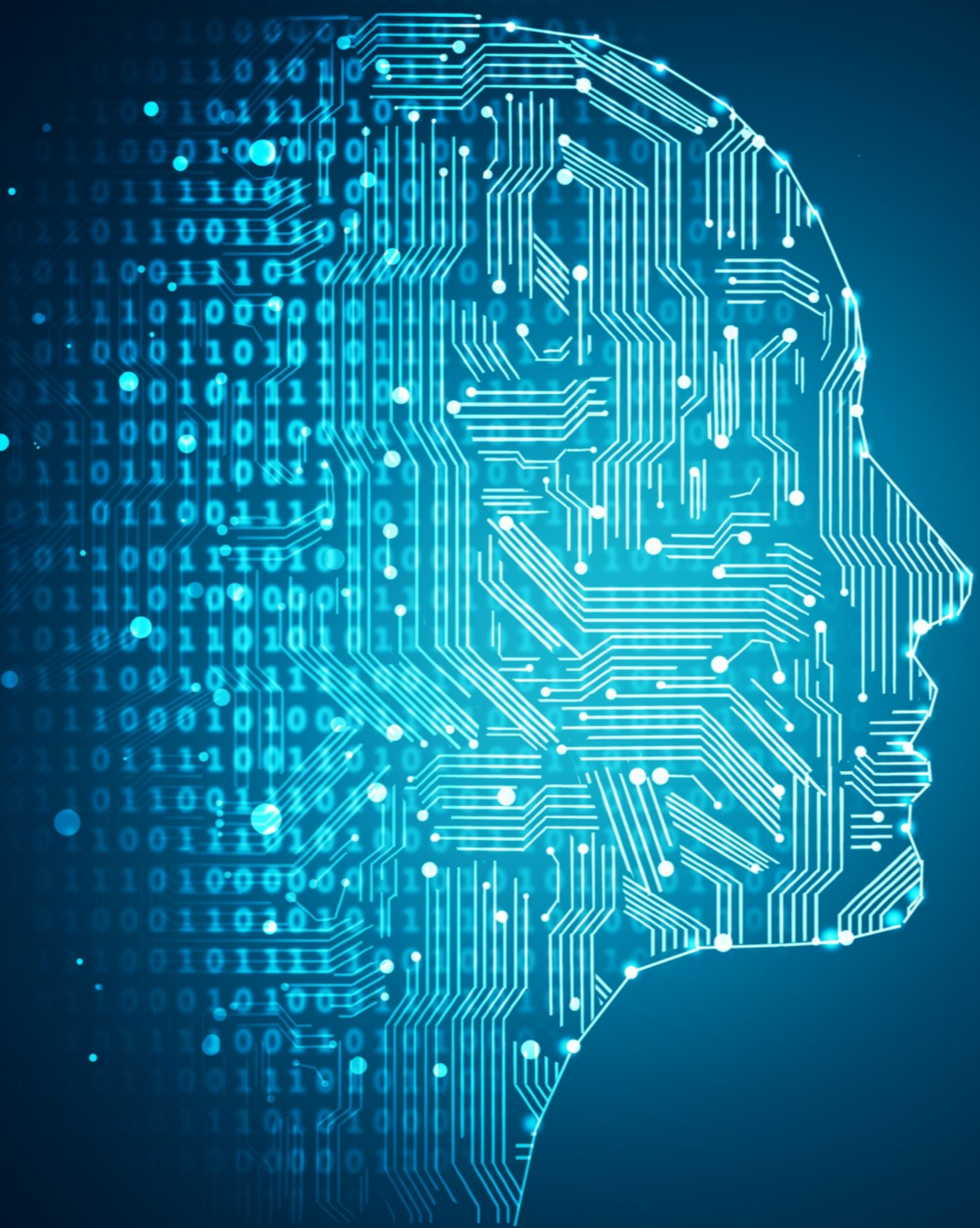


AI6103 Multi-layer Perceptron & Convolutional Neural Network

Boyang Li, Albert

School of Computer Science and Engineering

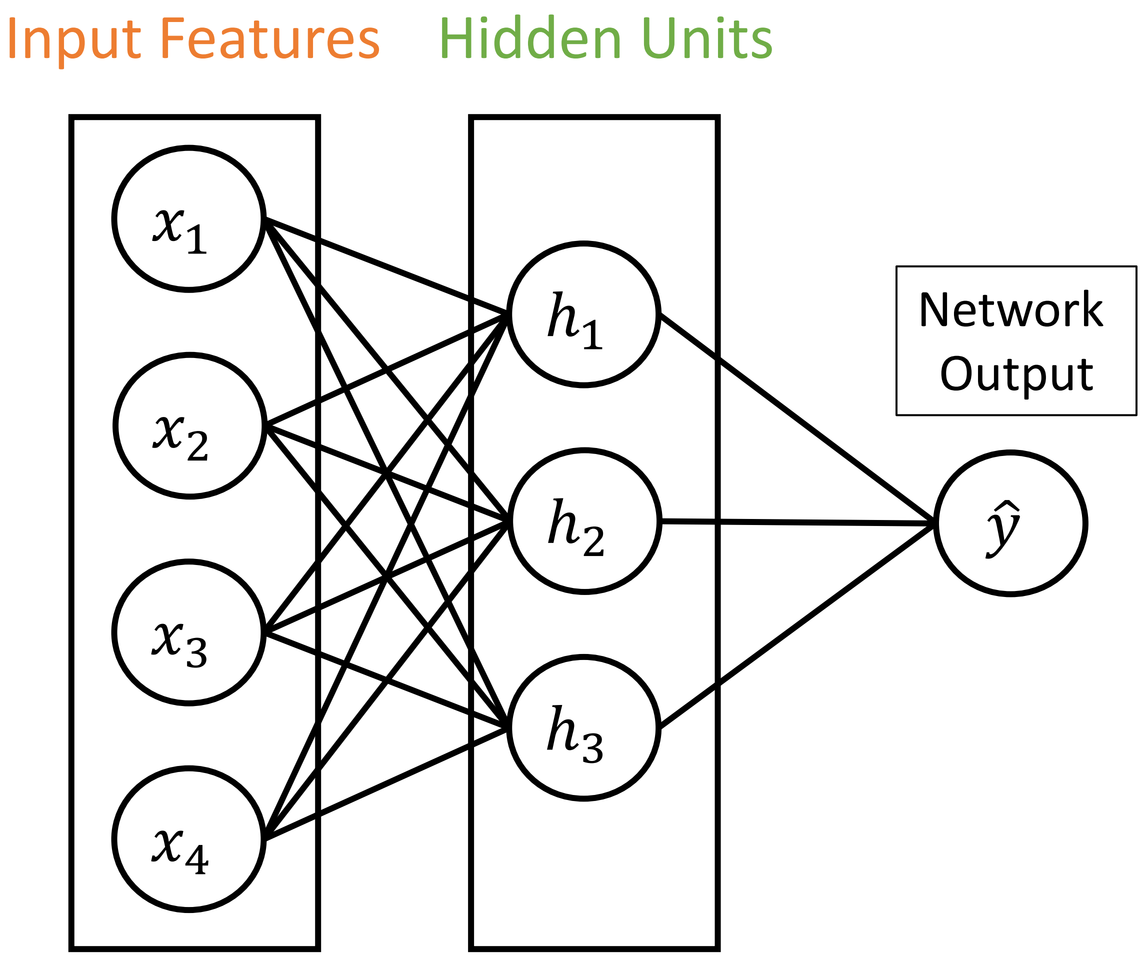
Nanyang Technological University

Outline

- Multi-layer Perceptron
- Multi-class classification
    - Cross-entropy loss
- Convolutional Networks
    - Convolution
    - Activation: ReLU, Leaky ReLU, etc.
    - Batch Normalization
    - Skip Connections
- The ResNet Architecture
- Some Image Recognition Datasets

Multi-layer Perceptron

Multi-layer Perceptron



Multi-layer Perceptron

- Every intermediate nodes have a set of $\boldsymbol{\beta}$
- Layer 1, neuron 1

$$z_{1,1} = \sigma\left(\boldsymbol{\beta}_{1,1}^{\top}\boldsymbol{x}\right)$$

- Layer 1, neuron 2

$$z_{1,2} = \sigma\left(\boldsymbol{\beta}_{1,2}^{\top}\boldsymbol{x}\right)$$

- Put in matrix form

$$\boldsymbol{z}_1 = \begin{pmatrix} z_{1,1} \\ z_{1,2} \end{pmatrix} = \sigma\left(\begin{pmatrix} \boldsymbol{\beta}_{1,1}^{\top} \\ \boldsymbol{\beta}_{1,2}^{\top} \end{pmatrix}\boldsymbol{x}\right) = \sigma(W_1\boldsymbol{x})$$

Multi-layer Perceptron

- Create L layers

$$f(x) = \sigma\left(w_L \dots \sigma\left(W_2\, \sigma(W_1 \boldsymbol{x})\right)\right)$$

- These matrices can be of arbitrary size, as long as the multiplication works out
- W_1 is 10 by p, W_2 is 100 by 10, W_3 is 341 by 100, etc
- w_L is a vector, which is 1 by K.
- The output $f(x)$ is a scalar, denoting $P(y = 1|x)$

Multi-layer Perceptron

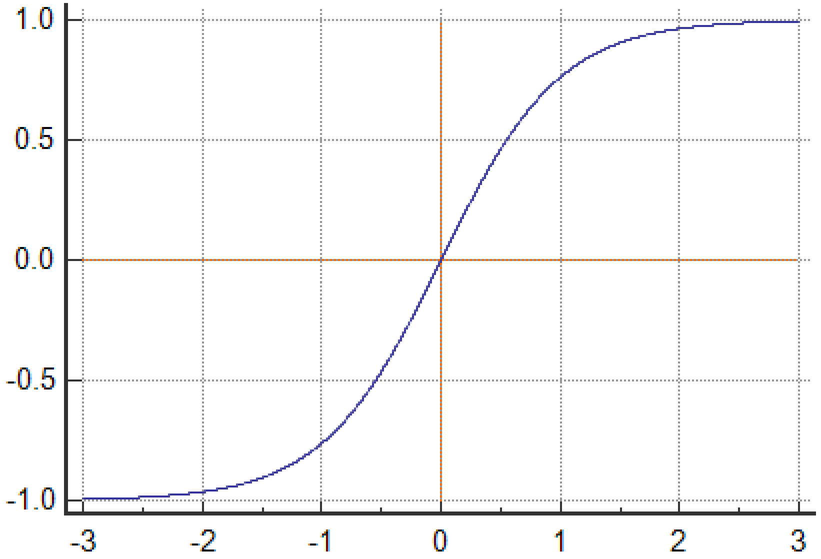
- Create L layers

$$f(x) = \sigma\left(w_L \ldots \sigma\big(W_2\, \sigma(W_1 \boldsymbol{x})\big)\right)$$

- These matrices can be of arbitrary size, as long as the multiplication works out
- W_1 is 10 by p, W_2 is 100 by 10, W_3 is 341 by 100, etc
- w_L is a vector, which is 1 by K.

Activation Function: Hyperbolic Tangent

- $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$
- Like sigmoid, squashes $(-\infty, +\infty)$ to a finite range.
- Unlike sigmoid, it has a larger range $(-1, 1)$



Multi-class Classification

$$\boldsymbol{z}_L = w_L \ldots \sigma\big(W_2\, \sigma(W_1 \boldsymbol{x})\big)$$

- We now have M classes.
- w_L is a vector, which is M by K.
- We put $\boldsymbol{z}_L$ through a softmax function

$$\boldsymbol{\theta} = \mathrm{softmax}(\boldsymbol{z}_L) = \left[\frac{\exp z_{L,1}}{\sum_j \exp z_{L,j}}, \frac{\exp z_{L,2}}{\sum_j \exp z_{L,j}}, \ldots, \frac{\exp z_{L,M}}{\sum_j \exp z_{L,j}}\right]$$

- The logits $z_{L,i}$ are unbounded. They could be negative or greater than 1
- Softmax ensures that the output is a valid probability distribution

  $(\theta_i > 0, \sum \theta_i = 1)$

Multi-class Cross-entropy

Binary Cross-entropy

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \mathrm{y_i} \log f_W(\boldsymbol{x}_i) + (1 - \mathrm{y_i}) \log\big(1 - f_W(\boldsymbol{x}_i)\big)$$

Multi-class Cross-entropy

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \mathbf{y}_{\mathrm{i}}^{\top} \log f_W(\boldsymbol{x}_i)$$

$f_W(\boldsymbol{x}_i)$ is a vector of probabilities, such as the output of the softmax operation.

$\mathbf{y}_{\mathrm{i}}$ is a one-hot vector [0, … 1, …, 0] with the GT class set to 1.

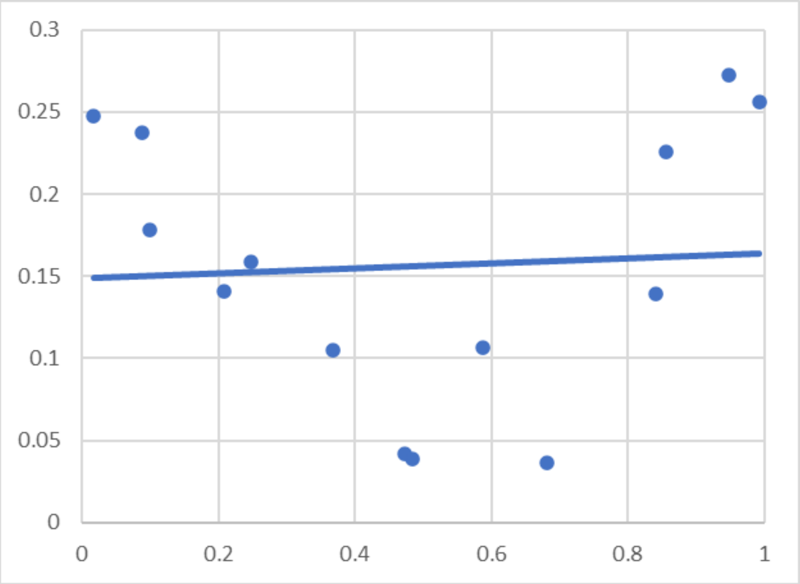
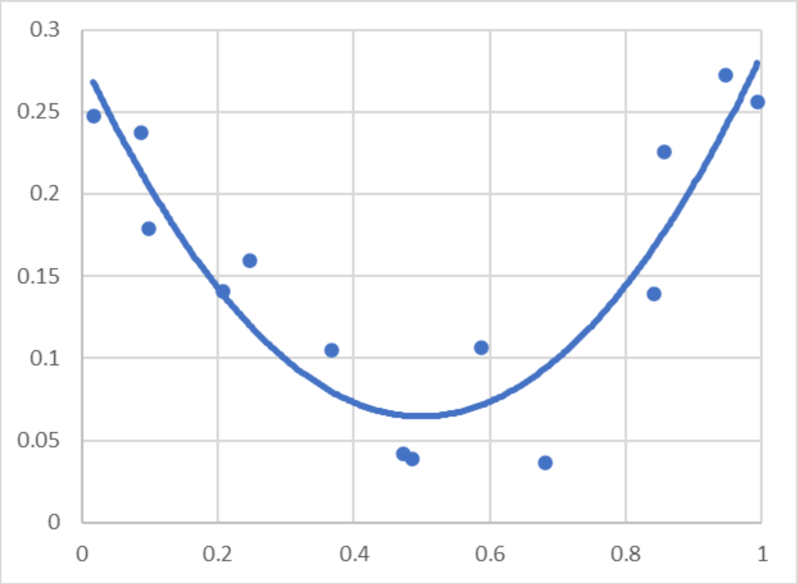
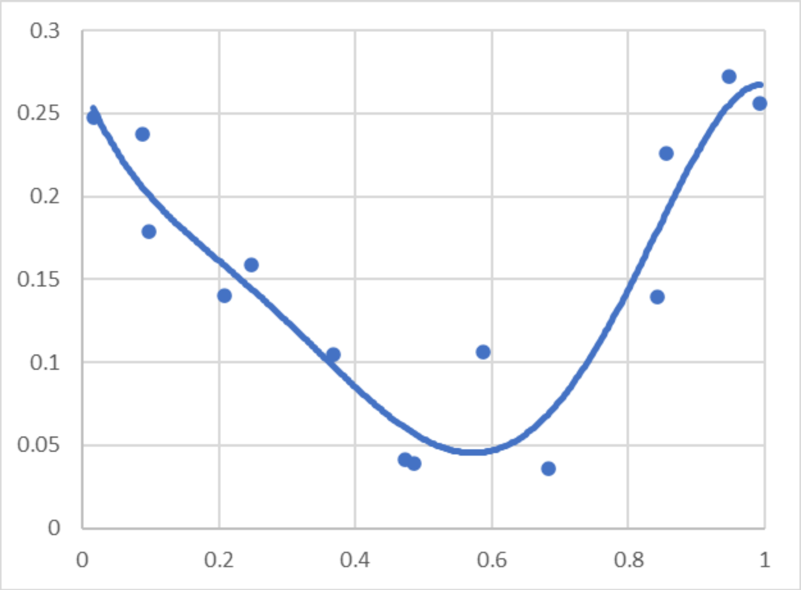
Multi-layer Perceptron is Powerful

- **The Universal Approximation Theorem**
- A two-layer MLP can represent an arbitrary “well-behaving” function in the domain of real numbers.
- That is, if we know the ground-truth function, we can design W such that the network is very close to the function.

Multi-layer Perceptron is Powerful

- **The Universal Approximation Theorem**
- A two-layer MLP can represent an arbitrary “well-behaving” function in the domain of real numbers.
- However, the real question is if we can learn such a function from data without knowing the ground-truth function.
- The learning of MLPs turns out to be hard.

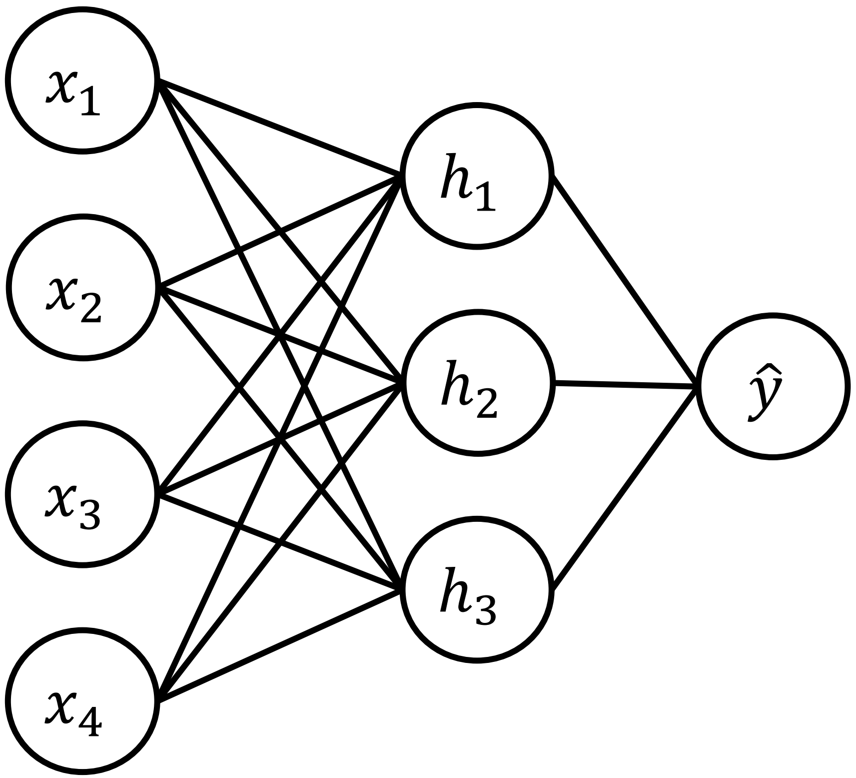
Underfitting vs. Overfitting

Straight Line	Quadratic Curve	5^{th} -order Curve
		
Not describing the data	Describing the data well	Taking the data too literally

Our goal is not to only fit training data but to generalize to unseen data!

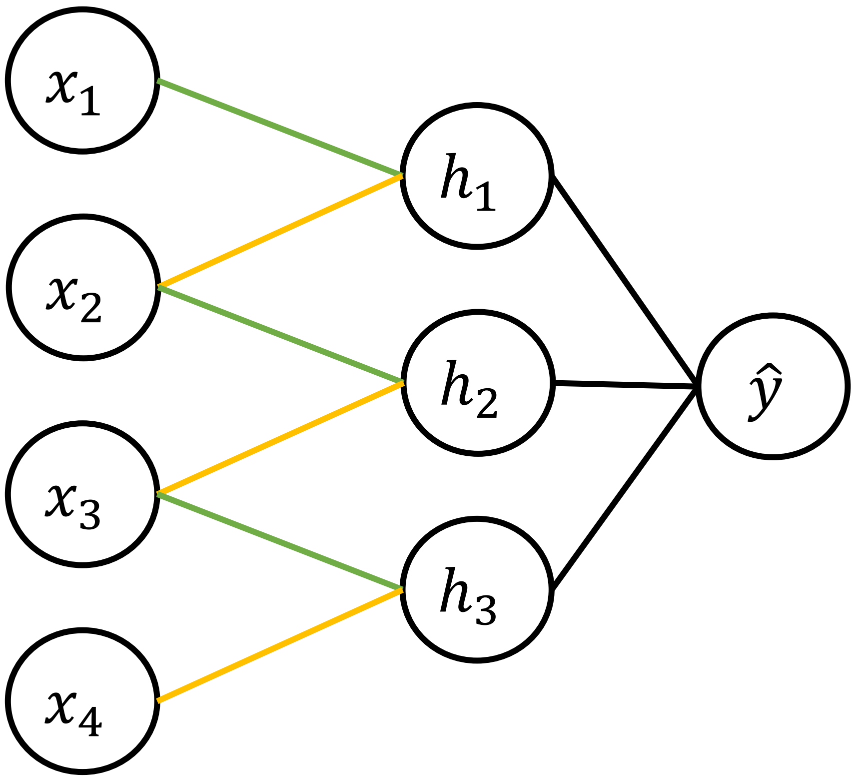
Convolutional Networks

Convolutional Neural Network



- MLPs have many parameters
- 4x3=12 parameters between $\boldsymbol{x}$ and $\boldsymbol{h}$
- When the amount of data stays the same, the more parameters, the more difficult to estimate them accurately. This may lead to overfitting.

1D Convolutional Neural Network



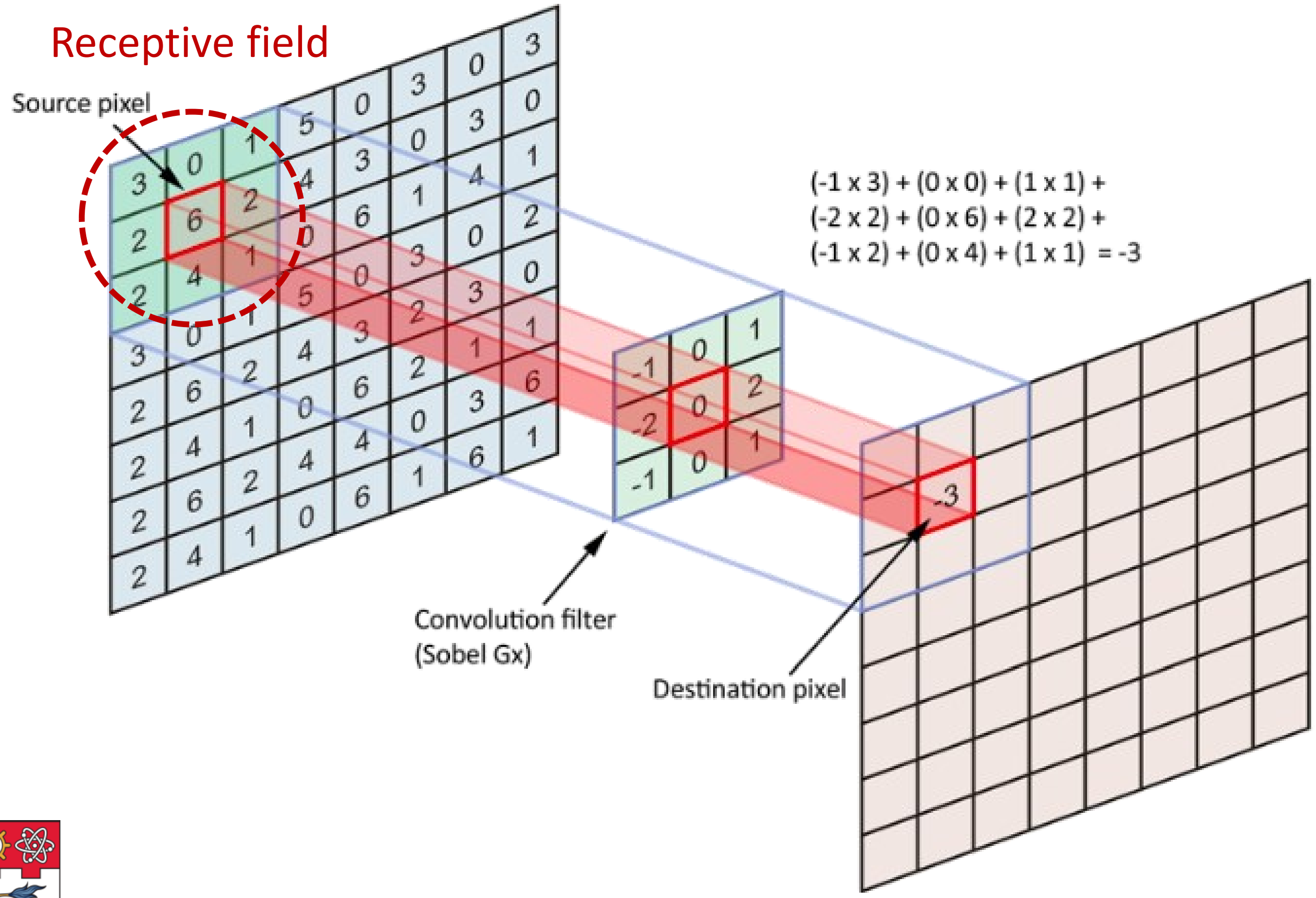
- Convolutional networks reduce the total number of parameters by
  - Local connection
  - Weight sharing
- Now only 2 weights to learn

2D Convolution

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

2D Convolution



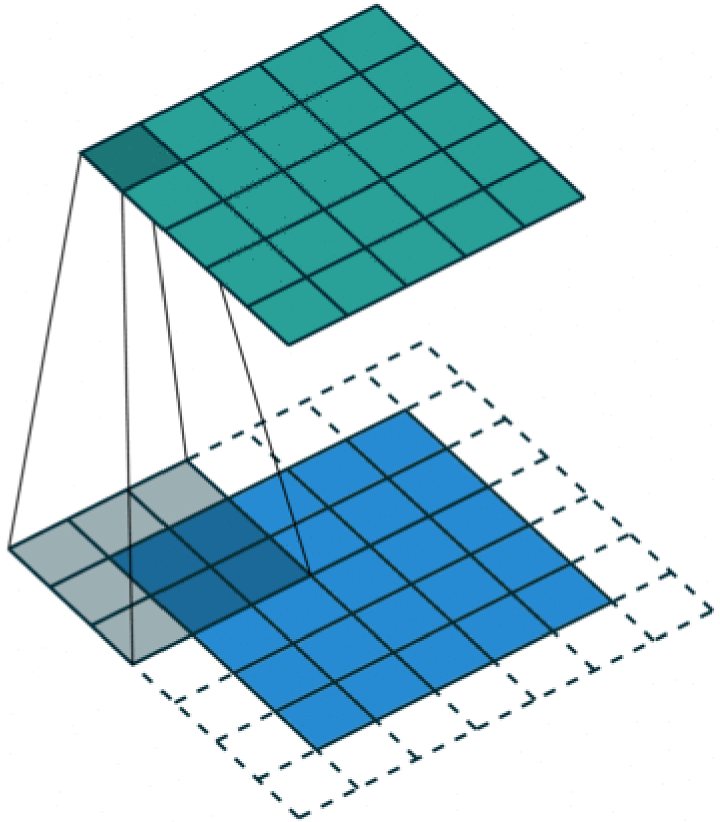
Local connections

Every pixel in the next level depends on a small area in the input.

Weight sharing

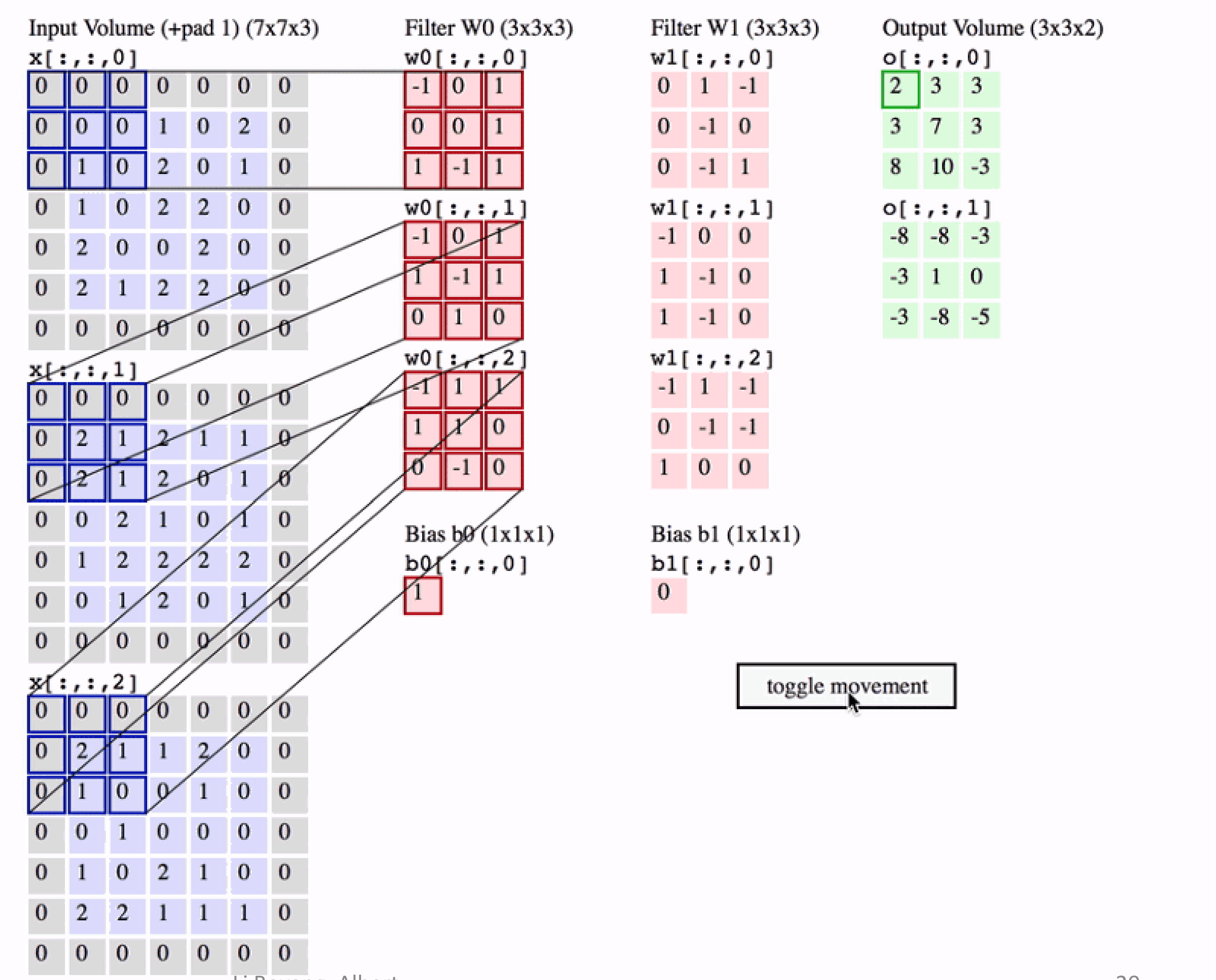
The filter weights are shared by all locations

2D Convolution

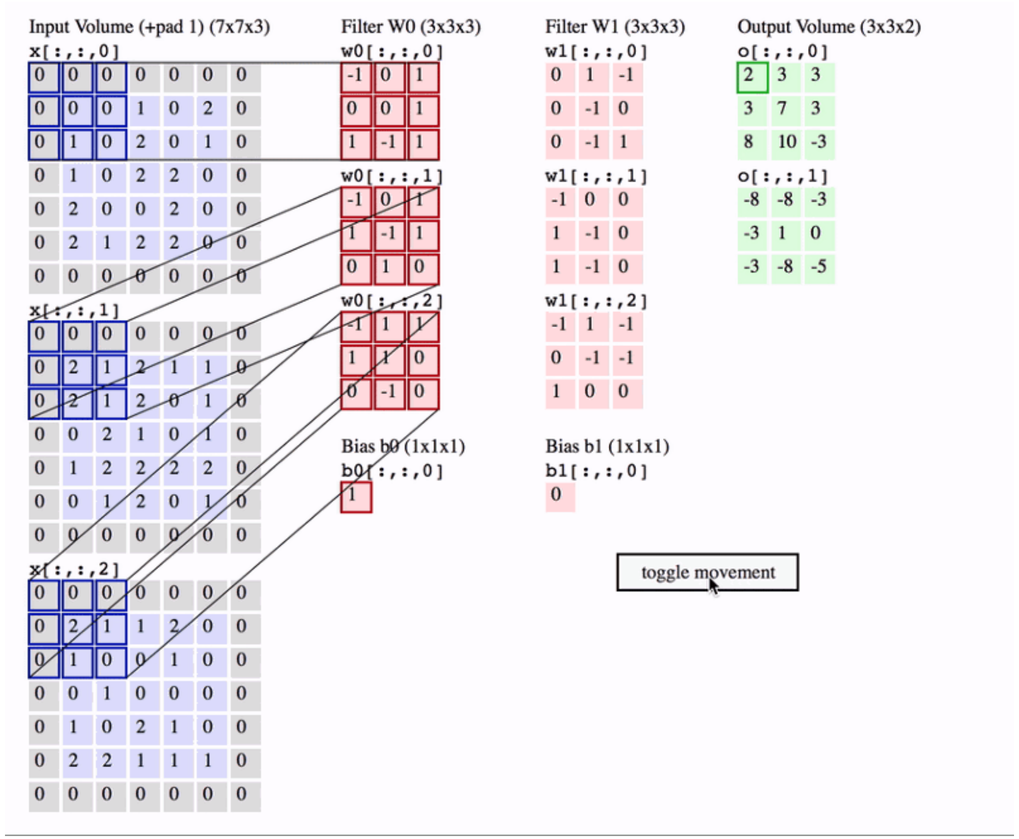


- Use zero padding around the input to maintain the size of the feature maps.

2D Convolution with Multiple Channels

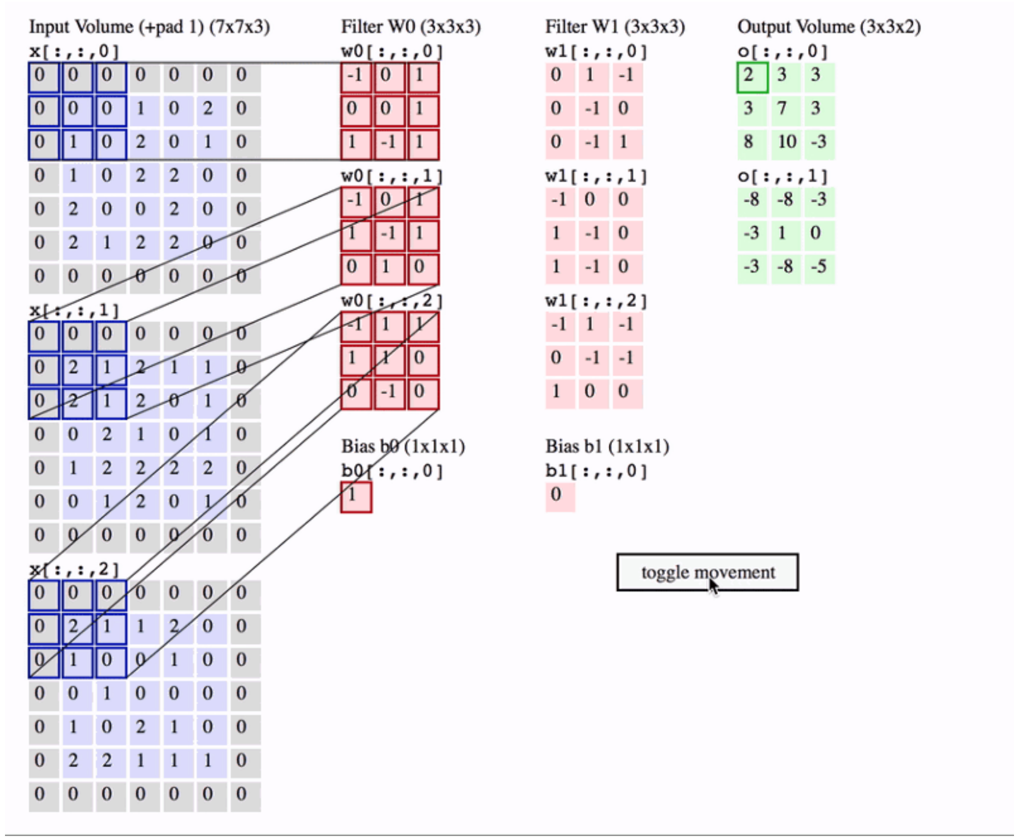


2D Convolution



- Input size : $B \times H \times W \times C$ (order 4 tensor)
- B: batch size (will discuss with SGD)
  - The number of data points in one forward operation
- H: height of the image / feature map
- W: width of the image / feature map
- C: number of channels

2D CNN: Number of Parameters



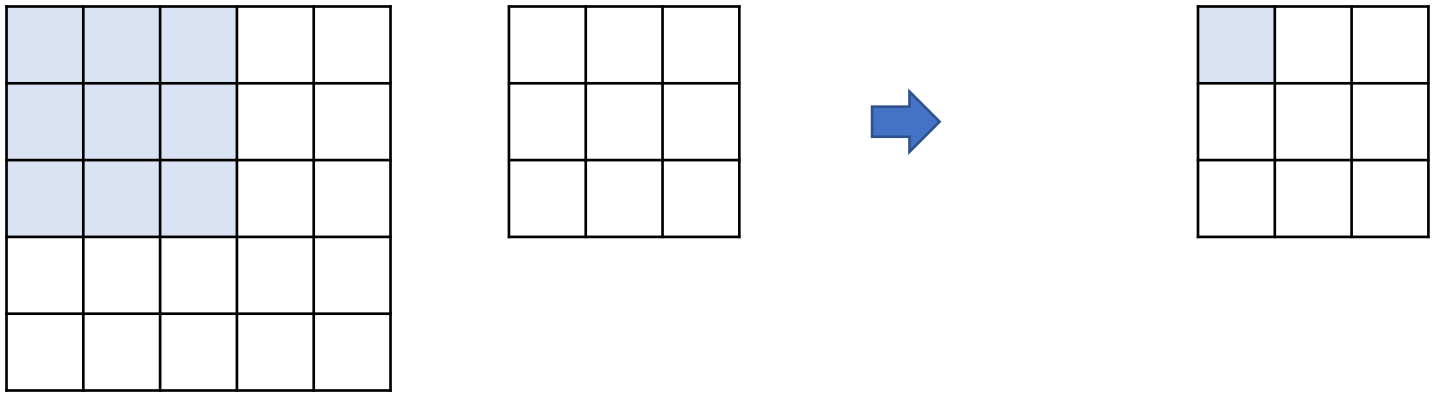
- Kernel / filter size: $K \times K$
  - K is usually an odd number so that there is a central pixel.
- # Input Channels: C_{in}
- # Output Channels = # Kernels / Filters: C_{out}
- Total number of parameters = $K \times K \times C_{\text{in}} \times C_{\text{out}}$

Example

- Kernel / filter size: $K = 3$
- # Input Channels: $C_{\text{in}} = 3$
- # Output Channels = # Filters: $C_{\text{out}} = 64$
- Total number of parameters = $K \times K \times C_{\text{in}} \times C_{\text{out}} = 1728$
- For a 32×32 input image, an MLP will have $(32 \times 32 \times 3) \times (32 \times 32 \times 64) = 201{,}326{,}592$ parameters!

Sizes of Feature Maps w/o Strides or Dilation

Input size: $n \times n$ Filter size: $k \times k$ Output size: $(\mathrm{n}-\mathrm{k}+1) \times (\mathrm{n}-\mathrm{k}+1)$



The number of times we can slide the filter by 1 is $n-k$. Add 1 for the position before sliding.

To make the size constant across layers, use $(k-1)$ rows and $(k-1)$ columns of zero-padding.

Hyperparameters for Convolution: Padding

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	2	3	4	5	0	0
0	0	6	7	8	9	10	0	0
0	0	11	12	13	14	15	0	0
0	0	16	17	18	19	20	0	0
0	0	21	22	23	24	25	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Zero padding
(**<u>the most used</u>**)

13	12	11	12	13	14	15	14	13
8	7	6	7	8	9	10	9	8
3	2	1	2	3	4	5	4	3
8	7	6	7	8	9	10	9	8
13	12	11	12	13	14	15	14	13
18	17	16	17	18	19	20	19	18
23	22	21	22	23	24	25	24	23
17	16	16	17	18	19	20	19	18
13	12	11	12	13	14	15	14	13

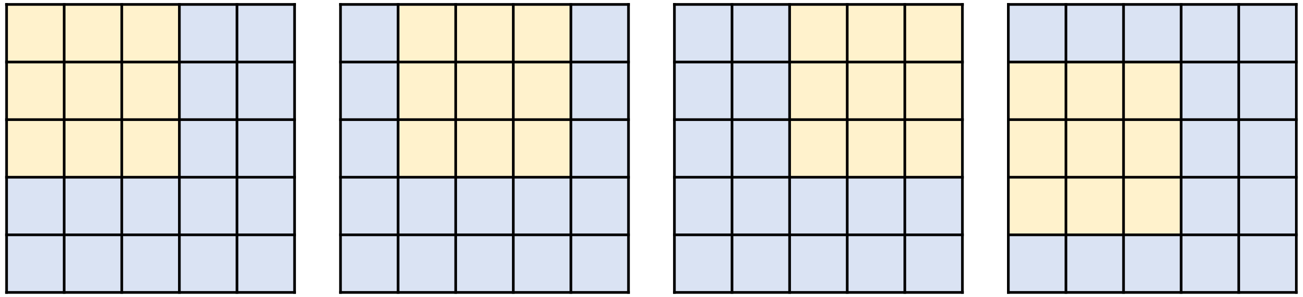
Mirror padding
Mirroring the actual values w.r.t the border

1	1	1	2	3	4	5	5	5
1	1	1	2	3	4	5	5	5
1	1	1	2	3	4	5	5	5
6	6	6	7	8	9	10	10	10
11	11	11	12	13	14	15	15	15
16	16	16	17	18	19	20	20	20
21	21	21	22	23	24	25	25	25
21	21	21	22	23	24	25	25	25
21	21	21	22	23	24	25	25	25

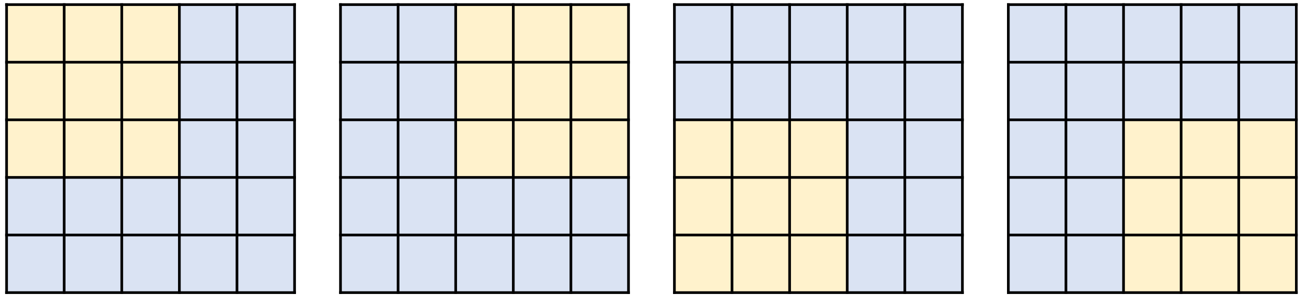
Duplicate padding
Duplicating the closest values on the border

Hyperparameters for Convolution: Stride

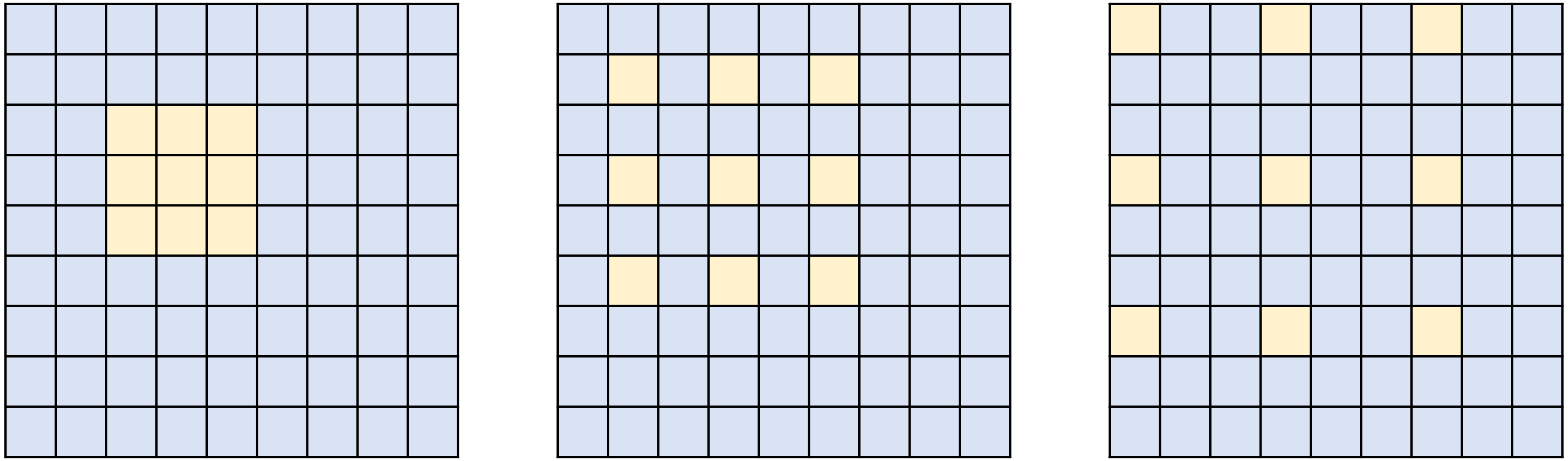
Stride=1
Move 1 pixel at one time



Stride=2
Move 2 pixel at one time
Outputs a smaller feature map



Hyperparameters for Convolution: Dilation

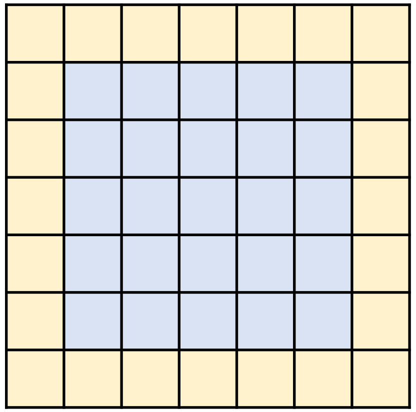


Dilation=1
Utilizing adjacent input pixels

Dilation=2
Utilizing input pixels that are 1 pixel apart (covering 5x5 areas)

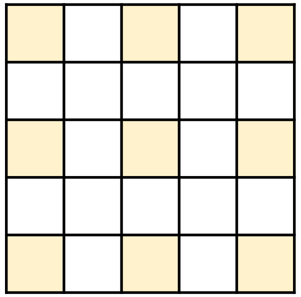
Dilation=3
Utilizing input pixels that are 2 pixel apart (covering 7x7 areas)

Sizes of Feature Maps



padding = 1

Effective input size:
$(\mathrm{n} + \text{padding} \times 2)^2$



dilation = 2

Effective kernel size:
$(\text{dilation} \times (k - 1) + 1)^2$

Shape:

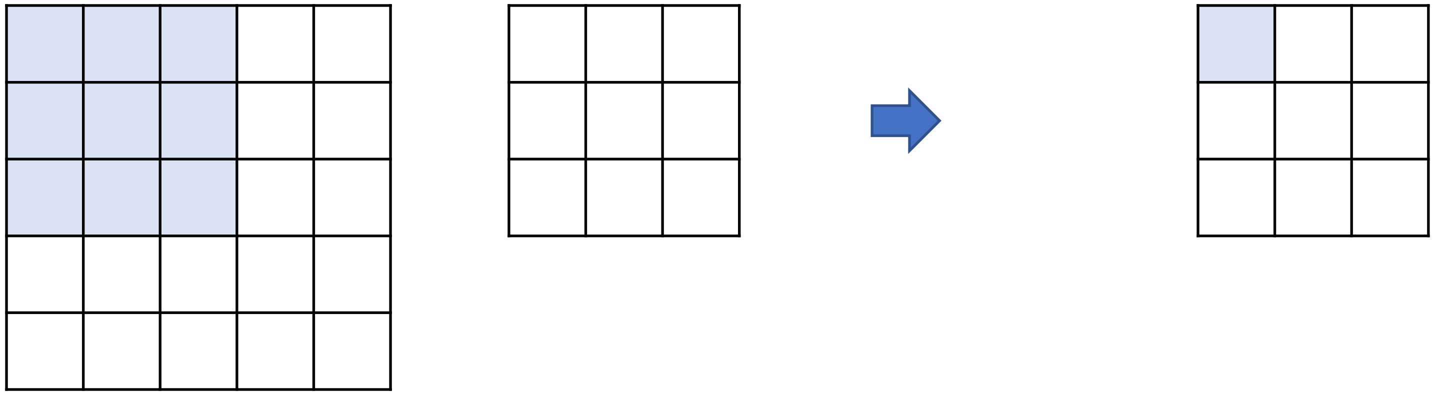
- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Perceptive Field

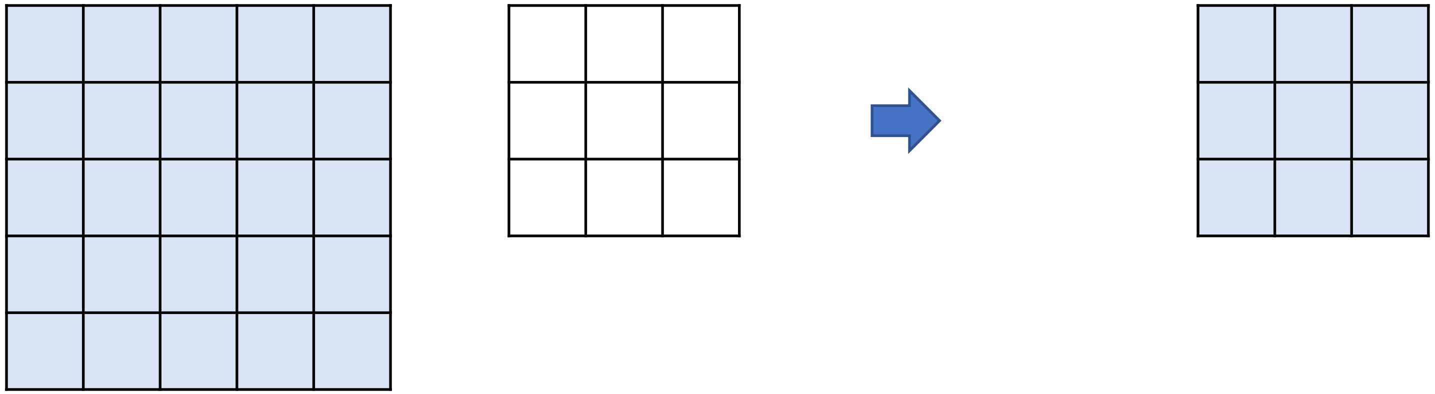
Input size: $n \times n$ Filter size: $k \times k$ Output size: $(\mathrm{n} - \mathrm{k} + 1) \times (\mathrm{n} - \mathrm{k} + 1)$



One pixel in the next layer "sees" $k \times k$ pixels from the previous layer.

Perceptive Field

Input size: $n \times n$ Filter size: $k \times k$ Output size: $k \times k$



$k \times k$ pixels in the next layer "see" $(2k - 1) \times (2k - 1)$ pixels from the previous layer.

$$n - k + 1 = k \qquad n = ?$$

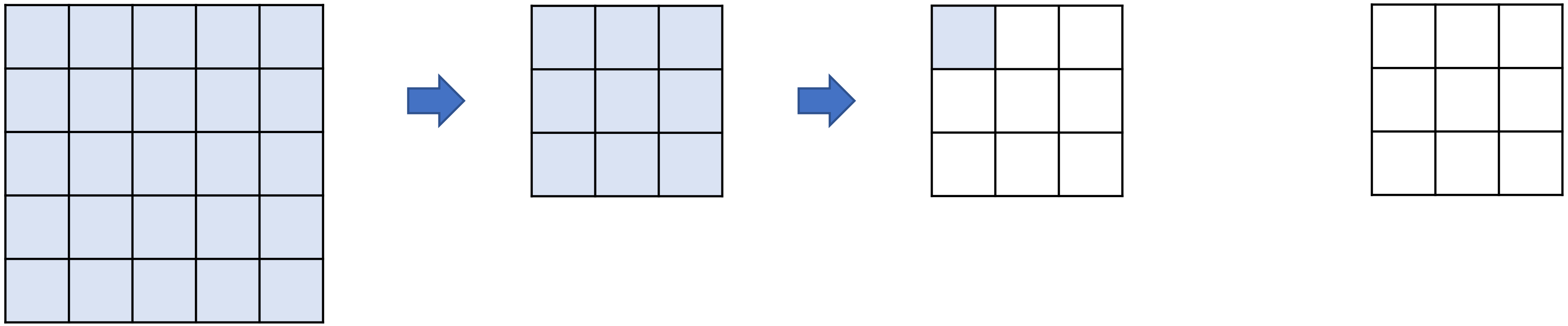
Perceptive Field

Layer l: $(2k-1) \times (2k-1)$

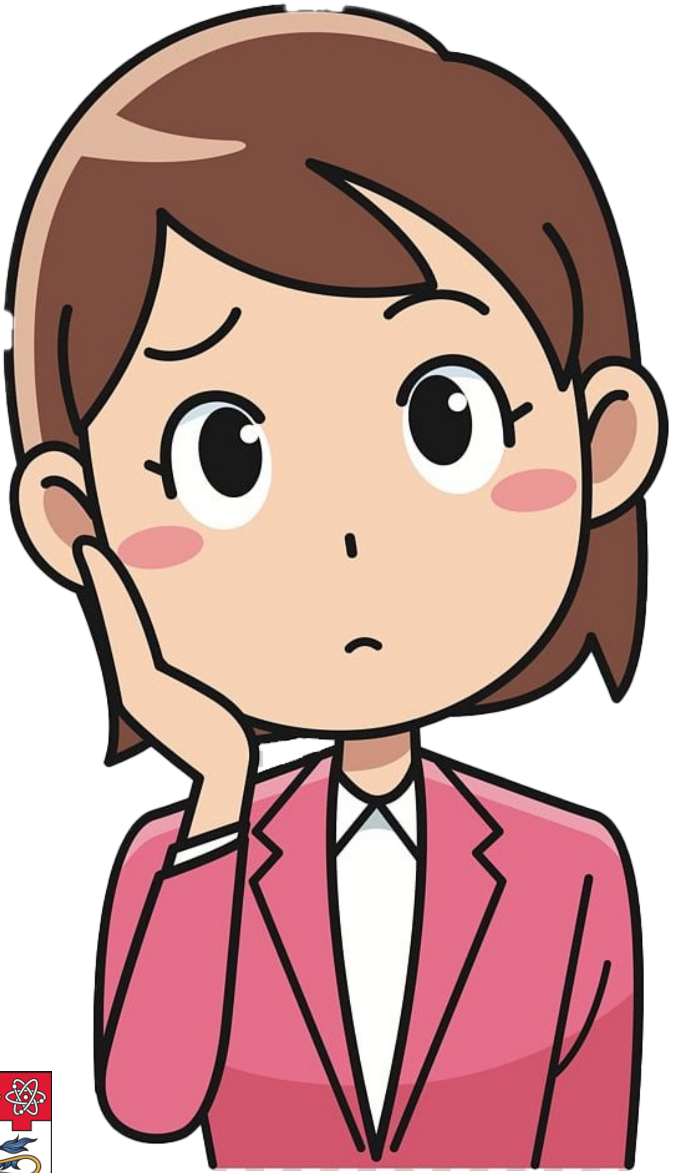
Layer $l+1$: $k \times k$

Layer $l+2$: 1×1

Filter size: $k \times k$



Putting everything together, we can see that a pixel at Layer 2 "sees" information from $(2k-1) \times (2k-1)$ pixels from the raw image. This is known as the <u>perceptive field</u> corresponding to the pixel.



What about three convolution layers?

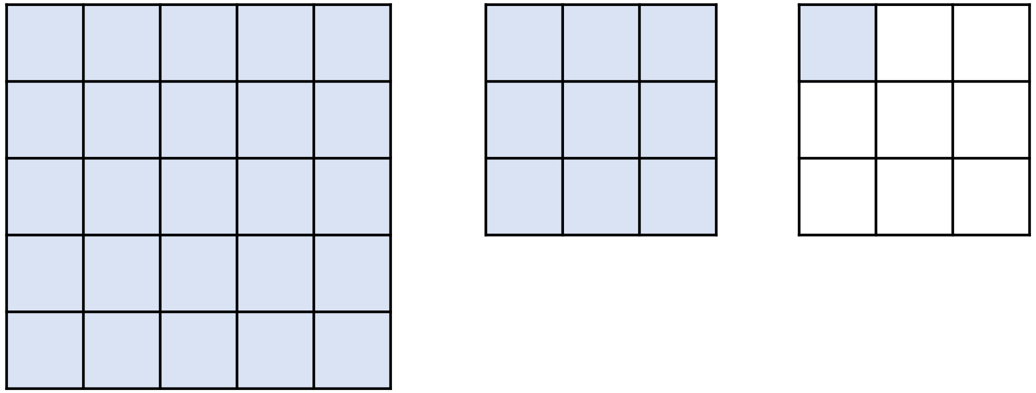
Layer $l-1$

?

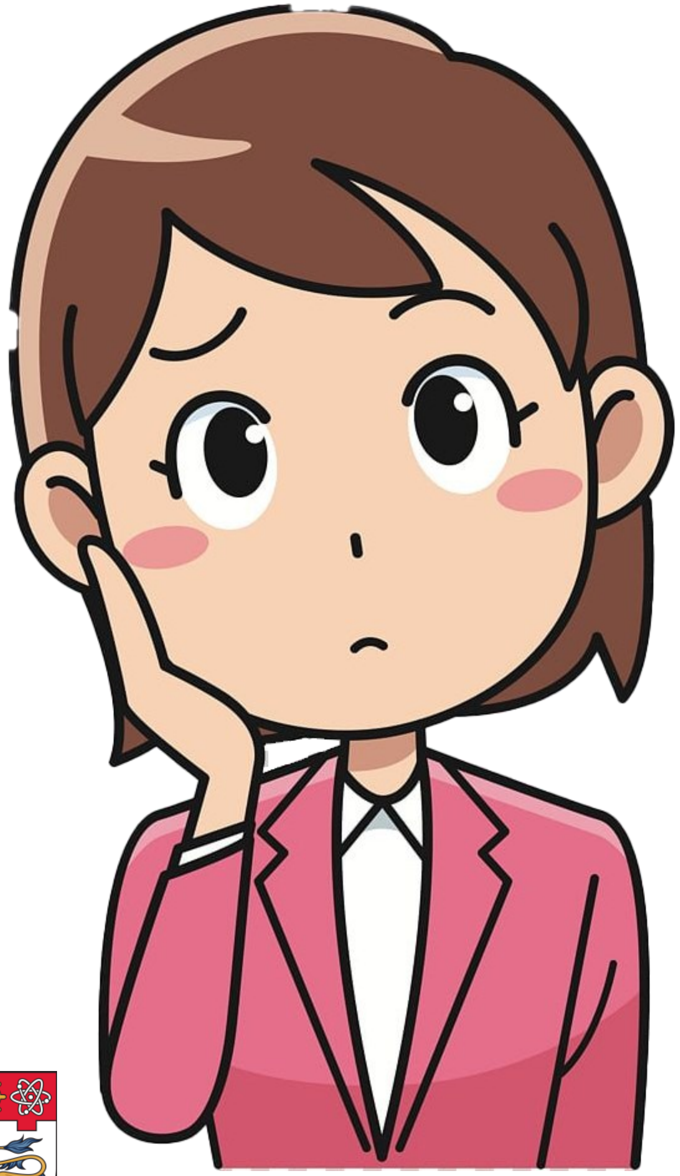
Layer l:
$(2k-1)\times(2k-1)$

Layer $l+1$:
$k\times k$

Layer $l+2$:
1×1



https://www.wooclap.com/SZEINA

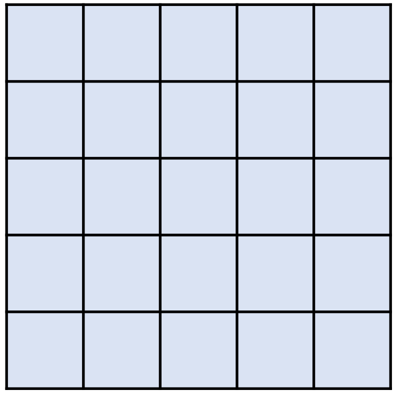


What about three convolution layers?

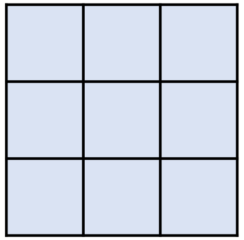
Layer $l-1$

?

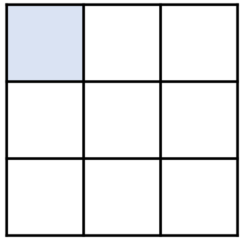
Layer l:
$(2k-1) \times (2k-1)$



Layer $l+1$:
$k \times k$



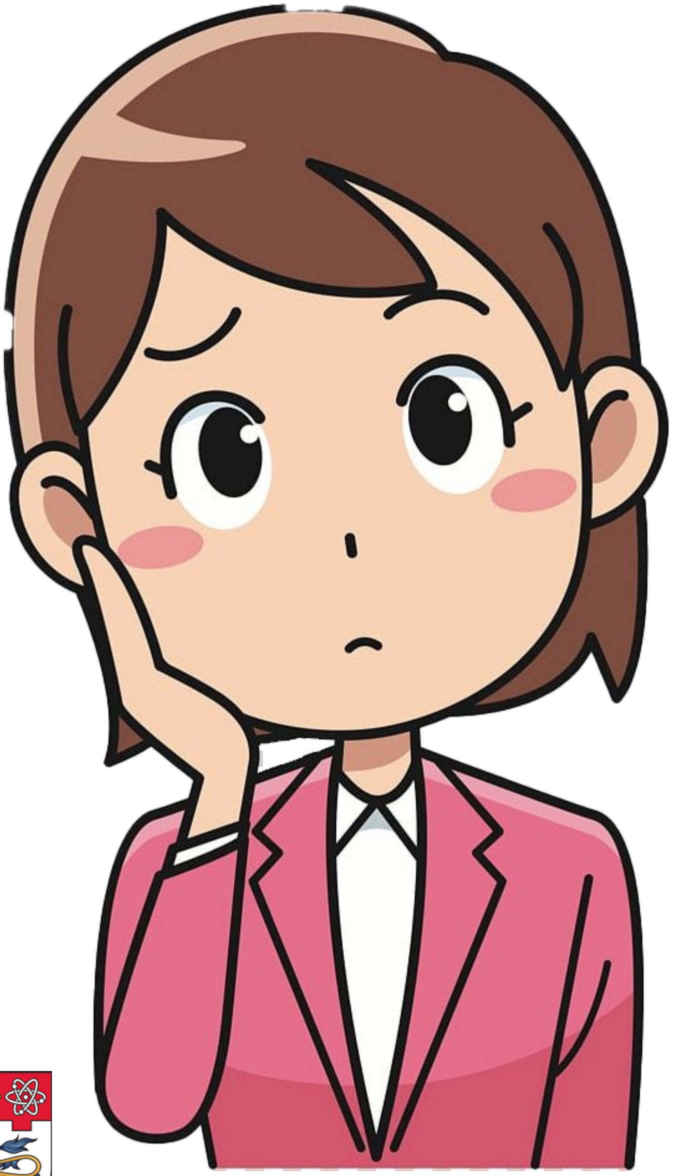
Layer $l+2$:
1×1



Going from layer $l+1$ to layer l

$$n - k + 1 = k$$

$$n = ?$$

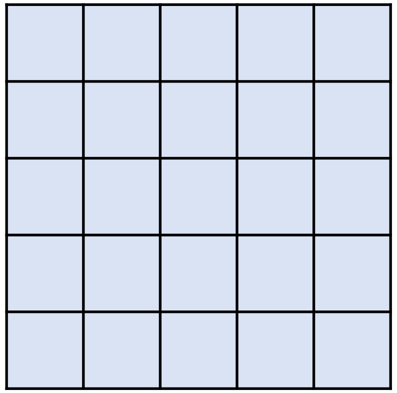


What about three convolution layers?

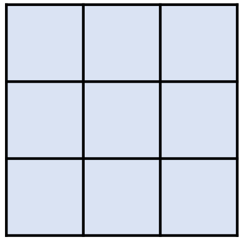
Layer $l-1$

?

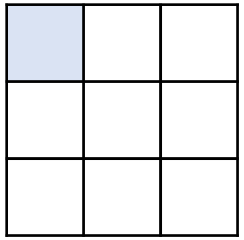
Layer l:
$(2k-1) \times (2k-1)$



Layer $l+1$:
$k \times k$



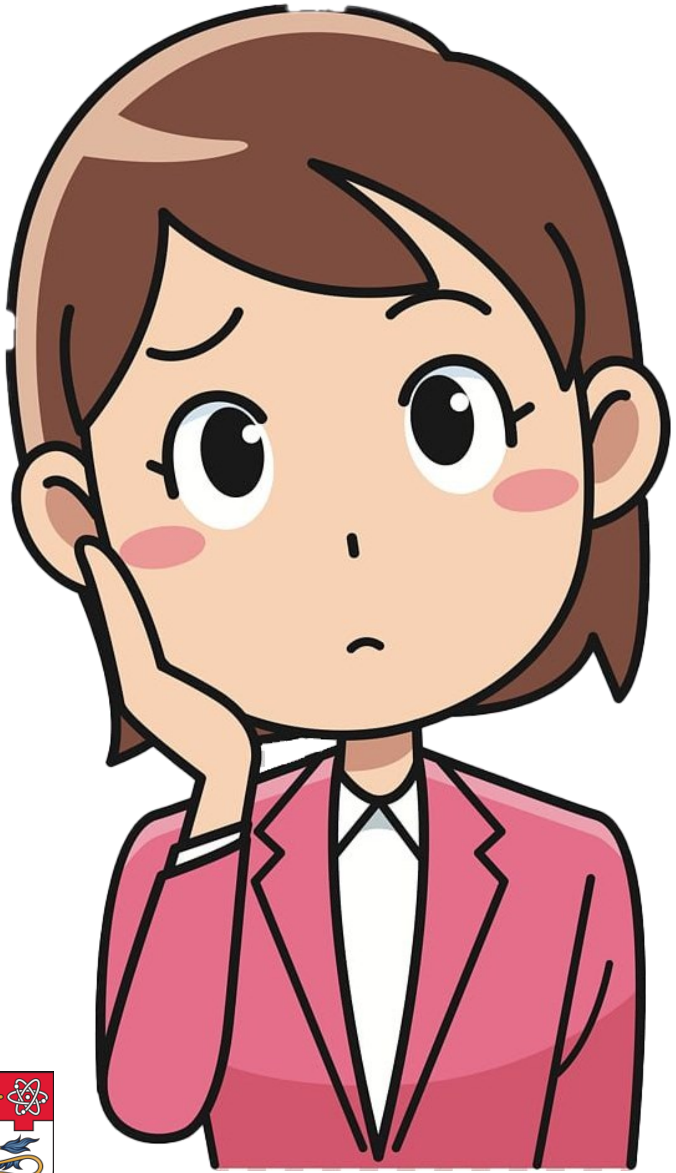
Layer $l+2$:
1×1



Going from layer l to layer $l-1$

$$n - k + 1 = 2\mathrm{k} - 1$$

$$n = 3k - 2$$



Perceptive field when dilation and stride are not 1?

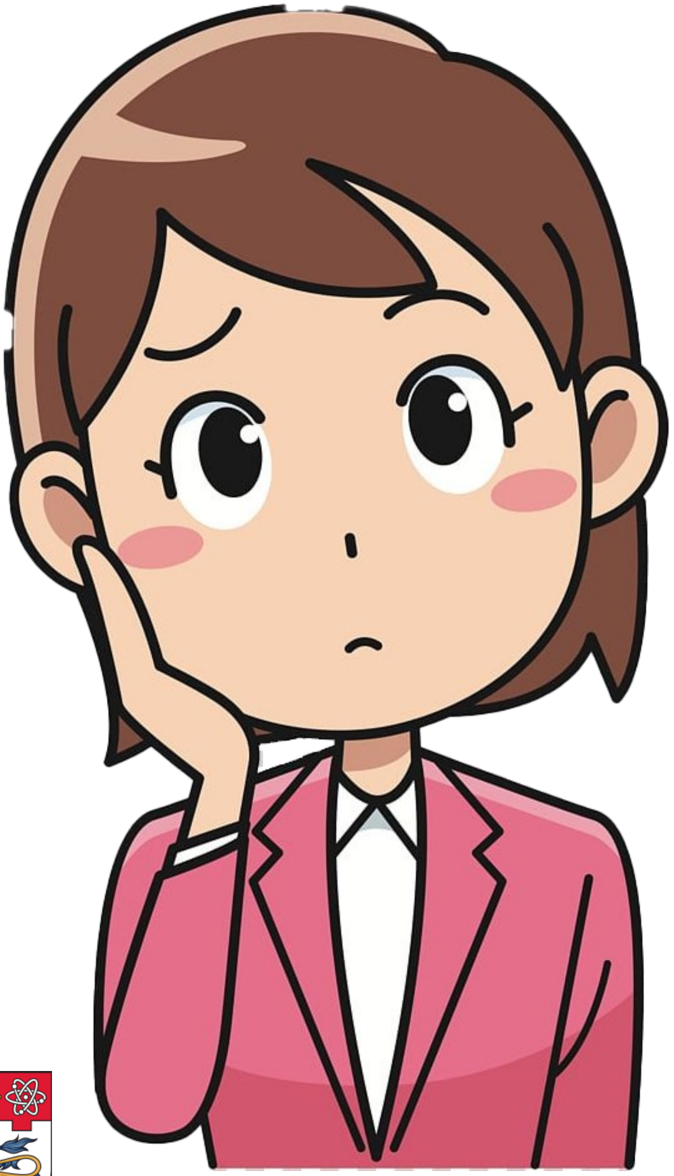
Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Perceptive field when dilation and stride are not 1?



Shape:

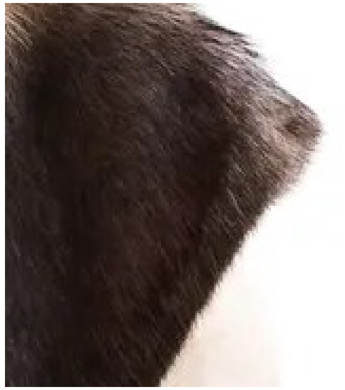
- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Use the above equation to figure out how many pixels from Layer l are needed to produce $\mathrm{k} \times k$ pixels at Layer $l + 1$

Receptive Field and Resolution



High resolution image
Small perceptive field

Cannot recognize image

Low resolution image
Small perceptive field

Suitable for image recognition

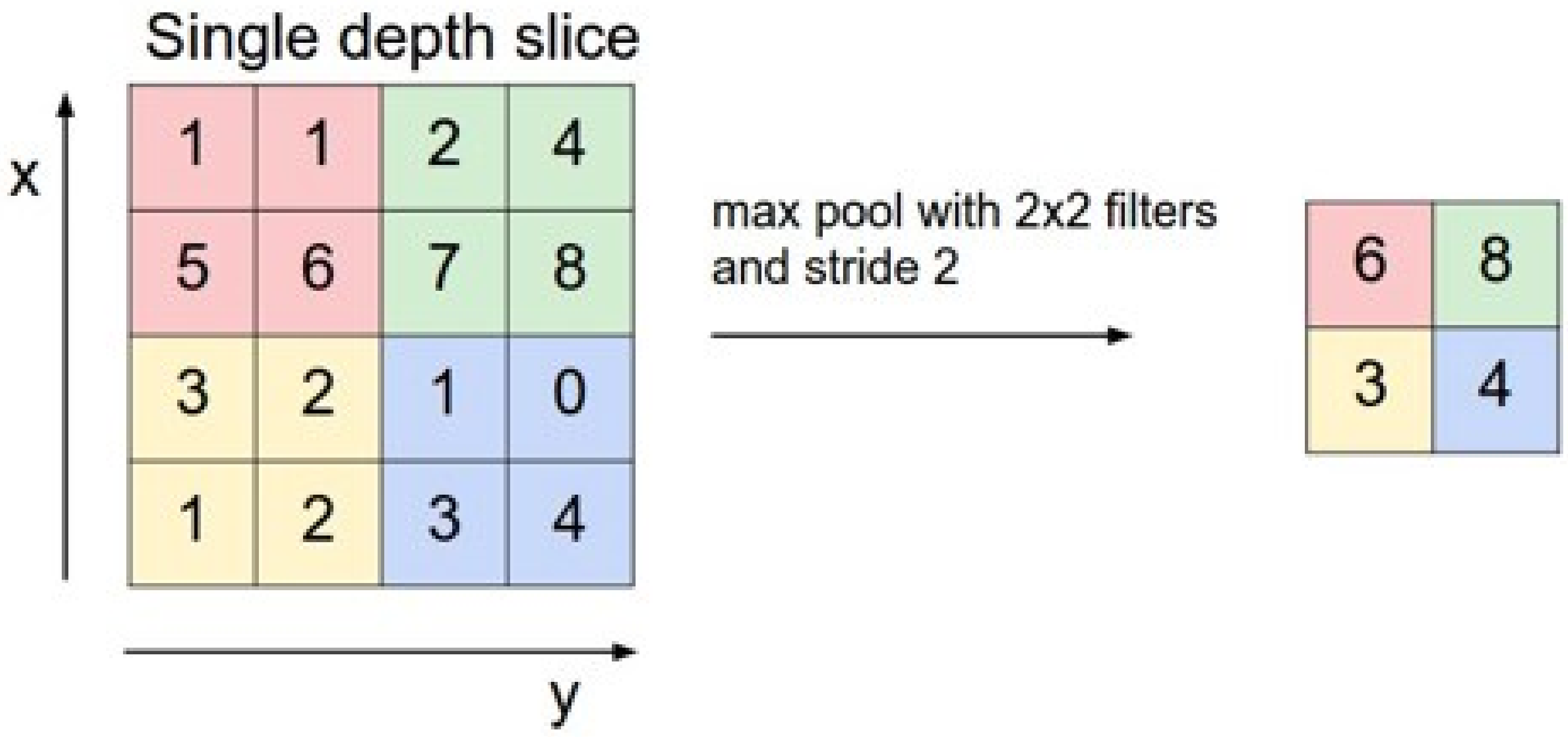
High resolution image
Large perceptive field

High computational cost

Receptive Field and Resolution

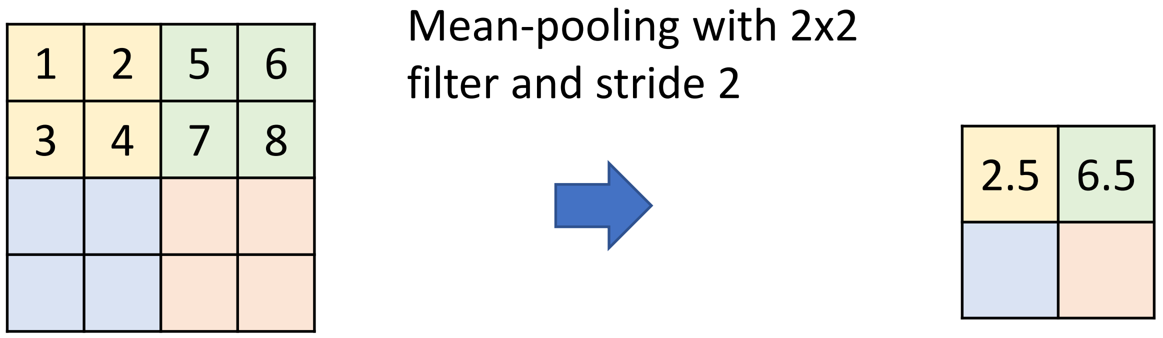
- In neural networks for image recognition, a common practice is to gradually lower the resolution of the feature maps.
- The input image is of relatively high resolution (e.g., 224 x 224 for ImageNet)
- Throughout the network, we apply a few subsampling operations.

Downsampling: Max-pooling



The filter size and the stride are usually equal.

Downsampling: Mean-pooling



For pooling operations, the filter size and the stride are usually (but not always) equal.

Downsampling: Convolution with Stride > 1

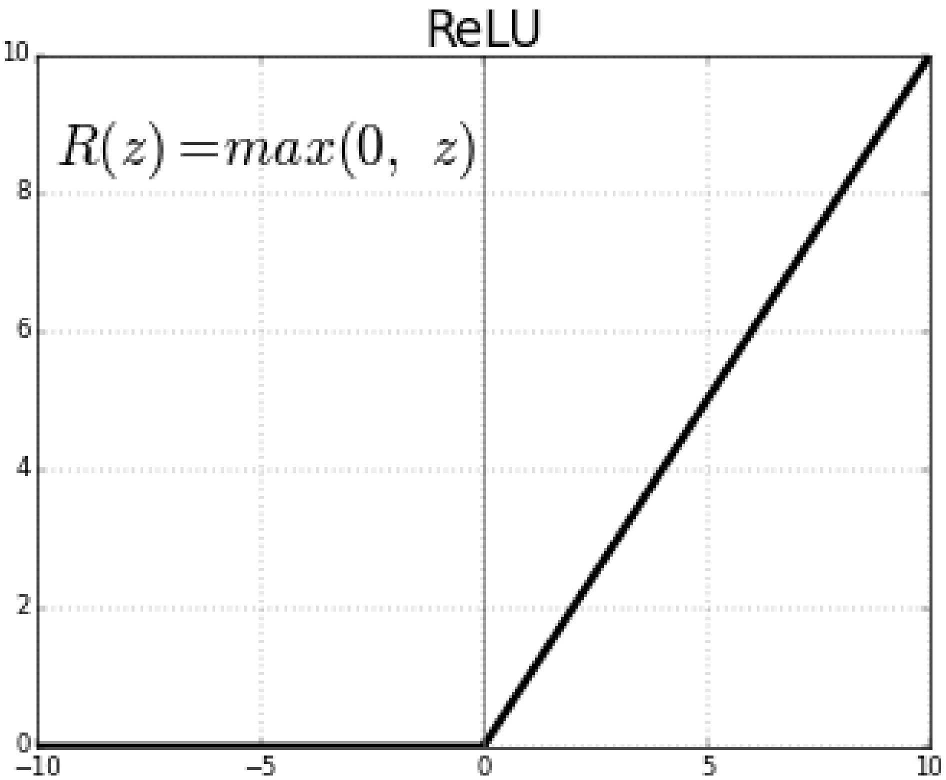
Shape:

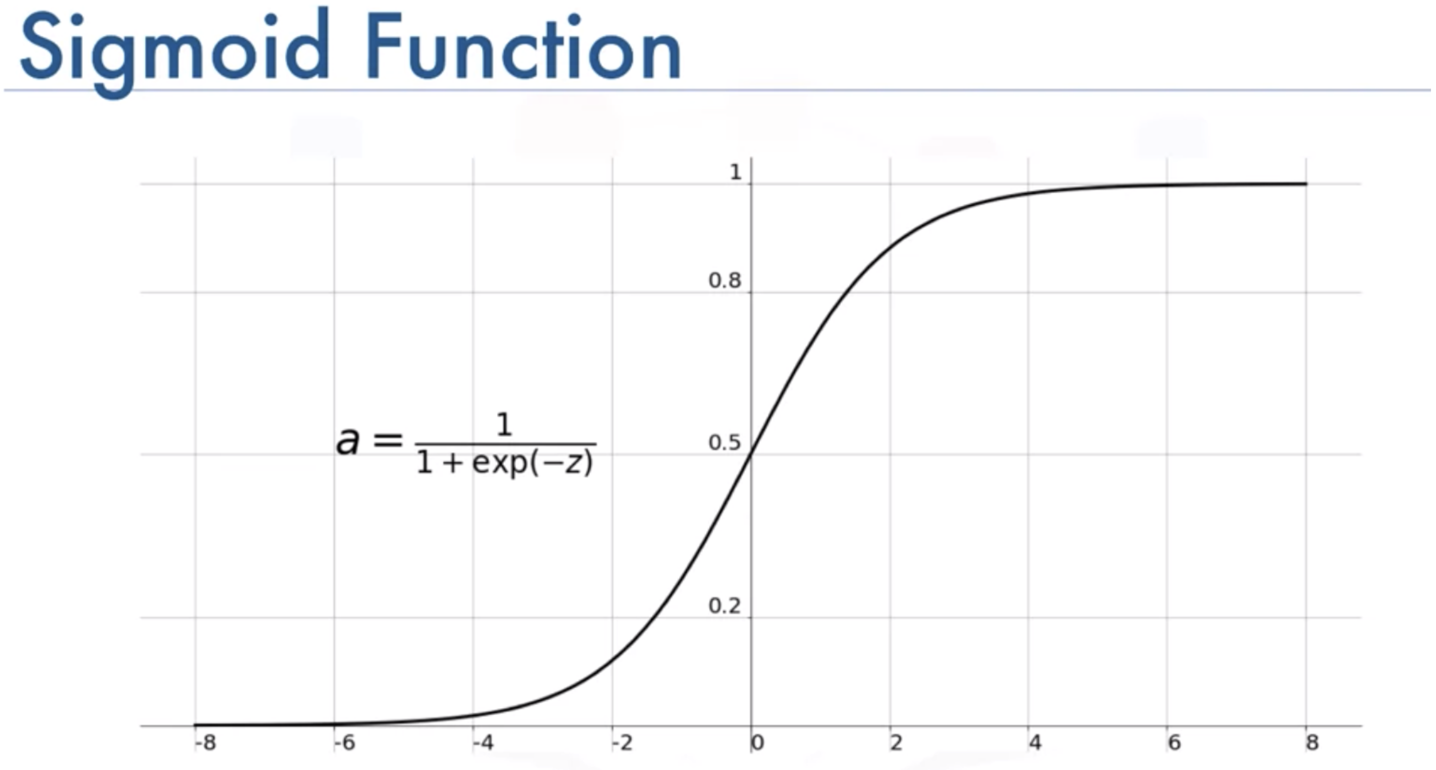
- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\boxed{\text{stride}[0]}} + 1 \right\rfloor$$

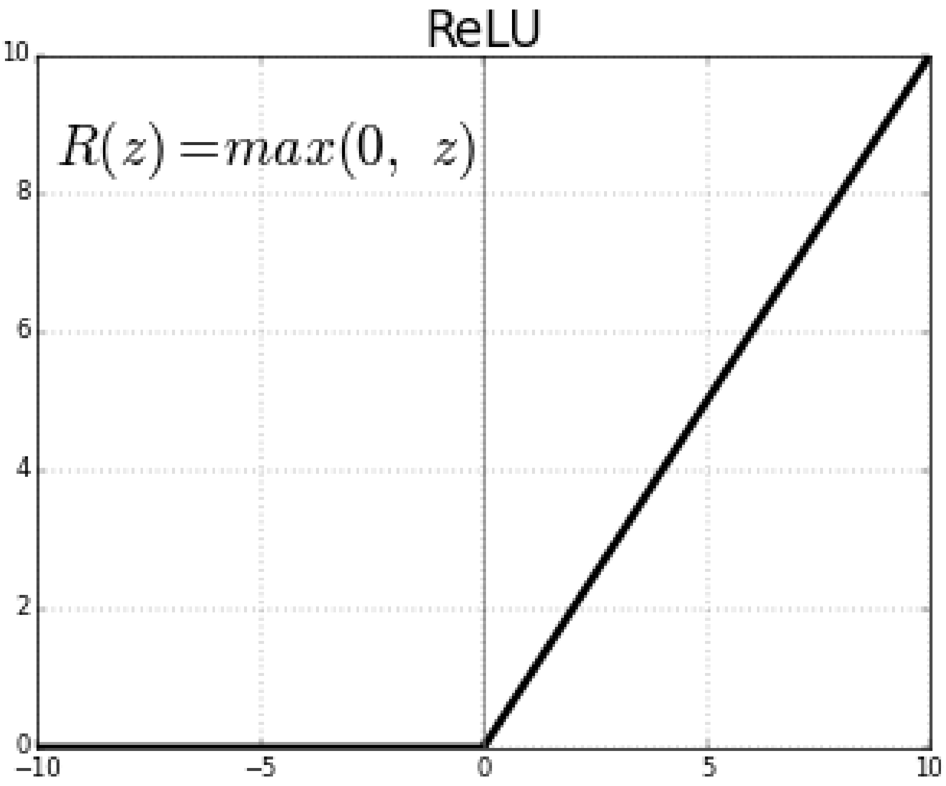
$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\boxed{\text{stride}[1]}} + 1 \right\rfloor$$

The ReLU Activation Function



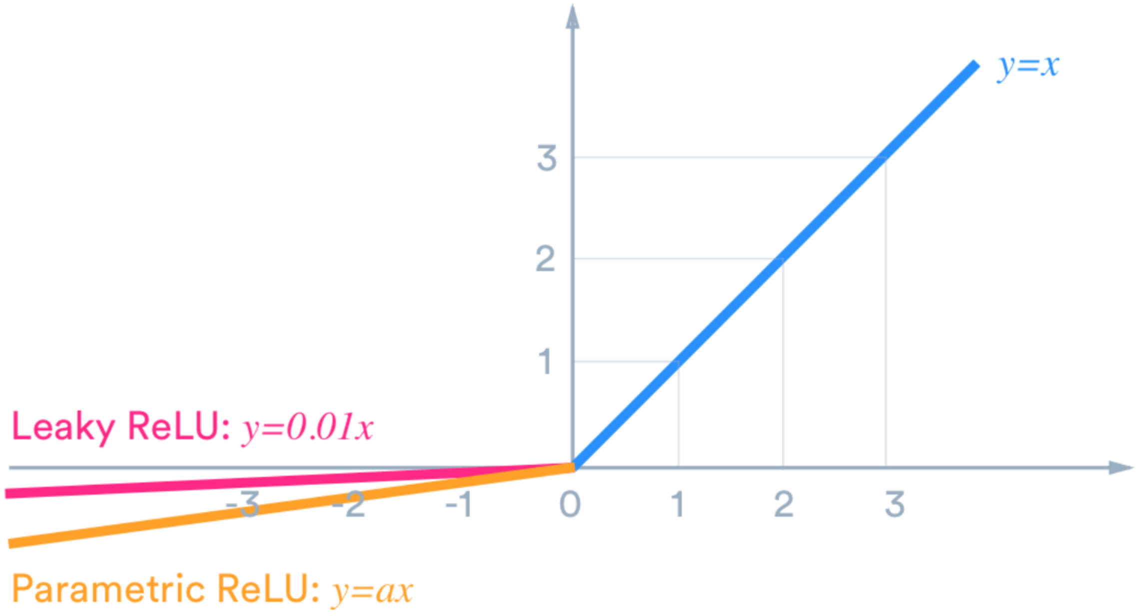


The ReLU Activation Function



- $f(z) = \max(0, z)$
- $= \begin{cases} 0, \text{if } x < 0 \\ x, \text{if } x \geq 0 \end{cases}$
- Benefits:
  - Non-linear
  - Creates many zero elements (i.e., sparse activations)
  - Good gradients when z > 0

Leaky / Parametric ReLU



- ReLU has zero gradient when $z < 0$
- Once that happens, z will never receive any update
- Hence, a "leaky" version:
- $f(z) = \begin{cases} az, \text{if } z < 0 \\ z, \text{if } z \geq 0 \end{cases}$
- $0 < a < 1$

Leaky / Parametric ReLU

CLASS `torch.nn.LeakyReLU(negative_slope=0.01, inplace=False)` [SOURCE]

Applies the element-wise function:

$$\text{LeakyReLU}(x) = \max(0, x) + \text{negative\_slope} * \min(0, x)$$

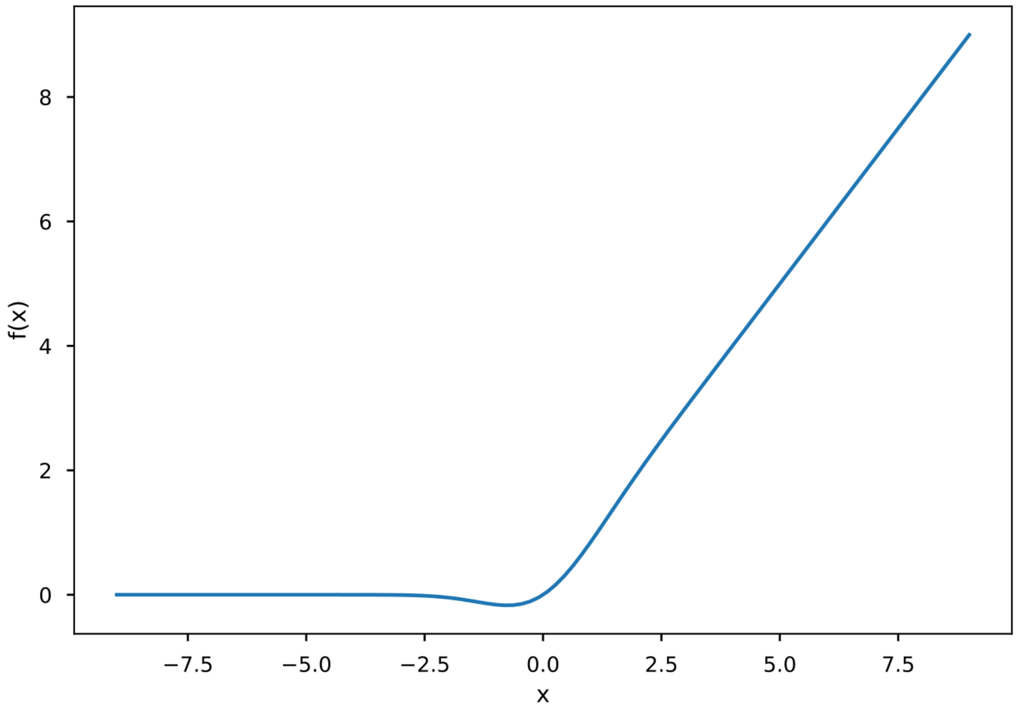
or

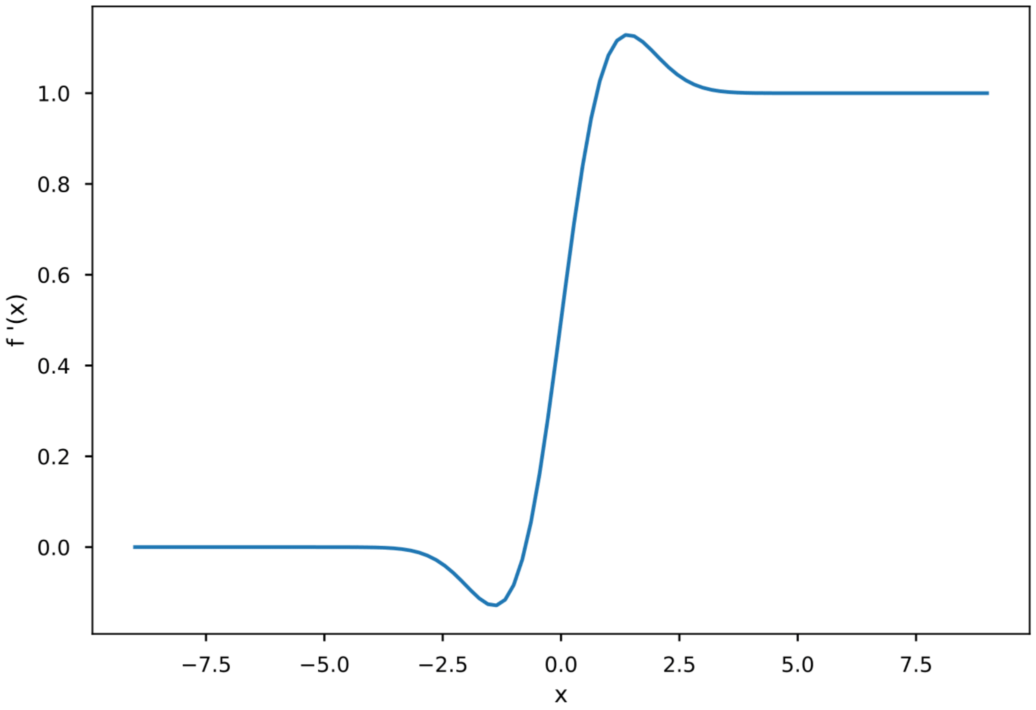
$$\text{LeakyRELU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \text{negative\_slope} \times x, & \text{otherwise} \end{cases}$$

GeLU

Intuitively, GeLU provides a “second chance” before one heads into zero-gradient territory.

GELU function and its Derivative





Input Normalization / Whitening

- We usually normalize the input images by channel
- So that each channel is expected to have zero mean and unit standard deviation.
- Balances the contribution of each channel to the output
- Often improves training

- Consider the simple model $y = \beta_1 x_1 + \beta_2 x_2$
- If β_1 and β_2 are of similar magnitudes, and $x_1 \ll x_2$, $\beta_2 x_2$ will dominate.
- To balance things, we can let $\beta_1 \gg \beta_2$, but that can lead to optimization difficulties.
    - β_1 and β_2 would require different learning rates for proper gradient-based updates.

Input Normalization / Whitening

- We usually normalize the input images by channel
- So that each channel is expected to have zero mean and unit standard deviation.
- Balances the contribution of each channel to the output
- Often improves training

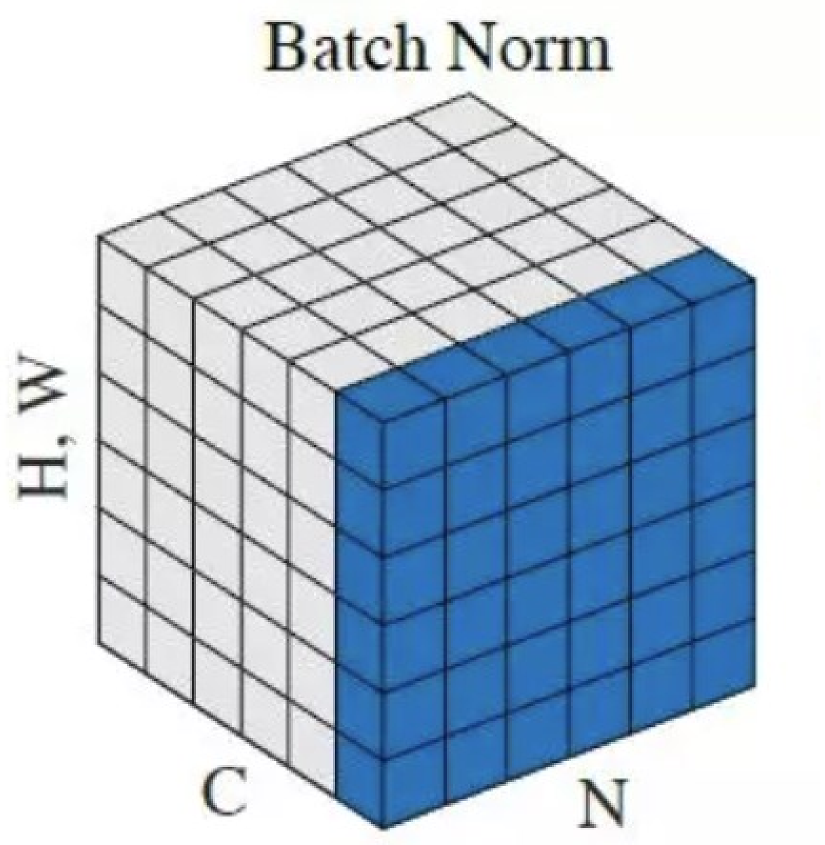
- Procedure: compute dataset-wide channel mean μ_R, μ_G, μ_B and stdev $\sigma_R, \sigma_G, \sigma_B$
- For input $X = [\boldsymbol{x}_R, \boldsymbol{x}_G, \boldsymbol{x}_B]$, compute

$$\tilde{X} = \left[\frac{\boldsymbol{x}_R - \mu_R}{\sigma_R}, \frac{\boldsymbol{x}_G - \mu_G}{\sigma_G}, \frac{\boldsymbol{x}_B - \mu_B}{\sigma_B}\right]$$

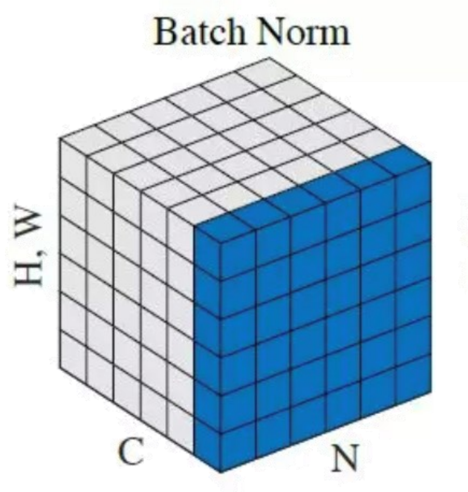
- Use $\tilde{X}$ in place of X

Batch Normalization

- In deep neural networks, the internal features lack any normalization.
- Batch normalization addresses this.
- We use the mini-batch statistics in place of the whole-dataset statistics.
- The feature map has size $B \times H \times W \times C$
- From $B \times H \times W$ elements, we compute batch-wide channel mean μ_c and standard deviation σ_c.



Batch Normalization



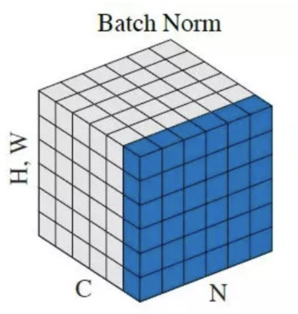
- Given: batch-wide channel mean μ_c and standard deviation σ_c.
- For feature map $X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_C]$, compute per channel statistics μ_i, σ_i

$$\widetilde{\boldsymbol{x}}_i = \gamma_i \frac{\boldsymbol{x}_i - \mu_i}{\sigma_i + \epsilon} + \beta_i$$

- ϵ is a small constant ($\approx 10^{-6}$) to avoid division by zero.

- Use $\tilde{X} = [\widetilde{\boldsymbol{x}}_1, \dots, \widetilde{\boldsymbol{x}}_C]$ in place of X
- γ_i and β_i are learnable scalar parameters
  - One pair per channel

Batch Normalization



- Replacing whole-dataset statistics with the mini-batch statistics requires the batch size to be reasonable large.
- **Why?**
- Recall from the probability theory section that the variance of the mean estimator $\hat{\mu}_{\text{MLE}}$ is $\frac{Var(X)}{N}$.
- N is the number of elements participating in the mean calculation $H \times W \times B$.
- When $N = H \times W \times B$ is too small, the mean estimator is unreliable.
- Similar conclusion for $\hat{\sigma}$

Batch Normalization: Inference

- Dataset-wise statistics are good but too expensive
- Training utilizes the batch-wise mean and standard deviation.
- During inference (i.e., testing), we keep a running mean and a running variance.
  - Running mean: $\mu = \alpha\mu + (1-\alpha)\mu_{\text{batch}}$
  - Running variance: $\sigma^2 = \alpha\sigma^2 + (1-\alpha)\sigma_{\text{batch}}^2$

- In PyTorch, such training/inference behavior change is controlled by `model.eval()` and `model.train()`
- Note: `model.eval()` does not turn off automatic differentiation.
- That is controlled by `torch.no_grad()` and `torch.inference_mode()`
- During training, μ_i and σ_i participate in the gradient calculation.

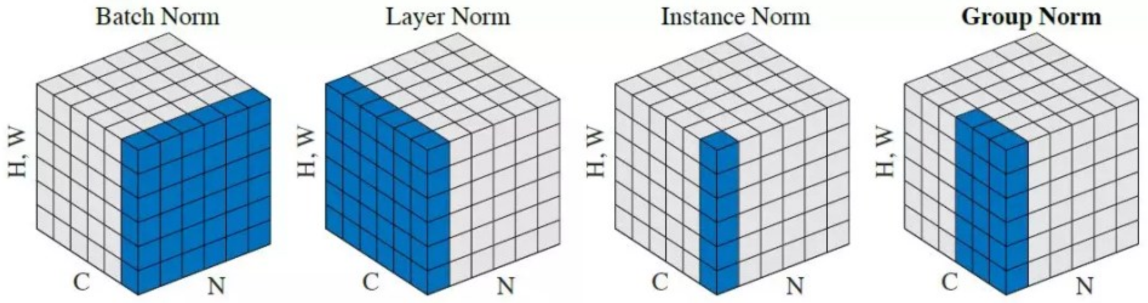
Why the difference?

- At training time, the network parameters are continuously updated.
- After an update, feature distributions will change.
- Thus, old values of μ and σ are unreliable.

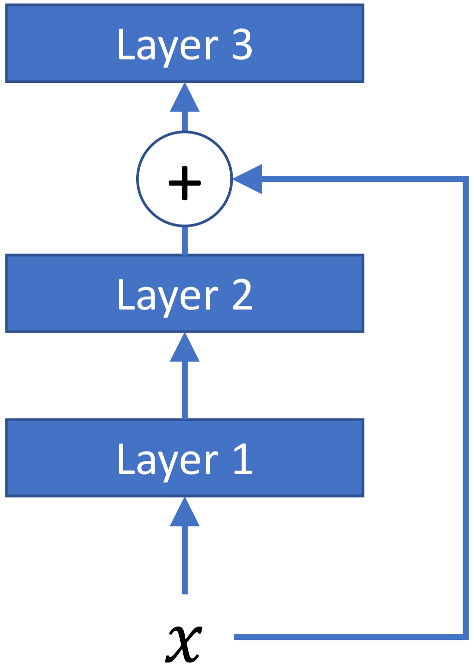
- At inference time, the network parameters are fixed.
- The feature distribution remains the same.

Other Normalization Forms

- To avoid interference from other samples in the batch, we can normalize one sample by itself (Instance Normalization).
- We can normalize several channels together (Group Normalization).
- The Transformer model (covered later) uses Layer Normalization



Skip Connection



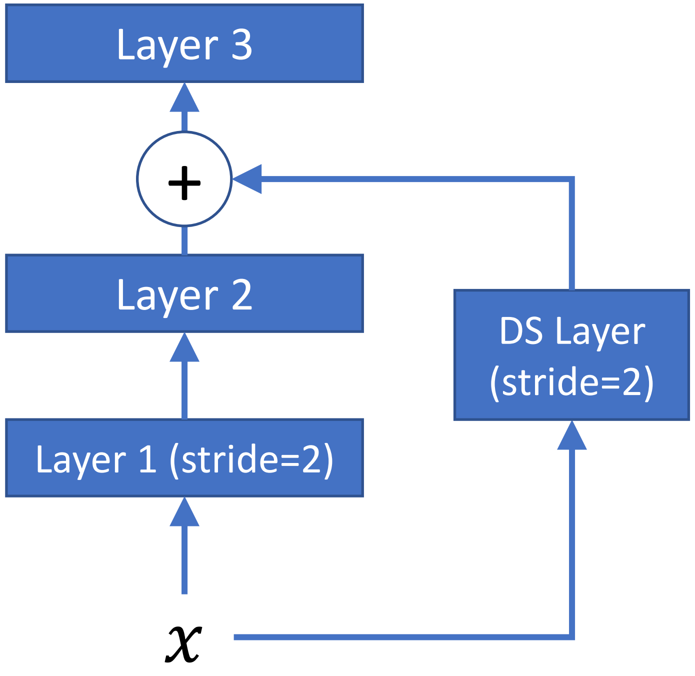
- The first two layers:

$$\alpha\left(f_2\left(\alpha\left(f_1(x)\right)\right)\right)$$

- With the skip connection:

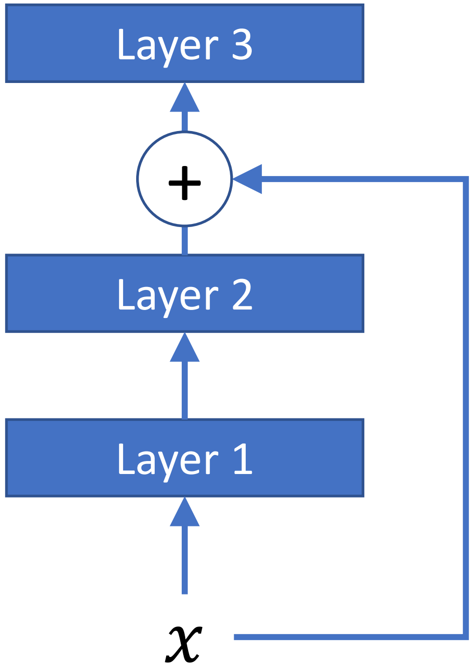
$$\alpha\left(f_2\left(\alpha\left(f_1(x)\right)\right)\right)+x$$

Skip Connection – Feature Map Size



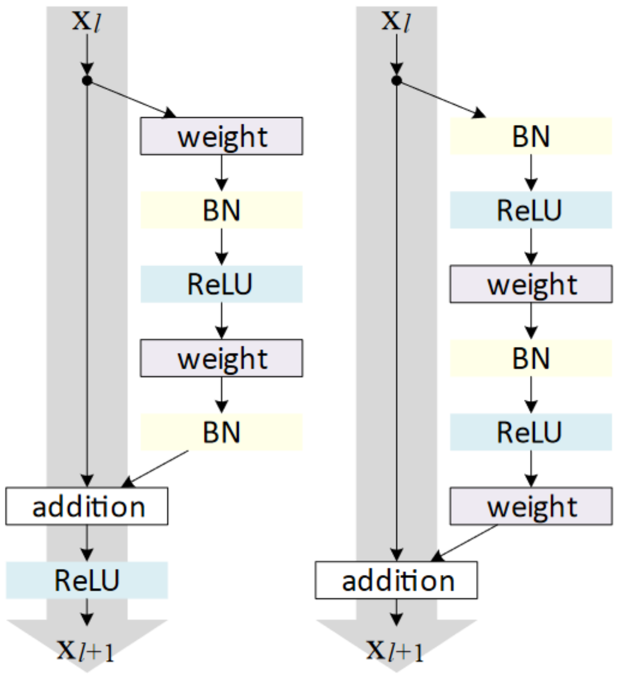
- When Layer 1 performs downsampling, the size of x and $f(x)$ become different.
- As a remedy, we use another downsampling convolution layer on the skip connection.
  - 1×1 kernels with stride = 2
  - Followed by BatchNorm but not activation.

Skip Connection



- Benefits of skip connection
  - Creates another path for gradient backpropagation
  - Allows the network to easily learn identity mappings (why?)

Skip Connection



(a) original (b) proposed

- Left: original setup in [1]
- Right: improved skip connection from [2]
- In the BN+ReLU setup, about 50% of outputs are zero.
- Some network variations reverse the order of ReLU and BatchNorm.

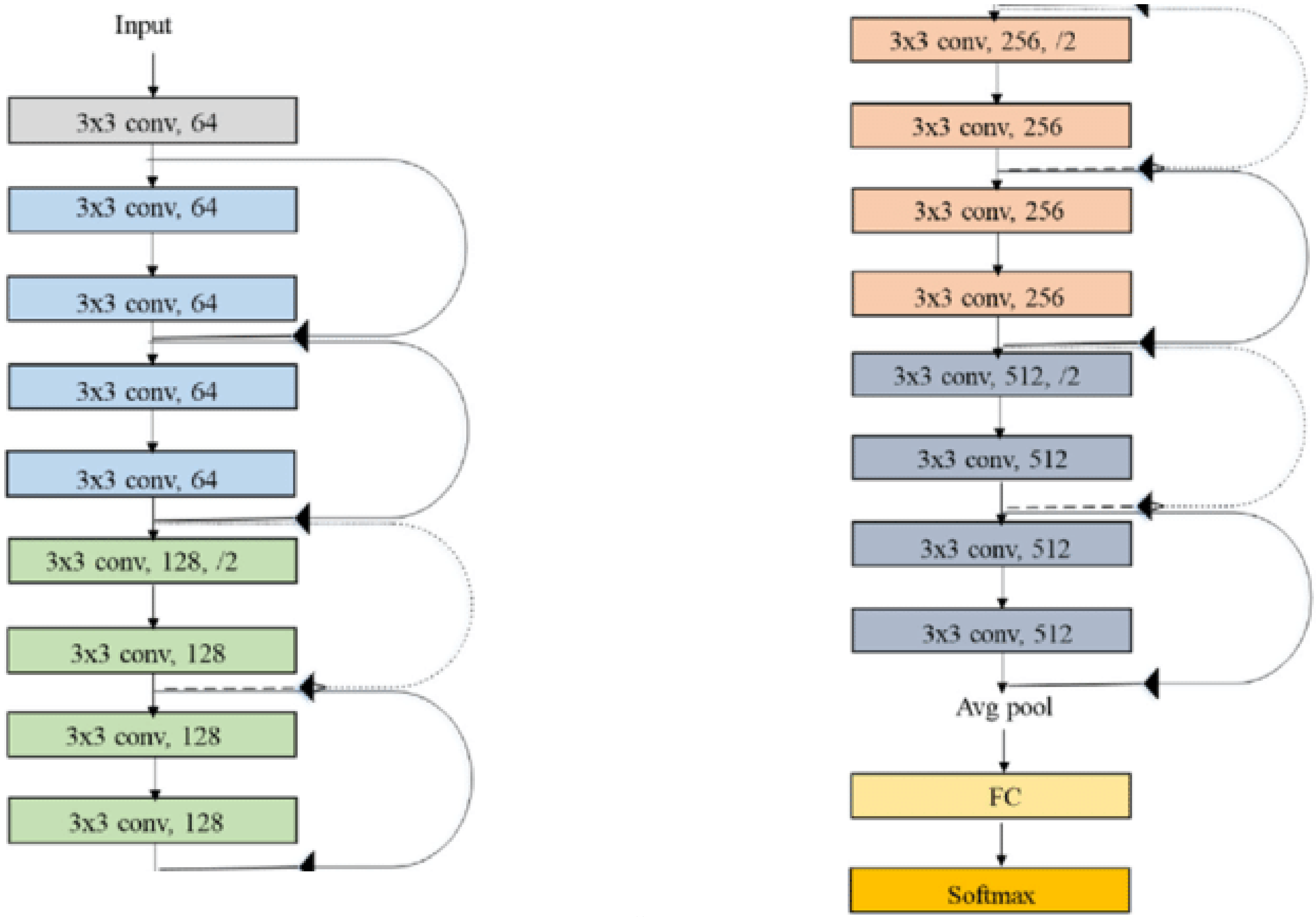
[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. CVPR. 2016.
[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity Mappings in Deep Residual Networks. 2016.

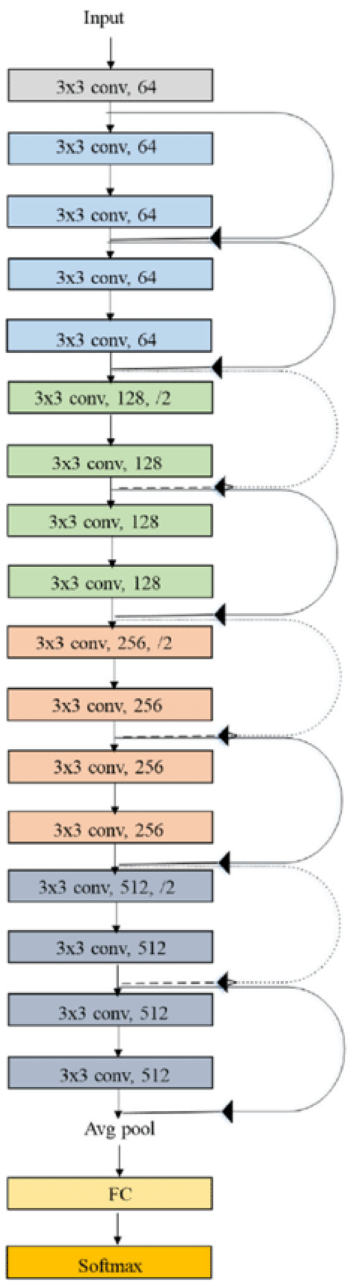
Building Blocks So Far

- Convolution layers
- MLP layers (a.k.a. linear layers, fully connected layers, dense layers)
- Max / Mean Pooling
- ReLU activation
- Residual connection
- Batch normalization

Time to put them together!

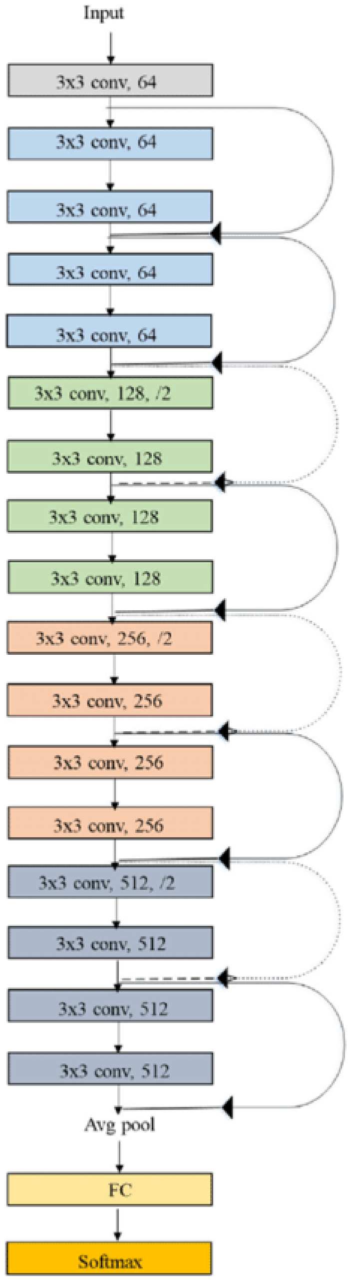
Putting them together: ResNet





Putting them together: ResNet

- 5 groups of convolution layers, each group operating at one resolution
- The first convolution is the "stem". There is only one in this group.
- The rest of groups always begins with a downsampling operation, either from pooling or strided convolution.
- The convolutions end with a global average pooling, which averages across all pixels in each channel.
- The final fully connected layer (an MLP layer) creates the logits.



Putting them together: ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64 3×3, 64] ×2	[3×3, 64 3×3, 64] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3
conv3_x	28×28	[3×3, 128 3×3, 128] ×2	[3×3, 128 3×3, 128] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×8
conv4_x	14×14	[3×3, 256 3×3, 256] ×2	[3×3, 256 3×3, 256] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×23	[1×1, 256 3×3, 256 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512 3×3, 512] ×2	[3×3, 512 3×3, 512] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

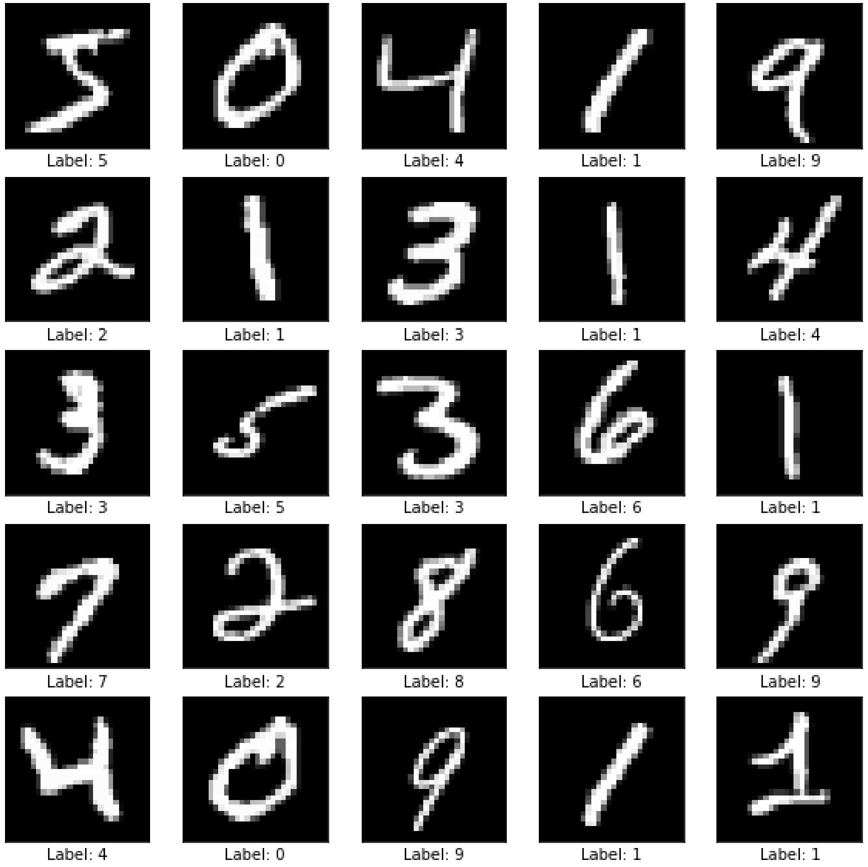
4 stages at different spatial resolution
Conv1 is sometimes called “the stem”

Commonly Use Datasets

- MNIST (LeCun 1998)
- SVHN (Netzer et al. 2011)
- Fashion-MNIST (Xiao et al. 2017)
- CIFAR-10 (Krizhevsky, 2009)
- CIFAR-100 (Krizhevsky, 2009)
- ImageNet-1000 (Jia Deng et al, 2009)

MNIST (LeCun 1998)

- Classification of hand-written digits (from 0 to 9)
- Image size: 28x28
- Training set: 60,000 images
    - 6000 for each digit
    - Needs to be split into training and validation
- Test set of 10,000 examples
    - 1000 for each digit
- Typical performance > 99%



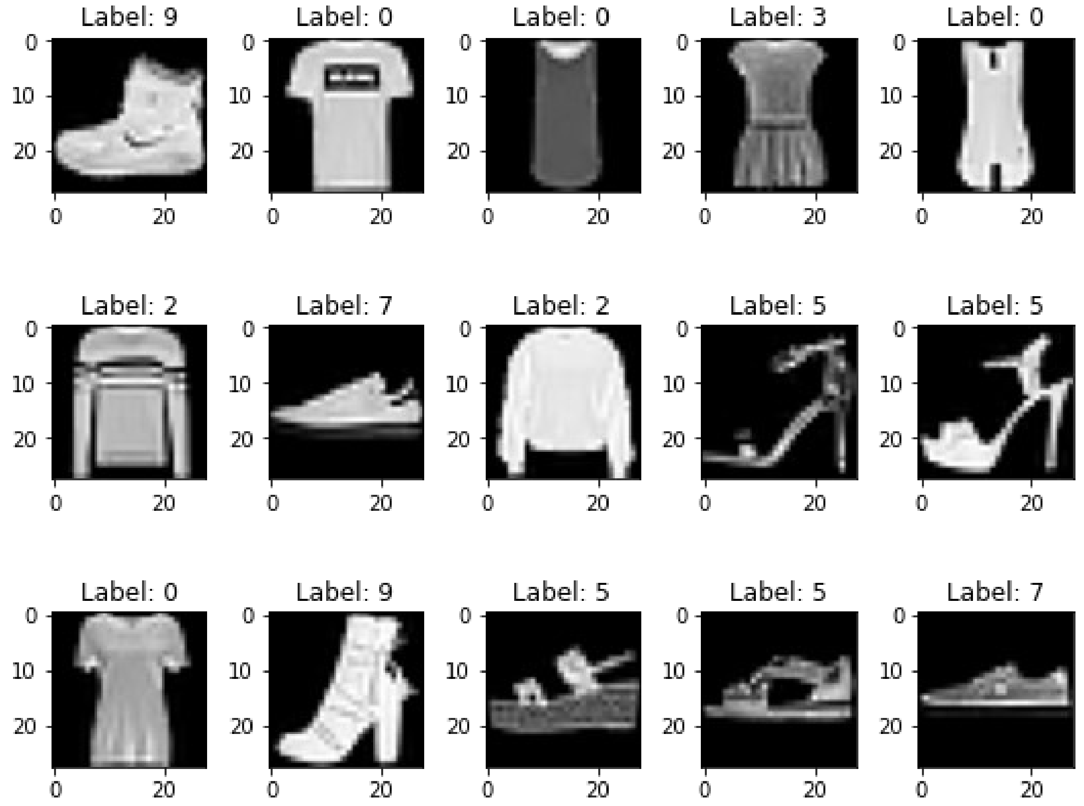
SVHN (Netzer et al. 2011)

- Street View House Numbers (10 classes)
- Median height = 28 pixels
- Training set: 73,257 images
  - Needs to be split into training and validation
- Test set of 26,032 examples
- Typical performance > 95%

Fashion-MNIST (Xiao et al. 2017)

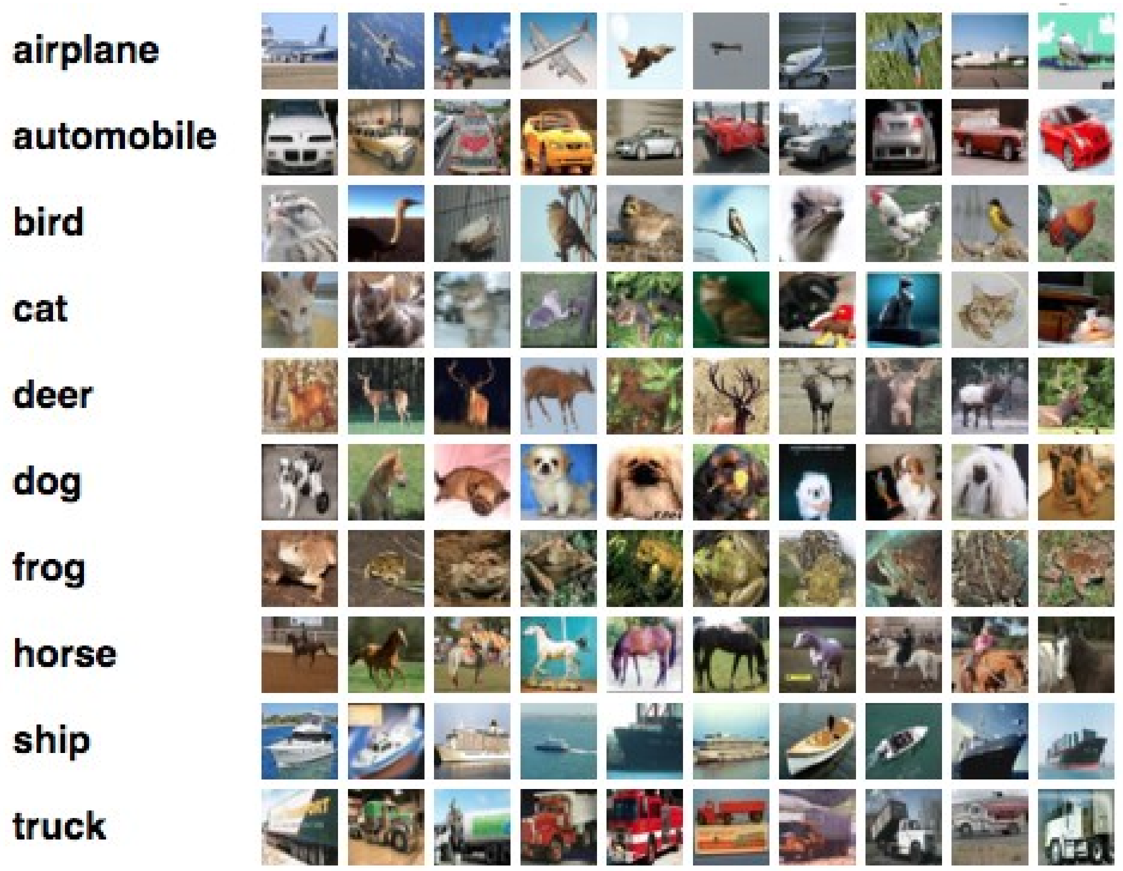
Label	Description	Label	Description
0	T-shirt/top	5	Sandal
1	Trouser	6	Shirt
2	Pullover	7	Sneaker
3	Dress	8	Bag
4	Coat	9	Ankle boot

- Classification of fashion items (10 classes)
- Image size: 28x28
- Training set: 60,000 images
  - 6000 for each class
  - Needs to be split into training and validation
- Test set of 10,000 examples
  - 1000 for each class
- Typical performance > 89%



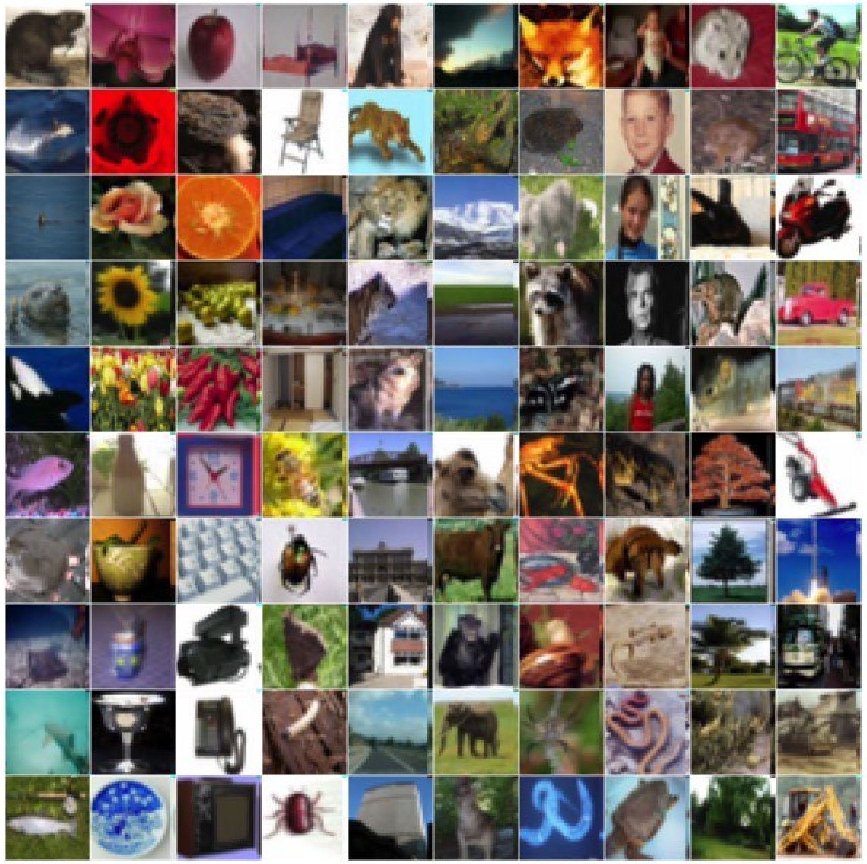
CIFAR-10 (Krizhevsky, 2009)

- Classification of natural images (10 classes)
- Image size: 32x32
- Training set: 50,000 images
  - 5000 for each class
  - Needs to be split into training and validation
- Test set of 10,000 examples
  - 1000 for each class
- Typical performance > 85%
  - ResNet-18: 93%



CIFAR-100 (Krizhevsky, 2009)

- Classification of natural images (100 classes, 20 super classes)
- Image size: 32x32
- Training set: 50,000 images
  - 500 for each class
  - Needs to be split into training and validation
- Test set of 10,000 examples
  - 100 for each class
- Typical performance > 65%



ImageNet-1000 (Jia Deng et al, 2009)

- Also known as ILSVRC
- A subset of “ImageNet”, a large database modeled after WordNet
- Classification of natural images (1000 classes)
- Image size: variable, typically resized to 256x256 and 224x224
- Training: 1,281,167 images
- Validation: 50,000 images
- Test: 100,000 examples
- Best model in 2021: > 90%