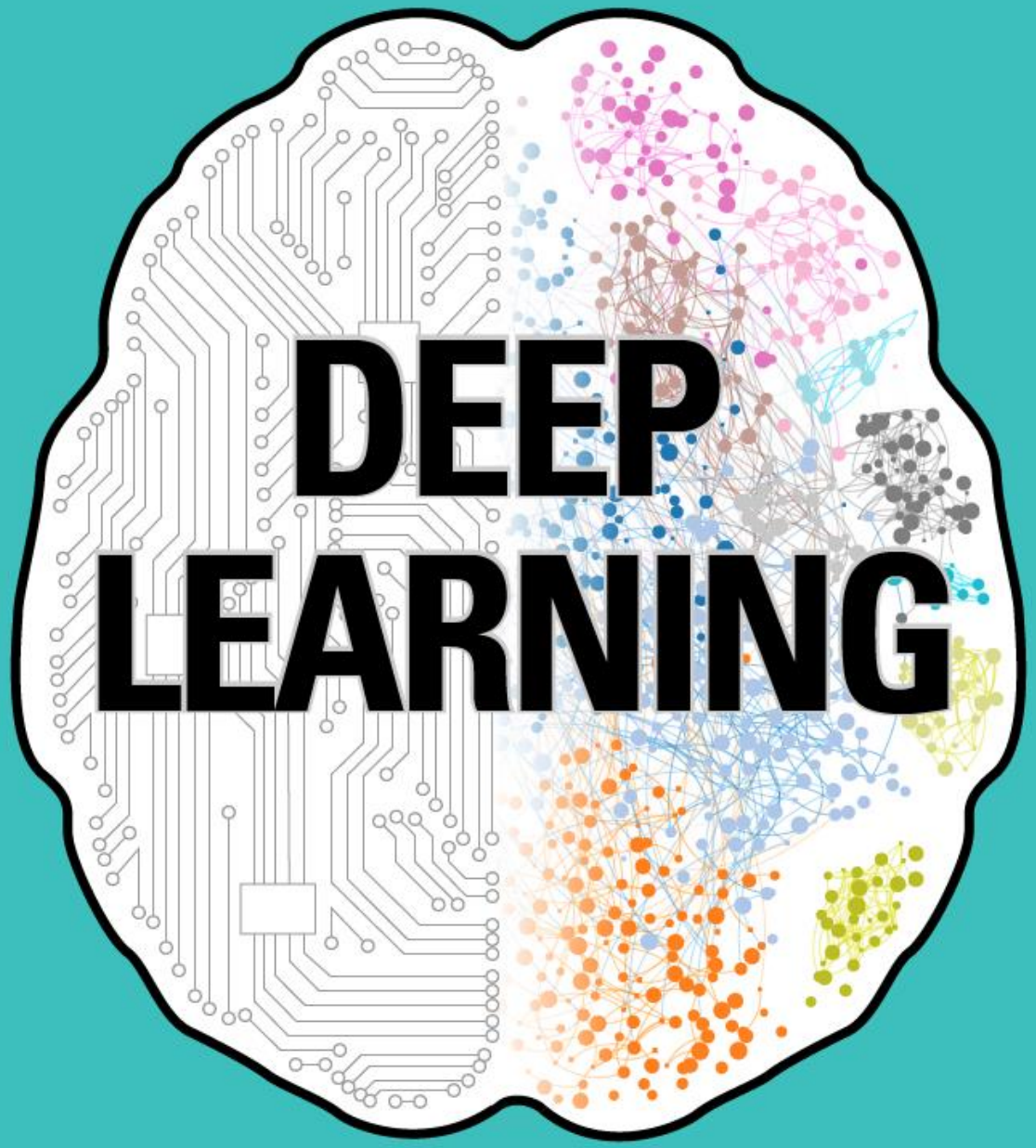
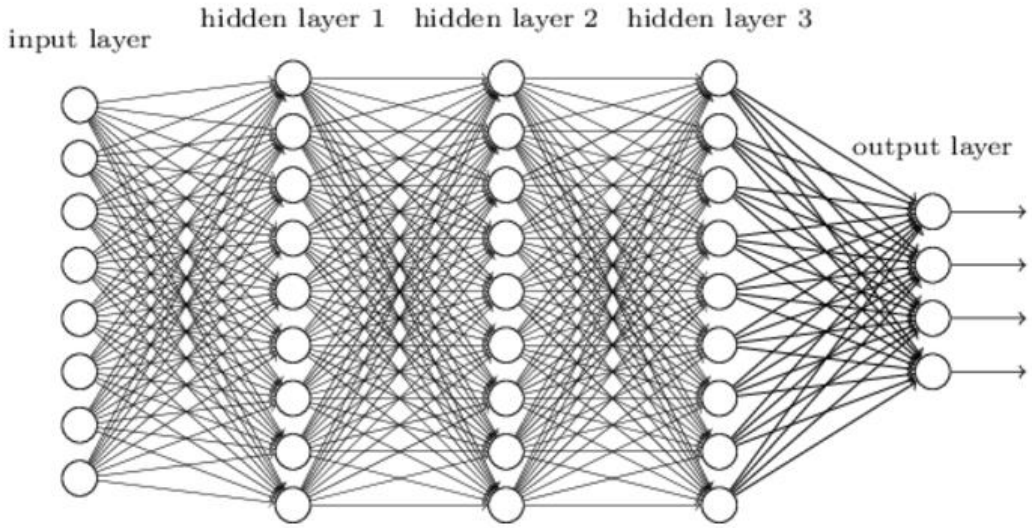


Neural Networks & Deep Learning

http://neuralnetworksanddeeplearning.com/chap6.html

Introducing convolutional networks

Motivation



- We did this using networks in which adjacent network layers are fully connected to one another.
- That is, every neuron in the network is connected to every neuron in adjacent layers.

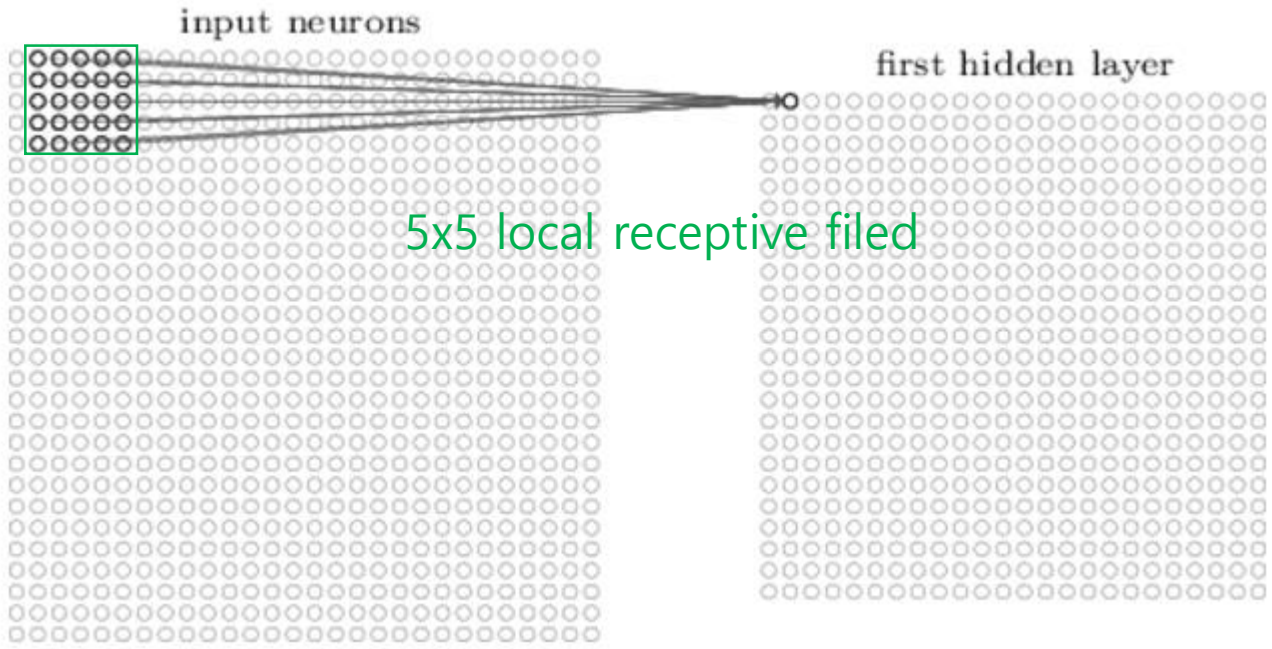
- But upon reflection, it's strange to use networks with fully-connected layers to classify images.
- The reason is that such a network architecture does not take into account the spatial structure of the images.
- For instance, it treats input pixels which are far apart and close together on exactly the same footing.

- Such concepts of spatial structure must instead be inferred from the training data.

- Convolutional neural networks use three basic ideas: *local receptive fields*, *shared weights*, and *pooling*.

Introducing convolutional networks

Local receptive fields



28x28 input image

24x24 neurons

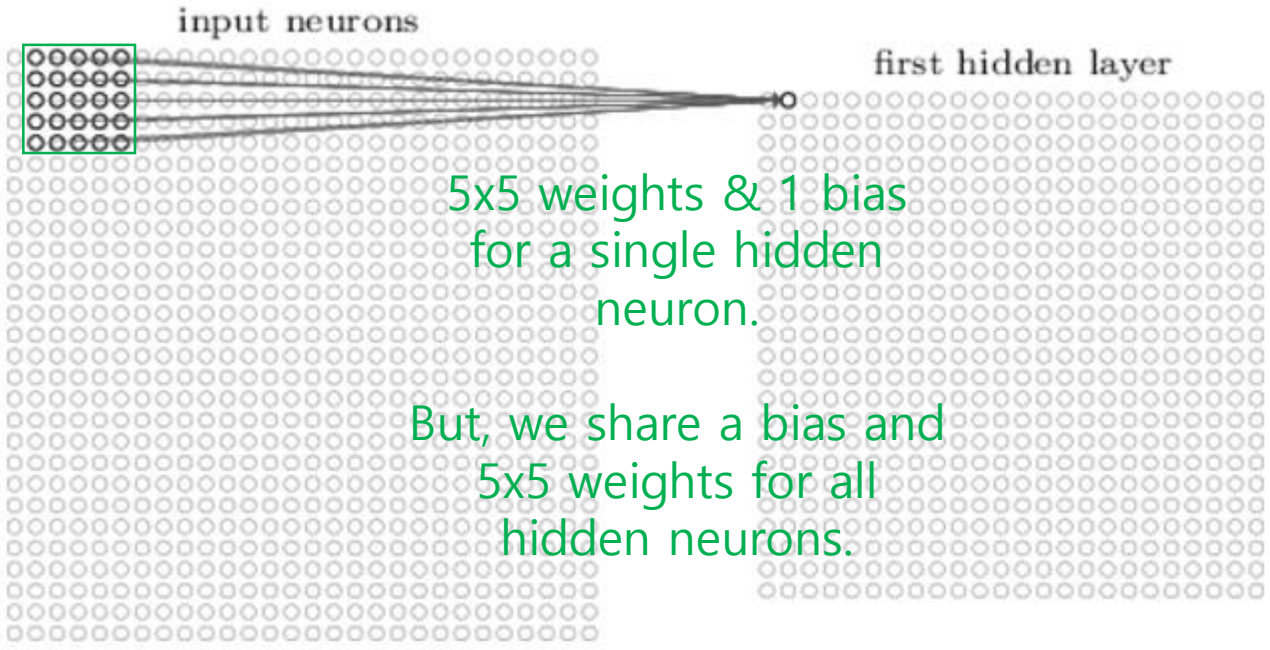
- Stride length indicates distance between local receptive fields.
- In this example, stride length is 1 pixel.

- If we're interested in trying different stride lengths then we can use validation data to pick out the stride length which gives the best performance.

- The same approach may also be used to choose the size of the local receptive field.

Introducing convolutional networks

Shared weights and biases



$$\sigma\left(b + \sum_{l=0}^{4}\sum_{m=0}^{4} w_{l,m} a_{j+l,k+m}\right)$$

- Fully connected : (28x28+1)x(24x24) = 452160 params
- convolution : (5x5+1) = 26 params (0.005%)

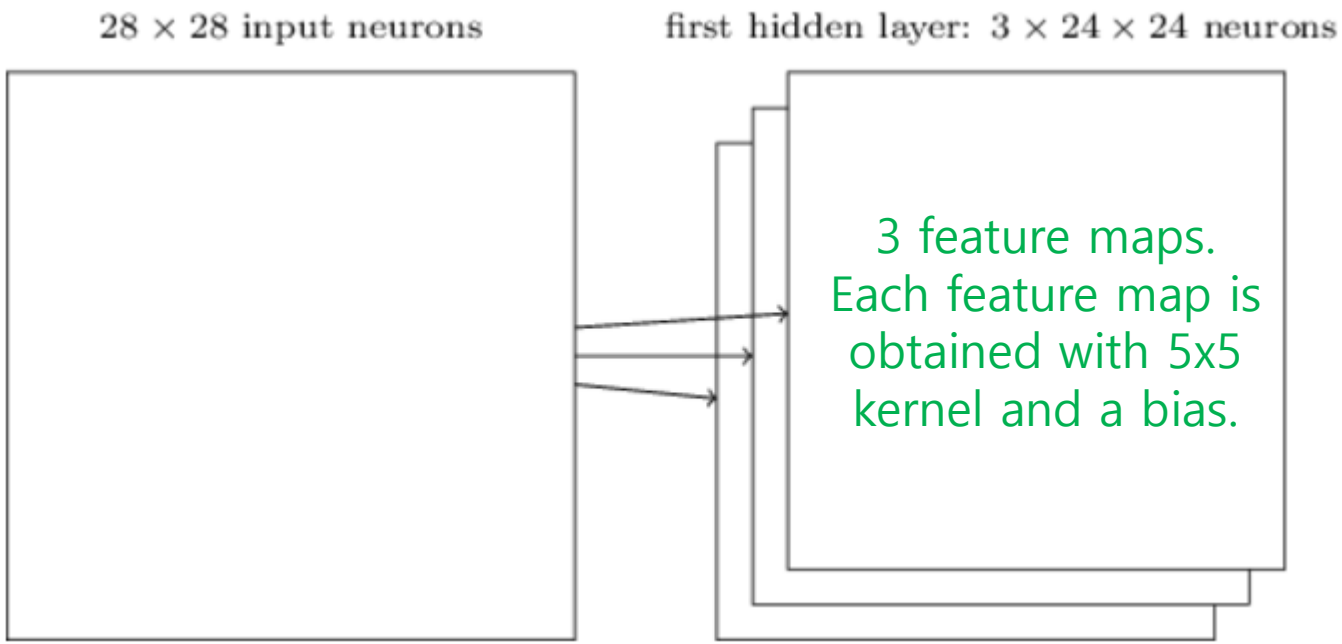
- This means that all the neurons in the first hidden layer detect exactly the same feature, just at different locations in the input image. ➔ spatially invariant information

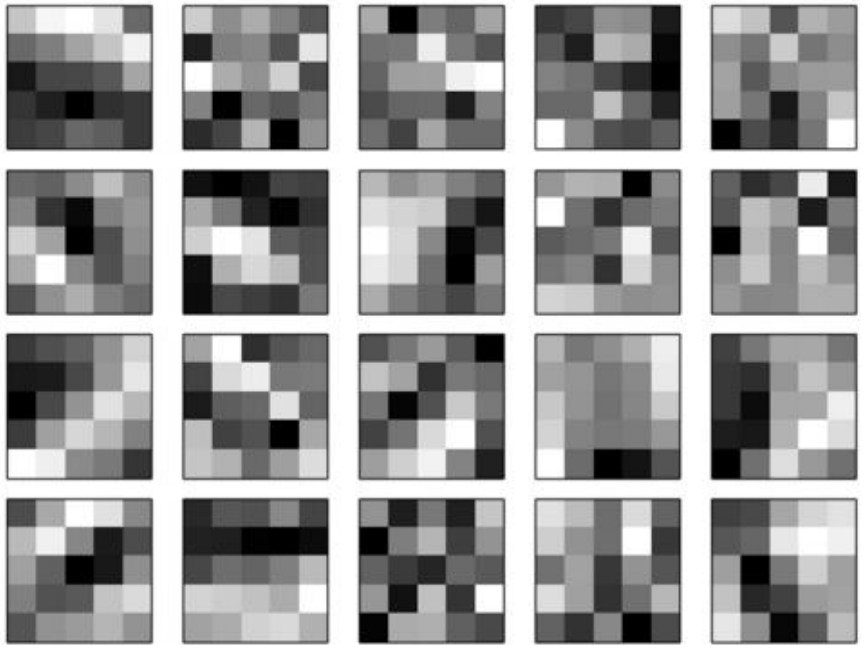
- For this reason, we sometimes call the map from the input layer to the hidden layer a *feature map*. We call the weights defining the feature map the *shared weights*.
- And we call the bias defining the feature map in this way the *shared bias*.
- The shared weights and bias are often said to define a *kernel* or *filter*.

Introducing convolutional networks

Shared weights and biases

- A kernel produces a feature map.
- If we want more feature maps, more kernels must be learned independently.





In a famous NN, LeNet-5 uses 6 feature maps at 1st convolution layer, each associated to a 5x5 local perceptive field.

20 kernels at the 1st layer in the last example, these are learned from MNIST

White (negative weight) ↔ Black (positive weight)

Introducing convolutional networks

Shared weights and biases

Pros

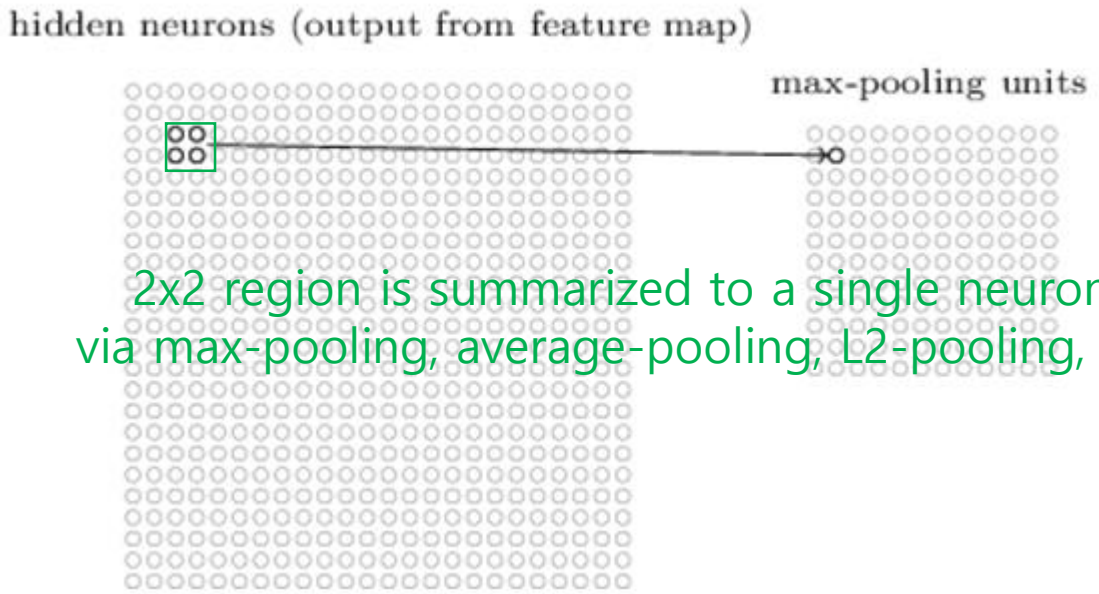
1. It reduced the number of parameters involved in a convolutional network
   : 20 feature maps require (5x5+1)x20 = 520 parameters
   : Fully connected layer with 784 input neurons and 30 hidden neurons requires (784x30+30) = 23550 parameters

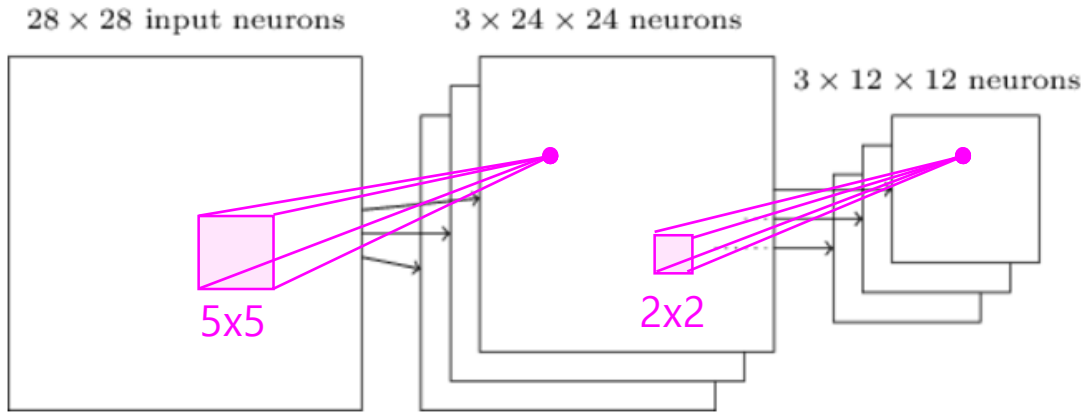
2. It produces translation invariant features which well accounts for spatial structures of image

Introducing convolutional networks

Pooling layers

Pooling layers are usually used immediately after convolutional layers and simplify the information in the output from the convolutional layer.

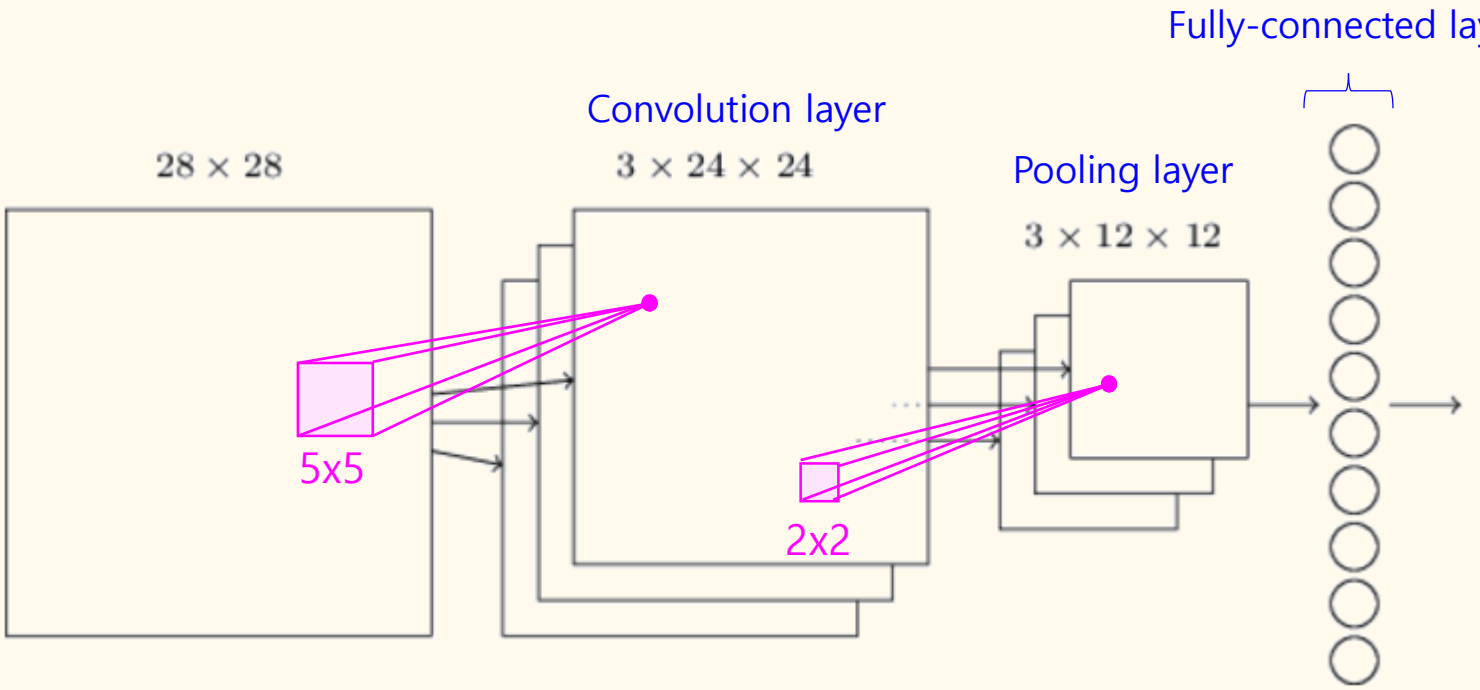




- We can think of max-pooling as a way for the network to ask whether a given feature is found anywhere in a region of the image. It then throws away the exact positional information.
- The intuition is that once a feature has been found, its exact location isn't as important as its rough location relative to other features.
- A big benefit is that there are many fewer pooled features, and so this helps reduce the number of parameters needed in later layers.

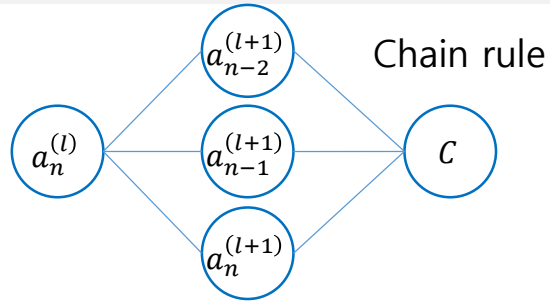
Introducing convolutional networks

Putting it all together



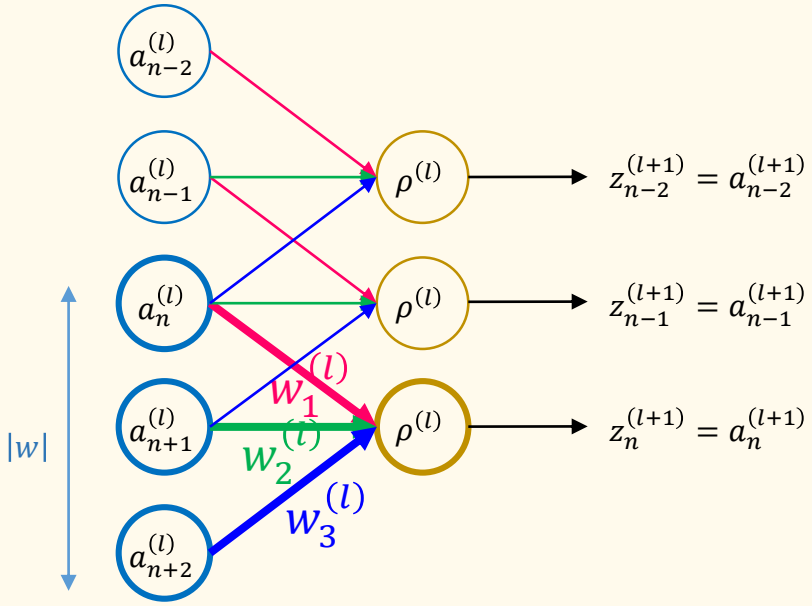
# of parameters : 3x(5x5+1)+10x(3x12x12+1)=78+4321=4399

Backpropagation in convolutional networks



Convolution layer

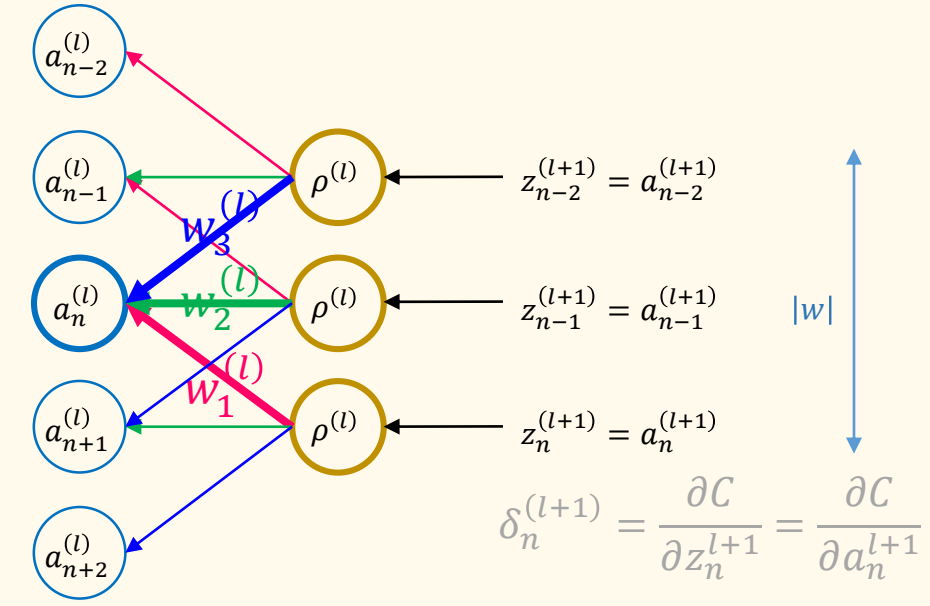
Forward



$$a_n^{(l+1)} = \sum_{i=1}^{|w|} w_i^{(l)} a_{n+i-1}^{(l)} = \left(a^{(l)} * w^{(l)}\right)[n]$$

$$\frac{\partial a_{n-i+1}^{(l+1)}}{\partial a_n^{(l)}} = w_i^{(l)} \qquad \frac{\partial a_n^{(l+1)}}{\partial w_i^{(l)}} = a_{n+i-1}^{(l)}$$

Backward



$$\frac{\partial C}{\partial a_n^{(l)}} = \sum_{i=1}^{|w|} \frac{\partial C}{\partial a_{n-i+1}^{(l+1)}} \frac{\partial a_{n-i+1}^{(l+1)}}{\partial a_n^{(l)}} = \sum_{i=1}^{|w|} \delta_{n-i+1}^{(l+1)} w_i^{(l)}$$

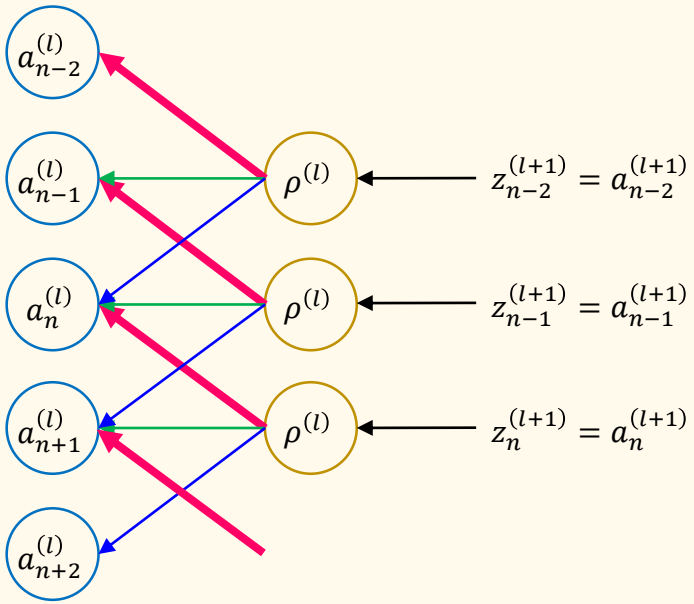
In order to follows the same notation, 'w' must be flipped

$$\frac{\partial C}{\partial a_n^{(l)}} = \sum_{i=1}^{|w|} w_{flip,i}^{(l)} \delta_{n+i-|w|}^{(l+1)} = \left(\delta^{(l+1)} * w_{flip}^{(l)}\right)[n]$$

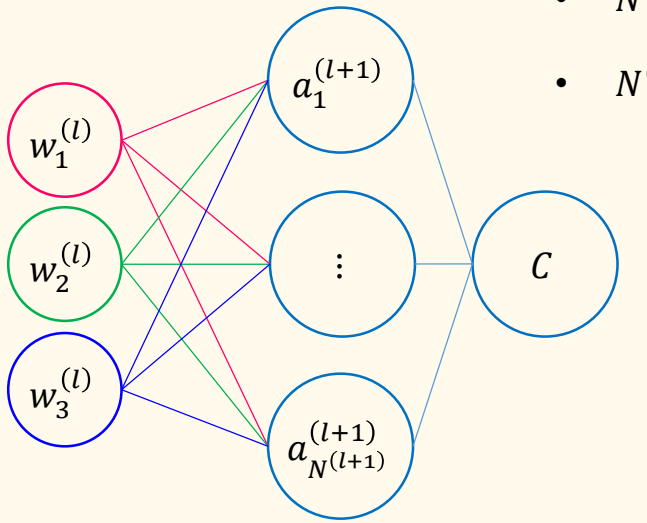
Backpropagation in convolutional networks

Convolution layer

Backward



Chain rule

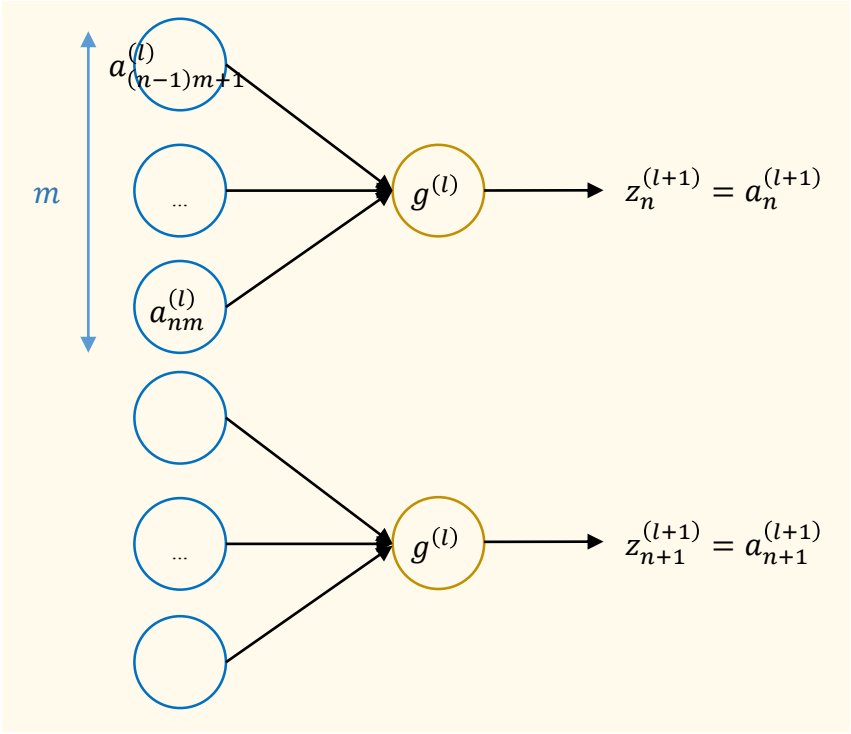


- $N^{(l)}$: *the number of neurons in the* (l)*th layer*
- $N^{(l+1)} = N^{(l)} - |w| + 1$

$$\frac{\partial C}{\partial w_i^{(l)}} = \sum_{n=1}^{N^{(l+1)}} \frac{\partial C}{\partial a_n^{(l+1)}} \frac{\partial a_n^{(l+1)}}{w_i^{(l)}} = \sum_{n=1}^{N^{(l)}-|w|+1} \delta_n^{(l+1)} a_{n+i-1}^{(l)}$$

Backpropagation in convolutional networks

Pooling layer



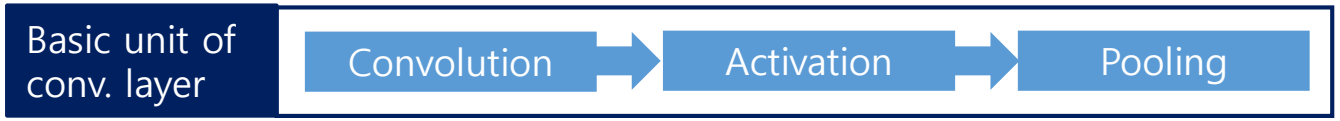
Pooling/Subsampling Functions

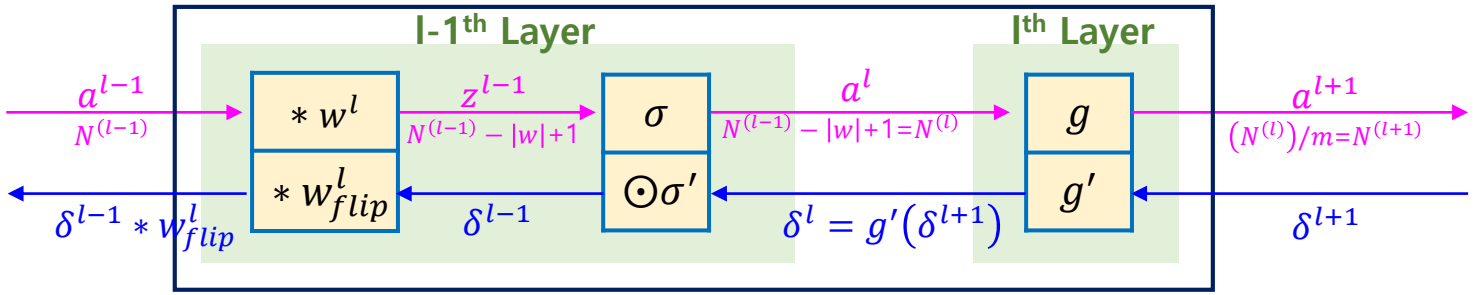
$$g(x) = \begin{cases} \dfrac{\sum_{k=1}^{m} x_k}{m}, \dfrac{\partial g}{\partial x} = \dfrac{\mathbf{1}}{m} & \text{mean pooling} \\ \max(x), \dfrac{\partial g}{\partial x_i} = \begin{cases} 1 \text{ if } x_i = \max(x) \\ 0 \text{ otherwise} \end{cases} & \text{max pooling} \\ \|x\|_p = \left(\sum_{k=1}^{m} |x_k|^p \right)^{1/p}, \dfrac{\partial g}{\partial x_i} = \left(\sum_{k=1}^{m} |x_k|^p \right)^{1/p-1} |x_i|^{p-1} & \text{L}^\text{p} \text{ pooling} \\ \text{or any other differentiable } \mathbf{R}^m \to \mathbf{R} \text{ functions} \end{cases}$$

$$\frac{\partial C}{\partial a^{(l)}_{(n-1)m+1:nm}} = \frac{\partial C}{\partial z^{(l+1)}_n} \frac{\partial z^{(l+1)}_n}{\partial a^{(l)}_{(n-1)m+1:nm}} = \delta^{(l+1)}_n \frac{\partial z^{(l+1)}_n}{\partial a^{(l)}_{(n-1)m+1:nm}} \triangleq \delta^{(l+1)}_n \cdot \left(g^{(l)}\right)'_{(n-1)m+1:nm}$$

Backpropagation in convolutional networks

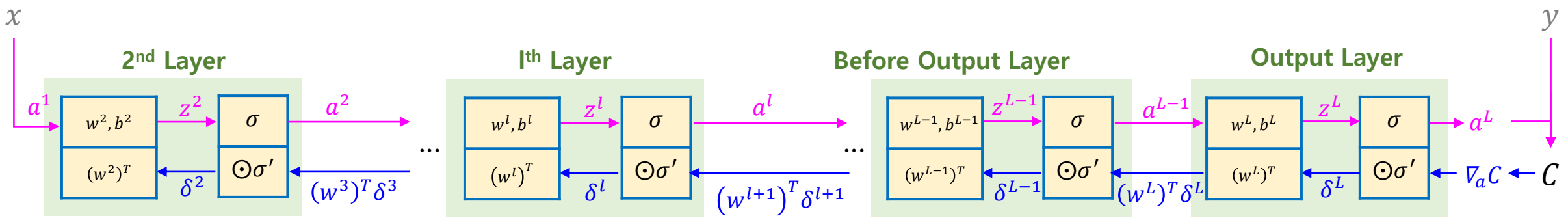
Summary





$$\delta^{l-1} = \sigma'(z^{l-1}) \odot \delta^l$$

For a training sample (x, y)



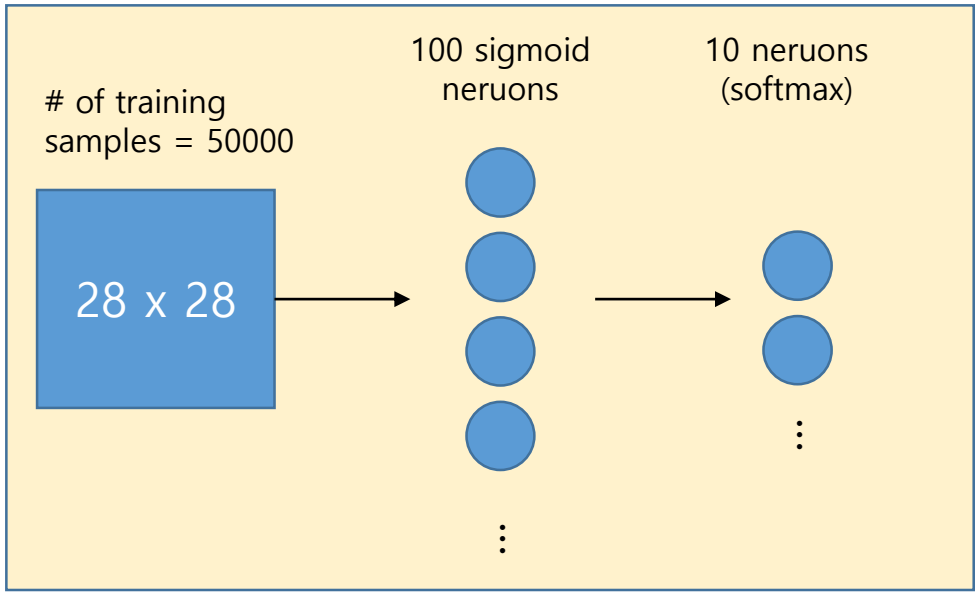
$$\delta^l = \sigma'(z^l) \odot \left((w^{l+1})^T \delta^{l+1}\right)$$

$$\delta^{L-1} = \sigma'(z^{L-1}) \odot ((w^L)^T \delta^L)$$

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

Convolutional neural networks in practice

Baseline

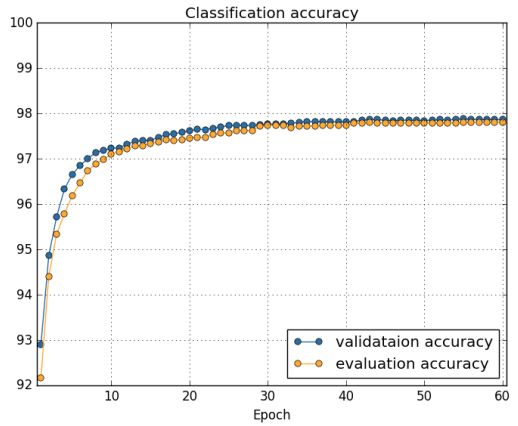


$\eta = 0.1, 60\ epochs, 10\ mini - batch\ size$

n = 100 and Cross-entropy cost function without regularization gives **97%** in previous chapter.

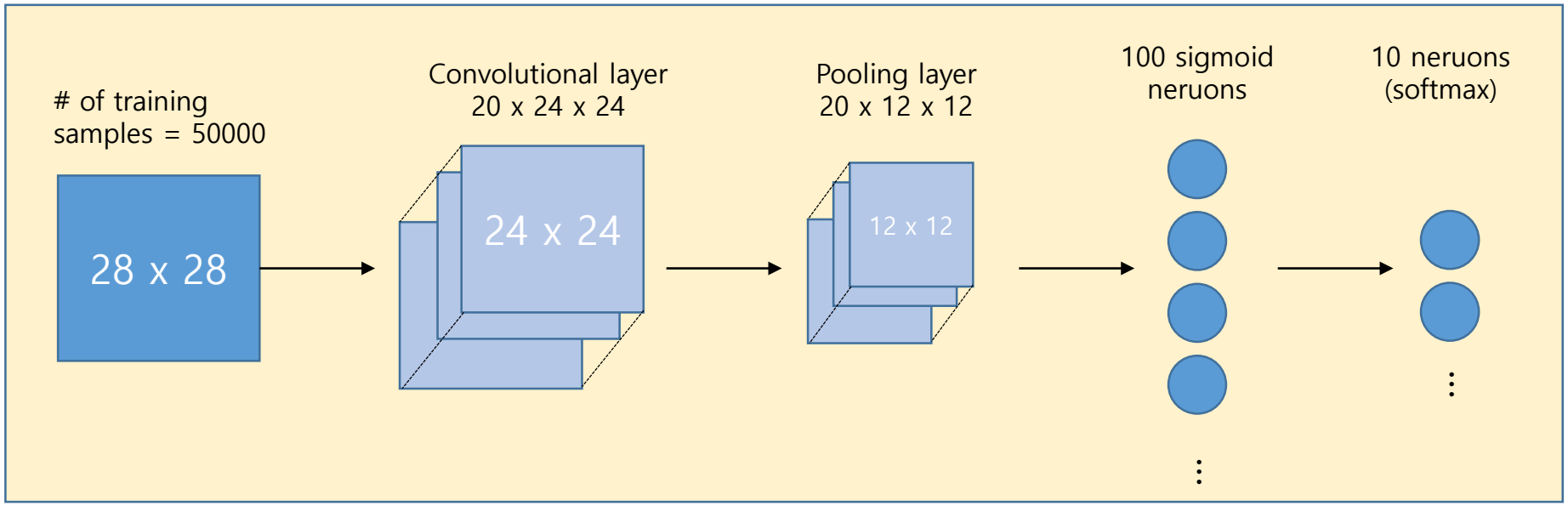
Cross-entropy cost function → log-likelihood cost function
Output layer is replaced with softmax-layer.
These are very common and give **97.80%** accuracy.

python train.py --model baseline



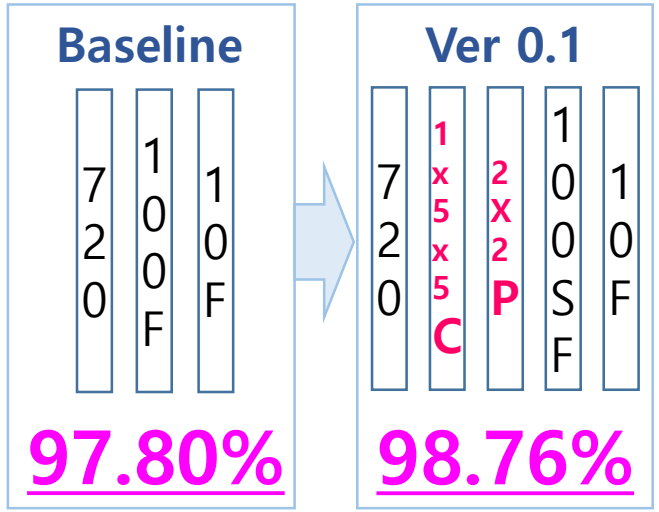
Convolutional neural networks in practice

Ver 0.1 (Inserting Conv.+Pooling layers)



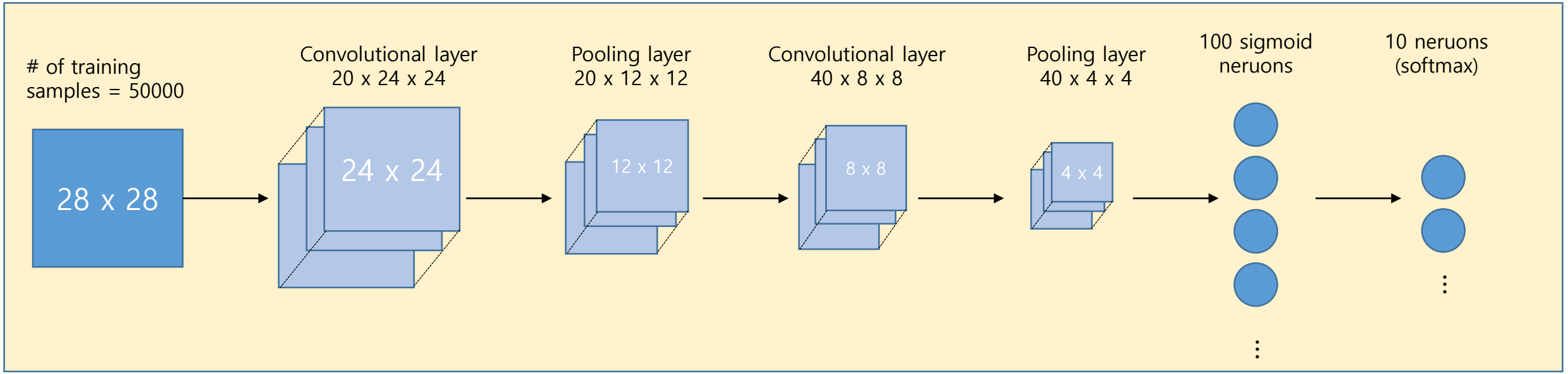
$\eta = 0.1, 60\ epochs, 10\ mini-batch\ size$

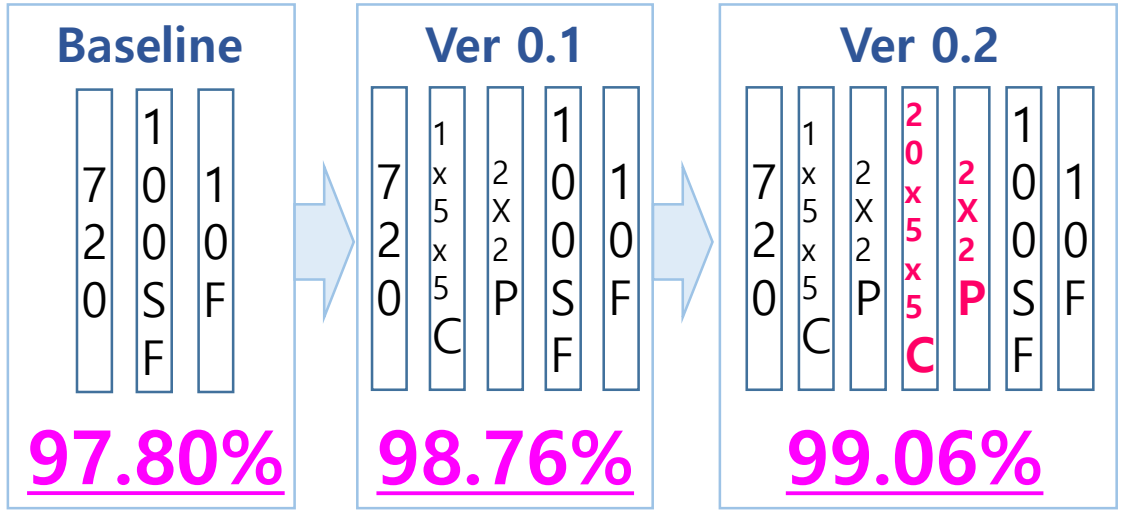
python train.py --model v1



Convolutional neural networks in practice

Ver 0.2 (Inserting Conv.+Pooling layers)





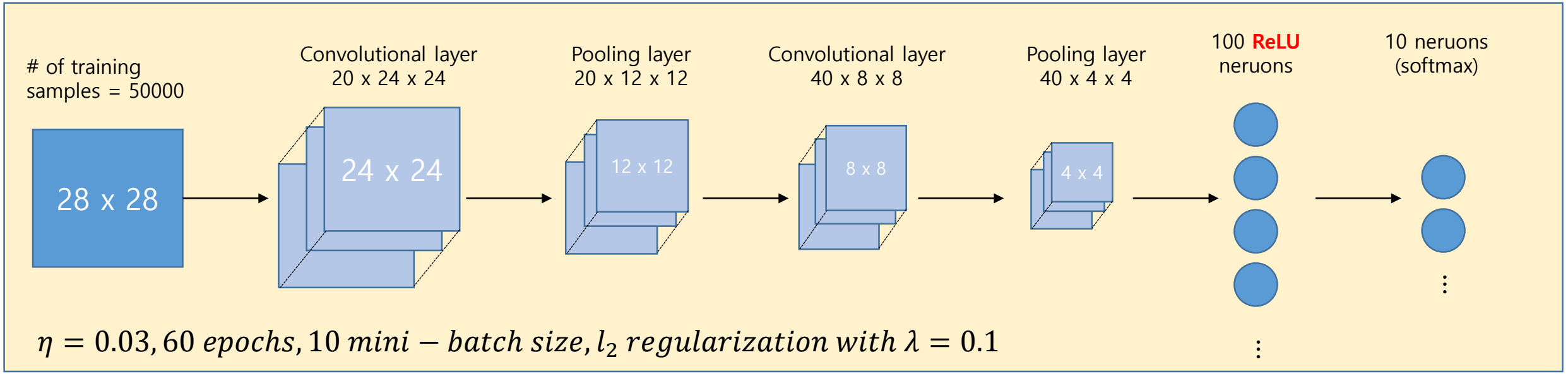
$\eta = 0.1, 60\ epochs, 10\ mini-batch\ size$

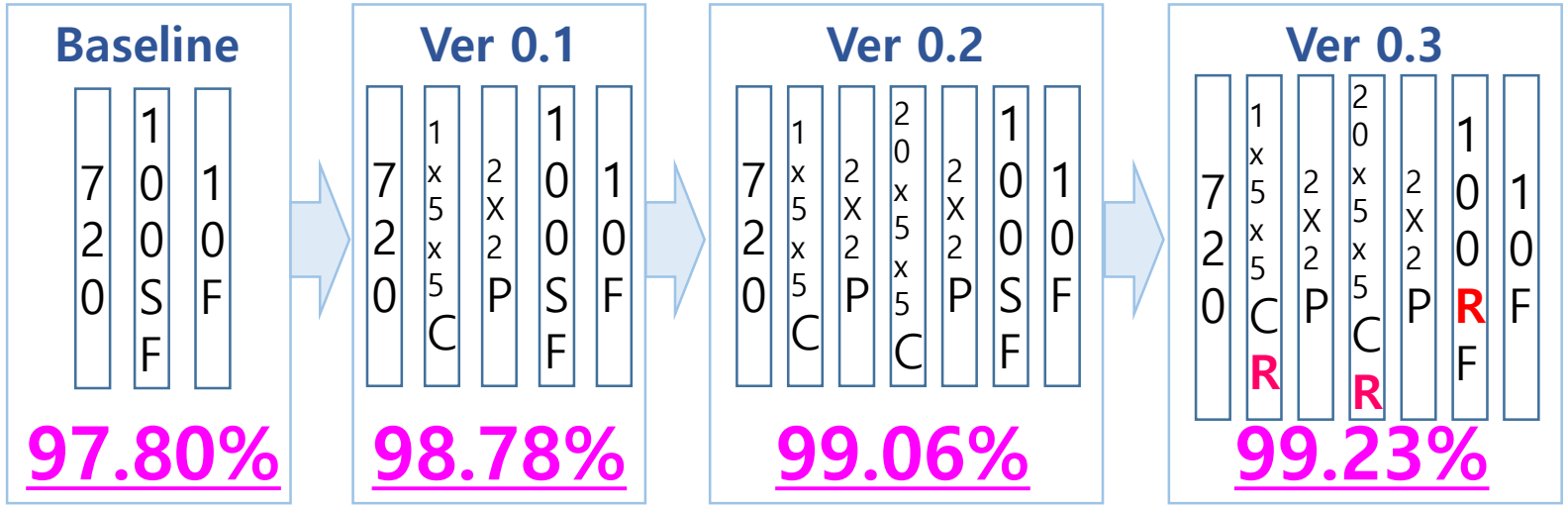
python train.py --model v2

Convolutional neural networks in practice

Ver 0.3 (Replacing ReLU with Sigmoid)

python train.py --model v3





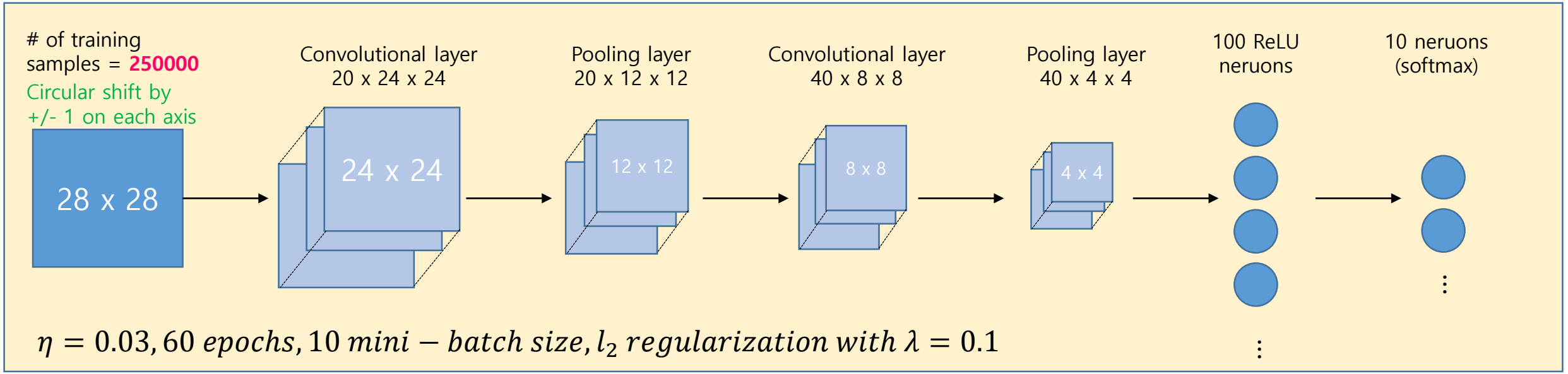
Networks based on rectified linear units consistently outperformed networks based on sigmoid activation functions.

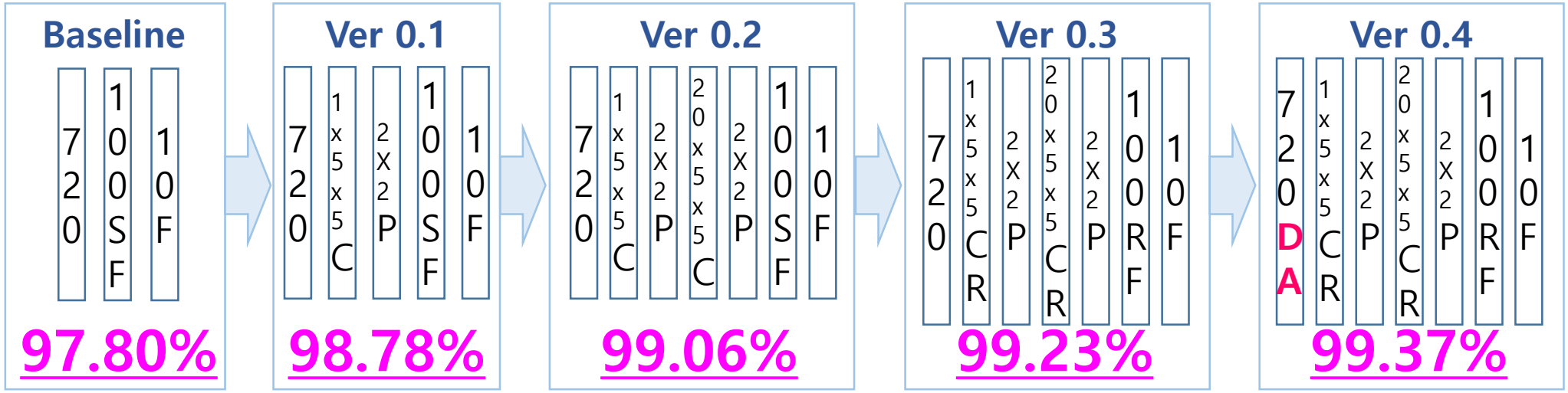
What makes the rectified linear activation function better than the sigmoid or tanh functions? At present, we have a poor understanding of the answer to this question.

Convolutional neural networks in practice

Ver 0.4 (Expanding the training data : data augmentation)

python train.py --model v4

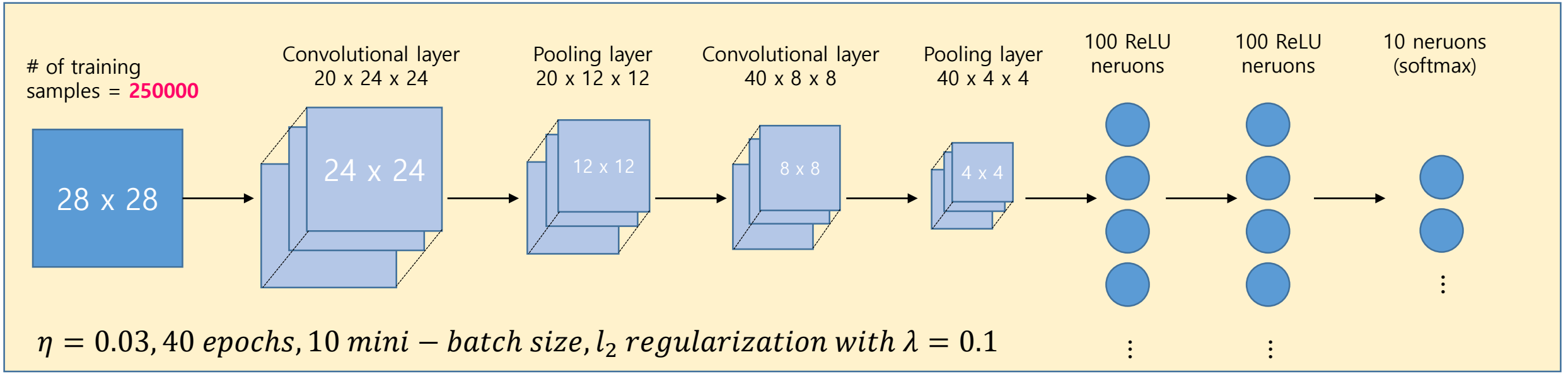




Convolutional neural networks in practice

Ver 0.5 (Inserting an extra fully-connected layer)

python train.py --model v5



Baseline

720 | 100 SF | 10 F

97.80%

Ver 0.1

720 | 1x5x5 C | 2X2 P | 100 SF | 10 F

98.78%

Ver 0.2

720 | 1x5x5 C | 2X2 P | 20x5x5 C | 2X2 P | 100 SF | 10 F

99.06%

Ver 0.3

720 | 1x5x5 CR | 2X2 P | 20x5x5 CR | 2X2 P | 100 RF | 10 F

99.23%

Ver 0.4

720 DA | 1x5x5 CR | 2X2 P | 20x5x5 CR | 2X2 P | 100 RF | 10 F

99.37%

Ver 0.5

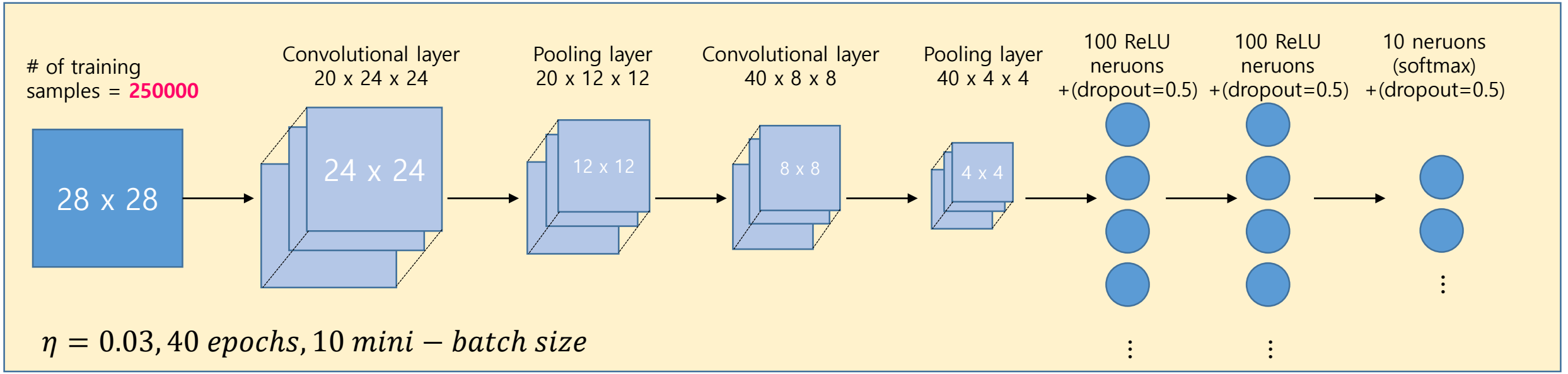
720 DA | 1x5x5 CR | 2X2 P | 20x5x5 CR | 2X2 P | 100 R | **100 R** | 10 F

99.43%

Convolutional neural networks in practice

Ver 0.5 (Employing dropout technique)

python train.py --model v6



Ver 0.1 — 720 | 1x5x5C | 2x2P | 100SF | 10F — **98.78%**

Ver 0.2 — 720 | 1x5x5C | 2x2P | 20x5x5C | 2x2P | 100SF | 10F — **99.06%**

Ver 0.3 — 720 | 1x5x5CR | 2x2P | 20x5x5CR | 2x2P | 100RF | 10F — **99.23%**

Ver 0.4 — 720DA | 1x5x5CR | 2x2P | 20x5x5CR | 2x2P | 100RF | 10F — **99.37%**

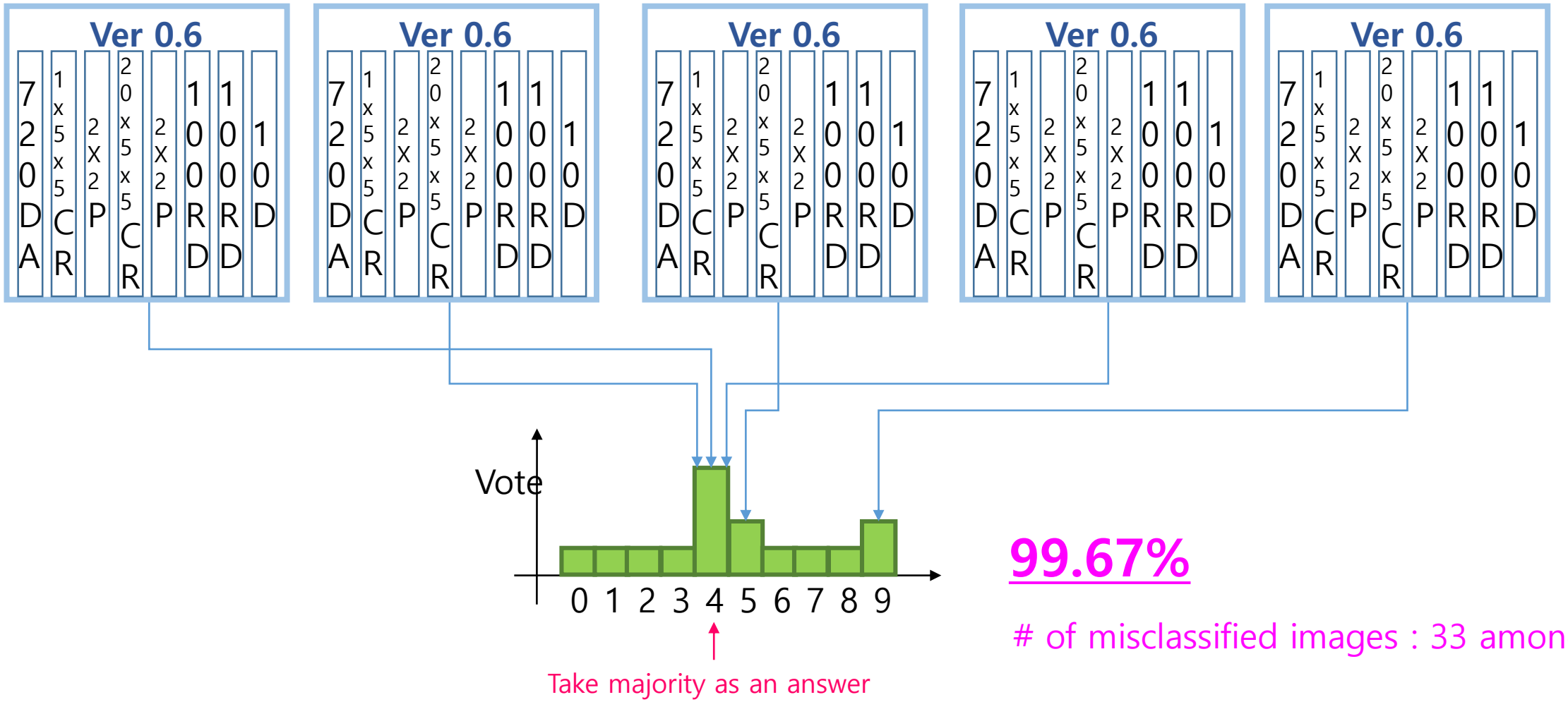
Ver 0.5 — 720DA | 1x5x5CR | 2x2P | 20x5x5CR | 2x2P | 100R | 100R | 10F — **99.43%**

Ver 0.6 — 720DA | 1x5x5CR | 2x2P | 20x5x5CR | 2x2P | 100RD | 100RD | 10D — **99.60%**

```
Training mini-batch number 351000
Training mini-batch number 352000
Training mini-batch number 353000
Training mini-batch number 354000
Training mini-batch number 355000
Training mini-batch number 356000
Training mini-batch number 357000
Training mini-batch number 358000
Training mini-batch number 359000
Training mini-batch number 360000
Training mini-batch number 361000
Training mini-batch number 362000
Training mini-batch number 363000
Training mini-batch number 364000
Training mini-batch number 365000
Training mini-batch number 366000
Training mini-batch number 367000
Training mini-batch number 368000
Training mini-batch number 369000
Training mini-batch number 370000
Training mini-batch number 371000
Training mini-batch number 372000
Training mini-batch number 373000
Training mini-batch number 374000
Epoch 14: validation accuracy 99.54%
This is the best validation accuracy to date.
The corresponding test accuracy is 99.62%
Training mini-batch number 375000
Training mini-batch number 376000
Training mini-batch number 377000
Training mini-batch number 378000
Training mini-batch number 379000
Training mini-batch number 380000
Training mini-batch number 381000
Training mini-batch number 382000
Training mini-batch number 383000
```

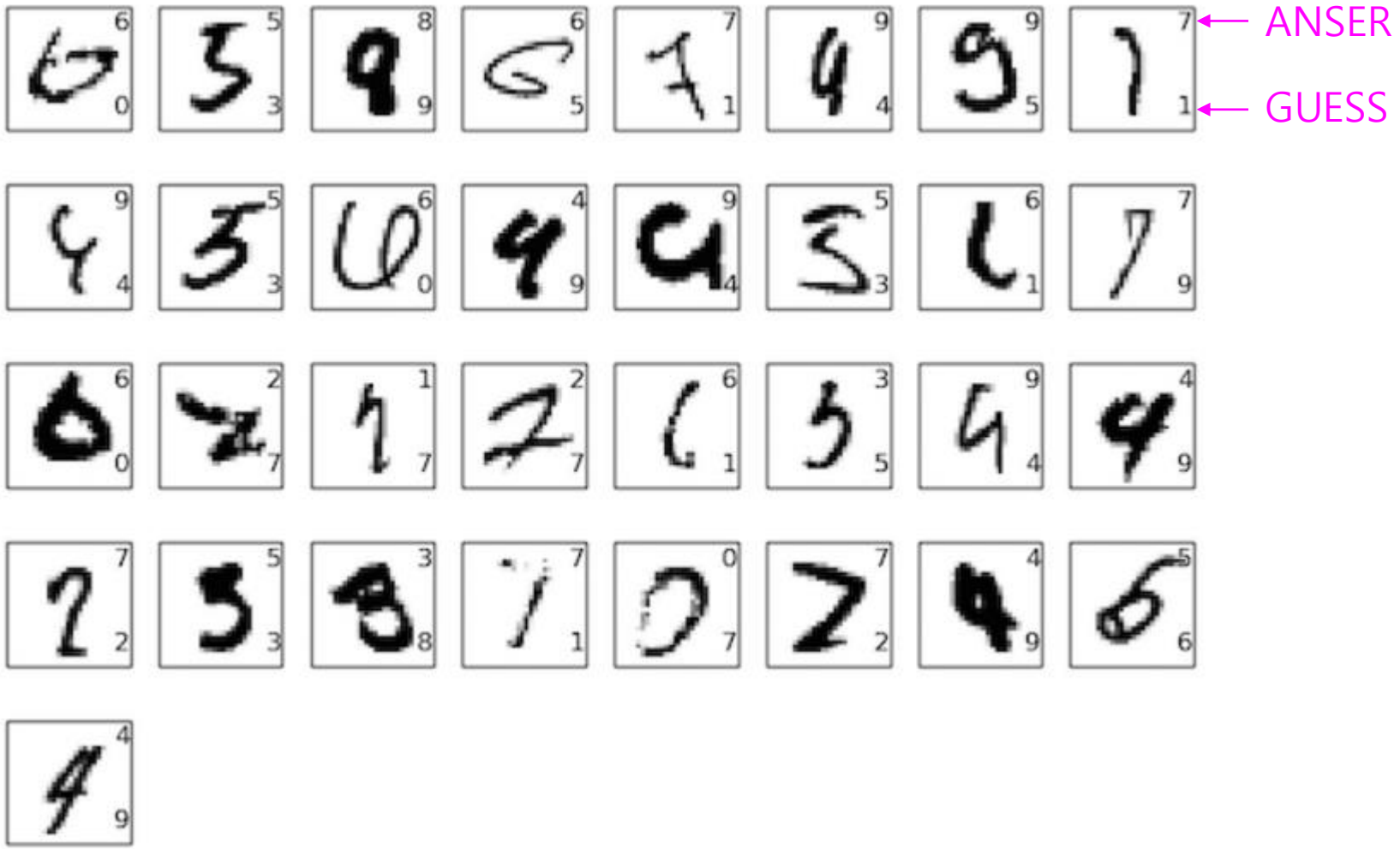
Convolutional neural networks in practice

Ver 0.7 (Ensemble of neural networks)



Convolutional neural networks in practice

Ver 0.6 (Ensemble of neural networks)

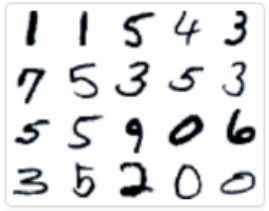


Convolutional neural networks in practice

Going further

Please visit the following URL

http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html



MNIST 50 results collected

Units: error %

> Classify handwriten digits. Some additional results are available on the original dataset page.

Result	Method	Venue	Details
0.21%	Regularization of Neural Networks using DropConnect	ICML 2013	
0.23%	Multi-column Deep Neural Networks for Image Classification	CVPR 2012	
0.23%	APAC: Augmented PAttern Classification with Neural Networks	arXiv 2015	
0.24%	Batch-normalized Maxout Network in Network	arXiv 2015	Details
0.29%	Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree	AISTATS 2016	Details
0.31%	Recurrent Convolutional Neural Network for Object Recognition	CVPR 2015	
0.31%	On the Importance of Normalisation Layers in Deep Learning with Piecewise Linear Activation Units	arXiv 2015	
0.32%	Fractional Max-Pooling	arXiv 2015	Details
0.33%	Competitive Multi-scale Convolution	arXiv 2015	

Convolutional neural networks in practice

How have we avoided the vanishing/exploding gradient problems?

Major

(1) Using convolutional layers greatly reduces the number of parameters in those layers, making the learning problem much easier;
(2) Using more powerful regularization techniques (notably dropout and convolutional layers) to reduce overfitting, which is otherwise more of a problem in more complex networks;
(3) Using rectified linear units instead of sigmoid neurons, to speed up training - empirically, often by a factor of 3-5;
(4) Using GPUs and being willing to train for a long period of time.

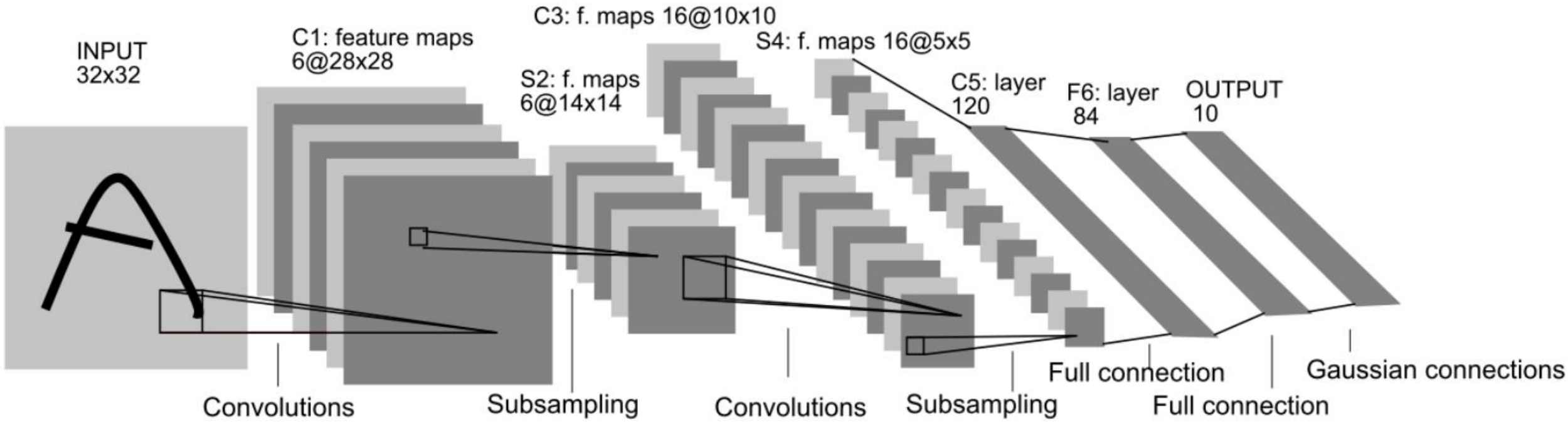
Minor

(1) making use of sufficiently large data sets (to help avoid overfitting);
(2) using the right cost function (to avoid a learning slowdown);
(3) using good weight initializations (also to avoid a learning slowdown, due to neuron saturation);
(4) algorithmically expanding the training data.

Recent progress in image recognition

LeNet 5

Gradient-Based Learning Applied to Document Recognition,
by Y. LeCun, L. Bottou, Y. Bengio and P. Haffner (1998).



- Cx: Convolutional layer
- Sx: Subsample layer
- Fx: Fully connected layer

Recent progress in image recognition

Convolution

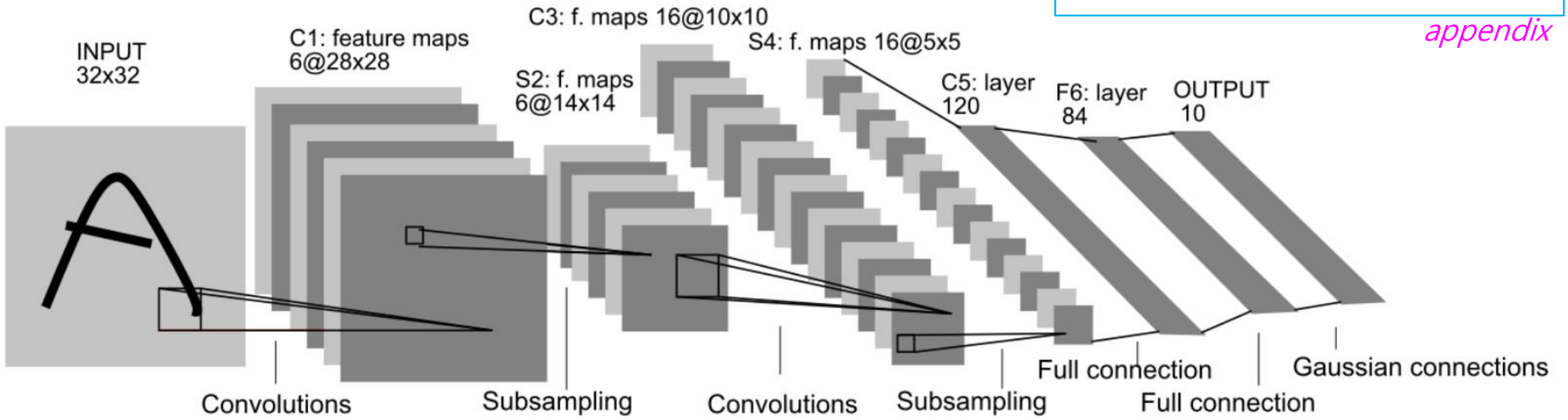
$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

Pooling

$$o = \left\lfloor \frac{i - k}{s} \right\rfloor + 1$$

appendix

LeNet 5



	C1	S2	C3	S4	C5	F6	Output
Input	32x32x1 • i=32	28x28x6 • i=28	14x14x6 • i=14	10x10x16 • i=10	5x5x16 • i=10	• i=120	• i=84
Params.	K=[6,1,5,5] • k=5, s=1, p=0	• k=2, s=2	K=[16,6,5,5] • k=5, p=0, s=1	• k=2, s=2	K=[120,16,5,5] • k=5, p=0, s=1	• W : 84x120 • b : 84x1	• W : 10x84 • b : 10x1
# of params.	(5x5+1)x6= 156	0	(6x5x5+1)x16 = 2416	0	(16x5x5+1)x120= 48120	84*(120+1)= 10164	10x(84+1)= 850
Output	28x28x6 • o=28	14x14x6 • o=14	10x10x16 • o=10	5x5x16 • o=5	1x1x120 • o=1	• o=84	• o=10

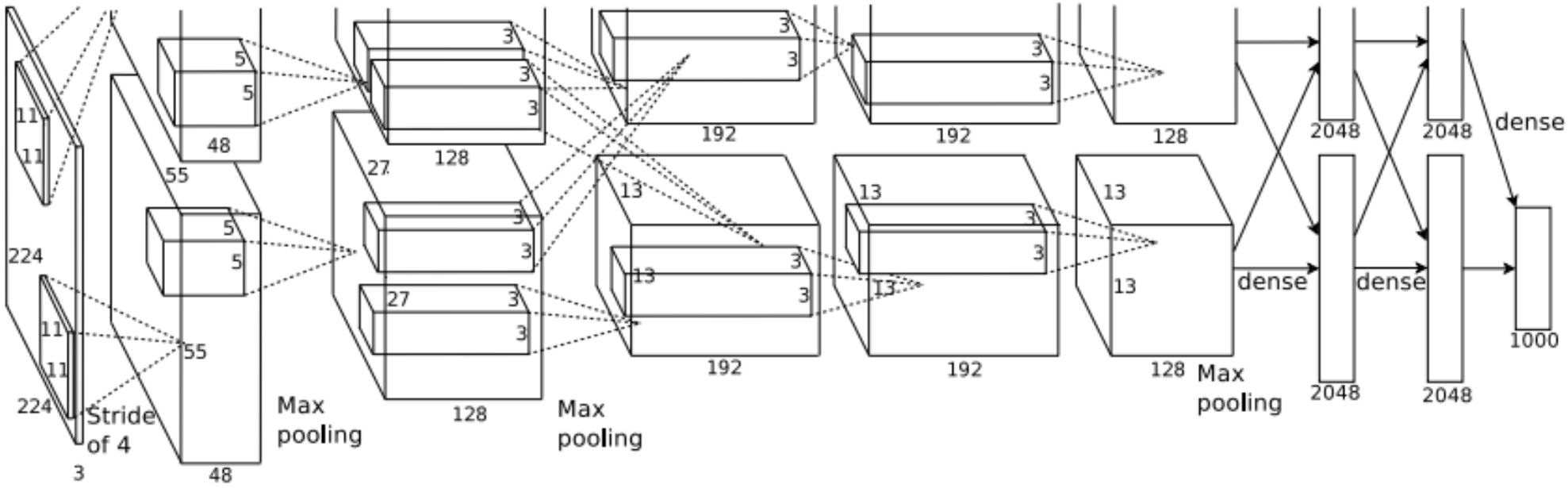
Recent progress in image recognition

AlexNet

ImageNet classification with deep convolutional neural networks,
by Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton (2012).

(1) '12 ILSVRC Top-5 accuracy : 84.7% (2nd place is with 73.8%)
- Single image contains multiple-objects. That's why top-5 is considered

(2) Note that many layers are split into 2 parts, corresponding to the 2 GPUs. (due to memory capacity)

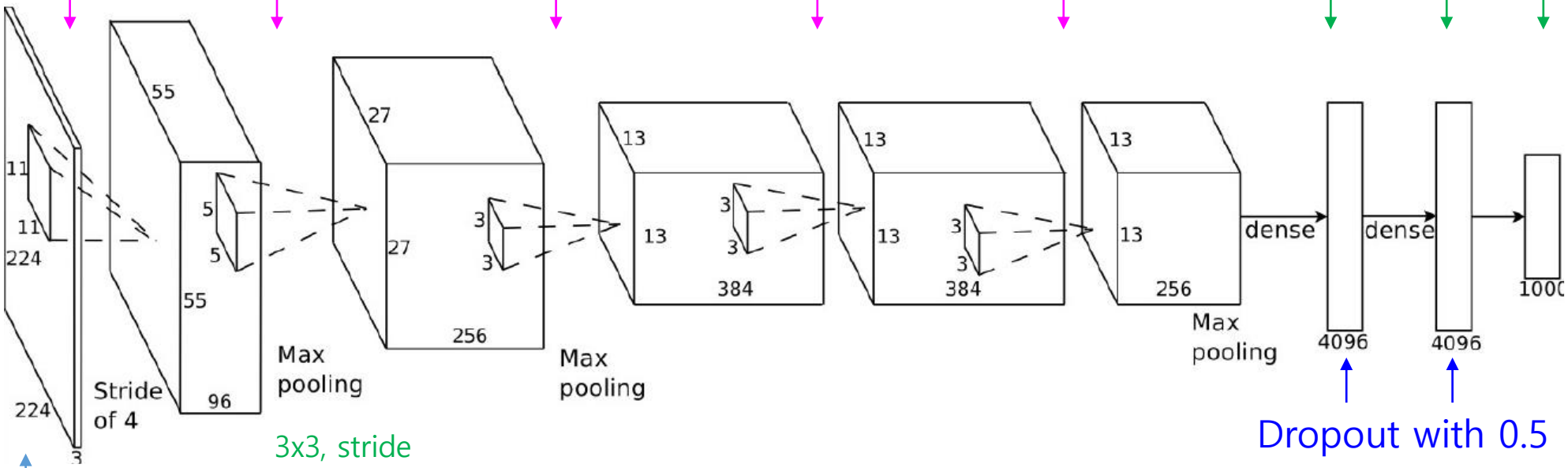


Recent progress in image recognition

AlexNet https://github.com/uoguelph-mlrg/theano_alexnet

5 convolutional layer including ReLU

3 fully connected layers



1. Scaling to 256x? or ?x256
2. Cropping to 256x256
3. Subtracting mean image
4. Cropping to 227x227 (5 subimages)

Data Augmentation

L2 regularization

Momentum-based mini-batch Stochastic gradient descent

Local contrast normalization

Momentum = 0.9
Initial learning rate = 0.01
Periodically drop learning rate by a factor of 10

Recent progress in image recognition

AlexNet : Local response normalization

- No need to input normalization with ReLUs.
- But still the following local normalization scheme helps generalization.

$$b^i_{x,y} = a^i_{x,y} \Big/ \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

Response-normalized activity

Activity of a neuron computed by applying kernel I at position (x,y) and then applying the ReLU nonlinearity

$k = 2$, $n = 5$, $\alpha = 10^{-4}$, $\beta = 0.75$

- Response normalization reduces top-1 and top-5 error rates by 1.4% and 1.2% , respectively.

Recent progress in image recognition

Convolution

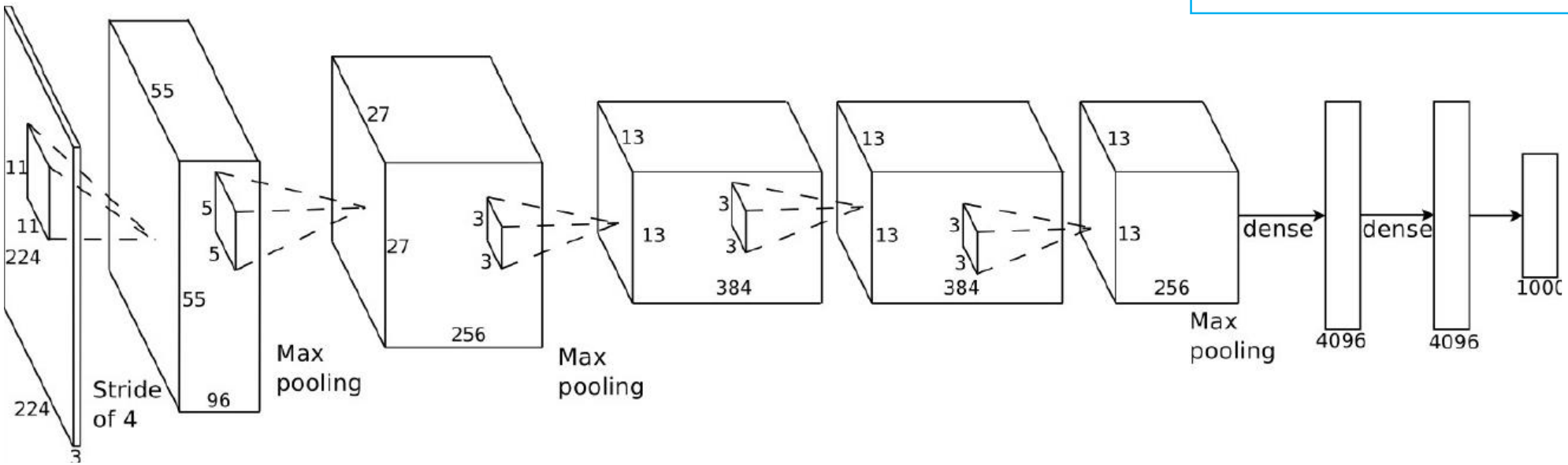
$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

Pooling

$$o = \left\lfloor \frac{i - k}{s} \right\rfloor + 1$$

appendix

AlexNet



https://github.com/uoguelph-mlrg/theano_alexnet

alex_net.py

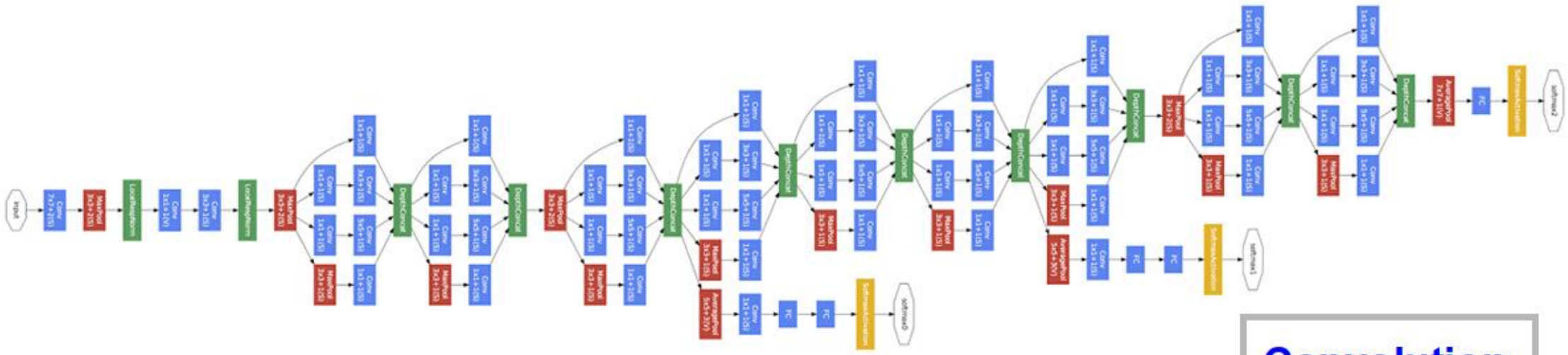
	conv1+relu1+norm1	Pooling1	Conv2+relu2+norm2	Pooling2
Input	227x227x3 • i=227	55x55x96 • i=55	27x27x96 • i=55	27x27x256 • i=13
Params	K=[96,3,11,11] • k=11, s=4, p=0	• k=3, s=2	K=[256,96,5,5] • k=5, p=2, s=1 (half padding)	• k=3, s=2
Output	55x55x96 • o=55	27x27x96 • o=27	27x27x256 • o=27	13x13x256 • o=13

Recent progress in image recognition

GoogLeNet

Going deeper with convolutions, by Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (2014).

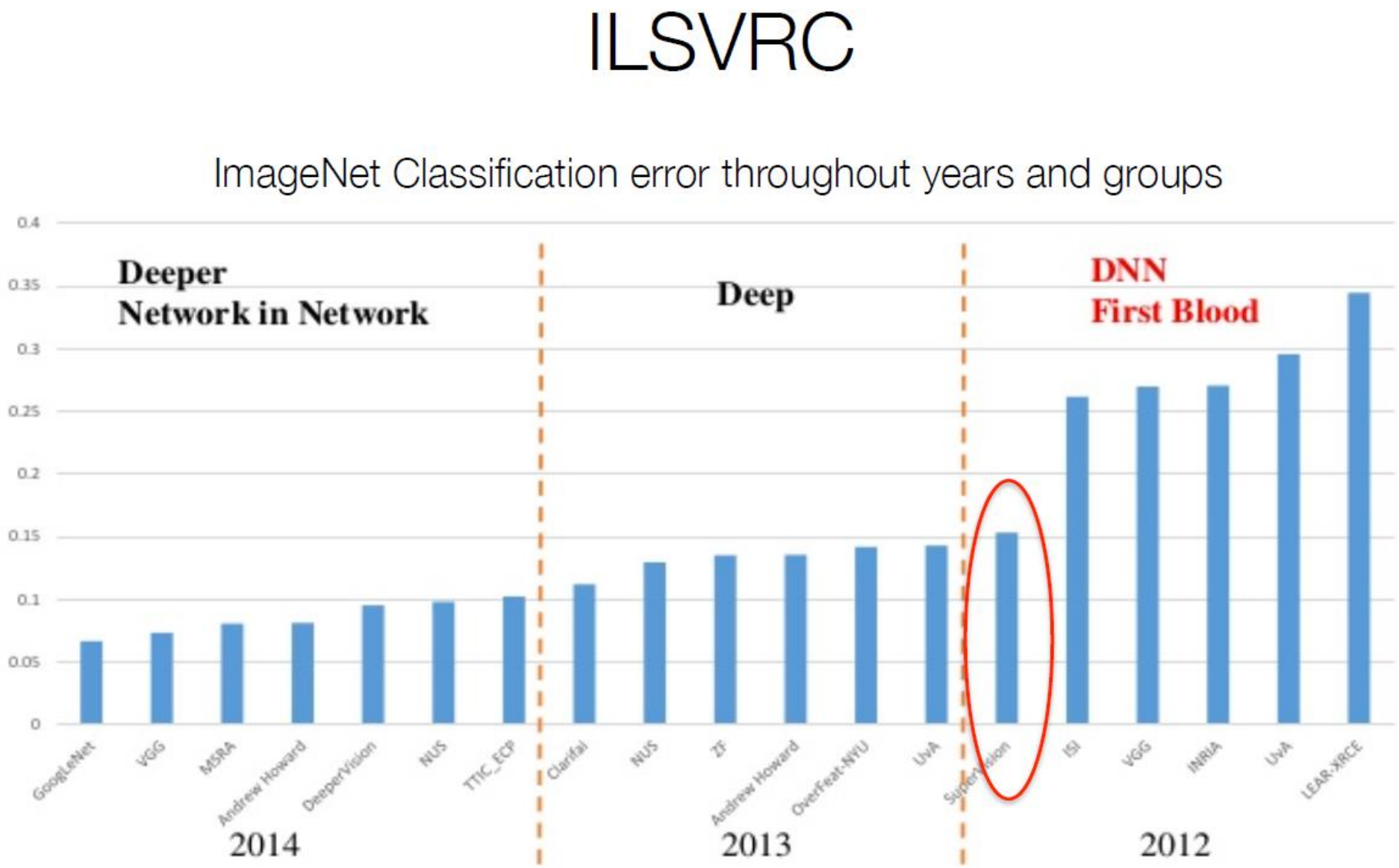
(1) '14 ILSVRC Top-5 accuracy : 93.33% ('13 Winner 88.3%, '12 Winner-AlexNet 84.7%)
(2) Human error : 5.1% VS GoogLeNet error : 6.8%
http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/



25 convolution layers

Convolution
Pooling
Softmax
Other

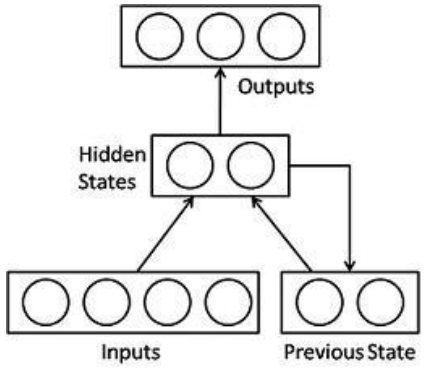
Recent progress in image recognition

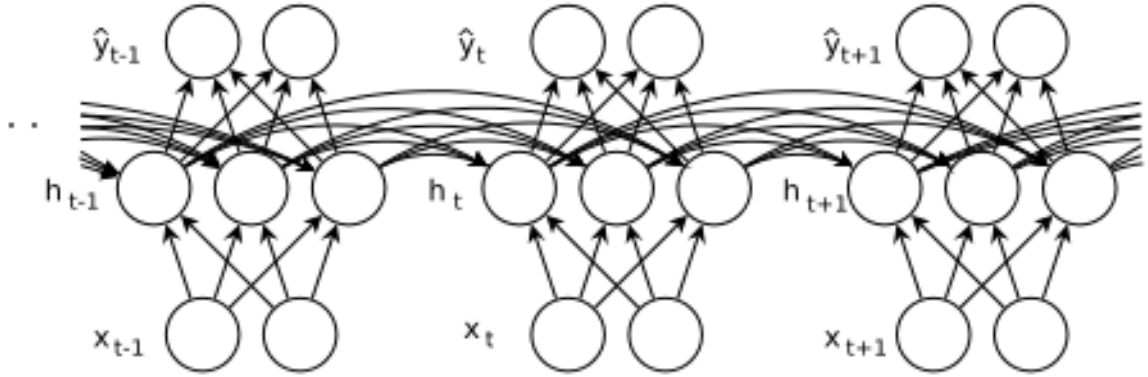


Other approaches to deep neural nets

Recurrent neural networks (RNNs) & Long short-term memory units (LSTMs)

The activations of hidden and output neurons won't be determined just by the current input to the network, but also by earlier inputs.





Time varying function could be modeled with RNNs

1. Speech recognition
2. Video classification
3. Learning to Execute http://arxiv.org/abs/1410.4615
   (input : a character-by-charter description of a Python program, output : execution results of code !!!)
4. .....

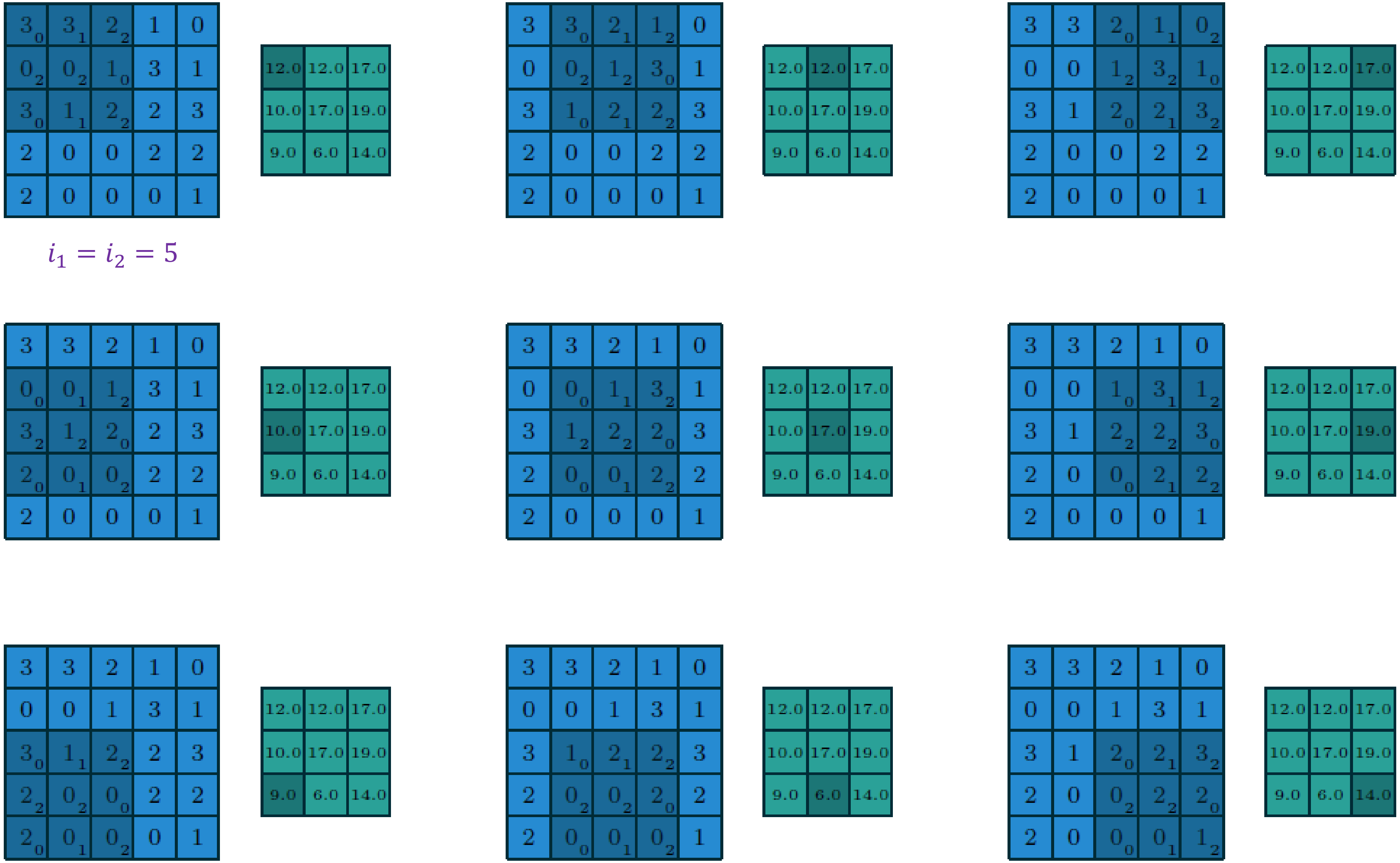
One challenge affecting RNNs is that early models turned out to be very difficult to train, harder even than deep feedforward networks. (Gradient vanishing or exploding) → LSTM

Discrete convolution : single feature map in-out

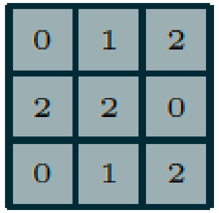
i_j: *input size along axis j*

k_j: *kernel size along axis j*

N: *dimension of kernel*



$i_1 = i_2 = 5$



$k_1 = k_2 = 3$

$N = 2$

Discrete convolution : single feature map in-out

i_j: *input size along axis j*

k_j: *kernel size along axis j*

N: *dimension of kernel*

s_j: *stride (distance between consecutive positions of the kernel) along axis j*

p_j: *zero padding (number of zeros concatenated at the beginning and the end of an axis) along axis j*

0	1	2
2	2	0
0	1	2

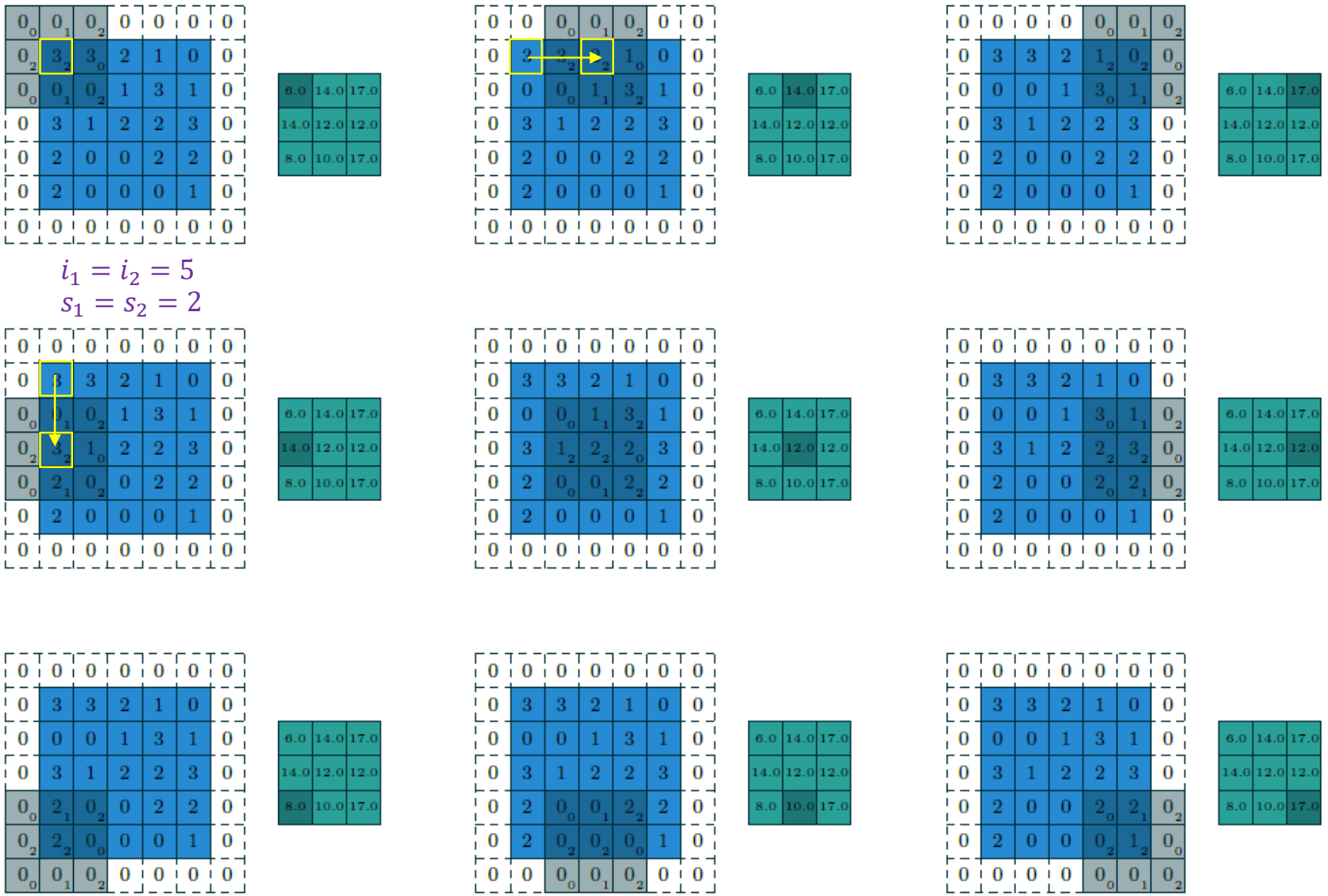
$k_1 = k_2 = 3$

$N = 2$

$p_1 = p_2 = 1$

$i_1 = i_2 = 5$

$s_1 = s_2 = 2$



Discrete convolution : multi feature map in-out

i_j: *input size along axis* j

k_j: *kernel size along axis* j

N: *dimension of kernel*

s_j: *stride (distance between consecutive positions of the kernel) along axis* j

p_j: *zero padding (number of zeros concatenated at the beginning and the end of an axis) along axis* j

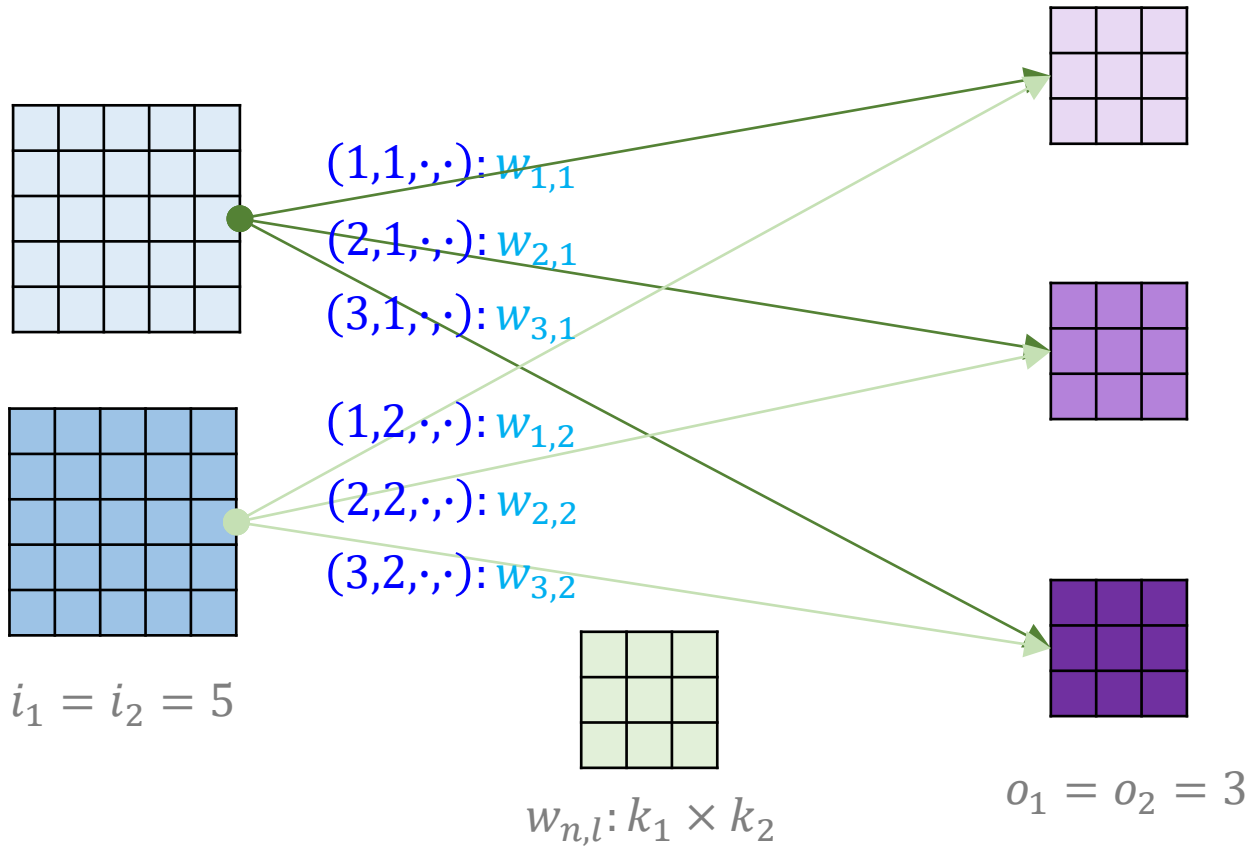
Kernel define $(n, m, k_1, \cdots, k_N)$

$n \equiv$ *number of output feature maps*

$m \equiv$ *number of input feature maps*

Interpretation via 2D kernels

(3,2,3,3) There are 3 x 2=6 kernels with the size of 3 by 3



Discrete convolution : multi feature map in-out

i_j: *input size along axis* j

k_j: *kernel size along axis* j

N: *dimension of kernel*

s_j: *stride (distance between consecutive positions of the kernel) along axis* j

p_j: *zero padding (number of zeros concatenated at the beginning and the end of an axis) along axis* j

Kernel define $(n, m, k_1, \cdots, k_N)$

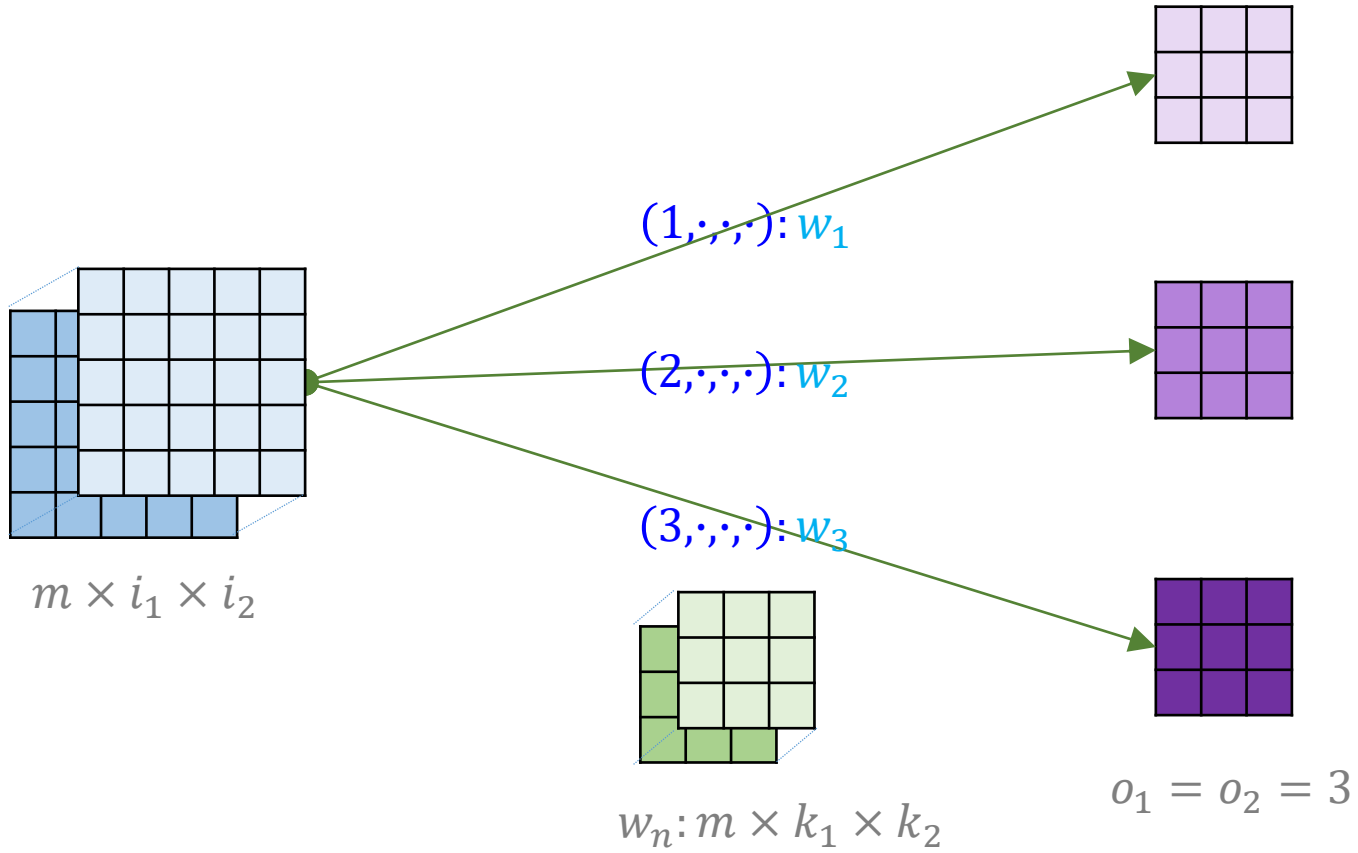
$n \equiv$ *number of output feature maps*

$m \equiv$ *number of input feature maps*

Interpretation via 3D kernels

$(3,2,3,3)$

There are **3** kernels with the size of **2** by **3** by **3**

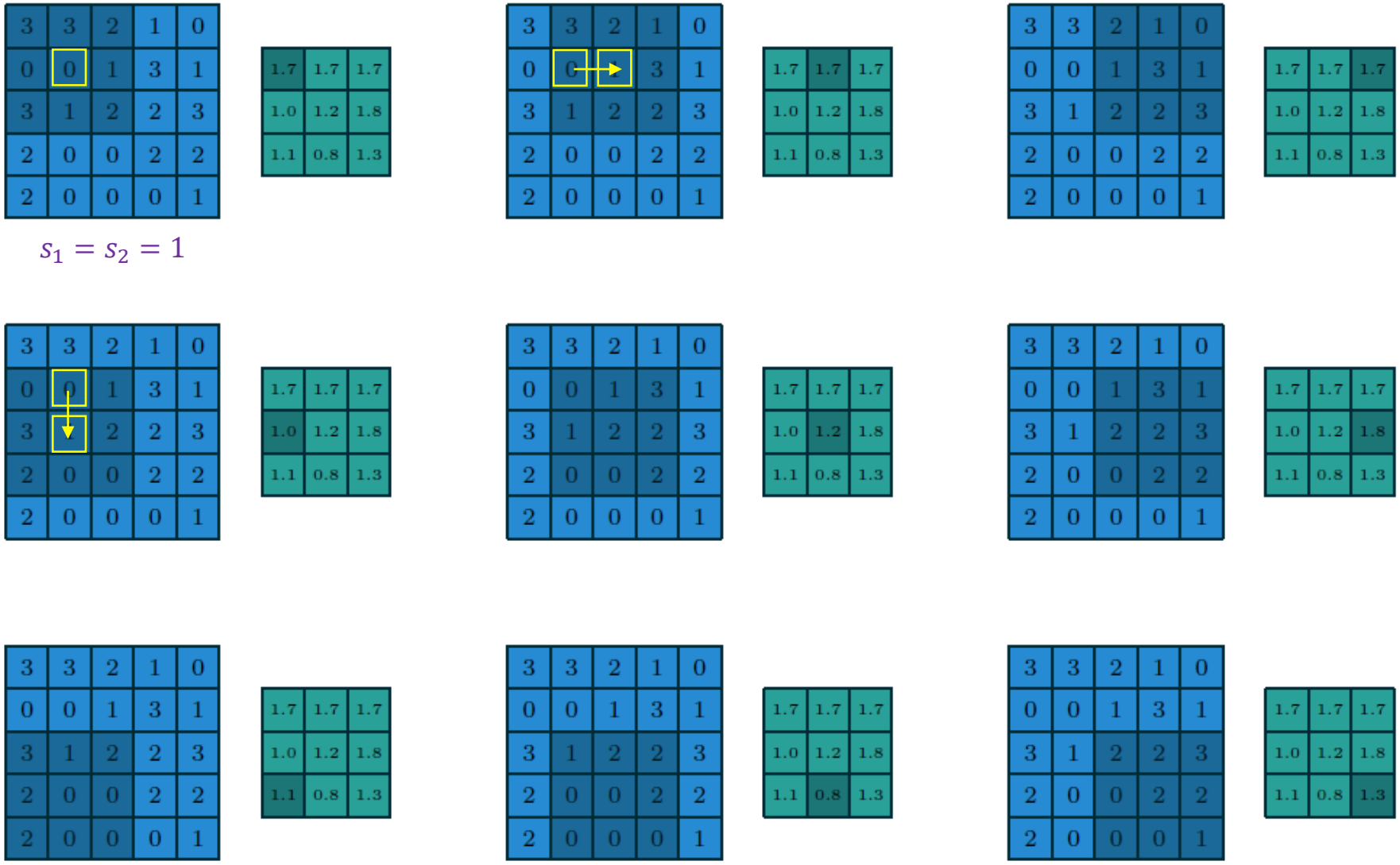


Pooling – Average pooling

i_j: *input size along axis* j

k_j: *pooling window size along axis* j

s_j: *stride (distance between consecutive positions of the kernel) along axis* j

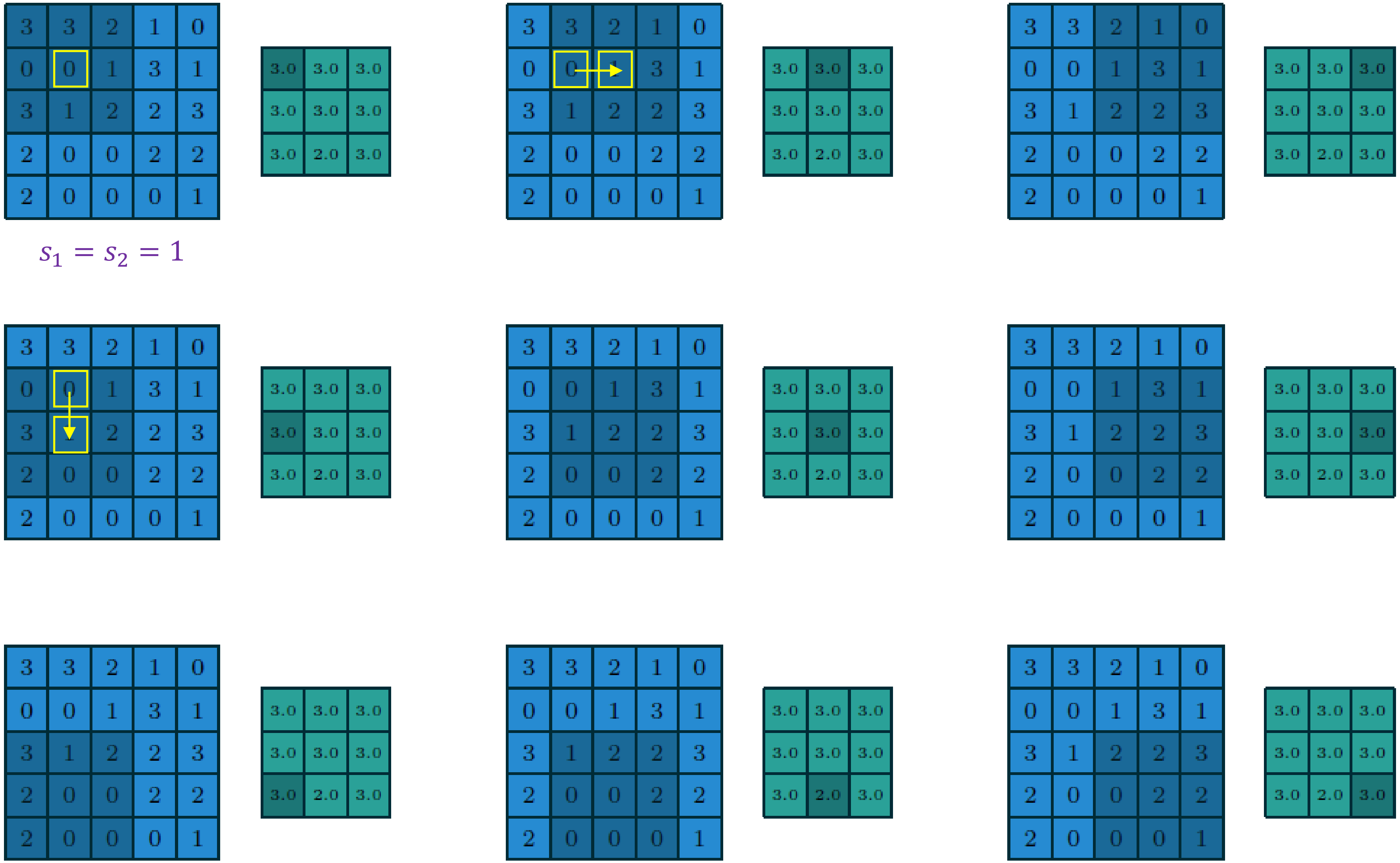


Pooling – Max pooling

i_j: input size along axis j

k_j: pooling window size along axis j

s_j: stride (distance between consecutive positions of the kernel) along axis j



Convolution arithmetic – No zero padding, unit strides

Relationship 1. No zero padding, unit strides

For any i and k, and for $s = 1$ and $p = 0$,

$$o = (i - k) + 1$$

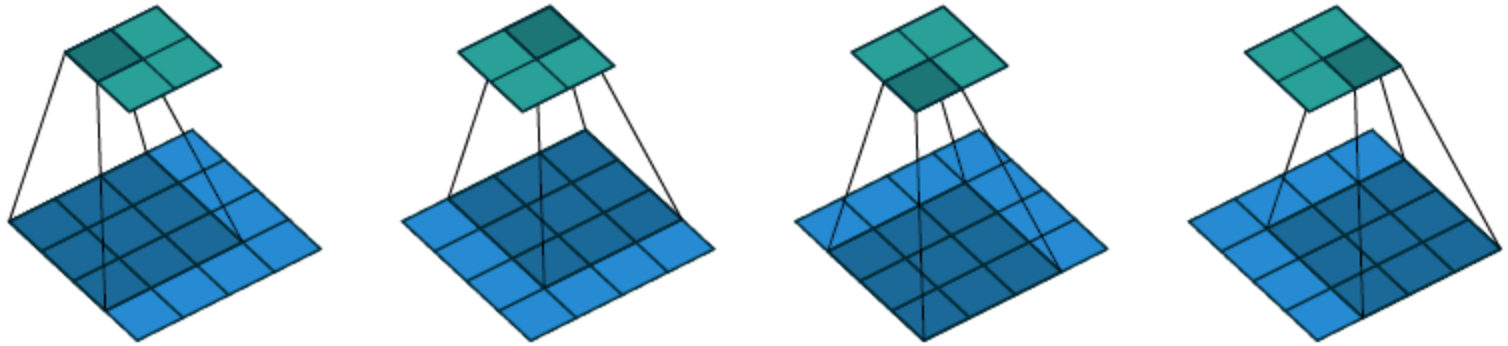


Figure 2.1: (No padding, no strides) Convolving a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$).

Convolution arithmetic – Zero padding, unit strides

Relationship 2. Zero padding, unit strides

For any i and k, and p, and for $s = 1$,

$$o = (i - k) + 2p + 1$$

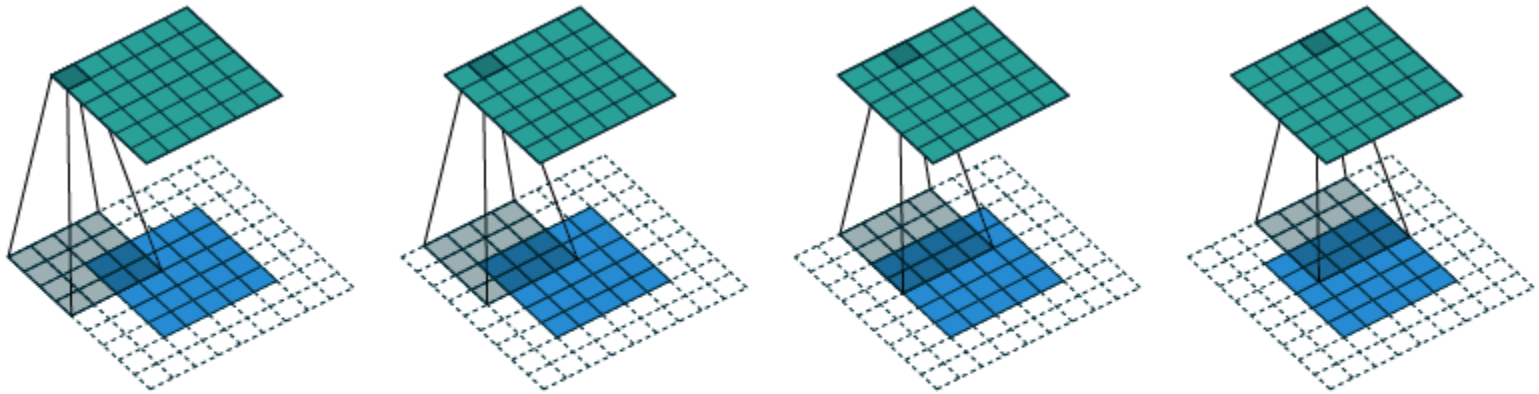


Figure 2.2: (Arbitrary padding, no strides) Convolving a 4×4 kernel over a 5×5 input padded with a 2×2 border of zeros using unit strides (i.e., $i = 5$, $k = 4$, $s = 1$ and $p = 2$).

Convolution arithmetic – Half (same) padding, unit strides

Relationship 3. Half padding, unit strides

$$For\ any\ i\ and\ for\ k\ odd\ (k = 2n+1, n \in \mathrm{N}), s = 1\ \ and\ p = \lfloor k/2 \rfloor = n,$$
$$o = i + 2\lfloor k/2 \rfloor - (k-1)$$
$$= i + 2n - 2n$$
$$= i$$

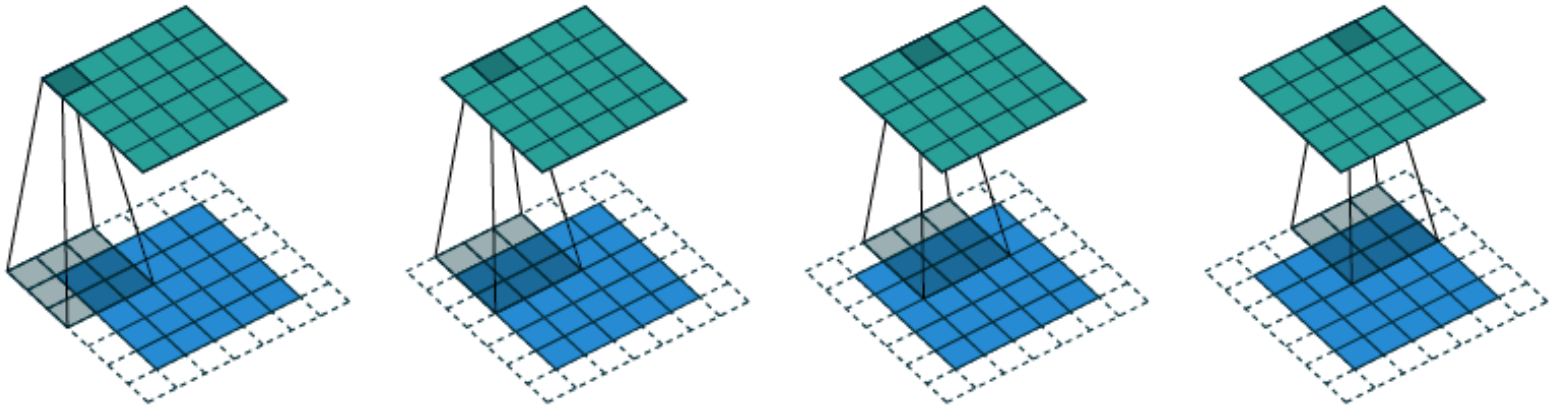


Figure 2.3: (Half padding, no strides) Convolving a 3×3 kernel over a 5×5 input using half padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 1$).

Convolution arithmetic – Full padding, unit strides

Relationship 4. Full padding, unit strides

For any i and for k, and for $p = k - 1$ and $s = 1$,

$$o = i + 2(k-1) - (k-1)$$
$$= i + (k-1)$$

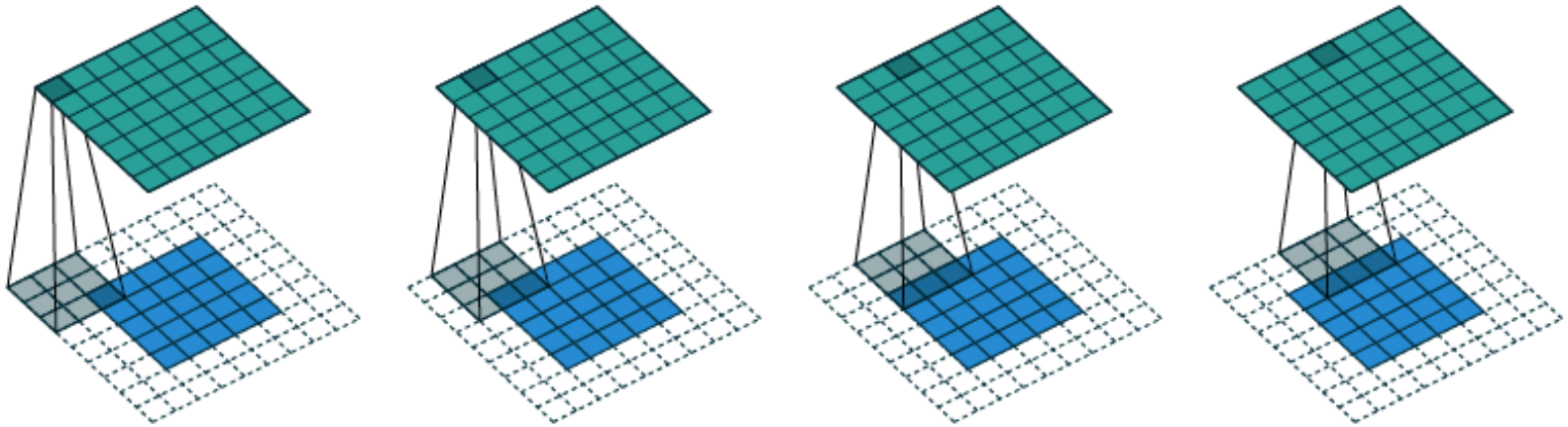


Figure 2.4: (Full padding, no strides) Convolving a 3×3 kernel over a 5×5 input using full padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 2$).

Convolution arithmetic – No zero padding, non-unit strides

Relationship 5. No zero padding, non-unit strides

For any i , k, *and* s *for* $p = 0$,

$$o = \left\lfloor \frac{i - k}{s} \right\rfloor + 1$$

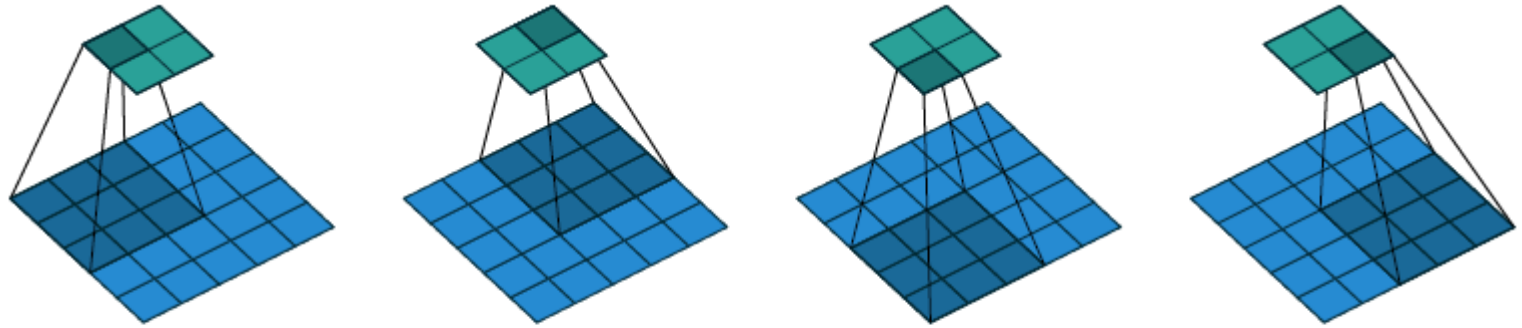


Figure 2.5: (No zero padding, arbitrary strides) Convolving a 3×3 kernel over a 5×5 input using 2×2 strides (i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 0$).

Convolution arithmetic – Zero padding, non-unit strides

Relationship 6. Zero padding, non-unit strides

For any i , k, p *and* s,

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

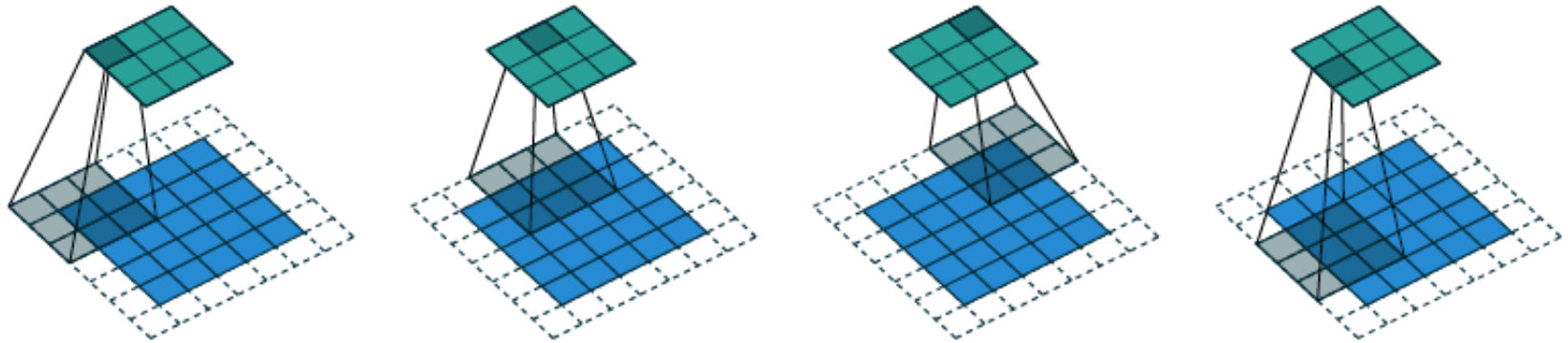


Figure 2.6: (Arbitrary padding and strides) Convolving a 3×3 kernel over a 5×5 input padded with a 1×1 border of zeros using 2×2 strides (i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 1$).

Convolution arithmetic – Zero padding, non-unit strides

Relationship 6. Zero padding, non-unit strides

For any i , k, p *and* s,

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

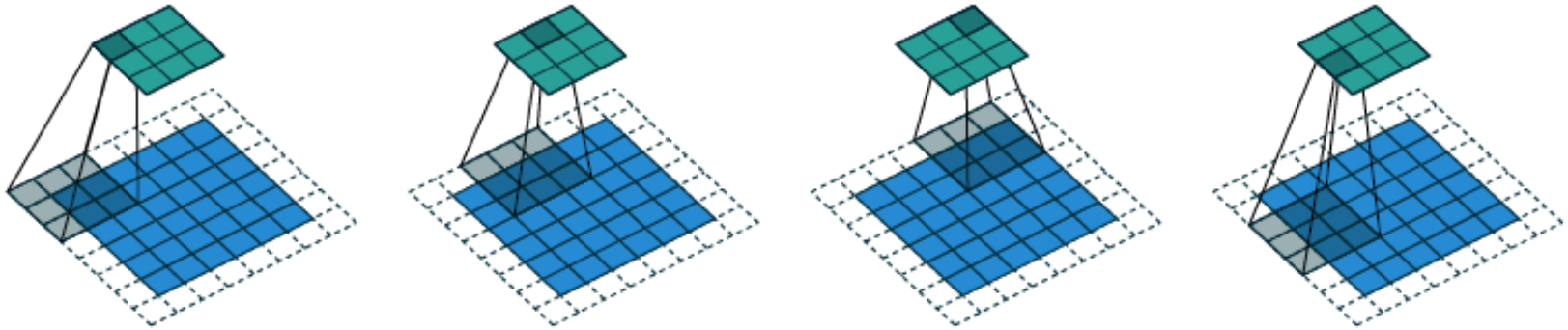


Figure 2.7: (Arbitrary padding and strides) Convolving a 3×3 kernel over a 6×6 input padded with a 1×1 border of zeros using 2×2 strides (i.e., $i = 6$, $k = 3$, $s = 2$ and $p = 1$). In this case, the bottom row and right column of the zero padded input are not covered by the kernel.

Pooling arithmetic

Relationship 7. Pooling arithmetic

For any i , k and s,

$$o = \left\lfloor \frac{i - k}{s} \right\rfloor + 1$$

Transposed convolution – Convolution as a matrix

Example

Relationship 1. No zero padding, unit strides

For any i and k, and for $s = 1$ and $p = 0$,

$$o = (i - k) + 1$$

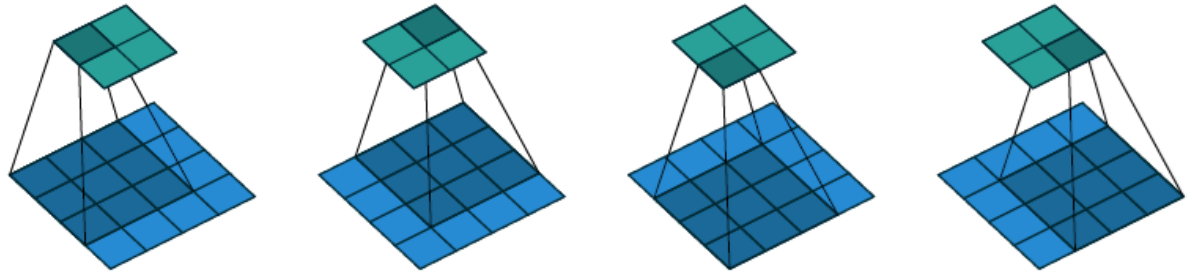


Figure 2.1: (No padding, no strides) Convolving a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$).

Convolution as a matrix

Kernel: 3x3

Input: 4x4 → Column vector: 16 x 1

$$\begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix} \cdot \begin{pmatrix} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \\ \\ \\ \end{pmatrix}$$

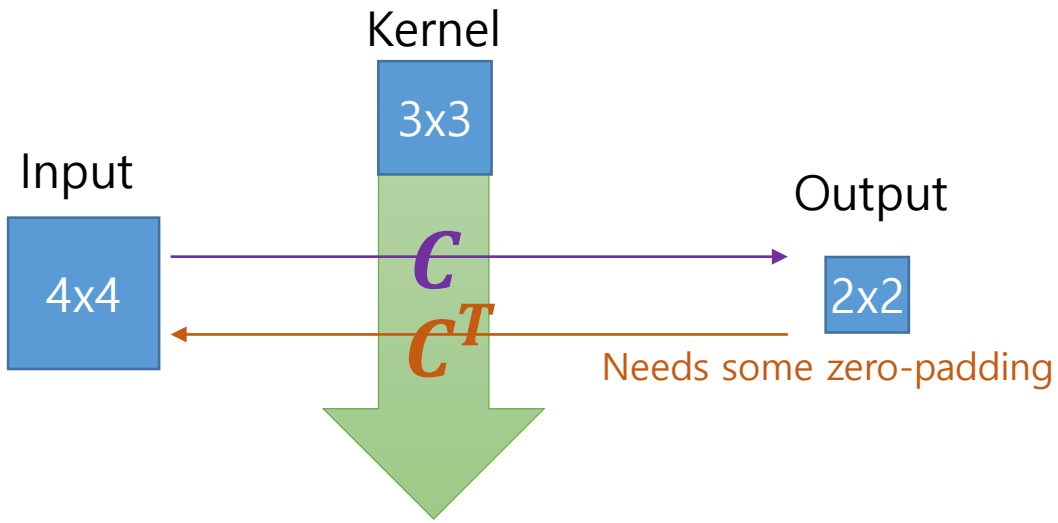
Column vector: 4 x 1 → Output: 2x2

C : Sparse matrix (4 x 16)

C : Forward prediction ↔ $\mathbf{C}^T$: Error backpropagation

Transposed convolution– No zero padding, unit strides

Example



Relationship 8. No zero padding, unit strides, transposed

A convolution described by $s = 1, p = 0$ *and* k *has an associated transposed convolution described by* $k' = k, s' = s,$ *and* $p' = k - 1$ *and its output size is*

$$o' = i' + (k - 1)$$

$o = i'$
$i = o'$

Relationship 1. No zero padding, unit strides

For any i *and* $k,$ *and for* $s = 1$ *and* $p = 0,$

$$o = (i - k) + 1$$

Transposed convolution– No zero padding, unit strides

Relationship 8. No zero padding, unit strides, transposed

A convolution described by $s = 1, p = 0$ *and* k *has an associated transposed convolution described by* $k' = k, s' = s,$ *and* $p' = k - 1$ *and its output size is*

$$o' = i' + (k - 1)$$

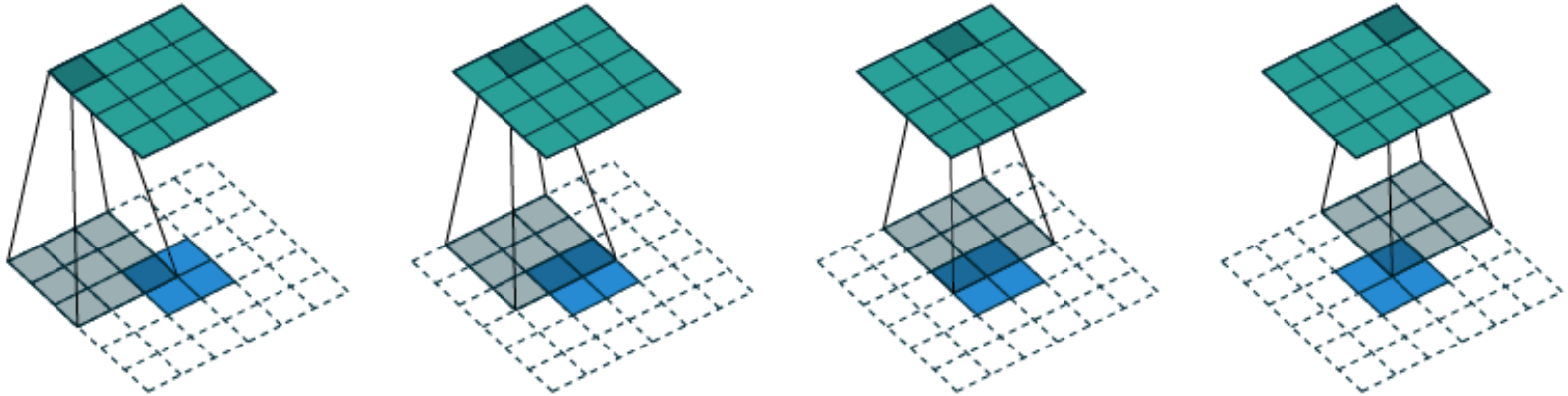


Figure 4.1: The transpose of convolving a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$). It is equivalent to convolving a 3×3 kernel over a 2×2 input padded with a 2×2 border of zeros using unit strides (i.e., $i' = 2$, $k' = k$, $s' = 1$ and $p' = 2$).

Transposed convolution– Zero padding, unit strides

Relationship 9. Zero padding, unit strides, transposed

A convolution described by $s = 1, k$ *and* p *has an associated transposed convolution described by* $k' = k, s' = s,$ *and* $p' = k - p - 1$ *and its output size is*

$$o' = i' + (k - 1) - 2p$$

$o = i'$
$i = o'$

Relationship 2. Zero padding, unit strides

For any i *and* $k,$ *and* $p,$ *and for* $s = 1$ *and* $p = 0,$

$$o = (i - k) + 2p + 1$$

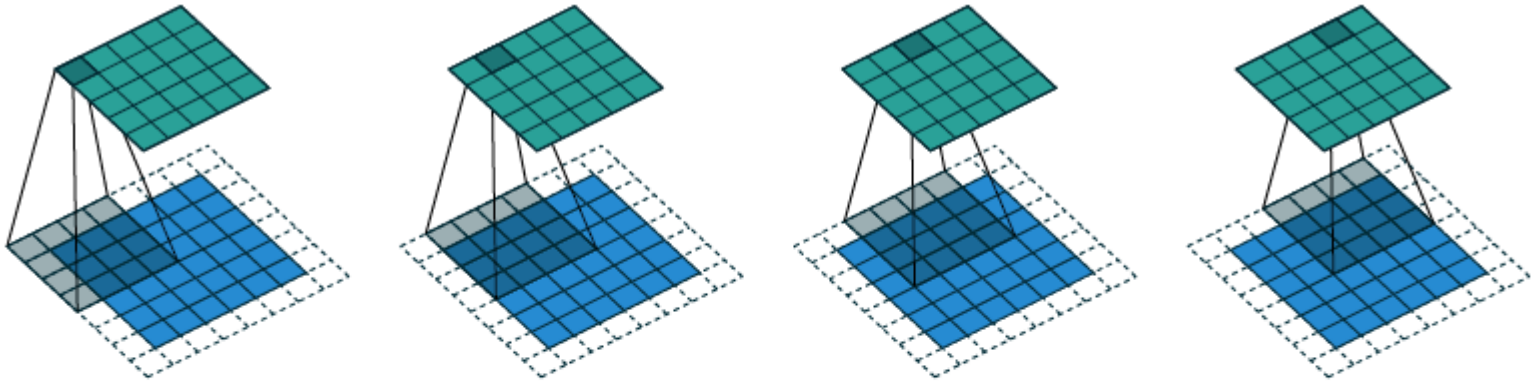


Figure 4.2: The transpose of convolving a 4×4 kernel over a 5×5 input padded with a 2×2 border of zeros using unit strides (i.e., $i = 5$, $k = 4$, $s = 1$ and $p = 2$). It is equivalent to convolving a 4×4 kernel over a 6×6 input padded with a 1×1 border of zeros using unit strides (i.e., $i' = 6$, $k' = k$, $s' = 1$ and $p' = 1$).

Transposed convolution– Half (same) padding, unit strides

Relationship 10. Half padding, unit strides, transposed

A convolution described by $k = 2n + 1, n \in \mathrm{N}, s = 1$ *and* $p = \lfloor k/2 \rfloor = n$ *has an associated transposed convolution described by* $k' = k, s' = s,$ *and* $p' = p$ *and its output size is*

$$o' = i' + (k - 1) - 2p$$
$$= i' + 2n - 2n$$
$$= i'$$

$o = i'$
$i = o'$

Relationship 3. Half padding, unit strides

For any i *and for* k *odd* $(k = 2n + 1, n \in \mathrm{N}), s = 1$ *and* $p = \lfloor k/2 \rfloor = n,$

$$o = i$$

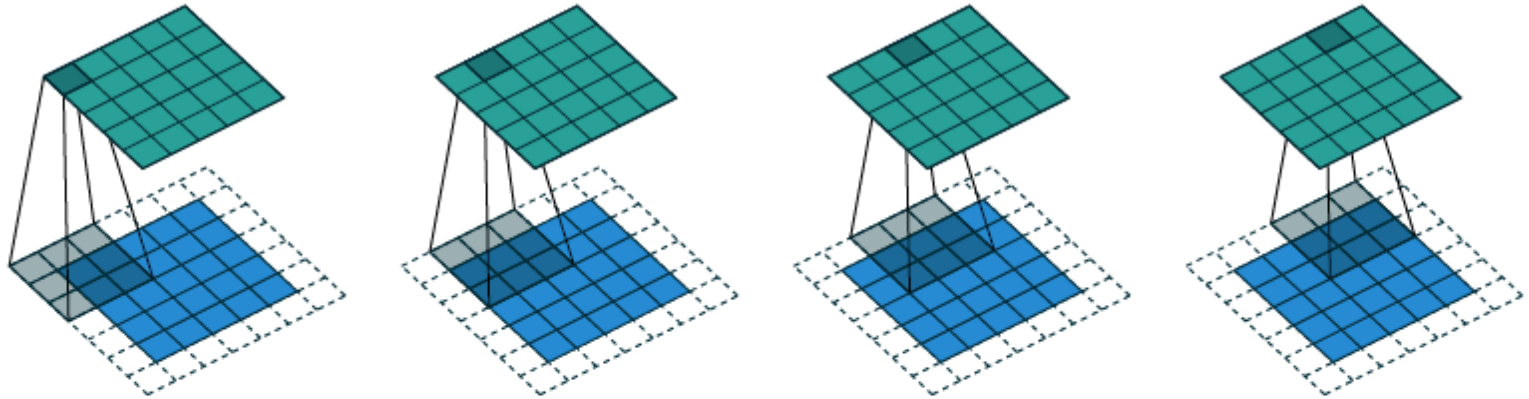


Figure 4.3: The transpose of convolving a 3×3 kernel over a 5×5 input using half padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 1$). It is equivalent to convolving a 3×3 kernel over a 5×5 input using half padding and unit strides (i.e., $i' = 5$, $k' = k$, $s' = 1$ and $p' = 1$).

Transposed convolution– Full padding, unit strides

Relationship 11. Full padding, unit strides, transposed

A convolution described by $s = 1, k$ *and* $p = k - 1$ *has an associated transposed convolution described by* $k' = k, s' = s,$ *and* $p' = 0$ *and its output size is*

$$o' = i' + (k - 1) - 2p$$
$$= i' - (k - 1)$$

$o = i'$
$i = o'$

Relationship 4. Full padding, unit strides

For any i *and for* $k,$ *and for* $p = k - 1$ *and* $s = 1,$

$$o = i + 2(k - 1) - (k - 1)$$
$$= i + (k - 1)$$

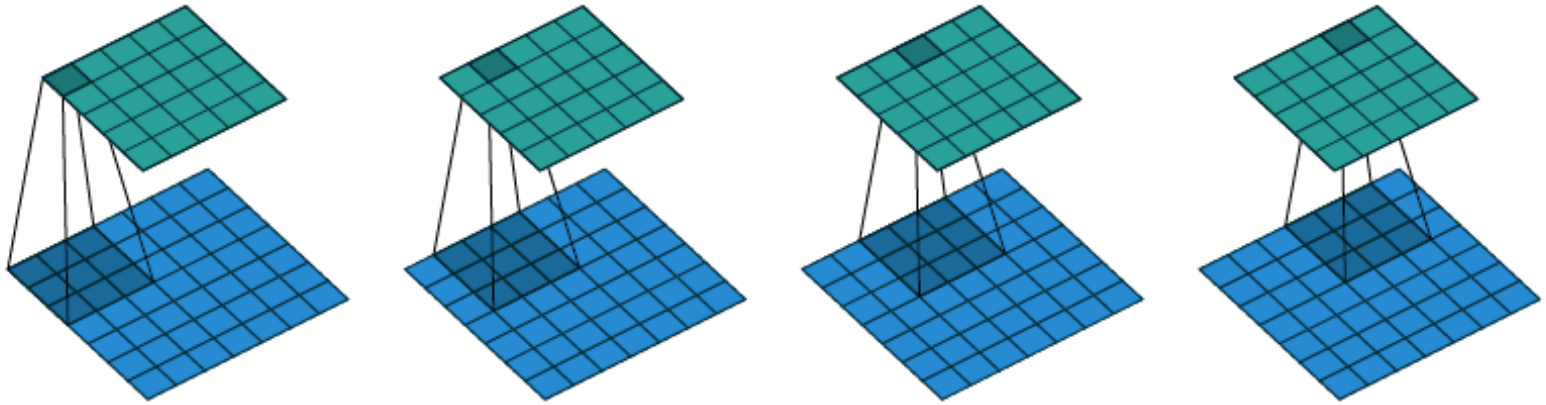


Figure 4.4: The transpose of convolving a 3×3 kernel over a 5×5 input using full padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 2$). It is equivalent to convolving a 3×3 kernel over a 7×7 input using unit strides (i.e., $i' = 7$, $k' = k$, $s' = 1$ and $p' = 0$).

Transposed convolution– No zero padding, non-unit strides

Relationship 12. No zero padding, non-unit strides, transposed

A convolution described by $p = 0, k$ and s and whose input size is such that $i - k$ is a multiple of s has an associated transposed convolution described by $\tilde{i}'$, $k' = k$, $s' = 1$, and $p' = k - 1$, where $\tilde{i}'$ is the size of the stretched input obtained by adding $s - 1$ zeros between each input unit, and its output size is

$$o' = s(i' - 1) + k$$

$o = i'$
$i = o'$

Relationship 5. No zero padding, non-unit strides

For any i, k, and s for $p = 0$,

$$o = \left\lfloor \frac{i - k}{s} \right\rfloor + 1$$

That's why transposed convolutions are sometimes called fractionally strided convolutions.

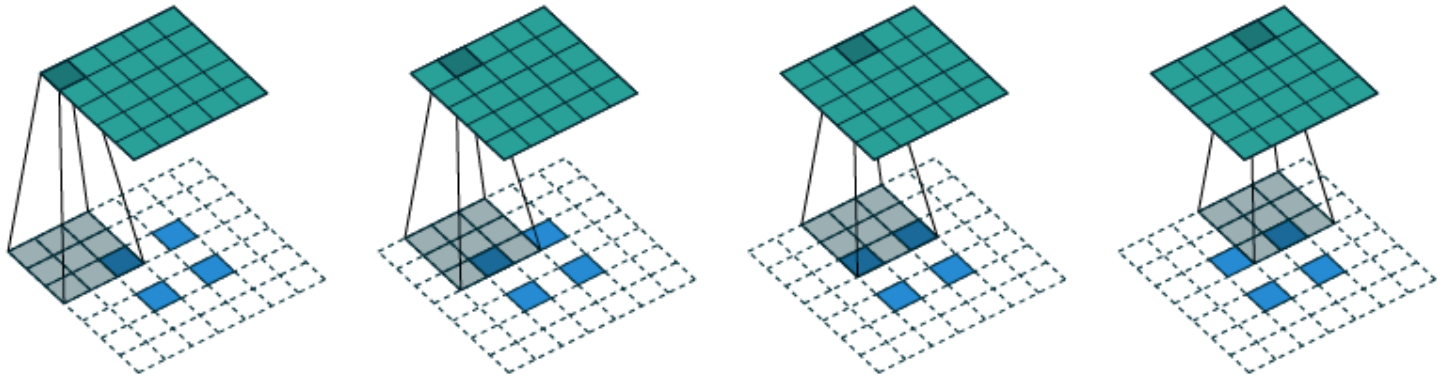


Figure 4.5: The transpose of convolving a 3×3 kernel over a 5×5 input using 2×2 strides (i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 0$). It is equivalent to convolving a 3×3 kernel over a 2×2 input (with 1 zero inserted between inputs) padded with a 2×2 border of zeros using unit strides (i.e., $i' = 2$, $\tilde{i}' = 3$, $k' = k$, $s' = 1$ and $p' = 2$).

Transposed convolution– Zero padding, non-unit strides

Relationship 13. Zero padding, non-unit strides, transposed

A convolution described by k, s and p and whose input size is such that $i + 2p - k$ is a multiple of s has an associated transposed convolution described by $\tilde{i}'$, $k' = k$, $s' = 1$, and $p' = k - p - 1$, where $\tilde{i}'$ is the size of the stretched input obtained by adding $s - 1$ zeros between each input unit, and its output size is

$$o' = s(i' - 1) + k - 2p$$

$o = i'$
$i = o'$

Relationship 6. Zero padding, non-unit strides

For any i, k, p and s,

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

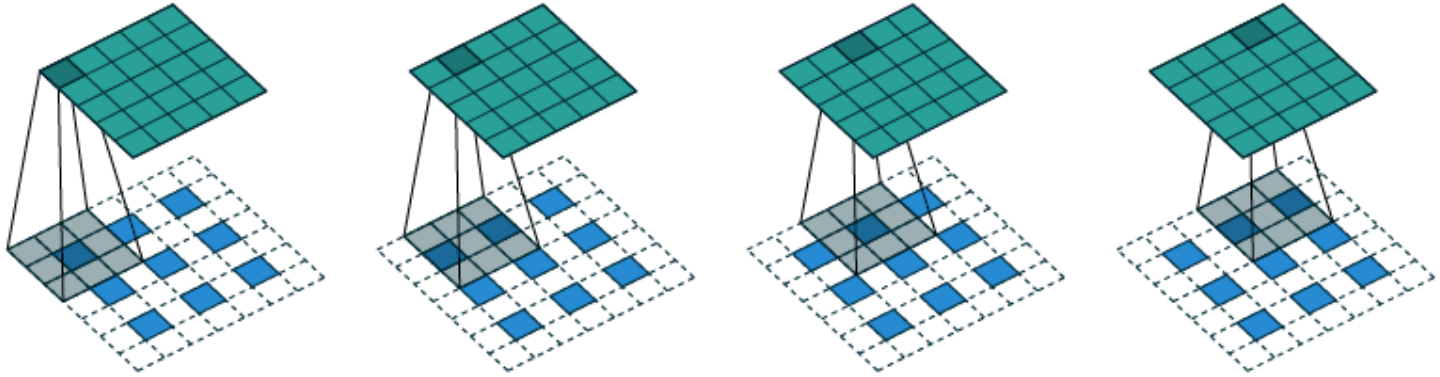


Figure 4.6: The transpose of convolving a 3×3 kernel over a 5×5 input padded with a 1×1 border of zeros using 2×2 strides (i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 1$). It is equivalent to convolving a 3×3 kernel over a 2×2 input (with 1 zero inserted between inputs) padded with a 1×1 border of zeros using unit strides (i.e., $i' = 3$, $\tilde{i}' = 5$, $k' = k$, $s' = 1$ and $p' = 1$).