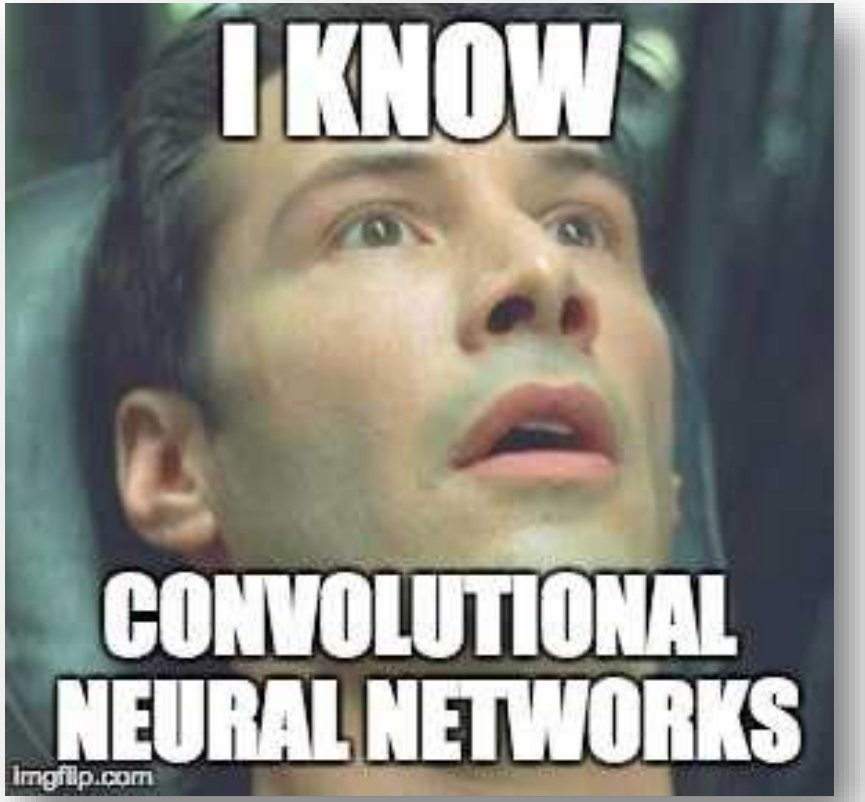
Reminders / Announcements

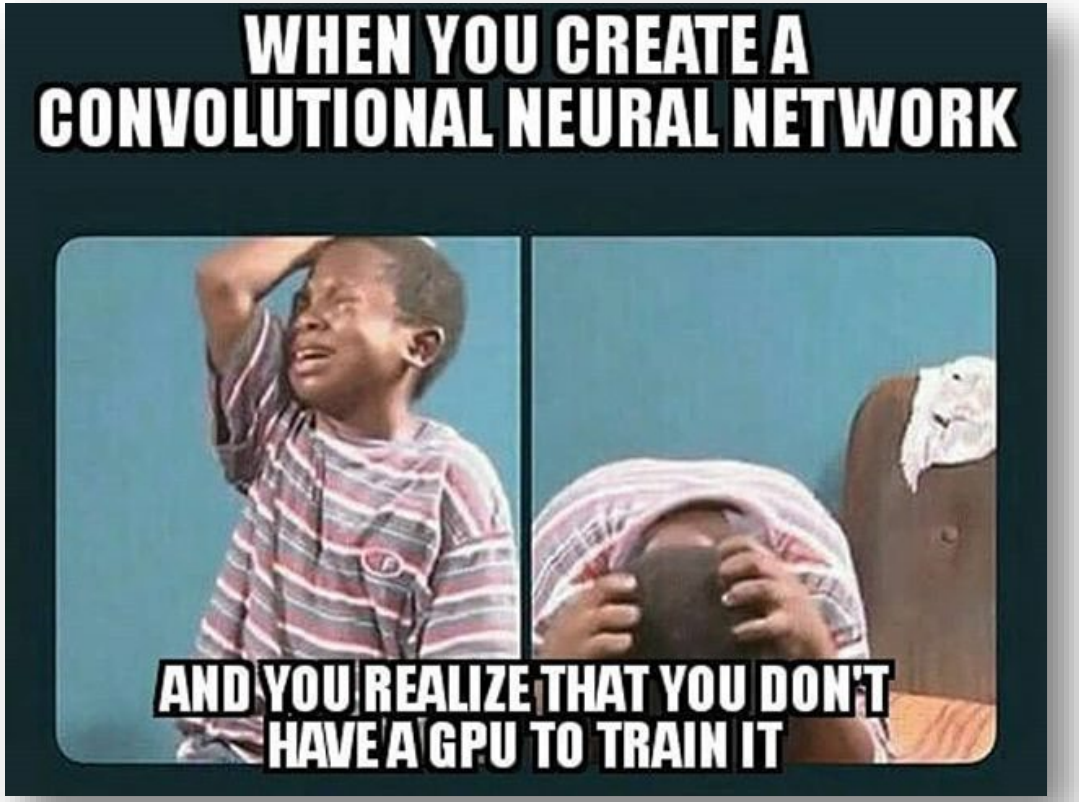
- **Project Proposal has been graded**
  - Goal: to give you detailed feedback. Grading is “light” (Min: 90%, Max: 99%)
  - Compute: Google Colab Pro is free for 1 year: https://blog.google/outreach-initiatives/education/colab-higher-education/
  - Oct 22: PyTorch tutorial in class (taught by Yu Liu)
- **Midterm Exam is on 10/20**
  - In class; closed-book; 1 hour; everything up to and including 10/15 lecture
  - More details in the next class.
- **Homework 1 is being graded (ETA ~1 week)**
- **Homework 2 will be released this weekend (and due ~ Nov 3)**

Your Project Topics are Cool !

- Self-supervised learning for medical imaging
- Combined stereo-mono depth estimation
- Interactive Video Generation
- Interactive Image Segmentation
- Anomaly Detection in Astrophysics
- Enhancing Robustness of AI Watermarking
- Fish Growth Monitoring via Vision
- Calorie/Nutrition Estimation from Images
- Vision-based Parking Monitoring
- Photogrammetry
- Detection of “AI-Generated” Images
- Embodied Visual Category Discovery
- Adversarial Defense in Event-Based Vision
- Paraboloid CNNs

Lecture 10
Convolutional Neural Networks






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

(Unrelated) Dog vs Food

[Karen Zack, @teenybiscuit]

(Unrelated) Dog vs Food

[Karen Zack, @teenybiscuit]

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

“AlexNet” — Won the ILSVRC2012 Challenge

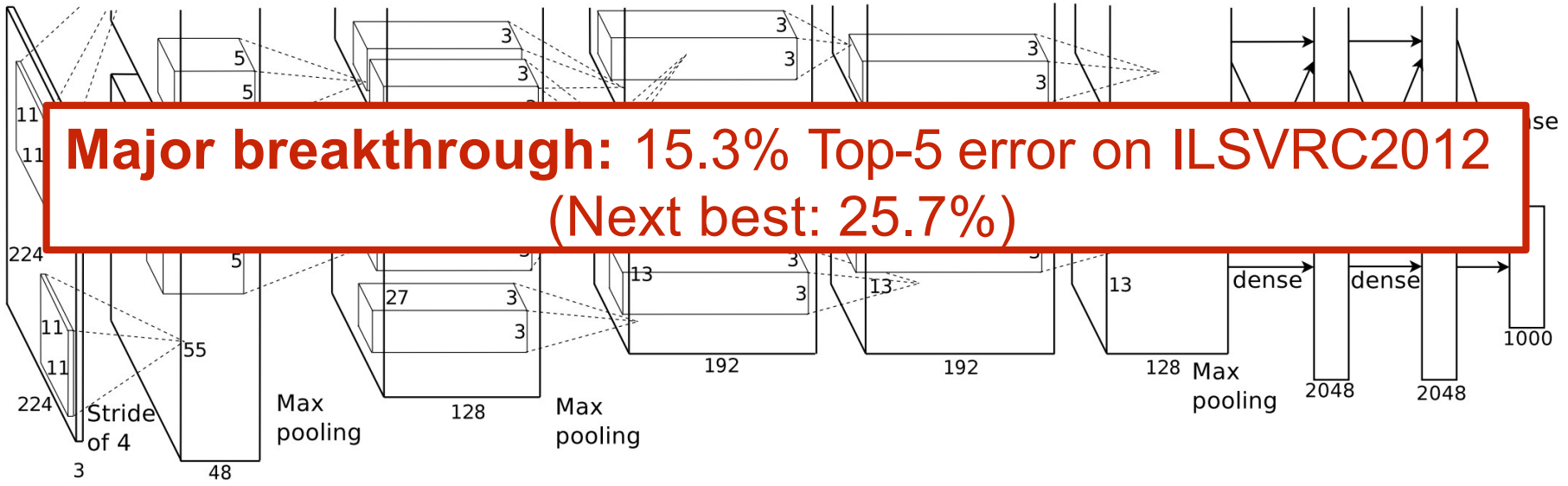
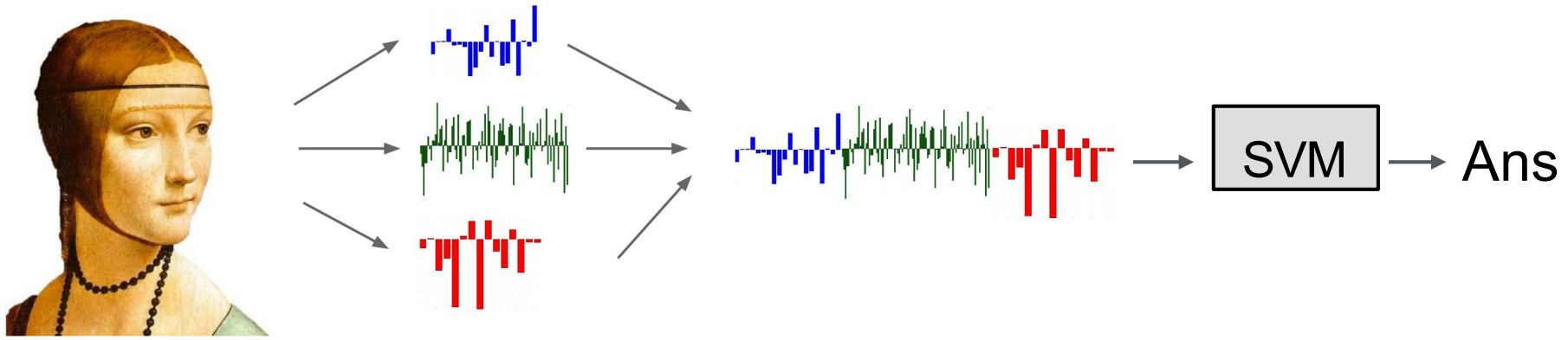


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input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

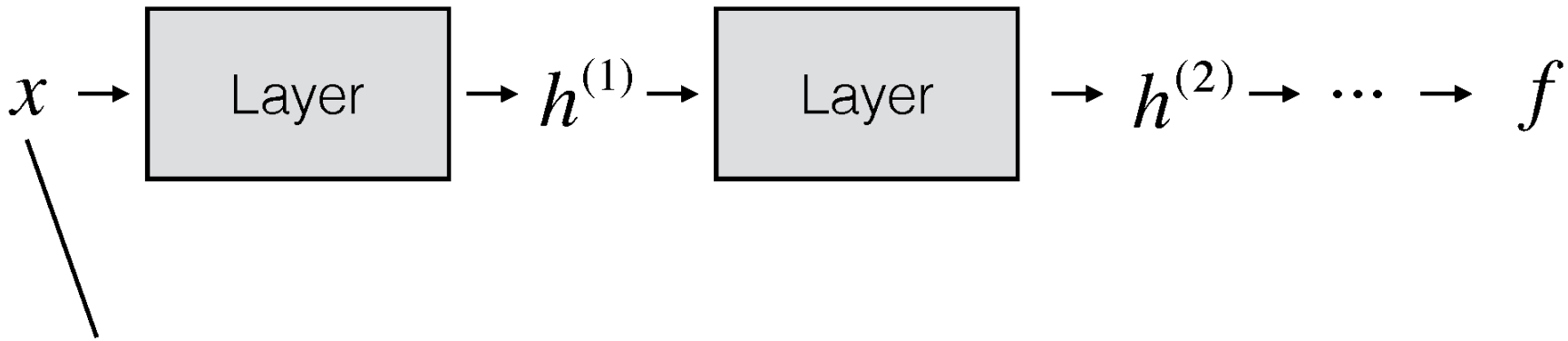
Perform everything with a big neural network, trained end-to-end

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

ConvNets

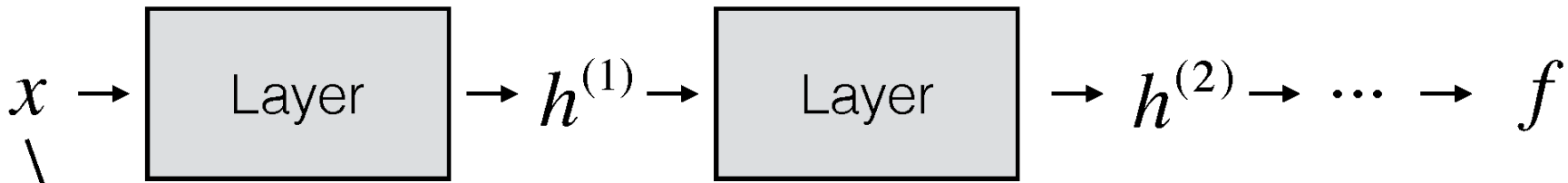
They’re just neural networks with
3D activations and weight sharing

What shape should the activations have?



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

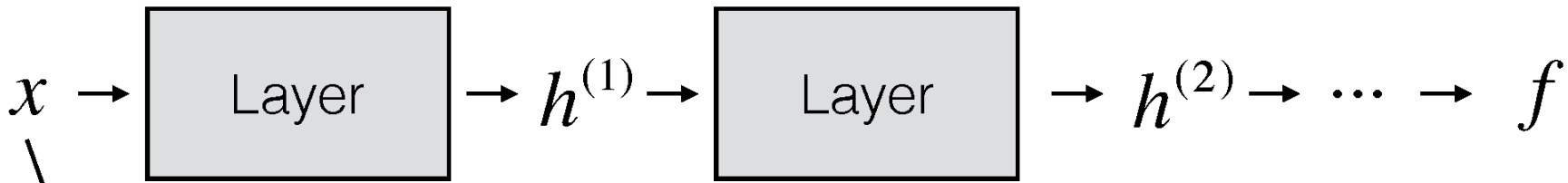
What shape should the activations have?



- 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

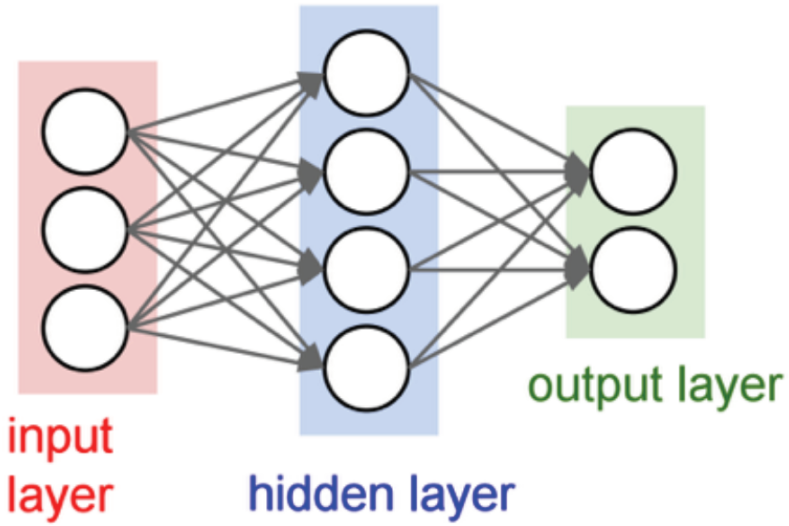
- What about keeping everything in 3D?

ConvNets

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

3D Activations

before:



(1D vectors)

Figure: Andrej Karpathy

3D Activations

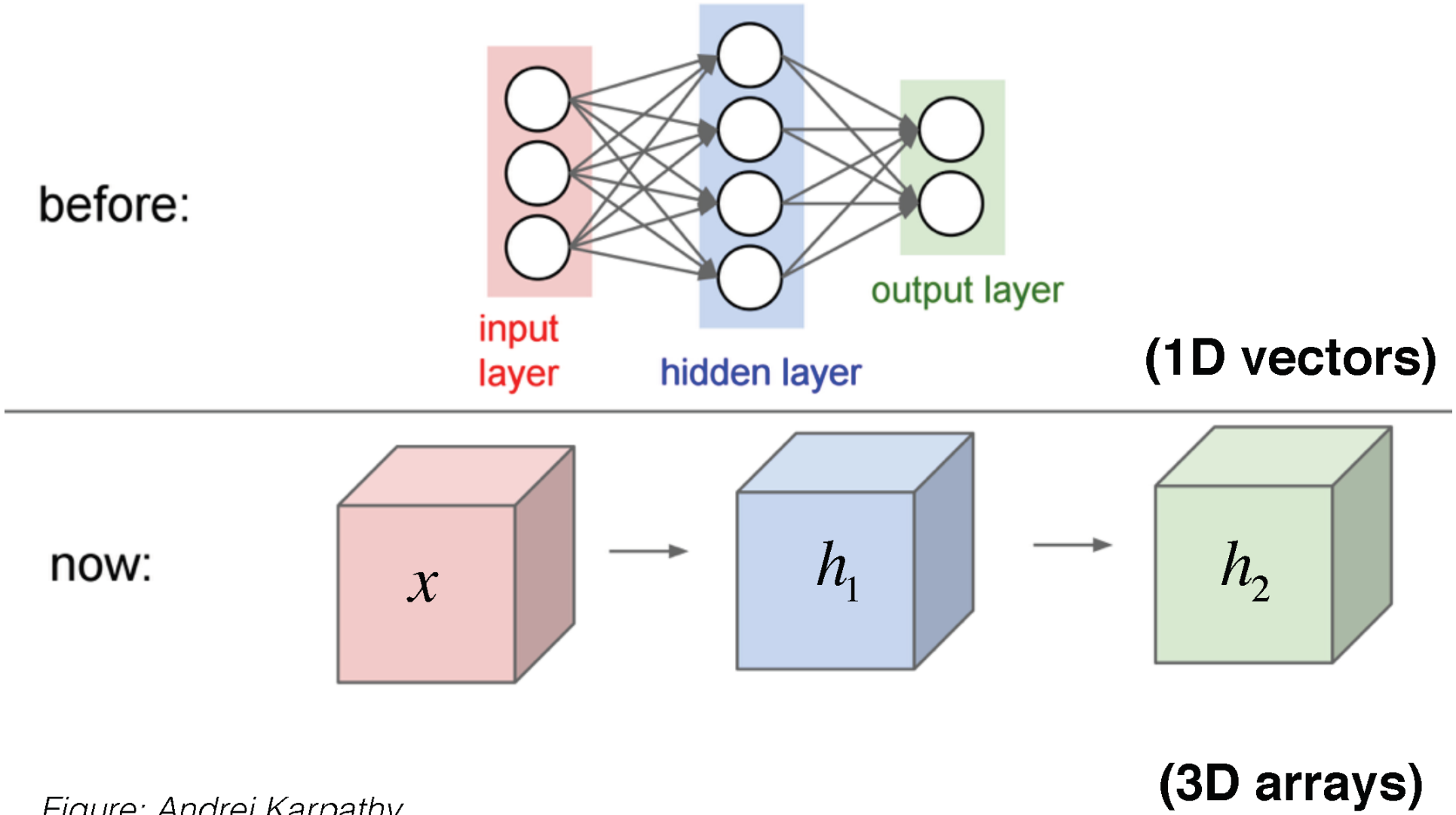


Figure: Andrej Karpathy

3D Activations

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

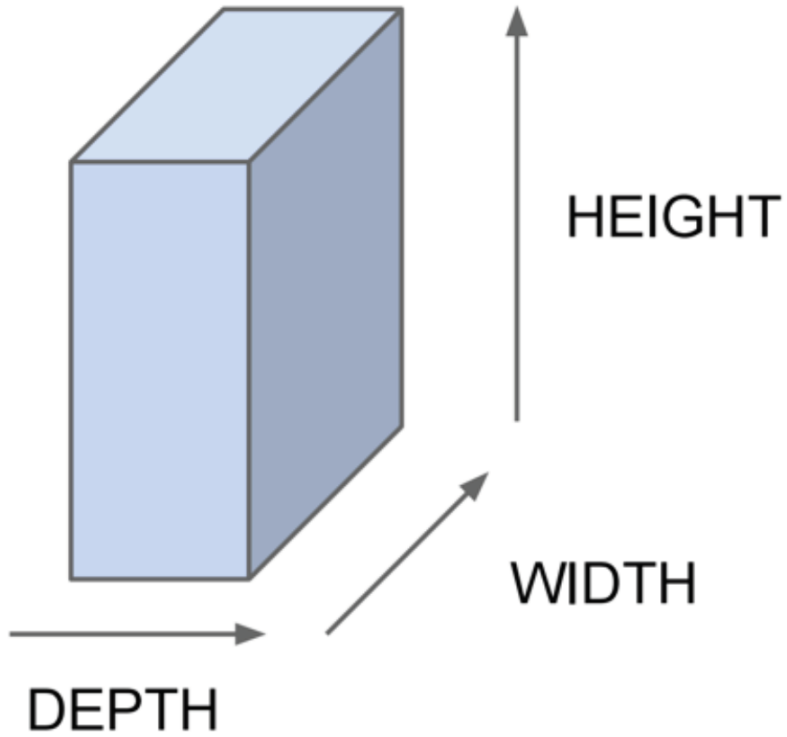
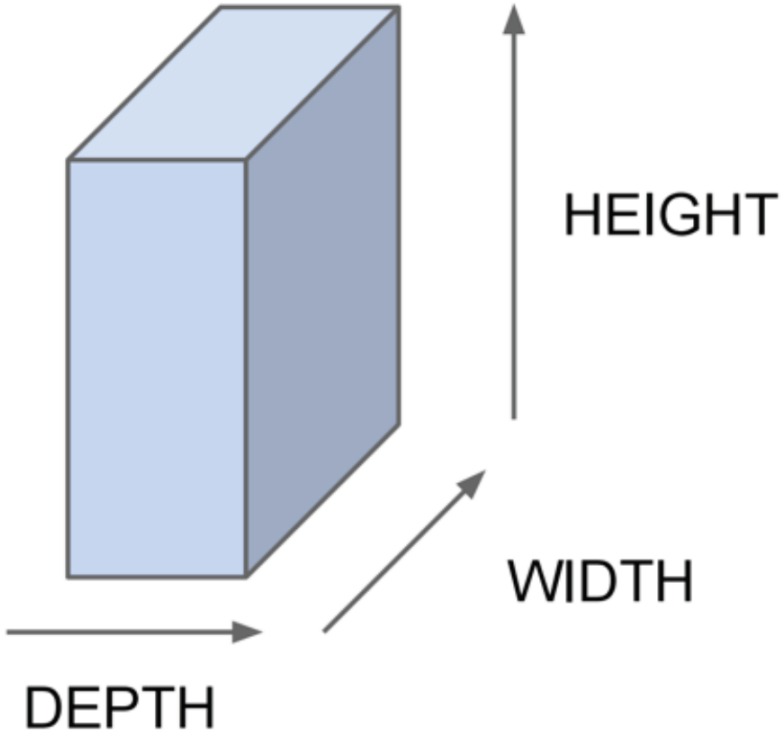


Figure: Andrej Karpathy

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)

Figure: Andrej Karpathy

3D Activations

1D Activations:

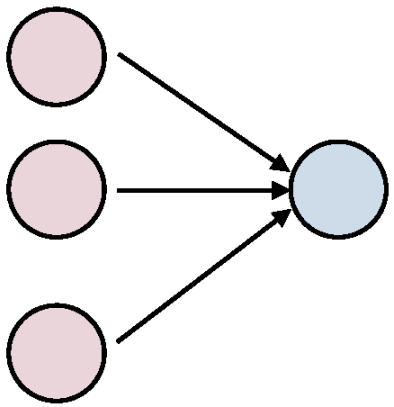
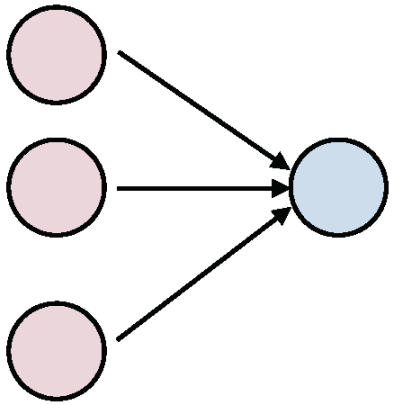


Figure: Andrej Karpathy

3D Activations

1D Activations:



3D Activations:

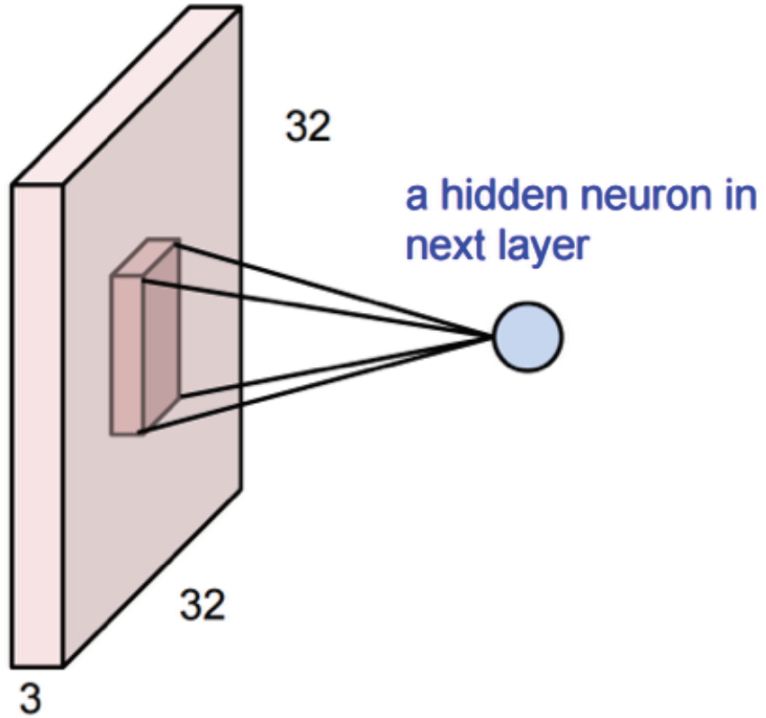
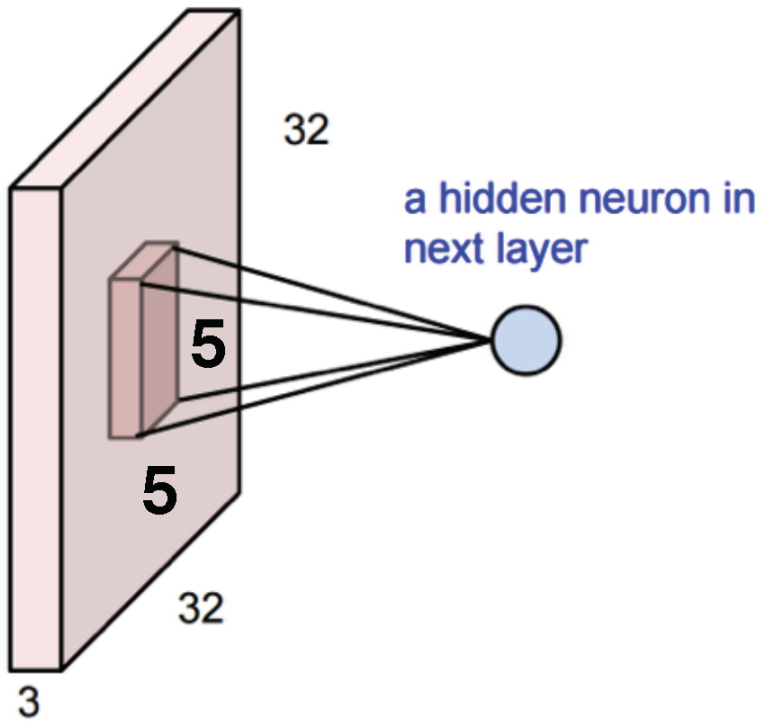


Figure: Andrej Karpathy

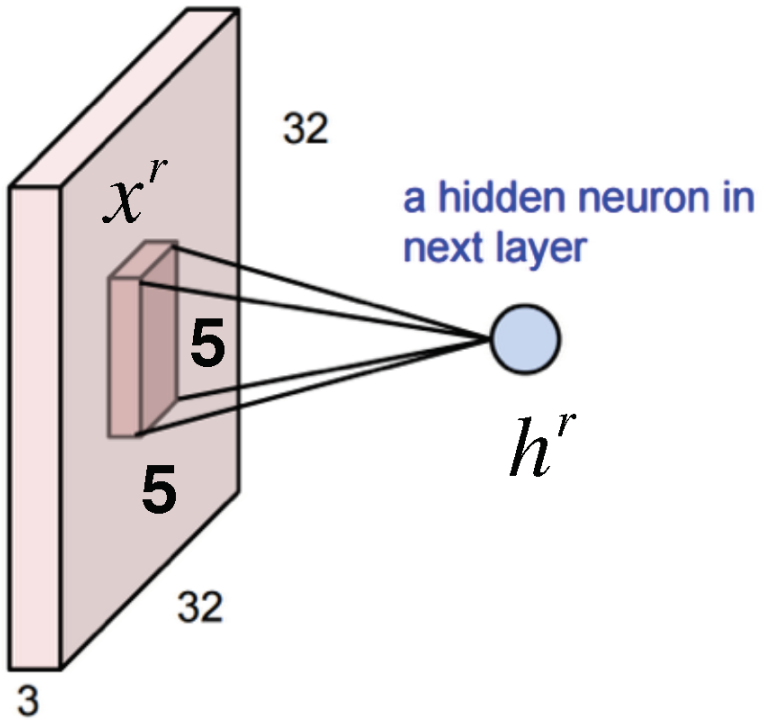
3D Activations



- 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)

Figure: Andrej Karpathy

3D Activations

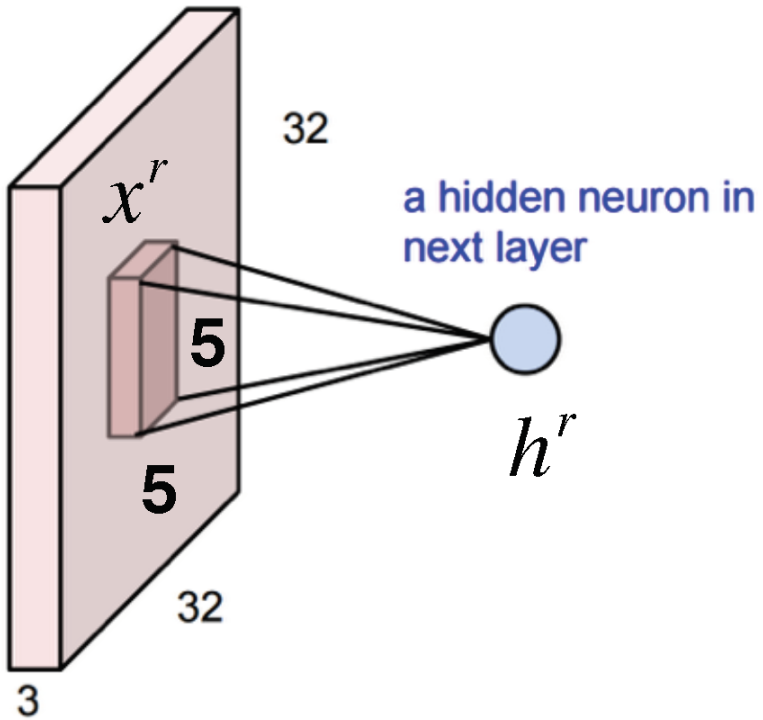


Example: consider the region of the input "x^r"

With output neuron h^r

Figure: Andrej Karpathy

3D Activations



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

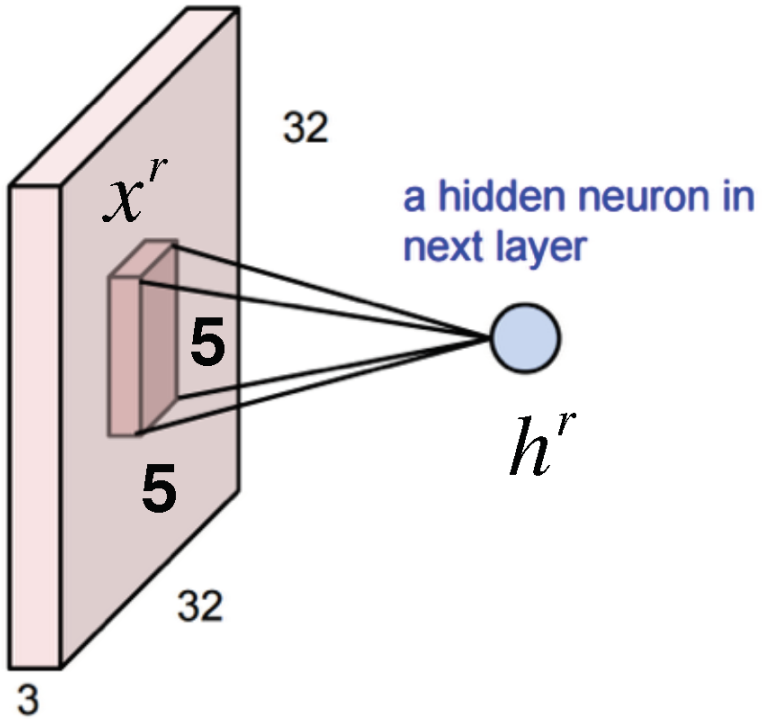
With output neuron h^r

Then the output is:

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

Figure: Andrej Karpathy

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

Figure: Andrej Karpathy

3D Activations

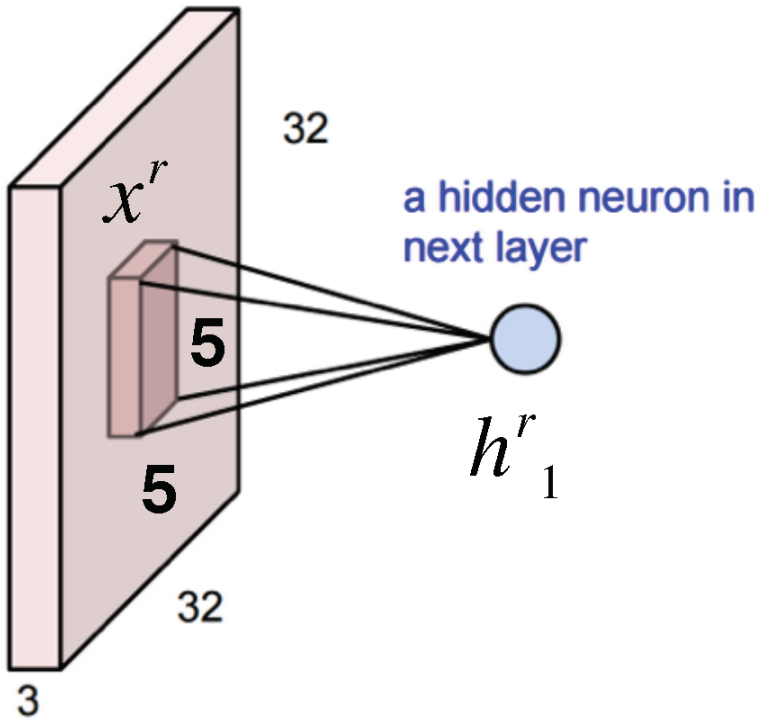


Figure: Andrej Karpathy

3D Activations

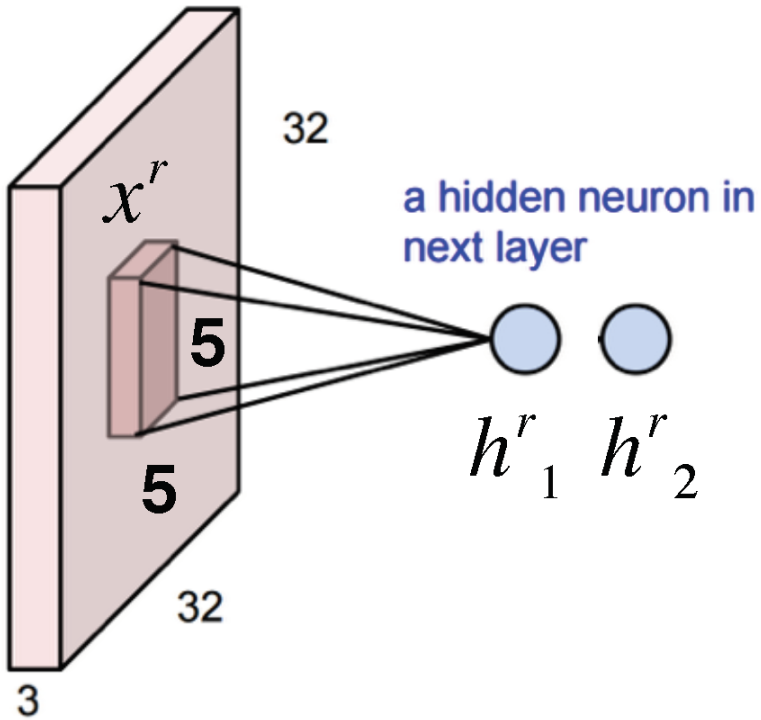
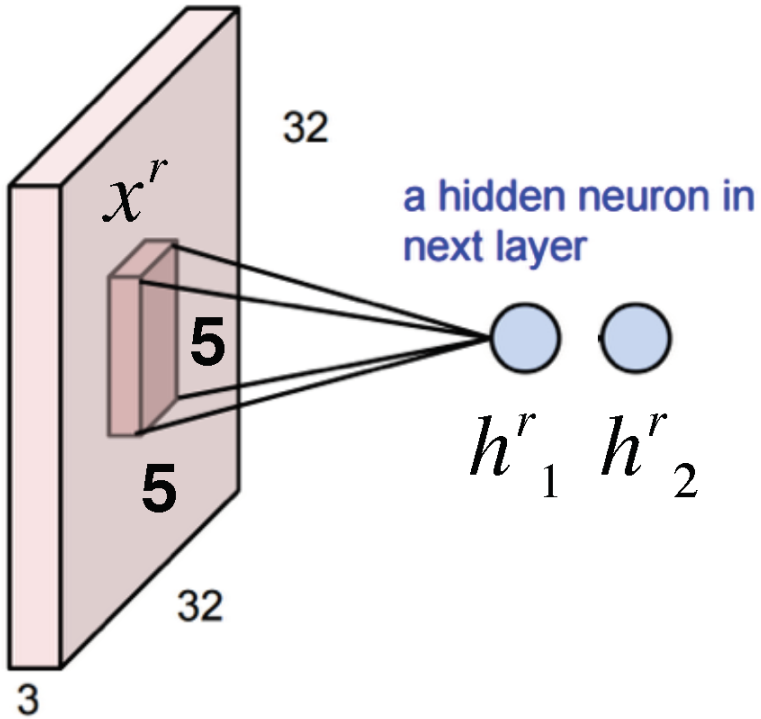


Figure: Andrej Karpathy

3D Activations



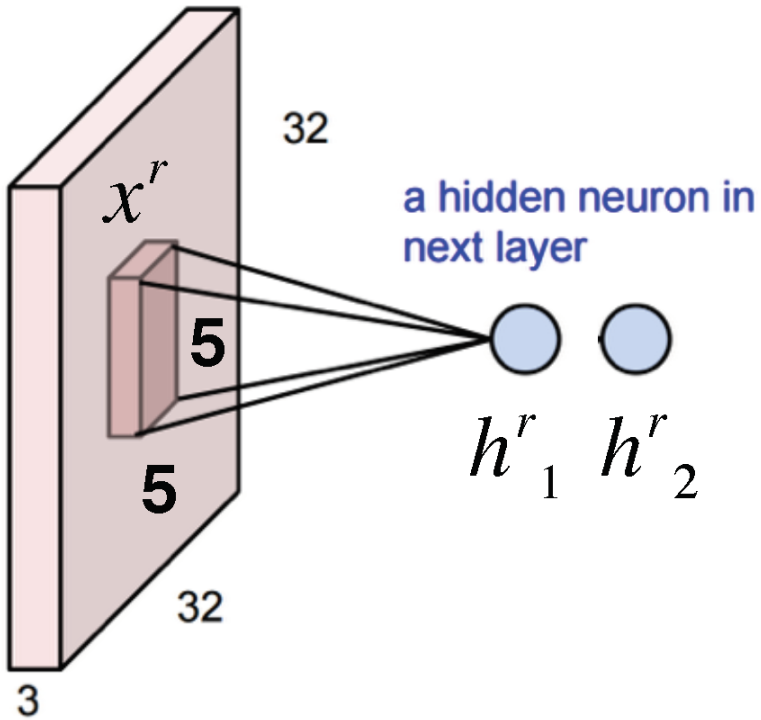
With **2** output neurons

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

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

Figure: Andrej Karpathy

3D Activations



With **2** output neurons

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

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

Figure: Andrej Karpathy

3D Activations

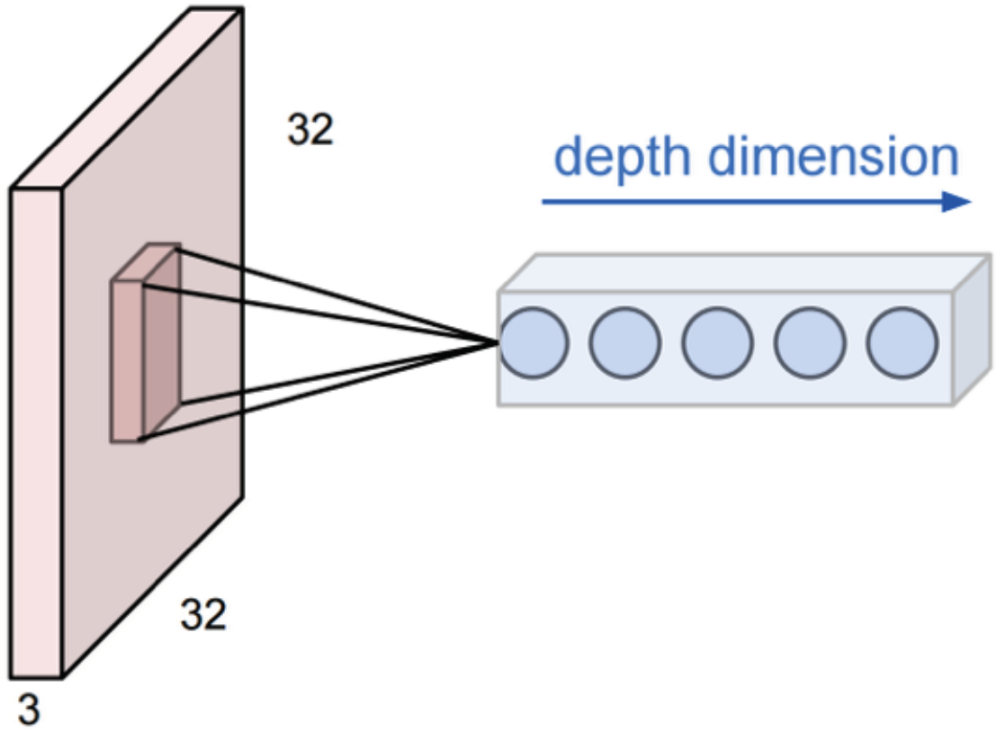
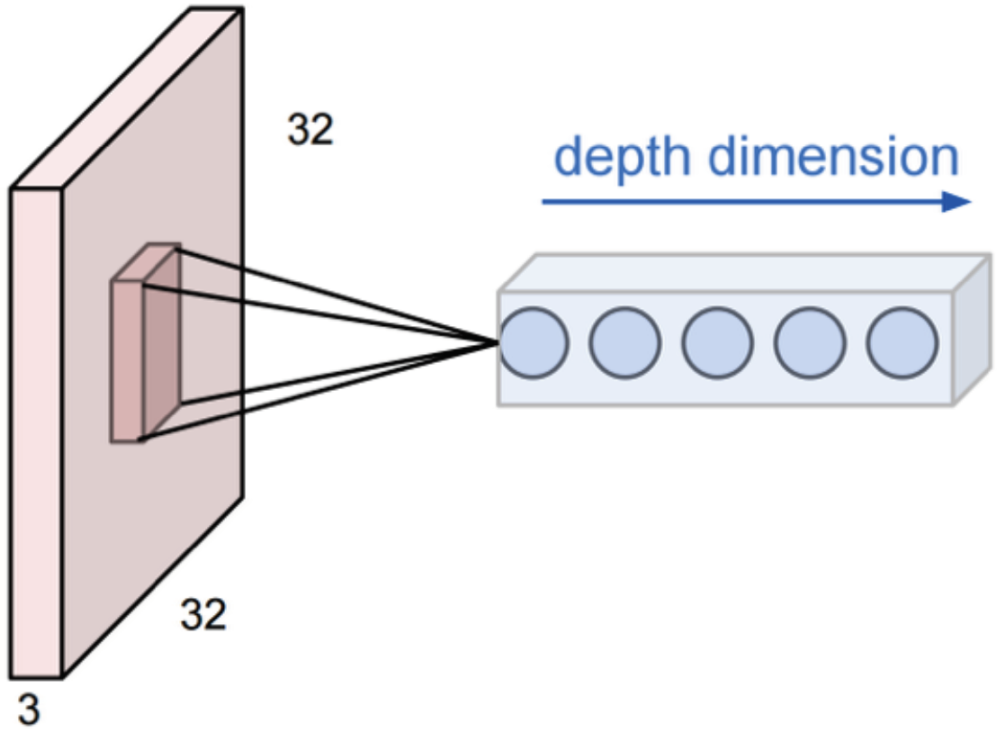


Figure: Andrej Karpathy

3D Activations

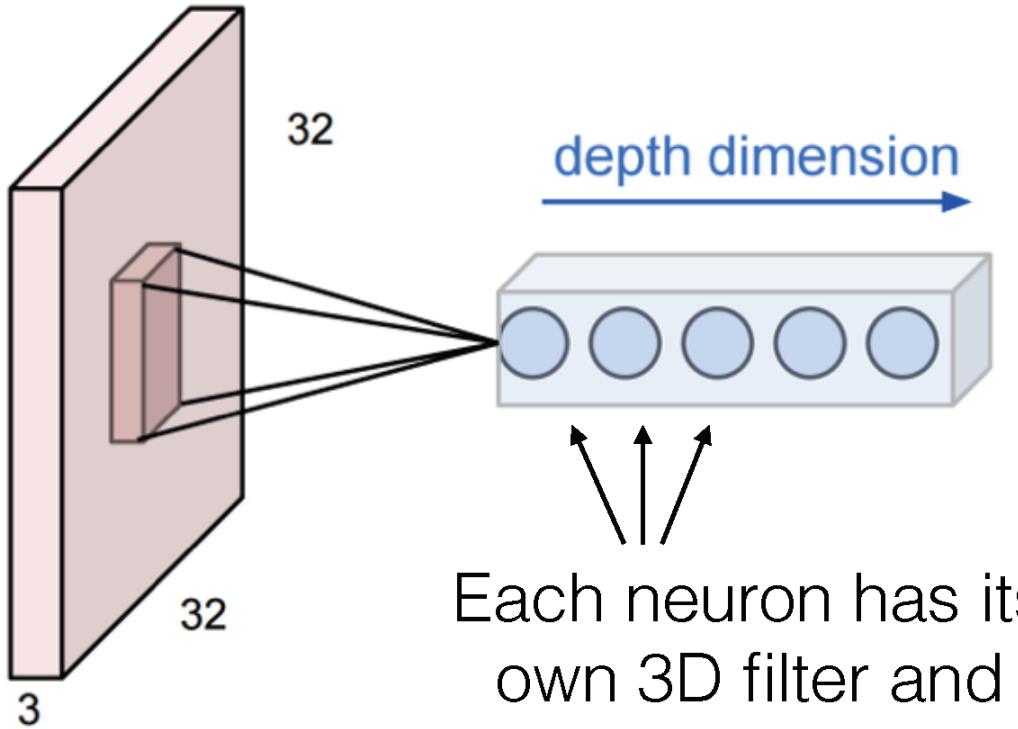


We can keep adding more outputs

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

Figure: Andrej Karpathy

3D Activations



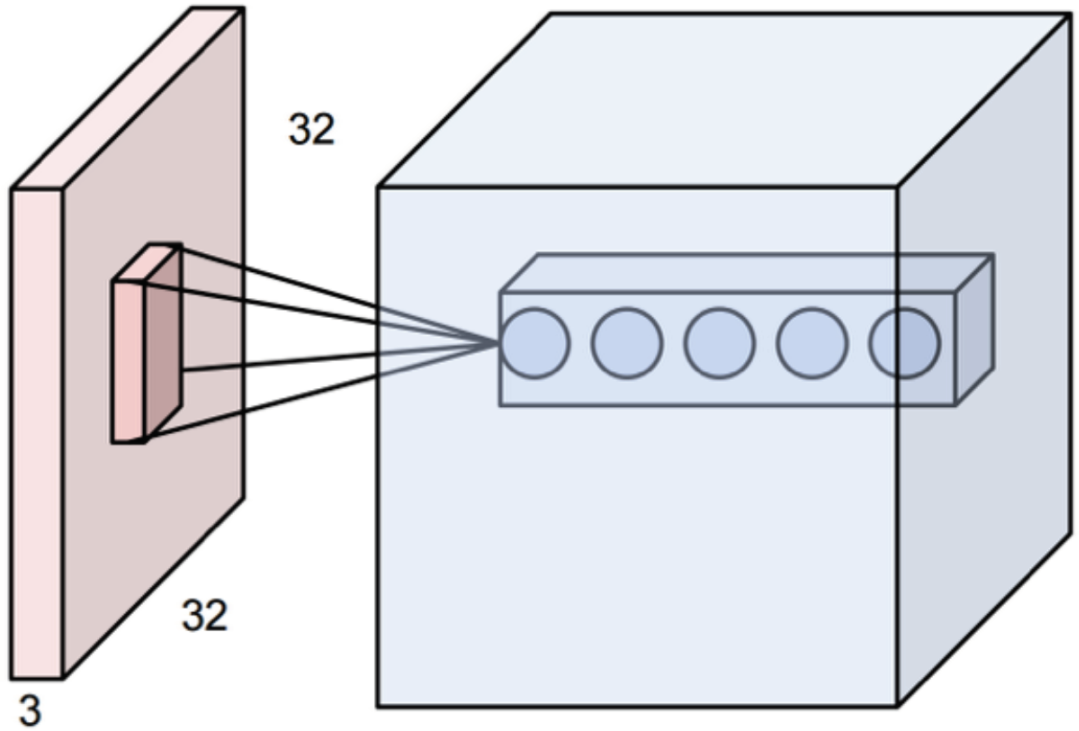
Each neuron has its own 3D filter and own (scalar) bias

We can keep adding more outputs

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

Figure: Andrej Karpathy

3D Activations

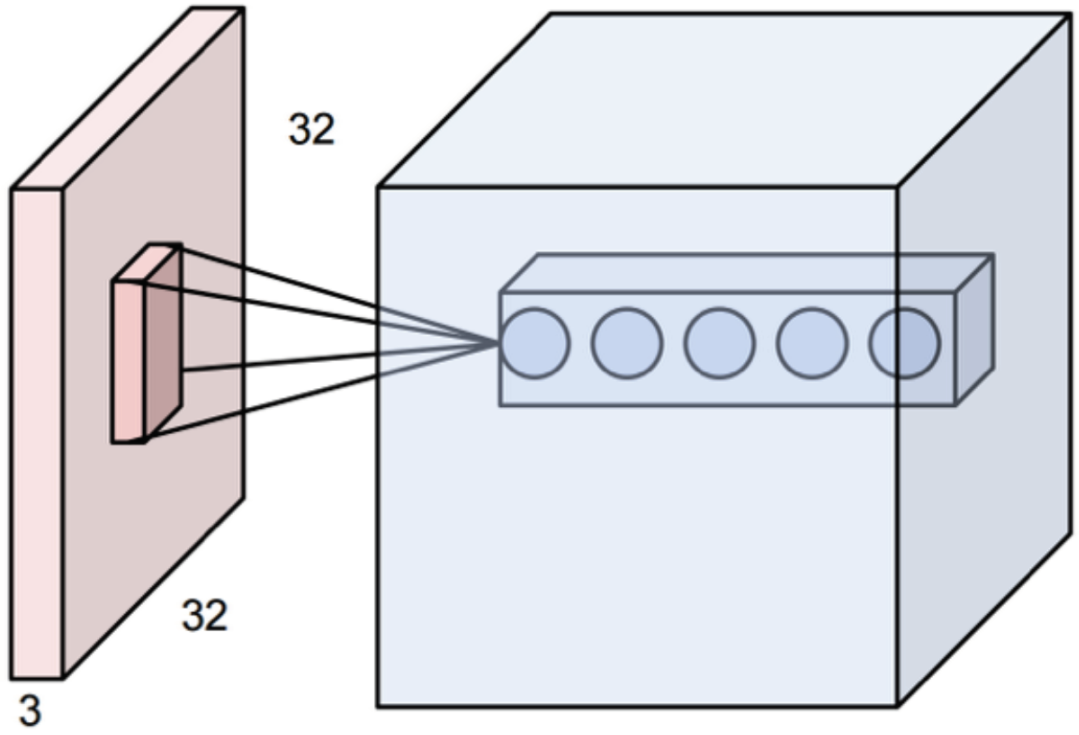


Now repeat this
across the input

D sets of weights
(also called filters)

Figure: Andrej Karpathy

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)

Figure: Andrej Karpathy

3D Activations

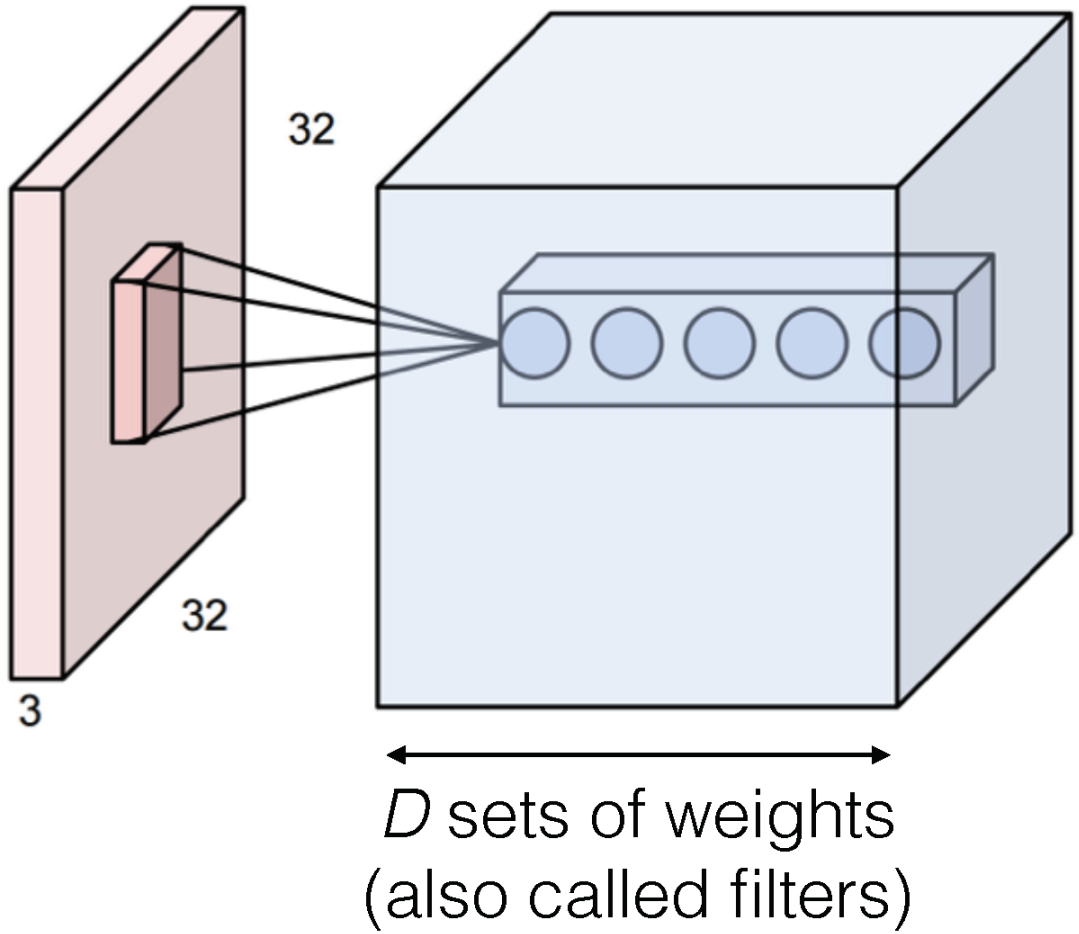
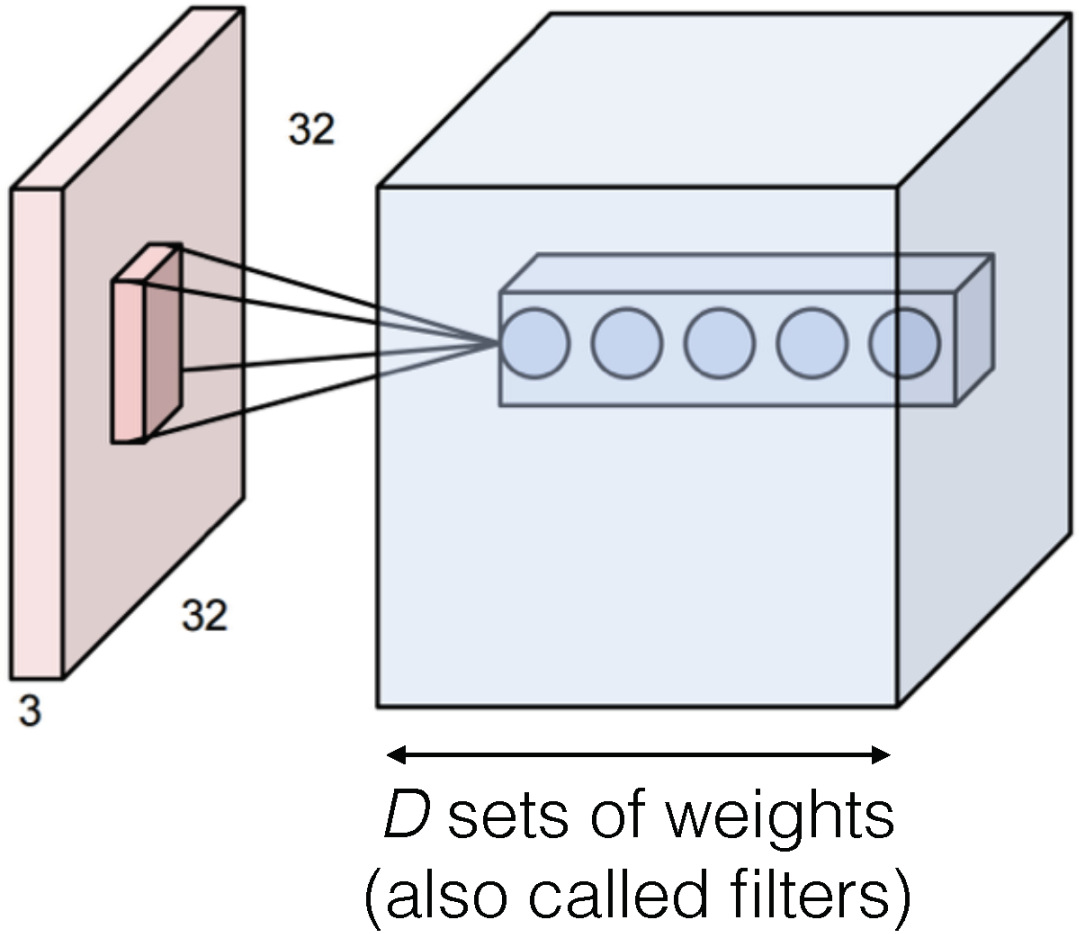


Figure: Andrej Karpathy

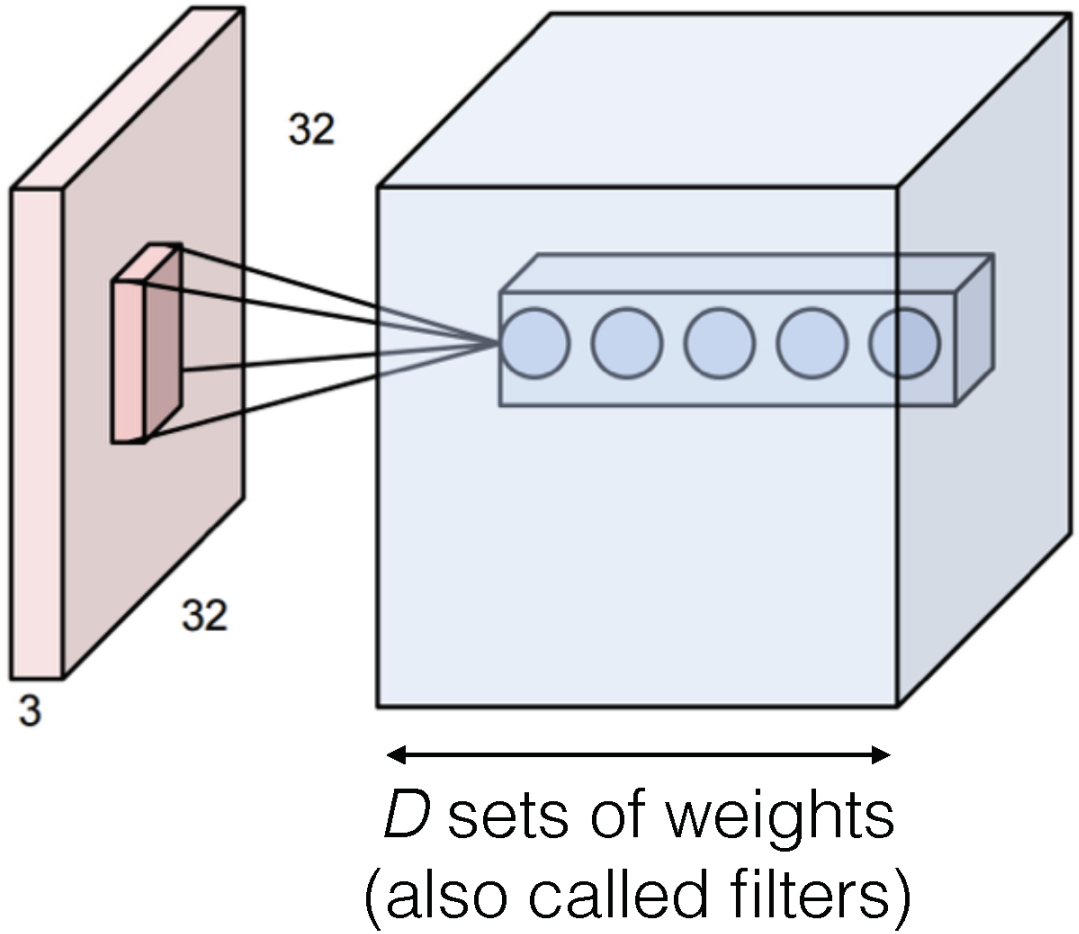
3D Activations



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

Figure: Andrej Karpathy

3D Activations

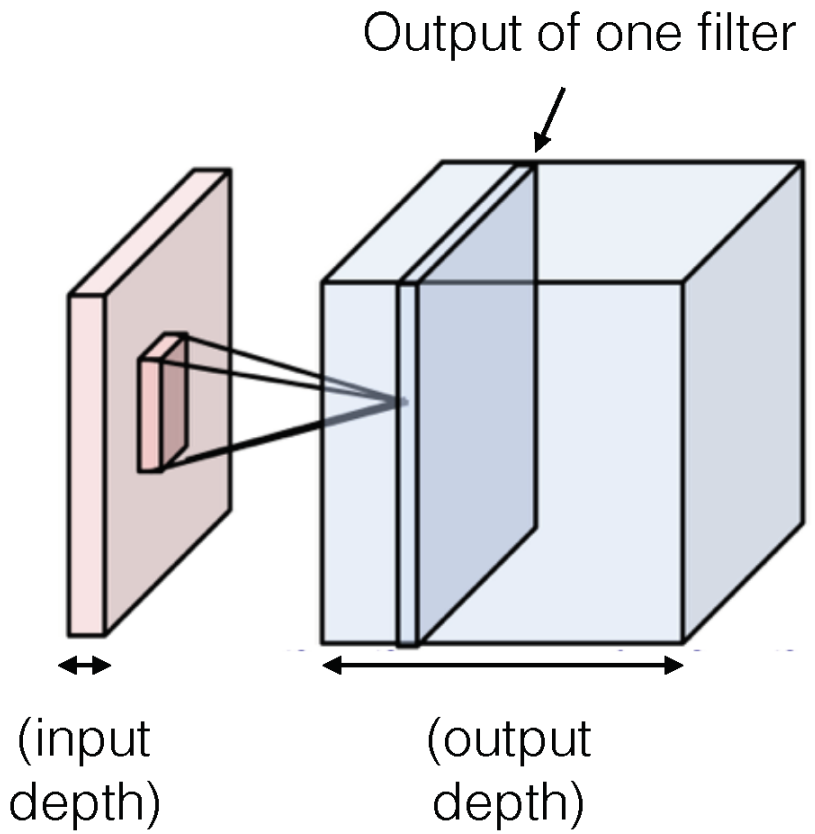


With weight sharing, this is called **convolution**

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

Figure: Andrej Karpathy

3D Activations

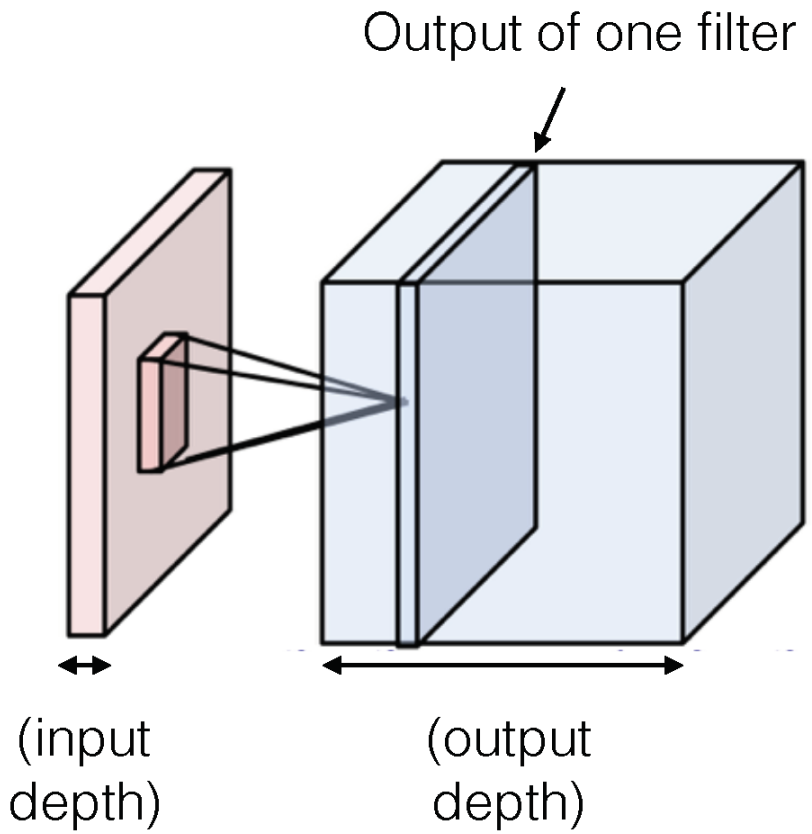


One set of weights gives one slice in the output

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)

Fixed Filters

original

Vertical Sobel filter:

1	2	1
0	0	0
-1	-2	-1

CONVOLUTION

vertical Sobel filter

Learned Filters

original

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

CONVOLUTION

Learned Features

3D Activations

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

(Input)

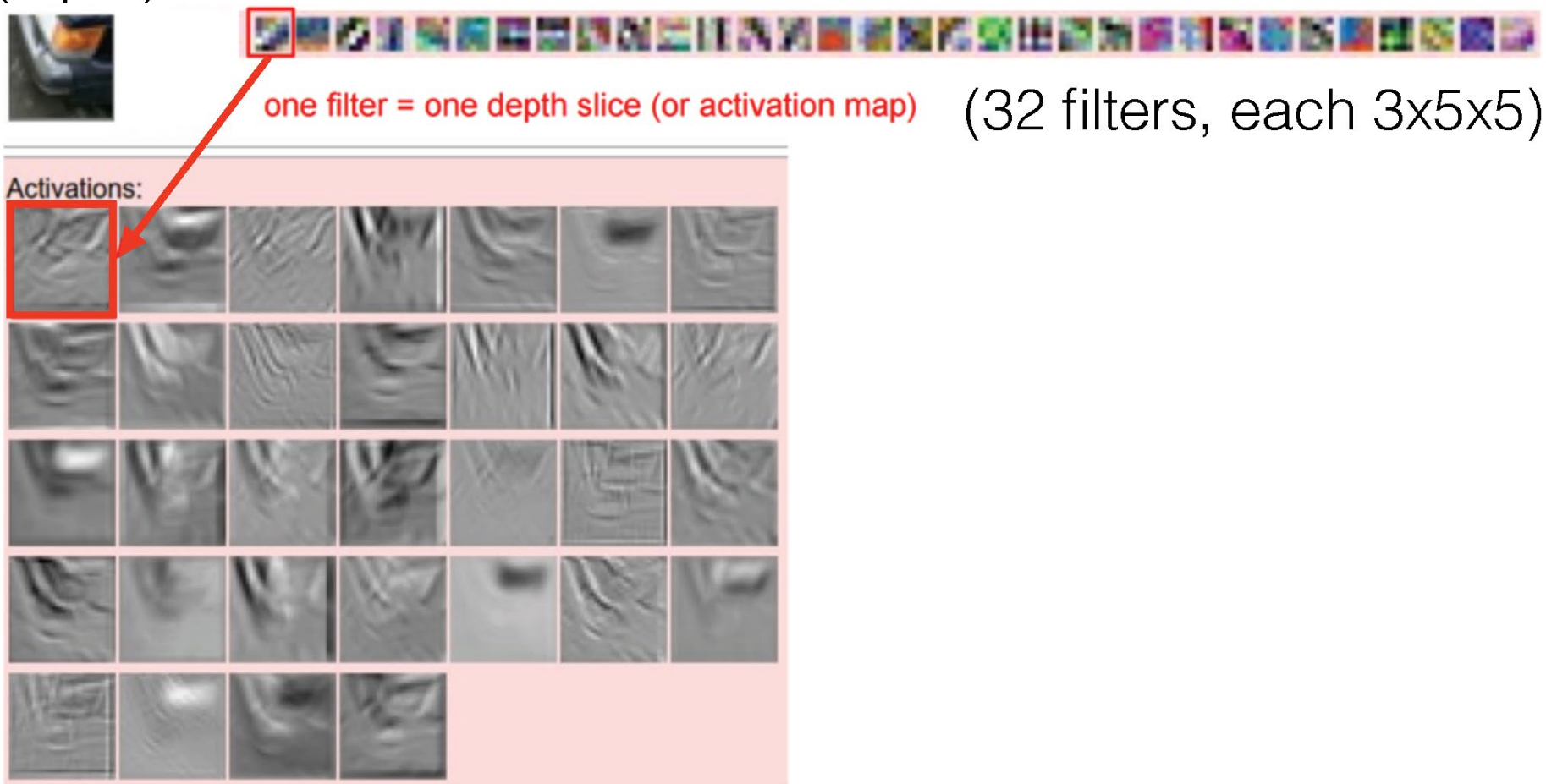


Figure: Andrej Karpathy

3D Activations

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

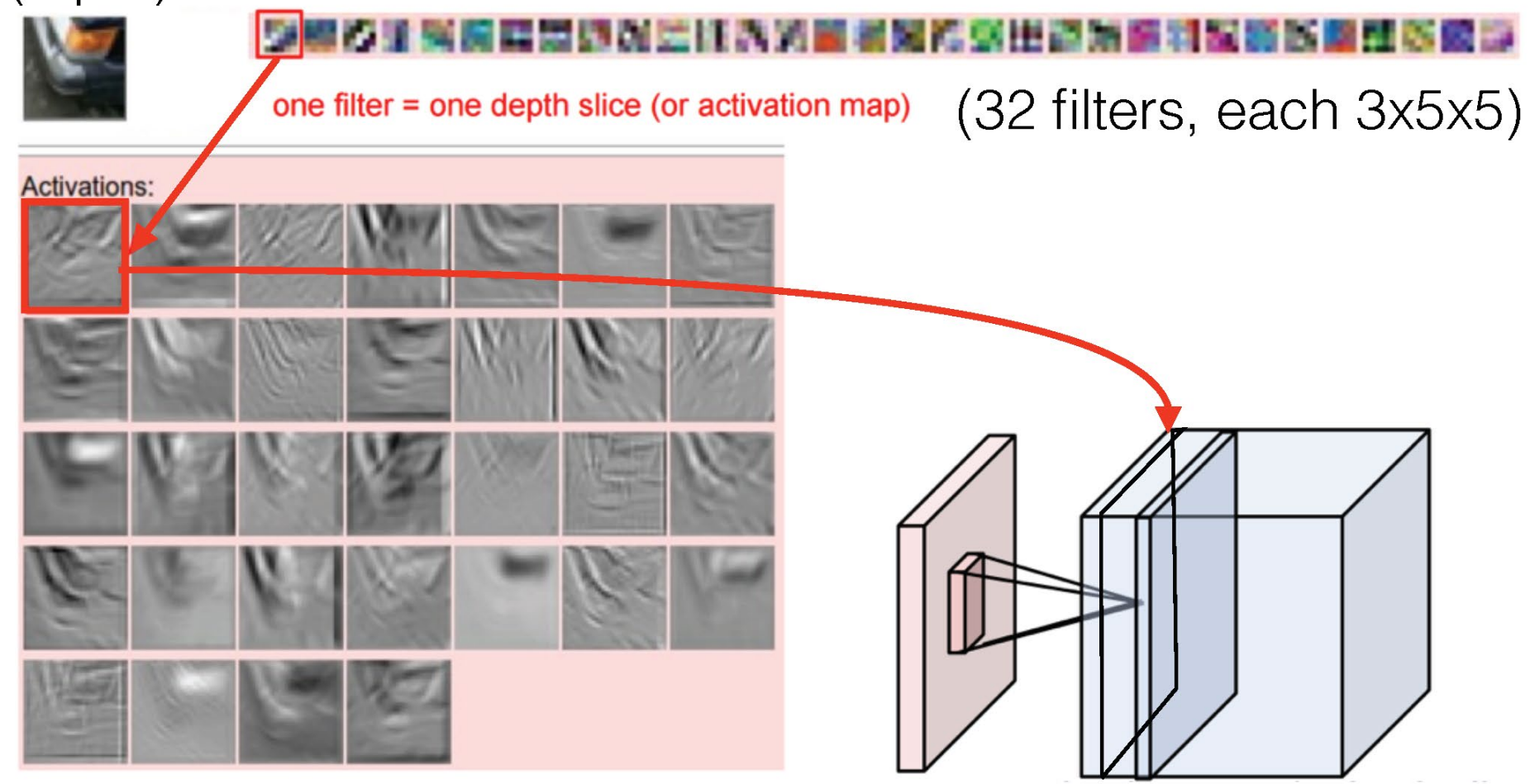


Figure: Andrej Karpathy

3D Activations

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

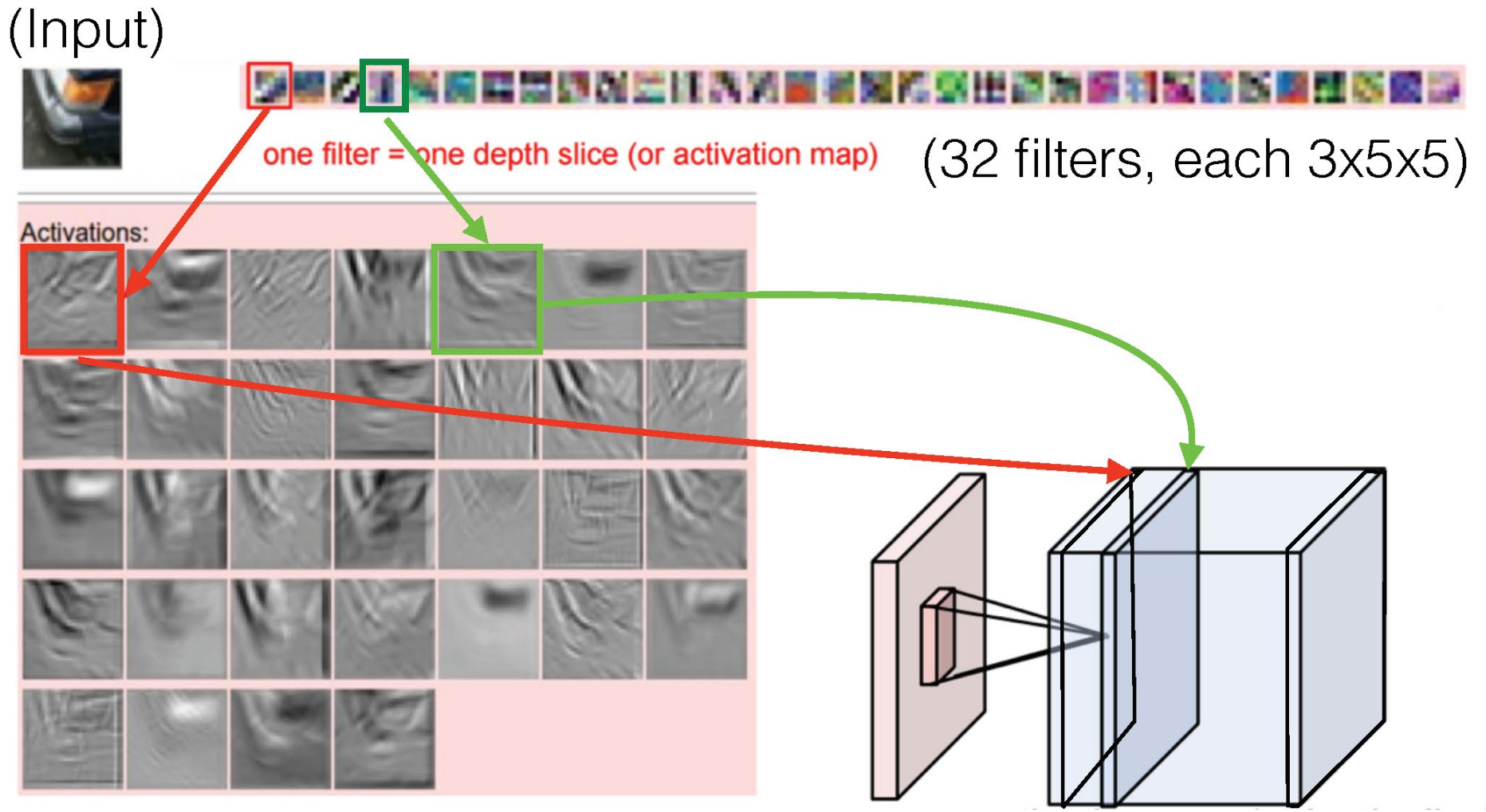


Figure: Andrej Karpathy

3D Activations

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

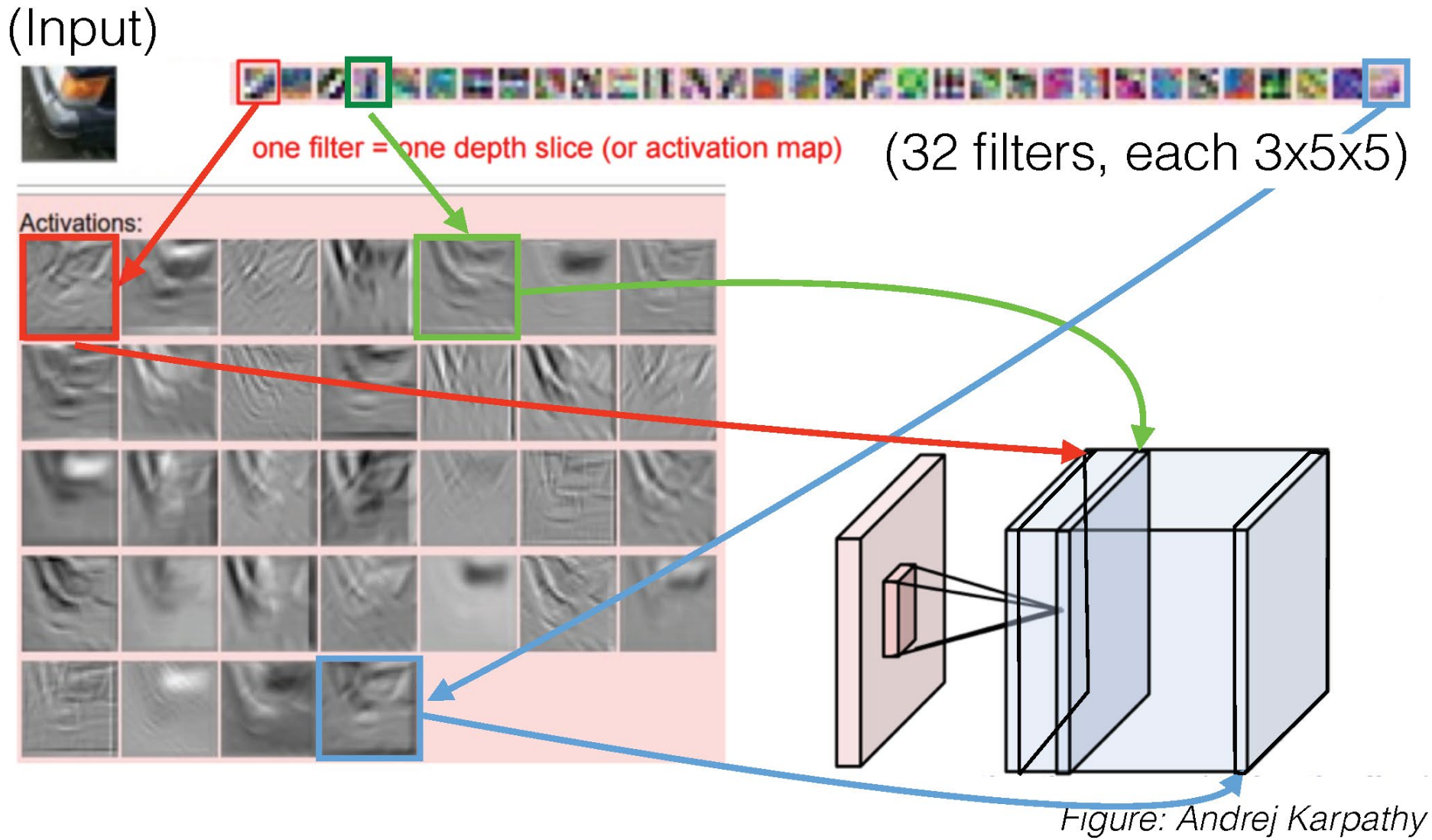
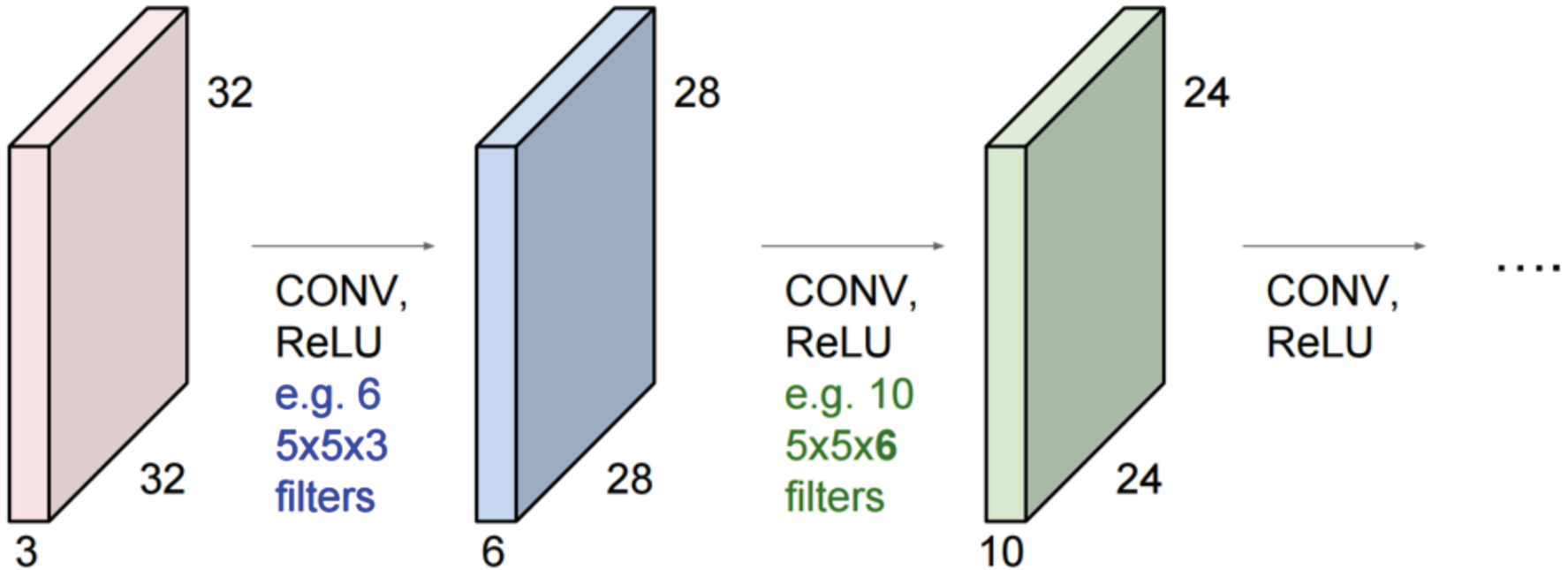
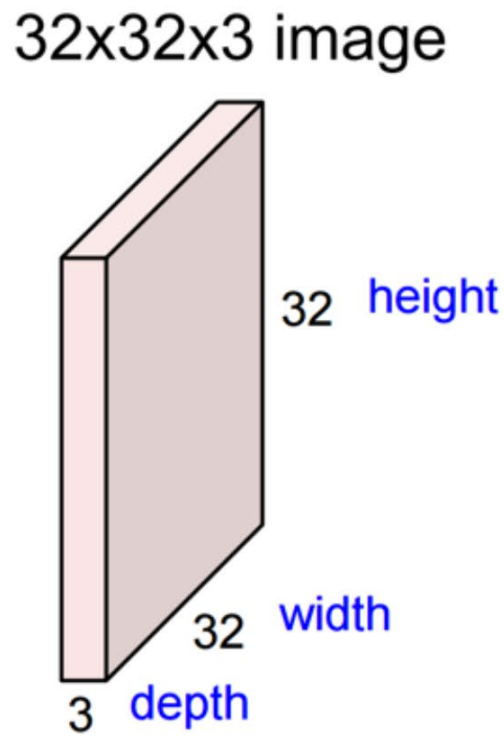


Figure: Andrej Karpathy

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

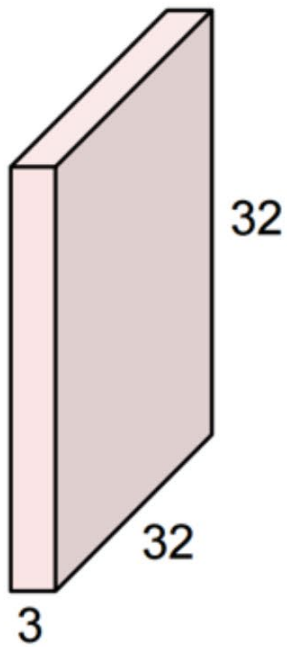


Convolution Layer

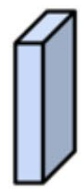


Convolution Layer

32x32x3 image



5x5x3 filter

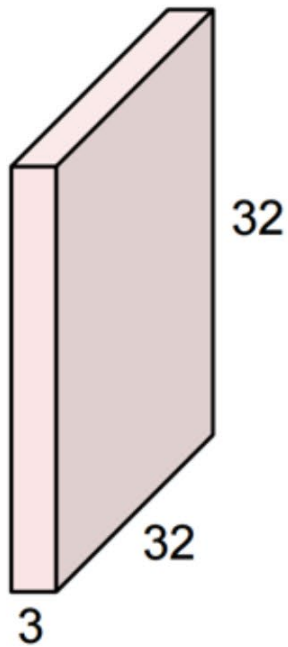


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

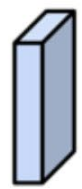
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

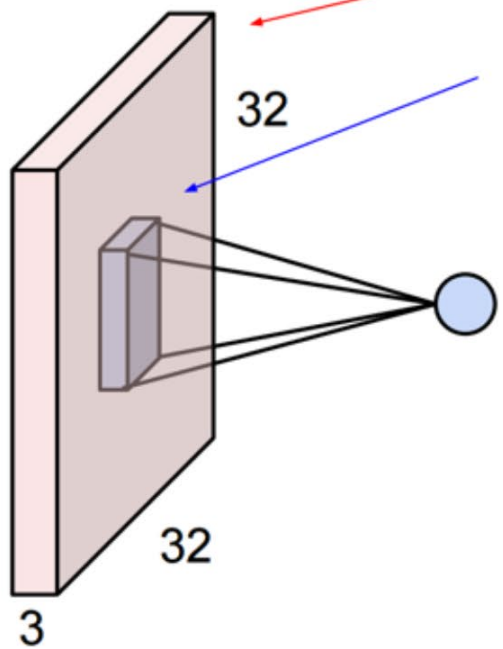


5x5x3 filter



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

Convolution Layer



32x32x3 image

5x5x3 filter w

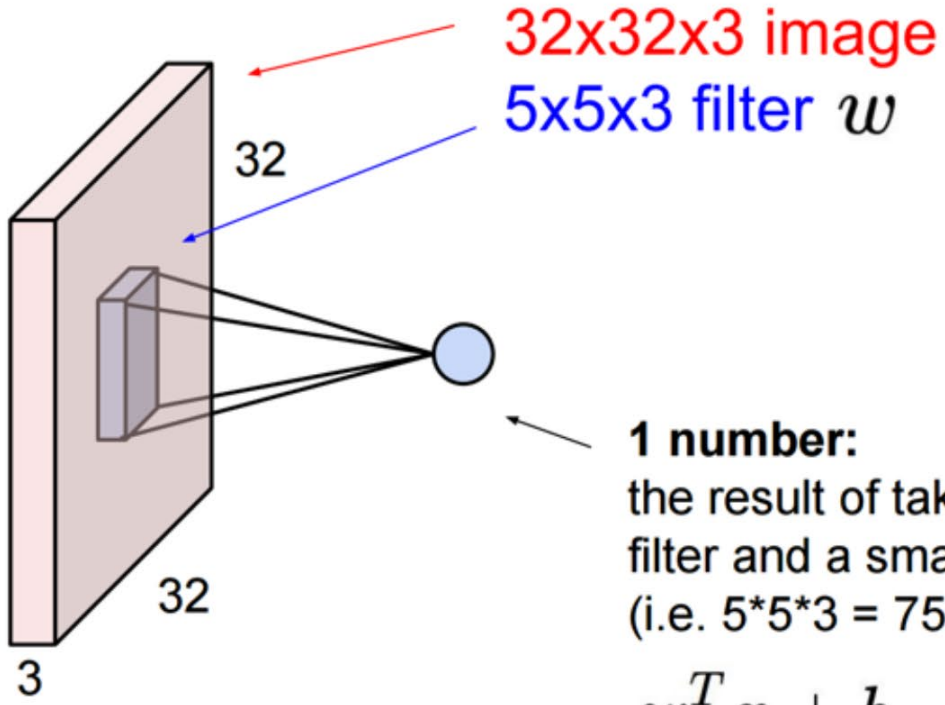
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

What will the output size be?

You will need to make some assumptions ...

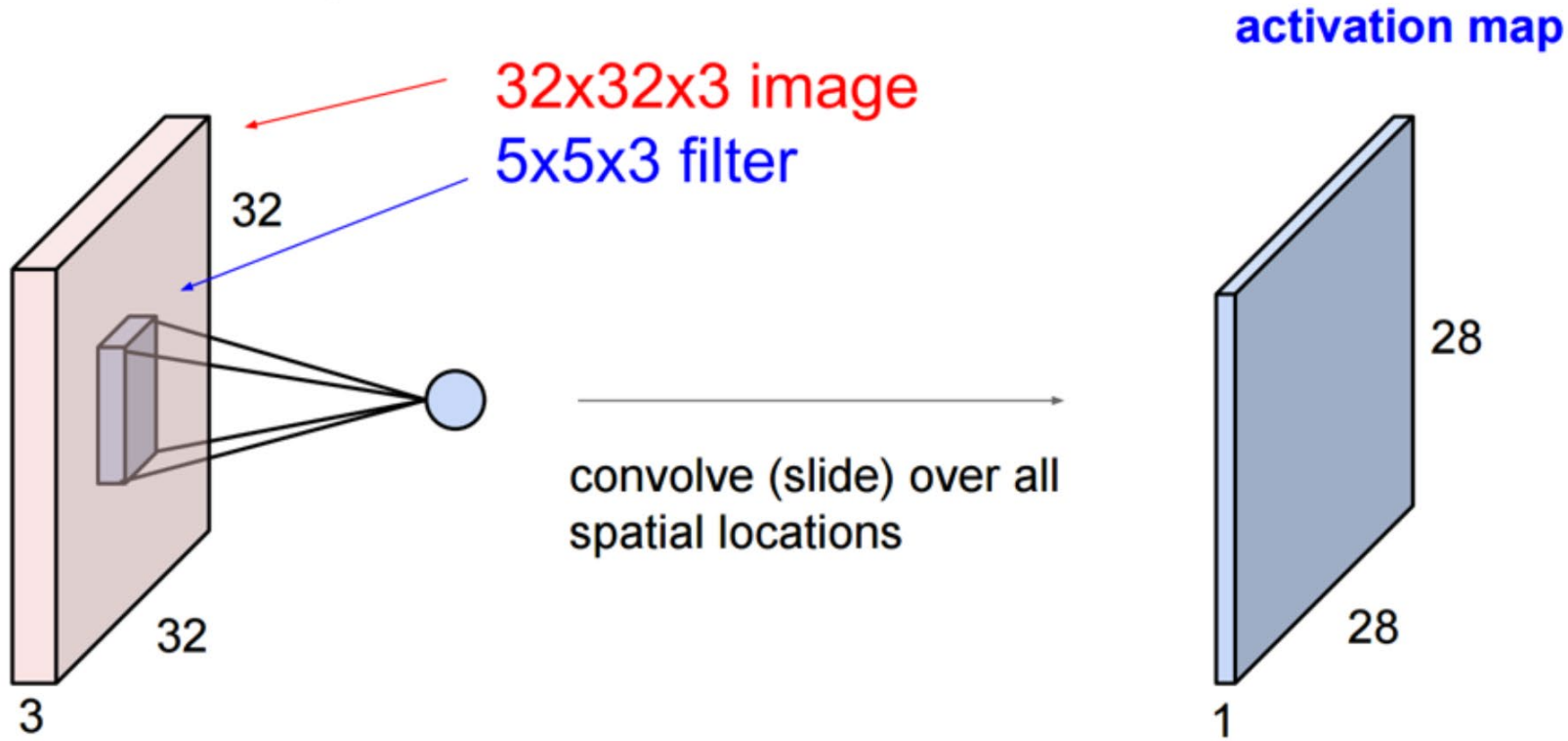
Convolution Layer



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)

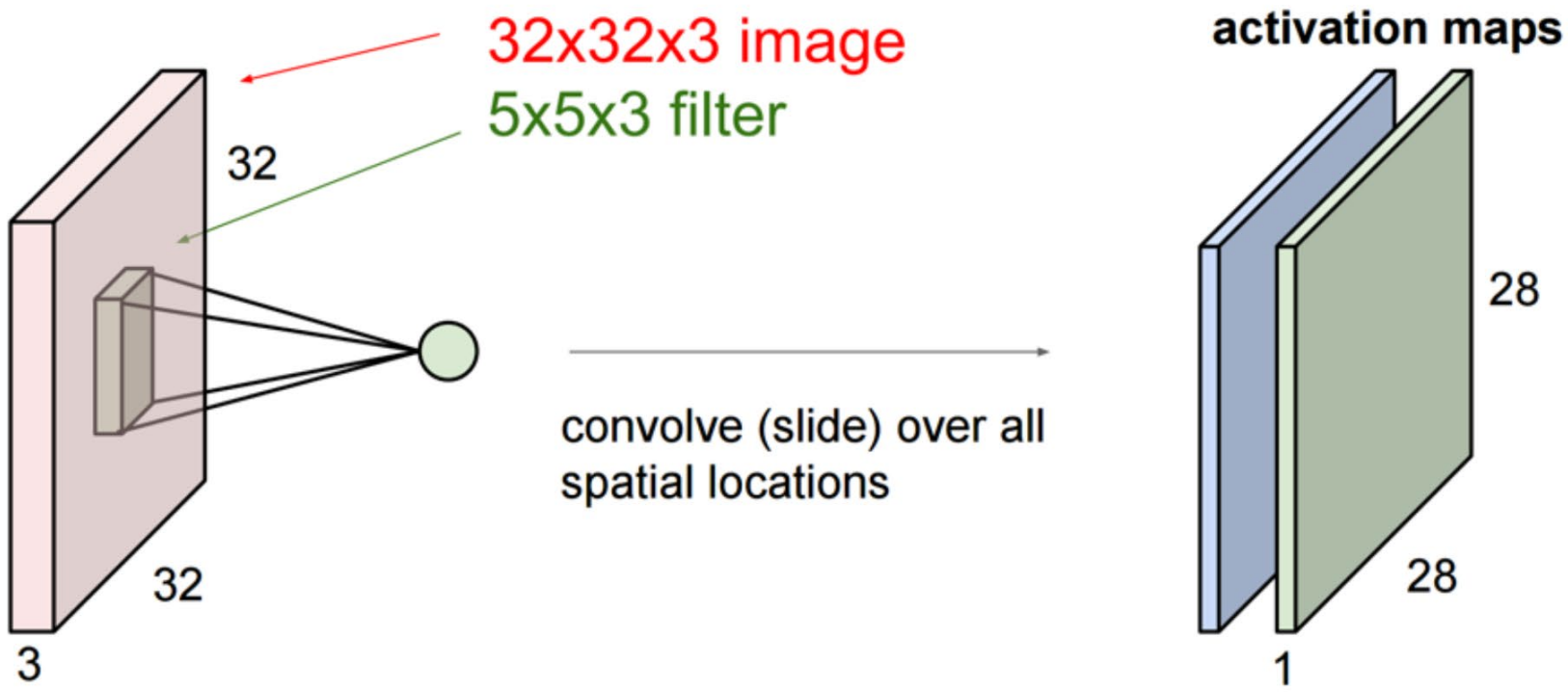
$$w^T x + b$$

Convolution Layer



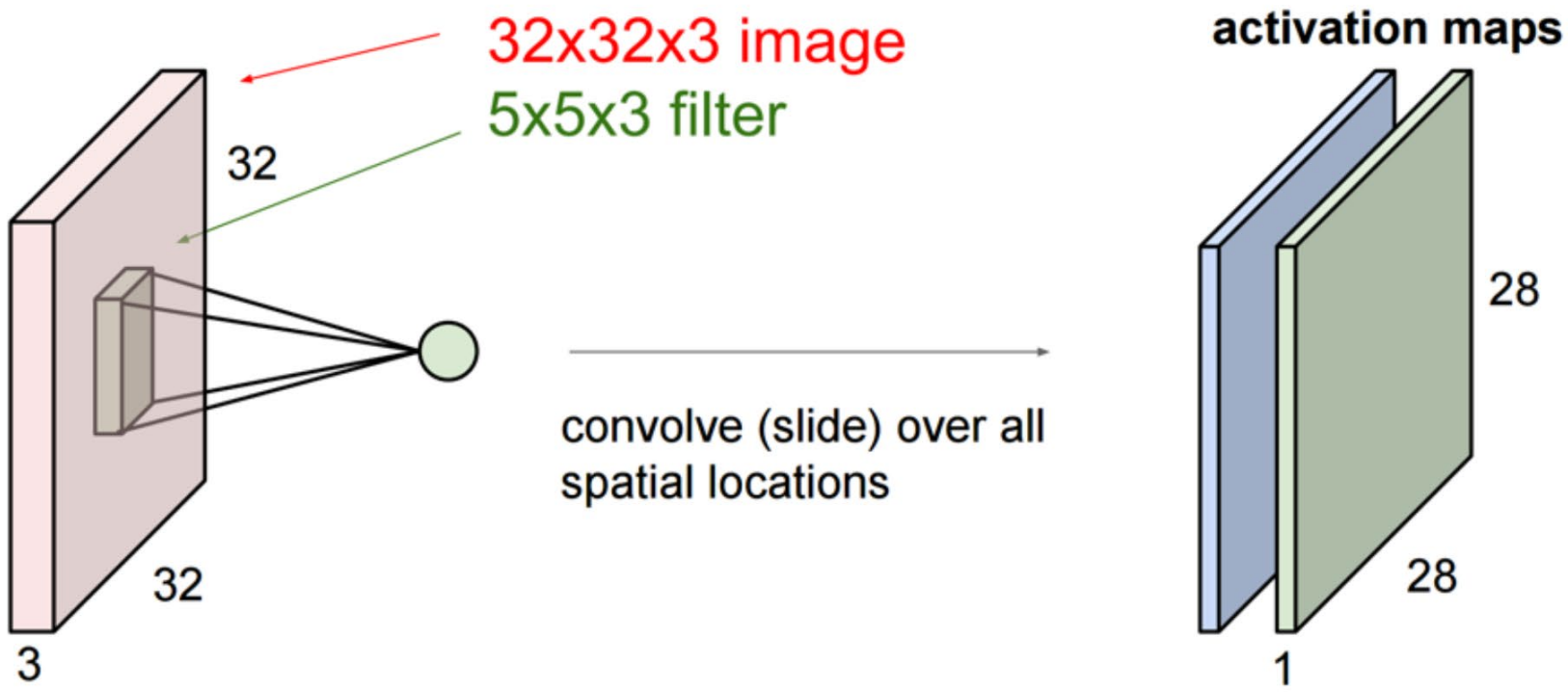
Convolution Layer

Consider a second filter ...

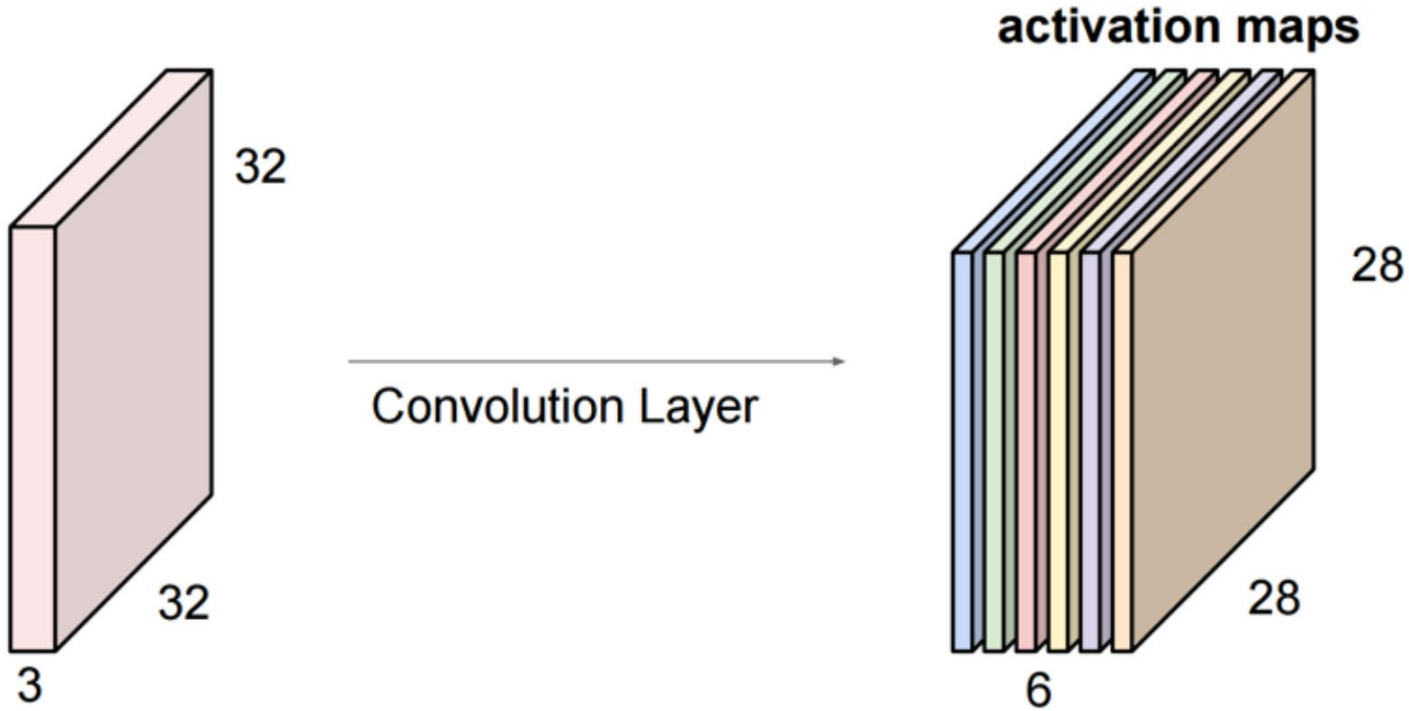


Convolution Layer

What will the output size be if we have 6 filters?



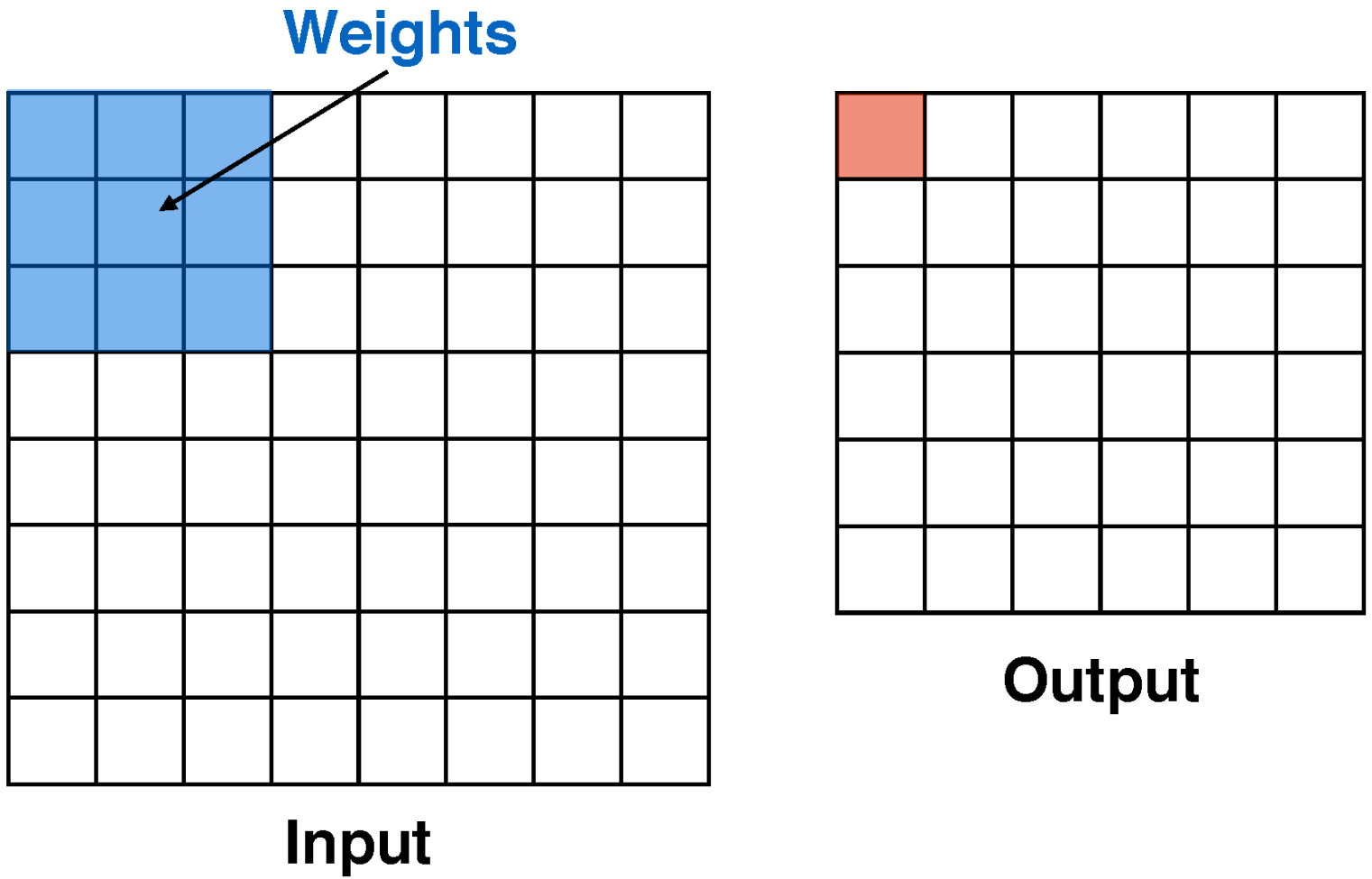
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

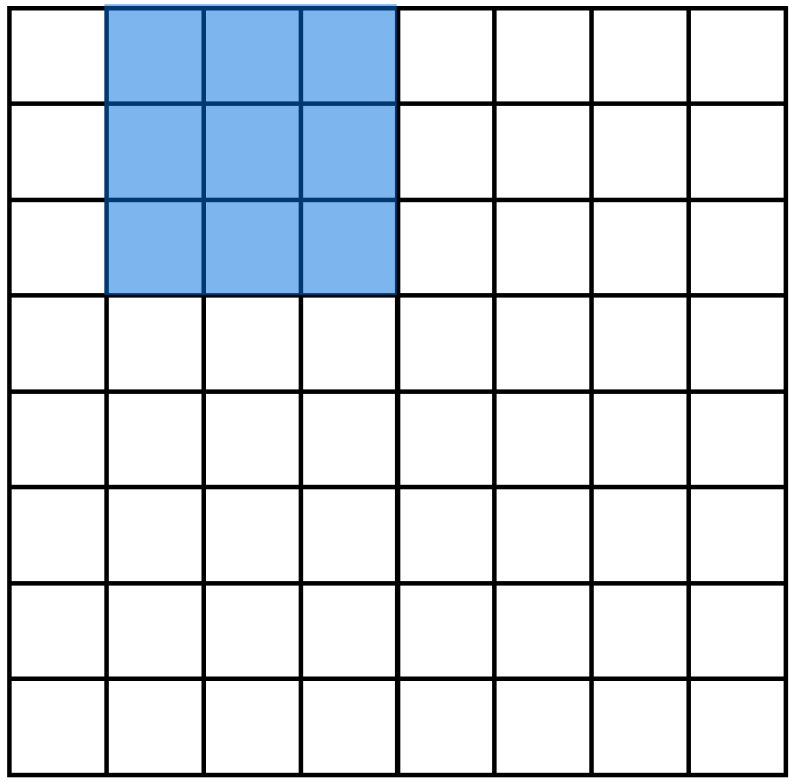
Convolution: Stride

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

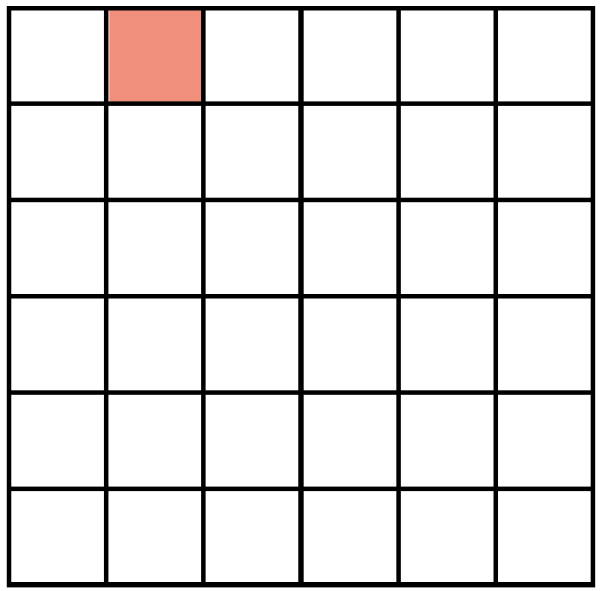


Convolution: Stride

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



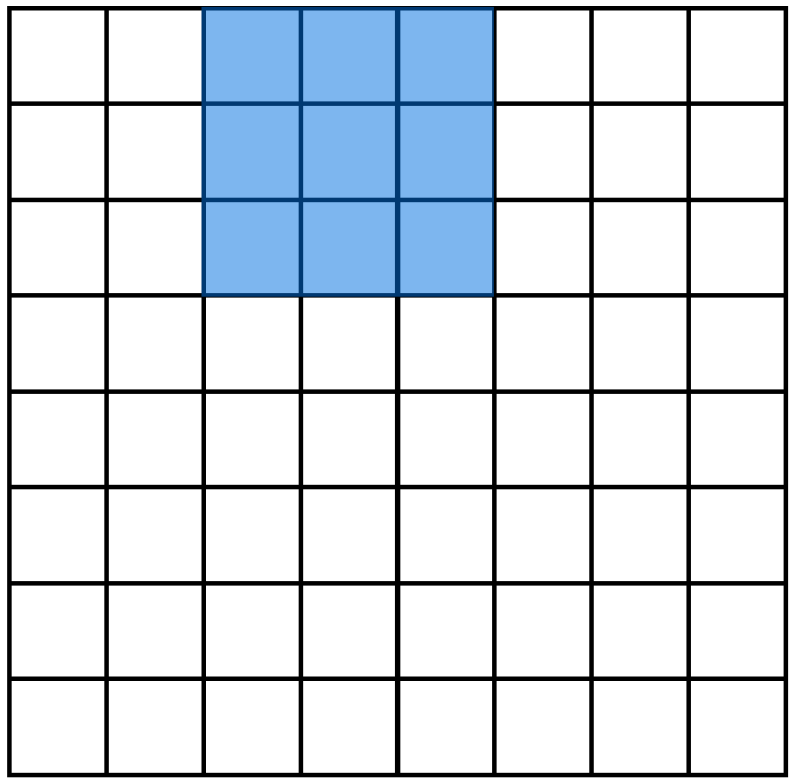
Input



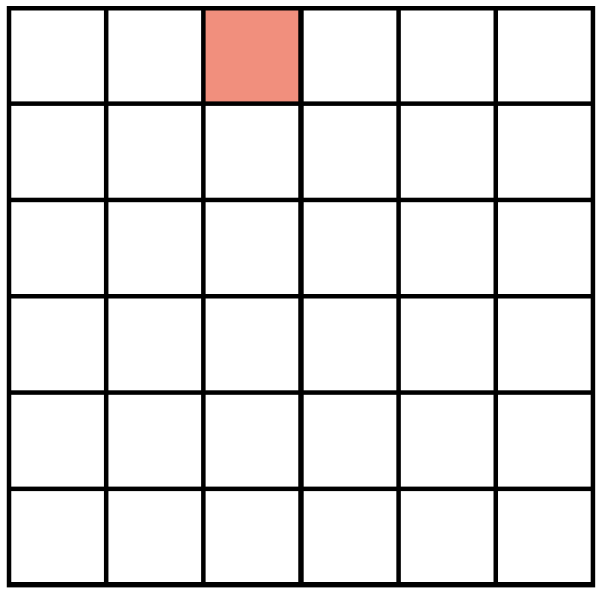
Output

Convolution: Stride

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



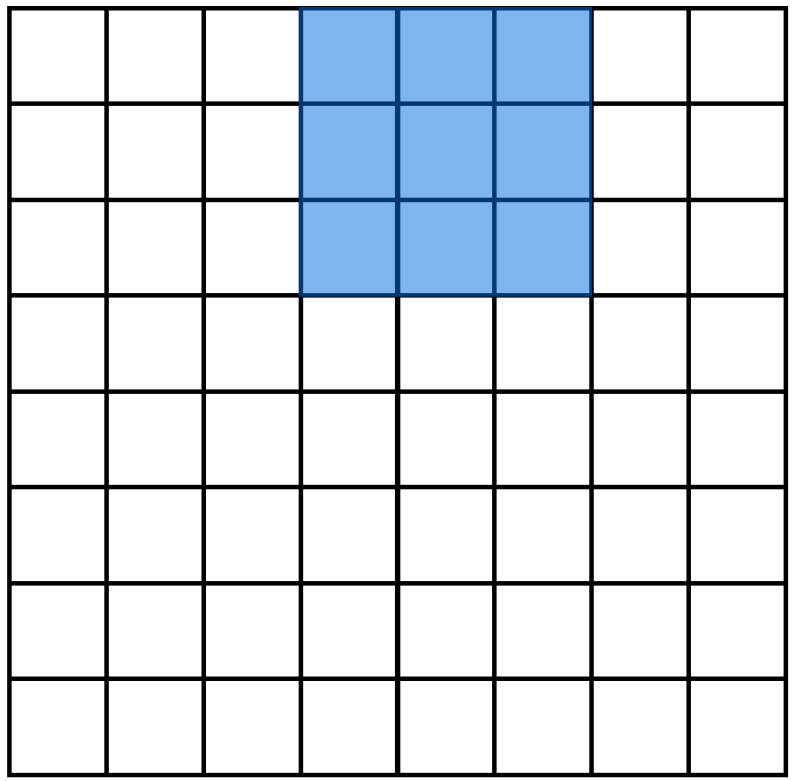
Input



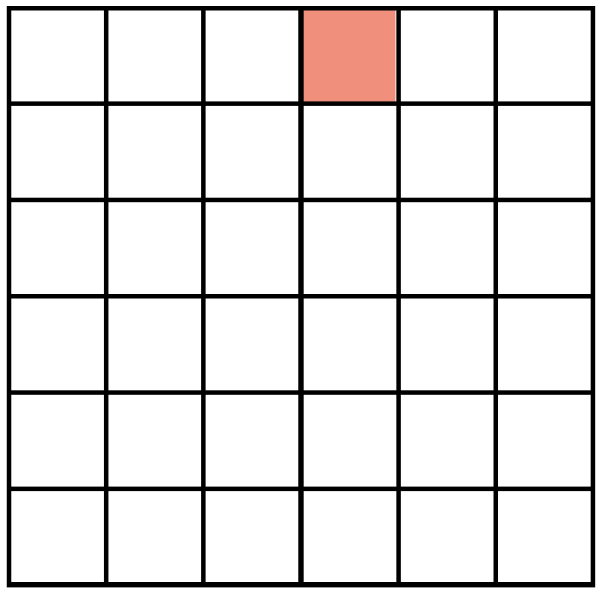
Output

Convolution: Stride

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



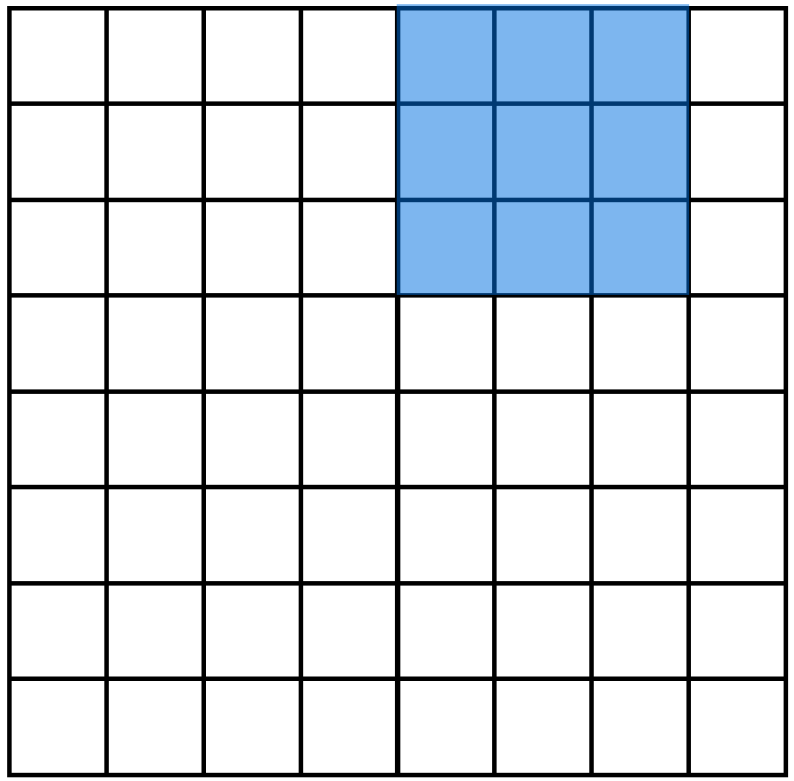
Input



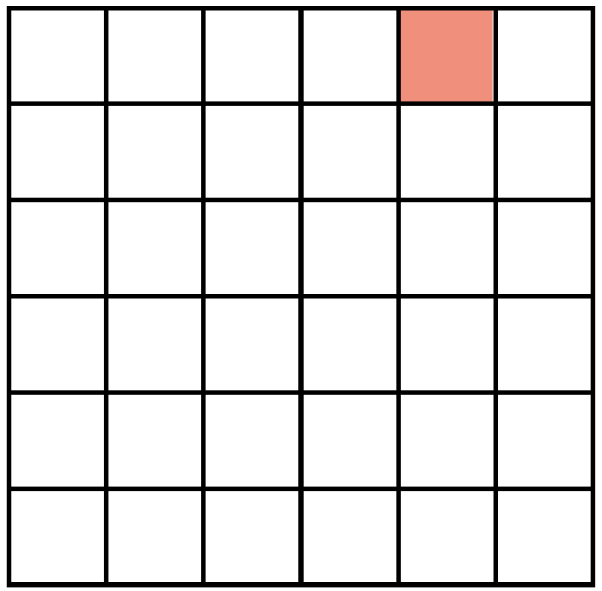
Output

Convolution: Stride

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



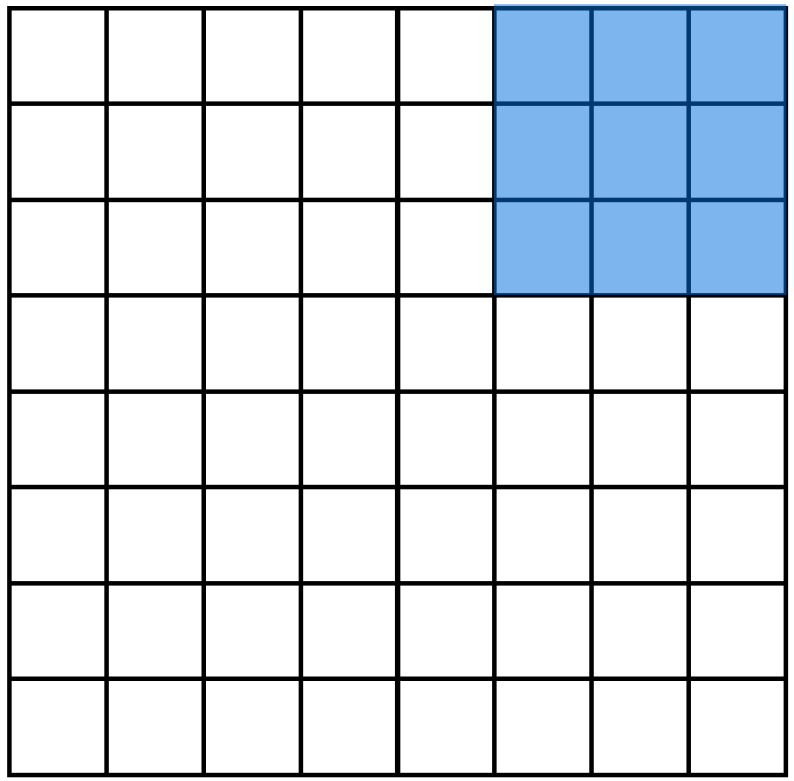
Input



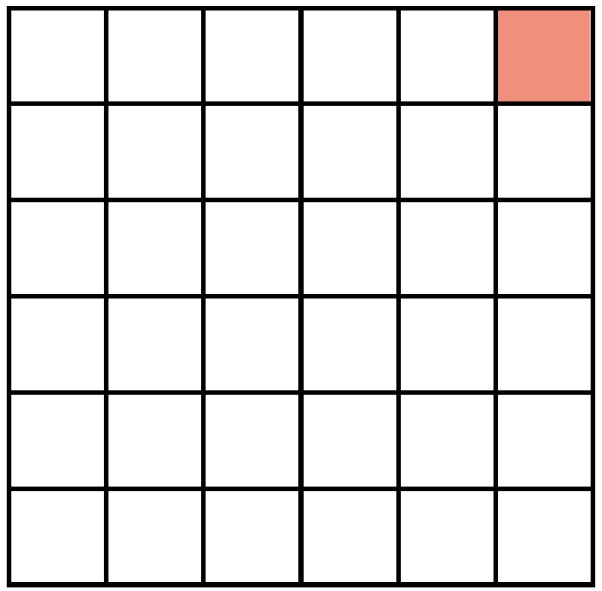
Output

Convolution: Stride

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



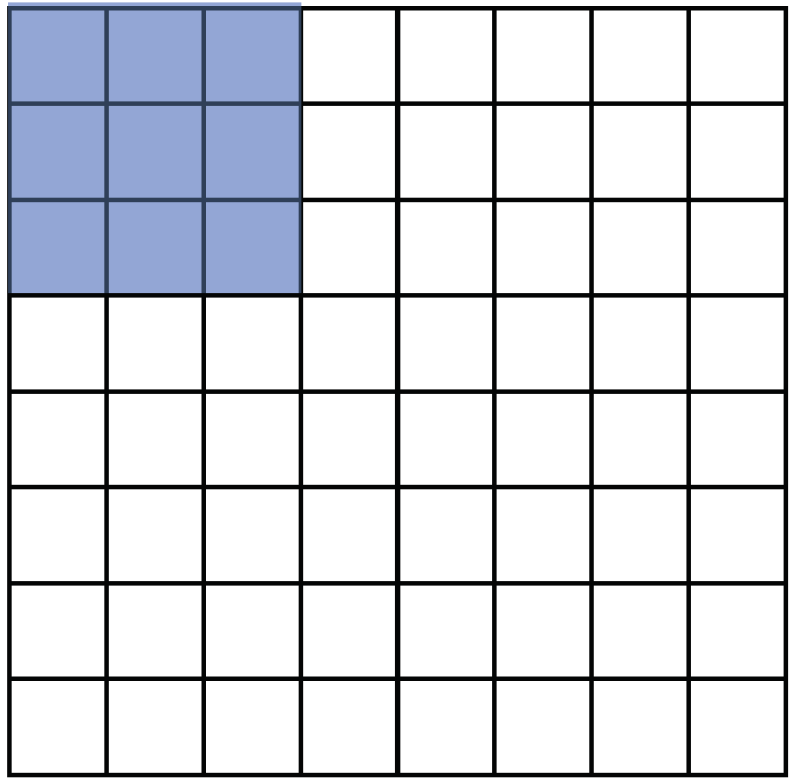
Input



Output

Convolution: Stride

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



Input

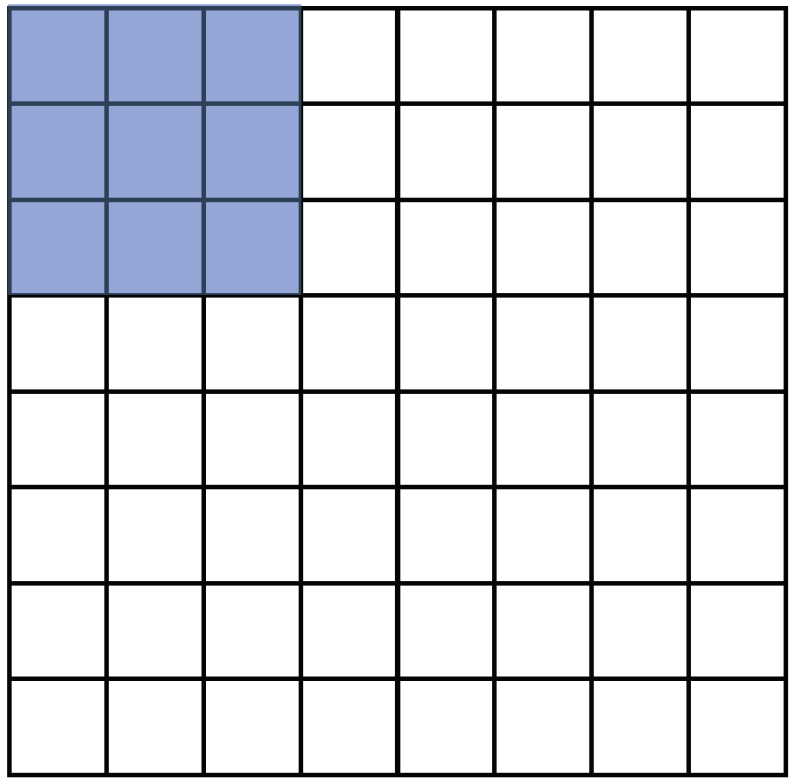
Recall that at each position, we are doing a **3D** sum:

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

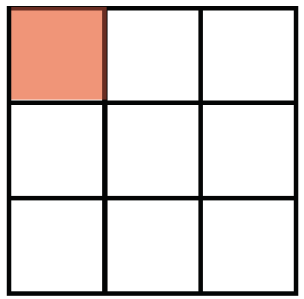
(channel, row, column)

Convolution: Stride

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



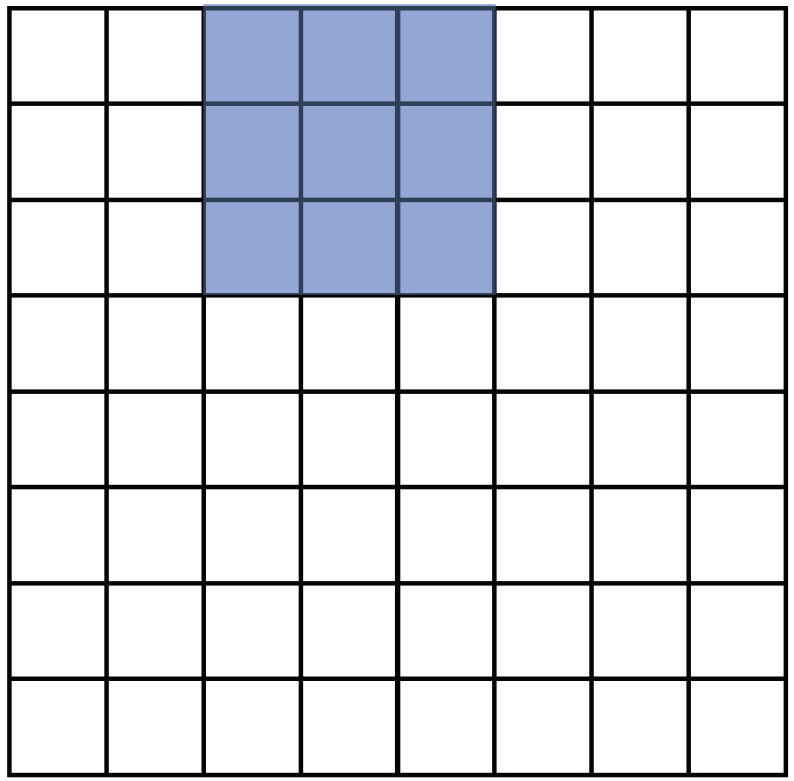
Input



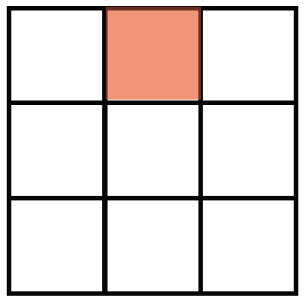
Output

Convolution: Stride

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



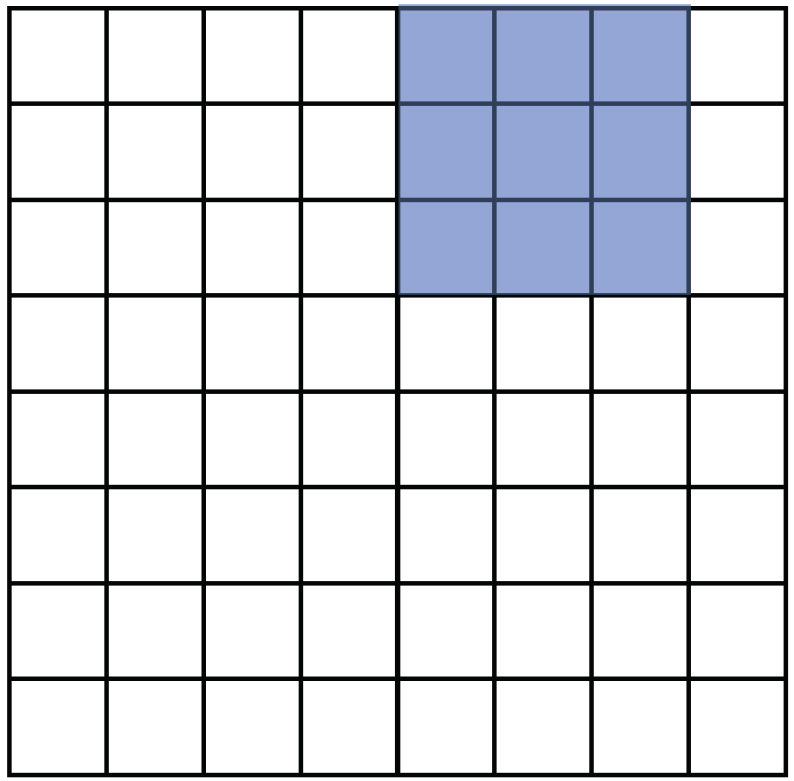
Input



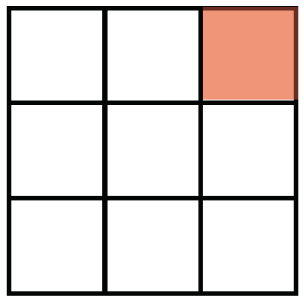
Output

Convolution: Stride

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



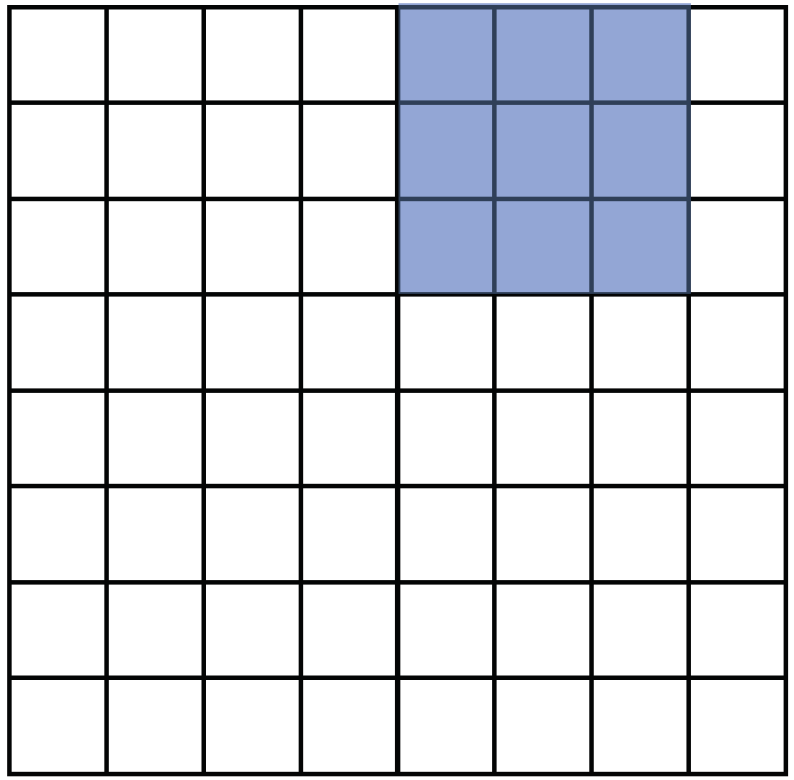
Input



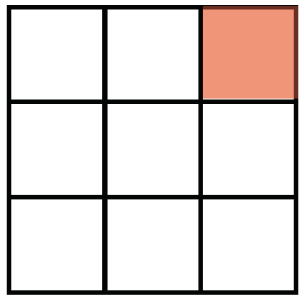
Output

Convolution: Stride

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



Input



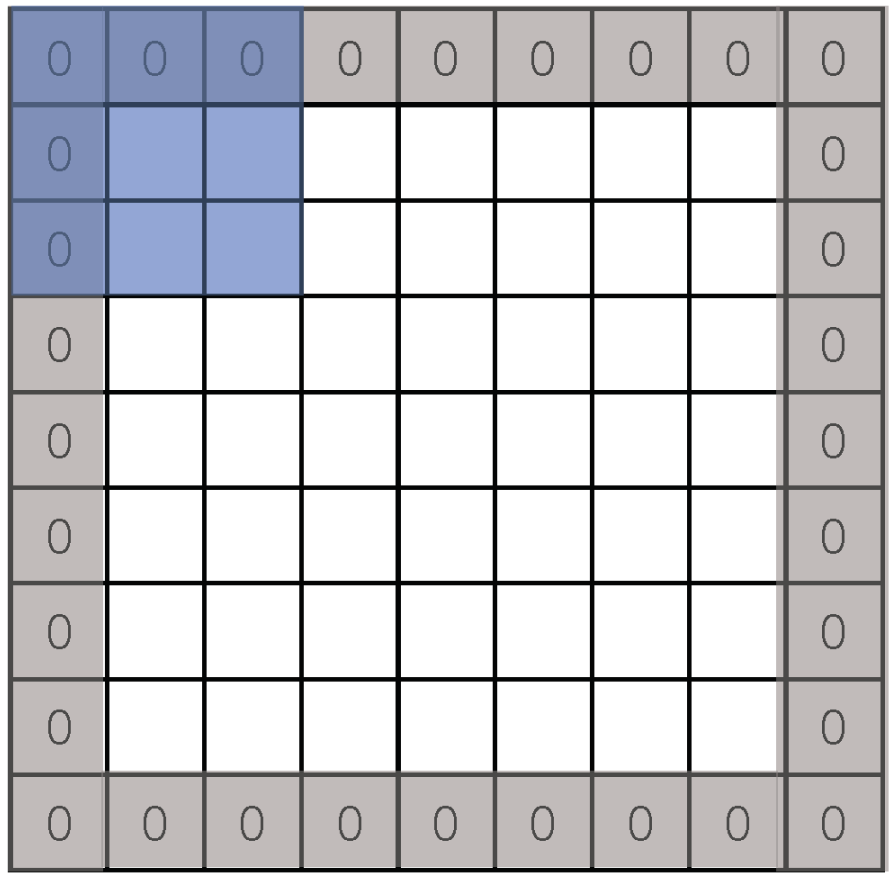
Output

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

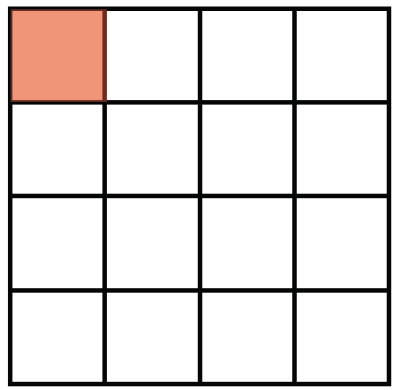
- The output is also half the size of the input

Convolution: Padding

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



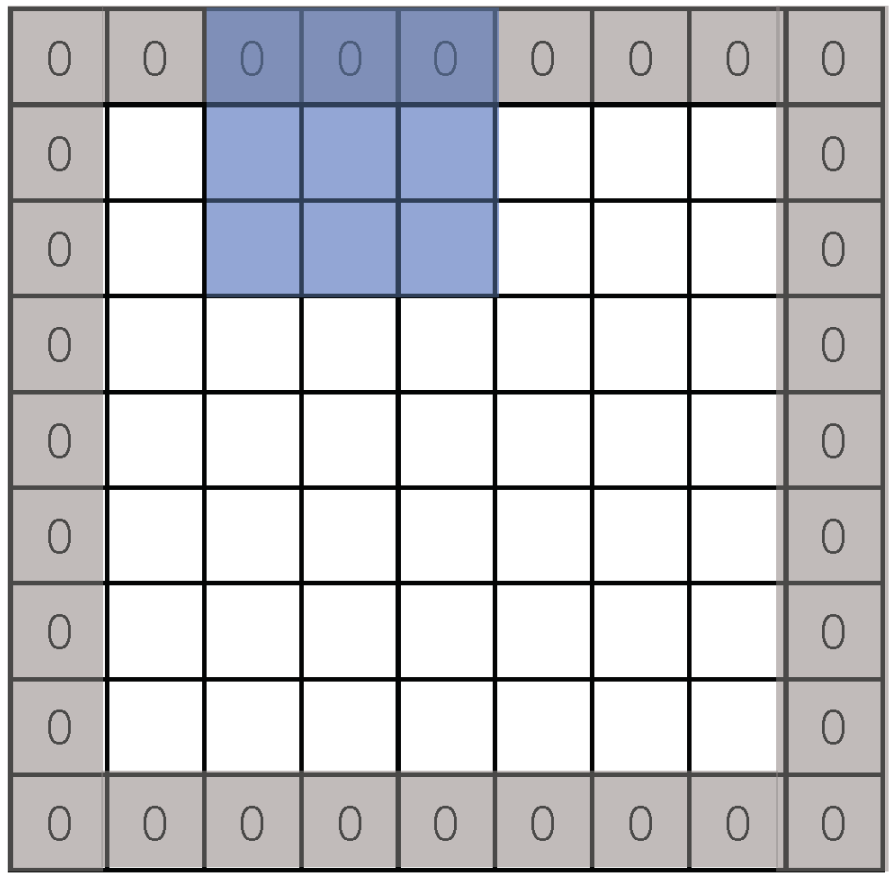
Input



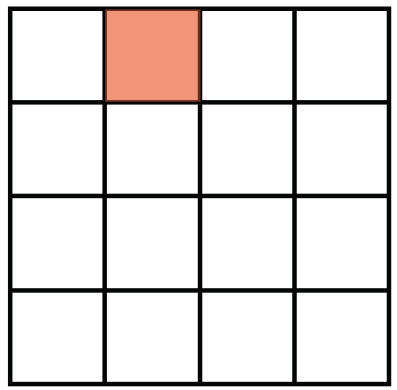
Output

Convolution: Padding

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



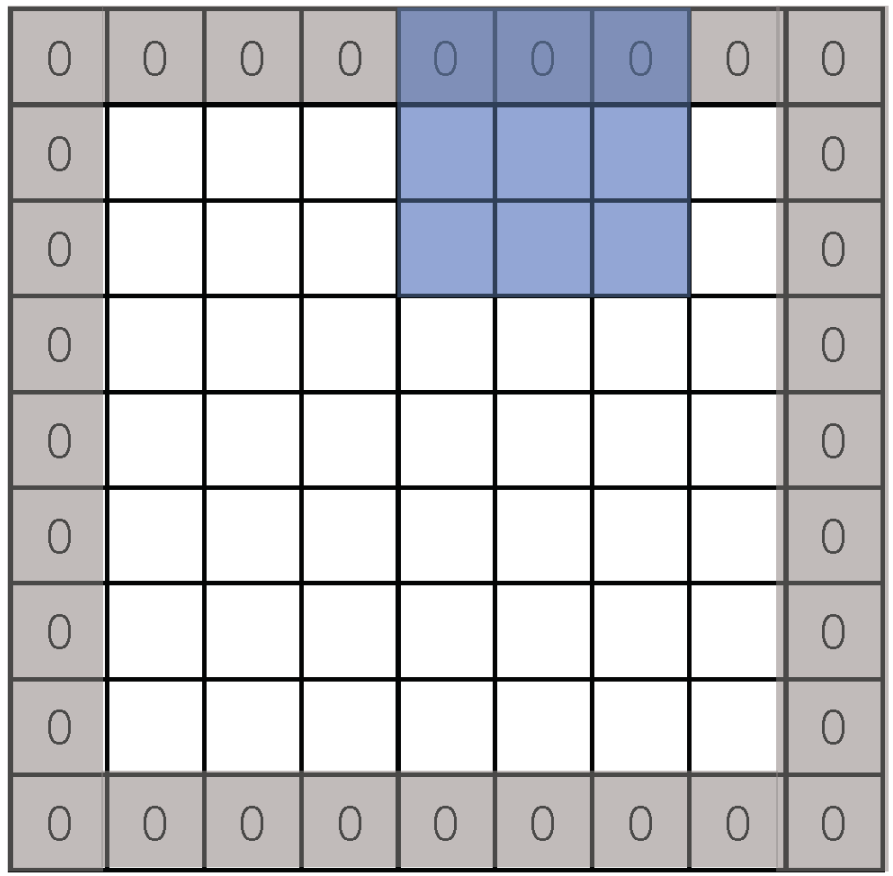
Input



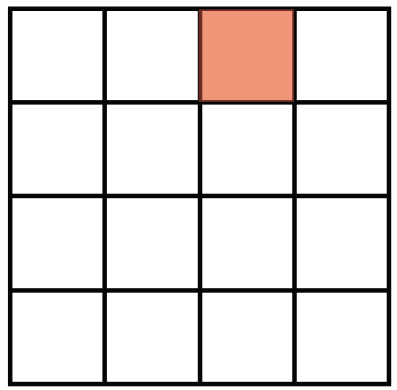
Output

Convolution: Padding

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



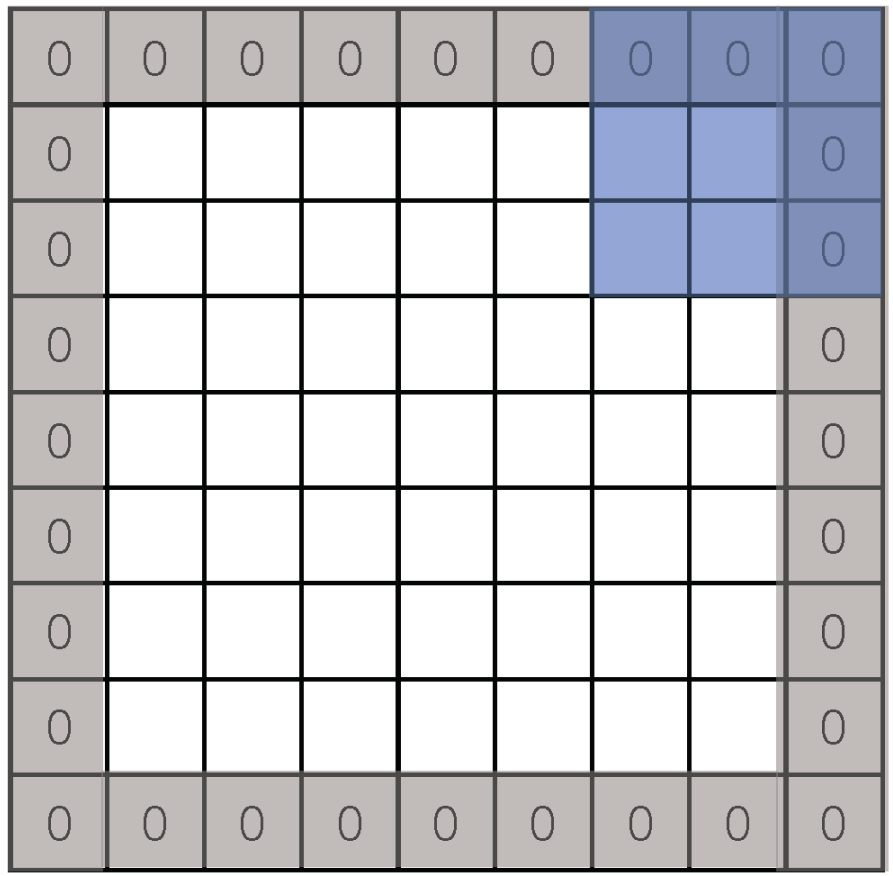
Input



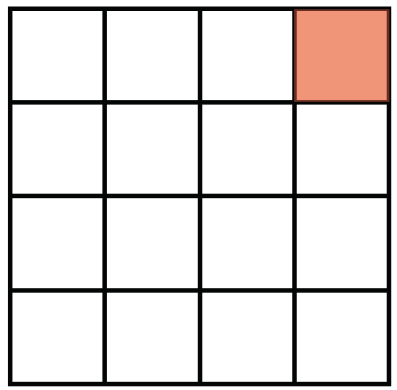
Output

Convolution: Padding

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



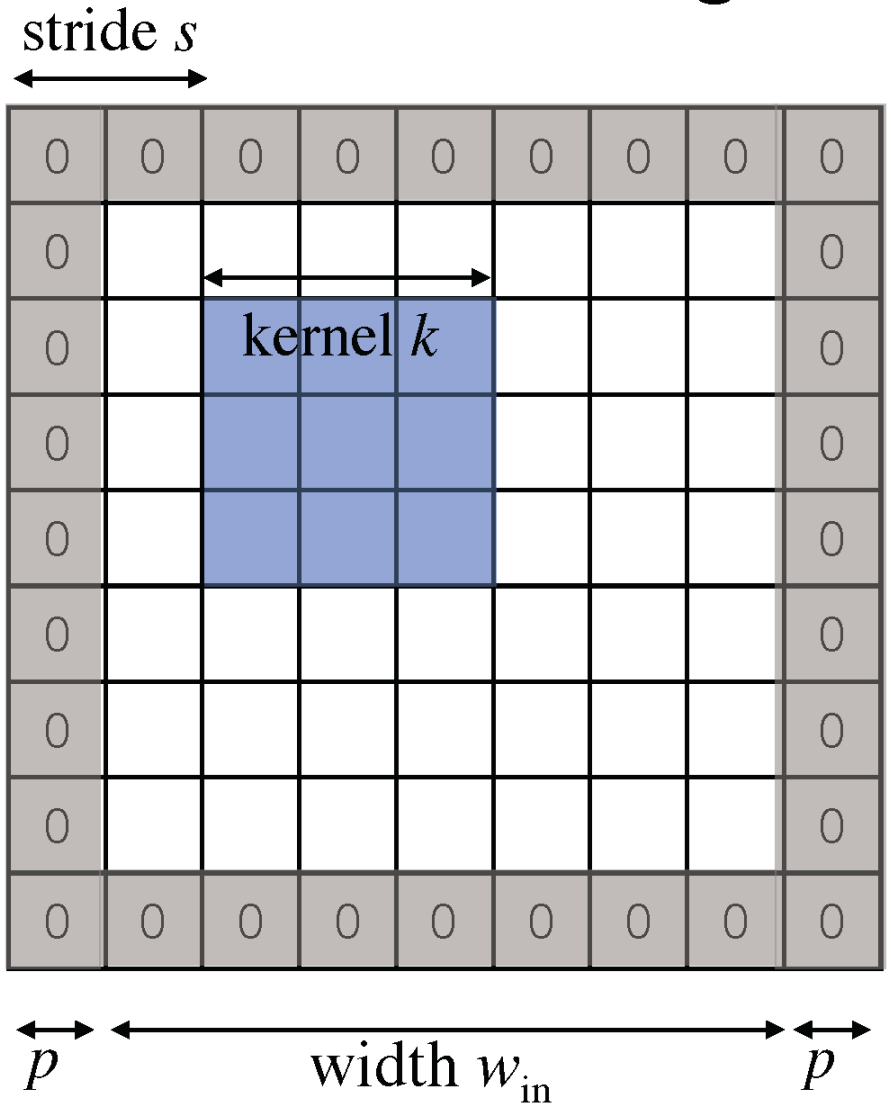
Input



Output

Convolution:

How big is the output?

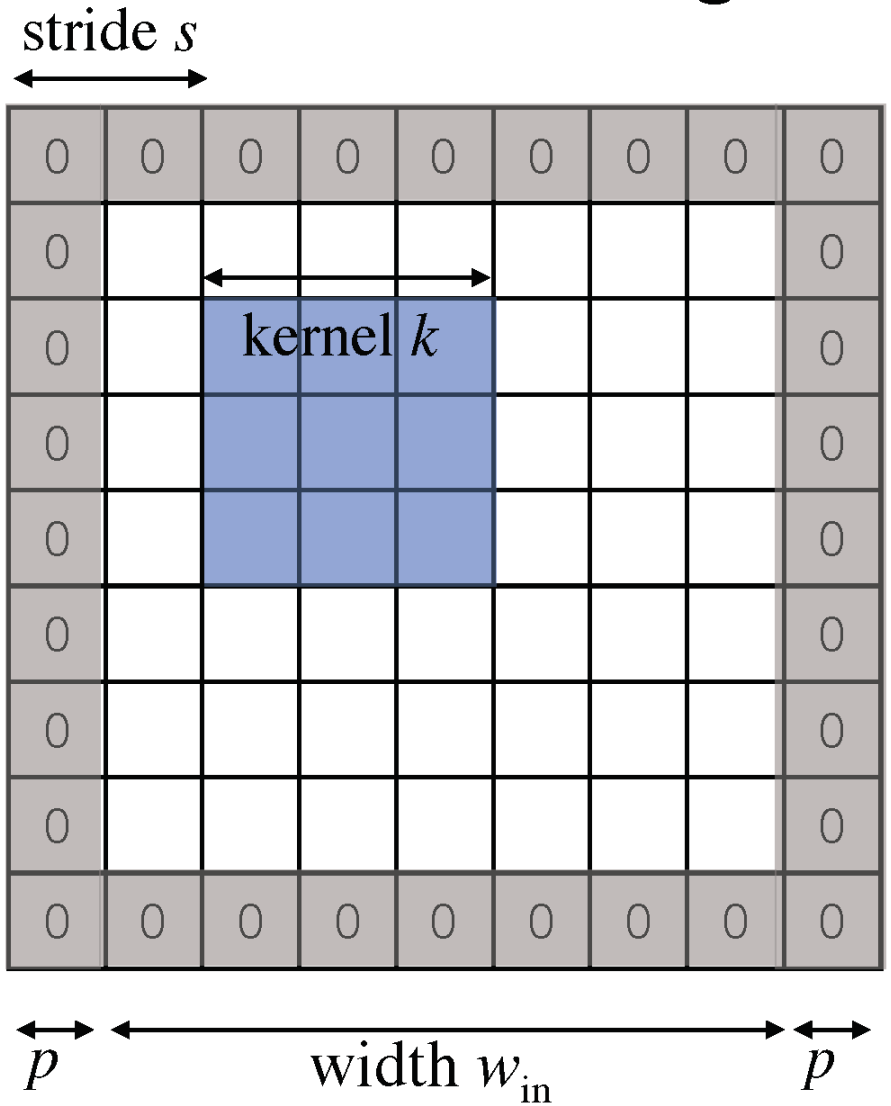


In general, the output has size:

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

Convolution:

How big is the output?



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

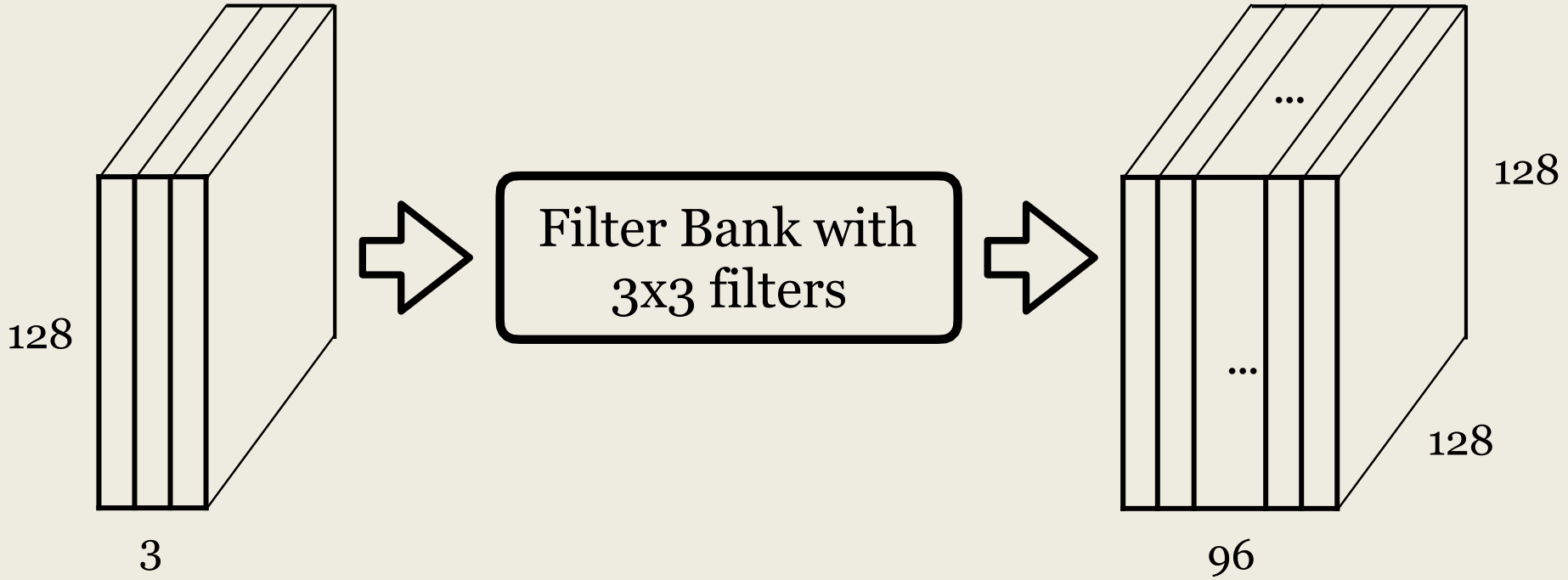
$$w_{\text{out}} = \left\lfloor \frac{w_{\text{in}} + 2p - k}{s} \right\rfloor + 1$$

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

$$= w_{\text{in}}$$

VGGNet [Simonyan 2014] uses filters of this shape

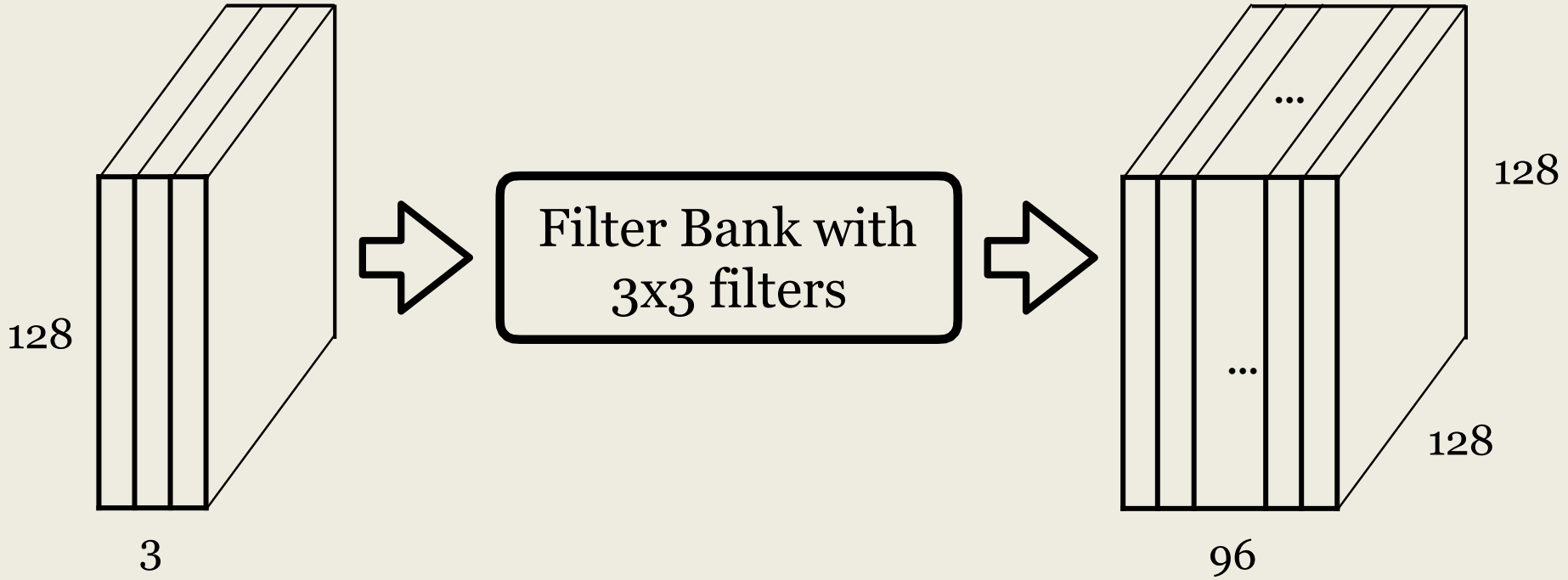
Knowledge Check ...



How many parameters does **each *filter*** have?

(a) 9 (b) 27 (c) 96 (d) 864

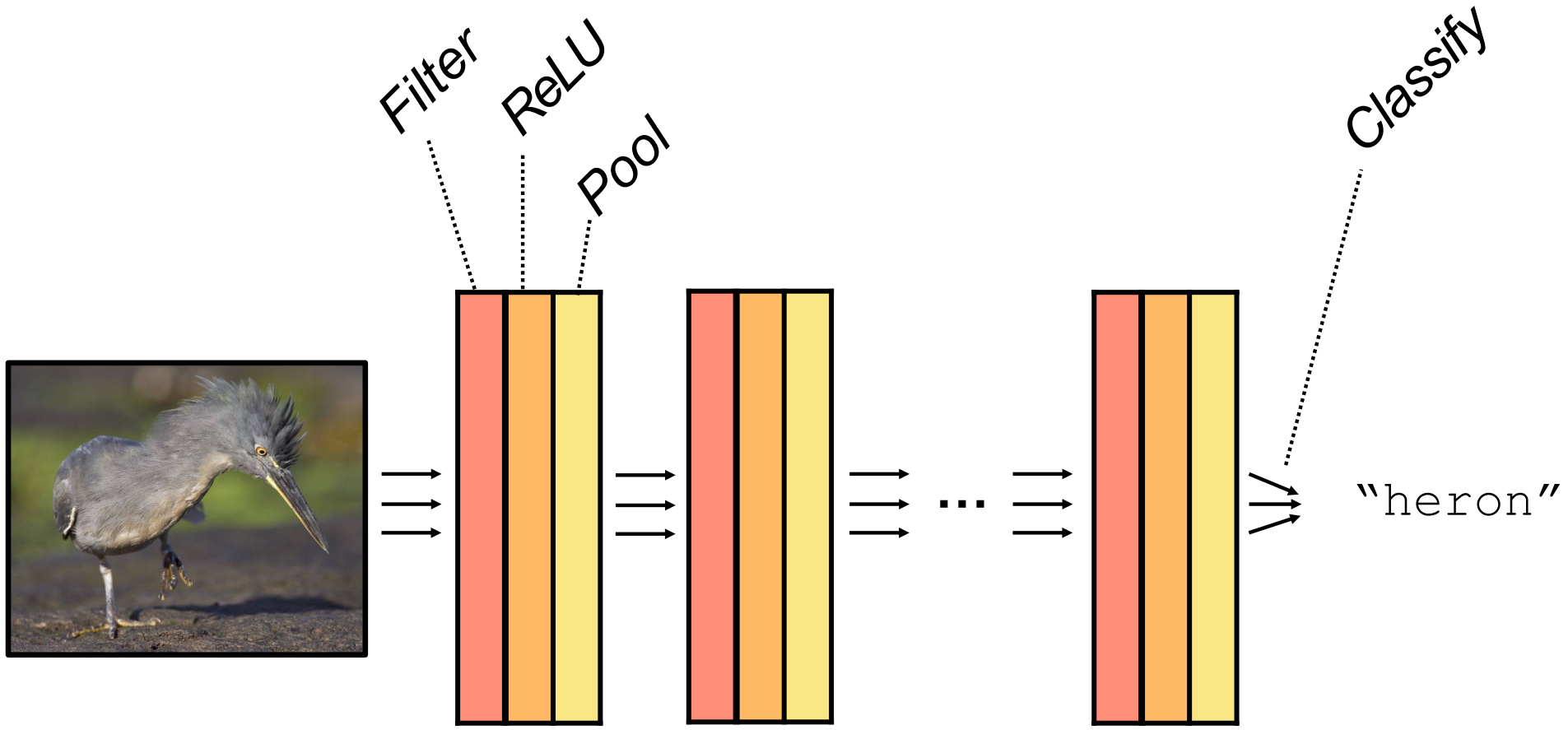
Knowledge Check ...



How many filters are in the bank?

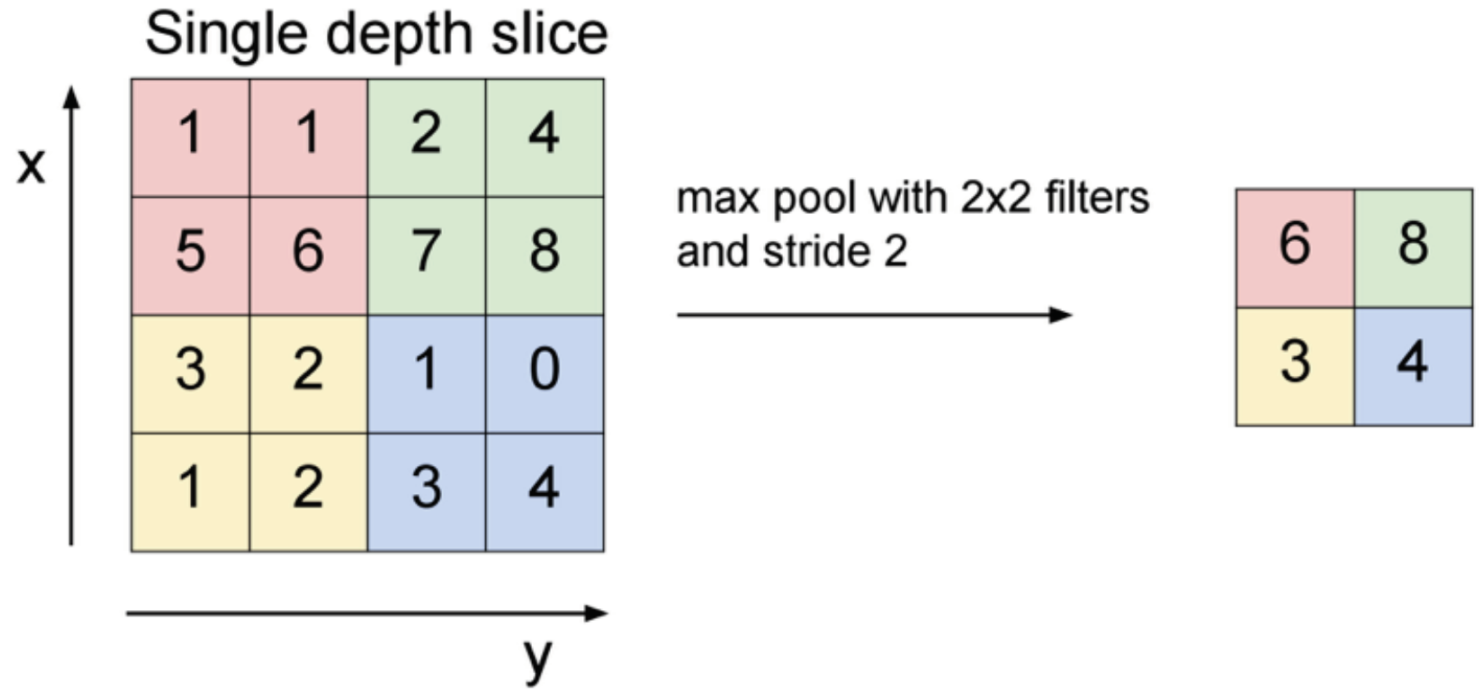
(a) 3 (b) 27 (c) 96 (d) can't say

Computation in a neural net



$$f(\mathbf{x}) = f_L(\ldots f_2(f_1(\mathbf{x})))$$

Max Pooling



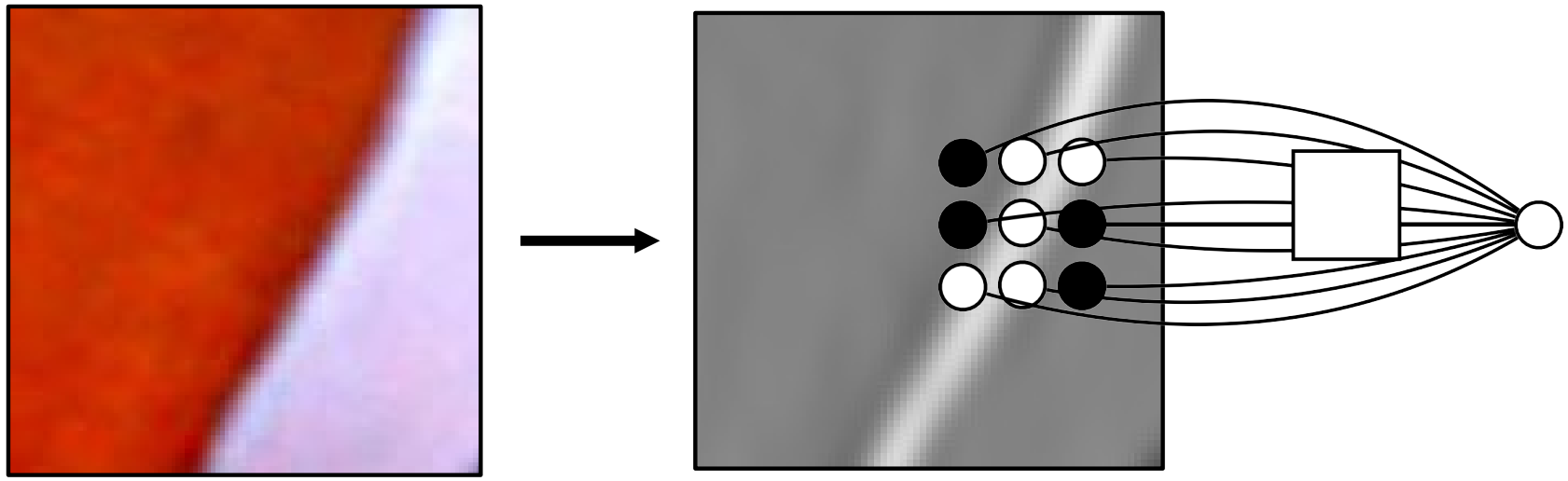
What's the backprop rule for max pooling?

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

Figure: Andrej Karpathy

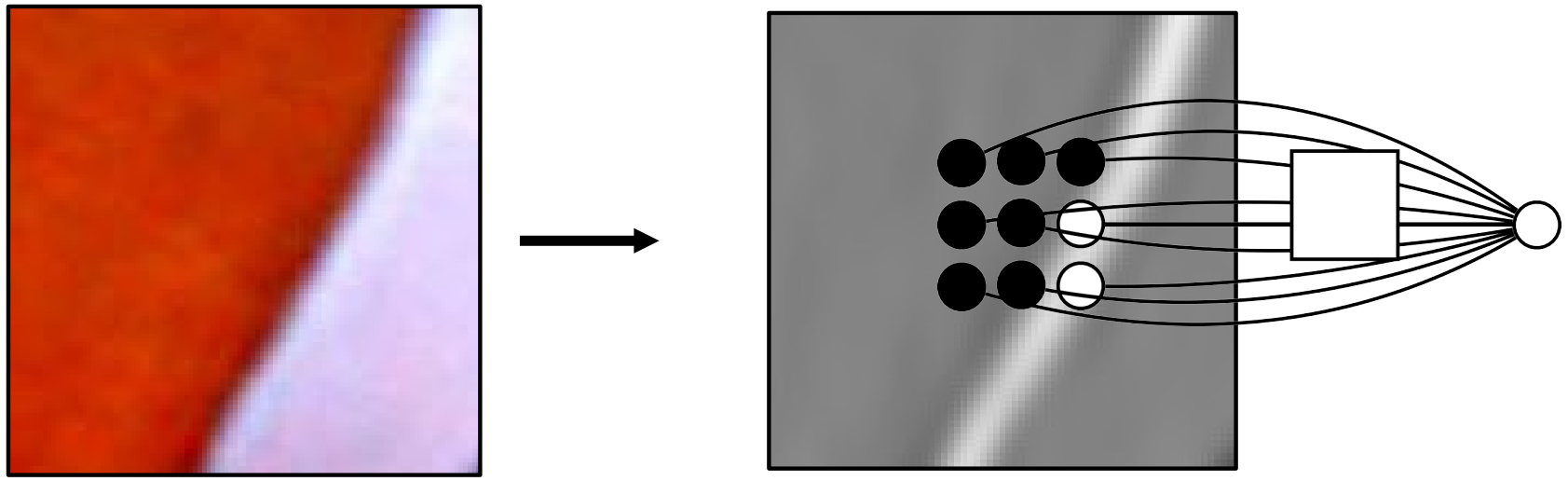
Pooling — Why?

Pooling across spatial locations achieves stability w.r.t. small translations:



Pooling — Why?

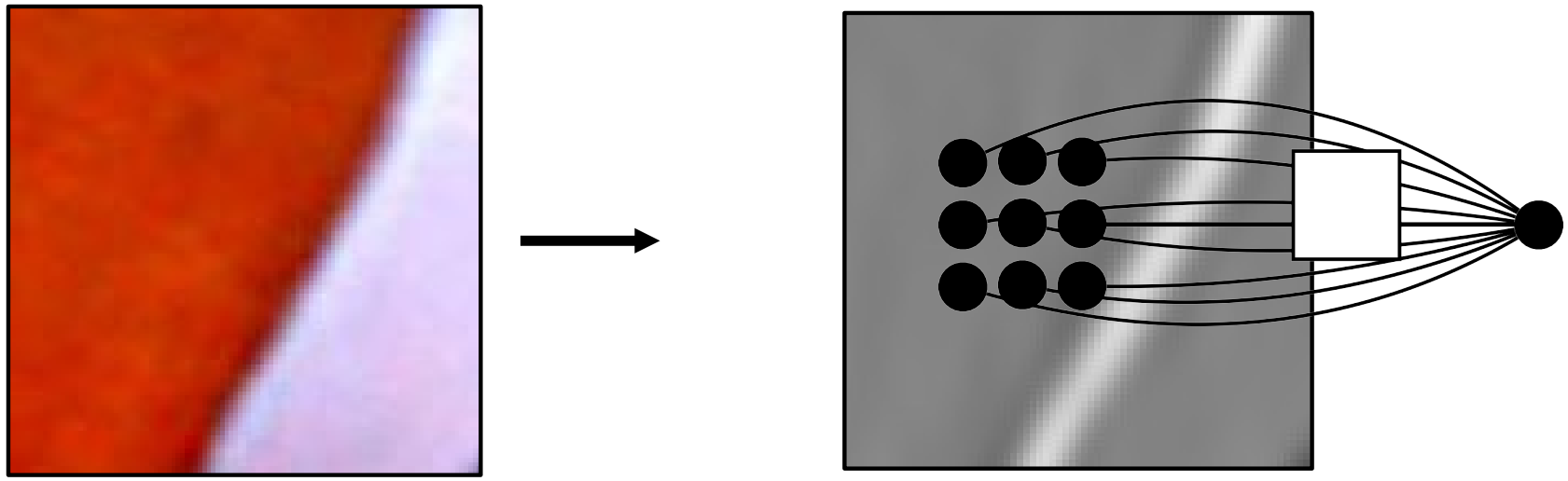
Pooling across spatial locations achieves stability w.r.t. small translations:



large response regardless of exact position of edge

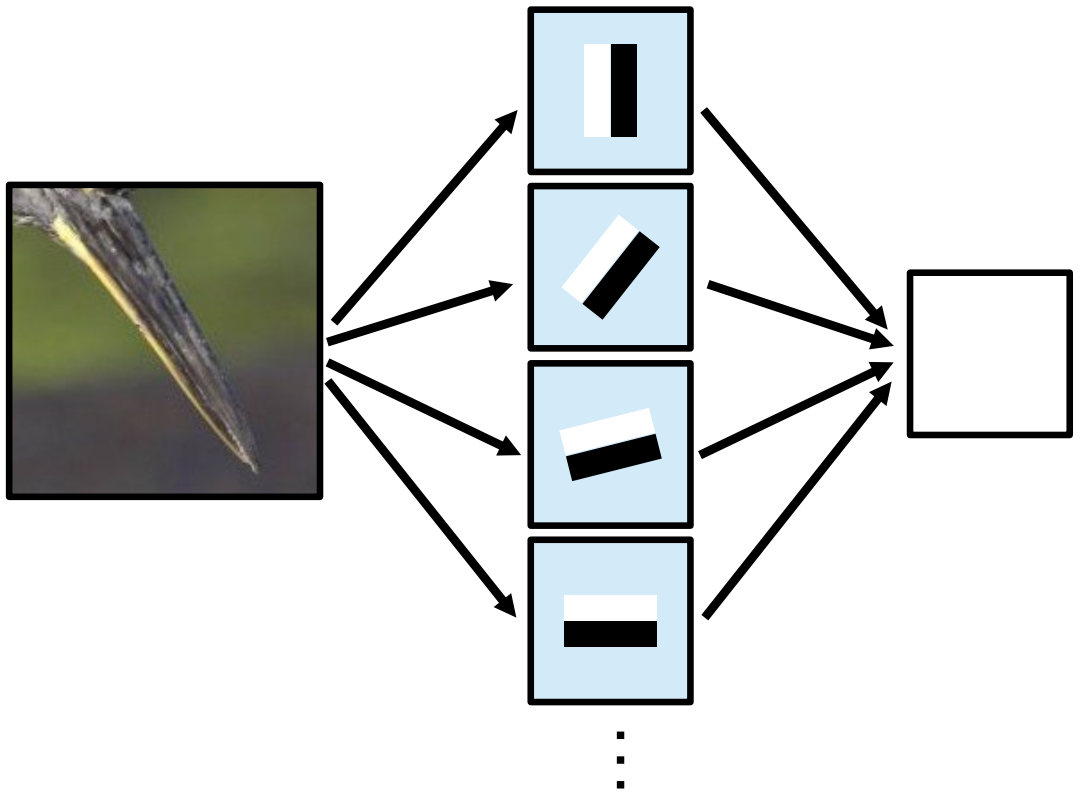
Pooling — Why?

Pooling across spatial locations achieves stability w.r.t. small translations:



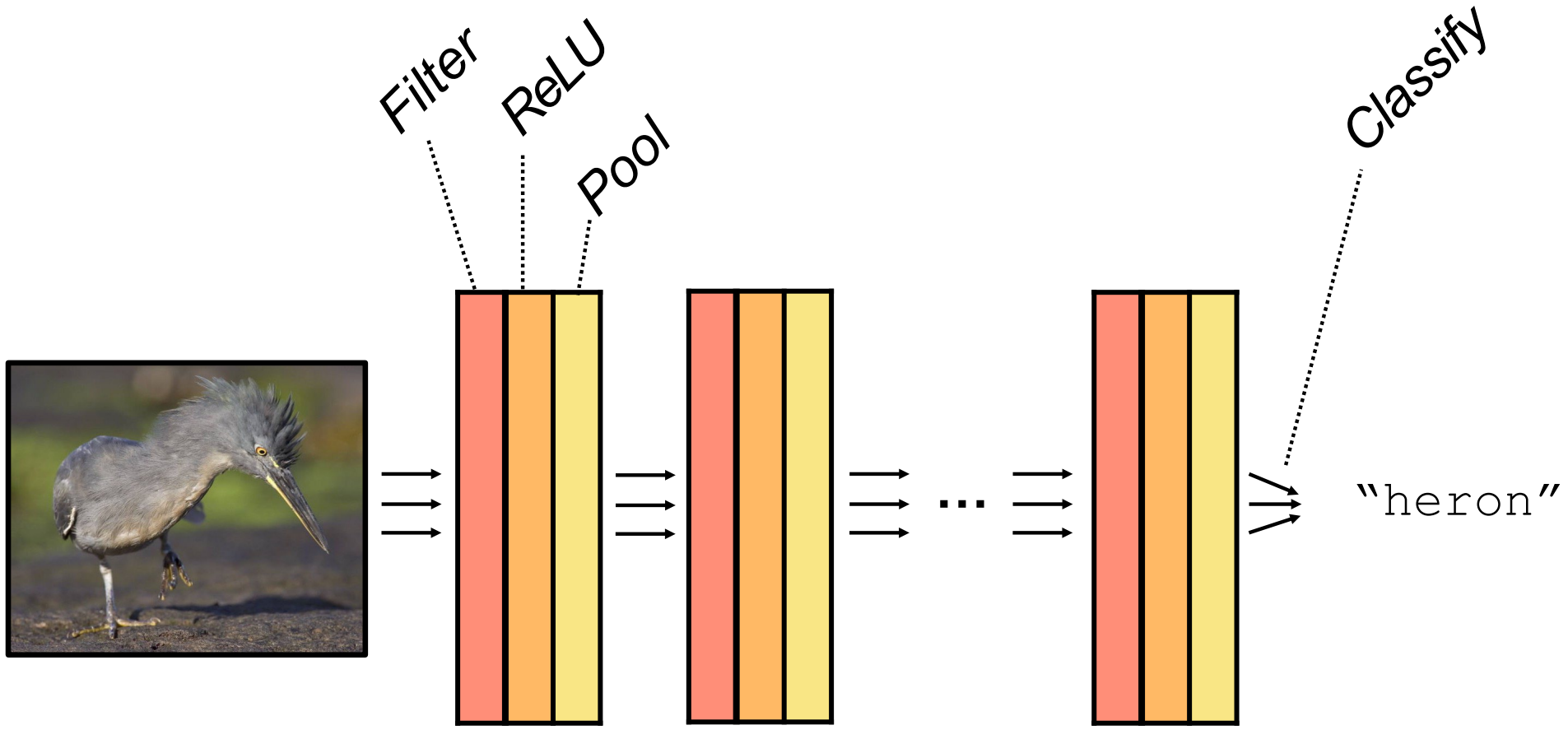
Pooling *across channels* — Why?

Pooling across feature channels (filter outputs)
can achieve other kinds of invariances:



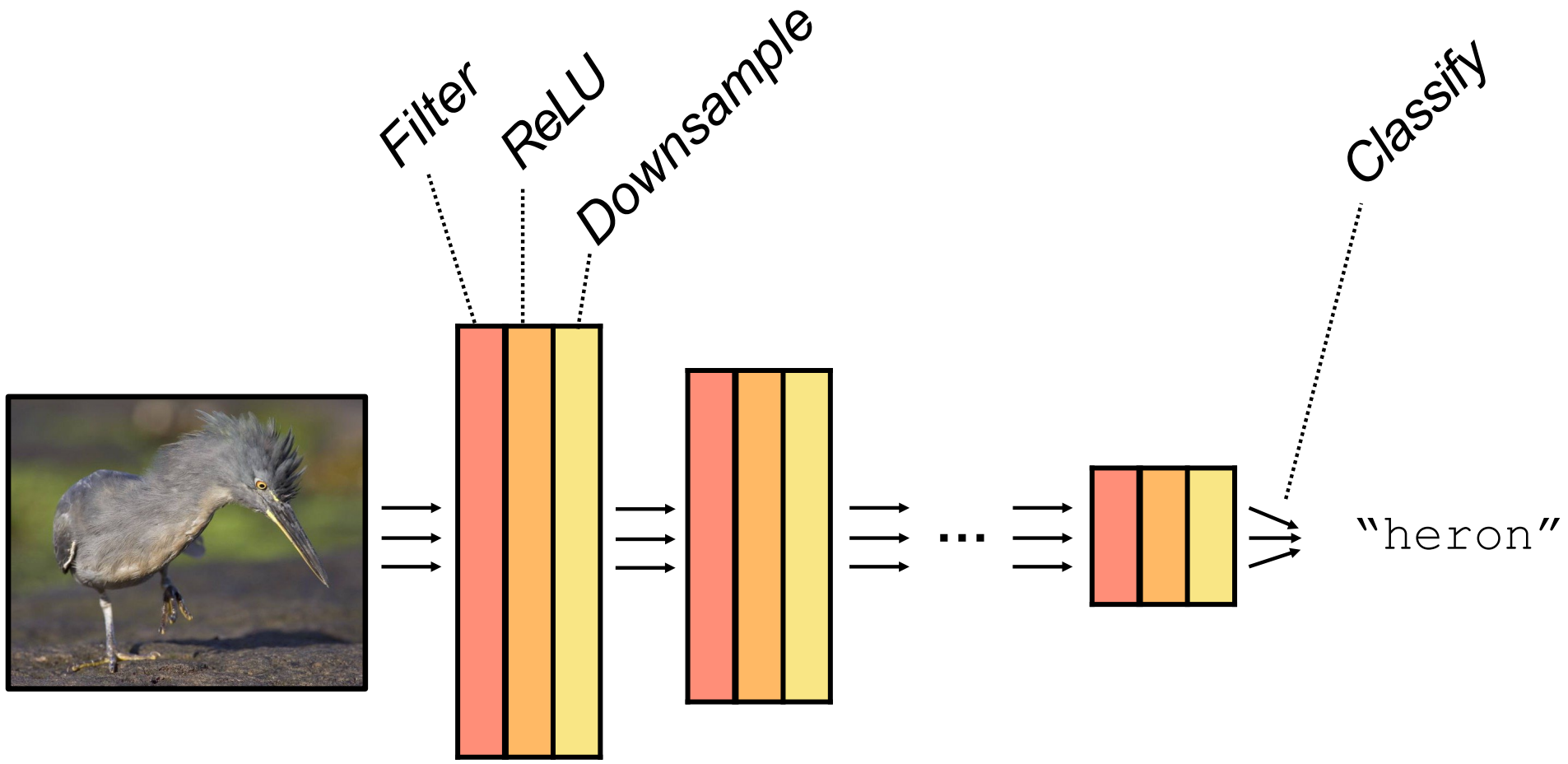
large response
for any edge,
regardless of its
orientation

Computation in a neural net



$$f(\mathbf{x}) = f_L(\ldots f_2(f_1(\mathbf{x})))$$

Computation in a neural net



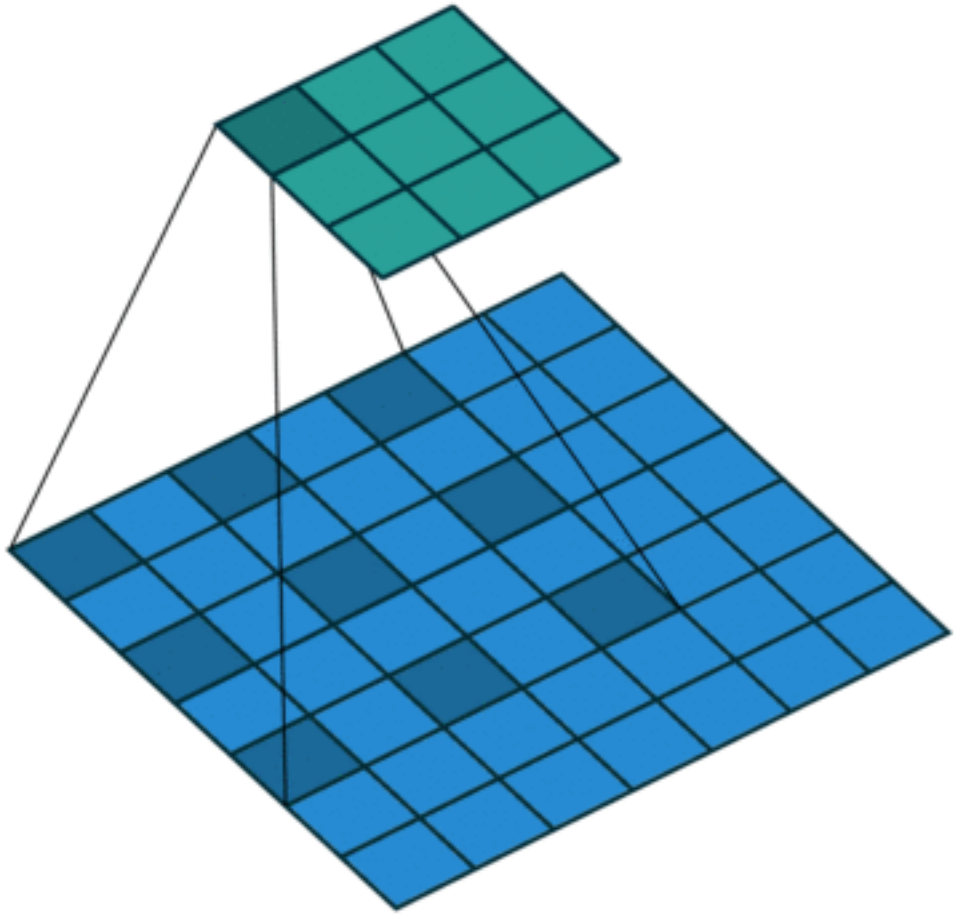
$$f(\mathbf{x}) = f_L(\ldots f_2(f_1(\mathbf{x})))$$

Pooling vs Downsampling

Dilated Convolutions

Allows increasing the receptive field of the convolutional layer

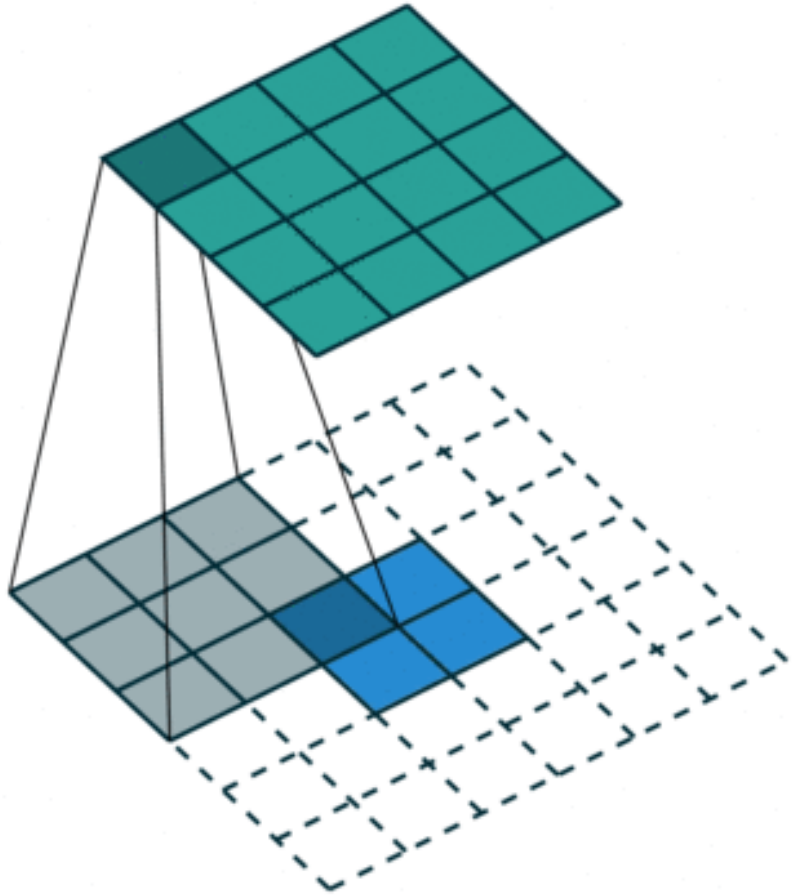
Useful for looking at larger spatial context without looking at every pixel



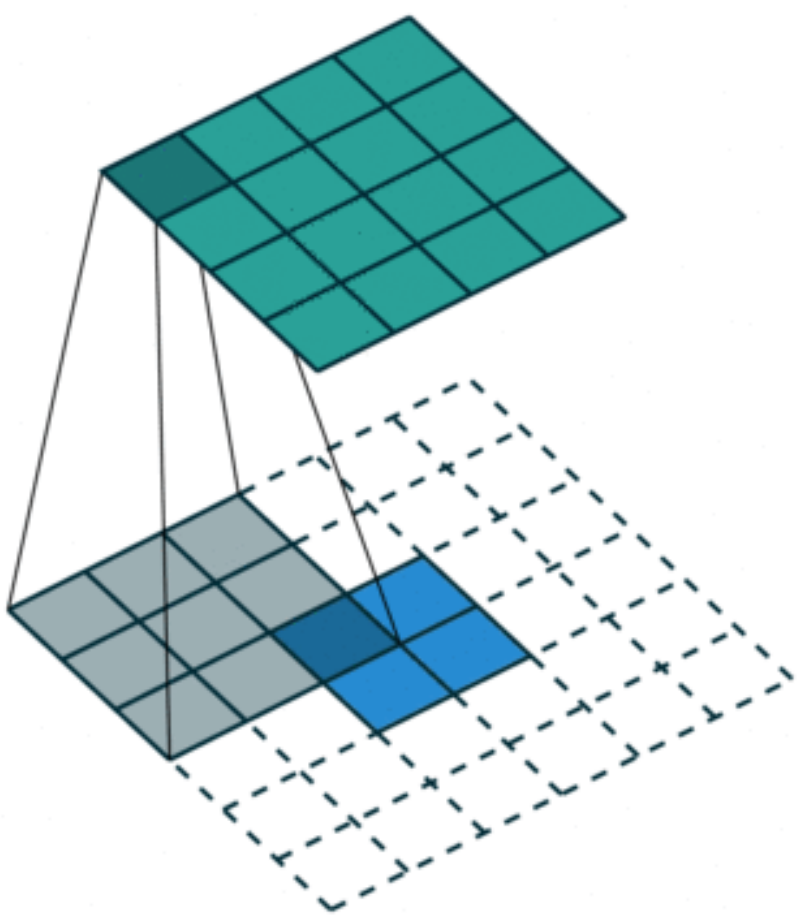
Transposed Convolution

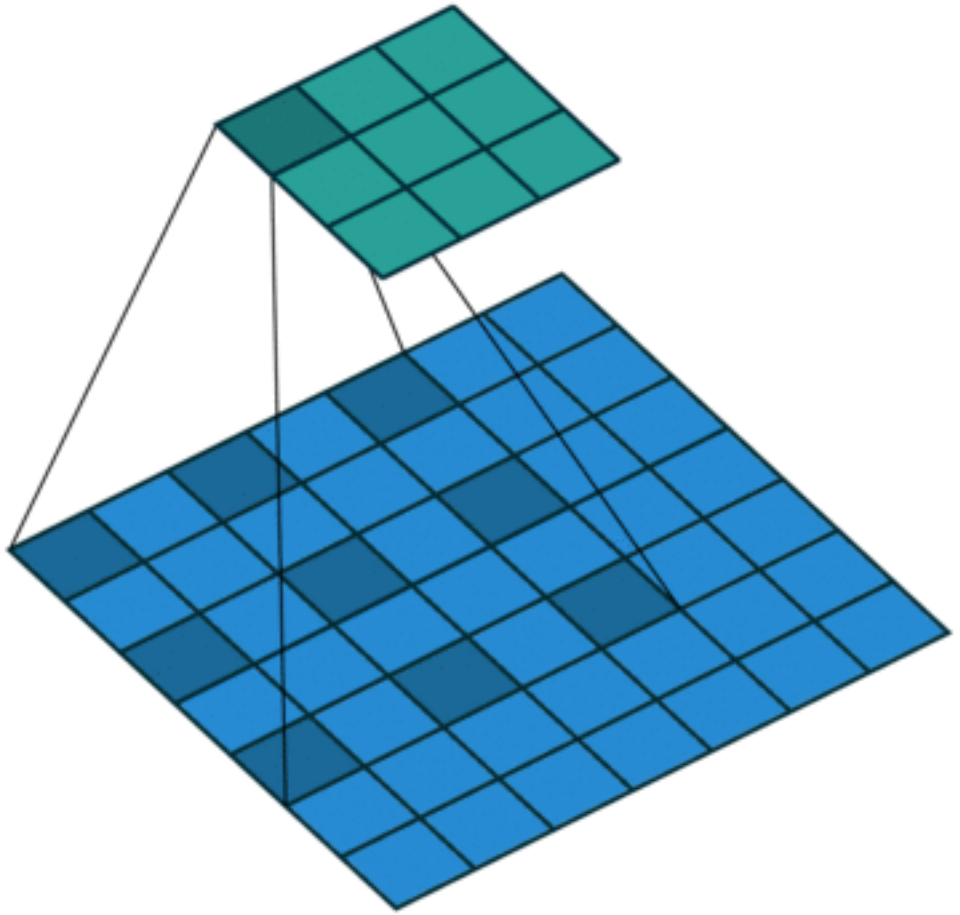
The transposed convolution *a.k.a*

- deconvolution layer
- fractionally strided convolution



Transposed Conv vs. Dilated Conv



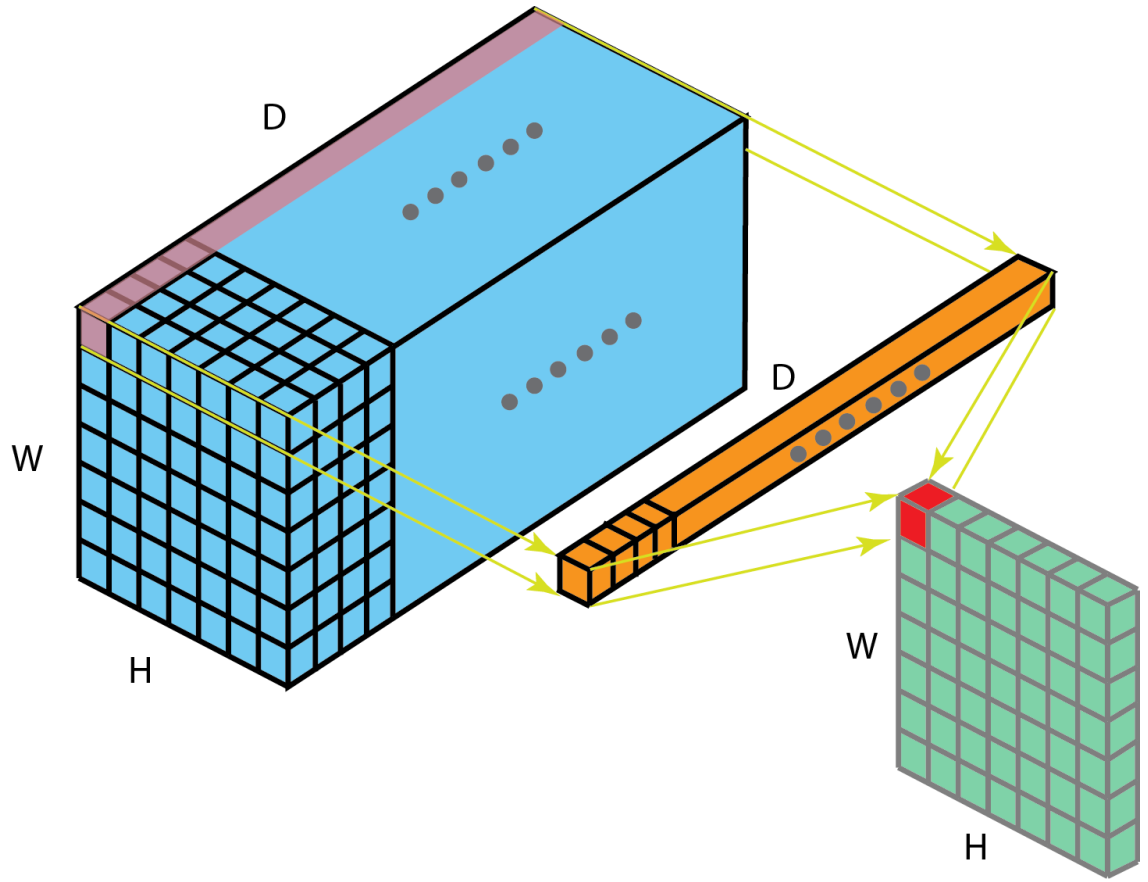


1x1 convolution

How is this not just multiplication?

Multiplications followed by a RELU activation

Good for dimensionality reduction, efficient storage



Example ConvNet

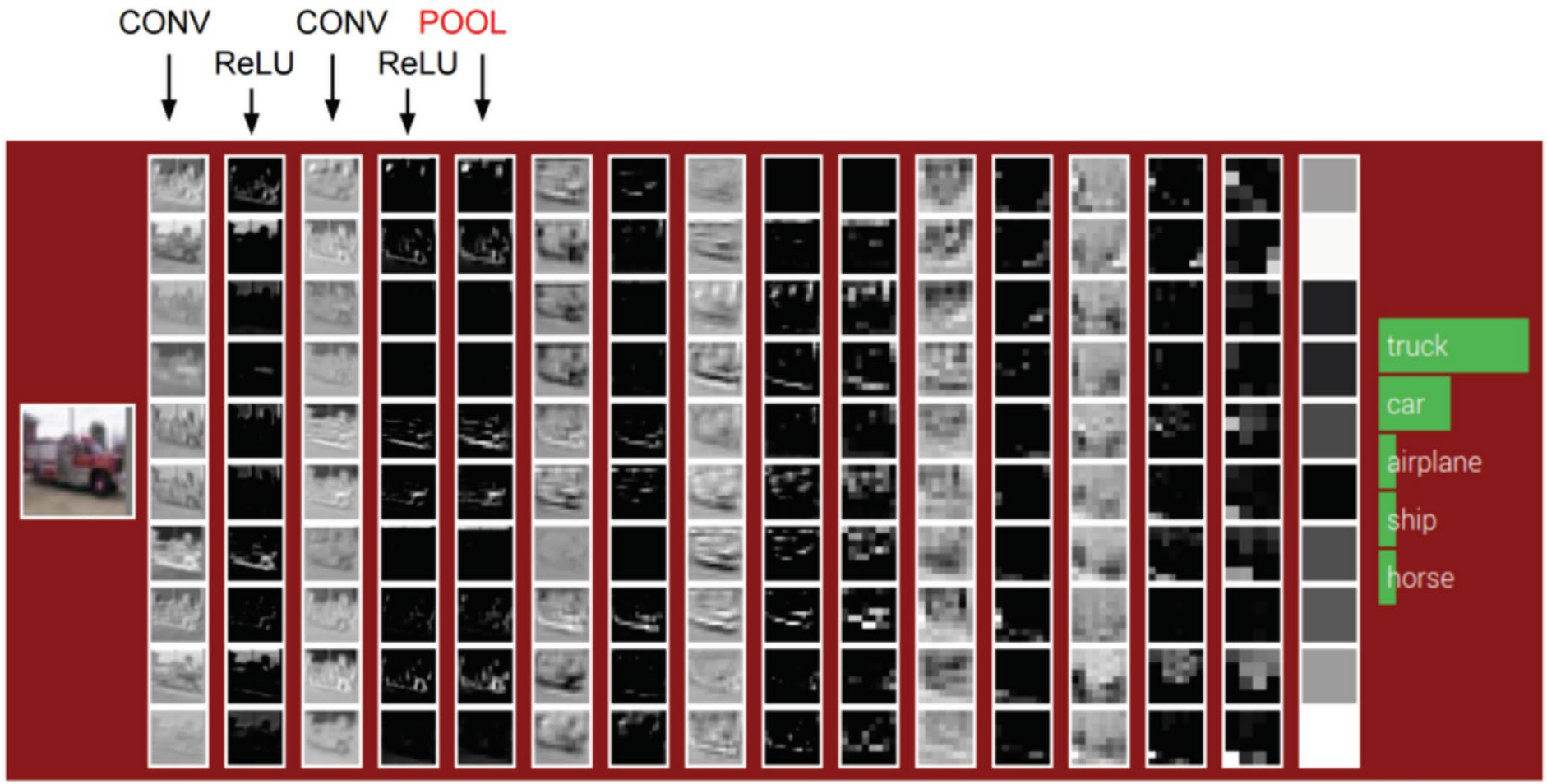


Figure: Andrej Karpathy

Example ConvNet

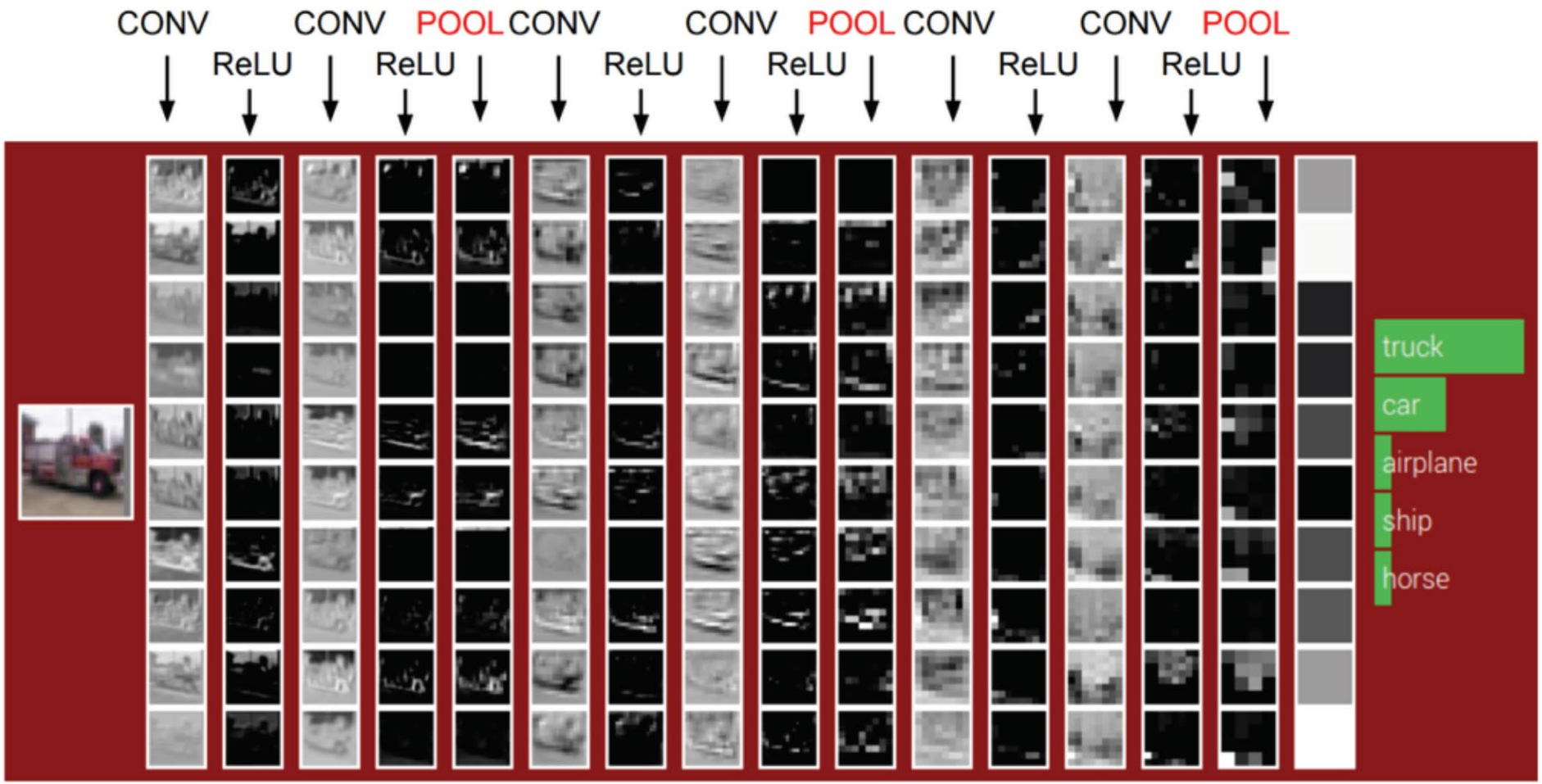


Figure: Andrej Karpathy

Example ConvNet

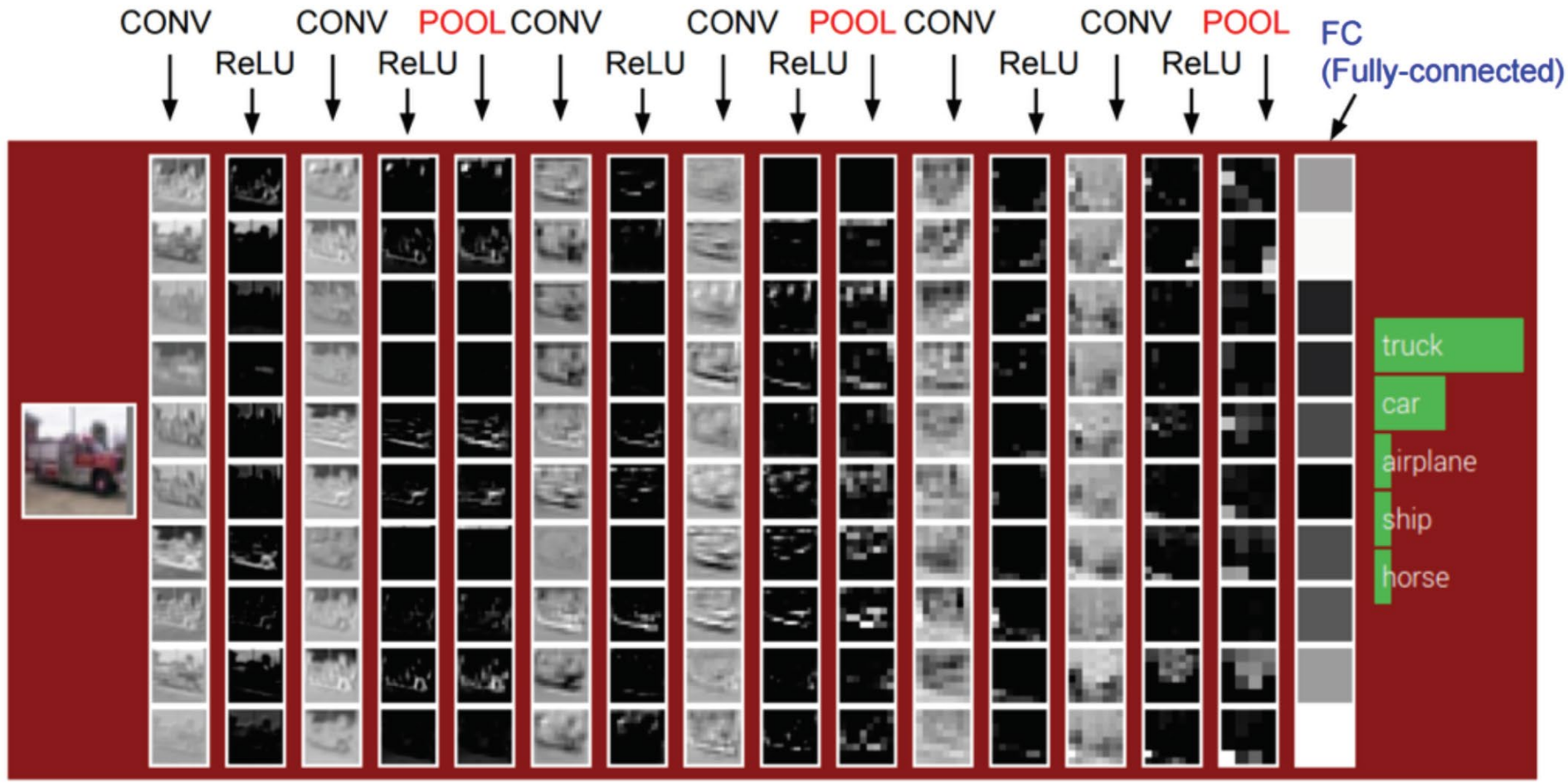
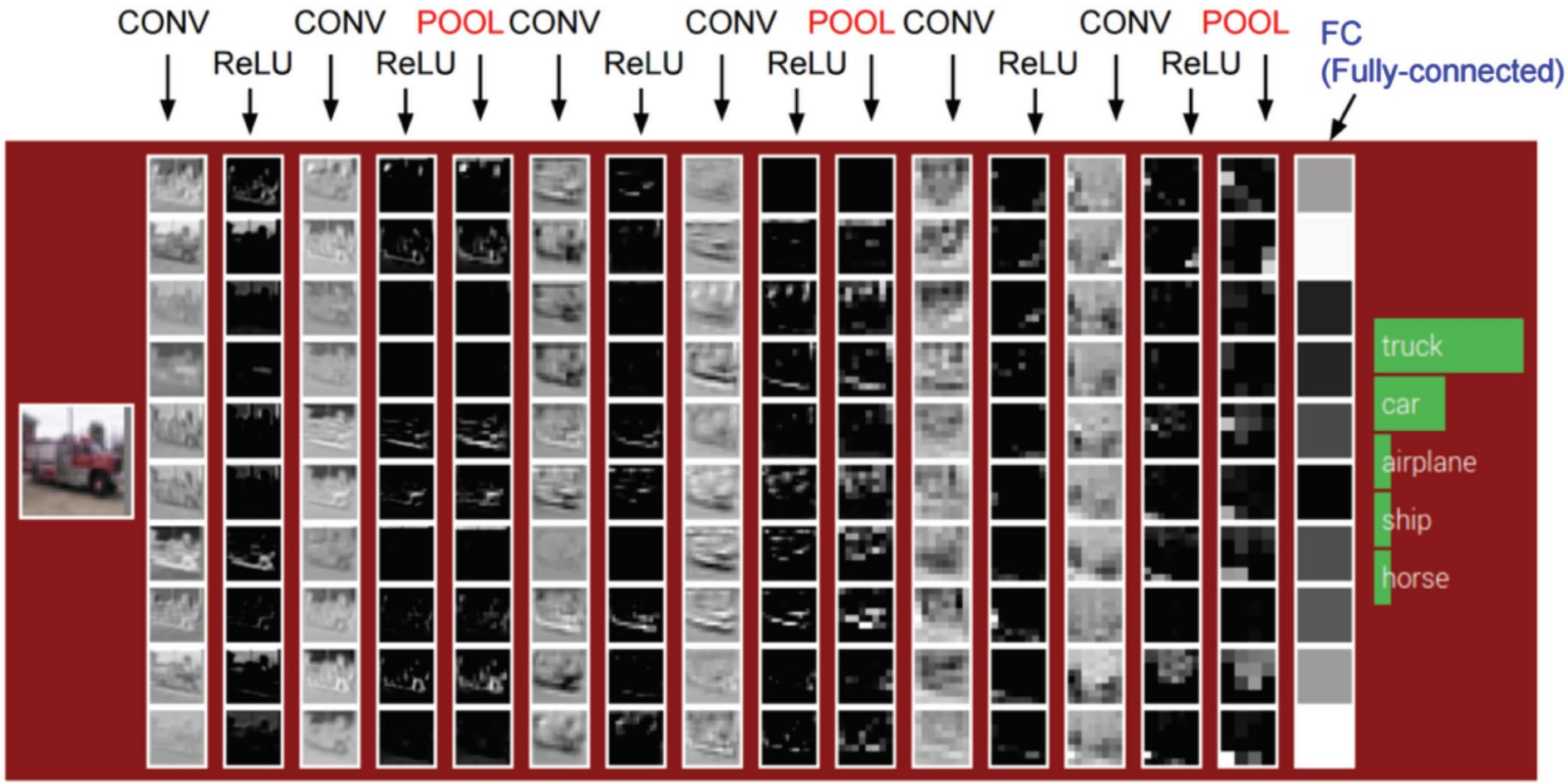


Figure: Andrej Karpathy

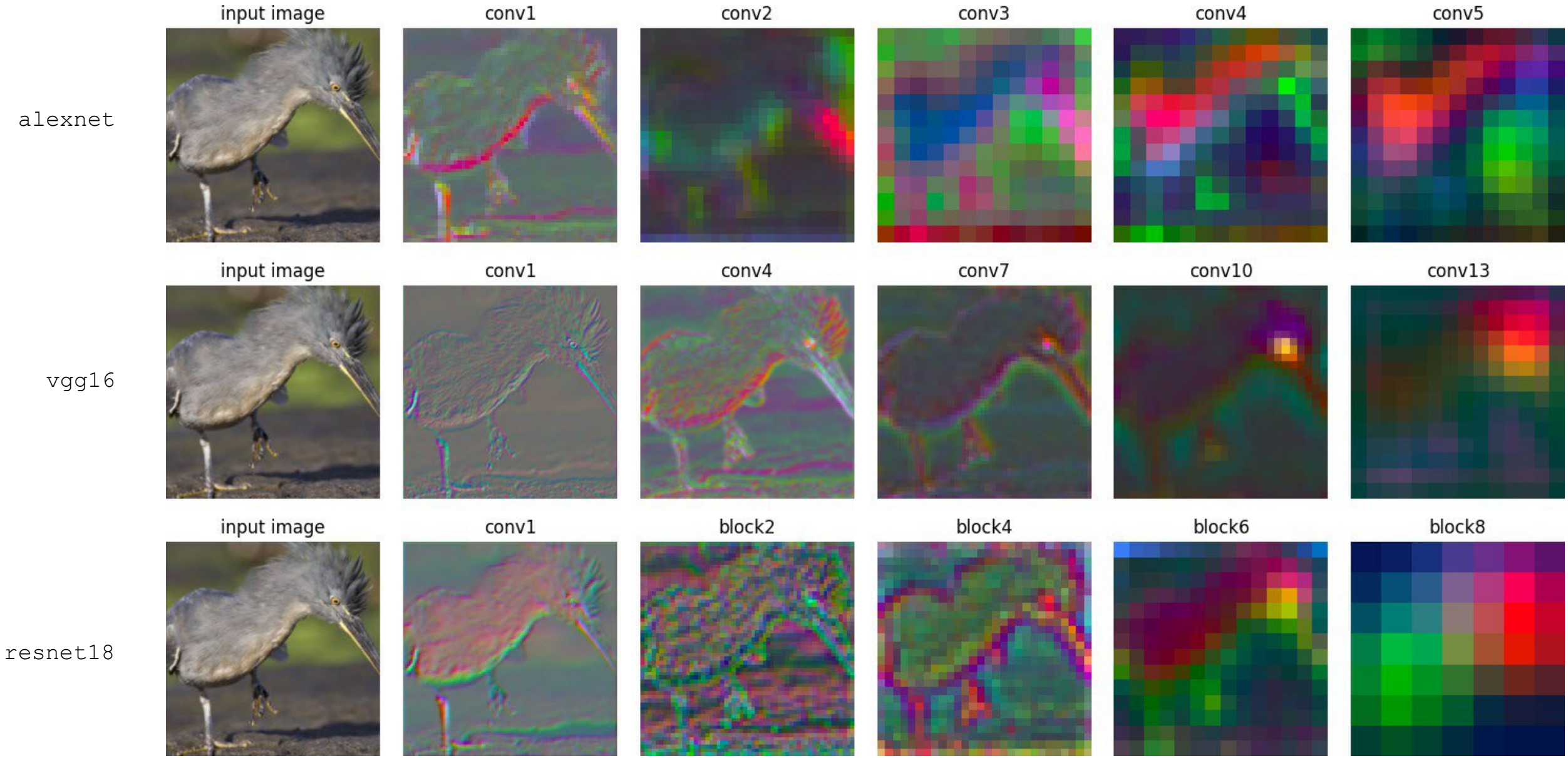
Example ConvNet



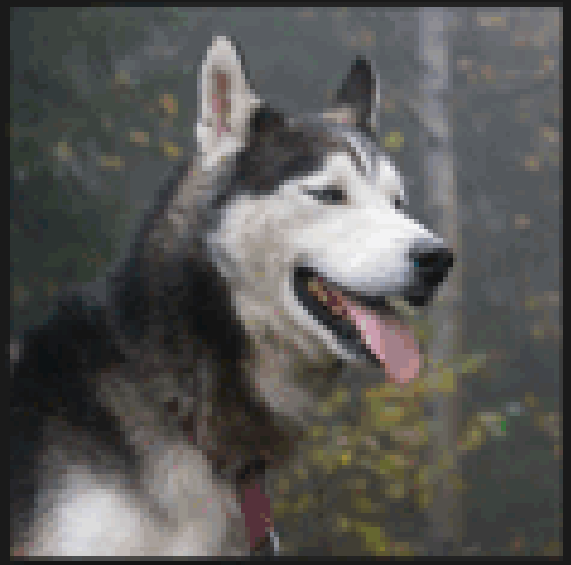
10x3x3 conv filters, stride 1, pad 1

2x2 pool filters, stride 2

Figure: Andrej Karpathy

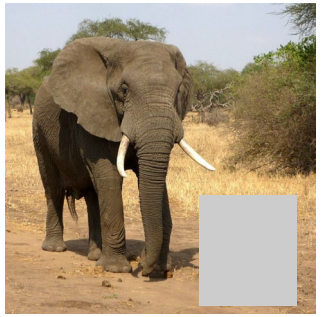

alexnet
input image
conv1
conv2
conv3
conv4
conv5
vgg16
input image
conv1
conv4
conv7
conv10
conv13
resnet18
input image
conv1
block2
block4
block6
block8

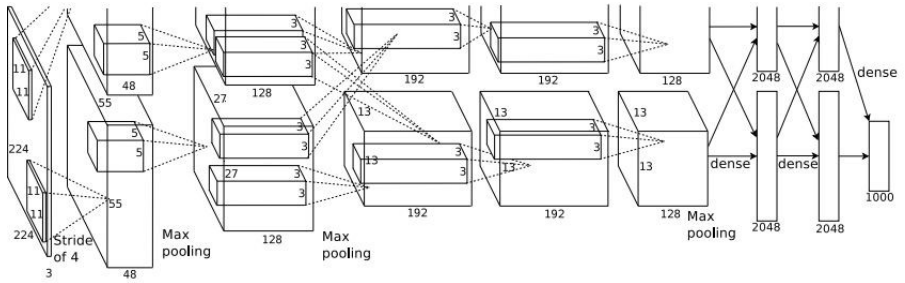
Layer Visualizations



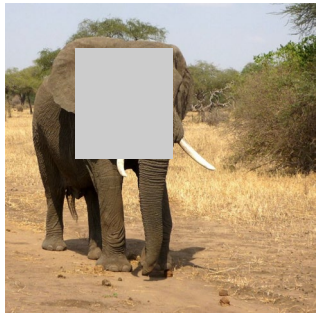
Which pixels matter: Saliency via Occlusion

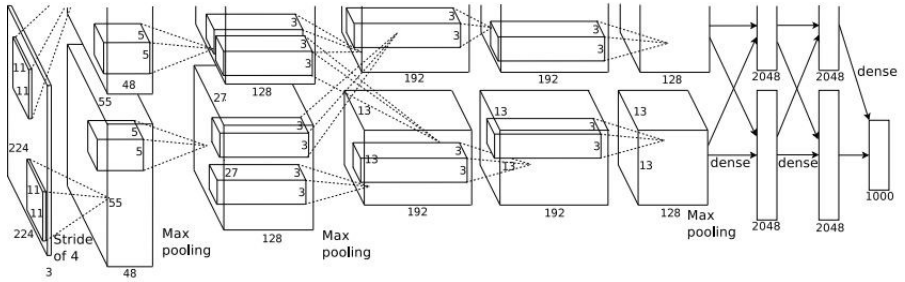
Mask part of the image before feeding to CNN,
check how much predicted probabilities change





P(elephant) = 0.95



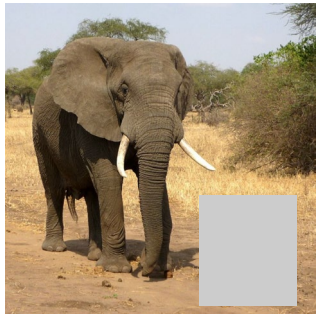


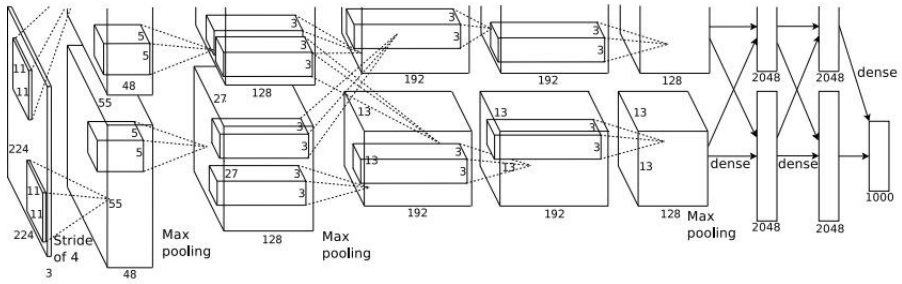
P(elephant) = 0.75

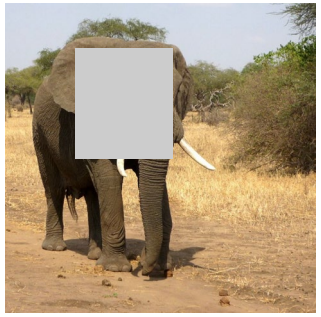
Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

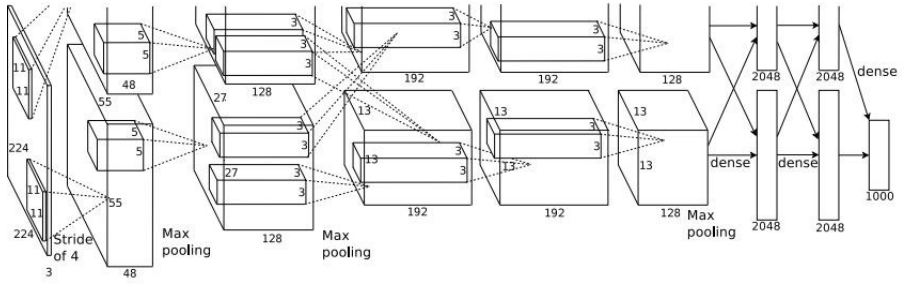
Which pixels matter: Saliency via Occlusion

Mask part of the image before feeding to CNN, check how much predicted probabilities change

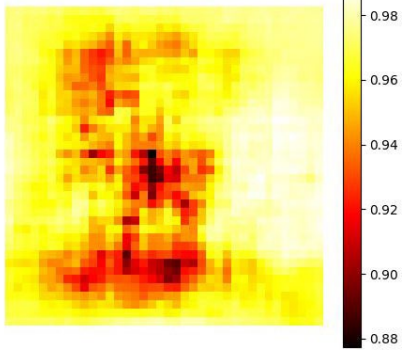




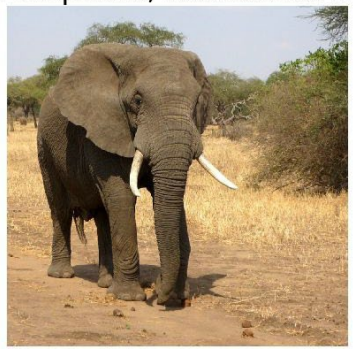


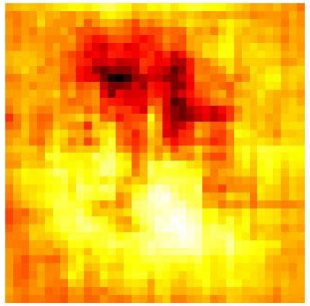


schooner

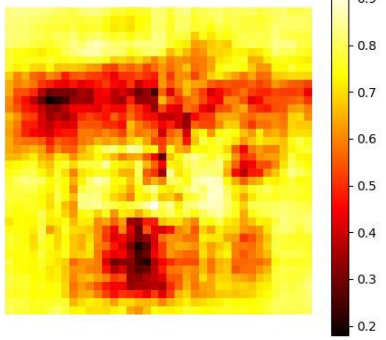


African elephant, Loxodonta africana





go-kart



Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

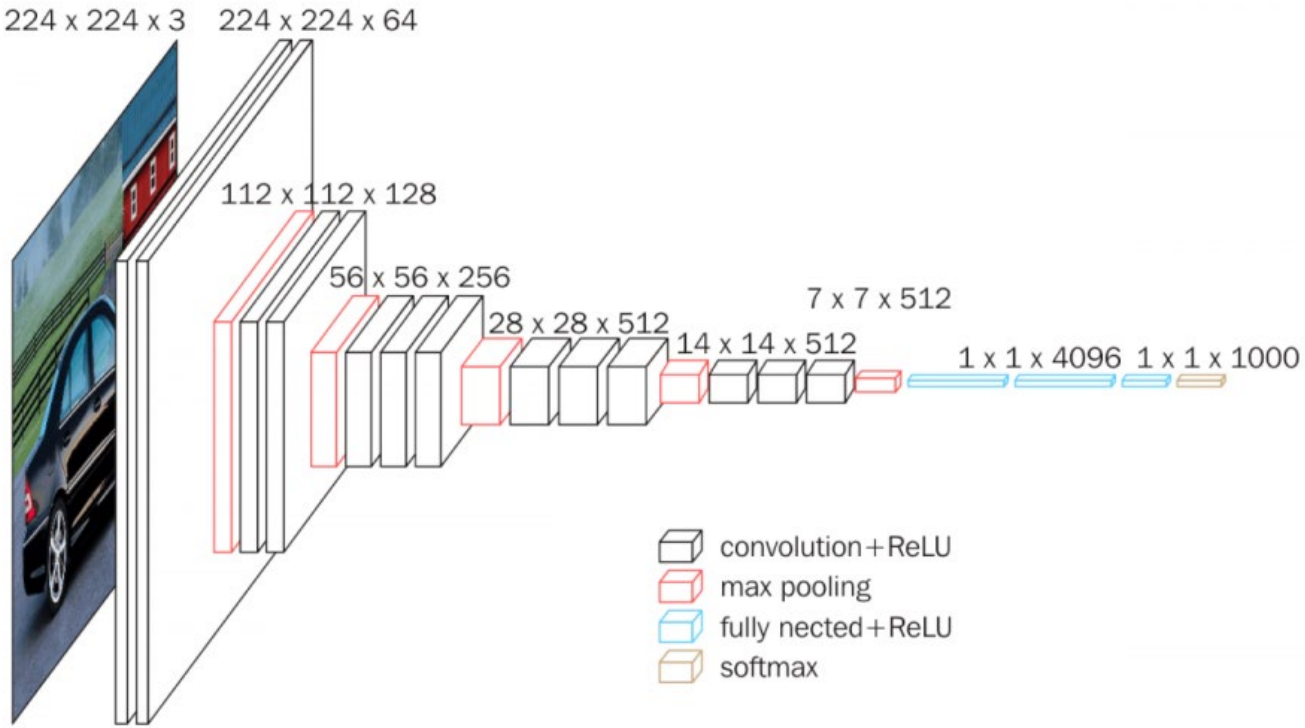
Common Architectures

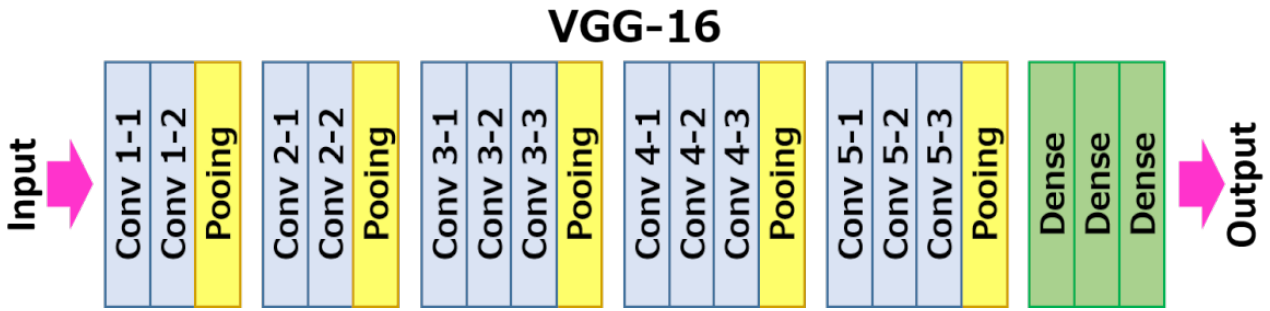
Common Architectures

(We will revisit these in detail when we study specific tasks like object detection, image generation, etc.)

VGG

- Simonyan and Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition"
- Used to be very common (before ResNets)





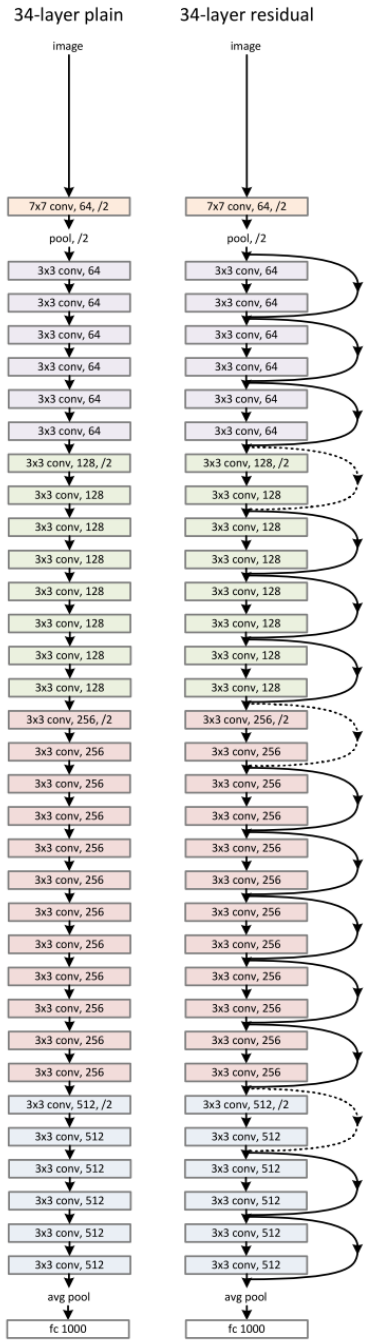
ResNet

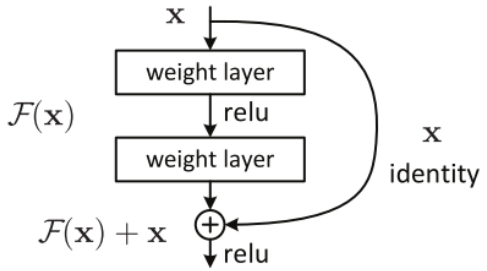
- He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian (2016). "Deep Residual Learning for Image Recognition" (PDF). Proc. Computer Vision and Pattern Recognition (CVPR), IEEE.

- Deep networks with more layers does not always mean better performance (vanishing gradient problem)

- Residual blocks = has skip connections

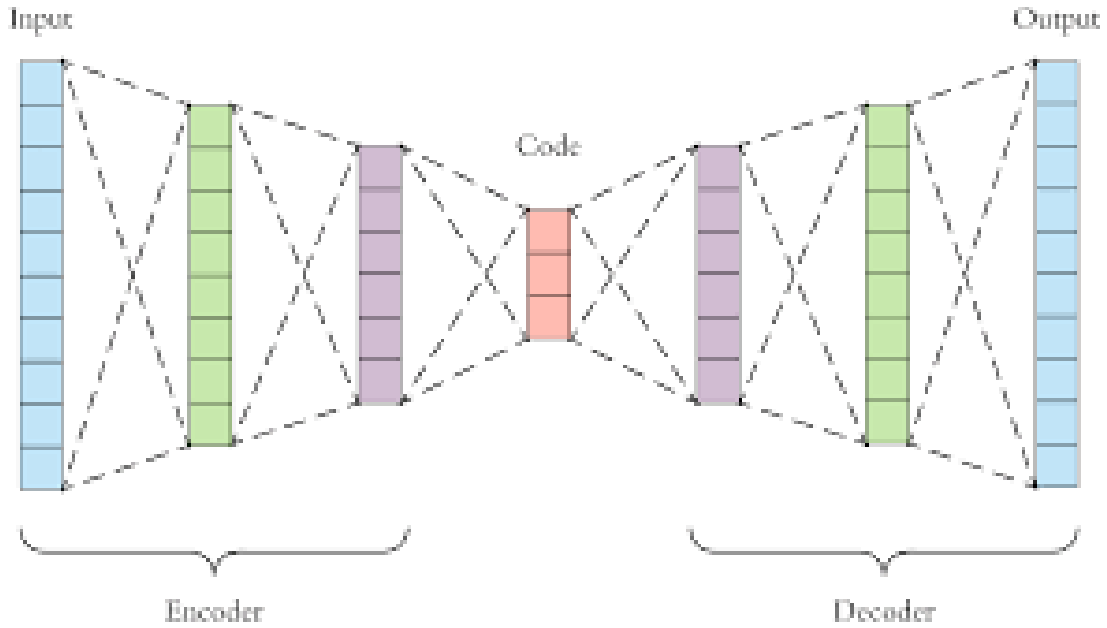
- Skipped layers train faster at the beginning, then later are filled out





Autoencoder

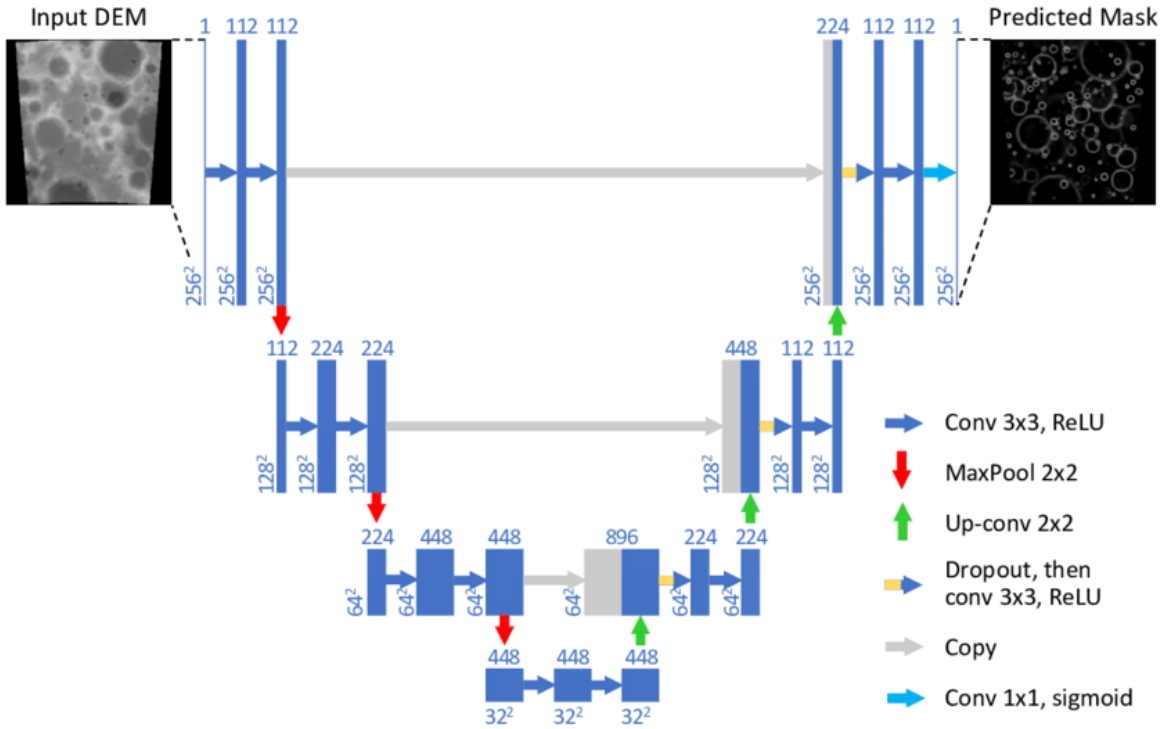
- Can be done with either fully connected or convolutional layers
- Idea is to reduce the input to a bottleneck or latent code, then reconstruct it again
- Sometimes can be used to train a feature extractor by enforcing the output = input, and then use the first part of the network as a feature extractor



U-Net

- Common architecture for image reconstruction tasks

- Features skip connections and transposed convolutions (up-conv)



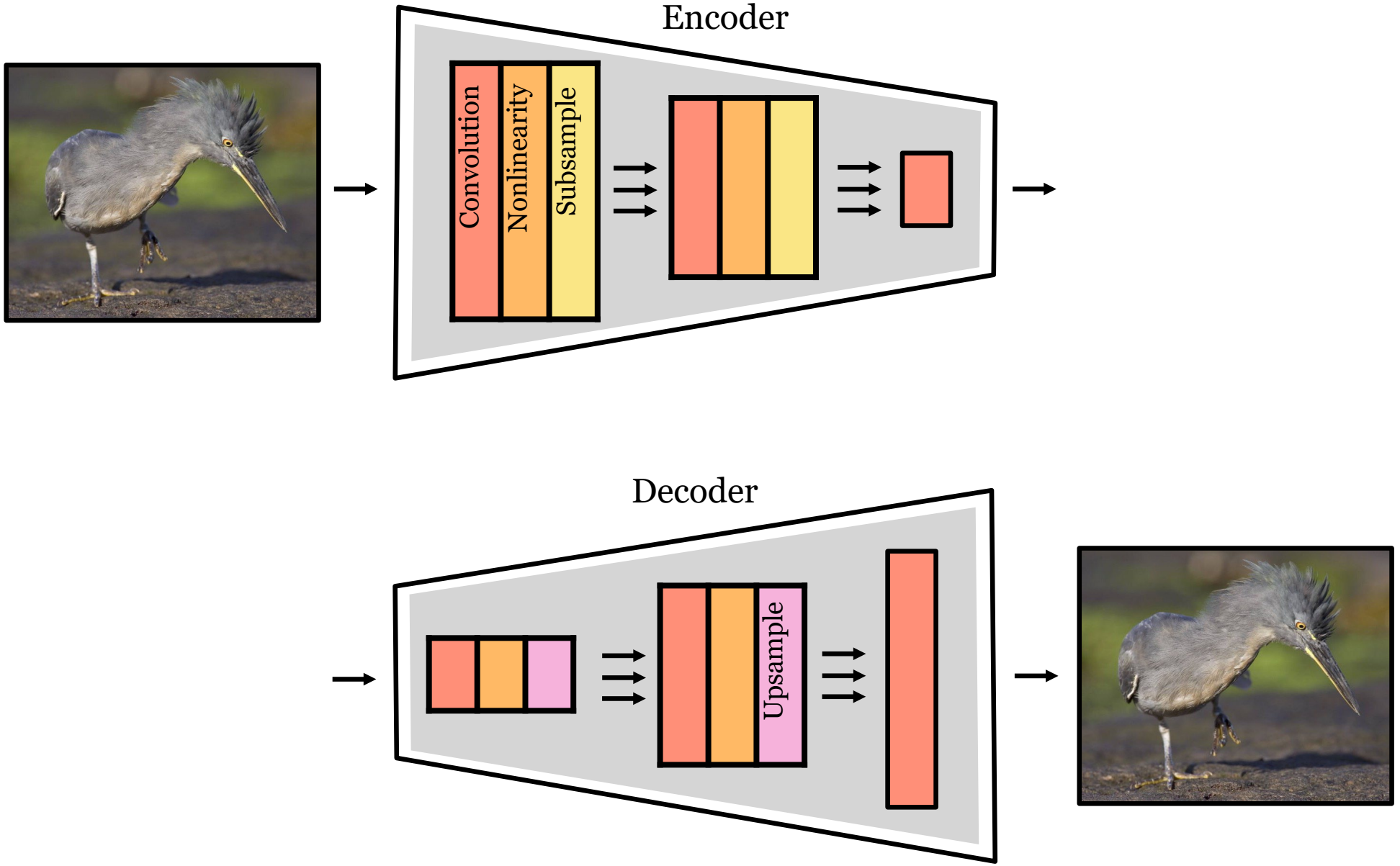

Encoder
Convolution
Nonlinearity
Subsample
Decoder
Upsample

Image-to-image

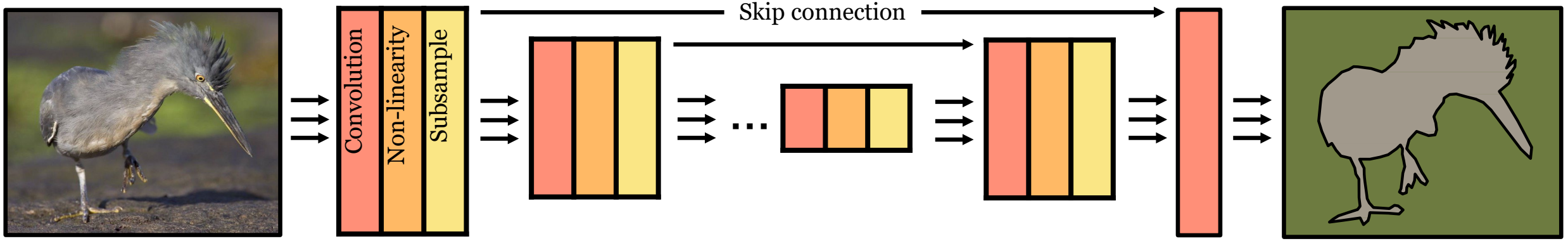
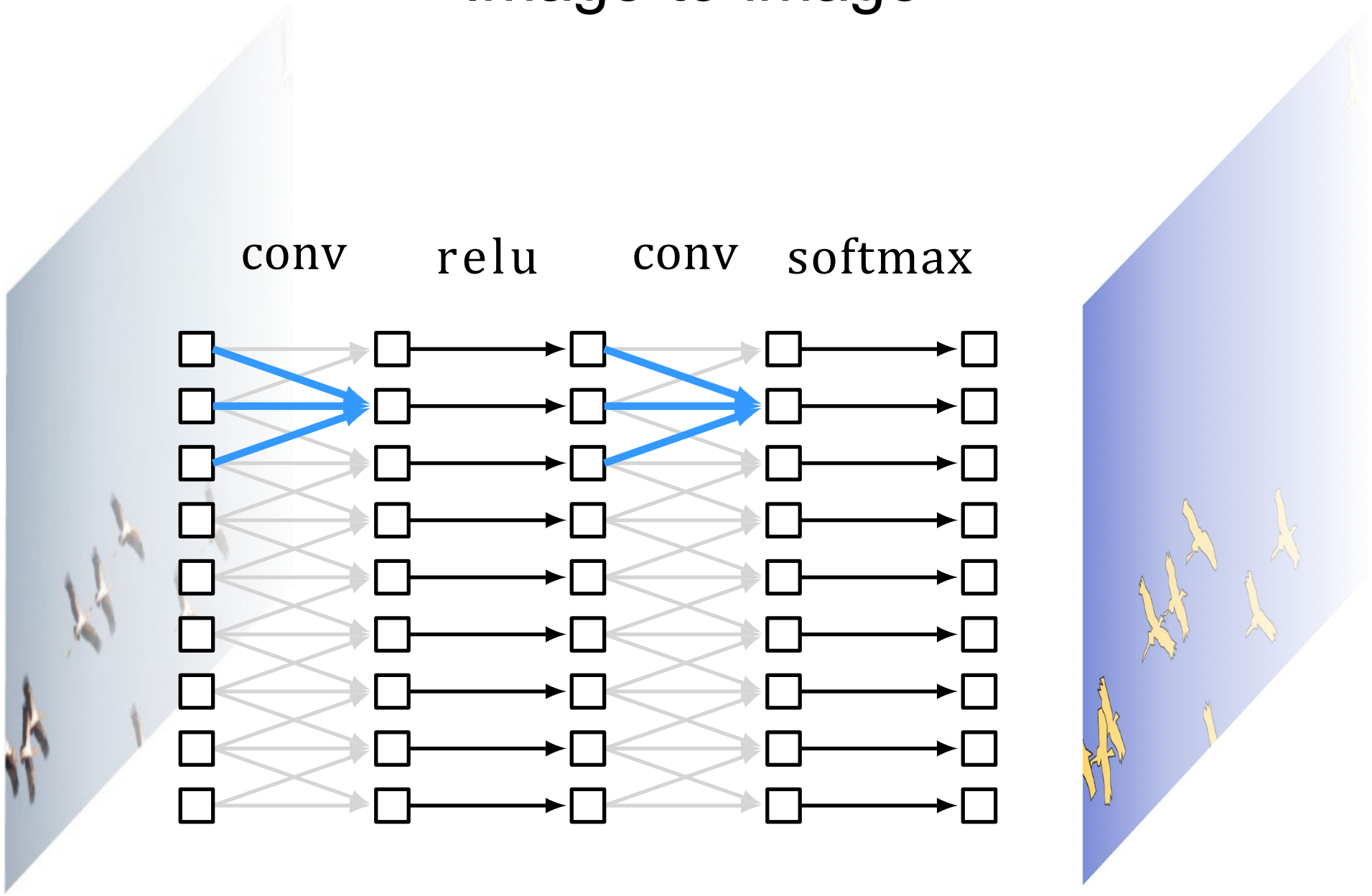
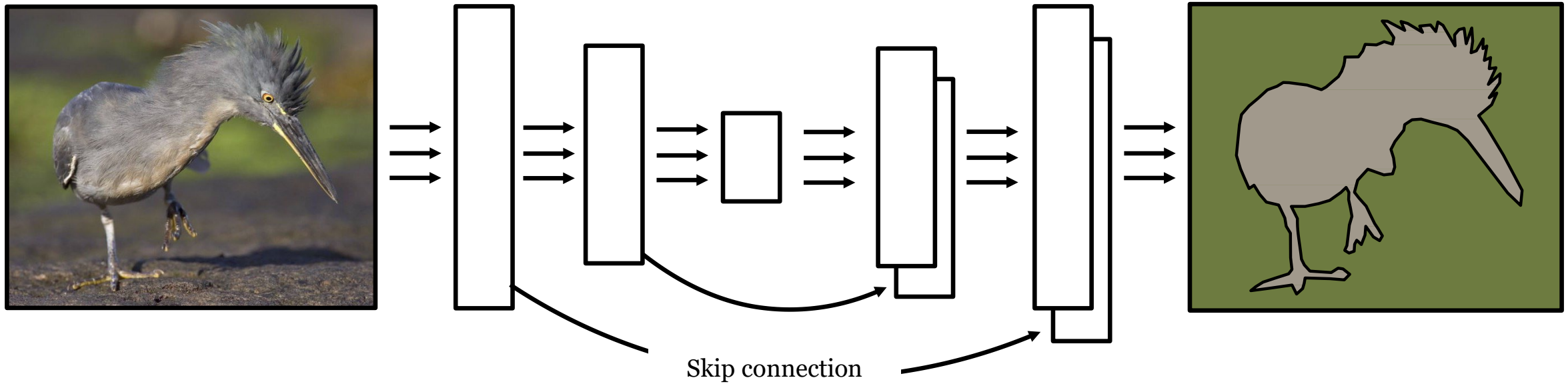


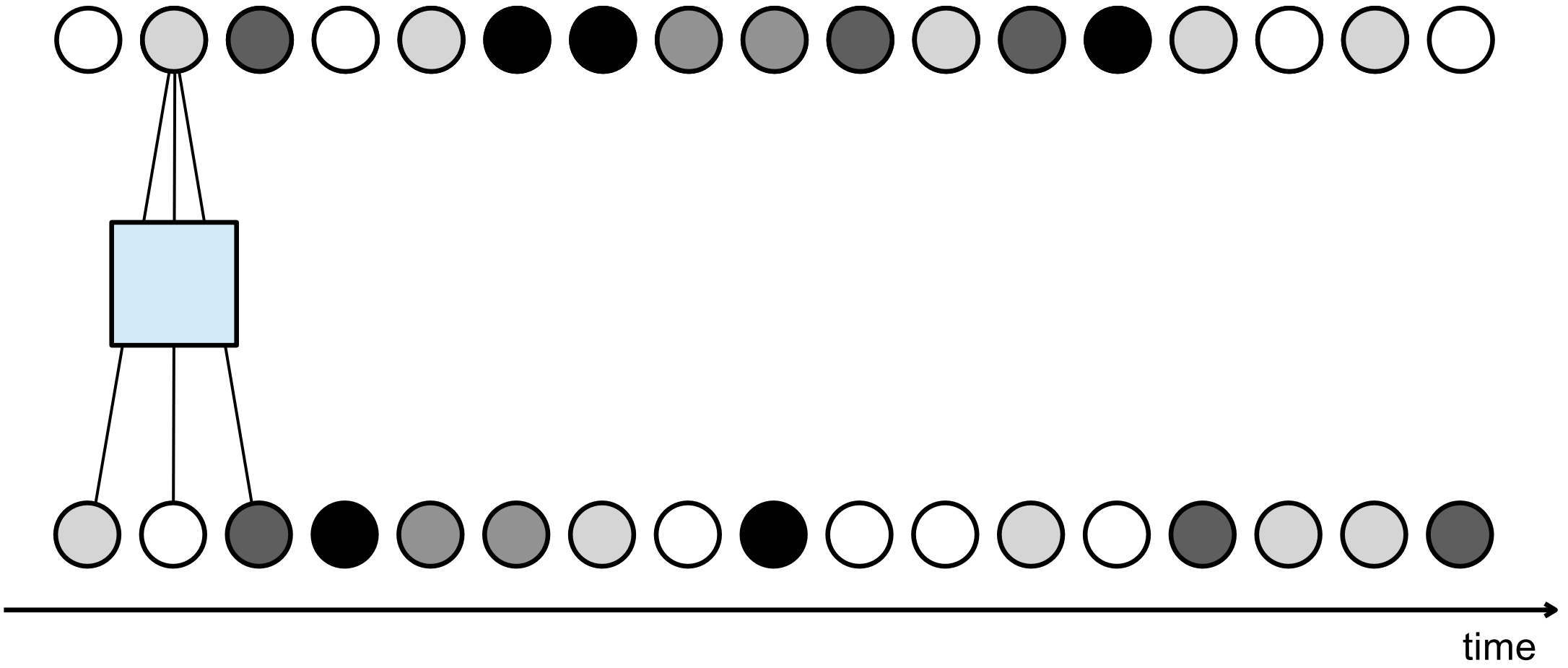
Image-to-image

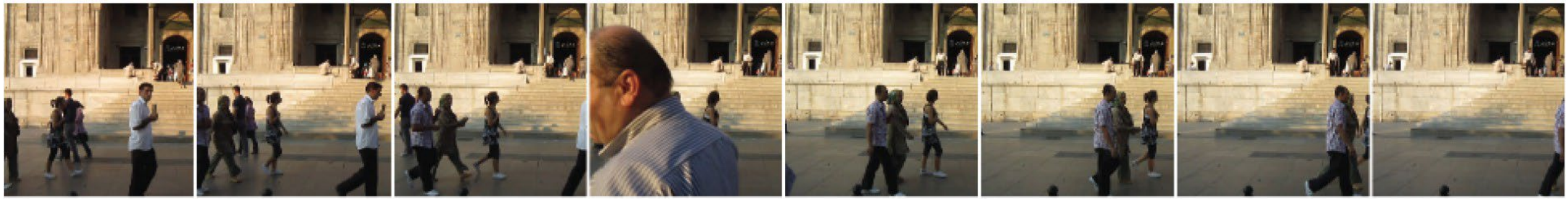


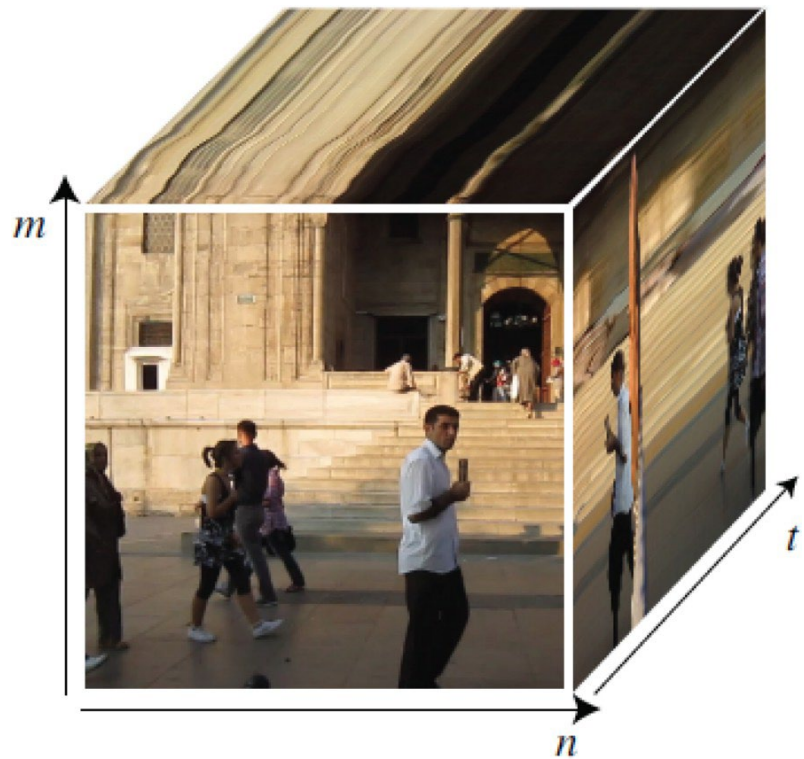
U-net



Convolutions in time





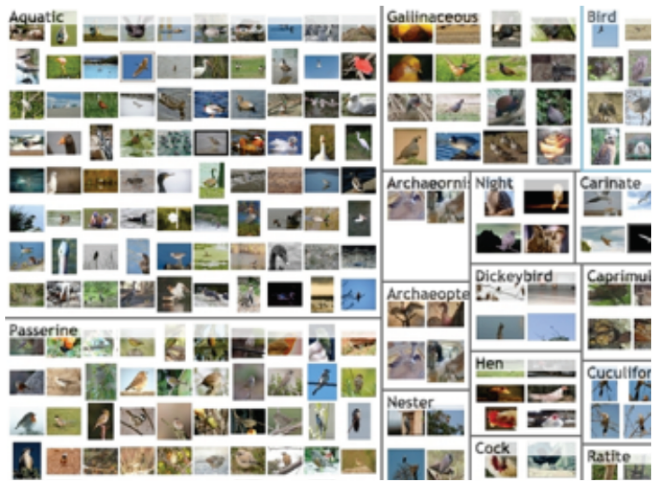

m
n
t

Training ConvNets

How do you actually train these things?

Roughly speaking:

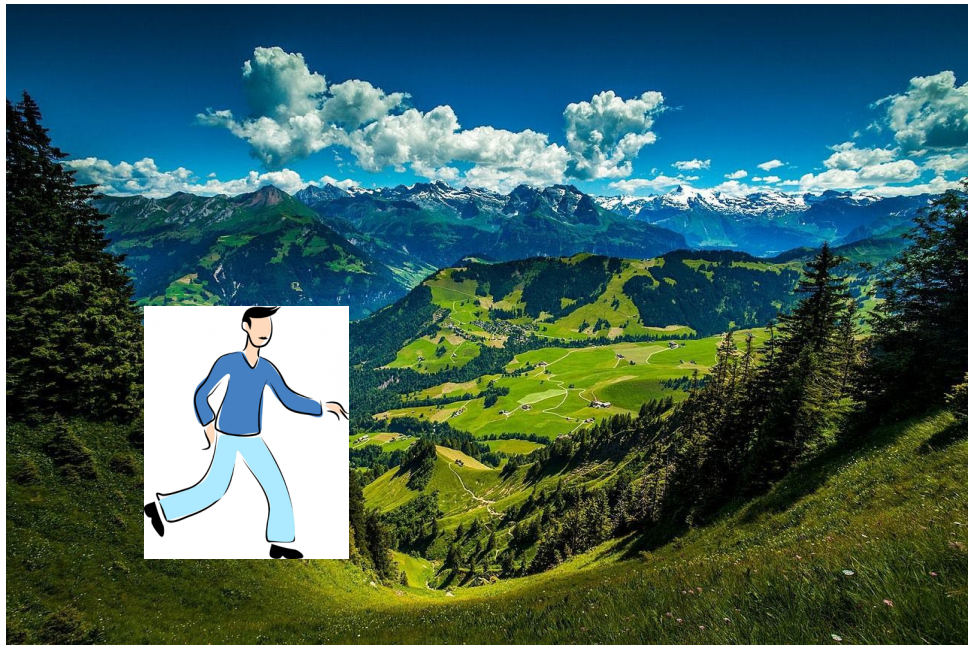
Gather labeled data

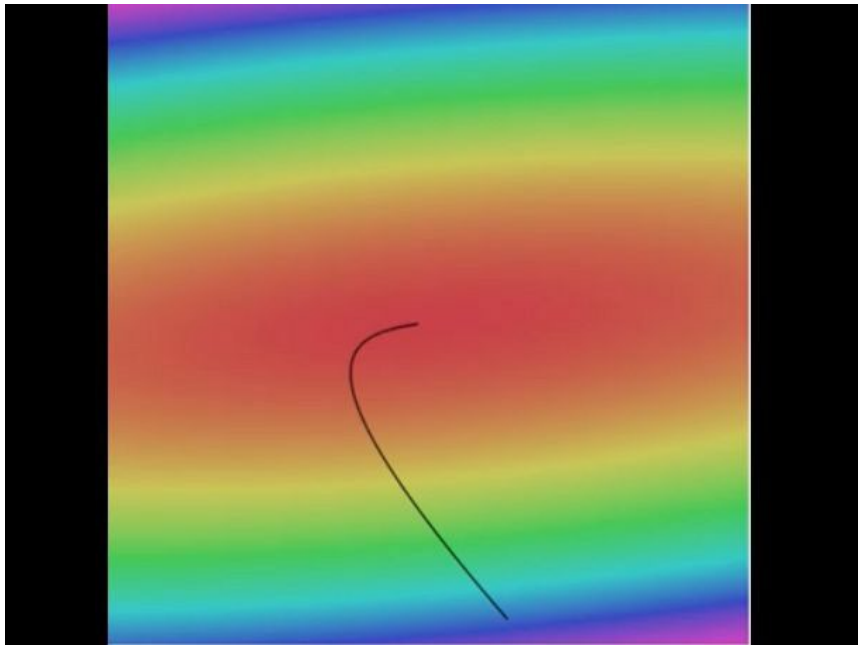


Find a ConvNet architecture

Minimize the loss

Learning network parameters through optimization





```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Mini-batch Gradient Descent

Loop:

1. Sample a batch of training data (~100 images)
2. Forwards pass: compute loss (avg. over batch)
3. Backwards pass: compute gradient
4. Update all parameters

Note: usually called “stochastic gradient descent” even though SGD has a batch size of 1

Training a convolutional neural network

- Split and preprocess your data
- Choose your network architecture
- Initialize the weights
- Find a learning rate and regularization strength
- Minimize the loss and monitor progress
- Fiddle with knobs

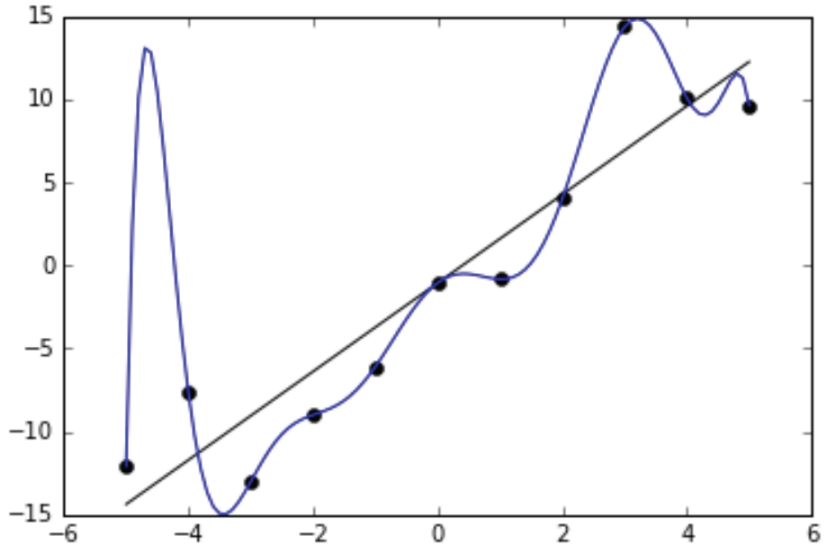
Recall: Overfitting

Overfitting

Overfitting: modeling noise in the training set instead of the "true" underlying relationship

Underfitting: insufficiently modeling the relationship in the training set

General rule: models that are "bigger" or have more capacity are more likely to overfit

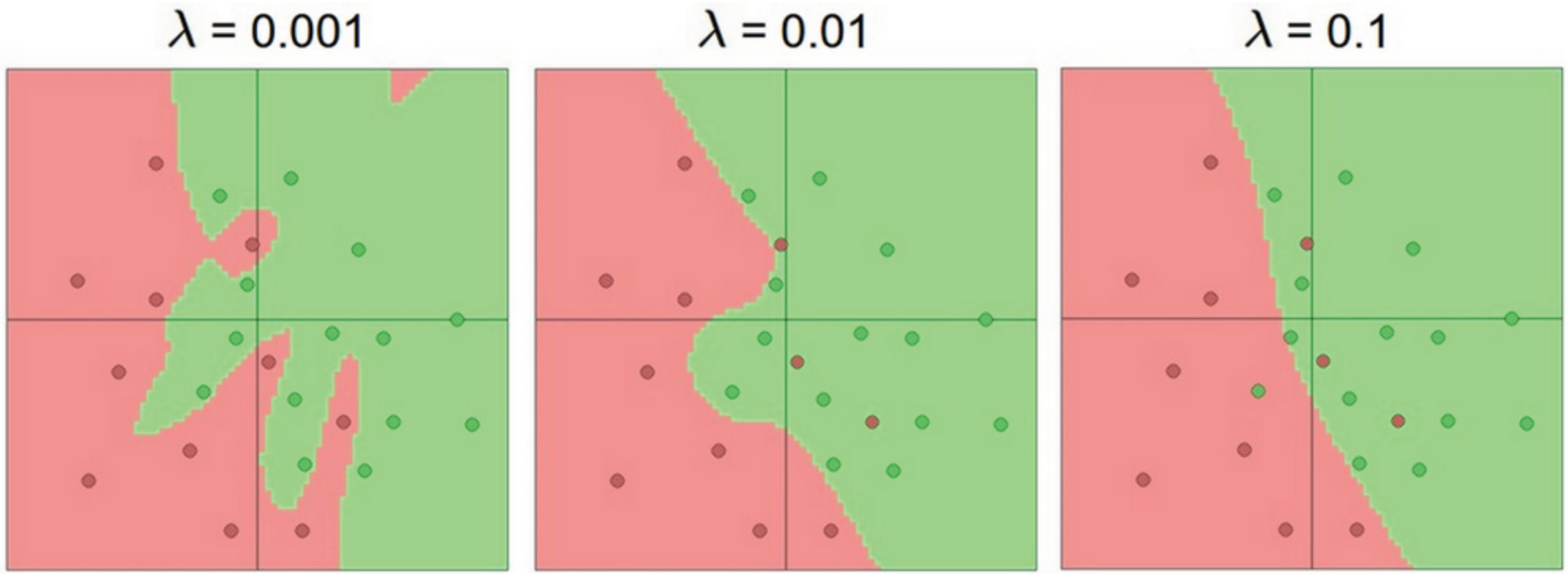


[Image: https://en.wikipedia.org/wiki/File:Overfitted_Data.png]

Regularization

Regularization reduces overfitting:

$$L = L_{\text{data}} + L_{\text{reg}} \qquad L_{\text{reg}} = \lambda \frac{1}{2} ||W||_2^2$$



[Andrej Karpathy http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html]

Example Regularizers

L2 regularization $L_{\text{reg}} = \lambda \frac{1}{2} ||W||_2^2$

(L2 regularization encourages small weights)

L1 regularization $L_{\text{reg}} = \lambda ||W||_1 = \lambda \sum_{ij} |W_{ij}|$

(L1 regularization encourages sparse weights:
weights are encouraged to reduce to exactly zero)

"Elastic net" $L_{\text{reg}} = \lambda_1 ||W||_1 + \lambda_2 ||W||_2^2$

(combine L1 and L2 regularization)

Max norm

Clamp weights to some max norm $||W||_2^2 \leq c$

“Weight decay”

Regularization is also called “weight decay” because the weights “decay” each iteration:

$$L_{\text{reg}} = \lambda \frac{1}{2} \|W\|_2^2 \quad \longrightarrow \quad \frac{\partial L}{\partial W} = \lambda W$$

Gradient descent step:

$$W \leftarrow W - \alpha\lambda W - \frac{\partial L_{\text{data}}}{\partial W}$$

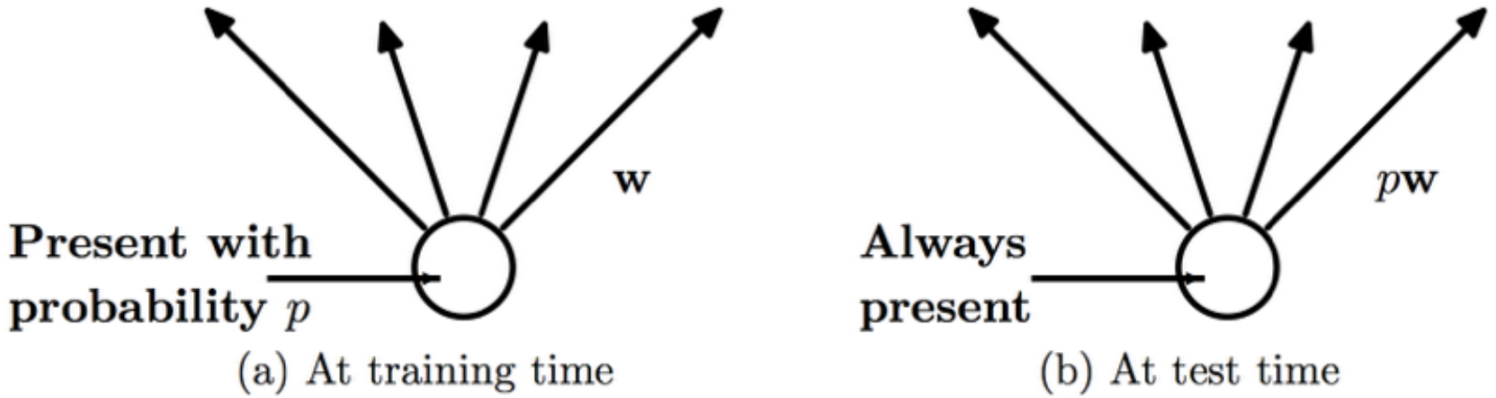
Weight decay: $\alpha\lambda$ (weights always decay by this amount)

Note: biases are sometimes excluded from regularization

[Andrej Karpathy http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html]

Dropout

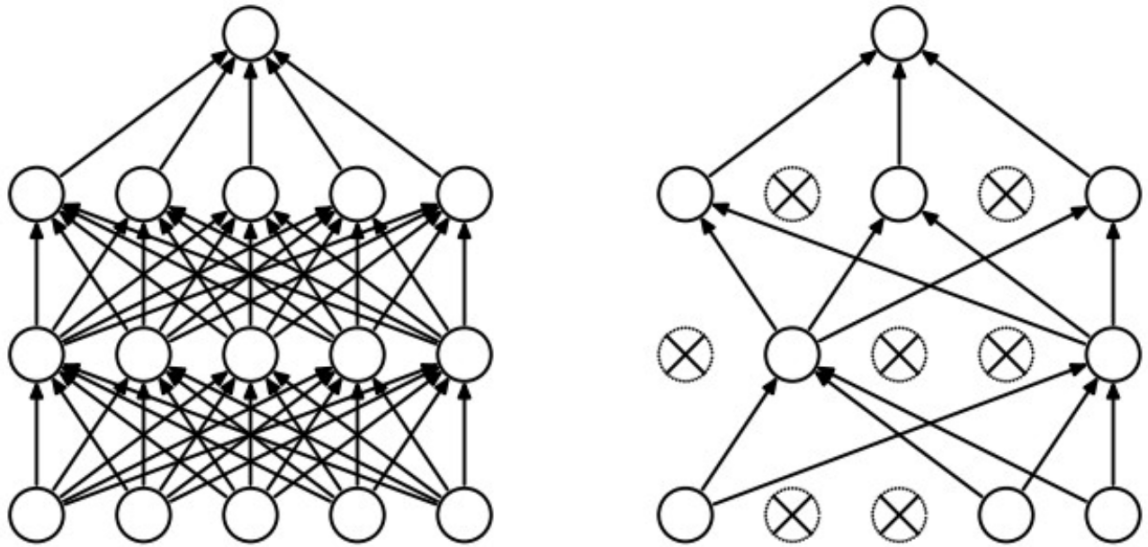
Simple but powerful technique to reduce overfitting:



[Srivasta et al, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, JMLR 2014]

Dropout

Simple but powerful technique to reduce overfitting:



(a) Standard Neural Net (b) After applying dropout.

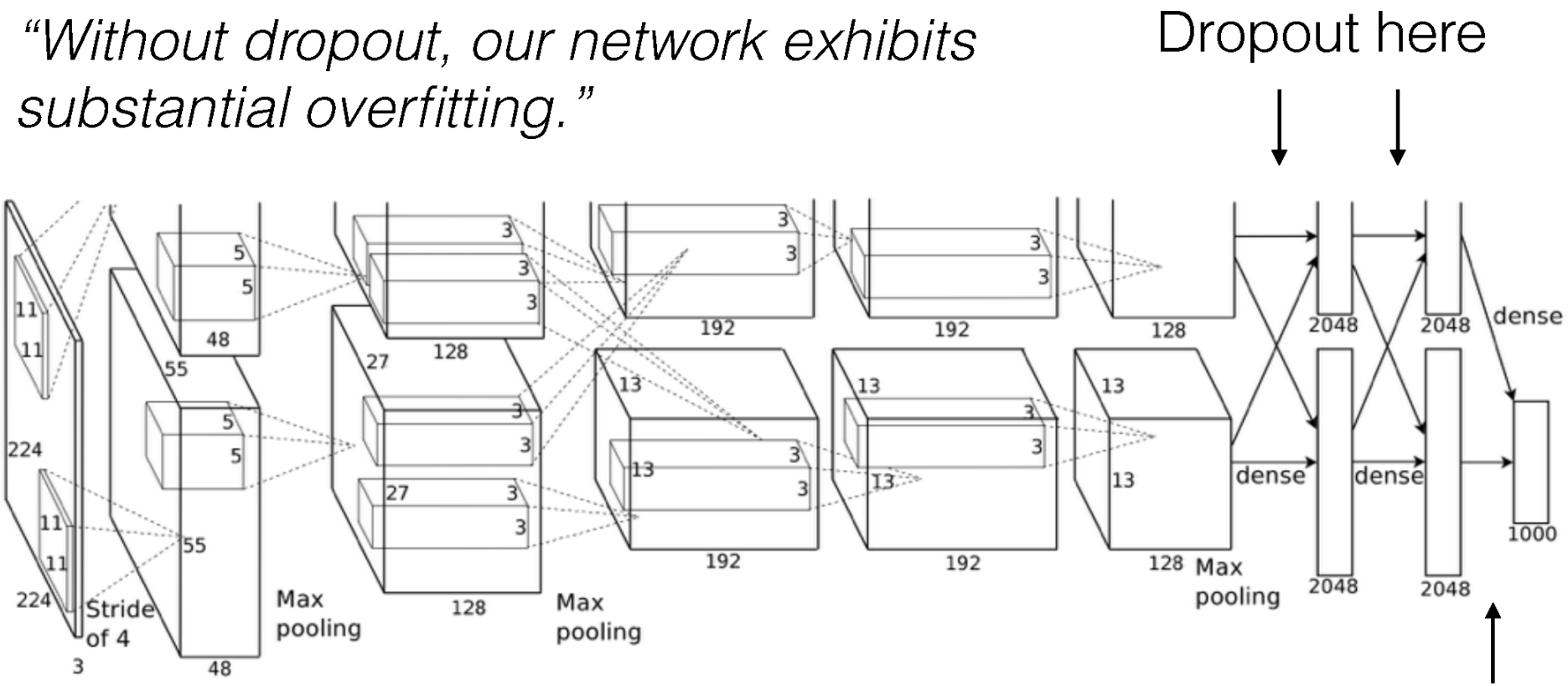
Note: Dropout can be interpreted as an approximation to taking the geometric mean of an ensemble of exponentially many models

[Srivasta et al, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, JMLR 2014]

Dropout

Case study: [Krizhevsky 2012]

"Without dropout, our network exhibits substantial overfitting."



But not here — why?

[Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012]

Summary of things to fiddle

- Network architecture
- Learning rate, decay schedule, update type
- Regularization (L2, L1, maxnorm, dropout, …)
- Loss function (softmax, SVM, …)
- Weight initialization

Neural network parameters