# Computer Vision
# 2020년 봄 학기

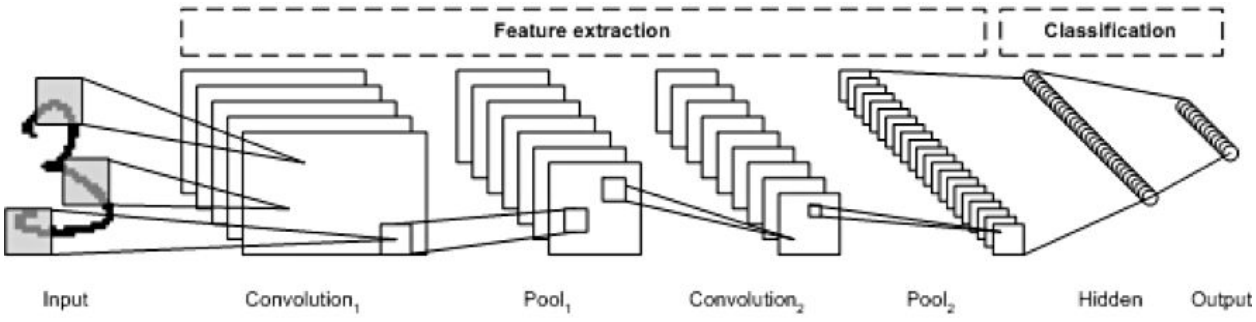
2020년 2월 29일
https://www.github.com/KU-BIG/KUBIG_2020_Spring

## 1. Convolutional Neural Network (CNN)

### 1.1 Fully Connected Layer

Fully Connected Layer의 input data는 1차원 배열 형태여야 한다. 따라서 한 장의 컬러 사진이 input data로 들어왔을 경우 해당 3차원 데이터를 1차원으로 평면화(flatten)해야한다. 그 과정에서 3차원, 4차원 데이터의 공간 정보는 손실된다. 그 결과 이미지의 공간 정보 유실로 인해 data의 정보가 부족해지고, 특징 추출이 어려워지며 이로 인해 낮은 정확도의 비효율적인 학습이 진행된다. CNN은 이미지의 공간 정보를 유지한 상태로 학습하기 위해 개발된 뉴럴 네트워크이다.

### 1.2 CNN, Convolutional Neural Network




출처 : http://taewan.kim/post/cnn/

Convolutional Neural Network(이하 CNN)은 위 이미지와 같이 이미지의 특징을 추출하는 부분과 클래스를 분류하는 부분으로 나눌 수 있다. 특징 추출 영역은 Convolution Layer와 Pooling Layer를 여러 겹 쌓는 형태로 구성된다.

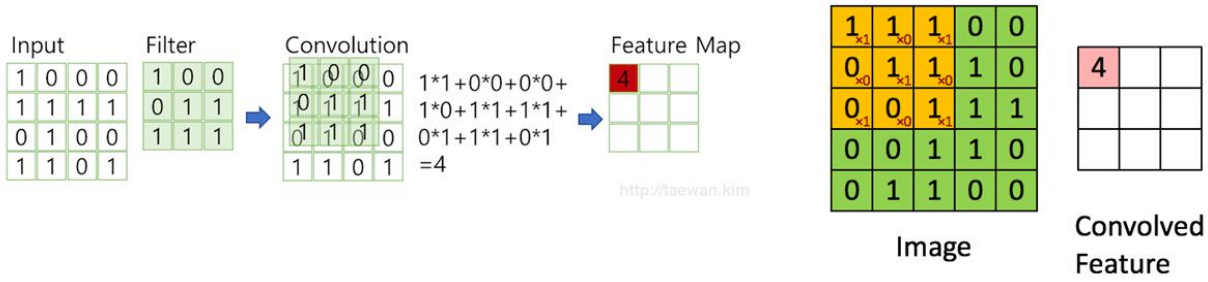
Convolutional Layer는 입력 데이터에 필터를 적용한 후 활성화 함수를 반영하는 필수 요소이다. Convolutional Layer 다음에 위치하는 Pooling Layer는 선택적인 레이어이다. CNN 마지막 부분에는 이미지 분류를 위한 Fully Connected Layer가 추가된다. CNN과 FC Layer를 연결하기 위해 이미지의 특징을 추출하는 부분과 이미지를 분류하는 부분 사이에 이미지 형태의 데이터를 배열 형태로 만드는 Flatten Layer가 위치한다. CNN은 이미지 특징 추출을 위하여 입력 데이터 위를 필터가 순회하며 합성곱을 계산하고, 그 계산 결과를 이용하여 Feature map을 만든다. Convolution Layer는 Filter 크기, Stride, Padding 적용 여부, Max Pooling 크기에 따라서 출력 데이터의 shape이 변경된다.

Convolutional Layer는 기존의 Fully Connected Neural Network와는 달리, 각 레이어의 입출력 데이터의 형상을 유지하며, 이미지의 공간 정보를 유지하고 인접 이미지와의 특징을 효과적으로 인식할 수 있다. 또한 복수의 필터로 이미지의 특징을 추출하고 학습하며, 추출한 이미지의 특징을 모으고 강화하는 Pooling layer와 연결된다. 또한 필터를 공유 파라미터로 사용하기 때문에, 일반 인공 신경망과 비교하여 적은 수의 학습 파라미터를 가진다.

## 1.3 CNN의 주요 용어 설명

CNN의 주요 용어들을 그림과 함께 설명하면 다음과 같다.

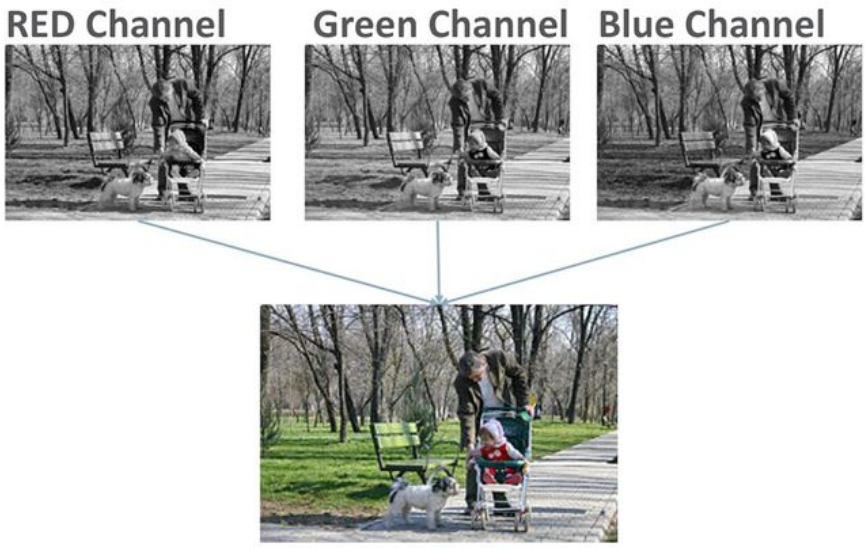
### 1.3.1. Convolution(합성 곱)




출처 :http://taewan.kim/post/cnn/,
https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363

필터는 입력 데이터를 지정한 간격으로 순회하면서 합성곱(convolution)을 계산한다. 여기서 지정된 간격으로 필터를 순회하는 간격을 Stride라고 한다. 위의 그림은 채널이 1개인 입력 데이터를 (3, 3) 크기의 필터로 합성곱하는 과정을 설명한다. 합성곱을 통해 데이터의 공간 정보가 유지될 수 있다.

### 1.3.2 Channel(채널)

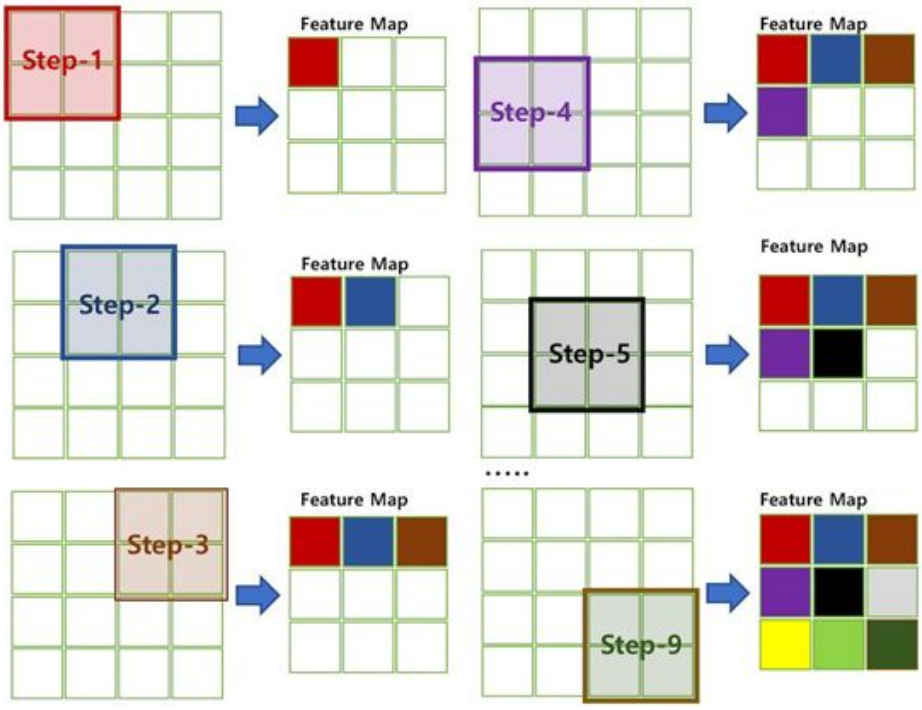



이미지 출처: https://en.wikipedia.org/wiki/Channel_(digital_image)

이미지 데이터에서의 픽셀 하나하나는 실수 값이다. 컬러 사진은 천연색을 표현하기 위해서 각 픽셀을 RGB 3개의 실수로 표현한 3차원 데이터이다. 반면에 흑백 명암만을 표현하는 흑백 사진은 3차원이 아닌 2차원 데이터로 1개 채널로 구성된다. 높이가 39 픽셀이고 폭이 31 픽셀인 컬러 사진 데이터의 shape은 (39, 31, 3)으로 표현한다. 반면에 해당 사진 데이터가 흑백일 경우 shape은 (39, 31, 1)이다.

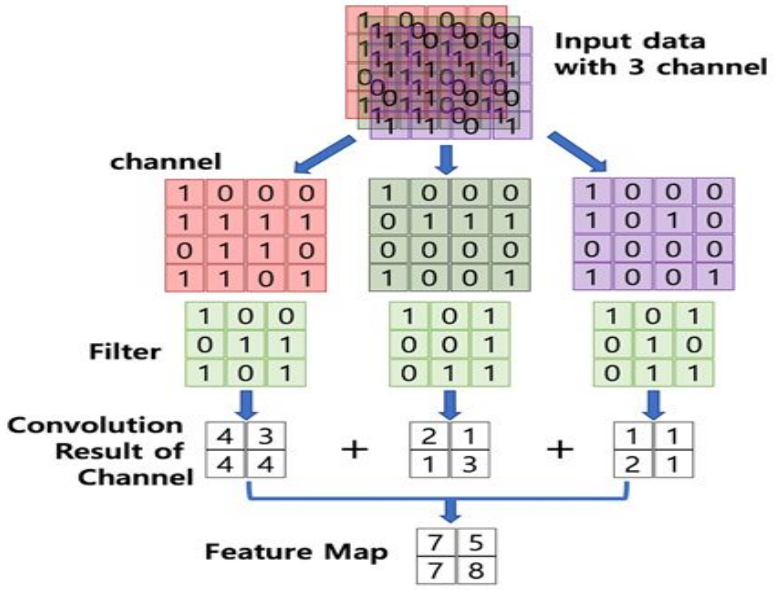
Convolution Layer에 유입되는 입력 데이터에는 한개 이상의 필터가 적용된다. 입력 데이터가 convolution layer의 한개의 필터를 거치고 나면 Feature map의 하나의 채널을 구성하게 된다. Convolution Layer에 n개의 필터가 적용된다면 출력 데이터는 n개의 채널을 갖게 된다.

## 1.3.3. Filter(필터) & Stride(스트라이드)




출처 : http://taewan.kim/post/cnn/

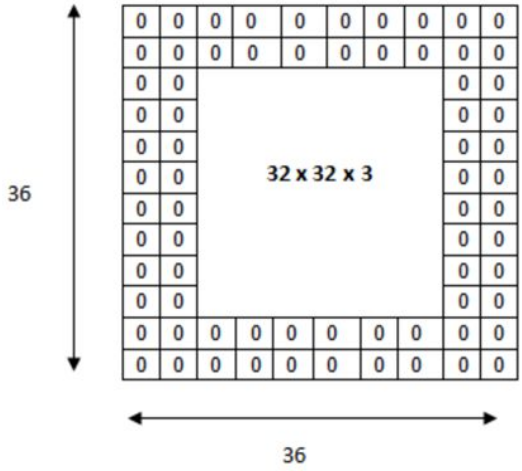
필터는 이미지의 특징을 찾아내기 위한 공용 파라미터이다. Filter 를 Kernel이라고 하기도 한다. 필터는 일반적으로 (4, 4)나 (3, 3)과 같은 정사각행렬로 정의된다. CNN에서 학습의 대상은 필터의 파라미터이다. 그림과 같이 입력 데이터를 지정된 간격으로 순회하면서 채널별로 합성곱을 한다. 여기서 필터가 데이터를 순회하는 간격을 Stride라고 한다. 그림은 stride가 1인 필터가 입력 데이터를 순회하는 모습이다. stride가 2인 경우 필터는 2칸씩 이동하며 합성곱을 계산한다.




출처 : http://taewan.kim/post/cnn/

Convolution Layer의 입력 데이터를 필터가 순회하며 합성곱을 통해 만든 결과물을 feature map 또는 activation map이라고 한다. 하나의 Convolution Layer에 크기가 같은 여러 개의 필터를 적용할 수 있다. 이 경우에 feature map에는 필터 갯수만큼의 채널이 만들어진다. 즉, 입력 데이터에 적용된 필터 갯수는 출력 데이터인 feature map의 채널의 수가 된다. 예를 들어, 컬러 이미지와 같이 입력 데이터가 여러 채널을 갖는 경우, 필터는 각 채널을 순회하며 합성곱을 계산한 후 채널별로 feature map을 만든다. 그리고 각 채널의 feature map을 합산한 후 최종 feature map이 반환된다. 입력 데이터는 채널 수와 상관없이 필터 1개 당 1개의 feature map이 만들어진다.

### 1.3.4. Padding(패딩)

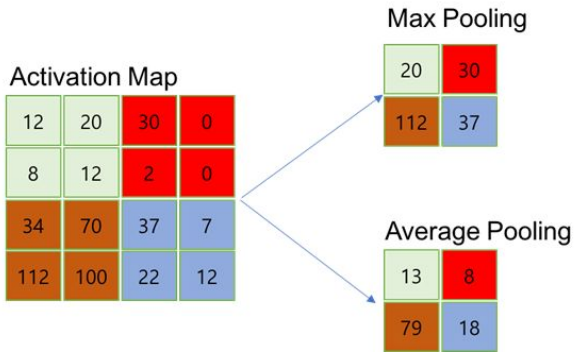



출처:
https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

Convolution Layer를 거치고 나면, filter size와 stride의 작용으로 인해 feature map의 크기는 입력 데이터보다 작아진다. Convolution Layer의 출력 데이터가 줄어들 경우 여러 개의 Layer를 쌓을 수 없게 되며, 그 결과 데이터로부터 더욱 정확한 정보를 얻는 것이 어려워질 수 있다. Convolution Layer의 출력 데이터가 줄어드는 것을 방지하기 위해 패딩이 고안되었다. 패딩은 입력 데이터의 외곽에 지정된 픽셀만큼 특정 값으로 채워 넣는 것을 의미한다. 보통은 0을 사용한다.

그림은 (32, 32, 3) 데이터의 외곽에 2 pixel을 추가하여 (36, 36, 3) 행렬을 만드는 모습이다. Padding을 통해 Convolution Layer의 출력 데이터의 사이즈를 조정할 수 있으며, 그 밖에도 외곽을 '0'으로 둘러쌈으로써 학습 과정에서 인공신경망이 이미지의 외곽을 인식하도록 하는 학습 효과도 있다.

### 1.3.5. Pooling Layer




출처 : http://taewan.kim/post/cnn/

Pooling Layer는 Convolution Layer의 feature map을 입력으로 받아서, 출력 데이터의 크기를 줄이거나 특정 데이터를 강조하는 용도로 사용된다. Pooling Layer를 처리하는 방법으로는 여러가지가 있지만, 그 중에서 그림에서 보여지는 Max Pooling과 Average Pooling이 빈번하게 사용된다. 정사각행렬의 특정 영역 내에 속한 값들의 최댓값 또는 평균을 구하는 방식으로 작동한다. 일반적으로 Pooling의 크기와 stride를 같게 설정하여 모든 원소가 한번씩 처리되도록 한다.

Pooling Layer는 Convolution Layer와 달리, 1) 학습의 대상이 되는 파라미터가 없다. 2) Pooling Layer를 통과하면 행렬의 크기가 감소한다. 3) Pooling Layer를 통해 채널의 수가 변경되지 않는다는 특징이 있으며, CNN에서는 그 중 주로 Max Pooling을 사용한다.

## 1.4 레이어별 출력 데이터 산정

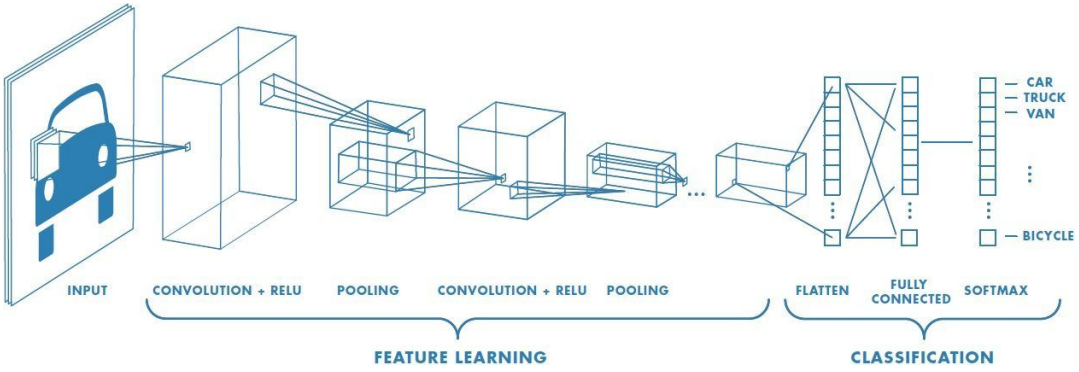
Feature map의 크기는 입력 데이터에 대한 필터의 크기와 stride의 크기에 따라서 결정된다. 공식은 다음과 같다.

$$OutputHeight = OH = \frac{(H + 2P - FH)}{S} + 1$$
$$OutputWeight = OW = \frac{(W + 2P - FW)}{S} + 1$$

(H, W: 입력데이터의 높이와 폭, FH, FW: 필터의 높이와 폭, S: Stride, P: Padding size)

위 식의 결과는 자연수가 되어야 한다. 또한 Convolution Layer 다음에 Pooling Layer가 온다면, feature map의 행과 열의 크기는 pooling layer 크기의 배수여야 한다. 만약 pooling layer가 (3, 3)이라면, 위 식의 결과는 3의 배수인 자연수여야 한다. 이 조건을 만족하도록 filter의 크기, stride, pooling layer의 크기 및 padding의 사이즈를 조절해야 한다.

## 1.5 CNN Architecture



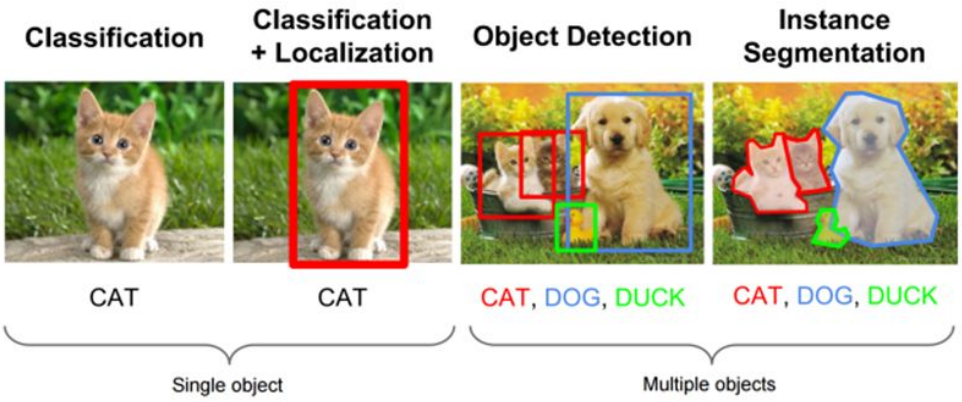

출처: https://kr.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

그림은 전형적인 CNN 구성을 보여준다. CNN은 Convolution Layer와 Max Pooling layer를 반복적으로 쌓는 '특징 추출(feature extraction)' 부분과, fully connected layer를 구성하고 마지막 출력층에 softmax를 적용한 '분류(classifiaction)' 부분으로 나뉜다. CNN을 구성하면서 filter, stride, padding을 조절하여 특징 추출부분의 입력과 출력 크기를 계산하고 맞추는 작업이 중요하다.

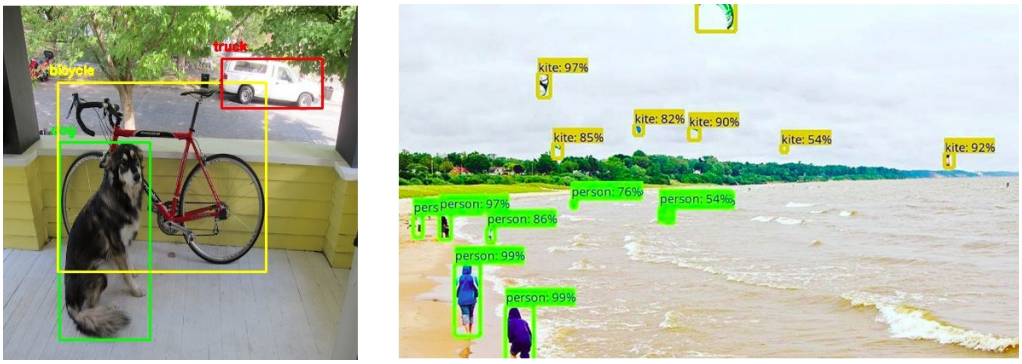
# 2. Detection/Segmentation

## 2.1 Object Detection

### 2.1.1. Object Detection이란?




출처:https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852

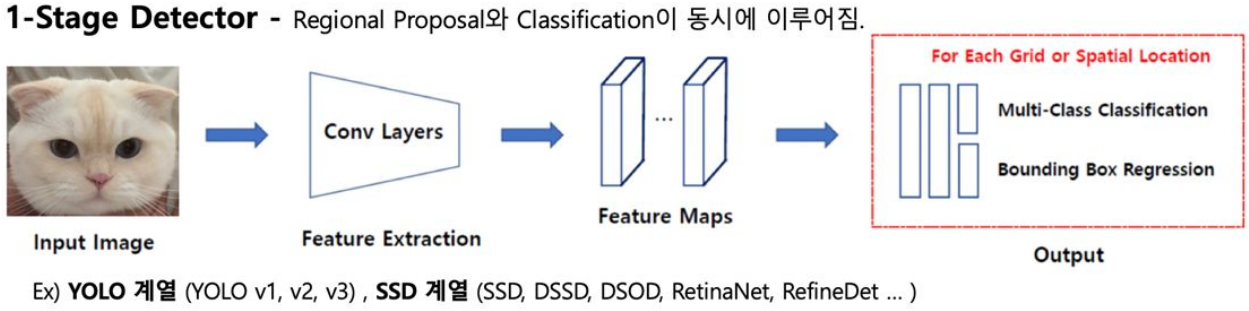


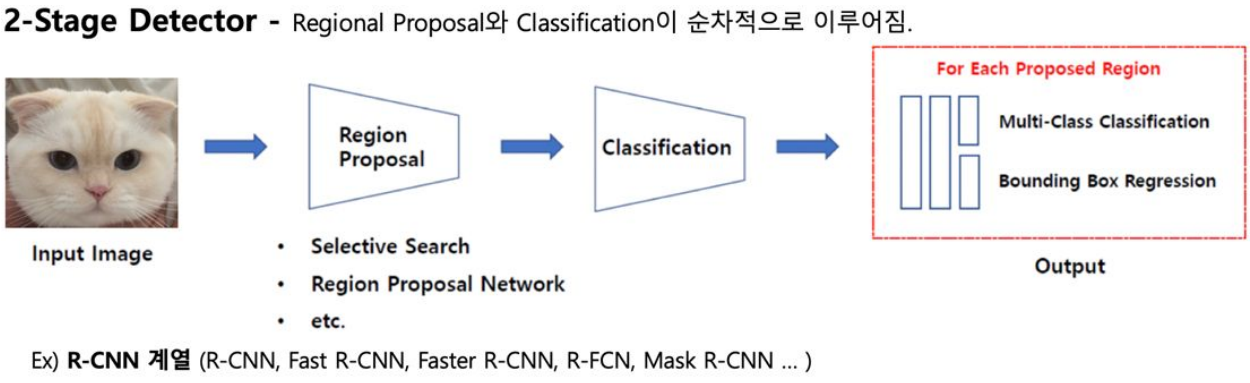

출처:
https://medium.com/@enriqueav/object-detection-with-yolo-implementations-and-how-to-use-them-5da928356035

Object Detection이란 여러 물체에 대해 어떤 물체인지 분류하는 Classification 문제와 그 물체가 어디에 있는지 파악하여 박스를 통해(Bounding Box) 위치 정보를 나타내는 Localization 문제를 풀어낸다. 쉽게 말해서 Object Detection = Classification +

Localization이다. Object Detection을 자율주행자동차, CCTV Surveillance, 스포츠 경기, 무인 점포 등등 많은 곳에 쓰인다.

### 2.1.2. 2-Stage Detector VS 1-Stage Detector



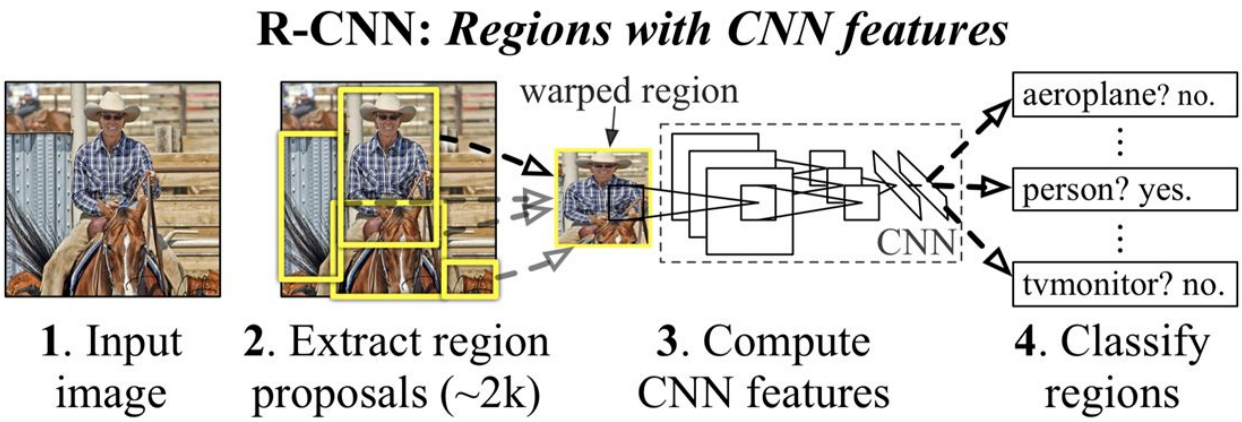




출처: https://nuggy875.tistory.com/21

Deep Learning을 이용한 Object Detection은 크게 2-Stage Detector와 1-Stage Detector로 나눌 수 있다. 1-stage와 2-stage의 구분은 classification과 localization을 동시에 수행하느냐 순차적으로 수행하느냐이다. 따라서 1-stage Detector는 비교적으로 빠르지만 정확도가 낮고, 2-stage Detector는 비교적으로 느리지만 정확도가 높다. 2-stage Detector는 CNN을 처음 적용시킨 R-CNN 계열이 대표적이며, 1-stage Detector는 YOLO(You Look Only Once) 계열과 SSD 계열 등이 포함된다.

## 2.1.3. R-CNN

2-stage Detector R-CNN 계열의 선두주자이자, Object Detection 분야에 최초로 Deep Learning(CNN)을 적용시킨 R-CNN 논문을 소개하고자 한다. 이미지 분류(classification)는 한 개의 객체(Object)가 그려져 있는 이미지가 있을 때, 이 객체가 무엇인지 알아내는 문제이다. 이와 다르게 물체 인식(Object Detection) 알고리즘은 이미지 내에 관심이 있는 객체의 위치(Region of Interest)에 물체의 위치를 알려주기 위한 Bounding Box를 그려줘야 하고, 다수의 Bounding Box를 다양한 Object 종류에 대하여 찾아줘야 하기 때문에 이미지 분류보다는 훨씬 복잡한 문제이다. 때문에 2012년 CNN의 등장 이후 Object Detection분야에는 적용되지 못하다가 2014년 R-CNN의 등장으로 Object Detection분야에 최초로 적용되었다.




출처: https://arxiv.org/pdf/1311.2524.pdf

R-CNN의 구조는 다음과 같다.

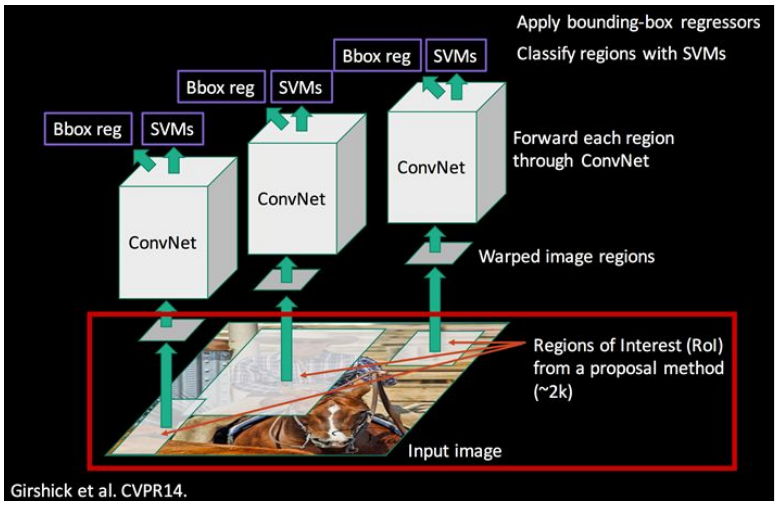
1. 이미지를 input으로 넣는다.
2. Selective Search 알고리즘을 통해 2000개의 영역(Bounding Box)를 추출하여 잘라낸다.(Cropping)
3. 이를 CNN모델에 넣기 위해 각 Box들을 전부 동일한 사이즈로 (224 x 224) 찌그러뜨린다.(Warping)
4. 2000개의 찌그러든 이미지를 CNN 모델에 각각 집어넣는다.
5. 각각 Classification을 진행하여 결과를 도출한다.

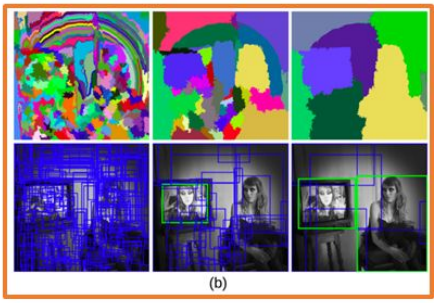
R-CNN은 2-stage Detector로서, 물체의 위치를 찾는 일(Region Proposal)과 찾은 물체를 분류하는 일(Region Classification)을 단계적으로 진행한다. 이 논문에서는 총 세 가지의 모듈로 두 task를 수행한다.

1) Region Proposal - 카테고리와 무관하게 물체의 영역을 찾는 모듈
2) CNN - 각각의 영역으로부터 고정된 크기의 feature vector를 뽑아내는 CNN
3) SVM - classification을 위한 선형 지도학습 모델(Support Vector Machine)

위 세가지 구조를 토대로 R-CNN의 흐름을 파악해보겠다.

### 2.1.3.1. Region Proposal




출처: https://donghwa-kim.github.io/SelectiveSearch.html,
https://www.saagie.com/blog/object-detection-part2/

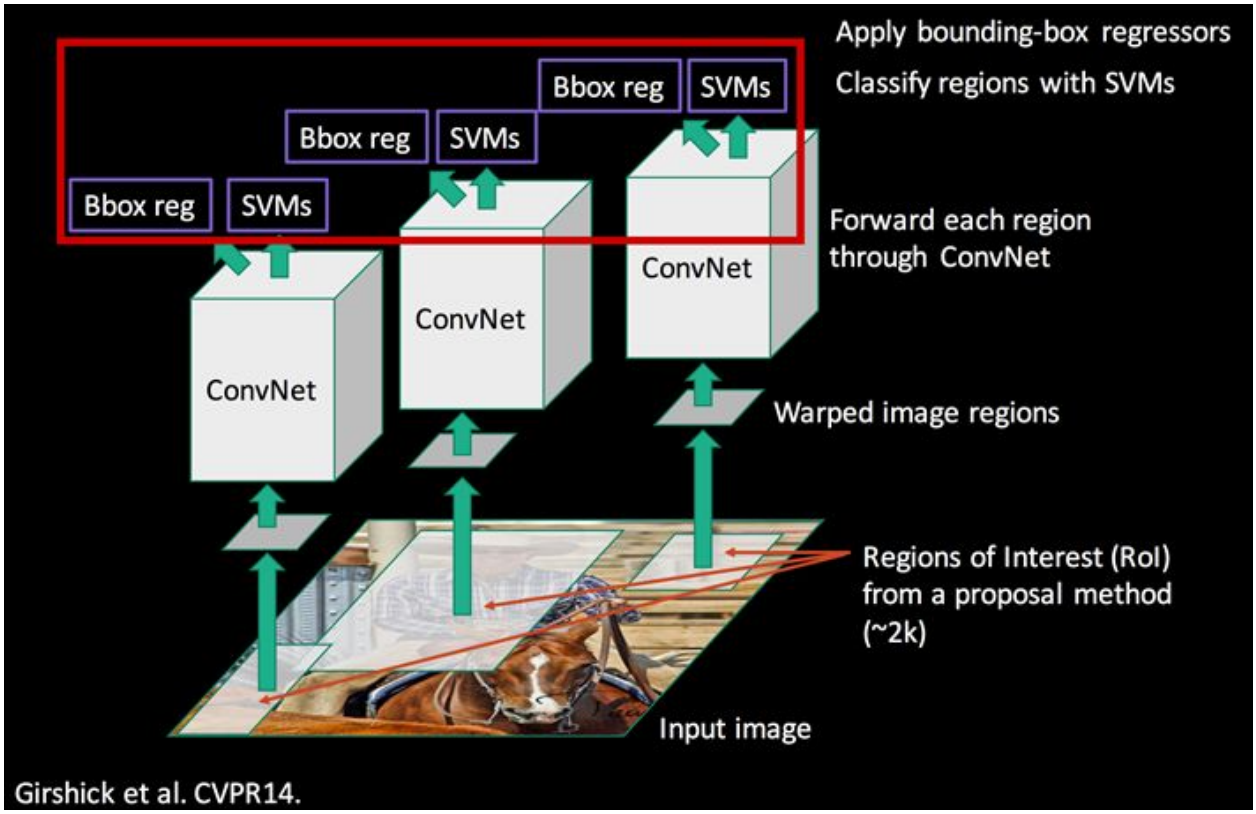
R-CNN은 Region Proposal 단계에서 Selective Search라는 알고리즘을 사용한다. 이 알고리즘은 객체와 주변 간의 색감, 질감 차이, 다른 물체에 에워쌓여있는지 등의 여부를 통해 물체의 위치를 파악할 수 있는 알고리즘이다. 오른쪽 그림과 같이 Bounding Box들을 랜덤하게 많이 생성하고, 이들을 조금씩 합해나가며 물체를 인식한다. 이 알고리즘을 통해 한 이미지에서 2000개의 Region을 뽑아내고, 이들을 모두 CNN에 넣기 위해 같은 사이즈(224 x 224)로 찌그러뜨려 통일시키는 작업(warping)을 거친다.

2.1.3.2. CNN(Convolutional Neural Network)

앞서 Selective Search를 통해 생성된 2000개의 이미지를 224 x 224로 Warping하여 각각 CNN에 넣어준다. 여기서 CNN은 CNN Architecture의 시초격인 AlexNet의 구조를 거의 그대로 가져다 썼으며, Object Detection 용으로 마지막 부분만 조금 수정되었다.

각각의 warped된 224 x 224 이미지는 CNN을 거치며 고정된 길이의 4096차원의 벡터(4096 dimensional feature vector)로 출력된다.

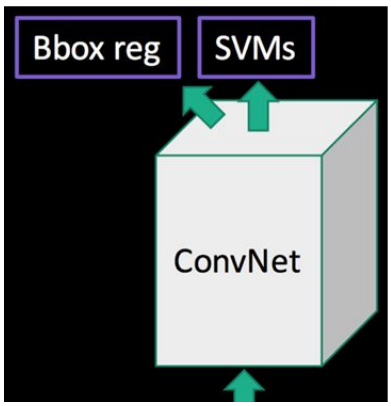
2.1.3.3. SVM(Support Vector Machine)




출처: https://nuggy875.tistory.com/21

CNN 모델로부터 feature vector가 추출되고 training label이 적용되고 나면, Linear SVM을 이용하여 classification이 진행된다. SVM은 CNN으로부터 추출된 각각의 feature vector들의 점수를 class별로 매기고, 1) 객체인지 아닌지 2) 객체라면 어떤 객체인지 등을 판별하는 역할을 한다.

2.1.3.4. Bounding Box Regression




$$\{(P^i, G^i)\}_{i=1,\ldots,N}, \text{where } P^i = (P^i_x, P^i_y, P^i_w, P^i_h)$$

$$\hat{G}_x = P_w d_x(P) + P_x \quad (1)$$
$$\hat{G}_y = P_h d_y(P) + P_y \quad (2)$$
$$\hat{G}_w = P_w \exp(d_w(P)) \quad (3)$$
$$\hat{G}_h = P_h \exp(d_h(P)). \quad (4)$$
$$t_x = (G_x - P_x)/P_w \quad (6)$$
$$t_y = (G_y - P_y)/P_h \quad (7)$$
$$t_w = \log(G_w/P_w) \quad (8)$$
$$t_h = \log(G_h/P_h). \quad (9)$$
$$\mathbf{w}_\star = \underset{\hat{\mathbf{w}}_\star}{\operatorname{argmin}} \sum_i^N (t^i_\star - \hat{\mathbf{w}}_\star^{\mathrm{T}} \phi_5(P^i))^2 + \lambda \|\hat{\mathbf{w}}_\star\|^2. \quad (5)$$
$$d_\star(P) = \mathbf{w}_\star^{\mathrm{T}} \phi_5(P)$$

출처: https://nuggy875.tistory.com/21

Selective Search로 만들어낸 Bounding Box가 물체를 정확히 잡아내지 못하는 경우를 대비하여, 물체를 정확히 감싸도록 조정해주는 선형회귀 모델(Bounding Box Regression)을 넣었다. BBox의 input 값은 N개의 Training Pairs로 이루어져 있다. x, y, w, h,는 각각 Bounding Box의 x, y좌표(위치), box의 width(너비)와 height(높이)이다. P는 선택된 Bounding Box이고 G는 Bounding Box의 Ground Truth(실제값)이다. 선택된 P를 G에 맞추도록 transform하는 것을 학습하는 것이 Bounding Box Regression의 목표이다. R-CNN 계열의 방법의 Bounding Box Regression은 전부 오른쪽과 같은 공식을 사용한다.

(1), (2), (3), (4)의 변수들은 G(Ground Truth)와 최대한 가까워져야하는 변수인 G hat이고, 각각을 G와 t로 치환하여 나타낸 것이 (6), (7), (8), (9)이다. (5) 는 loss function으로, t와 d의 차이인 loss를 줄여나가는 방향으로 학습해야 한다. lambda로 표현된 식은 regularization을 위한 것으로, 여기서는 각 공식을 통해 어떤 방식으로 예측값인 P가 실제값인 G에 가깝도록 학습되는지 보는 것이 중요하다.

### 2.1.4 R-CNN의 단점과 의의

초기의 R-CNN에는 여러가지 단점이 존재한다.

#### 2.1.4.1. 처리 속도가 느리다.

R-CNN은 Selective Search에서 뽑아낸 2000개의 영역 이미지들 각각에 대해 CNN을 적용시키기 때문에 2000번의 CNN으로 인해 이미지 처리 속도가 매우 느리다. 따라서 이후의 모델은 전체 이미지를 하나의 CNN에 넣어 나온 feature map에서 물체가 있을 영역을 찾아낸다. 그리고 Region Proposal에 사용되는 Selective가 CPU를 사용하는 알고리즘인 이유도 R-CNN의 프로세스를 느리게 만드는 요인 중 하나이다.

#### 2.1.4.2 복잡하다

R-CNN은 Multi-Stage Training을 수행하며, CNN, SVM, 그리고 Bounding Box Regression까지 총 세가지의 모델을 필요로 하는 복잡한 구조를 가진다.

#### 2.1.4.3. end-to-end Back Propagation이 안된다.

R-CNN은 Multi-Stage Training을 수행하며, SVM, Bounding Box Regression에서 학습한 결과가 CNN을 업데이트 시키지는 못한다. 왜냐하면 뒷 부분의 SVM과 Bounding Box Regression에서 학습한 결과는 ConvNet으로의 역전파가 불가능하여 end-to-end로 학습되지는 않는다.

위와 같은 단점들이 존재함에도 불구하고, R-CNN은 최초로 Object Detection에 Deep Learning 방법인 CNN을 적용시켰다는 점과, 이후 2-stage detector들의 구조에 막대한 영향을 미쳤다는 점에서 의미가 큰 논문이다.

## 2.2 Image Segmentation

앞서 설명한 Object Detection이 물체가 있는 위치를 찾아 물체 주변의 bounding box를 찾는 문제였다면, Image Segmentation은 이미지를 픽셀 단위로 구분해 각 픽셀이 어느 범주에 속하는지 예측한다. 즉, object의 형상을 따라서 object의 영역을 표시하는 것이다.

Image Segmentation의 종류에는 Semantic Segmentation과 Instance Segmentation이 있다.


출처 : http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

### 2.2.1 Semantic Segmentation

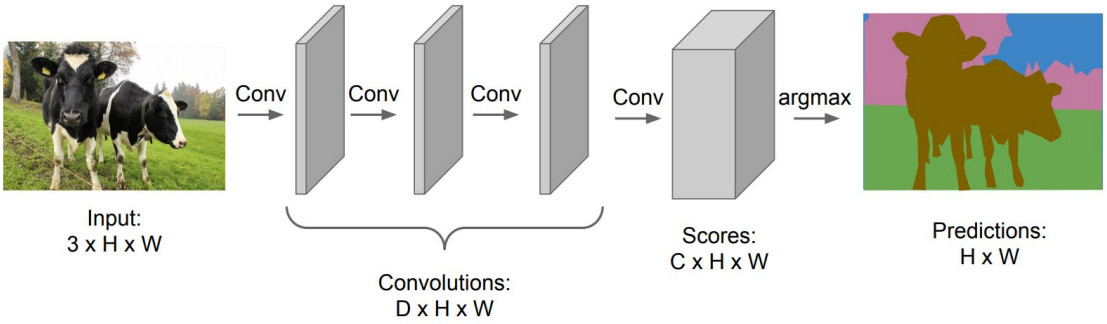
Semantic segmentation은 이미지의 모든 픽셀을 사전에 정의된 class로 분류한다. 여기서 주의할 점이 있는데 semantic segmentation은 동일한 class의 개별 instance를 구별하지 않는다는 것이다.


출처 : http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

예를 들어, 위 이미지의 모든 픽셀은 trees, sky, cow, grass로 분류할 수 있다. 실제 사진에서는 소가 2마리 있다는 것을 알 수 있다. 그러나 semantic segmentation에선 2마리라는 사실을 인지하지 못한다. 오로지 그 픽셀 자체가 어떤 class에 속해있는지에만 관심을 가지기 때문이다. 이를 해결하는 방법은 추후 Instance Segmentation에서 다루도록 한다.

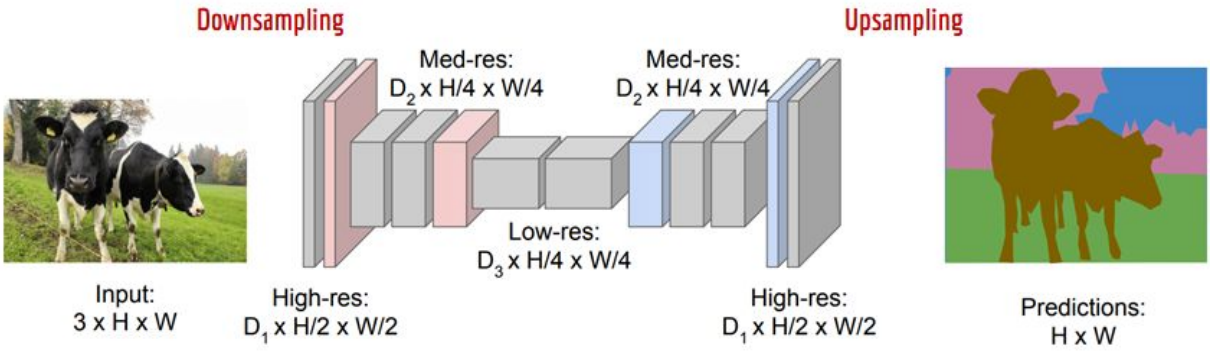
### 2.2.1.1 Fully Convolutional Network (FCN)

Semantic Segmentation에 활용되는 여러 접근방법 중 가장 잘 알려진 모델은 Fully Convolutional Network이다. 앞서 설명한 바와 같이 픽셀 하나하나를 분류해야 하는 만큼 픽셀의 위치가 중요한데, 이미지의 크기를 유지시키기 위해 zero-padding을 집어넣어 convolution을 진행할 수 있다. 그러나 이는 convolutional layer 내 parameter의 개수가 많아져 계산 비용이 크다는 단점이 있다.



출처 : http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

이는 convolutional network 내 downsampling과 upsampling을 추가해 해결이 가능하다. 앞서 배웠듯이 CNN에선 downsampling만을 하여 차원을 줄인 뒤 바로 classification을 진행하는 반면, 이 구조에서는 저차원의 feature map을 다시 upsampling 해 원래 해상도의 이미지로 복구한다. 그 뒤, 각 픽셀마다 class를 예측하고 cross-entropy를 적용시켜 학습시킨다.




출처 : http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

## 2.2.2 Instance Segmentation

Instance Segmentation은 Semantic Segmentation에서 더 발전된 것으로, 같은 class이더라도 서로 다른 instances로 구분해 준다.

출처 : http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

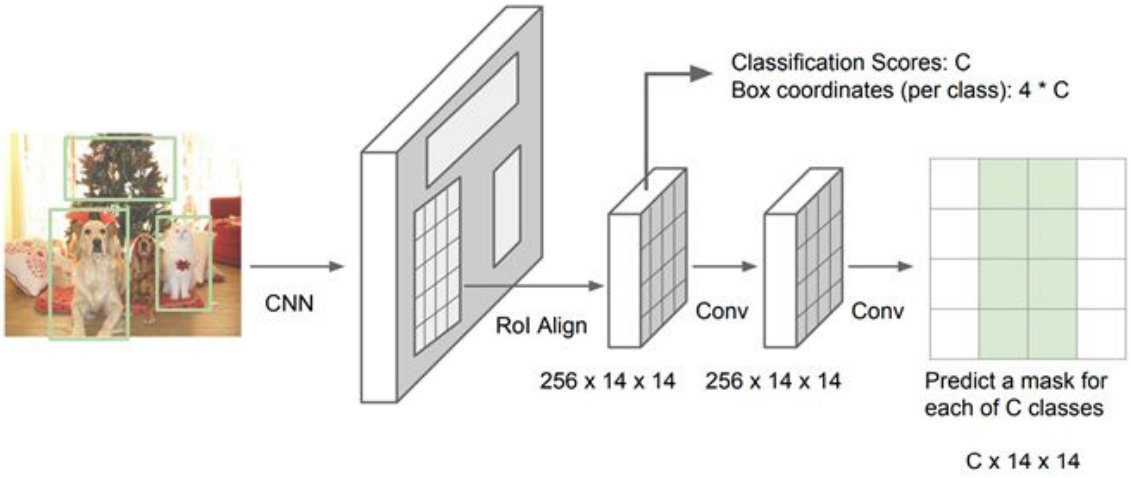
위 이미지에서 여러명의 사람이 겹쳐있고 모두 같은 class 이나 각자 다른 객체로 인식이 되어있는 것을 확인할 수 있다.

### 2.2.2.1 Mask R-CNN

Mask R-CNN은 instance segmentation을 위해 사용되는 기법이며, object detection을 위한 R-CNN과 semantic segmentation을 위한 convolutional network를 합쳐놓은 모델이다.

우선 입력 이미지에서 convolutional layer로 region proposal를 진행한 뒤, 계산의 편의를 위해 region의 크기를 통일시킨다(warping). 그 뒤, 이 region들은 또다른 convolutional layer를 거쳐 픽셀 별로 class 분류된다.

이때 Mask R-CNN이 학습하는 것은 총 3가지다. 1) RoI의 classification이 맞게 이루어졌는지 확인하며, 2) bounding box의 위치가 맞게 설정되었는지 확인한다. 그리고 3) 각 픽셀이 제대로 분류 되었는지 비교한다.



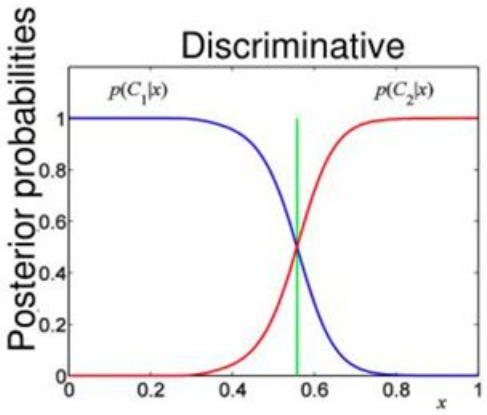

출처 : http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

# 3. Generative Model

## 3.1 Introduction

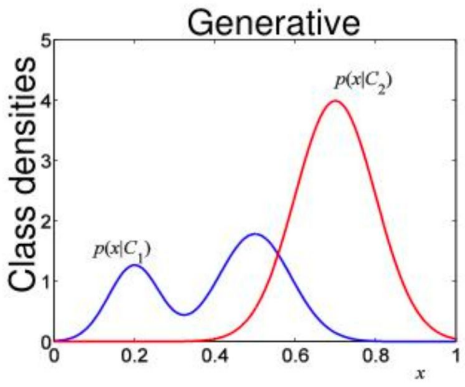
Classification 문제는 크게 discriminative model과 generative model로 나눌 수 있다.

우리가 이전에 배운 모델은 전부 discriminative model에 속한다. Discriminative model은 입력 이미지가 들어왔을 때 오로지 이를 분류하는데만 집중한다. 즉, 특정 입력값 X에 대한 조건부 확률 P(Y|X)를 학습하는 것이다. 예를 들어, input으로 강아지 혹은 고양이의 사진이 주어졌을 때, 이 사진이 강아지일 확률과 고양이일 확률을 배우는 것이다.



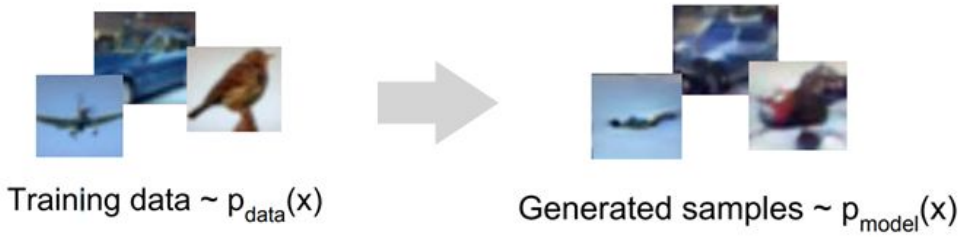

출처 : http://ratsgo.github.io

그에 비해 Generative model의 경우 단순히 input을 분류하는 것을 넘어서 각 class의 분포 자체를 이해한다. 즉, 결합확률분포 P(X,Y)를 학습하는것이다. 이는 generative model이 bayes rule을 사용해 P(Y|X)를 간접적으로 도출할 수 있는 것 뿐만 아니라, class를 정확히 알고 있기 때문에 각 class에 해당되는 새로운 데이터를 생성할 수 있게 된다는 뜻이다. 예시로 강아지와 고양이 사진이 input으로 주어졌을때, generative model은 강아지와 고양이의 특성을 제대로 이해하며 아예 다른 강아지, 고양이의 사진을 만들 수 있다.




출처 : http://ratsgo.github.io

Generative model에 학습시킬 데이터가 주어지면, 모델은 이 데이터의 분포와 동일한 분포의 새로운 sample을 생성한다.




출처 : http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

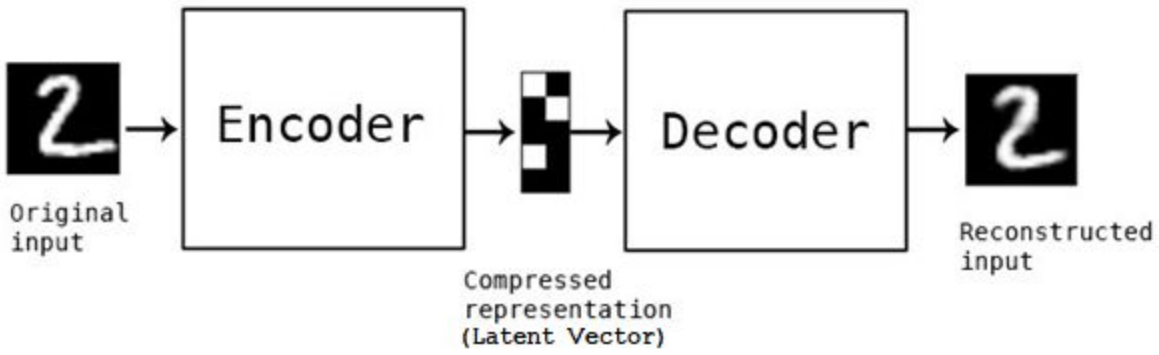
Generative model에는 다양한 유형이 있다. 그 중 가장 유명한 두가지를 다룰 예정이다.

## 3.2 Variational Auto-Encoder (VAE)

### 3.2.1 Auto-Encoder (AE)

VAE를 설명하기에 앞서, VAE의 기초적인 배경이 되는 Auto-Encoder(AE)에 대해 알아보기로 한다. Auto-encoder를 한 문장으로 표현하면 “unsupervised learning for reconstruction of data”이다. Auto-encoder는 데이터 생성을 목적으로 하지 않으며, label

되지 않은 학습 데이터로부터 저차원의 feature map을 얻기 위한 비지도학습이다. Auto-encoder는 encoder와 decoder 두 부분으로 구성되어 있는데, encoder로 입력을 내부 표현(latent vector)으로 변환하고, decoder로 내부 표현(latent vector)을 출력으로 변환한다. 이때 출력은 입력과 최대한 유사하게 구현해야 한다. 따라서 AE에선 encoder가 입력 이미지를 얼마나 잘 축소시켰는지가 키 포인트다. 구조는 convolutional layers를 이용해 설계한다.



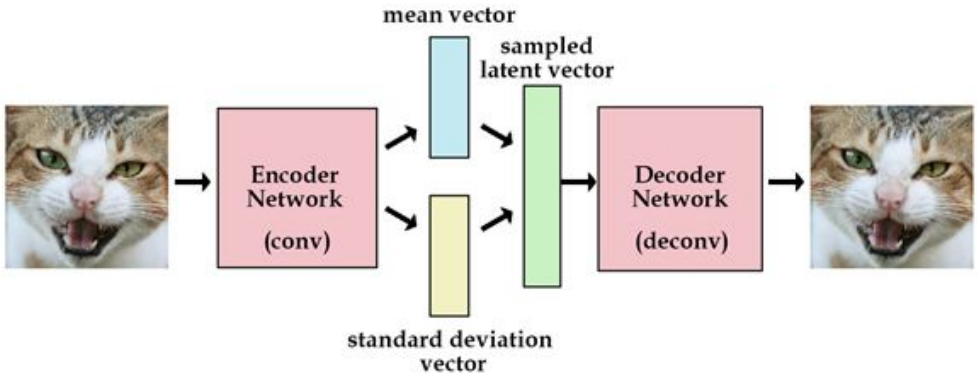

출처 : https://keraskorea.github.io/posts/2018-10-23-keras_autoencoder/

이 Auto-Encoder를 발전시켜 새로운 이미지를 생성할 수 있는 generative model이 개발됐는데, 바로 Variational Auto-Encoder(VAE)이다.

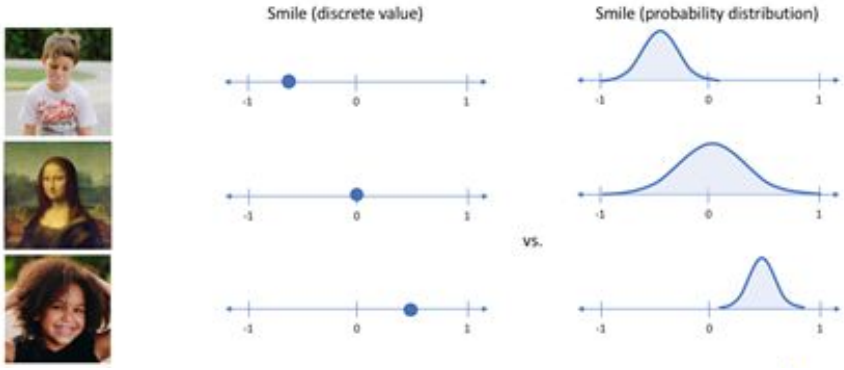
### 3.2.2 Variational Auto-Encoder (VAE)

앞서 설명한 Auto-Encoder(AE)와 마찬가지로 VAE 역시 encoder와 decoder로 이루어져 있다. 그러나 입력 이미지를 압축시키는 과정에서 generative model의 특성이 나타난다. AE와 VAE를 구분짓는 차이점은 바로 이 latent vector의 차이에서 온다는 것이다.

AE에서 feature map으로 차원 축소를 할때, 각 feature는 하나의 값으로 설명된다. 반면, VAE에서는 두개의 latent vector가 만들어지는데, 바로 mean vector와 standard deviation vector다. 이 두 벡터로 인해 각 feature는 하나의 값이 아닌 분포로 표현이 될 수 있다.




출처 : https://ratsgo.github.io

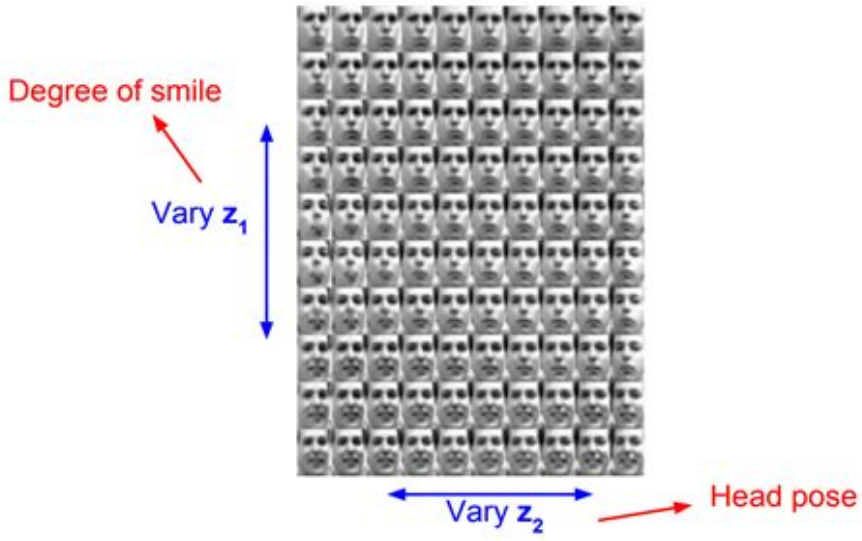



<AE latent vector vs. VAE latent vector>
출처 : https://www.jeremyjordan.me/variational-autoencoders/

입력데이터 내 특징의 평균과 분산을 구함으로써 각 특징에 대한 확률분포를 알아낼 수 있으며, 새로운 데이터를 생성할 수 있는 여건을 갖추게 되는 것이다. 여기서 분포는 정규분포를 전제로 하고 있다.

이렇게 산출된 분포에서 다음으로 random sampling을 진행하고, 샘플링 된 latent vector는 decoder를 통해 원래 해상도로 복구된다. random sampling인 만큼 출력물은 매번 실행될 때 마다 조금씩 변형된다.

다음은 하나의 입력 이미지를 VAE를 통해 출력한 결과물들이다.

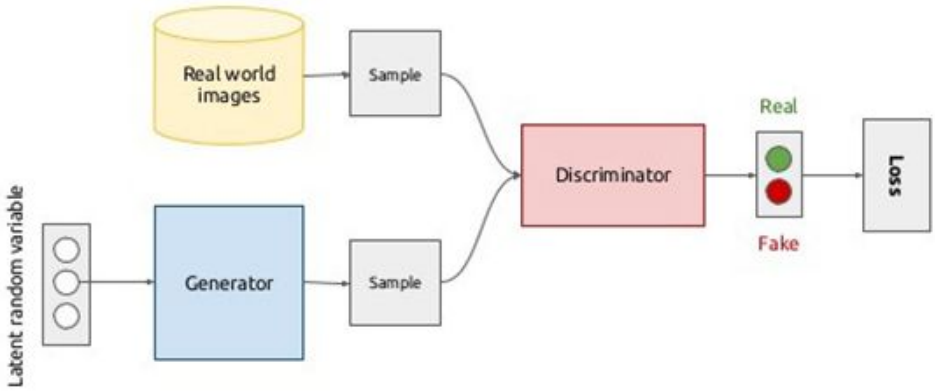



출처 : http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

## 3.3 Generative Adversarial Network (GAN)

### 3.3.1 GAN

명시적으로 분포를 모델링하는 VAE와 달리, GAN은 데이터의 밀도 함수를 적용하지 않고 게임 이론으로 접근한다. 이름 그대로 대립 학습(adversarial learning)을 통해 생성모델(generative model)을 구축하는 신경망(Neural Network)을 뜻한다. GAN은 두가지 모델을 동시에 사용하는데, 가짜 데이터를 생성하는 Generator와 진짜, 가짜를 구분하는 Discriminator가 그것으로 두 모델은 교대로 학습하며 경쟁하게 된다. Generator는 noise vector를 입력으로 받아 이를 실제 데이터와 매우 유사하게 생성해 Discriminator를 속이도록 학습하며 Discriminator는 이 가짜 데이터가 진짜 데이터로 오판이 되지 않도록 정확히 구별하는 것을 학습한다.



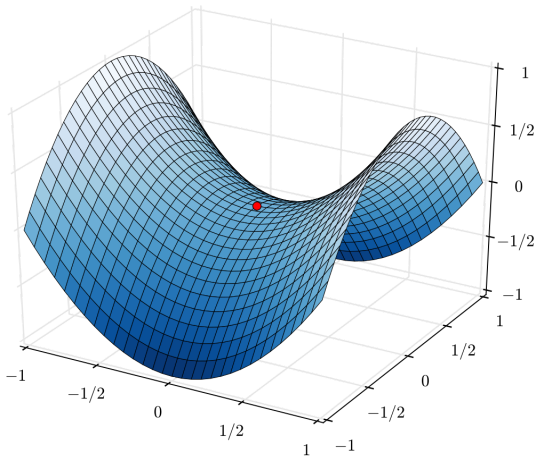

출처 : https://www.elfarchive.org/2017/11/generative-adversal-network.html

GAN은 objective function으로 아래 minimax problem을 풀게 된다.

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

위 수식에서 D(x)는 discriminator의 출력값이고 진짜일 확률을 의미하는 0과 1사이의 값이다. 따라서 데이터가 진짜일땐 1, 가짜일땐 0을 내놓는다. 따라서 두번째 항의 D(G(z))는 Generator가 만든 데이터 G(z)가 진짜라고 판단하면 1, 아니면 0을 출력한다. Discriminator는 D(x)를 1에 가깝게, D(G(z))를 0에 가깝게 해야 하기 때문에 위 식을 최대화시키도록 학습한다. Generator는 D(G(z))를 1에 가깝게 해야 하기 때문에 위 식을 최소화해야 한다.

이렇게 training을 하다 보면 가장 최적인 지점에 도달을 하게 되는데, 만들어낸 데이터의 분포와 실제 데이터의 분포가 같을 때($p_{data}=p_g$) optimal point이며, 이를 nash equilibrium이라 한다. 그러나 이렇게 번갈아 가며 최대화, 최소화를 하는 과정이 다소 unstable해 optimal point근처에 계속 맴돌며 도달을 못하는 경우가 발생할수 있다.

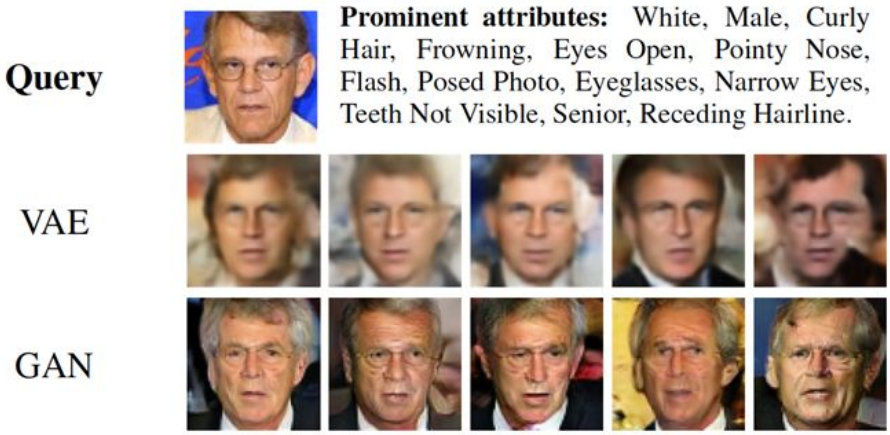



출처 : https://en.wikipedia.org/wiki/Saddle_point

학습이 끝나면 최종적으로 generator 부분만을 떼어내 새로운 이미지를 생성하는데 쓴다.

### 3.3.2 VAE와의 비교

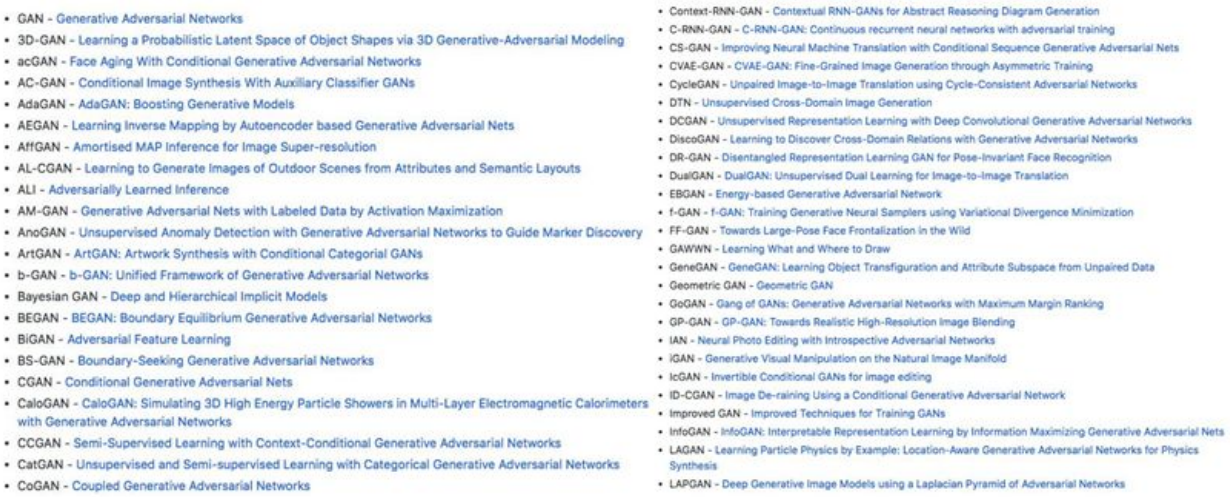
VAE과 GAN을 비교하면 어떤 차이점이 있을까? 우선 VAE는 기존의 입력 데이터의 분포로부터 비슷한 데이터를 생성하며, 학습을 할때 비교적 안정적이고 쉽다. 그러나 출력 결과물이 선명하지 않다는 단점이 있다. 그에 반해 GAN은 입력 데이터로부터 아예 다른 분포의 데이터 sample을 만들며, 분포를 이해하는 것이 아니라 generator가 알아서 학습하는 것이기 때문에 복잡한 분포도 학습이 가능하다. 또한 discriminator를 속일만큼 가짜 데이터와 진짜 데이터와 구분이 가지 않아야 하기 때문에 출력 이미지는 매우 선명하게 표현된다. 그러나 앞서 말했던 minimax problem의 non-converge 하는 특성으로 인해 학습이 불안정하다는 단점이 있다. 아래 사진은 query가 주어졌을 때 각각 VAE와 GAN가 출력한 결과물이다.




출처 : https://lh4.googleusercontent.com

### 3.3.3 Types of GANs

GAN은 2014년 Ian Goodfellow로부터 개발됐는데, 그 이후로 많은 종류의 GAN이 쏟아져 왔으며, 현재는 GAN에 대한 새로운 논문이 매주 발표된다. 아래는 'The GAN Zoo'라는 GAN 관련 논문들을 정리한 repository이다.




출처 : https://github.com/hindupuravinash/the-gan-zoo

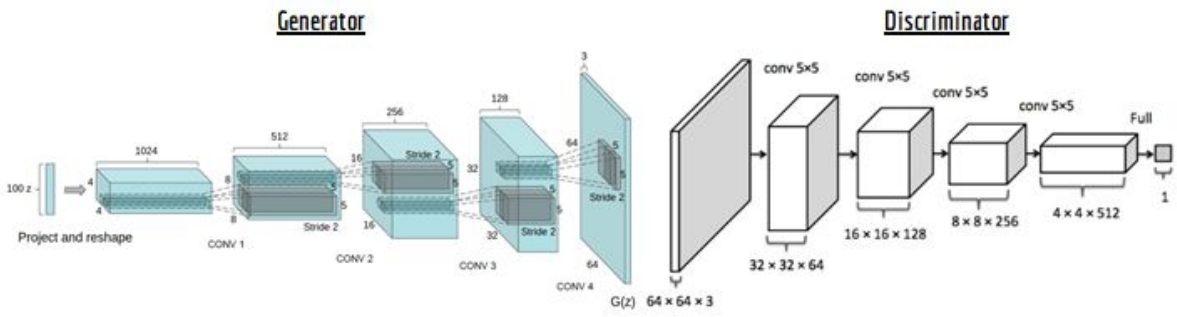
그 중 가장 많이 쓰이는 GAN 종류 세가지를 살펴볼 예정이다.

3.3.3.1 Deep Convolutional GAN (DCGAN)

DCGAN은 가장 성공적인 GAN 디자인 중 하나인데, 기존 GAN 구조의 Generator와 Discriminator에 convolutional layers를 적용한다. 구조의 상세한 내용은 다음과 같다.

- Generator는 fractionally-strided convolutions를 사용해 up-sampling을 시행한다.
- Discriminator는 max pooling 없이 strided convolutions를 사용해 down-sampling을 시행한다.
- fully connected hidden layers 와 pooling을 없앤다.
- batch normalization을 사용한다
- Generator는 ReLU와 tanh 활성화 함수를, Discriminator는 LeakyReLU와 sigmoid를 사용한다.

Max pooling을 사용하지 않는 이유는 데이터 생성 시 객체의 윤곽과 위치가 중요한데, 이는 반대로 pixel들의 관계를 유연하게 만들어 주기 때문이다.




출처 : https://angrypark.github.io/

아래 이미지는 DCGAN을 사용해 만들어진 가상의 침실이다.

출처 : https://angrypark.github.io/

또한 DCGAN는 vector arithmetic하다. Word embedding에서 “king”-”man”+”woman” = “queen”인 것처럼 이미지도 벡터 연산이 가능하다.

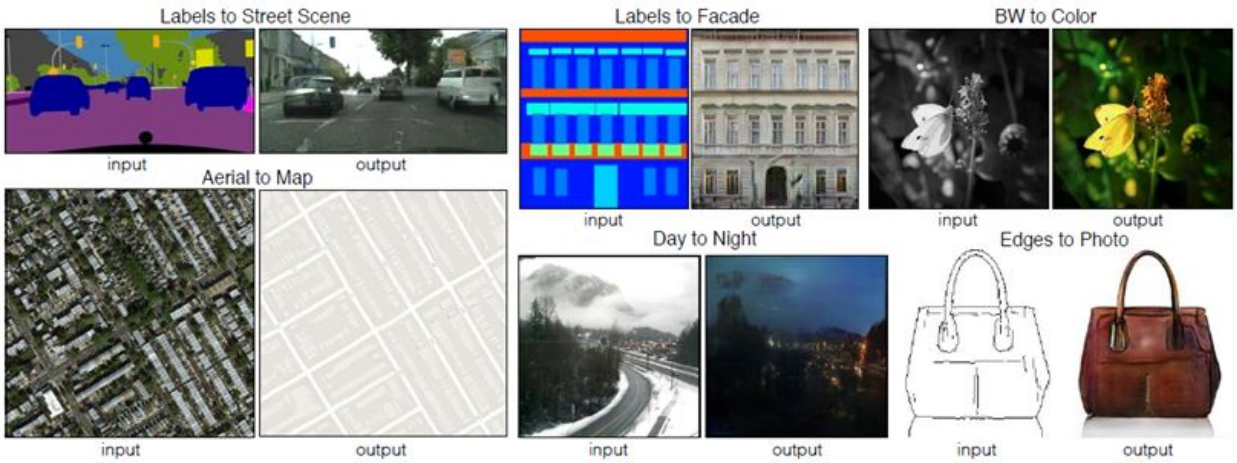

출처 : http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

3.3.3.2 Conditional GAN (cGAN)

cGAN이란 기존 GAN의 Generator와 Discriminator에 y라는 조건이 붙는 것이다. y는 어떠한 정보라도 될 수 있는데, 그림의 윤곽이나 흑백-컬러 변환 등을 설정할 수 있다. 이 조건을 입력에 같이 집어넣어 학습시킨다. Objective function은 아래와 같다.

$$min_G max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[logD(x|y)] + \mathbb{E}_{x \sim p_z(z)}[log(1 - D(G(z|y)))]$$

다음은 cGAN을 사용하여 생성된 예시들이다.



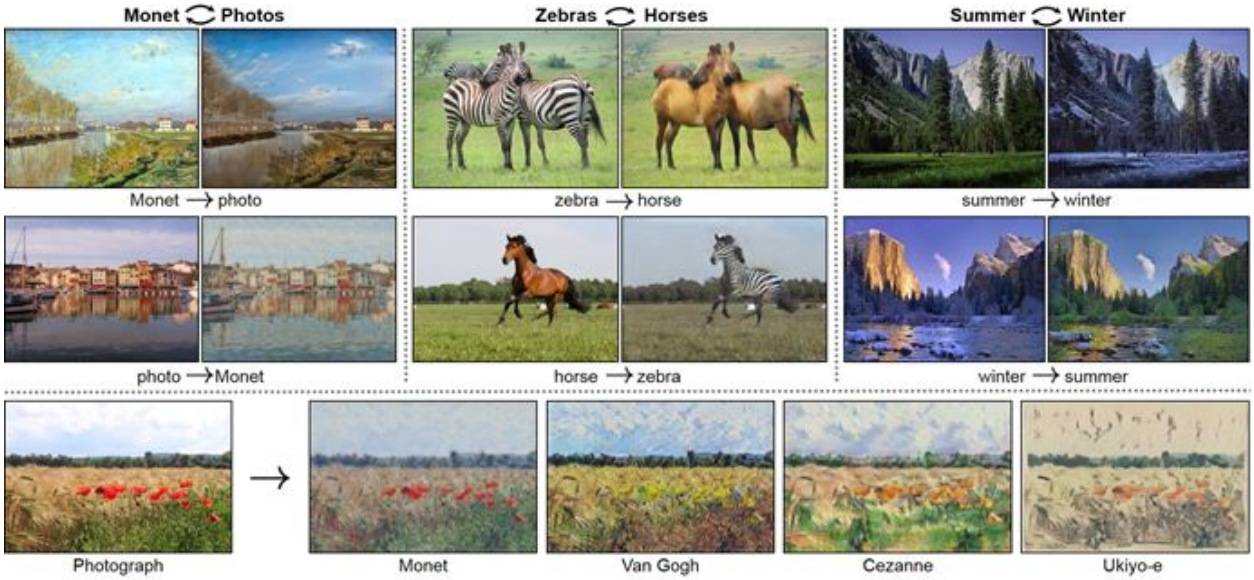

출처 : http://wewinserv.tistory.com

조건이 붙은 cGAN 모델을 학습하기 위해선 실제 사진에서 원하는 조건의 label image를 추출한다. 예를 들어 BW to Color 모델을 학습하기 위해 실제 컬러 이미지를 흑백으로 바꾼 후 모델이 얼마나 채색을 잘했는지 원래 사진과 비교하는 것이다.

3.3.3.3 CycleGAN

CycleGAN은 짝지어지지 않은 이미지를 변환할 때 아주 유용하게 쓰인다. 예를들어, 고흐의 그림을 실제 사진으로 변환하고 싶을때, 고흐의 그림과 실제 풍경은 paired되어 있지 않아 비교가 불가능 하다. 따라서 실제 사진을 모델에 적용시킨 결과물을 반대로 적용했을 때 원본과 얼마나 유사한지를 확인하는 것이다. 결론적으로 2개의

discriminator와 2개의 generator를 학습하는 것이며, 양방향 변환이 가능하다. 아래는 CycleGAN을 사용하여 생성된 예시들이다.




출처 : http://junyanz.github.io