**CNN(Convolutional Neural Network) – "GoogLeNet (part1)"**

2014년 ILSVRC는 구글의 GoogLeNet이 차지하였고,

아주 근소한 차이로 옥스퍼드 대학교의 VGGNet이 2위를 차지한다.

그런데 여기서 주목해야 할 것은 2014년부터 CNN의 구조에 큰 변화가 나타나기 시작한다는 점이다.

AlexNet이나 ZFNet 그리고 원조격인 LeNet5는 2014년 구조에 비하면 아주 단순한 편이며,

전형적인 형태를 취하고 있고, 망의 깊이도 10 layer 미만이다.

2014년 변화의 특징은 한마디로 "**deeper**"라고 표현을 할 수 있다.

GoogLeNet이나 VGGNet은

2012년 Krizhevsky의 AlexNet에서 촉발된 에너지를 바탕으로 새로운 변화를 모색하게 되었으며,

CNN을 통한 학습 능력이 훨씬 더 커지게 되었음을 입증하였다.

이번 Class부터 GoogLeNet의 구조에 대하여 설명할 예정이다.

이해를 돕기 위해 여러 part로 구성이 될 예정이며,

VGGNet은 GoogLeNet에 대한 Class를 마치고 따로 자세히 설명을 할 예정이기는 하지만,

중간 중간에 필요에 따라 VGGNet에 대한 설명도 곁들일 예정이다.

**망은 깊어진다 (deeper and deeper) !!**

CNN의 성능을 향상시키는 가장 직접적인 방식은 망의 크기를 늘리는 것이다.

여기서 망의 크기를 늘린다는 것은 단순하게 망의 layer 수(depth)를 늘리는 것뿐만 아니라,

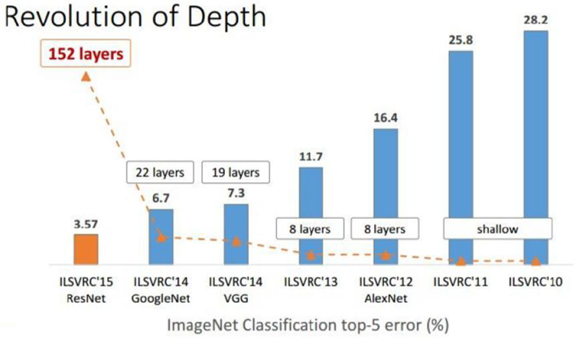
각 layer에 있는 unit의 수(width)도 늘리는 것을 의미한다.

특히 ImageNet 데이터와 같이 대용량 데이터를 이용해 학습을 하는 경우는 거의 필수적이라고 할 수 있다.

ㅎ

아래 그림은 image classification의 성능 향상을 위해

CNN의 구조가 어떻게 바뀌고 있는지를 명쾌하게 보여주는 그림이다.



2013년까지는 CNN 망의 깊이가 10 layer 미만이었지만,

2014년의 대표주자인 GoogLeNet과 VGGNet은 각각 22 layer와 19 layer로 깊어지게(deeper) 된다.

물론 top-5 에러율도 각각 6.7%와 7.3%로 낮아지게 된다.

AlexNet의 결과가 나온 뒤 불과 2년 만에 에러율을 약 10% 정도 낮추는 쾌거를 이루게 된다.

2015년 우승을 한 ResNet은 망의 깊이가 152 layer 로 더욱 깊어지게 되며,

top-5 에러율도 3.57%로 더욱 낮아지게 된다.

**망이 깊어지면, 부작용(side effect)는 없나?**

망의 크기를 늘리면 성능을 더 높일 수 있지만,

적절하지 못하면 다음 2가지 중대한 문제를 만날 수도 있다.

우선 망이 커지면 커질수록 자유 파라미터(free parameter)의 수가 증가하게 되며,

이렇게 되면 특히 학습에 사용할 데이터 양이 제한적인 경우에 더 심각한 문제가 되지만,

망이 overfitting에 빠질 가능성이 높아진다.

(즉, 학습 데이터에만 특화된 결과가 만들어져, 실제 테스트 set에 적용하면 만족할 만한 결과가 나오지 못할 수 있다.)

그리고 대량의 데이터에 사람이 일일이 label을 달아주는 것도 쉬운 일이 아니다.

또 다른 문제는 망의 크기가 커지면 그만큼 연산량이 늘어나게 된다.

예를 들어 필터의 개수가 증가하게 되면, 연산량은 제곱으로 늘어나게 된다.

연산 능력이 뛰어난 GPU를 사용하더라도 연산량의 증가는 심각한 문제가 된다.

그리고, ZFNet을 학습하면서 살펴보았던 것처럼,

학습이 잘못되어 filter의 kernel이 특정한 무리로 쏠리게 된다면,

기껏 망의 크기를 늘렸음에도 불구하고, 최적의 결과를 얻지 못할 수도 있다.

GoogLeNet보다 layer 수가 작은 AlexNet 경우를 살펴보자.

[[Part Ⅴ. Best CNN Architecture] 3. AlexNet [1]](http://laonple.blog.me/220654387455) 에서 살펴본 것처럼, AlexNet은 엄청난 연산을 필요로 한다.

AlexNet은 자유 파라미터의 개수가 6000만개이고 약 6억 3000만개의 connection으로 이루어져 있으며,

엔비디아의 GTX580 dual-GPU를 이용하여도 학습 시간이 일주일 넘게 소요되었다.

단순하게 망을 깊게 만든다면, 자유 파라미터의 개수가 더욱 많아질 것이고,

connection도 엄청나게 많아지면서, 학습에 필요한 시간도 더욱 길어지게 된다.

또한 parameter 값이 정해지더라도 실제 연산을 할 때의 연산량 역시 무시할 수 없게 되고 말 것이다.

그리고 모바일이나 embedded 시스템에서 CNN을 활용하고자 한다면,

연산 능력이나 메모리 사용 등에서 PC 를 사용할 때보다 훨씬 제한될 수 밖에 없기 때문에,

단순히 망을 깊게 만든 것이 아니라, 뭔가 구조적인 고민이 필요하다.

**GoogLeNet과 Inception**

크리스토퍼 놀란 감독의 영화 인셉션(Inception)을 보면,

남에게 어떤 생각(꿈)을 주입하거나, 남의 생각을 읽어내는 내용이 나온다.

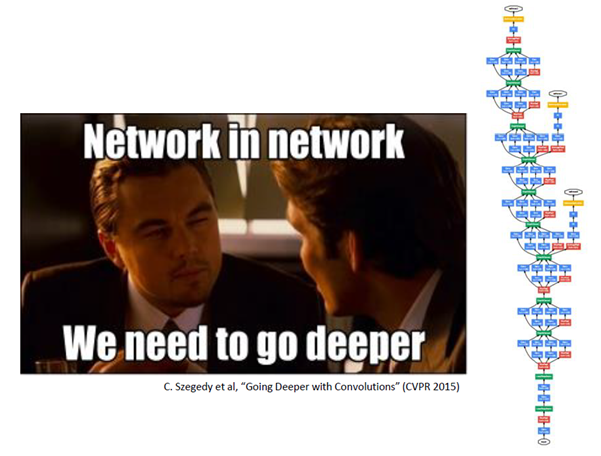
구글의 연구팀들은 그 영화에서 컨셉을 따와 인셉션이라는 이름을 갖는 CNN모듈을 만들어 낸다.

구글의 소개 자료를 보면 항상 다음과 같은 그림이 등장한다.

이는 더 깊은 CNN 구조를 사용하면 더욱 좋은 성능을 얻을 수 있다는 것 때문인 것 같고,

또한 인셉션의 내용이 남의 생각을 읽어내듯,

DNN을 이용한 데이터로부터 중요한 정보를 얻어내는 것을 연상하여 지은 이름 같다.



구글이 발표한 인셉션의 기본 구조는 아래와 같으며, 대충 봐도 무지 복잡해 보인다.

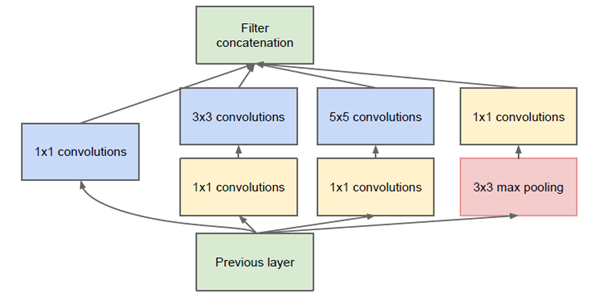
같은 layer에 서로 다른 크기를 갖는 convolutional filter를 적용하여 다른 scale의 feature를 얻을 수 있도록 했다.

그림에서처럼, 1x1 convolution을 적절히 사용하여 차원을 줄이고(reduce dimension) 망이 깊어졌을 때

연산량이 늘어나는 문제를 해결하였다.

이 부분은 다음 class에서 자세하게 설명할 예정이다.

(비록 복잡해 보이지만, 그 개념과 원리를 이해하면 충분히 공감이 갈 구조이다.)



GoogLeNet은 구글의 연구팀들이 인셉션 모듈을 고안한 뒤에 2014년 ILSVRC에 참가하기 위한 버전으로 내놓은 것이며,

인셉션의 구조는 다양한 형식으로 적용이 가능하다.

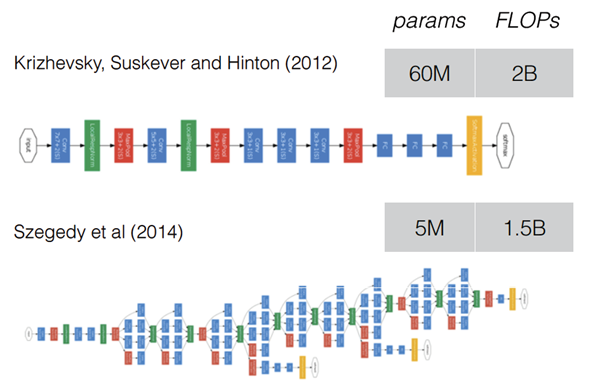
AlexNet과 GoogLeNet을 비교한 그림은 아래와 같다.

놀라운 부분은 망의 깊이는 훨씬 깊은데 free parameter의 수는 1/12 수준이고 전체 연산량의 숫자도 AlexNet에 비해 적다는 것을 알 수가 있다.

GoogLeNet과 인셉션에 대한 설계 철학을 정확하게 이해를 하면 그 이유를 알 수 있으며,

역시 설명할 부분이 많아 다음 Class 에서 다룰 예정이다.

참고로 GoogLeNet에는 총 9개의 인셉션 모듈이 적용되어 있다.



정리하자면, 구글의 연구팀들은 망을 더 깊게 하여 성능 향상을 꾀하면서도,

연산량의 증가를 늘리지 않는 CNN 구조를 개발하기 위해 많은 연구를 하였다.

결과적으로 초기 CNN 구조가 적합하지 않다는 것을 발견하였으며,

효과적으로 차원을 줄이면서 망을 깊게 할 수 있는 방법으로 인셉션 모듈을 개발하였다.

그 후에도 구글의 개발팀들은 인셉션을 더욱 발전시켜 자신들의 CNN연구의 기본으로 삼았으며,

관련 논문들도 많이 발표를 하였다.

이번 Class에서는 구들의 GoogLeNet이 이전 CNN 구조와 많이 다르다는 것에 대하여 간단하게 살펴보았다.

다음 Class부터는 그들의 설계 철학과 왜 이런 구조를 발전시키게 되었는지에 대해

여러 번에 걸쳐 상세하게 살펴볼 예정이다.

**CNN(Convolutional Neural Network) – “GoogLeNet (part2)”**

[지난 Class](http://laonple.blog.me/220686328027)에서 구글의 GoogLeNet에 대하여 간단하게 살펴보았다.

그 동안 살펴본 LeNet-5, AlexNet, ZFNet 등은 그런대로 이해하는데 무리가 없을 것이라 생각되지만,

다소 괴상하게(?) 생긴 GoogLeNet은 왠지 부담스러울 것 같다.

GoogLeNet에서는 망의 깊이 및 넓이가 모두 커지고,

중간에 분기되는 부분도 있고, “인셉션”이라는 생소한 모듈이 등장한다.

하지만, 이 모든 것들이 구글의 연구팀들이 최초로 개발한 것은 아니며,

이들 역시 타 연구자들의 연구와 자신들의 연구를 융합하여 발전시킨 결과이다.

구글은 자신들의 구조를 설계함에 있어 크게 2개의 논문을 참조하고 있으며,

그 중 인셉션 모듈 및 전체 구조에 관련된 부분은 싱가포르 국립 대학의

“Min Lin”이 2013년에 발표한 “Network In Network” 구조를 더욱 발전 시킨 것이다.

이번 Class에서는 NIN(Network In Network)의 구조를 살펴보고,

이후 구글의 인셉션의 구조를 살펴볼 예정이다.

**NIN(Network In Network) 구조와 설계 철학**

NIN은 말 그대로 네트워크 속의 네트워크를 뜻한다.

일반적인 CNN 구조는 feature extraction 부분(conv + pooling layer)과

classifier 부분(fully connected neural network)으로 구성이 된다.

Convolutional layer와 pooling layer를 번갈아 사용하는 layer 군 여러 개를 사용하여 feature를 추출하고,

최종 feature vector를 classifier 역할을 하는 fully-connected neural network을 이용하여 처리 하였다.

NIN 설계자는 CNN의 convolutional layer가

local receptive field에서 feature를 추출해내는 능력은 우수하지만,

filter의 특징이 linear하기 때문에 non-linear한 성질을 갖는 feature를 추출하기엔 어려움이 있으므로,

이 부분을 극복하기 위해 feature-map의 개수를 늘려야 하는 문제에 주목했다.

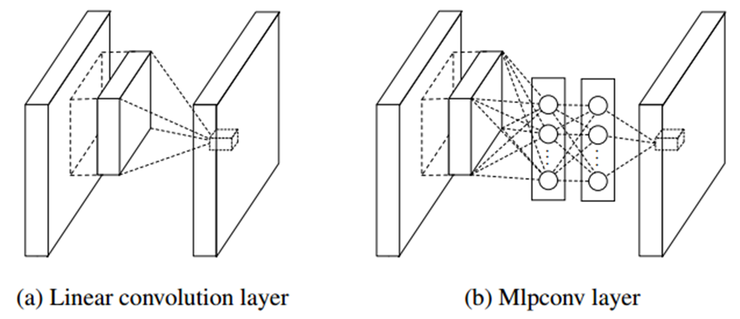
필터의 개수를 늘리게 되면 연산량이 늘어나는 문제가 있다.

그래서 NIN 설계자는 local receptive field 안에서 좀 더 feature를 잘 추출해낼 수 있는 방법을 연구하였으며,

이를 위해 micro neural network를 설계하였다.

이들은 convolution을 수행하기 위한 filter 대신에 MLP(Multi-Layer Perceptron)를 사용하여

feature를 추출하도록 하였으며, 그 구조는 아래 그림과 같다.



CNN은 filter의 커널을 입력 영상의 전체 영역으로 stride 간격만큼씩 옮겨가면서 연산을 수행한다.

반면에 NIN에서는 convolution 커널 대신에 MLP를 사용하며,

전체 영역을 sweeping 하면서 옮겨가는 방식은 CNN과 유사하다.

MLP를 사용했을 때의 장점은 convolution kernel 보다는 non-linear 한 성질을 잘 활용할 수 있기 때문에

feature를 추출할 수 있는 능력이 우수하다는 점이다.

또한 1x1 convolution을 사용하여 feature-map을 줄일 수 있도록 하였으며,

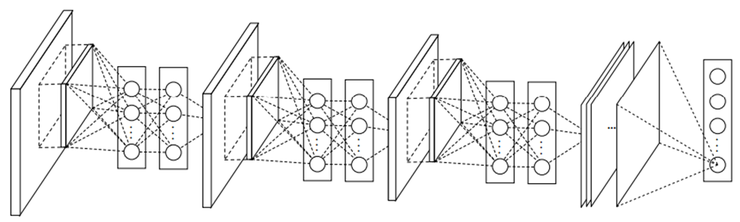
이 기술은 GoogLeNet의 인셉션에 그대로 적용이 된다.

1x1 convolution은 뒤에 좀 더 자세하게 설명을 할 예정이다.

NIN 이름이 붙은 이유는 망을 깊게 만들기 위해, mlpconv layer를 여러 개를 쌓아 사용을 하기 때문이며,

그래서 네트워크 안에 네트워크가 있다는 개념이 만들어지게 되었다.

GoogLeNet에서도 인셉션 모듈을 총 9개를 사용하기 때문에 개념적으로는 NIN과 맥이 닿아 있다고 볼 수 있다.



위 그림은 mlpconv layer 3개를 사용한 구조이다.

NIN 구조가 기존 CNN과 또 다른 점은

CNN의 최종단에서 흔히 보이는 fully-connected neural network이 없다는 점이다. (위 그림 참조)

Fully-connected NN 대신에 최종단에 “**Global average pooling**”을 사용하였다.

이는 앞에서 효과적으로 feature-vector를 추출하였기 때문에,

이렇게 추출된 vector 들에 대한 pooling 만으로도 충분하다고 주장을 하고 있다.

Average pooling 만으로 classifier 역할을 할 수 있기 때문에 overfitting의 문제를 회피할 수 있고,

연산량이 대폭 줄어드는 이점도 얻을 수 있다.

CNN의 최종단에 있는 fully-connected NN은 전체 free parameter 중 90% 수준에 육박하기 때문에,

많은 파라미터의 수로 인해 overfitting에 빠질 가능성이 아주 높으며,

이를 피하기 위해서는 앞선 Class에서 이미 살펴 본 것처럼,

“dropout” 기법을 적용해야 하지만,

NIN의 경우는 average pooling 만으로 classification을 수행할 수 있기 때문에

overfitting에 빠질 가능성이 현저하게 줄어들게 된다.

물론 GoogLeNet에서도 global average pooling이 적용이 되어 있다.

**1x1 Convolution이란?**

Convolution은 local receptive field의 개념을 적용하기 때문에

7x7, 5x5, 3x3과 같이 주변 픽셀들의 정보를 같이 활용을 한다.

그런데 괴상한(?) 이름의 1x1 convolution 이라는 개념이 나온다.

1x1 convolution을 하는 결정적인 이유는 차원을 줄이는 것이다.

GoogLeNet 소개 논문에 나오는 것처럼,

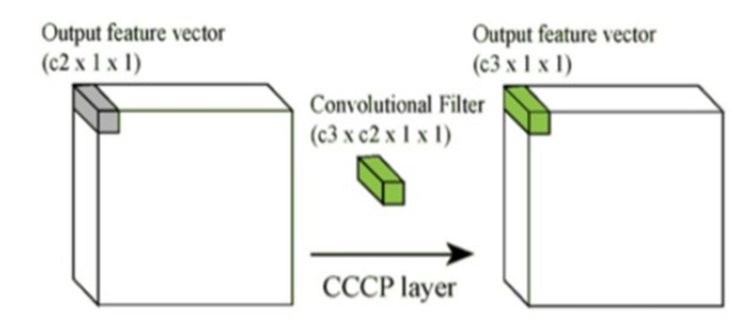
Hebbian principle(Neurons that fire together, wire together)에 의해 차원의 줄일 수 있다.

1x1 convolution을 수행하면, 여러 개의 feature-map으로부터 비슷한 성질을 갖는 것들을 묶어낼 수 있고,

결과적으로 feature-map의 숫자를 줄일 수 있으며,

feature-map의 숫자가 줄어들게 되면 연산량을 줄일 수 있게 된다.

또한 연산량이 줄어들게 되면, 망이 더 깊어질 수 있는 여지가 생기게 된다.



위 그림에서 “C2 > C3”의 관계가 만들어지면,

차원을 줄이는 것과 같은 효과를 얻을 수 있기 때문에,

GoogLeNet을 포함한 최신 CNN 구조에서는 1x1 convolution을 많이 사용한다.

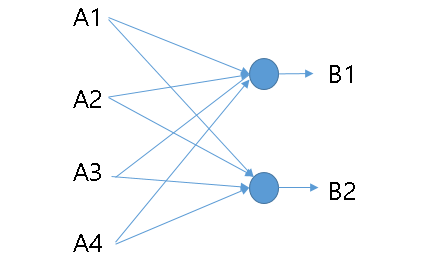
1x1 convolution은 처음에는 개념적으로 쉽게 와닿지 않는다.

논문이나 설명 글을 참고할 때 1x1 convolution을 1-layer fully-connected neural network이라고도 하는데,

그 이유는 1x1 convolution이 fully-connected와 동일한 방식이기 때문이다.

만약에 입력 feature-map c2의 갯수가 4이고, 출력 feature-map c3의 갯수가 2인 경우를 가정해보면,

1x1 convolution은 아래 그림과 같이 표현할 수 있다.



결과적으로 보면 4개의 feature-map으로부터 입력을 받아, 학습을 통해 얻어진

learned parameter를 통해 4개의 feature-map이 2개의 feature-map으로 결정이 된다.

즉, 차원이 줄어들게 되며, 이를 통해 연산량을 절감하게 된다.

또한, neuron에는 활성함수로 RELU를 사용하게 되면, 추가로 non-linearity를 얻을 수

있는 이점도 있다.

이상으로 1x1 convolution에 대한 간단한 설명을 마치고,

나중에 VGGNet을 설명할 때,

convolution을 어떻게 구현할 것인가 및 차원을 어떻게 줄여갈 것인가에 대하여 자세히 설명할 예정이다.

**구글의 인셉션(Inception)**

구글은 인셉션에 대한 개발을 하면서 NIN 구조를 많이 참고하였다.

Local receptive field에서 더 다양한 feature를 추출하기 위해

여러 개의 convolution을 병렬적으로 활용하려고 하였다.

원래 1x1 convolution, 3x3 및 5x5 convolution, 3x3 max pooling을 나란히 놓는 구조를 고안하였다.

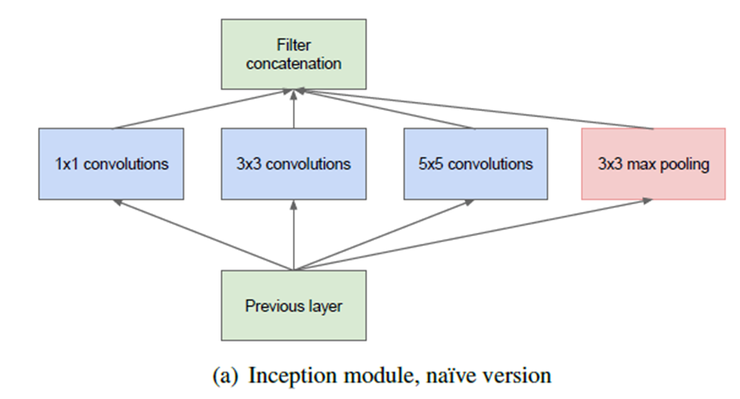
다양한 scale의 feature를 추출하기에 적합한 구조가 된다.

하지만 곧 문제에 부딪치게 된다. 3x3과 5x5 convolution은 연산량의 관점에서 보면,

expensive unit(값 비싼 대가를 치러야 하는 unit)이 된다.

망의 깊이가 깊지 않을 때는 큰 문제가 아니나,

망의 깊이와 넓이가 깊어지는 GoogLeNet 구조에서는 치명적인 결과가 될 수도 있다.



그래서 아래의 그림과 같이 3x3 convolution과 5x5 convolution의 앞에,

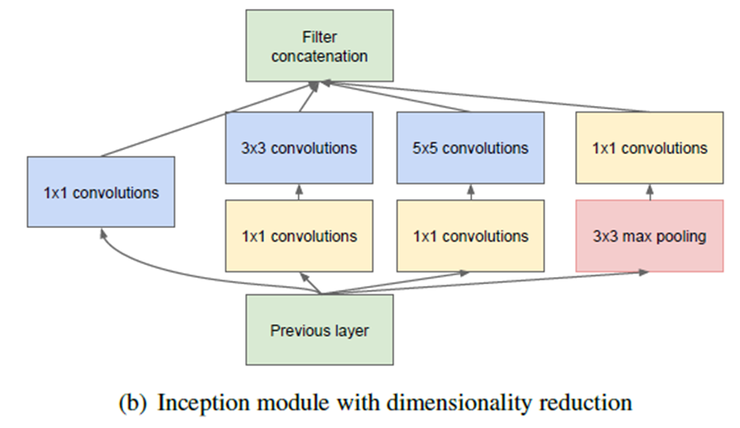
1x1 convolution을 cascade 구조로 두고, 1x1 convolution을 통해 feature-map의 개수(차원)를 줄이게 되면,

feature 추출을 위한 여러 scale을 확보하면서도, 연산량의 균형을 맞출 수 있게 된다.

GoogLeNet의 22 layer까지 깊어질 수 있는 것도 따지고 보면,

1x1 convolution을 통해 연산량을 조절할 수 있었기 때문에, 이렇게 깊은 수준까지 망을 구성할 수 있게 된 것이다.

3x3 max pooling에 대해서도 1x1 convolution을 둔다.



NIN에서는 MLP를 이용하여 non-linear feature를 얻어내는 점에 주목을 했지만,

MLP는 결국 fully-connected neural network의 형태이고, 구조적인 관점에서도 그리 익숙하지 못하다.

반면에 구글은 기존의 CNN 구조에서 크게 벗어나지 않으면서도 효과적으로 feature를 추출할 수 있게 되었다.

요약하면, 인셉션 모듈을 통해 GoogLeNet 은 AlexNet에 비해 망의 깊이는 훨씬 깊은데

free parameter의 수는 1/12 수준이고 전체 연산량도 AlexNet에 비해 적다는 것을 알 수가 있다.

참고로 GoogLeNet에는 총 9개의 인셉션 모듈이 적용되어 있다.

이제 인셉션 모듈에 대한 이해를 하였으니,

다음 Class부터는 GoogLeNet의 전체 구조에 대한 설계 철학과

왜 이런 구조를 발전시키게 되었는지에 대해 상세하게 살펴볼 예정이다.

2014년에 발표된 GoogLeNet과 VGGNet 등에서 가장 주목할 만한 특징은 망이 깊어지고 넓어졌다는 점이며, 이후CNN 구조의 대세를 이루게 된다. 그럼 왜 이렇게 망은 넓어지고 깊어지는 것일까? 답은 깊어진 망을 이용하게 되면문제 해결능력, 즉 학습 능력이 증가하기 때문이다.

하지만, 망이 깊어지게 되면 앞서 살펴보았듯이, 적절한 hyper-parameter의 설정이나 초기값 설정이 없다면overfitting 문제에 빠질 가능성이 훨씬 증가하게 되며, 연산량의 문제로 인해 아주 성능 좋은 시스템을 사용하여 학습을 할지라도 시간이 너무 오래 걸리고, 그로 인해 최적의 결과를 얻어내지 못할 수도 있게 된다. 또한 스마트 폰이나 embedded system 에서는 아주 제한된 하드웨어 자원으로 인해, 망이 깊어지는 것에 대해서는 꿈도 꾸지 못할 수도 있다.

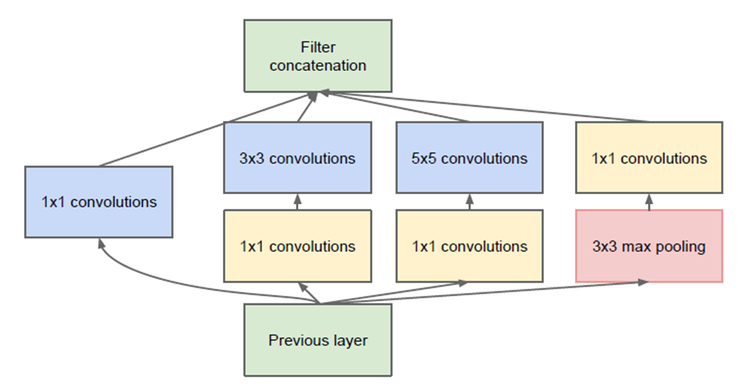
GoogLeNet 설계자들은 단순하게 hyper-parameter나 초기값을 적절하게 잘 설정하는 것만으로는 한계가 있다는 점을 잘 이해하였기 때문에 전체 구조적인 측면을 주목 하였으며, 다양한 CNN 구조 실험을 통해, 연산량을 유지하면서 망의 깊이와 넓이를 증가시킬 수 있는 구조를 얻게 되었다.

그 결과를 2014년 ILSVRC 대회 우승을 통하여 입증을 하였으며, 2015년에 발표한 그들의 또 다른 논문을 살펴보면, 좀 더 구조적인 개선을 통하여, 그 성능이 다시 2배 정도 좋아지는 쾌거를 이루게 되었다.

지난 2개의 Class를 통해, GoogLeNet의 핵심인 1x1 convolution, NIN(Network-in-Network), 인셉션 모듈의 기본 개념에 대하여 살펴보았기 때문에, 이번 Class에서는 그 개념들을 결합하여 GoogLeNet의 전체 구조에 대하여 살펴볼예정이다.

**GoogleLeNet의 핵심 철학 및 구조**

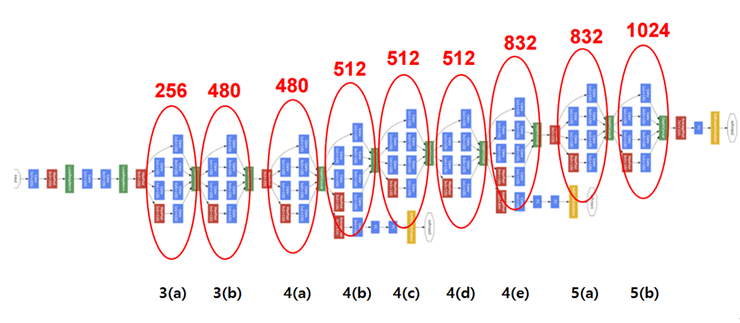
GoogLeNet 의 핵심 설계 철학은 주어진 하드웨어 자원을 최대한 효율적으로 이용하면서도 학습 능력은 극대화 할수 있도록 깊고 넓은 망을 갖는 구조를 설계하는 것이다.



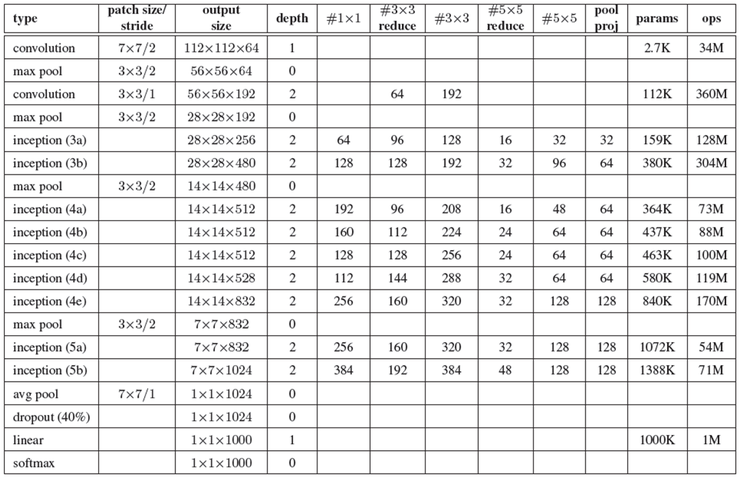
이를 위해 [[Part Ⅴ. Best CNN Architecture] 5. GoogLeNet [1]](http://laonple.blog.me/220686328027) 와 [5. GoogLeNet [2]](http://laonple.blog.me/220692793375) 에서 살펴본 것처럼, 인셉션이라는 독특한 모듈을 만들어 낸다. 인셉션 모듈에 있는 다양한 크기의 convolution kernel(그림에서 파란색 부분)을 통해 다양한 scale의 feature를 효과적으로 추출하는 것이 가능해졌다. 또한 인셉션 모듈 내부의 여러 곳에서 사용되는 (위 그림의 노란색 부분) 1x1 convolution layer를 통해, 연산량을 크게 경감시킬 수 있게 되어, 결과적으로 망의 넓이와 깊이를 증가시킬 수 있는 기반이 마련 되었다. 이 인셉션 모듈을 통해 NIN(Network-in-Network) 구조를 갖는 deep CNN 구현이 가능하게 되었다.

인셉션 모듈을 개발하고, 2014년 ILSVRC에 참가하기 위한 구조 결정을 마치고, GoogLeNet이라는 이름을 붙인다. 인셉션은 모듈 개념이기 때문에, CNN을 적용하고자 하는 목적이나 분야에 따라 그 조합을 다르게 가져갈 수도 있다. GoogLeNet의 경우는 파라미터가 있는 layer를 기준으로 총 22개의 layer를 갖는 말 그대로 deep network이 만들어진다. 그리고 실제로 망에 들어 있는 unit의 수를 합하면 약 100개의 unit이 된다. 2014년 이전에는 볼 수 없었던방대한(?) 네트웍이 만들어진 셈이다.

GoogLeNet에는 아래 그림과 같이, 총 9개의 인셉션 모듈이 적용이 되었다. 그림에서 빨간색 동그라미가 인셉션 모듈에 해당이 된다. 망이 워낙 방대하여 작은 그림으로 표현하기에 아쉬움이 있다.



위 그림에서 파란색 유닛은 convolutional layer를 의미하고, 빨간색은 max-pooling 유닛을 뜻하며, 노란색 유닛은Softmax layer이고, 녹색은 기타 function을 가리킨다. 인셉션 모듈을 나타내는 동그라미 위에 있는 숫자는 각 단계에서 얻어지는 feature-map의 수를 나타낸다.



GoogLeNet의 각 layer의 구조는 위 표와 같다. 위 표는 얼핏 보기에는 복잡해 보이지만, 조금만 살펴보면 그 내용을정확하게 이해할 수 있다.

  Patch size/stride: 커널의 크기와 stride 간격을 말한다. 최초의 convolution에 있는 7x7/2의 의미는 receptive field의 크기가 7x7인 filter를 2픽셀 간격으로 적용한다는 뜻이다.

  Output size: 얻어지는 feature-map의 크기 및 개수를 나타낸다. 112x112x64의 의미는 224x224 크기의 이미지에 2픽셀 간격으로 7x7 filter를 적용하여 총 64개의 feature-map이 얻어졌다는 뜻이다.

  Depth: 연속적인 convolution layer의 개수를 의미한다. 첫번째 convolution layer는 depth가 1이고, 두번째와인셉션이 적용되어 있는 부분은 모두 2로 되어 있는 이유는 2개의 convolution을 연속적으로 적용하기 때문이다.

  #1x1: 1x1 convolution을 의미하며, 그 행에 있는 숫자는 1x1 convolution을 수행한 뒤 얻어지는 feature-map의 개수를 말한다. 첫번째 인셉션 3(a)의 #1x1 위치에 있는 숫자가 64인데 이것은 이전 layer의 192개feature-map을 입력으로 받아 64개의 feature-map이 얻어졌다는 뜻이다. 즉, 192차원이 64차원으로 줄어들게 된다.

  #3x3 reduce: 이것은 3x3 convolution 앞쪽에 있는 1x1 convolution 을 의미하여 마찬가지로 인셉션 3(a)의 수를 보면 96이 있는데, 이것은 3x3 convolution을 수행하기 전에 192차원을 96차원으로 줄인 것을 의미한다.

  #3x3: 1x1 convolution에 의해 차원이 줄어든 feature map에 3x3 convolution을 적용한다. 인셉션 3(a)의 숫자128은 최종적으로 1x1 convolution과 3x3 convolution을 연속으로 적용하여 128개의 feature-map이 얻어졌다는 뜻이다.

  #5x5 reduce: 해석 방법은 “#3x3 reduce”와 동일하다.

  #5x5: 해석 방법은 “#3x3”과 동일하다. #5x5는 좀 더 넓은 영역에 걸쳐 있는 feature를 추출하기 위한 용도로인셉션 모듈에 적용이 되었다.

  Pool/proj: 이 부분은 max-pooling과 max-pooling 뒤에 오는 1x1 convolution을 적용한 것을 의미한다. 인셉션3(a) 열의 숫자 32 는 max-pooling과 1x1 convolution을 거쳐 총 32개의 feature-map이 얻어졌다는 뜻이다.

  Params: 해당 layer에 있는 free parameter의 개수를 나타내며, 입출력 feature-map의 數에 비례한다. 인셉션3(a) 열에 있는 숫자 159K는 총 256개의 feature-map을 만들기 위해 159K의 free-parameter가 적용되었다는 뜻이다.

  Ops: 연산의 수를 나타낸다. 연산의 수는 feature-map의 수와 입출력 feature-map의 크기에 비례한다. 인셉션 3(a)의 단계에서는 총 128M의 연산을 수행한다.

위 설명에 따라 표에 있는 각각의 숫자들의 의미를 해석해 보면, GoogLeNet의 구조를 좀 더 친숙하게 이해할 수 있다.

인셉션 3(a)에는 256이라는 빨간색 숫자가 적혀 있는데, 이것은 인셉션 3(a)를 통해 총 256개의 feature-map이 만들어졌다는 뜻이며, 이것은 1x1 convolution을 통해 64개, 1x1과 3x3 연속 convolution을 통해 128개, 1x1과 5x5 연속convolution을 통해 32개, max-pooling과 1x1 convolution을 통해 32개를 적용하여 도합 256개의 feature-map을 얻을 수 있게 되었다는 뜻이다.

3x3보다는 5x5 convolution을 통해 얻는 feature-map의 개수가 작은 이유는 5x5 convolution이 훨씬 연산량을 많이필요로 하기 때문이며, 입력 이미지의 크기가 이미 28x28로 줄어든 상황에서는 3x3으로 얻을 수 있는 feature가 5x5로 얻을 수 있는 feature보다 많기 때문일 것이다.

만약에 3x3이나 5x5 convolution 앞에 1x1 convolution이 없다면 어떻게 되었을까? 3x3 convolution의 경우는 1x1 convolution을 통해 192개의 feature-map이 96개로 줄었기 때문에 50% 정도 연산량이 줄었으며, 5x5의 경우는 192개가 16개로 줄었기 때문에, 약 91.7% 정도 연산량이 줄어들게 되었다. 특히 5x5에서 연산량이 크게 줄었기 때문에, 1x1 convolution을 통한 차원 절감의 효과를 크게 누릴 수 있다.

이번 Class에서는 GoogLeNet의 구조에 대하여 설명을 하였다. GoogLeNet은 익숙한 기존 CNN 구조와 다른 점이많아 상세한 설명을 하려면, 지면이 많이 소요된다. 다음 Class에서는 GoogLeNet에 적용된 Auxiliary classifier 및Inception –V2, Inception-V3 구조 등에 대하여 살펴볼 예정이다.

**CNN(Convolutional Neural Network) – “GoogLeNet (part4)”**

지난 3개의 Class를 통해 Inception 모듈과 GoogLeNet의 기본 개념에 대하여 살펴보았다.

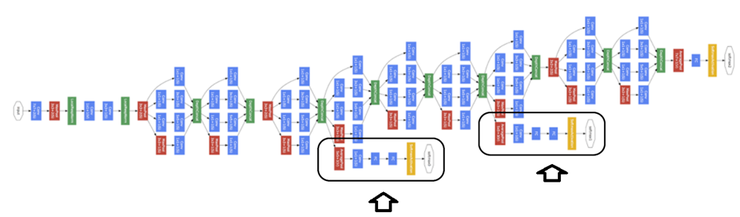
이번 Class에서는 GoogLeNet이 갖고 있는 독특한 블락인 Auxiliary classifier 및

큰 필터 크기를 갖는 convolution kernel을 인수 분해(factorization)하여

작은 크기의 여러 단의 convolution으로 구현하는 방법에 대하여 살펴볼 예정이다.

**Auxiliary classifier**

아래 그림의 GoogLeNet 블락도를 보면 Auxiliary classifier라고 불리는 독특한 유닛이 있다.



이전의 CNN 구조에서는 볼 수 없었던 독특한 구조이다.

망이 깊어지면서 생기는 큰 문제 중 하나는 vanishing gradient 문제이며,

이로 인해  학습 속도가 아주 느려지거나 overfitting 문제가 발생한다.

신경망에서는 최종 단의 error을 역전파(back-propagation)를 시키면서 파라미터 값을 갱신한다.

그런데 gradient 값들이 0 근처로 가게 되면, 학습 속도가 아주 느려지거나

파라미터의 변화가 별로 없어 학습 결과가 더 나빠지는 현상이 발생할 수 있다.

 ([[Part Ⅲ. Neural Networks 최적화] 5. 학습 속도 저하 현상의 원인](http://laonple.blog.me/220548474619) 참조)

활성함수로 sigmoid 함수를 쓰는 경우

[[Part Ⅲ. Neural Networks 최적화] 5. 학습 속도 저하 현상의 원인](http://laonple.blog.me/220548474619)에서 살펴보았던 것처럼,

sigmoid 함수의 특성상 일부 구간의 제외하면 미분값이 거의 0으로 수렴하기 때문에

출력 에러의 크기와 상관없이 학습 속도가 느려진다.

또한 Class 15에서 살펴본 것처럼, cross-entropy 함수를 사용하면 좀 더 개선은 되지만

본질적인 문제가 해결이 되는 것은 아니다.

요즘 DNN(Deep Neural Network)에서는 활성함수로 ReLU를 사용한다.

Sigmoid나 cross-entropy를 사용할 때보다 많은 이점이 있기 때문이며,

비로소 DNN을 구현할 수 있는 기반이 마련되었지만,

여러 layer를 거치면서 작은 값들이 계속 곱해지다 보면,

0근처로 수렴되면서 역시 vanishing gradient 문제에 빠질 수 있고,

망이 깊어질수록 이 가능성이 커진다.

GoogLeNet에서는 이 문제를 극복하기 위해 Auxiliary classifier를 중간 2곳에 두었다.

학습을 할 때는 이 Auxiliary classifier를 이용하여 vanishing gradient 문제를 피하고,

수렴을 더 좋게 해주면서 학습 결과가 좋게 된다.

이 Auxiliary classifier는 GoogLeNet 논문에 자세한 설명은 없지만,

이후 DNN을 연구하는 사람들이 이 구조에 대한 개선이나 이론적인 설명을 위한 논문을 발표 했으며,

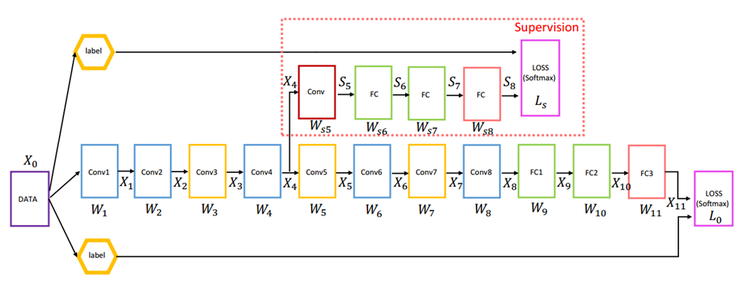
대표적인 논문을 소개하면 다음과 같다.

 > Deeply supervised nets (C.Y. Lee, S. Xie 등)

 > Training Deeper Convolutional Networks with Deep SuperVision(Liwei Wang, Chen-Yu Lee 등)

이 중 Liwei Wang의 논문이 좀 더 체계적으로 Auxiliary classifier에 대한 설명을 하고 있으니,

이 논문을 참고하면 도움이 될 것 같다.



위 그림은 Liwei Wang의 논문에 나오는 실험용 DNN의 구조이며,

X4의 위치에 그들이 SuperVision이라고 부르는 Auxiliary classifier를 배치하고,

back-propagation 시에 X4 위치에서는 Auxiliary classifier와 최종 출력으로부터 정상적인

back-propagation 결과를 결합시킨다.

이렇게 되면, Auxiliary classifier의 back propagation 결과가 더해지기 때문에 X4 위치에서

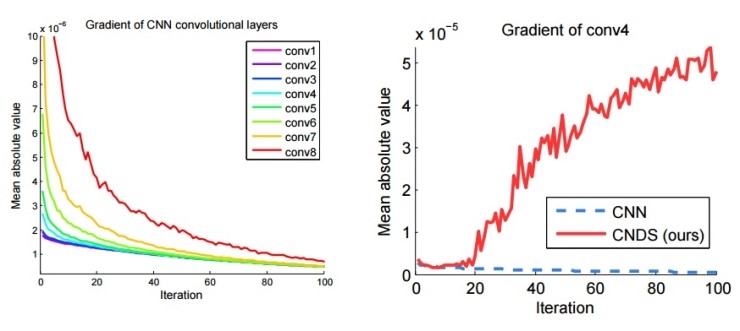
gradient가 작아지는 문제를 피할 수 있다.

GoogLeNet에서는 어느 위치에 Auxiliary classifier를 놓을 것인지,

어떤 결과를 얻었는지 명확하게 밝히지 않았지만,

Liwei 연구팀은 초기 (10~50)번 정도의 iteration을 통해 gradient가 어떻게 움직이는지 확인을 하고,

그 위치에 Auxiliary classifier를 붙이는 것이 좋다고 논문에서 밝혔다.



위 그림에서 왼쪽 그림은 Axiliary classifier가 없는 경우에  X4를 비롯한 X4 뒷단의 gradient가 현저하게

작아지는 것을 보여주는 그림이다.

오른쪽 그림은 Auxiliary classifier가 없는 경우는 파란색 점선과 같이 0에 근접한 값을 갖는 반면,

빨간색 선은 Auxiliary classifier에 의해 gradient 값이 다시 증가하게 되는 것을 보여주며,

결과적으로 더 안정적인 학습 결과를 얻을 수 있게 된다.

이후 2015년 후반에 GoogLeNet의 첫번째 저자 Szegedy가 다시 발표한 논문

“Rethinking the Inception Architecture for Computer Vision” 에

다시 Auxiliary classifier에 대한 이야기가 잠깐 나오는데,

여기에 따르면, Auxiliary classifier가 “Regularizer”와 같은 역할을 하며,

최종 단의 Main classifier가 중간의 side branch에 있는 Auxiliary classifier가 batch-normalize 되었거나

 drop-out layer를 갖고 있으면 결과가 더 좋아진다는 언급이 있다.

“batch-normalization”에 대해서는 나중에 따로 다룰 예정이다.

학습이 끝나고, 학습된 DNN을 이용할 때는 Auxiliary classifier는 삭제한다.

즉, Auxiliary classifier는 학습을 도와주기 위한 도우미 역할만을 하고,

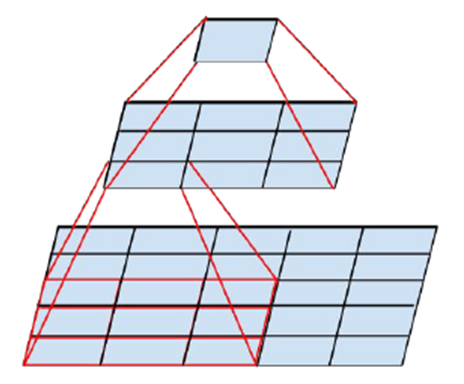
학습을 통해 결과를 얻게 되면, 본래의 역할을 다했기 때문에 제거한다.

**Factorizing Convolutions**

큰 필터 크기를 갖는 convolution 커널을 인수 분해 하면,

작은 커널 여러 개로 구성된 deep network를 만들 수 있으며,

이렇게 되면 parameter의 수가 더 줄어들면서 망은 깊어지는 효과를 얻을 수 있다.



위 그림은5x5 convolution을 2 layer의 3x3 convolution으로 구현한 경우를 보여준다.

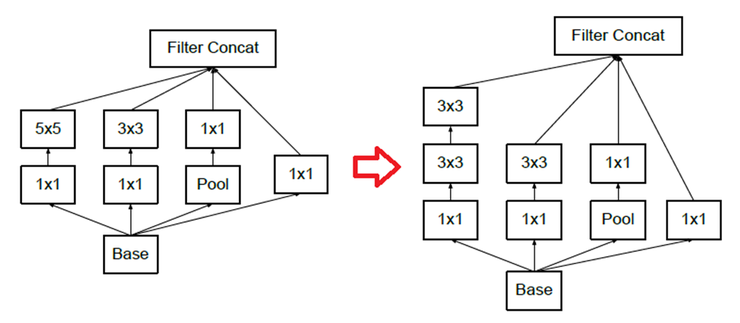
5x5 convolution은 3x3 convolution에 비해 더 넓은 영역에 걸쳐 있는 특징을 1번에 추출할 수 있지만,

25/9 = 2.78배 비싼 유닛이다.

이 5x5 convolution은 2단의 3x3 convolution을 사용해 구현이 가능하며,

이 경우는 free parameter의 수는 (9+9)로 5x5 convolution의25와 비교하면 28% 만큼 절감이 가능하다.

 아래 그림은 이 방식을 적용하여 원래의 Inception을 변형시킨 결과이다.



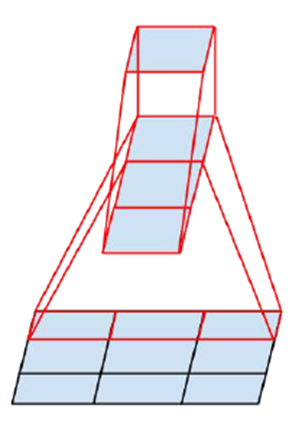
7x7 convolution의 경우도 위 경우와 마찬가지로 3 단의 3x3 convolution으로 대응이 가능하며,

이 경우는 (9+9+9) = 27이 되어 49 와 비교하여 45% 만큼 절감할 수 있다.

5x5나 7x7 convolution을 여러 단의 3x3 convolution과 같이 symmetry를 유지하는 방식으로의

인수 분해가 가능하지만, symmetry를 유지 하지 않고 row 방향 혹은 column 방향으로 인수 분해 하는 것도 가능하다.

아래 그림은 3x3 convolution을 1x3 convolution과 3x1 convolution으로 분해한 것이다.

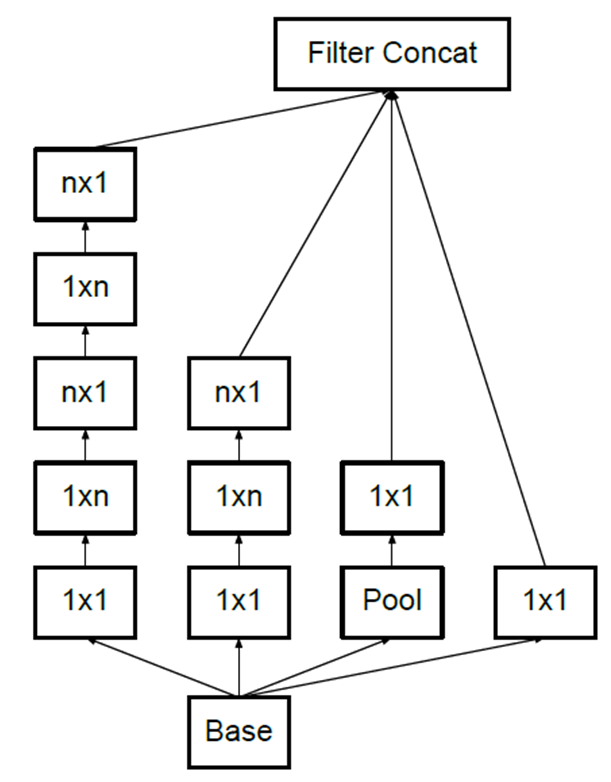


이렇게 되면, free parameter의 수는 (3+3) = 6이 되어, 9와 비교하면 33% 절감이 된다.

비슷하게 n x n convolution은 1xn 및 nx1로 분해가 가능하며, n이 클수록 파라미터 절감 효과가 커진다.

이것을 인셉션 구조에 표현을 하면 아래 그림과 같이 된다. 아래 그림에서 n = 3이면

앞서 살펴본 인셉션 모듈과 동일하다.



이번 Class에서는 GoogLeNet에 있는 Auxiliary classifier가 어떤 역할을 하는가에 대하여 살펴보았고,

큰 filter 크기를 갖는 convolution kernel을 인수 분해를 통해 여러 단의 작은 크기를 갖는 convolution으로

대체하게 되면 free-paramameter의 개수가 줄면서 연산량의 절감을 가져올 수 있다는 것을 살펴보았다.

큰 필터를 균일한 크기의 3x3으로 표현하는 것은 VGGNet의 핵심 아이디어이며,

여기서 설명한 인수 분해 기술과 다음 시간에 설명할 “효과적으로 grid 크기를 줄이는 기술”은

구글이 자신들의 인셉선 구조를 발전시킨 Inception-V2 및 Inception-V3의 핵심 아이디어가 된다.

다음 Class 에서는 효과적으로 grid 크기를 줄이는 기술 및 구글의 Inception–V2, Inception-V3 구조 등에

대하여 살펴볼 예정이다.

통상적인 CNN의 구조를 보면, convolutional layer 뒤에 pooling layer를 두고,

feature-map의 grid(해상도) 크기를 줄이는 것이 일반적이었다.

하지만 inception module을 보면

여러개의 convolutional layer와 pooling layer가 나란히 있는 좀 독특한 모양을 볼 수 있다.

이번 Class에서는 왜 이런 구조가 만들어졌는지를 살펴볼 예정이며,

이를 통해서 효과적으로 grid 크기를 줄이는 원리를 이해하고자 한다.

또한 GoogLeNet part4에서 살펴본 것처럼,

convolutional kernel에 대한 인수 분해 기술과 기타 최적화 기술을 사용하여,

인셉션 모듈을 개선하는 방법에 대해서 살펴보고,

이를 통해 성능이 어떻게 개선되었는지를 살펴볼 예정이다.

2012년 이후로 CNN 구조는 춘추전국시대와 비슷한 느낌이 든다.

누군가가 특정 방식으로 성능을 개선하면, 다른 그룹에서는 다른 방식으로 성능을 개선하였으며,

ILSVRC의 결과도 거의 매년 2배씩 좋아지고 있는 셈이다.

2016년은 누가 어떤 구조를 들고 나와 또 세상을 깜짝 놀라게 할지 기대가 된다.

**효과적으로 해상도(grid size)를 줄이는 방법**

앞에서도 이야기를 했듯이, grid 크기를 줄이는 대표적인 방식은 convolution을 수행할 때

stride를 1 이상의 값으로 설정하거나, pooling을 사용하는 것이다.

([[Part Ⅳ. CNN] 4. Convolution Layer [2]](http://laonple.blog.me/220624485850) 참고)

​

최대값을 취하는 max-pooling 방식과, 평균을 취하는 average pooling에 대해서는

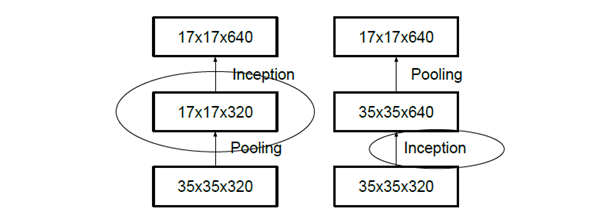
이미 [[Part Ⅳ. CNN] 3. CNN 구조](http://laonple.blog.me/220608018546) 에서 설명을 하였으니, 이 자료를 참고하면 된다.

전형적인 CNN 구조에서는 convolutional layer를 이용하여,

이전 feature-map으로부터 의미 있는 특징을 추출하며,

이 때 convolutional kernel의 개수는 숨어 있는 특징을 잘 추출할 수 있도록 충분한 수가 있어야 함은 물론이다.

그리고 다음 단에 오는 pooling layer를 이용해 feature-map의 크기를 줄이는 것이 일반적인 방식이다.



위 그림에서는 convolution 대신에 Inception으로 표시가 되어 있지만,

이것을 convolution이라고 생각을 해도 큰 차이는 없다.

35x35 크기의 320개의 feature-map을 입력으로 하여 17x17 크기의 640개의 feature-map을 얻고자 한다면,

위 방식 중 어느 쪽이 효과적으로 grid 크기를 줄이는 방식일까?

먼저 왼쪽 방식은 35x35x320 feature-map에 먼저 pooling을 적용하여 크기를 절반으로 줄인다.

뒤에 Inception을 적용하여 17x17 크기의 640개 feature-map을 얻었다.

연산량 관점에서만 보면 이 방식이 효율적인 것처럼 보이지만,

Pooling 단계를 거치면서, 原 feature-map에 있는 숨어 있는 정보(representational concept)가

사라지게 될 가능성이 있으므로 효율성 관점에서 보면 최적이라고 보기는 어렵다.

반면에 오른쪽은 Inception module을 먼저 적용하여 640개의 feature-map을 얻은 후에

pooling을 적용하여 feature-map의 크기를 줄였다.

이 경우에는 큰 크기의 feature-map에 Inception을 적용하였기 때문에

연산량의 관점에서 보면, 결과적으로 4배가 많은 셈이 된다.

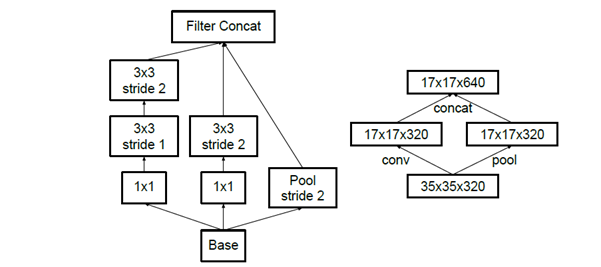
당연히 feature-map의 크기를 줄이기 전에 Inception을 적용하였기 때문에

숨은 특징을 더 잘 찾아낼 가능성은 높아진다.

그렇다면 좀 더 효과적으로 grid 크기를 줄이는 방법은 무엇일까?

Szegedy(GoogLeNet 설계자중 한 명)는

자신의 논문(Rethinking the inception architecture for computer vision)에서 아래 그림과 같은 구조를 제시한다.



먼저 왼쪽은 구조가 Inception module과 비슷하다는 것을 발견할 수 있을 것이다.

(다른 점이 있다면, 최종단에 stride 값을 "1"이 아니라 "2"로 적용했다는 점이다)

Pooling layer 및 convolutional layer를 나란하게 배치하고, 최종 단에 stride 값을 "2"를 적용하게 되면,

결과적으로 5x5, 3x3 convolution을 통해 local feature를 추출하면서 stride 2를 통해 크기가 줄고,

또한 pooling layer를 통해서도 크기를 줄이고 그 결과를 결합하는 방식이다.

오른쪽은 왼쪽보다는 좀 더 단순한 방법으로 stride 2를 갖는 convolution을 통해

320개의 feature-map을 추출하고 pooling layer를 통해 다시 320개의 feature-map을 추출함으로써

효율성과 연산량의 절감을 동시에 달성할 수 있게 되었다.

**Inception-V2**

2014년 ILSVRC를 참가할 당시에는 구글은 전년도 ZFNet의 결과보다 거의 2배 정도의 성능을 얻었기 때문에,

그리고 사람들이 식별할 수 있는 수준에 육박했기 때문에 Inception-V1 구조에 만족하였을 것 같다.

하지만, 불과 1년 후 2015 ILSVRC에서 마이크로소프트의 ResNet이

GoogLeNet 결과보다 거의 2배 좋은 성능으로 우승을 한다.

추측이기는 하지만, 이 결과에 고무되어 다시 자신들의 구조를 재 검토를 했을 것 같다.

그 결과 Inception-V2 및 Inception-V3가 나오게 되었으며,

아마 지금은 더 성능을 올리는 방법에 대한 체계적인 연구를 하고 있을 것으로 추정이 된다.

2014년에 발표한 VGGNet은 GoogLeNet과 거의 유사한 성능을 보이면서,

구조도 3x3 convolution만을 사용하는 단순한 구조로 유명하다.

여기에서 많은 힌트를 얻은 것으로 보이며,

[[Part Ⅴ. Best CNN Architecture] 5. GoogLeNet [4]](http://laonple.blog.me/220710707354) 에서 살펴 본,

convolution kernel에 대한 인수분해를 통해 망은 더 깊어지게 되고,

효과적으로 연산량은 더 절감할 수 있게 된다.

18-layer의 VGGNet이 22-layer의 GoogLeNet보다 연산량이 3배 가량 많았기 때문에,

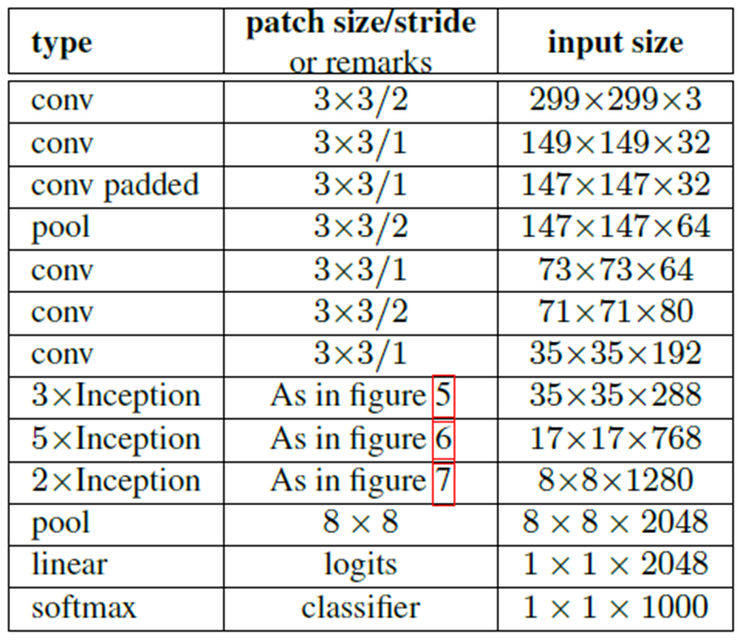
구글의 설계 철학과 맞지 않아 그들의 방식을 단순하게 따르지 않았고 단지 인수분해의 개념만을 따왔다.

본래의 Inception module을 인수분해 방식을 사용하여 좀 더 개선하고,

GoogLeNet 앞 단에 있던 7x7 convolution 등을 인수분해를 통해 작은 크기의 multi-layer 구조로 개선하였다.

아래 표는 Inception-V2의 layer 구조를 보여주는 표이다.

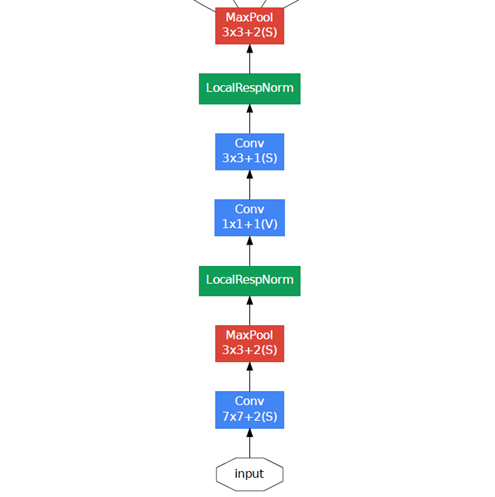
​



2014년 GoogLeNet에서는 입력 이미지로 224x224x3크기를 지원했지만,

2015년 Inception-V2 사용한 구조에서는 입력 이미지를 299x299x3의 크기를 지원할 수 있도록 하였다.

20014년 GoogLeNet의 앞 단 구조는 아래 그림과 같다.



맨 앞 단은 stride 2값을 적용한7x7 convolution 뒤에 max-pooling을 적용하여

이미지의 크기가 다시 1/4로 줄어들게 되어 56x56x64가 된다.

다음 단에서 1x1 convolution 및 3x3 convolution, max-pooling을 통해

인셉션 모듈의 입력으로는 28x28x192 크기가 적용이 된다.

하지만, Inception-V2를 적용한 2015년 구조에서는 7x7 convolution은

3개의 3x3 convolution으로 layer가 더 깊어지게 되었으며

pooling을 통해 73x73x64 크기의 feature-map이 얻어지게 된다.

[[Part Ⅴ. Best CNN Architecture] 5. GoogLeNet [4]](http://laonple.blog.me/220710707354) 에서 살펴본 인수분해 방식이 그대로 적용이 된다.

다음은 3개의 convolution을 통해 최종적으로 35x35x288 이미지가 얻어지게 된다.

2014년 구조에서는 1x1 convolution, 3x3 convolution, max-pooling을 거쳤지만,

2015년 구조는 3개의 convolution으로 구현을 하였고,

중간 과정에 stride 2를 적용하여 pooling의 효과를 얻을 수 있도록 하였다.

다음 단에는 인셉션 모듈을 적용하는 것은 비슷하나,

맨 앞 단의 인셉션 모듈의 개수가 2개에서 3개로 늘어났으며,

적용하는 인셉션 모듈의 구조도

[[Part Ⅴ. Best CNN Architecture] 5. GoogLeNet [4]](http://laonple.blog.me/220710707354) 에서 살펴본 구조로

치환이 되어 망이 더 깊어지면서 연산량은 절감할 수 있게 되었다.

최종단의 구조도 2014년 구조와 비슷하기는 하지만,

feature-map의 개수가 좀 더 많아졌다는 점이 다르다.

이런 구조 변화를 통해 22개의 layer를 갖던 2014년 구조에 비해,

총 42개의 layer로 깊어지게 되었지만,

연산량은 2.5배 늘어난 수준으로 여전히 효율성을 보인다.

그 결과는 아래 표와 같다.

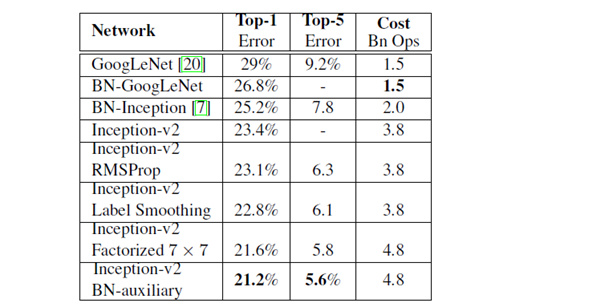
아래 표에서 GoogLeNet의 결과가 원래 결과보다 높게 보이는 이유는

data augmentation 기법을 적용하지 않고 single-crop을 했을 때의 결과이다.

아래는 Inception-V2를 다양하게 적용했을 때의 결과이며,

regularization 효과를 극대화 시키기 위해 batch-normalized auxiliary classifier를 적용하면,

성능이 5.6%까지 좋아진다는 것을 확인할 수 있다.



Muiti-crop을 144개까지 적용하고, Inception-V2의 성능을 더 극대화 시킨 Inception-V3 구조에서는

top-1 error율이 4.1%까지 떨어져 아주 우수한 성능을 보이게 된다.

이번 Class에서는 효과적으로 grid 크기를 줄이는 방법에 대하여 살펴보았다.

단순하게 pooling layer를 적용하는 것보다는

convolution layer와 같이 나란히 적용하는 것이 효과적이라는 것을 파악하였다.

또한 convolution kernel 에 대한 인수분해 방식을 적용하고,

앞 뒤 일부 구조 및 feature-map의 개수를 조정하는 것만으로도

성능을 상당히 개선할 수 있다는 것도 확인을 하였다.

GoogLeNet에서는 단순하게 분류(classification) 성능만 개선한 것이 아니라,

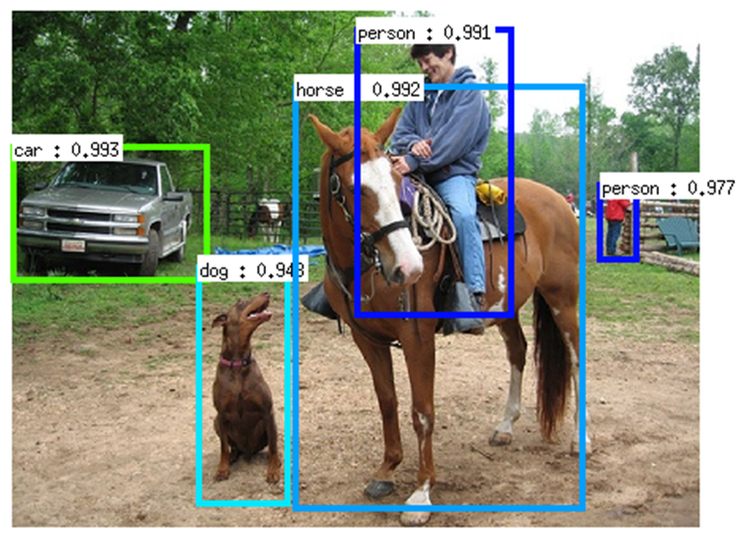
R-CNN(Regions with Convolutional Neural Networks)를 이용해

위치까지 포함한 object detection 성능을 개선하였다.

다음 Class 에서는 이 부분에 대하여 살펴볼 예정이다.

**CNN(Convolutional Neural Network) – “GoogLeNet (part6)”**

object classification은 영상 내에 특정 대상(object)이 있는지 여부를 가리는 행위로 2012년 AlexNet이 발표되면서거의 매년 성능이 2배 정도씩 좋아지는 경향을 보이고 있다. 하지만, object detection은 영상 내에 특정 대상이 존재하는지 여부를 판단하는 것 뿐만 아니라, 대상의 정확한 위치까지 파악을 하고 그것을 bounding box라고 부르는 사각형 영역으로 구분하는 것까지 수행을 하여야 하기 때문에 classification에 비해 훨씬 어렵다고 볼 수 있다.



Object detection은 machine learning 관련 지식뿐만 아니라, computer vision 관련 지식을 같이 필요로 하기 때문에, classification에 비해서는 상대적으로 어렵다고 볼 수 있고 성능 향상도 더뎠다. 그러다가 2013년 버클리 대학교의Ross Girshick 팀이 R-CNN(Regions with CNN features)이라는 방법을 발표하면서, 이전 detection 알고리즘에 비해 2배 이상의 성능 향상이 이루어지고, 뒤를 이어 여러 팀들이 속도와 성능을 개선하는 알고리즘을 속속 발표하게 된다.

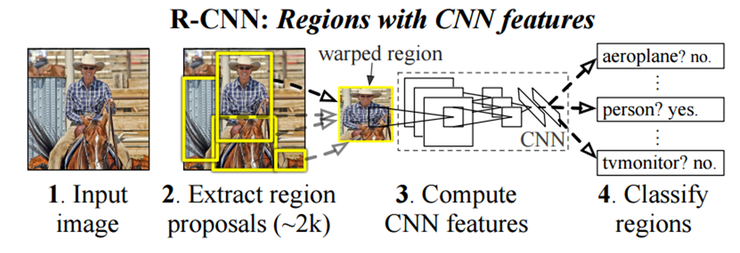
ILSVRC에서는 classification 성능뿐만 아니라, detection 성능도 같이 순위를 매긴다. GoogLeNet 설계자들은 이미앞서 살펴본 것처럼 인셉션 모듈이라는 독특한 유닛을 이용하여 classification의 성능 향상을 도모하였으며, 전년에발표된 Girshick의 R-CNN 개념을 적용하여 detection 분야에서도 1위를 차지하게 된다. 같은 해에 버클리 팀도 참여를 하였지만 mAP(mean Average Precision)에서 버클리 팀을 큰 폭으로 따돌리며 여유 있게 우승을 한다. 이는 버클리 팀이 AlexNet을 기반으로 사용하였지만,  성능면에서 GoogLeNet이 훨씬 뛰어났기 때문이라고 생각한다.

GoogLeNet에서는 R-CNN 개념과 비슷한 개념을 사용했다고만 언급을 하고 있고 어떻게 적용을 하였는지 상세 설명을 하지 않았기 때문에, 이번 class에서는 CNN feature에 기반한 detection 알고리즘인 R-CNN, SPPNet 에 대하여 살펴 볼 예정이다.

**R-CNN(Regions with CNN features)**

R-CNN 알고리즘이 발표되기 이전에 대부분 object detection에 주로 사용되던 방법은 SIFT(Scale Invariant Feature Transform)이나 HOG(Histogram of Gradient)에 기반한 알고리즘이다. SIFT나 HOG는 대부분 영상 내에 존재하는gradient 성분을 일정 블락으로 나누고 그 경향성을 이용하여 대상을 검출하는 방식을 사용한다.

영상에 존재하는 low-level feature에 기반하기 때문에 성능상의 한계가 분명히 존재하며, 이를 보완하기 위해 여러알고리즘을 섞어 사용하였다. (SIFT는 [[Part Ⅴ. Best CNN Architecture] 4. ZFNet [2]](http://laonple.blog.me/220676812642) 참고). 하지만 성능상의 한계로 인해 일정 정도 이상의 성능을 얻을 수 없었다.



버클리의 연구팀은 그동안 DPM이나 HOG 등을 이용한 detection 연구를 꾸준히 해왔던 팀이며, 2012년 AlexNet의연구 결과에 자극을 받아, CNN을 이용해 detection 연구를 하는 쪽으로 방향을 선회한 것으로 보인다. 특히 AlexNet 개발자들이 source code까지 open함에 따라 자신들의 아이디어를 실험해 볼 수 있는 기반을 얻게 된다. 그리하여classification 분야에서 얻은 괄목할만한 성과를 detection 분야에 적용할 수 있는 R-CNN이라는 독특한 구조를 발표하게 된다.

R-CNN은 위 그림과 같이 입력 영상으로부터 약 2000개의 후보 영역을 만든다. 이 때 사용하는 방법은 Selective search 방법(상세 내용은 Selective Search for Object Recognition 논문 참고)을 적용하여 후보 영역을 선정한다. Selective search는 Uijlings가 처음 발표를 하였으며, segmentation의 장점과 exhaustive search의 장점을 골고루활용을 하였으며, 영상 속에 있는 color나 texture 등 단순한 정보뿐만 아니라 영상 속에 내재된 계층 구조도 같이 활용을 한다.

Selective search를 통해 후보 영역을 선정하면, AlexNet이 224x224 크기의 이미지를 받아들이도록 되어 있기 때문에, 해당 영역을 warping이나 crop을 사용하여 224x224 크기로 만들고, 이것을 AlexNet을 약간 변형한 CNN에 인가하고 최종 출력에서 해당 영상을 대표할 수 있는 CNN feature vector를 얻어낸다.

다음은 linear SVM을 이용해 해당 영역을 분류한다.

결과적으로 보면, Computer Vision 관련 기술과 CNN 기술을 결합하여 뛰어난 성과를 얻게 된 것이다.

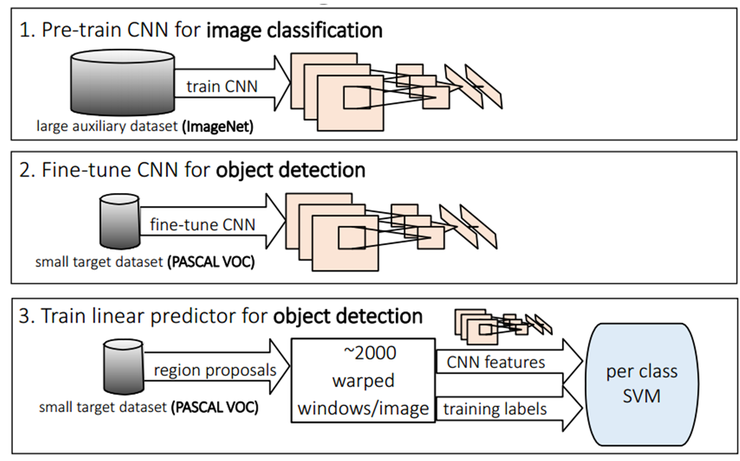
**R-CNN의 성능 향상과 학습 방법**

버클리 팀은 detection 성능 평가를 PASCAL VOC(Visual Object Class)를 사용하였는데, PASCAL VOC의 경우 label이 붙은 데이터의 양이 ILSVRC보다 상대적으로 적었기 때문에 ILSVRC 결과를 사용하여 CNN을 pre-training을 하였다. Pre-training에는 bounding box를 사용하지는 않았고, 단지 label 있는 ILSVRC data를 이용해 CNN에 있는 파라미터들이 적절한 값을 갖도록 하였다.

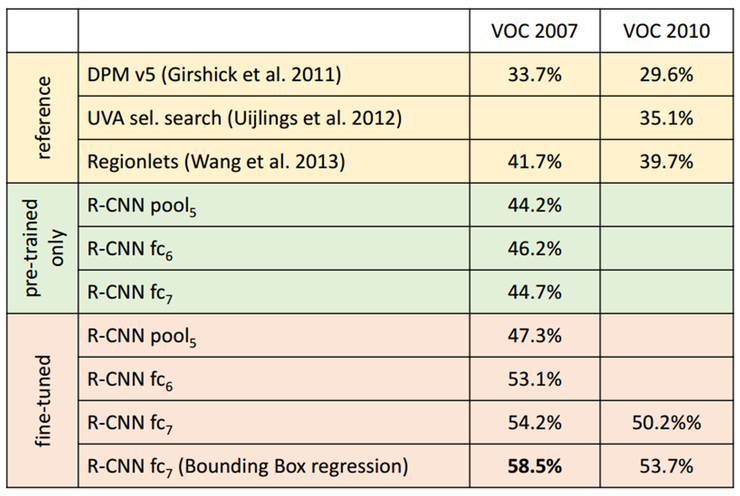
다음은 warped VOC를 이용해 CNN을 fine tuning을 한다. 이 때는 ground truth 데이터와 적어도 0.5 IoU(Intersection over Union: 교집합/합집합) 이상 되는 region 들만 postive로 하고 나머지는 전부 negative로 하여fine tuning을 시행한다. 이 때 모든 class에 대해 32개의 positive window와 96개의 background window를 적용하여128개의 mini-batch로 구성을 한다.

마지막으로 linear classifier의 성능을 개선을 위해 hard negative mining 방법을 적용하였다.

위 과정은 아래 그림과 같이 표현이 가능하다.



R-CNN을 적용하게 되면 기존 방법에 비해 아래 표와 같이 성능이 크게 개선되는 것을 확인할 수 있다.



**R-CNN의 문제점과 개선 알고리즘 SPPNet**

R-CNN은 region에 기반한 CNN feature를 사용하여 detection 성능을 크게 개선하였지만, 아무래도 처음 발표된 방식이라서 아래와 같은 문제점을 갖고 있다.

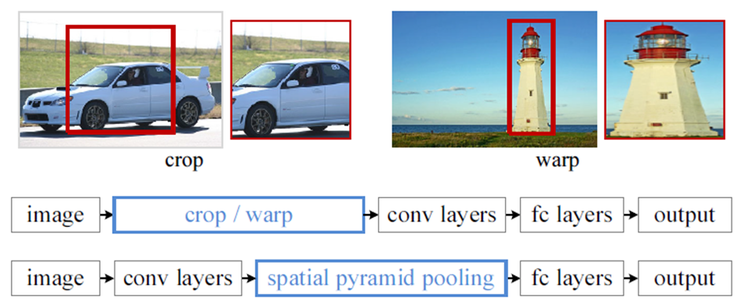
1.     AlexNet의 구조를 그대로 사용하였기 때문에 입력 이미지 크기를 강제로 224x224 크기로 맞추기 위해warping이나 crop을 사용했는데 이로 인한 이미지 변형이나 crop으로 인한 손실로 인해, 성능 저하가 일어날 수 있는 요인이 존재.

2.     2000여개에 이르는 region proposal에 대해 순차적으로 CNN을 수행해야 하기 때문에 학습이나 실제 run time이 긴 문제.

3.     사용하는 알고리즘이 특히, region proposal이나, SVM 튜닝 등이 GPU 사용에 적합하지 않다는 점.

위 문제는 SPPNet(Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition) 개발자들에 의해 상당 부분이 극복이 된다.

SPPNet은 마이크로소프트의 북경연구소에 근무하는 Kaiming He 등에 의해서 개발이 되었으며, R-CNN의 문제점을 파고 든 결과라고 보인다. 먼저 주목한 점은 AlexNet 중 convolutional layer 부분은 어차피 sliding window 방식을사용하기 때문에 영상의 크기에 영향을 받지 않으며, 뒷단에 오는 fully connected layer 부분만 영상의 크기에 영향을 받는다는 점이었다.



위 그림은 SPPNet 논문에 나오는 그림으로 crop이나 warping을 하면 위 그림처럼 왜곡이나 손실이 발생한다. 이 문제는 위 그림처럼 convolutional layer 다음에 spatial pyramid pooling layer를 두고 이 단계에서 pyramid 연산을 통해입력 영상의 크기를 대응할 수 있게 되면, 굳이 crop/warp를 사용하지 않아도 된다.

SPPNet은 BoW(Bag-of-Words) 개념을 사용한다. BoW란 특정 개체를 분류하는데 굵고 강한 특징에 의존하는 대신에서 작은 여러 개의 특징을 사용하면 개체를 잘 구별할 수 있다는 사실에 기반한다.

AlexNet이나 ZFNet과 같은 기존 신경망의 입력 영상의 크기가 고정이 되는 이유는 Convolutional layer는 영상의 크기에 영향을 받지 않지 않지만 fully-connected layer가 입력 영상의 크기에 제한을 받기 때문이다.

SPPNet 설계자들은 BoW 개념처럼 여러 단계의 피라미드 레벨에서 오는 자잘한 feature들을 fully-connected layer의 입력으로 사용하고, 피라미드의 출력을 영상의 크기에 관계없이 사전에 미리 정하면 더 이상 영상의 크기에 제한을 받지 않게 된다는 점에 주목을 하였다.

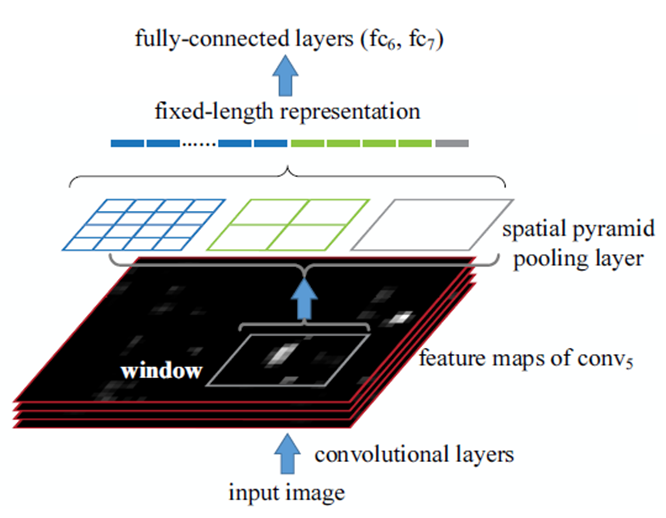
그래서 기존 ZFNet과 같은 신경망의 최종 convolutional layer를 pyramid pooling layer로 변환을 시키고, 최종 피라미드 layer에서는 직전 convolutional layer의 결과를 여러 단계의 피라미드로 나눈다. 가령 1단계는 영상 전체를 커버할 수 있도록 1x1 pooling, 2단계는 영상을 4개의 영역으로 구분한 2x2 pooling, 3단계는 영상을 9개의 영역으로 구분은 3x3 pooling 등 영상을 spatial bin이라고 불리는 총 M개의 영역으로 나눈다. 이렇게 얻어진 여러 단계의 결과는 각각을 concatenation 시킨 후 fully-connected layer의 입력으로 사용한다.

이 때입력 feature-map의 갯수가 k개 라면, 앞서 구한 M개의 벡터가 kM개의 차원으로 존재하는 셈이 된다.

SPPNet이 R-CNN에 비해 갖는 또 다른 장점 중 하나는 R-CNN은 각각의 후보 window에 대해 crop/warp를 한 후CNN 과정을 전부 거치지만, SPPNet에서는 영상 크기에 영향을 받지 않기 때문에 전체 영상에 대해 딱 1번convolutional layer를 거친 후 해당 window에 대하여 SPP를 수행한 후에 이후 과정을 거치기 때문에, 가장 시간이오래 걸리는 convolutional 과정을 건너 뛸 수가 있기 때문에 성능이 24 ~ 102 배 정도 빠르다.

SPPNet은 2014년 ILSVRC에 참여를 하였으며, detection 부분에서는 GoogLeNet에 이어 2위를 차지했으며, classification 분야에서는 3위를 차지하였다.

다음 그림은 SPPNet의 구조를 보여준다. AlexNet의 5번째 convolutional layer 다음에 SPP layer가 위치를 하며, 이후에 fully connected layer가 오는 구조를 취한다.



**GoogLeNet의 detection 소개 및 GoogLeNet 마무리**

GoogLeNet에서도 detection 구조에 대해서는 자세한 언급이 없다. 다만 R-CNN과 비슷한 구조를 사용하고 있으며, 인셉션 모델을 region classifier로 사용한다. 그리고 region proposal 단계에서는 selective search와 multi-box prediction을 혼합해서 사용한다. False positive의 수를 줄이기 위해 super-pixel의 크기를 2배 증가시켰으며, 이를통해 region proposal의 수를 1/2로 줄였으며, 이렇게 효과적으로 region proposal의 개수를 줄임으로써 mAP가 1%를 개선하였으며, 학습 시간의 문제로 bounding-box regression 은 적용하지 않았지만 성능은 R-CNN에 비해 훨씬좋은 결과를 얻을 수 있었다.

이번 class에서는 object detection 부분을 주로 다뤘다. 아무래도 detection 부분은 원조격인 R-CNN과 SPPNet에대한 부분을 다루는 편이 좋을 것 같아 이를 많이 소개했다.  Girshick은 나중에 마이크로소프트 사로 옮기고 Fast R-CNN이라는 더 개선된 알고리즘을 발표하기도 했지만, 이는 본 class의 범위를 벗어나는 것 같아 생략한다.

다음 Class부터는 2014년 ILSVRC에서 주목을 받았고, 이후 CNN 알고리즘을 개발할 때 많이 사용되는 VGGNet에대해서 살펴볼 예정이다.

**[출처]** [[Part Ⅴ. Best CNN Architecture] 5. GoogLeNet [6] - 라온피플 머신러닝 아카데미 -](https://blog.naver.com/laonple/220731472214)|**작성자** [비전솔루션 전문기업](https://blog.naver.com/laonple)