Department of Mathematics
and Computer Science

Advanced Machine Learning

Convolutional Networks

Lukas Galke

Spring 2026

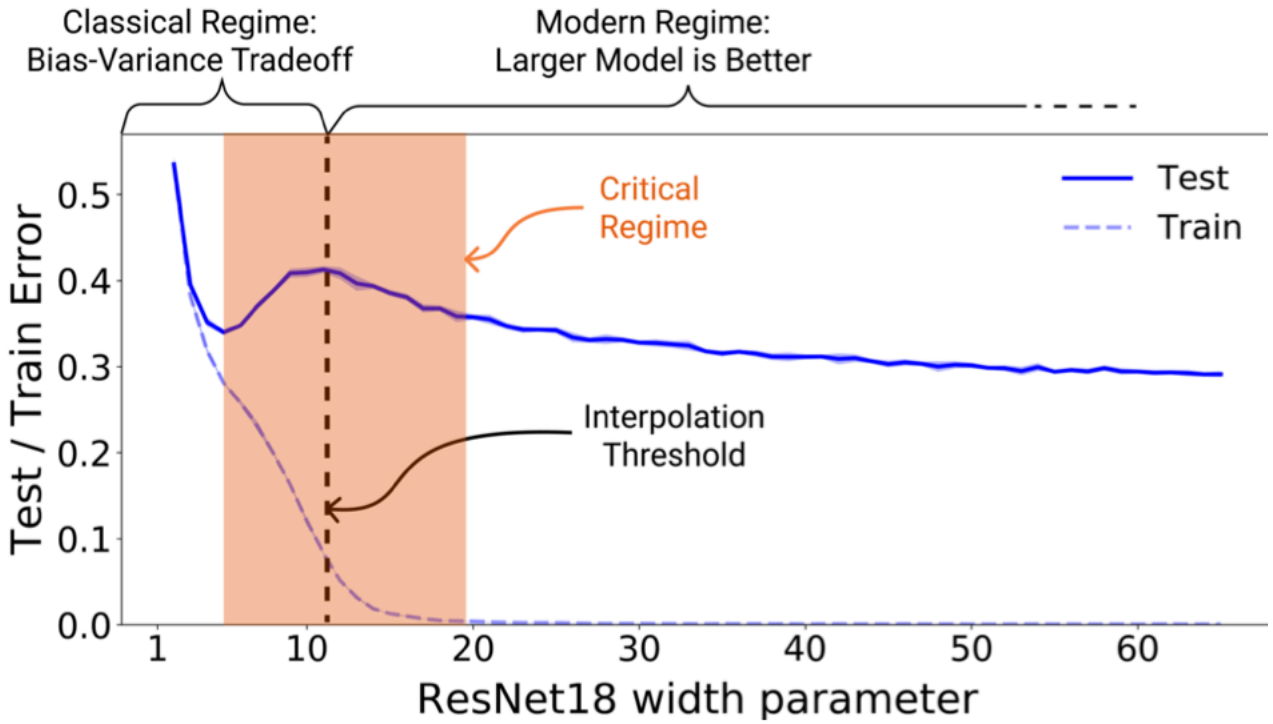
Keywords

- Convolution
- Parameter Sharing
- Pooling
- Normalization
- Residual connections

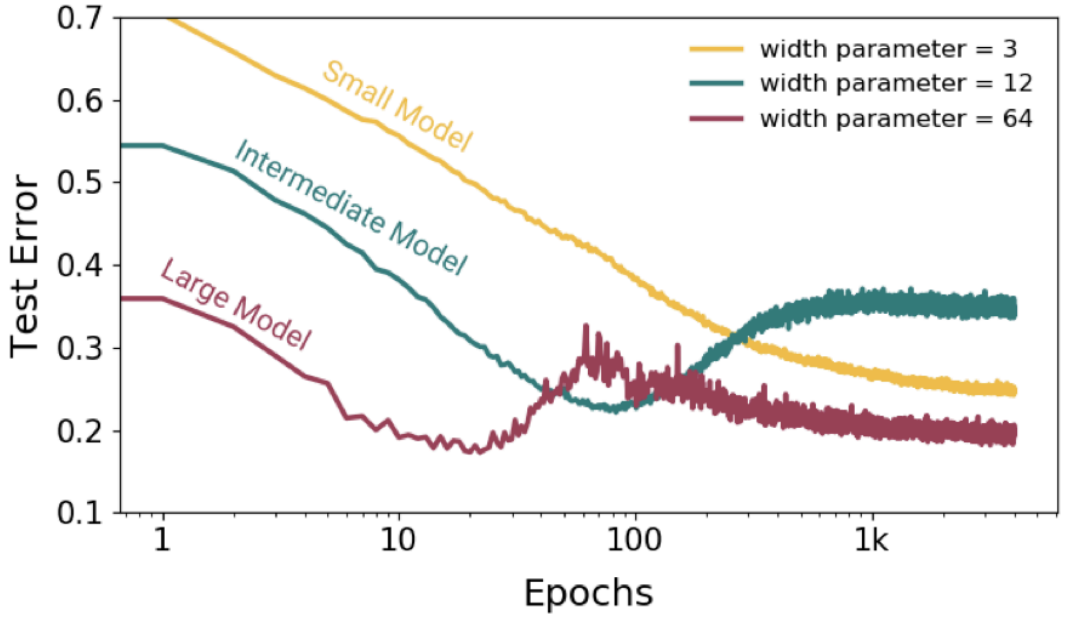
What have we learned so far?

- We know how to handle normal feed-forward networks
  - How to design them
  - How to define the error function
  - How to train them
- BUT: So far, our inputs have been rather small, i.e., we had just a few dimensions as input
- What if we now want to treat large inputs, like a full-blown high-resolution image

Recap: Deep Double Descent



~Model Capacity



~Training time

Source: Deep Double Descent,
https://arxiv.org/abs/1912.02292

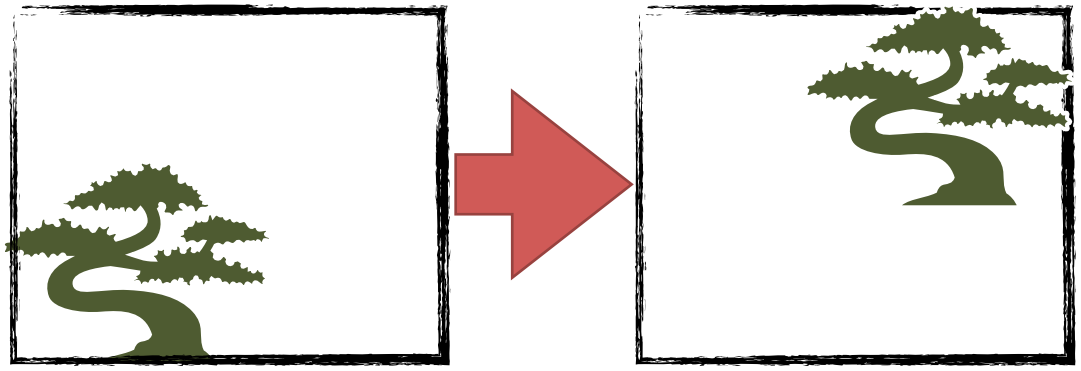
Recap from last lecture (Regularization)

- Best way to improve your model is to gather more data
- If that's not possible, **data augmentation** depending on domain knowledge (e.g., invariance)
- An L2 penalty (weight decay) may help to generalize better, but it shouldn't be too strong
- An L1 penalty on the weights nudges your model to be sparse (higher fraction of weights being 0)
- Early-stopping can help but be careful to not stop too early (use a very high patience to be safe)
- Ensembles Bagging (averaging many models) usually works but is very expensive with large neural nets
- **Dropout** (randomly zeroing out activations) is an efficient way to implement bagging with neural nets
- Adversarial training makes your model more robust to slight variations of the input

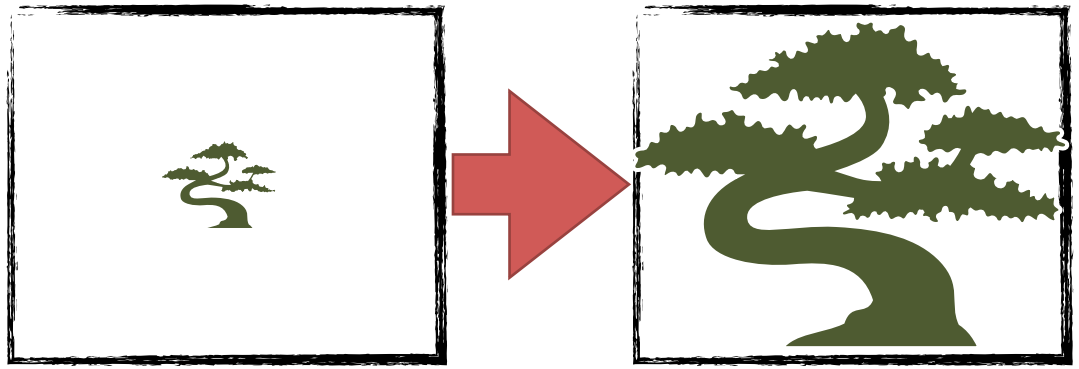
Recap: Symmetry and Invariance

Invariance:
S(I) = S(T(I))

Translation invariance



Scale invariance



Invariance: Model output is unchanged with respect to some transformation.

Symmetries: The transformations that leave other properties unchanged

Recap: Equivariance

S(T(I)) = T(S(I))

Model output changes in the same way with respect to some transformation

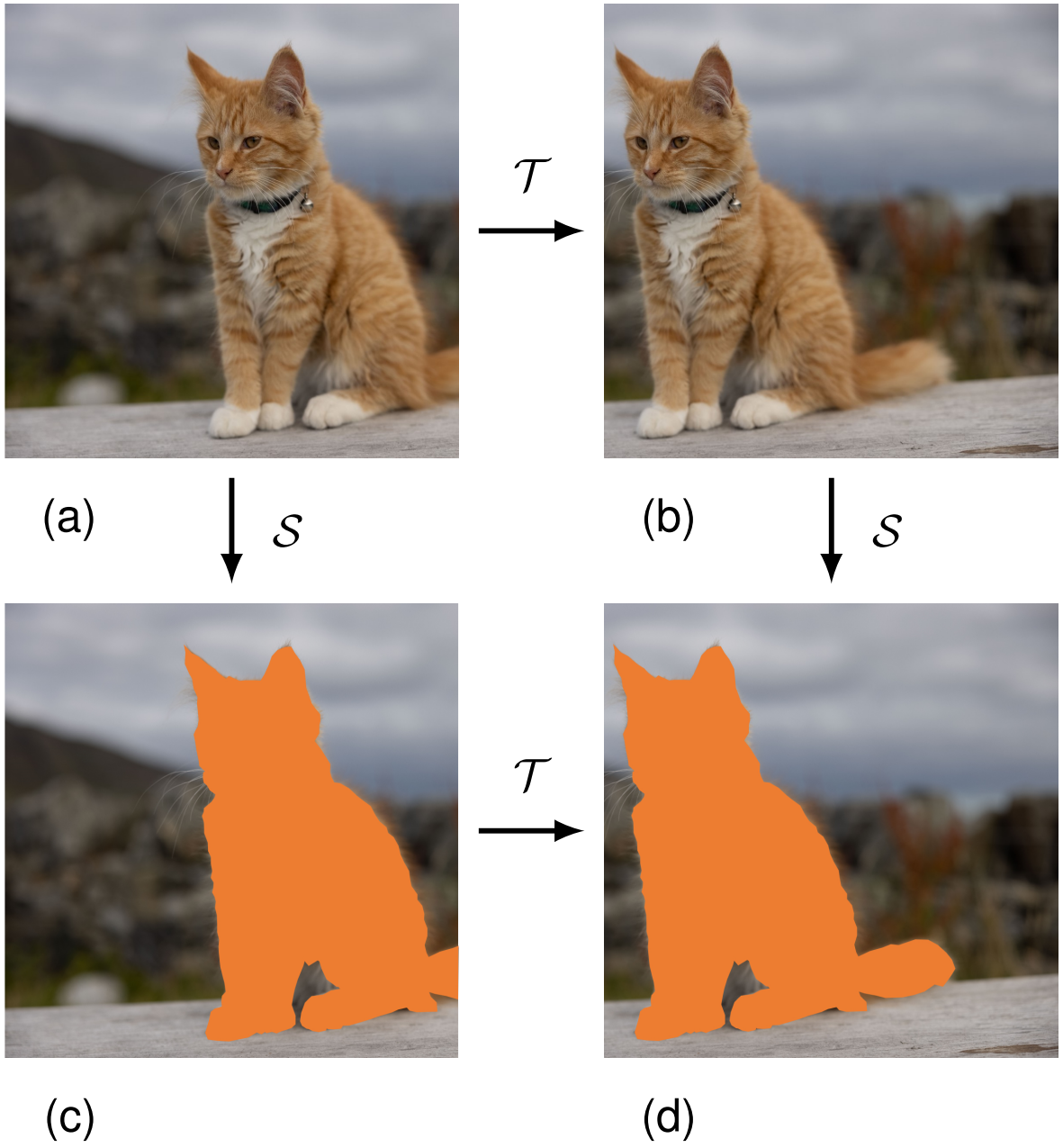


Figure from Bishop & Bishop (2024)

Problems when using large inputs

- Assume a standard mobile phone image with 8 Mega pixels
- Each pixel consists of 3 color channels, i.e., we end up with 24 million input "dimensions"
- Now we want to detect a face. We build a fully-connected layer where each node represents the presence of a certain feature, for instance:
  - One for eyes
  - One for hair
  - One for lips
  - …

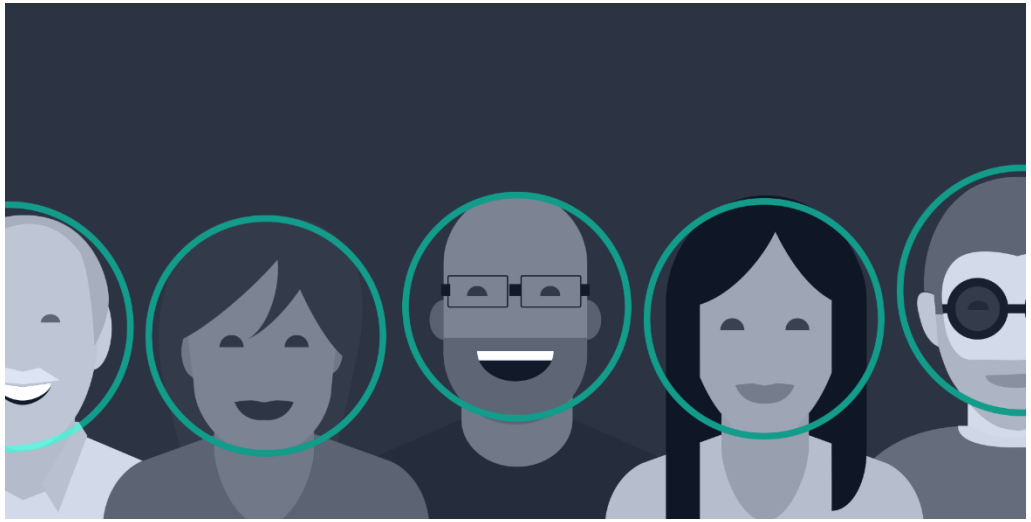
Problems when using large inputs

- Now we want to detect a face. We build a fully-connected layer where each node represents the presence of a certain feature, for instance:
  - One for eyes
  - One for hair
  - One for lips
  - …
- So that means even for a moderate number of features we aim to extract (in case we could so clearly define the purpose of the hidden layer). Extracting just 10 features, we end up with

240 million weights

Problems when using large inputs

- Even if we could just train those networks, we have another problem:
  - Images of faces for example might look vastly different
  - The head can be tilted, rotated, viewed from the side
  - But we want our network to be able to detect, e.g., an eye regardless of the position of the eye in the image:



Problems when using large inputs

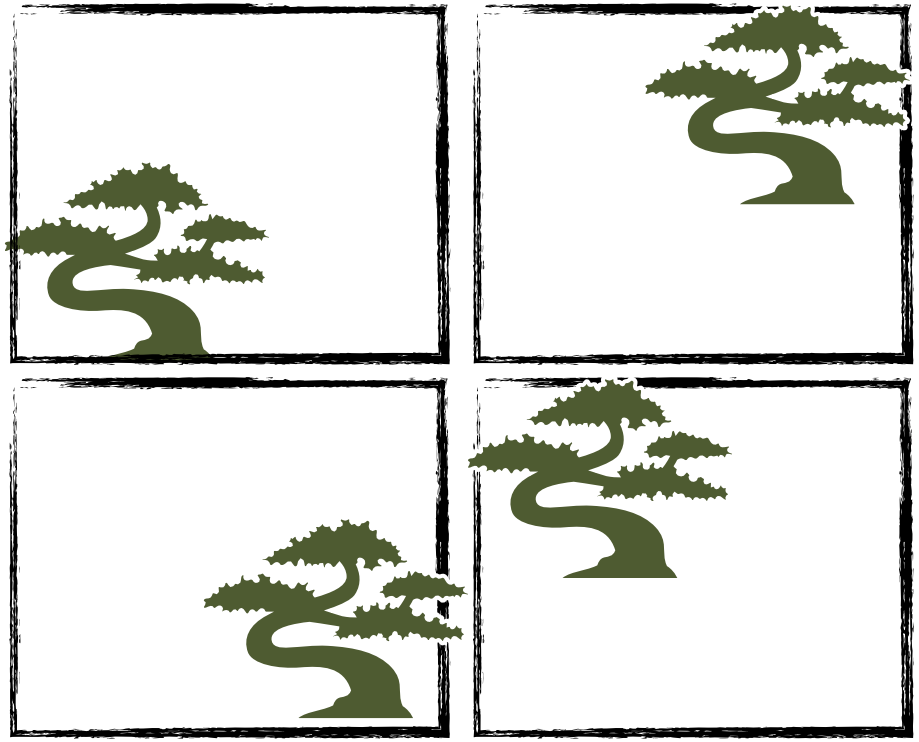
- Bottomline is: If a fully connected layer has never seen an eye at a certain position in the image, it had never the possibility to train the weights accordingly
- We would not only need to train an insane amount of data, but we additionally would also need to train with an insane amount data
- **We need something else which**
  - **keeps the number of weights in check**
  - **is translation invariant, i.e., we can detect the presence of a feature regardless where it is located within the feature**

The key ideas we want to exploit

- **Image statistics are translation invariant (objects and viewpoint translates)**
  - build this translation invariance into the model (rather than learning it)
  - tie lots of the weights together in the network
  - reduces number of parameters
- **Expect learned low-level features to be local (e.g. edge detector)**
  - build this into the model by allowing only local connectivity
  - this reduces the numbers of parameters further
- **Expect high-level features learned to be coarser**
  - build this into the model by subsampling more and more up the hierarchy
  - reduces the number of parameters again

Another perspective: Sample Complexity

➢ **Sample Complexity**: The number of training example it needs to successfully learn a target function.



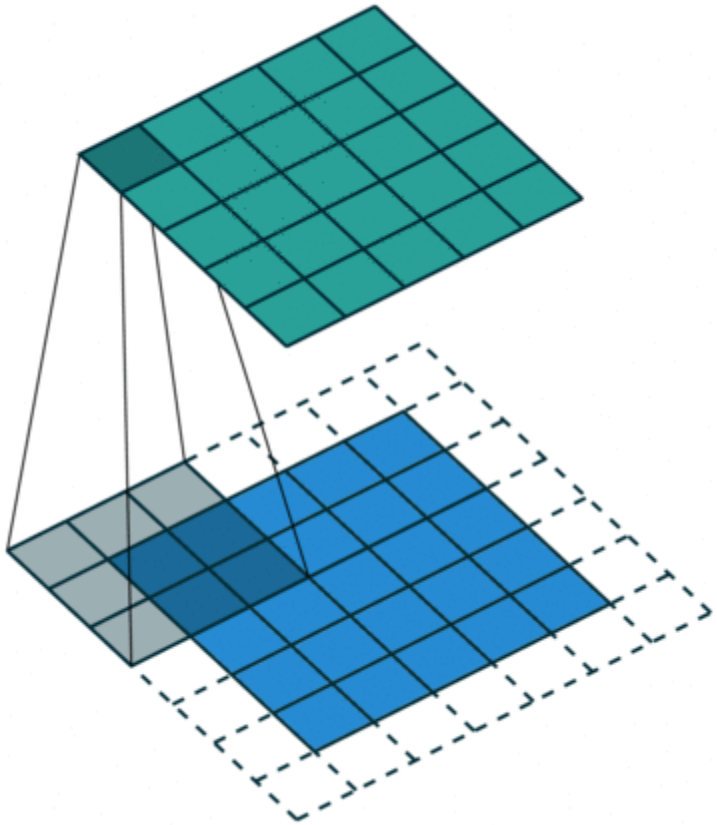
Not translation invariant
High sample complexity
Many data points needed
to generalize

Translation invariant
Low sample complexity
Little data points needed
to generalize

Keywords

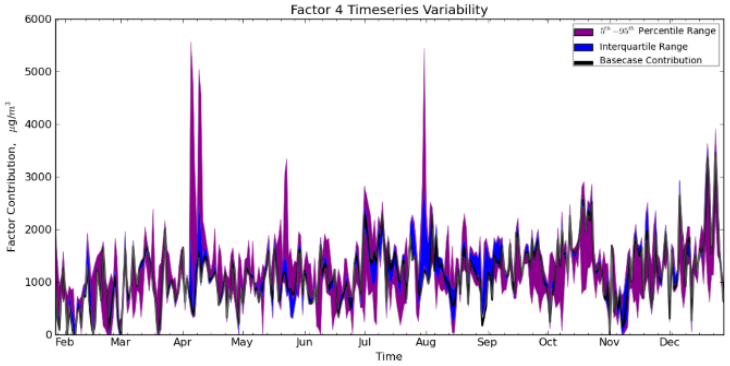
- **Convolution**
- Parameter Sharing
- Pooling
- Normalization
- Residual connections

Convolution Visualized

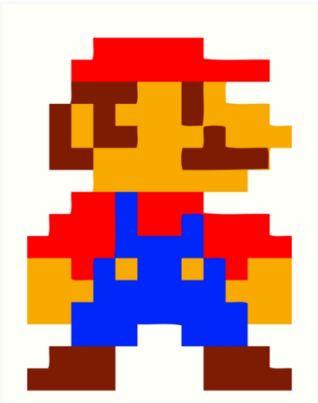


Convolutional Networks

- Convolutional neural networks (CNNs) are a specialized kind of neural network
  - Neural networks that use **convolution** in place of general matrix multiplication in at least one of their layers
- Well suited for data with a grid-like topology
- But: Convolution can be viewed as multiplication by a matrix



E.g. Ex: time-series data, which is a 1-D grid, taking samples at intervals



More obviously: Image data, which are 2-D grid of pixels

What is Convolution?

- Convolution is an operation on two functions of a real-valued argument
- Examples of the two functions
  - Tracking location of a spaceship by a laser sensor
    - A laser sensor provides a single output $x(t)$, the position of spaceship at time t
  - w a function of a real-valued argument
    - If laser sensor is noisy, we want a weighted average that gives more weight to recent observations
    - Weighting function is $w(a)$ where a is age of measurement
- Convolution is the smoothed estimate of the position of the spaceship

$$s(t) = \int x(a)w(t-a)da$$

What is Convolution?

- One-dimensional continuous case
  - Input $f(t)$ is convolved with a kernel $g(t)$

$$\big(f * g\big)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

- Note that $\big(f * g\big)(t) = (g * f)(t)$
- Terminology:
  - The first argument (here the function f) to the convolution is often referred to as the **input** and
  - The second argument (in this example, the function g) as the **kernel**.
  - The output is sometimes referred to as the **feature map**.

Convolution with Discrete Variables

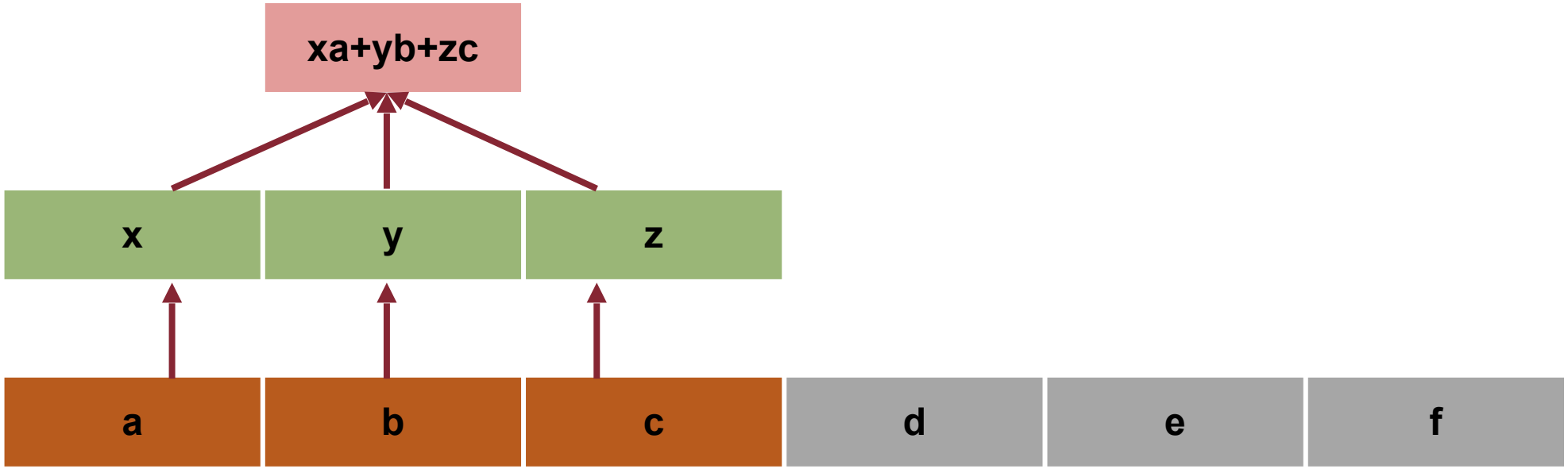
- Laser sensor may only provide data at regular intervals
- Time index t can take on only integer values
  - x and w are defined only on integer t

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

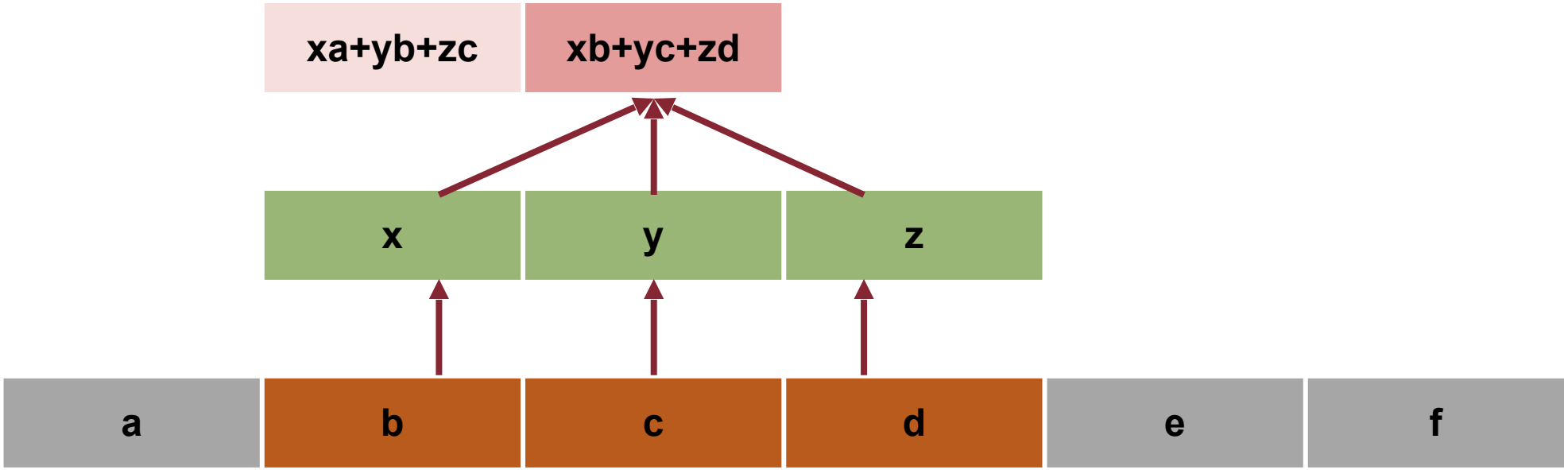
- In ML applications, input is a multidimensional array of data and the kernel is a multidimensional array of parameters that are adapted by the learning algorithm

- Input and kernel are explicitly stored separately

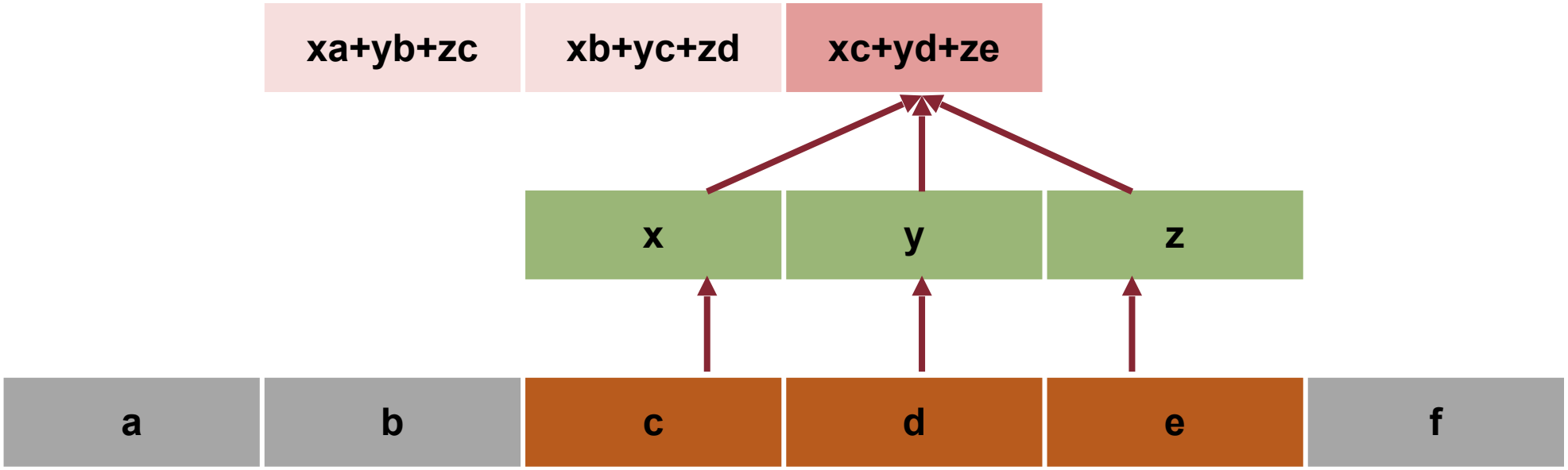
Discrete 1-Dimensional Case



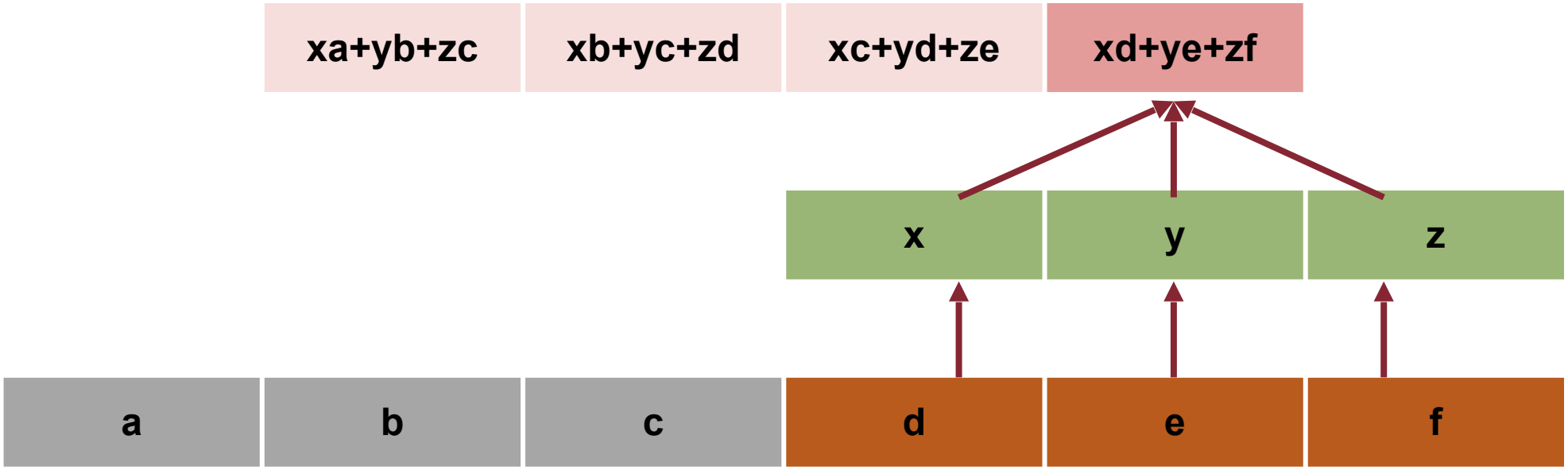
Discrete 1-Dimensional Case



Discrete 1-Dimensional Case

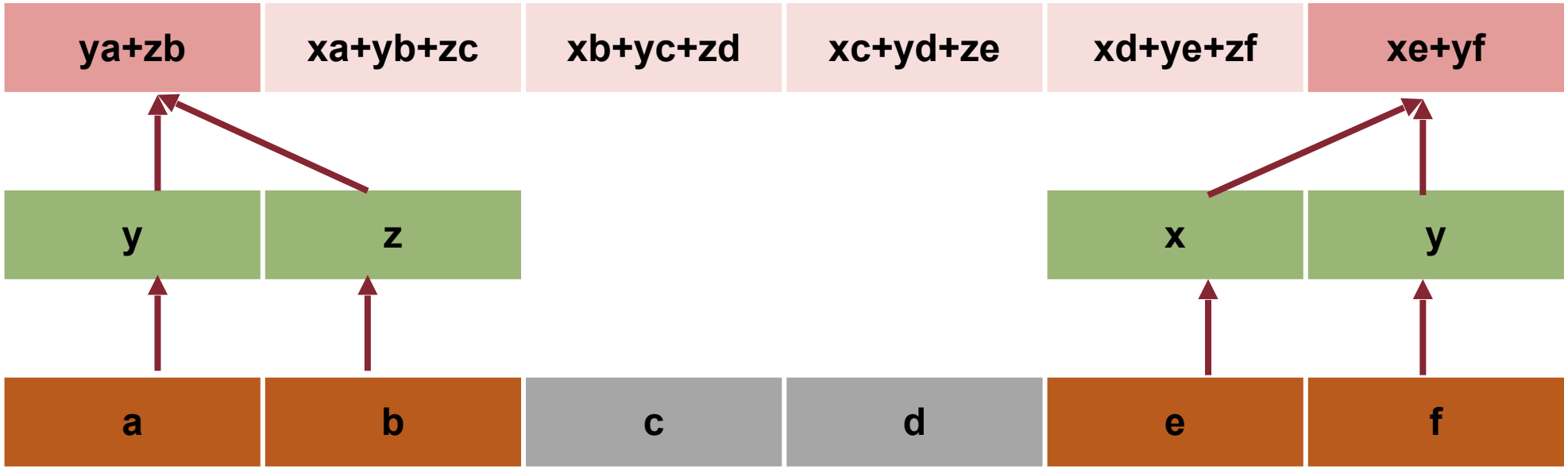


Discrete 1-Dimensional Case

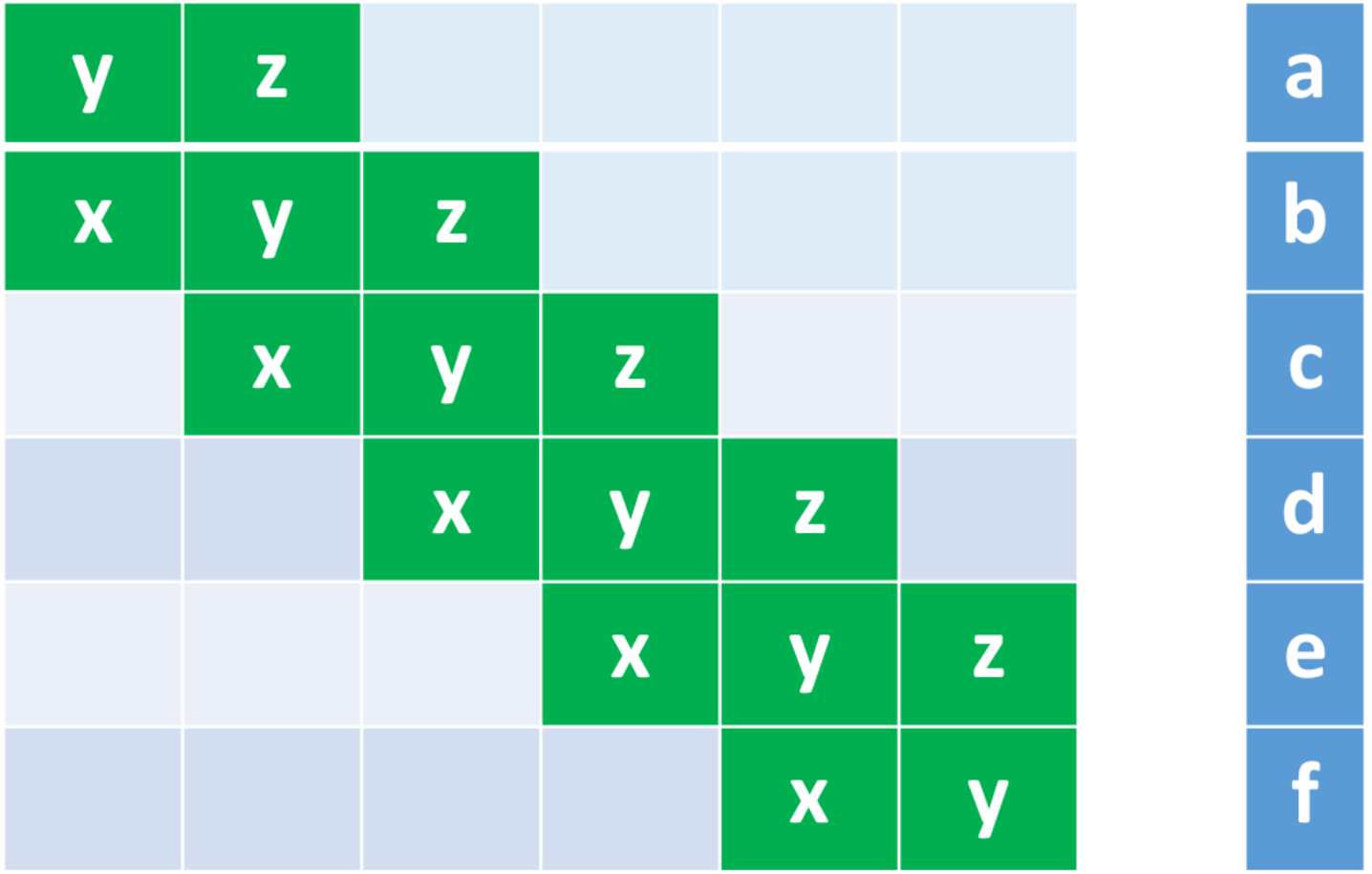


Discrete 1-Dimensional Case: Border Cases

- Border require special treatment
- Most commonly they are ignored
- Sometimes a smaller kernel is used then

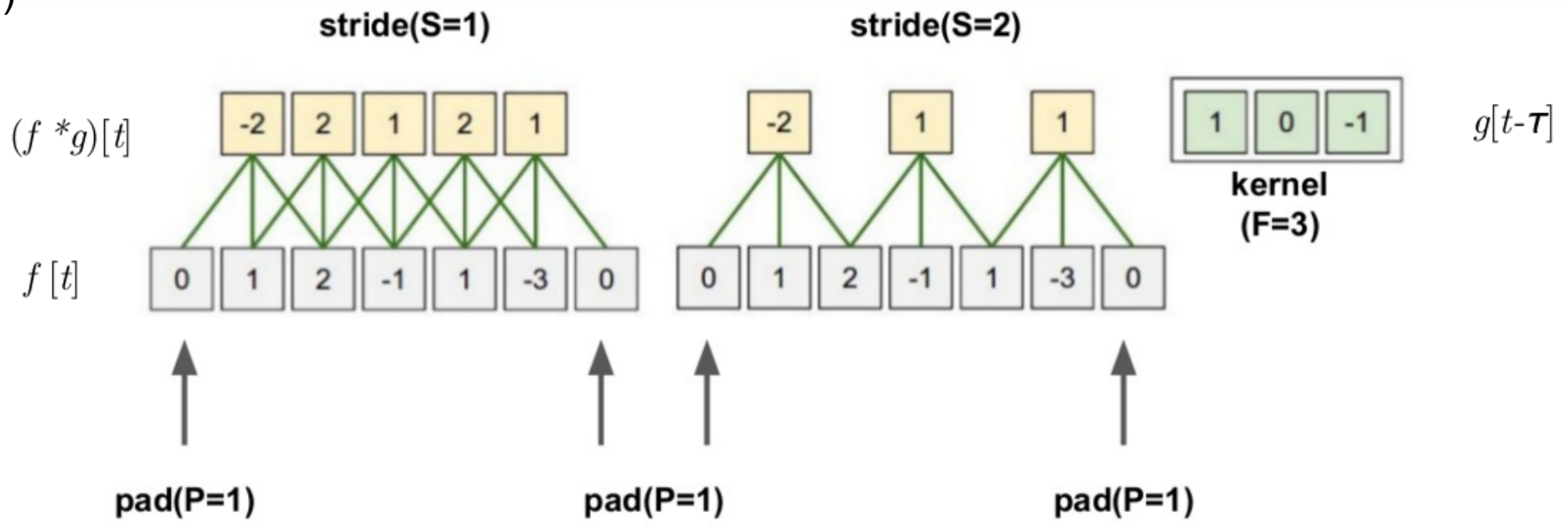


Seen as a Matrix Multiplication



Parameters of Convolution

- Kernel Size (F)
- Padding (P)
- Stride (S)



Two-dimensional Convolution

➢ If we use a 2D image I as input and use a 2D kernel K we have

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

➢ Since convolution is commutative, we can also write:

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

➢ Commutativity arises because we have flipped the kernel relative to the input

➢ As m increases, index to the input increases, but index to the kernel decreases

Cross-Correlation

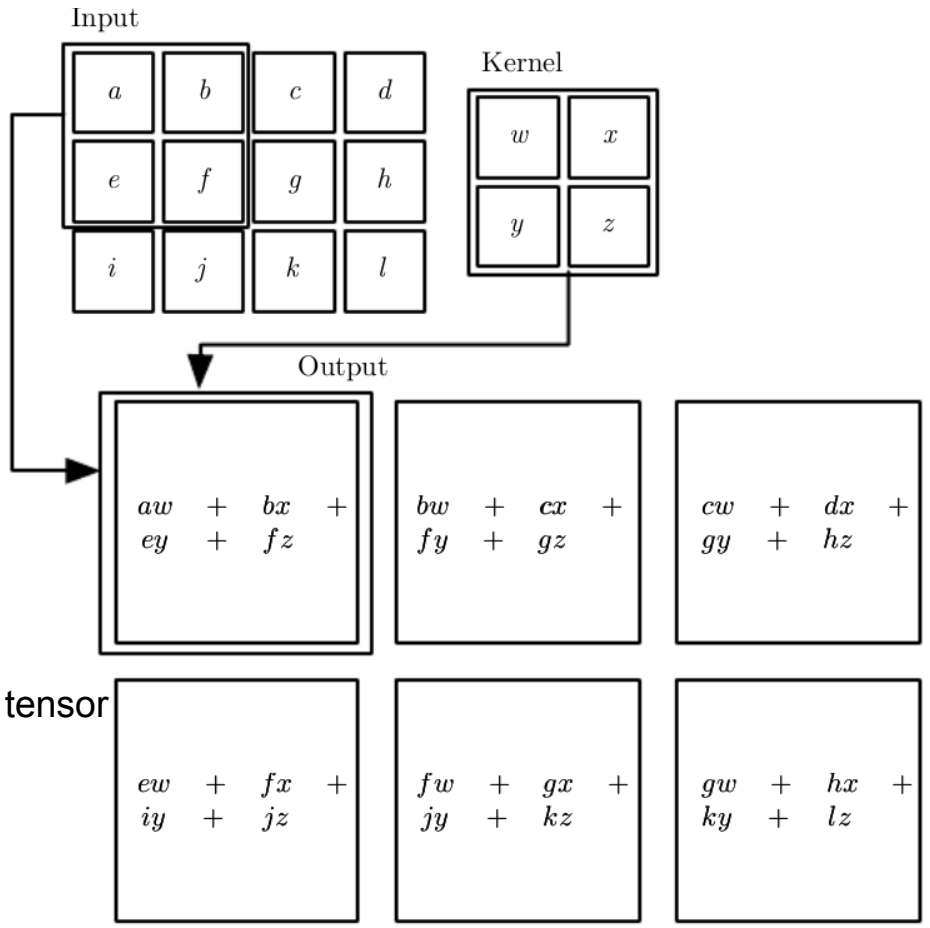
➢ Same as convolution, but without flipping the kernel

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

➢ Both referred to as convolution, whether kernel is flipped or not

➢ The learning algorithm will learn appropriate values of the kernel in the appropriate place

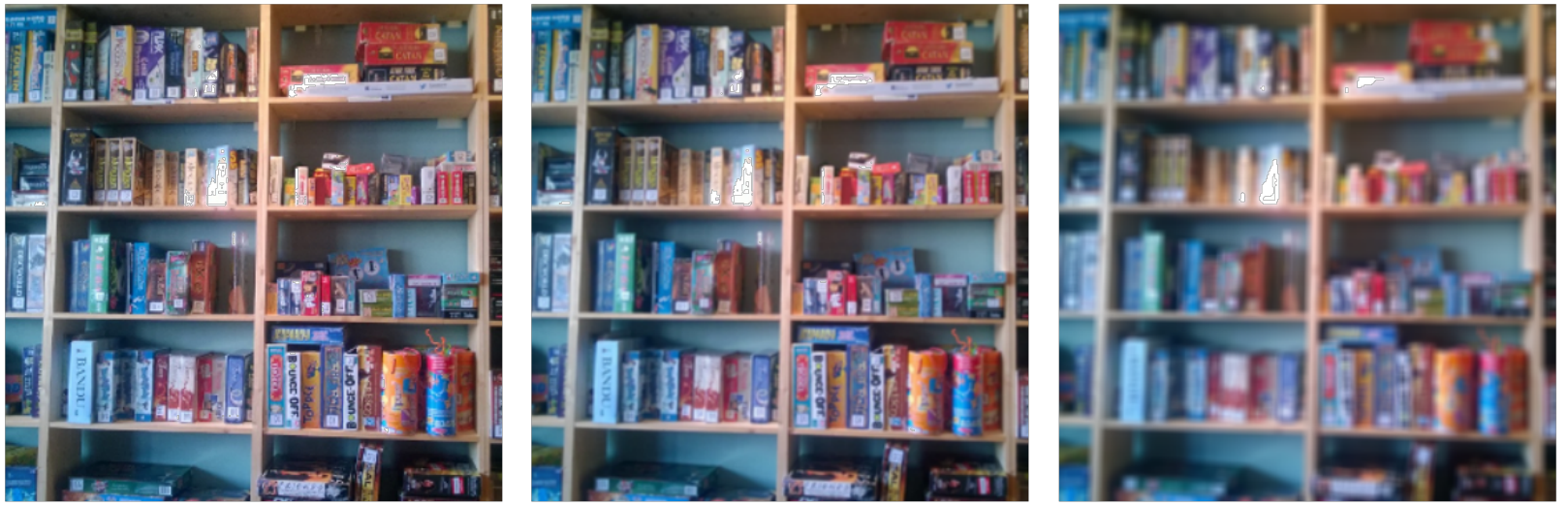
Example of 2D convolution



- Convolution without kernel flipping applied to a 2D tensor
- Output is restricted to case where kernel is situated entirely within the image
- Arrows show how upper-left of input tensor is used to form upper-left of output tensor

Classic Examples: Blurring Filter

$$K_{box} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \hat{K}_{box} = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$



the same image without any filtering, after a 3x3 box blur and after a 9x9 box blur

➢ https://www.taylorpetrick.com/blog/post/convolution-part3

Classic Examples: Edge Detection

➢ Simple

$$\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}$$

➢ Prewitt

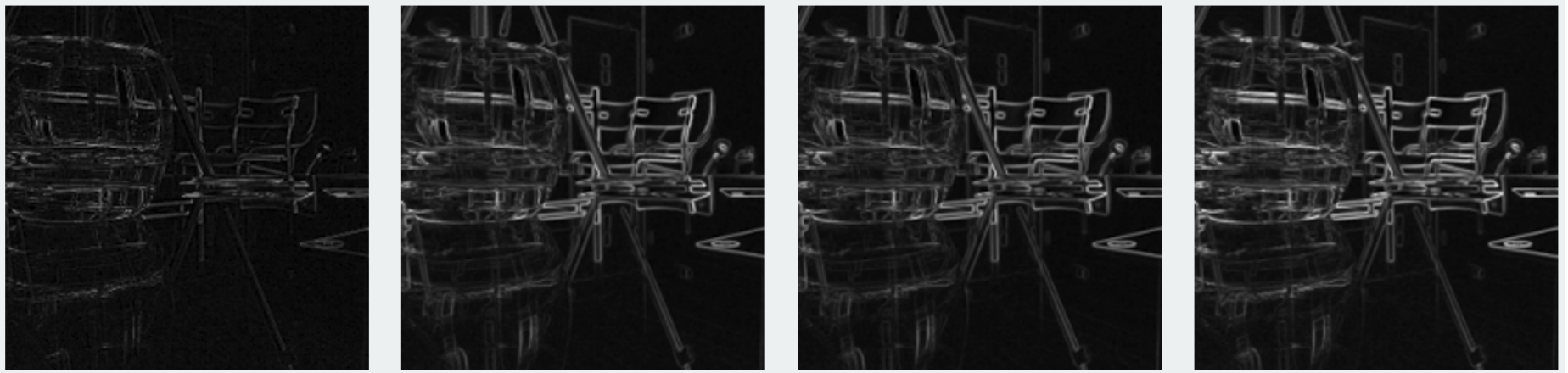
$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G_y} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix} * \mathbf{A}$$

➢ Sobel

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

➢ Kirsch (8 directions)

$$\mathbf{g}^{(1)} = \begin{bmatrix} +5 & +5 & +5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}, \mathbf{g}^{(2)} = \begin{bmatrix} +5 & +5 & -3 \\ +5 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}, \mathbf{g}^{(3)} = \begin{bmatrix} +5 & -3 & -3 \\ +5 & 0 & -3 \\ +5 & -3 & -3 \end{bmatrix}$$



comparison of simple, sobel, prewitt and kirsch edge detection filters

Classic Examples: Sharpen

$$K_{sharp} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} * amount$$

comparison of unfiltered image and sharpened images with amount=2 and amount=8

Classic Examples: Sharpen

Main Idea of convolution: Learn these weights

$$K_{sharp} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} * amount$$

comparison of unfiltered image and sharpened images with amount=2 and amount=8

Motivation for using convolution networks

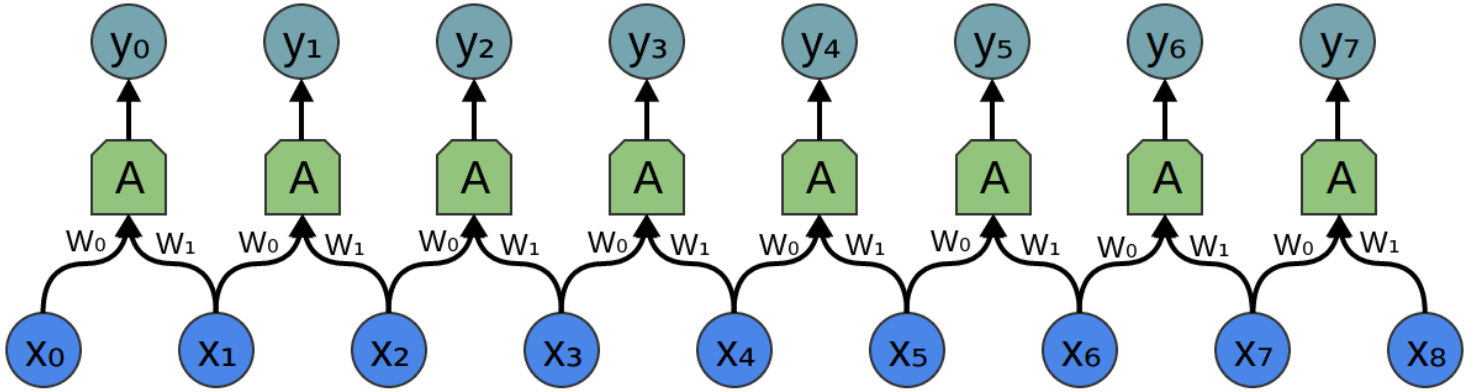
➢ Convolution leverages three important ideas to improve ML systems:

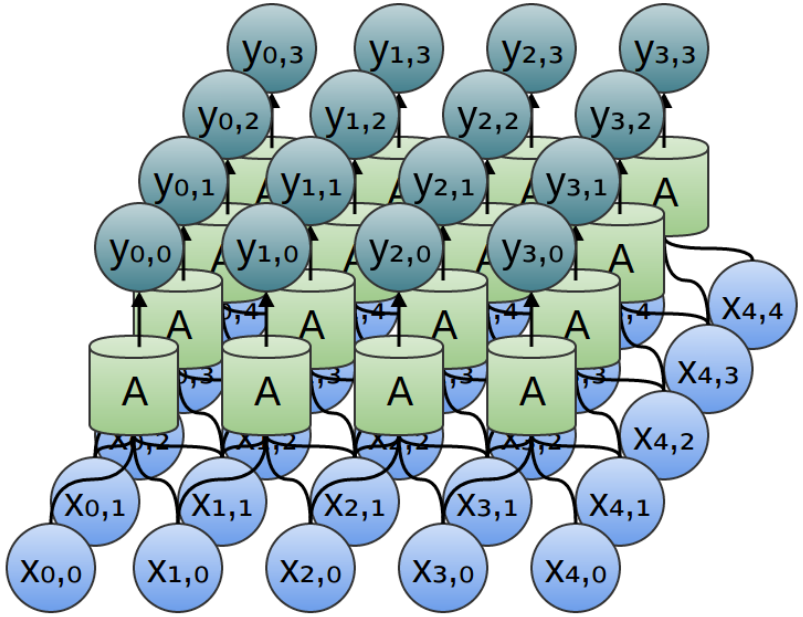
1. Sparse interactions
2. Parameter sharing
3. Equivariant representations

➢ Convolution also allows for working with inputs of variable size

Sparse connectivity due to Convolution

➢ Instead of learning a full matrix, only a couple of weights of the kernel need to be learned



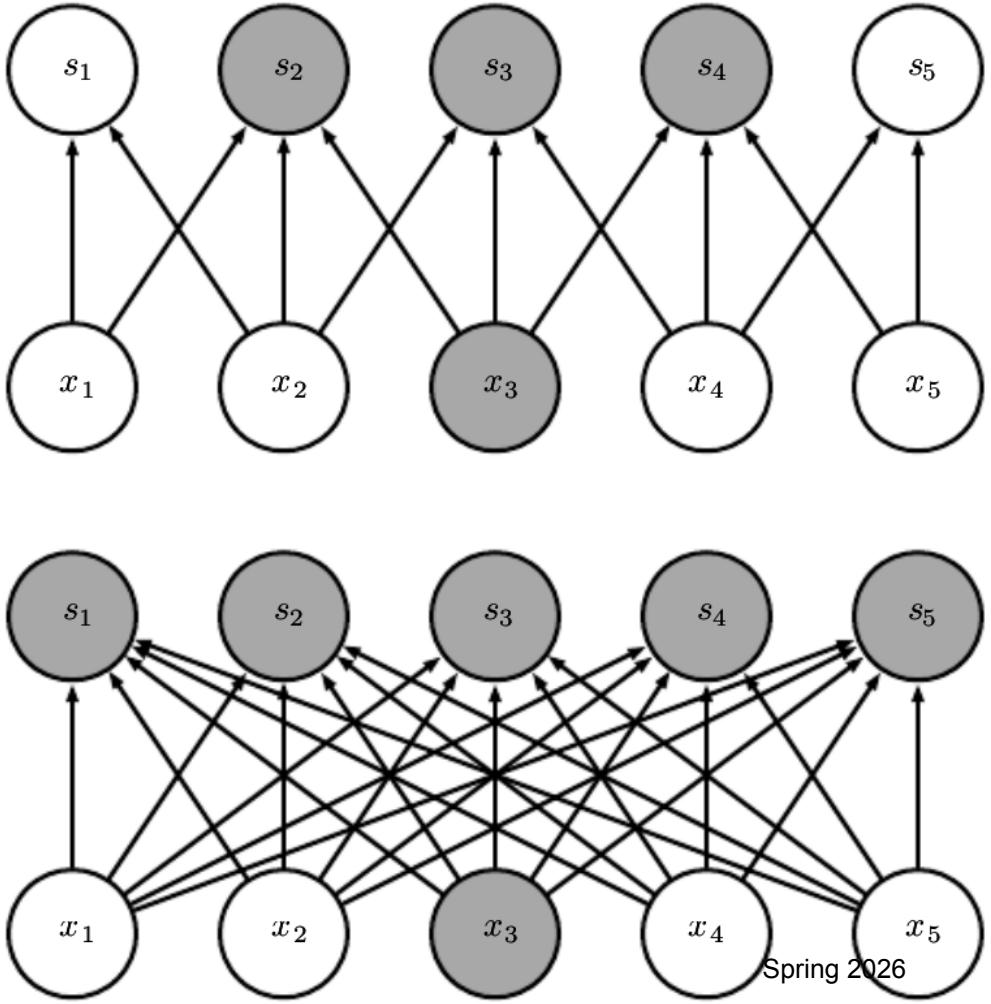


Sparse Connectivity: Input Perspective

Highlight one input x_3 and output units s affected by it

➢ Top: when s is formed by convolution with a kernel of width 3, only three outputs are affected by x_3

➢ Bottom: when s is formed by matrix multiplication connectivity is no longer sparse
=> All outputs are affected by x_3

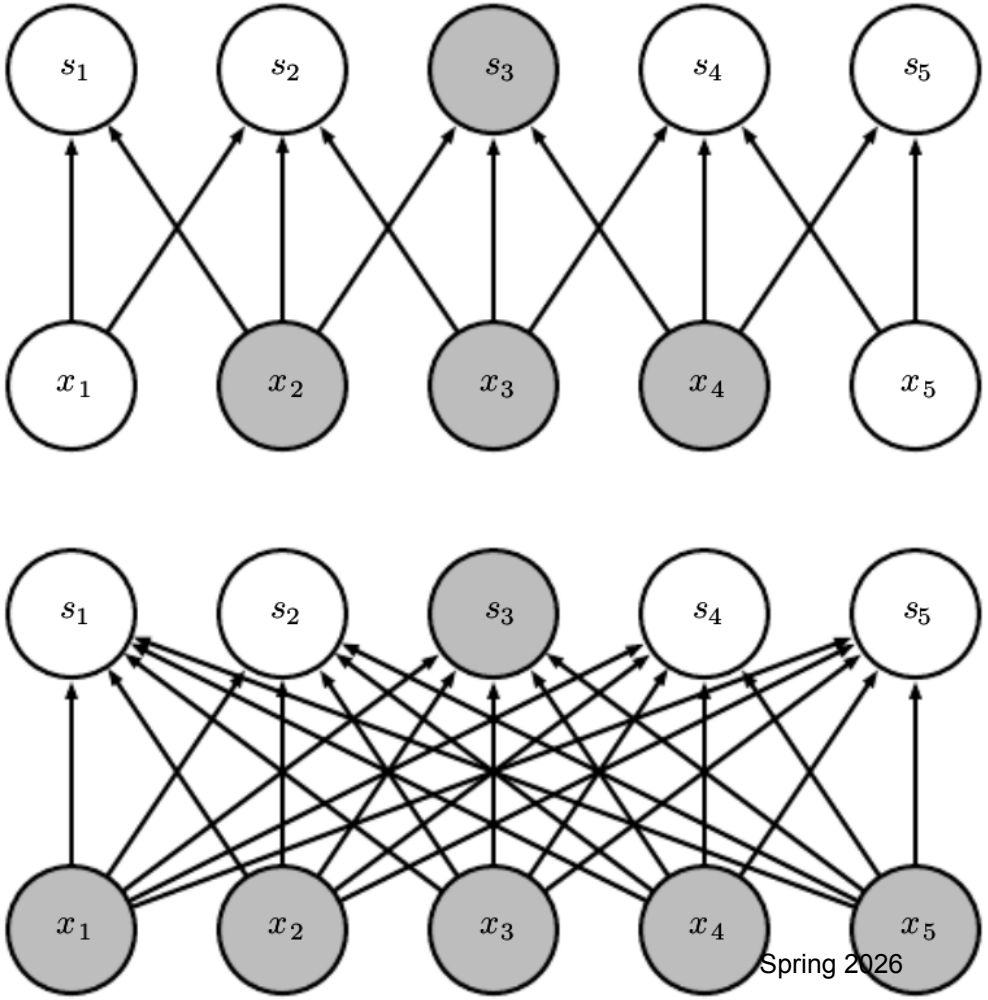


Sparse Connectivity: Output Perspective

Highlight one output s_3 and input units x affecting it:

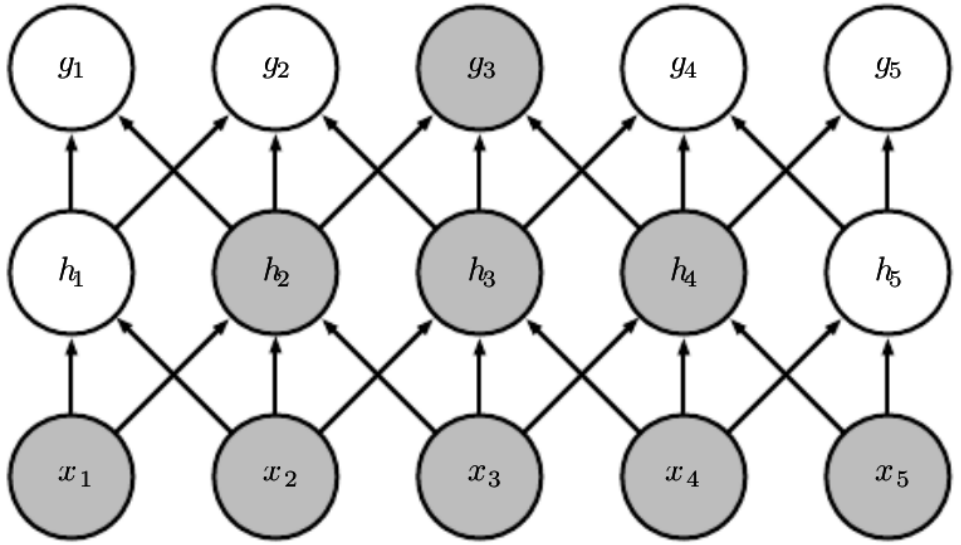
➢ Top: when s_3 is formed by convolution with a kernel of width 3, only three inputs are defining output s_3

➢ Bottom: when s_3 is formed by a matrix multiplication connectivity is no longer sparse
=> All inputs are defining output s_3



Performance with Reduced Connections

- It is possible to obtain good performance while keeping k several magnitudes lower than m
- In a deep neural network, units in deeper layers may indirectly interact with a larger portion of the input: Depth is the key
- This allows the network to efficiently describe complicated interactions between many variables from simple building blocks that only describe sparse interactions



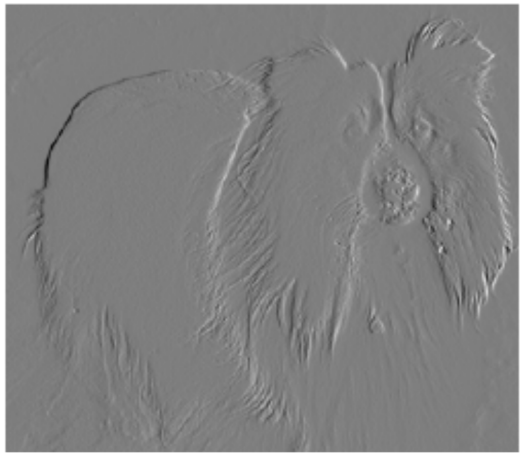
Keywords

- Convolution
- **Parameter Sharing**
- Pooling
- Normalization
- Residual connections

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 never revisited
- Parameter sharing is synonymous with tied weights
  - Value of the weight applied to one input is tied to a weight applied elsewhere
- In a Convolutional net, each member of the kernel is used in every position of the input (boundaries might be different)

Example: Efficiency of Convolution for Edge Detection

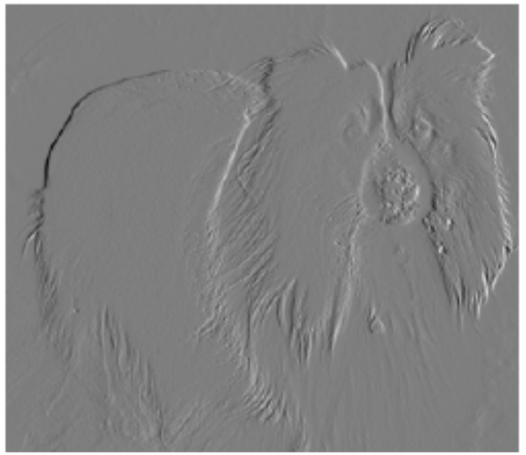


- Image on right formed by taking each pixel of input image and subtracting the value of its neighboring pixel on the left
- Input image is 320x280, output is 319x280
- Computational effort:

$$319 \cdot 320 \cdot 3 = 267{,}960 \text{ flops}$$

  - Each output pixel is calculated by a convolutional kernel with 2 entries, i.e. 2 multiplications, one add

Example: Efficiency of Convolution for Edge Detection



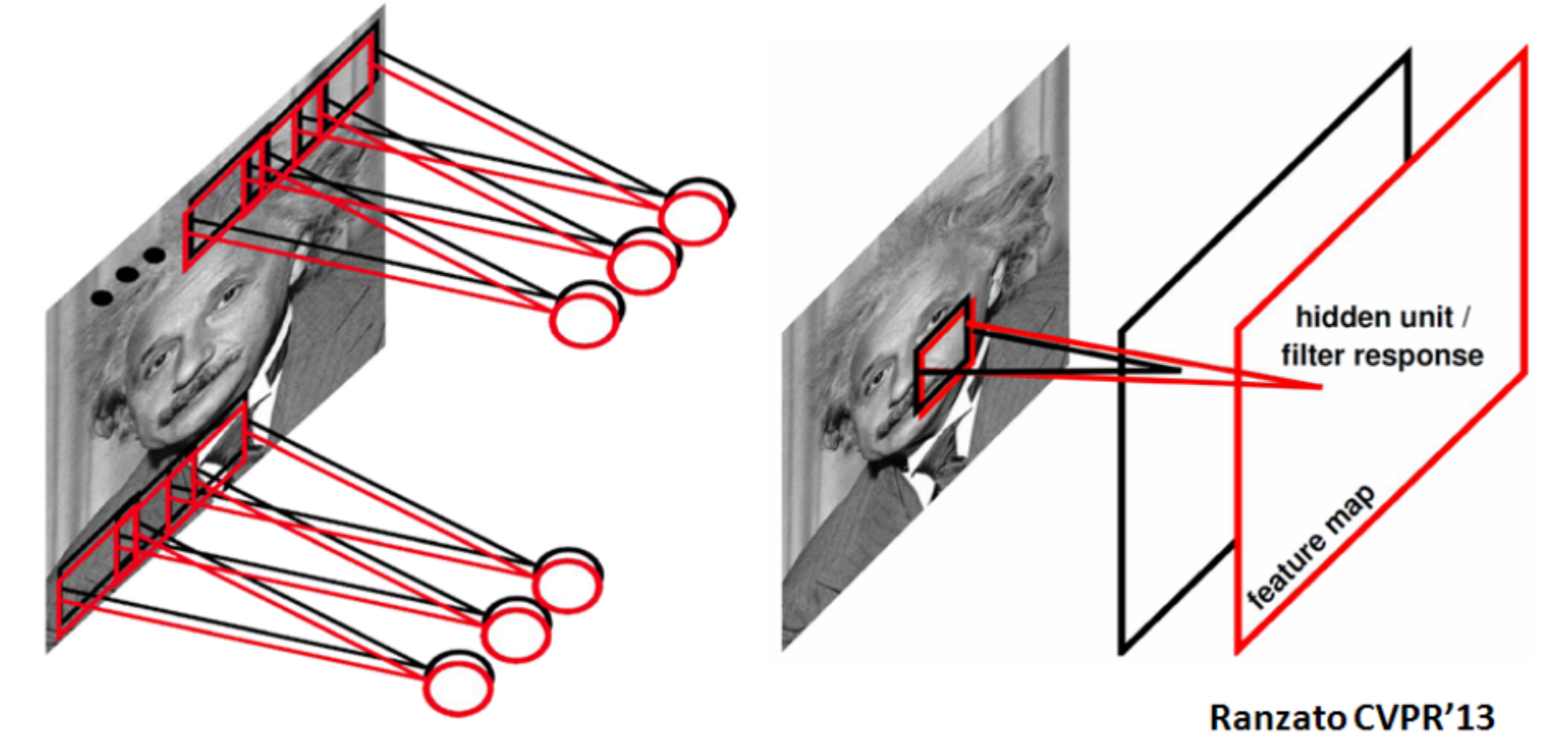
➢ Same operation with a full matrix with

$320 \cdot 280 \cdot 319 \cdot 280 = 8$ billion entries

➢ Storing the matrix (double vals): 8 billion * 8byte $\approx$ 60 Gbyte

➢ Performing the calculation: roughly 60,000 times less efficient computationally

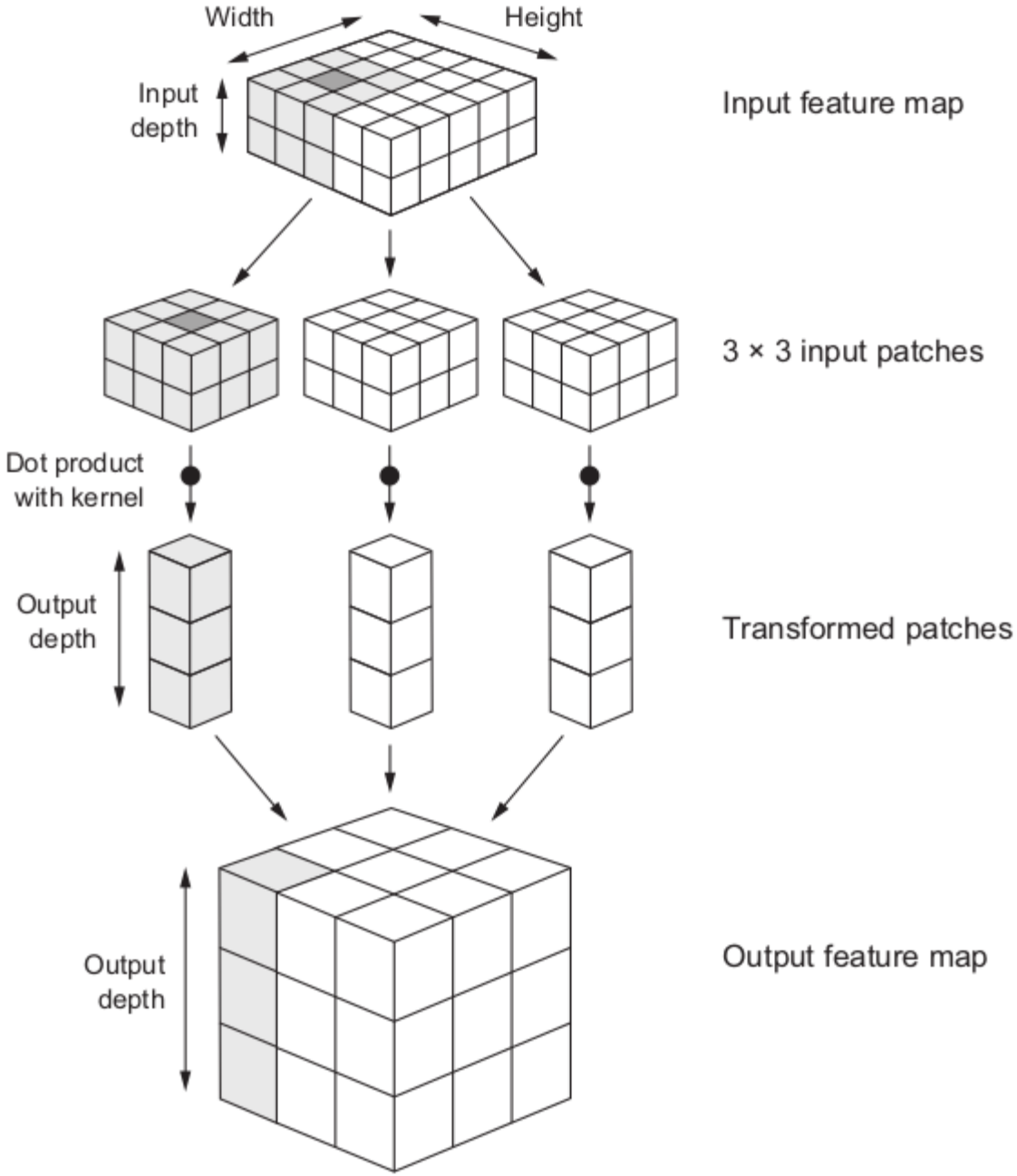
Depth of the Output Volume



Depth of the Output Volume

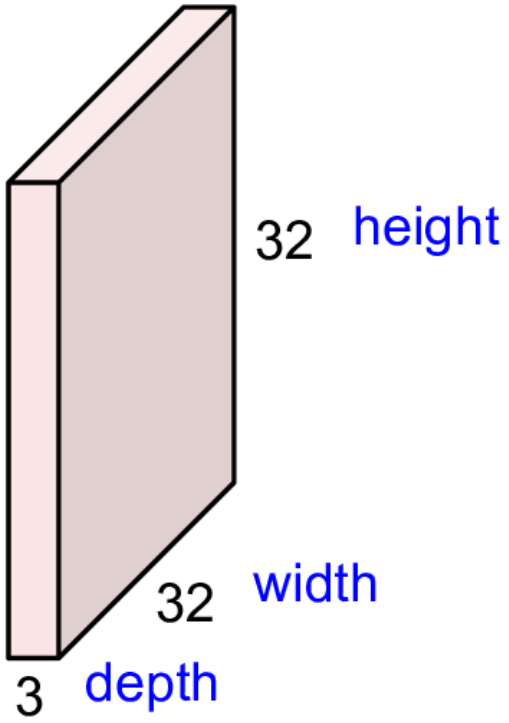
- The depth of the output volume is a hyperparameter
- It corresponds to the number of filters we would like to use, each learning to look for something different in the input.
- For example, if the first Convolutional Layer takes as input the raw image, then different neurons along the depth dimension may activate in presence of various oriented edges, or blobs of color.
- We will refer to a set of neurons that are all looking at the same region of the input as a depth column (some people also prefer the term fiber).

Depth of the Output Volume



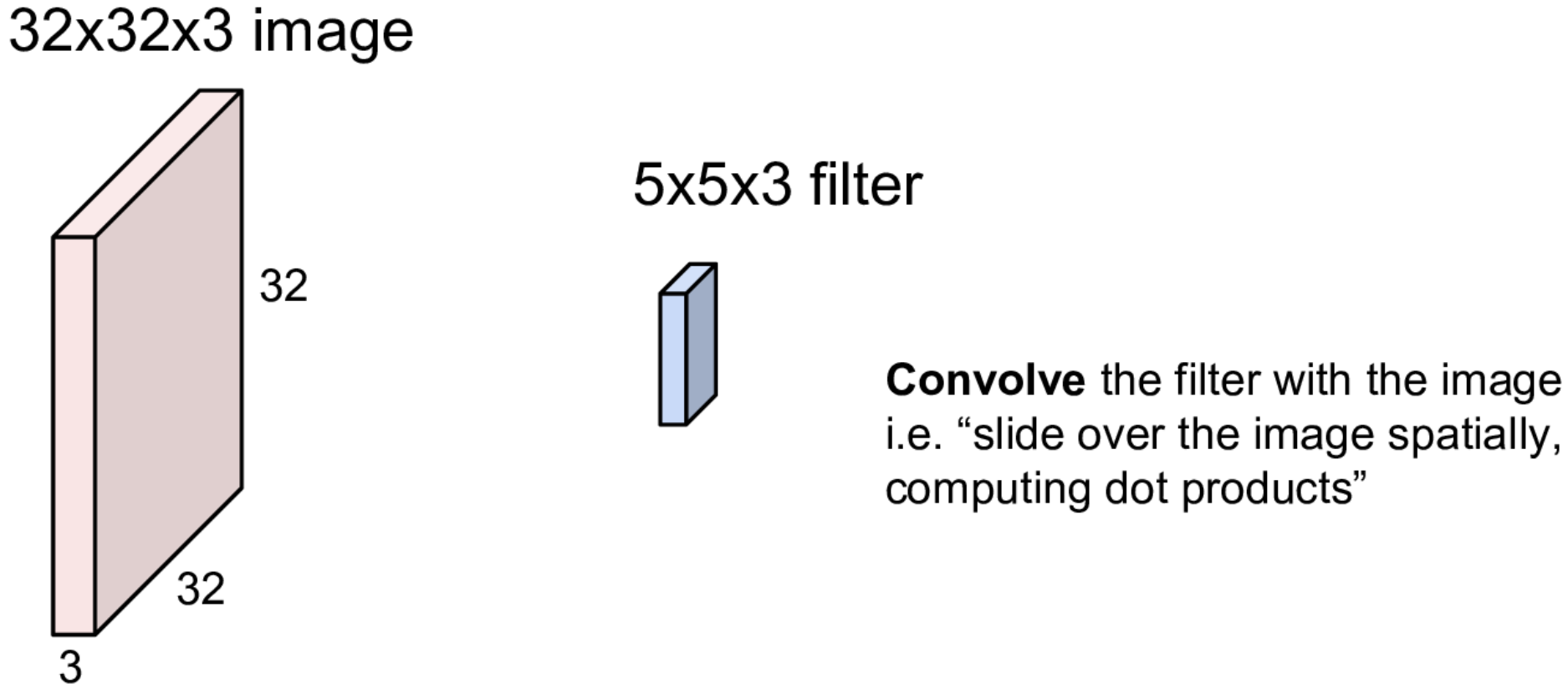
How Multiple Kernels Work

32x32x3 image -> preserve spatial structure



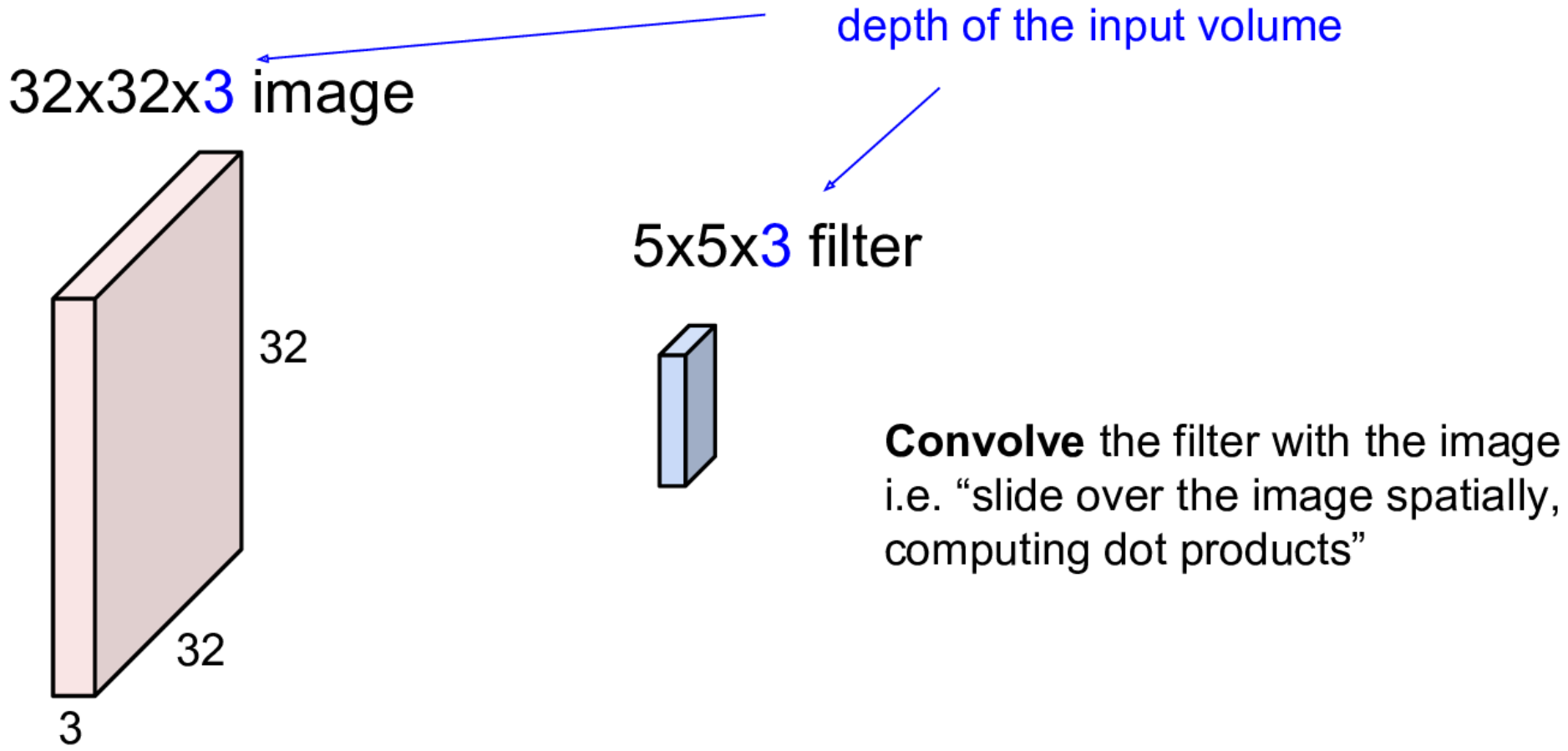
➢ http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

How Multiple Kernels Work



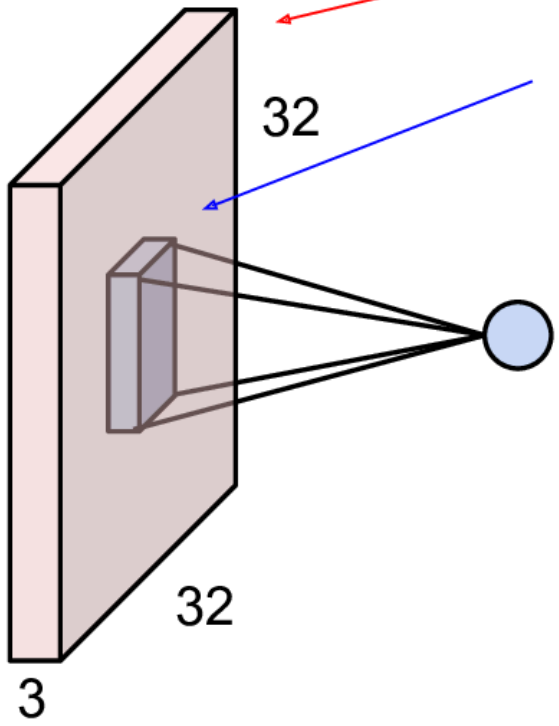
➢ http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

How Multiple Kernels Work



➢ http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

How Multiple Kernels Work

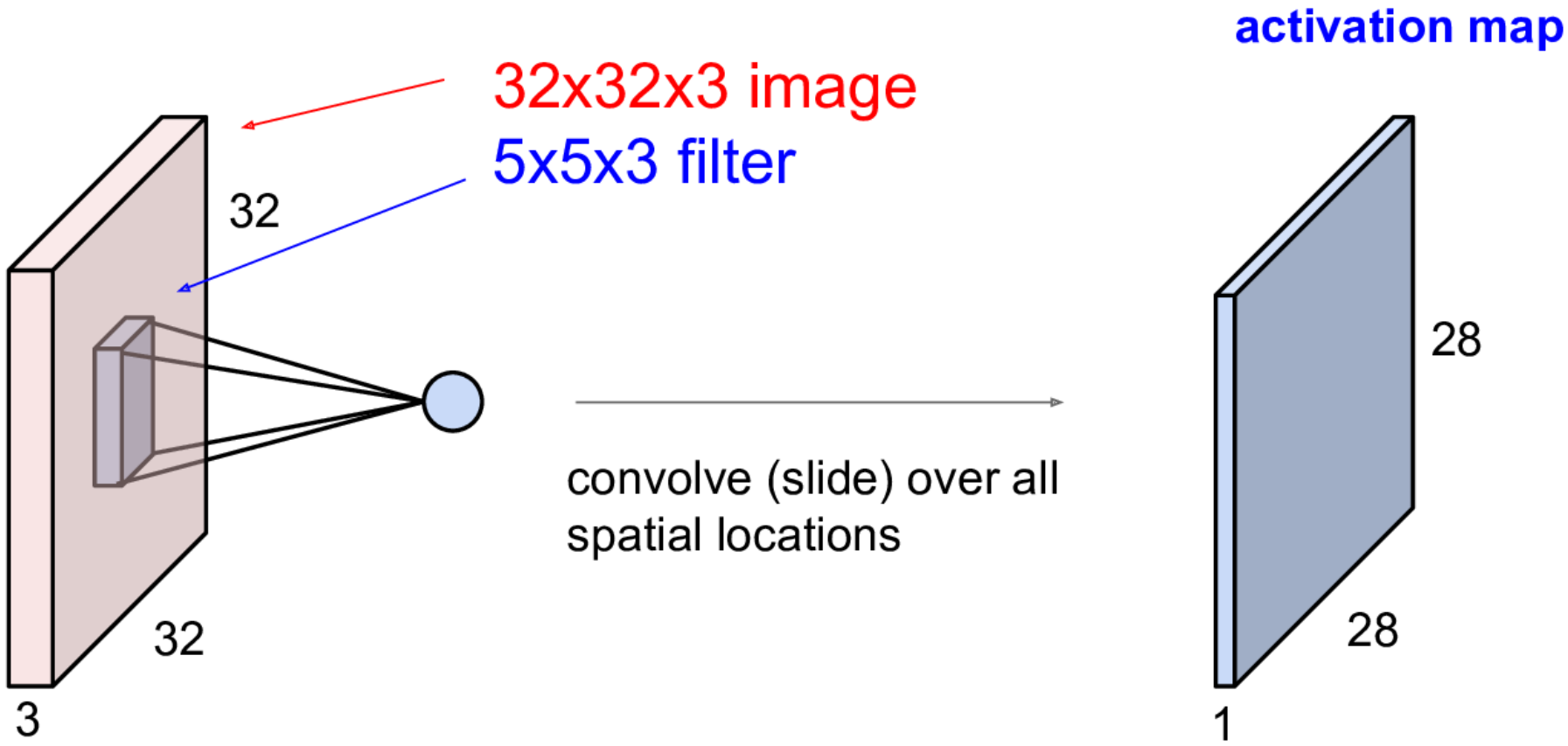


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

➢ http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

How Multiple Kernels Work

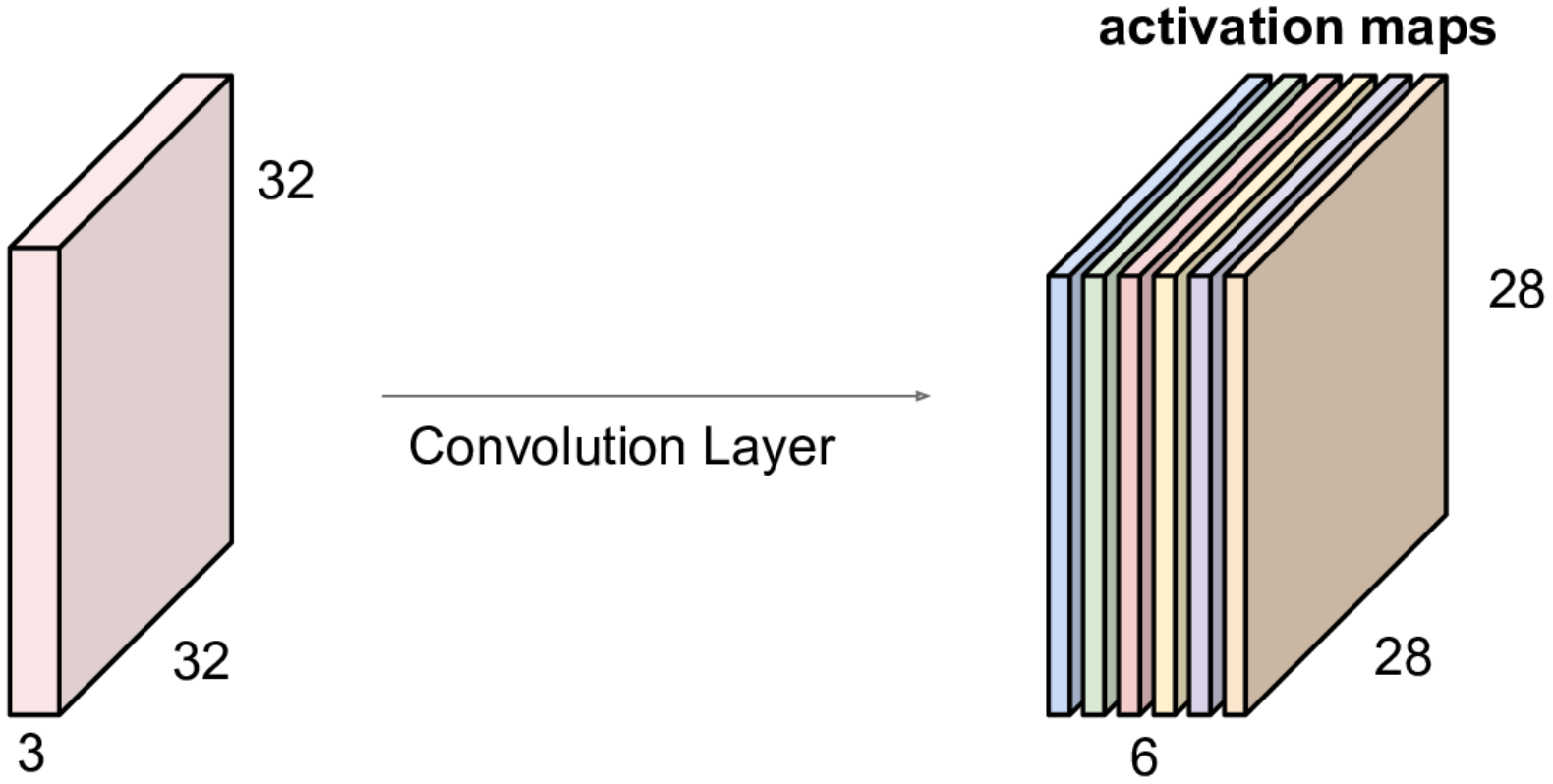


➢ http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

How Multiple Kernels Work

➢ http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

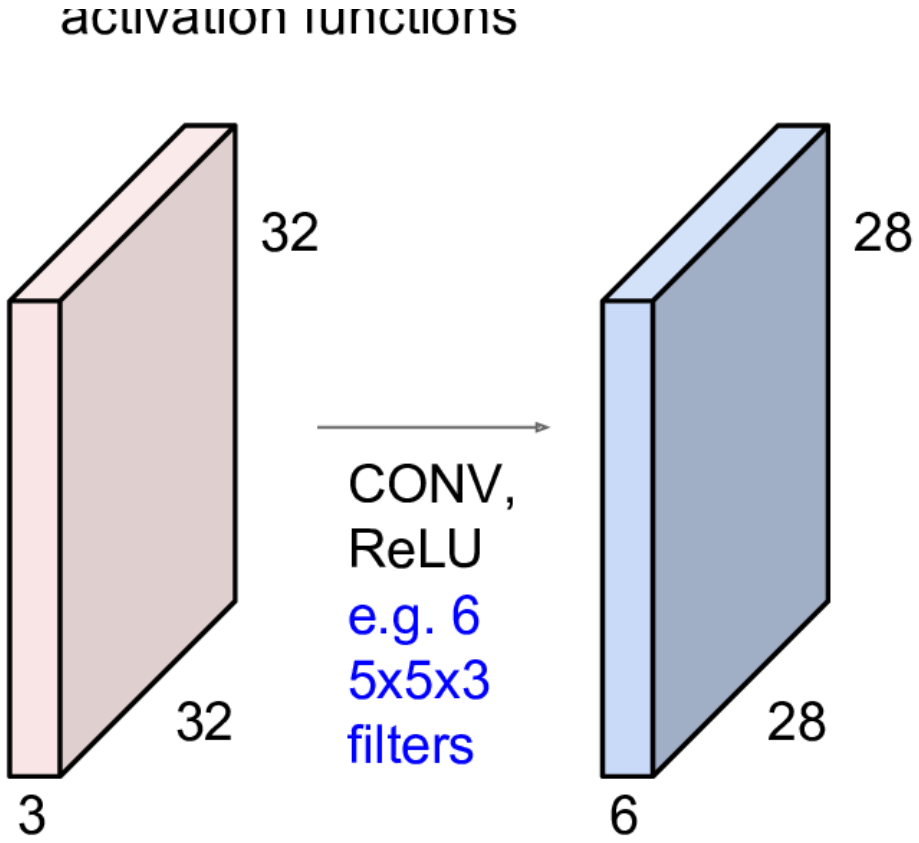
How Multiple Kernels Work



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

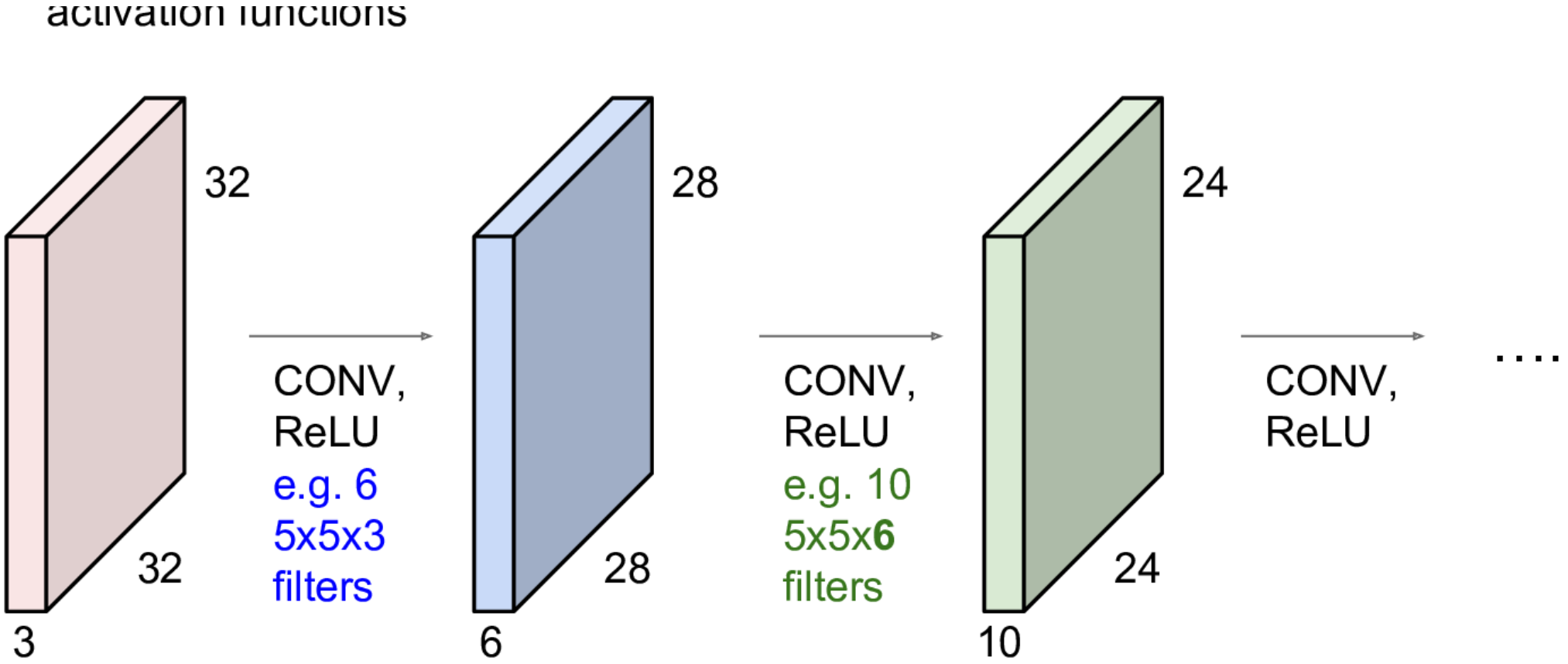
➢ http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

How Multiple Kernels Work



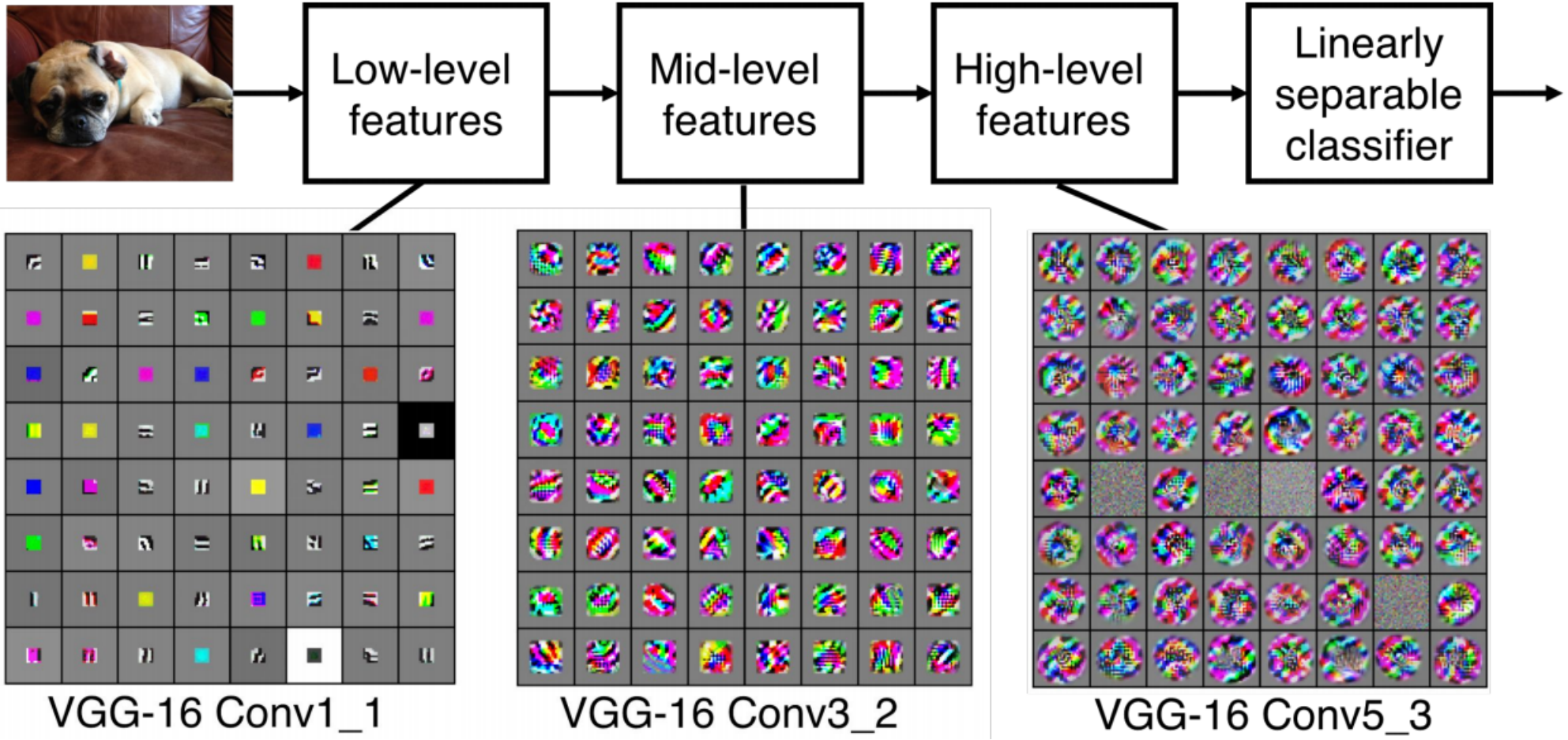
➢ http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

How Multiple Kernels Work



➢ http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

How Multiple Kernels Work



Equivariance of Convolution to Translation

- The particular form of parameter sharing leads to equivariance to translation
  - A function $f(x)$ is equivariant to a function g if $f\big(g(x)\big) = g\left(f(x)\right)$
  - Equivariant means that if the input changes, the output changes in the same way

- If g is a function that translates the input, i.e., that shifts it, then the convolution function is equivariant to g
  - $I(x, y)$ is image brightness at point (x, y)
  - $I' = g(I)$ is image function with $I'(x, y) = I(x - 1, y)$
  - If we apply g to I and then apply convolution, the output will be the same as if we applied convolution to I, then applied transformation g to the output

- Convolution enables building equivariant models
- But what about translation invariance?

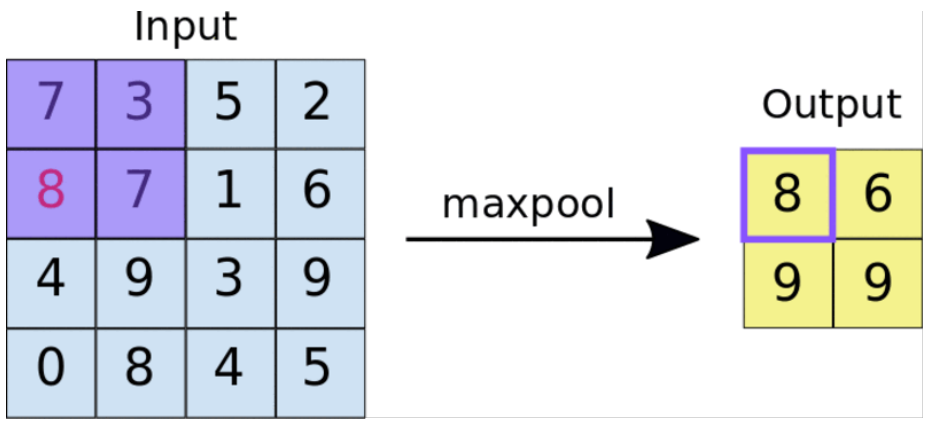
Keywords

- Convolution
- Parameter Sharing
- **Pooling**
- Normalization
- Residual connections

What else do we need?

- We have seen how convolution with multiple filters works and creates a feature-map
- Where do we put our activation functions? Do we need some in the first place?
- We also believe that high-order features are expected to be coarser than the low-level features
- How can we encode that?

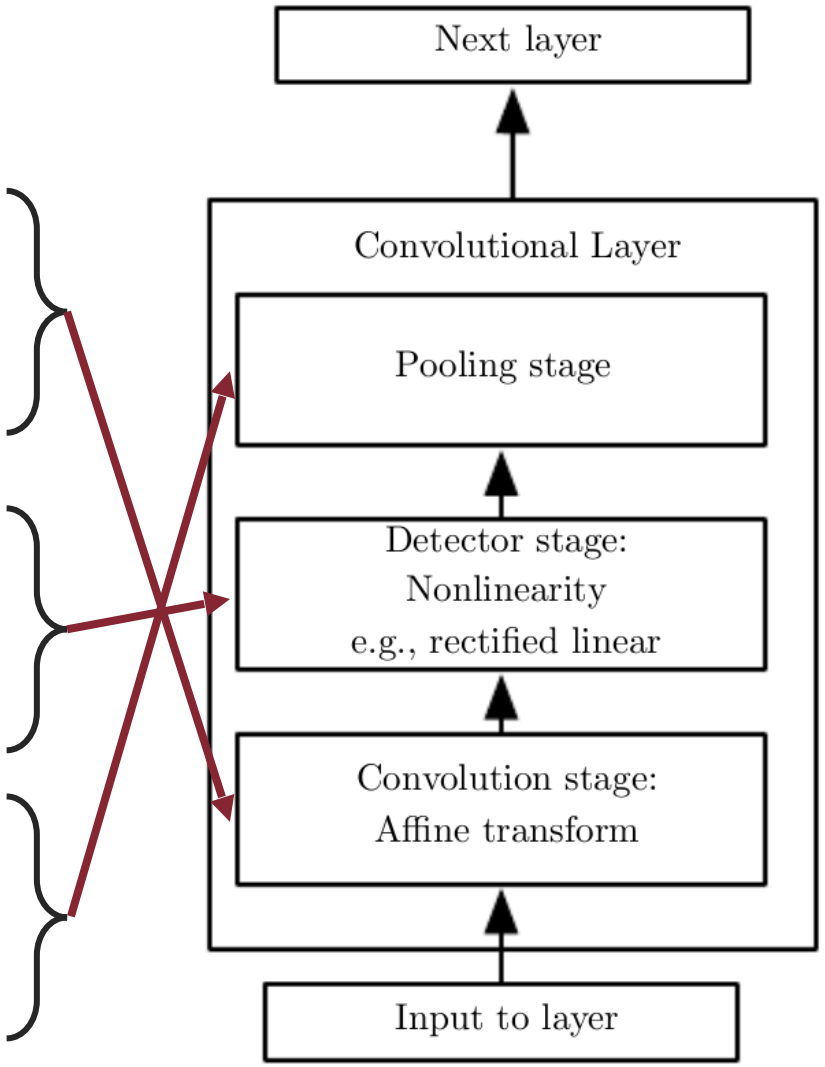
Max Pooling Example



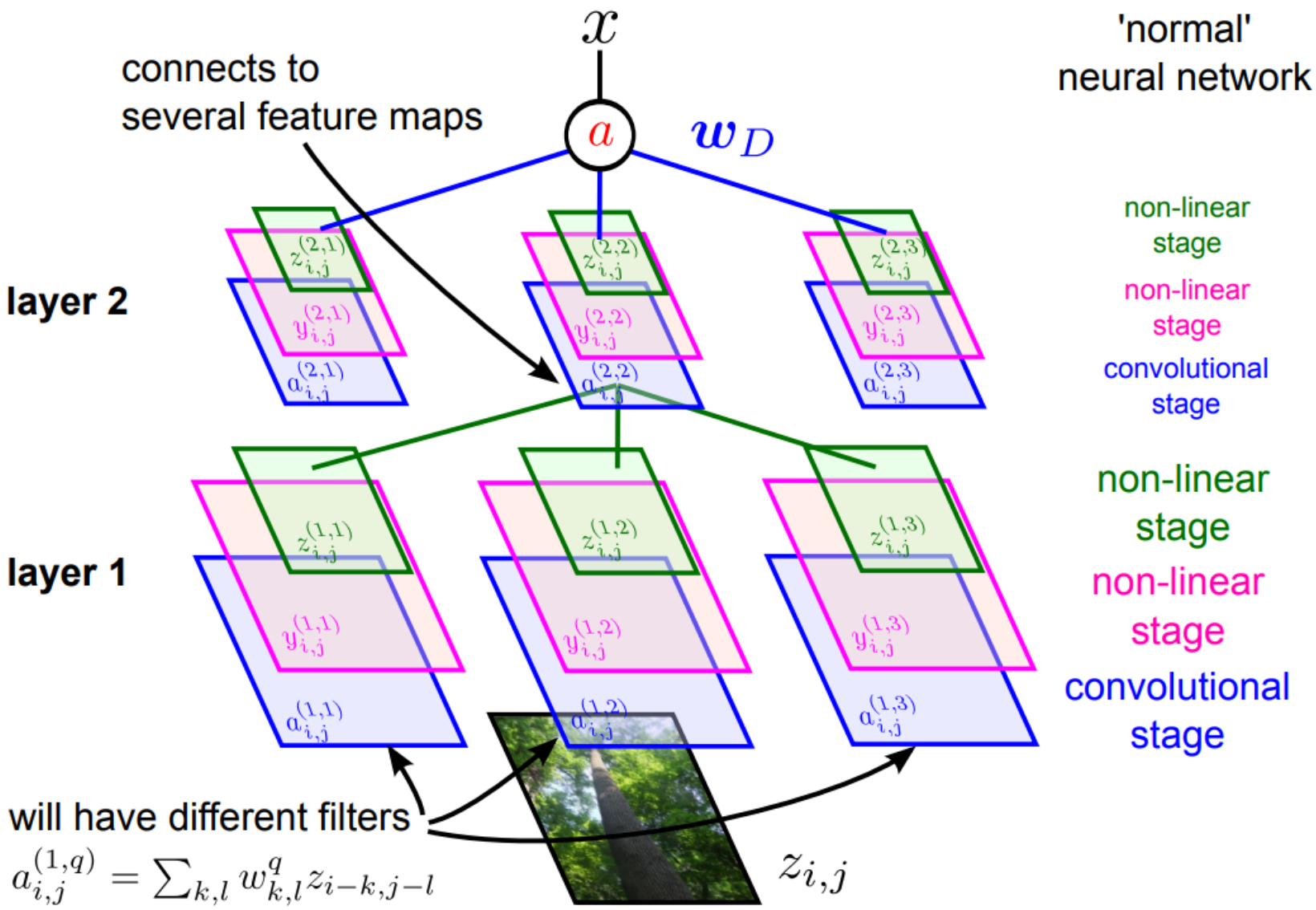
Pooling

Typical layer of a CNN consists of three stages:

- Stage 1 (**Convolution**):
  - Perform several convolutions in parallel to produce a set of linear activations
- Stage 2 (**Detector**):
  - Each linear activation is run through a nonlinear activation function (e.g. ReLU)
- Stage 3 (**Pooling**):
  - Use a pooling function to modify output of the layer further



Classic Setup for a Convolutional Network



Types of Pooling functions

➢ A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby inputs

Popular pooling functions:

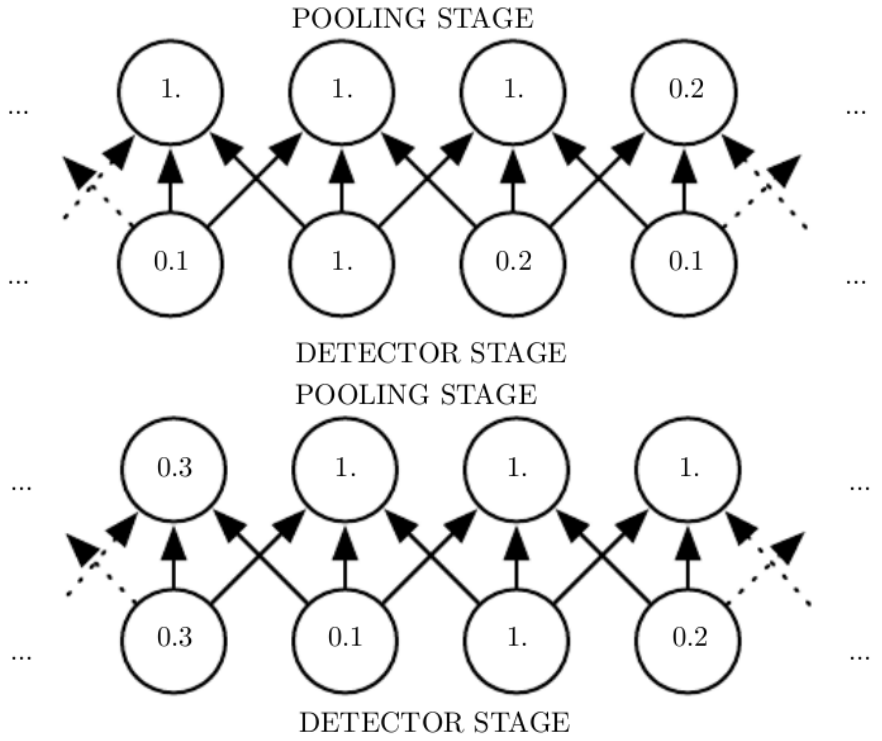
1. Max pooling operation reports the maximum output within a rectangular neighborhood
2. Average of a rectangular neighborhood
3. L^2 norm of a rectangular neighborhood
4. Weighted average based on the distance from the central pixel

Why Pooling?

- In all cases, pooling helps make the representation become approximately invariant to small translations of the input
- If we translate the input by a small amount values of most of the outputs does not change
- **Pooling can be viewed as adding a strong prior that the function the layer learns must be invariant to small translations**

Max pooling introduces invariance to translation

- View of middle of output of a convolutional layer
- Same network after the input has been shifted by one pixel
- Every input value has changed, but only half the values of output have changed



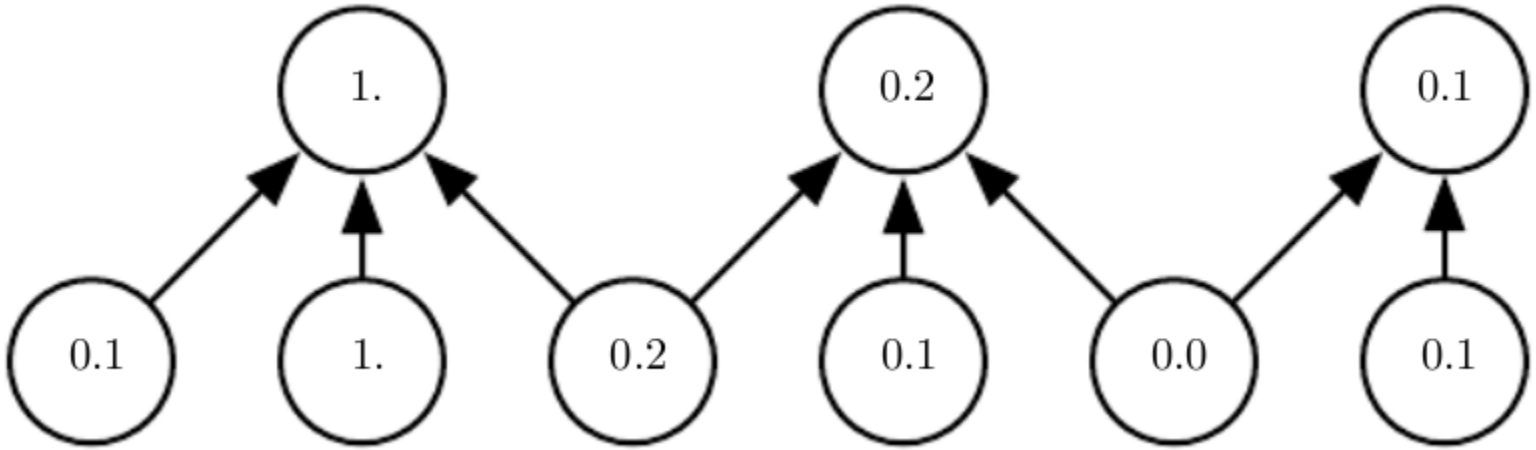
Importance of Translation Invariance

- Invariance to translation is important if we care about whether a feature is present rather than exactly where it is
  - For detecting a face, we just need to know that an eye is present in a region, not its exact location

- In other contexts, it is more important to preserve location of a feature
  - E.g., to determine a corner, we need to know whether two edges are present and test whether they meet

Pooling with Down-Sampling

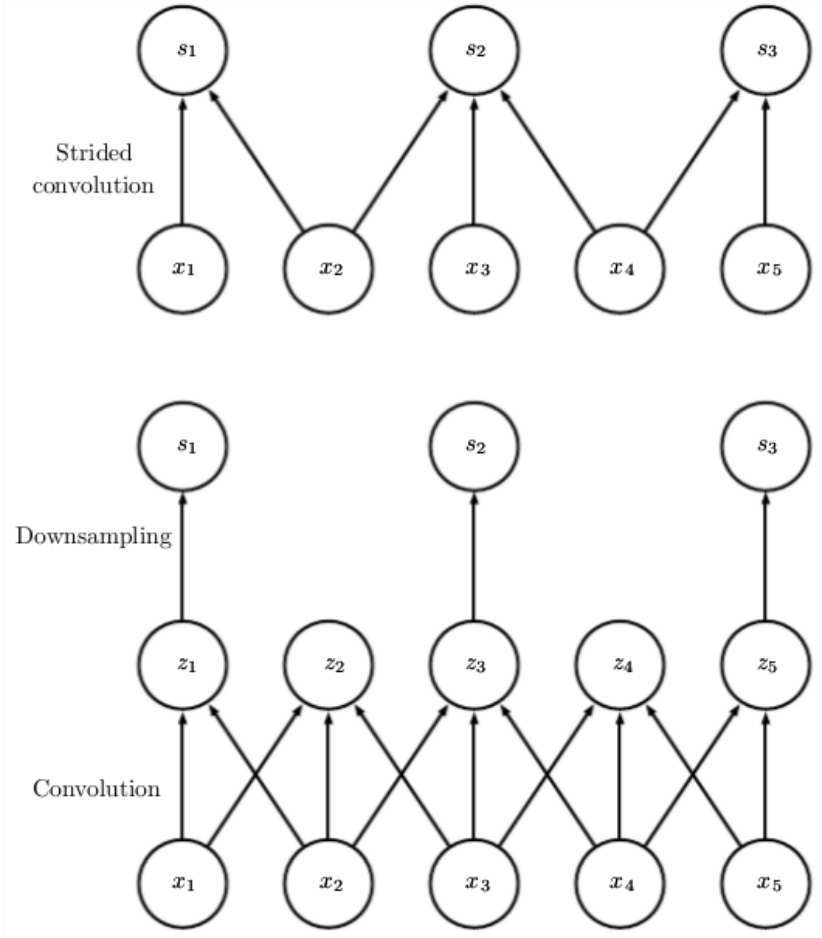
- Because pooling summarizes the responses over a whole neighborhood, it is possible to use fewer pooling units than detector units
  - By reporting summary statistics for pooling regions spaced k pixels apart rather than one pixel apart
  - Next layer has k times fewer inputs to process



Keywords

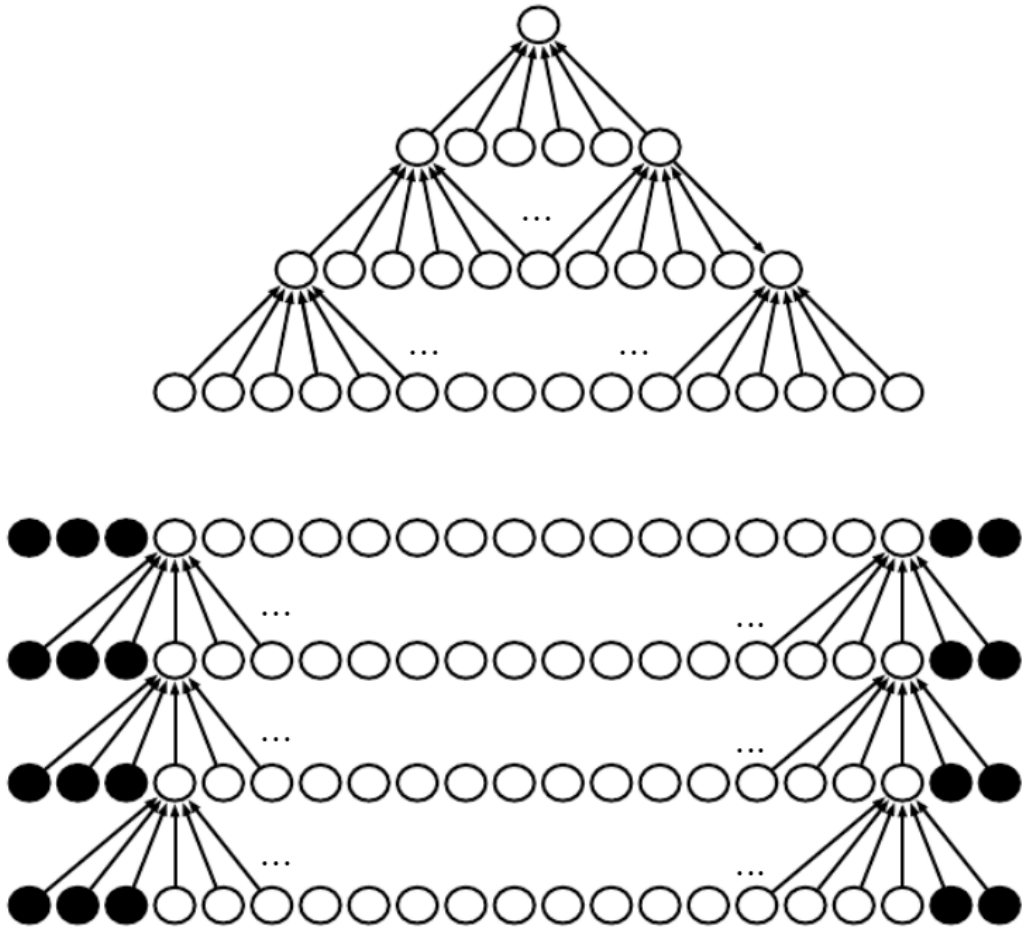
- **Convolution (variants)**
- Parameter Sharing
- Pooling
- Normalization
- Residual connections

Convolution with a stride: Implementation

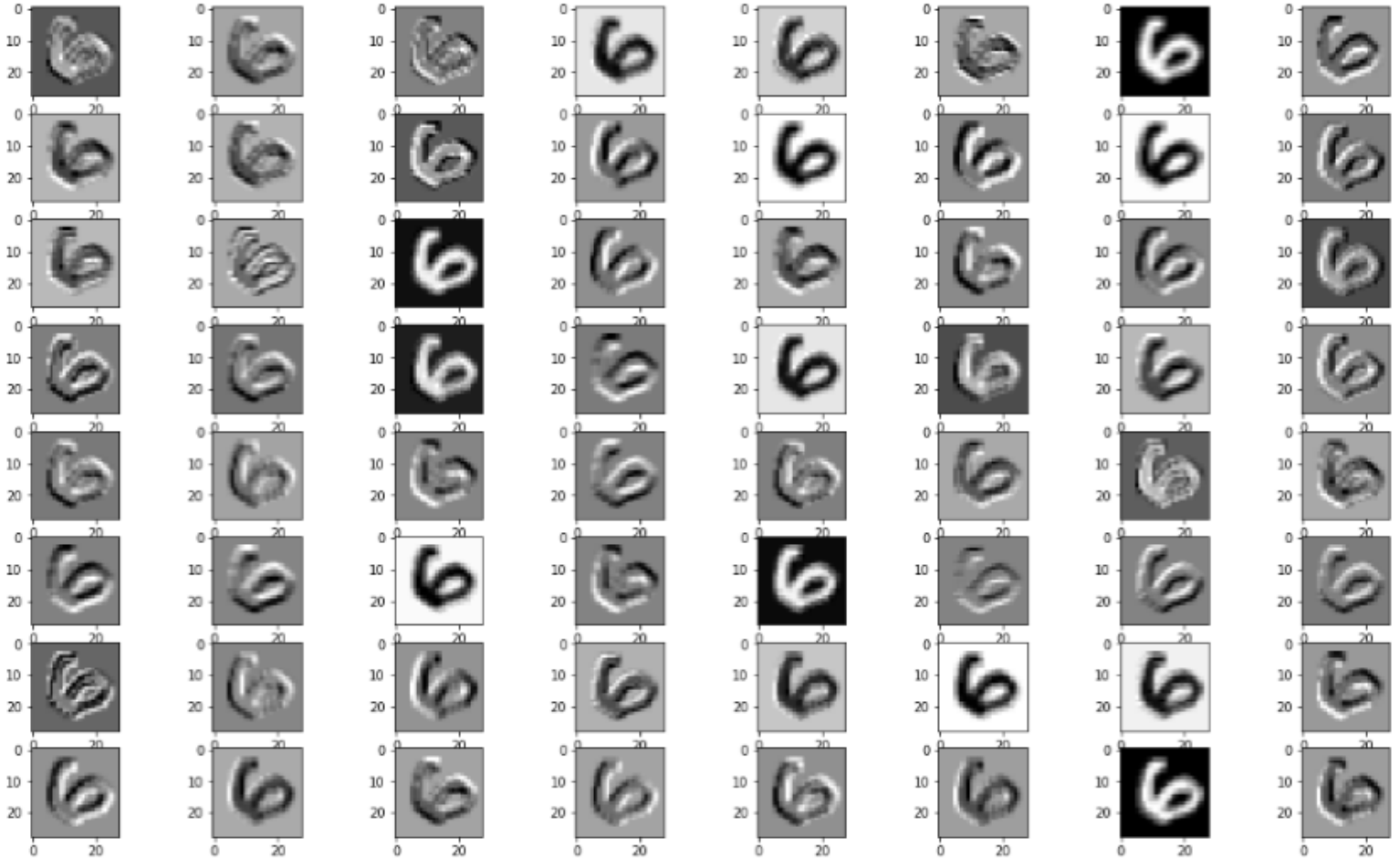


- Strided convolution is equivalent to normal convolution followed by a downsampling
- Second approach is computationally wasteful

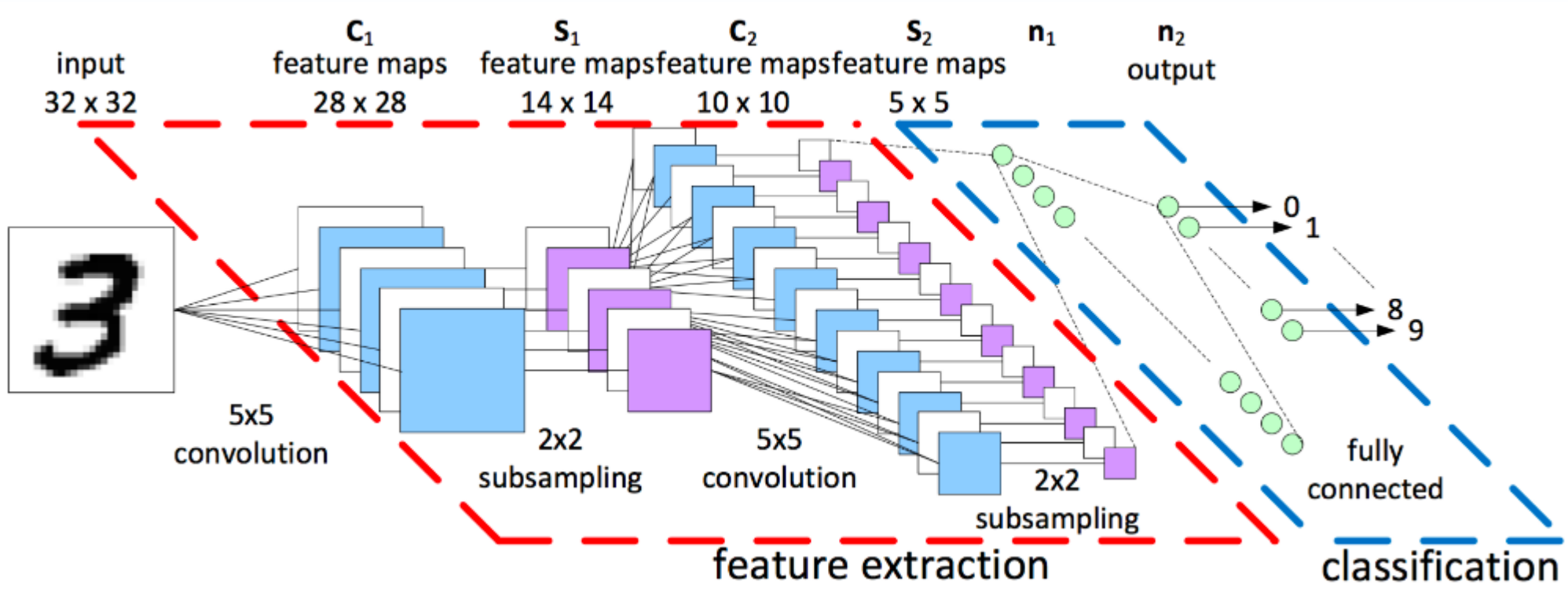
Effect of Zero-padding on network size



Visualize Activations

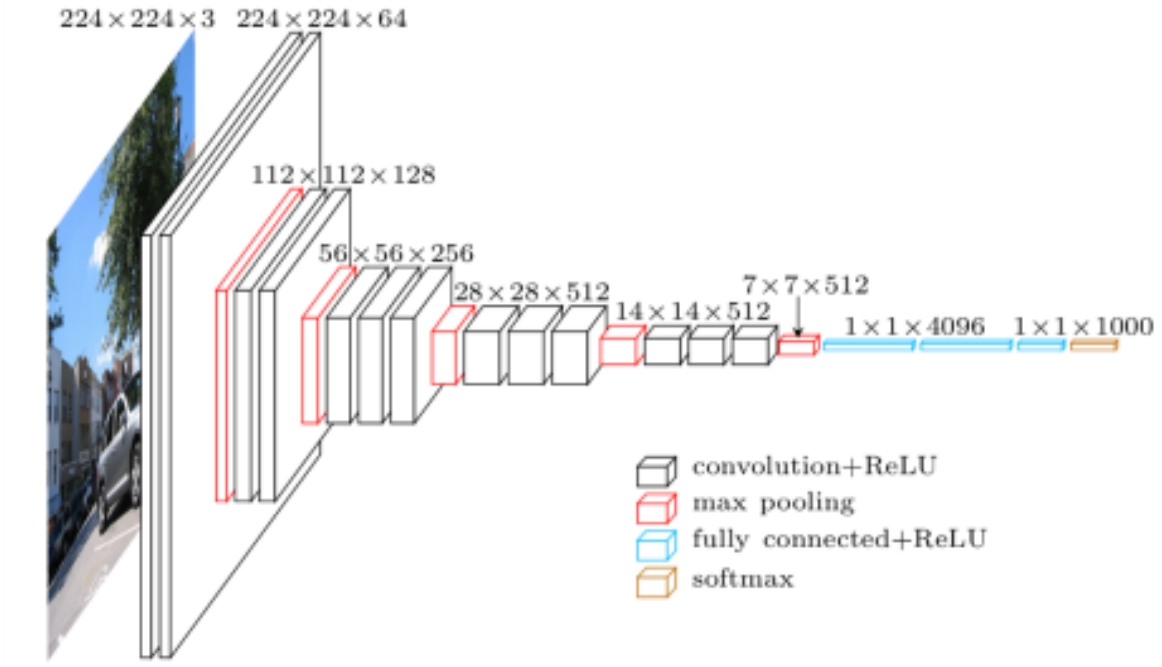


Example for a Convolutional Network



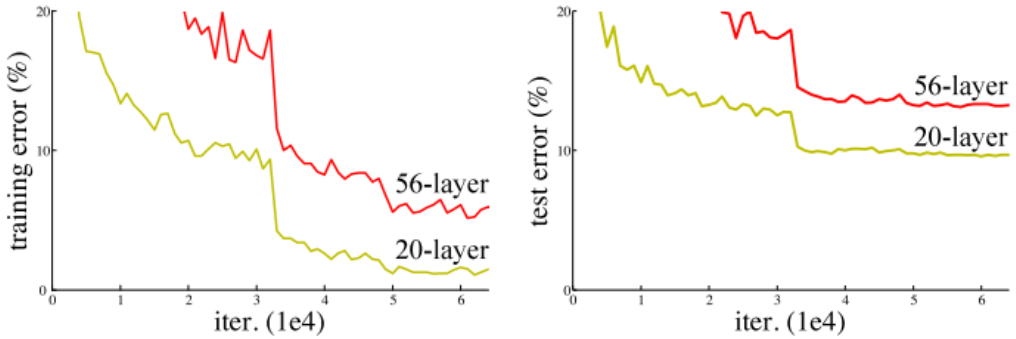
VGG Net

- VGG is a convolutional neural network model
  - "Very Deep Convolutional Networks for Large-Scale Image Recognition"
- The model achieves 93% top-5 test accuracy in ImageNet which is a dataset of over 14 million images belonging to 1000 classes.



Problems with Deep Networks

- Generally observed:
  - Deeper networks become increasingly hard to train
  - Gradients vanish or explode
  - Information is required to travel through a long distance

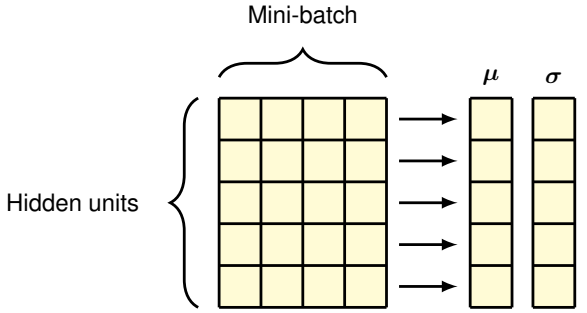


- Solution:
  - Ease training of deep network with more direct influence onto the gradient

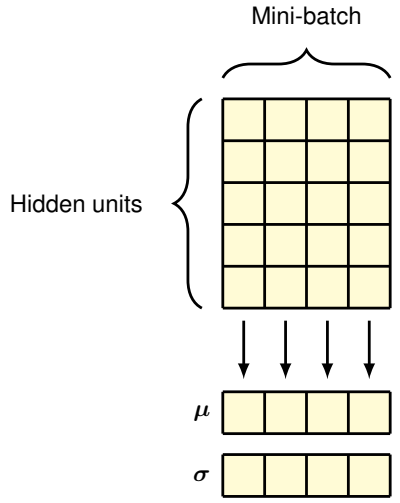
Keywords

- Convolution
- Parameter Sharing
- Pooling
- **Normalization**
- Residual connections

Normalization



- Normalizing the input data is important for NN training.
- We can also normalize the activations after each layer
- helps stabilize training of deep architectures
- **BatchNorm**: Calculate mean and SD over batch dimension
- **LayerNorm**: Calculate mean and SD over activation dimension
- Then subtract mean from activations and scale them with SD



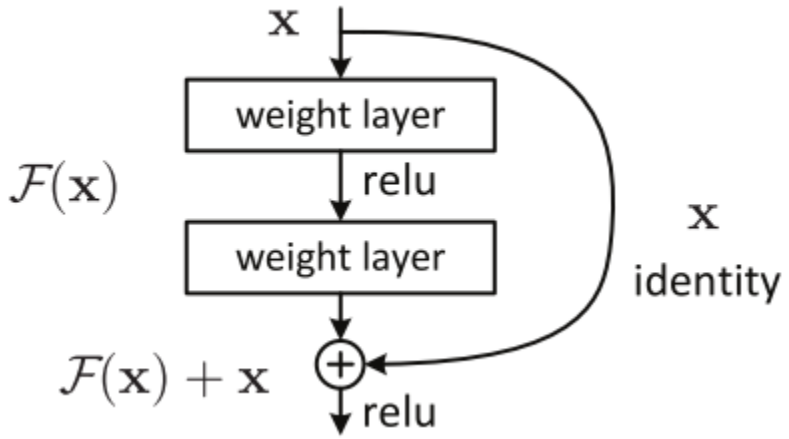
- **Caveat**: BatchNorm depends on batch size, can be noisy for small batches

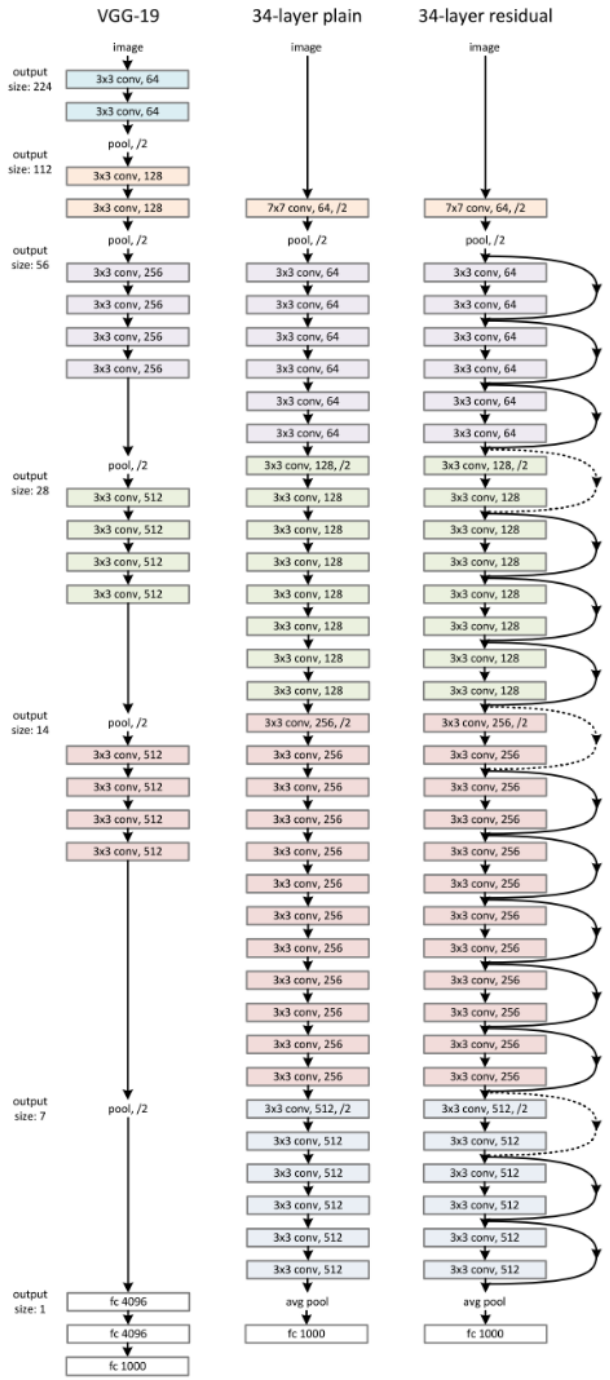
Keywords

- Convolution
- Parameter Sharing
- Pooling
- Normalization
- **Residual connections**

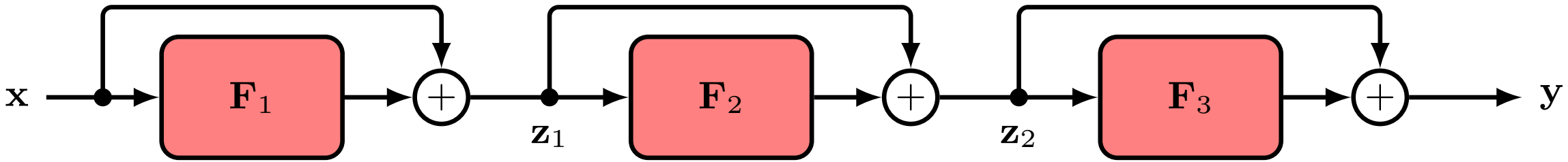
Idea of ResNet

- The network is constructed in blocks
- Introduces the identity shortcut connections

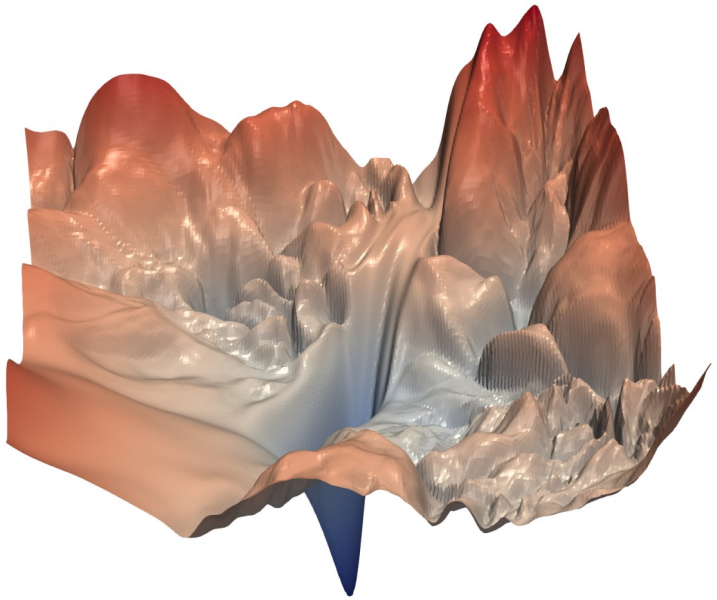




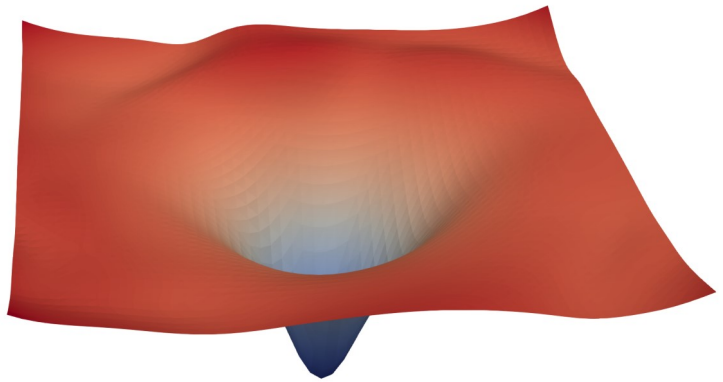
Residual Connections as a Regularizer



Loss landscape without residual connections



(a)

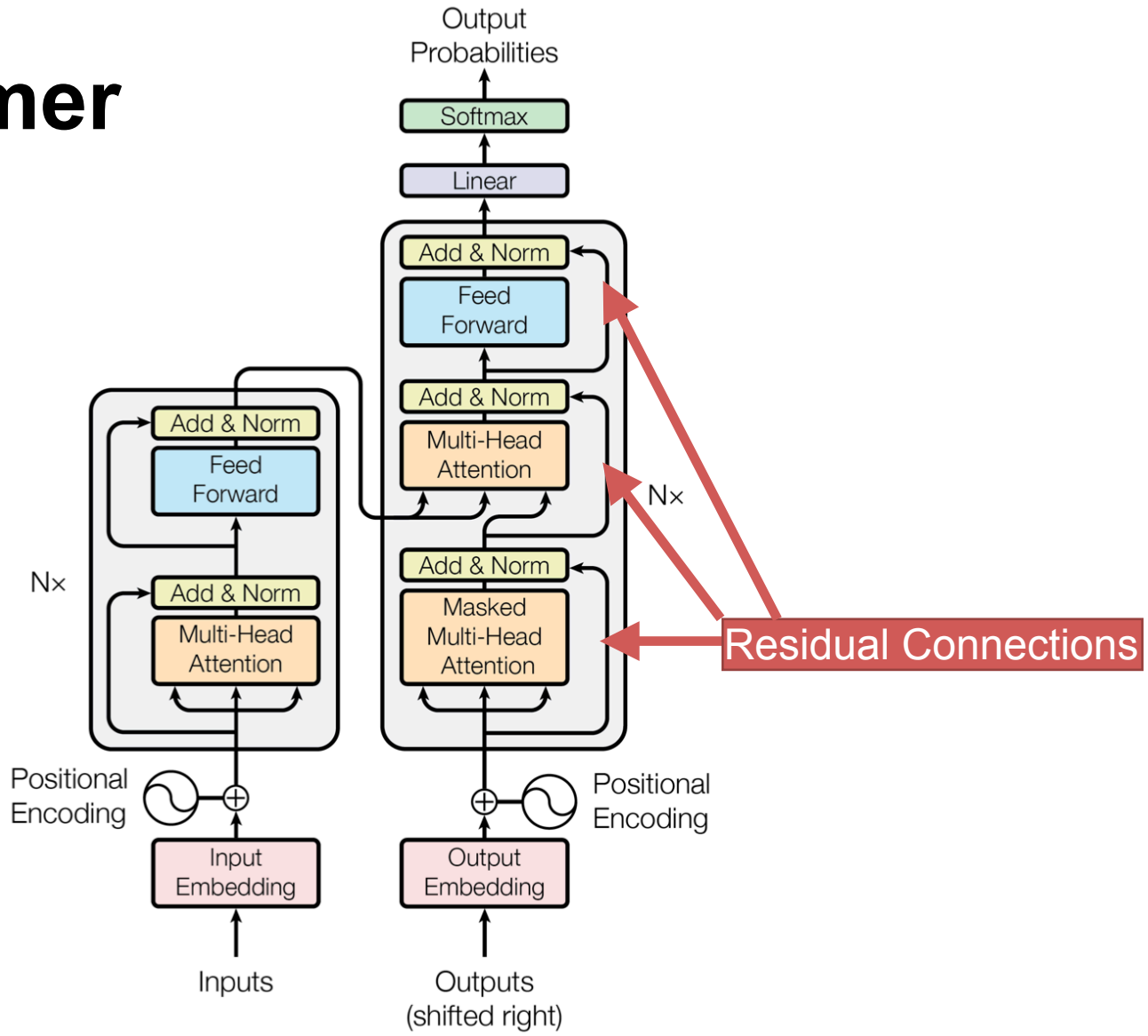


(b)

Loss landscape **with** residual connections

Preview: The Transformer

Vaswani et al. (2017)



Keywords

- Convolution
- Parameter Sharing
- Pooling
- Normalization
- Residual connections

How’s my teaching?

https://www.admonymous.co/lukasgalke