CSCI/EENG/ROBO 437/507/517: Introduction to Computer Vision

Kaveh Fathian
Assistant Professor
Computer Science Department
Colorado School of Mines

Lecture 31

- Attendance monitored via iClicker today
- **Homework 4** posted; due on Wed 11/20
- **Quiz 4** on Fri 11/22
- Bonus **project** & **competition** problem discussion

REVIEW

Fundamental Equations of Computer Vision

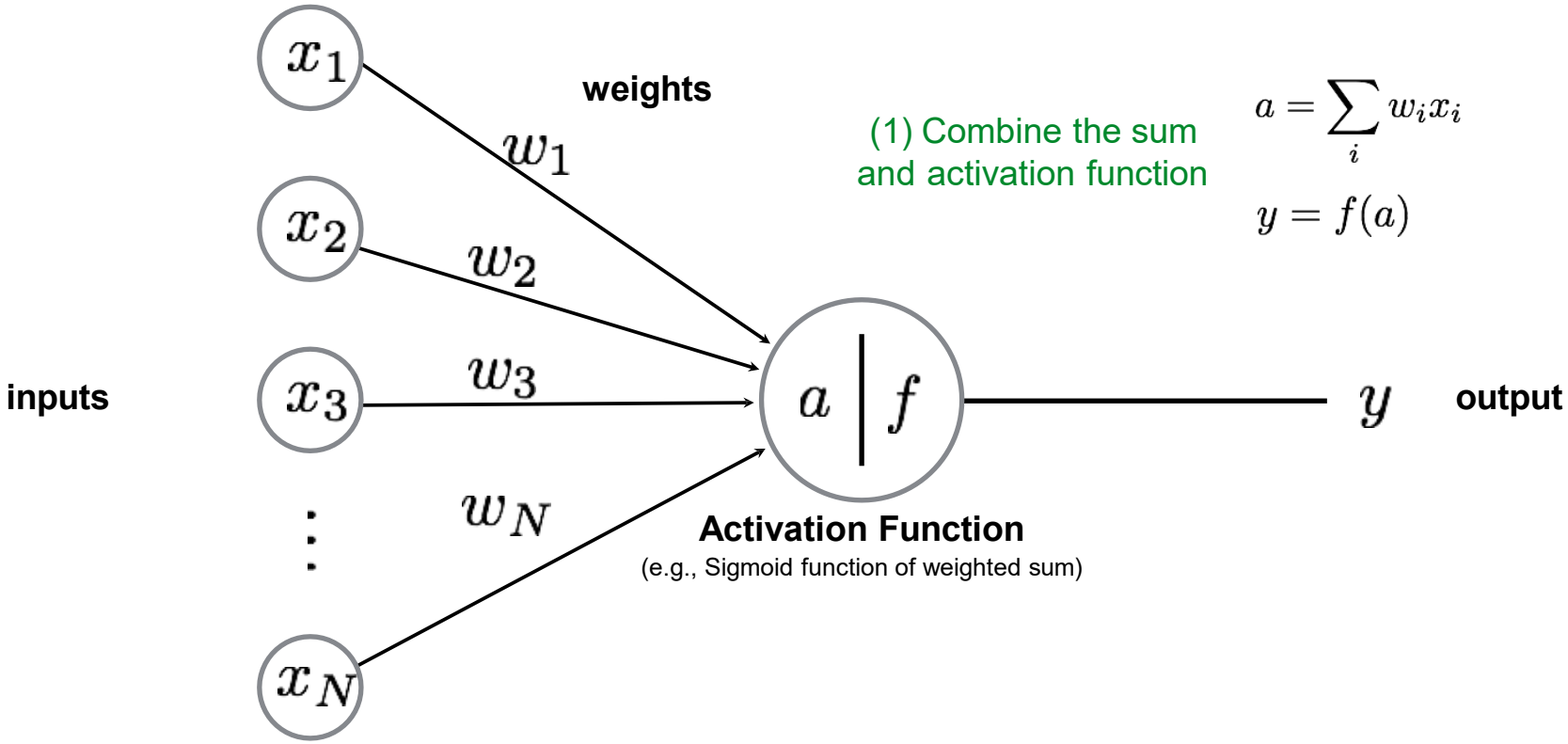
1. Image Filtering $$h[m,n] = \sum_{k,l} f[k,l]\, I[m+k, n+l]$$

2. Camera Geometry $$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{X} \qquad x^{\mathrm{T}} F x = 0$$

3. Machine Learning $$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2. \qquad y = \varphi(\sum_{i=1}^{n} w_i x_i + b) = \varphi(\mathbf{w}^T \mathbf{x} + b)$$

4. Optical Flow $$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$$

The Perceptron



(1) Combine the sum and activation function

$$a = \sum_i w_i x_i$$

$$y = f(a)$$

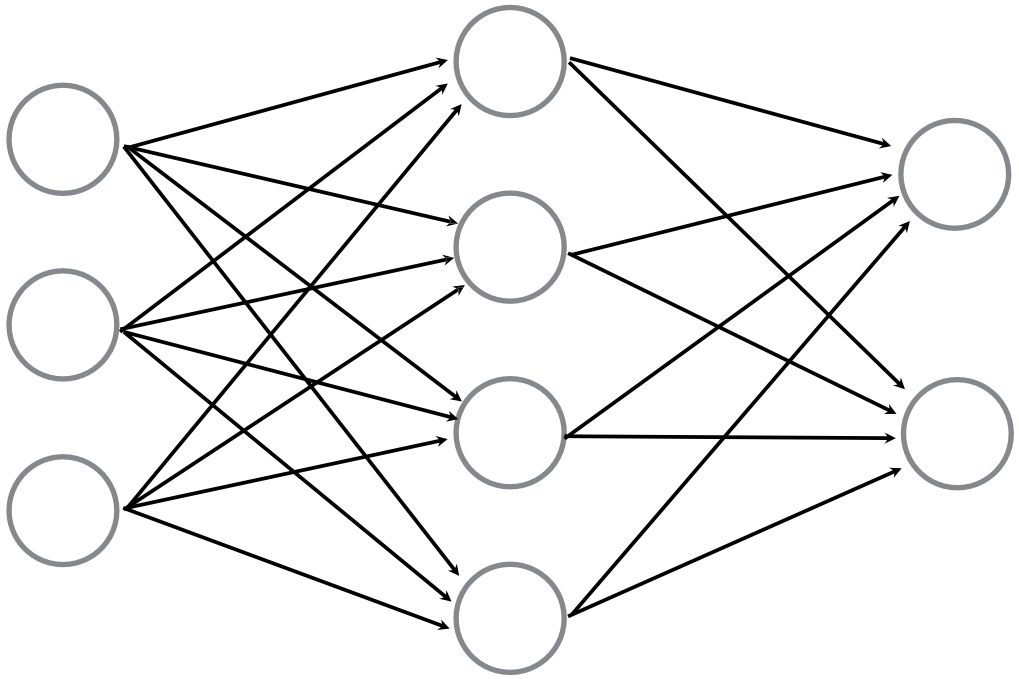
(2) suppress the bias term (less clutter)

$$x_N = 1$$

$$w_N = b$$

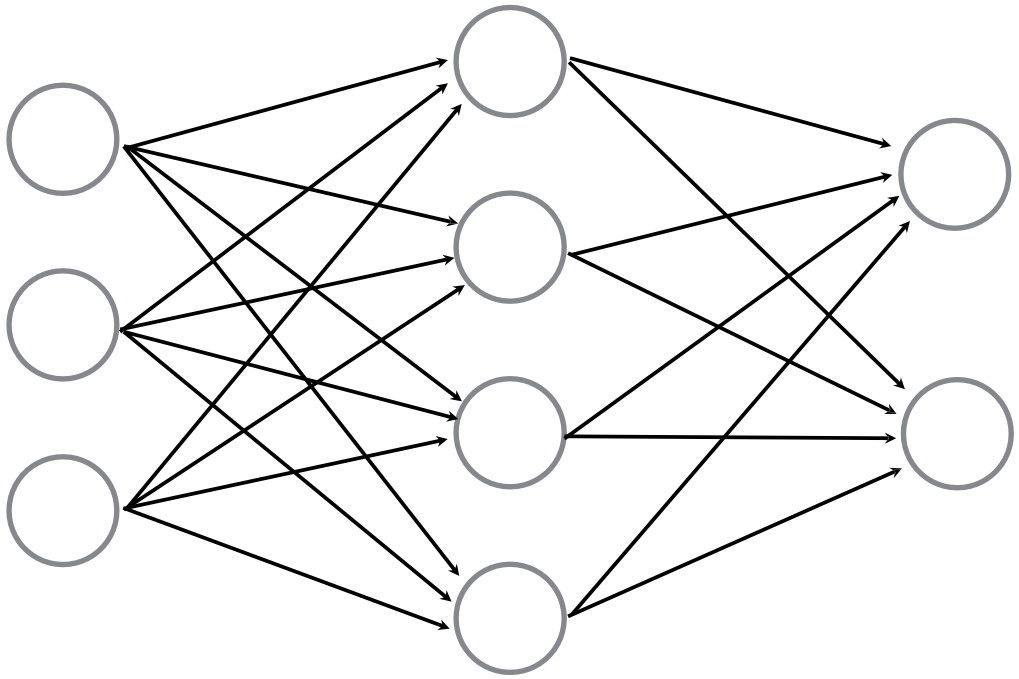
Neural Network

a collection of connected perceptrons



Neural Network

performance usually tops out at 2-3 layers, deeper networks don’t really improve performance...



...with the exception of **convolutional** networks for images

Stochastic Gradient Descent

- For each example sample $\{x_i, y_i\}$

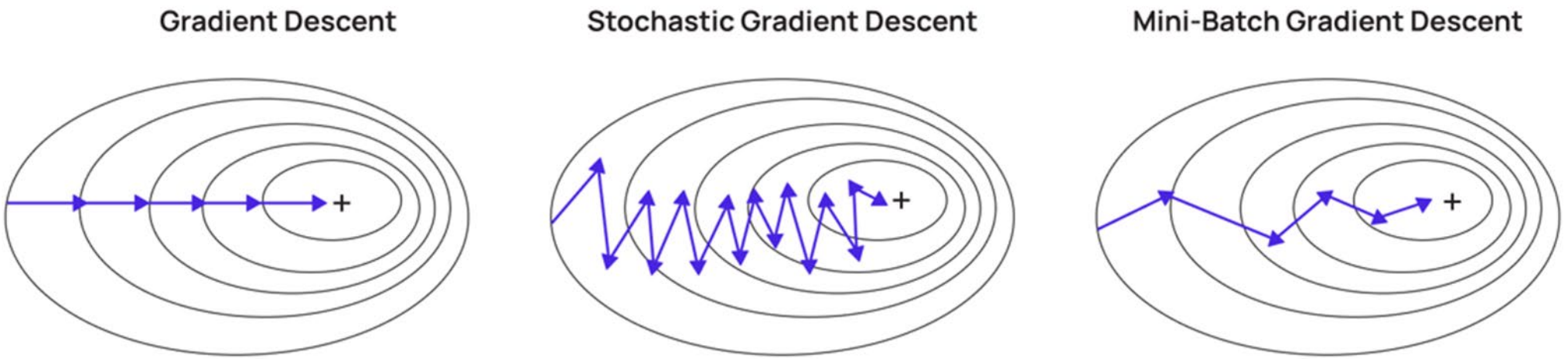
1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss $\mathcal{L}_i$
2. Update
   a. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$

vector of parameter partial derivatives

   b. Gradient update $\theta \leftarrow \theta + \eta \frac{\partial \mathcal{L}}{\partial \theta}$

vector of parameter update equations

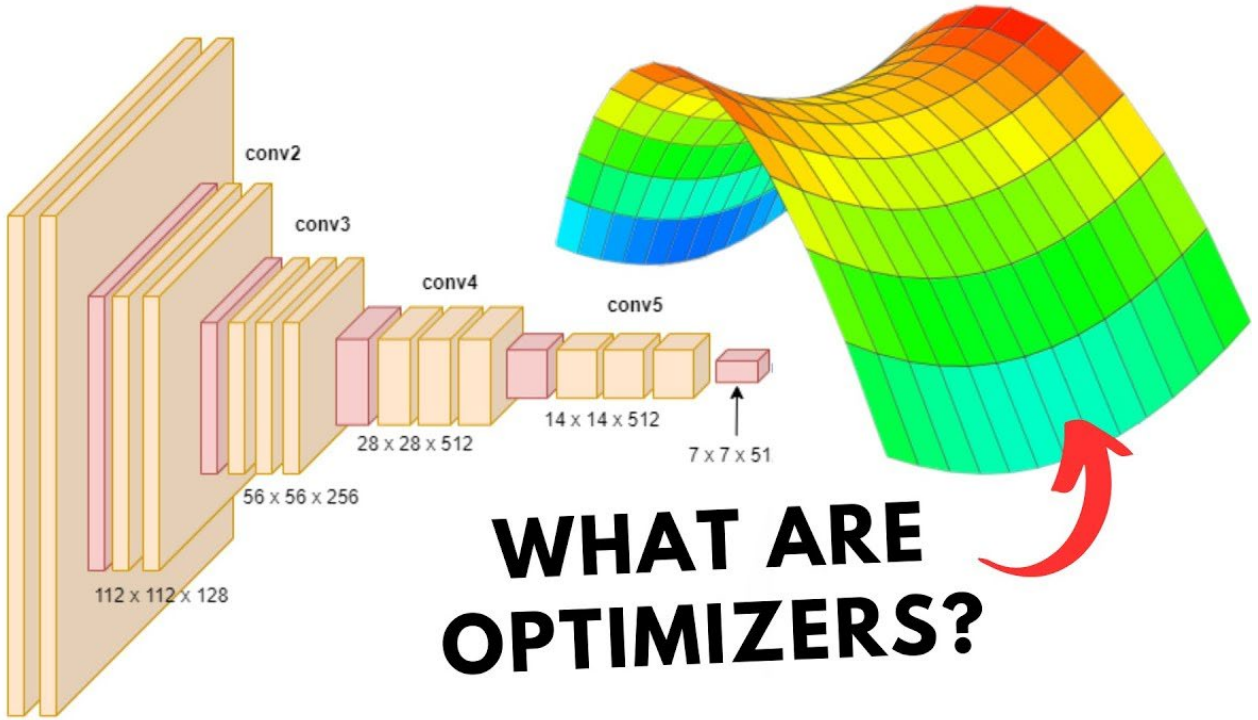
Summary



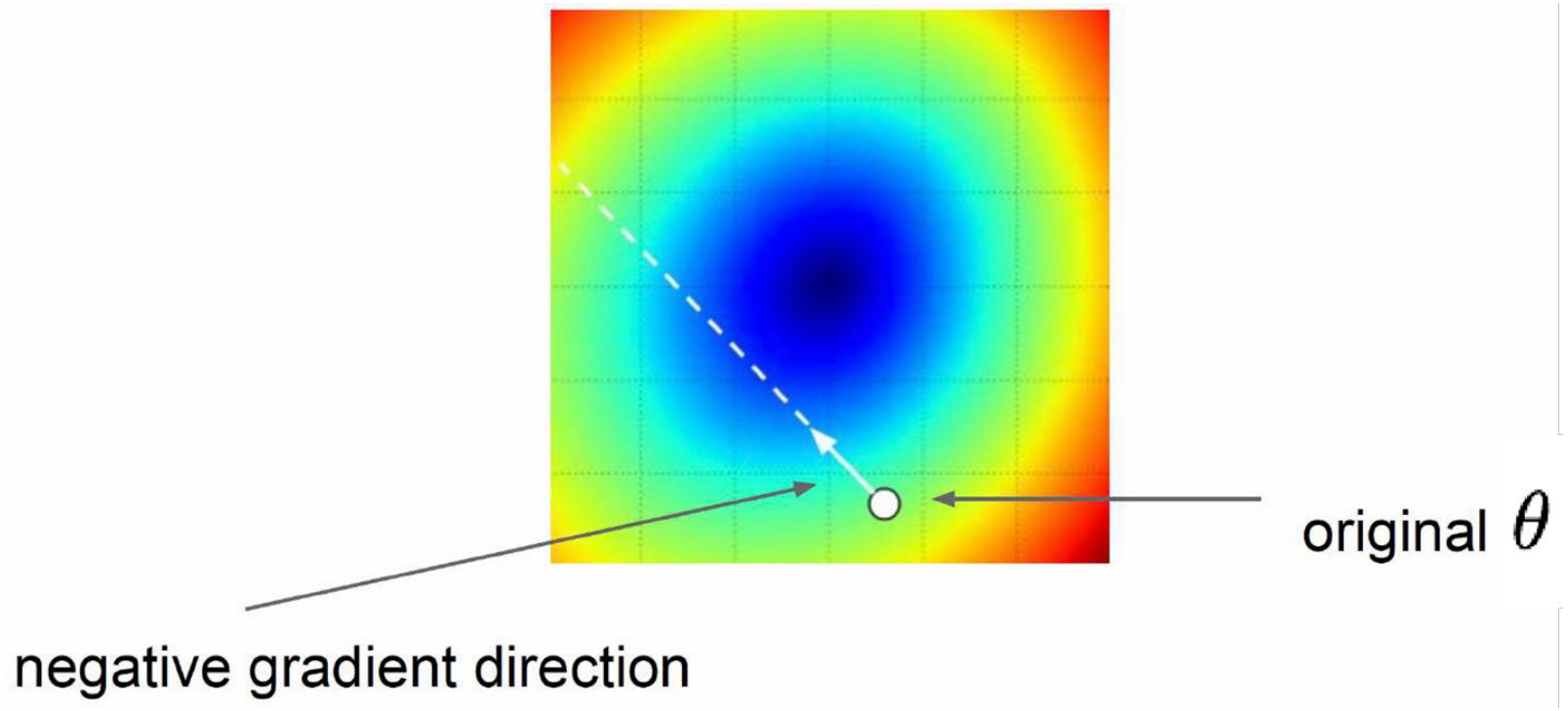
- Always use mini-batch gradient descent
- Incorrectly refer to it as “doing SGD” as everyone else
  (or call it batch gradient descent)
- The mini-batch size is a hyperparameter, but it is not very common to cross-validate over it (usually based on practical concerns, e.g. space/time efficiency)

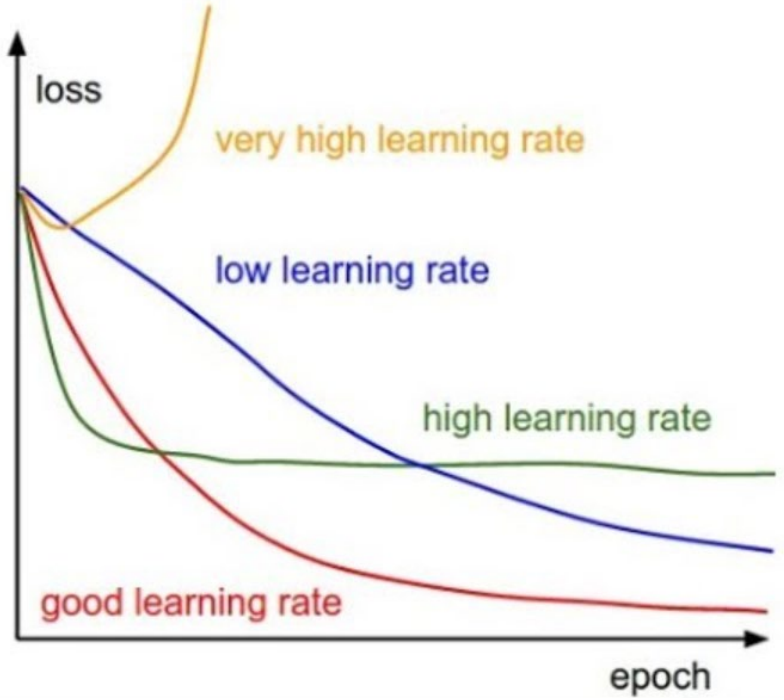
This Lecture

Notes on Optimization



Learning rates





$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$

Step size: learning rate

Too big: will miss the minimum

Too small: slow convergence

Learning rate scheduling

- Use different **learning rate** at each iteration
- Most common choice:

$$\eta_t = \frac{\eta_0}{\sqrt{t}}$$

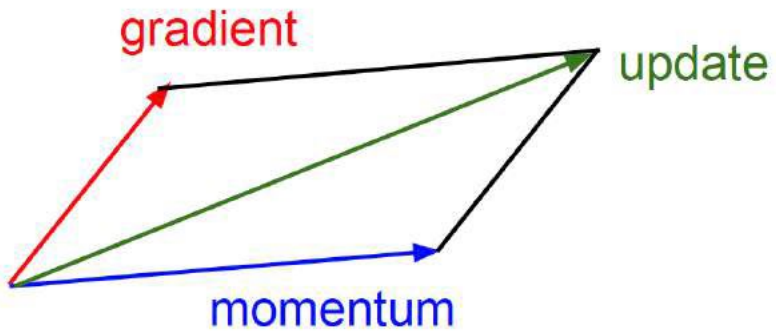
- Need to select initial learning rate η_0, important!!!
- More modern choice: **Adaptive** learning rates

$$\eta_t = G\left(\left\{\frac{\partial L}{\partial \theta}\right\}_{i=0}^{t}\right)$$

- Many choices for G (**Adam**, **Adagrad**, **Adadelta**)

Momentum Update

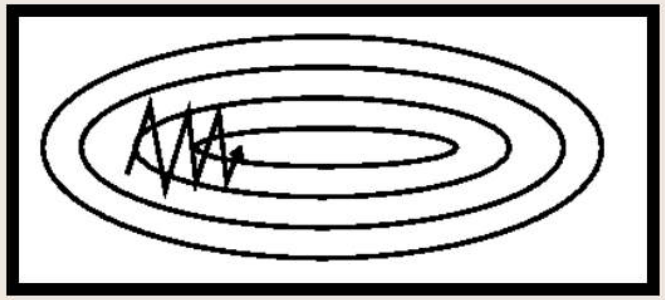
$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$



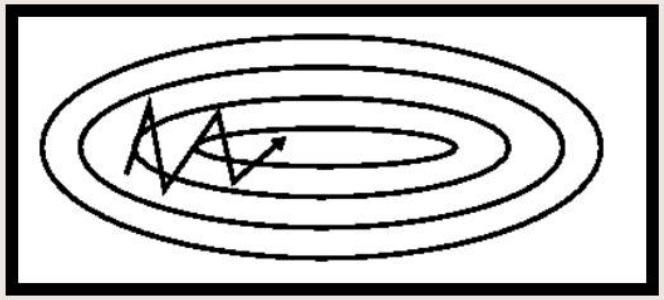
$$\Delta\theta \leftarrow w \frac{\partial L}{\partial \theta} + (1 - w)\Delta\theta$$

Take direction history into account!

```
weights_grad = evaluate_gradient(loss_fun, data, weights)
vel = vel * 0.9 - step_size * weights_grad
weights += vel
```



(Fig. 2a)



(Fig. 2b)

Many other ways to perform optimization…

- Second order methods that use the Hessian (or its approximation): BFGS, **LBFGS**, etc.

- Currently, the lesson from the trenches is that well-tuned SGD+Momentum is very hard to beat for CNNs.

- No consensus on Adam etc.: Seem to give faster performance to worse local minima

Derivatives

- Given f(x), where x is vector of inputs
  - Compute gradient of f at x: $\nabla f(x)$

How do we do differentiation?

Numerical differentiation

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$f(x+h) = f(x) + h\frac{df(x)}{dx}$$

Numerical differentiation

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$f(x+h) = f(x) + h\frac{df(x)}{dx}$$

Numerical differentiation is:

- Approximate
- Slow
- Numerically unstable
- Easy to write

Symbolic differentiation

- What Mathematica does: Automatically derive *analytical* expressions for derivative.

Symbolic differentiation

- What Mathematica does: Automatically derive *analytical* expressions for derivative.
- Often results in very redundant (and expensive to evaluate) expressions.

```
In[11]:= D[Log[1 + Exp[w * x + b]], w]
```

$$\text{Out[11]=} \quad \frac{e^{b+w\,x}\,w}{1+e^{b+w\,x}}$$

```
In[19]:= D[Log[1 + Exp[w2 * Log[1 + Exp[w1 * x + b1]] + b2]], w1]
```

$$\text{Out[19]=} \quad \frac{e^{b1+b2+w1\,x+w2\,\text{Log}\left[1+e^{b1+w1\,x}\right]}\,w2\,x}{\left(1+e^{b1+w1\,x}\right)\left(1+e^{b2+w2\,\text{Log}\left[1+e^{b1+w1\,x}\right]}\right)}$$

- Often intractable.

Automatic differentiation (autodiff)

- An autodiff system will convert the program into a sequence of primitive operations which have specified routines for computing derivatives.
- In this representation, backprop can be done in a completely mechanical way.

Original program:

$$z = wx + b$$
$$y = \frac{1}{1 + \exp(-z)}$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

Sequence of primitive operations:

$$t_1 = wx$$
$$z = t_1 + b$$
$$t_3 = -z$$
$$t_4 = \exp(t_3)$$
$$t_5 = 1 + t_4$$
$$y = 1/t_5$$
$$t_6 = y - t$$
$$t_7 = t_6^2$$
$$\mathcal{L} = t_7/2$$

In summary

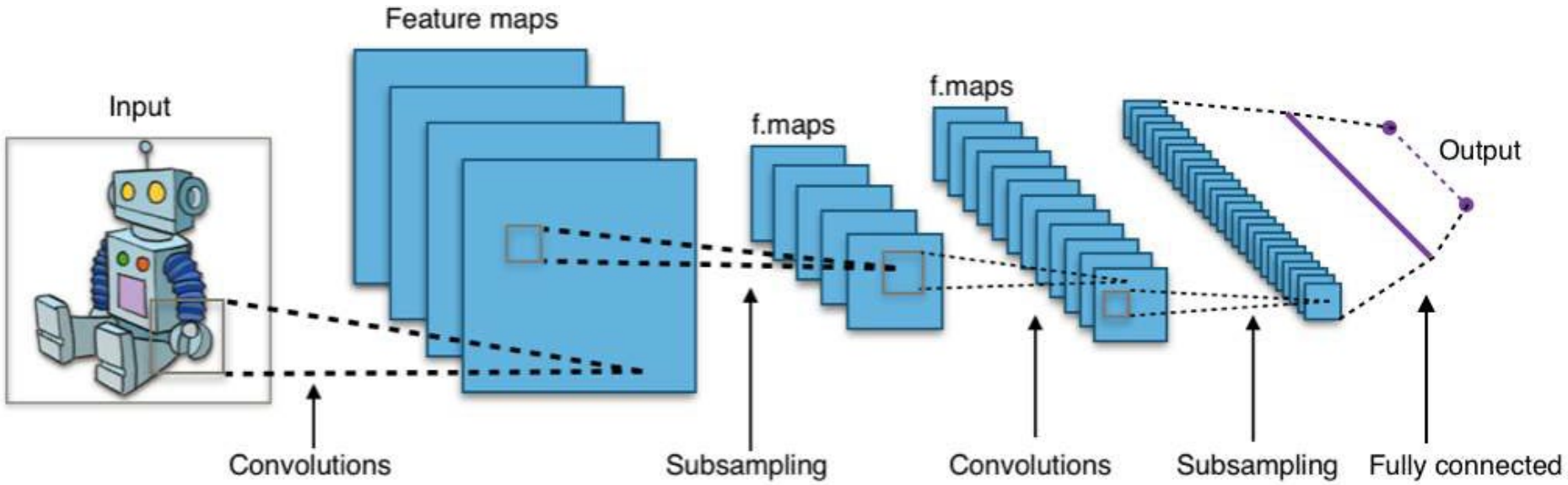
- Numerical gradient: easy to implement, bad to use.
- Symbolic gradient: sometimes useful, often intractable.
- Automatic gradient: exact, fast, error-prone.

In practice: Use symbolic gradient for small/trivial programs. Almost always use analytic gradient, but check correctness of implementation with numerical gradient.

- This is called a gradient check.

This Lecture

Convolutional Neural Networks

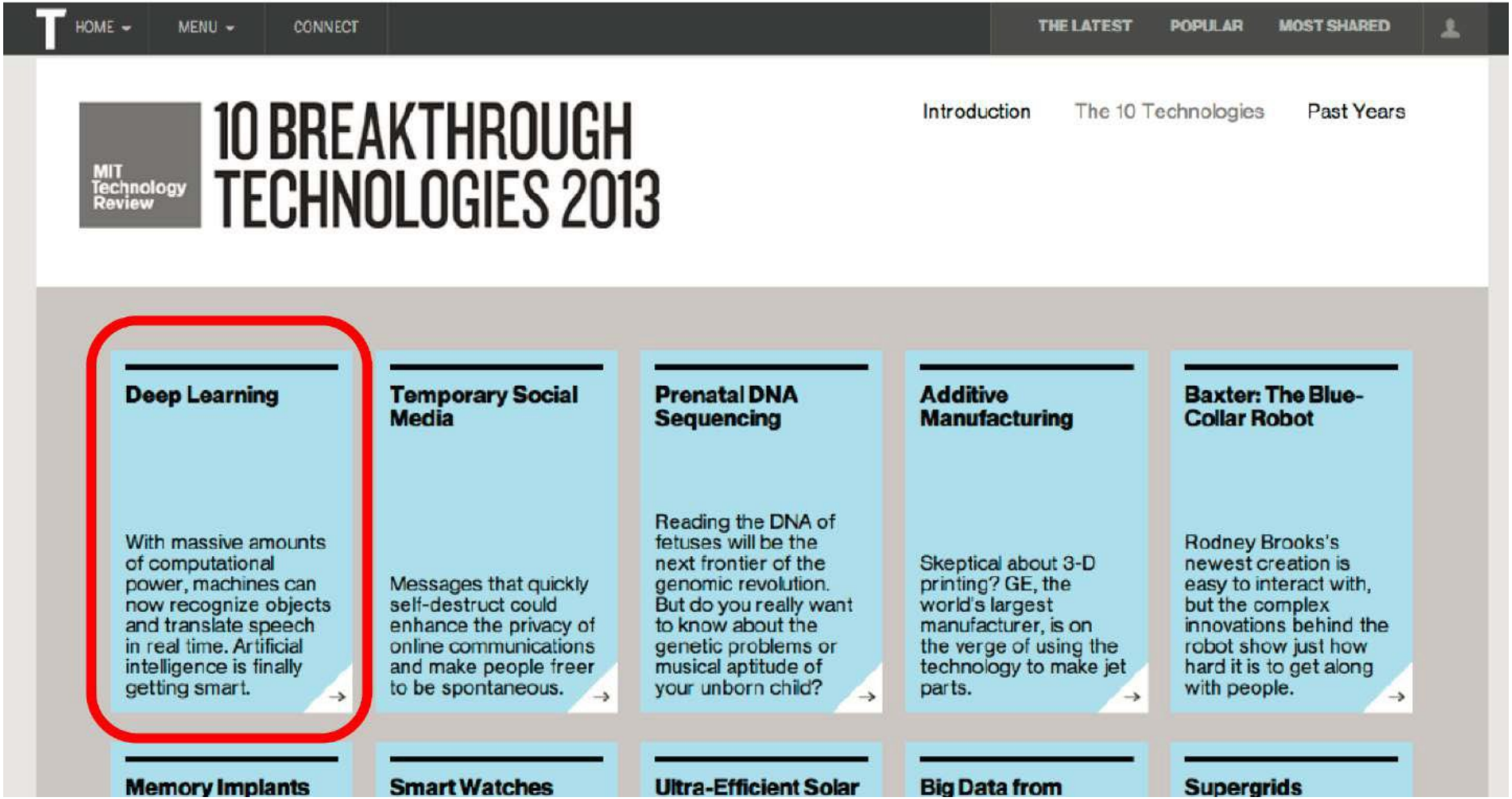


Aside: “CNN” vs “ConvNet”

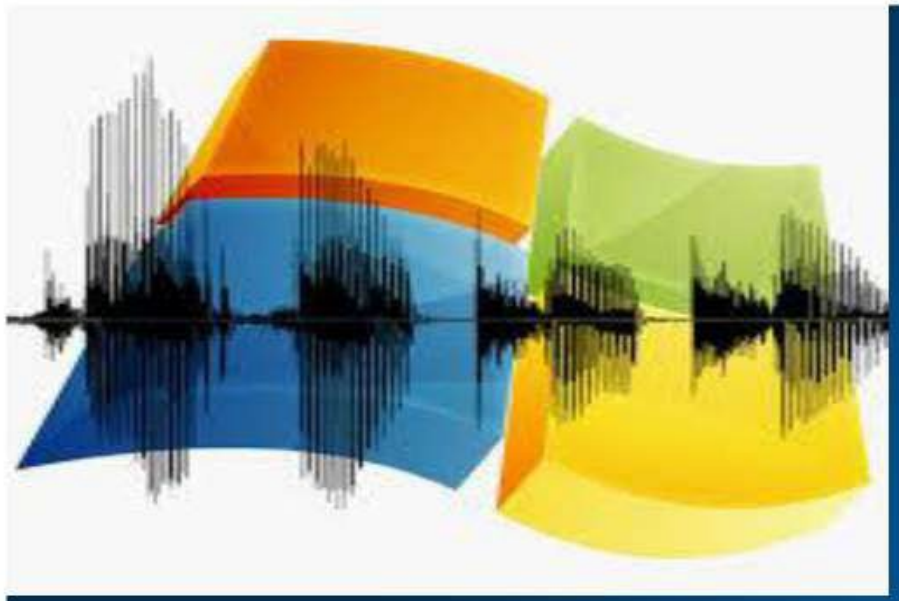
Note:

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

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

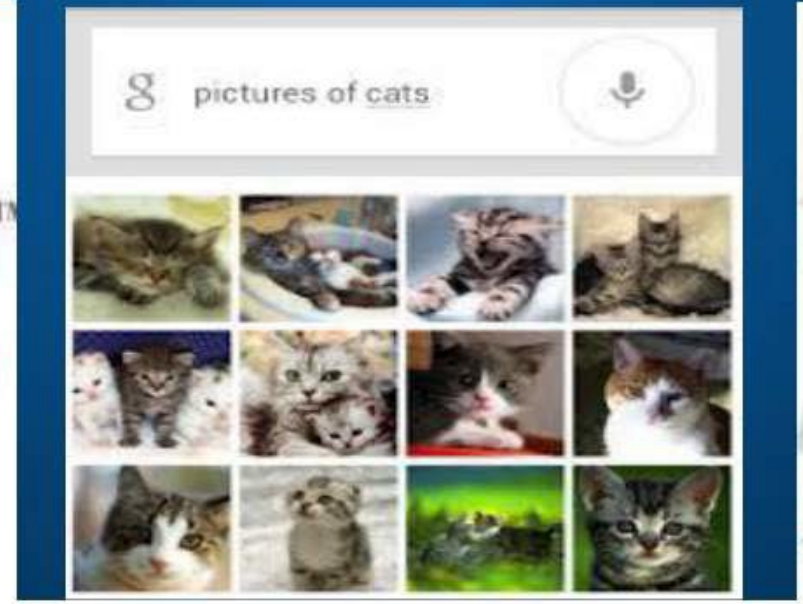
Products




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%)

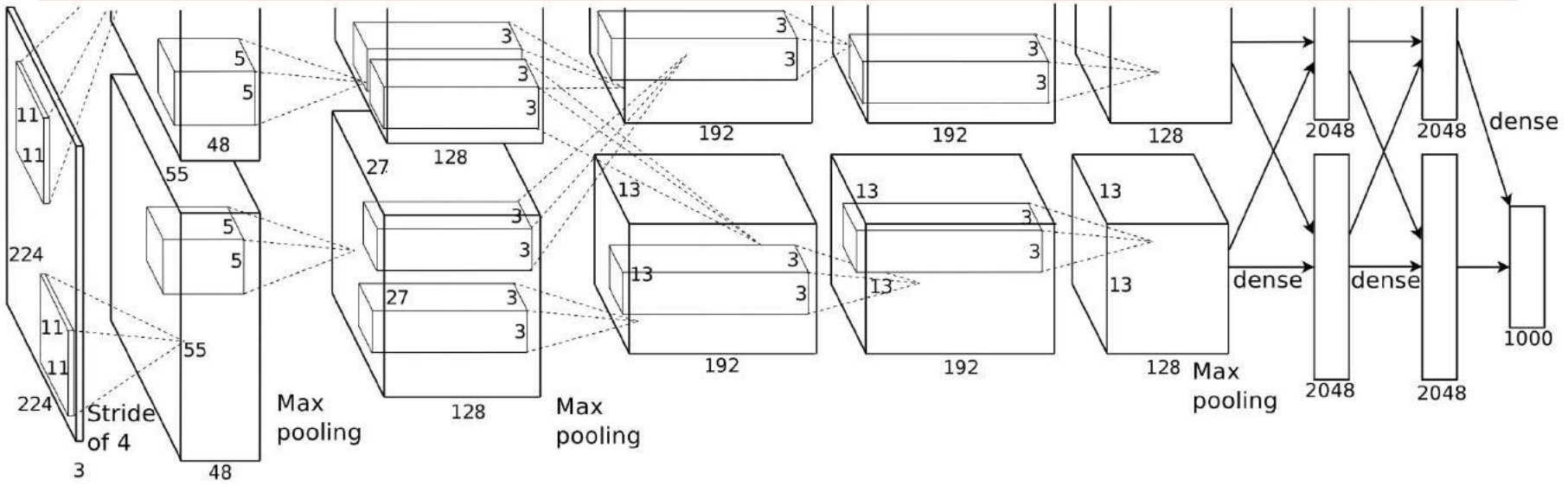
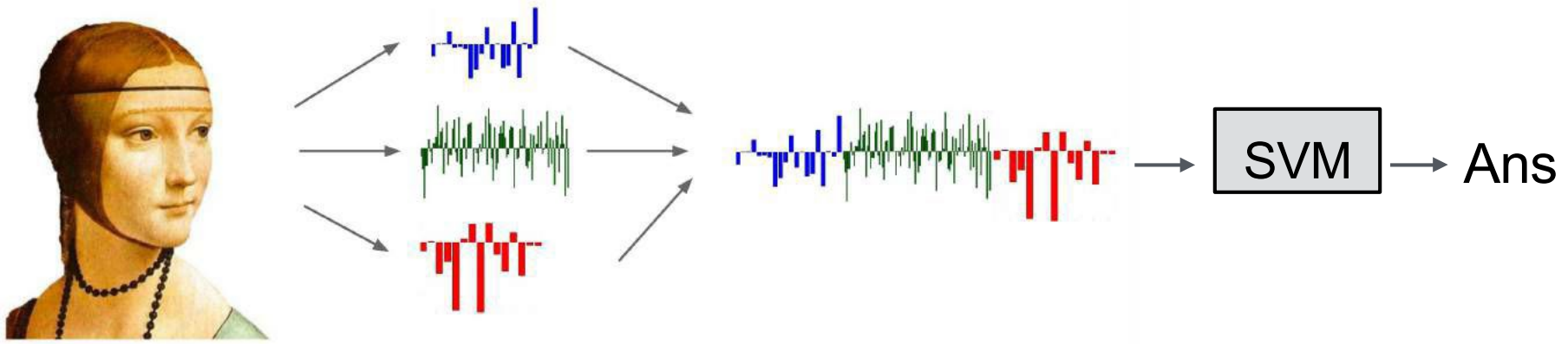


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input 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]

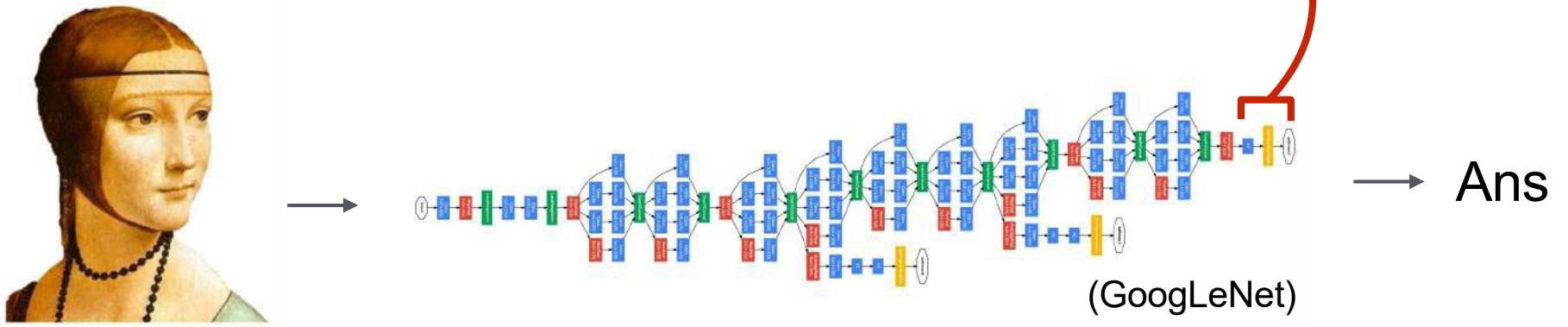
Recap: Before Deep Learning



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

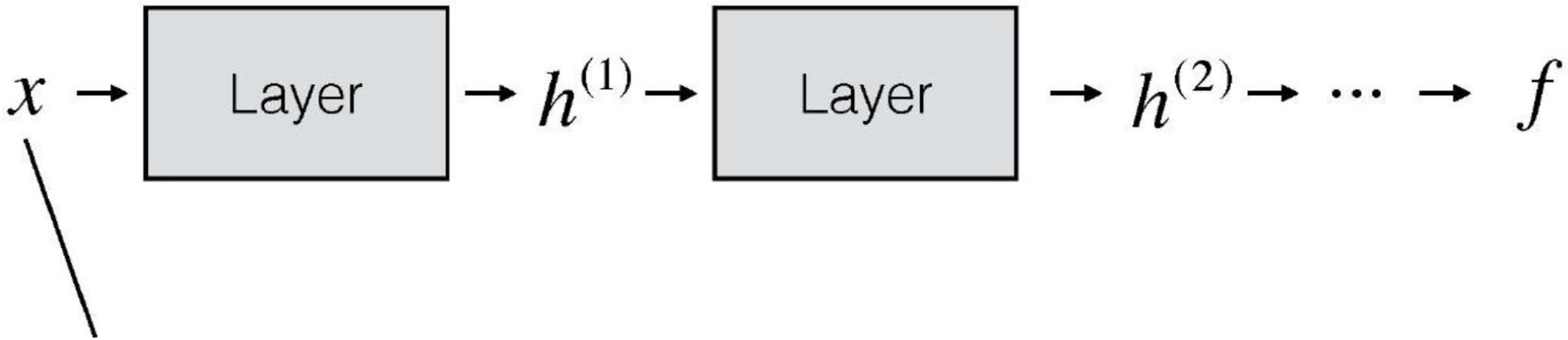
Perform everything with a big neural network, trained end-to-end

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

ConvNets

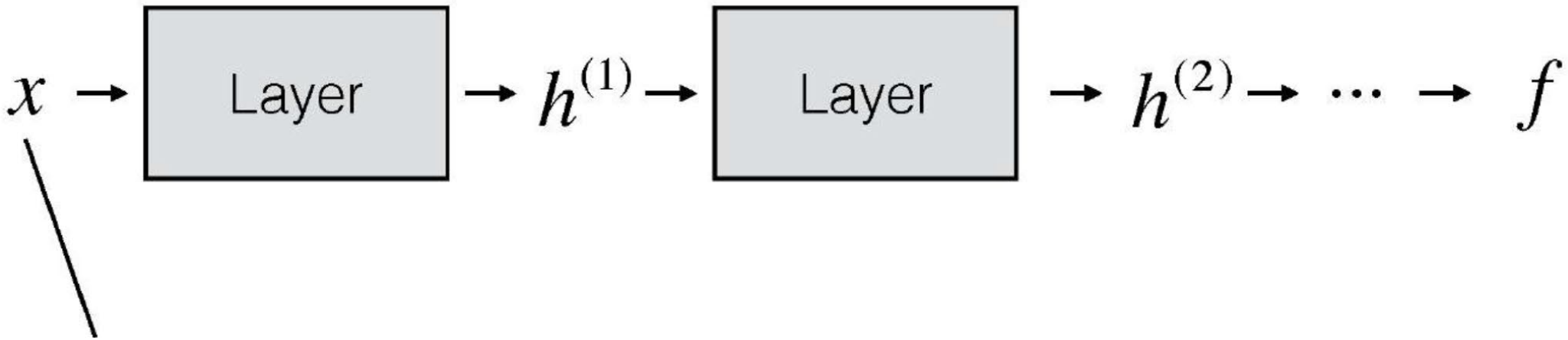
They're just neural networks with
3D activations and weight sharing

What shape should the activations have?



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

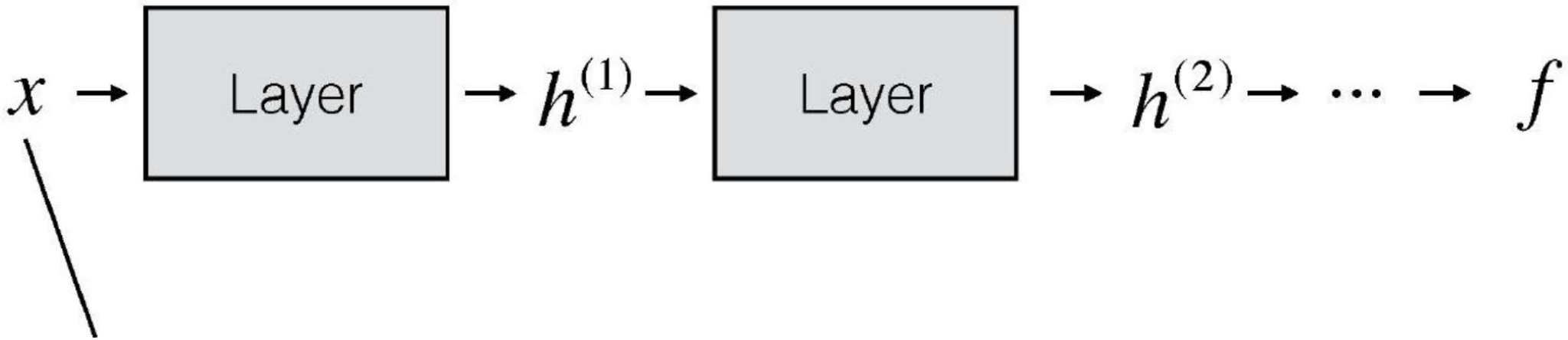
What shape should the activations have?



- 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?



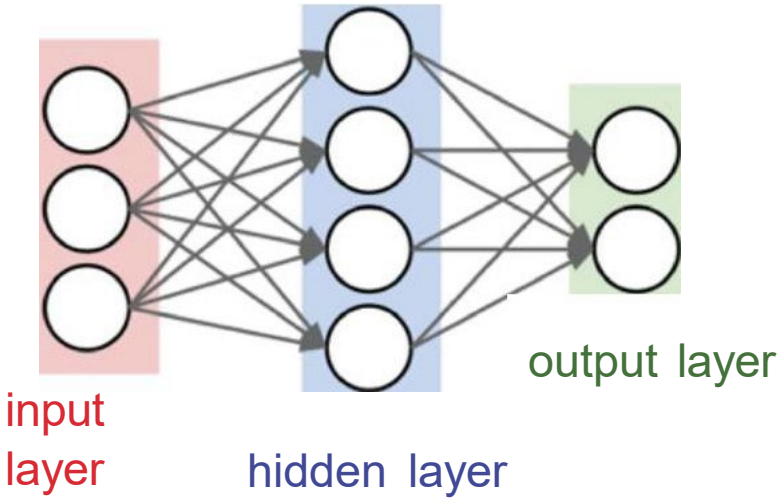
- The input is an image, which is 3D (RGB channel, height, width)
- We could flatten it to a 1D vector, but then we lose structure
- What about keeping everything in 3D?

ConvNets

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

3D Activations

before:

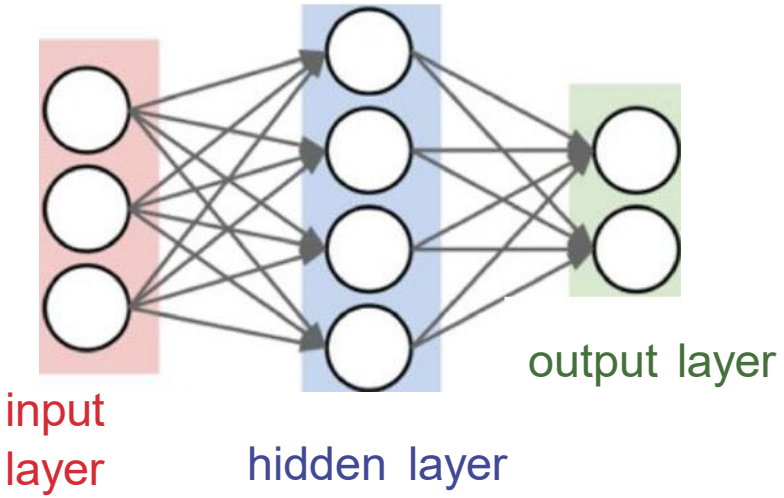


(1D vectors)

Figure: Andrej Karpathy

3D Activations

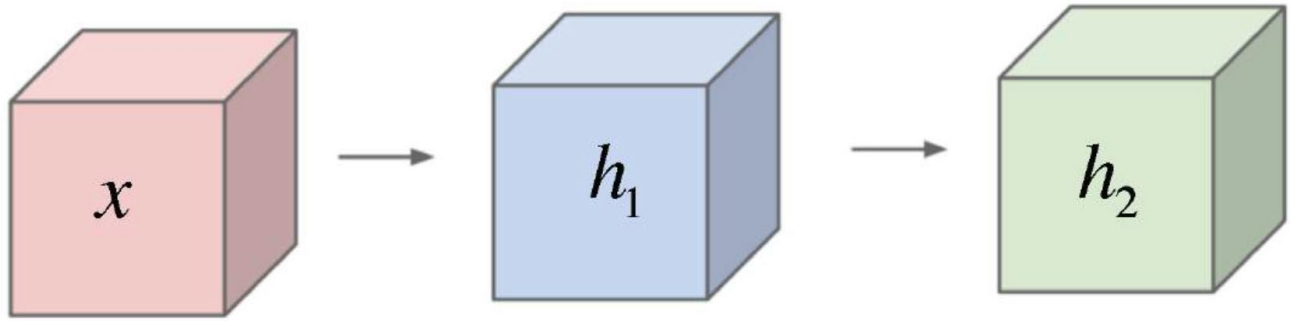
before:



(1D vectors)

---

now:



(3D arrays)

Figure: Andrej Karpathy

3D Activations

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

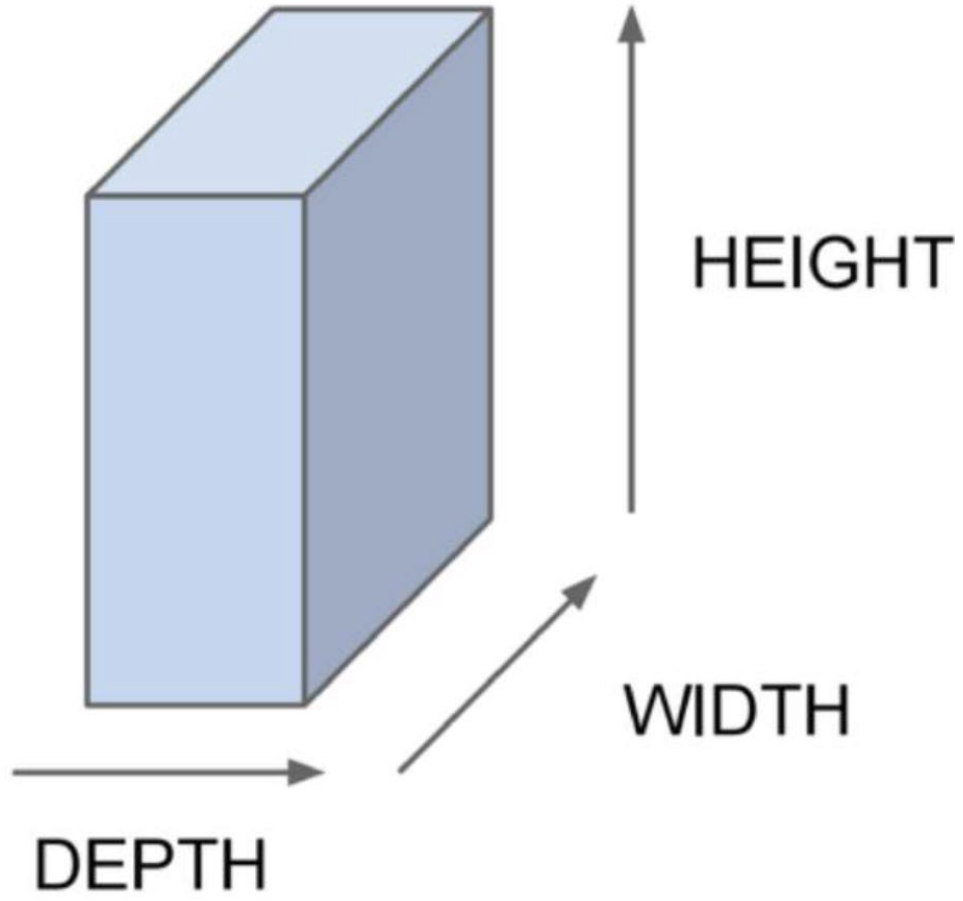
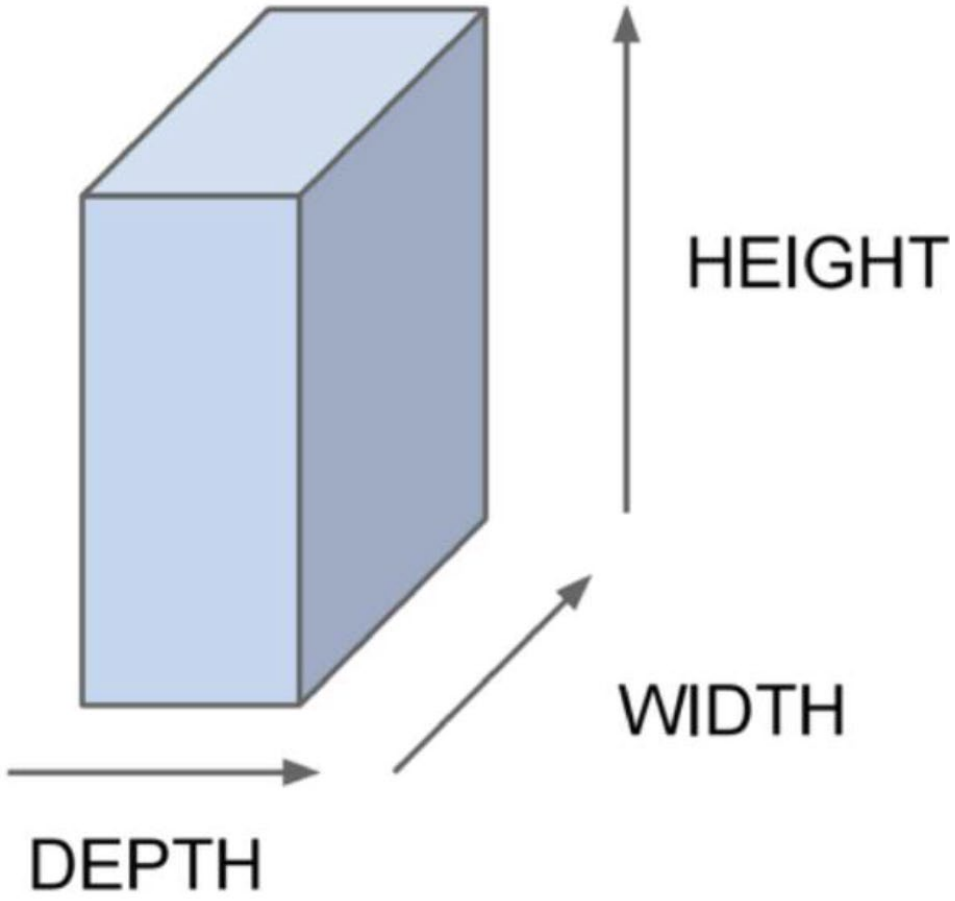


Figure: Andrej Karpathy

3D Activations

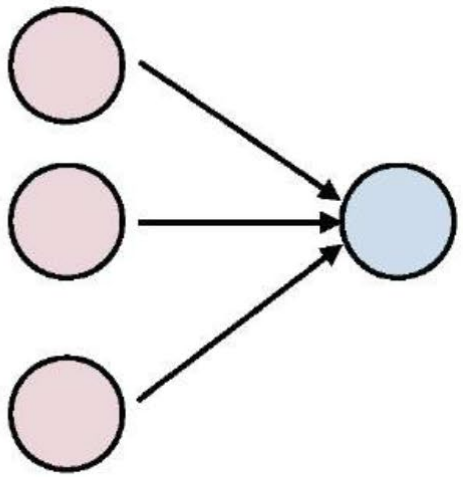
All Neural Net activations arranged in **3 dimensions:**



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

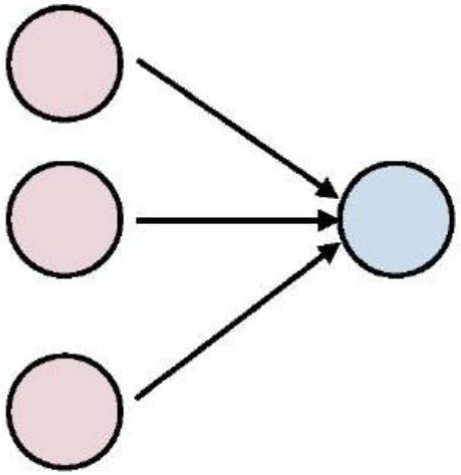
3D Activations

1D Activations:

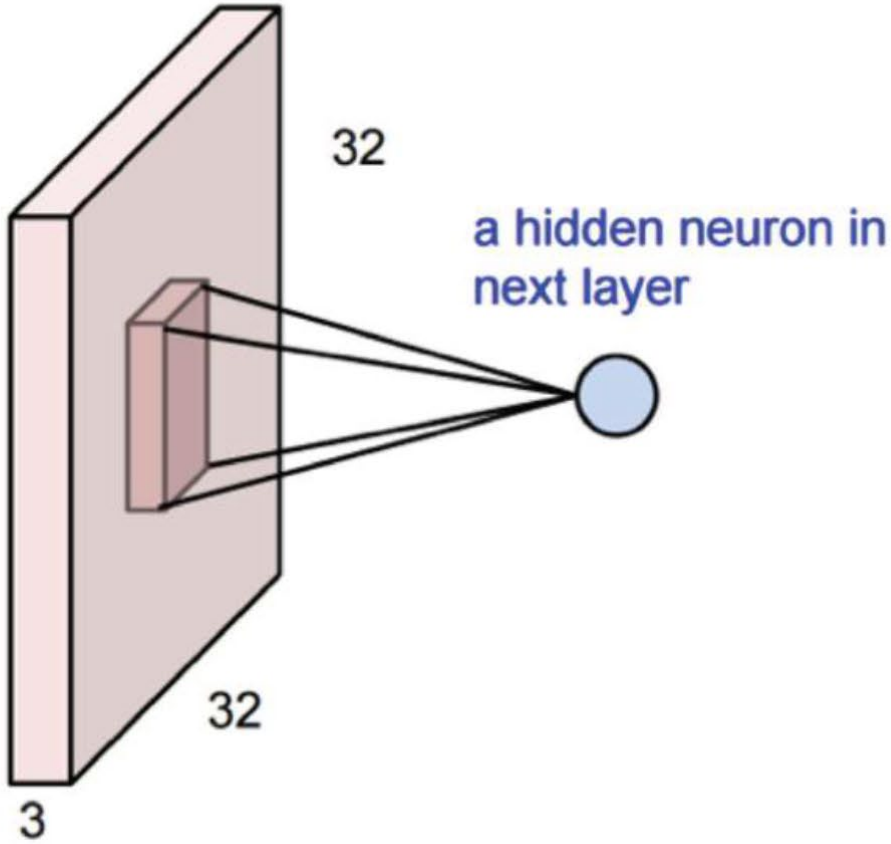


3D Activations

1D Activations:



3D Activations:



3D Activations

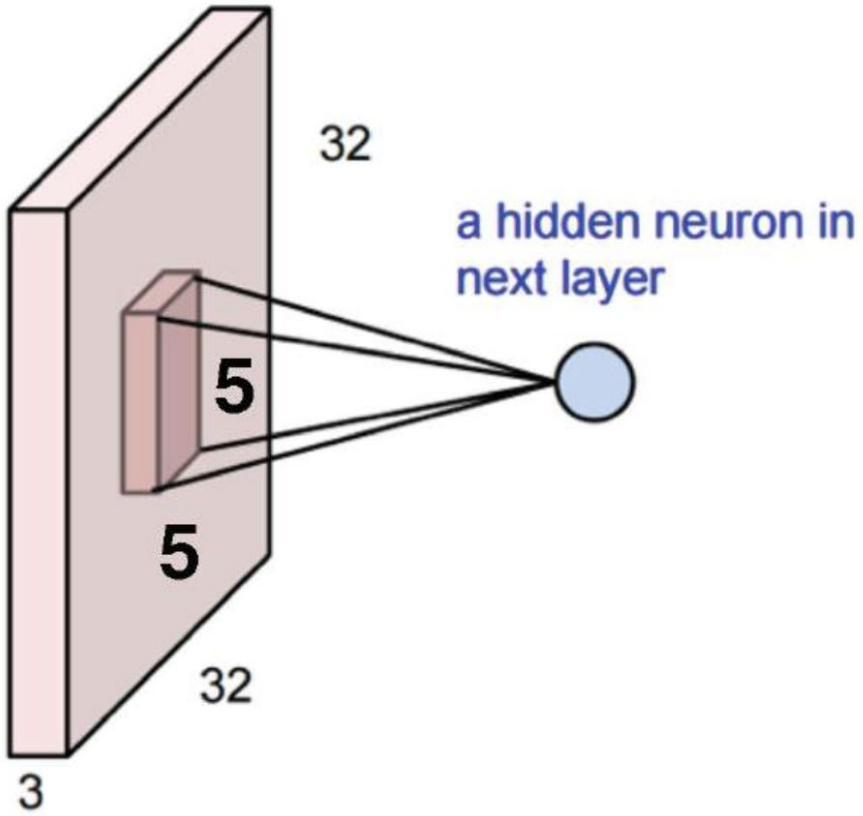
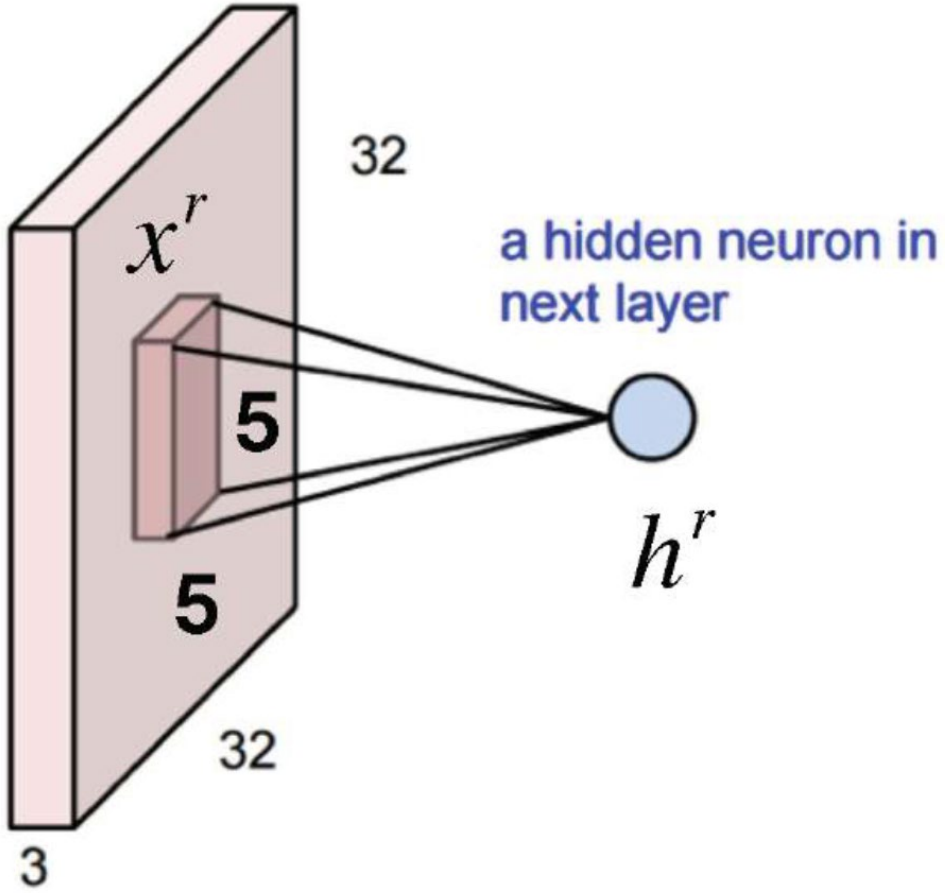


Figure: Andrej Karpathy

- 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"

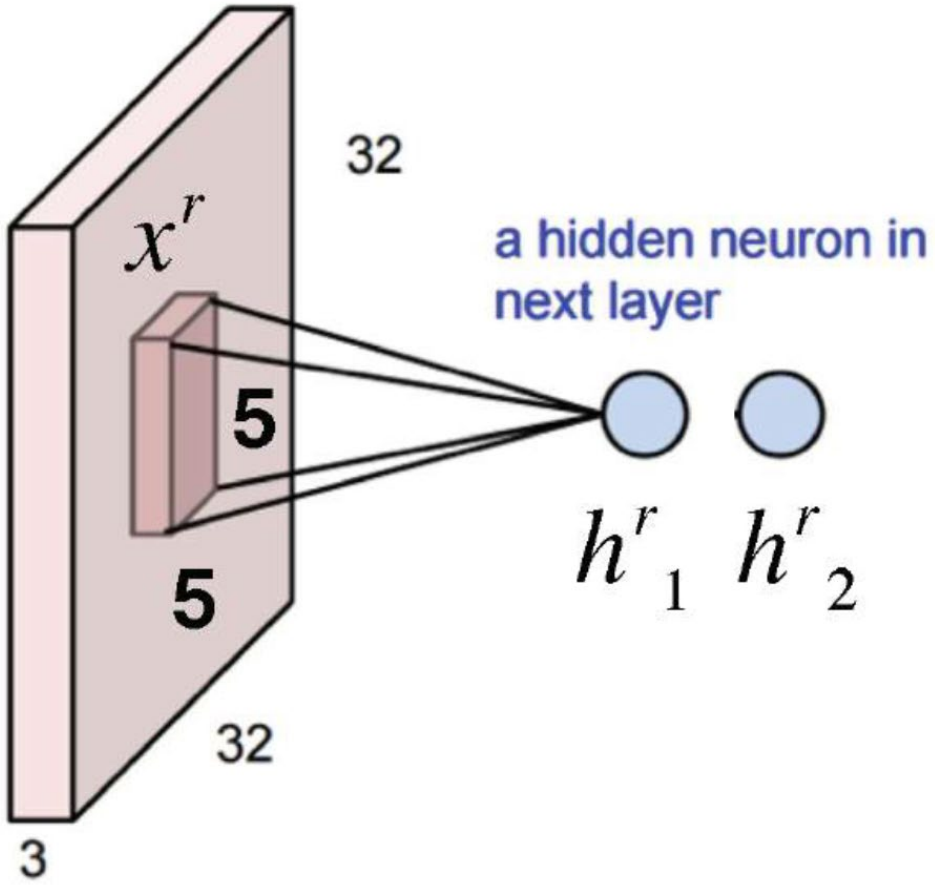
With output neuron h^r

Then the output is:

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

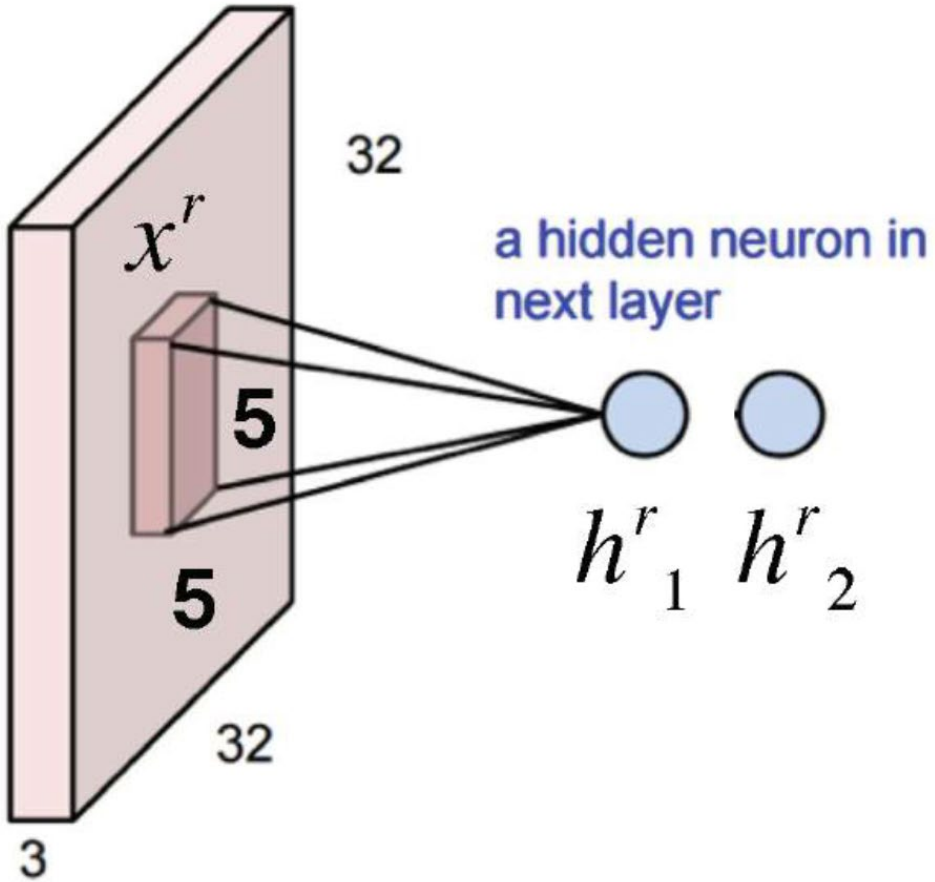
Sum over 3 axes

3D Activations



With **2 output** neurons

3D Activations

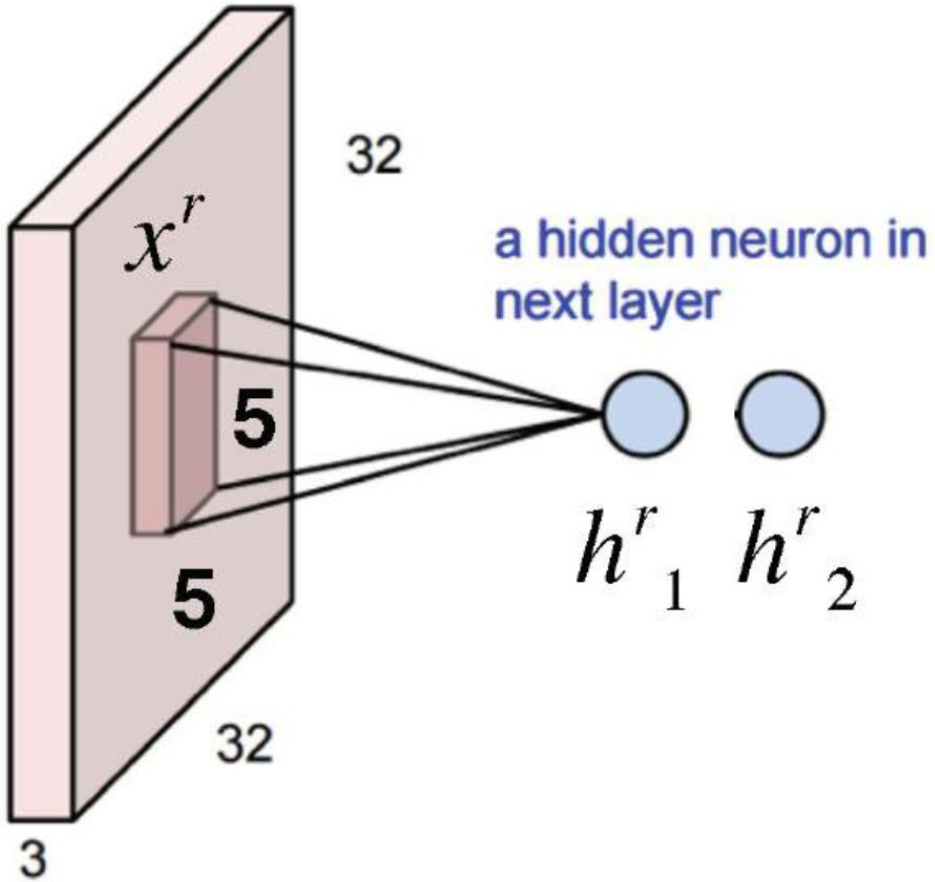


With **2 output** neurons

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

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

3D Activations

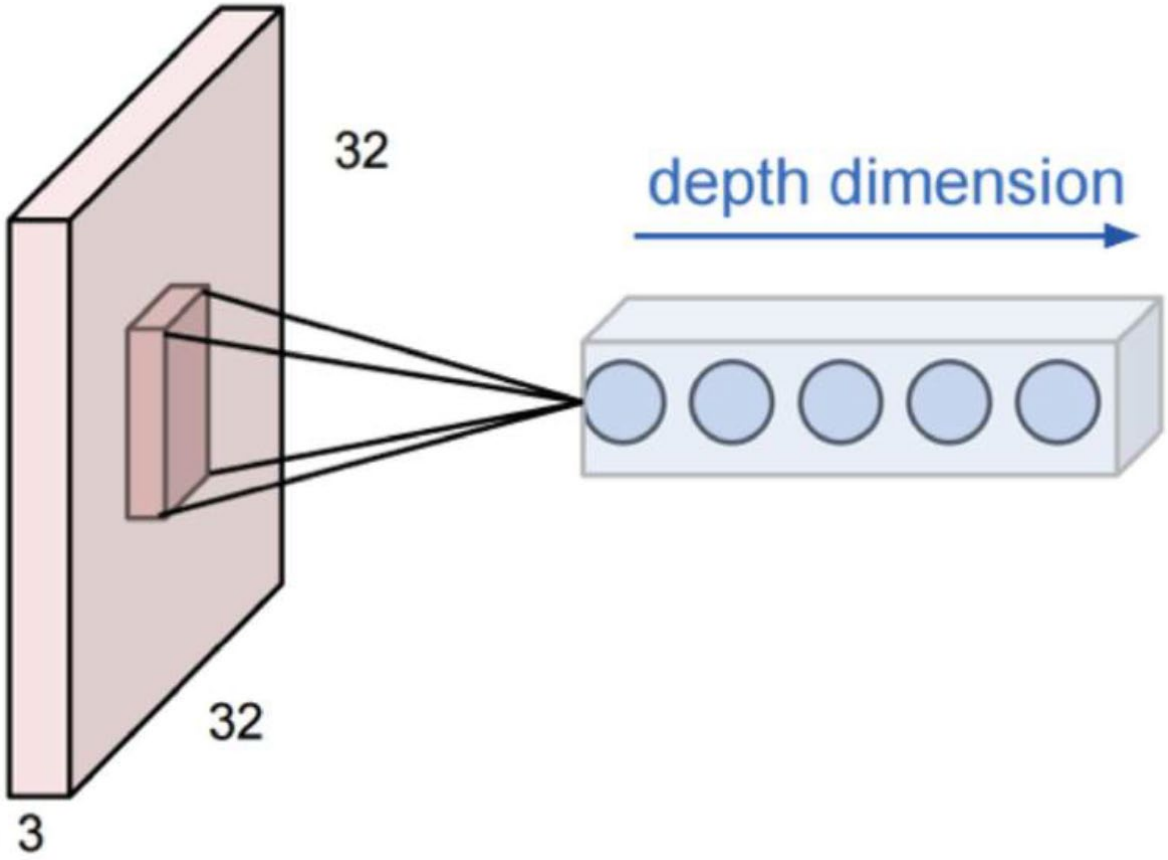


With **2 output** neurons

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

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

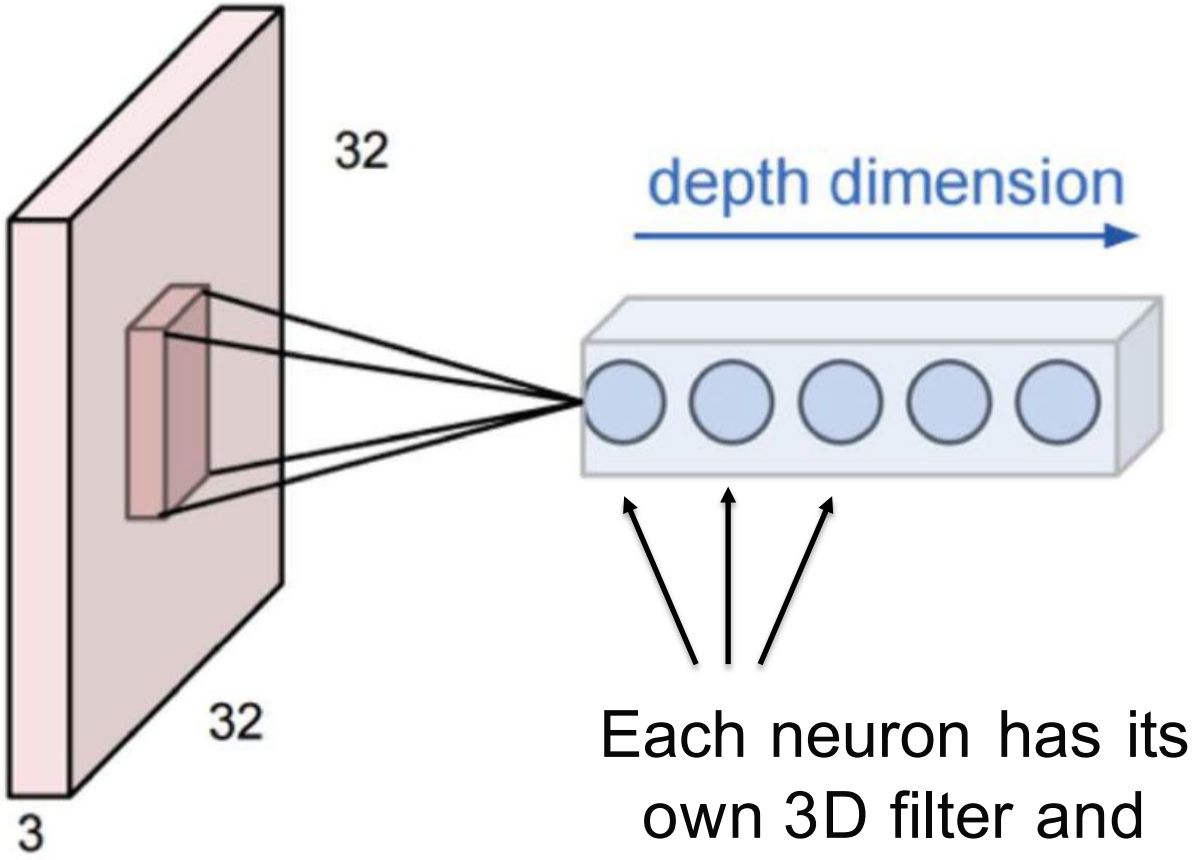
3D Activations



We can keep adding more outputs

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

3D Activations

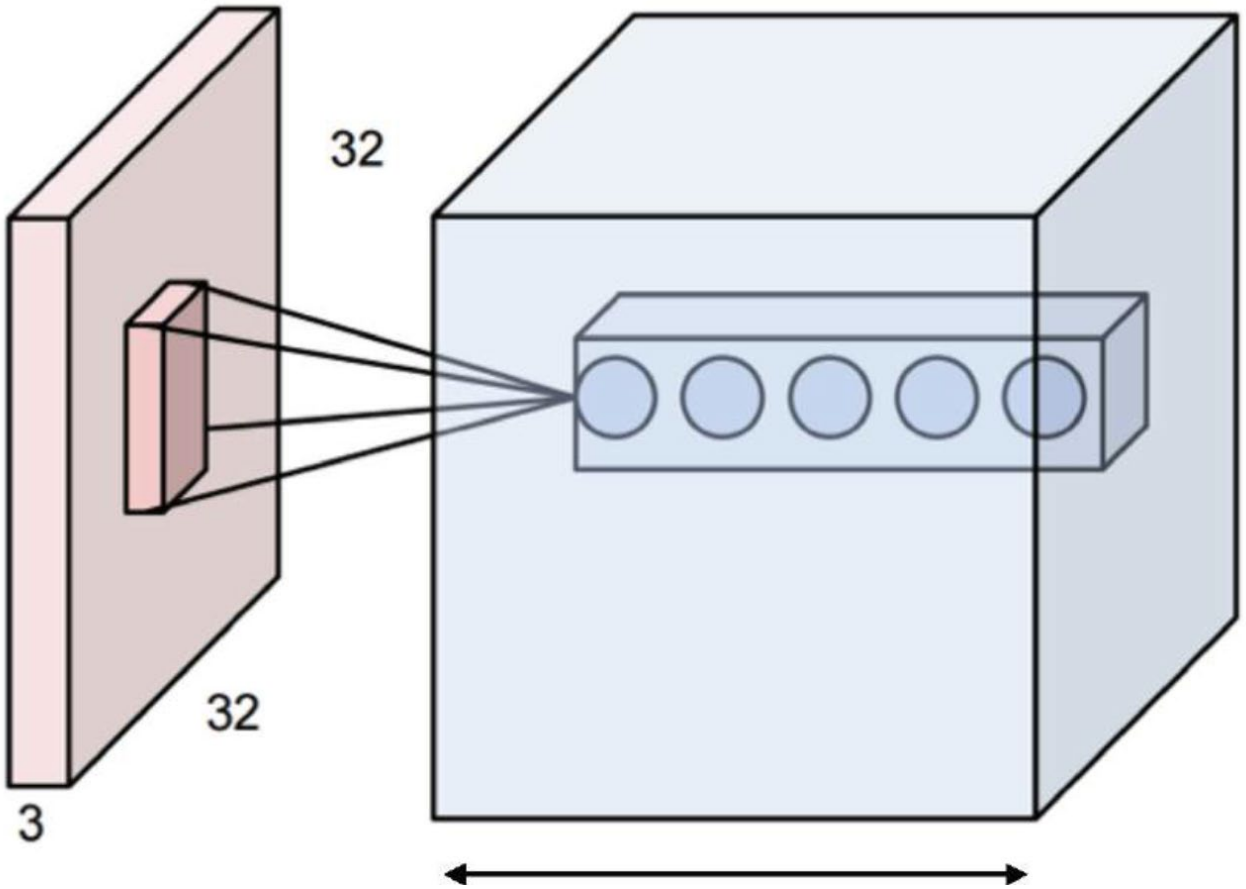


Each neuron has its own 3D filter and own (scalar) bias

We can keep adding more outputs

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

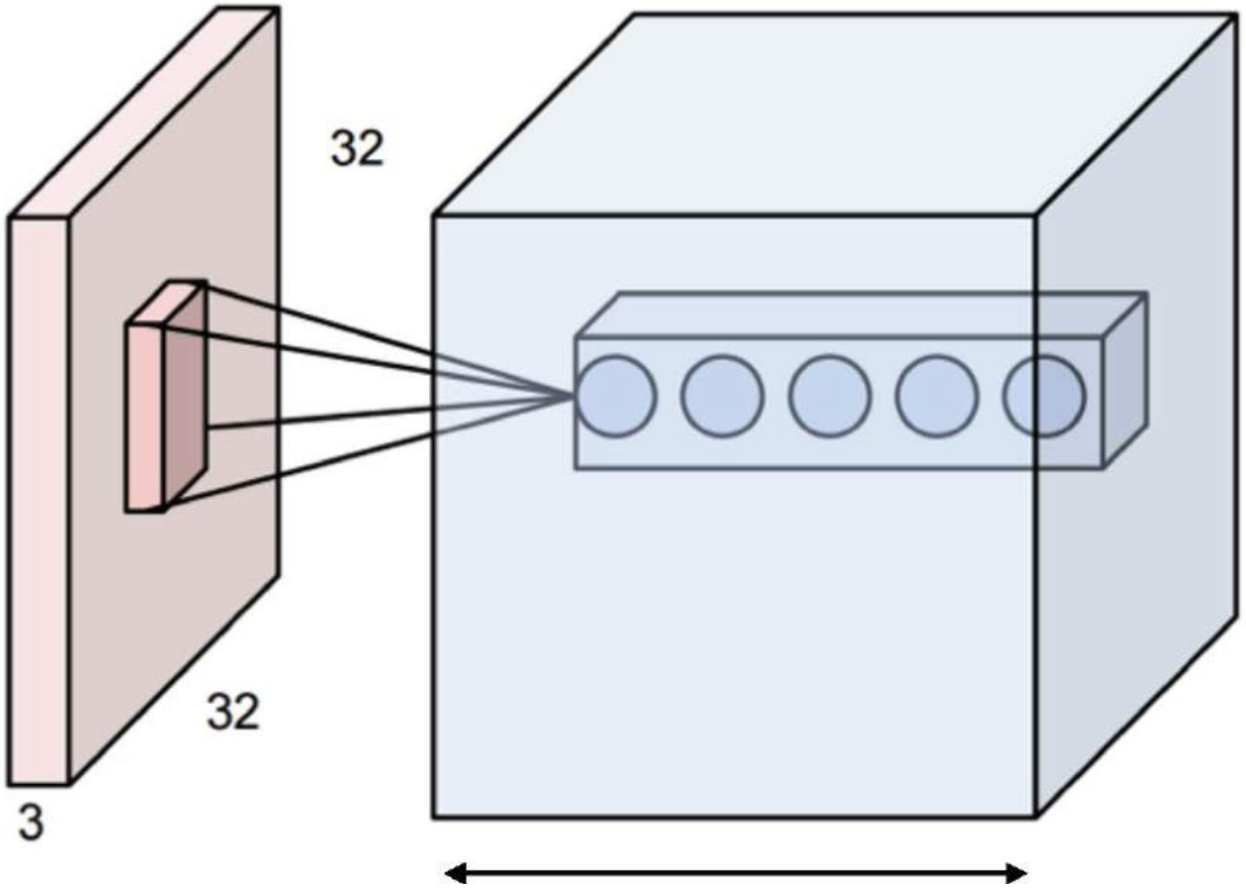
3D Activations



D sets of weights
(also called filters)

Now repeat this across the input

3D Activations



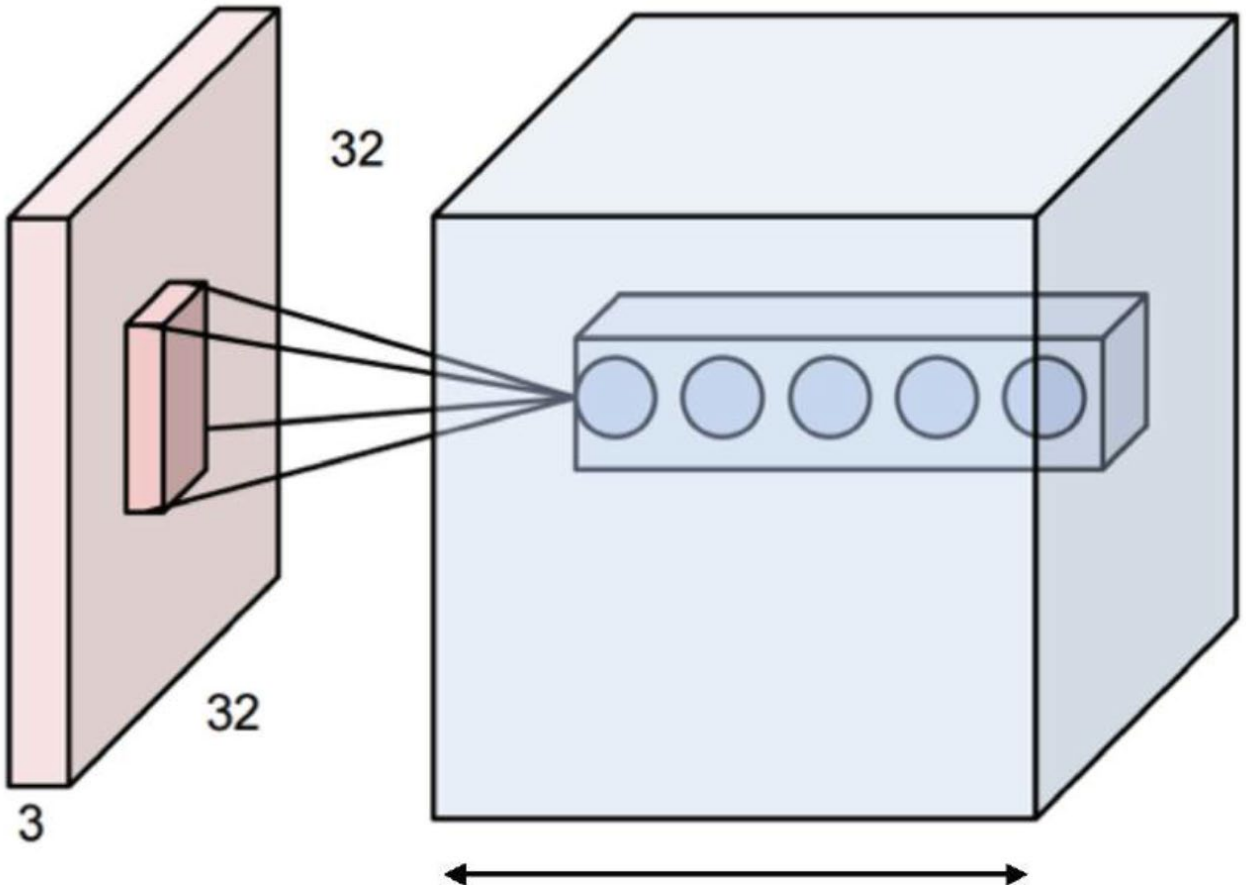
D sets of weights
(also called filters)

Now repeat this across the input

Weight sharing:

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

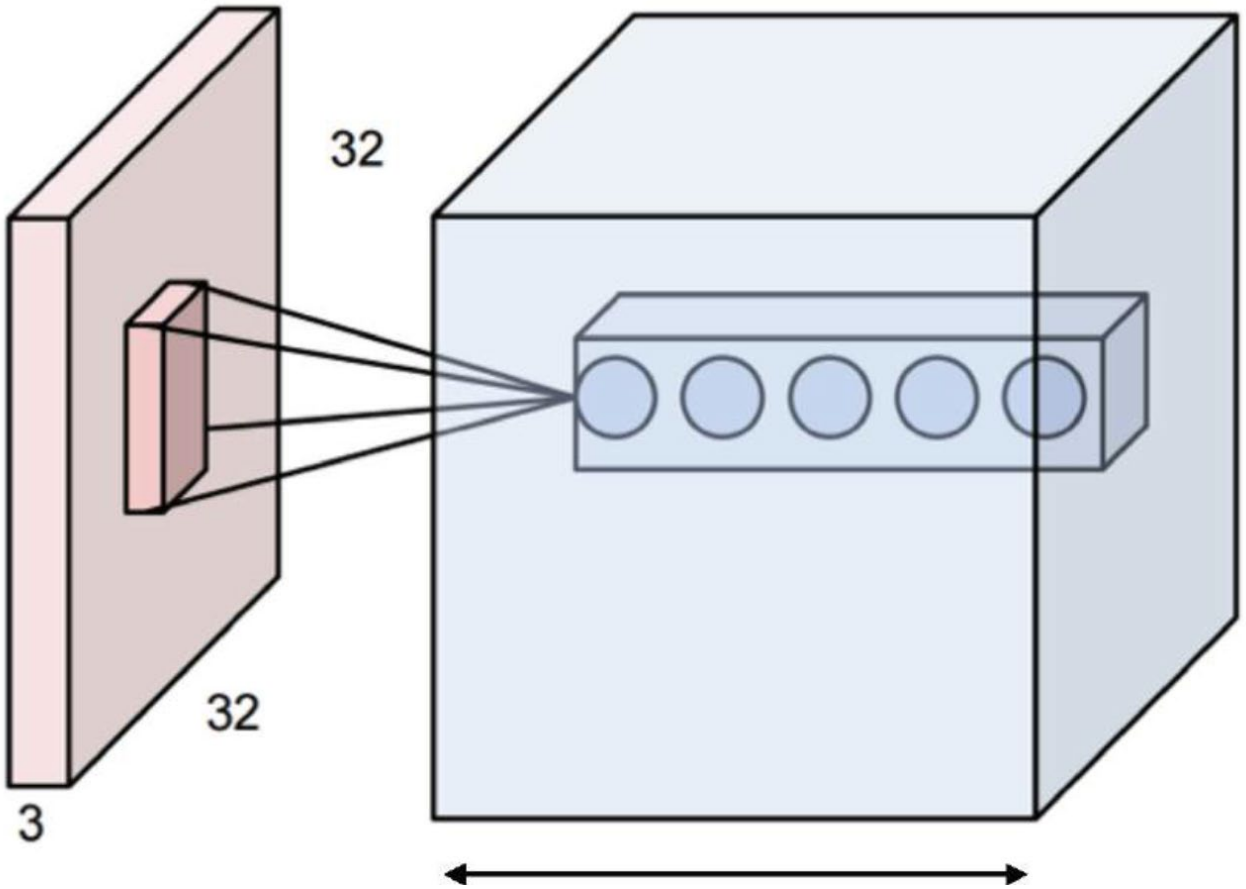
3D Activations



D sets of weights
(also called filters)

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

3D Activations

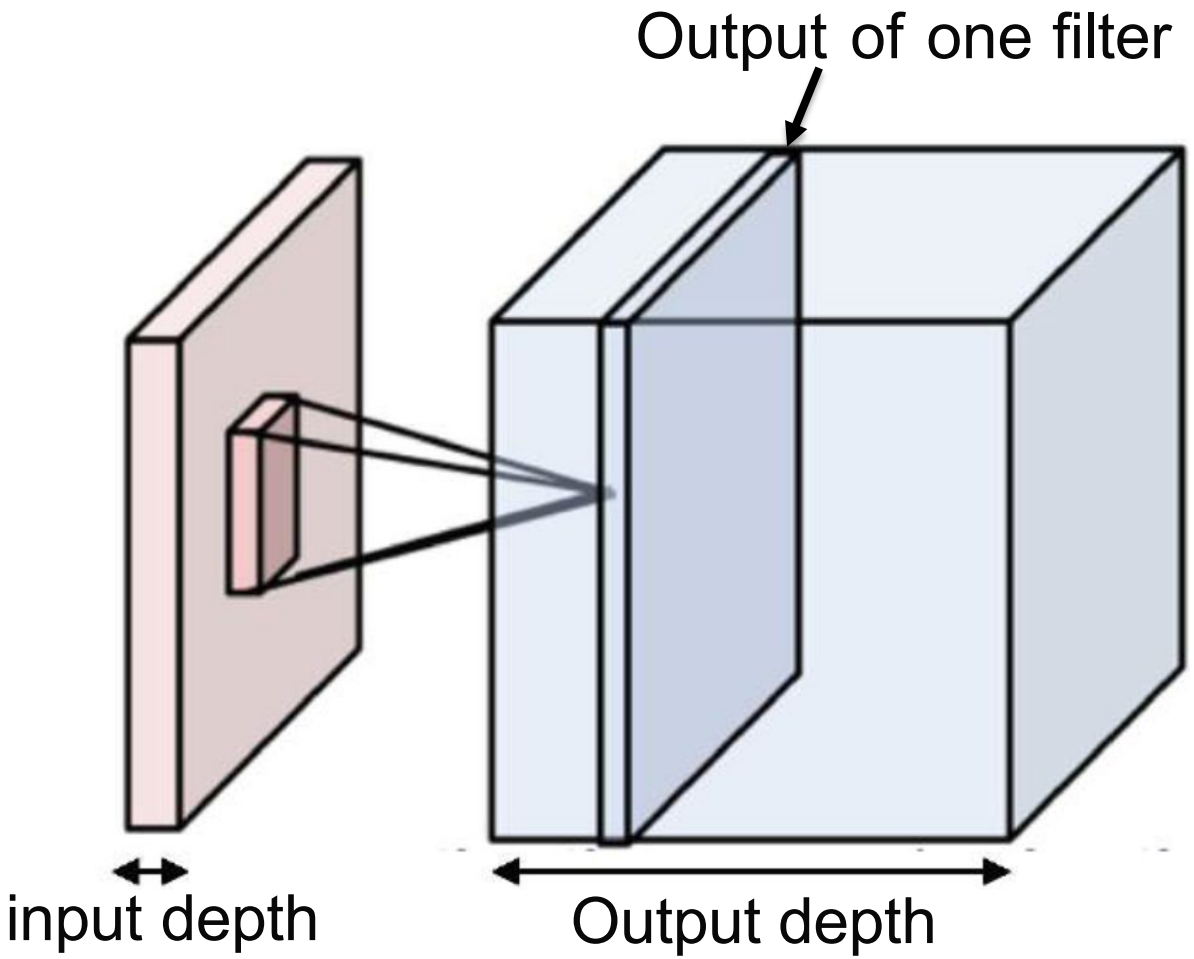


D sets of weights
(also called filters)

With weight sharing,
this is called
convolution

Without weight sharing,
this is called a
locally connected layer

3D Activations

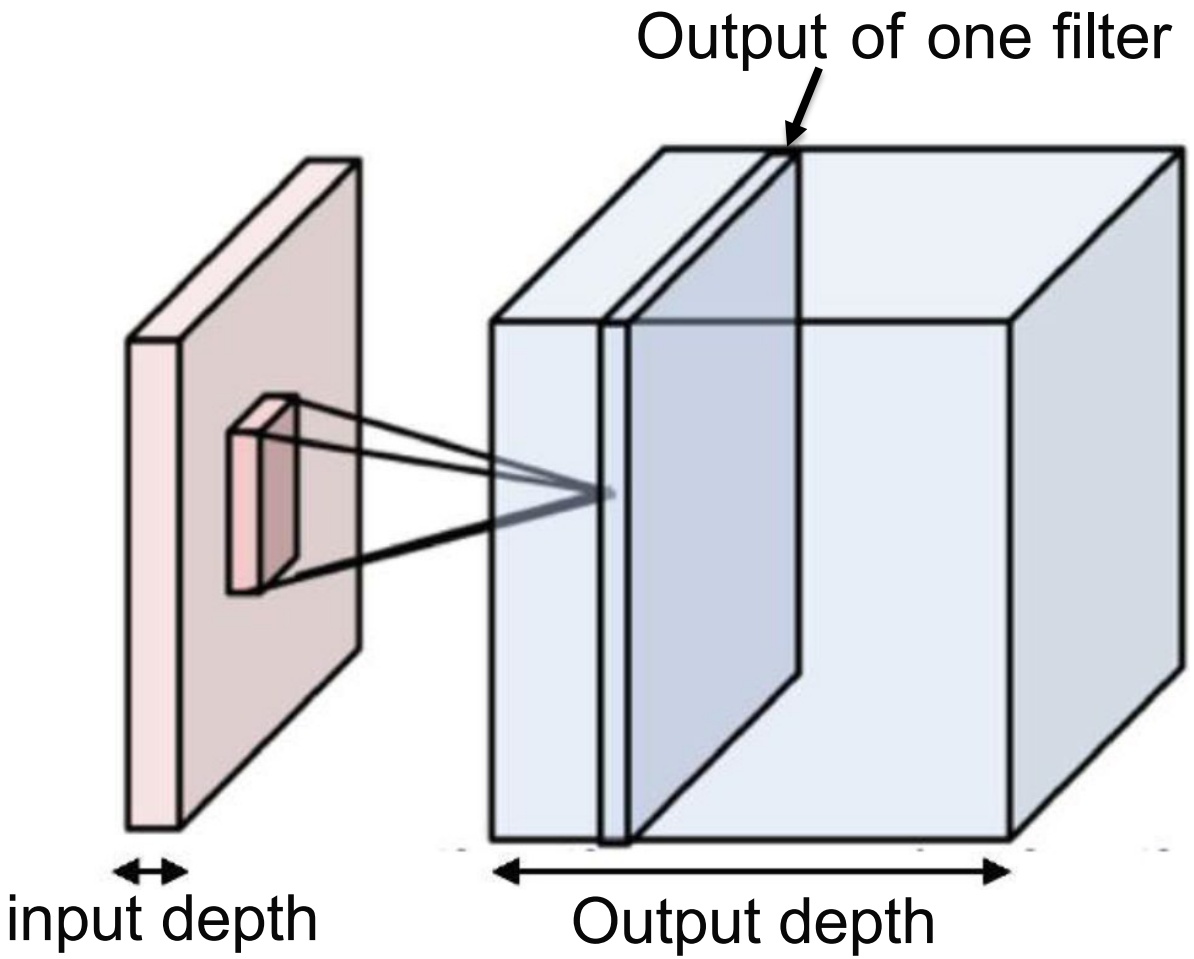


One set of weights gives one slice in the output

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:

Figure: Andrej Karpathy

3D Activations

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

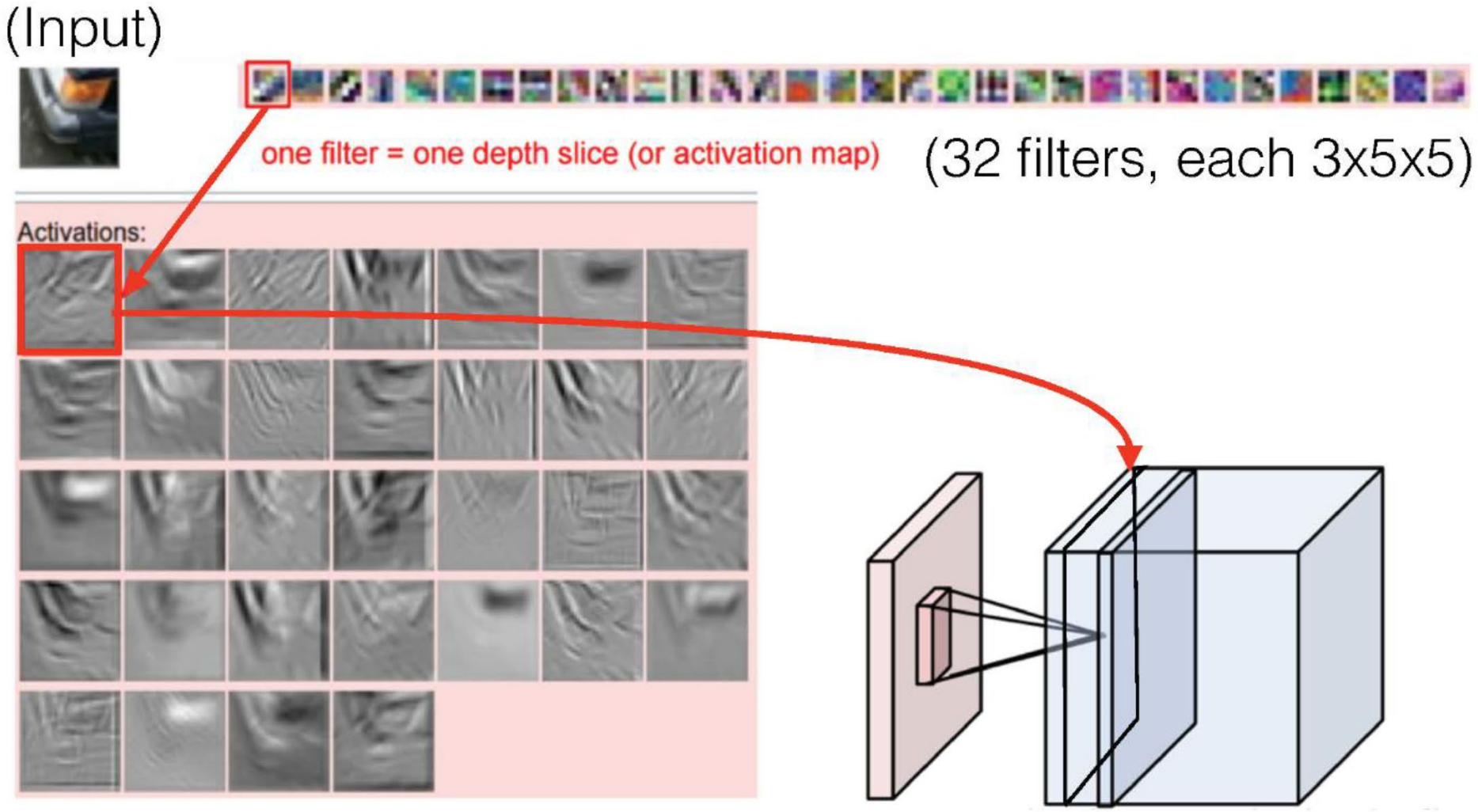


Figure: Andrej Karpathy

3D Activations

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

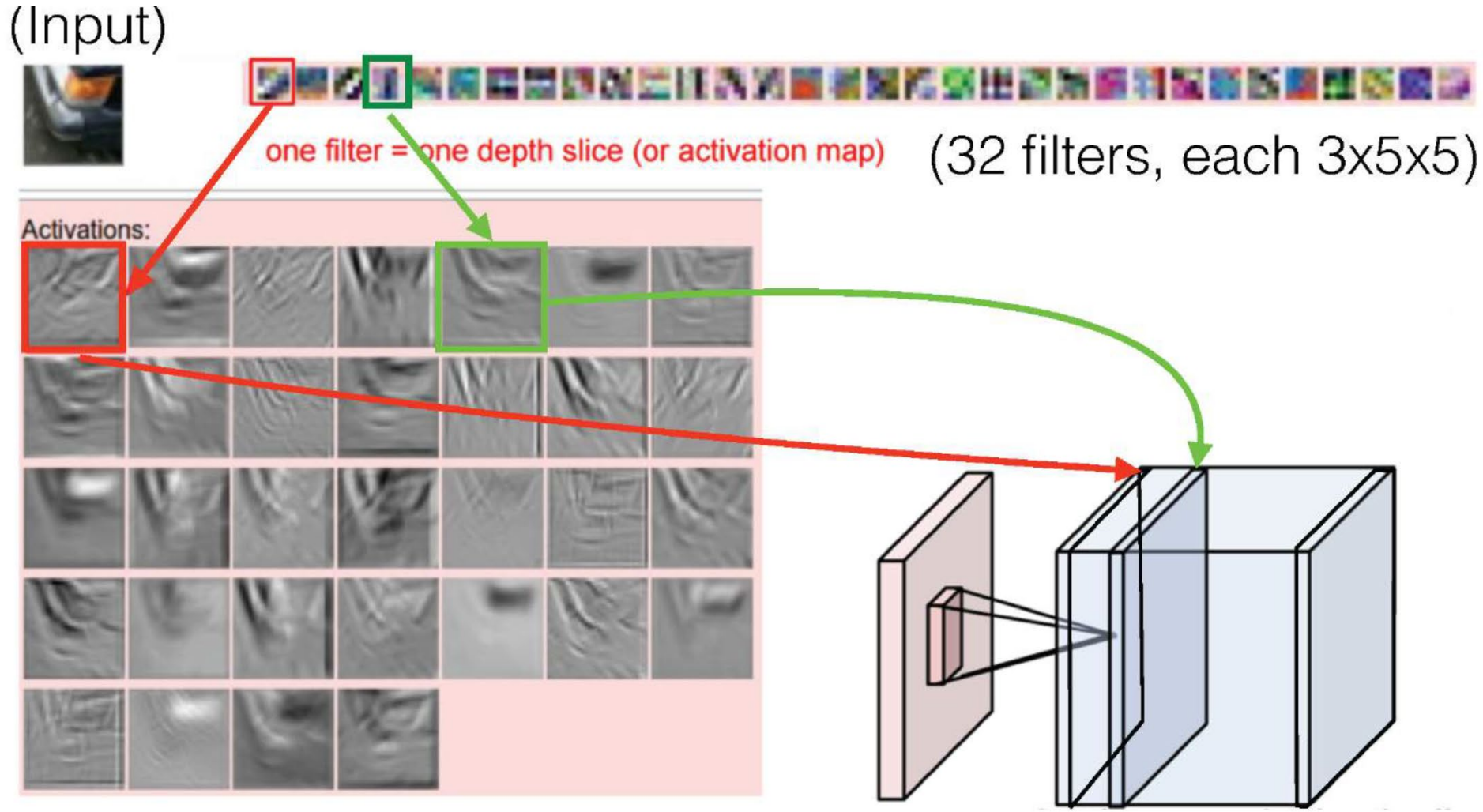


Figure: Andrej Karpathy

3D Activations

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

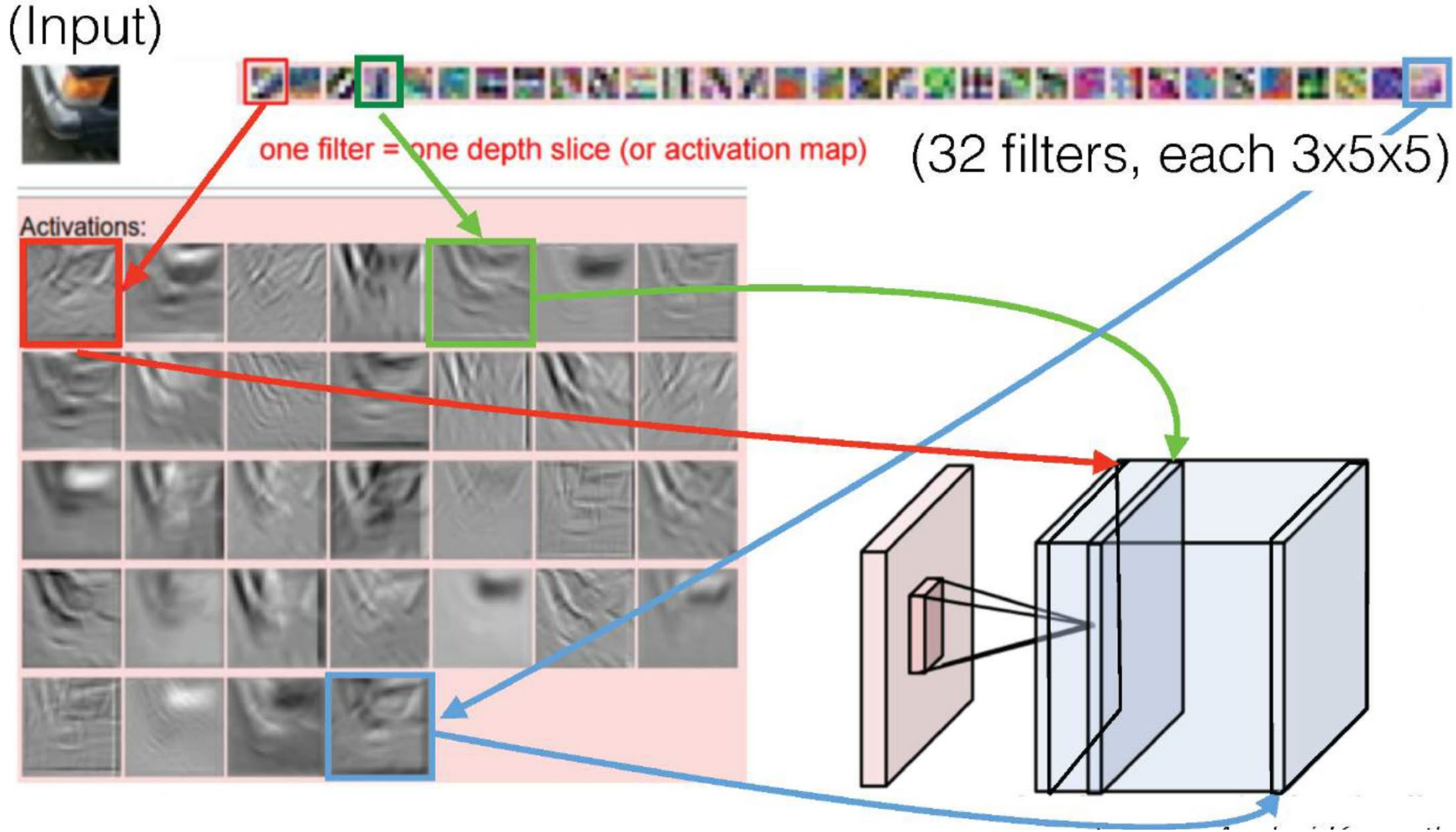
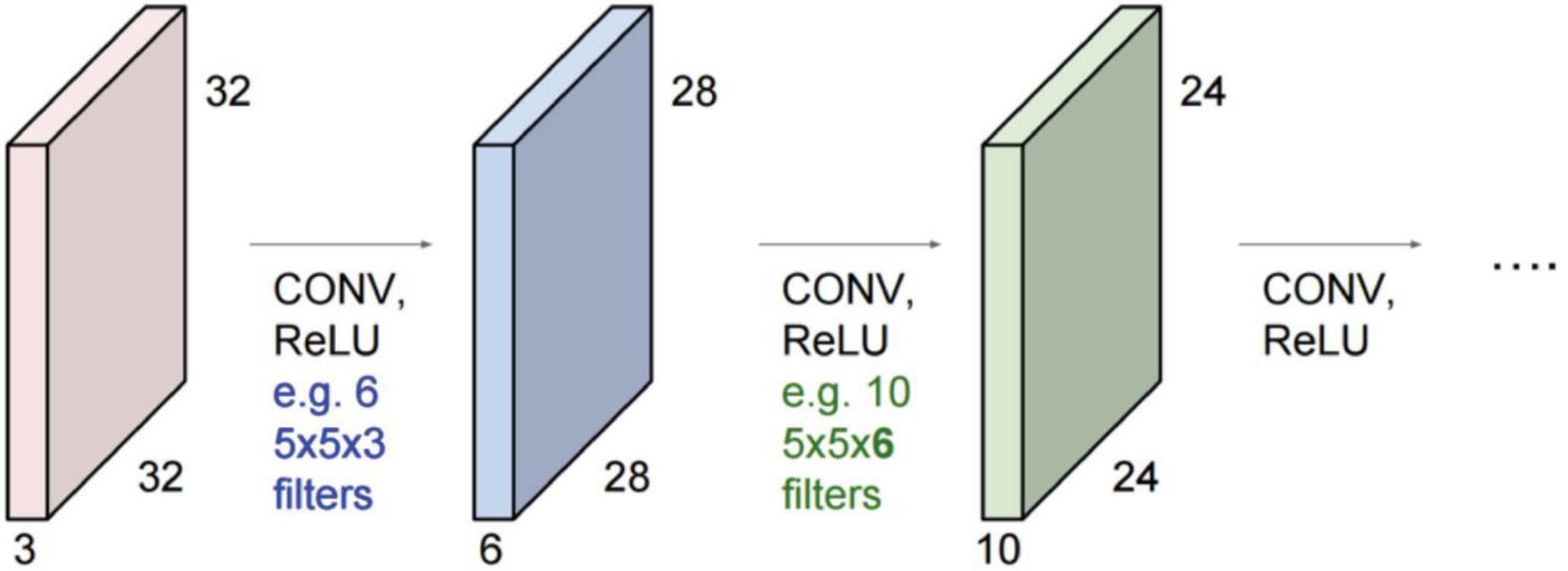


Figure: Andrej Karpathy

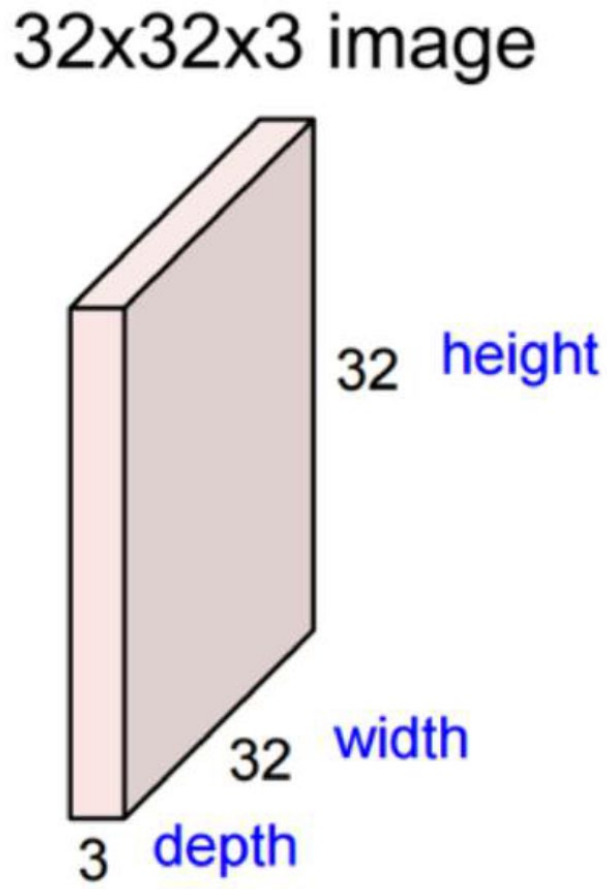
Recap

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



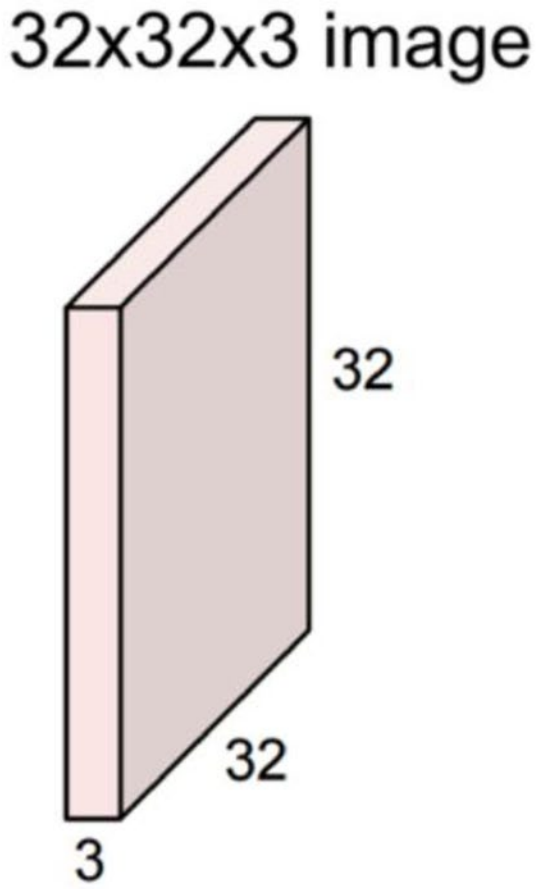
Recap

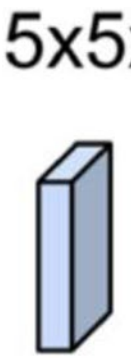
Convolution Layer



Recap

Convolution Layer

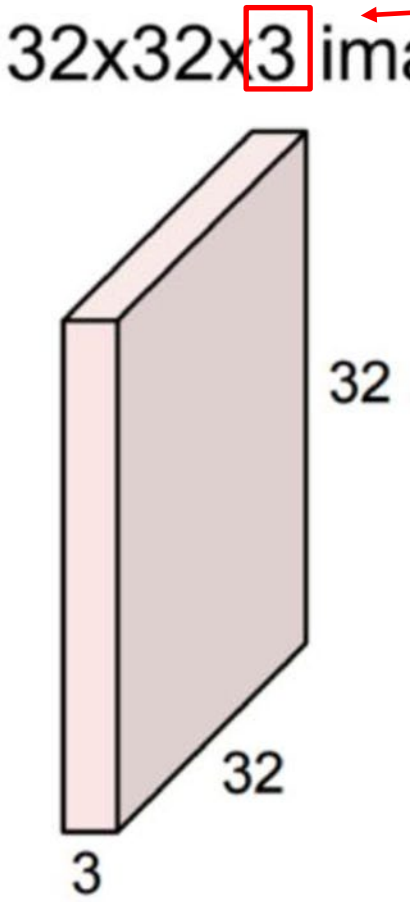




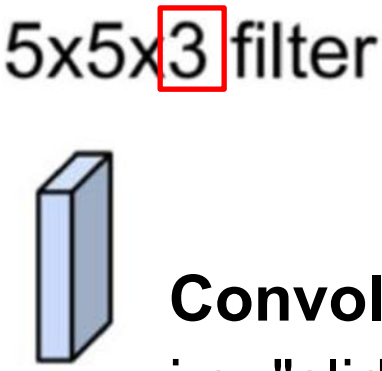
Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

Recap

Convolution Layer



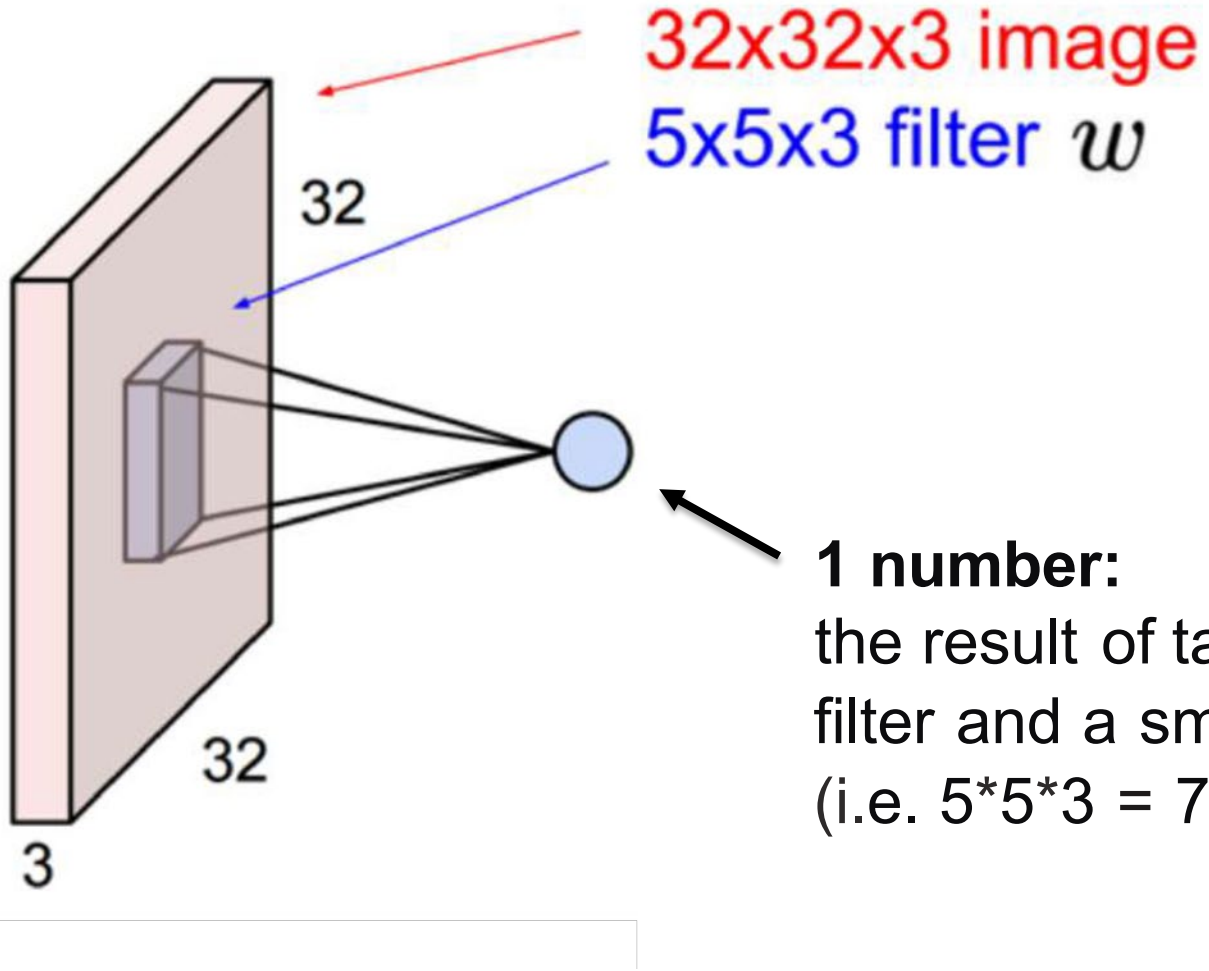
Filters always extend the full depth of the input volume



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

Recap

Convolution Layer

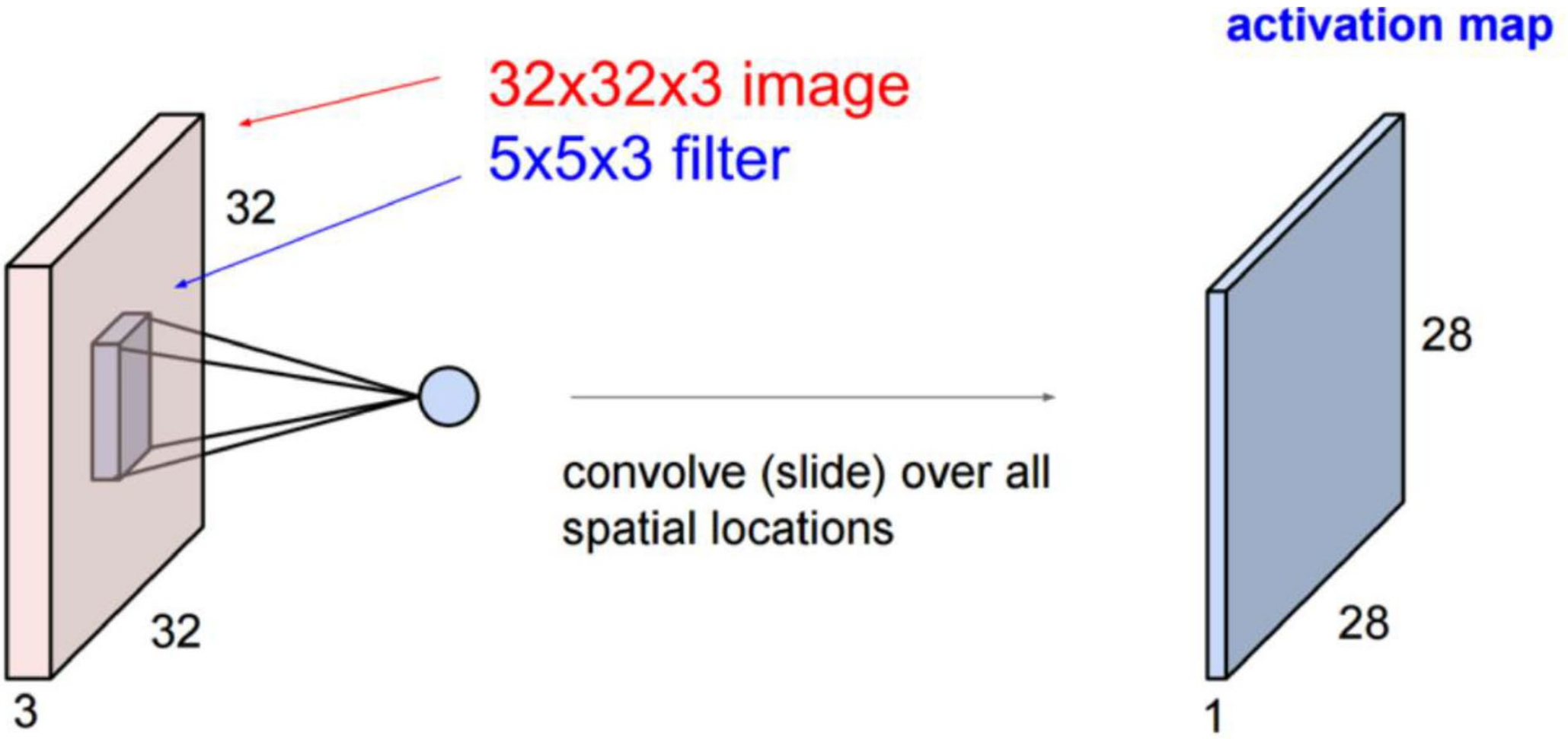


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product+bias)

$$\omega^t x + b$$

Recap

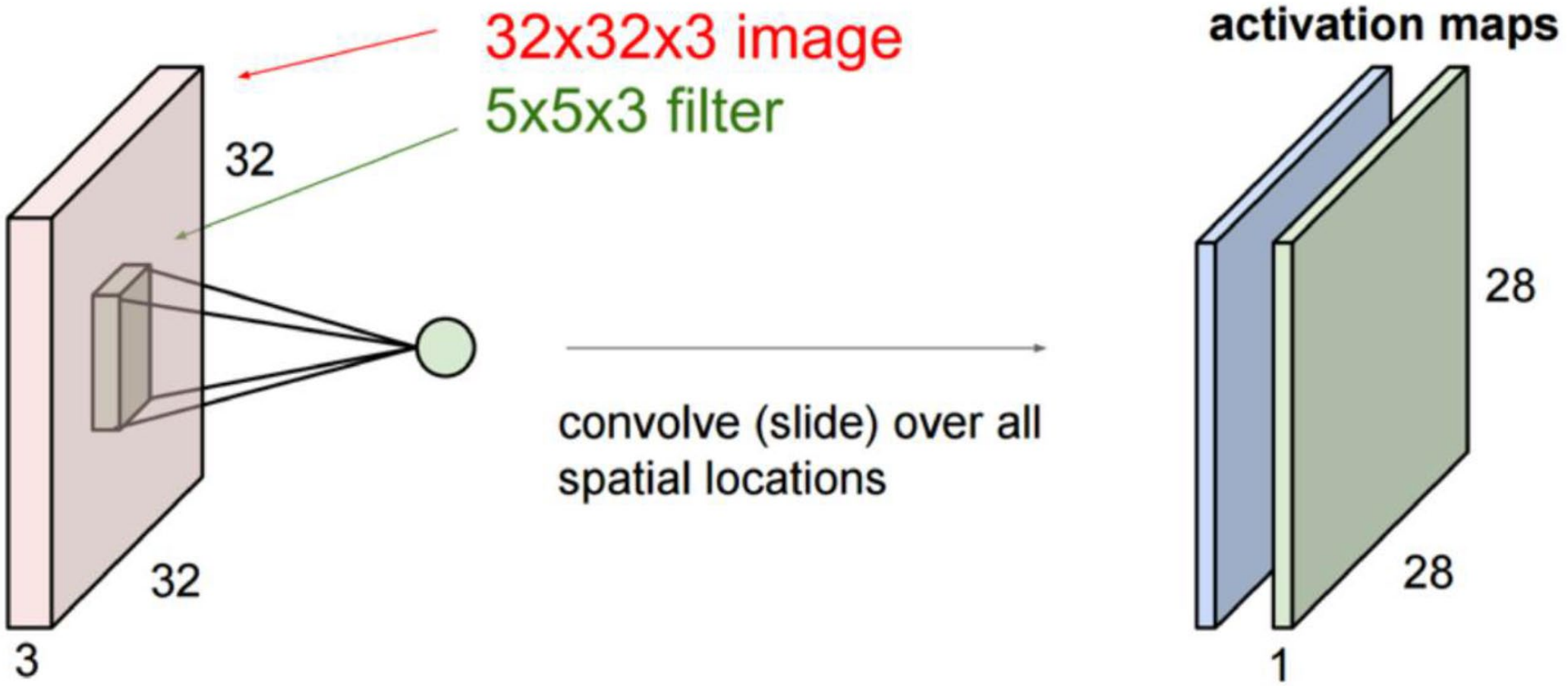
Convolution Layer



Recap

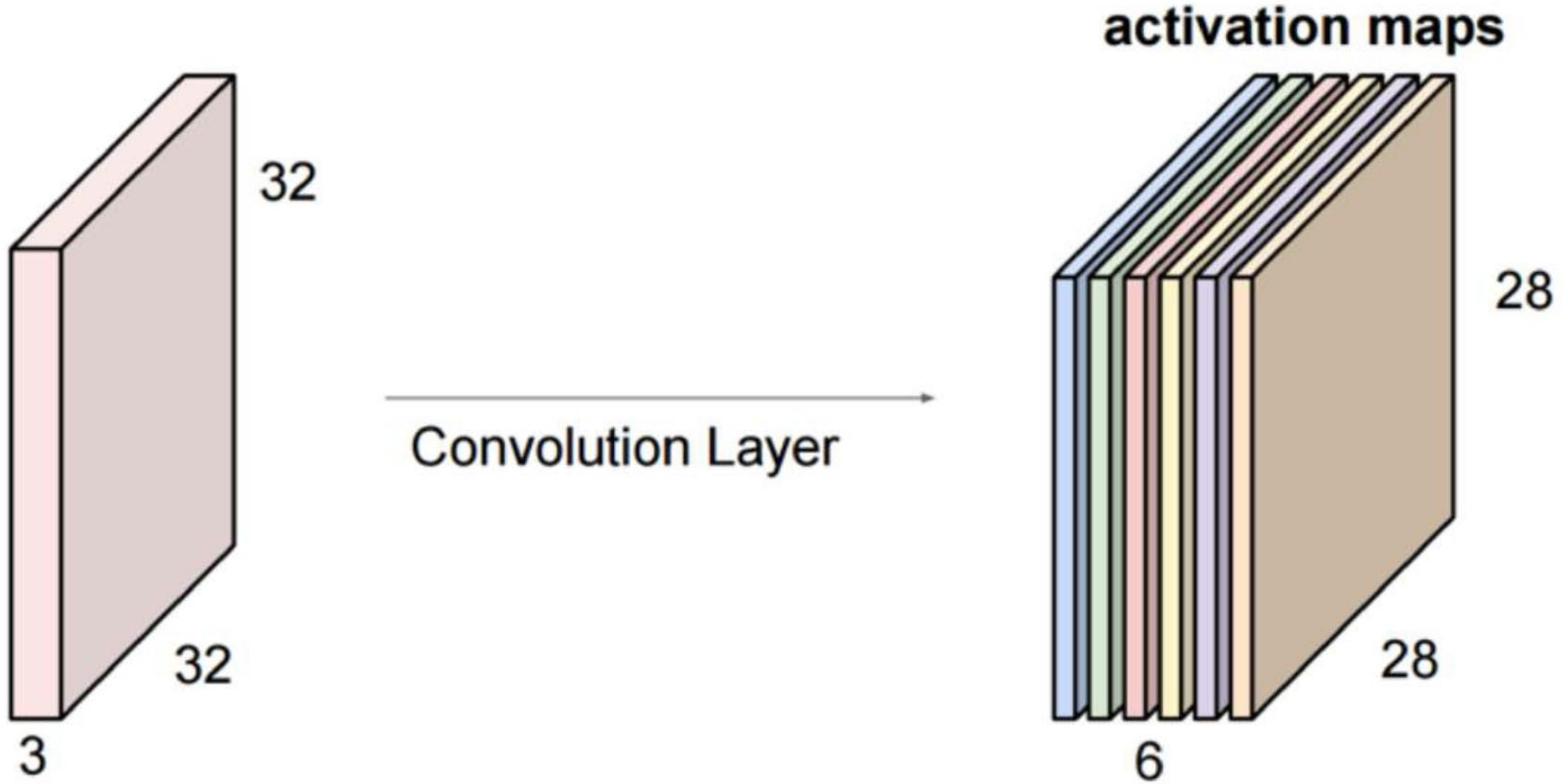
Convolution Layer

consider a second, green filter



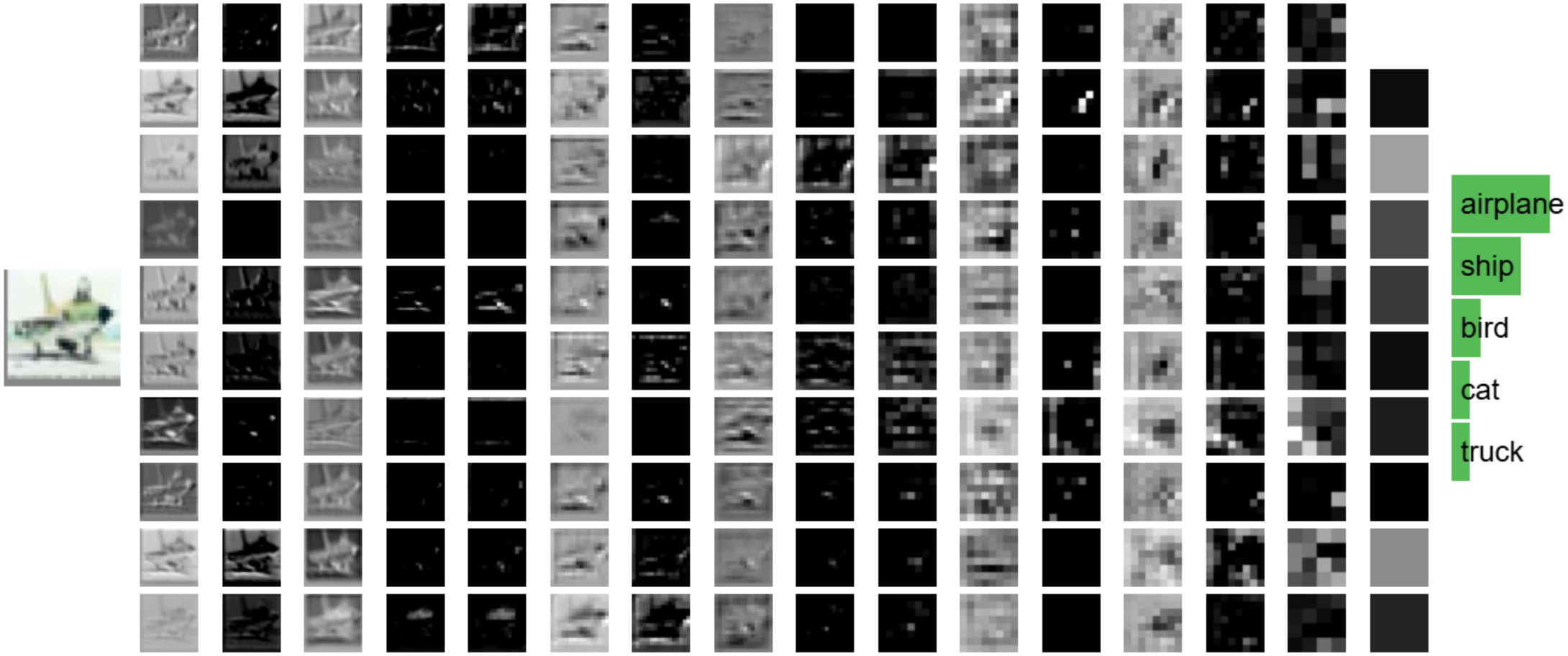
Recap

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

Demo



*This network is running live in your browser

http://cs231n.stanford.edu/