Deep Convolutional Neural Networks

Ray Jones & Verena Kaynig, IACS

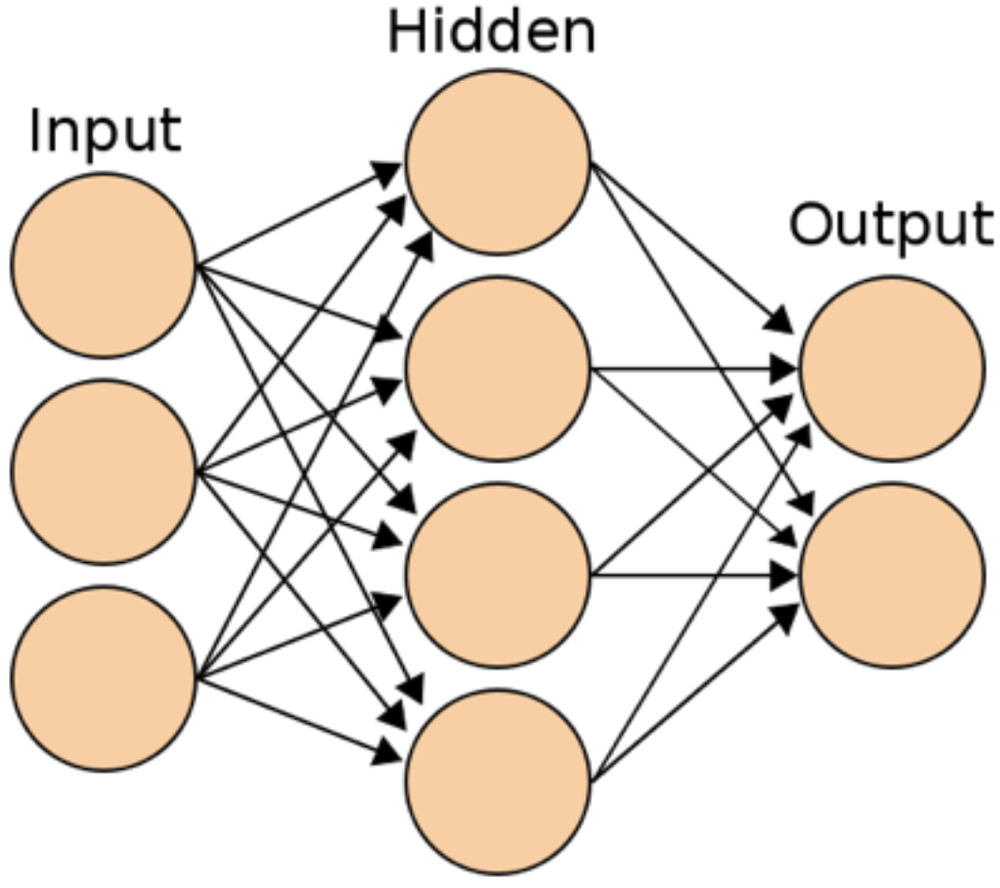
https://github.com/vkaynig/IACS_ComputeFest_DeepLearning

https://goo.gl/hfsI0u

Typical Feedforward NN

Series of matrix multiplies followed by nonlinear transforms:

$$\mathbf{x}_{i+1} = \sigma_i(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i)$$



Typical Feedforward NN

Series of matrix multiplies followed by nonlinear transforms:

$$\mathbf{x}_{i+1} = \sigma_i(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i)$$

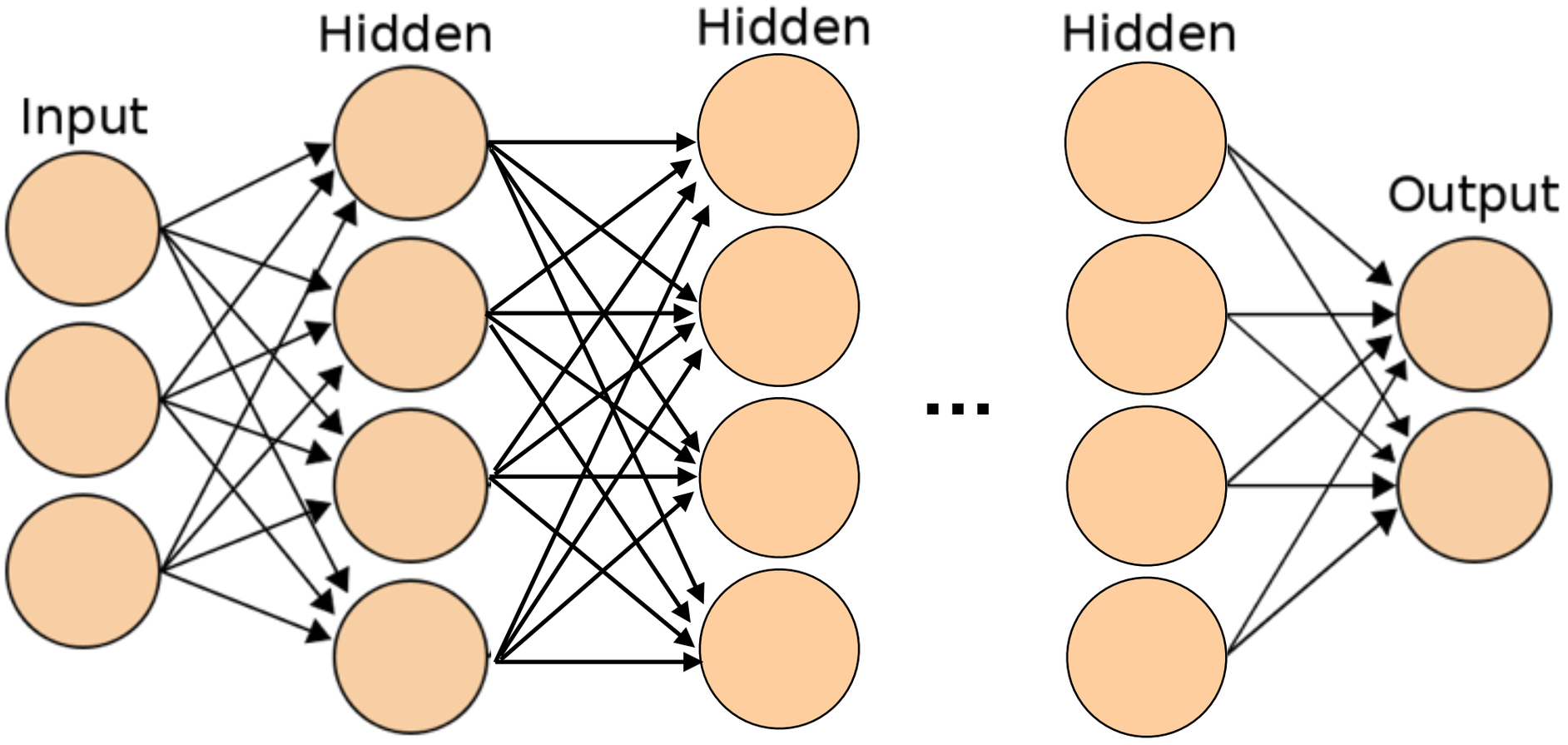
$\mathbf{x}_i$ = input data at layer i

σ_i = nonlinearity at layer i

$\mathbf{W}_i$ = weight matrix at layer i

$\mathbf{b}_i$ = bias at layer i

Deep NN



$$\mathbf{x}_{i+1} = \sigma_i(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i)$$

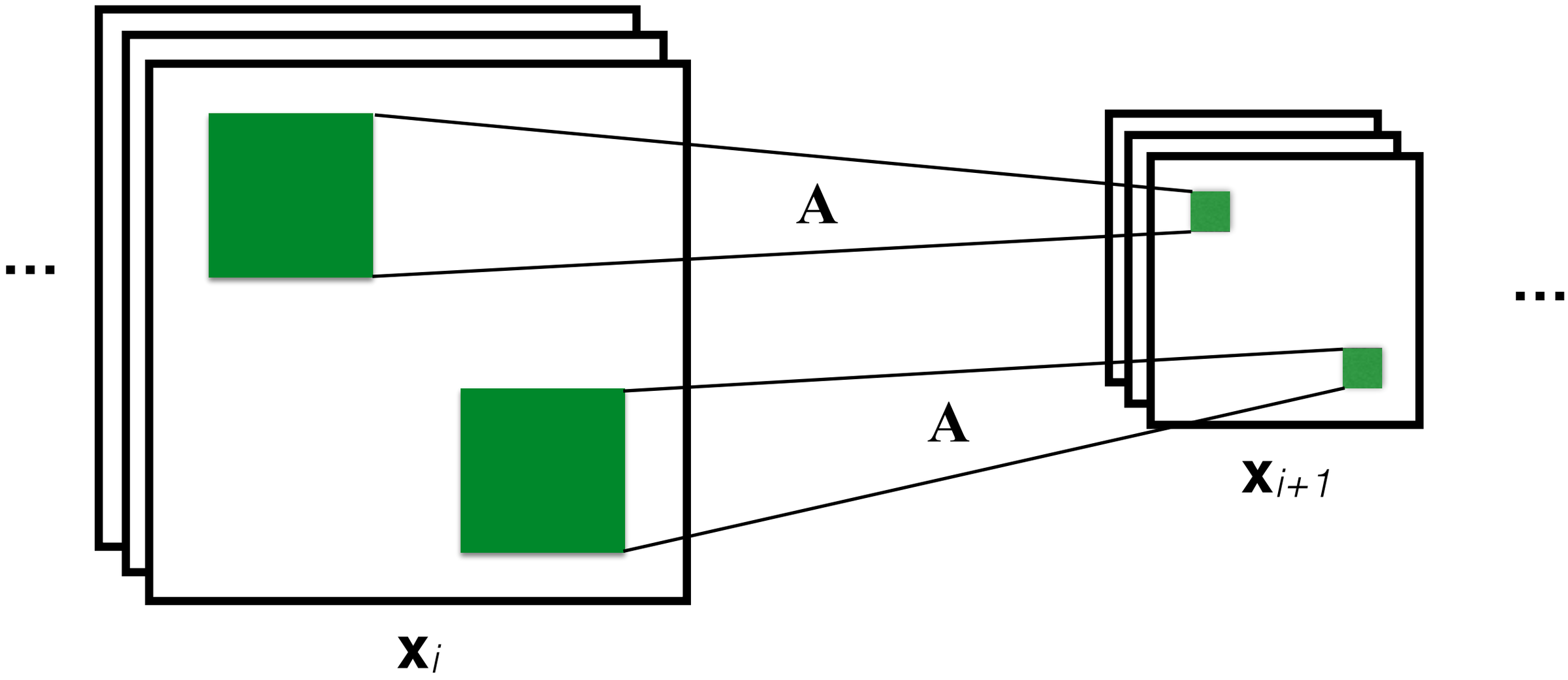
Deep vs. Convolutional

- Deep:
  - More powerful than than shallow networks.
  - Increased parameters = harder to train & evaluate.

- Convolutional:
  - Uses sparse & structured $\mathbf{W}$ between layers.
  - reduced training and evaluation cost

- Origins in Computer Vision
  - Large input space (images), …
  - With structure we can use.

Convolutional NN



$$\mathbf{x}_{i+1} = \sigma_i(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i)$$

Convolution <-> Same $\mathbf{A}$ at different locations.

$\mathbf{W}$ is made up of offset copies of $\mathbf{A}$

Vision works with High Dimensional Inputs

- A typical image will be several megapixels.
- 1K x 1K image => 1 million inputs
  - untenable to train a fully connected network
- Fortunately, images have some useful properties.

Images are Structured, Local, and Hierarchical

Nearby pixels are more strongly related than distant ones.

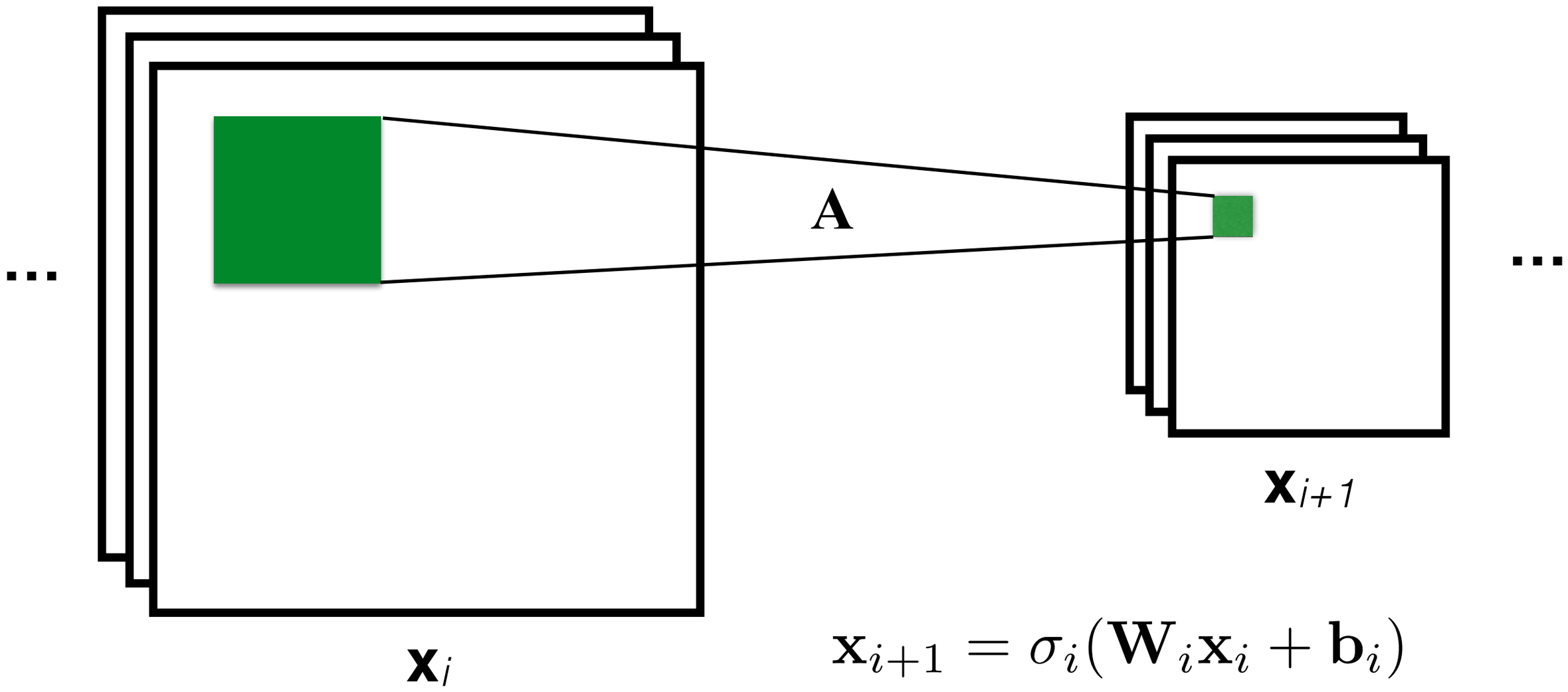
Objects are built up out of smaller parts.

Images are Invariant

CNNs for Vision

- Local Connectivity (locality)
- Parameter Sharing (translation invariant)
- Low-level features are learned (structure)
- Pooling (hierarchical)

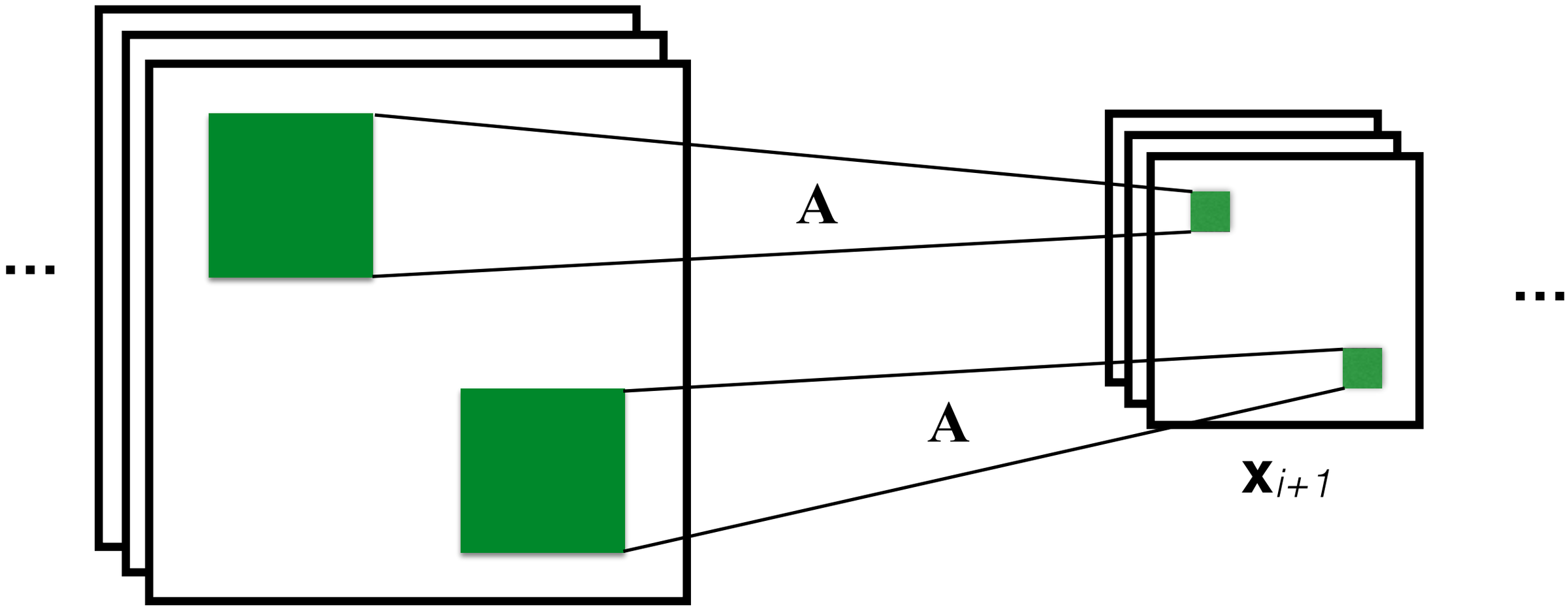
Local Connectivity



$$\mathbf{x}_{i+1} = \sigma_i(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i)$$

Each output has depends on only a small region of the input, so $\mathbf{W}$ is sparse.

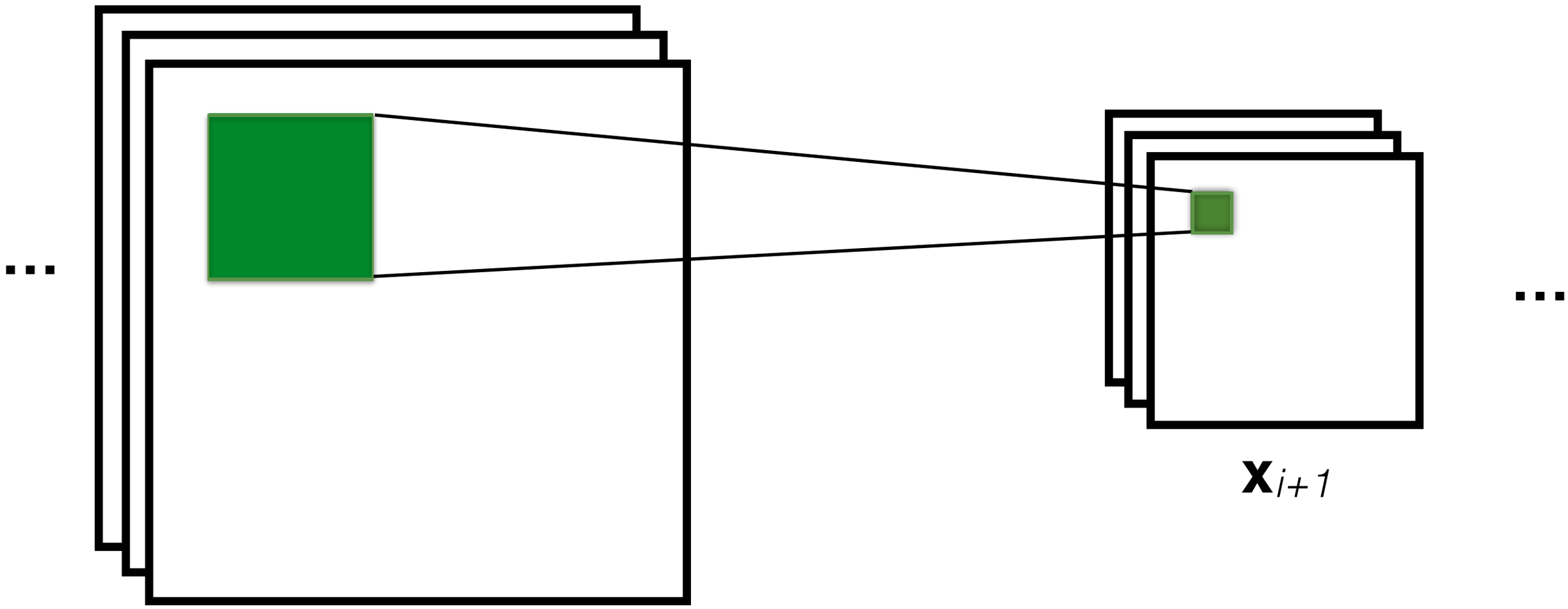
Parameter Sharing



$$\mathbf{x}_{i+1} = \sigma_i(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i)$$

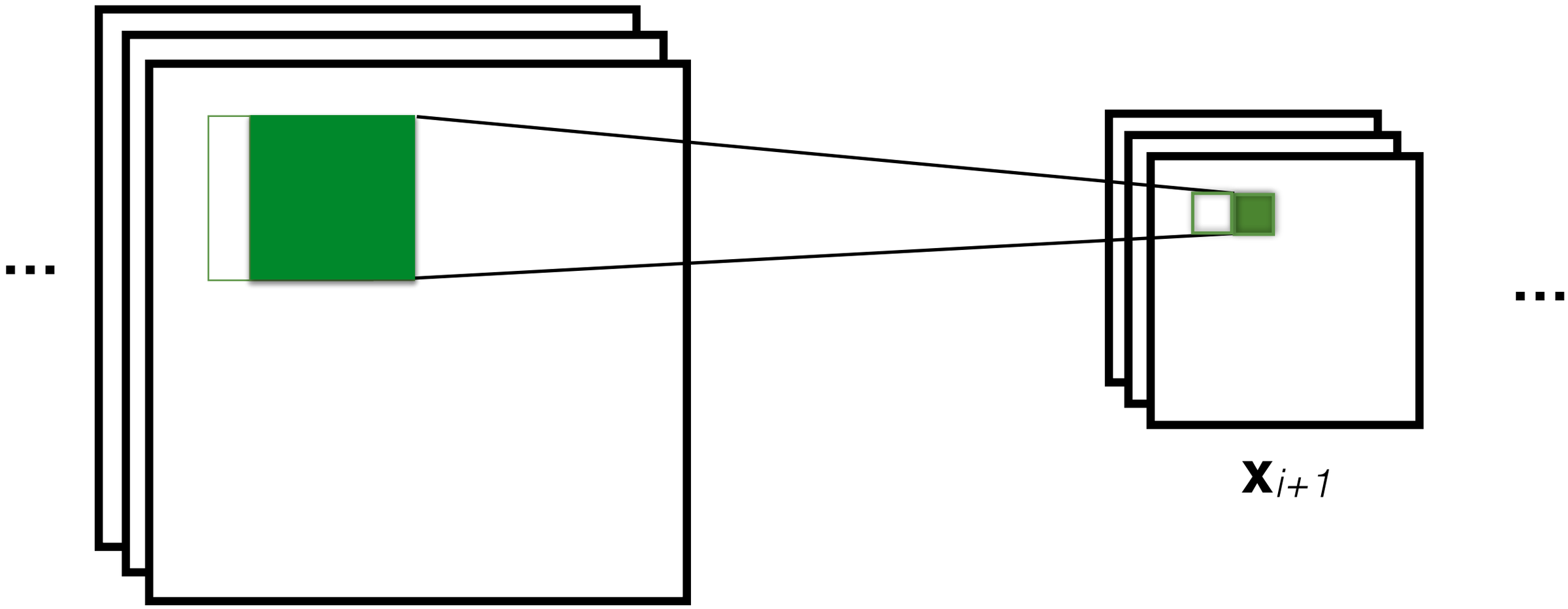
Each $\mathbf{A}$ is the same, reducing free parameters in $\mathbf{W}$.

Sliding window



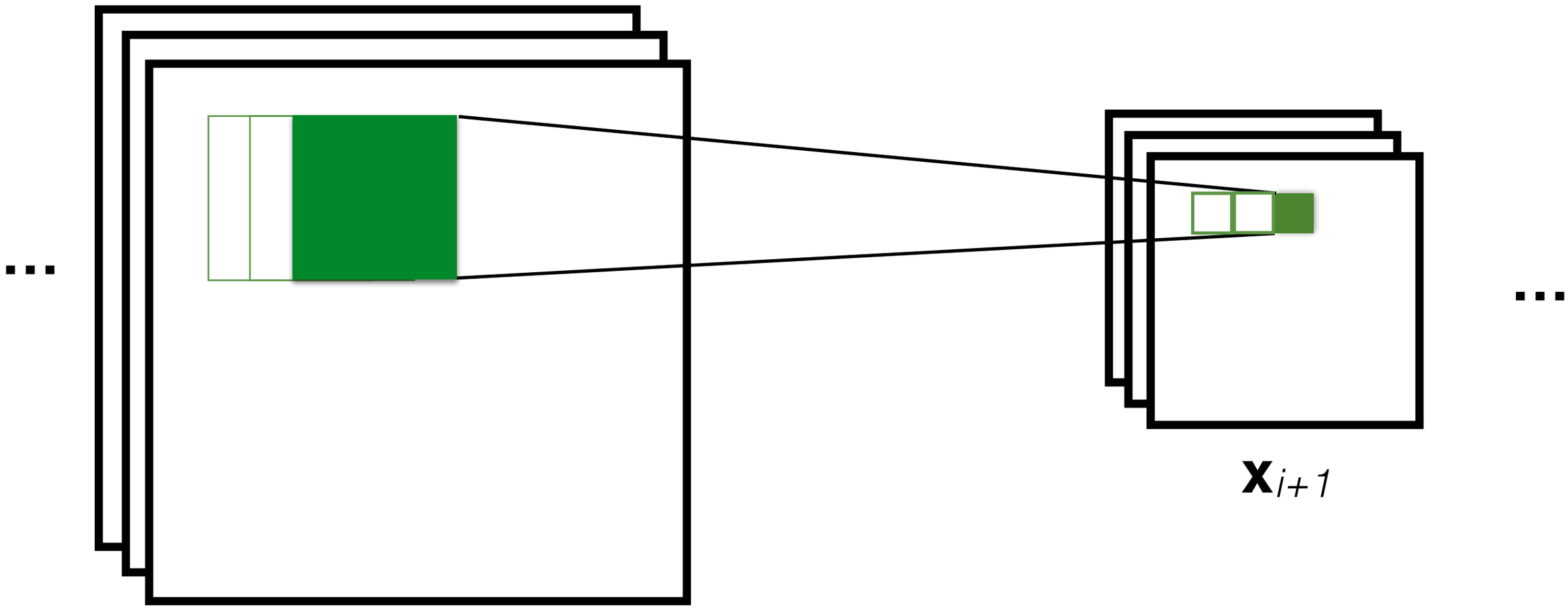
$$\mathbf{x}_{i+1} = \sigma_i(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i)$$

Sliding window



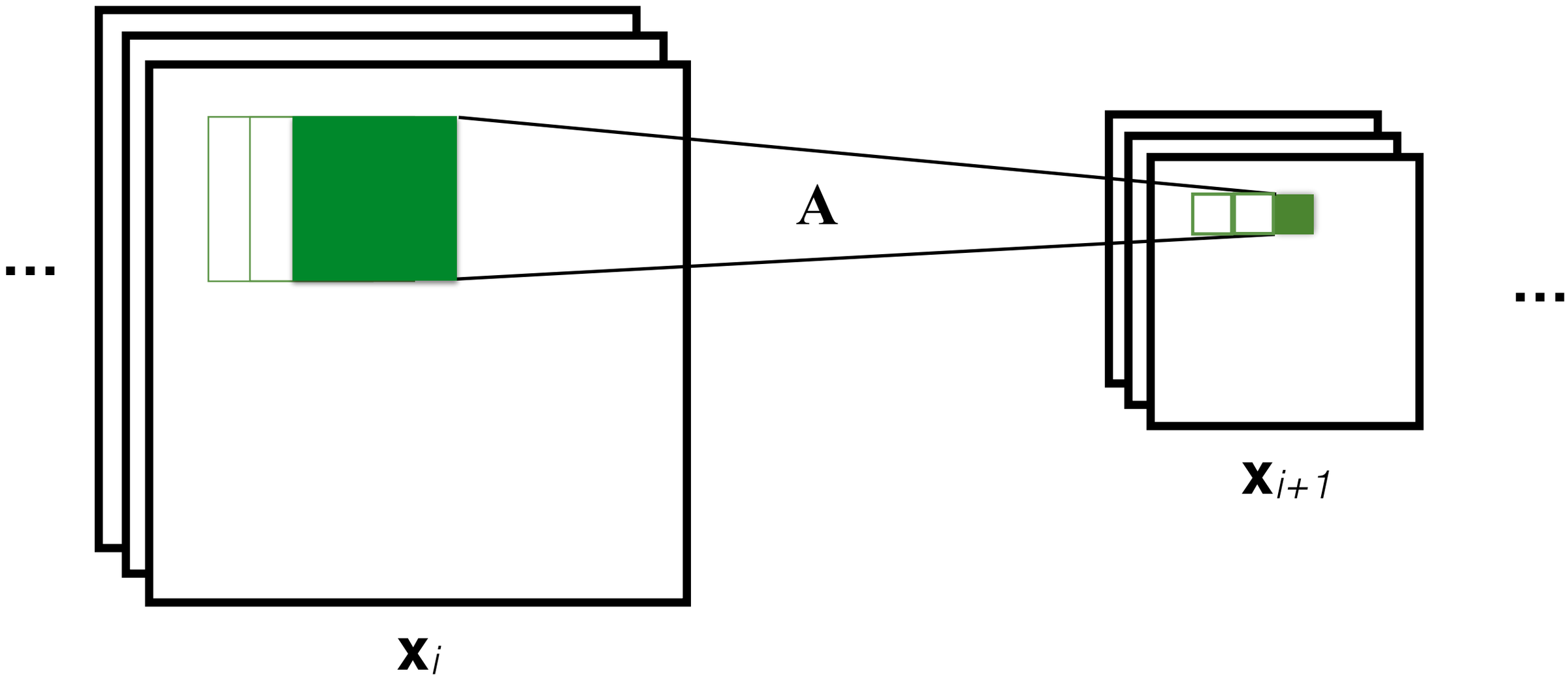
$$\mathbf{x}_{i+1} = \sigma_i(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i)$$

Sliding window



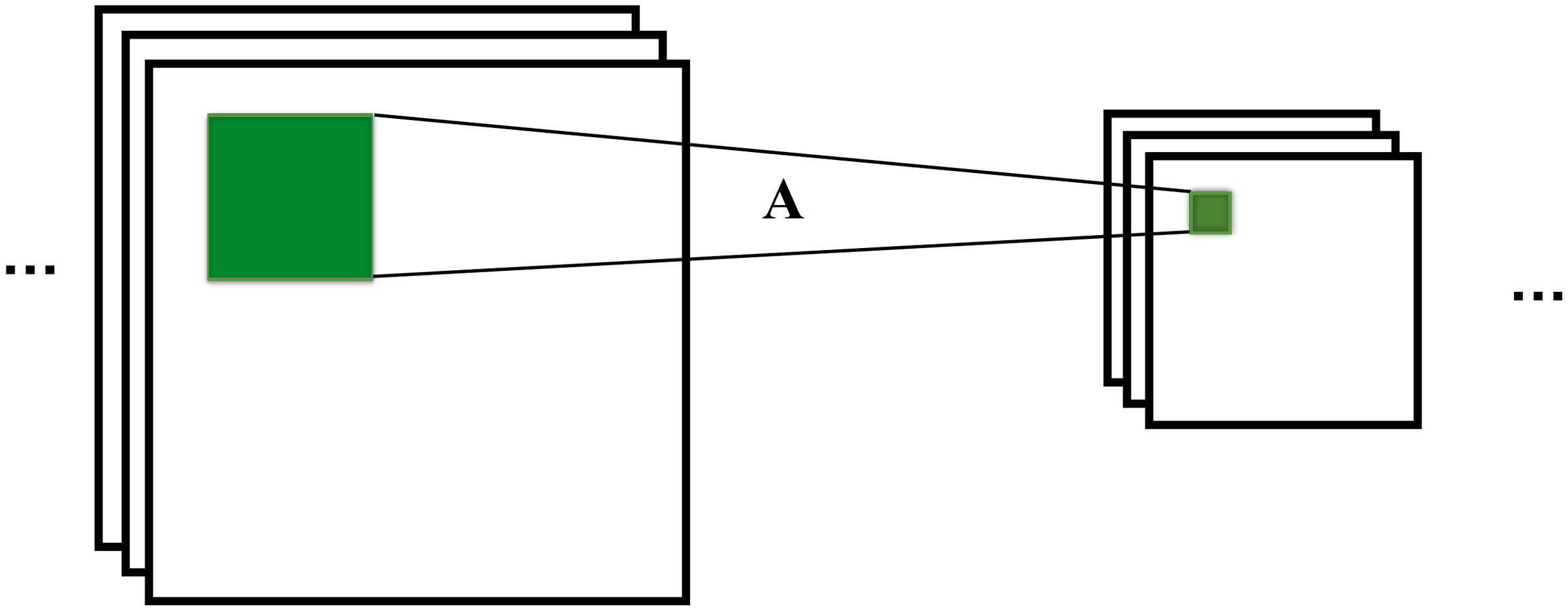
$$\mathbf{x}_{i+1} = \sigma_i(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i)$$

This is the same as the Convolution of A with $\mathbf{x}_i$



(or correlation with a flipped $\mathbf{A}$, if you prefer.)

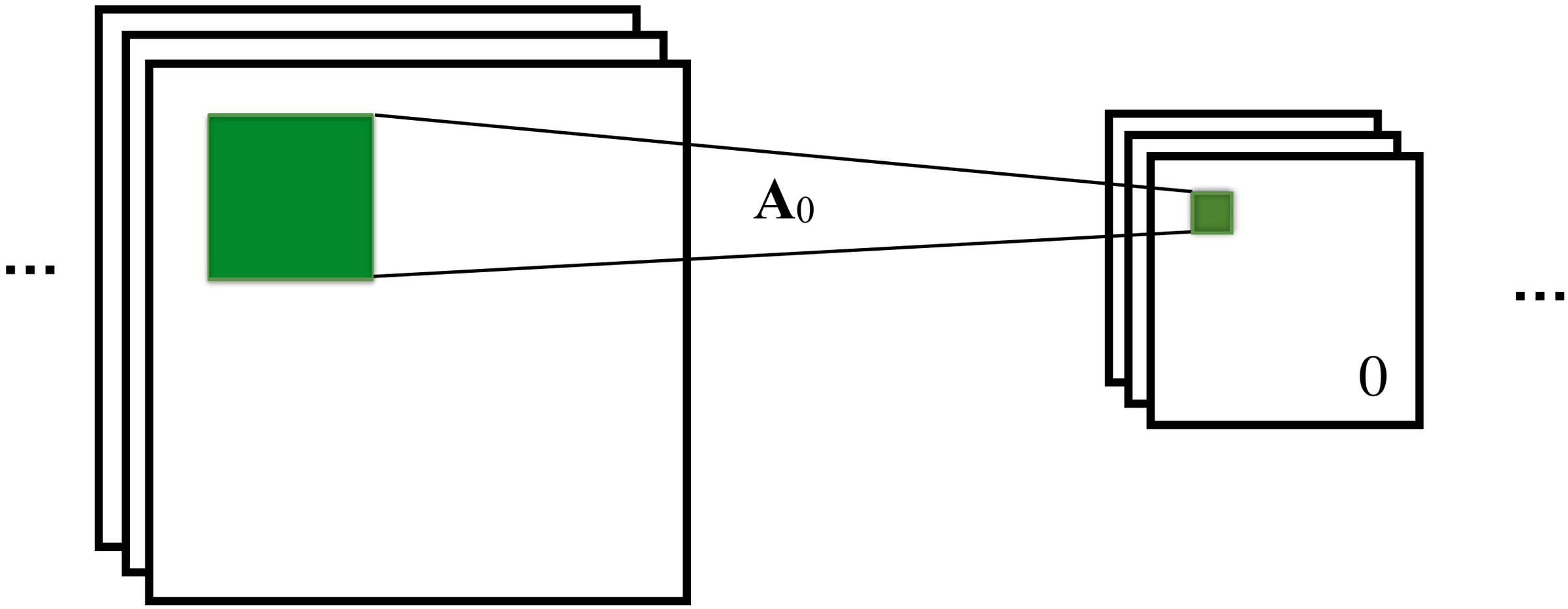
Input/output dimensions



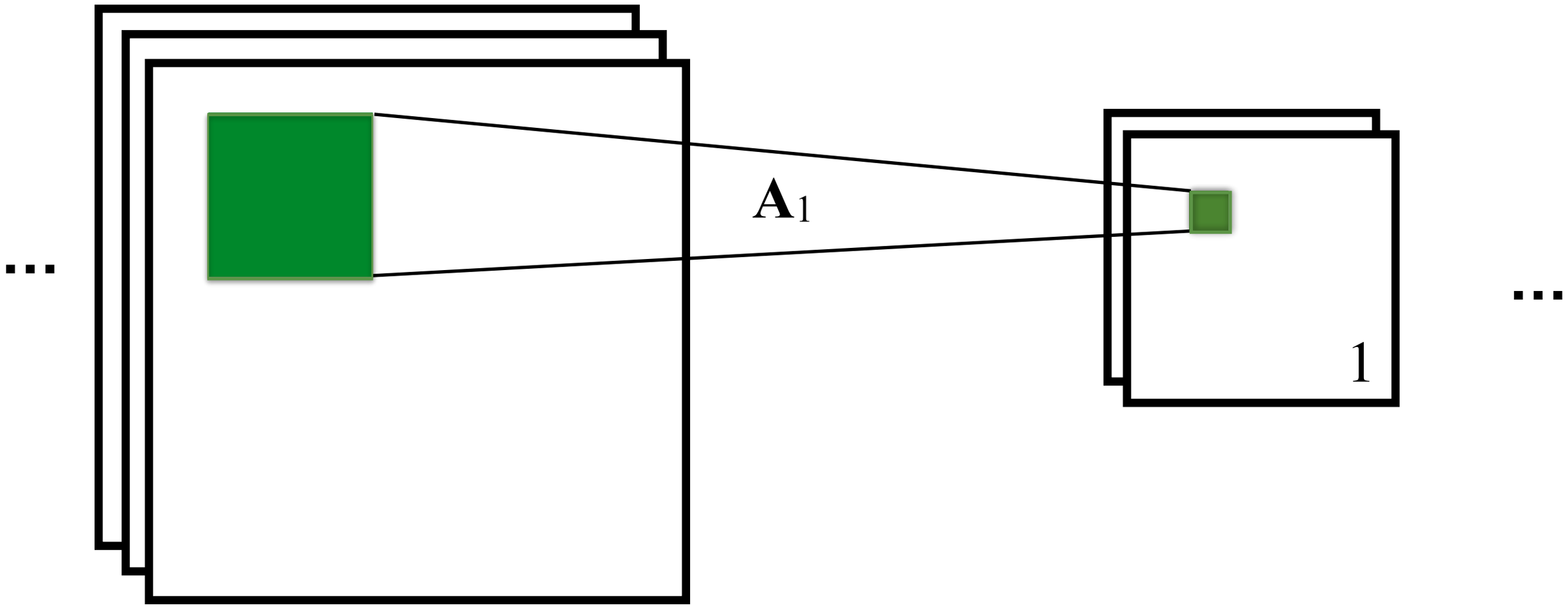
The input and output of each CNN layer may have several “channels”, aka “Feature Maps.

Each output has its own **A**.

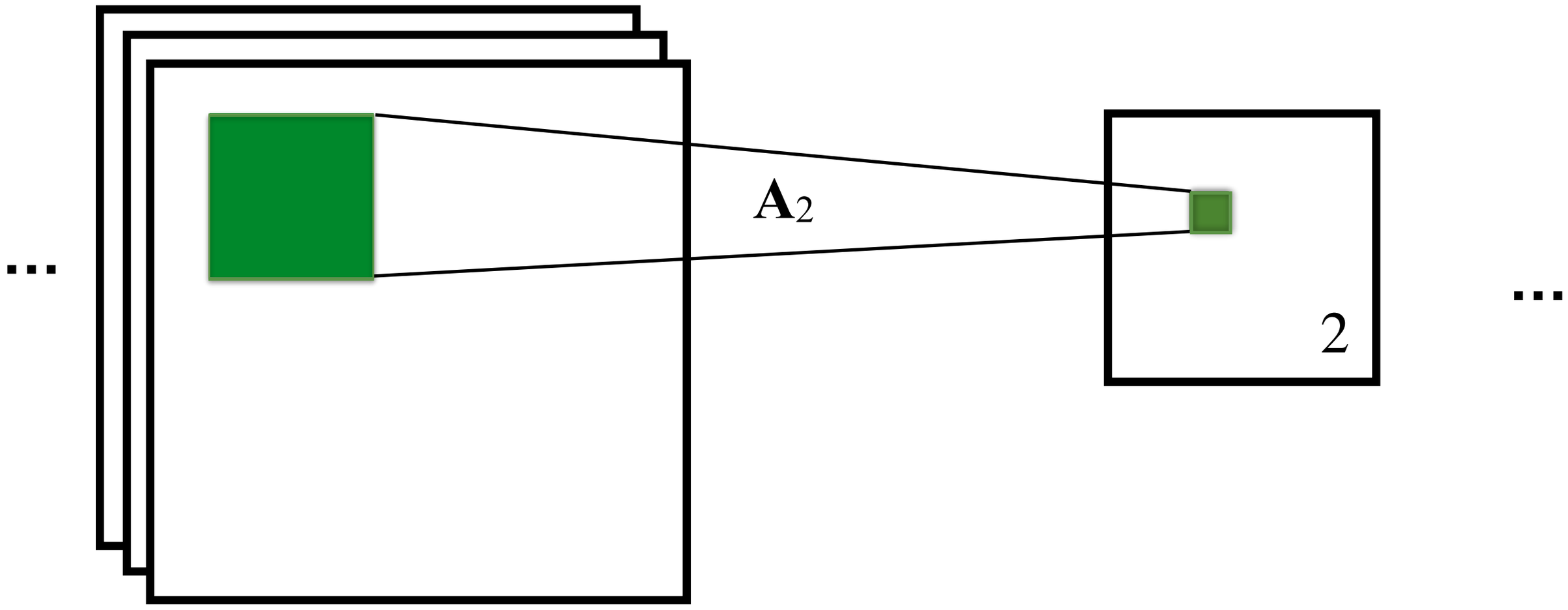
Input/output dimensions



Input/output dimensions



Input/output dimensions

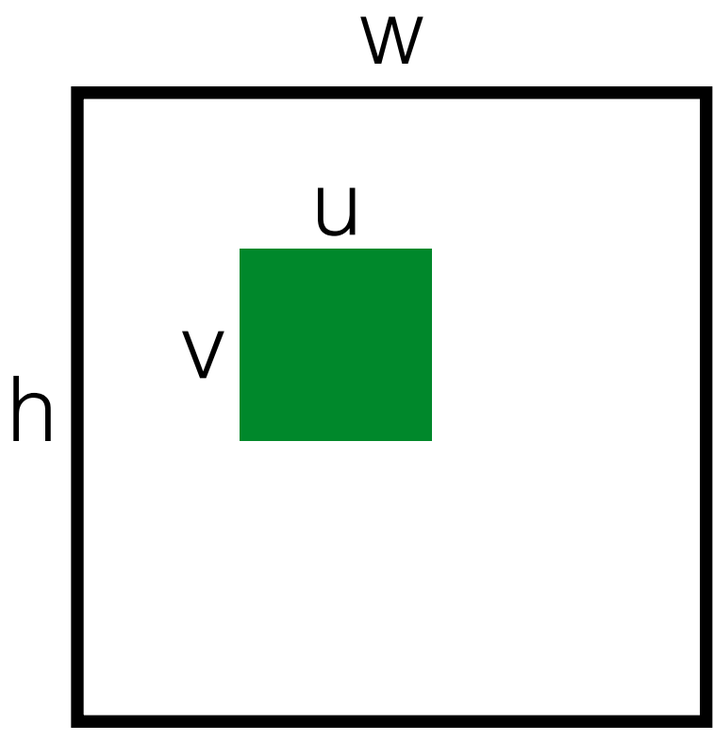


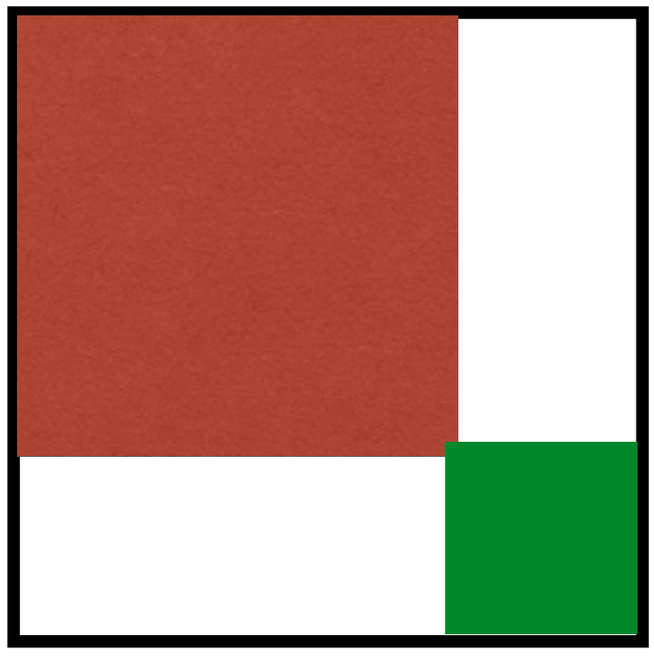
But!

- Each **A** uses the entire depth of its input.

- If the input is D x w x h
  and there are N filters of size D x u x v,
  the output will be of size:

N x (w - u + 1) x (h - v + 1)



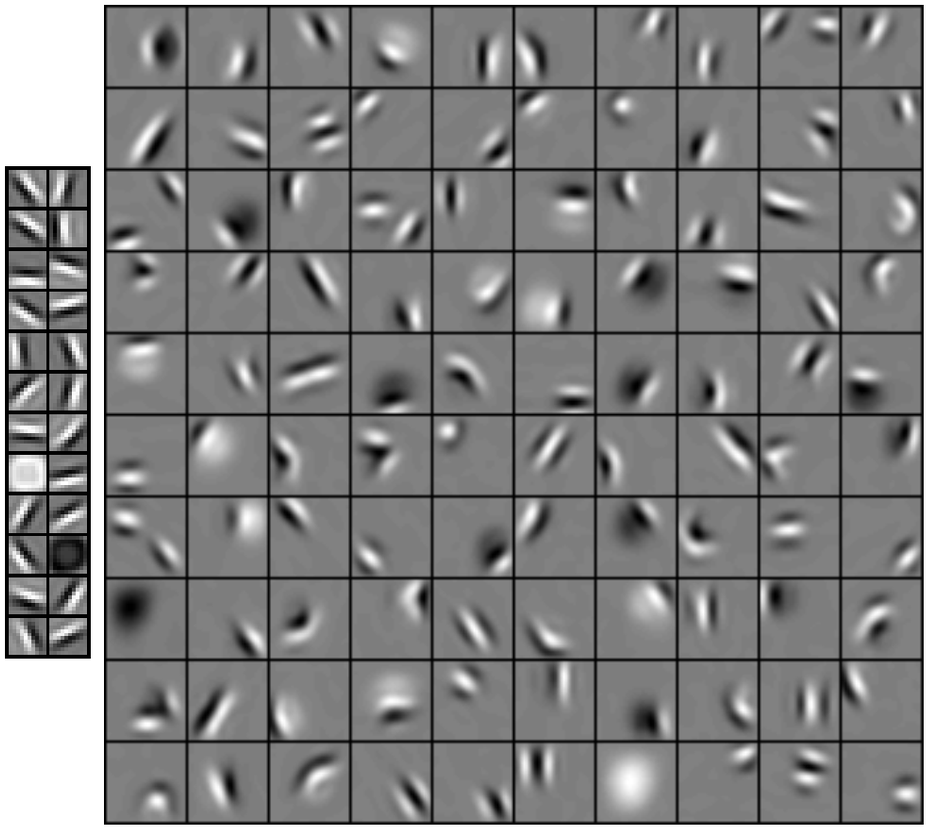


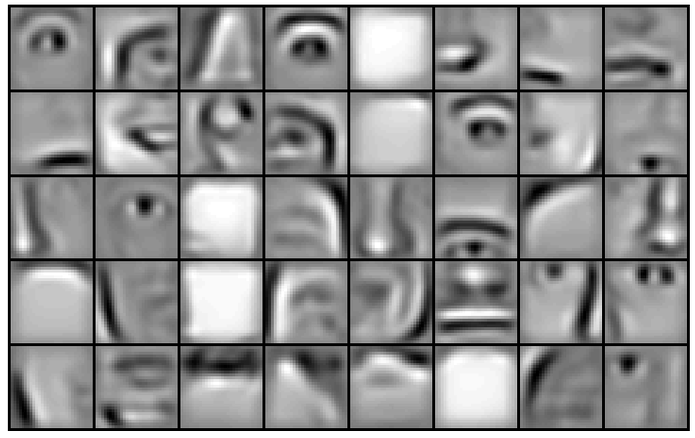
Benefits of using Convolutions

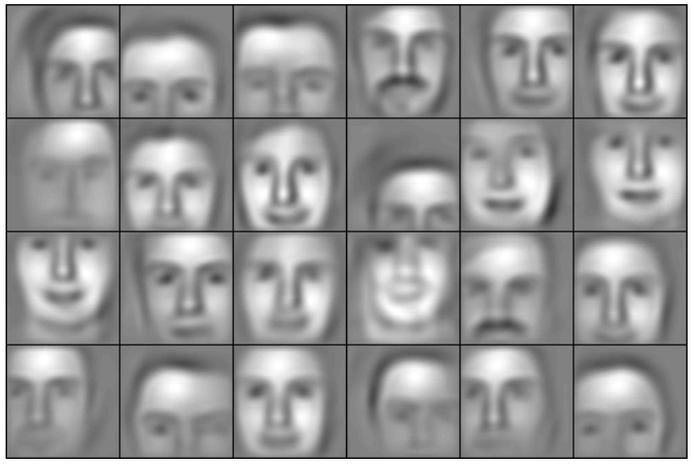
- Fewer parameters
- Fast implementations
- Locality preserving across layers

Images are Hierarchical

Big things are made up of smaller parts.



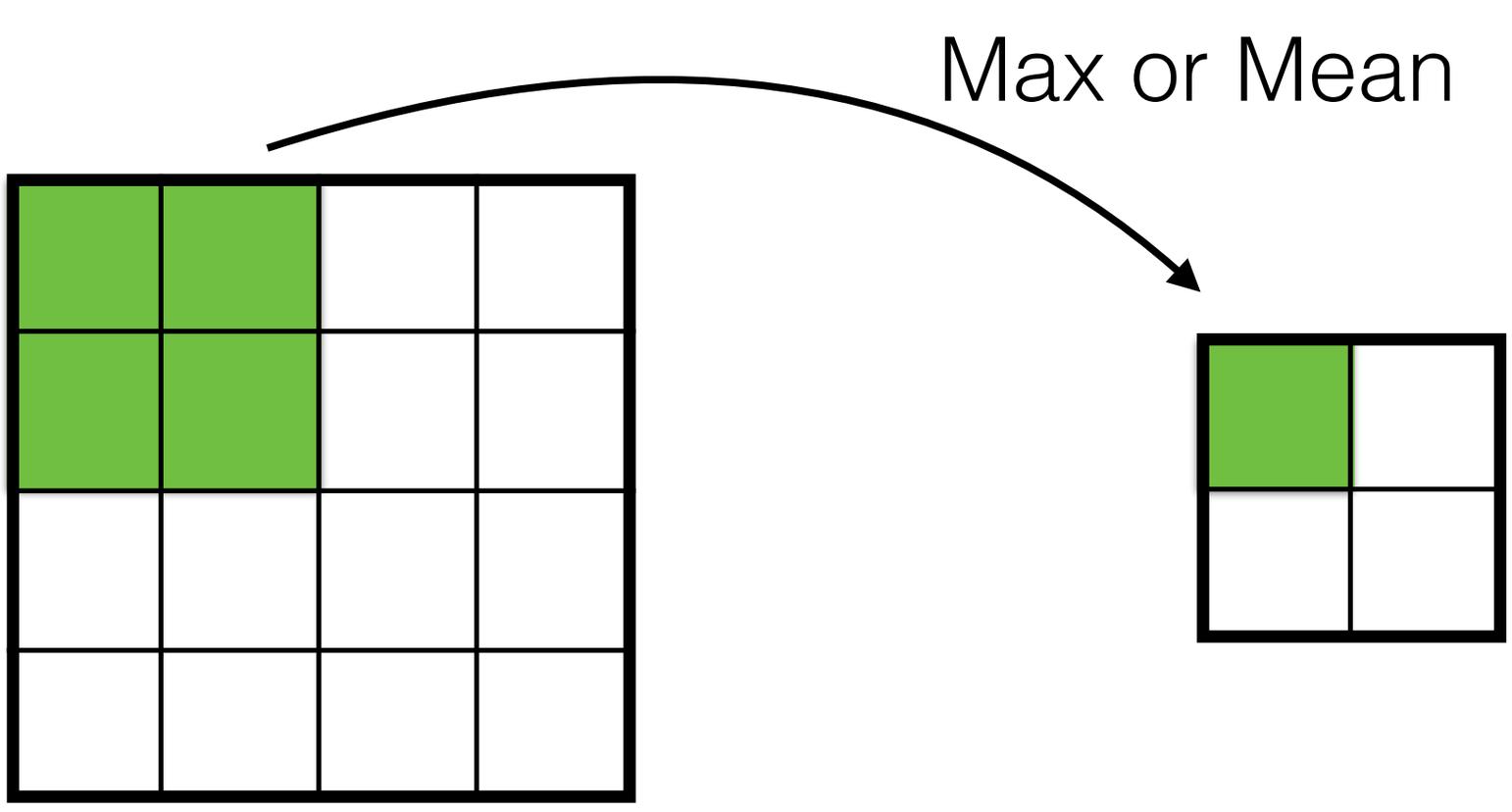




from Lee et al., *ICML,* 2009

Pooling

Combine small local regions into single output.
Adds hierarchy, translation-invariance, nonlinearity, data reduction.



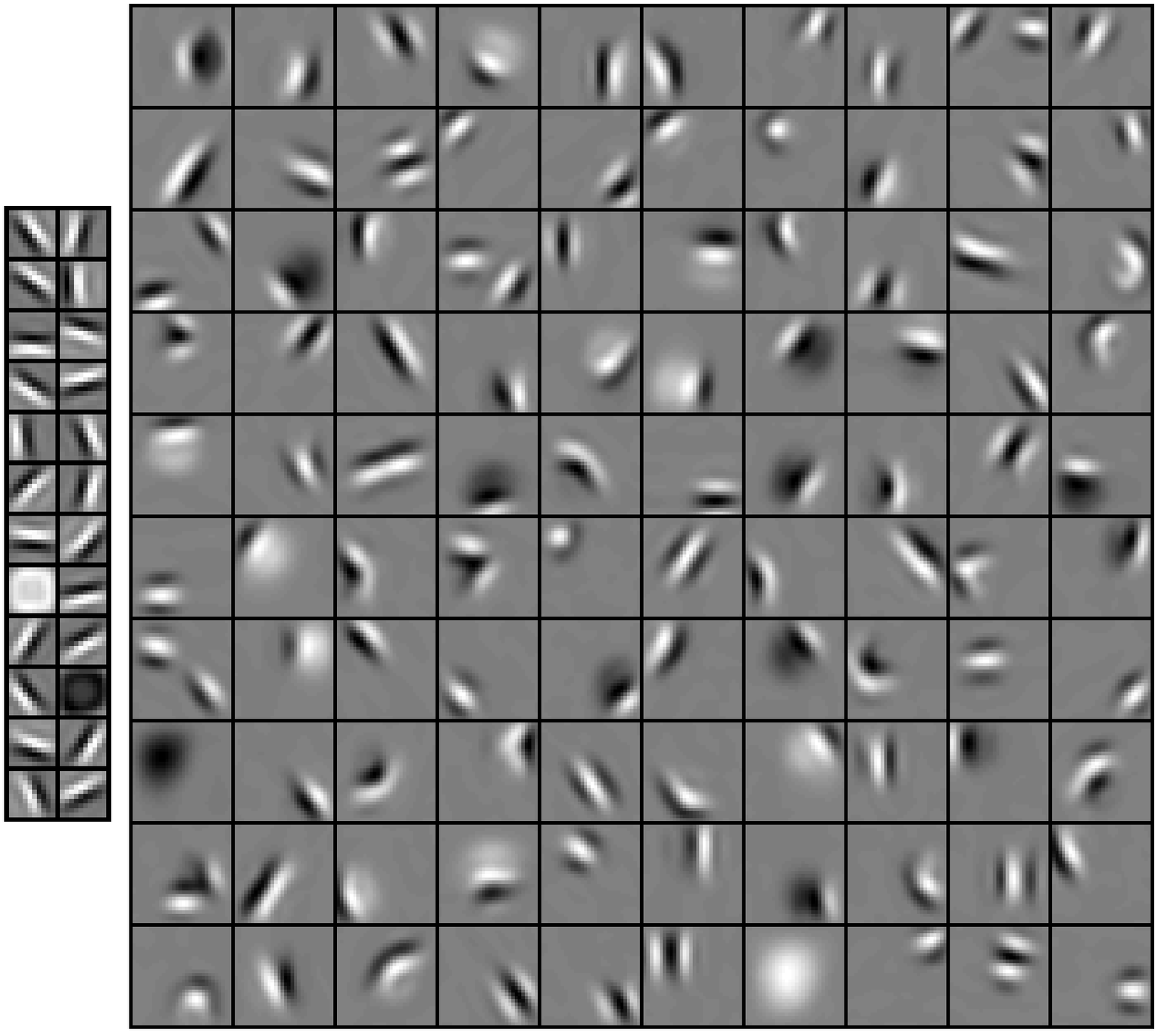
IPython Time

git clone https://github.com/vkaynig/IACS_ComputeFest_DeepLearning

ConvolutionalNetworks.ipynb

http://goo.gl/LCJGBn

Learned features from Natural Images



Lee et al., *ICML*, 2009

Advanced Methods

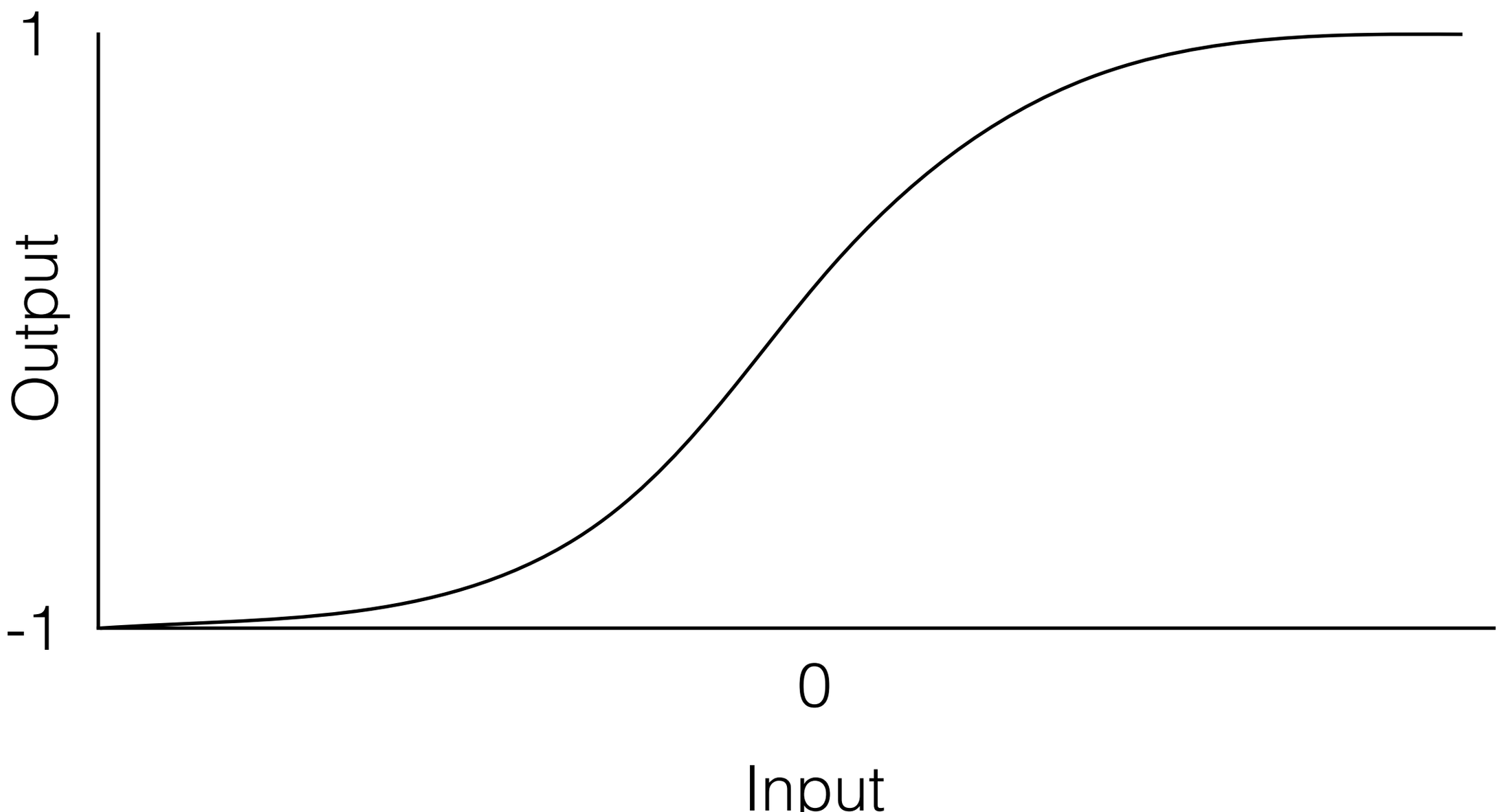
- Dropout & Batch Normalization
- Maxout
- Lasagne, Pylearn2, Caffe, Torch, TensorFlow
- “Intriguing properties of neural networks”

Dropout training

- Prevent outputs from becoming co-dependent.
- During training, randomly set some fraction of inputs at each layer to zero.
- After training, scale $\mathbf{W}$ to compensate for dropout.
- Hidden: dropout fraction 0.5
  Input: dropout fraction 0.2
- Theoretical links to model averaging.

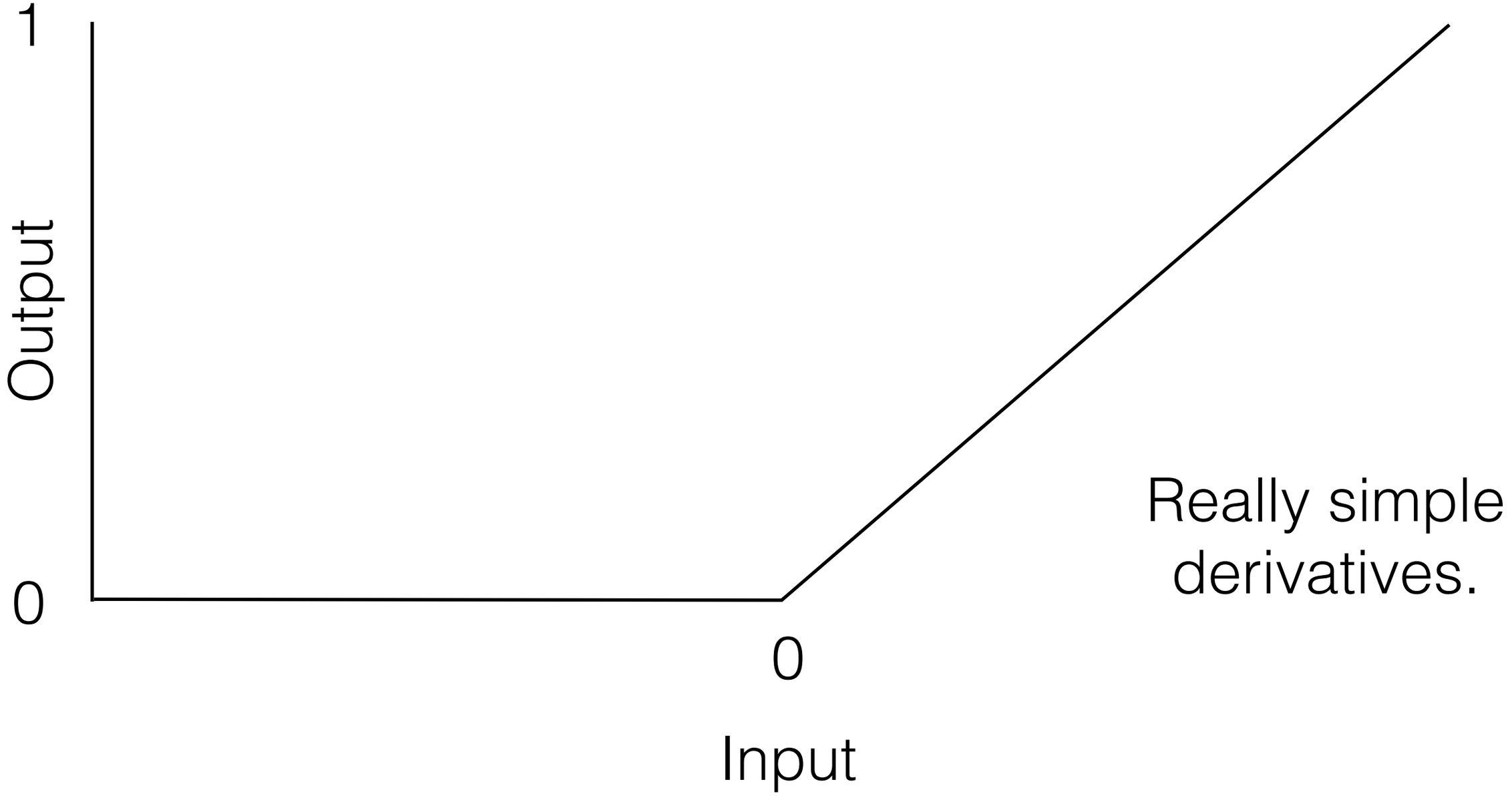
Nonlinear transformations

Sigmoid
Tanh(input)



Nonlinear transformations

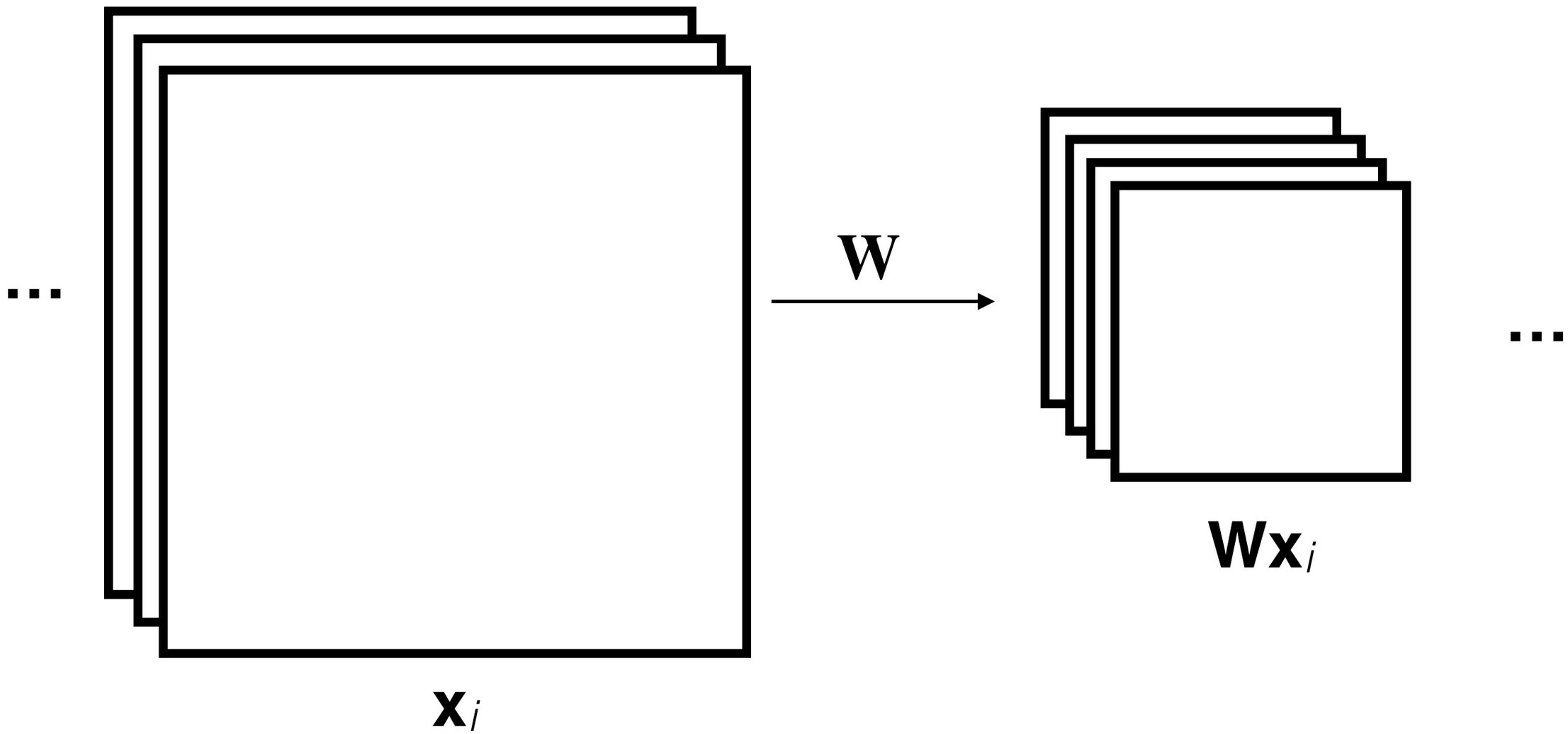
Rectified Linear Unit (ReLU)
max(input, 0)



Really simple
derivatives.

Maxout

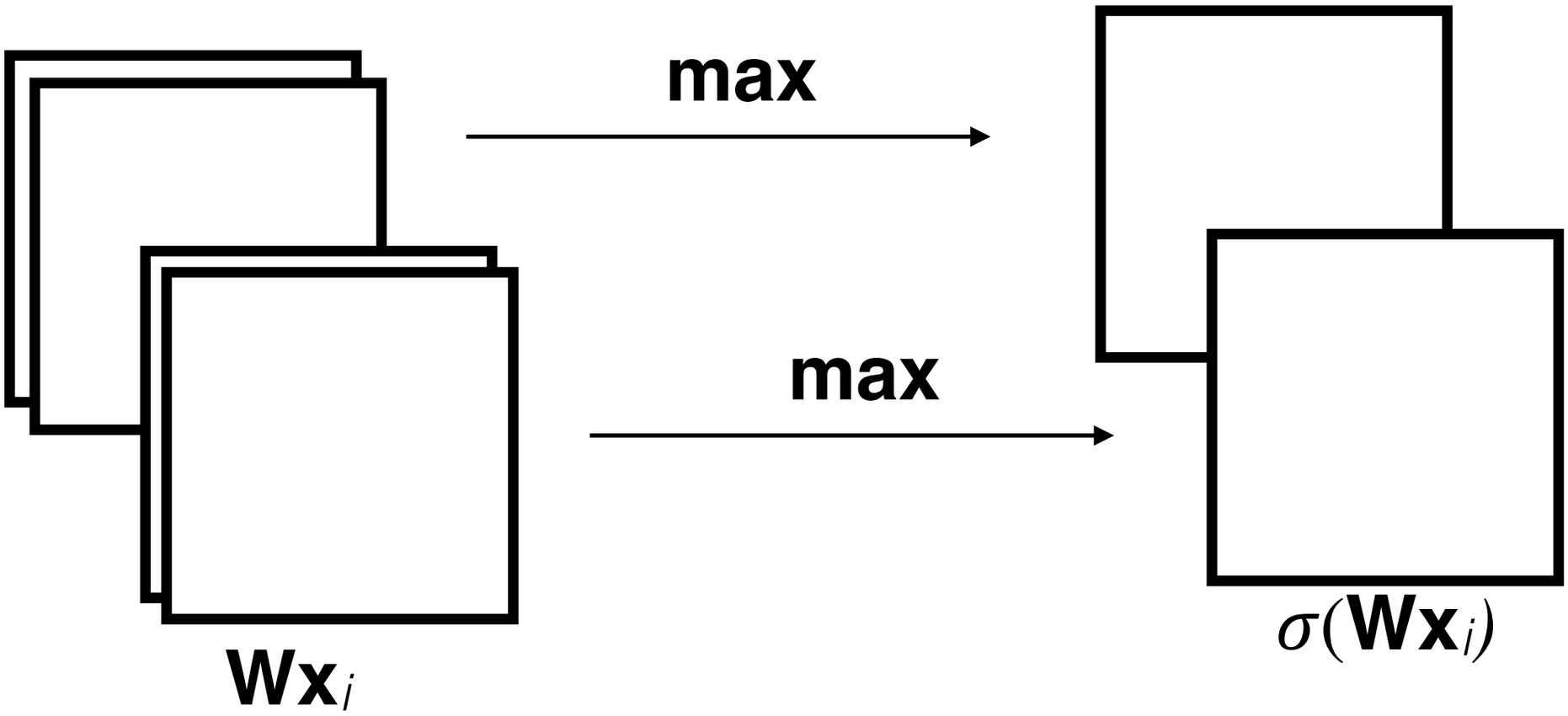
Like maxpooling, but using multiple feature maps.



$$\mathbf{x}_{i+1} = \sigma(\mathbf{W}\mathbf{x}_i + \mathbf{b})$$

Maxout

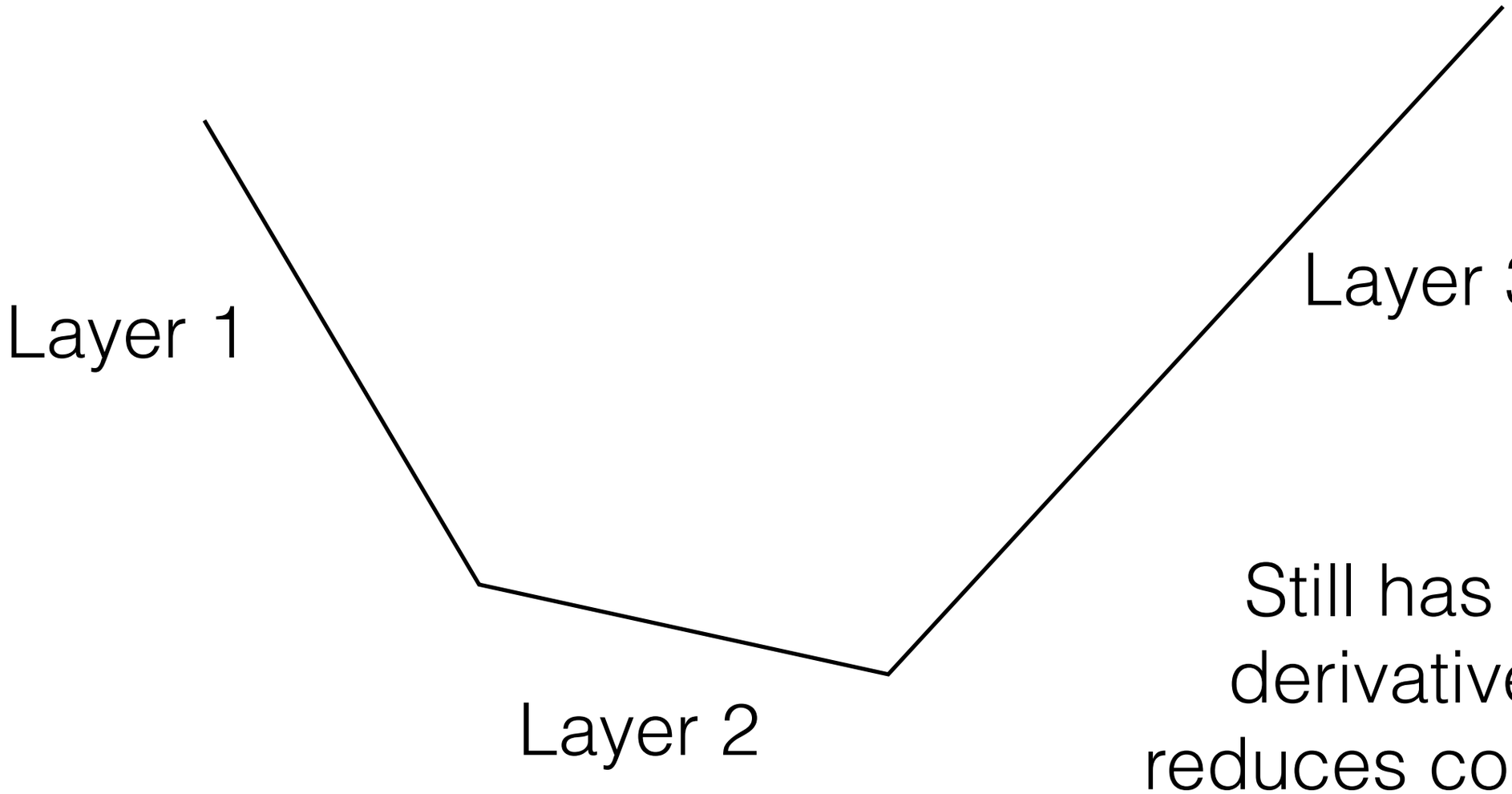
Like maxpooling, but using multiple feature maps.



$$\mathbf{x}_{i+1} = \sigma_i(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i)$$

Maxout

Like ReLU, but with multiple linear functions.



Still has simple derivatives, and reduces computation

Other systems

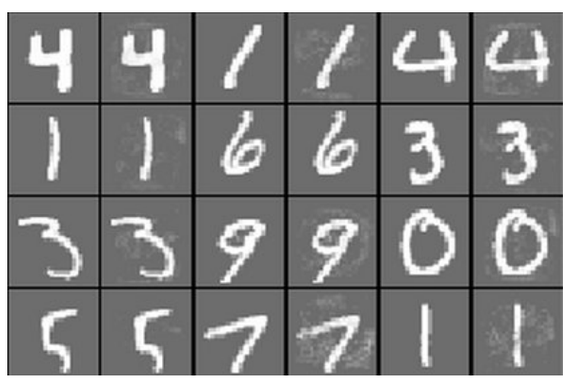
- Pylearn2 = Theano++, fast Convolutions, YAML.
- Lasagne = Simple wrapper around Theano (also Keras).
- Caffe = C++, vision-centric, limited flexibility **(*)**.
- torch = Lua, flexible, has torch-autogrid.
- TensorFlow = Python, flexible like Theano **(*)**.

Linearly Separable Filters

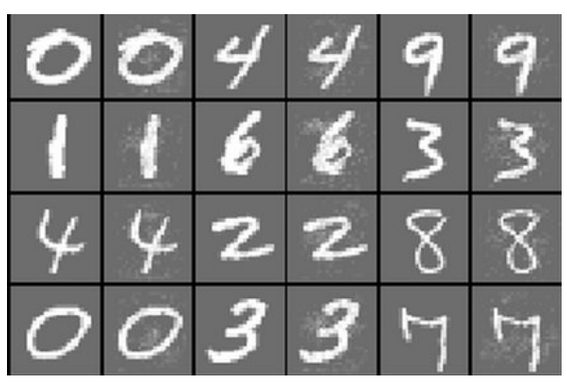
- CNNs are slow! (compared to many other techniques)
- Many 2D filters are “separable” (computable as 1D filters in X then Y).
- This is much faster.
- Sironi et al., 2014 show how to approximate learned filters with sums of a few separable ones.

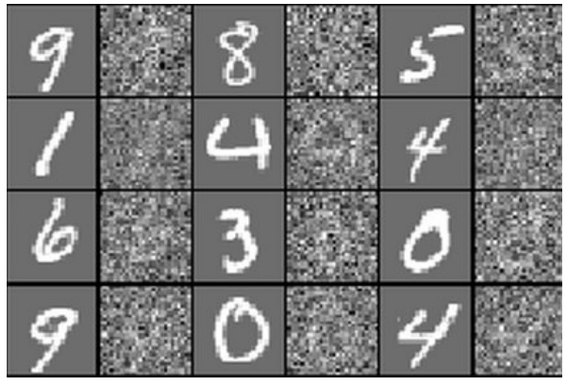
“Intriguing Properties of Neural Networks”

Classification “manifold” is very nonlinear, in CNNs.
Small offsets can cause significant label changes.



Even columns:
adversarial examples
(0% accuracy)

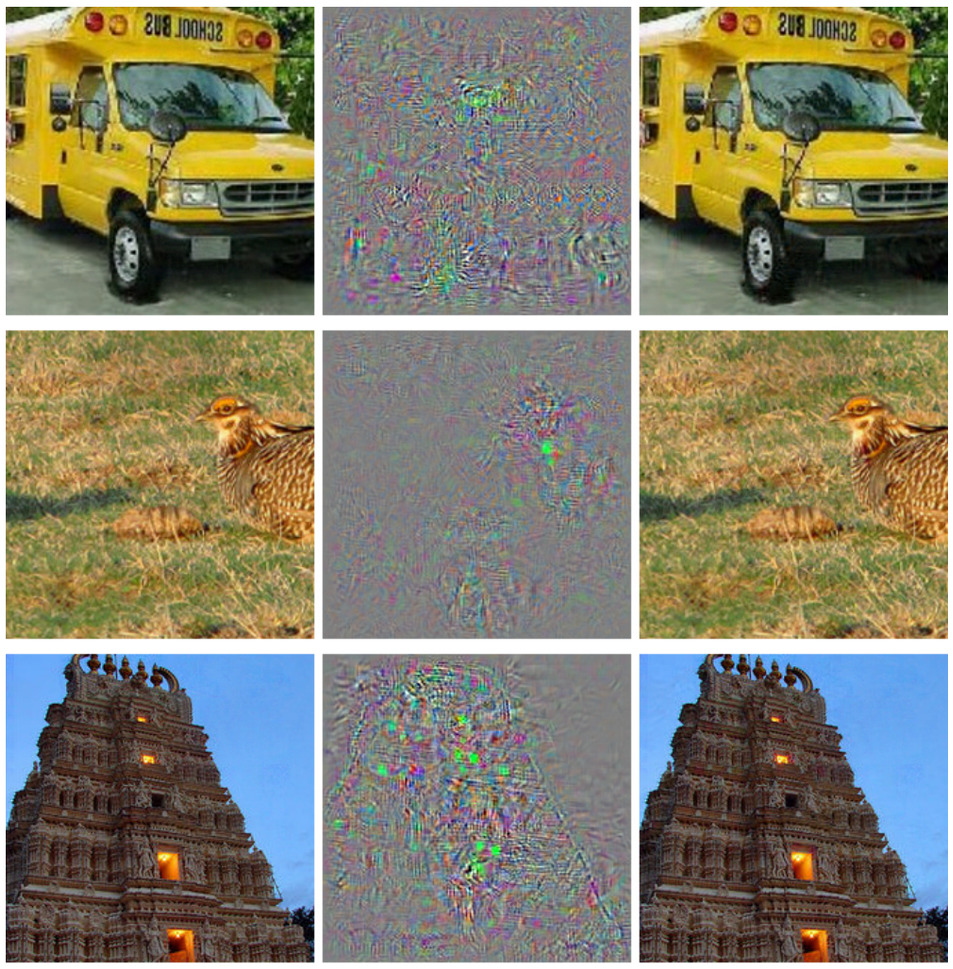


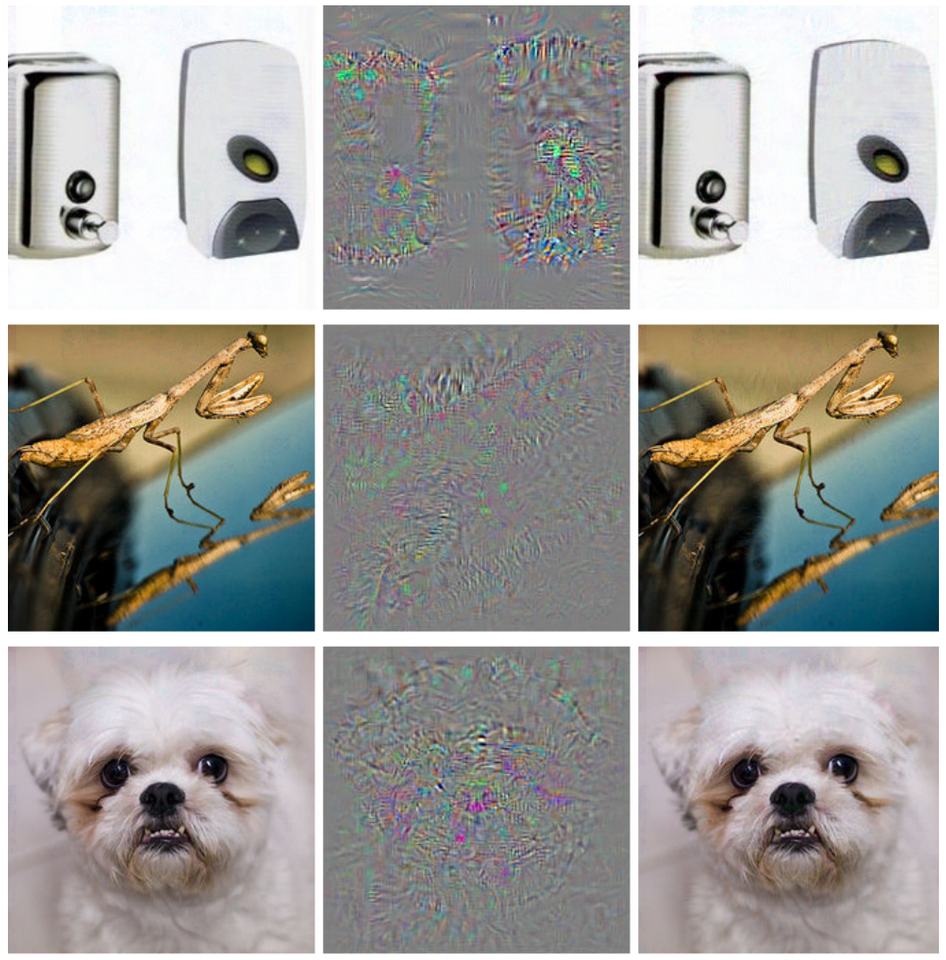


Even columns:
Gaussian noise
(50% accuracy)

Szegedy et al., 2014

“Intriguing Properties of Neural Networks”





Left: correctly identified
Center: added noise x10,
Right: “Ostrich”

Szegedy et al., 2014

Thank You!