Convolutional Neural Networks

1-1 Computer Vision

- 컴퓨터 비전 문제: Image Classification(고양이인가? (0/1)), Object Detection, Neural Style Transfer(하나의 사진을 다른 스타일로 변형)
- 컴퓨터 비전의 장애물: 입력이 매우 클 수 있음 – 64x64x3의 이미지는 12288의 입력값이 됨, 큰 이미지를 다룰 시 그 입력은 더 커지고, 표준 완전 연결 신경망을 사용해 가중치까지 더해준다면 그 값은 훨씬 더 커지게 됨 -> 과적합 발생 가능성, 많은 계산과 메모리 사용
- 하지만 CV 애플리케이션에서는 이미지의 크기에 대해서 고민하지 않아도 됨 -> 그러기 위해 **합성곱 연산**을 구현해야함 – 합성곱 신경망의 기본적인 빌딩블록이 됨

1-2 Edge Detection Examples

- 합성곱 작업은 합성곱 신경망의 핵심 요소 - 합성곱이 어떻게 작동하여 모서리를 감지할까?
- 사진에서 컴퓨터가 물체를 인식할 때, 수직인 모서리(세로선)를 가장 먼저 감지-> 수평의 모서리 감지

Vertical edge detection

6x6x1의 그레이 스케일 이미지 ***(합성곱)** 3x3의 필터(커널) = 4x4의 행렬

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

1	0	-1
1	0	-1
1	0	-1

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

이미지에 필터를 놓고 각 요소들을 곱해준 후의 총합을 차례대로 적어줌

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

6x6

*

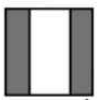
1	0	-1
1	0	-1
1	0	-1

3x3

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

*



결과값 행렬을 나타낸 이미지

: 밝은 영역이 중간에 있음 = 처음 이미지에서의 세로 경계선에 해당하는 부분

비록 크기가 안 맞고 검출된 경계선이 좀 두껍지만, 이는 작은 이미지를 예로 하고 있기 때문임.

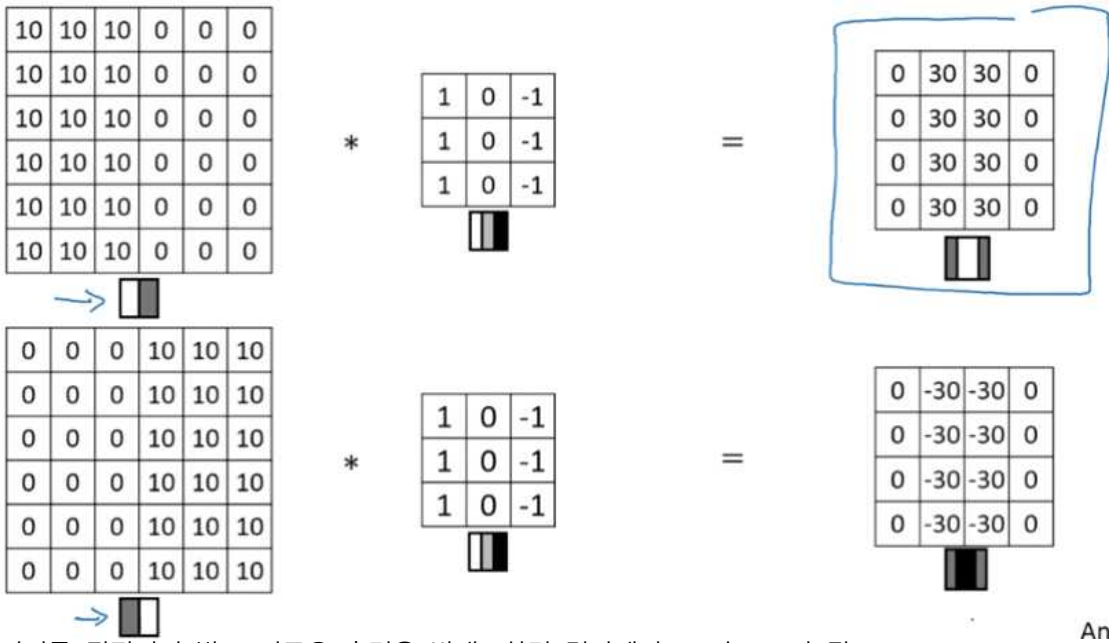
+) 수직 경계선 검출에서의 필터는 왼쪽에 밝은 픽셀이 있고, 오른쪽에 어두운 픽셀이 있음 (중간은 큰 영향 X)

- 프로그래밍 언어 파이썬: ConvForward, tensorflow: tf.nn.conv2d, keras: Conv2d

합성곱 연산을 합성곱 신경망의 기초 빌딩블록에서 사용하는 법

1-3 More Edge Detection

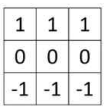
양(light)과 음(dark)의 윤곽선 차이 (서로 다른 밝기의 전환)/ 그 외에 다른 모서리 검출기



이미지를 뒤집어서 밝고 어두운 측면을 반대로하면 결과에서 30이 -30이 됨

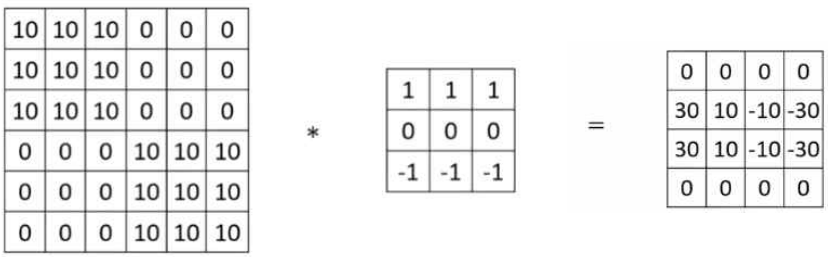
밝았다가 어두워지는지 / 어두웠다가 밝아지는지 차이를 알 수 있음

두 차이를 신경쓰지 않는다면 결과 행렬에 절대값을 씌워줘도 됨



Horizontal

수평선 검출 필터: 위쪽이 밝고 아래쪽이 어두움



- 2행 1열의 30은 위쪽이 밝고 아래쪽이 어두워서 강한 양의 윤곽선을 나타냄
- 2행 4열의 -30은 아래쪽이 밝고 위쪽이 어두워서 강한 음의 윤곽선을 나타냄
- 2행 2열의 10은 필터가 그 부분에서 왼쪽에 있는 양의 윤곽선과 오른쪽의 음의 윤곽선을 모두 인식했기 때문에 중간 크기의 값이 나오게 된 것 - 만약 이미지가 크거나 체커보드 형태였으면 이러한 중간값이 상대적으로 작아서 눈에 띄지 않을 것

필터에 어떤 숫자 조합을 쓸 지에 대한 논쟁 - sobel filter, schorr filter 등 많음

하지만 ... 딥러닝의 발전으로 윤곽선 검출을 위해 필터 내의 숫자를 수동으로 고를 필요 X

9개의 숫자를 변수로 설정하고 역전파로 학습된 숫자로 이루어진 필터와, 이미지를 합성곱 하면 좋은 윤곽선 검출기의 역할을 함 - 가로, 세로 뿐만 아니라, 45도로 기울어진 윤곽선처럼 원하는 어떤 형태의 윤곽선도 검출 가능함

1-4 Padding

Padding: 심층 신경망을 형성하기 위해서 합성곱 신경망을 변형하는 한 방식

- n x n 이미지를 f x f 필터로 합성곱한 결과는 n-f+1 x n-f+1 임
- 이 방법의 단점은 1) 합성곱 연산을 할 때마다 이미지가 축소됨 2) 가장자리 픽셀은 결과 이미지에 단 한 번만 사용됨(결과 이미지에 가장자리 근처 정보들을 제대로 사용 X)
- 이 단점의 해결방안은 합성곱 연산을 하기 전에 가장자리에 이미지를 덧대는 것임 (0을 덧대는 것이 일반적)

패딩을 얼마만큼 할 것인가?

1) Valid(유효) 합성곱: 패딩이 없음(p=0); n x n * f x f -> n-f+1 x n-f+1
2) Same(동일) 합성곱: (p 픽셀만큼) 패딩을 한 뒤 결과 이미지의 크기가 기존 이미지와 동일
   ; n x n * f x f -> n+2p-f+1 x n+2p-f+1
   n+2p-f+1= n --이를 풀면---> p= (f-1)/2 왼쪽의 식에 필터크기를 대입해 패딩 크기 결정

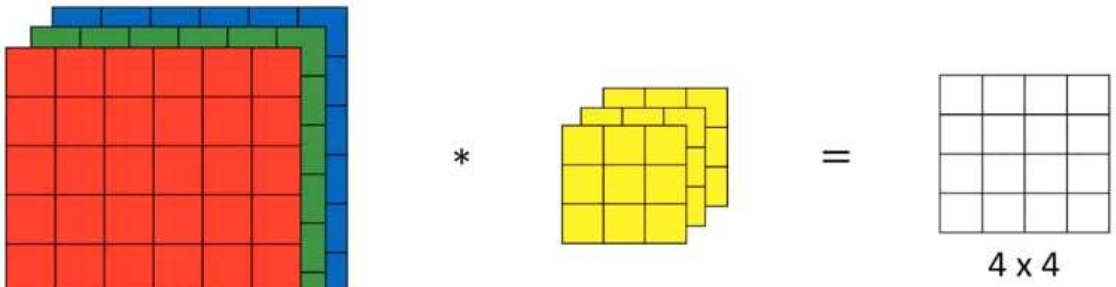
- f는 대부분 홀수 - 짝수이면 패딩이 비대칭이 됨

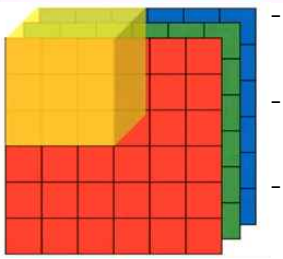
1-5 Strided Convolutions

- 스트라이드 합성곱은 합성곱 신경망의 기본 구성 요소
- stride = step = 몇 칸을 이동할까 결정
- n x n * f x f -> (n+2p-f/ s) +1 x (n+2p-f/ s) +1
- (n+2p-f/ s) +1이 정수가 아니라면 내림을 해줌

1-6 Convolutions Over Volumes 3D 입체형의 합성곱

- RGB 이미지는 3개의 색상 채널이 있음. 따라서 높이, 너비, 채널의 3D 이미지! 이러한 이미지의 윤곽선을 검출하거나 다른 특성을 알아보기 위해서는 3D 필터를 사용해야 함(이미지의 채널 수는 필터의 채널 수와 일치시켜야 함)
- 6x6x3 * 3x3x3 = 4x4x1





- 필터의 27개(3x3x3)의 숫자를 빨강, 초록, 파랑 채널에 해당하는 수와 곱해주고 더해줌
- 빨간색 채널의 세로 윤곽선을 검출하려면, 첫 번째 필터는:

1	1	1
0	0	0
-1	-1	-1

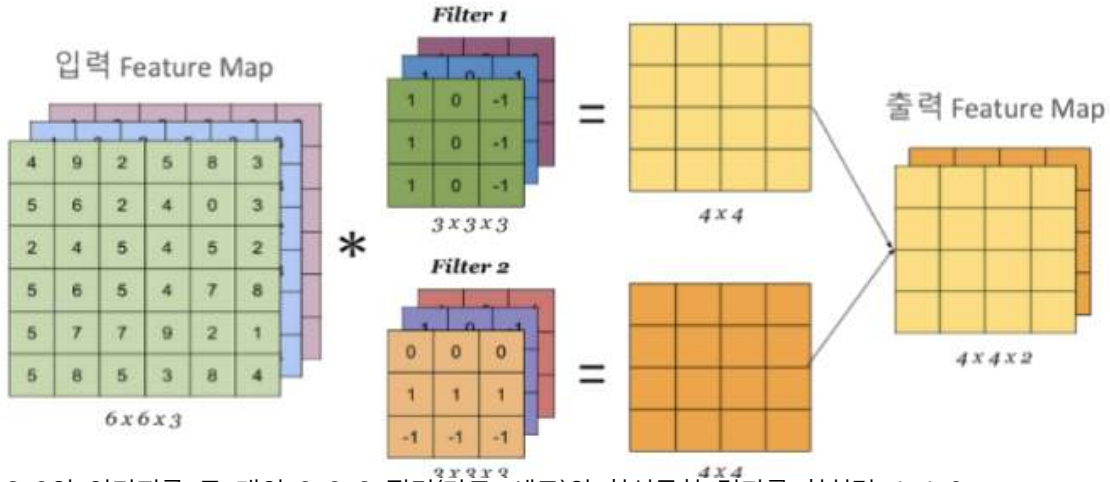
이고, 두 번째, 세 번째 필터는 모두 0으로 하면 됨

- 세로 수평선이 어떤 색이던 상관없다면, 세 가지 모두

1	1	1
0	0	0
-1	-1	-1

로 사용하면 됨

다른 윤곽선을 찾고자 한다면, 즉 여러 개의 필터를 동시에 사용하고자 한다면



6x6x3의 이미지를 두 개의 3x3x3 필터(가로, 세로)와 합성곱한 결과를 합치면 4x4x2

입체형에서의 합성곱은 효과적임.

- 세 채널의 RGB 이미지에 적용 가능함
- 가로와 세로 윤곽선처럼 두 개의 특성 또는 수백개의 특성을 검출할 수 있음 – 검출하고자 하는 특성의 수만큼 채널을 가질 수 있음

1-7 One Layer of a Convolutional Net
앞에서) 6x6x3의 이미지를 두 개의 3x3x3 필터(가로, 세로)와 합성곱한 결과를 합치면 4x4x2
이것을 합성곱 신경망 층으로 만들기 위해서 각각에 편향(b)을 더해주고 비선형성을 적용해야한다.
- b는 실수여야 함. 파이썬 브로드캐스팅을 통해 16개의 요소에 동일한 수(b)를 더해주고 비선형성을 적용해줌(ex. ReLU 비선형성 적용)
- 이렇게 6x6x3에서 4x4x2로 변환된 계산이 합성곱 신경망의 한 계층(layer)이 됨
- 이를 합성곱이 아닌 표준 신경망의 한 계층으로 연결시키기 위해서는 ...

합성곱 신경망 층의 매개변수 계산 방법: (신경망 계층의 크기 + 편향(bias)) x 필터 개수
ex) 10개의 필터가 있고, 각각 3 x 3 x 3 크기로 신경망의 한 계층에 있다면 이 층의 매개변수는 몇 개일까? : 각각의 필터는 3 x 3 x 3 크기로 27개의 변수를 가짐, 거기에 편향(1)을 더해줌. 그에 필터의 개수인 10을 곱해줌; (27 + 1) x 10 = 280개

입력 이미지의 크기와 변수의 수(280개로 고정)는 상관X. 아주 큰 이미지라도 적은 수의 필터로 여러 가지 다른 속성들을 검출 가능. 해당 합성곱의 성질을 이용해 과대적합을 방지 가능

합성곱 계층을 설명하는 표현 요약
layer l: 합성곱 계층. [l]은 특정 계층 l을 나타냄.
f^[l]: 필터의 크기. 특정 합성곱 계층 필터의 크기가 f x f 임을 나타냄.
p^[l]: 패딩의 양
s^[l]: 스트라이드
n_c^[l]: 필터의 개수
각 필터의 크기: f^[l] x f^[l] x n_C^[l-1]
* n_C^[l-1]: 필터의 채널 수와 입력의 채널 수가 동일해야하므로
Activations(편향과 비선형성을 적용한 뒤의 출력인 계층의 활성화 값): a^[l] = n_H^[l] × n_W^[l] × n_C^[l]
배치 경사 하강법 or 미니 배치 경사하강법 사용 시, A^[l] = m x n_H^[l] × n_W^[l] × n_C^[l]
Weights(가중치): f^[l] x f^[l] x n_C^[l - 1] x n_C^[l]
* 가중치의 개수는 필터를 전부 모은 것이므로 필터의 개수(n_C^[l])만큼 곱해줌
bias(편향): 필터마다 하나의 실수값인 편향을 가지기 때문에 n_C^[l] 개수만큼 존재

Input: n_H^[l-1] x n_W^[l-1] x n_c^[l-1] (H: 높이, W: 너비, c: 채널)
 * n_H, n_W, n_c이 계층l 이전 계층에서 온 것이기 때문에 [l-1] 사용
Ouput: n_H^[l] x n_W^[l] x n_c^[l]
* Ouput의 높이: n_H^[l] = {(n^[l-1] + 2p^[l] - f^[l]) / s} + 1
* Ouput의 너비: n_W^[l] = {(n^[l-1] + 2p^[l] - f^[l]) / s} + 1

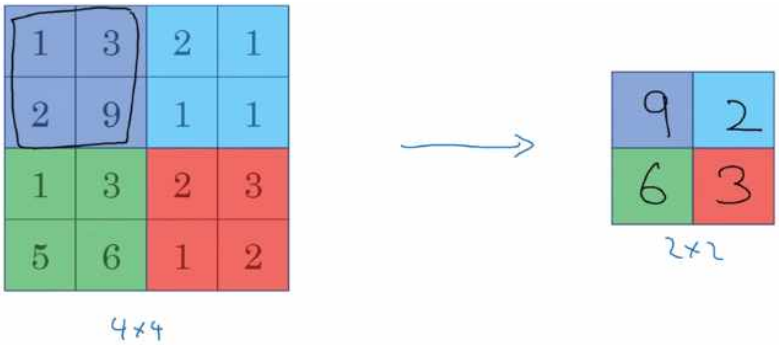
1-8 Simple Convolutional Network Example

- 신경망이 깊어질수록 높이와 너비가 비슷하게 유지되다가 줄어들고, 채널의 수는 늘어남
- 마지막 활성화값을 펼쳐서 하나의 벡터로 만든 뒤, 로지스틱 회귀 유닛이나 소프트맥스 유닛에 넣어 최종 예측을 함.
- 합성곱 신경망을 디자인 하는 일은 대부분 하이퍼파라미터(필터의 크기, 스트라이드, 패딩, 필터의 개수)를 선택하는 과정임
- 합성곱 신경망 층의 종류: Convolution layer(합성곱 층), Pooling layer(풀링층), Fully connected layer(완전 연결층)

1-9 Pooling Layers

합성곱층 외에도 합성곱 신경망은 풀링층을 사용하여 표현 크기를 줄임으로써 계산 속도를 높이고 특성을 더 잘 검출해낼 수 있음

1) Max pooling: 주어진 영역에서 최대값을 출력값으로 선택하여 픽셀의 크기를 축소하는 방법



- 2x2필터(f=2)와 stride=2를 적용한 것과 동일. f=2와 s=2를 max pooling의 하이퍼파라미터
- 한 특성이 필터의 한 부분에서 검출되면 높은 수
- 여러 하이퍼파라미터가 있지만 학습할 수 있는 변수가 없음. f,s는 고정된 값
- 결과 값 크기는 여전히 (n+2p-f/ s) +1 x (n+2p-f/ s) +1

일반적으로 f=2, s=2를 사용하기 때문에 높이가 너비가 절반이 된다.

2) Average pooling: 주어진 영역의 평균값을 계산하여 픽셀의 크기를 축소하는 방법

- 자주 사용되지 않음. 최대풀링이 훨씬 더 많이 사용됨. 평균 풀링은 신경망의 아주 깊은 곳에서 사용됨

요약

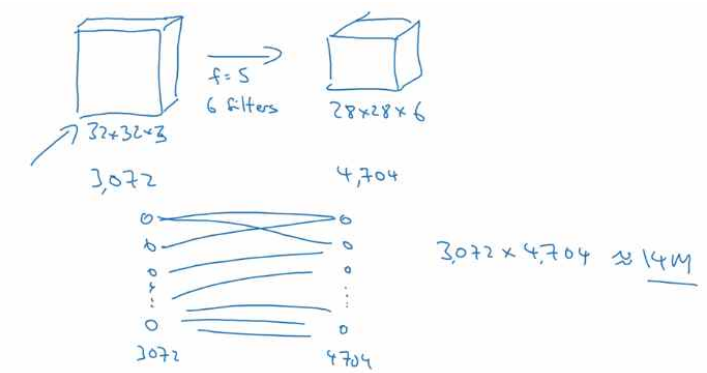
- 하이퍼파라미터 값인 f값과 s값은 보통 2로 선택되고, 높이와 너비를 절반 정도 줄어들게 함
- 최대 풀링에서는 패딩을 거의 사용하지 않음
- max pooling input shape: n_H x n_W x n_C
- max pooling output shape: {(n_H − f} / s} + 1 x {(n_W − f} / s} + 1 x n_C
  (pooling은 각 채널에 개별적으로 적용되기 때문에 입력 채널과 출력 채널이 일치)
- pooling은 학습하는 변수가 없음. -> 역전파 적용시 역전파가 가능한 변수가 없음
- hyperparameter가 있지만 고정된 값이라 학습할 수 있는 변수가 존재하지 않는다. 그래서 역전파를 적용해보면 역전파가 가능한 변수가 없다.

1-10 CNN Example

패턴: 합성곱층(CONV1) -> 풀링층(POOL1) -> 합성곱층(CONV2) -> 풀링층(POOL2) (반복) … -> FC(완전 연결층) -> Softmax(활성화함수)

- 합성곱층과 풀링층을 하나의 layer로 보기도 하고, 각각을 다른 layer로 보기도 함
- 신경망이 깊어질수록 높이와 너비는 감소하고, 채널의 수는 증가
- Activation Size(높이x너비x채널)는 신경망이 깊어질수록 점점 줄어듦 (너무 빠르게 감소한다면 낮은 성능을 의미할 수 있음)
- 신경망의 대부분의 Parameter는 FC(완전연결층)에 있음

1-11 Why Convolutions



ex) 입력이미지(32x32x3)에 필터(5x5, 6개)와 합성곱한 결과이미지(28x28x6)

- 신경망을 형성할 때 3072개(32x32x3)의 유닛을 한 층에 두고 4704개(28x28x6)의 유닛을 다음 층에 둔 다음, 각각의 뉴런들을 모두 연결하면 가중치 행렬의 변수의 개수는 약 1400만개 -> 훈련하기에 변수 多 -> 훈련이미지의 크기가 클 경우 변수의 개수는 훨씬 더 늘어남
- 합성곱 층의 변수의 개수를 보면, 각 필터는 25개(5x5)의 변수를 가지고 편향변수를 더하면 26개의 변수를 가짐. 6개의 필터가 있기 때문에 총 변수의 개수는 156개 -> 합성곱층의 변수의 수가 작게 유지됨
-

<합성곱 신경망이 적은 변수를 가지는 이유>

1. 변수 공유: 동일한 필터를 입력 이미지 여러 위치에 동일하게 적용-> 학습할 가중치 수 감소.

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
1	0	-1
1	0	-1

3x3

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

9개의 변수를 공유하며 16개의 출력을 계산함으로서 변수의 개수를 줄임

2. 희소 연결: 출력 값의 1행1열의 0은 3x3 합성곱으로 계산된 것으로, 입력의 3x3 영역에만 영향을 받음. 따라서 이 0은 36개의 입력 값 중 왼쪽 위의 9개와만 연결되어있고, 나머지 픽셀 값은 결과 값에 영향을 주지 않음

즉, 합성곱 연산이 입력 값에 필터 크기만큼의 영역에만 적용되고(작은 영역만을 사용해서 부분적으로 처리), 나머지 입력 값은 출력 값에 영향을 주지 않음 -> 신경망의 변수가 줄어들어 작은 훈련세트를 가지게 하고(연산량 감소) 과대적합 방지

- 합성곱 신경망은 이동 불변성 포착에 용이함. 동일 사진 내에서 몇 픽셀 이동한 이미지도 유사한 속성을 가지게 되고 동일한 결과를 얻을 수 있음. 모든 이미지의 위치에 동일한 필터를 적용하고, 초반과 이후의 층들에도 동일한 필터를 적용하기 때문에 신경망에서 자동으로 학습할 수 있고 이동불변성을 포착할 수 있음.

  * 이동불변성: 이미지 속의 물체가 위치가 살짝 바뀌어도, 합성곱 신경망은 여전히 그 물체를 인식할 수 있는 성질

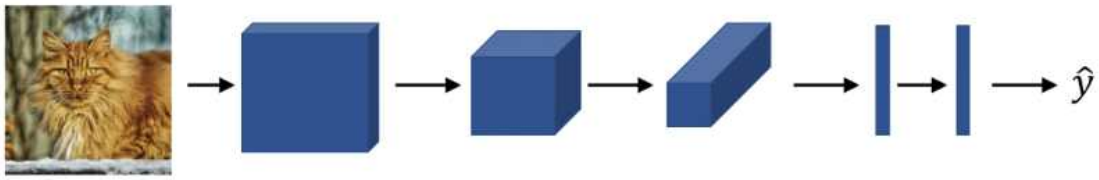
<이제 모두 조합해서 하나의 신경망을 훈련시켜보자>

고양이 인식기 구축

훈련세트(Training set)를 구성할 때, x는 이미지이고, y는 이진레이블 혹은 k개의 클래스 중 하나

합성곱 신경망 구조

Training set $(x^{(1)}, y^{(1)}) \ldots (x^{(m)}, y^{(m)})$.



- 이미지에서 시작- 합성곱과 풀링층 - 완전 연결층 - 소프트맥스 출력값인 y(y hat)의 예측값
- 합성곱 신경망과 완전 연결층은 변수 w와 편향 b를 가지는데 $\text{Cost } J = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$ 변수의 설정으로 비용 함수를 찾을 수 있음.
- 무작위로 w와 b를 초기화함으로써 비용 J(신경망의 훈련 세트에 대한 예측의 손실 합을 m으로 나눈 값) 계산 가능.
- 신경망을 훈련시키기 위해서는, 경사 하강법, 모멘트 경사 하강법, RMSprop 등 다양한 알고리즘을 사용해 변수를 최적화 가능 = 비용함수 J를 줄이기 위함
- 이것이 가능하다면 효율적인 고양이 인식기나 다른 검출기 구현 가능