AlexNet

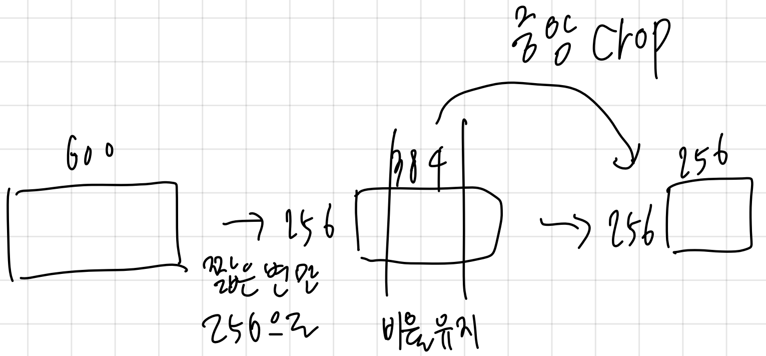
0. Abstract , 2. The Dataset

1. 이미지 종류 : ImageNet LSVRC-2010 의 120만장 / 1000 labels
2. 결과 : top-1 error 37.5% (모델이 가장 확률이 높다고 예측한 클래스가 정답이 아닌 비율)
   -5 17% (모델이 상위 5개로 예측한 클래스 중 정답이 없을 비율)

2012 top5 15.3%

3. 파라미터 : 6천만개
   뉴런 : 65만개
   Conv+ maxpool : 5개
   fc : 3개 (마지막 1000차원 softmax)
   activation func : ReLU
   Dropout

이미지 : 256X256 ⇒



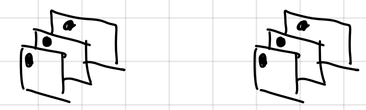
→ mean subtraction : 모든 이미지에서 각 동일한 위치 (x,y,c)를 평균하고 각 픽셀에서 빼줌 ⇒ "224 X 224" (참고 **4.1 Data Augmentation**)

$\mu_R = (R_1 + R_2)/2$ ⇒ $R_1' = R_1 - \mu_R$, $R_2' = R_2 - \mu_R$

μ_G 〃

μ_B 〃

R_1, G_1, B_1 R_2, G_2, B_2



1. Introduction

1. 한계 : 데이터셋 ↓ → 과적합 방지
   해결 데이터셋↑
   label-preserving 변환 : 증강기법

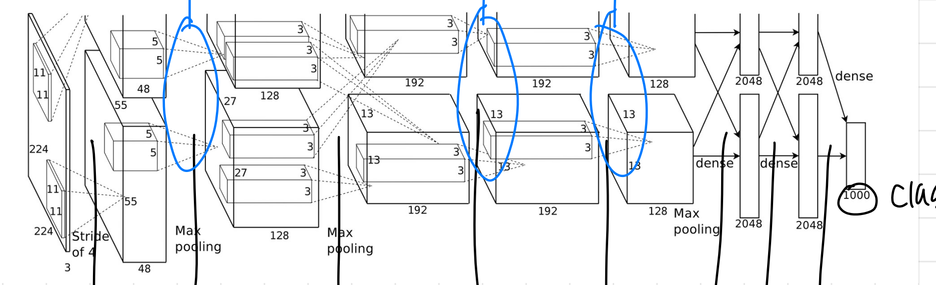
but, → 머신러닝 : 높은 learning capacity X
⇓
CNN : HXW로 용량 제어 통계적 정상성
올바른 가정 (stationarity of statistics : 각 패턴은 위치가 어디든 특성이 달라지지X
& locality of pixel dependency) : 근처에 있는 픽셀끼리 밀접 연관
픽셀 의존성의 국소성
⇓
단순 fc에 비해 연결수 파라미터 수↓, 성능↑

3. The Architecture

1. 전체구조 overall Architecture

같은 gpu 연결된 커널맵에만 연결시킴

← 모든 conv, fc 뒤에는 ReLU



Class 수

1 2 3 4 5 6 7 8

8 → 3번째 fc = softmax (각 클래스별 확률 분포)

각 뒤에 LRN 5개 conv
+
max pooling

모든 커널맵에 연결

5 → 뒤에 maxpooling

6 7 8 → 3 fc → 당연히 모든 이전 뉴런과 연결

logits 생성

$$P_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

→ 8개 학습가능 계층

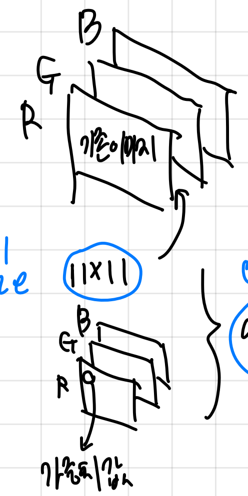
(파라미터는 다음과 같이 계산)

↳ ex) 224x224x3

필터 96 크기 11x11x3

⇒ $96 \times (11 \times 11 \times 3) + 96(\text{bias}) \approx 34k$

fc) 4096 x 4096 + 4096
in, out, bias

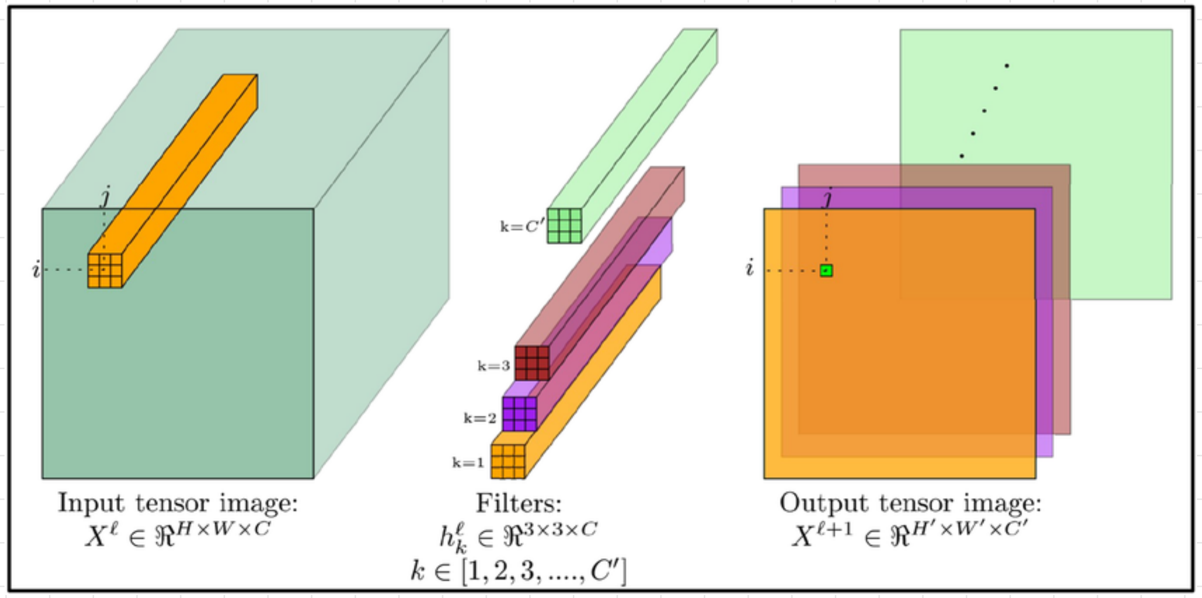


kernel_size: 11x11

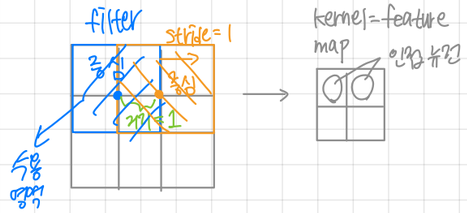
out_channels: 96개

⇒ 3차원 텐서 이미지에 3차원 필터를 씌워서 featuremap 생성

→ 이걸 서로 다른 96개 필터로 96개 f.m 생성



→



커널맵에서 인접 뉴런간 = 커널 개수
↑ 수용 영역의 중심간거리 = stride

← 커널맵 각 원소가 뉴런이고

출력 차원

	Layer	Img/f.m size	kernel_size	out_channels	stride	padding	
150528	Input	224×224×3					
253440	conv1	55×55×96	11×11×3	96	4	2	+LRN +ReLU
	max pooling	27×27×96	3×3×96		2		+ReLU
186624	conv2	27×27×256	5×5×96	256	1	2	+LRN +ReLU
	max pooling	13×13×256	3×3×256		2		+ReLU
64896	conv3	13×13×384	3×3×256	384	1	1	+ReLU
64896	conv4	13×13×384	3×3×384	384	1	1	+ReLU
43264	conv5	13×13×256	3×3×384	256	1	1	+ReLU
	max Pooling	6×6×256	3×3×256		2		+ReLU
	flatten	6×6×256=9216					
4096	fc1	9216 → 4096					+dropout
4096	fc2	4096 → 4096					+dropout
num_classes	fc3	4096 → num_classes					

$\frac{224+2\times2-11}{4}+1=[55.25]=55$

$\frac{55-3}{2}+1=27$

$\frac{27+2\times2-5}{1}+1=27$ (원래는 GPU 2개 여서 128/48

$\frac{27-3}{2}+1=13$

$\frac{13+2\times1-3}{1}+1=13$

$\frac{13+2\times1-3}{1}+1=13$ 192/192

$\frac{13+2\times1-3}{1}+1=13$ 128/192

$\frac{13-3}{2}+1=6$

→ 입력 차원 : 224×224×3=150528

2. ReLU (비선형성) : max(0,x)

→ 경사하강법 시 tanh, sigmoid 등 포화 비선형은 속도 느림 // ⊕ |tanh(x)| but, 과적합 방지 해결일 뿐, 빠른 학습 능력은 X

(GPU)

3. Local Response Normalization (국소 응답 정규화) : LRN // ⊕ LCN (국소 대비 정규화)

→ ReLU는 포화방지를 위한 입력정규화 필요X (positive 입력이면 학습 O)

→ ReLU 후에 local normalization 해주면 일반화 성능 ↑

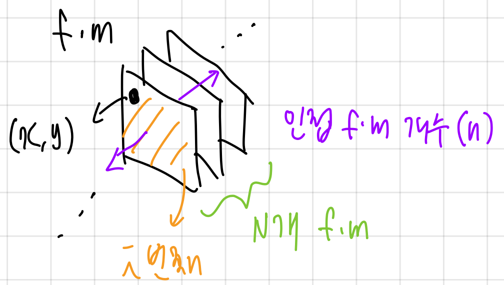
→ $$b^i_{x,y} = a^i_{x,y} \Big/ \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2\right)^\beta$$

ReLU 후의 N개 f.m 중 i번째 f.m 에서의 위치 (x,y)

△ : hyper-parameter
- k = 2
- n = 5
- α = 10⁻⁴
- β = 0.75

설정

⇒ top 1 1.4% ↓
top 5 1.2% ↓



4. Overlapping Pooling (중첩 풀링)

→ Pooling : 동일 피처맵(커널 맵)의 요소인 뉴런들에 대해 인접 뉴런들의 출력을 요약,
= 같은 필터에서 나온 피처맵

보통 풀링이 요약하는 영역이 겹치지 않도록 stride와 z (윈도우 크기)를 동일하게
(stride: 몇 칸씩 이동 / z: 한 번에 보는 크기 (z by z))

→ 중첩 : s < z
(s = 2, z = 3 설정)

↳ s = 2, z = 2 (비중첩) 일 때와 출력크기 동일하되,

top 1 0.4% ↓
top 5 0.3% ↓
과적합 ↓

4. Reducing overfitting (과적합 감소)

① label-preserving transformation 2가지

1. translation + horizontal reflection

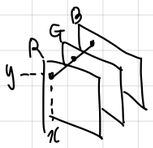
256 × 256
↓
224 × 224 패치 무작위 추출
(transforms.RandomCrop(224))

↳ transforms.RandomHorizontalFlip(P) : 전체 훈련 데이터셋 중 P 확률만 수평반전 (좌우반전)

→ test 시에도 사용 : 각 원본당 224 × 224 패치 5개 (모서리 4개 + 중앙 1개) + 각 좌우반전 ⇒ 각 총 10개 패치
↓
각 패치에 대한 softmax 후 평균

2. RGB 채널 강도 조정 → top1 1% 감소

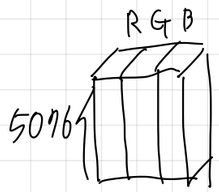
↳ 물체 정체성(identity)이 조명 강도나 색상 변화에 불변

→ 

Inner img

⇒ $I_{xy} = \begin{bmatrix} I_{xy}^{R} \\ I_{xy}^{G} \\ I_{xy}^{B} \end{bmatrix}$ 가 224×224 개 생성 (1개 img당) → N개 훈련 이미지

⇒ $(50176 \times N) \times 3$ (2차원 행렬)



N개 ⋮ (계속 50176 뒤에 50176개씩 이어붙임)

↓ 공분산 행렬

1) 각 채널 별로 M_R, M_G, M_B 구하고 각 열별 요소들에서 빼주기 ⇒ X 행렬

2) $\frac{1}{N} X^T \cdot X = \left[3 \times (50176 \times N)\right] \times \left[(50176 \times N) \times 3\right] = 3 \times 3$ 공분산 행렬 = Σ

3) PCA 수행: P(고유벡터), λ(고윳값)

↳ $\Sigma P = \lambda P$ 수행 ($\det(\Sigma - \lambda I) = 0$ 구하면 $\lambda_1, \lambda_2, \lambda_3$ 나오고 $\Sigma P = \lambda P$를 P구하기)

↓

$[P_1 \ P_2 \ P_3] \begin{bmatrix} \alpha_1 \lambda_1 \\ \alpha_2 \lambda_2 \\ \alpha_3 \lambda_3 \end{bmatrix}$ 의 값을 모든 각 픽셀에 더해줌

↓

α = 가우시안 분포에서 뽑은 난수

$\mu = 0, \sigma = 0.1$

② dropout : 순전파 시 일부 뉴런 끄기 (역전파로 랜덤히 사용x)

→ 서로 뉴런끼리 지내온 의존 방지, 매번 다른 구조의 (서브) 네트워크 = 앙상블 (단, 가중치는 같은 연결이라면 다른 네트워크여도 공유) / 단, 수렴까지 반복 횟수 2배↑

→ test: 모든 뉴런을 쓰되, 출력값에 ×0.5 → 앙상블의 (기하) 평균 효과 내기 위해서

5. Details of learning

1. Optimizer : SGD → optim.SGD(model.parameters(), lr, momentum, weight_decay)

→ momentum: 0.9 / weight-decay : 0.0005

↳ "관성": GD의 가파른 경사에서 진동하는 문제 해결하기 위해 이전 기울기의 이동방향 기억해서 현재 기울기에 더해서 업데이트 (이전 방향대로 계속해서)

↳ 과적합 막기 위한 Regularization (weight 커지는 것 막기)

: $loss(y,\hat{y}) + \lambda \sum_i w_i^2$ (loss 줄이는게 목표니까)

(training error도 줄어듦)

⇒ 최종 weight update 규칙: $W_{i+1} = W_i + v_{i+1}$

$= 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot W_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w}\Big|_{W_i} \right\rangle_{D_i}$

↳ i : iteration

v_i : momentum 변수

ϵ : lr

D_i : i번째 미니배치

$E(x,y) \in \left\langle \frac{\partial L}{\partial w}\Big|_{W_i} \right\rangle_{D_i}$: 해당 배치에 대한 손실함수 기울기

↳ 가중치 초기화: $\mu=0, \sigma=0.01$ (가우시안)

→ bias: conv 2,4,5 + fc 1,2,3 모두 1로 초기화해서 $y = W^T x + b$ 의 값이 음수가 되어서 ReLU=0 이 되는 문제 (ReLU 미분 $x>0 \to 1$, $x<0 \to 0$)를 막음

→ 학습 초기에 수렴 가속

(바이어스: 0)

2. batch size: 128 / epochs = 90 / lr = 0.01 (학습 계속 x (plateau), $\frac{1}{10}$씩 감소, 3번 감소)

6. Results

7. Discussion

→ depth가 성능 댚성에 핵심