Νευρο-Ασαφής Υπολογιστική
Neuro-Fuzzy Computing

Διδάσκων –
Δημήτριος Κατσαρός

@ Τμ. ΗΜΜΥ
Πανεπιστήμιο Θεσσαλίας

Διάλεξη 11η

Convolutional Neural Networks (CNNs)

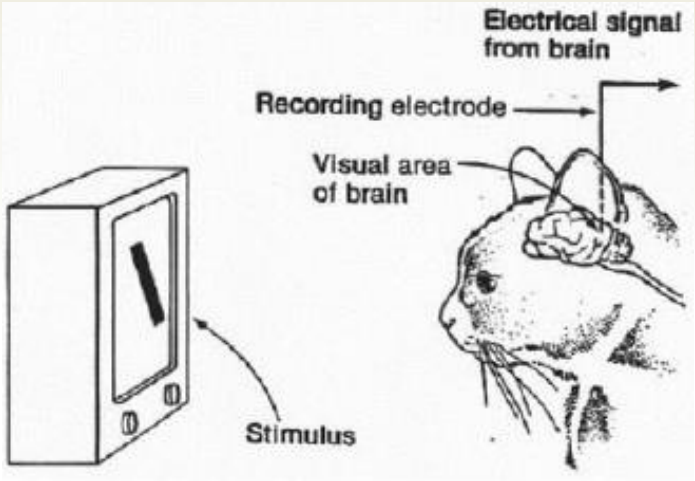
Introduction to Convolutional NN (CNNs)

- Convolutional neural networks are designed to work with grid-structured inputs, which have strong spatial dependencies in local regions of the grid
  - The most obvious example of grid-structured data is a 2-dimensional image
    - This type of data exhibits spatial dependencies, because adjacent spatial locations in an image often have similar color values of the individual pixels
    - An additional dimension captures the different colors, which creates a 3-dimensional input volume
  - Therefore, the features in a convolutional neural network have dependencies among one another based on spatial distances
  - Other forms of sequential data like text, time-series, and sequences can also be considered special cases of grid-structured data with various types of relationships among adjacent items

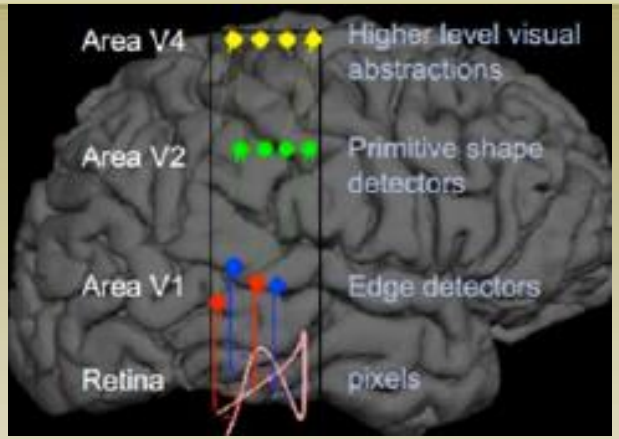
The neuroscientific basis of CNNs

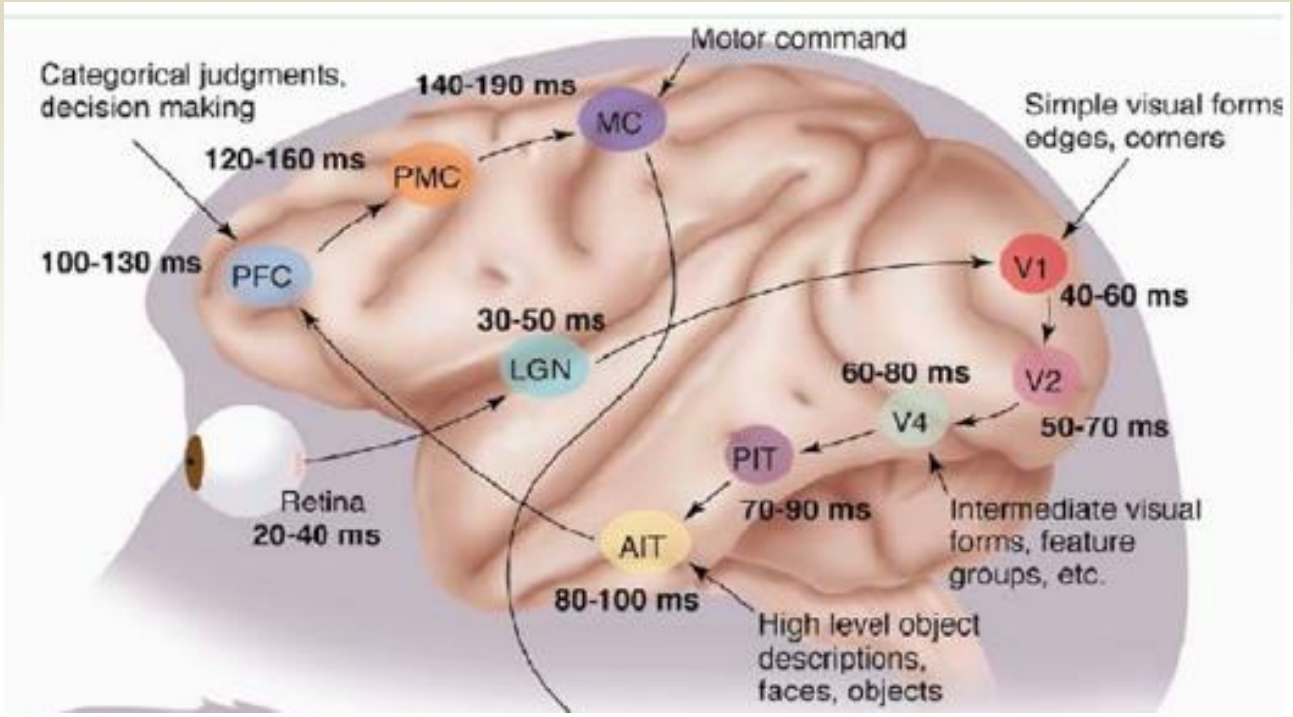
- The history of convolutional networks begins with neuroscientific experiments long before the relevant computational models were developed
- Neurophysiologists David Hubel and Torsten Wiesel ('59, '62, '68) collaborated for several years to determine many of the most basic facts about how the mammalian vision system works
  - Their accomplishments were eventually recognized with a Nobel prize (1981)
    - https://www.nobelprize.org/prizes/medicine/1981/summary/
- Their findings that have had the greatest influence on contemporary deep learning models were based on recording the activity of individual neurons in cats
- They observed how neurons in the cat's brain responded to images projected in precise locations on a screen in front of the cat
- Their great discovery was that neurons in the early visual system responded most strongly to very specific patterns of light, such as precisely oriented bars, but responded hardly at all to other patterns

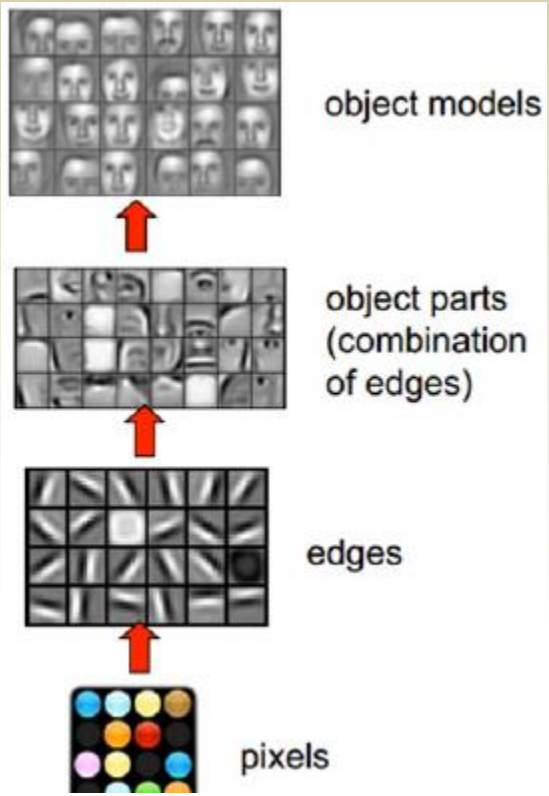
David H. Hubel and Torsten Wiesel



The neuroscientific basis of CNNs







The neuroscientific basis of CNNs

- In this simplified view, we focus on a part of the brain called V1, also known as the ***primary visual cortex***
- V1 is the first area of the brain that begins to perform significantly advanced processing of visual input
- In this cartoon view, images are formed by light arriving in the eye and stimulating the retina, the light-sensitive tissue in the back of the eye
- The neurons in the retina perform some simple preprocessing of the image but do not substantially alter the way it is represented
- The image then passes through the optic nerve and a brain region called the ***Lateral Geniculate Nucleus*** (***LGN***)
- The main role, as far as we are concerned here, of both of these anatomical regions is primarily just to carry the signal from the eye to V1, which is located at the back of the head

The neuroscientific basis of CNNs

- A convolutional network layer is designed to capture three properties of V1:

1. V1 is arranged in a spatial map. It actually has a two-dimensional structure mirroring the structure of the image in the retina
   - For example, light arriving at the lower half of the retina affects only the corresponding half of V1. Convolutional networks capture this property by having their features defined in terms of two dimensional maps
2. V1 contains many **simple cells**. A simple cell's activity can to some extent be characterized by a linear function of the image in a small, spatially localized *receptive field*. The detector units of a convolutional network are designed to emulate these properties of simple cells

The neuroscientific basis of CNNs

- A convolutional network layer is designed to capture three properties of V1 (cont'd):

3. V1 also contains many **complex cells**. These cells respond to features that are similar to those detected by simple cells, but complex cells are invariant to small shifts in the position of the feature
   - This inspires the pooling units of convolutional networks.

   Complex cells are also invariant to some changes in lighting that cannot be captured simply by pooling over spatial locations
   - These invariances have inspired some of the cross-channel pooling strategies in convolutional networks, such as maxout units

The neuroscientific basis of CNNs

- Though we know the most about V1, it is generally believed that the same basic principles apply to other areas of the visual system
- In our cartoon view of the visual system, the basic strategy of detection followed by pooling is repeatedly applied as we move deeper into the brain
- As we pass through multiple anatomical layers of the brain, we eventually find cells that respond to some specific concept and are invariant to many transformations of the input
- These cells have been nicknamed "grandmother cells"—the idea is that a person could have a neuron that activates when seeing an image of their grandmother, regardless of whether she appears in the left or right side of the image, whether the image is a close-up of her face or zoomed out shot of her entire body, whether she is brightly lit, or in shadow, etc.
- These grandmother cells have been shown to actually exist in the human brain, in a region called the ***Medial Temporal Lobe***

The neuroscientific basis of CNNs

- These medial temporal lobe neurons are somewhat more general than modern convolutional networks, which would not automatically generalize to identifying a person or object when reading its name
- The closest analog to a convolutional network's last layer of features is a brain area called the ***Inferotemporal Cortex*** (***IT***)
- When viewing an object, information flows from the retina, through the LGN, to V1, then onward to V2, then V4, then IT
- This happens within the first 100ms of glimpsing an object
- If a person is allowed to continue looking at the object for more time, then information will begin to flow backwards as the brain uses top-down feedback to update the activations in the lower level brain areas
- However, if we interrupt the person's gaze, and observe only the firing rates that result from the first 100ms of mostly feedforward activation, then IT proves to be very similar to a convolutional network
  - Convolutional networks can predict IT firing rates, and also perform very similarly to (time limited) humans on object recognition task

The neuroscientific basis of CNNs

- That being said, there are many differences between convolutional networks and the mammalian vision system
- Some of these differences are not yet known, because many basic questions about how the mammalian vision system works remain unanswered
- As a brief list:
  1. The human eye is mostly very low resolution, except for a tiny patch called the fovea. The fovea only observes an area about the size of a thumbnail held at arms length. Though we feel as if we can see an entire scene in high resolution, this is an illusion created by the subconscious part of our brain, as it stitches together several glimpses of small areas. Most convolutional networks actually receive large full resolution photographs as input. The human brain makes several eye movements called saccades to glimpse the most visually salient or task-relevant parts of a scene. Incorporating similar attention mechanisms into deep learning models is an active research direction. In the context of deep learning, attention mechanisms have been most successful for natural language processing, as described in section 12.4.5.1. Several visual models with foveation mechanisms have been developed but so far have not become the dominant approach

The neuroscientific basis of CNNs

- As a brief list (cont'd):
  2. The human visual system is integrated with many other senses, such as hearing, and factors like our moods and thoughts. Convolutional networks are purely visual
  3. The human visual system does much more than just recognize objects. It is able to understand entire scenes including many objects and relationships between objects, and processes rich 3-D geometric information needed for our bodies to interface with the world. Convolutional networks have been applied to some of these problems but these applications are in their infancy
  4. Even simple brain areas like V1 are heavily impacted by feedback from higher levels. Feedback has been explored extensively in neural network models but has not yet been shown to offer a compelling improvement
  5. While feed-forward IT firing rates capture much of the same information as convolutional network features, it is not clear how similar the intermediate computations are. The brain probably uses very different activation and pooling functions. An individual neuron's activation probably is not well-characterized by a single linear filter response. A recent model of V1 involves multiple quadratic filters for each neuron (Rust et al., 2005). Indeed our cartoon picture of "simple cells" and "complex cells" might create a nonexistent distinction; simple cells and complex cells might both be the same kind of cell but with their "parameters" enabling a continuum of behaviors ranging from what we call "simple" to what we call "complex"

The neuroscientific basis of CNNs

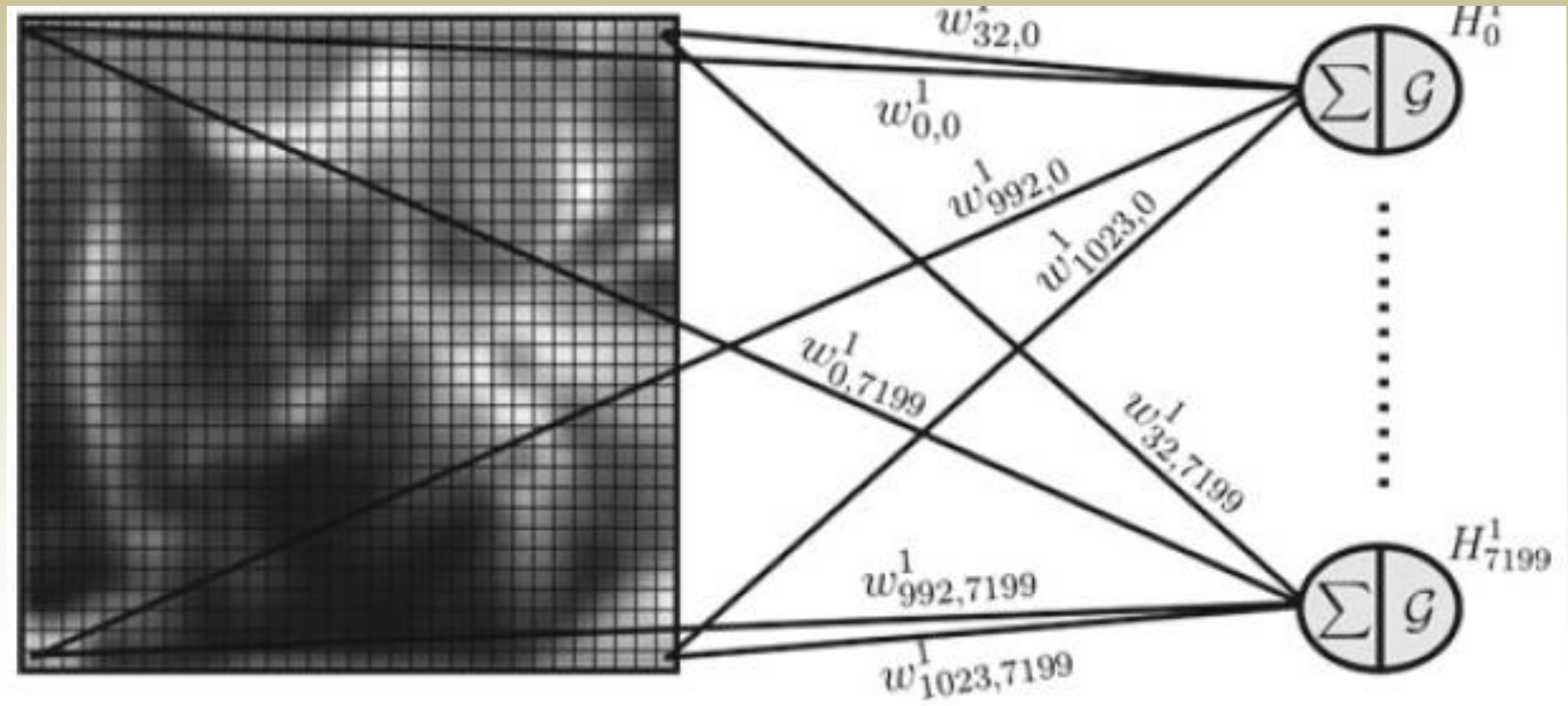
- It is also worth mentioning that neuroscience has told us relatively little about how to train convolutional networks
  - Model structures with parameter sharing across multiple spatial locations date back to early connectionist models of vision, but these models did not use the modern back-propagation algorithm and gradient descent. For example, the Neocognitron (Fukushima, 1980) incorporated most of the model architecture design elements of the modern convolutional network but relied on a layer-wise unsupervised clustering algorithm

Planes Don't Flap Wings to Fly

[Engineering success may start with biological inspiration, but then take a totally different path]

Motivational example (1/5)

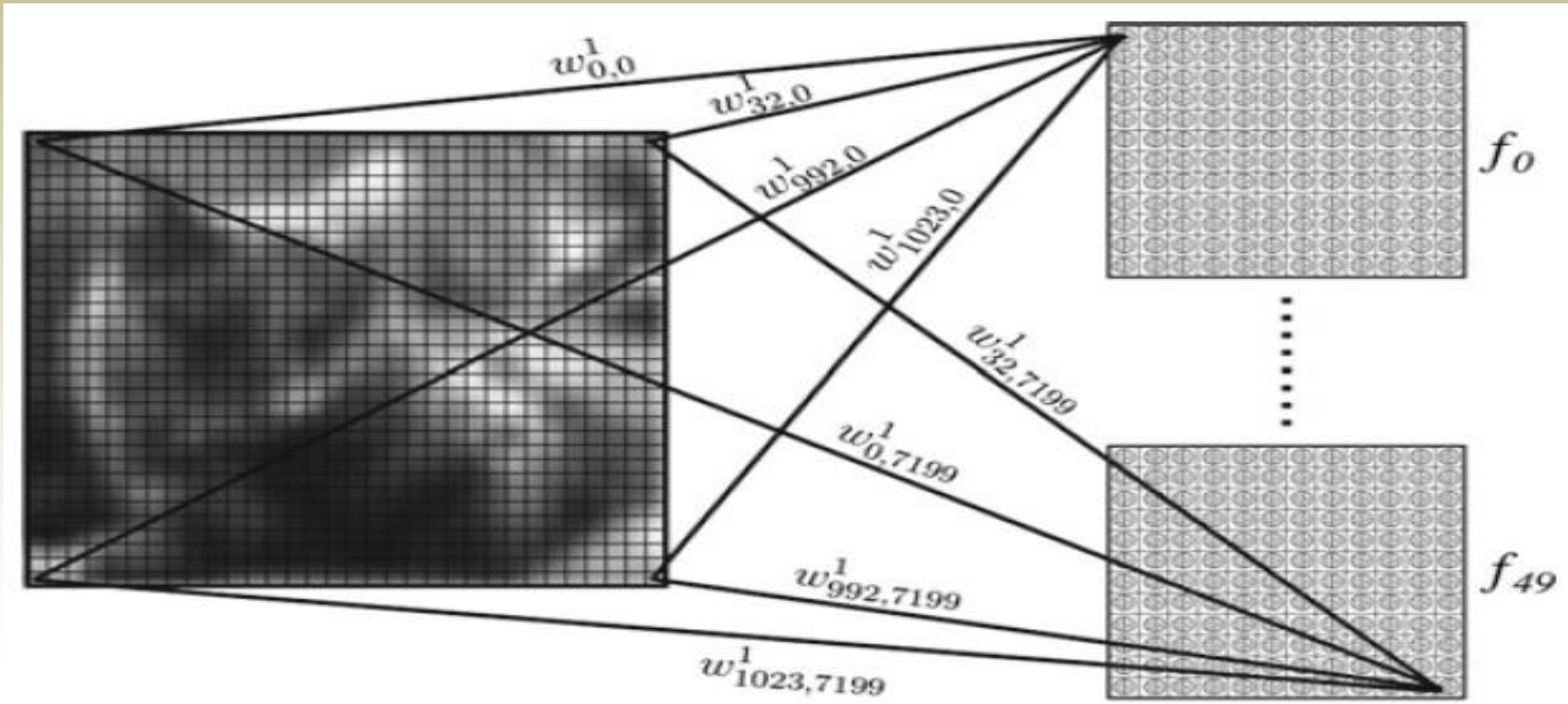
Assumptions: (a) image composed by 32 x 32 pixels, (b) 7200 hidden layer neurons, (c) fully connected each pixel with each hidden layer neuron



32 x 32 x 7200 = 7,372,800 parameters

Motivational example (2/5)

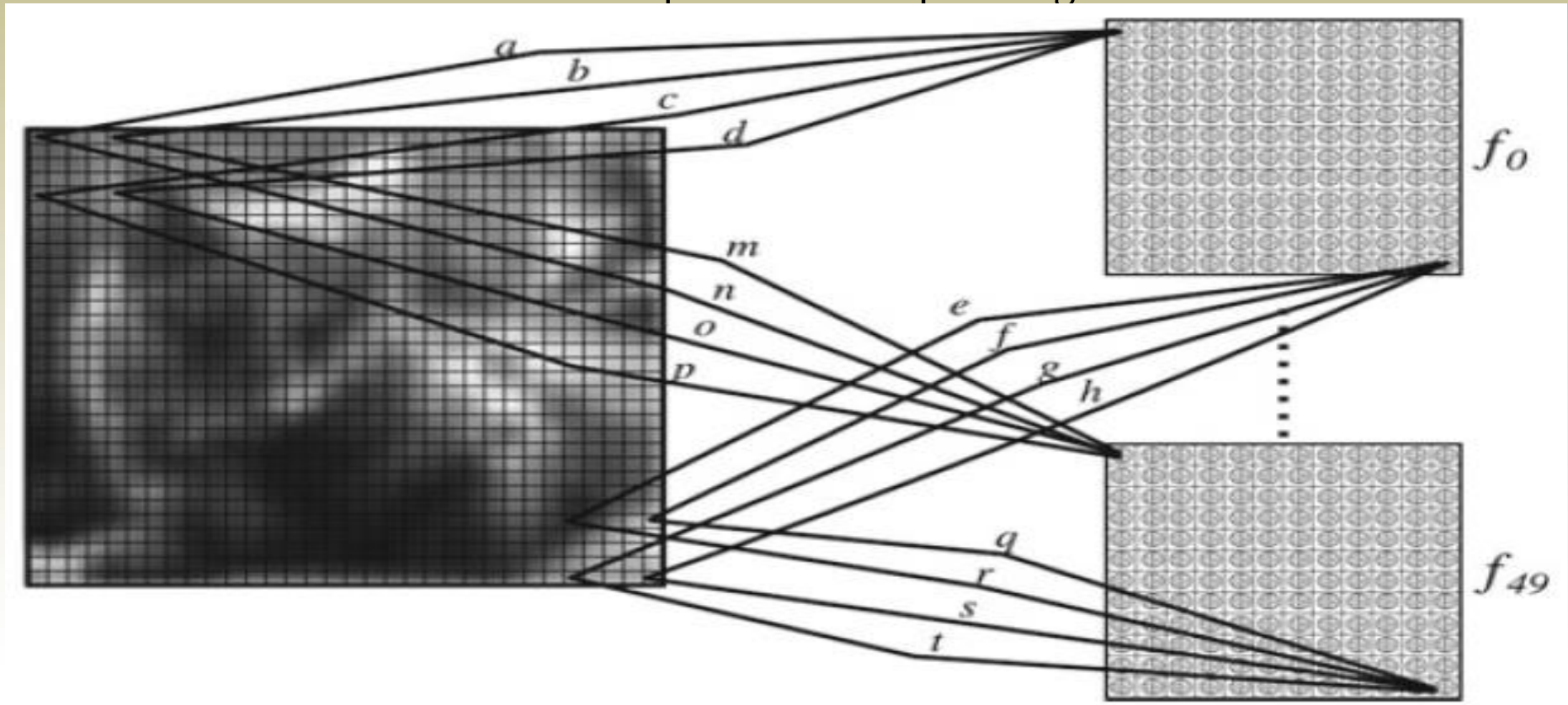
Organize neurons into 50 blocks composed by 12 x 12 neurons



Still 32 x 32 x 7200 = 7,372,800 parameters; Can exploit pixel geometry/correlation

Motivational example (3/5)

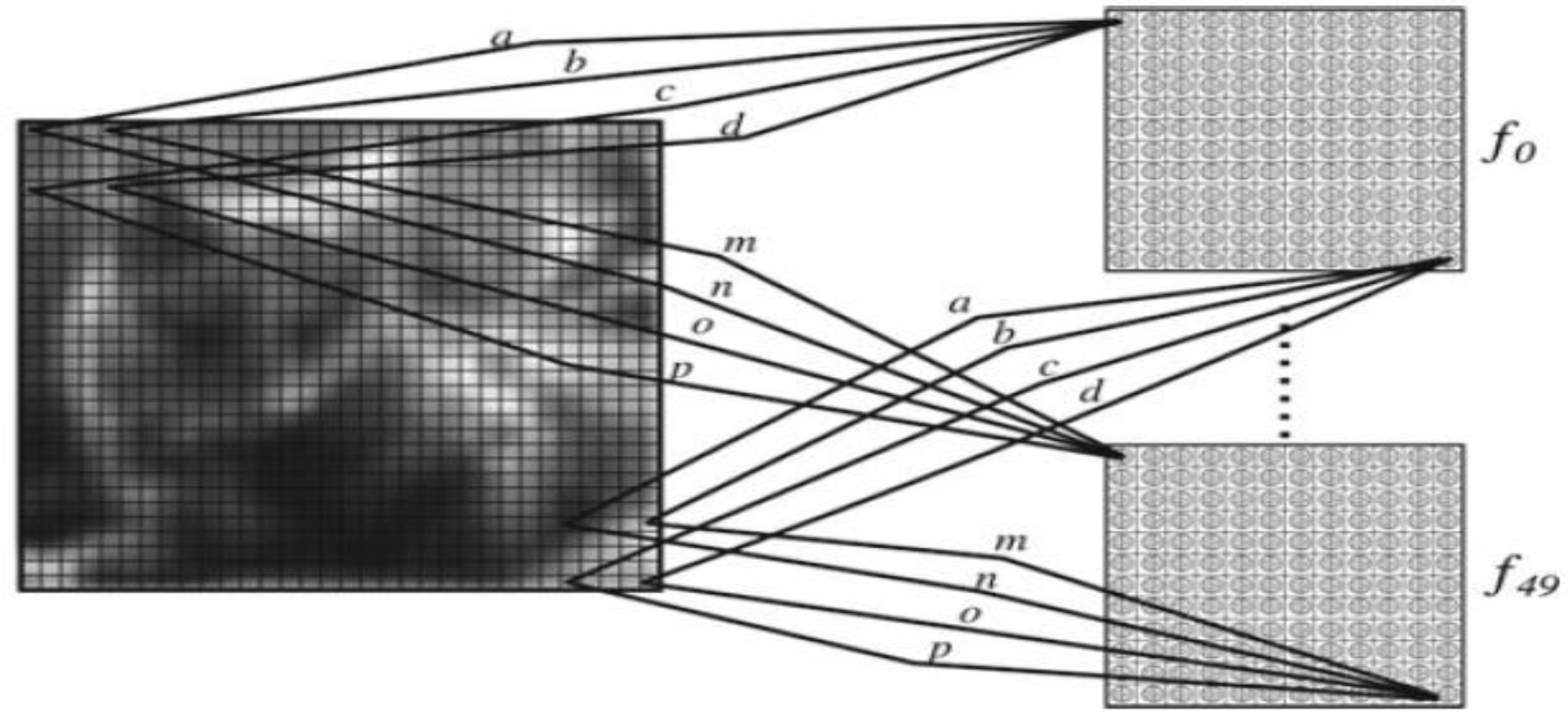
Correlation between far pixels is very low. Connect each neuron in each block to a 5×5 region in the image. Neurons in a block cover all the input image and extract information for each 5 × 5 patch in the input image



(5 x 5) x 50 x 12 x 12 = 180,000 parameters; 97.5% reduction

Motivational example (4/5)

Reduce number of weights by *weight sharing*



(5 x 5) x 50 = 1,250 parameters; 99.98% reduction (w.r.t. the fully connected layer)

Motivational example (5/5)

Denoting the neuron (*p*,*q*) in block *l* in the previous slide's figure by $f^{l}_{p,q}$, the output of this neuron is given by:

$$f^{l}_{p,q} = (\mathcal{G})\Big(\sum_{i=0}^{4}\sum_{j=0}^{4} = img(p+i, q+j)w^{l}_{p,q}\Big)$$

where $w^{l}_{p,q}$ shows the weight (*a*,*b*) in block I and p,q=0,1,…,11. Here *a*,*b* vary between 0 and 4 since each neuron is connected to a 5 × 5 region. The output of each block will have the same size as its block. Hence, in this example, the output of each block will be a 12 × 12 matrix. With this formulation and denoting the output matrix of l^{th} with $\mathbf{f}^{l}$, this matrix can be obtained by computing:

$$\mathbf{f}^{l}_{p,q} = (\mathcal{G})\Big(\sum_{i=0}^{4}\sum_{j=0}^{4} = img(p+i, q+j)w^{l}_{p,q}\Big)$$

The above equation is exactly analogous to convolving the 5×5 filter *w* with the input image. As the result, output of the l^{th} block is obtained by convolving the filter *w* on the input image

CNNs in the ILSVRC contest

- A factor that has played an important role in increasing the prominence of convolutional neural networks has been the annual ImageNet competition (also referred to as "ImageNet Large Scale Visual Recognition Challenge [ILSVRC]"). The ILSVRC competition uses the ImageNet data set
- Convolutional neural networks have been consistent winners of this contest since 2012
  - In fact, the dominance of convolutional neural networks for image classification is so well recognized today that almost all entries in recent editions of this contest have been convolutional neural networks. One of the earliest methods that achieved success in the 2012 ImageNet competition by a large margin was AlexNet
- Furthermore, the improvements in accuracy have been so extraordinarily large in the last few years that it has changed the landscape of research in the area
  - In spite of the fact that the vast majority of eye-catching performance gains have occurred from 2012 to 2015, the architectural differences between recent winners and some of the earliest convolutional neural networks are rather small at least at a conceptual level. Nevertheless, small details seem to matter a lot when working with almost all types of neural networks

Generic observations about CNNs

- The secret to the success of any neural architecture lies in tailoring the structure of the network with a semantic understanding of the domain at hand
- CNNs are heavily based on this principle, because they use sparse connections with a high-level of parameter-sharing in a domain-sensitive way
  - In other words, not all states in a particular layer are connected to those in the previous layer in an indiscriminate way. Rather, the value of a feature in a particular layer is connected only to a local spatial region in the previous layer with a consistent set of shared parameters across the full spatial footprint of the image
- A significant level of domain-aware regularization is also available in recurrent neural networks, which share the parameters from different temporal periods
  - This sharing is based on the assumption that temporal dependencies remain invariant with time
  - Recurrent neural networks are based on intuitive understanding of temporal relationships, whereas convolutional neural networks are based on an intuitive understanding of spatial relationships

Basic structure of CNNs

- In convolutional neural networks, the states in each layer are arranged according to a spatial grid structure. These spatial relationships are inherited from one layer to the next because each feature value is based on a small local spatial region in the previous layer
- It is important to maintain these spatial relationships among the grid cells, because the convolution operation and the transformation to the next layer is critically dependent on these relationships
- Each layer in the convolutional network is a 3-dimensional grid structure, which has a ***height***, ***width***, and ***depth***
  - The depth of a layer in a convolutional neural network should not be confused with the depth of the network itself
  - The word "depth" (when used in the context of a single layer) refers to the number of channels in each layer, such as the number of primary color channels (e.g., blue, green, and red) in the input image or the number of feature maps in the hidden layers

Basic structure of CNNs

- The convolutional neural network functions much like a traditional feed-forward neural network, except that the operations in its layers are spatially organized with sparse (and carefully designed) connections between layers
- The three types of layers that are commonly present in a convolutional neural network are
  - ***convolution,***
  - ***pooling***
  - ***ReLU***
    - The ReLU activation is no different from a traditional neural network.
  - In addition, a final ***set of layers is often fully connected*** and maps in an application-specific way to a set of output nodes
- In the following, we will describe each of the different types of operations and layers

Basic structure of CNNs

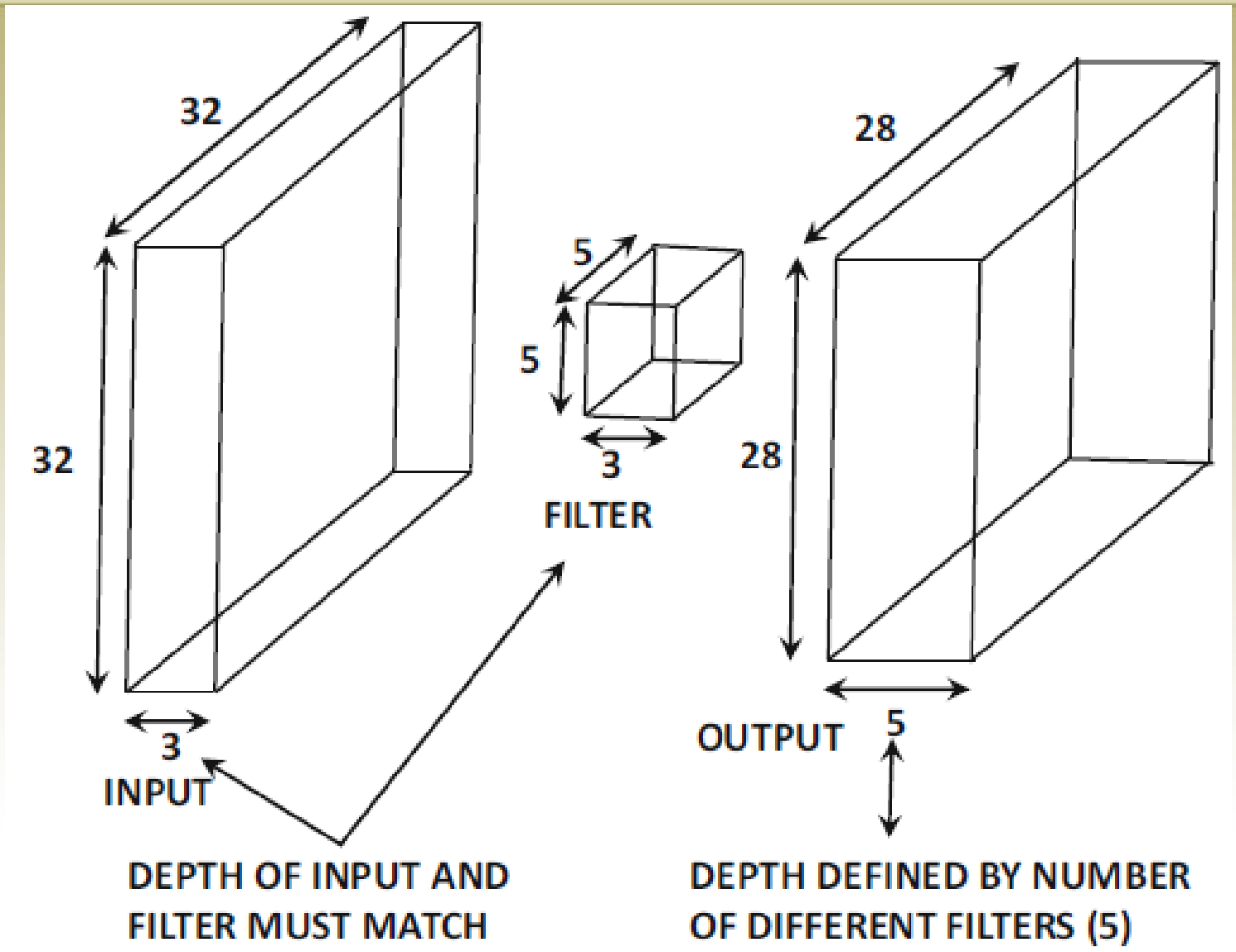
- Why do we need depth in each layer of a convolutional neural network?
  - To understand this point, let us examine how the input to the convolutional neural network is organized
- The input data to the convolutional neural network is organized into a 2-dimensional grid structure, and the values of the individual grid points are referred to as pixels
  - Each pixel, therefore, corresponds to a spatial location within the image
  - However, in order to encode the precise color of the pixel, we need a multidimensional array of values at each grid location
- In the RGB color scheme, we have an intensity of the three primary colors, corresponding to red, green, and blue, respectively
  - Therefore, if the spatial dimensions of an image are 32×32 pixels and the depth is 3 (corresponding to the RGB color channels), then the overall number of pixels in the image is $32 \times 32 \times 3$. This particular image size is quite common, and also occurs in a popularly used data set for benchmarking, known as CIFAR-10

Basic structure of CNNs

- An example of this organization is shown in the next slide

- It is natural to represent the input layer in this 3-dimensional structure because two dimensions are devoted to spatial relationships and a third dimension is devoted to the independent properties along these channels
  - For example, the intensities of the primary colors are the independent properties in the first layer
  - In the hidden layers, these independent properties correspond to various types of shapes extracted from local regions of the image
- For the purpose of discussion, assume that the input in the q-th layer is of size $L_q \times B_q \times d_q$
  - L_q refers to the height (or length), B_q refers to the width (or breadth), and d_q is the depth
  - In almost all image-centric applications, the values of L_q and B_q are the same.
    - However, we will work with separate notations for height and width in order to retain generality in presentation

Basic structure of CNNs



Basic structure of CNNs

- For the first (input) layer, these values are decided by the nature of the input data and its preprocessing
  - In the above example, the values are $L_1 = 32$, $B_1 = 32$, and $d_1 = 3$
- Later layers have exactly the same 3-dimensional organization, except that each of the d_q 2-dimensional grid of values for a particular input can no longer be considered a grid of raw pixels
- Furthermore, the value of d_q is much larger than three for the hidden layers because the number of independent properties of a given local region that are relevant to classification can be quite significant
- For $q > 1$, these grids of values are referred to as ***feature maps*** or ***activation maps***
- These values are analogous to the values in the hidden layers in a feed-forward network

Basic structure of CNNs

- In the convolutional neural network, the parameters are organized into sets of 3-dimensional structural units, known as ***filters*** or ***kernels***
  - Each filter is an array of numbers (the numbers are called **weights** or **parameters**)
- The filter is usually square in terms of its spatial dimensions, which are typically much smaller than those of the layer the filter is applied to
- On the other hand, *the depth of a filter is always same is the same as that of the layer to which it is applied*
  - Assume that the dimensions of the filter in the q-th layer are $F_q \times F_q \times d_q$
  - An example of a filter with $F_1 = 5$ and $d_1 = 3$ is shown in the figure of the Slide-20
  - It is common for the value of F_q to be small and odd
    - Examples of commonly used values of F_q are 3 and 5, although there are some interesting cases in which it is possible to use $F_q = 1$

Basic structure of CNNs

- The ***convolution operation*** places the filter at each possible position in the image (or hidden layer) so that the filter fully overlaps with the image, and performs a dot product between the $F_q \times F_q \times d_q$ parameters in the filter and the matching grid in the input volume (with same size $F_q \times F_q \times d_q$)
- The dot product is performed by treating the entries in the relevant 3-dimensional region of the input volume and the filter as vectors of size $F_q \times F_q \times d_q$, so that the elements in both vectors are ordered based on their corresponding positions in the grid-structured volume
- How many possible positions are there for placing the filter?
  - This question is important, because each such position therefore defines a spatial "pixel" (or, more accurately, a feature) in the next layer
  - In other words, the number of alignments between the filter and image defines the spatial height and width of the next hidden layer

Basic structure of CNNs

- The relative spatial positions of the features in the next layer are defined based on the relative positions of the upper left corners of the corresponding spatial grids in the previous layer
- When performing convolutions in the q-th layer, one can align the filter at $L_{q+1}=(L_q-F_q+1)$ positions along the height and $B_{q+1}=(B_q-F_q+1)$ along the width of the image (without having a portion of the filter "sticking out" from the borders of the image)
- This results in a total of $L_{q+1} \times B_{q+1}$ possible dot products, which defines the size of the next hidden layer
- In the previous example, the values of L_2 and B_2 are therefore defined as follows:

$$L_2 = 32 - 5 + 1 = 28$$

$$B_2 = 32 - 5 + 1 = 28$$

- The next hidden layer of size 28×28 is shown in the figure of Slide-20

Basic structure of CNNs

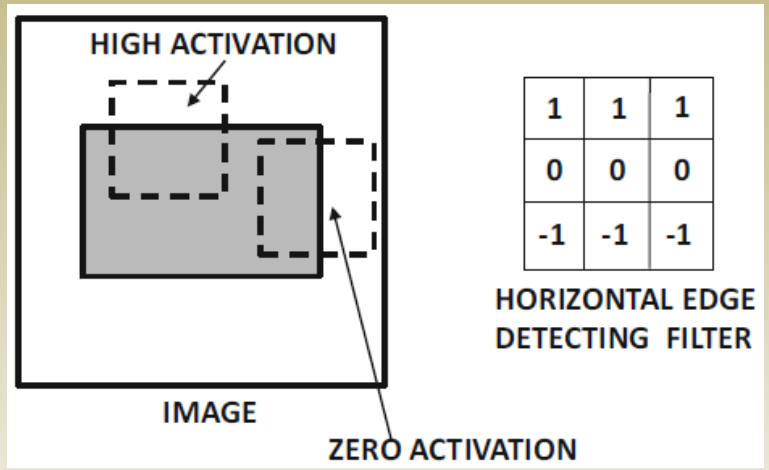
- However, this hidden layer also has a depth of size $d_2 = 5$. Where does this depth come from?
- This is achieved by using 5 different filters with their own independent sets of parameters. Each of these 5 sets of spatially arranged features obtained from the output of a single filter is referred to as a ***feature map***. Clearly, an increased number of feature maps is a result of a larger number of filters (i.e., parameter footprint)
- *The number of filters used in each layer controls the capacity of the model because it directly controls the number of parameters*
- Furthermore, increasing the number of filters in a particular layer increases the number of feature maps (i.e., depth) of the next layer
- It is possible for different layers to have very different numbers of feature maps, depending on the number of filters we use for the convolution operation in the previous layer

Basic structure of CNNs

- For example, the input layer typically only has three color channels, but it is possible for the each of the later hidden layers to have depths (i.e., number of feature maps) of more than 500
- The idea here is that each filter tries to identify a particular type of spatial pattern in a small rectangular region of the image, and therefore a large number of filters is required to capture a broad variety of the possible shapes that are combined to create the final image (unlike the case of the input layer, in which three RGB channels are sufficient)
- Typically, the later layers tend to have a smaller spatial footprint, but greater depth in terms of the number of feature maps

Basic structure of CNNs

- For example, the filter shown in the figure below represents a horizontal edge detector on a gray-scale image with one channel

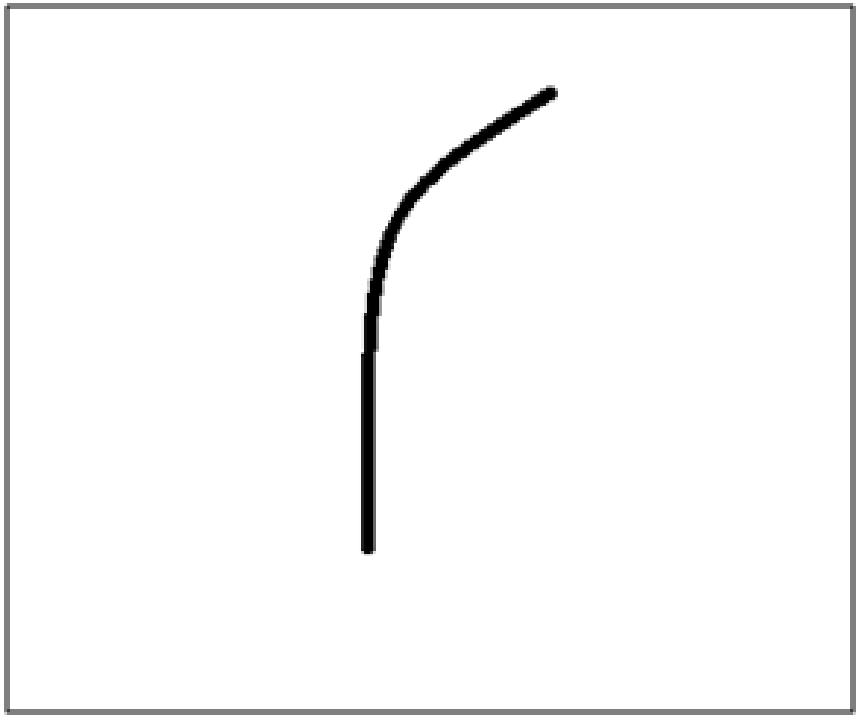


  - The resulting feature will have high activation at each position where a horizontal edge is seen. A perfectly vertical edge will give zero activation, whereas a slanted edge might give intermediate activation
- Therefore, sliding the filter everywhere in the image will already detect several key outlines of the image in a single feature map of the output volume
- Multiple filters are used to create an output volume with more than one feature map. For example, a different filter might create a spatial feature map of vertical edge activations

Example operation of a filter

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

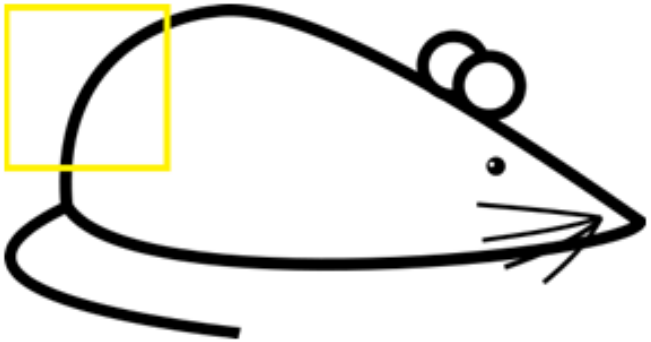
Pixel representation of filter



Visualization of a curve detector filter

Example operation of a filter

Original image



Visualization of the filter on the image

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Example operation of a filter

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

Operation on Lena of Sobel filters

Vertical Sobel filter

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

Horizontal Sobel filter

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

Basic structure of CNNs

- We are now ready to formally define the convolution operation
- The p-th filter in the q-th layer has parameters denoted by the 3-dimensional tensor $W^{(p,q)} = [w^{(p,q)}_{ijk}]$
  - The indices i, j, k indicate the positions along the height, width, and depth of the filter
- The feature maps in the q-th layer are represented by the 3-dimensional tensor $H^{(q)} = [h^{(q)}_{ijk}]$
- When the value of q is 1, the special case corresponding to the notation $H^{(1)}$ simply represents the input layer (which is not hidden)
- Then, the convolutional operations from the q-th layer to the (q+1)-th layer are defined as follows:

$$h^{(q+1)}_{i,j,p} = \sum_{r=1}^{F_q} \sum_{s=1}^{F_q} \sum_{k=1}^{d_q} w^{(p,q)}_{r,s,k} h^{(q)}_{i+r-1,j+s-1,k} \qquad \forall i \in \{1, \ldots, L_q - F_q + 1\}$$

$$\forall j \in \{1, \ldots, B_q - F_q + 1\}$$

$$\forall i \in \{1, \ldots, d_{q+1}\}$$

Basic structure of CNNs

- **The convolutional operation is really a simple dot product over the entire volume of the filter, which is repeated over all valid spatial positions (i, j) and filters (indexed by p)**
- It is intuitively helpful to understand a convolution operation by placing the filter at each of the 28×28 possible spatial positions in the first layer of figure in Slide-20 and performing a dot product between the vector of $5 \times 5 \times 3 = 75$ values in the filter and the corresponding 75 values in $H^{(1)}$
- Even though the size of the input layer in that figure is 32×32, there are only (32−5+1)×(32−5+1) possible spatial alignments between an input volume of size 32×32 and a filter of size 5×5
- One property of convolution is that it shows *equivariance to translation*
  - I.e., if we shift the pixel values in the input in any direction by one unit and then apply convolution, the corresponding feature values will shift with the input values. This is because of the shared parameters of the filter across the entire convolution. The reason for sharing parameters across the entire convolution is that the presence of a particular shape in any part of the image should be processed in the same way irrespective of its specific spatial location

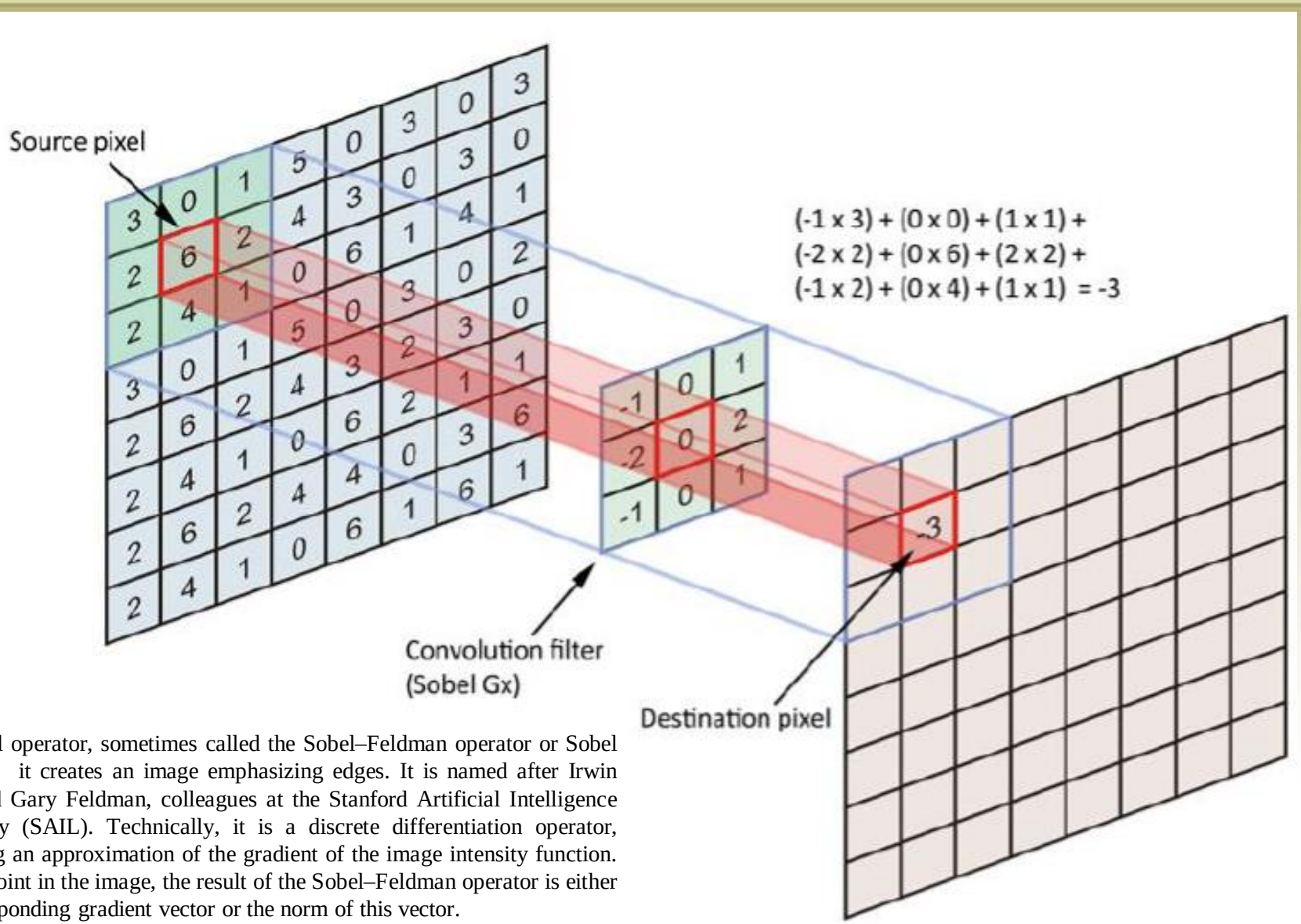
Example of a convolution

- We have shown an example of an input layer and a filter with depth 1 for simplicity (which does occur in the case of gray-scale images with a single color channel)
- Note that:
  1. **the depth of a layer must exactly match that of its filter/kernel, and**
  2. **the contributions of the dot products over all the feature maps in the corresponding grid region of a particular layer will need to be added (in the general case) to create a single output feature value in the next layer**
- The figure in the next slide depicts two specific examples of the convolution operations with a layer of size 7×7×1 and a 3×3×1 filter in the bottom row. Furthermore, the entire feature map of the next layer is shown on the upper right-hand side of figure
  - Examples of two convolution operations are shown in which the outputs are 16 and 26, respectively. These values are arrived at by using the following multiplication and aggregation operations:

$$5 \times 1 + 8 \times 1 + 1 \times 1 + 1 \times 2 = 16$$

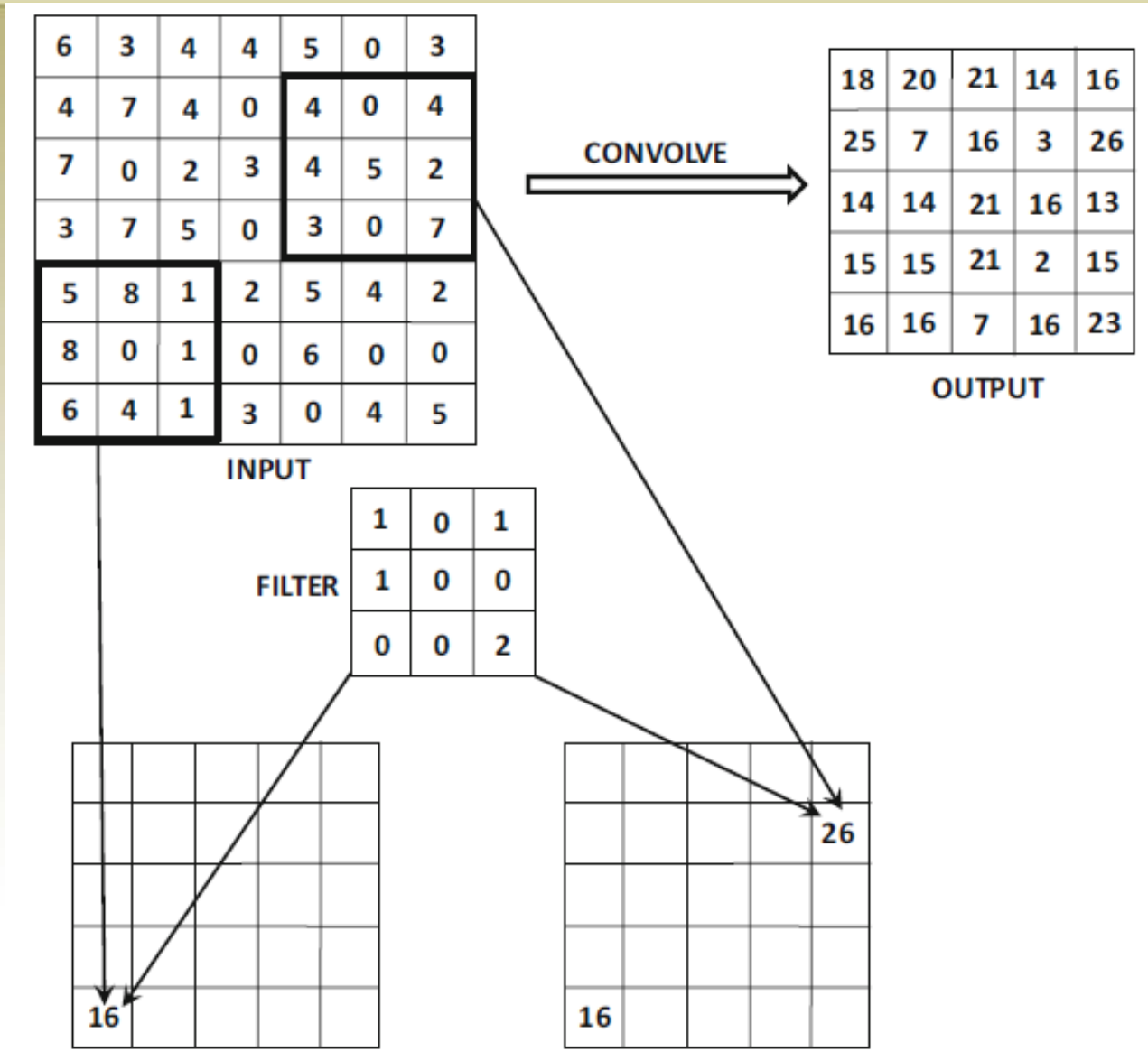
$$4 \times 1 + 4 \times 1 + 4 \times 1 + 7 \times 2 = 26$$

Example-1 of a convolution



The Sobel operator, sometimes called the Sobel–Feldman operator or Sobel filter, and it creates an image emphasizing edges. It is named after Irwin Sobel and Gary Feldman, colleagues at the Stanford Artificial Intelligence Laboratory (SAIL). Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel–Feldman operator is either the corresponding gradient vector or the norm of this vector.

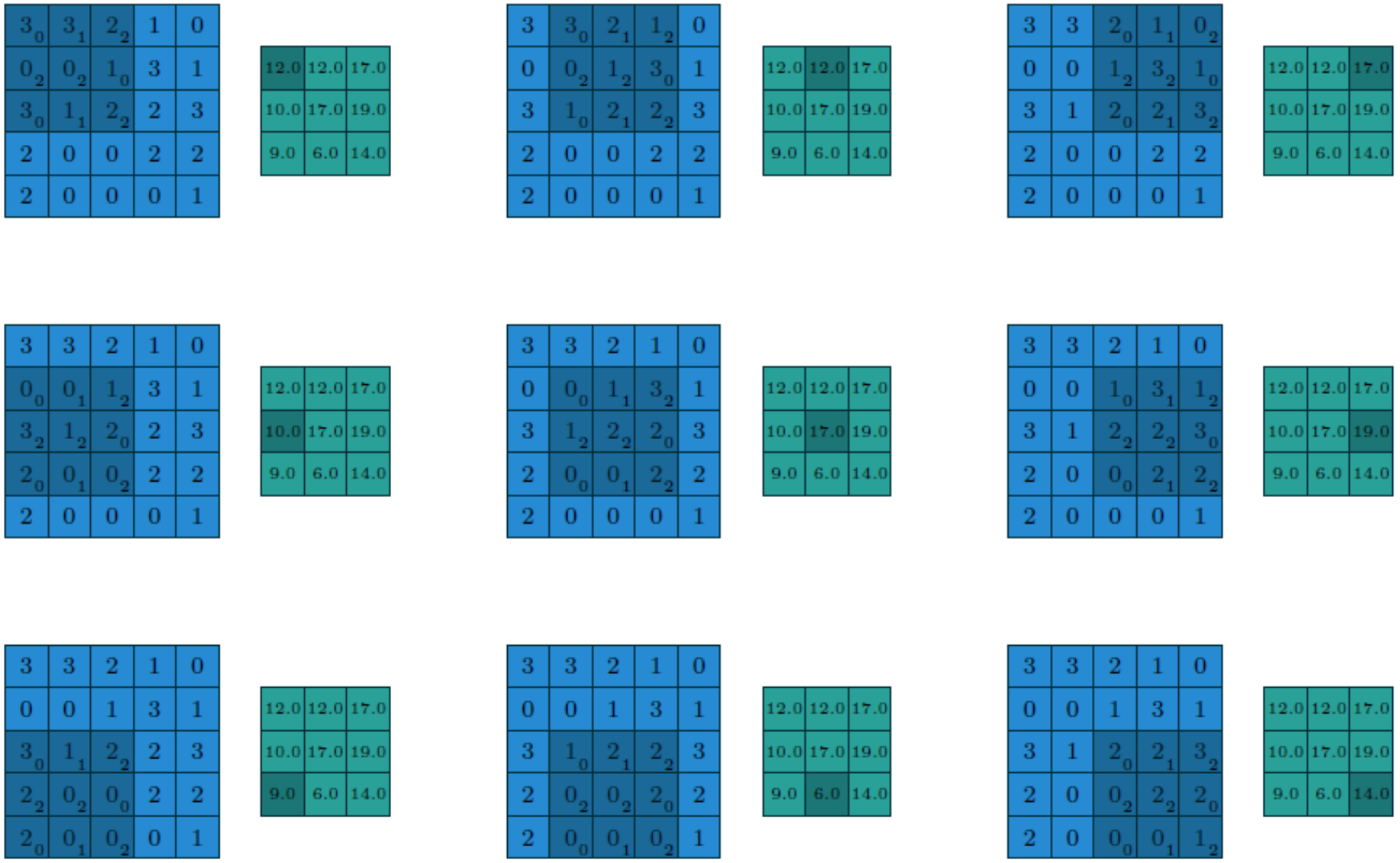
Example-2 of a convolution



Example-3 of a convolution

kernel=

0	1	2
2	2	0
0	1	2



Comments on convolution

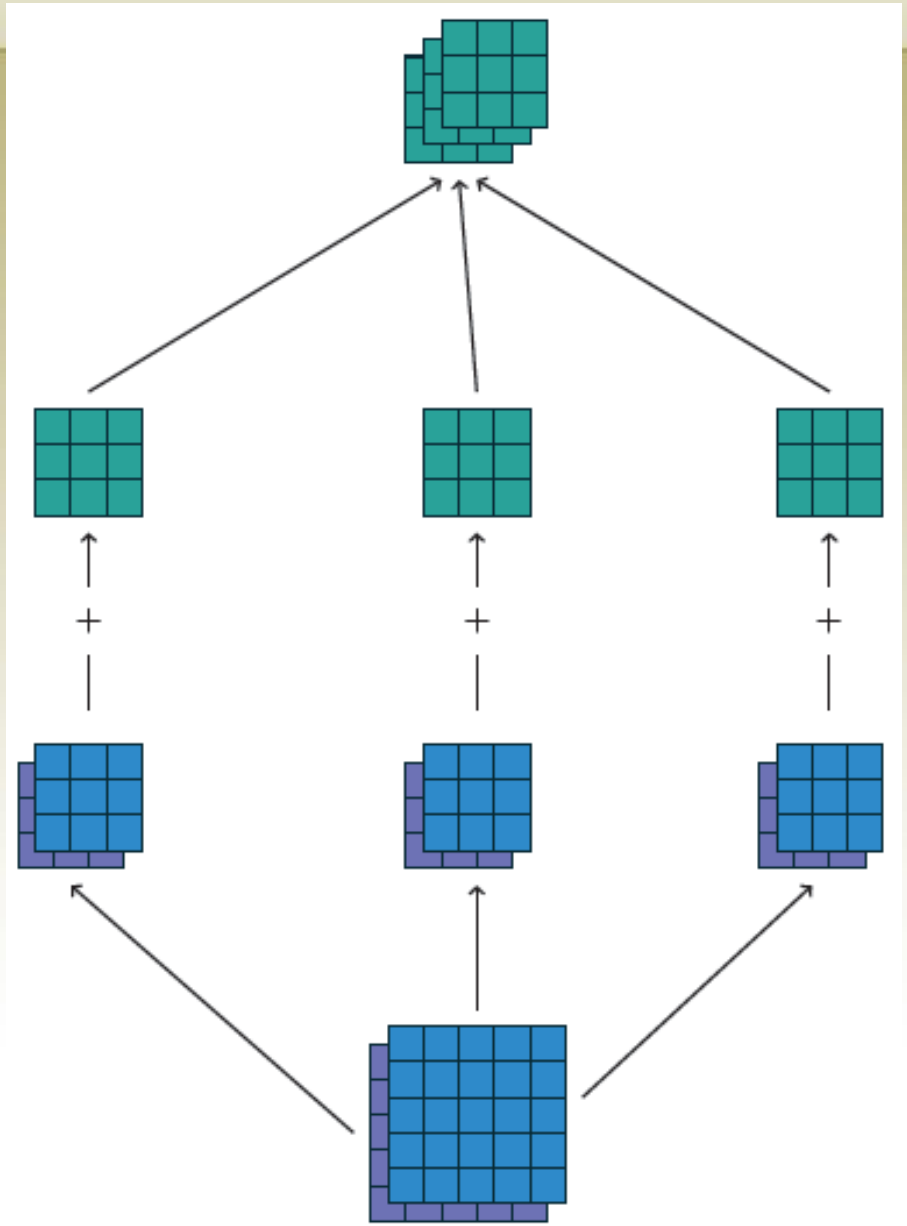
- A convolution in the q-th layer increases the *receptive field* (the region covered by a filter/kernel while it is sliding over the input) of a feature from the q-th layer to the (q+1)-th layer. In other words, each feature in the next layer captures a larger spatial region in the input layer
  - I.e., when using a 3 × 3 filter convolution successively in three layers, the activations in the first, second, and third hidden layers capture pixel regions of size 3×3, 5×5, and 7×7, respectively, in the original input image
- As we will see later, other types of operations increase the receptive fields further, as they reduce the size of the spatial footprint of the layers
- This is a natural consequence of the fact that features in later layers capture complex characteristics of the image over larger spatial regions, and then combine the simpler features in earlier layers

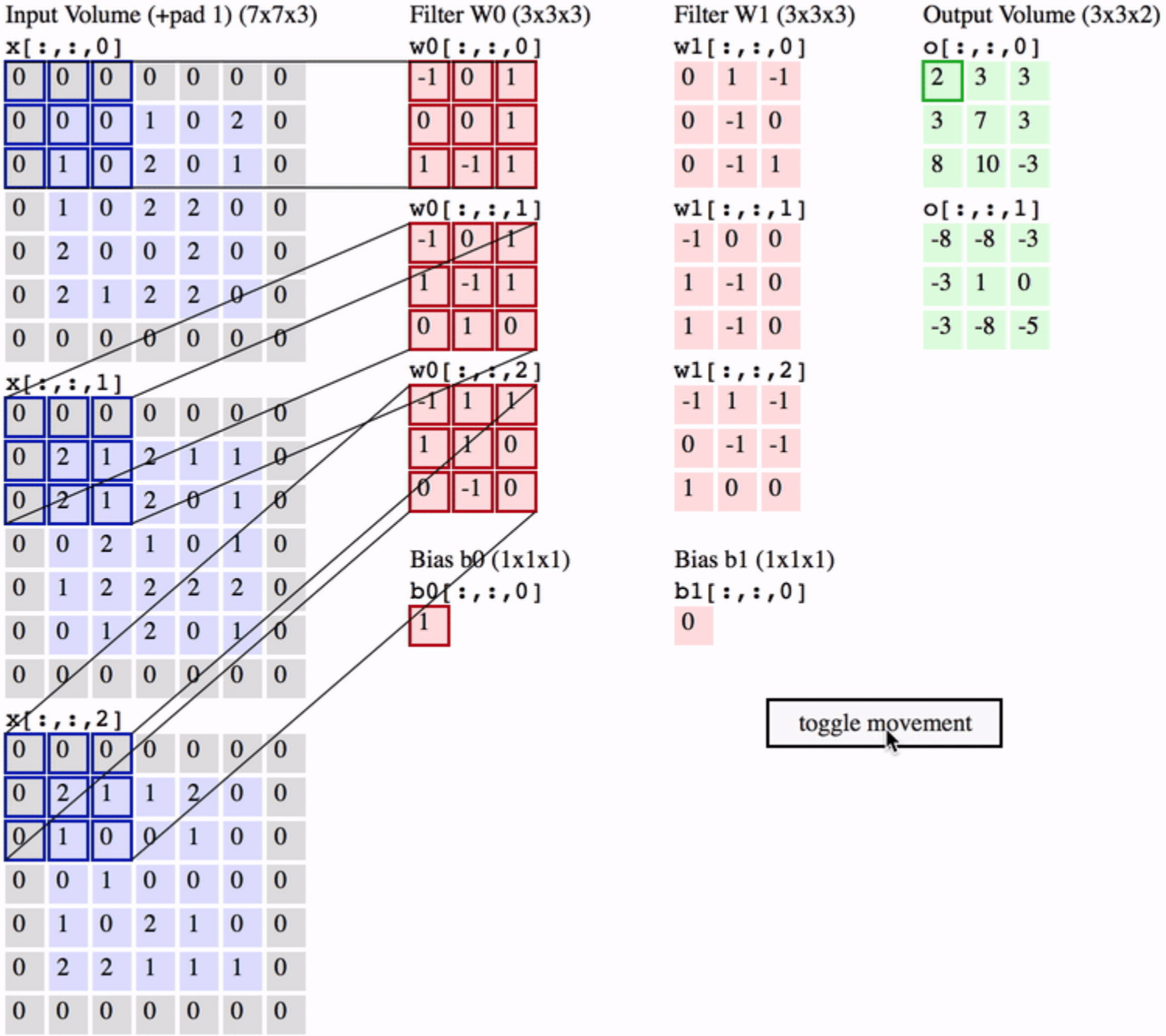
Comments on convolution

- When performing the operations from the q-th layer to the (q+1)-th layer, the depth d_{q+1} of the computed layer depends on the ***number*** of filters in the q-th layer, and it is independent of the depth of the q-th layer or any of its other dimensions
- In other words, the depth d_{q+1} in the (q+1)-th layer is always equal to the number of filters in the q-th layer
  - For example, the depth of the second layer in the figure of Slide-20 is 5, because a total of five filters are used in the first layer for the transformation
  - However, in order to perform the convolutions in the second layer (to create the third layer), one must now use filters of depth 5 in order to match the new depth of this layer, even though filters of depth 3 were used in the convolutions of the first layer (to create the second layer)

Convolution example with 2 kernels

- A convolution mapping from two input feature maps to three output feature maps using a 3x2x3x3 collection of kernels **w**
- In the left pathway, input feature map 1 is convolved with kernel $\mathbf{w}_{1,1}$ and input feature map 2 is convolved with kernel $\mathbf{w}_{1,2}$, and the results are summed together element-wise to form the first output feature map
- The same is repeated for the middle and right pathways to form the second and third feature maps, and all three output feature maps are grouped together to form the output




Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 0 2 0 1 0
0 1 0 2 2 0 0
0 2 0 0 2 0 0
0 2 1 2 2 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 2 1 2 1 1 0
0 2 1 2 0 1 0
0 0 2 1 0 1 0
0 1 2 2 2 2 0
0 0 1 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 1 2 0 0
0 1 0 0 1 0 0
0 0 1 0 0 0 0
0 1 0 2 1 0 0
0 2 2 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 0 1
0 0 1
1 -1 1
w0[:,:,1]
-1 0 1
1 -1 1
0 1 0
w0[:,:,2]
-1 1 1
1 1 0
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
0 1 -1
0 -1 0
0 -1 1
w1[:,:,1]
-1 0 0
1 -1 0
1 -1 0
w1[:,:,2]
-1 1 -1
0 -1 -1
1 0 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 3 3
3 7 3
8 10 -3
o[:,:,1]
-8 -8 -3
-3 1 0
-3 -8 -5
toggle movement

Padding

- The convolution operation reduces the size of the (q+1)-th layer in comparison with the size of the q-th layer
  - This type of reduction in size is not desirable in general, because it tends to lose some information along the borders of the image (or of the feature map, in the case of hidden layers)
- This problem can be resolved by using ***padding***. In padding, one adds $(F_q - 1)/2$ "pixels" all around the borders of the feature map in order to maintain the spatial footprint
  - Note that these pixels are really feature values in the case of padding hidden layers. The value of each of these padded feature values is set to 0, irrespective of whether the input or the hidden layers are being padded. As a result, the spatial height and width of the input volume will both increase by $(F_q - 1)$, which is exactly what they reduce by (in the output volume) after the convolution is performed. The padded portions do not contribute to the final dot product because their values are set to 0

Padding: Half-padding

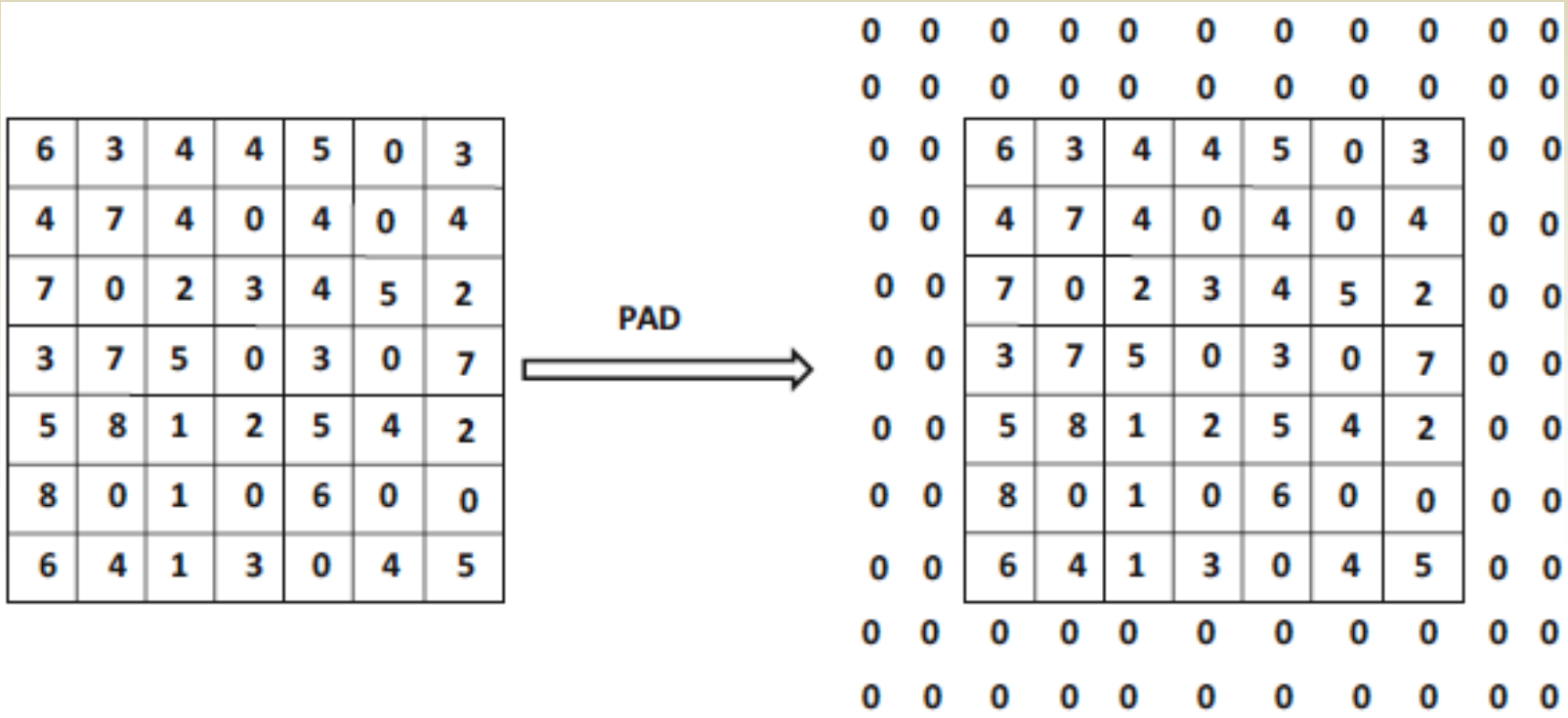
- In a sense, what padding does is to allow the convolution operation with a portion of the filter "sticking out" from the borders of the layer and then performing the dot product only over the portion of the layer where the values are defined
- This type of padding is referred to as ***half-padding*** because (almost) half the filter is sticking out from all sides of the spatial input in the case where the filter is placed in its extreme spatial position along the edges. Half-padding is designed to maintain the spatial footprint exactly

Padding: Valid padding

- When padding is not used, the resulting "padding" is also referred to as a ***valid padding***
- Valid padding generally does not work well from an experimental point of view. Using half-padding ensures that some of the critical information at the borders of the layer is represented in a standalone way
- In the case of valid padding, the contributions of the pixels on the borders of the layer will be under-represented compared to the central pixels in the next hidden layer, which is undesirable
- This under-representation will be compounded over multiple layers. Therefore, padding is typically performed in all layers, and not just in the first layer where the spatial locations correspond to input values

Padding: Half-padding example

- Consider a situation in which the layer has size 7×7×1 and the filter is of size 5×5×1
- Therefore, (5 − 1)/2 = 2 zeros are padded on all sides of the image
- As a result, the 7 × 7 spatial footprint first increases to 11 × 11 because of padding, and then it reduces back to 7 × 7 after performing the convolution



Padding: Full padding

- Another useful form of padding is ***full-padding***
- In full-padding, we allow (almost) the full filter to stick out from various sides of the input. In other words, a portion of the filter of size $F_q - 1$ is allowed to stick out from any side of the input with an overlap of only one spatial feature
- For example, the kernel and the input image might overlap at a single pixel at an extreme corner
- Therefore, the input is padded with $(F_q - 1)$ zeros on each side. In other words, each spatial dimension of the input increases by $2(F_q - 1)$
- Therefore, if the input dimensions in the original image are L_q and B_q, the padded spatial dimensions in the input volume become $L_q + 2(F_q - 1)$ and $B_q + 2(F_q - 1)$
- After performing the convolution, the feature-map dimensions in layer (q+1) become $L_q + F_q - 1$ and $B_q + F_q - 1$, respectively

Padding: Full padding

- While convolution normally reduces the spatial footprint, full padding increases the spatial footprint
- Interestingly, full-padding increases each dimension of the spatial footprint by the same value ($F_q - 1$) that no-padding decreases it

- *This relationship is not a coincidence because a "reverse" convolution operation can be implemented by applying another convolution on the fully padded output (of the original convolution) with an appropriately defined kernel of the same size*

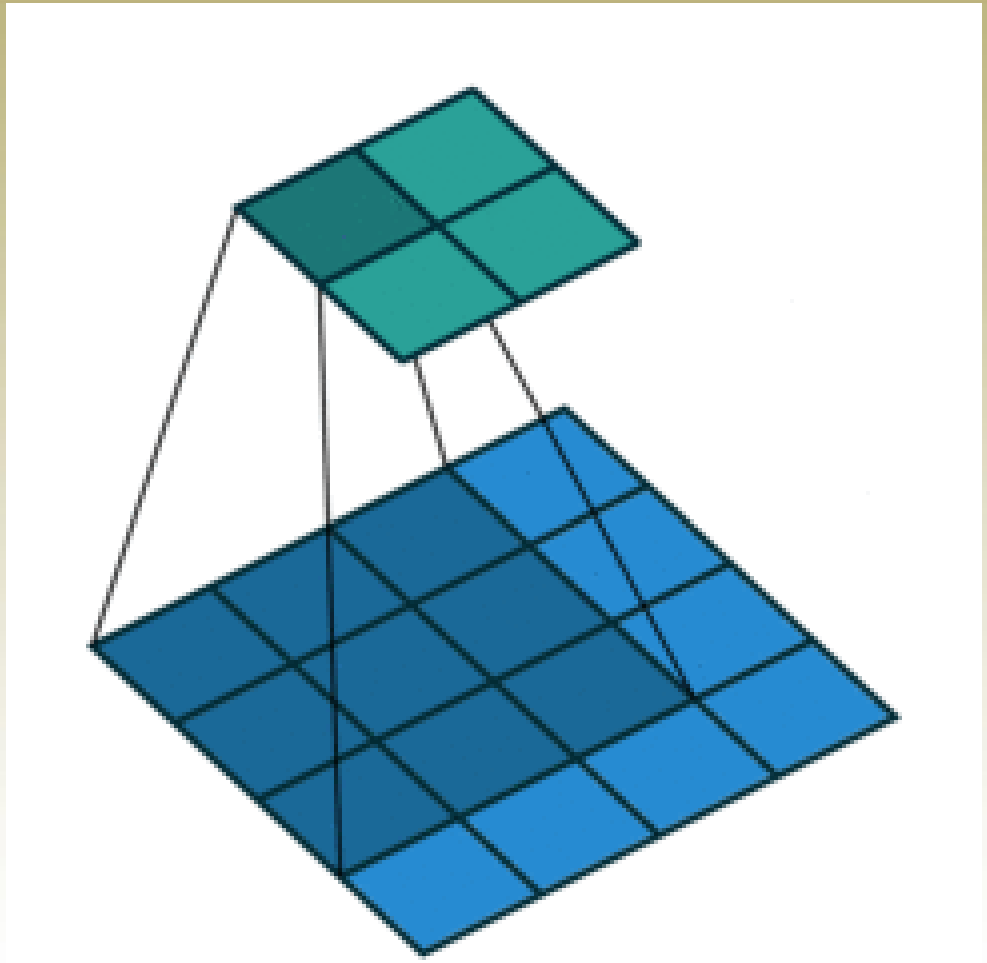
Strides

- There are other ways in which convolution can reduce the spatial footprint of the image (or hidden layer)
- The above approach performs the convolution at every position in the spatial location of the feature map
- However, it is not necessary to perform the convolution at every spatial position in the layer. One can reduce the level of granularity of the convolution by using the notion of ***strides***
- The description above corresponds to the case when a stride of 1 is used. When a stride of S_q is used in the q-th layer, the convolution is performed at the locations 1, $S_q + 1$, $2S_q + 1$, and so on along both spatial dimensions of the layer
- The spatial size of the output on performing this convolution1 has height of $(L_q - F_q)/S_q + 1$ and a width of $(B_q - F_q)/S_q + 1$
- As a result, the use of strides will result in a reduction of each spatial dimension of the layer by a factor of approximately S_q and the area by S^2_q , although the actual factor may vary because of edge effects

Strides

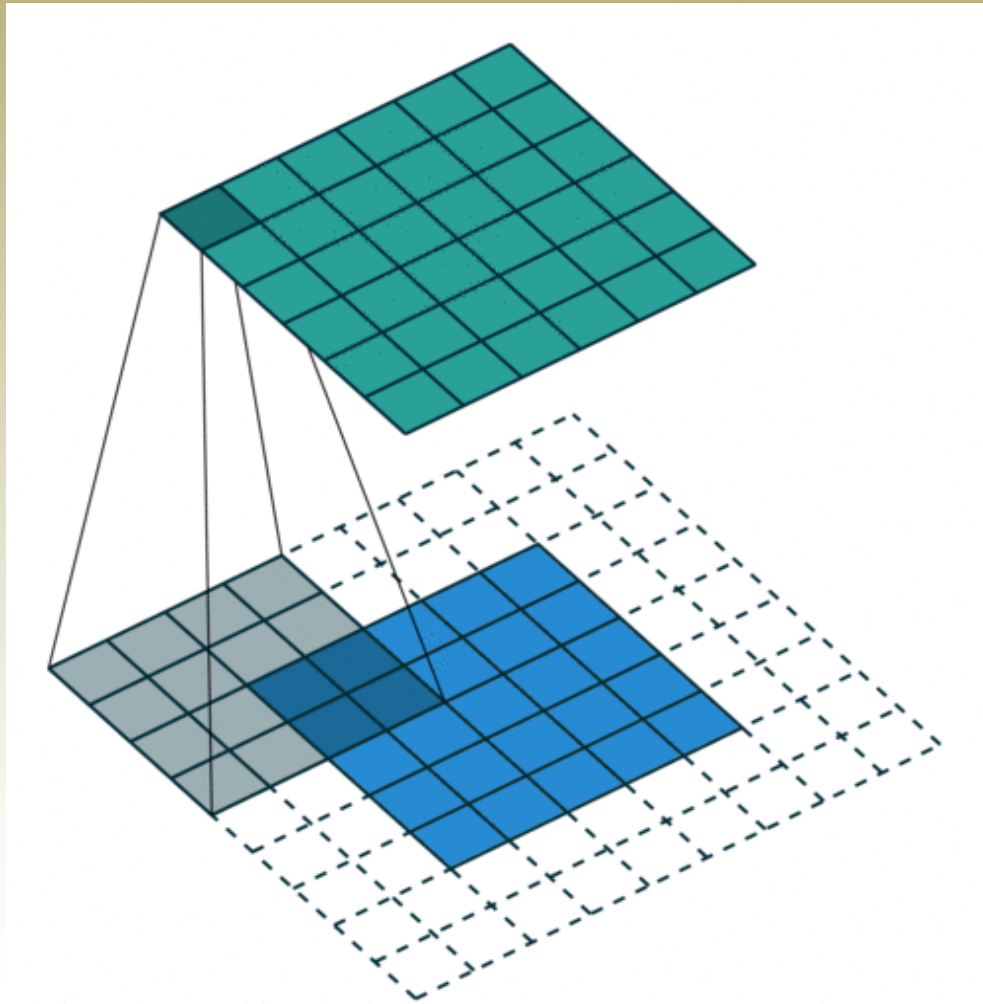
- It is most common to use a stride of 1, although a stride of 2 is occasionally used as well
  - It is rare to use strides more than 2 in normal circumstances
- Even though a stride of 4 was used in the input layer of the winning architecture of the ILSVRC competition of 2012, the winning entry in the subsequent year reduced the stride to 2 to improve accuracy
- Larger strides can be helpful in memory-constrained settings or to reduce overfitting if the spatial resolution is unnecessarily high
- Strides have the effect of rapidly increasing the receptive field of each feature in the hidden layer, while reducing the spatial footprint of the entire layer
  - An increased receptive field is useful in order to capture a complex feature in a larger spatial region of the image
  - As we will see later, the hierarchical feature engineering process of a convolutional neural network captures more complex shapes in later layers. Historically, the receptive fields have been increased with another operation, known as the max-pooling operation. In recent years, larger strides have been used in lieu of max-pooling operations, which will be discussed later

Convolution (https://github.com/vdumoulin/conv_arithmetic)



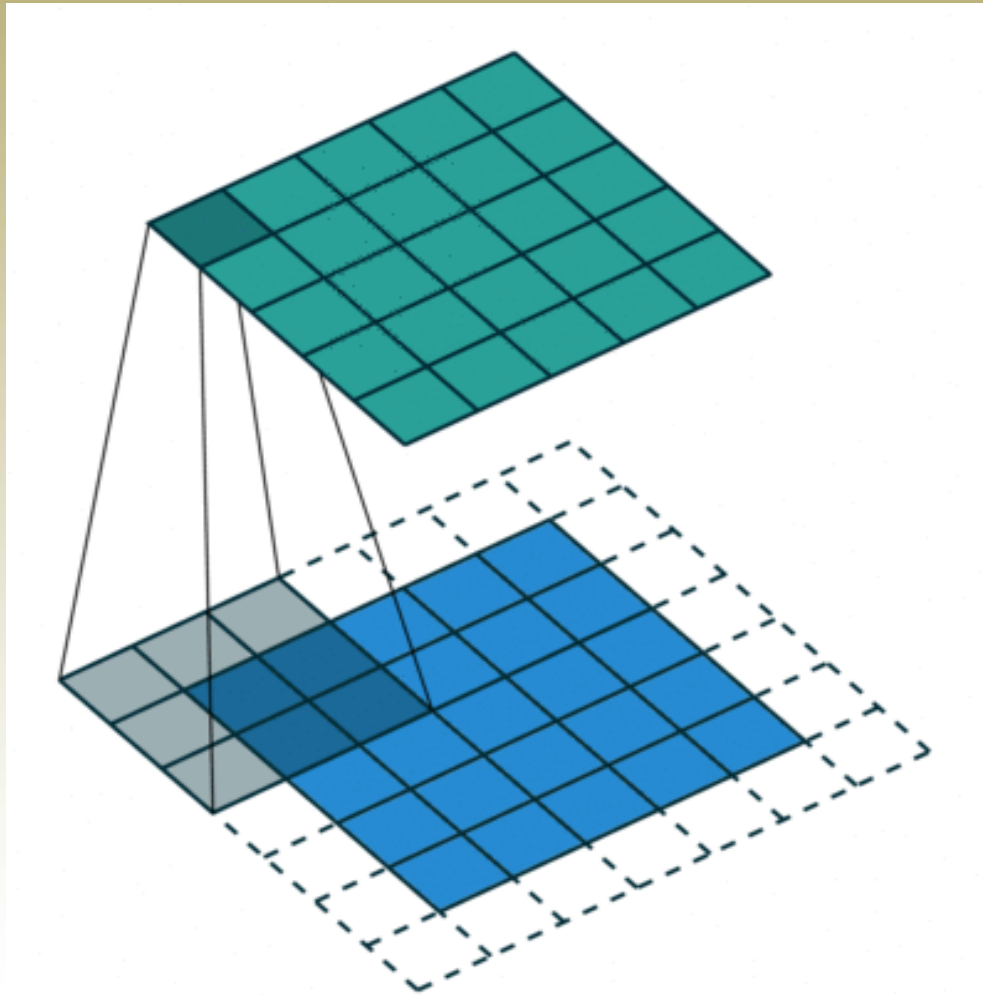
No padding, no strides

Convolution (https://github.com/vdumoulin/conv_arithmetic)



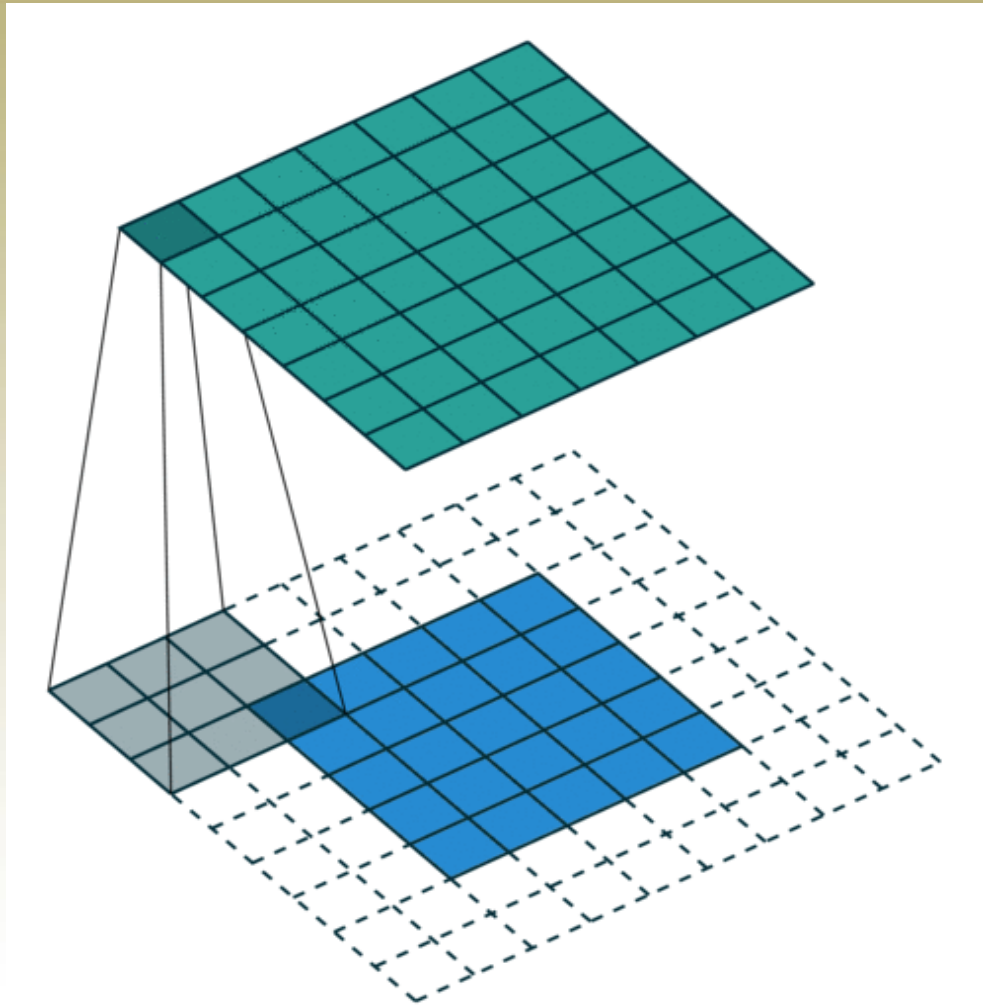
Arbitrary padding, no strides

Convolution (https://github.com/vdumoulin/conv_arithmetic)



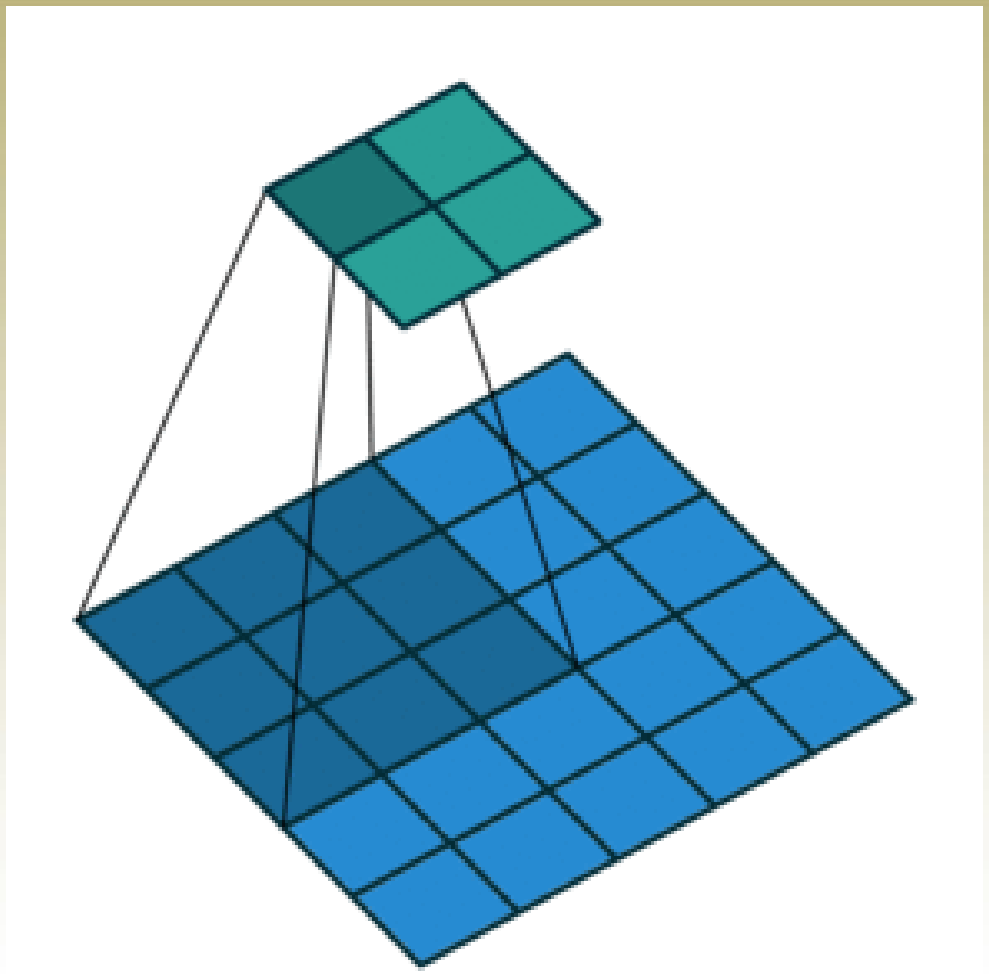
Half padding, no strides

Convolution (https://github.com/vdumoulin/conv_arithmetic)



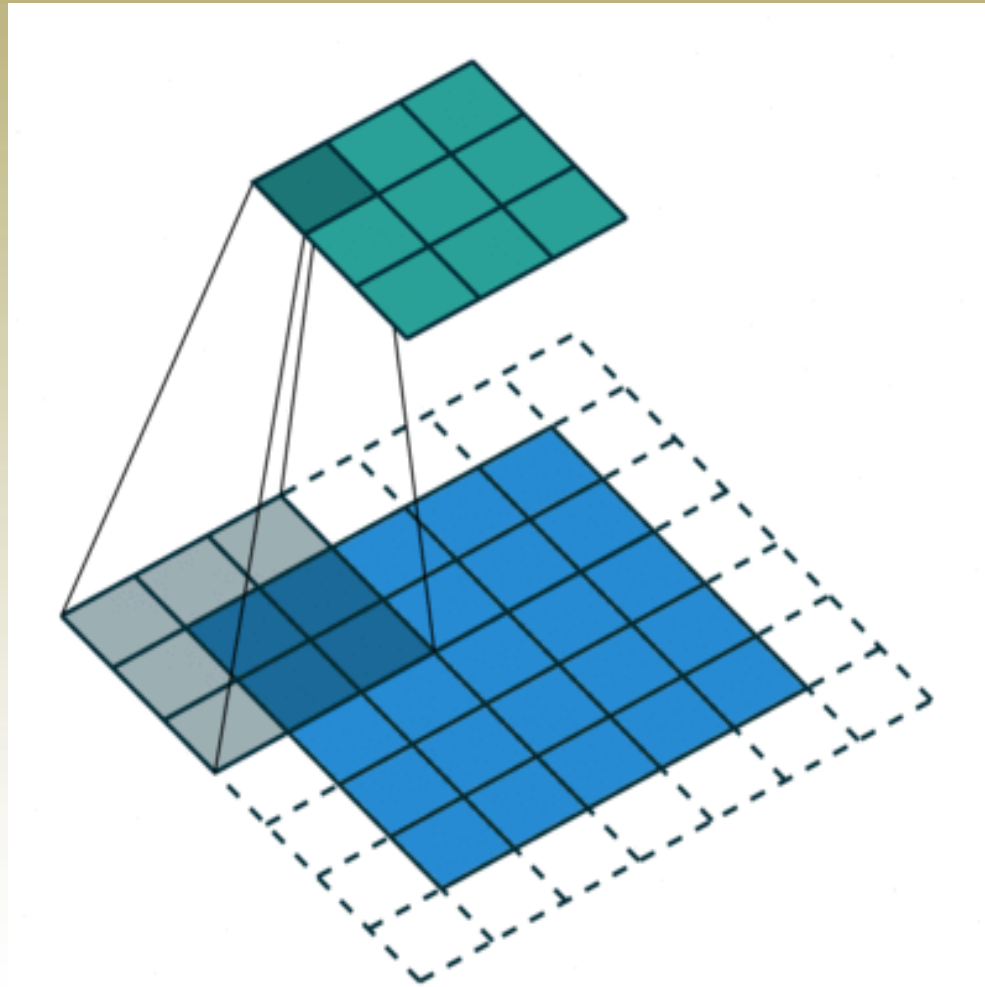
Full padding, no strides

Convolution (https://github.com/vdumoulin/conv_arithmetic)



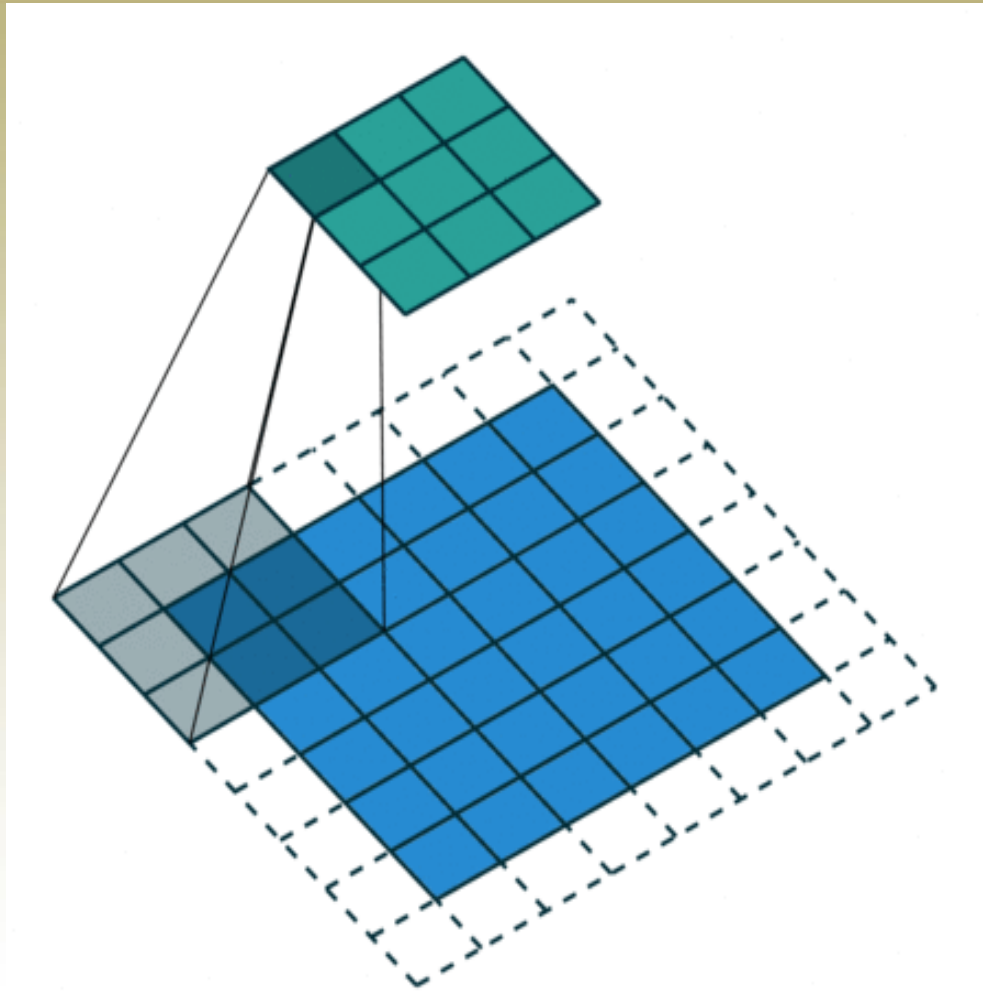
No padding, strides

Convolution (https://github.com/vdumoulin/conv_arithmetic)



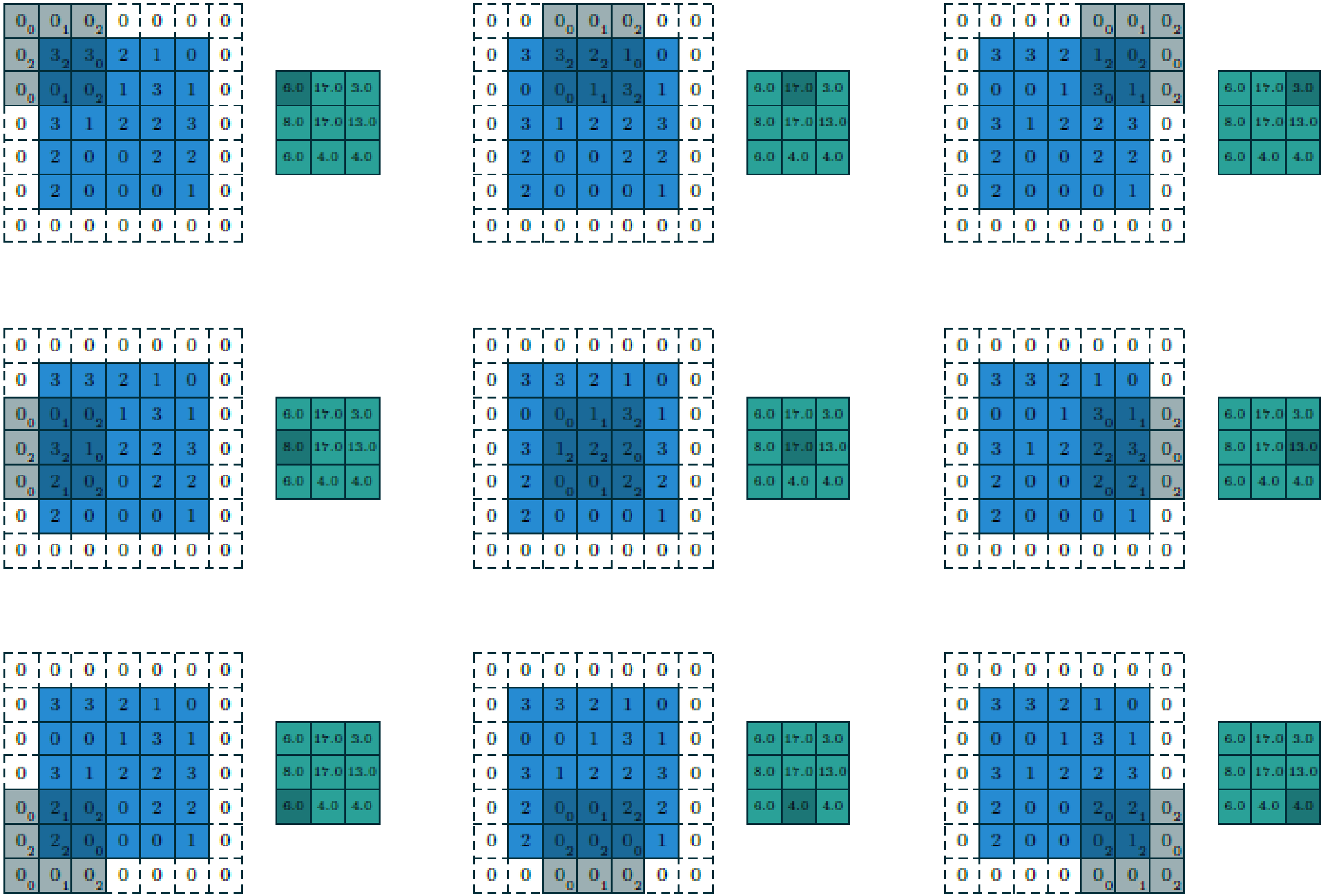
Padding, strides

Convolution (https://github.com/vdumoulin/conv_arithmetic)



Padding, strides (odd)

Example-4 of a convolution (3x3 kernel applied to a 5x5 input padded with a 1x1 border of zeros using 2x2 strides)



Typical settings

- It is common to use stride sizes of 1 in most settings. Even when strides are used, small strides of size 2 are used. Furthermore, it is common to have $L_q = B_q$.
  - In other words, it is desirable to work with square images. In cases where the input images are not square, preprocessing is used to enforce this property.
  - For example, one can extract square patches of the image to create the training data.
- The number of filters in each layer is often a power of 2, because this often results in more efficient processing
  - Such an approach also leads to hidden layer depths that are powers of 2
- Typical values of the spatial extent of the filter size (denoted by F_q) are 3 or 5
  - Small filter sizes often provide the best results, although some practical challenges exist in using filter sizes that are too small. Small filter sizes typically lead to deeper networks (for the same parameter footprint) and therefore tend to be more powerful
  - In fact, one of the top entries in an ILSVRC contest, referred to as VGG, was the first to experiment with a spatial filter dimension of only $F_q = 3$ for all layers, and the approach was found to work very well in comparison with larger filter sizes

Use of bias

- As in all neural networks, it is also possible to add biases to the forward operations
- Each unique filter in a layer is associated with its own bias
- Therefore, the p-th filter in the q-th layer has bias $b^{(p,q)}$
- When any convolution is performed with the p-th filter in the q-th layer, the value of $b^{(p,q)}$ is added to the dot product
- The use of the bias simply increases the number of parameters in each filter by 1, and therefore it is not a significant overhead
  - Like all other parameters, the bias is learned during backpropagation
  - One can treat the bias as a weight of a connection whose input is always set to +1. This special input is used in all convolutions, irrespective of the spatial location of the convolution
- Therefore, one can assume that a special pixel appears in the input whose value is always set to 1. Therefore, the number of input features in the q-th layer is $1+L_q \times B_q \times d_q$

The ReLU layer

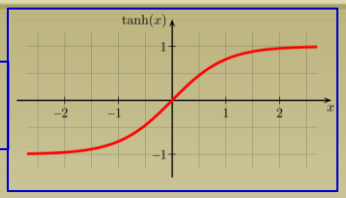
- The convolution operation is interleaved with the pooling and ReLU operations
- The ReLU activation is not very different from how it is applied in a traditional neural network

- For each of the $L_q \times B_q \times d_q$ values in a layer, the ReLU activation function is applied to it to create $L_q \times B_q \times d_q$ thresholded values
- These values are then passed on to the next layer

- Therefore, applying the ReLU does not change the dimensions of a layer because it is a simple one-to-one mapping of activation values

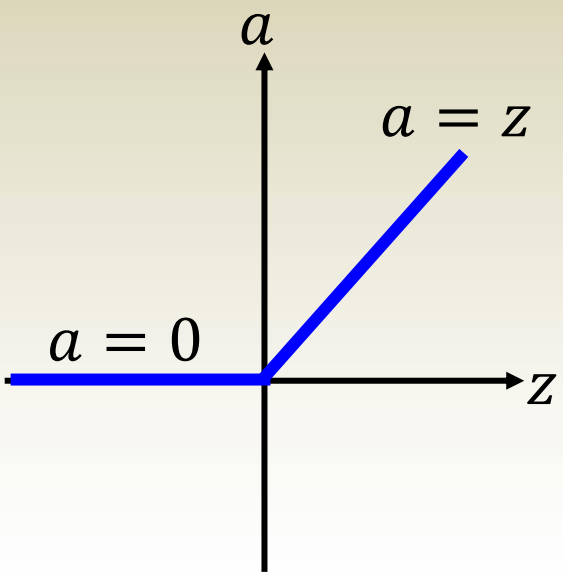
The ReLU activation function

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



- Rectified Linear Unit (ReLU): $f(z) = \max(0, z)$

we called it: *Positive linear* (*poslin*)

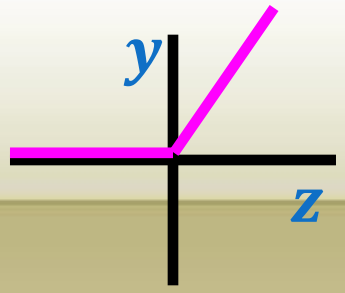


[Xavier Glorot, AISTATS'11]
[Andrew L. Maas, ICML'13]
[Kaiming He, arXiv'15]

Reason:

1. Fast to compute
2. Biological reason
3. Vanishing gradient problem

The ReLU activation function



- Activation function
  - $y = \max(0, z)$
- Derivative

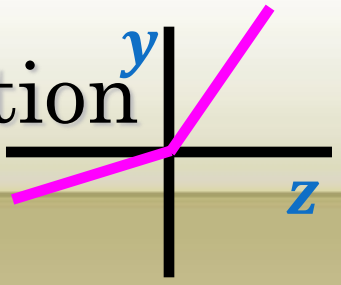
$$\frac{\partial y}{\partial z} = \begin{cases} 0 & z \leq 0 \\ 1 & \text{otherwise} \end{cases}$$

- Advantages
  - no squashing of back propagated error signal as long as unit is activated
  - discontinuity in derivative at z=0
- Disadvantages
  - can potentially lead to exploding gradients and activations
  - may waste units: units that are never activated above threshold won't learn

The ReLU layer

- In traditional neural networks, the activation function is combined with a linear transformation with a matrix of weights to create the next layer of activations
- **Similarly, a ReLU typically follows a convolution operation (which is the rough equivalent of the linear transformation in traditional neural networks), and the ReLU layer is often not explicitly shown in pictorial illustrations of the convolution neural network architectures**
- It is noteworthy that the use of the ReLU activation function is a recent evolution in neural network design
  - In the earlier years, saturating activation functions like sigmoid and tanh were used
  - Leaky ReLU, Randomized Leaky ReLU, Parameterized ReLU, Exponential Linear Units (ELU), Scaled Exponential Linear Units, hardtanh, softtanh, softsign, softmax, softplus
- It was shown in that the use of the ReLU has tremendous advantages over these activation functions both in terms of speed and accuracy
  - Increased speed is also connected to accuracy because it allows one to use deeper models and train them for a longer time

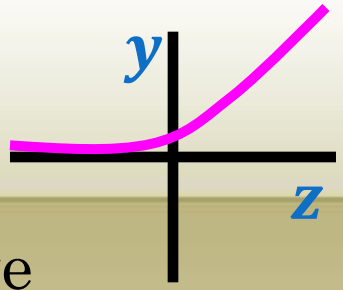
The Leaky ReLU activation function



- Activation function
  - $y = \begin{cases} z & z > 0 \\ \alpha z & \text{otherwise} \end{cases}$
- Derivative
  - $\frac{\partial y}{\partial z} = \begin{cases} 1 & z > 0 \\ \alpha & \text{otherwise} \end{cases}$

- Reduces to standard ReLU if $\alpha = 0$
- Trade off
  - $\alpha = 0$ leads to inefficient use of resources (underutilized units)
  - $\alpha = 1$ lose nonlinearity essential for interesting computation

The Softplus activation function

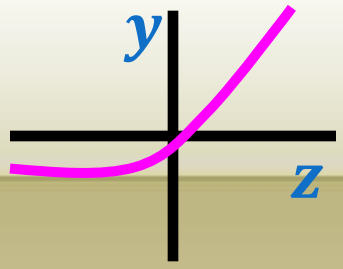


- Activation function
  - $y = \ln(1 + e^z)$
- Derivative

$$\frac{\partial y}{\partial z} = \frac{1}{1+e^{-z}} = \text{logistic}(z)$$

- Derivative
  - defined everywhere
  - zero only for $z \to -\infty$

Exponential Linear Unit (ELU)



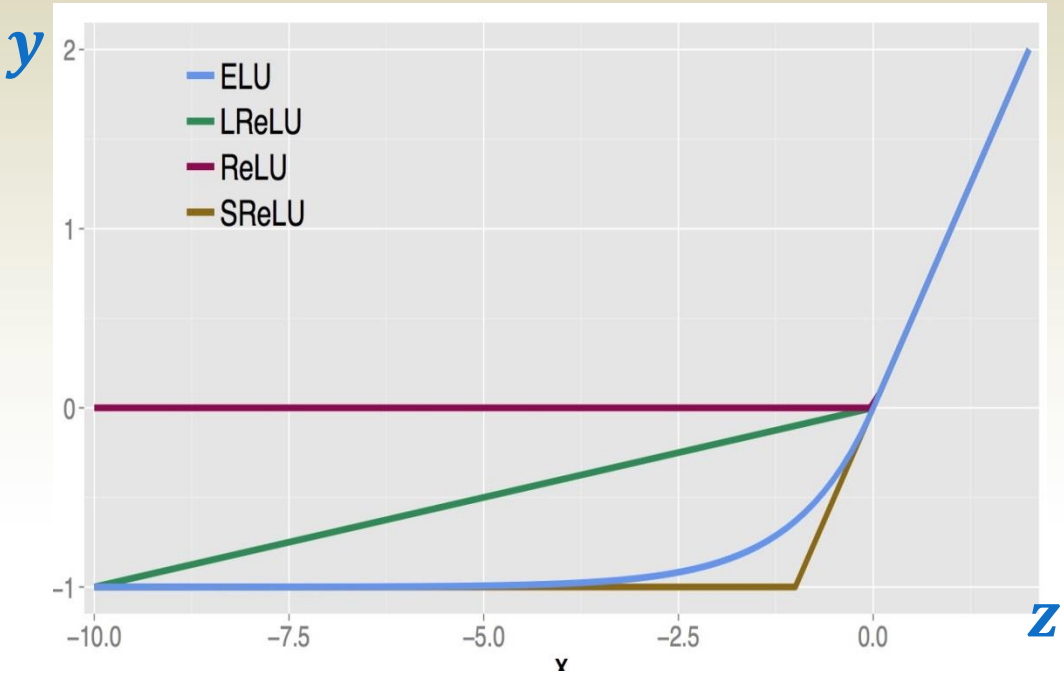
- Activation function

$$y = \begin{cases} z & z > 0 \\ \alpha(e^z - 1) & z \leq 0 \end{cases}$$

Derivative

$$\frac{\partial y}{\partial z} = \begin{cases} 1 & z > 0 \\ y + \alpha & z \leq 0 \end{cases}$$

- Reduces to standard ReLU if $\alpha = 0$



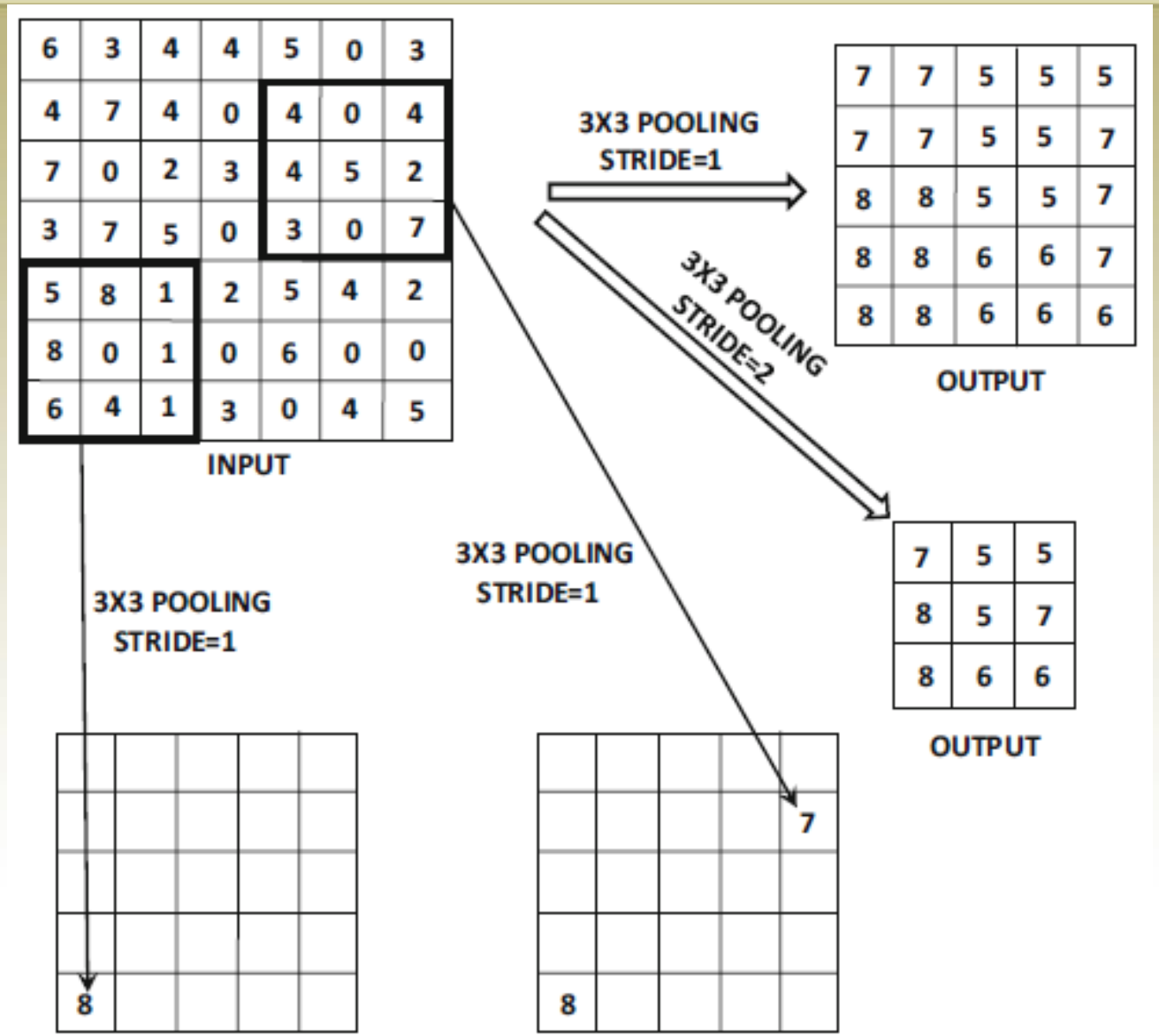
Pooling

- The pooling operation works on small grid regions of size $P_q \times P_q$ in each layer, and produces another layer with the same depth (unlike filters)
- For each square region of size $P_q \times P_q$ in each of the d_q activation maps, the maximum of these values is returned
- This approach is referred to as ***max-pooling***
- If a stride of 1 is used, then this will produce a new layer of size $(L_q - P_q + 1) \times (B_q - P_q + 1) \times d_q$
- However, it is more common to use a stride $Sq > 1$ in pooling. In such cases, the length of the new layer will be $(L_q - P_q)/Sq + 1$ and the breadth will be $(B_q - P_q)/S_q + 1$
- Therefore, pooling drastically reduces the spatial dimensions of each activation map

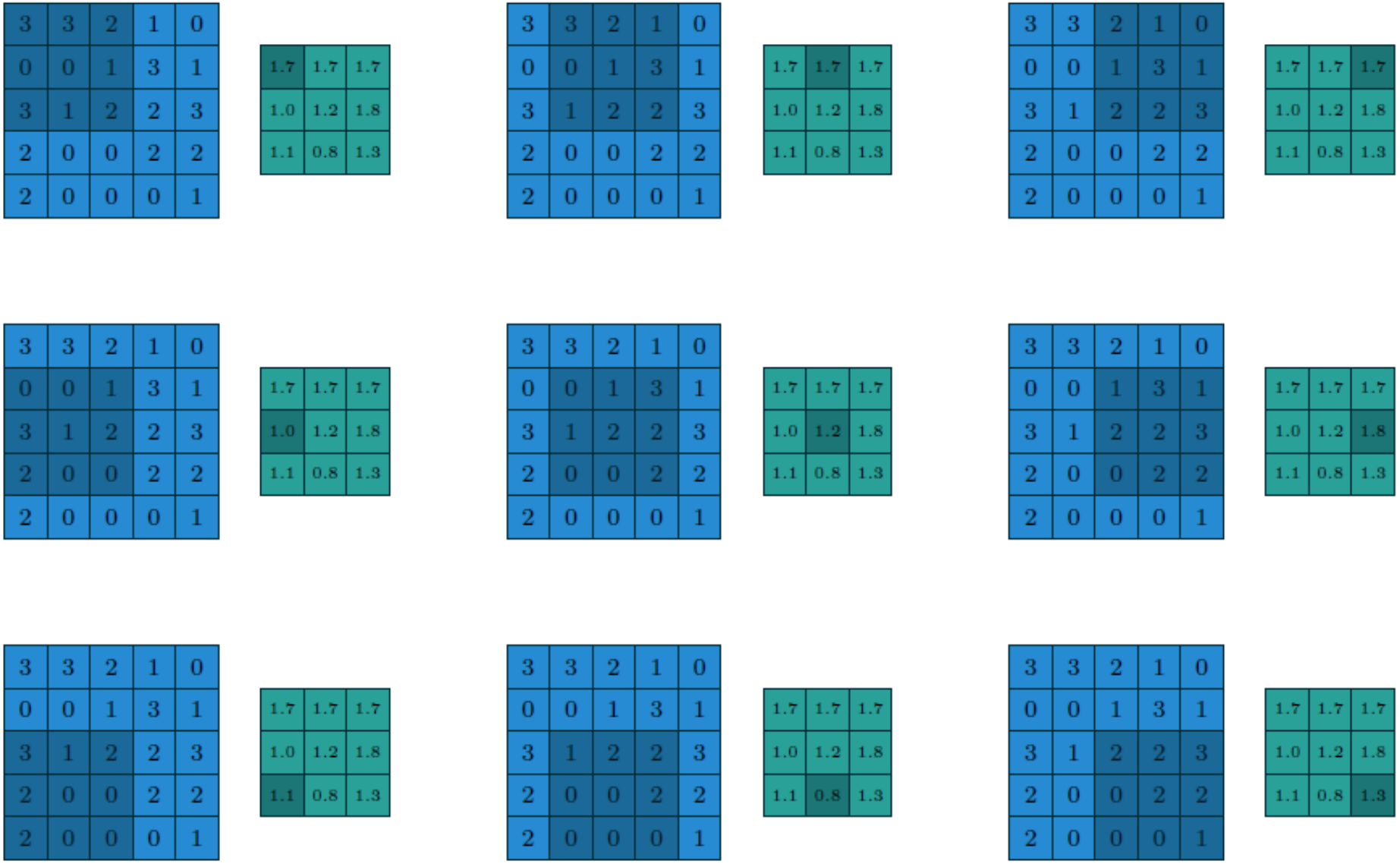
Pooling

- Unlike with convolution operations, pooling is done at the level of each activation map
- Whereas a convolution operation simultaneously uses all d_q feature maps in combination with a filter to produce a single feature value, pooling independently operates on each feature map to produce another feature map
  - Therefore, the operation of pooling does not change the number of feature maps
  - In other words, the depth of the layer created using pooling is the same as that of the layer on which the pooling operation was performed.
- Examples of pooling with strides of 1 and 2 are shown in next slide's figure. Here, we use pooling over 3×3 regions. The typical size P_q of the region over which one performs pooling is 2×2. At a stride of 2, there would be no overlap among the different regions being pooled, and it is quite common to use this type of setting
  - However, it has sometimes been suggested that it is desirable to have at least some overlap among the spatial units at which the pooling is performed, because it makes the approach less likely to overfit

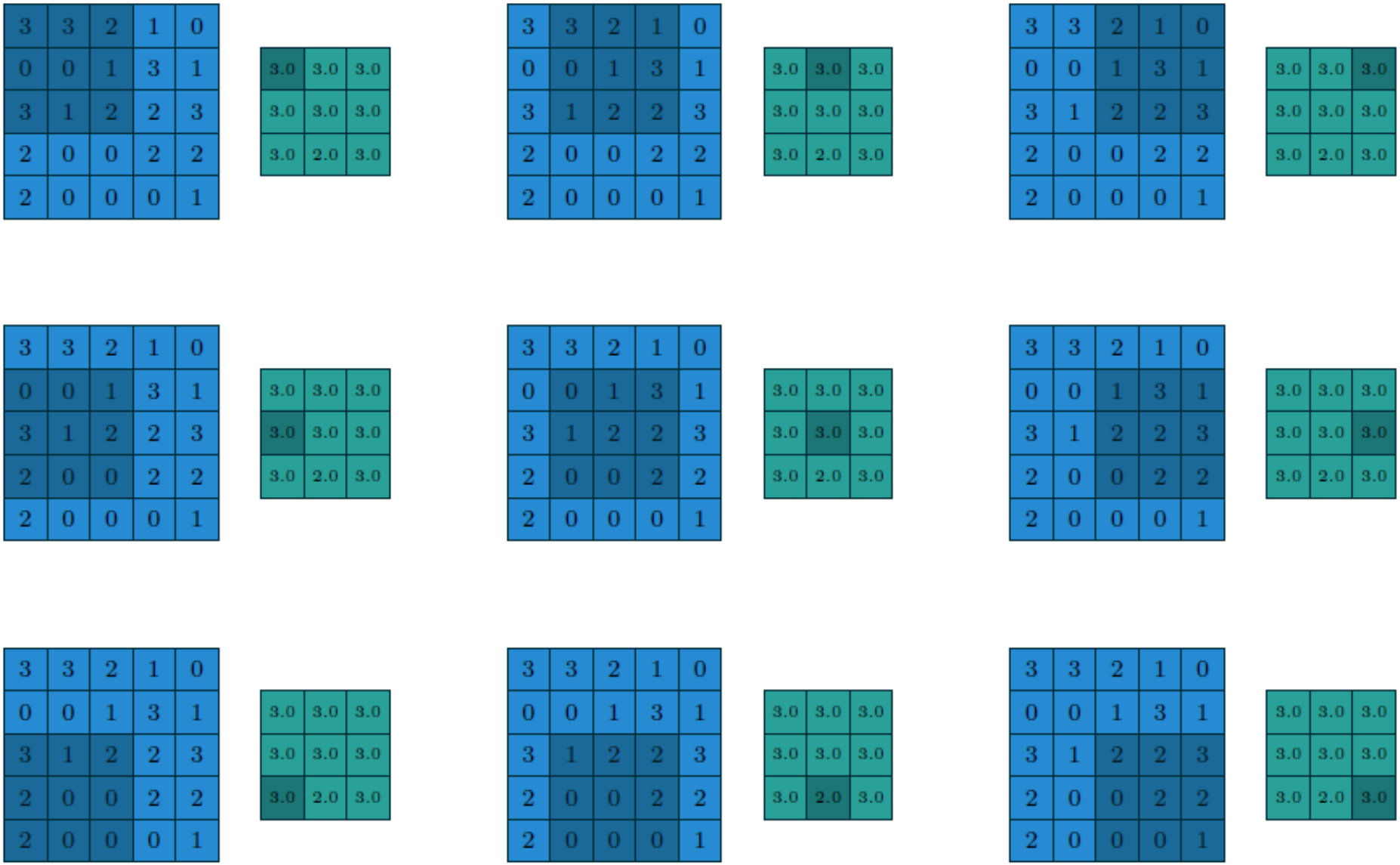
Max-pooling example



Avg-pooling example (output values of a 3x3 average pooling operation on a 5x5 input using 1x1 strides)

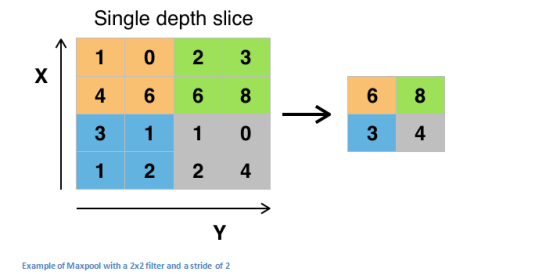


Max-pooling example (output values of a 3x3 max pooling operation on a 5x5 input using 1x1 strides)



Why pooling?

- The intuitive reasoning behind this layer is that once we know that a specific feature is in the original input volume (*there will be a high activation value*), its exact location is not as important as its relative location to the other features
- Aw we said earlier, this layer drastically reduces the spatial dimension (the length and the width change but not the depth) of the input volume
- This serves two main purposes:
  - The first is that the amount of parameters or weights is reduced (e.g., by 75% for a 2x2 filter with a stride equal to 2 on a 4x4 input), thus lessening the computation cost
  - The second is that it will control overfitting



Pooling

- It is common to use pooling with 2×2 filters and a stride of 2, when it is desired to reduce the spatial footprint of the activation maps
- Pooling results in (some) invariance to translation because shifting the image slightly does not change the activation map significantly.
- This property is referred to as *translation invariance*. The idea is that similar images often have very different relative locations of the distinctive shapes within them, and translation invariance helps in being able to classify such images in a similar way
  - For example, one should be able to classify a bird as a bird, irrespective of where it occurs in the image

Pooling

- Another important purpose of pooling is that it increases the size of the receptive field while reducing the spatial footprint of the layer because of the use of strides larger than 1
- Increased sizes of receptive fields are needed to be able to capture larger regions of the image within a complex feature in later layers
- Most of the rapid reductions in spatial footprints of the layers (and corresponding increases in receptive fields of the features) are caused by the pooling operations
- Convolutions increase the receptive field only gently unless the stride is larger than 1
- In recent years, it has been suggested that pooling is not always necessary. One can design a network with only convolutional and ReLU operations, and obtain the expansion of the receptive field by using larger strides within the convolutional operations
  - Therefore, there is an emerging trend in recent years to get rid of the max-pooling layers altogether
  - However, this trend has not been fully established and validated so far

Fully Connected Layers

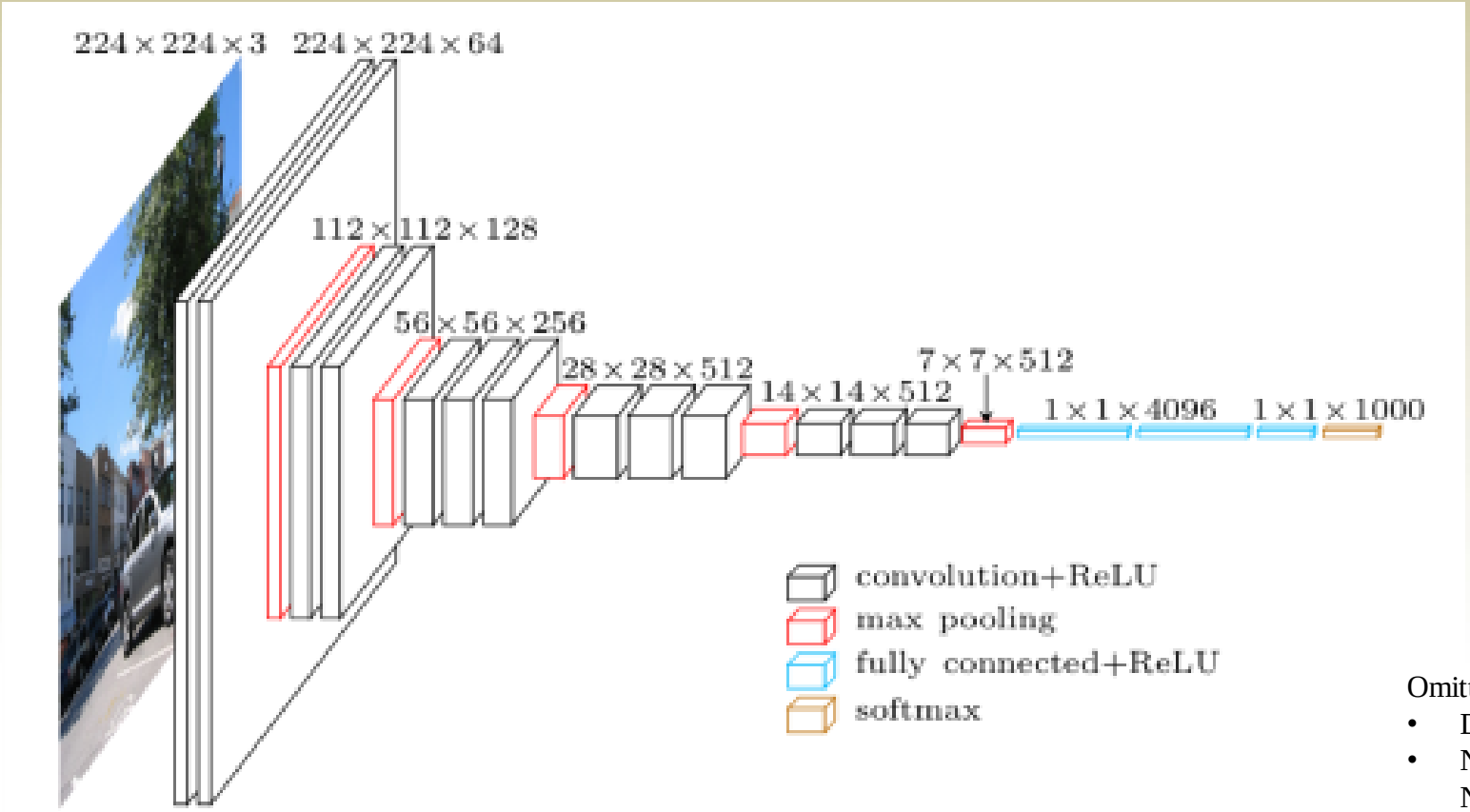
- Each feature in the final spatial layer is connected to each hidden state in the first fully connected layer
- This layer functions in exactly the same way as a traditional feed-forward network
  - In most cases, one might use more than one fully connected layer to increase the power of the computations towards the end
  - The connections among these layers are exactly structured like a traditional feed-forward network
  - Since the fully connected layers are densely connected, the vast majority of parameters lie in the fully connected layers
    - For example, if each of two fully connected layers has 4096 hidden units, then the connections between them have more than 16 million weights
    - Similarly, the connections from the last spatial layer to the first fully connected layer will have a large number of parameters.
- Even though the convolutional layers have a larger number of activations (and a larger memory footprint), the fully connected layers often have a larger number of connections (and parameter footprint)

Fully Connected Layers

- The reason that activations contribute to the memory footprint more significantly is that the number of activations are multiplied by mini-batch size while tracking variables in the forward and backward passes of backpropagation
- These trade-offs are useful to keep in mind while choosing neural-network design based on specific types of resource constraints (e.g., data versus memory availability)
- It is noteworthy that the nature of the fully-connected layer can be sensitive to the application at hand
  - For example, the nature of the fully-connected layer for a classification application would be somewhat different from the case of a segmentation application
- The output layer of a convolutional neural network is designed in an application-specific way. In the following, we will consider the representative application of classification. In such a case, the output layer is fully connected to every neuron in the penultimate layer, and has a weight associated with it
  - One might use the logistic, softmax, or linear activation depending on the nature of the application (e.g., classification or regression).

Overall architecture of a CNN

- Max-pooling layers are interleaved with the convolutional/ReLU layers, although the former occurs less frequently in deep architectures
  - This is because pooling drastically reduces the spatial size of the feature map, and only a few pooling operations are required to reduce the spatial map to a small constant size



Omitted:

- Dropout layers
- Network in Network layers