MIDDLE EAST TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

CENG 403 Introduction to Deep Learning

Week 9b

Sinan Kalkan

Previously on CENG403

Disadvantages of MLPs: Dimensionality

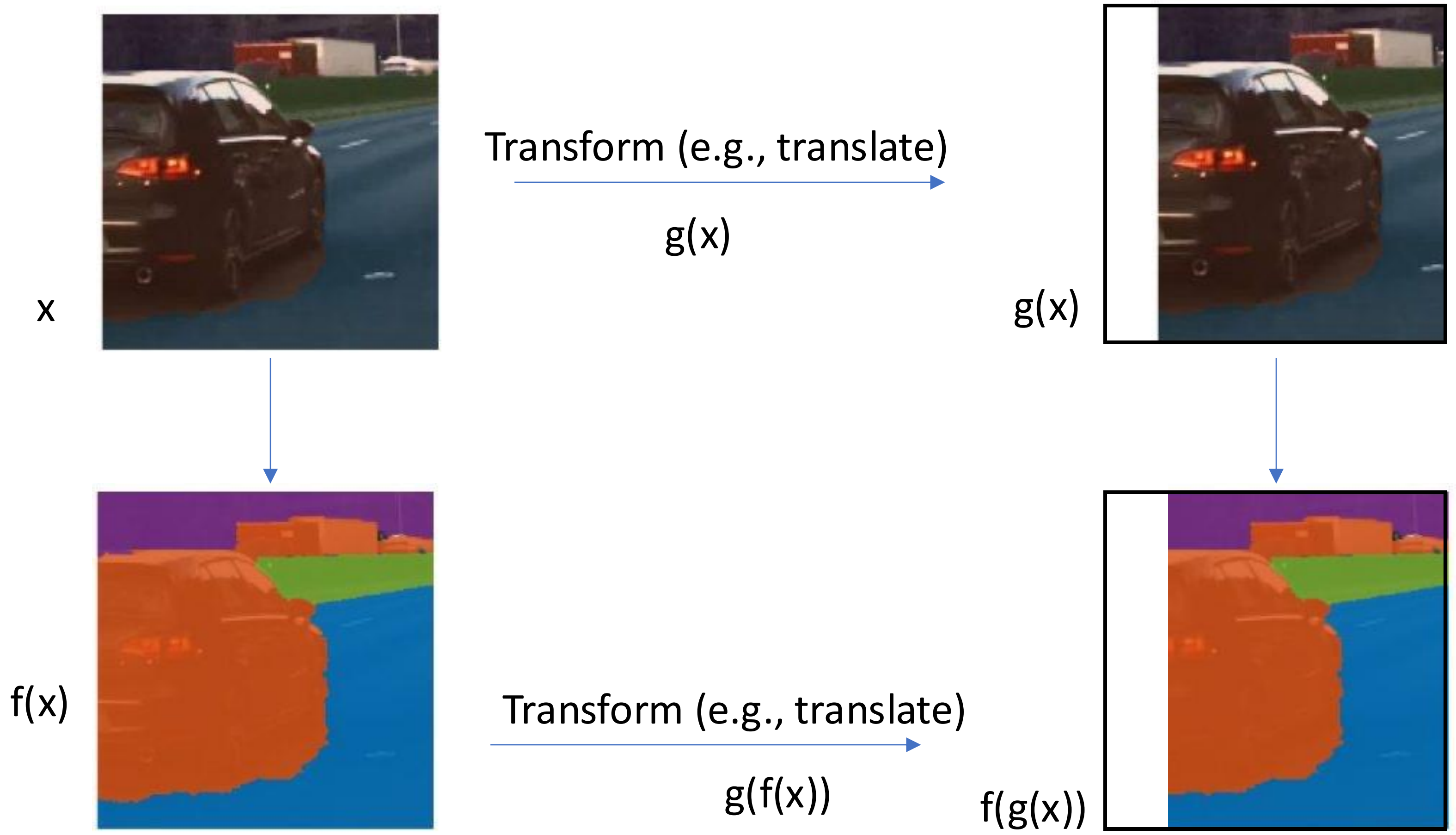
- The number of parameters in an MLP is high for practical problems
  - e.g., for grayscale images with 1000x1000 resolution, a fully-connected layer with 1000 neurons requires 10^9 parameters.
- The number of parameters in an MLP increases quadratically with an increase in input dimensionality
- For example, for a fully-connected layer with n_{in} input neurons and n_{out} output neurons:
  - Number of parameters: $n_{in} \times n_{out}$
  - Assuming proportional decrease in layer size, e.g. $n_{out} = n_{in}/10$, gives: $n_{in} \times n_{out} = n_{in}^2/10$
  - Increasing n_{in} by d yields a change of $\mathcal{O}(d^2)$.
- This is a problem because:
  - More parameters => larger model size & more computational complexity.
  - More parameters => more data for good generalization.
- Teaser for CNNs:
  - Input size does not affect model size (in general)

Input Image & Its Segmentation:
https://www.mathworks.com/discovery/image-segmentation.html

Previously on CENG403

Equivariance vs. Invariance

- Equivariant problem: image segmentation: f(g(x)) = g(f(x))

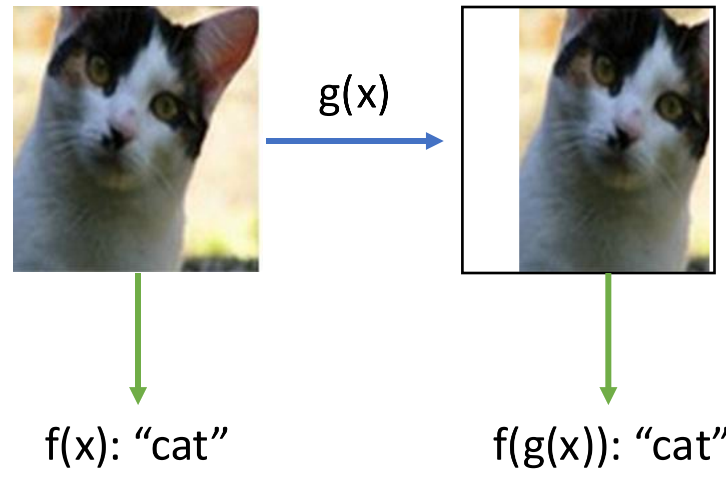


Equivariance vs. Invariance

- Invariant problem: object recognition.
  - f(g(x)) = f(x)

- Teaser for CNNs:
  - Pooling provides invariance, convolution provides equivariance. To a certain degree.

An invariant problem – object recognition:

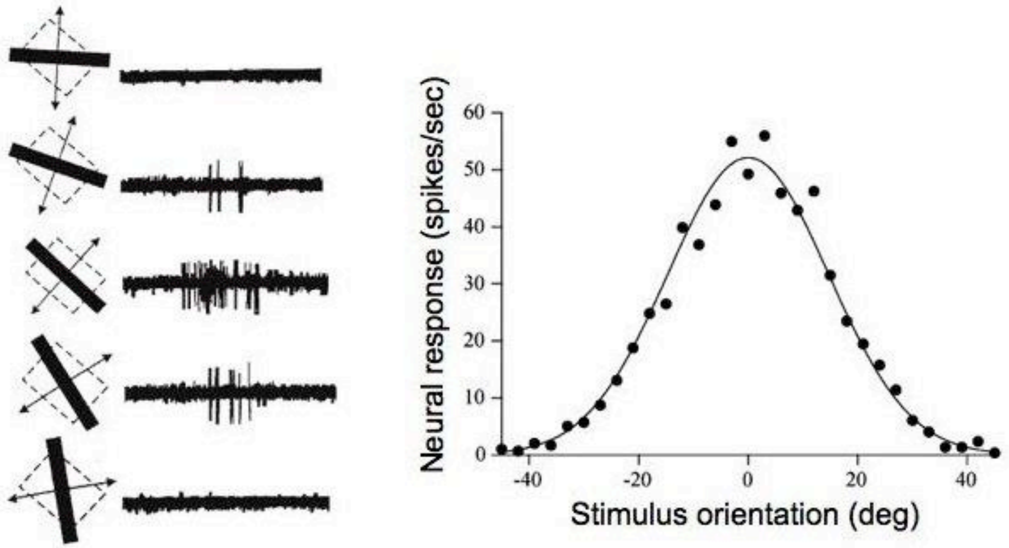


Previously on CENG403

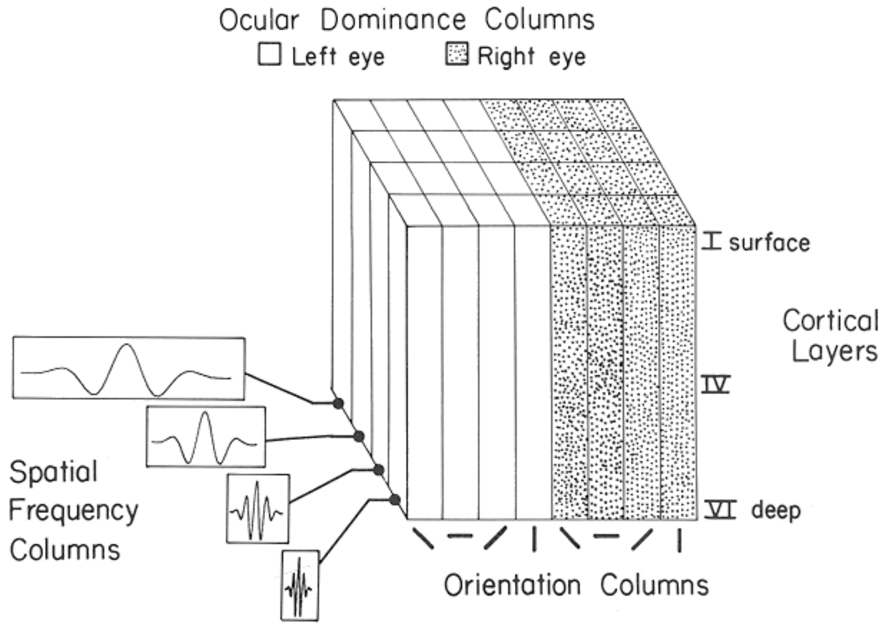
An Alternative to MLPs

Solution (inspiration):

- Hubel & Wiesel (1960s): Brain neurons are not fully connected. They have local receptive fields



Hubel & Wiesel, 1968



Model of Striate Module in Cats

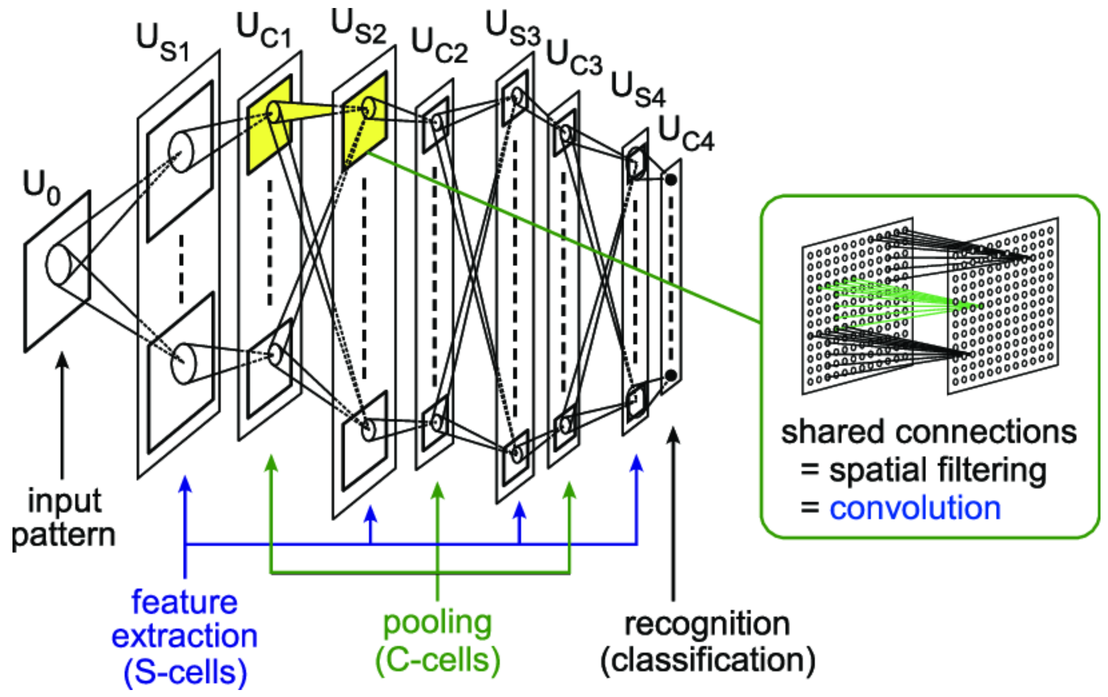
http://fourier.eng.hmc.edu/e180/lectures/retina/node1.html

An Alternative to MLPs

Solution: Neocognitron (Fukushima, 1979):

A neural network model unaffected by shift in position, applied to Japanese handwritten character recognition.

- S (simple) cells: local feature extraction.
- C (complex) cells: provide tolerance to deformation, e.g. shift.
- Self-organization-based learning method.



U_{S1}
U_{C1}
U_{S2}
U_{C2}
U_{S3}
U_{C3}
U_{S4}
U_{C4}
U_0
input pattern
feature extraction (S-cells)
pooling (C-cells)
recognition (classification)
shared connections = spatial filtering = convolution

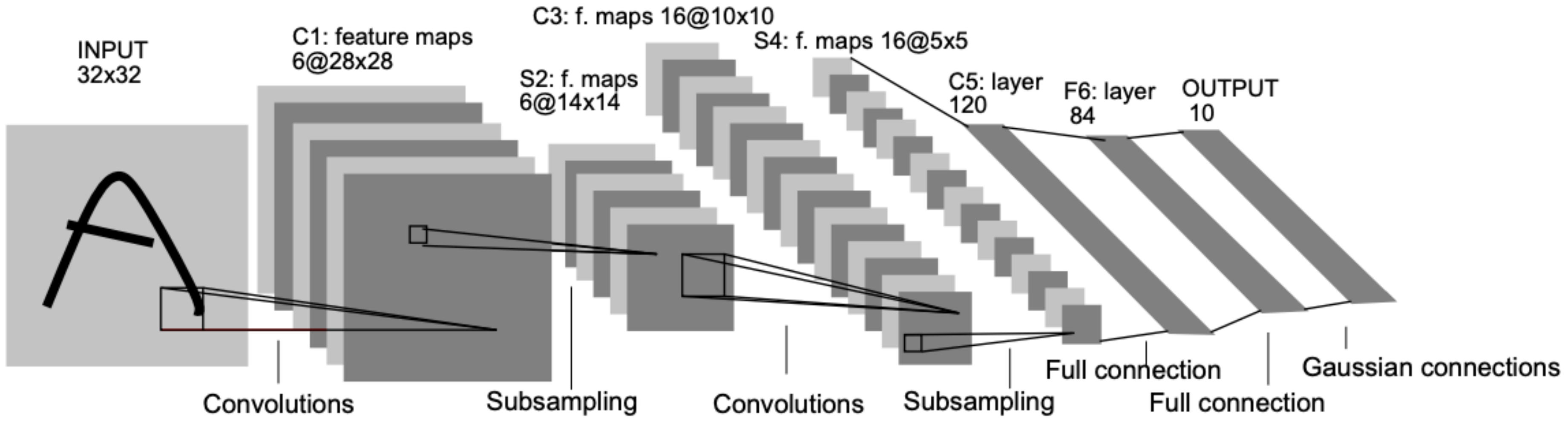
Figure: Fukushima (2019), Recent advances in the deep CNN neocognitron.

Previously on CENG403

An Alternative to MLPs

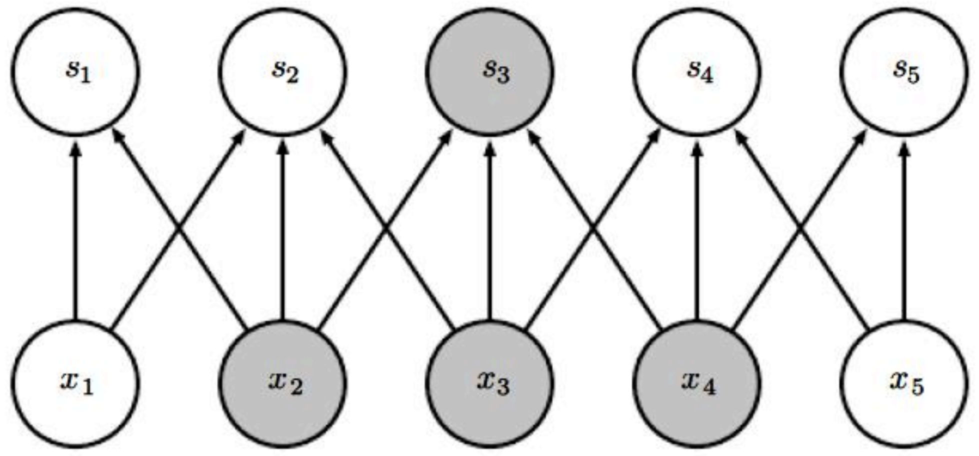
Solution: Convolutional Neural Networks (Lecun, 1998)

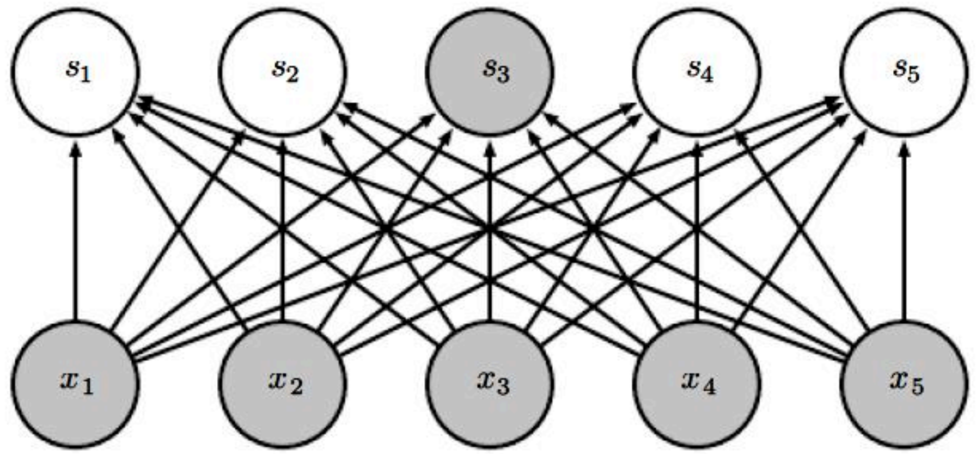
- Gradient descent
- Weights shared
- Document recognition



Lecun, 1998

CNNs vs. MLPs: Dimensionality





When things go deep, an output may depend on all or most of the input:

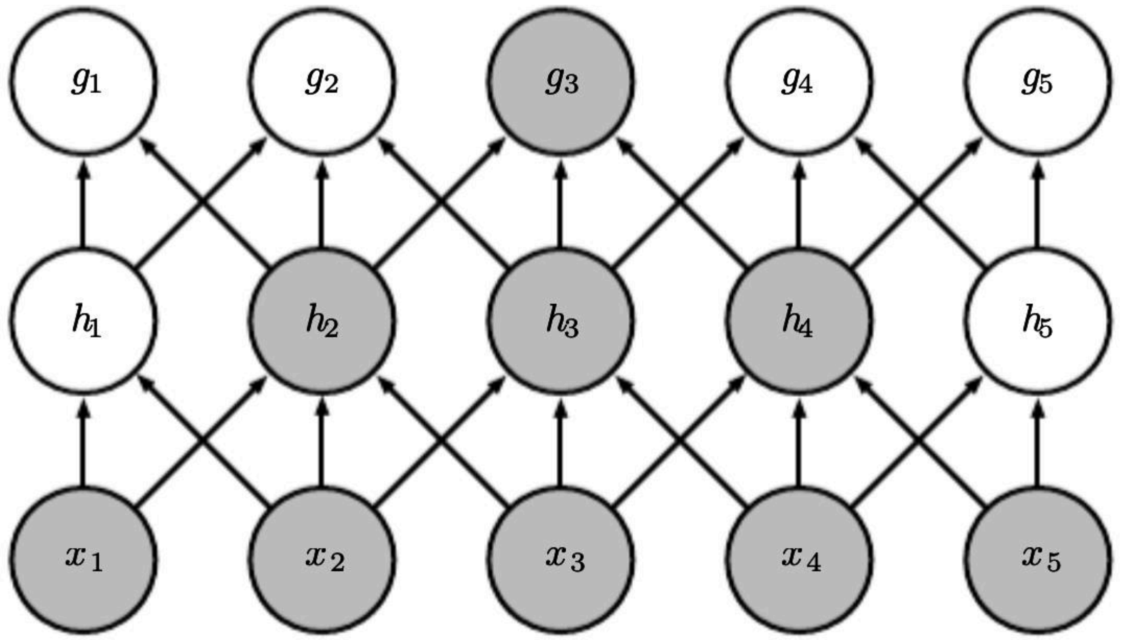
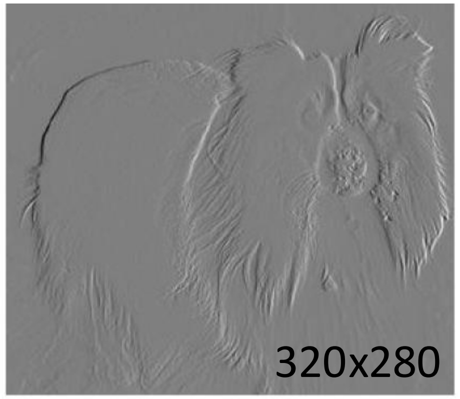


Figure: Goodfellow et al., “Deep Learning”, MIT Press, 2016.

Previously on CENG403

CNNs vs. MLPs: Dimensionality

- Parameter sharing
  - In regular ANN, each weight is independent
- In CNN, a layer might re-apply the same convolution and therefore, share the parameters of a convolution
  - Reduces storage and learning time



- For a neuron in the next layer:
  - With ANN: 320x280x320x280 multiplications
  - With CNN: 320x280x3x3 multiplications

Figure: Goodfellow et al., "Deep Learning", MIT Press, 2016.

Previously on CENG403

CNNs vs. MLPs: Equivariance & Invariance

- Equivariant to translation
  - The output will be the same, just translated, since the weights are shared.

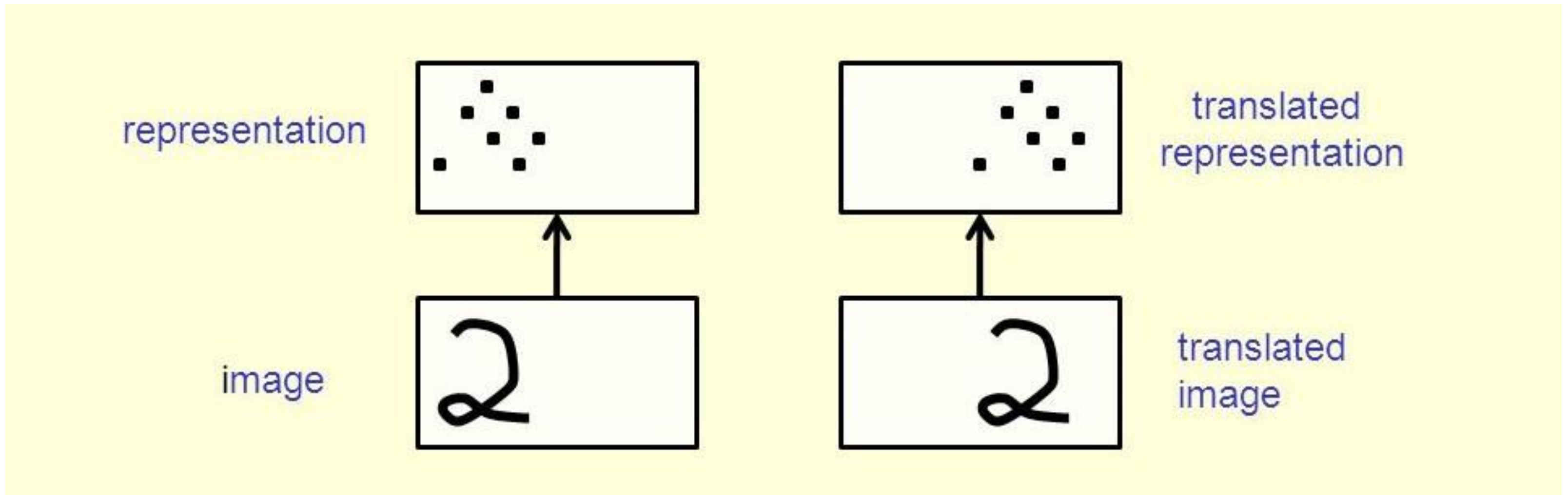
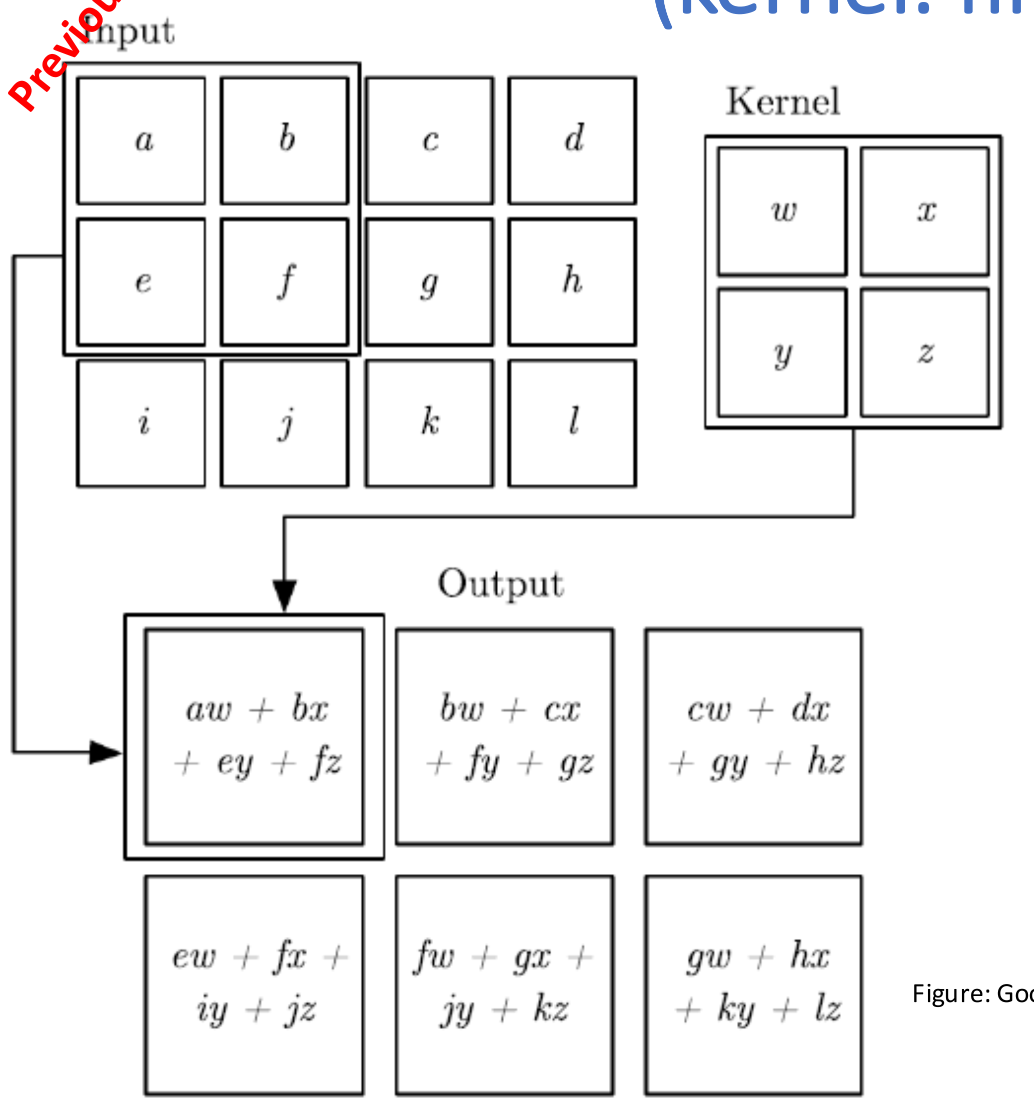


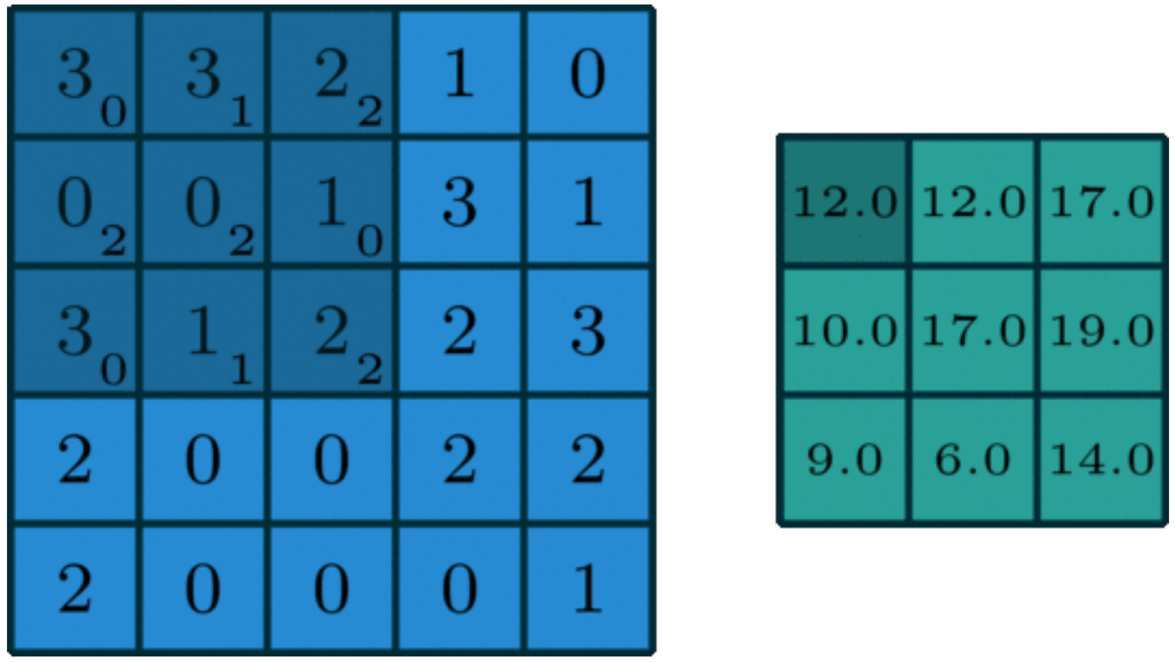
Figure: https://towardsdatascience.com/translational-invariance-vs-translational-equivariance-f9fbc8fca63a

- Not equivariant to scale or rotation.

Previously on CENG403

Example multi-dimensional convolution
(kernel: finite impulse response)





https://github.com/vdumoulin/conv_arithmetic

Figure: Goodfellow et al., “Deep Learning”, MIT Press, 2016.

Previously on CENG403

CNN layers

- Operations in a CNN:
  - Convolution (in parallel) to produce pre-synaptic activations
  - Detector: Non-linear function
  - Pooling: A summary of a neighborhood

- Pooling of a region in a feature/activation map:
  - Max
  - Average
  - L2 norm
  - Weighted average acc. to the distance to the center
  - …

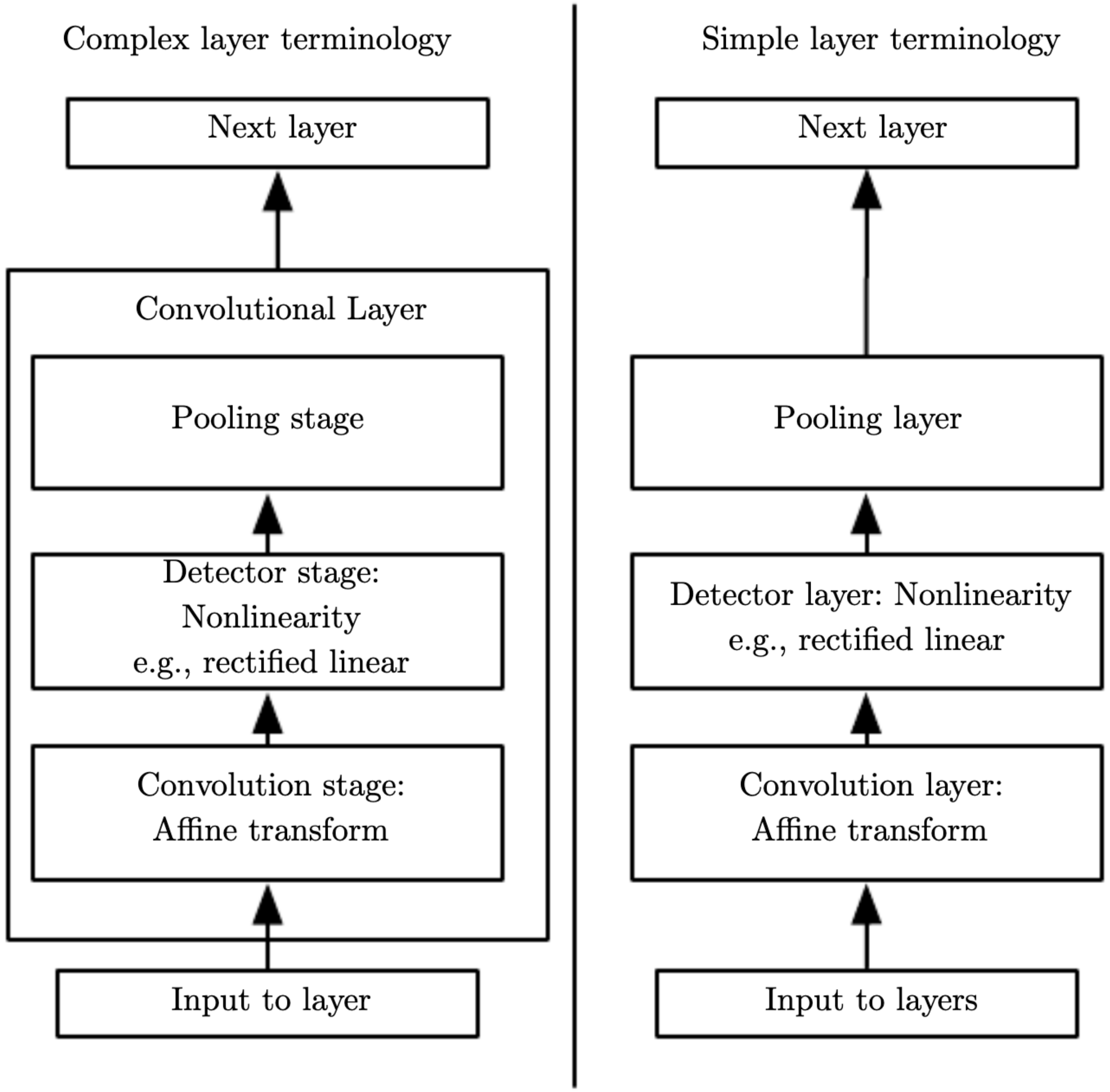
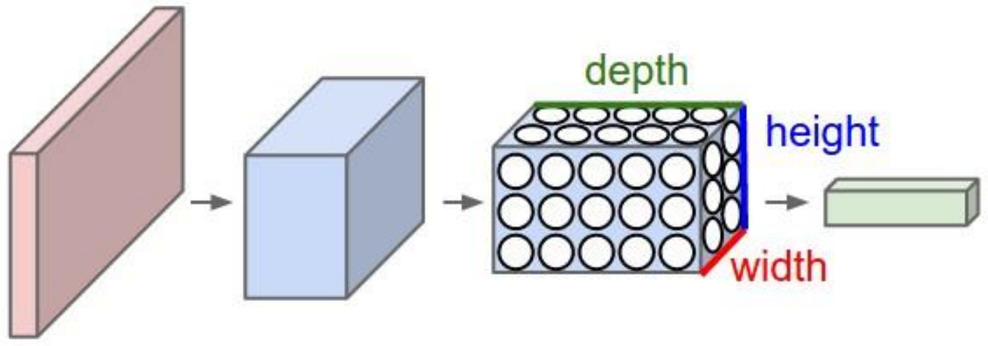

Complex layer terminology
Next layer
Convolutional Layer
Pooling stage
Detector stage: Nonlinearity e.g., rectified linear
Convolution stage: Affine transform
Input to layer
Simple layer terminology
Next layer
Pooling layer
Detector layer: Nonlinearity e.g., rectified linear
Convolution layer: Affine transform
Input to layers

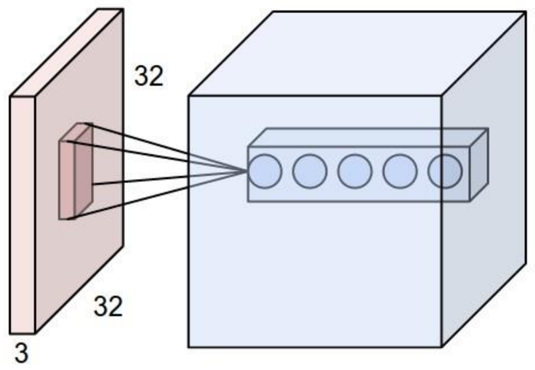
Figure: Goodfellow et al., "Deep Learning", MIT Press, 2016.

Previously on CENG403

Connectivity in CNN

- Local: The behavior of a neuron does not change other than being restricted to a subspace of the input.
- Each neuron is connected to a slice of the previous layer
- A layer is actually a volume having a certain width x height and depth (or channel)
- A neuron is connected to a subspace of width x height but to all channels (depth)
- Example: CIFAR-10
  - Input: 32 x 32 x 3 (3 for RGB channels)
  - A neuron in the next layer with receptive field size 5x5 has input from a volume of 5x5x3.





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

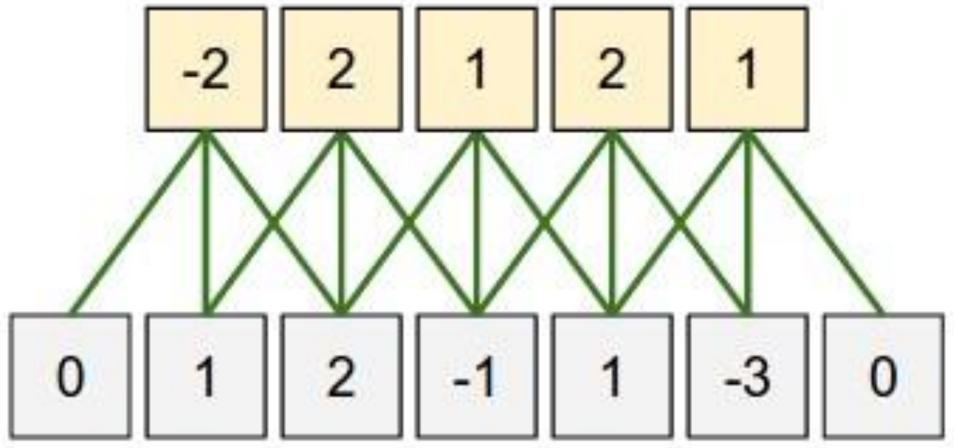
Today

- Convolutional Neural Networks (CNNs)
  - Continue with convolution
  - Types of convolution in CNNs

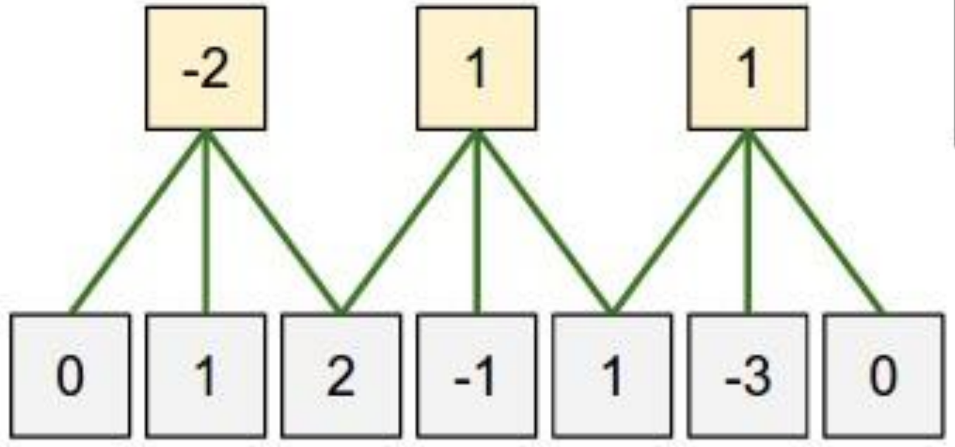
Important parameters

Stride

- The amount of space between neighboring receptive fields
- If it is small, RFs overlap more
- It it is big, RFs overlap less

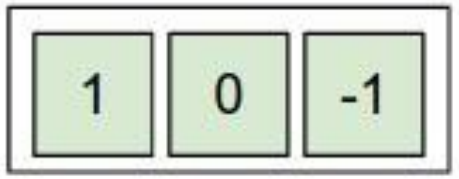


Stride = 1



Stride = 2

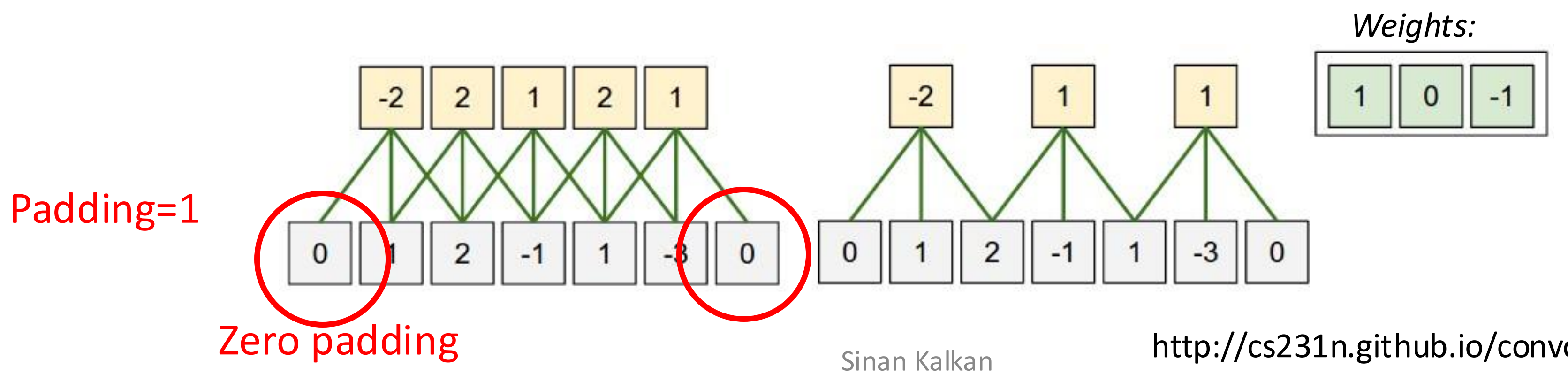
Weights:



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

Important parameters

- Depth (number of channels)
  - We will have more neurons getting input from the same receptive field
  - This is similar to the hidden neurons with connections to the same input
  - These neurons learn to become selective to the presence of different signals in the same receptive field
- How to handle the boundaries?
  - i. Option 1: Don't process the boundaries. Only process pixels on which convolution window can be placed fully.
  - ii. Option 2: Zero-pad the input so that convolution can be performed at the boundary pixels.



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

Padding illustration

- Only convolution layers are shown.
- Top: no padding ➔ layers shrink in size.
- Bottom: zero padding ➔ layers keep their size fixed.

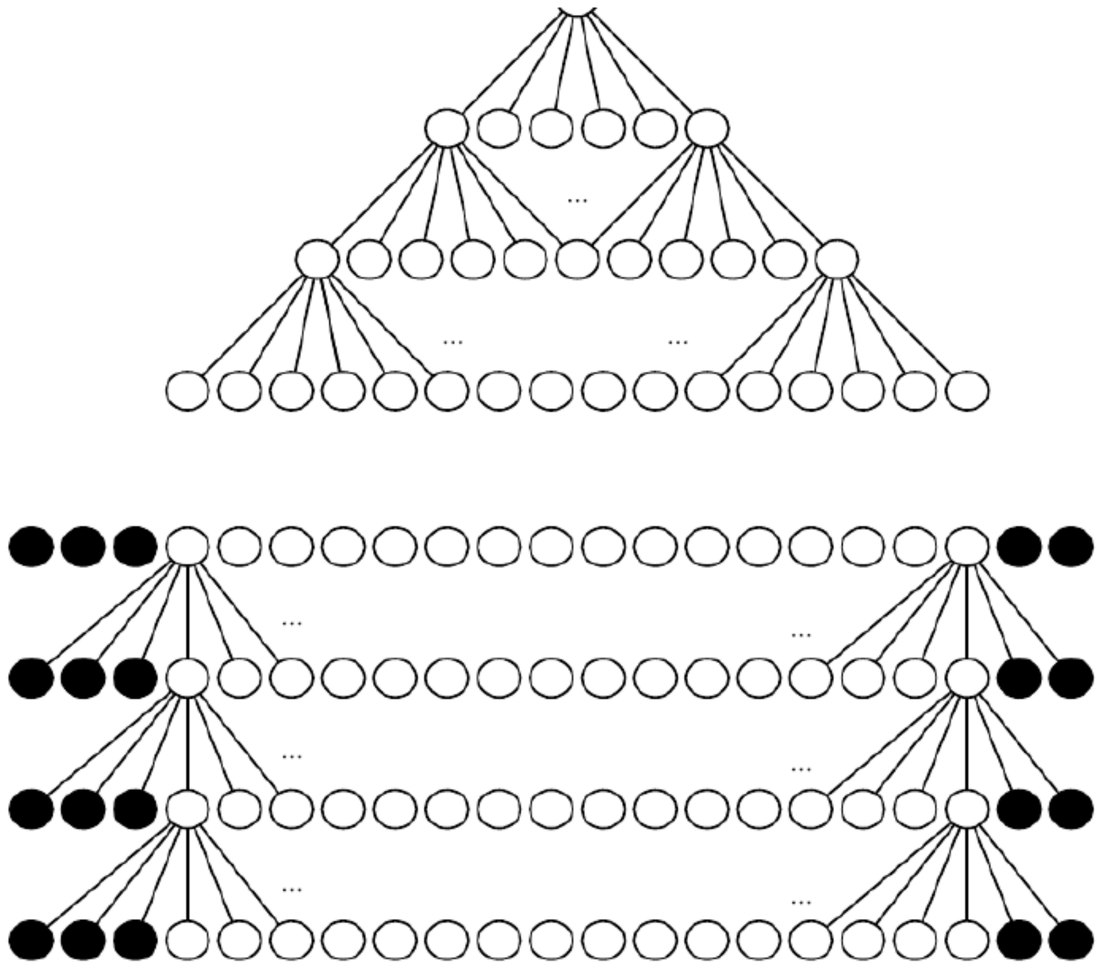


Figure 9.11: *The effect of zero padding on network size*: Consider a convolutional network with a kernel of width six at every layer. In this example, do not use any pooling, so only the convolution operation itself shrinks the network size. *Top)* In this convolutional network, we do not use any implicit zero padding. This causes the representation to shrink by five pixels at each layer. Starting from an input of sixteen pixels, we are only able to have three convolutional layers, and the last layer does not ever move the kernel, so arguably only two of the layers are truly convolutional. The rate of shrinking can be mitigated by using smaller kernels, but smaller kernels are less expressive and some shrinking is inevitable in this kind of architecture. *Bottom)* By adding five implicit zeroes to each layer, we prevent the representation from shrinking with depth. This allows us to make an arbitrarily deep convolutional network.

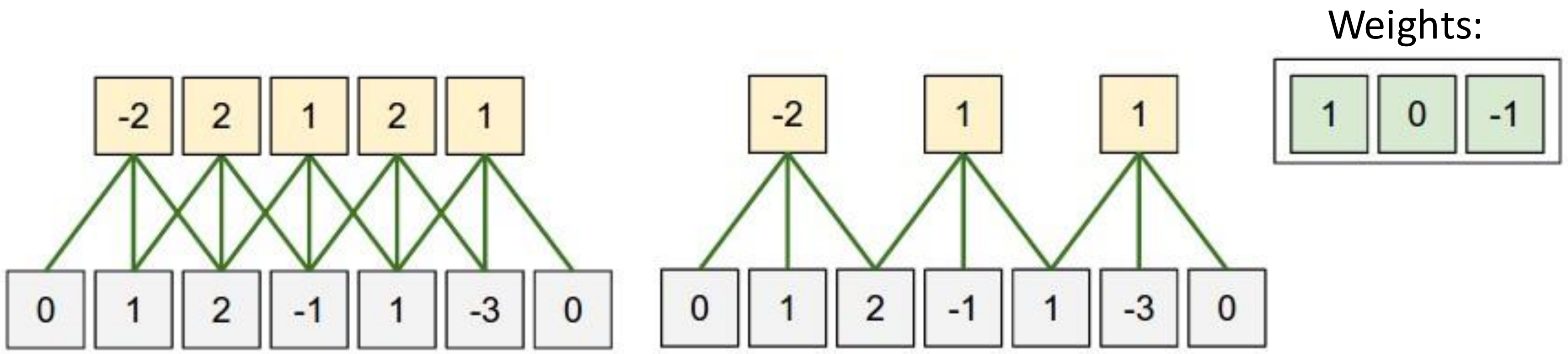
Figure: Goodfellow et al., "Deep Learning", MIT Press, 2016.

Size of the next layer

- Along a dimension:
  - W: Size of the input
  - F: Size of the receptive field
  - S: Stride
  - P: Amount of zero-padding
- Then: the number of neurons as the output of a convolution layer:

$$\frac{W - F + 2P}{S} + 1$$

- If this number is not an integer, your strides are incorrect and your neurons cannot tile to cover the input volume



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

Size of the next layer

- Arranging these hyperparameters can be problematic
- Example:
- If W=10, P=0, and F=3, then

$$\frac{W - F + 2P}{S} + 1 = \frac{10 - 3 + 0}{S} + 1 = \frac{7}{S} + 1$$

i.e., S cannot be an integer other than 1 or 7.

- Zero-padding is your friend here.

Real example – AlexNet (Krizhevsky et al., 2012)

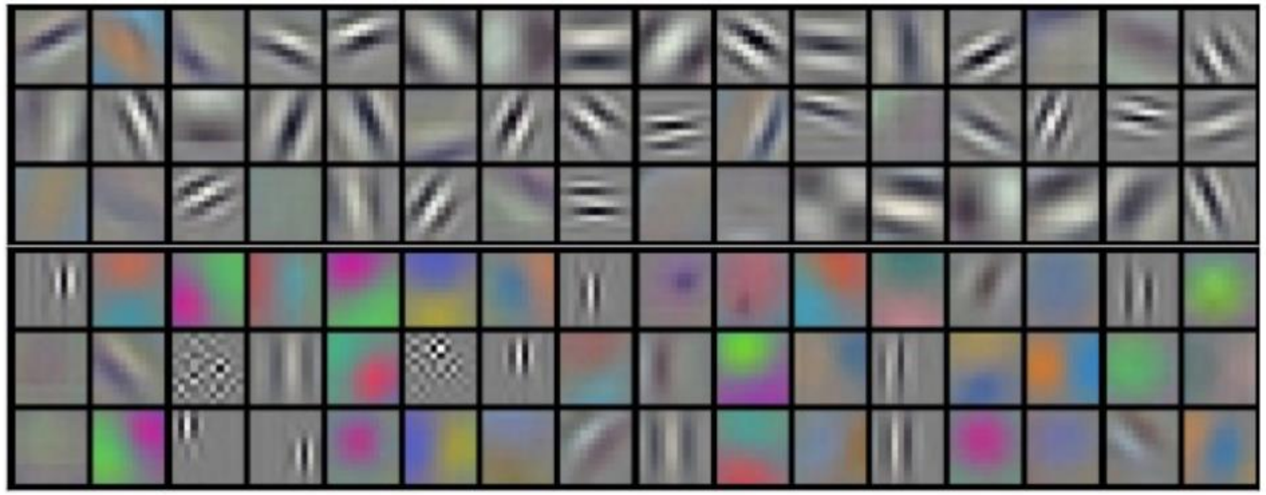
- Image size: 227×227×3
- W=227, F=11, S=4, P=0 ➔ $\frac{227-11}{S} + 1 = 55$

  (55 => the width of the convolution layer)
- Convolution layer: 55×55×96 neurons

  (96: the depth, the number of channels)
- Therefore, the first layer has 55×55×96 = 290,400 neurons
  - Each has 11×11×3 receptive field ➔ 363 weights and 1 bias
  - Then, 290,400×364 = 105,705,600 parameters just for the first convolution layer (if there were no weight sharing)
  - With weight sharing: 96 x 364 = 34,944

Real example – AlexNet (Krizhevsky et al., 2012)

- However, we can share the parameters
  - For each channel (slice of depth), have the same set of weights
  - If 96 channels, this means 96 different set of weights
  - Then, 96×364 = 34,944 parameters
  - 364 weights shared by 55×55 neurons in each channel



Example filters learned by Krizhevsky et al. Each of the 96 filters shown here is of size [11x11x3], and each one is shared by the 55*55 neurons in one depth slice. Notice that the parameter sharing assumption is relatively reasonable: If detecting a horizontal edge is important at some location in the image, it should intuitively be useful at some other location as well due to the translationally-invariant structure of images. There is therefore no need to relearn to detect a horizontal edge at every one of the 55*55 distinct locations in the Conv layer output volume.

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

More on connectivity

Small RF & Stacking

- E.g., 3 CONV layers of 3x3 RFs
- Pros:
  - Same extent for these example figures
  - With non-linearity added on 2nd and 3rd layers ➔ More expressive! More representational capacity!
  - Less parameters:

    3 layers x [(3 x 3 x C) x C] = 27CxC
- Cons?

Large RF & Single Layer

- 7x7 RFs of single CONV layer
- Pros?
- Cons:
  - One layer => Linear capacity
  - More parameters:

    (7x7xC)xC = 49CxC

So, we prefer a stack of small filter sizes against big ones

Implementation Details: NumPy example

- Suppose input is `X` of shape (11,11,4)
- Depth slice at depth d (i.e., channel d): `X[:,:,d]`
- Depth column at position (x,y): `X[x,y,:]`
- F: 5, P:0 (no padding), S=2
  - Output volume (V) width, height = (11-5+0)/2+1 = 4
- Example computation for some neurons in first channel:

```
V[0,0,0] = np.sum(X[:5,:5,:] * W0) + b0
V[1,0,0] = np.sum(X[2:7,:5,:] * W0) + b0
V[2,0,0] = np.sum(X[4:9,:5,:] * W0) + b0
V[3,0,0] = np.sum(X[6:11,:5,:] * W0) + b0
```

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

- Note that this is just along one dimension (x)

Implementation Details: NumPy example

- A second activation map (channel):

```
V[0,0,1] = np.sum(X[:5,:5,:] * W1) + b1
V[1,0,1] = np.sum(X[2:7,:5,:] * W1) + b1
V[2,0,1] = np.sum(X[4:9,:5,:] * W1) + b1
V[3,0,1] = np.sum(X[6:11,:5,:] * W1) + b1
V[0,1,1] = np.sum(X[:5,2:7,:] * W1) + b1
V[2,3,1] = np.sum(X[4:9,6:11,:] * W1) + b1
```

(example of going along y)

(or along both)

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

Summary. To summarize, the Conv Layer:

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

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

Types of Convolution: Unshared convolution

- In some cases, sharing the weights does not make sense
  - When?

- Different parts of the input might require different types of processing/features

- In such a case, we just have a network with local connectivity

- E.g., a face.
  - Features are not repeated across the space.

Types of Convolution:
Dilated (Atrous) Convolution

Purpose: Increase effective receptive field size without increasing parameters.

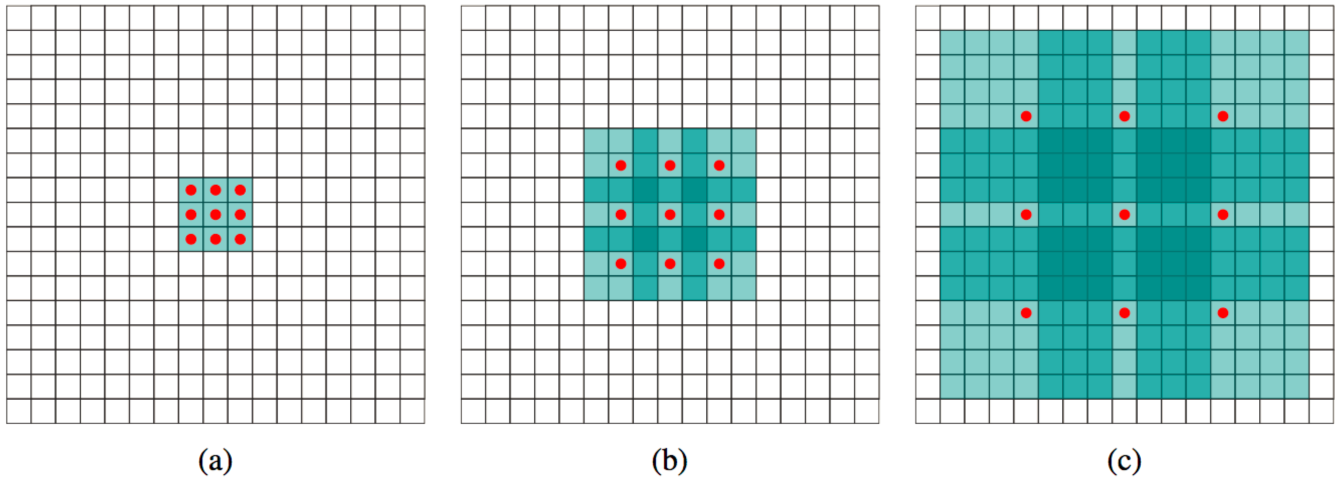
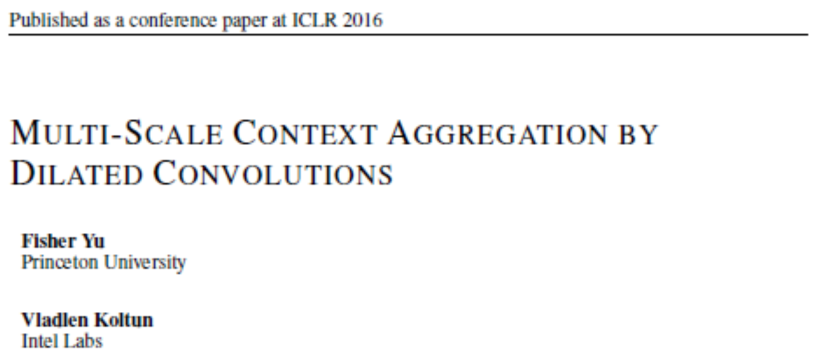


Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element in F_2 has a receptive field of 7×7. (c) F_3 is produced from F_2 by a 4-dilated convolution; each element in F_3 has a receptive field of 15×15. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.

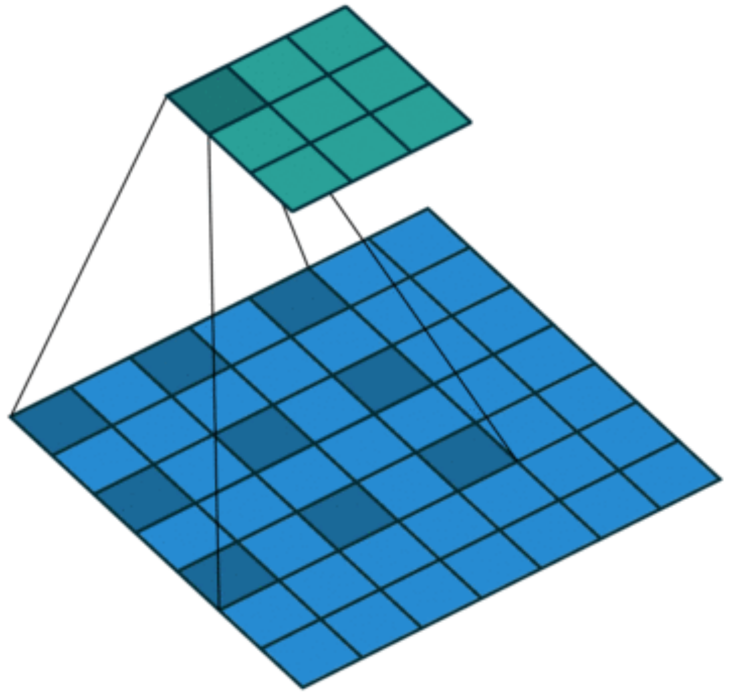


Published as a conference paper at ICLR 2016

MULTI-SCALE CONTEXT AGGREGATION BY DILATED CONVOLUTIONS

Fisher Yu
Princeton University

Vladlen Koltun
Intel Labs



https://github.com/vdumoulin/conv_arithmetic

Types of Convolution: Transposed Convolution

Purpose: Increasing layer width+height (upsampling).

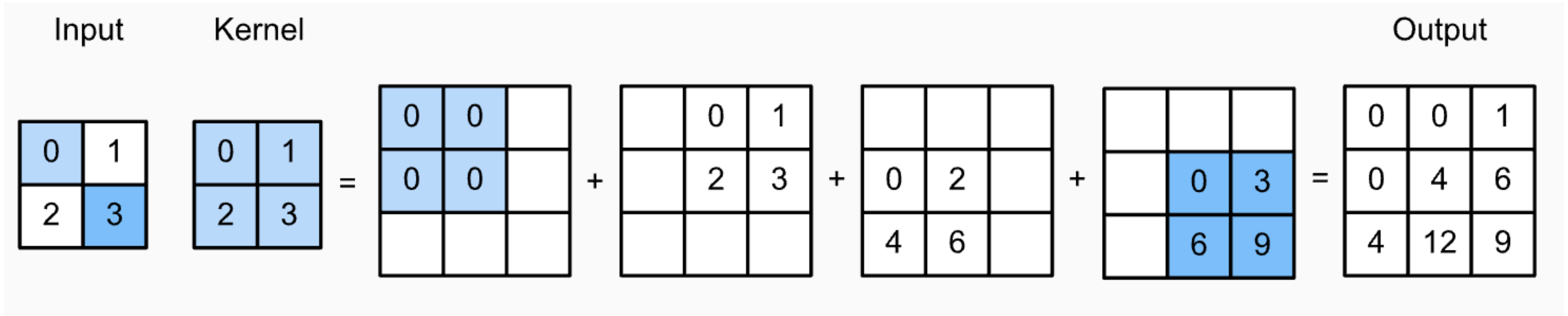


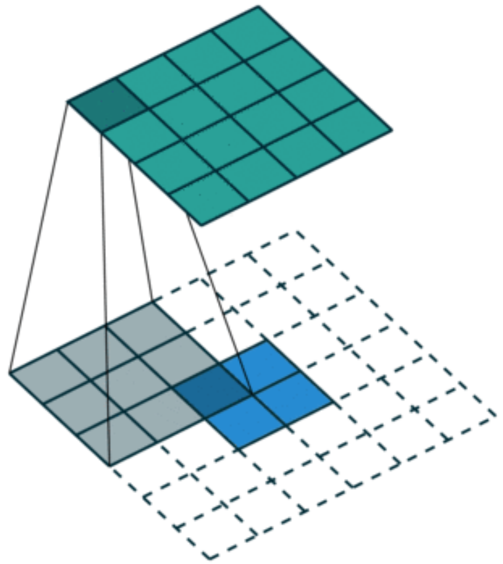
Fig. 13.10.1 Transposed convolution layer with a 2×2 kernel.

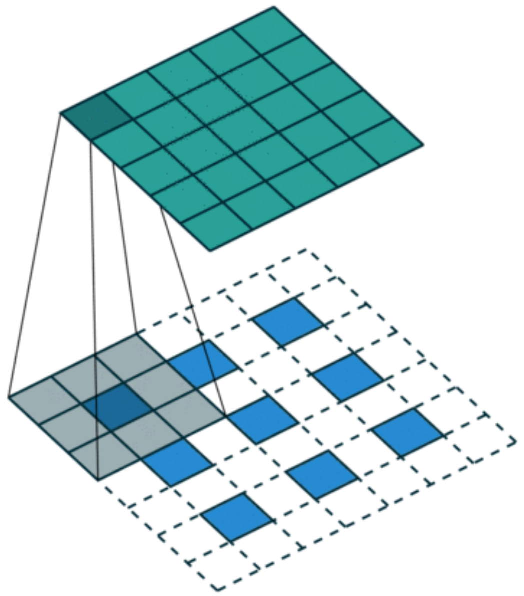
Figure: https://d2l.ai/chapter_computer-vision/transposed-conv.html

The size of the output:

- Regular convolution: $O = \frac{W-F+2\times P}{S} + 1$
- Transpose convolution: $W = (O-1) \times S + F - 2 \times P$

Types of Convolution:
Upsampling with Padding or Dilation

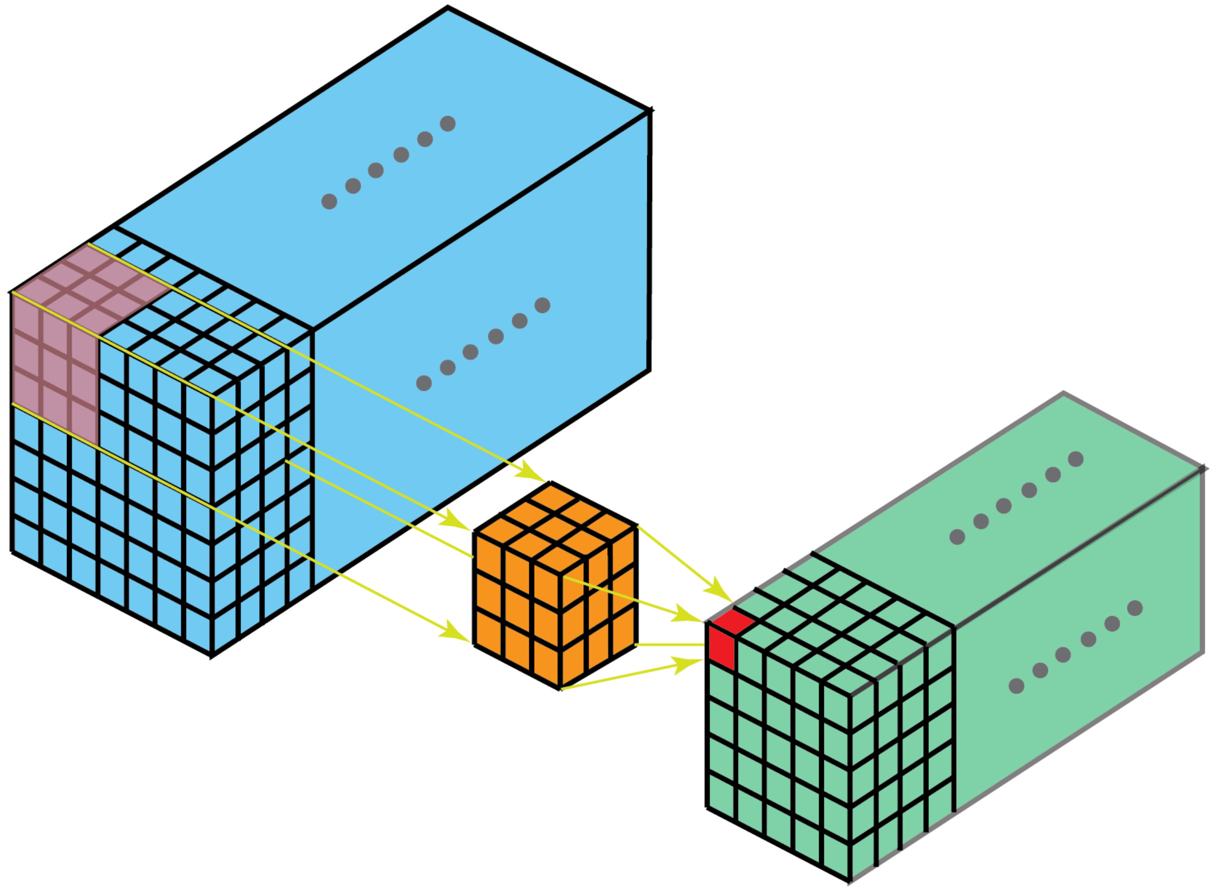




https://github.com/vdumoulin/conv_arithmetic

Types of Convolution: 3D Convolution

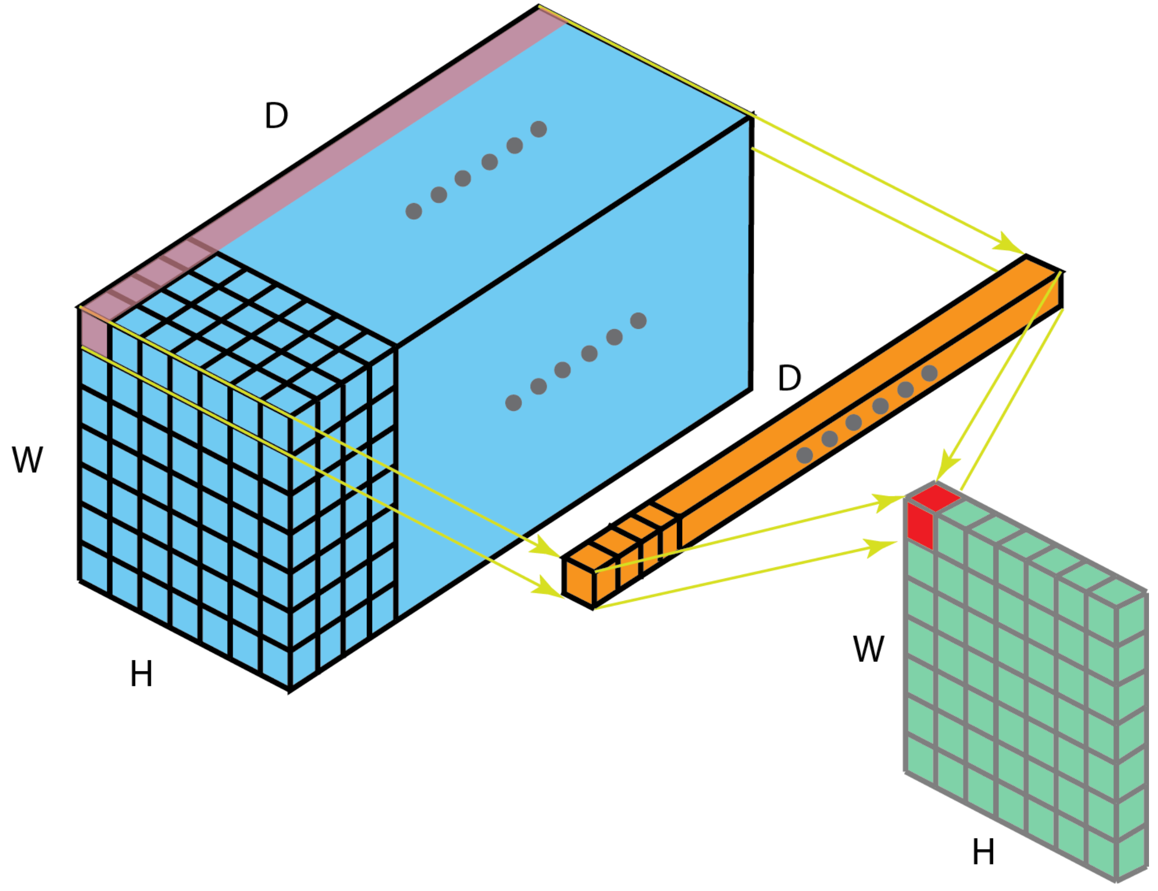
Purpose: Work with 3D data, e.g. learn spatial + temporal representations for videos.



https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Types of Convolution: 1x1 Convolution

Purpose: Reduce number of channels.

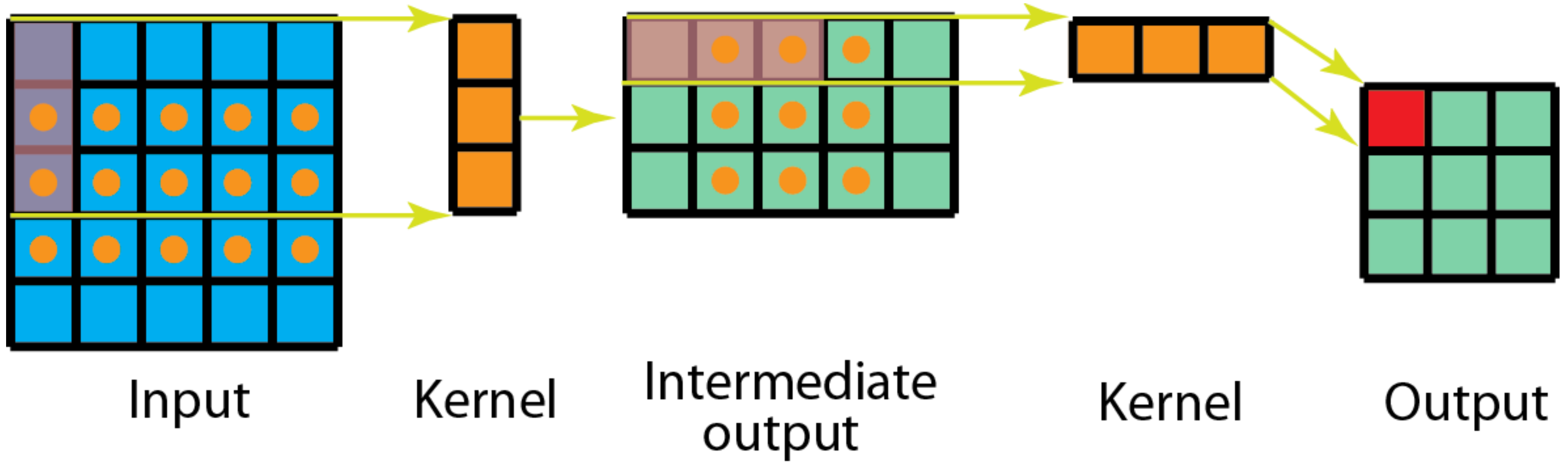


https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Types of Convolution: Separable Convolution

Purpose: Reduce number of parameters and multiplications.

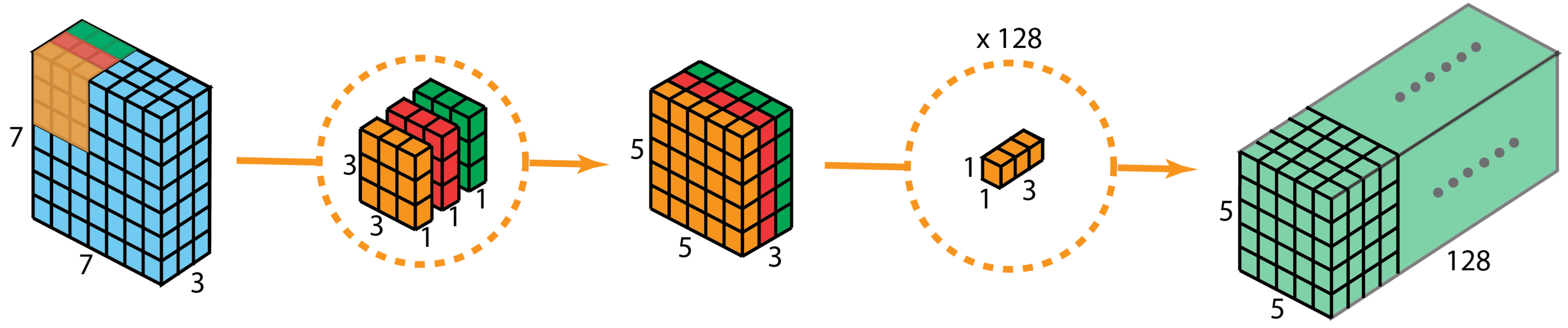
$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$



https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

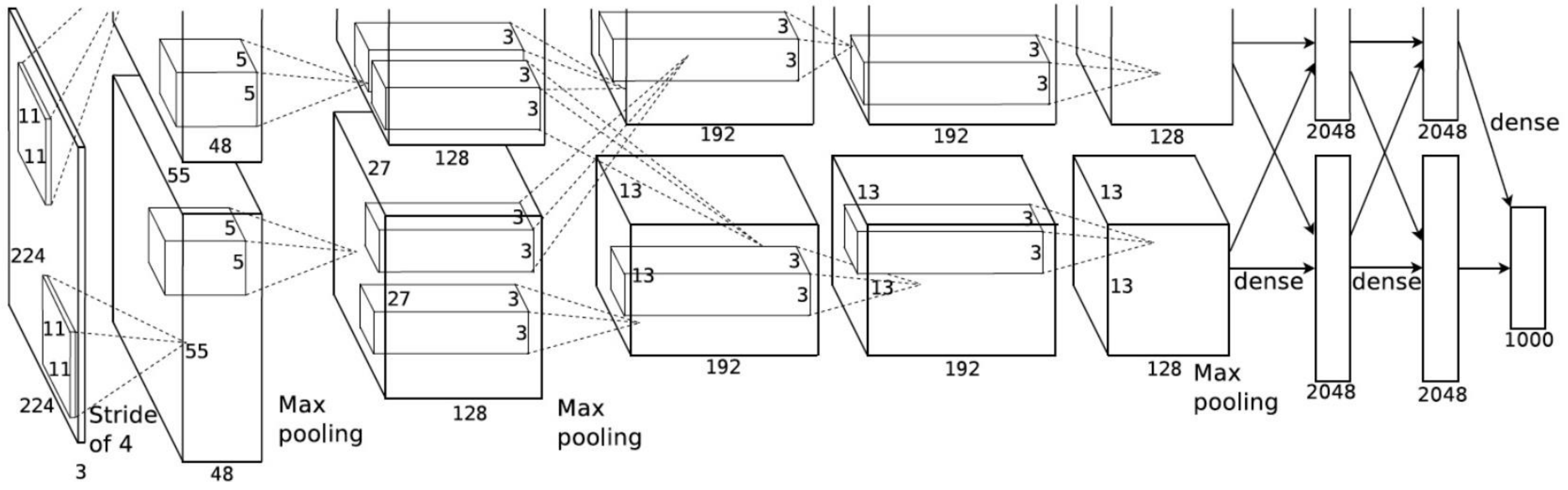
Types of Convolution: Depth-wise Separable Convolution

Purpose: Reduce number of parameters and multiplications.

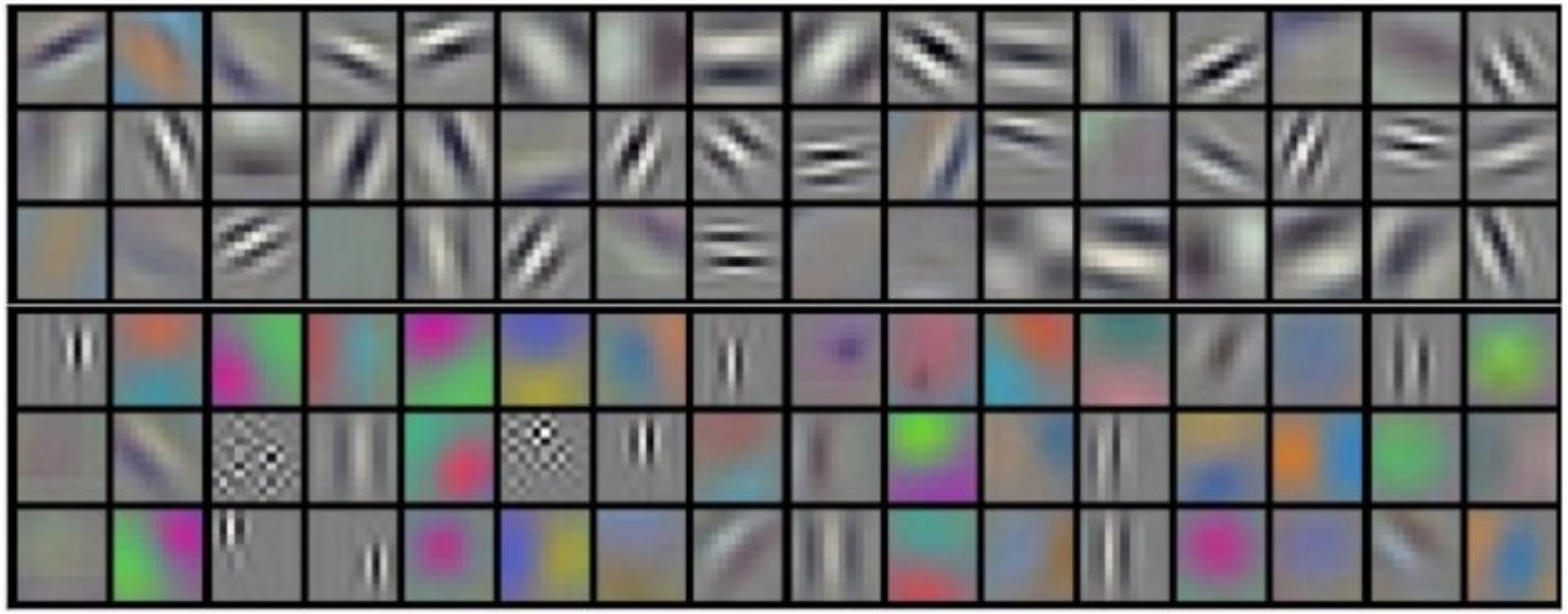


https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Types of Convolution: Group Convolution

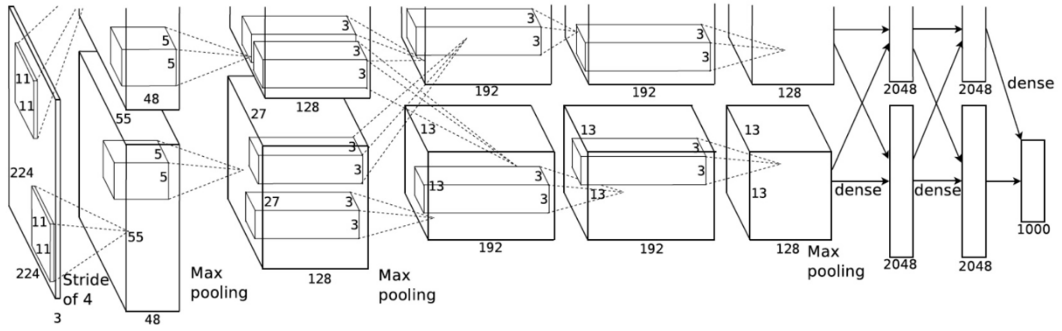


AlexNet (Krizhevsky et al.)

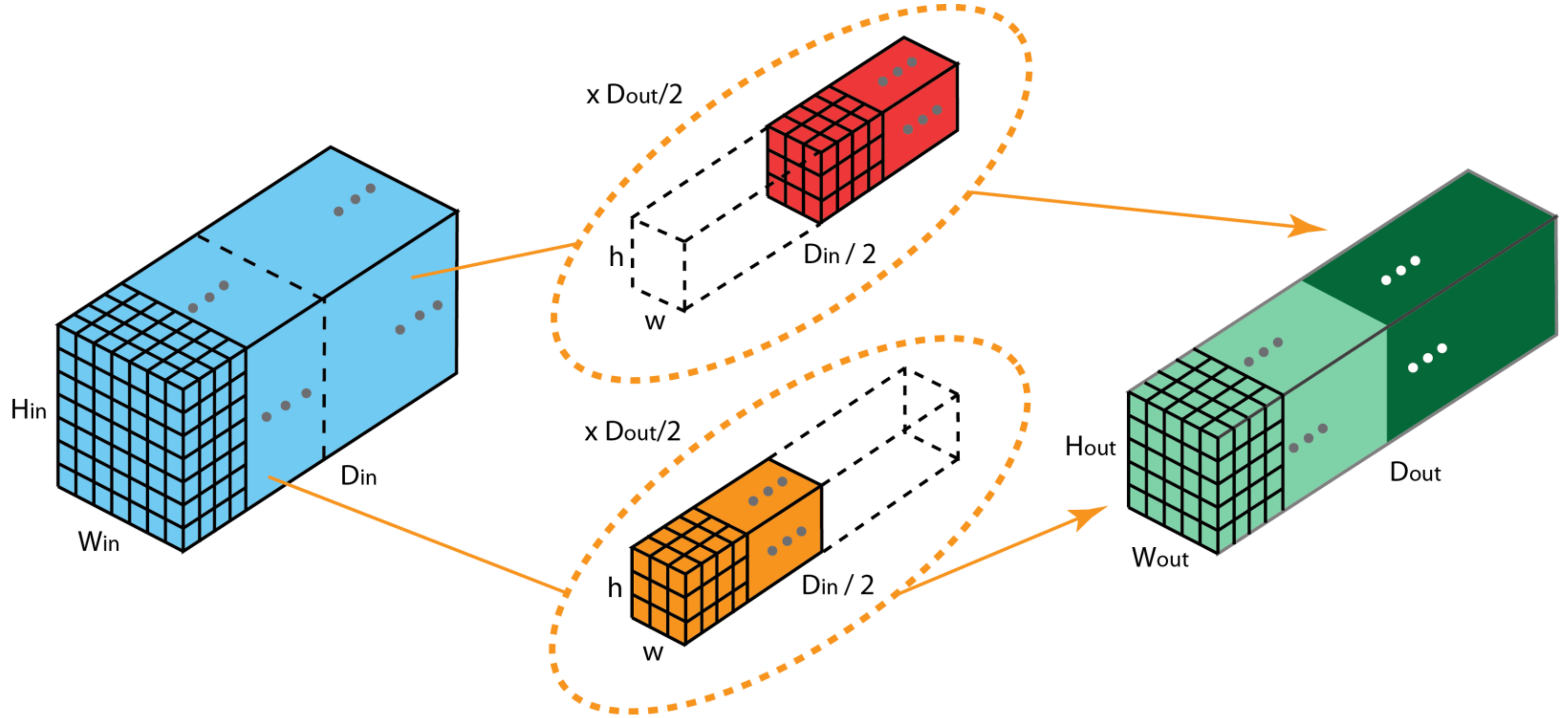


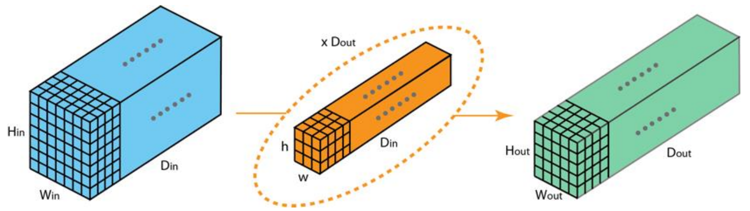
Types of Convolution:
Group Convolution

Purpose: Reduce number of parameters and multiplications.



AlexNet

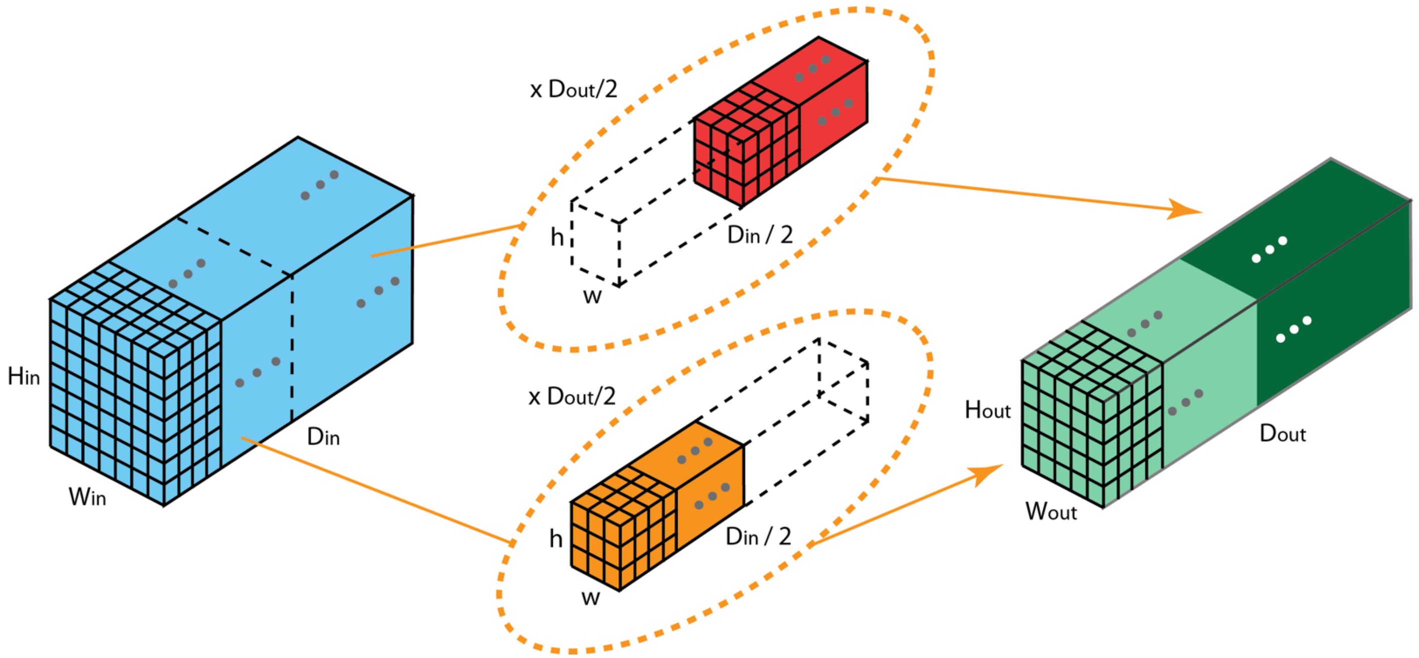




Normal Convolution

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Types of Convolution: Group Convolution



- Benefits:
  - Efficiency in training (distribute groups to different GPUs)
  - Decrease in # of parameters as the # of groups increases
  - Better performance?

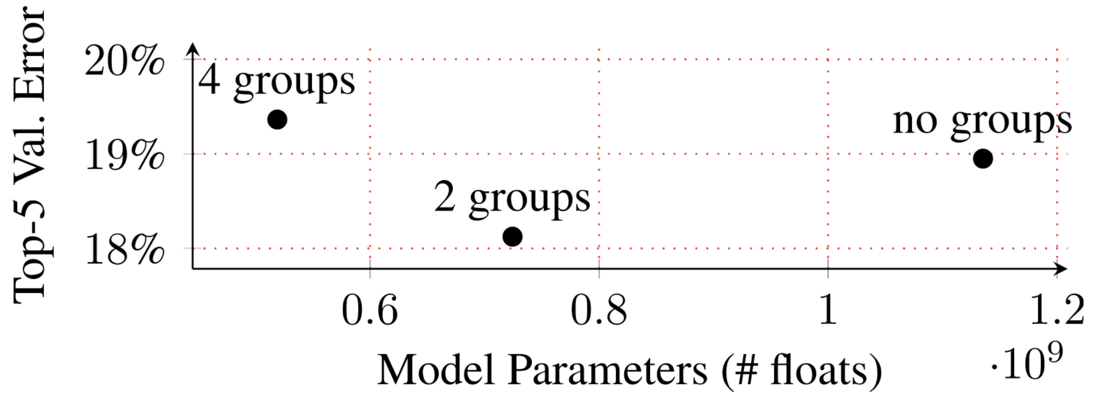


Figure: https://blog.yani.ai/filter-group-tutorial/

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Types of Convolution: Deformable Convolution

Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., & Wei, Y. (2017). Deformable convolutional networks. ICCV.

Purpose: Flexible receptive field.

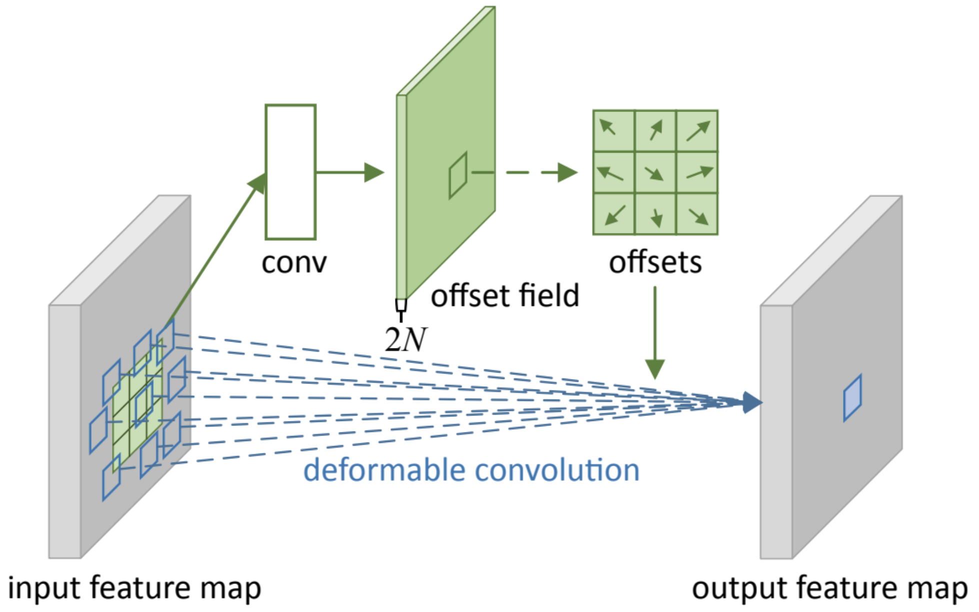


Figure 2: Illustration of 3 × 3 deformable convolution.

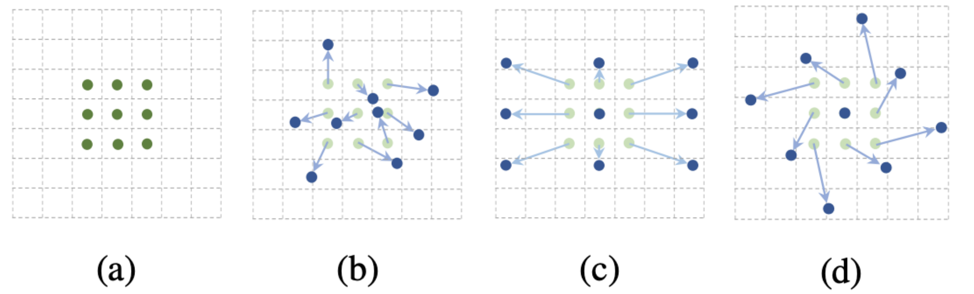
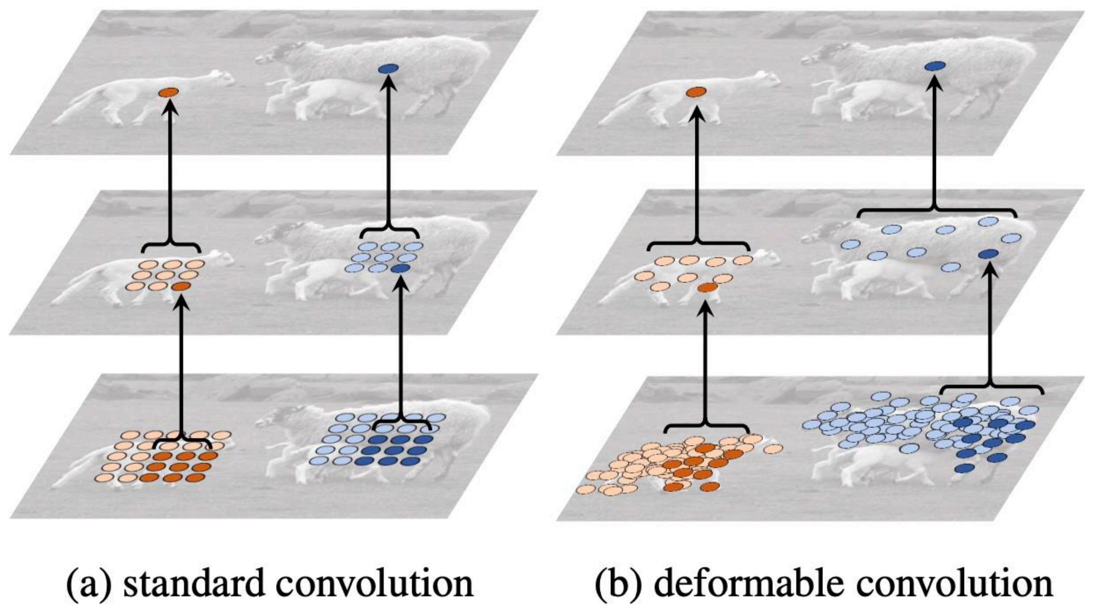


Figure 1: Illustration of the sampling locations in 3 × 3 standard and deformable convolutions. (a) regular sampling grid (green points) of standard convolution. (b) deformed sampling locations (dark blue points) with augmented offsets (light blue arrows) in deformable convolution. (c)(d) are special cases of (b), showing that the deformable convolution generalizes various transformations for scale, (anisotropic) aspect ratio and rotation.



Types of Convolution: Deformable Convolution

Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., & Wei, Y. (2017). Deformable convolutional networks. ICCV.

$$\mathcal{R} = \{(-1,-1), (-1,0), \ldots, (0,1), (1,1)\}$$

defines a 3×3 kernel with dilation 1.

For each location $\mathbf{p}_0$ on the output feature map $\mathbf{y}$, we have

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n), \qquad (1)$$

where $\mathbf{p}_n$ enumerates the locations in $\mathcal{R}$.

In deformable convolution, the regular grid $\mathcal{R}$ is augmented with offsets $\{\Delta \mathbf{p}_n | n = 1, ..., N\}$, where $N = |\mathcal{R}|$. Eq. (1) becomes

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta \mathbf{p}_n). \qquad (2)$$

Now, the sampling is on the irregular and offset locations $\mathbf{p}_n + \Delta \mathbf{p}_n$. As the offset $\Delta \mathbf{p}_n$ is typically fractional, Eq. (2) is implemented via bilinear interpolation as

$$\mathbf{x}(\mathbf{p}) = \sum_{\mathbf{q}} G(\mathbf{q}, \mathbf{p}) \cdot \mathbf{x}(\mathbf{q}), \qquad (3)$$

where $\mathbf{p}$ denotes an arbitrary (fractional) location ($\mathbf{p} = \mathbf{p}_0 + \mathbf{p}_n + \Delta \mathbf{p}_n$ for Eq. (2)), $\mathbf{q}$ enumerates all integral spatial locations in the feature map $\mathbf{x}$, and $G(\cdot, \cdot)$ is the bilinear interpolation kernel. Note that G is two dimensional. It is separated into two one dimensional kernels as

$$G(\mathbf{q}, \mathbf{p}) = g(q_x, p_x) \cdot g(q_y, p_y), \qquad (4)$$

where $g(a, b) = max(0, 1 - |a - b|)$. Eq. (3) is fast to compute as $G(\mathbf{q}, \mathbf{p})$ is non-zero only for a few $\mathbf{q}$s.

In the deformable convolution Eq. (2), the gradient w.r.t. the offset $\Delta \mathbf{p}_n$ is computed as

$$\begin{aligned} \frac{\partial \mathbf{y}(\mathbf{p}_0)}{\partial \Delta \mathbf{p}_n} &= \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \frac{\partial \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta \mathbf{p}_n)}{\partial \Delta \mathbf{p}_n} \\ &= \sum_{\mathbf{p}_n \in \mathcal{R}} \left[\mathbf{w}(\mathbf{p}_n) \cdot \sum_{\mathbf{q}} \frac{\partial G(\mathbf{q}, \mathbf{p}_0 + \mathbf{p}_n + \Delta \mathbf{p}_n)}{\partial \Delta \mathbf{p}_n} \mathbf{x}(\mathbf{q}) \right], \end{aligned} \qquad (7)$$

where the term $\frac{\partial G(\mathbf{q}, \mathbf{p}_0 + \mathbf{p}_n + \Delta \mathbf{p}_n)}{\partial \Delta \mathbf{p}_n}$ can be derived from Eq. (4). Note that the offset $\Delta \mathbf{p}_n$ is 2D and we use $\partial \Delta \mathbf{p}_n$ to denote $\partial \Delta p_n^x$ and $\partial \Delta p_n^y$ for simplicity.