F + 2P) / S + 1$ (i.e. 가로/세로는 같은 방식으로 계산됨)

3. $D_2 = K$

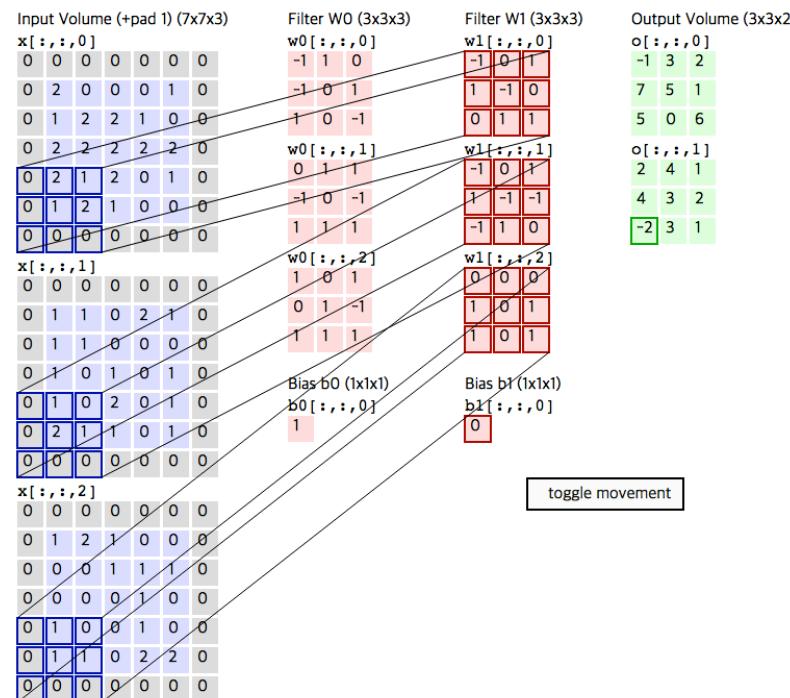
▣ 파라미터 sharing로 인해 필터당 $F \cdot F \cdot D$ 1개의 가중치를 가져서 총 $(F \cdot F \cdot D) \cdot K$ 개의 가중치와 K 개의 바이어스를 갖게 된다.

▣ 출력 볼륨에서 d 번째 depth slice ($W_2 \times H_2$ 크기)는 입력 볼륨에 d 번째 필터를 stride S 만큼 옮겨가며 컨볼루션 한 뒤 d 번째 바이어스를 더한 결과이다.

▣ 흔한 Hyperparameter 기본 세팅은 $F=3, S=1, P=1$ 이다.

Convolution Demo

- ▣ 컨볼루션 데모. 아래는 컨볼루션 레이어 데모이다. 3차원 볼륨은 시각화하기 힘드므로 각 행마다 depth slice를 하나씩 배치했다. 각 볼륨은 입력 볼륨(파란색), 가중치 볼륨(빨간색), 출력 볼륨(녹색)으로 이뤄진다. 입력 볼륨의 크기는 $W=5, H=5, D=3$ 이고 컨볼루션 레이어의 파라미터들은 $K=2, F=3, S=2, P=1$ 이다. 즉, 2개의 $3 \times 3 \times 3$ 크기의 필터가 각각 stride 2마다 적용된다. 그러므로 출력 볼륨의 spatial 크기 (가로/세로)는 $(5 - 3 + 2)/2 + 1 = 3$ 이다. 제로 패딩 $P=1$ 이 적용되어 입력 볼륨의 가장자리가 모두 0으로 되어있다는 것을 확인할 수 있다. 아래의 영상에서 하이라이트 표시된 입력(파란색)과 필터(빨간색)이 elementwise로 곱해진 뒤 하나로 더해지고 bias가 더해지는 걸 볼 수 있다.



출력 볼륨의 가로.세로 크기
 $(W-F+2P)/S + 1$

Pooling Layer

- ConvNet 구조 내에 컨볼루션 레이어들 중간중간에 주기적으로 풀링 레이어를 넣는 것이 일반적이다. 풀링 레이어가 하는 일은 네트워크의 파라미터의 개수나 연산량을 줄이기 위해 representation의 spatial한 사이즈를 줄이는 것이다. 이는 오버피팅을 조절하는 효과도 가지고 있다. 풀링 레이어는 MAX 연산을 각 depth slice에 대해 독립적으로 적용하여 spatial한 크기를 줄인다. 사이즈 2x2와 stride 2가 가장 많이 사용되는 풀링 레이어이다. 각 depth slice를 가로/세로축을 따라 1/2로 downsampling해 75%의 액티베이션은 버리게 된다. 이 경우 MAX 연산은 4개 숫자 중 최대값을 선택하게 된다 (같은 depth slice 내의 2x2 영역). Depth 차원은 변하지 않는다. 풀링 레이어의 특징들은 일반적으로 아래와 같다:

- $W_1 \times H_1 \times D_1$ 사이즈의 입력을 받는다

- 2가지 hyperparameter를 필요로 한다.

1. Spatial extent F

2. Stride S

- $W_2 \times H_2 \times D_2$ 사이즈의 볼륨을 만든다.

1. $W_2 = (W_1 - F) / S + 1$

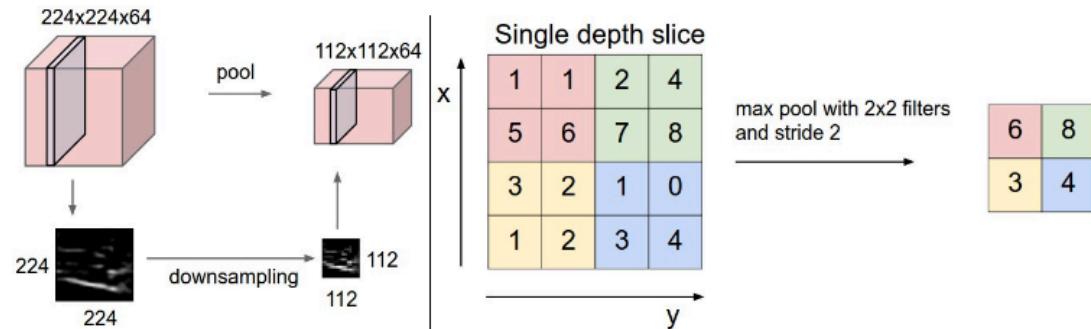
2. $H_2 = (H_1 - F) / S + 1$

3. $D_2 = D_1$

- 입력에 대해 항상 같은 연산을 하므로 파라미터는 따로 존재하지 않는다. 풀링 레이어에는 보통 제로 패딩을 하지 않는다.

- 일반적으로 실전에서는 두 종류의 max 풀링 레이어만 널리 쓰인다. 하나는 overlapping 풀링이라고도 불리는 $F=3, S=2$ 이고 하나는 더 자주 쓰이는 $F=2, S=2$ 이다. 큰 리셉티브 필드에 대해서 풀링을 하면 보통 너무 많은 정보를 버리게 된다.

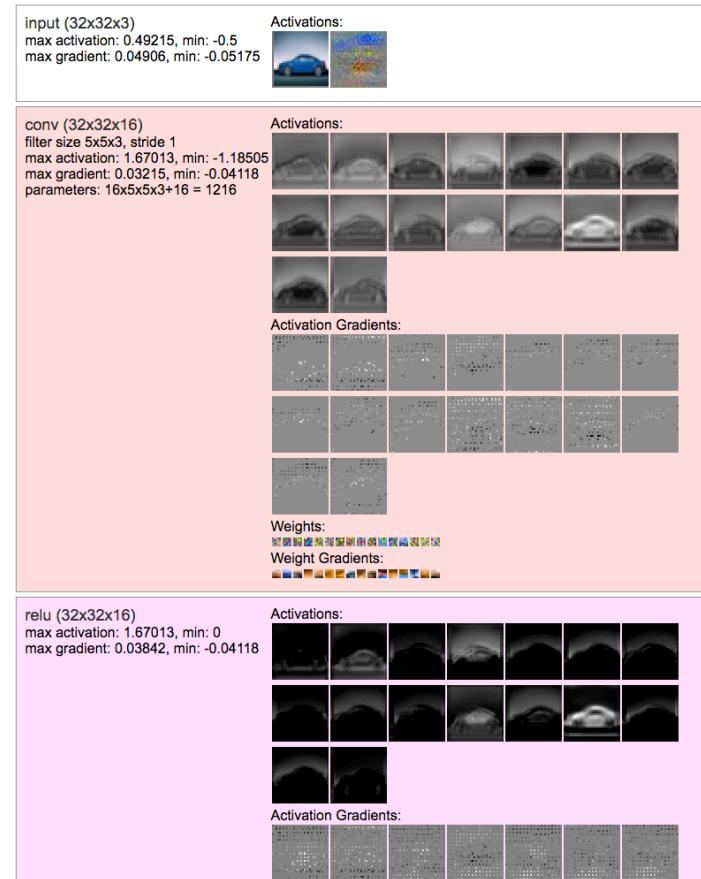
- 일반적인 풀링.** Max 풀링 뿐 아니라 average 풀링, L2-norm 풀링 등 다른 연산으로 풀링할 수도 있다. Average 풀링은 과거에 많이 쓰였으나 최근에는 Max 풀링이 더 좋은 성능을 보이며 점차 쓰이지 않고 있다.



Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. Left: In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. Right: The most common downsampling operation is max, giving rise to max pooling, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

ConvNetJS Demo

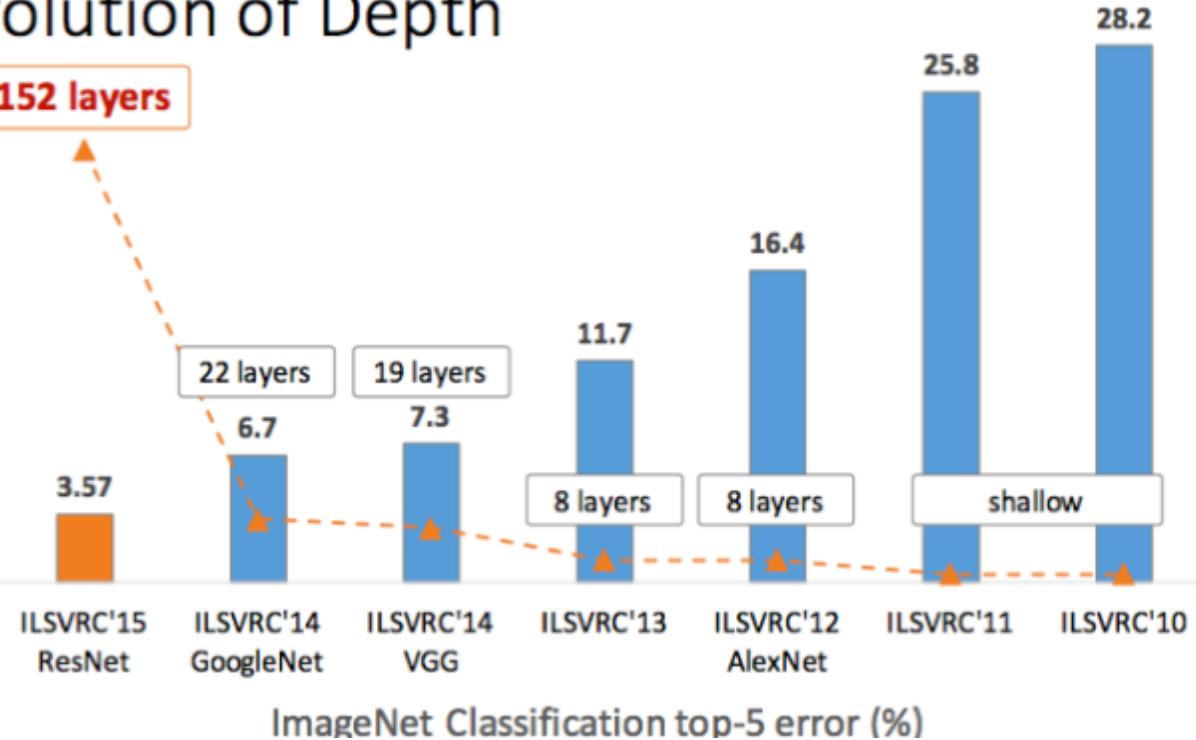
❑ <http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>



ILSVRC에서 제안된 대표적인 CNNs

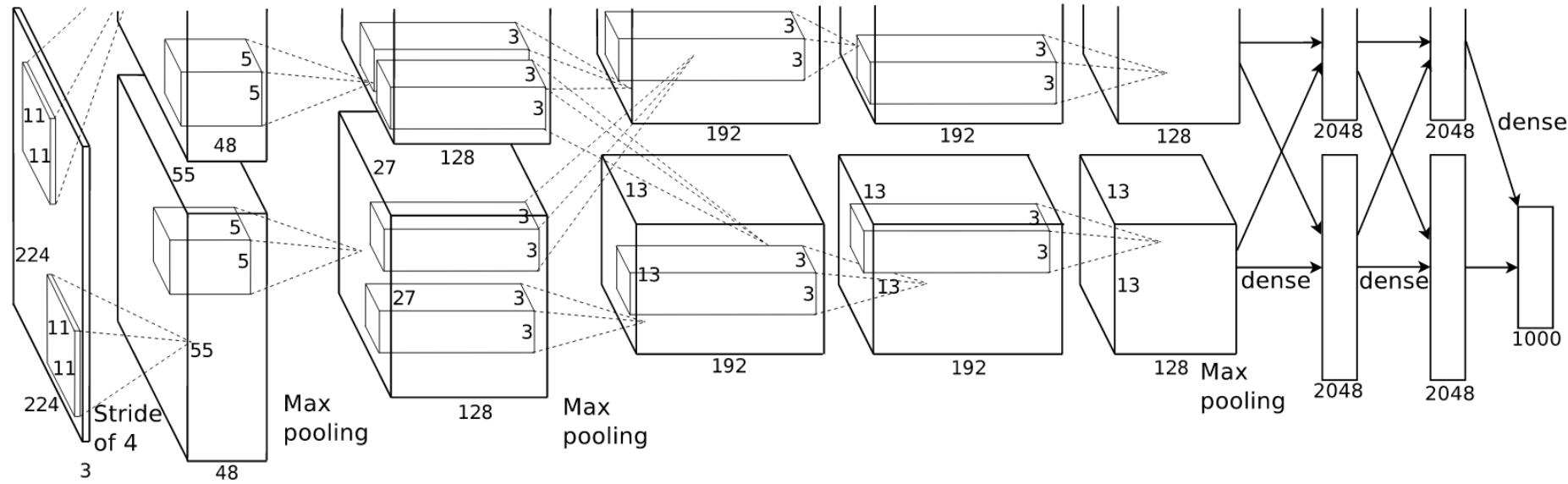
- 연도별 ILSVRC 우승 모델들의 Layer Depth들

Revolution of Depth



AlexNet (2012년)

- ▣ 2012년 LeNet의 Scale을 키운 AlexNet을 제안
- ▣ ILSVRC-2012 1등



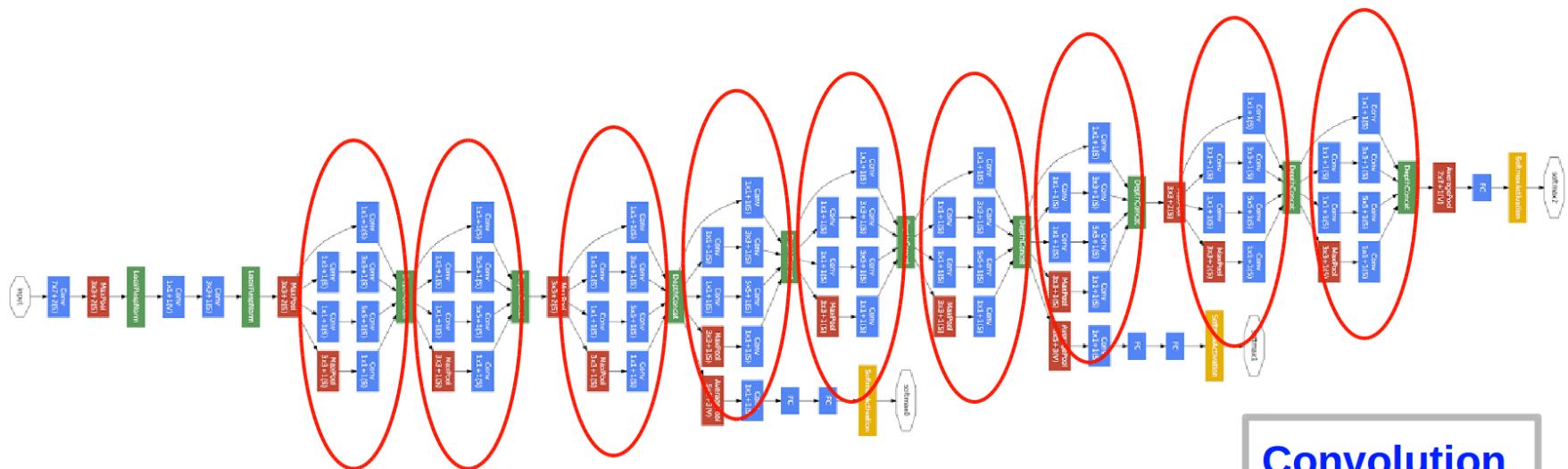
VGGnet (2014년)

▣ ILSVRC-2014 2등

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

GoogLeNet-Inception v1- (2014년)

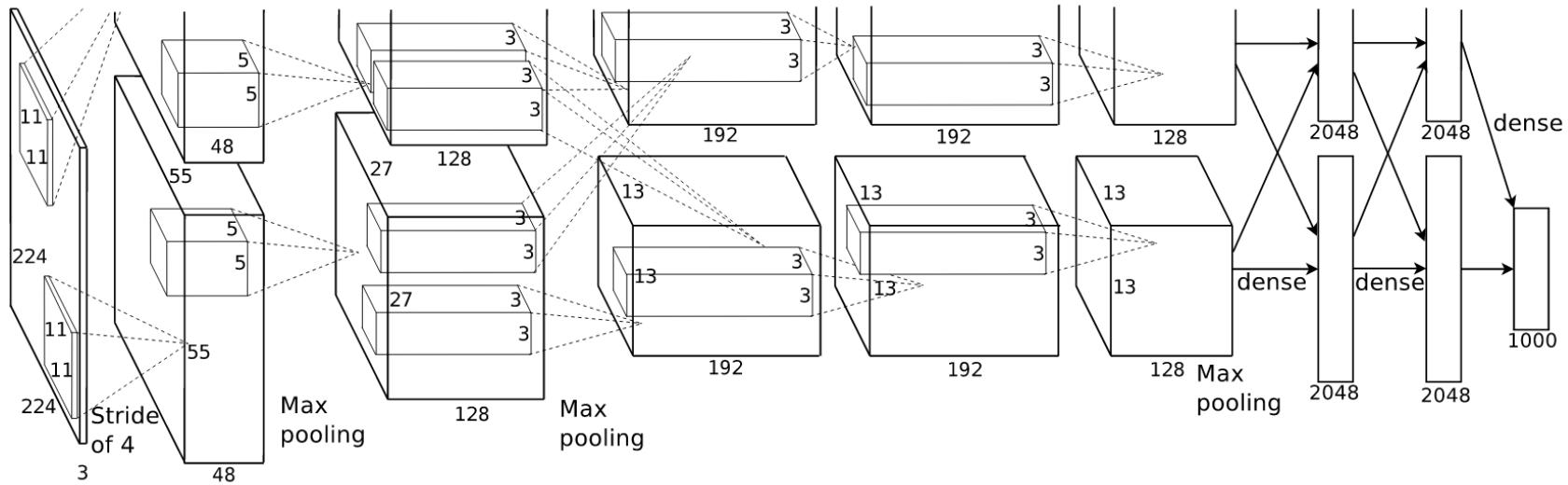
■ ILSVRC-2014 1등



Convolution
Pooling
Softmax
Concat/Normalize

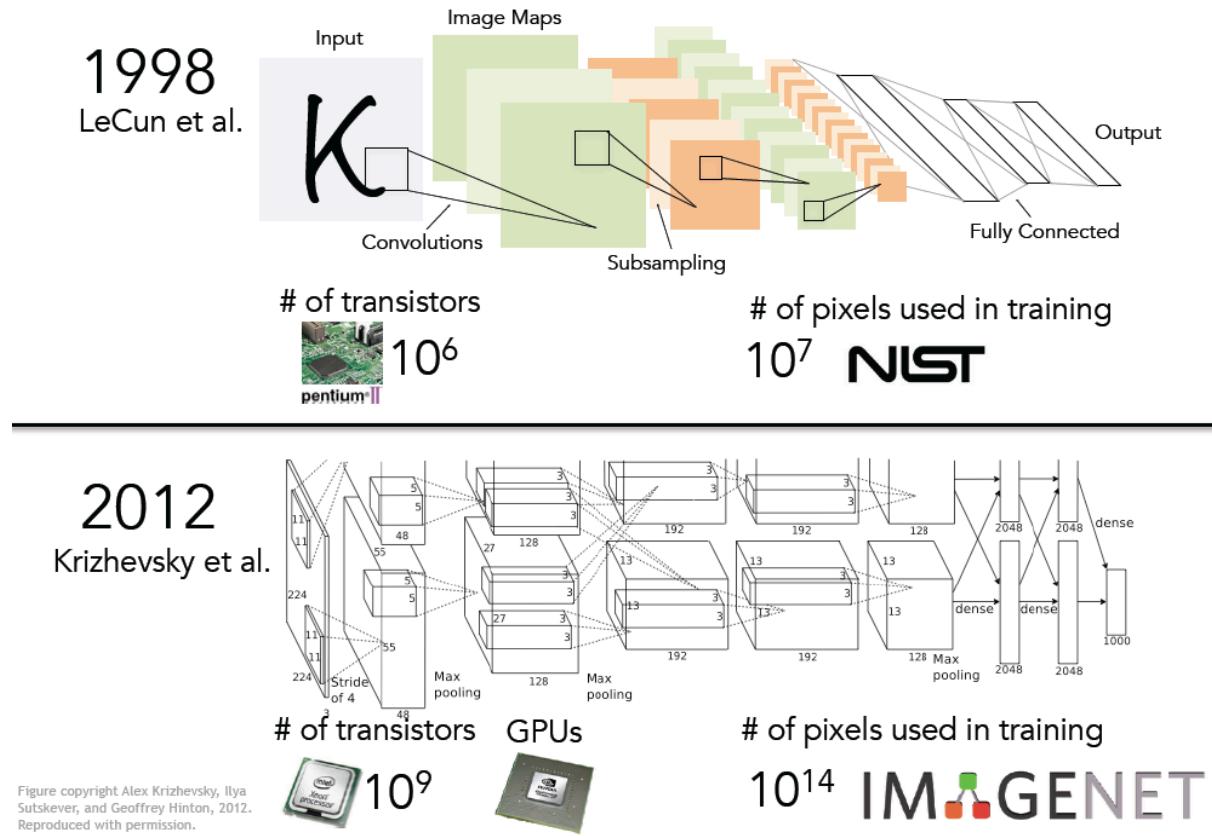
논문 리뷰 - ImageNet Classification with Deep Convolutional Neural Networks

- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks.", NIPS 2012.
- **핵심 아이디어** : GPU를 이용한 Deep Convolutional Neural Networks(AlexNet)를 이용해서 Image Classification을 수행
- <https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>



Deep Convolutional Neural Networks(AlexNet)

- ▣ 아이디어 자체는 1998년 LeNet과 크게 다르지 않음



ImageNet Large-Scale Visual Recognition Challenge(ILSVRC)

- ImageNet Large-Scale Visual Recognition Challenge(ILSVRC)
- ILSVRC는 ImageNet의 데이터베이스의 일부분을 사용한다. 1000개의 카테고리를 가지고 있고 120만장의 training images과 5만장의 validation images, 15만장의 testing images로 구성되어 있다.

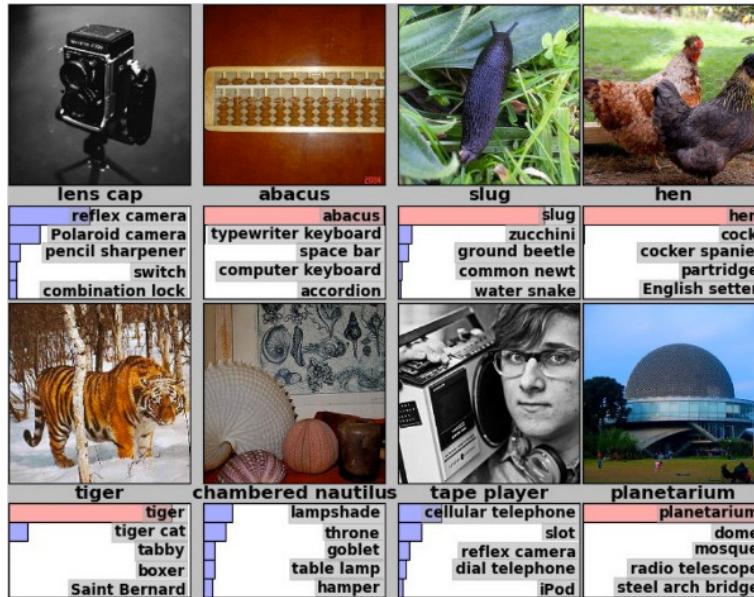


ImageNet Large Scale Visual Recognition Challenges



ImageNet Large-Scale Visual Recognition Challenge(ILSVRC)-2012

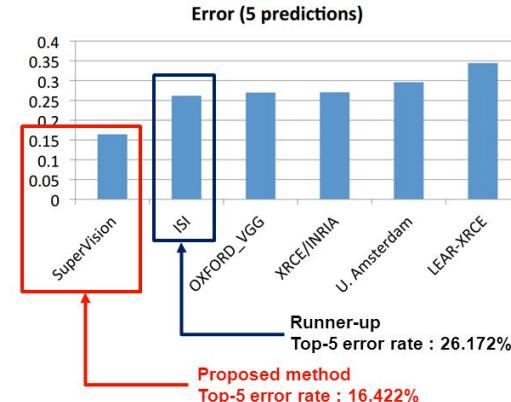
- 대회 성능 측정 방식 : Top-1 & Top-5
- ILSVRC-2012 대회에서 AlexNet(SuperVision)은 다른 방법들을 큰 격차로 따돌리고 우승



reference : <http://image-net.org/challenges/LSVRC/2012/ilsvrc2012.pdf>

Results

• ILSVRC-2012 results



16

KAIST

Model Architecture

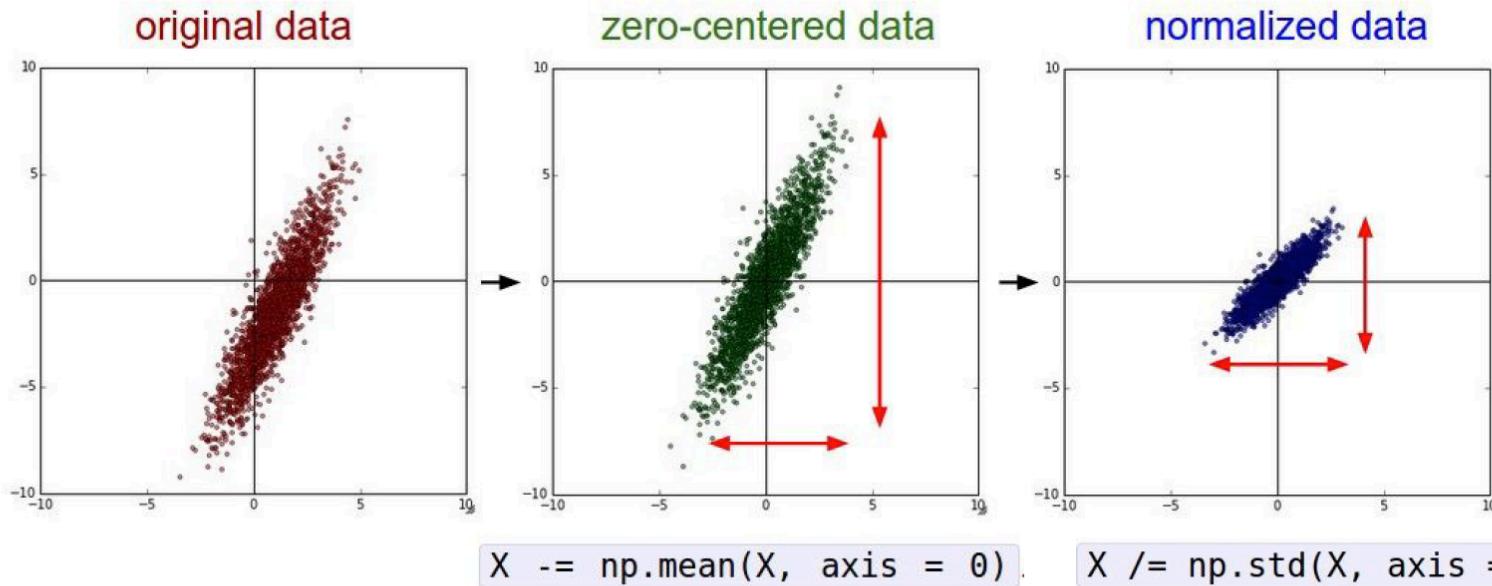
- ▣ **Input** : 평균값을 뺀 256x256 크기의 RGB

이미지 (ImageNet 데이터베이스의 이미지 크기는 다양하기 때문에 짧은 부분을 256으로 맞추고 남은 부분은 Crop함)

- ▣ **Output** : 1000개의 Label (e.g. cat, tiger, hen, ...)에 대한 확률

Data Preprocessing

Step 1: Preprocess the data

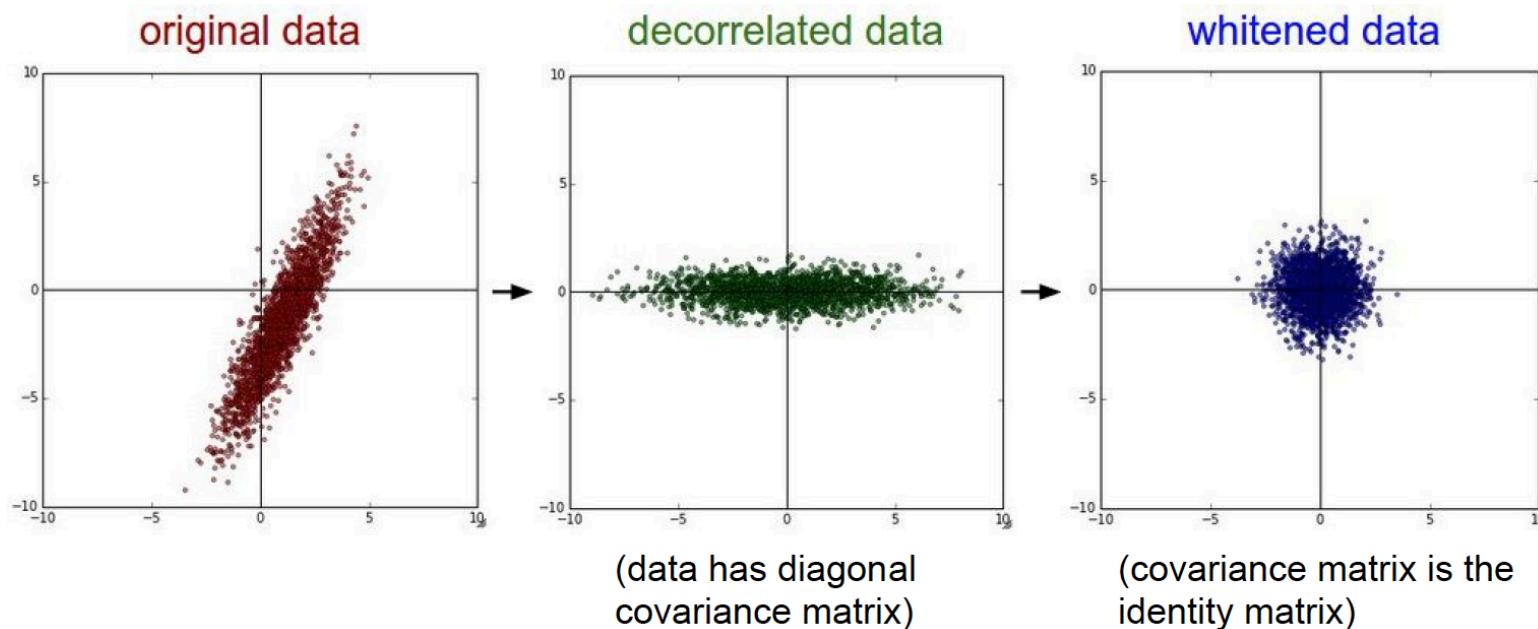


(Assume X [NxD] is data matrix,
each example in a row)

Data Preprocessing

Step 1: Preprocess the data

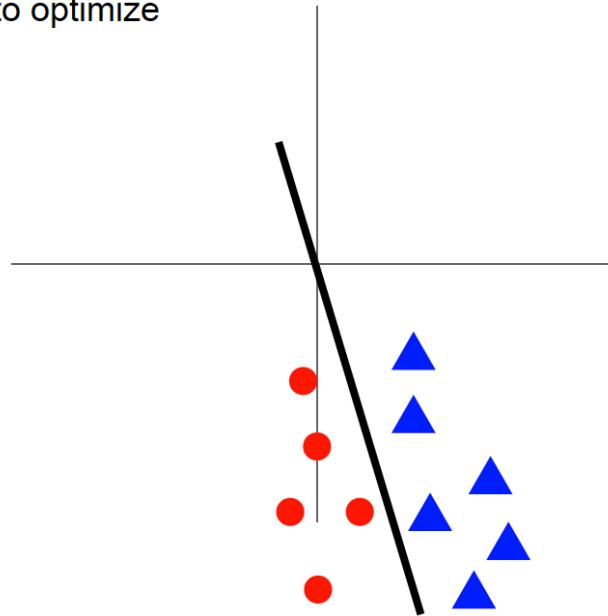
In practice, you may also see **PCA** and **Whitening** of the data



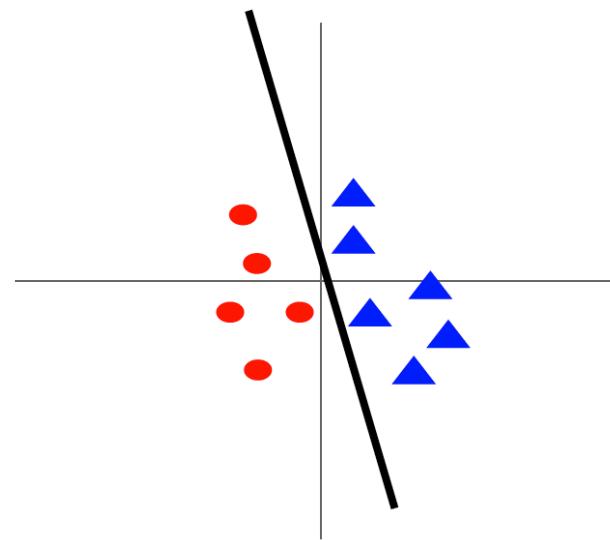
Data Preprocessing의 장점

Last time: Data Preprocessing

Before normalization: classification loss very sensitive to changes in weight matrix; hard to optimize



After normalization: less sensitive to small changes in weights; easier to optimize



Data Preprocessing In Practice

TLDR: In practice for Images: center only

e.g. consider CIFAR-10 example with [32,32,3] images

- Subtract the mean image (e.g. AlexNet)
(mean image = [32,32,3] array)
- Subtract per-channel mean (e.g. VGGNet)
(mean along each channel = 3 numbers)

Not common to normalize variance, to do PCA or whitening

새로운 기법 1,2: ReLU NonLinearity, Multiple-GPU

- ▣ 새로운 기법 1,2: ReLU NonLinearity, Multiple-GPU(2개의 GPU를 사용)
- ▣ ReLU NonLinearity : Overfitting을 방지하고 학습속도가 훨씬 빨라짐(아래 실험의 경우 약 6배가 더 빠름)

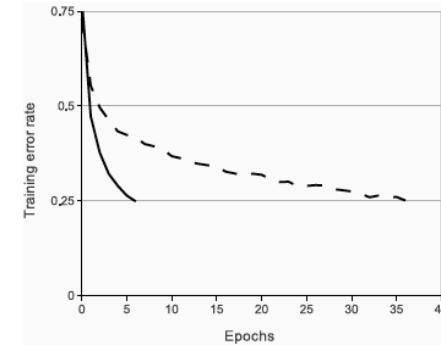
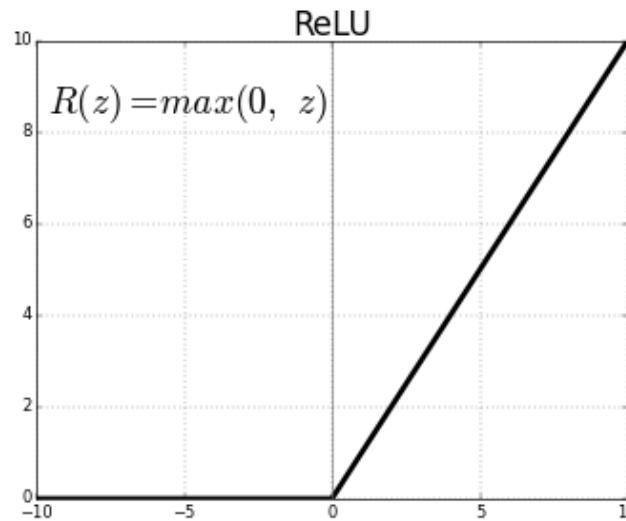
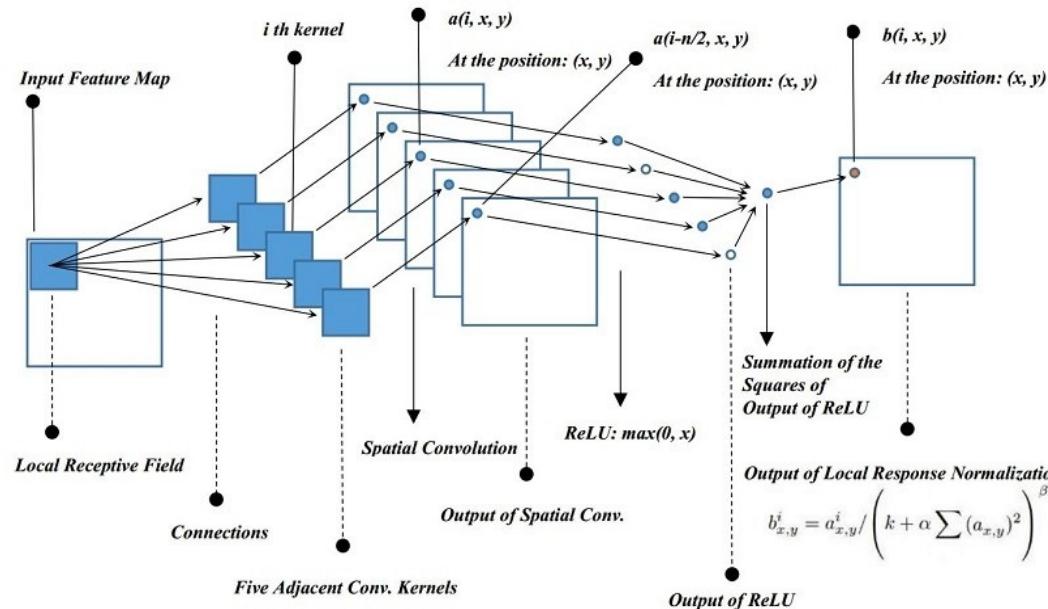


Figure 1: A four-layer convolutional neural network with ReLUs (solid line) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (dashed line). The learning rates for each network were chosen independently to make training as fast as possible. No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

새로운 기법 3: Local Response Normalization

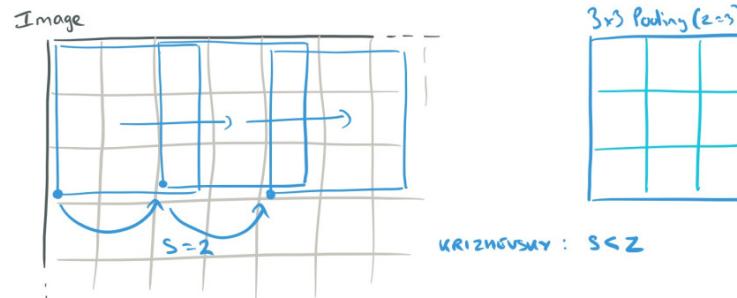
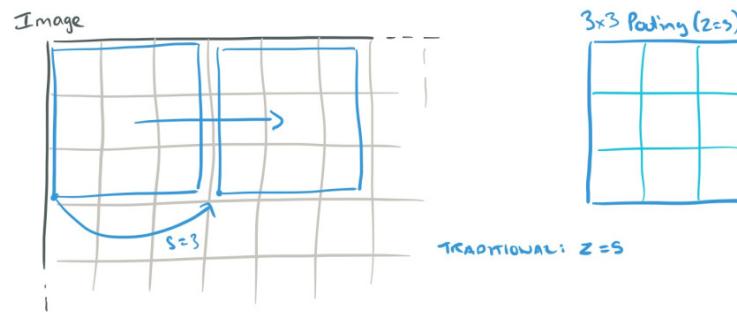
- ▣ 새로운 기법 3: Local Response Normalization (generalization을 도와줌-강한 뉴런이 약한 뉴런의 값을 막는 현상을 억제함)
- ▣ Local Response Normalization: top-1과 top-5 에러율을 각각 1.4%, 1.2% 줄임.

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



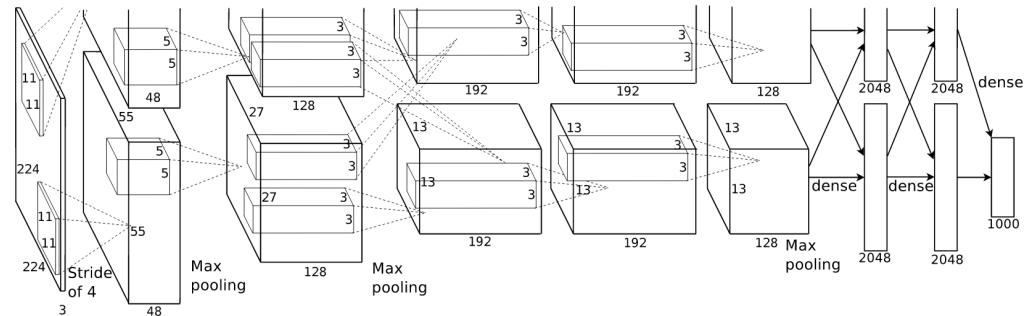
새로운 기법 4 : Overlapping Pooling

- ▣ 새로운 기법 4: Overlapping Pooling
- ▣ Overlapping Pooling: 기존에는 stride와 filter size의 크기를 같게 하여($s=z$) pooling할 때 overlapping이 일어나지 않게 사용함. 하지만, 이 논문에서는 $s=2$ (stride), $z=3$ (Filter size)을 사용하여 pooling을 overlapping함
- ▣ top-1과 top-5 에러율을 각각 0.4%, 0.3% 줄임.



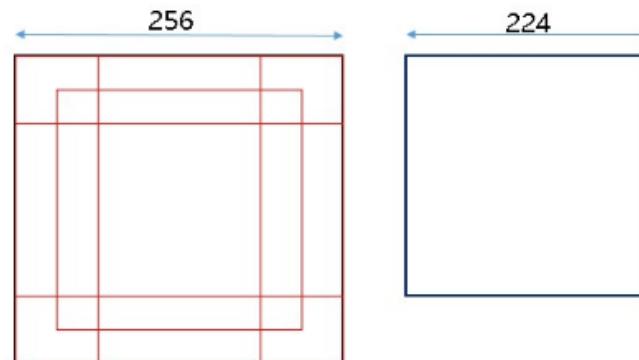
Overall Architecture Summary

- ▣ 2번째, 4번째, 5번째 컨볼루션 레이어와 각각의 다음 레이어는 같은 GPU에서 연결되어 있다.
- ▣ 3번째 컨볼루션 레이어에서는 모든 커널이 4번째 레이어와 연결되어 있다. 풀리-커넥티드 레이어는 이전 레이어와 모두 연결되어 있다. ReLU 비선형 레이어는 모든 convolutional 레이어와 풀리-커넥티드 레이어의 output에 적용한다.
- ▣ [227x227x3] INPUT (논문에서는 224x224로 잘못표현 됨-Zero-Padding 3이 미리 추가된걸로 추측-)
[55x55x96] CONV1 : 96@ 11x11, s = 4, p = 0
[27x27x96] MAX POOL1 : 3x3, s = 2
[27x27x96] NORM1 :
[27x27x256] CONV2 : 256@ 5x5, s = 1, p = 0
[13x13x256] MAX POOL2 : 3x3, s = 2
[13x13x256] NORM2 :
[13x13x384] CONV3 : 384@ 3x3, s = 1, p = 0
[13x13x384] CONV4 : 384@ 3x3, s = 1, p = 0
[13x13x256] CONV5 : 256@ 3x3, s = 1, p = 0
[6x6x256] MAX POOL3 : 3x3, s = 2
[4096] FC6 : 4096 neurons
[4096] FC7 : 4096 neurons
[1000] FC8 : 1000 neurons



Overfitting 방지 기법 1 – Data Augmentation

- 1. 이미지 개수 증가 - 원본 이미지 크기 256x256에서 224x224 사이즈의 패치를 랜덤하게 추출한다. 상하 대칭으로도 이미지를 똑같은 방법으로 추출한다. 이렇게 추출하면 한 개의 원본 이미지로 $2048 - (256-224) * (256-224) * 2 = 2048$ -가지의 경우의 수가 나온다. 테스트 단계에서는 원본, 상하반전 이미지에서 5개씩 224x224 패치를 추출하여 softmax의 평균을 내어 추측한다. 이 때 5개의 패치는 4개의 코너와 중앙에서 추출한다.
- 2. 이미지 RGB 값 변화 - PCA를 통해 테스트 이미지의 RGB 채널 강도를 변화시키는 augmentation을 진행한다.



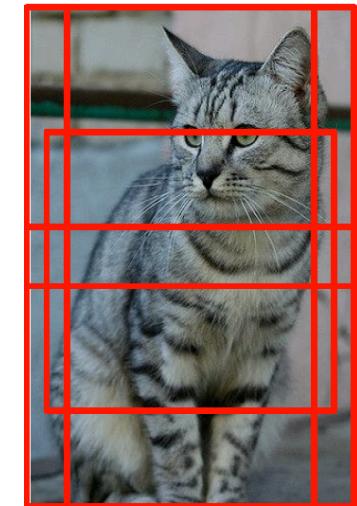
Overfitting 방지 기법 1 – Data Augmentation

Data Augmentation Random crops and scales

Training: sample random crops / scales

ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224×224 patch



Testing: average a fixed set of crops

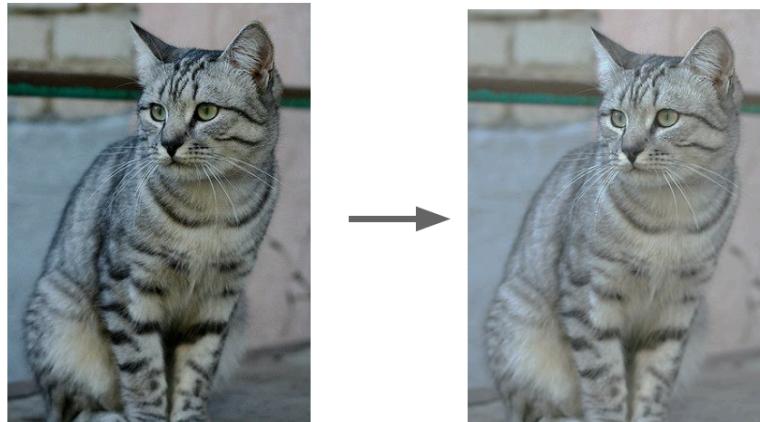
ResNet:

1. Resize image at 5 scales: {224, 256, 384, 480, 640}
2. For each size, use 10 224×224 crops: 4 corners + center, + flips

Overfitting 방지 기법 1 – Data Augmentation

Data Augmentation Color Jitter

Simple: Randomize contrast and brightness



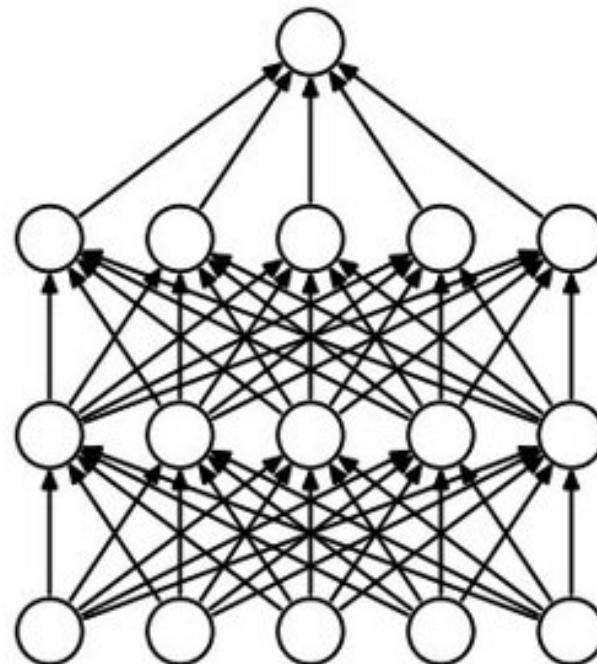
More Complex:

1. Apply PCA to all [R, G, B] pixels in training set
2. Sample a “color offset” along principal component directions
3. Add offset to all pixels of a training image

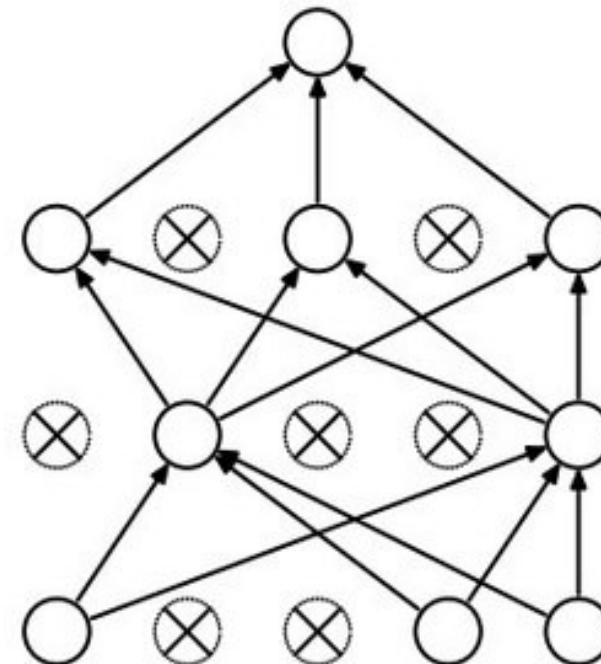
(As seen in [Krizhevsky et al. 2012], ResNet, etc)

Overfitting 방지 기법 2 – DropOut

- DropOut을 통해 Overfitting을 방지.



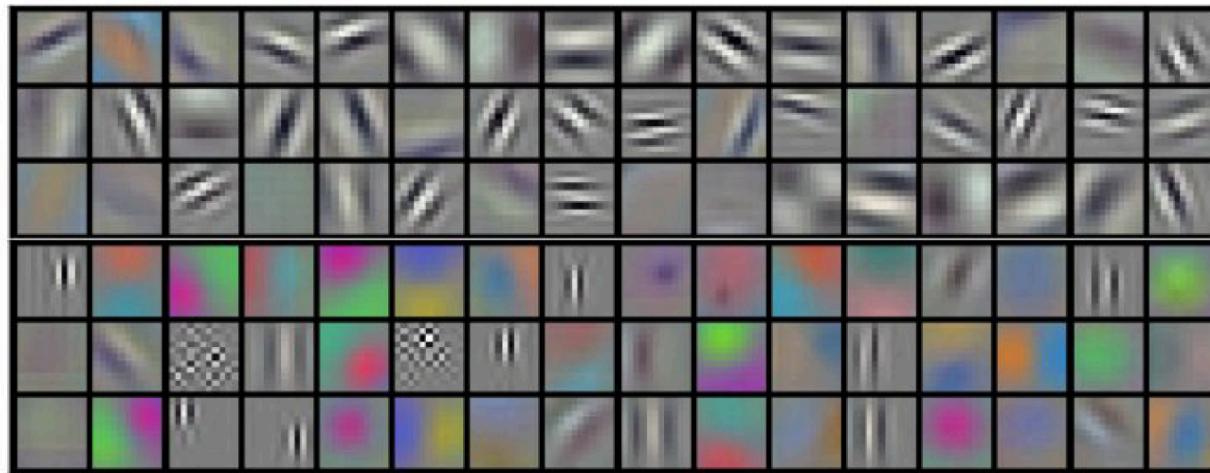
(a) Standard Neural Net



(b) After applying dropout.

학습된 필터들

- ▣ 트레이닝 결과로 학습된 필터들
- ▣ GPU1의 커널은 대부분 컬러와 상관없으며, GPU2의 커널은 대부분 컬러와 관련이 있다. 이러한 특수성은 모든 동작에서 발생하며, 랜덤하게 초기 가중치를 설정하는 것과는 독립적이다.



Experiment Result 1

- ▣ 다른 방법들과의 성능 비교
- ▣ ILSVRC-2011 데이터로 pre-training한 뒤에 ILSVRC-2012 데이터로 Fine-Tuning한 경우 더 좋은 성능을 얻을 수 있었다.

Model	Top-1	Top-5
<i>Sparse coding [2]</i>	47.1%	28.2%
<i>SIFT + FVs [24]</i>	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.

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%	16.4%
1 CNN*	39.0%	16.6%	—
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.

Experiment Result 2

■ 실험결과

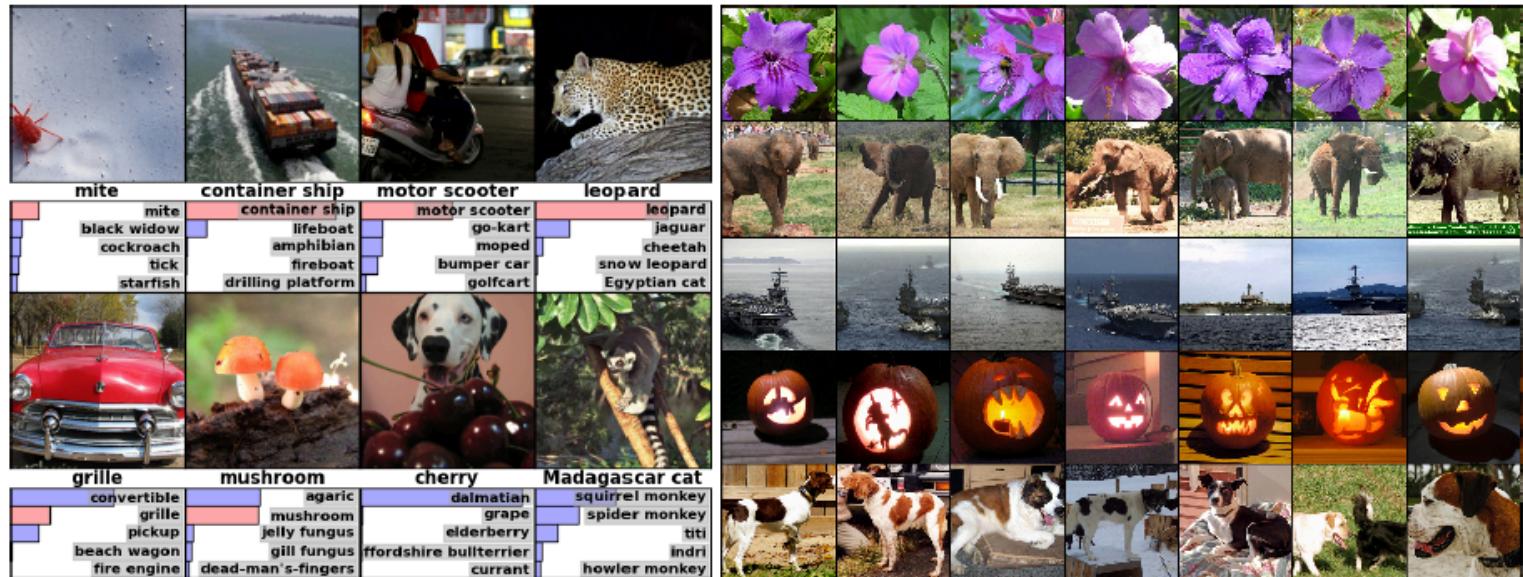


Figure 4: (Left) Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). (Right) Five ILSVRC-2010 test images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

논문 리뷰 - VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

- Simonyan, Karen, and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition.", ICLR 2015
- 핵심 아이디어** : 3x3 Convolution을 이용한 Deep CNNs을 제안, 깊이(Depth)가 성능에 중요한 영향을 끼치는 것을 실험적으로 증명함. **ILSVRC-2014 대회**에서 GoogLeNet에 이어 **2등**을 차지함, 하지만 GoogLeNet-Inception v1-에 비해 훨씬 간단한 구조로 인해 더욱 널리 사용됨
- VGGNet이란 이름은 저자들이 속한 Oxford 대학의 Visual Geometry Group에서 따옴
- <https://arxiv.org/pdf/1409.1556.pdf>

ConvNet Configuration						
A	A-LRN	B	C	D	E	
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers	
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	
			maxpool			
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	
			maxpool			
conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv1-256	conv3-256 conv3-256	conv3-256 conv3-256
			maxpool			
conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv1-512	conv3-512 conv3-512	conv3-512 conv3-512
			maxpool			
conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv1-512	conv3-512 conv3-512	conv3-512 conv3-512
			maxpool			
FC-4096						
FC-4096						
FC-1000						
						soft-max

Model Architecture

- ▣ A, A-LRN, B, C, D, E 5가지 모델(11 layers->19 layers)에 대해 실험을 진행함
- ▣ 깊이가 깊어질수록 성능이 좋아짐
- ▣ 실험결과 Local Response Normalization(LRN)은 성능에 큰 영향을 주지 않는다는 사실을 발견 함

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: Number of parameters (in millions).

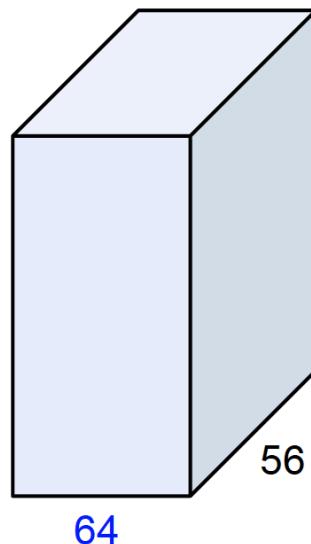
Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

1x1 Convolution의 사용

▣ 1x1 Convolution의 장점

1. Dimension Reduction
2. More ReLU(More Non-Linearity)

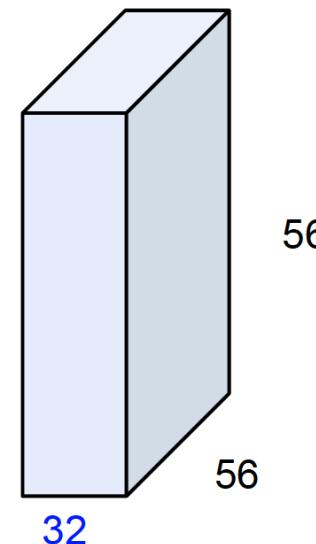
Reminder: 1x1 convolutions



1x1 CONV
with 32 filters

preserves spatial
dimensions, reduces depth!

Projects depth to lower
dimension (combination of
feature maps)



Experiment Result

▣ 모델들의 성능 비교

1. 깊이(Depth)가 깊을수록 성능이 좋아짐
2. Local Response Normalization(LRN)은 성능에 별 영향을 주지 못해서 제거함

ConvNet Configuration						
A	A-LRN	B	C	D	E	
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers	
input (224×224 RGB image)						
conv3-64	conv3-64	conv3-64	conv3-64	conv3-64	conv3-64	conv3-64
LRN		conv3-64	conv3-64	conv3-64	conv3-64	
maxpool						
conv3-128	conv3-128	conv3-128	conv3-128	conv3-128	conv3-128	conv3-128
	conv3-128	conv3-128	conv3-128	conv3-128	conv3-128	conv3-128
maxpool						
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
maxpool						
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
maxpool						
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
maxpool						
FC-4096						
FC-4096						
FC-1000						
soft-max						

Table 3: ConvNet performance at a single test scale.

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
	256	256	28.1	9.4
C	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
	256	256	27.0	8.8
D	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
	256	256	27.3	9.0
E	384	384	26.9	8.7
	[256;512]	384	25.5	8.0

Overall Architecture Summary

INPUT: [224x224x3] memory: $224 \times 224 \times 3 = 150\text{K}$ params: 0 (not counting biases)

CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2\text{M}$ params: $(3 \times 3 \times 3) \times 64 = 1,728$

CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2\text{M}$ params: $(3 \times 3 \times 64) \times 64 = 36,864$

POOL2: [112x112x64] memory: $112 \times 112 \times 64 = 800\text{K}$ params: 0

CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6\text{M}$ params: $(3 \times 3 \times 64) \times 128 = 73,728$

CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6\text{M}$ params: $(3 \times 3 \times 128) \times 128 = 147,456$

POOL2: [56x56x128] memory: $56 \times 56 \times 128 = 400\text{K}$ params: 0

CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800\text{K}$ params: $(3 \times 3 \times 128) \times 256 = 294,912$

CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800\text{K}$ params: $(3 \times 3 \times 256) \times 256 = 589,824$

CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800\text{K}$ params: $(3 \times 3 \times 256) \times 256 = 589,824$

POOL2: [28x28x256] memory: $28 \times 28 \times 256 = 200\text{K}$ params: 0

CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400\text{K}$ params: $(3 \times 3 \times 256) \times 512 = 1,179,648$

CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400\text{K}$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400\text{K}$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

POOL2: [14x14x512] memory: $14 \times 14 \times 512 = 100\text{K}$ params: 0

CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100\text{K}$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100\text{K}$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100\text{K}$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

POOL2: [7x7x512] memory: $7 \times 7 \times 512 = 25\text{K}$ params: 0

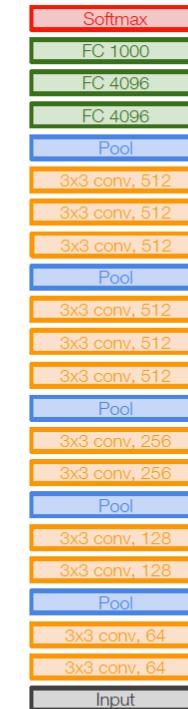
FC: [1x1x4096] memory: 4096 params: $7 \times 7 \times 512 \times 4096 = 102,760,448$

FC: [1x1x4096] memory: 4096 params: $4096 \times 4096 = 16,777,216$

FC: [1x1x1000] memory: 1000 params: $4096 \times 1000 = 4,096,000$

TOTAL memory: $24\text{M} * 4 \text{ bytes} \approx 96\text{MB / image}$ (only forward! ~ 2 for bwd)

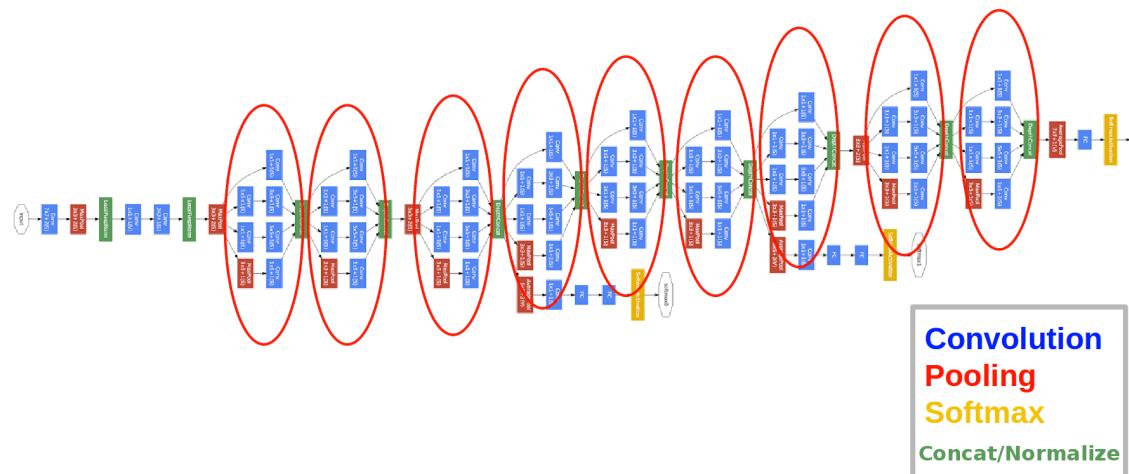
TOTAL params: 138M parameters



VGG16

논문 리뷰 - Going Deeper with Convolutions

- Szegedy, Christian, et al, "Going deeper with convolutions.", CVPR 2015.
- **핵심 아이디어** : LeNet에 기반해서 Google이 제안한 CNNs 모델인 GoogLeNet을 제안, AlexNet에 비해 **12배나 적은 parameter**로 더욱 좋은 성능을 보임. **ILSVRC-2014 대회**에서 **1등**을 차지함.
- Inception v1 모델을 제안
- <https://arxiv.org/pdf/1409.4842.pdf>



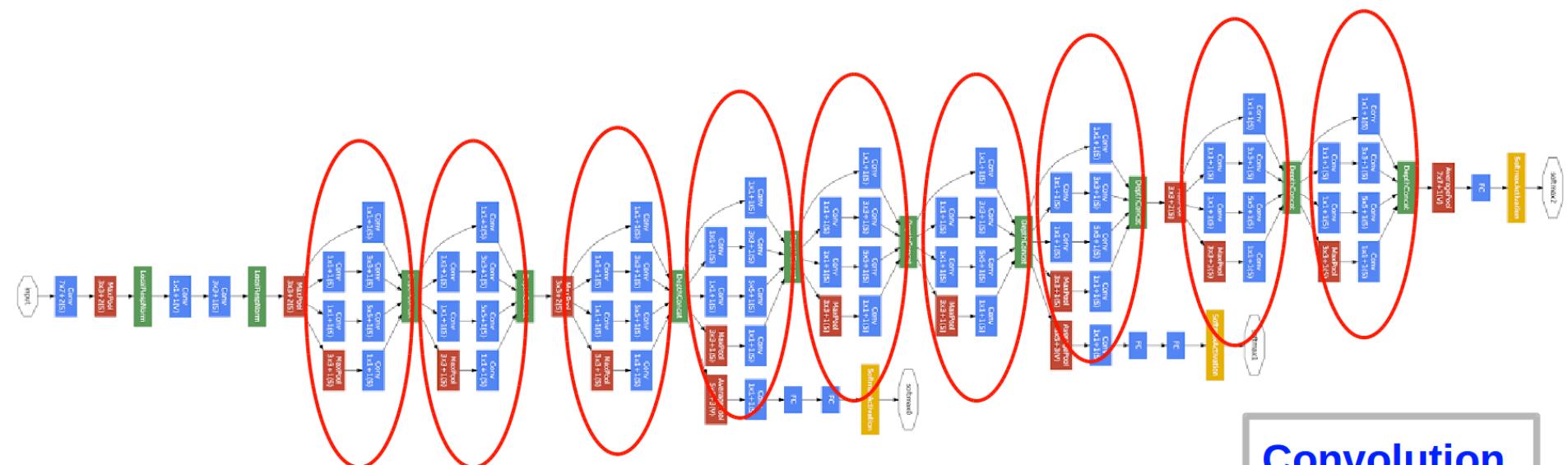
GoogLeNet 핵심 아이디어

- ▣ 최대한 파라미터를 줄이면서 네트워크를 깊고 넓게 디자인하고자 함. (레이어가 깊더라도 연결이 Sparse하다면 파라미터수가 줄어듬.) 이는 Overfitting을 방지하는 효과를 얻을 수 있다.
- ▣ 하지만, 연산은 Dense하게 처리하는 것을 목표로 함
- ▣ 딥러닝 계산은 결국 매트릭스 곱으로 처리 되는데 이때 매트릭스의 값이 Sparse하다면(예를 들어, 100x100 매트릭스에 1개의 값만 있고 나머지는 모두 0이라면) 낭비되는 연산이 많아짐. 따라서 **매트릭스 연산을 할때 데이터가 Sparse해지는 현상을 방지하는 것을 목표로 한다.**
- ▣ 이를 위해, 영화 Inception에서 이름을 따온 Inception Module을 제안
- ▣ ILSVRC-2014에서 1등을 차지한 GoogLeNet은 Inception 모델의 여러 구현 중에 하나이다.



Model Architecture

- GoogLeNet의 전체 모델 훑어보기



Convolution
Pooling
Softmax
Concat/Normalize

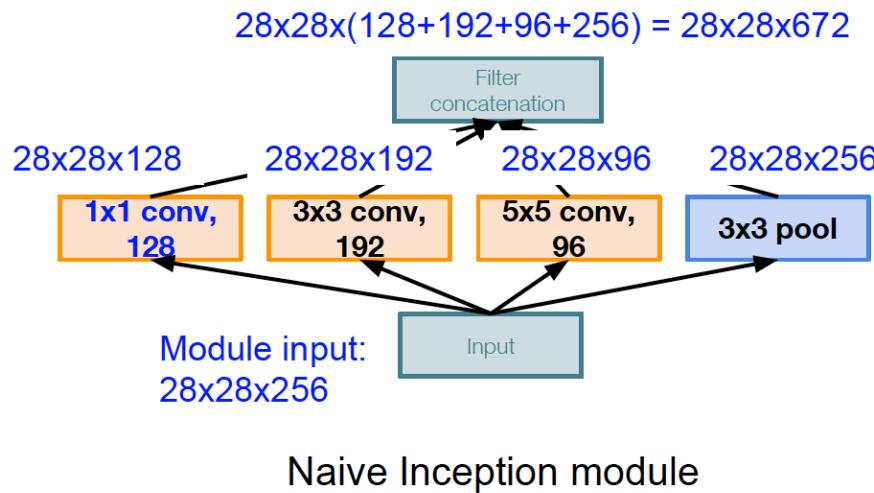
Inception Module

Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q3: What is output size after filter concatenation?



Q: What is the problem with this?
[Hint: Computational complexity]

Conv Ops:

[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256

Total: 854M ops

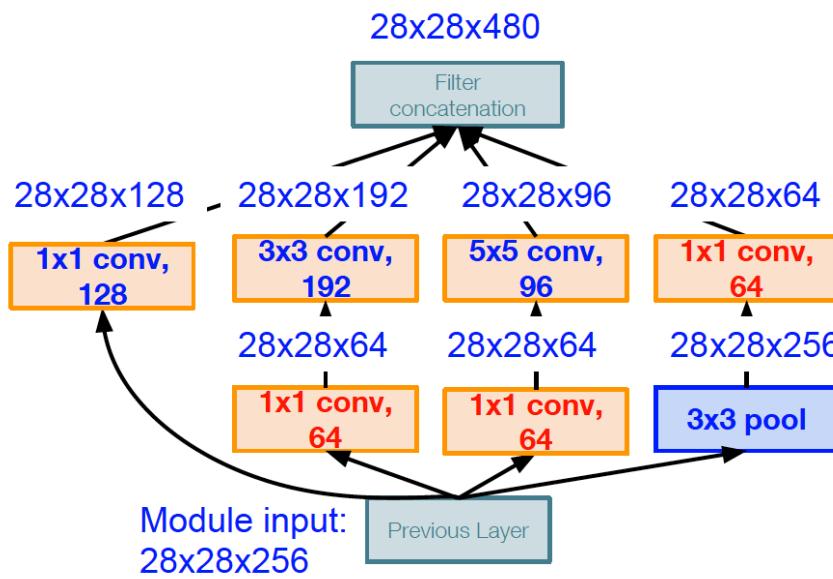
Very expensive compute

Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer!

Inception Module with Dimension Reductions

Case Study: GoogLeNet

[Szegedy et al., 2014]



Inception module with dimension reduction

Using same parallel layers as naive example, and adding “1x1 conv, 64 filter” bottlenecks:

Conv Ops:

- [1x1 conv, 64] 28x28x64x1x1x256
- [1x1 conv, 64] 28x28x64x1x1x256
- [1x1 conv, 128] 28x28x128x1x1x256
- [3x3 conv, 192] 28x28x192x3x3x64
- [5x5 conv, 96] 28x28x96x5x5x64
- [1x1 conv, 64] 28x28x64x1x1x256

Total: 358M ops

Compared to 854M ops for naive version
Bottleneck can also reduce depth after pooling layer

GoogLeNet Model Architecture Summary

- 입력 이미지 : 224x224x3 크기의 이미지 (Mean Subtracted)

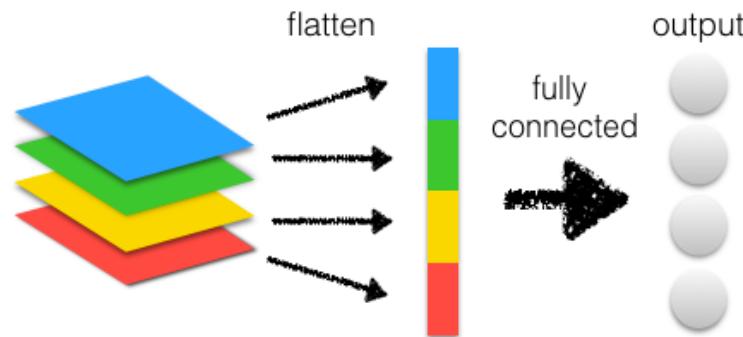
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Average Pooling Instead Fully Connected Layers

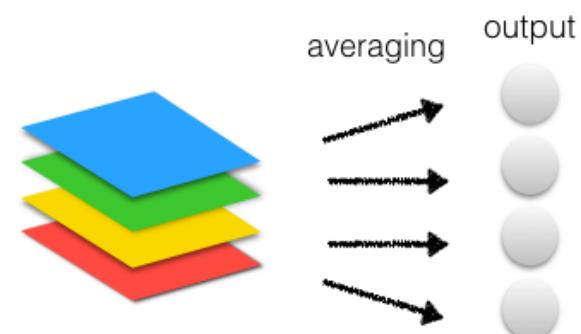
■ Global Average Pooling의 장점

1. 학습해야할 파라미터 개수가 줄어듬에 따라 Overfitting의 가능성이 낮아짐
2. Spatial 정보의 평균을 취하므로 Input의 translation에 강건하다.

Fully Connected Layer

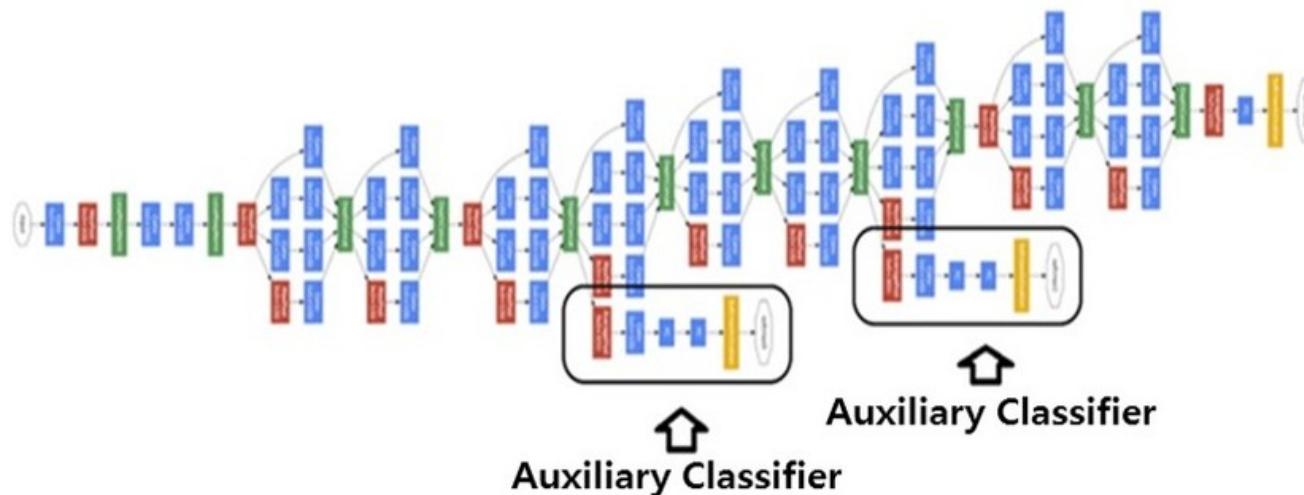


Global Average Pooling



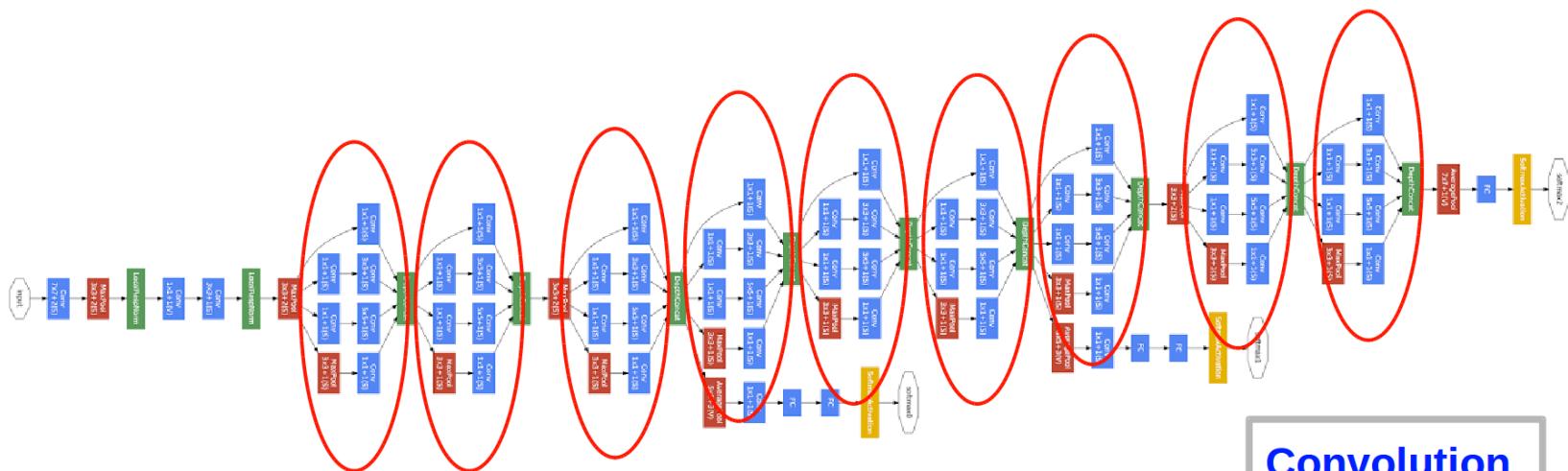
Auxiliary Classifier

- 레이어가 깊어지면서 발생할 수 있는 Vanishing Gradient Problem을 방지하고자 Auxiliary Classifier를 추가
- Test시에는 이를 사용하지 않는다.



Model Architecture

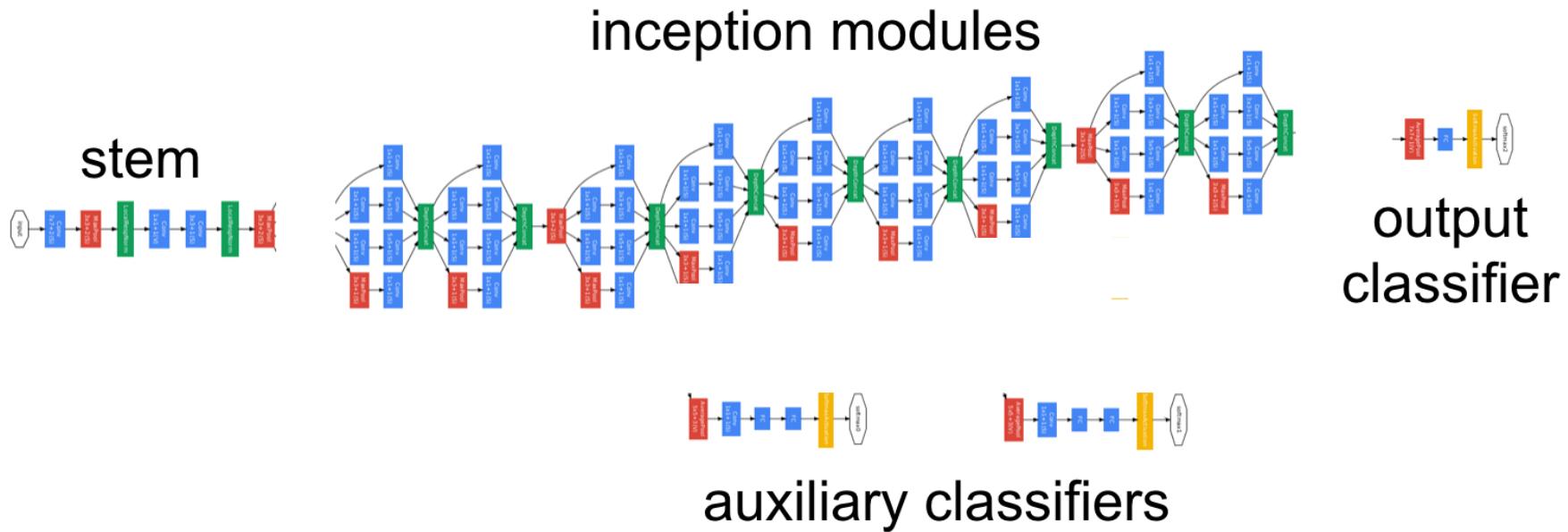
■ 전체 모델을 다시 한번 Remind



Convolution
Pooling
Softmax
Concat/Normalize

Model Architecture

- 전체 모델을 부분별로 살펴보면 아래와 같다.



Experiment Result

- 대회에서는 GoogLeNet 7개를 Ensemble Learning 해서 제출 함
- Specifically, we resize the image to 4 scales where the shorter dimension (height or width) is 256, 288, 320 and 352 respectively, take the left, center and right square of these resized images (in the case of portrait images, we take the top, center and bottom squares).
- For each square, we then take the 4 corners and the center 224x224 crop as well as the square resized to 224x224 , and their mirrored versions. This results in $4 \times 3 \times 6 \times 2 = 144$ crops per image.

Team	Year	Place	Error (top-5)	Uses external data
SuperVision	2012	1st	16.4%	no
SuperVision	2012	1st	15.3%	Imagenet 22k
Clarifai	2013	1st	11.7%	no
Clarifai	2013	1st	11.2%	Imagenet 22k
MSRA	2014	3rd	7.35%	no
VGG	2014	2nd	7.32%	no
GoogLeNet	2014	1st	6.67%	no

Table 2: Classification performance

Number of models	Number of Crops	Cost	Top-5 error	compared to base
1	1	1	10.07%	base
1	10	10	9.15%	-0.92%
1	144	144	7.89%	-2.18%
7	1	7	8.09%	-1.98%
7	10	70	7.62%	-2.45%
7	144	1008	6.67%	-3.45%

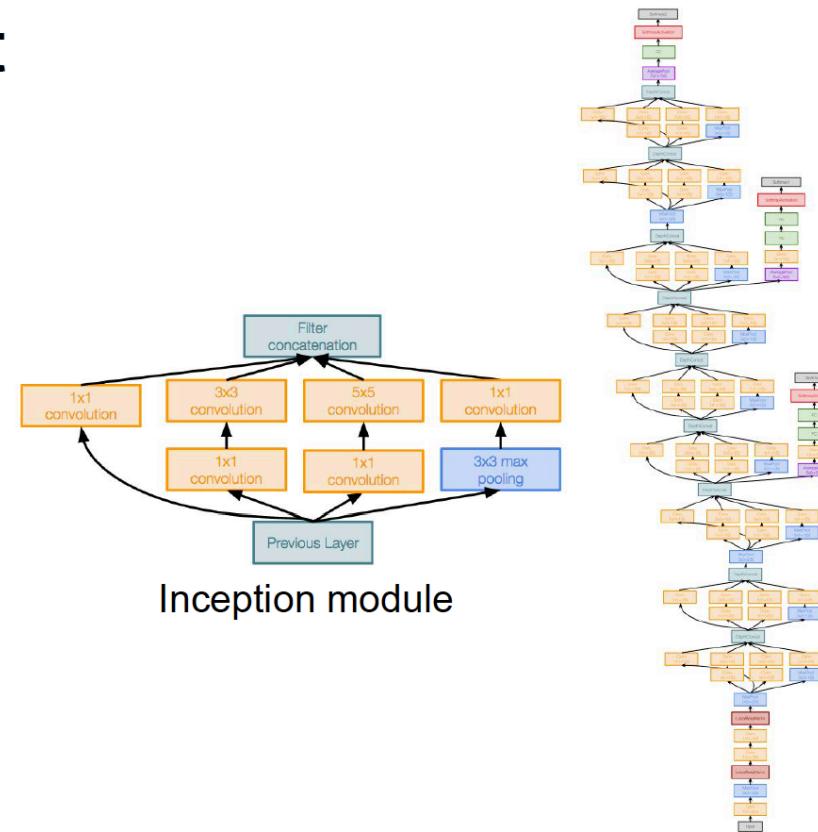
GoogLeNet Summary

Case Study: GoogLeNet

[Szegedy et al., 2014]

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- No FC layers
- 12x less params than AlexNet
- ILSVRC’14 classification winner (6.7% top 5 error)



Tf.nn.conv2d

Getting dimensions right

```
tf.nn.conv2d(input, filter, strides, padding,  
use_cudnn_on_gpu=None, data_format=None, name=None)
```

Input: Batch size x Height x Width x Channels

Filter: Height x Width x Input Channels x Output Channels
(e.g. [5, 5, 3, 64])

Strides: 4 element 1-D tensor, strides in each direction
(often [1, 1, 1, 1] or [1, 2, 2, 1])

Padding: ‘SAME’ or ‘VALID’

Data_format: default to NHWC

Tf.control_dependencies

Control Dependencies

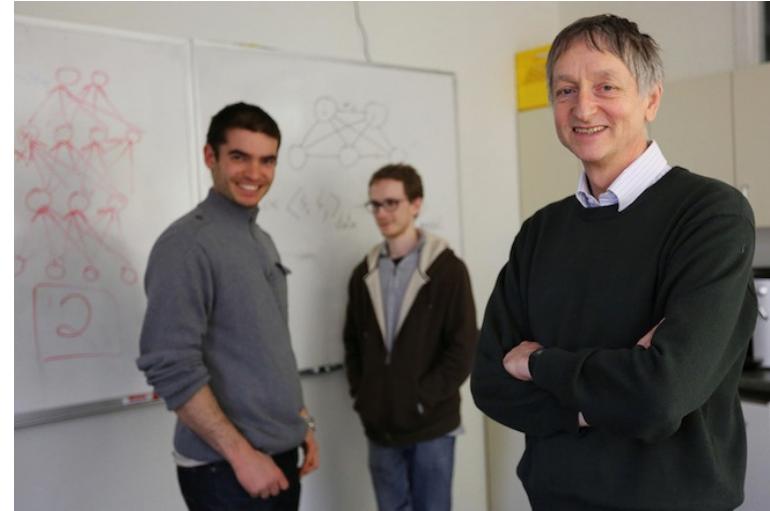
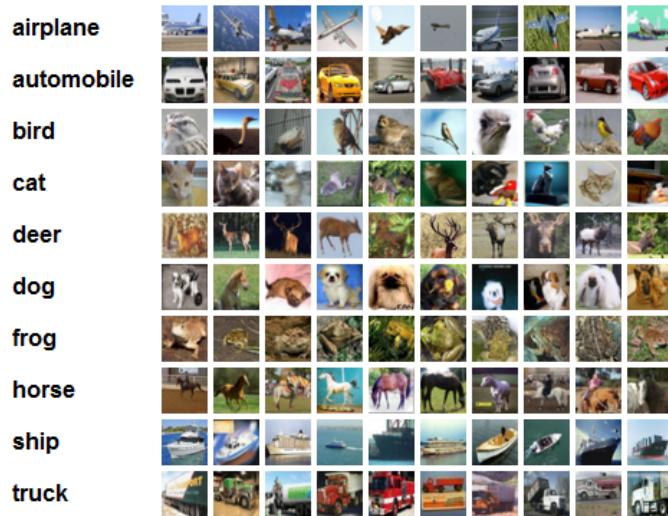
```
tf.Graph.control_dependencies(control_inputs)
```

defines which ops should be run first

```
# your graph g have 5 ops: a, b, c, d, e
with g.control_dependencies([a, b, c]):
    # 'd' and 'e' will only run after 'a', 'b', and 'c' have executed.
    d = ...
    e = ...
```

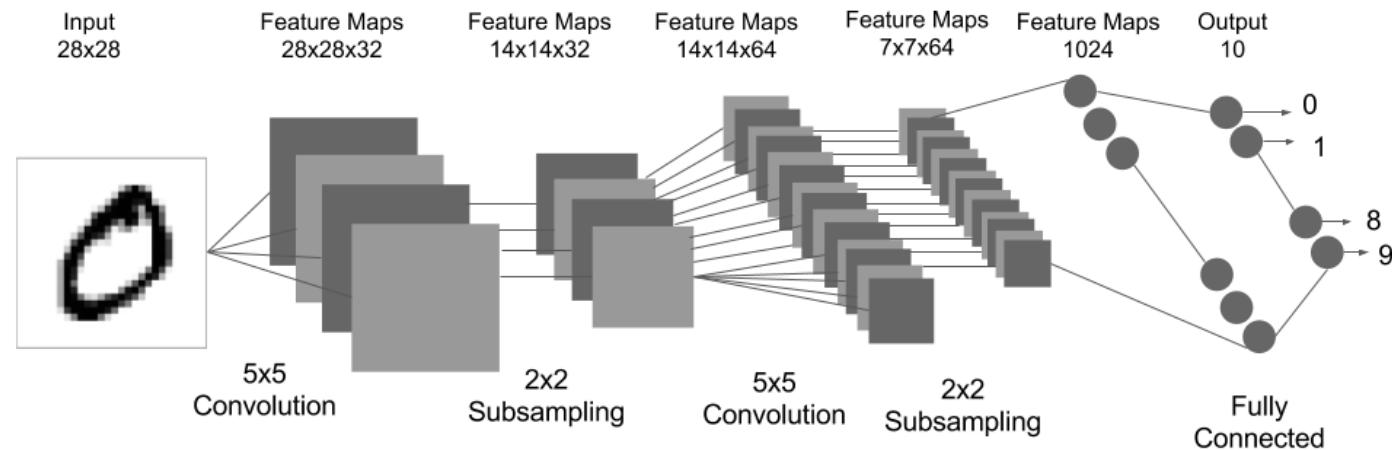
CIFAR-10 Database

- CIFAR-10 Database : 10개의 클래스, 50,000개의 training images, 10,000개의 testing images로 구성되어 있다.
- 이미지 크기 : 32x32 Pixels
- 레이블들: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.



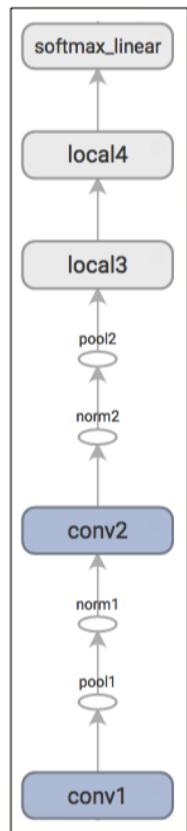
TensorFlow를 이용한 MNIST 숫자분류를 위한 CNNs 구현

- ▣ https://github.com/solaris33/dl_cv_tensorflow_10weeks/blob/master/week1/mnist_classification_with_convolutional_neural_networks.py
- ▣ 정확도 : 약 99.1%



TensorFlow를 이용한 CIFAR-10 이미지 분류를 위한 CNNs 구현

- TensorFlow를 이용한 CIFAR-10 이미지 분류를 위한 CNNs 구현



Layer Name	설명
conv1	convolution과 ReLU activation으로 구성된 Convolutional Layer
pool1	Max Pooling Layer
norm1	local response normalization Layer
conv2	convolution과 ReLU activation으로 구성된 Convolutional Layer
norm2	local response normalization Layer
pool2	Max Pooling Layer
local3	ReLU Activation을 이용하는 Fully Connected Layers
local4	ReLU Activation을 이용하는 Fully Connected Layers
softmax_linear	logits을 생성하기 위한 linear transformation

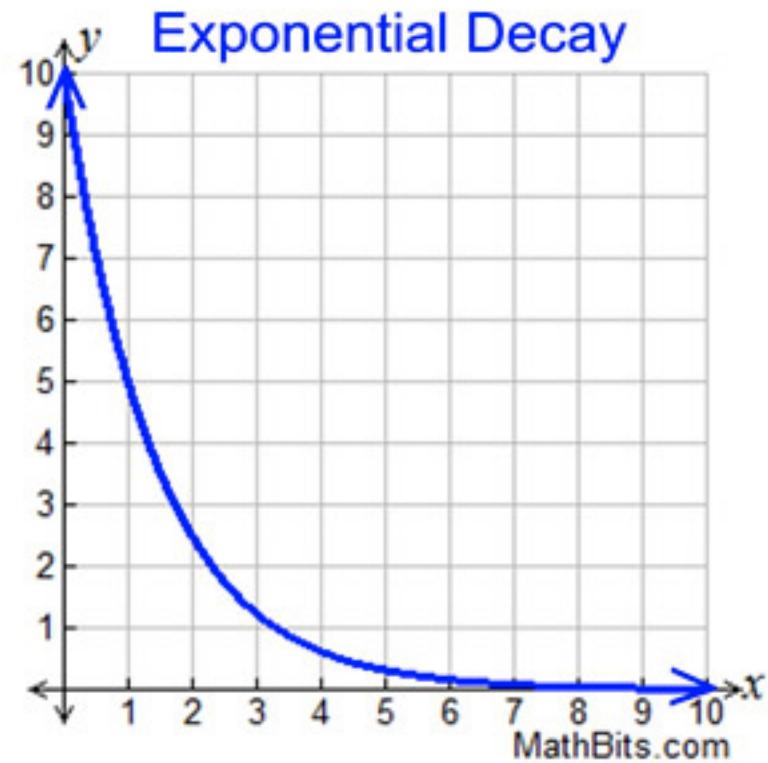
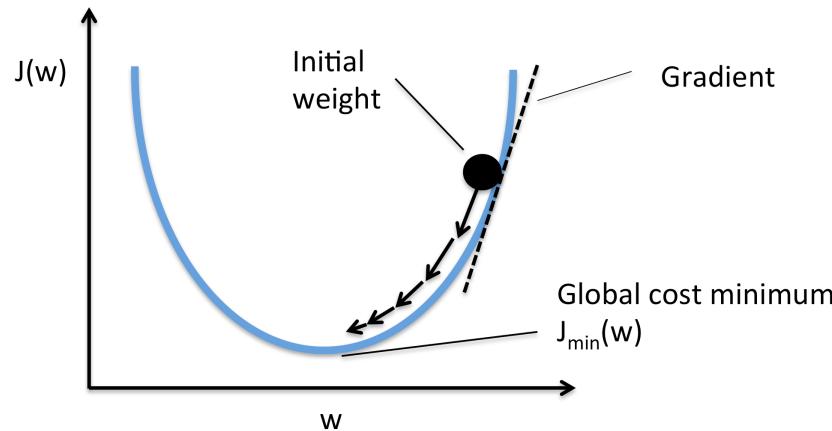
TensorFlow를 이용한 CIFAR-10 이미지 분류를 위한 CNNs 구현

- ▣ https://github.com/solaris33/dl_cv_tensorflow_10weeks/tree/master/week2/cifar10
- ▣ 약 86.6% 정확도

```
202    # conv1
203    with tf.variable_scope('conv1') as scope:
204        kernel = _variable_with_weight_decay('weights',
205                                              shape=[5, 5, 3, 64],
206                                              stddev=5e-2,
207                                              wd=0.0)
208        conv = tf.nn.conv2d(images, kernel, [1, 1, 1, 1], padding='SAME')
209        biases = _variable_on_cpu('biases', [64], tf.constant_initializer(0.0))
210        pre_activation = tf.nn.bias_add(conv, biases)
211        conv1 = tf.nn.relu(pre_activation, name=scope.name)
212        _activation_summary(conv1)
213
214    # pool1
215    pool1 = tf.nn.max_pool(conv1, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1],
216                           padding='SAME', name='pool1')
217    # norm1
218    norm1 = tf.nn.lrn(pool1, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75,
219                      name='norm1')
220
221    # conv2
222    with tf.variable_scope('conv2') as scope:
223        kernel = _variable_with_weight_decay('weights',
224                                              shape=[5, 5, 64, 64],
225                                              stddev=5e-2,
226                                              wd=0.0)
227        conv = tf.nn.conv2d(norm1, kernel, [1, 1, 1, 1], padding='SAME')
228        biases = _variable_on_cpu('biases', [64], tf.constant_initializer(0.1))
229        pre_activation = tf.nn.bias_add(conv, biases)
230        conv2 = tf.nn.relu(pre_activation, name=scope.name)
231        _activation_summary(conv2)
232
233    # norm2
234    norm2 = tf.nn.lrn(conv2, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75,
235                      name='norm2')
```

Learning Rate Decay

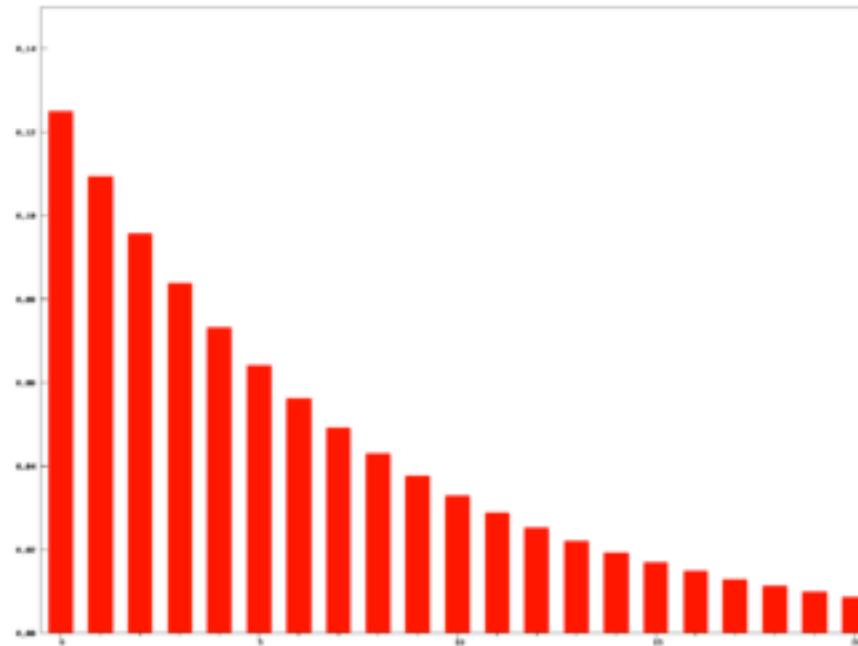
- 시간이 지날수록 Learning Rate를 감소시켜주는 방법



Exponential Moving Average

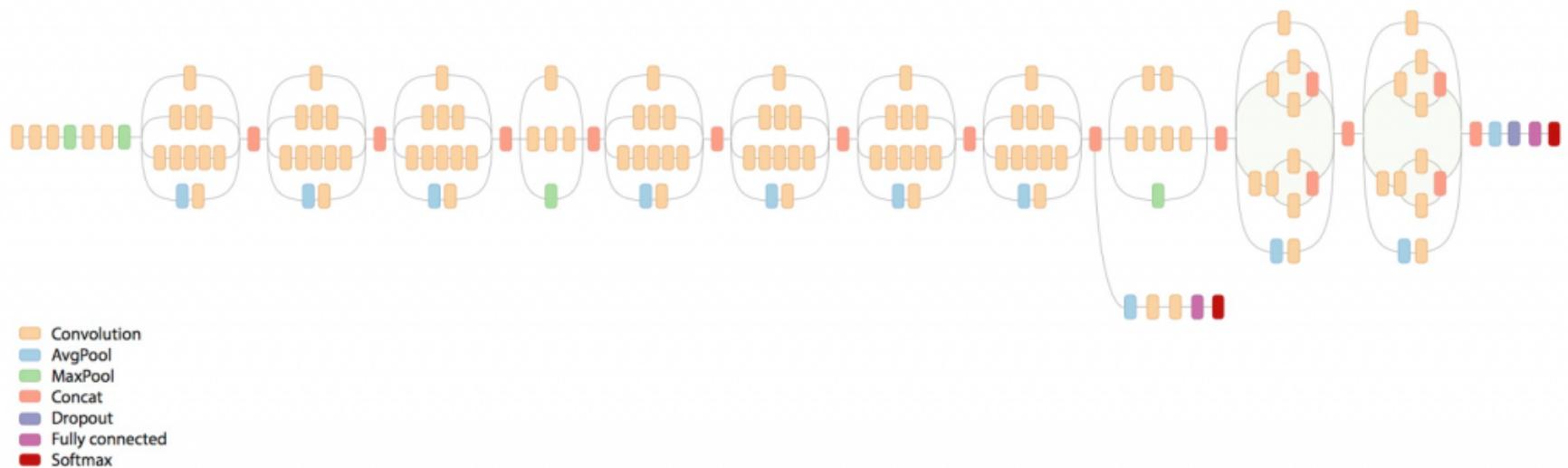
- ▣ 가장 최근 데이터에 큰 가중치를 주어서 계산한 이동 평균값.

$$S_t = \begin{cases} Y_1, & t = 1 \\ \alpha \cdot Y_t + (1 - \alpha) \cdot S_{t-1}, & t > 1 \end{cases}$$



Inception v3 Model

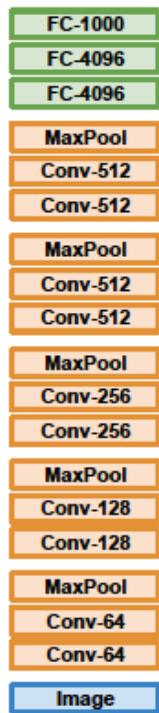
■ Inception v3



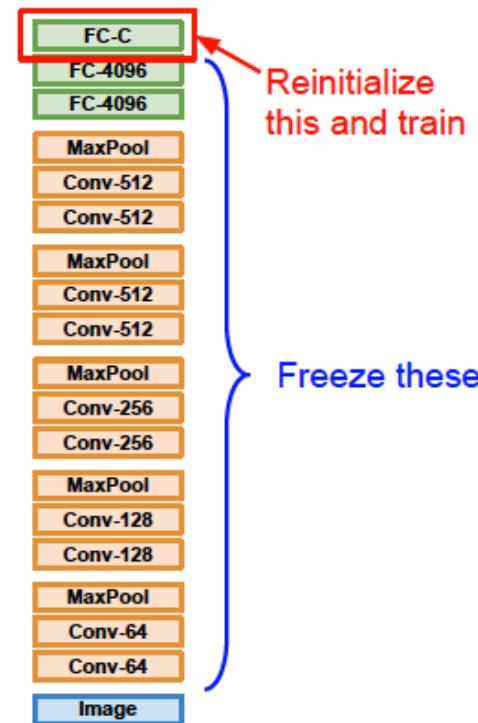
Transfer Learning

Transfer Learning with CNNs

1. Train on Imagenet



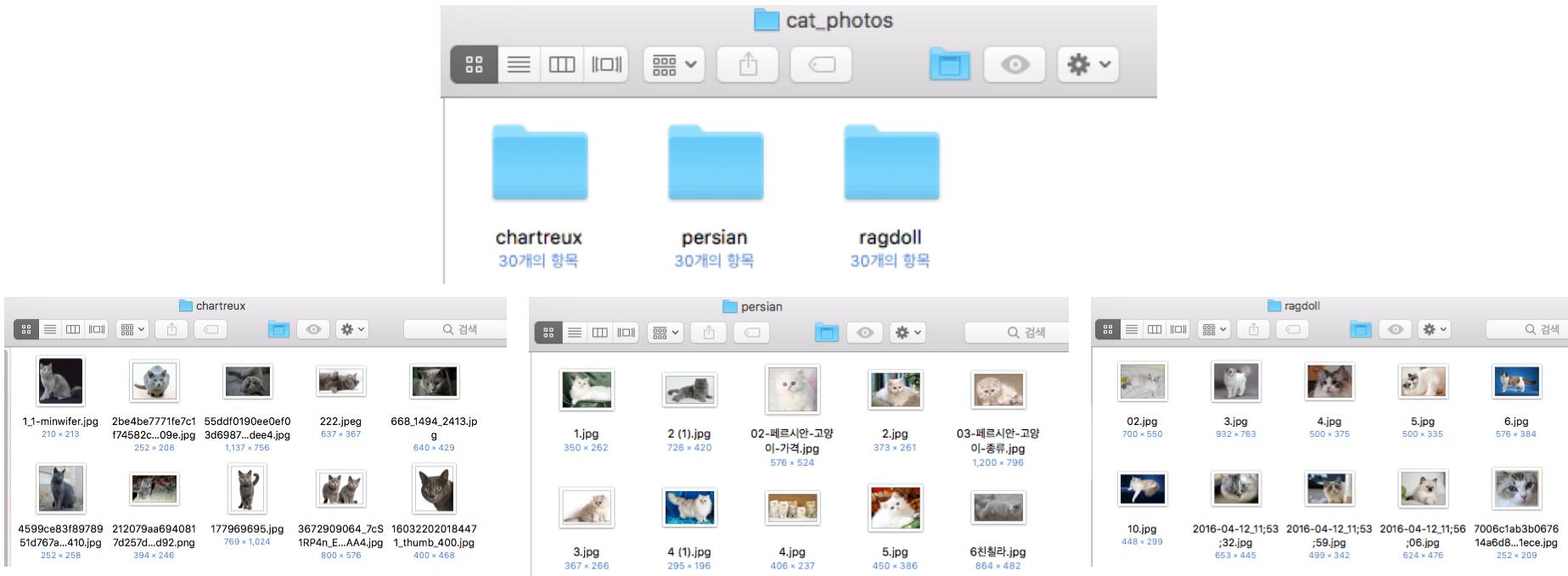
2. Small Dataset (C classes)



Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

TensorFlow를 이용해 Inception v3 Retraining을 통한 Image Classification 구현

- ▣ ~/cat_photos 경로에 다음과 같이 샹트룩스(chartreux), 페르시안(persian), 래그돌(ragdoll) label을 가진 폴더를 만들고 각각의 폴더 안에 각 종류에 해당하는 고양이 이미지를 30개씩 jpg(jpeg)로 저장하였다.
- ▣ python inceptionv3_retrain.py --image_dir ~/cat_photos
- ▣ https://github.com/solaris33/dl_cv_tensorflow_10weeks/tree/master/week2/inception_v3_retraining



TensorFlow를 이용해 Inception v3 Retraining을 통한 Image Classification 구현

- /tmp/test_chartreux.jpg 경로에 샹트록스 이미지를 넣어 놓고 실행하자
- python retrain_run_inference.py



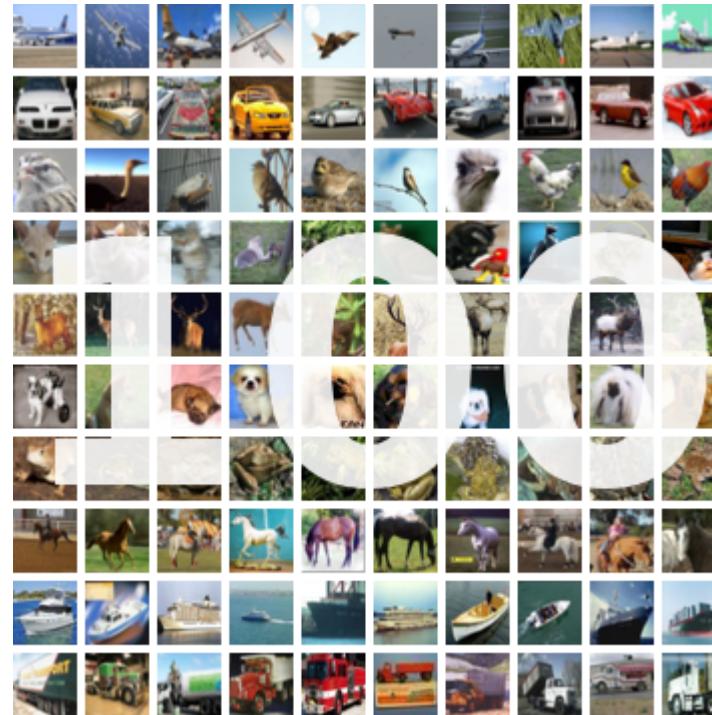
test_chartreux.jpg

```
b'chartreux\n' (score = 0.99727)  
b'ragdoll\n' (score = 0.00159)  
b'persian\n' (score = 0.00114)
```

Result

과제 – CIFAR-100 데이터셋에 대한 CNN Classifier 구현

- TensorFlow를 이용해서 CIFAR-100 Dataset 분류를 위한 CNN Classifier를 구현해봅니다.



Questions & Answers

Thank You!