9장. 인셉션 모델과 레스넷 모델 : 꽃 이미지 분류 신경망

1. 개요

이번 과제는 지난 수년 간 발표된 주요 합성곱 신경망 모델 가운데 '거대 구조'의 대표적 사례라고 할 수 있는 인셉션 모델과 '깊은 구조'의 대표적 사례라고 할 수 있는 레스넷 모델을 통해 레지듀얼-꽃 모델을 수행하고 분석하였습니다

2. 이론

< 인셉션 모델 >

: 인셉션 모듈을 반복 활용해서 신경망 규모를 크게 늘린 모델이다. 2012년 구글넷이 처음 고안한 인셉션 모듈은 이미지 인식의 성능을 획기적으로 개선해 2014년 이미지넷 경진대회 우승의 원동력이 되었다. 인셉션 모델의 구성은 인터넷 등에 자세히 공개되어 있지만 규모가 너무 커서 좀처럼 구현해 실험할 엄두가 나지 않는다. 하지만 막상 살펴보면 만들기 힘든 구조도 아니며, 구현을 위해 깊이 살펴보는 과정에서 신경망 구조에 대한 깊은 이해와 처리 방법에 대한 통찰을 얻을 수 있다고 한다.

< 인셉션 모델과 인셉션 모듈 >

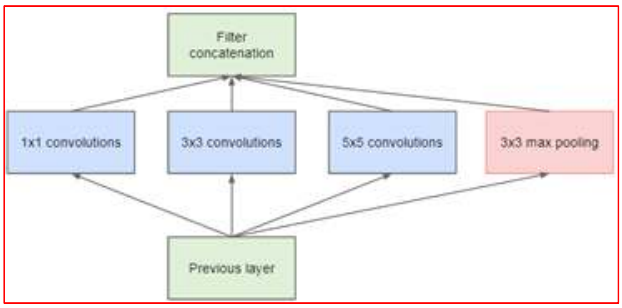
: 인셉션 모델은 다양한 병렬 구조를 이용하여 인셉션 모듈이라는 중간 구성단위를 둔다. 인셉션 모듈은 하나의 입력을 몇몇 분기에서 병렬 처리한 후 그 결과들을 한데 모아 다음 단계로 전달한다. 참고로 합성곱 신경망의 출력은 복수의 채널을 가지며 각 분기의 처리 결과들을 별도의 채널로 나열해주면 되기 때문에 쉽게 결과를 한데 모을 수 있다. 인셉션 모듈은 다양한 형태로 구성이 가능하며 대표적인 2가지는 다음과 같다.

< 기본형 인셉션 모듈 >

: 기본형 인셉션 모듈은 [mb_size, xh, xw, xchn] 형태의 4차원 입력 텐서에 대해 4갈래의 병렬 처리를 거쳐 출력을 생성한다. 4갈래 처리는 각각 1X1 합성곱, 3X3 합성곱, 5X5 합성곱, 3X3 최대치 풀링 계층으로서 합성곱 계층과 풀링 계층이 섞여 있으며 각기 다른 커널 크기를 갖는다. 이는 어느 1가지 분석 방법에 전적으로 의존하기보다 여러 방법으로 분석하고 그 결과를 종합하는 편이 좋은 결과를 가져올 것이라는 믿음에 기반을 둔 접근 방법이다. 이는 우리가 이전에 과제로 하였던 앙상블의 bagging 기법이라고도 부른다.

4갈래 처리의 결과는 합병 필터에 의해 텐서 1개로 합쳐진다. 합병 필터의 처리는 4갈래 처리 결과들을 별도의 채널들로서 차례차례 나열해 배치하는 단순한 방식으로 이루어진다. 따라서 각 갈래의 처리 결과는 채널 축을 제외한 나머지 3가지 축에 대해 같은 형태를 가져야 한다. 이 조건은 각 갈래의 처리에 'SAME' 방식으로 패딩 처리를 하면서 건너뛰기를 하지 않을 경우 만족된다. 이 경우 모든 갈래의 출력이 출력 채널 수만 제외하고는 입력과 같은 형태를 갖게 된다. 특히 풀링은 보통 이미지 해상도를 줄이는 용도로 이용되지만 인셉션 모듈 안에서는 보폭을 [1,1]로 지정하여 이미지 크기를 줄이지 않으며 단지 이미지 픽셀들을 합성곱과 다른 방식으로 처리하기 위해 이용될 뿐이다.

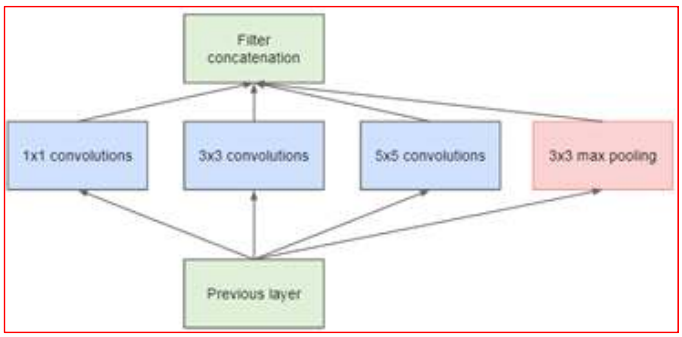
여기서 1X1 합성곱은 무의미한 픽셀 간의 1:1 대응이 아니다. 1X1 합성곱은 픽셀별로 입력 채널 전체와 출력 채널 전체를 연결하는 구조로서 각 픽셀 위치에 대해 출력 채널별로 입력 채널 정보를 종합한 내용을 생성한다. 한편 채널 수가 표시되어 있지 않은 아래의 그림만으로는 병렬구조 전체의 입출력 사이의 채널 수 관계를 알 수 없다. 그러나 4번째 갈래가 채널 수 변화가 없는 풀링 처리로써 입력과 같은 채널 수를 가질 것이며 따라서 여기에 나머지 3갈래의 채널수가 더해질 것을 고려하면 전체 출력 채널 수는 무조건 입력 채널 수보다 커지게 된다.



<기본형 인셉션 모듈>

< 개선된 인셉션 모듈 >

: 개선된 인셉션 모듈은 기본형 인셉션 모듈과 마찬가지로 4개의 분기를 가지며 각 분기에서 하는 일들도 비슷하다. 게다가 처리 전후 1X1 합성곱 계층을 추가하거나 하나의 5X5 합성곱 계층을 2개의 3X3 합성곱 계층으로 대치하는 방법을 이용해 처리의 품질을 높이면서도 전체적인 계산량을 줄여 처리 속도를 높이려고 노력하고 있다. 일단 2번째 분기에서 3X3 합성곱 계층 앞에 1X1 합성곱 계층을 추가한 것은 인접 픽셀들에 대한 처리에 앞서 각 픽셀 위치별로 채널 정보를 종합하는 단계를 두어 품질을 높여보려는 것이다. 마지막 분기에서 3X3 최대치 풀링 계층 뒤에 추가된 1X1 합성곱 계층 역시 비슷하게 픽셀별 채널 정보를 종합하는 의미가 존재한다. 하지만 이처럼 풀링 계층 전후에 1X1 합성곱 계층을 두는 것은 분기의 채널 수 변경을 가능하게 만든다는 의미도 있다. 추가된 합성곱 계층 덕분에 채널 수에 변화를 줄 수 없는 풀링 계층의 한계를 넘어설 수 있다. 아래의 그림을 보며 알아가보도록 하자.

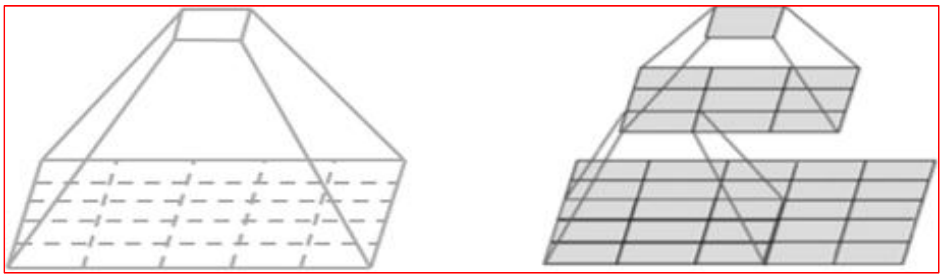


< 개선된 인셉션 모듈 >

한편 3번째 분기에는 1개의 5X5 합성곱 계층 대신 3X3 합성곱 계층을 사용하였다. 기본형 인셉션 모듈과 개선된 인셉션 모듈의 3번째 분기는 위의 그림에서 보듯 똑같이 5X5 영역의 입력 픽셀 정보를 종합하여 출력 픽셀 1개를 생성한다.

< 개선된 인셉션 모듈 - 5X5 합성곱 계층의 효율적 처리 >

: 아래의 그림에서 ①은 5X5 합성곱 연산에서 5X5 = 25회 곱셈을 수행한다. ②는 3X3 합성곱 연산 2번, 즉, (3X3)+(3X3) = 18회 곱셈을 수행한다. 이 때, 제일 위의 계층의 1개의 픽셀 게산을 위해 중간 계층 9개의 픽셀이 모두 바닥 계층으로부터 계산되어야 하기 때문에 필요한 곱셈수를 (3X3)+9X(3X3) = 90회로 착각할 수 도 있다. 그러나 중간 계층의 각 픽셀은 바닥 계층으로부터 1번 계산되면 제일 위의 계층 9개의 픽셀 계산에 반복적으로 활용되는 것을 잊지말아야 한다.



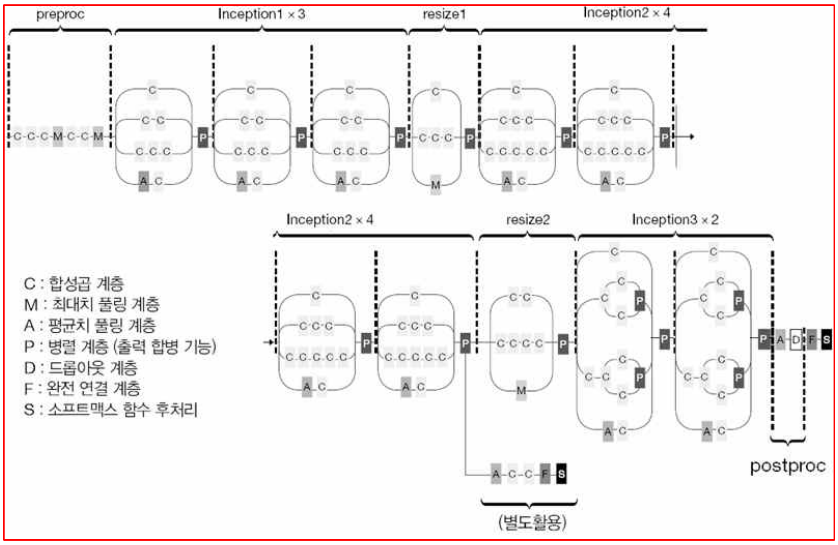
① 기본형 인셉션 모듈의 합성곱연산 **② 개선된 인셉션 모듈의 2단계 3X3 합성곱 연산**

합성곱 연산을 이처럼 변형시키는 주된 목적은 앞서 1X1 합성곱 계층 추가와는 달리 학습 품질의 개선보다 계산량의 절감을 통한 학습 속도의 개선이라 할 수 있다. 이와 비슷한 변형으로 1X7 합성곱 계층과 7X1 합성곱 계층을 연달아 배치하여 7X7합성곱 계층을 대신하는 방법도 존재한다고 한다. 이 방법은 7X7 영역의 정보를 종합하여 출력 픽셀을 만들어내면서도 픽셀당 계산 부담을 7X7 =49회에서 1X7 + 7X1 = 14회로 크게 줄인다고 한다.

< 인셉션 - v3 모델의 구조 >

: 구글넷에서는 인셉션 모듈의 다양한 형태로 활용하여 인셉션-v1부터 인셉션-v4에 이르는 다양한 인셉션 모델을 선보였다. 다음 그림은 2014년 이미지넷 경진대회에서 우승을 차지한 인셉션-v3의 구조를 나타낸 것이다. 그림에는 이 모델에 사용되는 계층들이 표시되어 있으며 계층 간의 연결이 선으로 표현되어 있다. 수직 점선으로 분리된 덩어리들이 인셉션 모듈에 해당한다. 병렬 처리 없이 맨 앞과 맨뒤에 직렬 연결된 두 부분은 전처리 모듈과 후처리 모듈이다. 또한 '별도 활용'으로 표시된 부분은 학습 결과를 다른 용도에 활용할 수 있도록 중간 처리 결과를 따로

얻어내는 부분이다. 인셉션-v3 모델은 100개가 넘는 계층과 50층에 가까운 깊이를 갖는 거대 신경망이다.

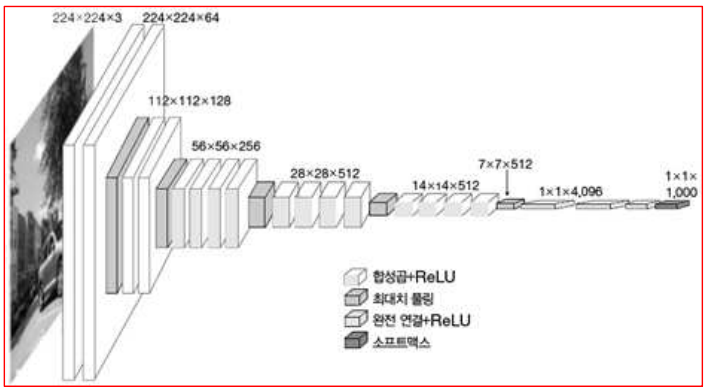


< 레스넷 모델 >

: 2015년 이미지넷 경진대회에서 우승을 차지한 레스넷 모델은 여러 딥러닝 모델 가운데 특히 깊은 구조의 신경망에 속한다. 레스넷 모델은 매우 많은 수의 레지듀얼 블록을 차례차례 연결해 생성한다. 여기에서 레지듀얼 블록은 합성곱 계층의 처리 결과에 기존의 입력을 더하는 방법을 이용해 각 합성곱 계층의 학습이 미분 수준의 작은 변화에 집중되도록 만들어주는 간단하면서도 독특한 병렬 처리 구조다. 딥러닝 분야에서 오래 전부터 신경망의 계층 구성이 깊어질수록 추상화 능력이 높아져 문제 해결 능력이 커질 것으로 기대해왔다. 하지만 깊은 구조의 신경망은 학습시키기가 매우 힘들었고 드롭아웃이나 배치 정규화 등 여러 가지 정규화 기법을 적용하여도 어느 한도 이상의 깊이 증가는 오히려 정확도를 떨어트리는 결과를 초래했다고 한다. 하지만 이런 문제는 여러 가지 노력을 통해 점점 극복되고 있는데 레스넷 모델은 그중 대표적인 사례에 해당한다고 한다.

< VGC-19 모델 >

: 레스넷 개발은 2014년 이미지넷 경연대회에서 준우승을 차지한 VGC-19 모델을 받아 만들어졌다. 레지듀얼 블록을 도입하기 이전이어서 아직 레스넷 모듈로 볼 수 없는 VGC-19 모델은 아래의 그림과 같은 처리 과정을 가진다.



VGC-19 모델에서는 16개의 합성곱 계층과 5개의 풀링 계층을 이용해 224X224X3 해상도의 입력 이미지에서 7X7X512의 미니맵을 얻어낸다. 그 후 3개의 완전 연결 계층을 이용해 1000가지 이미지 카테고리에 대한 로짓값 벡터를 출력한다. 이 과정을 계층 구성으로 바꿔 표현하면 교재의 그림과 같이 나타낼 수 있다.

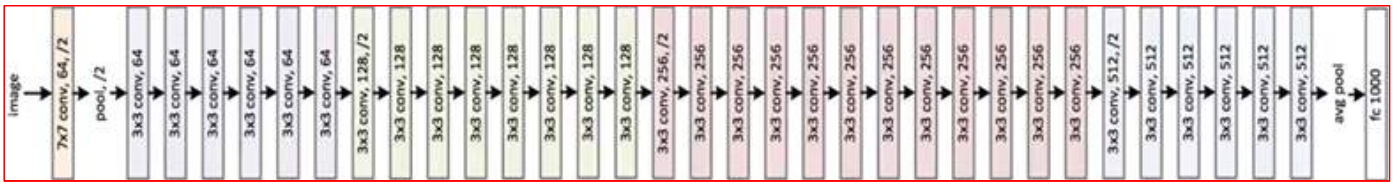
VGC-19 모델에서는 연달아 배치된 3X3 합성곱 계층들 뒤에 최대치 풀링 계층 1개가 나타나는 형태로 구성되는 모듈 구조를 다섯 번 반복해 이용한다. 처음 두 모듈은 다루는 이미지가 아직 커서 계산 부담을 줄일 의도로 합성곱 계층을 두 개만 배치하지만, 이미지가 작아지는 뒤에는 3 모듈에서 각각 합성곱 계층을 네 개씩 이용한다. 각 모듈에서 첫 합성곱 계층은 채널 수를 절반으로 줄이며 이어지는 합성곱 계층이나 풀링 계층들은 이 채널 수를 계속 유지한다. 이미지 해상도는 합성곱 계층에서는 변하지 않지만 풀링 계층을 거칠 때마다 가로와 세로 방향에서 각각 절반으로

줄어든다. 따라서 모듈 하나를 거칠 때마다 은닉 벡터의 크기는 전체적으로 반으로 줄어들면서 갈수록 압축적이고 추상적인 정보가 생성되는 것이다.

VGC-19는 신경망 뒷단에 폭이 4096인 완전 연결 계층이 3번이나 연달아 나타난다. 실제로는 이 뒤에도 출력 계층이 존재하기 때문에 전체적으로 완전 연결 계층은 무려 네 차례나 나타나는 셈이다. 분류 대상 이미지 카테고리 개수가 1000이어서 출력 계층은 폭이 1000인 완전 연결 계층 형태가 되기 때문이다. 하지만 요즘 거대 신경망에서 완전 연결 계층을 이처럼 여러 단계 두는 경우는 드물다고 한다. 오히려 대부분의 거대 신경망들은 은닉 계층 구성에 완전 연결 계층 구성에 완전 연결 계층을 전혀 포함시키지 않는다. 하지만 은닉 계층들 뒤에는 완전 연결 형태의 출력 계층이 숨어 있다고 볼 수 있다.

< 플레인-34 모델 >

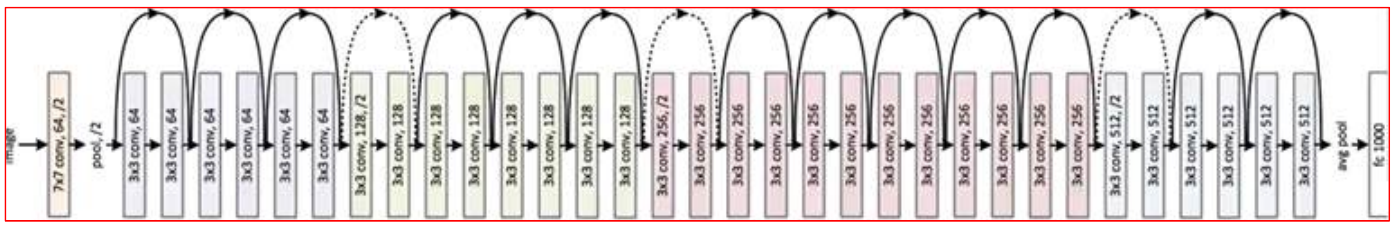
VGC-19모델 이후에 이를 확장하여 고안된 구조가 플레인-34모델이다. 플레인-34는 3X3 합성곱 계층을 33층으로 늘리는 대신 완전 연결 계층을 출력 계층에만 1개 남겨두었다. 또한 합성곱 계층 4개에 보폭을 지정해 이미지 해상도를 절반으로 줄였다. 이에 따라 풀링 계층들이 대부분 사라지고 앞뒤의 2개만 남게 되었다. 플레인-34 모델은 아래의 그림과 같은 구성을 갖는다.



VGC-19나 플레인-34의 취지는 인셉션 모델과 무척 대비된다. 다양한 커널 크기의 합성곱 계층을 병렬로 이용하면서 앙상블 효과를 기대하는 인셉션 모델과 달리 3X3 이라는 가장 기본적인 합성곱 계층 1가지만을 단 한줄로 길게 배치하여 집요하게 반복해서 이용한다. 아마도 이는 224X224X3 형태의 입력 이미지가 반복되는 이 과정을 통해 7X7X512 형태로 변형되어 가면서 마치 진화 과정처럼 작은 변화가 점진적으로 누적되어 필요한 큰 변모를 가져오기를 기대하는 생각에서 비롯된 것 같다.

< 레지듀얼 블록과 레지듀얼-34 모델 >

: 작은 변화의 누적이라는 VGC-19 모델이나 플레인-34 모델에서의 기대를 더욱 확실히 한 구조가 다음의 그림과 같은 구성을 갖는 레지듀얼-34 모델이다.

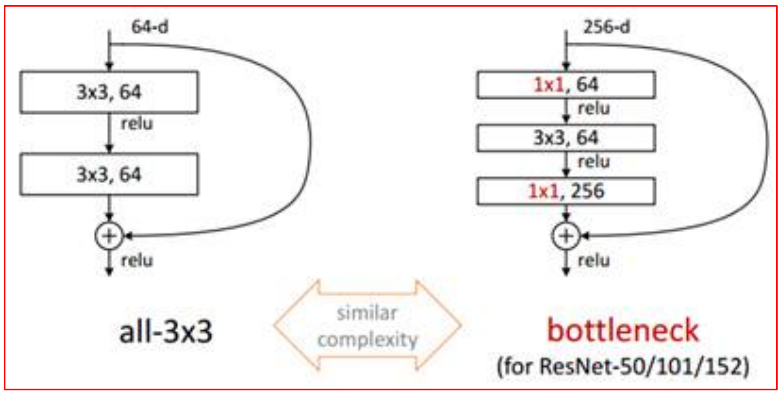


레지듀얼-34 모델에서는 플레인-34 모델의 플레인 블록들을 레지듀얼 블록들로 대체하였다. 플레인 블록의 경우 2개의 연속된 3X3 합성곱 계층을 형성하지만 레지듀얼 블록은 플레인 블록 처리 결과에 블록의 입력을 가산한 형태이다. 여기서 레지듀얼 블록의 의미는 다음과 같다.

- 입력 x에 작은 변화가 점진적으로 더해진다는 의미에서 플레인 블록 출력을 P(x) =x+p(x), 레지듀얼 블록 출력을 R(x) =x+r(x) 라 하면 플레인 블록은 두 개의 합성곱 계층이 x+p(x) 전체를 만들어내야 하지만 레지듀얼 블록은 작은 변화 r(x) 의 포착에만 집중하면 된다. 이로써 입력 내용 재현 부담을 덜어주는 만큼 성능 향상 기대 가능하며, 성공적 결과 덕분에 레지듀얼 네트워크를 줄인 레스넷이라는 모델 이라는 이름도 생겨났다.

< 보틀넥 블록과 레스넷-152 모델 >

: 보틀넥 블록은 아래의 왼쪽 그림의 레지듀얼 블록을 오른쪽 그림처럼 변형시켜 연산량을 줄이면서도 입출력 채널 수를 크게 늘려 더욱 효과적인 처리가 가능하게 하였다. 이러한 점에 착안하여 고안된 보틀넥 블록을 이용해 레스넷-50, 레스넷-101, 레스넷-152 등 매우 깊은 구조의 레스넷 모델들이 만들어졌다.



보틀넥 모델은 연산량을 줄이면서도 입출력 채널 수를 크게 늘려 효과적인 처리 기대하였으며, 기존의 레지듀얼 블록은 64채널 입력 채널 수 유지한 채 3 × 3 합성곱 두 번 수행 (커널 파라미터 수: 3×3×64×64+3×3×64×64 = 73,728개)하는 반면, 보틀넥 블록은 256채널 입력을 내부적으로 64채널로 변환해 처리하고 전후의 1 × 1 합성곱 계층을 이용해 채널수 축소 및 복원한다. (커널 파라미터 수: 1×1×256×64+3×3×64×64+1×1×64×256 = 69,632개) 또한, 합성곱 연산에 필요한 곱셈 횟수 역시 같은 비율로 감소한다. 이에 따른 보틀넥 모델의 장점과 단점을 정리하여 설명하면 다음과 같다.

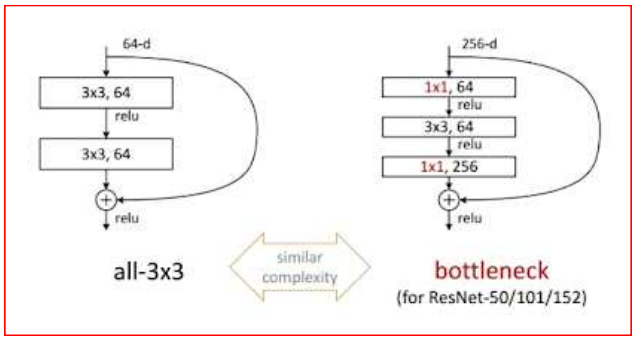
[보틀넥 모듈의 장점]
- 보다 적은 수의 파라미터와 더 적은 양의 연산 수행으로 효율 상승할 수 있다.
- 외부적으로는 오히려 정보를 64채널에서 256채널로 네 배나 늘려서 제공한다.
- 빈번히 수행되는 1 × 1 합성곱 계층 처리 (채널 간의 관계가 자주 종합되고 조정된다.)

[보틀넥 블록의 단점]
- 출력 픽셀에 반영되는 입력 영역의 크기가 3 × 3으로 감소
  (레지듀얼 블록: 두 번의 3 × 3 합성곱 처리로 5 × 5 입력 영역 반영)
- 보틀넥 블록 거칠 때마다 영향을 미치는 입력 영역의 폭이 넓어짐
- 깊은 구조의 레스넷 모델에서는 이런 단점이 문제가 되지 않음

[보틀넥 블록을 이용한 매우 깊은 구조의 레스넷 모델]
- 레스넷-50, 레스넷-101, 레스넷-152
- 레스넷-152 모델: 50개의 보틀넥 블록을 순차적으로 연결해 이용한다.
- 레지듀얼 블록이나 보틀넥 블록 및 이들의 다양한 변형 활용이 연구 중이다.
- 1,000 계층을 넘는 깊이를 갖는 레스넷 모델들도 활발히 연구 중이다.

1. 보틀넥 모듈의 특징 및 장단점은 무엇인가?

- 다음 그림에서 왼쪽의 레지듀얼 블록을 오른쪽 그림처럼 변형시키면 연산량을 줄이면서도 입출력 채널 수를 크게 늘려 더욱 효과적인 처리가 가능하다. 이러한 점에 착안하여 고안된 보틀넥 블록을 이용해 레스넷-50, 레스넷-101, 레스넷-152 등 매우 깊은 구조의 레스넷 모델들이 만들어졌다.



이렇게 보틀넥 모듈은 연산량을 줄이면서도 입출력 채널 수를 크게 늘려 더욱 효과적인 처리가 가능한 특징을 가지며 장단점들은 위에서 언급한 것과 같으며 다음과 같다.

[보틀넥 모듈의 장점]
- 보다 적은 수의 파라미터와 더 적은 양의 연산 수행으로 효율 상승할 수 있다.
- 외부적으로는 오히려 정보를 64채널에서 256채널로 네 배나 늘려서 제공한다.
- 빈번히 수행되는 1 × 1 합성곱 계층 처리 (채널 간의 관계가 자주 종합되고 조정된다.)

[보틀넥 블록의 단점]
- 출력 픽셀에 반영되는 입력 영역의 크기가 3 × 3으로 감소
  (레지듀얼 블록: 두 번의 3 × 3 합성곱 처리로 5 × 5 입력 영역 반영)
- 보틀넥 블록 거칠 때마다 영향을 미치는 입력 영역의 폭이 넓어짐
- 깊은 구조의 레스넷 모델에서는 이런 단점이 문제가 되지 않음

[보틀넥 블록을 이용한 매우 깊은 구조의 레스넷 모델]
- 레스넷-50, 레스넷-101, 레스넷-152
- 레스넷-152 모델: 50개의 보틀넥 블록을 순차적으로 연결해 이용한다.
- 레지듀얼 블록이나 보틀넥 블록 및 이들의 다양한 변형 활용이 연구 중이다.
- 1,000 계층을 넘는 깊이를 갖는 레스넷 모델들도 활발히 연구 중이다.

< 두 모델의 구현을 위해 필요한 확장들 >
: 인셉션 모델은 병렬 구조가, 레스넷 모델은 신경망 처리 결과와 입력의 합산이 특징이며 정리하면 다음과 같다.

[인셉션 모델과 레스넷 모델의 특징]
- 인셉션 모델은 병렬 구조, 레스넷 모델은 레지듀얼 입력의 합산이다.
- 신경망 은닉 계층의 구성이 다소 복잡해질 뿐
  -> 입력 이미지들에 대한 선택 분류 처리의 큰 틀은 변하지 않는다.
  -> 선택분류 기능의 학습과 평가에 관련한 프로그램 골격에 대한 변화는 불필요하다.

[필요한 확장 개요]
- 세 가지 복합 처리 계층의 추가
  -> 병렬(parallel) 계층은 인셉션 모델을 위한 병렬 처리 기능이 지원가능하다.
  -> 순차(serial) 계층은 병렬 계층 분기 내에서의 순차 처리 기능이 지원가능하다.
  -> 합산(add) 계층은 레지듀얼 입력 합산 기능이 지원가능하다.
- 두 가지 편의 계층의 추가
  -> 반복(loop) 계층은 동일하거나 유사한 내용의 반복을 간단하게 처리하는 계층이다.
  -> 사용자정의(custom) 계층은 매크로 이용한 가독성 높은 모듈 표현을 지원하는 계층을 의미한다.
- 합성곱 계층과 풀링 계층의 기능 확장
  -> 패딩 방식, 커널 크기, 건너뛰기 보폭에 제한 없이 지원한다.
- 합성곱 계층과 합산 계층에 'actions' 옵션 추가
  -> [선형연산, 활성화 함수 적용, 배치 정규화]의 수행 여부 및 처리 순서를 제어한다.

2. 인셉션 모델과 레스넷 모델의 비교

- 인셉션 모델의 경우 '거대 구조'의 대표적 사례라고 할 수 있으며, 레스넷 모델은 '깊은 구조'의 대표적 사례라고 할 수 있다.

- 인셉션 모델은 병렬 구조, 레스넷 모델은 레지듀얼 입력의 합산 형태를 띄고 있다.

- 두 가지 모델 모두 신경망 은닉 계층의 구성이 다소 복잡해질 뿐 입력 이미지들에 대한 선택 분류 처리의 큰 틀은 변하지 않는다. 또한, 선택분류 기능의 학습과 평가에 관련한 프로그램 골격의 변화는 모델들에게 필요하지 않다.

- 레스넷 모델과 주요하게 관련된 모델인 플레인-34 모델의 경우 인셉션 모델과 비교하였을 때, 다양한 커널 크기의 합성곱 계층을 병렬 형태로 이용한다. 또한, 점진적 변화를 기대하고는 있으나 주로 앙상블 효과를 기대하고 있다.

- 인셉션 모델은 '거대 구조'의 대표적 사례라고 할 수 있으며, 인셉션 모듈을 반복 활용해서 신경망 규모를 크게 늘린 모델이다. 인셉션 모듈은 병렬 구조 지원을 위한 인셉션 모델의 다양한 중간 구성단위를 의미한다. 하나의 입력을 다수의 분기에서 병렬 처리할 수 있으며, 각 분기의 처리 결과들을 모아 다음 단계로 전달한다. 인셉션 모델은 앙상블의 배깅 기법에 기반을 두고 접근하고 있다.

- 레스넷 모델은 여러 딥러닝 모델 가운데 특히 깊은 구조의 신경망에 속한다. 레스넷 모델은 매우 많은 수의 레지듀얼 블록을 차례차례 연결해 생성한다. 여기에서 레지듀얼 블록은 합성곱 계층의 처리 결과에 기존의 입력을 더하는 방법을 이용해 각 합성곱 계층의 학습이 미분 수준의 작은 변화에 집중되도록 만들어주는 간단하면서도 독특한 병렬 처리 구조다. 깊은 구조의 신경망은 학습시키기가 매우 힘들었으며 드롭아웃이나 배치 정규화 등 여러 가지 정규화 기법을 적용하여도 어느 한도 이상의 깊이 증가는 오히려 정확도를 떨어트리는 결과를 초래했다. 하지만 이런 문제는 여러 가지 노력을 통해 점점 극복되고 있다고 한다.

- 레스넷 모델의 경우 보틀넥 블록을 활용하여 기존의 레스넷 보다 더 깊은 구조의 레스넷 모델들을 만들 수 있다.

3. 실험 결과 및 파라미터별 학습결과

3-1. < cnn_inception_test.ipynb >

1. model_flower_LAB (act : LA)

```
Model model_flower_LA train started:
    Epoch 2: cost=1.601, accuracy=0.238/0.230 (841/841 secs)
    Epoch 4: cost=1.601, accuracy=0.244/0.220 (840/1681 secs)
    Epoch 6: cost=1.601, accuracy=0.244/0.230 (838/2519 secs)
    Epoch 8: cost=1.600, accuracy=0.245/0.220 (847/3366 secs)
    Epoch 10: cost=1.601, accuracy=0.244/0.230 (716/4082 secs)
Model model_flower_LA train ended in 4082 secs:
Model model_flower_LA test report: accuracy = 0.242, (19 secs)

Model model_flower_LA Visualization
```



```
추정확률분포 [18,24,18,17,22] => 추정 dandelion : 정답 dandelion => O
추정확률분포 [18,24,18,17,22] => 추정 dandelion : 정답 rose => X
추정확률분포 [18,24,18,17,22] => 추정 dandelion : 정답 tulip => X
```

- '#act'값을 전달하여 model_flower_LA 객체의 모든 합성곱 계층의 'actions' 옵션값을 'LA'로 지정하였다.
- 'L'은 선형 연산, 즉 커널을 이용한 합성곱 연산을 수행을 의미하고 이 후 'A'로 지정되는 비선형 활성화 함수인 ReLU 함수를 적용하여 진행된다. ('B'는 정규화를 의미한다.)
- 실험 결과는 24.2%로 30%도 되지 않는 정확도를 보였으며, 학습이 크게 부족한 상태임을 확인할 수 있었다. 또한, 중간 로그 출력에서 확인되는 cost값이 미미하게 변화하는 걸로 보아 학습 횟수의 변화에도 더 나은 결과를 얻어진다고 확신할 수 없다.
- 시각화 함수에서 출력된 추정확률 분포가 세 데이터에 대해 모두 같은 것을 볼 때 아마도 모든 데이터에 대해 dandelion으로 답을 추정하고 있는 상태임을 확인할 수 있었다.
- 이로써 평가 데이터셋 안의 dandelion의 비율이 24.2%일 것으로 추측할 수 있다.

2. model_flower_LAB (act : LAB)

```
Model model_flower_LAB train started:
    Epoch 2: cost=1.529, accuracy=0.308/0.240 (570/570 secs)
    Epoch 4: cost=1.492, accuracy=0.313/0.230 (572/1142 secs)
    Epoch 6: cost=1.552, accuracy=0.294/0.250 (569/1711 secs)
    Epoch 8: cost=1.540, accuracy=0.306/0.230 (571/2282 secs)
    Epoch 10: cost=1.512, accuracy=0.330/0.290 (578/2860 secs)
Model model_flower_LAB train ended in 2860 secs:
Model model_flower_LAB test report: accuracy = 0.306, (22 secs)

Model model_flower_LAB Visualization
```

```
추정확률분포 [ 4,19,12,33,32] => 추정 sunflower : 정답 tulip => X
추정확률분포 [33,39,18, 2, 8] => 추정 dandelion : 정답 daisy => X
추정확률분포 [22,31,16,15,17] => 추정 dandelion : 정답 dandelion => O
```

- '#act'값을 전달하여 model_flower_LA 객체의 모든 합성곱 계층의 'actions' 옵션값을 'LAB'로 지정하였다. (이전의 모델에 정규화까지 진행한 형태이다.)
- 선형 연산과 ReLU 함수의 적용으로 얻어진 출력에 배치 정규화를 적용한 것이다.
- 실험 결과 30.6%로 1번의 모델의 결과보다 약 6% 정도 향상된 모습을 확인할 수 있었다.
- 인셉션-꽃 모델을 다른 형태로 변형시켜 적용해본다면 좀 더 나은 성능을 얻을 수 있을 것으로 보인다.
- 시각화 함수에서 출력된 추정확률 분포가 이전과는 다르게 일정하지 않으나 여전히 dandelion으로 많은 결과를 예측하는 모습을 보였다.

3-2. <cnn_resnet_test.ipynb >

3. 레지듀얼 입력이 없는 플레인-꽃 모델로 학습

```
16: full, [1, 1, 64]=>[6] pm:64x6+6=390
Total parameter count: 173702
Model plain_flower train started:
    Epoch 2: cost=1.644, accuracy=0.281/0.220 (395/395 secs)
    Epoch 4: cost=1.569, accuracy=0.309/0.180 (363/758 secs)
    Epoch 6: cost=1.516, accuracy=0.337/0.220 (365/1123 secs)
    Epoch 8: cost=1.484, accuracy=0.344/0.180 (359/1482 secs)
    Epoch 10: cost=1.456, accuracy=0.369/0.290 (356/1838 secs)
Model plain_flower train ended in 1838 secs:
Model plain_flower test report: accuracy = 0.297, (24 secs)

Model plain_flower Visualization
```

```
추정확률분포 [67,33, 0, 0, 0, 0] => 추정 daisy : 정답 daisy => O
추정확률분포 [21,32, 1, 1,45, 0] => 추정 sunflower : 정답 tulip => X
추정확률분포 [62,35, 0, 3, 0, 0] => 추정 daisy : 정답 tulip => X
```

- 플레인-꽃 모델은 플레인-34 모델 구성에 사용된 매크로 모듈들을 이용해 쉽게 정의되며, 약 17만 개의 파라미터를 갖는다고 한다.
- 10 에포크로 학습을 시킨 결과 정확도는 29.7%로 낮은 성능을 보이고 있음을 확인할 수 있었다.
- cost가 꾸준하게 줄어들고 있음을 고려하면 학습횟수를 늘리면 현재 성능보다 더나은 결과를 얻을 수 있을 것이다.

4. 레지듀얼 입력을 갖는 레지듀얼-꽃 모델로 학습시키기

```
18: full, [1, 1, 64]=>[6] pm:64x6+6=390
Total parameter count: 173702
Model residual_flower train started:
    Epoch 2: cost=1.368, accuracy=0.438/0.250 (367/367 secs)
    Epoch 4: cost=1.286, accuracy=0.467/0.330 (351/718 secs)
    Epoch 6: cost=1.239, accuracy=0.496/0.290 (350/1068 secs)
    Epoch 8: cost=1.171, accuracy=0.532/0.180 (357/1425 secs)
    Epoch 10: cost=1.120, accuracy=0.544/0.250 (372/1797 secs)
Model residual_flower train ended in 1797 secs:
Model residual_flower test report: accuracy = 0.312, (24 secs)

Model residual_flower Visualization
```

```
추정확률분포 [14,11, 1,42, 0,32] => 추정 rose : 정답 sunflower => X
추정확률분포 [10,16, 1,47, 0,25] => 추정 rose : 정답 dandelion => X
추정확률분포 [ 9,13, 2,46, 0,30] => 추정 rose : 정답 sunflower => X
```

- 플레인-꽃 모델과 비슷한 구조이지만 레지듀얼 입력을 추가하여 레스넷 모델로 변형시킨 레지듀얼-꽃 모델을 만들고 학습시켜 보았다.
- 레지듀얼-꽃 모델 역시 레지듀얼-34 모델 구성에 사용된 매크로 모듈들을 이용하여 쉽게 정의할 수 있었다.
- 레지듀얼 입력이 추가된 점을 제외하고는 플레인-꽃 모델과 전체적으로 같은 구조를 띄고 있다.
- 이전의 실험과 같이 10 에포크를 학습시킨 결과 31.2%로 이전의 실험과 비슷한 성능을 보이고 있다.
- 미미한 변화이지만, 레지듀얼 입력을 추가하는 변화로 인해 플레인-꽃 모델보다 정확도를 조금이나마 상승시킬 수 있었다.

5. 보틀넥 블록을 이용하는 보틀넥-꽃 모델로 학습

```
24: full, [1, 1, 256]=>[6] pm:256x6+6=1542
Total parameter count: 190182
Model bottleneck_flower train started:
    Epoch 2: cost=1.189, accuracy=0.512/0.310 (431/431 secs)
    Epoch 4: cost=1.091, accuracy=0.570/0.290 (408/839 secs)
    Epoch 6: cost=0.994, accuracy=0.605/0.520 (410/1249 secs)
    Epoch 8: cost=0.884, accuracy=0.662/0.570 (430/1679 secs)
    Epoch 10: cost=0.852, accuracy=0.672/0.500 (424/2103 secs)
Model bottleneck_flower train ended in 2103 secs:
Model bottleneck_flower test report: accuracy = 0.490, (25 secs)

Model bottleneck_flower Visualization
```

```
추정확률분포 [ 3, 2, 0, 0,91, 4] => 추정 sunflower : 정답 daisy => X
추정확률분포 [ 1, 1, 0, 0, 2,96] => 추정 tulip : 정답 tulip => O
추정확률분포 [ 0,30, 0, 0,69, 1] => 추정 sunflower : 정답 sunflower => O
```

- 보틀넥-꽃 모델은 보틀넥-152 모델 구성에 사용된 매크로 모듈들을 이용해 쉽게 정의할 수 있다.
- 이번 실험의 모델은 합성곱 계층 19개, 풀링 계층 4개, 출력 계층 1개 등 총 24개의 계층으로 구성되어 있으며, 약 19만개의 파라미터를 갖는다.
- 10 에포크 학습시킨 결과 49.0%의 정확도를 확인할 수 있었다. 정확도가 이전의 실험에 비해 약 20%정도 향상된 모습을 볼 수 있었다.
- 학습 횟수를 현재 설정보다 늘린다면 현재 정확도보다 높은 정확도를 얻을 수 있을 것으로 보인다. 그러나 모델의 규모가 크기 때문에 학습 시간이 그만큼 오래 걸릴 것을 감안해야 한다.
- 인셉션 모델, 레스넷 모델 2개의 모델의 꽃 이미지 분류에서 보여주는 품질이 아주 인상적이지는 못하였다. 두 개의 모델 모두 50%가 안되는 성능을 보였으며, 이 것의 원인은 적은 양의 데이터일 것으로 보인다.
- 하지만 신경망의 구조, 하이퍼파라미터 값을 적절히 바꾸어주고 학습 횟수를 늘린다면 이전의 결과들 보다 향상된 결과들을 볼 수 있을 것으로 추측된다.
- 이번 실험을 통해 인셉션, 레스넷 2개의 모델을 사용해 보았으나 확실히 모델의 규모 때문에 학습시간이 기대이상으로 길어졌고 그로 인해 하이퍼파라미터에 변화를 주는 것은 쉽지 않았다. 하지만 위에서 언급한 것처럼 적절한 하이퍼파라미터에 학습횟수를 증가시킨다면 성능을 많이 향상시킬 수 있을 것으로 보인다.