

Deep Learning

Lecture 7 – Convolutional Neural Networks

Prof. Dr.-Ing. Andreas Geiger

Autonomous Vision Group
University of Tübingen / MPI-IS



e l l i s
European Laboratory for Learning and Intelligent Systems

Agenda

7.1 Convolution Layers

7.2 Downsampling

7.3 Upsampling

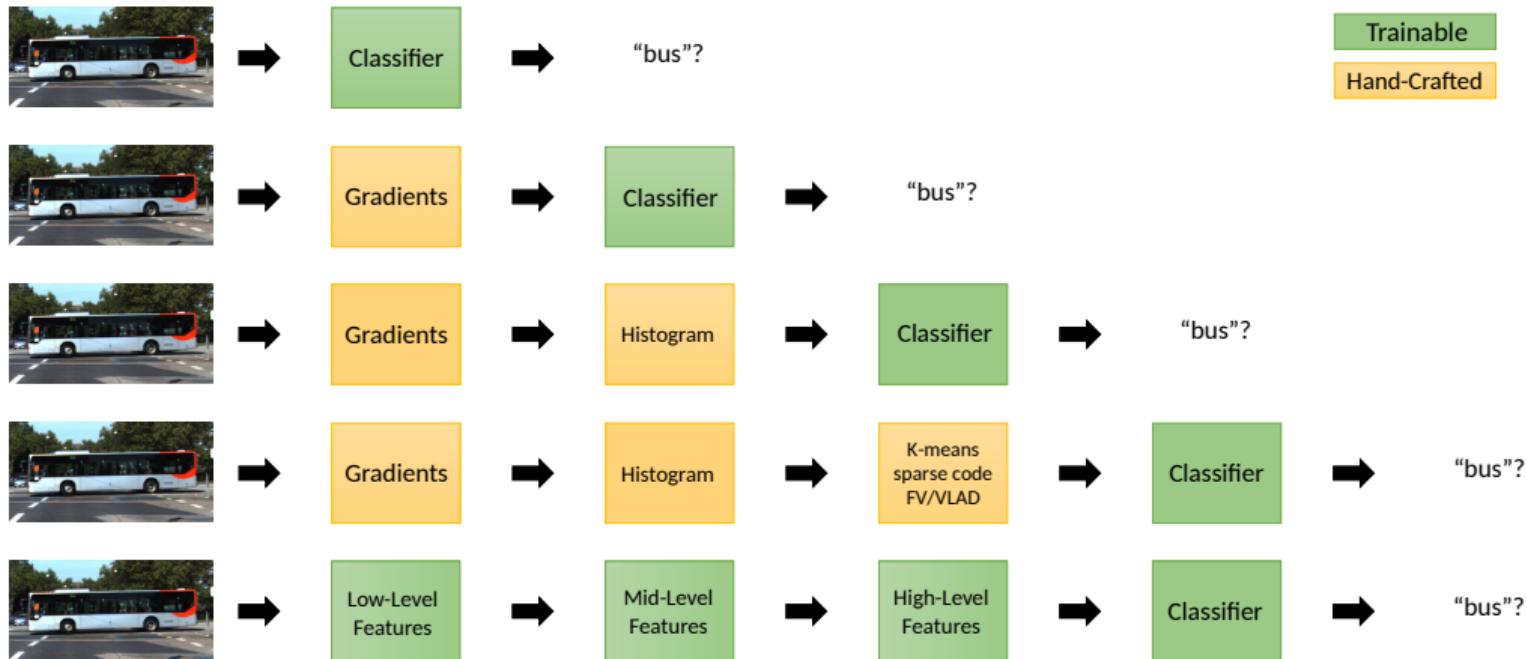
7.4 Architectures

7.5 Visualization

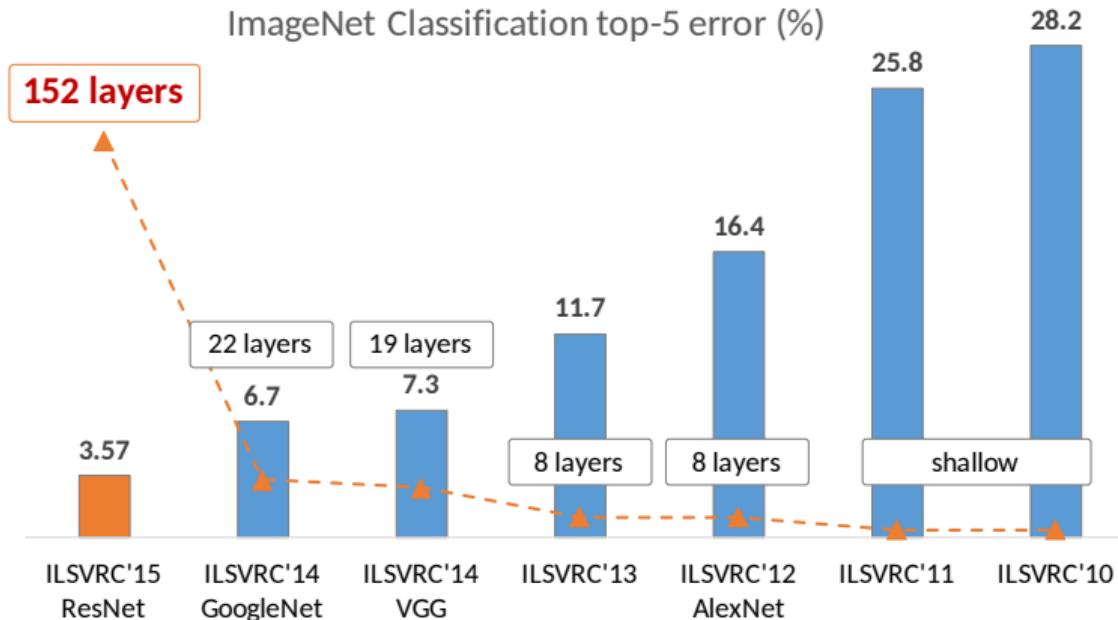
7.1

Convolution Layers

Representation Learning

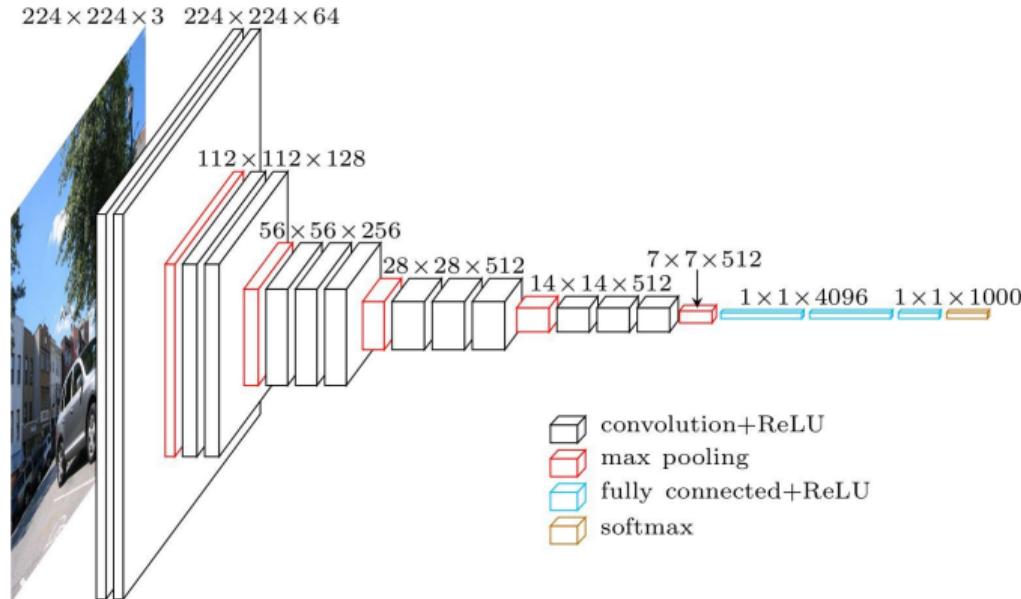


ImageNet Large Scale Visual Recognition Challenge



- ▶ Classification into 1000 object categories. Current state-of-the-art: 1.3 %

VGG Network



3 Types of Layers:

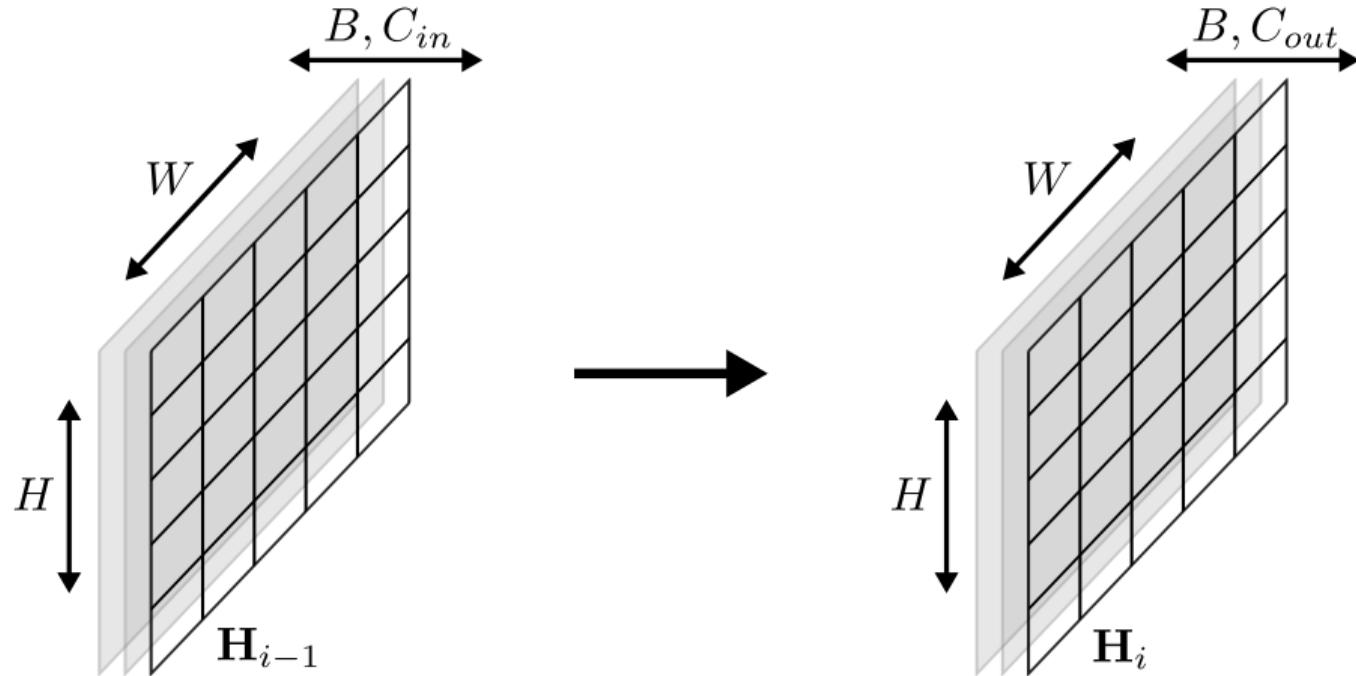
- ▶ Convolution layers, pooling layers and fully connected layers

Notation

We will use the following **notation**:

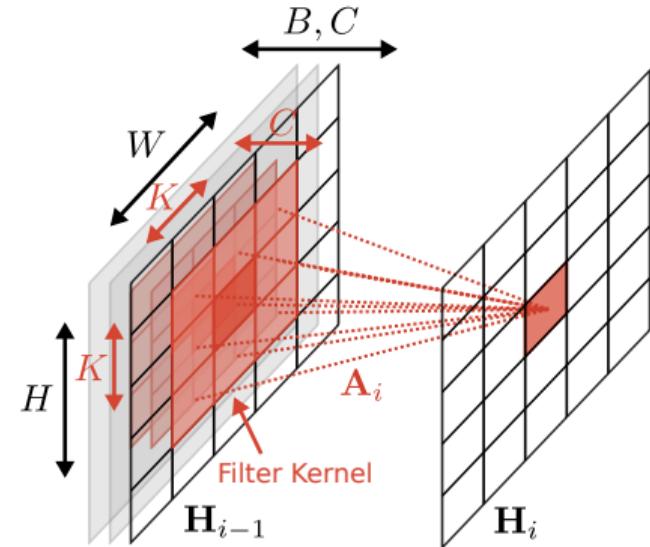
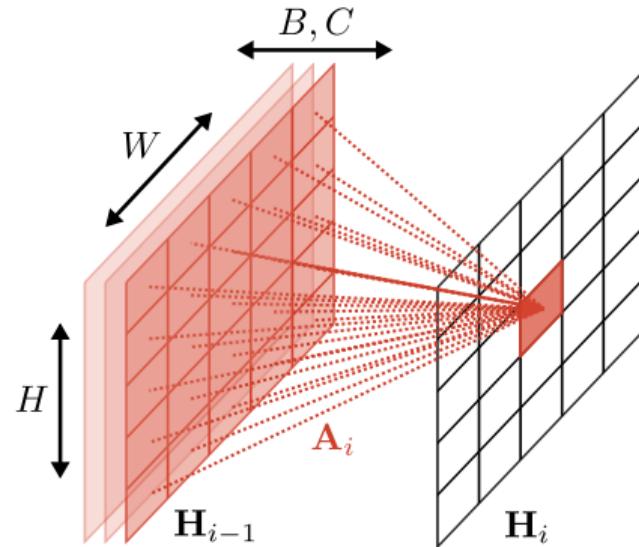
- ▶ B : **Batch size**
- ▶ W : **Width** of image or feature map
- ▶ H : **Height** of image or feature map
- ▶ C : Number of feature **channels** (=neurons per pixel)
 - ▶ C_{in} : Number of feature channels in current layer
 - ▶ C_{out} : Number of feature channels in next layer
 - ▶ $C_{in} = 1$ for first layer if input is a grayscale image
 - ▶ $C_{in} = 3$ for first layer if input is a color image
- ▶ $K \times K$: Convolutional filter **kernel size**

Network Layers with Spatial Dimension



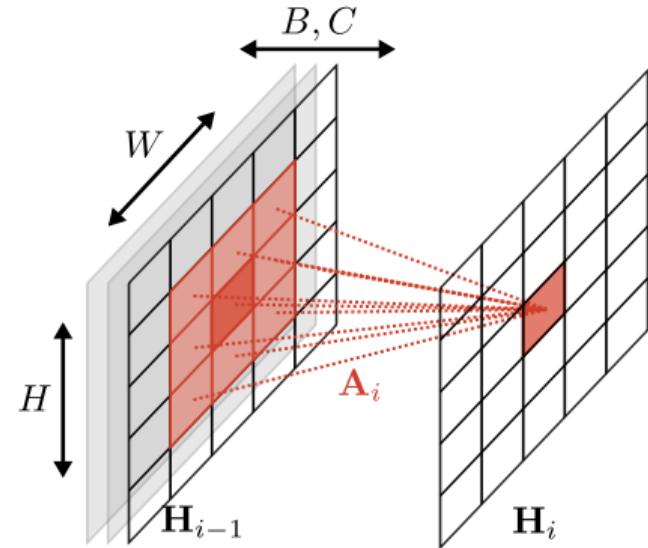
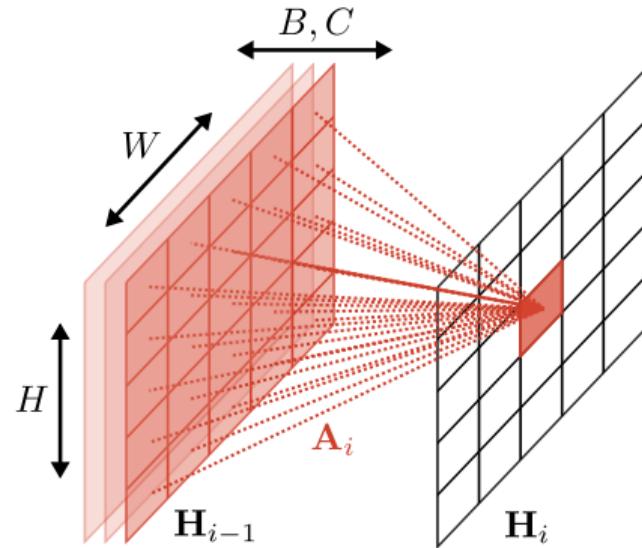
- ▶ In contrast to MLPs, ConvNet layers have a **spatial extent**

Fully Connected vs. Convolutional Layers



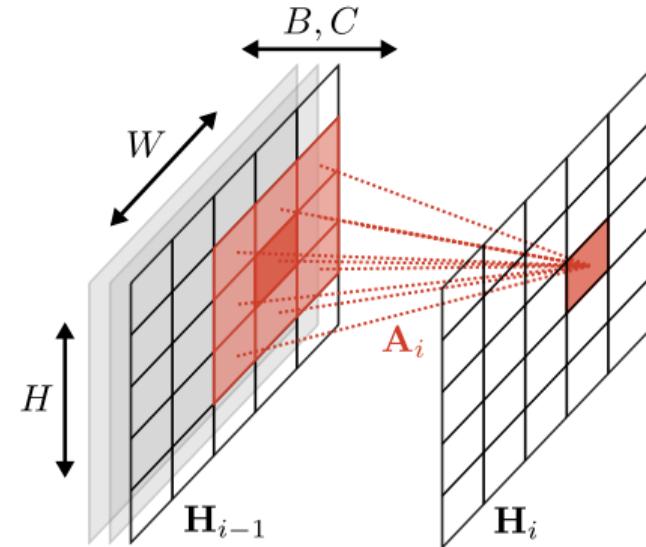
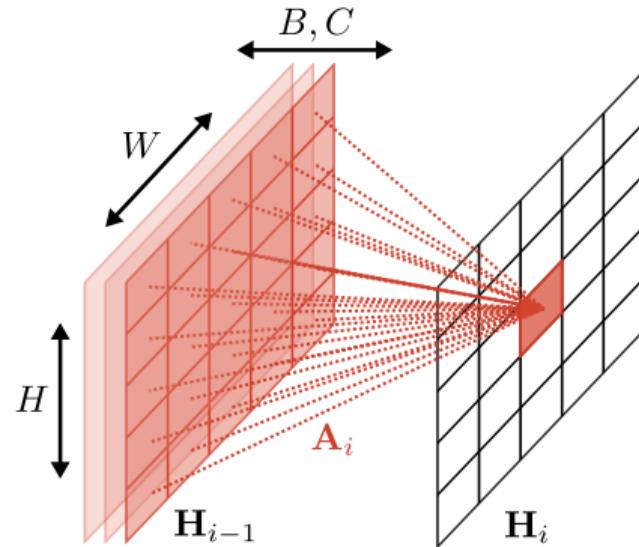
- **Fully connected layer:** #Weights = $W \times H \times C_{out} \times (W \times H \times C_{in} + 1)$
- **Convolutional layer:** #Weights = $C_{out} \times (K \times K \times C_{in} + 1)$ ("weight sharing")
- With C_{in} input and C_{out} output channels, layer size $W \times H$ and kernel size $K \times K$
- Convolution kernel is three-dimensional. Only 1 output channel shown for clarity.

Fully Connected vs. Convolutional Layers



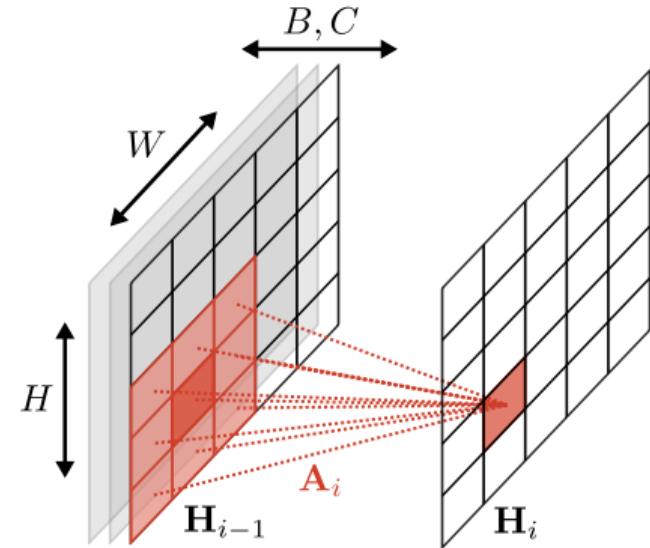
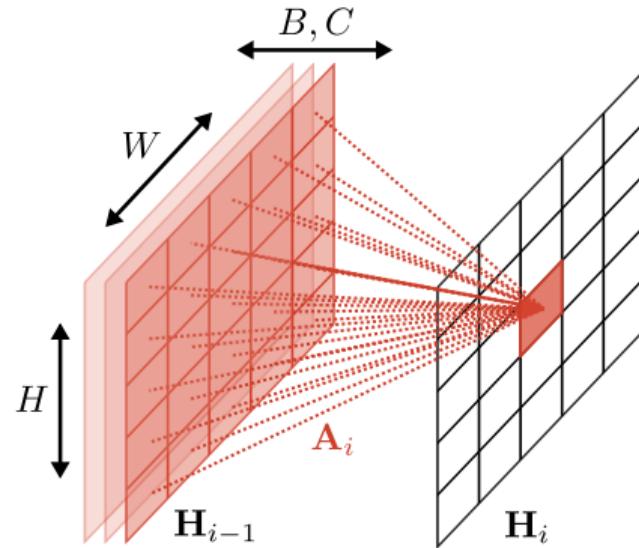
- **Fully connected layer:** #Weights = $W \times H \times C_{out} \times (W \times H \times C_{in} + 1)$
- **Convolutional layer:** #Weights = $C_{out} \times (K \times K \times C_{in} + 1)$ ("weight sharing")
- With C_{in} input and C_{out} output channels, layer size $W \times H$ and kernel size $K \times K$
- Convolution kernel is three-dimensional. Only 1 output channel shown for clarity.

Fully Connected vs. Convolutional Layers



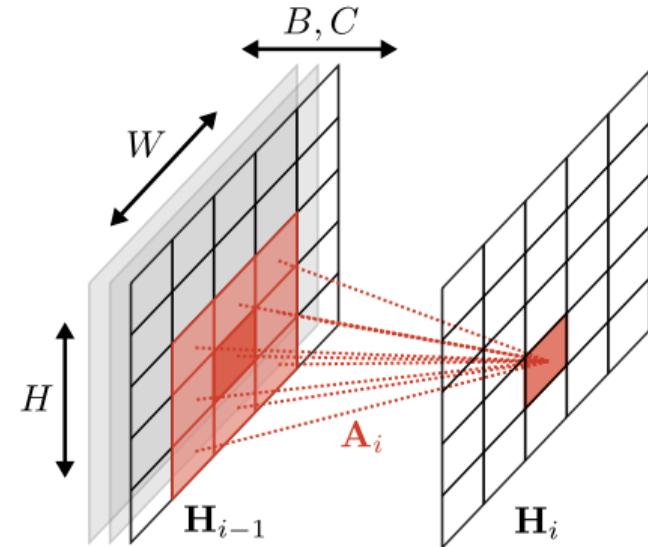
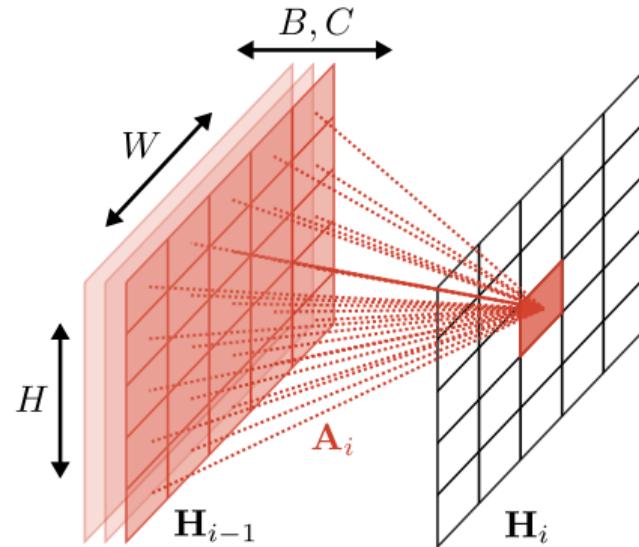
- **Fully connected layer:** #Weights = $W \times H \times C_{out} \times (W \times H \times C_{in} + 1)$
- **Convolutional layer:** #Weights = $C_{out} \times (K \times K \times C_{in} + 1)$ ("weight sharing")
- With C_{in} input and C_{out} output channels, layer size $W \times H$ and kernel size $K \times K$
- Convolution kernel is three-dimensional. Only 1 output channel shown for clarity.

Fully Connected vs. Convolutional Layers



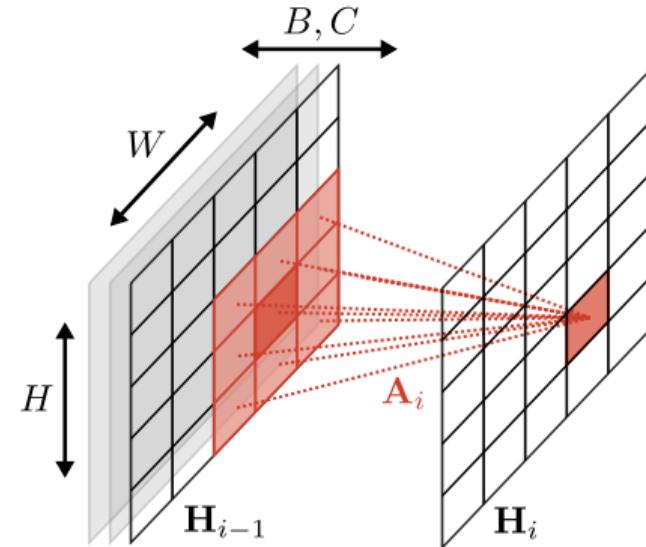
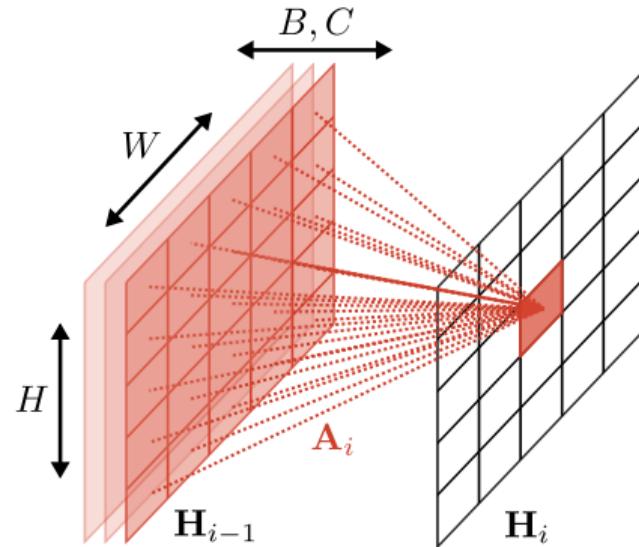
- **Fully connected layer:** #Weights = $W \times H \times C_{out} \times (W \times H \times C_{in} + 1)$
- **Convolutional layer:** #Weights = $C_{out} \times (K \times K \times C_{in} + 1)$ ("weight sharing")
- With C_{in} input and C_{out} output channels, layer size $W \times H$ and kernel size $K \times K$
- Convolution kernel is three-dimensional. Only 1 output channel shown for clarity.

Fully Connected vs. Convolutional Layers



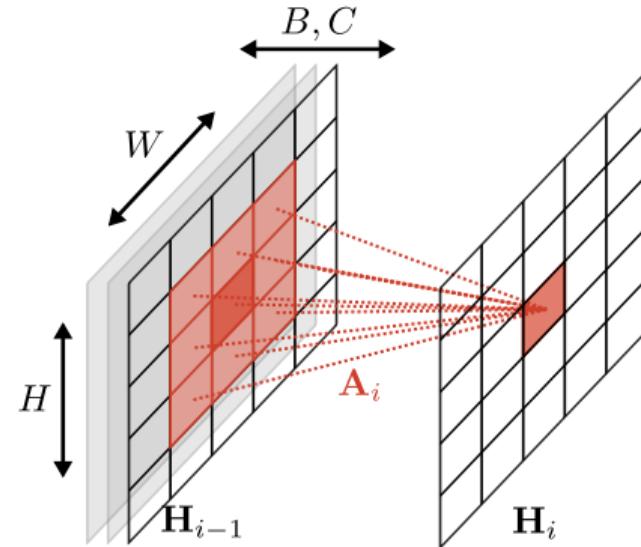
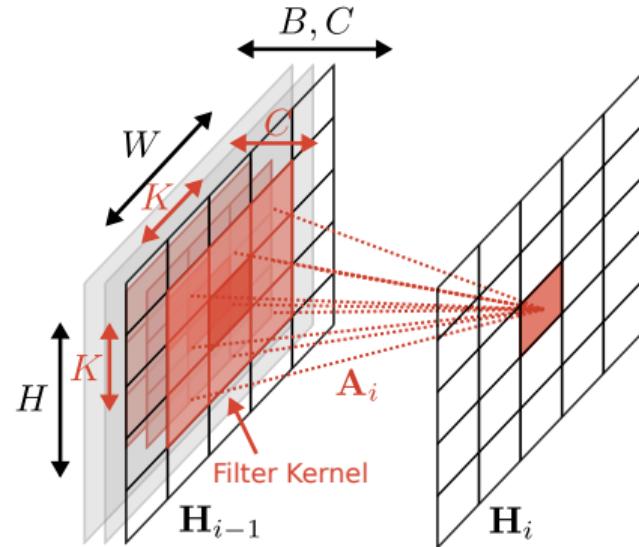
- **Fully connected layer:** #Weights = $W \times H \times C_{out} \times (W \times H \times C_{in} + 1)$
- **Convolutional layer:** #Weights = $C_{out} \times (K \times K \times C_{in} + 1)$ ("weight sharing")
- With C_{in} input and C_{out} output channels, layer size $W \times H$ and kernel size $K \times K$
- Convolution kernel is three-dimensional. Only 1 output channel shown for clarity.

Fully Connected vs. Convolutional Layers



- **Fully connected layer:** #Weights = $W \times H \times C_{out} \times (W \times H \times C_{in} + 1)$
- **Convolutional layer:** #Weights = $C_{out} \times (K \times K \times C_{in} + 1)$ ("weight sharing")
- With C_{in} input and C_{out} output channels, layer size $W \times H$ and kernel size $K \times K$
- Convolution kernel is three-dimensional. Only 1 output channel shown for clarity.

Remark on Illustration Conventions



- ▶ Convolution kernels are 3D, but in illustrations we drop the C dimension for clarity.
- ▶ Usually, multiple convolution kernels are convolved with the input, each producing a different output channel. In illustrations, we show a single convolution for clarity.
- ▶ We will use Einstein notation to make all operations formally clear and concise.

Einstein Notation

Einstein Notation

We use a modified form of **Einstein Notation**

where **capital letters** are used to denote **slices of a tensor**:

- ▶ $A[i, j]$ denotes **one element** of the matrix A
- ▶ $A[i, J]$ denotes the **i'th row** of matrix A
- ▶ $A[I, j]$ denotes the **j'th column** of matrix A
- ▶ $A[I, J]$ denotes the **full matrix** A

- ▶ $H[i, j, k]$ denotes **one element** of the tensor H
- ▶ ...

Einstein Notation

Repeated capital letters in a product denote **summation** over those letters:

$$\begin{aligned}\mathbf{y} = \mathbf{Ax} &\equiv y[i] = \sum_j A[i, j]x[j] \\ &\equiv y[i] = A[i, J]x[J]\end{aligned}$$

$$\begin{aligned}\mathbf{y} = \mathbf{x}^\top \mathbf{A} &\equiv y[j] = \sum_i A[i, j]x[i] \\ &\equiv y[j] = A[I, j]x[I]\end{aligned}$$

Multiple capital letters denote summation over the **combination** of indices:

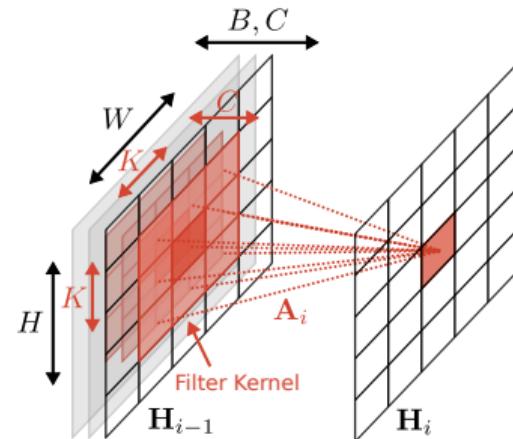
$$c = \sum_{i,j} A[i, j]B[i, j] = A[I, J]B[I, J]$$

Einstein Notation

- ▶ The indeces of tensors typically have **types** such as “batch index”, “x coordinate”, “y coordinate” or “channel index”
- ▶ Writing a tensor as $A[B, X, Y, C]$ with batch B , coordinates X, Y and feature channels C makes the type of the tensor elements and the order of the indeces **explicit**. It therefore also disambiguates \mathbf{A} from \mathbf{A}^\top .
- ▶ We will therefore use Einstein notation to describe the (hidden) layers and filter weights in a CNN
- ▶ Einstein notation can also be used in **NumPy** (`numpy.einsum`)

Convolution Layer

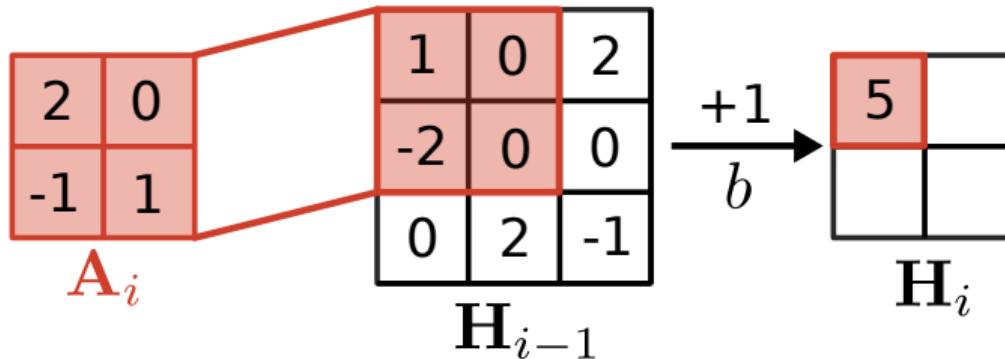
Convolution Layer



$$\underbrace{H_i[b, x, y, c_{out}]}_{\text{Current Layer}} = g \left(\underbrace{A_i[\Delta X, \Delta Y, C_{in}, c_{out}]}_{\text{Weights}} \underbrace{H_{i-1}[b, x + \Delta X, y + \Delta Y, C_{in}]}_{\text{Prev. Layer}} + \underbrace{b_i[c_{out}]}_{\text{Bias}} \right)$$

- b : Batch index x, y : Spatial locations c_{in}, c_{out} : Feature channels g : Act. func.
- $H_i[b, x, y, d]$ denotes feature d at image position x, y in layer i of batch b
- A_i and b_i depend on i as every layer has its own parameters
- Einstein summation considers all combinations over a range of indices ($\Delta X, \Delta Y$)

Convolution Example

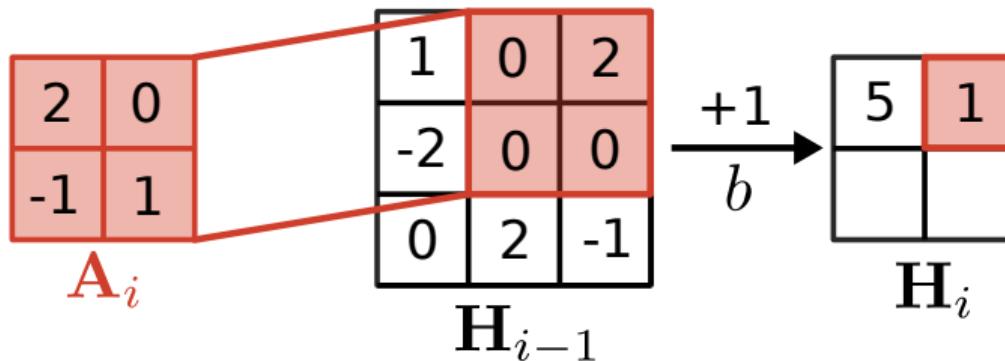


$$2 \cdot 1 + (-1) \cdot (-2) + 1 = 5$$

Remarks:

- Technically, ConvNets implement **correlation** (not convolution) operations
- Low-rank kernels can be implemented by efficient **separable convolutions**

Convolution Example

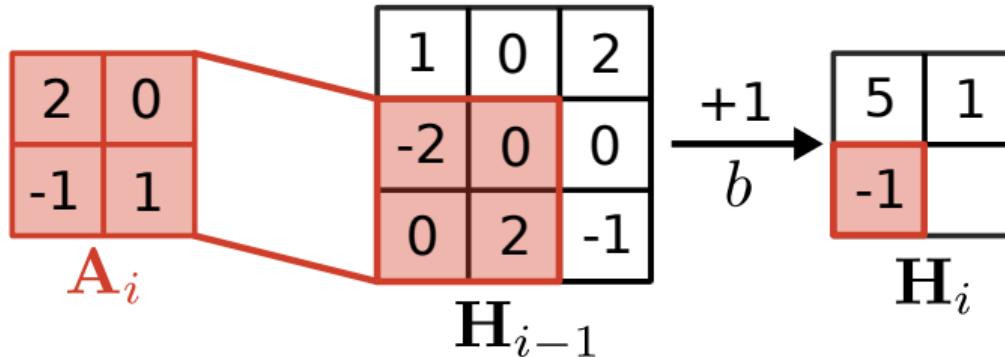


$$0 + 1 = 1$$

Remarks:

- ▶ Technically, ConvNets implement **correlation** (not convolution) operations
- ▶ Low-rank kernels can be implemented by efficient **separable convolutions**

Convolution Example

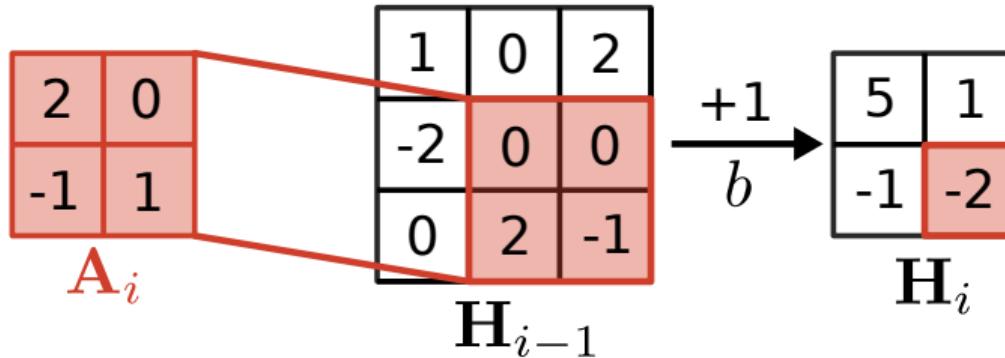


$$2 \cdot (-2) + 1 \cdot 2 + 1 = -1$$

Remarks:

- Technically, ConvNets implement **correlation** (not convolution) operations
- Low-rank kernels can be implemented by efficient **separable convolutions**

Convolution Example



$$(-1) \cdot 2 + 1 \cdot (-1) + 1 = -2$$

Remarks:

- Technically, ConvNets implement **correlation** (not convolution) operations
- Low-rank kernels can be implemented by efficient **separable convolutions**

Convolution Example

 $*$

0	1	0
1	4	1
0	1	0



- ▶ Common filters. In deep learning our goal is to learn the filter kernels.

Convolution Example



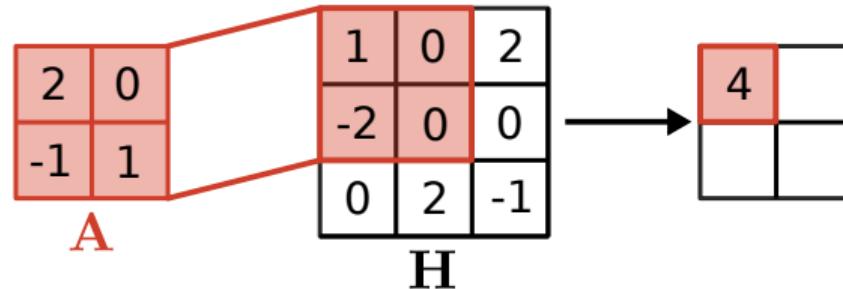
*

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



- ▶ Common filters. In deep learning our goal is to learn the filter kernels.

Convolution Operator

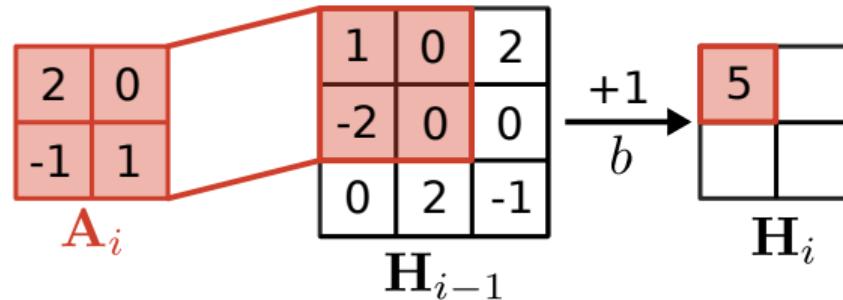


Definition of Convolution Operator:

$$[\mathbf{A} * \mathbf{H}](\mathbf{x}) = \sum_{\Delta \mathbf{x} \in \mathbb{Z}^2} \mathbf{A}(\Delta \mathbf{x}) \mathbf{H}(\mathbf{x} + \Delta \mathbf{x})$$

- ▶ Convolution of feature map \mathbf{H} with **filter kernel \mathbf{A}**
- ▶ Convolutions are **translation equivariant**

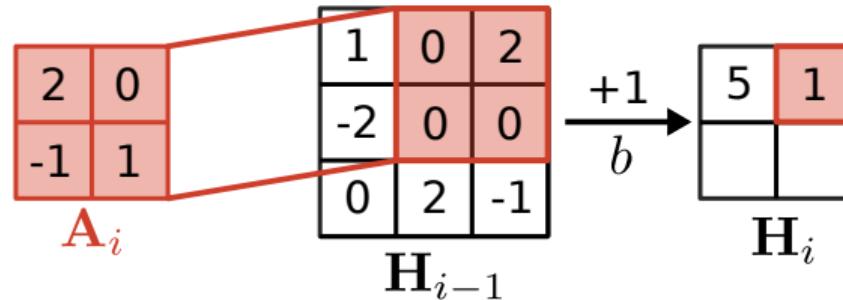
Convolution Example



Example (assuming 0-based indexing):

$$[\mathbf{A} * \mathbf{H}] \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \mathbf{A} \begin{pmatrix} 0 \\ 0 \end{pmatrix} \mathbf{H} \begin{pmatrix} 0+0 \\ 0+0 \end{pmatrix} + \mathbf{A} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \mathbf{H} \begin{pmatrix} 0+0 \\ 0+1 \end{pmatrix} \\ + \mathbf{A} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \mathbf{H} \begin{pmatrix} 0+1 \\ 0+0 \end{pmatrix} + \mathbf{A} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \mathbf{H} \begin{pmatrix} 0+1 \\ 0+1 \end{pmatrix}$$

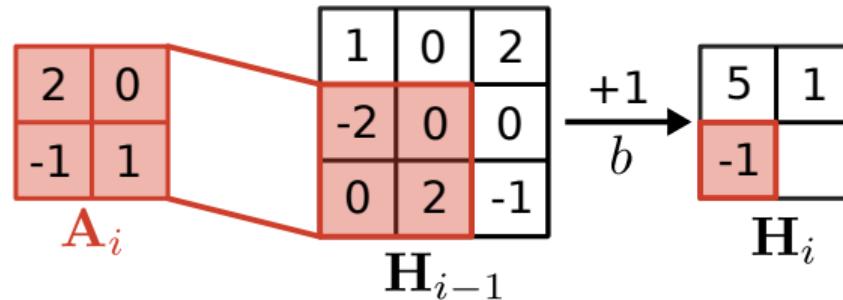
Convolution Example



Example (assuming 0-based indexing):

$$[\mathbf{A} * \mathbf{H}] \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \mathbf{A} \begin{pmatrix} 0 \\ 0 \end{pmatrix} \mathbf{H} \begin{pmatrix} 0+0 \\ 1+0 \end{pmatrix} + \mathbf{A} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \mathbf{H} \begin{pmatrix} 0+0 \\ 1+1 \end{pmatrix} \\ + \mathbf{A} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \mathbf{H} \begin{pmatrix} 0+1 \\ 1+0 \end{pmatrix} + \mathbf{A} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \mathbf{H} \begin{pmatrix} 0+1 \\ 1+1 \end{pmatrix}$$

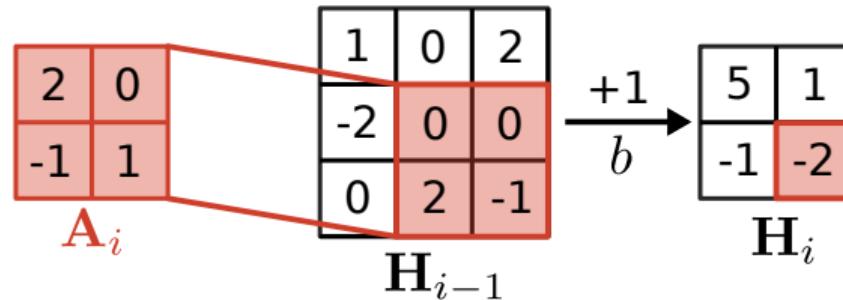
Convolution Example



Example (assuming 0-based indexing):

$$[\mathbf{A} * \mathbf{H}] \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \mathbf{A} \begin{pmatrix} 0 \\ 0 \end{pmatrix} \mathbf{H} \begin{pmatrix} 1+0 \\ 0+0 \end{pmatrix} + \mathbf{A} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \mathbf{H} \begin{pmatrix} 1+0 \\ 0+1 \end{pmatrix} \\ + \mathbf{A} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \mathbf{H} \begin{pmatrix} 1+1 \\ 0+0 \end{pmatrix} + \mathbf{A} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \mathbf{H} \begin{pmatrix} 1+1 \\ 0+1 \end{pmatrix}$$

Convolution Example



Example (assuming 0-based indexing):

$$[\mathbf{A} * \mathbf{H}] \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \mathbf{A} \begin{pmatrix} 0 \\ 0 \end{pmatrix} \mathbf{H} \begin{pmatrix} 1+0 \\ 1+0 \end{pmatrix} + \mathbf{A} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \mathbf{H} \begin{pmatrix} 1+0 \\ 1+1 \end{pmatrix} \\ + \mathbf{A} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \mathbf{H} \begin{pmatrix} 1+1 \\ 1+0 \end{pmatrix} + \mathbf{A} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \mathbf{H} \begin{pmatrix} 1+1 \\ 1+1 \end{pmatrix}$$

Equivariance

Invariance vs. Equivariance

Invariance:

$$f(\mathbf{H}) = f(\mathcal{T}_\theta[\mathbf{H}])$$

Equivariance:

$$\mathcal{T}_\theta[f](\mathbf{H}) = f(\mathcal{T}_\theta[\mathbf{H}])$$

- ▶ With feature map \mathbf{H} , operation $f(\cdot)$ and transformation \mathcal{T}_θ
- ▶ An operation is **invariant** if it returns the same result for any transformed input
- ▶ An operation is **equivariant** if its output transforms as its input for some specific transformation type (in the case of ConvNets: translation)

Translation Equivariance of Convolution

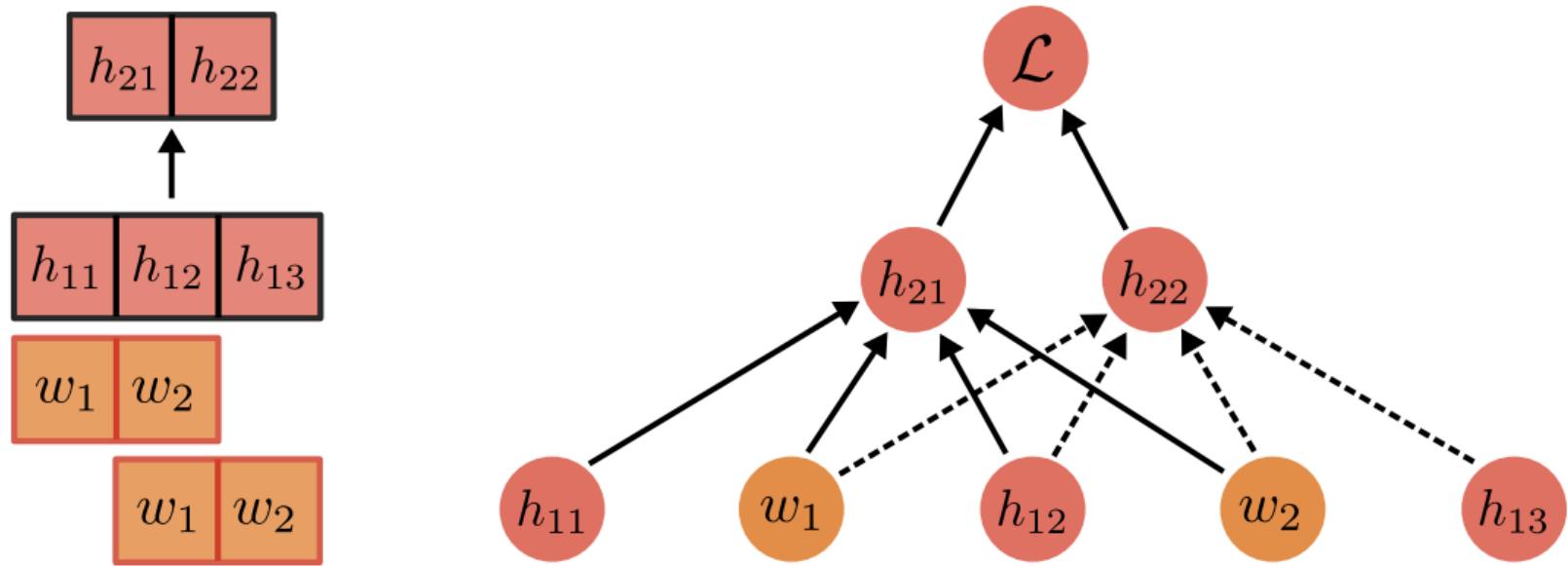
Proof:

$$\begin{aligned} [\mathbf{A} * \mathcal{T}_{\mathbf{t}}[\mathbf{H}]](\mathbf{x}) &= \sum_{\Delta \mathbf{x} \in \mathbb{Z}^2} \mathbf{A}(\Delta \mathbf{x}) \mathcal{T}_{\mathbf{t}}[\mathbf{H}](\mathbf{x} + \Delta \mathbf{x}) && \text{definition of convolution} \\ &= \sum_{\Delta \mathbf{x} \in \mathbb{Z}^2} \mathbf{A}(\Delta \mathbf{x}) \mathbf{H}(\mathbf{x} + \Delta \mathbf{x} - \mathbf{t}) && \text{expanding translation operator} \\ &= \sum_{\Delta \mathbf{x} \in \mathbb{Z}^2} \mathbf{A}(\Delta \mathbf{x}) \mathbf{H}((\mathbf{x} - \mathbf{t}) + \Delta \mathbf{x}) && \text{rearranging} \\ &= [\mathbf{A} * \mathbf{H}](\mathbf{x} - \mathbf{t}) = \mathcal{T}_{\mathbf{t}}[\mathbf{A} * \mathbf{H}](\mathbf{x}) && \text{definition of convolution} \end{aligned}$$

- ▶ Convolutions are **translation equivariant** but not translation invariant
- ▶ Shifting the input, results in a **shifted output** (exception: image boundaries)
- ▶ Interpretation as **inductive bias** built into ConvNet architecture

Implementation as Computation Graph

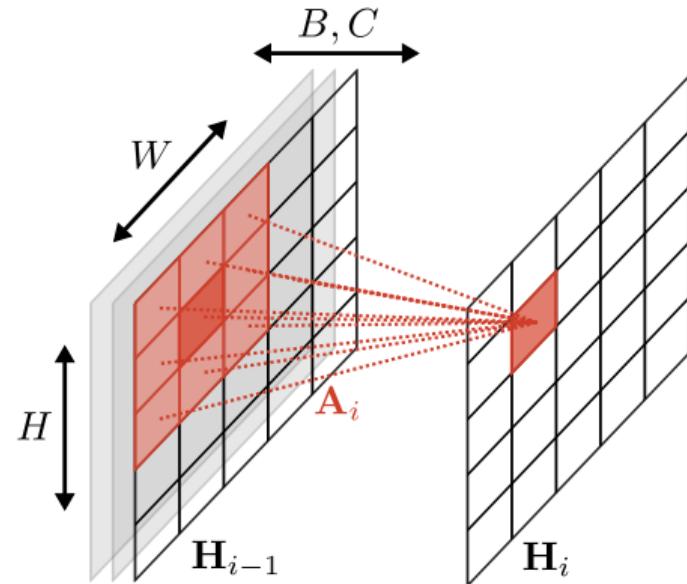
Computation Graph



- Gradients get **accumulated** (summed up) across locations due to **weight sharing**

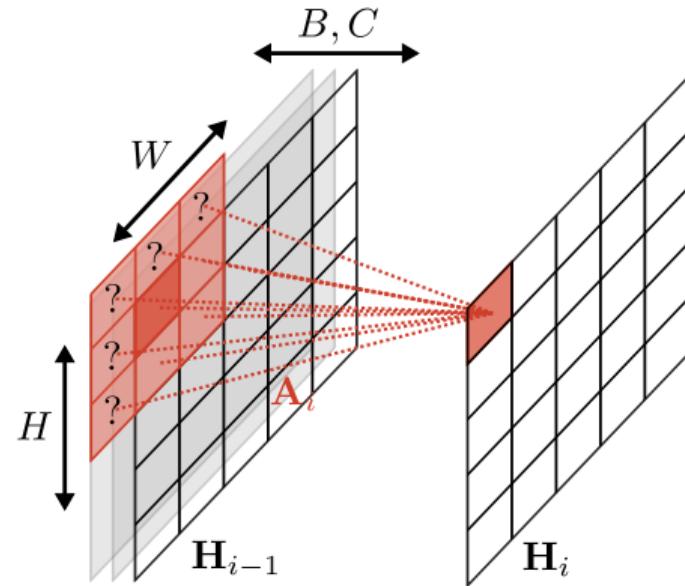
Padding

Padding



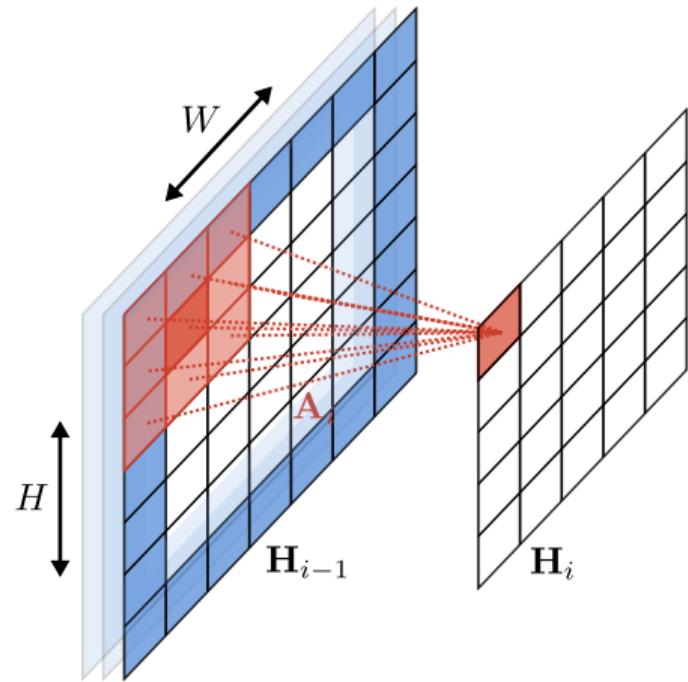
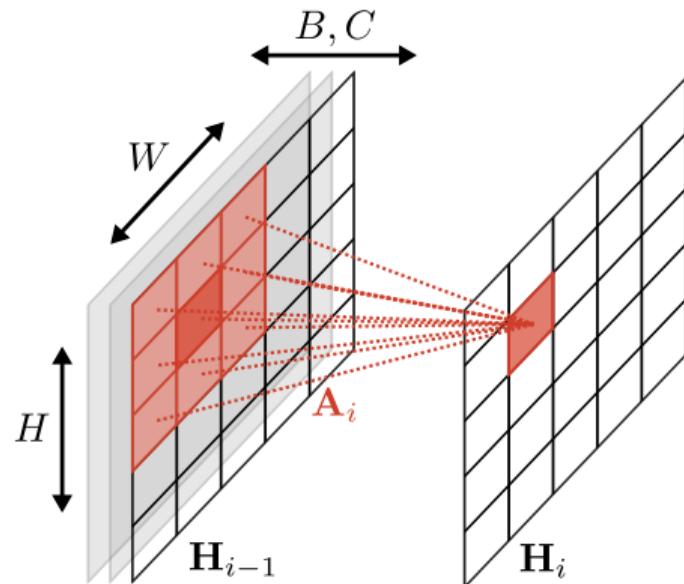
- ▶ Convolutions can only be executed if kernel lies entirely **within input domain**
- ▶ Often undesirable and **inconvenient** as it couples architecture and input size

Padding



- ▶ Convolutions can only be executed if kernel lies entirely **within input domain**
- ▶ Often undesirable and **inconvenient** as it couples architecture and input size

Padding



Idea of Padding:

- ▶ Add boundary of appropriate size with zeros (blue) around input tensor

Padding

- ▶ Let $H'[\cdot]$ denote a padded version of the tensor $H[\cdot]$
- ▶ In NumPy, this can be achieved as follows (with P the padding size):

```
H_pad = np.zeros(H+2P, W+2P)
```

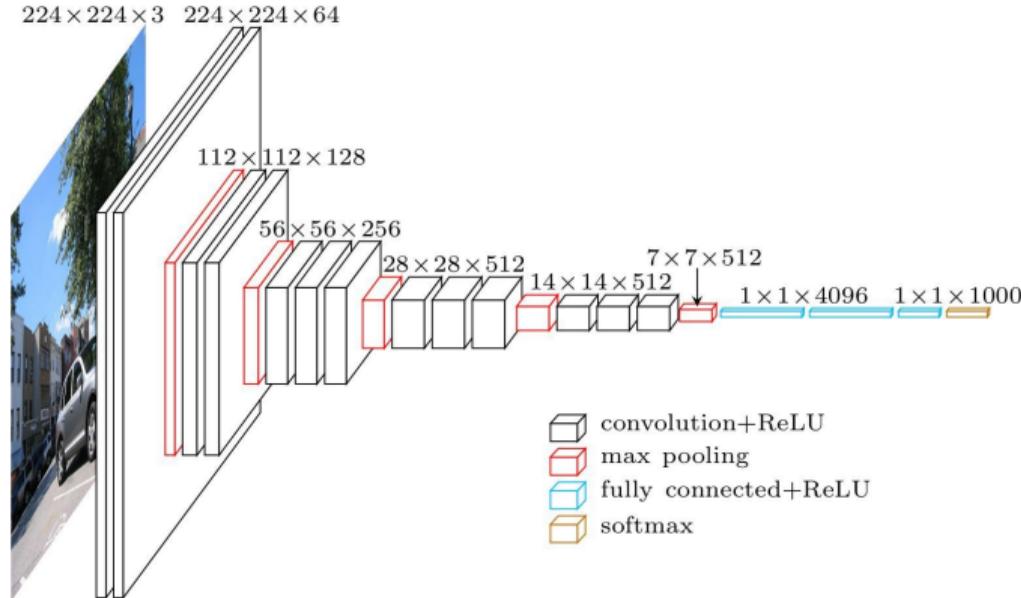
```
H_pad[P:H+P, P:W+P] = H
```

- ▶ The output of the **padded convolution** is given by:

$$\underbrace{H_i[b, x, y, c_{out}]}_{\text{Current Layer}} = g \left(\underbrace{A_i[\Delta X, \Delta Y, C_{in}, c_{out}]}_{\text{Weights}} \underbrace{H'_{i-1}[b, x + \Delta X, y + \Delta Y, C_{in}]}_{\text{Prev. Layer}} + \underbrace{b_i[c_{out}]}_{\text{Bias}} \right)$$

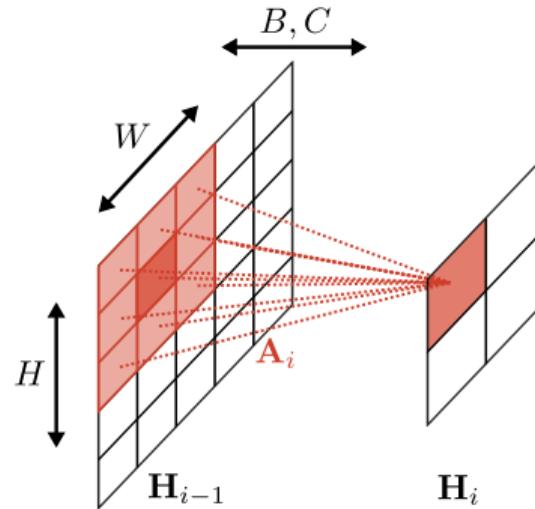
- ▶ Note: if the input is padded, then Δx and Δy are non-negative

Downsampling



- Downsampling **reduces the spatial resolution** (e.g., for image level predictions)
- Downsampling **increases the receptive field** (which pixels influence a neuron)

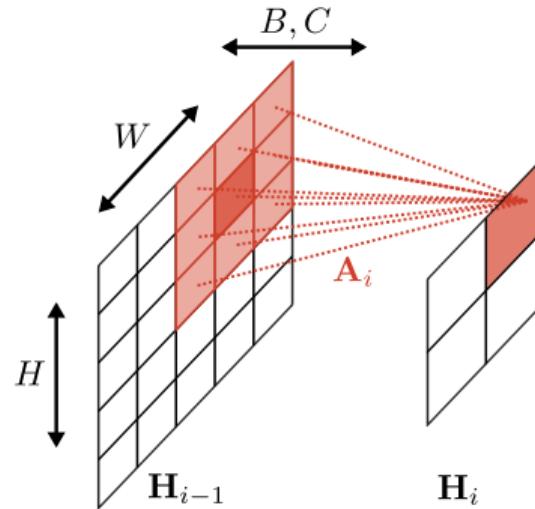
Pooling



$$\underbrace{H_i[b, x, y, c]}_{\text{Current Layer}} = \max_{\Delta x, \Delta y} \underbrace{H_{i-1}[b, s \cdot x + \Delta x, s \cdot y + \Delta y, c]}_{\text{Prev. Layer}}$$

- ▶ Typically, stride $s = 2$ and kernel size $2 \times 2 \Rightarrow$ **reduces spatial dimensions** by 2
- ▶ Pooling has **no parameters** (typical pooling operations: max, min, mean)
- ▶ Pooling is **applied to each channel separately** \Rightarrow keeps number of channels

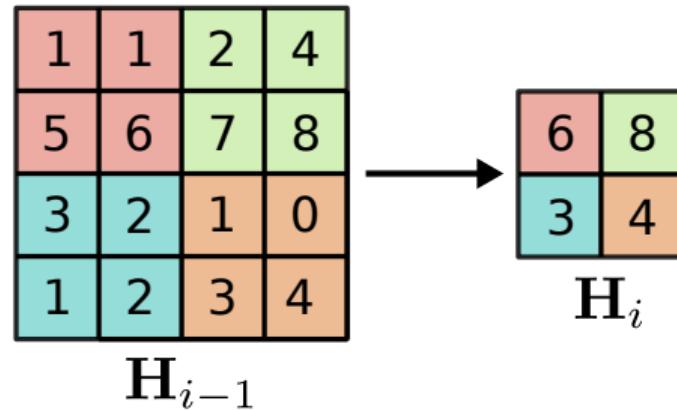
Pooling



$$\underbrace{H_i[b, \mathbf{x}, \mathbf{y}, c]}_{\text{Current Layer}} = \max_{\Delta x, \Delta y} \underbrace{H_{i-1}[b, s \cdot \mathbf{x} + \Delta x, s \cdot \mathbf{y} + \Delta y, c]}_{\text{Prev. Layer}}$$

- ▶ Typically, stride $s = 2$ and kernel size $2 \times 2 \Rightarrow$ **reduces spatial dimensions** by 2
- ▶ Pooling has **no parameters** (typical pooling operations: max, min, mean)
- ▶ Pooling is **applied to each channel separately** \Rightarrow keeps number of channels

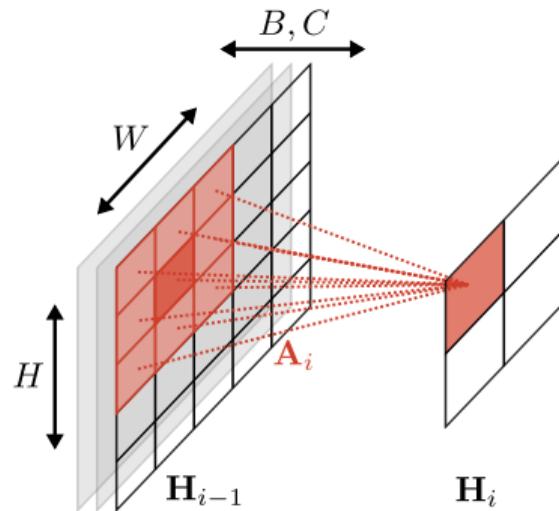
Pooling Example



$$\underbrace{H_i[b, \cancel{x}, \cancel{y}, c]}_{\text{Current Layer}} = \max_{\Delta x, \Delta y} \underbrace{H_{i-1}[b, s \cdot \cancel{x} + \Delta x, s \cdot \cancel{y} + \Delta y, c]}_{\text{Prev. Layer}}$$

- ▶ Max pooling with stride $s = 2$ and kernel size 2×2 reduces spatial dimension by 2
- ▶ Remark: Max pooling provides **invariance** to small translations of the input

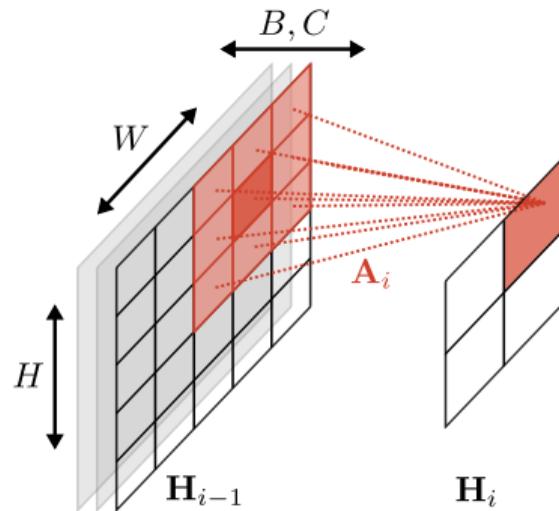
Strided Convolution



$$\underbrace{H_i[b, \mathbf{x}, \mathbf{y}, c_{out}]}_{\text{Current Layer}} = g \left(\underbrace{A_i[\Delta X, \Delta Y, C_{in}, c_{out}]}_{\text{Weights}} \underbrace{H_{i-1}[b, s \cdot \mathbf{x} + \Delta X, s \cdot \mathbf{y} + \Delta Y, C_{in}]}_{\text{Prev. Layer}} + \underbrace{b_i[c_{out}]}_{\text{Bias}} \right)$$

- **Move convolution filter** (with parameters \mathbf{A} and \mathbf{b}) by stride s
- **Learned downsampling.** Often replaces max pooling today (e.g., in ResNet)

Strided Convolution

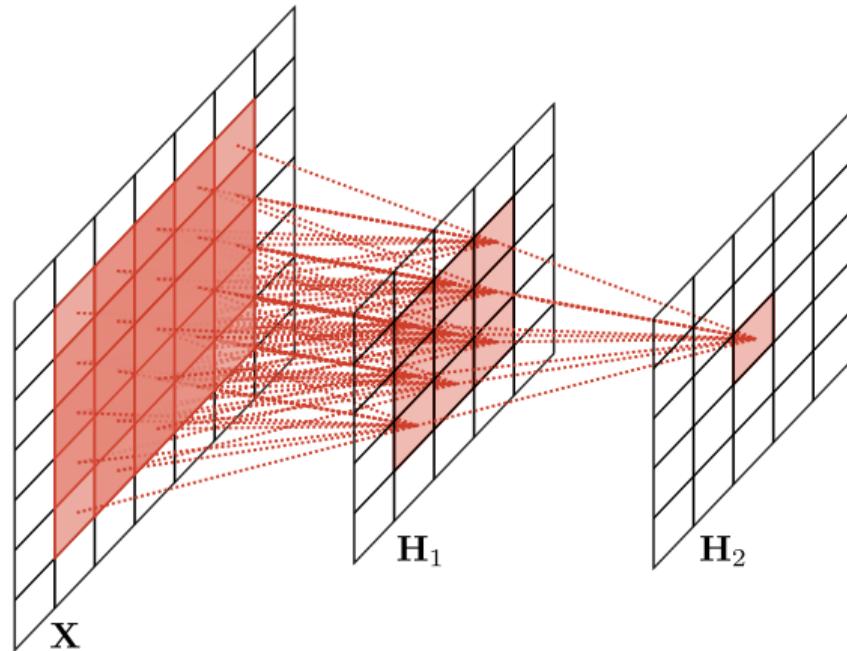


$$\underbrace{H_i[b, \mathbf{x}, \mathbf{y}, c_{out}]}_{\text{Current Layer}} = g \left(\underbrace{A_i[\Delta X, \Delta Y, C_{in}, c_{out}]}_{\text{Weights}} \underbrace{H_{i-1}[b, s \cdot \mathbf{x} + \Delta X, s \cdot \mathbf{y} + \Delta Y, C_{in}]}_{\text{Prev. Layer}} + \underbrace{b_i[c_{out}]}_{\text{Bias}} \right)$$

- **Move convolution filter** (with parameters \mathbf{A} and \mathbf{b}) by stride s
- **Learned downsampling.** Often replaces max pooling today (e.g., in ResNet)

Receptive Field and Arithmetics

Receptive Field

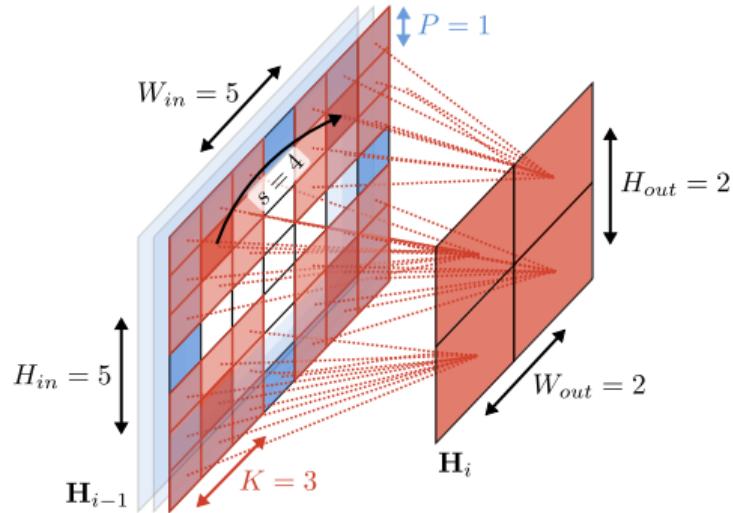


- **Receptive field** of a neuron is defined as all pixels in the input \mathbf{X} which influence it

Arithmetics

Assuming:

- ▶ Input: $W_{in} \times H_{in}$
- ▶ Filter: $K \times K$
- ▶ Padding: P
- ▶ Stride: s

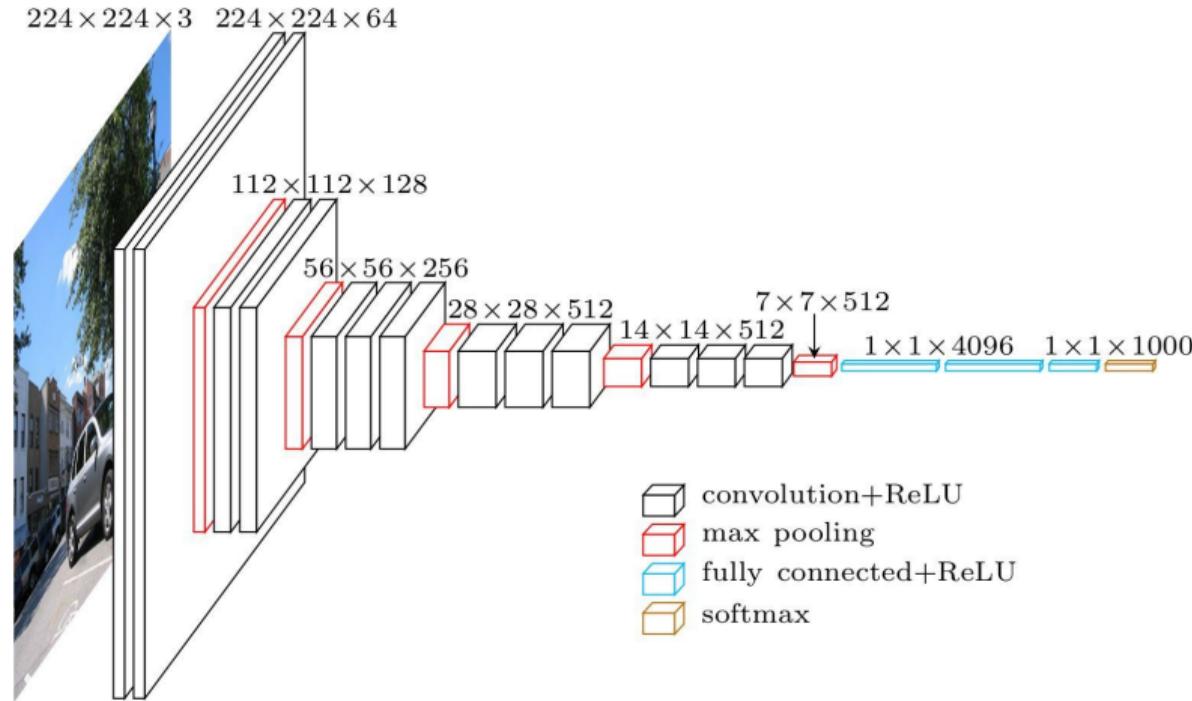


Then, the output has size:

$$\underbrace{\left(\left\lfloor \frac{W_{in} + 2P - K}{s} \right\rfloor + 1 \right)}_{W_{out}} \times \underbrace{\left(\left\lfloor \frac{H_{in} + 2P - K}{s} \right\rfloor + 1 \right)}_{H_{out}}$$

Fully Connected Layers

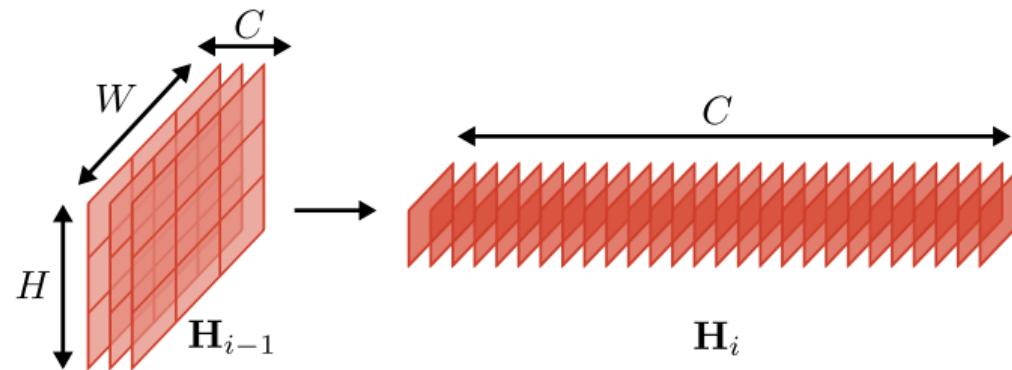
Fully Connected Layers



- ▶ Fully connected layers are most **memory intensive** part of VGG architecture

Fully Connected Layers

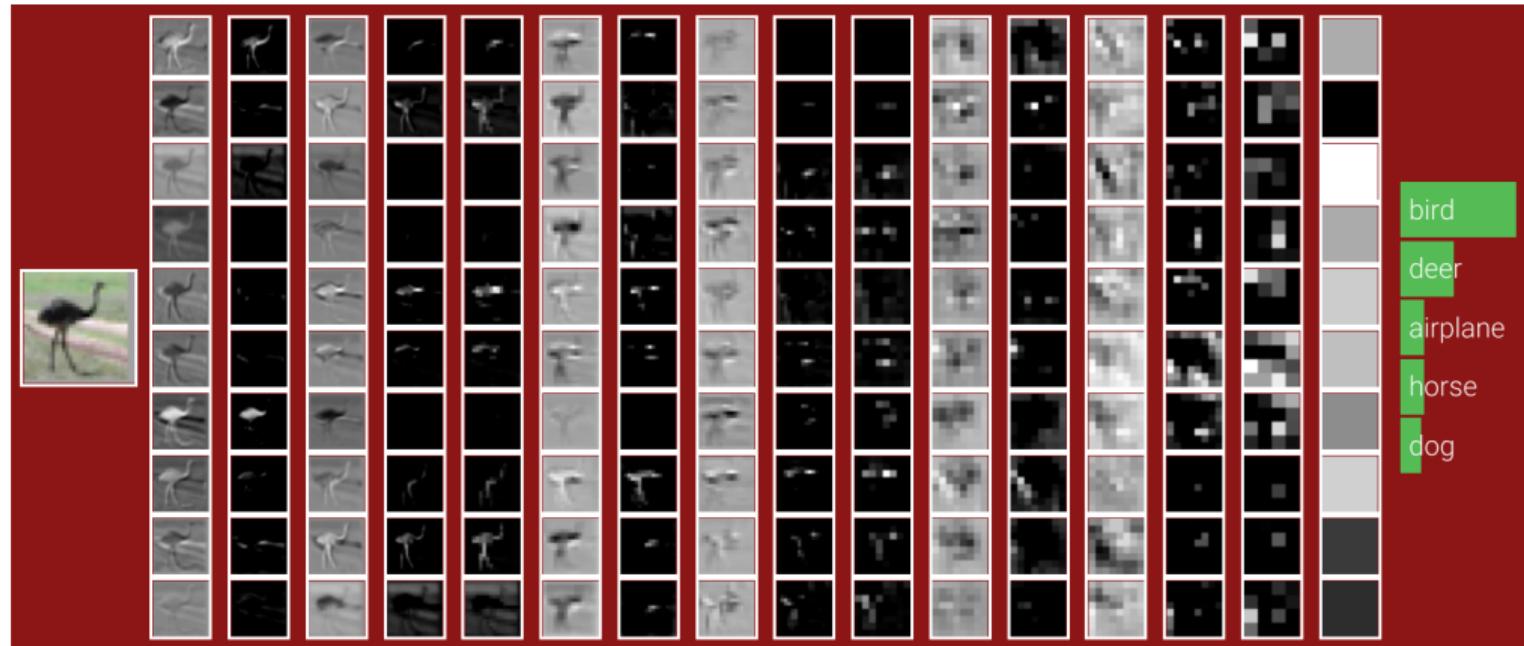
- Reshape $H_{i-1}[B, X, Y, C]$ into $H_i[B, C]$



- Now X and Y are **reshaped** into the feature channel dimension C
- A fully connected layer in **Einstein notation** can be written as:

$$\underbrace{H_i[b, c_{out}]}_{\text{Current Layer}} = g \left(\underbrace{A_i[c_{out}, C_{in}]}_{\text{Weights}} \underbrace{H_{i-1}[b, C_{in}]}_{\text{Prev. Layer}} + \underbrace{b_i[c_{out}]}_{\text{Bias}} \right)$$

Convolution Network Example

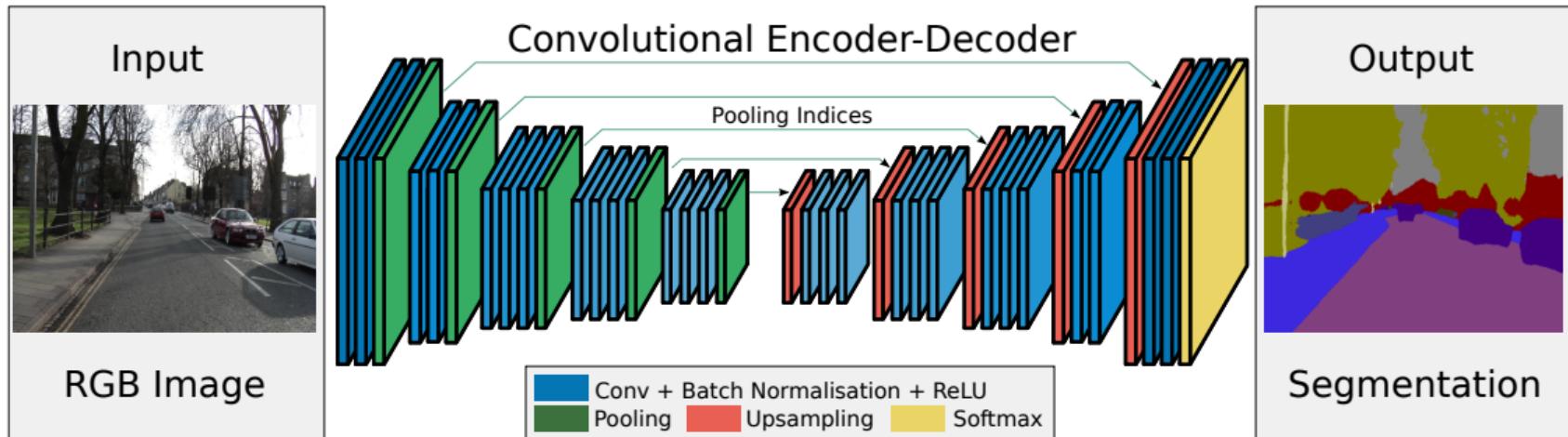


<http://cs231n.stanford.edu/2017/index.html>

7.3

Upsampling

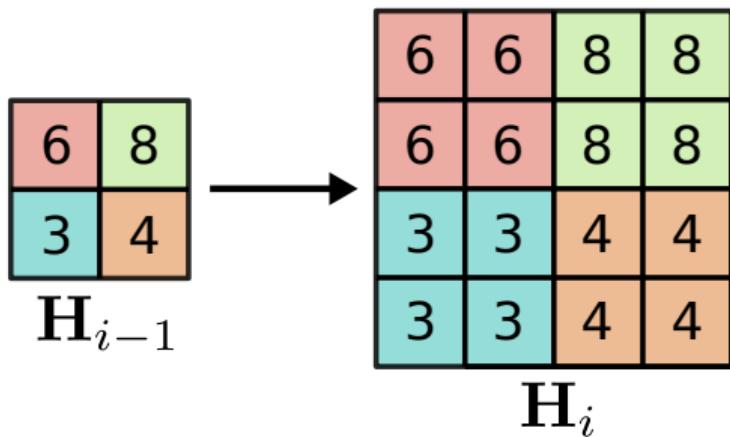
Upsampling



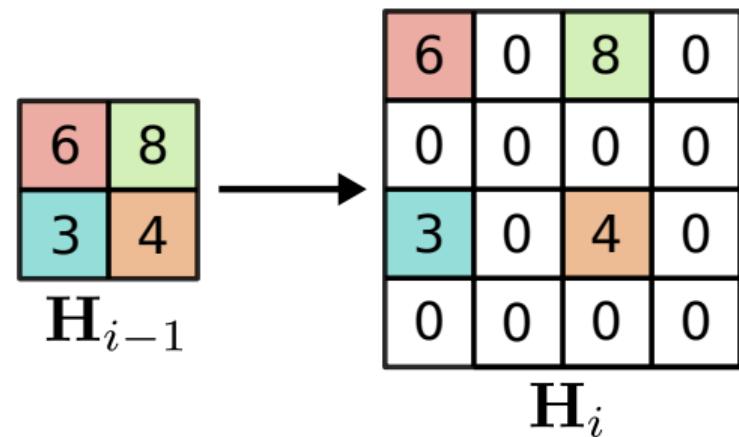
- ▶ If **pixel-level outputs** are desired, we have to **upsample** the features again
- ▶ Downsampling provides strong features with large receptive field
- ▶ Upsampling yields output at the same resolution as input

Upsampling

Nearest Neighbor Upsampling

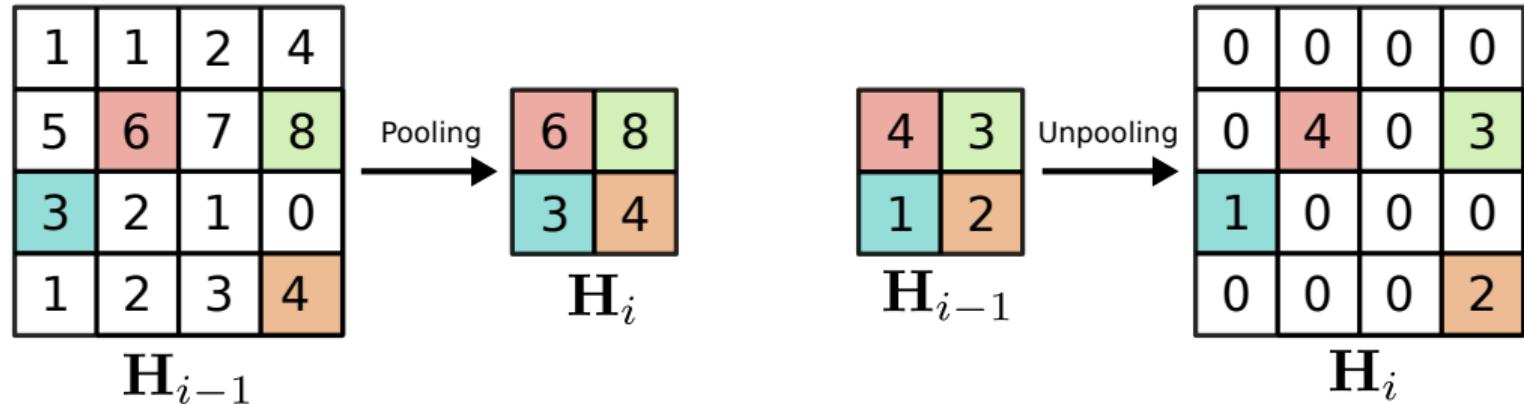


Bed of Nails Upsampling



- ▶ **Nearest neighbor:** scale each channel using nearest neighbor interpolation
- ▶ **Bilinear:** scale each channel using bilinear neighbor interpolation
- ▶ **Bed of nails:** insert elements at sparse location (followed by convolution)

Max Unpooling



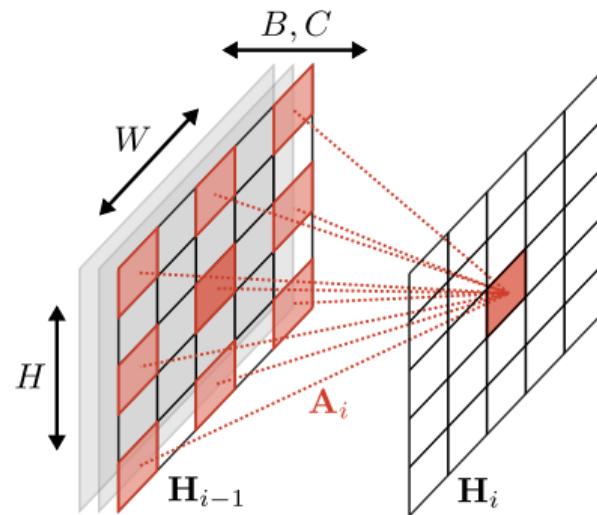
- ▶ For unpooling, **remember** which element was **maximum** during pooling
- ▶ Requires corresponding pairs of downsampling and upsampling layers
- ▶ This approach has been used in SegNet

Dilated Convolution

Dilated Convolution

- ▶ Dilated convolutions are an alternative to combining downsampling and upsampling operations in order to reach a large receptive field size quickly
- ▶ Dilated convolutions increase the receptive field of standard convolutions without increasing the number of parameters
- ▶ Thus, a network with dilated convolutions is able to perform image-level predictions (e.g., semantic segmentation) without upsampling and downsampling
- ▶ Without dilated convolutions, a very large number of layers of standard convolutions would be required to reach the same receptive field size without using upsampling and downsampling (as in a U-Net)
- ▶ Remark: Dilated convolutions are not upsampling operations

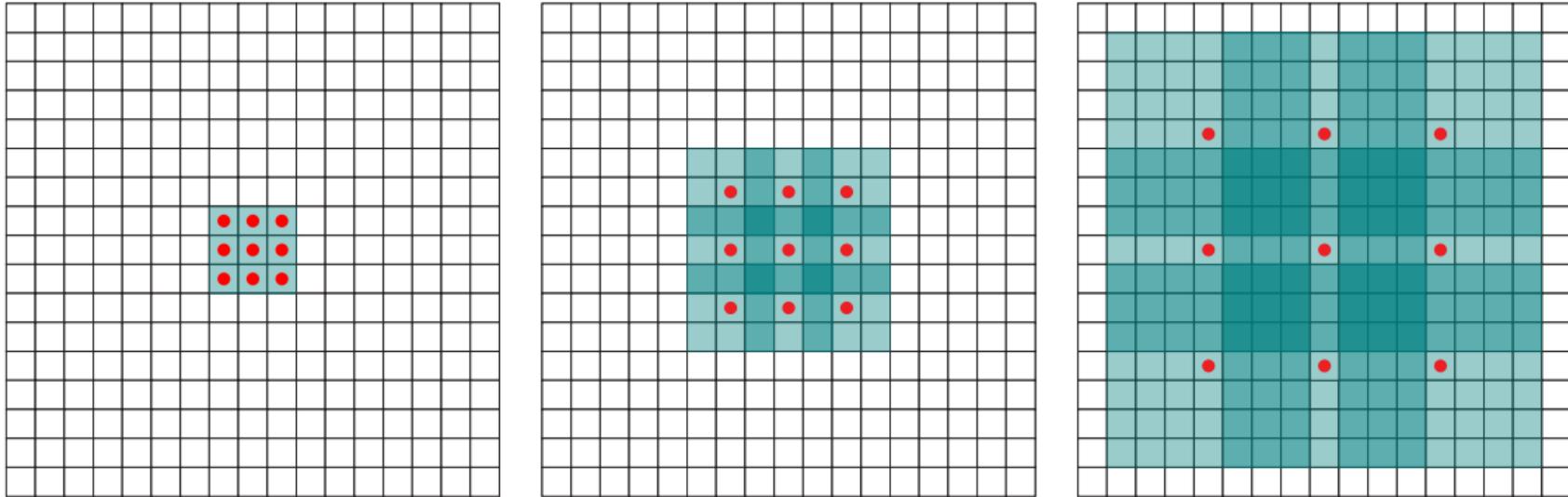
Dilated Convolution



$$\underbrace{H_i[b, x, y, c_{out}]}_{\text{Current Layer}} = g \left(\underbrace{A_i[\Delta X, \Delta Y, C_{in}, c_{out}]}_{\text{Weights}} \underbrace{H_{i-1}[b, x + d \cdot \Delta X, y + d \cdot \Delta Y, C_{in}]}_{\text{Prev. Layer}} + \underbrace{b_i[c_{out}]}_{\text{Bias}} \right)$$

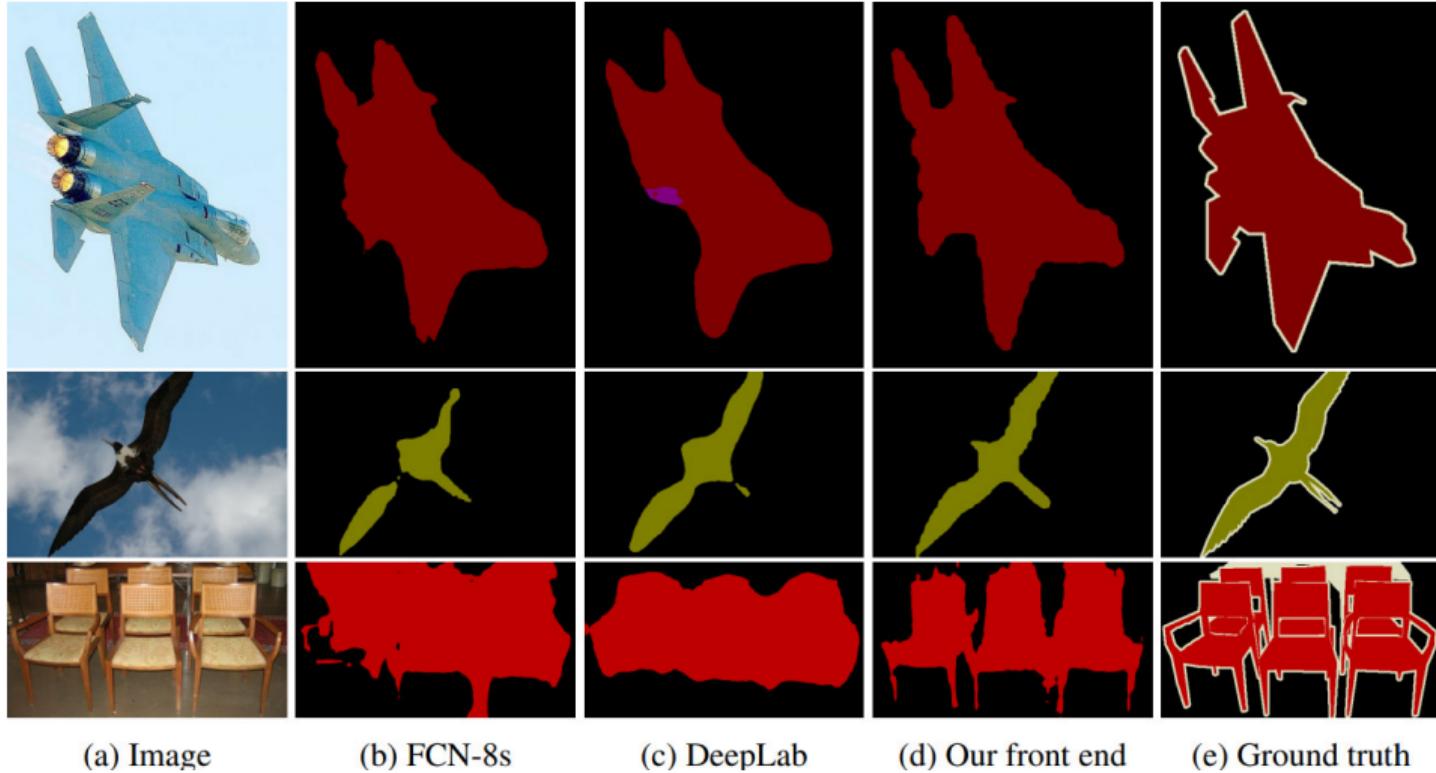
- Dilation factor d (here $d = 2$) leads to more “distributed” sampling of the input
- **Increases receptive field** without increasing parameters or reducing spatial dim.

Dilated Convolution



- ▶ **No pooling** required to increase the receptive field
- ▶ Can produce predictions at the **same resolution** as the inputs
- ▶ Examples: semantic segmentation, depth, optical flow, ...

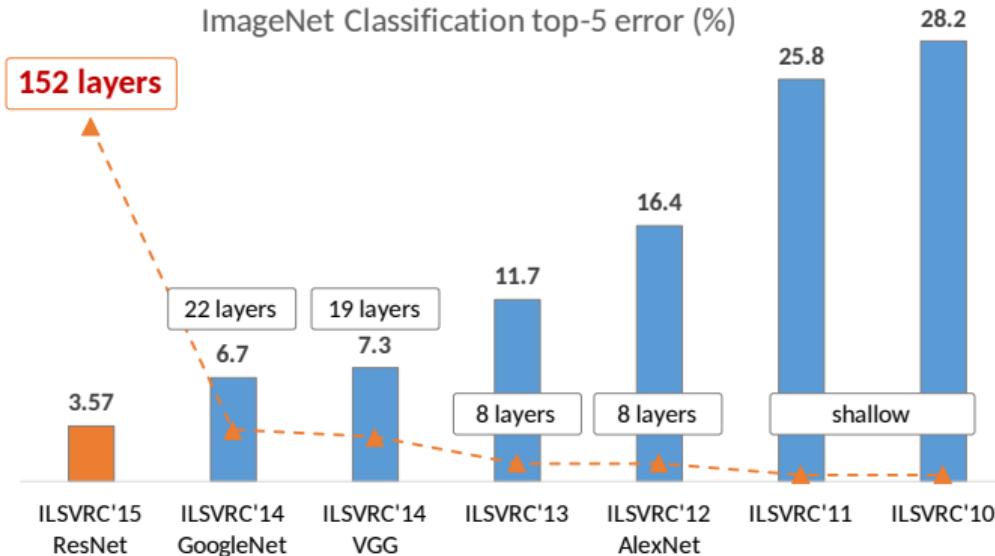
Dilated Convolution



7.4

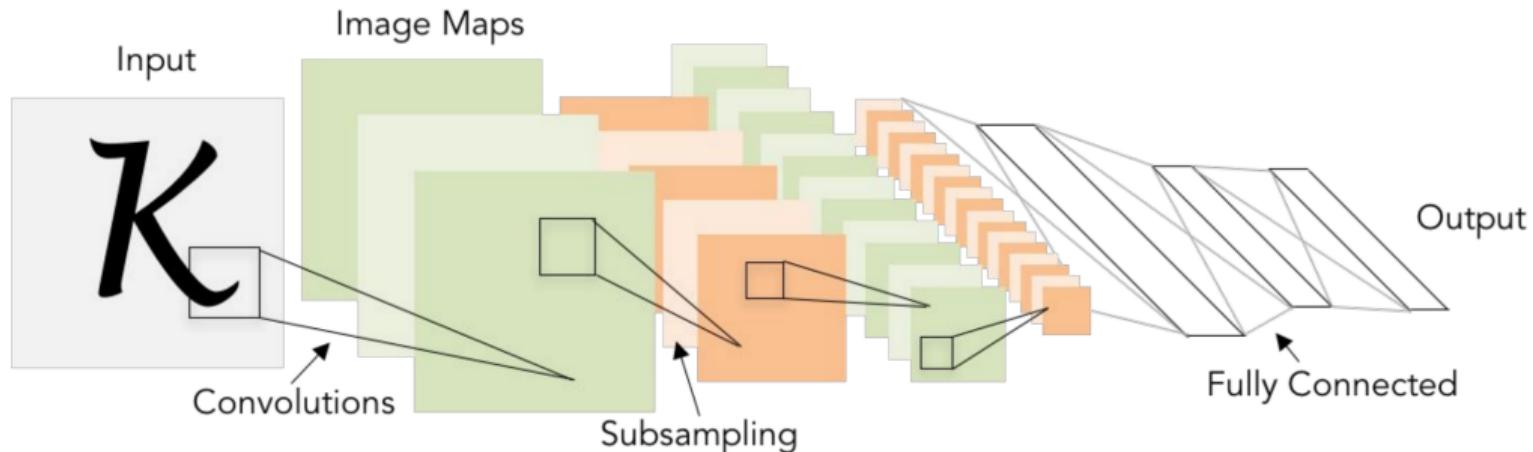
Architectures

ImageNet Large Scale Visual Recognition Challenge



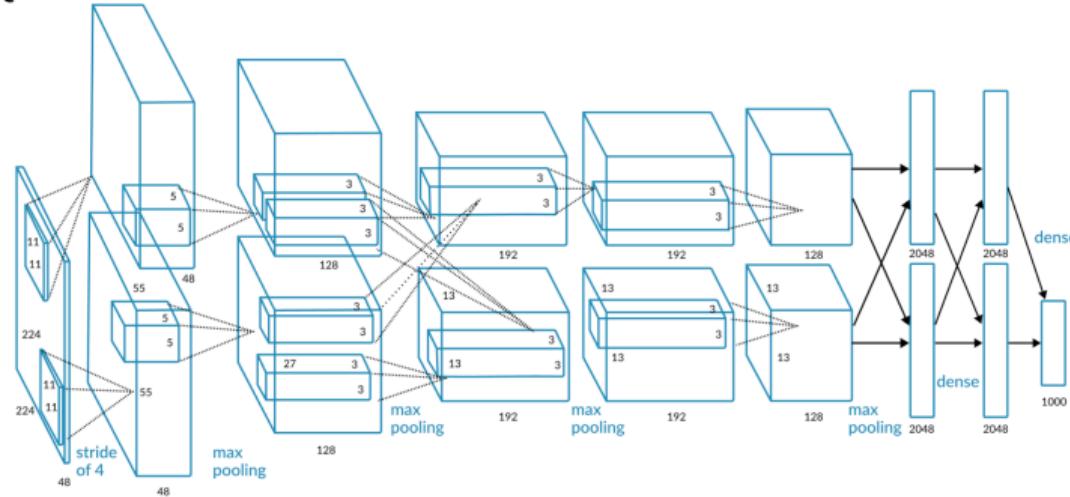
- Deeper networks are better (provided enough training data and compute)
- Nearly all state-of-the-art networks for image applications use convolution layers

1998: LeNet-5



- ▶ 2 convolution layers (5×5), 2 pooling layers (2×2), 2 fully connected layers
- ▶ Achieved state-of-the-art accuracy on MNIST (prior to ImageNet)

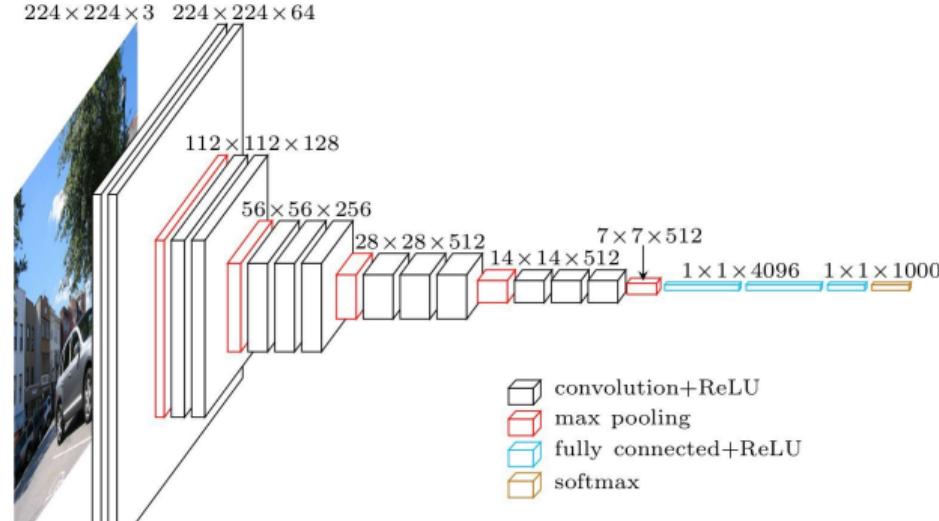
2012: AlexNet



AlexNet:

- ▶ 8 layers, ReLUs, dropout, data augmentation, trained on 2 GTX 580 GPUs
- ▶ Number of feature channels increase with depth, spatial resolution decreases
- ▶ Triggered deep learning revolution, showed that CNNs work well in practice

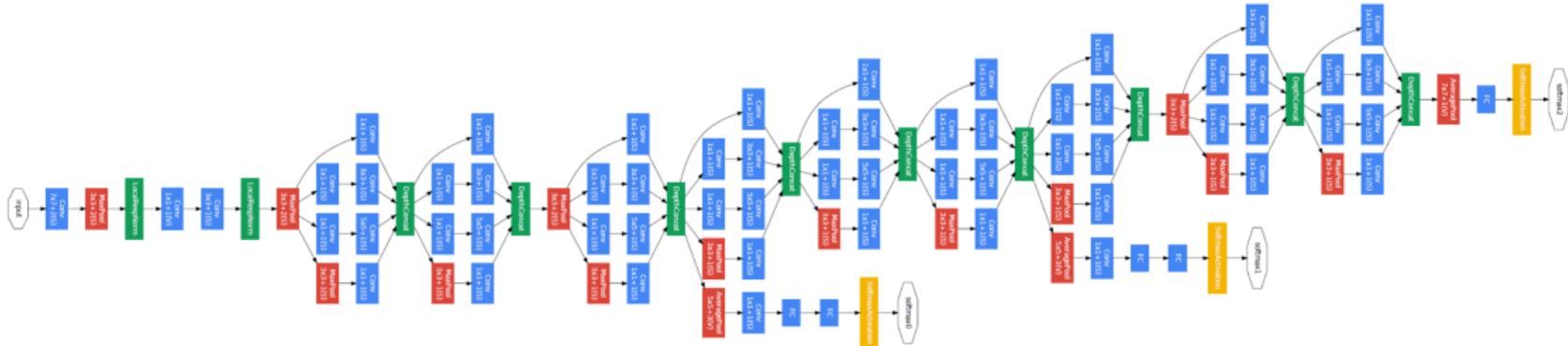
2015: VGG



VGG:

- ▶ Uses 3×3 convolutions everywhere (same expressiveness, fewer parameters)
- ▶ Three 3×3 layers: same receptive field as one 7×7 layer, but less parameters
- ▶ 2 Variants: 16 and 19 Layers. Showed that deeper networks are better.

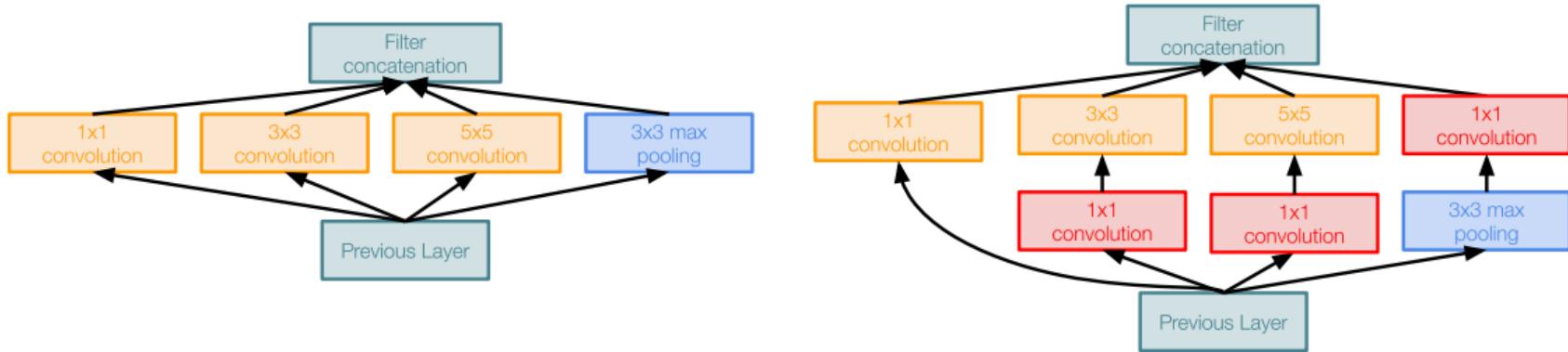
2015: Inception / GoogLeNet



Inception:

- ▶ 22 layers. Inception modules utilize conv/pool operations with varying filter size
- ▶ Multiple intermediate classification heads to improve gradient flow
- ▶ Global average pooling (no FC layers), 27x less parameters (5 million) than VGG-16
- ▶ Uses 1×1 convolutions to reduce number of features \Rightarrow higher efficiency

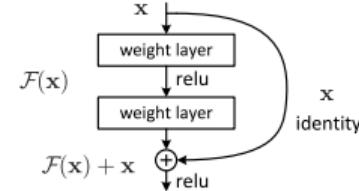
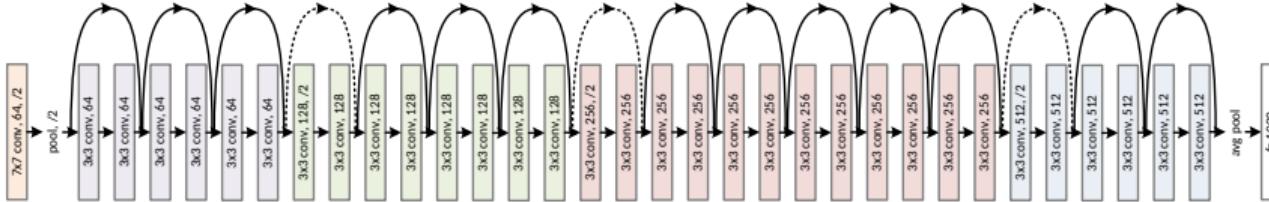
2015: Inception / GoogLeNet



Inception:

- ▶ 22 layers. Inception modules utilize conv/pool operations with varying filter size
- ▶ Multiple intermediate classification heads to improve gradient flow
- ▶ Global average pooling (no FC layers), 27x less parameters (5 million) than VGG-16
- ▶ Uses 1×1 convolutions to reduce number of features \Rightarrow higher efficiency

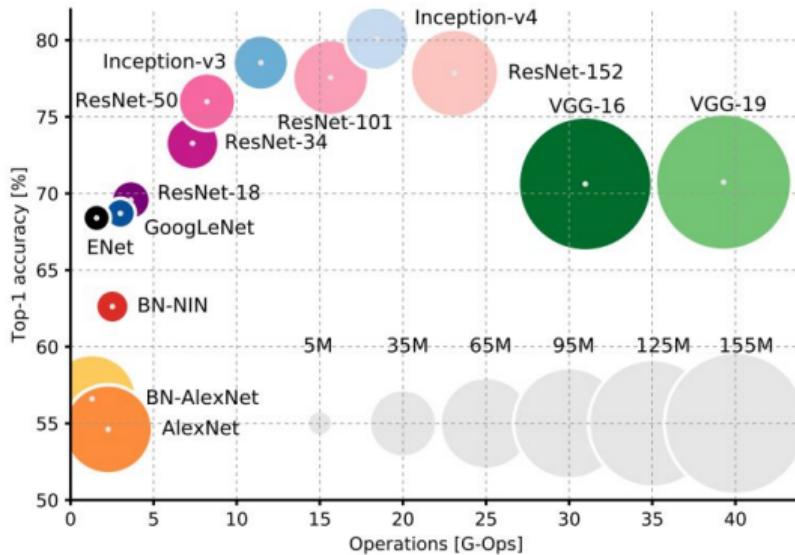
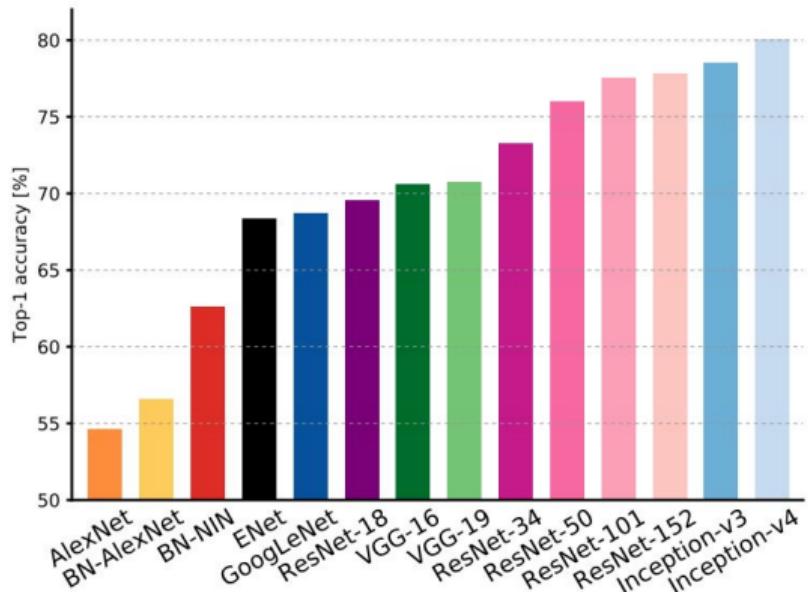
2016: ResNet



ResNet:

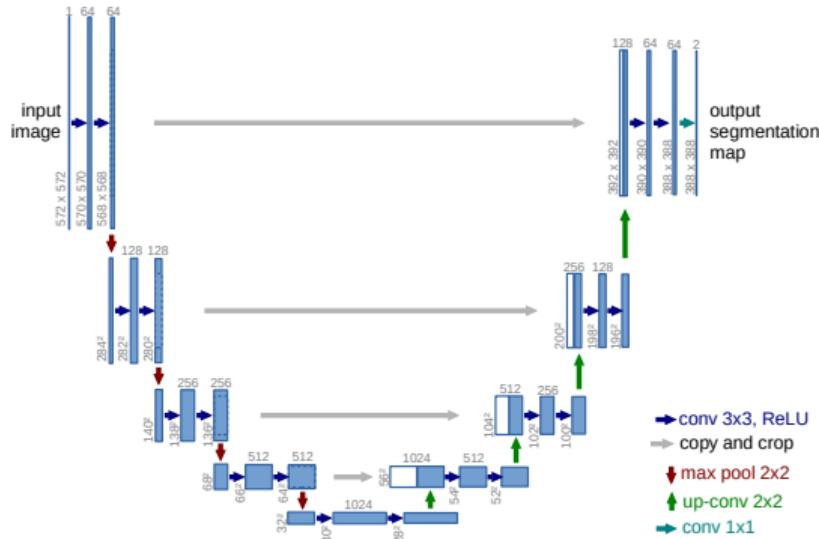
- Residual connections allow for training deeper networks (up to 152 layers)
- Very simple and regular network structure with 3×3 convolutions
- Uses strided convolutions for downsampling
- ResNet and ResNet-like architectures are dominating today

Complexity



- VGG has most parameters and is slowest
- ResNet/Inception/GoogLeNet are faster and have fewer parameters

U-Net



U-Net:

- Max-pooling, up-convolutions and skip-connections
- Defacto standard for many tasks with image output (e.g., depth, segmentation)

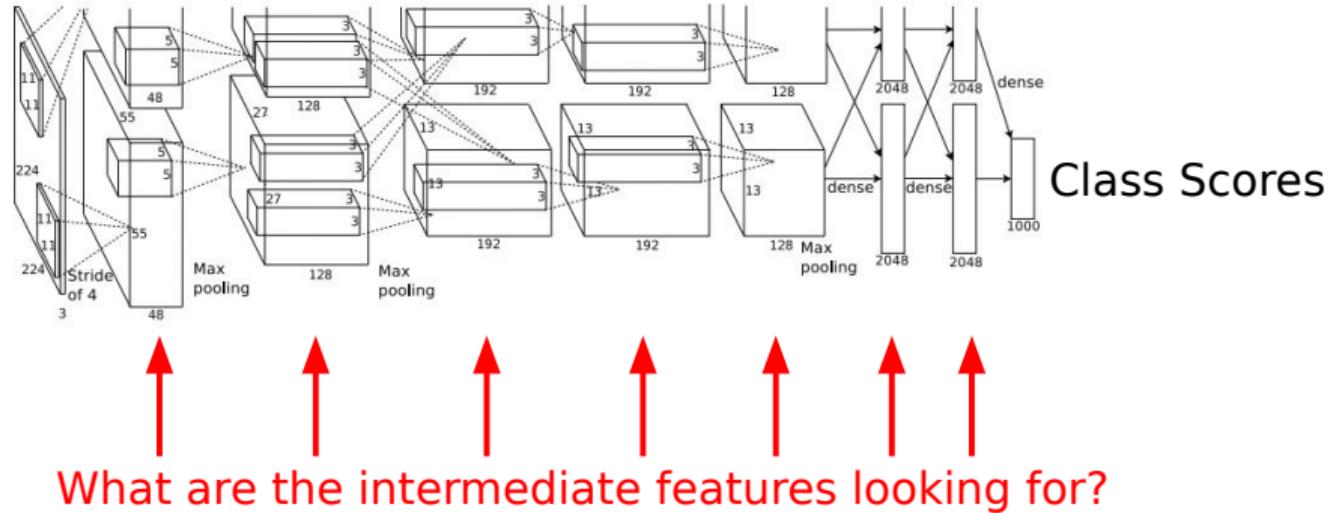
7.5

Visualization

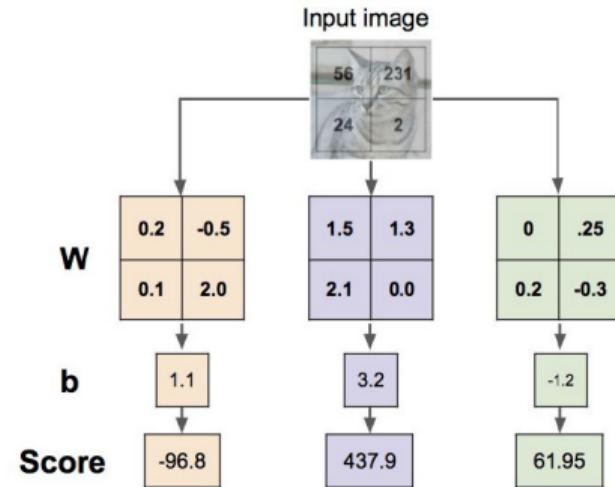
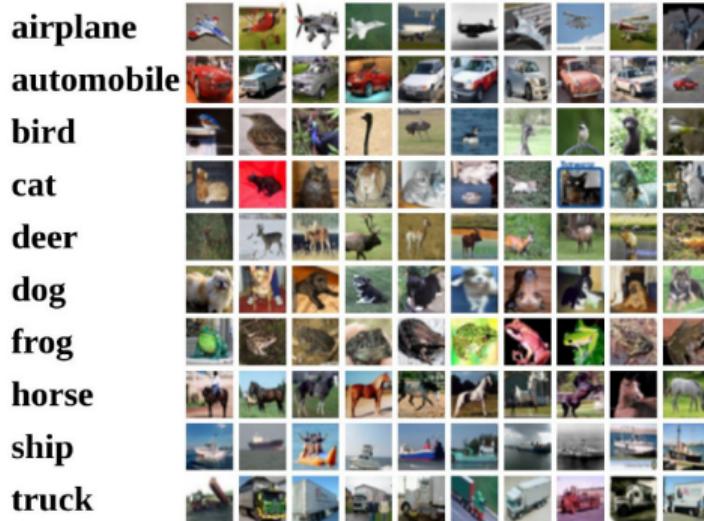
Visualization



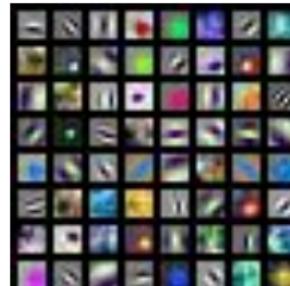
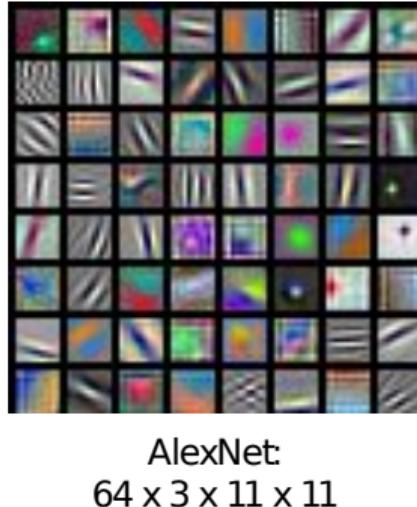
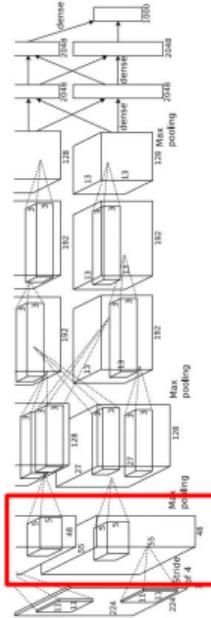
Input Image



Visualizing a Linear Classifier



First Layers: Visualization of Features



- ▶ First layer learns **simple gradient/Gabor filters** ⇒ connection to V1 cells in brain
- ▶ Visualization less useful for later layers

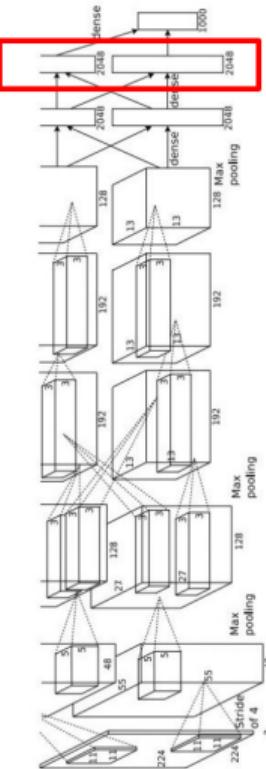
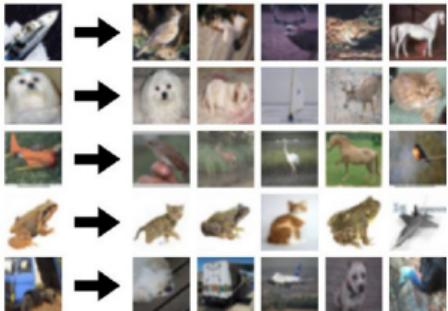
Last Layer: Nearest Neighbors

4096-dim vector

Test image L2 Nearest neighbors in feature space



Recall: Nearest neighbors
in pixel space

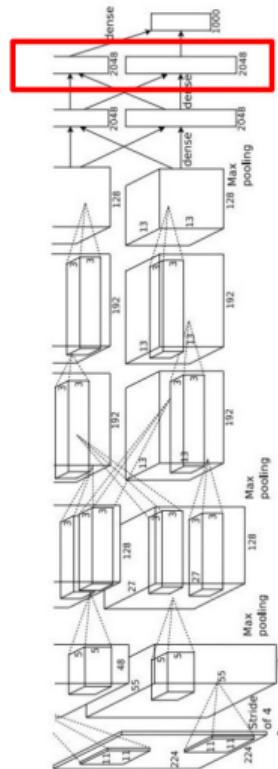
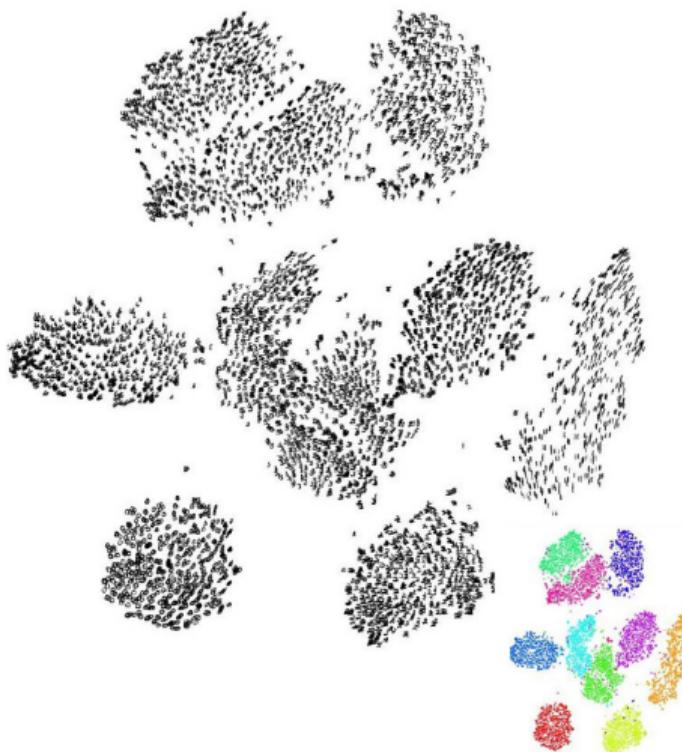


Last Layer: Dimensionality Reduction

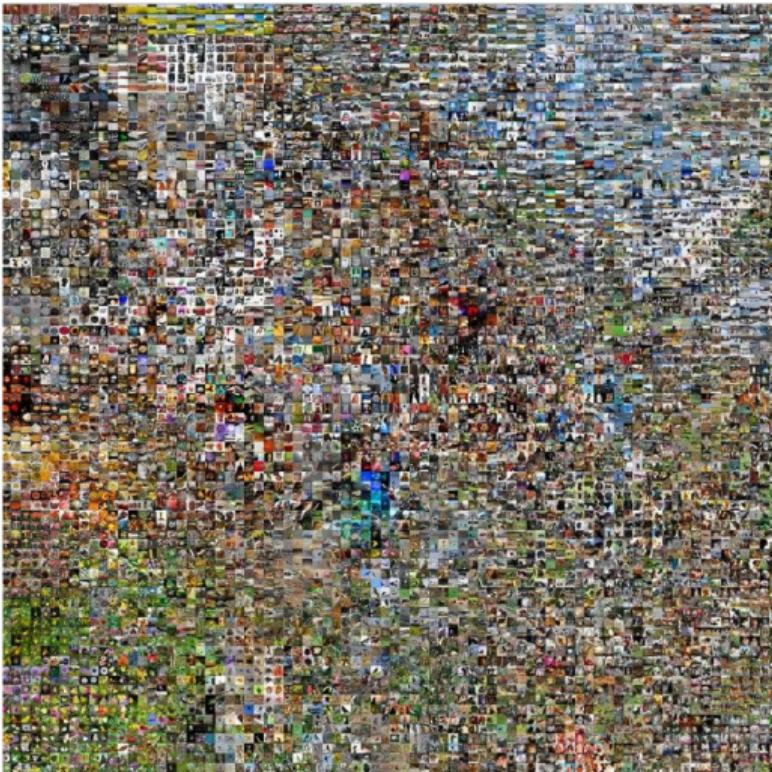
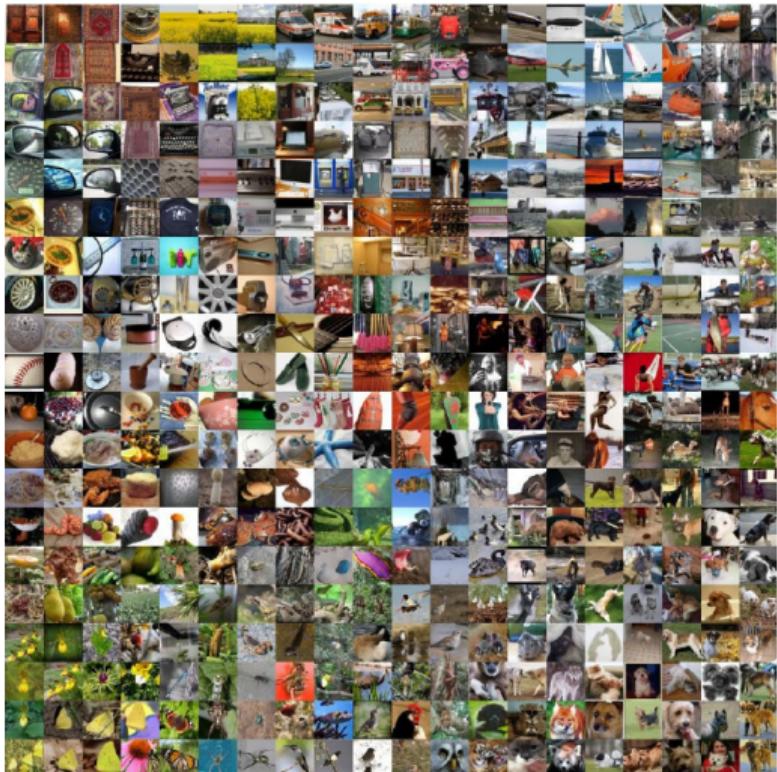
Visualize the “space” of FC7 feature vectors by reducing dimensionality of vectors from 4096 to 2 dimensions

Simple algorithm: Principal Component Analysis (PCA)

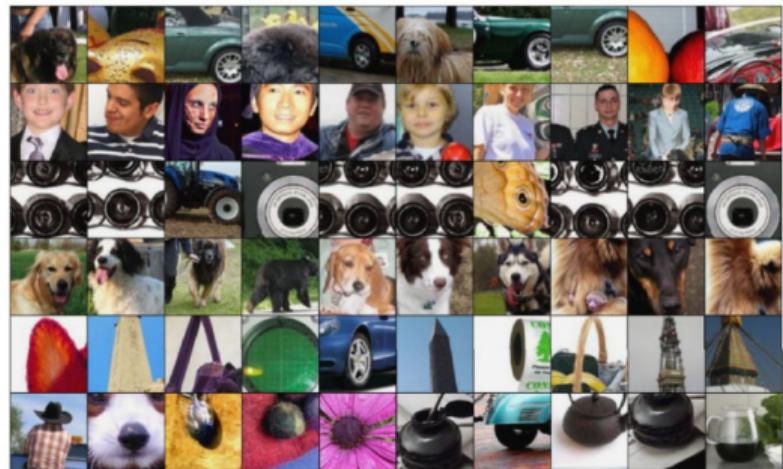
More complex: t-SNE



Last Layer: Dimensionality Reduction

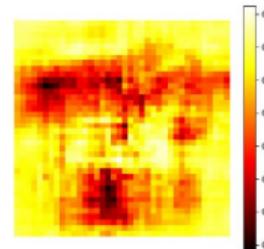
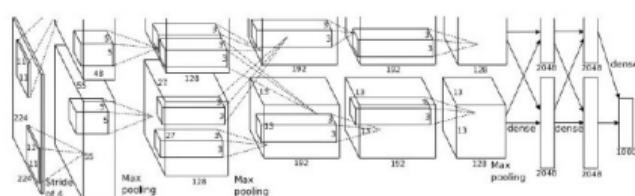
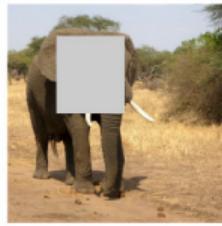
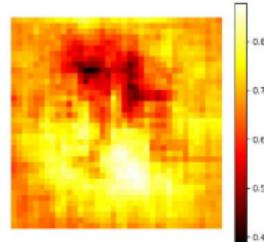
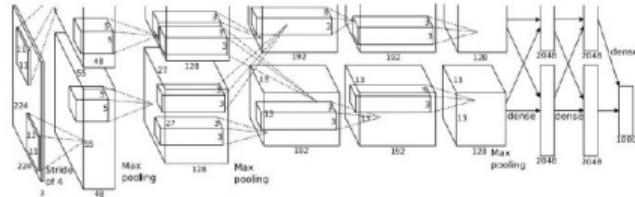


Maximally Activating Patches



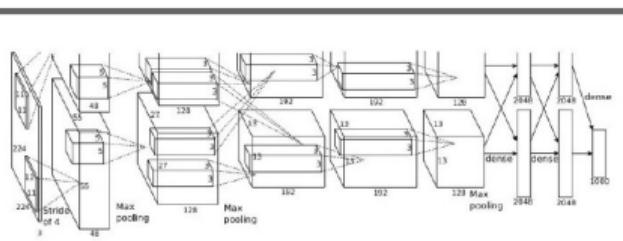
- ▶ Run many images through network
- ▶ Record which images/patches maximally **activate a specific neuron**
- ▶ Here: each row corresponds to one neuron in the last convolutional layer

Saliency via Masking

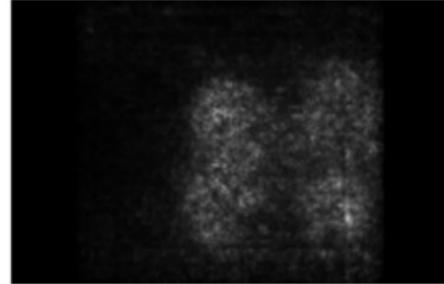


- ▶ **Mask image partially** before feeding to CNN (try many image locations)
- ▶ How much do the predicted probabilities for the correct class change?
- ▶ Results in **saliency map**

Saliency via Backpropagation

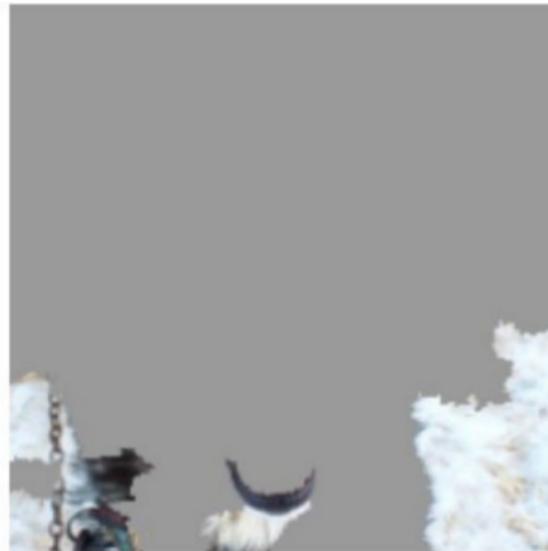
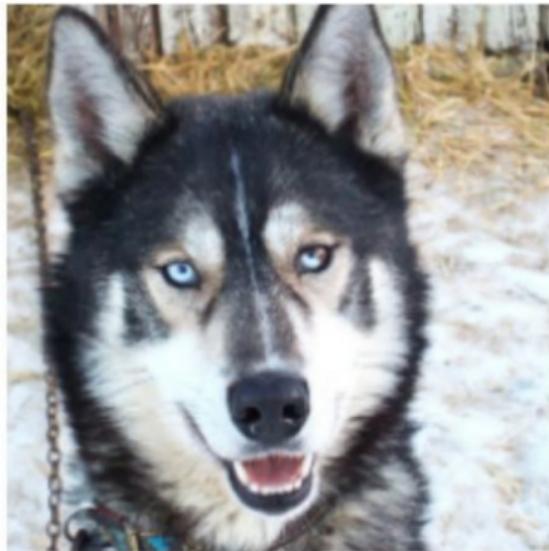


Dog



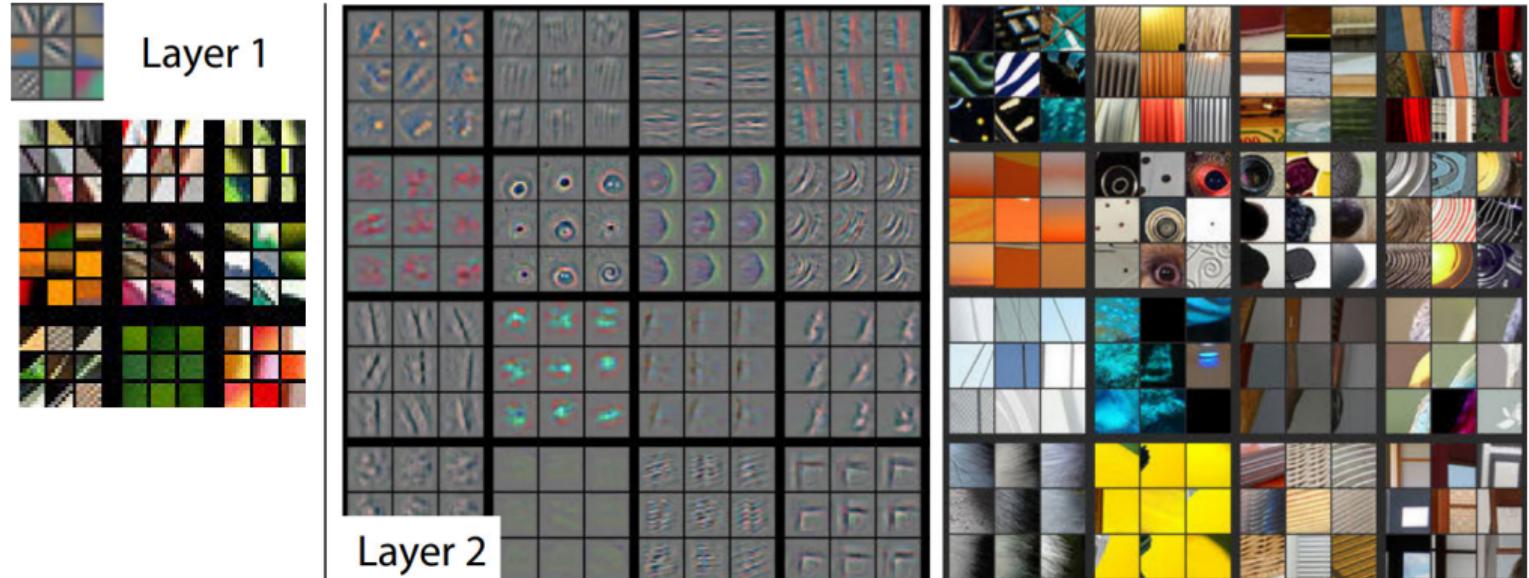
- ▶ Compute gradient of class score with respect to image pixels
- ▶ Changing which pixels of the input affects class score most strongly?
- ▶ Threshold to obtain **saliency map**

Saliency via Backpropagation



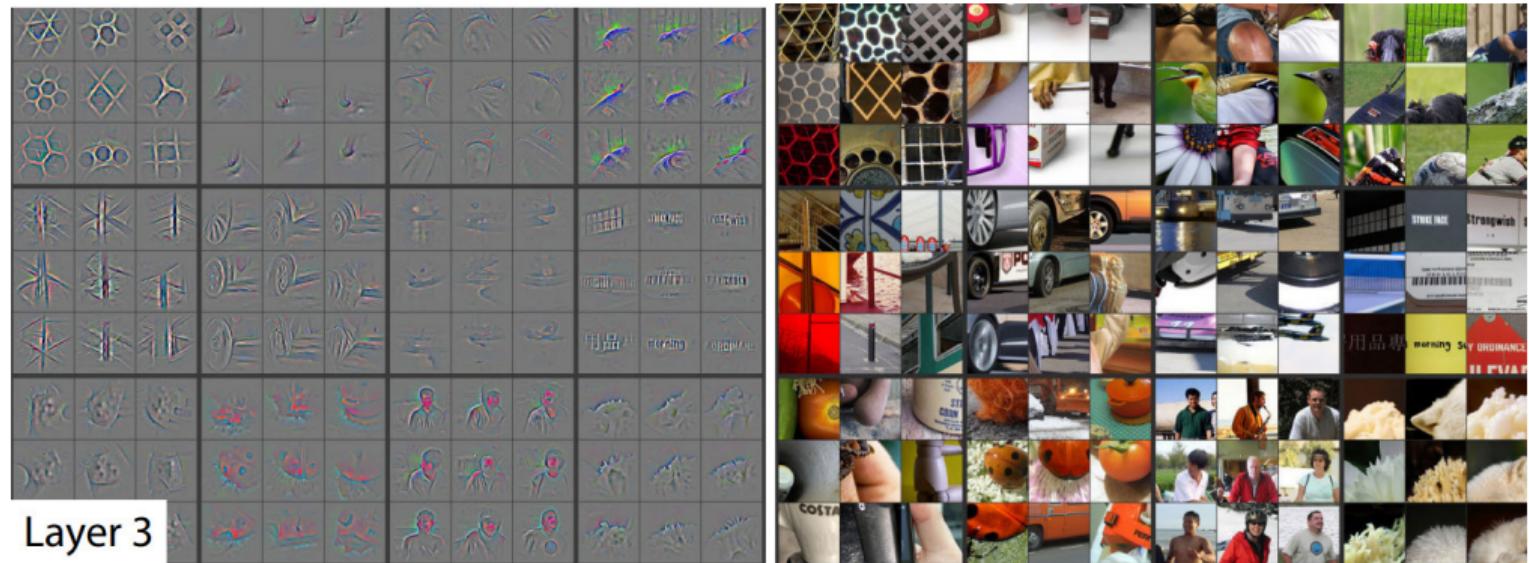
- ▶ Saliency maps can also **uncover unwanted biases** in the data/model
- ▶ In this case, the classifier detects snow to wrongly classify dog as wolf

Deconvolution



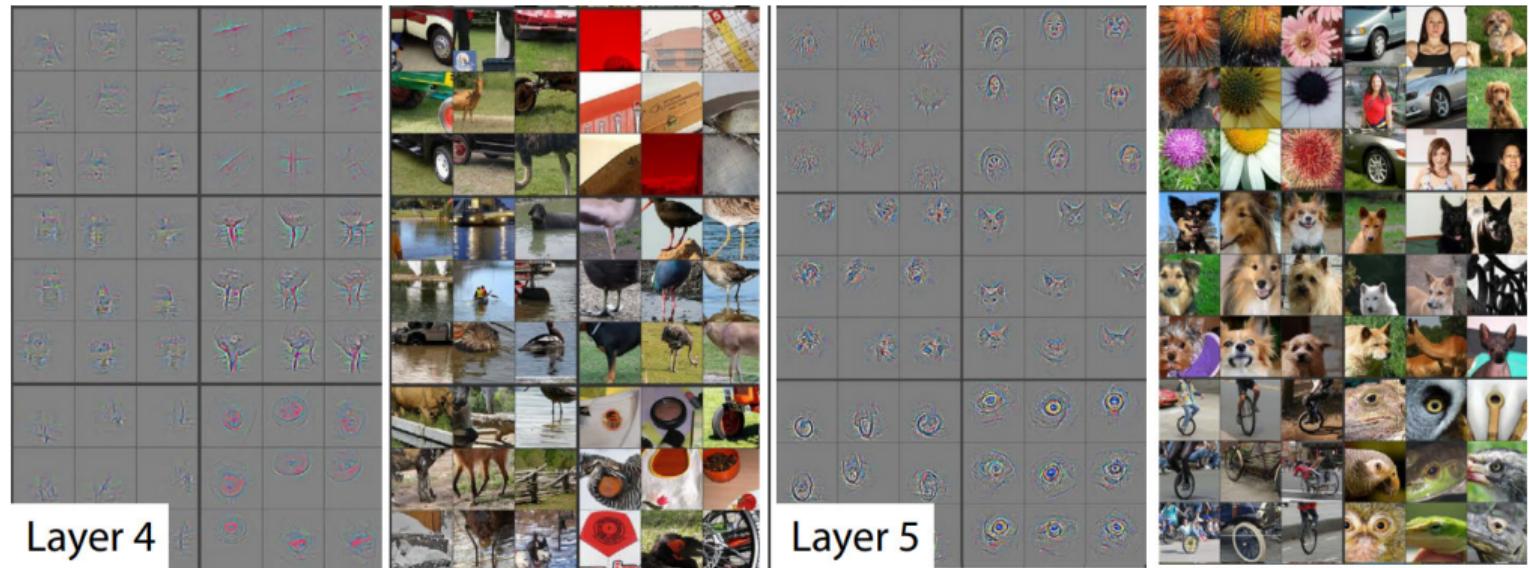
- **DeConvNet** constructs a ConvNet in reverse order, starting from one neuron
- Remembers index of maximal activations during pooling for backward pass

Deconvolution



- ▶ **DeConvNet** constructs a ConvNet in reverse order, starting from one neuron
- ▶ Remembers index of maximal activations during pooling for backward pass

Deconvolution



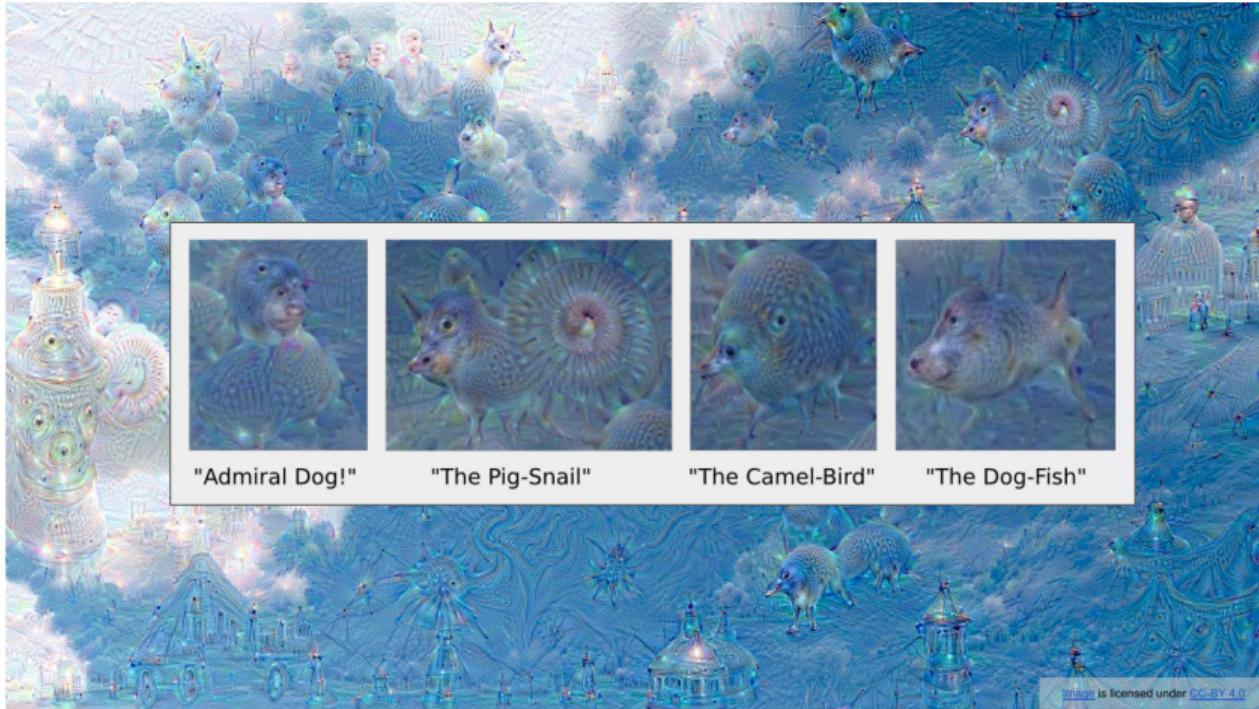
- **DeConvNet** constructs a ConvNet in reverse order, starting from one neuron
- Remembers index of maximal activations during pooling for backward pass

DeepDream



- ▶ **Amplify activation** of chosen layer wrt. a given image (gradient = activation)

DeepDream



- **Amplify activation** of chosen layer wrt. a given image (gradient = activation)