## Convolutional Neural Networks

[합성곱 연산 – (수직)에지 검출]

그레이 스케일 6 X 6 X 1 행렬 이미지

수직선 엣지를 검출하기 위해 3 X 3 행렬의 필터(커널) 필요

=> 아다마르 곱 취하여 컨볼브하면 4 X 4 행렬의 이미지 생성 (6-3+1)

- 제대로 생성되지 못하면, 이미지의 크기를 크게 하기

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

Vertical

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 0 | 0 | 0 |
| -1 | -1 | -1 |

Horizontal

수직엣지 : 왼쪽은 밝고, 오른쪽은 어두움

수평엣지 : 위쪽이 밝고, 아래쪽이 어두움

1000 X 1000 이미지처럼 매우 큰 경우, 전환 영역을 볼 수 없다. 중간 값은 이미지 사이즈에 비해 꽤 작다. 다양한 필터를 사용해 수직 및 수평 엣지를 찾을 수 있다.

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 2 | 0 | -2 |
| 1 | 0 | -1 |

Sobel filter

소벨필터

장점 : 중앙 행, 중앙 픽셀에 조금 더 무게를 실어 더 견고하게 만듦

| | | |
|---|---|---|
| 3 | 0 | -3 |
| 10 | 0 | -10 |
| 3 | 0 | -3 |

Scharr filter

샤르 필터

: 오직 수직 엣지 검출만을 위한 것

[합성곱 연산 단점]

1) 필터를 컨볼루션하여 이미지 크기가 축소됨

2) 이미지 엣지 근처에서 많은 정보가 버려짐
(모서리나 가장자리에 있는 픽셀은 하나에서만 사용되는 것처럼 터치됨(적게 사용),
반면 중간에 있는 픽셀을 선택하면 해당 픽셀과 겹치는 필터영역이 많이 있음)

⇨ 합성곱 연산을 최대로 적용 -> 이미지를 패딩 : 이미지 가장자리에 추가 경계가 있는 이미지를 0으로 p 크기만큼 패드
출력 : n + 2p – f + 1 (n : 첫 이미지 크기, p : 패딩 크기, f : 필터 크기)

모든 셀에 영향을 미치기에 이미지의 모서리나 엣지의 정보를 버리지 않음

<패딩의 양>

1) 밸리드 합성곱 : 기본적으로 패딩이 없음 (p=0)

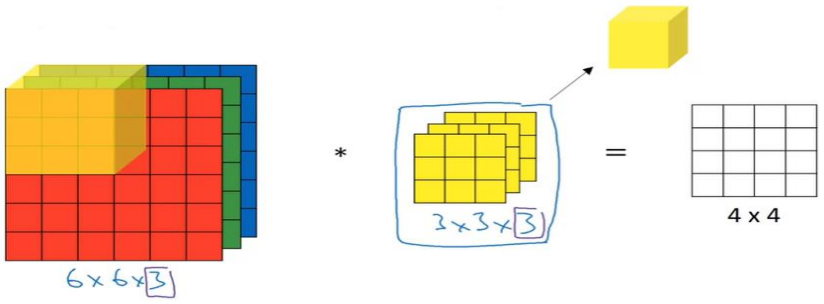
2) 세임 합성곱 : 패딩을 할 때, 입력크기 = 출력크기
n + 2p – f + 1 = n => p = (f – 1) / 2 (f는 주로 홀수)
* f가 짝수이면, 비대칭 패딩을 수행해야
* 홀수 크기의 필터 있을 때 중앙 위치를 가지며, 구별자를 갖는 것이 좋음

[스트라이드 컨볼루션]

출력값 : [(n + 2p – f) / s ] + 1 -> 정수가 아니면, 가장 가까운 정수로 반올림 (버림)

[RGB이미지 3차원]

6(height) X 6(width) X 3 (depth / #channels : RGB) -> filter 3 X 3 X 3 -> 출력 4 X 4 X 1
이미지의 채널 수 = 필터의 채널 수




R * 필터의 숫자
+ G * 필터의 숫자
+ B * 필터의 숫자
= 4 X 4 이미지 각 셀
(27번의 곱셈 후 전부 더해)

[여러 개 필터 사용]

2개의 필터(수직 엣지 검출기, 수평 엣지 검출기) -> 출력 4 X 4 X 2 크기

⇨ 입력 n X n X n_c -- 컨볼브 --> f X f X n_c -- 출력 - -> (n-f+1) X (n-f+1) X n_c′
(n_c′ : 사용한 필터의 개수)

[단일 합성곱 신경망] – 필터크기, 스트라이드, 패딩 또는 필터 개수 -> 하이퍼파라미터

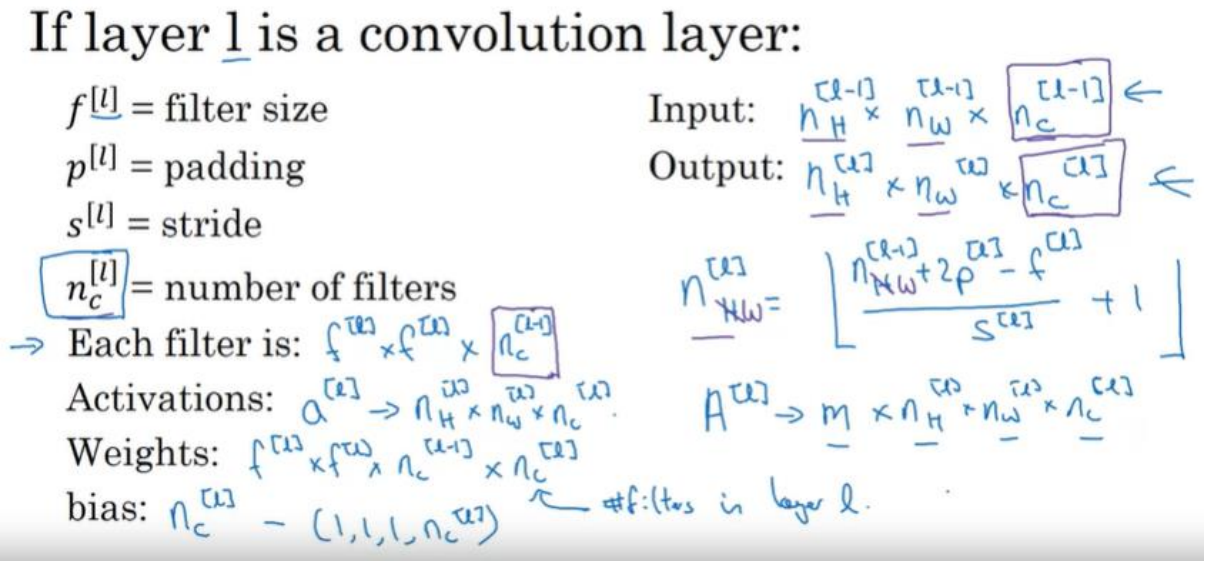
출력에 각각에 편향(실수)을 더하고, 활성화함수(비선형성) 적용

z[1] = w[1] * a[0] + b[1]  (a[0] = X)
a[1] = g(z[1])

- 입력 이미지 : a[0] (X) , 필터 : w[1] 과 유사한 역할

입력 이미지 6 X 6 X 3 / 필터 3 X 3 X 3 (2개) / 출력 4 X 4 X 2
합성곱 연산할 때 27 X 2에 해당하는 숫자 갖고, 모두 곱하고 4 X 4 행렬을 얻기 위해 일차 함수 계산 -> 이 4 X 4 행렬 : w[1] * a[0] 과 유사한 역할
4 X 4 X 2 : a[1] 과 유사한 역할

필터 3 X 3 X 3 크기의 10개의 필터로 신경망의 한 레이어가 가지는 매개변수
: ( 27 + 1(편향) ) * 10 = 280개의 매개 변수 (이미지 크기에 관계 없이 변수의 수는 280개로 고정)
-> 과대적합 방지



[심층 합성곱 신경망]

높이, 너비 -> 신경망 깊어질수록 비슷하게 유지되다 줄어듦
채널의 수 -> 신경망 깊어질수록 늘어남

[풀링 레이어] – Max Pooling / Average Pooling

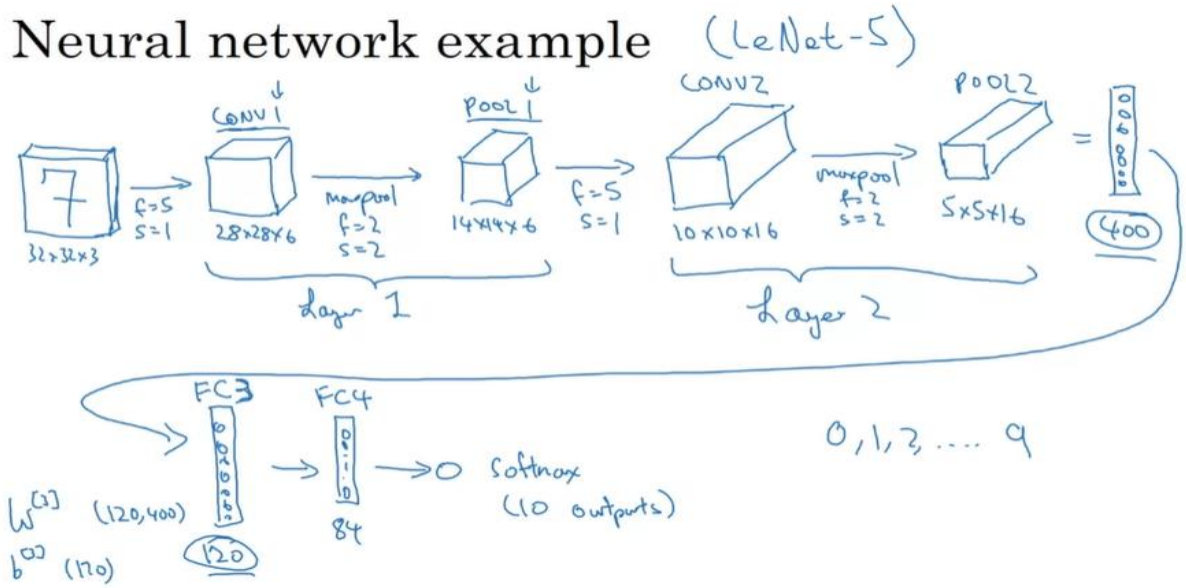
한 특성이 필터의 한 부분에서 검출되면 높은 수를 남기고, 특성이 검출되지 않아 특성이 오른쪽 상단 사분면에 존재하지 않으면 그 안의 최대값은 여전히 작은 수로 남음

여러 하이퍼파라미터 존재 (필터크기, 스트라이드 크기, max/average 선택)
학습할 수 있는 변수는 없다. -> 역전파가 가능한 변수가 없다.(가중치, 편향 없음)

입력 이미지 : 5 X 5 X 2  필터크기 3 X 3, 스트라이드=1  출력 이미지 : 3 X 3 X 2
입력 이미지 깊이(채널수) = 출력 이미지 깊이(채널수)

** 보통 Max Pooling f= 2 , s = 2  -> 입력이미지 10이면, 출력은 5 (/2)

Max Pooling 에선 패딩 거의 사용하지 않는다. (p=0)




Max Pooling 레이어의 매개 변수가 따로 없고, 합성곱 레이어가 상대적으로 적은 변수 가짐
신경망의 대부분의 변수는 완전 연결 레이어에 있는 경향이 있다.
활성값의 크기도 신경망이 깊어질수록 점점 감소한다. (너무 빠르게 감소하면 성능 좋지 않음)

## Neural network example

| | Activation shape | Activation Size | # parameters |
|---|---|---|---|
| Input: | (32,32,3) | 3,072 | 0 |
| CONV1 (f=5, s=1) | (28,28,6) | 4,704 | 456 |
| POOL1 | (14,14,6) | 1,176 | 0 |
| CONV2 (f=5, s=1) | (10,10,16) | 1,600 | 2,416 |
| POOL2 | (5,5,16) | 400 | 0 |
| FC3 | (120,1) | 120 | 48,120 |
| FC4 | (84,1) | 84 | 10,164 |
| Softmax | (10,1) | 10 | 850 |

파라미터 개수

= (f * f * n_C_prev + 1) * n_C

필터크기 + 1 * 현재이미지출력크기

456 = 5 * 5 * 3 * 6 + 6

2416 = 5 * 5 * 6 * 16 + 16

48120 = (5*5*16) * 120 + 120

10164 = 120 * 84 + 84

850 = 84 * 10 + 10

[컨볼루션 층의 장점] - 파라미터 공유와 연결의 희소성

conv네트워크가 작은 파라미터에 도달한 이유 (2)

- 파라미터 공유 : 수직 엣지 검출기와 같은 특성의 감지기를 관찰함으로써 동기부여 되고, 이미지의 한 부분에서 유용하다면 이미지의 다른 부분에서도 유용할 수 있다. => 각 출력은 동일한 파라미터 중 많은 부분에서 다른 위치에서 수직 모서리나 다른 특성을 감지하도록 같은 파라미터 사용 가능 => 파라미터 수를 줄이는 방법

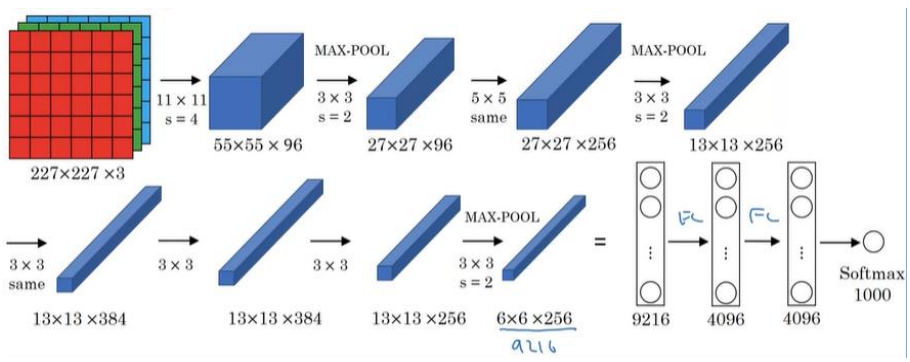
- 연결이 희박함 : 필터 입력격자판(셀)에만 의존, 필터 댔을 때 나머지 픽셀은 이 출력에 전혀 영향 주지 않음

[Classic Networks] – LeNet-5, AlexNet, VGG, ResNet

1) LeNet-5 (위의 그림) – 60000개 파라미터

신경망 깊어질수록 n_H, n_W 작아지고, n_C 커짐

2) AlexNet - 6천만개 파라미터




가치 활성화 기능(Relu)

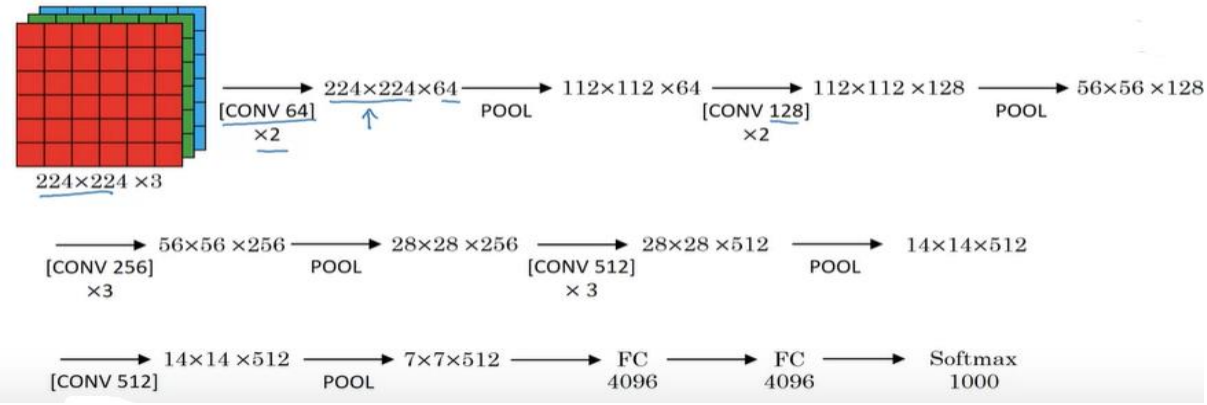
3) VGG-16 (16개의 레이어에 가중이 있음) – 1억 3천 8백만개의 파라미터

신경망 아키텍처 단순성 : 균일

높이와 너비가 감소함에 따라 Pooling layer가 매번 2배씩 감소, 채널의 수는 2배 증가

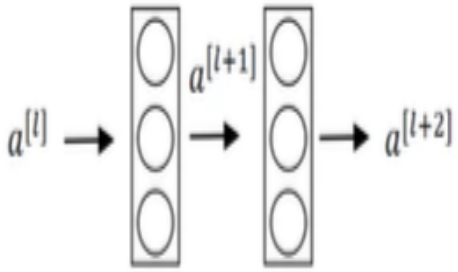
많은 하이퍼파라미터, 더 간단한 네트워크 사용해 conv-layers에만 집중

하나의 stride와 언제나 동일한 패딩의 3 X 3 필터인 컨볼레이어
+ stirde=2를 가진 2 X 2 MaxPooling layer




[ResNet]

깊은 신경망은 기울기 소실/폭발 문제들로 훈련시키기 어려움
한 계층에서 활성화를 가져와 신경망의 더 깊은 다른 계층으로 전달할 수 있는 건너뛰기 연결을 통해 ResNet 구축하여 심층 신경망 훈련




<main path>

a[l] -> linear -> ReLU – a[l+1] -->linear -> ReLU -> a[l+2]

<linear>

z[l+1] = W[l+1] * a[l] + b[l+1]

<ReLU>

a[l+1] = g(z[l+1]) / a[l+2] = g(z[l+2])

<short cut / skip connection>

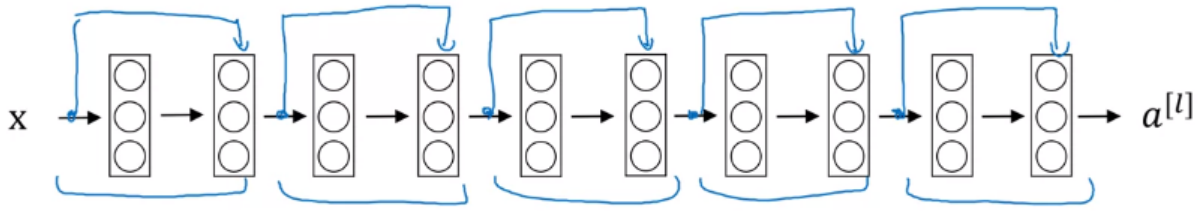
a[l]의 정보는 신경망으로 훨씬 깊숙이 들어가고, a[l] -> ReLU -> a[l+2]

a[l+2] = g(z[l+2] + a[l]) = g(w[l+2] * a[l+1] + b[l+2] + a[l])

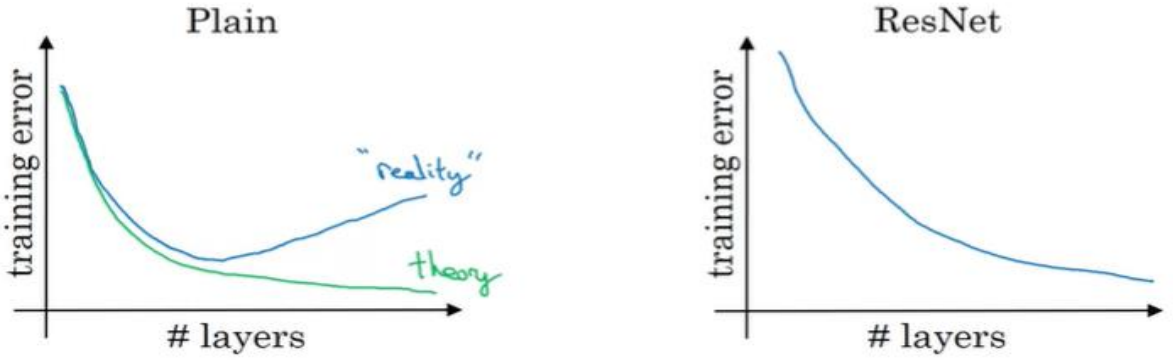
a[l]은 한 레이어를 건너뛰거나 거의 두 레이어를 건너 뛰어 신경망 깊숙이 정보를 처리한다.

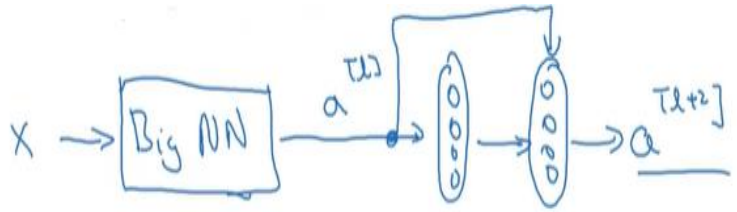
- Plain Network(평범한 망) -> ResNet

모든 건너뛰기 연결 추가, 각 두 레이어를 잔류 블록(residual block)으로 전환 =>5개의 잔류 블록




중간계 활성화는 신경망에 더 깊이 들어갈 수 있게 해주고, 이는 기울기 소실/폭발 문제 처리 + 성능 저하 없이 훨씬 더 깊은 신경망 훈련시킬 수 있다




a[l+2] = g(z[l+2] + a[l])
= g(w[l+2] * a[l+1] + b[l+2] + a[l])
-> L2 정규화 사용하면, w[l+2]값 축소, b[l+2]도 축소

IF w[l+2] = b[l+2] = 0, g(a[l]) = a[l+2] => 항등함수가 잔류 블록을 학습하기 쉽다. (a[l+2] = a[l])
두 레이어 추가해도 a[l]을 a[l+2]에 복사하는 것만 해도 항등 함수를 쉽게 배울 수 있어, 두 개의 추가 레이어를 더하고 잔류블럭을 큰 신경망의 중간이나 끝 어딘가에 더해도 수행능력은 저해되지 않음

z[l+2]와 a[l]의 차원이 같다고 가정할 때, ResNet에선 동일한 컨볼루션을 많이 사용하고 있기에, 파란점(추가한 레이어 전의)의 차원은 출력 레이어의 차원과 같다. => 같은 컨볼루션은 차원 유지
입출력이 다른 차원일 경우, 예로 a[l]은 128차원, z[l+2]는 128차원이므로 a[l+2]는 256차원

a[l+2] = g(z[l+2] + a[l]) = g(w[l+2] * a[l+1] + b[l+2] + W_s * a[l])
(W_s : 추가 매트릭스 256 X 128차원 / W_s * a[l] = 256차원)

ResNet은 동일한 차원의 특성 벡터를 더함 (동일한 컨볼루션) -> 차원 보존
풀링레이어 -> W_s로 차원 조정

[1 X 1 Convolutions]

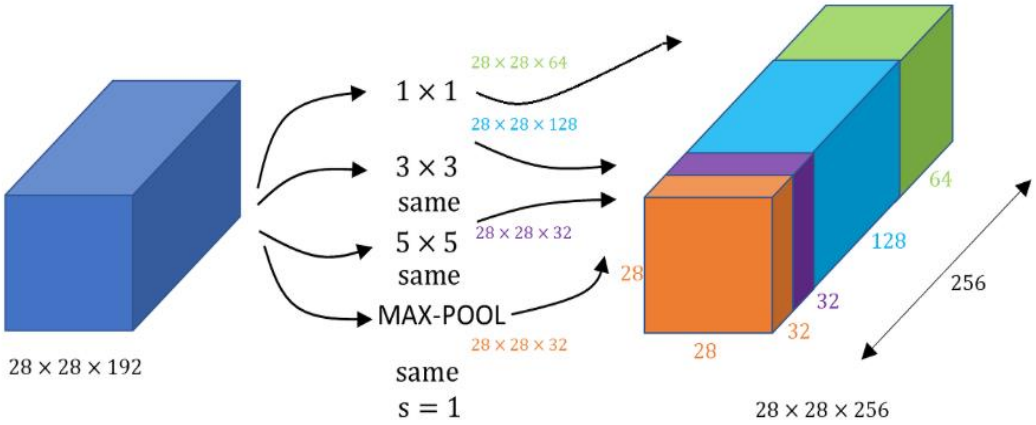
입력 이미지 : 6 X 6 X 1 -> 필터 : 1 X 1 X 1 => 필터의 숫자 값으로 곱하는 출력

Pooling : n_H, n_W 만을 줄이기만 함

1 X 1 Convolutions : 채널의 수(n_C) 조절 가능, 비선형성, 계산량 감소

[ML] ResNet & Inception Network란? (tistory.com)

[Inception Network] – n_H, n_W는 변하지 않고, n_C만 변함




인셉션모듈

3개의 Conv + 1개의 Pooling 모든 작업을 수행하고, 모든 출력을 연결할 수 있다.

- 5 X 5 필터의 계산비용 문제(28 X 28 X 192 -> 5 X 5 X 192 X 32 -> 28 X 28 X 32)
  (28 * 28 * 32) * (5 * 5 * 192) = 1억 2천만
- 28 X 28 X 192 -> 1 X 1 X 192 X 16 -> 28 X 28 X 16 -> 5 X 5 X 16 X 32 -> 28 X 28 X 32
  (28 * 28 * 16) * (1 * 1* 192) + (28 * 28 * 32) * (5 * 5 * 16) = 1240만

병목현상 레이어(1 X 1 Conv)를 생성해 계산 비용을 크게 줄임

[MobileNet] – Depthwise separable 컨볼루션을 신경망으로 사용해 MobileNet 구축

Depthwise separable 컨볼루션 = Depthwise 컨볼루션 + pointwise 컨볼루션

- Depthwise 컨볼루션

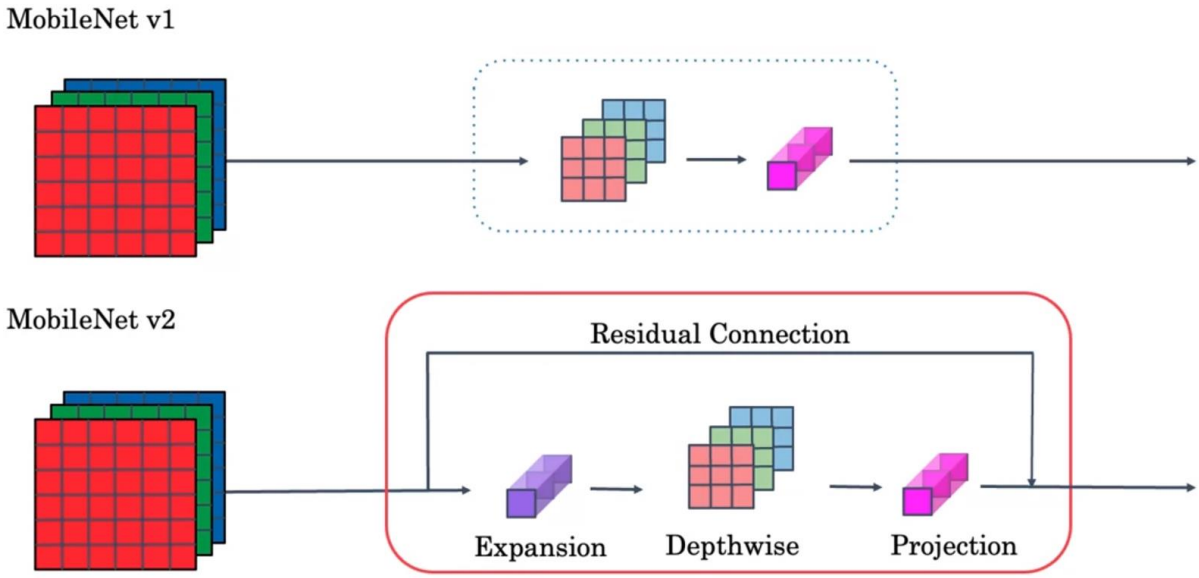
입력 이미지 n X n X n_C -> 필터 f X f -> 출력 이미지 n_out X n_out X n_C
계산비용 : (f * f) * (n_out * n_out * n_C)

- Pointwise 컨볼루션

입력 n_out X n_out X n_C -> 필터 1 X 1 X n_C X n_C′ -> 최종 출력 n_out X n_out X n_C′
계산비용 : (1 * 1 * n_C) * (n_out * n_out * n_C′)

<같은 입력 이미지를 같은 출력의 이미지 크기로 할 때의 계산비용 비교>

(Depthwise 계산비용 + Pointwise) / 일반 컨볼루션 계산비용 = (1/n_C′) + (1/f^2) 만큼 절약




MobileNet v1 : 블록을 13번 반복

MobileNet v2 : 잔류 연결의 추가(이전 레이어에서 입력 가져와 합산하거나, 다음 레이어로 직접 전달해 구성요소가 더 효율적으로 뒤로 전파되도록) + 확장 레이어(n_C(차원)증가) – 17번 작업
* Projection : 확장한 것을 다시 하향 투영
=> 병목 현상 블록

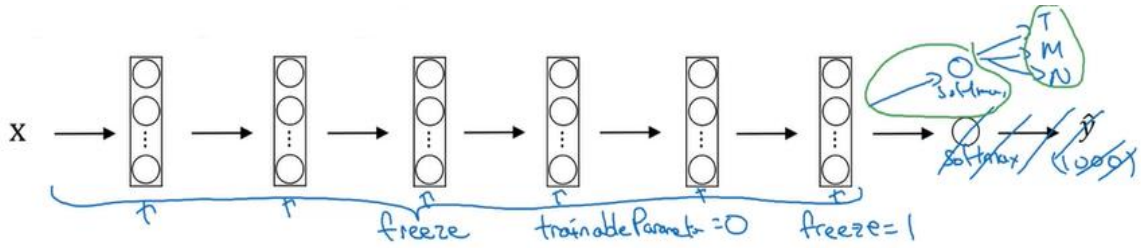
[EfficientNet] : 특정 장치의 신경망을 자동으로 확장하거나 축소하는 방법

입력 이미지 해상도 r , 새로운 네트워크 깊이 일정, 일정한 폭의 레이어

1) 고해상도 이미지 사용(Higher Resolution) : 새로운 이미지 해상도 r
2) 네트워크 신경망 깊게(Deeper)
3) 레이어 너비 넓힘(Wider)

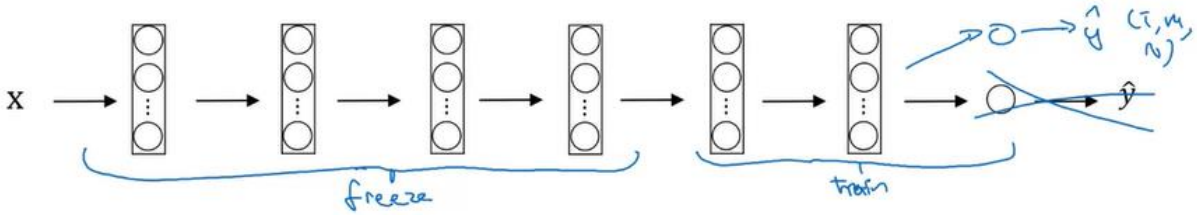
[전이 학습]

- 작은 훈련셋 - Softmax layer만 훈련하고, 이전 layer 방법은 고정한다.




훈련가능한 파라미터 0인 경우가 있고, 때로는 고정이 1인 파라미터를 가지게 된다.

- 큰 훈련셋 – 더 적은 레이어 고정, 나중의 레이어만 훈련한다.




마지막 몇 개의 레이어를 사용해 초기화로 사용하고, 그곳에서 기울기 하강 수행
OR 마지막 몇 개의 레이어 날리고, 새로운 은닉 유닛과 최종 softmax 출력으로 사용

데이터셋 커질수록 고정된 레이어의 수 줄고, 훈련하는 레이어 수 늘어날 수 있다.
데이터셋을 선택해 하나의 softmax 유닛을 훈련시키기 위한 데이터뿐만 아니라 최종 네트워크의 마지막 몇 레이어를 구성하는 다른 크기의 신경망을 훈련시키기 위한 데이터도 가지고 있을 수 있다.

- 많은 양의 데이터 – 네트워크 전체를 초기화하고 훈련하는 것과 같다.
랜덤초기화를 수행한 후, 기울기 하강, 네트워크의 모든 방법과 레이어 업데이트하는 훈련 =>"컨볼루션 네트워크의 훈련을 위한 전이 학습"

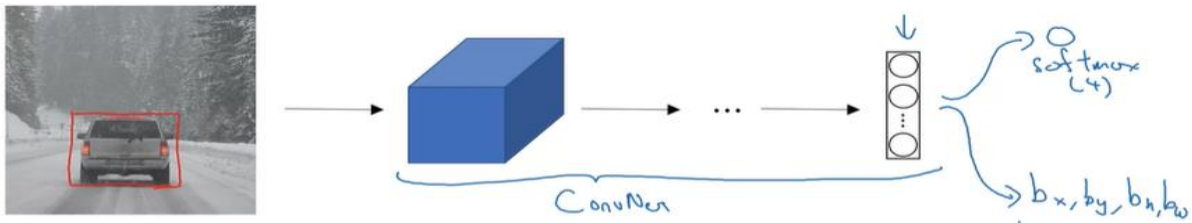
[데이터 증강]

유형1)
수직축에 미러링(수평으로 뒤집고, 이미지에서 인식하려는 내용이 모두 보존되면 좋다.)
+ 무작위 크롭
=> 이미지에 적용하면 왜곡시키고, 다양한 형태의 지엽적인 뒤틀림을 적용할 수 있다.

유형2) 색상이동

[Object Localization]

이미지 속에서 감지하고 있는 객체가 어디에 있는지 알아내는 것 -> 하나의 객체
(** Detection : 여러 개 객체 존재)




Softmax : 클래스 4개 / bx, by, bh, bw : 감지된 객체의 경계상자 출력하는 출력 장치(파라미터화)

<타겟라벨 y> : 8개의 구성 요소가짐 (8차원 벡터)

1 - pedestrian
2 - car
3 - motorcycle
4 - background

<- 분류 라벨

객체가 클래스 1,2,3이 되면 Pc = 1

만약 배경 분류라면, 감지하고자 하는 객체가 아무것도 없다면, Pc = 0

y = [Pc, bx, by, bh, bw, c1, c2, c3]  # Pc = 1일 때, 클래스 c1, c2, c3을 출력값으로 입력해야 함
y = [0, ?, ?, ?, ?, ?, ?, ?]  # Pc = 0 => 손실함수에서 2~8번째 성분은 고려하지 않음

[Object Detection]

** 랜드마크 detection : (bx, by) = (l1x, l1y)

<Sliding windows detection>

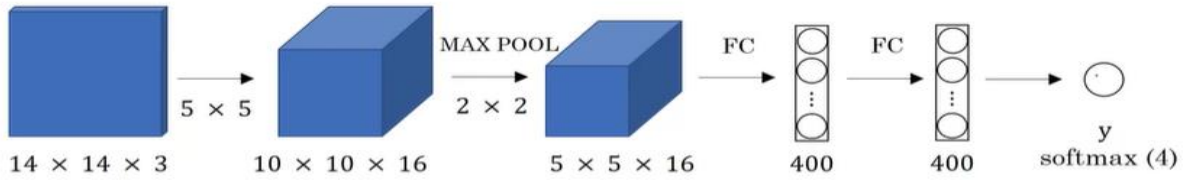
특정 크기의 윈도우 선택 -> ConvNet(테스트 이미지)에 윈도우 입력 -> 객체 있는지 추측(1/0)
-> 이미지 내의 모든 위치에서 윈도우 슬라이드할 때까지 계속 진행(스트라이드 사용)

컴퓨팅 비용이라는 단점 : 이미지에 있는 많은 사각 영역들을 잘라내고, ConvNet을 통해 각각 독립적으로 실행할 수 있다.
(큰 스트라이드/사이즈 사용(거친 입상도) -> ConvNet 통과하는데 필요한 윈도우 수 줄일 수 있다.
작은 스트라이드/사이즈 사용(고운) -> 정확하지만, ConvNet 통과하는 계산 비용이 매우 높다.)

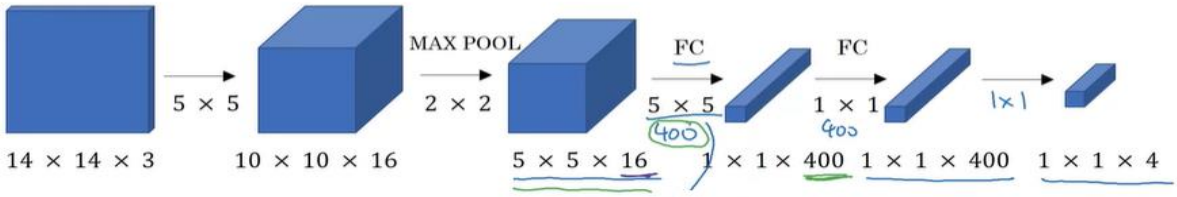
** Window 크기마다 ConvNet 실행하며, Window가 이미지 끝에 도달할 때까지 반복해야 하므로 매우 느림

<Convolutional Implementation of Sliding Windows : FC layer -> Conv layer> ; 계산비용 문제해결

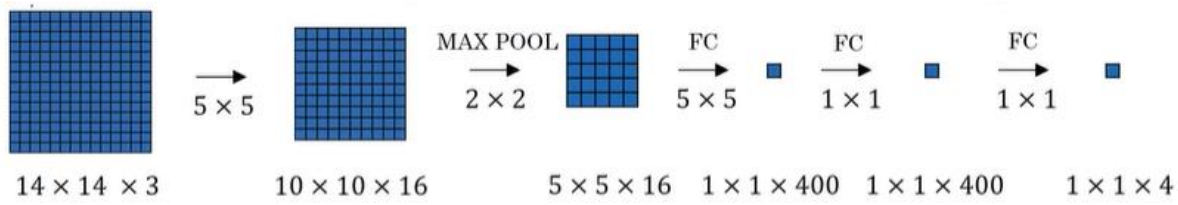
각각의 Window에 ConvNet을 실행할 필요 없이, 전체 Windeow에 한 번의 ConvNet만 실행가능



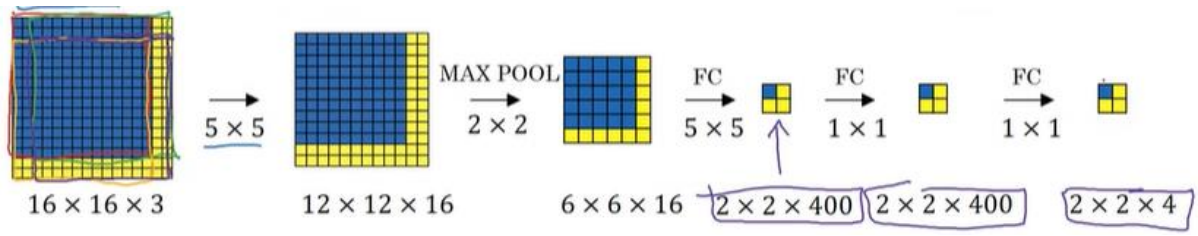

▲일반 convolution ▼FC -> Conv로 전환




5 X 5 필터 400개 사용 => 400개의 노드 세트가 아닌 1 X 1 X 400 볼륨으로 간주 (400개 노드 각각은 5 X 5 X 16 필터 가짐, 이전 레이어의 5 X 5 X 16 액티베이션에 대한 임의의 선형 함수)




슬라이딩 윈도우가 14 X 14 이미지를 3개의 이미지로 변환 // "볼륨"




테스트이미지 크기 16 X 16 X 3

4개의 레이블을 얻으려면 이 ConvNet를 네 번 실행한다. 하지만, 이 4가지 ConvNet에 의해 수행된 많은 계산은 매우 중복적이므로 슬라이딩 윈도우의 컨볼루션 실행은 ConvNet에서 네 클래스들이 많은 계산을 공유할 수 있게 한다.

출력 2 X 2 X 4 볼륨의 오른쪽상단 1 X 1 X 4 볼륨은 입력 이미지의 녹색 이미지로 실행한 결과 (전체 이미지에 대해 4개의 라벨을 얻은 것과 동일)

=> 독립적으로 입력 이미지의 네 부분 집합상에 있는 네 개의 전파를 실행하도록 강요하기 보다는 네 개 모두를 하나의 계산형식으로 결합하고, 공통 이미지 영역에 있는 많은 계산을 공유해 빠름

입력 이미지에 대해 스트라이드 2로 신경망을 실행하는 것 = 맥스 풀링 2

** 각 Bounding Box가 객체 정확하게 표현하지 못한다는 단점 존재

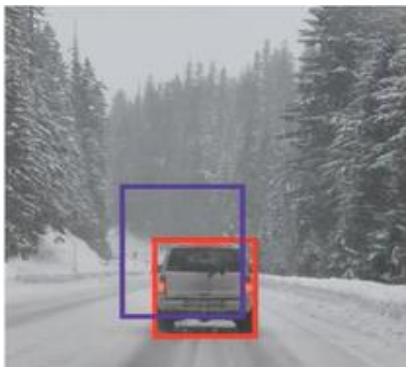
[Bounding box predictions]

경계상자가 객체 위치와 완벽하게 일치하지 않을 수 있다.
경계상자가 완전히 정사각형이 아니기 때문에 실제로는 약간 더 넓은 직사각형처럼 보일 수 있다.
=> 이러한 경계상자의 단점 보완 -> YOLO 알고리즘 (You Only Look Once)

입력이미지 100 X 100 -> 3 X 3 격자판 -> 출력의 전체 볼륨 3 X 3 X 8
훈련을 위한 라벨 : 9개의 격자판 셀 각각에 대해 레이블 Y 지정(Y : 8차원 벡터 – 위의 예)
객체의 도달 중간 점을 취한 다음, 중간 점을 포함하는 격자판 셀에 객체를 할당(0<= bx, by <=1)

YOLO의 장점 : 정확한 경계상자 출력, 컨볼루션 구현이어서 매우 빠르게 실행/실시간 가능
실제로는 19 X 19 크기 같은 미세한 격자판 사용 -> 동일한 격자판(하나의 격자판 셀) 안에 여러 객체가 할당 될 가능성 줄여줌

[객체 감지 알고리즘 평가 Intersection Over Union(IOU)] ; 합집합에 대한 교집합



두 개의 경계 상자의 합집합에 대한 교집합 계산
= 교집합 크기 / 합집합 크기

IOU >= 0.5 : Correct

: 현지화를 매핑하는 방법
; 알고리즘이 객체를 올바르게 감지하고 현지화하는 횟수를 세어보면 객체가 올바르게 현지화되었는지 여부 정의할 수 있다.

[Non-max Suppression 억제] – 각 객체를 독립적으로 억제

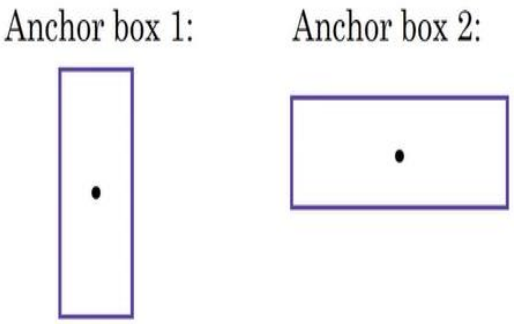
이전 : 동일한 객체에 대해 다수의 감지를 찾음, 객체를 한 번만 탐지하는 게 아닌 여러 번 탐지
(훈련 시에는 객체의 중심점이 있는 그리드 셀만 $P_c=1$로 설정해 객체를 인식하는 것으로 다뤘지만, 테스트 시에는 각 그리드 셀마다 객체의 중심점을 가지고 있다고 인식할 수 있음)
-> non-max 억제 : 각 객체를 한 번만 탐지하는지 확인하는 방법

: 각각의 객체의 감지와 관련된 확률($P_c$가 감지될) 확인 : 높은 것(IOU)만 남기고 나머지는 억제

[Anchor boxes] ; 하나의 격자판 셀이 여러 객체 탐지

보행자와 차량의 중간 점(bx, by)이 거의 같은 위치, 둘 다 동일한 격자판 셀에 속함

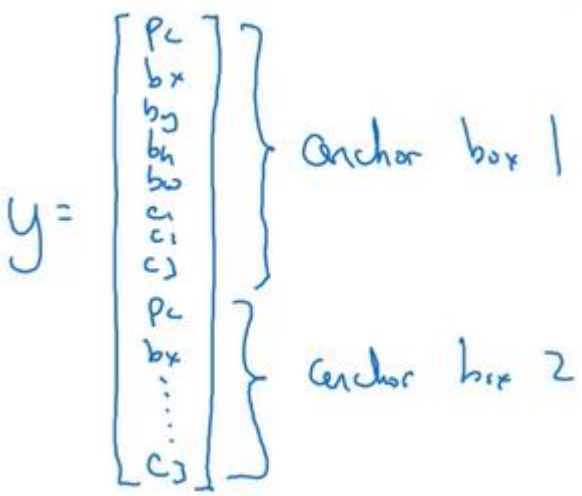
격자판 셀의 경우, Y가 3가지 원인(보행자, 자동차, 오토바이)을 감지할 때, 벡터를 출력하면 두 개 감지를 출력할 수 없다. -> 출력할 감지 중 하나 선택해야함 (벡터를 사용한다면, 1개의 경계 상자 좌표만 할당할 수 있기에 1개의 객체만 탐지 가능)




앵커박스 : 탐지하려는 객체의 모양 미리 정의해놓고, 객체가 탐지되었을 때 어떤 앵커박스와 유사한지 비교해 벡터 값 할당(IOU가 가장 높은 격자판 셀 가진) (5개 이상도 되는데, 일단 여기선 2개)

y는 두 번 반복(앵커박스 개수만큼)

보행자는 앵커박스1과 더 유사하고,
자동차는 앵커박스2와 더 유사하므로,
y의 위는 보행자임을 인코딩,
아래는 자동차임을 인코딩




출력 y : 3 X 3 X 16 (3 X 3 X 2 X 8)

IF 동일한 격자판 셀에 앵커박스 모양 같거나 / 앵커박스 두 개, 객체 세 개인 경우
=> 이 알고리즘 문제 => 디폴트 타이브레이커

[R-CNN] ; 느림

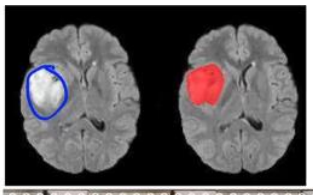
Window sliding하는데, 알고리즘이 양방향으로 작동시킬 수 있지만, 아무런 객체가 없는 많은 지역들을 크로스파이어 한다는 단점이 있다. -> R-CNN : 지역들을 가리킴

Segmentaion 알고리즘 : 약 2000개의 block 발견하고, 그 block 주위에 경계 상자 배치하고, 이에 대해 크로스파이어 함

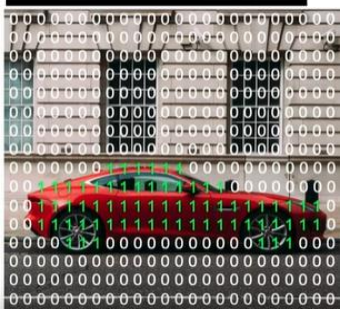
[Semantic Segmentation with U-Net]

: 감지된 객체 주위에 면밀한 윤곽선 그려 개체에 속하는 픽셀과 않은 픽셀을 정확하게 파악

경계 상자를 그리려고 하는 것 보다, 모든 픽셀에 라벨을 붙임
(ex. 자율주행차 -> 어느 도로가 주행하기에 안전한지 정확히 파악)

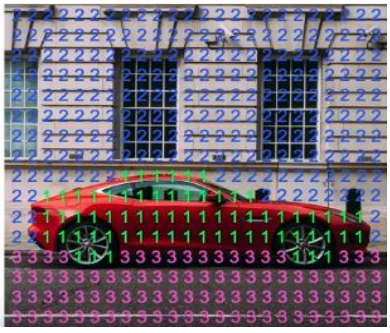


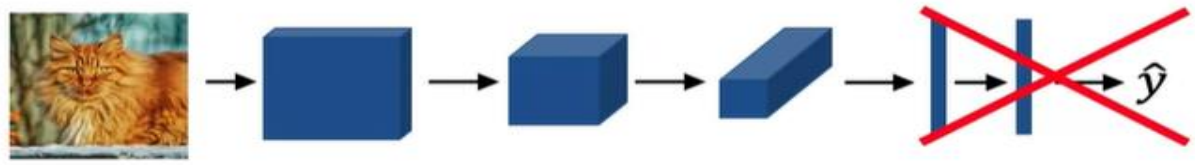
ex. 뇌 MRI 스캔은 뇌종양 검출에 사용되는데, 자동으로 종양을 분할하는 단위 알고리즘 사용(U – Net)

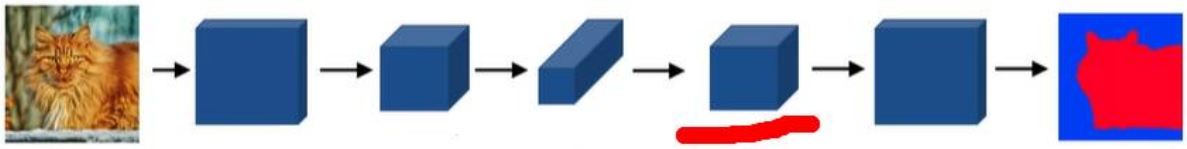


U-Net :
모든 단일 픽셀에
대해 1 or 0으로 출력

Segmentation )
차량 : 1 / 건물 : 2 / 도로 : 3

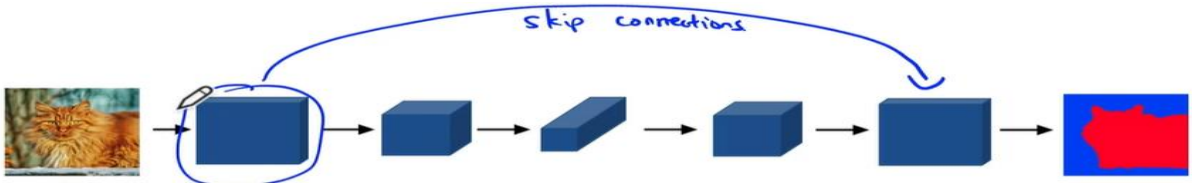




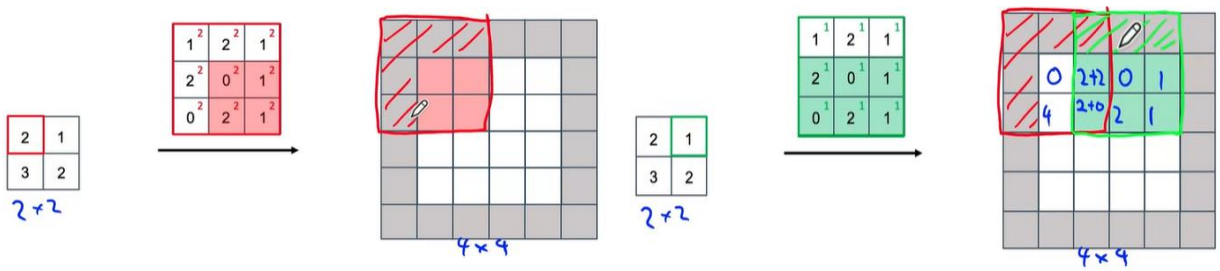
Semantic Segmentation (빨간줄 ;단위 아키텍처 ; 높이와 부피는 다시 높아지고, 채널 수는 줄음)
- 일반 convolution : 몇 레이어를 지우고, 이미지 차원은 왼쪽에서 오른쪽으로 갈수록 작아졌다.
- 전치 Convolution : 점차 출력에 필요한 전체 크기로 확대하기 위해 더 커져야 한다

<더 효율적인 동작>




초기 레이어에서 이후 레이어로의 연결을 건너뜀 ; 이전 활성화 블록은 이후 블록에 직접 복사됨
단위 아키텍처(이후 블록의 전)는 높은 수준의 컨텍스트 정보 갖지만, 미세한 공간적 정보는 누락
(공간 해상도 낮기에) -> 건너뜀으로써 픽셀 위치마다 많은 좋은 물질이 이 픽셀 안에 들어있는지

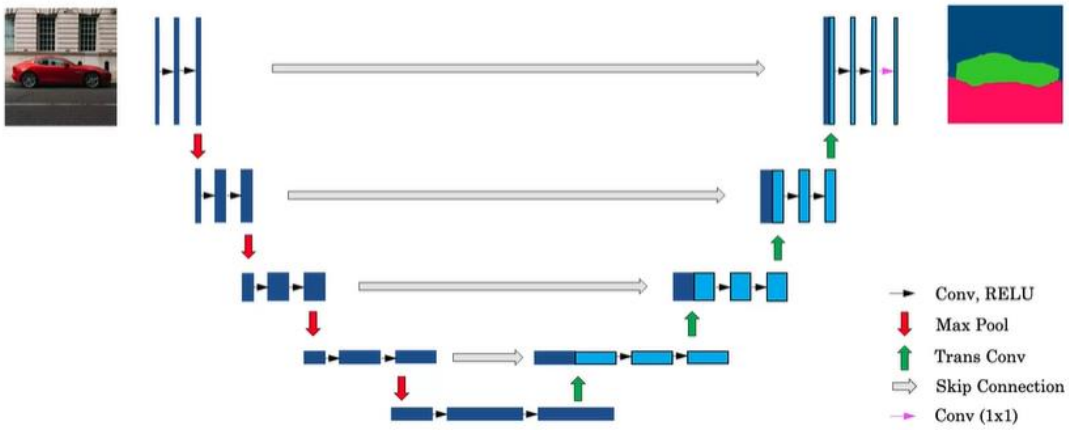
[Transpose Convolutions] ; 작은 입력을 큰 출력으로 바꿈



2 X 2 입력 -> 활성화되면 3 X 3 필터로 변환 -> 4 X 4 출력으로 확대 (p = 1, s = 2)
출력에 패딩, 필터 배치

패딩 안의 부분은 무시
빨간색과 초록색 상자가 겹치는 경우, 두 값을 더함

[U-Net 아키텍처]




전치 convolution에서 어두운 파란색은 왼쪽(일반 컨볼루션)에서 복사된 것,
밝은 파란색은 전치 컨볼루션에서 나온 채널(줄임)

한 개의 컨볼루션(Conv (1 X 1))을 사용해 Segmentation 맵에 매핑하여 결과 출력

입력 : h X w X 3 (channel 수) → 출력 : h X w X n_classes

[One-shot Learning]

한 가지 이미지를 통해 학습해서 그 사람을 다시 인식

각 class에 따른 하나의 training 이미지만으로, 이미지를 인식하고 분류하는 모델 만듦(얼굴 인식)
하지만, 각 class에 따른 training 이미지 수 적다면, 이미지 인식하고 분류하는 모델 만드는 것 어려움 -> augmentation 같은 방법으로 샘플 수 늘려 해결할 수 있지만, 새로운 이미지 추가되면 새로 학습해야 한다는 문제 => 실시간에 적합하지 않음

➔ 이미지 간의 유사도

신경망 d를 나타내는 함수

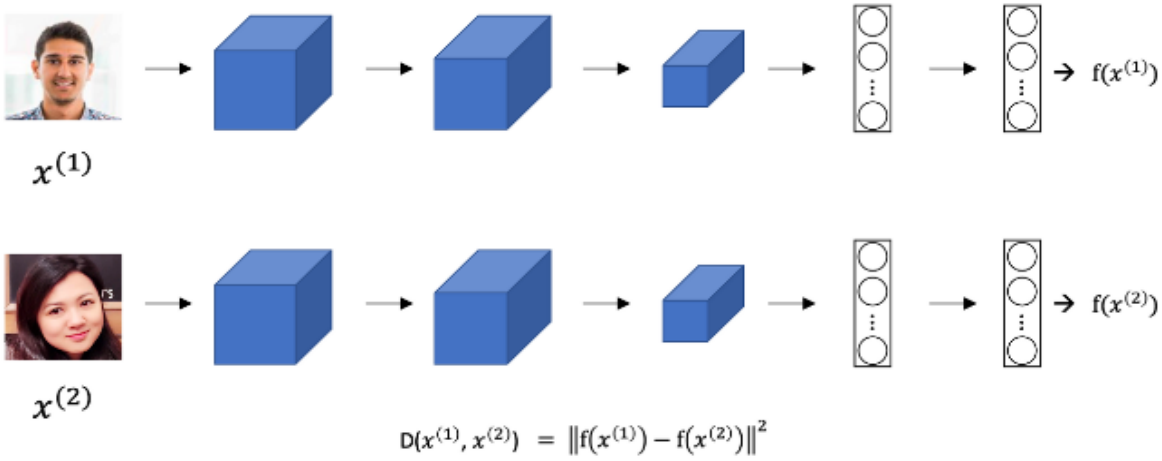
$$d(img1,img2) = \text{degree of difference between images}$$

$$\text{If } d(img1,img2) \leq \tau \quad \text{"same"}$$

$$> \tau \quad \text{"different"}$$

두 이미지가 같은 사람인 경우, 작은 값 출력
매우 다른 두 사람 이미지이면 큰 값 출력

[Siamese Network] ; 두 개의 서로 다른 입력으로 두 개의 동일한 컨볼루션 신경망을 실행하고, 그것들을 비교하는 것




F(x(1)) : x(1)의 입력 이미지를 인코딩하는 것
두 번째 그림을 같은 신경망에 같은 파라미터로 전송해 다른 벡터를 얻고, 두 번째 그림 암호화

신경망의 모든 층에서 파라미터 변경하면 다른 인코딩으로 출력하게 된다. 역전파를 사용하고 모든 파라미터를 변화시킨다.

[Triplet Loss 삼중항 손실함수]

이미지 쌍 비교

앵커의 이미지를 보고, 이미지와 긍정적인 이미지(같은 사람)는 멀리 하려고 한다.
-> 같은 사람이지만, 비슷하다는 뜻
앵커의 이미지를 보고, 거리가 멀리 떨어져 있는 부정적인 이미지(다른 사람)는 앵커를 사용한다.
=> 동시에 앵커(A) / 긍정적(P) / 부정적(N) 이미지 3가지를 봄
- A : 입력 이미지 인코딩 벡터
- P : 입력 이미지와 동일한 사람의 인코딩 벡터
- N : 입력 이미지와 다른 사람의 인코딩 벡터

d(A, P) <= d(A, N)
||f(A) – f(P)||**2 <= ||f(A) – f(N)||**2

모든 이미지에 대한 인코딩이 다른 모든 이미지에 대한 인코딩과 동일하면, 0벡터가 된다.

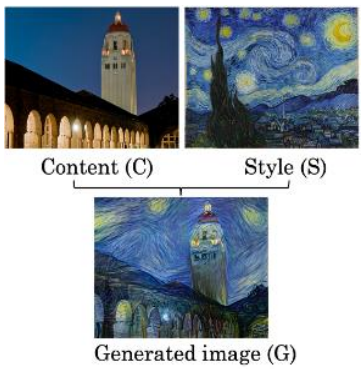
||f(A) – f(P)||**2 - ||f(A) – f(N)||**2 + alpha <= 0
(alpha = margin ; P와 N 멀어질수 있도록 밀어냄)

손실함수 L(A, P, N) = max(||f(A) – f(P)||**2 - ||f(A) – f(N)||**2 + alpha, 0)

단점 )

A, P, N에 사용되는 이미지를 random하게 골랐을 때, Loss가 쉽게 0이 된다. -> 훈련 잘 안됨
=> 구분하기 어려운 샘플 조합 고르는 것이 중요 !
=> 무작위로 이미지 고르는 것이 아닌, 세 개의 이미지가 비슷한, d(A, P)와 d(A, N)의 차이가 크지 않은 이미지 우선적으로 사용하는 것이 좋다.

[Neural Style Transfer] ; Content Image 형태는 유지하면서, Style과 유사한 새로운 Image G




J(G) = alpha * J_content(C, G) + beta * J_style(S, G)
콘텐츠 이미지 C의 콘텐츠와 생성된 이미지 G의 콘텐츠가 얼마나 비슷한지 측정 + 스타일이 얼마나 비슷한지 측정

G를 임의로 초기화하여 원하는 차원으로 만들기 (백색소음 이미지)
-> J(G) 최소화하는데, G의 픽셀값 업데이트하며 이미지 G 얻음

<Content Cost Function>

Hidden layer l 을 사용해 콘텐츠 비용 계산

IF l이 매우 작은 숫자인 경우, G와 C가 매우 비슷한 픽셀 값 가지도록 만듦

IF l이 매우 크면(심층 레이어 사용), C에 있는 객체를 G의 어딘가에 객체가 있는지 확인

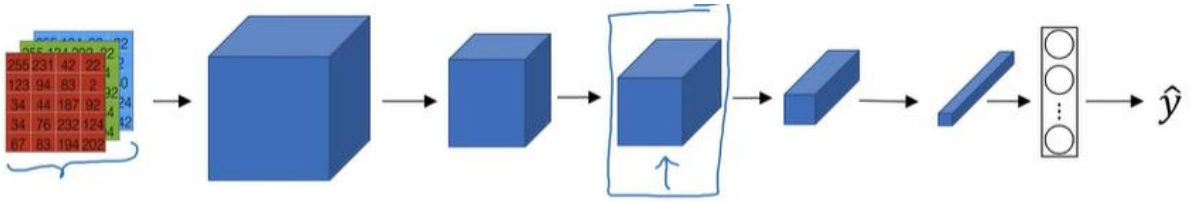
사전 훈련된(pre-trained) ConvNet 사용

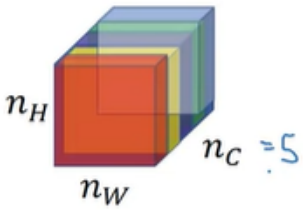
a[l](C)와 a[l](G)가 유사하다면, 같은 내용을 갖고 있는 것을 의미

J_content(C, G) = 1/lambda * ||a[l](C) – a[l](G)|| **2

**a[l](C) : 이미지 C의 레이어 l의 활성화

<Style Cost Function>






첫 번째 채널(빨간색)과두 번째 채널(노란색)의 두 활성의 상관관계

= 생성된 이미지에서 비슷한 정도 측정

## Style matrix

Let $a^{[l]}_{i,j,k}$ = activation at $(i,j,k)$. $G^{[l]}$ is $n_c^{[l]} \times n_c^{[l]}$

$$J^{[l]}_{style}(S,G) = \frac{1}{(2n_H^{[l]} n_W^{[l]} n_C^{[l]})^2} \sum_k \sum_{k'} \left( G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'} \right)^2$$