SiTE
AAiT
AAU

Course Title: Deep Learning

Credit Hour: 3

Instructor: Fantahun B. (PhD) ✉ meetfantaai@gmail.com

Office: NB #

Ch-3 Convolutional Neural Networks

May-2023, AA

Ch-3 Convolutional Neural Networks

Agenda:

- The Convolution Operation
- Motivation
- Pooling
- Illustration
- CNN Architectures (assignment/project)
- CNN Applications

Ch-3 Convolutional Neural Networks

Objectives

After completing this chapter students will be able to:

- Explain the convolution and pooling operations
- Appreciate CNNs as powerful deep learning algorithms
- Appreciate the development of the various CNN architectures
- Implement several CNN architectures and solve real world problems

Ch-3 CNNs : Overview

- Convolutional networks convolutional (LeCun, 1989), also known as neural networks or CNNs, are a specialized kind of neural network for processing data that has a known, grid-like topology. Examples include
  - time-series data, which can be thought of as a 1D grid taking samples at regular time intervals, and
  - image data, which can be thought of as a 2D grid of pixels.
- The name "convolutional neural network" indicates that the network employs a mathematical operation called ***convolution*** (a specialized kind of linear operation).
- *CNNs are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.*
- CNNs have been very successful in practical applications.

Ch-3 CNNs: The Convolution Operation

- In its most general form, convolution is an operation on two functions of a realvalued argument.
- To motivate the definition of convolution, we start with examples of two functions we might use.
  - Suppose we are tracking the location of a spaceship with a laser sensor. Our laser sensor provides a single output x(t), the position of the spaceship at time t. Both x and t are real-valued, i.e., we can get a different reading from the laser sensor at any instant in time.
  - Now suppose that our laser sensor is somewhat noisy. To obtain a less noisy estimate of the spaceship's position, we would like to average together several measurements.
  - Of course, more recent measurements are more relevant, so we will want this to be a weighted average that gives more weight to recent measurements.

Ch-3 CNNs: The Convolution Operation

- We can do this with a weighting function w(a), where a is the age of a measurement.
- If we apply such a weighted average operation at every moment, we obtain a new function providing a smoothed estimate of the position s of the spaceship:

$$s(t) = \int x(a)w(t-a)da \tag{9.1}$$

- This operation is called convolution.

- The convolution operation is typically denoted with an asterisk:

$$s(t) = (x * w)(t) \tag{9.2}$$

Ch-3 CNNs: The Convolution Operation

- In our example, ω needs to be a valid probability density function, or the output is not a weighted average.
- Also, ω needs to be 0 for all negative arguments, or it will look into the future, which is presumably beyond our capabilities.
- These limitations are particular to our example though.
- In general, convolution is defined for any functions for which the above integral is defined, and may be used for other purposes besides taking weighted averages.
- In convolutional network terminology,
  - the first argument (in this example, the function x) to the convolution is often referred to as the input and
  - the second argument (in this example, the function w) as the kernel.
  - The output is sometimes referred to as the feature map.

Ch-3 CNNs: The Convolution Operation

- In our example, the idea of a laser sensor that can provide measurements at every instant in time is not realistic. Usually, when we work with data on a computer, time will be discretized, and our sensor will provide data at regular intervals.
- In our example, it might be more realistic to assume that our laser provides a measurement once per second. The time index t can then take on only integer values.
- If we now assume that x and w are defined only on integer t, we can define the discrete convolution:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \qquad (9.3)$$

- In machine learning applications, the input is usually a multidimensional array of data and the kernel is usually a multidimensional array of parameters (tensors).

Ch-3 CNNs: The Convolution Operation

- B/c each element of the input and kernel must be explicitly stored separately, we usually assume that these functions are zero everywhere but the finite set of points for which we store the values.
- That is, in practice we can implement the infinite summation as a summation over a finite number of array elements.
- Finally, we often use convolutions over more than one axis at a time. For example, if we use a two-dimensional image I as our input, we probably also want to use a two-dimensional kernel K:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \qquad (9.4)$$

- Convolution is commutative, hence we can equivalently write:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n). \qquad (9.5)$$

Ch-3 CNNs: The Convolution Operation

- Usually the latter formula is more straightforward to implement in a machine learning library, because there is less variation in the range of valid values of m and n.
- The commutative property of convolution arises because we have flipped the kernel relative to the input, in the sense that as m increases, the index into the input increases, but the index into the kernel decreases.
- The only reason to flip the kernel is to obtain the commutative property. While the commutative property is useful for writing proofs, it is not usually an important property of a neural network implementation. Instead, many neural network libraries implement a related function called the cross-correlation, which is the same as convolution but without flipping the kernel:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n). \qquad (9.6)$$

Ch-3 CNNs: The Convolution Operation

- Many machine learning libraries implement cross-correlation but call it convolution. In this text we will follow this convention of calling both operations convolution, and specify whether we mean to flip the kernel or not in contexts where kernel flipping is relevant.
- In the context of machine learning, the learning algorithm will learn the appropriate values of the kernel in the appropriate place, so an algorithm based on convolution with kernel flipping will learn a kernel that is flipped relative to the kernel learned by an algorithm without the flipping.
- It is also rare for convolution to be used alone in machine learning; instead convolution is used simultaneously with other functions, and the combination of these functions does not commute regardless of whether the convolution operation flips its kernel or not.
- See Fig. 9.1 for an example of convolution (without kernel flipping) applied to a 2-D tensor.

Ch-3 CNNs: The Convolution Operation

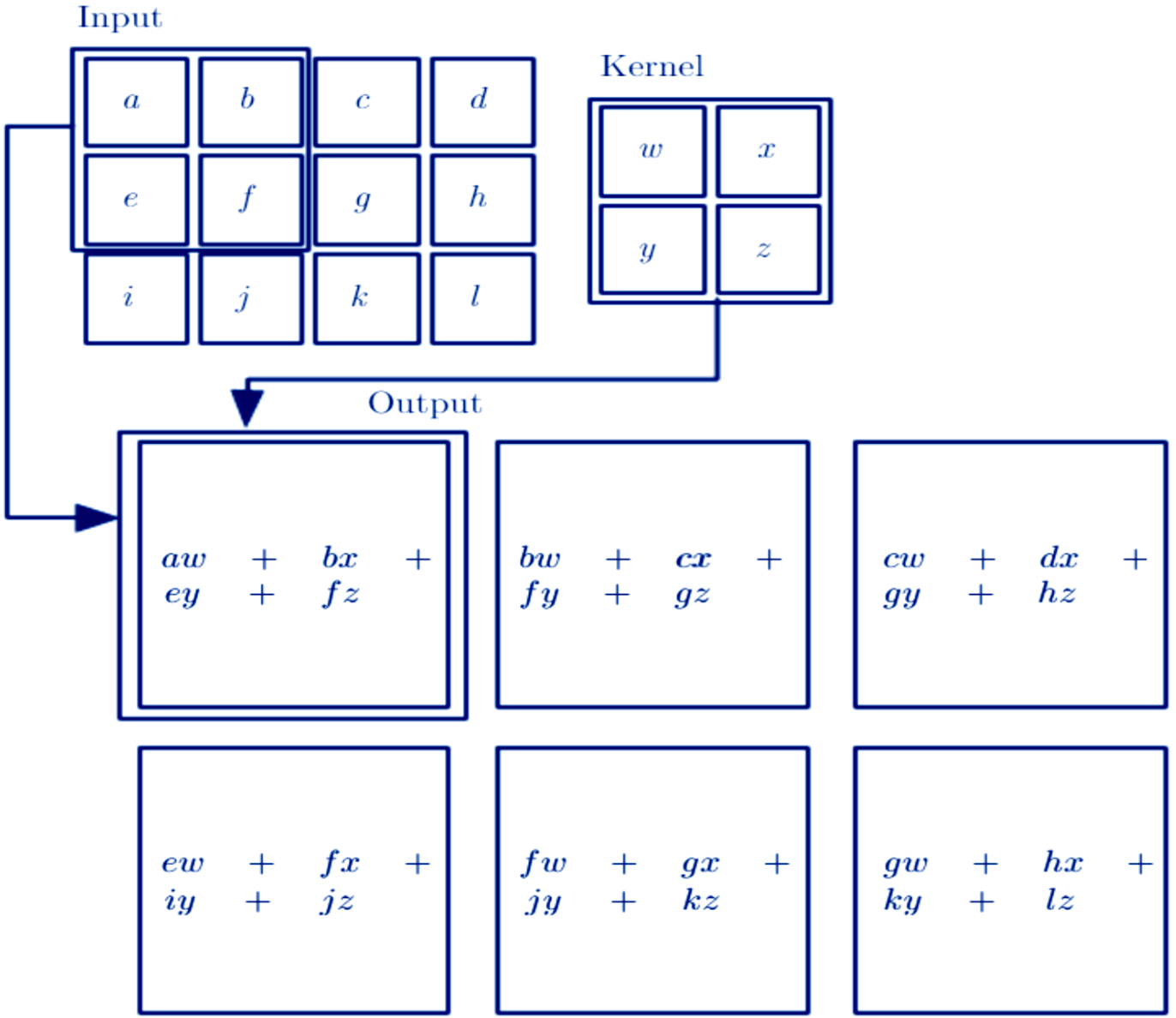


Figure 9.1: An example of 2-D convolution without kernel-flipping. In this case we restrict the output to only positions where the kernel lies entirely within the image, called "valid" convolution in some contexts.

Ch-3 CNNs: The Convolution Operation

- Discrete convolution can be viewed as multiplication by a matrix. However, the matrix has several entries constrained to be equal to other entries. For example, for univariate discrete convolution, each row of the matrix is constrained to be equal to the row above shifted by one element. This is known as a Toeplitz matrix.
- In two dimensions, a doubly block circulant matrix corresponds to convolution.
- In addition to these constraints that several elements be equal to each other, convolution usually corresponds to a very sparse matrix (a matrix whose entries are mostly equal to zero). This is because the kernel is usually much smaller than the input image.

Ch-3 CNNs: The Convolution Operation

- Any neural network algorithm that works with matrix multiplication and does not depend on specific properties of the matrix structure should work with convolution, without requiring any further changes to the neural network.
- Typical convolutional neural networks do make use of further specializations in order to deal with large inputs efficiently, but these are not strictly necessary from a theoretical perspective.

Ch-3 CNNs: Motivation

- Convolution leverages three important ideas that can help improve a machine learning system:
  - sparse interactions
  - parameter sharing equivariant , and
  - representations.
  - Moreover, convolution provides a means for working with inputs of variable size.

Ch-3 CNNs: Motivation

Sparse Interactions

- Traditional neural network layers use matrix multiplication by a matrix of parameters with a separate parameter describing the interaction between each input unit and each output unit. This means every output unit interacts with every input unit.
- Convolutional networks, however, typically have sparse interactions / sparse connectivity /sparse weights. This is accomplished by making the kernel smaller than the input.
- For example, when processing an image, the input image might have thousands or millions of pixels, but we can detect small, meaningful features such as edges with kernels that occupy only tens or hundreds of pixels. We need to store fewer parameters:
  - reduces the memory requirements of the model and
  - improves its statistical efficiency).

Ch-3 CNNs: Motivation

Sparse Interactions

- Computing the output requires fewer operations.
- These improvements in efficiency are usually quite large.
- If there are m inputs and n outputs:
  - matrix multiplication requires m×n parameters → O(m × n) runtime (per example).
  - If we limit the number of connections each output may have to k, then the sparsely connected approach requires only k × n parameters and O(k × n) runtime.
  - For many practical applications, it is possible to obtain good performance on the machine learning task while keeping k several orders of magnitude smaller than m.

Ch-3 CNNs: Motivation

Sparse Interactions

- Graphical demonstrations of sparse connectivity, Fig. 9.2 and Fig. 9.3.

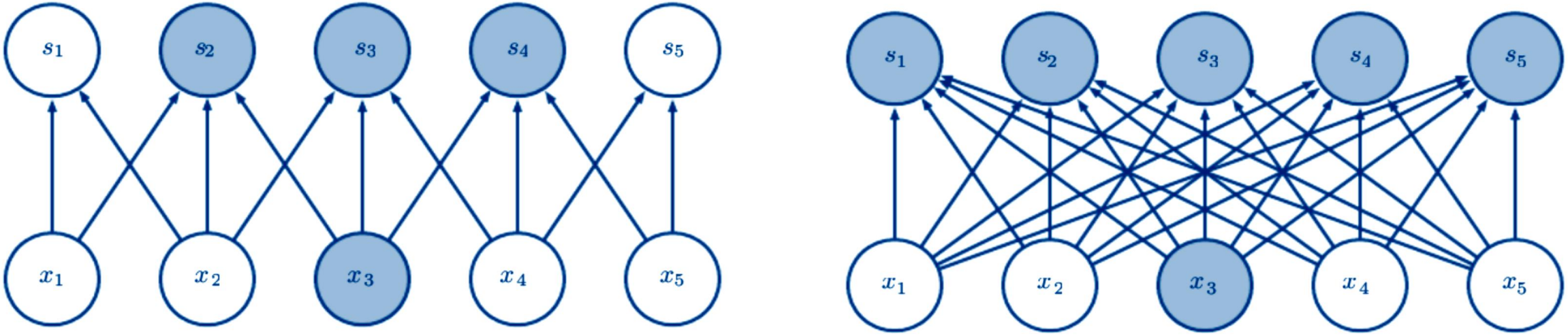


Figure 9.2: Sparse connectivity, viewed from below: We highlight one input unit, x_3, and also highlight the output units in s that are affected by this unit. **(Left)** When s is formed by convolution with a kernel of width 3, only three outputs are affected by x_3. **(Right)** When s is formed by matrix multiplication, connectivity is no longer sparse, so all of the outputs are affected by x_3.

Ch-3 CNNs: Motivation

Sparse Interactions

- Graphical demonstrations of sparse connectivity, Fig. 9.2 and Fig. 9.3.

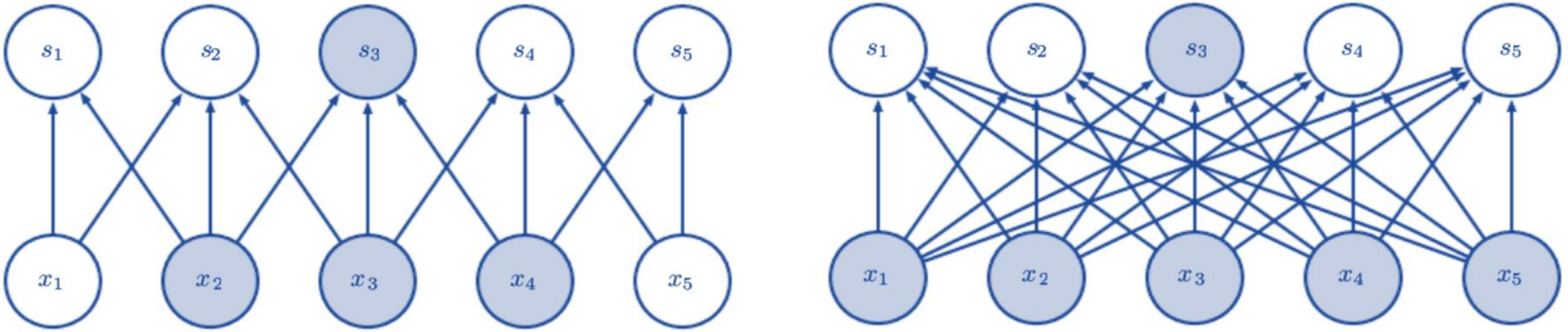


Figure 9.3: Sparse connectivity, viewed from above: We highlight one output unit,s3, and also highlight the input units in x that affect this unit. These units are known as the receptive field of s3. **(Left)** When s is formed by convolution with a kernel of width 3, only three inputs affect s3. (**Right)** When s is formed by matrix multiplication, connectivity is no longer sparse, so all of the inputs affect s3.

Ch-3 CNNs: Motivation

Sparse Interactions

- In a deep CNNs, units in the deeper layers may indirectly interact with a larger portion of the input, as shown in Fig. 9.4. This allows the network to efficiently describe complicated interactions between many variables by constructing such interactions from simple building blocks that each describe only sparse interactions.

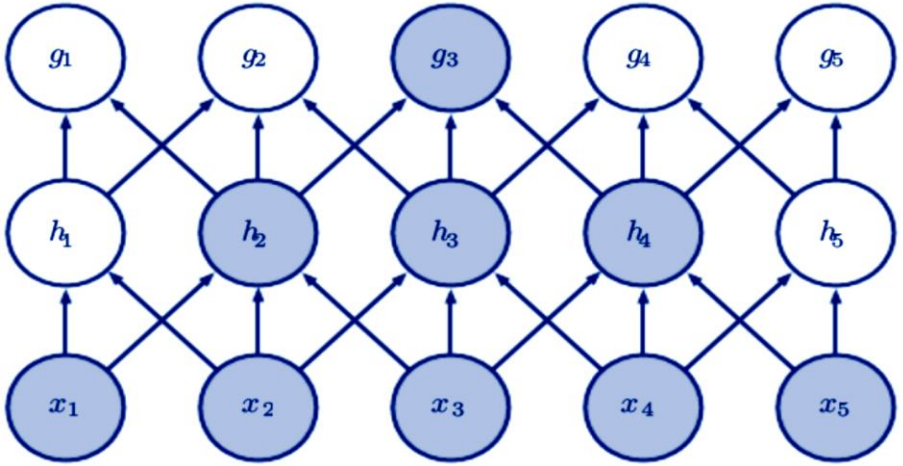


Figure 9.4: The receptive field of the units in the deeper layers of a convolutional network is larger than the receptive field of the units in the shallow layers. This effect increases if the network includes architectural features like strided convolution (Fig. 9.12) or pooling (Sec. 9.3). This means that even though direct connections in a convolutional net are very sparse, units in the deeper layers can be indirectly connected to all or most of the input image.

Ch-3 CNNs: Motivation

Parameter sharing

- Parameter sharing refers to using the same parameter for more than one function in a model.
- In a traditional neural net, each element of the weight matrix is used exactly once when computing the output of a layer. It is multiplied by one element of the input and then never revisited.
- In a CNN, each member of the kernel is used at every position of the input (except perhaps some of the boundary pixels, depending on the design decisions regarding the boundary).

Ch-3 CNNs: Motivation

Parameter sharing

- The parameter sharing used by the convolution operation means that rather than learning a separate set of parameters for every location, we learn only one set.
- This does not affect the runtime of forward propagation—it is still O(k × n)—but it does further reduce the storage requirements of the model to k parameters.
- Since m and n are usually roughly the same size, k is practically insignificant compared to m× n.
- Convolution is thus dramatically more efficient than dense matrix multiplication in terms of the memory requirements and statistical efficiency.

Ch-3 CNNs: Motivation

Parameter sharing

- Graphical depiction of how parameter sharing works, Fig. 9.5.

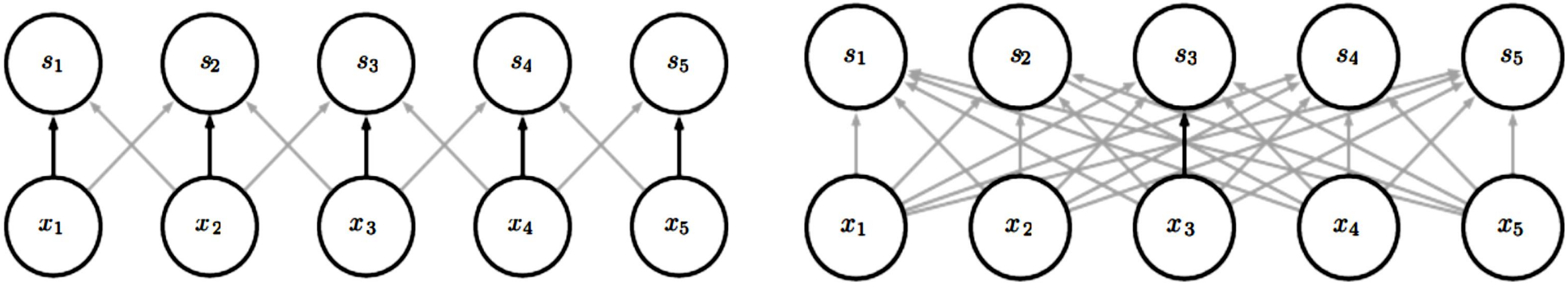


Figure 9.5: Parameter sharing: Black arrows indicate the connections that use a particular parameter in two different models. **(Left)** The black arrows indicate uses of the central element of a 3-element kernel in a convolutional model. Due to parameter sharing, this single parameter is used at all input locations. **(Right)** The single black arrow indicates the use of the central element of the weight matrix in a fully connected model. This model has no parameter sharing so the parameter is used only once.

Ch-3 CNNs: Motivation

- As an example of both of these first two principles in action, Fig. 9.6 shows how sparse connectivity and parameter sharing can dramatically improve the efficiency of a linear function for detecting edges in an image.

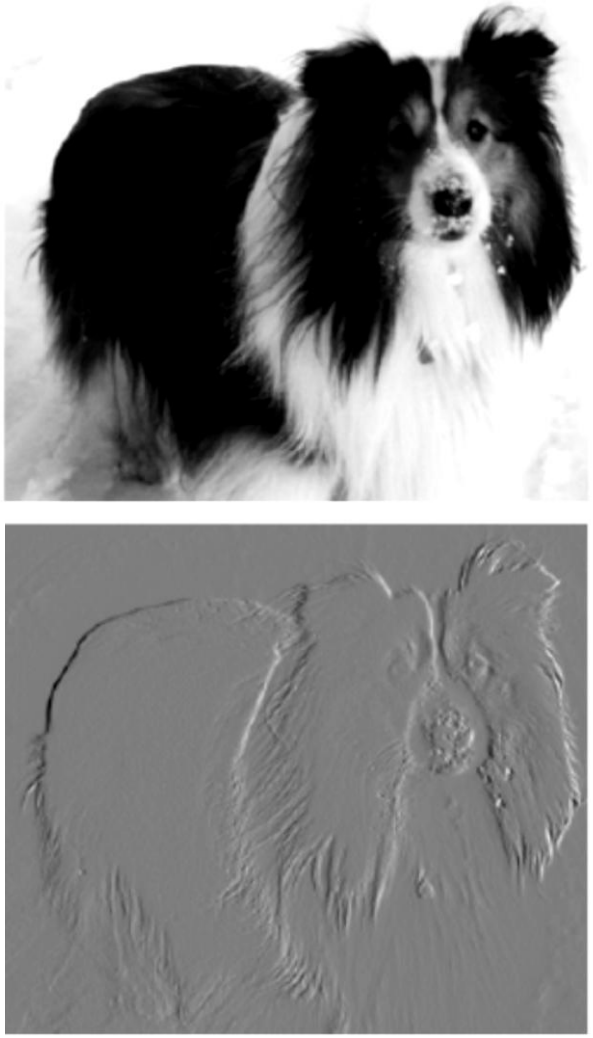


Figure 9.6: Efficiency of edge detection.
The image at the bottom was formed by taking each pixel in the original image and subtracting the value of its neighboring pixel on the left. This shows the strength of all of the vertically oriented edges in the input image, which can be a useful operation for object detection. Both images are 280 pixels tall. The input image is 320 pixels wide while the output image is 319 pixels wide. This transformation can be described by a convolution kernel containing two elements, and requires 319 × 280 × 3 = **267, 960** floating point operations (*two multiplications* and *one addition* per output pixel) to compute using convolution. To describe the same transformation with a matrix multiplication would take 320× 280 × 319 × 280, or **over eight billion**, entries in the matrix, making convolution four billion times more efficient for representing this transformation. The straightforward matrix multiplication algorithm performs over sixteen billion floating point operations, making convolution roughly 60,000 times more efficient computationally. Of course, most of the entries of the matrix would be zero. If we stored only the nonzero entries of the matrix, then both matrix multiplication and convolution would require the same number of floating point operations to compute. The matrix would still need to contain 2× 319 × 280 = 178, 640 entries. Convolution is an extremely efficient way of describing transformations that apply the same linear transformation of a small, local region across the entire input. (Photo credit: Paula Goodfellow)

Ch-3 CNNs: Motivation

- In the case of convolution, the particular form of parameter sharing causes the layer to have a property called equivariance to translation.
- To say a function is equivariant means that if the input changes, the output changes in the same way. Specifically, a function f(x) is equivariant to a function g if f(g(x)) = g(f(x)).
- In the case of convolution, if we let g be any function that translates the input, i.e., shifts it, then the convolution function is equivariant to g.
  - For example, let I be a function giving image brightness at integer coordinates. Let g be a function mapping one image function to another image function, such that I' = g(I) is the image function with I'(x, y) = I(x−1, y).
  - This shifts every pixel of I one unit to the right. If we apply this transformation to I , then apply convolution, the result will be the same as if we applied convolution to I', then applied the transformation g to the output.

Ch-3 CNNs: Motivation

- When processing time series data, this means that convolution produces a sort of timeline that shows when different features appear in the input.
- If we move an event later in time in the input, the exact same representation of it will appear in the output, just later in time.
- Similarly with images, convolution creates a 2-D map of where certain features appear in the input.
- If we move the object in the input, its representation will move the same amount in the output. This is useful for when we know that some function of a small number of neighboring pixels is useful when applied to multiple input locations.
  - For example, when processing images, it is useful to detect edges in the first layer of a convolutional network. The same edges appear more or less everywhere in the image, so it is practical to share parameters across the entire image.

Ch-3 CNNs: Motivation

- In some cases, *we may not wish to share parameters across the entire image*.
  - For example, if we are processing images that are cropped to be centered on an individual's face, we probably want to extract different features at different locations—the part of the network processing the top of the face needs to look for eyebrows, while the part of the network processing the bottom of the face needs to look for a chin.
- Convolution is not naturally equivariant *to some other transformations, such as changes in the scale or rotation of an image*.

Ch-3 CNNs: Motivation

- Other mechanisms are necessary for handling these kinds of transformations.
- Finally, some kinds of data cannot be processed by neural networks defined by matrix multiplication with a fixed-shape matrix.
- Convolution enables processing of some of these kinds of data. We discuss this further in Sec. 9.7.

Ch-3 CNNs: Pooling

- A typical layer of a convolutional network consists of three stages (see Fig. 9.7).
  1) first stage, the layer performs several convolutions in parallel to produce a set of linear activations.
  2) second stage, each linear activation is run through a nonlinear activation function, such as the rectified linear activation function. This stage is sometimes called the *detector stage*.
  3) third stage, we use a pooling function to modify the output of the layer further. A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.
     - *max pooling* (Zhou and Chellappa, 1988) operation reports the maximum output within a rectangular neighborhood.
     - *average pooling* reports average of a rectangular neighborhood,
     - the *L2 norm* of a rectangular neighborhood, or
     - a *weighted average* based on the distance from the central pixel.

Ch-3 CNNs: Pooling

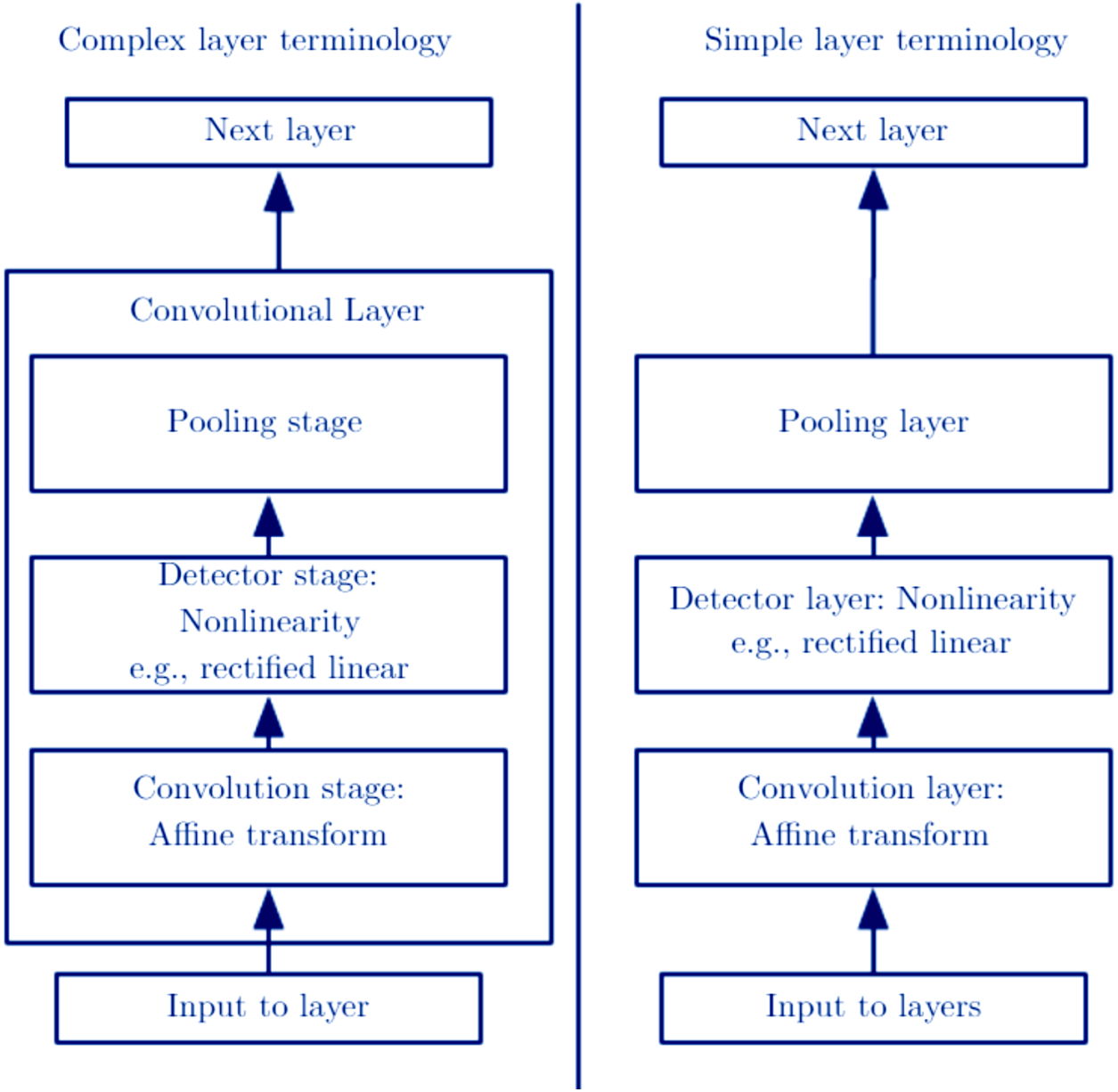


Figure 9.7: The components of a typical convolutional neural network layer. There are two commonly used sets of terminology for describing these layers.

- (Left) In this terminology, the convolutional net is viewed as a small number of relatively complex layers, with each layer having many "stages." In this terminology, there is a one-to-one mapping between kernel tensors and network layers. In this book we generally use this terminology.
- (Right) In this terminology, the convolutional net is viewed as a larger number of simple layers; every step of processing is regarded as a layer in its own right. This means that not every "layer" has parameters.

Ch-3 CNNs: Pooling

- In all cases, pooling helps to *make the representation become approximately invariant to small translations of the input*. Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change. See Fig. 9.8 for an example of how this works.
- Invariance to local translation can be a very useful property if we care more about the existence of some feature than its location.
  - For example, when determining whether an image contains a face, we need not know the location of the eyes with pixel-perfect accuracy, we just need to know that there is an eye on the left side of the face and an eye on the right side of the face.
- In other contexts, it is more important to preserve the location of a feature.
  - For example, if we want to find a corner defined by two edges meeting at a specific orientation, we need to preserve the location of the edges well enough to test whether they meet.

Ch-3 CNNs: Pooling

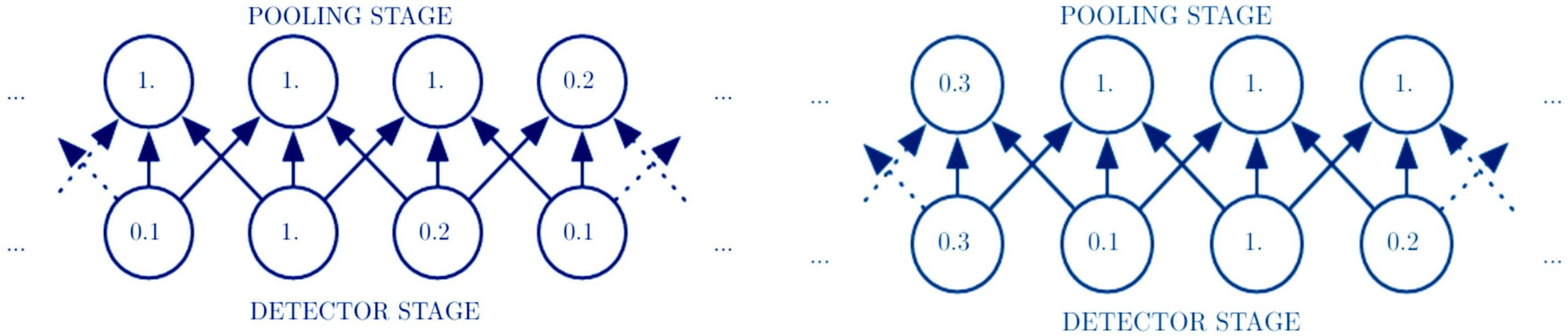


Figure 9.8: Max pooling introduces invariance. (Left) A view of the middle of the output of a convolutional layer. The bottom row shows outputs of the nonlinearity. The top row shows the outputs of max pooling, with a stride of one pixel between pooling regions and a pooling region width of three pixels. (Right) A view of the same network, after the input has been shifted to the right by one pixel. *Every value in the bottom row has changed, but only half of the values in the top row have changed*, because the max pooling units are only sensitive to the maximum value in the neighborhood, not its exact location.

Ch-3 CNNs: Pooling

- The use of pooling can be viewed as adding an infinitely strong prior that the function the layer learns must be invariant to small translations. When this assumption is correct, it can greatly improve the statistical efficiency of the network.
- Pooling over spatial regions produces invariance to translation, but if we pool over the outputs of separately parametrized convolutions, the features can learn which transformations to become invariant to (see Fig. 9.9).
- Because pooling summarizes the responses over a whole neighborhood, it is possible to use fewer pooling units than detector units, by reporting summary statistics for pooling regions spaced k pixels apart rather than 1 pixel apart.
- See Fig. 9.10 for an example. This improves the computational efficiency of the network because the next layer has roughly k times fewer inputs to process.

Ch-3 CNNs: Pooling

- When the number of parameters in the next layer is a function of its input size (such as when the next layer is fully connected and based on matrix multiplication) this reduction in the input size can also result in improved statistical efficiency and reduced memory requirements for storing the parameters.
- For many tasks, pooling is essential for handling inputs of varying size.
  - For example, if we want to classify images of variable size, the input to the classification layer must have a fixed size.
  - This is usually accomplished by varying the size of an offset between pooling regions so that the classification layer always receives the same number of summary statistics regardless of the input size.

Ch-3 CNNs: Pooling

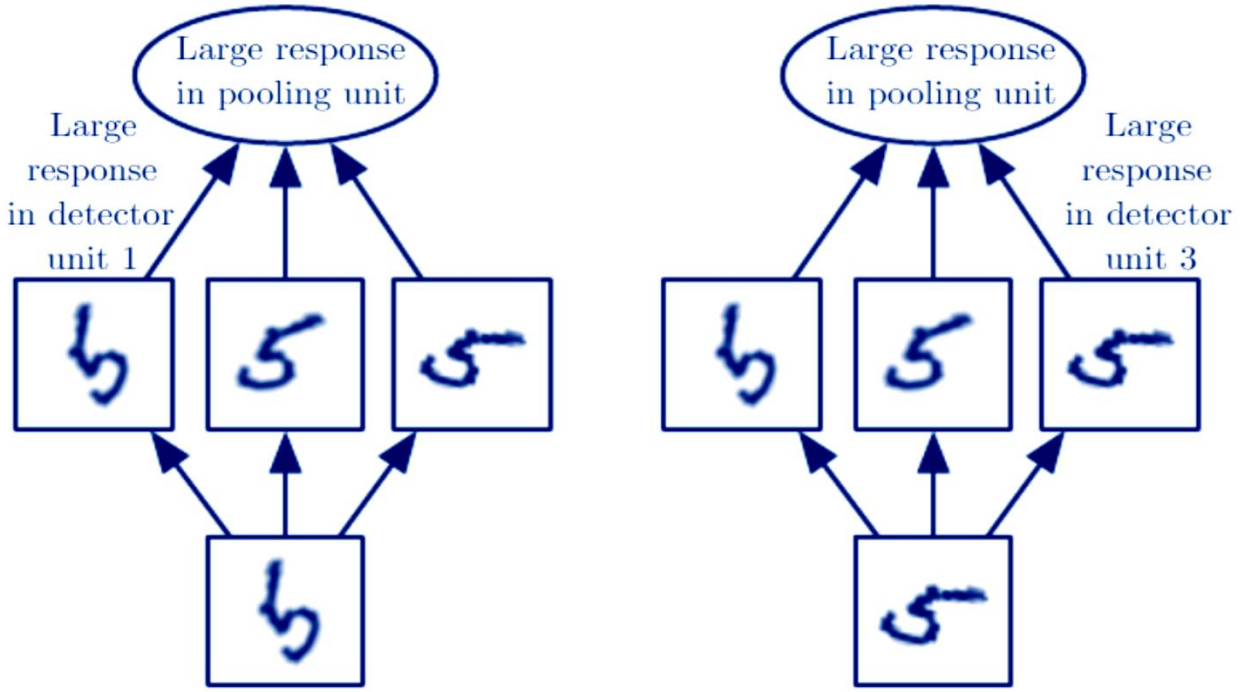


Figure 9.9: Example of learned invariances: A pooling unit that pools over multiple features that are learned with separate parameters can learn to be invariant to transformations of the input. Here we show how a set of three learned filters and a max pooling unit can learn to become invariant to rotation. All three filters are intended to detect a hand-written 5. Each filter attempts to match a slightly different orientation of the 5. When a 5 appears in the input, the corresponding filter will match it and cause a large activation in a detector unit. The max pooling unit then has a large activation regardless of which pooling unit was activated. We show here how the network processes two different inputs, resulting in two different detector units being activated. The effect on the pooling unit is roughly the same either way. This principle is leveraged by maxout networks (Goodfellow et al., 2013a) and other convolutional networks. Max pooling over spatial positions is naturally invariant to translation; this multi-channel approach is only necessary for learning other transformations.

Ch-3 CNNs: Pooling

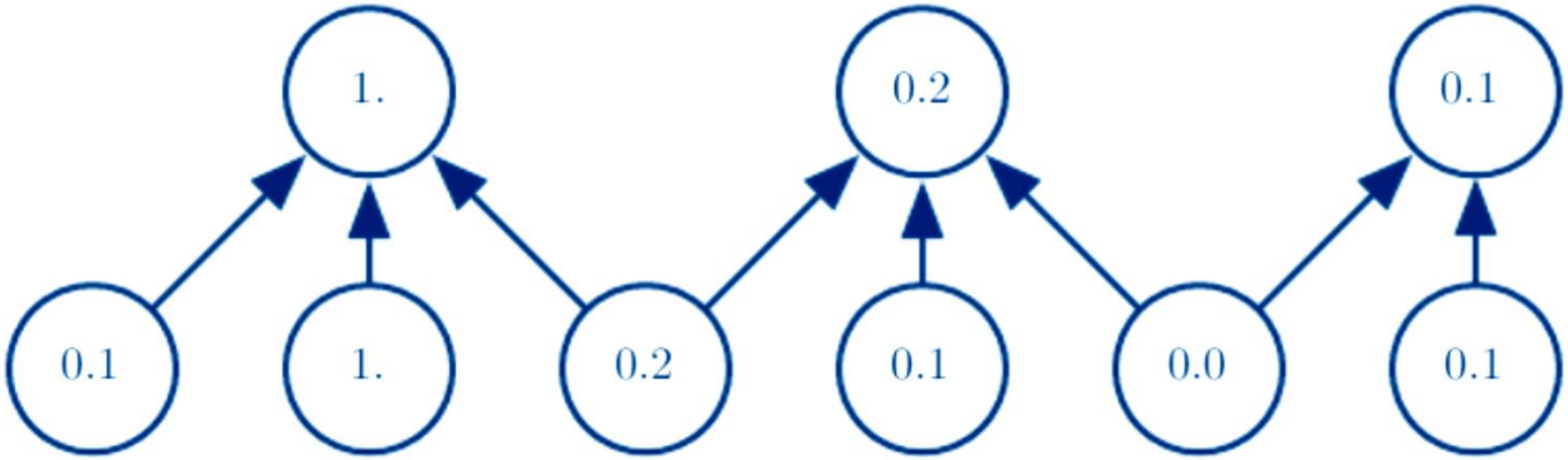


Figure 9.10: Pooling with downsampling. Here we use max-pooling with a pool width of three and a stride between pools of two. This reduces the representation size by a factor of two, which reduces the computational and statistical burden on the next layer. Note that the rightmost pooling region has a smaller size, but must be included if we do not want to ignore some of the detector units.

Ch-3 CNNs: Pooling

- Some theoretical work gives guidance as to which kinds of pooling one should use in various situations (Boureau et al., 2010).
- It is also possible to dynamically pool features together, for example, by running a clustering algorithm on the locations of interesting features (Boureau et al., 2011). This approach yields a different set of pooling regions for each image.
- Another approach is to learn a single pooling structure that is then applied to all images (Jia et al., 2012).
- Some examples of complete convolutional network architectures for classification using convolution and pooling are shown in Fig. 9.11.

Ch-3 CNNs:

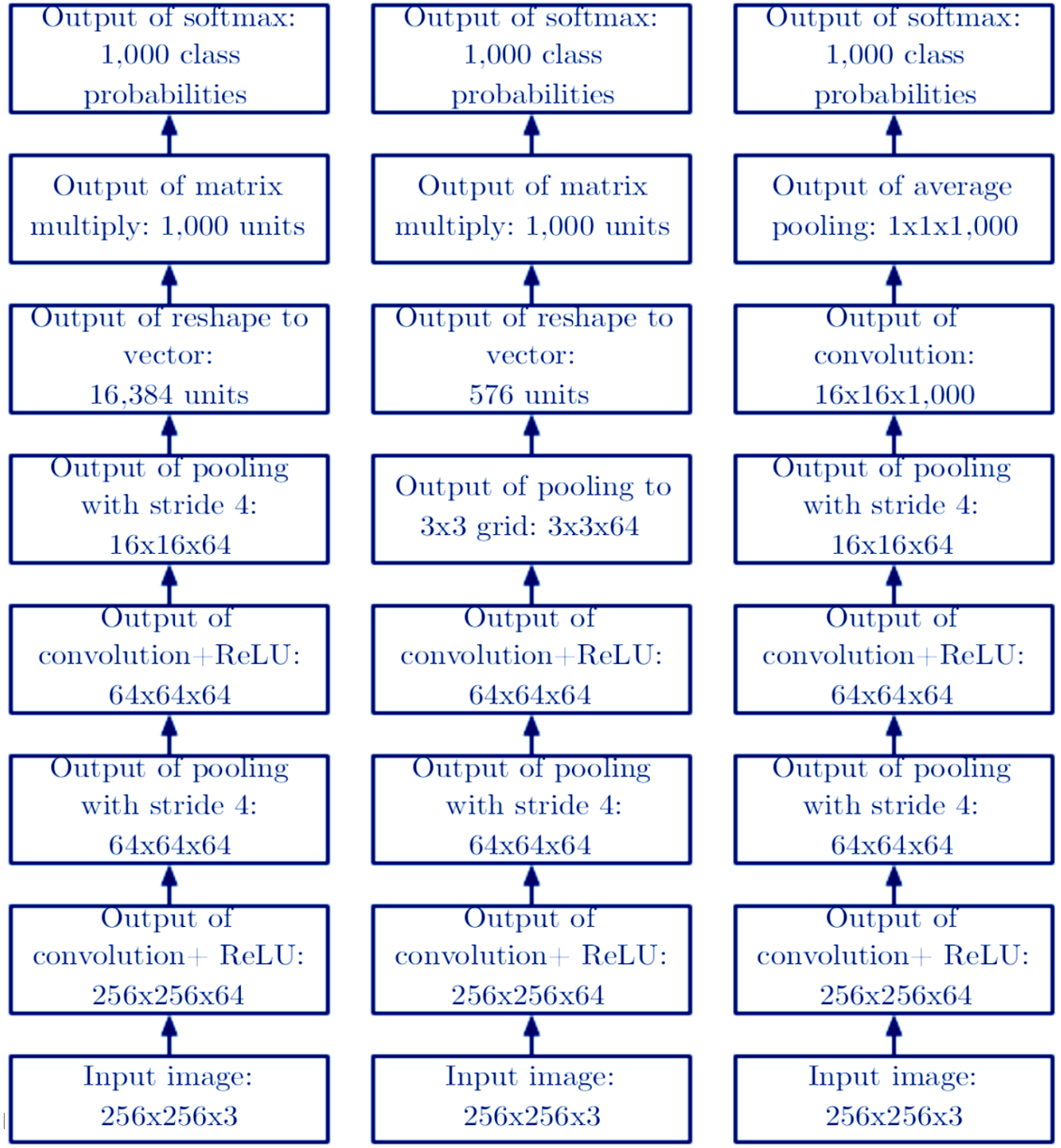
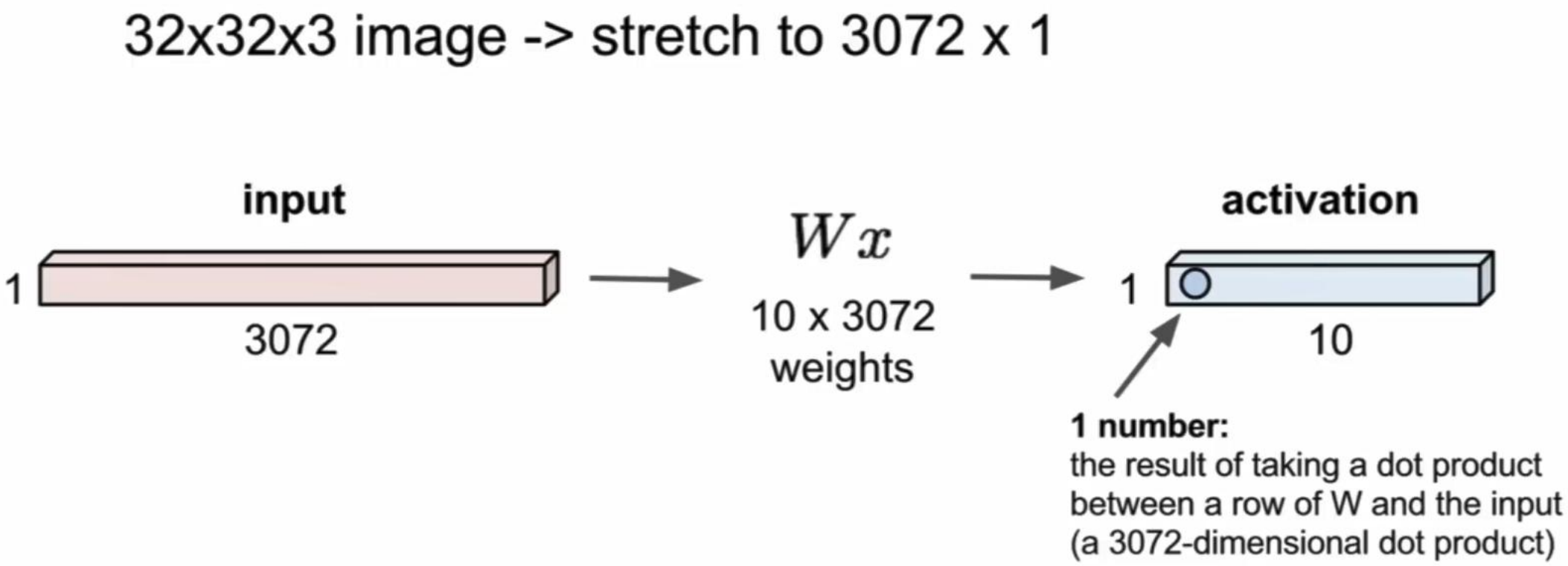


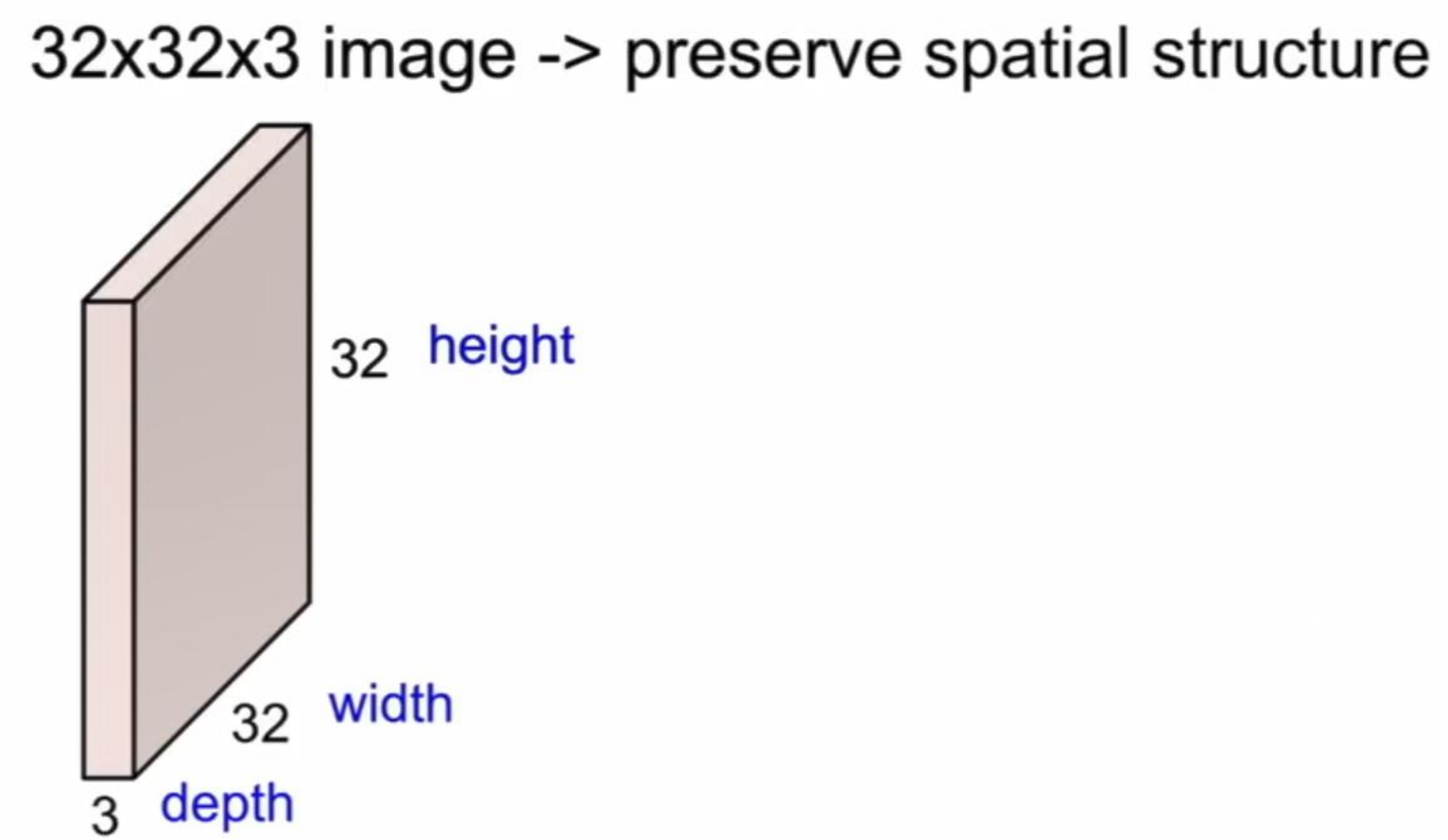
Figure 9.11: Examples of architectures for classification with convolutional networks. The specific strides and depths used in this figure are not advisable for real use; they are designed to be very shallow in order to fit onto the page. Real convolutional networks also often involve significant amounts of branching, unlike the chain structures used here for simplicity. (Left) A convolutional network that processes a fixed image size. After alternating between convolution and pooling for a few layers, the tensor for the convolutional feature map is reshaped to flatten out the spatial dimensions. The rest of the network is an ordinary feedforward network classifier, as described in Chapter 6. (Center) A convolutional network that processes a variable-sized image, but still maintains a fully connected section. This network uses a pooling operation with variably-sized pools but a fixed number of pools, in order to provide a fixed-size vector of 576 units to the fully connected portion of the network. (Right) A convolutional network that does not have any fully connected weight layer. Instead, the last convolutional layer outputs one feature map per class. The model presumably learns a map of how likely each class is to occur at each spatial location. Averaging a feature map down to a single value provides the argument to the softmax classifier at the top.

Ch-3 CNNs: Illustration
Fully Connected Layer



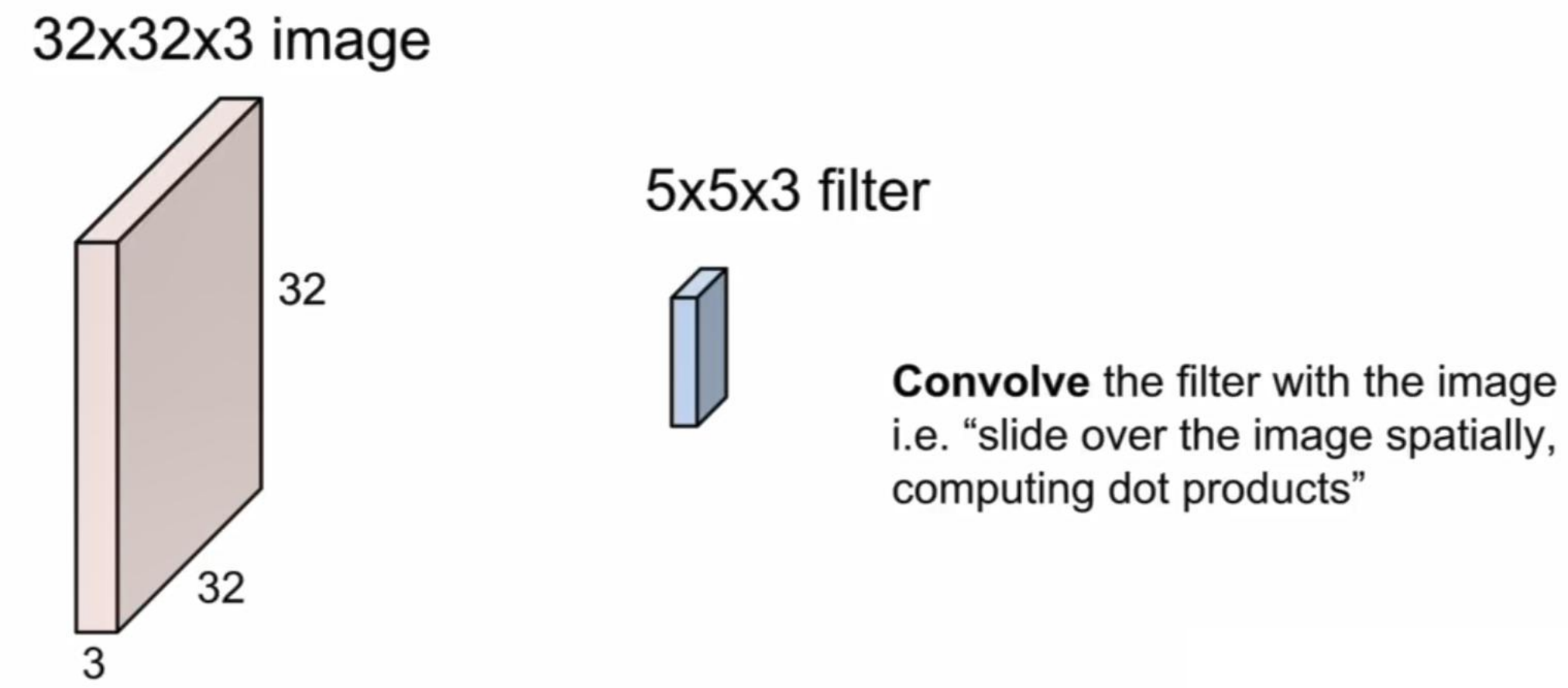
Ch-3 CNNs: Illustration

- Convolution Layer



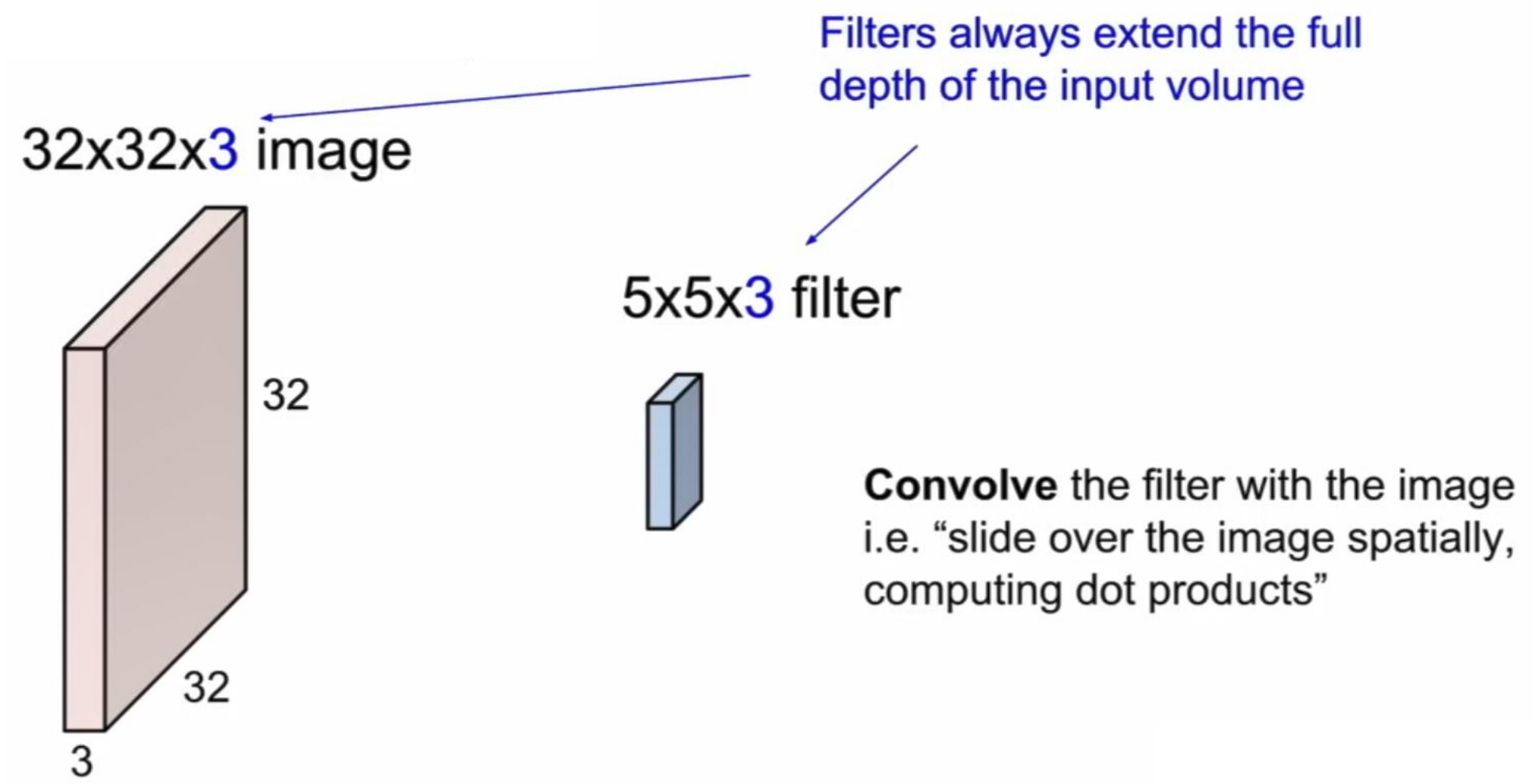
Ch-3 CNNs: Illustration

- Convolution Layer



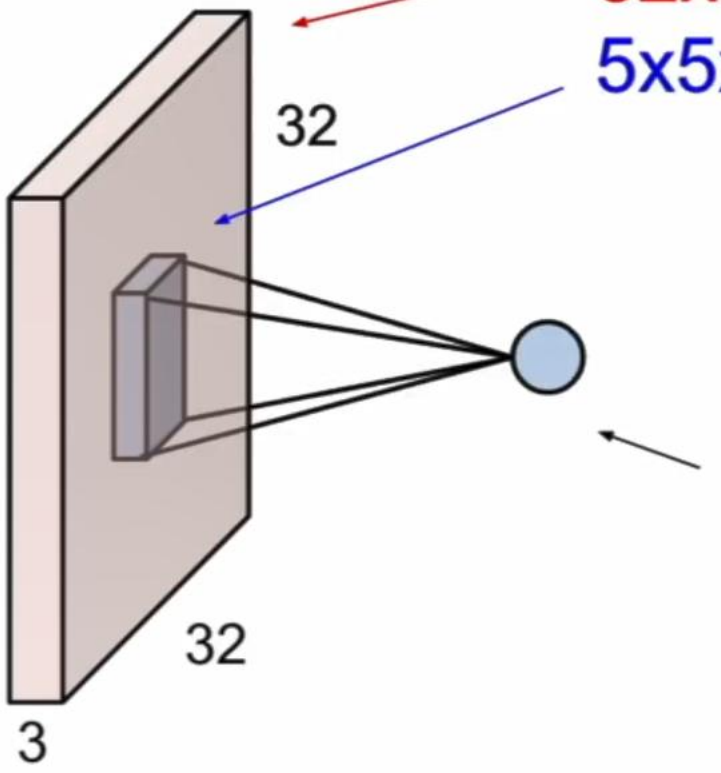
Ch-3 CNNs: Illustration

- Convolution Layer



Ch-3 CNNs: Illustration

- Convolution Layer

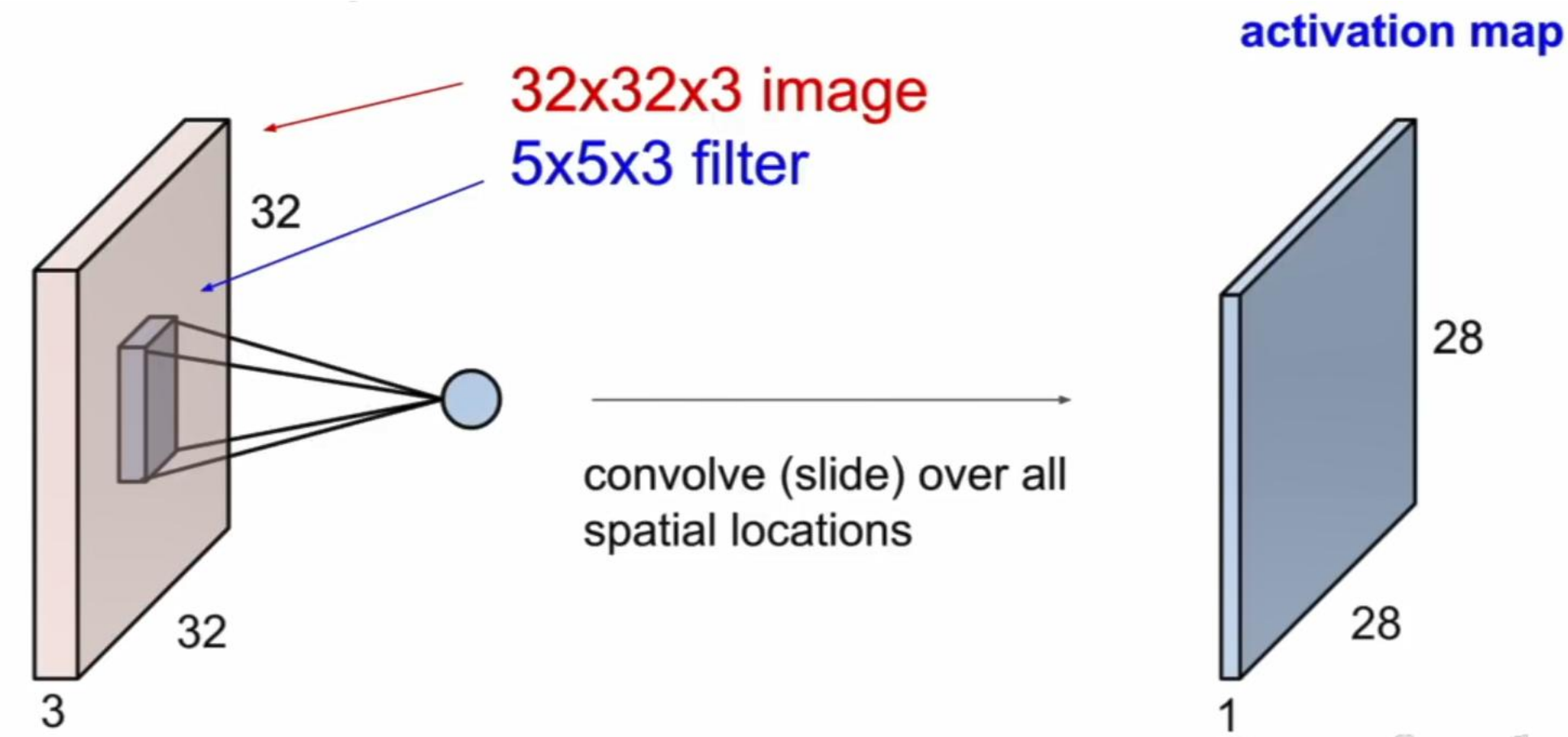


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

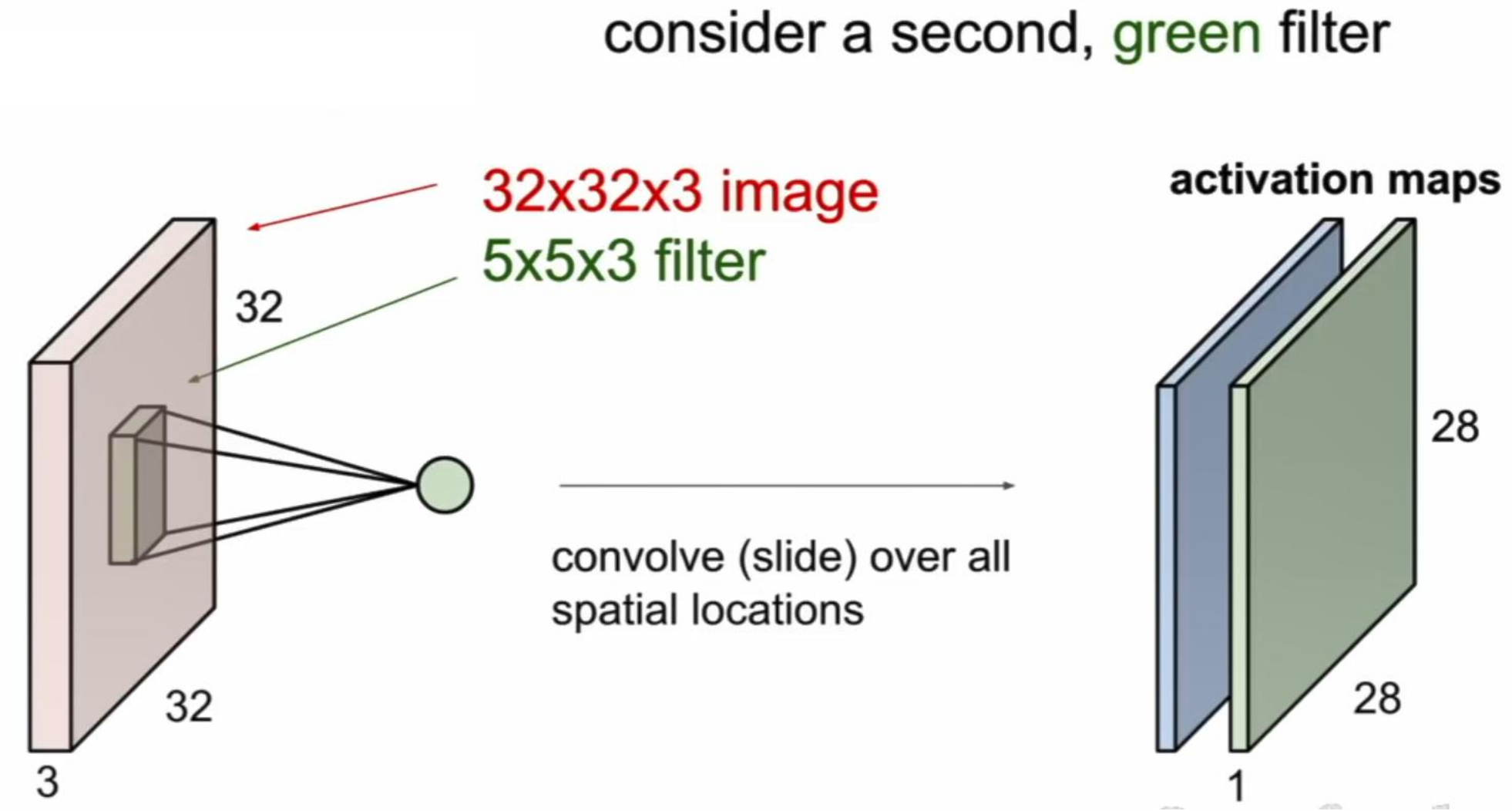
Ch-3 CNNs: Illustration

- Convolution Layer



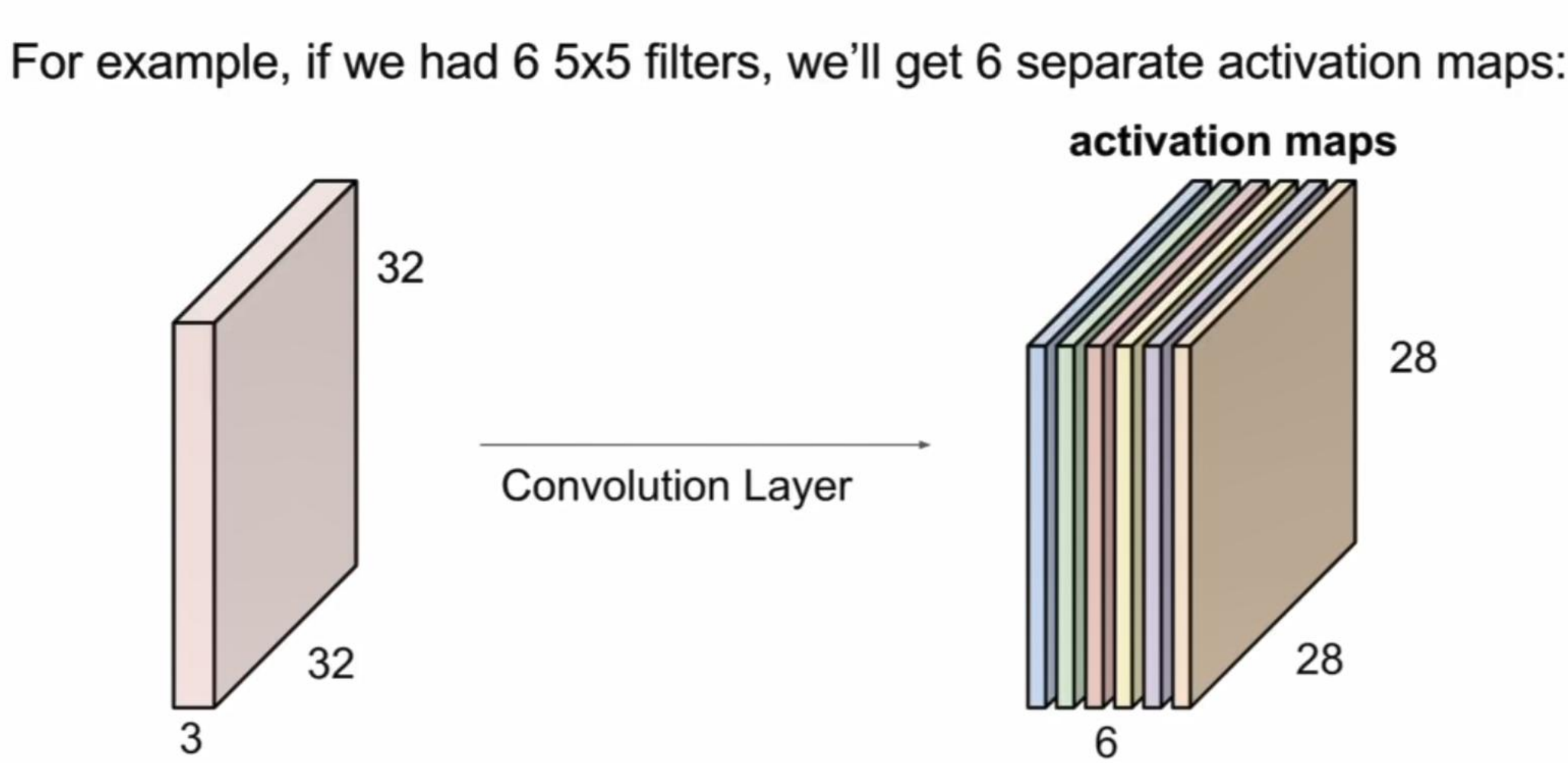
Ch-3 CNNs: Illustration

- Convolution Layer



Ch-3 CNNs: Illustration

- Convolution Layer

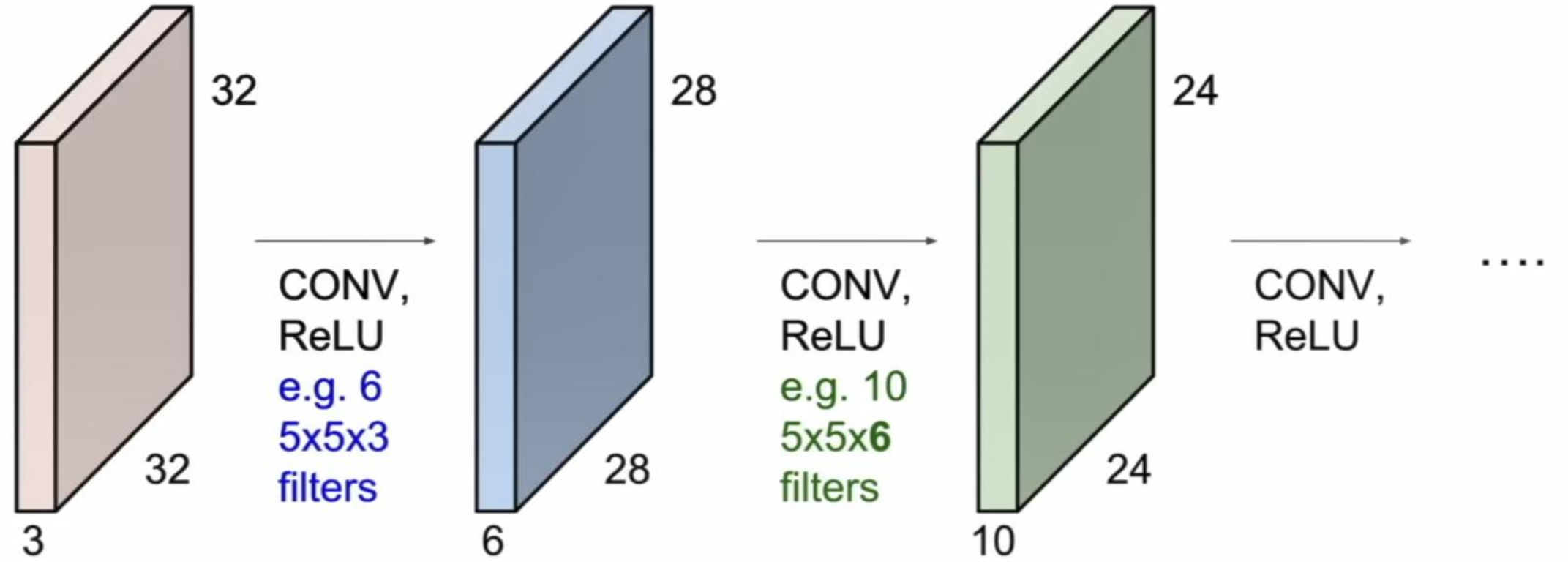


- We stack these up to get a "new image" of size 28x28x6

Ch-3 CNNs: Illustration

Convolution Layer

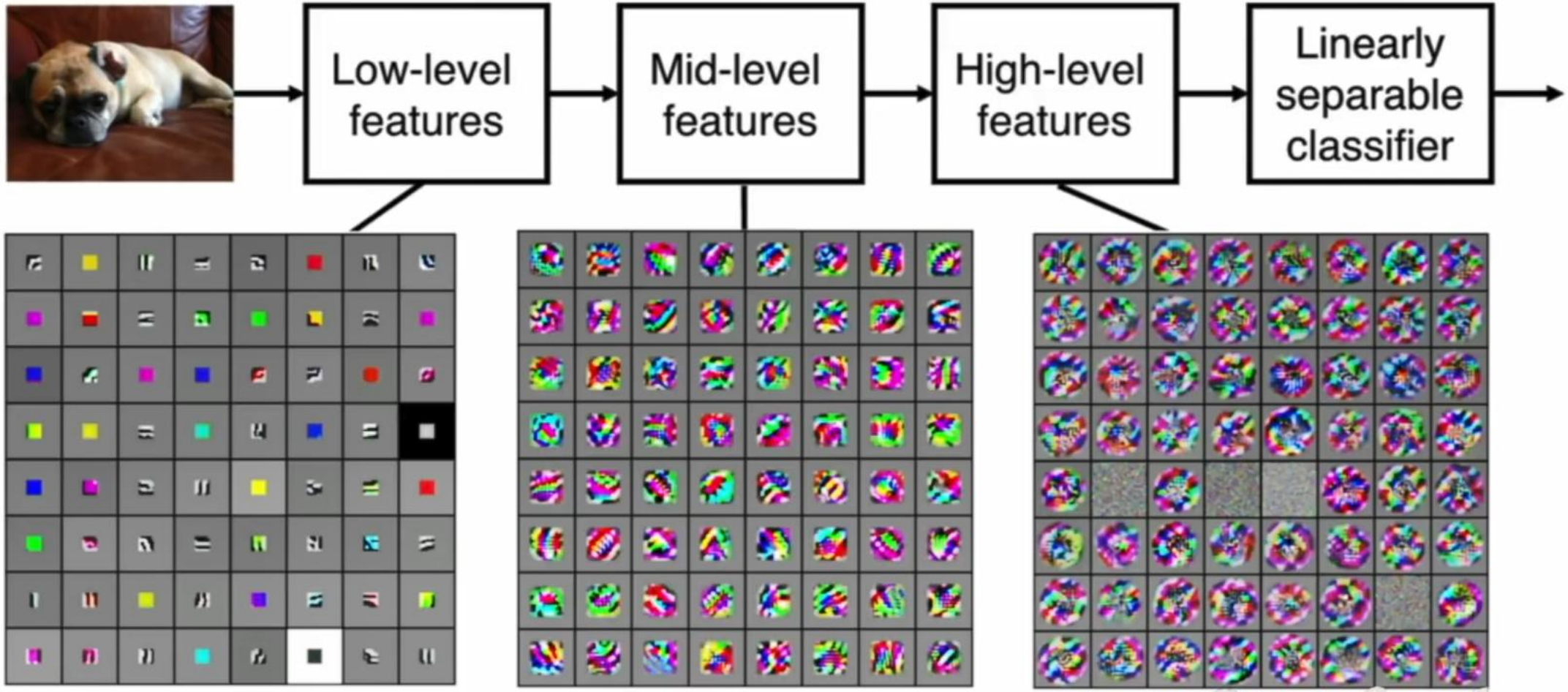
- ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.



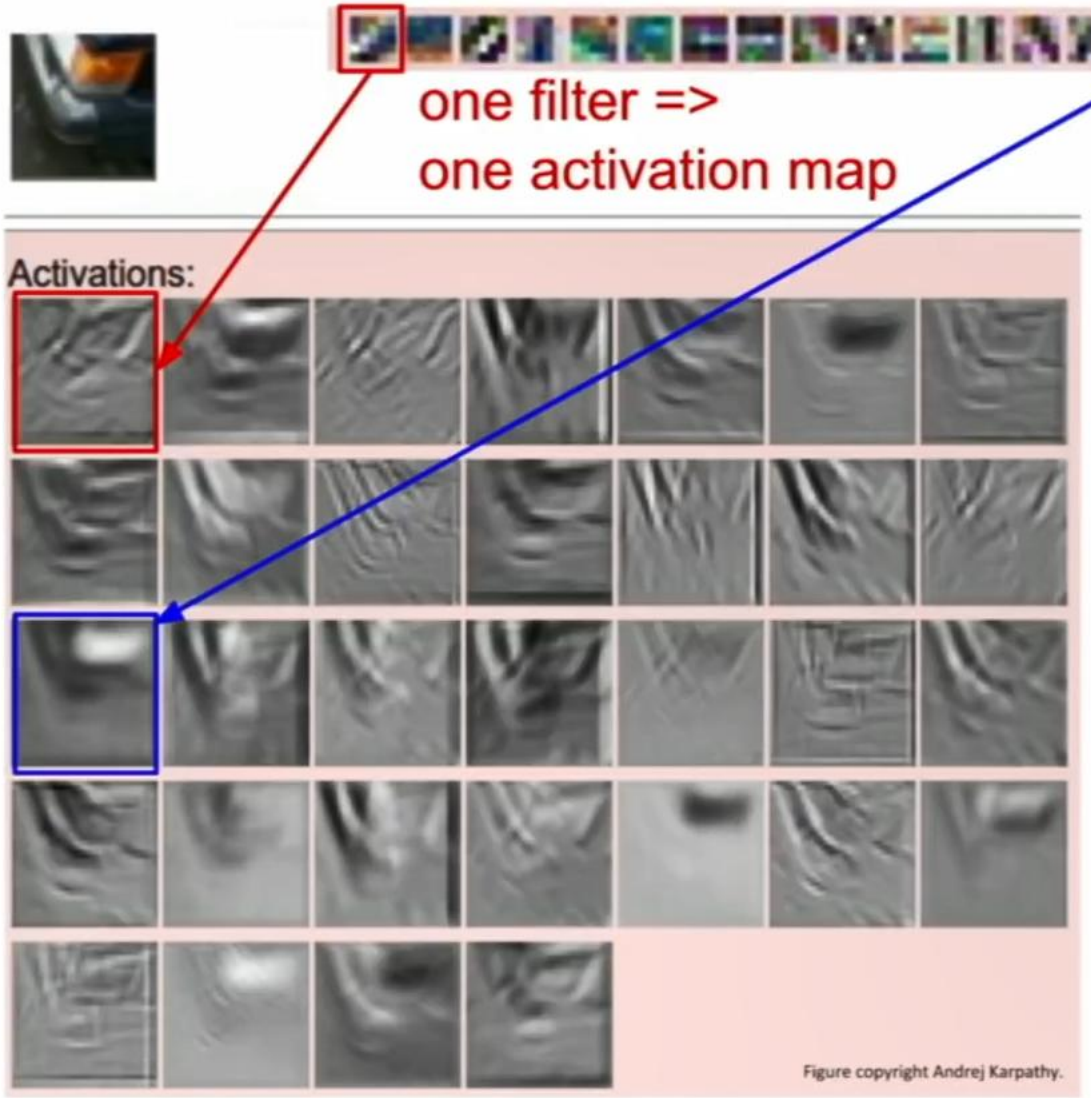
Ch-3 CNNs: Illustration

Convolution Layer

- Layers stacked together → hierarchical filters



Ch-3 CNNs: Illustration
Convolution Layer



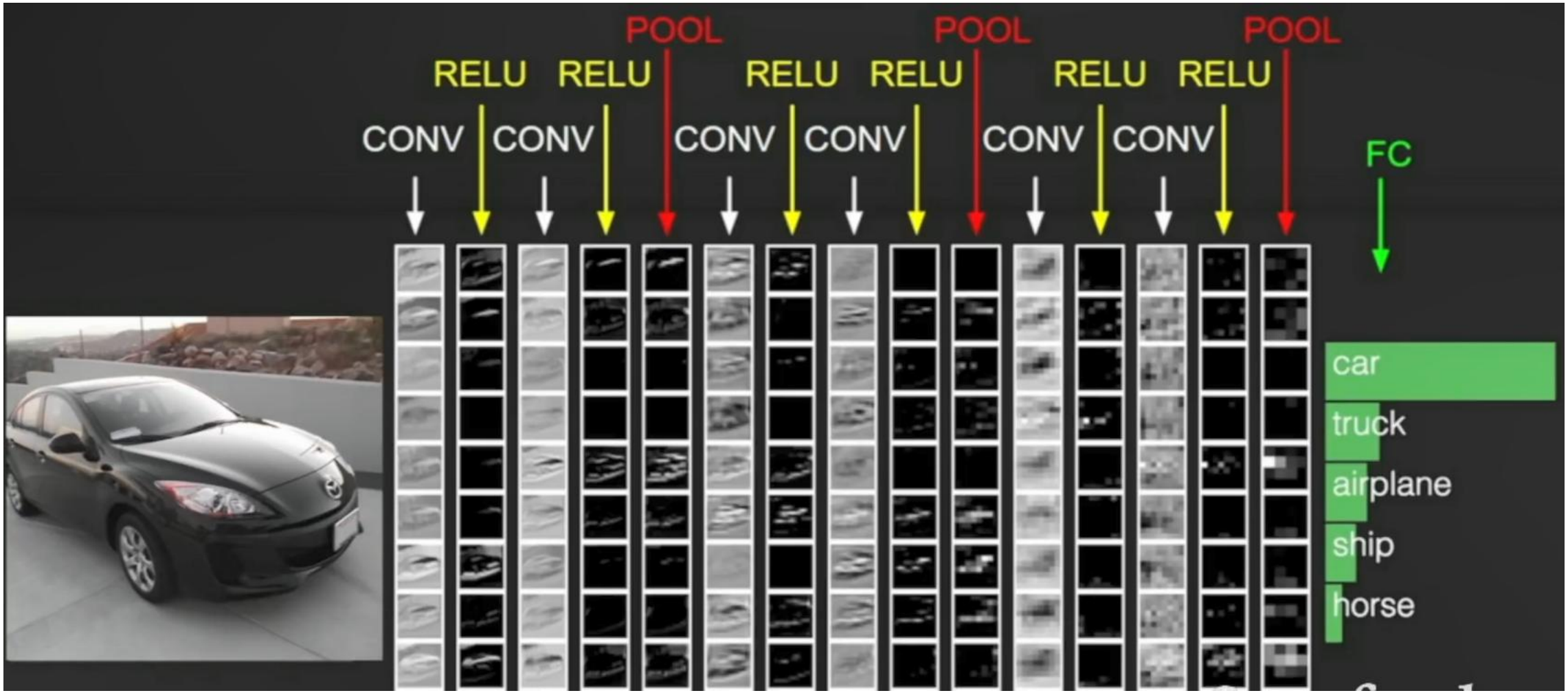
example 5x5 filters
(32 total)

We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

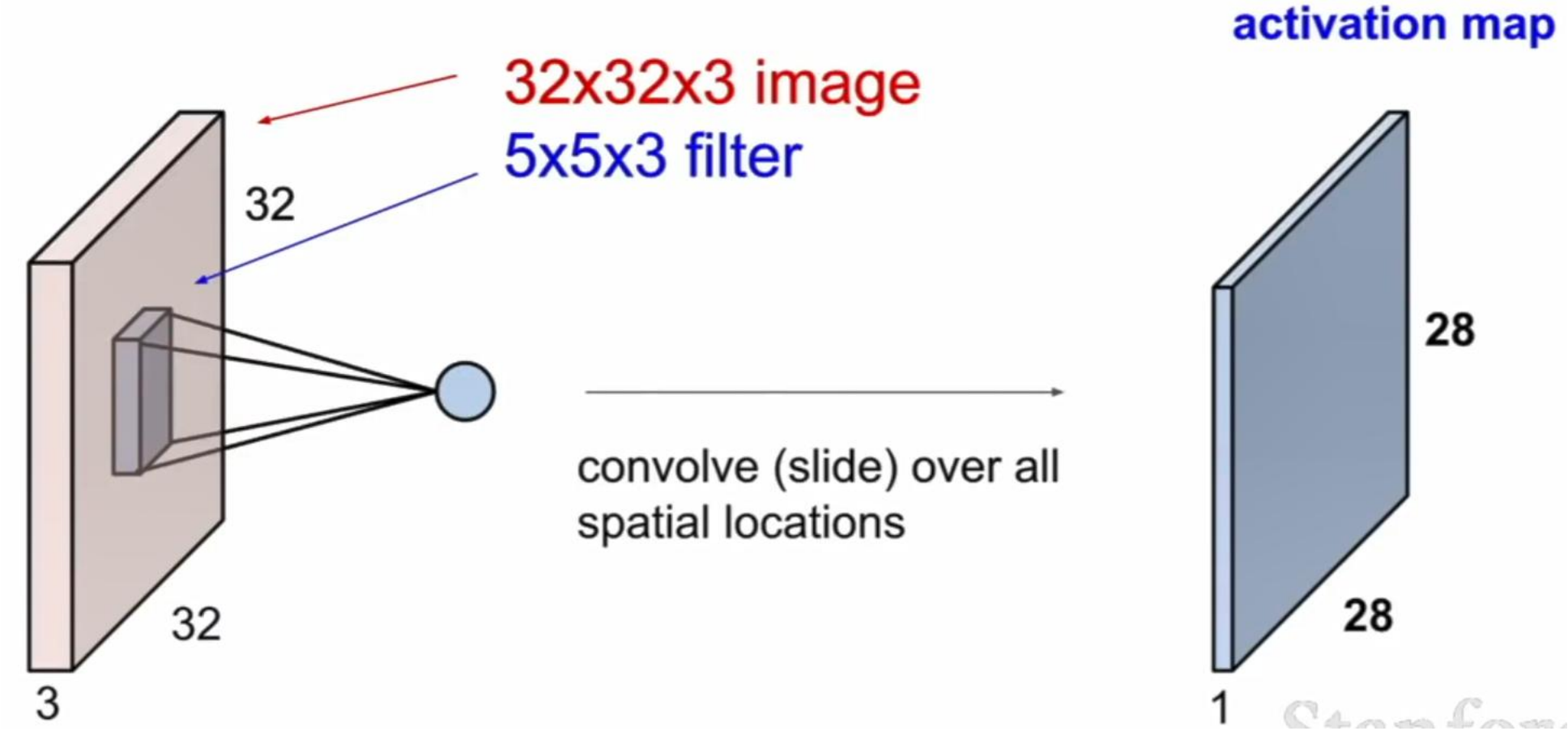
elementwise multiplication and sum of a filter and the signal (image)

Ch-3 CNNs: Illustration



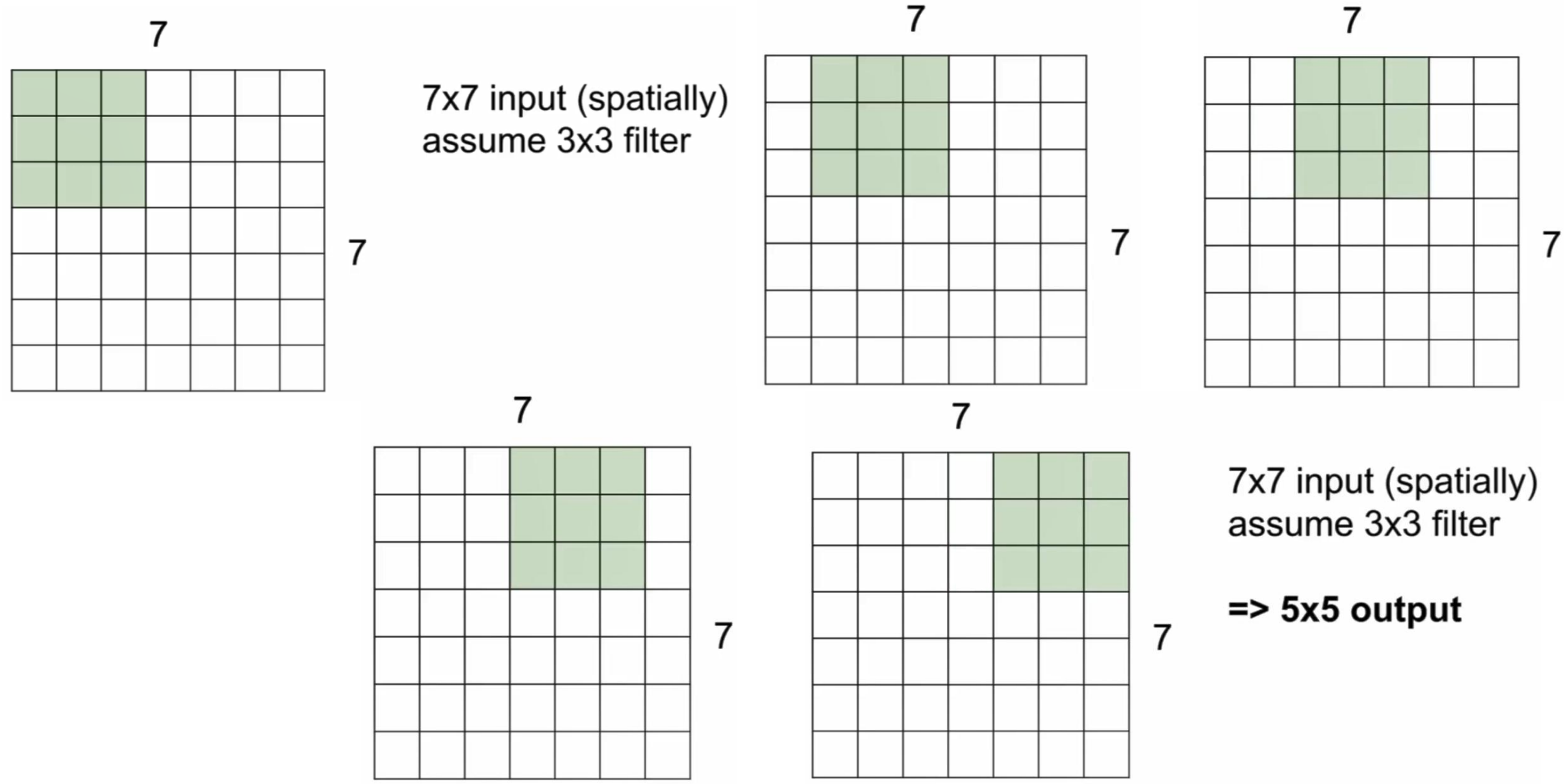
Ch-3 CNNs: Illustration

Examples: a closer look at spatial dimensions



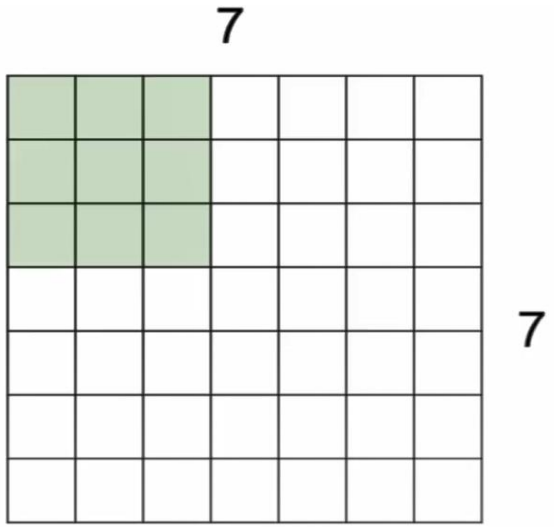
Ch-3 CNNs: Illustration

Examples: a closer look at spatial dimensions

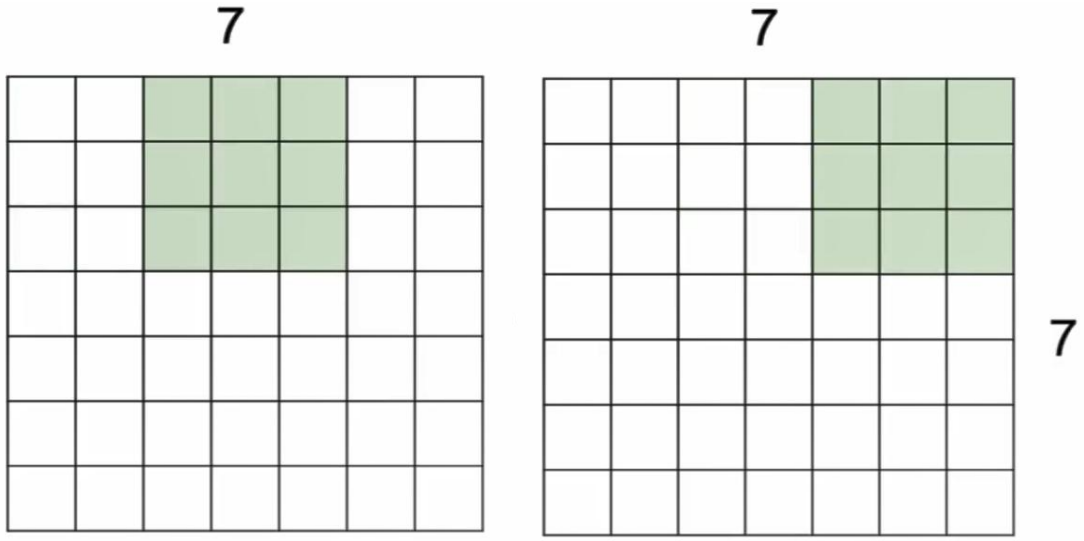


Ch-3 CNNs: Illustration

Examples: a closer look at spatial dimensions



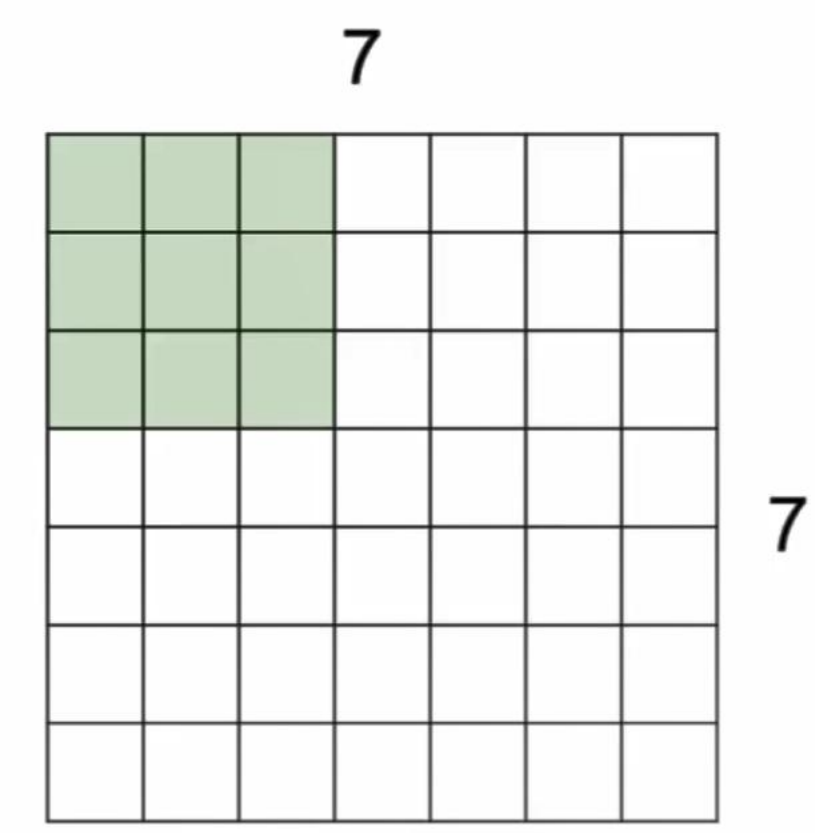
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Ch-3 CNNs: Illustration

Examples: a closer look at spatial dimensions

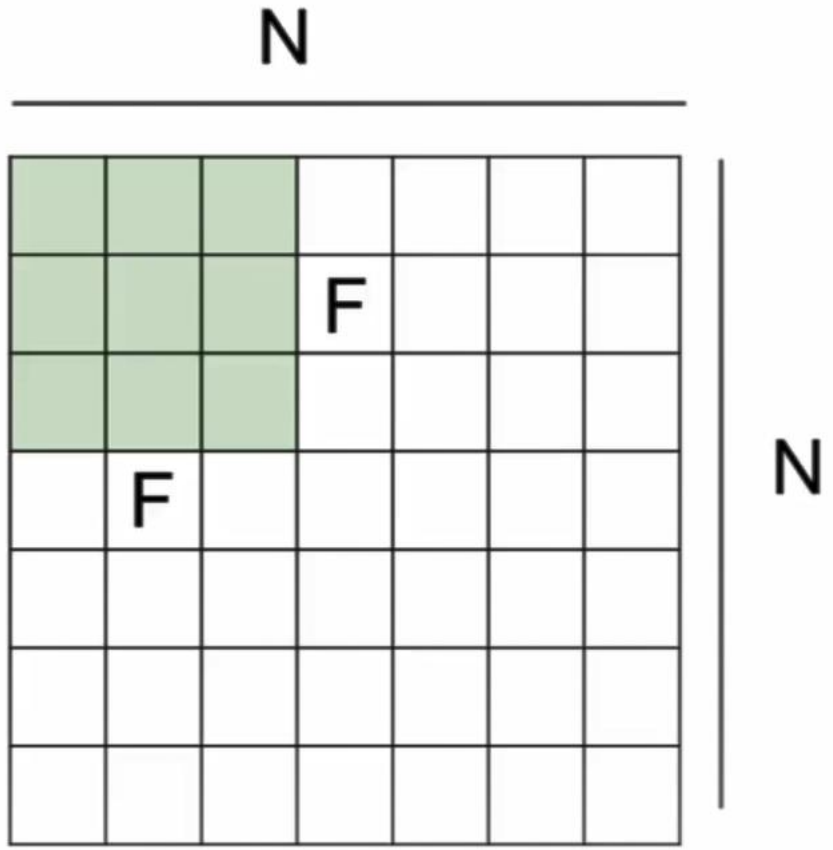


7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

Ch-3 CNNs: Illustration

Examples: a closer look at spatial dimensions



Output Size

(N-F/Stride) +1

e.g. N = 7, F = 3:

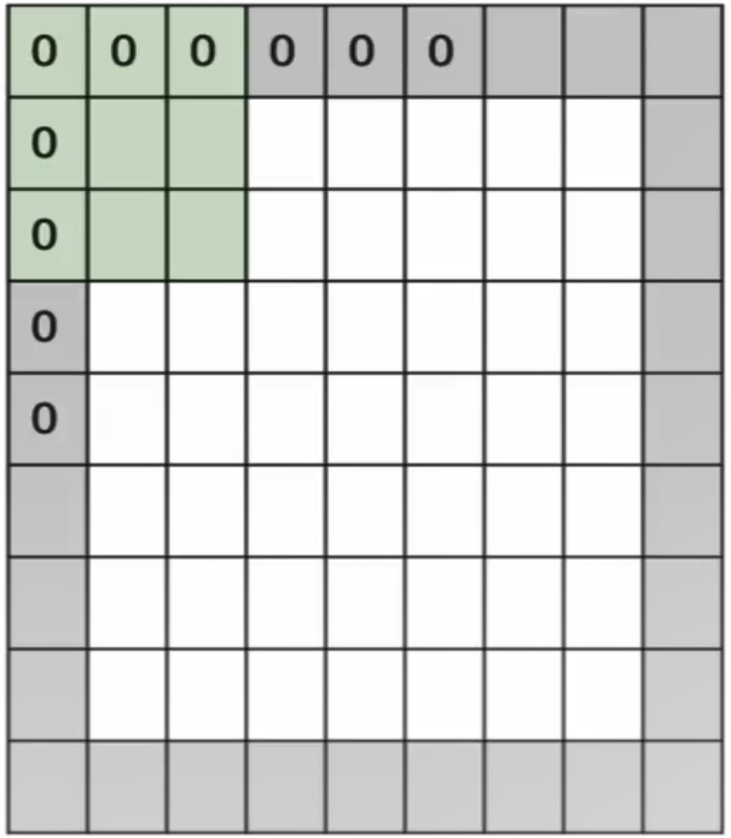
stride 1 => (7 - 3)/1 + 1 = 5

stride 2 => (7 - 3)/2 + 1 = 3

~~stride 3 => (7 - 3)/3 + 1 = 2.33 :\~~

Ch-3 CNNs: Illustration

- Examples: In practice, its common to zero-pad the border



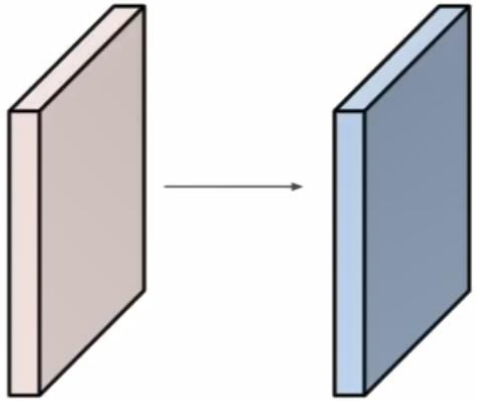
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output !

in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Ch-3 CNNs: Illustration

- Examples:



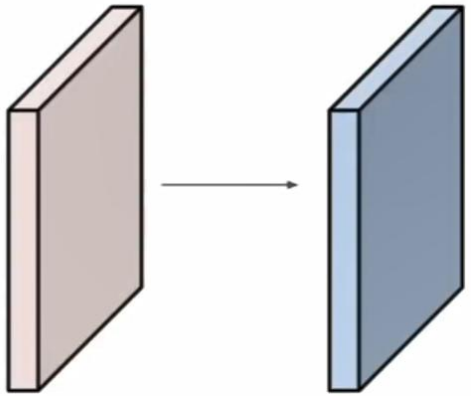
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Ch-3 CNNs: Illustration

- Examples:



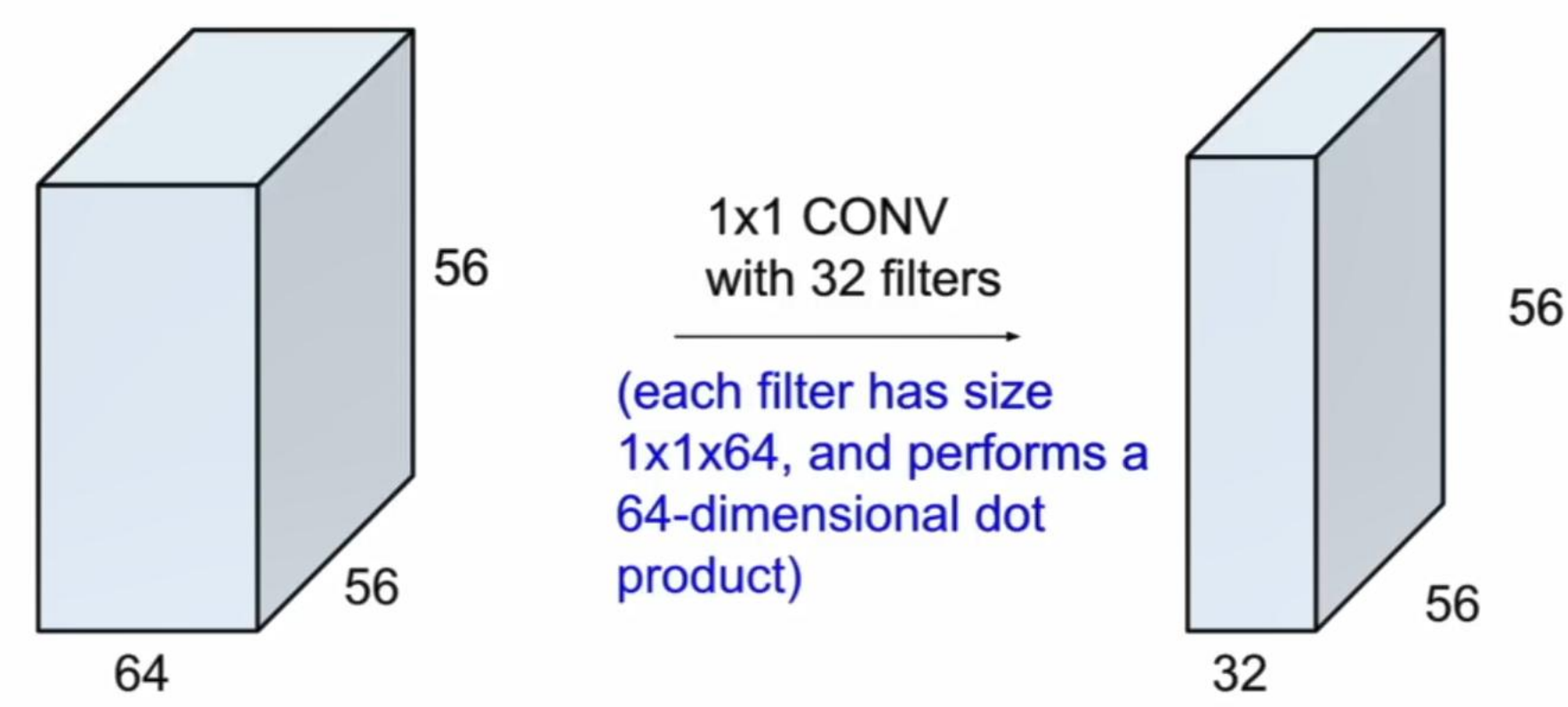
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Ch-3 CNNs: Illustration

- Does having a 1x1 convolution layer make sense?



Ch-3 CNNs: Illustration

- Summary of convolution layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Ch-3 CNNs: Illustration

- Summary of convolution layer: common settings

K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Ch-3 CNNs: Convolution and Pooling as an Infinitely Strong Prior

- Recall the concept of a prior probability distribution from Sec. 5.2. This is a probability distribution over the parameters of a model that encodes our beliefs about what models are reasonable, before we have seen any data.
- Priors can be considered weak or strong depending on how concentrated the probability density in the prior is.
  - A weak prior is a prior distribution with high entropy, such as a Gaussian distribution with high variance. Such a prior allows the data to move the parameters more or less freely.
  - A strong prior has very low entropy, such as a Gaussian distribution with low variance. Such a prior plays a more active role in determining where the parameters end up.
- An infinitely strong prior places zero probability on some parameters and says that these parameter values are completely forbidden, regardless of how much support the data gives to those values.

Ch-3 CNNs: Convolution and Pooling as an Infinitely Strong Prior

- We can imagine a convolutional net as being similar to a fully connected net, but with an infinitely strong prior over its weights.
  - This infinitely strong prior says that the weights for one hidden unit must be identical to the weights of its neighbor, but shifted in space.
  - The prior also says that the weights must be zero, except for in the small, spatially contiguous receptive field assigned to that hidden unit.
- Overall, we can think of the use of convolution as introducing an infinitely strong prior probability distribution over the parameters of a layer.
  - This prior says that the function the layer should learn contains only local interactions and is equivariant to translation.
- Likewise, the use of pooling is an infinitely strong prior that each unit should be invariant to small translations.

Ch-3 CNNs: Convolution and Pooling as an Infinitely Strong Prior

- Of course, implementing a convolutional net as a fully connected net with an infinitely strong prior would be extremely computationally wasteful.
- But thinking of a convolutional net as a fully connected net with an infinitely strong prior can give us some insights into how convolutional nets work.
  - One key insight is that convolution and pooling can cause underfitting. Like any prior, convolution and pooling are only useful when the assumptions made by the prior are reasonably accurate. If a task relies on preserving precise spatial information, then using pooling on all features can increase the training error.
  - Some convolutional network architectures (Szegedy et al., 2014a) are designed to use pooling on some channels but not on other channels, in order to get both highly invariant features and features that will not underfit when the translation invariance prior is incorrect.
  - When a task involves incorporating information from very distant locations in the input, then the prior imposed by convolution may be inappropriate.

Ch-3 CNNs: Convolution and Pooling as an Infinitely Strong Prior

- Another key insight from this view is that we should only compare convolutional models to other convolutional models in benchmarks of statistical learning performance.
- Models that do not use convolution would be able to learn even if we permuted all of the pixels in the image.
- For many image datasets, there are separate benchmarks for models that are permutation invariant and must discover the concept of topology via learning, and models that have the knowledge of spatial relationships hard-coded into them by their designer.

Ch-3 CNNs: Variants of the Basic Convolution Function

Zero Padding

- Three special cases of the zero-padding setting are worth mentioning.
  - Valid convolution: no zero-padding, the convolution kernel is only allowed to visit positions where the entire kernel is contained entirely within the image; valid convolution.
    - all pixels in the output are a function of the same number of pixels in the input, so the behavior of an output pixel is somewhat more regular.
    - size of the output shrinks at each layer. If the input image has width m and the kernel has width k, the output will be of width m-k+1.
    - Limits the number of conv layers that can be included in the network
  - Same convolution: enough zero-padding is added to keep the size of the output equal to the size of the input.
    - In this case, the network can contain as many convolutional layers as the available hardware can support, since the operation of convolution does not modify the architectural possibilities available to the next layer.
    - However, the input pixels near the border influence fewer output pixels than the input pixels near the center. This can make the border pixels somewhat underrepresented in the model. This motivates the other extreme case,

Ch-3 CNNs: Variants of the Basic Convolution Function
Zero Padding

- Three special cases of the zero-padding setting are worth mentioning.
  - Full convolution: enough zeroes are added for every pixel to be visited k times in each direction, resulting in an output image of width m+ k − 1.
  - In this case, the output pixels near the border are a function of fewer pixels than the output pixels near the center.
  - This can make it difficult to learn a single kernel that performs well at all positions in the convolutional feature map.
  - Usually the optimal amount of zero padding (in terms of test set classification accuracy) lies somewhere between “valid” and “same” convolution.

Ch-3 CNNs: Convolution Architecture Developments

- Covered through your assignment/mini-projects
  - LeNet-5
  - AlexNet [2012]
  - VGGNet [2014]
  - GoogleNet [2014]
  - ResNet [2015]
  - FractalNet (1016)
  - DenseNet (2017)
  - SqueezNet (2016)
  - EfficientNet (2019)
  - RegNet (2020)
  - NFNet (2021)

Ch-3 CNNs: Applications of CNNs

- Image Classification
  - *Description*: Identifying the category or class of an object within an image.
  - *Example*: Classifying images from the ImageNet dataset into thousands of different categories.
  - *Applications*: Autonomous vehicles, content moderation, medical imaging diagnostics.
- Object Detection
  - *Description*: Identifying and localizing objects within an image by drawing bounding boxes around them.
  - *Example*: Detecting pedestrians, cars, and traffic signs in autonomous driving.
  - *Applications*: Security surveillance, retail (customer tracking), robotics.

Ch-3 CNNs: Applications of CNNs

- Image Segmentation
  - *Description*: *Partitioning an image into different segments or regions, often on a pixel-level basis.*
  - *Example*: *Semantic segmentation in medical images, where each pixel is labeled according to the tissue type.*
  - *Applications*: *Medical imaging (tumor detection), satellite image analysis, augmented reality.*
- Facial Recognition
  - *Description*: Identifying or verifying a person based on their facial features.
  - *Example*: Unlocking smartphones using facial recognition.
  - *Applications*: Security systems, biometric authentication, social media tagging.

Ch-3 CNNs: Applications of CNNs

- Image Super-Resolution
  - *Description*: Enhancing the resolution of an image using deep learning techniques.
  - *Example*: Improving the quality of low-resolution images for medical diagnostics.
  - *Applications*: Medical imaging, satellite imagery, forensic analysis.
- Style Transfer
  - *Description*: Recreating an image in the style of another image.
  - *Example*: Applying the style of a famous painting to a photograph.
  - *Applications*: Art and entertainment, image editing software.

Ch-3 CNNs: Applications of CNNs

- Video Analysis
  - *Description*: Analyzing video content frame-by-frame for various purposes.
  - Example*: Action recognition in sports analytics or security footage.*
  - *Applications*: Video surveillance, autonomous vehicles, sports analytics.
- Medical Imaging
  - *Description*: Analyzing medical images such as X-rays, MRIs, and CT scans for diagnosis.
  - *Example*: Detecting pneumonia from chest X-rays or identifying tumors in MRI scans.
  - *Applications*: Diagnostic radiology, personalized medicine, telemedicine.

Ch-3 CNNs: Applications of CNNs

- Self-Driving Cars
  - Description: Perceiving and understanding the environment to make driving decisions.
  - Example: Object detection and lane-keeping systems.
  - Applications: Autonomous vehicles, advanced driver-assistance systems (ADAS).
- Natural Language Processing (NLP)
  - Description: CNNs are used in NLP tasks, typically for text classification or sentence modeling.
  - Example: Sentiment analysis, where CNNs are applied to sequences of words or characters.
  - Applications: Chatbots, sentiment analysis, language translation.

Ch-3 CNNs: Applications of CNNs

- Robotics and Automation
  - Description: Enhancing the visual perception systems of robots.
  - Example: Enabling robots to recognize and interact with objects in their environment.
  - Applications: Industrial automation, home robots, healthcare robots.
- Recommender Systems
  - Description: Using CNNs to analyze visual content (e.g., images) to improve recommendations.
  - Example: Recommending fashion products based on images a user has interacted with.
  - Applications: E-commerce, online advertising, content streaming platforms.

Ch-3 CNNs: Applications of CNNs

- Art Generation and Restoration
  - Description: Generating or restoring artwork using deep learning.
  - Example: Colorizing black-and-white photos or restoring old paintings.
  - Applications: Art restoration, entertainment, cultural heritage preservation.
- Augmented Reality (AR) and Virtual Reality (VR)
  - Description: Enhancing the interaction between the virtual and real world through visual data.
  - Example: Object recognition and placement in AR applications.
  - Applications: Gaming, retail, virtual try-ons.

Ch-3 CNNs: Applications of CNNs

- Speech Recognition
  - Description: CNNs can be used in combination with other architectures for recognizing speech patterns.
  - Example: Converting spoken language into text.
  - Applications: Virtual assistants, transcription services, language learning tools.
- Image Captioning
  - Description: Generating textual descriptions for images.
  - Example: Automatically describing the content of an image for accessibility purposes.
  - Applications: Assistive technology, content generation, social media platforms.

Ch-3 CNNs: Applications of CNNs

- Optical Character Recognition (OCR)
  - Description: Extracting text from images or scanned documents.
  - Example: Digitizing printed documents or recognizing handwritten text.
  - Applications: Document digitization, data entry automation, license plate recognition.

References

- Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, Ch-9. (text book)
- Michael Nielsen, Neural Networks and Deep Learning
- Fei Fei Li, Stanford University, lectures