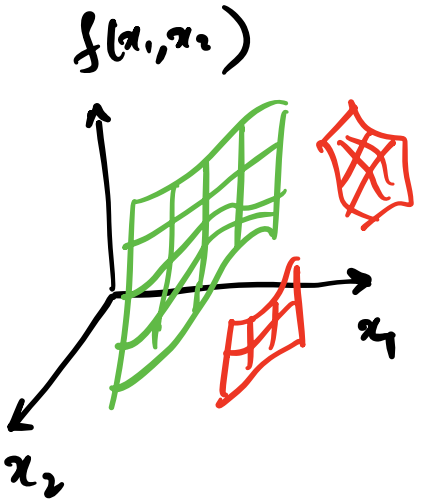
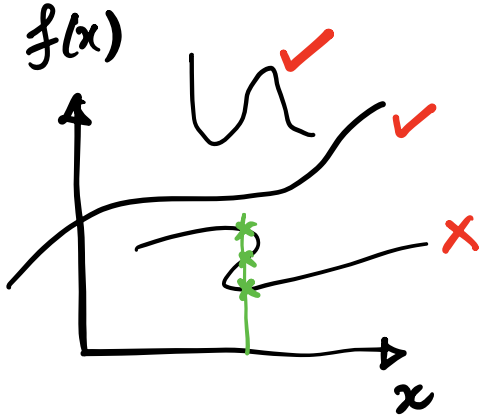
Linear layers

Computer Vision (CSCI 5520G)



Faisal Z. Qureshi

http://vclab.science.ontariotechu.ca



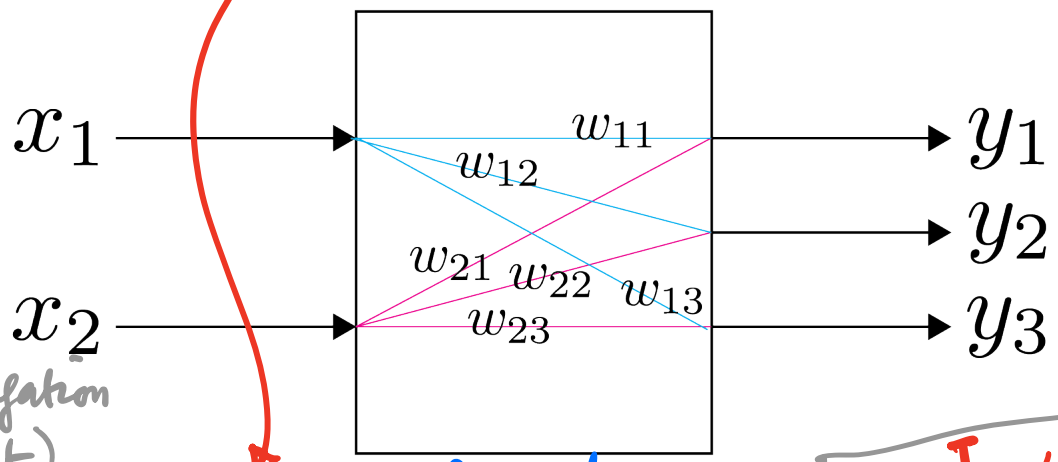
OntarioTech UNIVERSITY

$$\frac{f(x_1, x_2, x_3, \ldots, x_n)}{f(\theta_1, \theta_2, \ldots, \theta_n)}$$

$$\frac{\partial f}{\partial \theta} = \begin{bmatrix} \partial f / \partial \theta_1 \\ \vdots \\ \partial f / \partial \theta_n \end{bmatrix}$$

$$\begin{bmatrix} f(x_1, \ldots, x_n) \\ g(x_1, \ldots, x_n) \\ h(x_1, \ldots, x_n) \end{bmatrix}$$

Linear Layer



Vector-valued function

parameters

$$f: \quad y_1 = w_{11}x_1 + w_{21}x_2$$
$$g: \quad y_2 = w_{12}x_1 + w_{22}x_2$$
$$h: \quad y_3 = w_{13}x_1 + w_{23}x_2$$

output

input

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}^T = \left(\begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right)^T$$

(i) forward propagation (compute output)

(ii) ∂ output / ∂ input

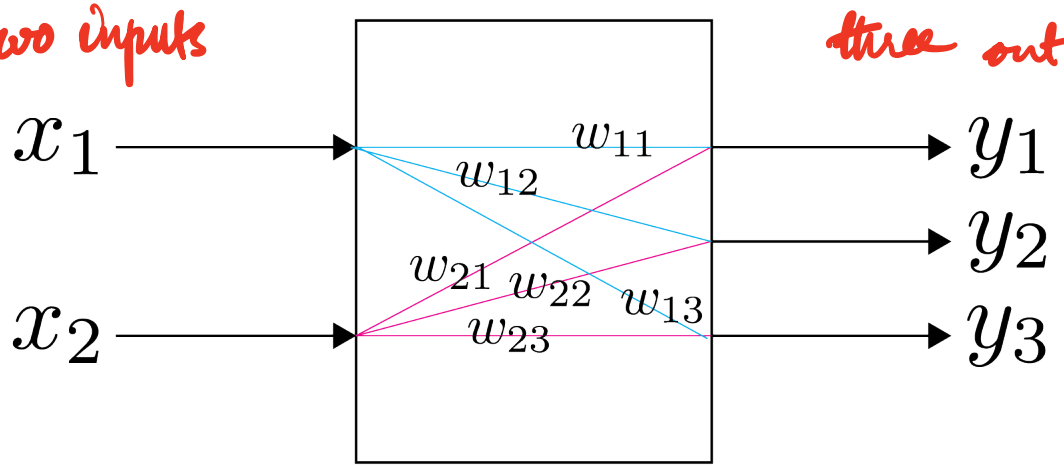
(iii) ∂ output / ∂ parameters

→ if we place this layer in a network, we should still be able to compute ∂ cost / ∂ parameters.

Linear Layer

two inputs

three outputs.



$$y_1 = w_{11}x_1 + w_{21}x_2$$

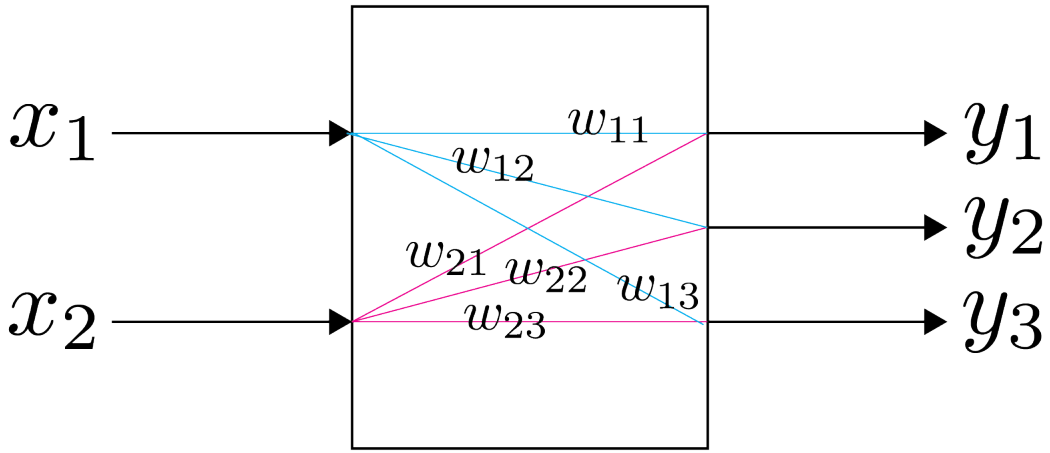
$$y_2 = w_{12}x_1 + w_{22}x_2$$

$$y_3 = w_{13}x_1 + w_{23}x_2$$

a vector-valued fn.

Re-writing in matrix form

Linear Layer



$$y_1 = w_{11}x_1 + w_{21}x_2$$
$$y_2 = w_{12}x_1 + w_{22}x_2$$
$$y_3 = w_{13}x_1 + w_{23}x_2$$

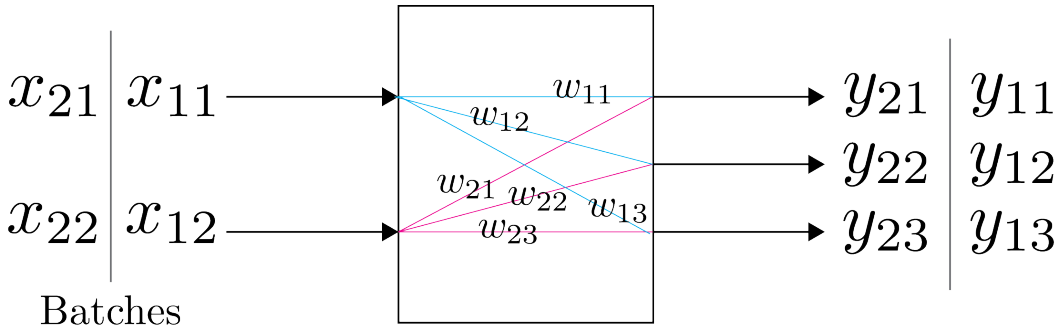
$(AB)^T = B^T A^T$

Re-writing in matrix form

$$\begin{pmatrix} y_1 & y_2 & y_3 \end{pmatrix} = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}$$

$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}^T$

Dealing with Batches



We can re-use the matrix form from before

1st item →
2nd item →

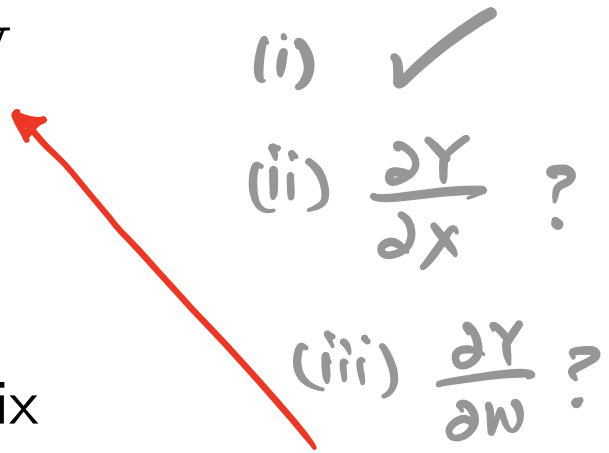
$$\underbrace{\begin{pmatrix} y_1^{(1)} & y_2^{(1)} & y_3^{(1)} \\ y_1^{(2)} & y_2^{(2)} & y_3^{(2)} \end{pmatrix}}_{\mathbf{Y}} = \underbrace{\begin{pmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \end{pmatrix}}_{\mathbf{X}} \underbrace{\begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}}_{\mathbf{W}}$$

batch batch

Forward Pass

The forward function for a linear layer is thus defined as

$$\mathbf{Y} = \mathbf{X}\mathbf{W}$$



where

- $\mathbf{X} \in \mathbb{R}^{B \times d_{\text{in}}}$ is the input
- $\mathbf{W} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ is the weight matrix
- $\mathbf{Y} \in \mathbb{R}^{B \times d_{\text{out}}}$ is the output

Here B refers to the batch size.

Forward Pass

The forward function for a linear layer is thus defined as

$$\mathbf{Y} = \mathbf{X}\mathbf{W}$$

where

- $\mathbf{X} \in \mathbb{R}^{B \times d_{\text{in}}}$ is the input
- $\mathbf{W} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ is the weight matrix
- $\mathbf{Y} \in \mathbb{R}^{B \times d_{\text{out}}}$ is the output

Here B refers to the batch size.

How would you handle the bias term?

Forward Pass

The forward function for a linear layer is thus defined as

$$\mathbf{Y} = \mathbf{X}\mathbf{W}$$

where

- $\mathbf{X} \in \mathbb{R}^{B \times d_{\text{in}}}$ is the input
- $\mathbf{W} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ is the weight matrix
- $\mathbf{Y} \in \mathbb{R}^{B \times d_{\text{out}}}$ is the output

Here B refers to the batch size.

How would you handle the bias term? Append a 1 to each input in $\mathbf{X}$.

Backpropagation

To backpropagate, we need:

- Gradient w.r.t input: $\frac{\partial C}{\partial \mathbf{X}}$
- Gradient w.r.t weights: $\frac{\partial C}{\partial \mathbf{W}}$

We assume that we already have $\frac{\partial C}{\partial \mathbf{Y}}$. This was backpropagated by later layers. Here C denotes loss or cost that we need to minimize to "train" the network.

Backpropagation

To backpropagate, we need:

- Gradient w.r.t input: $\frac{\partial C}{\partial \mathbf{X}}$
- Gradient w.r.t weights: $\frac{\partial C}{\partial \mathbf{W}}$

We assume that we already have $\frac{\partial C}{\partial \mathbf{Y}}$. This was backpropagated by later layers. Here C denotes loss or cost that we need to minimize to "train" the network.

How to compute $\frac{\partial C}{\partial \mathbf{X}}$?

Application of chain-rule yields

$$\frac{\partial C}{\partial \mathbf{X}} = \frac{\partial C}{\partial \mathbf{Y}} \frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$$

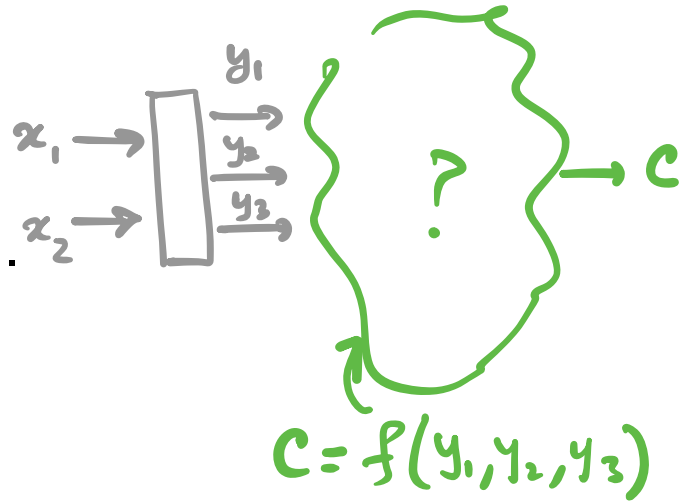
As have $\frac{\partial C}{\partial \mathbf{Y}}$ already. Lets figure out how to compute $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$.

Structure of $\frac{\partial C}{\partial \mathbf{Y}}$

- C (loss or cost) is a scalar.
- $\frac{\partial C}{\partial \mathbf{Y}}$ as the same size as $\mathbf{Y}$, i.e. $(B \times d_{\text{out}})$.

Structure of $\frac{\partial C}{\partial \mathbf{Y}}$

- C (loss or cost) is a scalar.
- $\frac{\partial C}{\partial \mathbf{Y}}$ as the same size as $\mathbf{Y}$, i.e. $(B \times d_{\text{out}})$.



For the **single input** case: $(x_1, x_2) \mapsto (y_1, y_2, y_3)$

$$\rightarrow \quad \frac{\partial C}{\partial \mathbf{Y}} = \begin{pmatrix} \frac{\partial C}{\partial y_1} & \frac{\partial C}{\partial y_2} & \frac{\partial C}{\partial y_3} \end{pmatrix}$$

Structure of $\frac{\partial C}{\partial \mathbf{Y}}$

- C (loss or cost) is a scalar.
- $\frac{\partial C}{\partial \mathbf{Y}}$ as the same size as $\mathbf{Y}$, i.e. $(B \times d_{\text{out}})$.

For the **batch input** case: $\begin{pmatrix} x_1^{(1)}, x_2^{(1)} \\ x_1^{(2)}, x_2^{(2)} \end{pmatrix} \mapsto \begin{pmatrix} y_1^{(1)}, y_2^{(1)}, y_3^{(1)} \\ y_1^{(2)}, y_2^{(2)}, y_3^{(2)} \end{pmatrix}$

$$\frac{\partial C}{\partial \mathbf{Y}} = \begin{pmatrix} \frac{\partial C}{\partial y_1^{(1)}} & \frac{\partial C}{\partial y_2^{(1)}} & \frac{\partial C}{\partial y_3^{(1)}} \\ \frac{\partial C}{\partial y_1^{(2)}} & \frac{\partial C}{\partial y_2^{(2)}} & \frac{\partial C}{\partial y_3^{(2)}} \end{pmatrix}$$

Computing $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$

Lets unpack $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$. We begin by considering a **single input**

$$\begin{pmatrix} y_1 & y_2 & y_3 \end{pmatrix} = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}$$

Expanding further

$$\begin{aligned} y_1 &= w_{11}x_1 + w_{21}x_2 \\ y_2 &= w_{12}x_1 + w_{22}x_2 \\ y_3 &= w_{13}x_1 + w_{23}x_2 \end{aligned}$$

This is a vector-valued function of two variables (x_1, x_2).

Computing $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$

Setting up the Jacobian (the matrix of all first-order partial derivatives of a vector-valued function)

$$\begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \\ \frac{\partial y_3}{\partial x_1} & \frac{\partial y_3}{\partial x_2} \end{pmatrix} = \begin{pmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{pmatrix} = \mathbf{W}^T$$

Computing $\frac{\partial C}{\partial \mathbf{X}}$

Lets compute $\frac{\partial C}{\partial x_1}$ and $\frac{\partial C}{\partial x_2}$

Computing $\frac{\partial C}{\partial \mathbf{X}}$

Lets compute $\frac{\partial C}{\partial x_1}$ and $\frac{\partial C}{\partial x_2}$

$$\frac{\partial C}{\partial x_1} = \frac{\partial C}{\partial y_1}\frac{\partial y_1}{\partial x_1} + \frac{\partial C}{\partial y_2}\frac{\partial y_2}{\partial x_1} + \frac{\partial C}{\partial y_3}\frac{\partial y_3}{\partial x_1}$$

Computing $\frac{\partial C}{\partial \mathbf{X}}$

Lets compute $\frac{\partial C}{\partial x_1}$ and $\frac{\partial C}{\partial x_2}$

$$\frac{\partial C}{\partial x_1} = \frac{\partial C}{\partial y_1}\frac{\partial y_1}{\partial x_1} + \frac{\partial C}{\partial y_2}\frac{\partial y_2}{\partial x_1} + \frac{\partial C}{\partial y_3}\frac{\partial y_3}{\partial x_1}$$

Similarly

$$\frac{\partial C}{\partial x_2} = \frac{\partial C}{\partial y_1}\frac{\partial y_1}{\partial x_2} + \frac{\partial C}{\partial y_2}\frac{\partial y_2}{\partial x_2} + \frac{\partial C}{\partial y_3}\frac{\partial y_3}{\partial x_2}$$

Computing $\frac{\partial C}{\partial \mathbf{X}}$

Lets compute $\frac{\partial C}{\partial x_1}$ and $\frac{\partial C}{\partial x_2}$

$$\frac{\partial C}{\partial x_1} = \frac{\partial C}{\partial y_1}\frac{\partial y_1}{\partial x_1} + \frac{\partial C}{\partial y_2}\frac{\partial y_2}{\partial x_1} + \frac{\partial C}{\partial y_3}\frac{\partial y_3}{\partial x_1}$$

Similarly

$$\frac{\partial C}{\partial x_2} = \frac{\partial C}{\partial y_1}\frac{\partial y_1}{\partial x_2} + \frac{\partial C}{\partial y_2}\frac{\partial y_2}{\partial x_2} + \frac{\partial C}{\partial y_3}\frac{\partial y_3}{\partial x_2}$$

Re-writing in matrix form

$$\begin{pmatrix} \frac{\partial C}{\partial x_1} & \frac{\partial C}{\partial x_2} \end{pmatrix} = \begin{pmatrix} \frac{\partial C}{\partial y_1} & \frac{\partial C}{\partial y_2} & \frac{\partial C}{\partial y_3} \end{pmatrix} \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \\ \frac{\partial y_3}{\partial x_1} & \frac{\partial y_3}{\partial x_2} \end{pmatrix}$$

Computing $\frac{\partial C}{\partial \mathbf{X}}$

Lets compute $\frac{\partial C}{\partial x_1}$ and $\frac{\partial C}{\partial x_2}$

$$\frac{\partial C}{\partial x_1} = \frac{\partial C}{\partial y_1}\frac{\partial y_1}{\partial x_1} + \frac{\partial C}{\partial y_2}\frac{\partial y_2}{\partial x_1} + \frac{\partial C}{\partial y_3}\frac{\partial y_3}{\partial x_1}$$

Similarly

$$\frac{\partial C}{\partial x_2} = \frac{\partial C}{\partial y_1}\frac{\partial y_1}{\partial x_2} + \frac{\partial C}{\partial y_2}\frac{\partial y_2}{\partial x_2} + \frac{\partial C}{\partial y_3}\frac{\partial y_3}{\partial x_2}$$

Re-writing in matrix form

$$\begin{pmatrix} \frac{\partial C}{\partial x_1} & \frac{\partial C}{\partial x_2} \end{pmatrix} = \begin{pmatrix} \frac{\partial C}{\partial y_1} & \frac{\partial C}{\partial y_2} & \frac{\partial C}{\partial y_3} \end{pmatrix} \underbrace{\begin{pmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{pmatrix}}_{\mathbf{W}^T}$$

Dealing with batch inputs

Lets consider **batch input**

$$\underbrace{\begin{pmatrix} y_1^{(1)} & y_2^{(1)} & y_3^{(1)} \\ y_1^{(2)} & y_2^{(2)} & y_3^{(2)} \end{pmatrix}}_{\mathbf{Y}} = \underbrace{\begin{pmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \end{pmatrix}}_{\mathbf{X}} \underbrace{\begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}}_{\mathbf{W}}$$

We write

$$\begin{pmatrix} \frac{\partial C}{\partial x_1^{(1)}} & \frac{\partial C}{\partial x_2^{(1)}} \\ \\ \frac{\partial C}{\partial x_1^{(2)}} & \frac{\partial C}{\partial x_2^{(2)}} \end{pmatrix} = \begin{pmatrix} \frac{\partial C}{\partial y_1^{(1)}} & \frac{\partial C}{\partial y_2^{(1)}} & \frac{\partial C}{\partial y_3^{(1)}} \\ \\ \frac{\partial C}{\partial y_1^{(2)}} & \frac{\partial C}{\partial y_2^{(2)}} & \frac{\partial C}{\partial y_3^{(2)}} \end{pmatrix} \mathbf{W}^T$$

Therefore

$$\frac{\partial C}{\partial \mathbf{X}} = \frac{\partial C}{\partial \mathbf{Y}} \mathbf{W}^T$$

Computing $\frac{\partial C}{\partial \mathbf{W}}$

Now let's turn our attention to computing $\frac{\partial C}{\partial \mathbf{W}}$.

Recall $\frac{\partial C}{\partial \mathbf{W}}$ has the same shape as $\mathbf{W}$

$$\frac{\partial C}{\partial \mathbf{W}} = \begin{pmatrix} \frac{\partial C}{\partial w_{11}} & \frac{\partial C}{\partial w_{12}} & \frac{\partial C}{\partial w_{13}} \\ \\ \frac{\partial C}{\partial w_{21}} & \frac{\partial C}{\partial w_{22}} & \frac{\partial C}{\partial w_{23}} \end{pmatrix}$$

Computing $\frac{\partial C}{\partial \mathbf{W}}$

Applying chain-rule, we have

$$\frac{\partial C}{\partial \mathbf{W}} = \underbrace{\frac{\partial C}{\partial \mathbf{Y}}}_{\delta} \frac{\partial \mathbf{Y}}{\partial \mathbf{W}}$$

Computing $\frac{\partial C}{\partial \mathbf{W}}$

Applying chain-rule, we have

$$\frac{\partial C}{\partial \mathbf{W}} = \underbrace{\frac{\partial C}{\partial \mathbf{Y}}}_{\checkmark} \frac{\partial \mathbf{Y}}{\partial \mathbf{W}}$$

Computing $\frac{\partial C}{\partial \mathbf{W}}$

(iii) ∂ output / ∂ parameters

Applying chain-rule, we have

$$\frac{\partial C}{\partial \mathbf{W}} = \underbrace{\frac{\partial C}{\partial \mathbf{Y}}}_{\checkmark} \underbrace{\frac{\partial \mathbf{Y}}{\partial \mathbf{W}}}_{?}$$

Computing $\frac{\partial C}{\partial w_{11}}, \cdots$

For **single input** case:

$y_1 = w_{11}x_1 + w_{21}x_2$, $y_2 = w_{12}x_1 + w_{22}x_2$, and $y_3 = w_{13}x_1 + w_{23}x_2$

Applying chain-rule, we have

$$\begin{aligned}\frac{\partial C}{\partial w_{11}} &= \frac{\partial C}{\partial y_1}\frac{\partial y_1}{\partial w_{11}} + \frac{\partial C}{\partial y_2}\frac{\partial y_2}{\partial w_{11}} + \frac{\partial C}{\partial y_3}\frac{\partial y_3}{\partial w_{11}} \\ &= \frac{\partial C}{\partial y_1}x_1\end{aligned}$$

Computing $\frac{\partial C}{\partial w_{11}}, \cdots$

For **single input** case:

$y_1 = w_{11}x_1 + w_{21}x_2$, $y_2 = w_{12}x_1 + w_{22}x_2$, and $y_3 = w_{13}x_1 + w_{23}x_2$

Applying chain-rule, we have

$$\begin{aligned}\frac{\partial C}{\partial w_{11}} &= \frac{\partial C}{\partial y_1}\frac{\partial y_1}{\partial w_{11}} + \frac{\partial C}{\partial y_2}\frac{\partial y_2}{\partial w_{11}} + \frac{\partial C}{\partial y_3}\frac{\partial y_3}{\partial w_{11}} \\ &= \frac{\partial C}{\partial y_1}x_1\end{aligned}$$

Simlarly

$\frac{\partial C}{\partial w_{12}} = \frac{\partial C}{\partial y_2}x_1$, $\frac{\partial C}{\partial w_{13}} = \frac{\partial C}{\partial y_3}x_1$,

$\frac{\partial C}{\partial w_{21}} = \frac{\partial C}{\partial y_1}x_2$, $\frac{\partial C}{\partial w_{22}} = \frac{\partial C}{\partial y_2}x_2$, and $\frac{\partial C}{\partial w_{23}} = \frac{\partial C}{\partial y_3}x_2$

Computing $\frac{\partial C}{\partial \mathbf{W}}$

Putting it all together for **single input** case

$$\frac{\partial C}{\partial \mathbf{W}} = \begin{pmatrix} \frac{\partial C}{\partial w_{11}} & \frac{\partial C}{\partial w_{12}} & \frac{\partial C}{\partial w_{13}} \\ \frac{\partial C}{\partial w_{21}} & \frac{\partial C}{\partial w_{22}} & \frac{\partial C}{\partial w_{23}} \end{pmatrix}$$

$$= \begin{pmatrix} \frac{\partial C}{\partial y_1} x_1 & \frac{\partial C}{\partial y_2} x_1 & \frac{\partial C}{\partial y_3} x_1 \\ \frac{\partial C}{\partial y_1} x_2 & \frac{\partial C}{\partial y_2} x_2 & \frac{\partial C}{\partial y_3} x_2 \end{pmatrix}$$

$$= \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \begin{pmatrix} \frac{\partial C}{\partial y_1} & \frac{\partial C}{\partial y_2} & \frac{\partial C}{\partial y_3} \end{pmatrix}$$

Computing $\frac{\partial C}{\partial \mathbf{W}}$

Putting it all together for **single input** case

$$
\begin{aligned}
\frac{\partial C}{\partial \mathbf{W}} &= \begin{pmatrix} \frac{\partial C}{\partial w_{11}} & \frac{\partial C}{\partial w_{12}} & \frac{\partial C}{\partial w_{13}} \\ \\ \frac{\partial C}{\partial w_{21}} & \frac{\partial C}{\partial w_{22}} & \frac{\partial C}{\partial w_{23}} \end{pmatrix} \\
&= \begin{pmatrix} \frac{\partial C}{\partial y_1} x_1 & \frac{\partial C}{\partial y_2} x_1 & \frac{\partial C}{\partial y_3} x_1 \\ \\ \frac{\partial C}{\partial y_1} x_2 & \frac{\partial C}{\partial y_2} x_2 & \frac{\partial C}{\partial y_3} x_2 \end{pmatrix} \\
&= \underbrace{\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}}_{\mathbf{X}^T} \underbrace{\begin{pmatrix} \frac{\partial C}{\partial y_1} & \frac{\partial C}{\partial y_2} & \frac{\partial C}{\partial y_3} \end{pmatrix}}_{\frac{\partial C}{\partial \mathbf{Y}}}
\end{aligned}
$$

Computing $\frac{\partial C}{\partial \mathbf{W}}$

For **batch input** case

$$\frac{\partial C}{\partial \mathbf{W}} = \underbrace{\begin{pmatrix} x_1^{(1)} & x_1^{(2)} \\ x_2^{(1)} & x_2^{(2)} \end{pmatrix}}_{\mathbf{X}^T} \begin{pmatrix} \frac{\partial C}{\partial y_1^{(1)}} & \frac{\partial C}{\partial y_2^{(1)}} & \frac{\partial C}{\partial y_3^{(1)}} \\ \frac{\partial C}{\partial y_1^{(2)}} & \frac{\partial C}{\partial y_2^{(2)}} & \frac{\partial C}{\partial y_3^{(2)}} \end{pmatrix}$$

Or more generally, we write

$$\frac{\partial C}{\partial \mathbf{W}} = \mathbf{X}^T \frac{\partial C}{\partial \mathbf{Y}}$$

Observations

- By using the *backpropagated* signal $\frac{\partial C}{\partial \mathbf{Y}}$, we are able to compute $\frac{\partial C}{\partial \mathbf{X}}$ and $\frac{\partial C}{\partial \mathbf{W}}$.
  - We still need to compute $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ and $\frac{\partial \mathbf{Y}}{\partial \mathbf{W}}$
- $\frac{\partial C}{\partial \mathbf{X}}$ is *backpropagated*.
- $\frac{\partial C}{\partial \mathbf{W}}$ is used to update weights $\mathbf{W}$ of this layer.

$f(x)$: $\frac{df}{dx}$, f'

$f(x_1, x_2)$: $\frac{\partial f}{\partial x_1}$, f_{x_1}

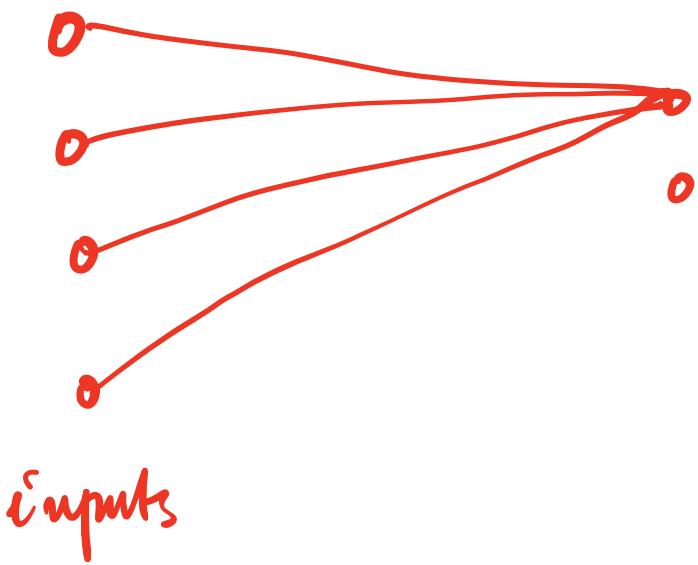
$\frac{\partial f}{\partial x_2}$, f_{x_2}

Useful Properties of Linear Layers

- **Global context**: Every output depends on every input.
- **Information mixing**: Inputs are combined to produce outputs.
- **Common usage**: Often the last layer in many networks.

Activation Functions

This description **ignores activation functions**, which are usually applied after linear layers.



Copyright and License

©Faisal Z. Qureshi

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Convolutional Neural Networks

Computer Vision (CSCI 5520G)

Faisal Z. Qureshi

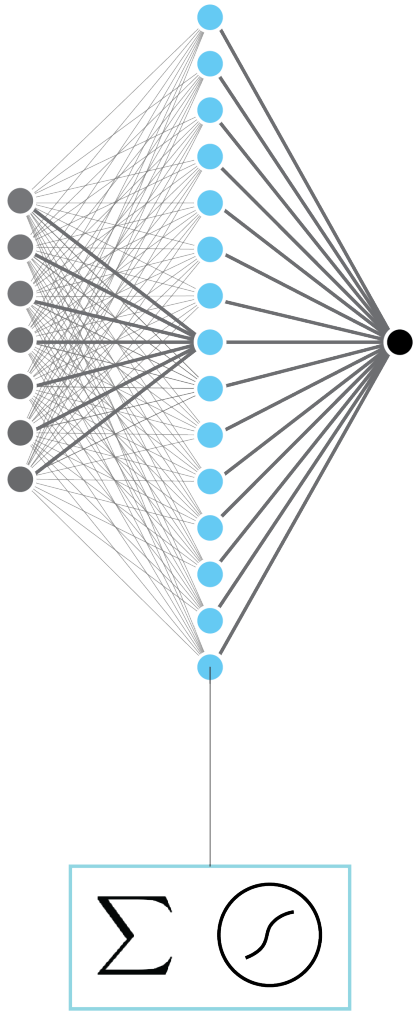
http://vclab.science.ontariotechu.ca

Lesson Plan

- Convolutional Networks
- Convolution
- Pooling layers
- Dilated convolutions
- Common architecture
  - GoogLeNet
  - ResNet
  - Densenet
  - Squeeze-and-Excitation network
- ConvNext

Classical neural networks for computer vision tasks

- **Q.** How many paramters per hidden layer unit?
- Computational issues
- Poor performance
  - The model is prone to overfitting
  - Model capacity issues



Convolutional neural networks

David Hubel and Torsten Wiesel studied cat visual cortex and showed that visual information goes through a series of processing steps (Hubeland Wiesel, 1959):

- edge detection;
- edge combination; and
- motion perception; etc.

Convolutional neural networks

David Hubel and Torsten Wiesel studied cat visual cortex and showed that visual information goes through a series of processing steps (Hubeland Wiesel, 1959):

- edge detection;
- edge combination; and
- motion perception; etc.

This suggests

- Neurons are spatially localized
- Topographic feature maps
- Hierarchical feature processing

Convolutional neural networks

David Hubel and Torsten Wiesel studied cat visual cortex and showed that visual information goes through a series of processing steps (Hubeland Wiesel, 1959):

- edge detection;
- edge combination; and
- motion perception; etc.

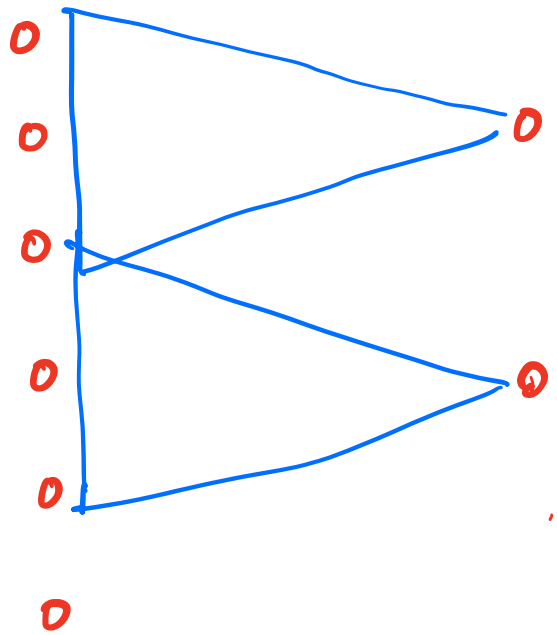
This suggests

- Neurons are spatially localized
- Topographic feature maps
- Hierarchical feature processing

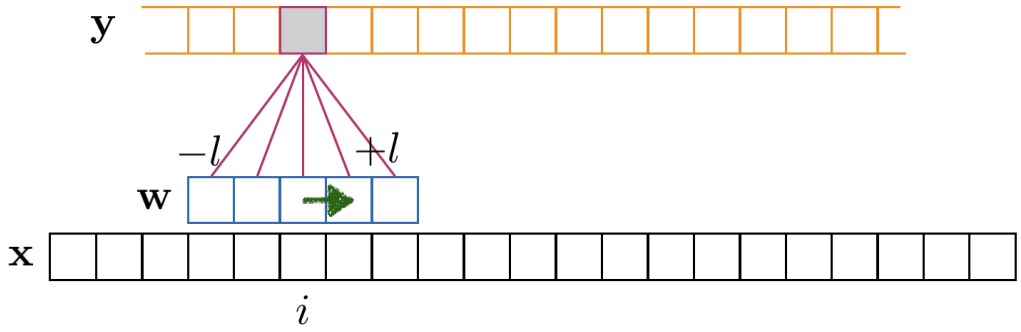
Convolutional layers achieve these properties

Convolutional layers

- Each output unit is a linear function of a localized subset of input units
- Same linear transformation is applied at each location
- Local features detection is translation invariant



Convolutions in 1D



Signal (x): $(x_0, \cdots, x_{n-1}) \in \mathbb{R}^n$

Kernel or filter (w): $(w_{-l}, \cdots, , w_l) \in \mathbb{R}^{2l+1}$

Output value at location j:

$$y_i = \sum_{i'=-l}^{l} w_{i'} x_{i-i'}$$

The entire process is represented as $\mathbf{y} = \mathbf{x} * \mathbf{w}$

parameters

forward fn. of a convolution layer

Convolutions vs. Cross-Correlation

Convolution

$$y_i = \sum_{i'=-l}^{l} w_{i'} x_{\underbrace{i - i'}_{\text{flipped}}}$$

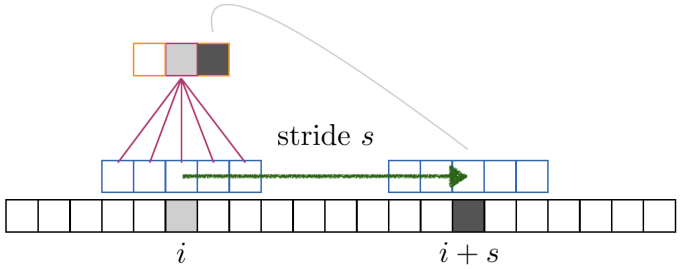
Cross-correlation

$$y_i = \sum_{i'=-l}^{l} w_{i'} x_{\underbrace{i + i'}_{\text{not flipped}}}$$

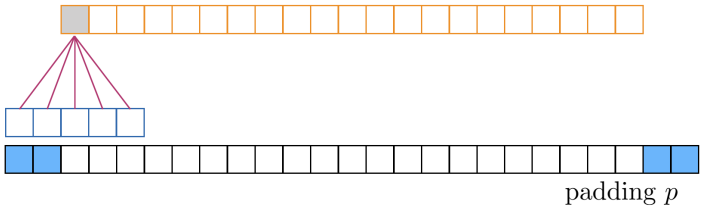
- Mathematical **convolution** flips the kernel; **cross-correlation** does not.
- Deep learning libraries typically implement **cross-correlation**, relying on learning to absorb the flip.

Stride, Padding, and Dilation

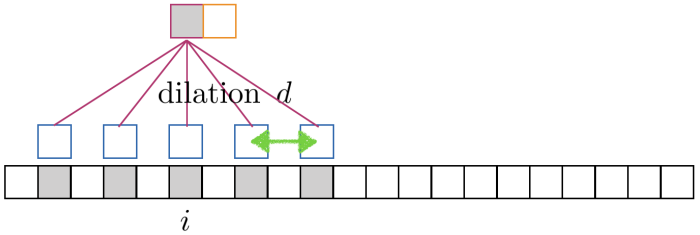
Stride s: number of positions we shift the kernel each step.



Padding p: number of zeros (or other values) added at boundaries.



Dilation d: increasing the input field for the kernel. Dilated convolutions are sometimes called atrous convolutions. (From French '\{a} trous" meaning with holes.)



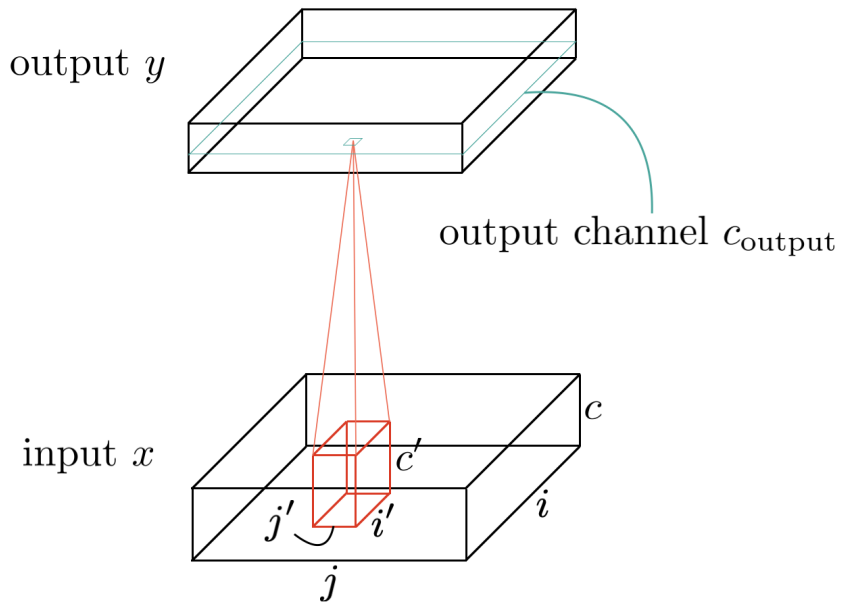
Size of the output

For 1D, output length:

$$L_{\text{out}} = \left\lfloor \frac{L_{\text{in}} + 2p - \overbrace{(d\,(K-1)+1)}^{\text{effective kernel size}}}{s} \right\rfloor + 1$$

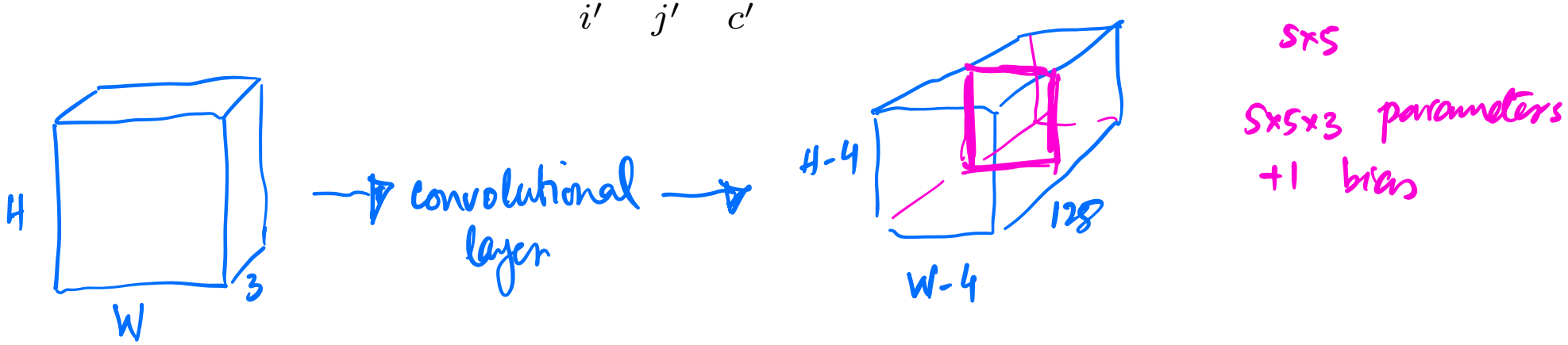
where p, s, and d refers to padding, stride, and dilation, respectively. $K = 2l + 1$ is the length of the kernel and L_{in} is the length of the input signal.

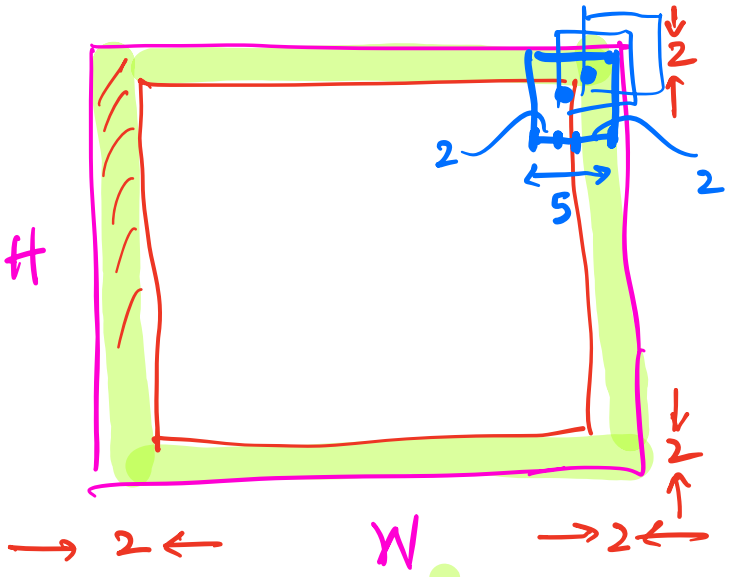
Convolutions in Higher Dimensions

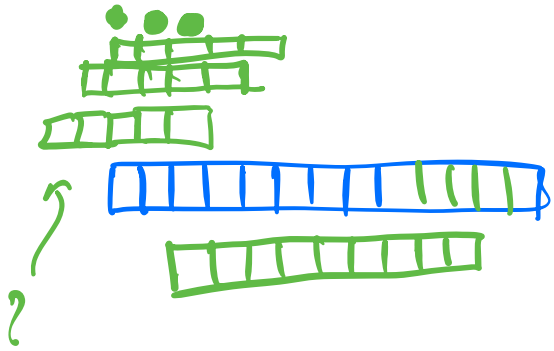


i, j and c index over input x height, width, and channels.
i', j' and c' index over filter w height, width, and channels.
Filter w creates output channel c_{output}

$$y_{ij,c_{\text{output}}} = \sum_{i'} \sum_{j'} \sum_{c'} w_{i'\,j'\,c'} x_{i-i'\,j-j'\,c-c'}$$







3×4

3	0	1	4
5	7	8	9
0	0	1	-1

*

3×3

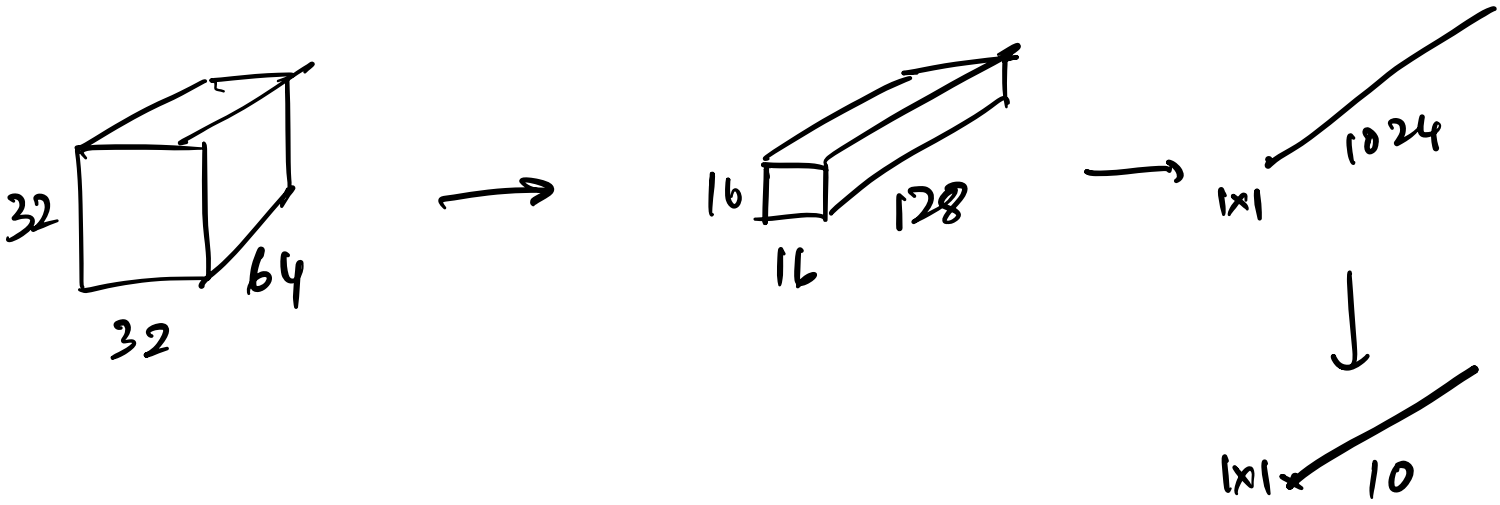
1	1	1
0	0	0
1	1	1

↓

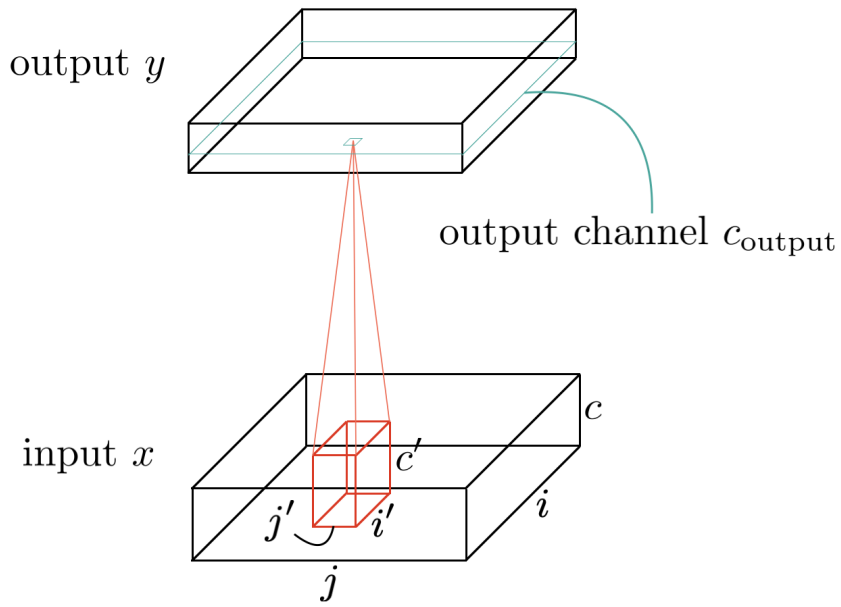
5	5

$(3)(1) + (0)(1) + (1)(1) + (5)(0) + (7)(0)$

$+(8)(0) + (0)(1) + (0)(1) + (1)(1)$


32
64
32
16
128
16
1×1
1024
1×1
10

Convolutions in Higher Dimensions

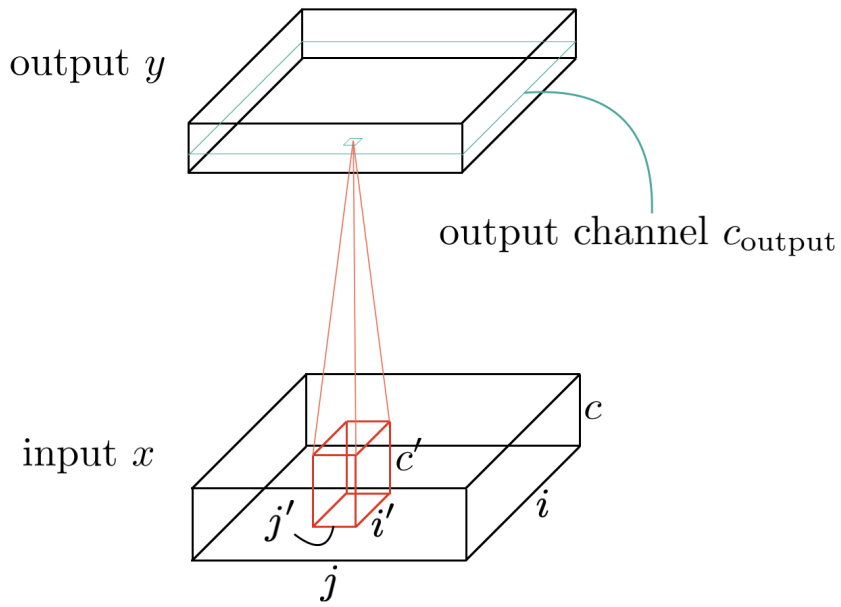


i, j and c index over input x height, width, and channels.
i', j' and c' index over filter w height, width, and channels.
Filter w creates output channel c_{output}

$$y_{ij,c_{\text{output}}} = \sum_{i'} \sum_{j'} \sum_{c'} w_{i'\,j'\,c'} x_{i-i'\,j-j'\,c-c'}$$

How do we extend this to deal with the situation when the output has more than one channels?

Convolutions in Higher Dimensions



i, j and c index over input x height, width, and channels.
i', j' and c' index over filter w height, width, and channels.
Filter w creates output channel c_{output}

$$y_{ij,c_{\text{output}}} = \sum_{i'} \sum_{j'} \sum_{c'} w_{i'\,j'\,c'} x_{i-i'\,j-j'\,c-c'}$$

How do we extend this to deal with the situation when the output has more than one channels? **Use one kernel per output channel.**

Number of Parameters

Consider a convolutional layer that takes an input feature map of size $H \times W \times C$ and create an output feature map of size $H' \times W' \times C'$. The convolutional layer uses an $h \times w$ filter. Write down the number of parameters for this convolutional layer.

Number of Parameters

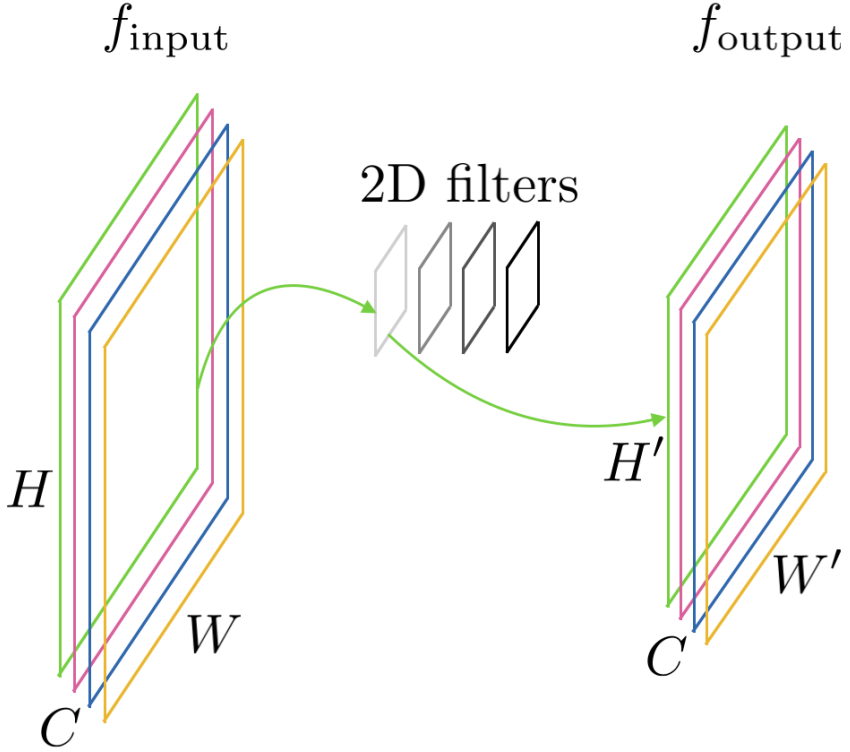
Consider a convolutional layer that takes an input feature map of size $H \times W \times C$ and create an output feature map of size $H' \times W' \times C'$. The convolutional layer uses an $h \times w$ filter. Write down the number of parameters for this convolutional layer.

Answer

$$((\underbrace{h \times w}_{\text{height and width}})(\underbrace{C}_{\text{input channels}}) + \underbrace{1}_{\text{bias}})(\underbrace{C'}_{\text{output channels}})$$

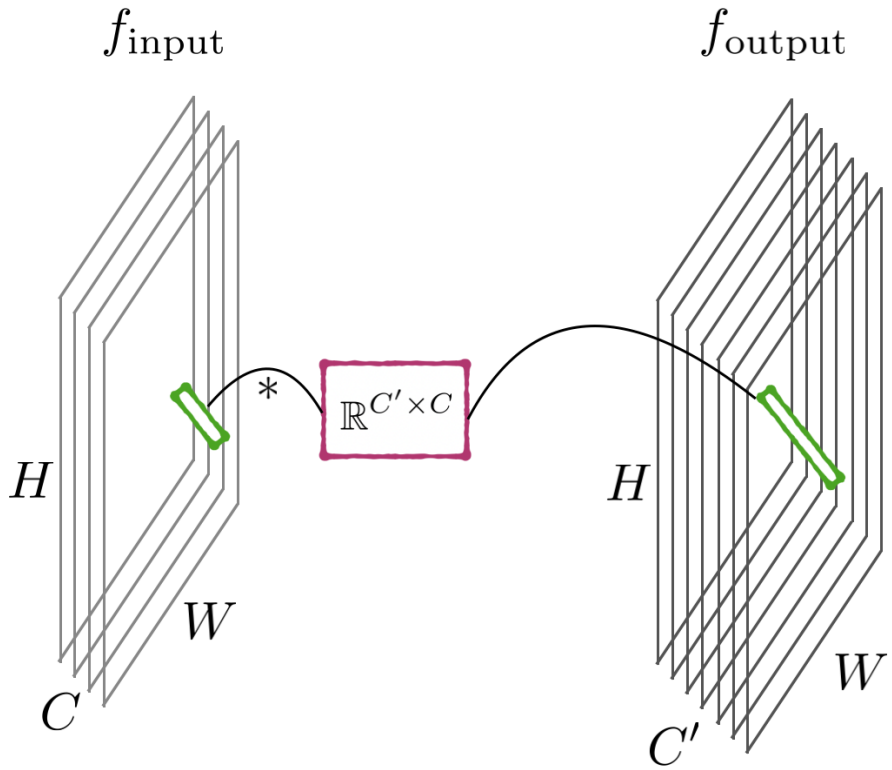
Depth-wise (channel-wise) Convolutions

Take an $H \times W \times C$ input feature f_{input} and map it to an $H' \times W' \times C$ feature f_{output} such that each output channel $c_{\text{output}} = c_{\text{input}} * w$, where c_{input} is the corresponding input channel and w is a 2D convolutional filter. E.g., output channel 2 depends only upon input channel 2. $*$ denotes the convolutional operator.



Point-wise (1×1) Convolutions

Take a $H \times W \times C$ input feature f_{input} and map it to a $H \times W \times C'$ output feature f_{output} such that each location $f_{\text{output}} = [i, j, c_{\text{output}}] = w_{c_{\text{output}}} \cdot f_{\text{input}}[i, j]$. Here $\cdot$ refers to dot product. $c_{output} \in [1, C']$.



Depth-wise Separable Convolutions

Depth-wise convolution (per channel spatial filter) + point-wise (1×1) convolution for cross-channel mixing

Consider a convolutional layer that takes an input feature map of size $H \times W \times C$ and create an output feature map of size $H' \times W' \times C'$. The convolutional layer uses C $h \times w$ filters followed by C' point-wise filters. Write down the number of parameters for this convolutional layer.

Depth-wise Separable Convolutions

Depth-wise convolution (per channel spatial filter) + point-wise (1×1) convolution for cross-channel mixing

Consider a convolutional layer that takes an input feature map of size $H \times W \times C$ and create an output feature map of size $H' \times W' \times C'$. The convolutional layer uses C $h \times w$ filters followed by C' point-wise filters. Write down the number of parameters for this convolutional layer.

Answer

$$\underbrace{((h \times w + 1)(C))}_{\text{depth-wise}} + \underbrace{(C \times C')}_{\text{point-wise}}$$

Depth-wise Separable Convolutions

Depth-wise convolution (per channel spatial filter) + point-wise (1×1) convolution for cross-channel mixing

Consider a convolutional layer that takes an input feature map of size $H \times W \times C$ and create an output feature map of size $H' \times W' \times C'$. The convolutional layer uses C $h \times w$ filters followed by C' point-wise filters. Write down the number of parameters for this convolutional layer.

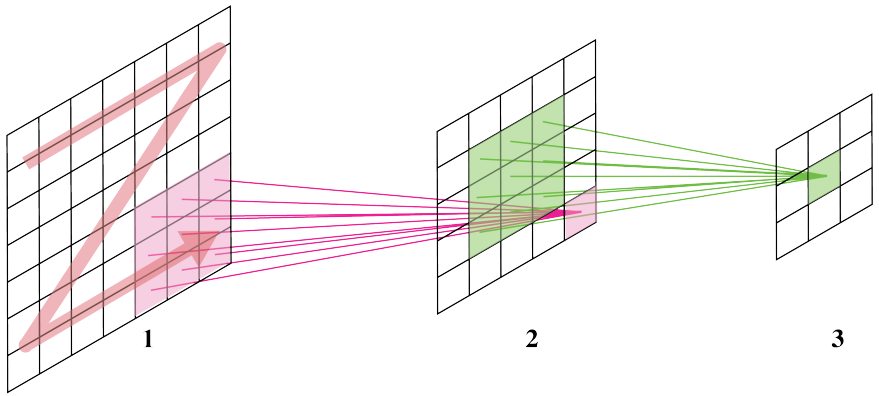
Answer

$$\underbrace{((h \times w + 1)(C))}_{\text{depth-wise}} + \underbrace{(C \times C')}_{\text{point-wise}}$$

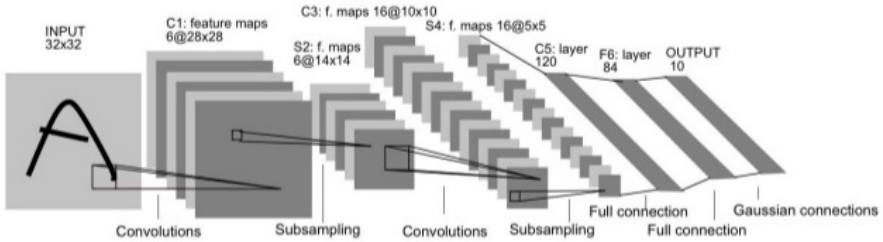
Depth-wise and point-wise convolutions are combined to cutdown on parameters and computation.

Convolutional neural network

- Convolutional layers provide architectural constraints
- Number of parameters depend upon kernel sizes and not the size of the input
- Inductive bias
  - Architectural constraints
  - Image augmentation
  - Regularization



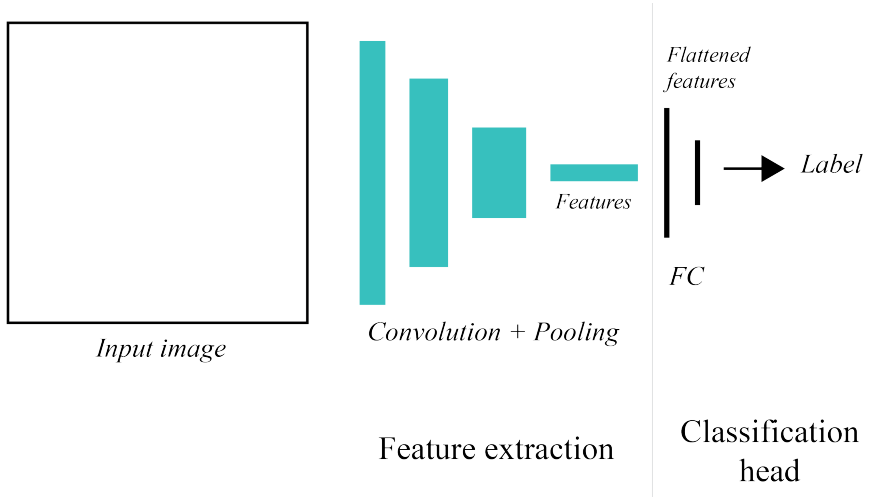
LeNet (classifying digits, LeCun 1988)



- The first few layers are convolution layers, and the last few layers are fully connected layers
  - Convolutional layers construct image features that are subsequently processed by fully connected layers for the purposes of digit classifisation
- The convolutional layers are compute heavy, but have fewer parameters
- The fully connected layer have far more parameters, but these are easy to compute

General idea

- Generally speaking we can interpret convolutional deep networks as composed of two parts: 1) a (latent) feature extractor and 2) task head.
- Feature extractor learns to construct powerful representations given an input. These representations are well-suited to the task at hand.
  - Feature extractor is often composed of convolution layers

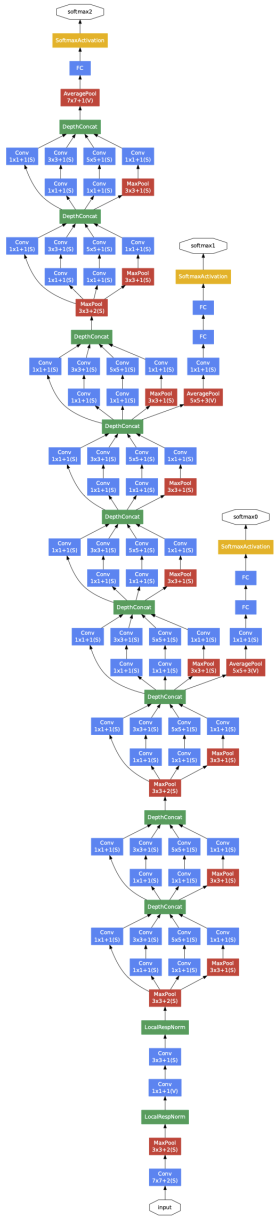


GoogLeNet

- Szegedy et al. 2014

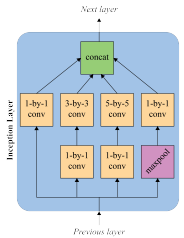
  Going Deeper with Convolutions

- Multiple feed-forward passes
- Inception module
  - An inception module aims to approximate local sparse structure in a CNN by using filters of different sizes (within the same block) whose output is concatenated and passed on to the next stage



Inception layer

- Acts as a bottleneck layer
- 1-by-1 convolutional layers are used to reduce feature channels
- Inception layer (simplified). Each conv is followed by a non-linear activation.



ResNet

- He et al. 2016
  Deep Residual Learning for Image Recognition

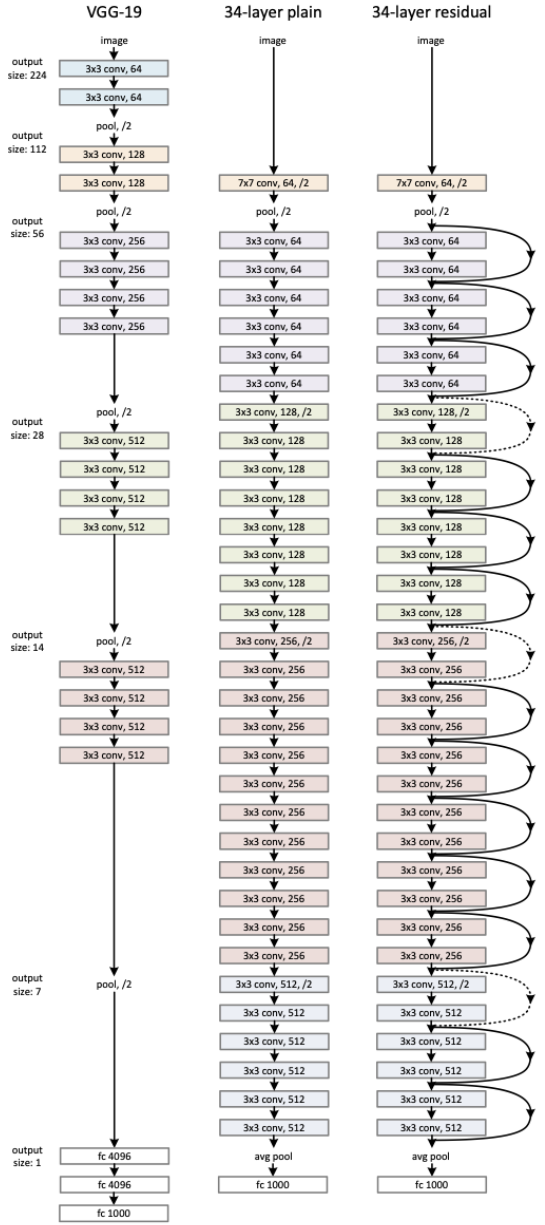
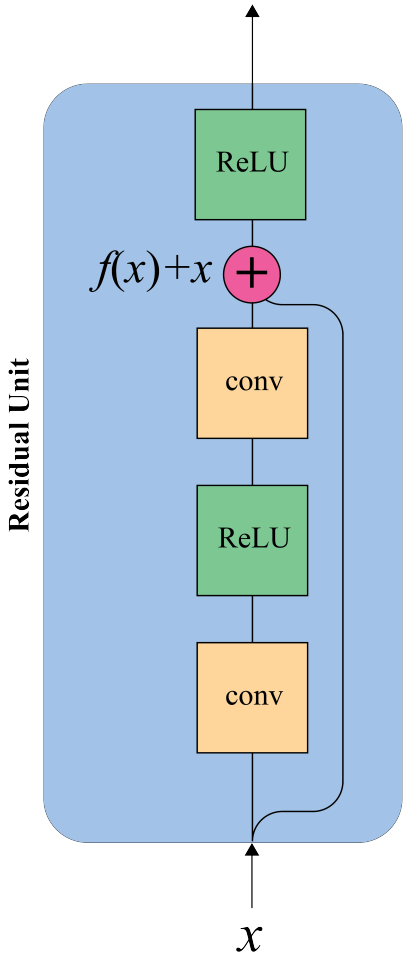


Figure on the right
Left: the VGG-19 model (19.6 billion FLOPs) as a reference. Middle: a plain network with 34 parameter layers (3.6 billion FLOPs). Right: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. Figure from He et al. 2016.

Residual unit

- Pass through connections adds the input of a layer to its output
- Deeper models are harder to train
- Learn residual function rather than direct mapping



Loss landscape with residual units

Notice the loss landscape with and without residual connections

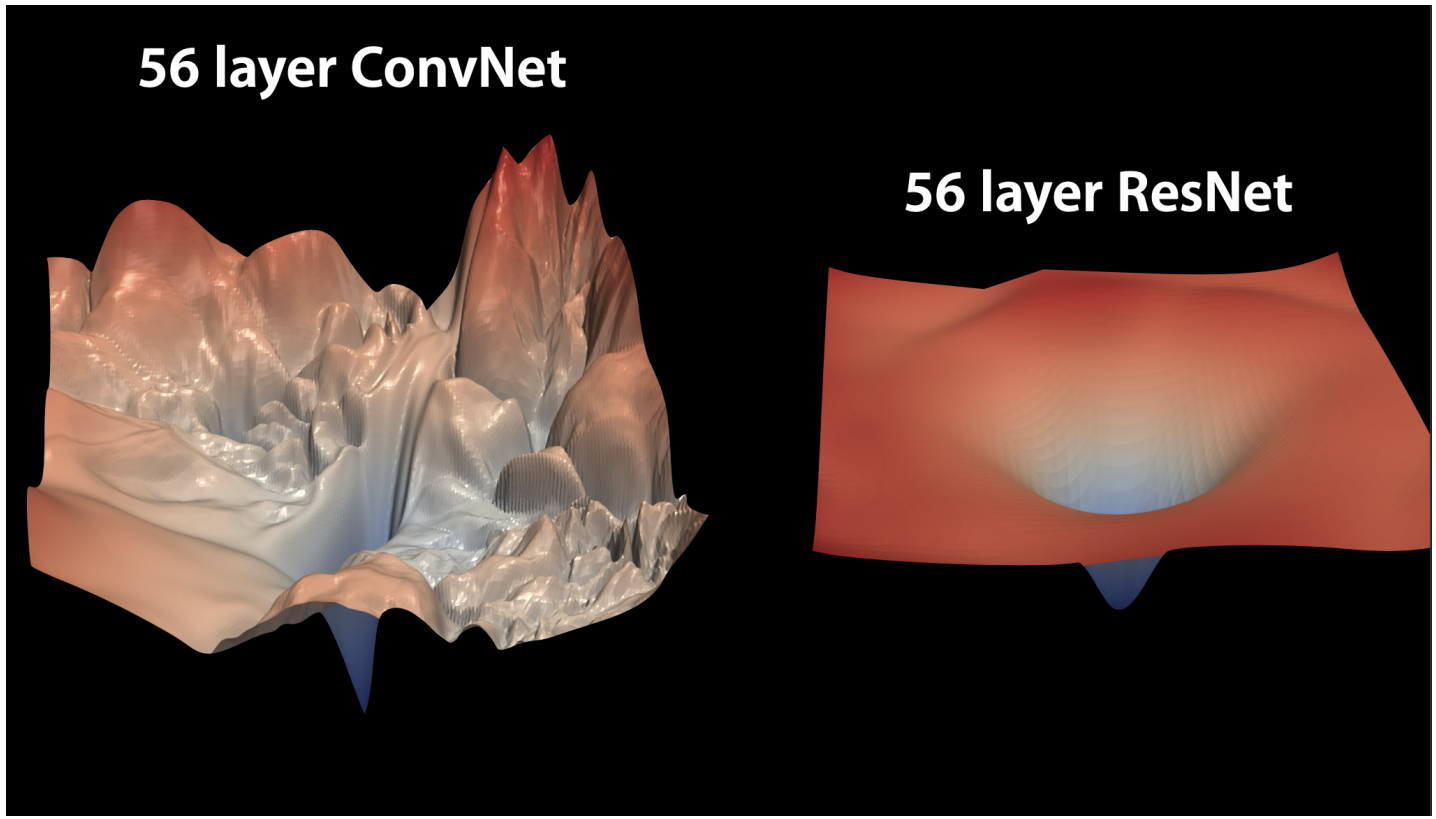


Figure taken from K. Derpanis notes on deep learning.

Densenet

- Huang et al. 2017 > Densely Connected Convolutional Networks

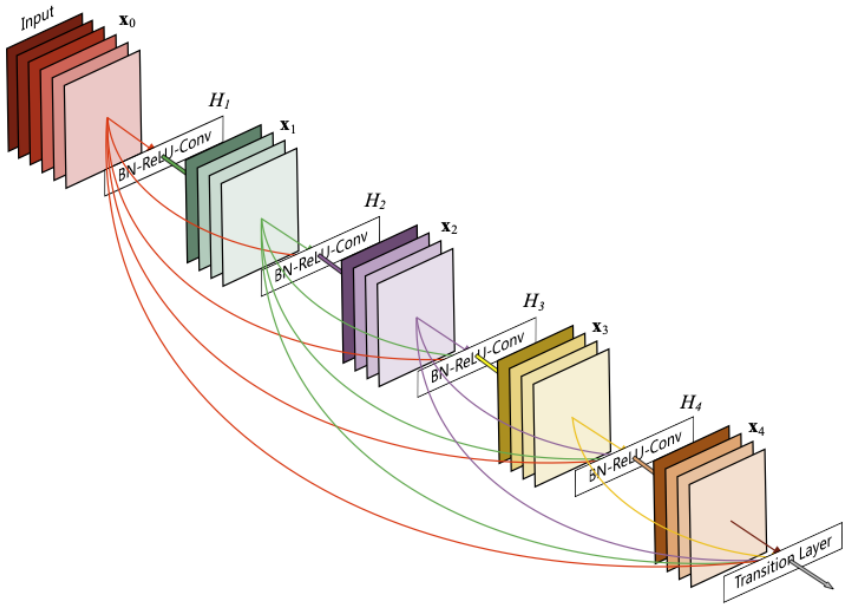


Figure from Huang et al. 2017.

Densenet

- Feature-maps learned by any of the layers can be accessed by all subsequent layers.
  - Encourages feature reuse throughout the network
  - Leads to more compact models
  - Supports diversified depth
- Improved training
  - Individual layers get additional supervision from loss function through shorter (more direct) connections
    - Similar to DSN (Lee et al. 2015) that attach classifiers to each hidden layer forcing intermediate layers to learn discriminative features
  - Scale to hundreds of layers without any optimization difficulties

Dense blocks and transition layers

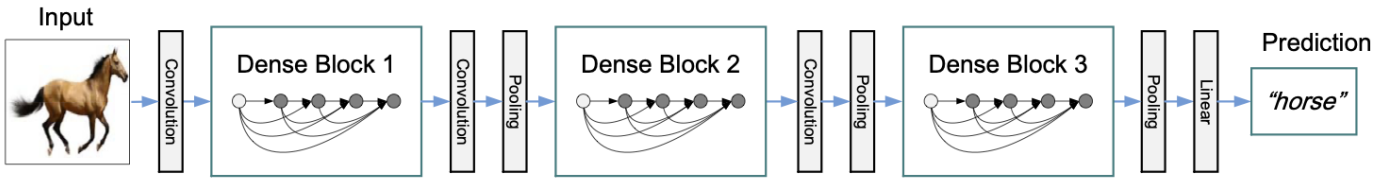


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

Figure from Huang et al. 2017

Densenet vs. Resnet

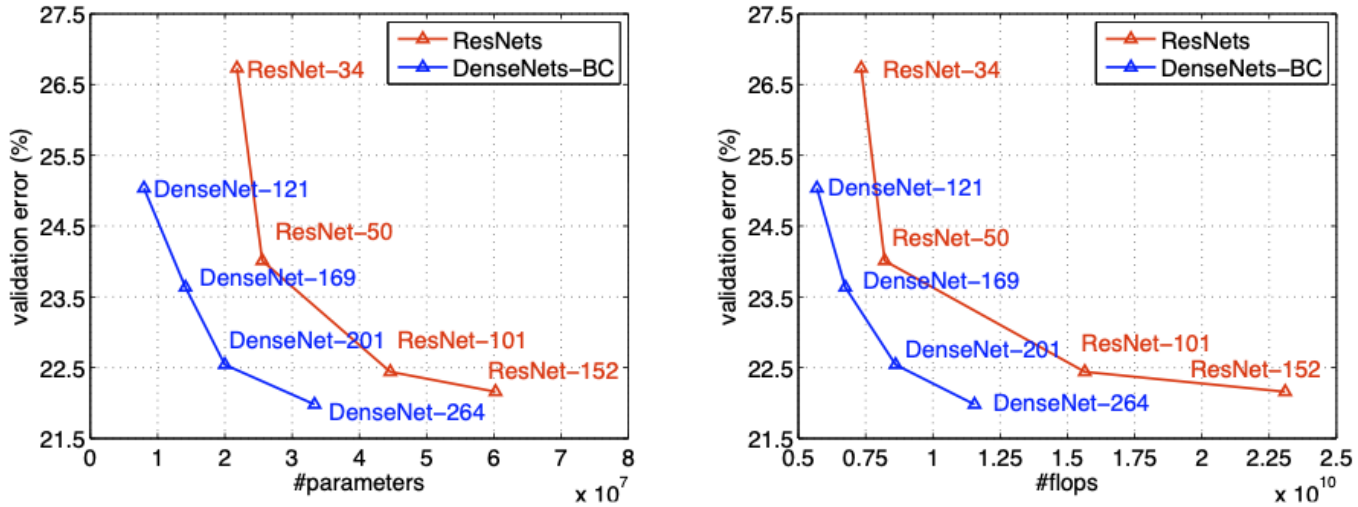


Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

Figure from Huang et al. 2017

Squeeze-and-Excitation Networks

- Hu et al. 2018
  Squeeze-and-Excitation Networks

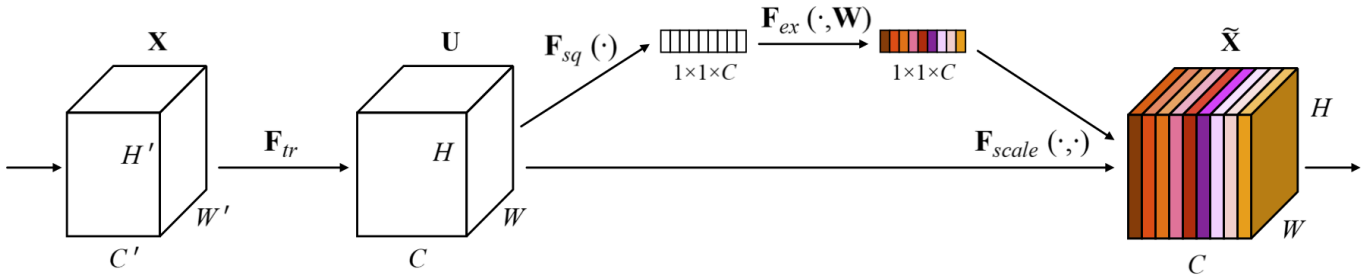
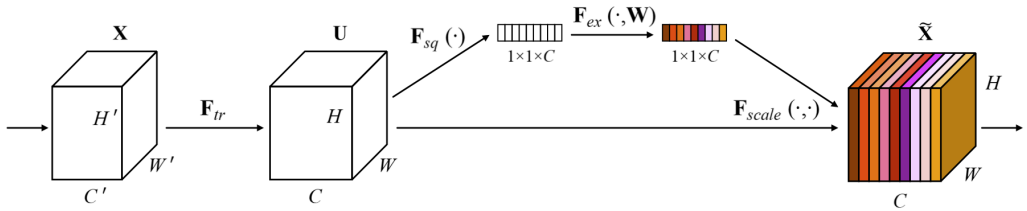


Figure from Hu et al. 2018

SE Block

- Squeeze operator
  - Allows global information to be used when computing channel-wise weights
- Excitation operator
  - Distribution across different classes is similar in early layers, suggesting that feature channels are "equally important" for different classes in early layers
  - Distribution becomes class-specific in deeper layers
- SE blocks may be used for model prunning and network compression



Squeeze-and-Excitation block (simplified).

SE Performance

TABLE 4
Classification error (%) on CIFAR-10.

	original	SENet
ResNet-110 [14]	6.37	5.21
ResNet-164 [14]	5.46	4.39
WRN-16-8 [67]	4.27	3.88
Shake-Shake 26 2x96d [68] + Cutout [69]	2.56	2.12

TABLE 5
Classification error (%) on CIFAR-100.

	original	SENet
ResNet-110 [14]	26.88	23.85
ResNet-164 [14]	24.33	21.31
WRN-16-8 [67]	20.43	19.14
Shake-Even 29 2x4x64d [68] + Cutout [69]	15.85	15.41

TABLE 6
Single-crop error rates (%) on Places365 validation set.

	top-1 err.	top-5 err.
Places-365-CNN [72]	41.07	11.48
ResNet-152 (ours)	41.15	11.61
SE-ResNet-152	**40.37**	**11.01**

TABLE 7
Faster R-CNN object detection results (%) on COCO *minival* set.

	AP@IoU=0.5	AP
ResNet-50	57.9	38.0
SE-ResNet-50	61.0	40.4
ResNet-101	60.1	39.9
SE-ResNet-101	62.7	41.9

Taken from Hu et al. 2018

Spatial attention

- SE computes channel weights; however, we can easily extend this idea to compute spatial weights to model some notion of spatial attention
  - The model will pay more attention to f

Other notable examples

- Larsson et al., 2016

  FractalNet: Ultra-Deep Neural Networks without Residuals

- Iandola et al., 2016

  SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and <0.5MB Model Size

- Howard et al., 2017

  MobileNet: Efficient Convolutional Neural Networks for Mobile Vision Applications

Other notable examples

- Xie et al., 2017

  Aggregated Residual Transformation for Deep Neural Networks

- Han et al., 2016

  Deep Pyramidal Residual Networks

- Chollet, 2017

  Xception: Deep Learning with Depthwise Separable Convolutions

Transformers (2017)

- Vaswani et al., 2017

  Attention Is All You Need

- Transformers use attention-based computation.
- These models are popular in Natural Language Processing community.
- GPT3 language model also uses attention-based computation and it has roughly 175 billion parameters.

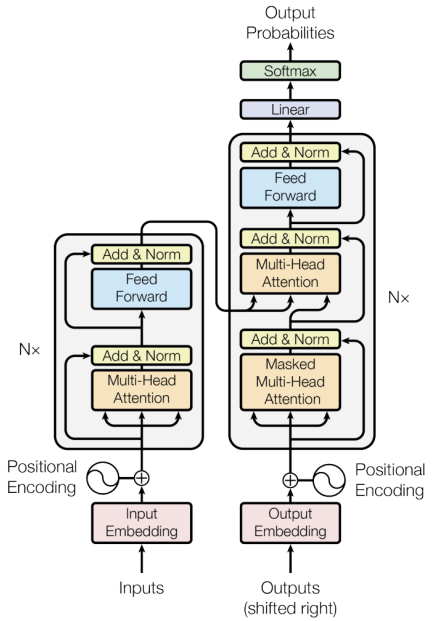


Figure 1: The Transformer - model architecture.

Attention

- Zhao et al., 2020

  Exploring Self-attention for Image Recognition

- Carion et al., 2020

  End-to-End Object Detection with Transformers

- Dosovitsky et al., 2020

  An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

- Zheng et al., 2020

  Rethinking Semantic Segmentation from a Sequence-to-Sequence Perspective with Transformers

Learning convolution kernels

- CNNs benefit from different kernel sizes at different layers
- Exploring all possible combinations of kernel sizes is infeasible in practice
- Romero et al. 2022
  FlexConv: Continuous Kernel Convolutions with Differentiable Kernel Sizes
- Riad et al. 2022
  Learning Strides in Convolutional Neural Networks

MLPs

- Tolstikhin et al. 2021
  MLP-Mixer: An all-MLP Architecture for Vision

- Melas-Kyriazi 2021
  Do You Even Need Attention? A Stack of Feed-Forward Layers Does Surprising Well on ImageNet

- Touvron et al. 2021
  ResMLP: Feedforward Networks for Image Classification with Data Efficient Training

A ConvNet for 2020

- Liu et al. 2020
  A ConvNet for 2020s

- Results on ImageNet

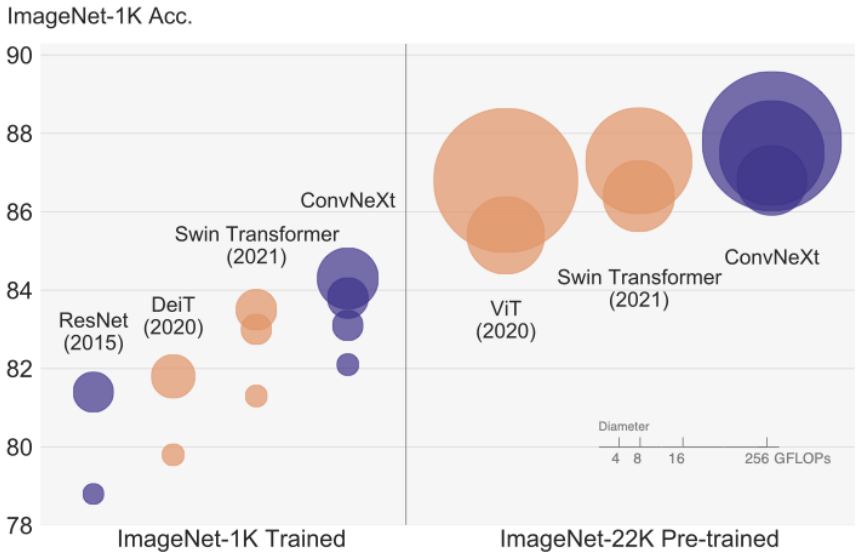


Figure 1. **ImageNet-1K classification** results for • ConvNets and ○ vision Transformers. Each bubble's area is proportional to FLOPs of a variant in a model family. ImageNet-1K/22K models here take $224^2/384^2$ images respectively. ResNet and ViT results were obtained with improved training procedures over the original papers. We demonstrate that a standard ConvNet model can achieve the same level of scalability as hierarchical vision Transformers while being much simpler in design.

Figure from Lie et al. 2020

Modernizing a ConvNet towards Swin (Hierarchical Vision Transformer)

Figure from Lie et al. 2020

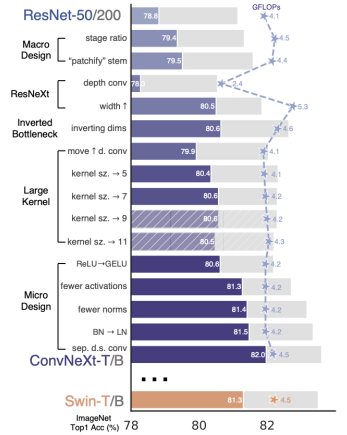


Figure 2. We modernize a standard ConvNet (ResNet) towards the design of a hierarchical vision Transformer (Swin), without introducing any attention-based modules. The foreground bars are model accuracies in the ResNet-50/Swin-T FLOP regime; results for the ResNet-200/Swin-B regime are shown with the gray bars. A hatched bar means the modification is not adopted. Detailed results for both regimes are in the appendix. Many Transformer architectural choices can be incorporated in a ConvNet, and they lead to increasingly better performance. In the end, our pure ConvNet model, named ConvNeXt, can outperform the Swin Transformer.

ConvNeXt: Key ideas

- Change stem to Patchify

  Replace the ResNet-style stem cell with a patchify layer implemented using a 4×4, stride 4 convolutional layer. The accuracy has changed from 79.4% to 79.5%.

  The stem cell in standard ResNet contains a 7×7 convolution layer with stride 2, followed by a max pool, which results in a $4\times$ downsampling of the input images.

ConvNeXt: Key ideas

- ResNeXtify
  - Grouped convolutions idea from Xie et al. 2016
    - Depthwise convolution where the number of groups equal to the number of channels. Similar to MobileNet and Xception.
    - Only mixes information in the spatial domain.

The combination of depthwise conv and 1×1 *convs leads to a separation of spatial and channel mixing, a property shared by vision Transformers, where each operation either mixes information across spatial or channel dimension, but not both.*

ConvNeXt: Key ideas

- Inverted bottleneck

  One important design in every Transformer block is that it creates an inverted bottleneck, i.e., the hidden dimension of the MLP block is four times wider than the input dimension.

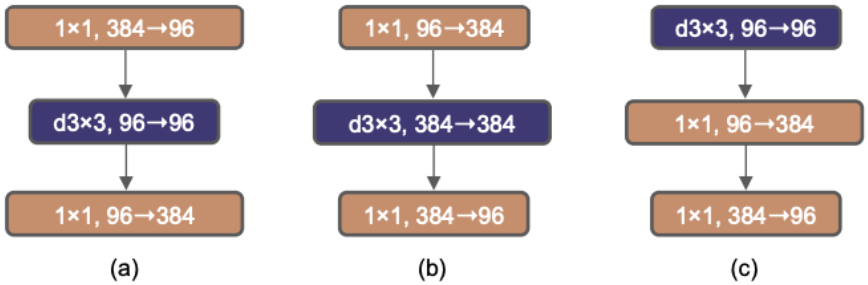


Figure 3. **Block modifications and resulted specifications. (a)** is a ResNeXt block; in **(b)** we create an inverted bottleneck block and in **(c)** the position of the spatial depthwise conv layer is moved up.

Figure from Lie et al. 2020

ConvNeXt: Key ideas

- Large kernel sizes

  One of the most distinguishing aspects of vision Transformers is their non-local self-attention, which enables each layer to have a global receptive field.

  To explore large kernels, one prerequisite is to move up the position of the depthwise conv layer.

ConvNeXt: Key ideas

- Replacing ReLU with GELU
  - Gaussian Error Linear Unit (Hendrycks and Gimpel, 2016)
- Use fewer activation functions
- Use fewer normalization layers
  - Replace batch normalization with layer normalization

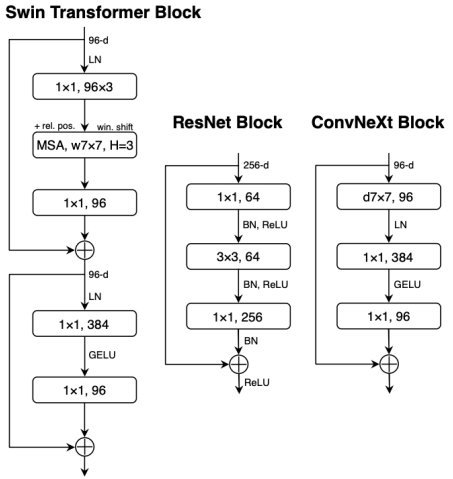


Figure 4. **Block designs** for a ResNet, a Swin Transformer, and a ConvNeXt. Swin Transformer's block is more sophisticated due to the presence of multiple specialized modules and two residual connections. For simplicity, we note the linear layers in Transformer MLP blocks also as "1×1 convs" since they are equivalent.

Figure from Lie et al. 2020

Normalization techniques

Wu and He, 2018

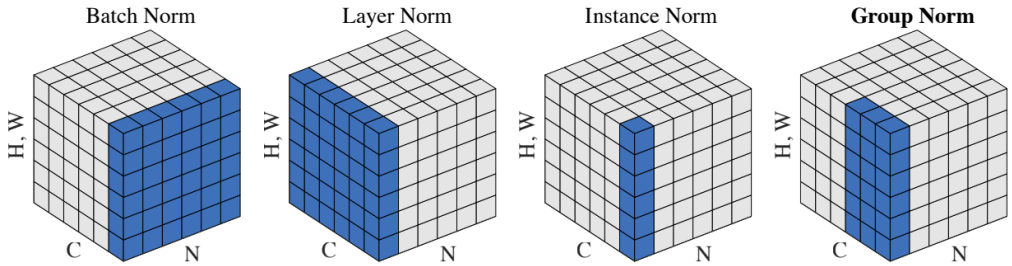


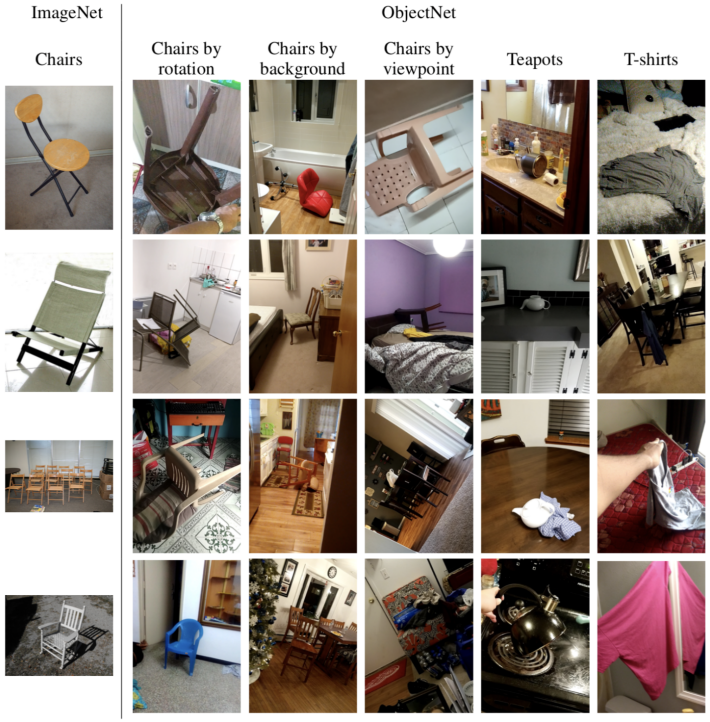
Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

Figure from Wu and He 2018

Is object detection solved?

- Barbu et al. 2019

  ObjectNet: A Large-Scale Bias-Controlled Dataset for Pushing the Limits of Object Recognition Models‘



- Performance on ObjectNet benchmark
  - 40 to 45% drop in performance

Afterward

- Adapted from Jeff Hawkins, Founder of Palm Computing.

  The key to object recognition is representation.

- Convolutional neural networks are particularly well-suited for computer vision tasks
- Convolutional layers "mimic" processing in visual cortex
- Exploits spatial relationship between neighbouring pixels
- Learns powerful representations that reduce the semantic gap

Practical matters: where to go from here?

- Deep learning is as much about engineering as it is about science
- Learn one of deep learning frameworks
  - Become an efficient coder
- Don't be afraid to use high-level deep learning tools to quickly prototype baselines (e.g., huggingface)
  - Deep learning projects share common features
    - Data loaders
    - Measuring performance, say accuracy, precision, etc.

Copyright and License

©Faisal Z. Qureshi

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.