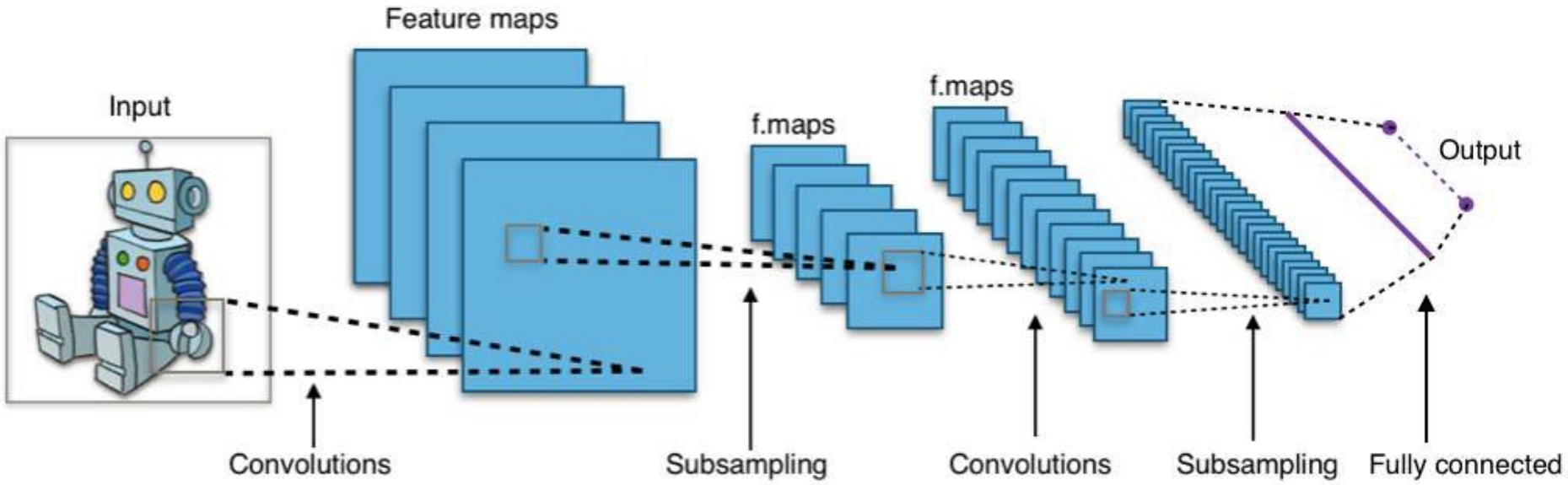
Introduction to Computer Vision

Kaveh Fathian
Assistant Professor
Computer Science Department
Colorado School of Mines

Lecture 20

Convolutional Neural Networks

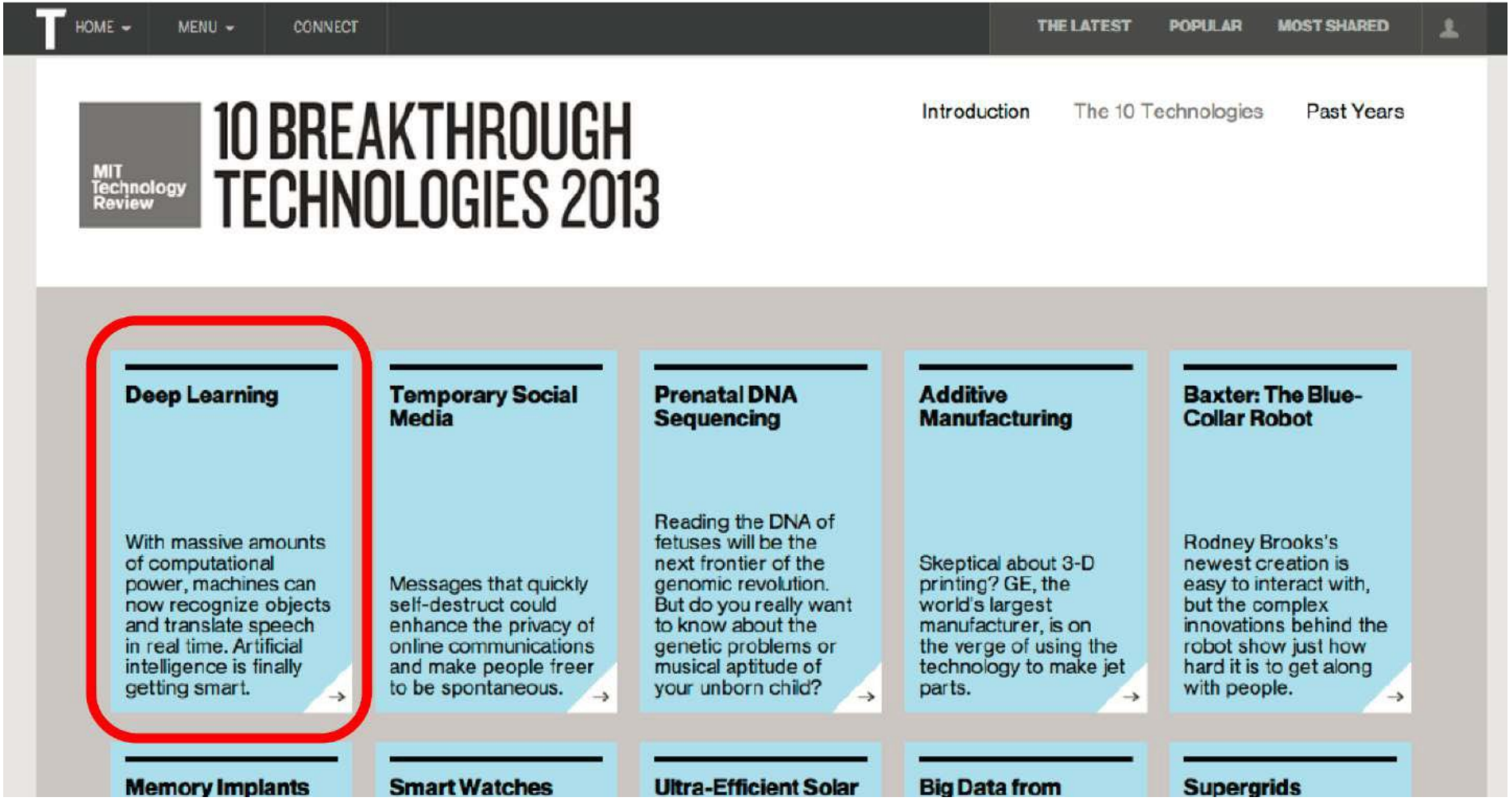


Aside: “CNN” vs “ConvNet”

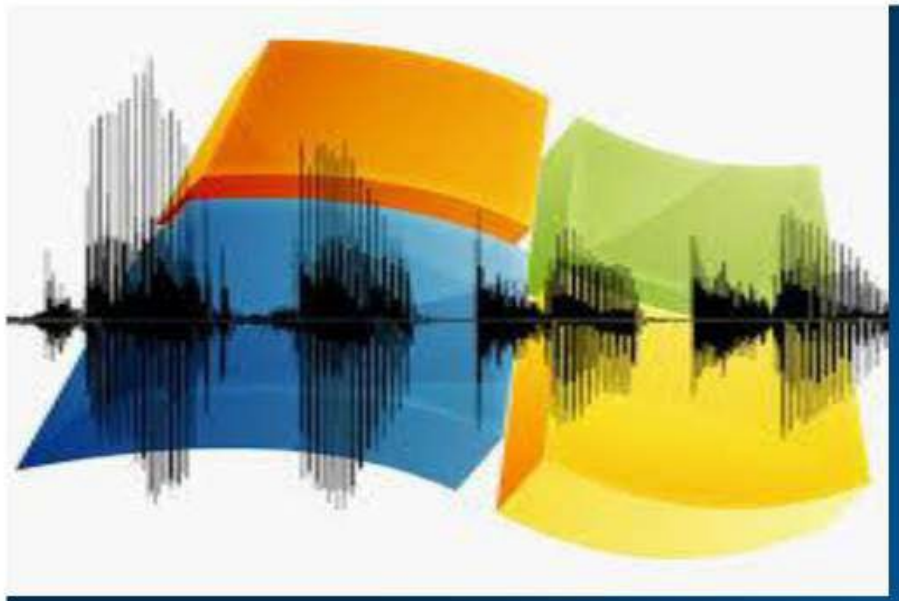
Note:

- There are many papers that use either phrase
- “ConvNet” is the preferred term by some, since “CNN” clashes with other things

Motivation


HOME
MENU
CONNECT
THE LATEST
POPULAR
MOST SHARED
MIT Technology Review
10 BREAKTHROUGH TECHNOLOGIES 2013
Introduction
The 10 Technologies
Past Years
Deep Learning
With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart.
Temporary Social Media
Messages that quickly self-destruct could enhance the privacy of online communications and make people freer to be spontaneous.
Prenatal DNA Sequencing
Reading the DNA of fetuses will be the next frontier of the genomic revolution. But do you really want to know about the genetic problems or musical aptitude of your unborn child?
Additive Manufacturing
Skeptical about 3-D printing? GE, the world's largest manufacturer, is on the verge of using the technology to make jet parts.
Baxter: The Blue-Collar Robot
Rodney Brooks's newest creation is easy to interact with, but the complex innovations behind the robot show just how hard it is to get along with people.
Memory Implants
Smart Watches
Ultra-Efficient Solar
Big Data from
Supergrids

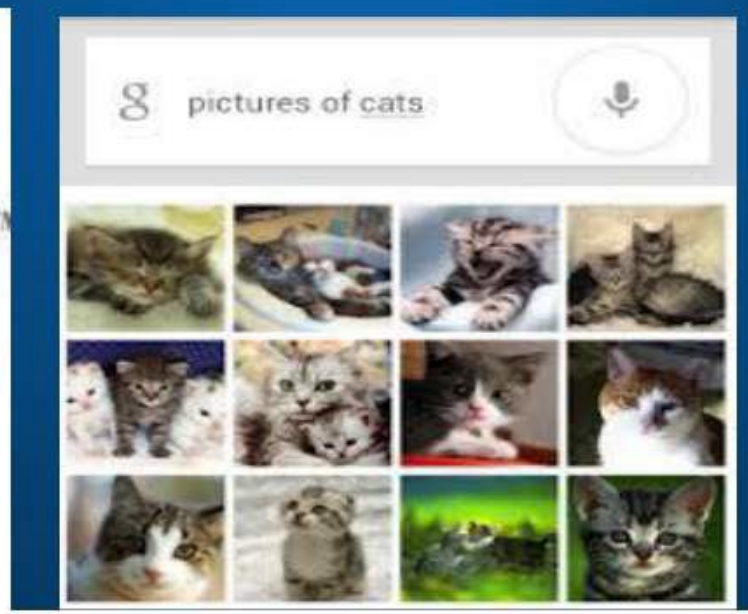
Products




WEB
IMAGES
VIDEOS
MAPS
NEWS
MORE
bing
golden retriever
Size
Color
Type
Layout
People
Date
License
Funny
Golden Retriever
Miniature
Golden Retriever
Baby
Golden Retriever
White
Golden Retriever


PhotoWonder
A magic photo beautifier, cool and easy to use!
Baidu 百度


Translate
Google


pictures of cats


flickr

CNNs in 2012: “SuperVision” (aka “AlexNet”)

“**AlexNet**” — Won the ILSVRC2012 Challenge

Major breakthrough: 15.3% Top-5 error on ILSVRC2012
(Next best: 25.7%)

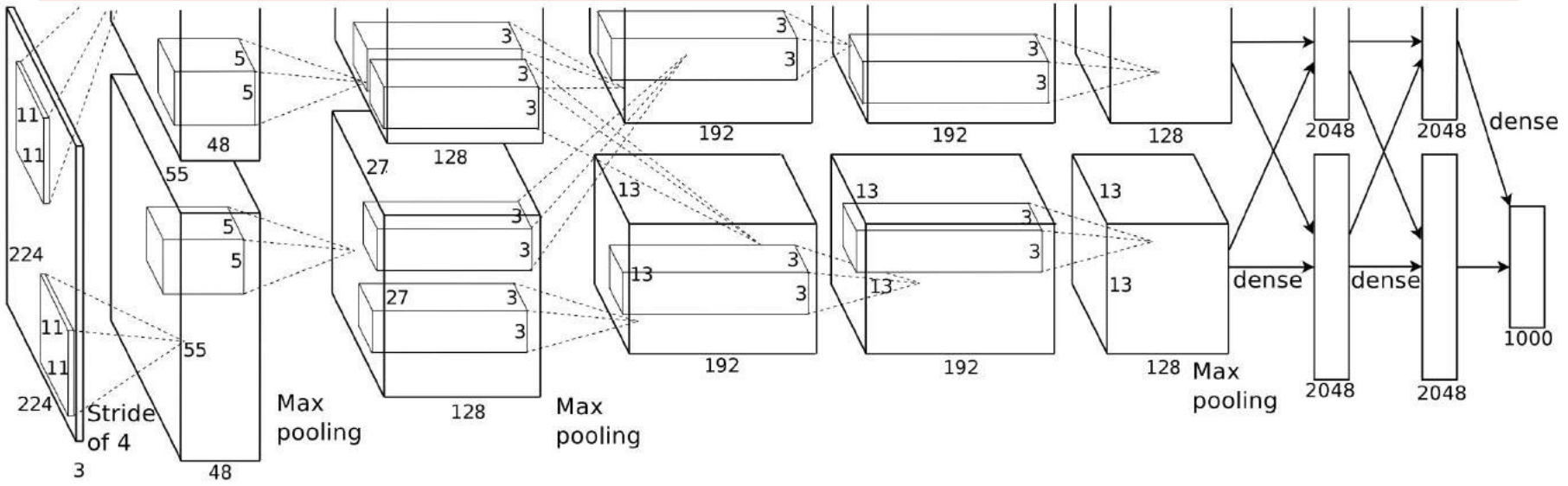
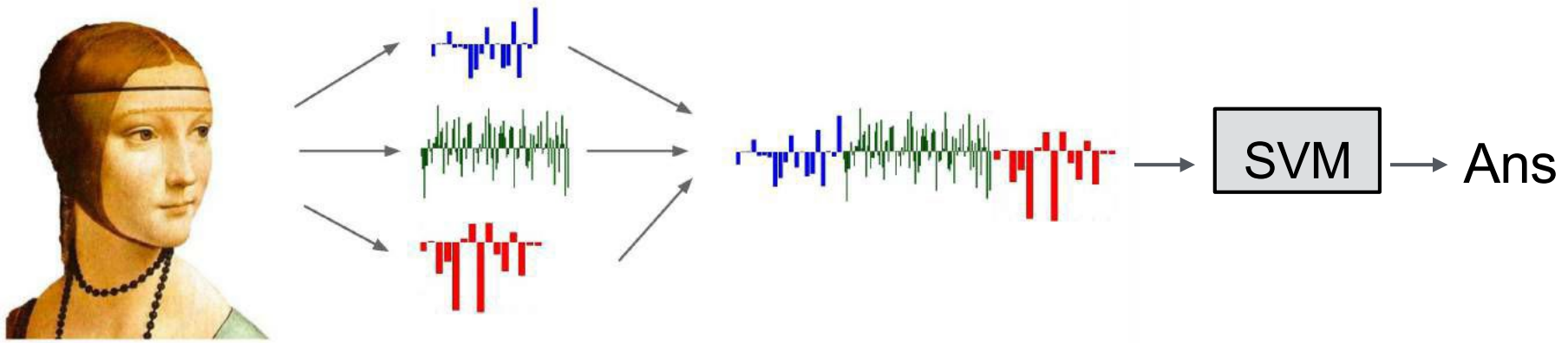


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input is 150,528-dimensional, and the number of neurons in the network’s remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

[Krizhevsky, Sutskever, Hinton. NIPS 2012]

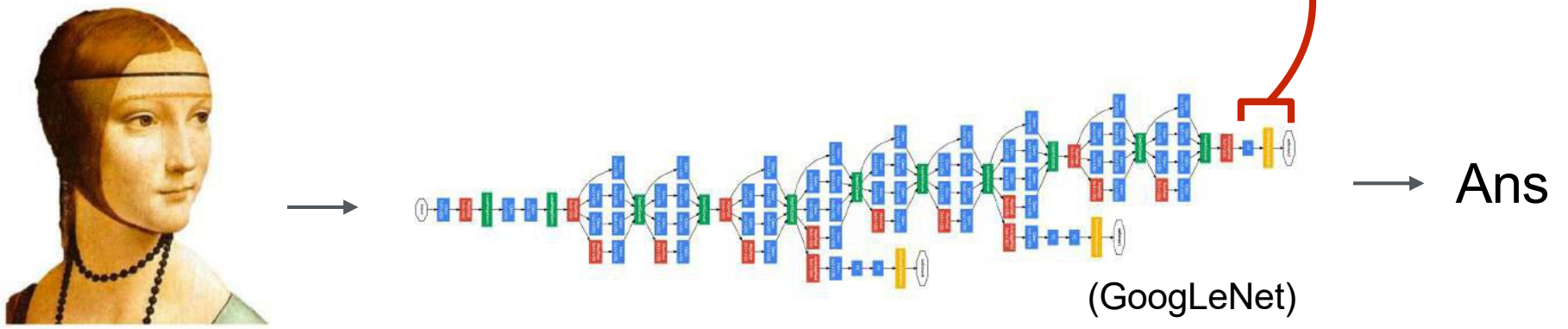
Recap: Before Deep Learning



Input Pixels | *Extract Features* | *Concatenate into a vector **x*** | *Linear Classifier*

Figure: Karpathy 2016

The last layer of (most) CNNs are linear classifiers



Input Pixels

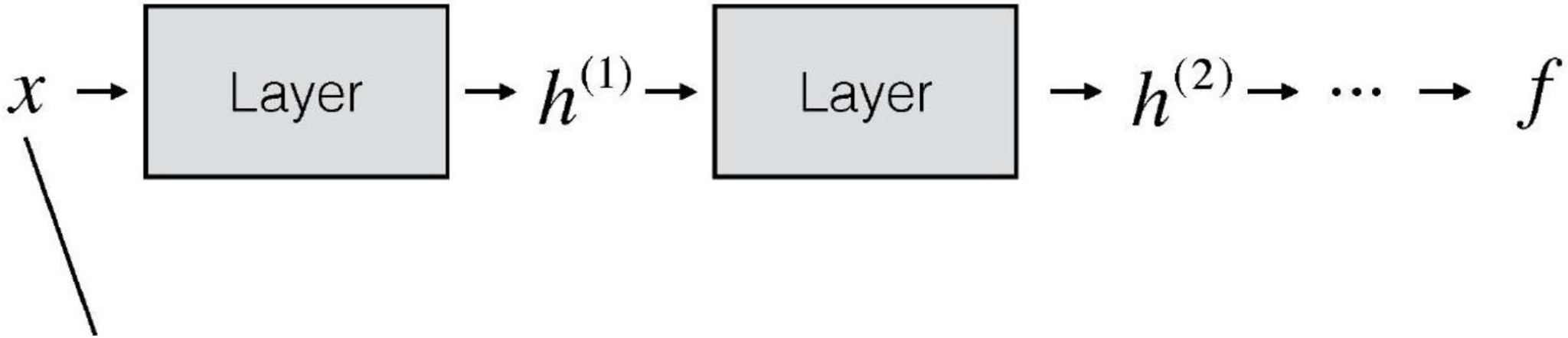
Perform everything with a big neural network, trained end-to-end

Key: perform enough processing so that by the time you get to the end of the network, the classes are linearly separable

CNNs

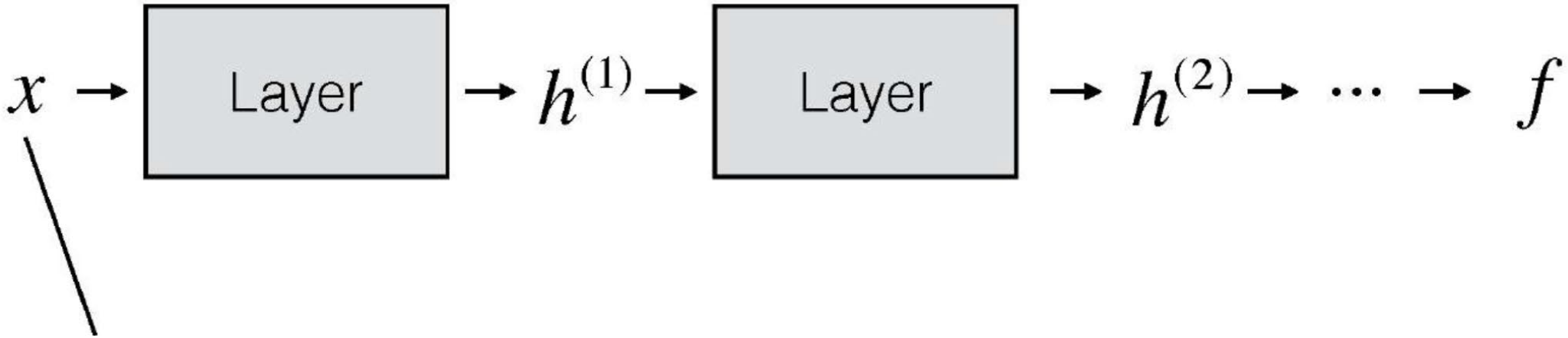
They're just neural networks with 3D activations and weight sharing

What shape should the activations have?



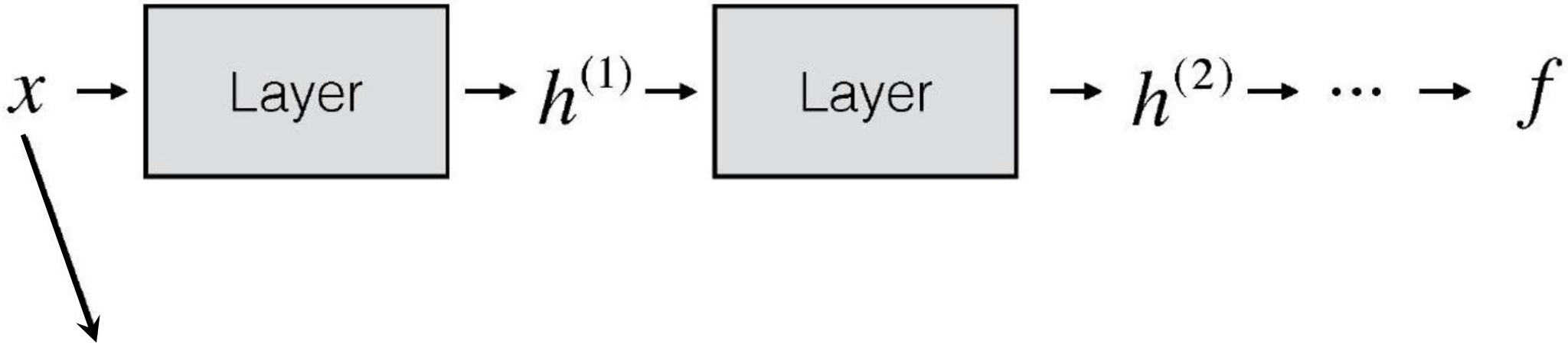
- The input is an image, which is 3D (RGB channel, height, width)

What shape should the activations have?



- The input is an image, which is 3D (RGB channel, height, width)
- We could flatten it to a 1D vector, but then we lose structure

What shape should the activations have?



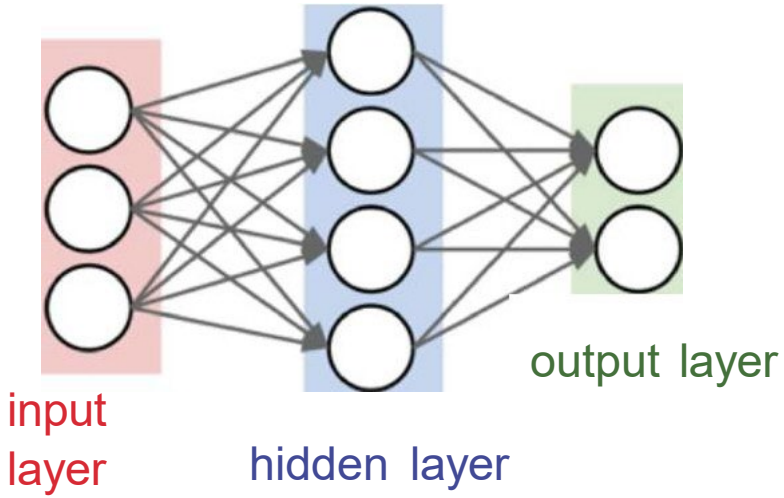
- The input is an image, which is 3D (RGB channel, height, width)
- We could flatten it to a 1D vector, but then we lose structure
- What about keeping everything in 3D?

CNNs

They're just neural networks with
3D activations and weight sharing

3D Activations

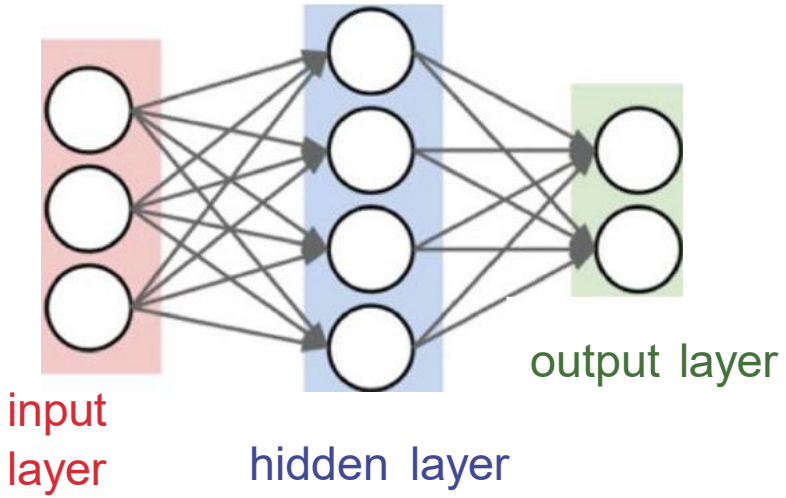
before:



(1D vectors)

3D Activations

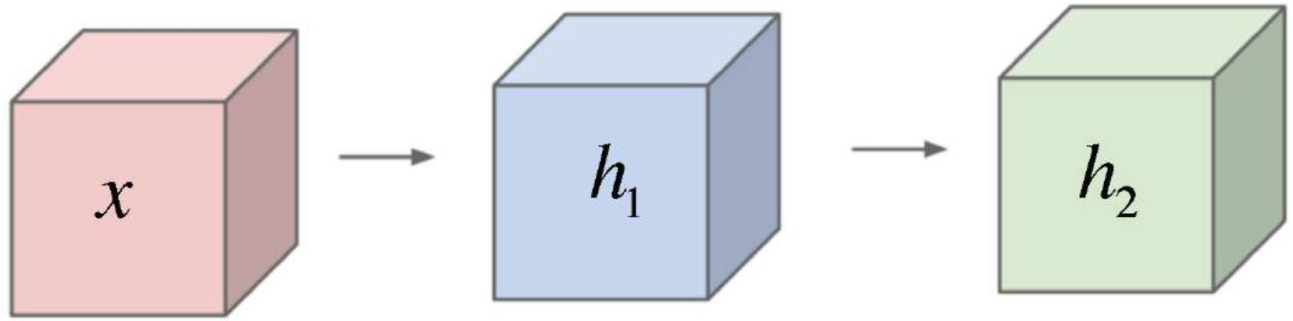
before:



(1D vectors)

---

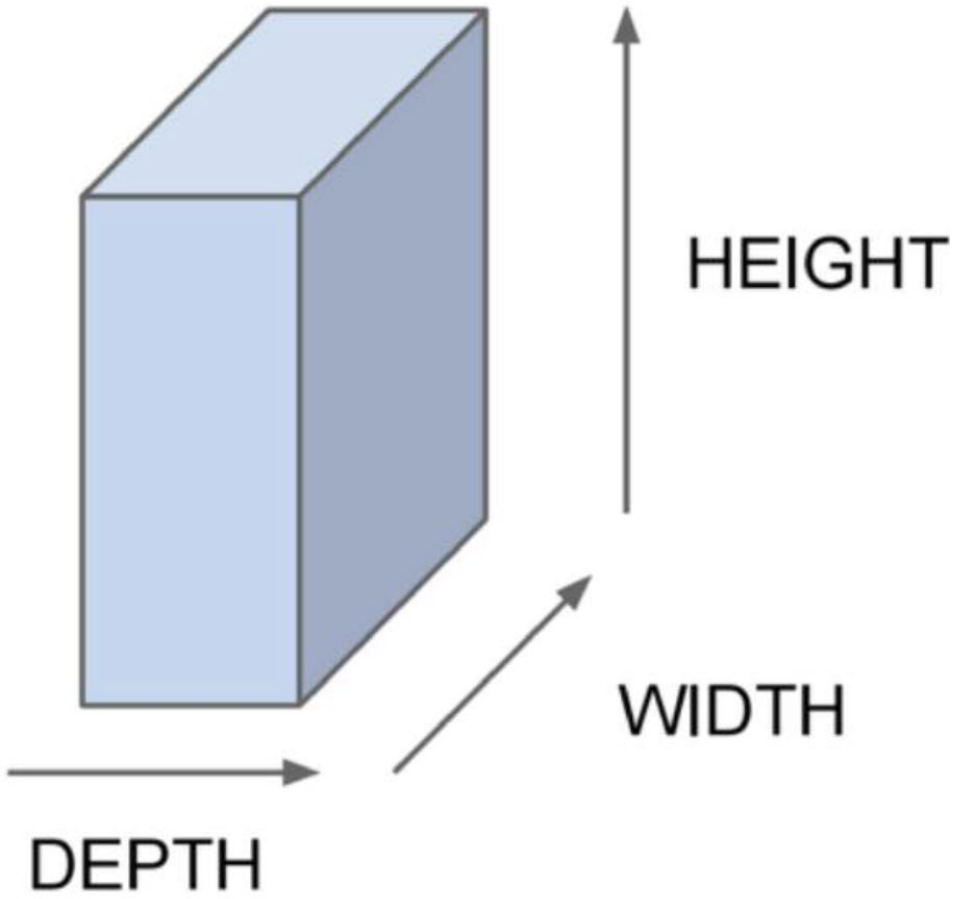
now:



(3D arrays)

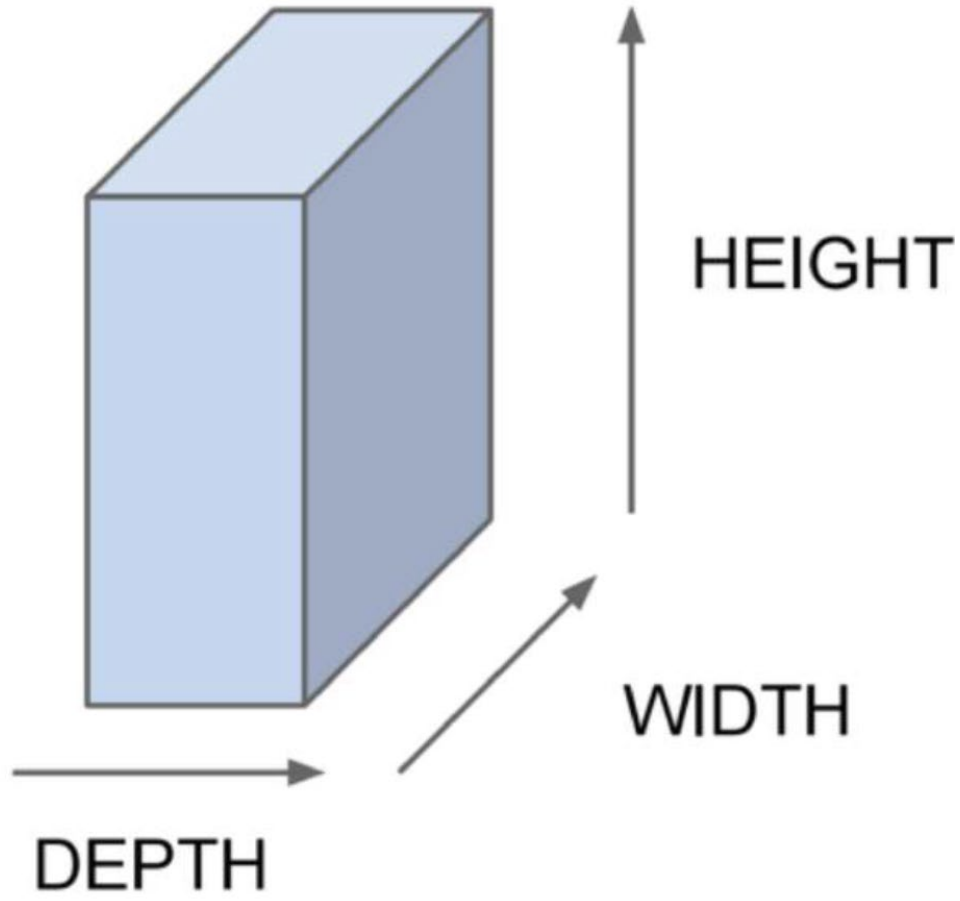
3D Activations

All Neural Net activations arranged in **3 dimensions:**



3D Activations

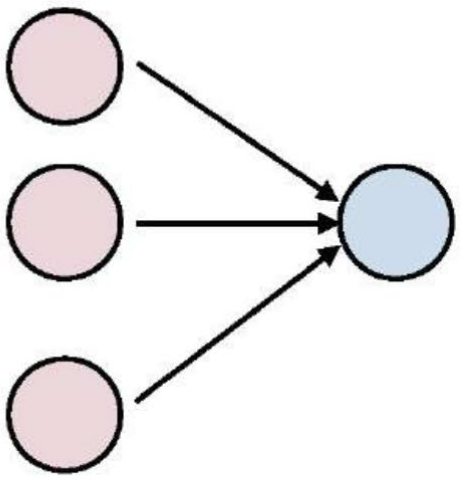
All Neural Net activations arranged in **3 dimensions:**



For example, a CIFAR-10 image is a 3x32x32 volume (3 depth - RGB channels, 32 height, 32 width)

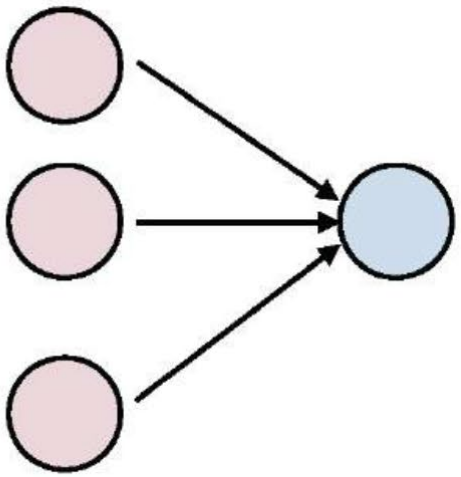
3D Activations

1D Activations:

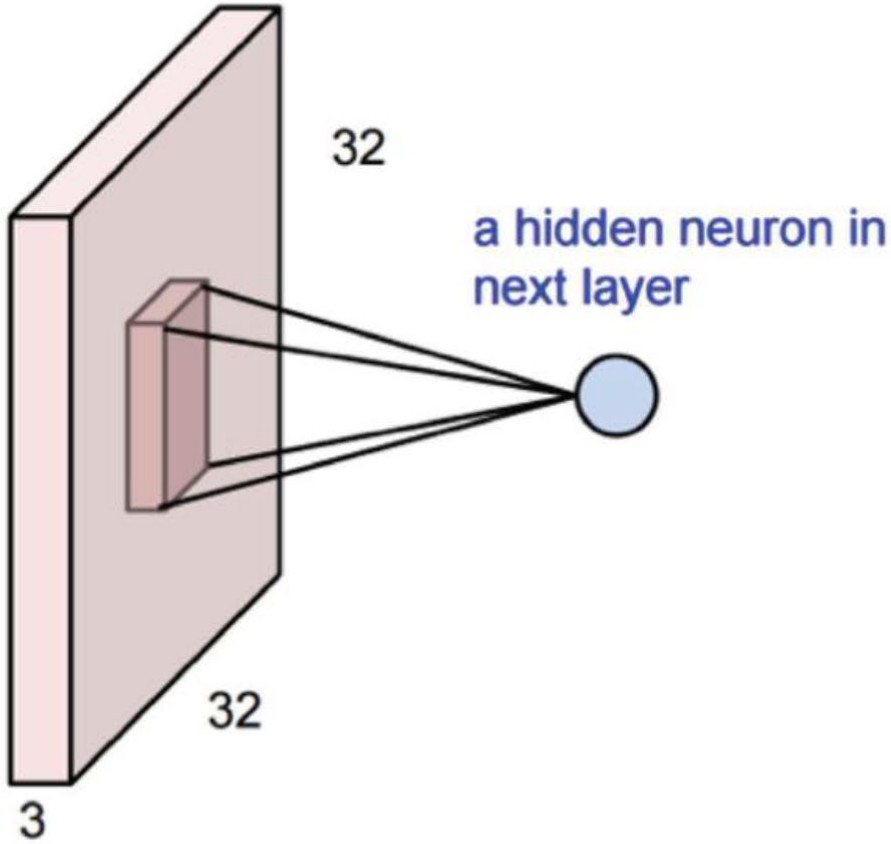


3D Activations

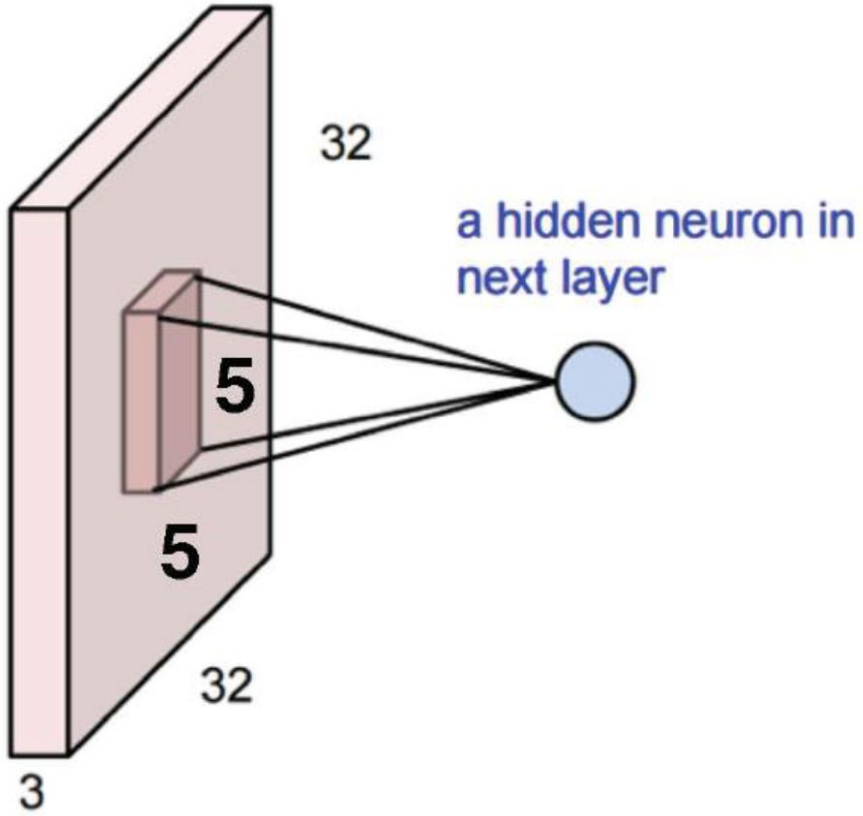
1D Activations:



3D Activations:

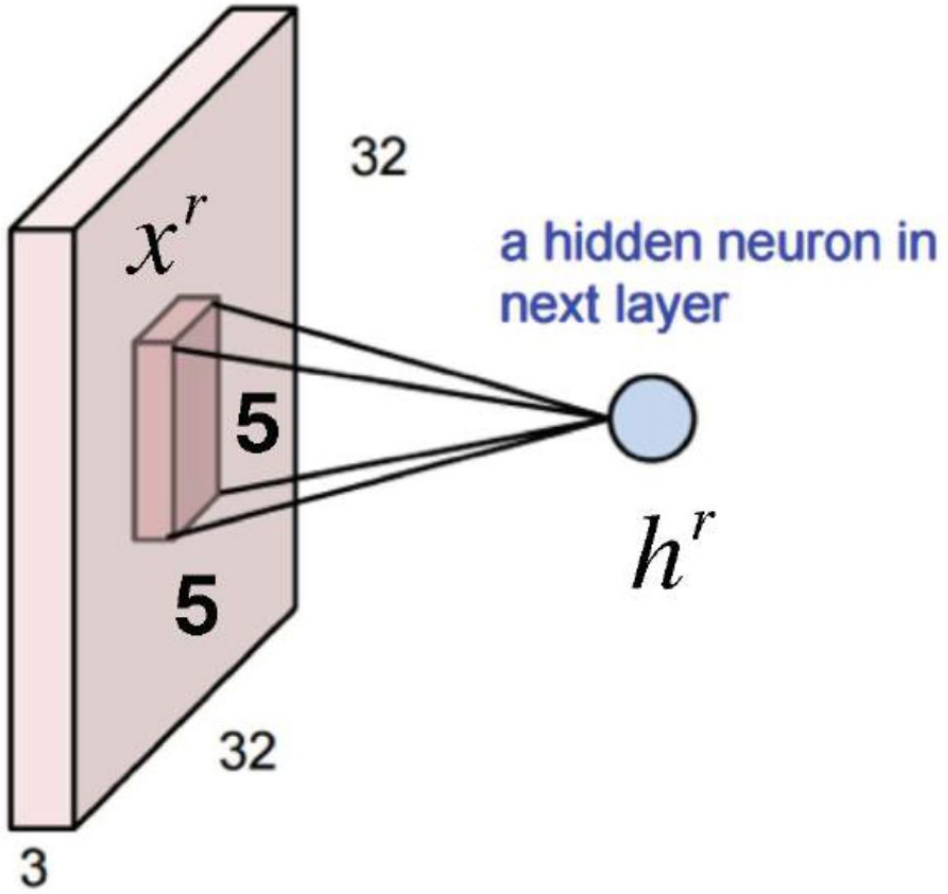


3D Activations



- The input is 3x32x32
- This neuron depends on a 3x5x5 chunk of the input
- The neuron also has a 3x5x5 set of weights and a bias (scalar)

3D Activations



Example: consider the region of the input "x^r"

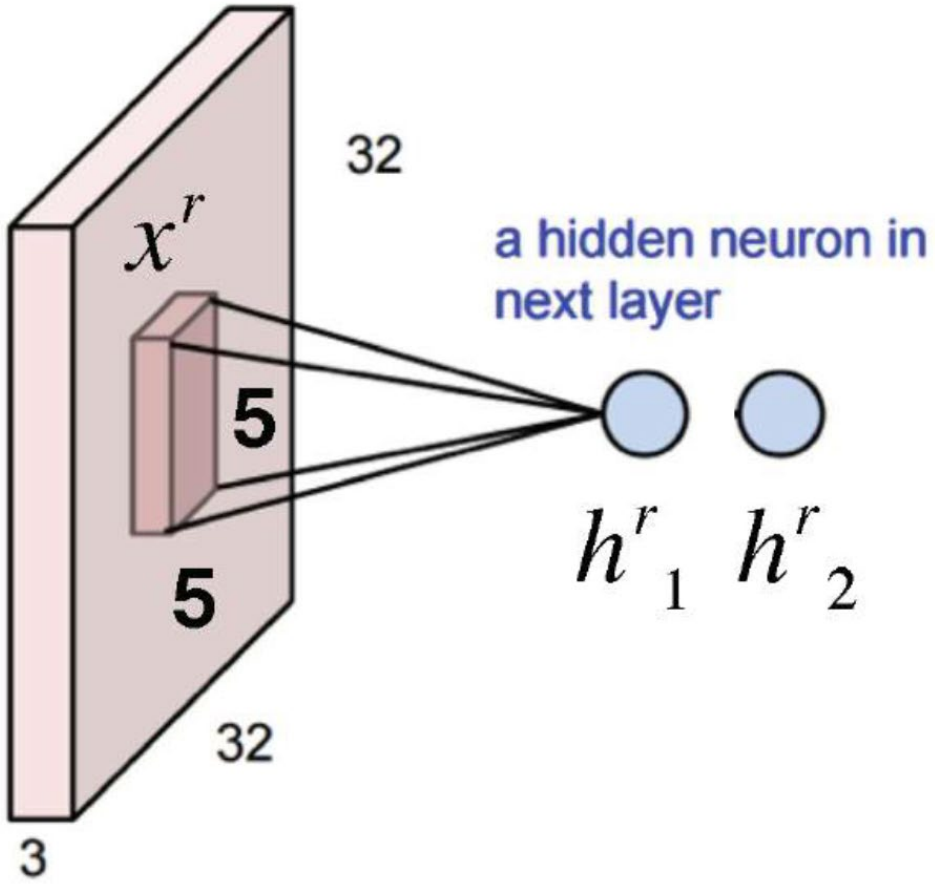
With output neuron h^r

Then the output is:

$$h^r = \sum_{ijk} x^r{}_{ijk} W_{ijk} + b$$

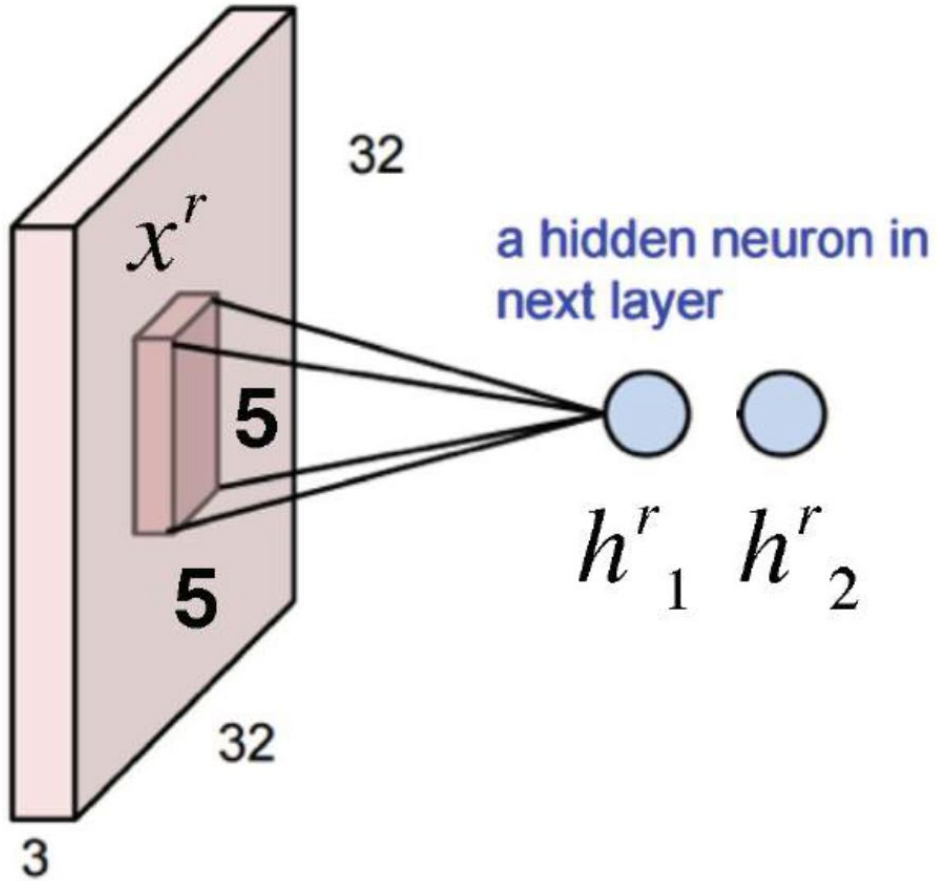
Sum over 3 axes

3D Activations



With **2 output** neurons

3D Activations

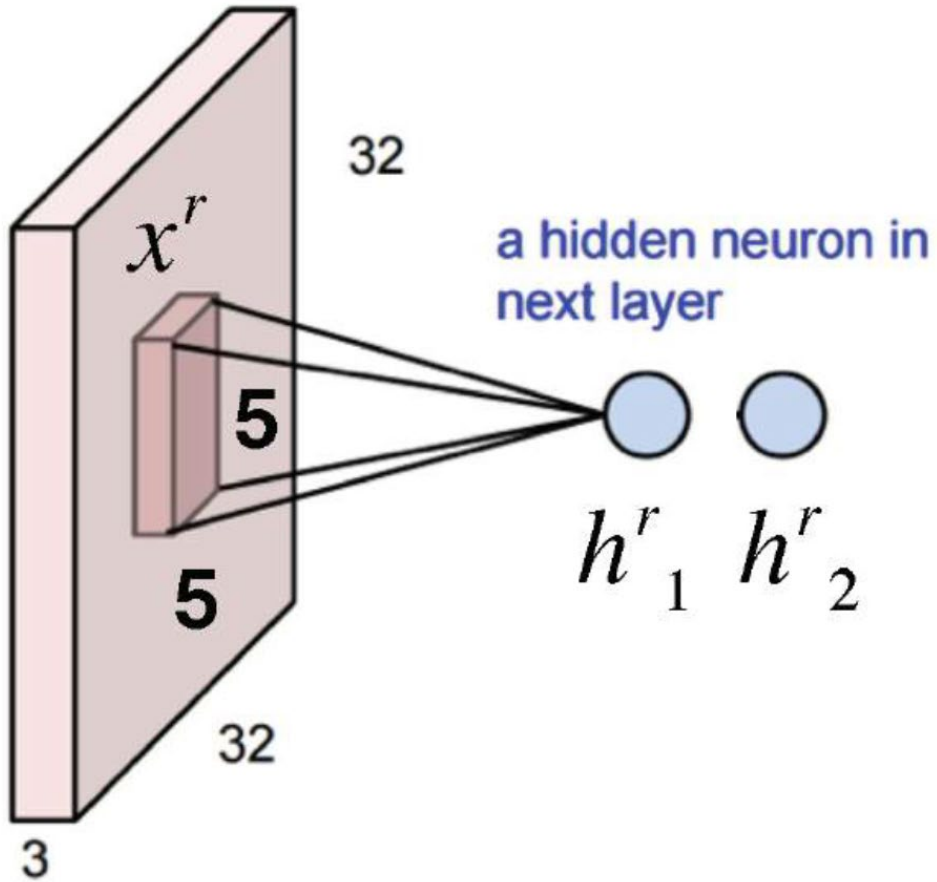


With **2 output** neurons

$$h^r{}_1 = \sum_{ijk} x^r{}_{ijk} W_{1ijk} + b_1$$

$$h^r{}_2 = \sum_{ijk} x^r{}_{ijk} W_{2ijk} + b_2$$

3D Activations

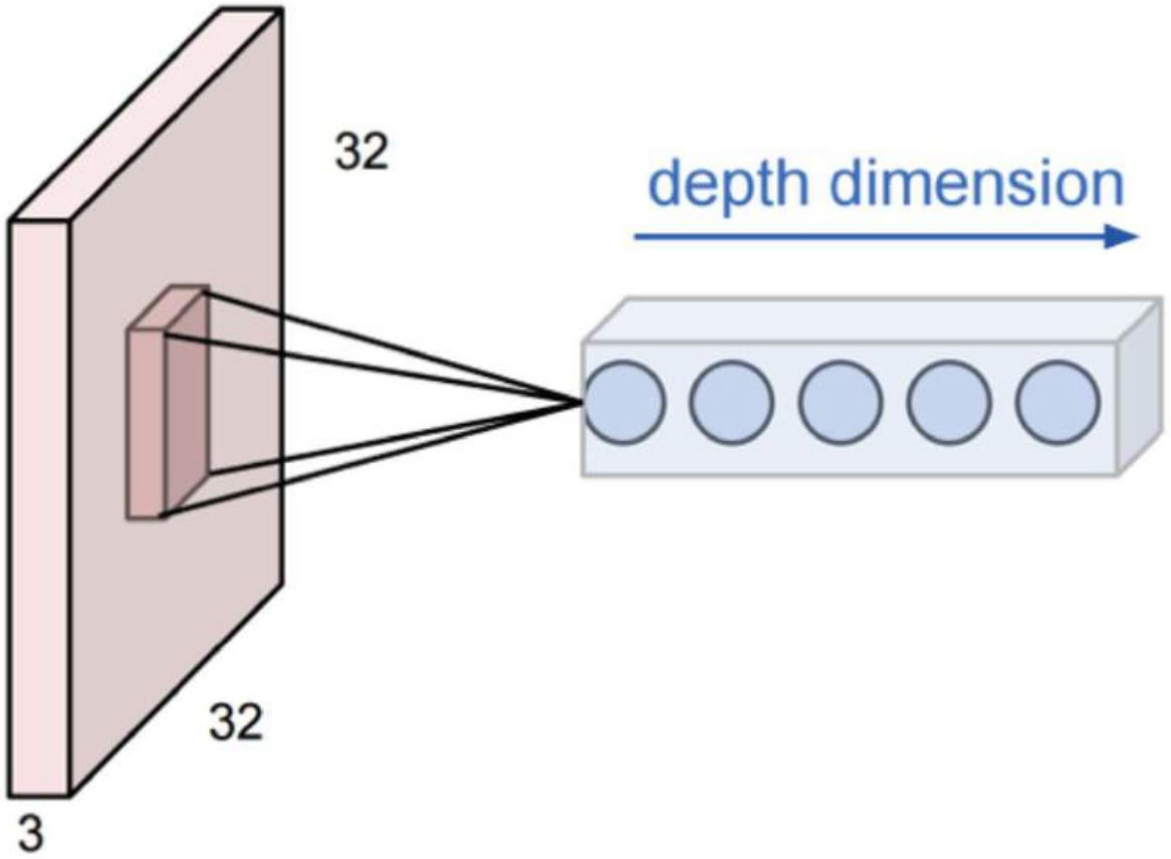


With **2 output** neurons

$$h^r_1 = \sum_{ijk} x^r_{ijk} W_{1ijk} + b_1$$

$$h^r_2 = \sum_{ijk} x^r_{ijk} W_{2ijk} + b_2$$

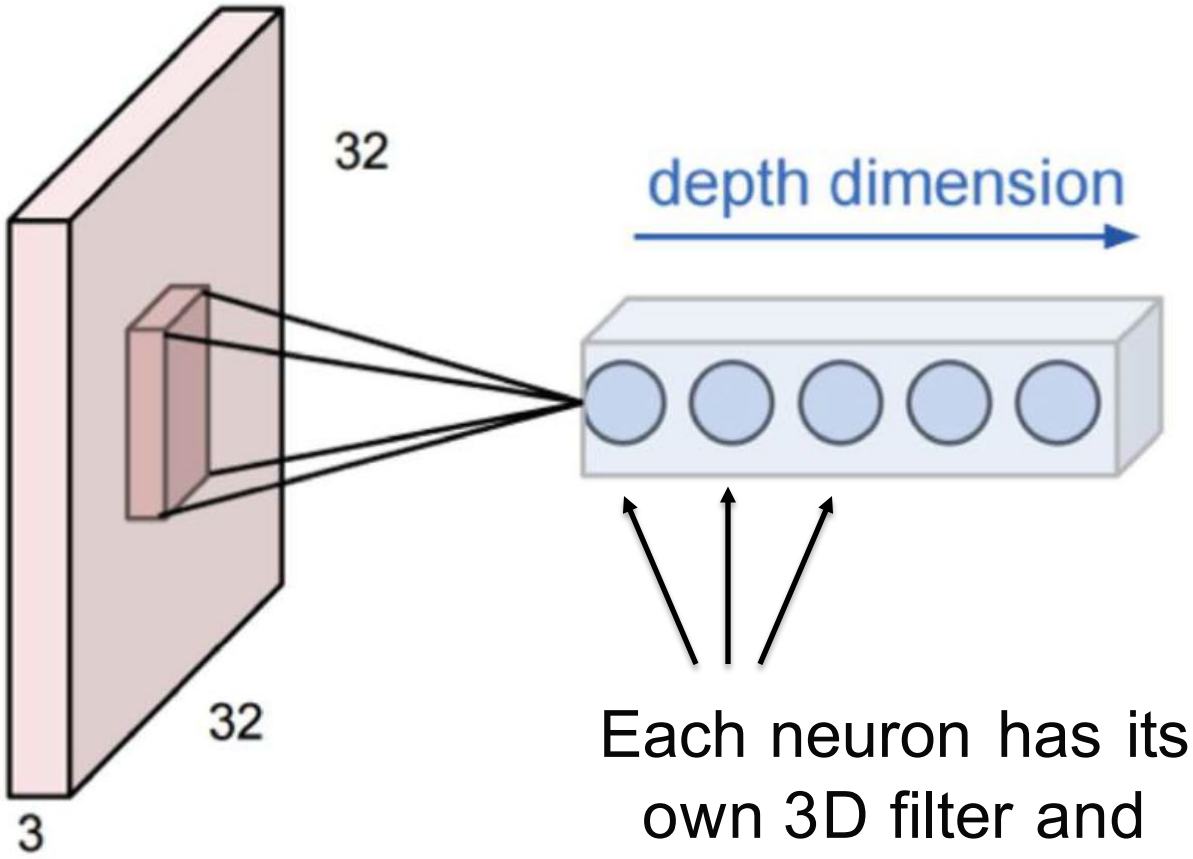
3D Activations



We can keep adding more outputs

These form a column in the output volume: [depth x 1 x 1]

3D Activations

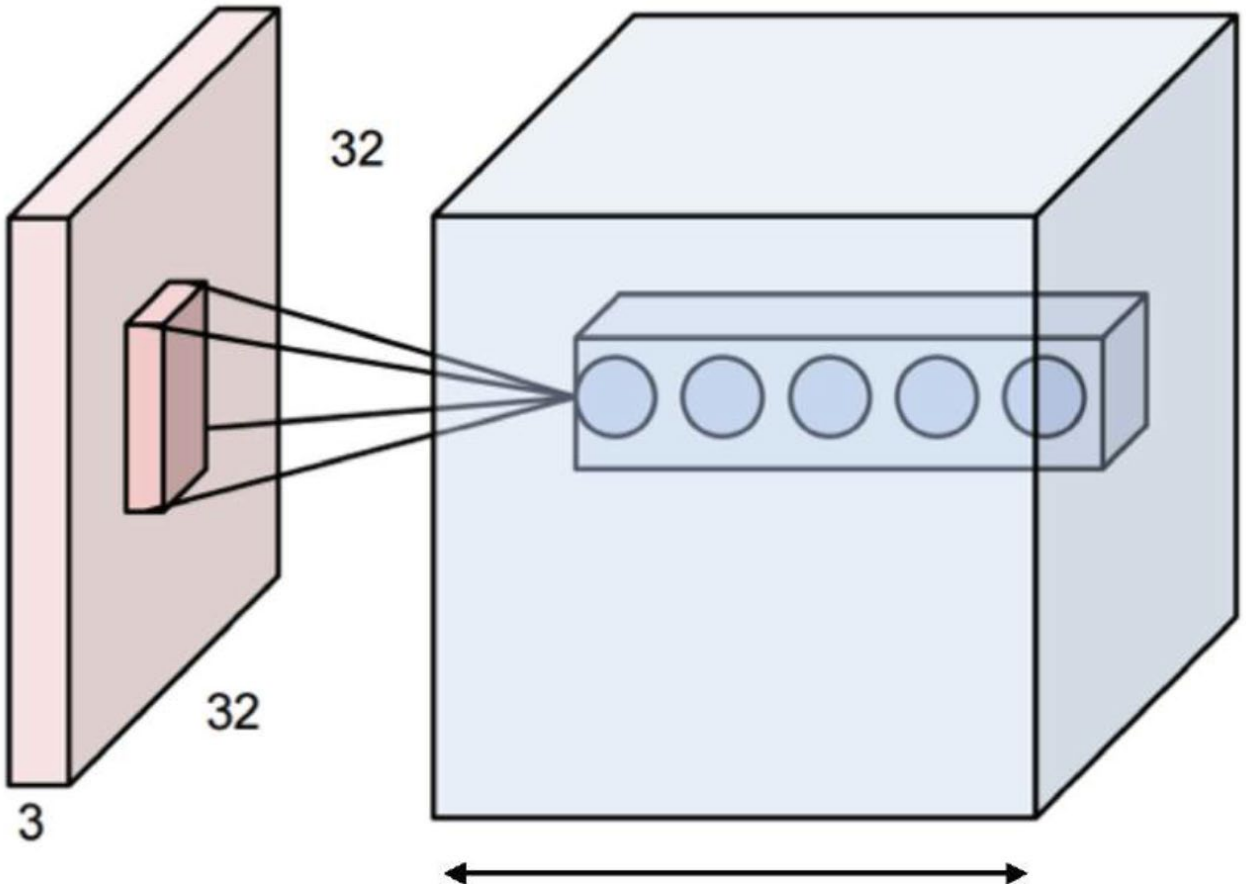


Each neuron has its own 3D filter and own (scalar) bias

We can keep adding more outputs

These form a column in the output volume: [depth x 1 x 1]

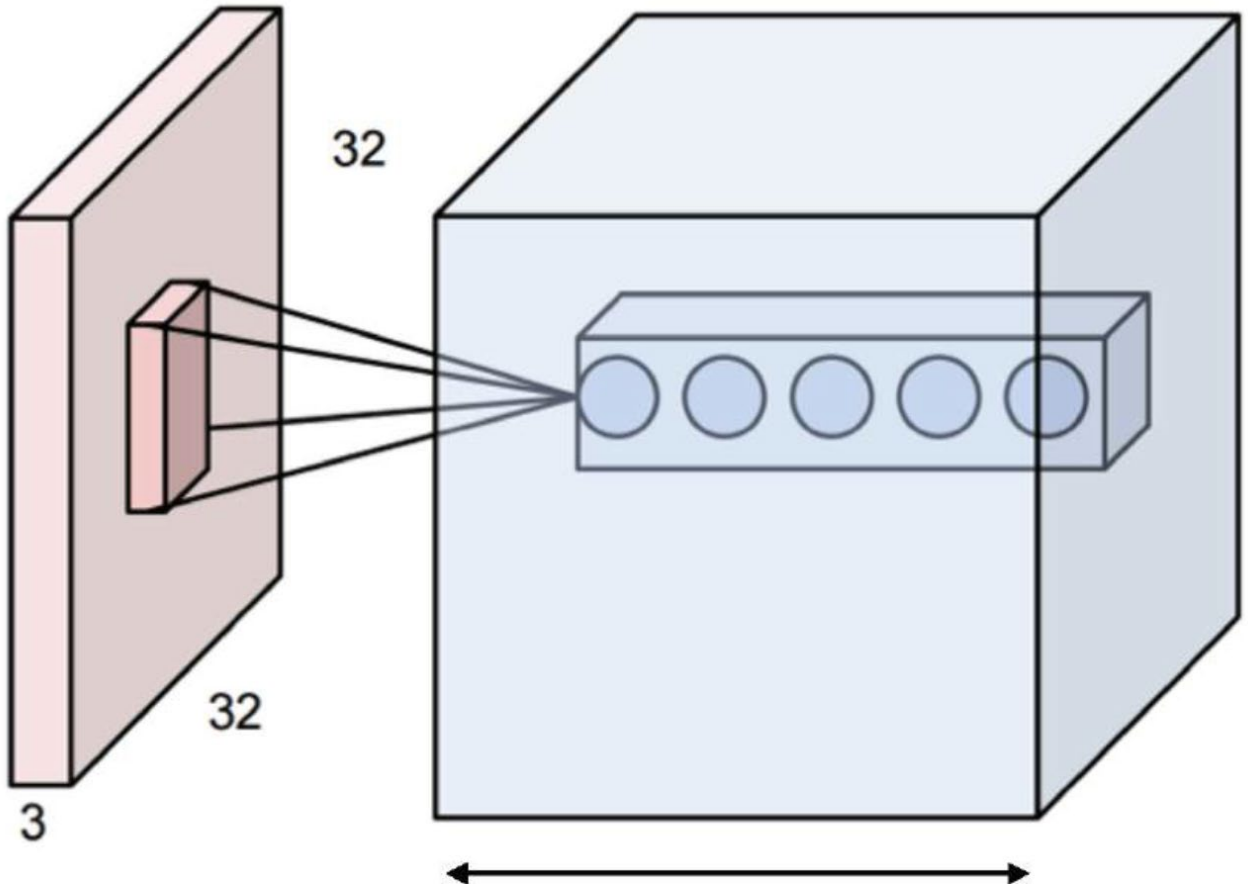
3D Activations



Now repeat this across the input

D sets of weights
(also called filters)

3D Activations



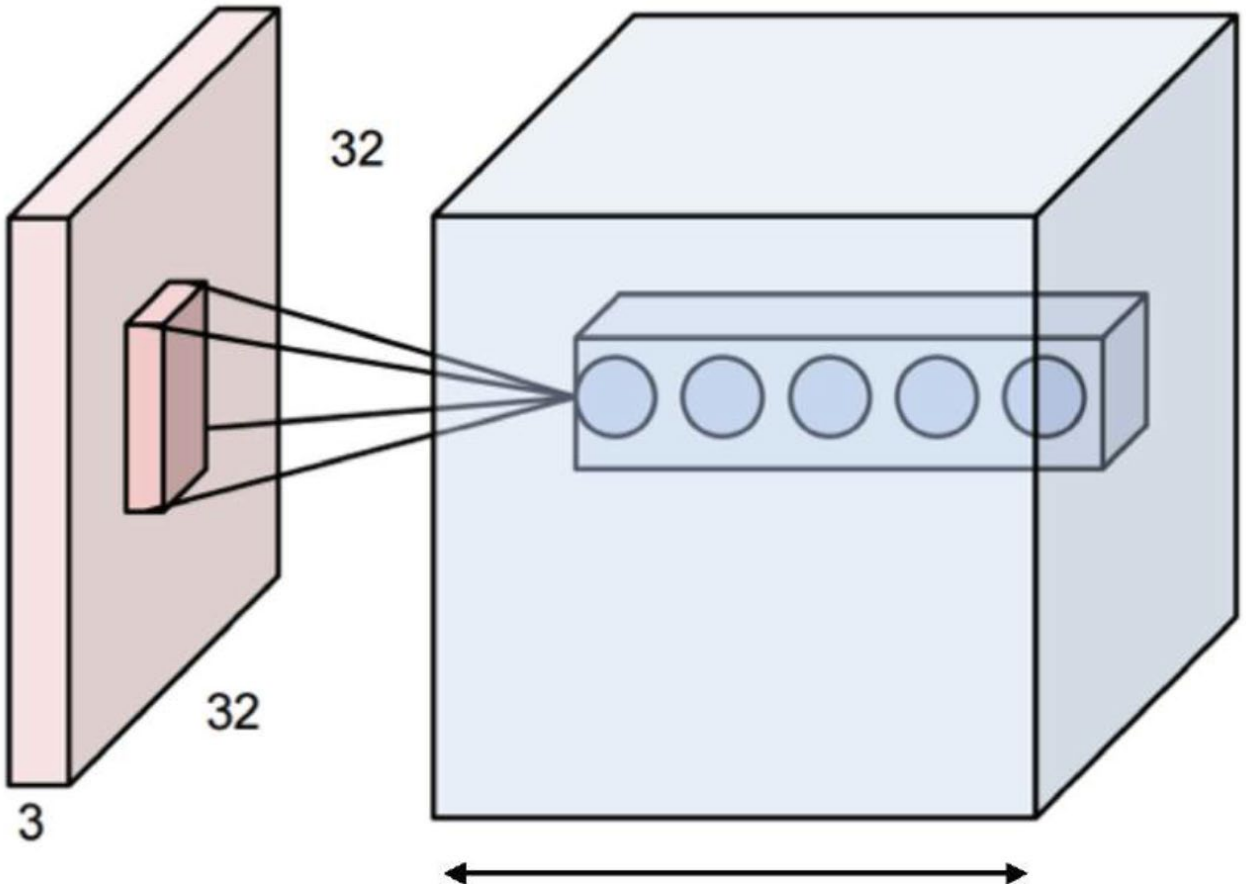
D sets of weights
(also called filters)

Now repeat this across the input

Weight sharing:

Each filter shares the same weights (but each depth index has its own set of weights)

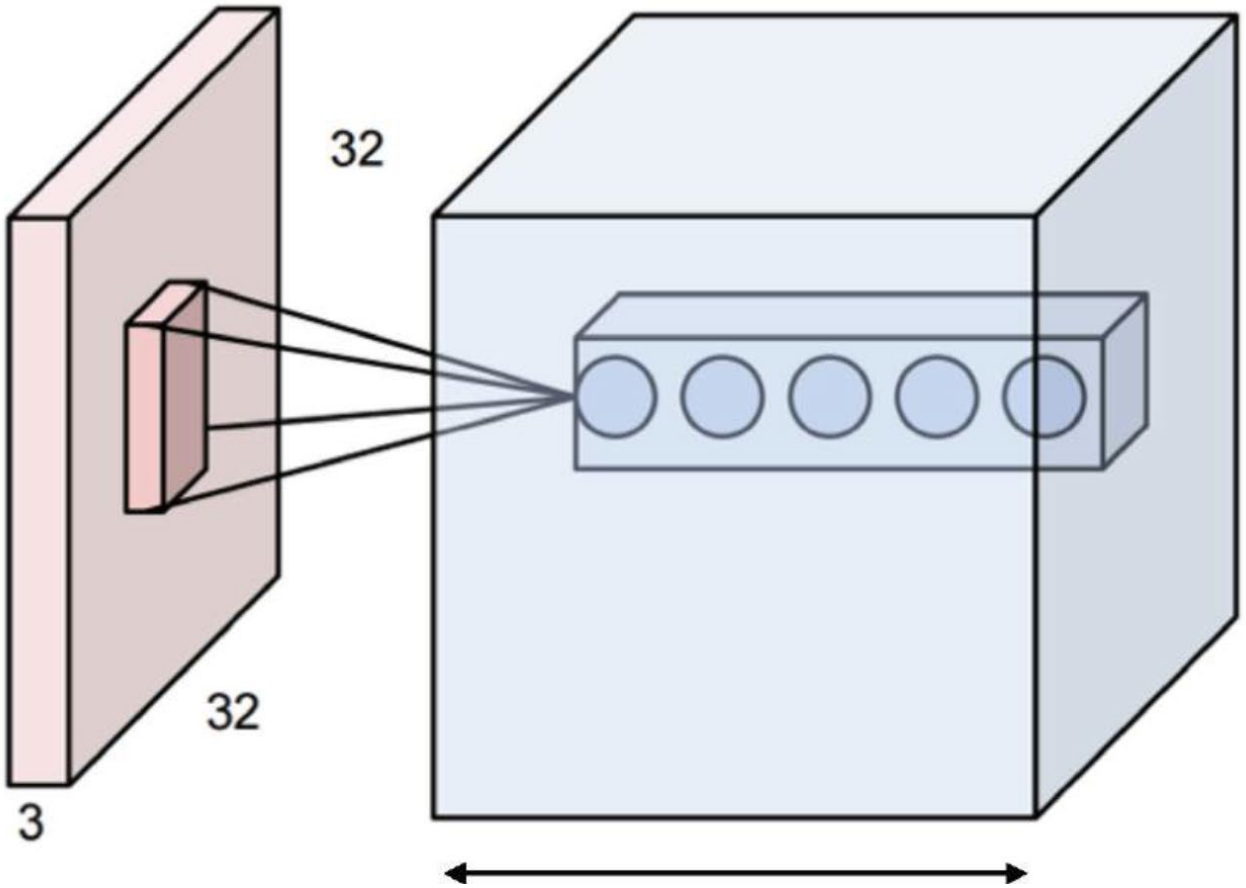
3D Activations



D sets of weights
(also called filters)

With weight sharing, this is called **convolution**

3D Activations

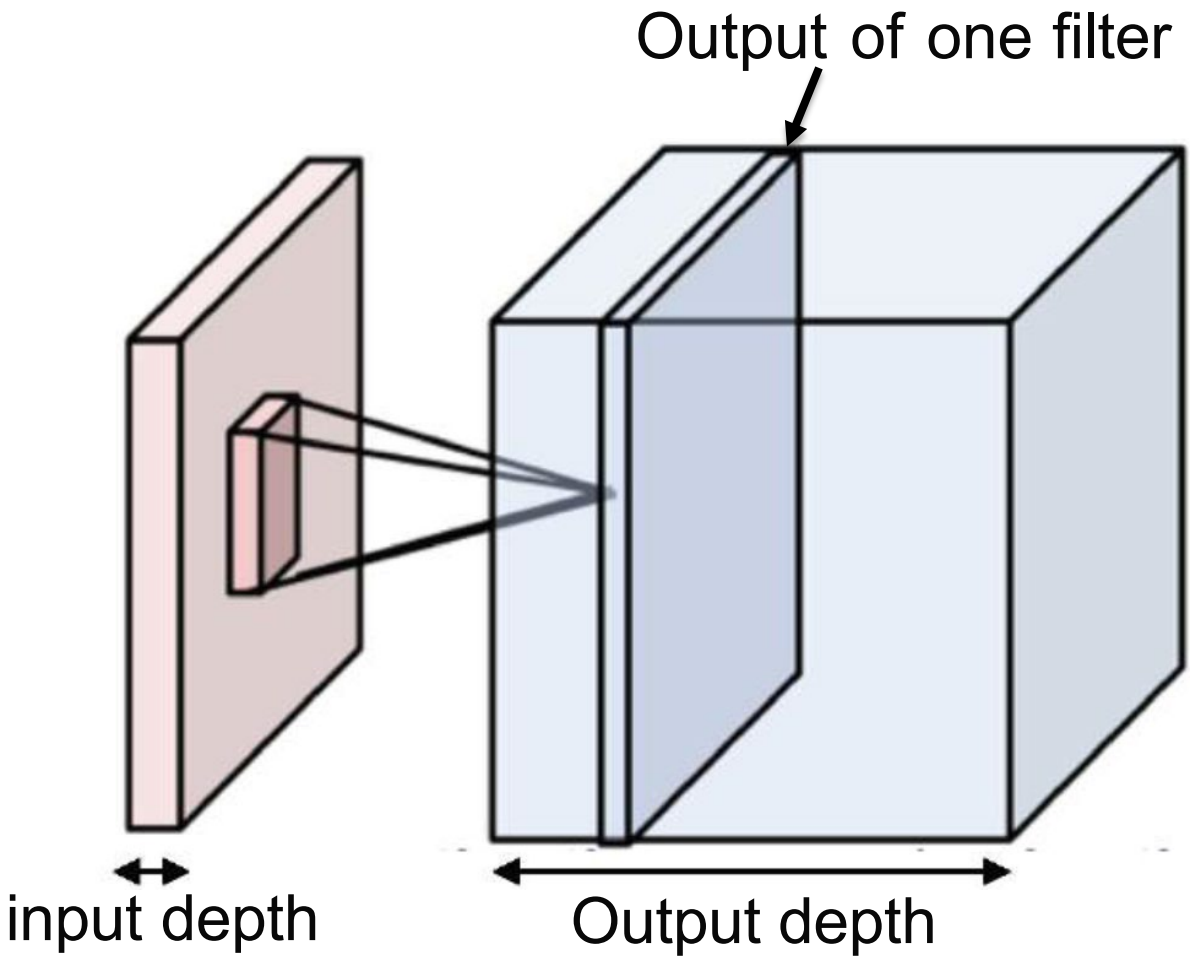


D sets of weights
(also called filters)

With weight sharing, this is called **convolution**

Without weight sharing, this is called a **locally connected layer**

3D Activations

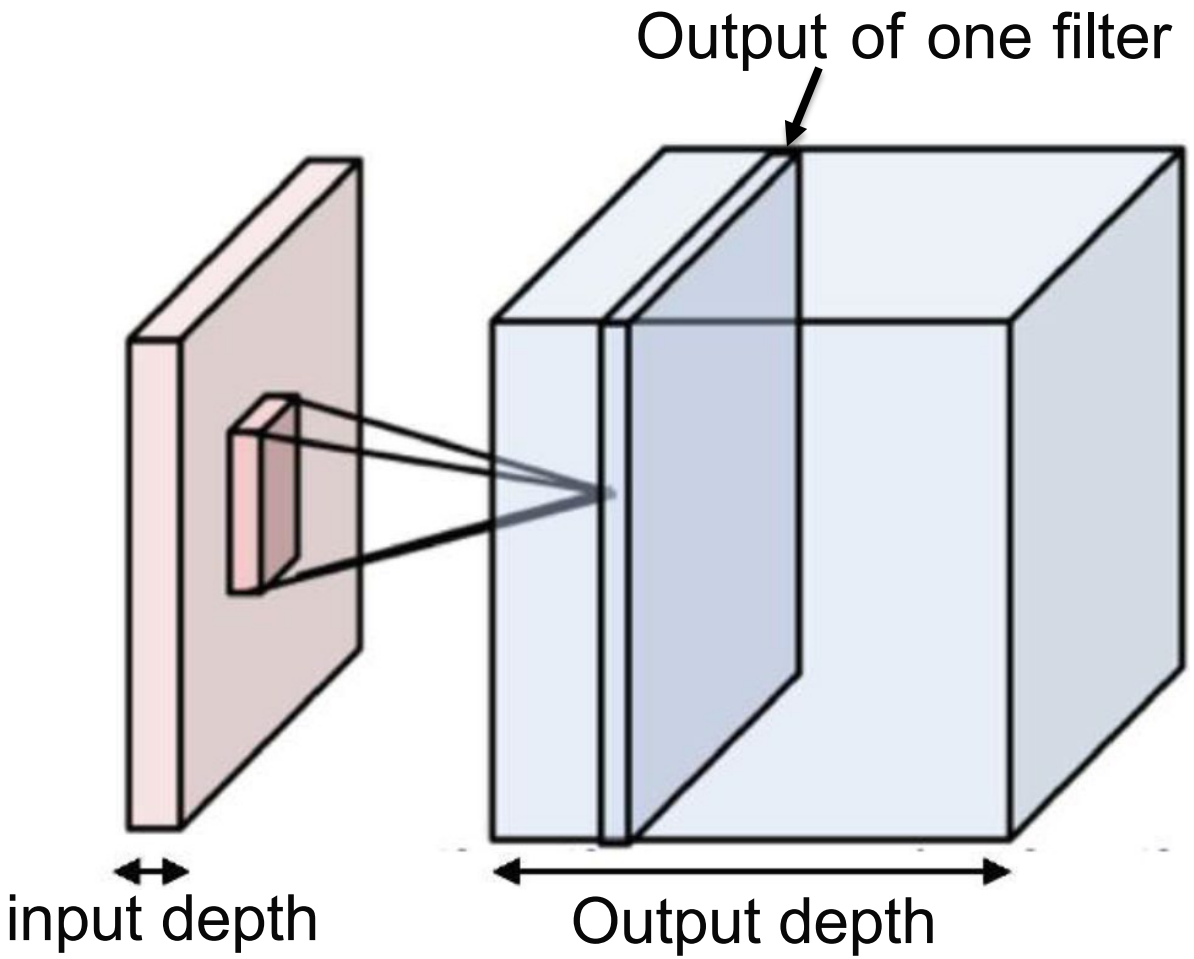


One set of weights gives one slice in the output

To get a 3D output of depth D, use D different filters

In practice, ConvNets use many filters (~64 to 1024)

3D Activations



One set of weights gives one slice in the output

To get a 3D output of depth D, use D different filters

In practice, ConvNets use many filters (~64 to 1024)

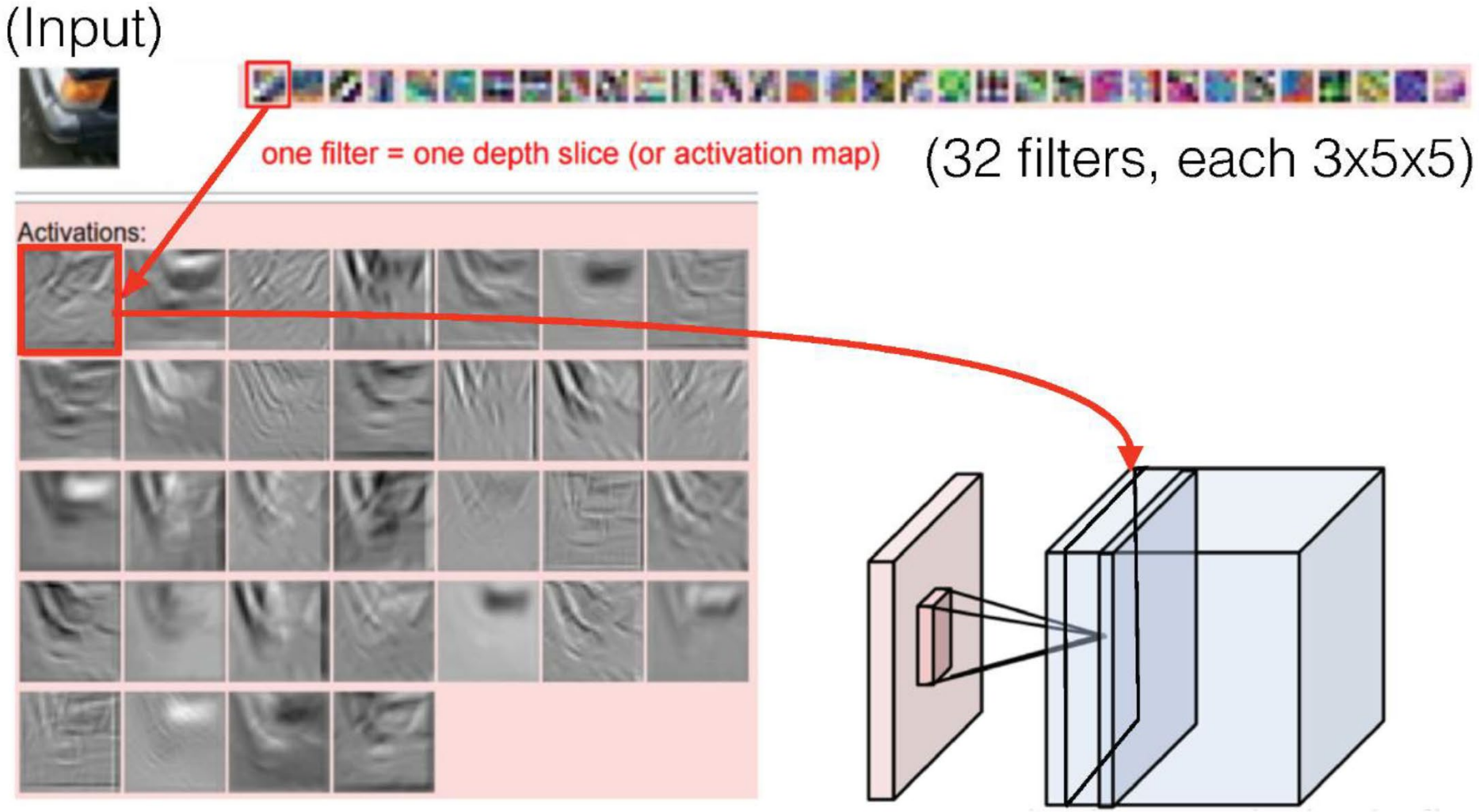
All together, the weights are **4** dimensional:
(output depth, input depth, kernel height, kernel width)

3D Activations

We can unravel the 3D cube and show each layer separately:

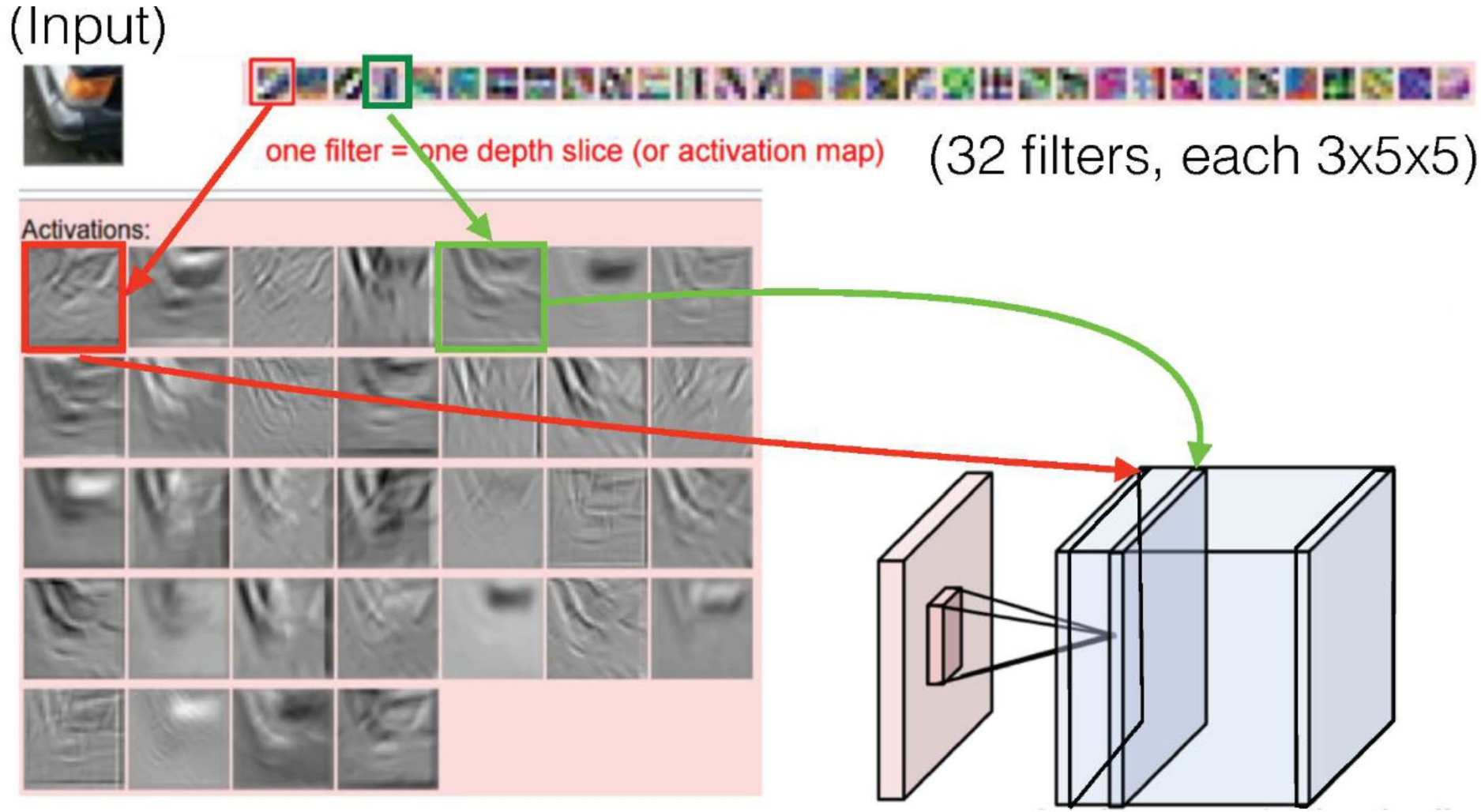
3D Activations

We can unravel the 3D cube and show each layer separately:



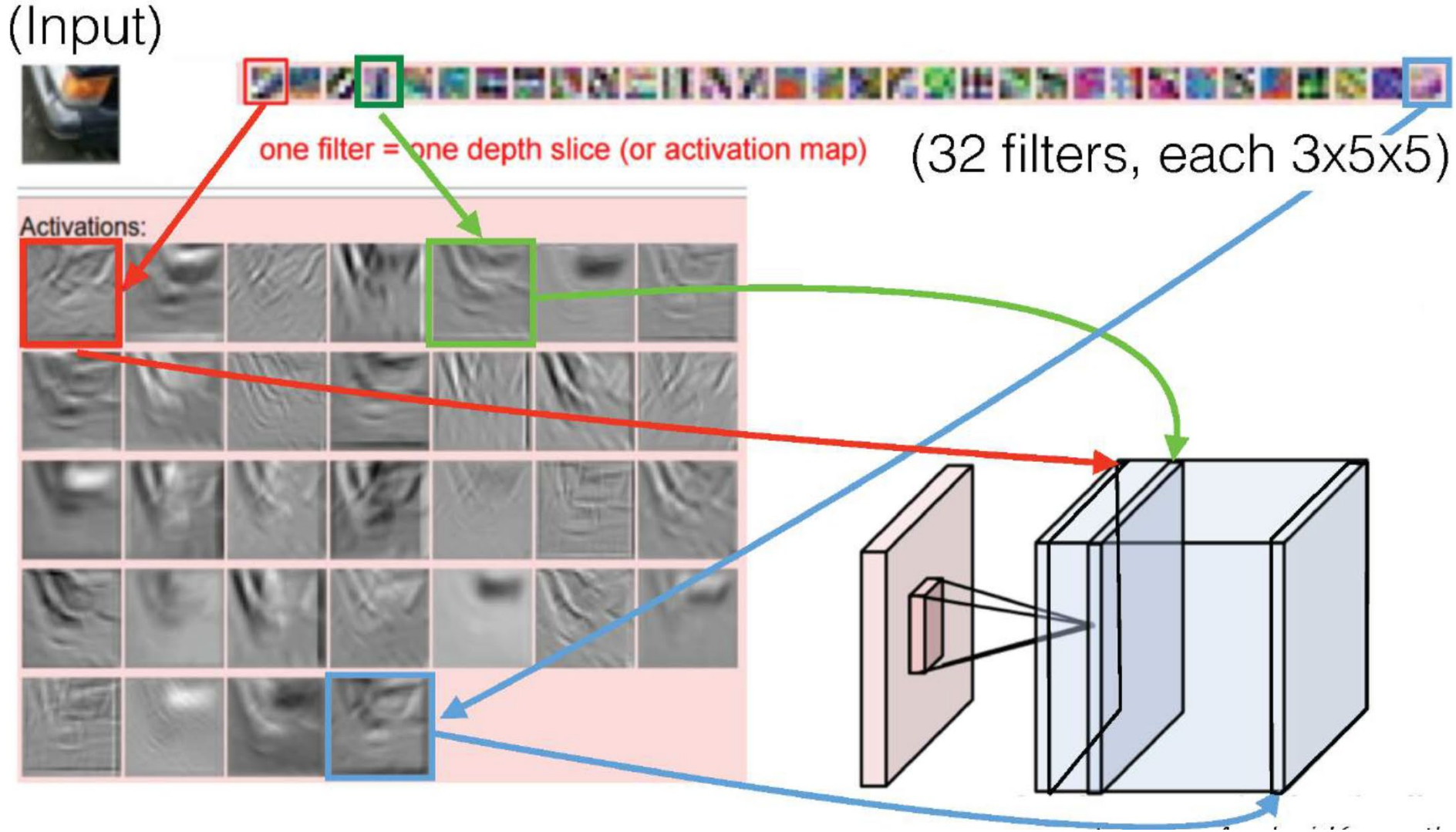
3D Activations

We can unravel the 3D cube and show each layer separately:



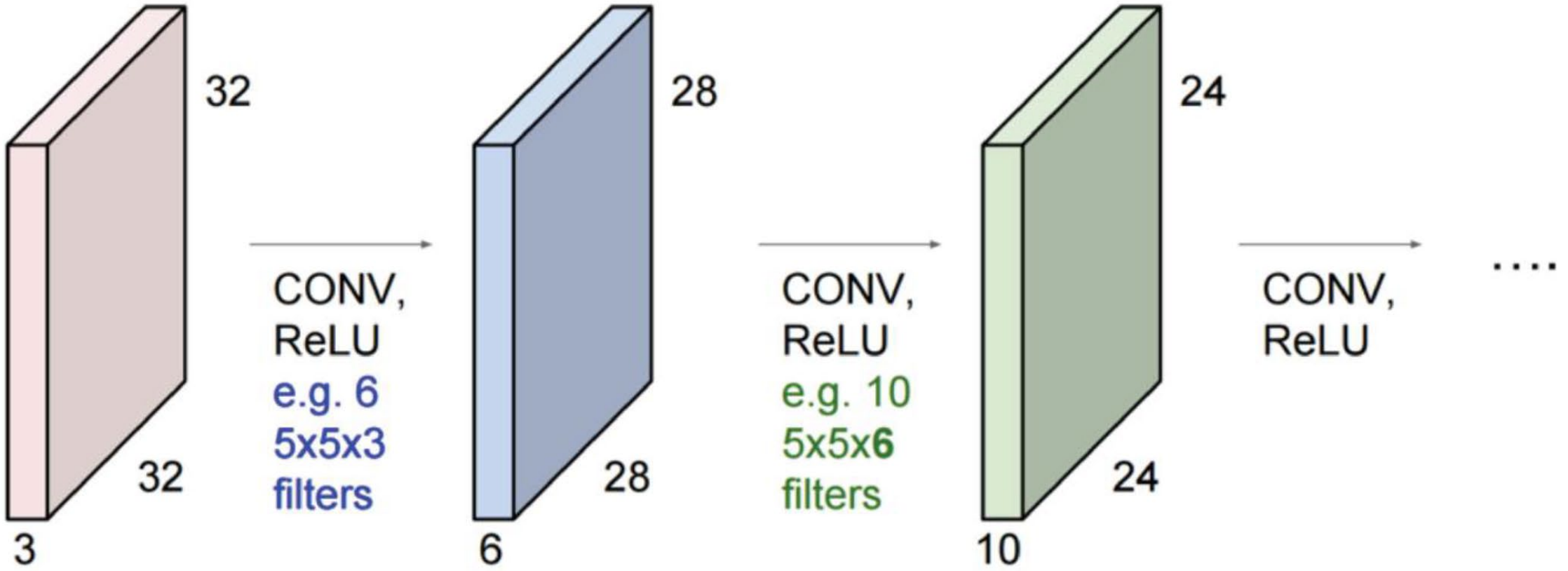
3D Activations

We can unravel the 3D cube and show each layer separately:



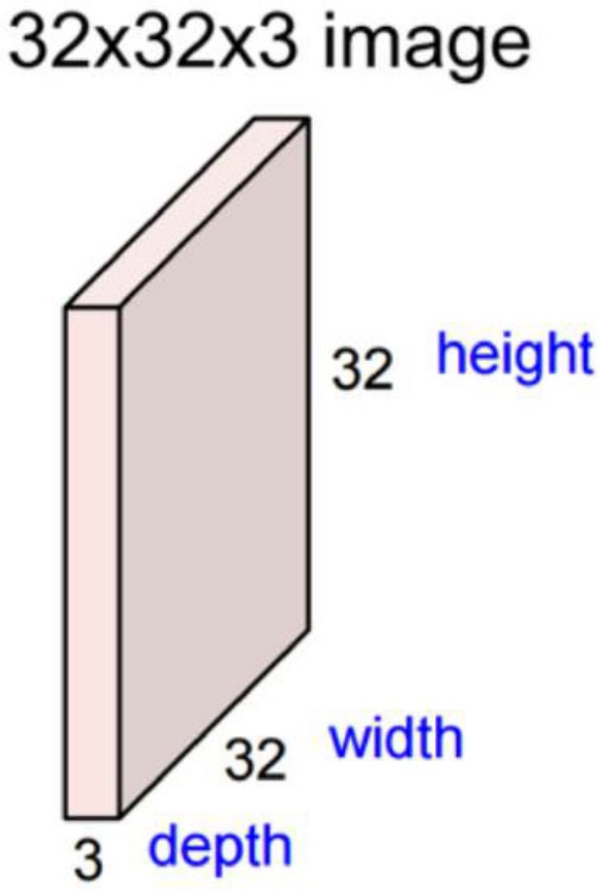
Recap

A **ConvNet** is a sequence of convolutional layers, interspersed with activation functions (and possibly other layer types)



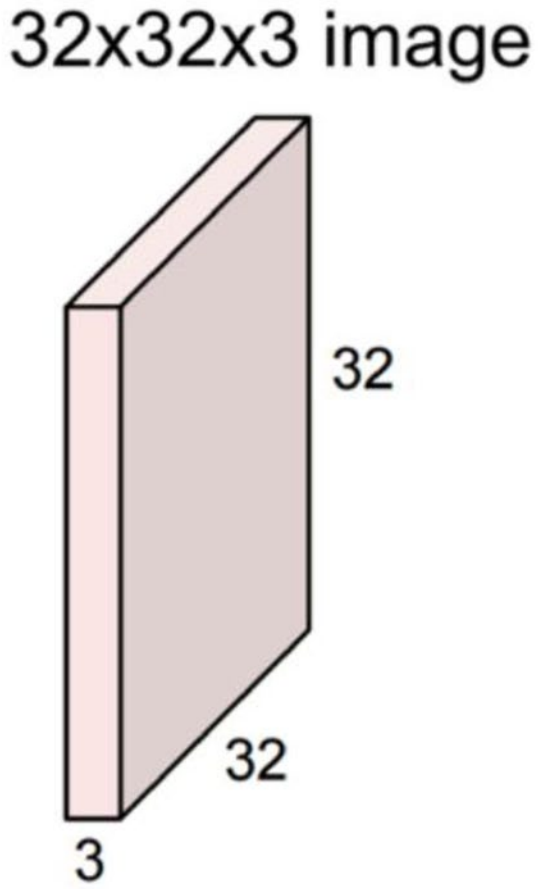
Recap

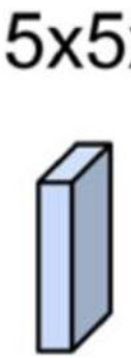
Convolution Layer



Recap

Convolution Layer

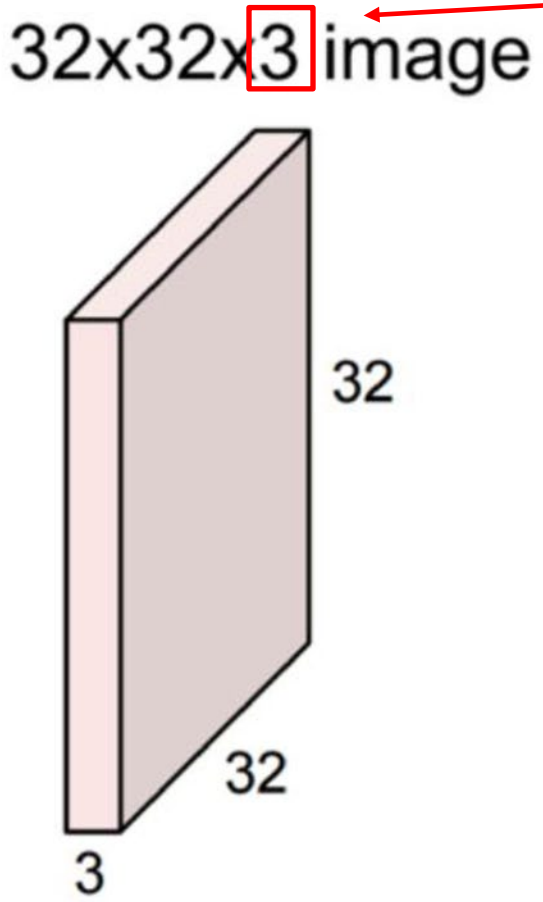




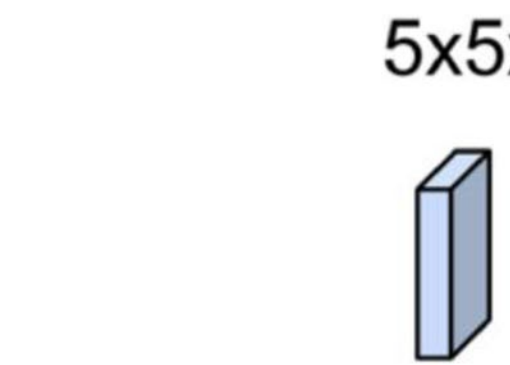
Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

Recap

Convolution Layer



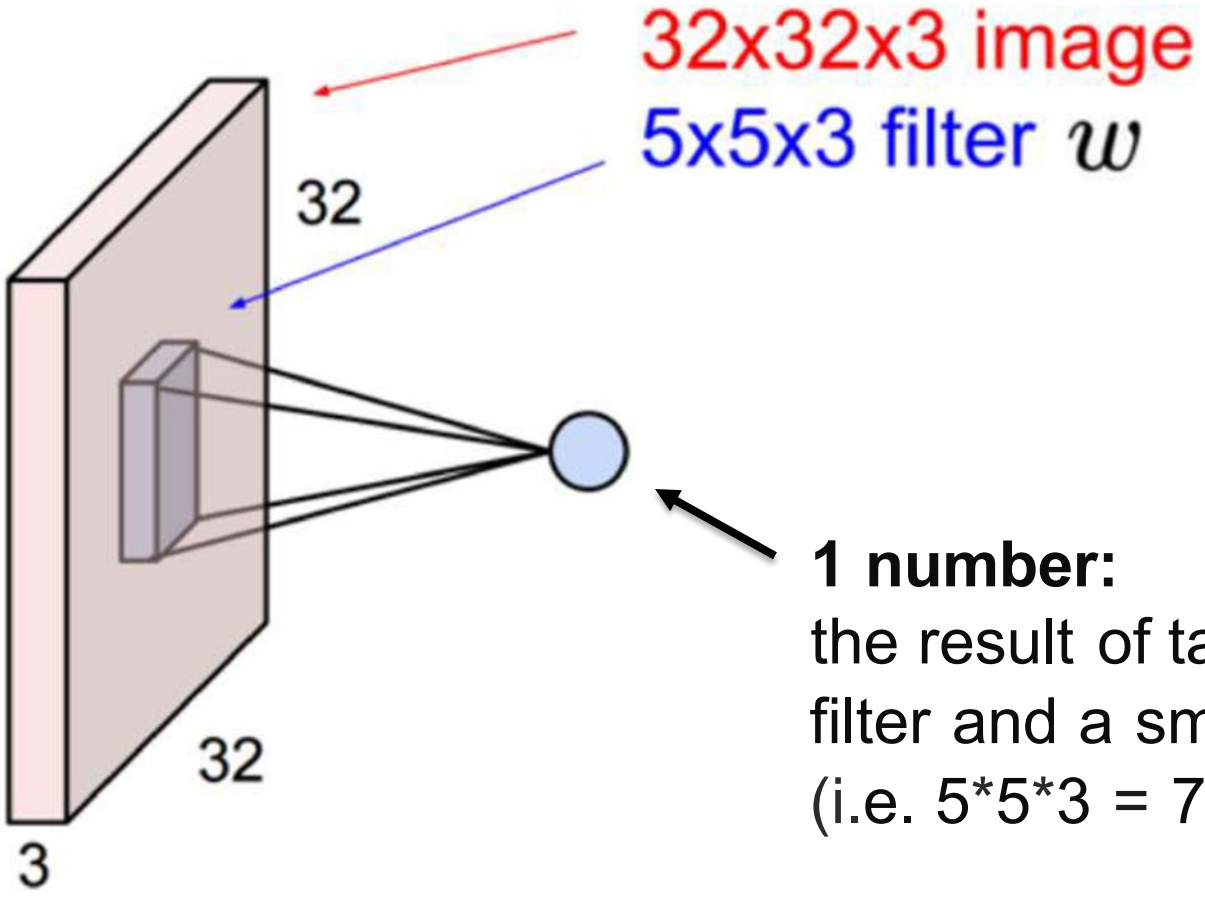
Filters always extend the full depth of the input volume



Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

Recap

Convolution Layer

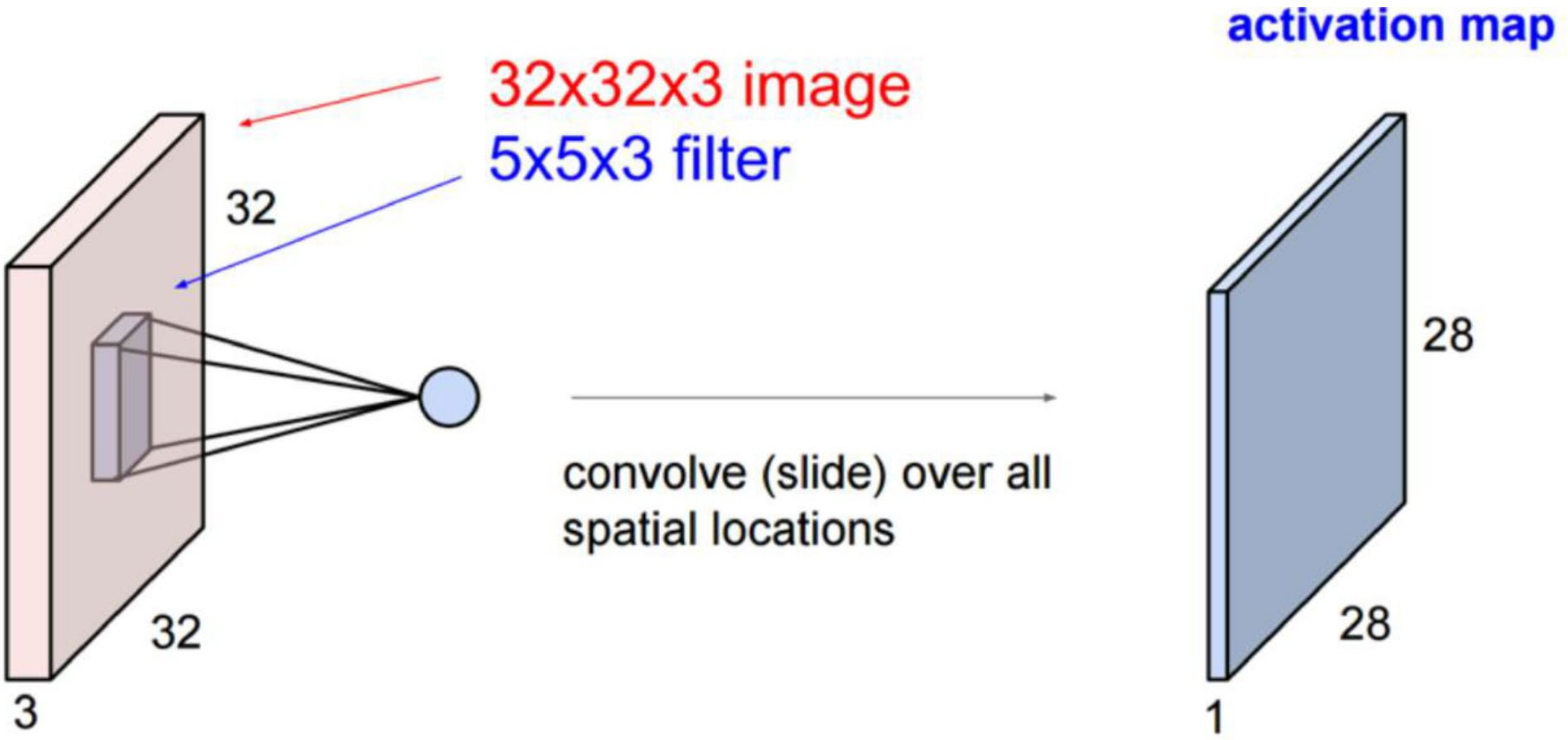


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product+bias)

$$\omega^t x + b$$

Recap

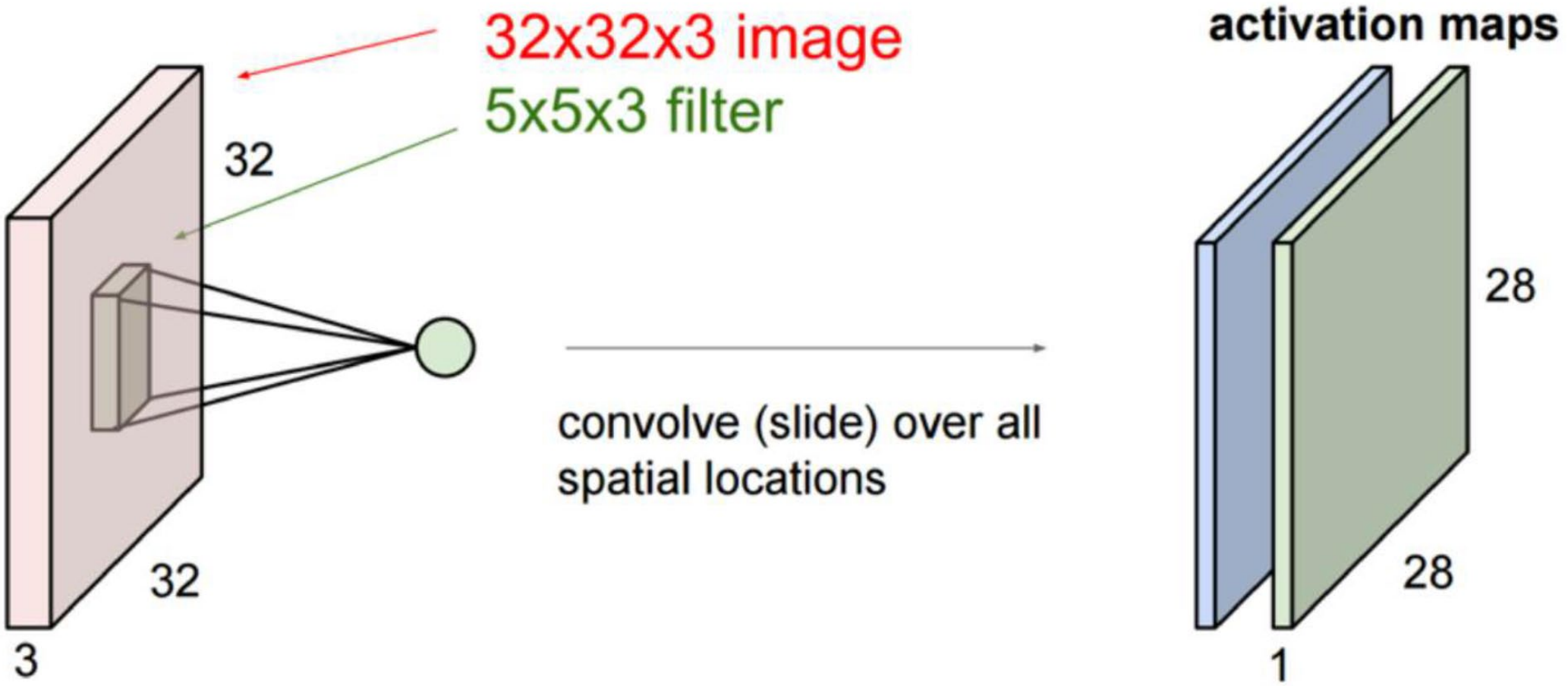
Convolution Layer



Recap

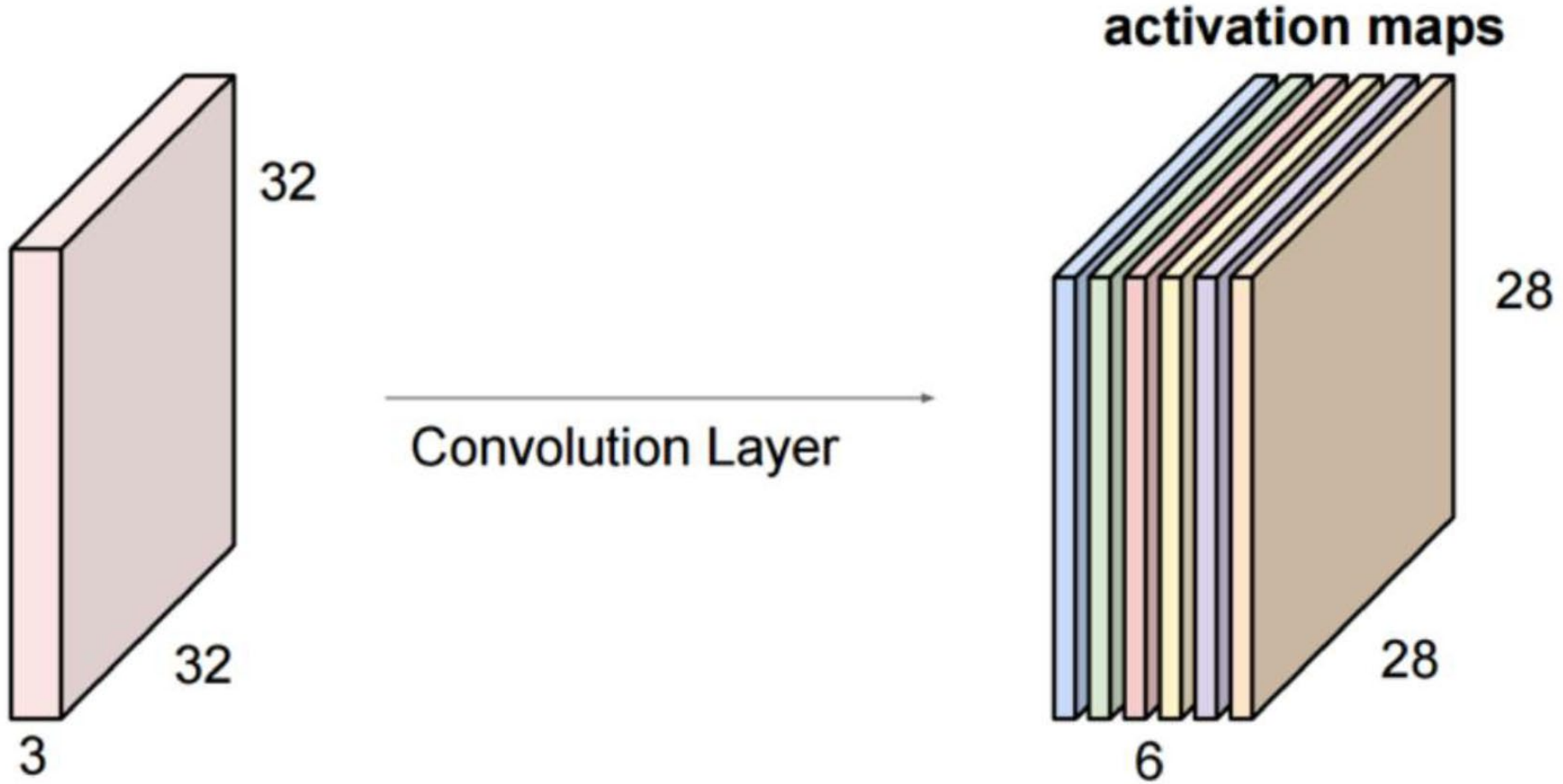
Convolution Layer

consider a second, green filter



Recap

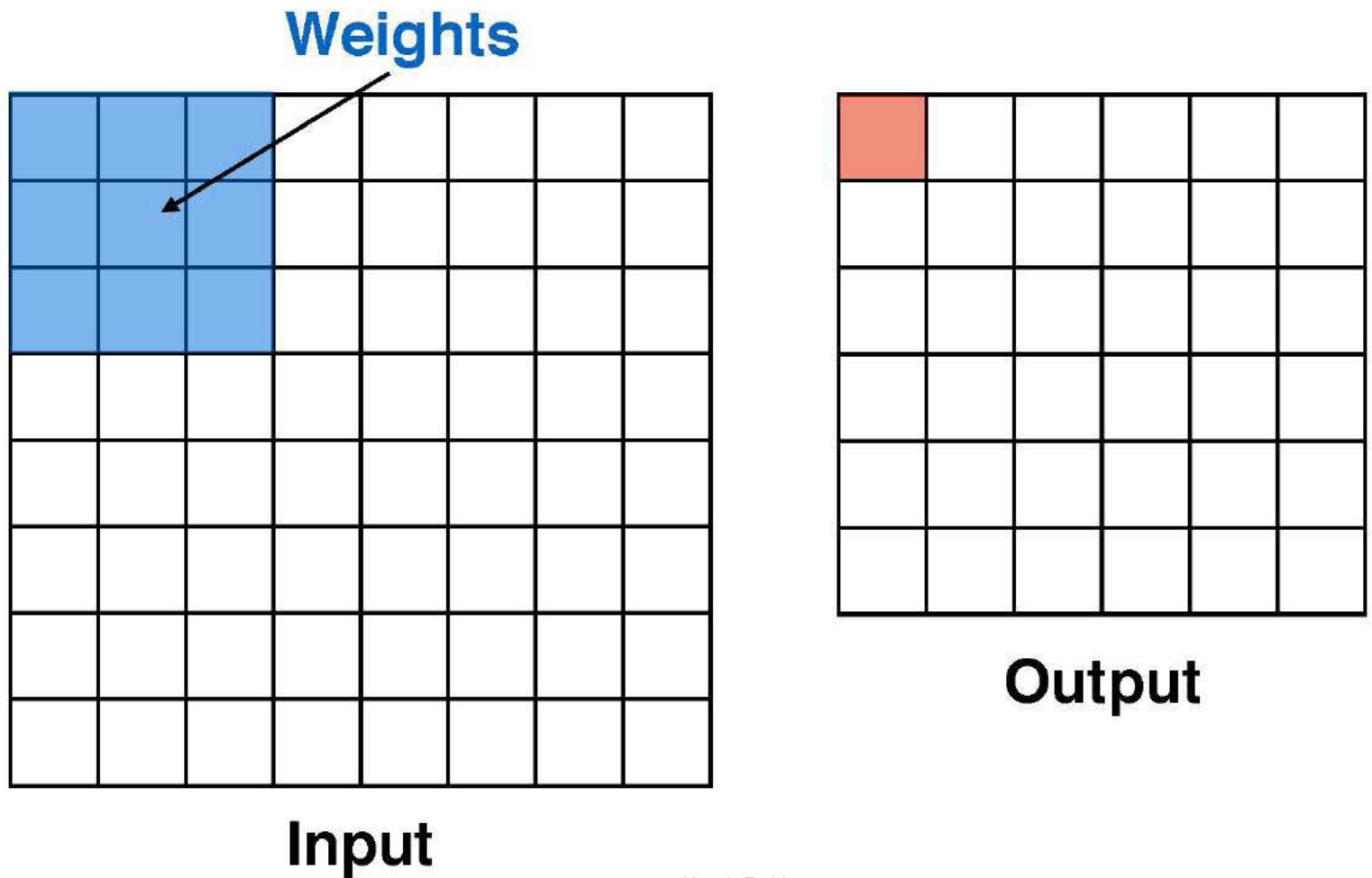
For example, if we had six 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6

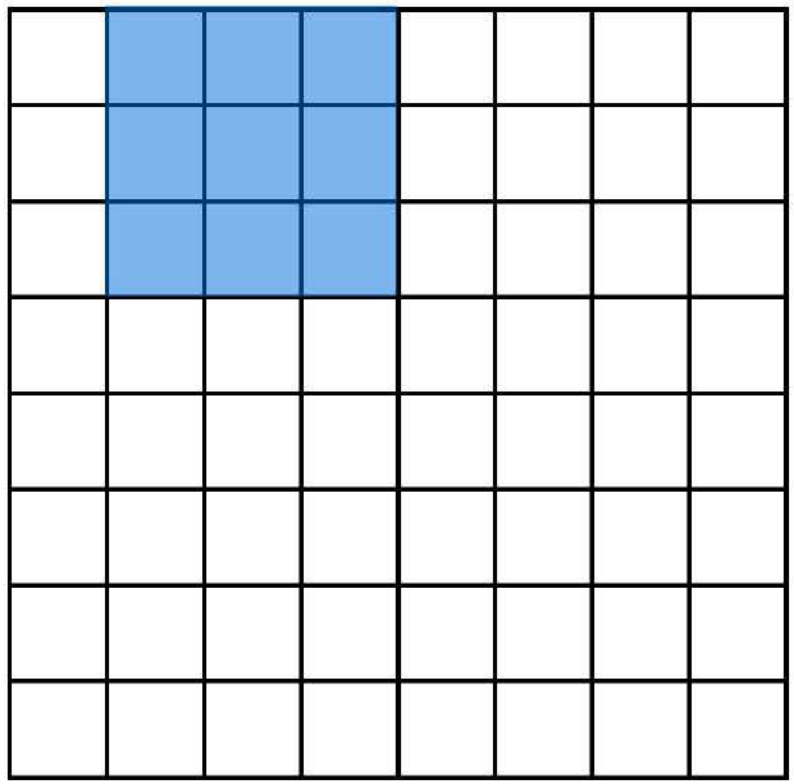
Convolution

During convolution, the weights “slide” along the input to generate each output

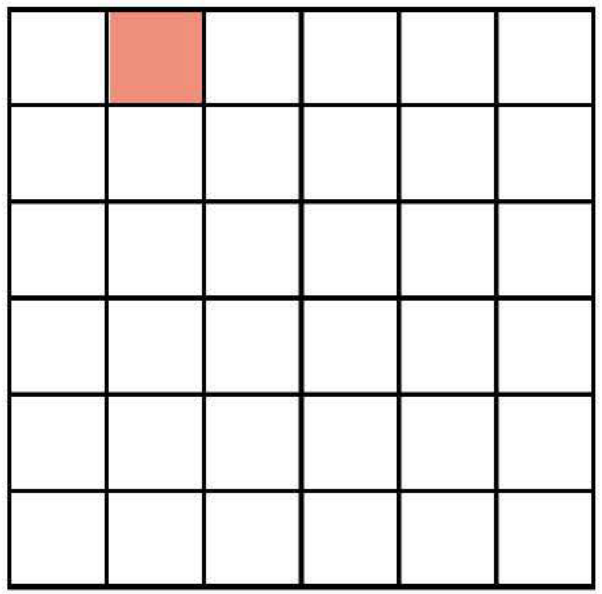


Convolution

During convolution, the weights “slide” along the input to generate each output



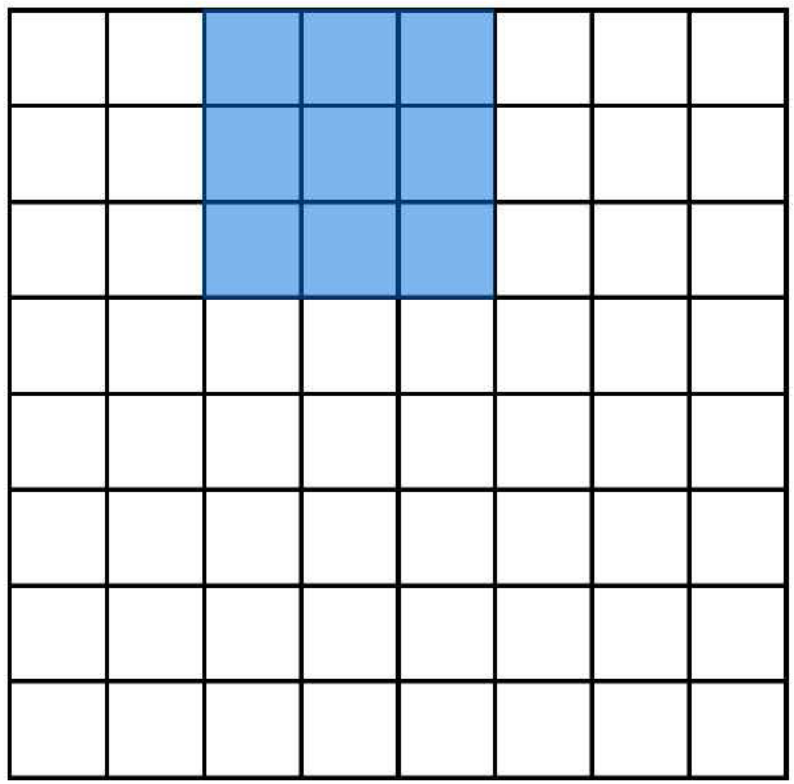
Input



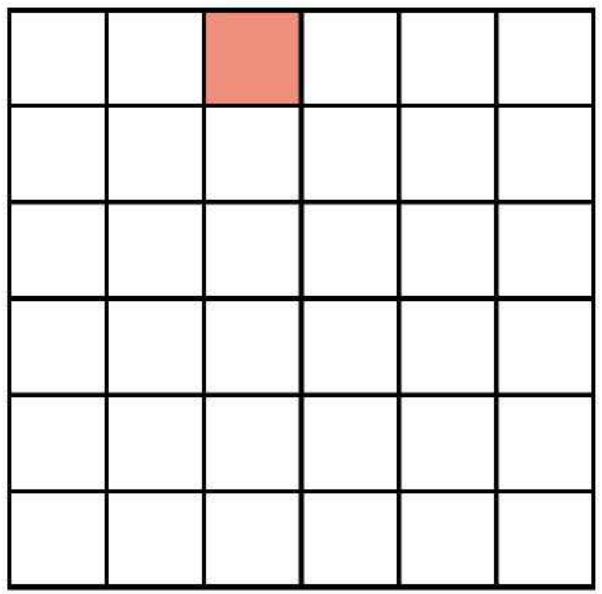
Output

Convolution

During convolution, the weights “slide” along the input to generate each output



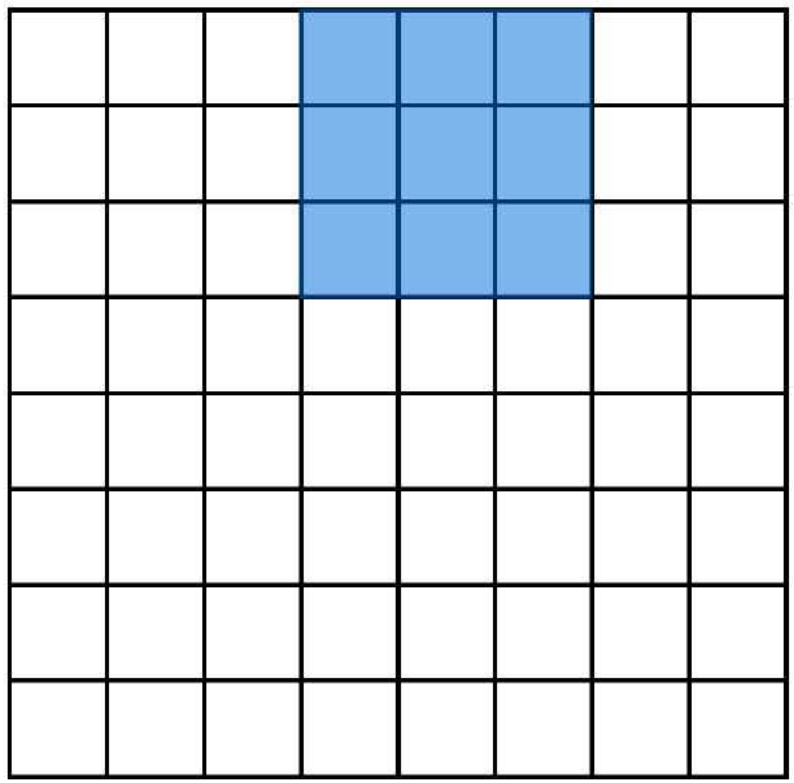
Input



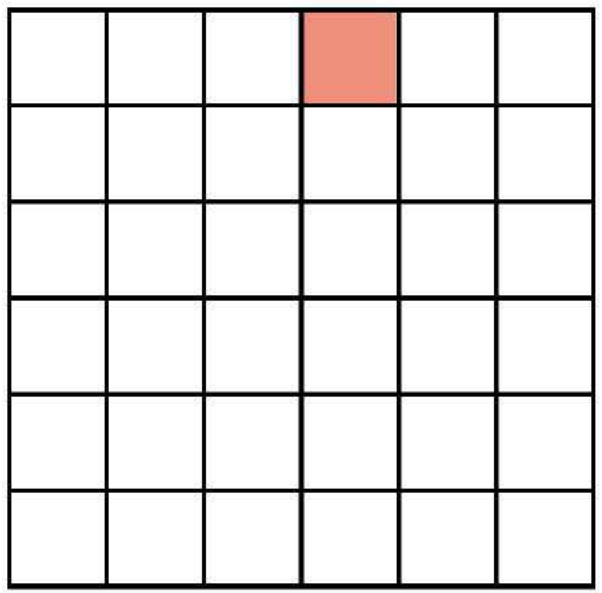
Output

Convolution

During convolution, the weights “slide” along the input to generate each output



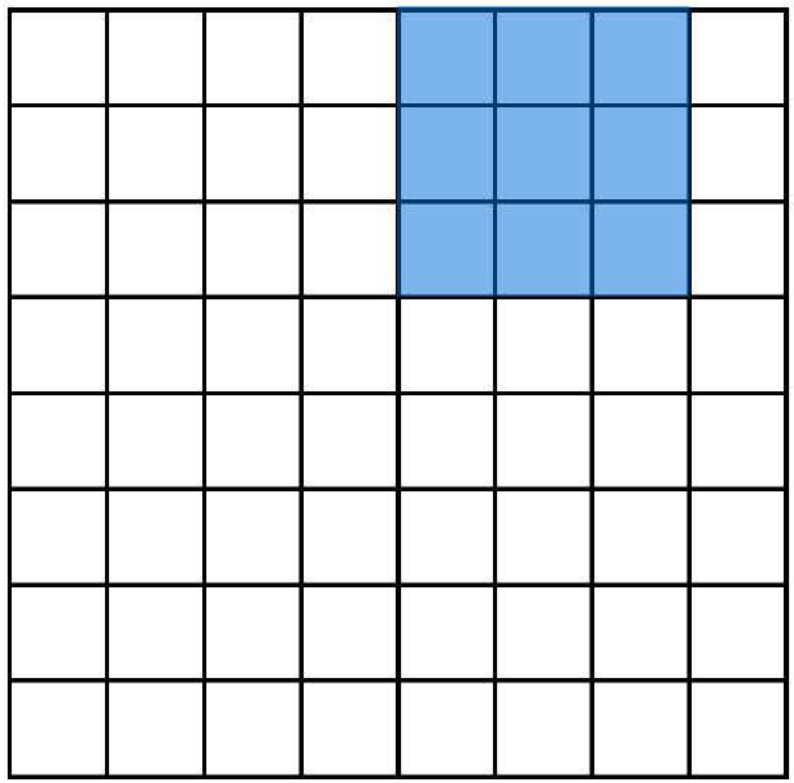
Input



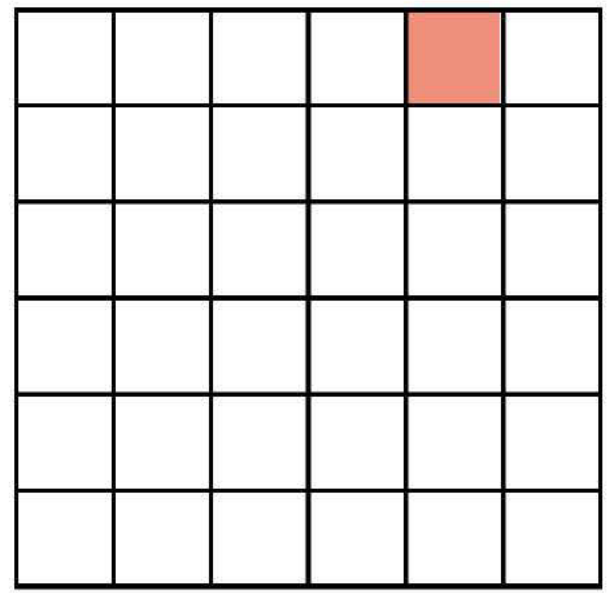
Output

Convolution

During convolution, the weights “slide” along the input to generate each output



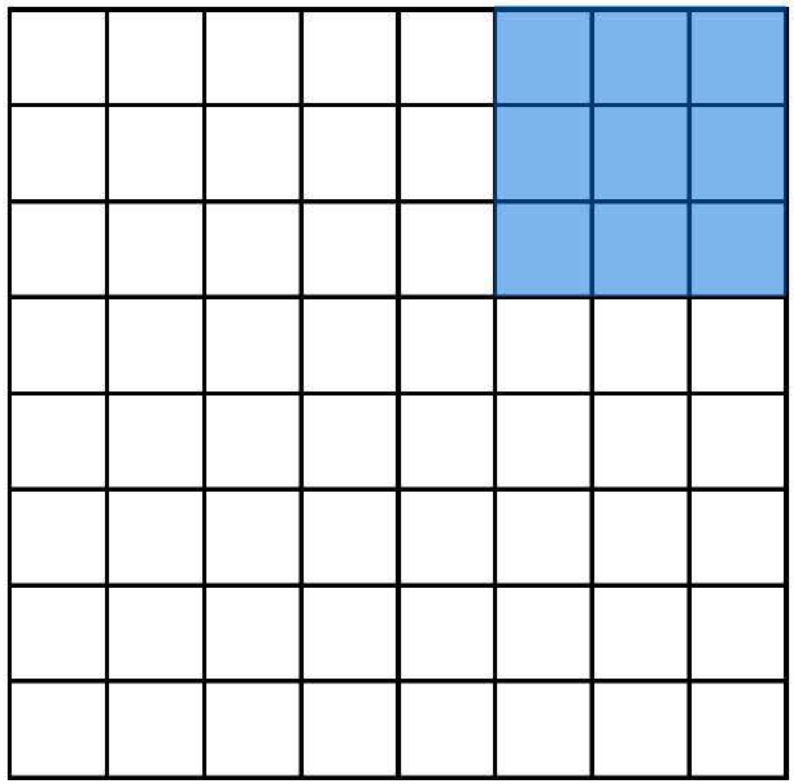
Input



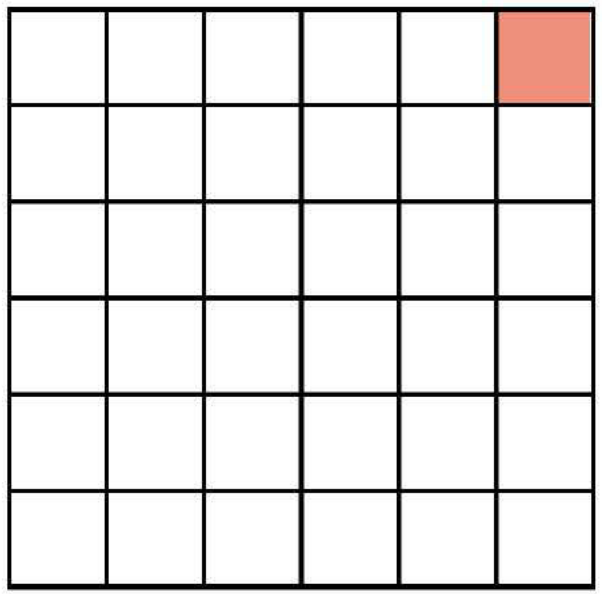
Output

Convolution

During convolution, the weights “slide” along the input to generate each output



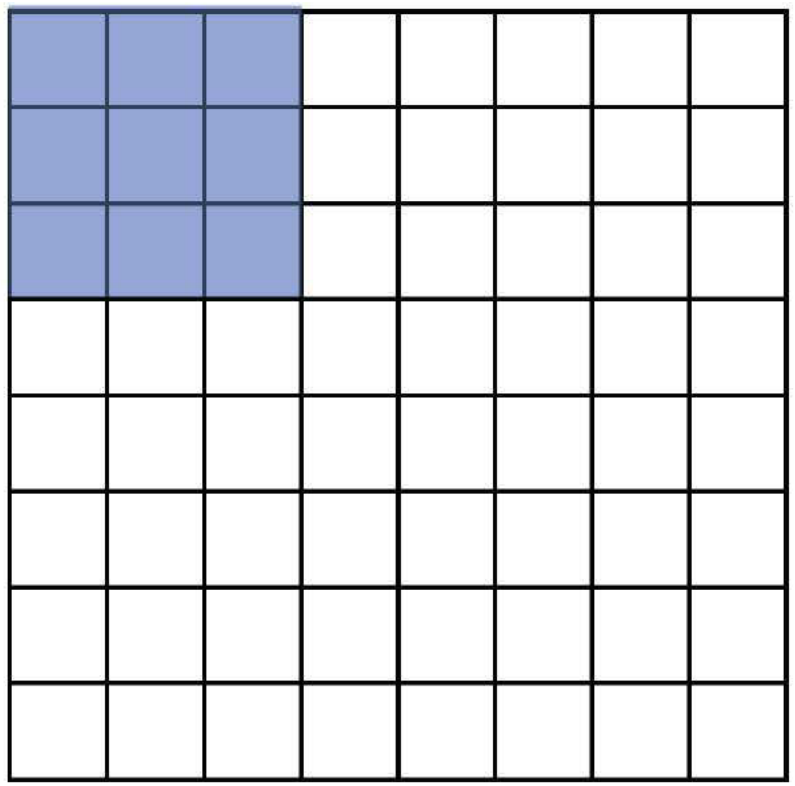
Input



Output

Convolution

During convolution, the weights "slide" along the input to generate each output



Input

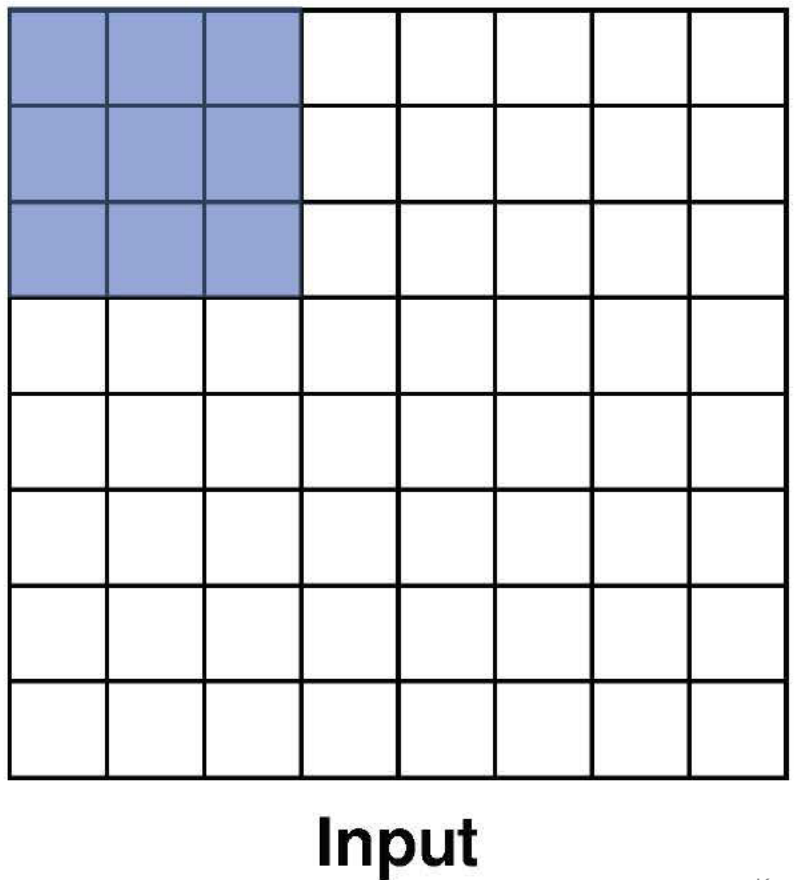
Recall that at each position, we are doing a **3D** sum:

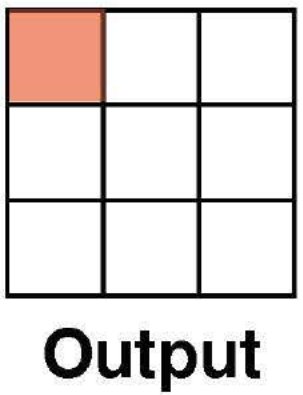
$$h^r = \sum_{ijk} x^r{}_{ijk} W_{ijk} + b$$

(channel, row, column)

Convolution: Stride

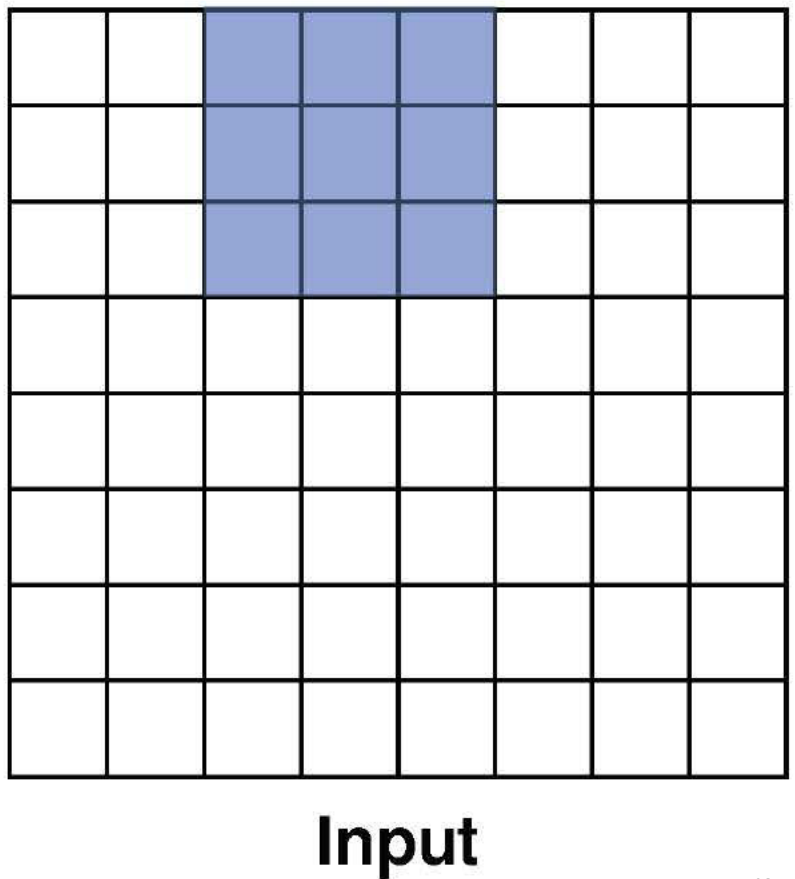
But we can also convolve with a **stride**, e.g. stride = 2

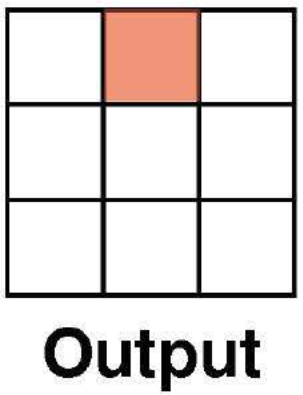




Convolution: Stride

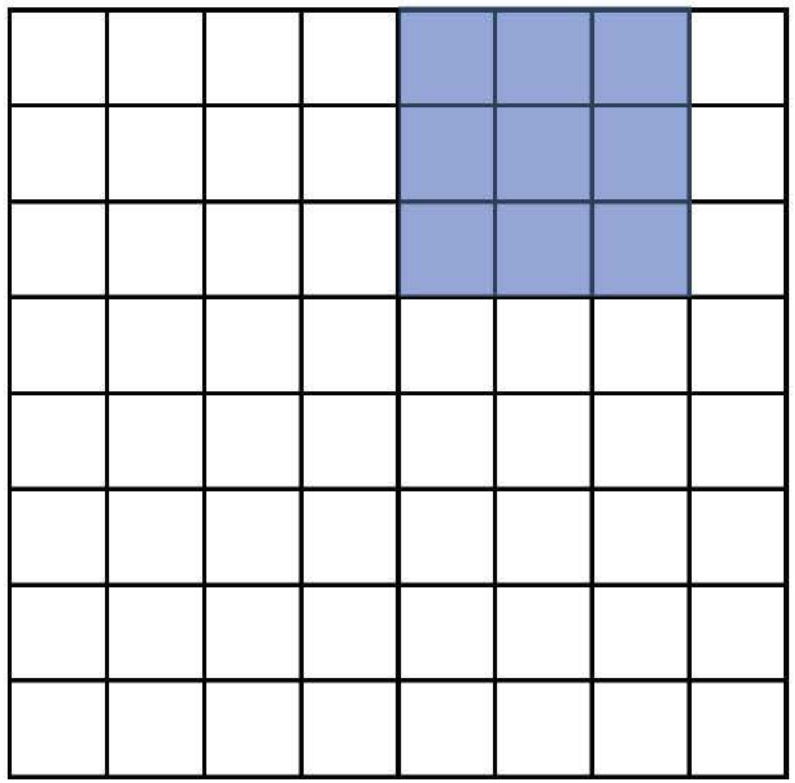
But we can also convolve with a **stride**, e.g. stride = 2



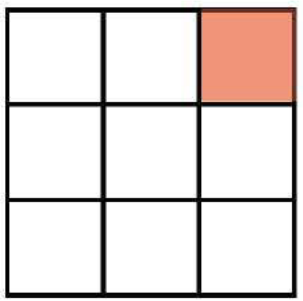


Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2



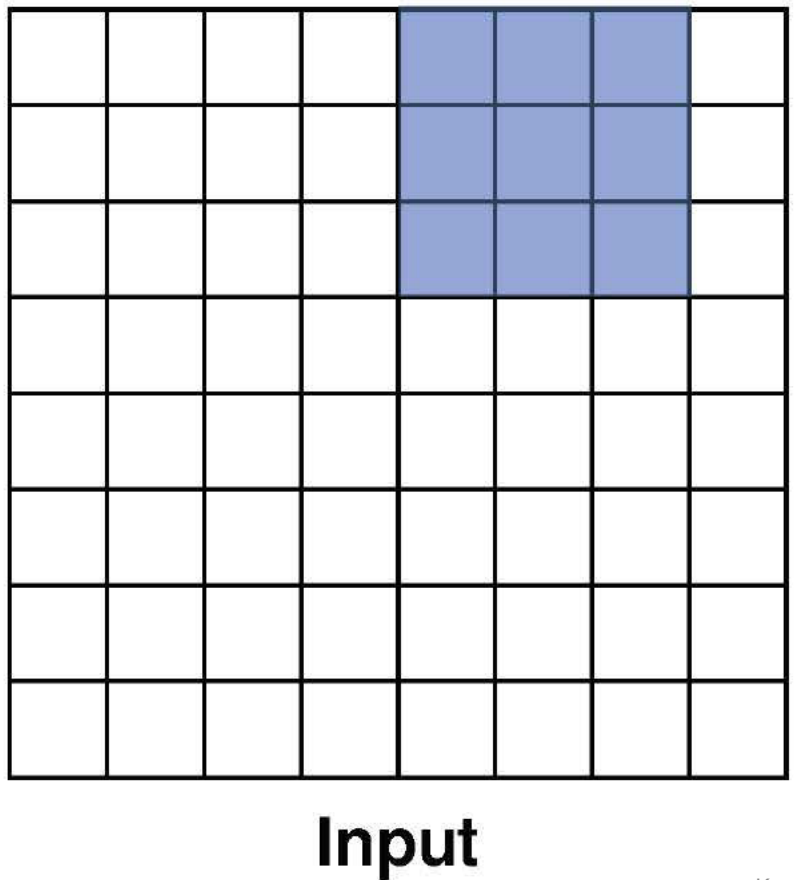
Input



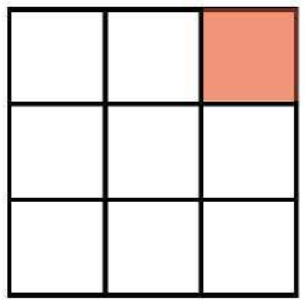
Output

Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2



Input



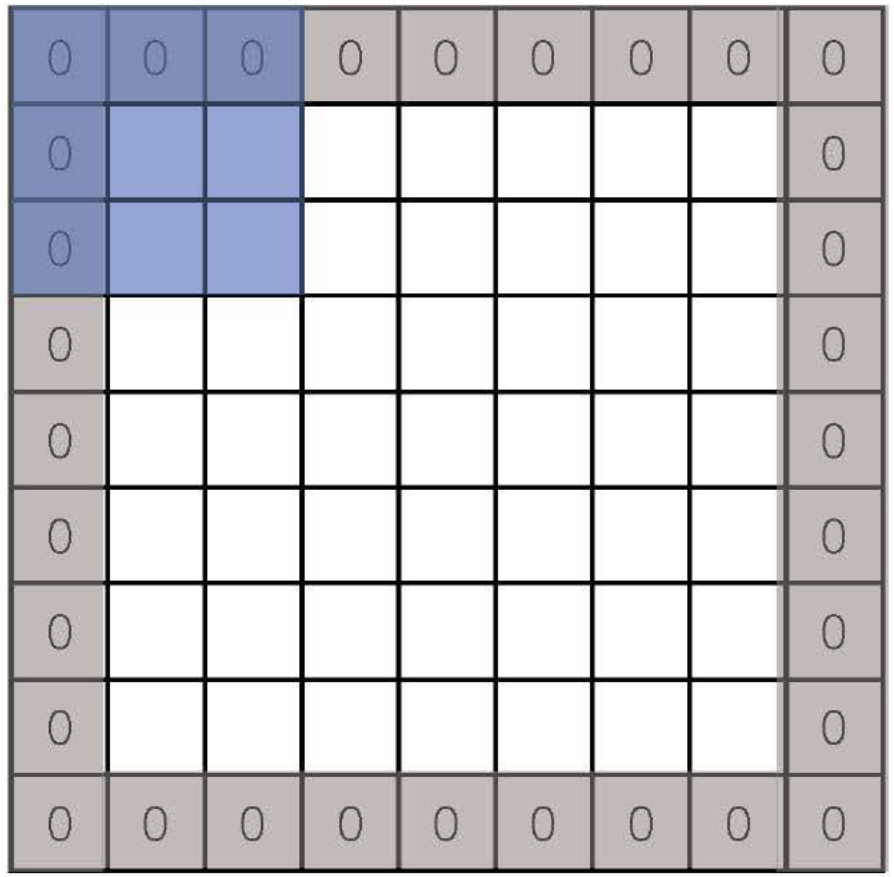
Output

- Notice that with certain strides, we may not be able to cover all of the input

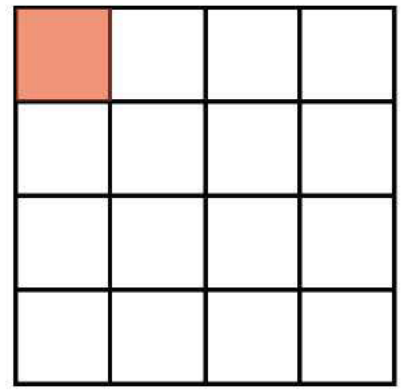
- The output is also half the size of the input

Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



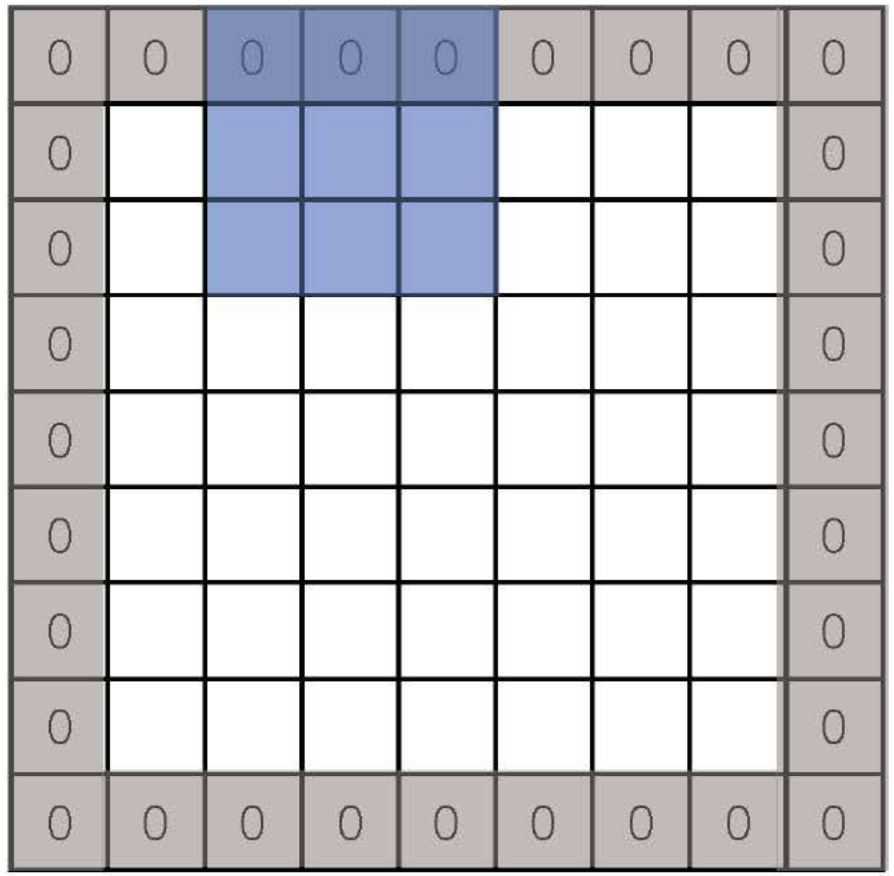
Input



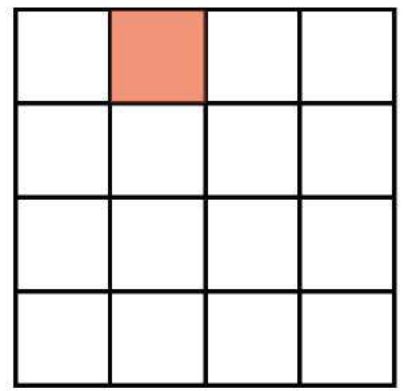
Output

Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



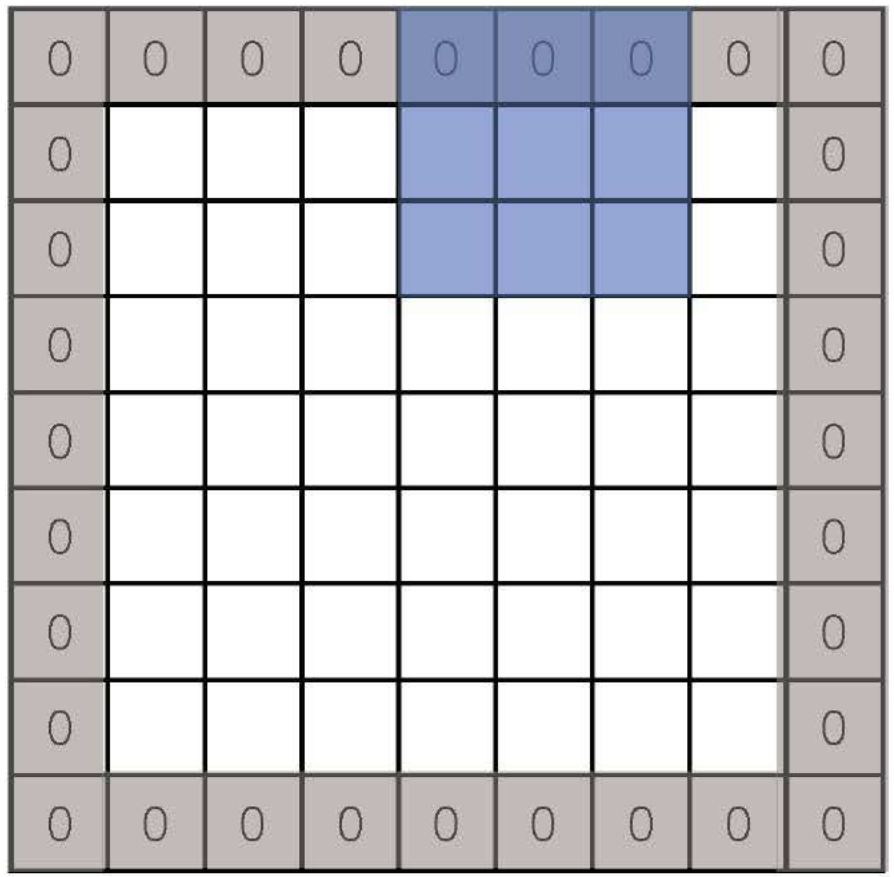
Input



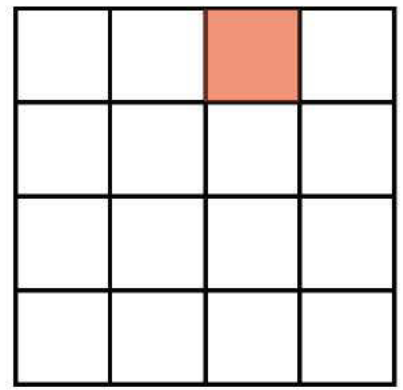
Output

Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



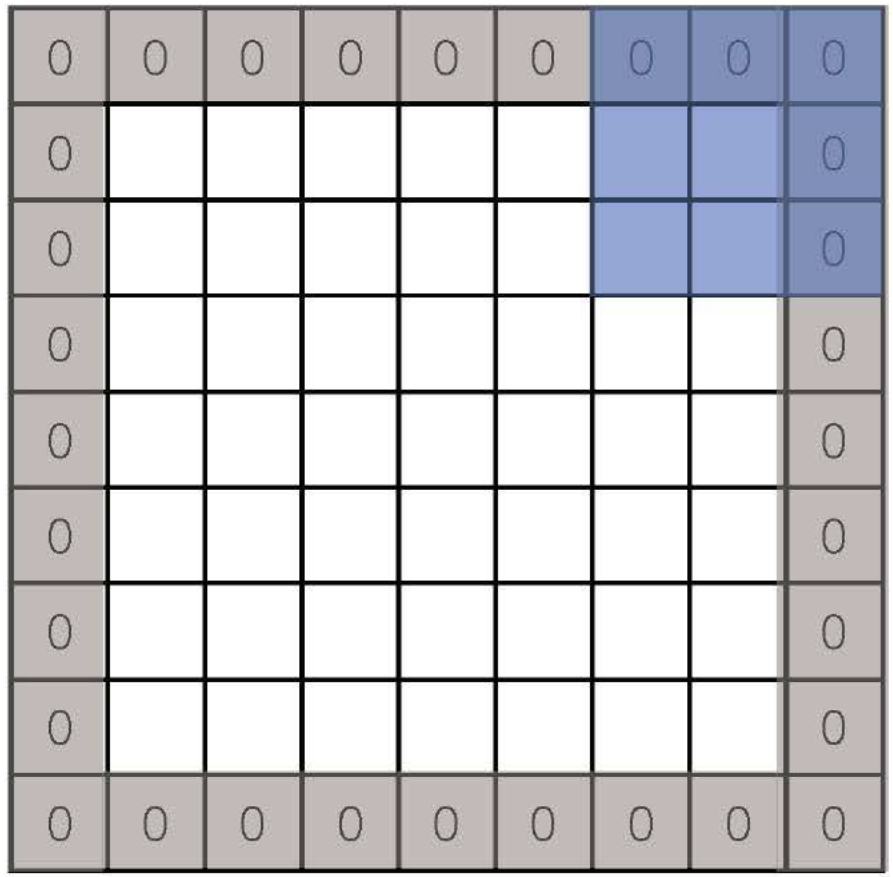
Input



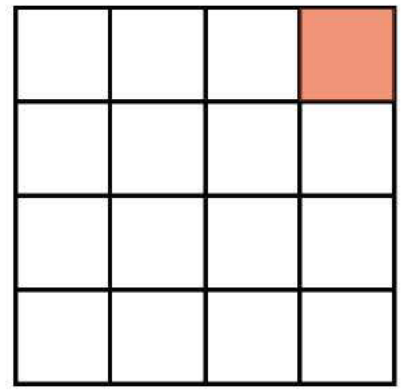
Output

Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



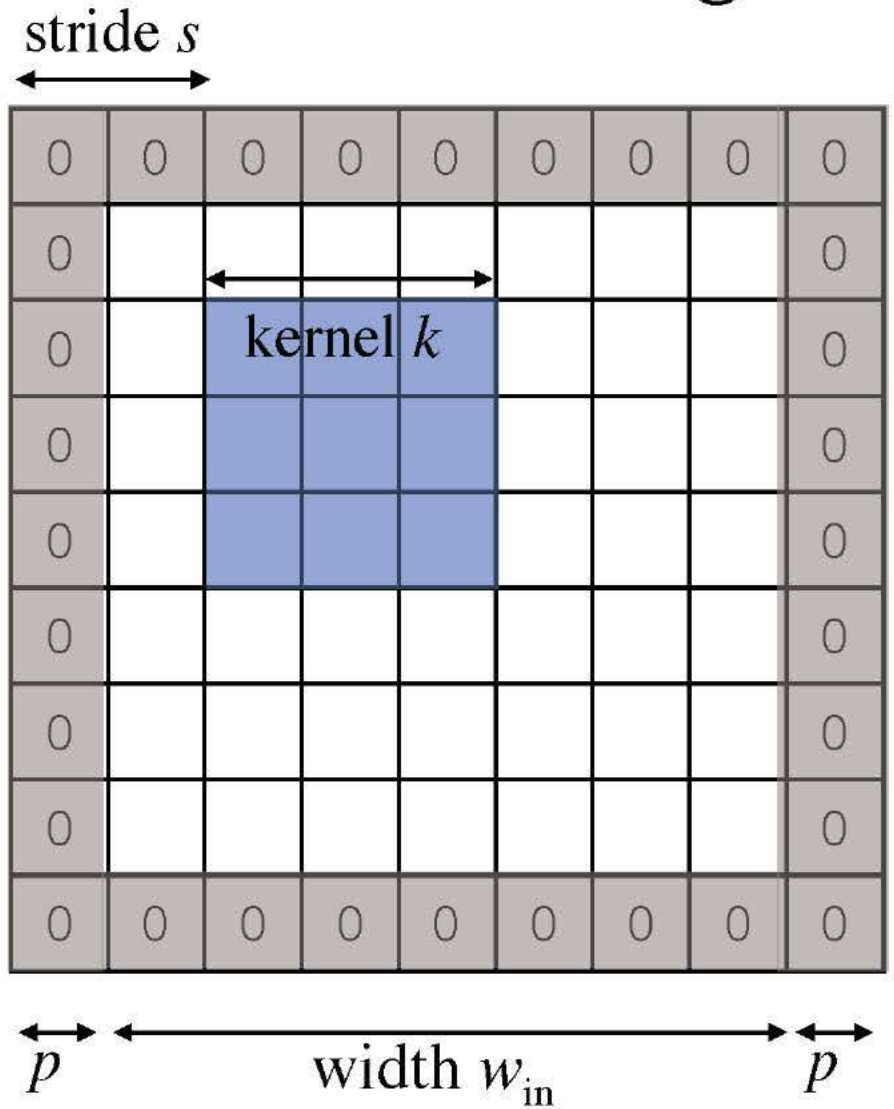
Input



Output

Convolution: Output

How big is the output?

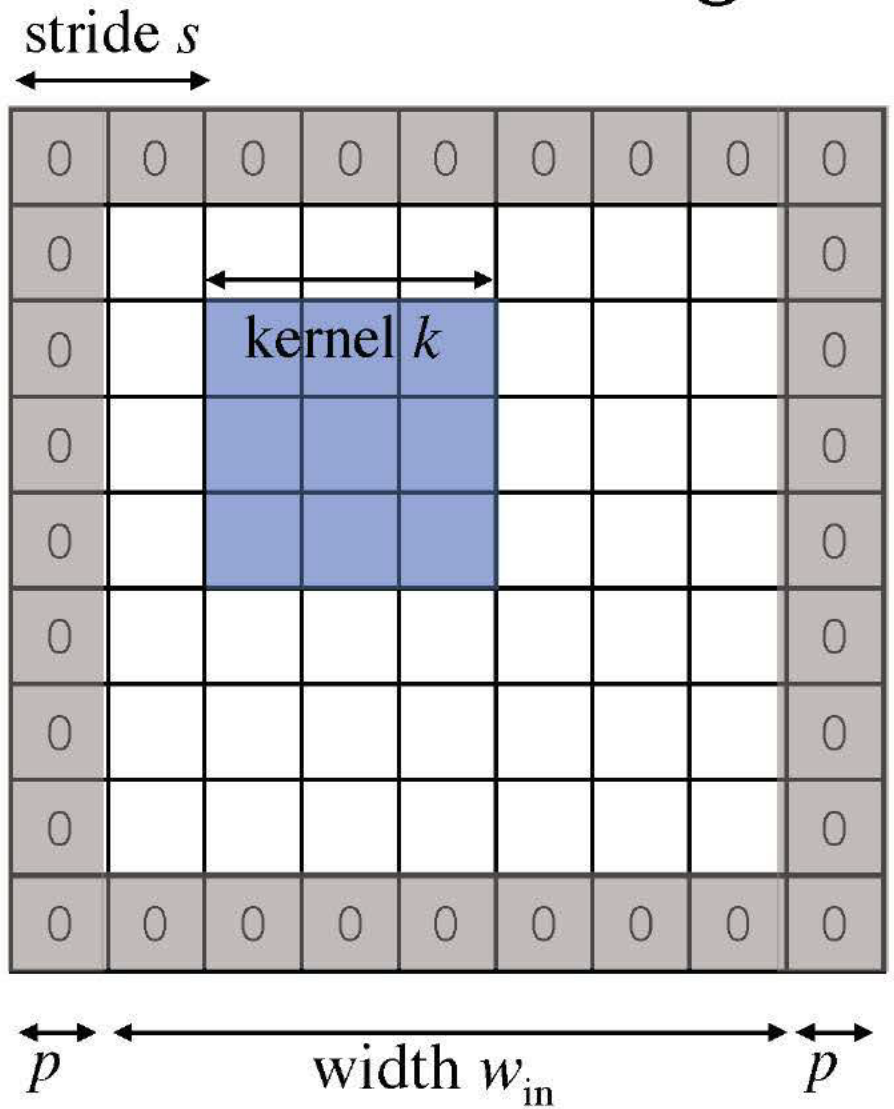


In general, the output has size:

$$w_{\text{out}} = \left\lfloor \frac{w_{\text{in}} + 2p - k}{s} \right\rfloor + 1$$

Convolution: Output

How big is the output?



Example: k=3, s=1, p=1

$$w_{out} = \left\lfloor \frac{w_{in} + 2p - k}{s} \right\rfloor + 1$$

$$= \left\lfloor \frac{w_{in} + 2 - 3}{1} \right\rfloor + 1$$

$$= w_{in}$$

VGGNet [Simonyan 2014] uses filters of this shape

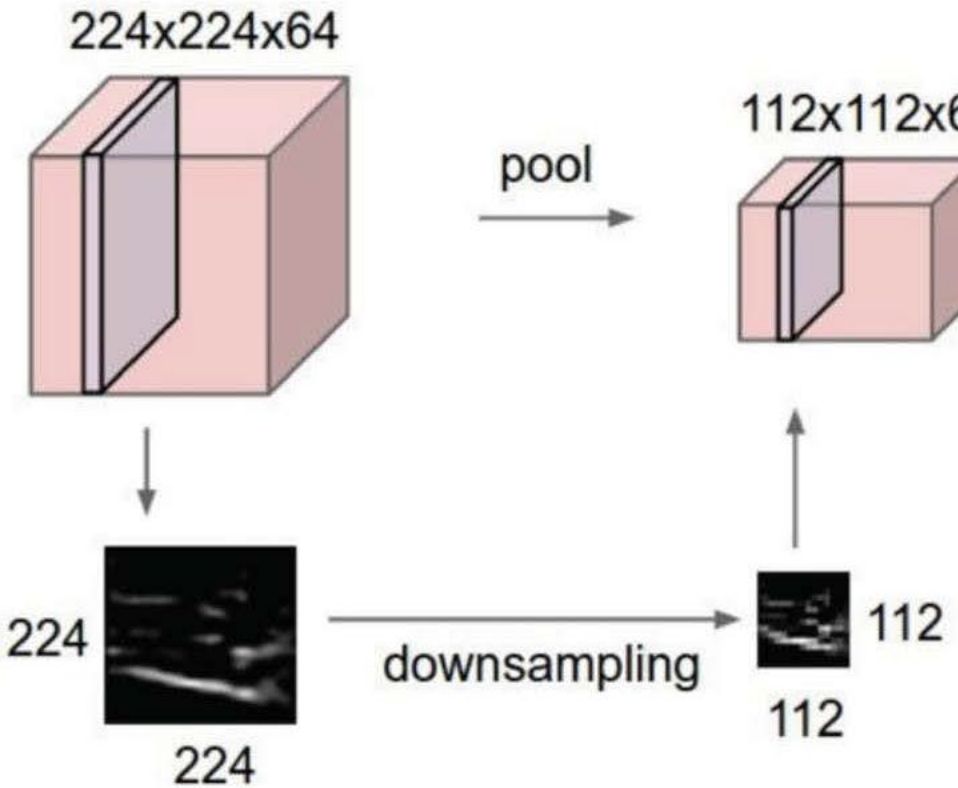
Pooling

For most ConvNets, **convolution** is often followed by **pooling**:

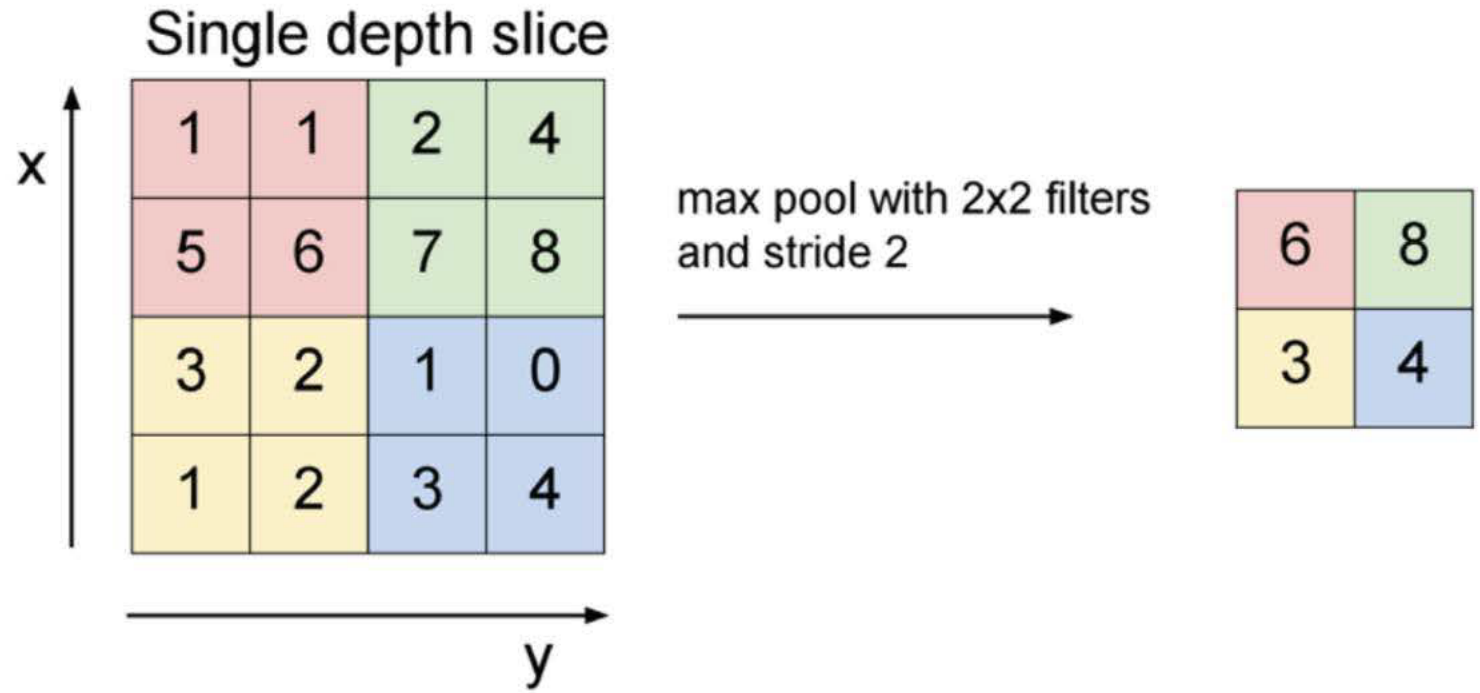
- Creates a smaller representation while retaining the most important information
- The “max” operation is the most common
- Why might “avg” be a poor choice?

Pooling

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



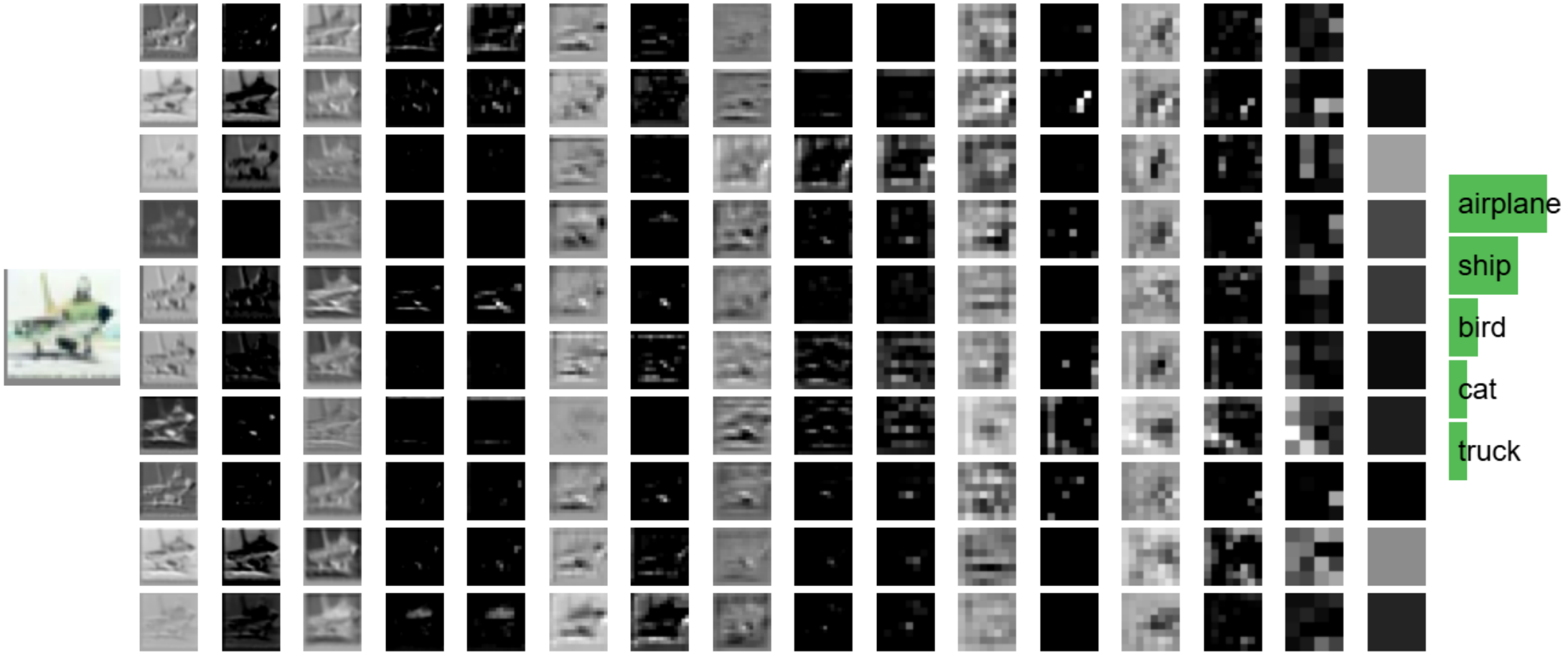
Max Pooling



What's the backprop rule for max pooling?

- In the forward pass, store the index that took the max
- The backprop gradient is the input gradient at that index

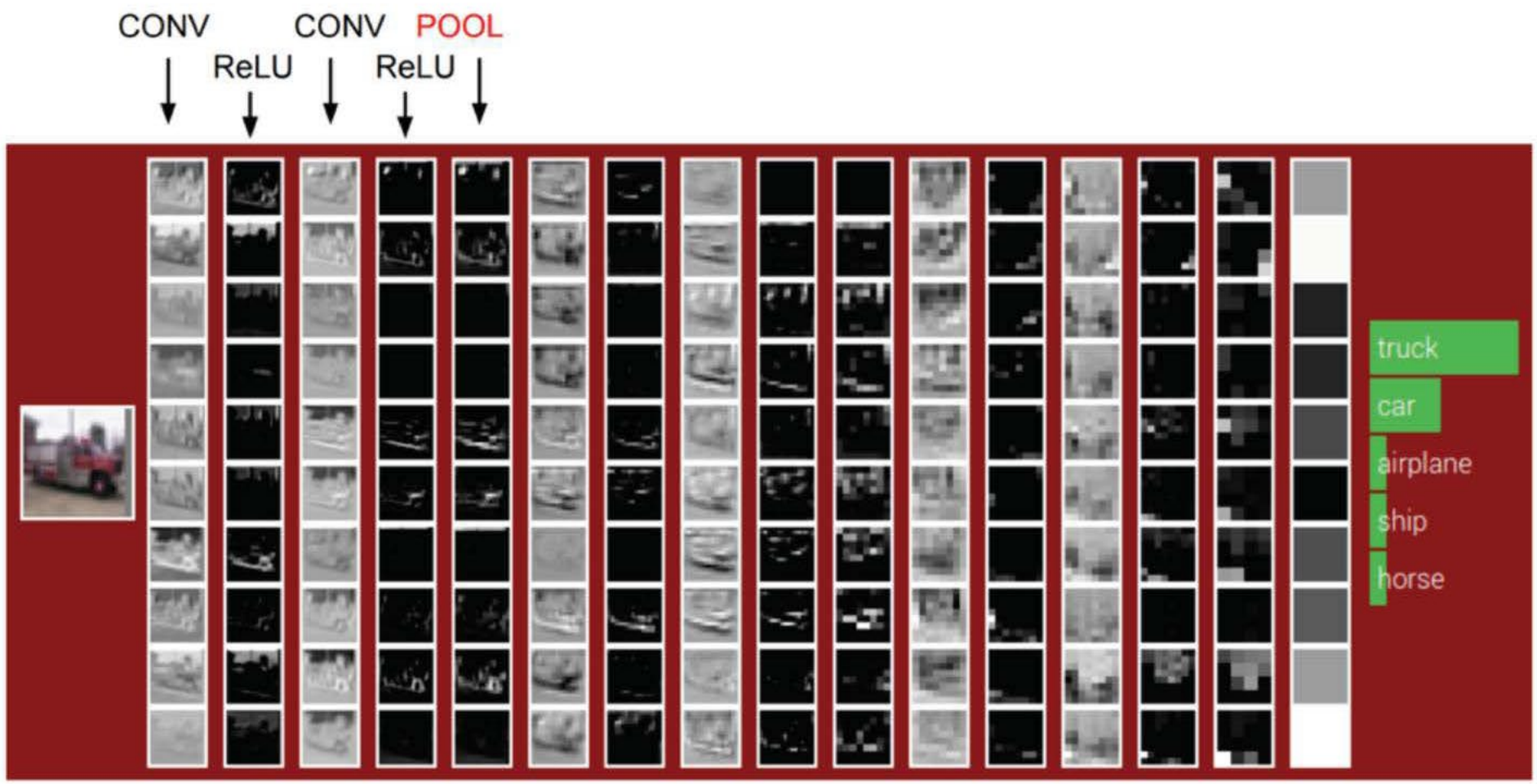
Demo



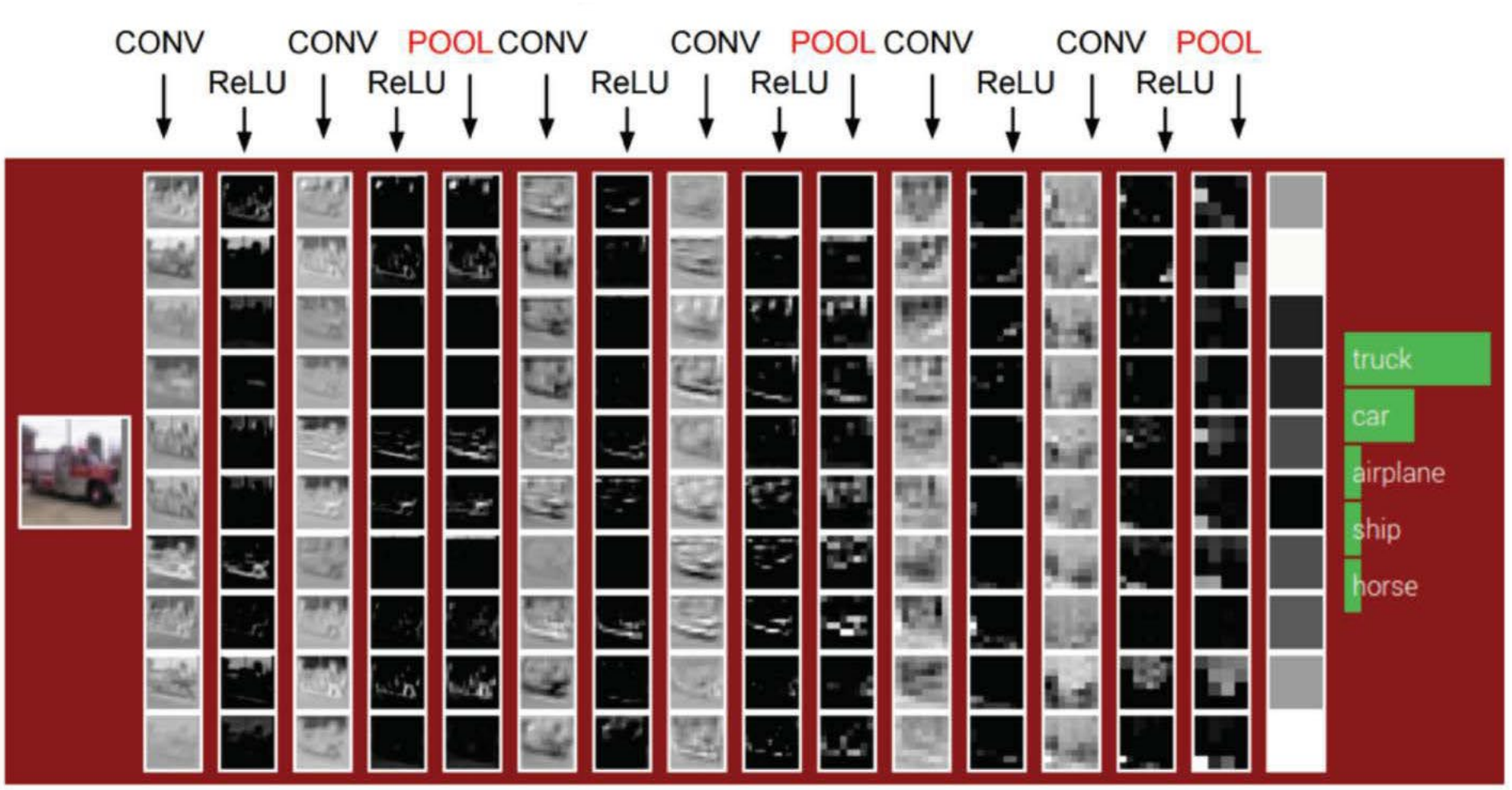
*This network is running live in your browser

http://cs231n.stanford.edu/

Example ConvNet



Example ConvNet



Example ConvNet

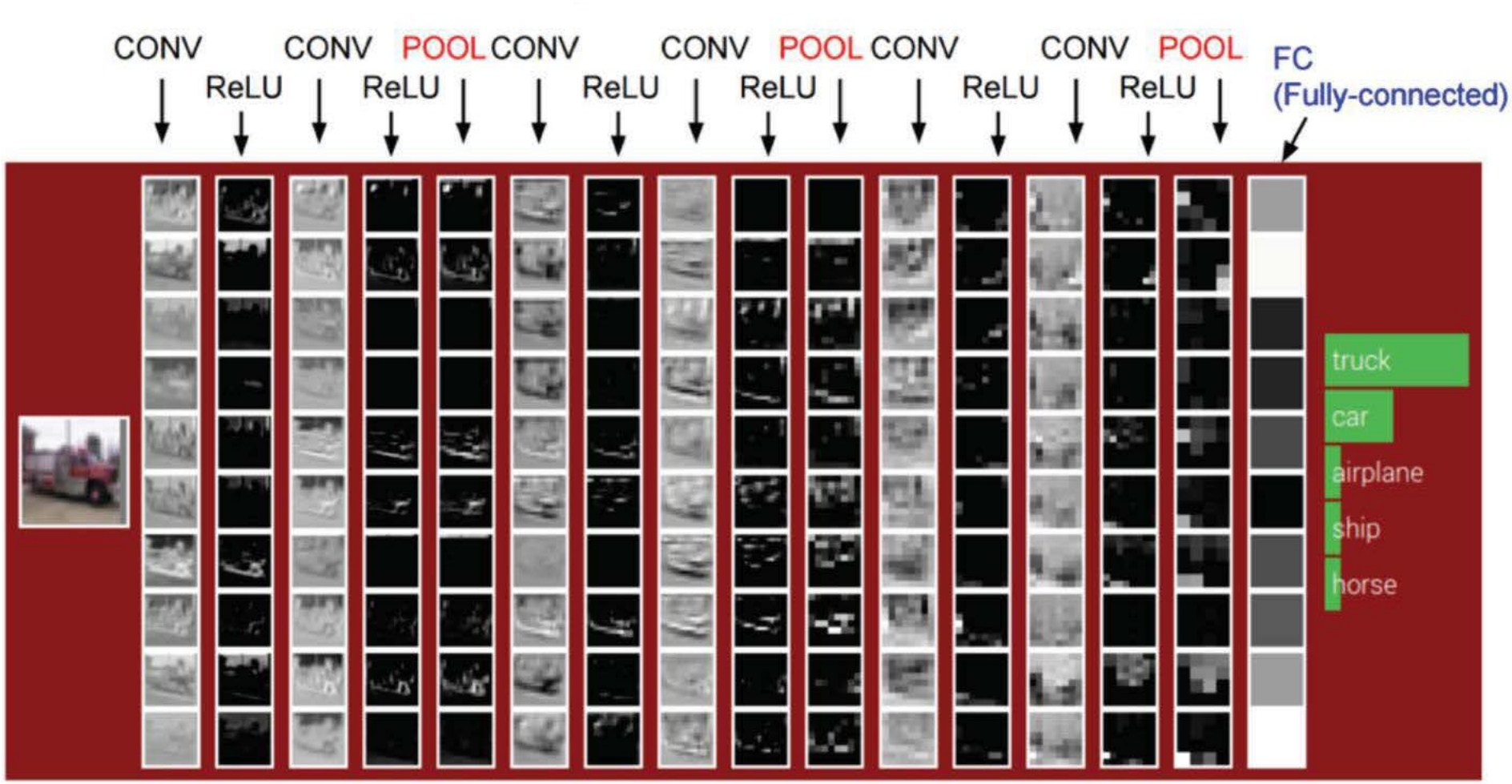
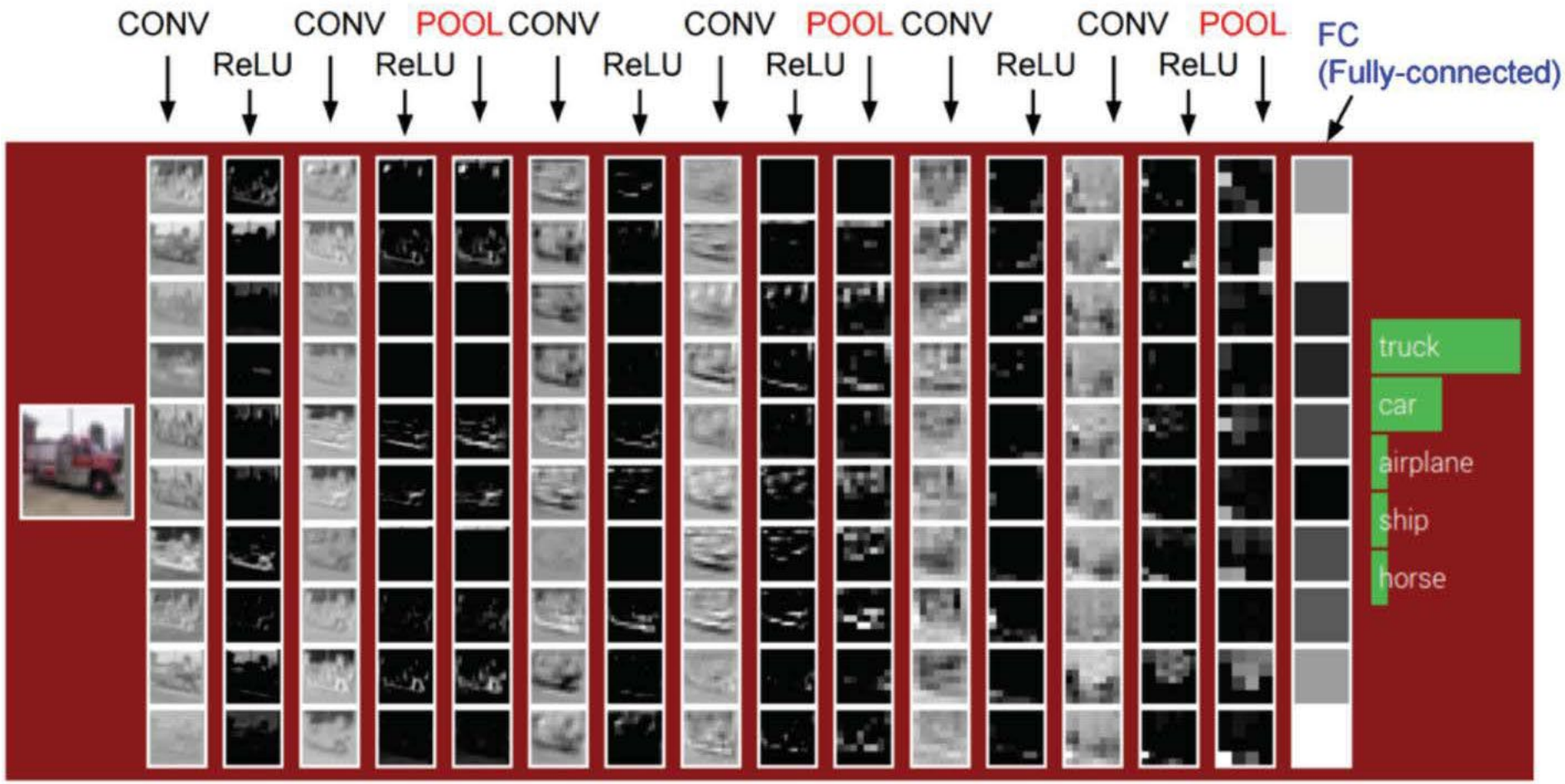


Figure: Andrej Karpathy

Example ConvNet



10x3x3 conv filters, stride 1, pad 1

2x2 pool filters, stride 2

CNNs in 2012: “SuperVision” (aka “AlexNet”)

“**AlexNet**” — Won the ILSVRC2012 Challenge

Major breakthrough: 15.3% Top-5 error on ILSVRC2012
(Next best: 25.7%)

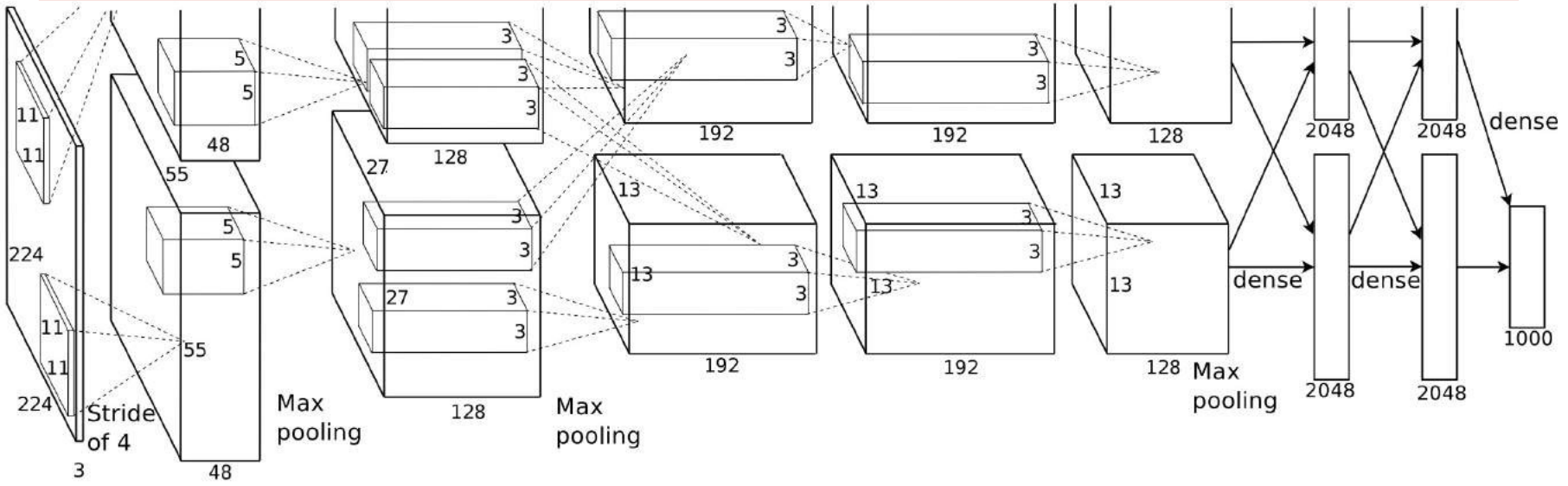
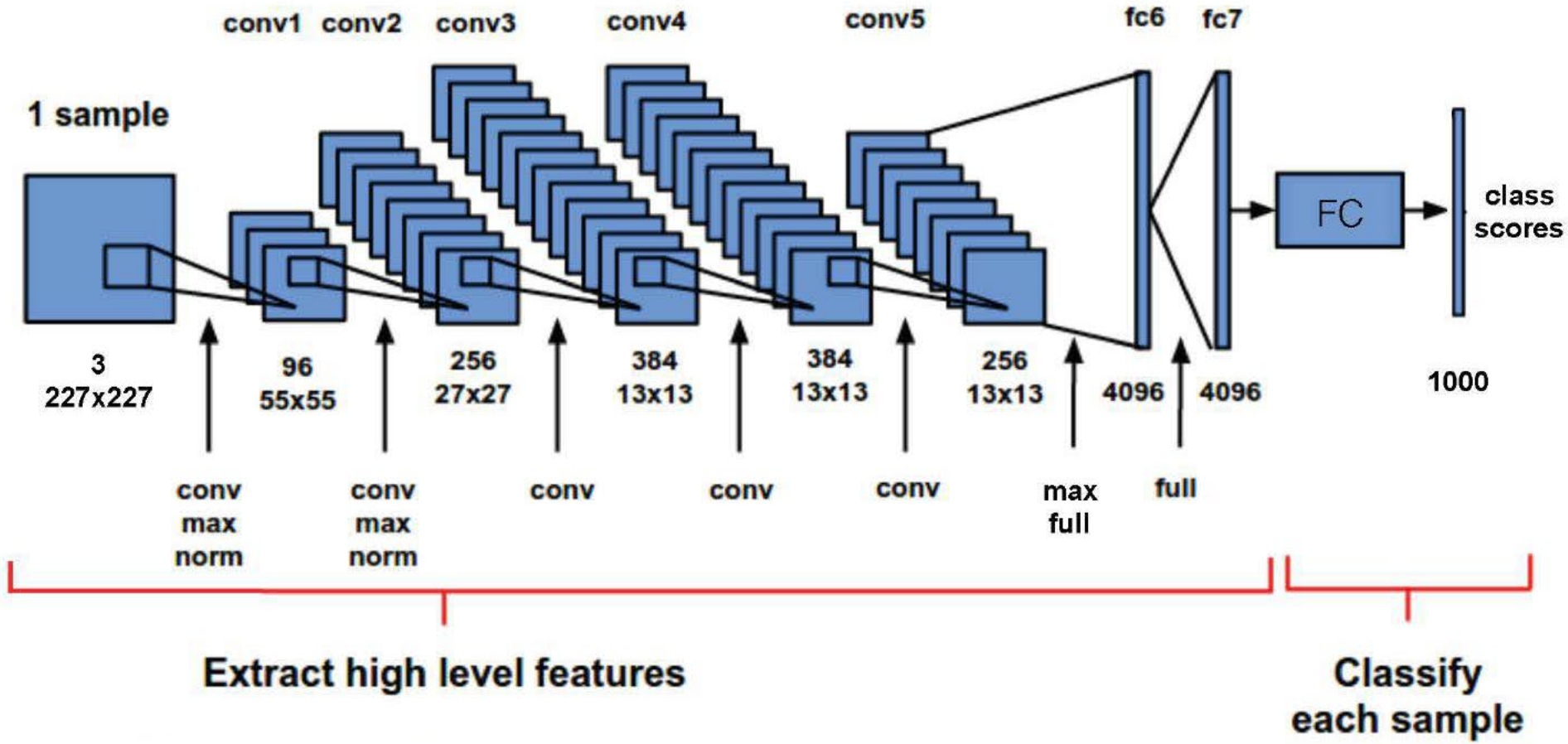


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input is 150,528-dimensional, and the number of neurons in the network’s remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

[Krizhevsky, Sutskever, Hinton. NIPS 2012]

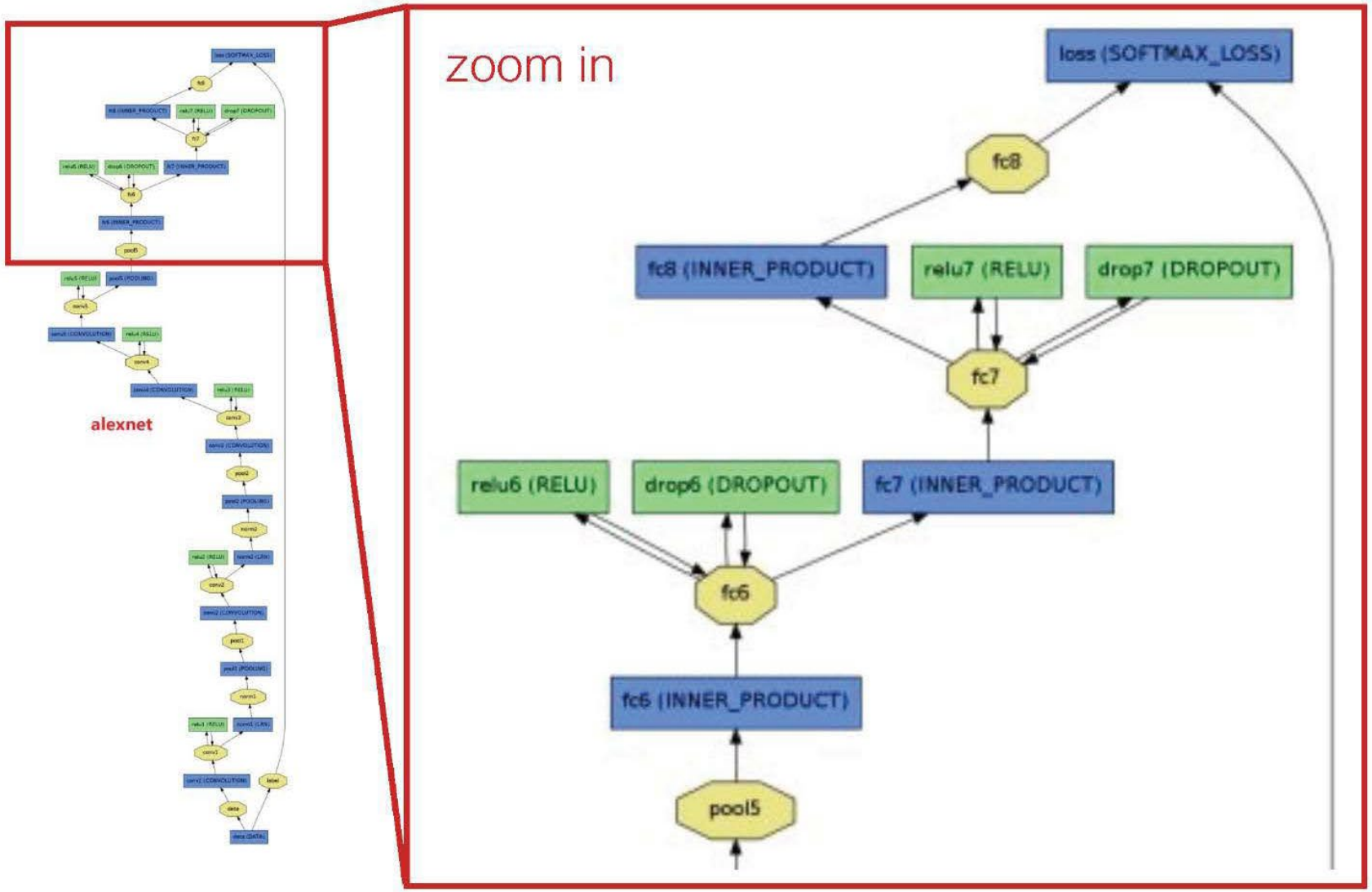
Example: AlexNet [Krizhevsky 2012]



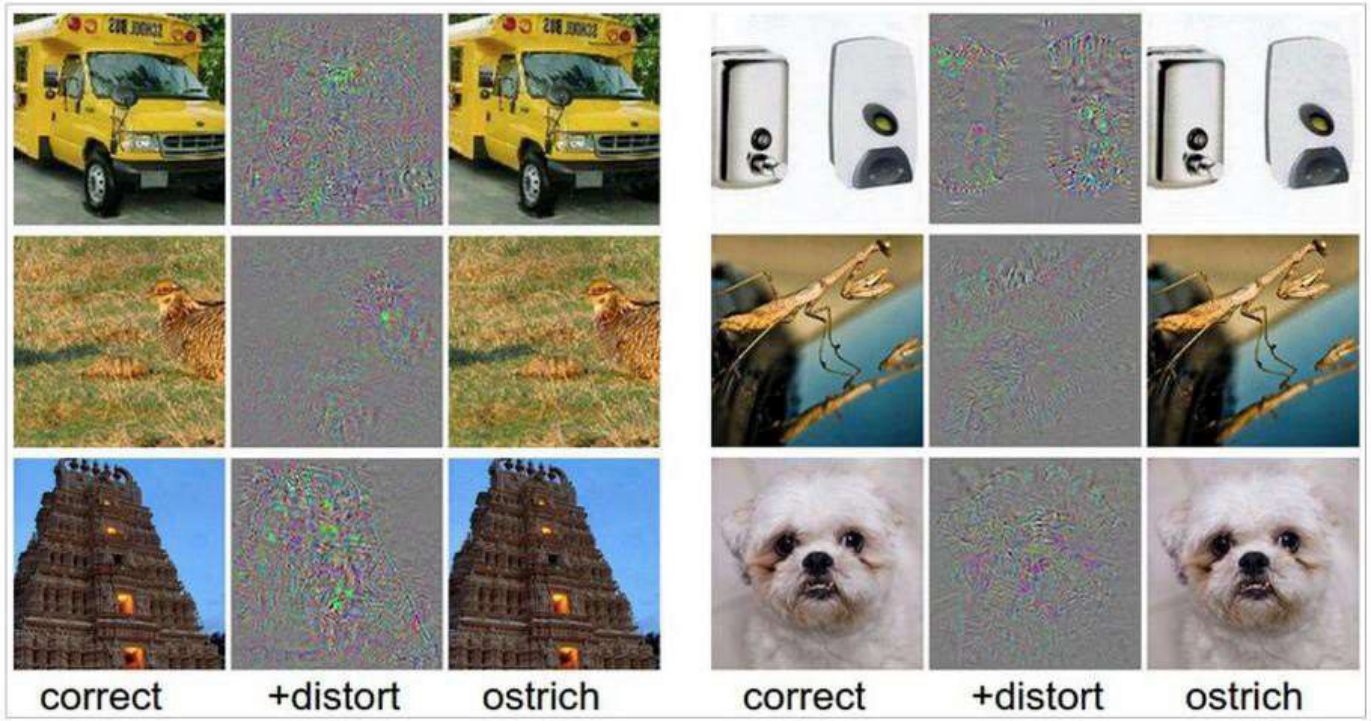
"max": max pooling
"norm": local response normalization
"full": fully connected

Figure: [Karnowski 2015] *(with corrections)*

Example: AlexNet [Krizhevsky 2012]



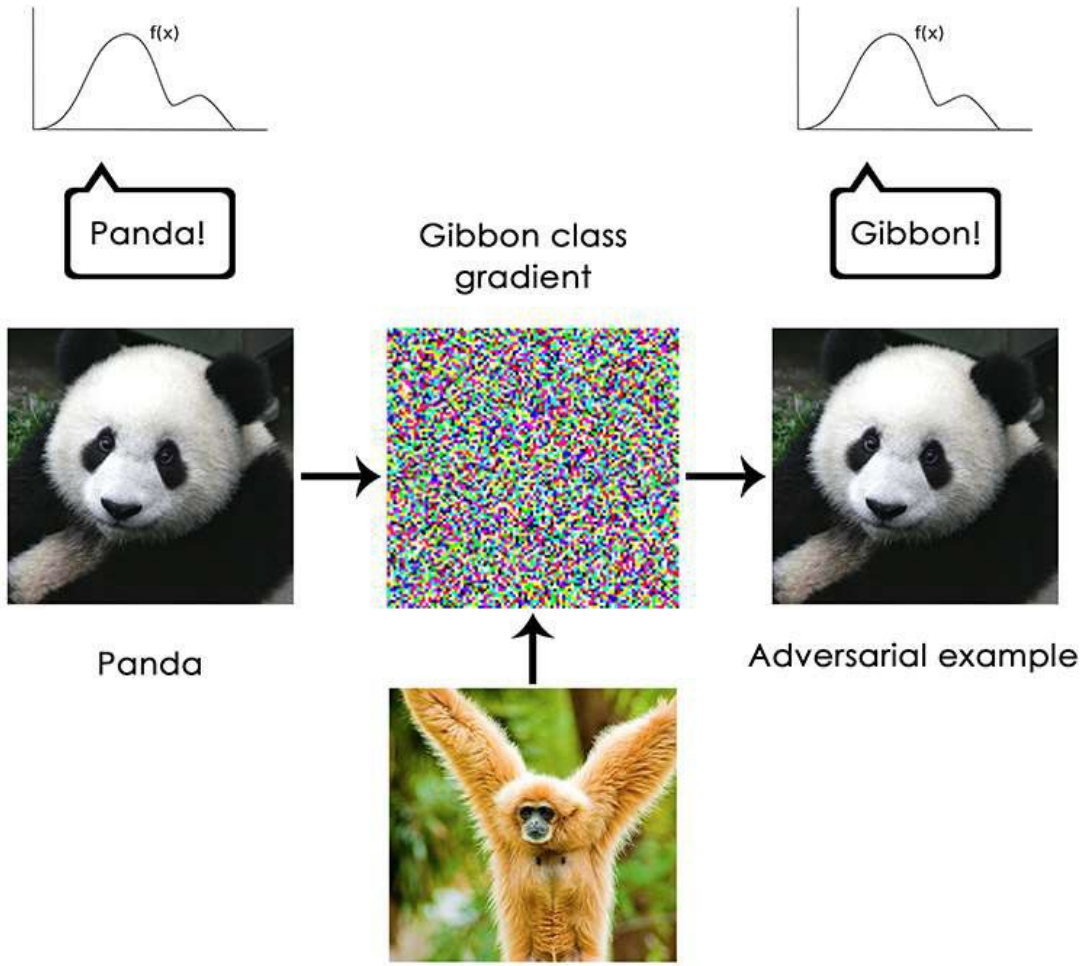
Breaking CNNs



Take a correctly classified image (left image in both columns), and add a tiny distortion (middle) to fool the ConvNet with the resulting image (right).

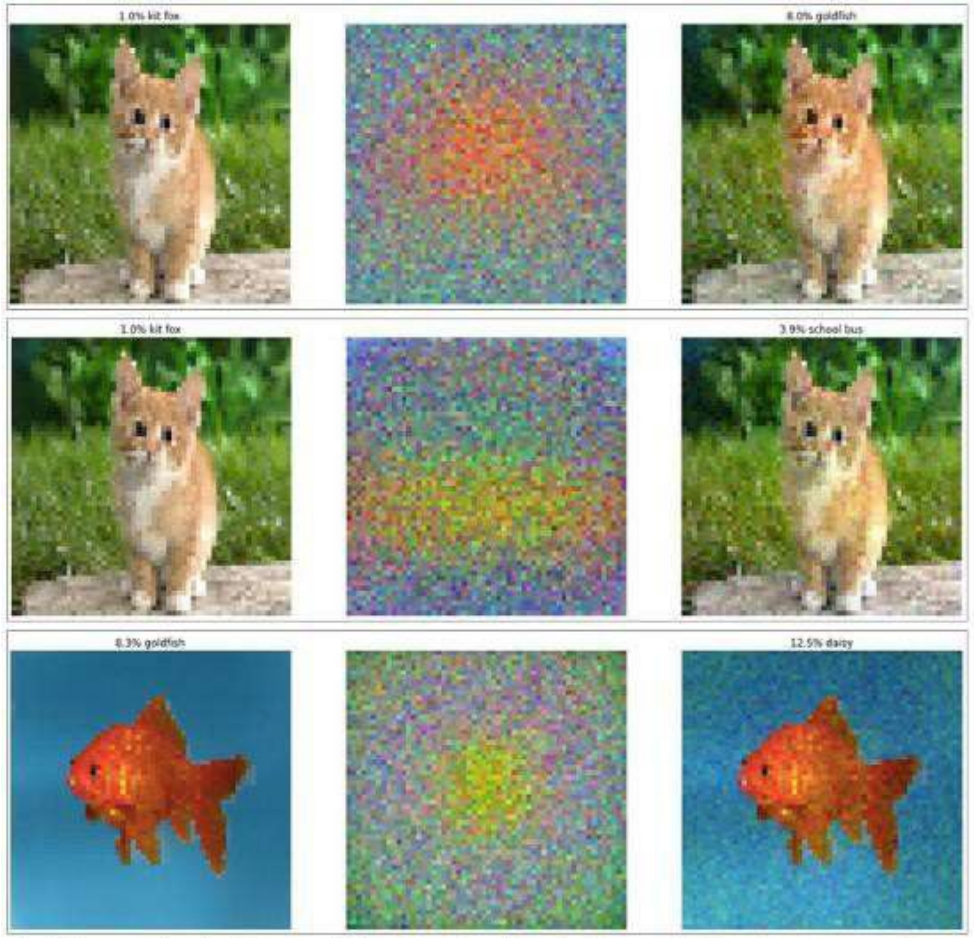
Intriguing properties of neural networks [Szegedy ICLR 2014].

Fooling a linear classifier



Francois Chollet - https://blog.keras.io/the-limitations-of-deep-learning.html

Fooling a linear classifier



Fooled linear classifier: The starting image (left) is classified as a kit fox. That's incorrect, but then what can you expect from a linear classifier? However, if we add a small amount "goldfish" weights to the image (top row, middle), suddenly the classifier is convinced that it's looking at one with high confidence. We can distort it with the school bus template instead if we wanted to.

http://karpathy.github.io/2015/03/30/breaking-convnets/

Perceptron weight update: add a small multiple of the example to the weight vector:
$w \leftarrow w + \alpha x$

To fool a linear classifier, add a small multiple of the weight vector to the training example:
$x \leftarrow x + \alpha w$

Adversarial Patches

- https://www.theverge.com/2018/1/3/16844842/ai-computer-vision-trick-adversarial-patches-google
- https://arxiv.org/pdf/1712.09665.pdf

[Brown et al., Google, 2018]

BBC | Your account | News | Sport | Weather | Shop | Earth | Travel | More | Search

NEWS

Home | Video | World | US & Canada | UK | Business | Tech | Science | Stories | Entertainment & Arts | Health | More

Technology

Single pixel change fools AI programs

Tiny changes can make image recognition systems think a school bus is an ostrich, find scientists.

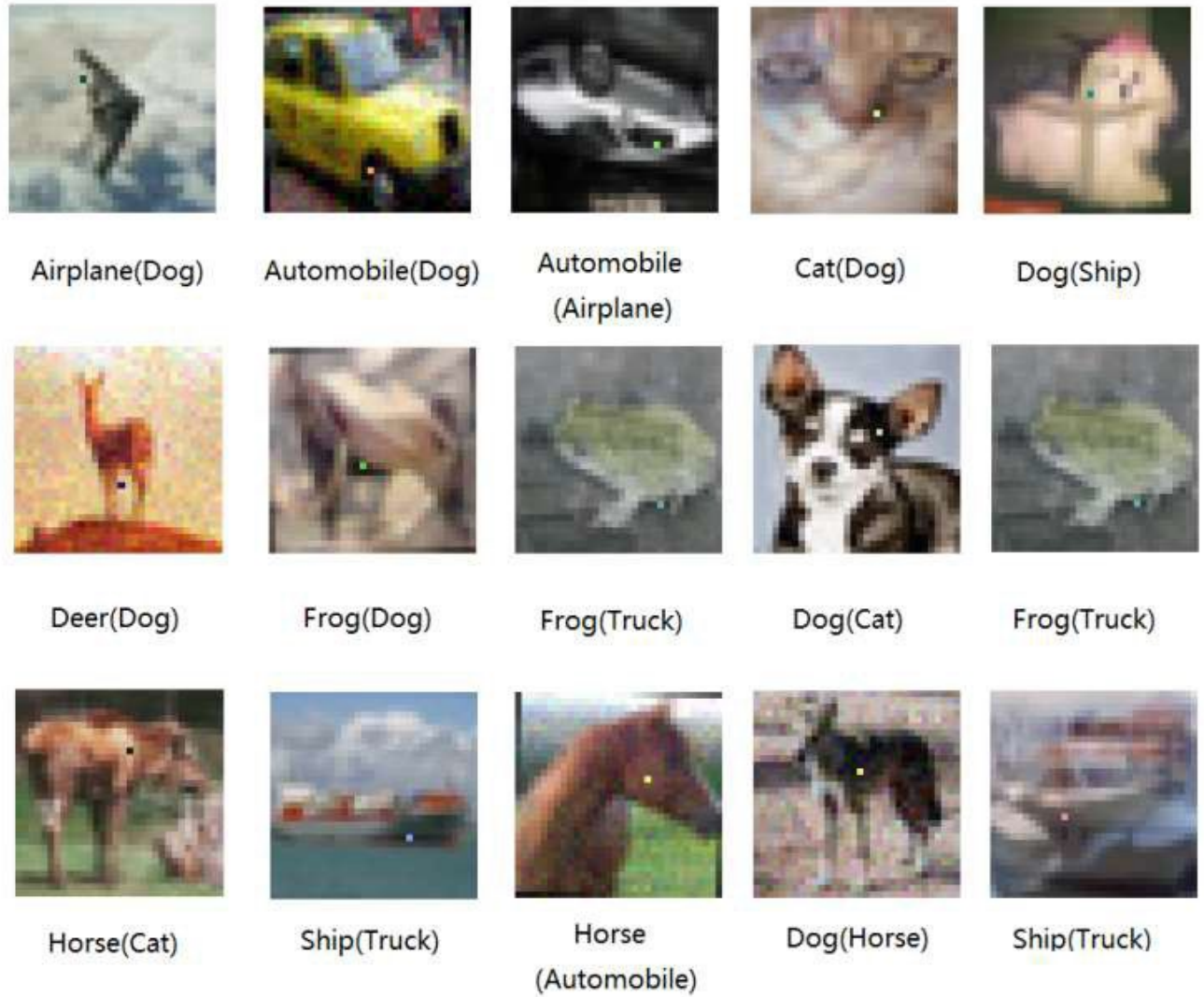
3 hours ago | Technology

Algorithm learns to recognise natural beauty

Artificial intelligence fools security

AI used to detect breast cancer

3rd November 2017



Su et al., One pixel attack for fooling deep neural networks
https://arxiv.org/abs/1710.08864

SYNTHESIZING ROBUST ADVERSARIAL EXAMPLES

Anish Athalye*[1,2], Logan Engstrom*[1,2], Andrew Ilyas*[1,2], Kevin Kwok[2]
[1]Massachusetts Institute of Technology, [2]LabSix
{aathalye,engstrom,ailyas}@mit.edu, kevin@labsix.org

https://arxiv.org/pdf/1707.07397.pdf