PHYS 139/239: Machine Learning in Physics

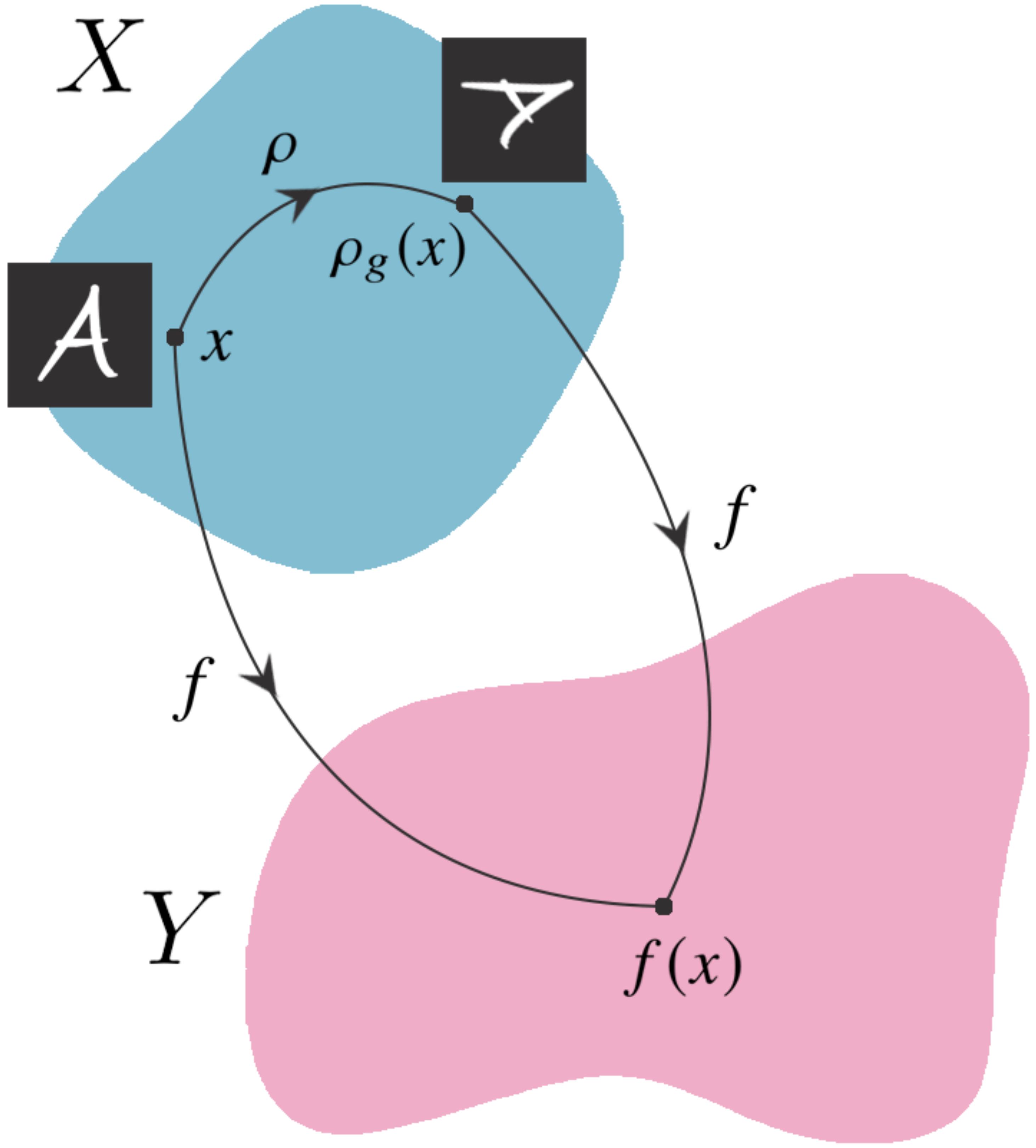
Lecture 7: Convolutional neural networks

Javier Duarte — January 31, 2023

Symmetries

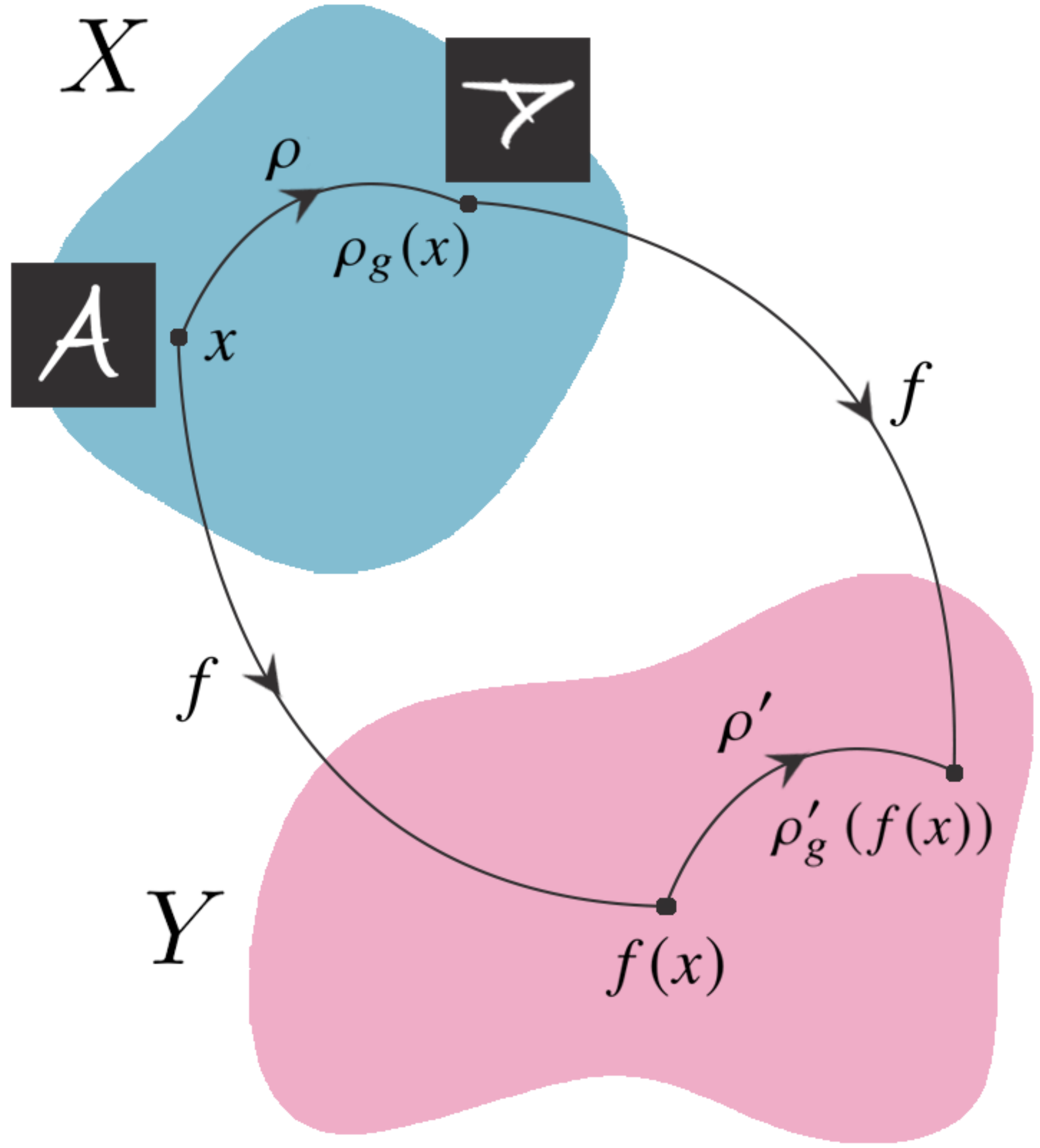
Invariance

$$f(\rho_g(x)) = f(x)$$



Equivariance

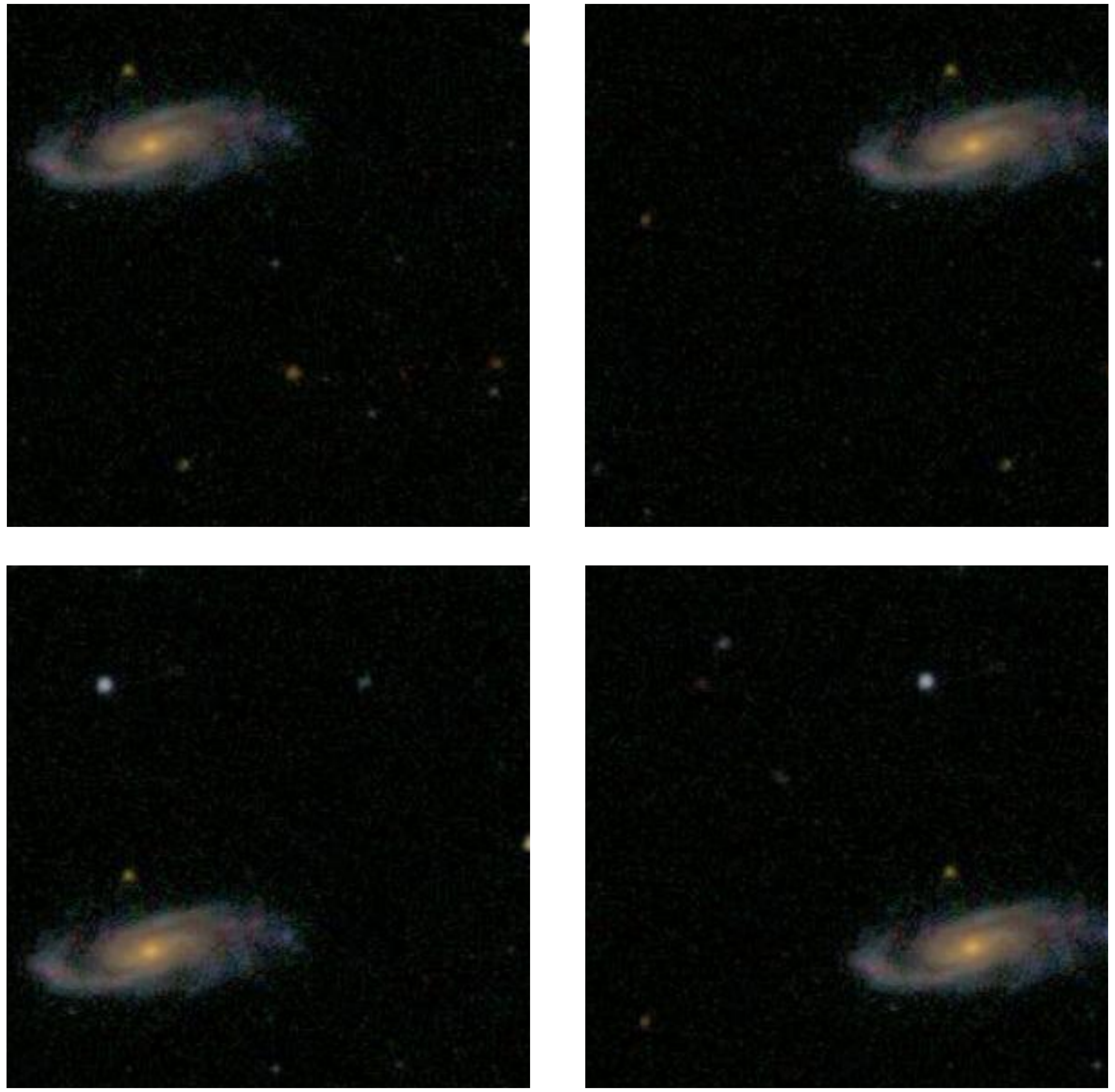
$$f(\rho_g(x)) = \rho'_g(f(x))$$



Translational invariance

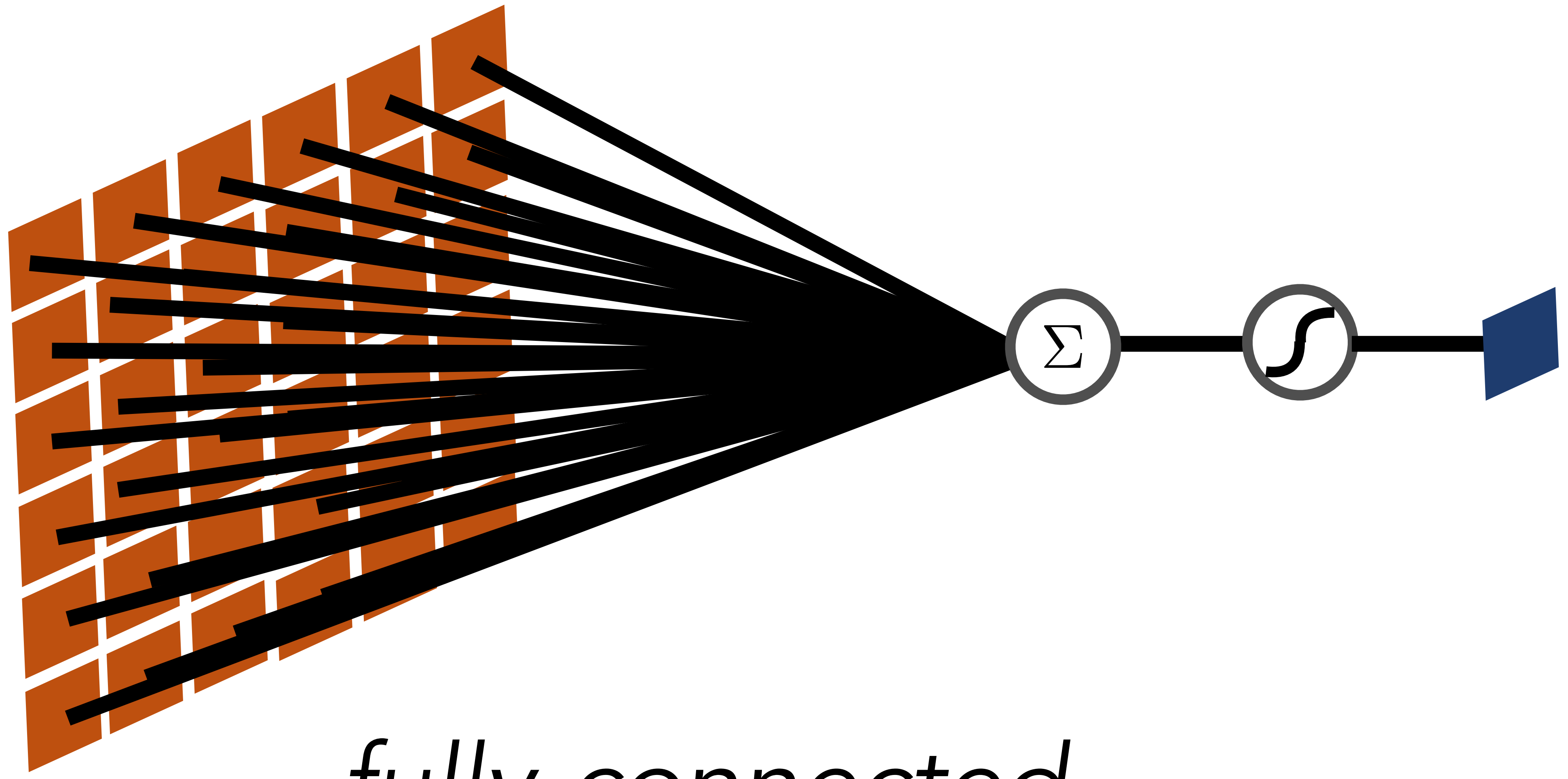
- For the purpose of classifying galaxy morphologies (e.g. spiral), the answer shouldn't depend on the absolute location of the pixels

- For simplicity, imagine there are 4 possible locations the galaxy might show up (top left, top right, bottom left, and bottom right)

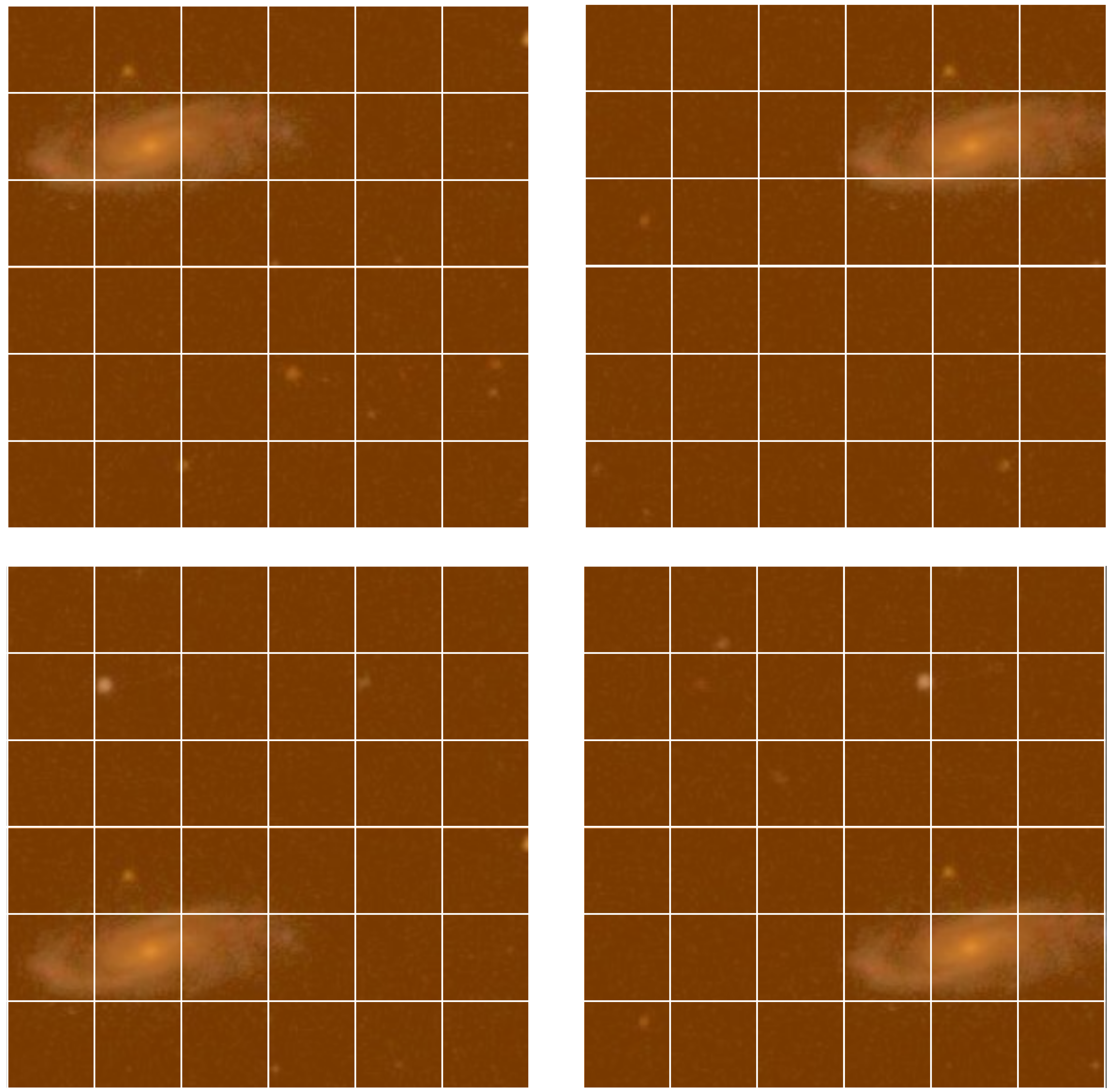


Fully-connected neural networks

Fully-connected neural networks
are not translation invariant

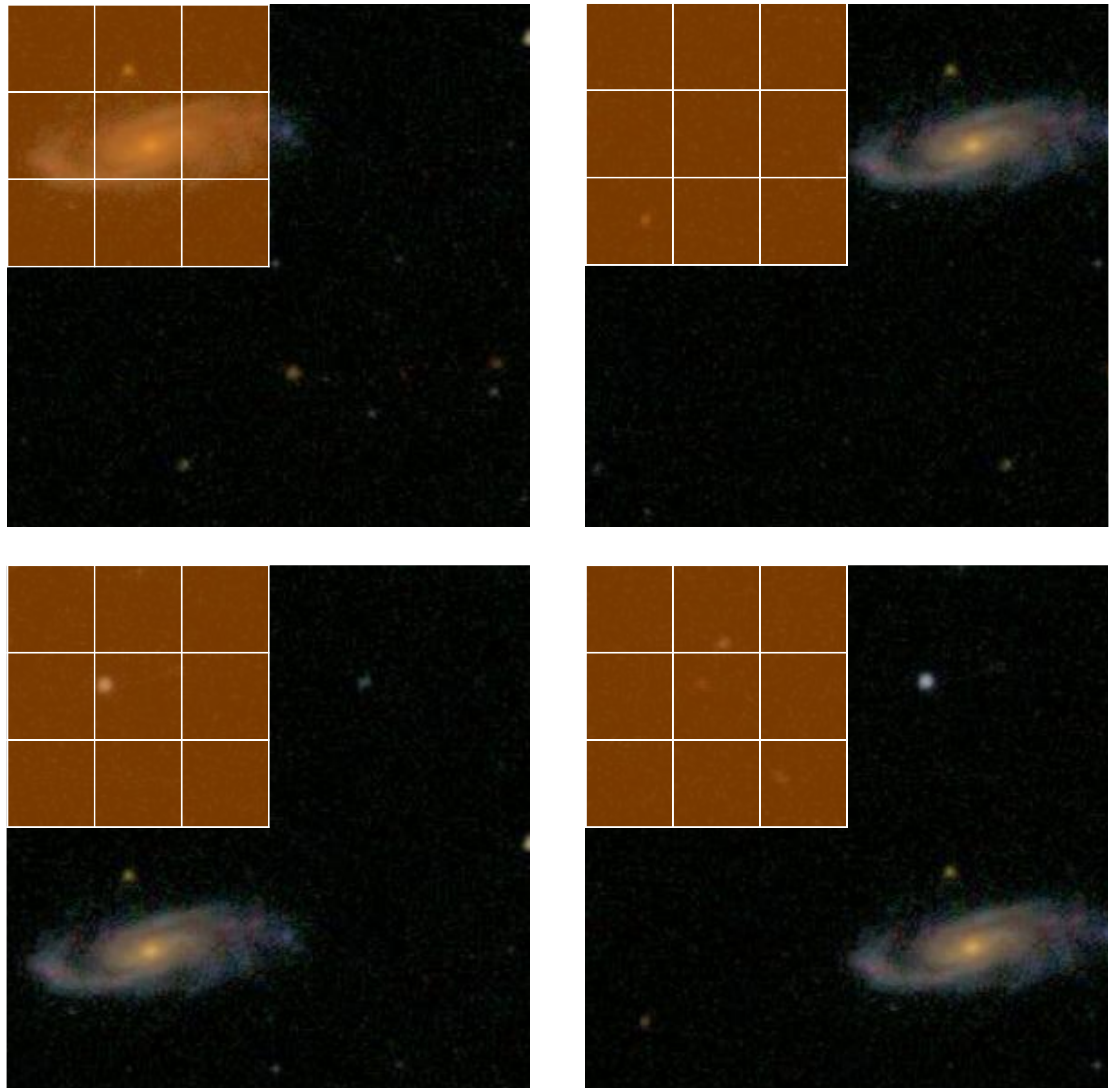


fully-connected



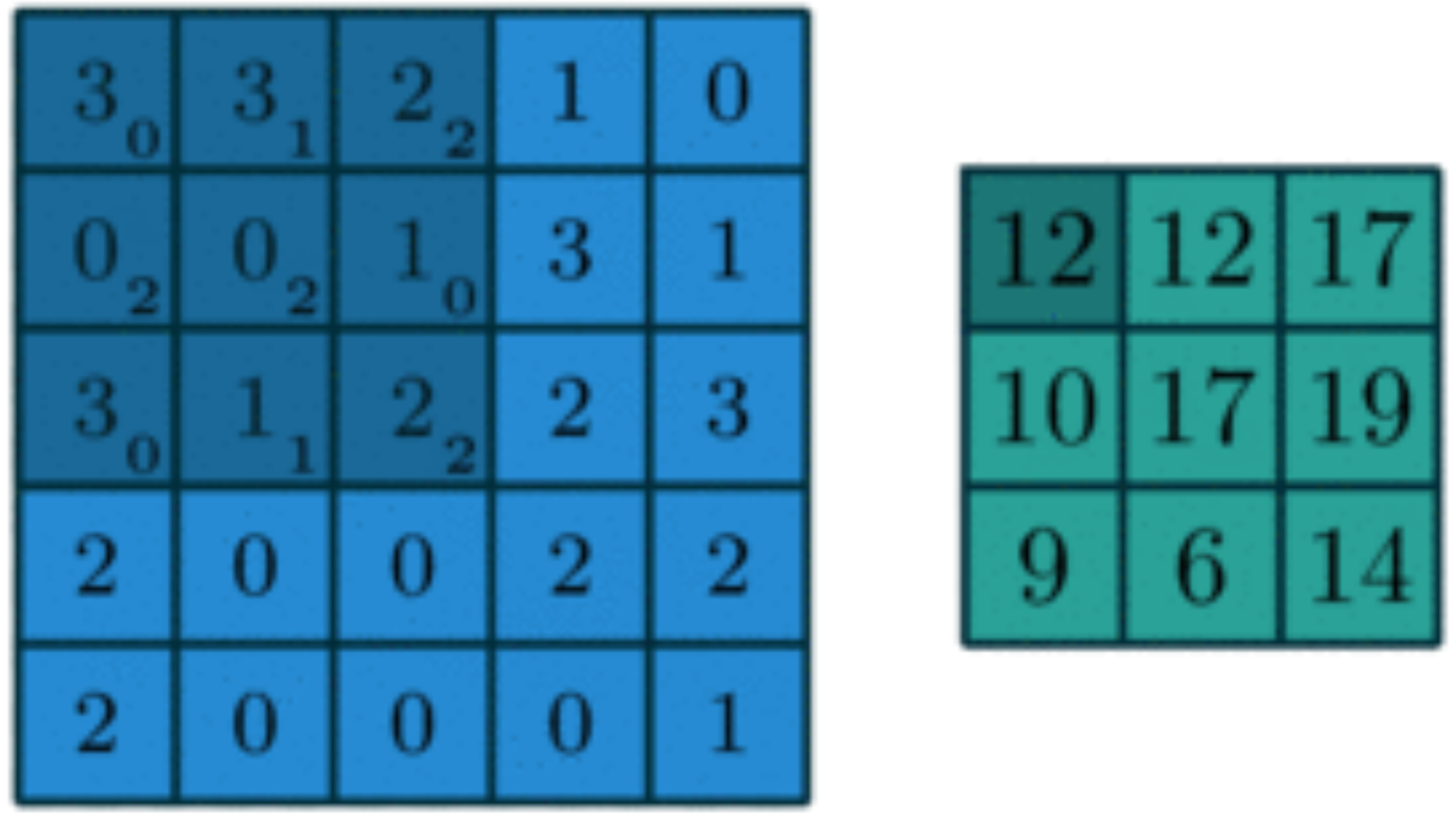
Fully-connected neural networks

What if the *same* fully-connected neural network is applied to each corner?



Convolutions

Filter weights: $\begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix}$



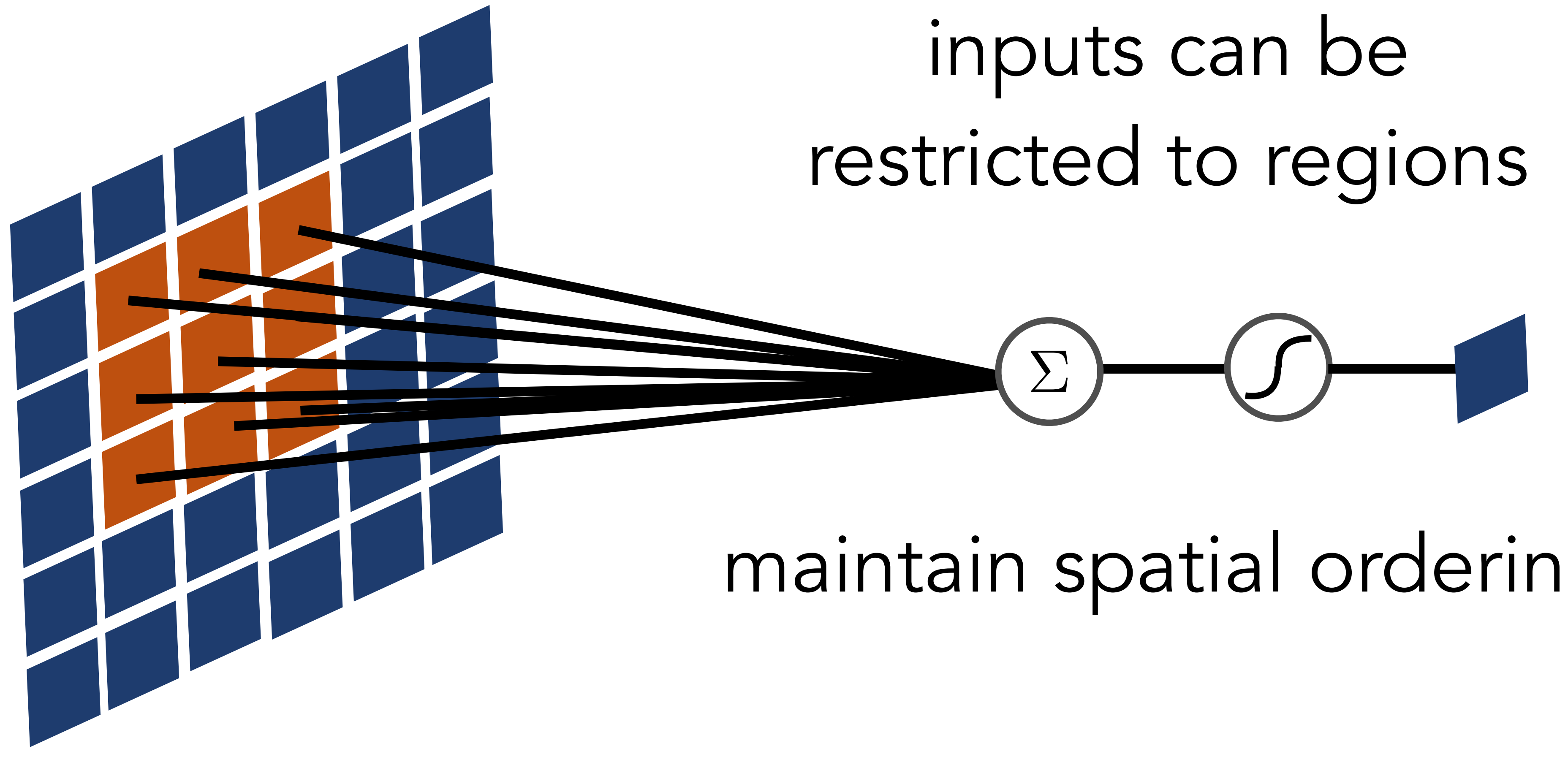
Theorem: the only **linear** and **translation-equivariant** operations are **convolutions**

Locality and translation invariance

Let's convert **locality** and **translation invariance** into *inductive biases*

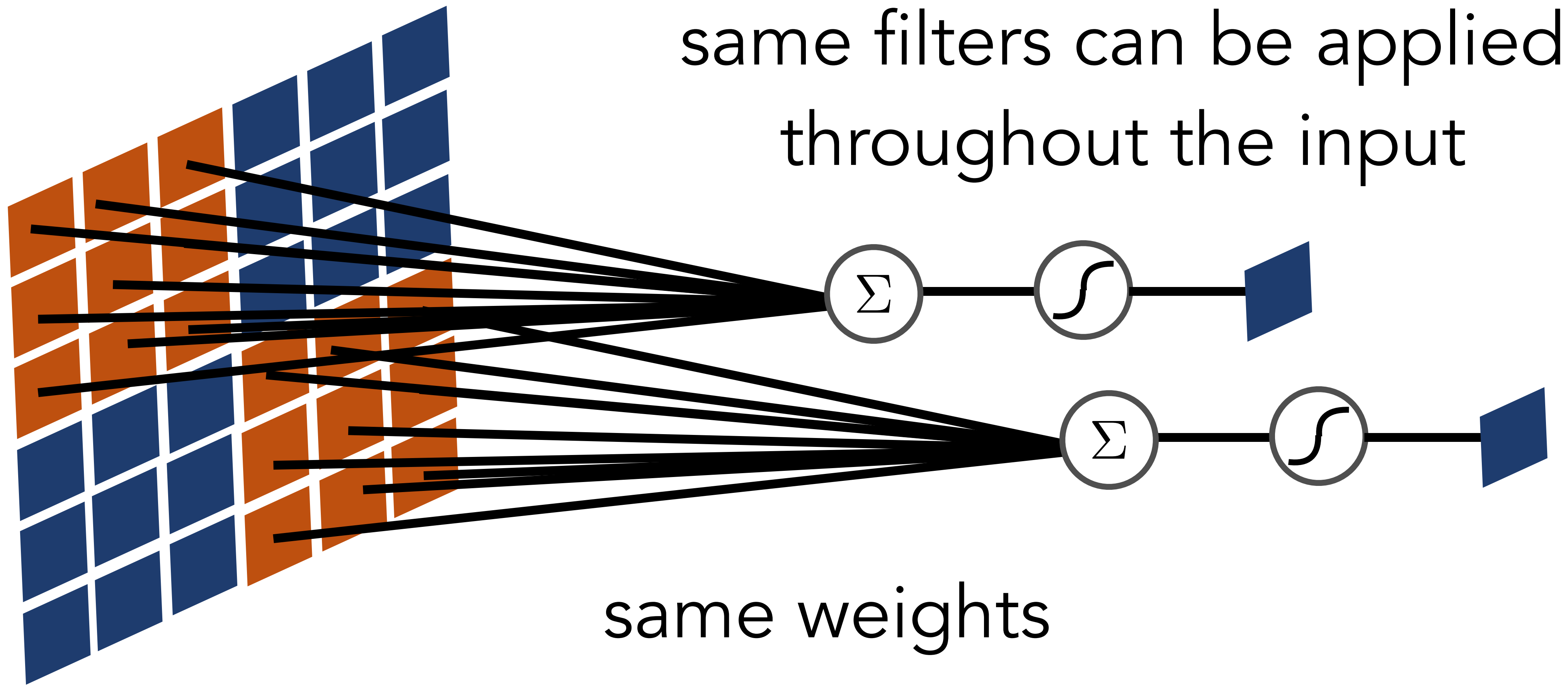
locality

nearby areas tend to contain stronger patterns



translation invariance

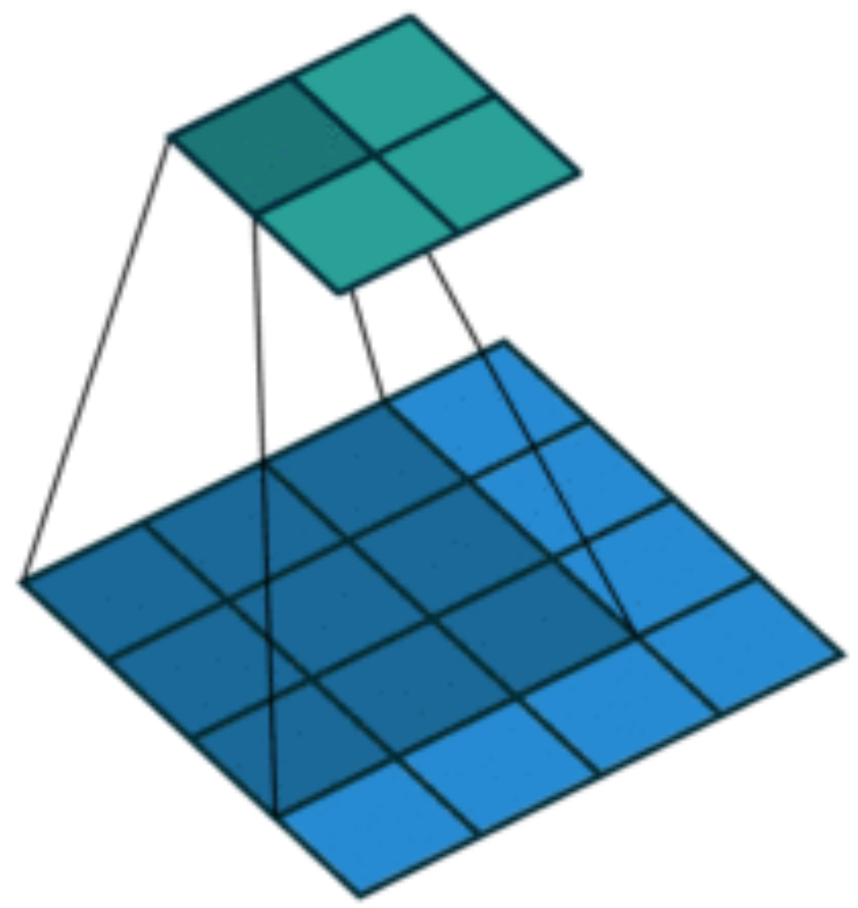
relative positions are relevant



2D convolution hyperparameters

- 4×4 input
- 3×3 filter
- 1×1 stride
- No zero padding

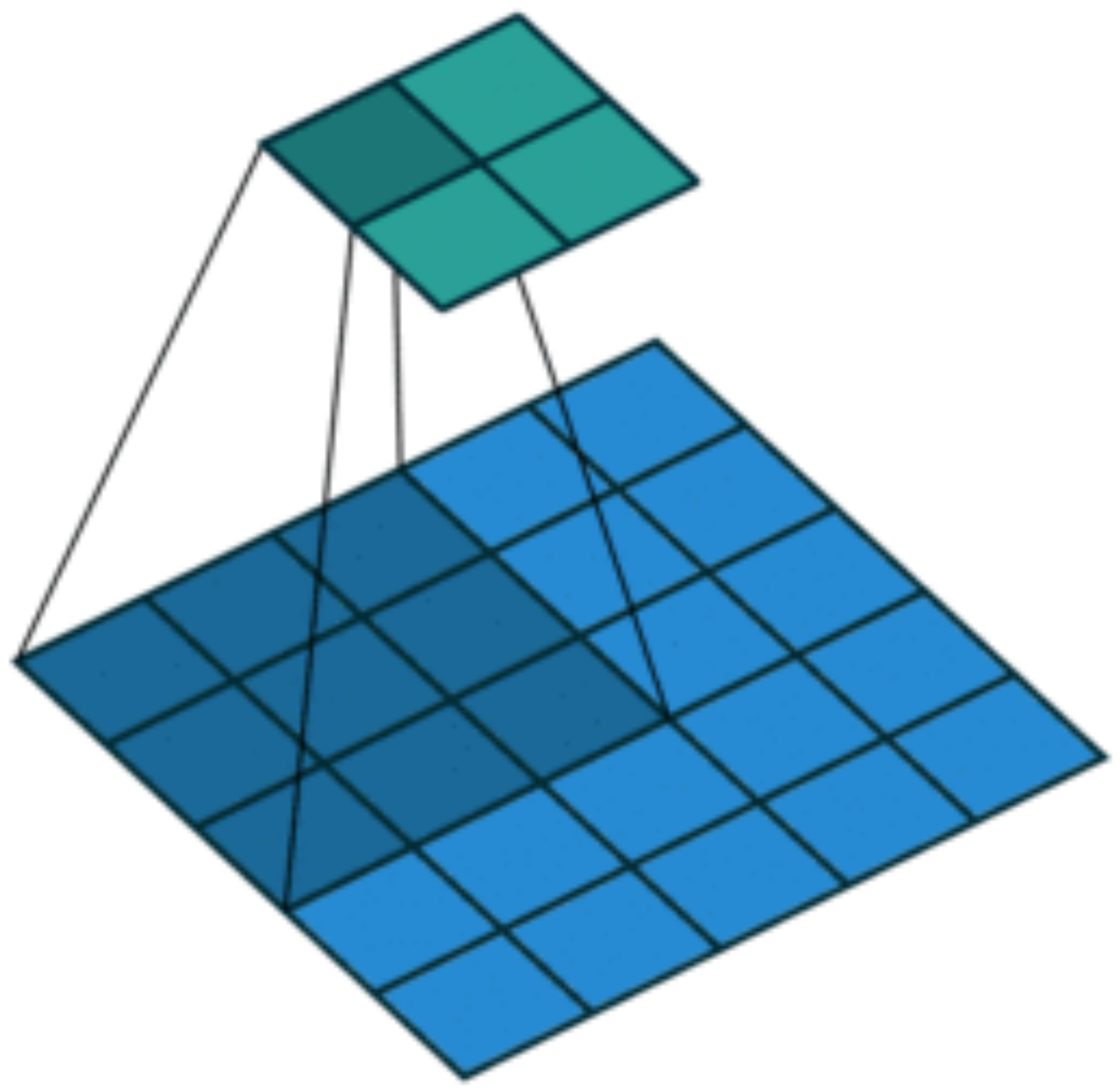
➡ 2×2 output



2D convolution hyperparameters

- 5×5 input
- 3×3 filter
- 2×2 stride
- No zero padding

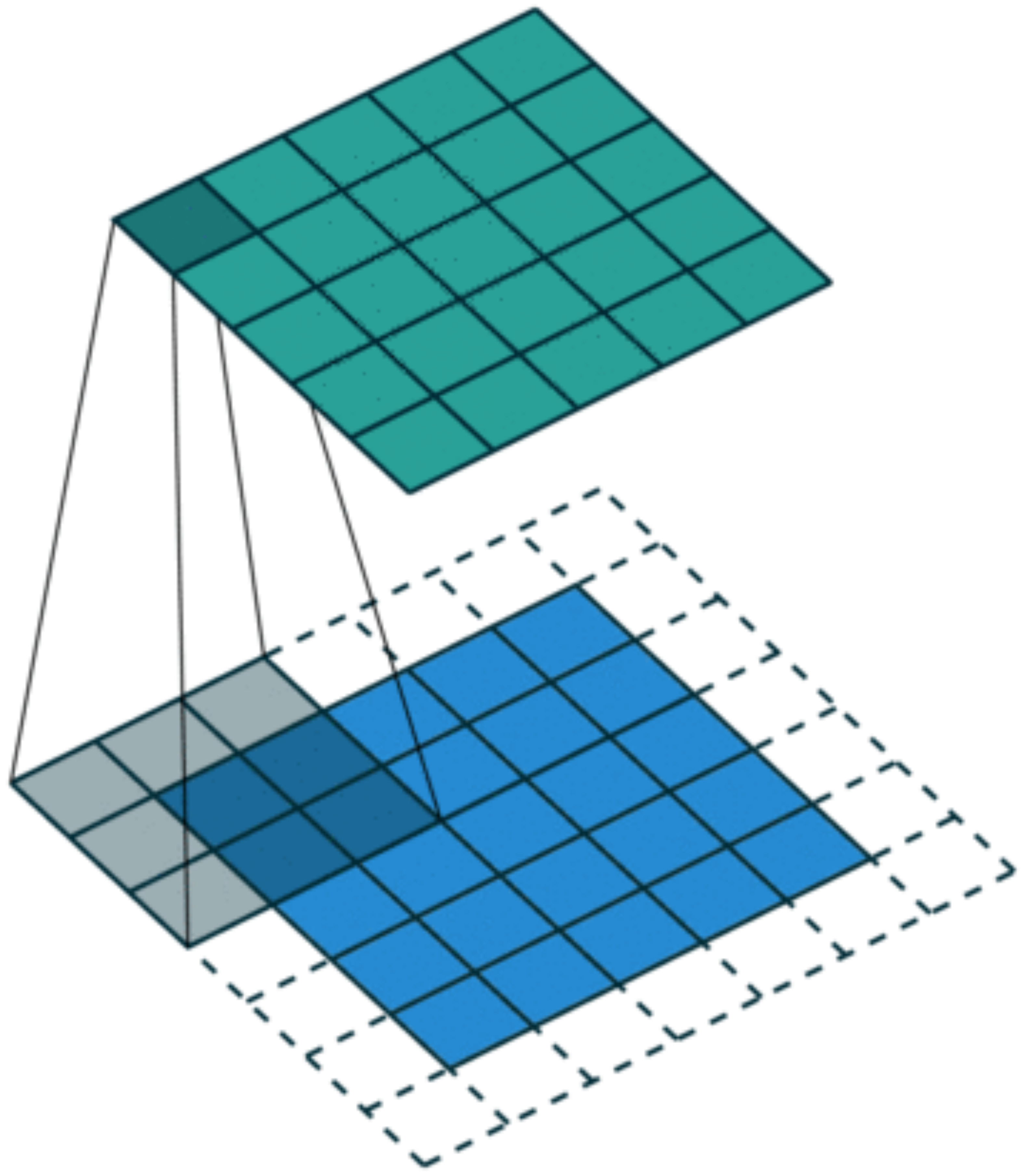
➡ 2×2 output



2D convolution hyperparameters

- 5×5 input
- 3×3 filter
- 1×1 stride
- “Same” zero padding

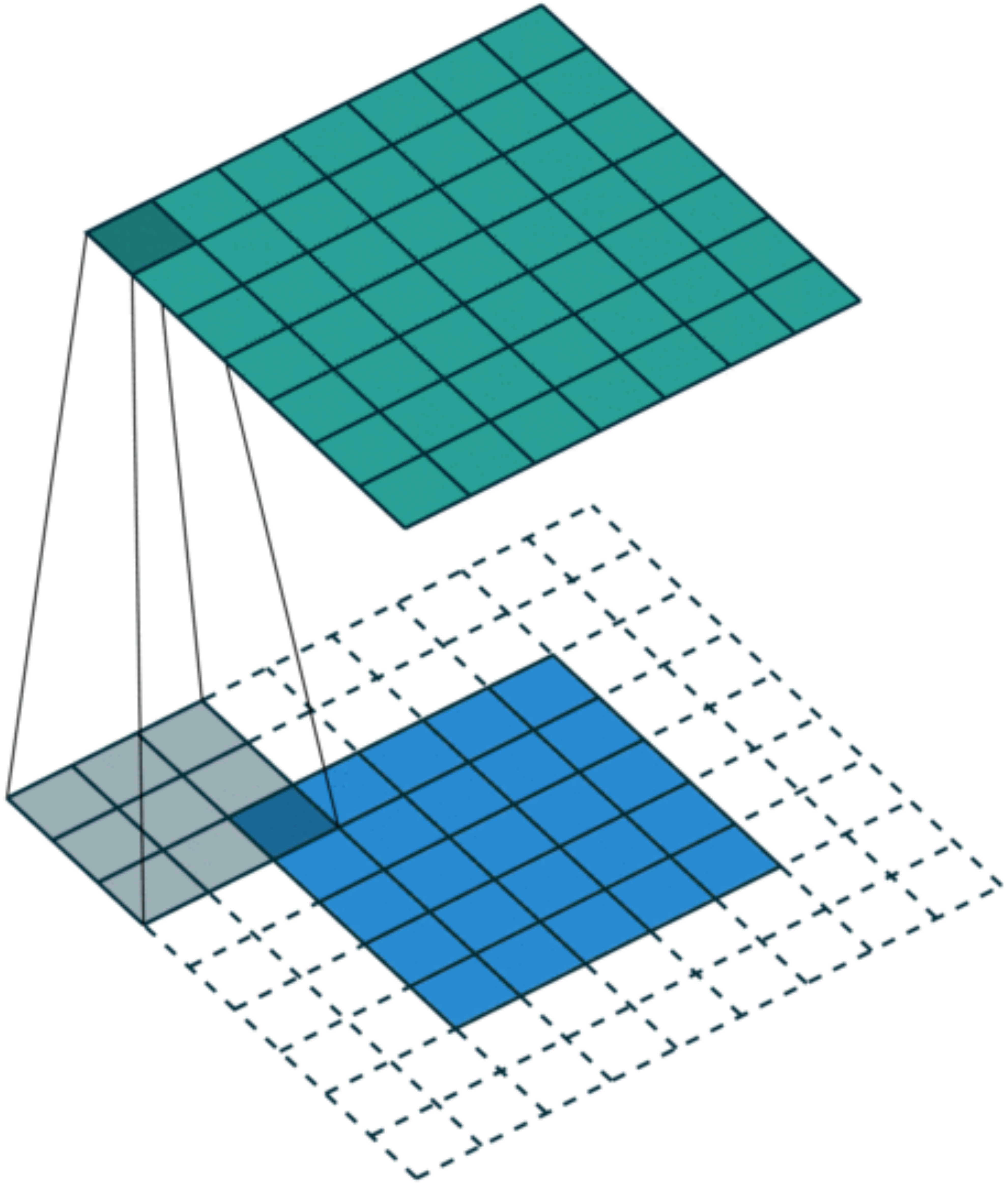
➡ 5×5 output



2D convolution hyperparameters

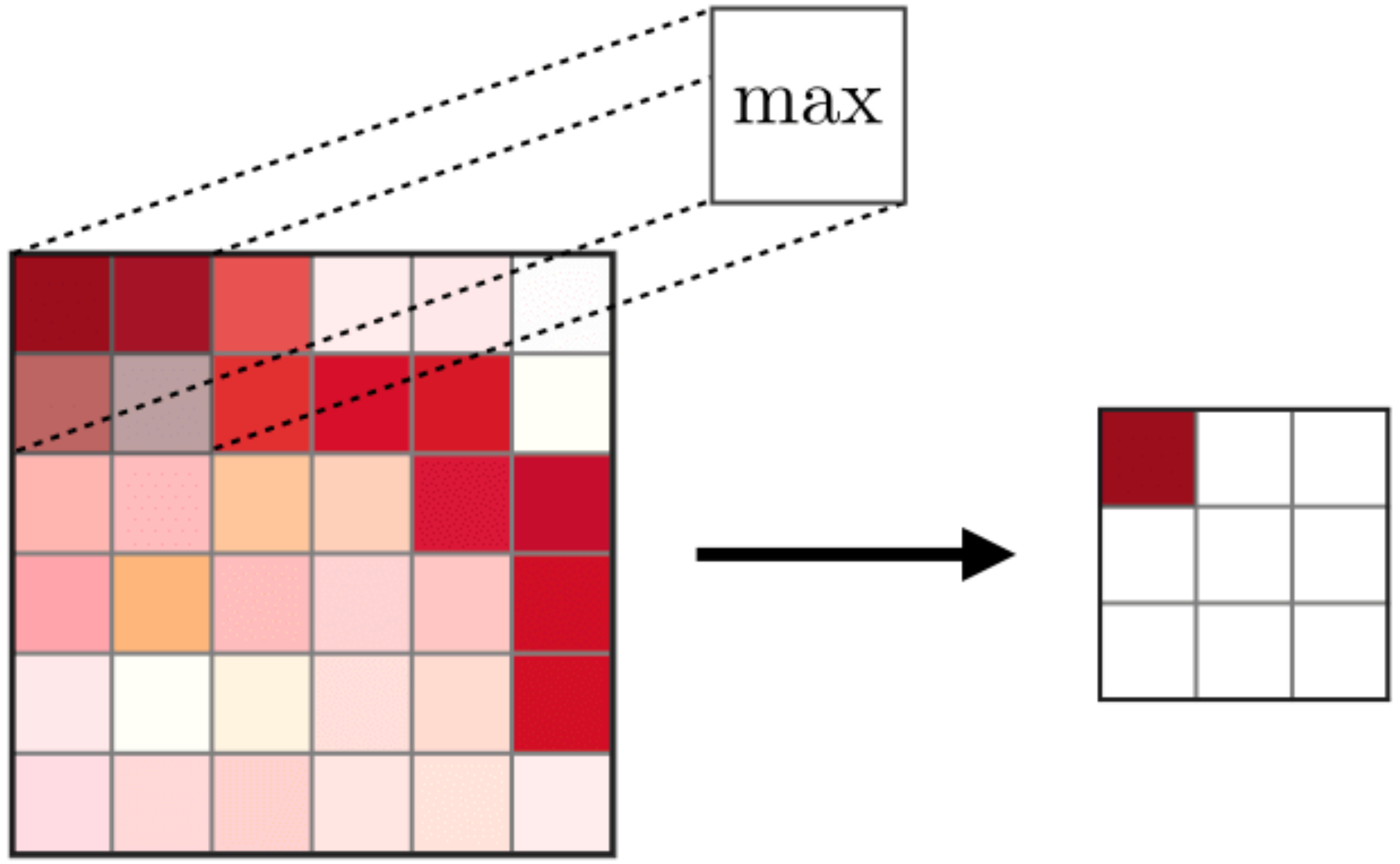
- 5×5 input
- 3×3 filter
- 1×1 stride
- “Full” zero padding

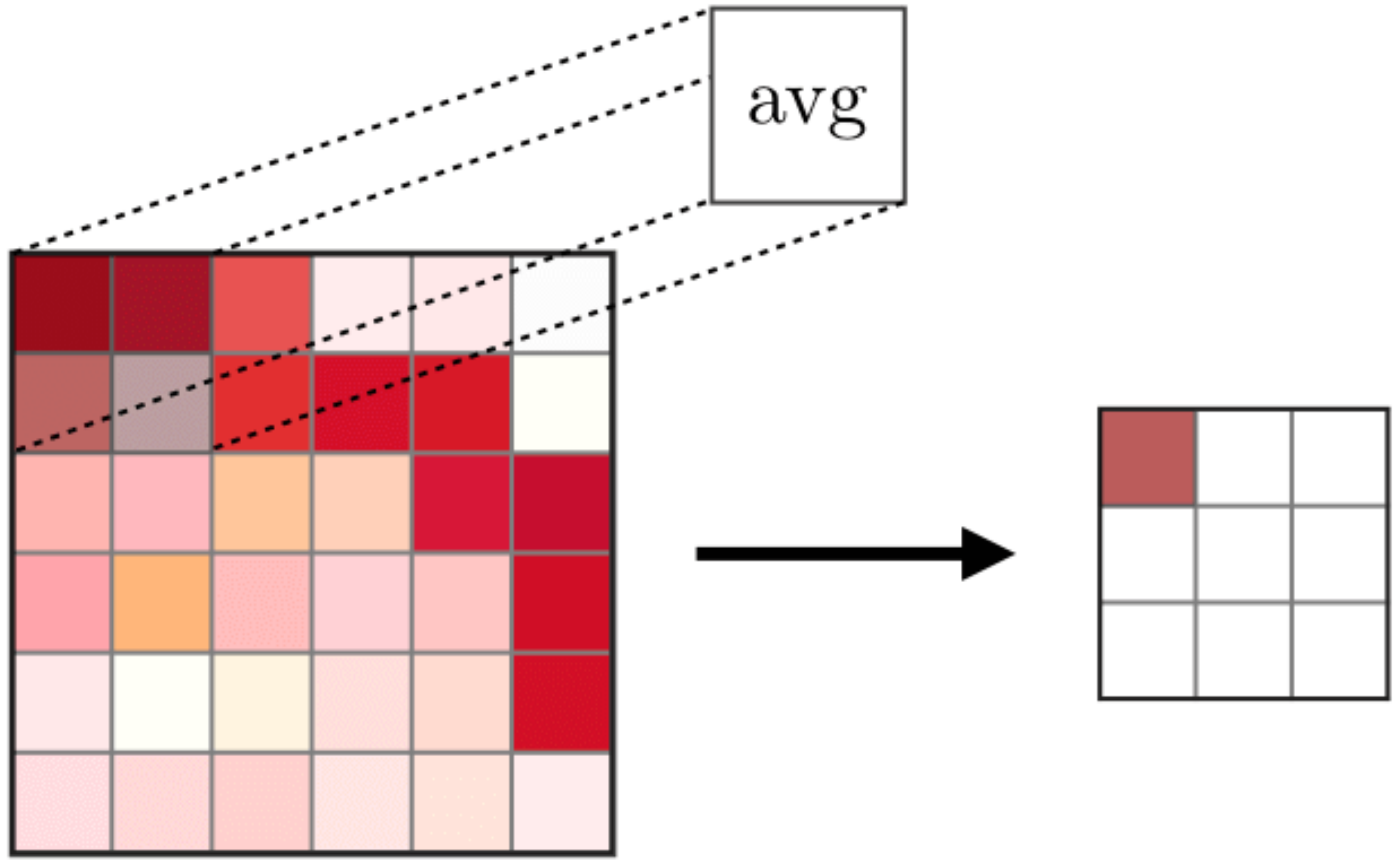
➡ 7×7 output



Pooling

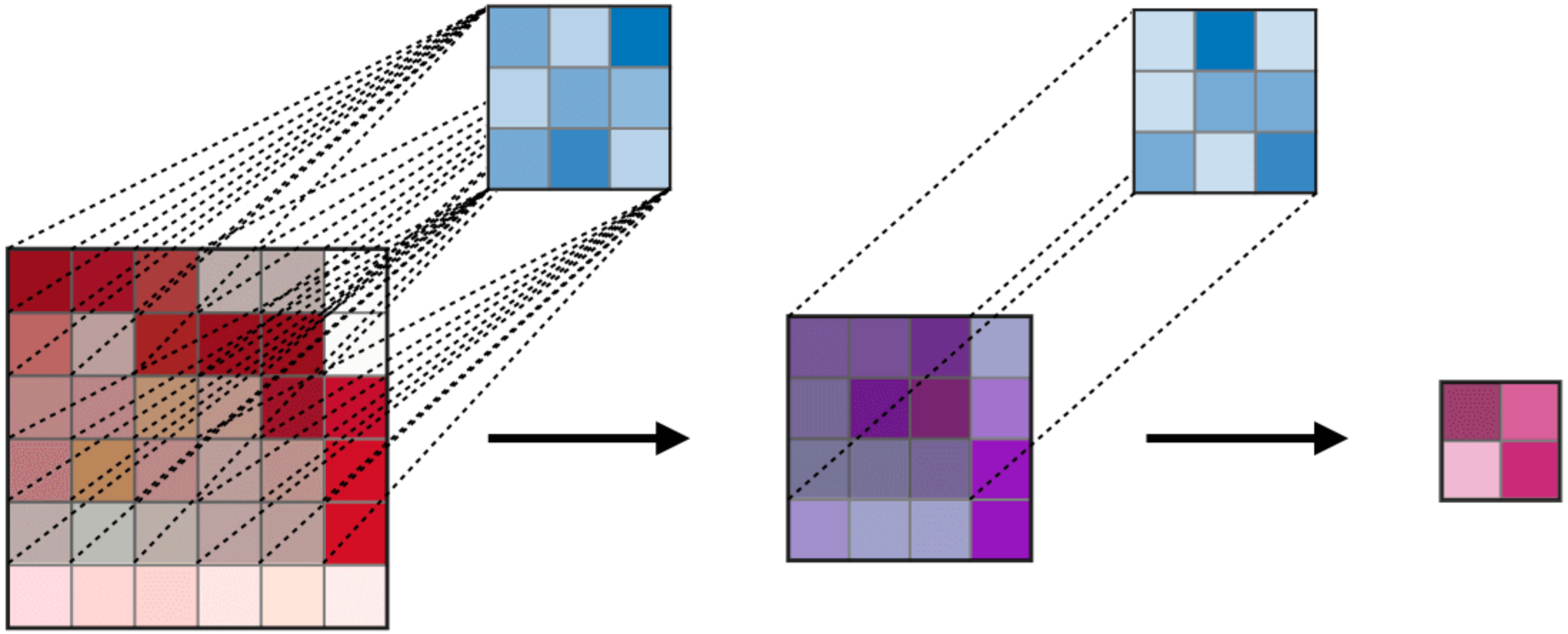
- Pooling: downsampling operation, typically applied after a convolution layer
- Max: Each pooling operation selects the maximum value of the current view
  - Most commonly used
- Average: Each pooling operation averages the values of the current view
  - “Smooths” image (may be undesirable); may better preserve information





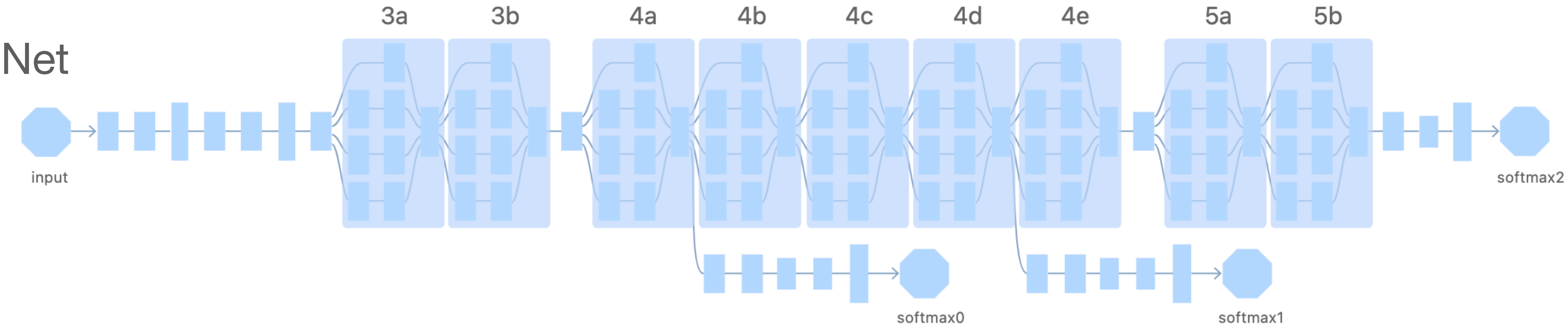
Receptive field

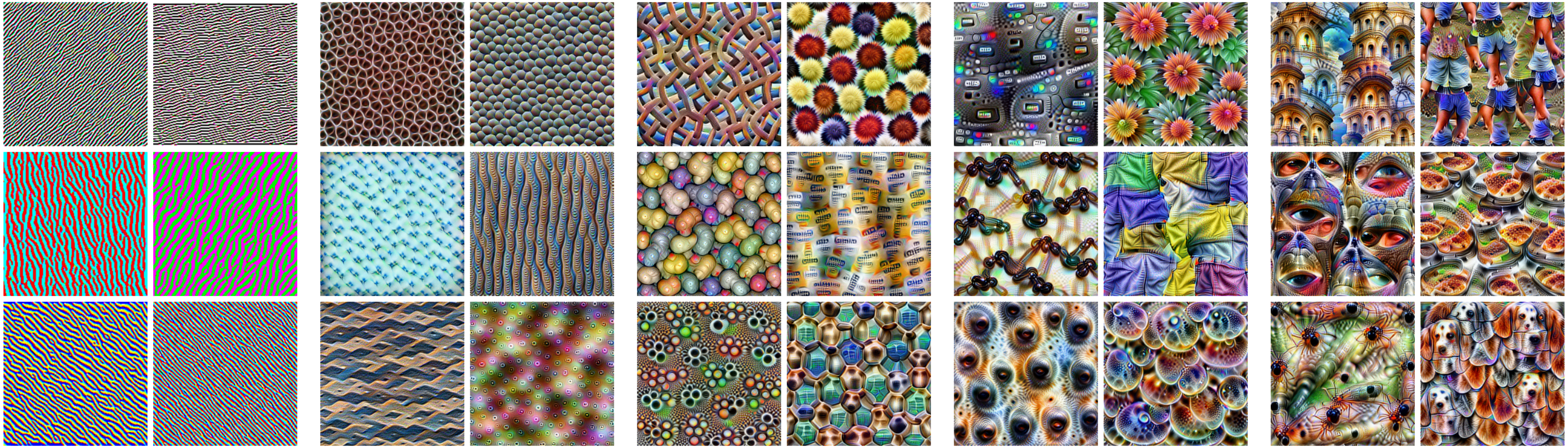
- The receptive field at layer k is the area of the input that each pixel of the kth activation map can "see"



Feature visualization

GoogLeNet



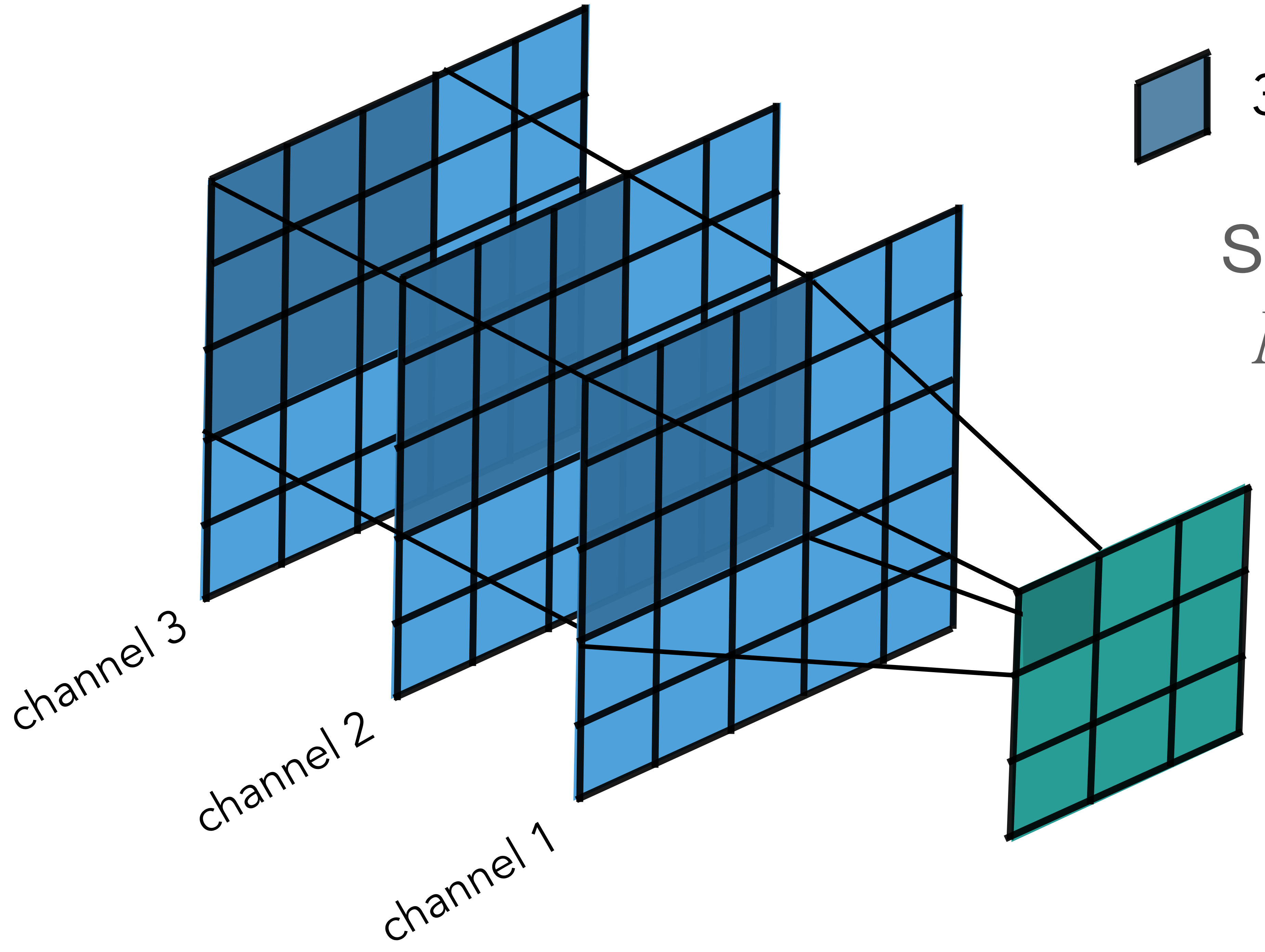


Edges (layer conv2d0) **Textures** (layer mixed3a) **Patterns** (layer mixed4a) **Parts** (layers mixed4b & mixed4c) **Objects** (layers mixed4d & mixed4e)

https://distill.pub/2017/feature-visualization/

Convolutions with multiple channels

filters are applied to all input channels



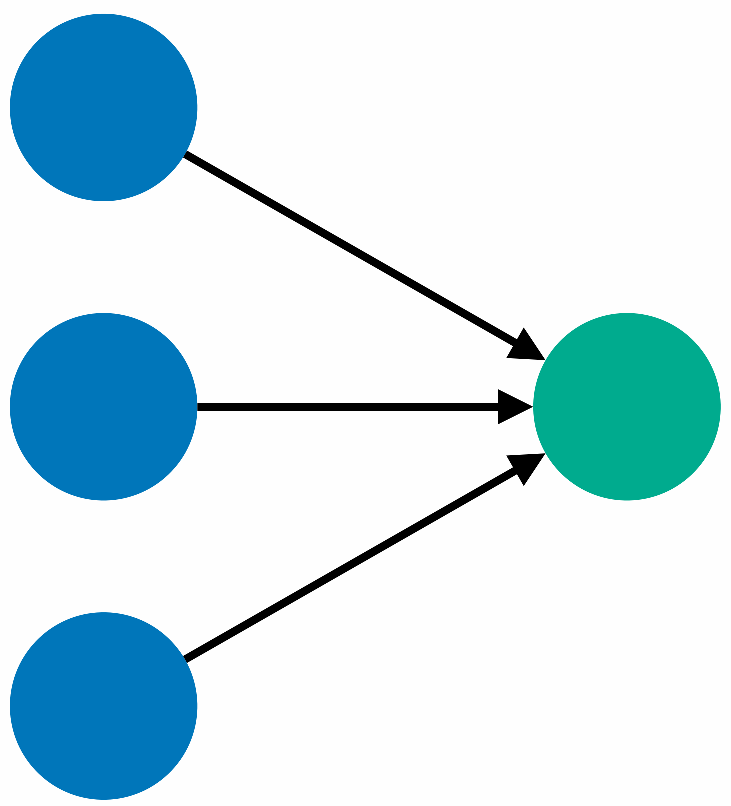
Convolutional Network

3 x 3 x 3 filter *tensor*

Size of a filter is
$N_{\text{in}} \times k_h \times k_w$

Fully-connected networks

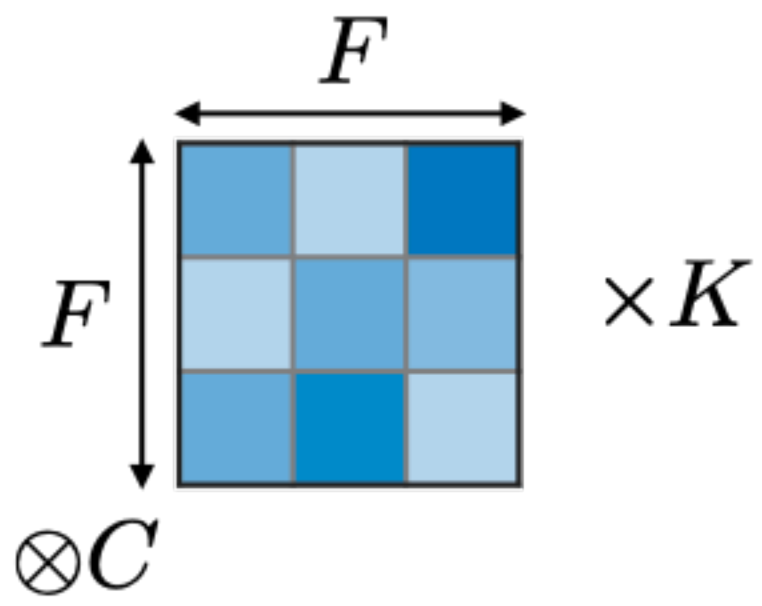
Size of a filter is N_{in}



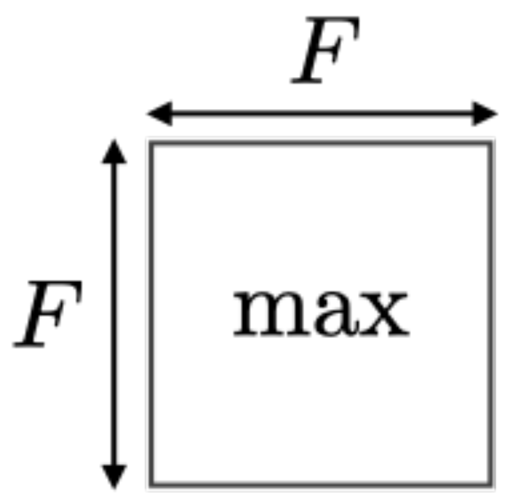
each filter results in a new output channel

How many parameters?

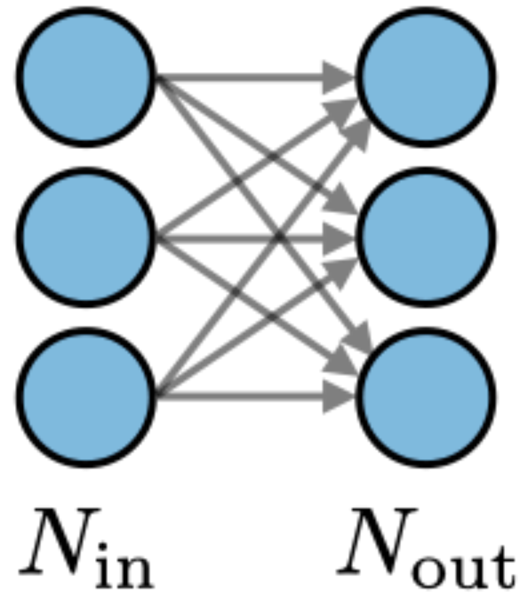
- Conv.:



- Pool.:



- FC:



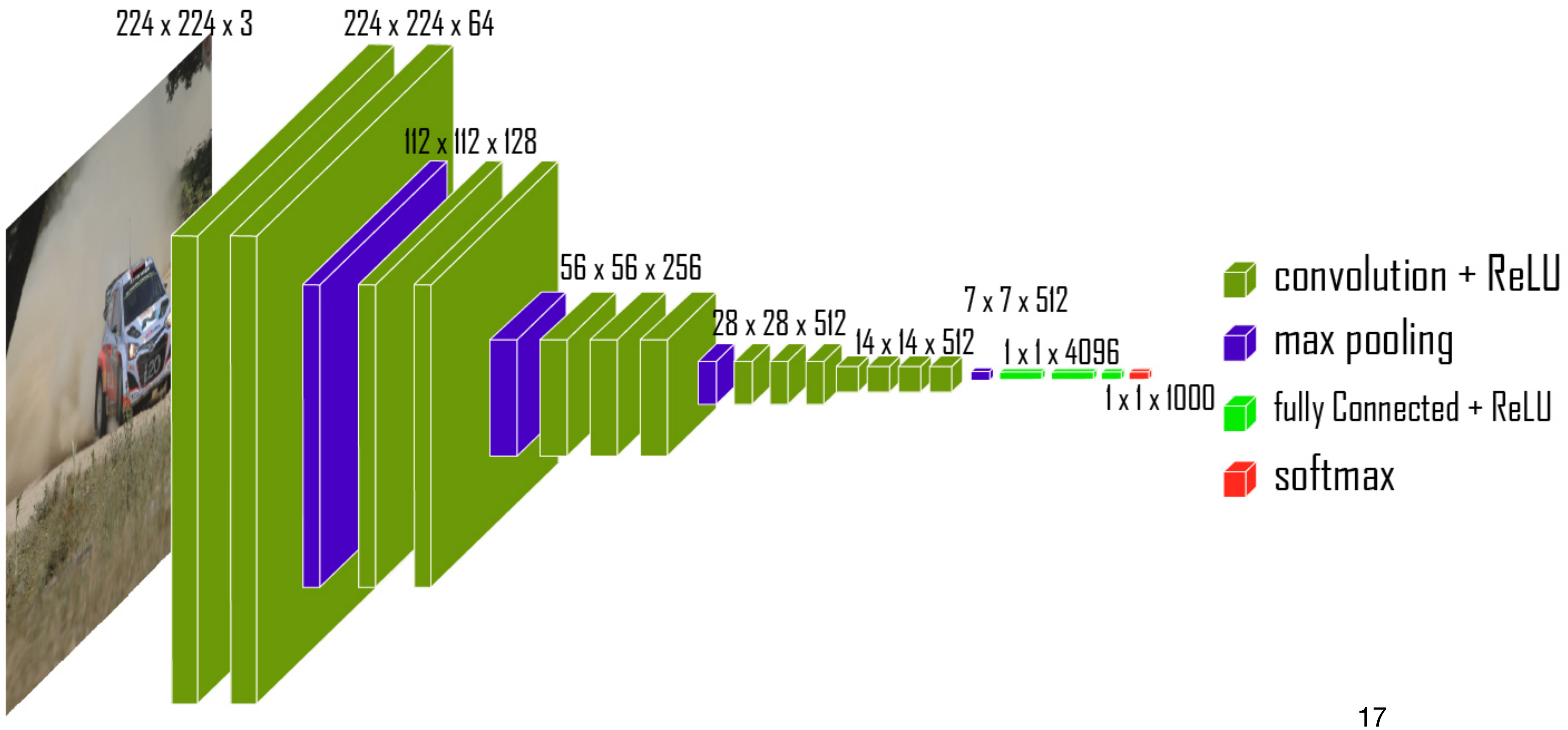
- Input: $I \times I \times C$
- Output: $O \times O \times K$
- Parameters: $(F^2 C + 1)K$

- Input: $I \times I \times C$
- Output: $O \times O \times C$
- Parameters: 0

- Input: N_{in}
- Output: N_{out}
- Parameters: $(N_{\text{in}} + 1)N_{\text{out}}$

In practice: VGG16

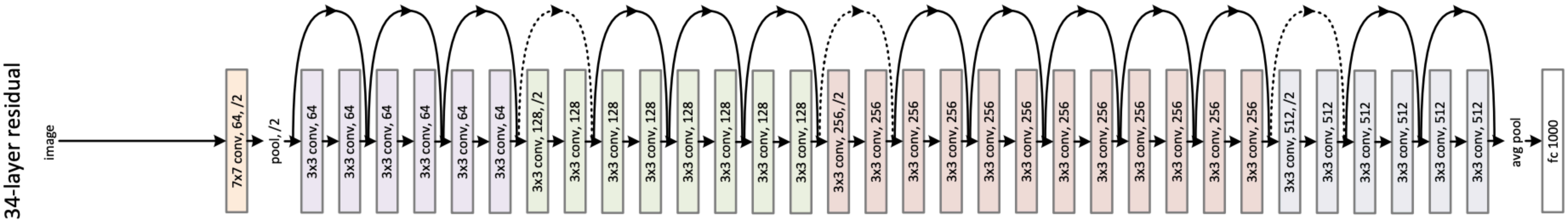
- VGG16 [arXiv:1409.1556] is a classic deep CNN
- 1st runner up 2014 ImageNet Large Scale Visual Recognition Competition (ILSVRC)




```python
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.models import Model

# Input
img_input = Input(shape=(224, 224, 3))

# Block 1
x = Conv2D(64, (3, 3), activation="relu", padding="same", name="block1_conv1")(
    img_input
)
x = Conv2D(64, (3, 3), activation="relu", padding="same", name="block1_conv2")(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name="block1_pool")(x)

# Block 2
x = Conv2D(128, (3, 3), activation="relu", padding="same", name="block2_conv1")(x)
x = Conv2D(128, (3, 3), activation="relu", padding="same", name="block2_conv2")(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name="block2_pool")(x)

# Block 3
x = Conv2D(256, (3, 3), activation="relu", padding="same", name="block3_conv1")(x)
x = Conv2D(256, (3, 3), activation="relu", padding="same", name="block3_conv2")(x)
x = Conv2D(256, (3, 3), activation="relu", padding="same", name="block3_conv3")(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name="block3_pool")(x)

# Block 4
x = Conv2D(512, (3, 3), activation="relu", padding="same", name="block4_conv1")(x)
x = Conv2D(512, (3, 3), activation="relu", padding="same", name="block4_conv2")(x)
x = Conv2D(512, (3, 3), activation="relu", padding="same", name="block4_conv3")(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name="block4_pool")(x)

# Block 5
x = Conv2D(512, (3, 3), activation="relu", padding="same", name="block5_conv1")(x)
x = Conv2D(512, (3, 3), activation="relu", padding="same", name="block5_conv2")(x)

# Classification block
x = Flatten(name="flatten")(x)
x = Dense(4096, activation="relu", name="fc1")(x)
x = Dense(4096, activation="relu", name="fc2")(x)

x = Dense(1000, activation="softmax", name="predictions")(x)

model = Model(inputs=img_input, outputs=x, name="vgg16")
```

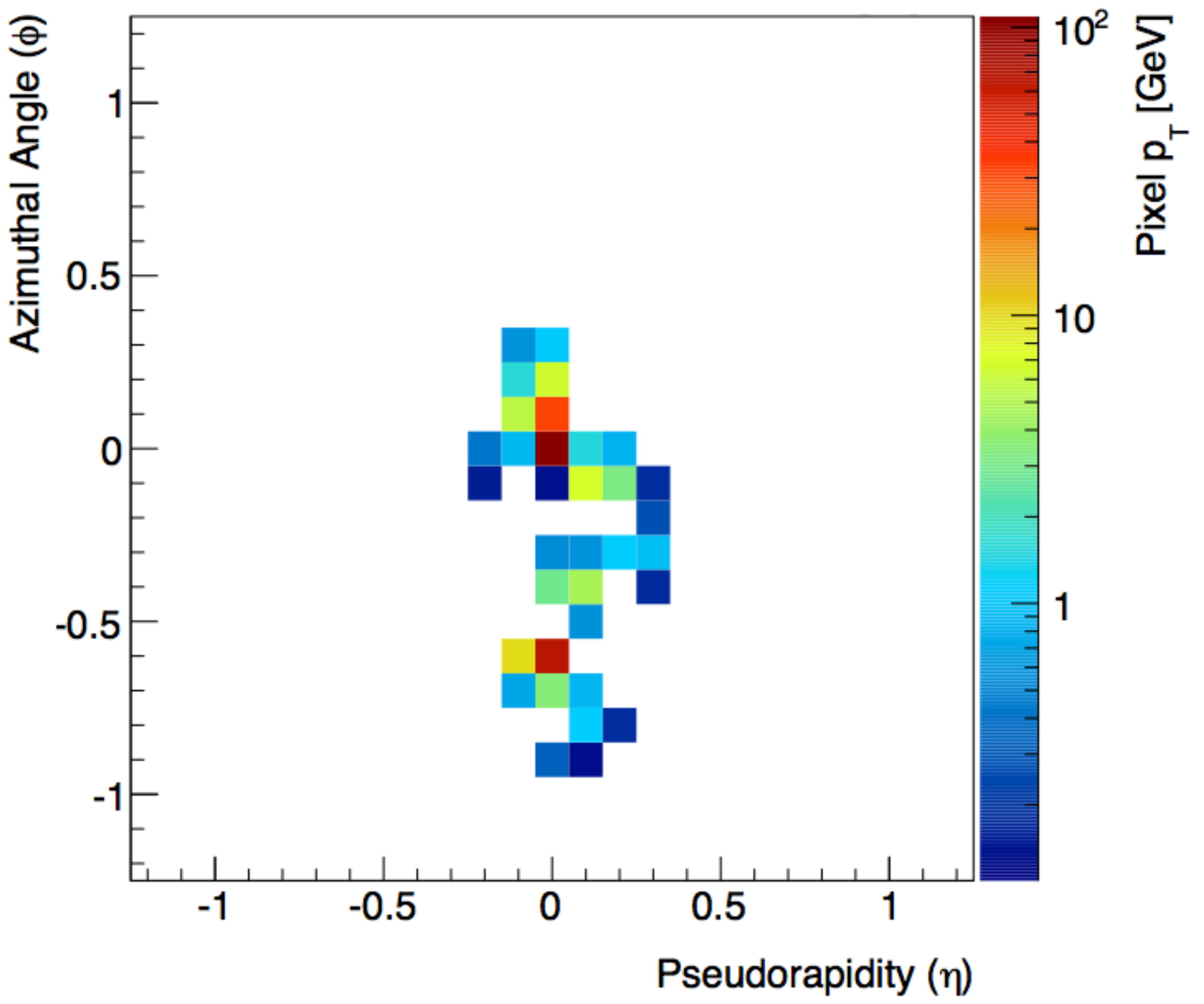
In practice: ResNet-34

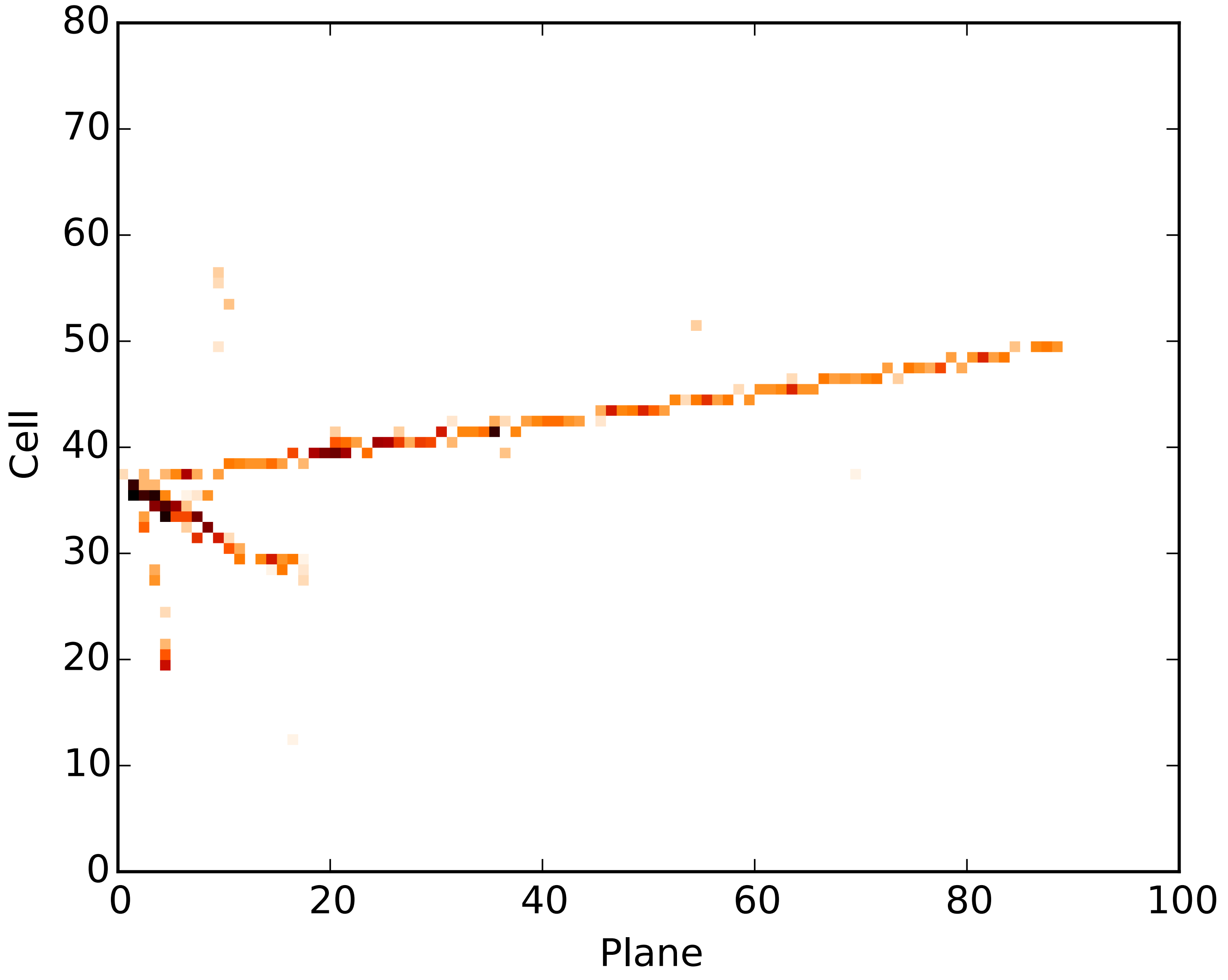
- Use skip connections to make CNN even deeper!
- ResNet-50 still used for many purposes

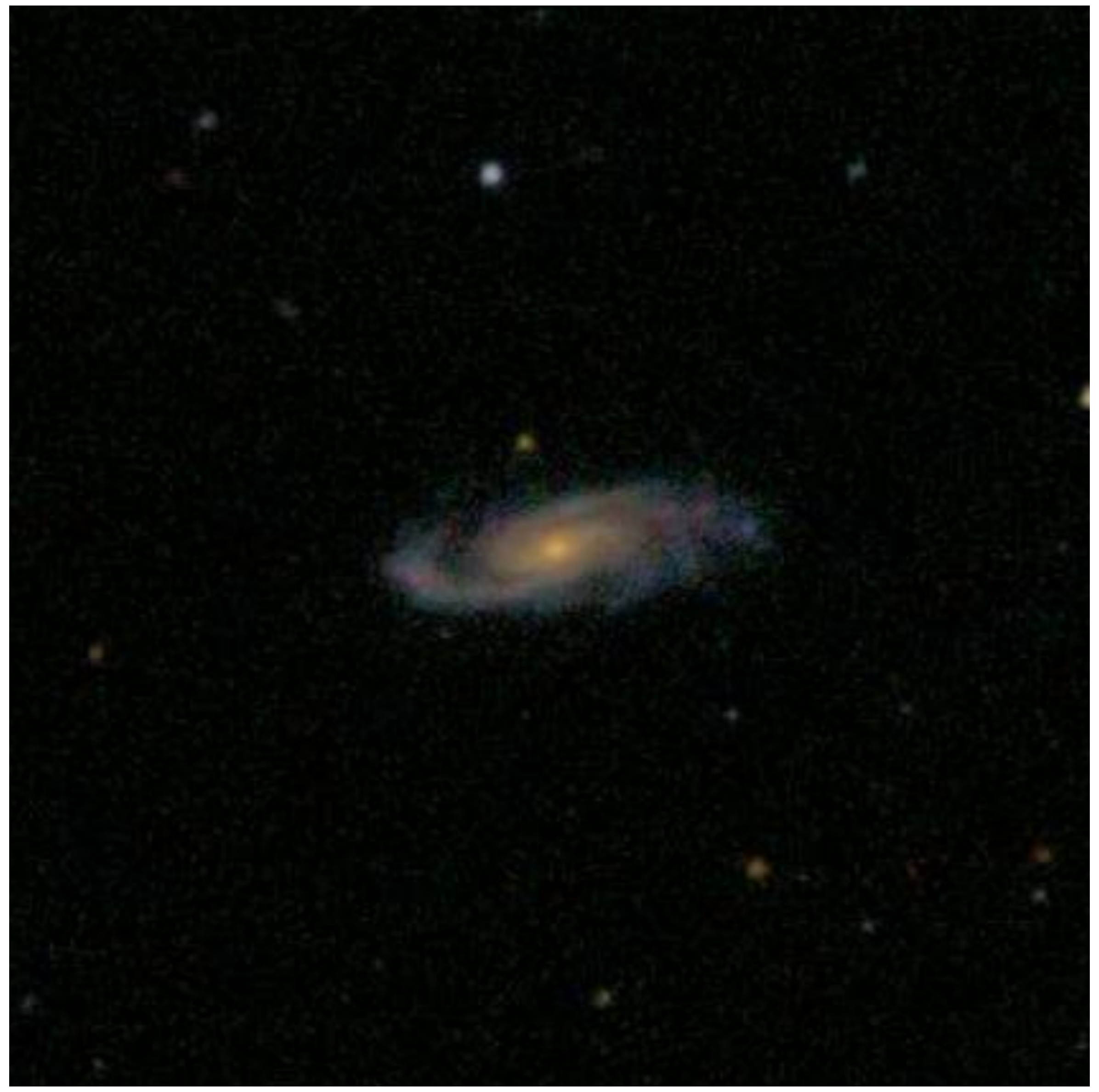


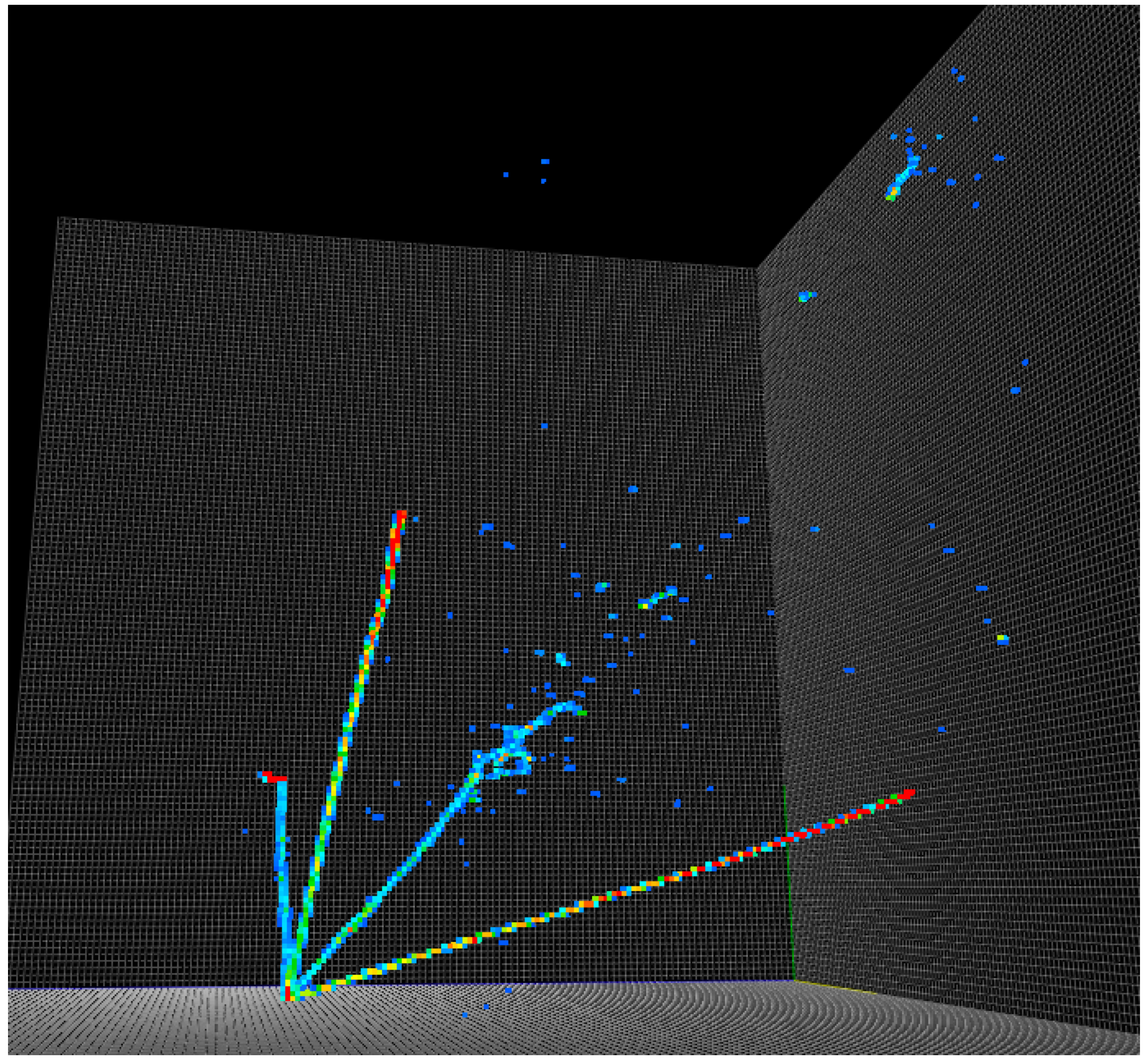
Sparsity

- In physics, image data is often *sparse* (mostly empty)
  - Even worse for 3D data
- How can we deal with this efficiently?



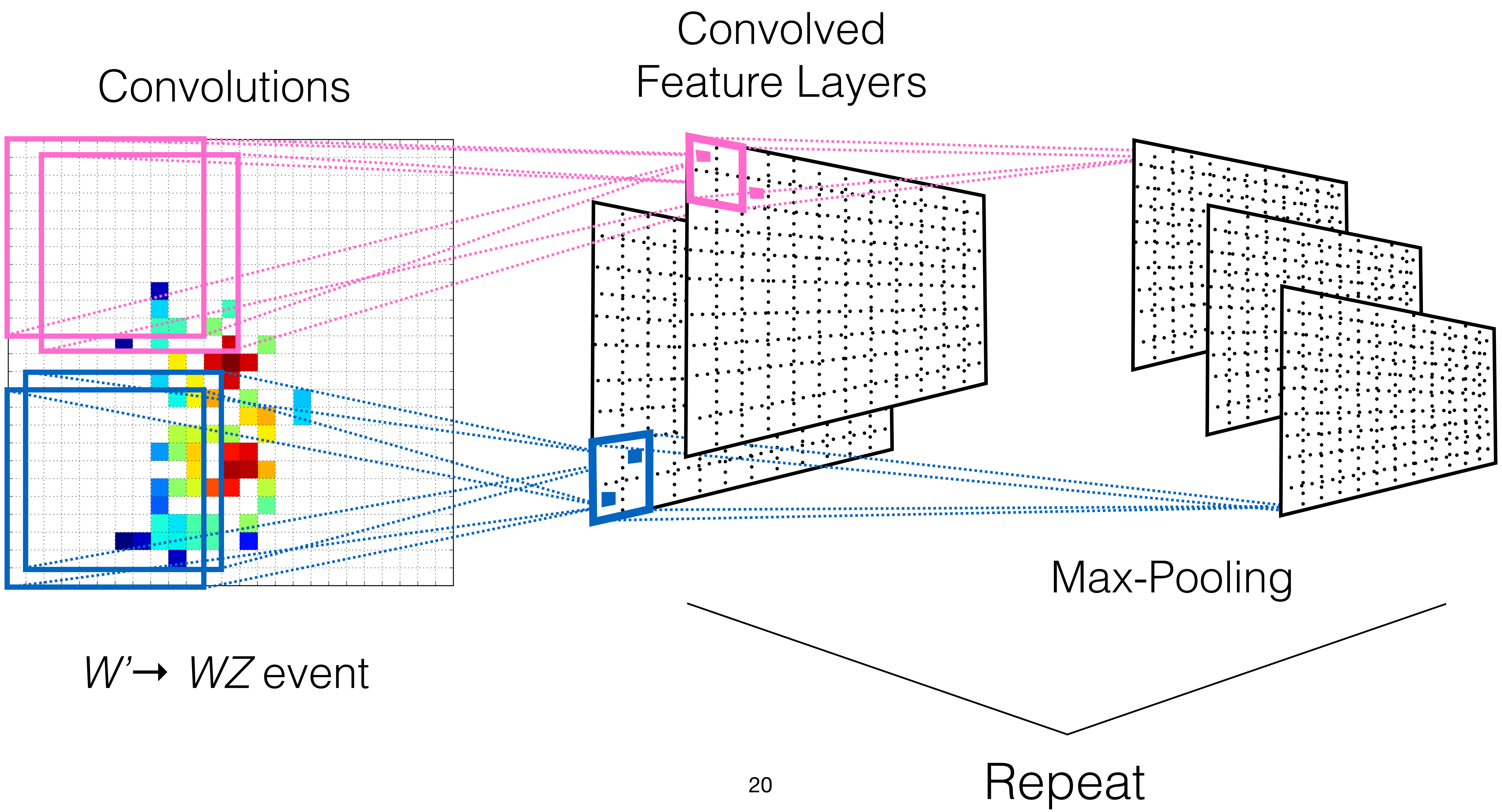






Simple: use larger filters

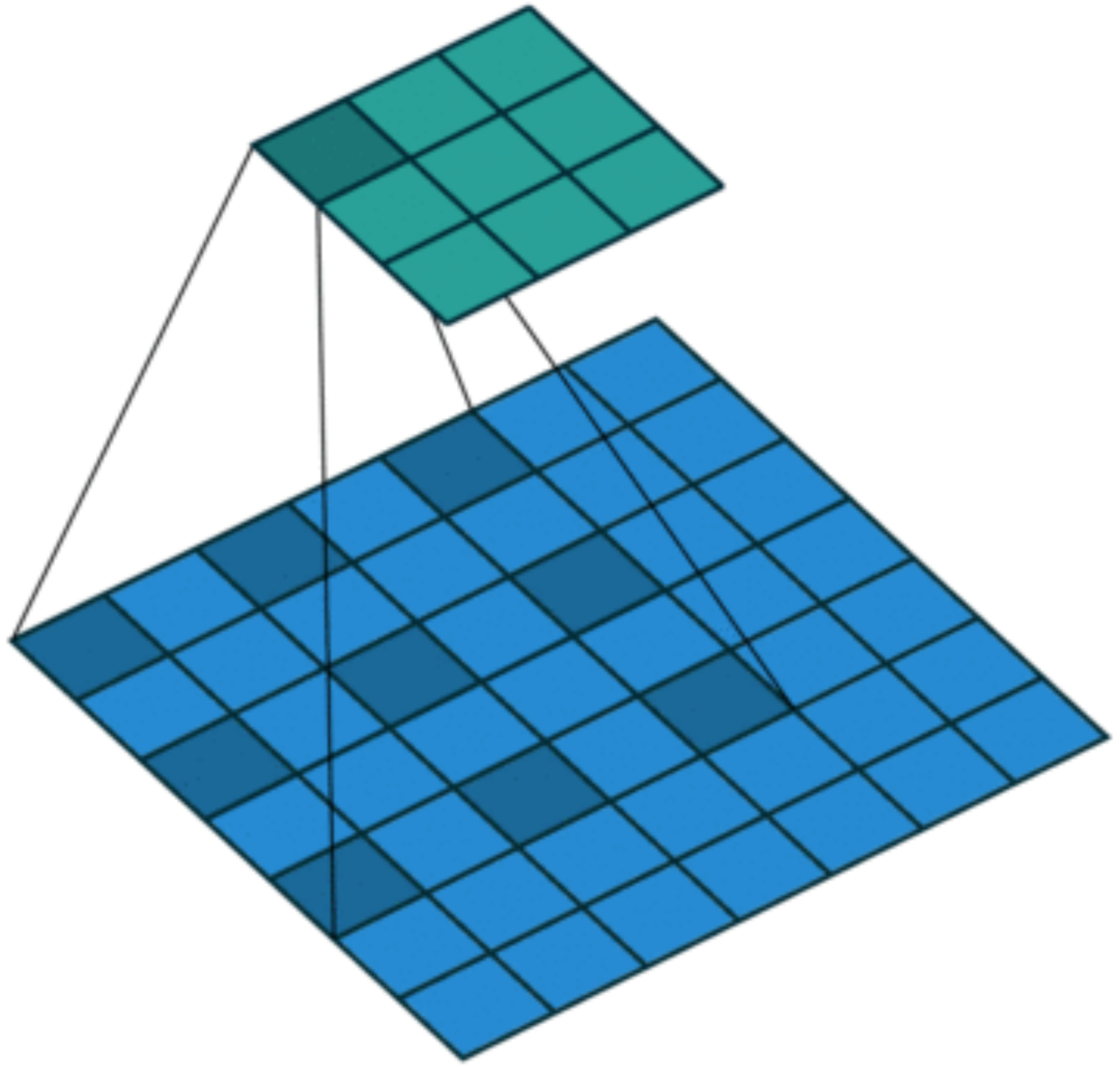
- While 3×3 filters or smaller are common for natural images, often need wider filters, e.g. 11×11, for sparse images



More exotic: dilated convolution

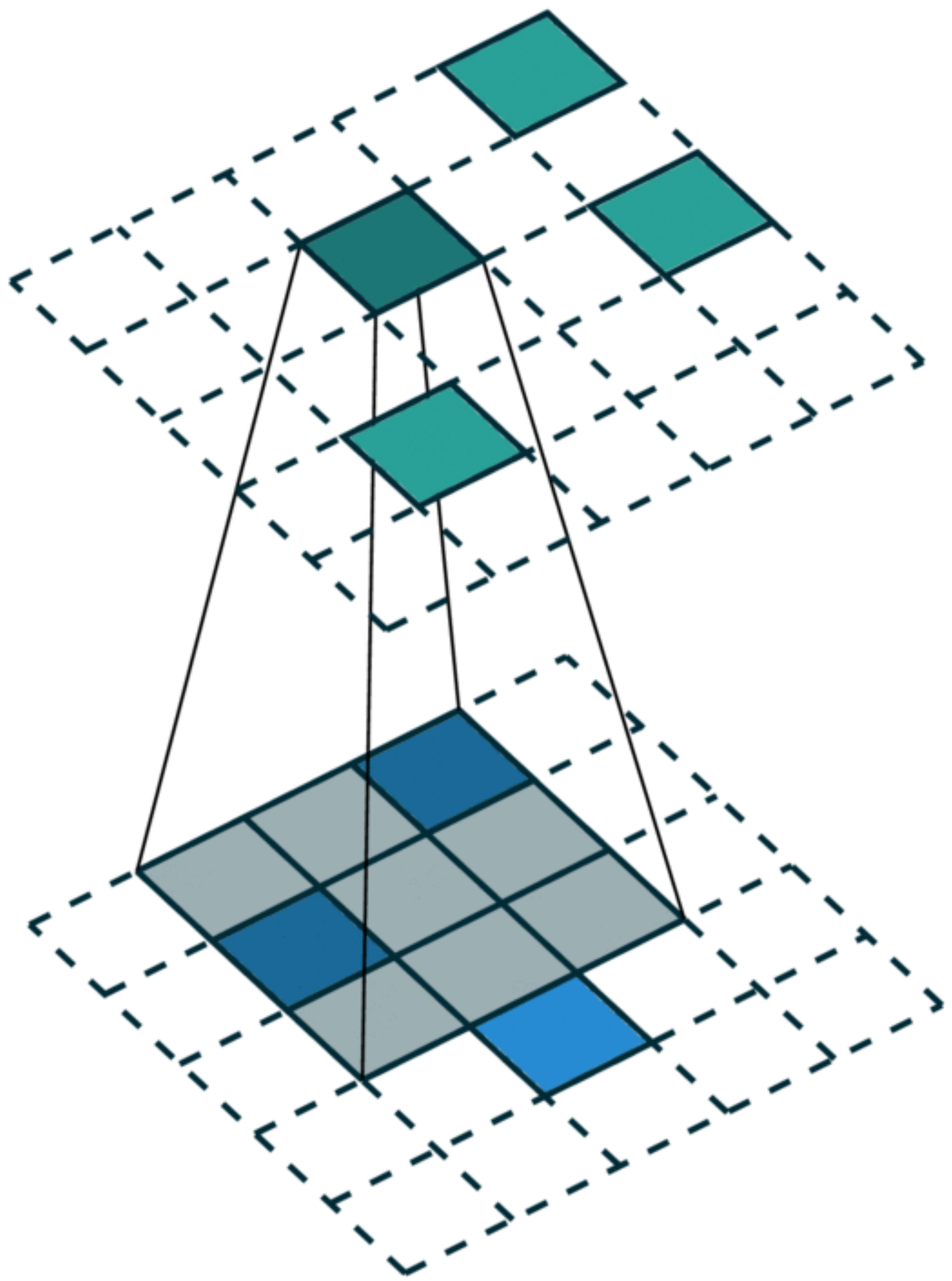
- 7×7 input
- 3×3 filter
- 1×1 stride
- No zero padding
- 2×2 dilation

➡ 3×3 output



More exotic: sparse convolution

- Uses sparse tensors as basic data representation
- Only computes convolution where output is nonzero
- Implementation: https://github.com/NVIDIA/MinkowskiEngine



Caution: Are CNNs actually translation invariant?

- Most CNNs are not ***architecturally*** invariant to translation, but they can learn to be by training on a data set that contains this regularity

- Also, architecture can be modified to be robust to small translations

Convolutional Neural Networks Are Not Invariant to Translation, but They Can Learn to Be

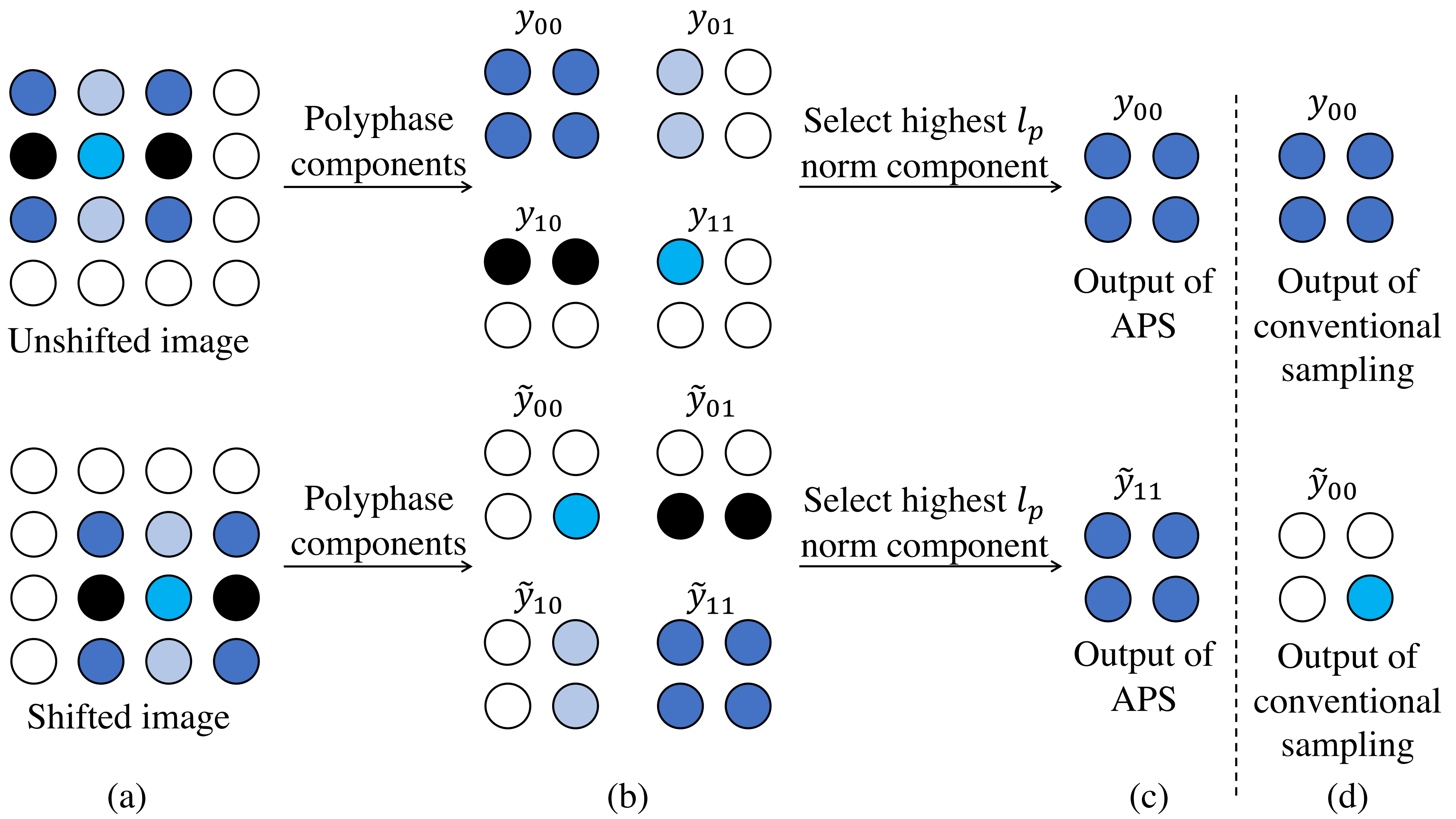
Valerio Biscione VALERIO.BISCIONE@BRISTOL.AC.UK
Department of Psychology
University of Bristol
Bristol BS8 1TL, United Kingdom

Jeffrey S. Bowers J.BOWERS@BRISTOL.AC.UK
Department of Psychology
University of Bristol
Bristol BS8 1TL, United Kingdom

Truly shift-invariant convolutional neural networks

Anadi Chaman
University of Illinois at Urbana-Champaign
achaman2@illinois.edu

Ivan Dokmanić
University of Basel
ivan.dokmanic@unibas.ch

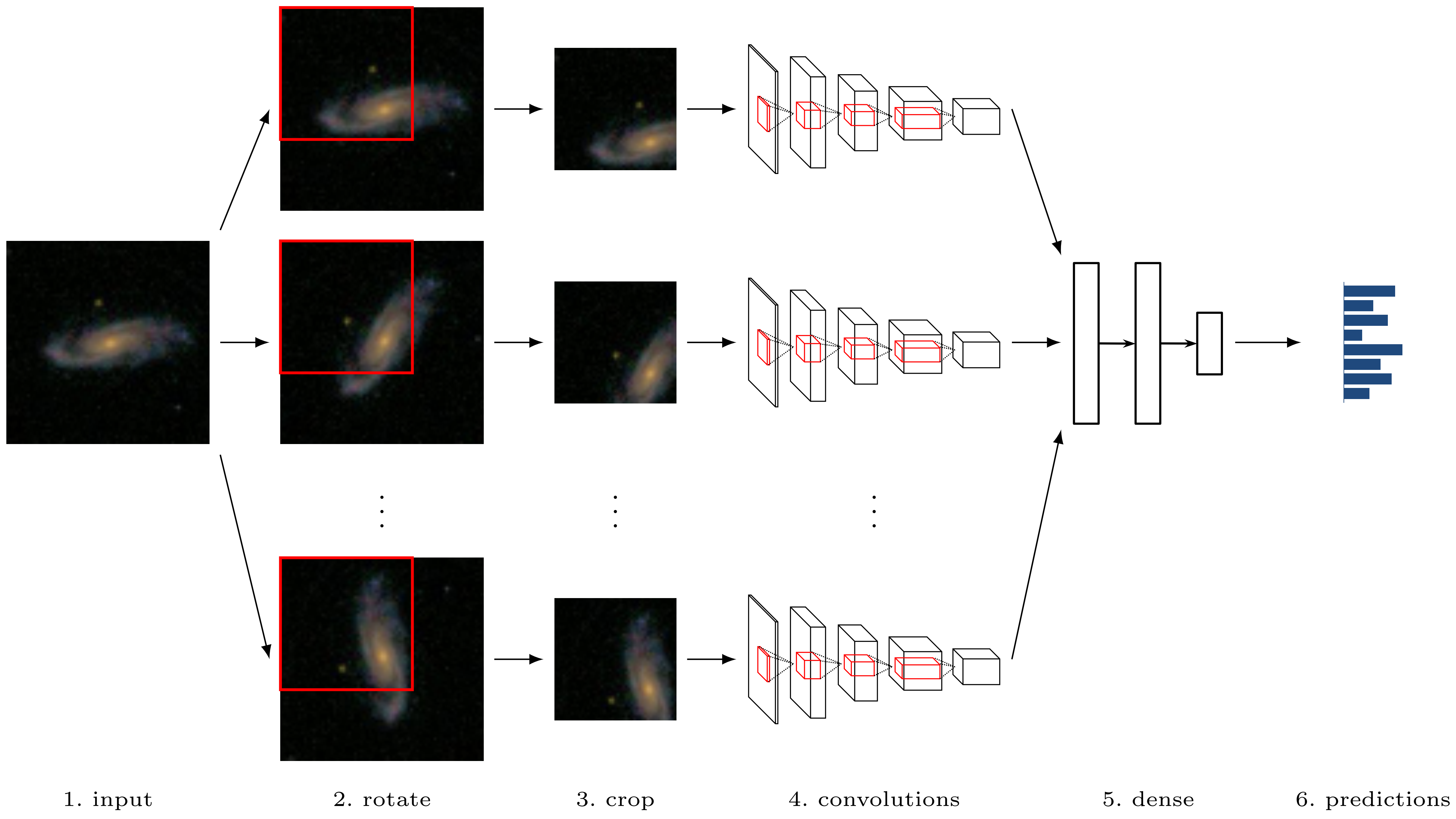


Generalizations: Data augmentations

Rotation-invariant convolutional neural networks for galaxy morphology prediction

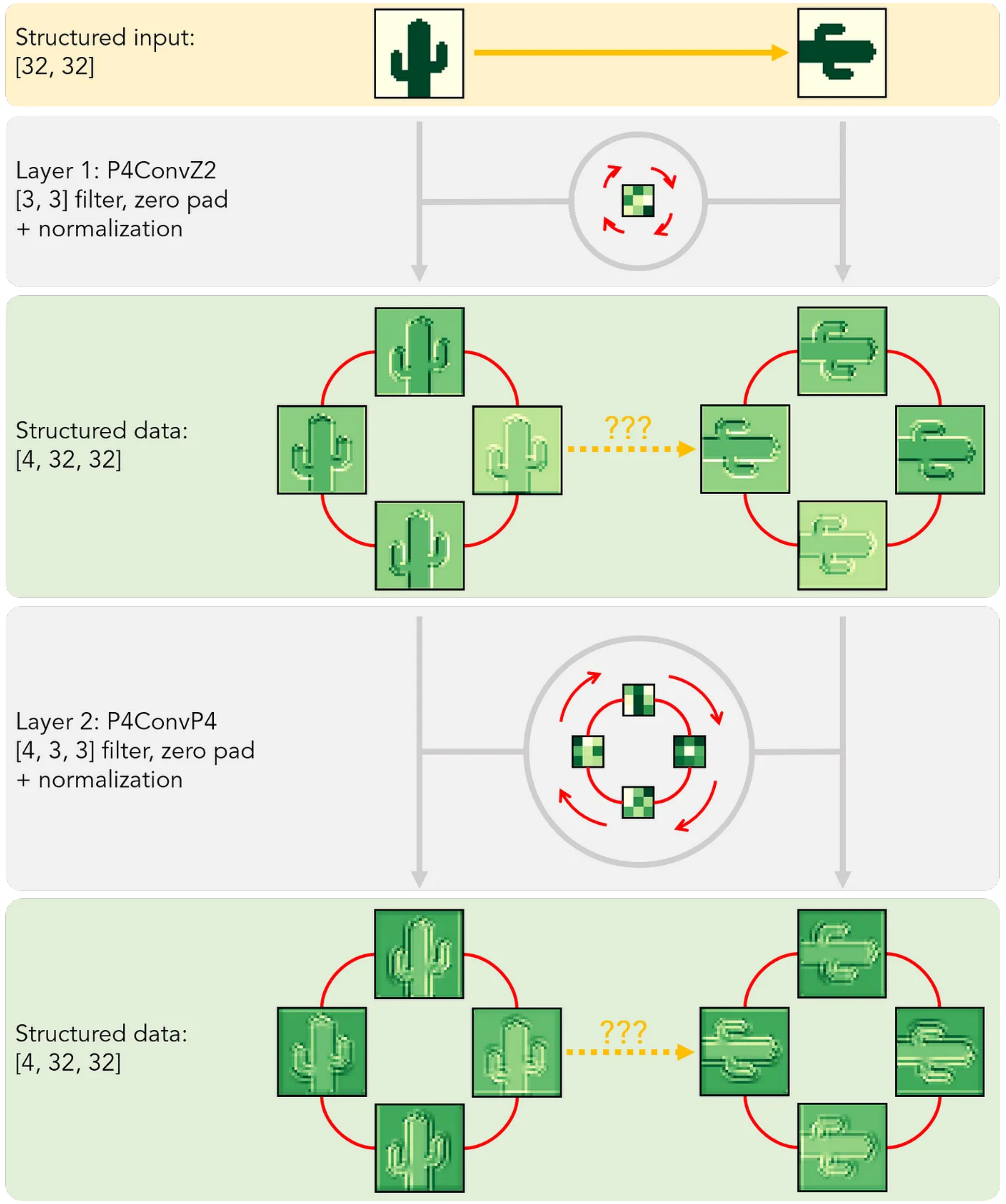
Sander Dieleman[1⋆], Kyle W. Willett[2⋆] and Joni Dambre[1]
[1] Electronics and Information Systems department, Ghent University, Sint-Pietersnieuwstraat 41, 9000 Ghent, Belgium
[2] School of Physics and Astronomy, University of Minnesota, 116 Church St SE, Minneapolis, MN 55455, USA

- One way to generalize CNNs to rotation-invariant operations:
  - Use data augmentations, concatenate feature maps, and apply dense layers



Generalizations: Other symmetry groups

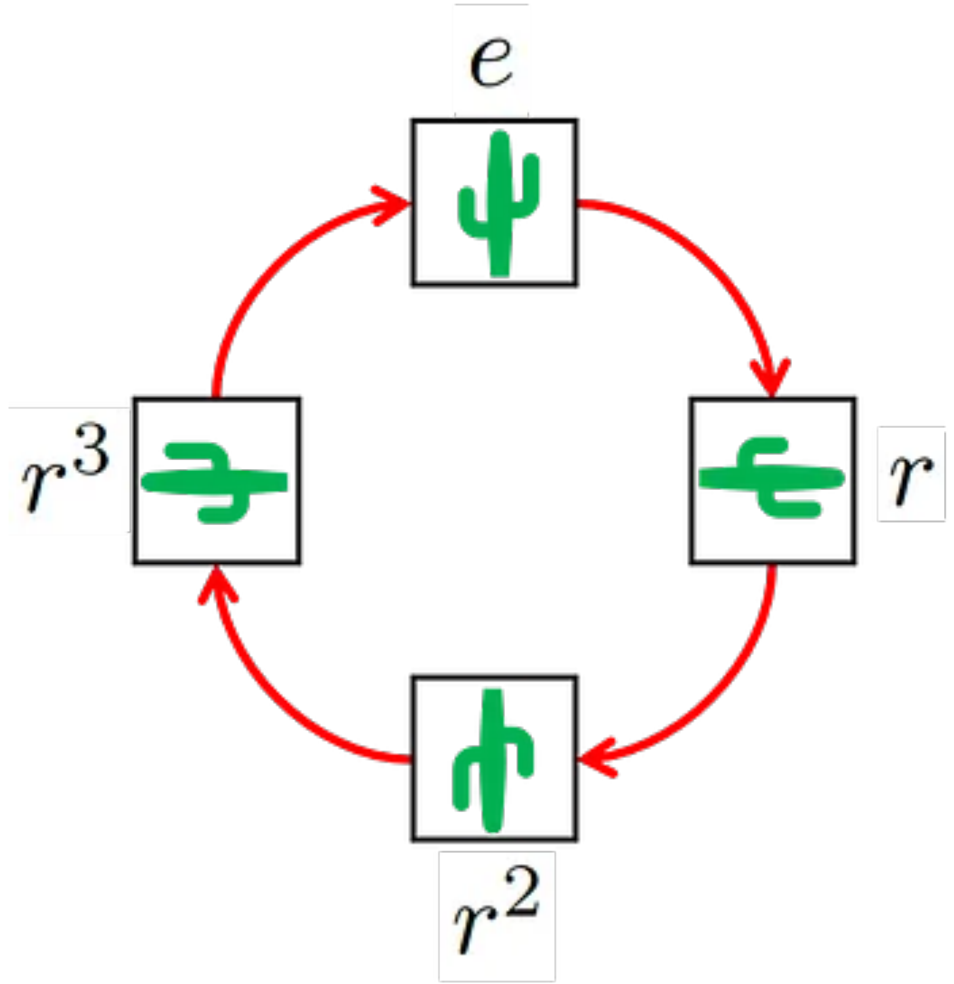
- By employing weight sharing across group actions, we can generalize to other symmetry groups



Group Equivariant Convolutional Networks

Taco S. Cohen T.S.COHEN@UVA.NL
University of Amsterdam

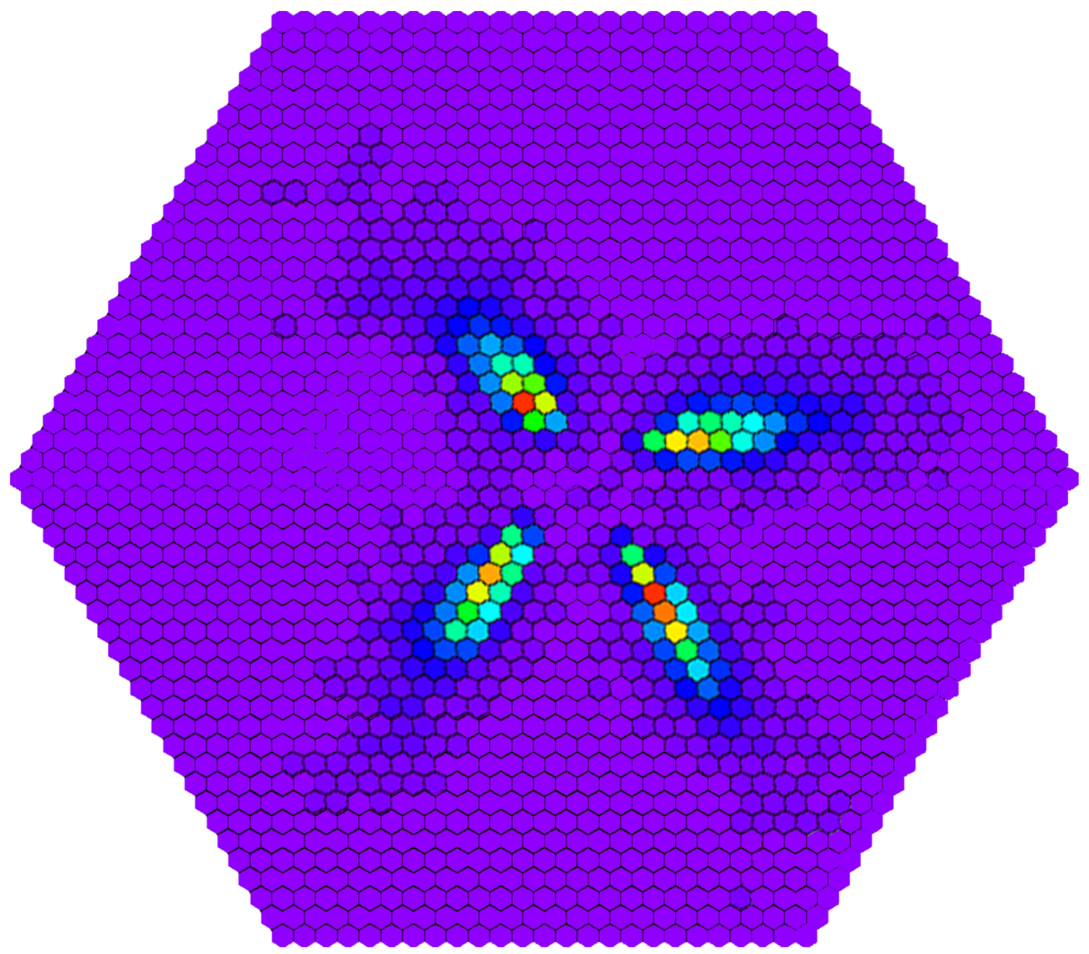
Max Welling M.WELLING@UVA.NL
University of Amsterdam
University of California Irvine
Canadian Institute for Advanced Research



https://medium.com/swlh/geometric-deep-learning-group-equivariant-convolutional-networks-ec687c7a7b41

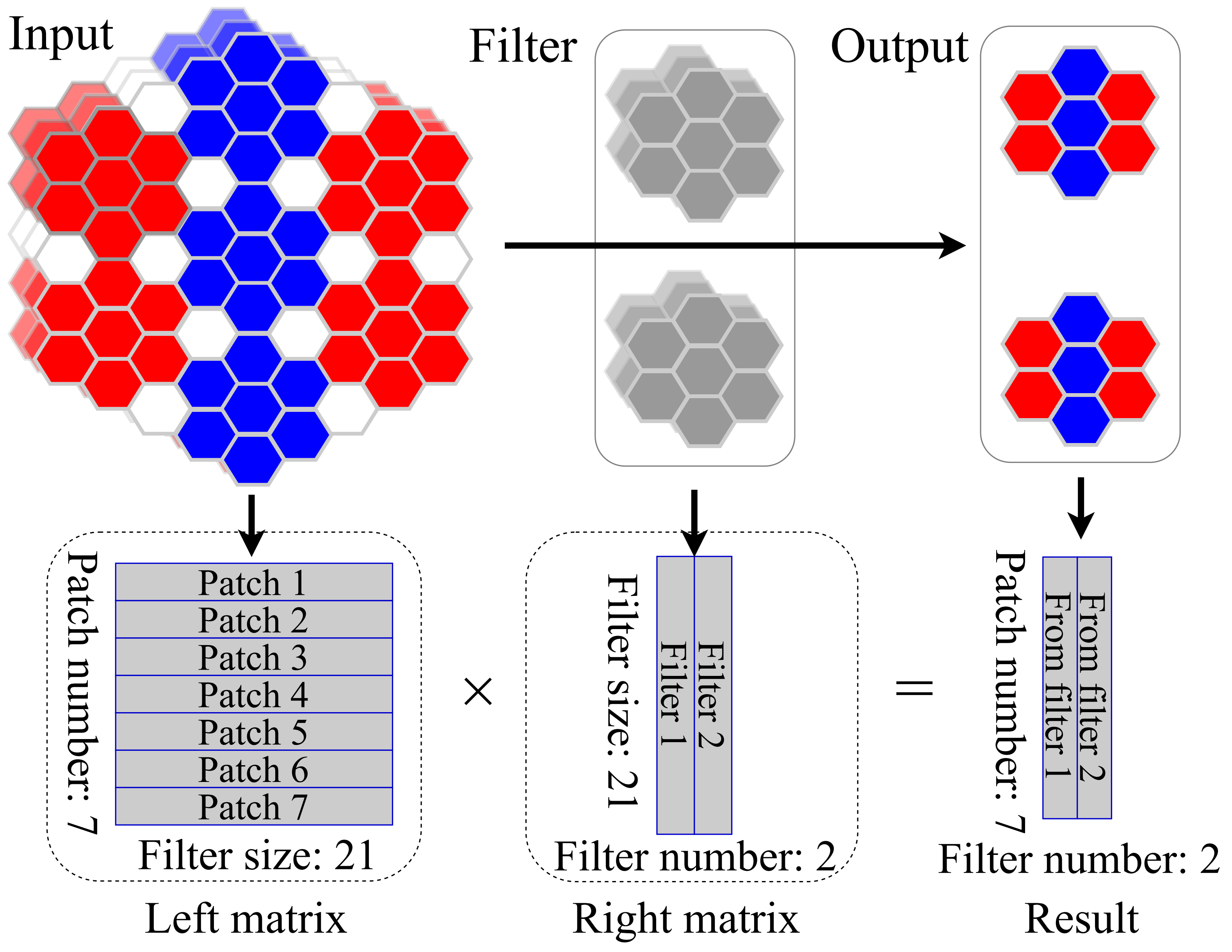
Generalizations: Other geometries

- Can generalize to other geometries like hexagonal data



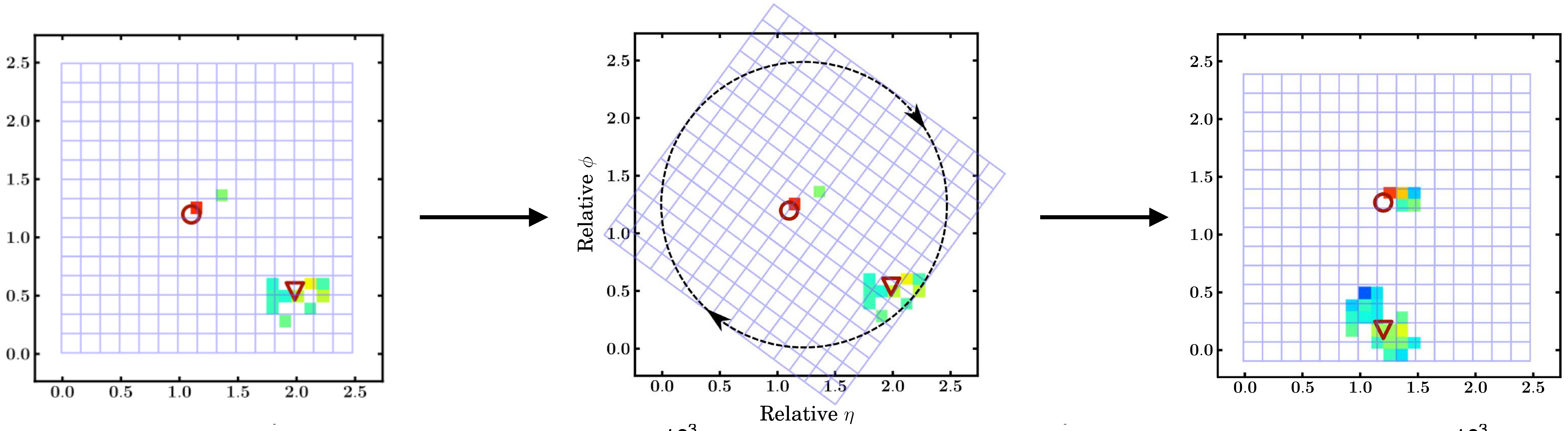
HexCNN: A Framework for Native Hexagonal Convolutional Neural Networks

Yunxiang Zhao[†], Qiuhong Ke[†], Flip Korn[‡], Jianzhong Qi[†], Rui Zhang [†*]
[†]The University of Melbourne, Australia, [‡]Google Research, USA
{yunxiangz@student., qiuhong.ke@, jianzhong.qi@, rui.zhang@ }unimelb.edu.au, flip@google.com

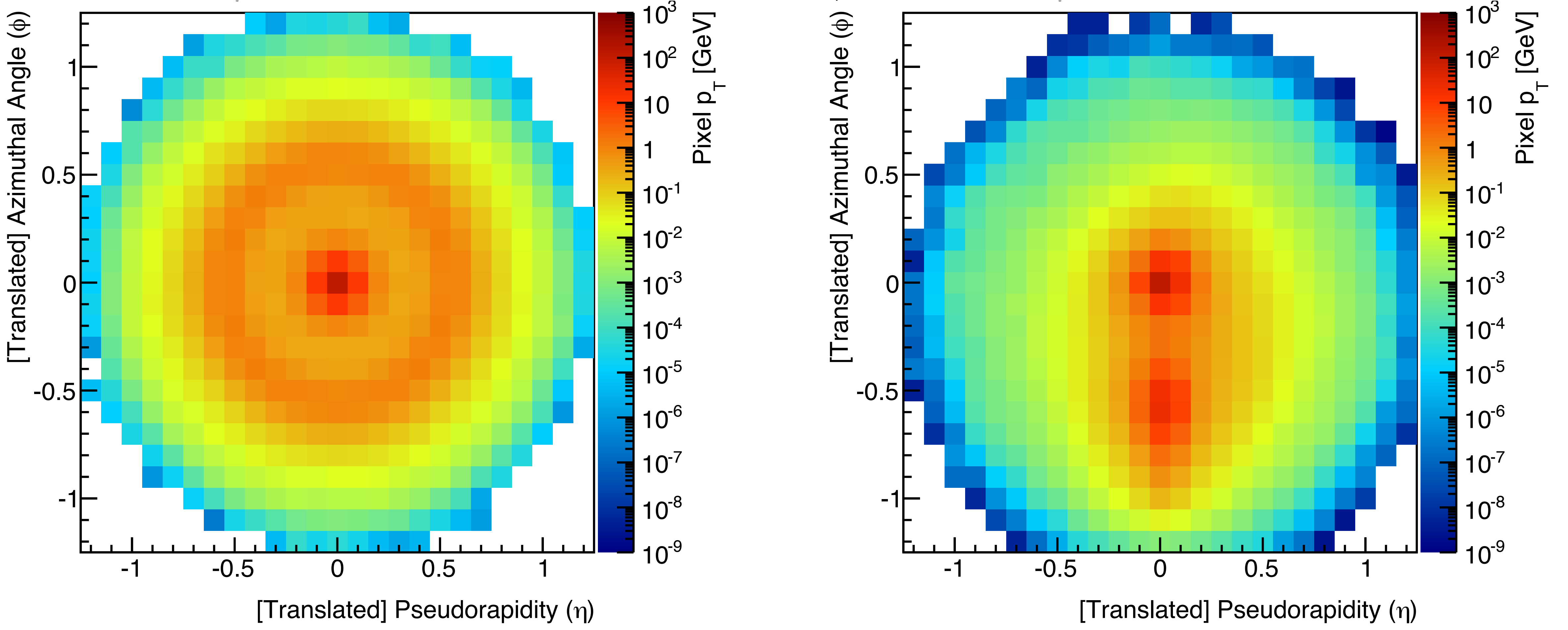


Caution: image preprocessing

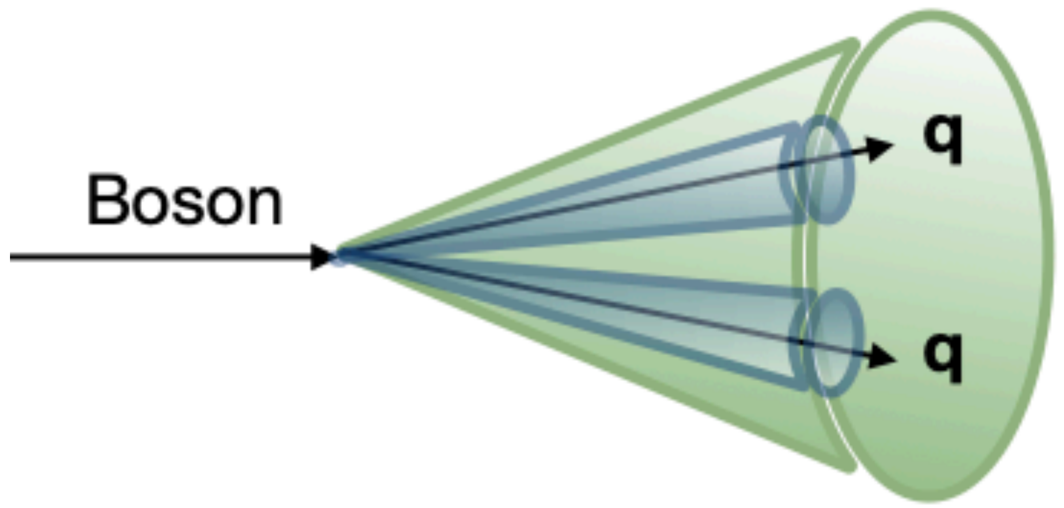
- Good practice to preprocess data, but beware of distortions to physically meaningful features



- Example: jet mass in $W(qq)$ jet images

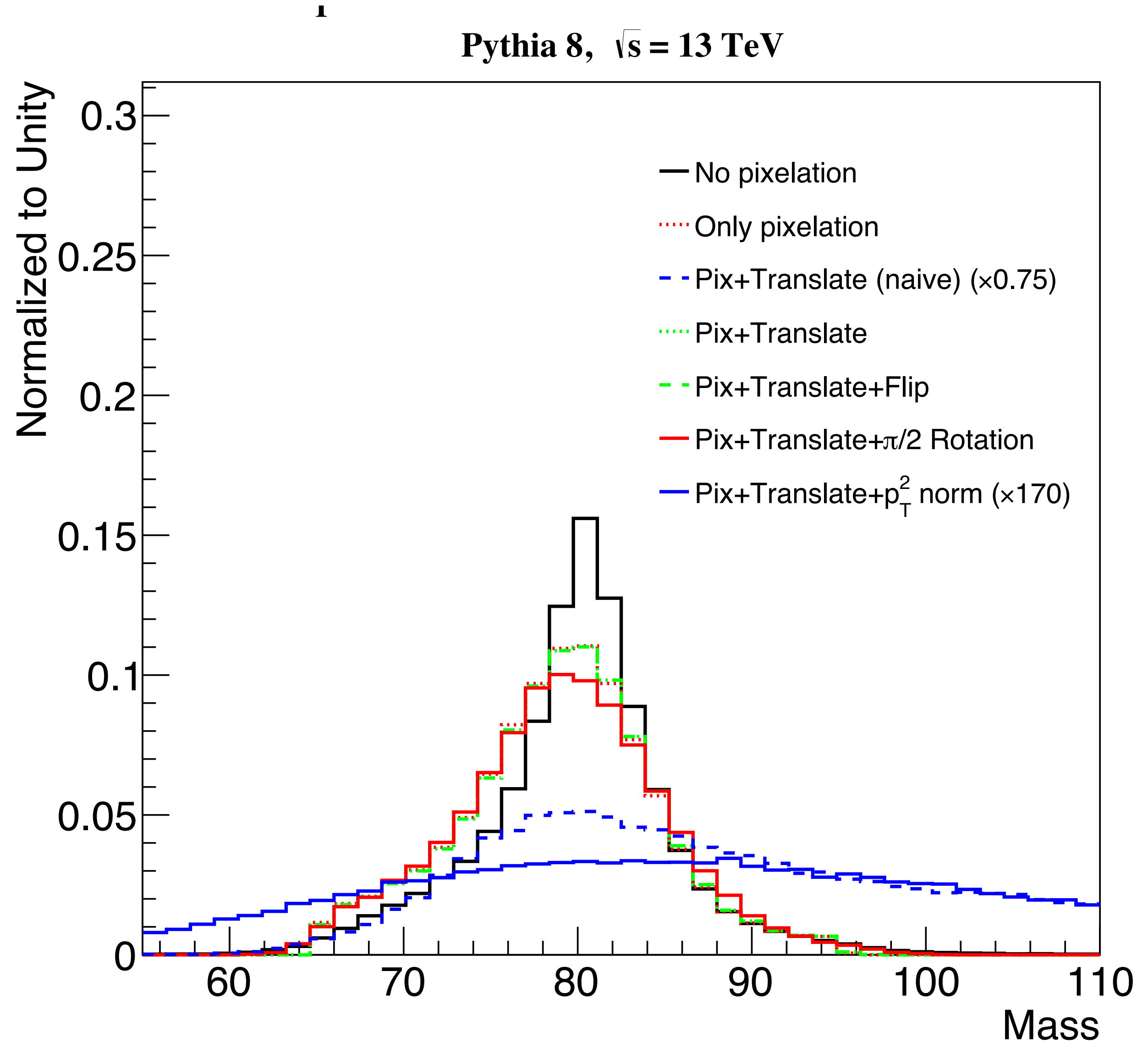


$$m^2 = \sum_{i<j} \frac{p_{T,i} p_{T,j}(1 - \cos\theta_{ij})}{\cosh\eta_i \cosh\eta_j}$$



  - Preprocessing: pixelization, rotation, flip, normalization

  - Preprocessing distorts distribution of the jet mass



- Can choose (Lorentz-invariant) preprocessing that preserves jet mass

Next time

- More on convolutional neural networks