BSAC Module7

바이오 인공지능 SW

2021. 11

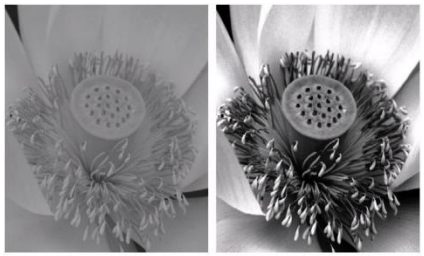
KPC

영상처리(Image Processing)

- 디지털 영상을 수정 및 조작하는 것으로 영상에 여러 연산이나 기술을 적용하여 사용자가 원하는 결과를 새롭게 얻어내는 과정

  - 영상을 더욱 높은 질의 영상으로 만들거나 일그러뜨리는 것
  - 영상의 두드러진 특징을 더욱 두드러지게 하는 것
  - 다른 영상의 일부분으로 새로운 영상을 만드는 것
  - 영상 획득 시, 변질된 영상을 복원시키는 것

영상 처리 기술의 사례



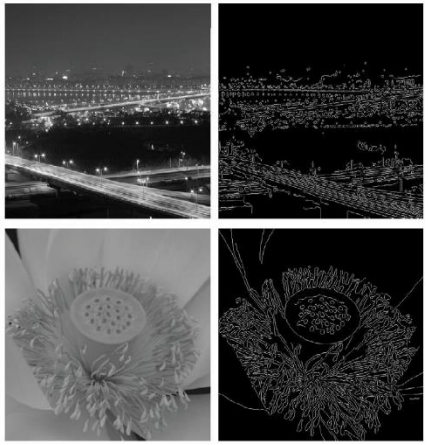
영상 개선 (평활화)

영상 복원

영상 변환 (이산 코사인 변환)

영상 인식 (지문 인식)

영상 압축 (JPEG 압축, 1:12/6/2)



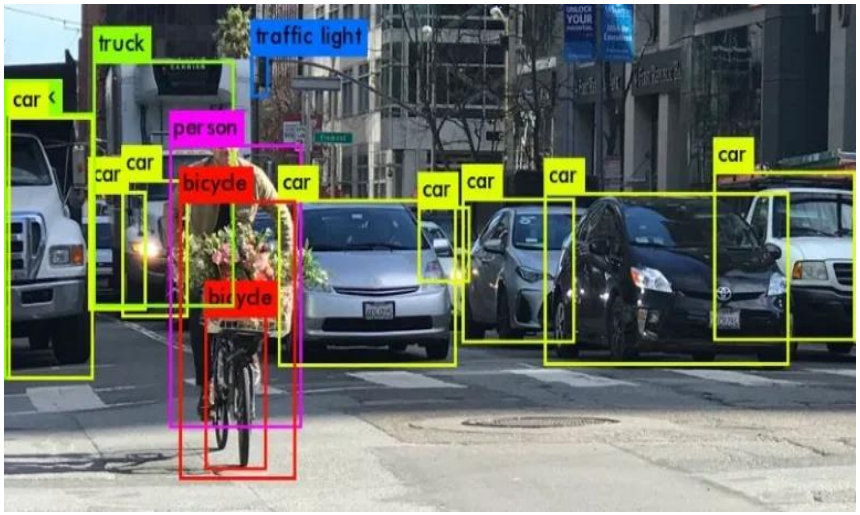
영상 분석 (윤곽선 검출)

컴퓨터 비전(Computer Vision)

- 컴퓨터 비전(Computer Vision)은 영상처리를 포함하는 포괄적인 개념으로, 영상에서 의미 있는 정보를 추출하는 기술
- 영상처리 기술을 통해 원본 영상을 원하는 새로운 영상으로 수정 및 조작한 뒤, 컴퓨터 비전 기술로 원하는 정보를 얻어내는 과정

객체 인식 (Object Recognition)	영상 속 물체가 무엇인지 인식하는 기술 및 여러 컴퓨터 비전 기술의 포괄적인 개념
객체 검출 (Object Detection)	영상 속에서 찾고자 하는 물체가 어디에 있는지 검출하는 기술
객체 추적 (Object Tracking)	영상 속 물체가 어디로 움직이는지 추적하고 해당 물체에 대한 검출과 인식을 유지하는 기술

컴퓨터 비젼 기술의 사례



객체 검출(Object Detection)

객체 추적(Object Tracking)

OpenCV 소개

- Open Source Computer Vision Library
- 영상처리(Image Processing)와 컴퓨터 비전(Computer Vision) 분야의 가장 대표적인 라이브러리(Library)

  - 사진 혹은 영상을 처리할 수 있는 기능
    - 영상 파일의 읽기 및 쓰기
    - 비디오 캡처 및 저장
    - 영상처리(Image Processing) 알고리즘 및 기술
    - 영상이나 비디오에서 얼굴, 눈, 자동차 등과 같은 특정 물체 분류(Classification) 등

  - 기본적인 기계학습(Machine Learning), 딥러닝(Deep Learning)을 응용할 수 있게 “Tensorflow”, “Torch/PyTorch” 등과 같은 프레임워크(framework)도 지원

OpenCV 주요 함수

- https://docs.opencv.org/4.5.0/index.html

함수	기능	함수	기능
cv2.imread()	이미지 읽기	cv2.imwrite()	선, 사각형, 원, 타원 도형 그리기
cv2.cvtColor()	이미지 컬러공간 변환	cv2.line()	
cv2.resize()	이미지 사이즈 변경	cv2.rectangle()	
cv2.imshow()	이미지 출력	cv2.circle()	
cv2.putText()	텍스트 삽입	cv2.ellipse()	
cv2.split()	이미지 채널 분리	cv2.normalize()	값의 범위 변경
cv2.merge()	이미지 채널 병합		

딥러닝 도구

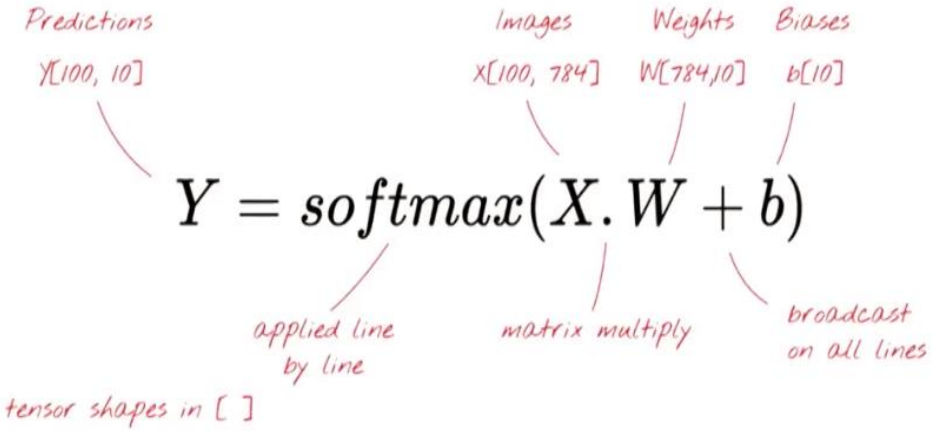
- 텐서플로우(Tensorflow)

  - 2015년 구글이 공개
  - 텐서(Tensor)를 사용하여 신경망을 구현하는 파이썬(Python) 라이브러리
  - 초기에는 신경망을 구현하기 위해서 텐서플로우를 사용하는 것이 기본이었으나, 지금은 케라스(Keras) 등 프로그래밍 구현이 더 편리하고 다양한 도구가 많이 개발

- 케라스(Keras)

  - 딥러닝 모델을 파이썬으로 쉽게 구축해주는 패키지
  - 텐서플로우나 테아노(Theano), Microsoft의 CNTK와 같은 딥러닝 플랫폼 위에서 동작
  - 텐서플로우가 원시적인 코딩이 가능하고 세세하게 기능을 구현할 수 있는 라이브러리이지만 초심자에게 사용법이 다소 복잡하게 느껴질 수 있다. 이에 반해 케라스는 초심자도 쉽게 신경망을 구현할 수 있는 라이브러리를 제공

GPU (Graphics Processing Unit)



딥러닝(Deep Learning)의 기본 식

- 신경망 모델을 학습하려면 GPU(Graphics Processing Unit) 장치를 추가적으로 활용해야 훈련(Training) 속도가 훨씬 빠르다.
  - 딥러닝은 기본적으로 행결(Matrix) 곱하기 연산
  - 딥러닝 신경망은 수없이 많은 텐서(Tensor)로 구성되어 훈련을 위해 순전파(Forward Propagation), 역전파(Back Propagation) 과정을 거치면서 수많은 연산이 이루어진다. 따라서 행렬연산에 강한 GPU가 CPU보다 훨씬 빠를 수밖에 없는 것이다.

GPU (Graphics Processing Unit)

- 가장 널리 사용되는 GPU는 NVDIA의 제품. 이를 사용하려면 먼저 다음을 설치해야 한다.
  - 드라이버 : CUDA
  - cuDNN(CUDA Deep Neural Network)
- GPU를 사용하려면 텐서플로우를 GPU 이용 가능 버전으로 설치해야 한다.
  - tensorflow-gpu
- 구글 코랩(Colab) 플랫폼을 이용해서 학습하면 GPU를 무료로 활용할 수 있다.

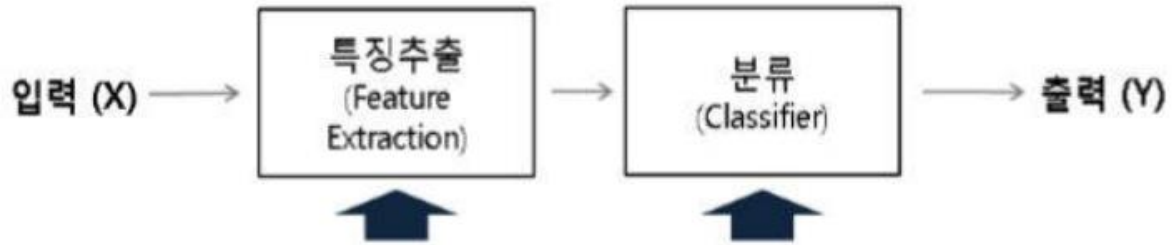
인공 신경망(ANN)

신경망의 특징

- 신경망(Neural Network)이 로지스틱 회귀(Logistic Regression), 랜덤 포레스트(Random Forest) 등 기타 머신러닝 모델과 가장 다른 점은 특성 공학(Feature Engineering)이 거의 필요가 없다는 것
- 기존의 머신러닝 알고리즘에서는 분석에 가장 효과적인 특성(Feature)을 찾는 작업이 필요하며, 이때 해당 응용 분야 전문가의 경험이 필요했다.
- 최적의 특성을 찾으려면 시간과 비용도 많이 소모
  - 딥러닝은 이런 과정이 거의 필요가 없게 되어 학습 과정을 단순화할 수 있게 되었고,
  - 사람의 개입을 최소화할 수 있게 되었다.
- 최적의 특성을 찾는 작업 자체가 신경망 학습 과정에서 반영되기 때문

신경망의 특징

< 머신러닝 (Machine Learning >



- 특징추출을 위한 알고리즘을 인간이 직접 제공
- 해당분야에 대한 지식 및 직관, 알고리즘 구축을 위한 상당한 노력 필요
- 컴퓨터 '학습'의 영역

< 딥러닝 (Deep Learning >

- 컴퓨터 '학습'의 영역

전이 학습 (Transfer Learning)

- 신경망의 또 다른 중요한 특징은 확장이 쉽다는 것
  - 다른 곳에서 다른 데이터로 학습한 모델을 그대로 불러서 재사용하기가 쉽다
  - 이를 전이학습(Transfer Learning)이라고 한다
  - 전이학습을 이용하면 새로운 데이터를 이용한 학습 시간도 줄일 수 있고 더 적은 데이터로도 성능이 좋은 모델을 만들 수가 있다.
- 전이학습은 모델을 학습시킨 도메인(Domain)과 적용 도메인이 달라도 효과적으로 사용할 수 있다.

- Source Domain: 최초의 모델 학습시킨 도메인
- Target Domain: 전이학습을 수행할 새로운 문제 도메인

신경망 모델 선택

- 다양한 구조의 신경망 모델이 소개되고 있으며, 성능도 나날이 발전
- CNN와 RNN을 결합하여 CNN으로 데이터 전처리와 같은 작업을 한 후 RNN을 수행하기도 함

- 신경망을 도입하는 방법은 처음에는 전이학습(Transfer Learning)을 활용하여 검증된 모델을 사용하여 내가 원하는 동작을 하는지 먼저 검증하는 것이 좋다.

  - 전이학습을 사용하는 데에는 어떠한 비용도 들지 않는다.
  - 만일 스스로 모델을 새로 만들어야 한다면, 처음에는 가능하면 간단한 모델을 만들고, 학습 데이터도 가능한 최소한으로 사용해 보는 것이 좋다.
  - 처음부터 좋은 성과를 내려고 하면 학습 시간이 오래 걸리고 시행착오로 낭비하는 시간이 많게 된다.
  - 따라서 점차적으로 도메인(Domain)의 영역을 확대시켜 모델의 영향 범위를 확대하는 것이 권장

딥러닝의 한계와 적용하기에 유용한 문제

- 딥러닝은 블랙박스 모델(은닉층의 존재로 인해)로 모델이 내린 결정의 이유를 정확히 알 수가 없으며, 질병 진단 등 설명이 필요한 분야에는 적용이 어렵다.

- 학습 및 훈련된 신경망 파라미터(Parameter)를 보면 그 내용을 해석하기가 어려우며, 어떤 특징(Feature)들로 인해 오동작하는지 조차도 해석하기 난해하기 그지없다. 따라서 신경망 모델 선택과 효용성 있는 데이터 구축에 많은 시간과 노력이 배분되어야 할 것이다.

- 추천 시스템, 자동 번역기, 컴퓨터 비전처럼 설명은 크게 필요치 않고, 적절한 동작에 초점이 맞춰진 문제인 경우에는 매우 유용하다.

- 동일한 성격의 다량의 데이터 분석에 유용하다. (이미지, 텍스트, 센서 데이터 등)

합성곱 신경망(CNN)

MLP vs CNN

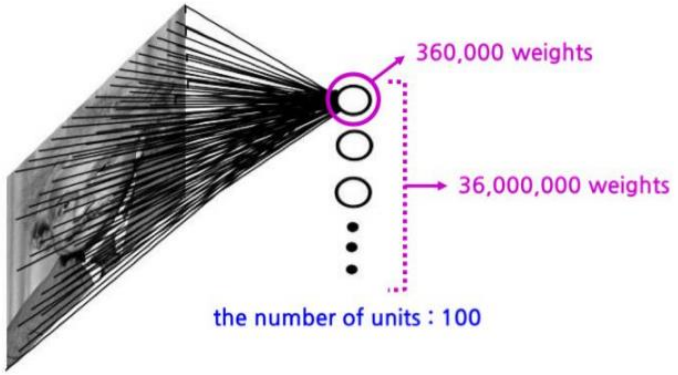
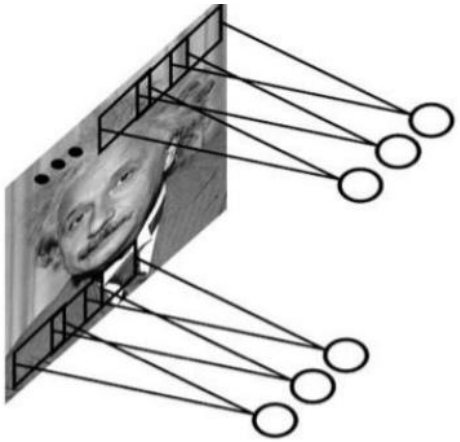


image size : 300x400x3



Convolution Neural Network (CNN)

Input

Conv Layer 1

Pooling Layer 1

Conv Layer 2

Pooling Layer 2

Conv Layer 3

Pooling Layer 3

Fully Connected Layer

0

1

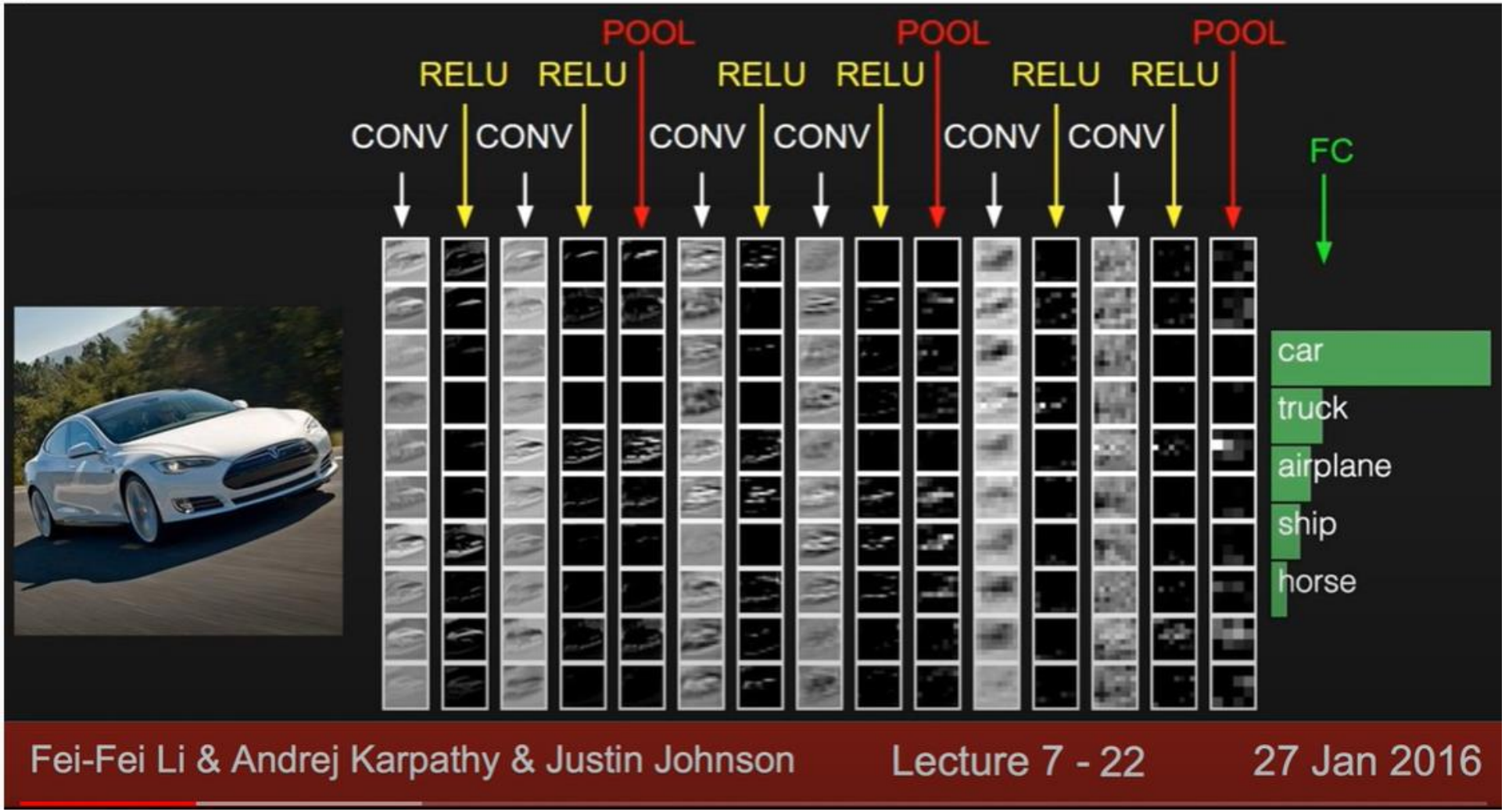
Output layer

Feature Extractor

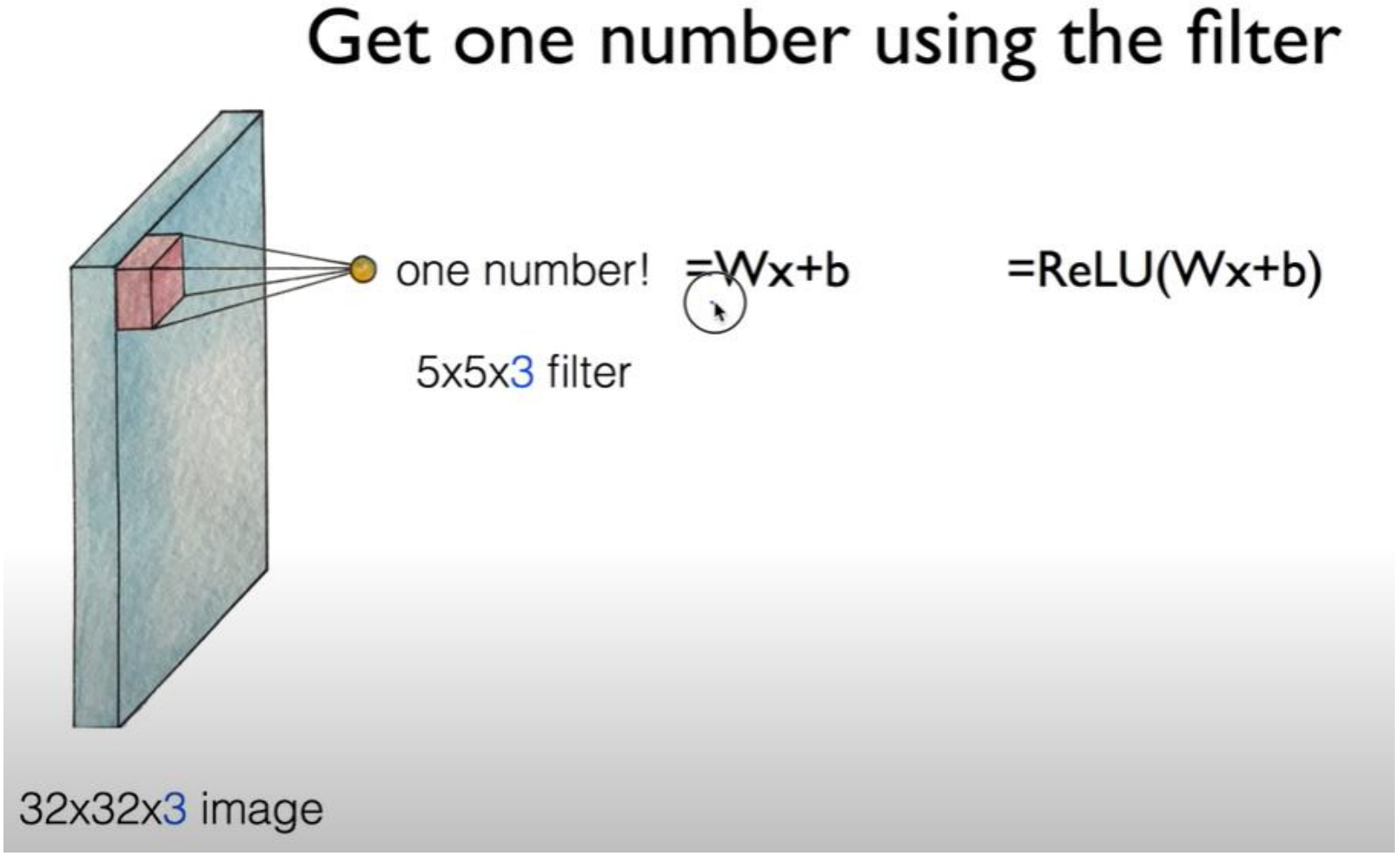
Classifier

CNN

preview:


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse
Fei-Fei Li & Andrej Karpathy & Justin Johnson
Lecture 7 - 22
27 Jan 2016

CNN 동작

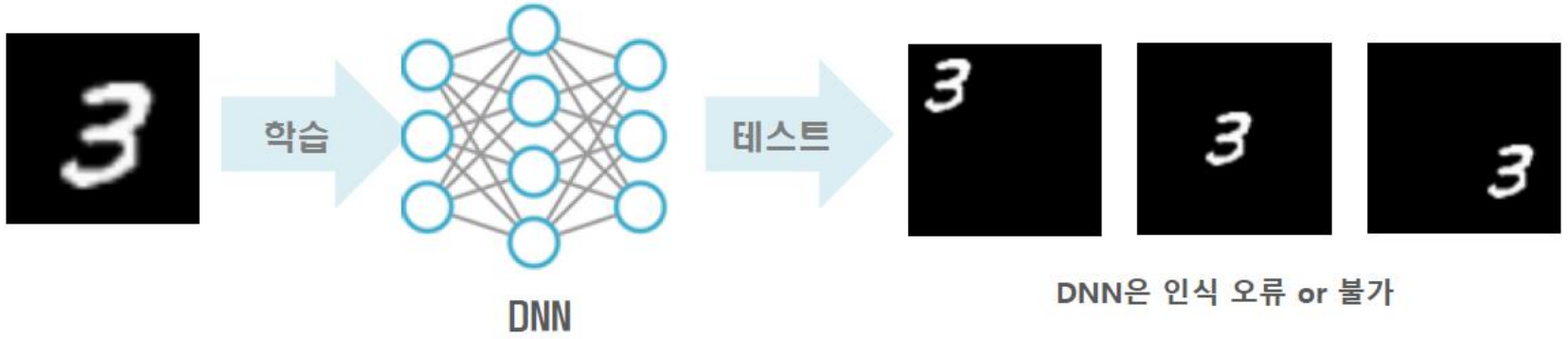

Get one number using the filter
one number!
=Wx+b
=ReLU(Wx+b)
5x5x3 filter
32x32x3 image

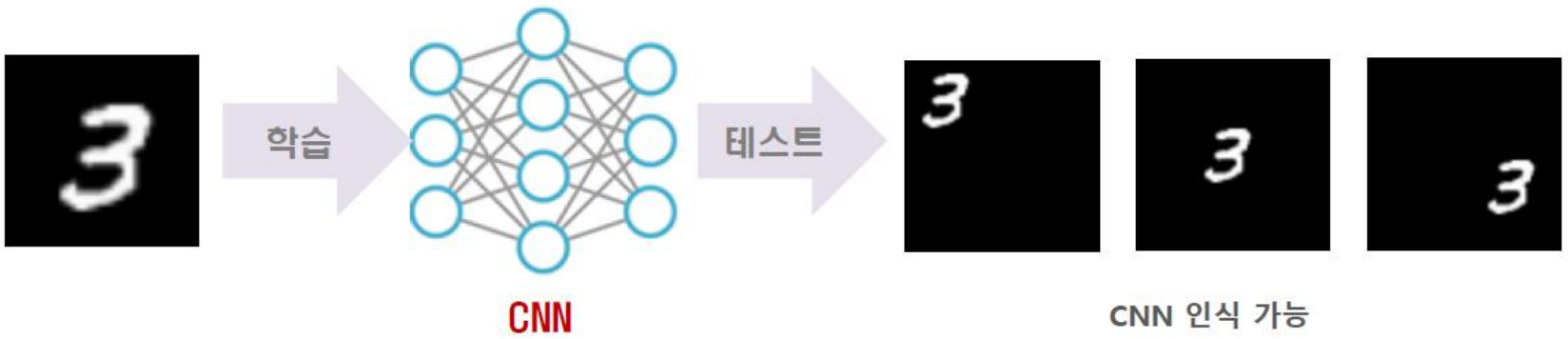
MLP의 한계

- 계층 수와 유닛(뉴런) 수의 곱에 비례하여 가중치 수가 급격히 증가
  - 1000개의 유닛으로 구성된 계층이 2개만 있어도 1백만개의 계수가 필요

- 이미지 처리에 사용할 경우
  - 학습 패턴의 위치에 민감하게 동작
  - 아래 세 개의 5는 패턴이 다르다고 판단한다. MLP로 이러한 숫자 인식을 하려면 숫자의 크기를 비슷하게 맞추어야 한다

MLP의 한계





CNN 특징

- MLP 신경망의 한계를 극복하기 위해서 제안
- 전결합망 구조를 사용하지 않고, 좁은 면적 단위(예를 들면 3x3 픽셀 단위)로 신호를 필터링하고, 그 결과를 다음 계층의 입력으로 사용
- 작은 공단 단위(패치)로 필터링하는 목적
  - 이미지의 특징을 구성하는 어떤 패턴을 공간상 위치에 상관없이 찾아내기 위해서

특성 맵 (Feature map)

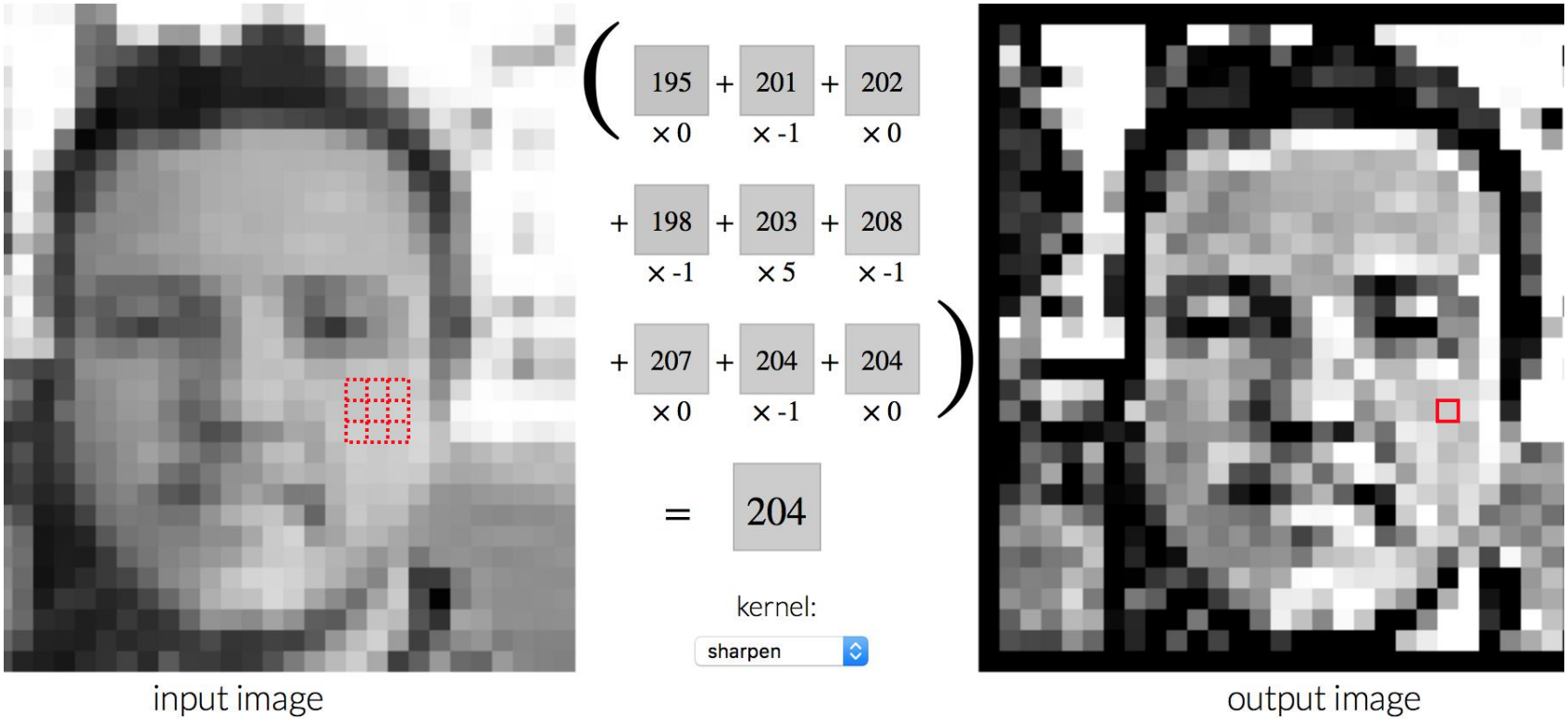
- 특징 패턴
  - 기하학적인 단위 모양(엣지, 대각선, 수평선, 수직선 등)
  - 질감(texture) 등

- 특성맵(feature map)
  - 다음 계층으로 넘겨주는 유효한 값을 활성값(activation)이라고 함
  - 활성값의 전체 집합

- CNN의 동작은, 마치 돋보기로 이미지 전체를 차례대로 스캔하면서 특정 성분을 파악하는 것과 같다

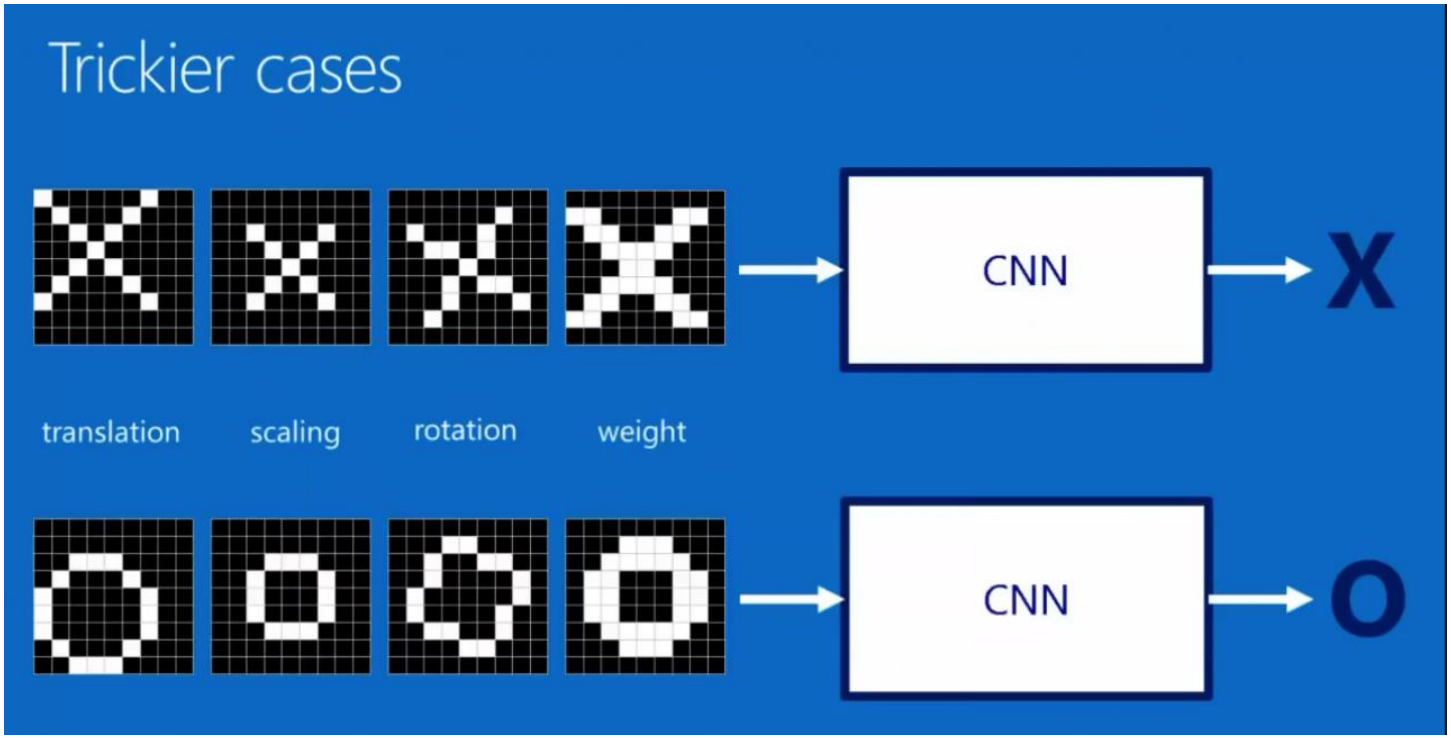
이미지 필터링

- 시각화


195 + 201 + 202
× 0 × -1 × 0
+ 198 + 203 + 208
× -1 × 5 × -1
+ 207 + 204 + 204
× 0 × -1 × 0
= 204
kernel:
sharpen
input image
output image

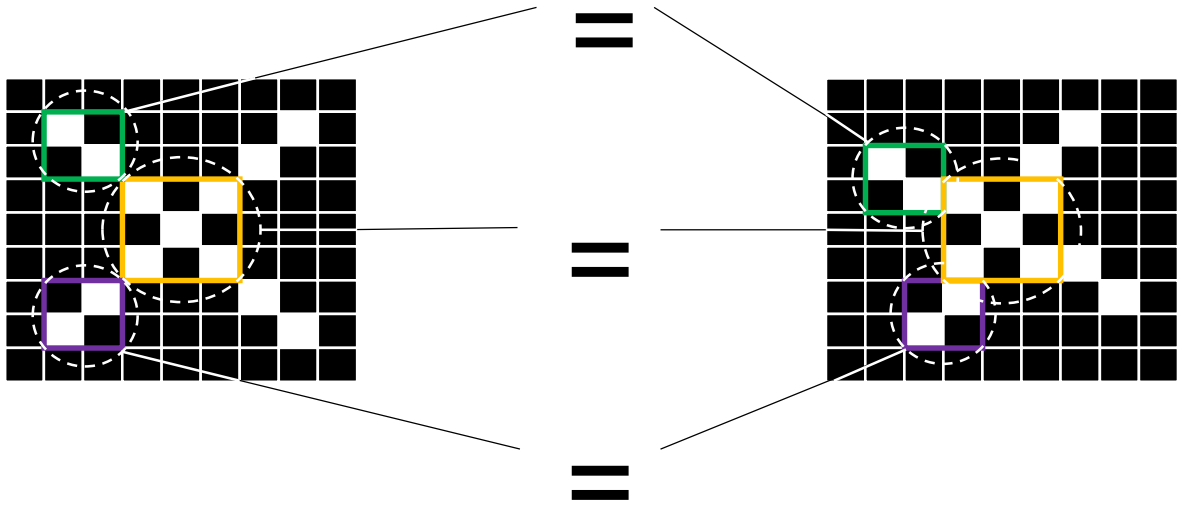
CNN 개념 소개

- 동영상(2:40) (원본(26:30))



동일한 패턴 검출

- 공간상의 위치를 픽셀 단위로 일 대 일 비교하면
  - 이 두 그림에서 흰 점이 일치하는 부분은 거의 없다.
  - 따라서 이미지를 벡터로 환산(즉, 1차원 벡터로 변환)하여 비교하면 서로 다르다고 판단
- 3x3 픽셀 단위로 나누어 관찰하면
  - 절대적인 위치는 다르지만 일치하는 패턴이 여러 곳에 나타나므로
  - 다른 위치에 있는 같은 패턴을 찾아낼 수 있다



위치, 크기, 두께, 각도 등에 invariant

- CNN의 필터는 바로 이러한 패턴의 활성화 성분 크기 즉, 활성값을 찾아 준다.
- 이러한 원리를 이용하여 CNN은 어떤 패턴의 크기, 절대적 위치, 두께, 회전 각도 등이 다르더라도 이를 찾아서 다음 계층으로 전달할 수 있다.

커널 (kernel) 수

- CNN에서는 특성맵을 한가지만 만드는 것이 아니라 수십~수백 가지를 만든다. (처음에는 3원색인 경우 3에서 출발)

- 왜냐하면 찾아내야 할 패턴이 가로성분, 세로성분, 대각선성분, 'X' 형 성분, 색상정보, 엣지 등 여러 가지가 있기 때문

-

- 어떤 계층의 합성곱 필터의 종류 수가 32라면 이 계층에서 생산되는 특성맵은 32개가 된다.

- 합성곱 필터를 커널(kernel)이라고 부르는데 커널수가 많을수록 다양한 패턴을 찾아낼 수 있다

- 그러나 모델의 복잡도가 너무 커지면 과대적합의 원인
  - 내부 parameter 수가 많아 학습 데이터를 너무 정교하게 모델링하기 때문

Convolution Filter

입력 데이터(height, width)에 대해 **필터(커널)**을
일정 간격(**Stride**) 만큼 이동해 가며 행렬 곱셈 연산 수행

1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1

입력 데이터

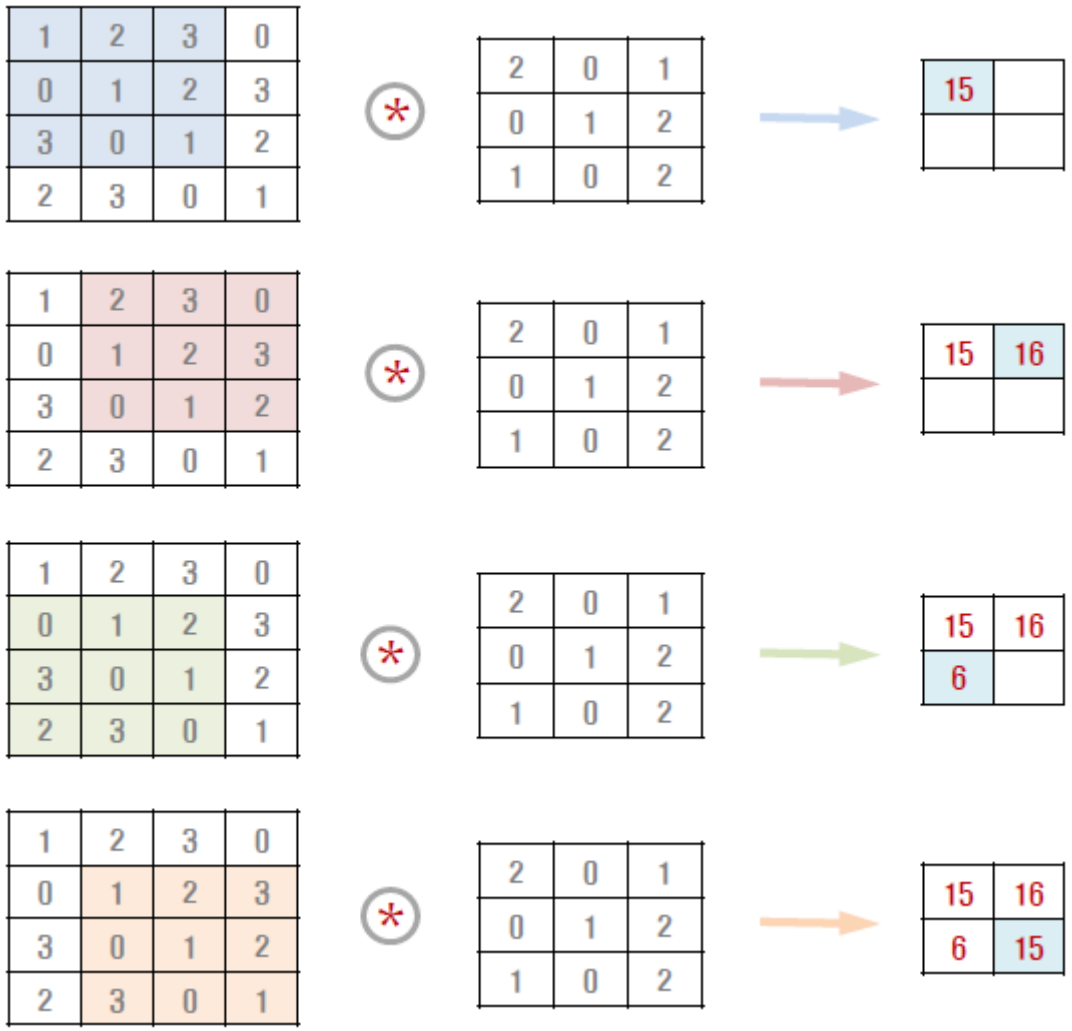
*

2	0	1
0	1	2
1	0	2

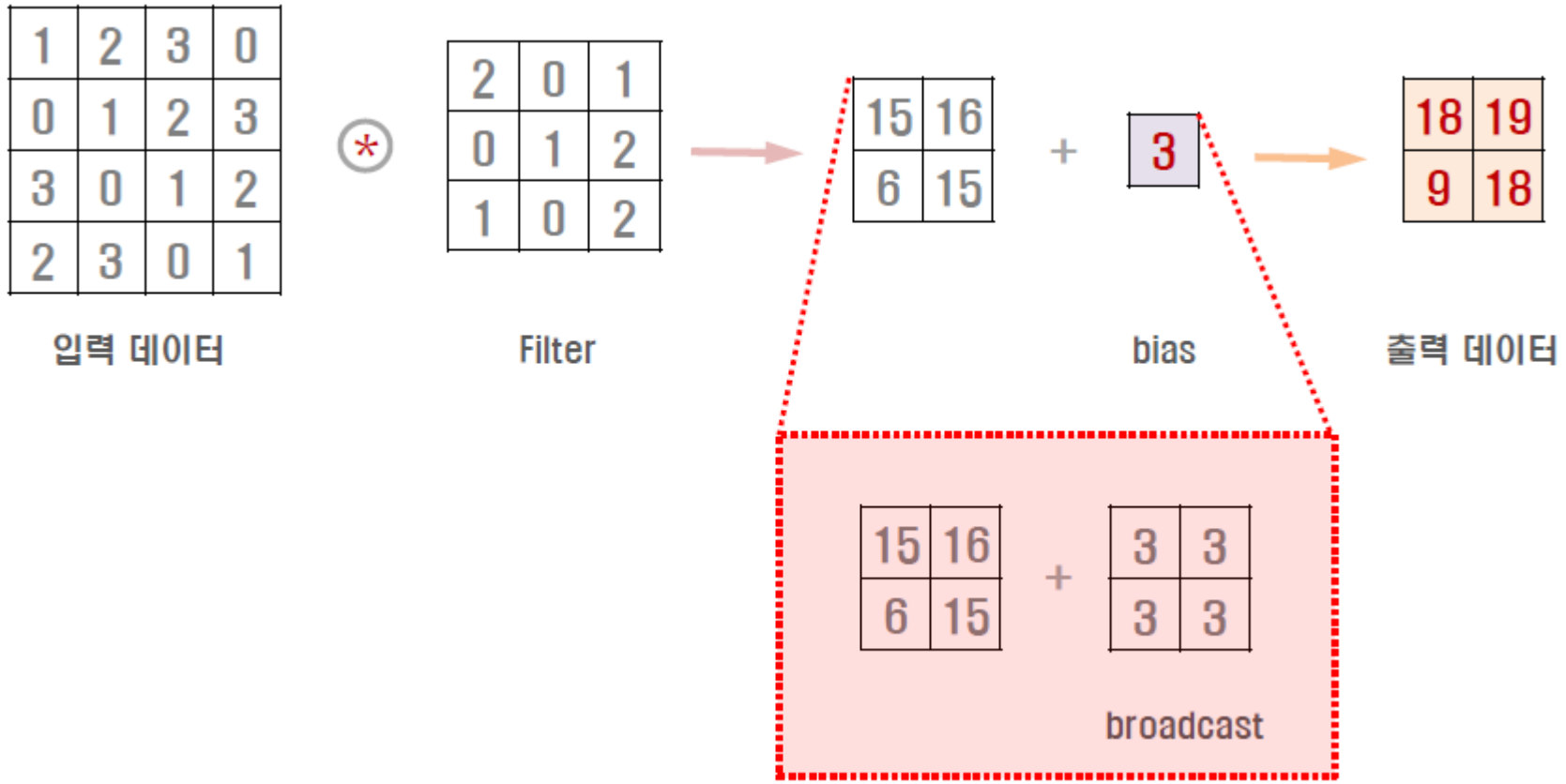
필터

15	16
6	15

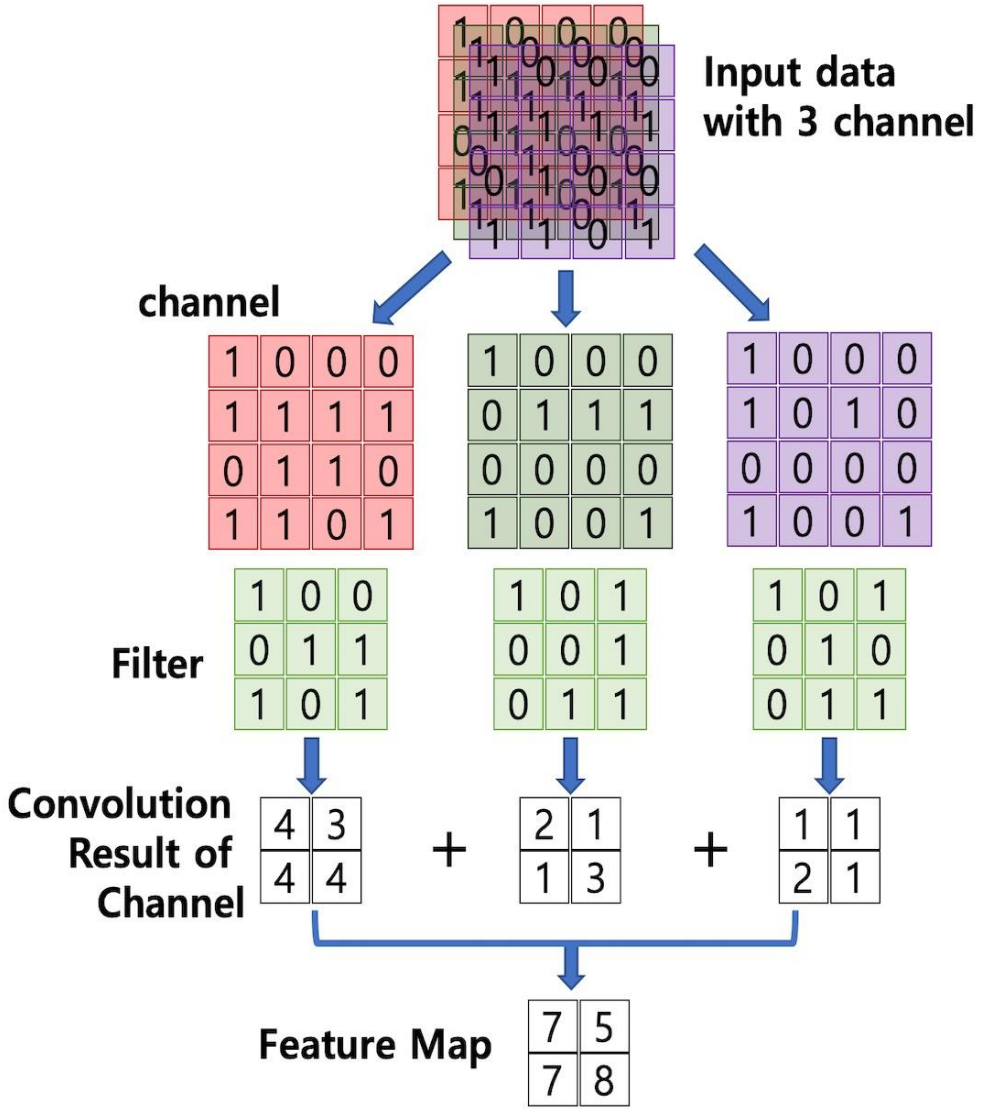
Convolution Filter 연산



Convolution Filter 연산



컬러 영상(채널=3) 합성곱(Convolution) 동작



패딩 (Padding)

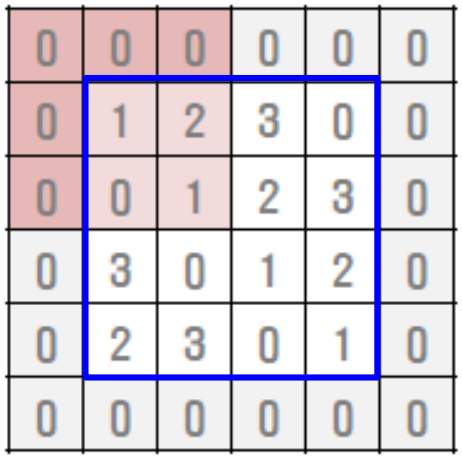
- 필터의 크기는 5x5나 7x7 등 임의의 크기로 정할 수 있다.

- 필터의 크기로 인해 가장자리 부분의 데이터가 부족
  - 입력과 출력의 크기가 달라지게 된다
  - 3x3 윈도우의 패치를 사용하는 가로, 세로 각 2 만큼씩 축소(상하좌우 가장자리에 픽셀이 한 줄씩 부족)

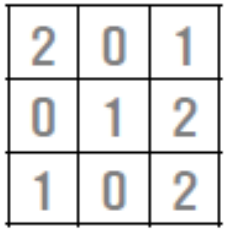
- 이를 보정하기 위해서 입력신호의 가장자리 부분에 보통 0을 미리 채워 넣는 것을 패딩(padding)

- Conv2D 계층 : padding 인자를 사용
  - valid : 패딩을 사용하지 말라는 의미
  - same : 출력의 차원이 입력과 같아지도록 적절한 수의 패딩을 자동으로 입력하라는 의미

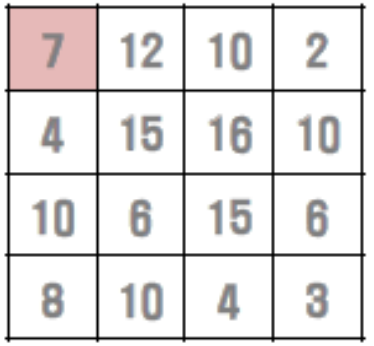
패딩 (Padding)



입력 데이터(4, 4)
zero padding(1,1)
추가(6,6)



Filter(3, 3)



출력 데이터(4, 4)

축소 샘플링 (subsampling)

- 합성곱을 수행한 결과 신호를 다음 계층으로 전달할 때, 모든 정보를 전달하지 않고 일부만 샘플링하여 넘겨주는 작업

- 축소 샘플링을 하는 이유
  - 좀 더 가치 있는 정보만을 다음 단계로 넘겨주기 위해서

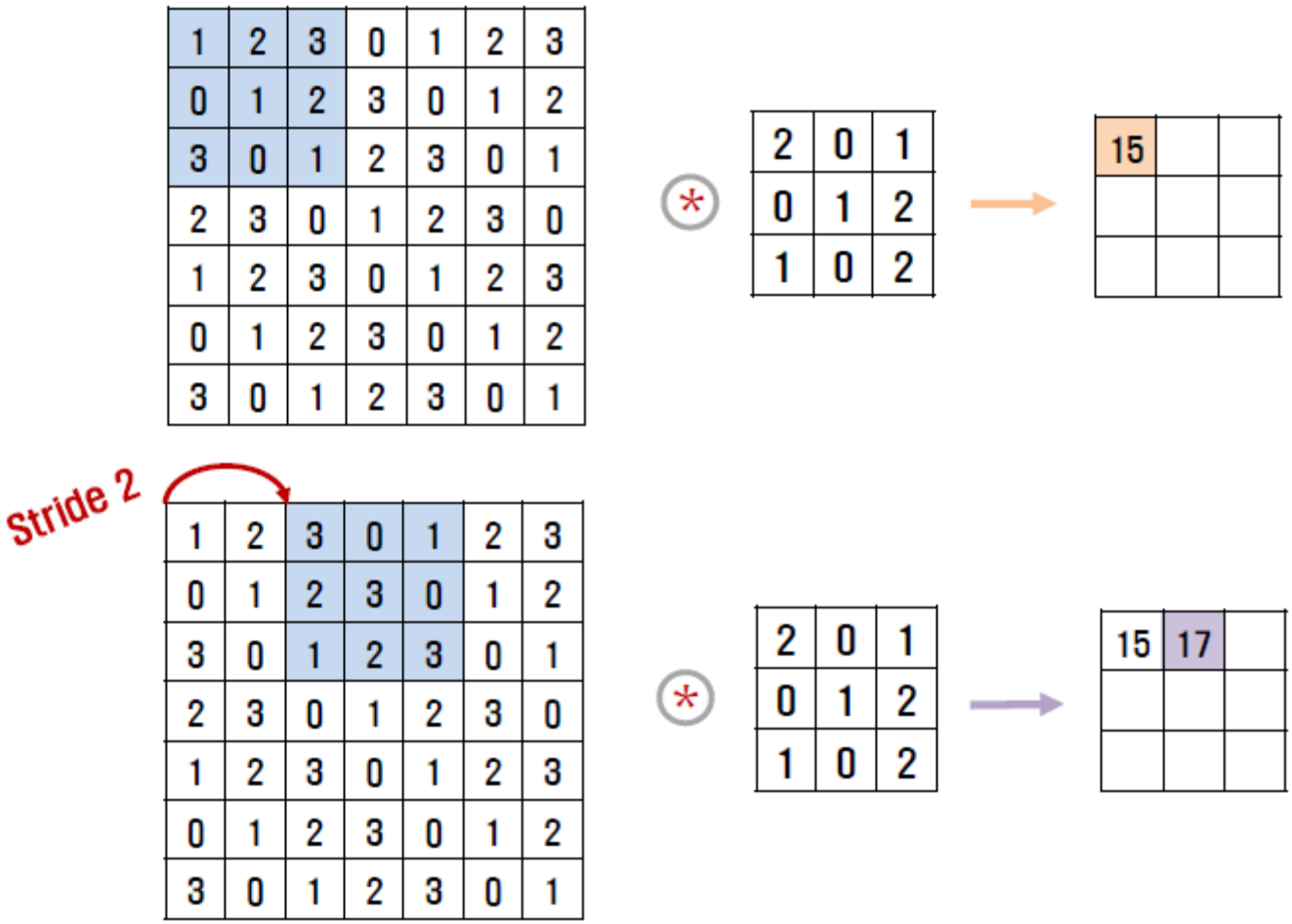
- 머신러닝의 최종 목적은 정보를 결국 줄여나가야 하며 따라서 핵심 정보만 다음 계층으로 전달하는 장치가 필요
  - 커널 수를 늘리면 특성맵의 숫자가 점차 커지게 된다.

- 축소 샘플링
  - 스트라이드(stride)
  - 풀링(pooling)

스트라이드 (Stride)

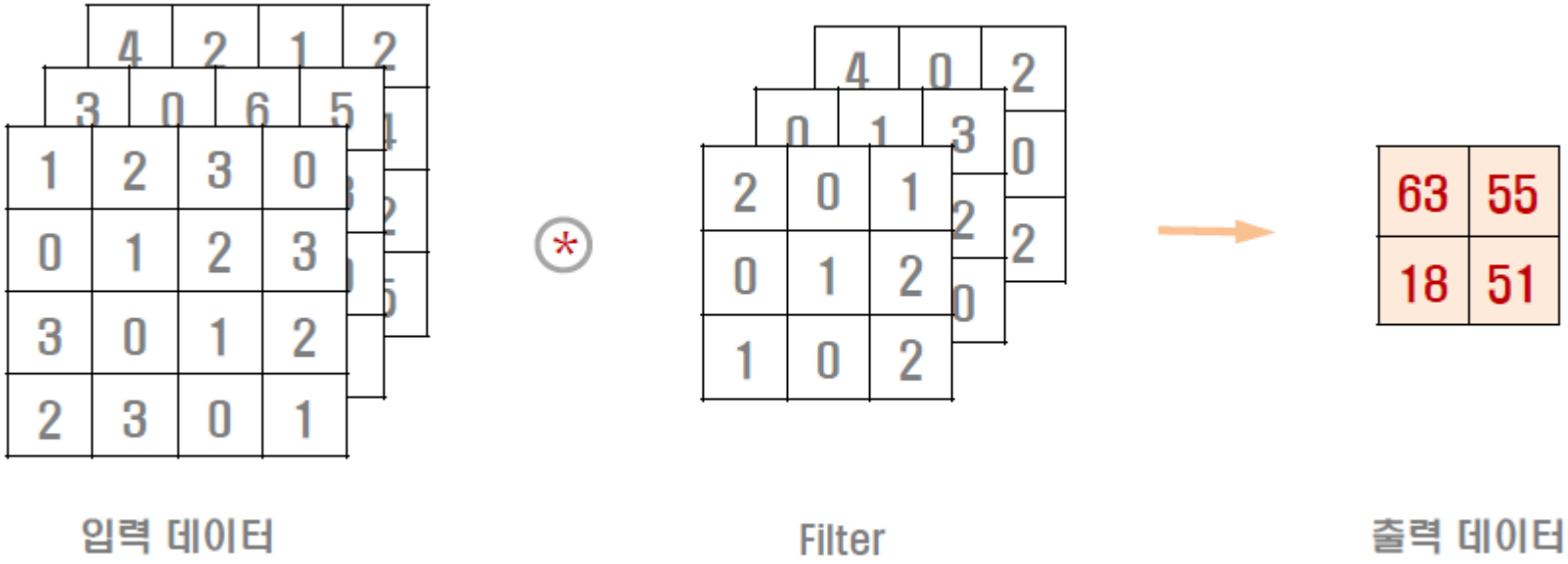
- 합성곱 필터링을 수행할 때 패치를 (예를 들면 3x3 크기)를 한 픽셀씩 옆으로 이동하면서 출력을 얻지 않고, 2 픽셀씩 또는 3 픽셀씩 건너 뛰면서 합성곱을 수행하는 방법
  - 스트라이드 2 : 출력 특성맵의 크기가 1/4로 축소
  - 스트라이드 3 : 출력 특성맵의 크기가 1/9로 축소

스트라이드 (Stride)

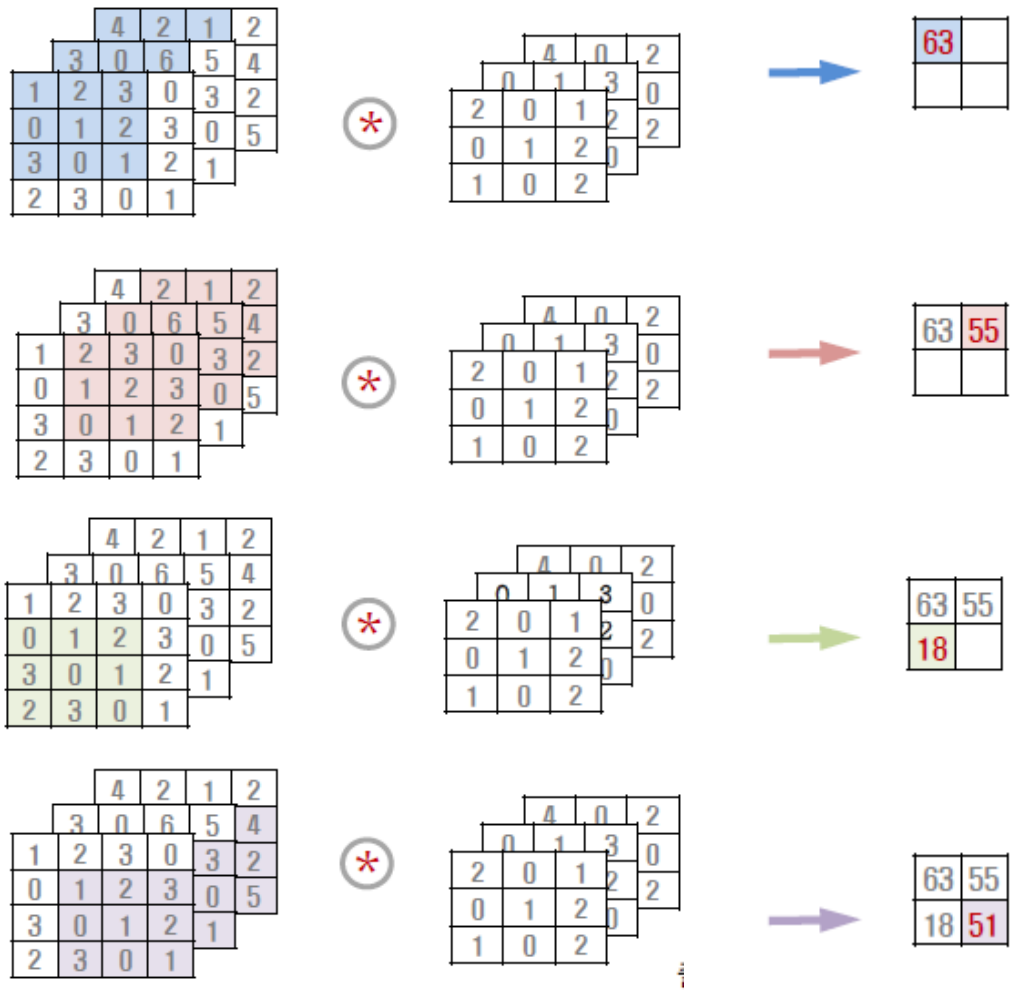


2차원 데이터 합성곱 연산 – 단일 필터

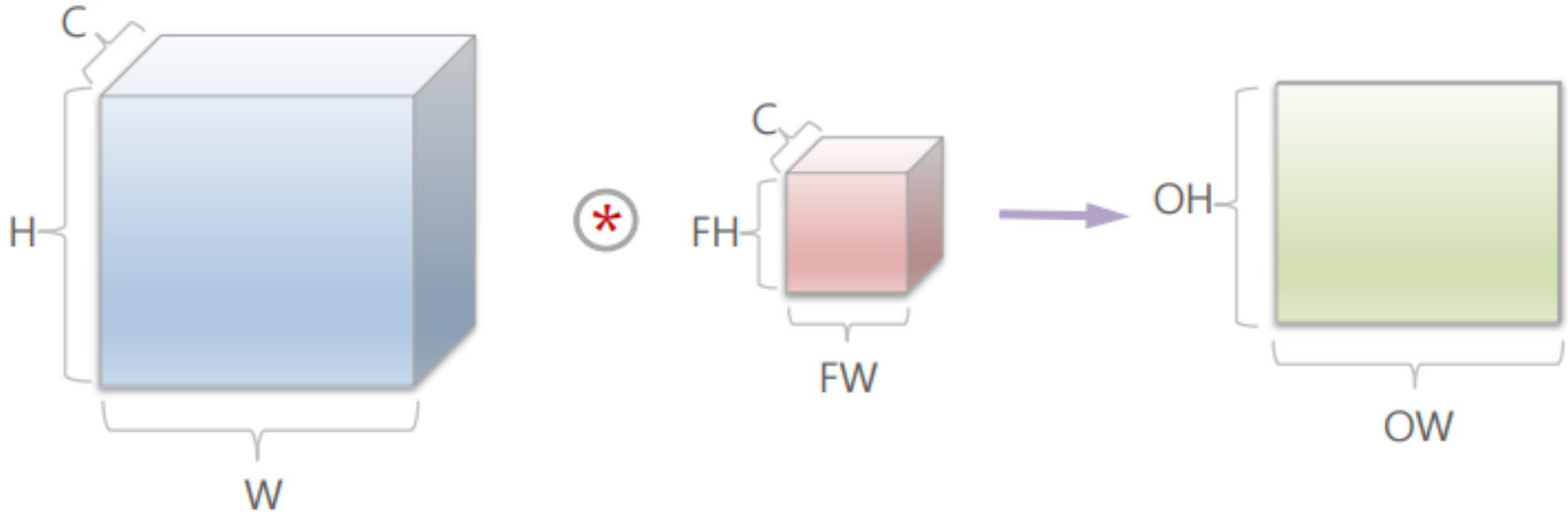
- 입력 데이터의 채널 수와 필터의 채널 수 일치
- 모든 채널의 필터가 같은 크기



2차원 데이터 합성곱 연산 – 단일 필터



3차원 데이터 합성곱 연산 – 단일 필터

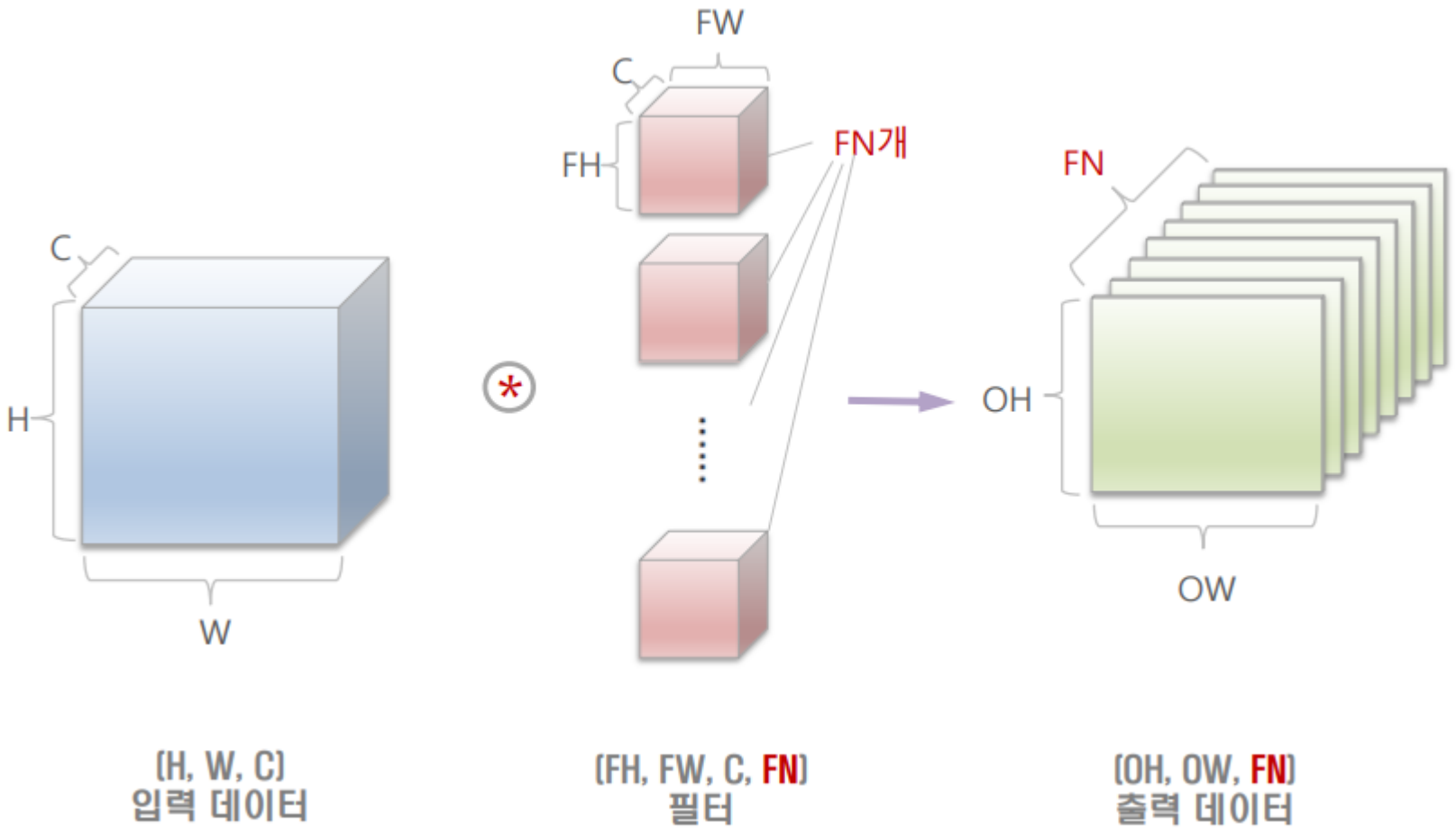


(H, W, C)
입력 데이터

(FH, FW, C)
필터

(OH, OW, 1)
출력 데이터

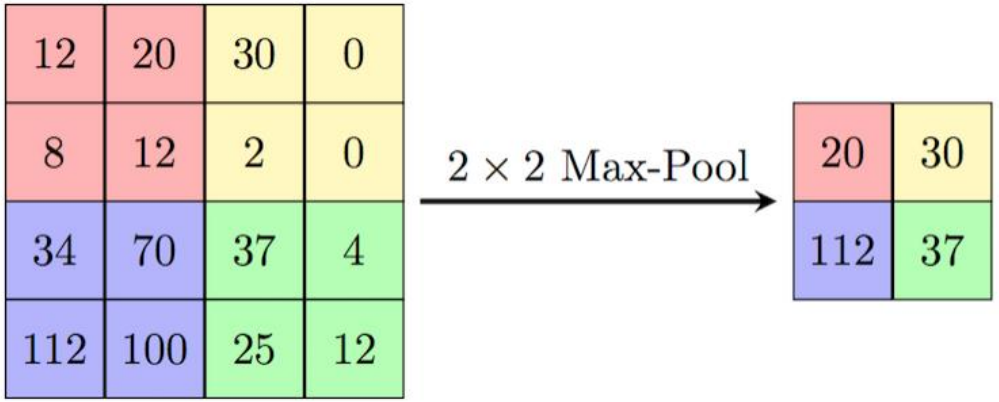
3차원 데이터 합성곱 연산 – 다중 필터



풀링 (Pooling)

- 축소 샘플링은 주로 풀링 작업을 통해 수행

- CNN에서 합성곱 수행 결과를 다음 계층으로 모두 넘기지 않고, 일정 범위 내에서 (예를 들면 2x2 픽셀 범위) 가장 큰 값을 하나만 선택하여 넘기는 방법을 사용



풀링 (Pooling)

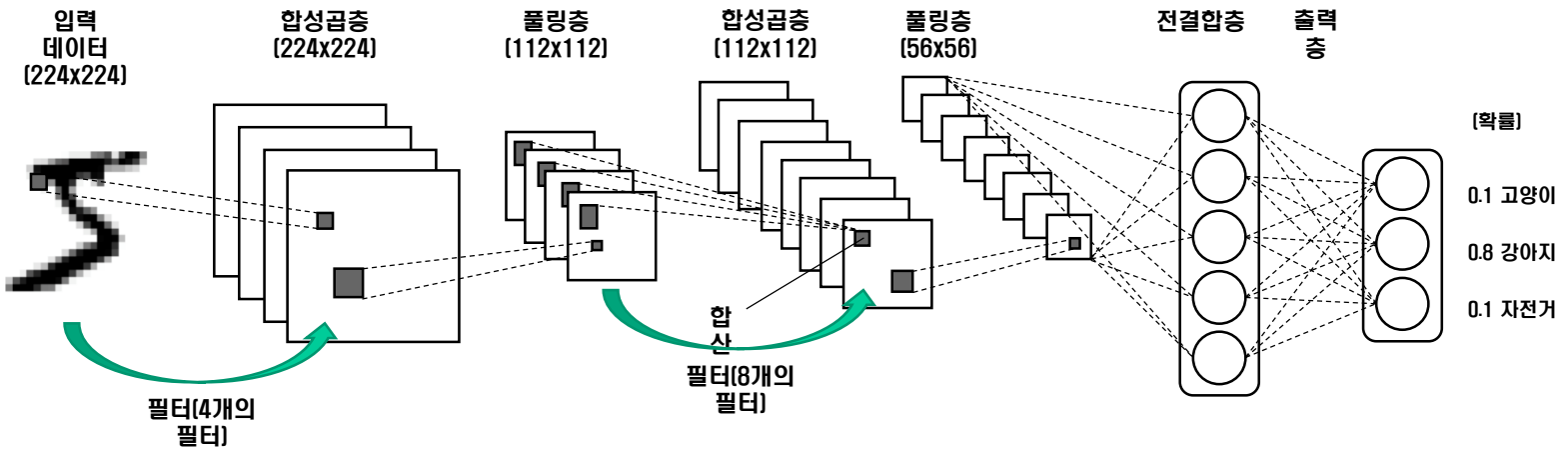
- 최대 풀링(max pooling)
  - 작은 지역 공간의 대표 정보만 남기고, 나머지 신호들을 제거하는 효과
  - 특정한 패턴이 공간상의 어느 위치에 있든 이 활성값이 다음 단계로 넘어가면서 좌우로 조금씩 움직일 수 있는 여지를 준다
  - 이러한 작업을 여러 단계 거치면 위치에 무관하게 특정 패턴의 유효한 값이 최종 출력단의 원하는 위치로 이동할 수 있게 된다.

- 또한 풀링은 과대적합을 해소하는 데에도 기여

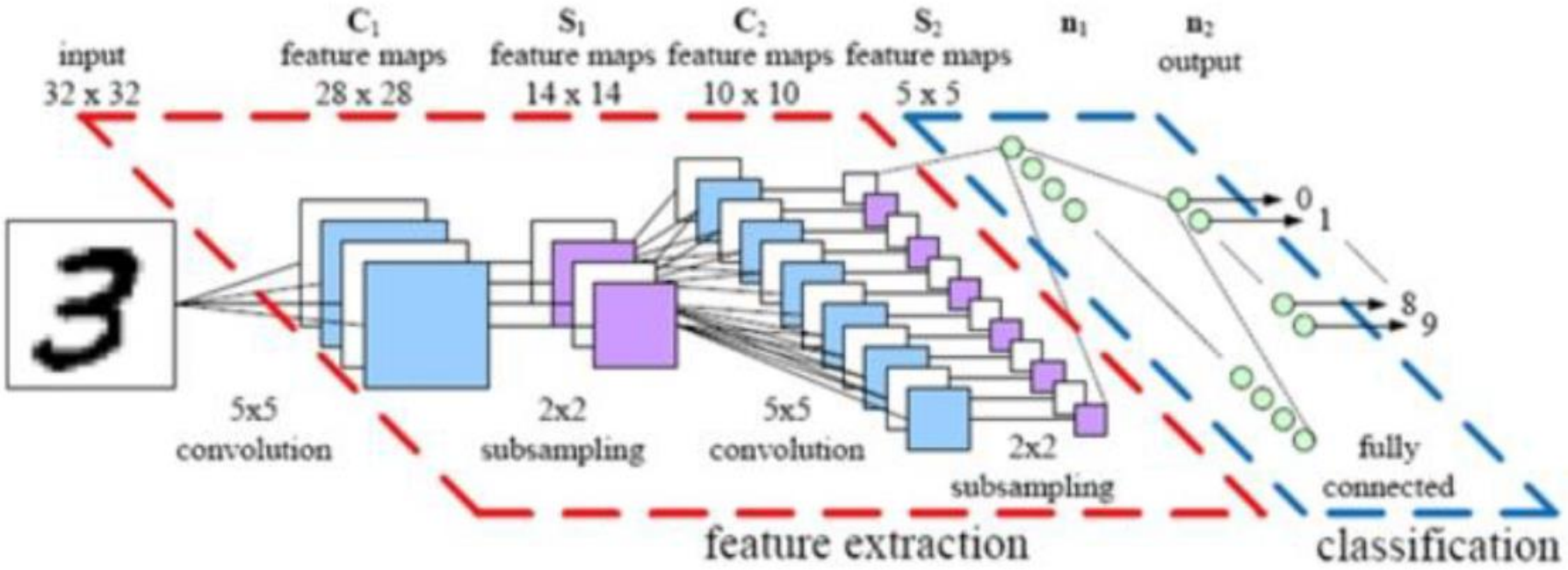
CNN 구성

- CNN은 일반적으로 커널 필터링과 활성화 함수 그리고 최대 풀링을 묶어서 하나의 계층을 형성한다.
  - 참고) MLP에서는 전결합망과 활성화함수를 묶어서 한 계층을 형성
- CNN에서도 다양한 구조의 활성화 함수를 사용하고 필요하면 전결합망을 사용한다.
- 분류를 수행하는 경우 최종 계층에서는 전결합망을 만들고 그 결과에 소프트맥스 함수(또는 시그모이드함수)를 적용

CNN 구성


입력
데이터
(224x224)
합성곱층
(224x224)
풀링층
(112x112)
합성곱층
(112x112)
풀링층
(56x56)
전결합층
출력
층
(확률)
0.1 고양이
0.8 강아지
0.1 자전거
합
산
필터(4개의
필터)
필터(8개의
필터)

CNN 구성


input
32 x 32
C1
feature maps
28 x 28
S1
feature maps
14 x 14
C2
feature maps
10 x 10
S2
feature maps
5 x 5
n1
n2
output
0
1
8
9
5x5
convolution
2x2
subsampling
5x5
convolution
2x2
subsampling
fully
connected
feature extraction
classification

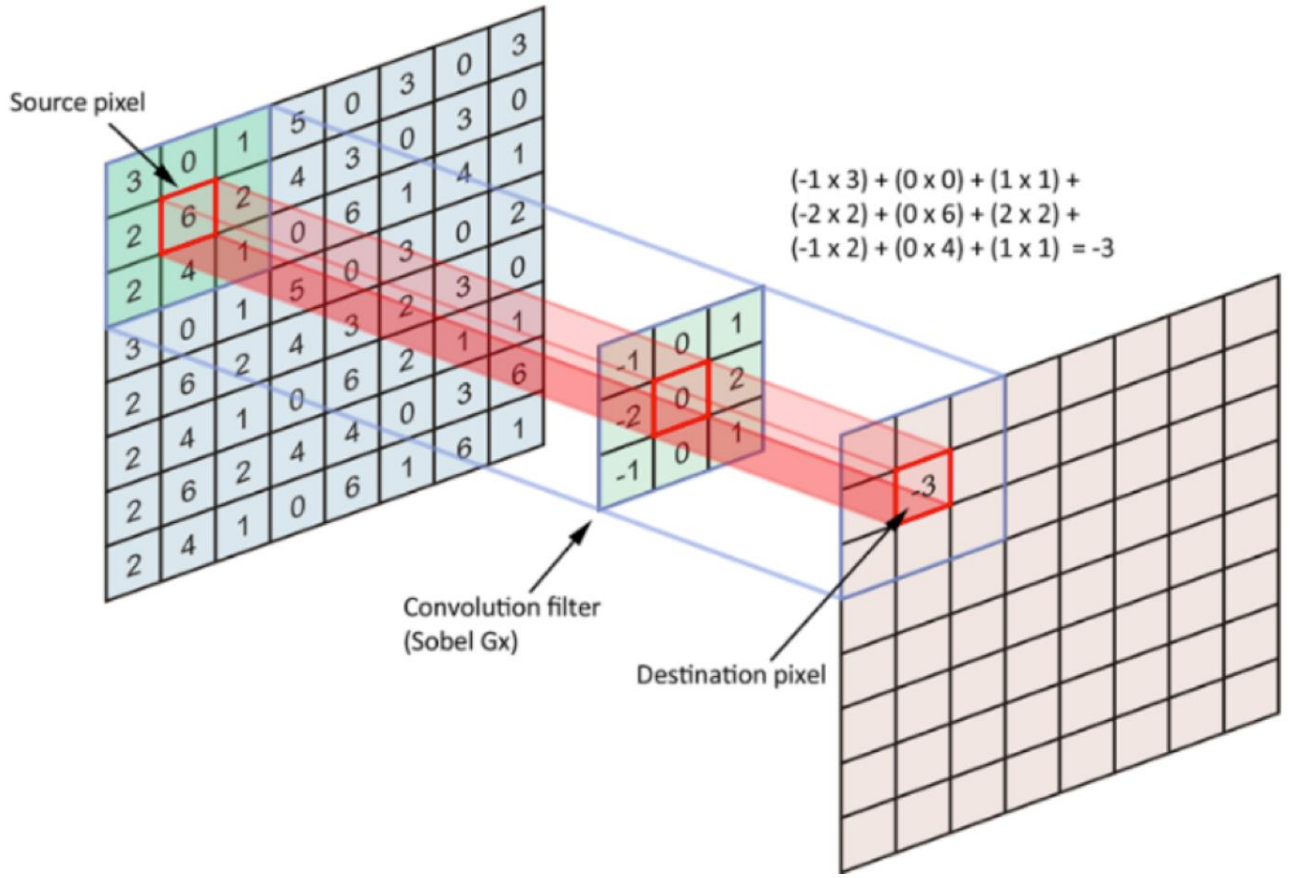
CNN의 특징

- MLP 구조
  - 입력 이미지 전체 픽셀의 특성을 한 번에 학습

- CNN
  - 입력 신호의 지역적인 특성들을 작은 단위로(예, 3x3 픽셀) 나누어 특정한 패턴이 있는지를 학습
  - 패턴으로는 기하학적인 단위 모양(엣지, 대각선, 수평선, 수직선 등)과 질감(texture) 등
  - 이러한 필터링은 스캔하듯이 입력 이미지 전체에 대해서 옆으로 이동하면서 적용
  - 따라서 CNN은 어떤 특정 패턴이 이미지 상에서 나타나는 위치가 변경되어도 이를 다시 찾아낼 수 있다

CNN의 특징

- 합성곱동작
- 다층 필터링

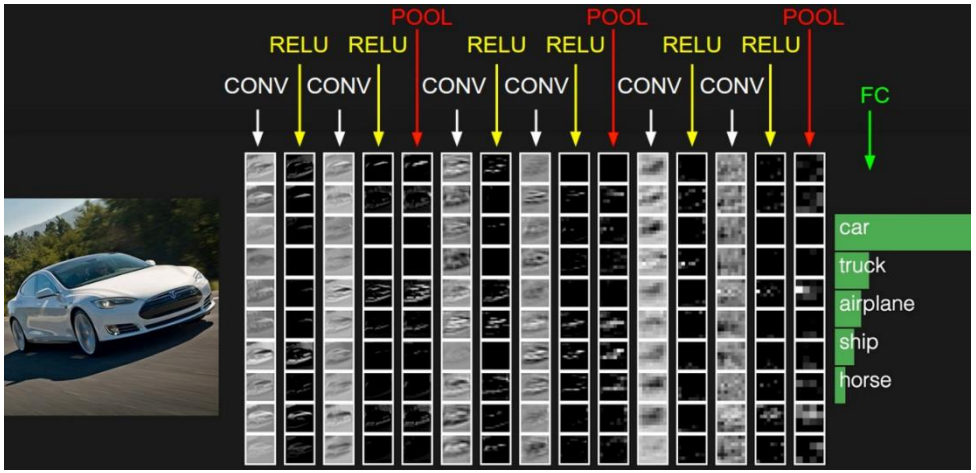


CNN의 특징

- CNN을 여러 계층으로 쌓으면 공간적인 계층구조(hierarchy)를 찾아낼 수 있다. 즉 계층을 올라가면서 점차 추상적인 내용을 파악할 수 있다

- 예를 들어 저층에서는 이미지의 기본적인 패턴을 찾고, 다음 계층에서는 이를 이용한 보다 복잡한 패턴을 찾는다

- 즉, 세밀한 것들이 모여서 큰 그림을 만들 듯이 이미지 전체를 이해하는 데는 이러한 공간적인 구조 파악이 중요한 역할을 한다



특성 맵 (Feature map)

- 평면을 구성하는 2차원 축 데이터와(폭, 높이), 깊이(depth) 축으로 구성 (이를 채널이라 함)

- 입력신호가 RGB 신호로 코딩된 경우, 깊이 축의 차원은 세가지 색을 각각 나타내는 3이 된다.

- 흑백으로 코딩된 경우 (MNIST처럼) 흑백의 그레이 스케일만 나타내면 되므로 깊이는 1이 된다.

- 컨볼류션 동작은 입력 특성맵으로부터 일정 영역을 추출하며 이들 영역들에 대해서 모두 동일한 변환(필터링)을 수행한다. 그 결과로 출력특성맵을 생성한다.

- 이 출력 특성맵도 역시 3D 텐서 구조를 갖는다. 필터링을 할 때 깊이의 크기를 변경할 수 있다. (필터의 개수가 깊이의 차원이 된다.)

특성 맵 (Feature map)

- **여기서** 주의할 것은 출력 특성맵 깊이의 각 채널이 여전히 입력 신호가 가지고 있던 RGB 신호 정보를 유지하는 것이 아니라는 것이다

- 이 채널은 입력 신호를 다양한 필터를 사용하여 필터링한 새로운 신호들의 집합을 나타낸다.

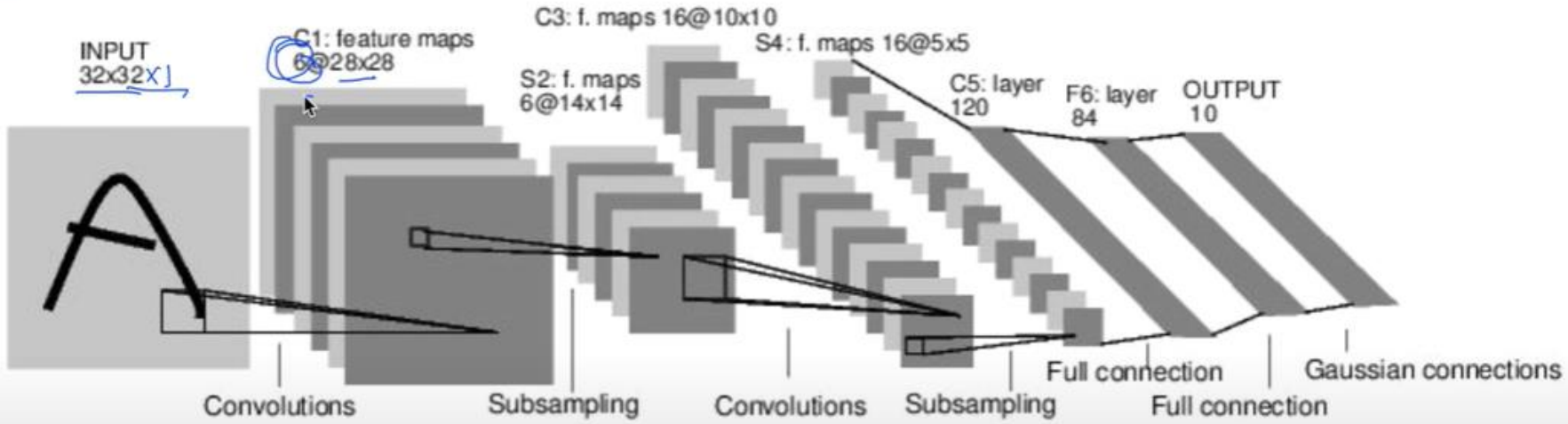
- **출력** 특성맵의 깊이(depth)는 컨볼류션 수행에 사용된 필터의 개수

- 컨볼류션 동작은 이 패치 크기의 윈도우를 옆으로 슬라이딩하면서 필터링 작업을 반복한다. 필터 가중치 매트릭스를 컨볼류션 커널이라고 한다.

- 필터 출력은 1차원 벡터가 되며 크기는 (output_depth,)가 된다. 이러한 1차원 벡터가 공간적으로 배치되어 출력 특성맵을 구성

Case Study: LeNet-5

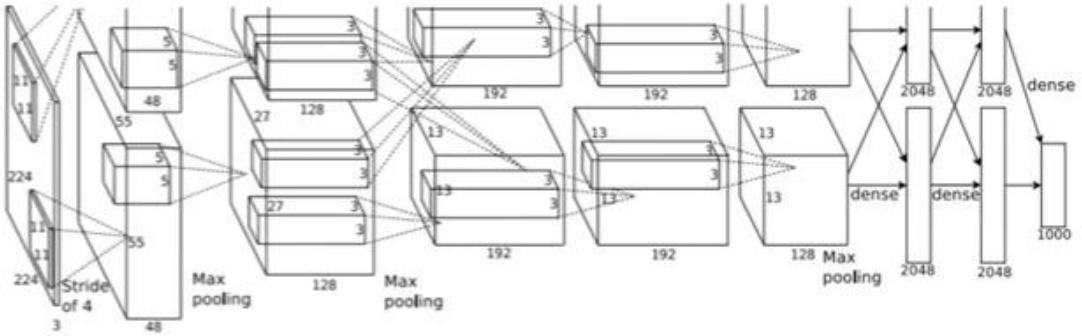
[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Case Study: AlexNet

[Krizhevsky et al. 2012]

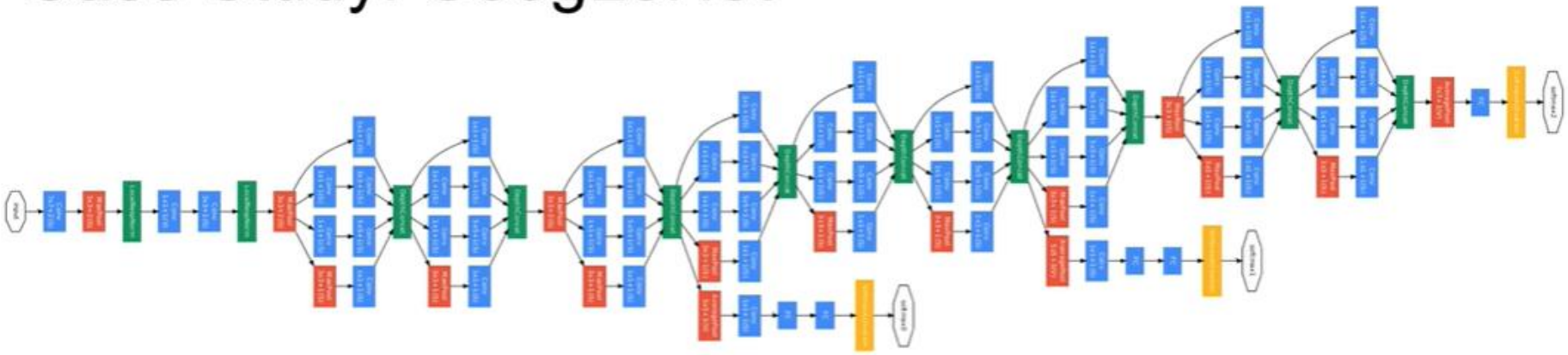


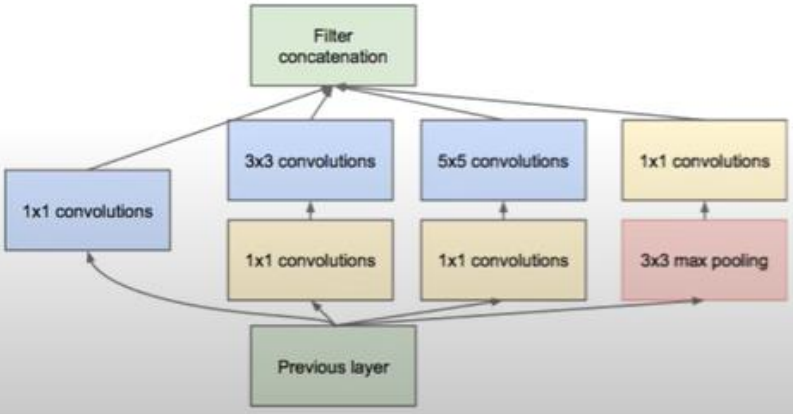
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Case Study: GoogLeNet

[Szegedy et al., 2014]



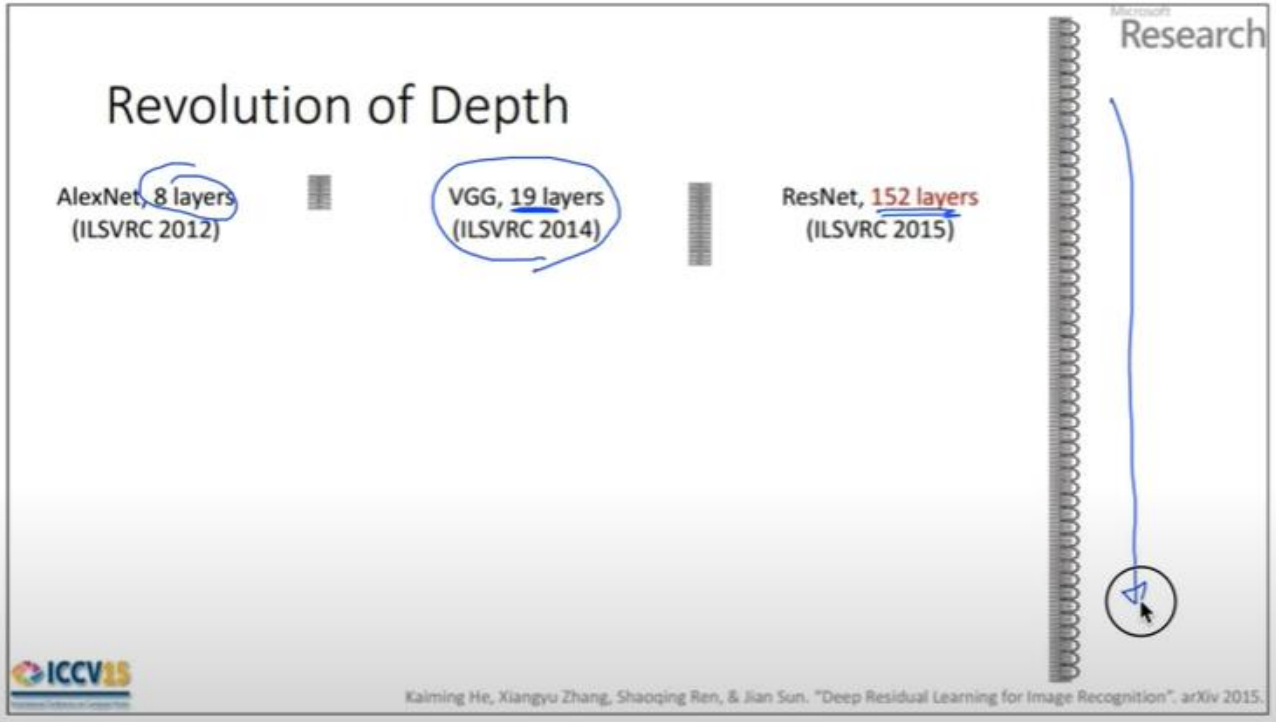


Inception module

ILSVRC 2014 winner (6.7% top 5 error)

Case Study: ResNet *[He et al., 2015]*

ILSVRC 2015 winner (3.6% top 5 error)



2-3 weeks of training on 8 GPU machine

at runtime: faster than a VGGNet! (even though it has 8x more layers)

(slide from Kaiming He's recent presentation)

신경망 성능 개선

과대 적합 (Overfitting)

- 신경망은 파라미터 갯수가 많아서 과대적합할 가능성이 항상 높다. 훈련 데이터 양이 많지 않을 때에는 특히 주의해야
  - 과대적합이 발생하면 신경망의 구조를 단순하게 만들어야 한다.
  - 과소적합 여부는 훈련 데이터에 대한 성능이 얼마나 낮은지 (평균치 수준에 머무는지)를 보고 판단

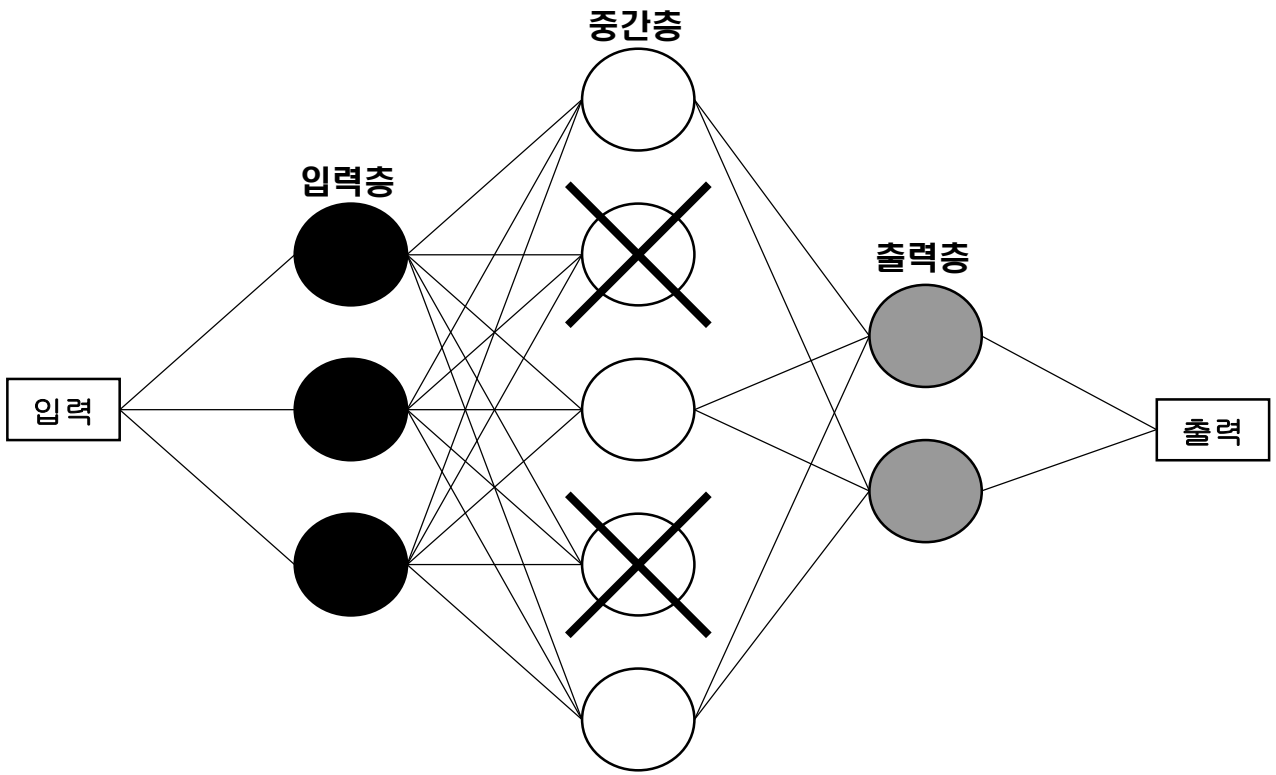
- 과대적합을 줄이는 작업을 일반화(generalize)를 위한 규제화 (regularize) 라고 부르는데,
  - 대표적인 규제화로 L2규제를 적용
    - 네트워크를 구성하는 파라미터 값이 가능한 균등하게 분포하도록 제한
  - 케라스에서는 regularizers.l2() 함수를 사용

드롭 아웃 (Drop out)

- 신호가 계층을 통과할 때 랜덤하게 유닛을 선택하여 신호를 전달하지 않는 방법
- 즉, 모든 신호를 다 사용하지 않고 고의적으로 신호의 일부를 누락시킨다
- 앙상블 효과를 얻어서 과대적합을 효과적으로 줄일 수 있다.
- 드롭아웃을 하면 입력과 출력간의 특정한 관계를 기억하지 못하게 하는 효과가 있어서 신경망이 보다 일반적인 학습을 할 수 있게 한다
- 네트워크의 기억을 랜덤하게 지우는 것이라고 볼 수 있다. 입출력의 특별한 관계를 평준화시키고 다양한 신경망 구조를 이용한 효과(즉, 앙상블 효과를 얻어 성능을 개선하는 방법)

드롭 아웃

- 드롭아웃은 학습을 하는 동안에만 적용
- 학습이 종료된 후 예측을 하는 단계에서는 모든 유닛을 사용하여 예측



출력단 활성화 함수 선택

- 분류를 하는 경우 대부분 소프트맥스 함수를 사용
  - 특정한 카테고리에 속할 확률을 구해준다
  - 즉, 신경망이 하나의 카테고리만 선택할 때는 소프트맥스를 사용

- **복수의 답이 가능한 분류문제**에서는 **시그모이드 함수**를 출력단에 사용하는 것이 나을 수가 있다
  - 시그모이드 함수를 사용하면 각 출력이 0~1의 값을 가지며
  - **출력들의 합은 1을 넘을 수가 있다**

- 회귀분석에 사용될 때에는 출력단에 선형함수(즉, 신호를 그대로 통과시키는 것)를 사용

최적의 신경망 구조

- 최적의 신경망을 구성하는 계층의 수, 유닛의 수, 배치 크기, 학습률의 설정 등이 어려운 과제

- 기본 전략 : 처음에는 구조를 간단히 출발

- 일단 동작을 확인하고 성능을 개선한다.
  - 계층이 2~3만으로도 동작하는지를 확인
  - 만일 2~3개의 계층으로 모델이 동작하지 않으면 계층 수를 늘려도 동작하지 않는다고 알려져 있다.

- 입력 데이터를 간단히 만들어 보는 것도 필요
  - 예를 들어 10가지 동물 이미지를 구분하는 모델이 필요하다고 하여도 우선 몇 가지 대표적인 동물들을 구분하는 모델을 먼저 만들어보는 것

배치 (Batch) 크기

- 배치 크기 : 신경망이 한번에 학습하는 입력 데이터 수

- 배치 크기가 클수록 학습이 정교하고, 기울기를 정확히 구할 수 있으나 계산량이 많아진다.
  - 필요한 메모리 사용량이 많아 메모리 오류가 날 가능성이 높다(특히 GPU를 사용할 때). 처음에는 배치 크기를 작게 16~32정도로 적게 잡고 시작

- 배치 크기가 작을 때에는 기울기가 상대적으로 정확하게 계산되지 못하므로 학습률도 작게 잡아야 한다.
  - 일단 작은 값의 학습률로 동작하는 것을 확인하고 학습률을 조금씩 크게 한다

배치 정규화 (Batch Normalization)

- 딥러닝 학습에서 해결해야할 가장 어려운 문제
  - Vanishing Gradient Problem

- 이는 기울기 값에 비례하여 학습시킬 때 기울기 즉, 미분값이 0에 가까워지면 변화량이 매우 적어지고 이것이 앞단의 계층으로 전파되는 양이 급속히 줄어들어 학습이 잘 되지 않는 현상

- 이러한 문제를 해결하기 위해 배치 정규화가 제시됨
  - 이는 "계층별로", 주어진 배치 데이터를 대상으로 정규화를 다시 수행하여 데이터의 분포가 너무 작거나 너무 커지지 않게 하는 방식
  - **학습 시 미니배치를 단위로 정규화 : (평균 0, 분산 1)**

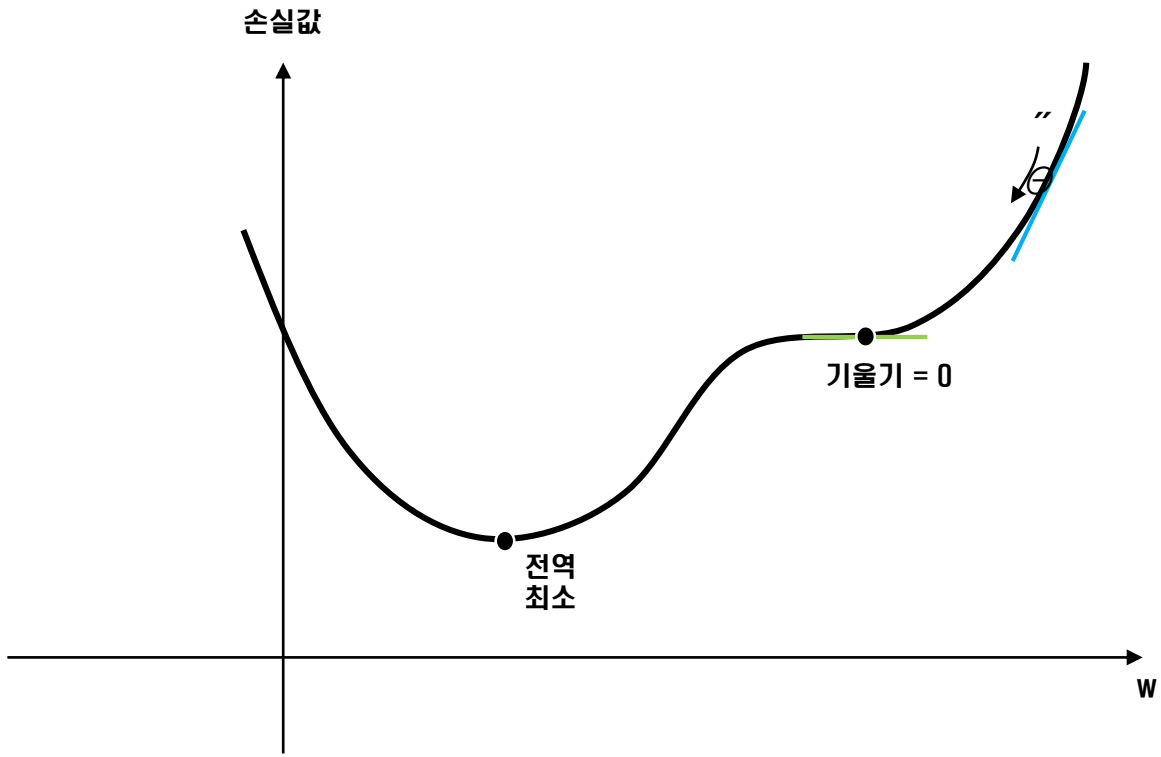
모멘텀 (Momentum) 알고리즘

- 머신러닝에서는 최적화 알고리즘으로 SGD이 널리 사용된다.

- 신경망에서는 좀 더 정교한 방법으로 모멘텀 기법을 사용한다.
  - 모멘텀 기법이란 기울기 뿐 아니라 가속도 항을 고려하여 움직이던 방향으로 계속 움직이려는 관성을 반영한 것이다.
  - 현재의 기울기(gradient) 에 비래하는 학습을 할 뿐 아니라 여기에 더해서 기울기의 “변화량”도 반영하게 된다.

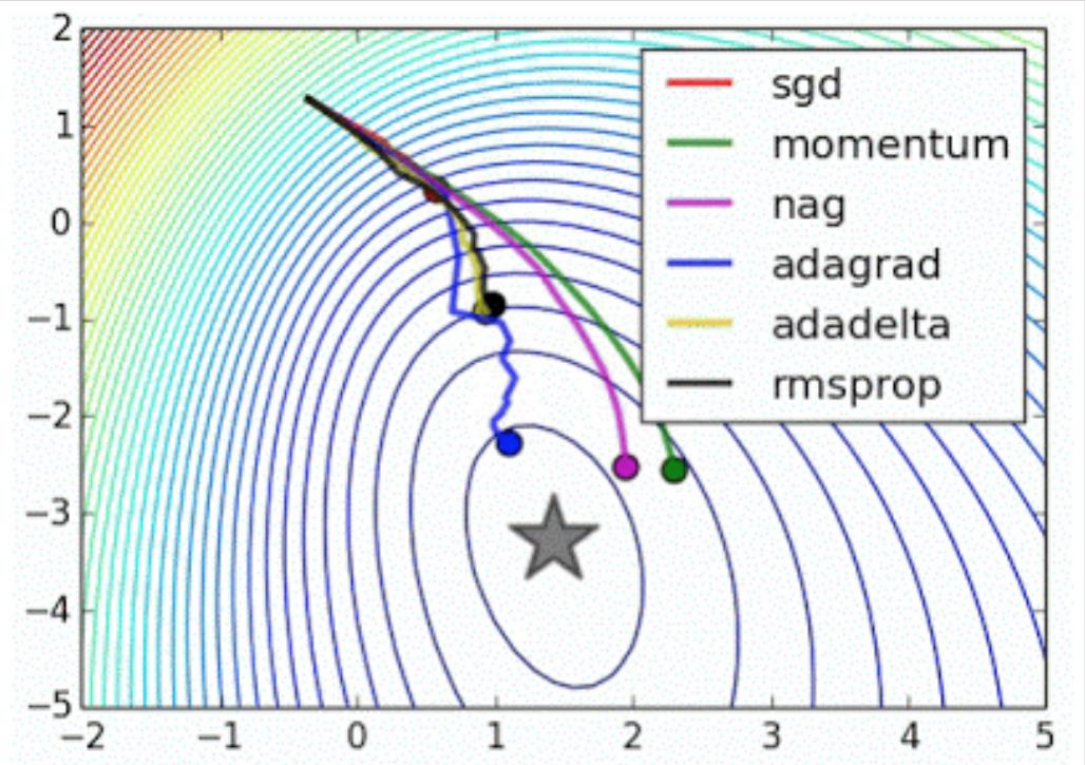
- 학습률을 적응형으로 감소하는(adaptive gradient) 방식인 AdaGrad도 널리 사용된다.
  - 이 방법에서는 학습률을 서서히 낮춘다.
  - 모멘텀 기법과 적응형 방법을 조합한 방법으로 Adam이 2015년 소개되었다. 현재 가장 널리 사용되고 있다.

모멘텀 (Momentum) 알고리즘



최적화 (Optimization) 알고리즘 비교

- 최적화 동작
  - https://goo.gl/Pdu4uW



최적화 (Optimization) 알고리즘

- 가속도 방식을 도입
  - 지역 최소를 지나가게 하여 전역 최소를 찾을 수 있게 한다
  - Momentum
  - Nesterov Momentum
  - Adam
  - RMSProp

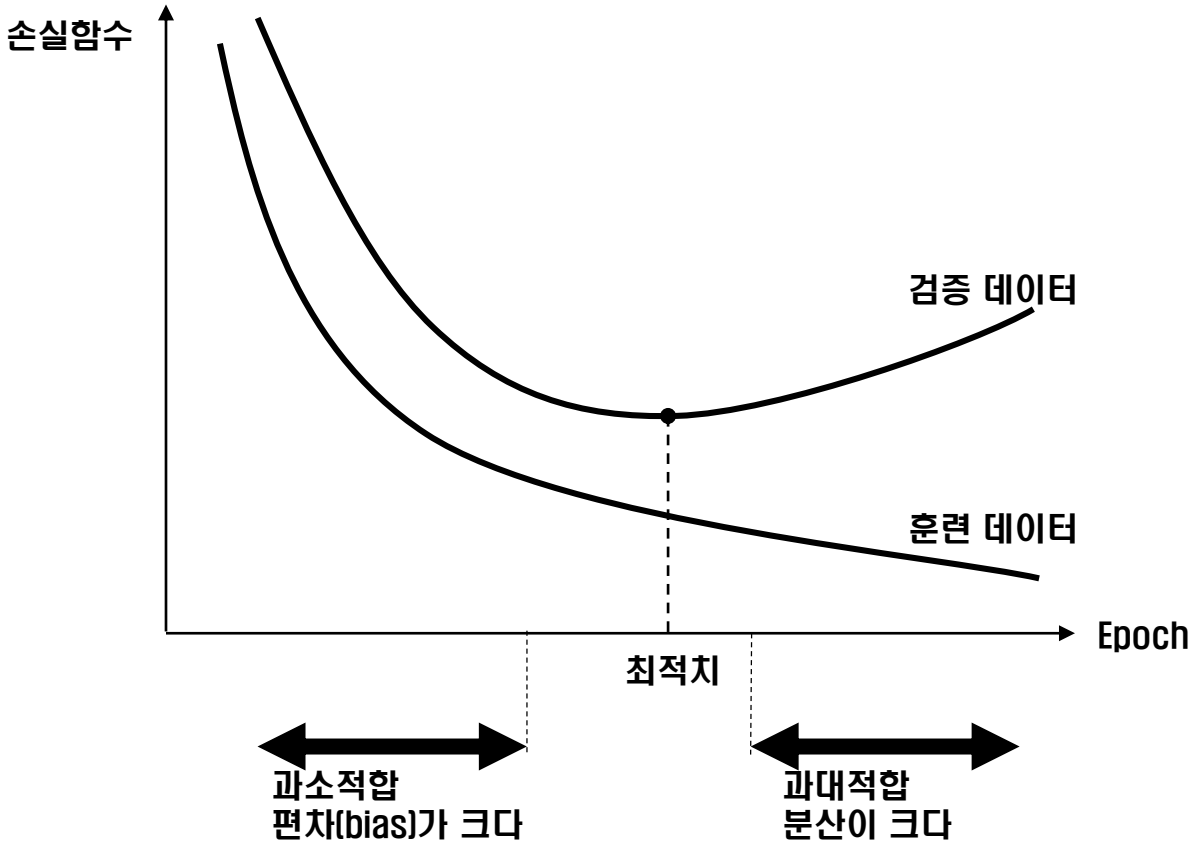
과대적합 검증

- 신경망은 파라미터가 많아서 과대적합되기 쉽다. 즉, 상세한 모델링이 가능하여, 훈련 데이터 수가 적으면 이 훈련 데이터의 속성을 모두 기억할 수 있어 과대적합되기 쉬운 것이다.

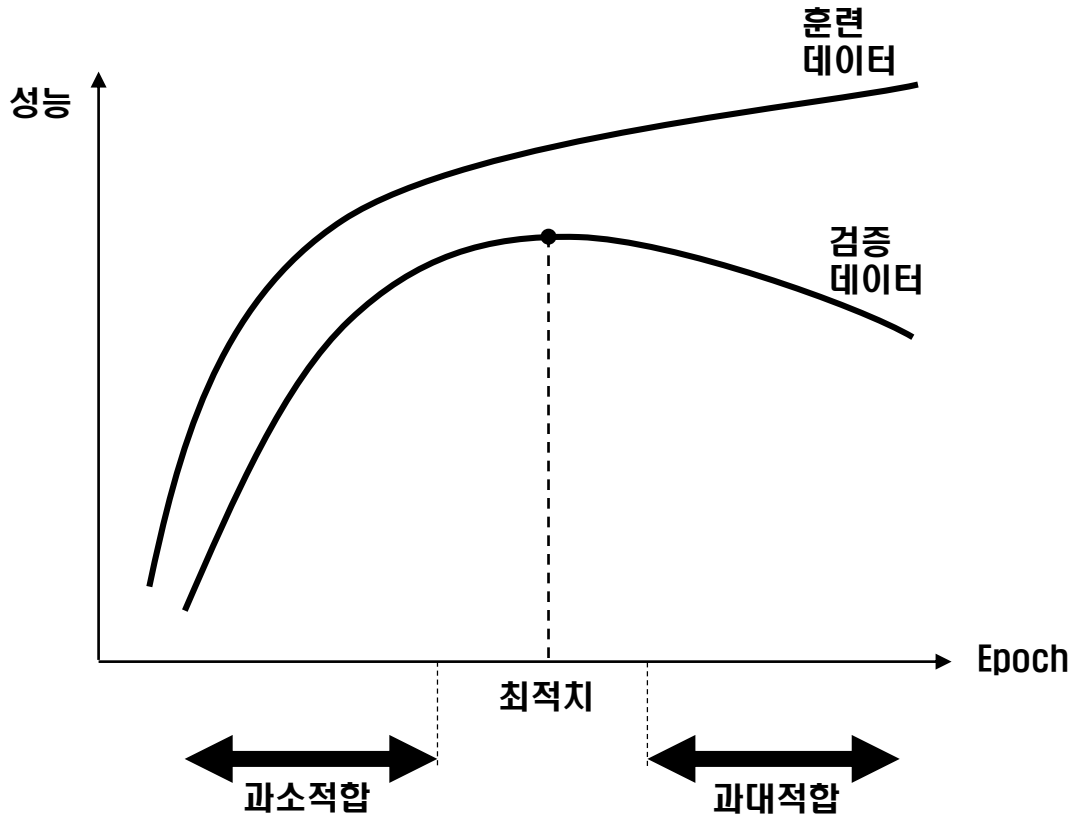
- 훈련데이터에 대해서는 계속 성능이 좋아지지만 검증 데이터에 대해서는 오히려 성능이 나빠진다면 과대적합한 것이다.

- 성능의 개선되는지를 측정하는 방법
  - 정확도 등 최종 성능 지표를 관찰하는 방법
  - 손실함수가 줄어드는지를 관찰하는 방법이 있다.

이포크 (Epoch) 증가에 따른 손실함수의 변화



이포크 (Epoch) 증가에 따른 성능의 변화



과대 적합을 피하는 방법

- 학습조기중단(early stopping)
- L1 이나 L2 규제
- 드롭아웃(dropout)
- 데이터 확장(data augmentation)
  - 훈련 데이터가 많은 것처럼 보이는 효과

데이터 확장 (Data Augmentation)

- 과대적합이 일어나는 이유 중 하나
  - 훈련데이터가 부족하기 때문

- 훈련 데이터가 충분히 많다면 과대적합을 줄일 수 있다.

- 데이터 확장이라 훈련 데이터를 다양하게 변형하여 변형된 새로은 훈련 데이터처럼 사용함으로써 마치 훈련 데이터 수가 늘어난 효과를 얻는 것이다.

- 데이터 확장을 사용하면 여러 이포크를 수행해도 똑같은 데이터를 가지고 학습하지 않게 된다.

데이터 확장 (Data Augmentation)

- rotation: 0° 에서 360° 사이에서 회전
- shifting: 랜덤하게 상하좌우로 이동
- rescaling: 랜덤하게 1.0 ~ 1.6배로 사진 확대
- flipping: 좌우, 또는 상하로 반전
- shearing: -20° 에서 20° 도 사이에서 왜곡
- stretching: 1.0 ~ 1.3배로 확장

데이터 확장 예

- “레이블링 된” 학습 데이터가 부족한 문제를 해결

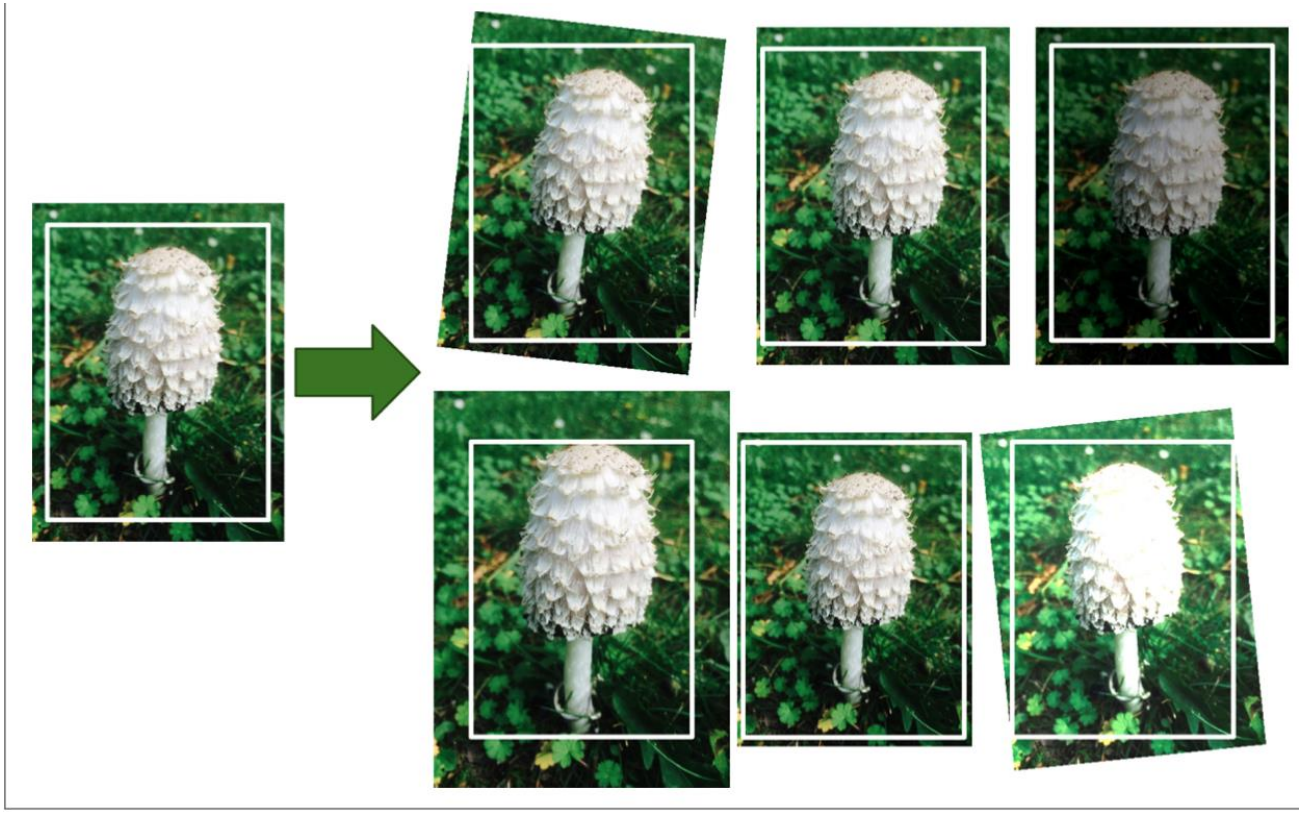
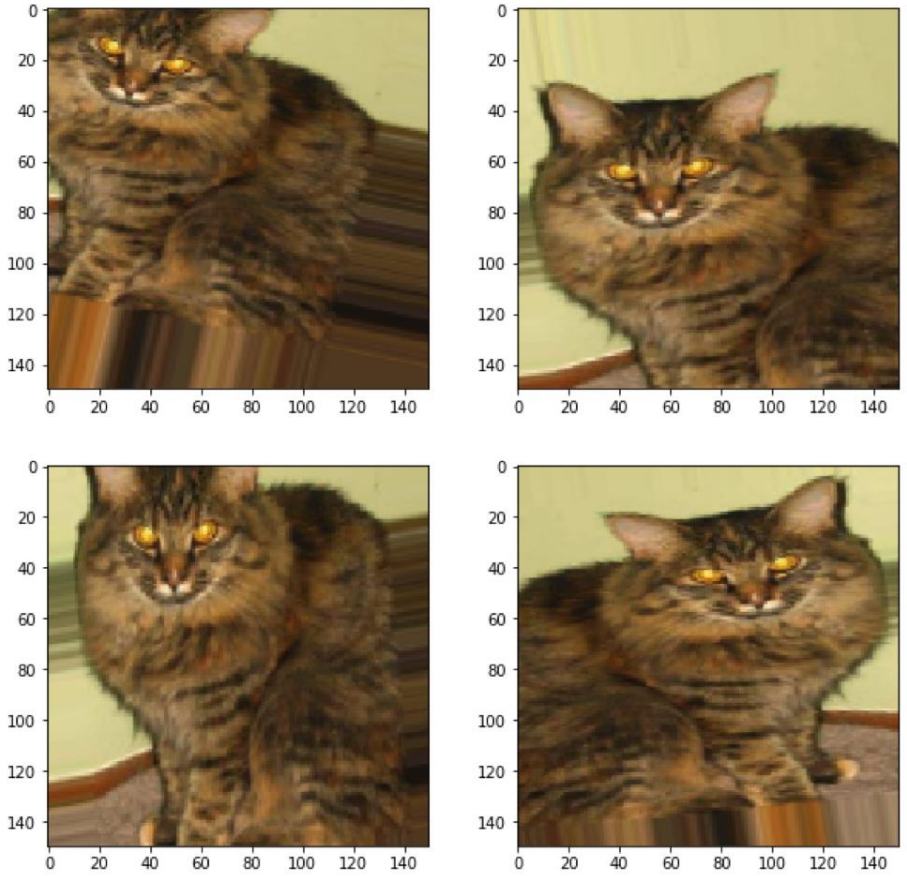


Figure 11-10. Generating new training instances from existing ones

데이터 확장 예

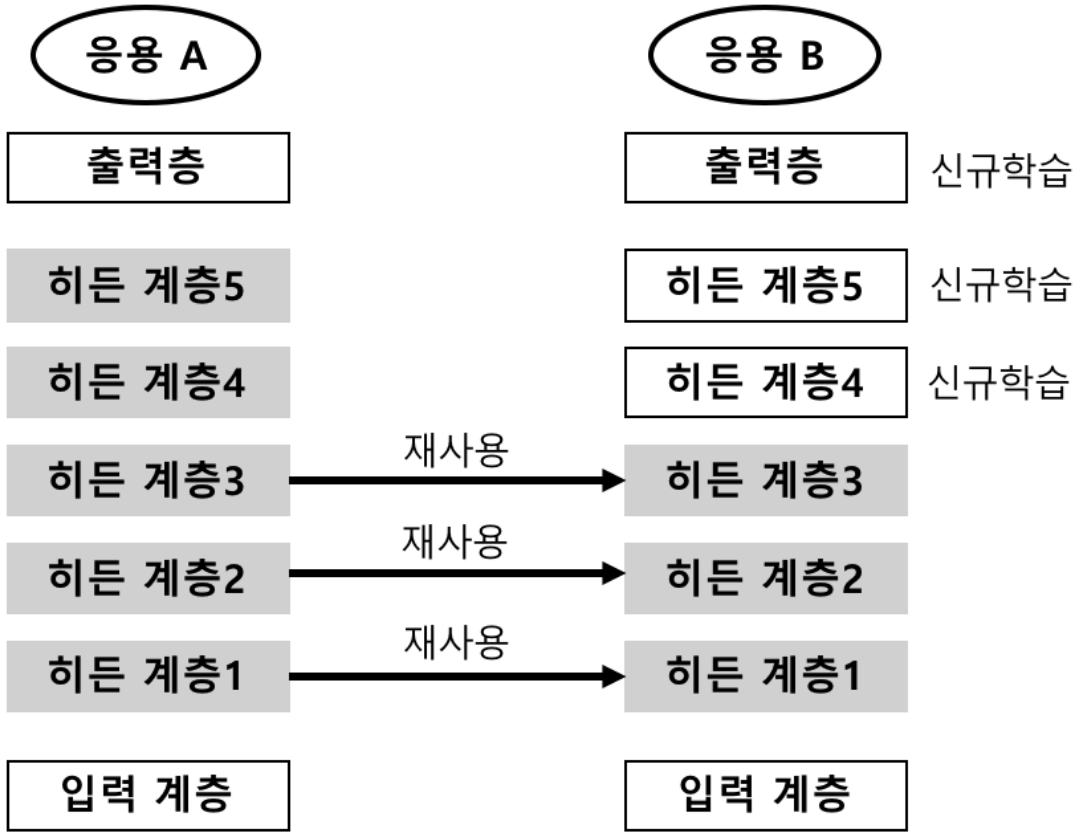


전이 학습

전이 학습 (Transfer Learning) 정의

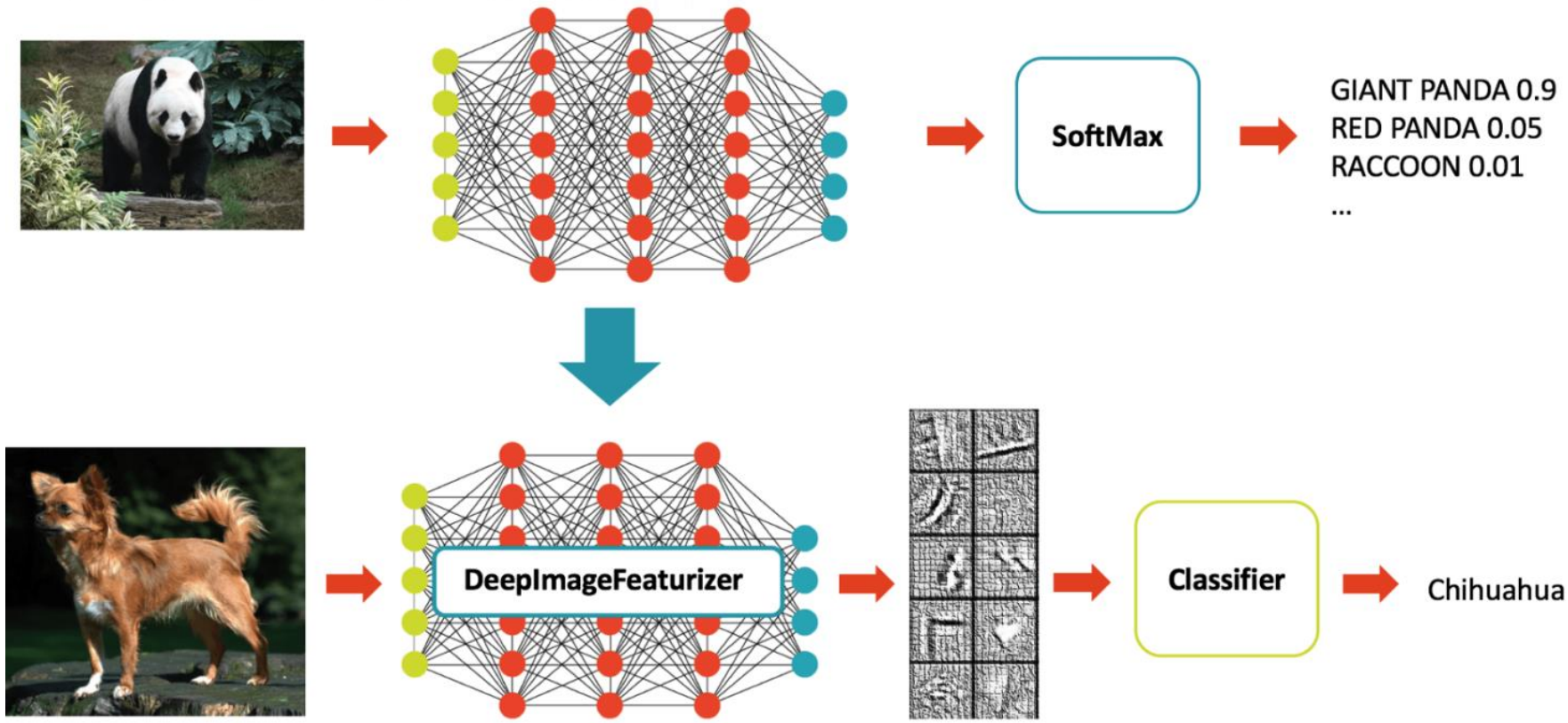
- 전이학습 이란 다른 데이터 셋을 사용하여 이미 학습한 모델을 유사한 다른 데이터를 인식하는데 사용하는 기법이다.
- 이 방법은 특히 새로 훈련시킬 데이터가 충분히 확보되지 못한 경우에 학습 효율을 높여준다.
- 이미지넷에는 동물이나 일상생활의 물건들을 주로 포함하여 1000종의 이미지를 갖고 있으며 140만장의 사진이 있다.
- 이미지넷에는 고양이 강아지를 포함한 많은 동물 이미지도 들어 있으며, 이를 강아지 고양이 분류 문제에 사용할 수 있다.
  - 여기서는 2014년에 소개된 VGG16 모델을 사용하겠다.
- 사전학습모델을 이용하는 방법은 특성 추출(feature extraction) 방식과 미세조정(fine-tuning) 방식이 있다.

전이 학습


응용 A
응용 B
출력층
출력층
신규학습
히든 계층5
히든 계층5
신규학습
히든 계층4
히든 계층4
신규학습
재사용
히든 계층3
히든 계층3
재사용
히든 계층2
히든 계층2
재사용
히든 계층1
히든 계층1
입력 계층
입력 계층

전이 학습

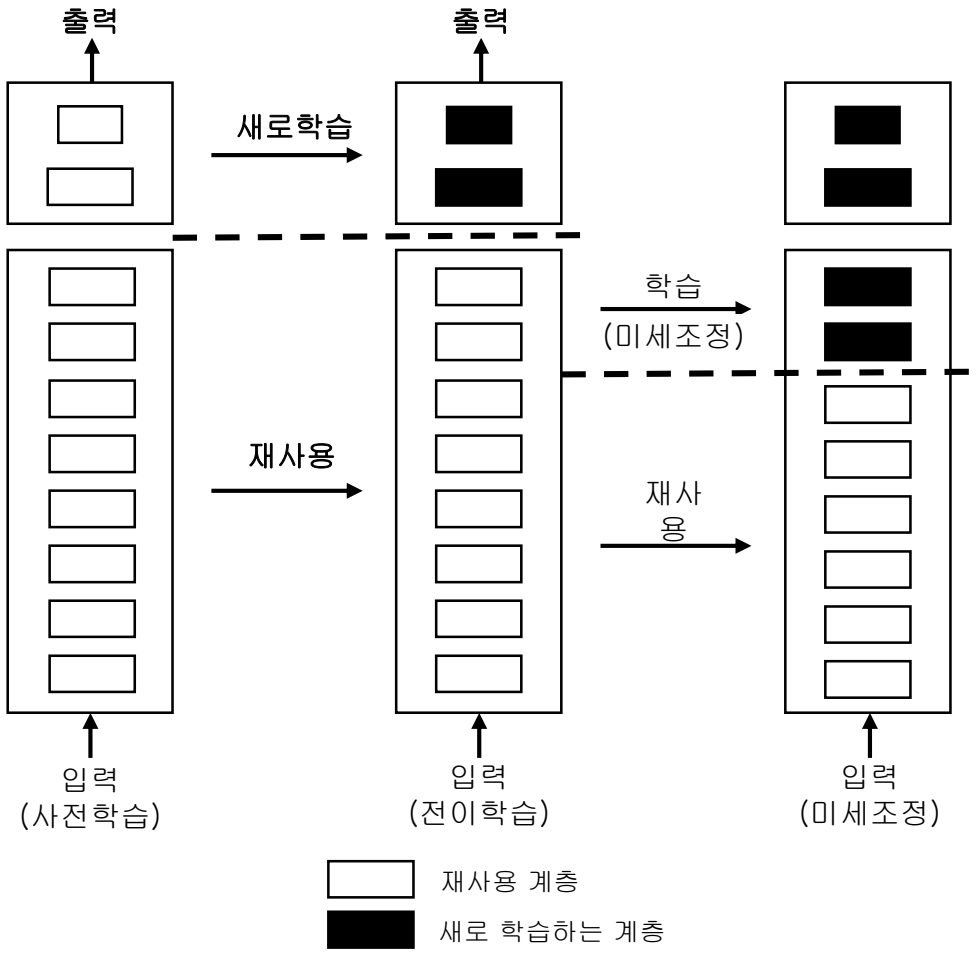
전이 학습


SoftMax
GIANT PANDA 0.9
RED PANDA 0.05
RACCOON 0.01
...
DeepImageFeaturizer
Classifier
Chihuahua

특성 추출 방식

- 특성 추출이란 이전의 네트워크로부터 배운 표현 방식을 사용하여 새로운 데이터 샘플에서 추가로 흥미로운 특성들을 추출하는 것
- 이렇게 얻은 특성들을 사용하여 새로운 분류기를 작동시키고 학습을 수행한다.
- 이미지 분류기는 컨볼류션 네트워트와 풀링 계층의 조합 그리고 전결합층 분류기 등 크게 두 부분으로 구성된다.
- 컨볼류션과 풀링으로 구성된 부분을 컨볼류션 베이스라고한다
- 특성 추출 방식은 컨볼류션 베이스를 그대로 사용하고, 새로운 데이터를 여기에 적용하여 훈련시키는 방식이다.

특성 추출 방식


출력
출력
새로학습
학습
(미세조정)
재사용
재사
용
입력
(사전학습)
입력
(전이학습)
입력
(미세조정)
재사용 계층
새로 학습하는 계층

특성 추출 방식

- 컨볼류션 베이스 부분만 재사용하는 이유는 이 부분은 상당히 일반적인 학습정보를 포함하고 있기 때문이다.

- 컨볼류션 계층에서도 재사용할 수 있는 정보의 수준은 몇 번째 계층인지에 따라 다르다. 모델의 앞단의 계층일수록 에지, 색상, 텍스처 등 일반적인 정보를 담는다.

- 반면에 뒷 부분의 깊은 계층일수록 추상적인 정보를 담는다 (예를 들어 고양이 귀, 강아지 귀 등)

- 새롭게 분류할 클래스의 종류가 사전 학습에 사용된 데이터와 특성이 매우 다르면, 컨볼류션 베이스 전체를 재사용해서는 안되고 앞단의 일부 계층만을 재사용해야 한다.

미세 조정 (Fine tuning) 방식

- 모델 베이스 중 상위 몇개의 계층은 전결합층 분류기와 함께 새로 학습시키는 방식이다.
- 최종 분류기의 계수가 랜덤하게 초기화 되어 있으므로 이를 먼저 학습시키며 이때 VGG16 모델의 컨볼류션 베이스를 초기에는 고정해야 한다.
- 먼저 분류기를 계수를 학습시킨 다음에 (즉, 이 동안은 미세조정을 하지 않도록 상위계층의 계수를 고정시켜 두고), 그 이후에 미세조정을 해야 한다.
- 처음부터 베이스 상위계층의 계수를 같이 훈련시키면 분류기에서 발생하는 큰 에러 값으로 인해, 사전 학습된 정보가 많이 손실된다.

미세 조정 절차

1) 사전 학습된 기본 네트워크 상단에 새로운 네트워크를 추가한다.
2) 기본 네트워크를 고정시킨다.
3) 새로 추가한 부분을 학습시킨다.
4) 기본 계층 중에 학습시킬 상위 부분의 고정을 푼다
5) 고정을 푼 계층과 새로 추가한 계층을 함께 훈련시킨다.

- 미세 조정을 천천히 수행하기 위해서 느린 학습 속도를 선택한다. 갑자기 큰 변화를 주면 사전 학습된 내용이 훼손되기 때문이다.

콜백(Callback)

- 케라스, 콜백 함수
  - 모델 학습을 하는 동안 중간 결과를 저장하거나
  - 조기 종료를 시키거나 할 수 있는 기능

- 콜백은 여러 가지를 동시에 지정할 수 있으며 이를 fit 함수의 callbacks 인자로 넘겨줄 수 있다.

콜백(Callback) – 주요 함수

- History()
  - 훈련중에 발생하는 여러 이벤트를 History 객체에 저장
  - keras.callbacks.History()

- 모델 저장: 매 이포크마다 모델을 저장
  - 저장하는 주기(period)와 베스트 모델만 저장 등을 지정

```
keras.callbacks.ModelCheckpoint(filepath,
                monitor='val_loss', verbose=0,
                save_best_only=False,
                save_weights_only=False, mode='auto', period=1)
```

콜백(Callback) – 주요 함수

- 조기 종료
  - 손실값, 성능 등 관찰 중인 지표가 어떤 조건을 만족하면 이포크 실행을 종료
  - 조기 종료 조건들을 인자로 지정

```
keras.callbacks.EarlyStopping(
        monitor='val_loss',
        min_delta=0, patience=0, verbose=0, mode='auto',
        baseline=None, restore_best_weights=False)
```

- 참고) https://keras.io/callbacks/#modelcheckpoint

실습

신경망 설계 – keras 주요 특징

- 모듈화 (Modularity)
  - 제공하는 모듈은 독립적으로 설정 가능하며, 가능한 최소한의 제약사항으로 서로 연결되어 있고 모델은 시퀀스 또는 그래프로 이러한 모듈들을 구성
  - 특히 신경망 층, 비용함수, 최적화기, 초기화기법, 활성화함수, 정규화기법은 모두 독립적인 모듈이며, 새로운 모델을 만들기 위해 이러한 모듈을 조합할 수 있다
- 최소주의 (Minimalism)
  - 각 모듈은 짧고 간결
  - 모든 코드는 한 번 훑어보는 것으로도 이해가능한 수준
  - 단 반복 속도와 혁신성에는 다소 떨어질 수가 있음

- 참고) https://tykimos.github.io/2017/01/27/Keras_Talk/

신경망 설계 – keras

- 쉬운 확장성
  - 새로운 클래스나 함수로 모듈을 아주 쉽게 추가
  - 따라서 고급 연구에 필요한 다양한 표현이 가능
- 파이썬 기반
  - Caffe 처럼 별도의 모델 설정 파일이 필요없음
  - 파이썬 코드로 모델들이 정의됨

신경망 설계 – keras

- 데이터셋 생성하기
  - 원본 데이터를 불러오거나 시뮬레이션을 통해 데이터를 생성
  - 데이터로부터 train, validation, test 데이터셋을 생성
  - 이 때 딥러닝 모델의 학습 및 평가를 할 수 있도록 포맷 변환

- 모델 구성하기
  - 시퀀스 모델을 생성한 뒤 필요한 레이어를 추가하여 구성
  - 좀 더 복잡한 모델이 필요할 때는 케라스 함수 API를 사용

- 모델 학습과정 설정하기
  - 학습하기 전에 학습에 대한 설정을 수행
  - 손실 함수 및 최적화 방법을 정의
  - 케라스에서는 compile() 함수를 사용

신경망 설계 – keras

- 모델 학습시키기
  - 훈련셋 train 데이터셋을 이용하여 구성한 모델로 학습
  - 케라스에서는 fit() 함수를 사용
- 학습과정 살펴보기
  - 모델 학습 시 train 데이터셋, validation 데이터셋의 손실 및 정확도를 측정
  - 반복횟수에 따른 손실 및 정확도 추이를 보면서 학습 상황을 판단
- 모델 평가하기
  - 준비된 test 데이터셋으로 학습한 모델을 평가
  - 케라스에서는 evaluate() 함수를 사용
- 모델 사용하기
  - 임의의 입력으로 모델의 출력을 예측
  - 케라스에서는 predict() 함수를 사용

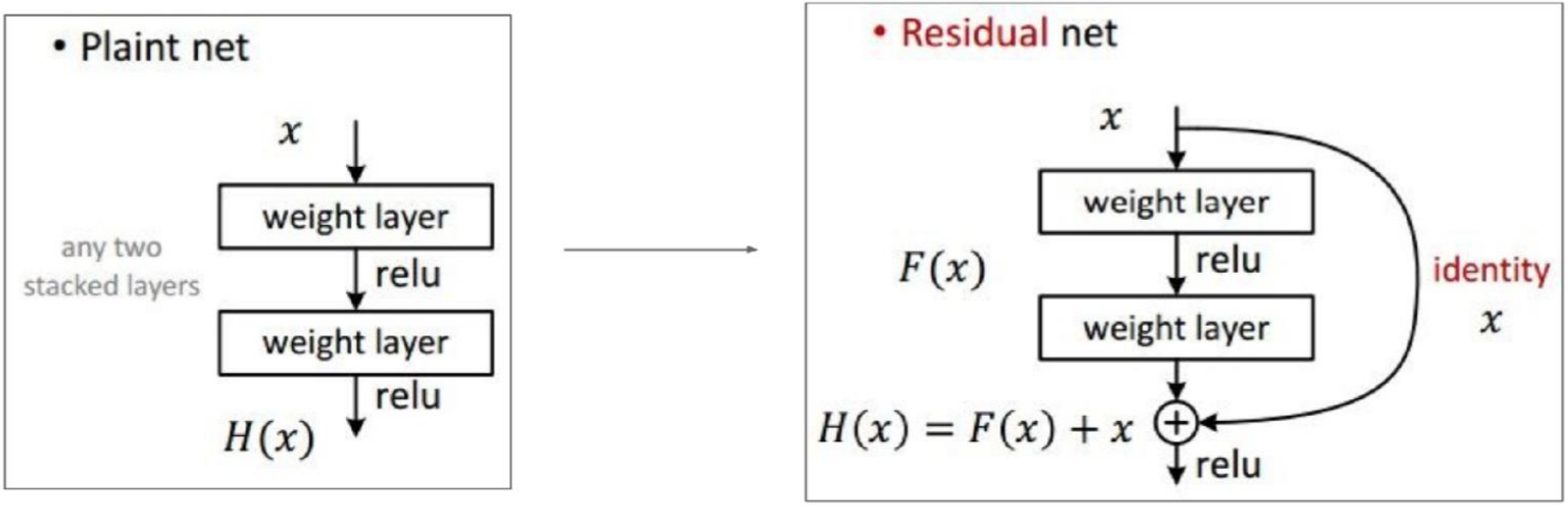
신경망 설계 – keras 요약

- Sequential 모형 클래스 객체 생성
- add () 메서드로 layer 추가
  - 입력단부터 순차적으로 추가
  - 각 layer – 출력 뉴런 갯수를 첫번째 인수
  - 최초의 layer – input_dim 인수로 입력 크기를 설정
  - activation : 활성화함수 설정
- compile () 메서드로 모형 완성
  - Loss : 비용함수 설정
  - optimizer : 최적화 알고리즘 설정
  - metrics : 훈련 단계에서 기록할 성능 기준 설정
- fit () 메서드로 학습
  - nb_epoch : epoch 횟수 설정
  - batch_size : 배치크기 설정
  - Verbose : 학습 중 출력되는 문구를 설정,

  * Jupyter Notebook 사용시 verbose=2로 설정, progress bar 나오지 않도록

Q & A

ResNet


• Plaint net
x
weight layer
relu
weight layer
relu
$H(x)$
any two
stacked layers
• Residual net
x
weight layer
relu
weight layer
$F(x)$
identity
x
$H(x) = F(x) + x$
relu

고양이 강아지 구분

- 고양이와 강아지 이미지를 구분하는 예를 소개하겠다.
- 이러한 분류에 MLP 모델을 사용하면 성능이 나쁘게 나온다.
- 데이터는 아래 Kaggle 사이트에서 다운로드 받을 수 있다.
  https://www.kaggle.com/c/dogs-vs-cats/data
- 캐글에서 원래 제공하는 데이터는 25,000 장이나 여기서는 2천 장의 사진만 사용하겠다.
- 적은 훈련 데이터를 사용하는 경우에 신경망 과대적합이 발생하기 쉽다. 이를 방지하기 위해 데이터 확장(augmentation)을 사용한다
- 전이학습을 소개한다.
  - 전이학습을 사용하면 훈련 시간도 줄이고, 훈련 데이터도 적게 필요하고, 분류 성능도 상당히 개선시킬 수 있다.

CNN을 이용한 MNIST

- MLP를 이용한 MNIST 숫자 인식 프로그램을 소개했었다.
  - 간단한 구조의 MLP를 이용한 MNIST 인식에서도 인식률이 97.8%의 높은 성능을 보였는데 이는 주어진 샘플 이미지가 잘 정리되어서 크기와 위치가 모두 균일했기 때문이었다.

- 여기서는 일반적인 이미지 (즉, 숫자의 위치와 크기가 랜덤한)에 대해서도 인식률을 높일 수 있는 CNN을 소개하겠다.

- CNN의 가장 큰 특징은 입력 신호에 여러 가지 필터(커널)를 적용한다는 것이다.
  - 이미지 처리의 경우 보통 3x3 크기의 작은 필터를 적용한다.
  - 예를 들어 붉은 색을 찾는 필터를 통과시키면 입력 이미지에서 붉은 색이 많은 부분을 찾아내고, 대각선 성분이 있는 곳을 찾는 필터를 통과시킨면 대각선 성분의 크기에 비례하는 출력은 얻는다.