📖

ImageNet Classification with Deep Convolutional Neural Networks [AlexNet]

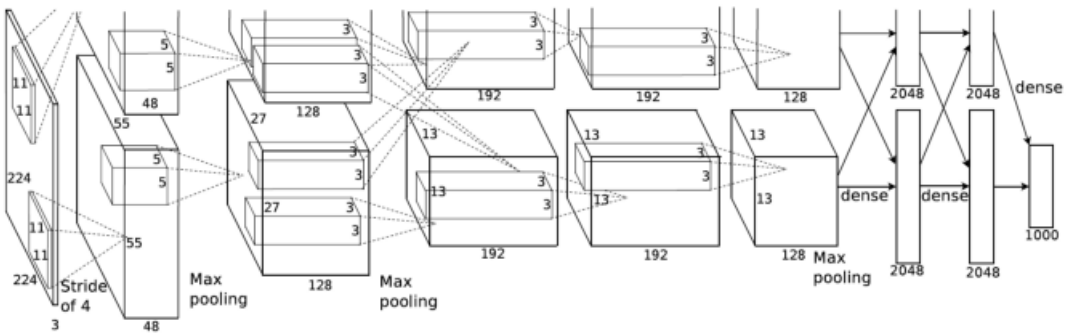
출처

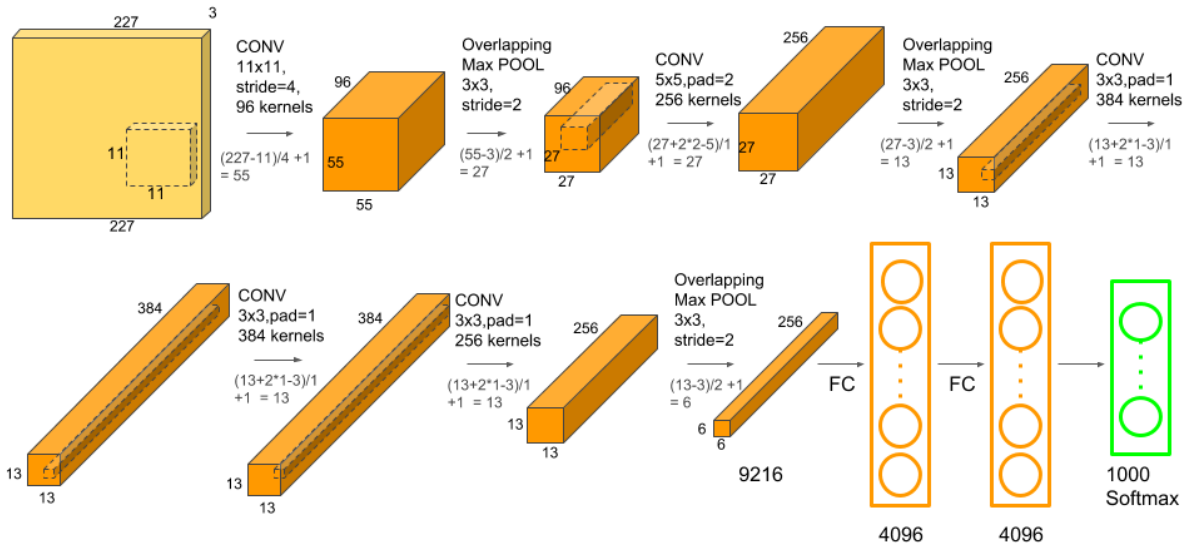
- https://learnopencv.com/understanding-alexnet/
- https://m.blog.naver.com/laonple/220654387455
- https://zl-zon.tistory.com/3?category=995058

ImageNet Classification with Deep Convolutional Neural Networks

AlexNet은 ILSVRC-2012에서 우승한 최초의 CNN 기반 우승 모델이다.

Architecture





AlexNet은 기본적으로 Convolution layer - pooling layer(subsampling layer) - normalization layer 구조가 두 번 반복되며 마지막에는 FC-layer가 몇개 붙어서 classification task를 진행한다.

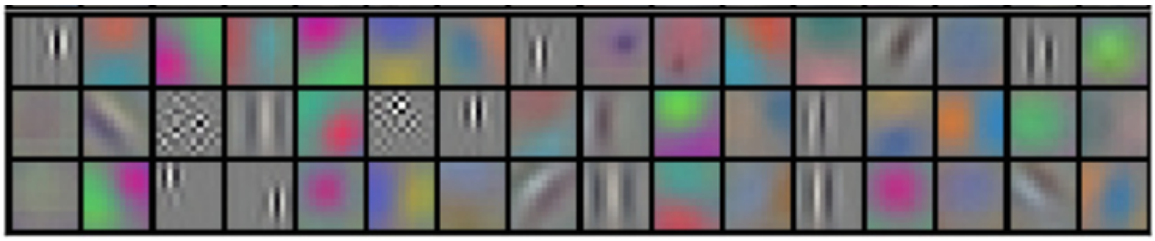
LeNet-5와 거의 비슷한 구조를 가지지만 크기가 너무 방대하다

LeNet의 경우 입력 Image의 크기가 1x32x32여서 모든 Convolution layer는 5x5 크기의 kernel을 사용했고, input channel(depth)도 1이였다. 이후 convolution 연산을 거치면서 depth가 증가하게 된다.

하지만 AlexNet의 경우 입력 Image의 크기가 3x 227x 227으로 매우 커서 첫 Convolution layer의 kernel의 size는 3x11x11로 굉장히 큰 receptive field를 사용하고 있다. 첫번째 convolution layer에서 stride 크기를 4를 적용하고 channel을 96개로 생성하므로 최종적으로 55x55x96의 형태의 1번째 Feature Map이 생성된다. 보시다시피 이렇게 매우 많은 Parameter들이 있어 예전 GPU 1개로는 이를 감당하기 힘들어서 2개의 GPU로 Parallel하게 연산을 쪼개서 진행한다.

GPU-1(위쪽)에서는 주로 컬러와 상관없는 정보를 추출하기 위한 kernel이 학습되고,
GPU-2에서는 주로 color에 관련된 정보를 추출하기 위한 kernel이 학습된다.

▼ 예제 Image



2번째 Convolution layer는 5x5x48 크기의 kernel을 사용하고 있으며 Stride는 첫번째 convolution layer와 달리 1이다. 이후 response normalization과 maxpooling 과정을 거쳐 Image크기를 256 x 27 x 27로 줄어들게 되며 256 x 3 x 3Kernel을 사용해 Convolution연산을 수행하고 최종적으로 384개의 Feature Map을 얻는다.

이때 GPU-1과 GPU-2의 결과를 모두 섞어 사용한다.

그 결과에 대해 normalization과 pooling을 거쳐 13 x 13 x 384 크기의 Image을 얻으며 이후 다시 384x3x3 kernel에 의해 256x13x13 Feature map이 만들어지고 이후 Normalization과 Pooling을 통해 256x6x6(9216) Feature Map이 만들어진다.

마지막 convolution layer를 통과한 Feature Map을 (아마) Flatten 해서 4096개의 Output을 가지는 Linear Layer를 통과해 Classification을 진행하고 최종적으로 1000개의 종류를 Classification하기위해 1000개의 Output을 가지는 Linear Layer로 마무리 합니다. 이후 Category에서 결과를 낼 수 있도록 softmax 함수가 적용이 됩니다.

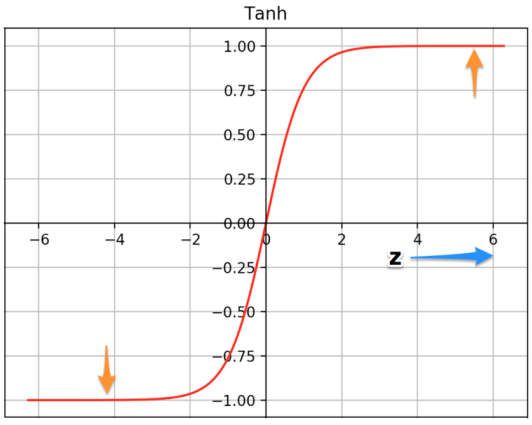
ReLU(Rectified Linear Unit)

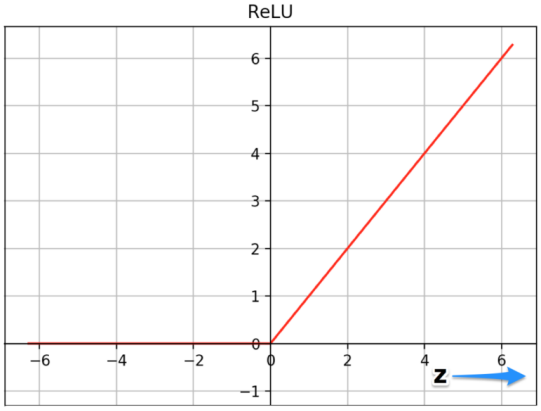
일반적으로 activation function으로 tanh, sigmoid를 사용하는 것과 달리 AlexNet에서는 activation function으로 ReLU를 사용한다. 기존의 tanh, sigmoid(Saturating activation function의 예시)의 경우 gradient vanishing의 문제가 있다. 이러한 문제를 ReLU를 사용함으로써 해결하고, 학습속도 또한 tanh, sigmoid 보다 훨씬 빠르다. tanh나 sigmoid 보다 약 6배 정도 빠르게 accuracy를 달성했다.

ReLU는 tanh,sigmoid 처럼 Non-linearity 라는 특성도 가지고 있다.

▼ example Image

The tanh function saturates(포화한다, 수렴한다) at very high or very low values 여기서의 gradient의 값이 0에 수렴하기 때문에 gradient descent로써 학습하는 양이 매우 적다. 그러나 ReLU는 그에 비해 0에 가깝지 않다.





Training on Multiple GPUs

이 팀이 사용한 GTX 580의 경우 memory가 3GB밖에 되질 않아서 GTX 580 2개를 parallel하게 사용함.

Local Response Normalization (LRN)

ReLU는 max(a,0)과 같은 형태인데, 양수 방향의 값은 그대로 사용하기 때문에 특정 activation map의 pixel값이 엄청나게 크다면 주변의 pixel도 영향을 받게 된다. 이러한 문제를 해결하기 위해 activation map의 같은 위치에 있는 픽셀 끼리 정규화하는 방식인 LRN을 사용한다. 현재는 이러한 정규화 방법대신 Batch Normalization을 주로 사용한다.

Overlapping Pooling

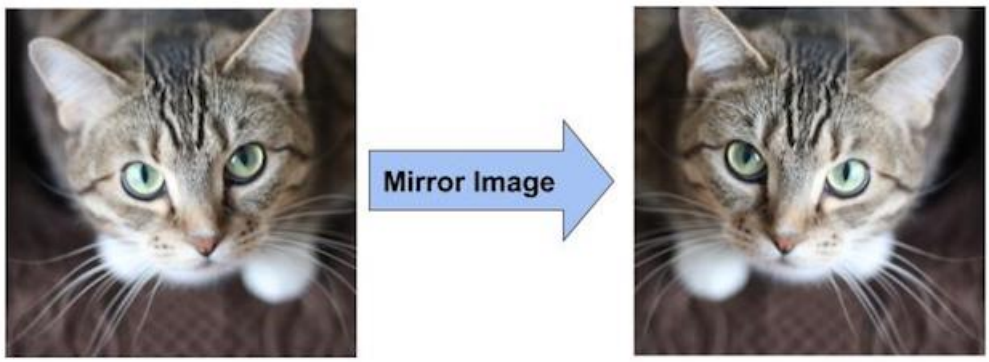
일반적인 pooling layer에서는 겹치지 않게 구성되는 반면, AlexNet의 pooling layer에서는 stride보다 더 큰 필터를 사용하여 겹치게 구성한다. 저자는 Pooling window의 크기는 3x3인데, Stride는 2로하여 3칸이 2칸밖에 넘어가지 않아 매 1칸씩 겹치게 Pooling 된다.

stride가 2 Pooling window의 크기를 2x2로 중첩되지 않은 Pooling window를 사용했을 때의 top-1 error rate 는 0.4% top-3 error rate 는 0.3%로 오류율을 줄이는데 도움이 되었다(?)

Reducing Overfitting

1. Data Augmentation

    ▼ example



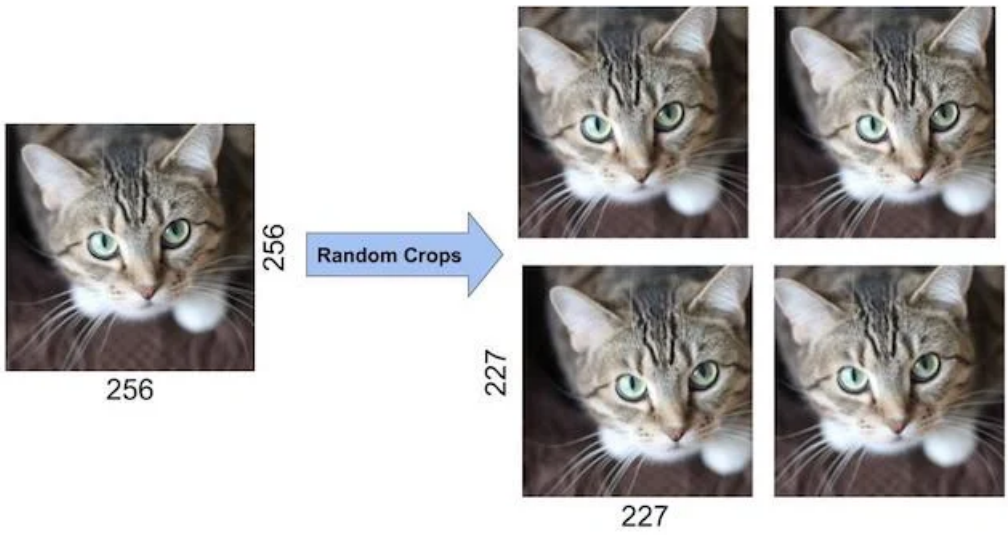
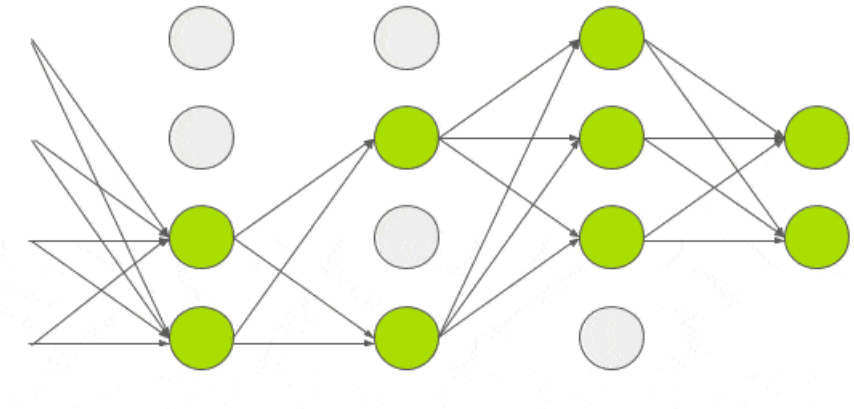


Image data에서 Overfitting을 줄이는 가장 쉽고 일반적인 방법은 label-preserving transformations을 이용하여 dataset을 인위적으로 늘려주는 것. 여기서는 2가지 방법으로 Data Augmentation을 시행하여 2가지 방법 모두 매우 적은 연산을 필요로 하므로 disk에 저장될 필요가 없고 image batch가 GPU에서 train되는 동안, CPU에서 그 다음 Image Batch의 Augmentation이 진행된다.

1. 256x256 원본 Image가 상하 대칭을 포함해 224x224의 patch가 되도록 무작위로 뽑아낸다.
2. ImageNet training set의 RGB pixel 값에 대하여 PCA를 수행한다. (normalize의 형태가 아닐까)

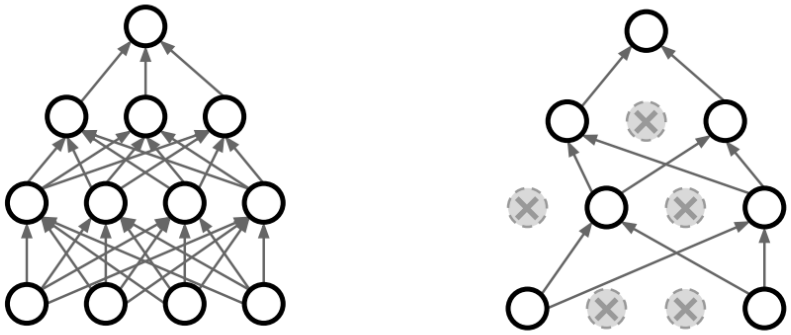
2. DropOut

▼ sample Image



Regularization: Dropout

In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common



Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014

위 그림처럼 일부 Hidden layer의 neuron 의 weight들의 output에 대한 확률을 0을 변경하는 것을 말함(논문에서는 Dropout rate는 50%)

Drop out으로 인해 0으로 변경된 node들은 foward 혹은 backward에 사용되지 않는다. 이로인해 edge와 edge 사이의 복잡한 상호 의존성을 감소시킨다.

단, dropout 사용시 학습 속도가 느려진다.