Convolutional Neural Networks

Antonio Barbalau
Bitdefender ML Team

* using slides, illustrations and ideas from prof. Simon Prince UDLB and Stanford CS231n

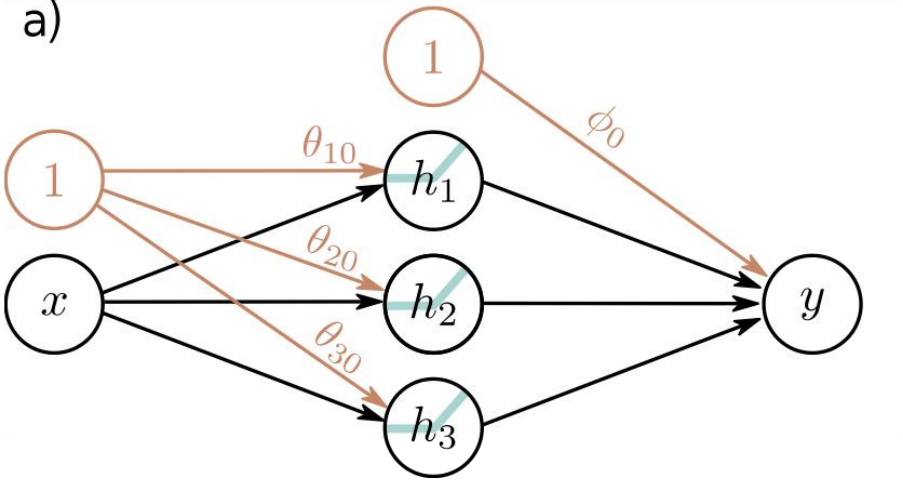
Recap

Depicting neural networks

$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x]$$

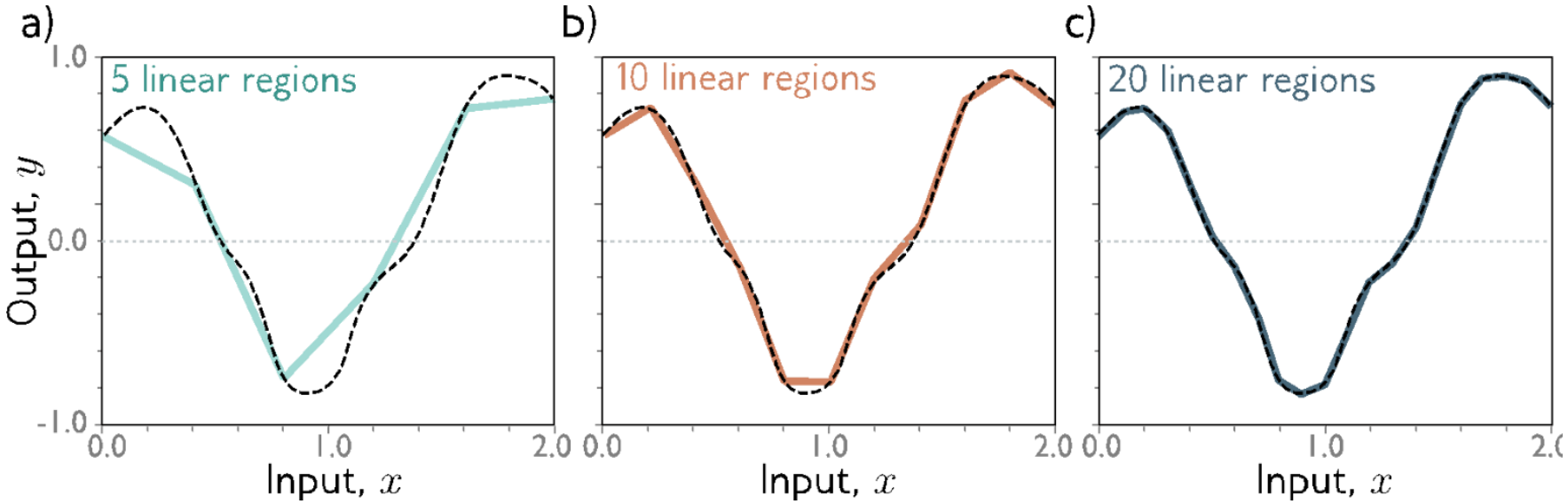
$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$

a)



Each parameter multiplies its sources and adds to its target

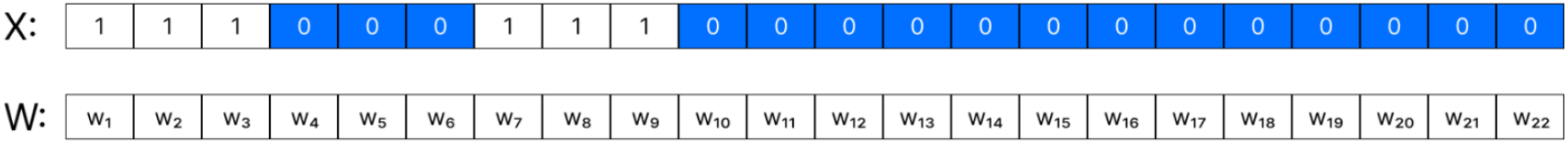
With enough hidden units...



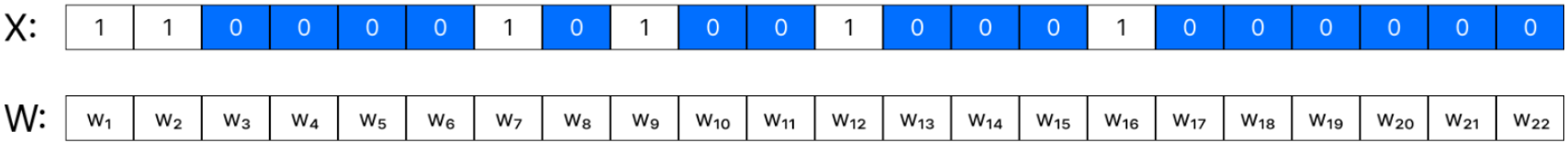
... we can describe any 1D function to arbitrary accuracy!

1D Example: Morse Code

Name a W and a b such that the neuron recognizes the first pattern while rejecting all other patterns.



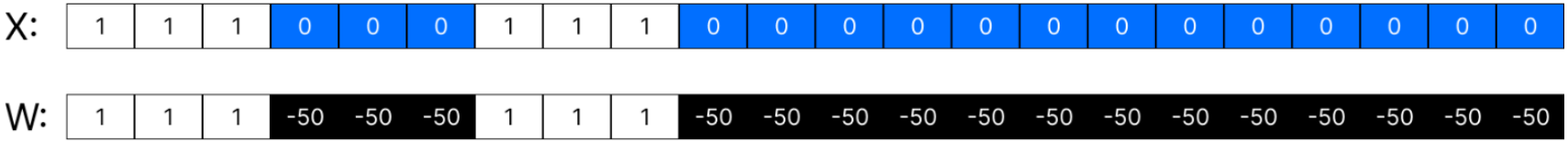
$$\text{ReLU}(\sum_i w_i x_i + b) > 0$$



$$\text{ReLU}(\sum_i w_i x_i + b) = 0$$

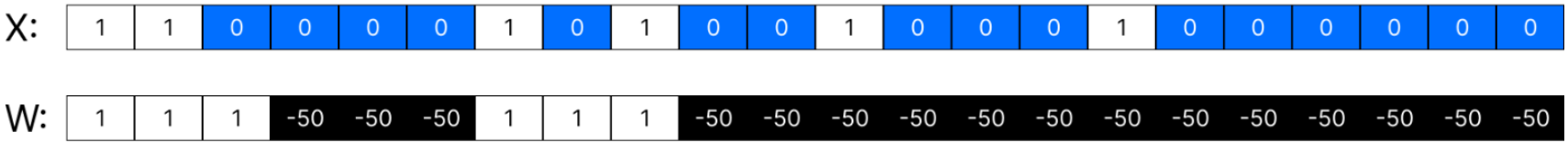
1D Example: Morse Code

Name a W and a b such that the neuron recognizes the first pattern while rejecting all other patterns.



$$\text{ReLU}(\sum_i w_i x_i + b) > 0$$

b = 5



$$\text{ReLU}(\sum_i w_i x_i + b) = 0$$

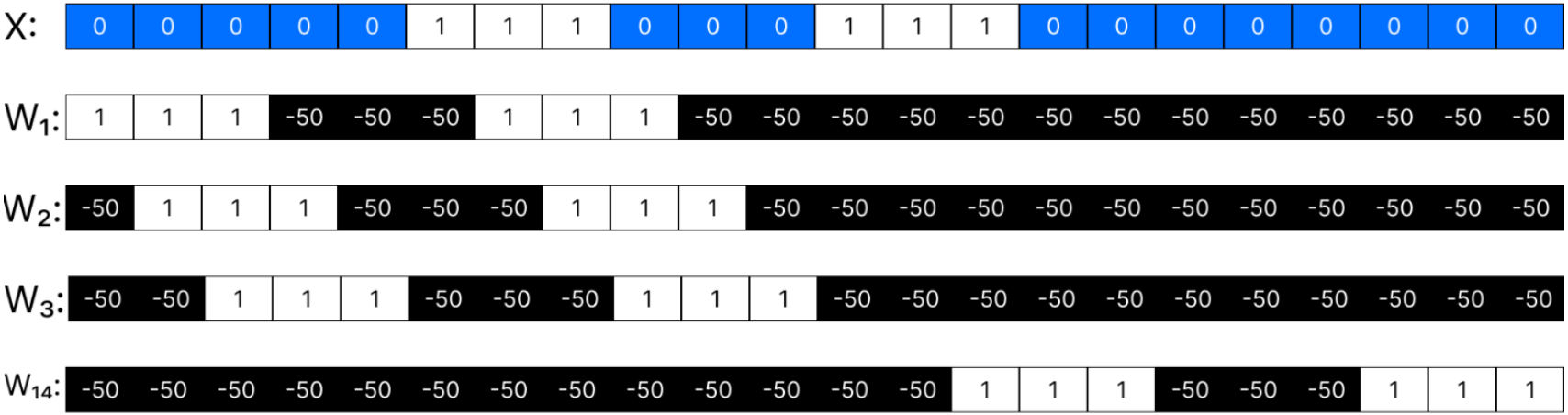
1D Example: Morse Code

What if sometimes the signal is delayed?

X: 

1D Example: Morse Code

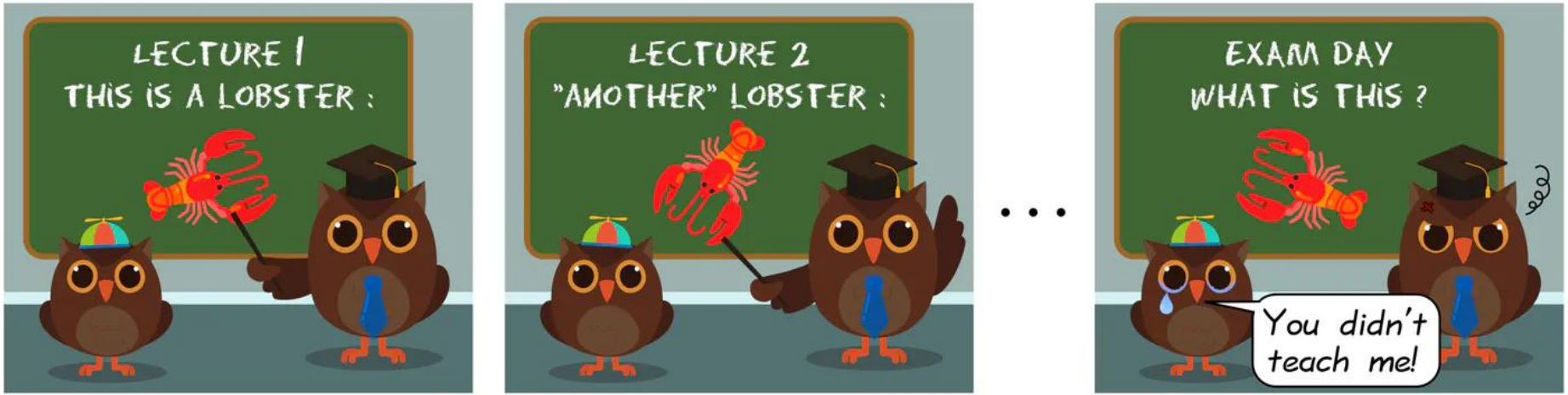
We can design a set of neurons encoding all possible positions, and employ a neuron from the next layer to perform “or” on top of those.



$$W_{15} = [1, 1, 1, \ldots, 1], \quad b_{15} = 0, \quad \text{i.e.} \quad A_{15} = \sum_{i=1}^{14} A_i > 0$$

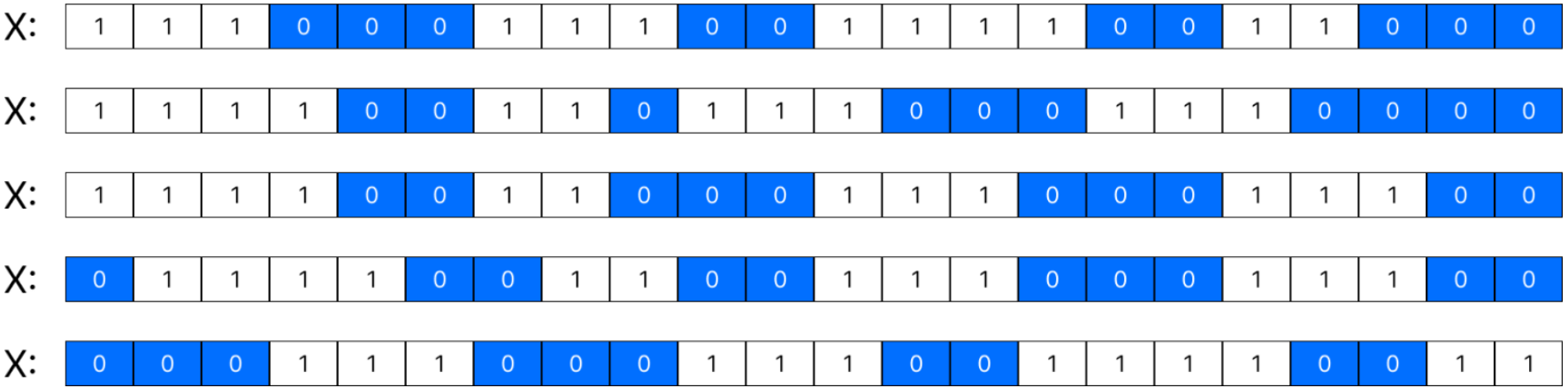
However

1. This requires lots of parameters
2. It requires training data featuring the signal in all given positions, so that the model has a chance to learn that it should structure its neurons like this
3. Even if for all patterns that we want to recognize, the data features all the positions, it will still be very hard to converge to the right solution



Even Worse

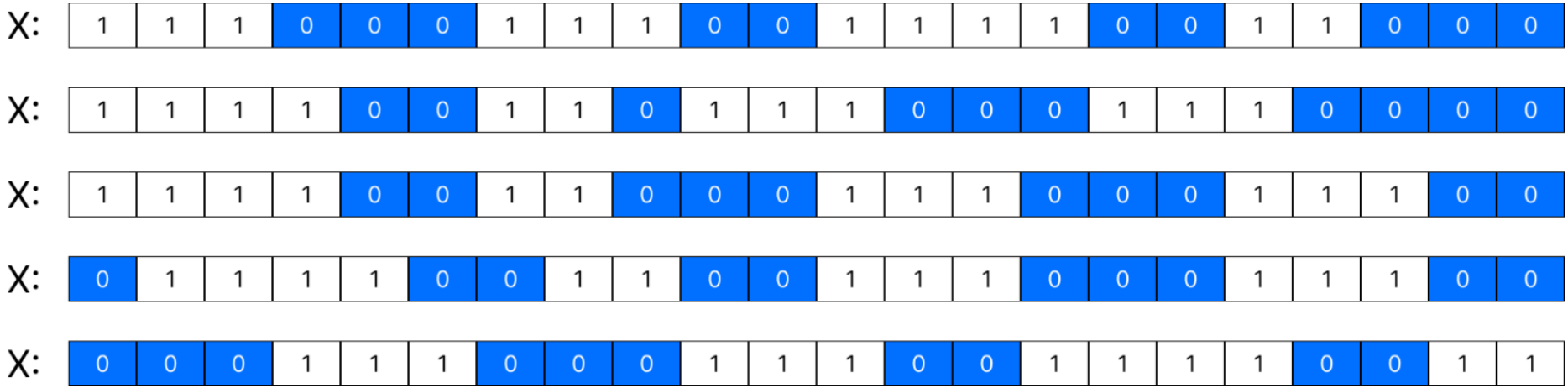
What if there are other patterns (words) we want to recognize? What if patterns can appear in any order, with any delay in between?



Invariance and Equivariance

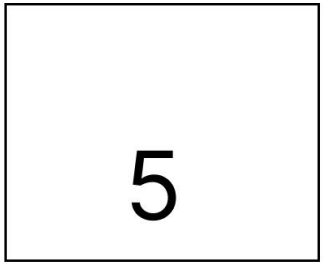
Invariance

A function f is invariant to a transformation t ⇔ f(x) = f(t(x)) for all x.

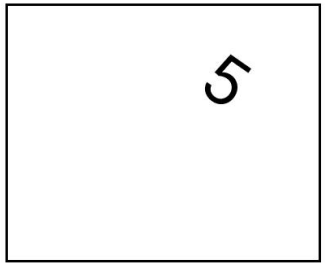


A model invariant to translation and permutation will predict the same output (the presence of both patterns) for all those inputs.

Invariance



{1, 2, 3, 4}



{3, 1, 4, 2}

A model invariant to

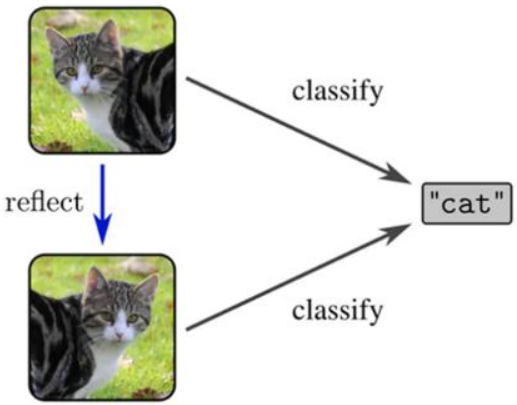
Translation

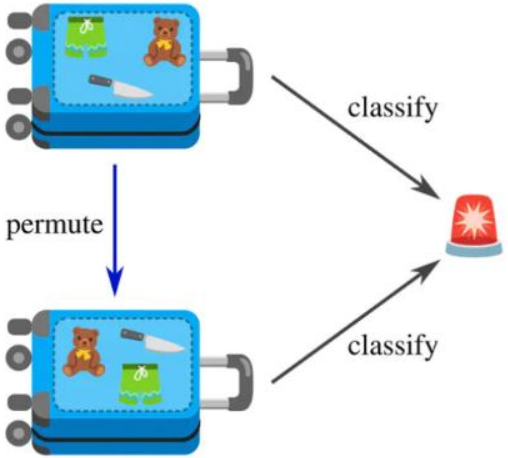
Translation, rotation, scaling

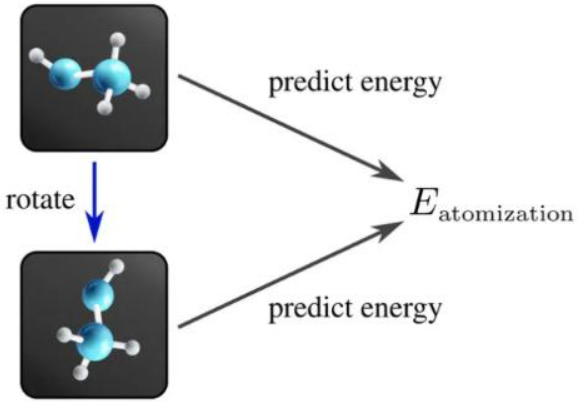
Permutation

would predict the same output for both elements in each column.

Invariance







source: https://maurice-weiler.gitlab.io/blog_post/cnn-book_1_equivariant_networks/

Invariance

classify

classify

Introduce human

Tinca Tinca Fish

Introduce cat

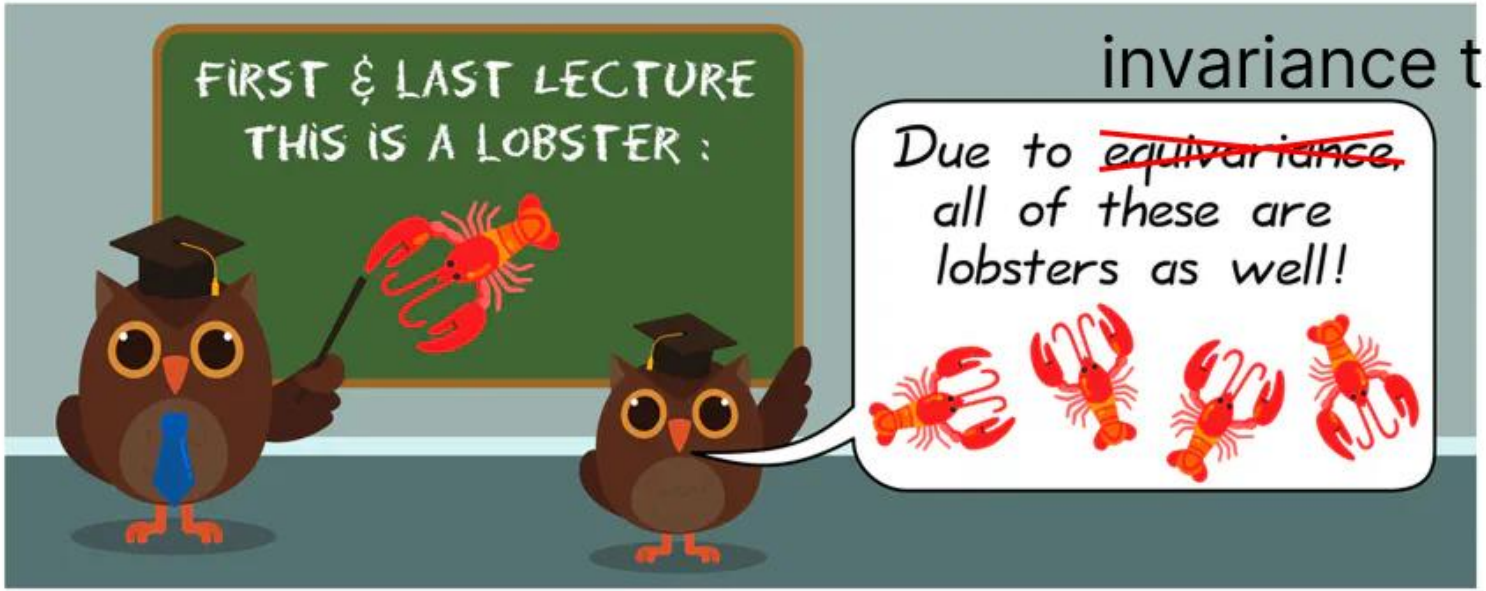
Human Sleeping

classify

classify

sources: ImageNet and https://www.landofcats.net/images/2016/11/Without-Cat-Vs.-With-Cat-Land-of-Cats.jpg

Ideal student



invariance to rotation

source for the original picture: https://maurice-weiler.gitlab.io/blog_post/cnn-book_1_equivariant_networks/

Equivariance

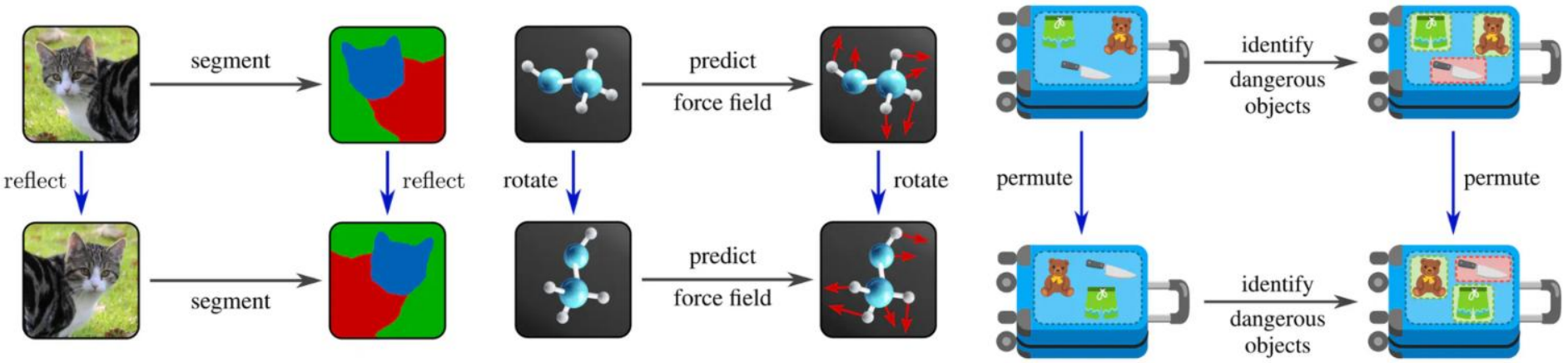
A function f is equivariant or covariant to a transformation t ⇔ f(t(x)) = t(f(x)) for all x.

A pixel-to-pixel network, such as an image segmentation module, should be equivariant to rotation, translation, flipping and object scaling.

Input

Output

Equivariance



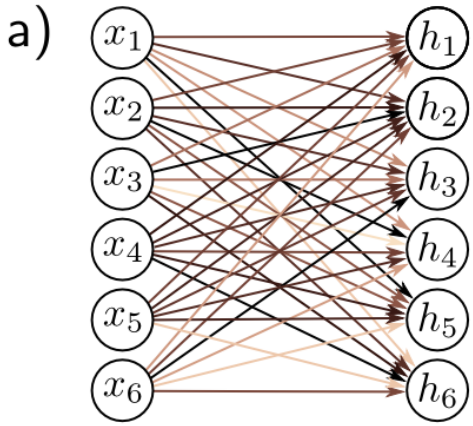
source: https://maurice-weiler.gitlab.io/blog_post/cnn-book_1_equivariant_networks/

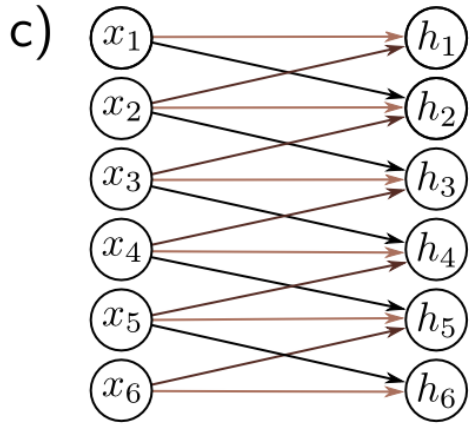
1D Convolution

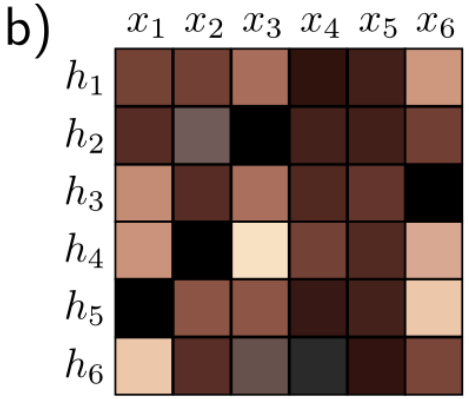
Back to our 1D example

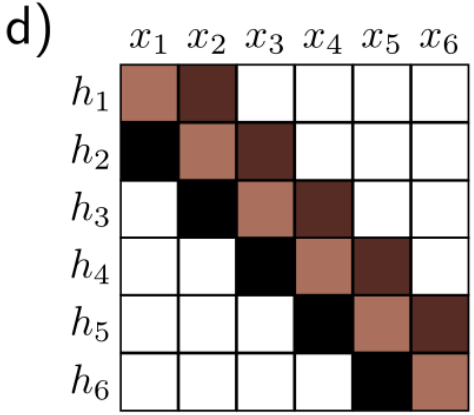
a) is depiction of a typical fully connected network with weights b)

c) is a depiction of our desired network behavior in which we search for a pattern by sliding it across the input; weights illustrated in d)



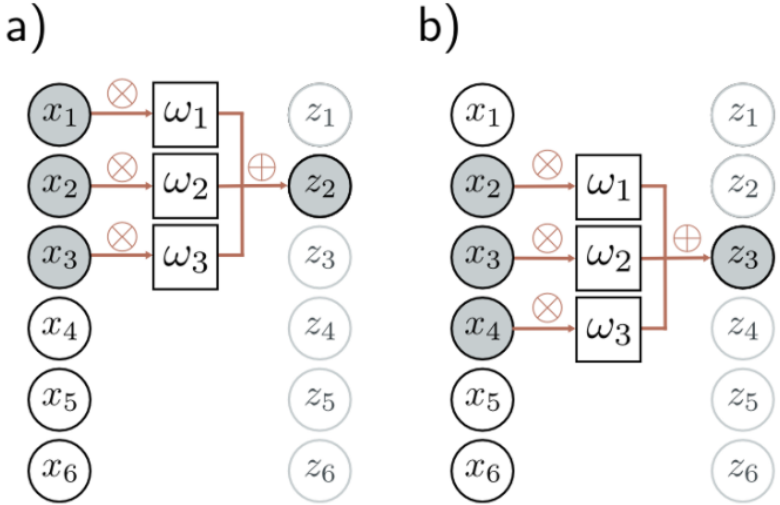






B

1D Convolution



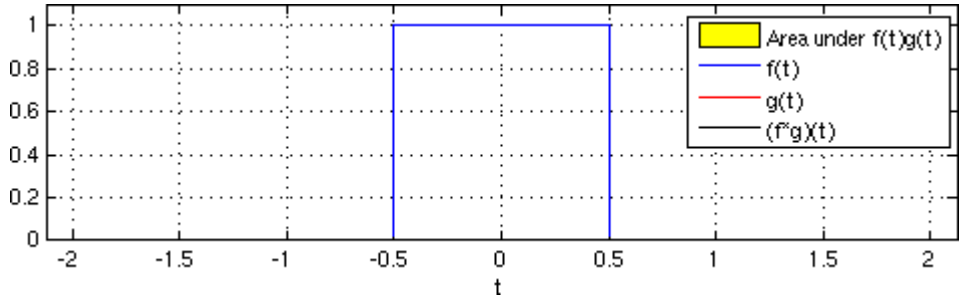
$$z_2 = w_1 x_1 + w_2 x_2 + w_3 x_3$$

$$z_i = w_1 x_{i-1} + w_2 x_i + w_3 x_{i+1}$$

The same weights are used at every position and are collectively called a convolution kernel or filter.
The convolution mechanism is equivariant to translation.

1D Convolution

Illustration for a 1D continuous signal:



Formula for a discrete convolution with kernel size 3:

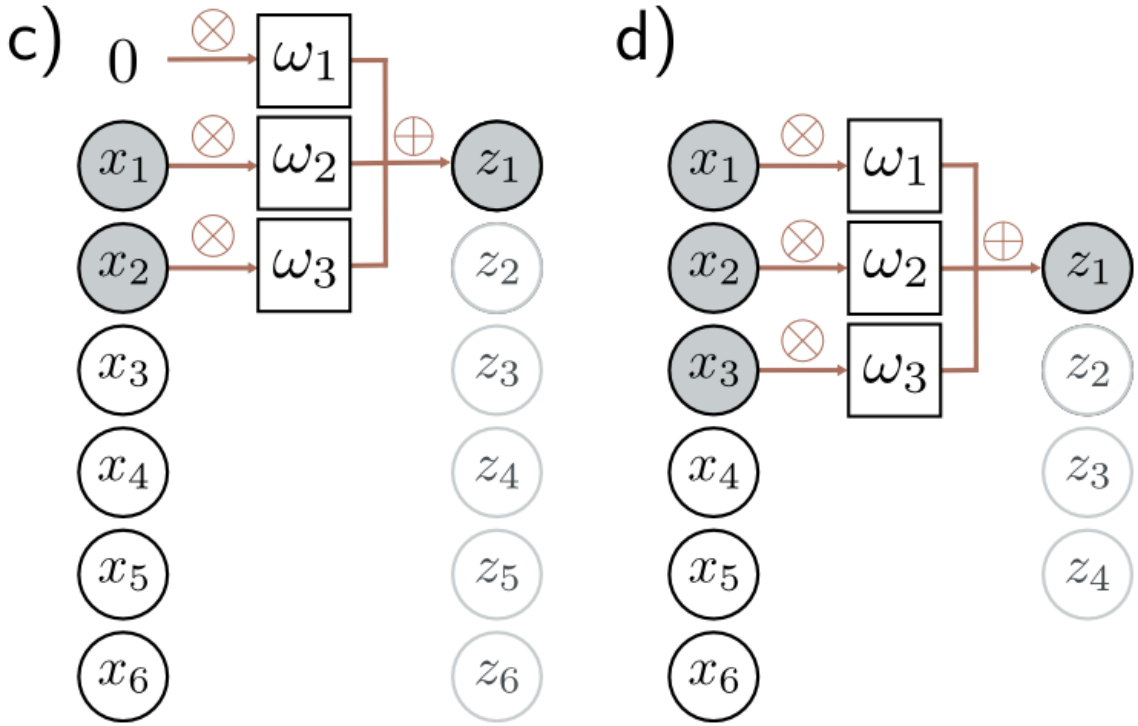
$$
\begin{aligned}
h_i &= \mathrm{a}\left[\beta + \omega_1 x_{i-1} + \omega_2 x_i + \omega_3 x_{i+1}\right] \\
&= \mathrm{a}\left[\beta + \sum_{j=1}^{3} \omega_j x_{i+j-2}\right],
\end{aligned}
$$

source: https://upload.wikimedia.org/wikipedia/commons/6/6e/Convolution_of_box_signal_with_itself.gif

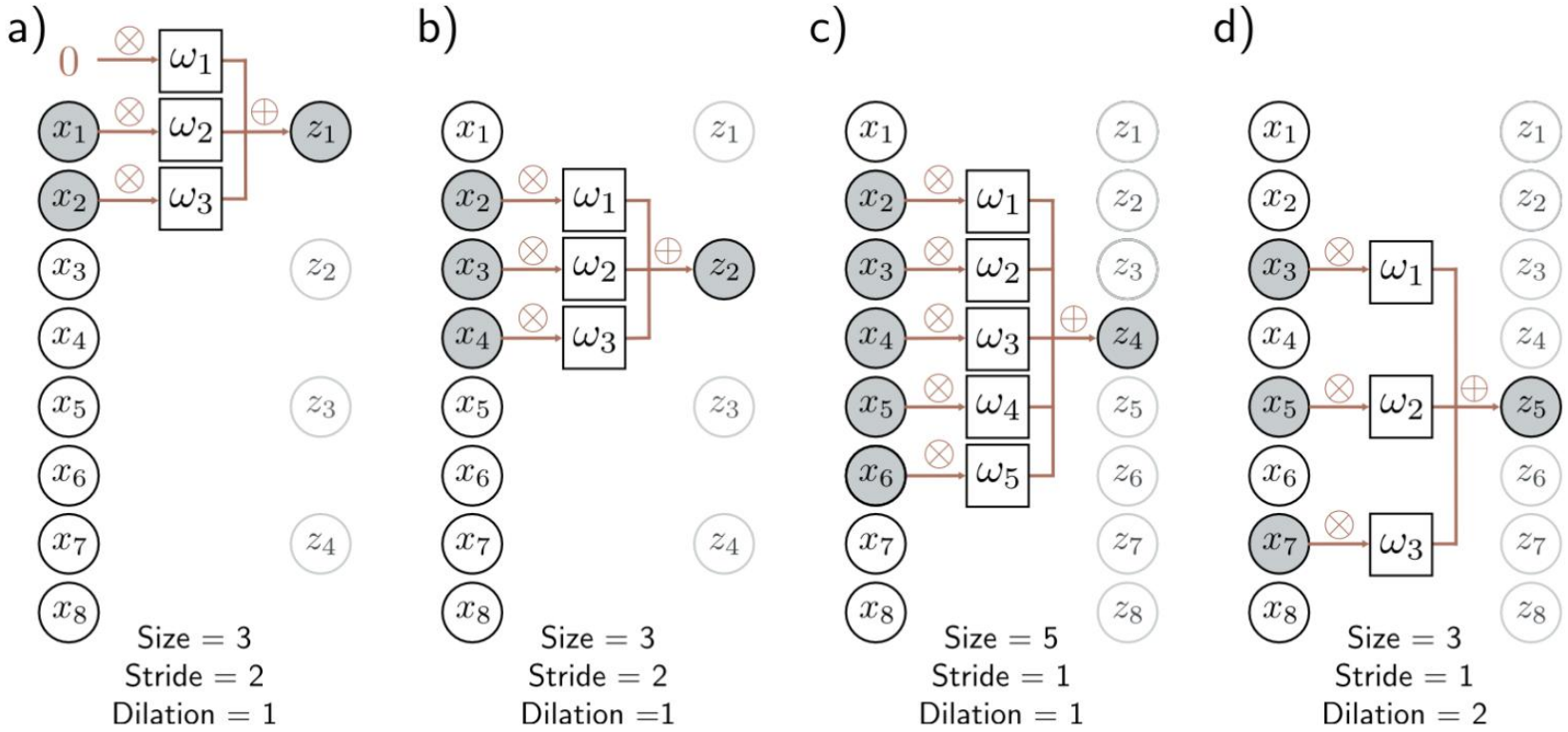
Padding

c) ‘Same’ vs d) ‘Valid’ padding

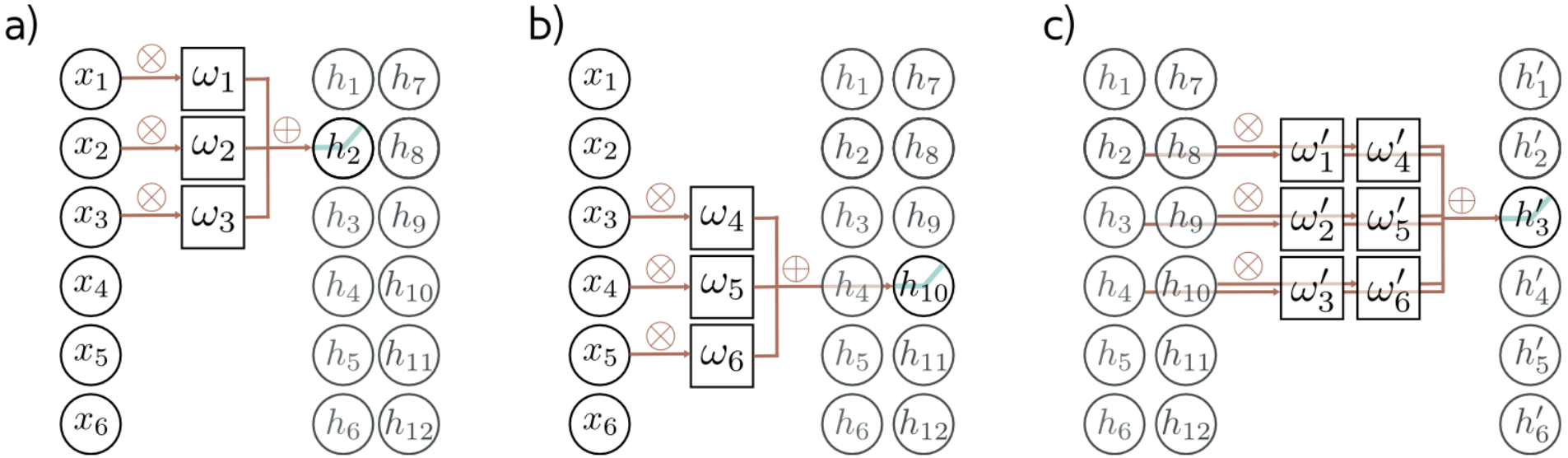
People almost always use the ‘same’ padding in order to preserve the size of the input.



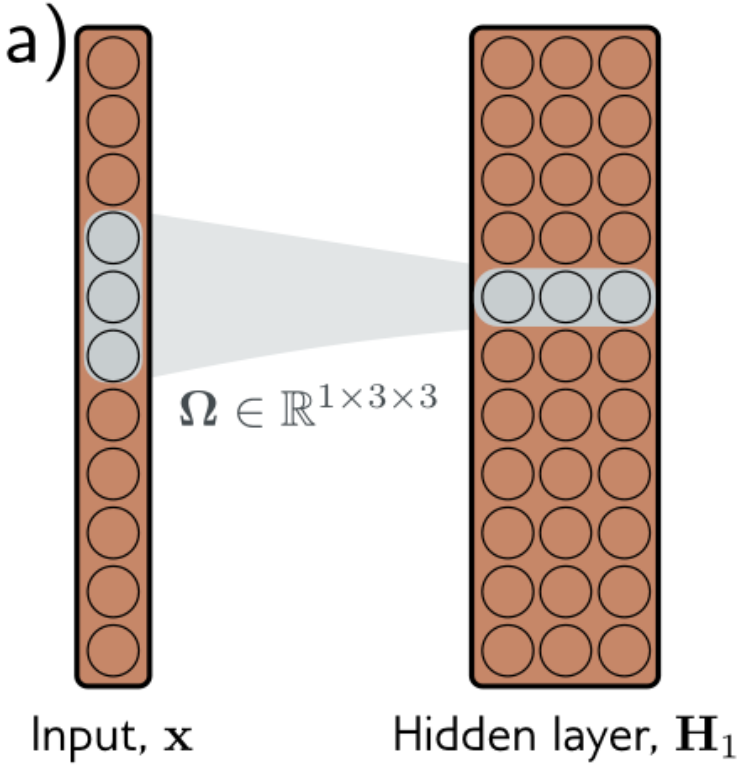
Size, Stride, Dilation



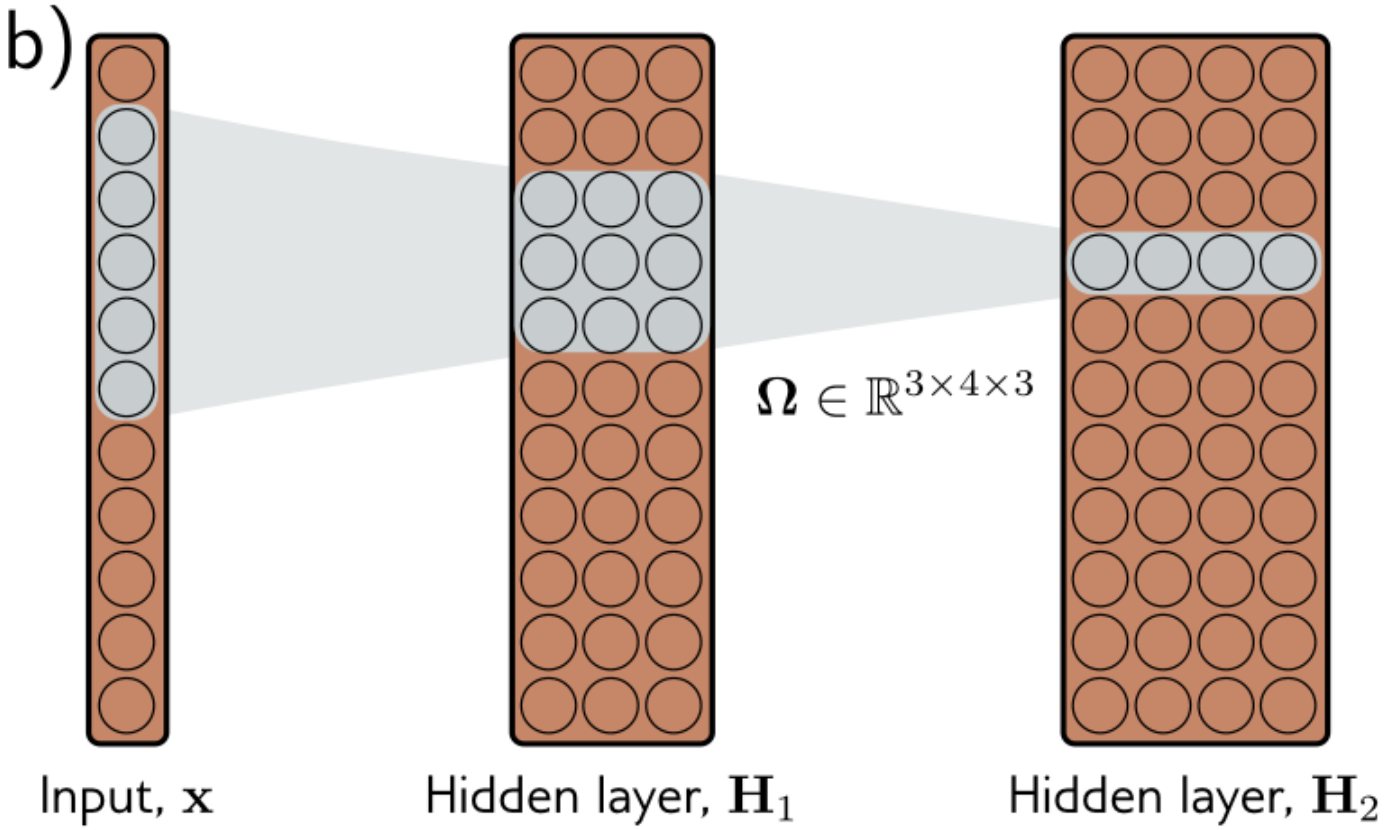
Channels


a)
x_1 x_2 x_3 x_4 x_5 x_6
ω_1 ω_2 ω_3
h_1 h_7 h_2 h_8 h_3 h_9 h_4 h_{10} h_5 h_{11} h_6 h_{12}
b)
x_1 x_2 x_3 x_4 x_5 x_6
ω_4 ω_5 ω_6
h_1 h_7 h_2 h_8 h_3 h_9 h_4 h_{10} h_5 h_{11} h_6 h_{12}
c)
h_1 h_7 h_2 h_8 h_3 h_9 h_4 h_{10} h_5 h_{11} h_6 h_{12}
ω'_1 ω'_4 ω'_2 ω'_5 ω'_3 ω'_6
h'_1 h'_2 h'_3 h'_4 h'_5 h'_6

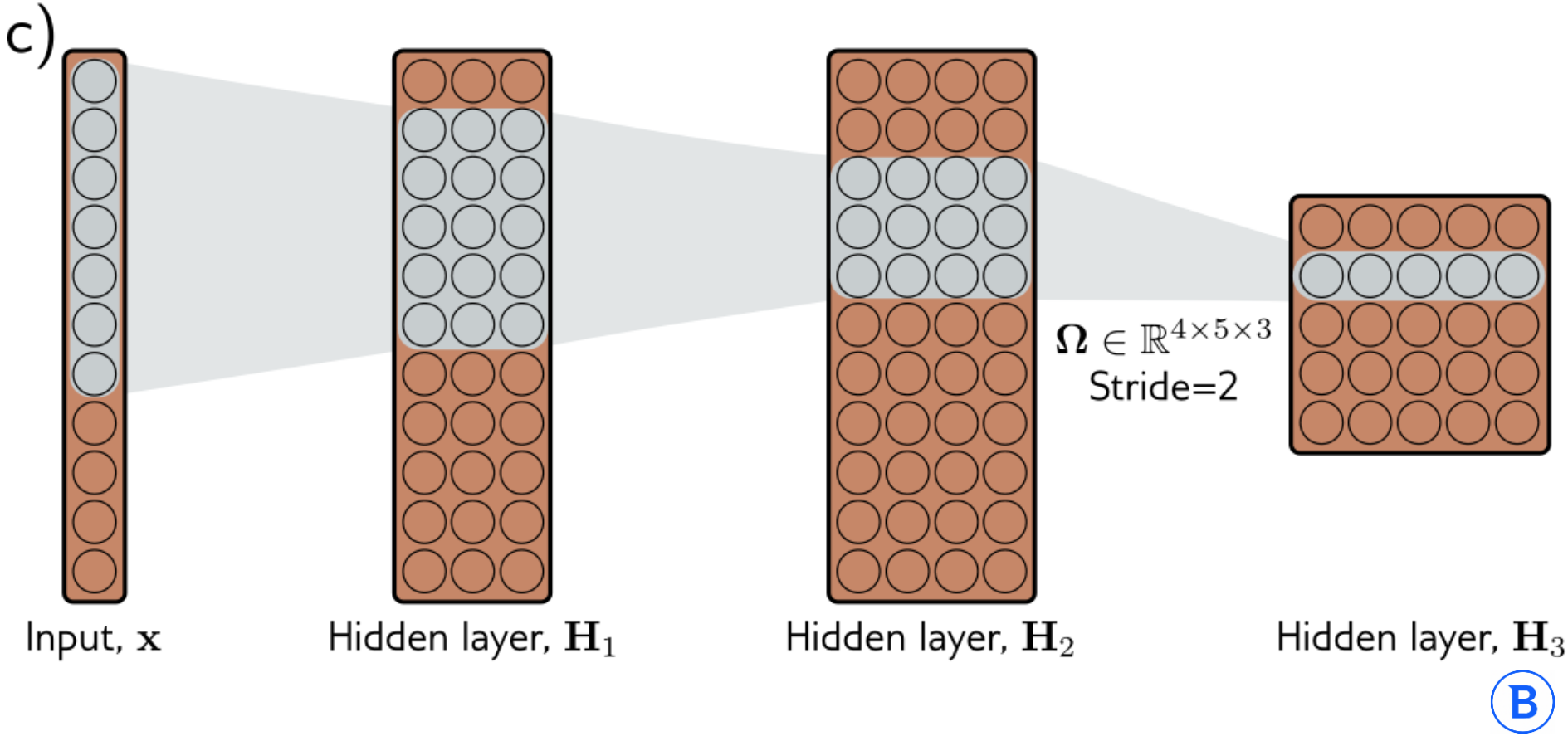
Receptive Fields



Receptive Fields

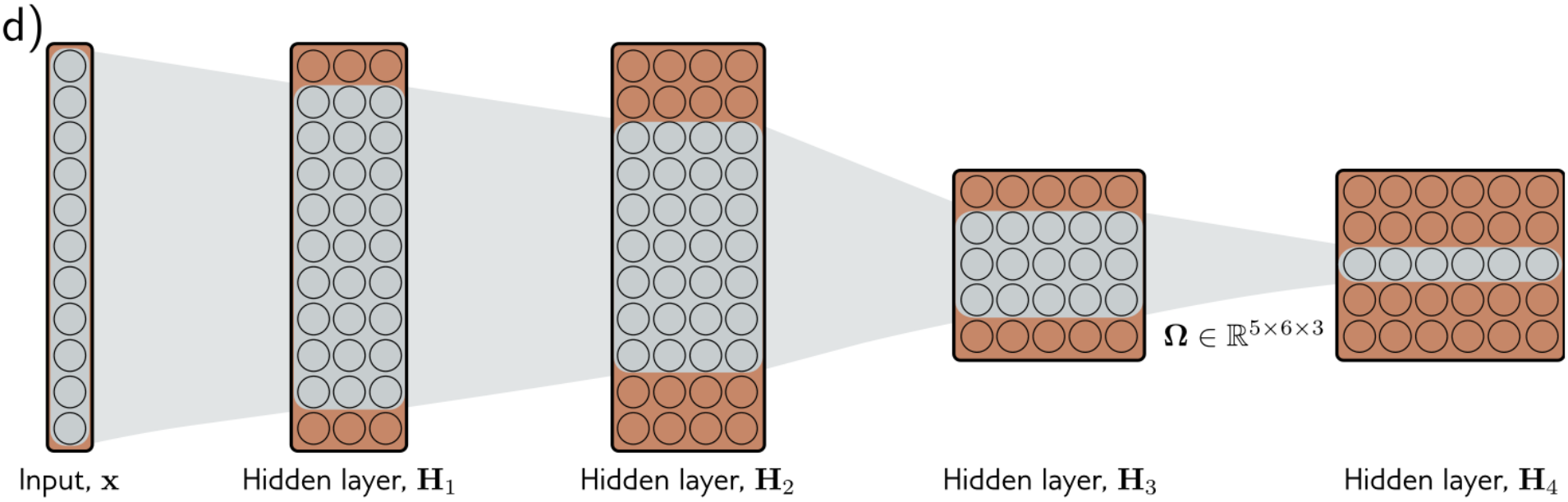


Receptive Fields

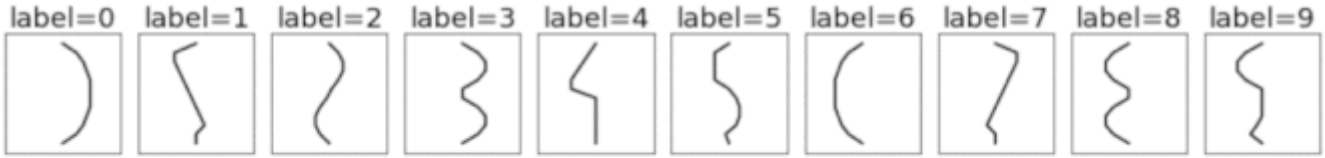


B

Receptive Fields

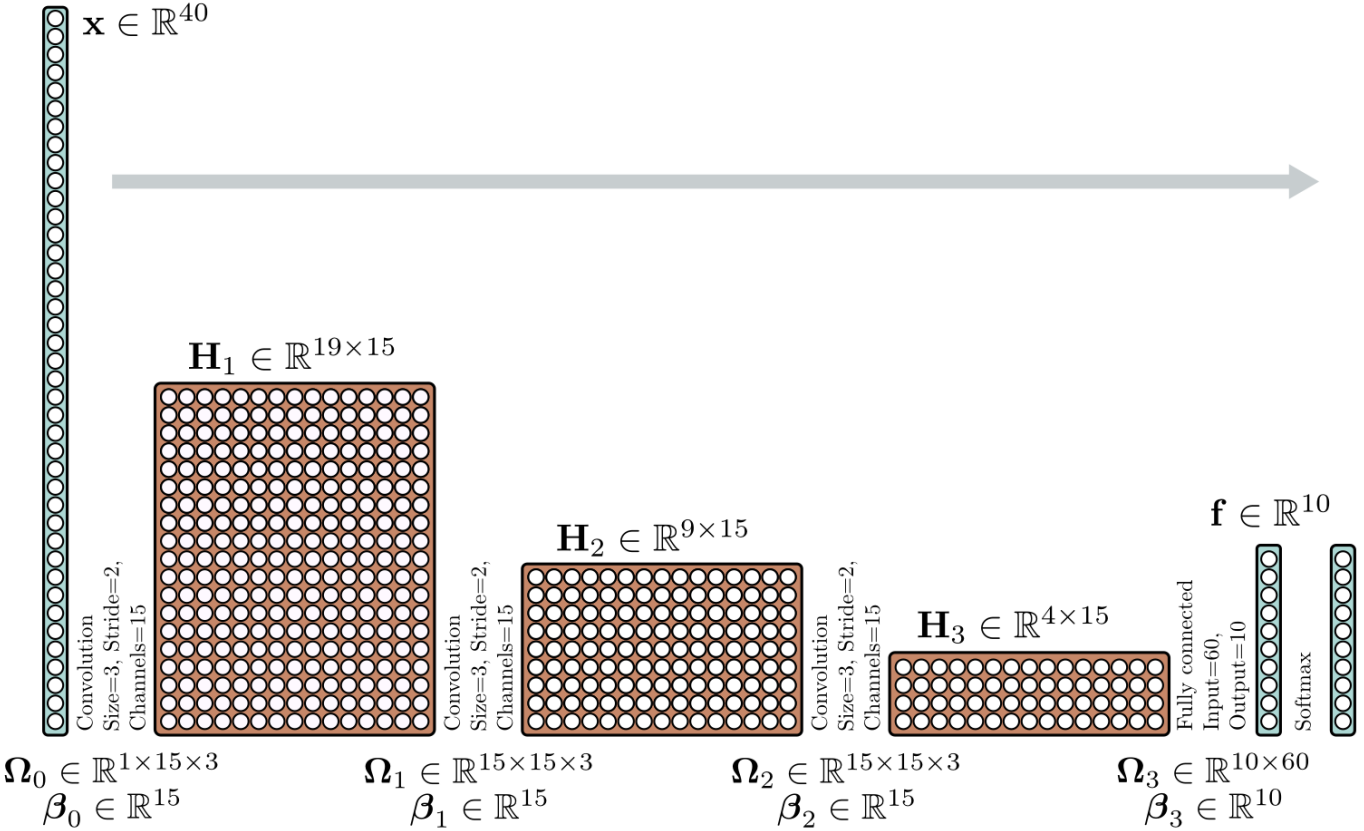


Application: 1D MNIST

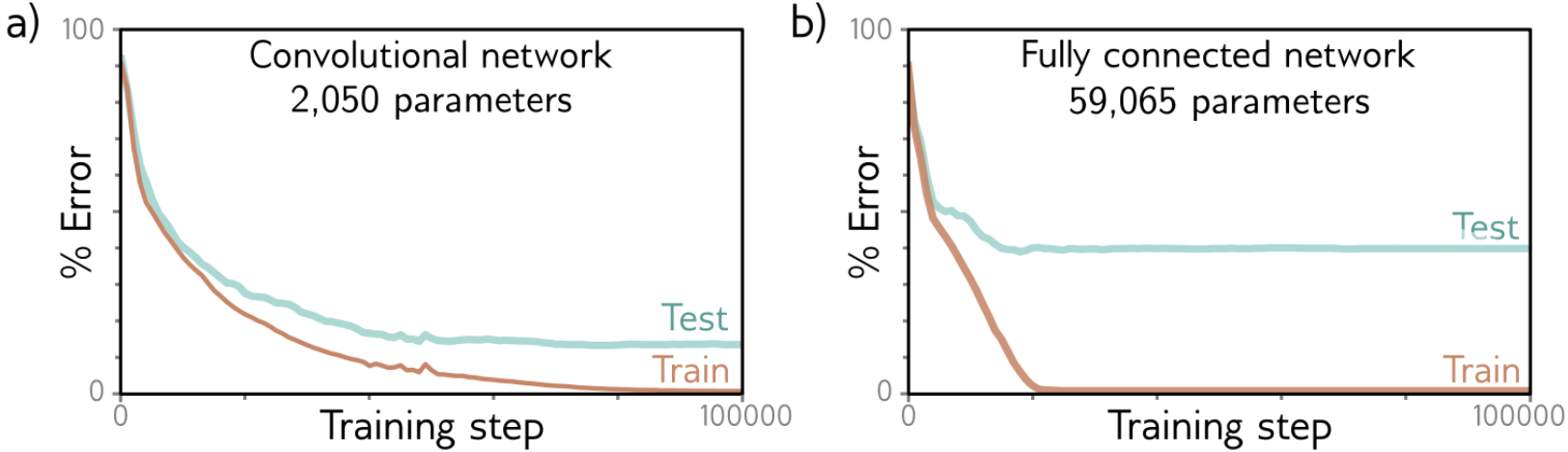


1. Each number is transformed in a 1D function (in this depiction, one value for each point on the Y axis)
2. Augmented (e.g. scaled, skewed)
3. Padded
4. Altered with noise
5. Augmented again

Application: 1D MNIST - Architecture



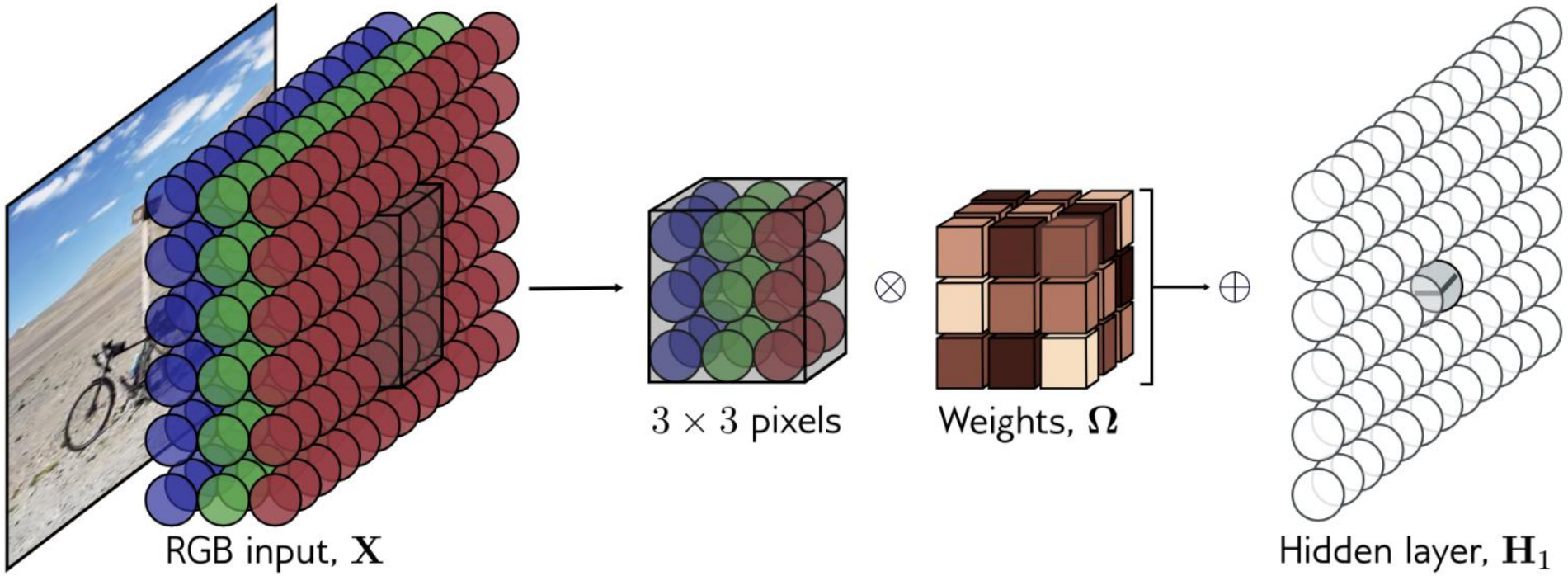
Application: 1D MNIST - Results



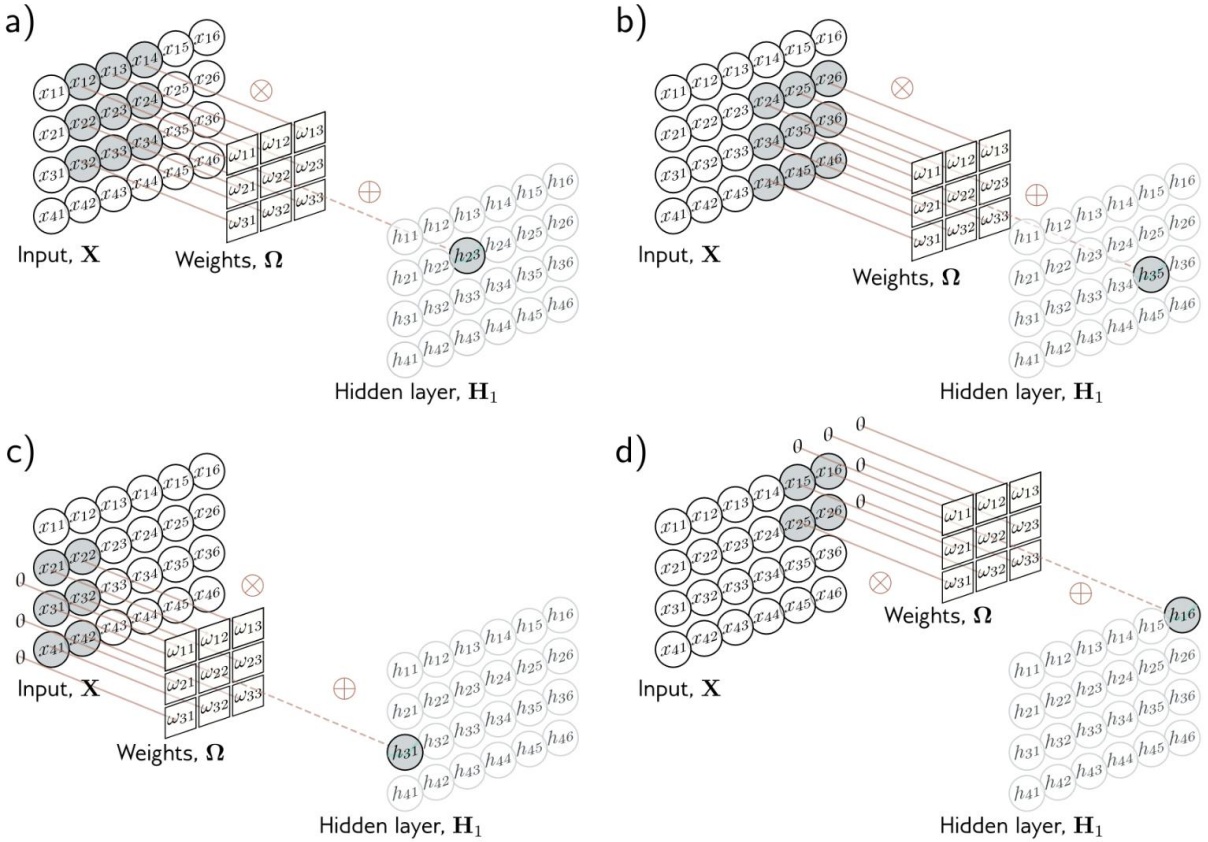
- 17% vs 40% test error
- almost 300 times less parameters
- even though the fully-connected architecture has the structure and enough parameters to replicate the convolutional model, the model does not converge to the desired solution given the allotted data
- the strong inductive bias helps reduce the amount of parameters needed and significantly improves the generalization capabilities of the model

2D Convolution

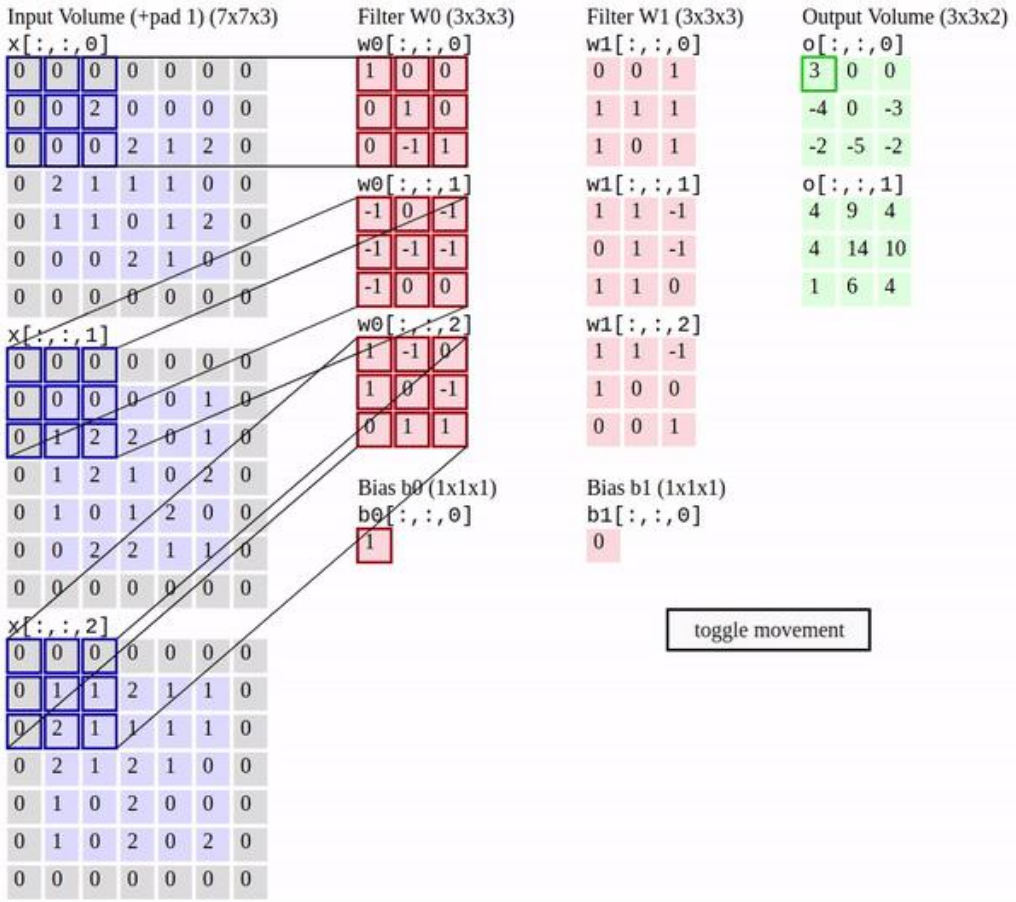
2D Convolution



2D Convolution



2D Convolution



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

Kernel Examples

X – Direction Kernel

-1	0	1
-2	0	2
-1	0	1

Vertical edges

Y – Direction Kernel

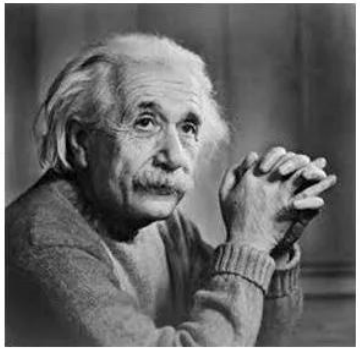
-1	-2	-1
0	0	0
1	2	1

Horizontal edges

source: https://datahacker.rs/edge-detection/

Kernel Examples

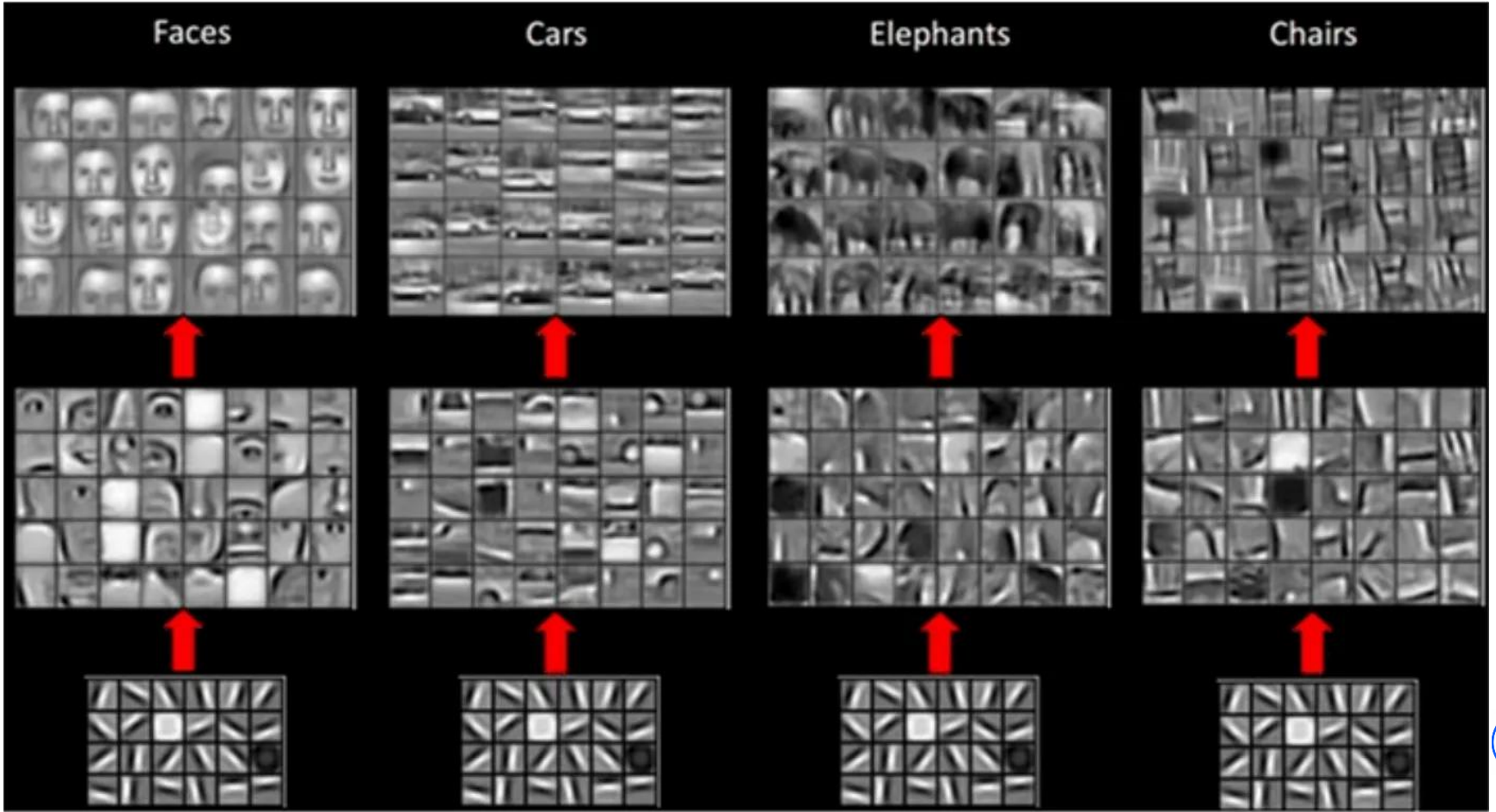
Vertical Edge Detection



Horizontal Edge Detection

source: https://mlarchive.com/deep-learning/understanding-convolutional-neural-networks/

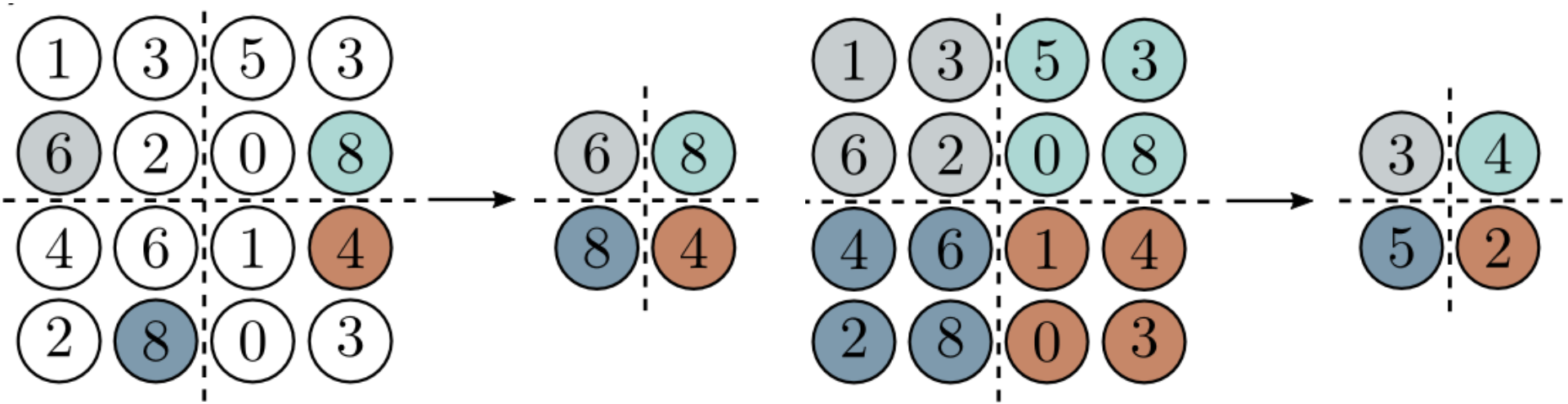
Convolutional Neural Networks



Convolutional Neural Networks

- The deeper the layer the more abstract the feature
- Few initial patterns to recognize
- Enormous amounts of feature combinations to recognize in the abstract layers
- The number of channels thus scales with depth => more data to process
- There is also a lot of spatial redundancy, as the abstract features encode a larger area

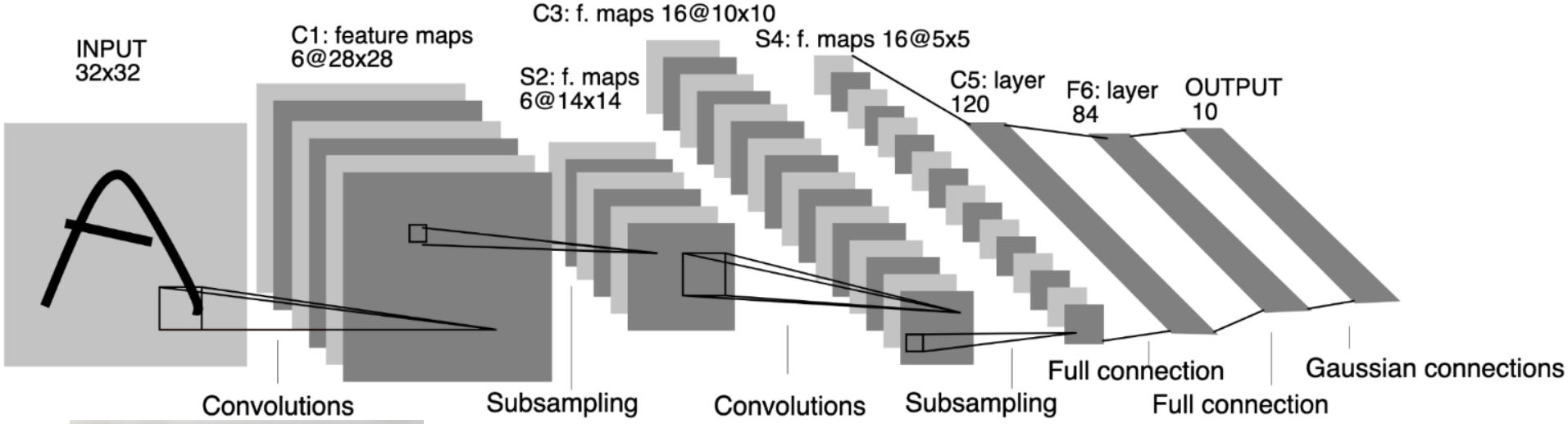
2D Downsampling



Max Pooling (2x2)

Average Pooling (2x2)

Convolutional Neural Networks

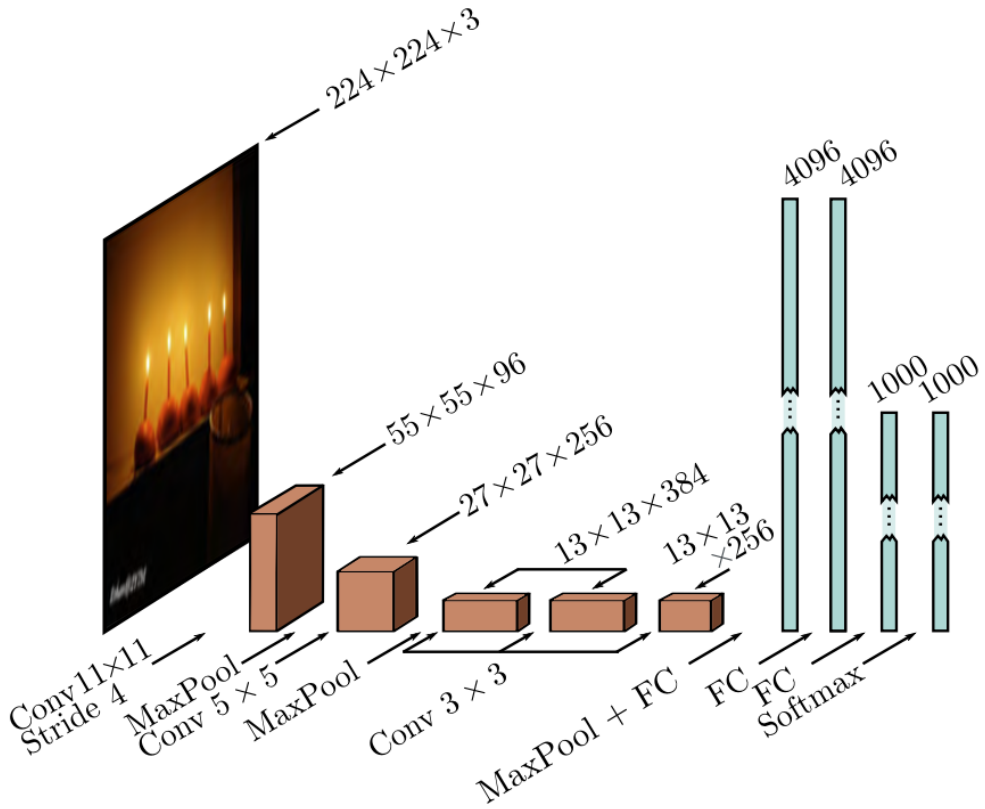


CNNs were introduced by Yann LeCun in 1989 in the NeurIPS paper ‘Handwritten Digit Recognition with a Back-Propagation Network’.
Image of LeNet, introduced by Yann LeCun in 1998, in ‘Gradient Based Learning Applied to Document Recognition’.

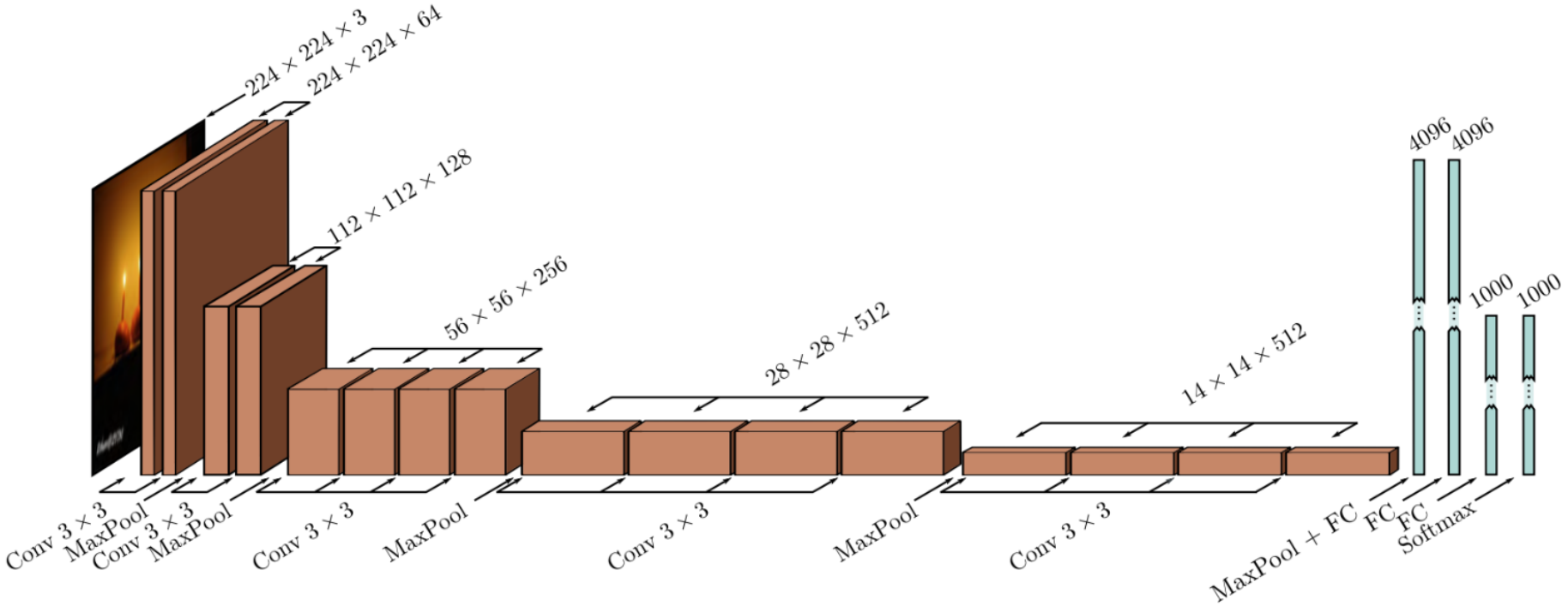
sources: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf ,
https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf

The Start of the Deep Learning Era

AlexNet (2012) the first neural network to exceed human performance on the ImageNet benchmark (1,2M images, 1K classes).



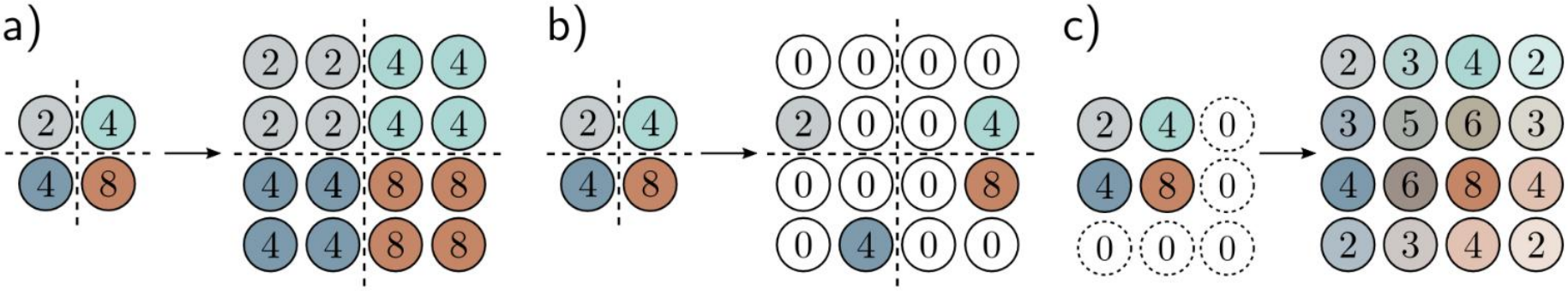
General CNN Architecture



VGG (2014): 3x3 kernels, 'same' padding, 2x2 MaxPooling, number of channels increased with depth.

Upsampling

Upsampling techniques



Replicate Values

Max Unpooling (saving max pooling positions)

Bilinear interpolation

Transposed Convolution

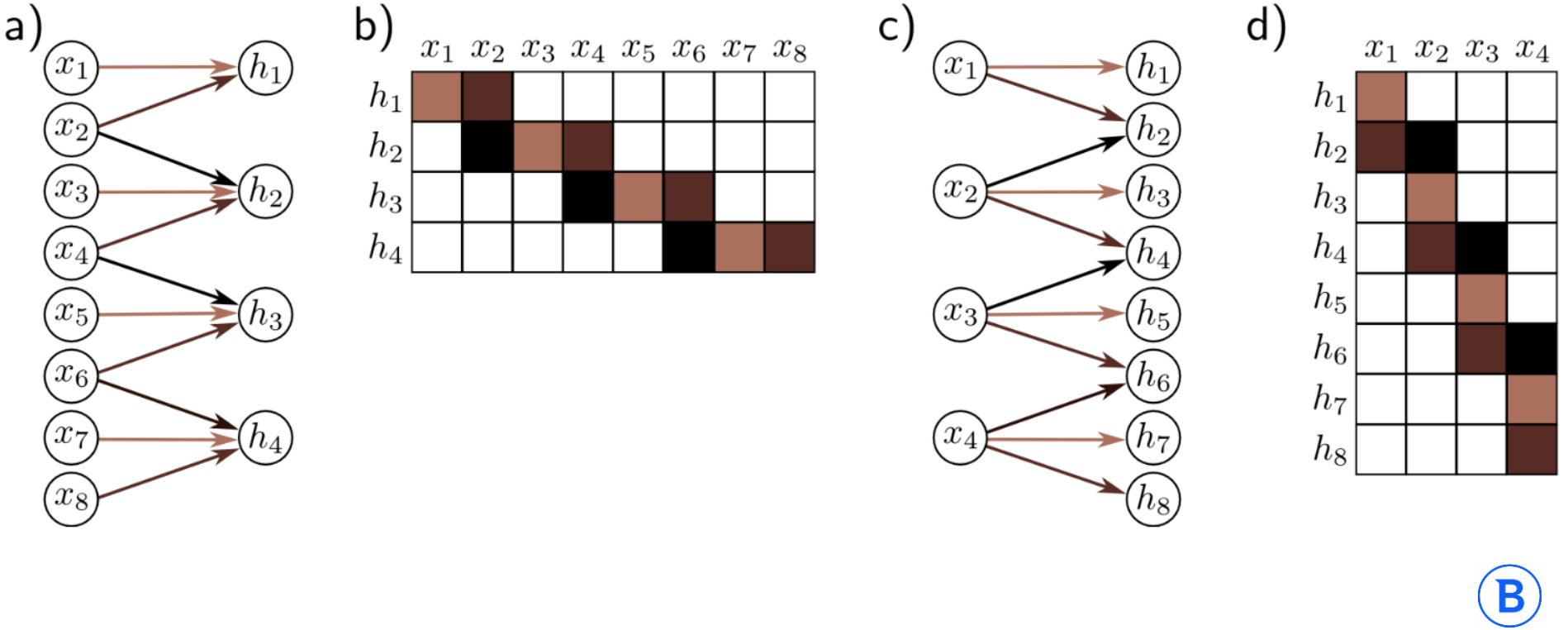
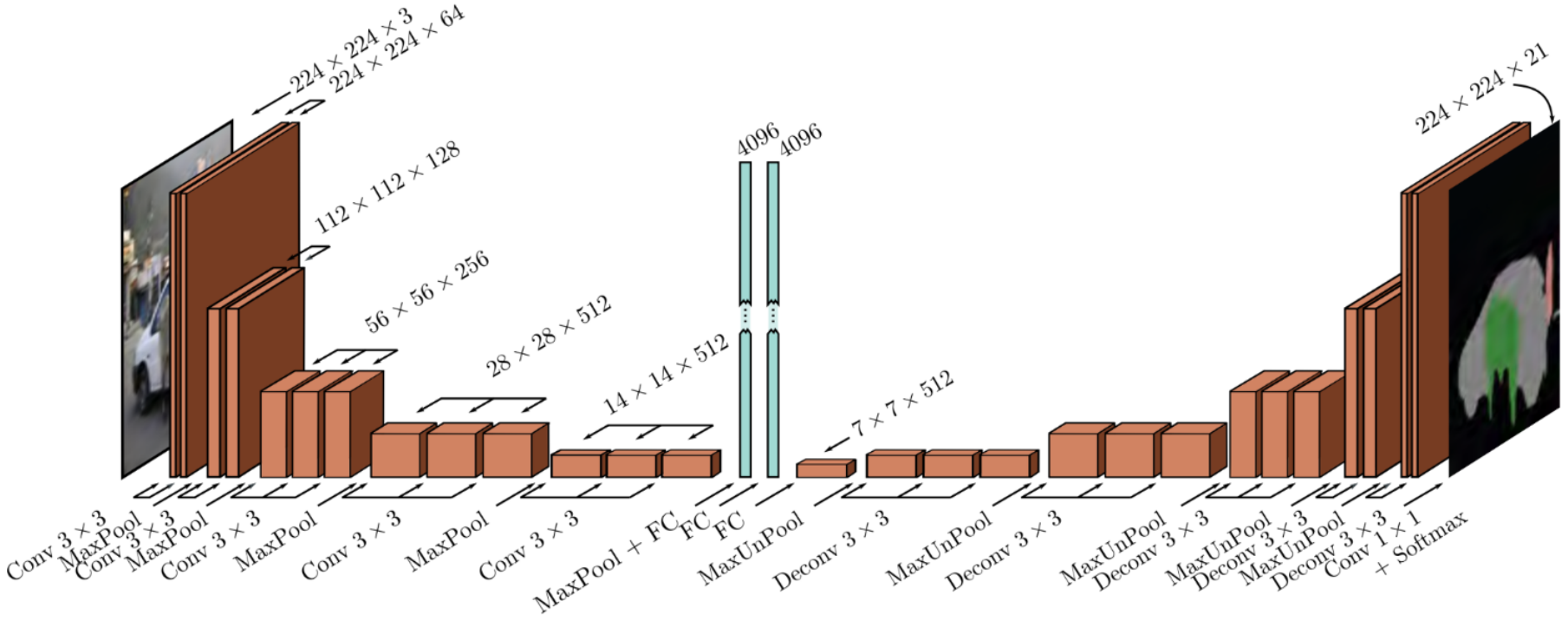


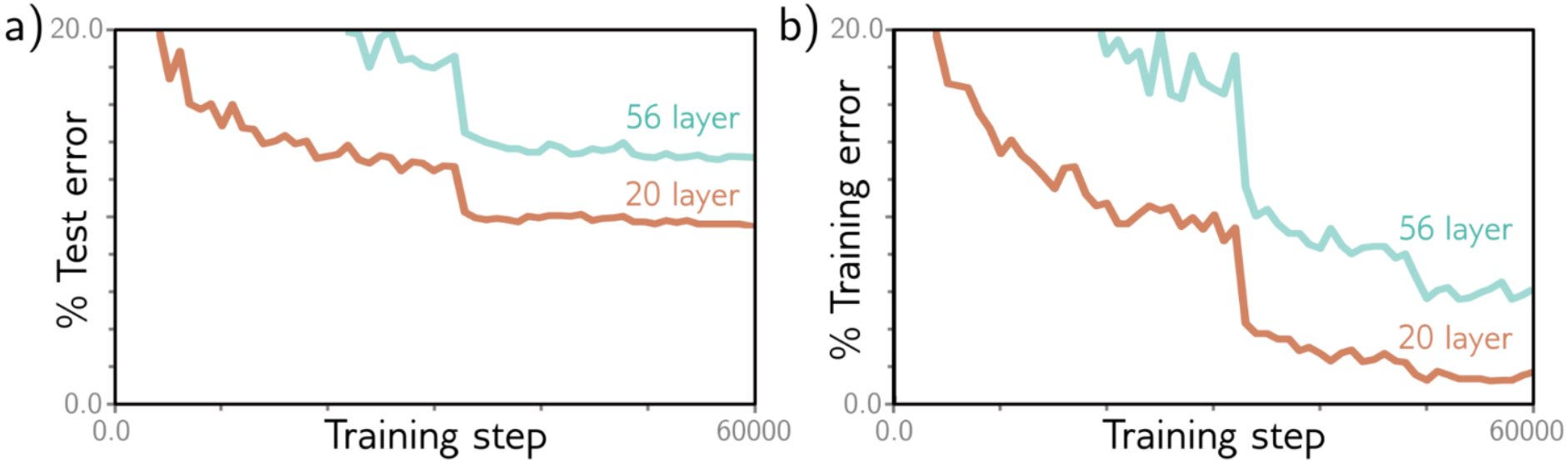
Image Segmentation Architecture



Noh, H., Hong, S., & Han, B. (ICCV 2015). Learning deconvolution network for semantic segmentation.

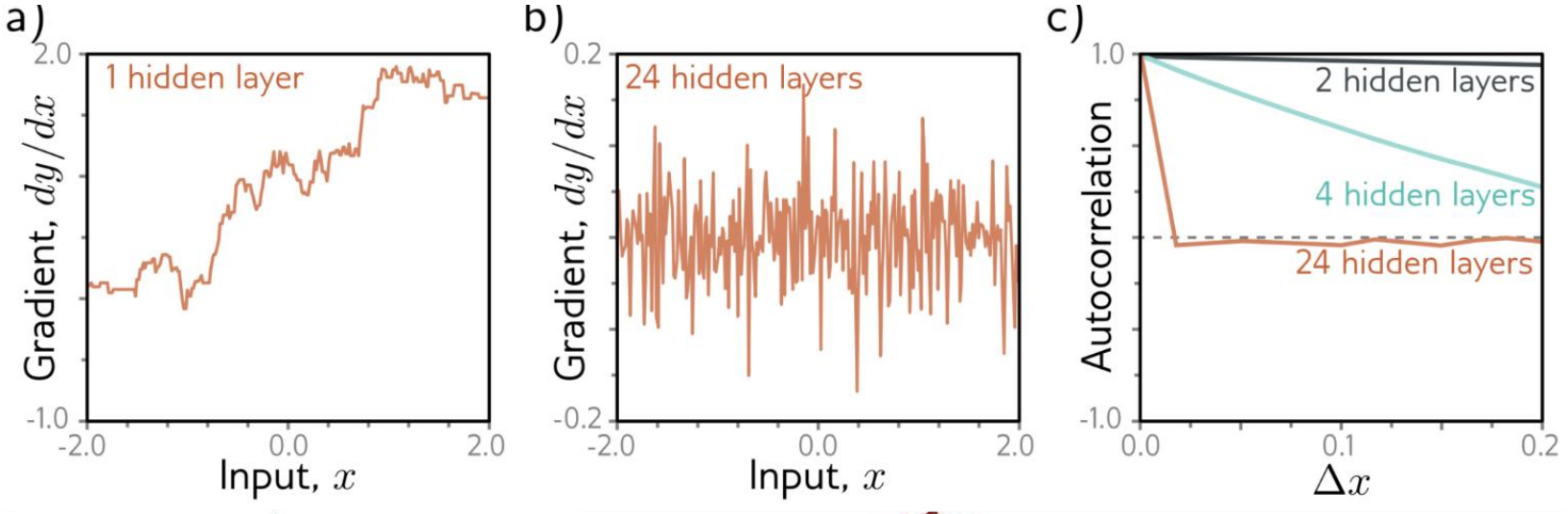
Scaling Up

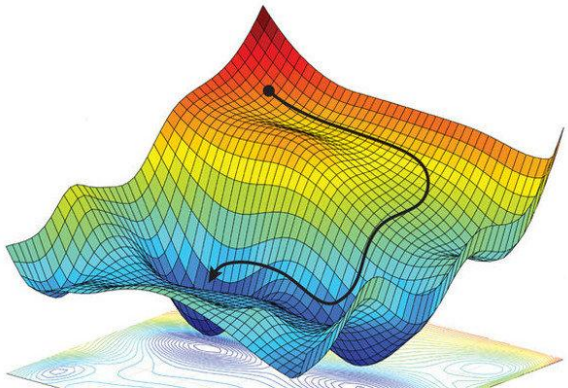
Adding More Layers

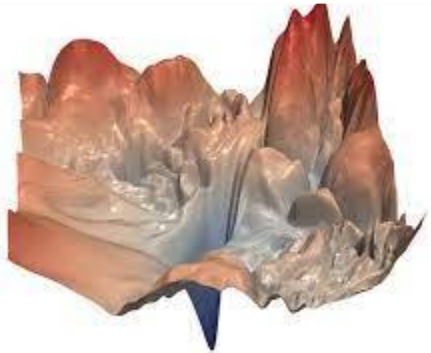


CNN performance on CIFAR 10. From He et al, 2015, “Deep Residual Learning for Image Recognition”.

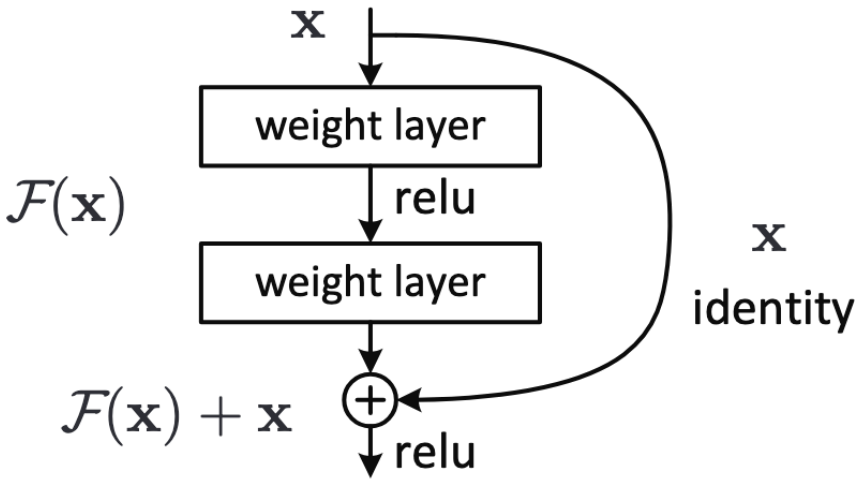
Shattered Gradients





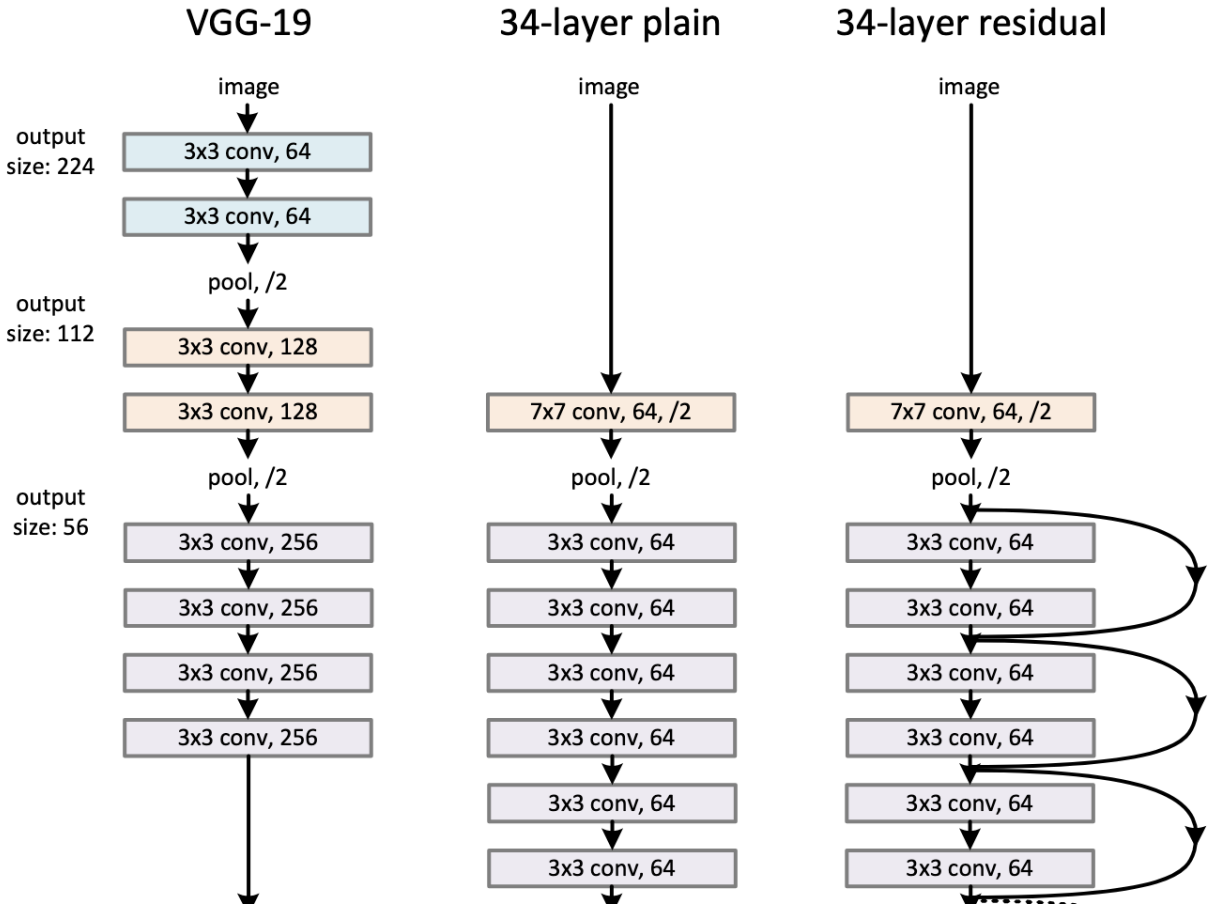


Residual Connections



$$
\begin{aligned}
\mathbf{h}_1 &= \mathbf{x} + \mathbf{f}_1[\mathbf{x}, \boldsymbol{\phi}_1] \\
\mathbf{h}_2 &= \mathbf{h}_1 + \mathbf{f}_2[\mathbf{h}_1, \boldsymbol{\phi}_2] \\
\mathbf{h}_3 &= \mathbf{h}_2 + \mathbf{f}_3[\mathbf{h}_2, \boldsymbol{\phi}_3] \\
\mathbf{y} &= \mathbf{h}_3 + \mathbf{f}_4[\mathbf{h}_3, \boldsymbol{\phi}_4],
\end{aligned}
$$

Residual Connections



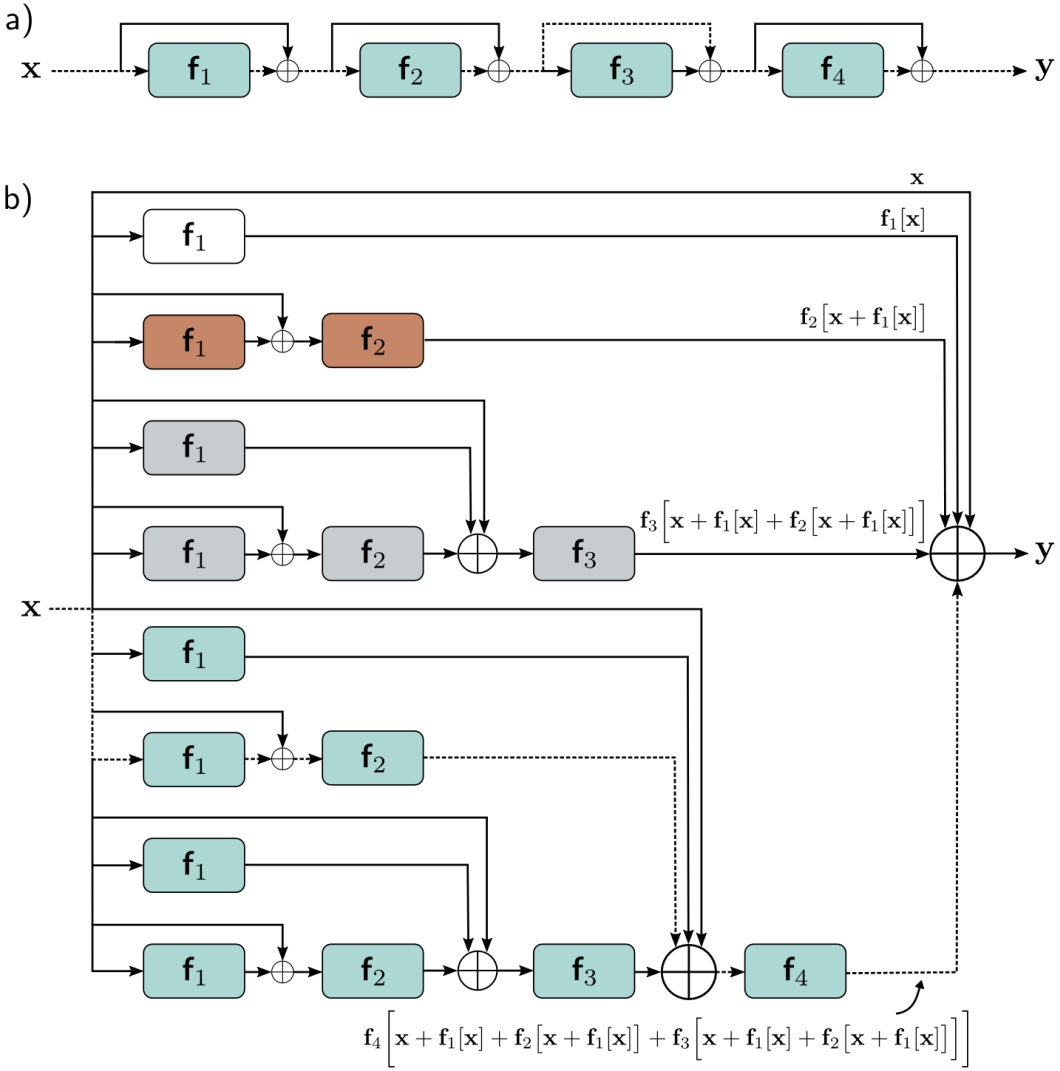
Residual Connections

$$
\begin{aligned}
\mathbf{h}_1 &= \mathbf{x} + \mathbf{f}_1[\mathbf{x}, \boldsymbol{\phi}_1] \\
\mathbf{h}_2 &= \mathbf{h}_1 + \mathbf{f}_2[\mathbf{h}_1, \boldsymbol{\phi}_2] \\
\mathbf{h}_3 &= \mathbf{h}_2 + \mathbf{f}_3[\mathbf{h}_2, \boldsymbol{\phi}_3] \\
\mathbf{y} &= \mathbf{h}_3 + \mathbf{f}_4[\mathbf{h}_3, \boldsymbol{\phi}_4],
\end{aligned}
$$

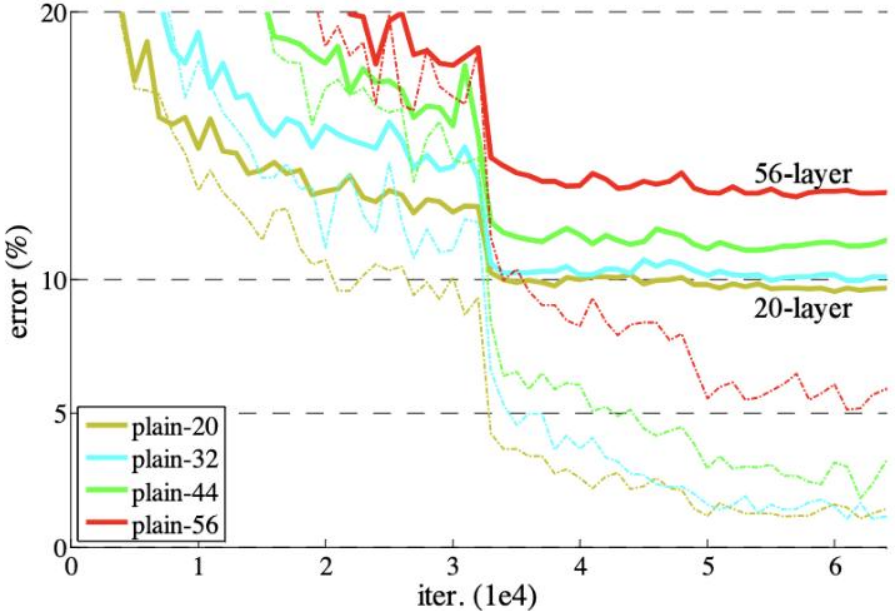
$$
\begin{aligned}
\mathbf{y} = \mathbf{x} &+ \mathbf{f}_1[\mathbf{x}] \\
&+ \mathbf{f}_2\Big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]\Big] \\
&+ \mathbf{f}_3\Big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}] + \mathbf{f}_2\Big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]\Big]\Big] \\
&+ \mathbf{f}_4\Big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}] + \mathbf{f}_2\Big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]\Big] + \mathbf{f}_3\Big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}] + \mathbf{f}_2\Big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]\Big]\Big]\Big],
\end{aligned}
$$

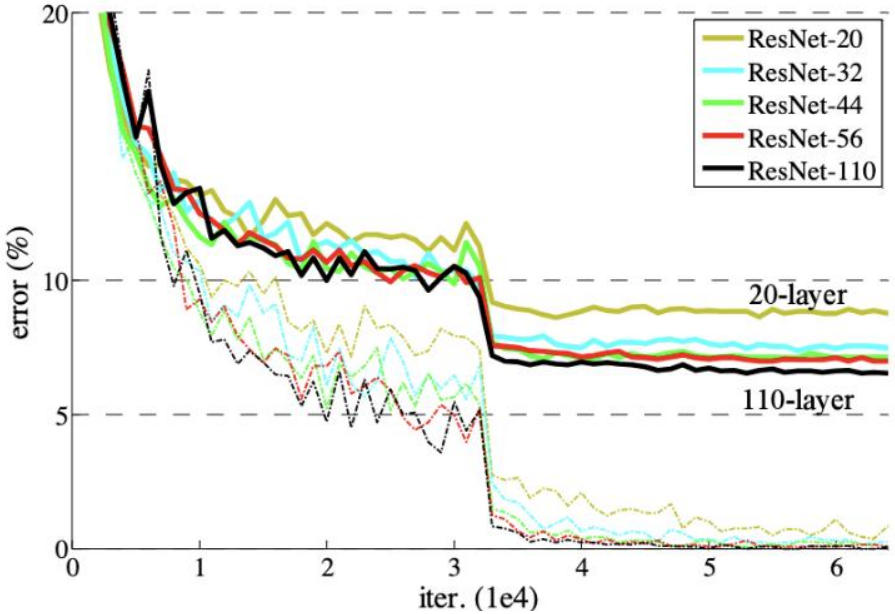
Residual Connections

Basically, a gradient highway.



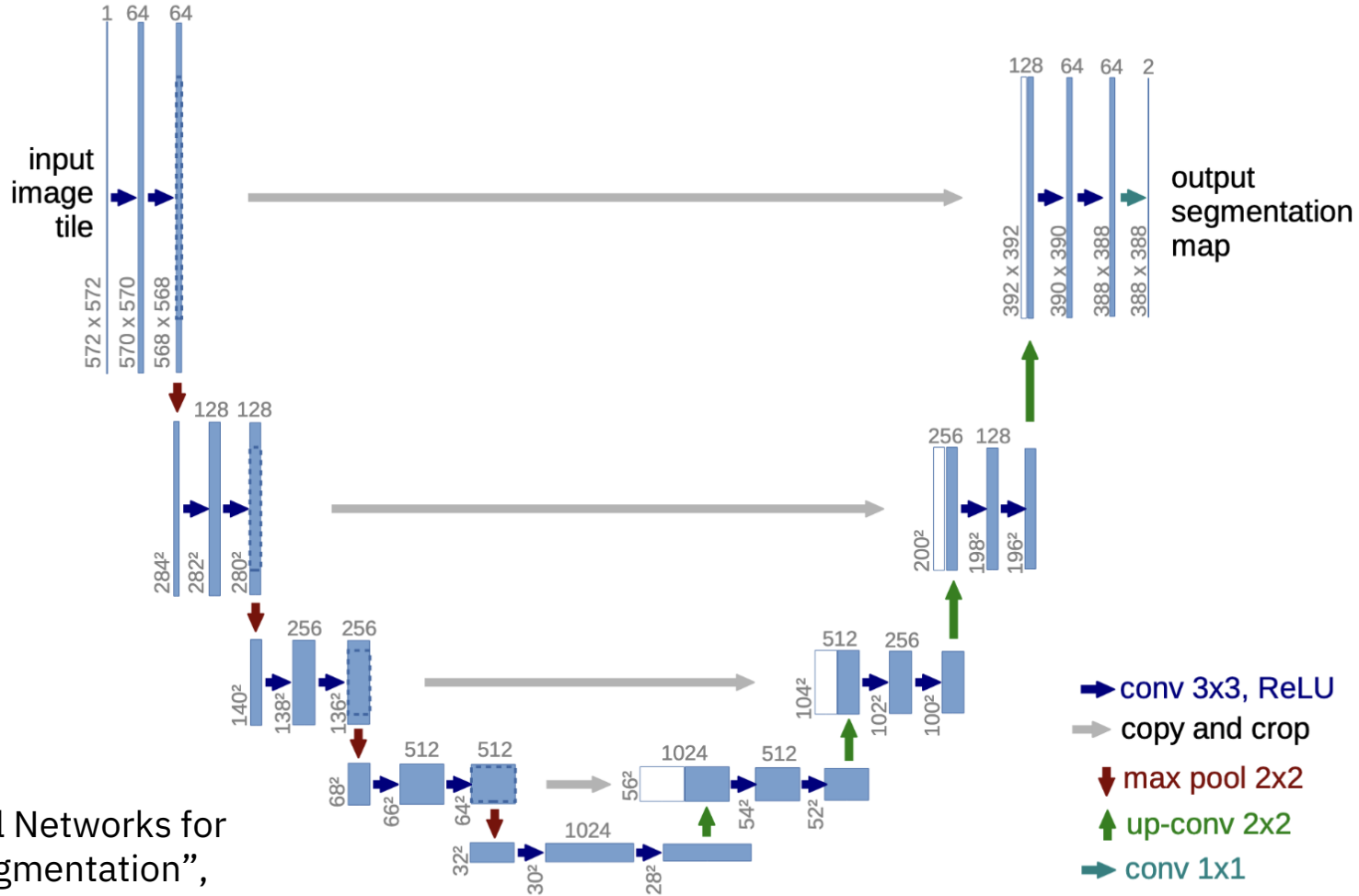
Residual Connections





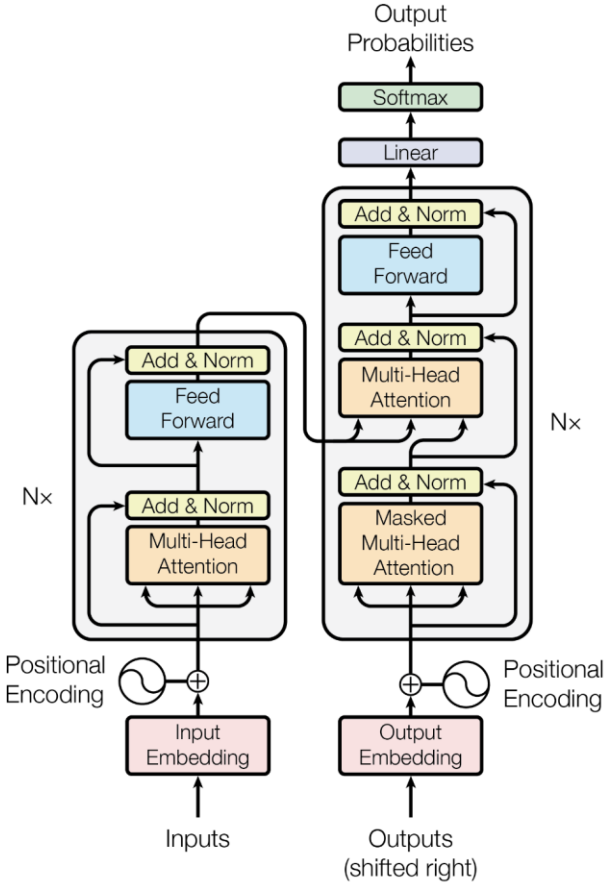
Results on CIFAR10. Bold lines present the test loss while dotted lines depict the training loss.

U-Net

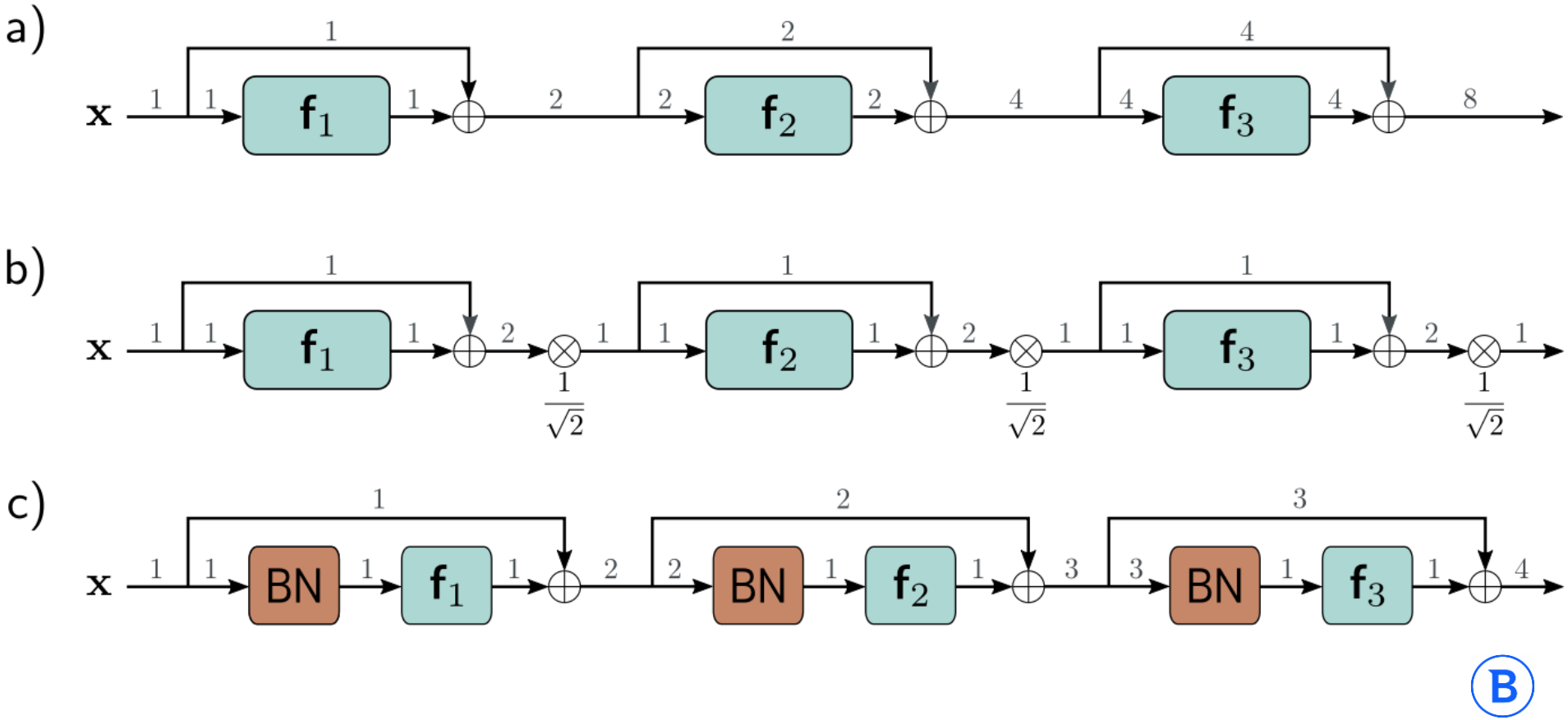


"U-Net: Convolutional Networks for Biomedical Image Segmentation", Ronneberge et al 2015

Residual Connections - At the Core of the Transformer Block



Batch Normalization



Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

“Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, Ioffe et al, 2015.

Batch Normalization

```
def __call__(self, x):
  # calculate the forward pass
  if self.training:
    xmean = x.mean(0, keepdim=True) # batch mean
    xvar = x.var(0, keepdim=True) # batch variance
  else:
    xmean = self.running_mean
    xvar = self.running_var
  xhat = (x - xmean) / torch.sqrt(xvar + self.eps) # normalize to unit variance
  self.out = self.gamma * xhat + self.beta
  # update the buffers
  if self.training:
    with torch.no_grad():
      self.running_mean = (1 - self.momentum) * self.running_mean + self.momentum * xmean
      self.running_var = (1 - self.momentum) * self.running_var + self.momentum * xvar
  return self.out
```

source: Let's build GPT: from scratch, in code, spelled out. Andrej Karpathy

Batch Normalization

For a given feature, normalizing its activations across the batch, i.e normalizing rows.

```
torch.manual_seed(1337)
module = BatchNorm1d(100)
x = torch.randn(32, 100) # batch size
x = module(x)
x.shape
```

```
torch.Size([32, 100])
```

```
x[:,0].mean(), x[:,0].std() # mean,st
```

```
(tensor(7.4506e-09), tensor(1.0000))
```

```
x[0,:].mean(), x[0,:].std() # mean,st
```

```
(tensor(0.0411), tensor(1.0431))
```

Layer Normalization

```
def __call__(self, x):
  # calculate the forward pass
  if self.training:
    xmean = x.mean(1, keepdim=True) # batch mean
    xvar = x.var(1, keepdim=True) # batch variance
  else:
    xmean = self.running_mean
    xvar = self.running_var
  xhat = (x - xmean) / torch.sqrt(xvar + self.eps) # normalize to unit variance
  self.out = self.gamma * xhat + self.beta
  # update the buffers
  if self.training:
    with torch.no_grad():
      self.running_mean = (1 - self.momentum) * self.running_mean + self.momentum * xmean
      self.running_var = (1 - self.momentum) * self.running_var + self.momentum * xvar
  return self.out
```

“Layer Normalization”, Ba et al, 2016

Layer Normalization

Normalize rows, not columns. For a given input in a batch, compute the mean and std of the features, and normalize those. Works for a batch of 1, same behavior during training and evaluation.

```
torch.manual_seed(1337)
module = BatchNorm1d(100)
x = torch.randn(32, 100) # batch size
x = module(x)
x.shape
```

```
torch.Size([32, 100])
```

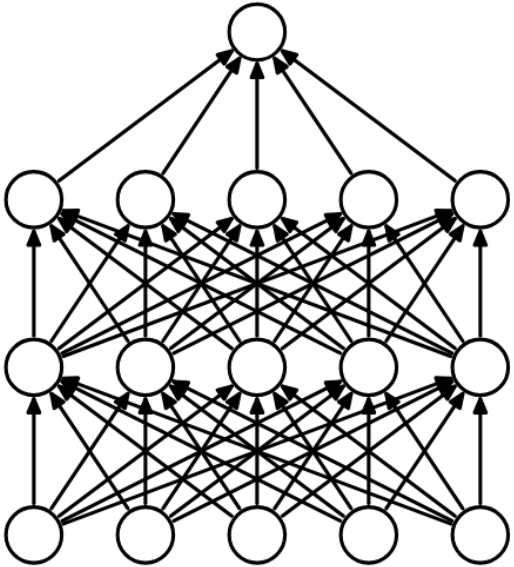
```
x[:,0].mean(), x[:,0].std() # mean,st
```

```
(tensor(0.1469), tensor(0.8803))
```

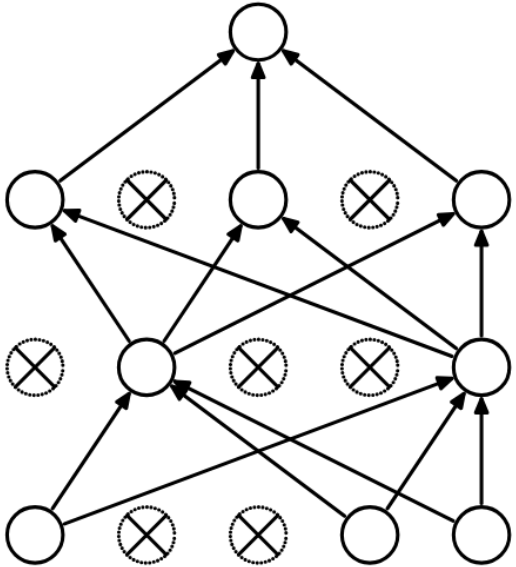
```
x[0,:].mean(), x[0,:].std() # mean,st
```

```
(tensor(-9.5367e-09), tensor(1.0000))
```

Dropout



(a) Standard Neural Net

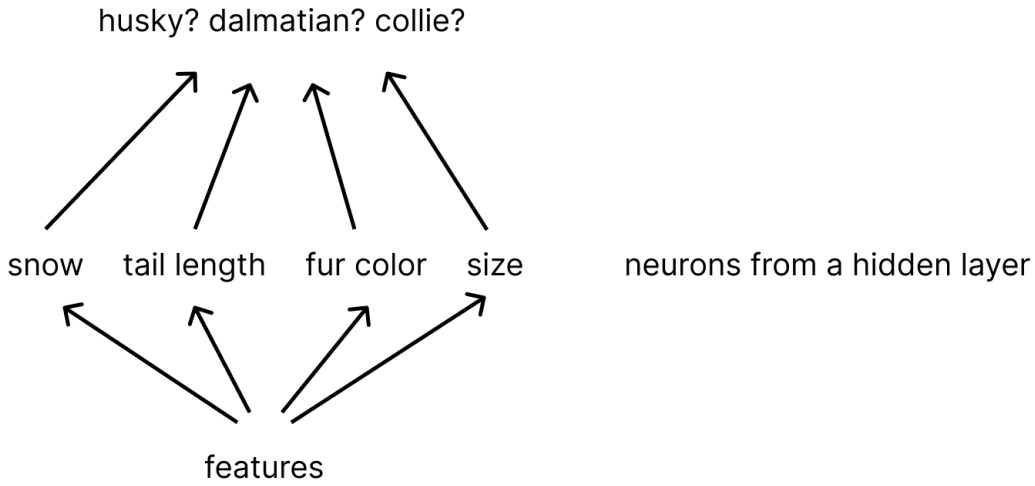


(b) After applying dropout.

- During training, for each input, an established percentage of activations, usually 50%, are dropped randomly.
- At test time, no neurons are dropped. The activations are scaled by a factor of ½ to account for the fact that the number of activations has doubled.
- This simulates an internal ensemble.

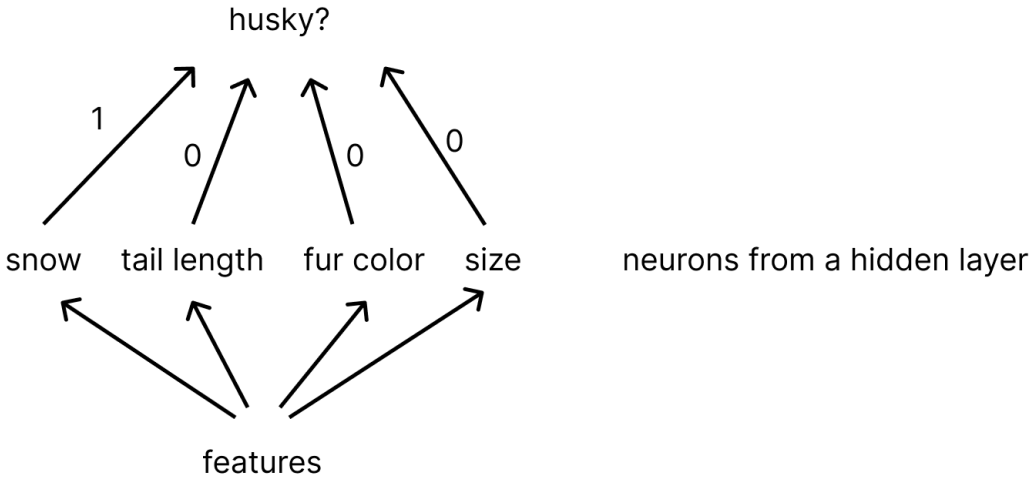
"Dropout: A Simple Way to Prevent Neural Networks from Overfitting", Srivastava et al, 2016.

Dropout



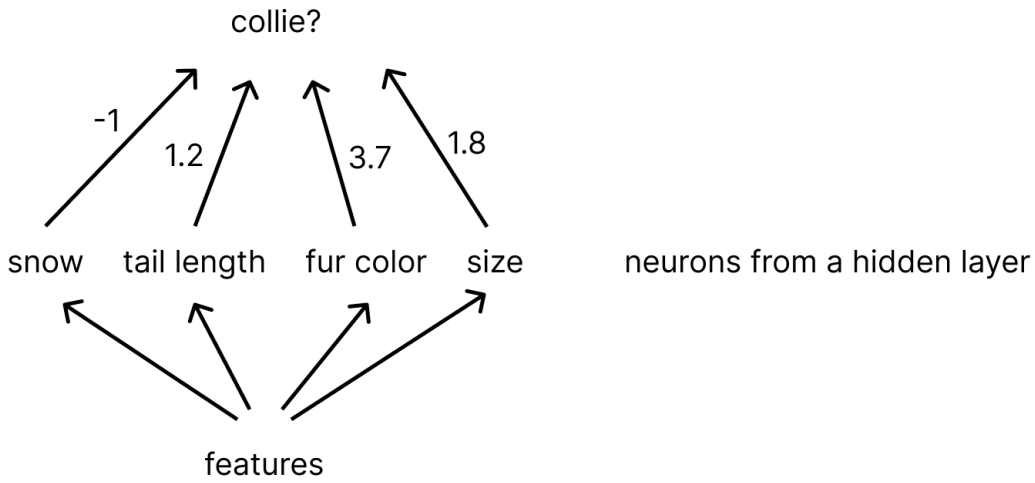
Consider a multi-class dog classification problem.

Dropout



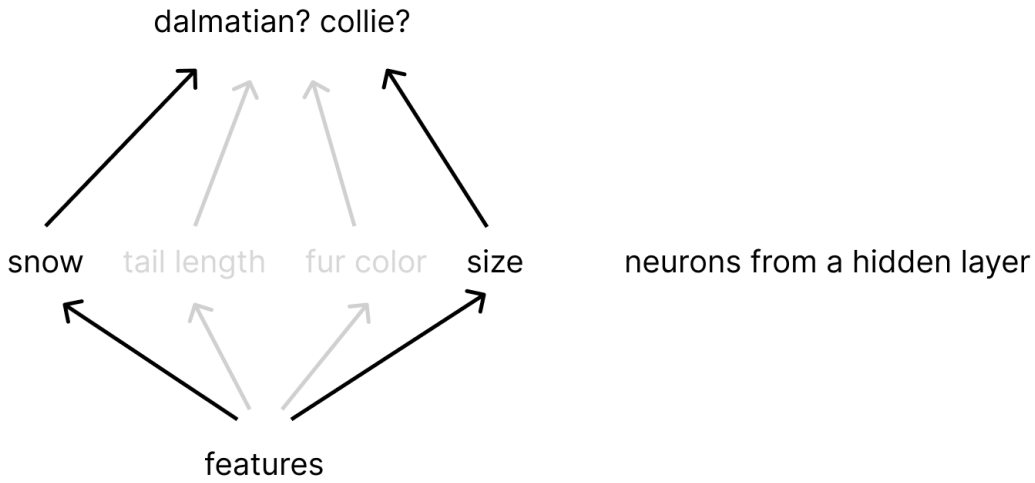
A network might learn to use the presence of snow as a clear indication that there is a husky in the picture and ignore all other features.

Dropout



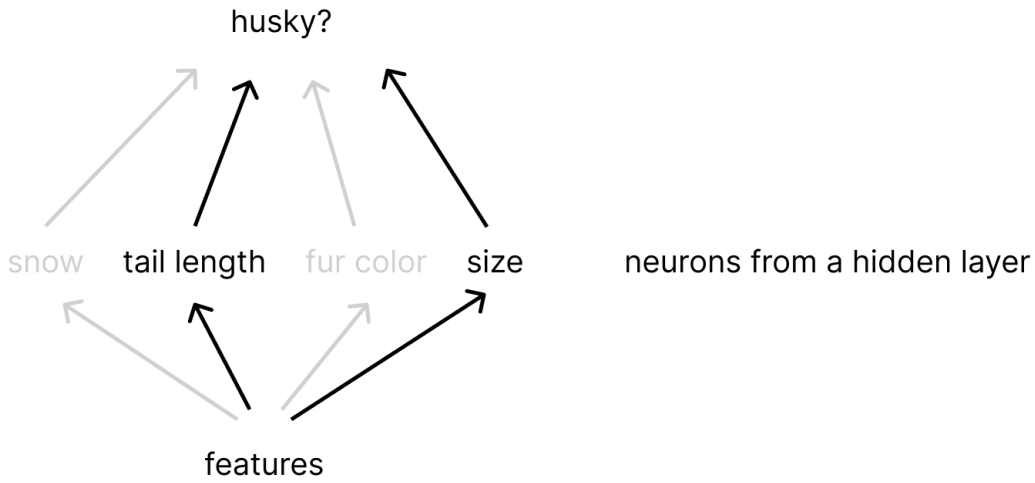
While using the other features for the other classes.

Dropout



- When ‘tail length’ and ‘fur color’ are dropped out, and the network is not able to distinguish between classes just based on the size of the dog and the presence of snow.
- Thus, it is forced to repurpose the ‘snow’ neuron, giving it a more generic role

Dropout

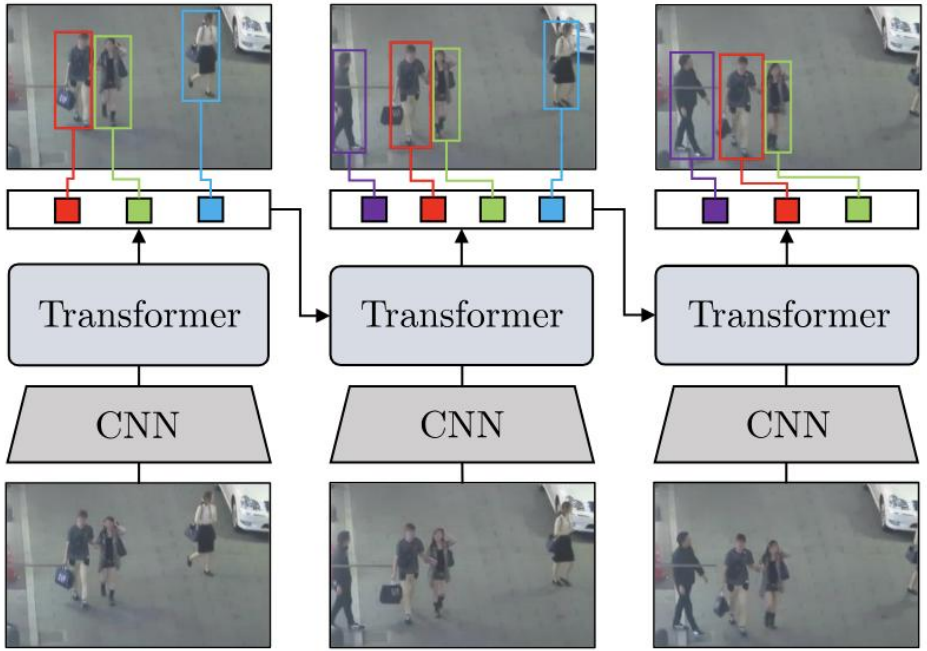


When the snow neuron is dropped out, the network also has to learn to use the other features to classify a husky.

Are convolutions relevant in the era of transformers?

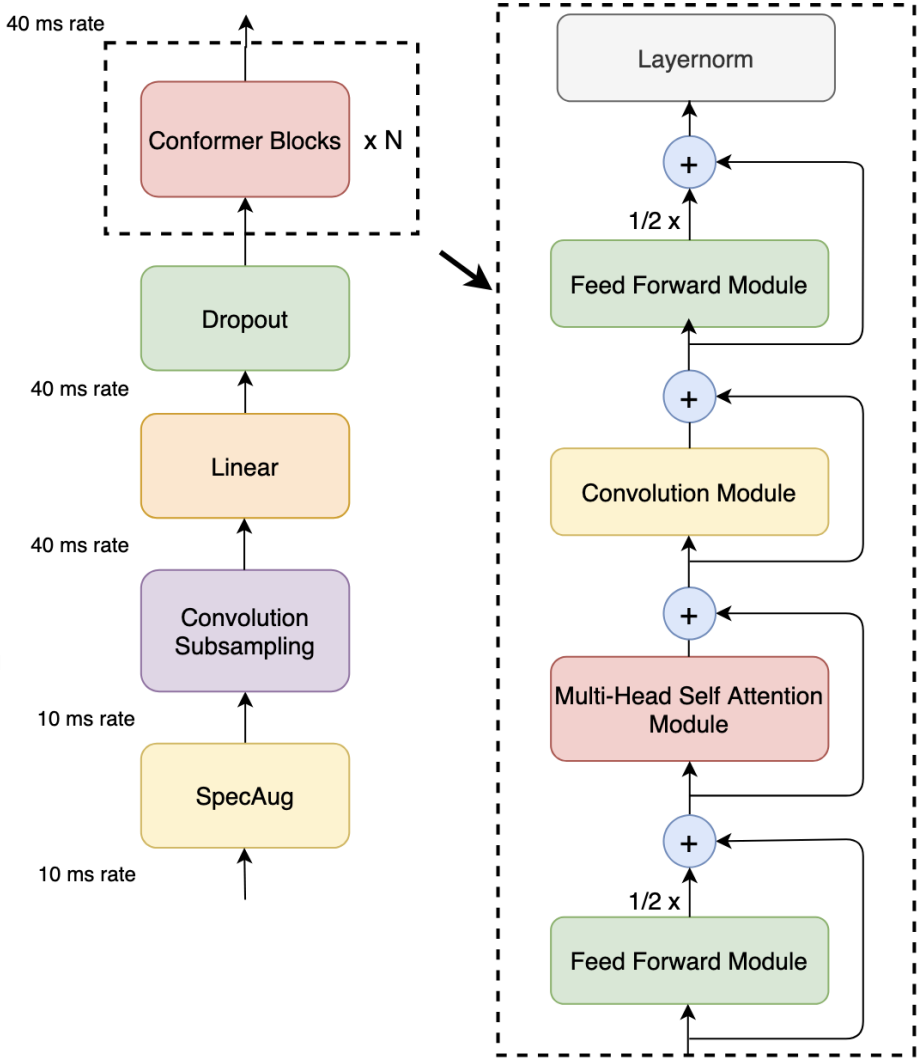
Convolutions Today

- Depiction of a Trackformer, from “TrackFormer: Multi-Object Tracking with Transformers”, Meinhardt et al, Facebook AI Research, 2022
- Effective for initial processing, e.g. detection of edges, basic shapes and objects
- The initial inductive bias is extremely useful, it requires less data to train and it guides the learning process



Convolutions Today

- Depiction of a Conformer, from “Conformer: Convolution-augmented Transformer for Speech Recognition”, Gulati et al, Google Inc, 2020.
- Effective even when employed all throughout the architecture when processing images or audio.
- Computationally very efficient. Nvidia released a fast-conformer speech recognition model in 2024, with state-of-the-art performance while achieving ~3x savings on compute and ~4x savings on memory, compared to SOTA transformers.
(https://developer.nvidia.com/blog/new-standard-for-speech-recognition-and-translation-from-the-nvidia-nemo-canary-model/)



Meta Movie Gen

Released on October 2024. State-of-the-art movie generation and manipulation.

Blogpost: https://ai.meta.com/research/movie-gen/

Paper: https://arxiv.org/pdf/2410.13720

Meta Movie Gen

TAE Encoder

Patchify

Gaussian noise

+pos embed

Timestep t

Scale & Shift

Nx Transformer Blocks

Cross-attention

UL2

Long-prompt MetaCLIP

ByT5

An emu holding a sign says “No, Movie Gen is the best”

TAE Decoder

Meta Movie Gen

TAE architecture. We adopt the architecture used for image autoencoders from (Rombach et al., 2022) and 'inflate' it by adding temporal parameters: a 1D temporal convolution after each 2D spatial convolution and a 1D temporal attention after each spatial attention. All temporal convolutions use symmetrical replicate padding. Temporal downsampling is performed via strided convolution with stride of 2, and upsampling by nearest-neighbour interpolation followed by convolution. Downsampling via strided convolution means that videos of any length are able to be encoded (notably including images, which are treated as single-frame videos) by discarding spurious output frames as shown in Figure 4. Similar to (Dai et al., 2023), we find that increasing the number of channels in the latent space $\mathbf{X}$ improves both the reconstruction and the generation performance. We use $C = 16$ in this work. We initialize the spatial parameters in the TAE using a pre-trained image autoencoder, and then add the temporal parameters to inflate the model as described above. After inflation, we jointly train the TAE on both images and videos, in a ratio of 1 batch of images to 3 batches of

Thank you for your time and attention.

Please send unstructured feedback if you have time, it means a lot to me.

antoniobarbalau@gmail.com