

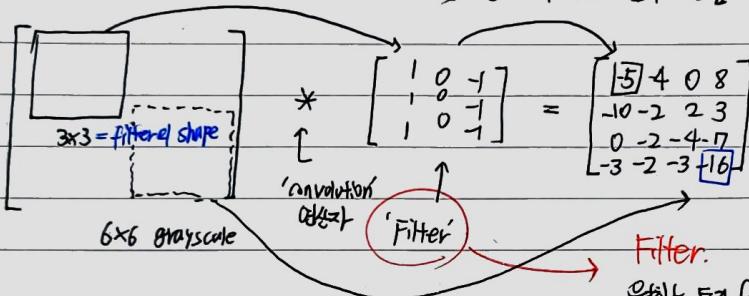
<1주차 Computer Vision>

- 1000×1000 image: (at fully connected layer)
 - $1000 \cdot 1000 \cdot 3 = 3 \text{ million} \Rightarrow A^{10}$
 - 1st hidden layer $\rightarrow 1000 \times 3$ 노드라면, $[1000, 3m]$ dimensional matrix.
- Using convolution layers to avoid expensive computing.

▷ Edge Detection

- Vertical / horizontal edge 검지.

element-wise 곱의 모든합



Filter:

유하는 특징 (feature)이 테이터에 존재하는지

검출해주는 함수!

- Vertical edge case)

bright dark

$$\begin{bmatrix} 10 & 0 & 10 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 10 & 10 & 10 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 6 & 30 & 30 & 0 \end{bmatrix}$$

상대적으로
밝은

'vertical edge'

* bright \rightarrow dark: edge는 (+) 값dark \rightarrow bright: edge는 (-) 값, abs() 사용.

- horizontal

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \leftarrow \text{bright}$$

$$\leftarrow \text{dark}$$

- Sobel filter • Scharr filter

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix}$$

- 이 filter의 속성을 직접 정할 필요 없이, weight로 추구하여
data로부터 자동으로 학습시킬 수 있음.

$$6 \times 6 \xrightarrow{\text{padding}} 8 \times 8 \xrightarrow{\text{padding}} 6 \times 6 : \text{이미지 보존!}$$

★ $n \times n$ img 가 $f \times f$ filter, padding P 고 conv 될 때, $\Rightarrow (n+2P-f+1, n+2P-f+1)$ matrix.
 "Same" convolution $P = (f-1)/2$

f 는 홀수 (center 가 있어야 하므로).

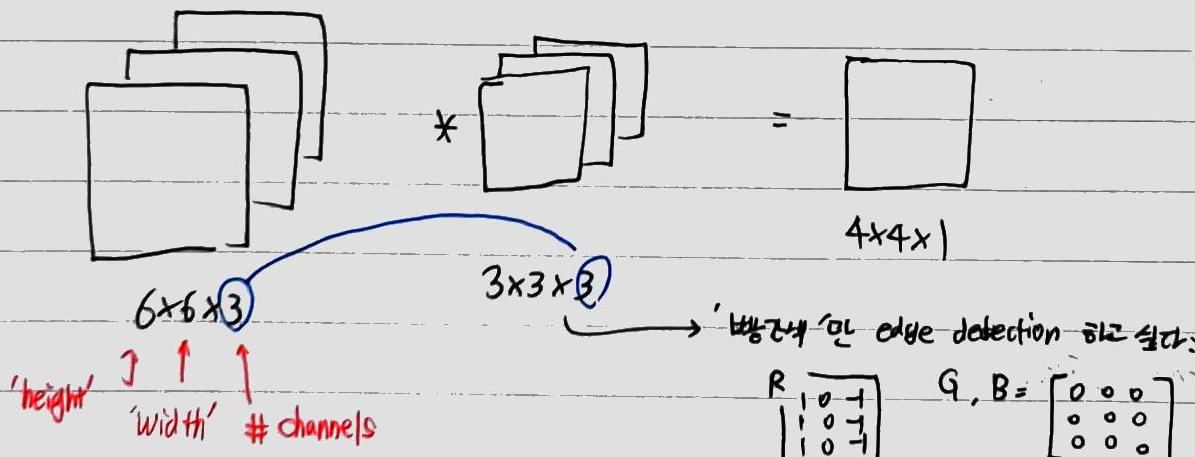
D Strided convolution

Stride S = 한 열선에 이동할 (jump) pixel의 수.

Image $n \times n$ Padding P filter $f \times f$ stride S	$\rightarrow [(\underline{n+2P-f})/S + 1, (\underline{n+2P-f})/S + 1]$ matrix ↓ 정수가 아닐 경우, 내림 (floored) $\lfloor Z \rfloor = \text{floor}(Z)$
--	--

D 3가지로 Volumetric conv

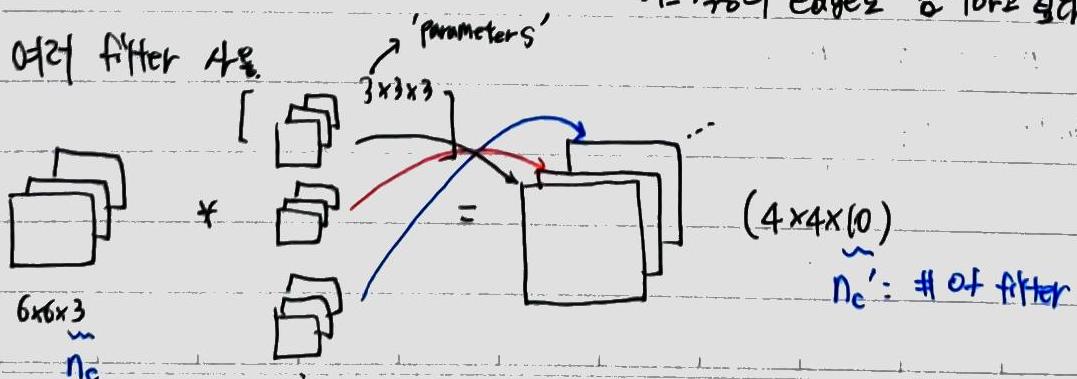
- On RGB images



$$R = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \quad G, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

어느 쪽의 edge도 감지하고 싶다: $R, G, B = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$

- 여러 filter 사용



▷ 1^층 conv network

- Input image: $6 \times 6 \times 3$ # $a^{[0]}$
- 10 Filters: $3 \times 3 \times 3$ # $w^{[0]}$
- Result Image: $4 \times 4 \times 10$ # $W^{[0]} a^{[0]}$
 - Add bias: $b = (10 \times 1)$, $W^{[0]} a^{[0]} + b = (4 \times 4 \times 10)$
- Apply ReLU: $A^{[1]} = \text{ReLU}(W^{[0]} a^{[0]} + b)$
 - parameters: $W, b - (3 \times 3 \times 3 + 1) \times 10 = 280$ 개, Input 3개의 상관계수.

▷ Notation

$f^{[l]}$: filter size

$p^{[l]}$: padding

$s^{[l]}$: stride

Input: $n_H^{[L-1]} \times n_W^{[L-1]} \times n_C^{[L-1]}$ $\leftarrow n^{[l]} = \left\lfloor \frac{n^{[L-1]} + 2p^{[L]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$

Output: $n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$ # channel =

Each filter is: $f^{[l]} \times f^{[l]} \times n_C^{[L-1]}$ 이 레이어에서 쓰는 필터 수

Activations: $a^{[l]} : n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$, Output 크기 일

($A^{[l]} : m \times n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$) # At minibatch training

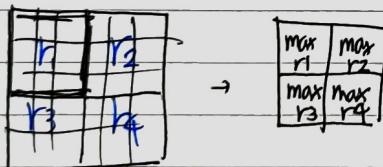
Weights: $f^{[l]} \times f^{[l]} \times n_C^{[L-1]} \times n_C^{[l]}$ # 1개의 filter \times 총 filter 수

bias: $n_C^{[l]} - (1, 1, 1, n_C^{[l]})$ Parameters: $(f^{[l]} \times f^{[l]} \times n_C^{[L-1]} + 1) \times n_C^{[l]}$

▷ Pooling layer

◦ Max pooling

- Input을 몇개의 영역으로 나누어 각 region의 최댓값만으로 층소.



$4 \times 4 \rightarrow 2 \times 2 : f=2, s=2, p=0$ (hyperparameters)

- hyperparameter를 우선 설정하면, parameter는 학습되지 않음.

◦ Average pooling

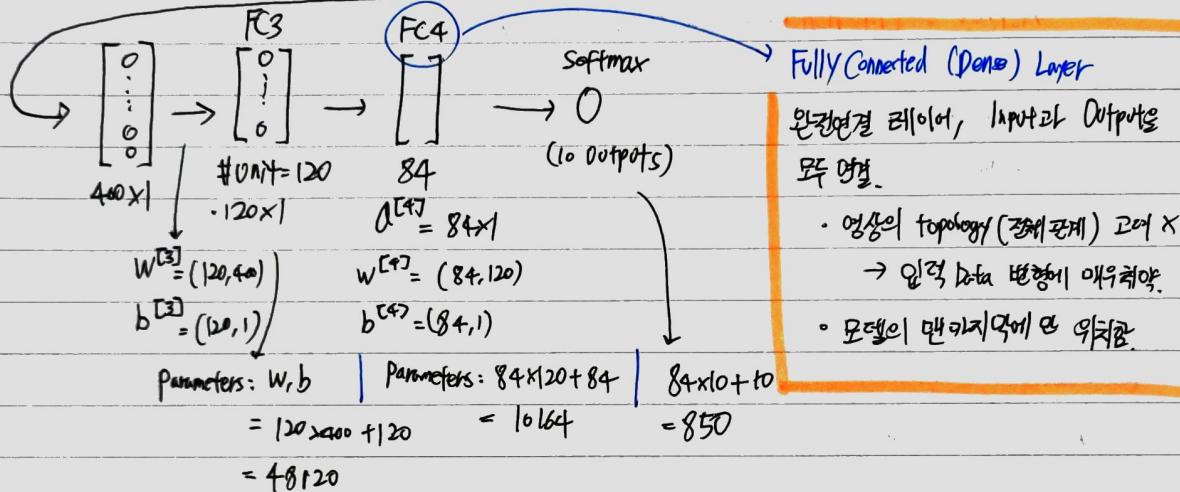
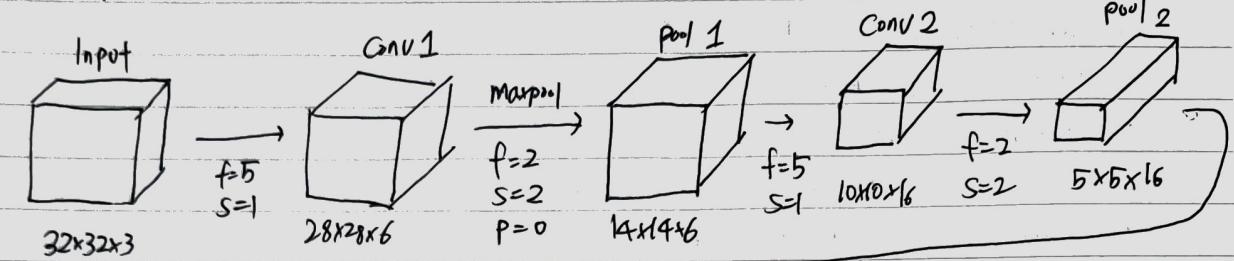
- 최댓값 대신 평균값 사용.

- 실제로는 Average 대신 Max가 주로 사용됨.

* Max pooling에 보통은 padding 미사용.

▷ One layer: Conv + pool

▷ Example: LeNet-5



▷ Convolution을 사용하는 이유?

① Parameter 공유: feature detector가 한 부분뿐 아니라 여러 다른 부분에서도 동일하게 사용됨.

② Sparsity of Connections: 1 Output이 대체 모든 Input을 사용하지 않음.

- image의 경우 filter 크기만큼만 Output에 연결됨.

- translation Invariance (변환 불변성): 구조가 이동된다면 해당 feature 보존

▷ 훈련

Input image \rightarrow (Conv \rightarrow pool) \rightarrow FC \rightarrow \hat{y}

$J = \frac{1}{m} \sum L(\hat{y}^{(i)}, y^{(i)})$ 를 최소화하려 gradient descent 사용.

▷ Quiz

- Parameter = #Weight + #bias = $f^{[1]}_{[1]} \times f^{[1]}_{[1]} \times n_c^{[1]} \times n_c^{[1]} + n_c^{[1]}$
- RGB image에서, $n_c^{[0]} = 3$

▷ 실습 #1

- def zero_pad (x, Pad) : return x-pad
- Conv-single-step (A-slice-prev, W, b) : return $Z = WA+b$
 - A-slice-prev: Input dataq slice, (f, f, n_C-prev)
 - $W = (f, f, n_C\text{-prev})$, $b = (1, 1, 1)$
- Conv forward (A-prev, W, b, hparameters) : return Z, cache
 - A-prev : (M, n_H-prev, n_W-prev, n_C-prev)
 - W : (f, f, n_C-prev, n_C), $b = (1, 1, 1, n_C)$
 - n_H와 n_W 계산
 - A-prev는 padding
 - A-slice를 정의할 때 corner₂ zed, [vert_start: vert_end, horiz_start, horiz_end, :]
 - h, w, f, s \in zed

▷ Backward

$$\begin{aligned} \cdot dA &= \sum_{h=0}^{n_H} \sum_{w=0}^{n_W} W_{hw} dZ_{hw} \\ \cdot dW &= \sum_{h=0}^{n_H} \sum_{w=0}^{n_W} a_{slice} dZ_{hw} \quad \cdot db = \sum_h \sum_w dZ_{hw} \end{aligned}$$

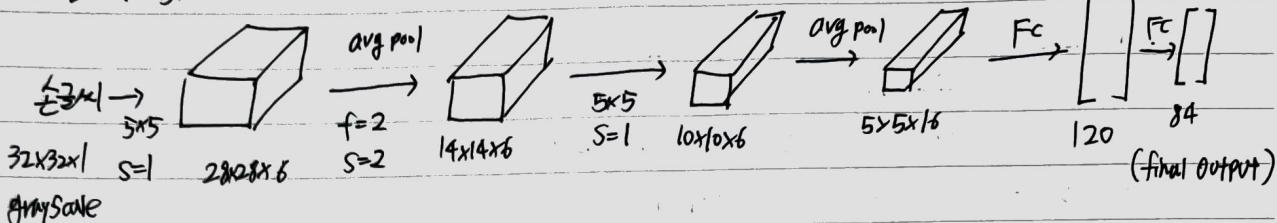
▷ 실습 #2

- Tensorflow에서는 자리 표시자 (공란일 일정 고려해도 대체) 필요.
- tf.placeholder (dtype, shape=[], name)
- tf.get_variable (name, shape, initializer=' ', ...)
- sess = tf.Session() : 텐서플로 연산 실행을 위한 class
 - sess.run (fetches, feed_dict, options, run_metadata)
 - 이 문제에서 사용 fetches = [optimizer, cost]
 - feed_dict = {X: minibatch_X
Y: minibatch_Y}
- tf.nn.conv2d (X, W, strides=[1, s, s, 1], padding='SAME')
- tf.nn.max_pool (A, ksize=[1, f, f, 1], strides=[1, s, s, 1], padding='SAME') = P
- tf.nn.relu (Z) = A
- tf.contrib.layers.flatten (P) = F
- tf.contrib.layers.fully_connected (F, num_outputs, activation_fn=, normalizer_fn=, ...) = Z[L]
- * np.squeeze (배열, 축) : 지정된 축을 차원 제거함.

[232], Case study]

D Classic Network

① Lenet-5



$\rightarrow 0$

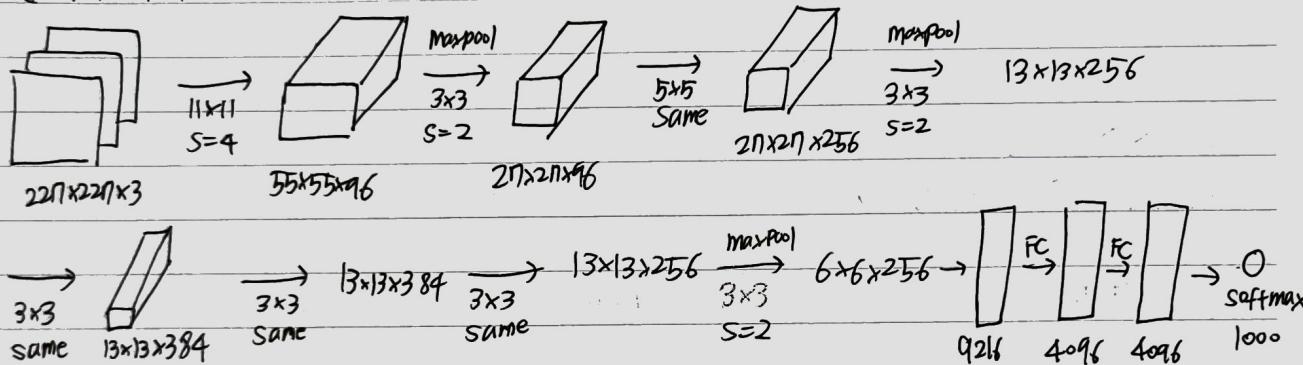
softmax
(10-way clif
= 0.9)

parameter: 6000074

layer 깊이 줄수록 $n_h, n_w \downarrow, n_c \uparrow$
 $(\text{conv-pool}) \times n \rightarrow \text{fc} \rightarrow \text{fc} \rightarrow \hat{y}$

Sigmoid, tanh 사용

② Alexnet



$\approx 60M$ parameters.

- ReLU activation 사용.

노벨로

- Multiple GPU 사용, Local Response Normalization layer 추가됨

③ VGG-16

• 많은 hyperparameter 때문 단순한 Network 구조.

• Conv = 3x3(filter), S=1, same

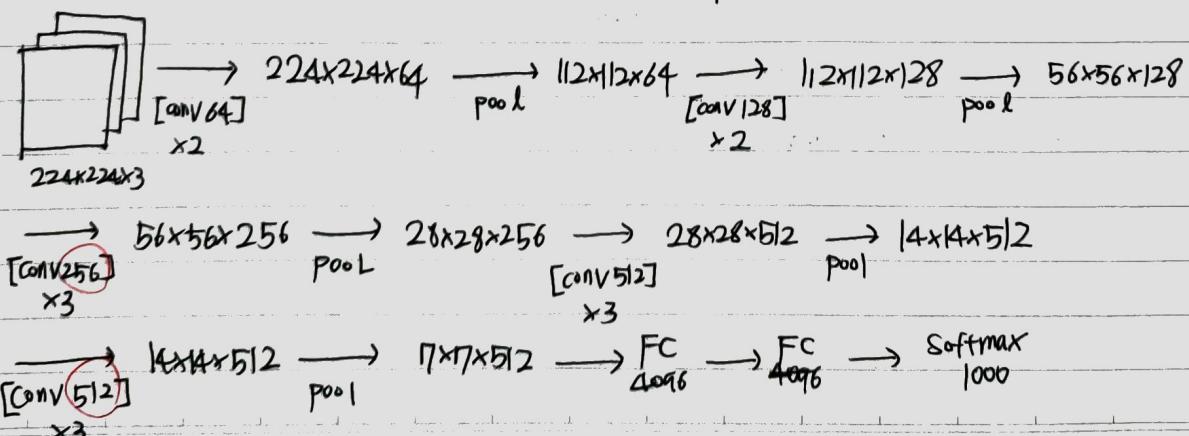
, 항상 동일함

• Maxpool = 2x2, S=2

• filter 개수 2배씩 증가,

• pooling 단계 차원 감소에 영향을 줌.

• Vgg16 \approx Vgg19



▷ ResNet

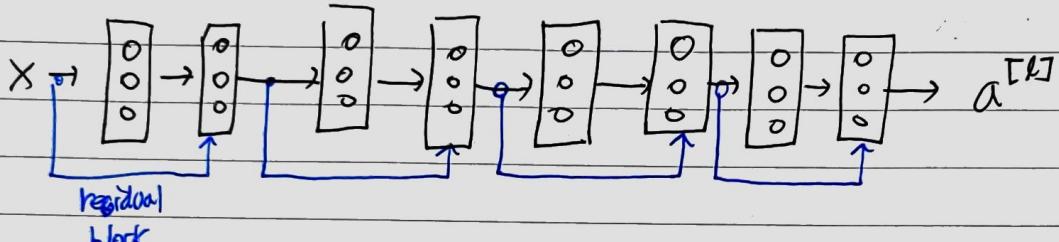
- residual block

$a^{[l+2]} \rightarrow \text{linear} \rightarrow \text{activation} \rightarrow a^{[l+1]} \rightarrow \text{linear} \rightarrow \text{activation} \rightarrow a^{[l+2]} \dots$ 로 표현되는

• "Shortcut", "skipping connection"

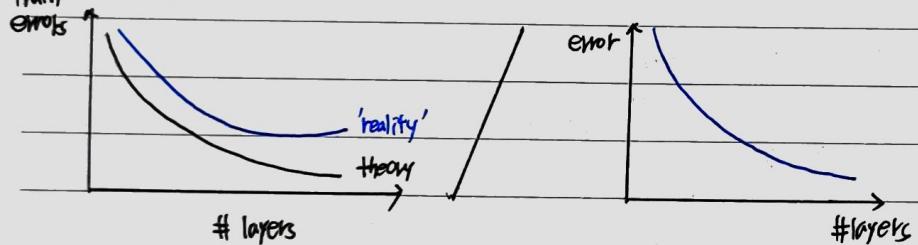
• 비선형성 추가로 단계에 바로 접근

$$a^{[l+2]} = g(z^{[l+2]}) \rightarrow g(z^{[l+2]} + a^{[l]})$$



train

errors



plain

ResNet

$$\begin{aligned} a^{[l+2]} &= g(z^{[l+2]} + a^{[l]}) , \text{ Relu} \\ &= g(W^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]}) \approx g(a^{[l]}) \end{aligned}$$

\uparrow Regularize: 0에 가까워짐. $= a^{[l]}$

→ layer를 더하는 것에는, 그리고 residual block을 더하는 것은

수행능력이 저하되지 않을음, (\because 향동함수를 학습하는 것은 매우 쉽기 때문)

→ 따라서 추가적인 연산을 줄이면서 이 추가된 layer에서 향동함수를 학습하는 것이 plain network에 비해 더 잘 수행할 것을 보장함.

◦ Dimension

- $a^{[l+2]} \equiv z^{[l+2]}$ 이고 가정

- 만약 이를 경우, 추가 matrix W_S 를 더해줄.

$$a^{[l+2]} = g(W_S a^{[l]})$$

256 256x256 128

$$6 \times 6 \times 32 * 1 \times 1 \times 32 = (6 \times 6 \times 1)$$

DATE.

NO.

▷ 1x1 convolution

- Useful when:
 - # channel ($= N_C$) 를 줄이고자 할 때 : feature transformation \Leftrightarrow Padding = $N_h, N_w \downarrow$ 사용
 - 연산 \downarrow (# channel \downarrow)
 - 비선형성을 추가하는 효과.
 - 초기 네트워크 구축에 유용한.

반면

▷ Inception Network 개념

- 1x1, 3x3, Conv 또는 Pooling 등의 여러 성백지 중 하나 선택이 아닌 모든 작업을 수행하고 Output을 stacking함.
- 대신, 모든 Output의 Dimension은 같아야 함.
- Computational Cost
 - : Output의 수 \times 1개의 Output을 계산하는데 필요한 곱셈의 수.

$$(28 \times 28 \times 32) \times (5 \times 5 \times 192) = 120M$$

◦ Using 1x1 Conv

(추가 step)

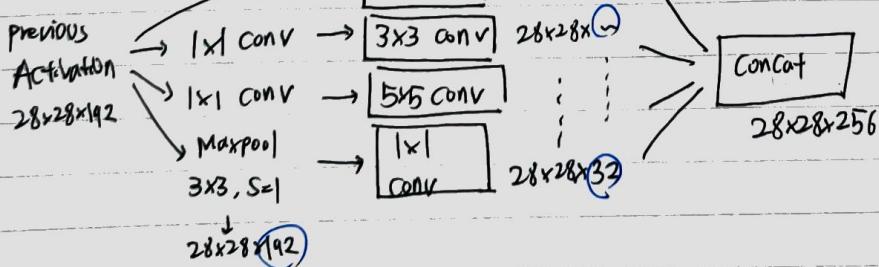
$$(28 \times 28 \times 16) \times (1 \times 1 \times 192) = 2.5M$$

$$+ (28 \times 28 \times 32) \times (5 \times 5 \times 16) = 10M$$

$$= 12.5M$$

▷ Inception Network

◦ Module



◦ Network - Module 들을 연결.

- softmax branch 중간에 결합, 실제로 값을 예측함

→ 중간에 계산된 feature들이 그대로 온지 확인함

→ Regularizing 효과, Overfitting 방지.

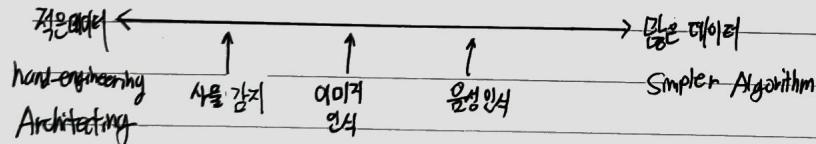
▷ Transfer Learning, 전이 학습

- Github 등 오픈소스 커뮤니티에서 코드, weight 다운로드
 - 기존 softmax layer를 나머지의 softmax unit으로 대체.
 - trainable=0 or freeze=1 등으로 softmax 이외의 모든 layer를凍結.
- ① Softmax (aych를 제외한 $[X \rightarrow \text{activation}]$ 가지를 pre-compute. // 작은 training set 보유 시.
- ② 데이터를 많이 보유했으면, freeze하는 층을 줄여도됨.
- $X \rightarrow \text{freeze} \rightarrow \text{training layer} \rightarrow \text{softmax}$
- ③ 데이터가 충분하다면, 오픈소스를 활용해 initialization을 시행할 수도 있음

▷ Data Augmentation (증강)

- ①
 - Mirroring
 - Random Cropping
 - rotation, shearing, local warping ...
- ②
 - R, G, B \rightarrow R+20, G-20, B+20 등과 같이 왜곡
 - 이미지 색상 변화에 강감하감.
 - PCA Color Augmentation : 차원별 색조를 동일하게 균형을 맞춤.
 - CPU/GPU로 학습시키기 전에, CPU thread에서 distort된 mini-batch를 생성할 수 있음.

▷ State of CV



• benchmark offed tips

- Ensemble: 독립적으로 여러개를 훈련 시켜서 Averaging.
- Multi-Crop of test time: • 10-Crop 후 Averaging.

▷ 실습 - ResNet 50 // Image 분류에 효과적인 방법

- Identity block: $a^{[L]} \equiv a^{[L+2]}$ 일때
- 1 layer: Conv2D \rightarrow batch Norm \rightarrow ReLU
- skips over 3 layers
- Convolutional block: $a^{[L]} \neq a^{[L+2]}$ 일때
 - Identity of shortcut path || Conv2D 추가.
 - \rightarrow Shortcut Path: Conv2D + batch normalization

<3주차, Object Detection>

▷ 객체 Localization

- 객체 탐색 (detection) : Object localization(위치추출) + classification
- 경계 box 자동으로 그리게 하기.

[Image clf + localization - 1개의 object 탐색.]

Detection: 여러 개의 object 탐색.

- classification with localization

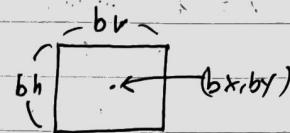
Ex) 자율주행차의 Object 카테고리: 1- 보행자

(softmax의 출력)

2- 차

3- 오토바이

4- 배경



→ Softmax가 bounding box를 출력하도록
bx, by, bh, bw 추가.

$$\rightarrow \hat{y} = \text{Class } 1, 2, 3, 4, \quad bx, by, bh, bw$$

- # 정의 (obj 1개일 때)

$$y = \begin{bmatrix} PC \\ bx \\ by \\ bh \\ bw \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} = \text{Object가 있는가? (0/1)}$$

ex) 보행자, 오토바이 X

→ 차선 존재: $y = \begin{bmatrix} 1 \\ bx \\ by \\ bh \\ bw \\ 0 \\ 0 \\ 0 \end{bmatrix}$

객체가 없을 때: $y = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$

상황에 따라
없음

◦ Loss function

$$\text{'MAE'}: L(\hat{y}, y) = \frac{1}{m} \sum (\hat{y}_i - y_i)$$

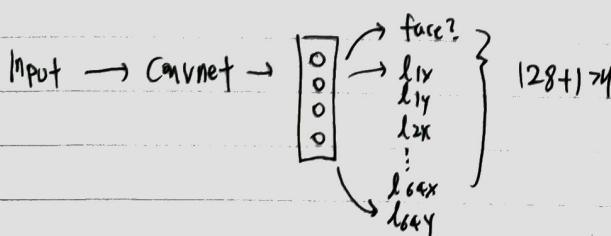
만약 제곱회수 사용할 시,

$$L(\hat{y}, y) = \begin{cases} \{(\hat{y}_1 - y_1)^2 + \dots + (\hat{y}_m - y_m)^2\} & \text{if } y_i = 1 \text{ 즉 물체가 존재할 때} \\ (\hat{y}_1 - y_1)^2 & \text{if } y_i = 0 \end{cases}$$

▷ Landmark Detection

- 특정한 landmark를 Output 시킴. (=3차)

Ex) 얼굴 인식에서 눈의 경계, 입 가막자리 등등



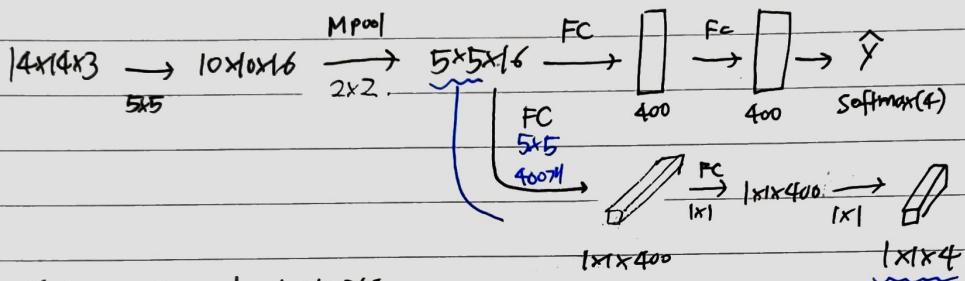
▷ Object Detection

• Sliding window detection

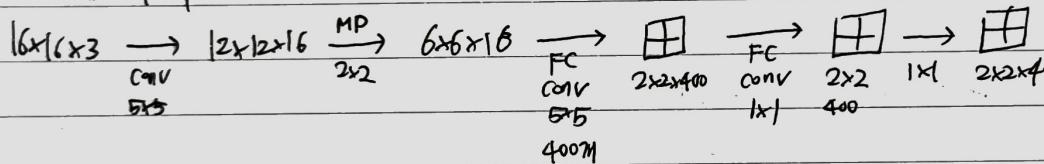
- 직사각형 영역을 결정한 후, 각각에 잘 맞는 유닛을 convolutional로 전달
- 직사각형 영역의 크기를 끌어 하여 2~3번 반복
- 탐색 obj에 있는 직사각형을 저장함.
- 큰 stride (coarse) = granularity ↑ : 성능을 향상
- 작은 stride = granularity ↓ : 계산 cost ↑ // convolutional approach로 해결 가능.

▷ convolutional Implement of Sliding Windows

- FC layer → Conv layer로 변환

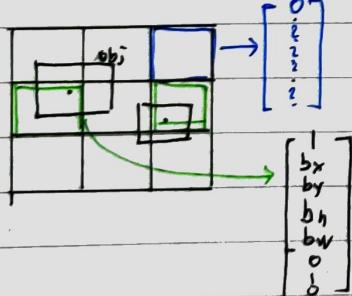


• 만약 4개의 window가 있을 경우



- 한번에 4개 윈도에 대한 분류 가능, 매우 빠르게 구현.

▷ Bounding box 예측



Yolo: 객체의 중간지점을 포함하는 그리드 셀에 객체 할당

⇒ 명시적으로 좌표를 Output할 수 있음

→ 9개의 grid cell, 각각 설마다 8차원 Output ~~vector~~ Vector

→ Target Output: 3x3x8



• 이후 여전자를 통해 학습

▷ bx, by, bh, bw 정의

원점위: (0,0) 오른쪽아래: (1,1)

$$y = \begin{bmatrix} 1 \\ bx \\ by \\ bh \\ bw \\ 0 \\ 1 \end{bmatrix}$$

격자내 위치, 0과 1사이 값

그 물체 크기 (너비/높이) / 격자 크기 → 상대적 크기임, 보다 큼 수 있음.

▷ Intersection Over Union (IOU)

- 알고리즘 평가에 사용됨.
- $\text{IOU} = (\text{True bounding box} \cap \text{Predicted box}) / (\text{True box} + \text{Predicted box} - \text{Intersection})$
- 관습적으로 $\text{IOU} \geq 0.5$ 일 경우 correct.

▷ 비-최대값 억제 (Non-max suppression)

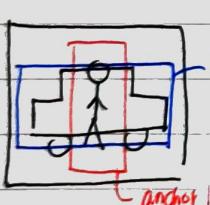
- 알고리즘이 같은 물체를 여러번 감지하는 것을 방지
 - 물체를 감지한 bounding box 중 감지 확률 P_c 가 최대인 상자를 고른다.
- ↳ 이후 해당 상자와 높은 IOU를 가진 상자를 제거.

<알고리즘>

1. $P_c \leq 0.6$ 인 모든 경계상자 버림
2. P_c 가 최대인 경계상자 기준
3. $\text{IOU} \geq 0.5$ 인 기준 경계상자 주위의 상자 버림
4. 1개만 남을 때까지 반복

▷ 앵커 박스 (Anchor Boxes)

- 이지 object는 grid cell 뿐 아니라 anchor box에도 할당될.
- (grid cell, anchor box with highest IOU)



$$y = \begin{bmatrix} P_c \\ b_x \\ s \\ C_3 \\ P_c \\ b_x \\ s \\ C_3 \end{bmatrix} = \begin{array}{l} \boxed{} \text{ Anchor 1} \\ \boxed{} \text{ Anchor 2} \end{array} \quad \begin{bmatrix} 1 & & & & & & & \\ b_x & b_y & : & & & & & \\ 1 & 0 & 1 & \dots & & & & \\ 0 & 1 & 0 & \dots & & & & \\ \vdots & \vdots & \vdots & \ddots & & & & \\ 1 & 1 & 1 & \dots & & & & \\ 0 & 0 & 0 & \dots & & & & \end{bmatrix} \quad \begin{array}{l} \text{(차원 맞을때)} \\ \text{각계가 맞아고 생각} \end{array}$$

▷ YOLO Algorithm

- 타겟 박스 shape: [物体 높이 x 객체 넓이 x (Anchorbox 개수 x Output vector의 차원)] : $3 \times 3 \times 16$ 꼴
- Output 박스: [물체종류여부, $b_x, b_y, b_w, b_h, \text{classes}]^T$
- 관습적으로 Non-Max Suppression 적용.
 ↳ 각각의 class에 대해 시행.

▷ 지역 제안 (Region Proposal)

① R-CNN

- 분할 알고리즘 (segmentation -) 사용, 몇 개의 지역만 고른 후 CNN 실행.

- 분할 알고리즘:

- 'blob' 을 찾음.

- 찾은 blob 주변에 bounding box 를 위치시키고 CNN 실행.

② Fast R-CNN

- Sliding window or Convolution 개념 적용. (conv implementation)

③ Faster R-CNN

- Fast RCNN + 지역 제안에도 convolution network 사용.

- 이것도 YOLO보다는 느림.

▷ 실습 (YOLO 사용, Car detection)

• 구조

$\text{IMAGE}(M, 608, 608, 3) \rightarrow \text{CNN} \rightarrow \text{ENCODING}(M, 19, 19, 5, 85)$

$\rightarrow \text{flatten to } (M, 19, 19, 425)$

• Class score

$$\text{Scores} = \max(P_c \times C_i)$$

= 객체 존재 확률 \times 특정 class에 속한 확률

$$\text{box-confidence} : (19 \times 19, 5, 1) \rightarrow P_c$$

$$\text{box-class-probs} : (19 \times 19, 5, 80) \rightarrow C_i$$

$$\text{boxes} : (19, 19, 5, 4) \rightarrow (b_x, b_y, b_w, b_h)$$

$$(b_1-x_1, b_1-y_1)$$

$$Y_1, X_1 = \text{maximum}$$

$$Y_2, X_2 = \text{minimum}$$

$$\rightarrow X_1 = \max(\text{box1}[0], \text{box2}[0])$$

$$X_2 = \min(\text{box1}[2], \text{box2}[2])$$

:

$$(b_1-x_2, b_1-y_2)$$

▷ 함수:

Score

① $\text{yolo-filter_boxes} (\text{box-confidences}, \text{boxes}, \text{box-class-probs}, \text{threshold})$: $\text{Score} \geq \text{threshold}$ 로 필터링.

• $\text{IOU}(\text{box1}, \text{box2})$

② $\text{yolo-non-max-suppression} (\text{Scores}, \text{boxes}, \text{classes}, \text{max_boxes}, \text{iou_threshold})$

$\rightarrow \text{yolo-eval} ()$: score threshold로 filtering한 NMS 수행.

▷ What is face recognition?

- face verification (검증) vs recognition

Verification

- Input = Image, 이름/ID

- Image == ID?

Recognition

- K명의 database O(1) K<100

- Input Image

- Input이 K명 중 어느 한 명일 때, Output ID.

▷ One-shot learning

~~☒~~ 각 원 (mn) class 를 '모든 사람' class 를 softmax unit 로 설정. (X)

- 각 원이 추가될 때마다 재학습이 필요.

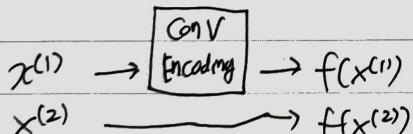
② 유사도 힘 사용.

 $d(\text{Img1}, \text{Img2})$ = 두 이미지의 차이 반환

If $d(\cdot) \leq \tau$ "Same"

otherwise, "different"

▷ 삼네트워크 (Siamese network)



$$d(x^{(1)}, x^{(2)}) = \|f(x^{(1)}) - f(x^{(2)})\|$$

: 두 개의 입력에 대해서 독립적으로 ConvNet을 통과시킨 뒤 비교.

• 학습

$x^{(1)}$ 와 $x^{(i)}$ 가 같은 사람이라면, $\|f(x^{(1)}) - f(x^{(i)})\|$ 는 작아야 함
다른 사람이라면, 최대화.

▷ 삼중형 손실 (Triplet loss)

◦ 하나의 Anchor 이미지 기준으로 (A), 같은 사람인 긍정 이미지 (P) 와 부정 이미지 (N) 의 거리를 구함.

$$\cdot d(x, y) = \|f(x) - f(y)\|^2 \quad // \quad d(A, P) \leq d(A, N)$$

$$\cdot \|f(A) - f(P)\|^2 \leq \|f(A) - f(N)\|^2 \rightarrow \|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 \leq 0$$

→ 적정한 해 $f(\cdot) = 0$ 을 반영하지 않도록 margin 추가

$$\Rightarrow \|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha \leq 0$$

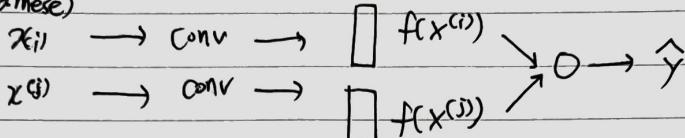
$$\cdot L(A, P, N) = \max(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0) \quad // \text{얼마나 응답하는 신경쓰지 않음.}$$

$$J = \sum_{i=1}^m L(A^{(i)}, P^{(i)}, N^{(i)})$$

- Training을 어렵게 만들기 위해, A,P,N 무작위 선택은 하지 말 것.
 $(\because d(A,N) \geq d(A,P) + x$ 를 만족하기 너무 쉽기 때문).
 - 어렵게 하려면, $d(A,N) \approx d(A,P)$ 여야 함.

▷ 열흘 경증 과 이전분류

(Siamese)



$$f(x^{(i)}) = x^{(i)} \text{의 } \text{인코딩}$$

$$\hat{y} = \sigma \left(\sum_{k=1}^{128} w_k |f(x^{(i)})_k - f(x^{(j)})_k| + b \right)$$

$$\frac{(f(x^{(i)})_k - f(x^{(j)})_k)^2}{f(x^{(i)})_k + f(x^{(j)})_k} \quad // \text{chi-square gate}$$

[산경의 스타일 복제], Neural style Transfer

내용 이미지 (E) → 새 이미지 (G)
스레일 (S)

(Image patch)

- Unit의 Activation 값을 최대화하는 이미지 조각들을 찾고, 해당 조각을 복원
 - layer 1: 모서리, 색 등 단순한 특성. (기법적임)
 - 갈수록 더 복잡한 형상과 pattern이 인식 가능. (각체 은전히 감지)

▷ Cost function (NST)

$$J(G) = \alpha J_{\text{content}}(C, G) + \beta J_{\text{style}}(S, G)$$

↗ C, G 가
 얼마나 비슷한지

↗ S, G 가
 얼마나 비슷한지

- G 생성 알고리즘
 - i) 무작위로 G 초기화
 - ex) $G = 100 \times 100 \times 3$
 - ii) GD 사용,) (G) 최소화

$$G := G - \frac{\partial}{\partial G} J(G)$$

▷ Content cost function

* hidden layer λ 에서 content 를 계산할 때

• pre-trained convnet (ex) vgg 사용

$$\Rightarrow J_{\text{content}}^{[L]}(C, G) = \frac{1}{2} \|a^{[L](C)} - a^{[L](G)}\|^2$$

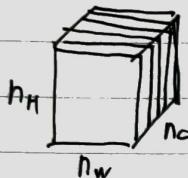
image의 content 징역에

Layer L activation이 쓰임.

▷ Style cost function

• Style은 'channel \in activation 1fol의 상관관계'

correlation between activations across channels.



• 채널 1과 2의 상관관계는 어떤가?

- 'Correlated': 특정 channel이 특정 값이, 다른 channel에서도 나온다.
 - '오른쪽'
 - '주름'
 - '원형' ...

\Rightarrow 스타일의 정도 = 이런 component들이 얼마나 많이 같이 나타나나/ 나타나지 않느냐.

◦ Style matrix

$a_{ijk}^{[L]}$ = activation at (i, j, k) , $G^{[L]} = n_C^{[L]} \times n_C^{[L]}$ dim.

$$G^{[L](S)}_{KK'} = \sum_{i=1}^{n_H} \sum_{j=1}^{n_W} a_{ijk}^{[L](S)} \cdot a_{ijk'}^{[L](S)} / \text{(이미지의 각각 위치마다)}$$

↑ ↑ ↑
H W C

Index of Channel

Gram matrix

H W C

• Correlated $\rightarrow G^{[L]}_{KK'} \uparrow$

Uncorr $\rightarrow \downarrow$

$$G^{[L](G)}_{KK'} = \sum_i \sum_j a_{ijk}^{[L](G)} \cdot a_{ijk'}^{[L](G)}$$

$$\therefore J_{\text{style}}^{[L]}(S, G) = \frac{1}{(\dots)} \|G^{[L](S)} - G^{[L](G)}\|_F^2 \quad \text{Frobenius norm}$$

↑
style term

$$= \frac{1}{(2n_H^{[L]} n_W^{[L]} n_C^{[L]})^2} \sum_{K} \sum_{K'} (G^{[L](S)}_{KK'} - G^{[L](G)}_{KK'})^2$$

$$\Rightarrow J_{\text{style}}(S, G) = \sum_L \lambda^{[L]} J_{\text{style}}^{[L]}(S, G)$$

Weight vector

▷ Gram matrix 계산

• activation을 reshape: $(H, W, C) \rightarrow (H \cdot W, C)$ ($= F$)

$$G^{[L]} = F^* F, T$$

$$\Rightarrow J = \alpha J_{\text{content}}(S, G) + \beta J_{\text{style}}(S, G)$$

$$G := G - \frac{\partial}{\partial G}(J(G))$$

▷ 1D, 3D Generalizations

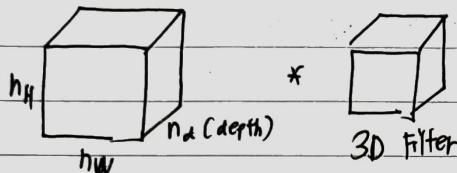
- 1D 데이터 : t-Series의 따른 심박수 (EKG)

t-Series로 따른 전압

$$\begin{array}{l} 14 \times 1 \rightarrow 10 \times 16 \rightarrow 6 \times 32 \\ F=5 \quad F=5 \\ S=1 \quad S=1 \\ 16개 \quad 32개 \end{array}$$

- 3D 데이터 : CT Scan.

* 인풋이 3D block 형태임.



$$\begin{array}{l} 14 \times 14 \times 14 \times 1 \rightarrow 10 \times 10 \times 10 \times 16 \rightarrow 6 \times 6 \times 6 \times 32 \dots \\ 16 \text{ filt} \quad 32 \text{ filt} \\ F=5 \quad F=5 \\ S=1 \quad S=1 \end{array}$$

▷ 실습

- tensorflow

tf.reshape (tensor, shape)

tf. reduce_sum (input, axis, keepdims)

tf. transpose (a, perm=[])

- perm ~~[0, 1, 2]~~ = 차원끼리 순서

- 디폴트: perm = [2, 1, 0]

$$2 \times 2 \times 3 \xrightarrow{T} 3 \times 2 \times 2 \quad \text{총 행 열}$$

$$\text{perm} = [1, 2, 0] : \quad 2 \times 2 \times 3 \rightarrow 2 \times 3 \times 2$$