모두를 위한 딥러닝 8장

신재현

병렬 컴퓨팅이란?

- 내부의 멀티코어 cpu다수를 활용하여 연산하는 컴퓨팅 방법
  → 독립적인 작업 프로그램 단위로 병렬처리
  → 하나의 큰 작업을 작은 단위로 분할하여 처리
  → 스레드 단위로 처리
  → 데이터 의존성이 없는 여러 개의 명령을 동시에 수행

손글씨 숫자를 인식하는 심층 CNN

- VGG 신경망을 참고
- -특징
  → 3*3의 작은 필터를 사용한 합성곱 계층
  → 활성화 함수는 RELU
  → Adam을 사용해 최적화
  → 가중치 초기값은 HE초기값

여러 계층을 사용하는 이유

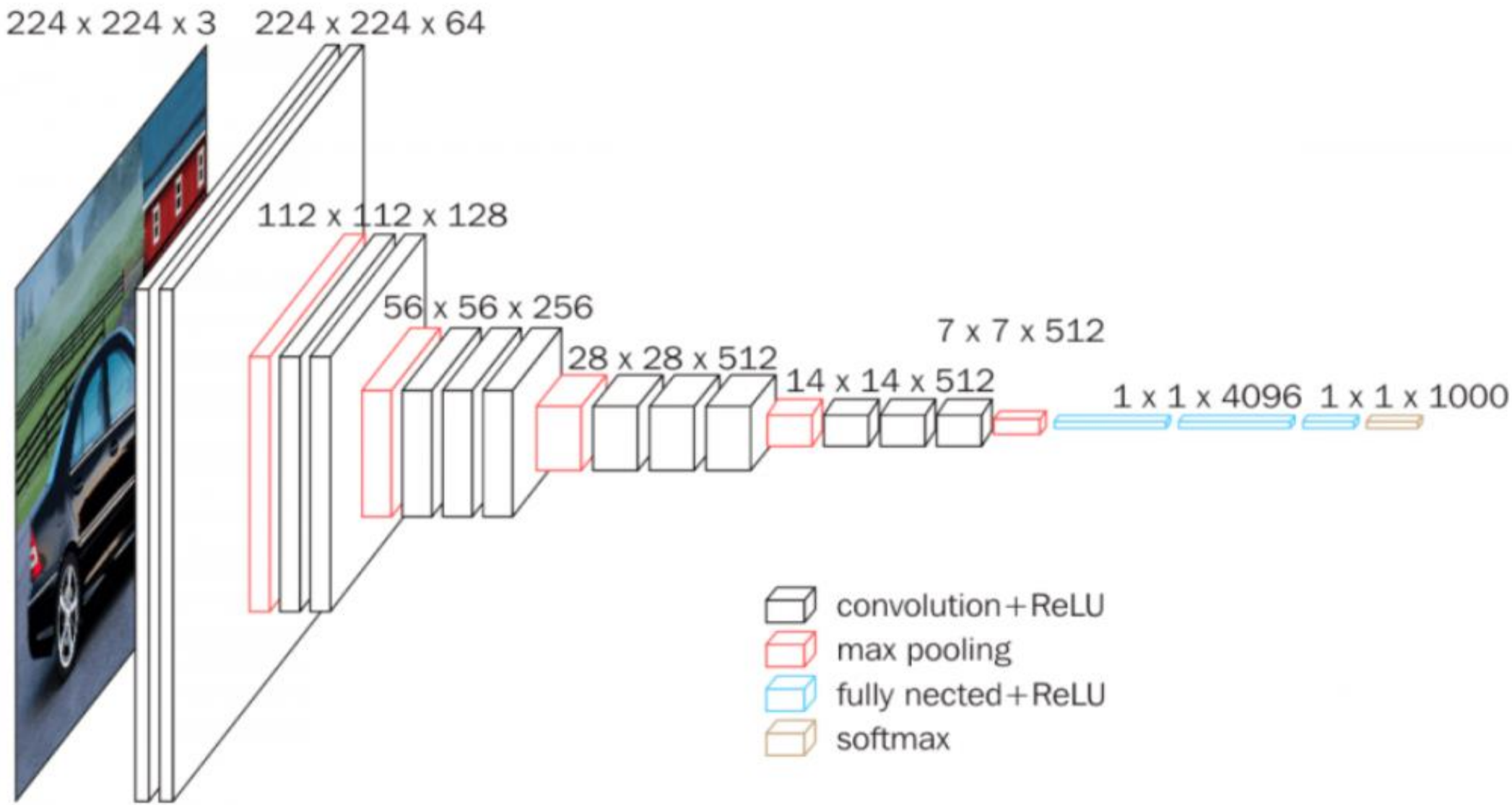
- 데이터 확장을 입력 이미지를 알
- 학습의 효율성 향상 → 층을 깊이 함으로써 각 층이 학습해야 할 문제를 풀기 쉬운 단순한 문제로 분해하여 효율적인 학습이 가능하다

VGG

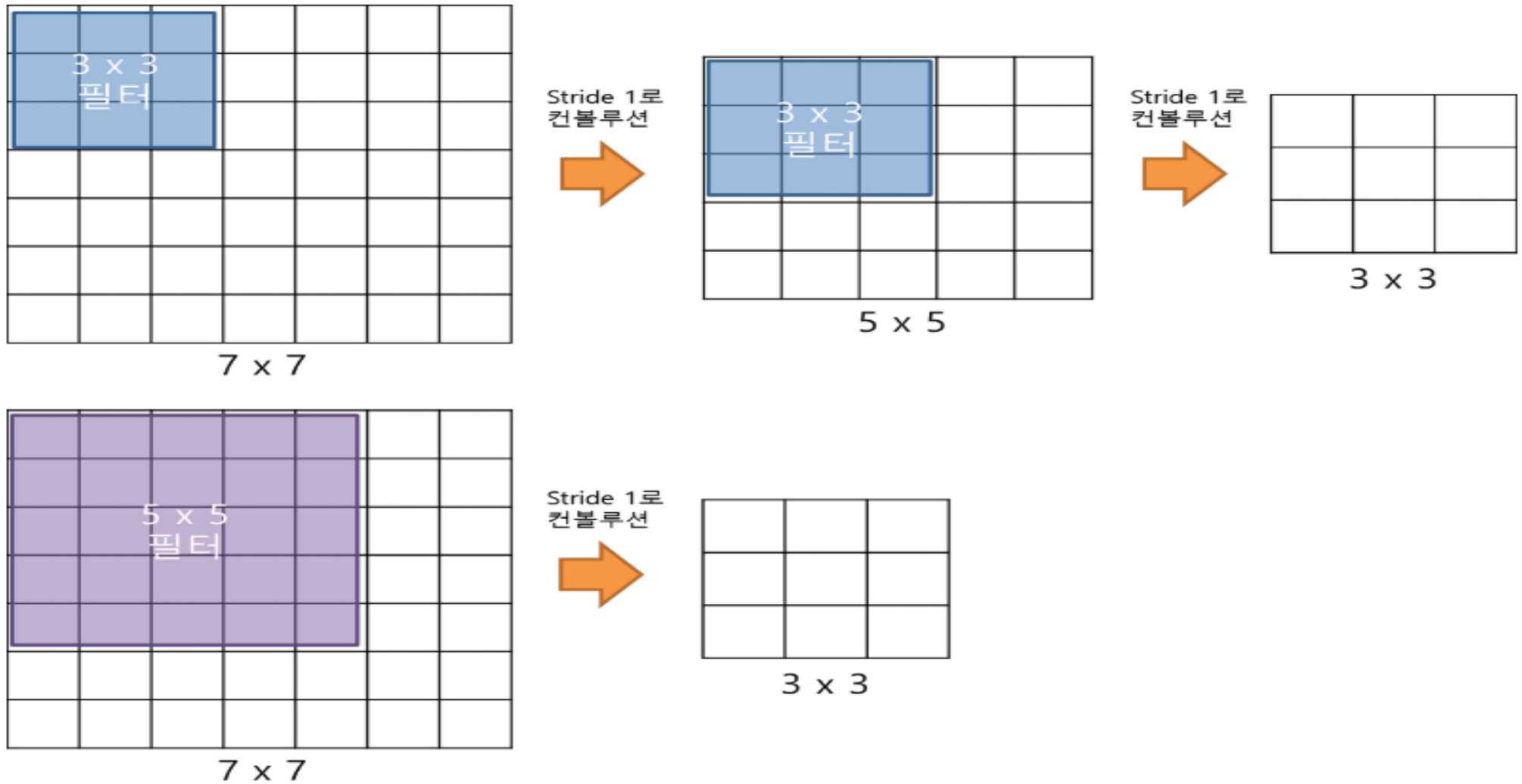
- 합성곱 계층과 풀링 계층으로 구성되는 CNN
- 풀링 계층을 두어 크기를 절반으로 줄이는 처리
- 구성이 간단하여 응용에 용이하여 기술자에 인기
- 합성곱 계층을 2-4회 연속으로 풀링 계층을 두어 크기를 절반으로 줄이는 처리를 반복한다. 완전 연결 계층을 통과시켜 결과를 출력한다.

VGG

In [0]:
```
def vgg(input_shape, n_classes):

  input = Input(input_shape)

  x = Conv2D(64, 3, padding='same', activation='relu')(input)
  x = Conv2D(64, 3, padding='same', activation='relu')(x)
  x = MaxPool2D(2, strides=2, padding='same')(x)

  x = Conv2D(128, 3, padding='same', activation='relu')(x)
  x = Conv2D(128, 3, padding='same', activation='relu')(x)
  x = MaxPool2D(2, strides=2, padding='same')(x)

  x = Conv2D(256, 3, padding='same', activation='relu')(x)
  x = Conv2D(256, 3, padding='same', activation='relu')(x)
  x = Conv2D(256, 3, padding='same', activation='relu')(x)
  x = MaxPool2D(2, strides=2, padding='same')(x)

  x = Conv2D(512, 3, padding='same', activation='relu')(x)
  x = Conv2D(512, 3, padding='same', activation='relu')(x)
  x = Conv2D(512, 3, padding='same', activation='relu')(x)
  x = MaxPool2D(2, strides=2, padding='same')(x)

  x = Conv2D(512, 3, padding='same', activation='relu')(x)
  x = Conv2D(512, 3, padding='same', activation='relu')(x)
  x = Conv2D(512, 3, padding='same', activation='relu')(x)
  x = MaxPool2D(2, strides=2, padding='same')(x)

  x = Flatten()(x)
  x = Dense(4096, activation='relu')(x)
  x = Dense(4096, activation='relu')(x)

  output = Dense(n_classes, activation='softmax')(x)

  model = Model(input, output)
  return model
```


224 x 224 x 3
224 x 224 x 64
112 x 112 x 128
56 x 56 x 256
28 x 28 x 512
14 x 14 x 512
7 x 7 x 512
1 x 1 x 4096
1 x 1 x 1000
convolution+ReLU
max pooling
fully nected+ReLU
softmax

VGGNet의 구조를 깊이 들여다보기에 앞서 먼저 집고 넘어가야할 것이 있다. 그것은 바로 3 x 3 필터로 두 차례 컨볼루션을 하는 것과 5 x 5 필터로 한 번 컨볼루션을 하는 것이 결과적으로 동일한 사이즈의 특성맵을 산출한다는 것이다(아래 그림 참고). 3 x 3 필터로 세 차례 컨볼루션 하는 것은 7 x 7 필터로 한 번 컨볼루션 하는 것과 대응된다.

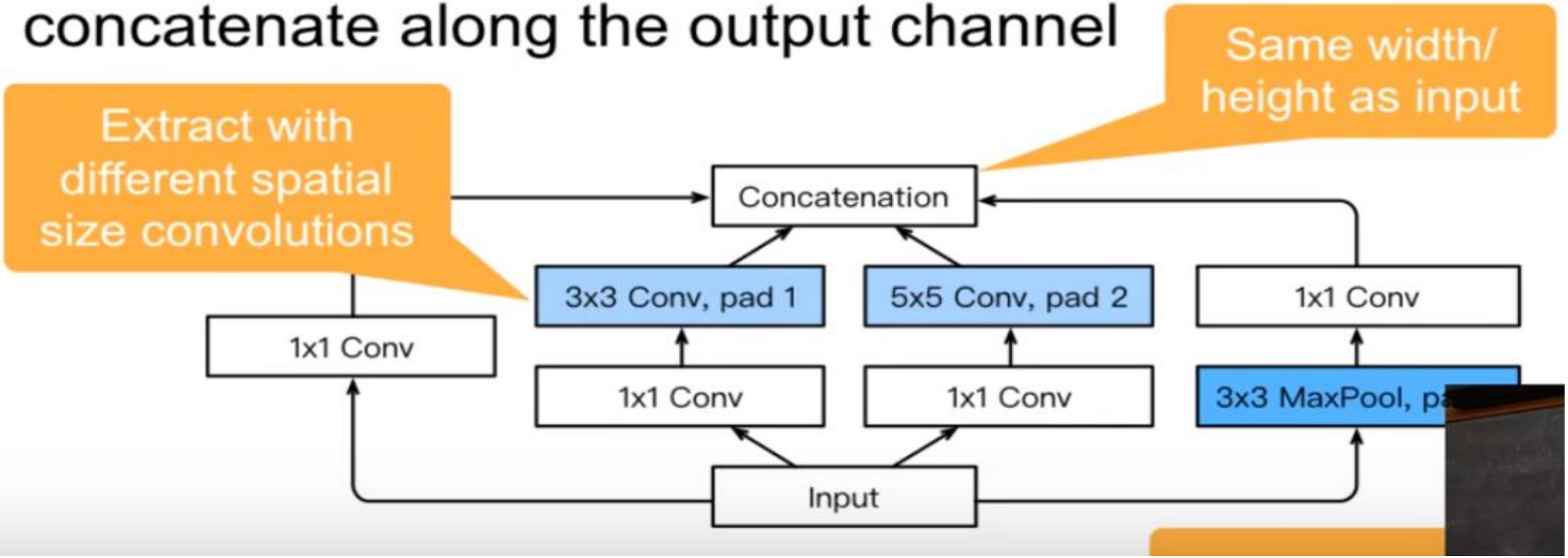


GoogleNet

- 그림의 사각형이 합성곱 계층과 풀링 계층등의 계층
- 세로 방향 깊이 뿐만 아니라 가로 방향도 깊다
- 가로 방향의 폭을 인셉션 구조라 한다
- 1*1의 합성곱 연산은 매개변수 제거와 고속 처리에 기여한다
- Inception structure

Inception Blocks

4 paths extract information from different aspects, the
concatenate along the output channel



googleNet이 Inception Block을

```
def googlenet(input_shape, n_classes):

  def inception_block(x, f):
    t1 = Conv2D(f[0], 1, activation='relu')(x)

    t2 = Conv2D(f[1], 1, activation='relu')(x)
    t2 = Conv2D(f[2], 3, padding='same', activation='relu')(t2)

    t3 = Conv2D(f[3], 1, activation='relu')(x)
    t3 = Conv2D(f[4], 5, padding='same', activation='relu')(t3)

    t4 = MaxPool2D(3, 1, padding='same')(x)
    t4 = Conv2D(f[5], 1, activation='relu')(t4)

    output = Concatenate()([t1, t2, t3, t4])
    return output


  input = Input(input_shape)

  x = Conv2D(64, 7, strides=2, padding='same', activation='relu')(input)
  x = MaxPool2D(3, strides=2, padding='same')(x)

  x = Conv2D(64, 1, activation='relu')(x)
  x = Conv2D(192, 3, padding='same', activation='relu')(x)
  x = MaxPool2D(3, strides=2)(x)

  x = inception_block(x, [64, 96, 128, 16, 32, 32])
  x = inception_block(x, [128, 128, 192, 32, 96, 64])
  x = MaxPool2D(3, strides=2, padding='same')(x)

  x = inception_block(x, [192, 96, 208, 16, 48, 64])
  x = inception_block(x, [160, 112, 224, 24, 64, 64])
  x = inception_block(x, [128, 128, 256, 24, 64, 64])
  x = inception_block(x, [112, 144, 288, 32, 64, 64])
  x = inception_block(x, [256, 160, 320, 32, 128, 128])
  x = MaxPool2D(3, strides=2, padding='same')(x)

  x = inception_block(x, [256, 160, 320, 32, 128, 128])
  x = inception_block(x, [384, 192, 384, 48, 128, 128])

  x = AvgPool2D(7, strides=1)(x)
  x = Dropout(0.4)(x)
```

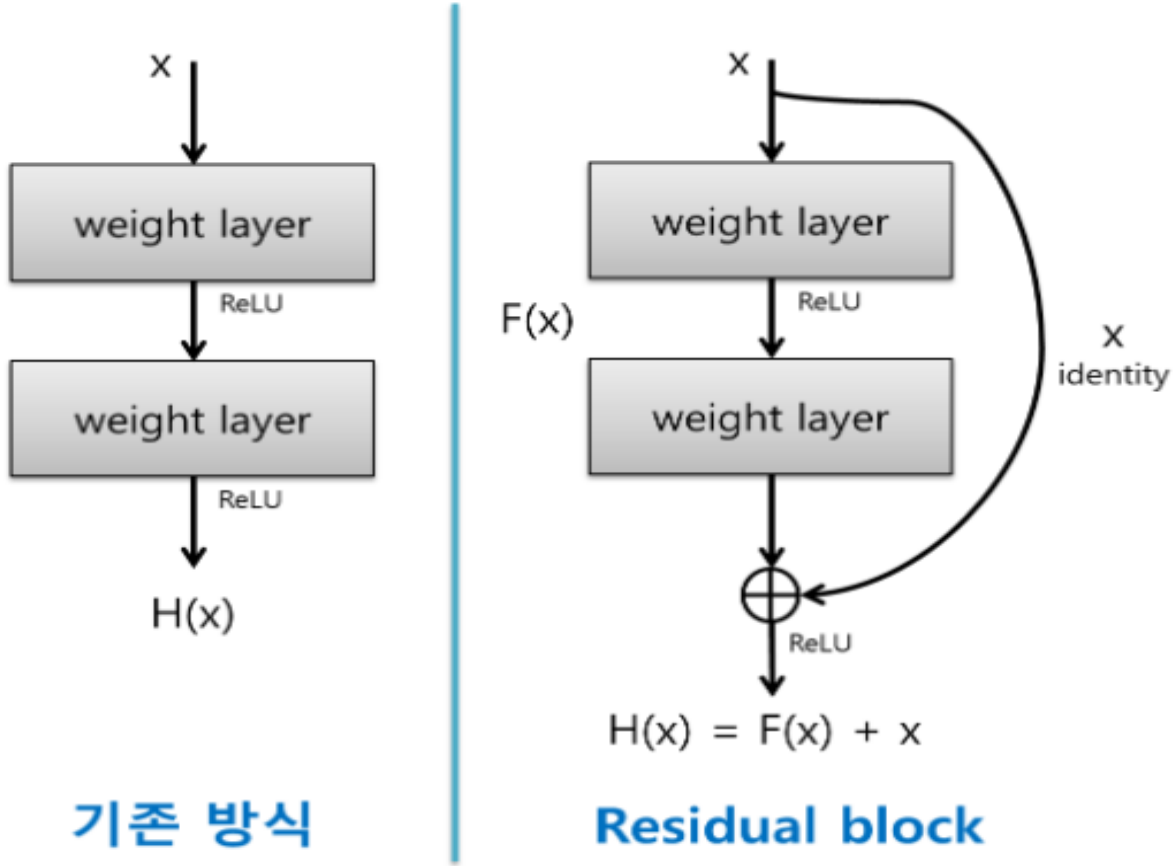
ResNet

- 층을 깊게 하는 것이 성능향상에 중요하다는 건 알고 있었는데, 층을 지나치게 늘려 오히려 성능이 떨어지는 경우가 발생한다.
- ResNet의 구성요소: weight layer는 합성곱 계층을 말한다.
- 기존의 신경망은 입력값 x를 타겟값 y로 매핑하는 함수 H(x)를 얻는 것이 목적이었다. 그러나 **ResNet은 F(x) + x를 최소화하는 것을 목적으로 한다.** 최소로 해주는 것이므로 ResNet이란 이름이 붙게 된다.

```
def resnet(input_shape, n_classes):

  def conv_bn_rl(x, f, k=1, s=1, p='same'):
    x = Conv2D(f, k, strides=s, padding=p)(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)
    return x


  def identity_block(tensor, f):
    x = conv_bn_rl(tensor, f)
    x = conv_bn_rl(x, f, 3)
    x = Conv2D(4*f, 1)(x)
    x = BatchNormalization()(x)

    x = Add()([x, tensor])
    output = ReLU()(x)
    return output


  def conv_block(tensor, f, s):
    x = conv_bn_rl(tensor, f)
    x = conv_bn_rl(x, f, 3, s)
    x = Conv2D(4*f, 1)(x)
    x = BatchNormalization()(x)

    shortcut = Conv2D(4*f, 1, strides=s)(tensor)
    shortcut = BatchNormalization()(shortcut)

    x = Add()([x, shortcut])
    output = ReLU()(x)
    return output


  def resnet_block(x, f, r, s=2):
    x = conv_block(x, f, s)
    for _ in range(r-1):
      x = identity_block(x, f)
    return x


  input = Input(input_shape)

  x = conv_bn_rl(input, 64, 7, 2)
  x = MaxPool2D(3, strides=2, padding='same')(x)

  x = resnet_block(x, 64, 3, 1)
  x = resnet_block(x, 128, 4)
  x = resnet_block(x, 256, 6)
  x = resnet_block(x, 512, 3)

  x = GlobalAvgPool2D()(x)

  output = Dense(n_classes, activation='softmax')(x)

  model = Model(input, output)
  return model
```

Residual Block

그것이 바로 ResNet의 핵심인 Residual Block의 출현을 가능케 했다. 아래 그림에서 오른쪽이 Residual Block을 나타낸다. 기존의 망과 차이가 있다면 입력값을 출력값에 더해줄 수 있도록 **지름길(shortcut)**을 하나 만들어준 것 뿐이다.



전페이지 부가설명

- x는 현시점에서 변할 수 없는 값이므로 F(x)를 0에 가깝게 만드는 것이 목적이 된다. F(x)가 0이 되면 출력과 입력이 모두 x로 같아지게 된다.
- F(x) = H(x) - x이므로 F(x)를 최소로 해준다는 것은 H(x) - x를 최소로 해주는 것과 동일한 의미를 지닌다. 여기서 H(x) - x를 **잔차(residual)**라고 한다. 즉, 잔차를 최소화해준다는 의미에서 Resnet이라 이름지어졌다.

입력이미지 - 후보영역 추출 - CNN 특징계산 - 영역 분류

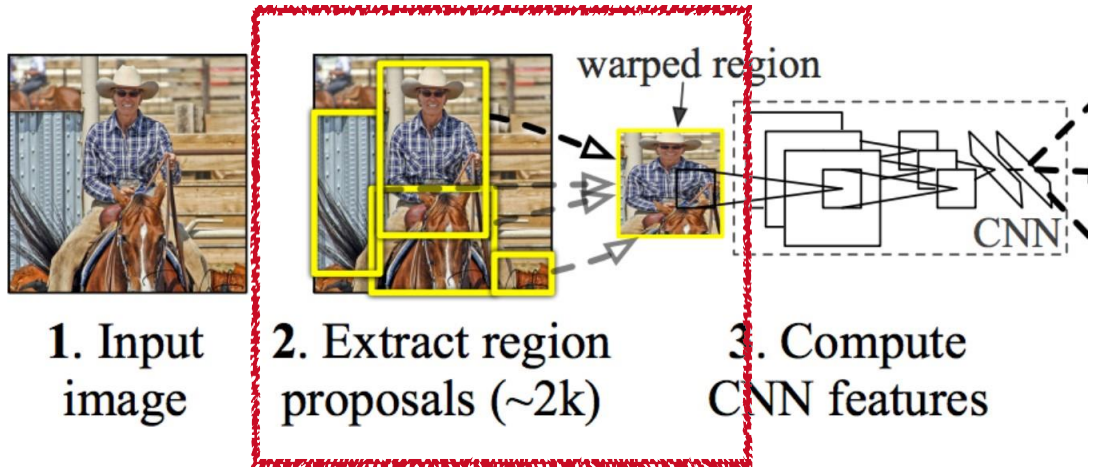
- Fast R-CNN

-모든 일을 하나의 CNN에서 처리하기 때문에 엄청 빠름

- R-CNN의 시간문제를 개선한 방법
  - ✓ROI Pooling 이라는 계층 도입
  - ✓CNN에서 얻어진 특징지도 (Feature map) 의 일부영역으로 부터 정규화된 특징을 추출
  - ✓Fast R-CNN은 각 RP(Region Proposal) 에 CNN을 반복적으로 적용하는 대신, 입력영상에 CNN을 한번만 적용하고 RIO Pooling 으로 객체를 판별하기 위한 특징을 추출 (R-CNN 에 비해서 시간단축)

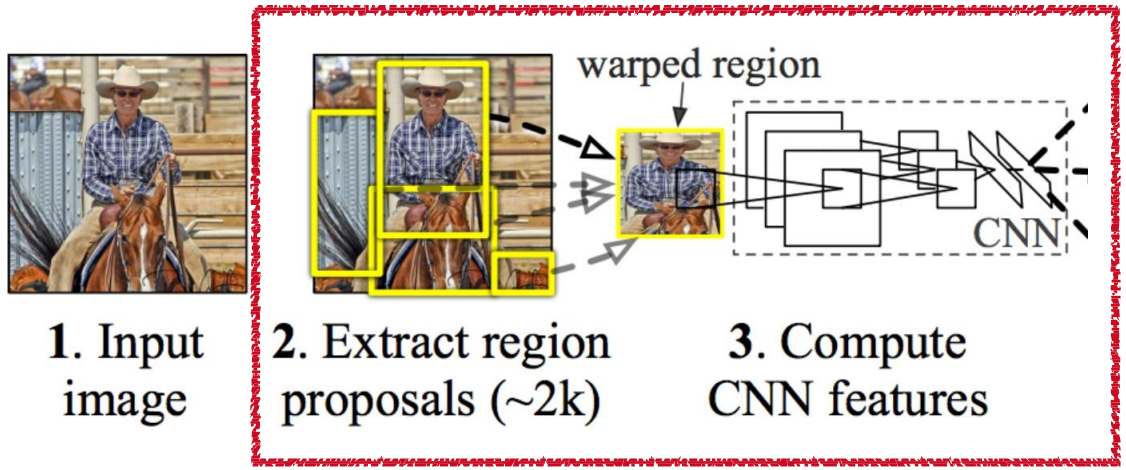
Fast R-CNN 의 문제점

- 성능 병목 = 바운딩박스를 만드는 Region Proposal 단계



Fast R-CNN 의 문제점 해결

- Region Proposal 단계를 CNN에 포함시켜 병목을 해결. = **Faster R-CNN 알고리즘**

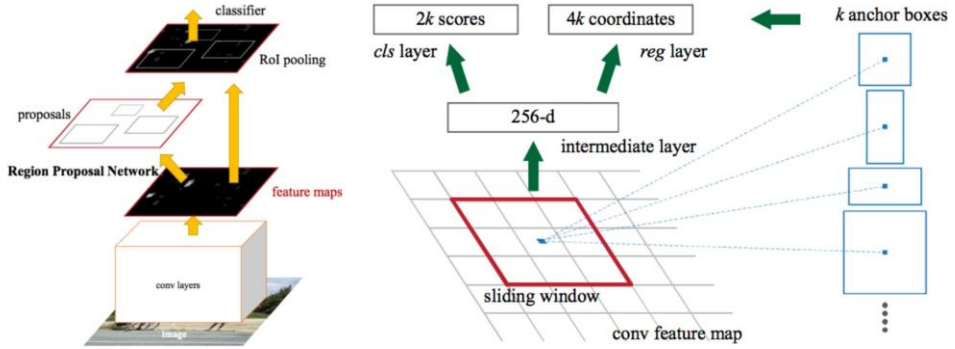


[통합]

Faster R-CNN 을 활용한 사물검출(object detection)

- Faster Regions With Convolutional Neural Network

Faster R-CNN

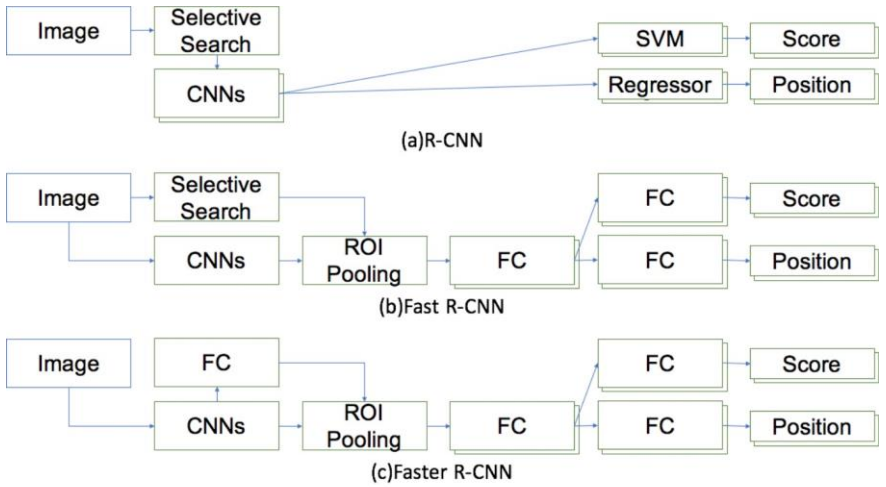


- PyTorch : https://github.com/longcw/faster_rcnn_pytorch

- **이미지영역 탐색 (Region Proposal) 또는 바운딩박스 (Bounding Box) 성능 문제 해결 - Fast R-CNN 문제해결**
  - 방법 : Region Proposal 단계를 CNN 에 포함시킴
- CNN 을 통과한 특성맵에서 슬라이딩 윈도우를 이용해 각 지점 (Anchor) 마다 가능한 바운딩박스의 좌표와 바운딩박스의 점수 계산
- 너무 홀쭉하거나 넓은 물체는 많지 않으므로 2:1, 1:1, 1:2 등 몇가지 타입으로도 가능.
- Faster R-CNN 은 작년에 마이크로 소프트에서 내놓은 대표적인 컴퓨터 비젼 연구 결과 중 하나.

Processing Flow

- R-CNN
- Fast R-CNN
- Faster R-CNN



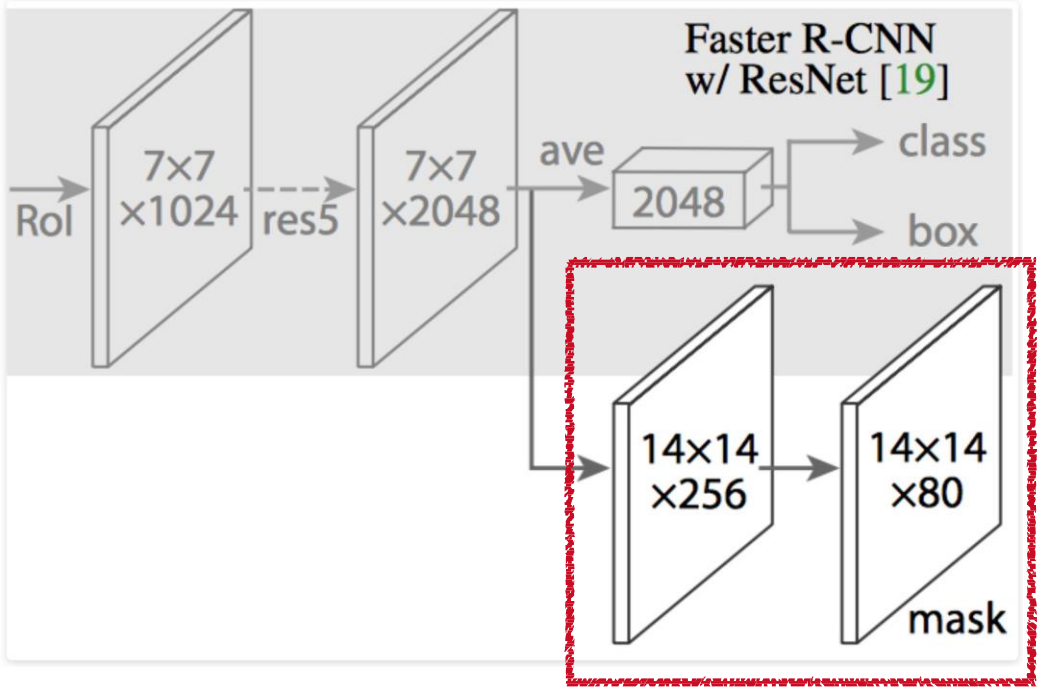
[그림 2] R-CNN, Fast R-CNN, Faster R-CNN 방법들의 차이

[참고] R-CNN 에서 Faster R-CNN 까지의 설명자료

- [영상강의] https://youtu.be/kcPAGIgBGRs
- [논문자료] https://arxiv.org/abs/1506.01497

Mask R-CNN 을 활용한 사물검출

- 페이스북 AI팀이 내놓은 알고리즘



➡ 분할된 이미지를 마스킹

➡ Faster R-CNN 에 각 픽셀이 오프젝트에 해당하는 것인지 아닌지를 마스킹하는 네트워크 (CNN) 추가한 것

➡ 마스크 = 바이너리 마스크

➡ 정확한 픽셀위치를 추출하기 위한 방법

- CNN 통과하면서 RoIPool 영역의 위치에 생기는 소숫점 오차를 '2D 선형보간법'을 통해 감소. 이를 RoI Align 이라함

RoI : Region of Interest

Mask R-CNN 을 활용한 사물검출

- 페이스북 AI팀이 내놓은 알고리즘

➡ 분할된 이미지를 마스킹

➡ Faster R-CNN 에 각 픽셀이 오프젝트에 해당하는 것인지 아닌지를 마스킹하는 네트워크 (CNN) 추가한 것

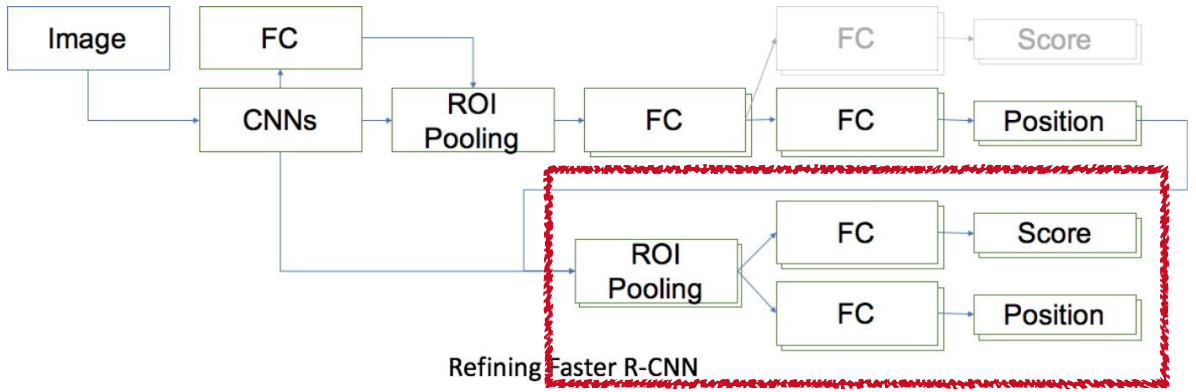
➡ 마스크 = 바이너리 마스크

➡ 정확한 픽셀위치를 추출하기 위한 방법

- CNN 통과하면서 RoIPool 영역의 위치에 생기는 소숫점 오차를 '2D 선형보간법'을 통해 감소. 이를 RoI (Region of Interest) Align 이라함

카카오의 RF-R-CNN

- Refining Faster R-CNN
- 로드뷰에서 촬영된 자동차의 번호판과 사람들의 얼굴을 블러 처리하기 위해 운영



반복구간

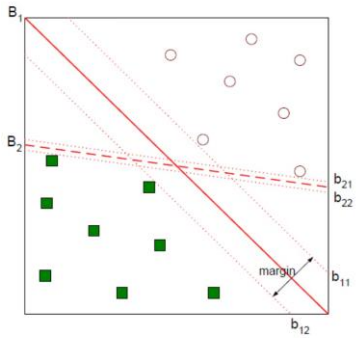
[그림 3] Faster R-CNN에서 얻어진 결과를 그대로 사용하는 대신, 추가적인 보정을 거쳐서 성능을 향상시킬 수 있는 RF-RCNN

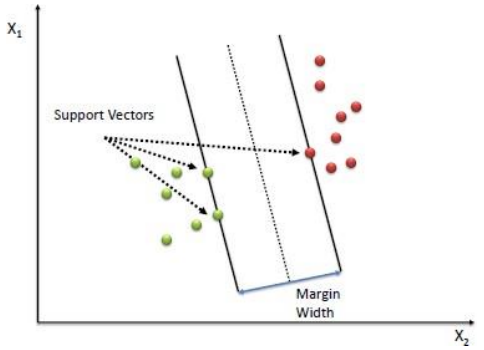
[참고] SVM (서포트 벡터머신)

- 후보영역 추출방식은 '스코어링 방식' 이며, 컴퓨터 비젼 분야에서 발전한 다양한 기법 사용
  - SVN (Support Vector Machine)

margin

두 범주를 나누는 분류 문제를 푼다고 가정해 보겠습니다. 아래 그림에서 직선 B_1 과 B_2 모두 두 클래스를 무난하게 분류하고 있음을 확인할 수 있습니다.





좀 더 나은 분류경계면을 꼽으라면 B_1 일 겁니다. 두 범주를 여유있게 가르고 있거든요. 위 그림에서 b_{12} 을 minus-plane, b_{11} 을 plus-plane, 이 둘 사이의 거리를 **마진(margin)**이라고 합니다. SVM은 이 마진을 최대화하는 분류 경계면을 찾는 기법입니다. 이를 도식적으로 나타내면 아래와 같습니다.

[참고] Selective Search

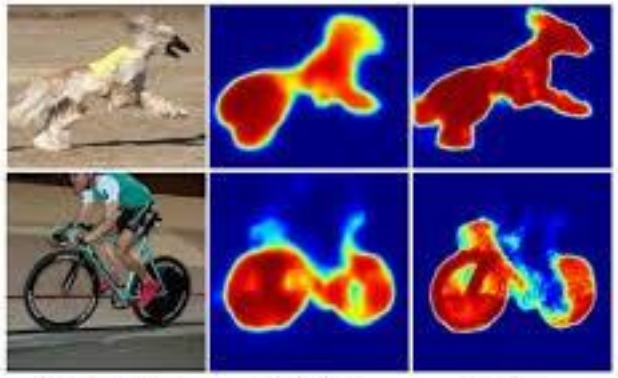
- 참고 사이트
  - http://laonple.blog.me/220918802749

- 후보영역 추출방식은 '스코어링방식' 이며, 컴퓨터 비젼 분야에서 발전한 다양한 기법 사용
  - 많은 연산을 필요로하는 딥러닝의 단점을 극복하기 위해서 나옴
  - 전통적인 객체 검출용 기법을 딥러닝에 적용하기 어려움 (피라미드표현 / 슬라이딩 윈도우즈)
  - 효율성이 높은 기존의 방법을 차용하여 객체가 있을 법한 후보영역 (RP : Region Proposal)탐색, 그 영역에 딥러닝 기법을 적용하는 기술

픽셀단위의 분할 기술

- 기존에 전체 이미지를 대상으로 분류하는 것이 아닌 픽셀단위로 분류 (추론)
- 픽셀 수 만큼 Forward 처리. 긴 시간이 필요

  합성곱 연산에서 불필요한 영역을 다시 계산하는 문제해결을 위해 FCN 기법 고안

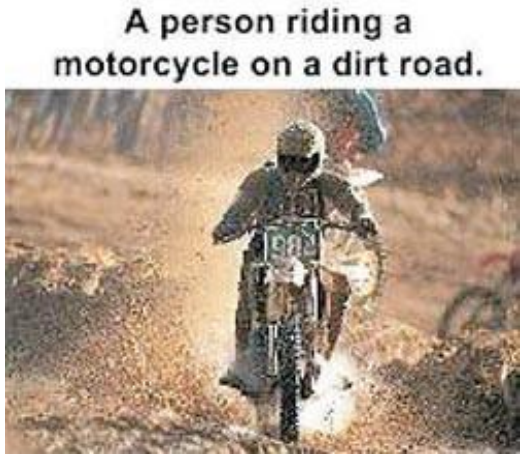


(a) Input image (b) FCN-8s (c) Ours

Figure 5. Comparison of class conditional probability maps from FCN and our network (top: dog, bottom: bicycle).

이미지 캡션

- 사진의 내용을 읽어 캡션을 자동으로 생성하는 기술
- NIC (Neural Image Caption) 모델이 대표적
  - CNN 과 자연어를 다루는 순환신경망 RNN 으로 구성
  - CNN으로 사진에서 특징을 추출하고 그 결과를 RNN으로 전달
  - RNN은 CNN으로 추출한 특징을 초기값으로 텍스트를 생성



[참고]

- RNN (Recurrent Neural Network)
  - 순환 신경망 네트워크 구조
  - 순환적 구조로 인해 이전에 생성한 정보에 영향을 받는 점이 특징
  - 예) '나' 라는 단어를 생성한 뒤 '잤다' 라는 단어를 생성하면, '나'라는 단어의 영향을 받아 '는' 이라는 조사가 자동으로 생성. 자
  - 연어와 시계열 데이터 등 연속성이 있는 다룰때 RNN 은 과정정보를 기억하면서 동작
- 멀티모달 프로세싱 (Multi-Modal Processing)
  - 사진이나 자연어와 같은 여러종류의 정보를 조합하고 처리하는 방법
  - 멀티모달센서를 이용한 인간자세 및 행동인식에 활용 중
    (3D가속도, 3D자이로, 3D지자기, 고도계, 시간등 11개의 다양한 센서 정보활용)

앞으로의 딥러닝 활용기술

1. 이미지 스타일 변환
2. 이미지 생성
3. 자율주행
4. 강화학습(Deep Q-Network)
5. 참고

이미지 스타일의 변환

- ‘고흐’ 의 작품을 교육시키고, 특정 이미지 입력하여 학습된 ‘고흐’ 스타일로 변환
- https://youtu.be/Uxax5EKg0zA

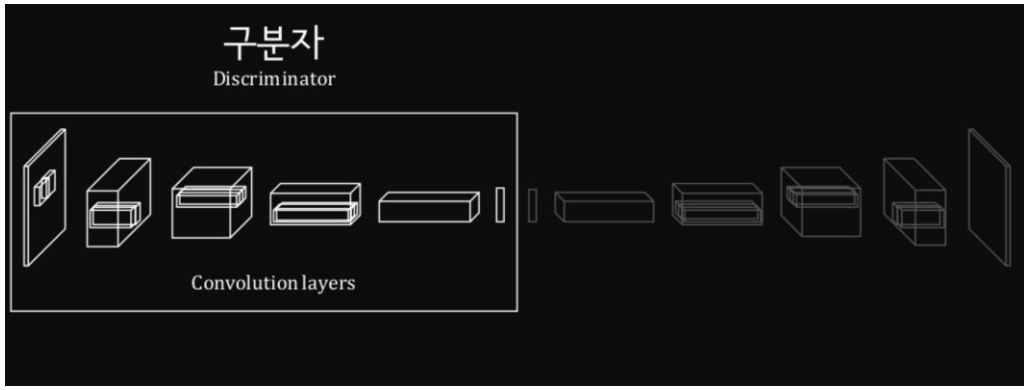
이미지 생성

- 먼저 대량의 이미지 학습 후, 완전히 새로운 그림을 그린다.
- DCGAN 활용
  - Deep Convolutional Generative Adversarial Network
    - Deep Learning
    - ConvNet
    - Generative (Model) : 생성모델
    - Adversarial (Learning) : 적대적 학습
    - (Neural) Networks : 인공신경망

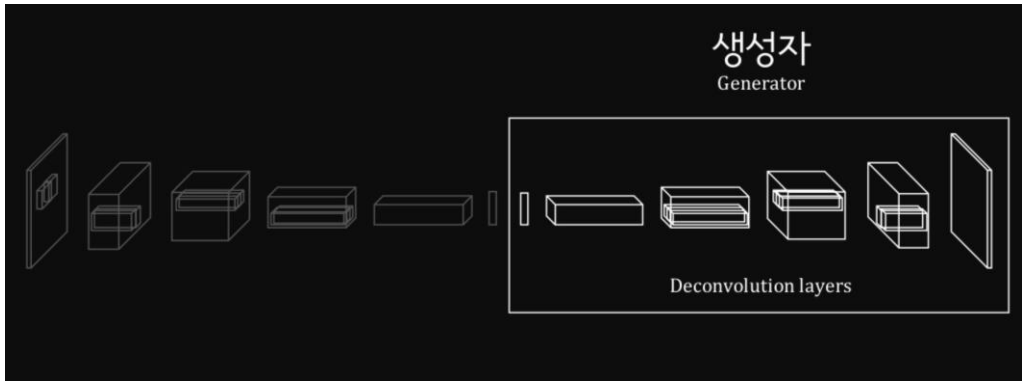
이미지 생성

- 먼저 대량의 이미지 학습 후, 완전히 새로운 그림을 그린다.
- DCGAN 활용
  - 이미지를 생성하는 과정을 모델링한 것
  - 딥러닝 사용
    - ✓ 생성자와 식별자로 불리우는 2개의 신경망을 이용
      - ➡생성자 : 진짜와 똑같은 이미지를 생성
      - ➡구분자 (식별자) : 생성자가 그린 이미지가 진짜인지 판정
      - ➡이 둘을 겨루도록 학습시켜 생성자는 더 정교한 이미지를 생성하고 식별자는 더 정확한 감정사로 발전

DCGAN 알고리즘 구조



진짜 이지미를 판별 (식별/구분)

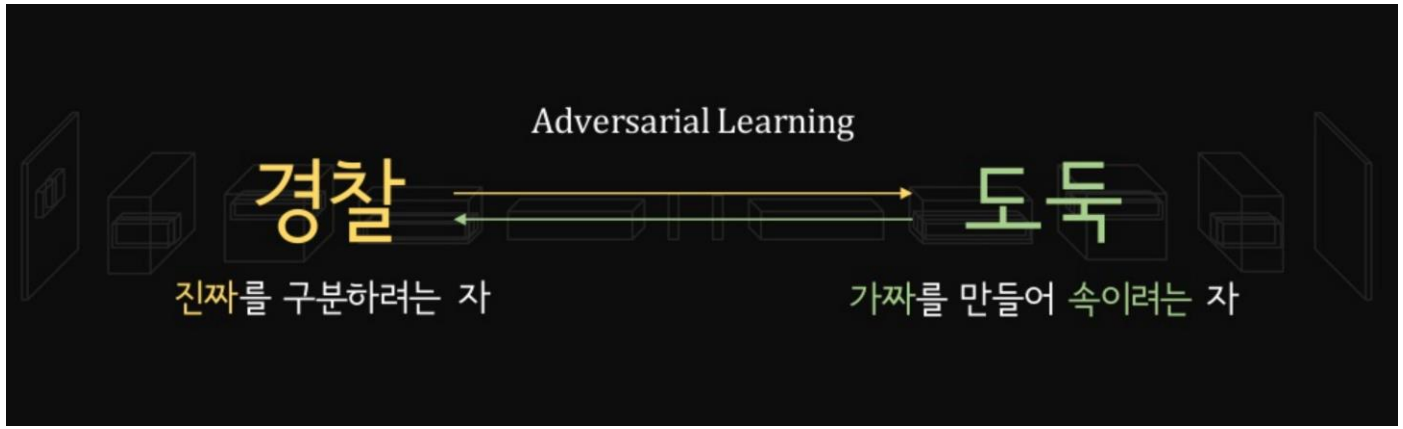


최대한 진짜같은 이미지 생성

8 딥러닝

딥러닝 기술의 미래 - DCGAN

적대적 학습 - 번갈아가며 학습



생성자 (도둑)은 가짜 미술작품을 만들어 돈을 벌고,

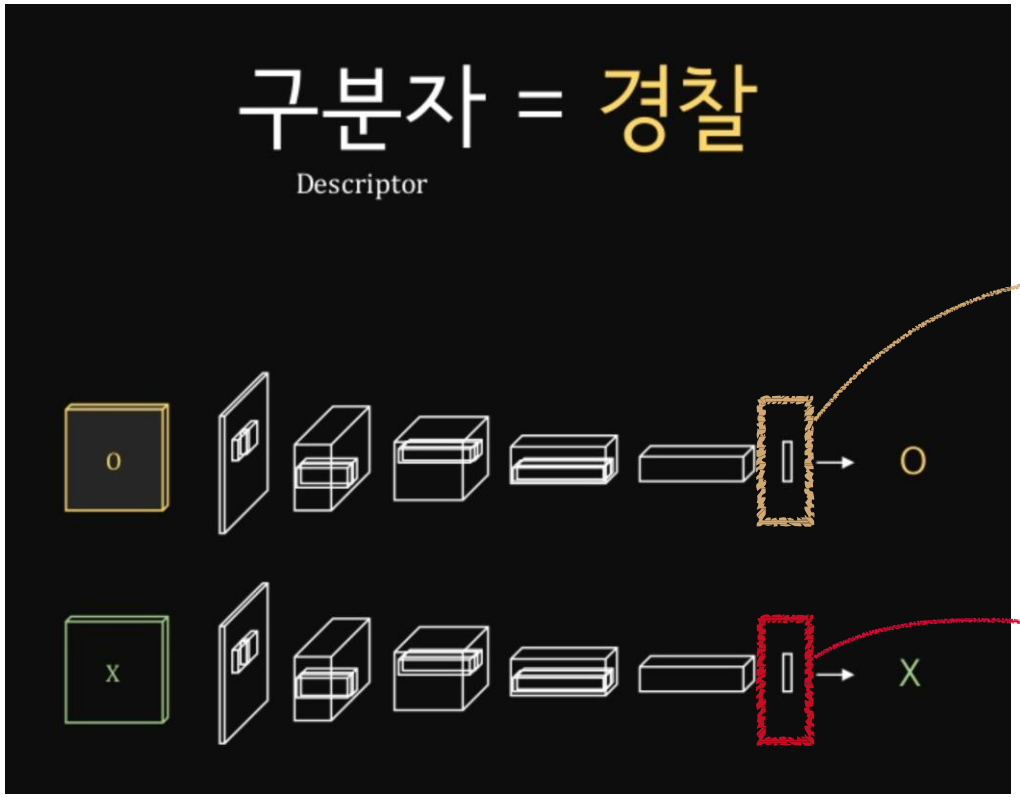
구분자 (경찰)은 가짜 미술작품을 적발할때마다 학습을 함.

생성자 (도둑) 의 실력이 정교해짐에 따라 구분자 (경찰)의 학습능력도 좋아짐

8장 딥러

딥러닝 기술의 미래 - DCGAN

적대적 학습

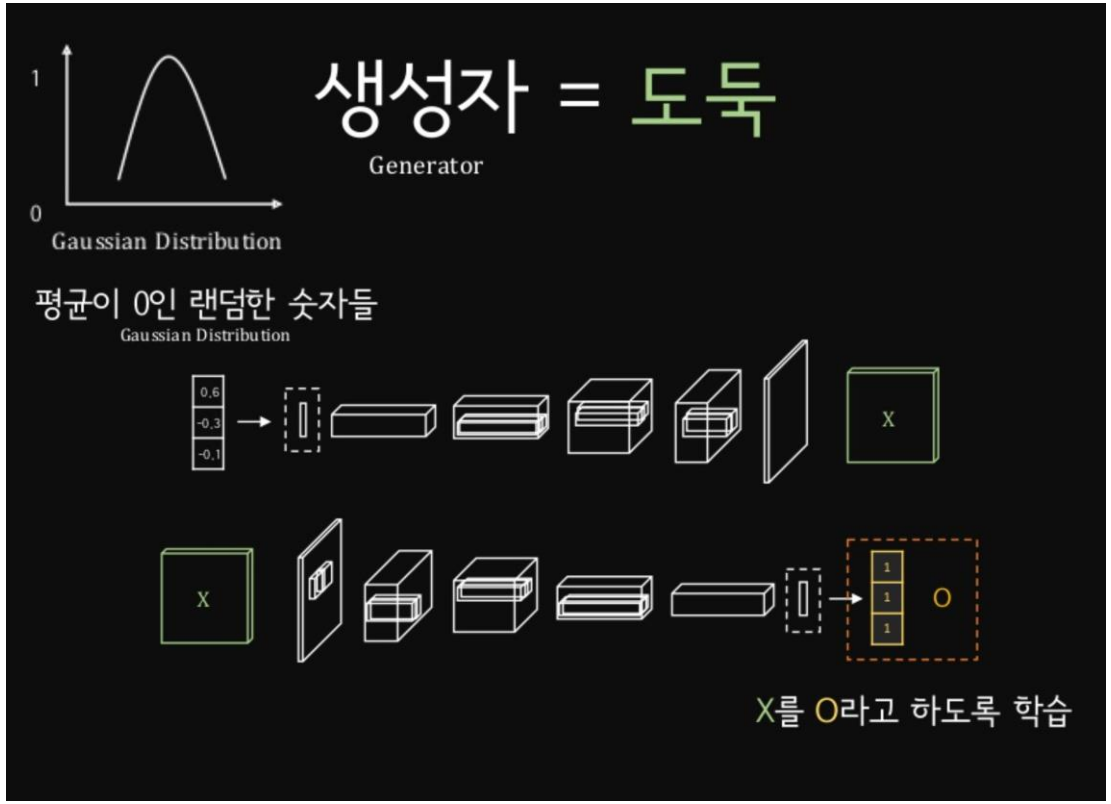


1
1
1

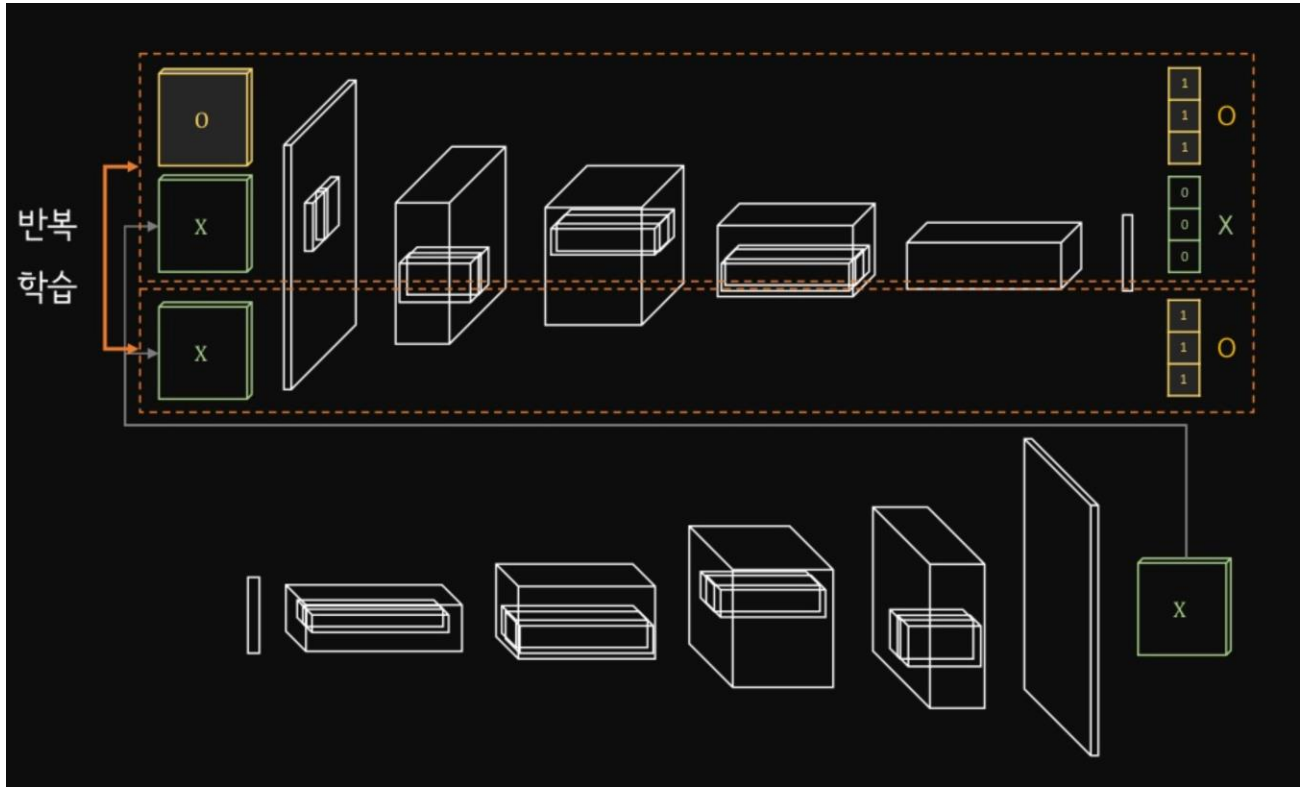
0
0
0

‘1’ 과 ‘0’ 으로 이루어진 벡터를 예측

적대적 학습


1
0
Gaussian Distribution
생성자 = 도둑
Generator
평균이 0인 랜덤한 숫자들
Gaussian Distribution
0.6
-0.3
-0.1
X
X
1
1
1
O
X를 O라고 하도록 학습

적대적 학습



8장 딥러닝

딥러닝 기술의 미래 - DCGAN

[출처] DCGAN 의 내용은 아래 링크에서 추출하여 요약한 것

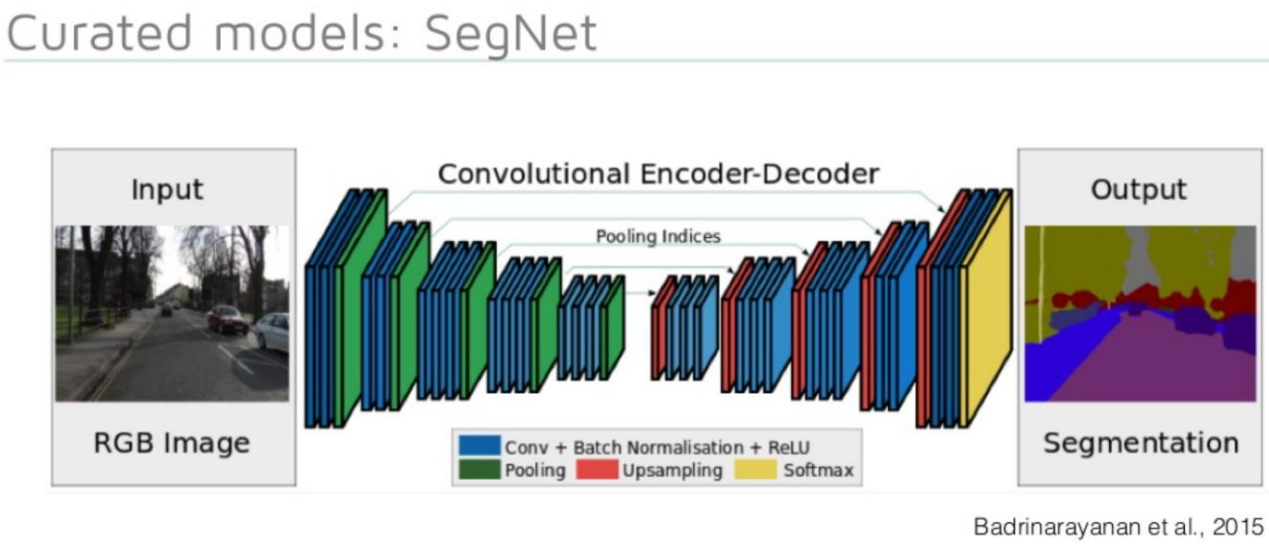
- 아래 링크의 발표자료에서 요약한 것입니다.

  아래 링크에 '텐서플로우' 구현 부분도 잘 나와있습니다.

https://www.slideshare.net/carpedm20/pycon-korea-2016

자율주행

- 주변상황을 정확하게 식별하게 도와주는 CNN 기반의 SegNet 활용
- 입력 이미지를 픽셀 수준에서 분할
- 참고 사이트 : http://mi.eng.cam.ac.uk/projects/segnet/tutorial.html



강화학습 (Deep Q-Network)

- 컴퓨터가 시행착오 과정을 거치면서 학습하도록 하려는 분야
- 쉬운 버전–> 시행착오를 통해 보상이 좋았던 행동은 더 하고 보상이 적었던 행동은 덜 하며 발전하는 과정
- 정확한 버전–> 순차적 의사 결정 문제에서 누적 보상을 최적화 하기 위해 시행 착오를 통해 행동을 교정하며 학습하는 과정

- 에이전트가 환경에 맞게 행동을 선택,하고 그 행동에 의해서 환경이 변한다는 것이 기본틀 보상은 정해진 것이 아니라 예상보상

✓ 벽돌깨기
✓ 자동차 레이싱
✓ 갤러그