

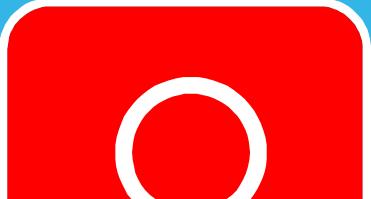
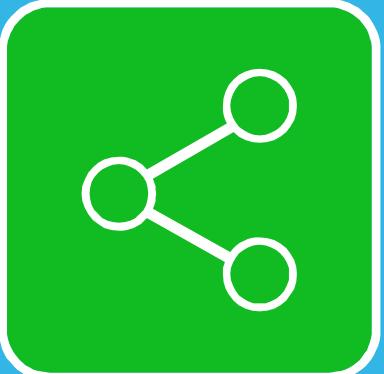
# Convolutional Neural Network

*Jaegul Choo* (주재걸)  
Korea University

<https://sites.google.com/site/jaegulchoo/>  
Most slides made by my student, Yunjey Choi



# Contents



- Introduction

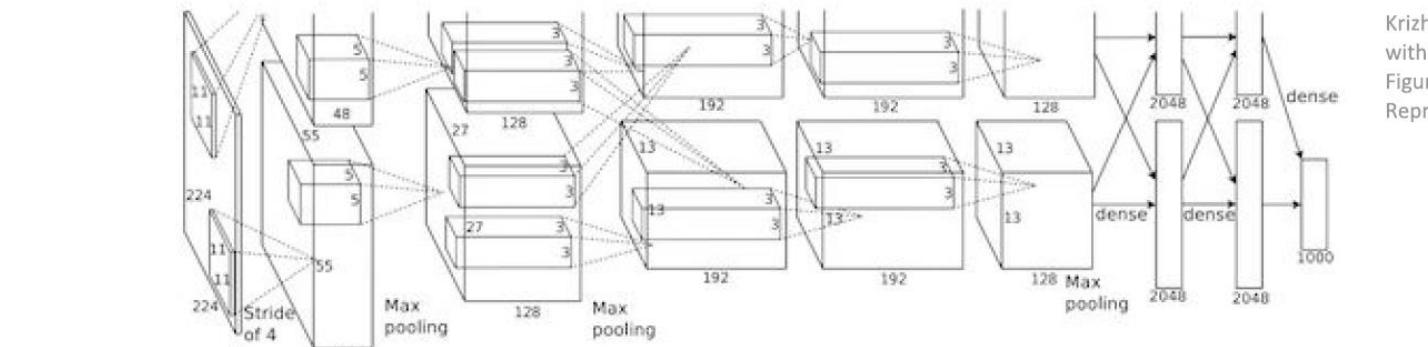
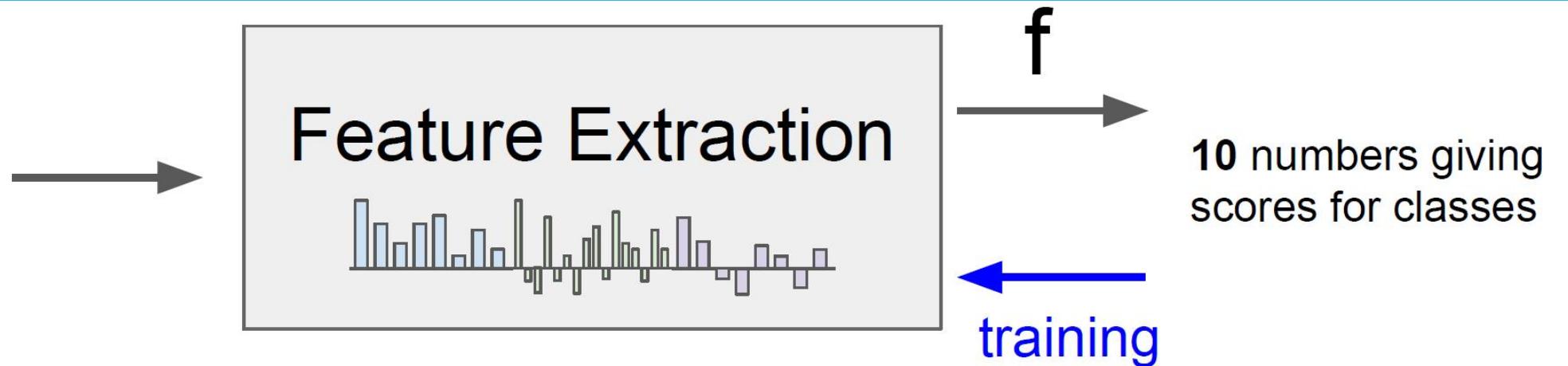
- Convolutional Neural Network

- Convolutional Neural Network2

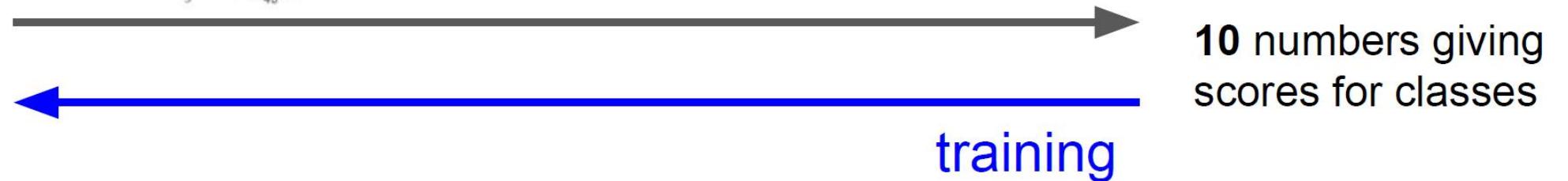
- CNN for Text Classification

- Advanced CNN Architectures

# 전처리로서 Feature 추출 방식 vs. ConvNets

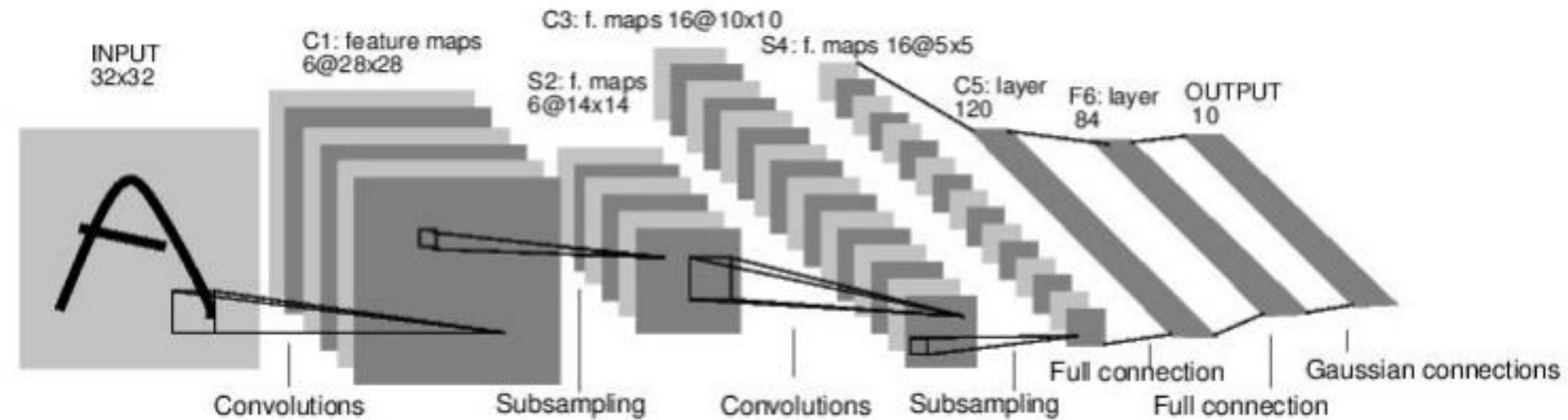


Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012.  
Figure copyright Krizhevsky, Sutskever, and Hinton, 2012.  
Reproduced with permission.



# LeNet: 초기 ConvNet 모델

Handwritten digit recognition에서 성공을 거둔 모델  
<https://youtu.be/yxuRnBEczUU>



[LeNet-5, LeCun 1980]

# AlexNet: CNN 발달에 박차를 가한 모델

2012년 ImageNet Challenge에서 AlexNet이 큰 격차로 1위를 차지함

## **ImageNet Classification with Deep Convolutional Neural Networks**

*[Krizhevsky, Sutskever, Hinton, 2012]*

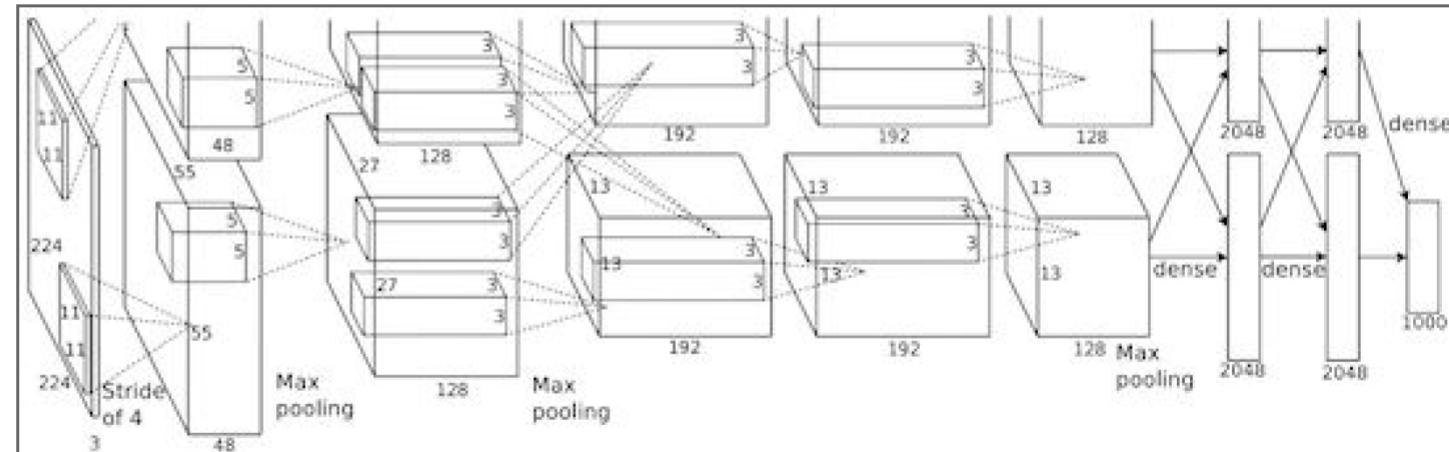
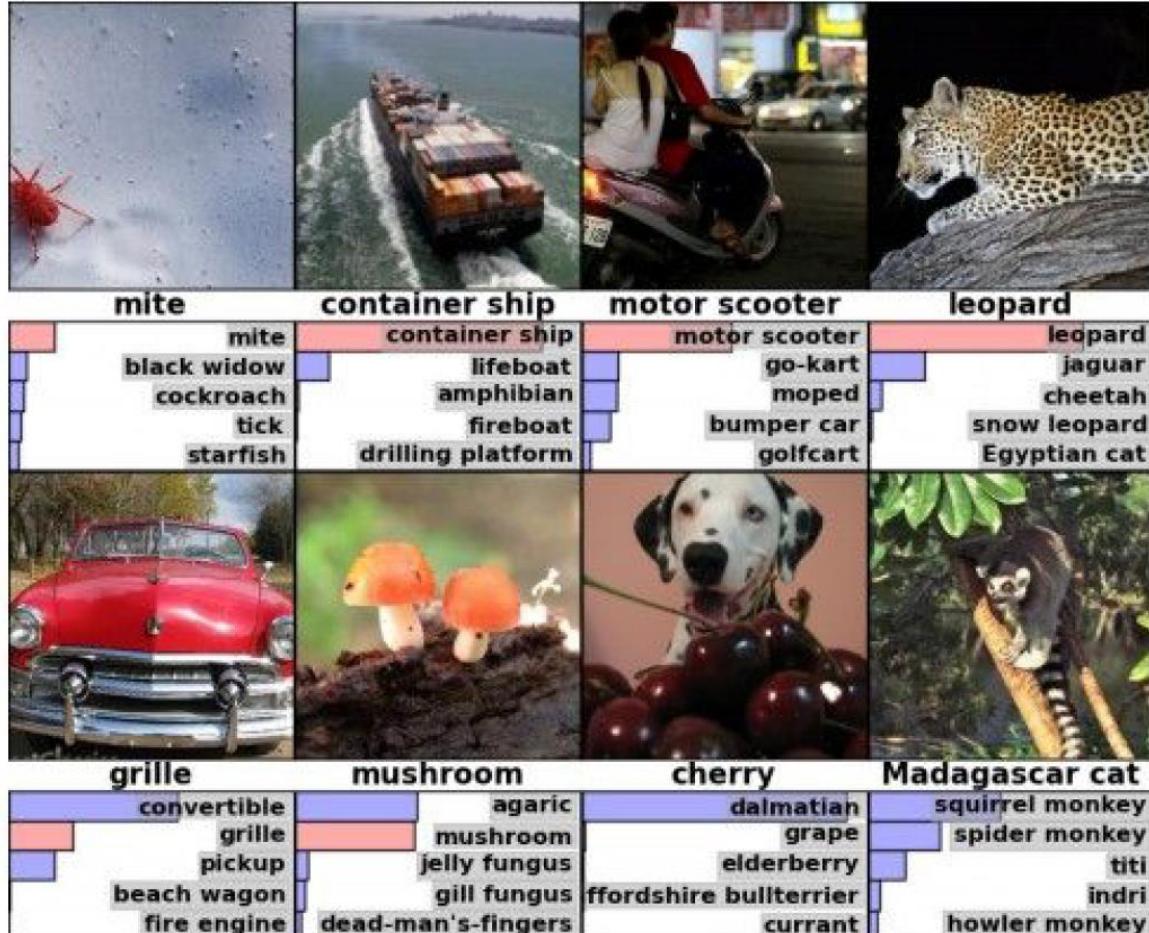


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

**“AlexNet”**

# ConvNets are Everywhere

Classification



Retrieval



Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

# ConvNets are Everywhere: 자율주행



self-driving cars

Photo by Lane McIntosh. Copyright CS231n 2017.



[This image](#) by GBPublic\_PR is licensed under [CC-BY 2.0](#)

## NVIDIA Tesla line

(these are the GPUs on rye01.stanford.edu)

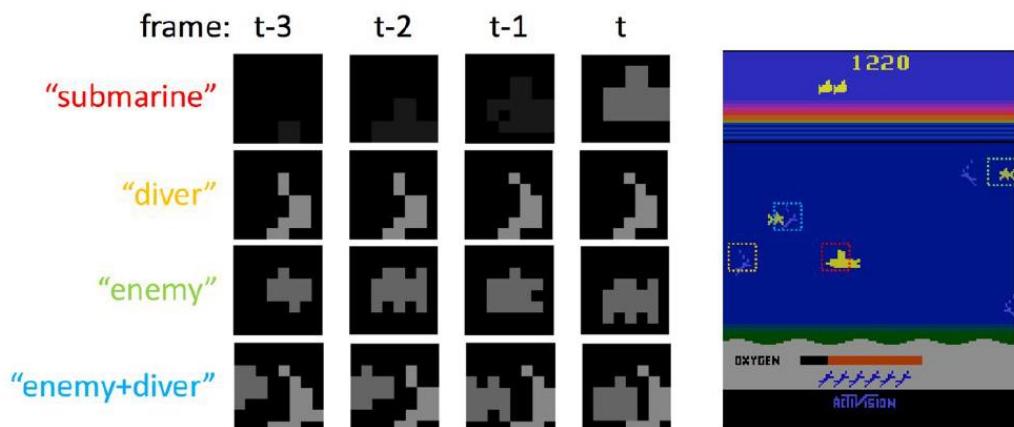
Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

# ConvNets are Everywhere

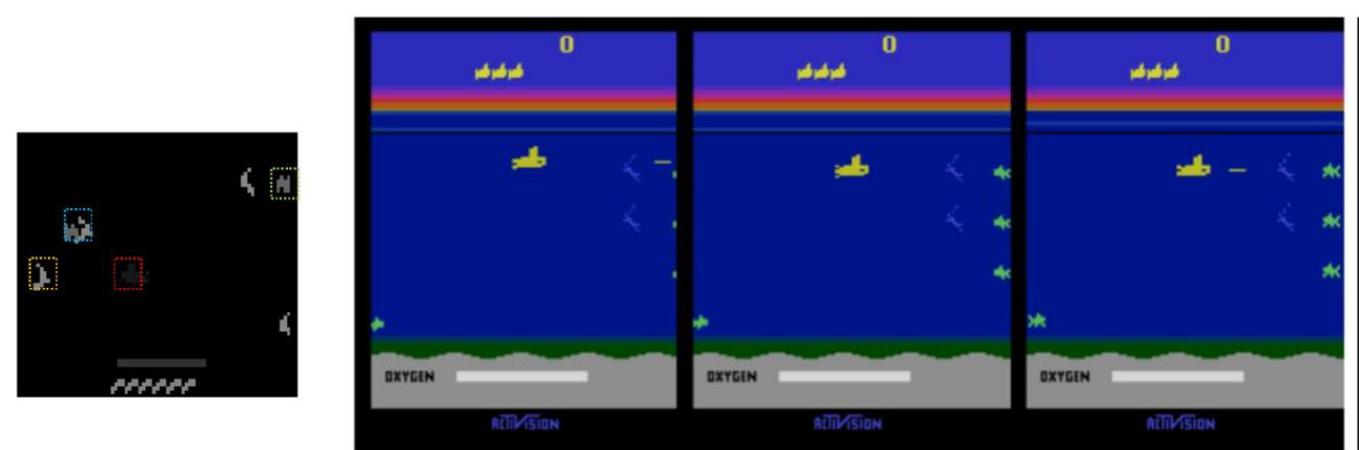


Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]

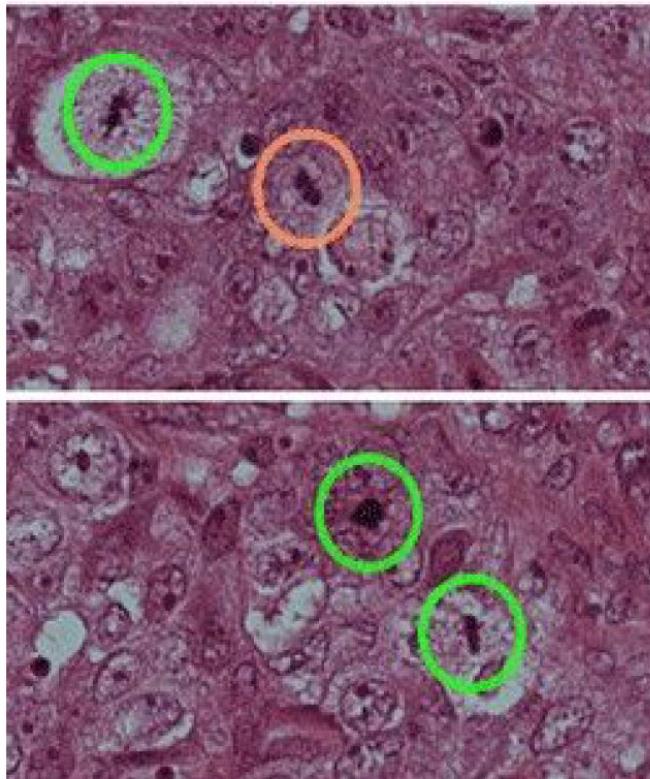


[Guo et al. 2014]



Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

# ConvNets are Everywhere



[Ciresan et al. 2013]



[Sermanet et al. 2011]  
[Ciresan et al.]

# ConvNets are Everywhere: Image Captioning

No errors

Minor errors

Somewhat related

Image  
Captioning



*A white teddy bear sitting in the grass*



*A man in a baseball uniform throwing a ball*



*A woman is holding a cat in her hand*



*A man riding a wave on top of a surfboard*



*A cat sitting on a suitcase on the floor*



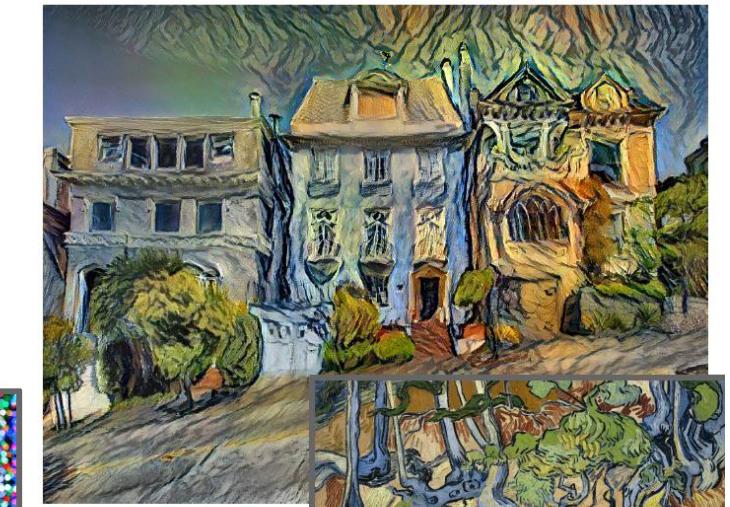
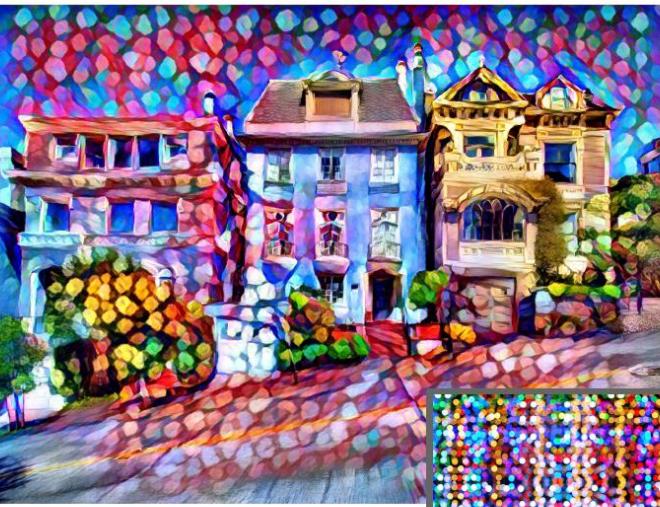
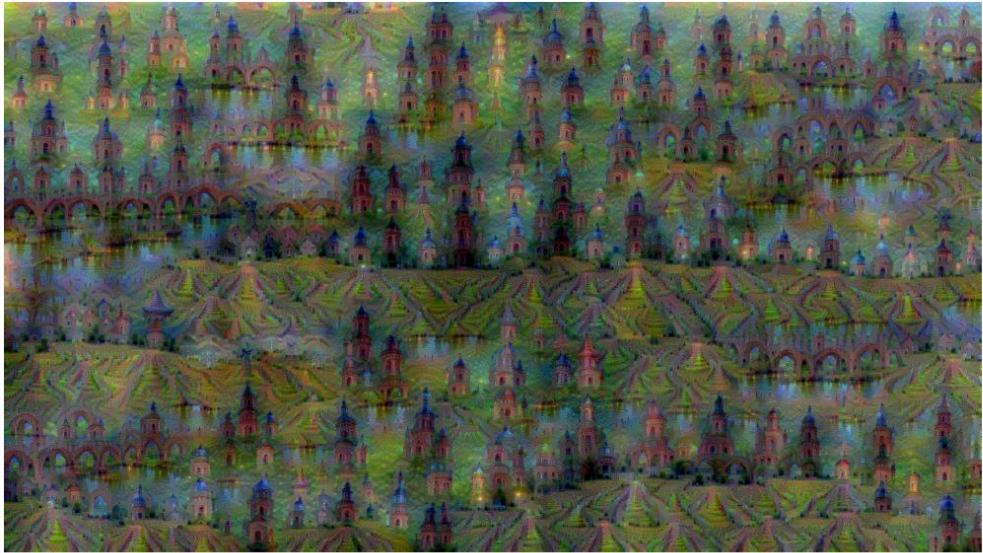
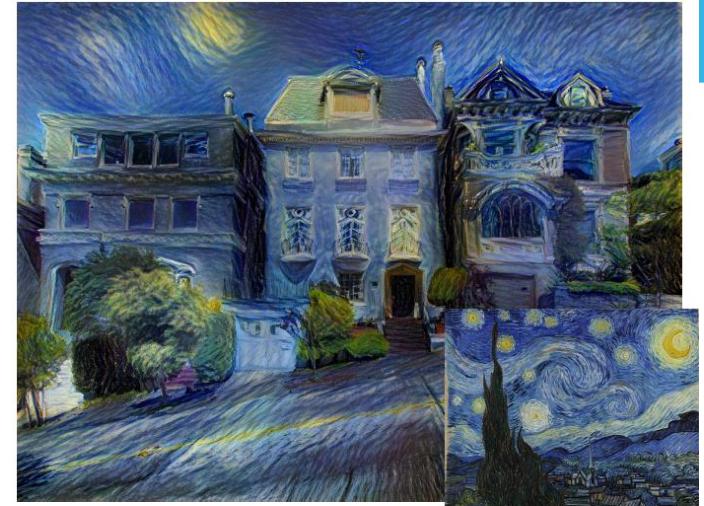
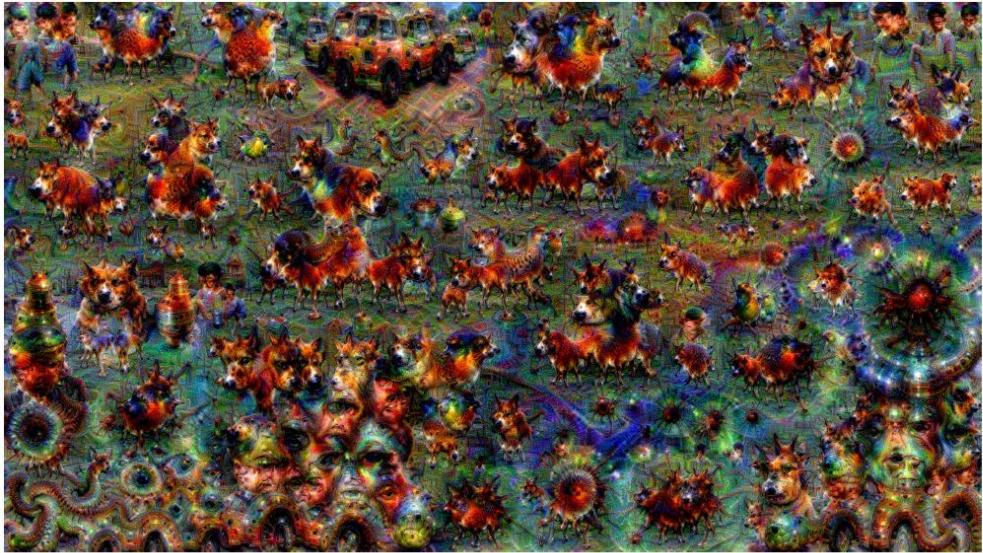
*A woman standing on a beach holding a surfboard*

[Vinyals et al., 2015]  
[Karpathy and Fei-Fei, 2015]

All images are CC0 Public domain:  
<https://pixabay.com/en/luggage-antique-cat-1643010/>  
<https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/>  
<https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/>  
<https://pixabay.com/en/woman-female-model-portrait-adult-983967/>  
<https://pixabay.com/en/handstand-lake-meditation-496008/>  
<https://pixabay.com/en/baseball-player-shortstop-infield-1045263/>

Captions generated by Justin Johnson using [Neuraltalk2](#)

# ConvNets are Everywhere



Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a [blog post](#) by Google Research.

[Original image](#) is CCO public domain

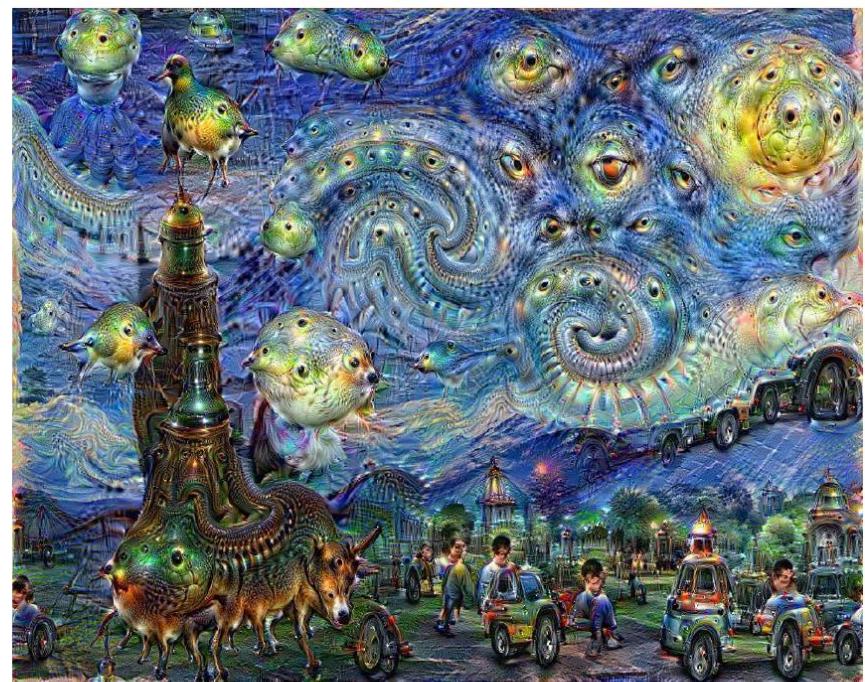
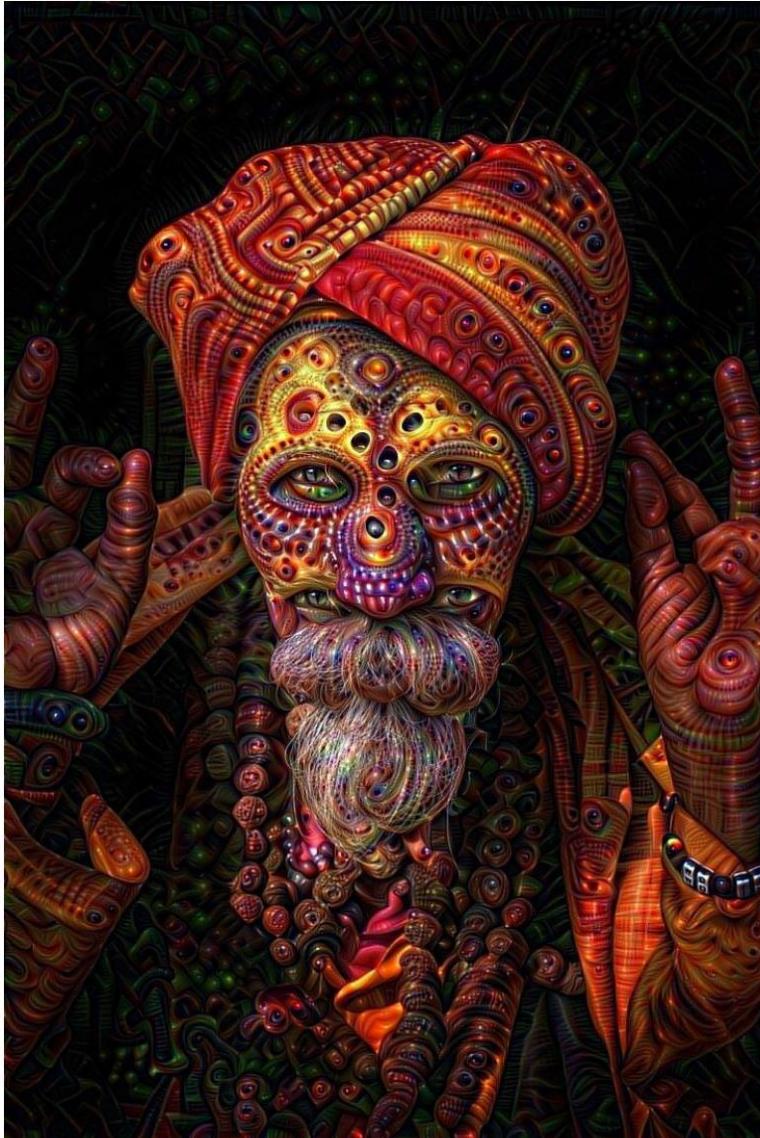
[Starry Night](#) and [Tree Roots](#) by Van Gogh are in the public domain

[Bokeh image](#) is in the public domain

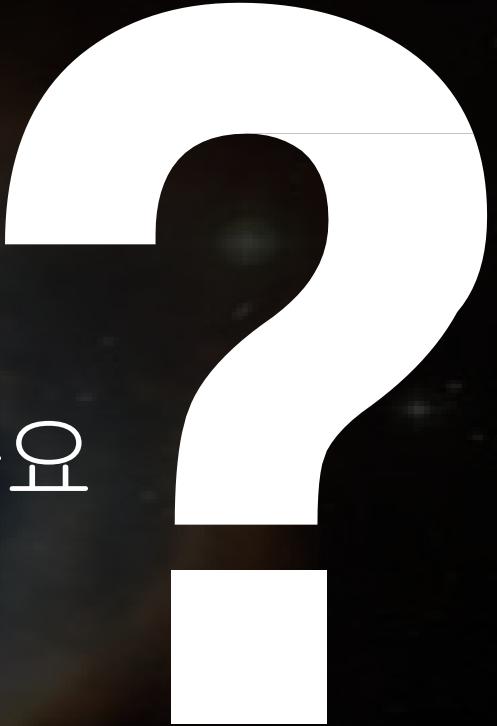
Stylized images copyright Justin Johnson, 2017;  
reproduced with permission

Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016  
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

# ConvNets are Everywhere: Deep Dream



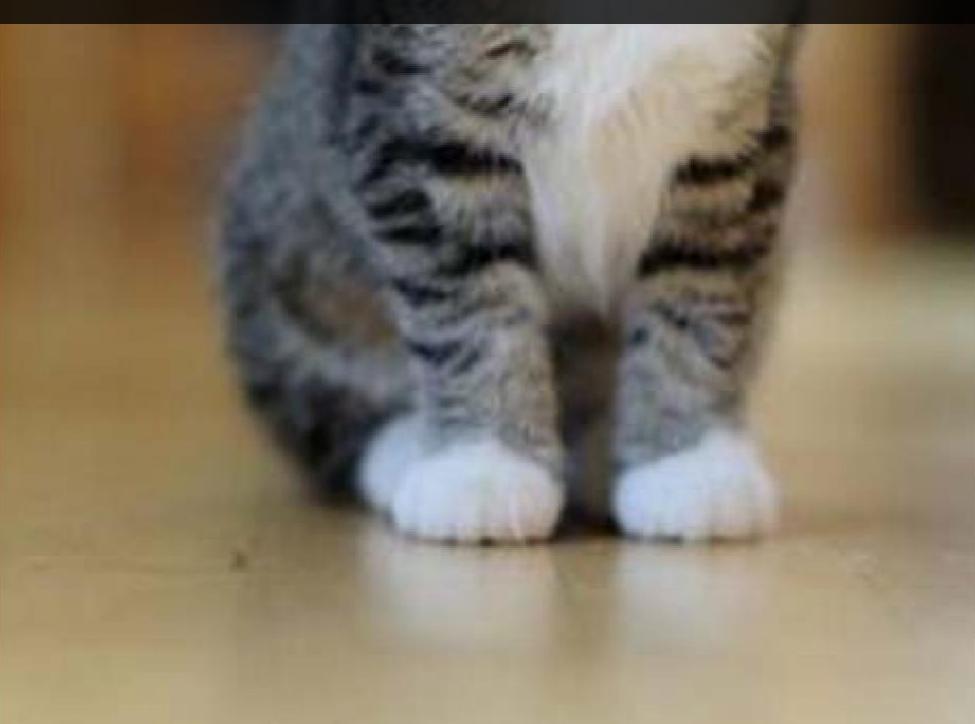
우리 눈에는 당연한 것이  
왜 컴퓨터에겐 어려운 걸까요







둘을 구분할 수 있겠습니까?



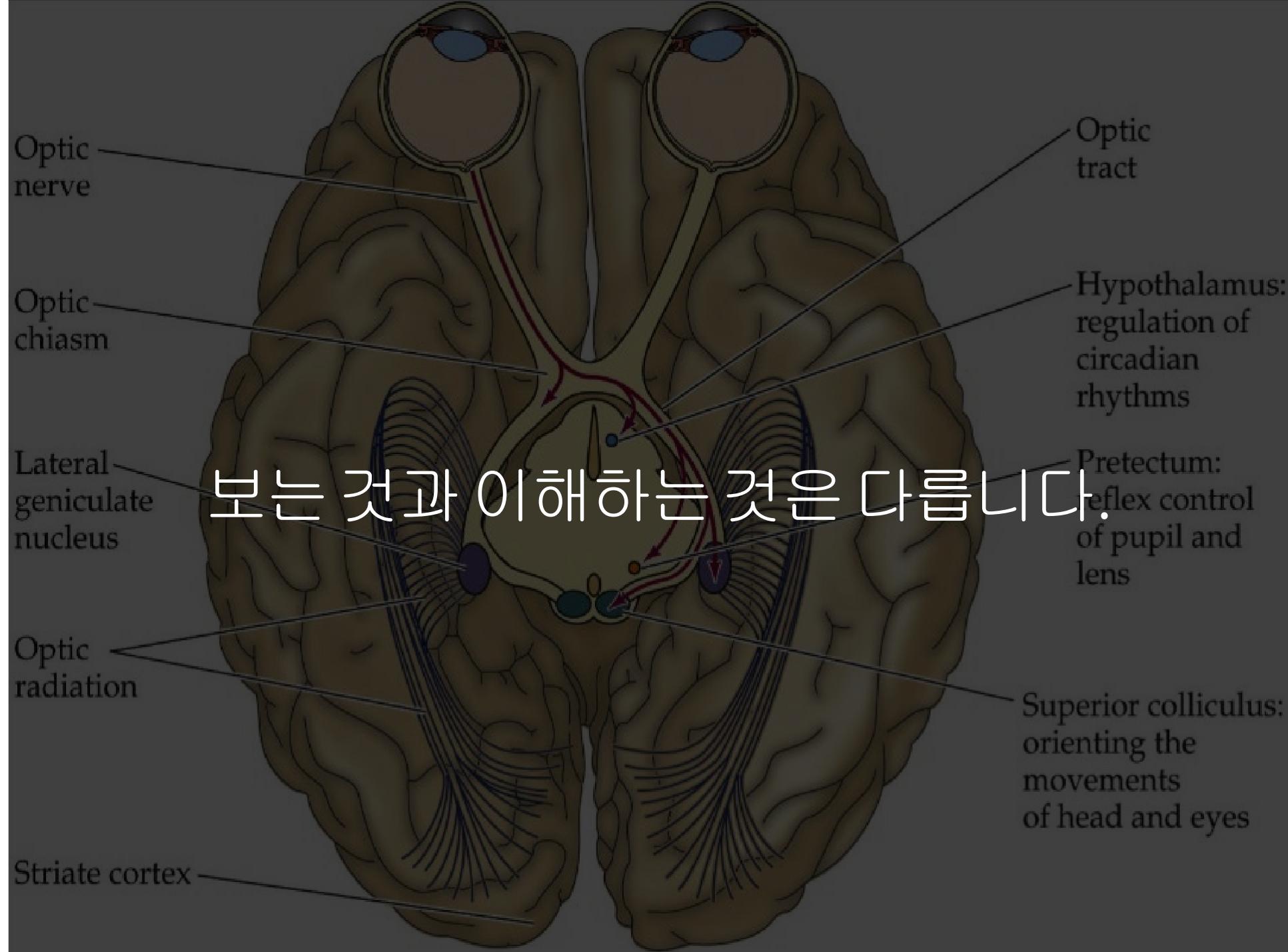


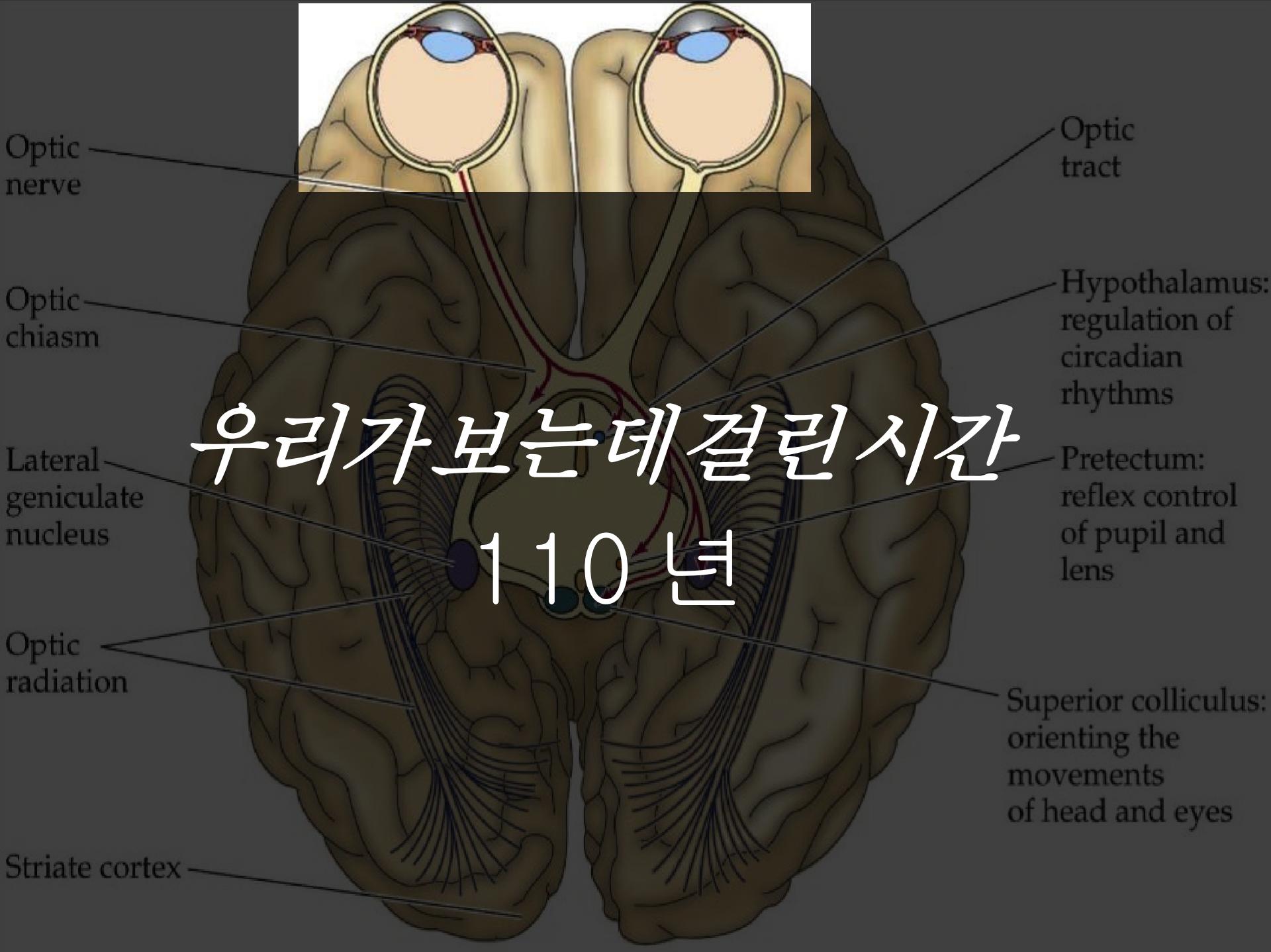
그렇다면, 컴퓨터의 관점에서 볼까요?





이제, 둘을 구분할 수 있겠나요?





우리가 본 것을 이해하는데 걸린 시간

540만년



단순한 숫자들의 나열로부터  
어떻게 이미지를 이해할 수 있을까요?

해결의 출발점은 퍼즐을 맞추듯



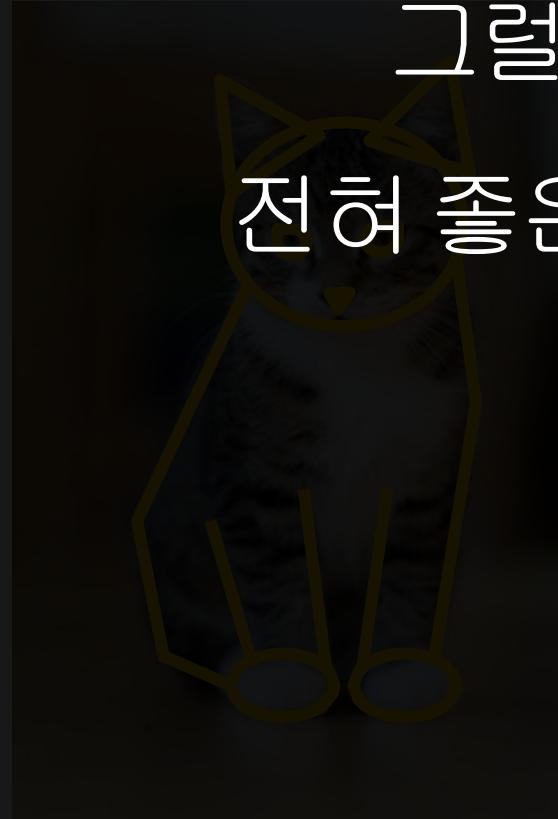
간단한 원리에서 출발합니다



173 173 173 172 170 170 170 170 170 170 170  
169 168 167 166 166 166 172 173 173 173 174  
174 175 175 175 177 177 178 178 178 178 179  
174 174 174 174 174 174 174 174 174 174 172  
171 170 170 170 170 170 176 176 176 176 178  
180 181 181 181 181 181 183 183 182 182 176  
176 176 176 176 176 176 175 175 175 172 172  
179 179 179 180 180 180 183 183 183 184 184  
185 186 187 187 187 187 182 180 180 180 180  
180 180 180 179 180 177 176 176 182 182 182  
183 183 184 184 184 184 183 183 187 188 189 190  
190 189 185 185 183 183 183 183 183 183 182  
182 181 180 179 179 179 185 185 185 186 187 187  
187 187 187 189 189 190 192 193 193 194 188  
188 188 188 188 188 188 188 188 187 186 185  
185 185 194 194 194 194 194 194 195 195 196  
196 196 197 197 198 198 198 197 199 199 196 195  
195 195 194 194 194 194 194 194 194 194 194 199  
199 199 199 199 199 199 199 200 200 200 201  
201 202 202 203 201 201 201 201 200 200 199 199  
198 199 199 199 199 199 199 203 203 203 203  
203 203 203 203 203 203 203 203 205 205 206 206  
206 204 203 203 203 203 202 202 202 202 203 203  
203 203 203 203 207 207 207 206 206 206 205  
205 206 207 207 207 208 208 209 209 206 206

“Cat”

그럴듯해 보이는 이 방법은  
전혀 좋은 효과를 거두지 못합니다.



174 175 175 176 178 178 178 179 179 178 178 178  
174 174 174 174 174 174 174 174 174 174 174 172  
170 170 170 170 170 170 170 170 170 170 170 178  
178 178 178 178 178 178 178 178 178 178 178 178  
182 182 182 182 182 182 182 182 182 182 182 180  
180 180 180 180 180 180 180 180 180 180 180 180  
183 183 183 183 183 183 183 183 183 183 183 190  
190 189 189 189 189 189 189 189 189 189 189 182  
182 181 180 179 179 179 185 185 185 186 186 187  
187 187 187 189 189 190 192 193 193 194 188  
188 188 188 188 188 188 188 188 188 187 186 185  
185 185 194 194 194 194 194 194 194 195 195 196  
196 196 197 197 198 198 197 199 199 195 196 195  
195 194 194 195 194 194 194 194 194 194 194 199  
199 199 199 199 199 199 199 200 200 200 200 201  
201 202 202 202 201 201 202 200 200 199 199  
198 199 199 199 199 199 199 200 200 200 200 201  
203 203 203 203 204 204 204 204 205 205 206 206  
206 204 203 203 203 202 202 202 202 203 203 203  
203 203 203 203 207 207 207 207 206 206 206 205  
205 206 207 207 207 207 208 208 209 209 206 206

“Cat”

그럴듯해 보이는 이 방법은  
전혀 좋은 효과를 거두지 못합니다.

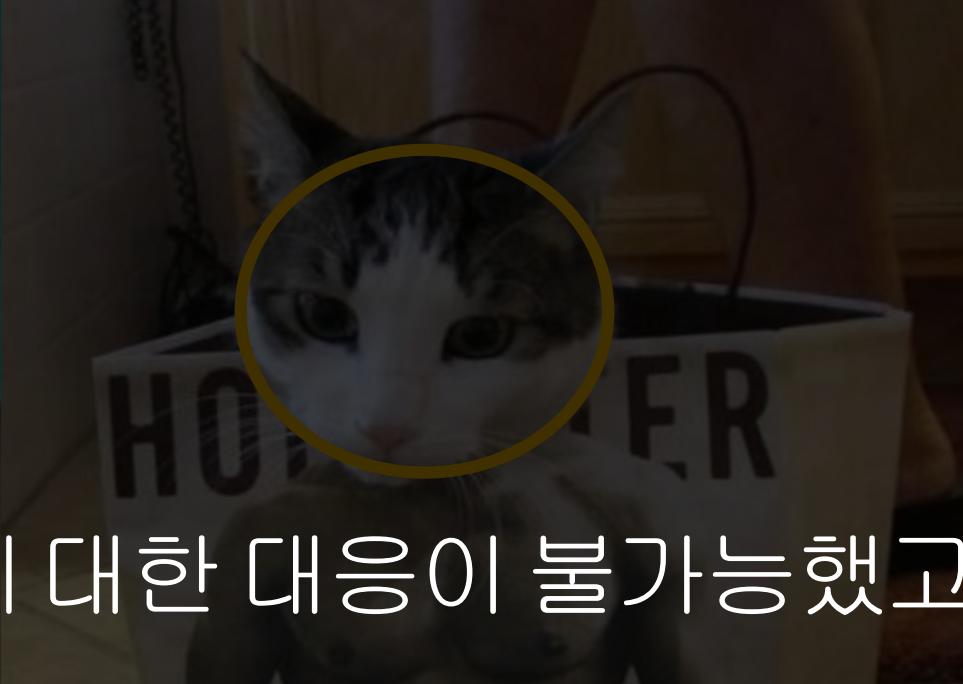
왜?

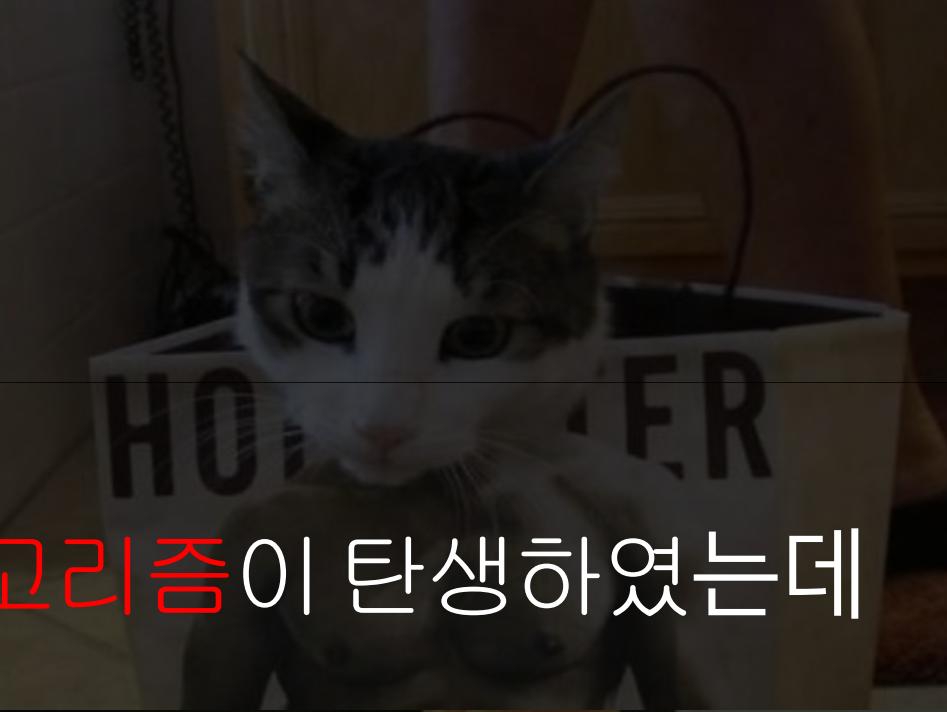
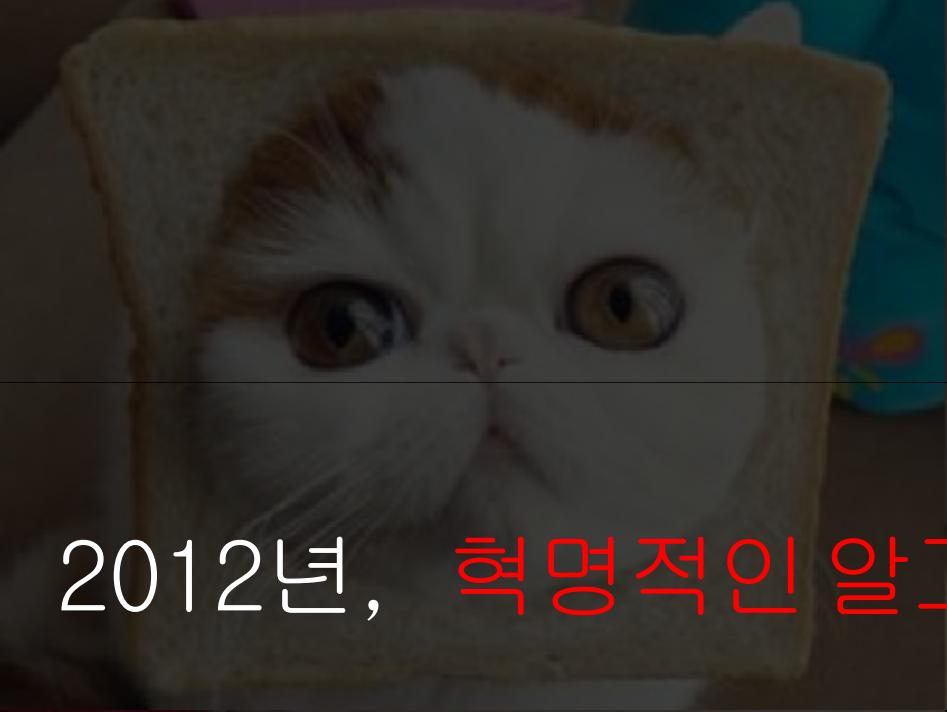
“Cat”



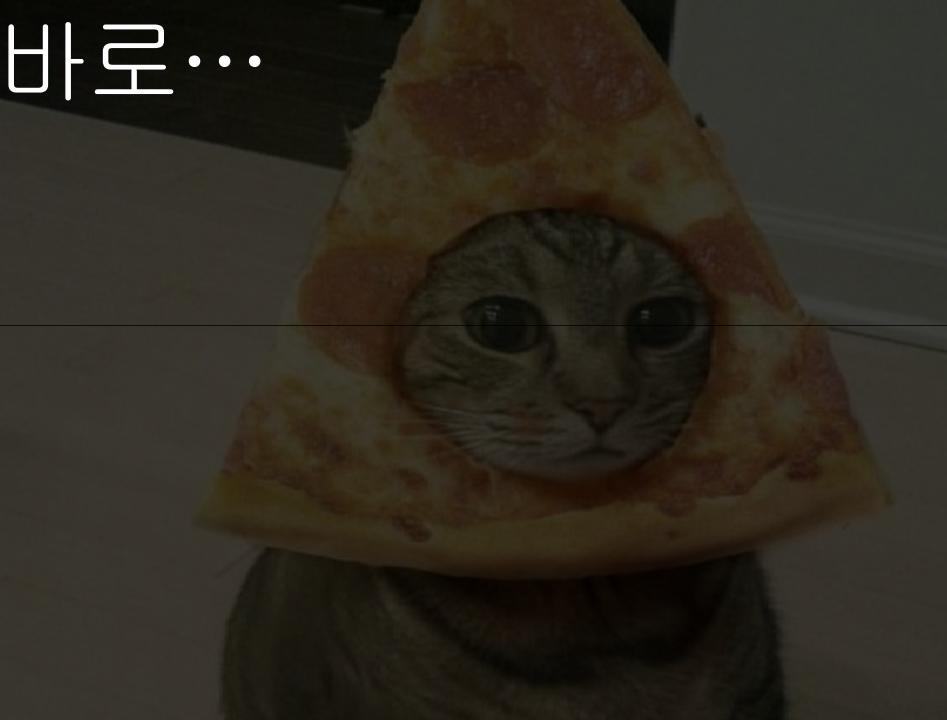


- 패턴은 예외 상황에 대한 대응이 불가능했고 결국 성능은 매우 미미하였습니다.

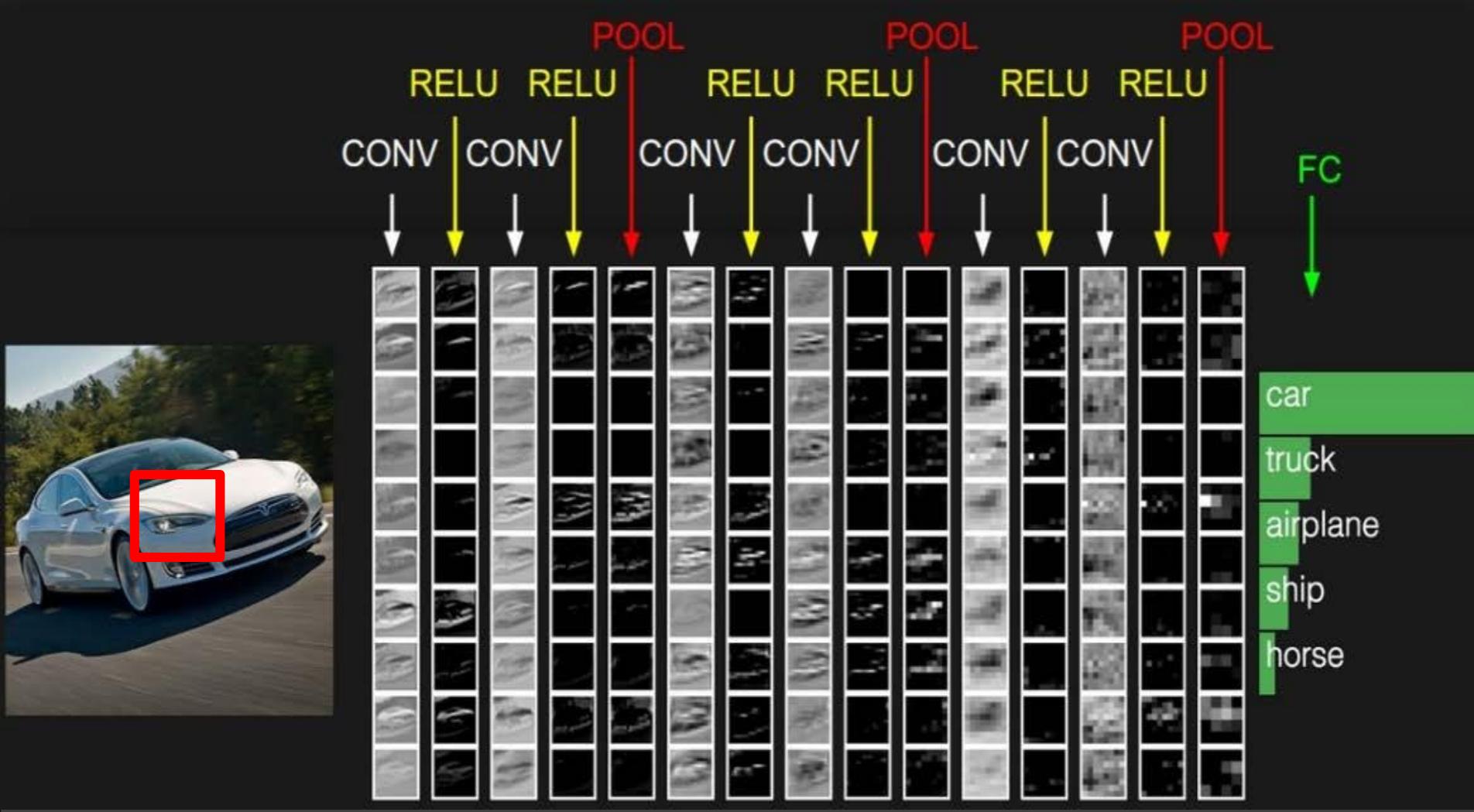




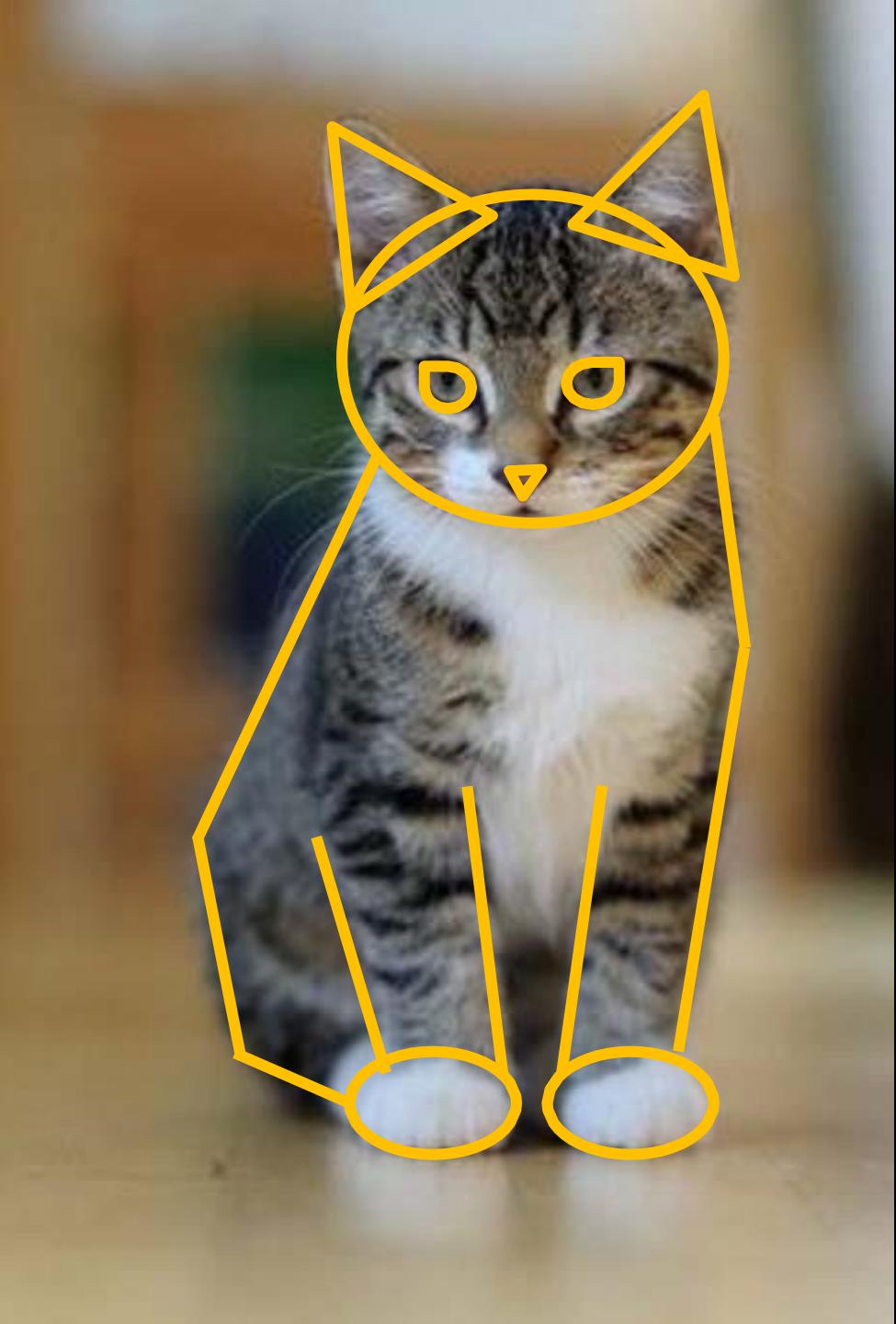
2012년, 혁명적인 알고리즘이 탄생하였는데



그것은 바로…



# Convolutional Neural Networks



예외 상황에  
취약했던 기존의  
알고리즘에 반해



CNN은 이렇게 작은  
filter를 순환시키며



CNN은 이렇게 작은  
filter를 순환시키며



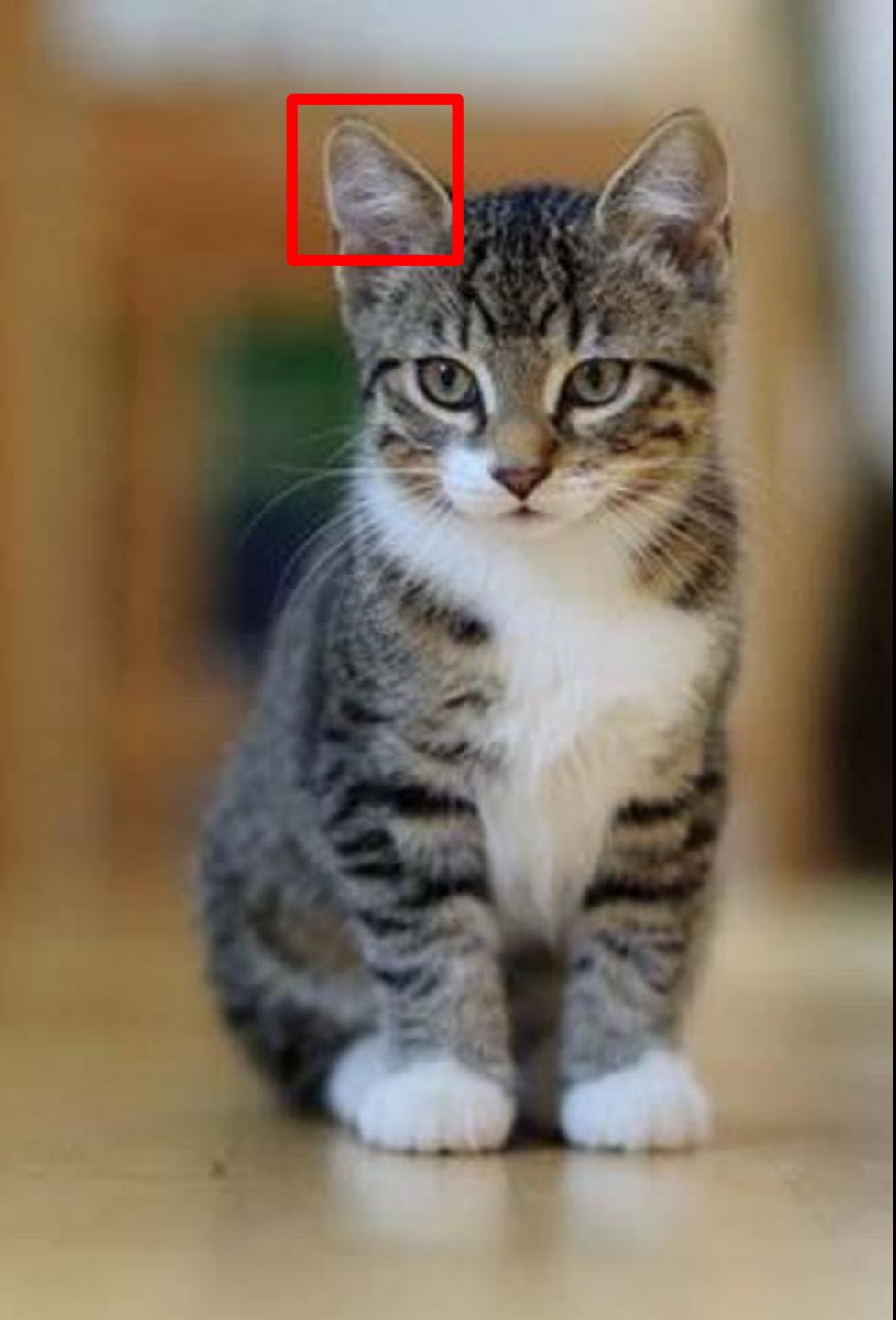
CNN은 이렇게 작은  
filter를 순환시키며



CNN은 이렇게 작은  
filter를 순환시키며

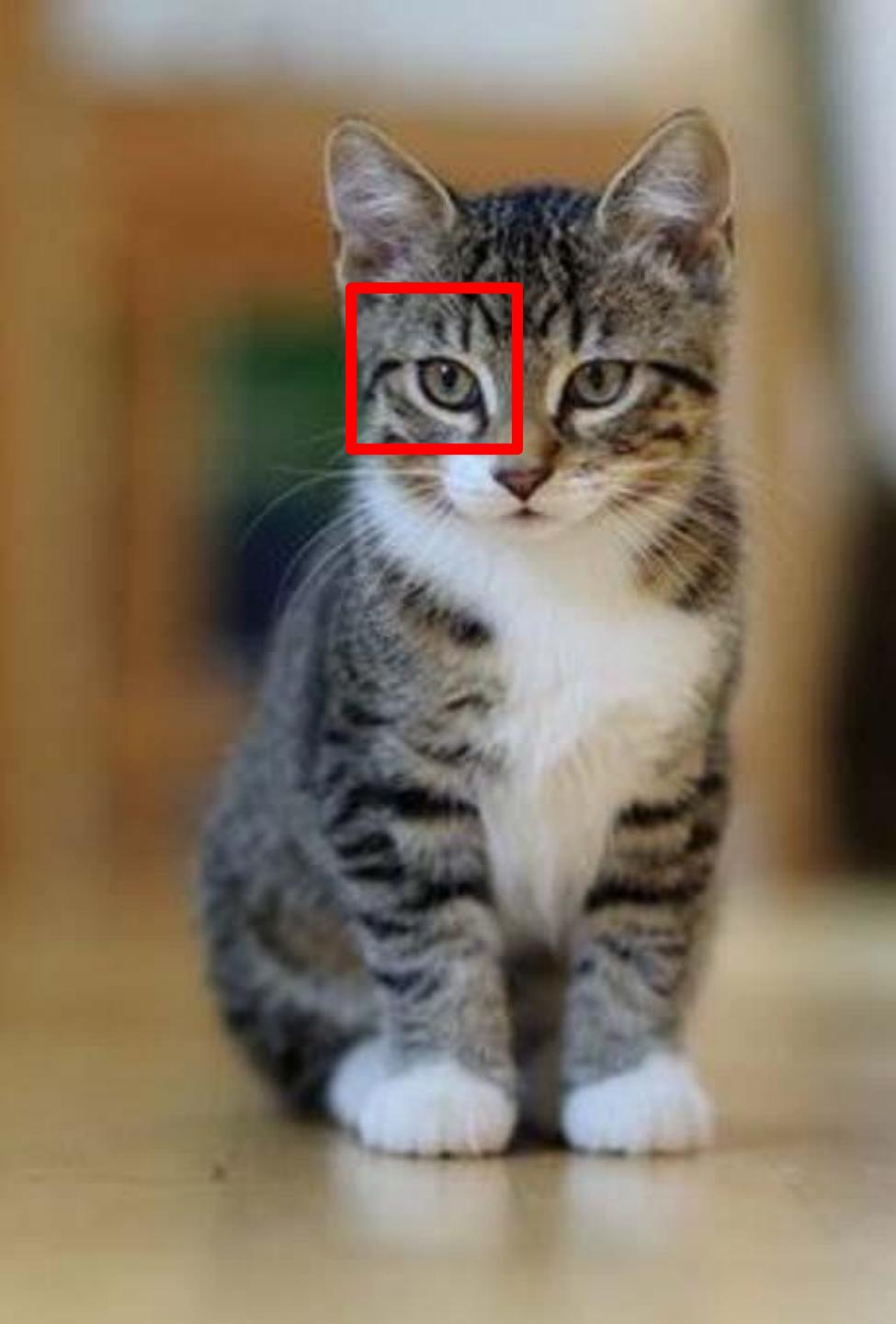


CNN은 이렇게 작은  
filter를 순환시키며



CNN은 이렇게 작은  
filter를 순환시키며

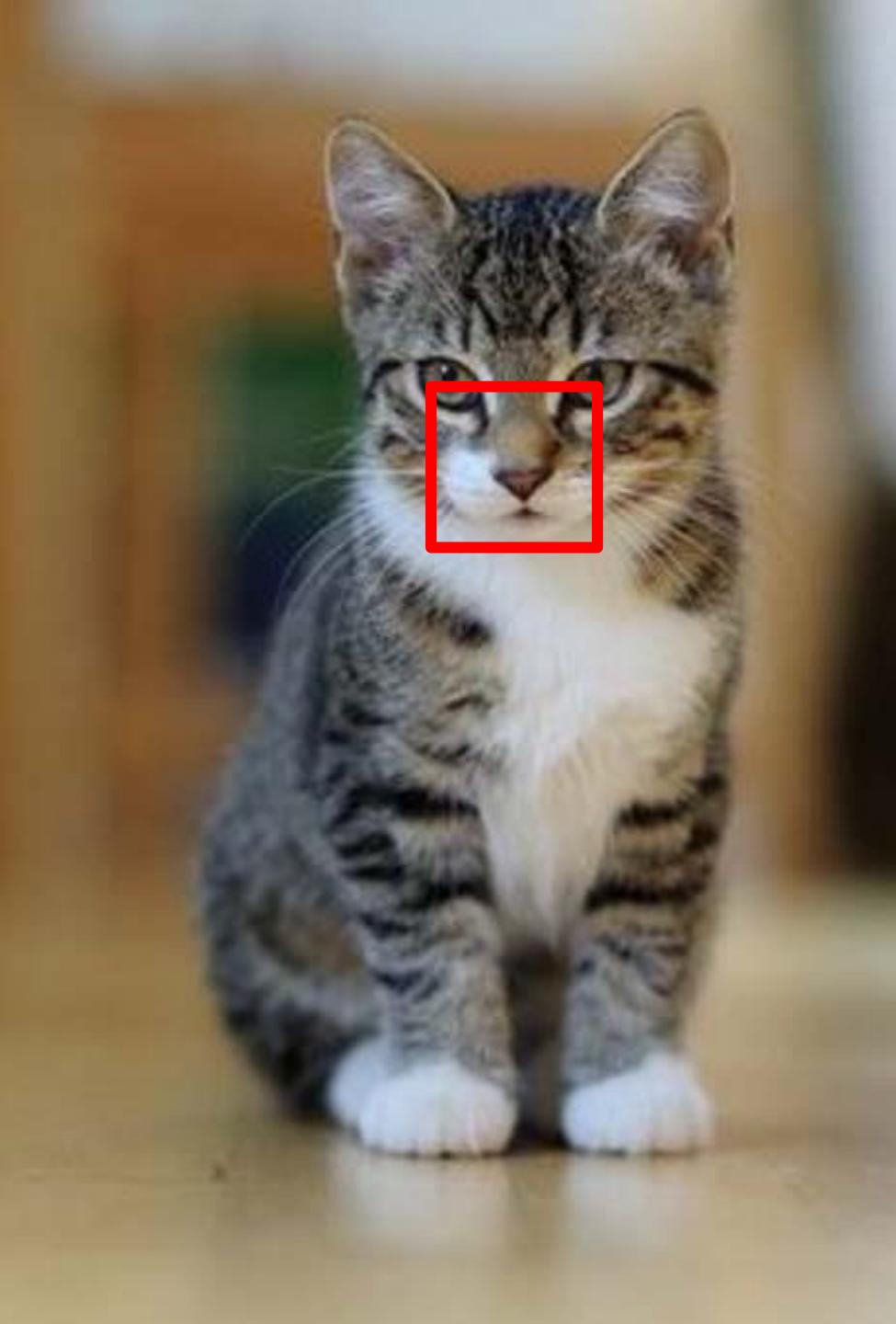
‘음 ? 고양이 귀다 ! ’



CNN은 이렇게 작은  
filter를 순환시키며

‘음 ? 고양이 귀다 ! ’

‘어 ? 고양이 눈이다 ! ’



CNN은 이렇게 작은  
filter를 순환시키며

‘음 ? 고양이 귀다 ! ’

‘어 ? 고양이 눈이다 ! ’

‘오 ? 고양이 코다 ! ’

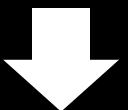


- CNN은 이렇게 작은  
filter를 순환시키며
- ‘음 ? 고양이 귀다 !’
  - ‘어 ? 고양이 눈이다 !’
  - ‘오 ? 고양이 코다 !’
  - ‘고양이 발이네 ?’



• 고양이

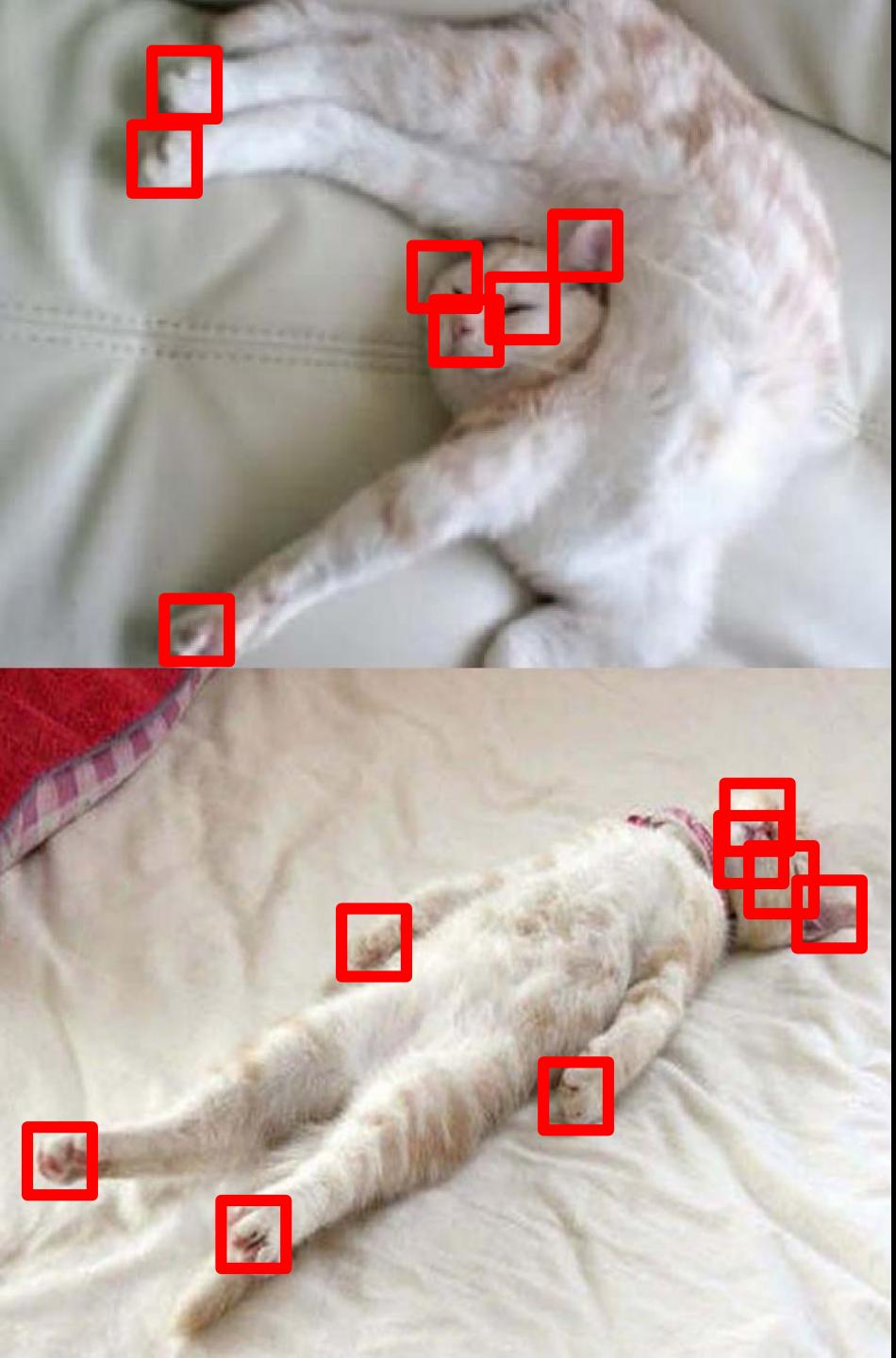
- CNN은 이렇게 작은  
filter를 순환시키며
- ‘음 ? 고양이 귀다 !’
  - ‘어 ? 고양이 눈이다 !’
  - ‘오 ? 고양이 코다 !’
  - ‘오호 , 고양이 발이네 ?’



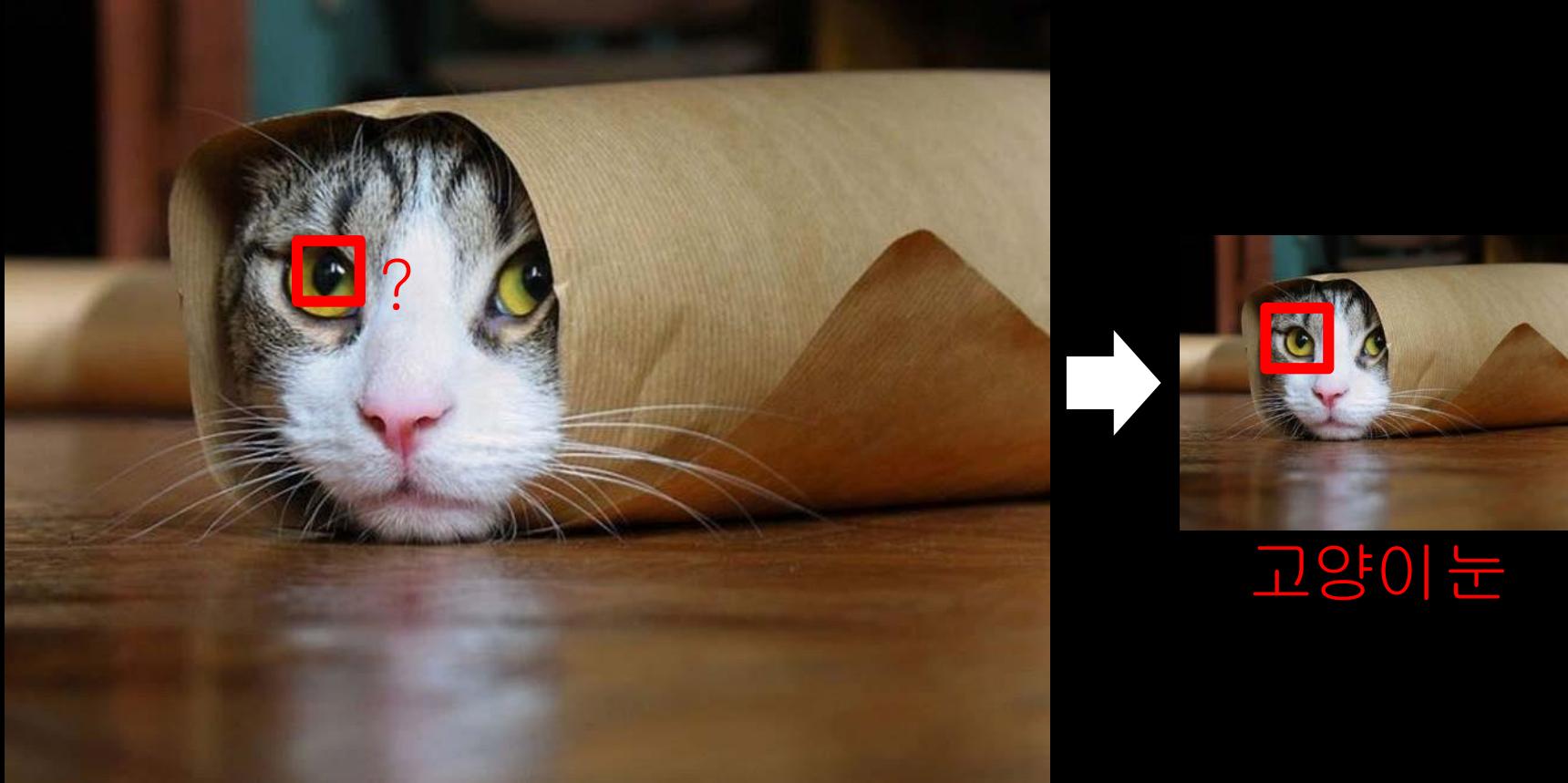
- “이건 고양이군 !”



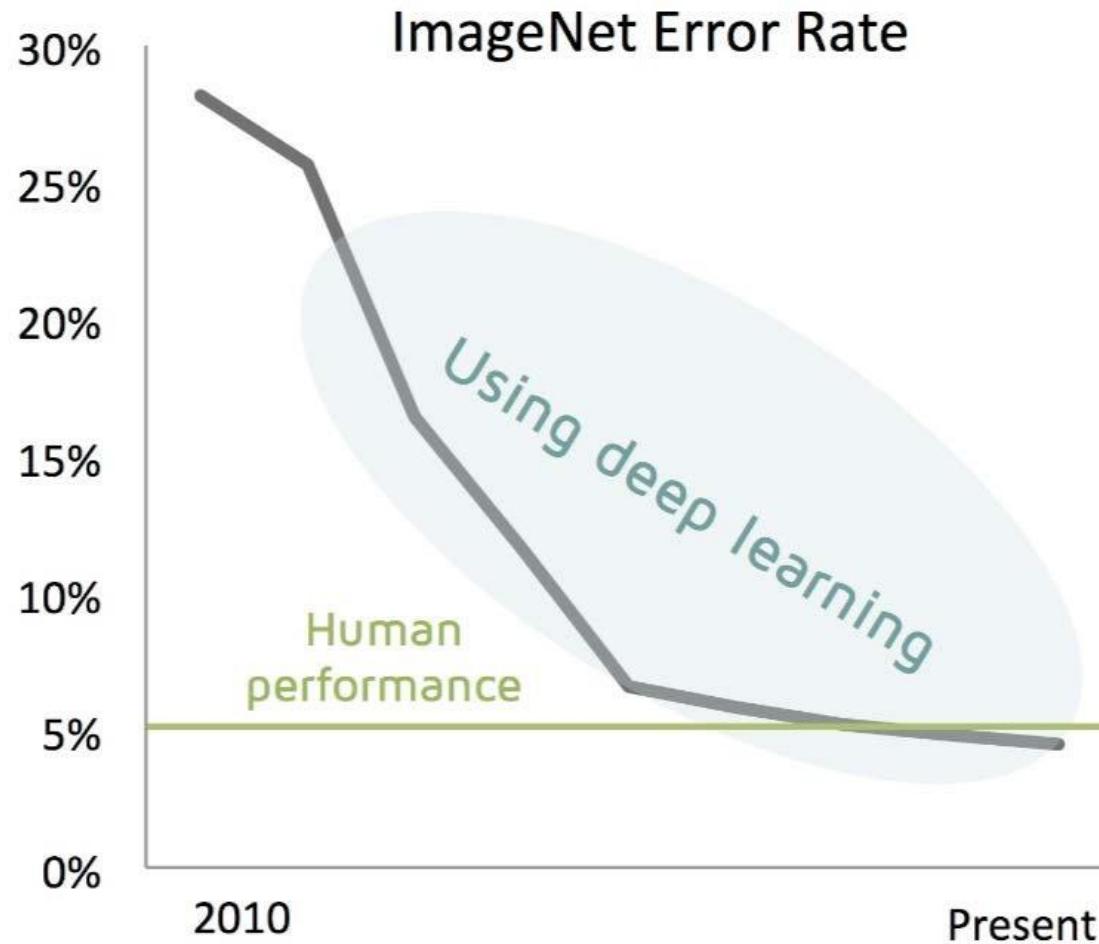
이런 방식을 통해  
각 특징들이 기묘하게  
분포되어 있어도



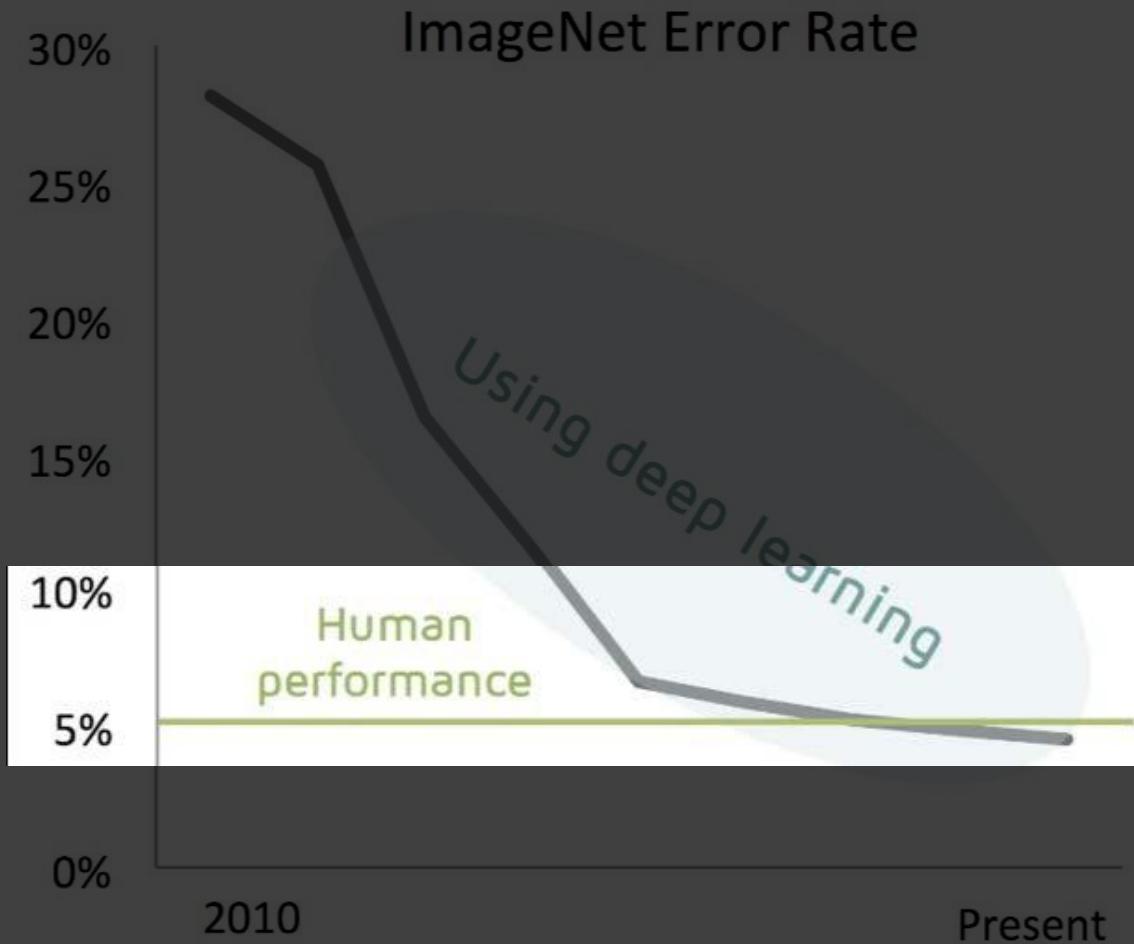
이런 방식을 통해  
각 특징들이 기묘하게  
분포되어 있어도  
정확하게 찾아내는  
높은 적응도를 갖습니다



또한 striding 기법을 통해 다양한 크기의  
물체에도 높은 적응도를 갖습니다.

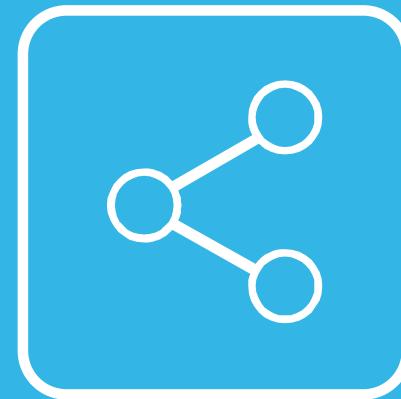


CNN(통틀어 Deep Learning)



GPU의 등장에 힘 입어 발달은 가속화되었고,  
급기야 **인간 성능**을 뛰어넘기에 이르렀습니다

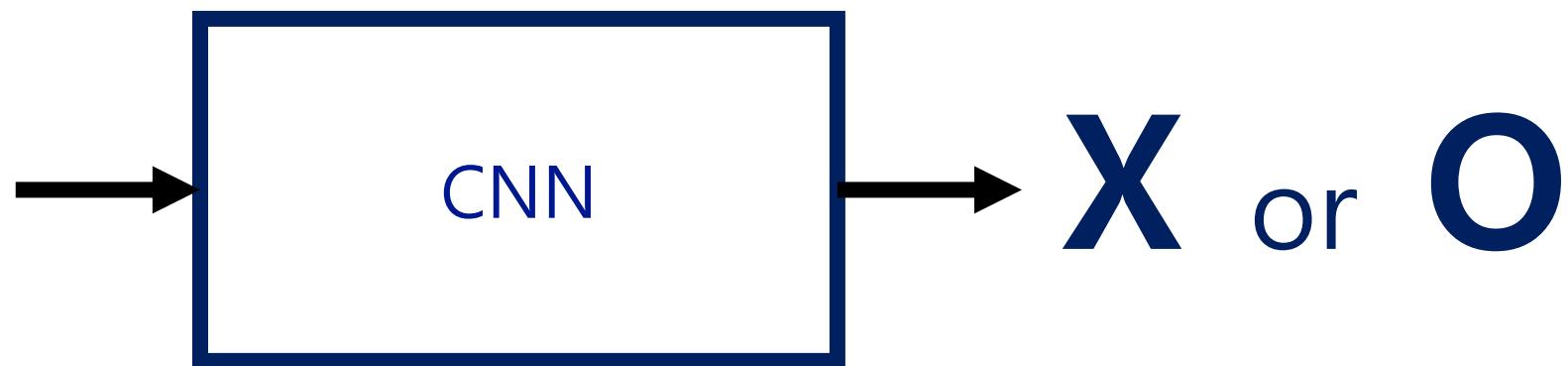
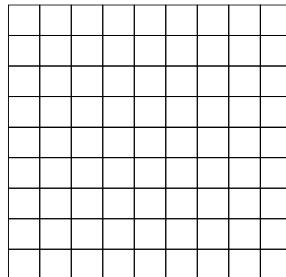
# 02 Convolutional Neural Network



# A Toy ConvNet: X's and O's

Says whether a picture is of an X or an O

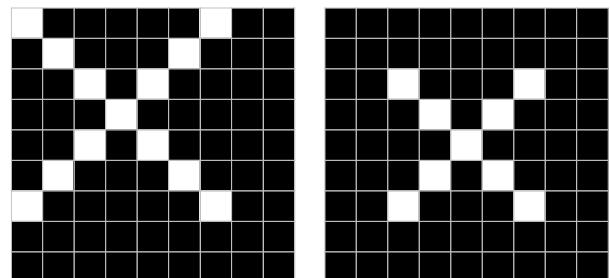
A two-dimensional  
array of pixels



For example



# Trickier cases

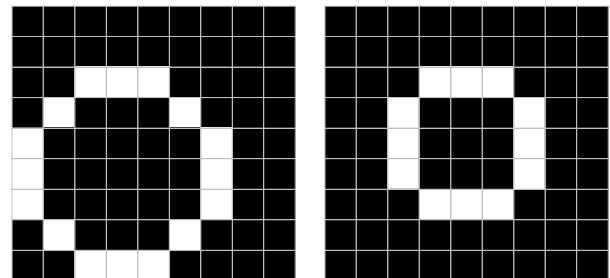


translation

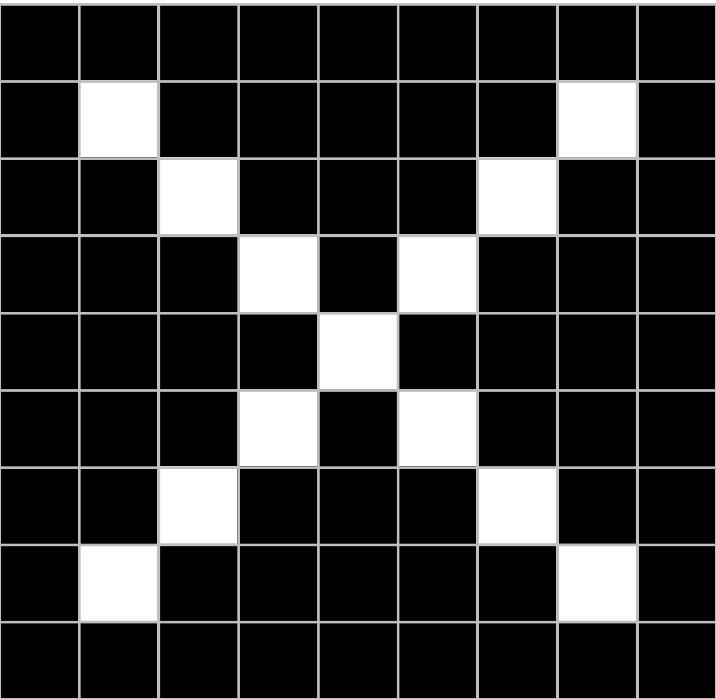
scaling

rotation

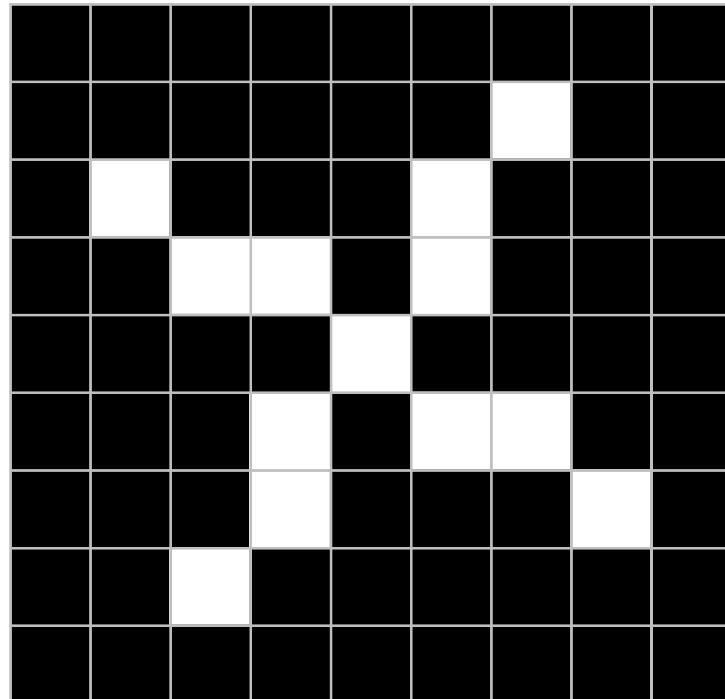
weight



# Deciding is hard



?



# What computers see

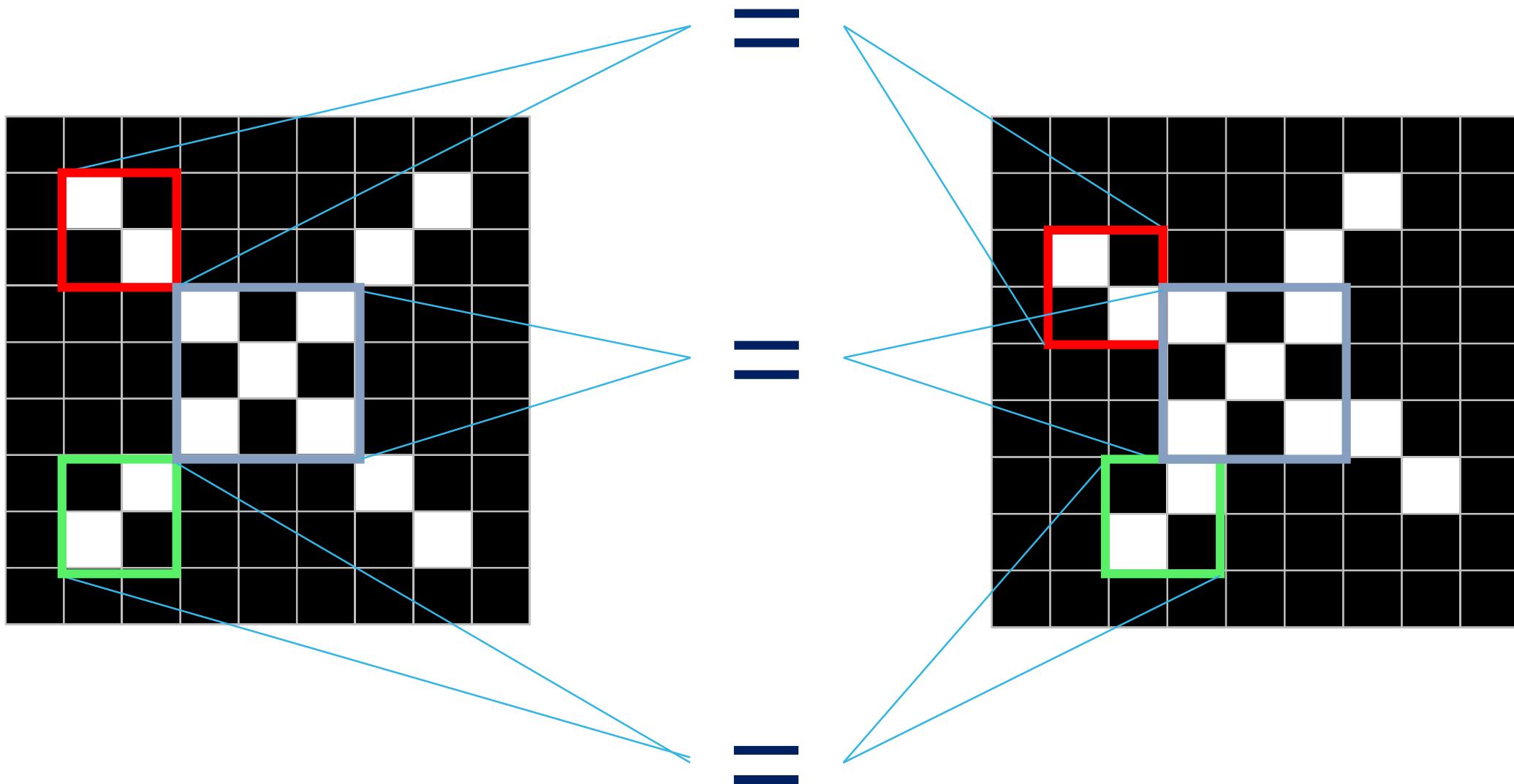


# What computers see

# Computers are literal



# ConvNets match pieces of the image



# Features match pieces of the image

1	-1	-1
-1	1	-1
-1	-1	1

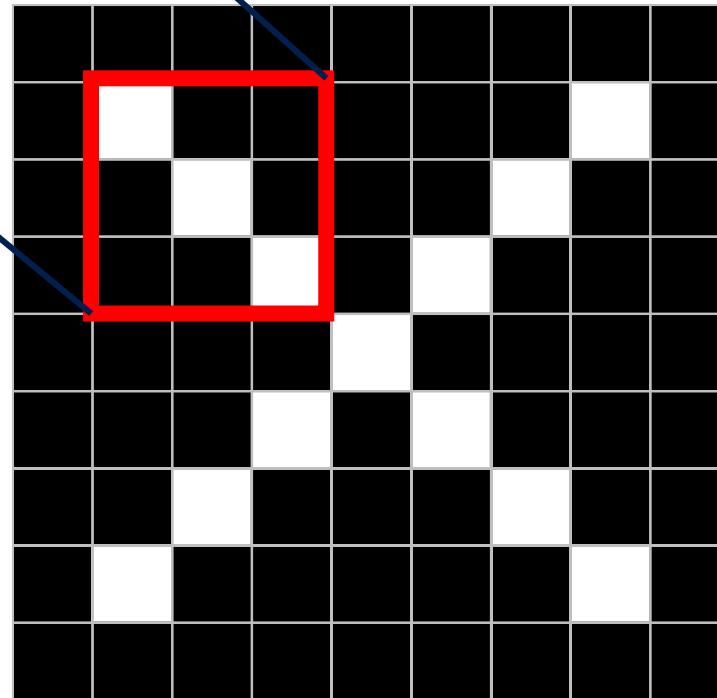
1	-1	1
-1	1	-1
1	-1	1

-1	-1	1
-1	1	-1
1	-1	-1

1	-1	-1
-1	1	-1
-1	-1	1

1	-1	1
-1	1	-1
1	-1	1

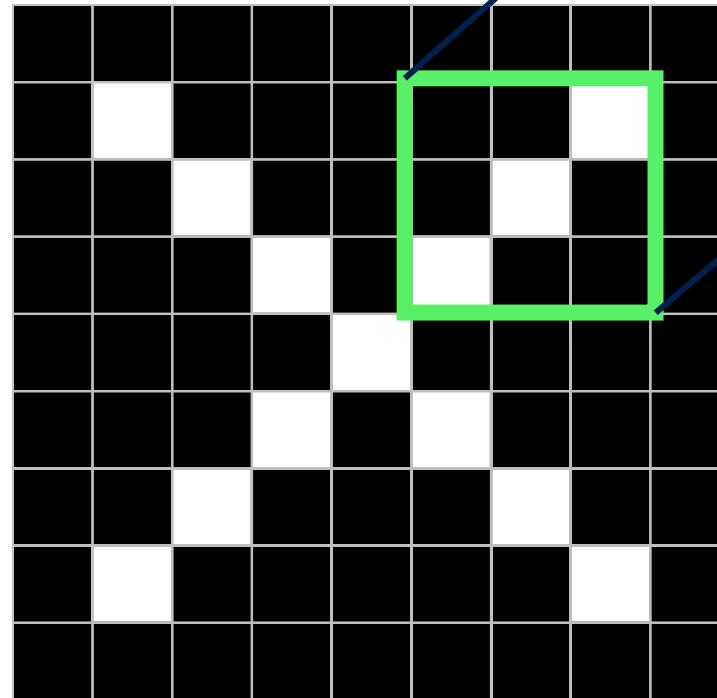
-1	-1	1
-1	1	-1
1	-1	-1



1	-1	-1
-1	1	-1
-1	-1	1

1	-1	1
-1	1	-1
1	-1	1

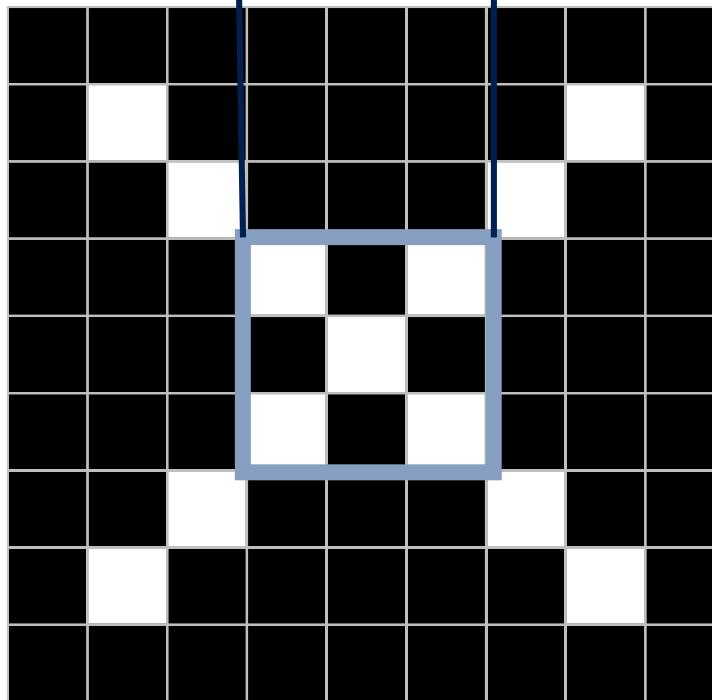
-1	-1	1
-1	1	-1
1	-1	-1



1	-1	-1
-1	1	-1
-1	-1	1

1	-1	1
-1	1	-1
1	-1	1

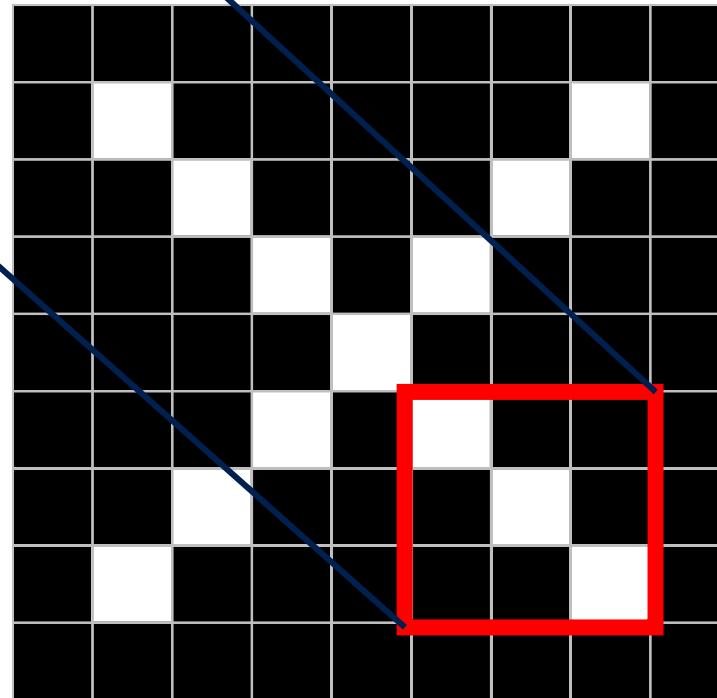
-1	-1	1
-1	1	-1
1	-1	-1



1	-1	-1
-1	1	-1
-1	-1	1

1	-1	1
-1	1	-1
1	-1	1

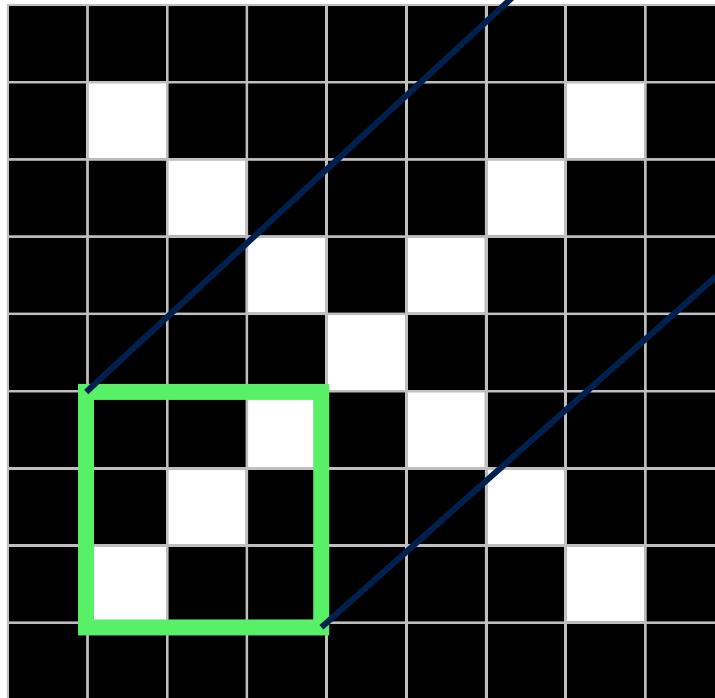
-1	-1	1
-1	1	-1
1	-1	-1



1	-1	-1
-1	1	-1
-1	-1	1

1	-1	1
-1	1	-1
1	-1	1

-1	-1	1
-1	1	-1
1	-1	-1



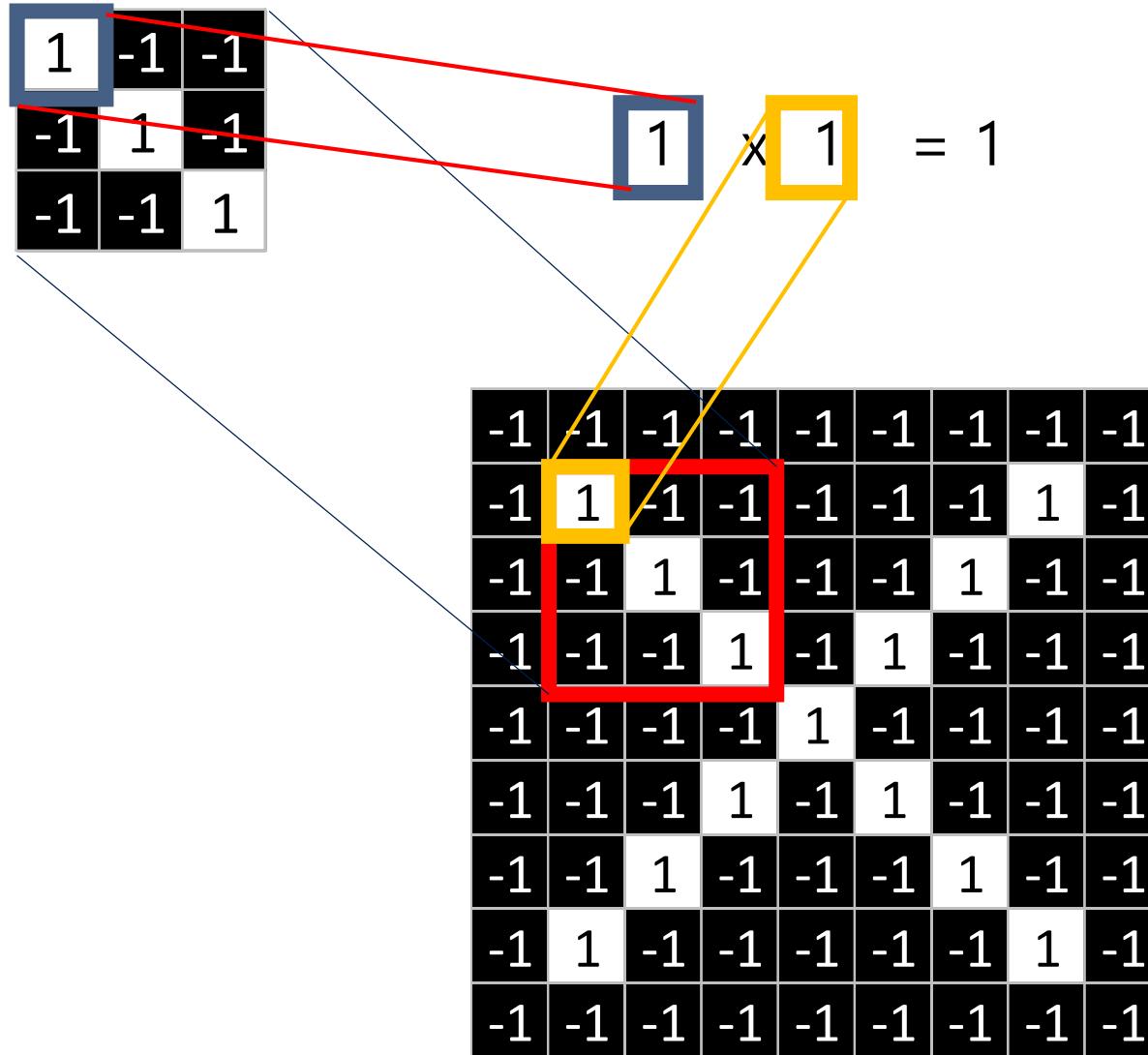
# Filtering: The math behind the match

1	-1	-1
-1	1	-1
-1	-1	1

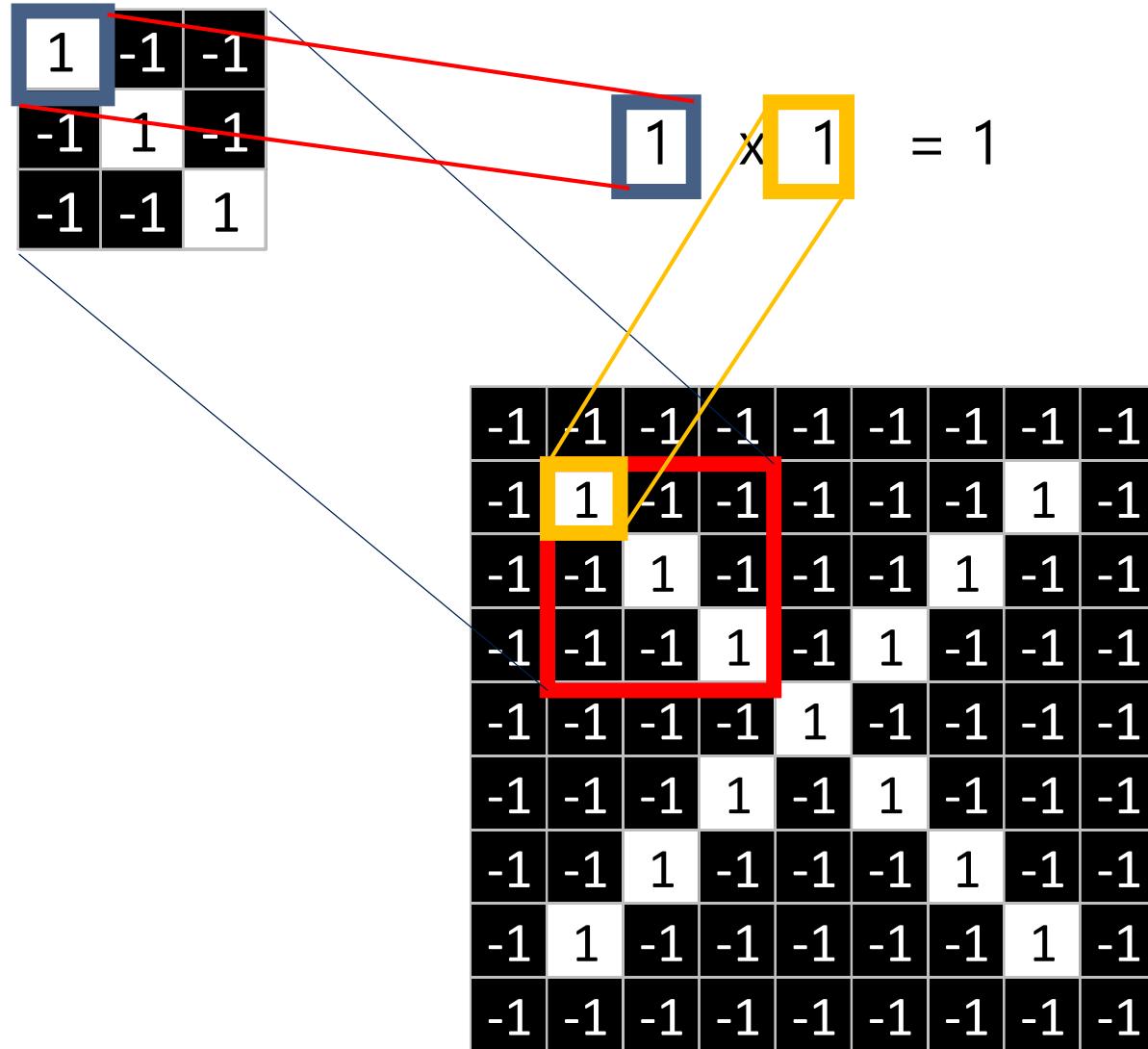
# Filtering: The math behind the match

1. Line up the feature and the image patch.
2. Multiply each image pixel by the corresponding feature pixel.
3. Add them up.
4. Divide by the total number of pixels in the feature. (optional)

# Filtering: The math behind the match



# Filtering: The math behind the match



# Filtering: The math behind the match

1	-1	-1
-1	1	-1
-1	-1	1

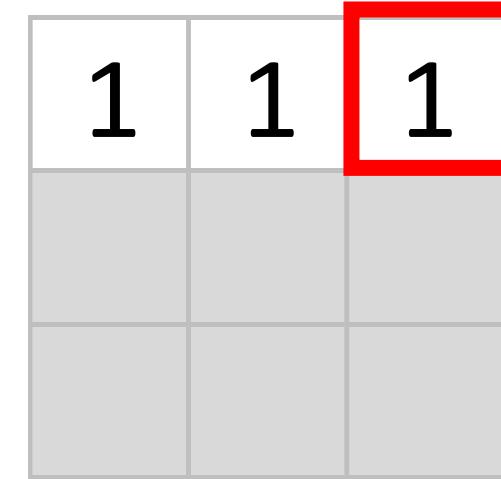
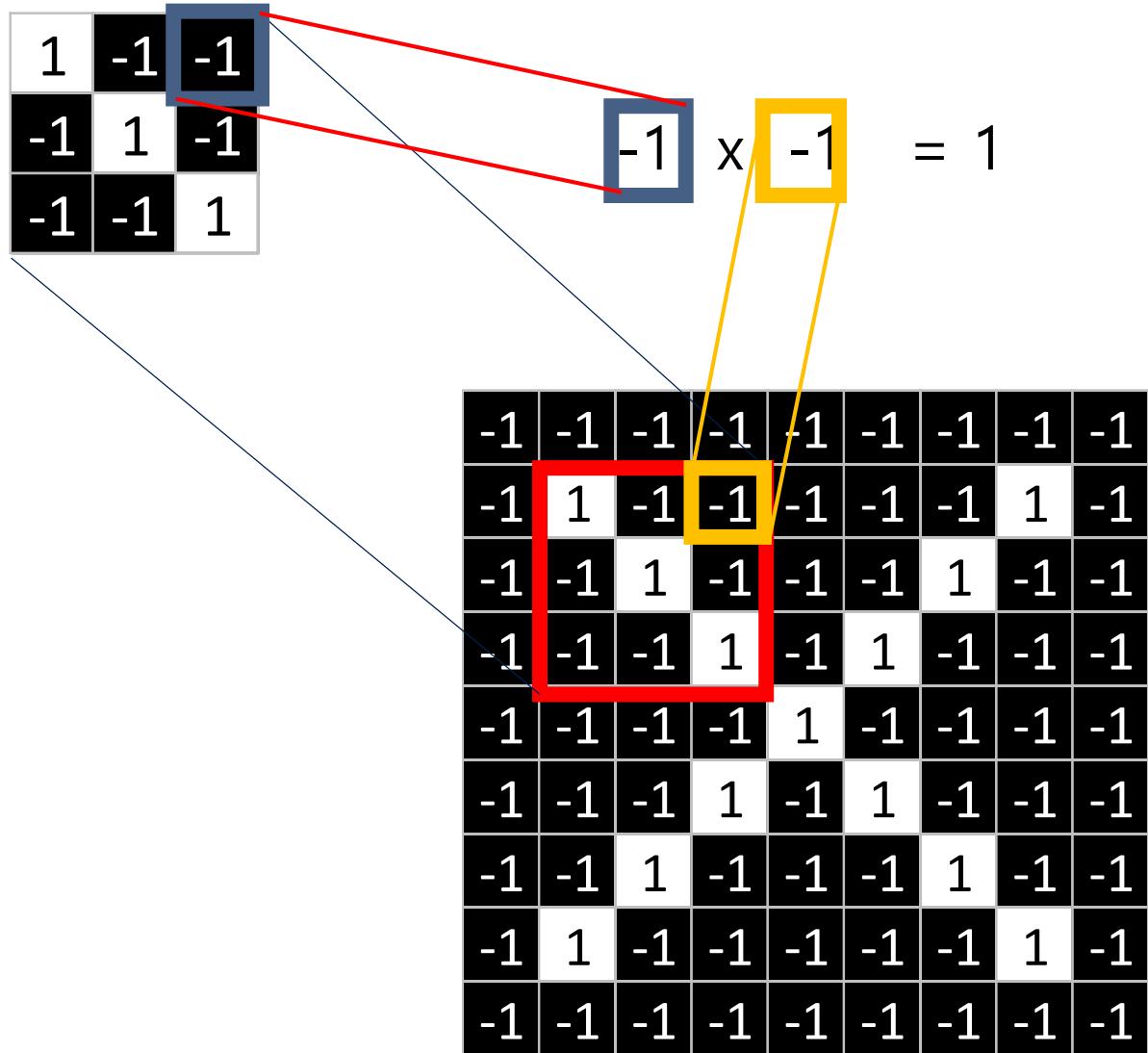
-1
----

$$\times \begin{matrix} -1 \end{matrix} = 1$$

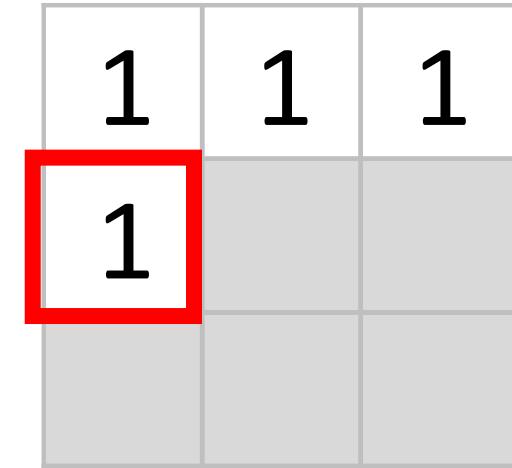
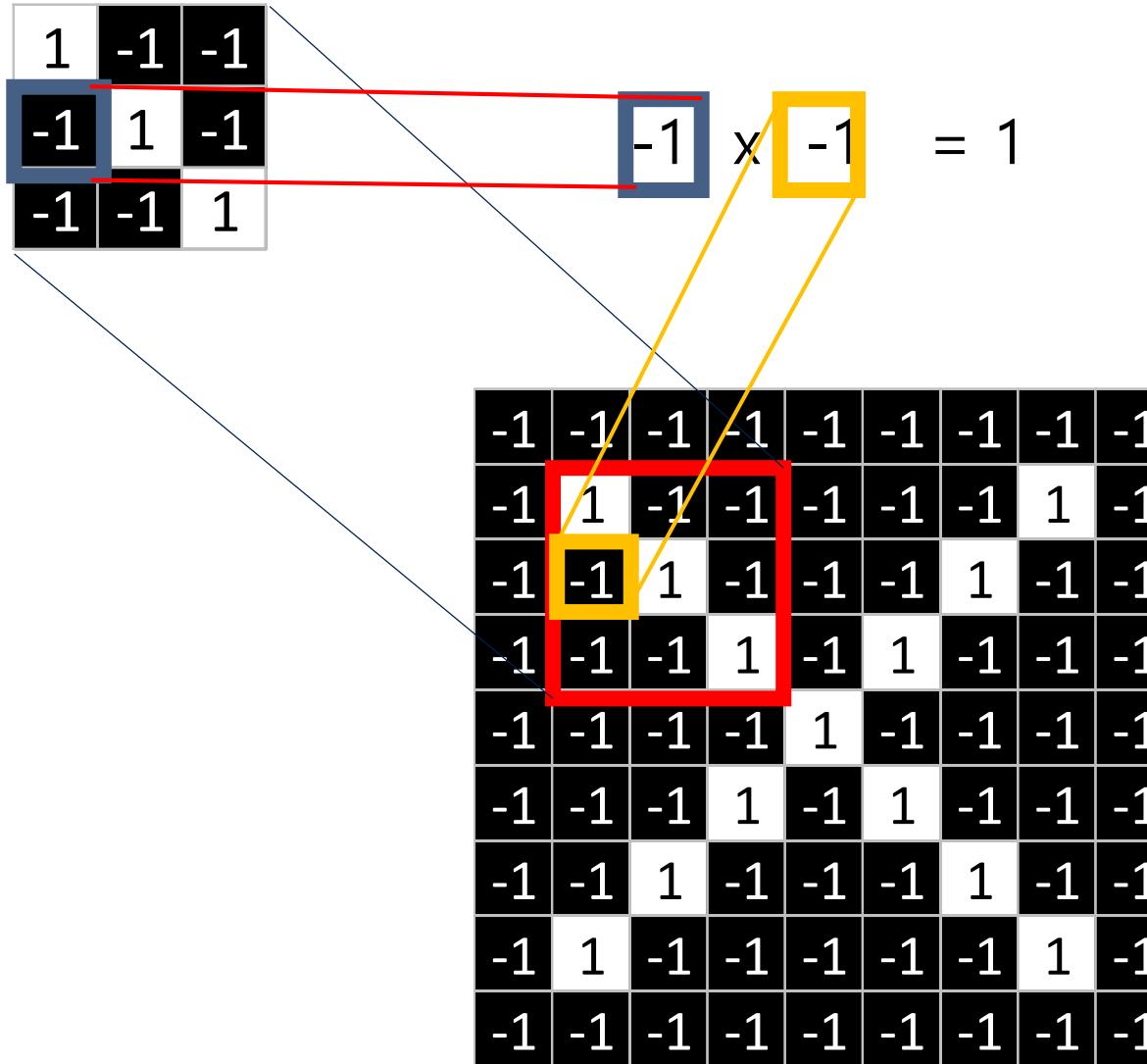
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1	1

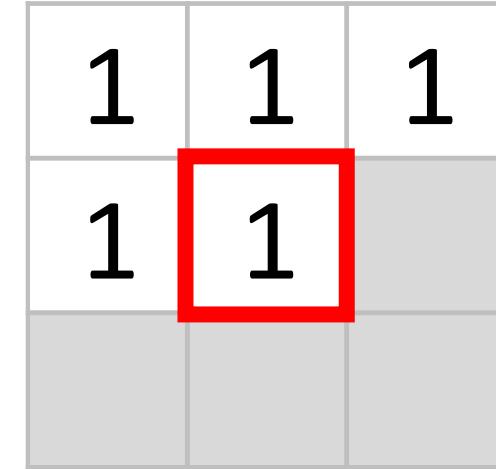
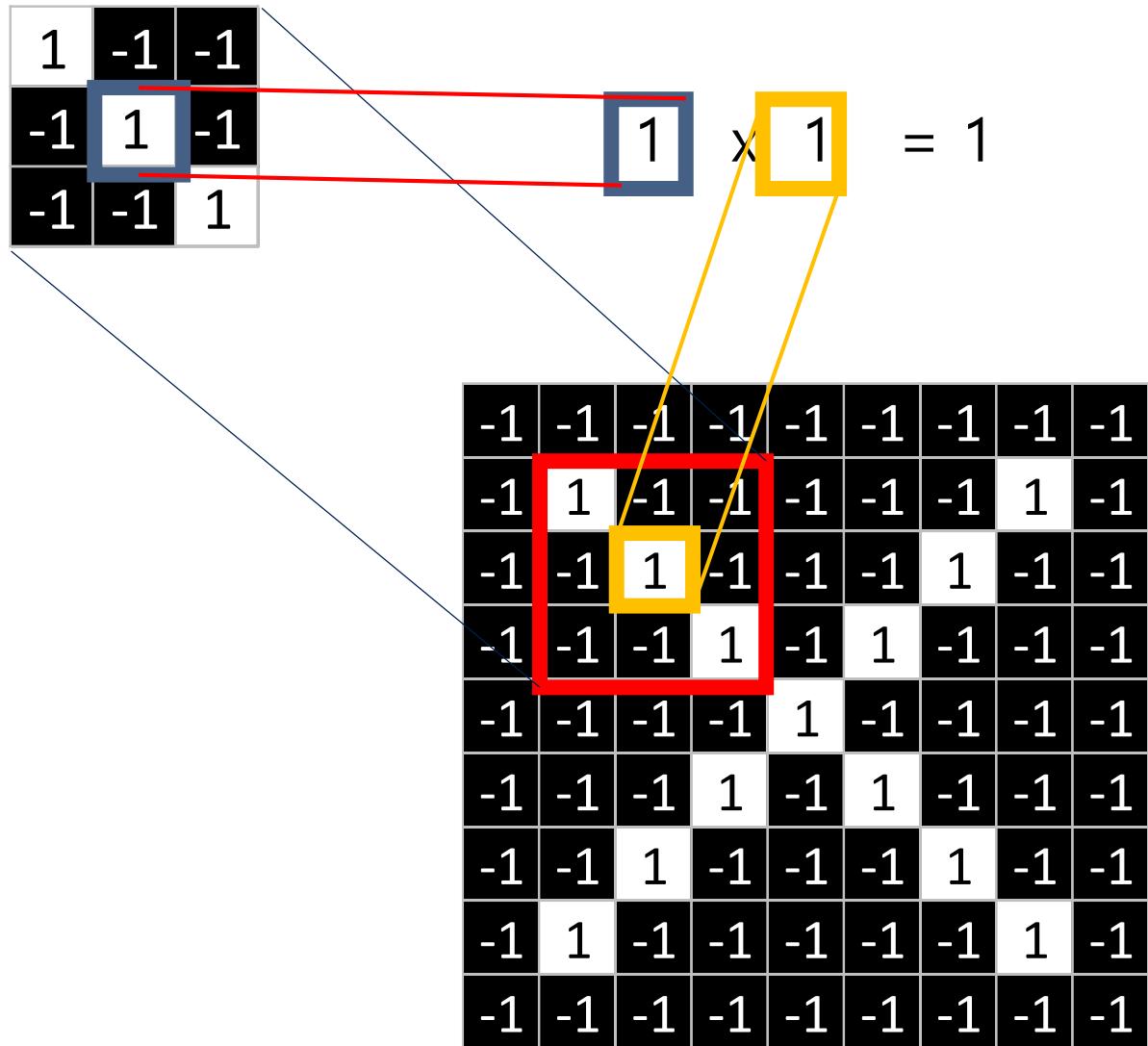
# Filtering: The math behind the match



# Filtering: The math behind the match



# Filtering: The math behind the match



# Filtering: The math behind the match

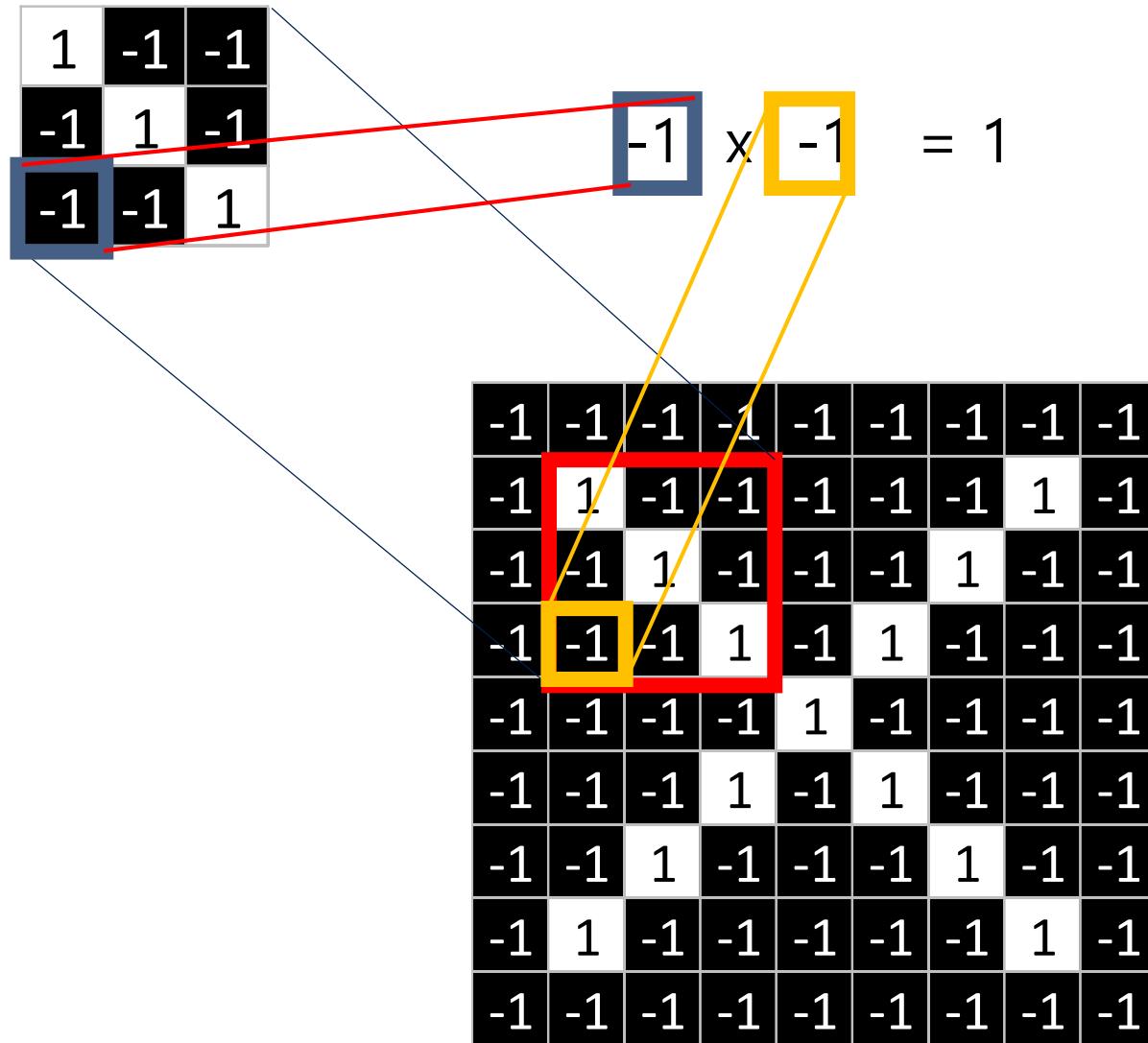
1	-1	-1
-1	1	-1
-1	-1	1

$$-1 \times -1 = 1$$

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

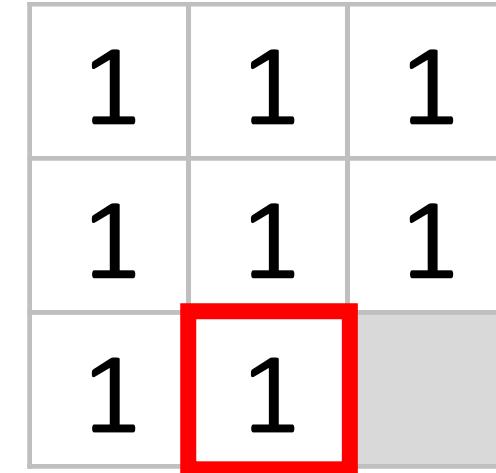
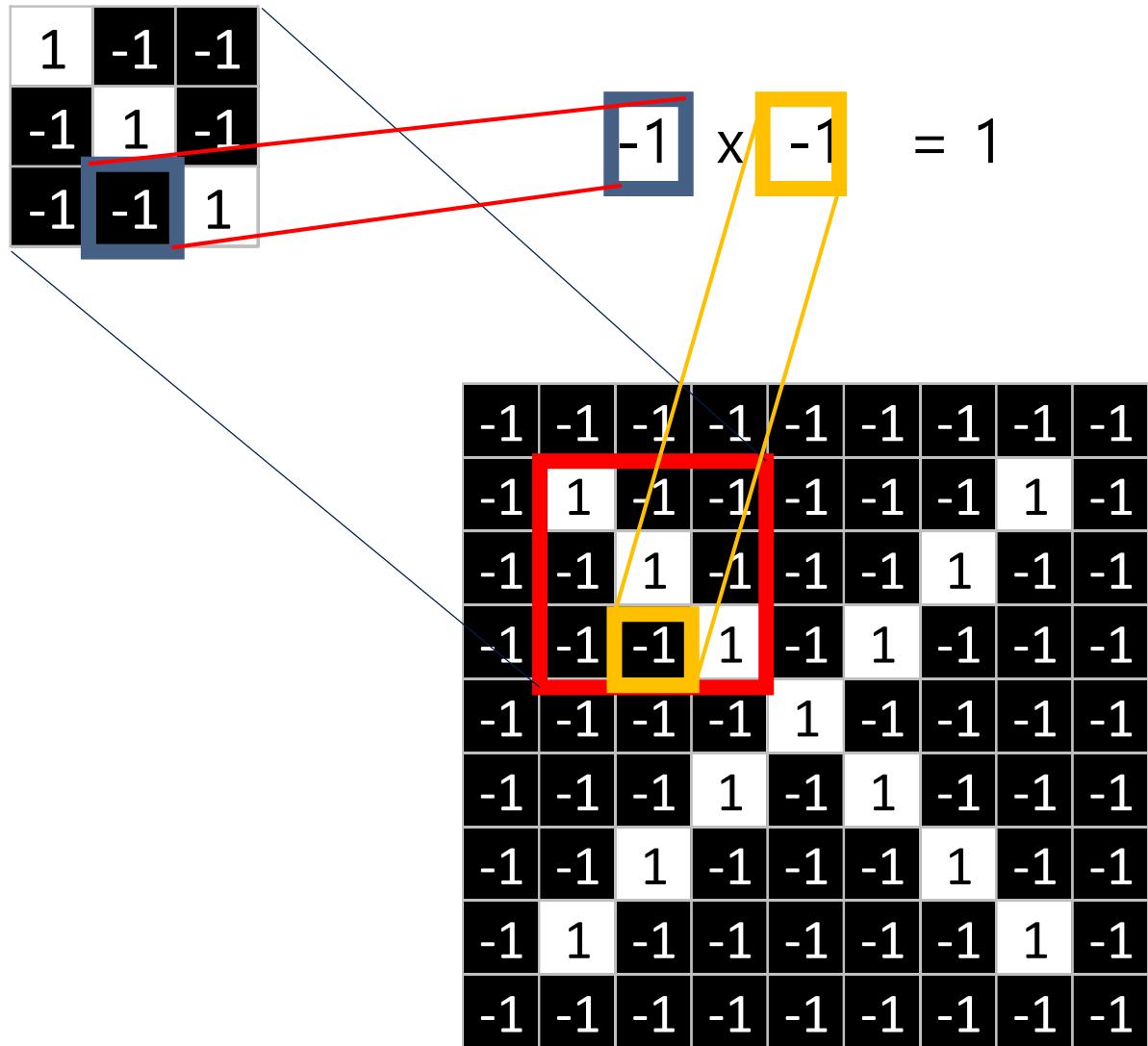
1	1	1
1	1	1

# Filtering: The math behind the match

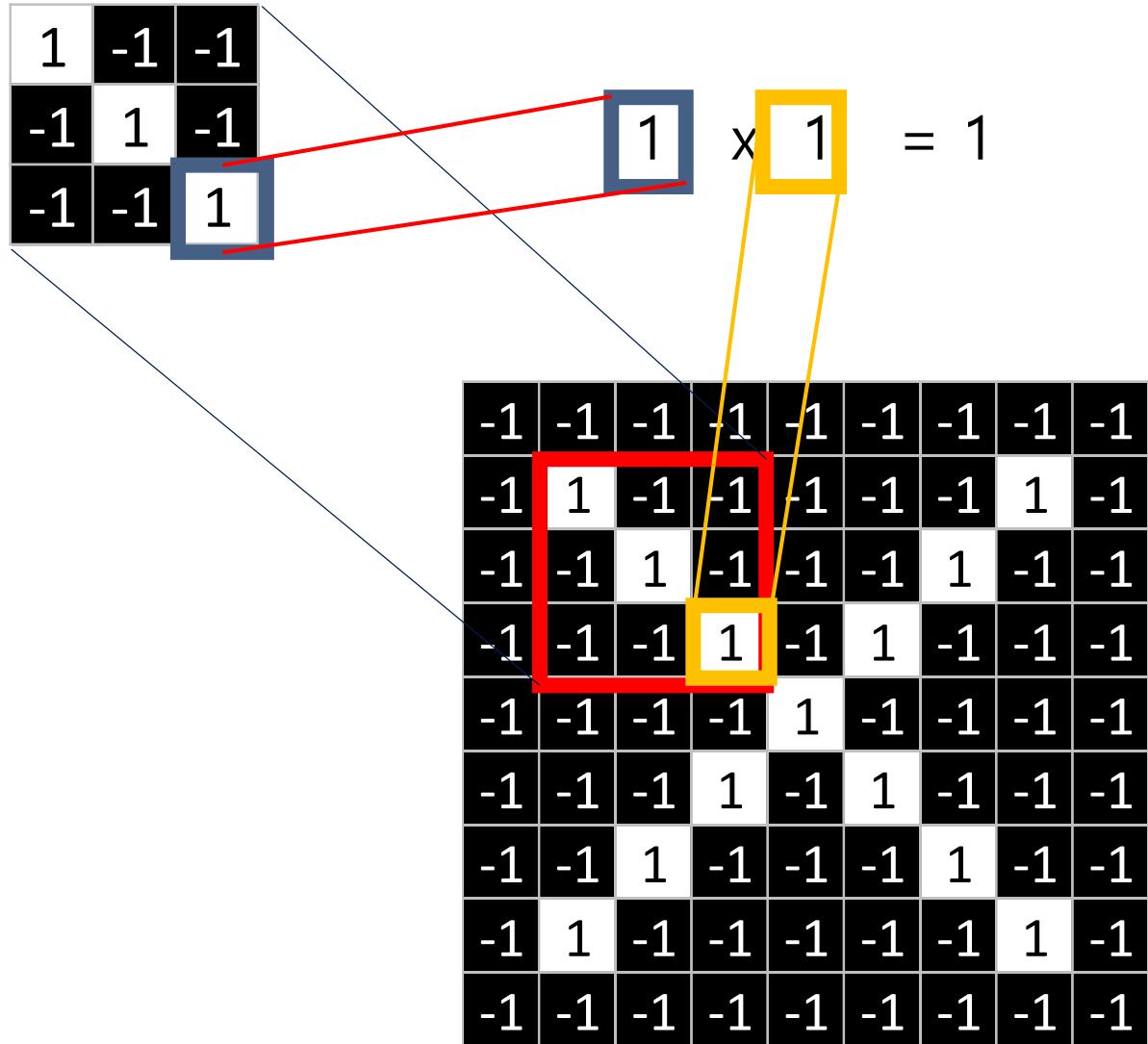


1	1	1
1	1	1
1		

# Filtering: The math behind the match



# Filtering: The math behind the match



1	1	1
1	1	1
1	1	1

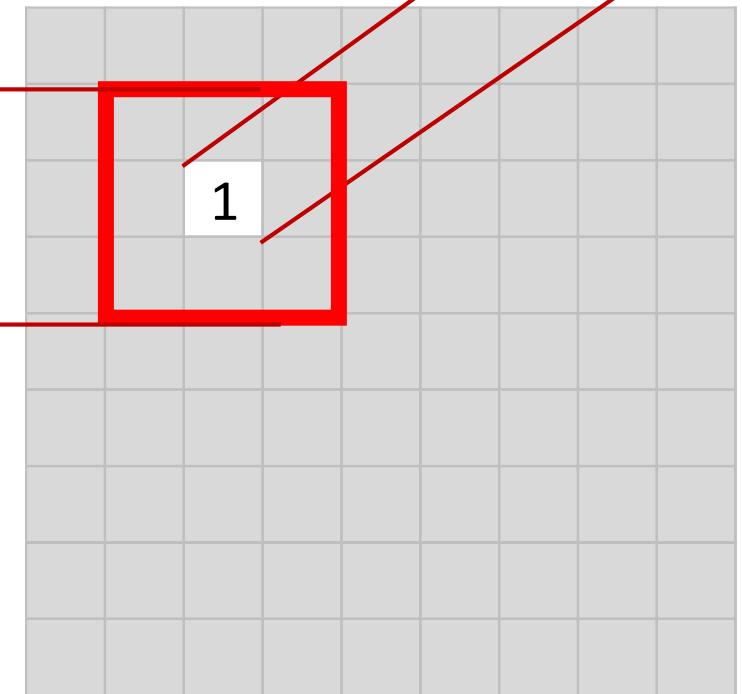
# Filtering: The math behind the match

1	-1	-1
-1	1	-1
-1	-1	1

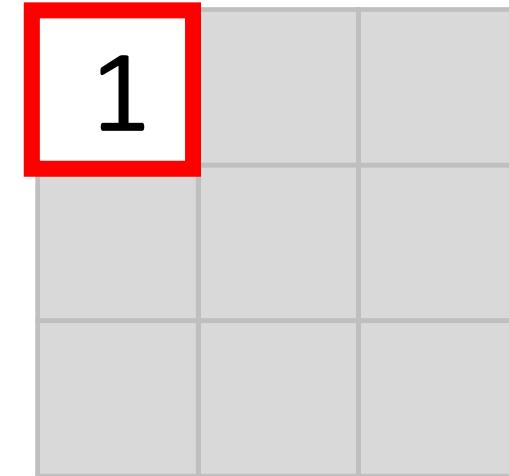
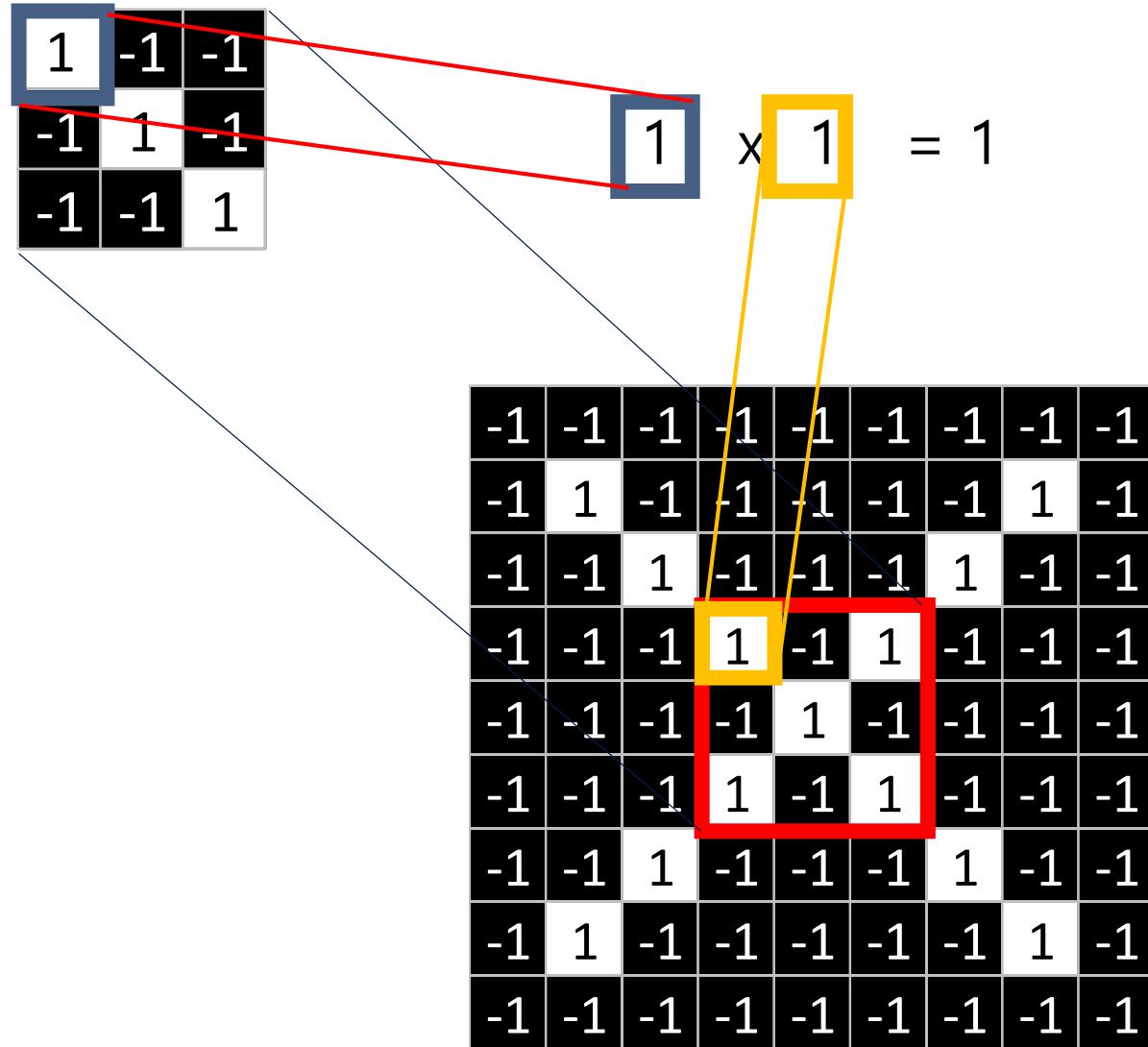
1	1	1
1	1	1
1	1	1

$$\frac{1 + 1 + 1 + 1 + 1 + 1 + 1 + 1}{9} = 1$$

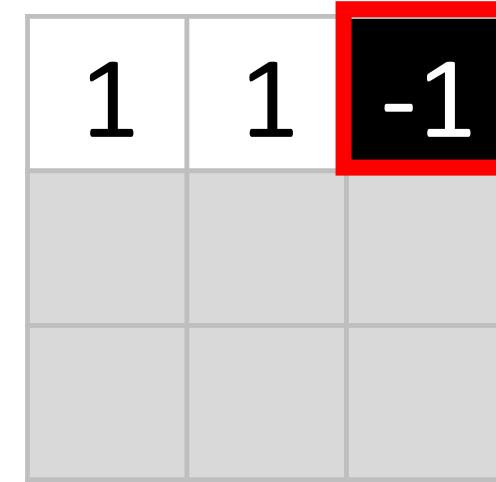
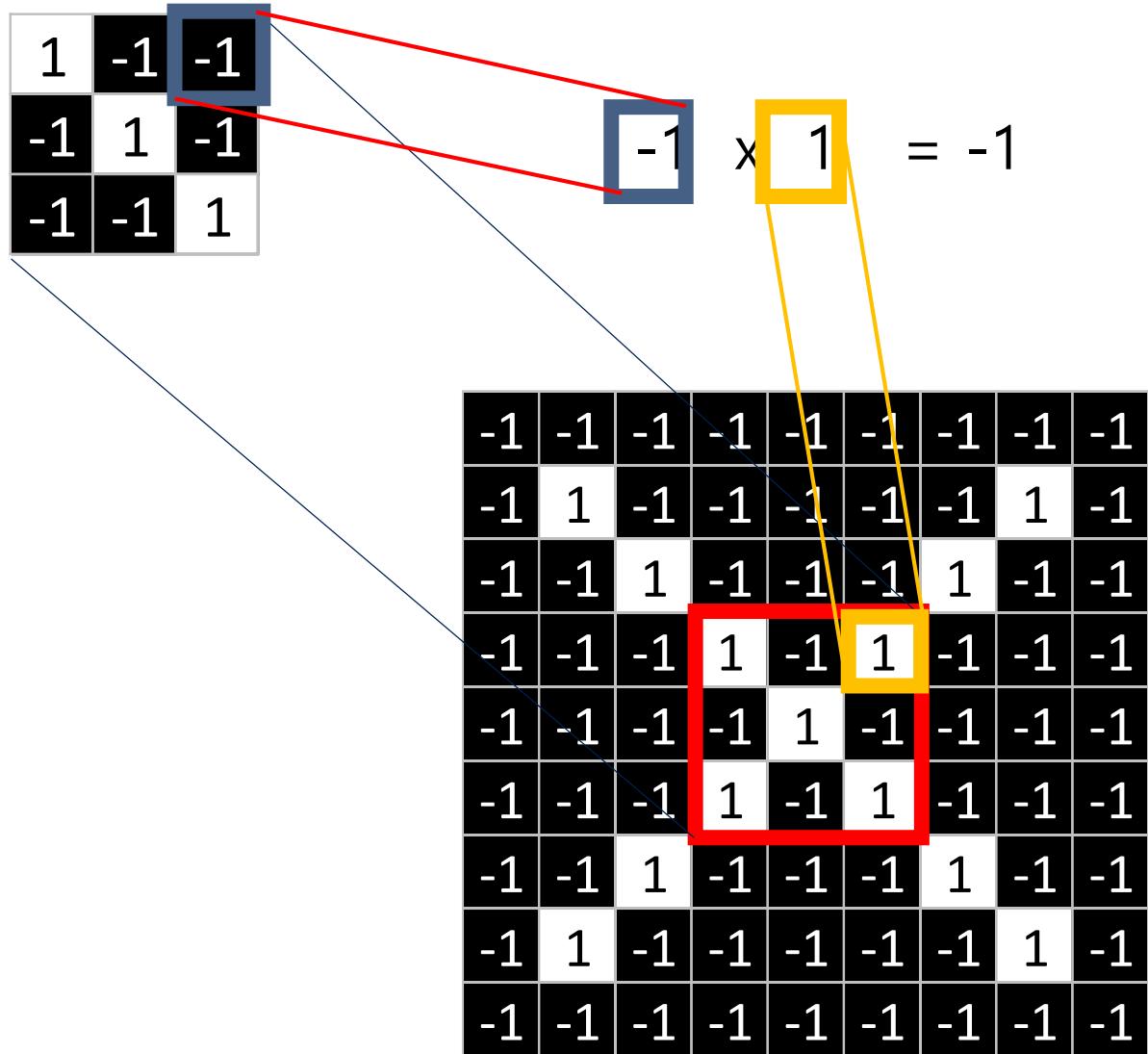
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



# Filtering: The math behind the match



# Filtering: The math behind the match



# Filtering: The math behind the match

1	-1	-1
-1	1	-1
-1	-1	1

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1	1	-1
1	1	1
-1	1	1

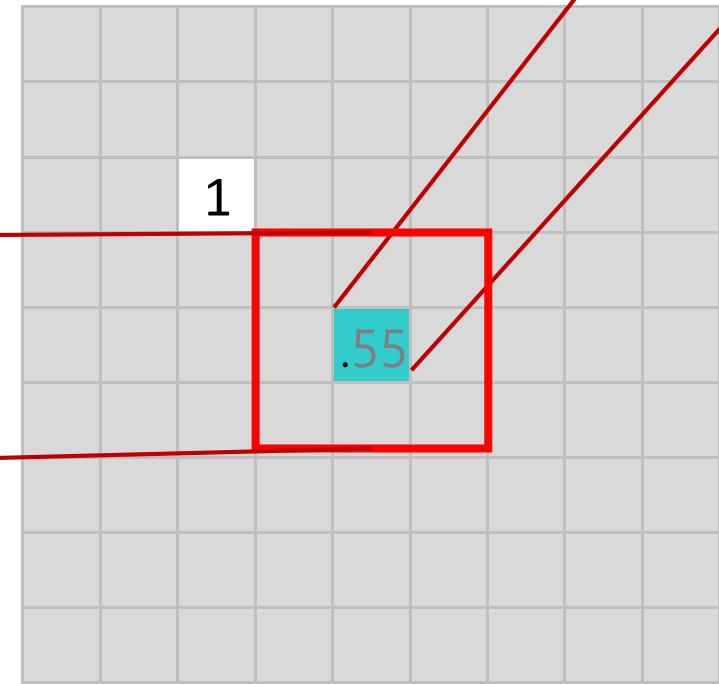
# Filtering: The math behind the match

1	-1	-1
-1	1	-1
-1	-1	1

1	1	-1
1	1	1
-1	1	1

$$\frac{1 + 1 - 1 + 1 + 1 + 1 - 1 + 1}{9} = .55$$

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



# Convolution: Trying every possible match

1	-1	-1
-1	1	-1
-1	-1	1

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

# Convolution: Trying every possible match

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	
-1	-1	1	-1	-1	-1	1	-1	-1	
-1	-1	-1	1	-1	1	-1	-1	-1	
-1	-1	-1	-1	1	-1	-1	-1	-1	
-1	-1	-1	-1	1	-1	-1	-1	-1	
-1	-1	-1	1	-1	1	-1	-1	-1	
-1	-1	1	-1	-1	-1	1	-1	-1	
-1	1	-1	-1	-1	-1	-1	1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	



1	-1	-1
-1	1	-1
-1	-1	1

=

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



1	-1	-1
-1	1	-1
-1	-1	1

=

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



1	-1	1
-1	1	-1
1	-1	1

=

0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.11	0.33	-0.77	1.00	-0.77	0.33	-0.11
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



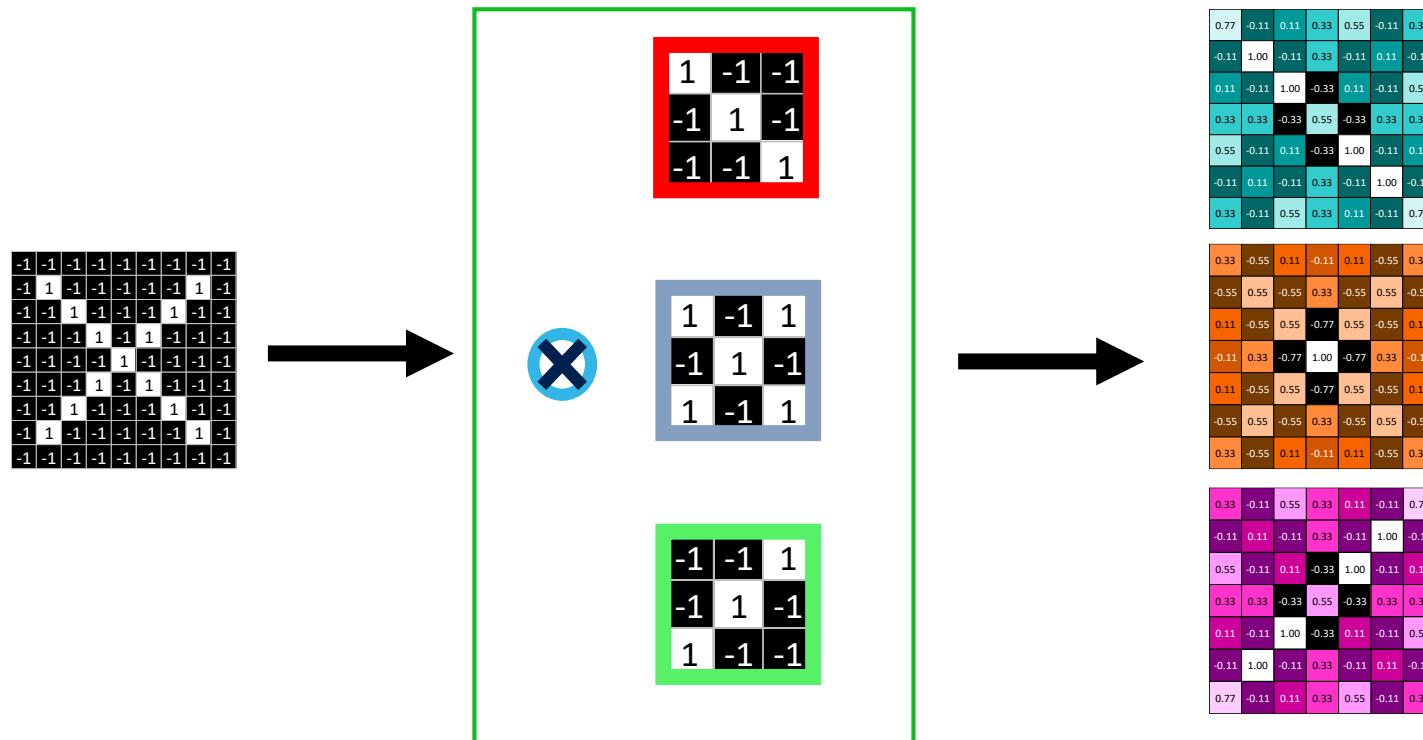
-1	-1	1
-1	1	-1
1	-1	-1

=

0.33	-0.11	0.55	0.33	0.11	-0.11	0.77
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33

# Convolution layer

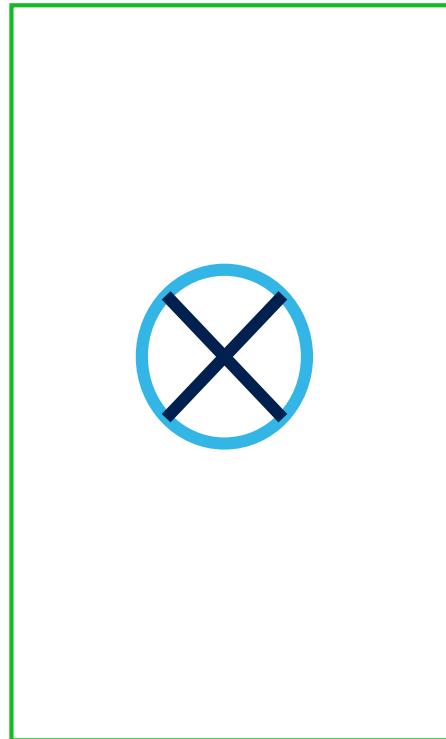
One image becomes a stack of filtered images



# Convolution layer

One image becomes a stack of filtered images

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	
-1	-1	1	-1	-1	-1	1	-1	-1	
-1	-1	-1	1	-1	1	-1	-1	-1	
-1	-1	-1	-1	1	-1	-1	-1	-1	
-1	-1	-1	-1	-1	1	-1	-1	-1	
-1	-1	-1	-1	-1	-1	1	-1	-1	
-1	1	-1	-1	-1	-1	-1	1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	1



0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

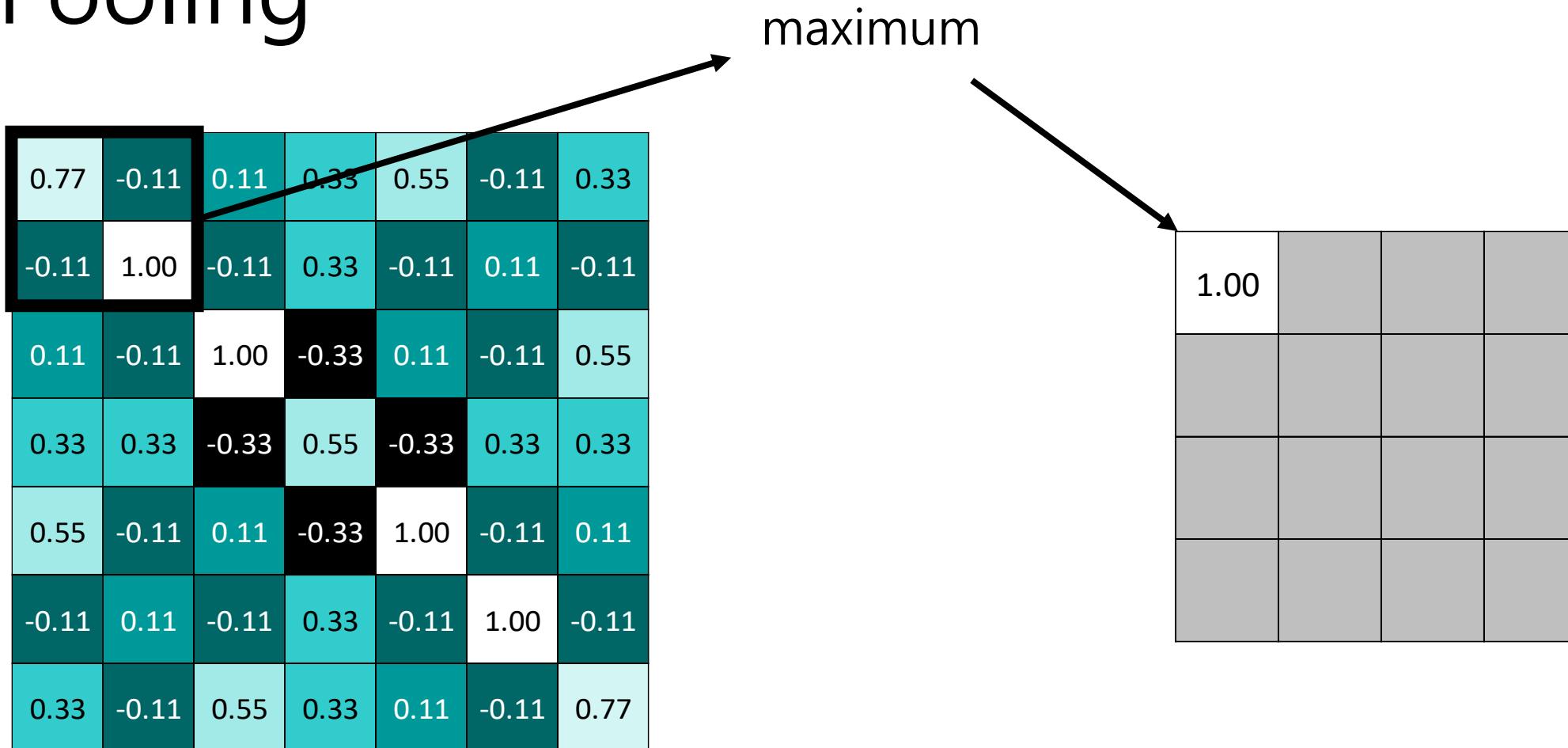
0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.11	0.33	-0.77	1.00	-0.77	0.33	-0.11
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33

0.33	-0.11	0.55	0.33	0.11	-0.11	0.77
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33

# Pooling: Shrinking the image stack

1. Pick a window size (usually 2 or 3).
2. Pick a stride (usually 2).
3. Walk your window across your filtered images.
4. From each window, take the maximum value.

# Pooling



# Pooling

maximum

0.77	-0.11	0.11	0.33	0.55	0.11	0.33	
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11	
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55	
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33	
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11	
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11	
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77	

1.00	0.33		

# Pooling

maximum

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

1.00	0.33	0.55	

# Pooling

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33	
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11	
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55	
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33	
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11	
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11	
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77	

maximum

1.00	0.33	0.55	0.33

# Pooling

maximum

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

1.00	0.33	0.55	0.33
0.33			

# Pooling

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

max pooling



1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77



1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.11	0.33	-0.77	1.00	-0.77	0.33	-0.11
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33



0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

0.33	-0.11	0.55	0.33	0.11	-0.11	0.77
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33



0.33	0.55	1.00	0.77
0.55	0.55	1.00	0.33
1.00	1.00	0.11	0.55
0.77	0.33	0.55	0.33

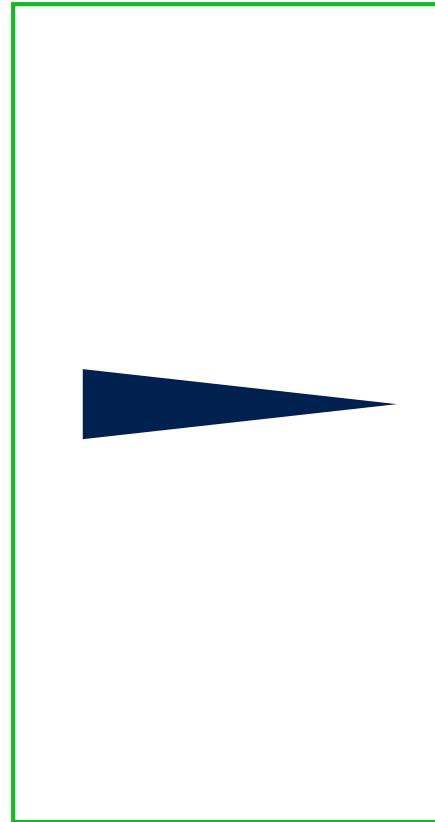
# Pooling layer

A stack of images becomes a stack of smaller images.

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.11	0.33	-0.77	1.00	-0.77	0.33	-0.11
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33

0.33	-0.11	0.55	0.33	0.11	-0.11	0.77
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33



1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

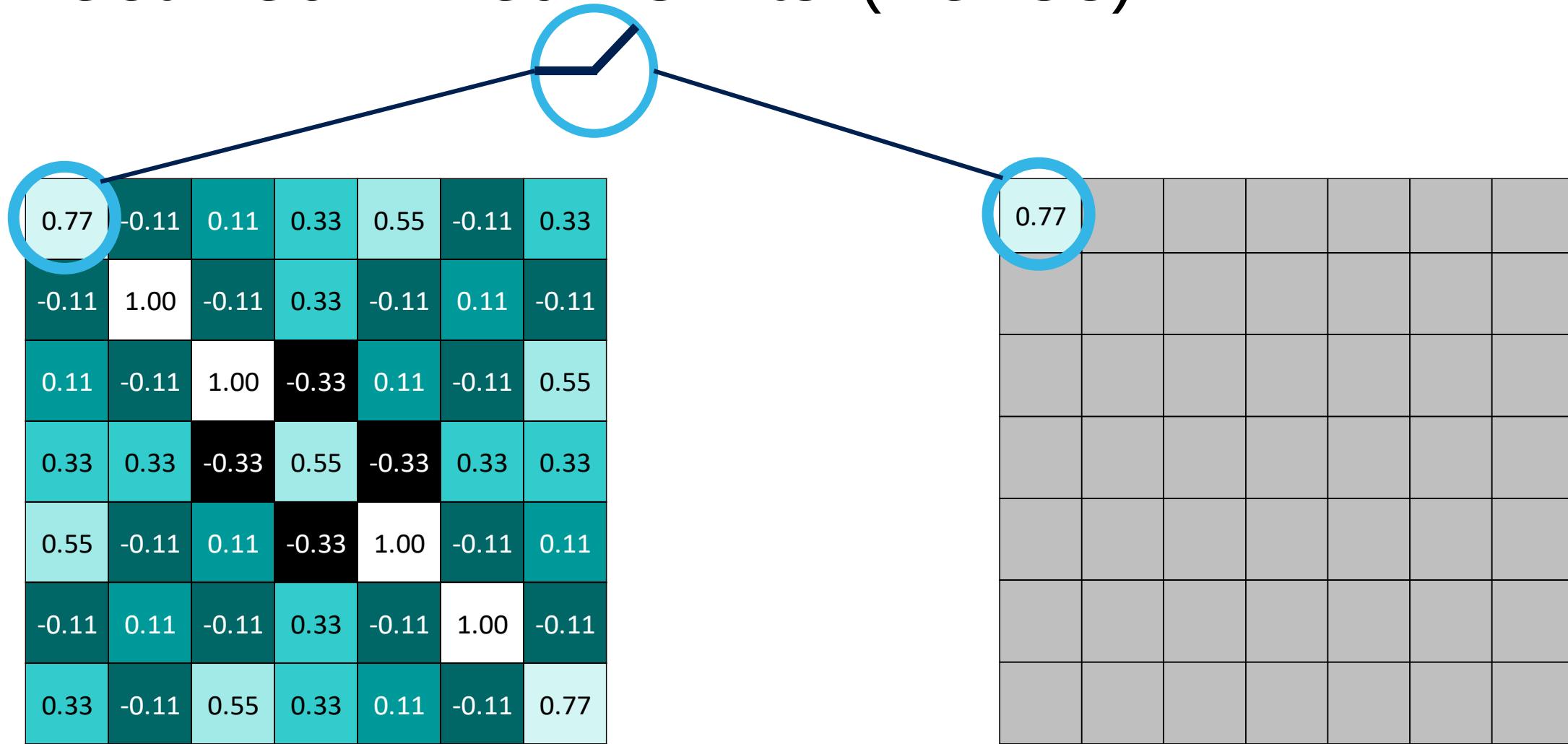
0.33	0.55	1.00	0.77
0.55	0.55	1.00	0.33
1.00	1.00	0.11	0.55
0.77	0.33	0.55	0.33

# Nonlinearity

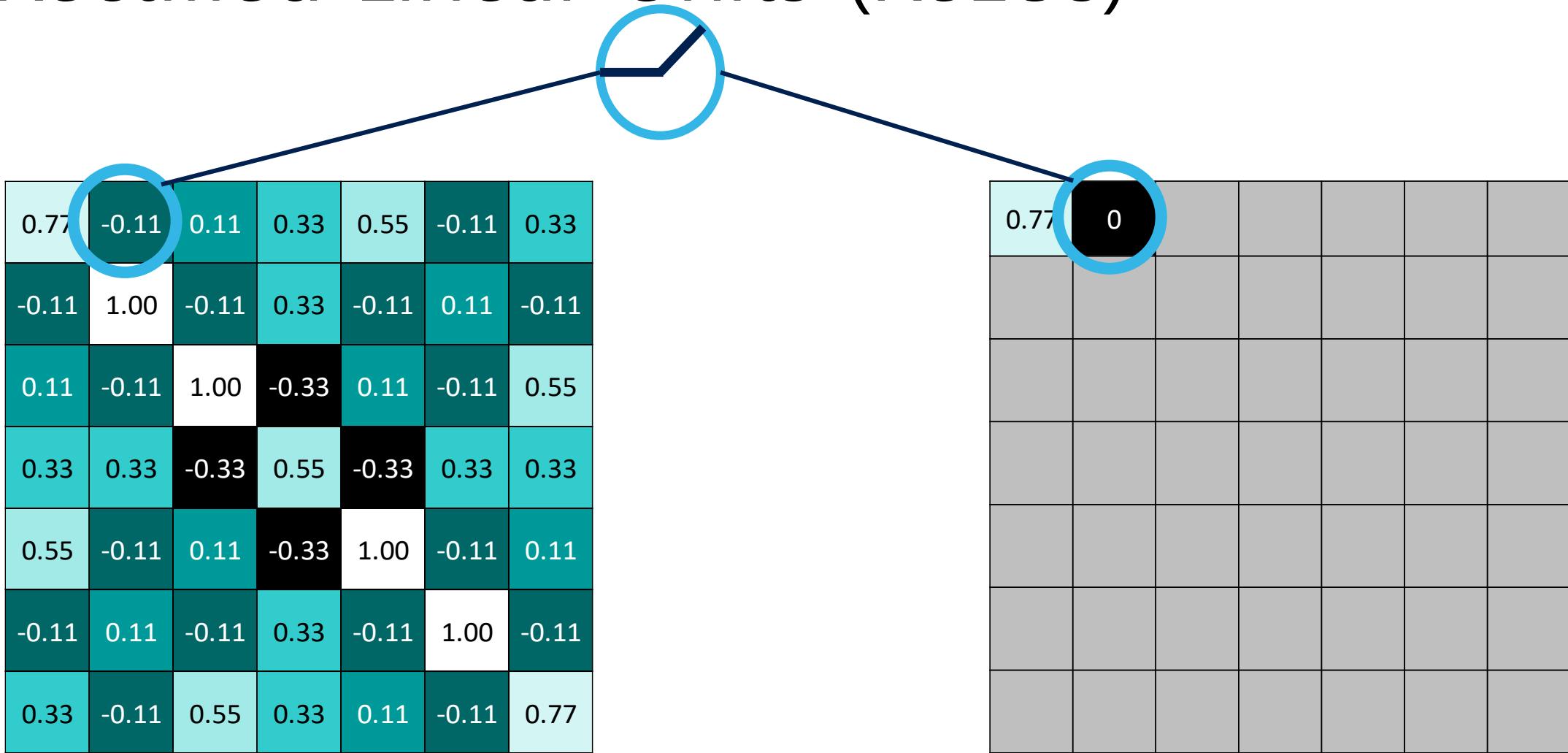
Keep the math from breaking by tweaking each of the values just a bit.

Change everything negative to zero.

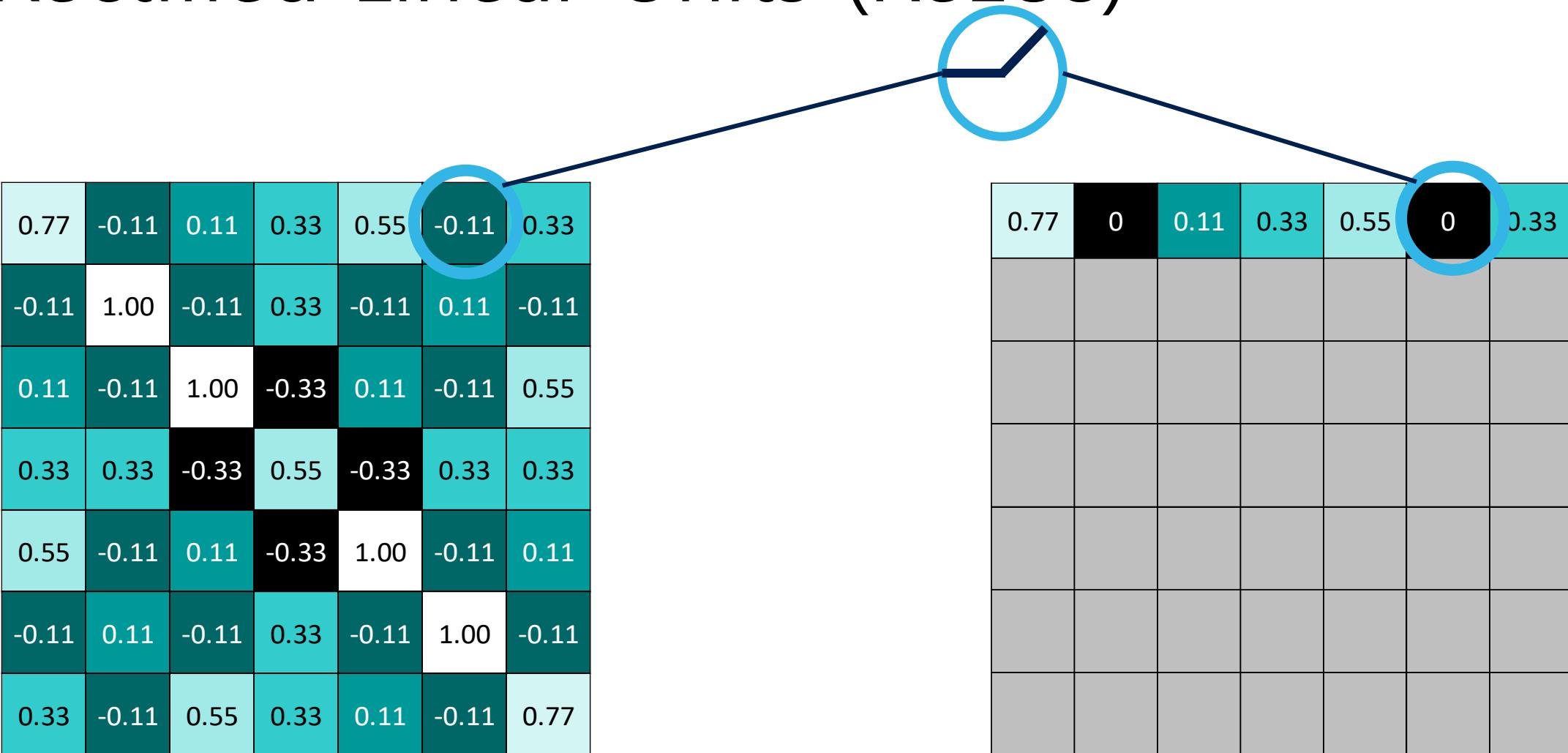
# Rectified Linear Units (ReLUs)



# Rectified Linear Units (ReLUs)



# Rectified Linear Units (ReLUs)



# Rectified Linear Units (ReLUs)

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77



0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

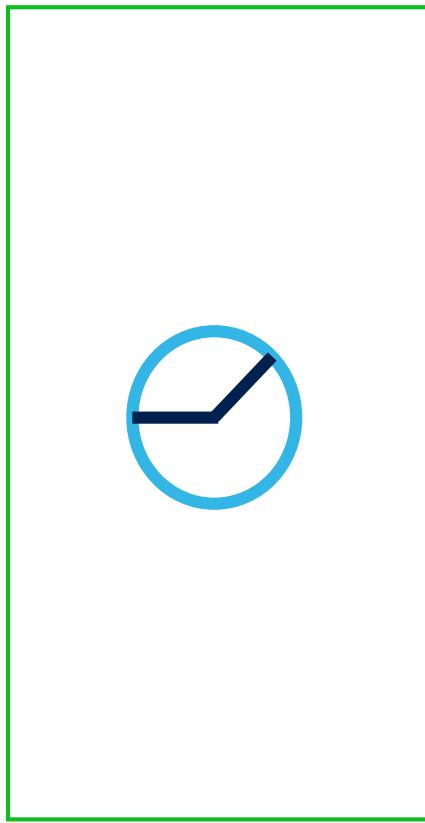
# ReLU layer

A stack of images becomes a stack of images with no negative values.

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.11	0.33	-0.77	1.00	-0.77	0.33	-0.11
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.55	0.55	-0.55	0.33	-0.55	0.55	0.55
0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33

0.33	-0.11	0.55	0.33	0.11	-0.11	0.77
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33



0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

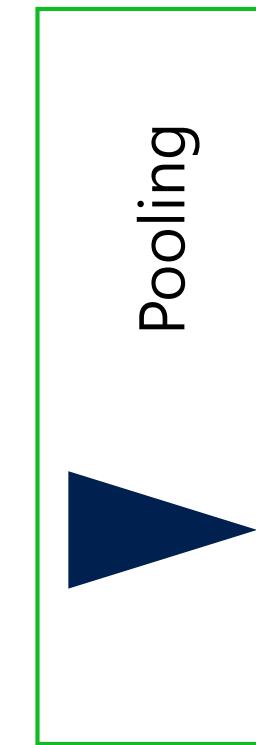
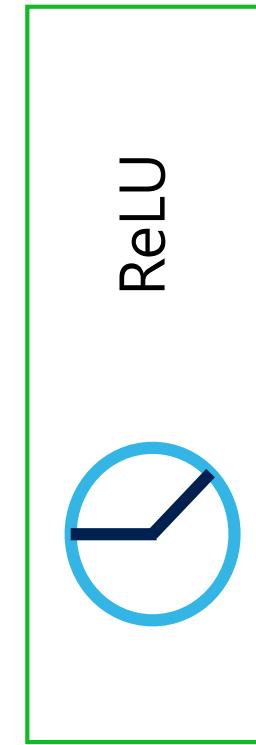
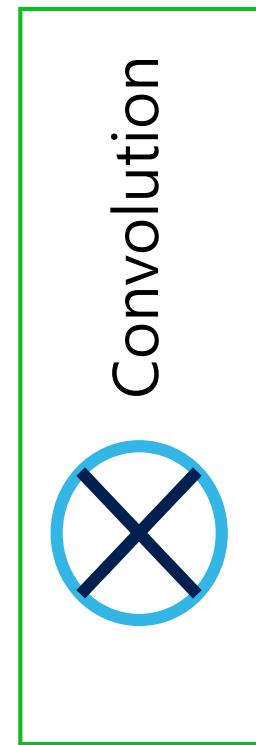
0.33	0	0.11	0	0.11	0	0.33
0	0.55	0	0.33	0	0.55	0
0.11	0	0.55	0	0.55	0	0.11
0	0.33	0	1.00	0	0.33	0
0.11	0	0.55	0	0.55	0	0.11
0	0.55	0	0.33	0	0.55	0
0.33	0	0.11	0	0.11	0	0.33

0.33	0	0.55	0.33	0.11	0	0.77
0	0.11	0	0.33	0	1.00	0
0.55	0	0.11	0	1.00	0	0.11
0.33	0.33	0	0.55	0	0.33	0.33
0.11	0	1.00	0	0.11	0	0.55
0	1.00	0	0.33	0	0.11	0
0.77	0	0.11	0.33	0.55	0	0.33

# Layers get stacked

The output of one becomes the input of the next.

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1



1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

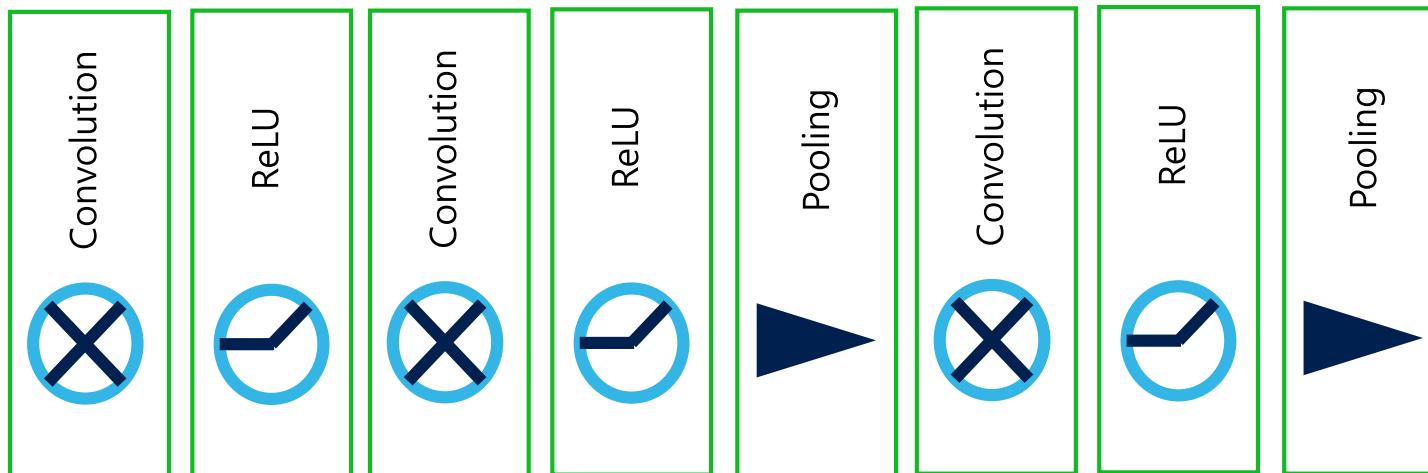
0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

0.33	0.55	1.00	0.77
0.55	0.55	1.00	0.33
1.00	1.00	0.11	0.55
0.77	0.33	0.55	0.33

# Deep stacking

Layers can be repeated several (or many) times.

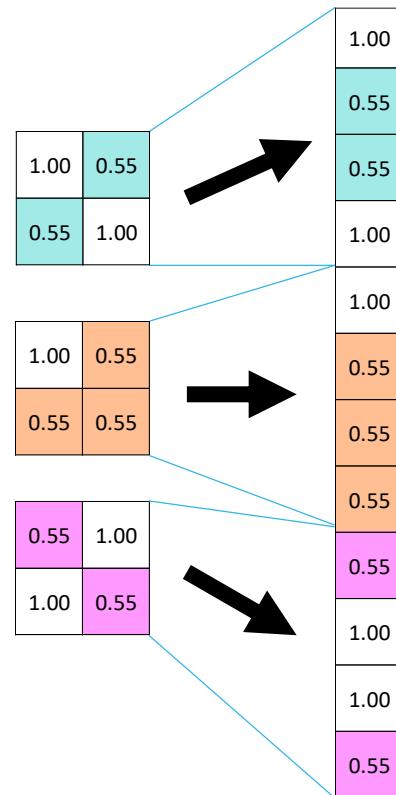
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	
-1	-1	1	-1	-1	-1	1	-1	-1	
-1	-1	-1	1	-1	1	-1	-1	-1	
-1	-1	-1	-1	1	-1	-1	-1	-1	
-1	-1	-1	-1	1	-1	-1	-1	-1	
-1	-1	-1	1	-1	1	-1	-1	-1	
-1	-1	1	-1	-1	-1	1	-1	-1	
-1	1	-1	-1	-1	-1	-1	1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	



1.00	0.55
0.55	1.00
1.00	0.55
0.55	0.55
0.55	1.00
1.00	0.55

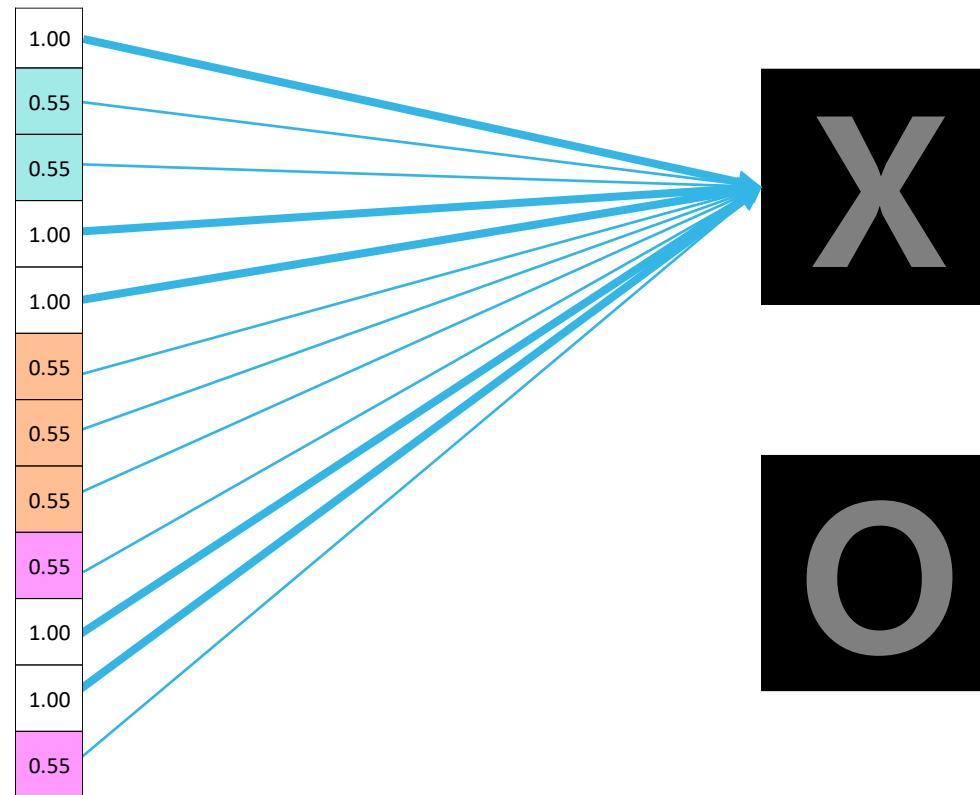
# Fully connected layer

Every value gets a vote



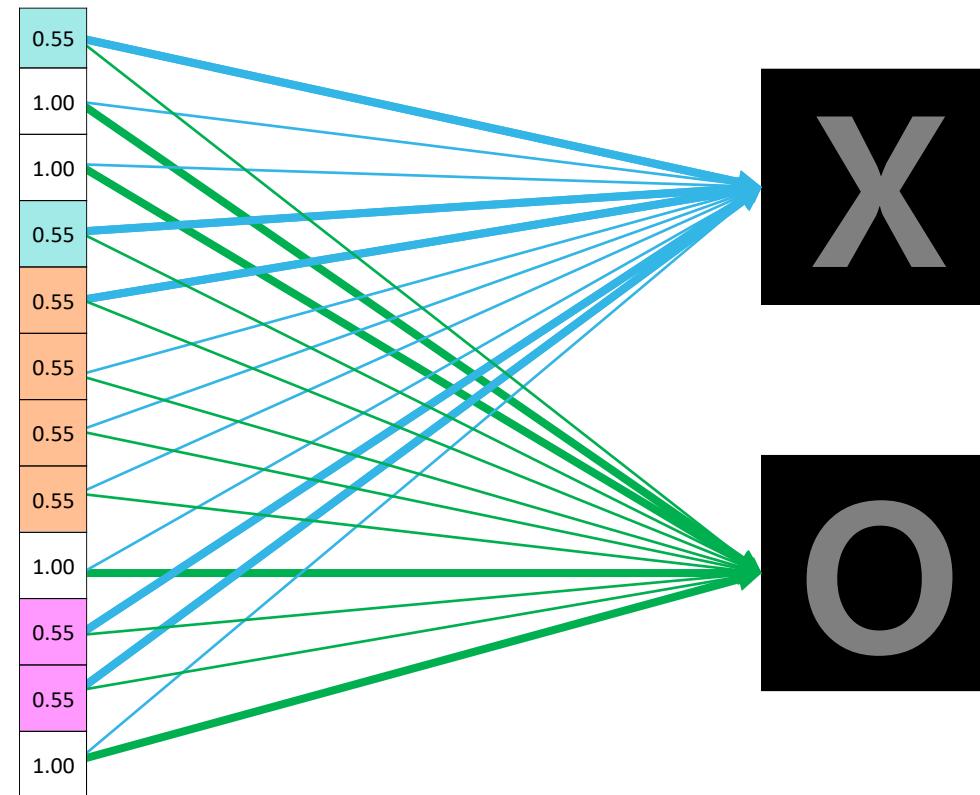
# Fully connected layer

Vote depends on how strongly a value predicts X or O



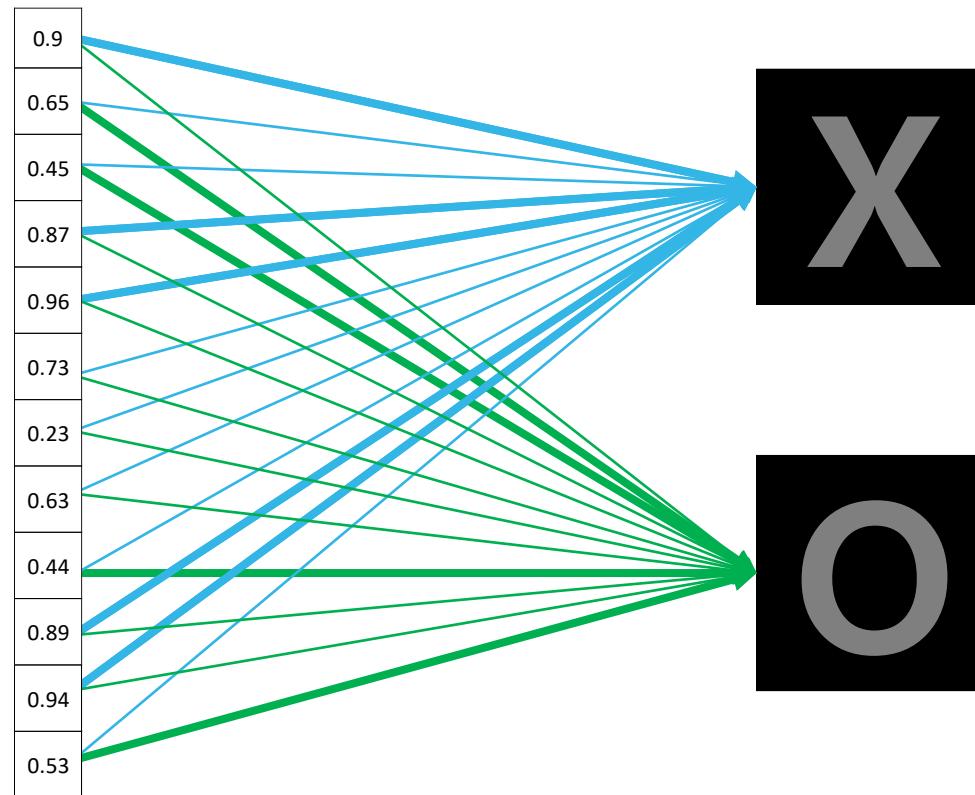
# Fully connected layer

Vote depends on how strongly a value predicts X or O



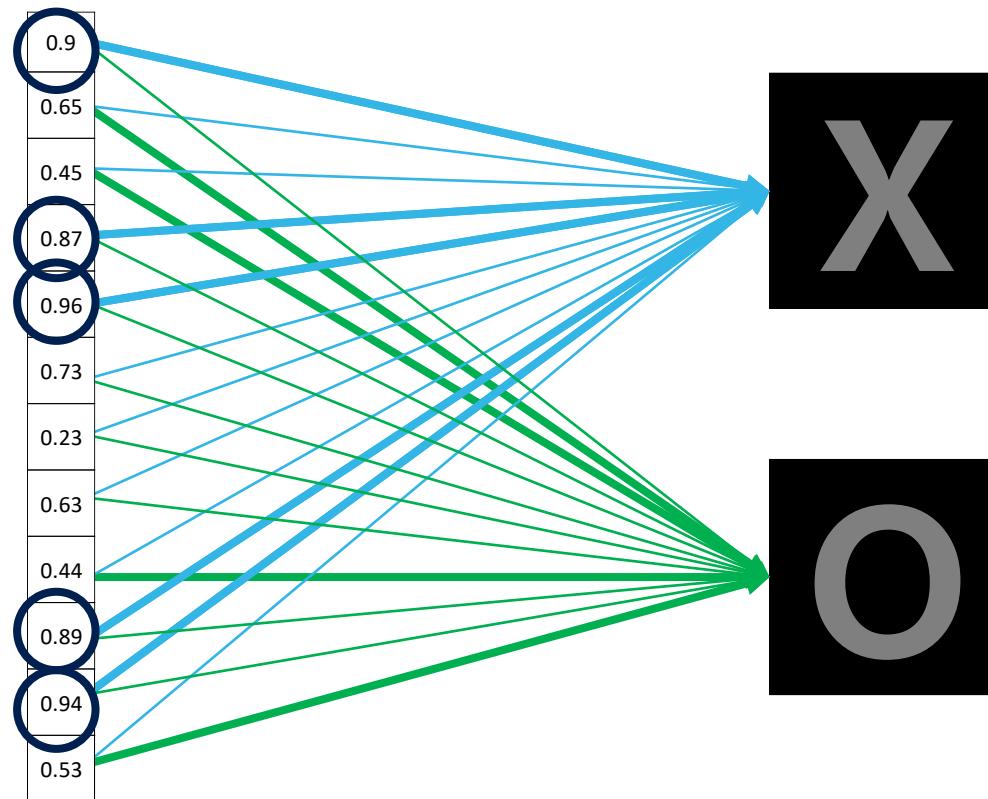
# Fully connected layer

Future values vote on X or O



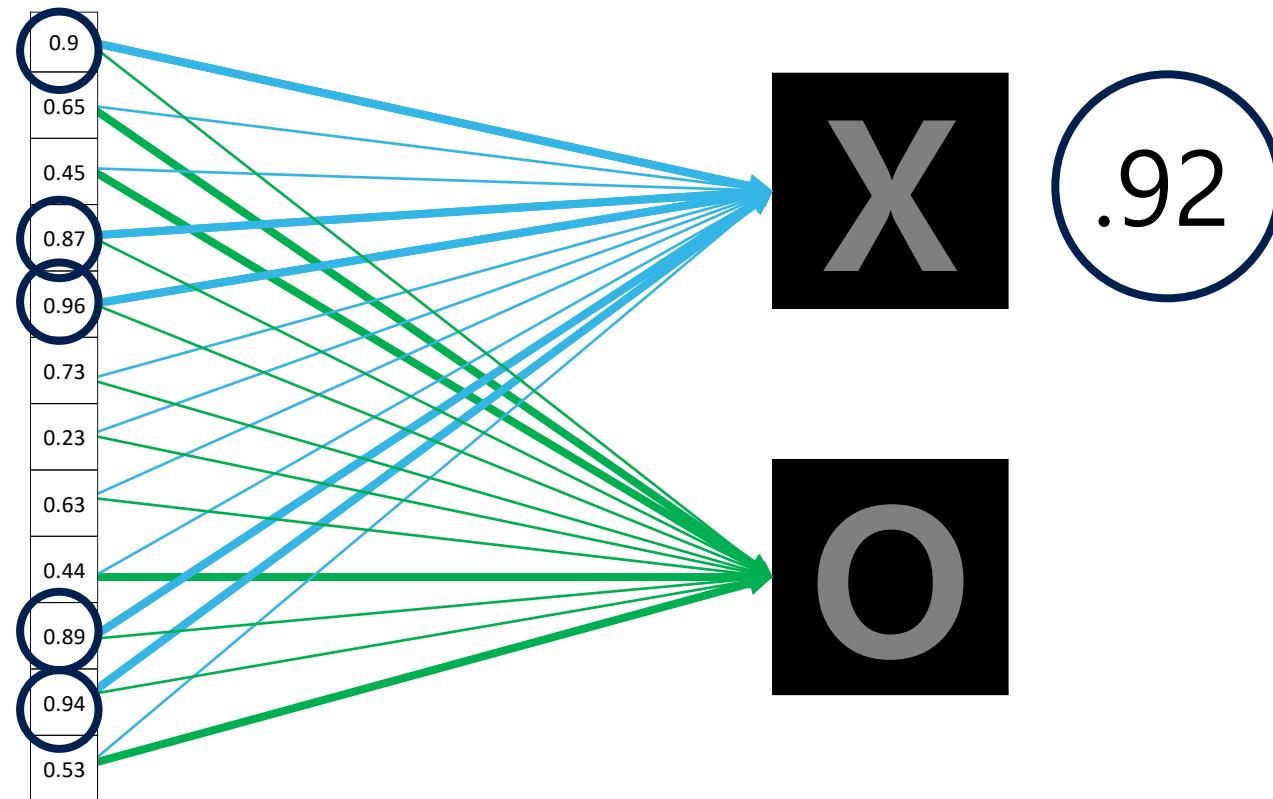
# Fully connected layer

Future values vote on X or O



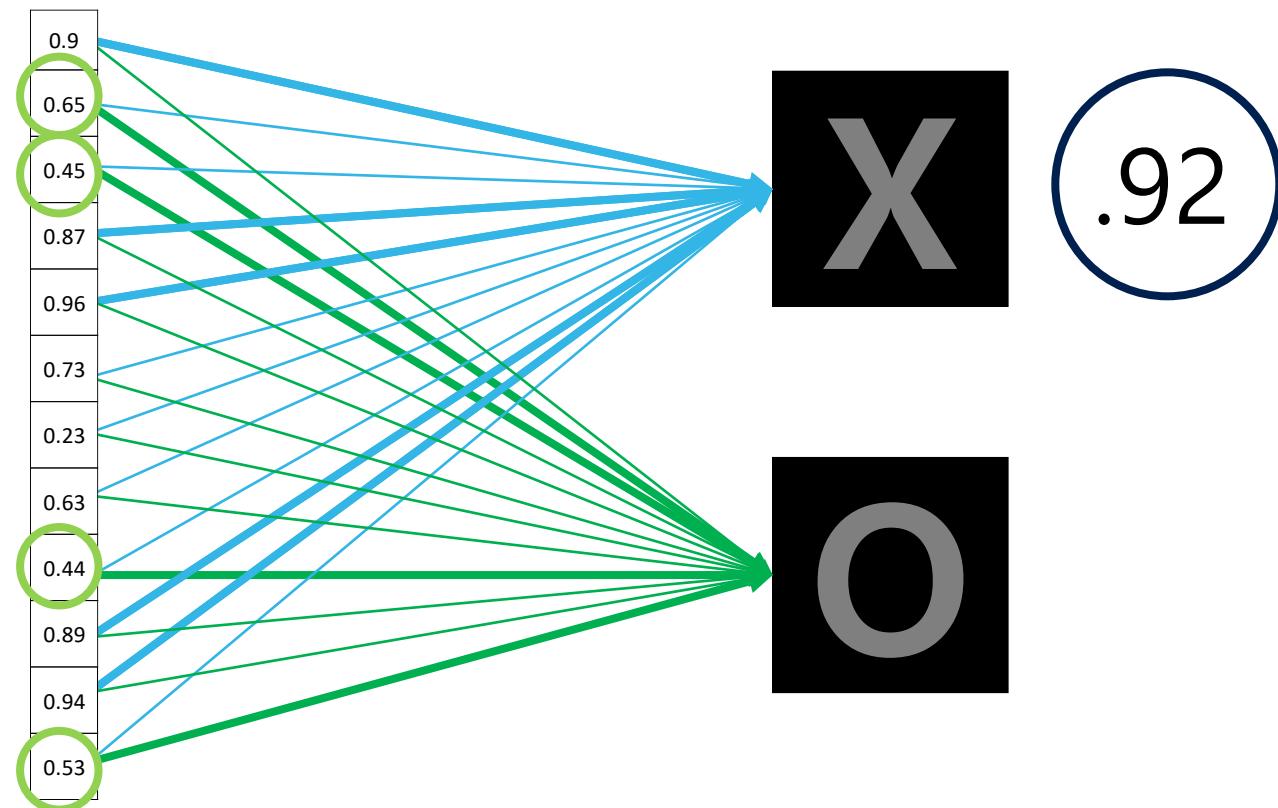
# Fully connected layer

Future values vote on X or O



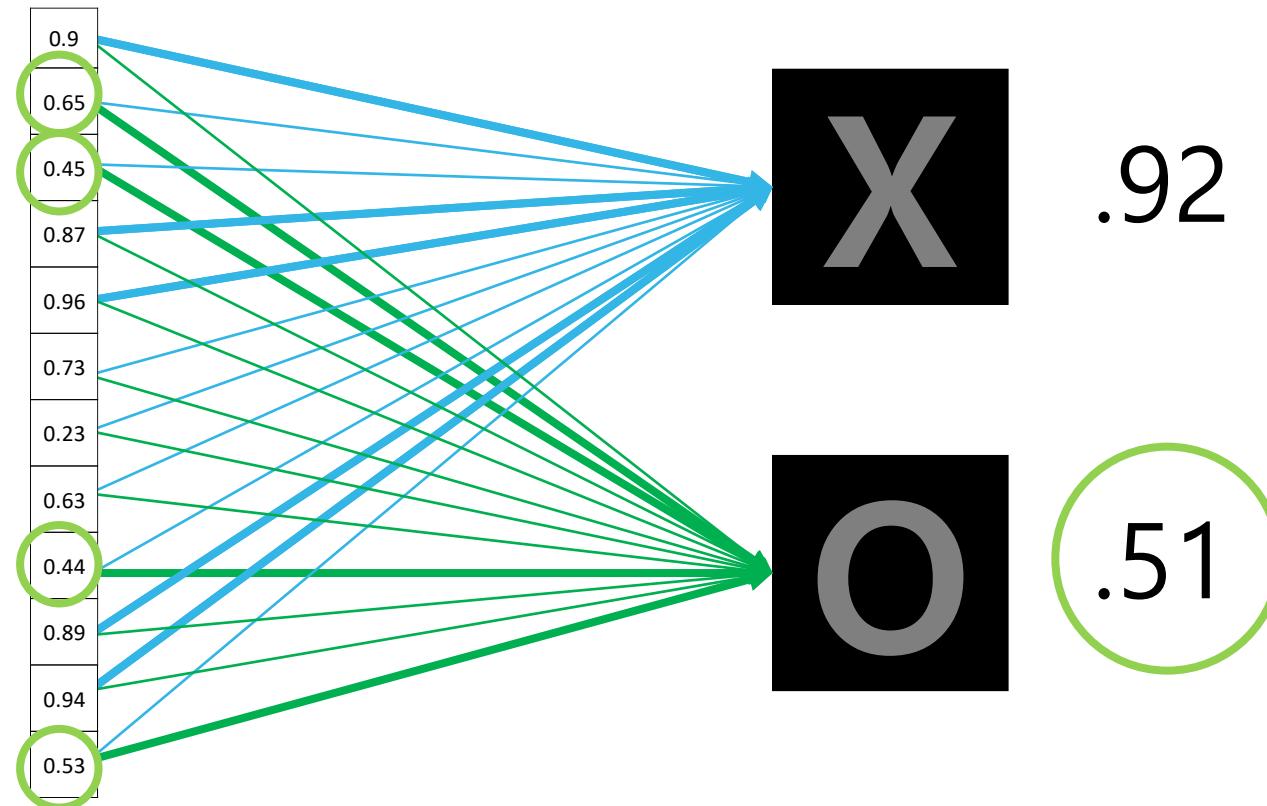
# Fully connected layer

Future values vote on X or O



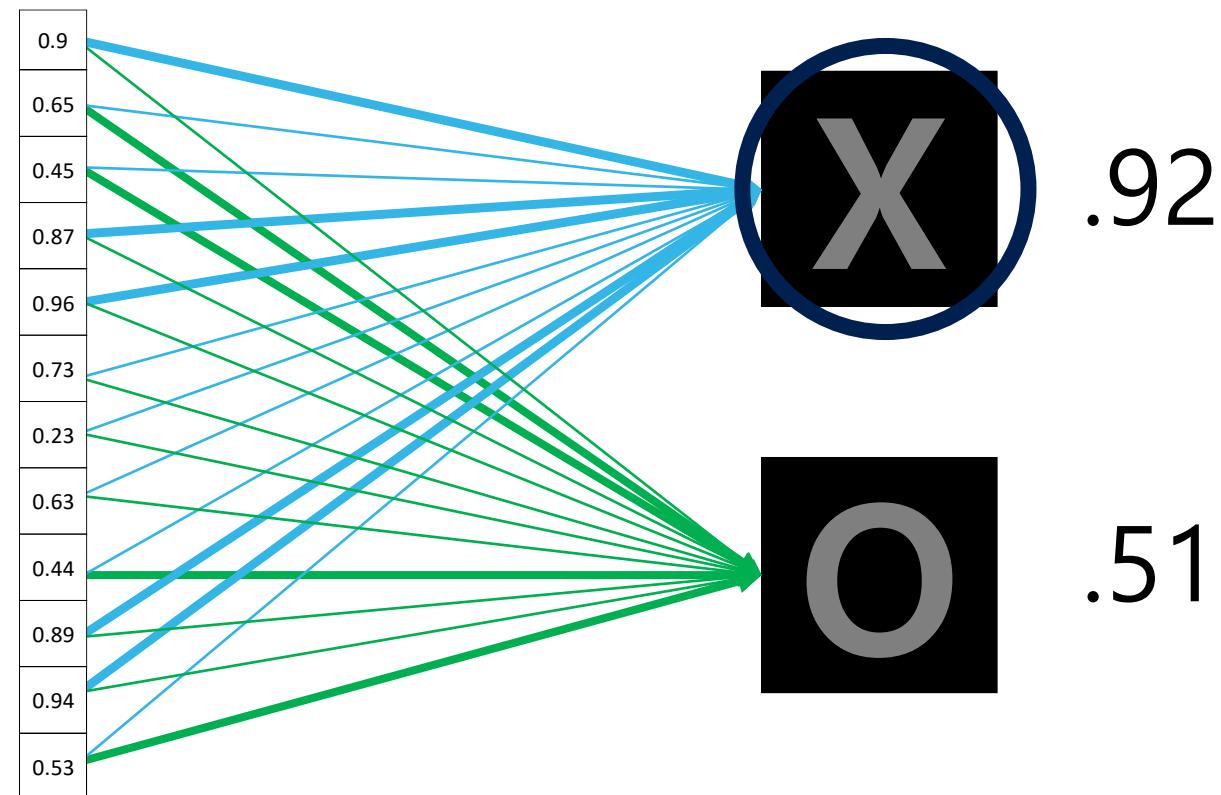
# Fully connected layer

Future values vote on X or O



# Fully connected layer

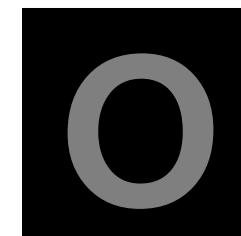
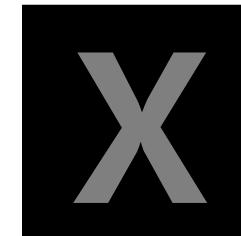
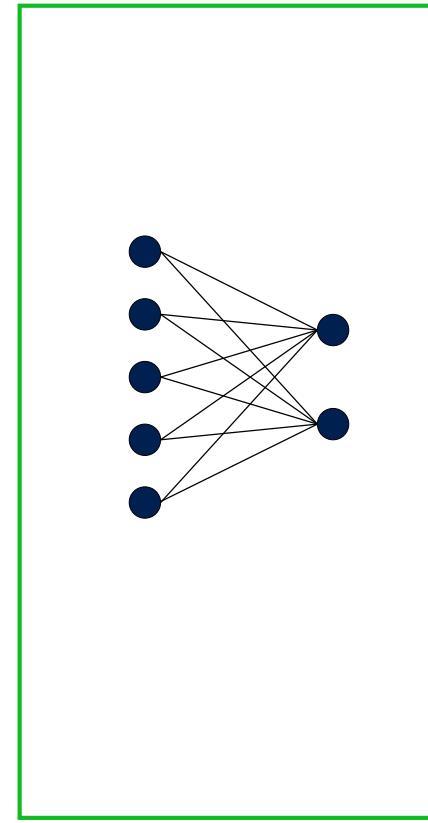
Future values vote on X or O



# Fully connected layer

A list of feature values becomes a list of votes.

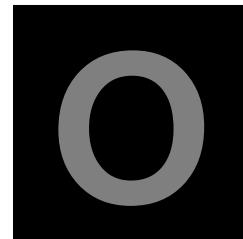
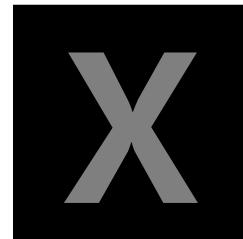
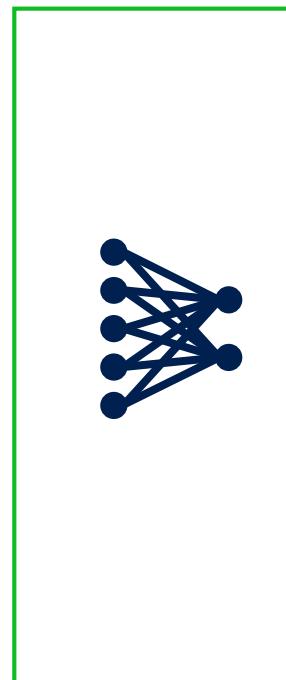
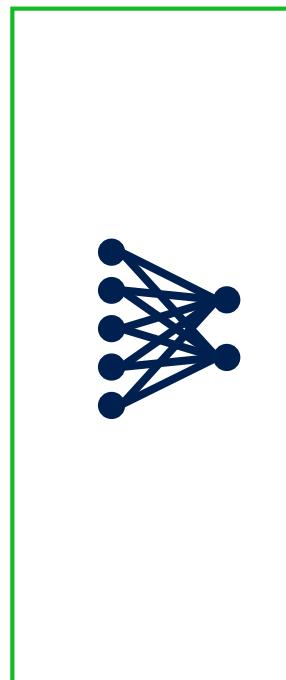
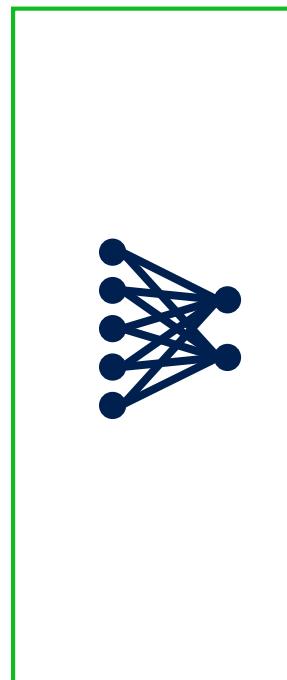
0.9
0.65
0.45
0.87
0.96
0.73
0.23
0.63
0.44
0.89
0.94
0.53



# Fully connected layer

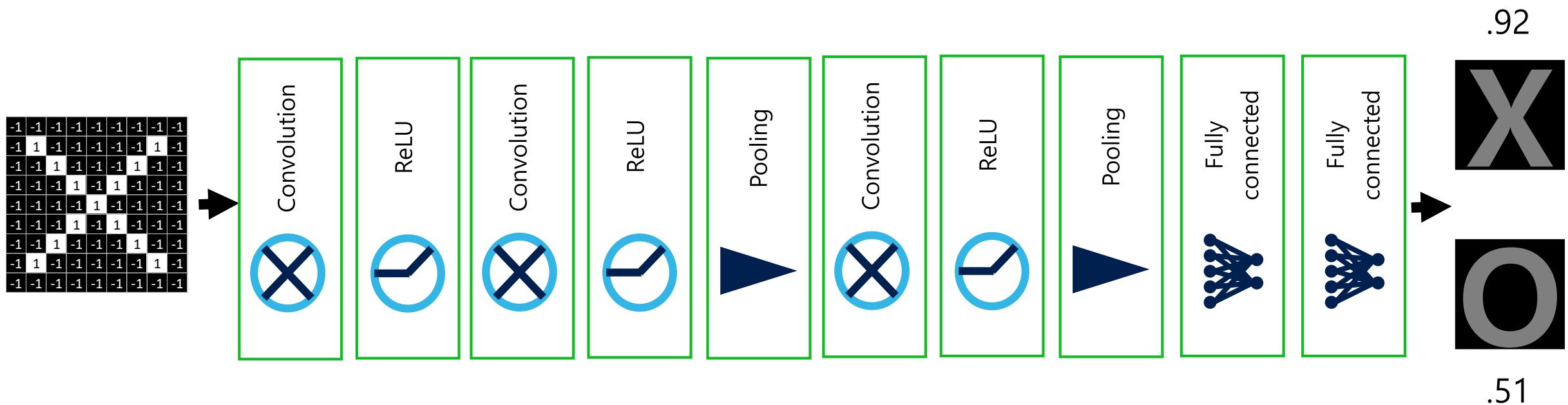
These can also be stacked.

0.9
0.65
0.45
0.87
0.96
0.73
0.23
0.63
0.44
0.89
0.94
0.53



# Putting it all together

A set of pixels becomes a set of votes.



# Learning

Q: Where do all the magic numbers come from?

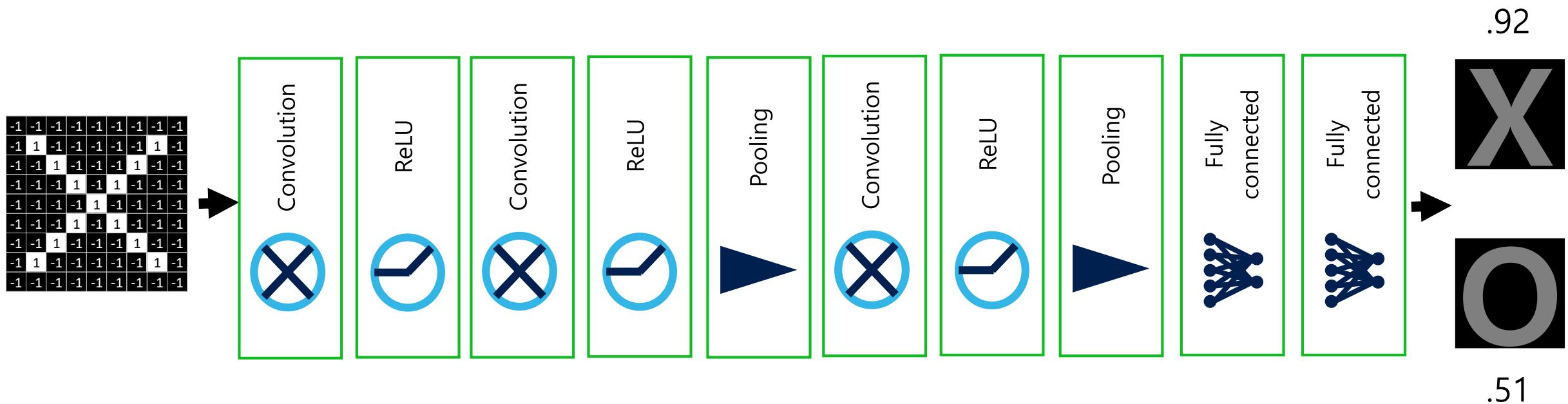
Features in convolutional layers

Voting weights in fully connected layers

A: Backpropagation

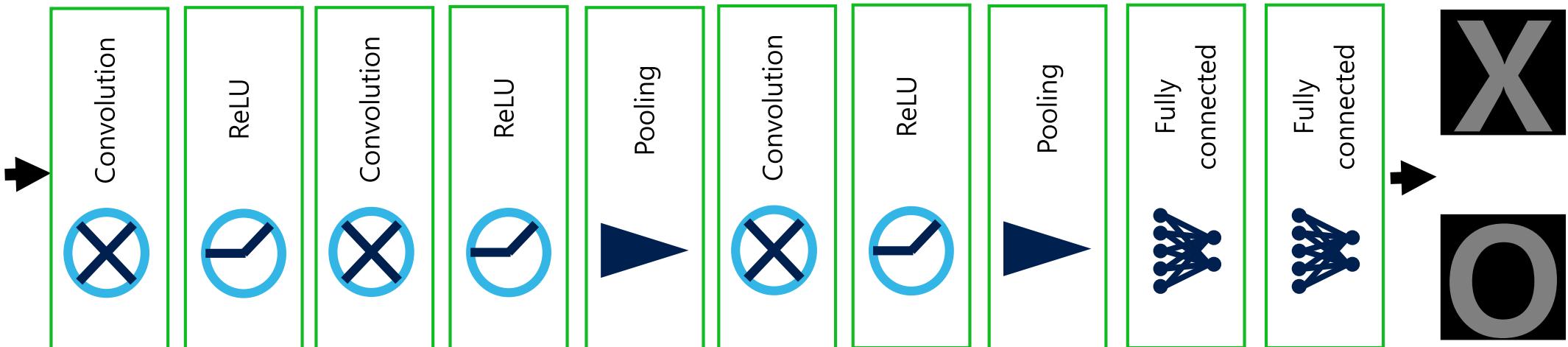
# Backprop

Error = right answer – actual answer



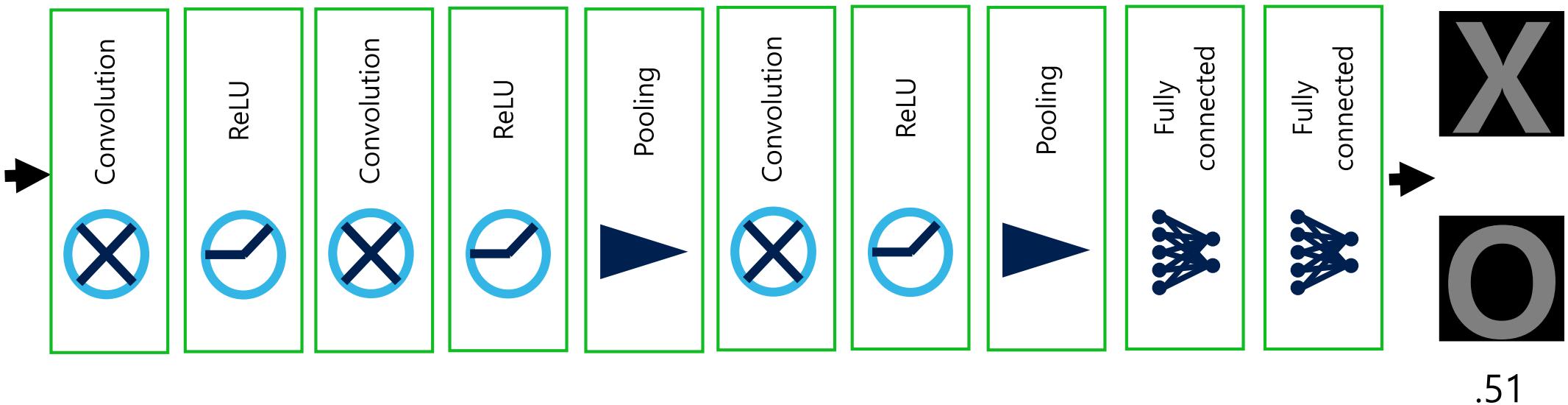
# Backprop

$\begin{matrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & \textcolor{white}{1} & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & 1 & -1 & -1 & 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{matrix}$



# Backprop

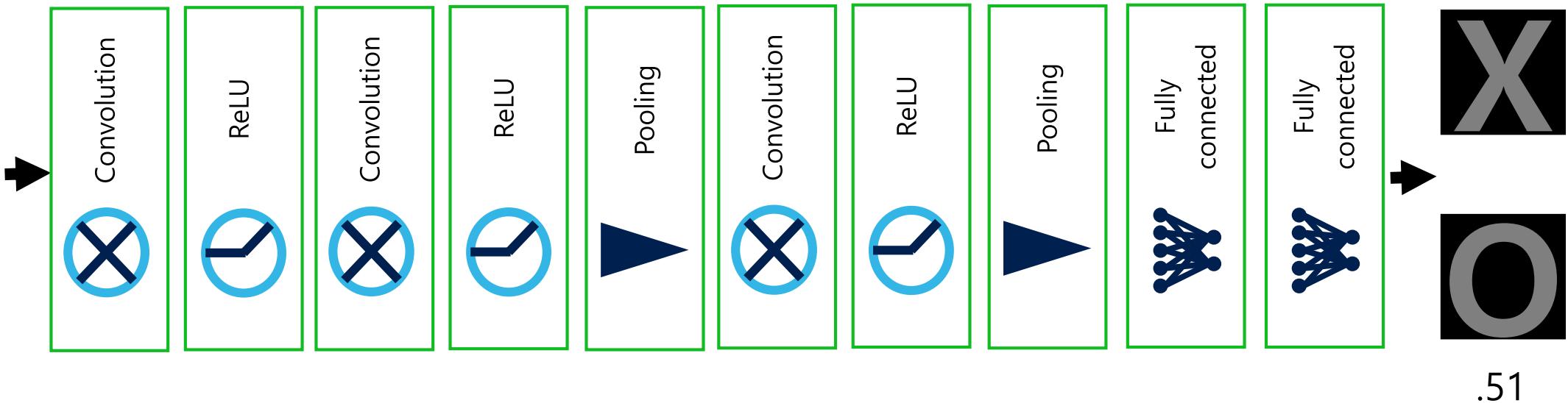
$\begin{matrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & -1 & -1 & -1 & -1 \\ -1 & -1 & 1 & -1 & -1 & 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{matrix}$



	Right answer	Actual answer	Error
X	1	0.92	
O			

# Backprop

$\begin{matrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 & -1 & 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{matrix}$

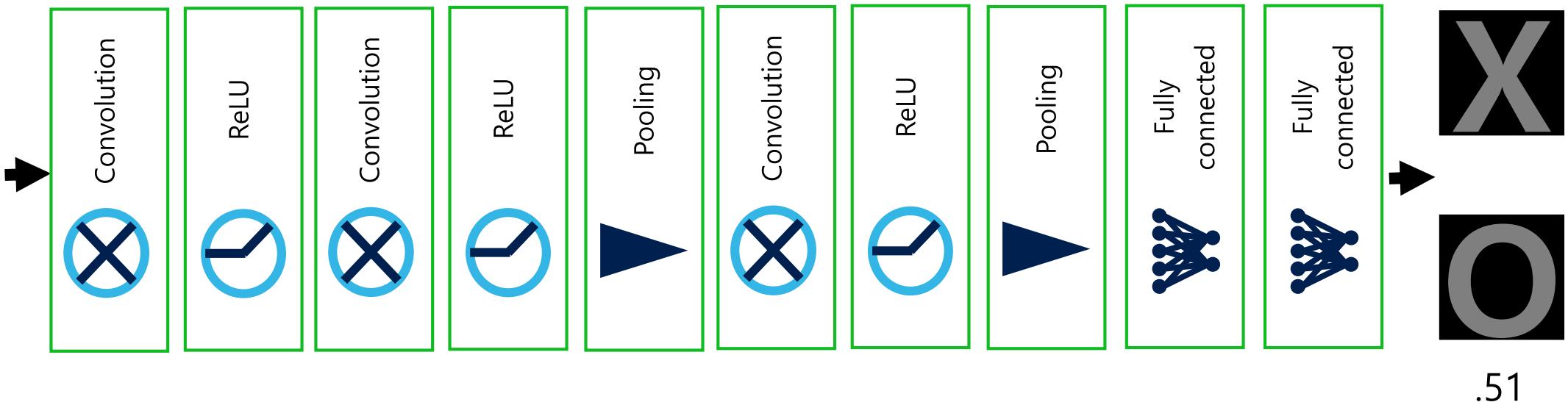


.92

.51

# Backprop

$\begin{matrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & \textcolor{white}{1} & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & 1 & -1 & -1 & 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{matrix}$



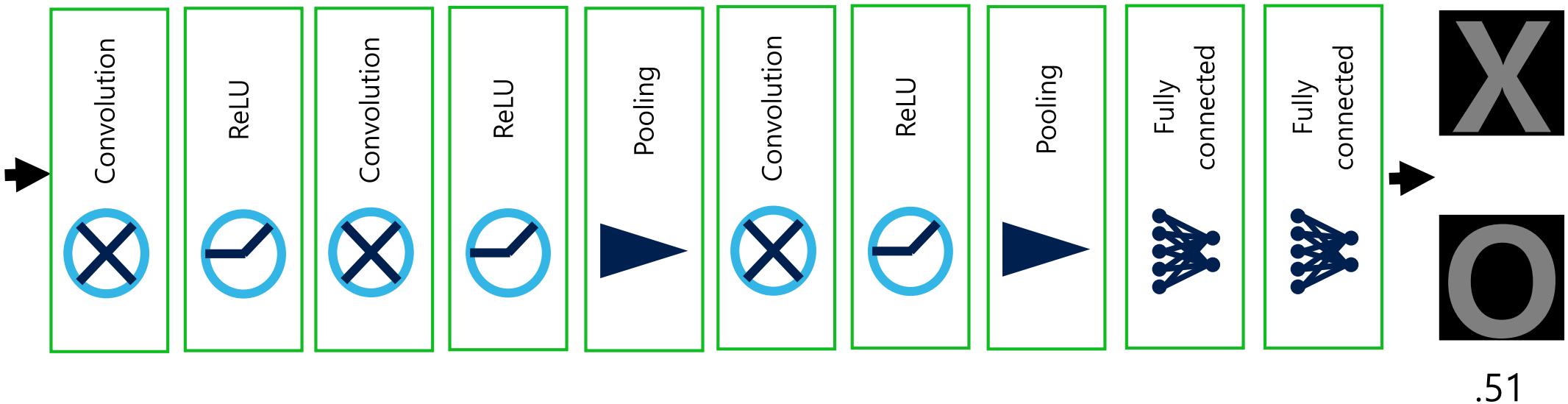
	Right answer	Actual answer	Error
X	1	0.92	0.08
O	0	0.51	0.49

.92

.51

# Backprop

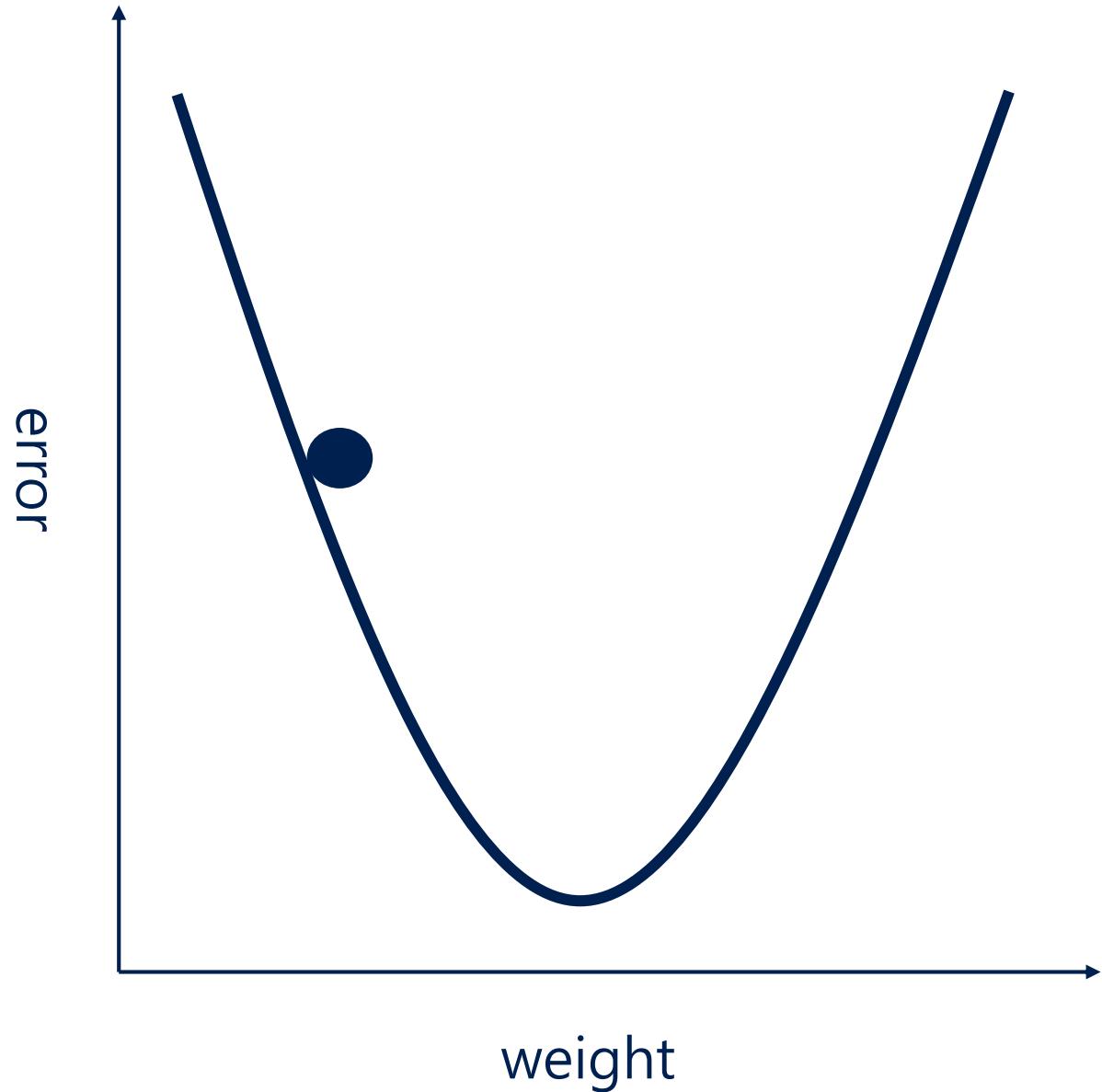
$\begin{matrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & \textcolor{red}{1} & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & 1 & -1 & -1 & 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{matrix}$



	Right answer	Actual answer	Error
X	1	0.92	0.08
O	0	0.51	0.49
Total		0.57	

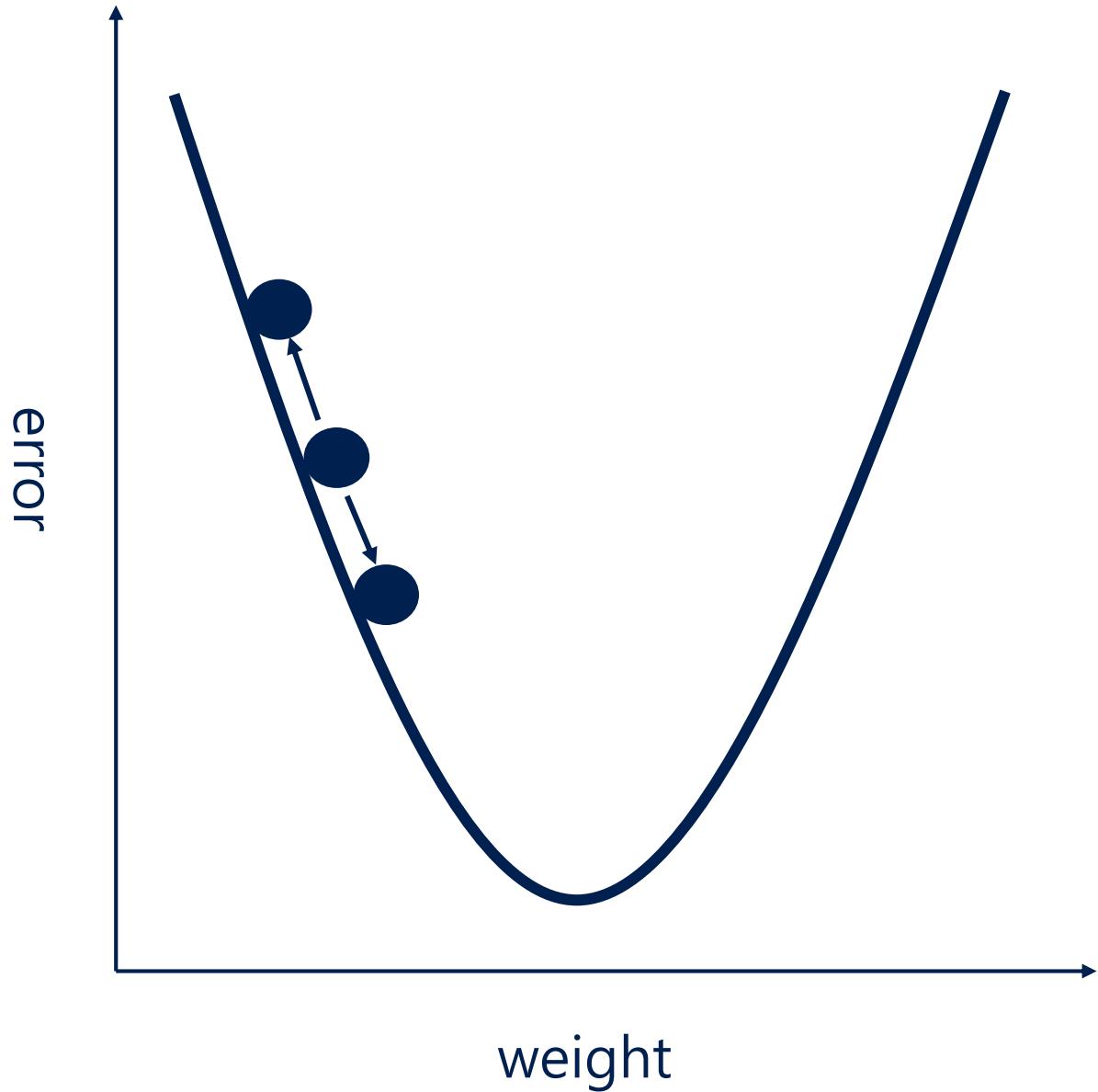
# Gradient descent

For each feature pixel and voting weight, adjust it up and down a bit and see how the error changes.



# Gradient descent

For each feature pixel and voting weight, adjust it up and down a bit and see how the error changes.



# Hyperparameters

## Convolution

- Number of filters

- Size of filters

## Pooling

- Window size

- Window stride

## Fully Connected

- Number of layers

- Number of neurons

# Architecture

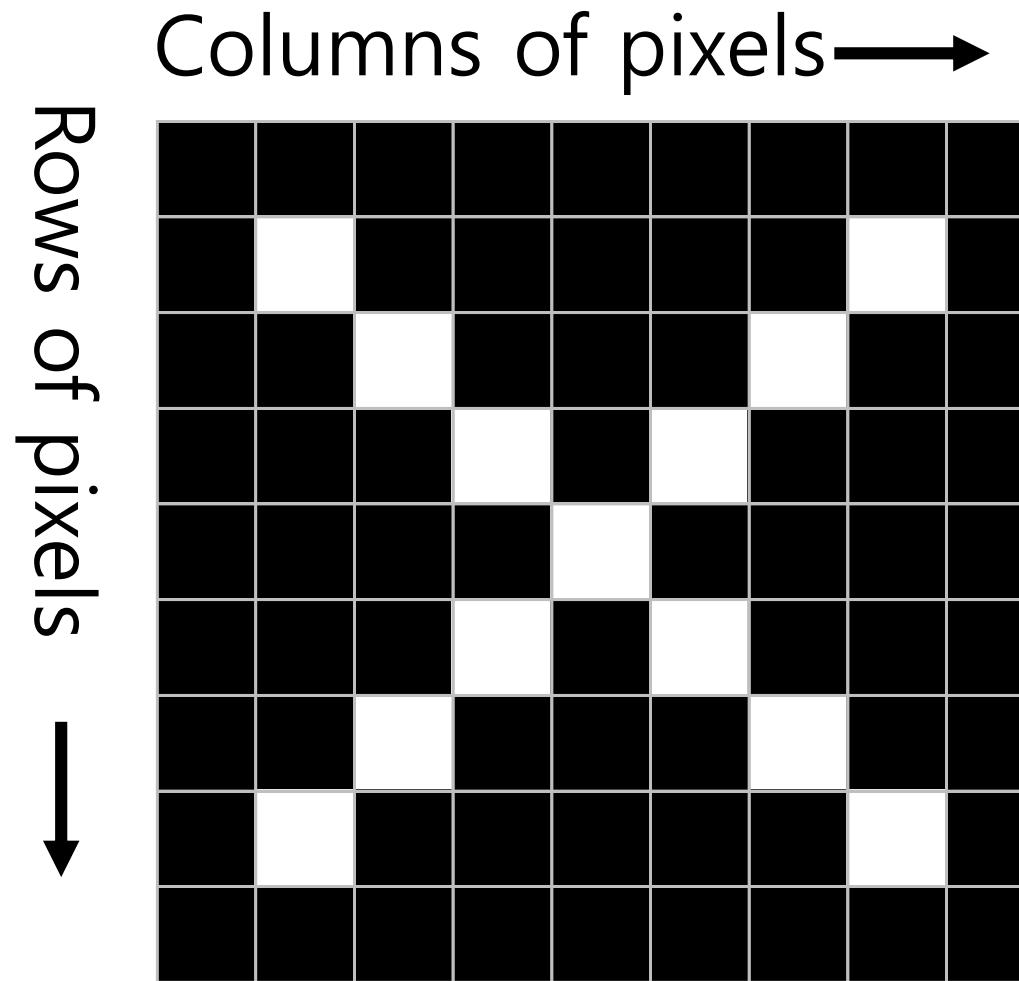
How many of each type of layer?  
In what order?

# Not just images

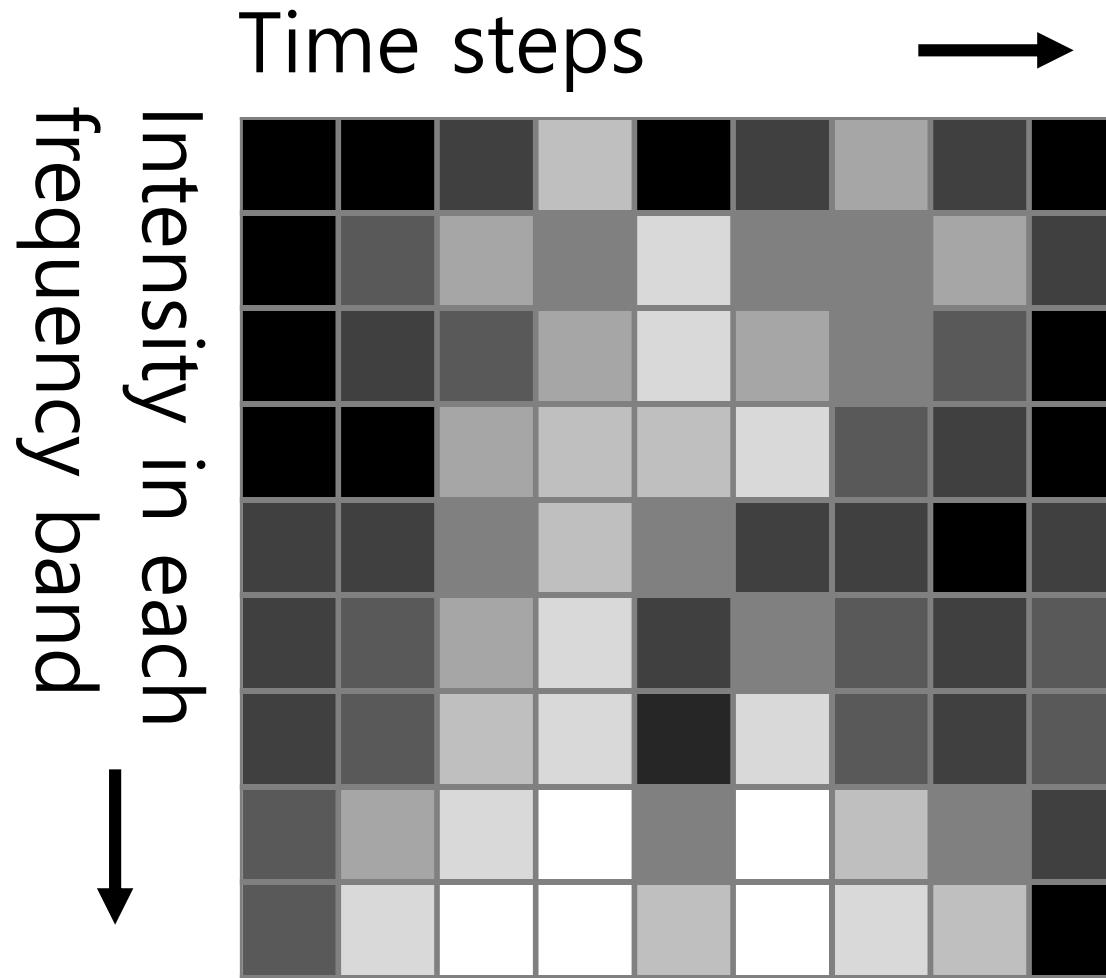
Any 2D (or 3D) data.

Things closer together are more closely related than things far away.

# Images



# Sound

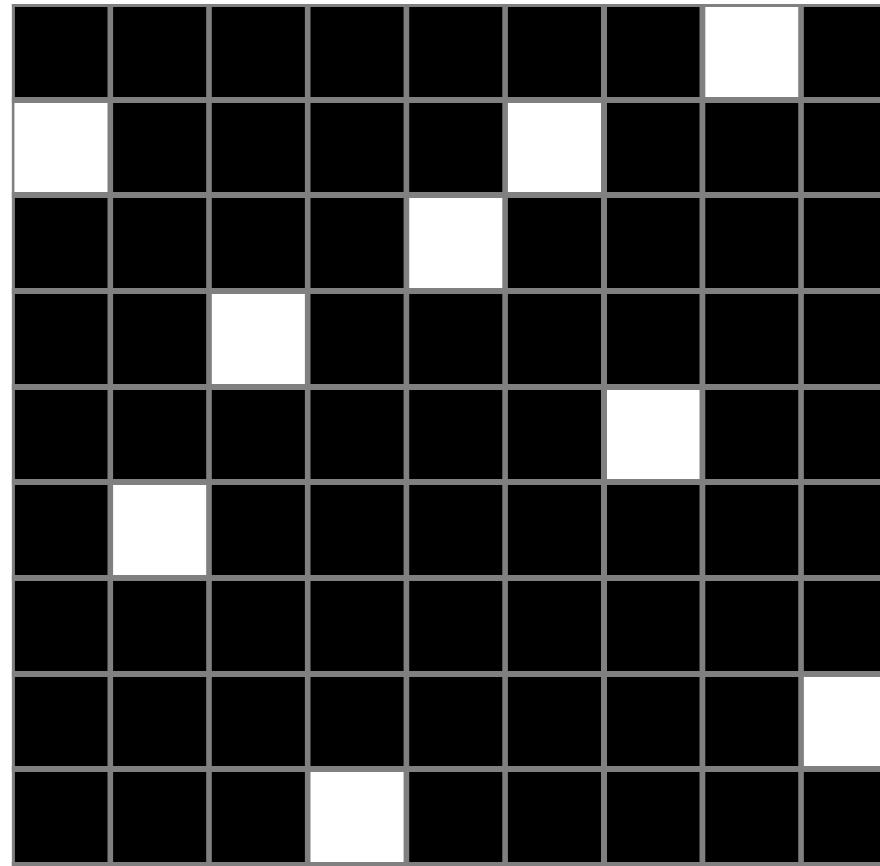


Text

Position in  
sentence



Words in  
dictionary



# Limitations

ConvNets only capture local “spatial” patterns in data.  
If the data can’t be made to look like an image, ConvNets are less useful.

# Customer data

Name, age,  
address, email,  
purchases,  
browsing activity,...



Customers

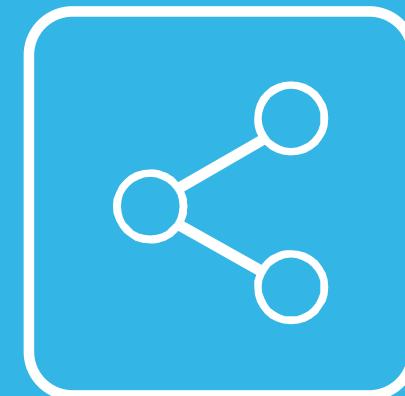


A	22	1A	<a href="mailto:a@a">a@a</a>	1	aa	a1.a	123	aa1
B	33	2B	<a href="mailto:b@b">b@b</a>	2	bb	b2.b	234	bb2
C	44	3C	<a href="mailto:c@c">c@c</a>	3	cc	c3.c	345	cc3
D	55	4D	<a href="mailto:d@d">d@d</a>	4	dd	d4.d	456	dd4
E	66	5E	<a href="mailto:e@e">e@e</a>	5	ee	e5.e	567	ee5
F	77	6F	<a href="mailto:f@f">f@f</a>	6	ff	f6.f	678	ff6
G	88	7G	<a href="mailto:g@g">g@g</a>	7	gg	g7.g	789	gg7
H	99	8H	<a href="mailto:h@h">h@h</a>	8	hh	h8.h	890	hh8
I	111	9I	<a href="mailto:i@i">i@i</a>	9	ii	i9.i	901	ii9

# Rule of thumb

If your data is just as useful after swapping any of your columns with each other, then you can't use Convolutional Neural Networks.

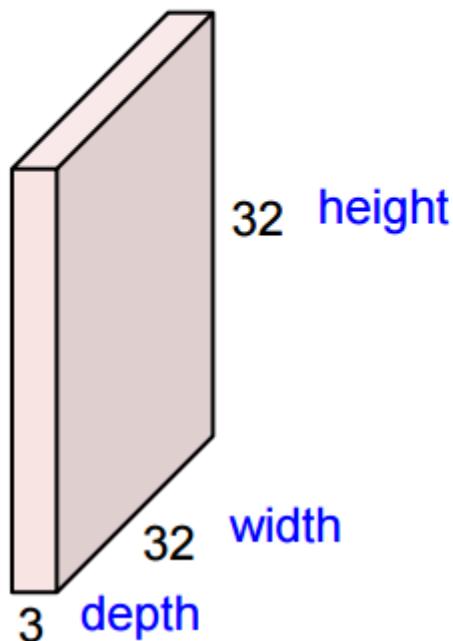
# 03 Convolutional Neural Network2



# Convolution Layer

## Convolution Layer

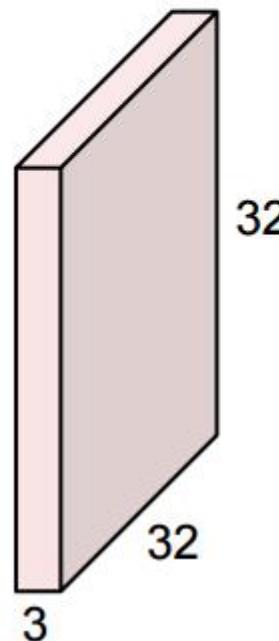
32x32x3 image



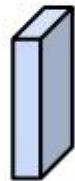
# Convolution Layer

## Convolution Layer

32x32x3 image



5x5x3 filter

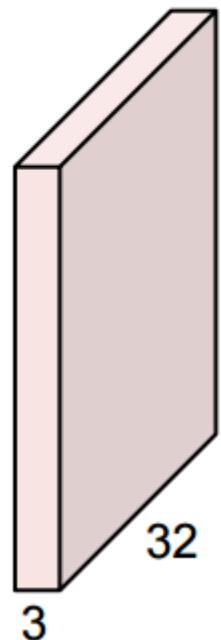


**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
computing dot products”

# Convolution Layer

## Convolution Layer

32x32x3 image



5x5x3 filter

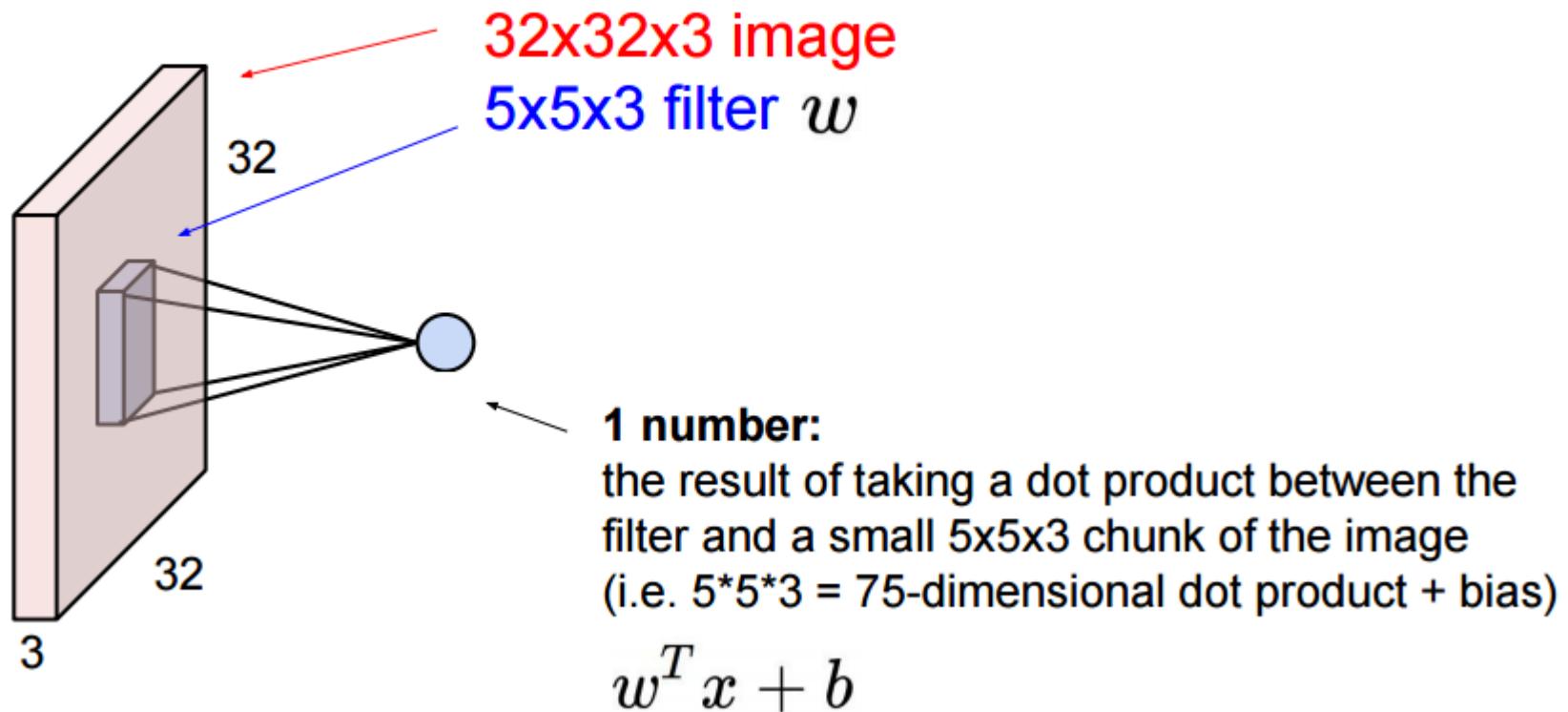


Filters always extend the full depth of the input volume

**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
computing dot products”

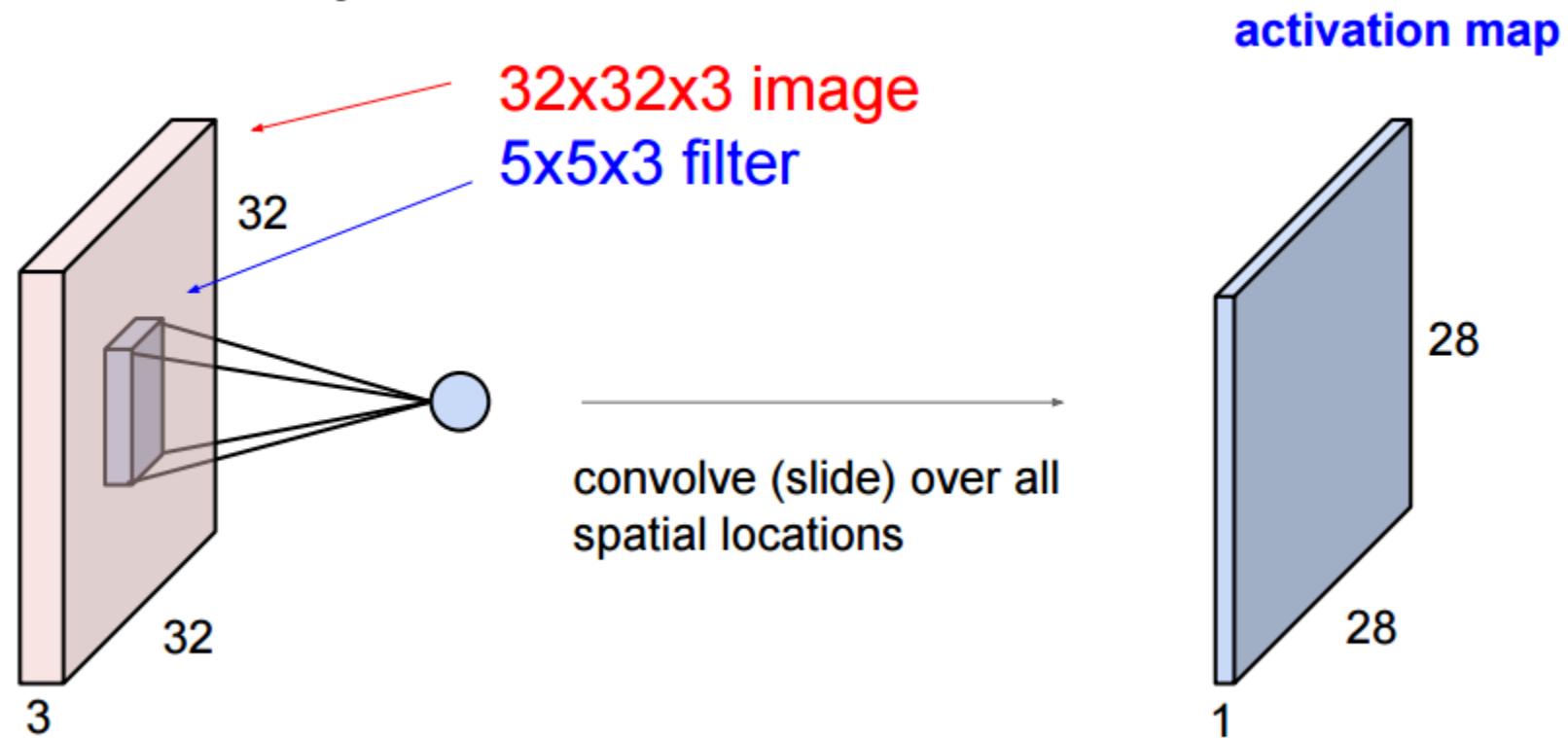
# Convolution Layer

## Convolution Layer



# Convolution Layer

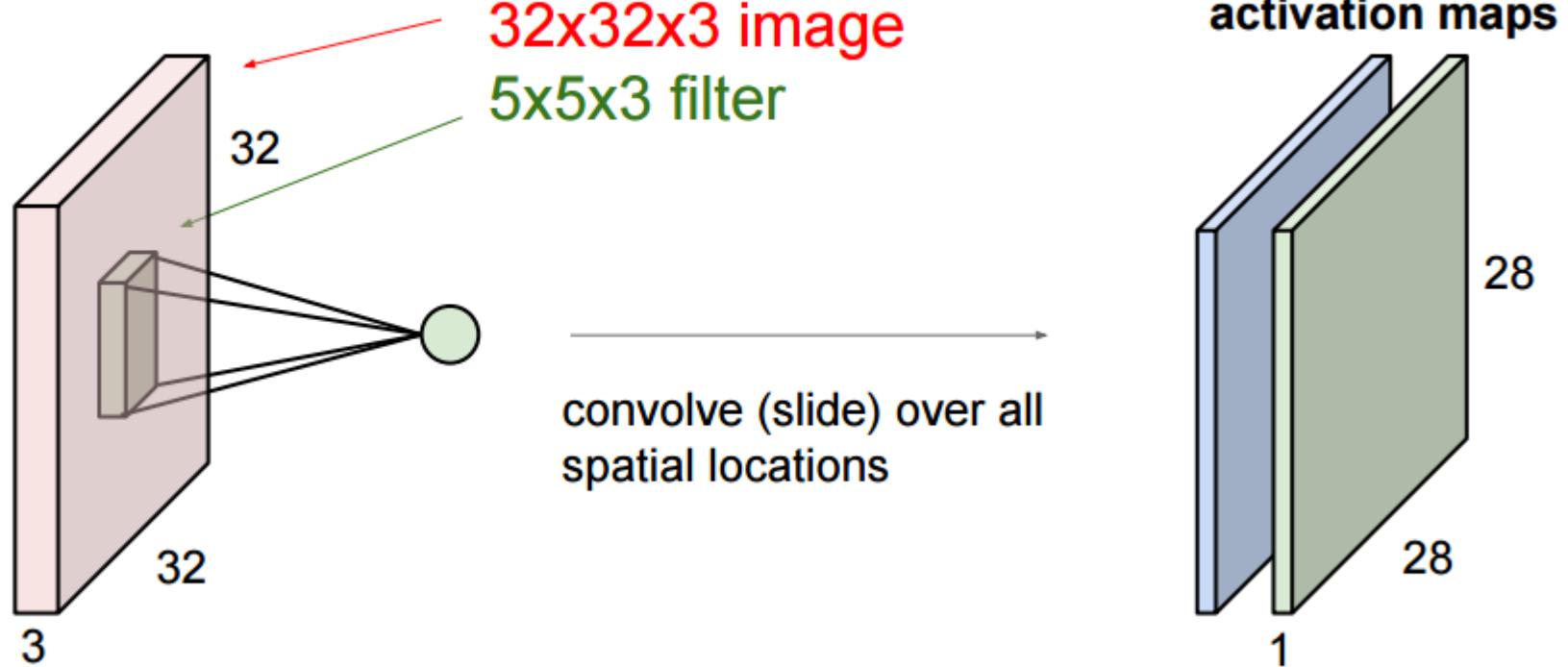
## Convolution Layer



# Convolution Layer

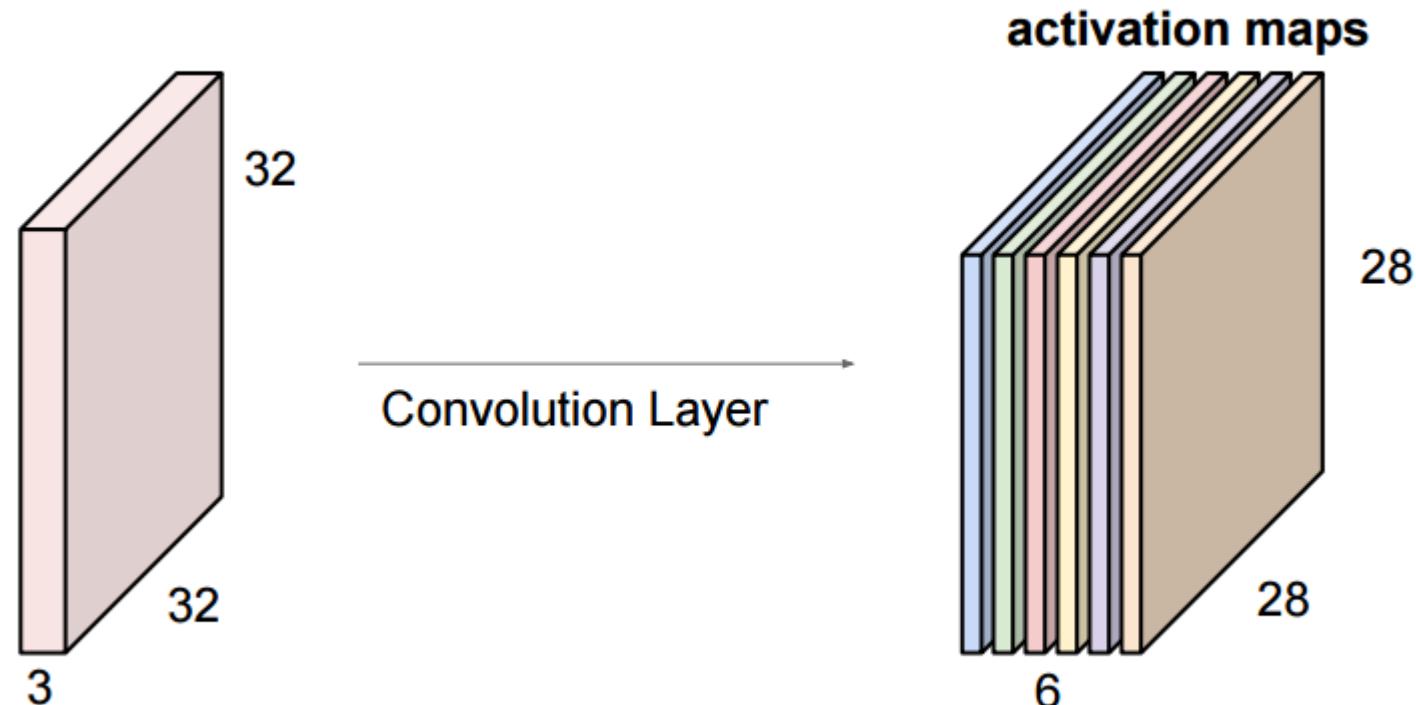
## Convolution Layer

consider a second, green filter



# Convolution Layer

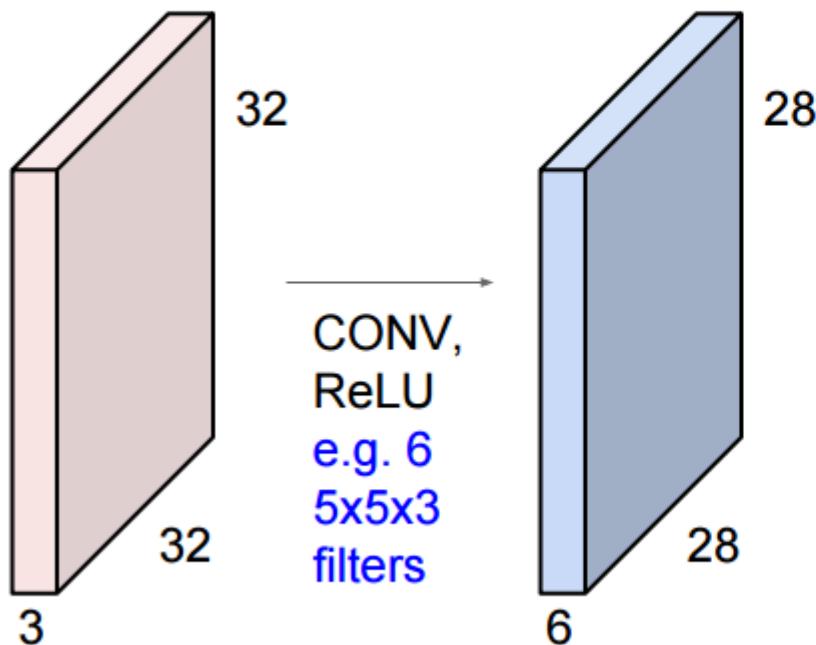
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a “new image” of size 28x28x6!

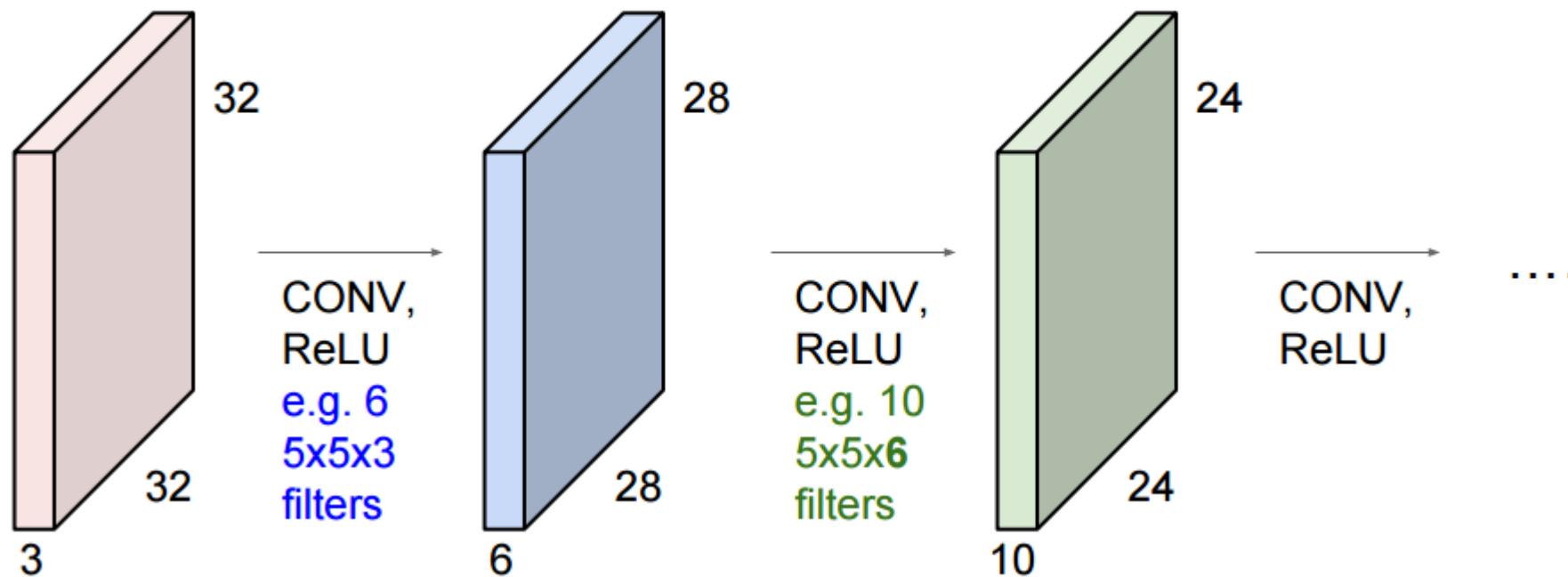
# Convolution Layer

**Preview:** ConvNet is a sequence of Convolution Layers, interspersed with activation functions



# Convolution Layer

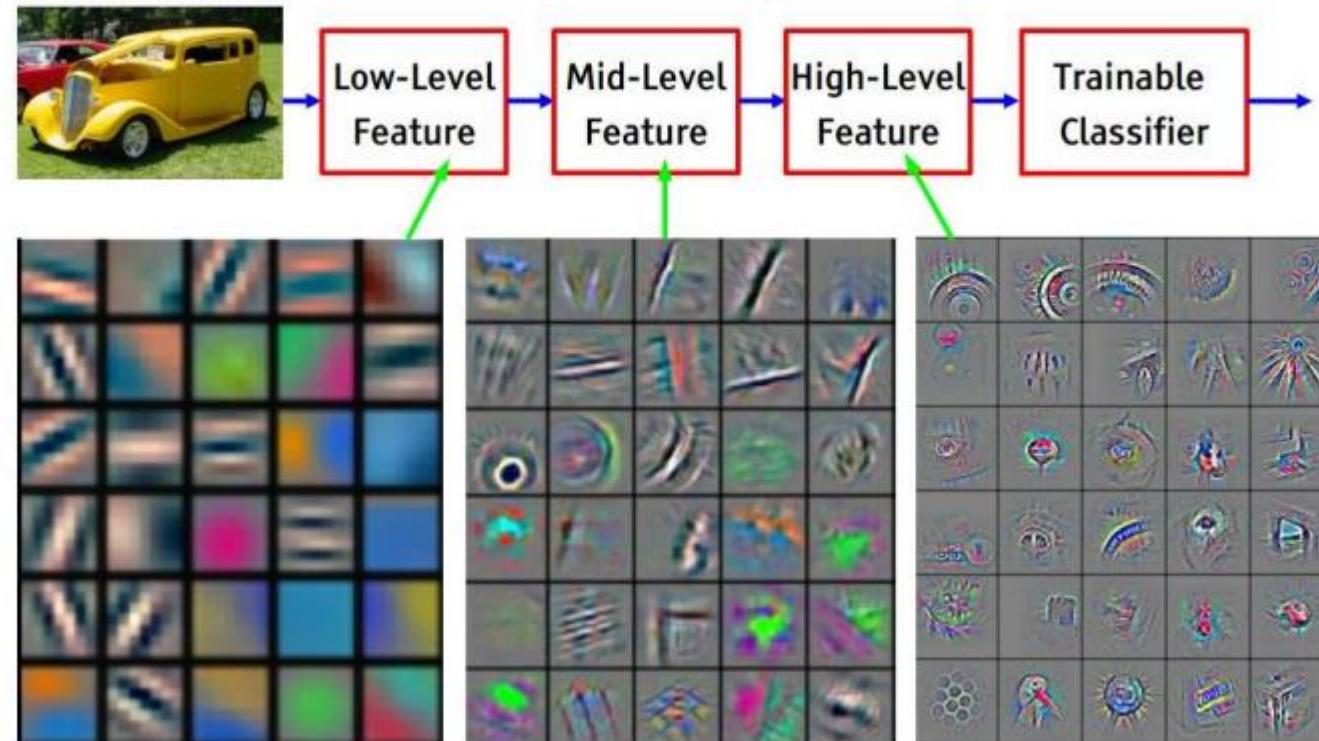
**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



# Feature Visualization

## Preview

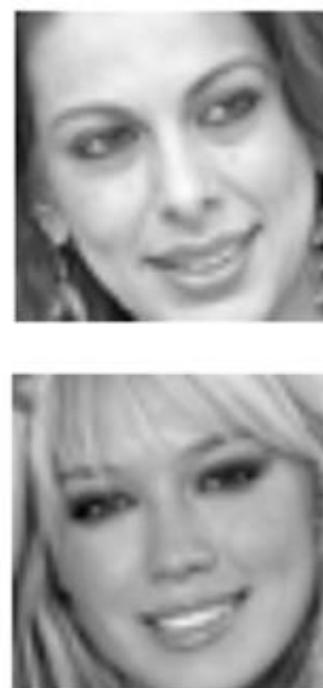
[From recent Yann LeCun slides]



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

# Feature Visualization

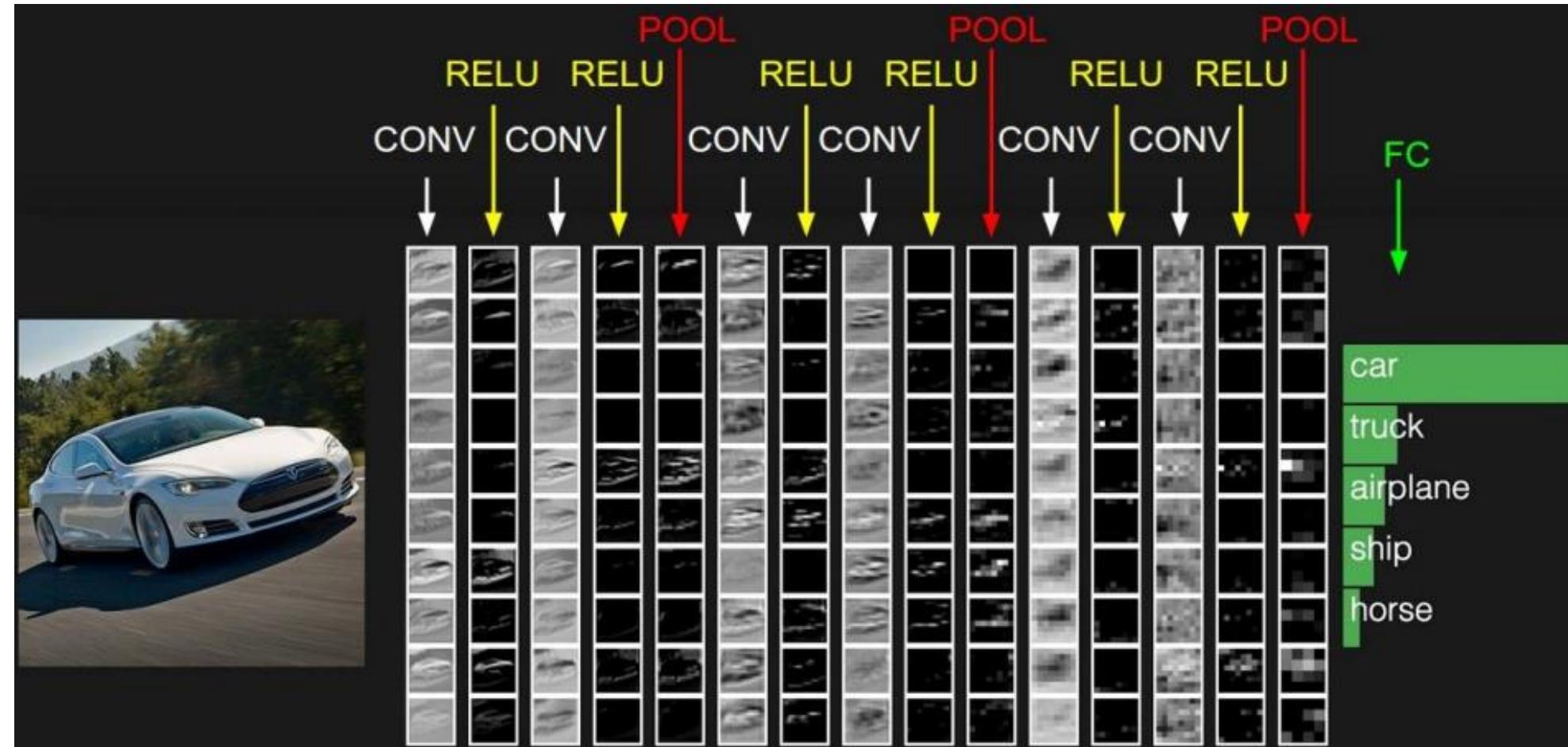
Each filter's role can be detected by deconvolution.



Visualization of patterns learned by the 26<sup>th</sup> filter in conv1 layer. (detecting edges)

Visualization of patterns learned by the 2<sup>nd</sup> filter in conv2 layer. (detecting eyes and lips)

# Entire Layers in Convnet



# Convolution Layer

In practice: Common to zero pad the border

0	0	0	0	0	0		
0							
0							
0							
0							

e.g. input 7x7

3x3 filter, applied with **stride 1**

**pad with 1 pixel border => what is the output?**

(recall:)

$$(N - F) / \text{stride} + 1$$

# Convolution Layer

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

**3x3** filter, applied with **stride 1**

**pad with 1 pixel** border => what is the output?

**7x7 output!**

# Convolution Layer

In practice: Common to zero pad the border

0	0	0	0	0	0	0			
0									
0									
0									
0									

e.g. input 7x7

3x3 filter, applied with **stride 1**

**pad with 1 pixel border => what is the output?**

**7x7 output!**

in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with  $(F-1)/2$ . (will preserve size spatially)

e.g.  $F = 3 \Rightarrow$  zero pad with 1

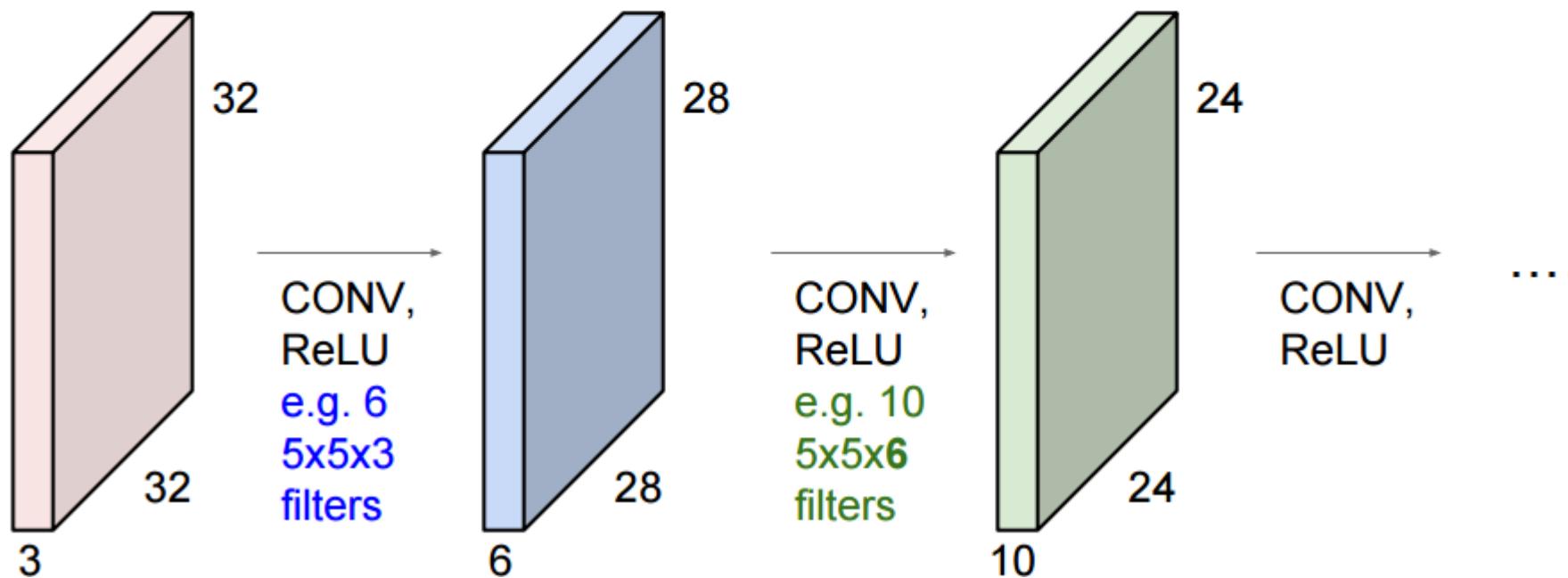
$F = 5 \Rightarrow$  zero pad with 2

$F = 7 \Rightarrow$  zero pad with 3

# Convolution Layer

**Remember back to...**

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!  
(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.

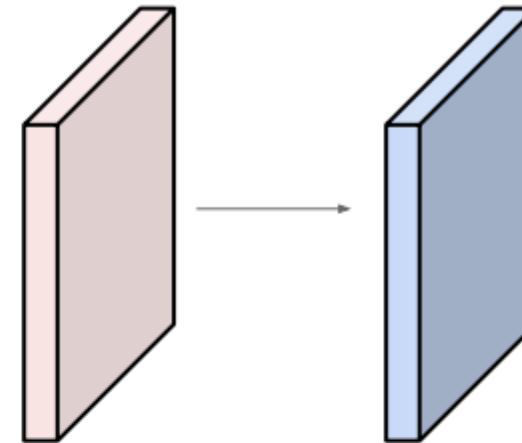


# Convolution Layer

Examples time:

Input volume: **32x32x3**  
10 5x5 filters with stride 1, pad 2

Output volume size: ?

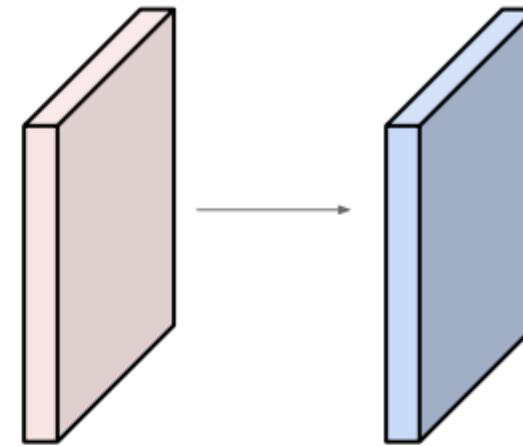


# Convolution Layer

Examples time:

Input volume: **32x32x3**

10 **5x5** filters with stride **1**, pad **2**



Output volume size:

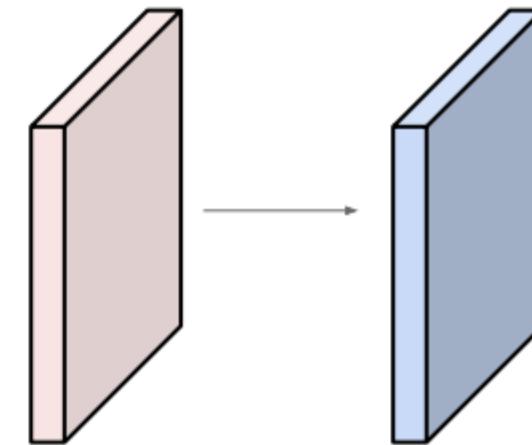
$(32+2*2-5)/1+1 = 32$  spatially, so

**32x32x10**

# Convolution Layer

Examples time:

Input volume: **32x32x3**  
10 5x5 filters with stride 1, pad 2



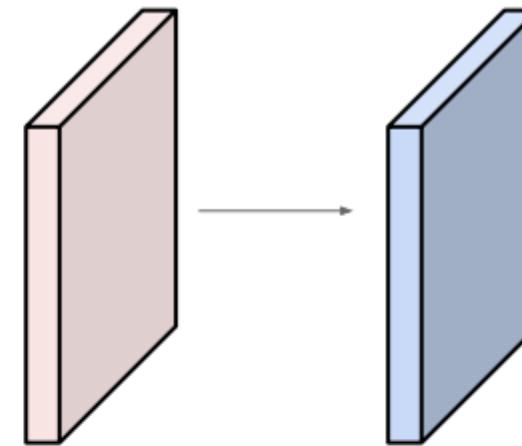
Number of parameters in this layer?

# Convolution Layer

Examples time:

Input volume: **32x32x3**

**10 5x5** filters with stride 1, pad 2



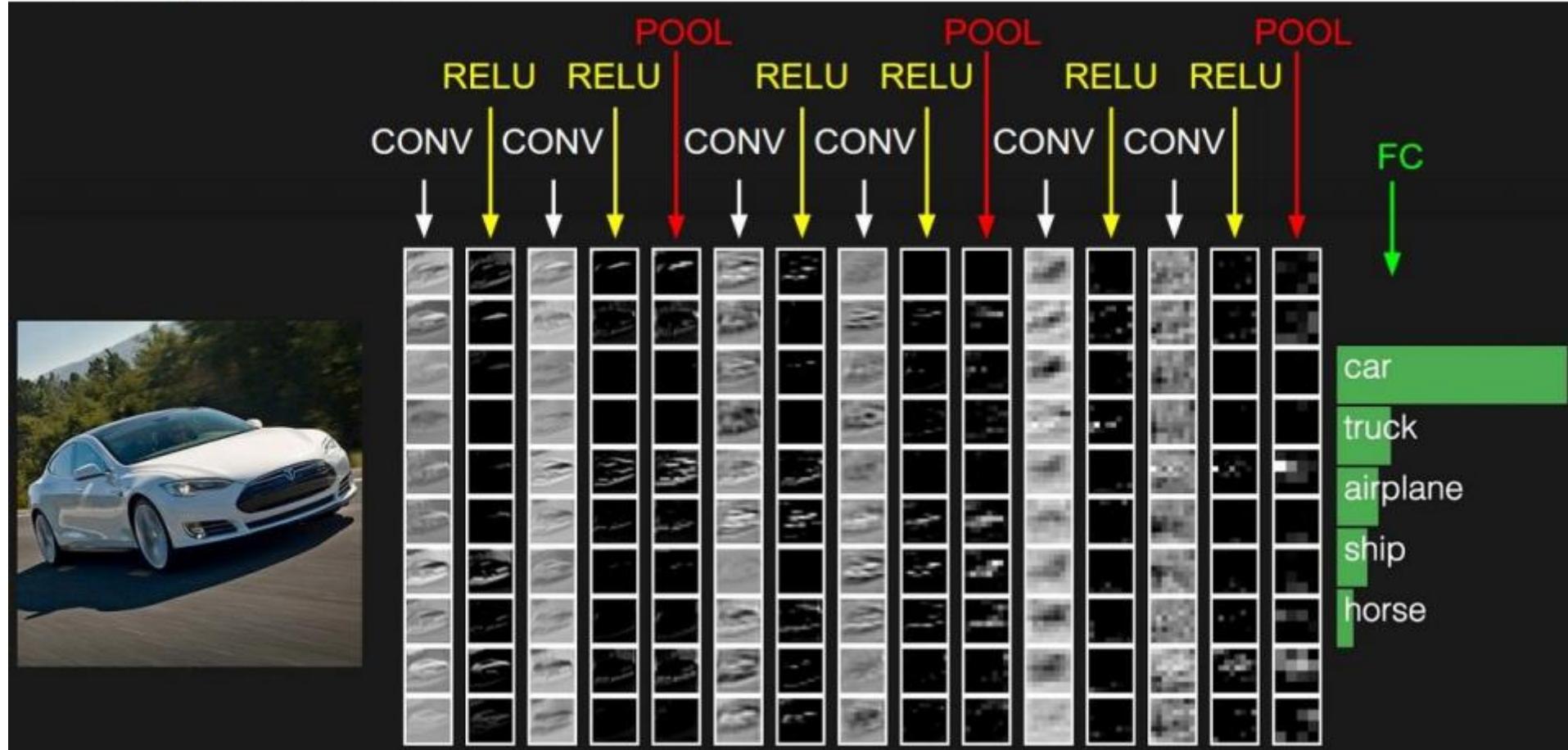
Number of parameters in this layer?

each filter has  $5*5*3 + 1 = 76$  params (+1 for bias)

$$\Rightarrow 76 * 10 = 760$$

# Convolution Layer

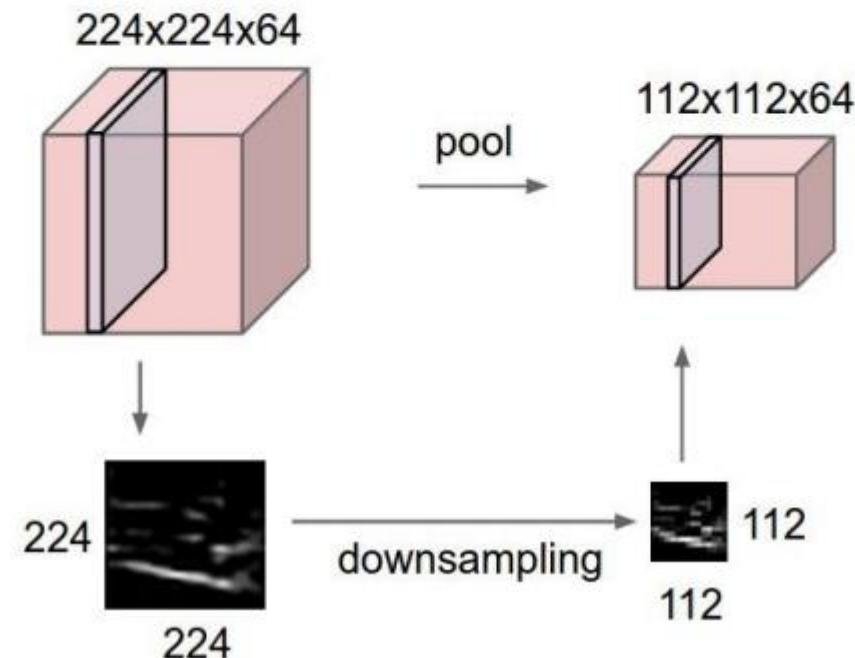
two more layers to go: POOL/FC



# Pooling Layer

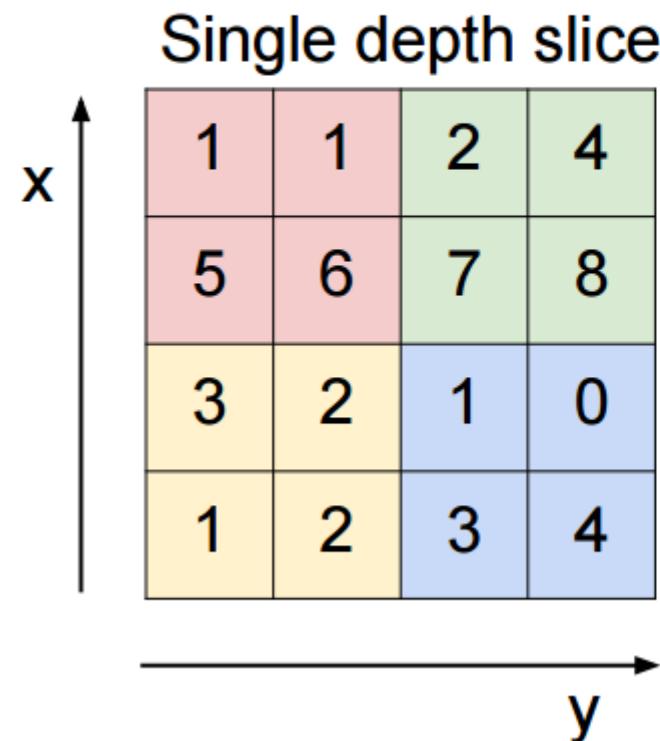
## Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



# Pooling Layer

## MAX POOLING



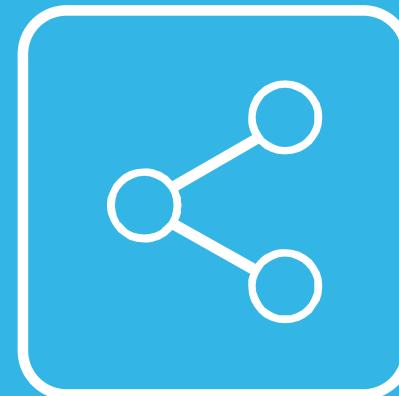
max pool with 2x2 filters  
and stride 2

An arrow points from the input matrix to the output matrix, indicating the transformation process.

6	8
3	4

# 05

# Advanced CNN Architectures

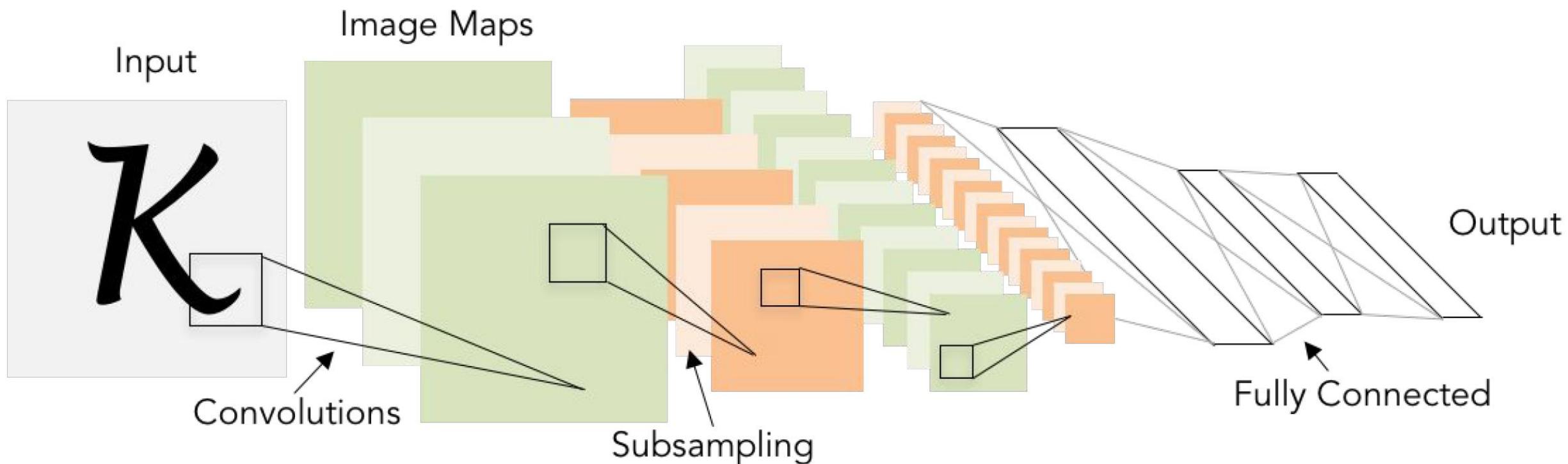


# CNN Architectures

## Case Studies

- AlexNet
- VGG
- GoogLeNet
- ResNet
- ...

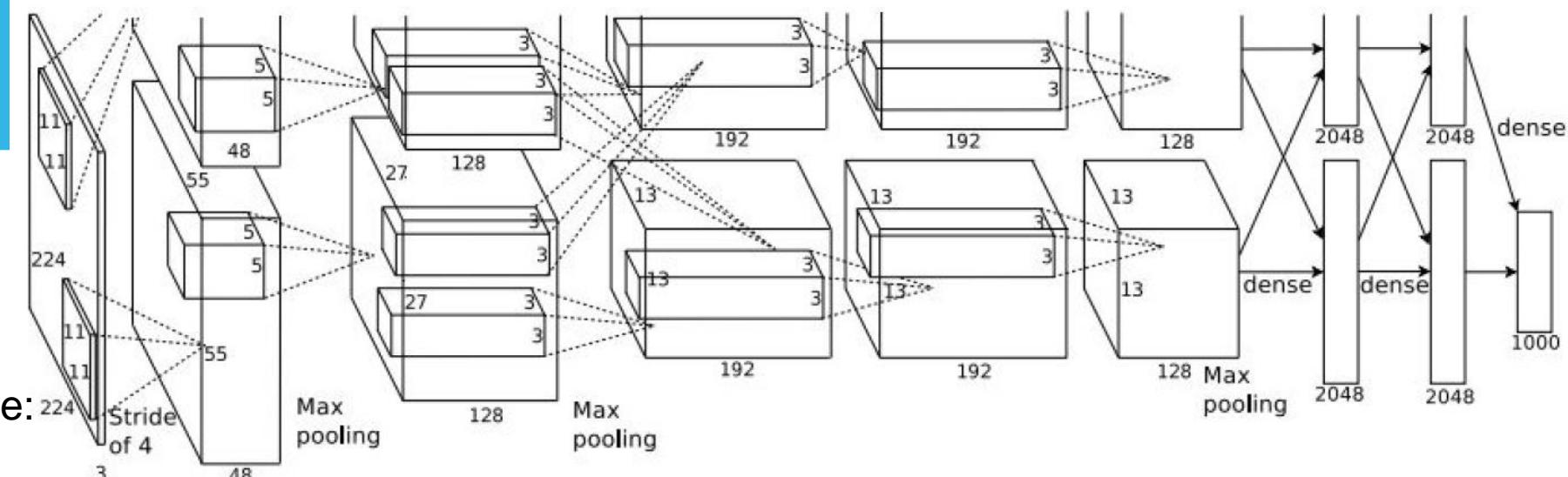
# Review: LeNet-5 [LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1

Subsampling (Pooling) layers were 2x2 applied at stride 2  
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

# Case Study: AlexNet [Krizhevsky et al. 2012]

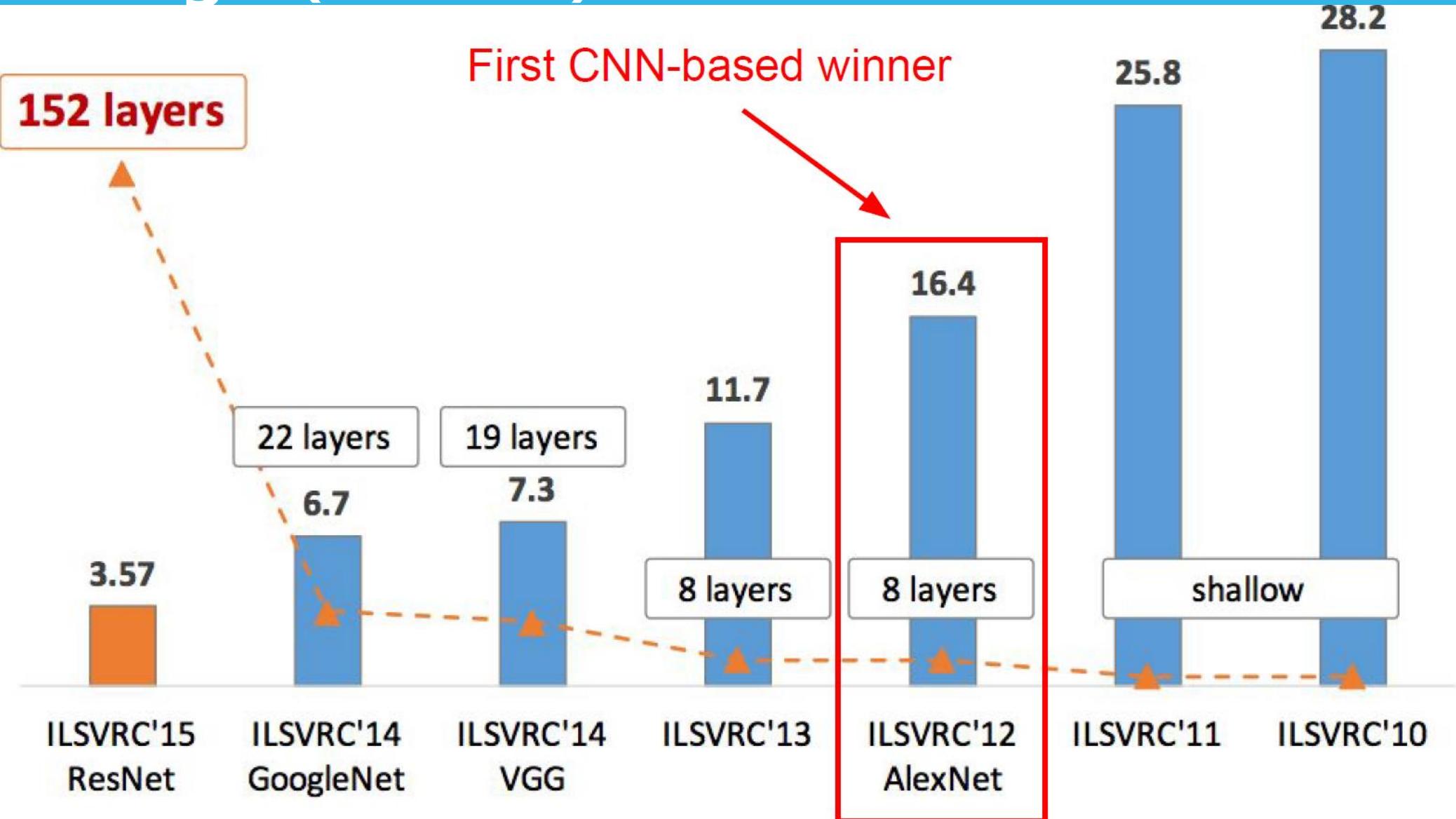


Full (simplified) AlexNet architecture:  
[227x227x3] INPUT  
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0  
[27x27x96] MAX POOL1: 3x3 filters at stride 2  
[27x27x96] NORM1: Normalization layer  
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2  
[13x13x256] MAX POOL2: 3x3 filters at stride 2  
[13x13x256] NORM2: Normalization layer  
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1  
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1  
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1  
[6x6x256] MAX POOL3: 3x3 filters at stride 2  
[4096] FC6: 4096 neurons  
[4096] FC7: 4096 neurons  
[1000] FC8: 1000 neurons (class scores)

## Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



# Case Study: VGGNet [Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)

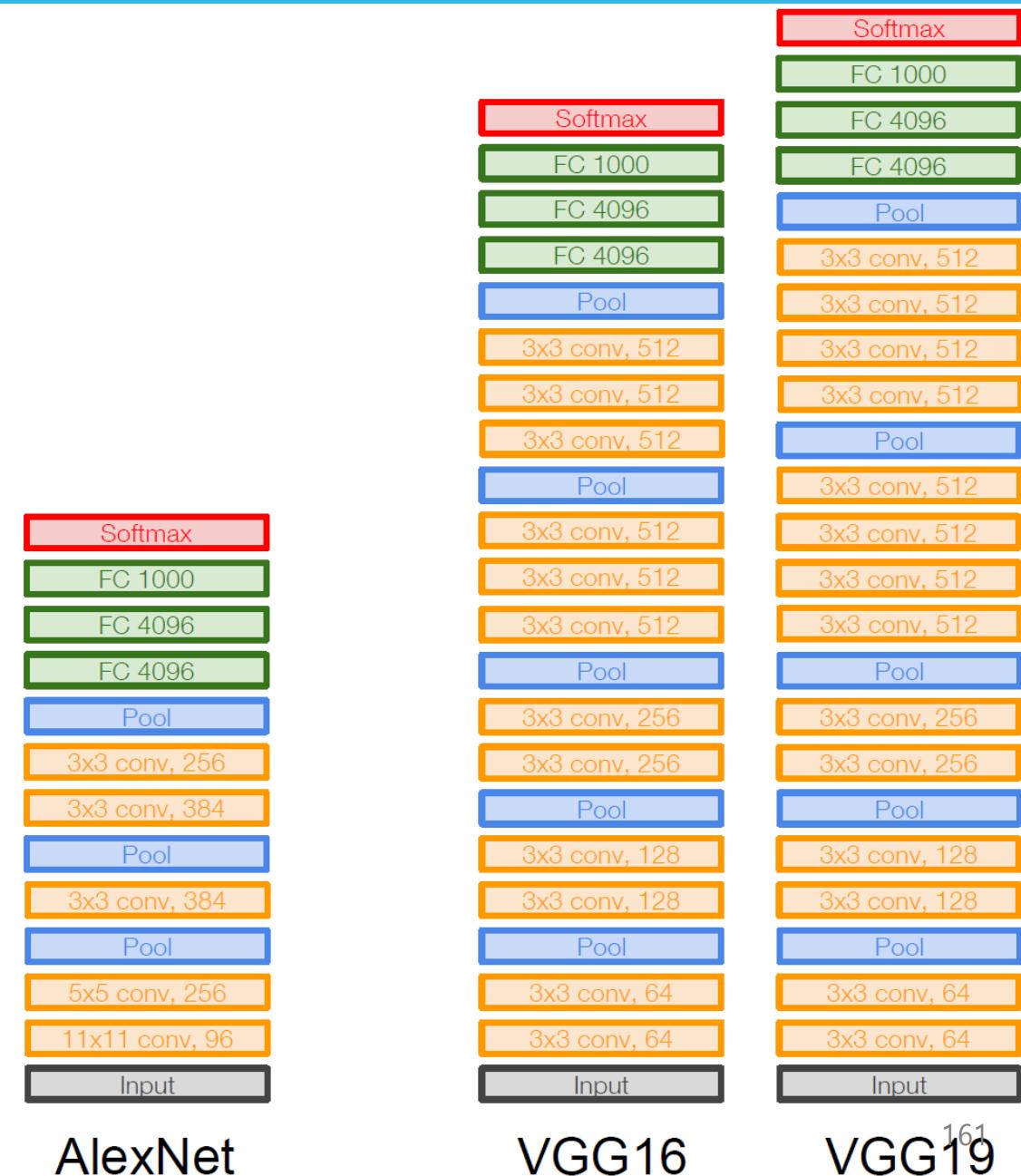
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1  
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13

(ZFNet)

-> 7.3% top 5 error in ILSVRC'14



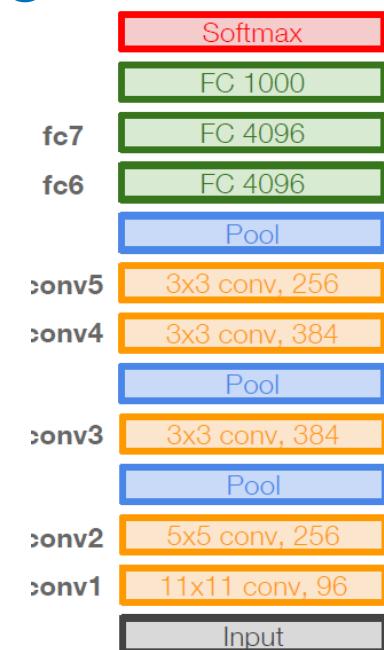
# Case Study: VGGNet [Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

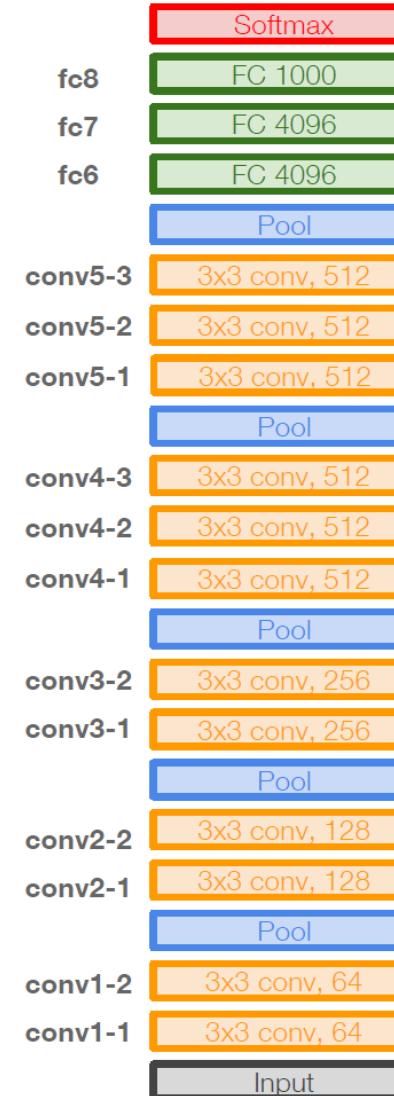
Stack of three 3x3 conv (stride 1)  
layers has same effective receptive  
field as one 7x7 conv layer

But deeper, more non-linearities

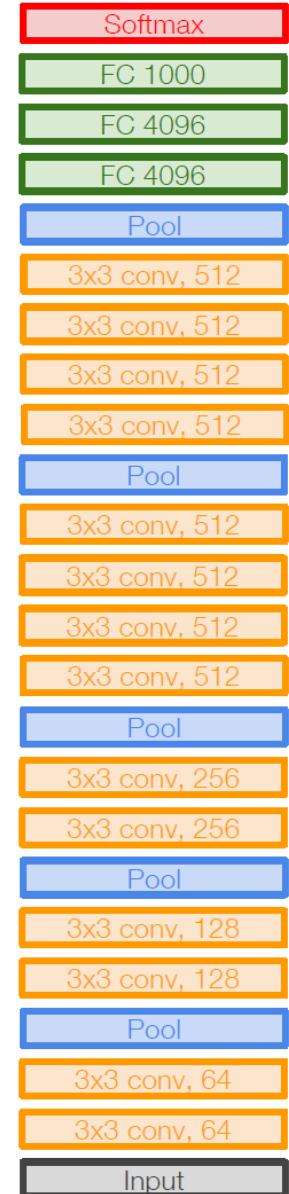
And fewer parameters:  $3 * (3^2 C^2)$   
vs.  $7^2 C^2$  for C channels per layer



AlexNet

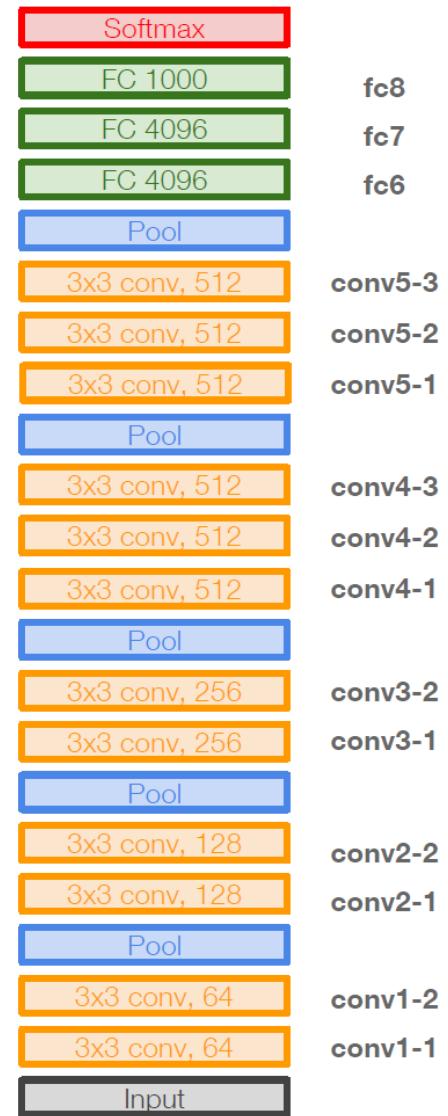


VGG16



VGG19

INPUT: [224x224x3] memory: 224\*224\*3=150K params: 0  
 CONV3-64: [224x224x64] memory: 224\*224\*64=3.2M params: (3\*3\*3)\*64 = 1,728  
 CONV3-64: [224x224x64] memory: 224\*224\*64=3.2M params: (3\*3\*64)\*64 = 36,864  
 POOL2: [112x112x64] memory: 112\*112\*64=800K params: 0  
 CONV3-128: [112x112x128] memory: 112\*112\*128=1.6M params: (3\*3\*64)\*128 = 73,728  
 CONV3-128: [112x112x128] memory: 112\*112\*128=1.6M params: (3\*3\*128)\*128 = 147,456  
 POOL2: [56x56x128] memory: 56\*56\*128=400K params: 0  
 CONV3-256: [56x56x256] memory: 56\*56\*256=800K params: (3\*3\*128)\*256 = 294,912  
 CONV3-256: [56x56x256] memory: 56\*56\*256=800K params: (3\*3\*256)\*256 = 589,824  
 CONV3-256: [56x56x256] memory: 56\*56\*256=800K params: (3\*3\*256)\*256 = 589,824  
 POOL2: [28x28x256] memory: 28\*28\*256=200K params: 0  
 CONV3-512: [28x28x512] memory: 28\*28\*512=400K params: (3\*3\*256)\*512 = 1,179,648  
 CONV3-512: [28x28x512] memory: 28\*28\*512=400K params: (3\*3\*512)\*512 = 2,359,296  
 CONV3-512: [28x28x512] memory: 28\*28\*512=400K params: (3\*3\*512)\*512 = 2,359,296  
 POOL2: [14x14x512] memory: 14\*14\*512=100K params: 0  
 CONV3-512: [14x14x512] memory: 14\*14\*512=100K params: (3\*3\*512)\*512 = 2,359,296  
 CONV3-512: [14x14x512] memory: 14\*14\*512=100K params: (3\*3\*512)\*512 = 2,359,296  
 CONV3-512: [14x14x512] memory: 14\*14\*512=100K params: (3\*3\*512)\*512 = 2,359,296  
 POOL2: [7x7x512] memory: 7\*7\*512=25K params: 0  
 FC: [1x1x4096] memory: 4096 params: 7\*7\*512\*4096 = 102,760,448  
 FC: [1x1x4096] memory: 4096 params: 4096\*4096 = 16,777,216  
 FC: [1x1x1000] memory: 1000 params: 4096\*1000 = 4,096,000



VGG16

Common names

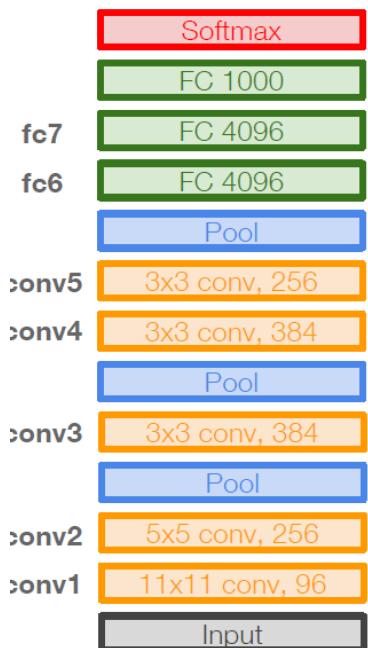
TOTAL memory: 24M \* 4 bytes ~= 96MB / image (only forward! ~\*2 for bwd)

TOTAL params: 138M parameters

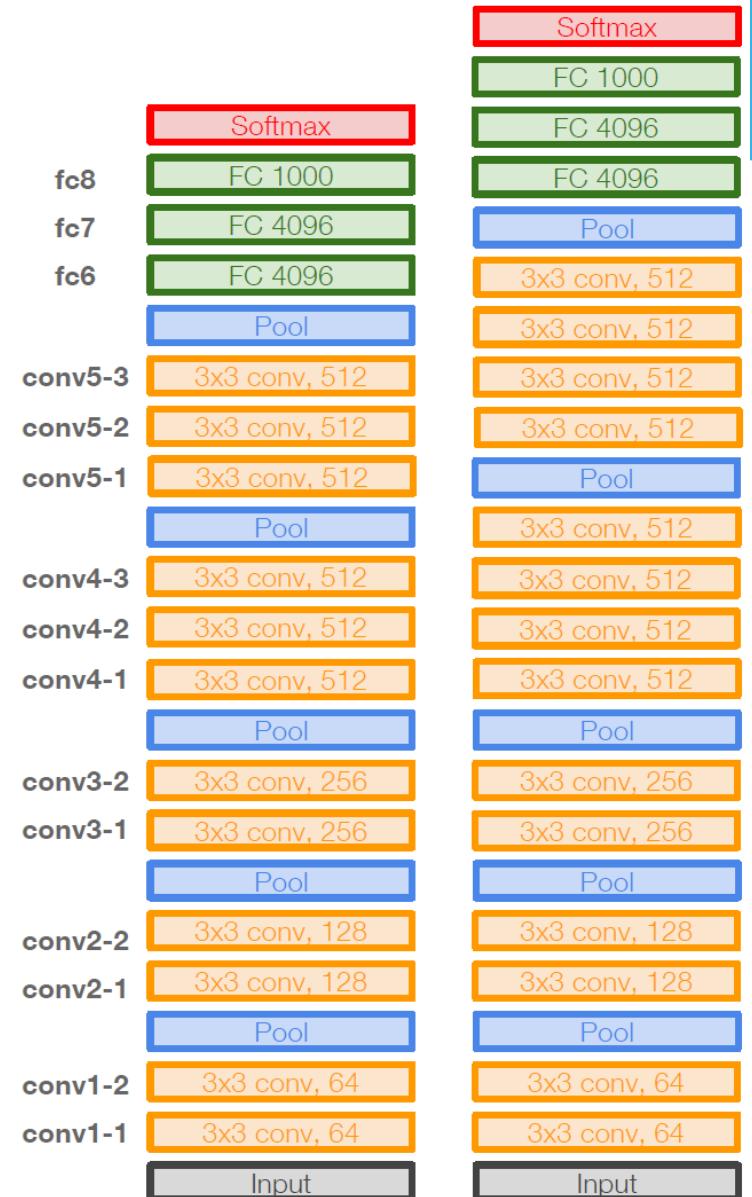
# Case Study: VGGNet [Simonyan and Zisserman, 2014]

Details:

- ILSVRC'14 2nd in classification,  
1st in localization
- Similar training procedure as  
Krizhevsky 2012
- No Local Response Normalisation  
(LRN)
- Use VGG16 or VGG19 (VGG19  
only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to  
other tasks



AlexNet



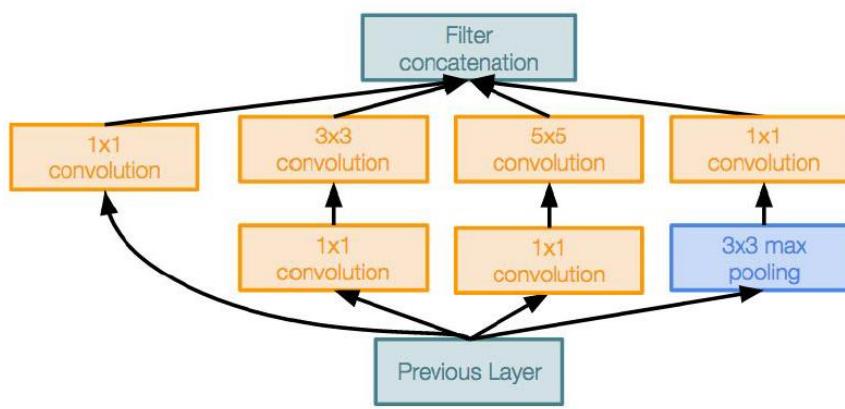
VGG16

VGG19

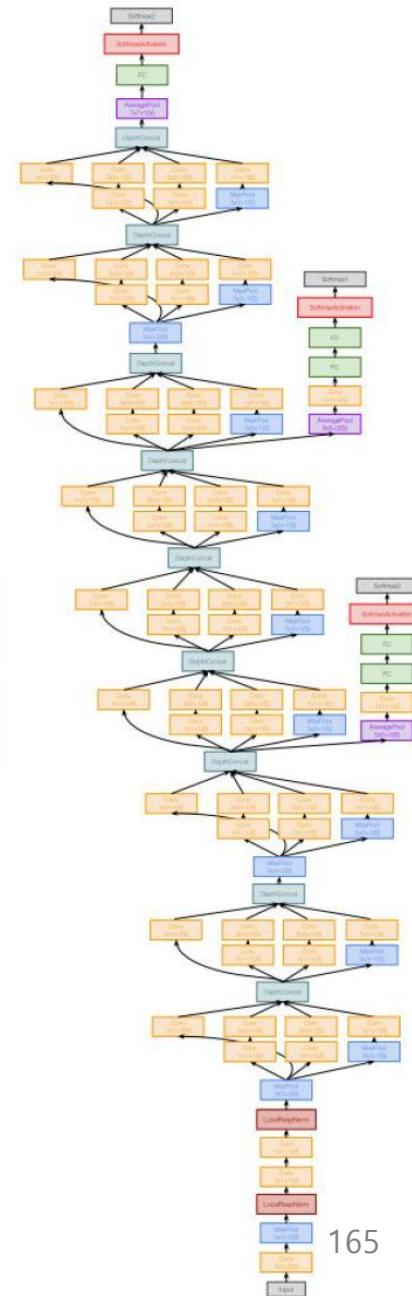
# Case Study: GoogLeNet [Szegedy et al., 2014]

# Deeper networks, with computational Efficiency

- 22 layers
  - Efficient “Inception” module
  - No FC layers
  - Only 5 million parameters!  
12x less than AlexNet
  - ILSVRC’14 classification winner  
(6.7% top 5 error)

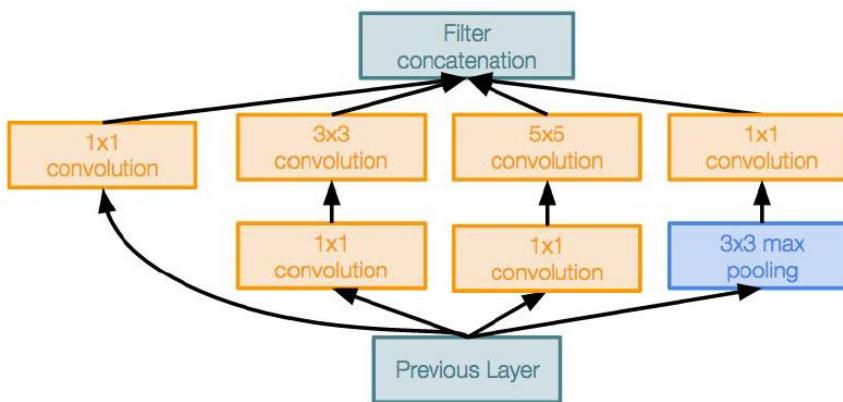


## Inception module

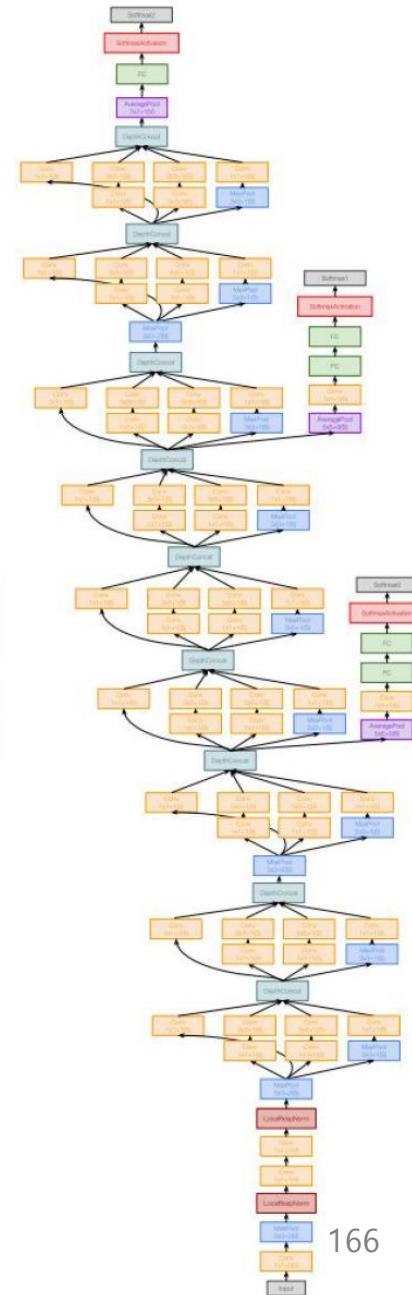


# Case Study: GoogLeNet [Szegedy et al., 2014]

“Inception module”: design a good local network topology (network within a network) and then stack these modules on top of each other.



Inception module

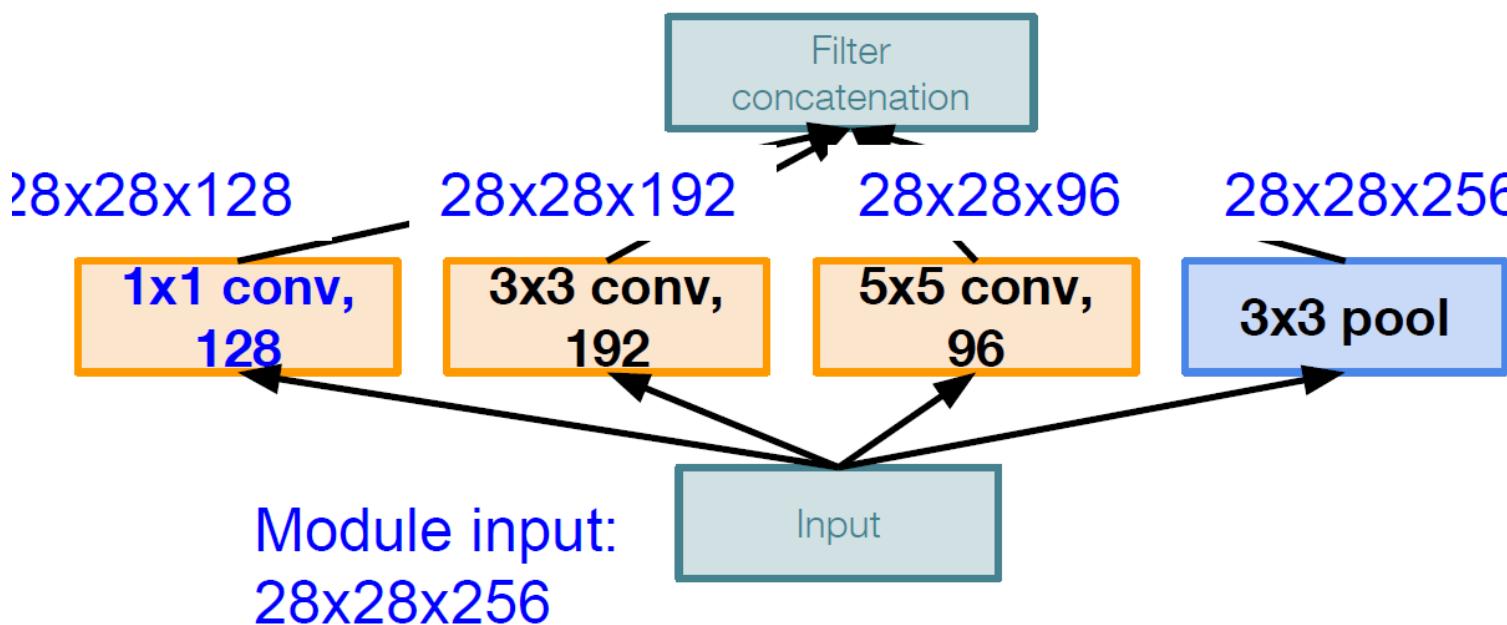


# Case Study: GoogLeNet [Szegedy et al., 2014]

Example:

Q3: What is output size after filter concatenation?

$$28 \times 28 \times (128 + 192 + 96 + 256) = \mathbf{529k}$$



Q: What is the problem with this?  
[Hint: Computational complexity]

Solution: “bottleneck” layers that use  $1 \times 1$  convolutions to reduce feature depth

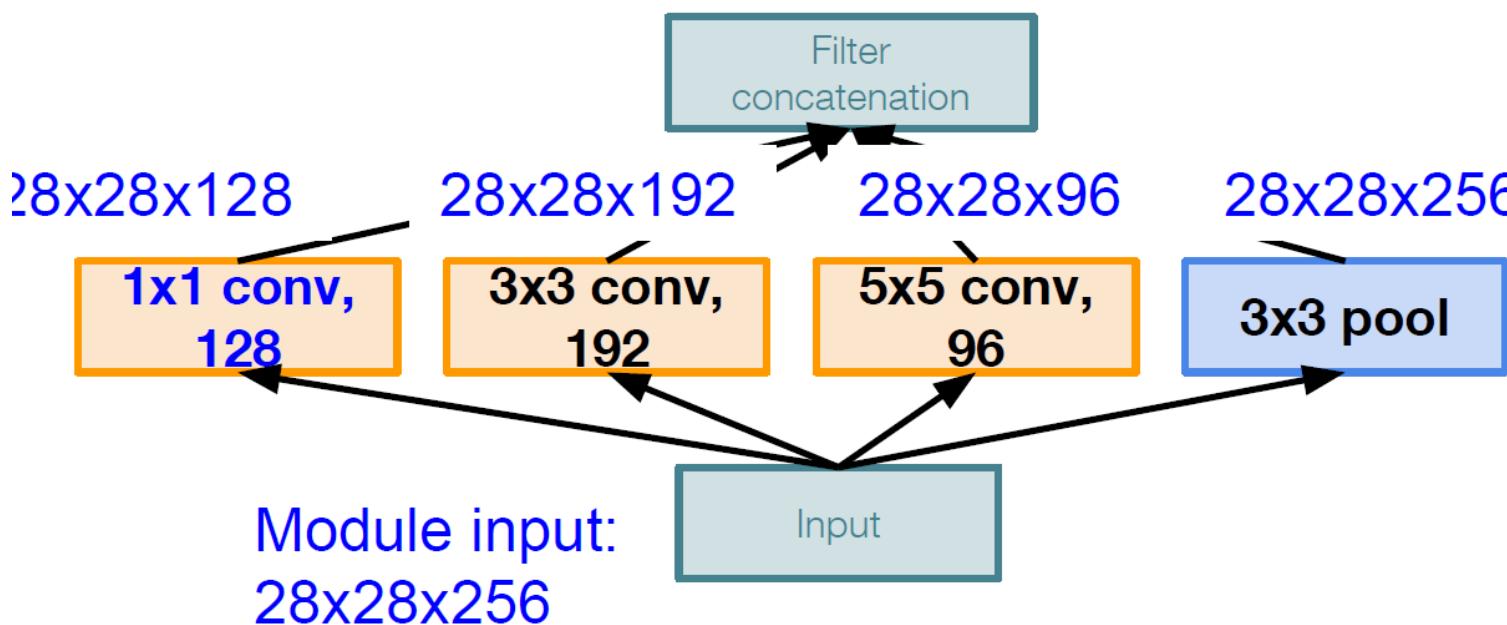
Naive Inception module

# Case Study: GoogLeNet [Szegedy et al., 2014]

Example:

Q3: What is output size after filter concatenation?

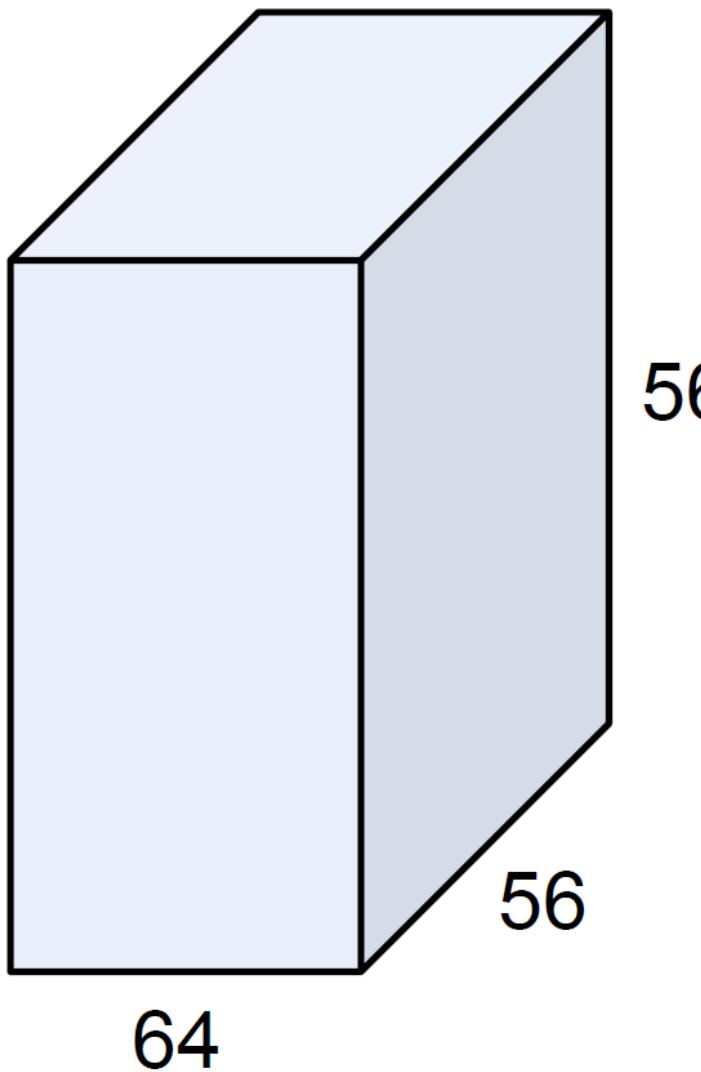
$$28 \times 28 \times (128 + 192 + 96 + 256) = \mathbf{529k}$$



Q: What is the problem with this?  
[Hint: Computational complexity]

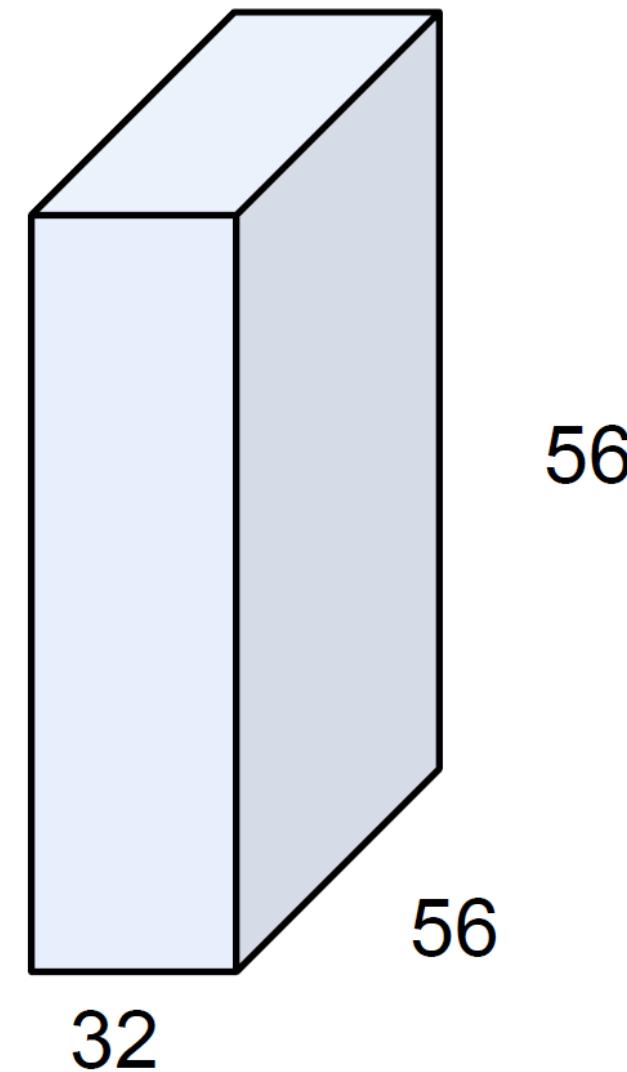
Solution: “bottleneck” layers that use  $1 \times 1$  convolutions to reduce feature depth

# 1x1 Convolutions

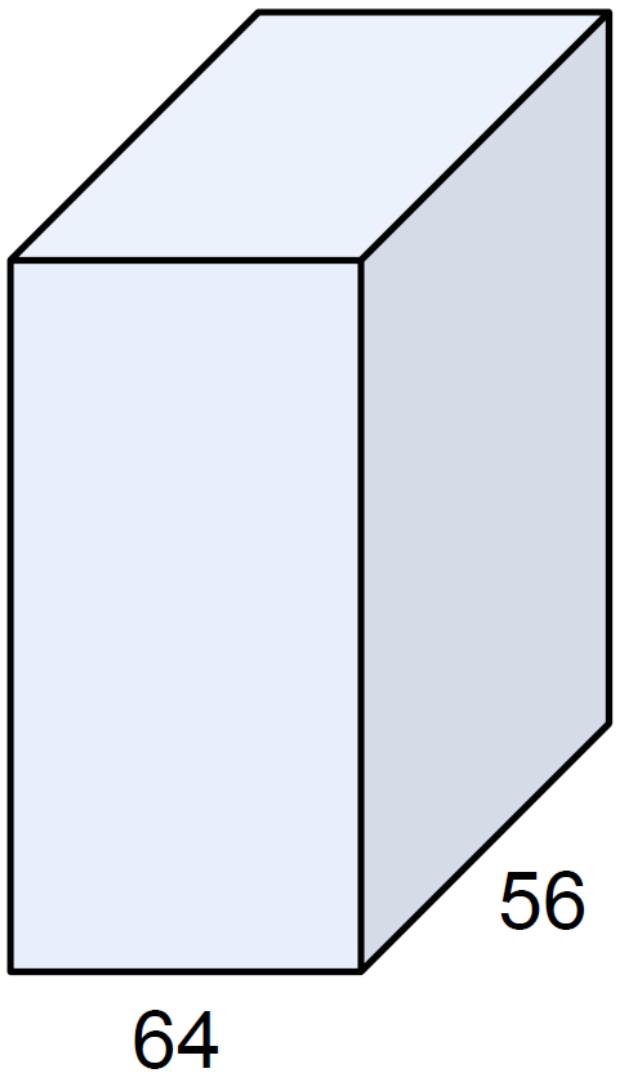


1x1 CONV  
with 32 filters

→  
(each filter has size  
1x1x64, and performs a  
64-dimensional dot  
product)

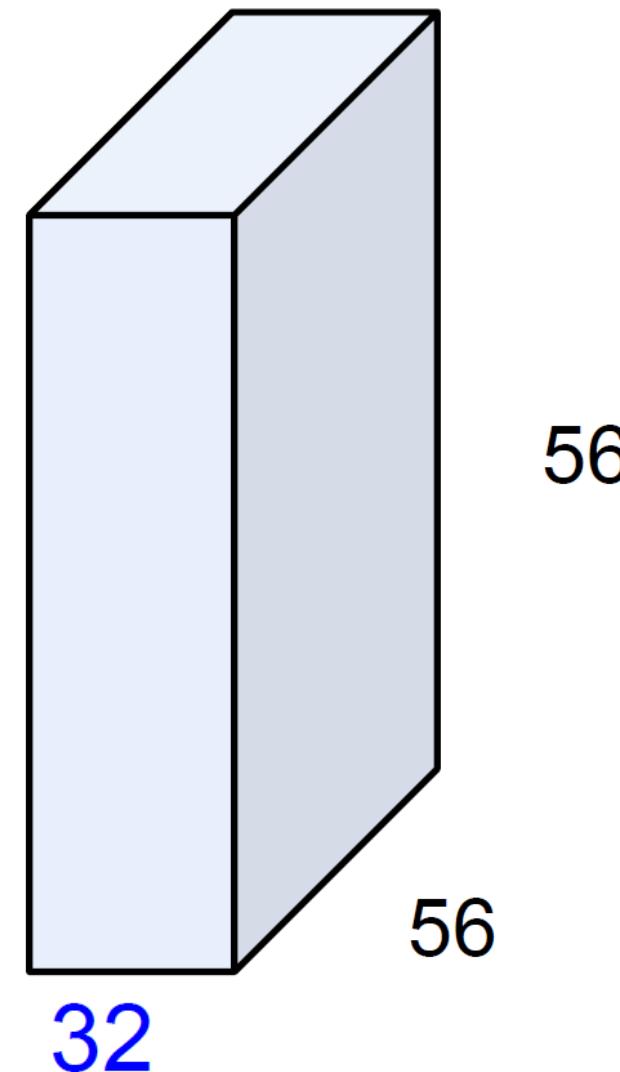


# 1x1 Convolutions

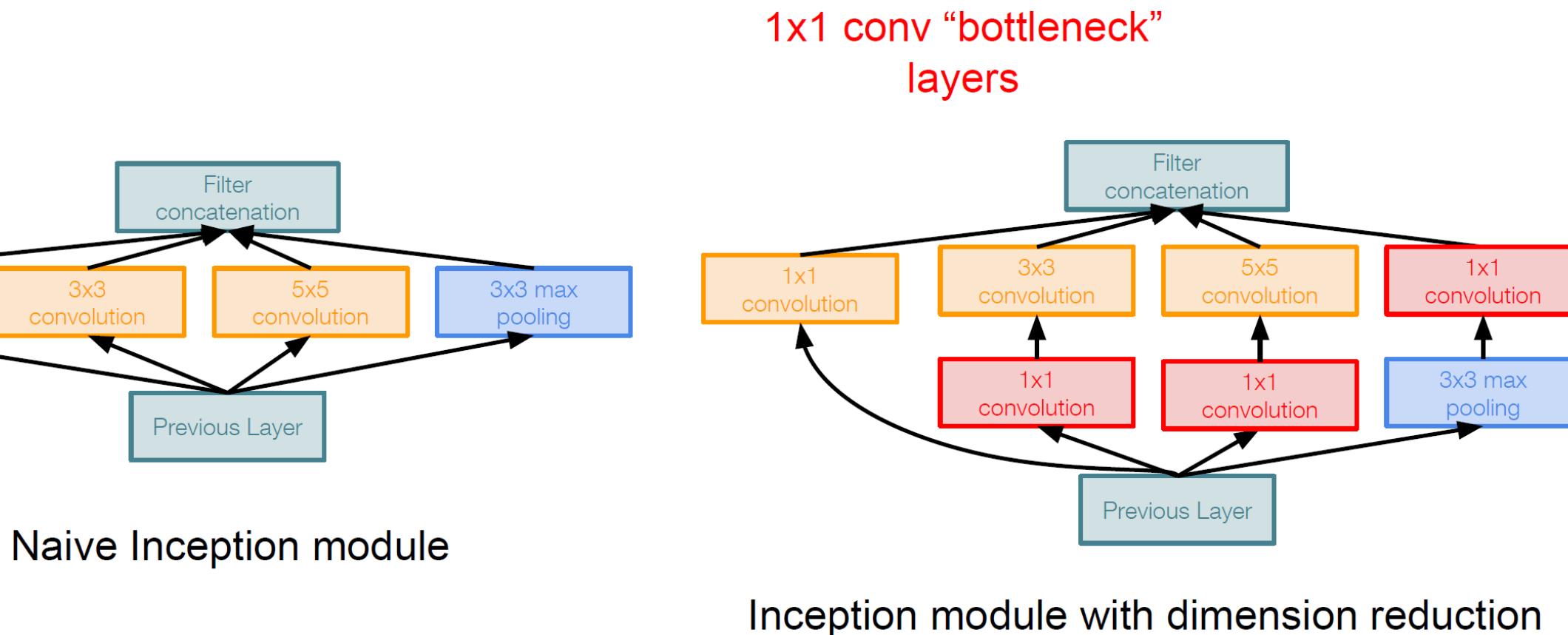


1x1 CONV  
with 32 filters

→  
preserves spatial  
dimensions, reduces depth!  
  
Projects depth to lower  
dimension (combination of  
feature maps)



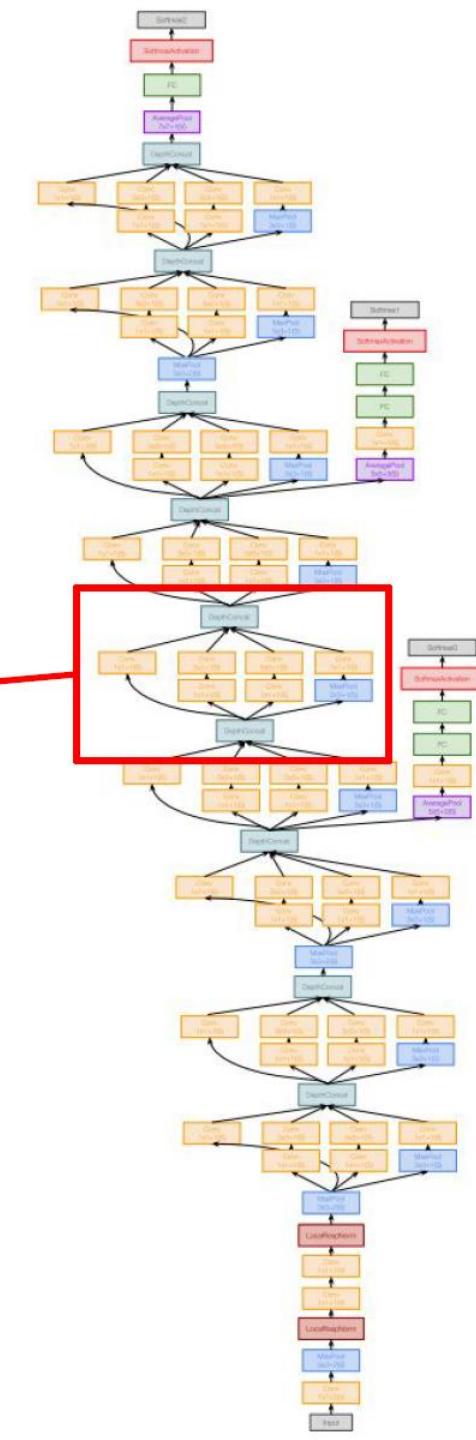
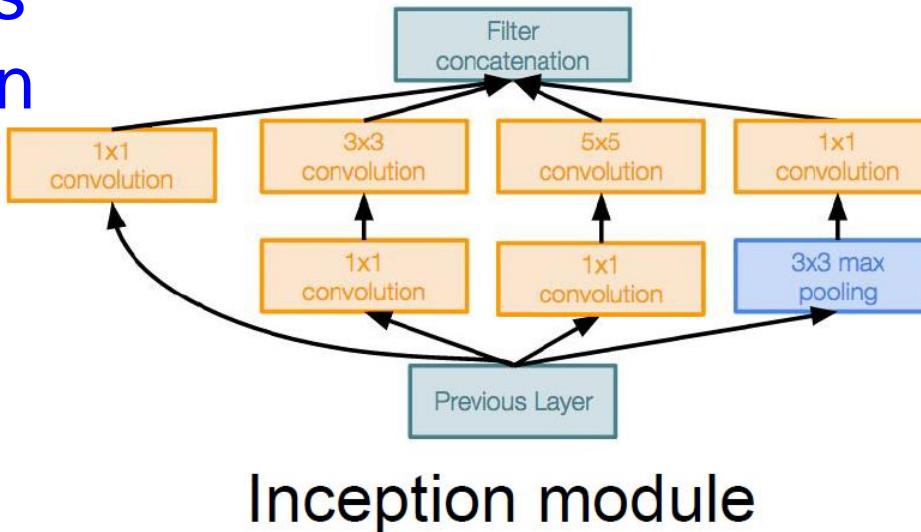
# Case Study: GoogLeNet [Szegedy et al., 2014]



# Case Study: GoogLeNet

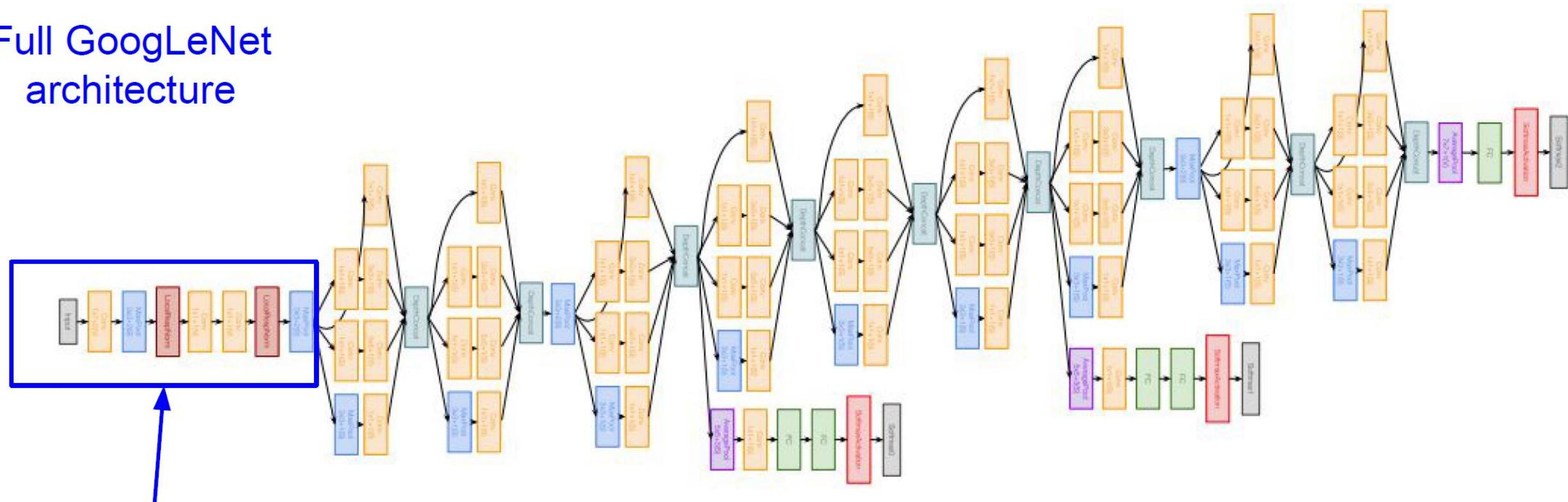
[Szegedy et al., 2014]

Stack Inception modules  
with dimension reduction  
on top of each other



# Case Study: GoogLeNet [Szegedy et al., 2014]

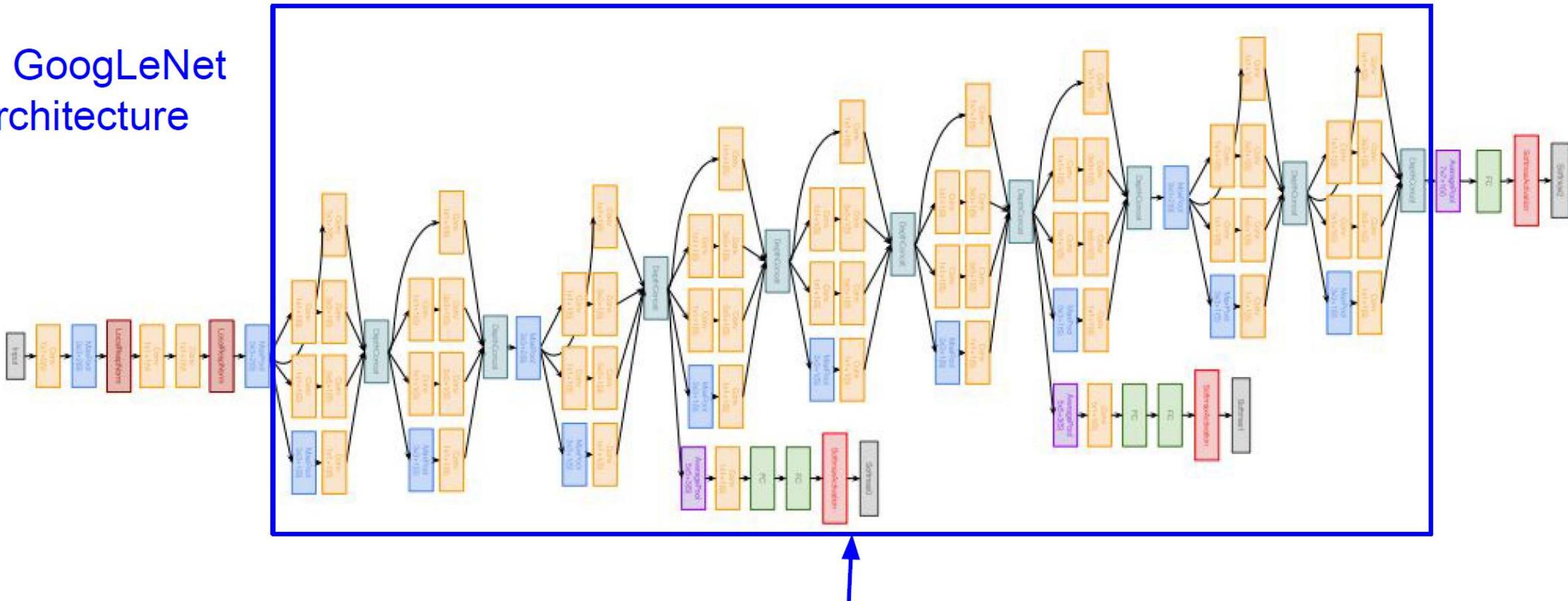
Full GoogLeNet  
architecture



Stem Network:  
Conv-Pool-  
2x Conv-Pool

# Case Study: GoogLeNet [Szegedy et al., 2014]

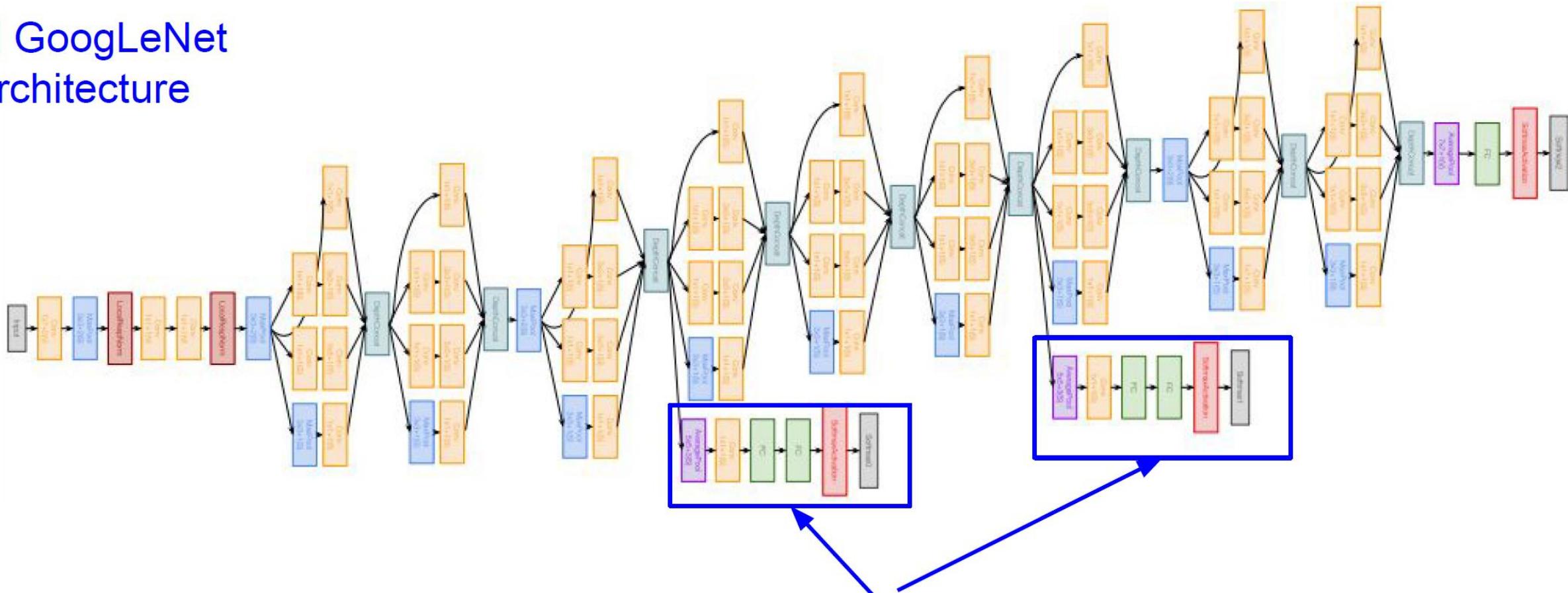
Full GoogLeNet architecture



Stacked Inception  
Modules

# Case Study: GoogLeNet [Szegedy et al., 2014]

Full GoogLeNet  
architecture

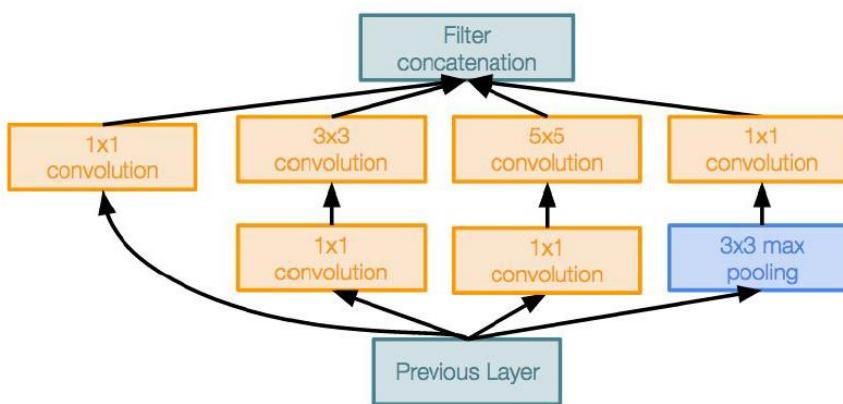


Auxiliary classification outputs to inject additional gradient at lower layers  
(AvgPool-1x1Conv-FC-FC-Softmax)

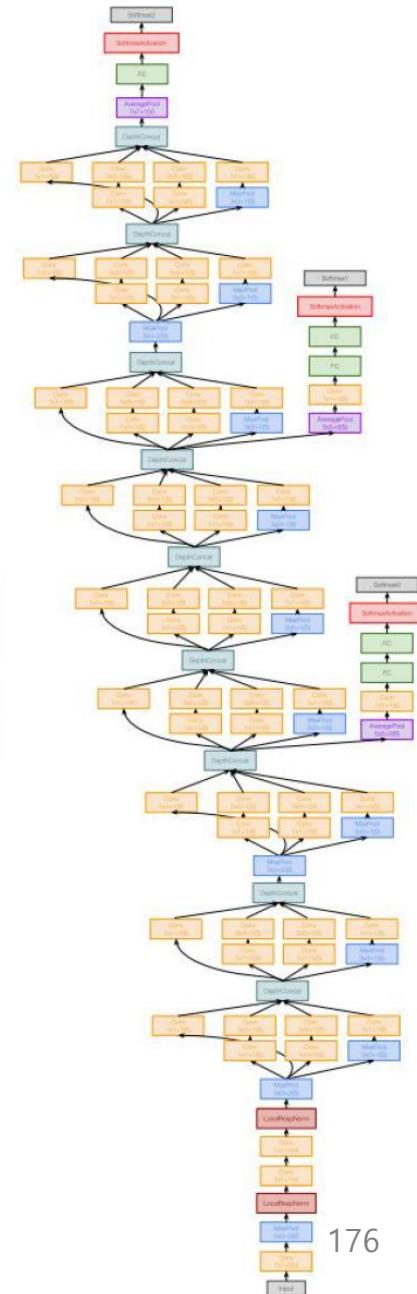
# Case Study: GoogLeNet [Szegedy et al., 2014]

Deeper networks, with computational Efficiency

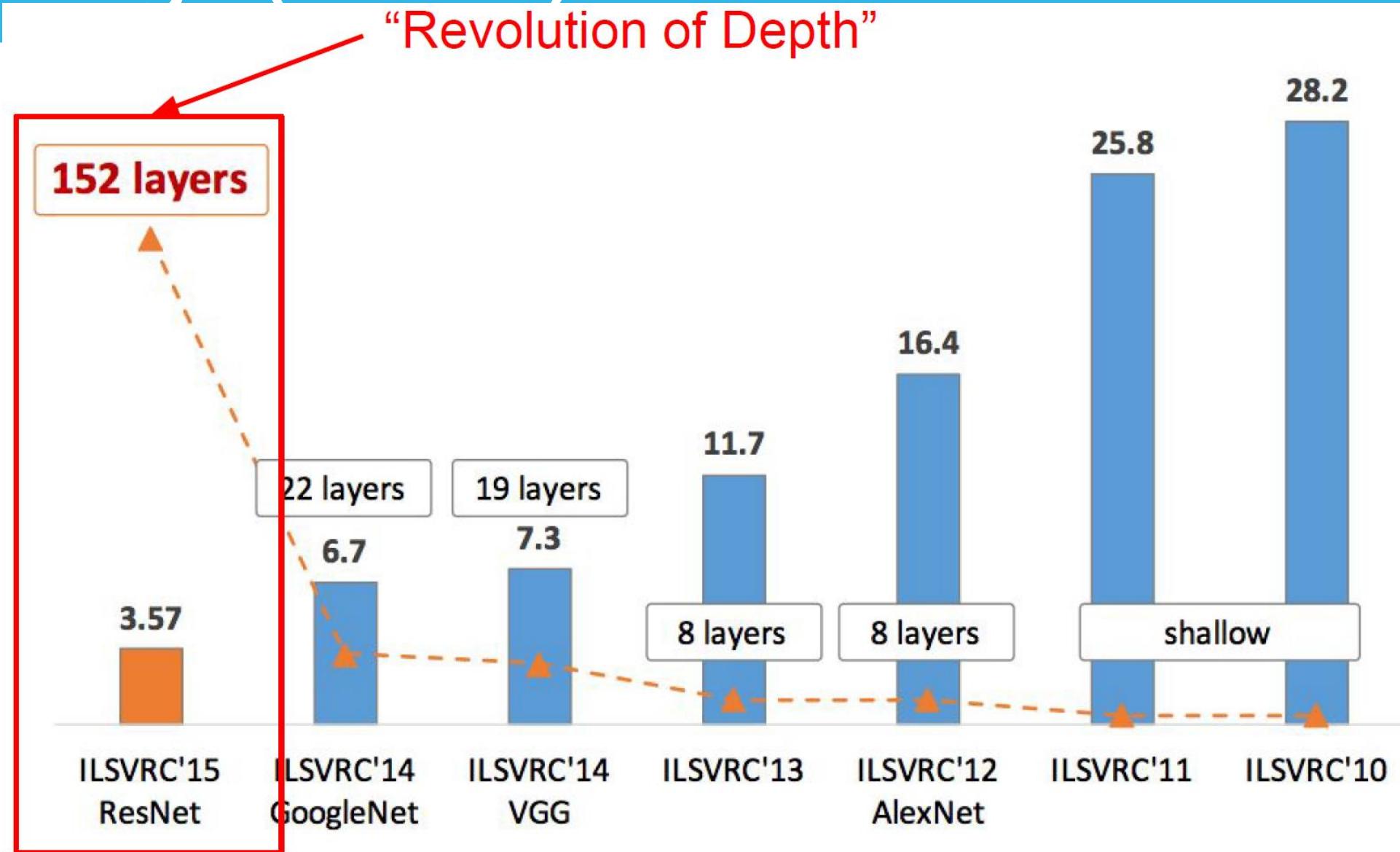
- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters!  
12x less than AlexNet
- ILSVRC’14 classification winner  
(6.7% top 5 error)



Inception module



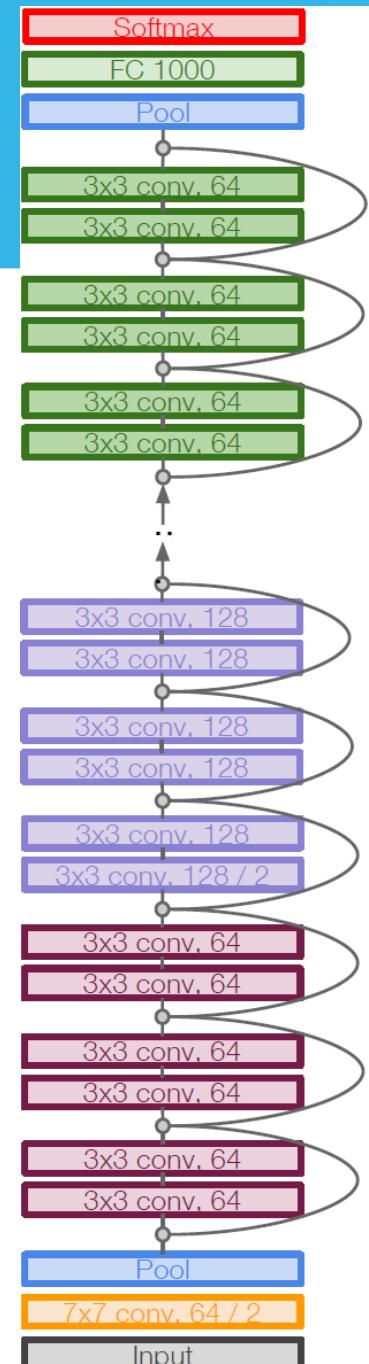
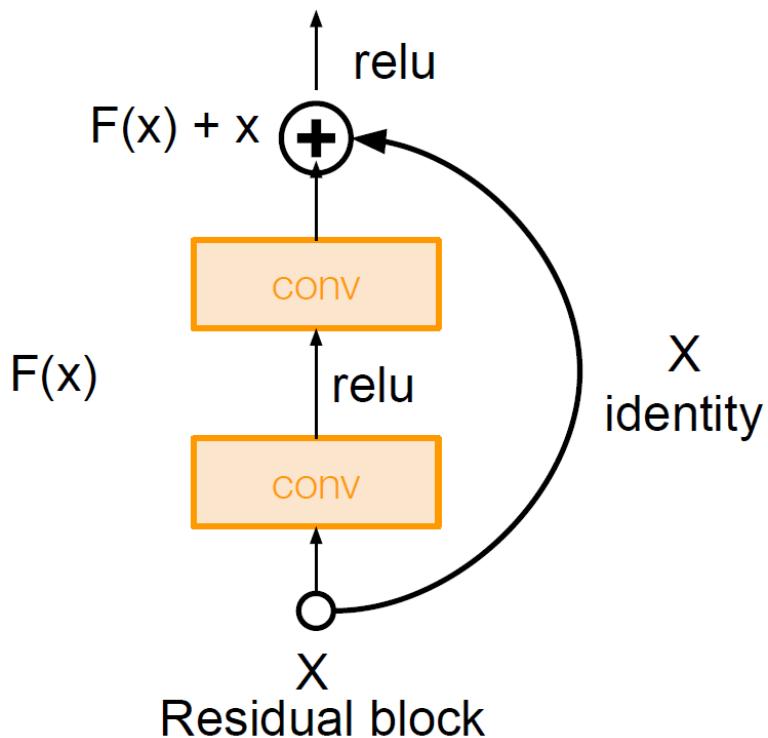
# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



# Case Study: ResNet [He et al., 2015]

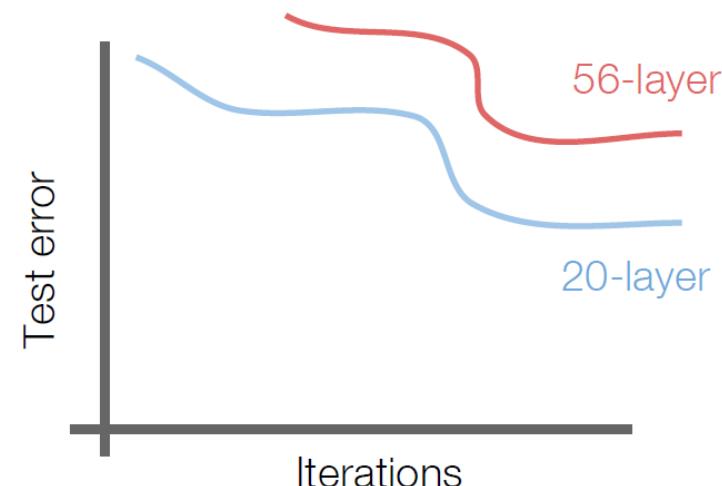
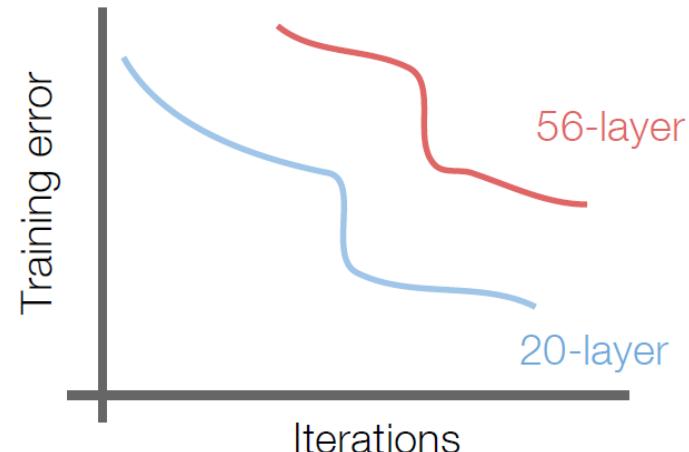
Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!



# Case Study: ResNet [He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



56-layer model performs worse on both training and test error  
-> The deeper model performs worse, but it's not caused by overfitting!

# Case Study: ResNet [He et al., 2015]

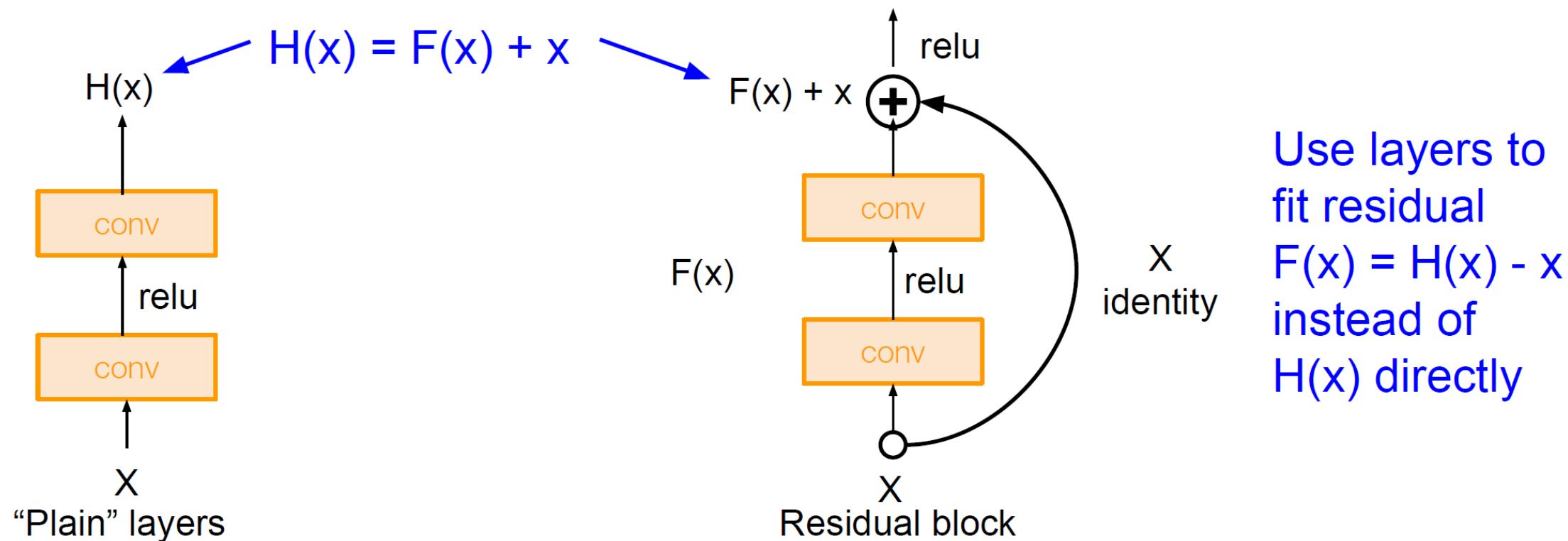
Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

The deeper model should be able to perform at least as well as the shallower model.

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

# Case Study: ResNet [He et al., 2015]

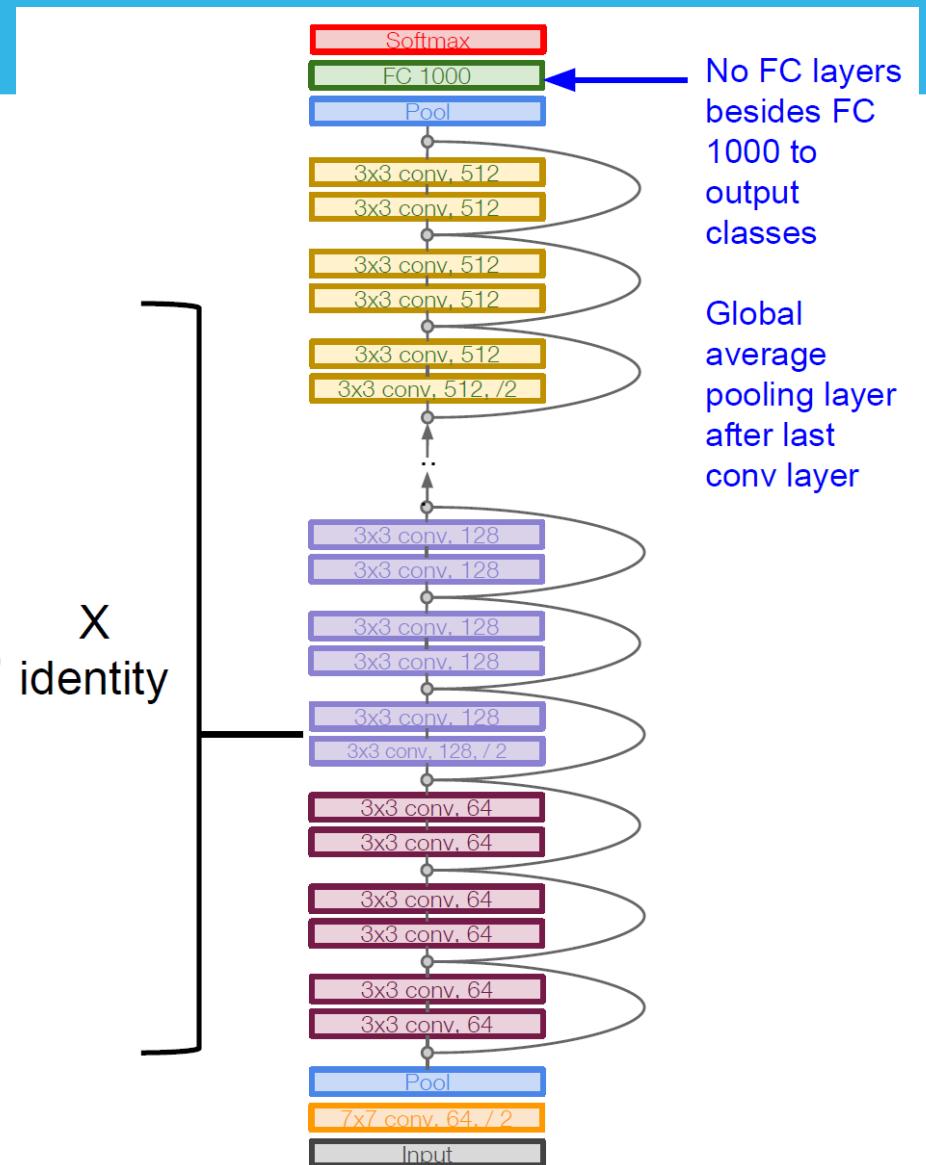
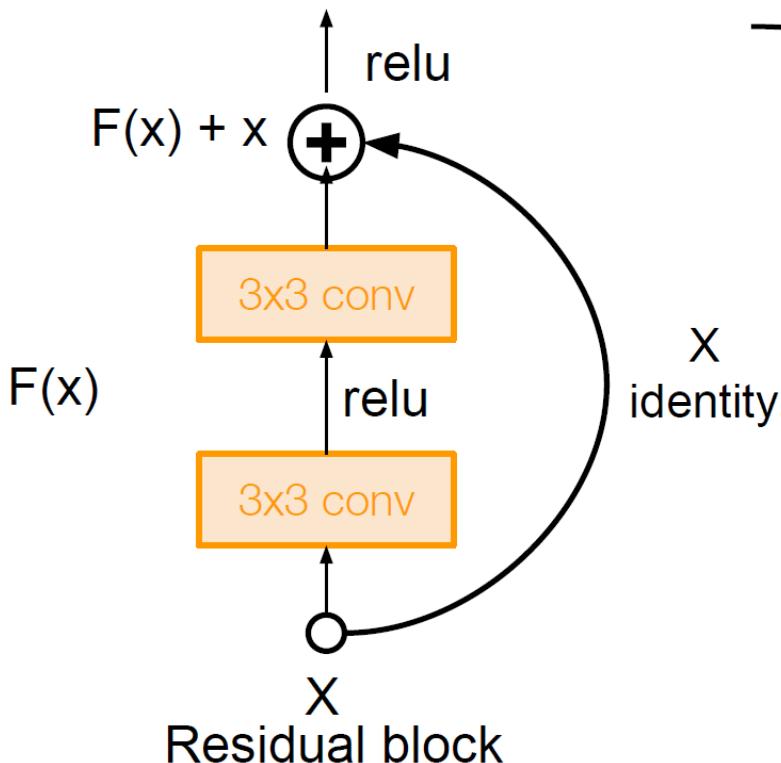
Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



# Case Study: ResNet [He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)



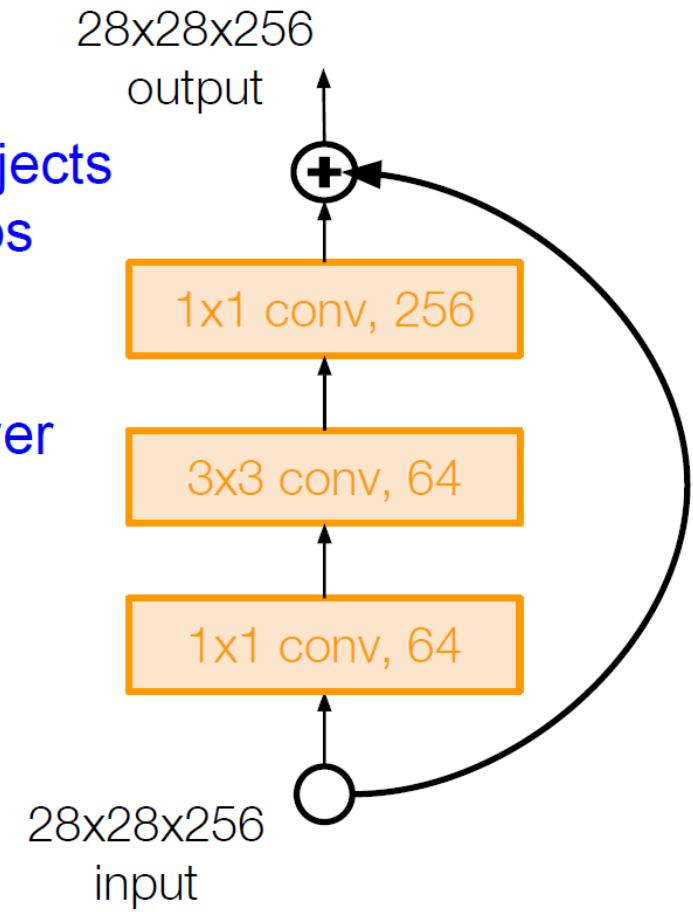
# Case Study: ResNet [He et al., 2015]

For deeper networks  
(ResNet-50+), use “bottleneck”  
layer to improve efficiency  
(similar to GoogLeNet)

1x1 conv, 256 filters projects  
back to 256 feature maps  
(28x28x256)

3x3 conv operates over  
only 64 feature maps

1x1 conv, 64 filters  
to project to  
28x28x64



# Case Study: ResNet [He et al., 2015]

## Experimental Results

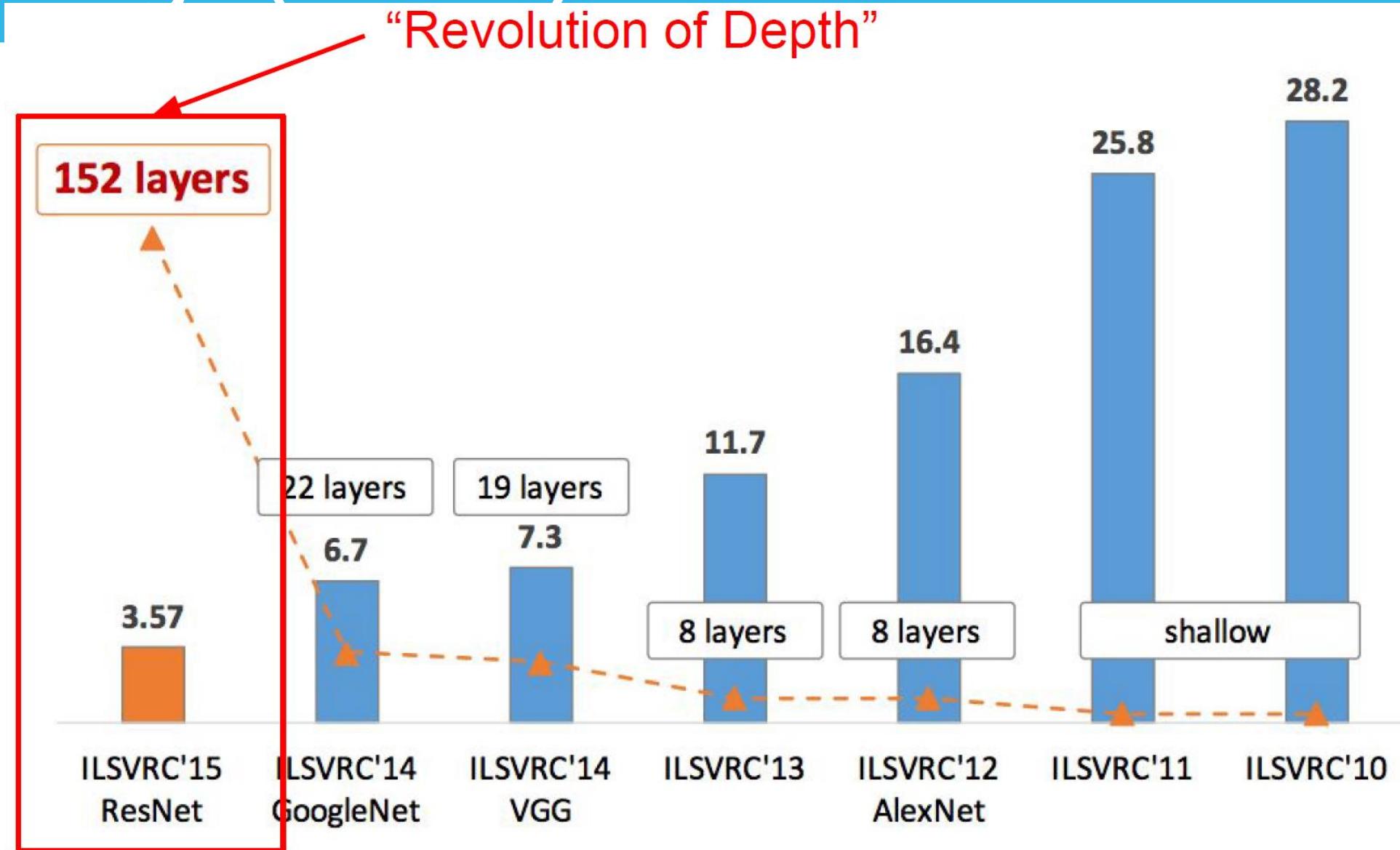
- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lower training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

### MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: “Ultra-deep” (quote Yann) **152-layer** nets
  - ImageNet Detection: **16%** better than 2nd
  - ImageNet Localization: **27%** better than 2nd
  - COCO Detection: **11%** better than 2nd
  - COCO Segmentation: **12%** better than 2nd

ILSVRC 2015 classification winner (3.6% top 5 error) -- better than “human performance”! (Russakovsky 2014)

# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

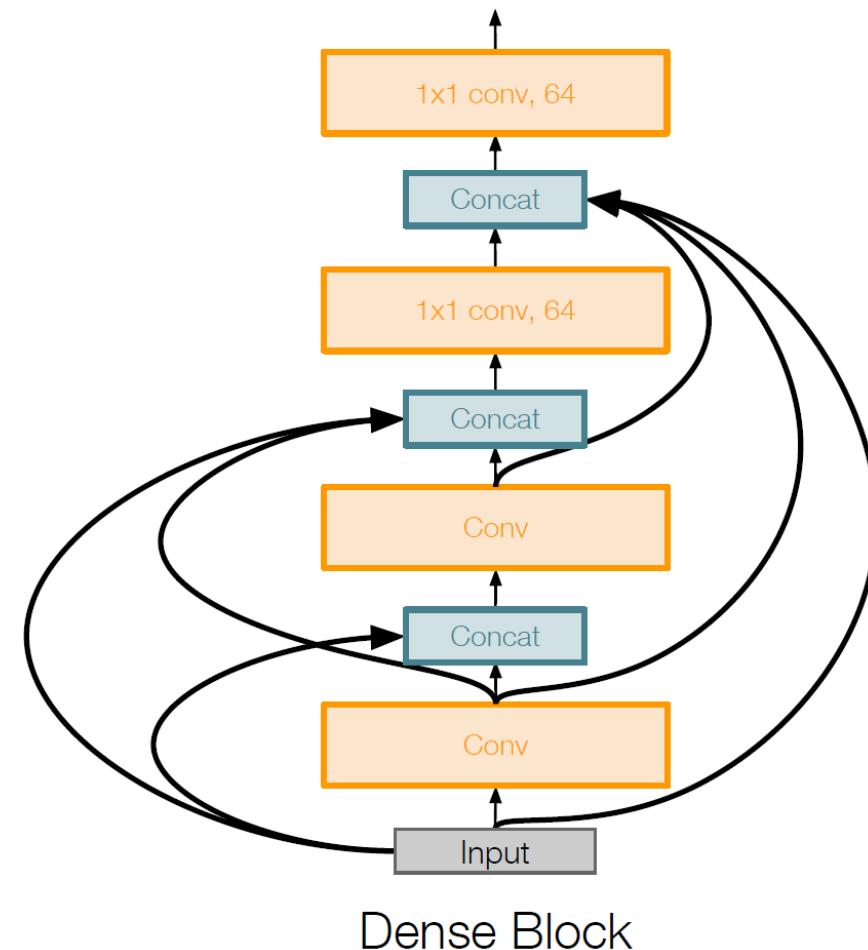


# Beyond ResNets

## Densely Connected Convolutional Networks

[Huang et al. 2017]

- Dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse



Softmax
FC
Pool
Dense Block 3
Conv
Pool
Conv
Dense Block 2
Conv
Pool
Conv
Dense Block 1
Conv
Input

Dense Block

# Summary: CNN Architectures

- VGG, GoogLeNet, ResNet all in wide use, available in model zoos
- ResNet current best default
- Trend towards extremely deep networks
- Significant research centers around design of layer / skip connections and improving gradient flow
- Even more recent trend towards examining necessity of depth vs. width and residual connections

# References

## Convolutional Neural Network

Bumsoo Kim, Computer Vision & Deep Learning pdf

[https://brohrer.github.io/how\\_convolutional\\_neural\\_networks\\_work.html](https://brohrer.github.io/how_convolutional_neural_networks_work.html)

<http://cs231n.stanford.edu/syllabus.html>

## Convolutional Neural Network for Text Classification

<http://cs224d.stanford.edu/syllabus.html>

<http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/>

<https://arxiv.org/abs/1510.03820>

<https://arxiv.org/abs/1408.5882>



# Q&A

# THANK YOU!

