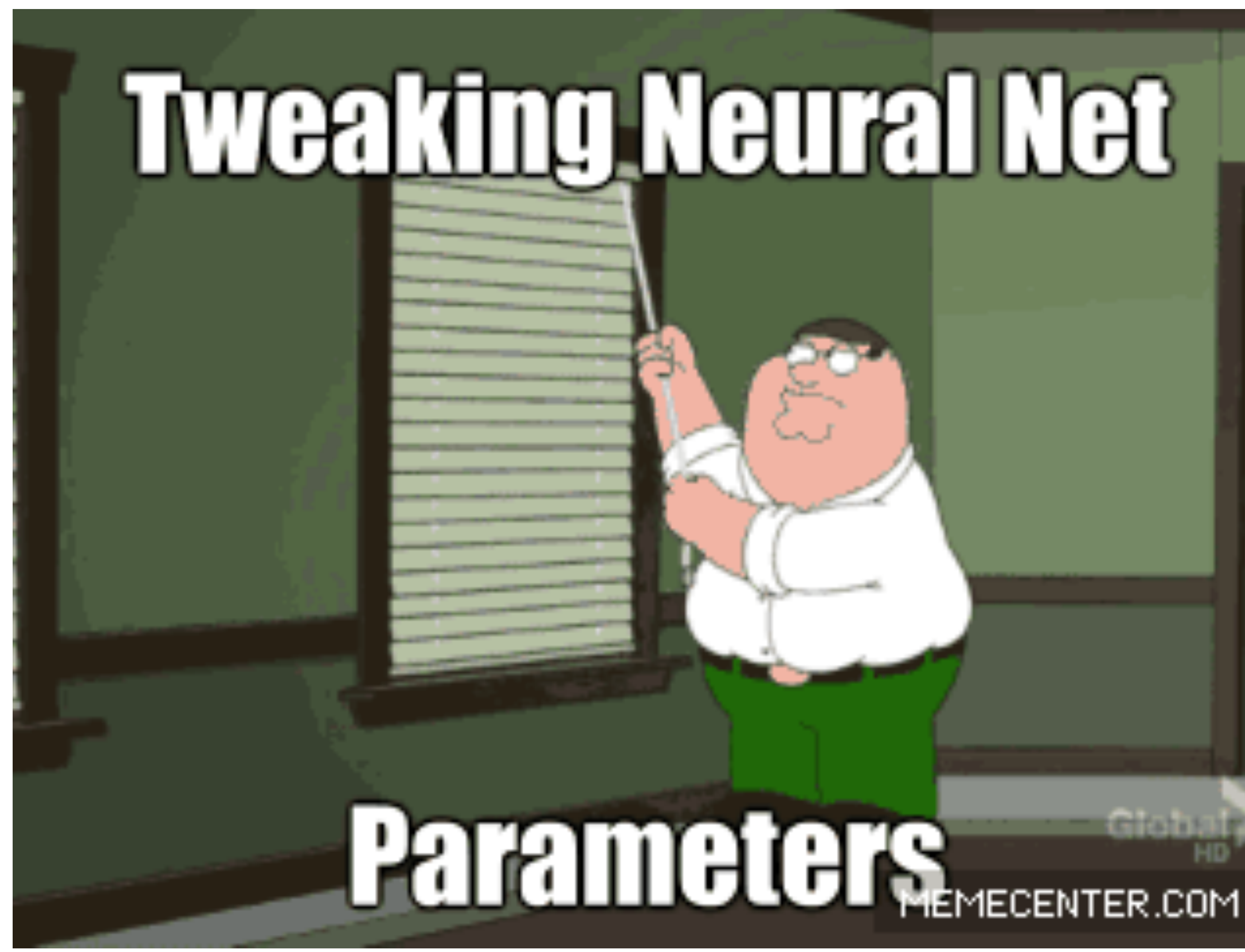
Lecture 37: ConvNets (Cont'd) and Training

CS 4670/5670
Sean Bell



[http://bbabenko.tumblr.com/post/83319141207/convolutional-learnings-things-i-learned-by]

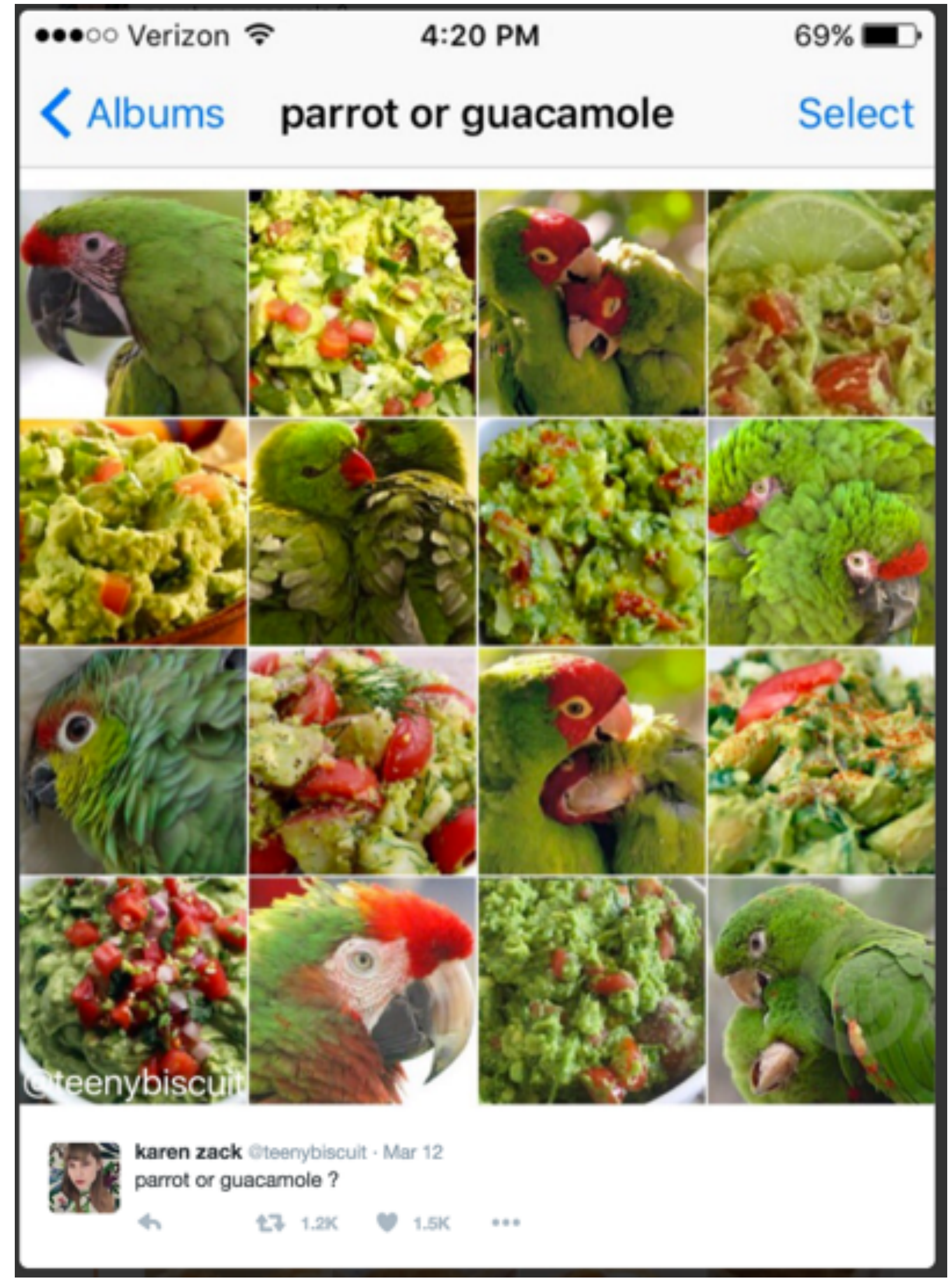
(Unrelated) Dog vs Food

[Karen Zack, @teenybiscuit]

(Unrelated) Dog vs Food

[Karen Zack, @teenybiscuit]

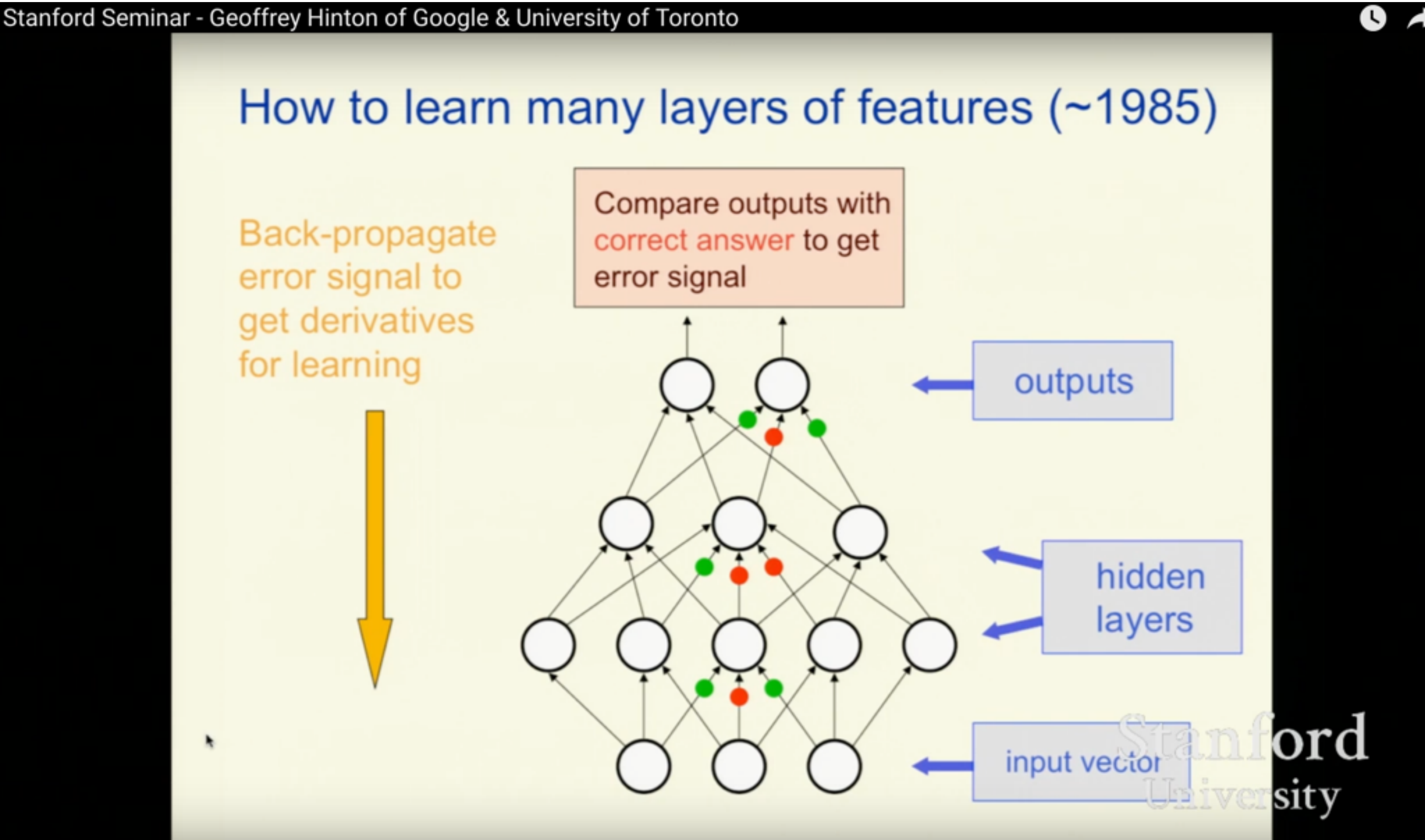
(Unrelated) Dog vs Food



[Karen Zack, @teenybiscuit]

(Recap) Backprop

From Geoff Hinton's seminar at Stanford yesterday



(Recap) Backprop

Parameters: $$\theta = \begin{bmatrix} \theta_1 & \theta_2 & \cdots \end{bmatrix}$$

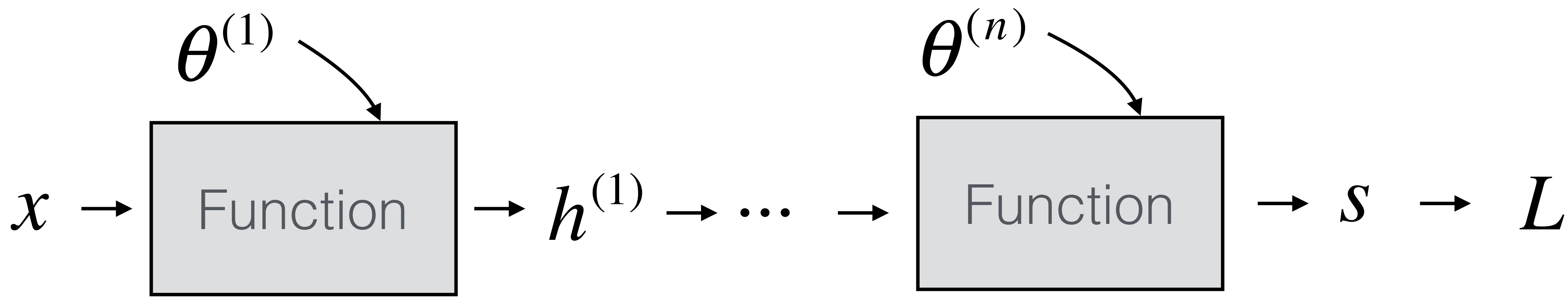
All of the weights and biases in the network, stacked together

Gradient: $$\frac{\partial L}{\partial \theta} = \begin{bmatrix} \frac{\partial L}{\partial \theta_1} & \frac{\partial L}{\partial \theta_2} & \cdots \end{bmatrix}$$

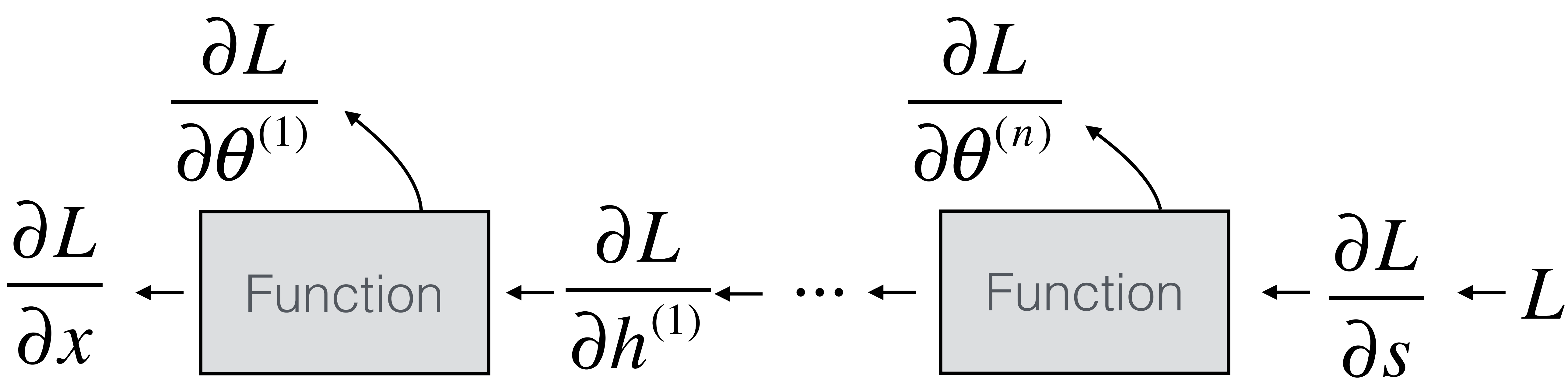
Intuition: "How fast would the error change if I change myself by a little bit"

(Recap) Backprop

Forward Propagation: compute the activations and loss

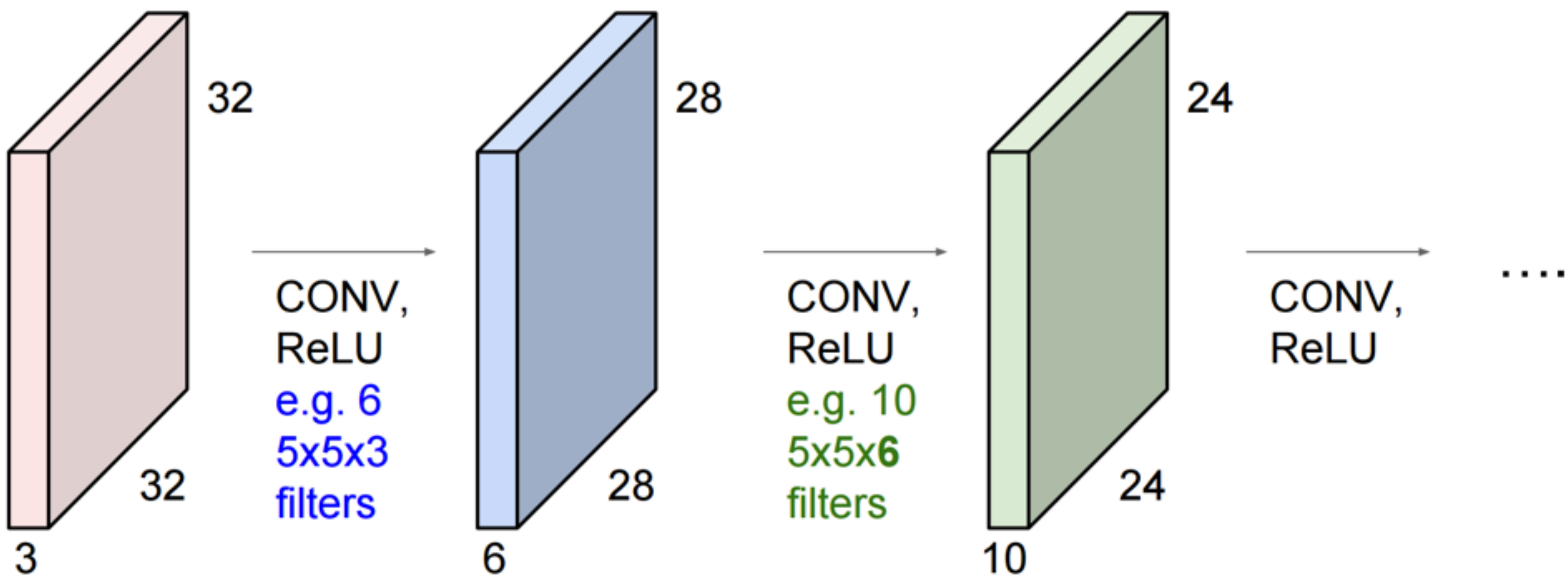


Backward Propagation: compute the gradient ("error signal")



(Recap)

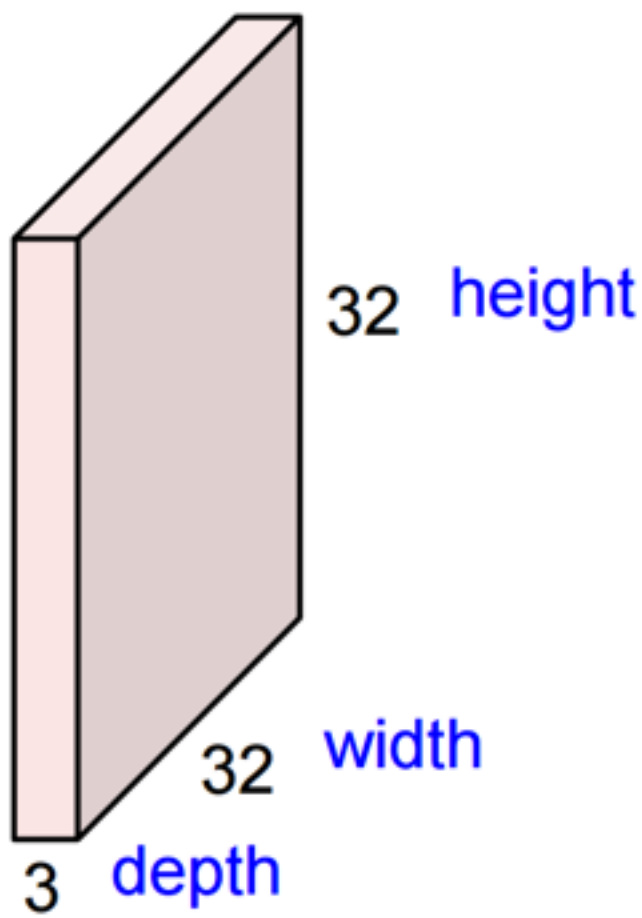
A **ConvNet** is a sequence of convolutional layers, interspersed with activation functions (and possibly other layer types)



(Recap)

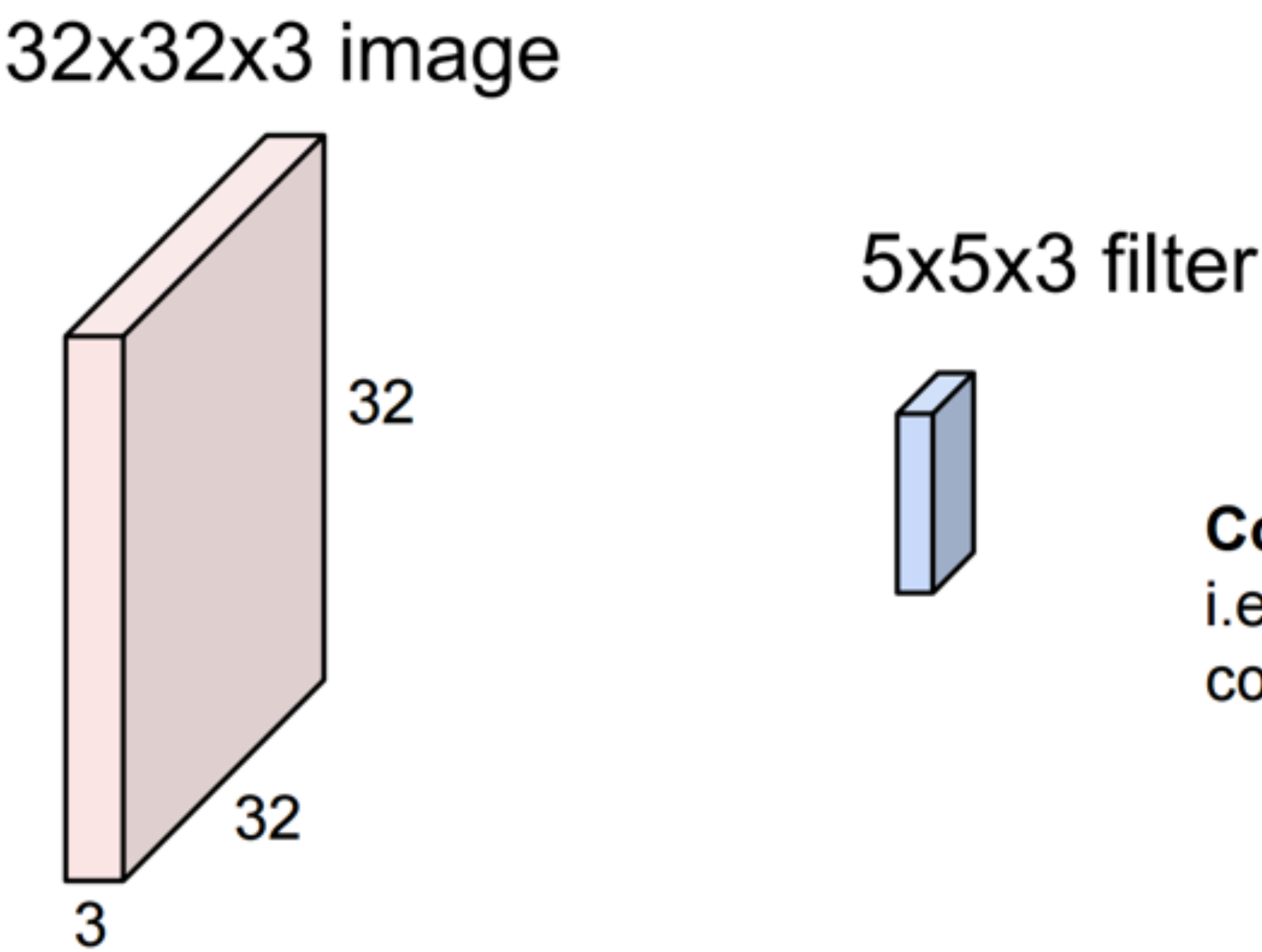
Convolution Layer

32x32x3 image



(Recap)

Convolution Layer



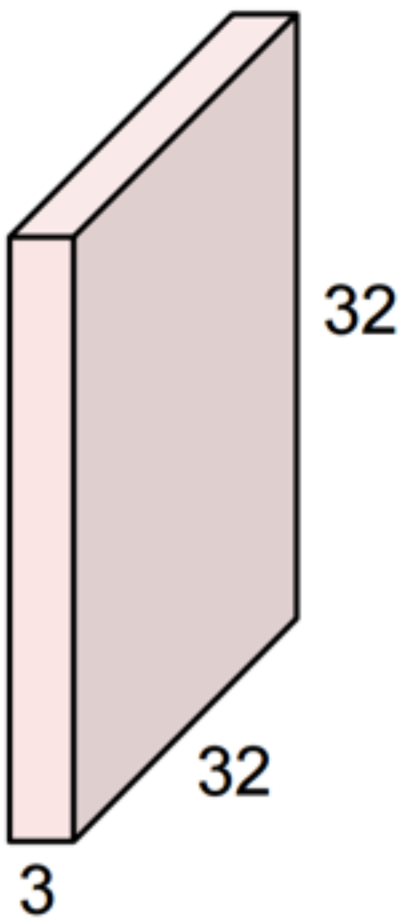
Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products"

(Recap)

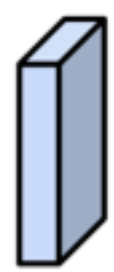
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



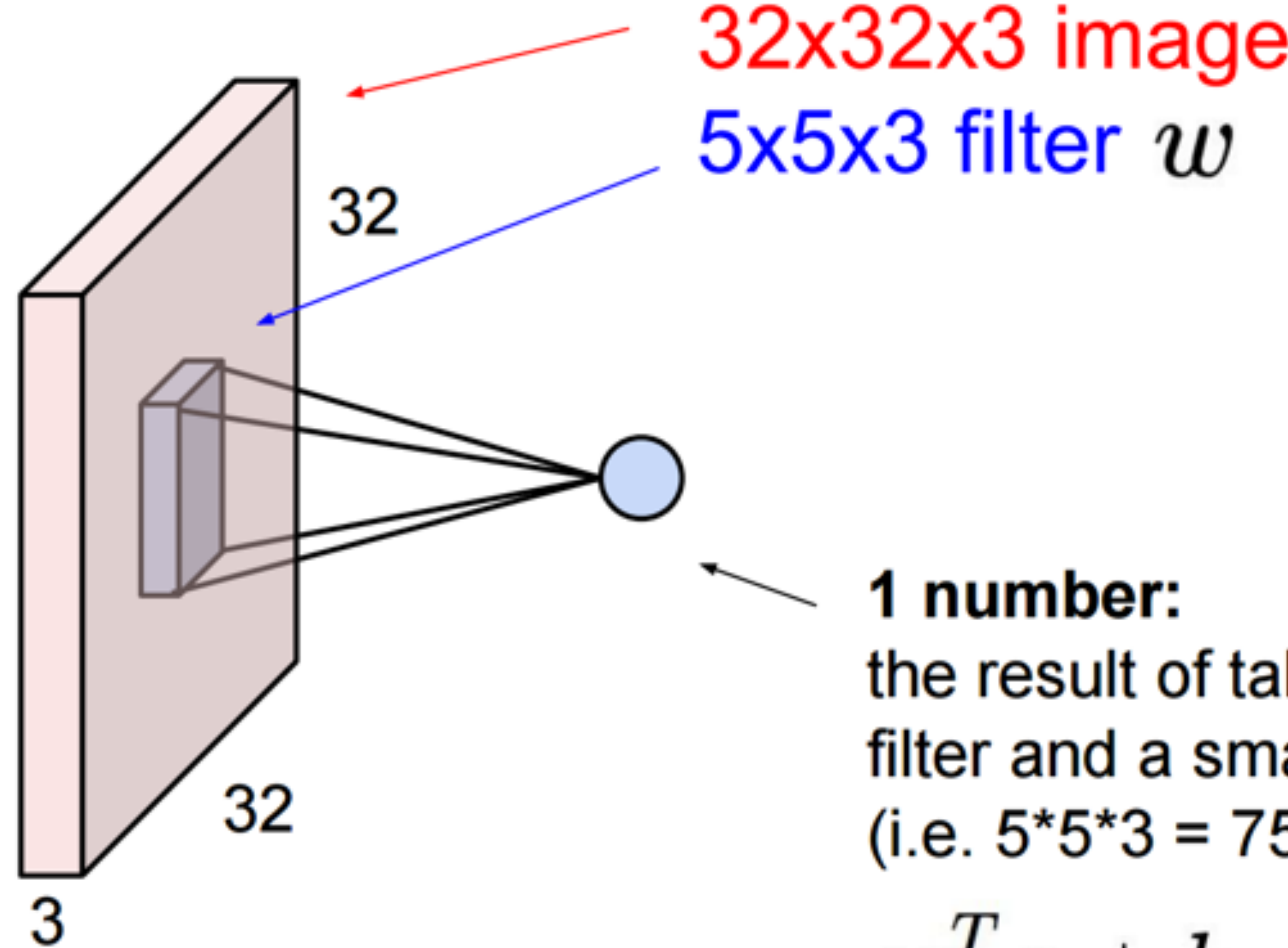
5x5x3 filter



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

(Recap)

Convolution Layer

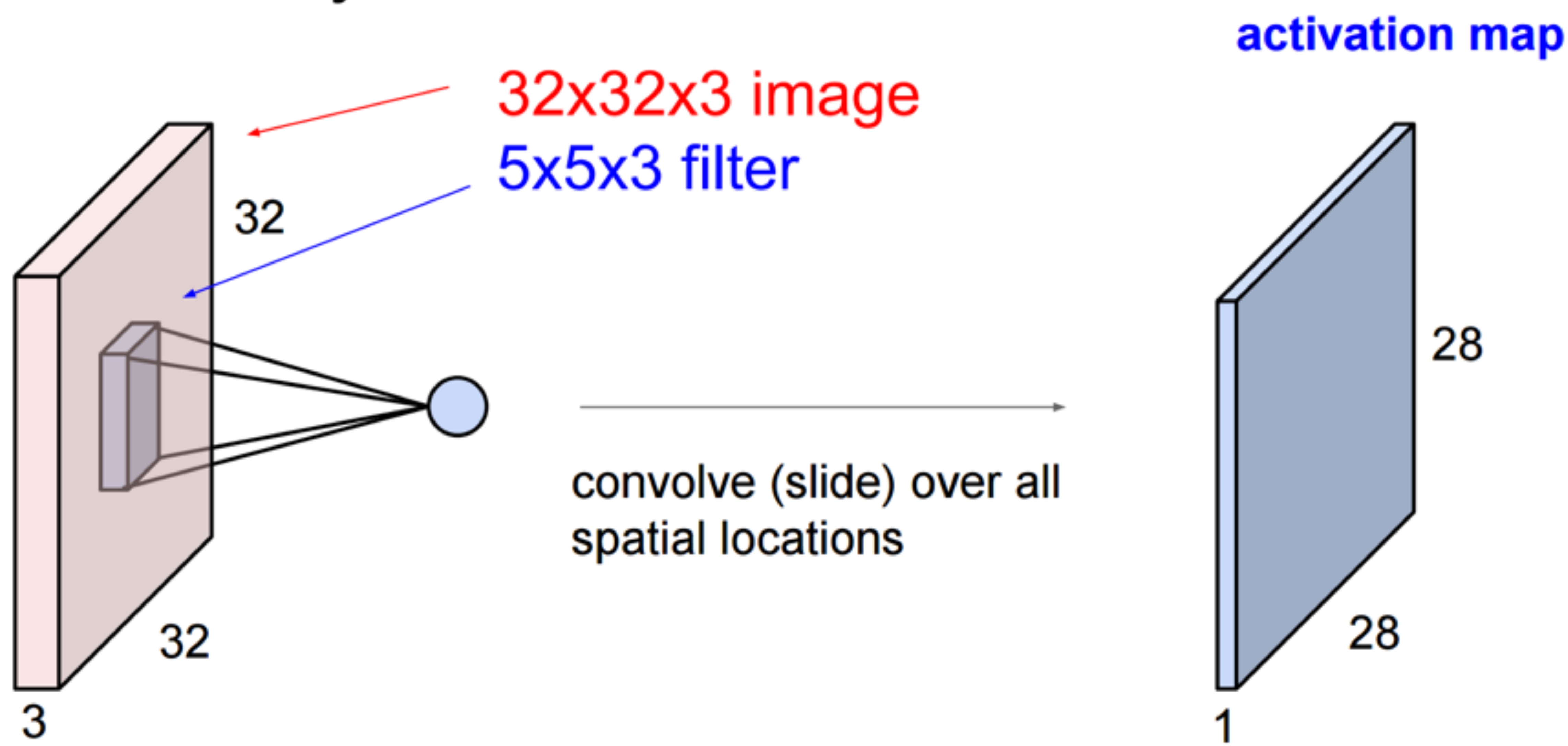


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

(Recap)

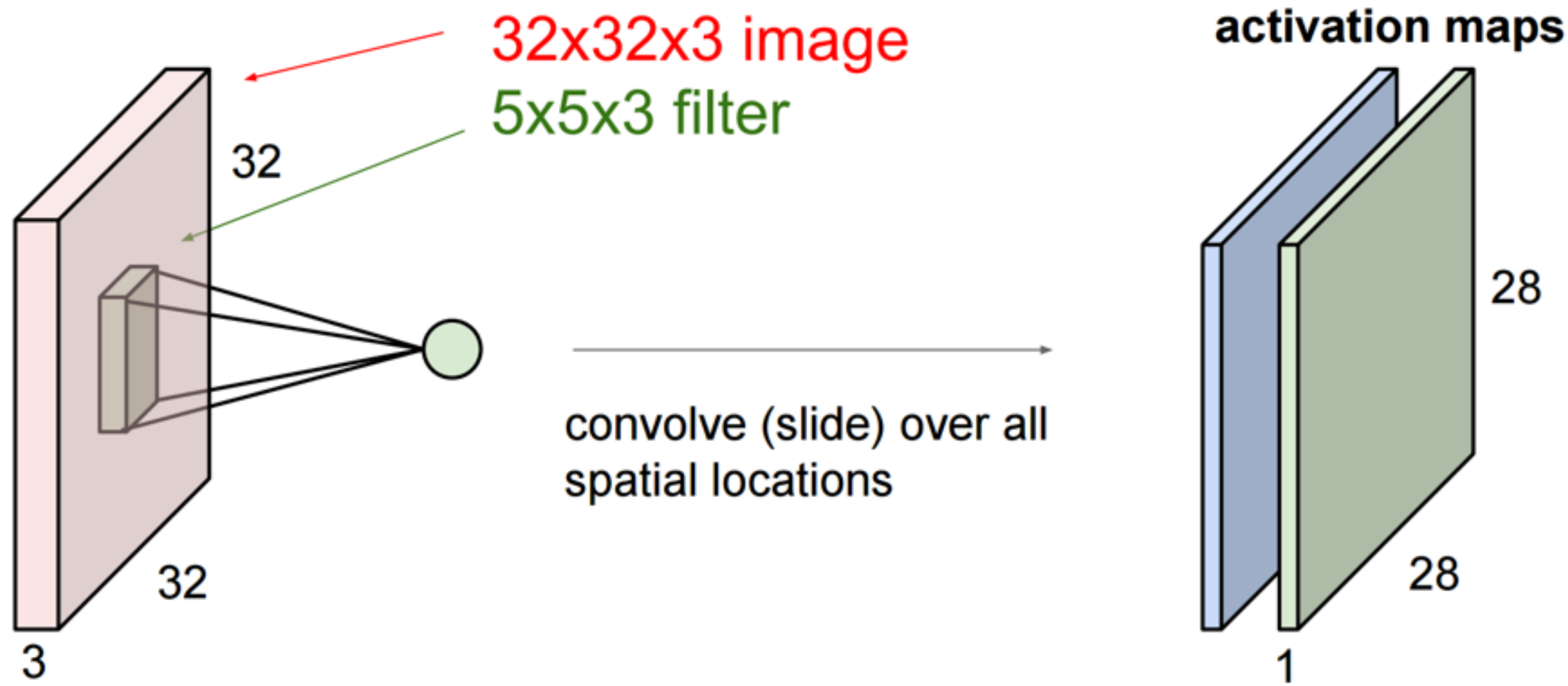
Convolution Layer



(Recap)

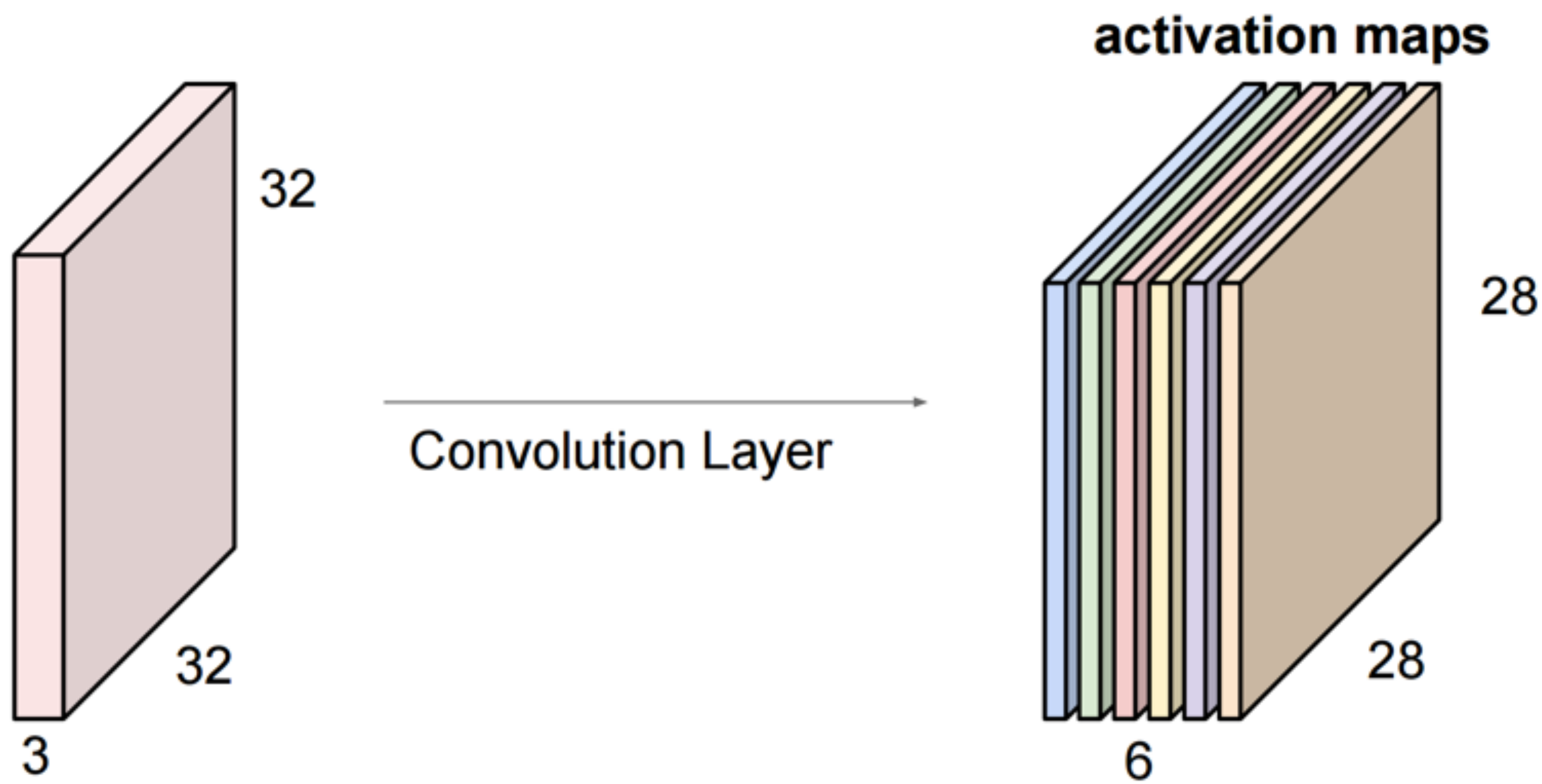
Convolution Layer

consider a second, green filter



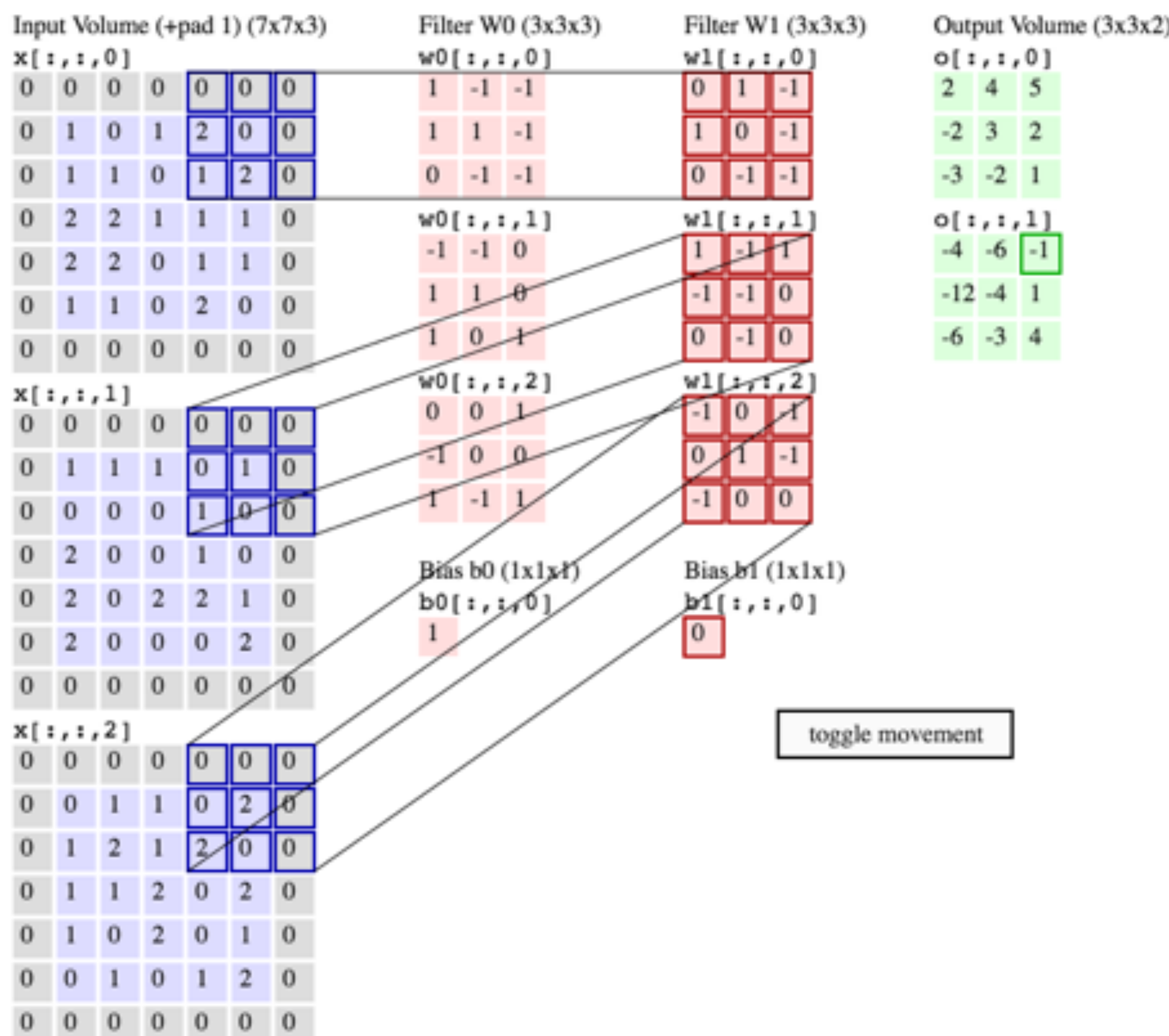
(Recap)

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!

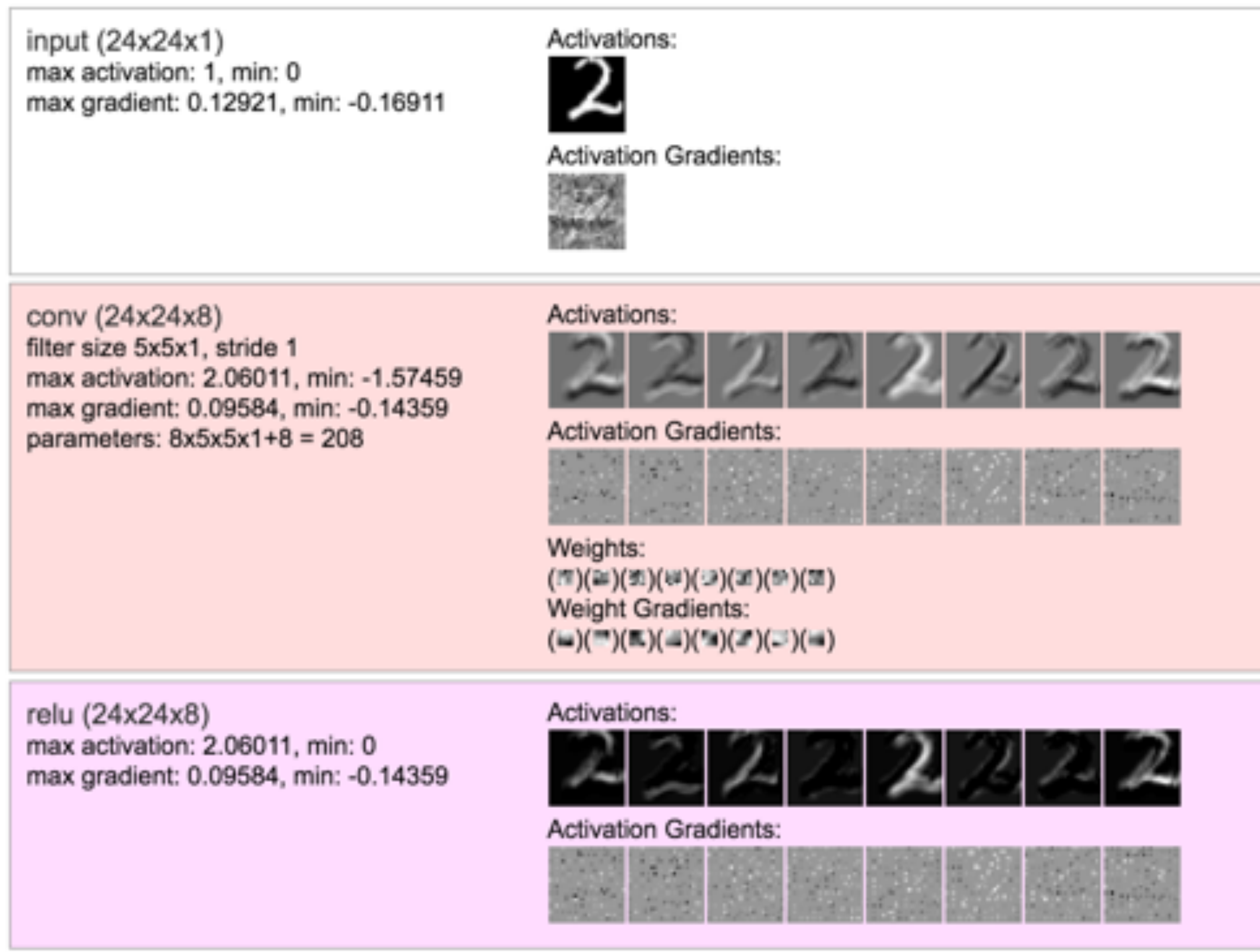
Web demo 1: Convolution



http://cs231n.github.io/convolutional-networks/

[Karpathy 2016]

Web demo 2: ConvNet in a Browser

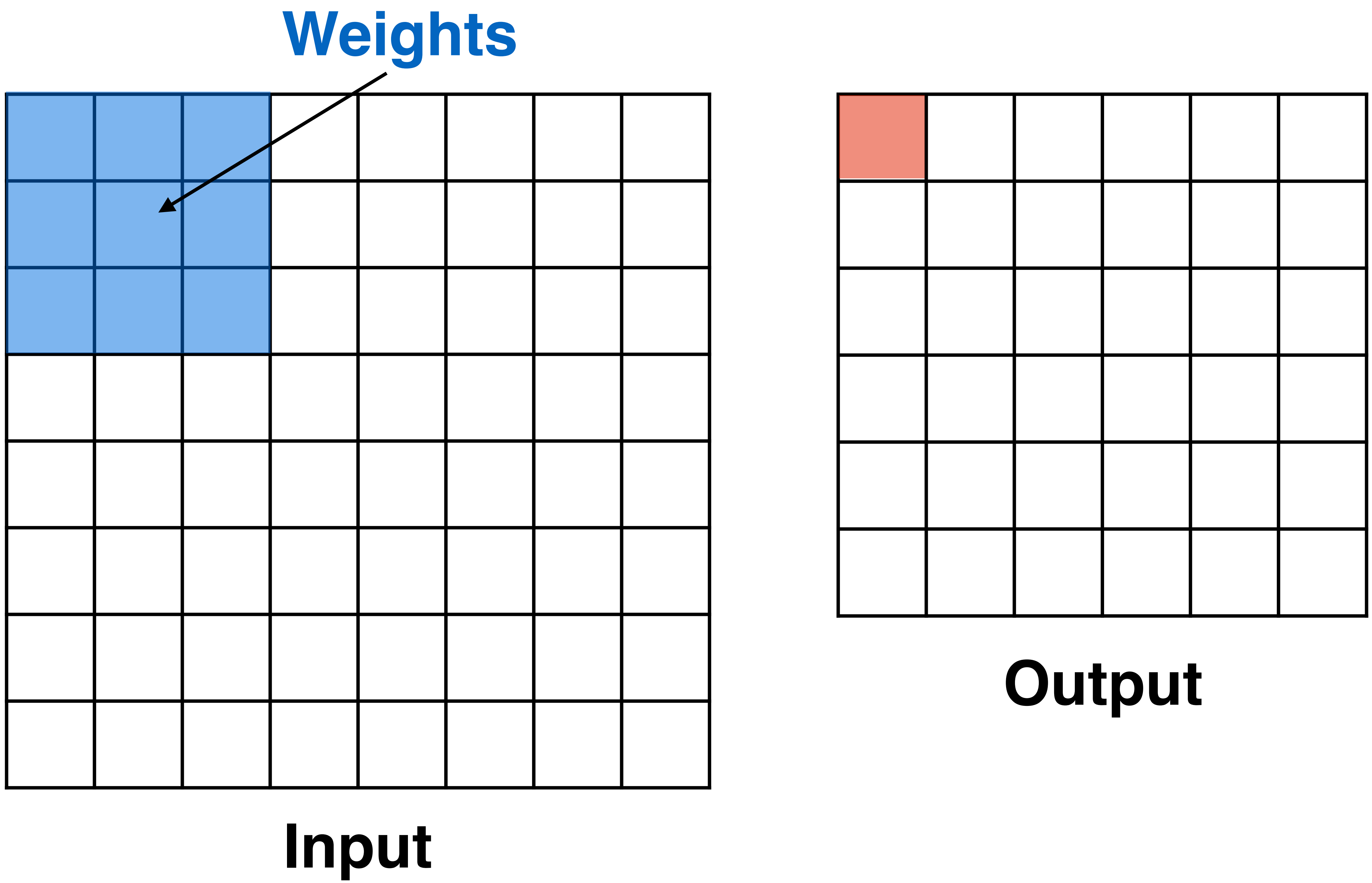


http://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

[Karpathy 2014]

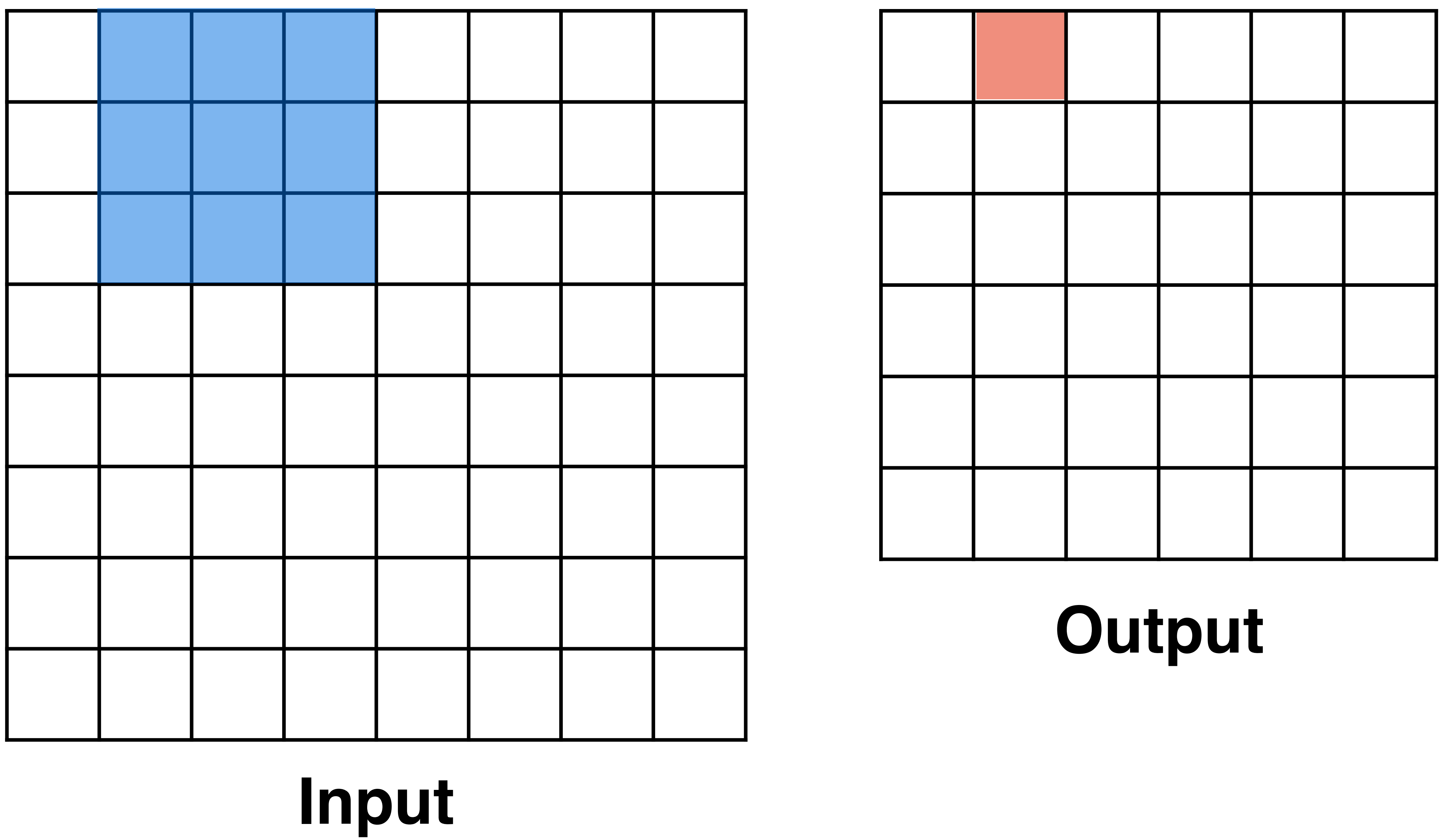
Convolution: Stride

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



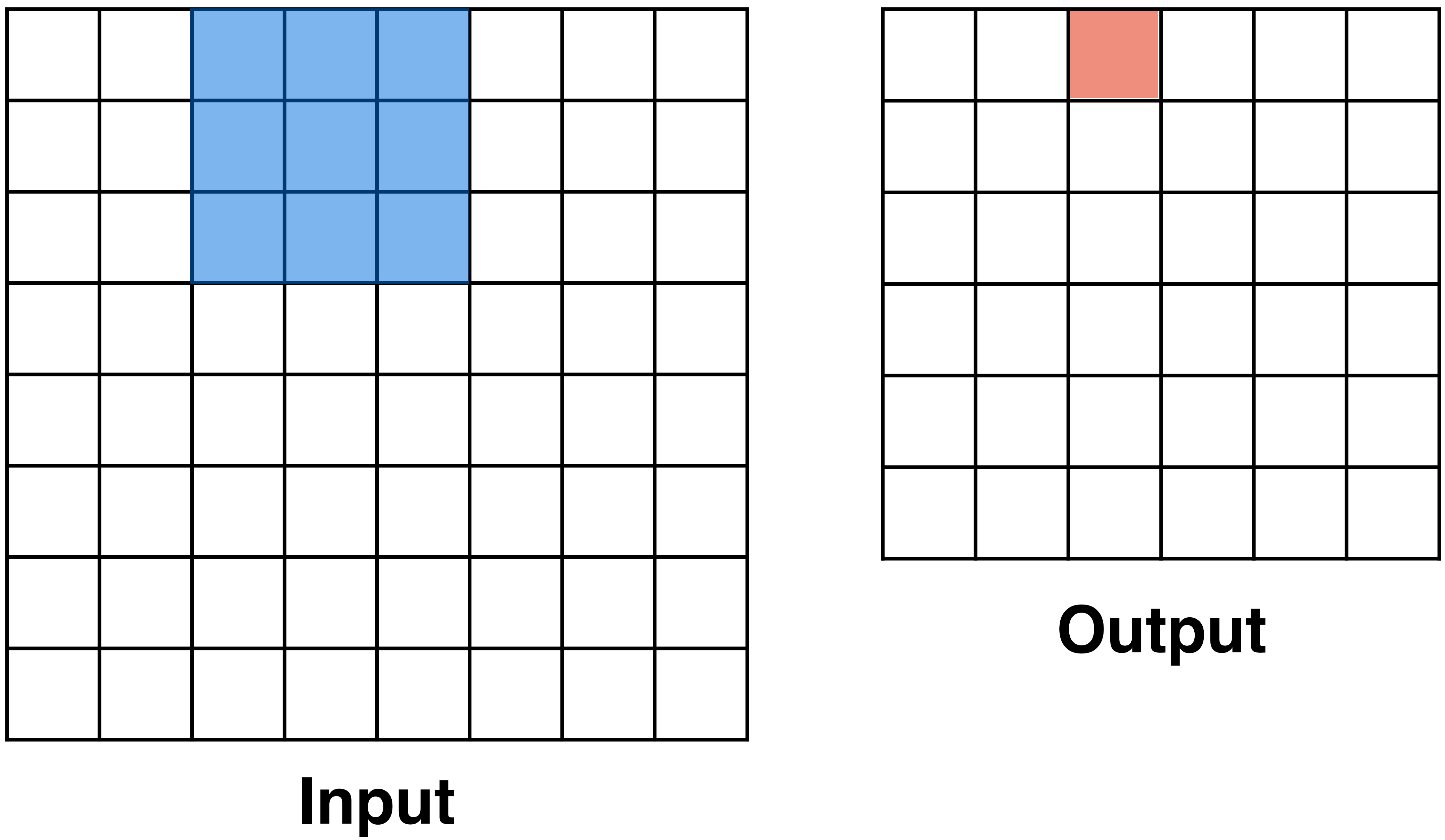
Convolution: Stride

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



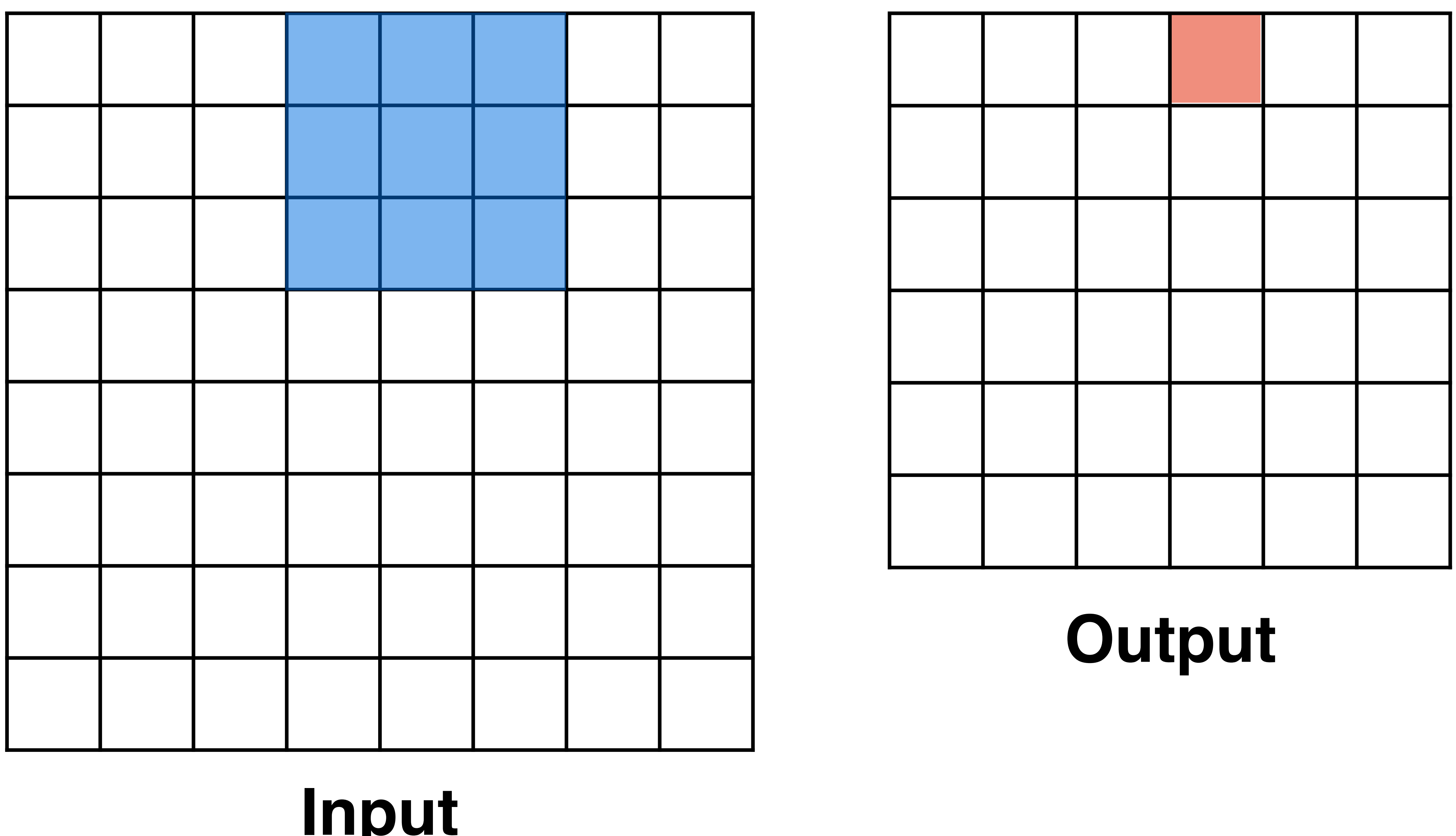
Convolution: Stride

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



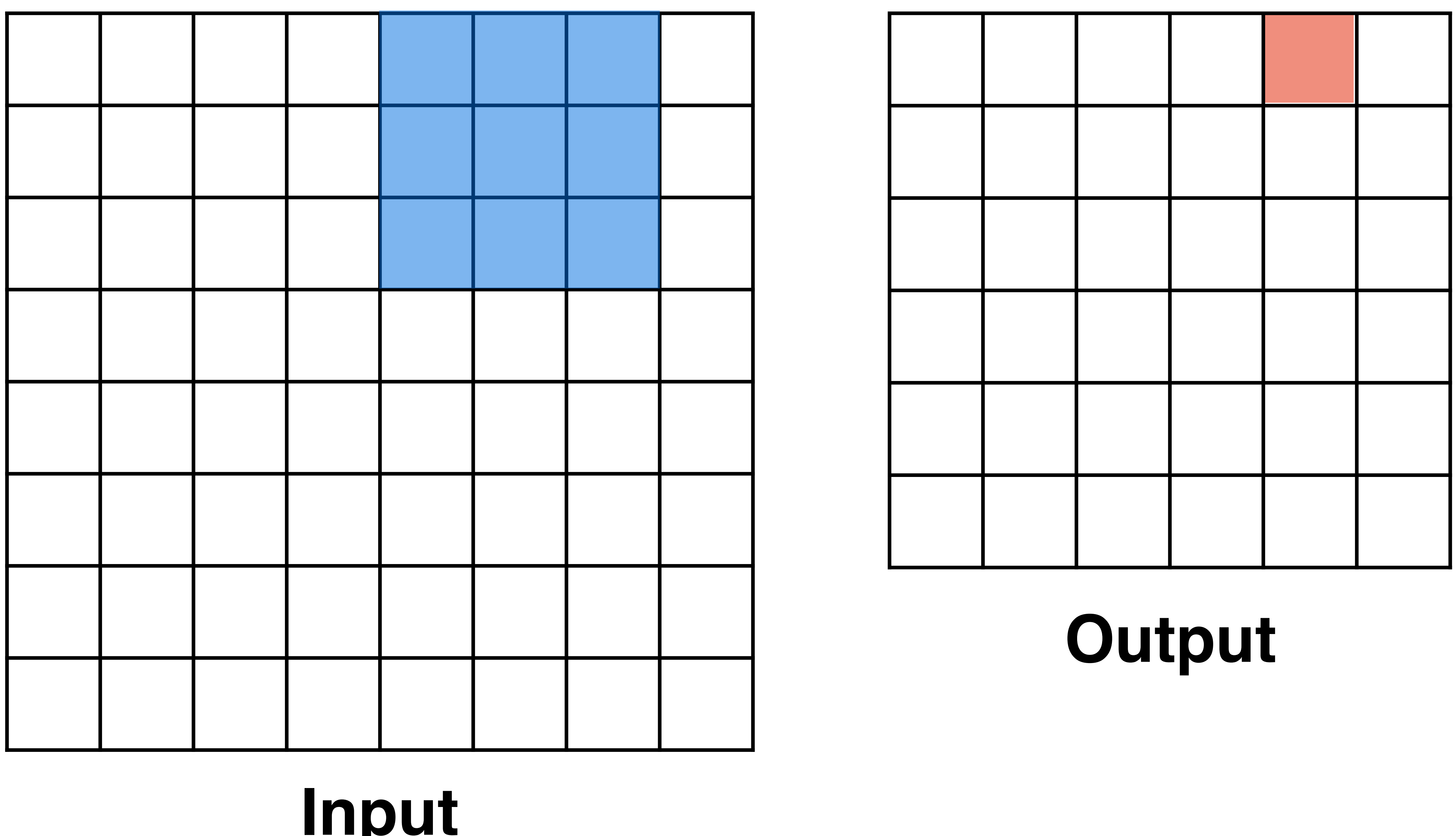
Convolution: Stride

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



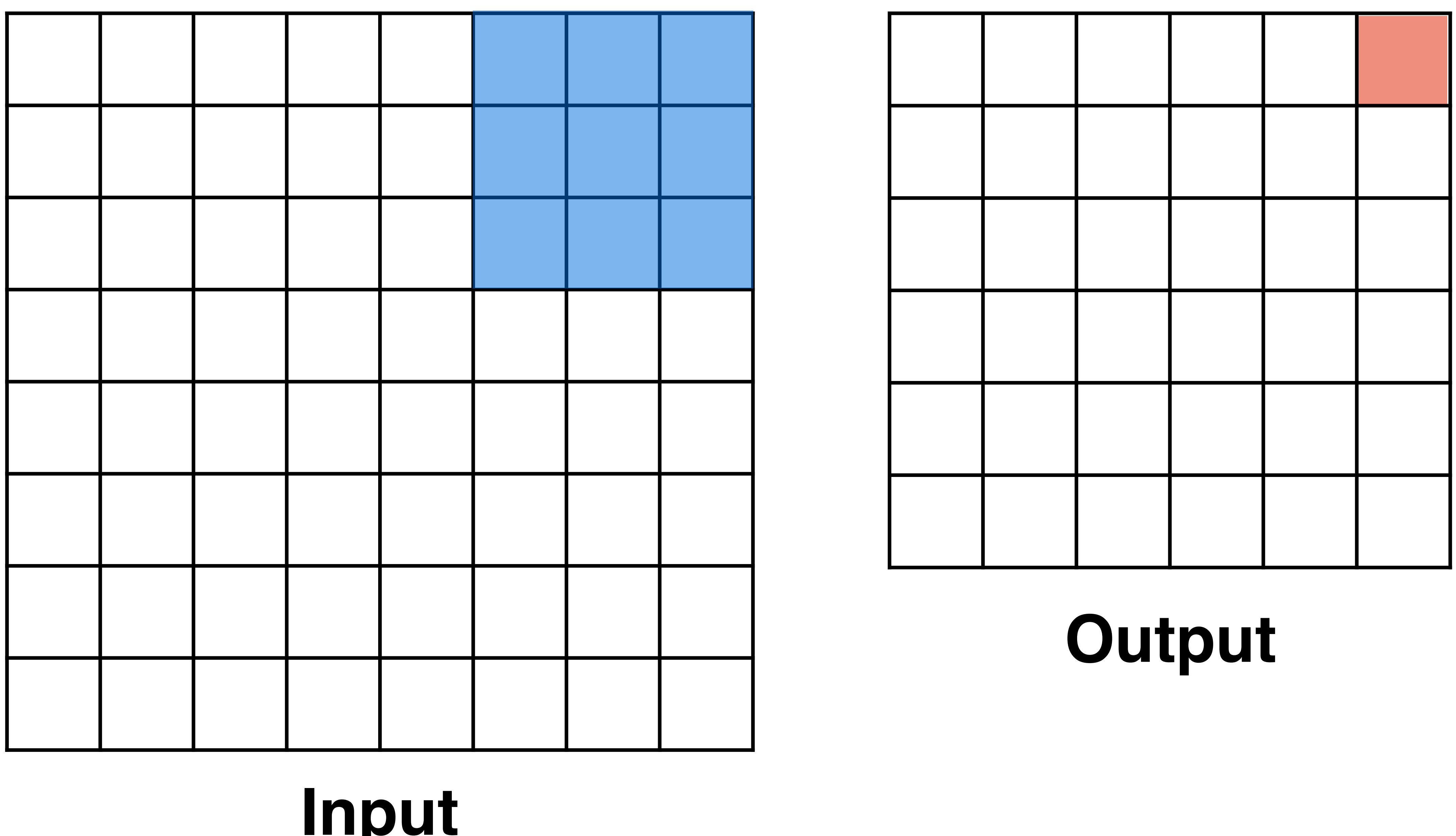
Convolution: Stride

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



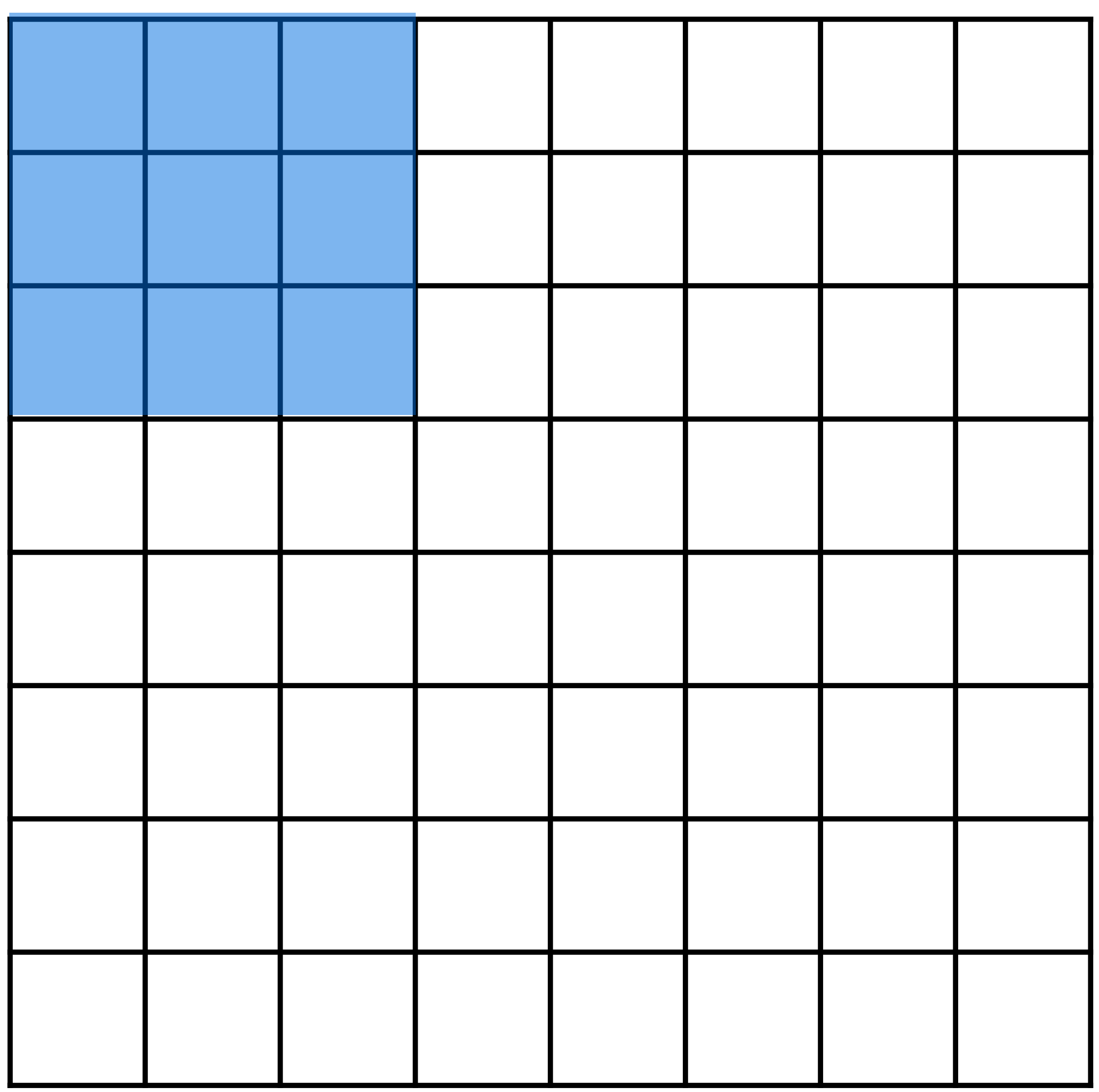
Convolution: Stride

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



Convolution: Stride

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



Input

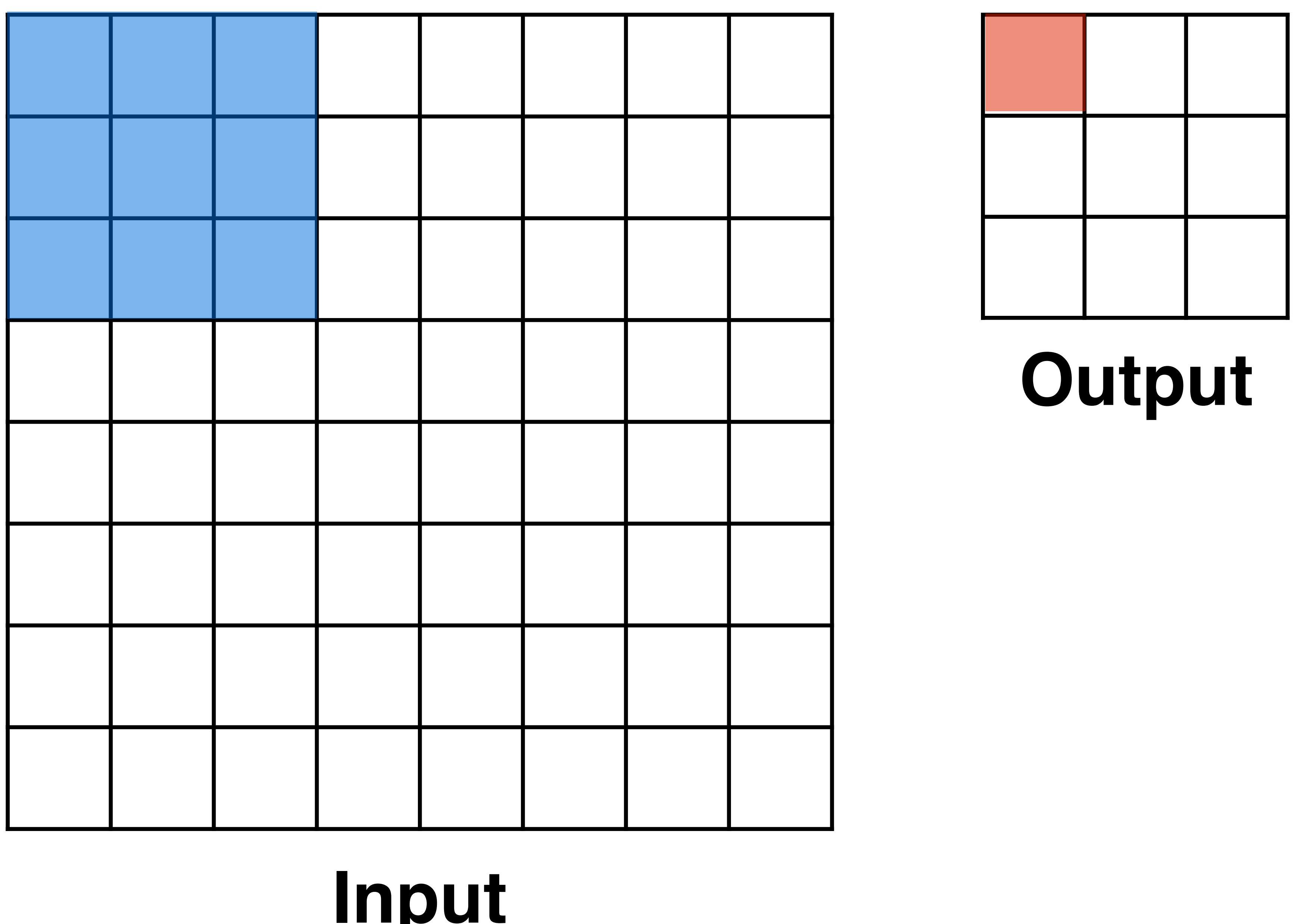
Recall that at each position, we are doing a **3D** sum:

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

(channel, row, column)

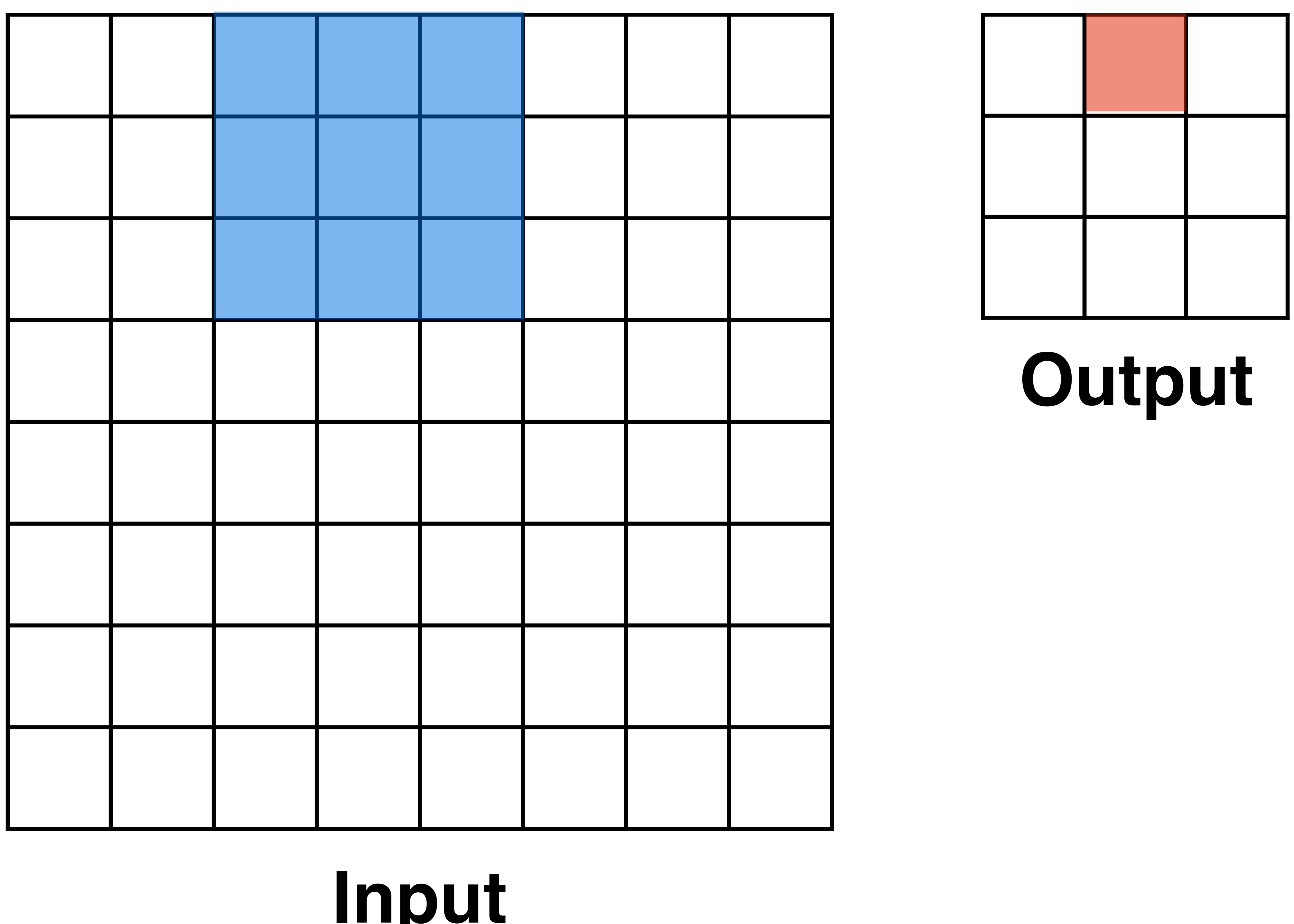
Convolution: Stride

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



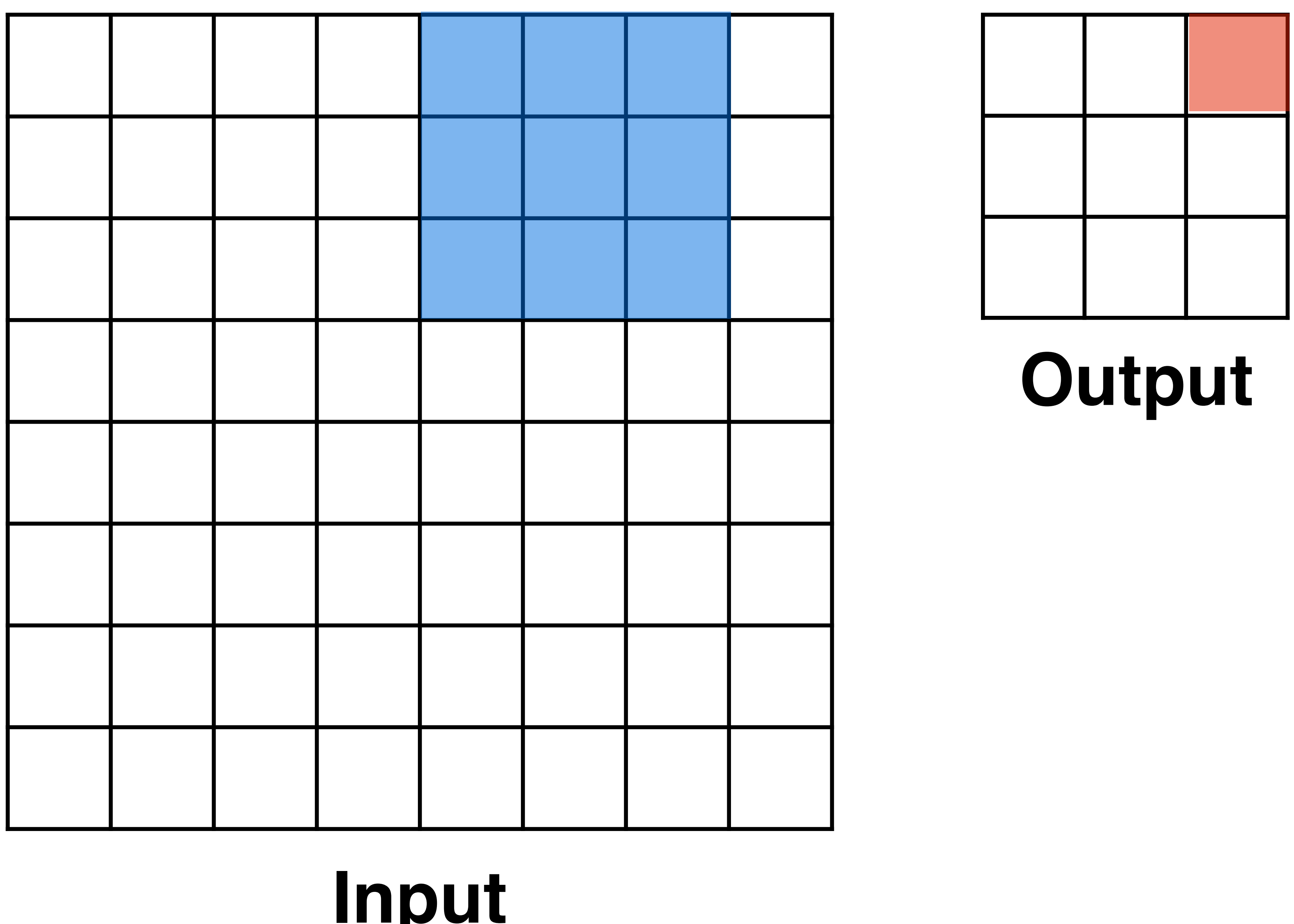
Convolution: Stride

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



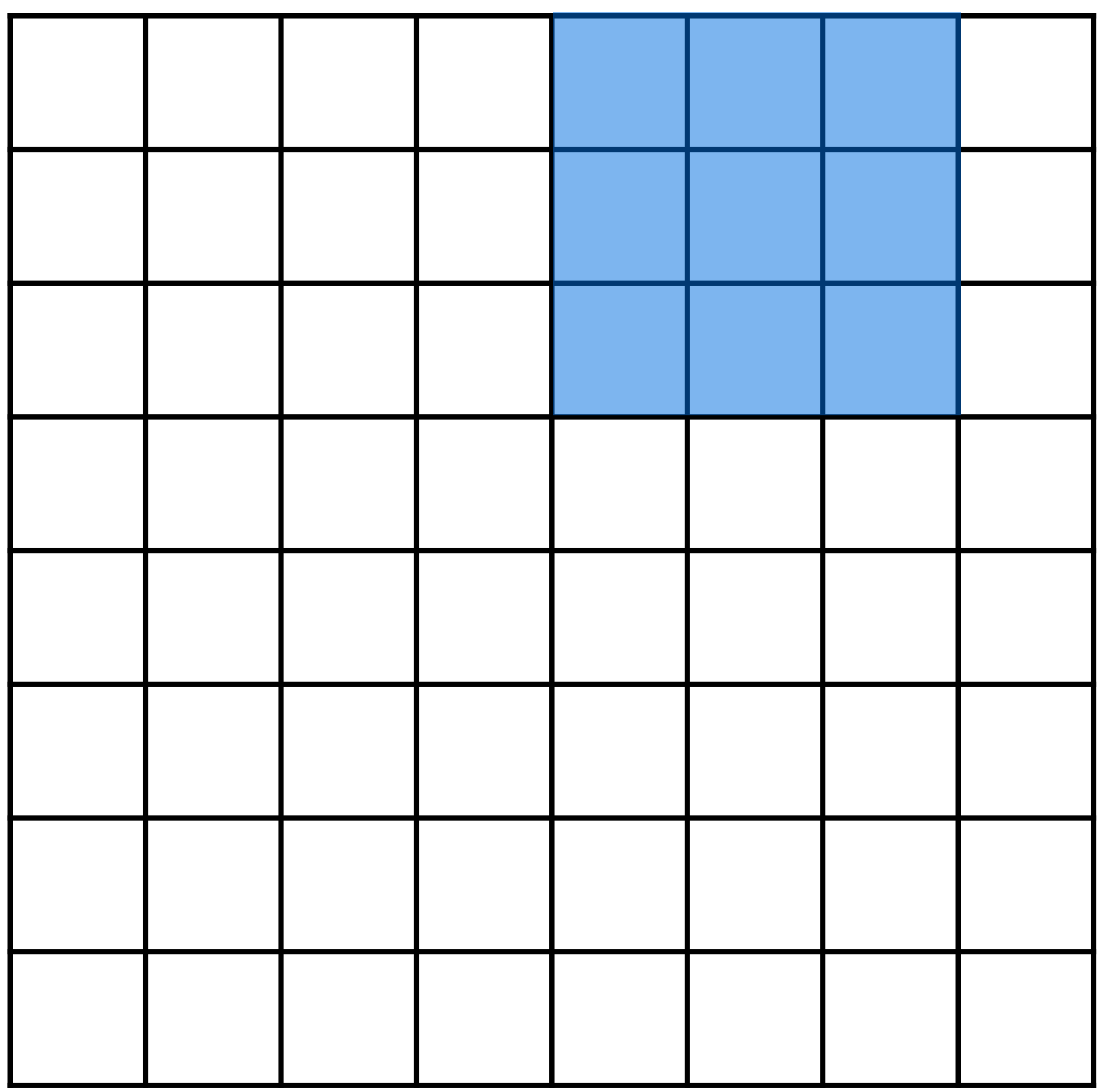
Convolution: Stride

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

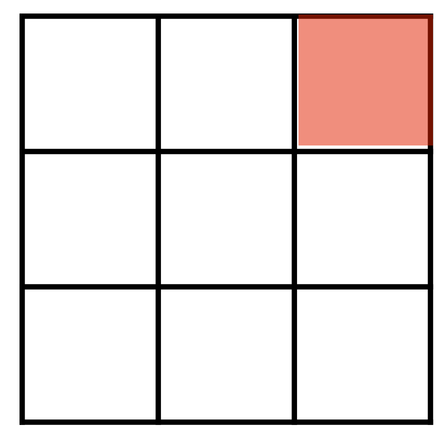


Convolution: Stride

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



Input



Output

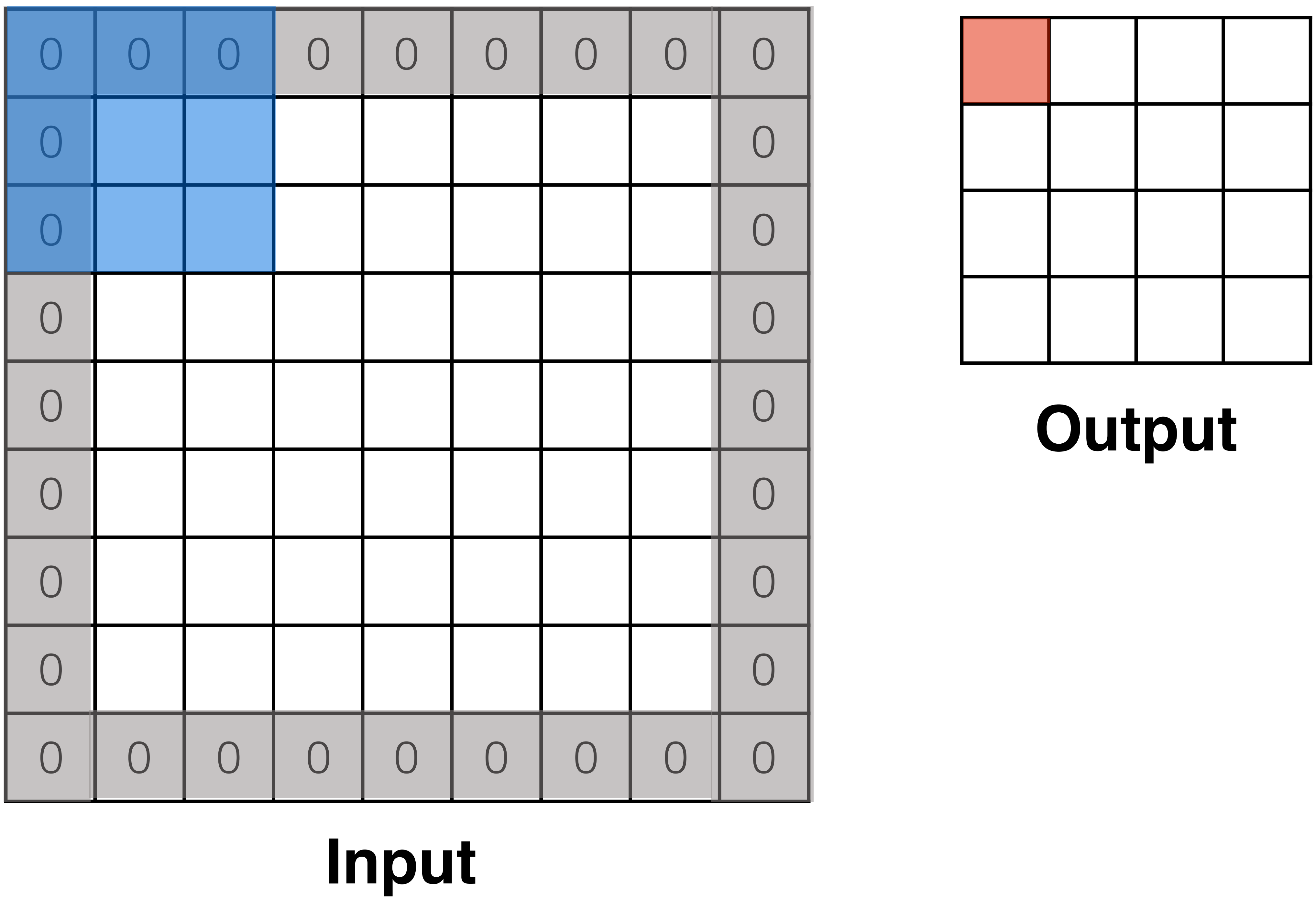
- Notice that with certain strides, we may not be able to cover all of the input

- The output is also half the size of the input

Convolution: Padding

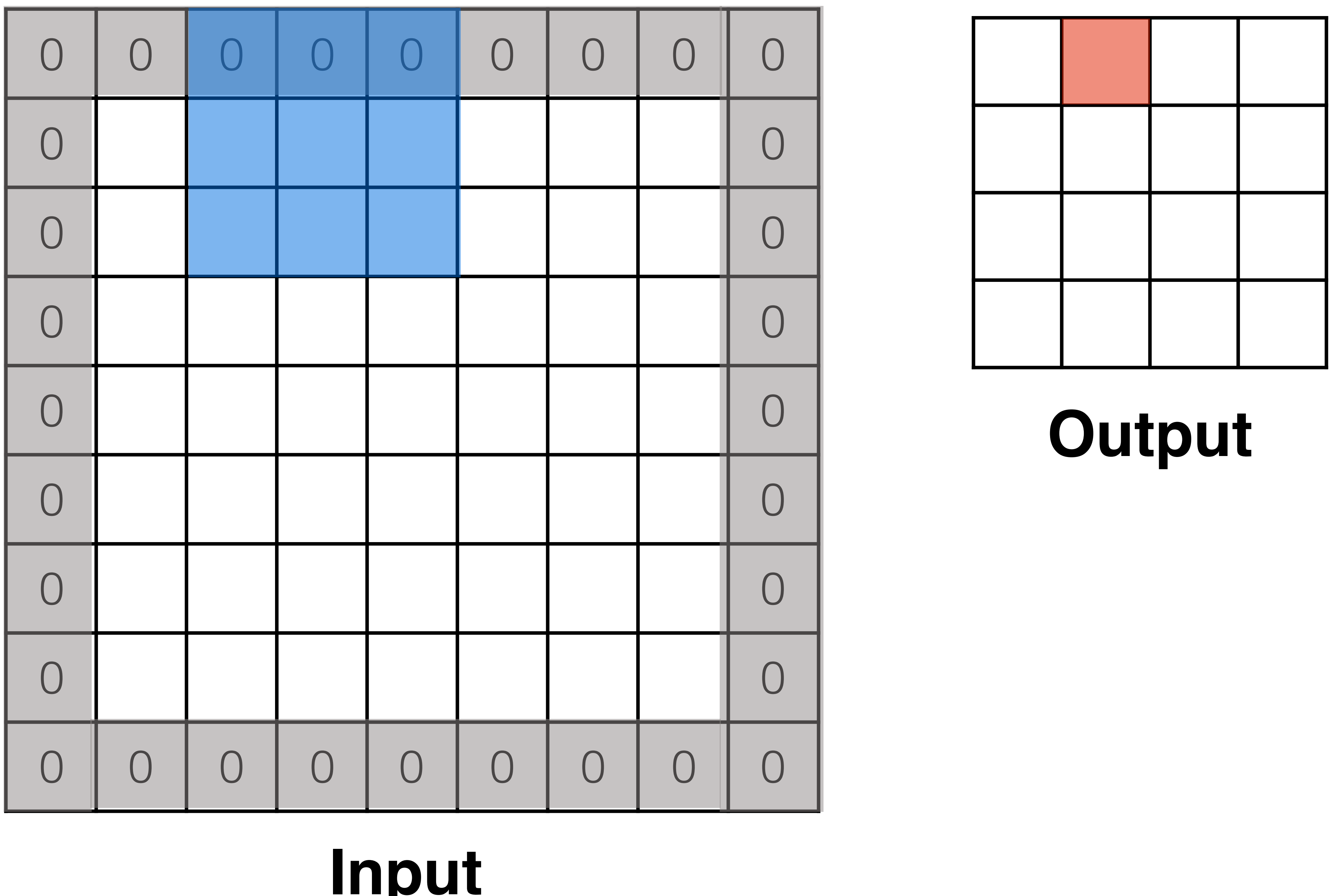
We can also pad the input with zeros.

Here, **pad = 1, stride = 2**



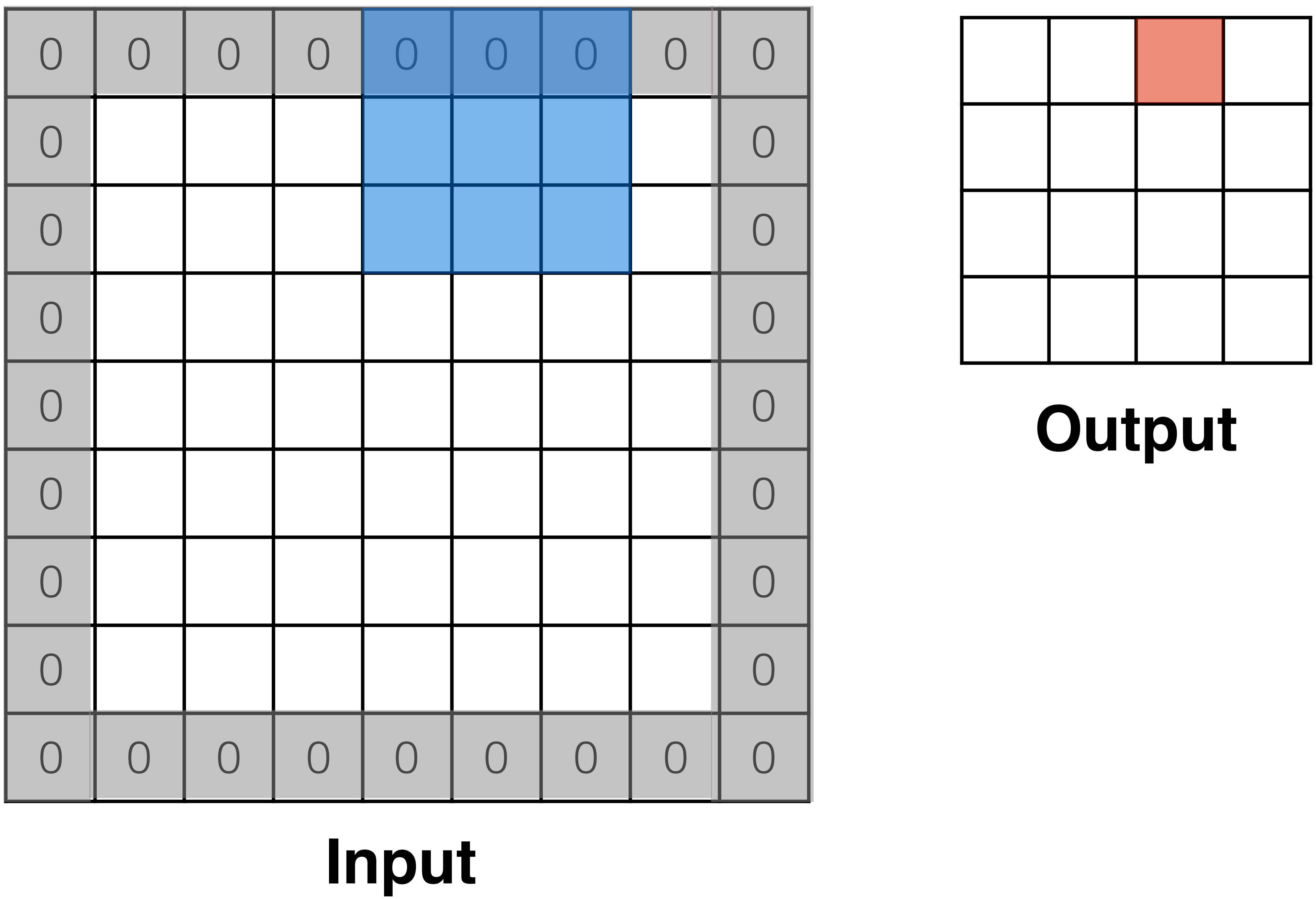
Convolution: Padding

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



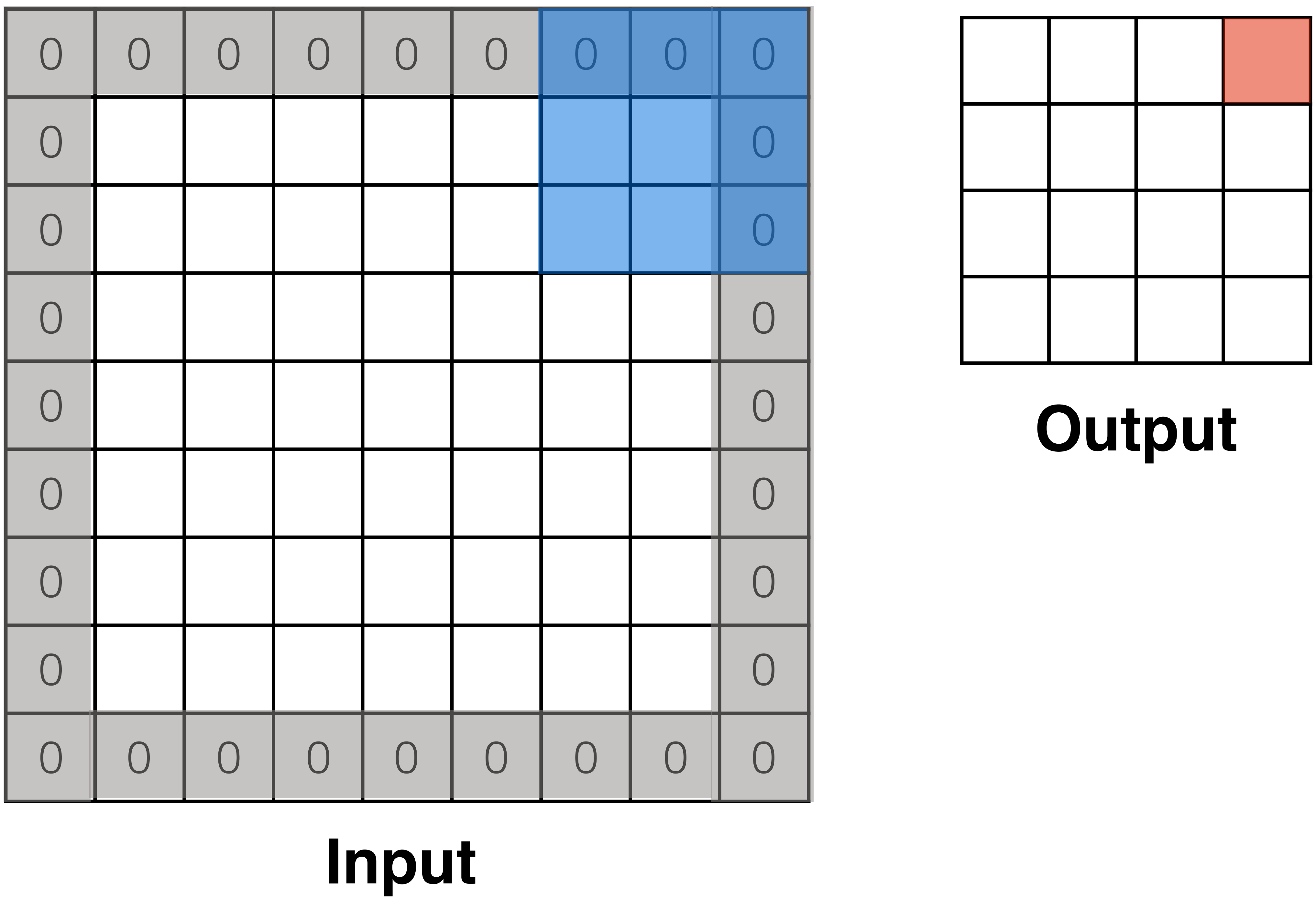
Convolution: Padding

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

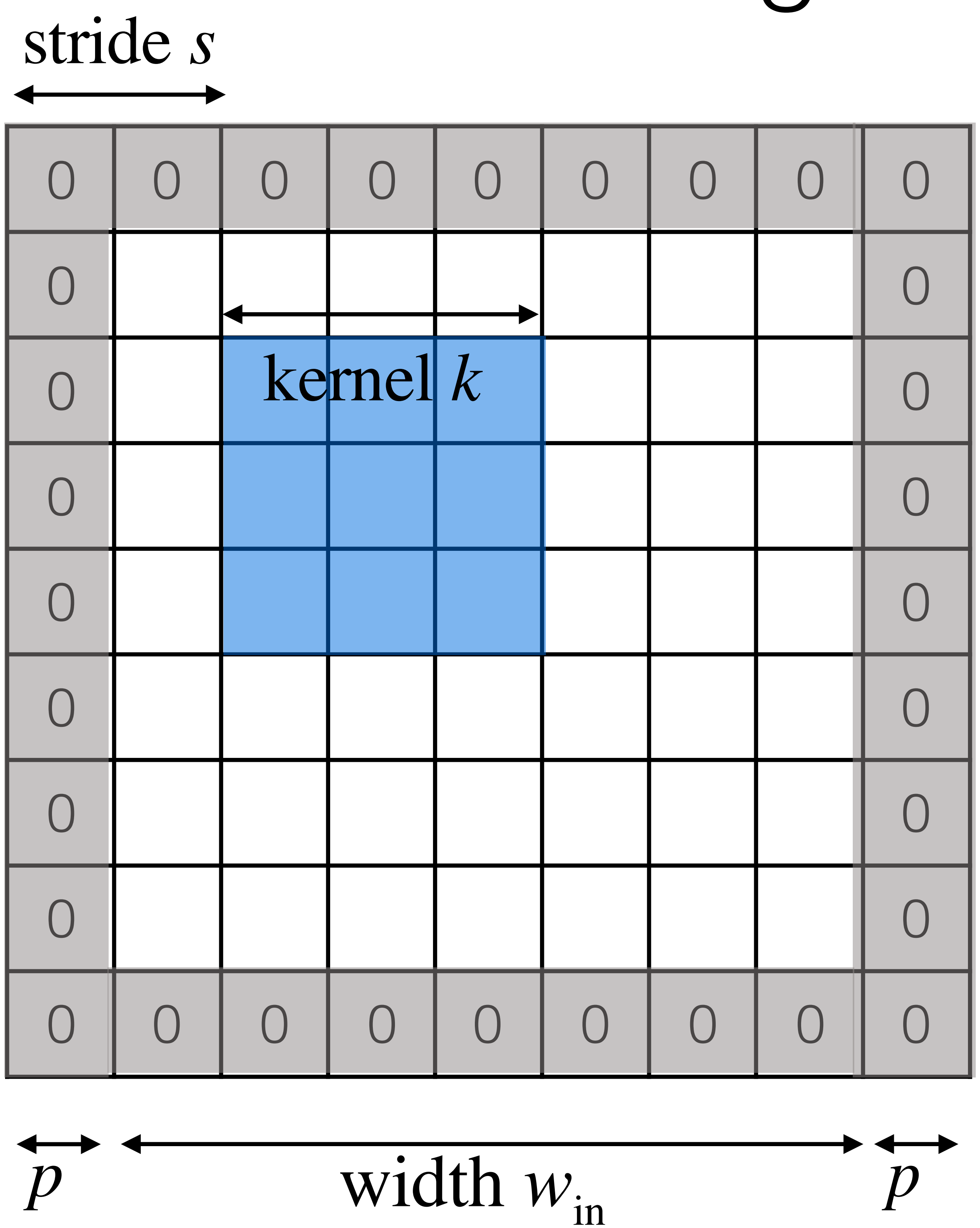


Convolution: Padding

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



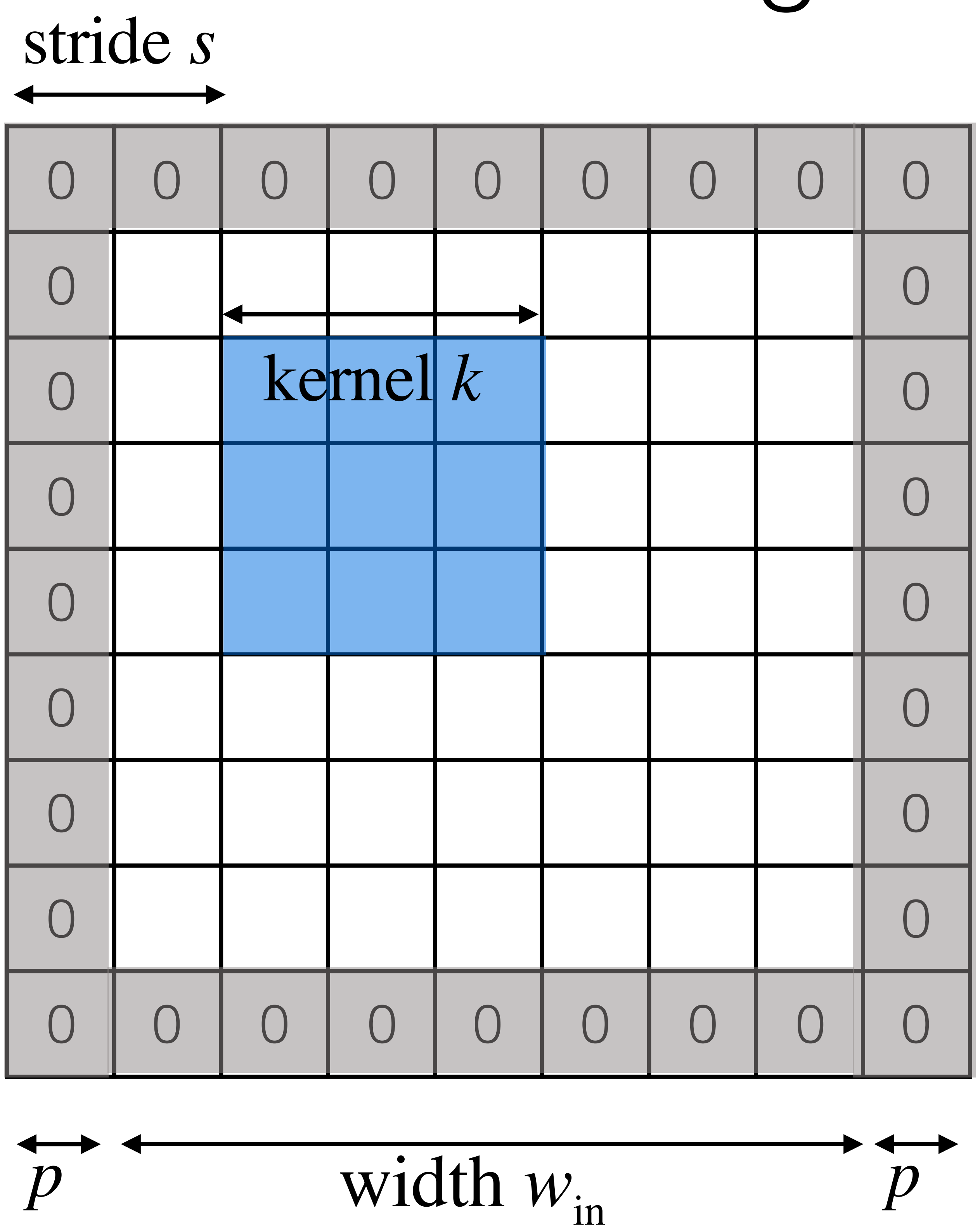
Convolution:
How big is the output?



In general, the output has size:

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

Convolution:
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?

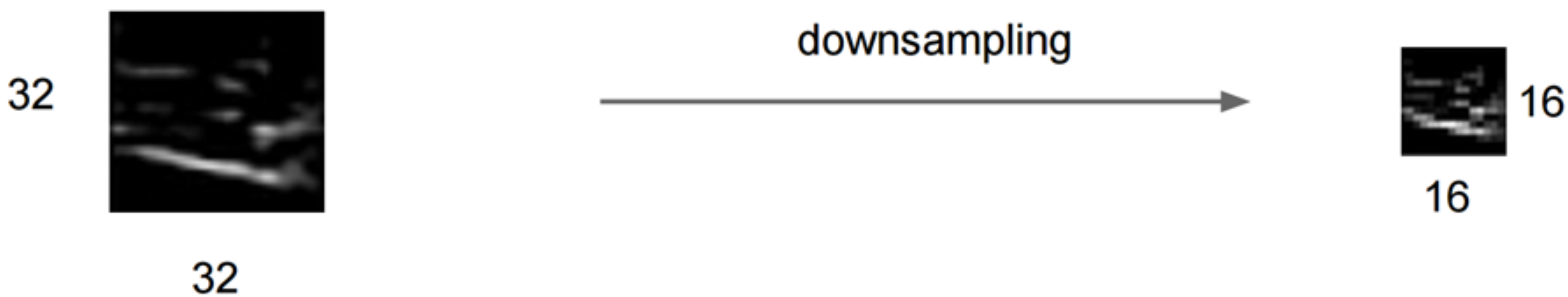
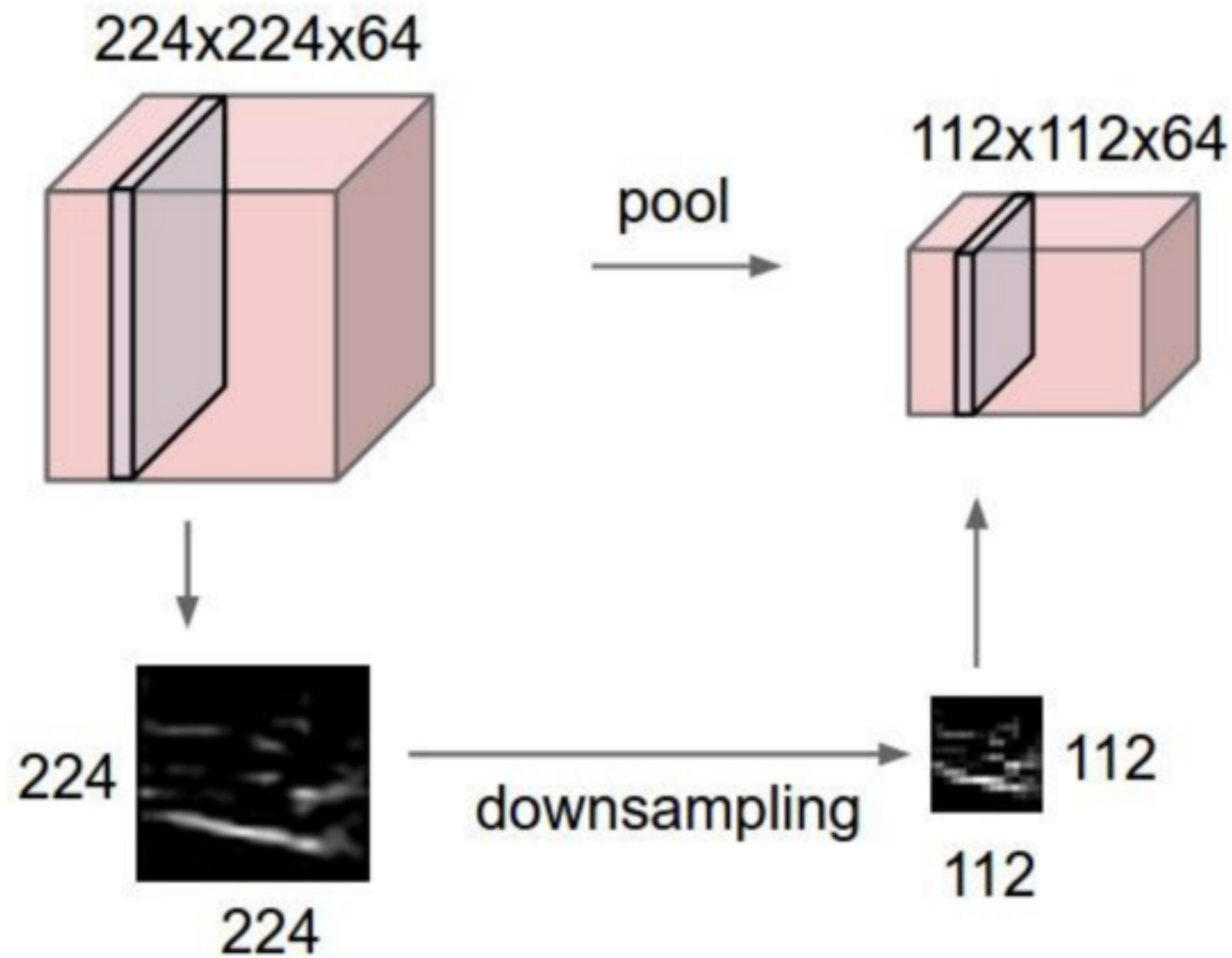


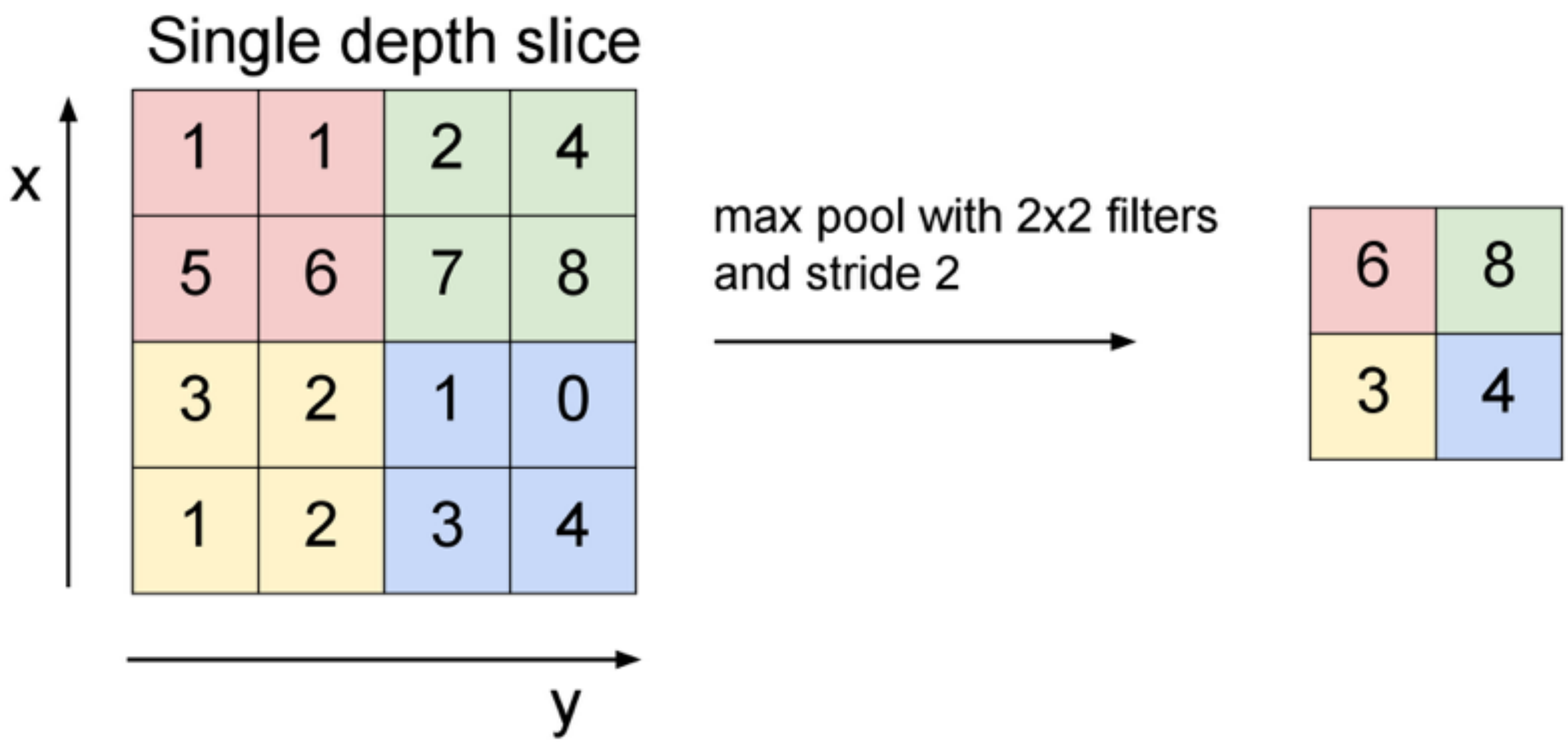
Figure: Andrej Karpathy

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

Figure: Andrej Karpathy

Example ConvNet

Figure: Andrej Karpathy

Example ConvNet

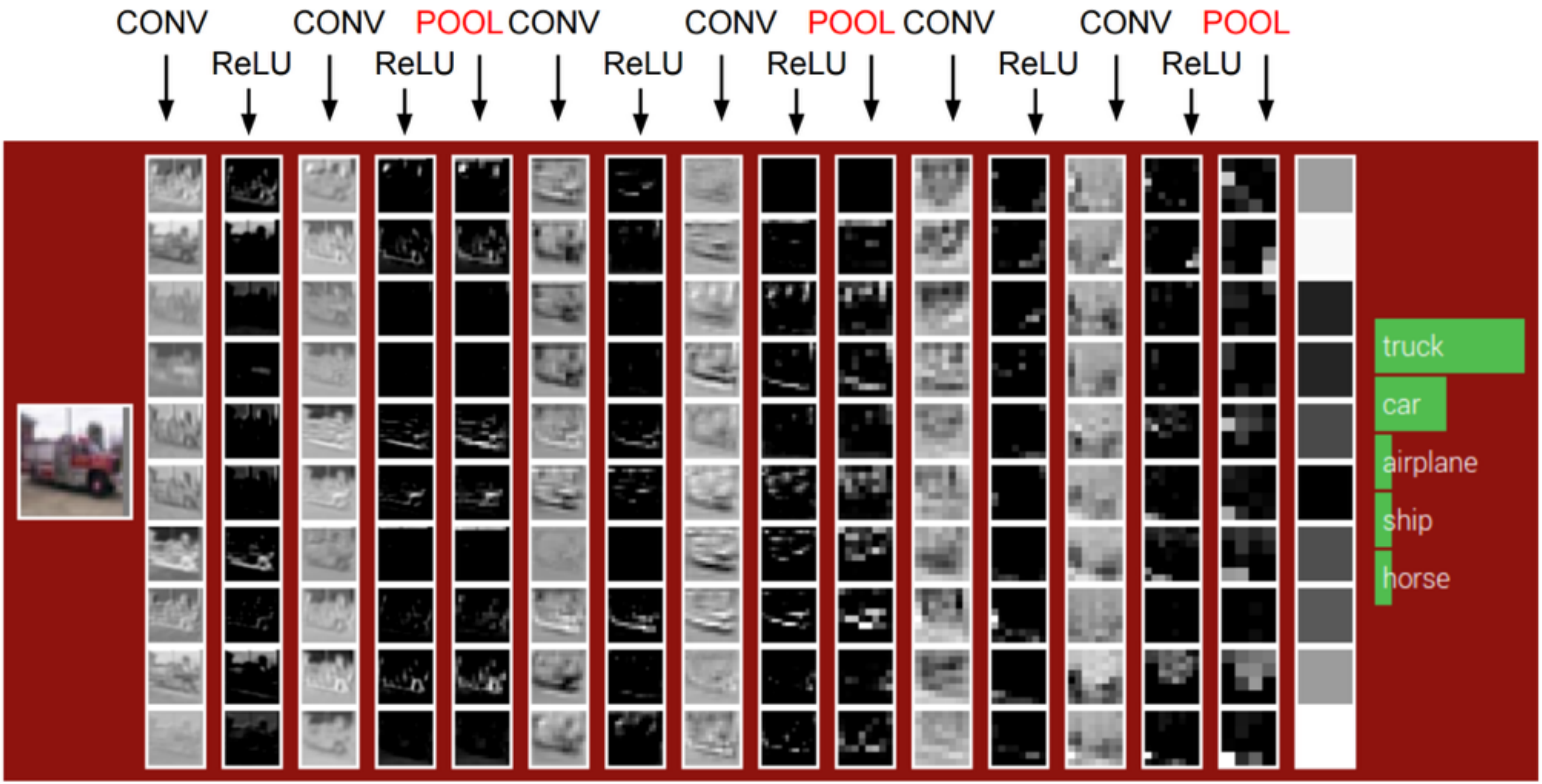


Figure: Andrej Karpathy

Example ConvNet

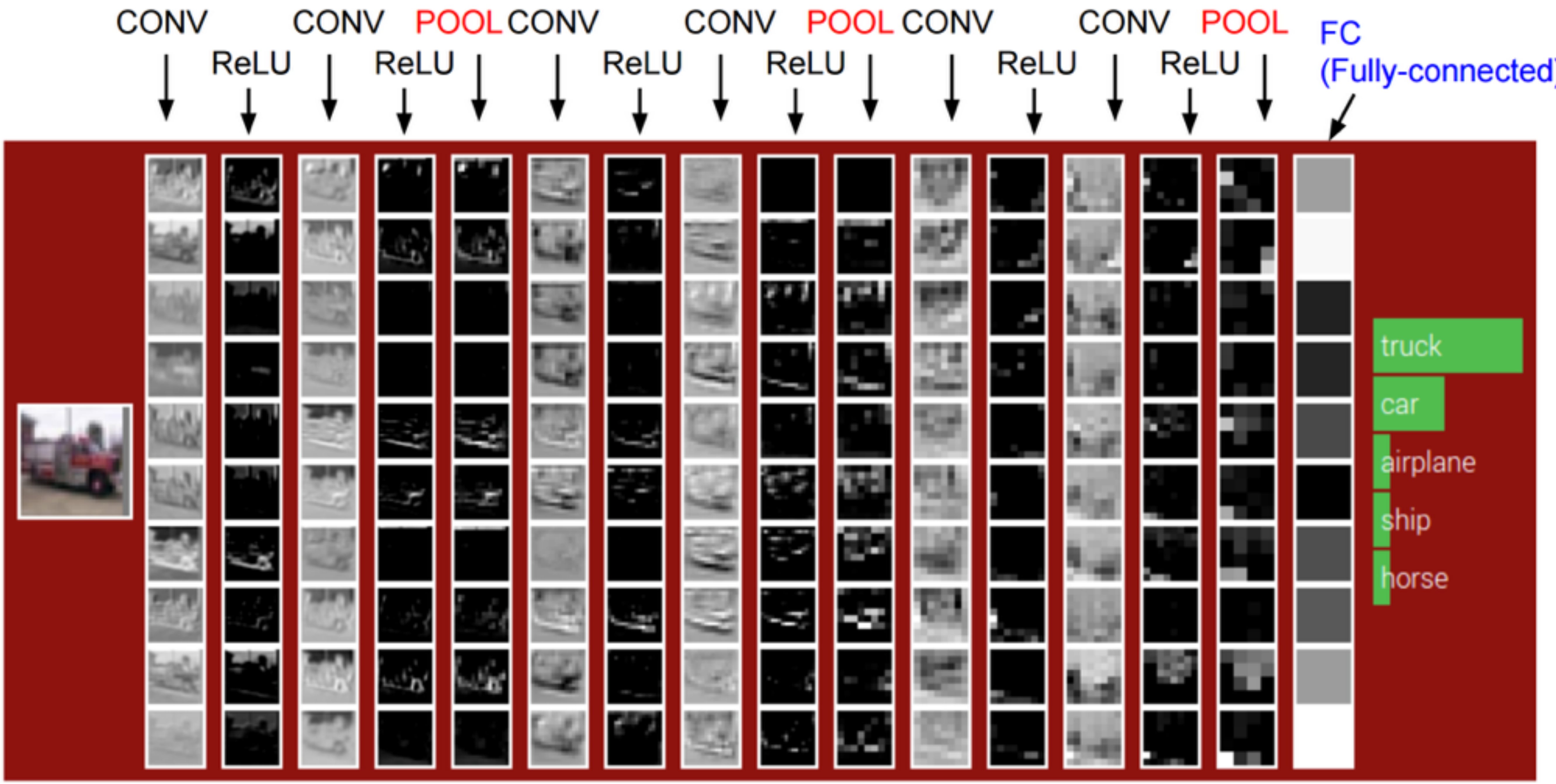
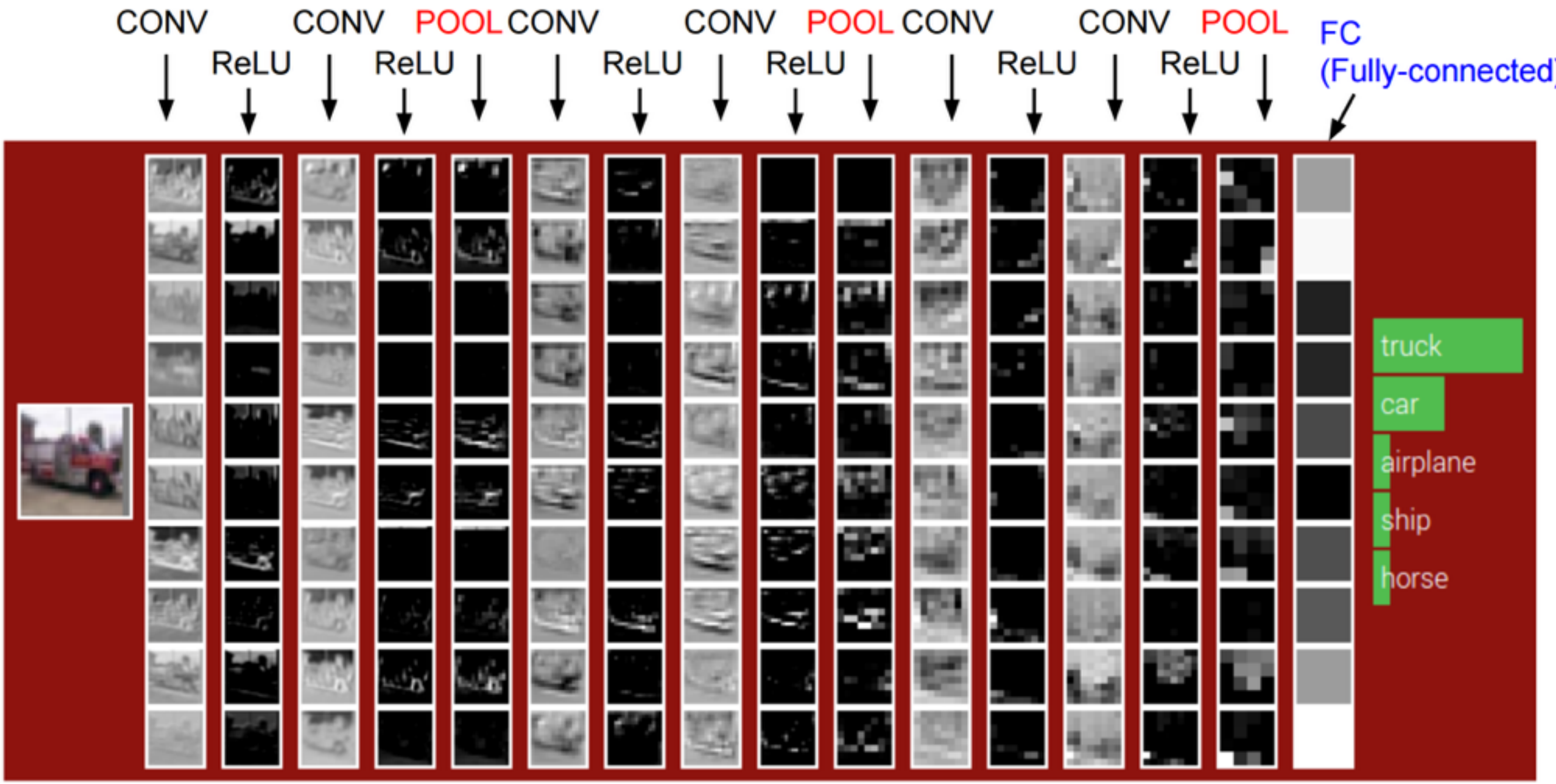


Figure: Andrej Karpathy

Example ConvNet

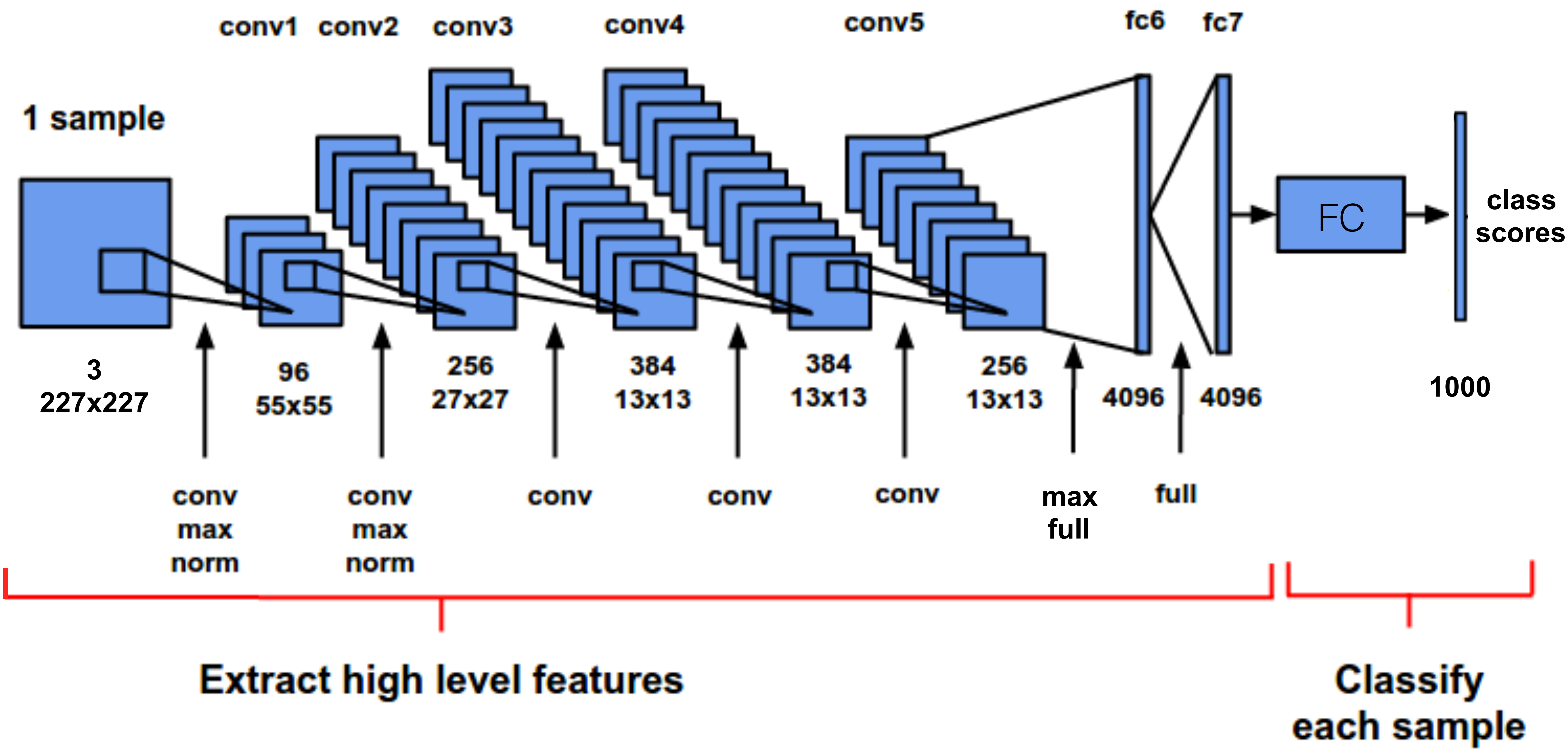


10x3x3 conv filters, stride 1, pad 1

2x2 pool filters, stride 2

Figure: Andrej Karpathy

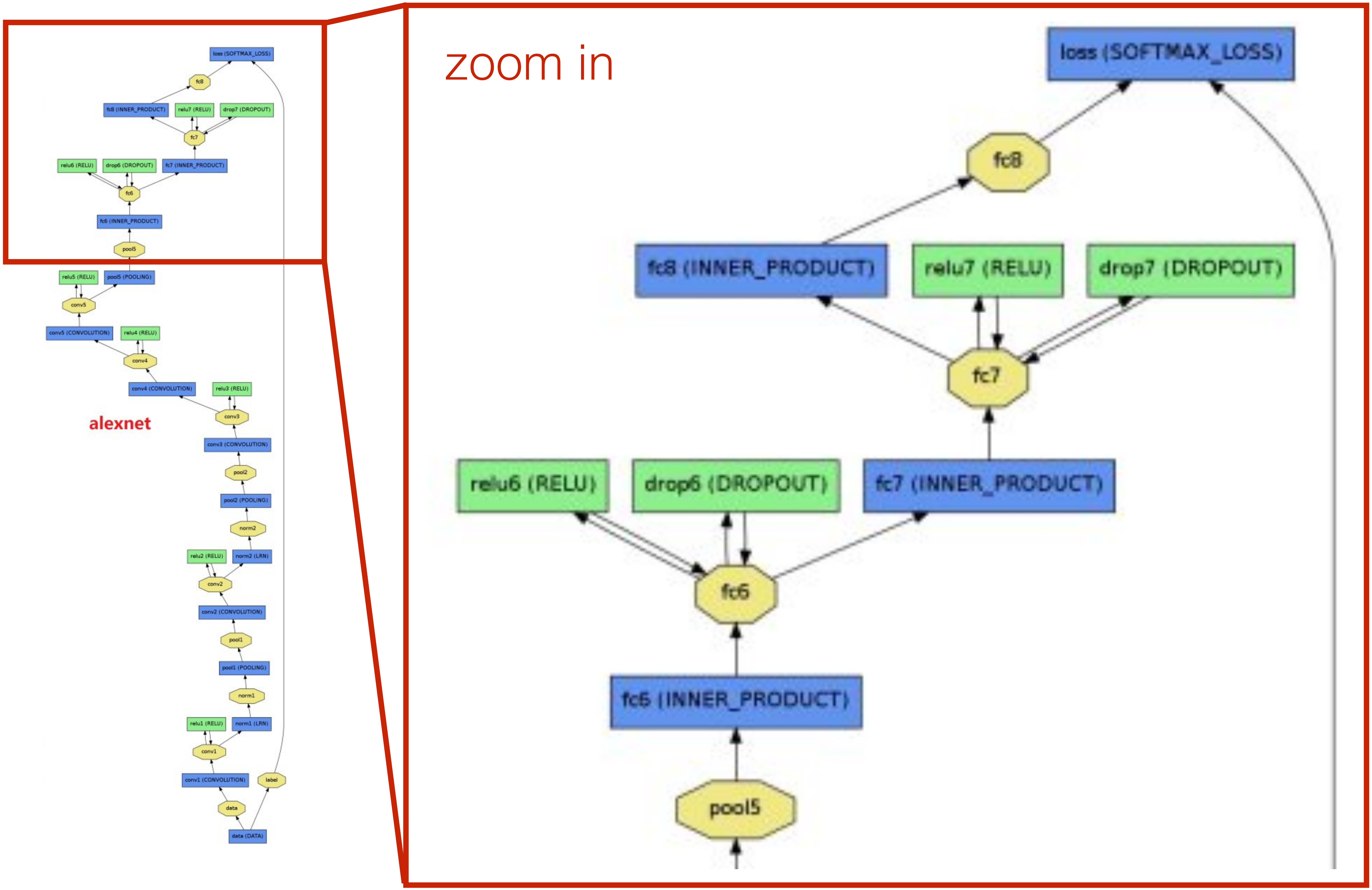
Example: AlexNet [Krizhevsky 2012]



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

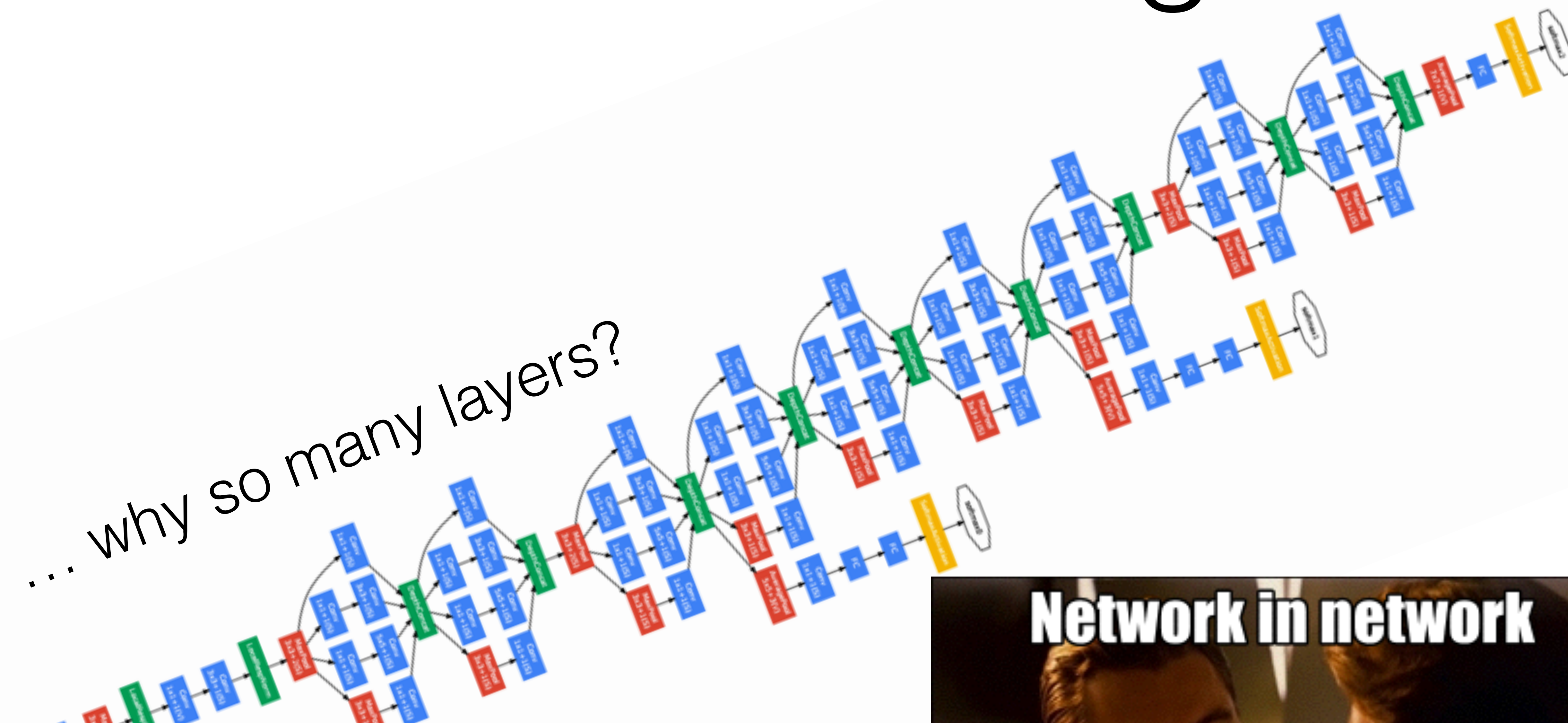
Figure: [Karnowski 2015] *(with corrections)*

Example: AlexNet [Krizhevsky 2012]



Questions?

How do you actually train these things?



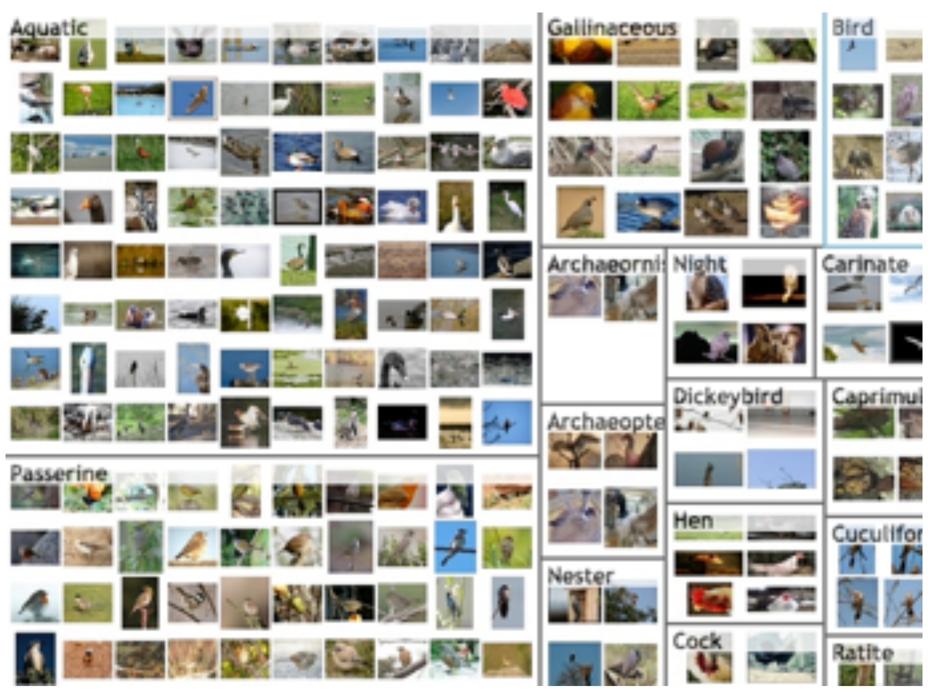
… why so many layers?

[Szegedy et al, 2014]

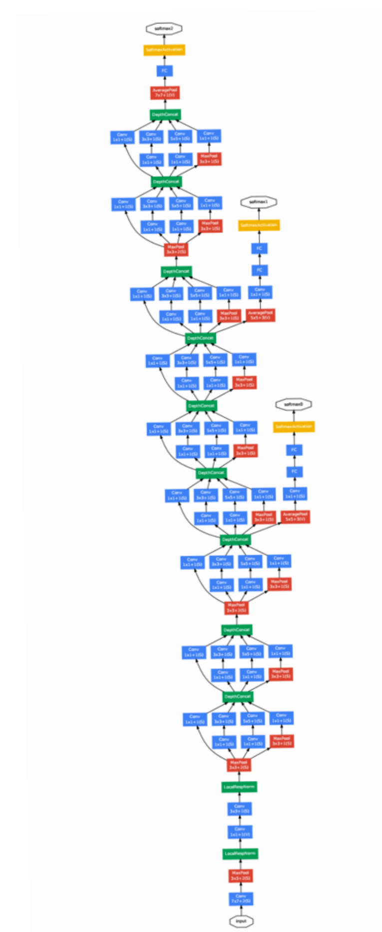
How do you actually train these things?

Roughly speaking:

Gather labeled data



Find a ConvNet architecture



Minimize the loss

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

Mini-batch Gradient Descent

Loop:

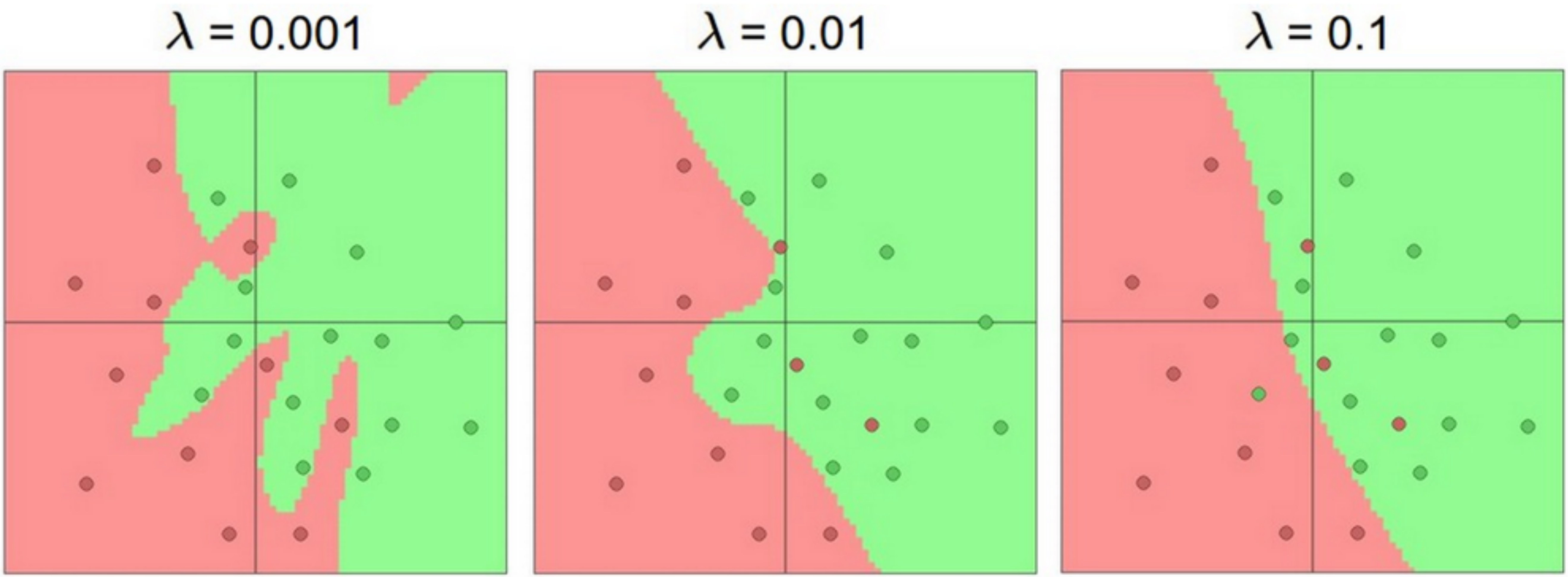
1. Sample a batch of training data (~100 images)
2. Forwards pass: compute loss (avg. over batch)
3. Backwards pass: compute gradient
4. Update all parameters

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

Regularization

Regularization reduces overfitting:

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



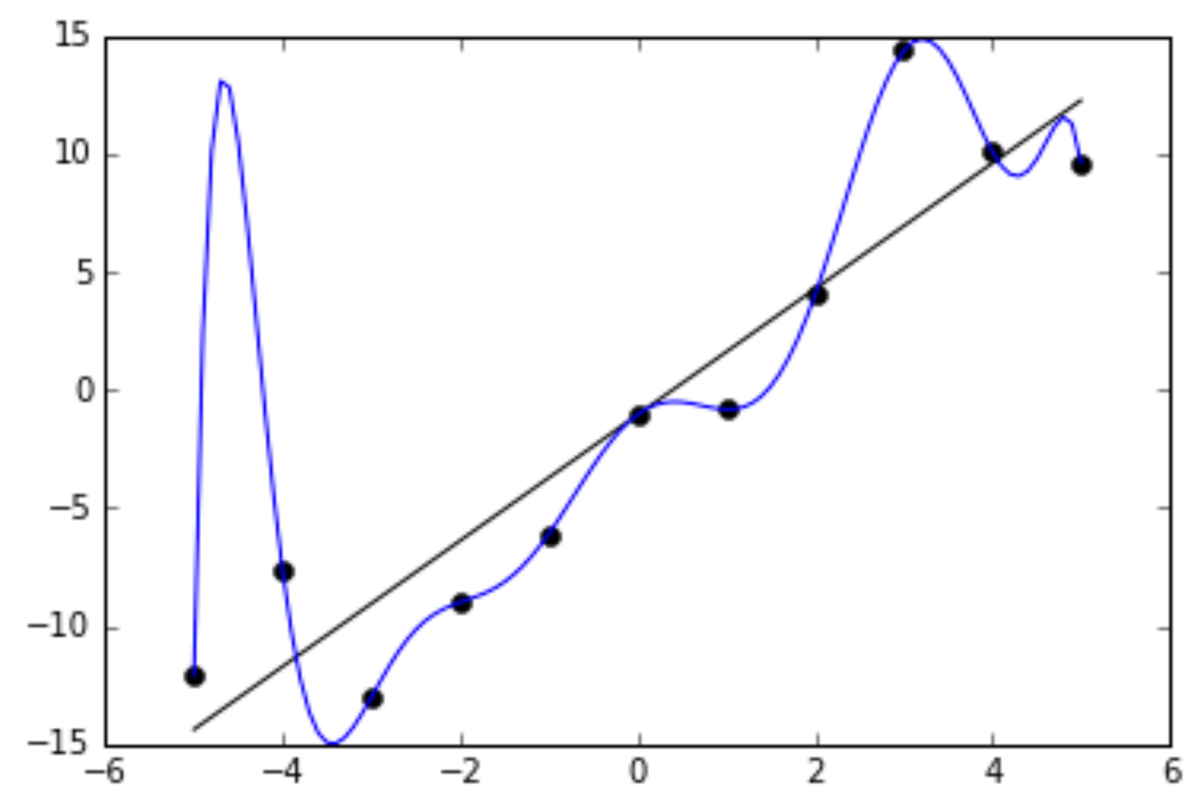
[Andrej Karpathy http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html]

Overfitting

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

Underfitting: insufficiently modeling the relationship in the training set

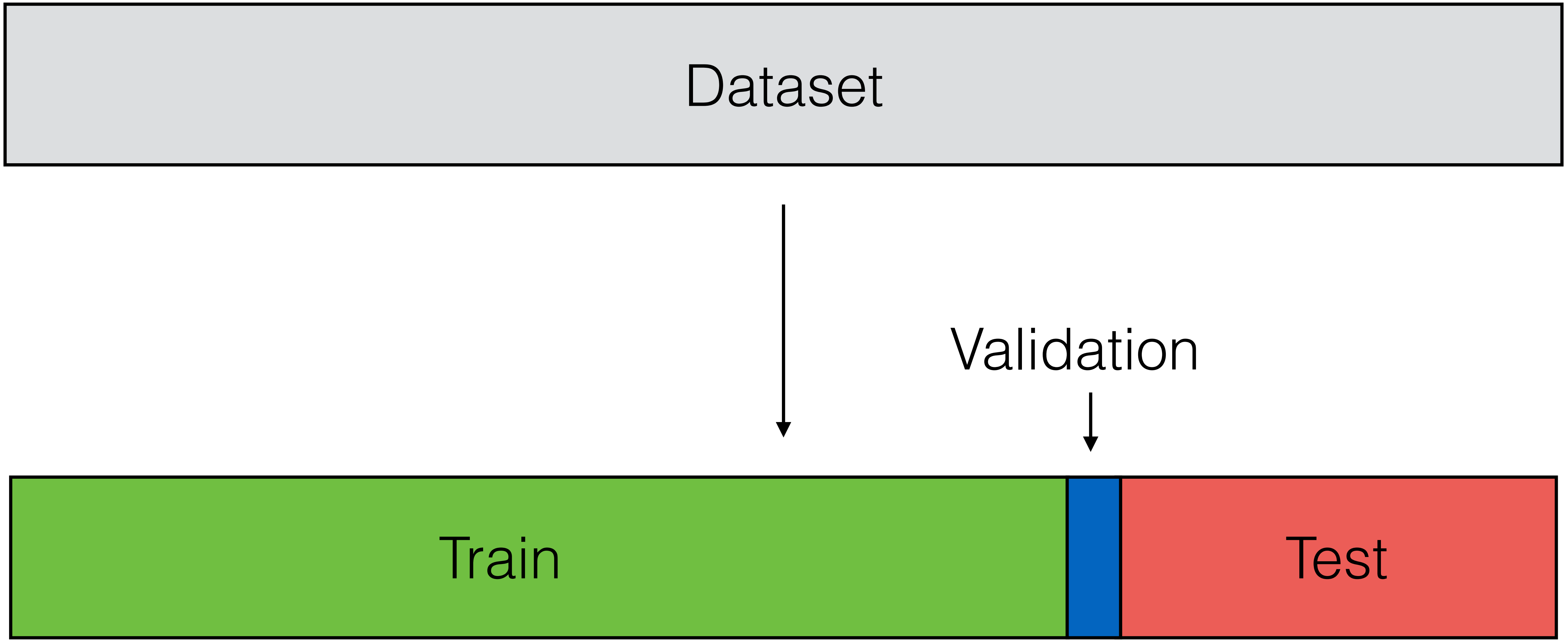
General rule: models that are “bigger” or have more capacity are more likely to overfit



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

(0) Dataset split

Split your data into “train”, “validation”, and “test”:



(0) Dataset split

Train: gradient descent and fine-tuning of parameters

Validation: determining hyper-parameters (learning rate, regularization strength, etc) and picking an architecture

Test: estimate real-world performance
(e.g. accuracy = fraction correctly classified)

(0) Dataset split

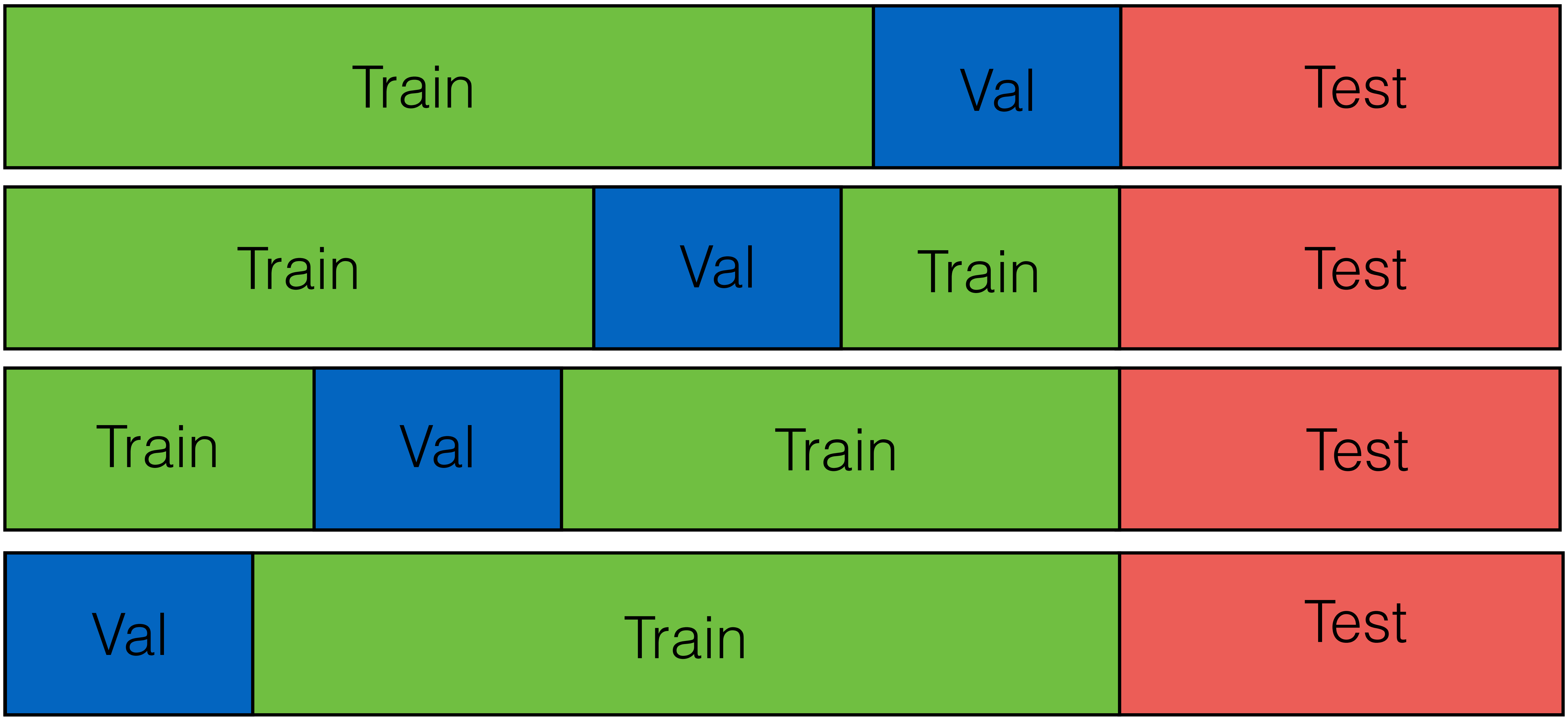
Be careful with false discovery:

To avoid false discovery, once we have used a test set once, we should *not use it again* (but nobody follows this rule, since it's expensive to collect datasets)

Instead, try and avoid looking at the test score until the end

(0) Dataset split

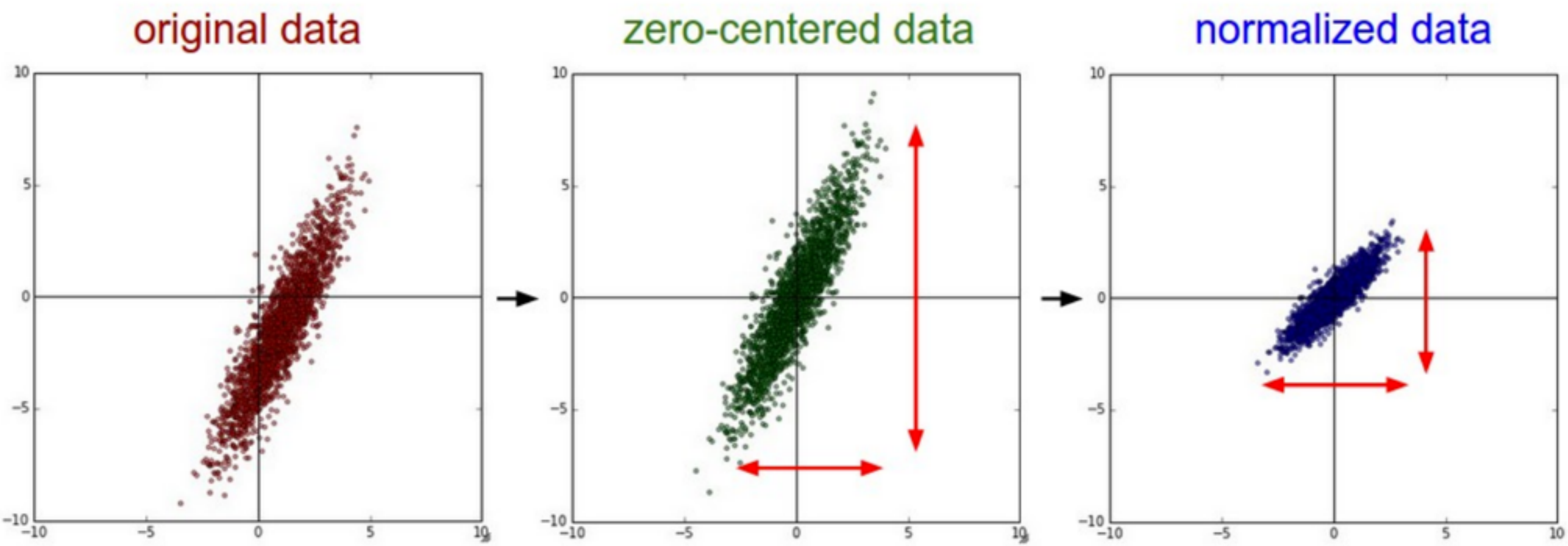
Cross-validation: cycle which data is used as validation



Average scores across validation splits

(1) Data preprocessing

Preprocess the data so that learning is better conditioned:



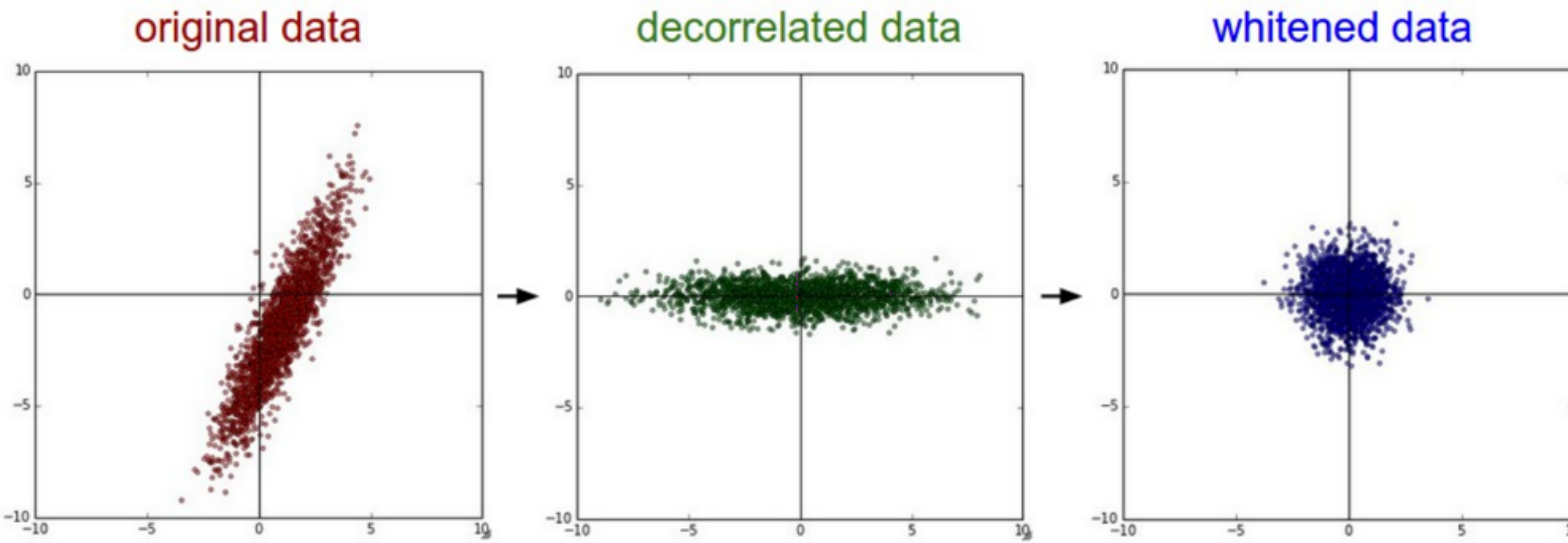

```
X -= np.mean(axis=0, keepdims=True)
```

```
X /= np.std(axis=0, keepdims=True)
```

Figure: Andrej Karpathy

(1) Data preprocessing

In practice, you may also see **PCA** and **Whitening** of the data:



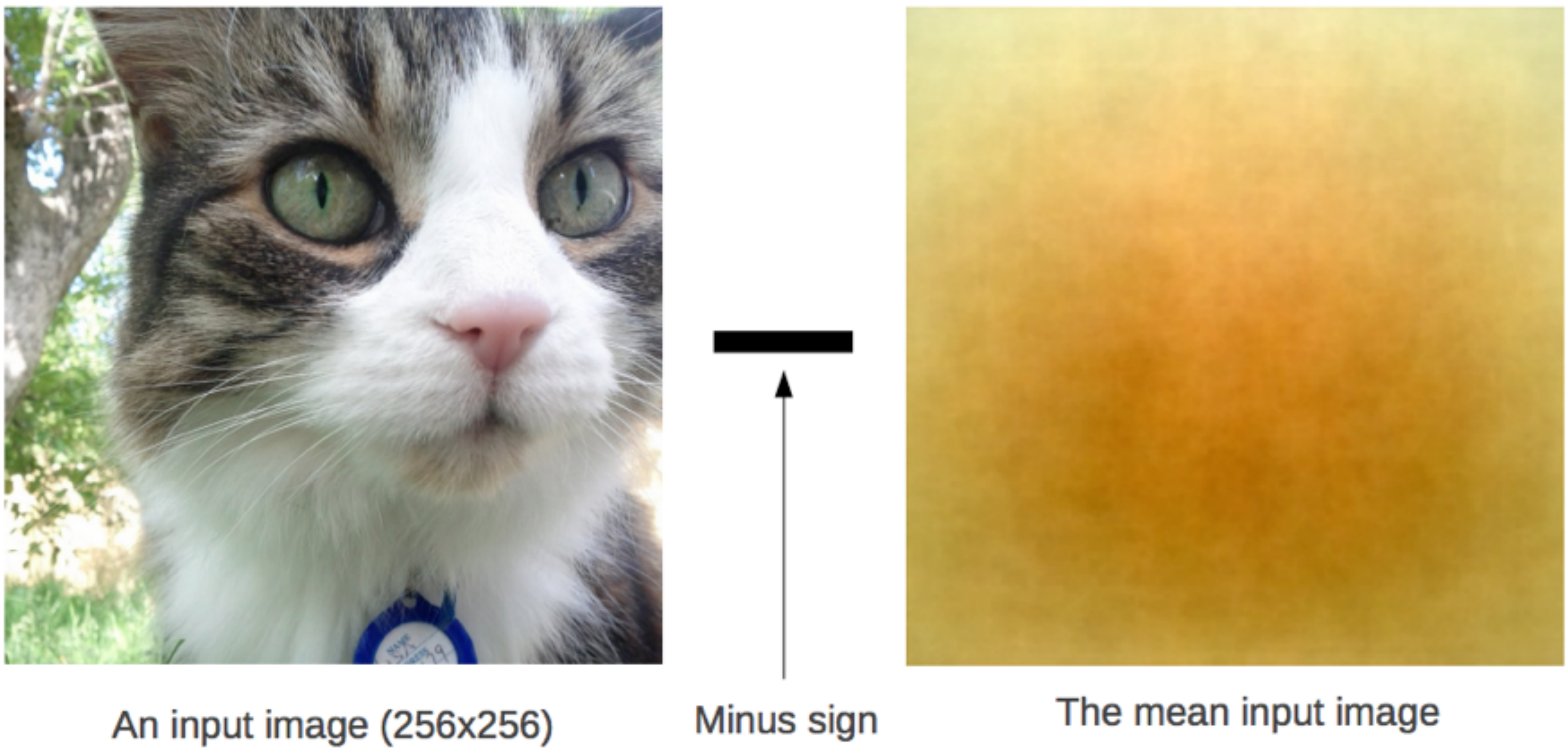
(data has diagonal covariance matrix)

(covariance matrix is the identity matrix)

Slide: Andrej Karpathy

(1) Data preprocessing

For ConvNets, typically only the mean is subtracted.



A per-channel mean also works (one value per R,G,B).

Figure: Alex Krizhevsky

(1) Data preprocessing

Augment the data — extract random crops from the input, with slightly jittered offsets. Without this, typical ConvNets (e.g. [Krizhevsky 2012]) overfit the data.

E.g. 224x224 patches extracted from 256x256 images

Randomly reflect horizontally

Perform the augmentation live during training

Figure: Alex Krizhevsky