Lecture 9B Convolutional Neural Networks

Lecture 10

MLPs → CNNs → Graph NN → RNN → Statespace → Transformers

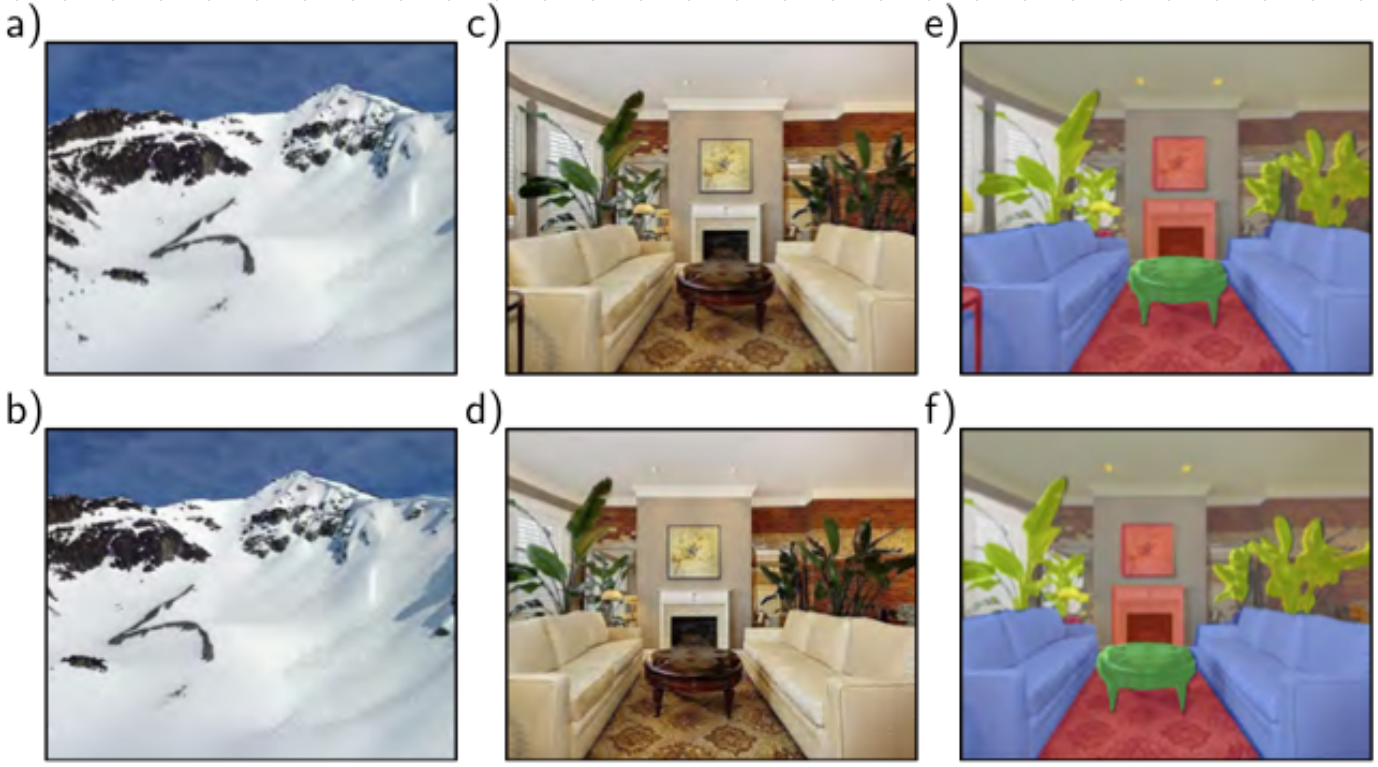
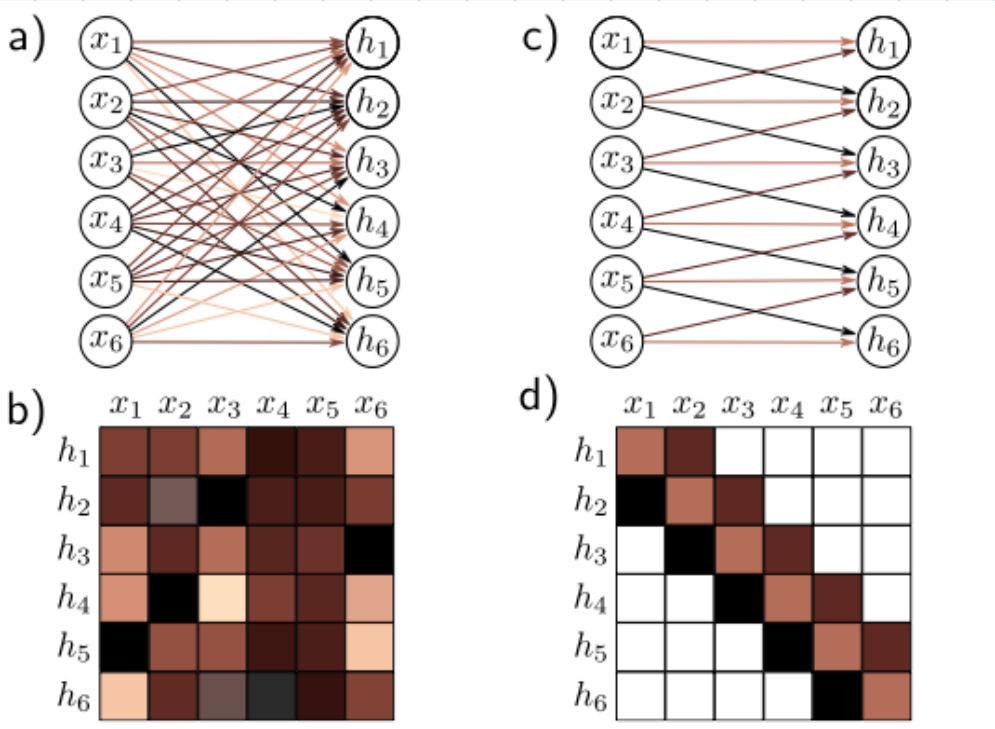


Figure 10.1 Invariance and equivariance for translation. a–b) In image classification, the goal is to categorize both images as "mountain" regardless of the horizontal shift that has occurred. In other words, we require the network prediction to be invariant to translation. c,e) The goal of semantic segmentation is to associate a label with each pixel. d,f) When the input image is translated, we want the output (colored overlay) to translate in the same way. In other words, we require the output to be equivariant with respect to translation. Panels c–f) adapted from Rousselham et al. (2021).

CNNs

1. Respect locality → convolutional structure with small filters
2. Respect symmetries / invariances
   - → weight sharing
   - → data augmentation
3. Hierarchichal structure (Parts make whole) → Depth.
   - → Multiresolution



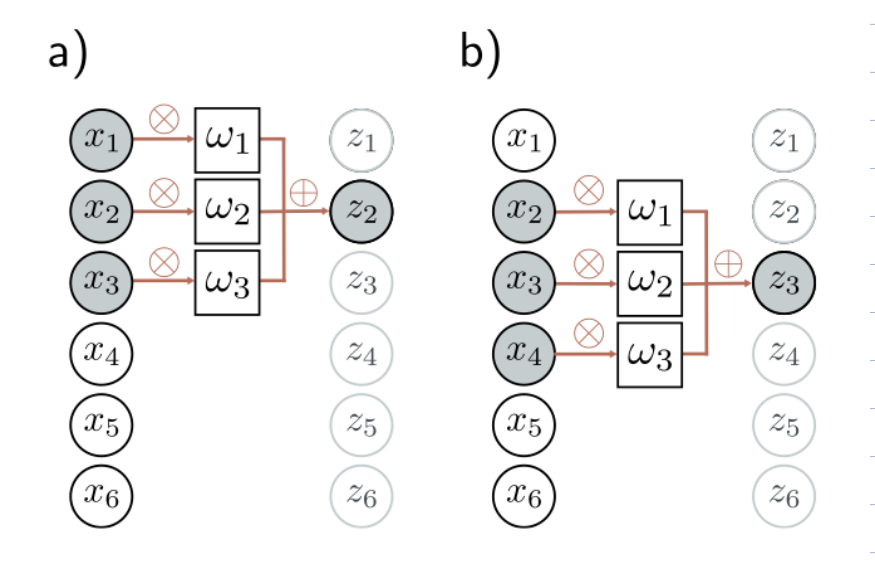
1D Convolution LTI Systems

$$y(t) = \int_{-\infty}^{\infty} x(\tau)\, h(t-\tau)\, d\tau$$

$$y[t] = \sum_{\tau=-\infty}^{\infty} x[\tau]\, h[t-\tau]$$

For FIR: $\sum_{\tau=-k}^{k} h(\tau) \cdot x[t-\tau]$.

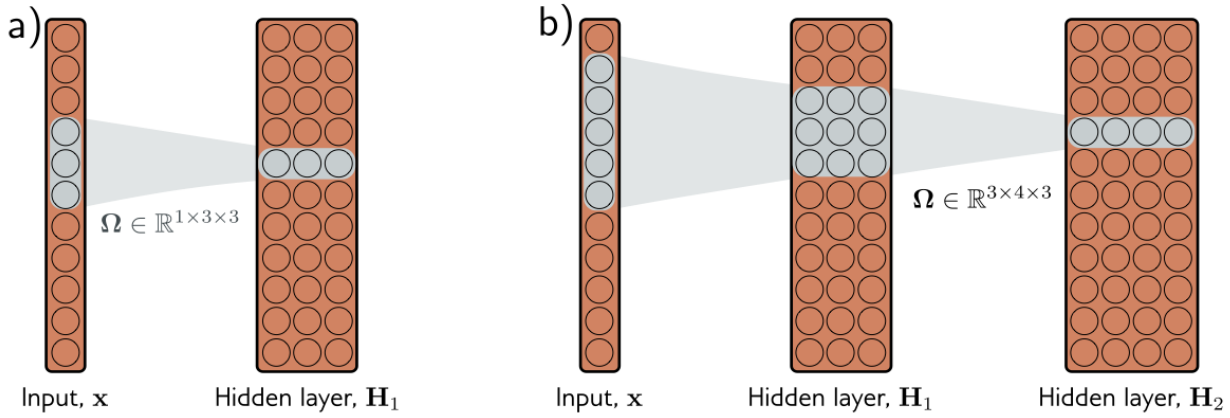
CNNs: No flip.



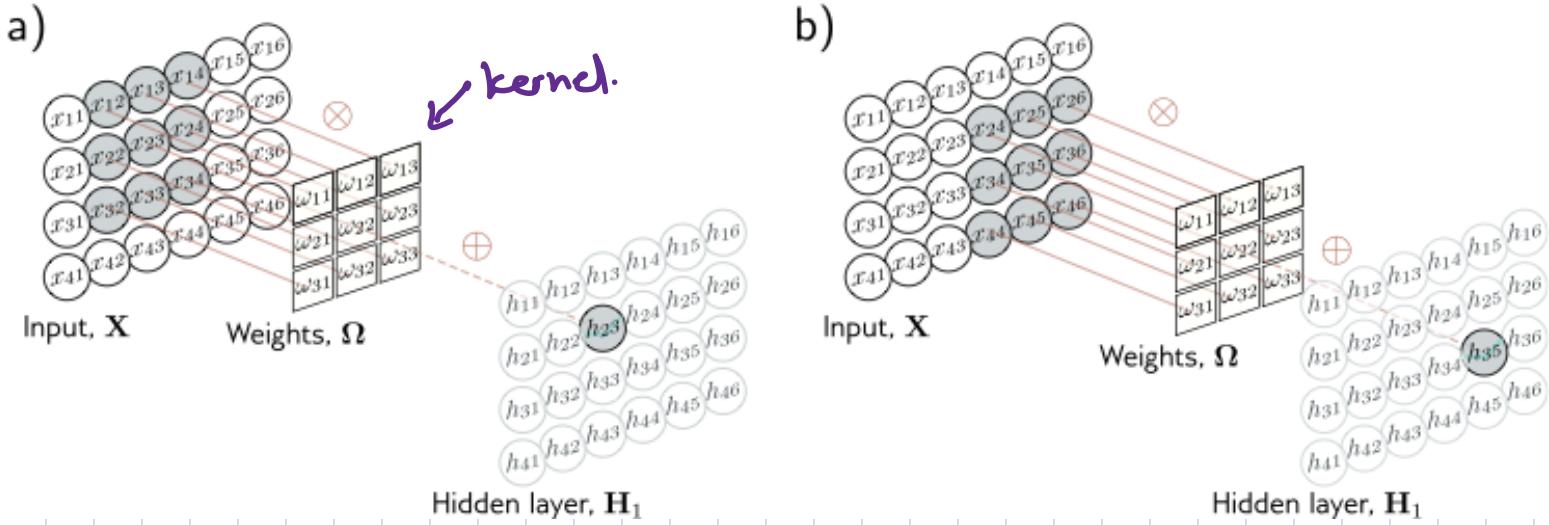
e.g.

$$h_i = a\left[\beta + \sum_{j=1}^{3} w_j x_{i+j-2}\right] = a\left[\beta + w_1 x_{i-1} + w_2 x_i + w_3 x_{i+1}\right]$$

Receptive fields



2D Convolution



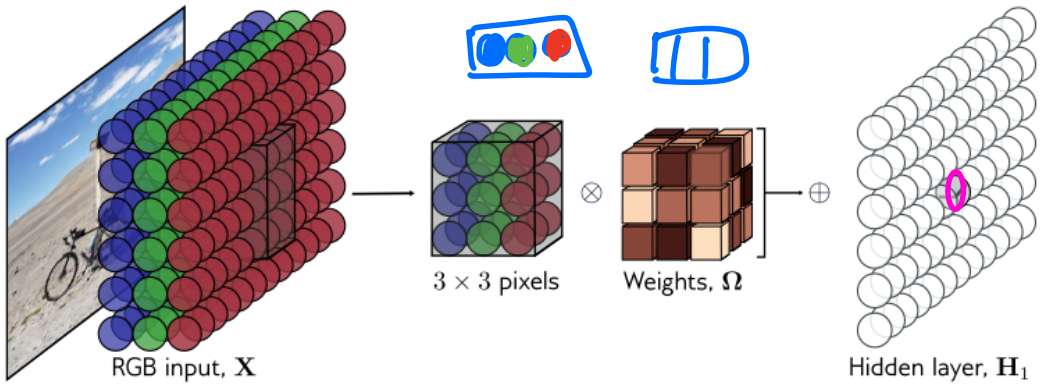


Figure 10.10 2D convolution applied to an image. The image is treated as a 2D input with three channels corresponding to the red, green, and blue components. With a 3×3 kernel, each pre-activation in the first hidden layer is computed by pointwise multiplying the 3×3×3 kernel weights with the 3×3 RGB image patch centered at the same position, summing, and adding the bias. To calculate all the pre-activations in the hidden layer, we "slide" the kernel over the image in both horizontal and vertical directions. The output is a 2D layer of hidden units. To create multiple output channels, we would repeat this process with multiple kernels, resulting in a 3D tensor of hidden units at hidden layer $\mathbf{H}_1$.

So what does a 1x1 conv do?

Common input: 2D image with 3 channels RGB.

To generate 1 hidden layer with a 3x3 kernel

we have $3 \times 3 \times 3$ weight

ch ↑ ht ↑ width ↑

To create C_o output channels, repeat this C_o times

With C_{in} input channels, $K \times K$ kernel need: $C_{in} \times C_o \times K \times K$ weights.

and C_o biases.

Multiple output channels prevent information loss.

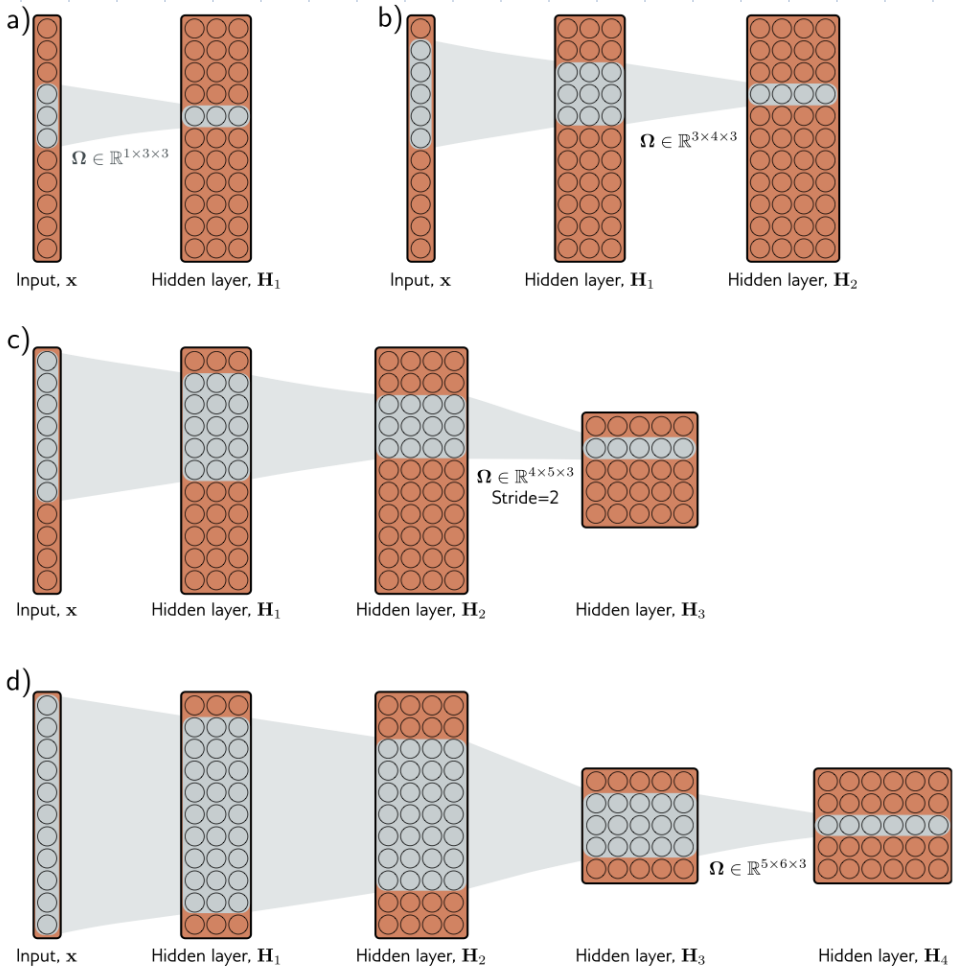
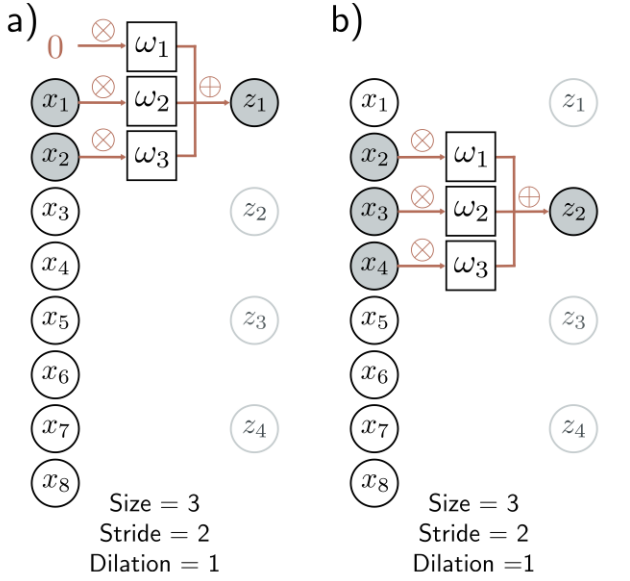


Figure 10.6 Receptive fields for network with kernel width of three. a) An input with eleven dimensions feeds into a hidden layer with three channels and convolution kernel of size three. The pre-activations of the three highlighted hidden units in the first hidden layer $\mathbf{H}_1$ are different weighted sums of the nearest three inputs, so the receptive field in $\mathbf{H}_1$ has size three. b) The pre-activations of the four highlighted hidden units in layer $\mathbf{H}_2$ each take a weighted sum of the three channels in layer $\mathbf{H}_1$ at each of the three nearest positions. Each hidden unit in layer $\mathbf{H}_1$ weights the nearest three input positions. Hence, hidden units in $\mathbf{H}_2$ have a receptive field size of five. c) The hidden units in the third layer (kernel size three, stride two) increases the receptive field size to seven. d) By the time we add a fourth layer, the receptive field of the hidden units at position three have a receptive field that covers the entire input.

Useful but inaccurate:
Channel as feature detector at the scale of its receptive field

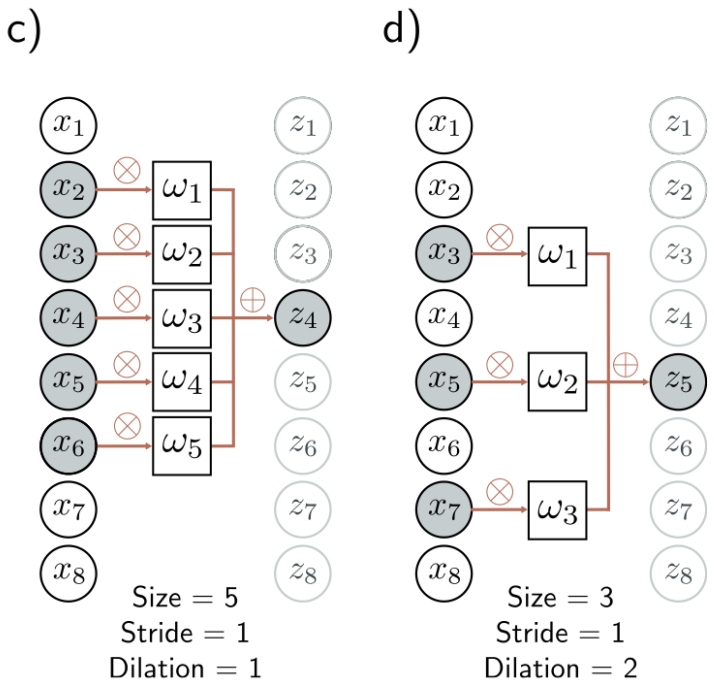
Downsampling via Stride → Why?

Too much information
Downsampling helps.
Changes/collects information

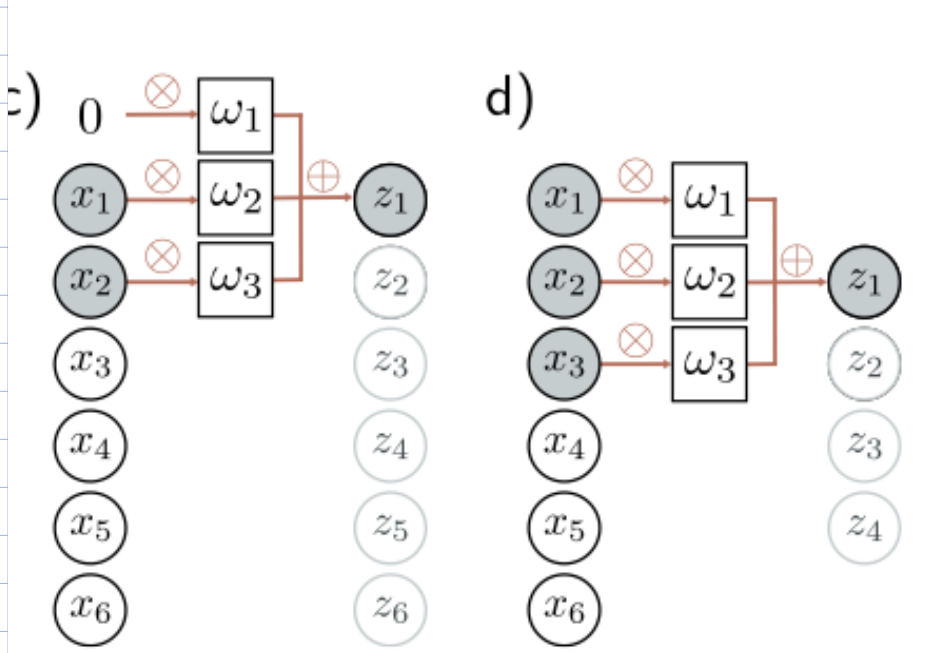


Dilation.

Get agg. over wider areas without increasing parameters.



Padding



Other options

- Mirror padding
- Circular padding.
- Average padding.

0-padding
"same-padding"

No padding.
only "valid convs"
output reduced in size.

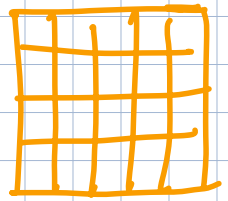
Examples

① 1×1 conv, 1 channel input, 1 channel output.
For every pixel, scale by a constant, and add a bias.

learnable parameters

Weights: 1
Biases: 1

② 1×1 conv, 1 channel input, Cout channel outputs.
For every pixel, scale by a constant, and add a bias but do it Cout times.



Weights: Cout
biases: Cout

③ 3×3 conv with 1 channel input, 1 channel output
For every pixel, scale the value at this pixel and immediate neighbours by individual weights, add them up, add bias.

Weights: 9

Biases: 1

④ kxk conv, 1channel input, 1channel output

Weights: k^2

Biases: 1

⑤ kxk conv, Cin channel input, 1 channel output

weights: $k^2 \cdot C_{in}$

Biases : 1

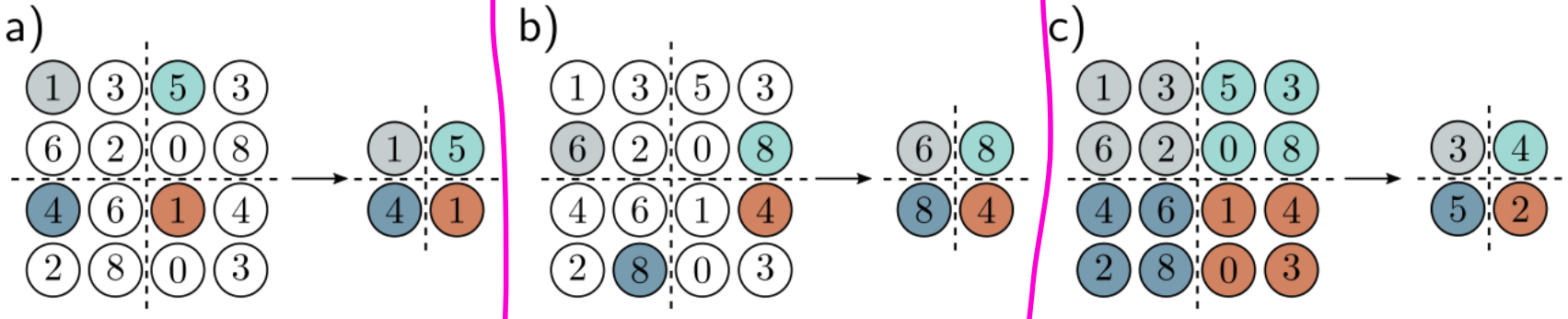
⑥ kxk conv, Cin channel inputs, Cout channel outputs.

Weights: $k^2 C_{in} \cdot C_{out}$

Biases: Cout

Q: How could we change # of channels, keeping 2D size the same?

Pooling



Stride/subsample | Max-pooling | Mean-pooling

Backprop w/ weight sharing

$$L = \sum_i (w_i x_i - y_i)^2$$

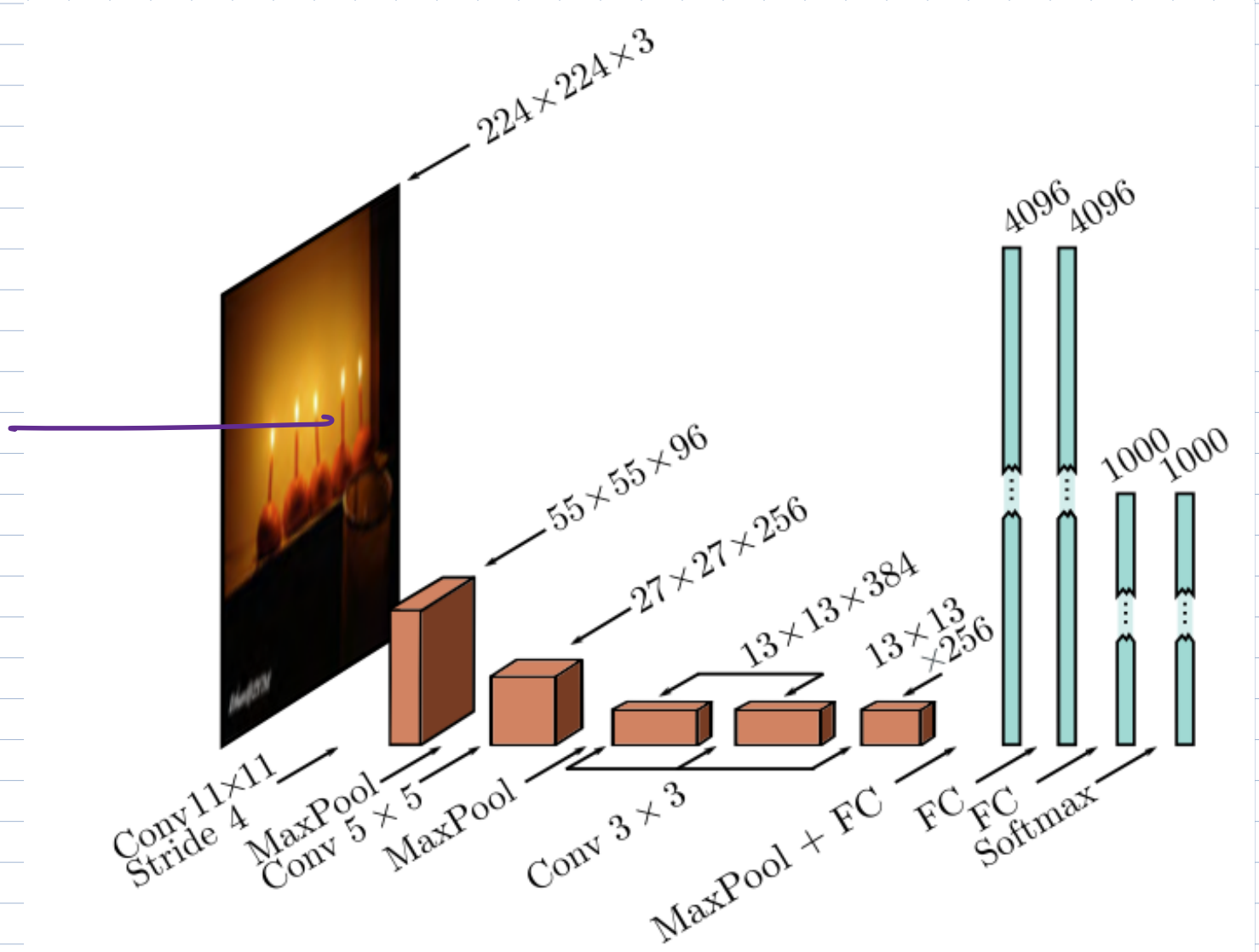
$$\frac{\partial L}{\partial a} = \frac{\partial}{\partial a}\left(\sum_i (a \cdot x_i - y_i)^2\right)$$

$$= \sum_i 2(a x_i - y_i) \cdot x_i$$

$$\vec{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

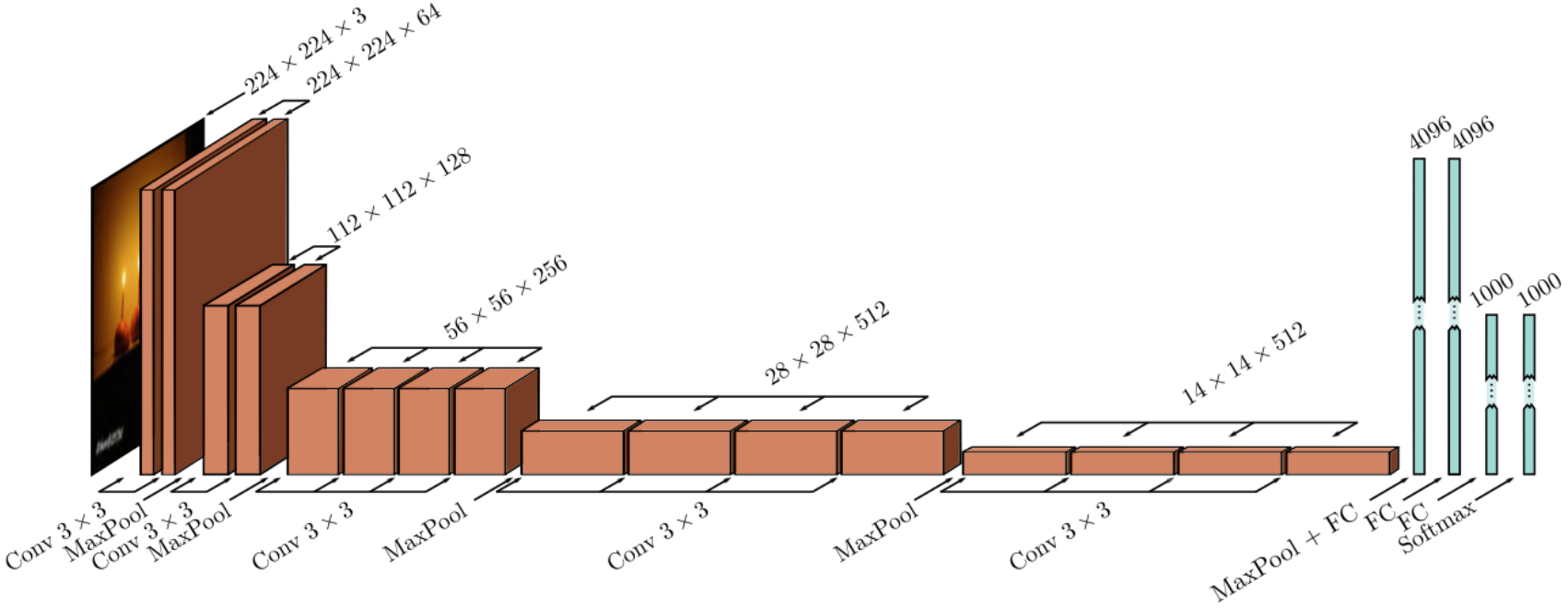
$$\vec{w} = \begin{bmatrix} a \\ a \\ a \\ \vdots \\ a \end{bmatrix}$$

Alex Net ~ 60 million parameters



- Data Augmentation
- Weight decay
- Dropout.

VGG net


224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
4096
4096
1000
1000
Conv 3 × 3
MaxPool
Conv 3 × 3
MaxPool
Conv 3 × 3
MaxPool
Conv 3 × 3
MaxPool
Conv 3 × 3
MaxPool + FC
FC
FC
Softmax

Data Augmentation.

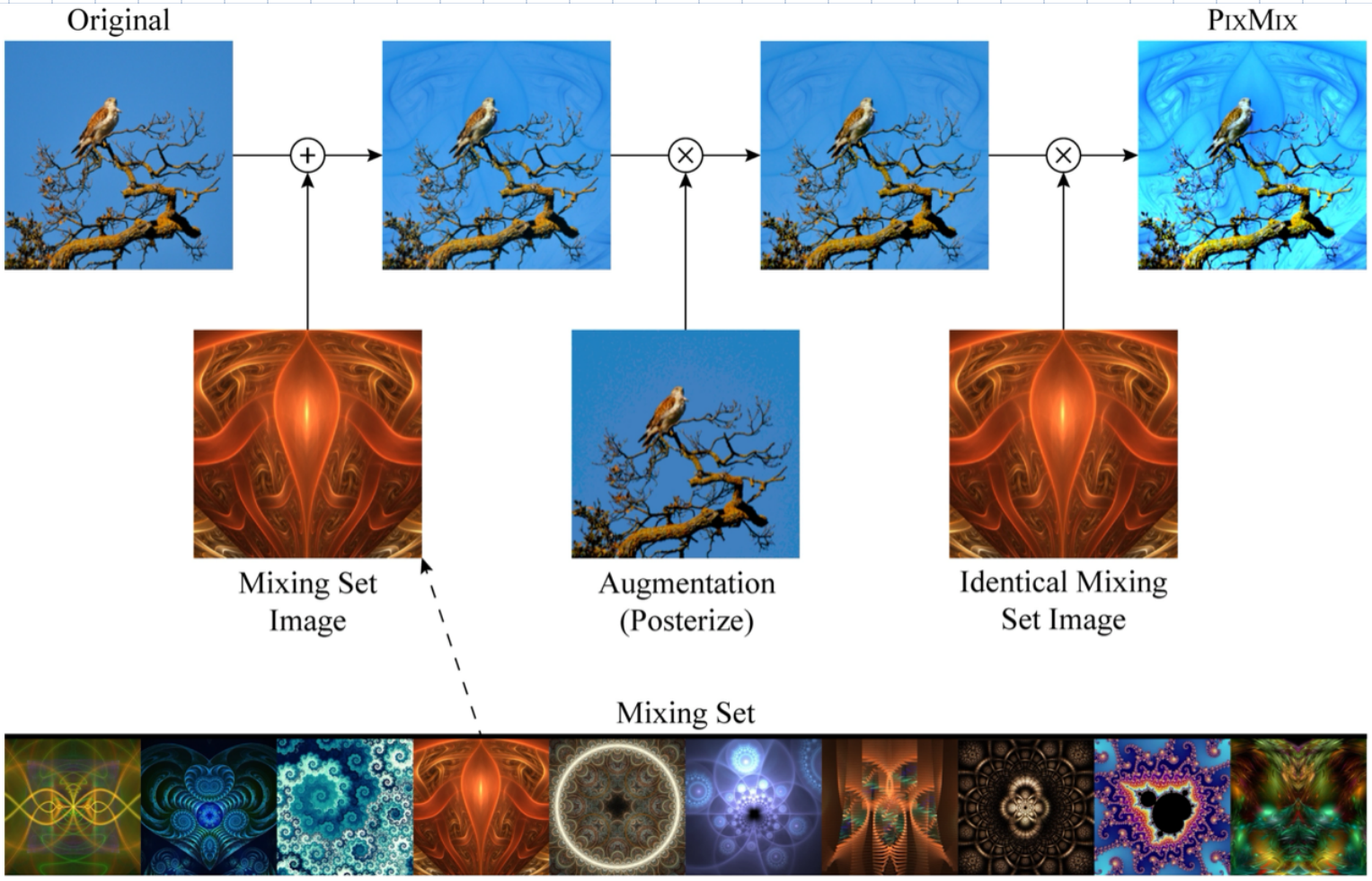
```
augmentations = [
    autocontrast, equalize, posterize, rotate, solarize, shear_x, shear_y,
    translate_x, translate_y
]

augmentations_all = [
    autocontrast, equalize, posterize, rotate, solarize, shear_x, shear_y,
    translate_x, translate_y, color, contrast, brightness, sharpness
]
```

PixMix Repo
Hendrycks. 2022.

Robustness to corruption, distribution shift




Original
PixMix
Mixing Set Image
Augmentation (Posterize)
Identical Mixing Set Image
Mixing Set

Dropout

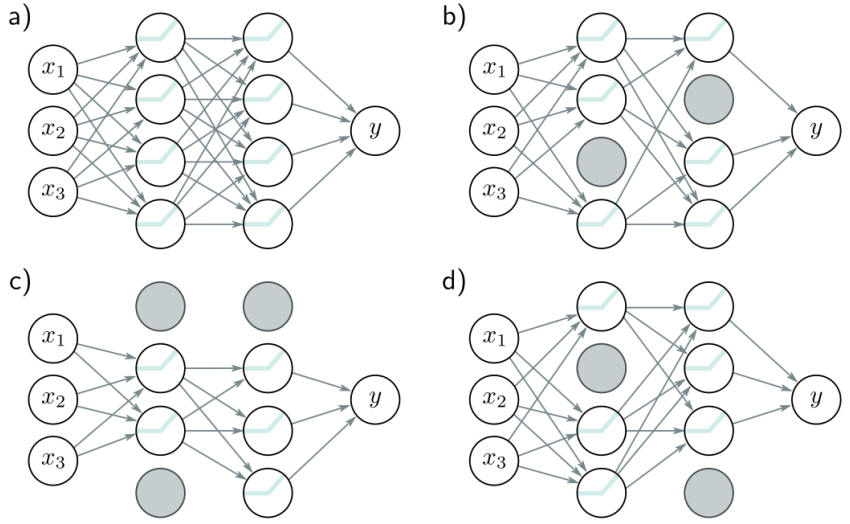


Figure 9.8 Dropout. a) Original network. b–d) At each training iteration, a random subset of hidden units is clamped to zero (gray nodes). The result is that the incoming and outgoing weights from these units have no effect, so we are training with a slightly different network each time.

Data augmentation applied inside our network during training

Idea: Zero out random activations inside network during training

Impact: Encourages redundancy.

$\vec{h}_l$: hidden layer

$$\tilde{h}_l[i] = \begin{cases} 0 & \text{with prob. } 1-p \\ h_l(i) & \text{with prob } p \end{cases}$$

} Do a coin toss ie multiply by Bernoulli(p)

Test time? Scale activations by p

Pytorch: Scale by $\frac{1}{p}$ at train time.

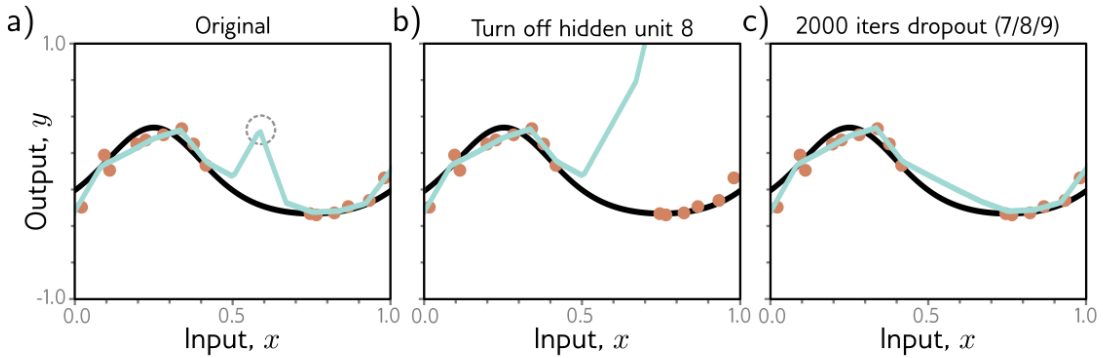


Figure 9.9 Dropout mechanism. a) An undesirable kink in the curve is caused by a sequential increase in the slope, decrease in the slope (at circled joint), and then another increase to return the curve to its original trajectory. Here we are using full-batch gradient descent, and the model (from figure 8.4) fits the data as well as possible, so further training won't remove the kink. b) Consider what happens if we remove the eighth hidden unit that produced the circled joint in panel (a), as might happen using dropout. Without the decrease in the slope, the right-hand side of the function takes an upwards trajectory, and a subsequent gradient descent step will aim to compensate for this change. c) Curve after 2000 iterations of (i) randomly removing one of the three hidden units that cause the kink and (ii) performing a gradient descent step. The kink does not affect the loss but is nonetheless removed by this approximation of the dropout mechanism.