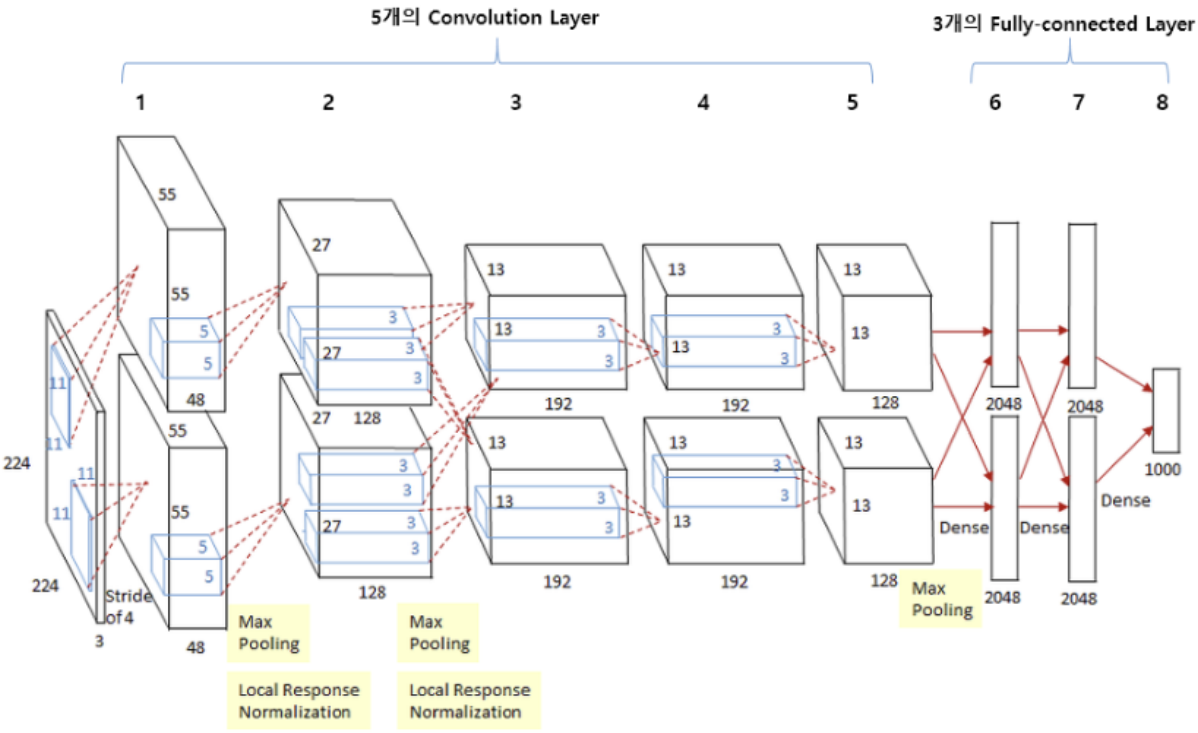
7주차 AlexNet

1. AlexNet의 구조



AlexNet은 LeNet5와 구조가 비슷하다. 가장 큰 변화는 병렬적인 구조로 설계되었다는 점이다.

5개 Conv-layer와 3개의 FC-layer, 총 8개의 layer와 활성화 함수는 ReLU, Pooling은 Max-pooling을 사용했다.

2, 4, 5번째 Conv-layer들은 이전 단계의 같은 채널의 feature-map(출력)과 연결되어 있다. 반면, 3번째 Conv-layer는 전 단계의 두 채널의 feature-map과 연결되어 있다.

1st Layer(convolution)

- **96개의 11x11x3사이즈 filter kernel**로 입력 영상을 convolution해준다. **stride는 4**로 설정, **zero-padding은 사용하지X**
    - Input Image size `227x227x3`
      (227x227사이즈의 RGB 컬러 이미지)

- Number of filters `3`
- Filter size** `11x11x3`
- Stride** `4`

- 그럼 **입력과 커널의 크기**를 통해 **feature map의 크기**를 알 수 있다. (padding포함한 건 추후 등장)

$$O_h = floor(\frac{I_h - K_h}{S} + 1)$$

$$O_w = floor(\frac{I_w - K_w}{S} + 1)$$

- I_h : 입력의 높이
- I_w : 입력의 너비
- K_h : 커널의 높이
- K_w : 커널의 너비
- S : 스트라이드
- O_h : 특성 맵의 높이
- O_w : 특성 맵의 너비

(227-11) / 4 +1 = 55, 11은 kernel로 줄어든 크기, 4는 stride 따라서 결과는

- Layer 1 Output** **`55 x 55 x 96`**(96장의 55x55사이즈 feature map)

- 이 다음 **ReLU함수로 활성화**해준다.
- 이어서 **3x3 overlapping max pooling**이 stride 2로 실행,
  그 결과 27x27x96 feature map을 갖게 된다.
  - (55-3)/2 + 1 = 27 따라서 27x27사이즈 나옴
- 여기에 더해 속도를 높이기 위해 **Local Response Normalization**이 실행된다.
  (feature map의 크기는 27x27x96으로 변함X)

  Local Response Normalization
  - GPU가 2개여서 96 → 48+48이 된다.
  - convolution이나 pooling시 특정 pixel값이 매우 높아 주변에 영향 주는 것을 방지하고자 다른 map의 같은 위치에 있는 pixel끼리 정규화(normalization)을 해줌

2nd Layer(Convolution)

- **128개의 5x5x48 kernel을 2번(128*2=256개)**사용해 이 전 단계의 feature map을 convolution, **stirde=1, zero-padding=2로** 설정
- 따라서 **27x27x256 feature map**을 얻게 된다.
  - 전 단계에서 96개(55x55)의 feature map이 48개씩 GPU 2개로 분산되었기에 5x5x48크기의 kernel을 각각 GPU에 1번씩 총 2번을 적용시킴
  - 결과도 128개씩 2개가 나옴

    - 참고로 padding이 있어 convolution 후 사이즈 변화는 X
- 이 다음 ReLU로 활성화
- 이어서 **3x3 overlapping max pooling이 stride 2**로 실행, 그 결과 **13x13x256 feature map**을 갖게 된다.
- 여기에 더해 속도를 높이기 위해 **Local Response Normalization**이 실행된다. (feature map의 크기는 13x13x256으로 변함X)

3rd Layer(Convolution)

- **총 384개의 3 x 3 x 256 kernel**을 사용하여 전 단계의 feature map(13x13x256)을 convolution해준다.
    - 3번째 layer에서는 각각의 GPU 채널이 **전 단계의 GPU 채널 2개 모두와 연결**되어 있으므로 [하나의 GPU 채널당 13X13X128의 feature map] X 2을 3X3X256의 커널로 convolution.
    - 결과는 하나의 GPU 채널당 192개가 나와서 모두 384개
- stride와 zero-padding 모두 1로 설정, 따라서 13 x 13 x 384 feature map(384장의 13 x 13 사이즈)을 얻게 된다.
- ReLU 함수로 활성화한다.

4th Layer(Convolution)

- **192개의 3 x 3 x 192 kernel**을 각 GPU 채널당 1번씩 2번**(총 384개**)사용하여 전 단계의 feature map(13x13x384)을 convolution해준다.
- stride와 zero-padding 모두 1로 설정, 따라서 13 x 13 x 384 feature map(384장의 13 x 13 사이즈)을 얻게 된다.
- ReLU 함수로 활성화한다.

5th Layer(Convolution)

- **128개의 3 x 3 x 192 kernel**을 각 GPU 채널당 1번씩 2번(**총 256개**) 사용해서 전 단계의 feature map(13x13x384)을 convolution해준다.
- stride와 zero-padding 모두 1로 설정한다. 따라서 13 x 13 x 256 feature map(256장의 13 x 13 사이즈 feature map들)을 얻게 된다.
- ReLU 함수로 활성화한다.
- 다음에 **3 x 3 overlapping max pooling을 stride 2**로 시행한다. 그 결과 **6 x 6 x 256 feature map**을 얻게 된다.

6th Layer(Fully Connected Layer)

- 6 x 6 x 256 feature map을 flatten해줘서 6 x 6 x 256 = 9216 개의 1차원 벡터로 만들어준다.
- 그것을 6th layer의 4096개의 뉴런과 **fully connected layer에 넣는다.**(각 GPU 채널당 2048개)
- 그 결과를 ReLU로 활성화

7th Layer(Fully Connected Layer)

- 4096개의 뉴런으로 구성되어 있다. 전 단계의 4096개 뉴런과 fully connected되어 있다.(각 GPU 채널당 2048개)
- 그 결과를 ReLU로 활성화

8th Layer(Fully Connected Layer)

- 1000개의 뉴런으로 구성되어 있다. 전 단계의 4096개 뉴런과 fully connected되어 있다.
- 1000개 뉴런의 출력값에 softmax 함수를 적용해 1000개 클래스 각각에 속할 확률을 나타낸다.

> 총, 약 6천만개의 파라미터가 훈련되어야 한다. LeNet-5에서 6만개의 파라미터가 훈련되야했던 것과 비교하면 천배나 많아졌다.
> 하지만 그만큼 컴퓨팅 기술도 좋아졌고, 훈련시간을 줄이기 위한 방법들도 사용되었기 때문에 훈련이 가능했다. 예전 기술 같으면 상상도 못할 연산량이다.