AlexNet (2012)

2021-12-04

Abstract

- 효과 : LSVRC-2010 contest 에 그 당시 가장 최근보다 잘함
  - top-1 error rate : top 1이 target 인지
  - Top-5 error rate : top 5중에 target이 있는지
- 구조 : 6천만 param, 65만 뉴런, 5개 Conv layer, max-pooling layers, 3개 FC layer(1000개의 softmax)
- 빠른 학습 : non-saturating neurons, Conv 연산에서 효율적으로 gpu 사용
- Overfitting 방지 : Dropout

1. Introduction

- 최근에 큰 dataset 수집이 가능해짐
- CNN은 connection과 param이 적어서 train 하기 쉽다(?)
- GPU가 좋아져서 train하기 편해짐
- 5 Conv, 3 FC – Depth가 중요하다
- 현재 Network 사이즈는 GPU의 한계로 한계가 있다.

2. Dataset

- ImageNet : 1.5천만 이미지, 2.2만 category
- 크기가 제각각 -> 256 x 256으로 만듦
  - 짧은 변 256으로 resize, 나머지는 crop
- 전처리 : 모든 training image에서 평균을 뺌(위치별? 모든 픽셀?)

3. Architecture
- ReLU

- ReLU는 tanh보다 학습이 빠르게 된다. Non-saturating, nonlinearity
  - Saturation은 입력신호의 총합이 크거나 작을 때 기울기가 0에 가까워지는 것 -> vanishing gradient (변화가 느려지는 것?)
  - Nonlinearity : 출력된 결과에 반응하도록 기울기가 변하는 함수 사용

3. Architecture – multiple GPUs

- 당시 GPU : 3GB, cross-GPU parallelization(gpu끼리 바로 read, write)
- 따라서 GPU 2개를 사용하고, 네트워크 크기 키움
- 특정 layer에서만 GPU communication함(layer 3은 두 gpu에서 input 받음, layer 4는 같은 gpu에서 input 받음?)
- 네트워크 크기를 키워서 error rate을 크게 줄임
- 경미하게 시간도 줄임

3. Architecture
– Local Response Normalization

- RelU는 saturation 방지를 위한 normalization 할 필요가 없다.
- 하지만 일반화를 위해 normalization 함.
- N개의 인접한 kernel map의 activity로 계산

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

- 몇개의 layer에서 ReLU 다음으로 사용
- Brightness normalization (mean activation을 빼지 않았기 때문?)
- Error Rate 줄임

3. Architecture – Overlapping Pooling

- Pooling layer는 인접 뉴런들의 결과를 summarize 하는 기능
- Overlapping 해서 overfitting 줄임

3. Architecture
– Overall Architecture

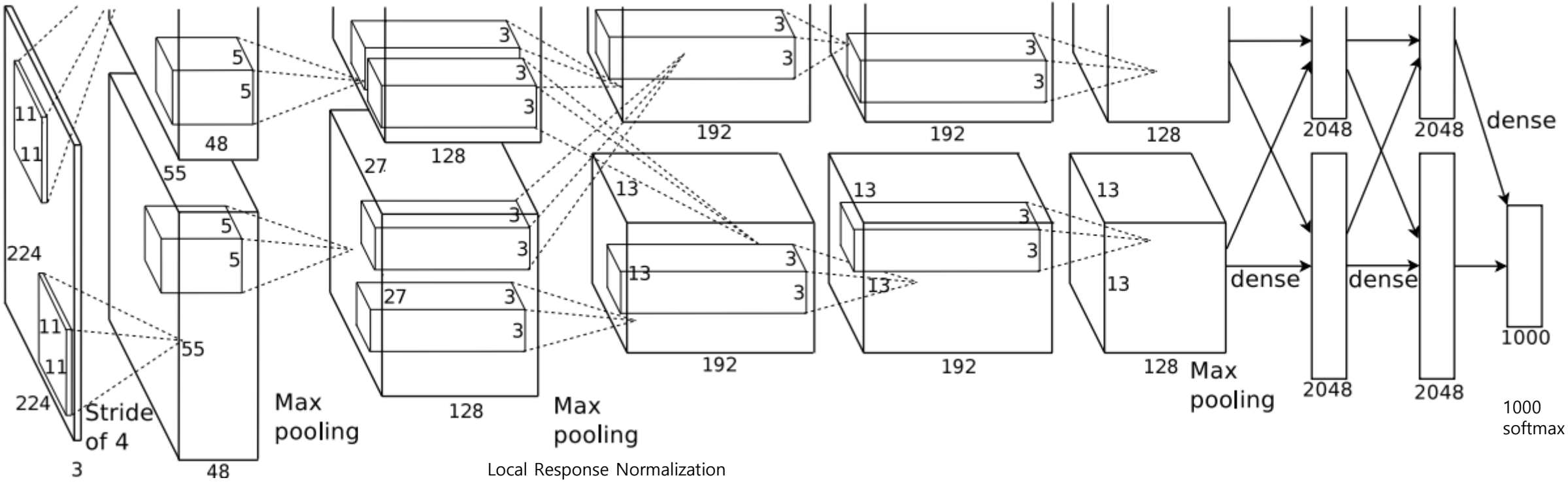
11x11x3의 96 kernels
Stride는 4

5x5x48의 256 kernels

3x3x256의 384 kernels

3x3x192의 384 kernels

3x3x192의 256 kernels



Local Response Normalization

Local Response Normalization
GPU parallelization 사용
3layer는 2layer의 모든 커널과 연결

Conv, FC 모두 ReLU 사용

4. Reducing Overfitting - Data augmetation

- Overfitting 줄이기 위해 2가지 방법으로 data augmentation함
- 256x256 패치에서 224x224만큼 자르고, horizontal 반전해서 train
- 224x224 4 코너와 중심 패치 이렇게 5개 패치를 만들고 horizontal 반전해서 10개 패치로 test 하고 평균냄
- RGB pixel을 PCA해서 eigenvector와 eigenvalue로 나누고, 픽셀마다 random variable을 다르게하여 계산후 픽셀별로 더함
  - Illumination 효과로 error rate 줄임

4. Reducing Overfitting - Dropout

- Dropout 사용해서 뉴런의 co-adaptations 줄임
  - Co-adaptation : 한 뉴런이 다른 뉴런의 영향을 크게 받는 것
- 2 FC layer에 사용
- Test 할때는 결과에 0.5를 곱함

5. Details of learning

- SGD : mini-batch 사용
- Weight는 zero-mean으로 초기화
- Bias는 0과 1로 초기화
- 모든 layer의 learning rate 같게, validation error가 변화가 없으면 10 나누기
- 90 cycles, 5-6 days 학습

6. Results

- Top-1 error 37.5, Top-5 error 17.0
- Validation error와 test error는 큰 차이가 없었다.
- 정성적 평가 : 중심에서 벗어난 이미지도 잘 추론, top-5 label도 reasonable(leopard, jaguar, cheetah)
- Feature activation으로 L2 거리로 유사한 사진들을 모아봤는데 비슷함 -> autoencoder와 같은 효과

7. Discussion

- Depth가 중요하다
- 간단한 실험을 위해 Unsupervised pretraining을 사용하지 않음
- Network를 크게 하고, 학습을 오래했지만, 사람의 짧은 시각적 판단력을 따라가기엔 부족함

궁금한 부분

cropped out the central 256×256 patch from the resulting image. We did not pre-process the images in any other way, except for subtracting the mean activity over the training set from each pixel. So

픽셀 위치별 평균? 전체 픽셀별 평균?

We are not the first to consider alternatives to traditional neuron models in CNNs. For example, Jarrett et al. [11] claim that the nonlinearity $f(x) = |\tanh(x)|$ works particularly well with their type of contrast normalization followed by local average pooling on the Caltech-101 dataset. However, on this dataset the primary concern is preventing overfitting, so the effect

Tanh 의 절대값으로 한 이유가 뭘까?
그리고 이걸로 했을때 왜 overfitting을 막을 수 있지?

before training begins. This sort of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels. The constants k, n, α, and β are hyper-parameters whose

(해결) response 정규화가 측면 억제를 구현한다.
하나의 뉴런의 활동이 이웃한 뉴런들의 활동에 의해 감소되는 것

궁금한 부분

This scheme bears some resemblance to the local contrast normalization scheme of Jarrett et al. [11], but ours would be more correctly termed "brightness normalization", since we do not subtract the mean activity. Response normalization reduces our top-1 and top-5 error rates by 1.4% and 1.2%,

평균을 빼지 않으면 왜 Brightness normalization?

a distribution over the 1000 class labels. Our network maximizes the multinomial logistic regression objective, which is equivalent to maximizing the average across training cases of the log-probability of the correct label under the prediction distribution.

해석이 안되는 문장?

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random 224×224 patches (and their horizontal reflections) from the 256×256 images and training our network on these extracted patches[4]. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly inter-

어떻게 training set이 2048배 늘은거지?

궁금한 부분

values, respectively, and α_i is the aforementioned random variable. Each α_i is drawn only once for all the pixels of a particular training image until that image is used for training again, at which point it is re-drawn. This scheme approximately captures an important property of natural images,

모든 픽셀에 같은 값을 적용한거고, 재사용 없이 한번만 사용했다고 이해해도 되나?

but all these architectures share weights. This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to

co-adaptation
어떤 뉴런이 다른 특정 뉴런에 의존적으로 변하는 것.
신경망의 학습 중, 어느 시점에서 같은 층의 두 개 이상의 노드의 입력 및 출력 연결강도가 같아지면,
아무리 학습이 진행되어도 그 노드들은 같은 일을 수행하게 되어 불필요한 중복이 생기는 문제를 말한다.
즉 연결강도들이 학습을 통해 업데이트 되더라도 이들은 계속해서 서로 같은 입출력 연결 강도들을 유지하게 되고
이는 결국 하나의 노드로 작동하는 것으로써, 이후 어떠한 학습을 통해서도 이들은 다른 값으로 나눠질 수 없고
상호 적응하는 노드들에는 낭비가 발생하는 것이다. 결국 이것은 컴퓨팅 파워와 메모리의 낭비로 이어진다.

드랍아웃은 이러한 상호적응 문제를 해소한다.
즉, 드랍아웃이 임의로 노드들을 생략할 때 이러한 상호 적응 중인 노드들 중 일부는 생략하고 일부는 생략하지 않게 되므로 학습 중 상호 적응이 발생한 노드들이 분리될 수 있어서 상호 적응 문제를 회피할 수 있게 된다.

궁금한 부분

tors is inefficient, but it could be made efficient by training an auto-encoder to compress these vectors to short binary codes. This should produce a much better image retrieval method than applying auto-encoders to the raw pixels [14], which does not make use of image labels and hence has a tendency

이때 사용한 모델을 Auto-encoder 처럼 사용할 수 있을까?
Feature를 뽑아내는 네트워크로 사용할 수 있을까?

느낀 점

- GPU의 메모리가 작을 때여서 복잡한 네트워크를 만들어야 했던 것이 느껴졌다.
- Data augmentation 방법들이 신기했다. PCA 사용하는 방법이 특히 신기했다. 원래도 많이 쓰이는 방법인지 궁금하다.
- 실험할때 어디까지 세팅을 직접 컨트롤 해야하는지에 대해 생각해봐야 할 거같다.
- 평가 방법도 신기했다. Feature activation으로 L2 Distance를 비교해서 유사하다고 판단되는 이미지를 정성적으로 평가하는게 신기했다.
- Dropout 후에 추론할때 0.5를 곱해서 하는 부분도 처음 알았다.
- 이미지의 특징을 뽑아내는 것을 하고 싶었는데, 이렇게 학습한 모델로 할 수 있을지 궁금해졌다.