

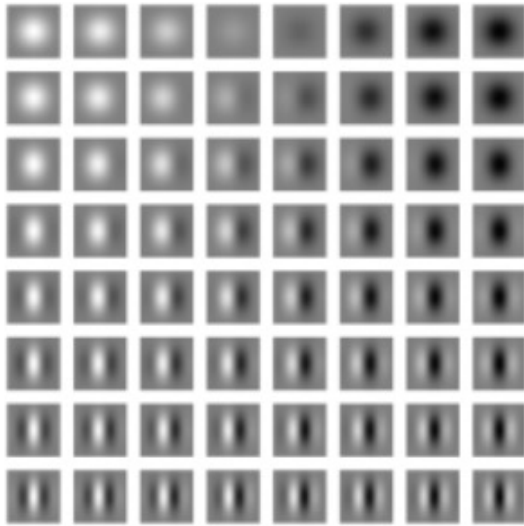
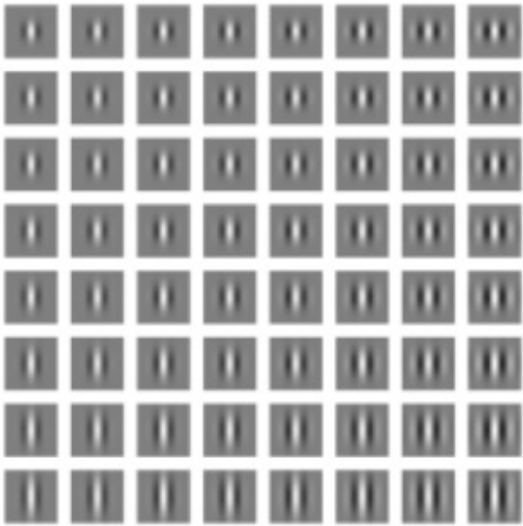
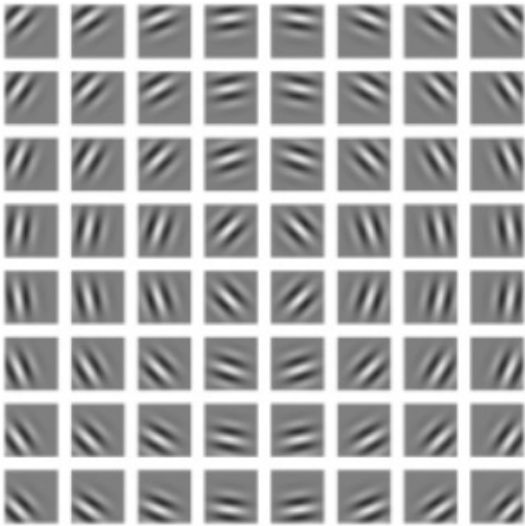
Convolutional Neural Networks (CNN)

David I. Inouye

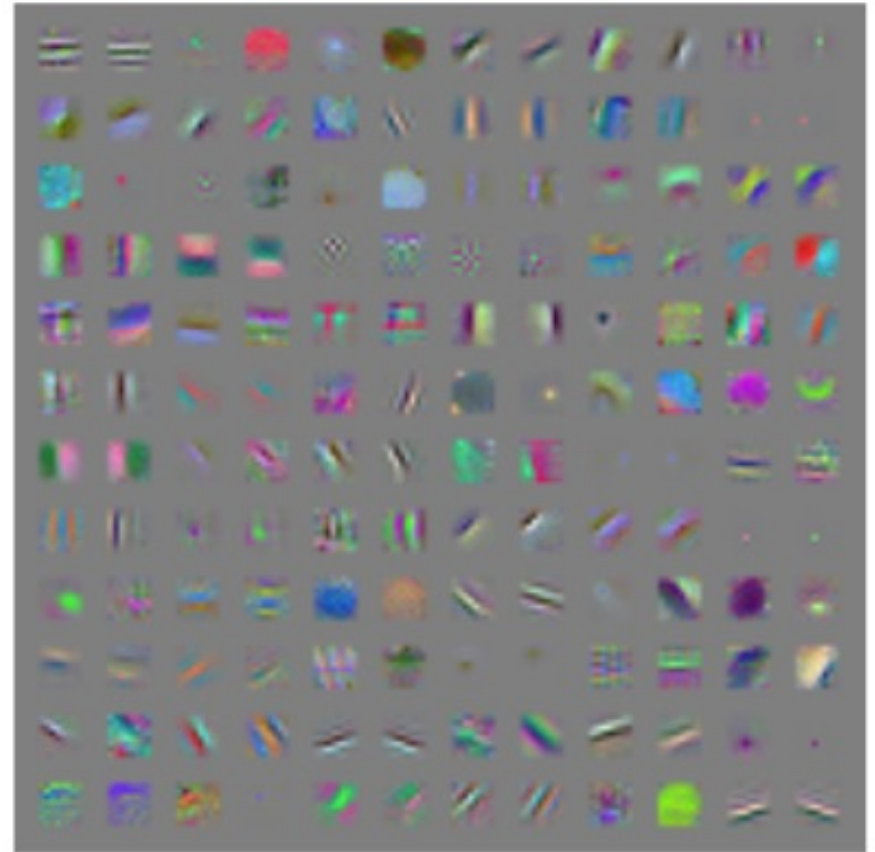
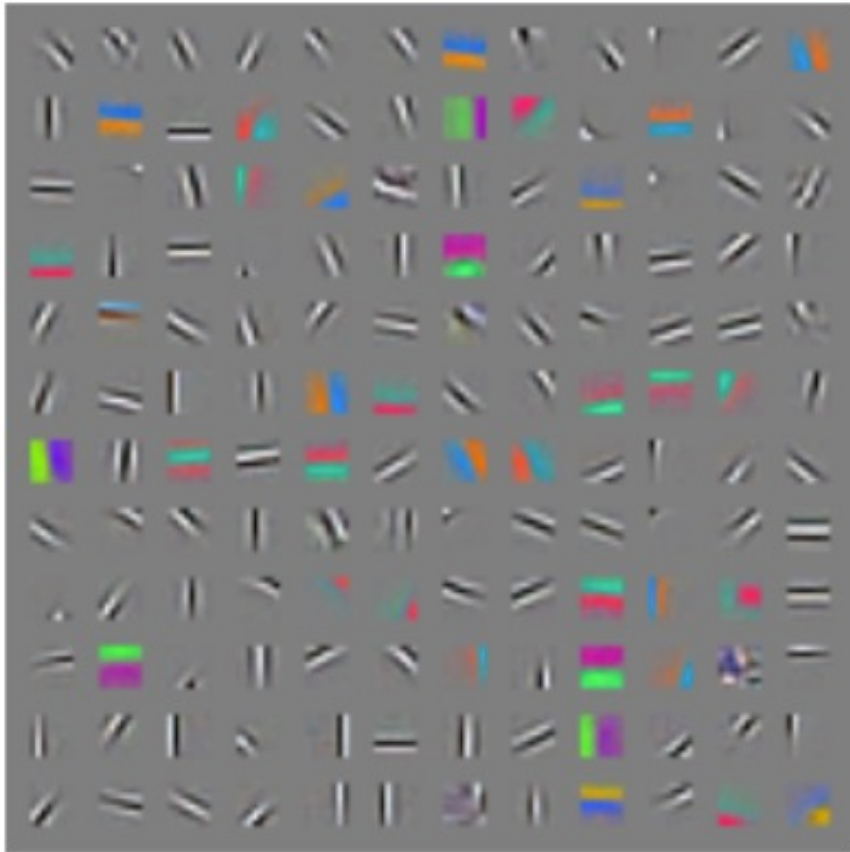
Why convolutional networks?

- ▶ Neuroscientific inspiration
- ▶ Computational reasons
 - ▶ Sparse computation (compared to full deep networks)
 - ▶ Shared parameters (only a small number of shared parameters)
 - ▶ Translation invariance

Motivation for convolution networks:
Gabor functions derived from neuroscience experiments are simple convolutional filters [DL, ch. 9]



Convolutional networks automatically learn filters similar to Gabor functions [DL, ch. 9]



1D convolutions are similar but slightly different than signal processing / math convolutions

x

1	2	3	2	5	1
---	---	---	---	---	---

f

1	2
---	---

y

5	8	7	12	7
---	---	---	----	---

Padding or stride parameters alter the computation and output shape



1D convolutions are similar but slightly different than signal processing / math convolutions

x

1	2	3	2	5	1
---	---	---	---	---	---

f

1	2
---	---

 Zero padding of 1

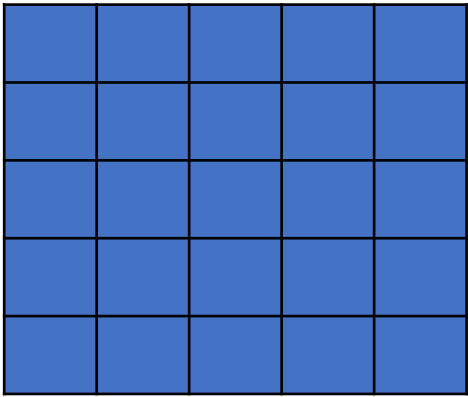
y

2	5	8	7	12	7	1
---	---	---	---	----	---	---

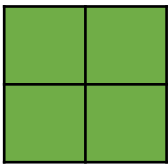
Switch to demo of 1D

2D convolutions are simple generalizations to matrices

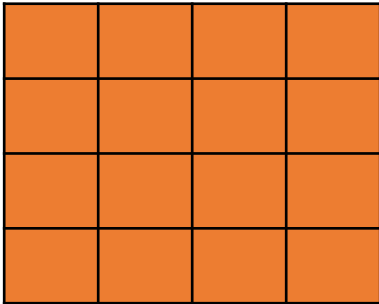
x



f

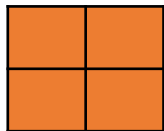


y



Stride of 2

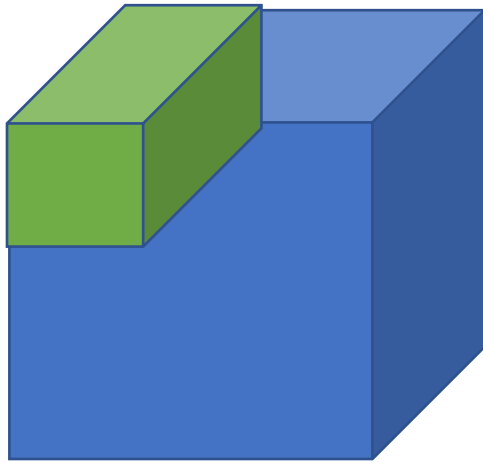
y



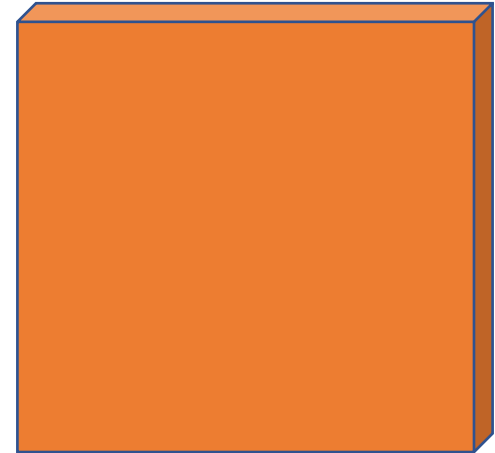
Switch to demo of 2D

3D convolutions are similar but usually channel dimension is assumed

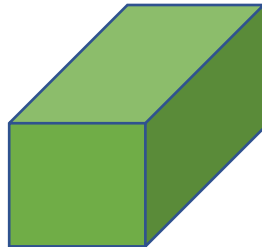
$$x \in \mathcal{R}^{c \times h \times w}$$



$$y \in \mathcal{R}^{1 \times h' \times w'}$$



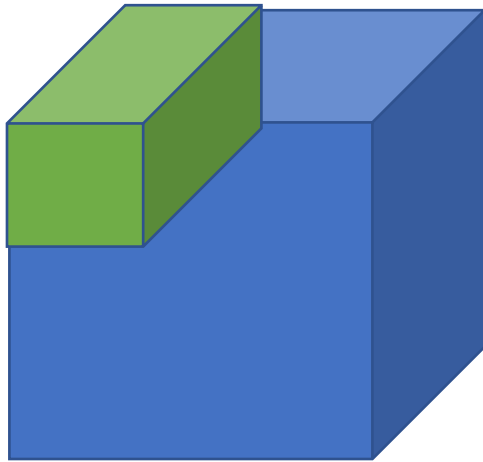
$$f \in \mathcal{R}^{c \times f_h \times f_w}$$



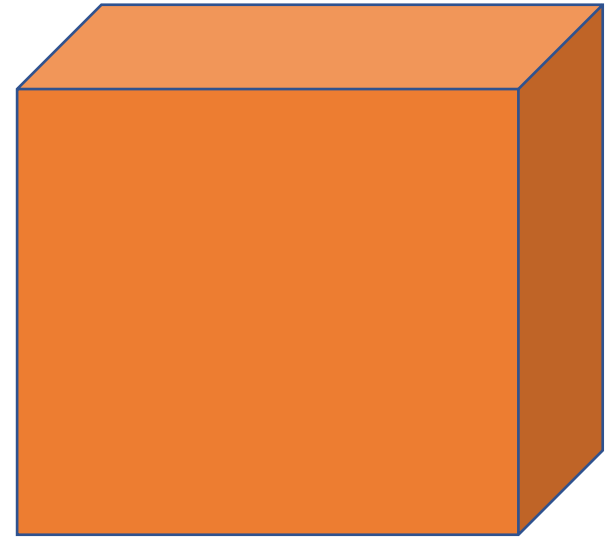
“ $f_h \times f_w$ convolution” (channel dimension is assumed)

Multiple convolutions increase the output channel dimension

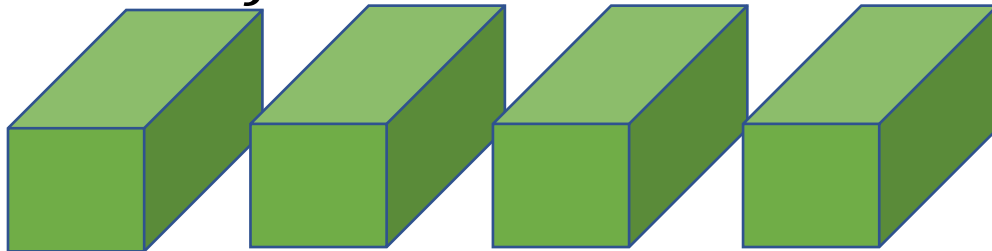
$$x \in \mathcal{R}^{c \times h \times w}$$



$$y \in \mathcal{R}^{4 \times h' \times w'}$$



$$f_j \in \mathcal{R}^{c \times f_h \times f_w}$$



Reasoning about input and output shapes is important for debugging and designing CNNs

- ▶ Convolution input parameters

- ▶ $ChannelIn = C_{in}$

- ▶ $ChannelOut = C_{out}$ (equivalent to # filters)

- ▶ $KernelSize = [K_0, K_1]$

- ▶ $Stride = [S_0, S_1]$

- ▶ $Padding = [P_0, P_1]$

- ▶ $C_{out} = \# \text{ filters}$

- ▶ Output spatial dimensions

- ▶ $H_{out} = \left\lfloor \frac{(H_{in} + 2 P_0 - K_0)}{S_0} + 1 \right\rfloor$

- ▶ $W_{out} = \left\lfloor \frac{(W_{in} + 2 P_1 - K_1)}{S_1} + 1 \right\rfloor$

- ▶ Output batch dimension should match input

Common convolution configurations

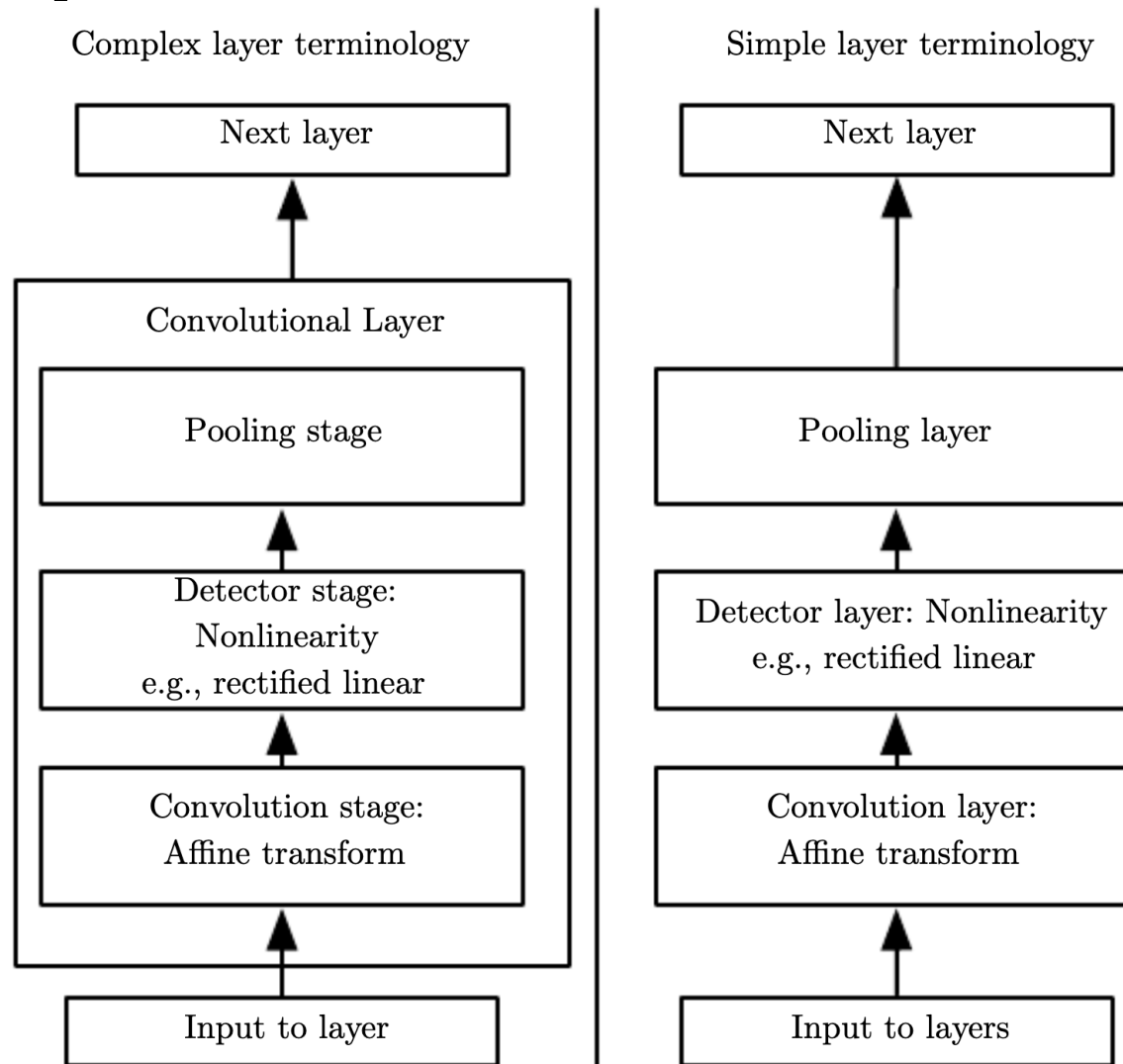
$$H_{out} = \left\lfloor \frac{(H_{in} + 2 P_0 - K_0)}{S_0} + 1 \right\rfloor$$

- ▶ Output has same height and width as input
 - ▶ 1 x 1 convolution with padding=0, stride=1
 - ▶ 3 x 3 convolution with padding=1, stride=1
 - ▶ 5 x 5 convolution with padding=2, stride=1
- ▶ Output has half the height and width of input
 - ▶ 2 x 2 convolution with padding=0, stride=2
 - ▶ 4 x 4 convolution with padding=1, stride=2

Switch to demo of 3D, activation functions, and pooling

Standard Convolutional Layer Terminology

[DL, ch. 9]



Demo of CIFAR-10 CNN in Pytorch

Two important modern CNN
architecture concepts:
batch normalization and
residual networks

Batch normalization dynamically normalizes each feature to have zero mean and unit variance

- ▶ Basic idea: Normalize input batch of each layer during the forward pass

1. Input is **minibatch** of data $X^t \in \mathbb{R}^{m \times d}$ at iteration t
2. Compute mean and standard deviation for every feature

$$\mu_j^t = \mathbb{E}[x_j^t], \sigma_j^t = \sqrt{\mathbb{E}[(x_j^t - \mu_j^t)^2]}, \quad \forall j \in \{1, \dots, d\}$$

3. Normalize each feature (note different for every batch)

$$\tilde{x}_{i,j}^t = \frac{(x_{i,j}^t - \mu_j^t)}{\sigma_j^t}$$

4. Output \tilde{X}^t

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. In *Advances in Neural Information Processing Systems* (pp. 2483-2493).

Because BatchNorm removes linear effects, extra linear parameters are also learned

- ▶ The form of this final update is:

$$\tilde{x}_{i,j}^t = \frac{(x_{i,j}^t - \mu_j^t)}{\sigma_j^t} \cdot \gamma_j + \beta_j$$

- ▶ Where γ_j and β_j are learnable parameters
- ▶ While μ_j^t and σ_j^t are computed from the **minibatch**
- ▶ But how do we compute μ_j^t and σ_j^t about during test time (i.e., no minibatch)?
- ▶ Use running average of mean and variance

$$\begin{aligned}\mu_{run}^t &= \lambda \mu_{run}^{t-1} + (1 - \lambda) \mu_{batch}^t \\ \sigma_{run}^{2t} &= \lambda \sigma_{run}^{2t-1} + (1 - \lambda) \sigma_{batch}^{2t}\end{aligned}$$

For CNNs, the channel dimension is treated as a “feature”

- ▶ If the input minibatch tensor is $X^t \in \mathbb{R}^{m \times c \times h \times w}$, then the channel dimension c is treated as a feature:

$$\mu_j^t = \mathbb{E}[x_j^t], \sigma_j^t = \sqrt{\mathbb{E}[(x_j^t - \mu_j^t)^2]},$$
$$\forall j \in \{1, \dots, c\}$$

- ▶ Where the mean is taken over **both** the batch dimension m **and** the spatial dimensions h and w
 - ▶ Called “Spatial Batch Normalization”
- ▶ Variants: Instance, Group or Layer Normalization

<https://pytorch.org/docs/stable/nn.html#normalization-layers>

BatchNorm can stabilize and accelerate training of deep models

- ▶ To use in practice:
 - ▶ Only normalize batches during training (`model.train()`)
 - ▶ **Turn off** after training (`model.eval()`)
 - ▶ Uses running average of mean and variance
- ▶ Surprisingly effective at stabilizing training, reducing training time, and producing better models
- ▶ Not fully understood why it works

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. In *Advances in Neural Information Processing Systems* (pp. 2483-2493).

Demo of batch normalization in PyTorch

Residual networks add the input to the output of the CNN

- ▶ Most deep model layers have the form:

$$y = f(x)$$

- ▶ Where f could be any function including a convolutional layer like $f(x) = \sigma(\text{Conv}(\sigma(\text{Conv}(x))))$

- ▶ Residual layers add back in the input

$$y = f(x) + x$$

- ▶ Notice that $f(x)$ models the difference between x and y (hence the name residual)

A residual network enables deeper networks because gradient information can flow between layers

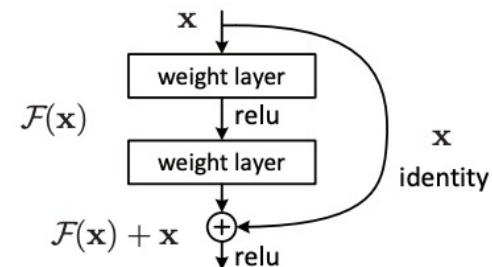
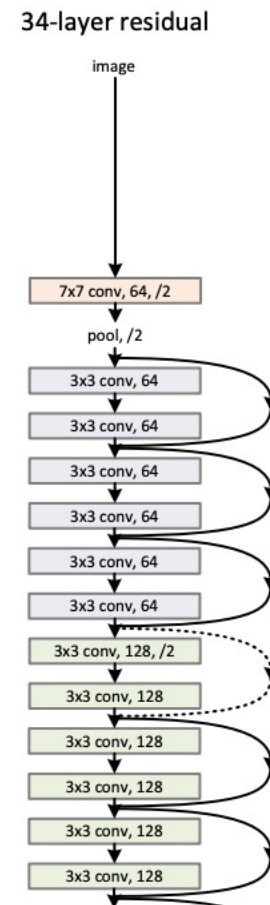


Figure 2. Residual learning: a building block.

- ▶ A data flow diagram shows the “shortcut” connections
- ▶ Consider composing 2 residual layers:
 - ▶ $z^{(1)} = f_1(x) + x$
 - ▶ $z^{(2)} = f_2(z^{(1)}) + z^{(1)}$
 - ▶ Or, equivalently

$$z^{(2)} = f_2(f_1(x) + x) + f_1(x) + x$$
- ▶ If the residuals = 0, then this is merely the identity function



Images from: He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

Detail: If the dimensionality is not the same, then use either fully connected layer or convolution layer to match

- ▶ In the 1D case, suppose $f(x): \mathbb{R}^d \rightarrow \mathbb{R}^m$, then we need to multiply x by linear operator to match the dimension

$$y = f(x) + Wx, \quad \text{where } W \in \mathbb{R}^{m \times d}$$

- ▶ Similarly, for images, if $f(x): \mathbb{R}^{c \times h \times w} \rightarrow \mathbb{R}^{c' \times h' \times w'}$, we can apply a convolution layer to match the dimensions

$$y = f(x) + \text{conv}(x),$$

where $\text{conv}(\cdot): \mathbb{R}^{c \times h \times w} \rightarrow \mathbb{R}^{c' \times h' \times w'}$

Demo of CNN with very simple residual network

U-Nets have an autoencoder structure with skip connections for **semantic segmentation** task

- ▶ Concatenation + convolution rather than residual skip connections
- ▶ **Any** (pretrained) classification backbone can be used for encoder
- ▶ State-of-the-art semantic segmentation are based on this idea

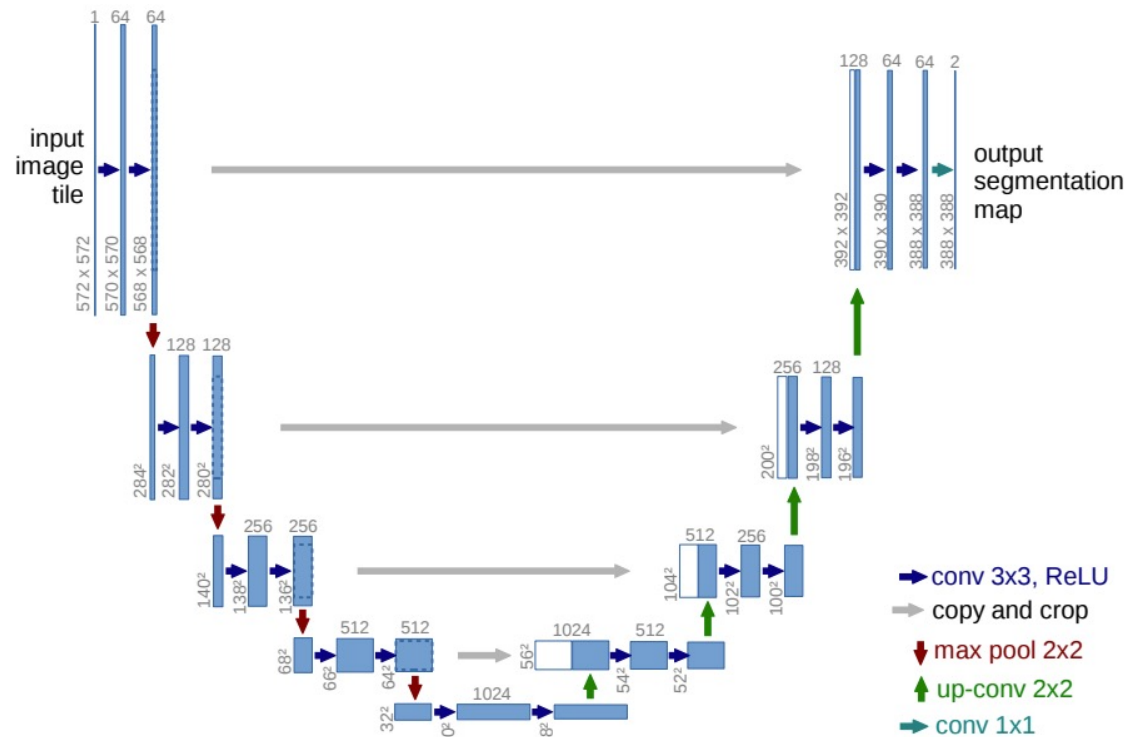


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Figure from: Ronneberger, O., Fischer, P., & Brox, T. (2015, October). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention* (pp. 234-241). Springer, Cham.