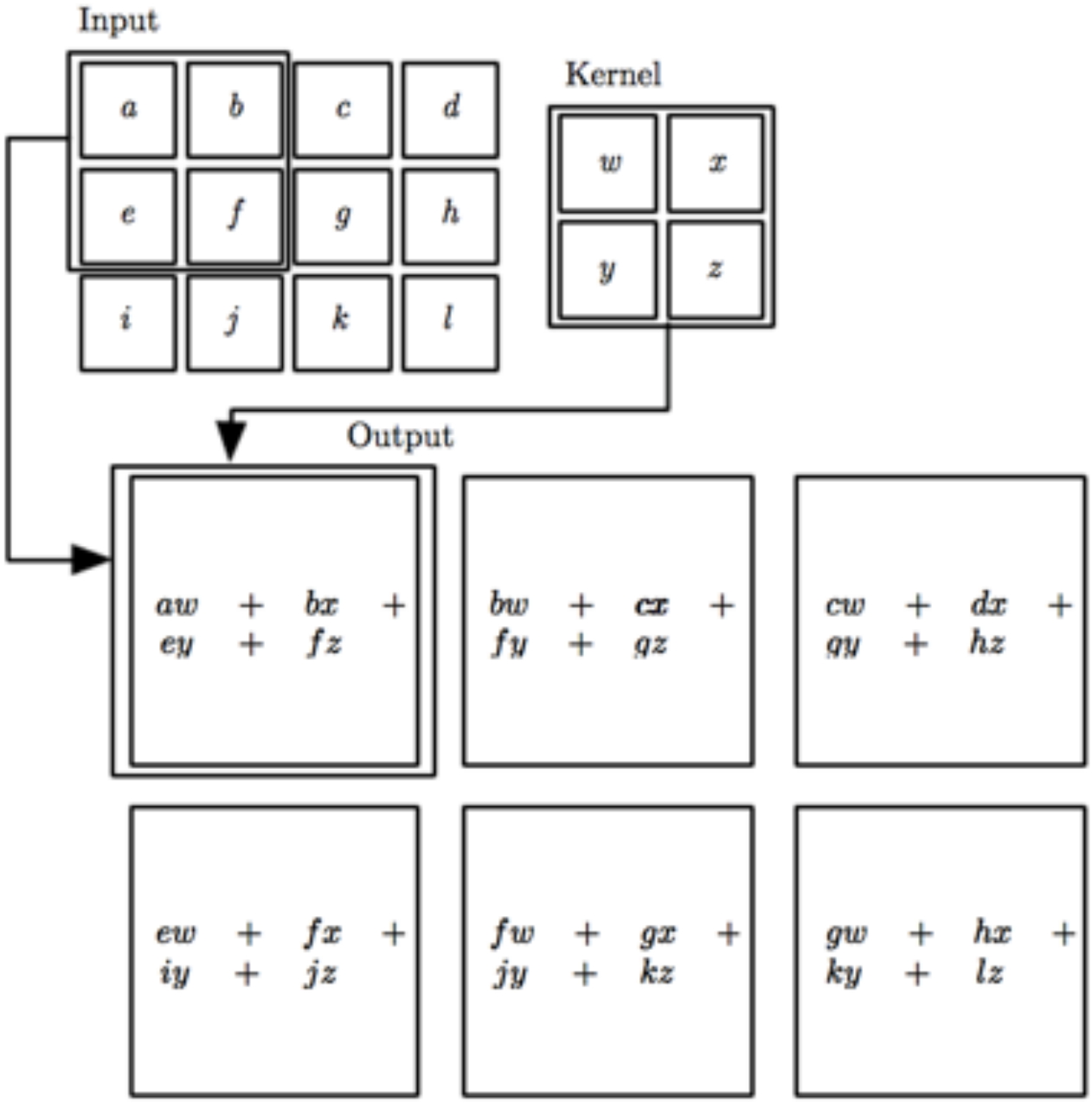
Machine Learning

NN and Deep Learning

Fabio Vandin

December 13^{th}, 2021

Convolution in CNNs (continue)



Convolution: Properties

The use of convolution leads to useful properties:

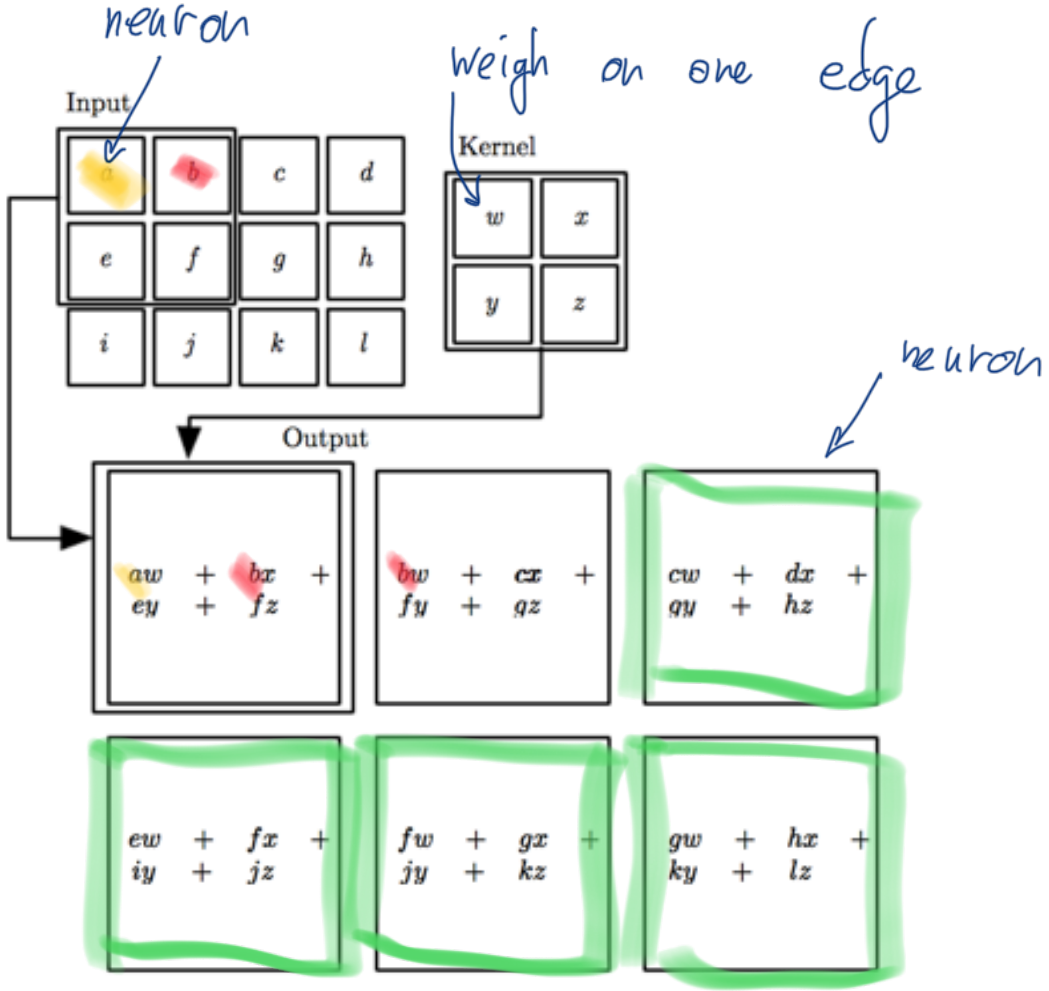
- sparse interactions ⇒ solo piccola porzione di archi possibili
- parameter sharing ⇒ kernel
- equivariant representations ⇒ trovare immagine ovunque in input

Sparse interactions and parameter sharing: reduce the number of parameters ⇒ can be thought as a form of regularization!

Equivariant representations: useful for images (same image/object moved in space)

Convolution: Sparse Interactions

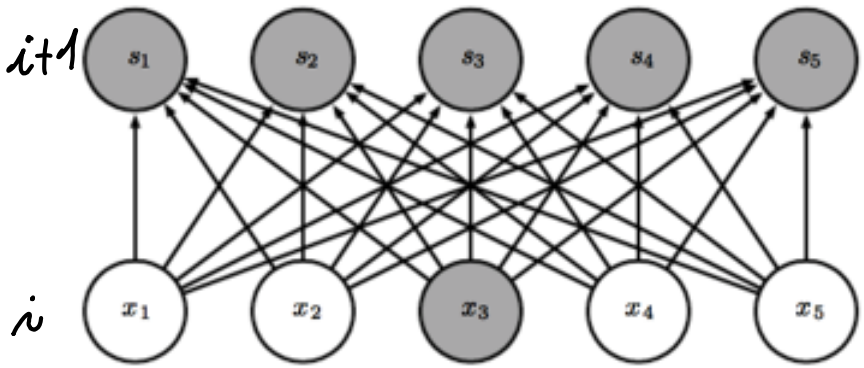
Sparse interactions many edges do not exist in the network = have 0 weight



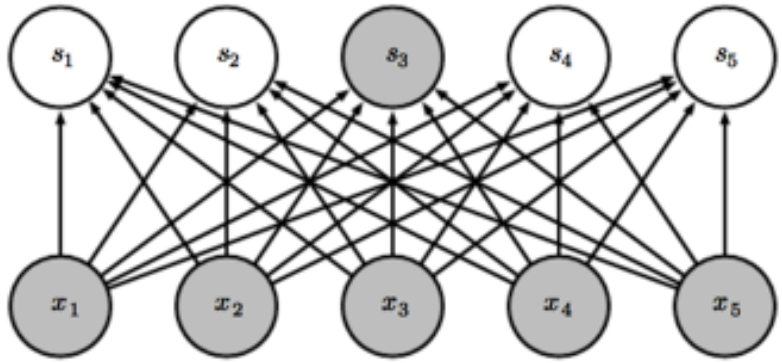
Convolution: Sparse Interactions (continue)

Standard NN

Outgoing edges

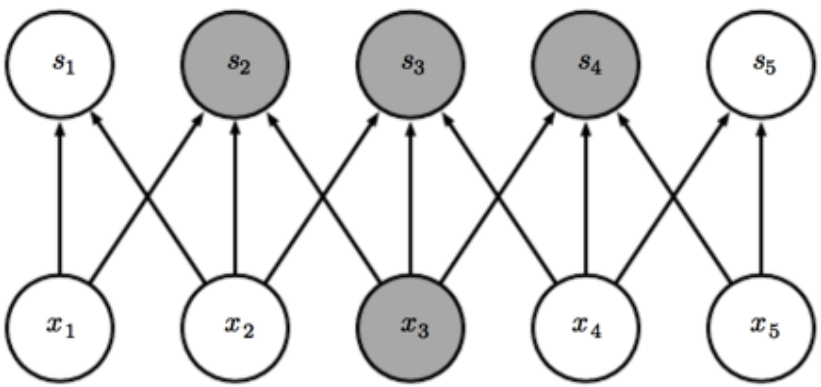


Incoming edges

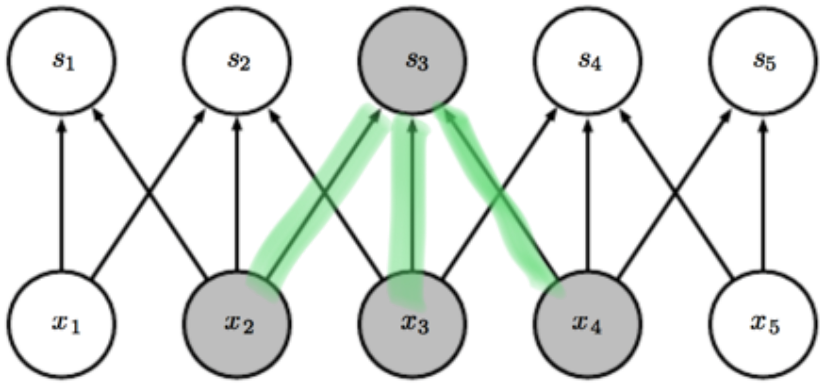


Convolutional NN

Outgoing edges



Incoming edges

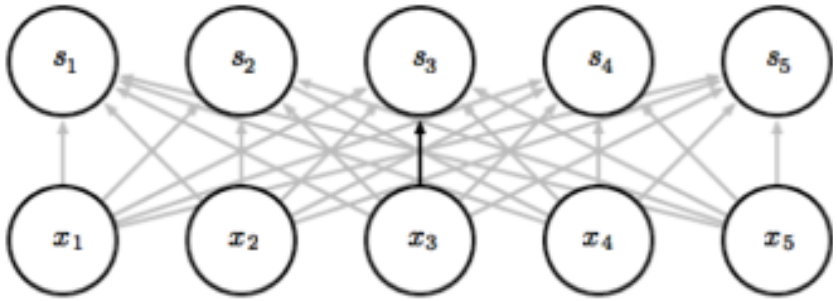


Parameters Sharing

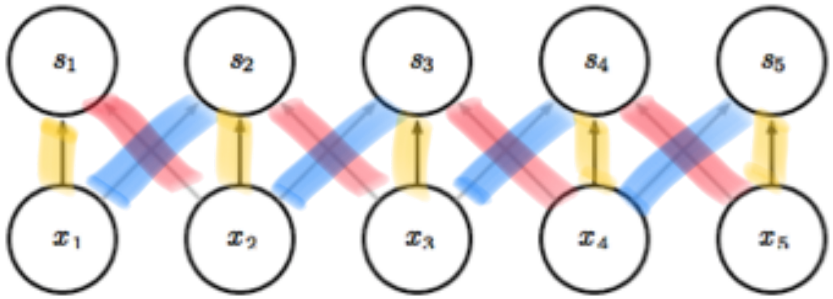
Parameters Sharing = using the same parameter for more than one function in a model

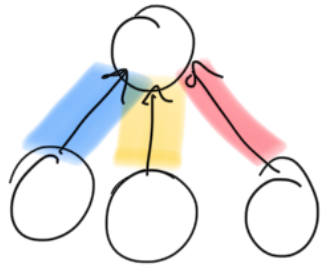
Therefore: rather than learning a separate set of parameters for every input location, we learn only one set kernel come modello

traditional NN



convolutional NN

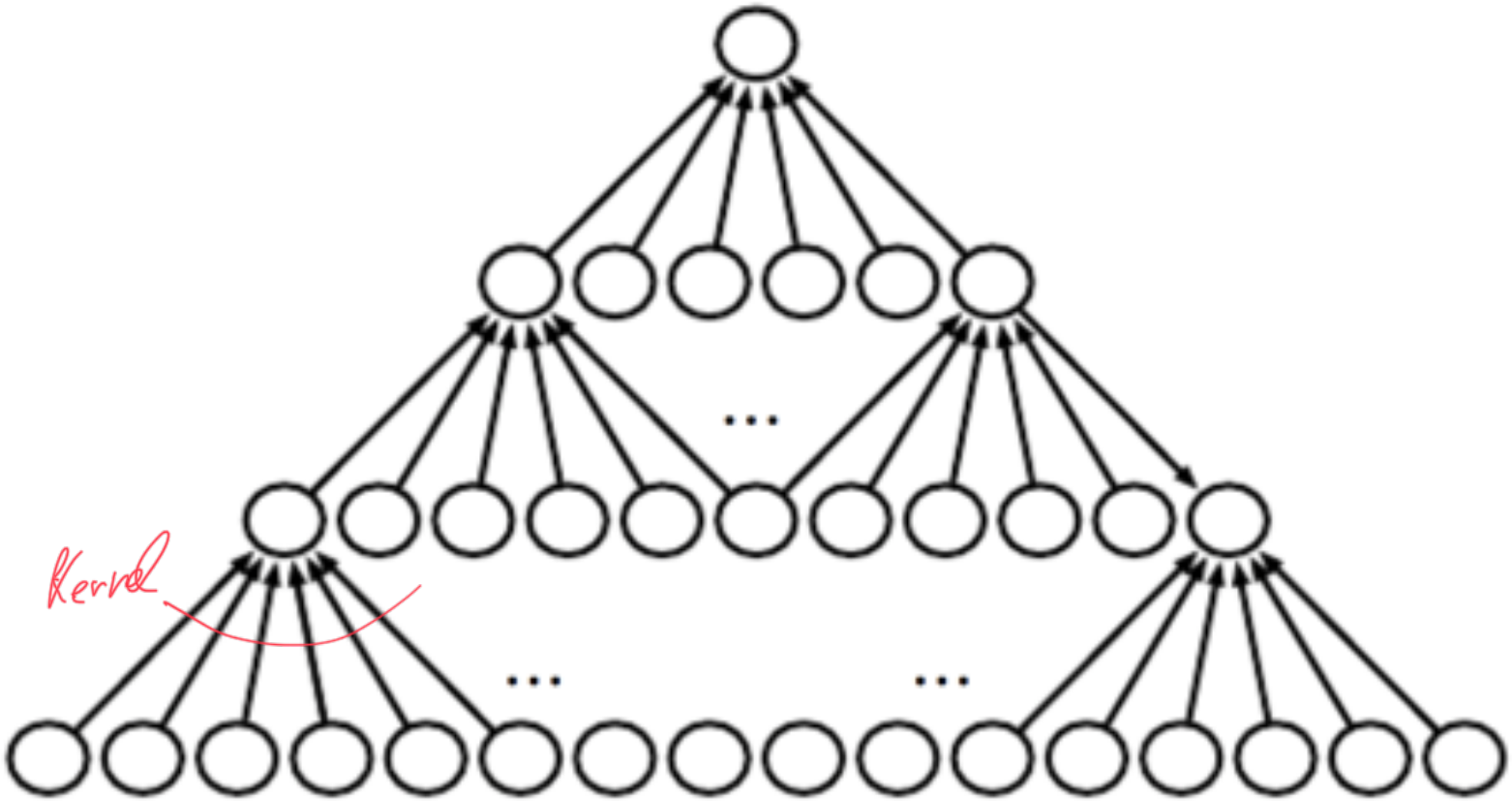




CNNs: Zero Padding

Sometimes want to apply the kernel the same number of times to each input or not to reduce dimension of next layer $\Rightarrow$ need to add extra (virtual) inputs $\Rightarrow$ **zero-padding**

No padding:

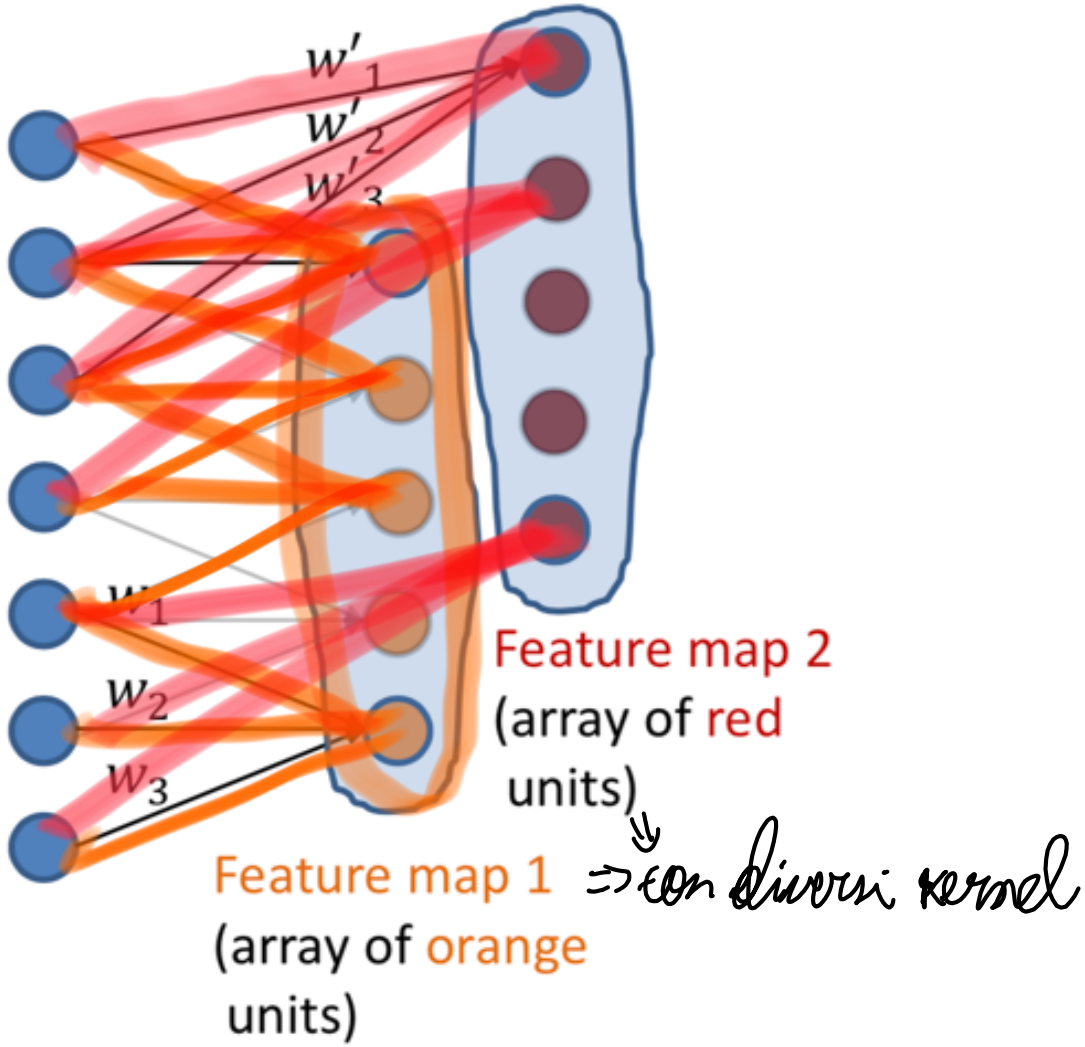


CNNs: Zero Padding

Sometimes want to apply the kernel the same number of times to each input or not to reduce dimension of next layer $\Rightarrow$ need to add extra (virtual) inputs $\Rightarrow$ **zero-padding**

Zero padding: aggiungere (nodi con) 0

Multiple Feature Maps



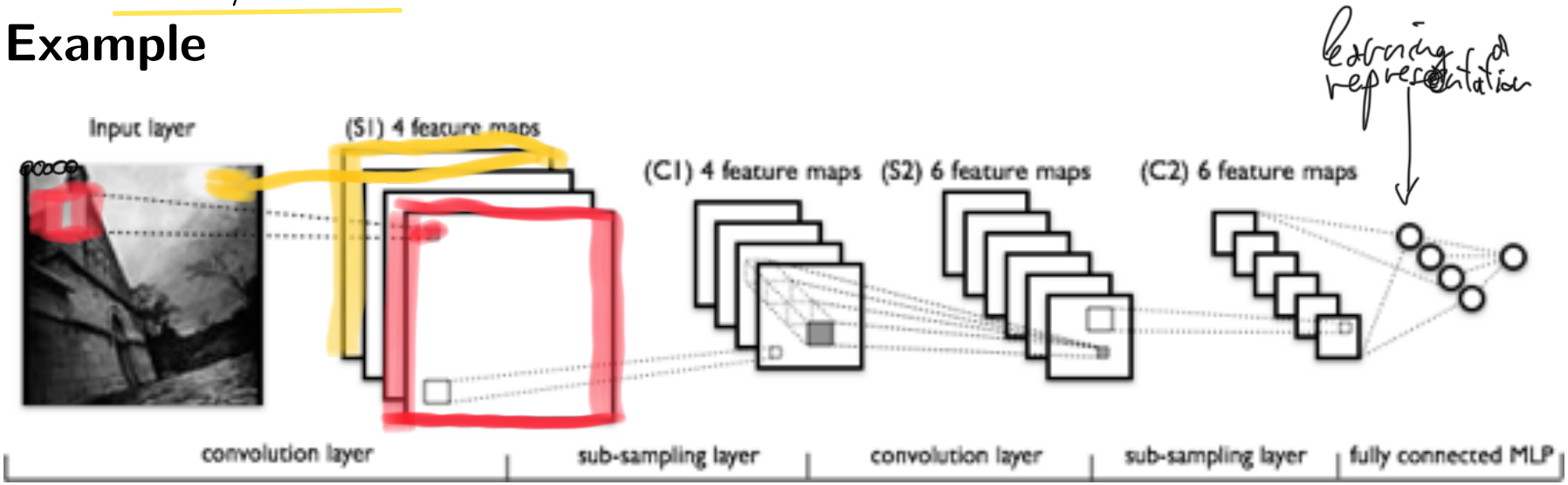
- All **orange units** compute the same function but with a different input windows
- **Orange** and **red** units compute different functions

CNNs: Convolutional Layers

In a CNN:

- there are **convolutional layers** (implement convolution) and other types of layers
- the convolutional layer models consists of several filters/kernels

Example

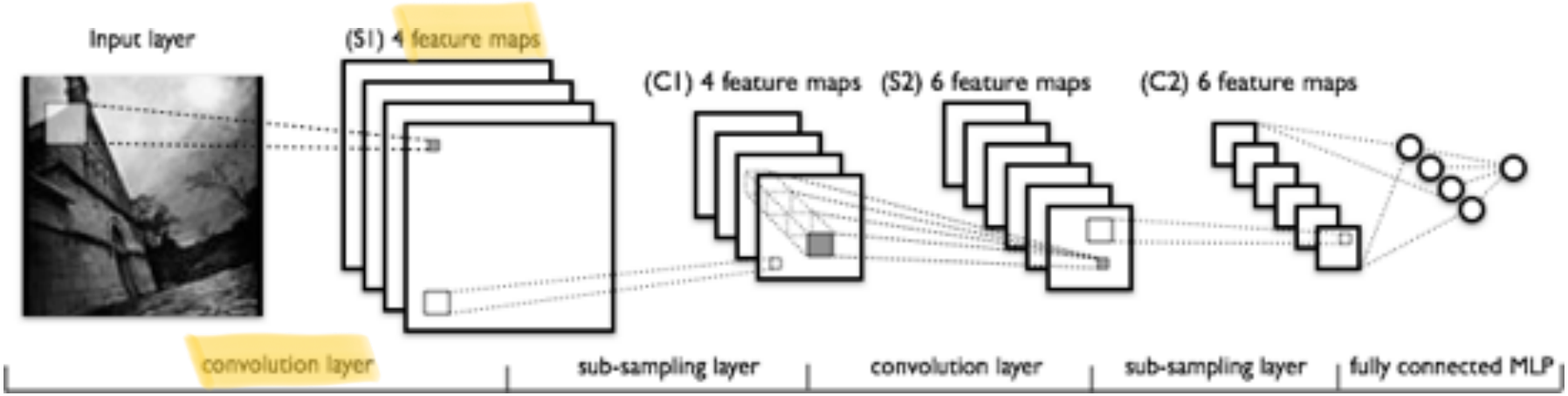


CNNs: Convolutional Layers

In a CNN:

- there are **convolutional layers** (implement convolution) and other types of layers
- the convolutional layer models consists of several filters/kernels

Example



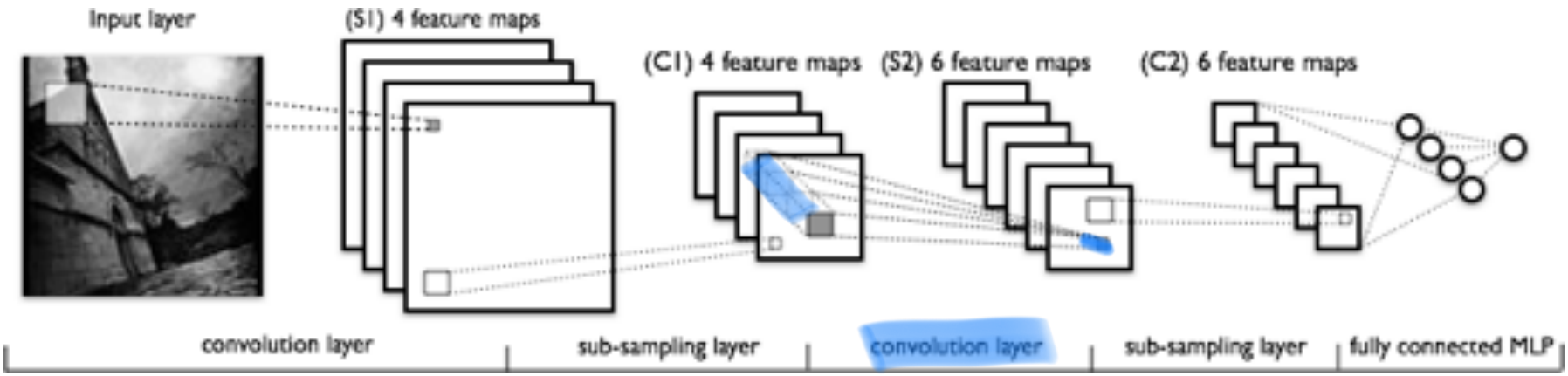
- 4 kernels are learned by the CNN in the first convolutional layer

CNNs: Convolutional Layers

In a CNN:

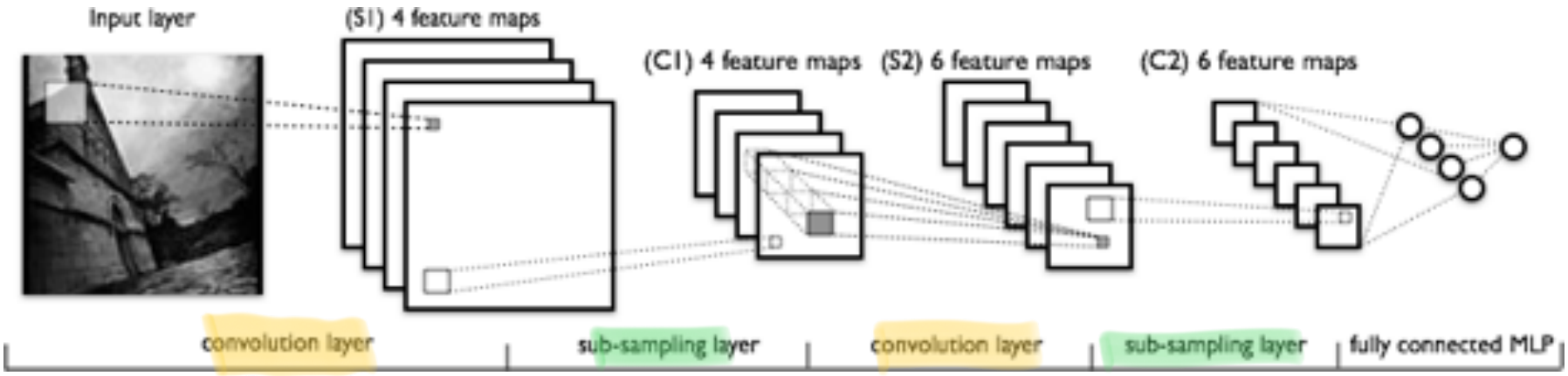
- there are **convolutional layers** (implement convolution) and other types of layers
- the convolutional layer models consists of several filters/kernels

Example



- 4 kernels are learned by the CNN in the first convolutional layer
- 6 kernels are learned by the CNN in the second convolutional layer

CNNs: After Convolution

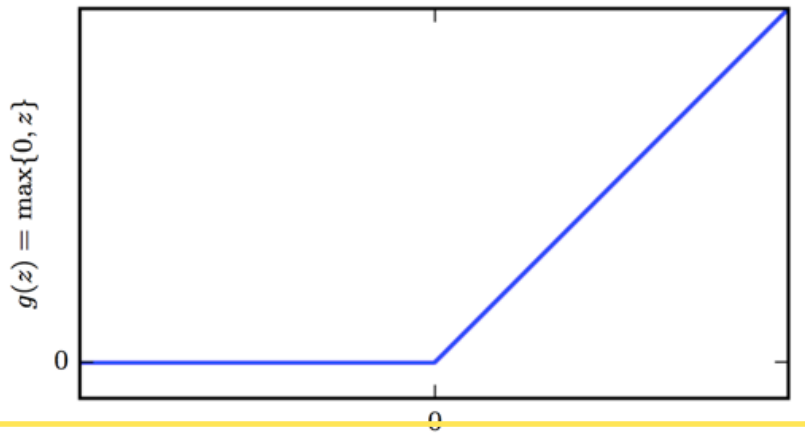


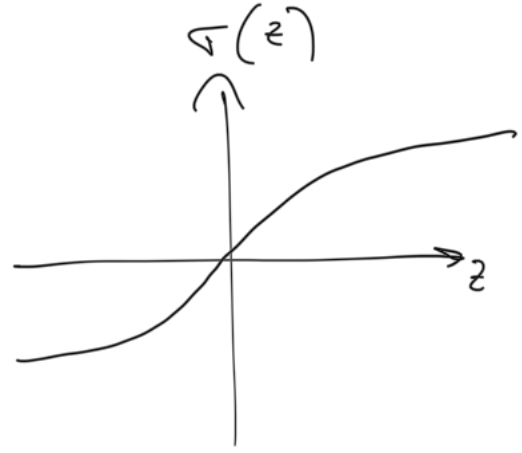
After a convolutional layer:

- nonlinear function: **ReLU (Rectified Linear Unit)**
- **pooling** layer, often combined with **subsampling**

ReLU (Rectified Linear Unit)

$$\sigma(z) = \max\{0, z\}$$

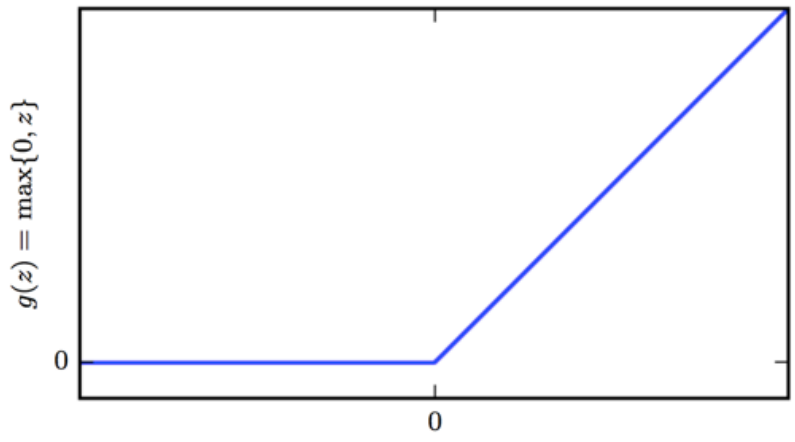




- helps convergence (almost linear)
- does not hurt expressiveness
- prevents the *vanishing gradient function* (gradient for sigmoid o threshold functions is ≈ 0 when weights have large absolute value $\Rightarrow$ weights do not change)
- most common activation function for deep NNs

ReLU (Rectified Linear Unit)

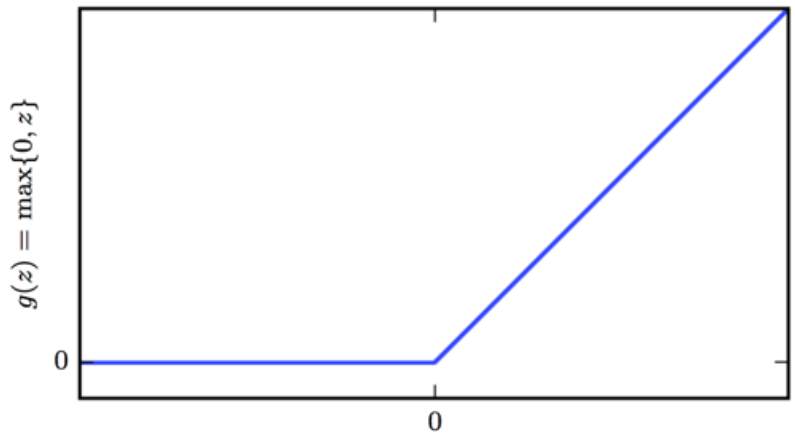
$$\sigma(z) = \max\{0, z\}$$



- helps convergence (almost linear)
- does not hurt expressiveness
- prevents the *vanishing gradient function* (gradient for sigmoid o threshold functions is ≈ 0 when weights have large absolute value $\Rightarrow$ weights do not change)
- most common activation function for deep NNs
- extremely common for CNNs

ReLU (Rectified Linear Unit)

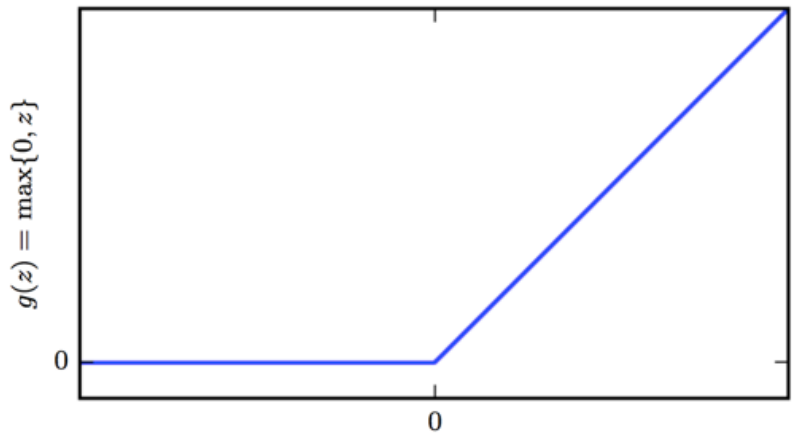
$$\sigma(z) = \max\{0, z\}$$



- helps convergence (almost linear)
- does not hurt expressiveness
- prevents the *vanishing gradient function* (gradient for sigmoid o threshold functions is ≈ 0 when weights have large absolute value $\Rightarrow$ weights do not change)
- most common activation function for deep NNs
- extremely common for CNNs
- leads to sparse activation

ReLU (Rectified Linear Unit)

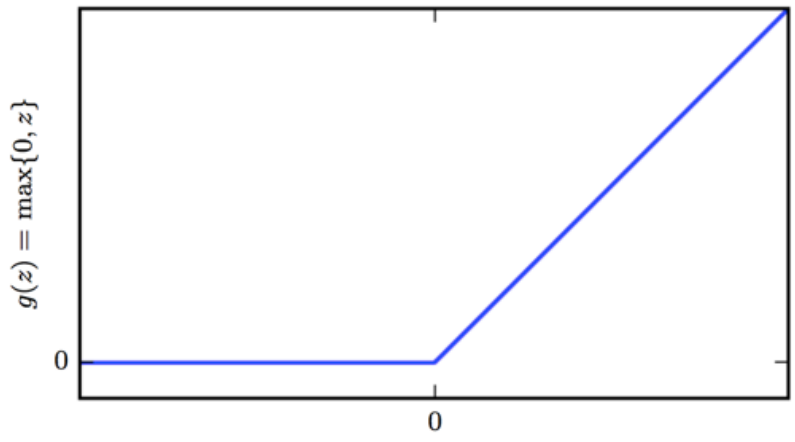
$$\sigma(z) = \max\{0, z\}$$



- helps convergence (almost linear)
- does not hurt expressiveness
- prevents the *vanishing gradient function* (gradient for sigmoid o threshold functions is ≈ 0 when weights have large absolute value $\Rightarrow$ weights do not change)
- most common activation function for deep NNs
- extremely common for CNNs
- leads to sparse activation
- more efficiently computable than other activation functions

ReLU (Rectified Linear Unit)

$$\sigma(z) = \max\{0, z\}$$



- helps convergence (almost linear)
- does not hurt expressiveness
- prevents the *vanishing gradient function* (gradient for sigmoid o threshold functions is ≈ 0 when weights have large absolute value $\Rightarrow$ weights do not change)
- most common activation function for deep NNs
- extremely common for CNNs
- leads to sparse activation
- more efficiently computable than other activation functions

The layer with ReLU activation is sometimes called **detector layer**

Pooling

Pooling

Pooling function: replaces the output of the network at a certain location with a summary statistic of the nearby outputs.

Pooling

Pooling function: replaces the output of the network at a certain location with a summary statistic of the nearby outputs.

(Popular) Examples:

Pooling

Pooling function: replaces the output of the network at a certain location with a summary statistic of the nearby outputs.

(Popular) Examples:

- *maximum* of a rectangular neighborhood (**max pooling**)

Pooling

Pooling function: replaces the output of the network at a certain location with a summary statistic of the nearby outputs.

(Popular) Examples:

- *maximum* of a rectangular neighborhood (**max pooling**)
- *average* of a rectangular neighborhood (**average pooling**)

Pooling

Pooling function: replaces the output of the network at a certain location with a summary statistic of the nearby outputs.

(Popular) Examples:

- *maximum* of a rectangular neighborhood (**max pooling**)
- *average* of a rectangular neighborhood (**average pooling**)
- weighted average based on the distance from the network location

Pooling

Pooling function: replaces the output of the network at a certain location with a summary statistic of the nearby outputs.

(Popular) Examples:

- *maximum* of a rectangular neighborhood (**max pooling**)
- *average* of a rectangular neighborhood (**average pooling**)
- weighted average based on the distance from the network location
- ...

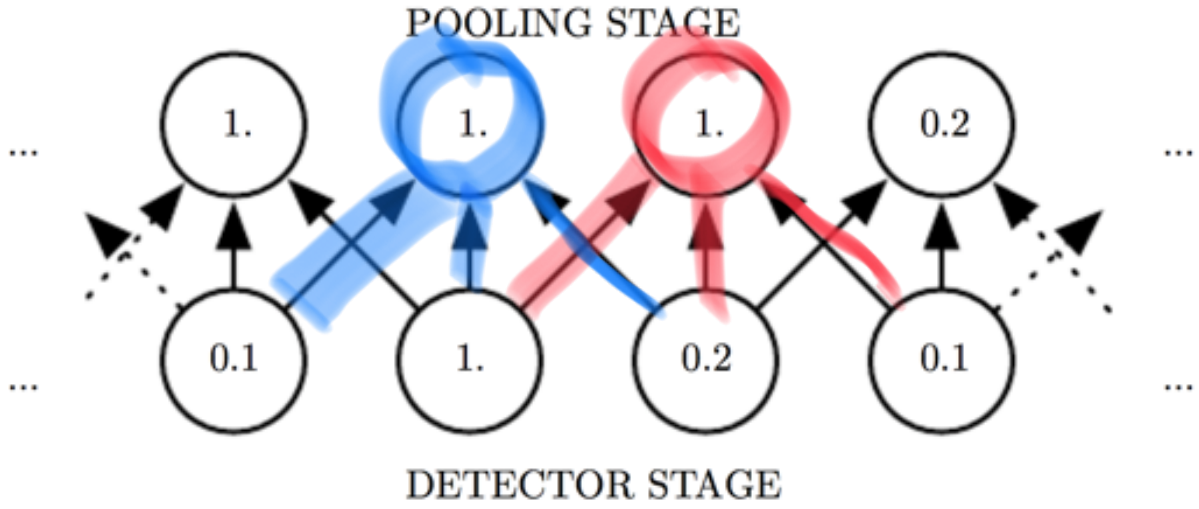
Pooling

Pooling function: replaces the output of the network at a certain location with a summary statistic of the nearby outputs.

(Popular) Examples:

- *maximum* of a rectangular neighborhood (**max pooling**)
- *average* of a rectangular neighborhood (**average pooling**)
- weighted average based on the distance from the network location
- ...

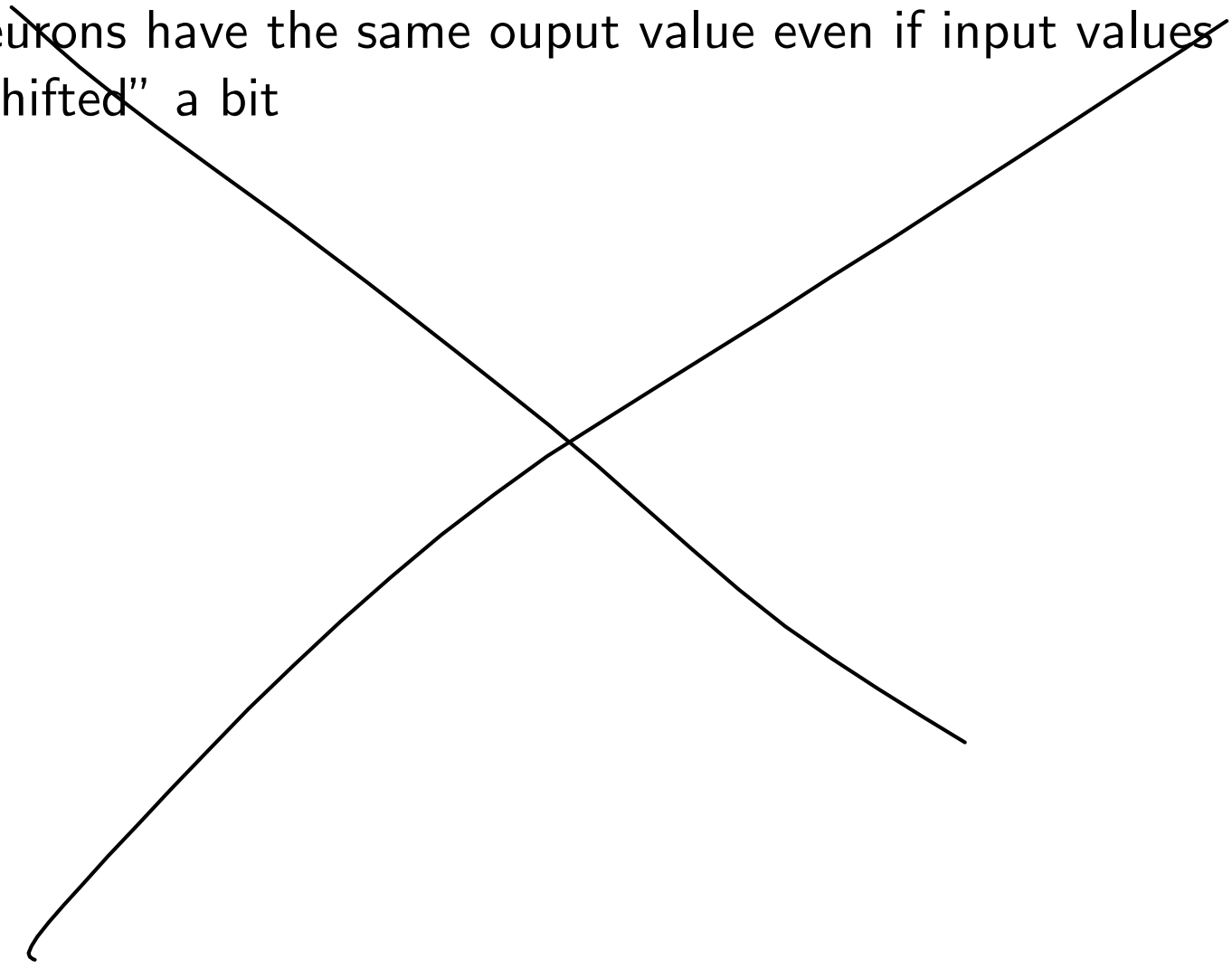
Example: Max Pooling



Max Pooling and Invariance

Max Pooling and Invariance

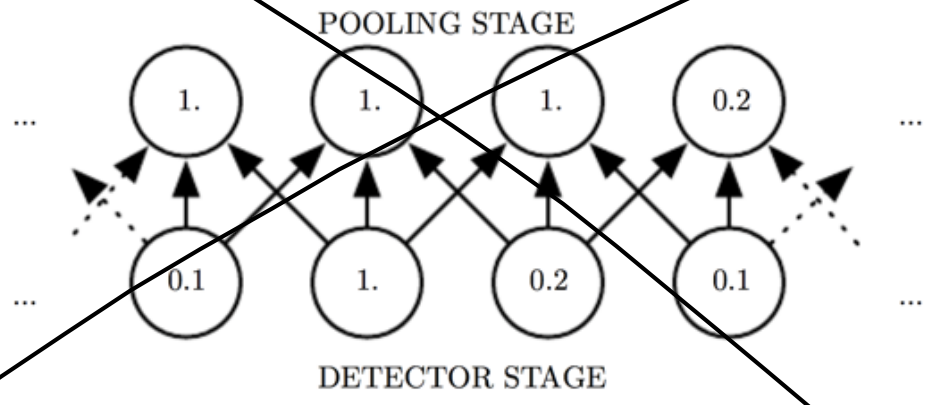
Max pooling introduces **invariance** to local translation: many neurons have the same ouput value even if input values are "shifted" a bit



Max Pooling and Invariance

Max pooling introduces **invariance** to local translation: many neurons have the same ouput value even if input values are "shifted" a bit

Before shift

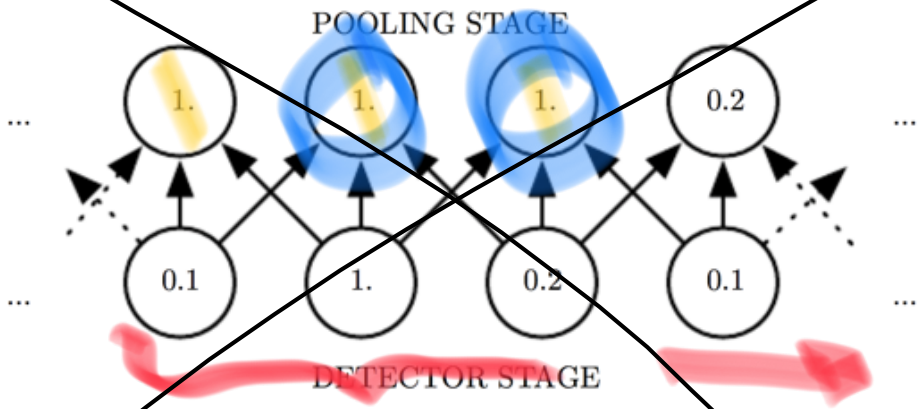


← representation

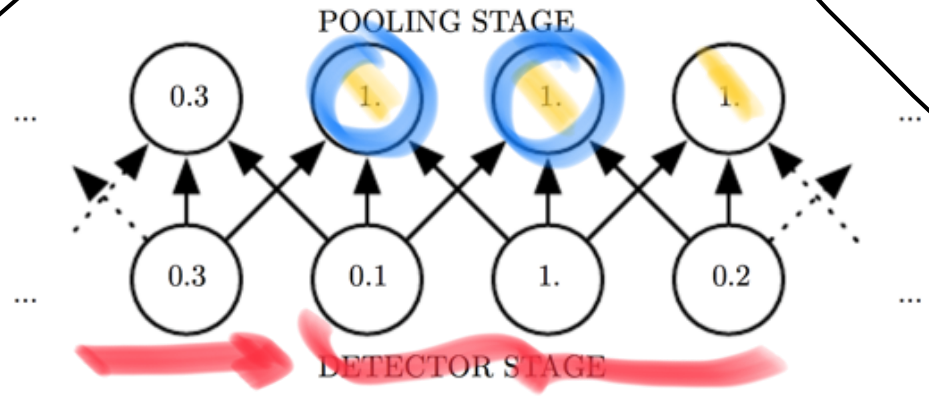
Max Pooling and Invariance

Max pooling introduces **invariance** to local translation: many neurons have the same ouput value even if input values are "shifted" a bit

Before shift



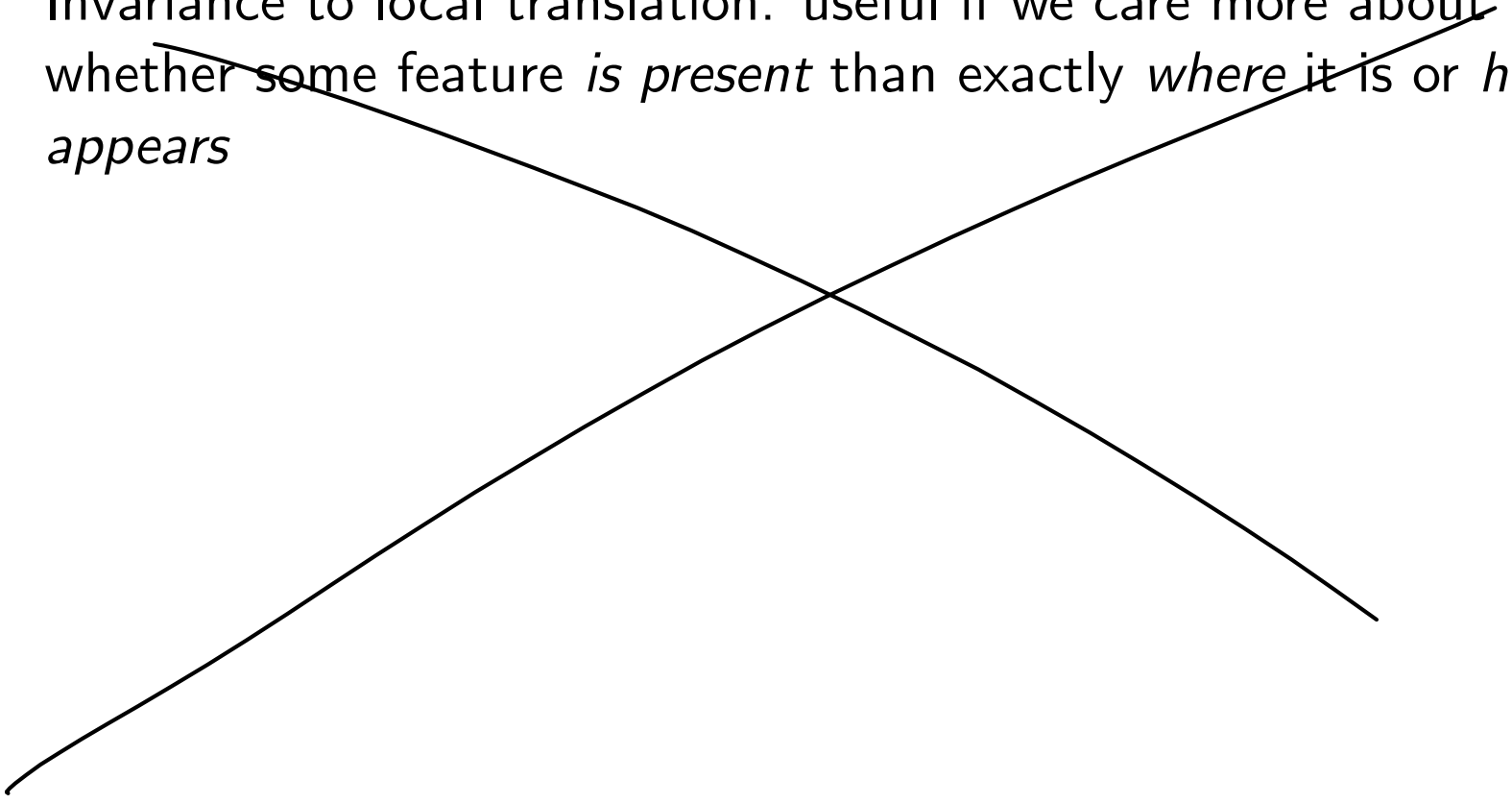
After shift (to the right by 1)



Max Pooling and Invariance (continue)

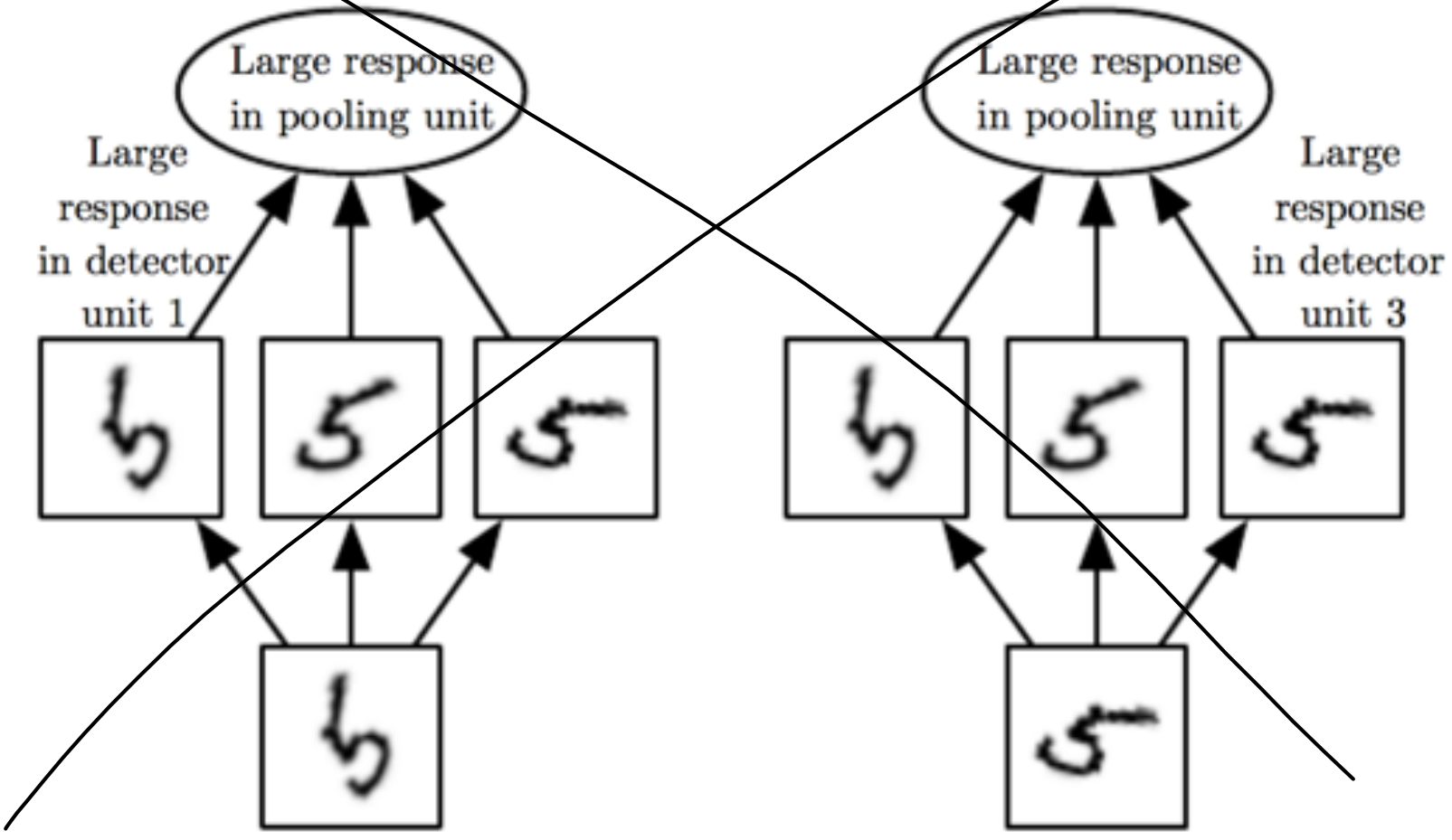
Max Pooling and Invariance (continue)

Invariance to local translation: useful if we care more about whether some feature *is present* than exactly *where* it is or *how it appears*



Max Pooling and Invariance (continue)

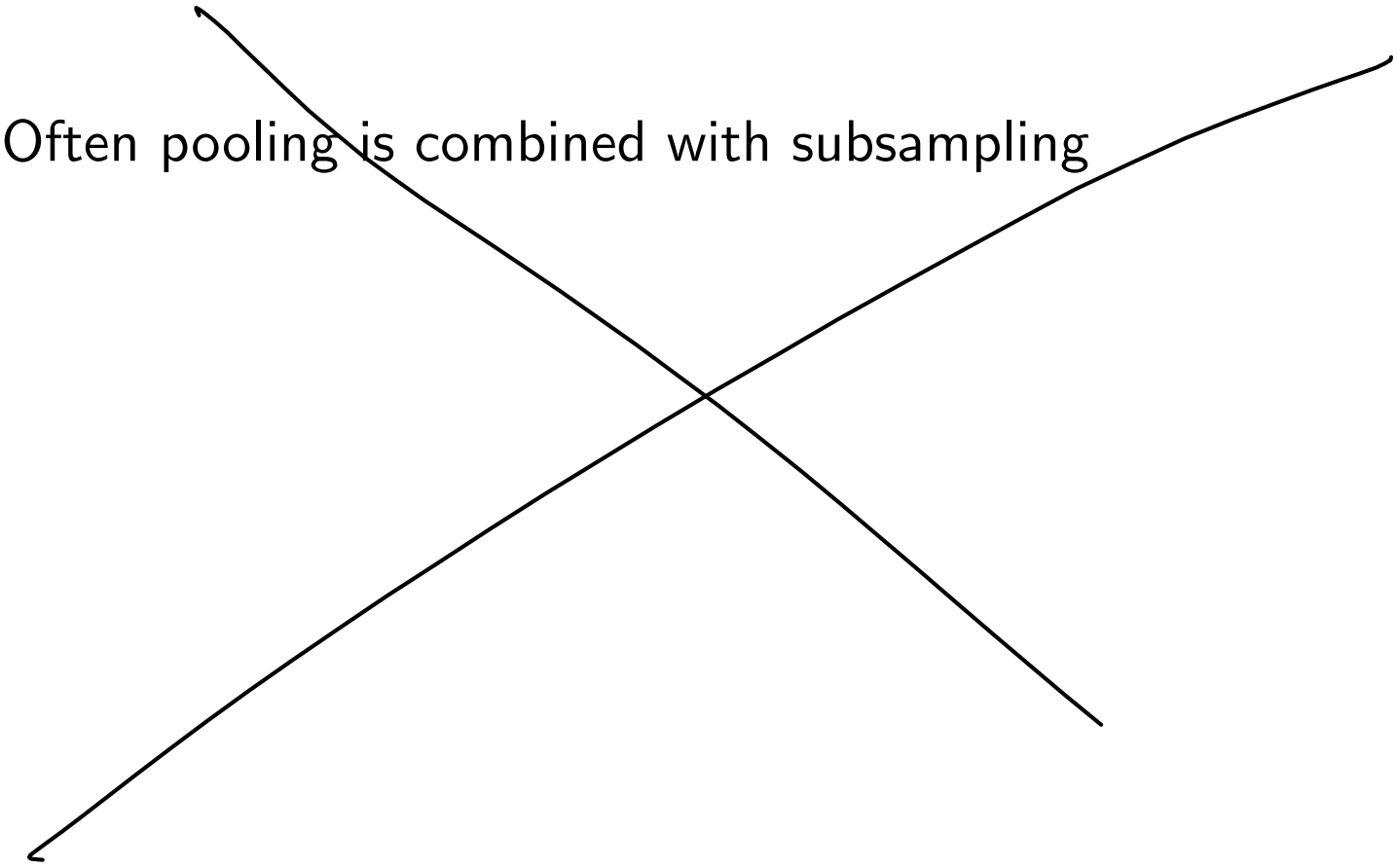
Invariance to local translation: useful if we care more about whether some feature *is present* than exactly *where* it is or *how it appears*



Pooling and Subsampling

Pooling and Subsampling

Often pooling is combined with subsampling



Pooling and Subsampling

Often pooling is combined with subsampling

Skip the application of pooling for a given number of steps = **stride**

Pooling and Subsampling

Often pooling is combined with subsampling

Skip the application of pooling for a given number of steps = **stride**

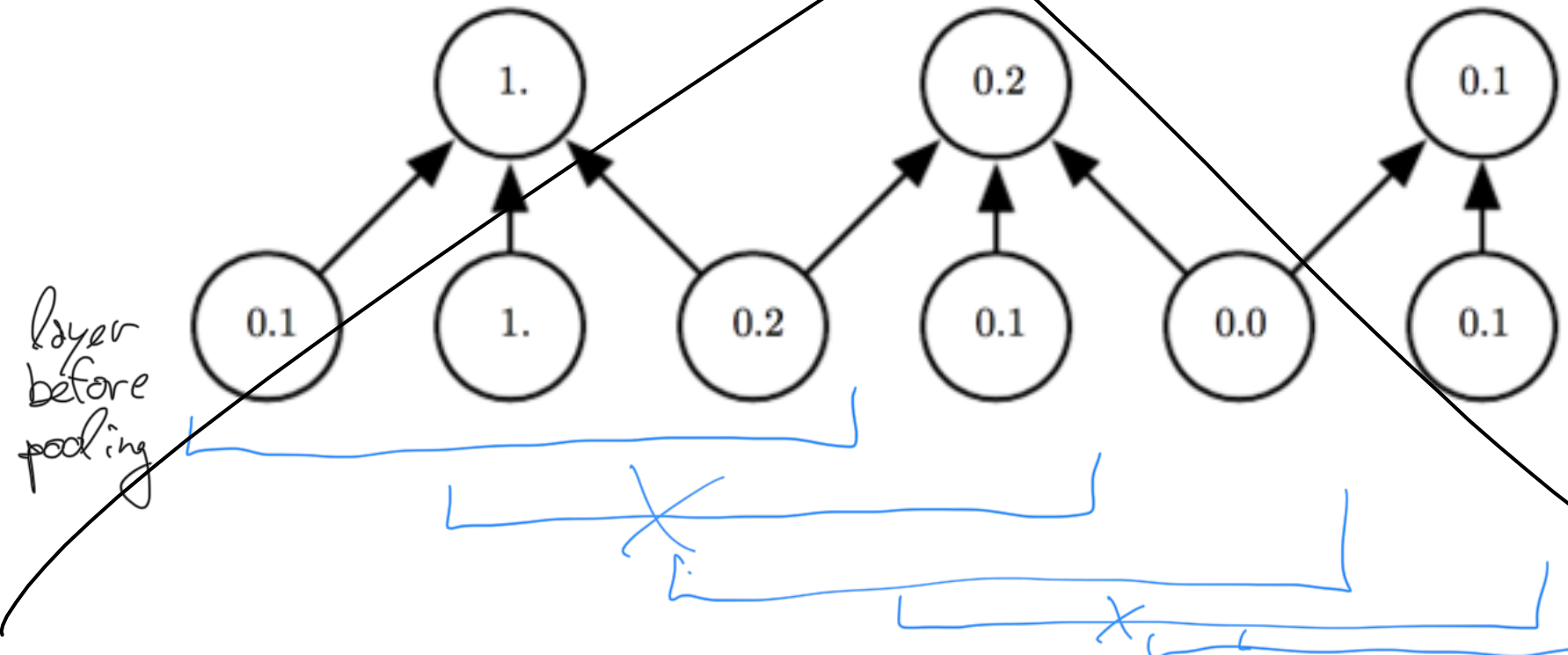
Example: max pooling, pool width = 3, stride =2

Pooling and Subsampling

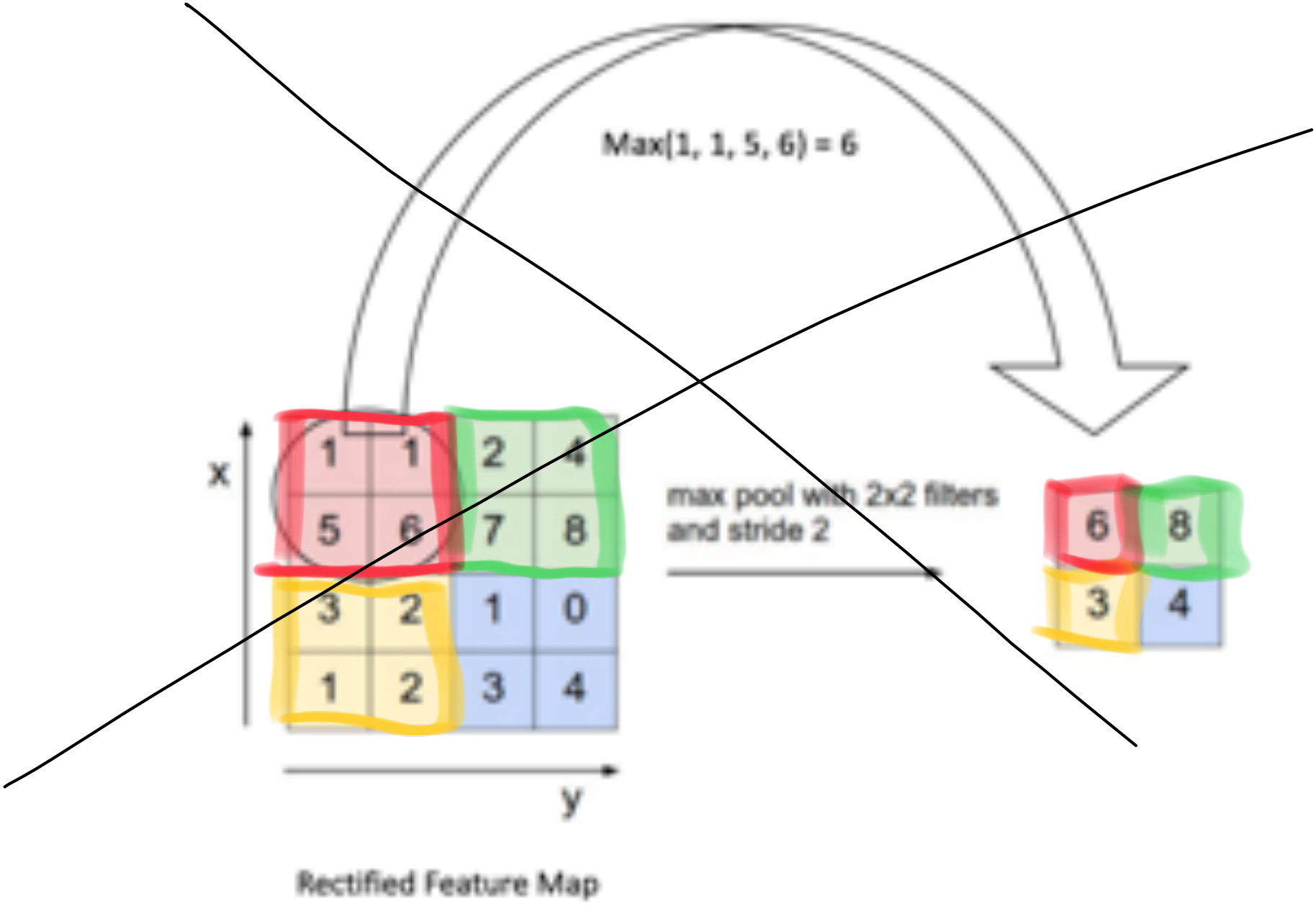
Often pooling is combined with subsampling

Skip the application of pooling for a given number of steps = **stride**

Example: max pooling, pool width = 3, stride =2



Max Pooling and Subsampling: Example



Pooling and Subsampling: Notes

Pooling and subsampling have some nice properties:

Pooling and Subsampling: Notes

Pooling and subsampling have some nice properties:

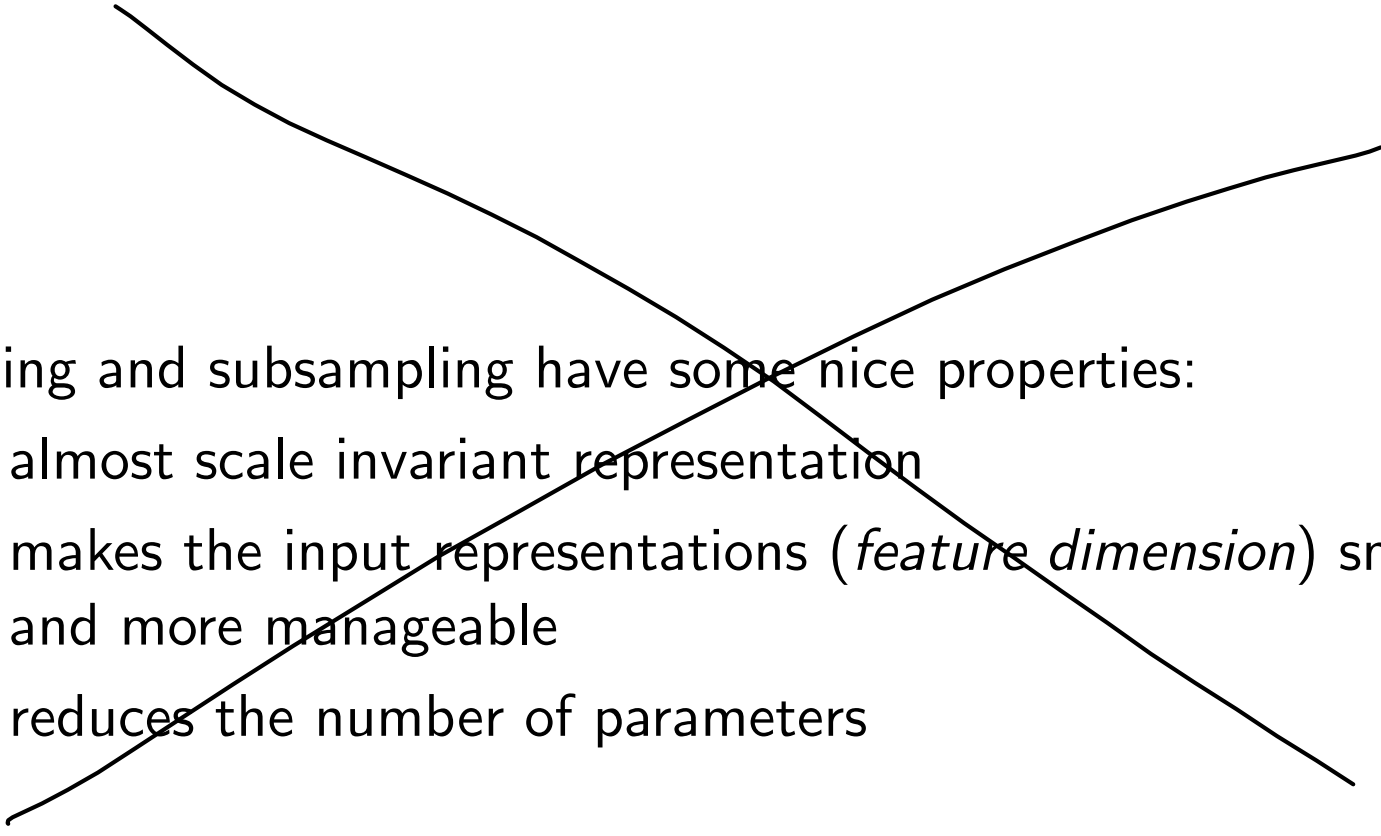
- almost scale invariant representation

Pooling and Subsampling: Notes

Pooling and subsampling have some nice properties:

- almost scale invariant representation
- makes the input representations (*feature dimension*) smaller and more manageable

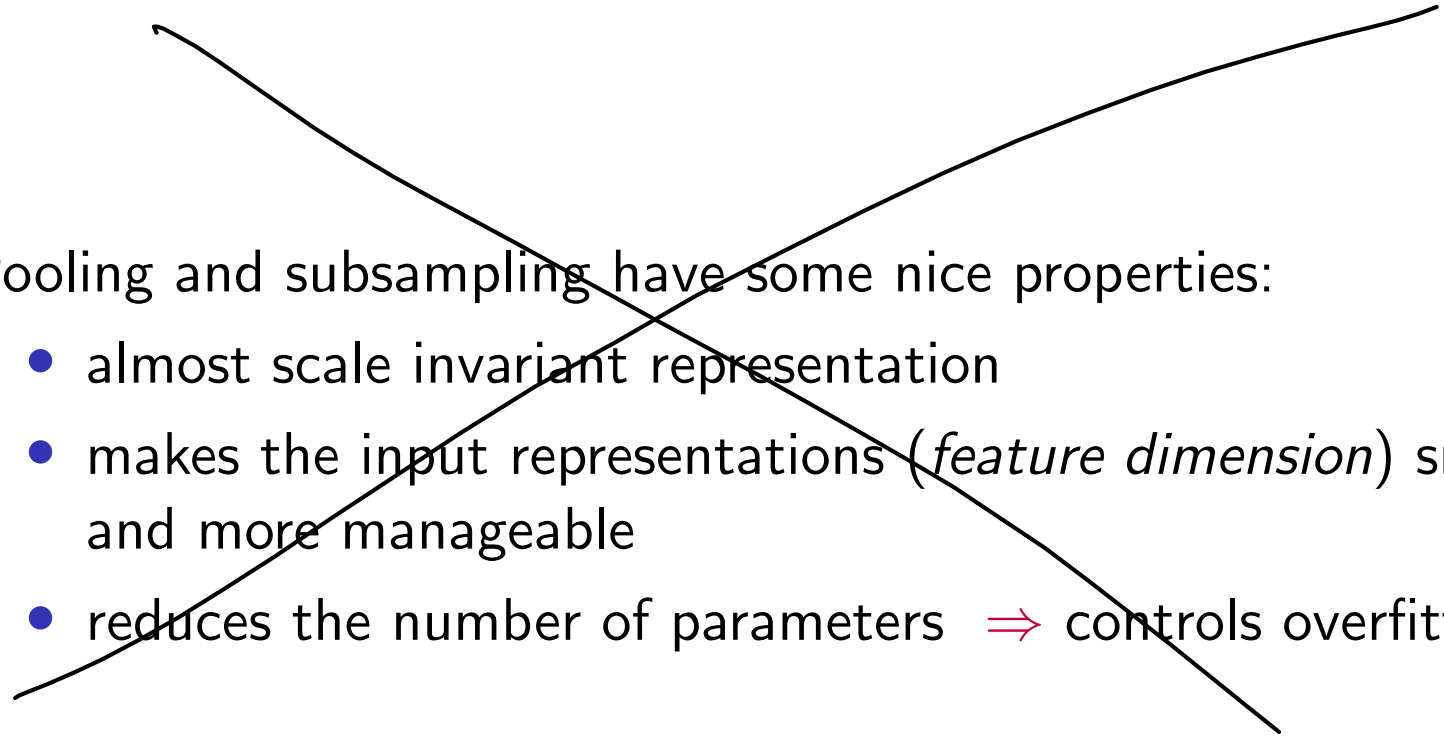
Pooling and Subsampling: Notes



Pooling and subsampling have some nice properties:

- almost scale invariant representation
- makes the input representations (*feature dimension*) smaller and more manageable
- reduces the number of parameters

Pooling and Subsampling: Notes



Pooling and subsampling have some nice properties:

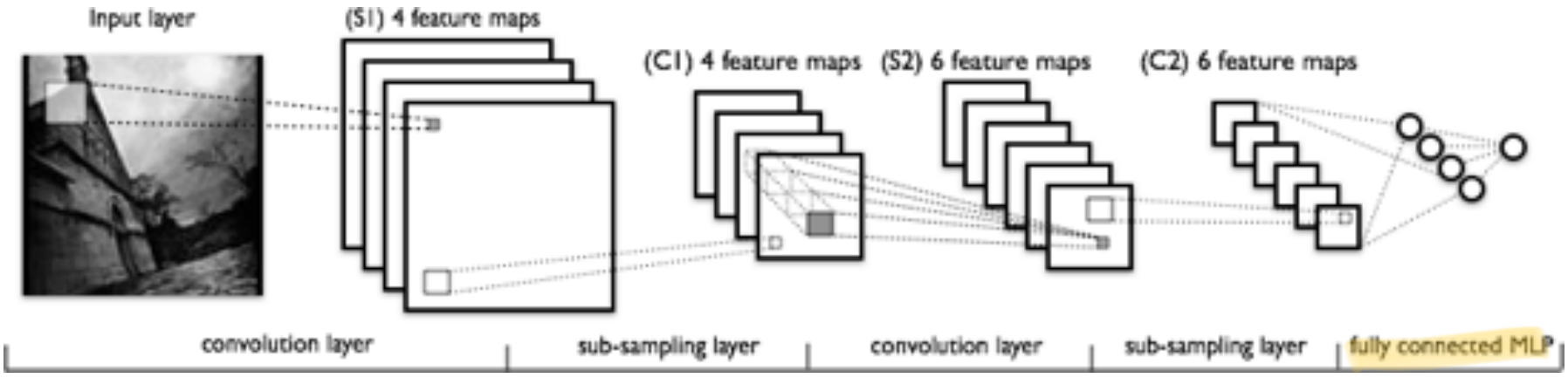
- almost scale invariant representation
- makes the input representations (*feature dimension*) smaller and more manageable
- reduces the number of parameters $\Rightarrow$ controls overfitting

Pooling and Subsampling: Notes

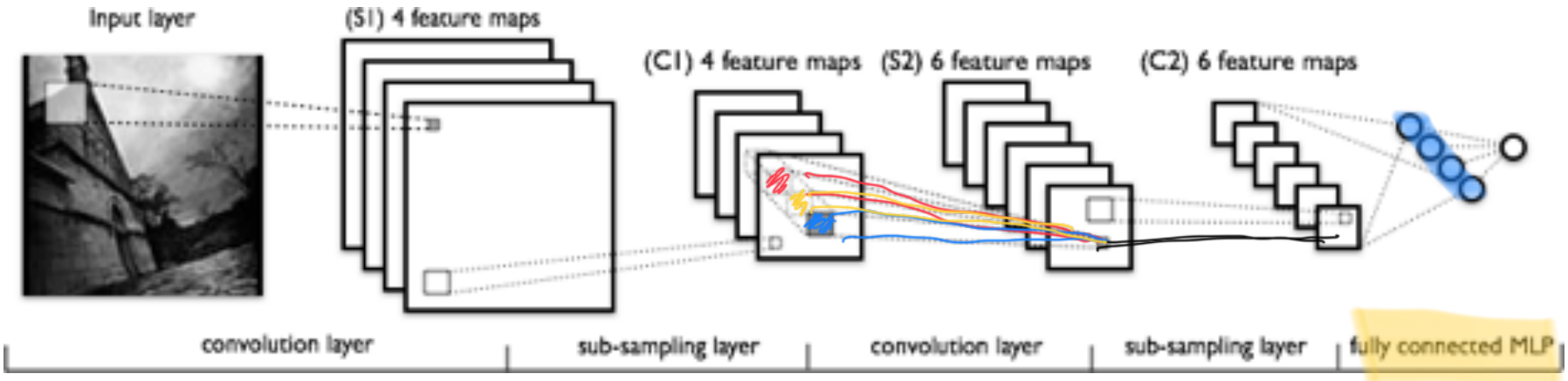
Pooling and subsampling have some nice properties:

- almost scale invariant representation
- makes the input representations (*feature dimension*) smaller and more manageable
- reduces the number of parameters $\Rightarrow$ controls overfitting
- reduces computation

CNN: Last Layer(s)



CNN: Last Layer(s)



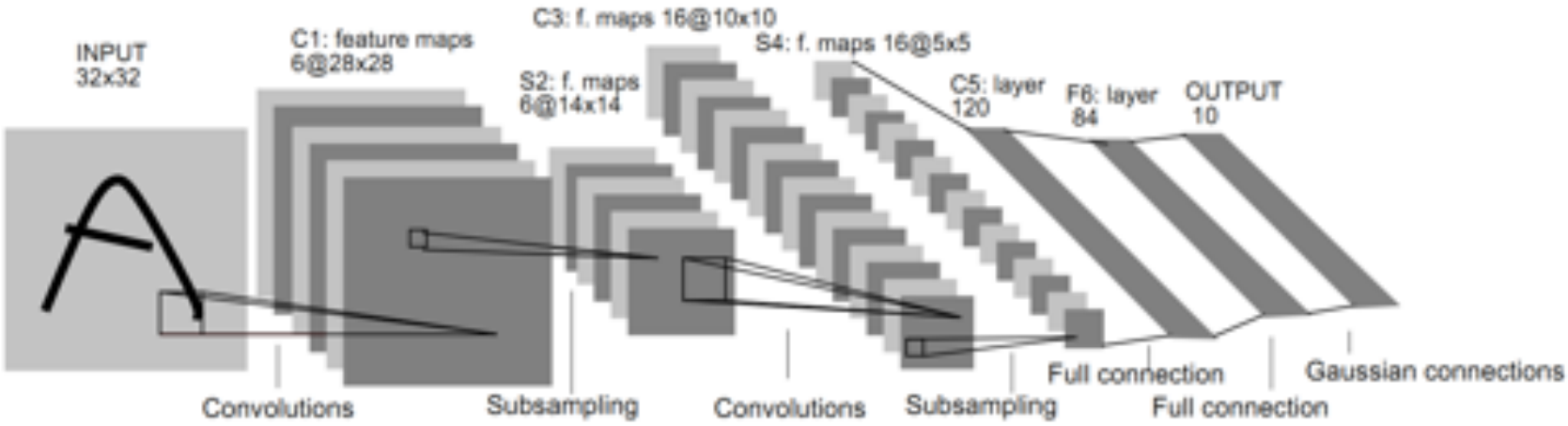
At the end: fully connected NN (MLP - Multi Layer Perceptron), which learns from the features extracted at the previous hidden layers

CNN: LeNet-5

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proc. IEEE 86(11): 2278-2324, 1998.

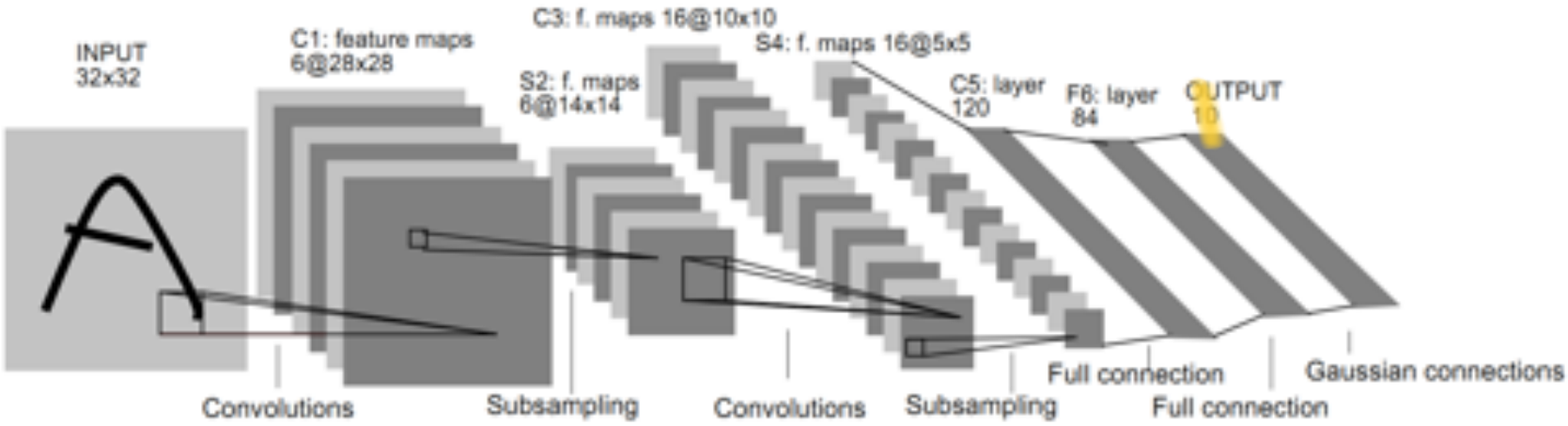
CNN: LeNet-5

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proc. IEEE 86(11): 2278-2324, 1998.



CNN: LeNet-5

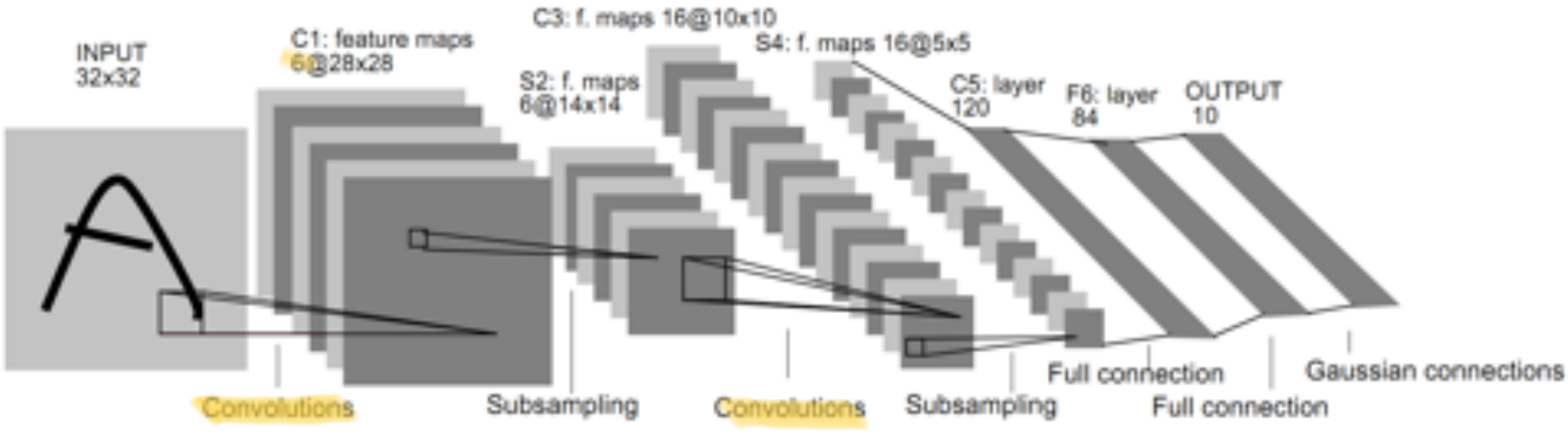
Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proc. IEEE 86(11): 2278-2324, 1998.



- task: digit recognition (output: $\Pr[\text{INPUT} = i]$ for $i = 0, 1, \ldots, 9$)

CNN: LeNet-5

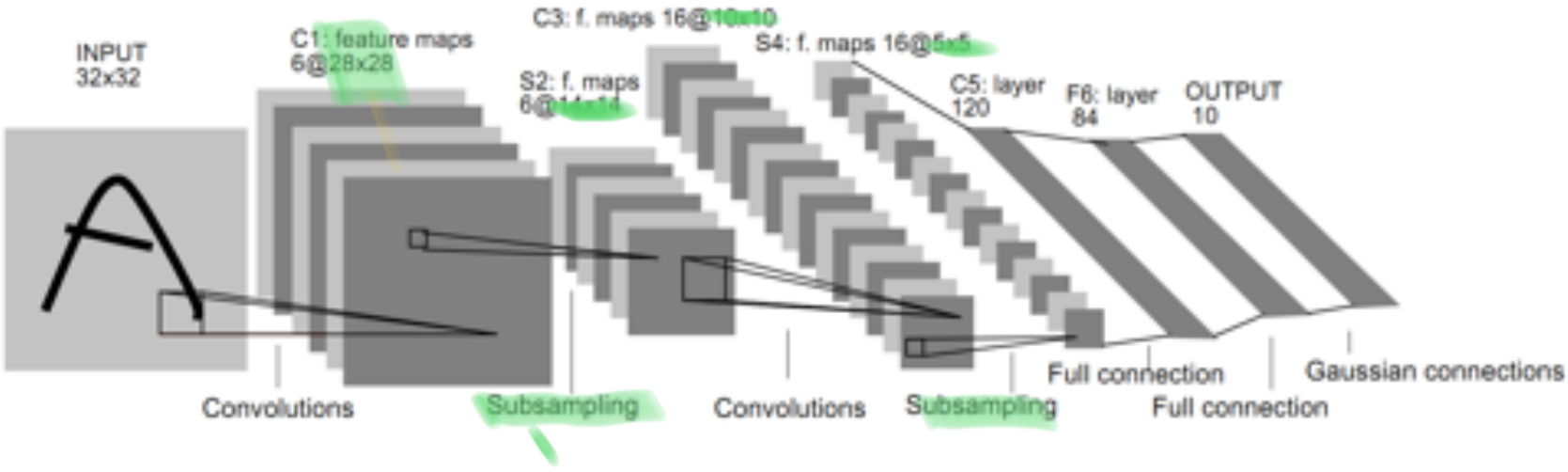
Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proc. IEEE 86(11): 2278-2324, 1998.



- task: digit recognition (output: $\Pr[\text{INPUT} = i]$ for $i = 0, 1, \ldots, 9$)
- convolutional filters: 5×5

CNN: LeNet-5

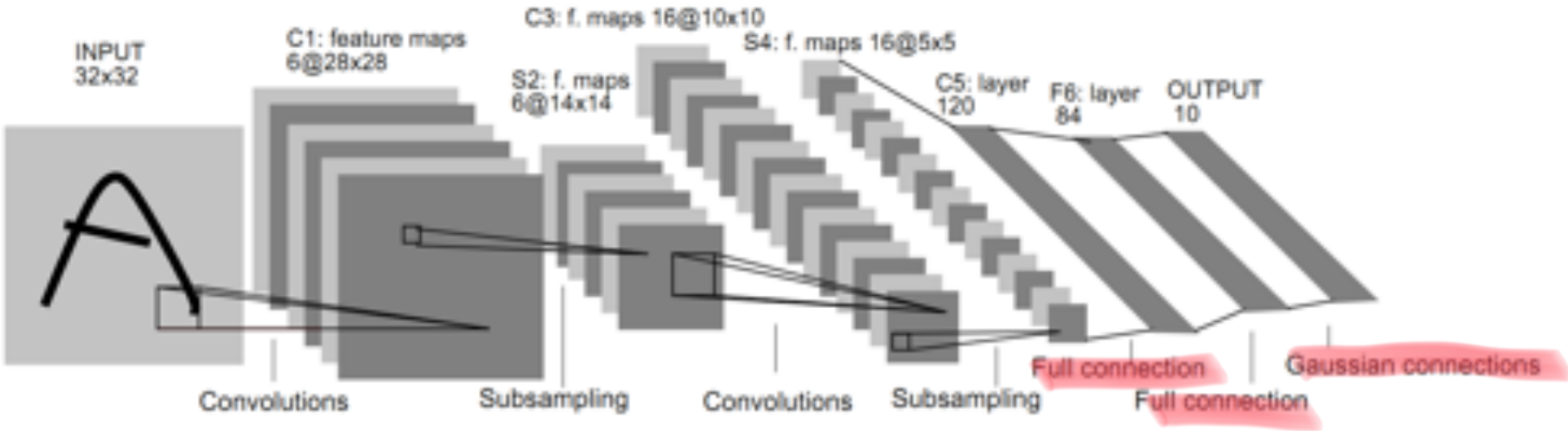
Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proc. IEEE 86(11): 2278-2324, 1998.



- task: digit recognition (output: $\Pr[\text{INPUT} = i]$ for $i = 0, 1, \ldots, 9$)
- convolutional filters: 5×5
- subsampling and pooling filters: 2×2, stride $= 2$

CNN: LeNet-5

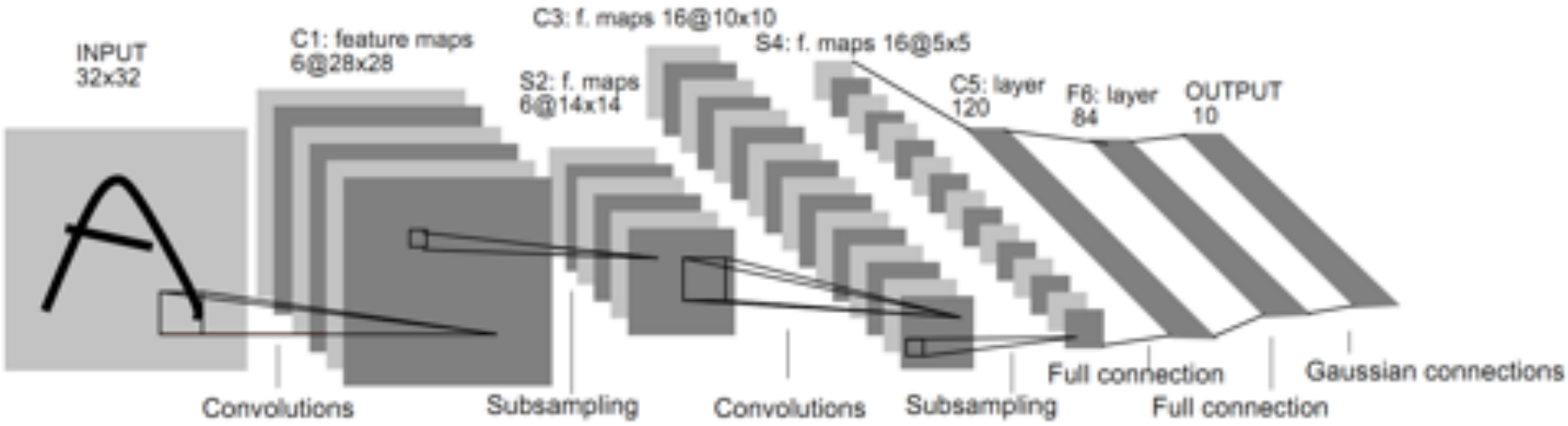
Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proc. IEEE 86(11): 2278-2324, 1998.



- task: digit recognition (output: $\Pr[\text{INPUT} = i]$ for $i = 0, 1, \ldots, 9$)
- convolutional filters: 5×5
- subsampling and pooling filters: 2×2, stride $= 2$
- used (some form of) average pooling

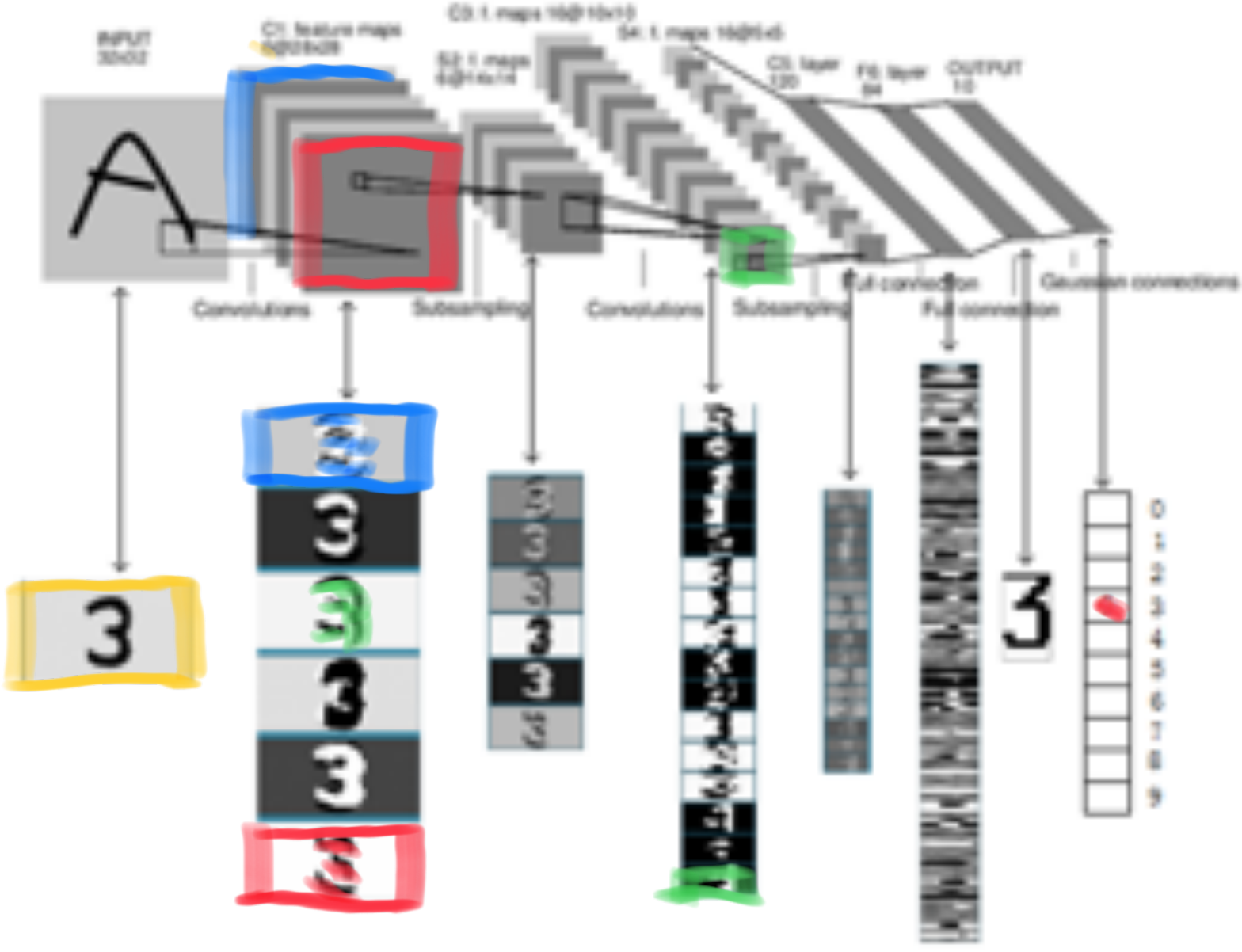
CNN: LeNet-5

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proc. IEEE 86(11): 2278-2324, 1998.

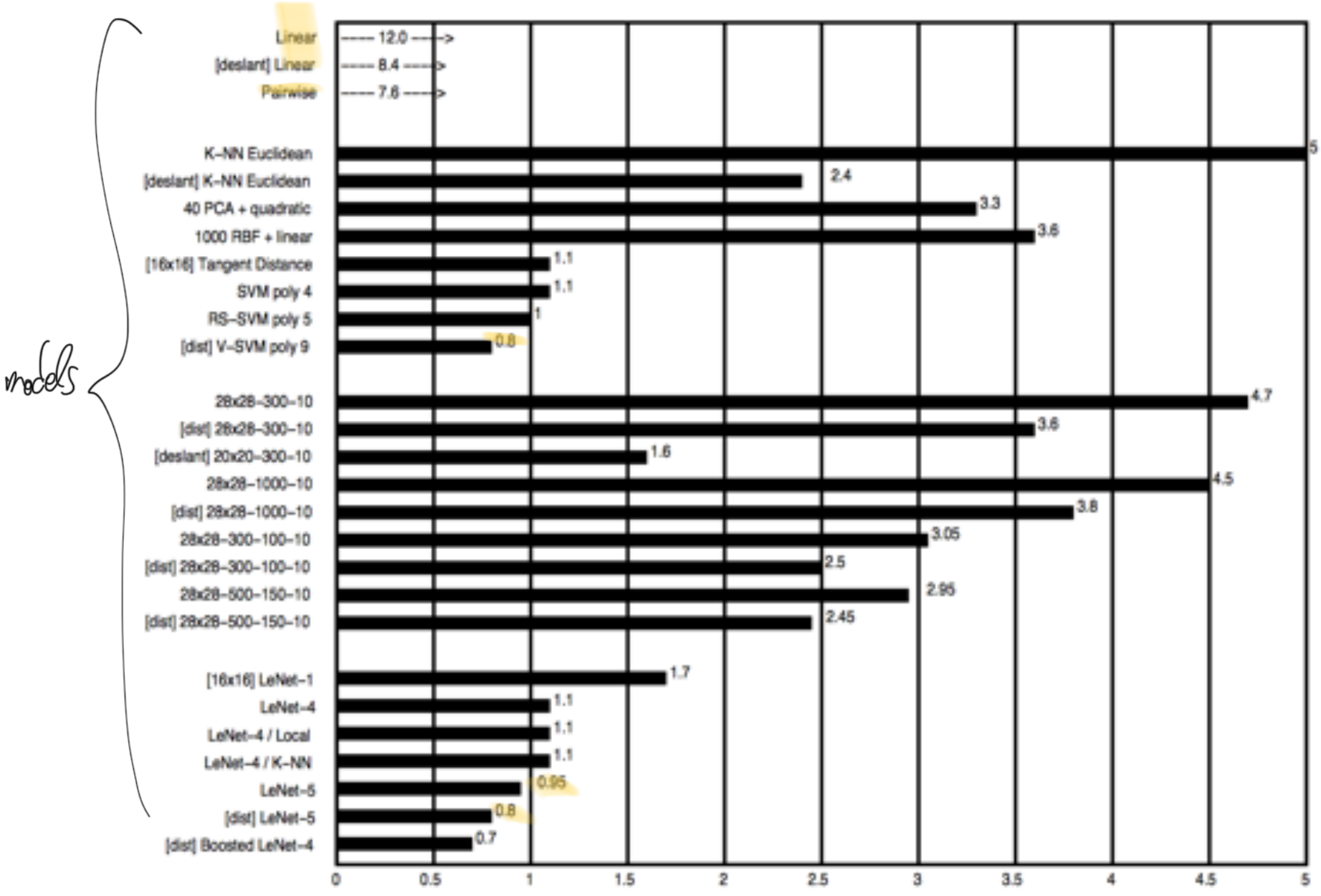


- task: digit recognition (output: $\Pr[\text{INPUT} = i]$ for $i = 0, 1, \ldots, 9$)
- convolutional filters: 5×5
- subsampling and pooling filters: 2×2, stride $= 2$
- used (some form of) average pooling
- trained on MNIST digit dataset with 60K training examples

LeNet-5 Features



LeNet-5 Results



How well do CNN perform?

ImageNet competition: "The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) evaluates algorithms for object detection and image classification at large scale" (http://www.image-net.org/challenges/LSVRC/)

Uses CNN; does not uses CNN

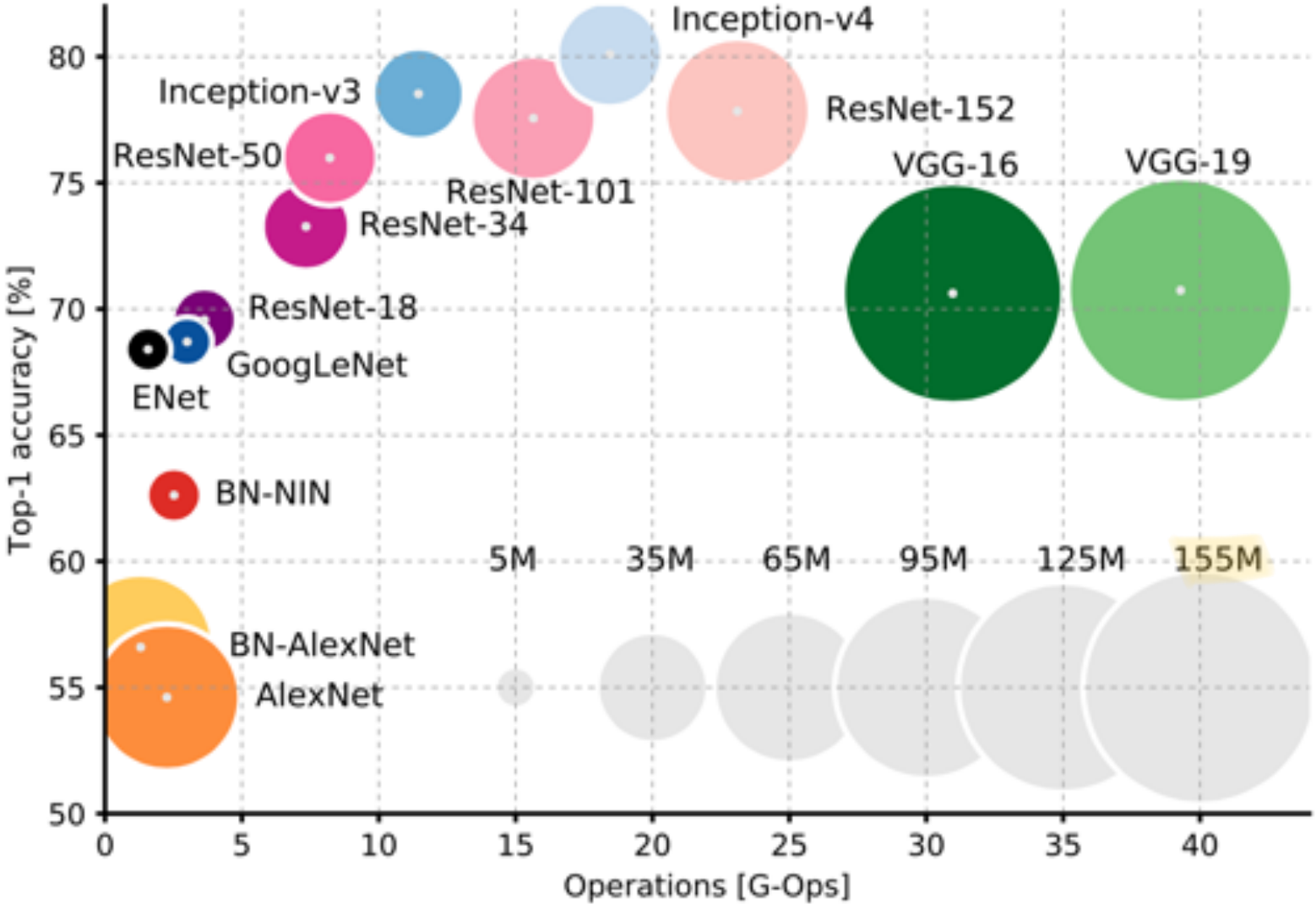
2012 Teams	%error
Supervision (Toronto)	15.3
ISI (Tokyo)	26.1
VGG (Oxford)	26.9
XRCE/INRIA	27.0
UvA (Amsterdam)	29.6
INRIA/LEAR	33.4

2013 Teams	%error
Clarifai (NYU spinoff)	11.7
NUS (singapore)	12.9
Zeiler-Fergus (NYU)	13.5
A. Howard	13.5
OverFeat (NYU)	14.1
UvA (Amsterdam)	14.2
Adobe	15.2
VGG (Oxford)	15.2
VGG (Oxford)	23.0

2014 Teams	%error
GoogLeNet	6.6
VGG (Oxford)	7.3
MSRA	8.0
A. Howard	8.1
DeeperVision	9.5
NUS-BST	9.7
TTIC-ECP	10.2
XYZ	11.2
UvA	12.1

2017: WMW wins with error $< 2.3\%$ - CNN with > 150 layers!

Many more CNNs have been proposed...



CNNs: Other Important Details

- how to learn parameters for a CNN? SGD with backpropagation!
  Note: ReLU is not differentiable $\Rightarrow$ use right/left/average derivative...
- **Input normalization**: divide each element of input to make sure it is in [0, 1]
- **Initialization is important**: one trick that works well in practice is to initialize the bias to be zero and initialize the each weight to be random in $[-1/d, 1/d]$, where d is the dimension of the input
- **Mini-batches**: at each iteration of SGD we calculate the average loss on k random examples for $k > 1$. Advantages:
  - reduces the variance of the update direction (w.r.t. the full gradient) $\Rightarrow$ converges faster
  - don't loose a lot in running time (parallel implementation)
- many variants of SGD... e.g., ADAM
- techniques to help avoiding overfitting
- a specific loss function
- ...

ADAM [Kingma and Ba 2014]

ADAM = Adaptive Moment Estimation Iterative method. Let $\mathbf{g}^{(t)} \in \nabla f(\mathbf{x})$. Updates at iteration t:

1. $\mathbf{m}^{(t)} \leftarrow \beta_1 \mathbf{m}^{(t-1)} + (1-\beta_1)\mathbf{g}^{(t)}$
2. $\mathbf{v}^{(t)} \leftarrow \beta_2 \mathbf{m}^{(t-1)} + (1-\beta_2)(\mathbf{g}^{(t)})^2$
3. $\hat{\mathbf{m}}^{(t)} \leftarrow \mathbf{m}^{(t)}/(1-\beta_1^t)$
4. $\hat{\mathbf{v}}^{(t)} \leftarrow \mathbf{v}^{(t)}/(1-\beta_2^t)$
5. $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \hat{\mathbf{m}}^{(t)}/(\hat{\mathbf{v}}^{(t)} + \delta)$

$\mathbf{m}^{(t)}$: first moment (*momentum*, helps for non-smooth loss functions)
$\mathbf{v}^{(t)}$: second moment (better estimates and convergence)
$\hat{\mathbf{m}}^{(t)}$: unbiased estimate of first moment (because estimates start at 0)
$\hat{\mathbf{v}}^{(t)}$: unbiased estimate of second moment (because estimates start at 0)

ADAM (continue)

β_1, β_2, η: parameters

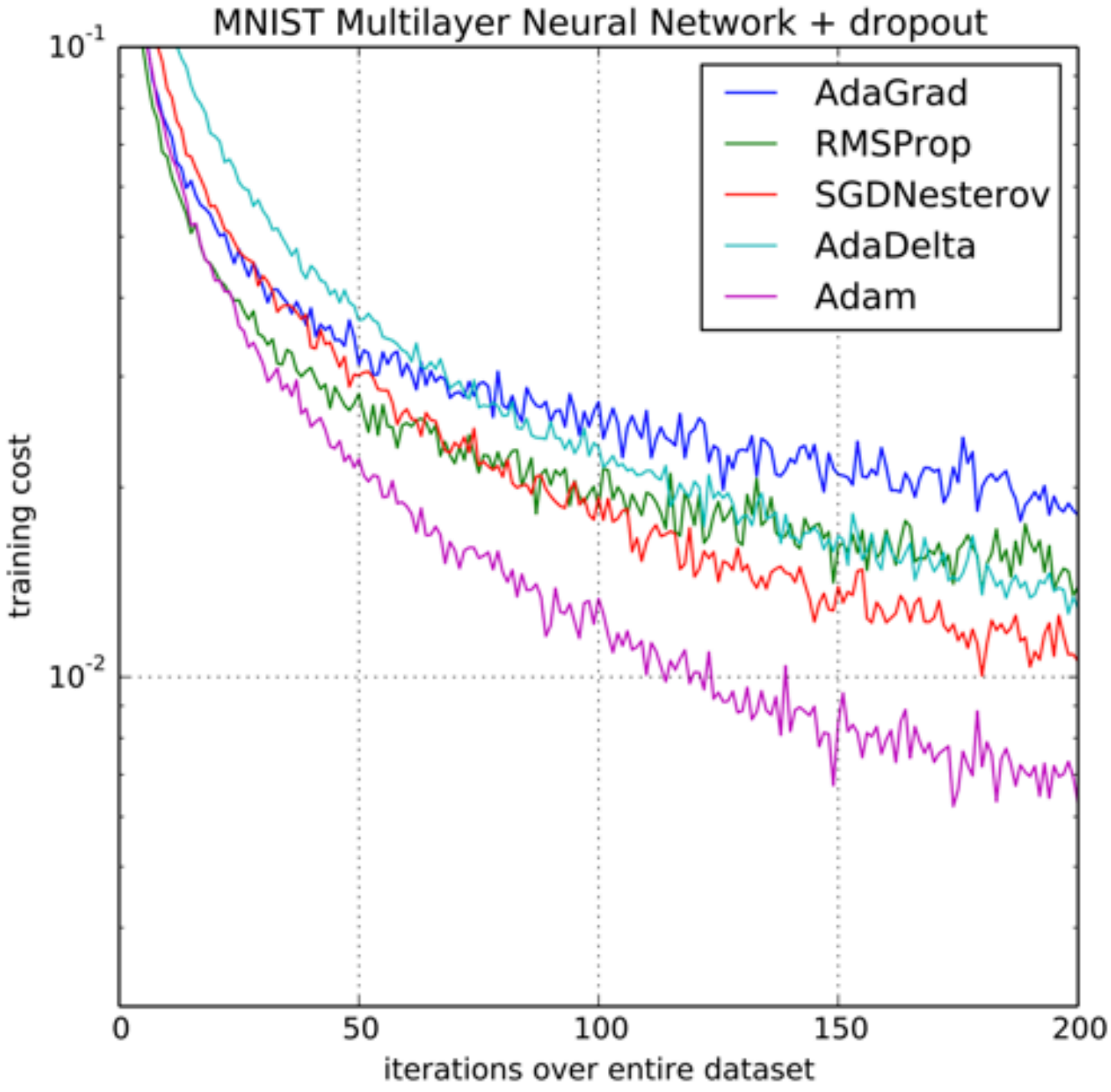
How to set them?

Validation, etc...

Good starting point:

- $\beta_1 = 0.9$
- $\beta_2 = 0.999$
- $\eta = 0.001$ or $\eta = 5 \times 10^{-4}$

ADAM: Results



Techniques to Help Avoiding Overfitting

- regularization! (same as for other models!)
- dropout
- early stopping
- data augmentation

Dropout

It provides a computationally inexpensive but powerful method of regularizing a broad family of models

Is a **bagging method**

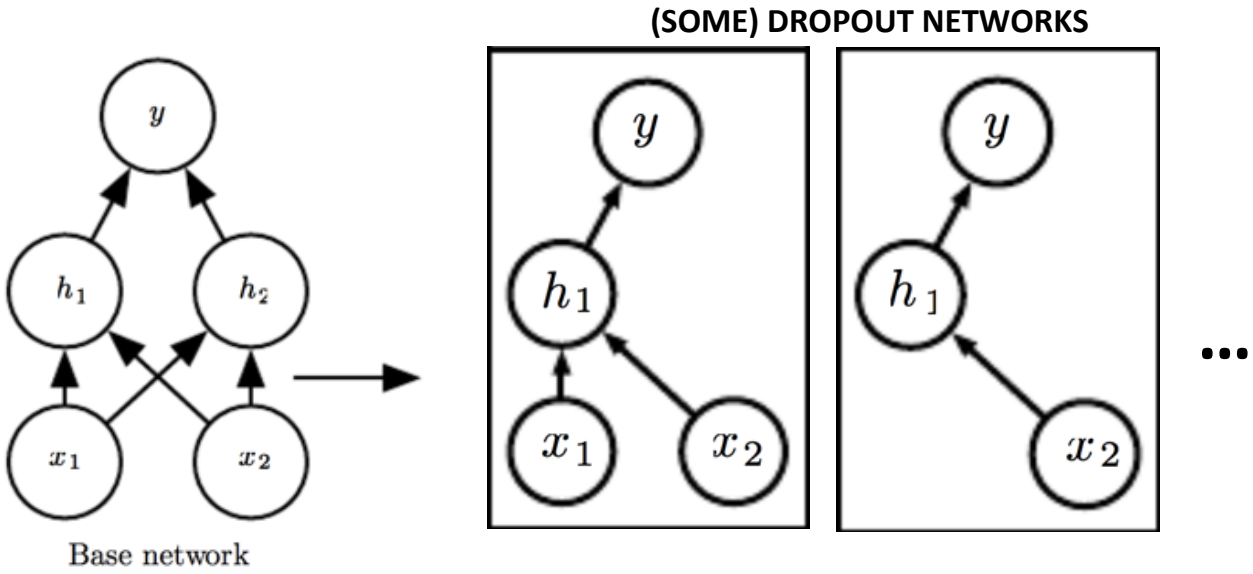
- **bagging** = training multiple models and evaluating multiple models on each test example
- impractical when each model is a large neural network: training and evaluating such networks is costly in terms of runtime and memory

Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of many neural networks

Dropout (continue)

In particular: dropout trains the ensemble consisting of all subnetworks that can be formed by removing nonoutput units from an underlying base network

- *remove unit* = set its output weights to 0's



Dropout (continue)

To train with dropout:

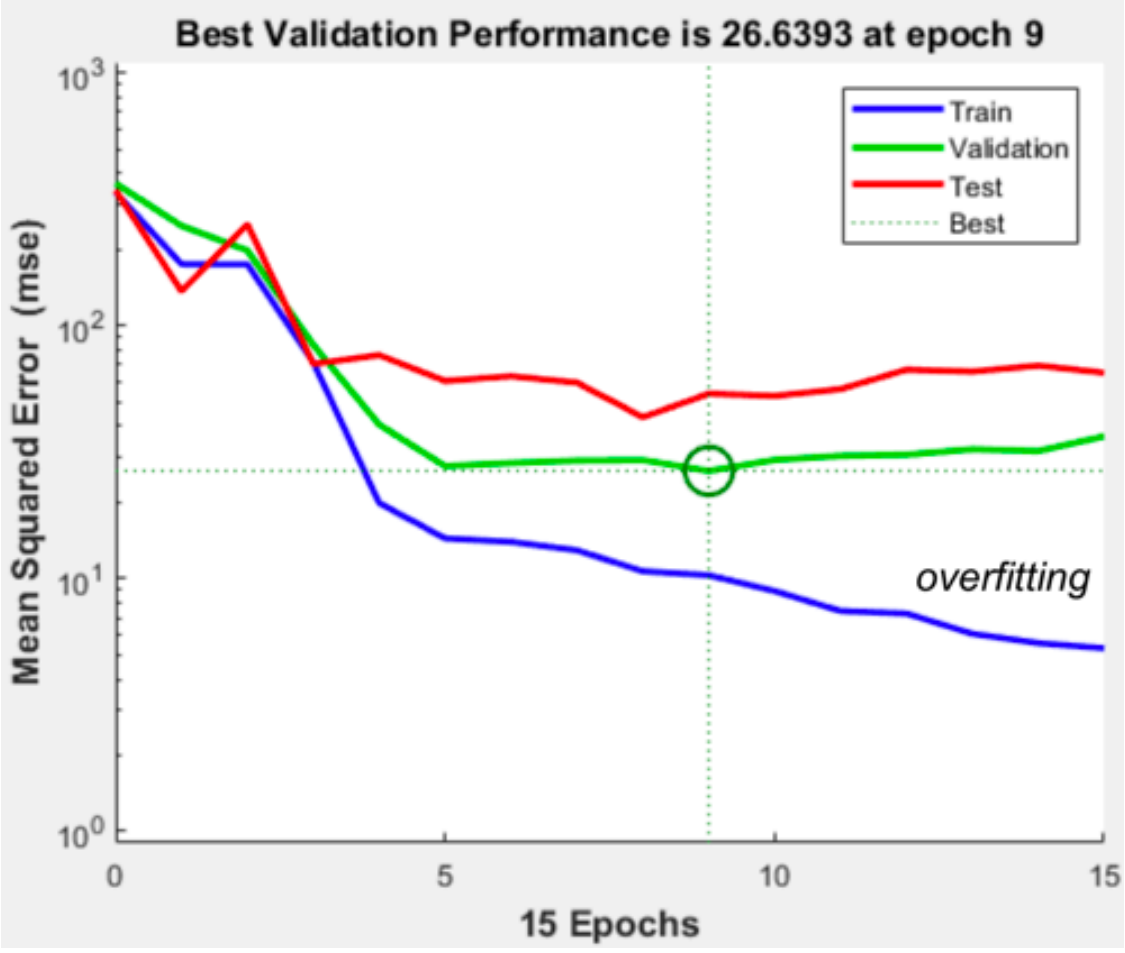
- use a minibatch-based SGD (or other learning algorithm that makes small steps)
- each time an input example is loaded into a minibatch:
  - randomly sample a different binary mask to apply to all the input and hidden units in the network;
  - mask for each unit is sampled independently from all the others;
  - probability of sampling a mask value of one (causing a unit to be included) is a hyperparameter fixed before training begins
  - run forward propagation, back-propagation, and learning update

Typically:

- input unit is included with probability 0.8
- hidden unit is included with probability 0.5

Why does it work? Still not well understood!

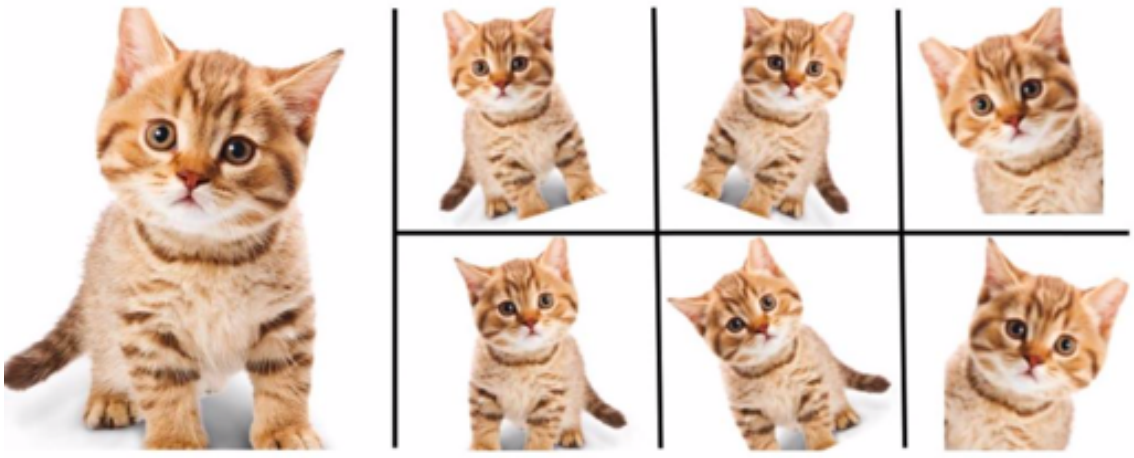
Early Stopping

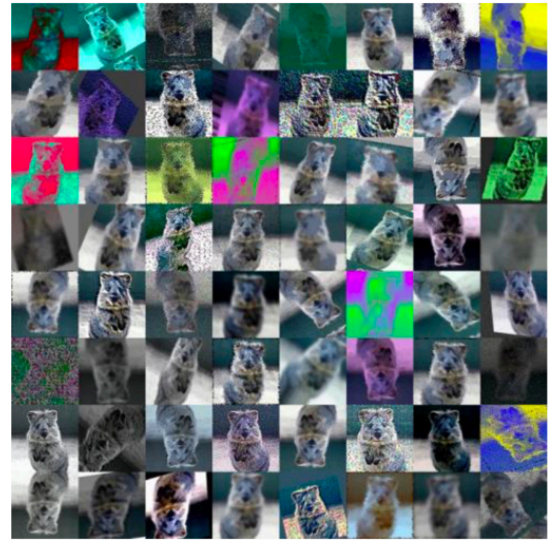


- Use validation error to decide when to stop training
- Stop when monitored loss has not improved after n subsequent epochs
- Parameter n is called *patience*

Data Augmentation

- Add a little bit of variance to the data and "virtually" increase number of training samples
- Artificially add noise
- Apply random transformations o crop part of the data
  - Resize/rescaledata
  - Rotate
  - Custom transformations depending on data type (e.g. for images: flip horizontally, contrast and saturation, ...)





Loss Function

Classification: *cross entropy* is commonly used (instead of 0-1 loss)

Hypotheses set: functions with (prediction) value in 0 and 1
$\Rightarrow$ use sigmoid activation function for output

Cross entropy loss:

$$\ell(h,(\mathbf{x},y)) = -y \log h(\mathbf{x}) - (1-y) \log (1-h(\mathbf{x}))$$

Cross entropy is a convex function $\Rightarrow$ SGD works better

The best hypothesis h w.r.t. to the cross entropy loss is:
$h(\mathbf{x}) = \Pr[y = 1|\mathbf{x}]$ (related to Bayes optimal predictor!)

For multiclass classification: k classes $(0, 1, \ldots, k)$

- output is vector $\mathbf{y}$ with $y_i = 1$ if correct class is i, 0 otherwise
- $h(\mathbf{x}) \in (0,1)^d$ with $h_i(\mathbf{x}) =$ probability label of $\mathbf{x}$ is i, $1 \leq i \leq k$
- $\ell(h,(\mathbf{x},\mathbf{y})) = -\sum_{i=1}^{k} y_i \log (h_i(\mathbf{x}))$

Deep Learning: What's Missing?

- running time for training: still very high!
- design choices: number of layers, types of layers, number of nodes, etc.... $\Rightarrow$ how?
- theoretical foundations still not completed...
  - "Machine Learning has become the new alchemy"
    Ali Rahimi (talk at NeurIPS 2017 - Test-of-time award presentation)
- interpretability of models not always possible
  - how do you explain hiring somebody? not giving them a loan? deciding to cure somebody?
  - *fairness* in ML...

Credits

- Yann LeCun CVPR 2015 keynote
- *Deep Learning*, I. Goodfellow, Y. Bengio, A. Courville, (http://www.deeplearningbook.org)