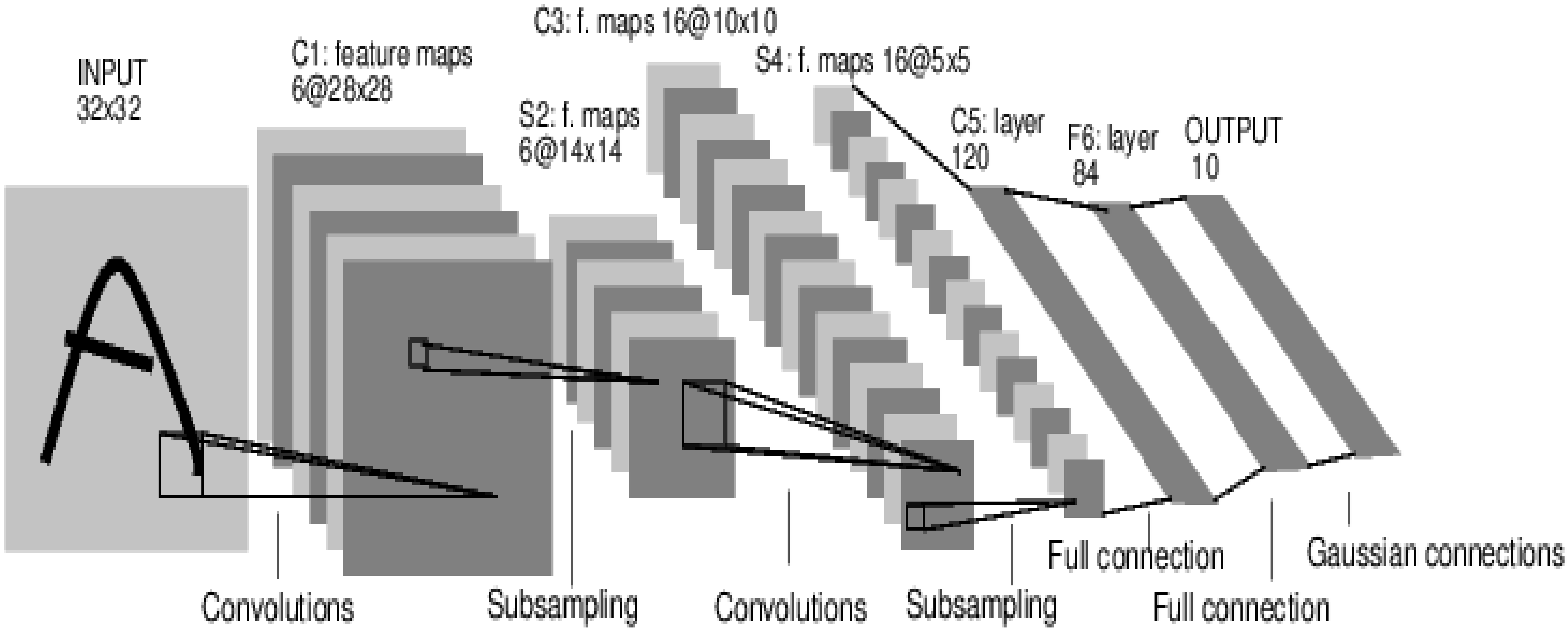
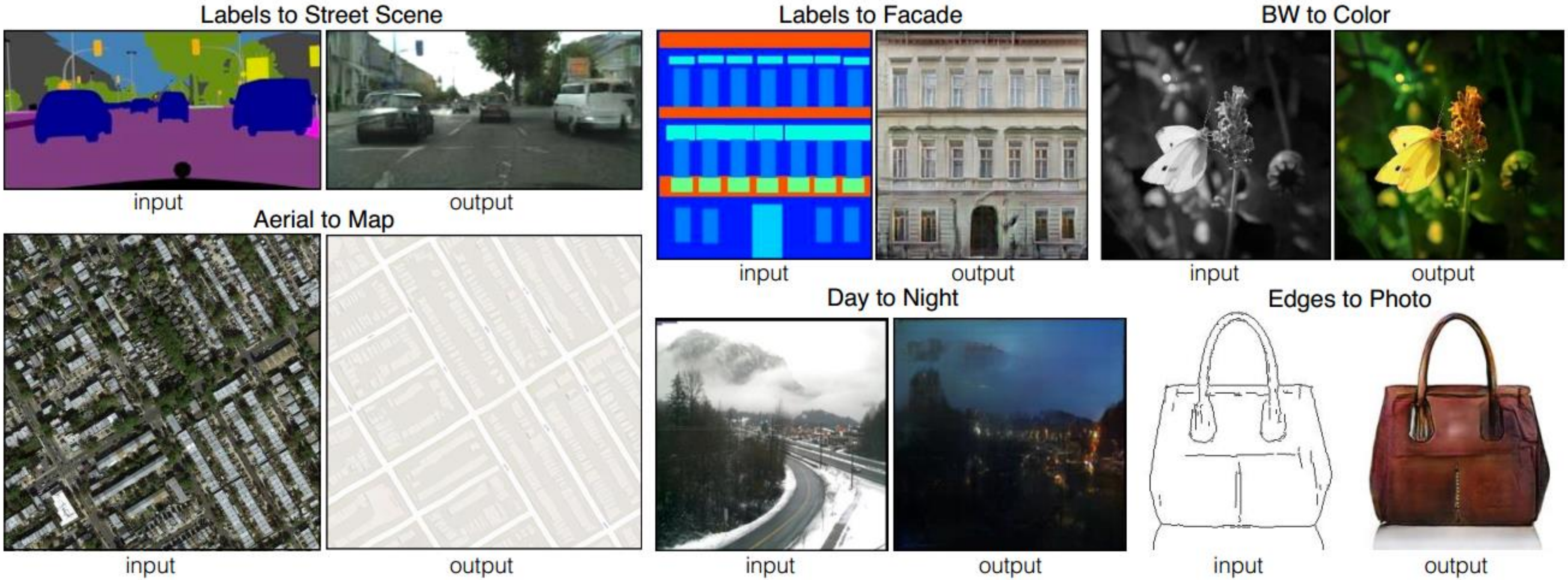
ConvNets, continued



Alexei Efros, CS280, Spring 2024

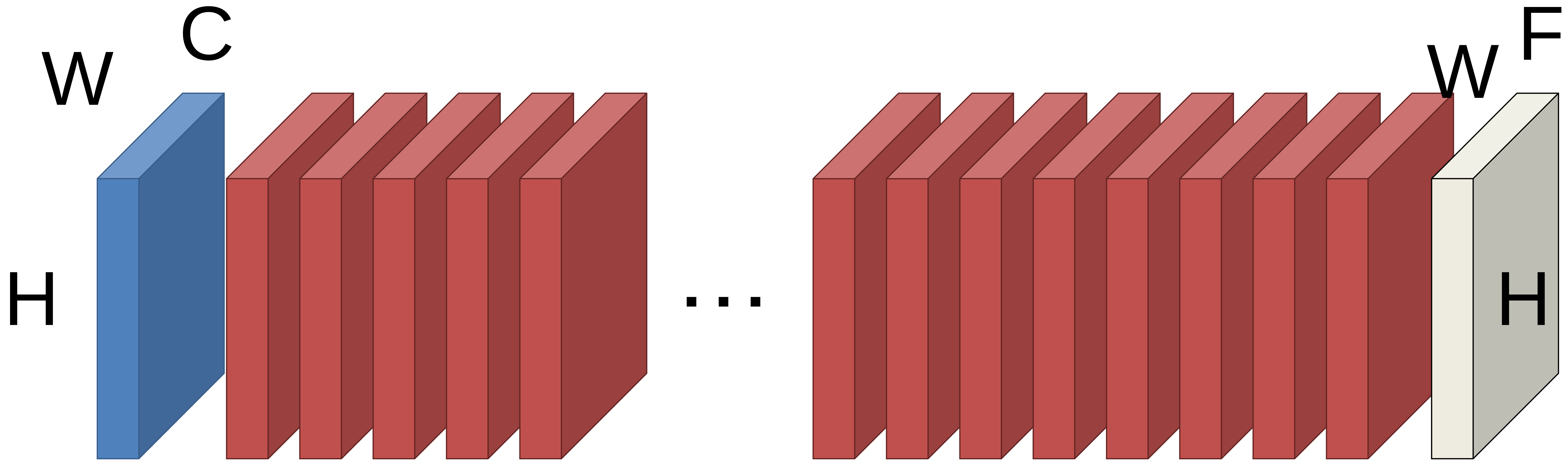
Generic Image-to-Image Translation



First – Two “Wrong” Ways

- It’s helpful to see two “wrong” ways to do this.

Why Not Stack Convolutions?

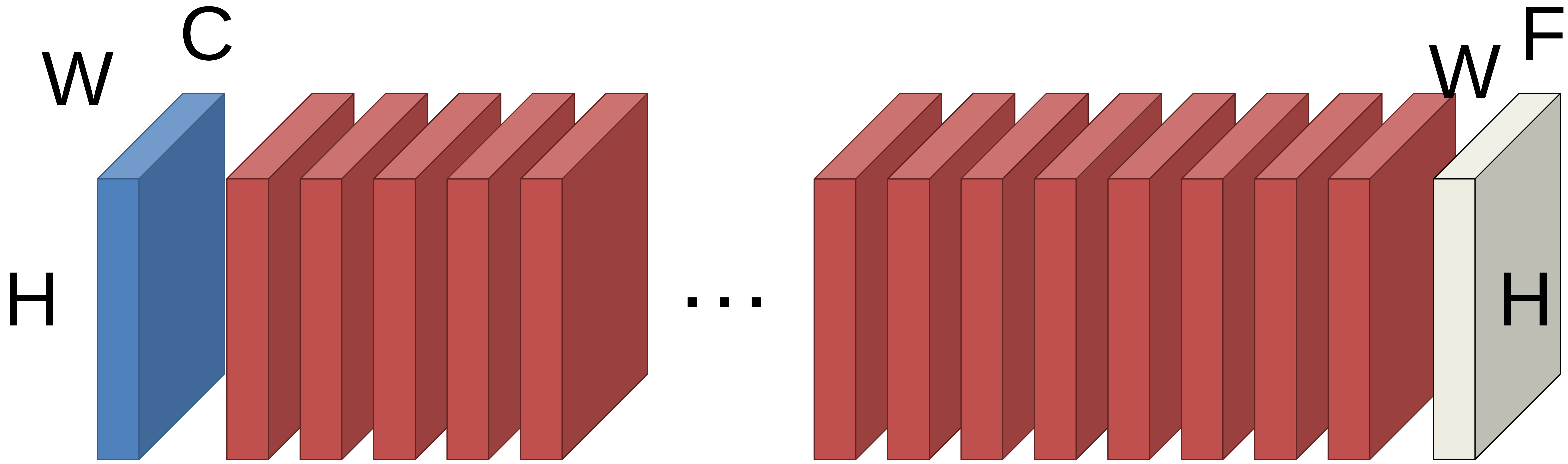


n 3x3 convs have a receptive field of 2n+1 pixels

How many convolutions until >=200 pixels?

100

Why Not Stack Convolutions?



Suppose 200 3x3 filters/layer, H=W=400

Storage/layer/image: 200 * 400 * 400 * 4 bytes = 122MB

Uh oh!*

*100 layers, batch size of 20 = 238GB of memory!

Idea #2

Crop out every sub-window and predict the label in the middle.

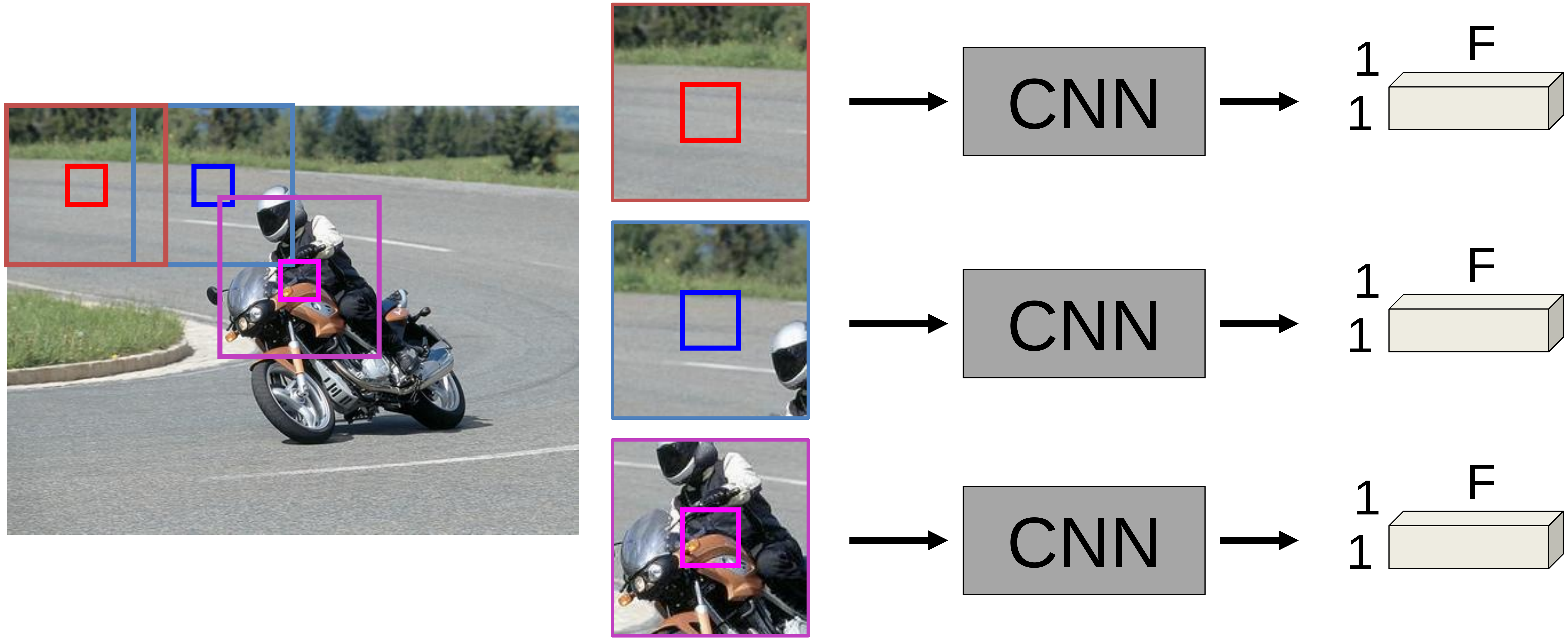
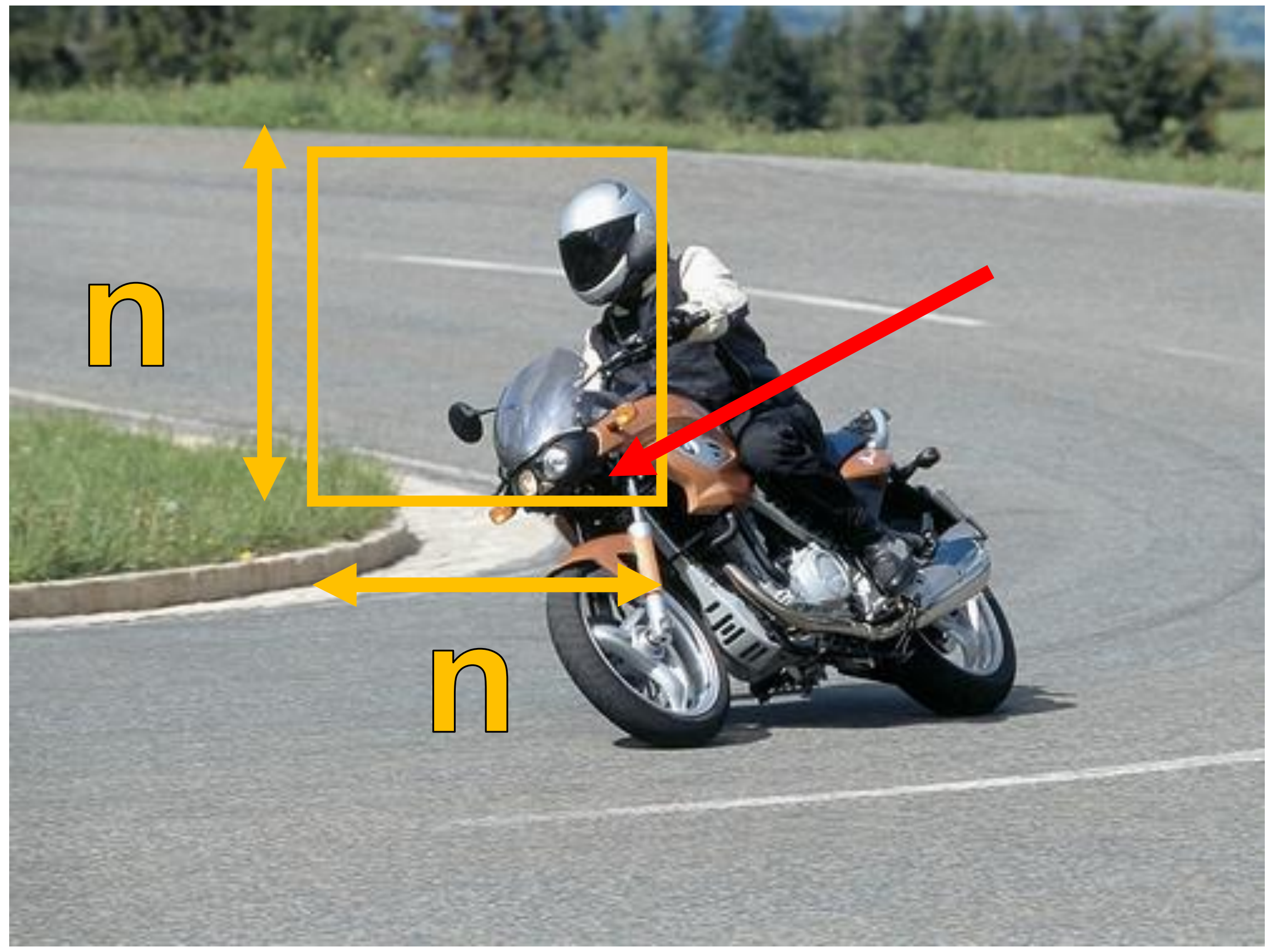


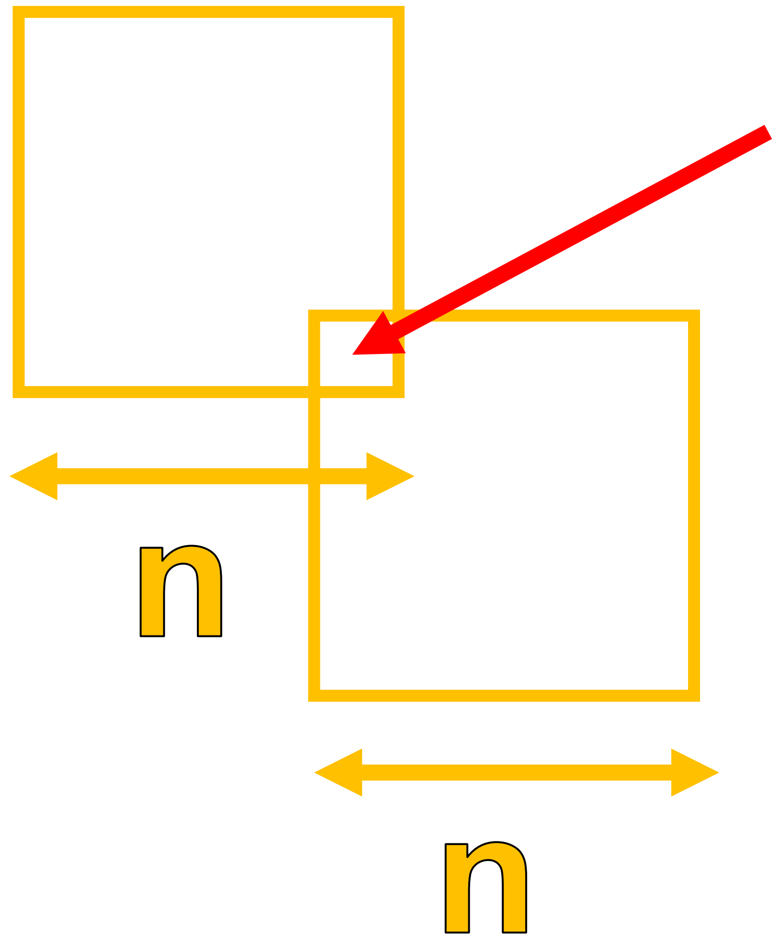
Image credit: PASCAL VOC, Everingham et al.

Idea #2

Meet “Gabor”. We extract NxN patches and do independent CNNs. **How many times does Gabor filter the red pixel?**

Gabor





Answer:
(2n-1)*(2n-1)

Image credit: PASCAL VOC, Everingham et al.

The Big Issue

We need to:

1. Have large receptive fields to figure out what we’re looking at
2. Not waste a ton of time or memory while doing so

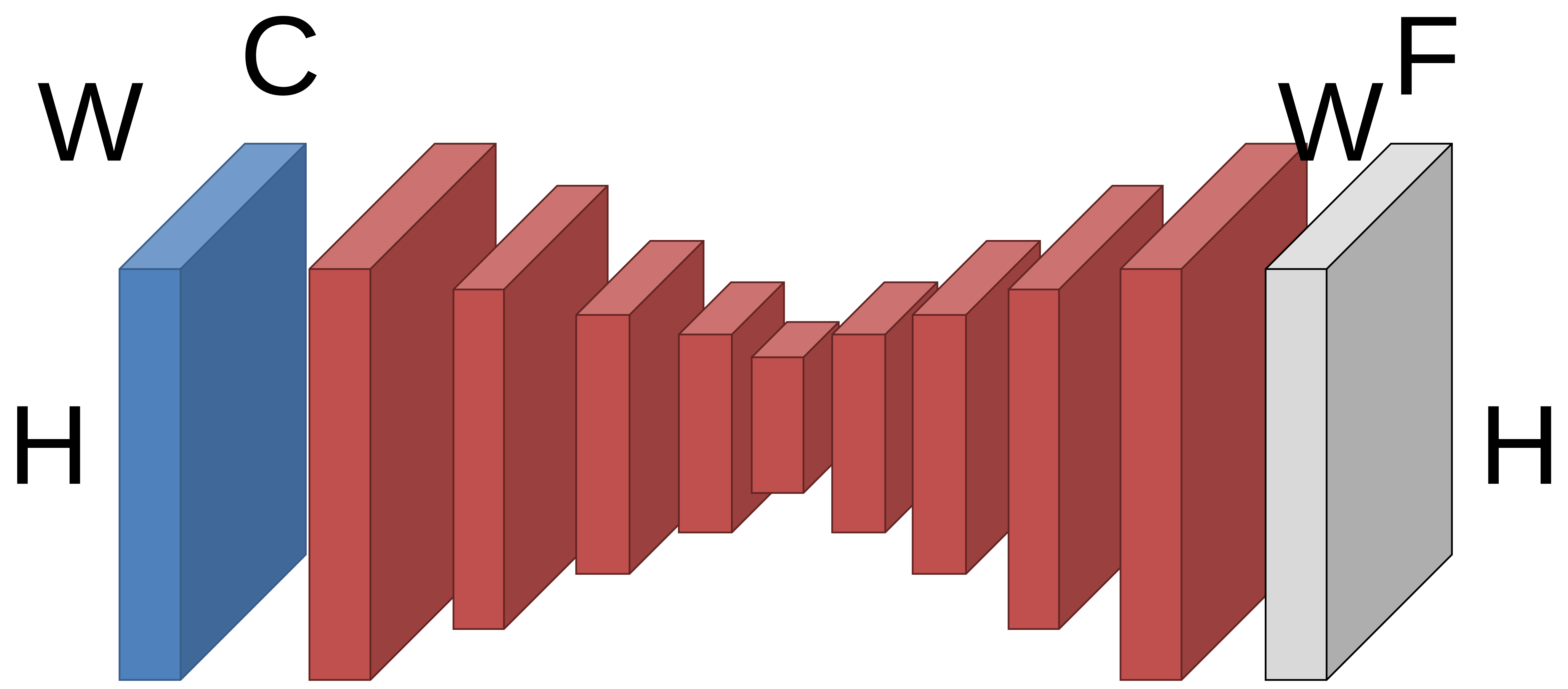
These two objectives are in total conflict

Encoder-Decoder

Key idea: First **downsample** towards middle of network. Then **upsample** from middle.

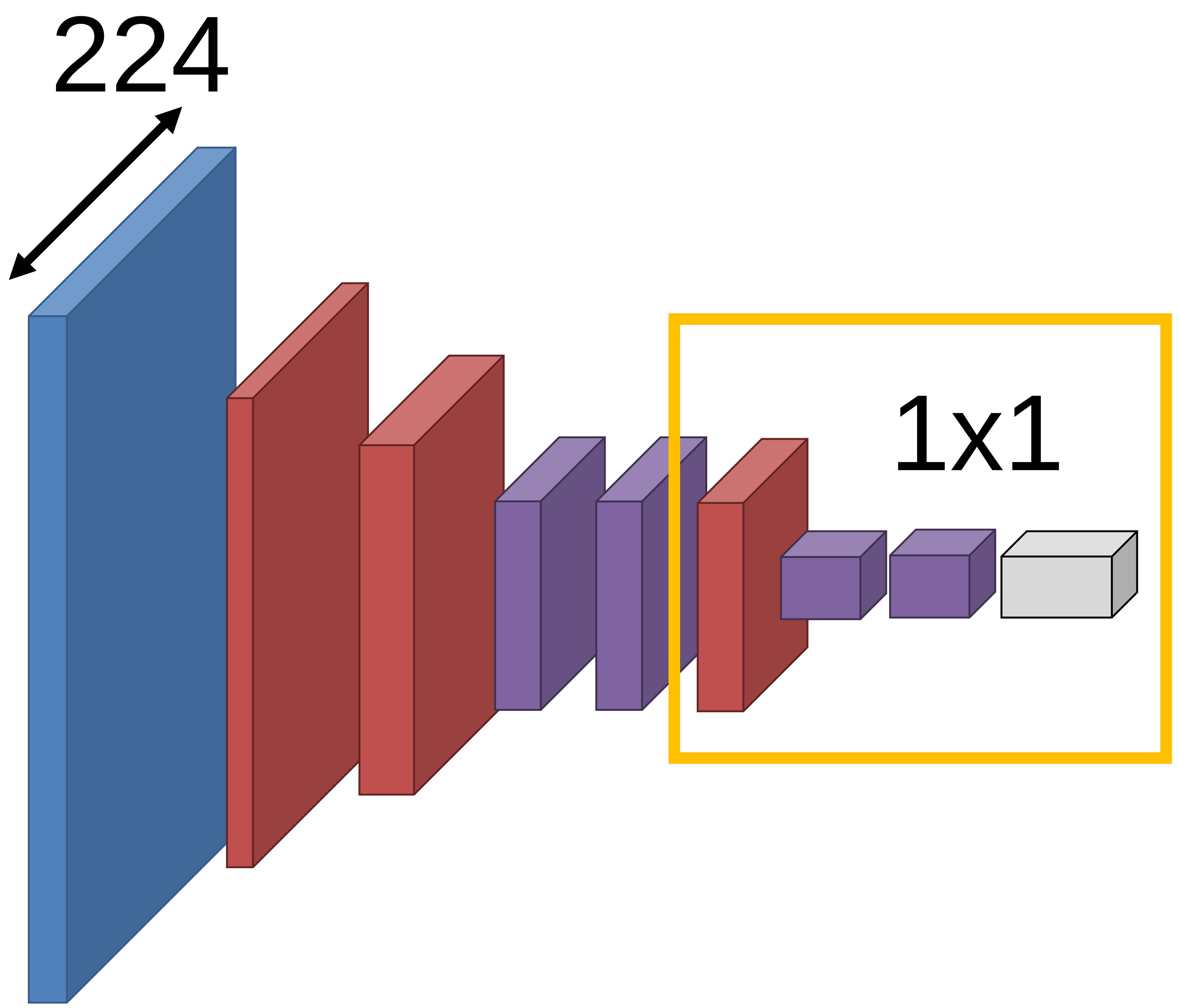
How do we downsample?

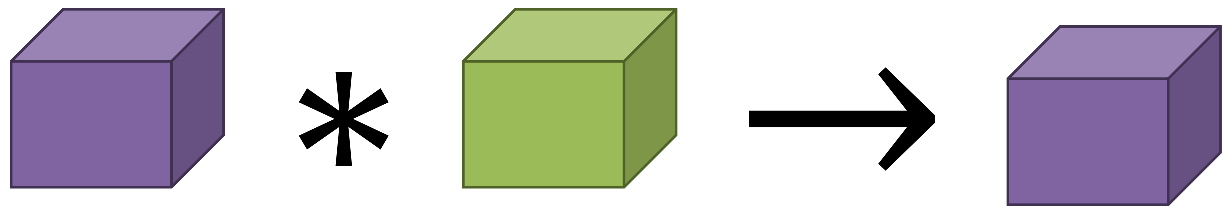
Convolutions, pooling



Where Do We Get Parameters?

Convnet that maps images to vectors



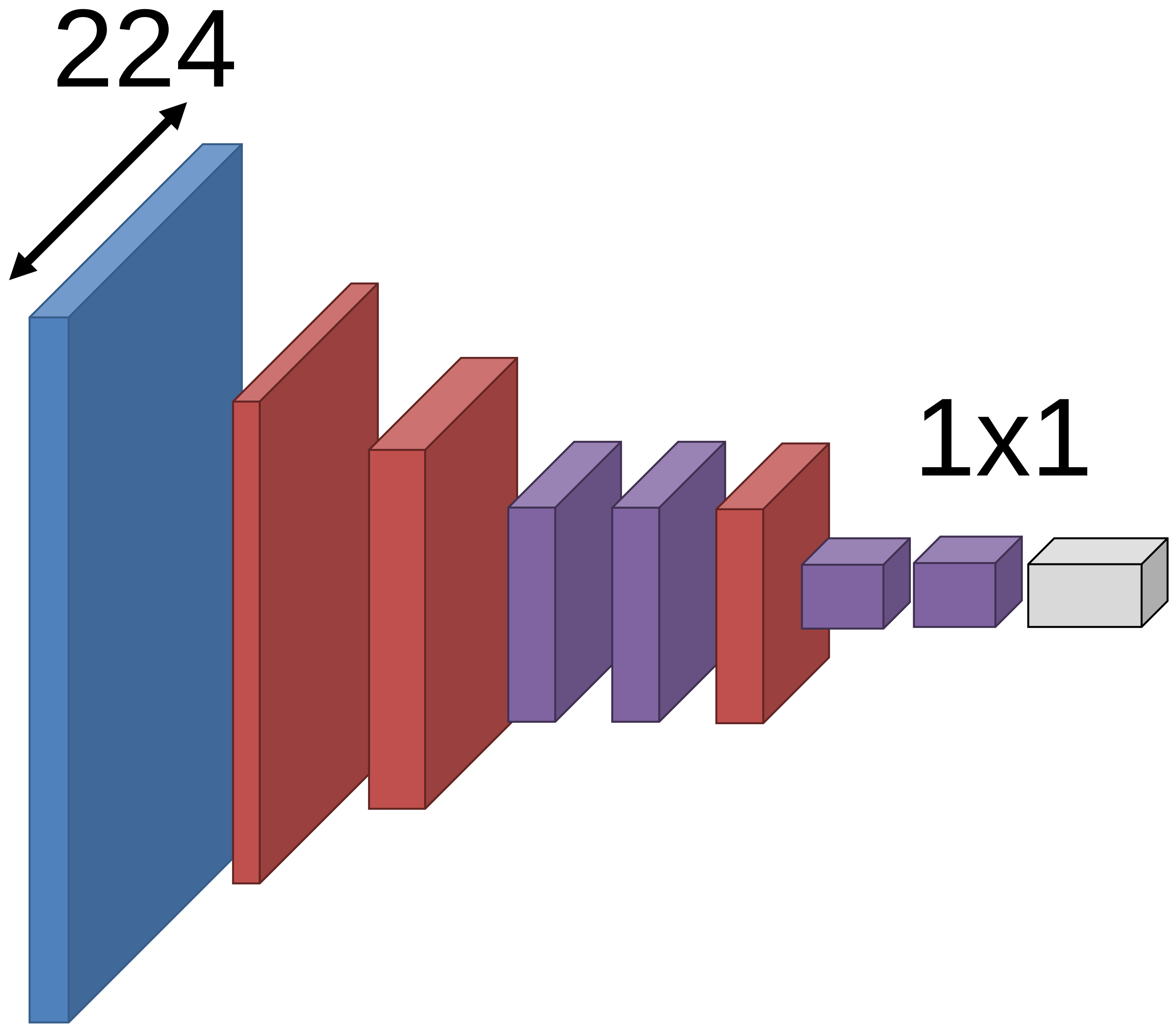


Recall that we can rewrite any vector-vector operations via 1x1 convolutions

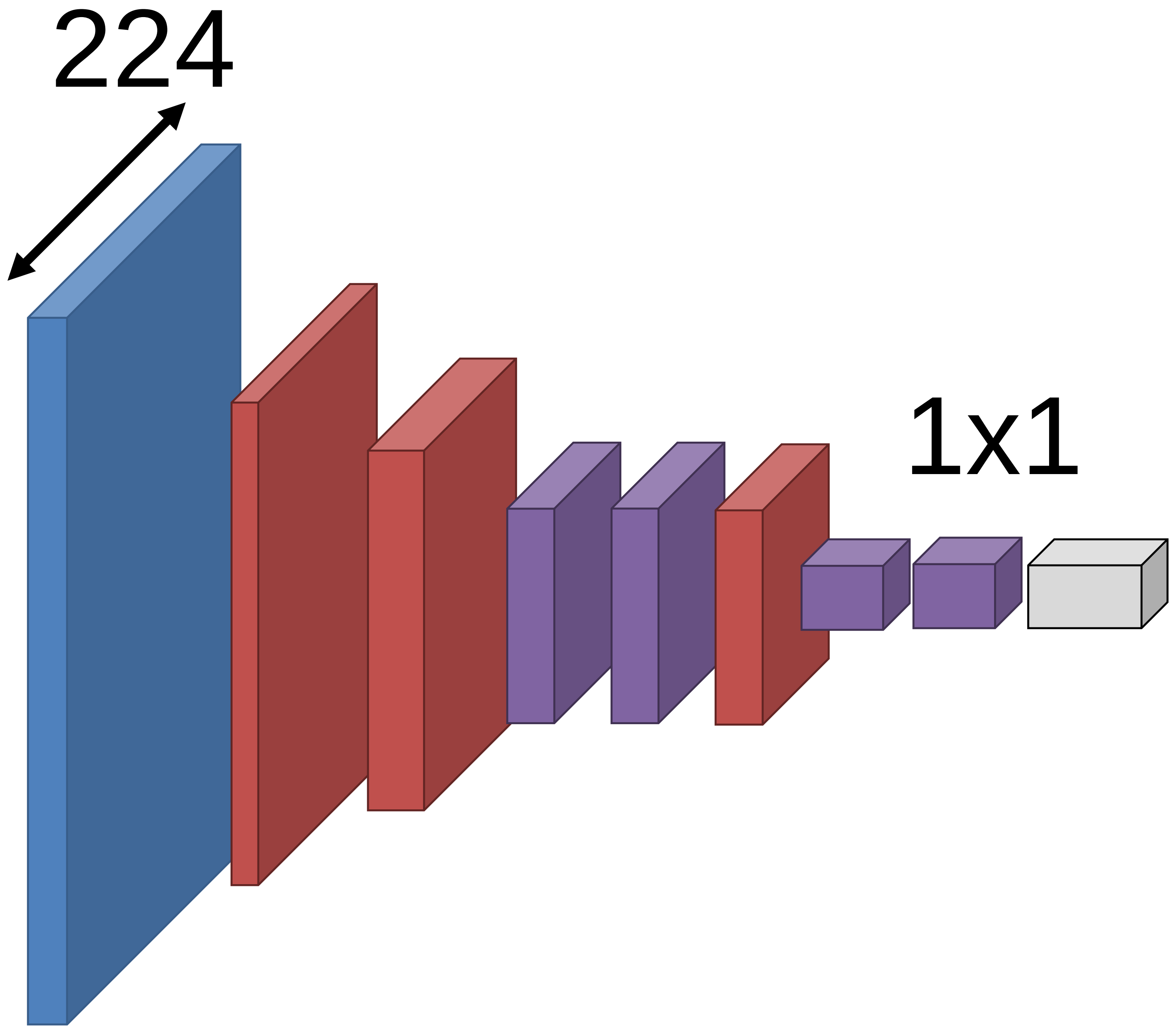
Long et al. *Fully Convolutional Networks For Semantic Segmentation*. CVPR 2014

Where Do We Get Parameters?

Convnet that maps images to vectors



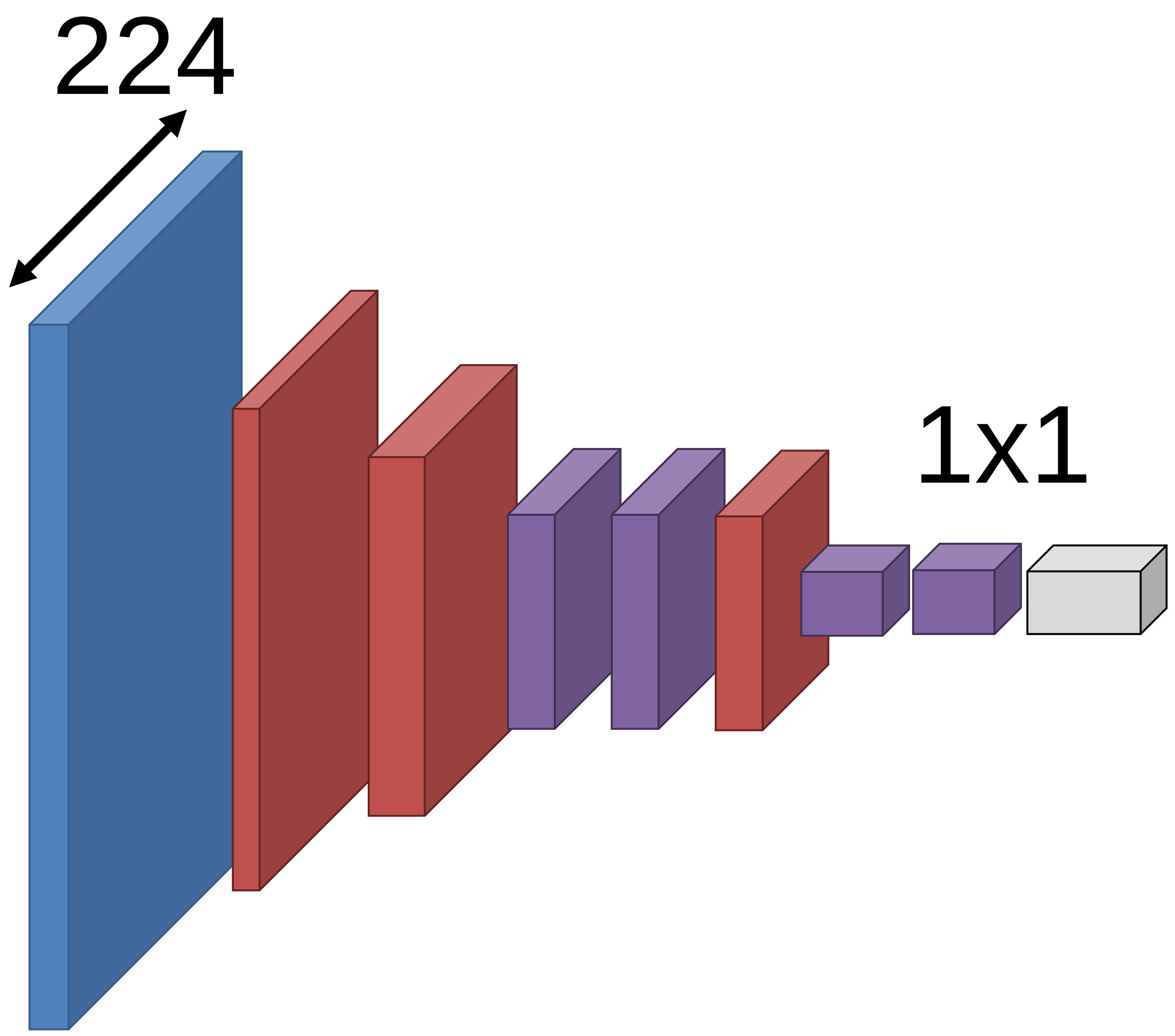
Convnet that maps images to images



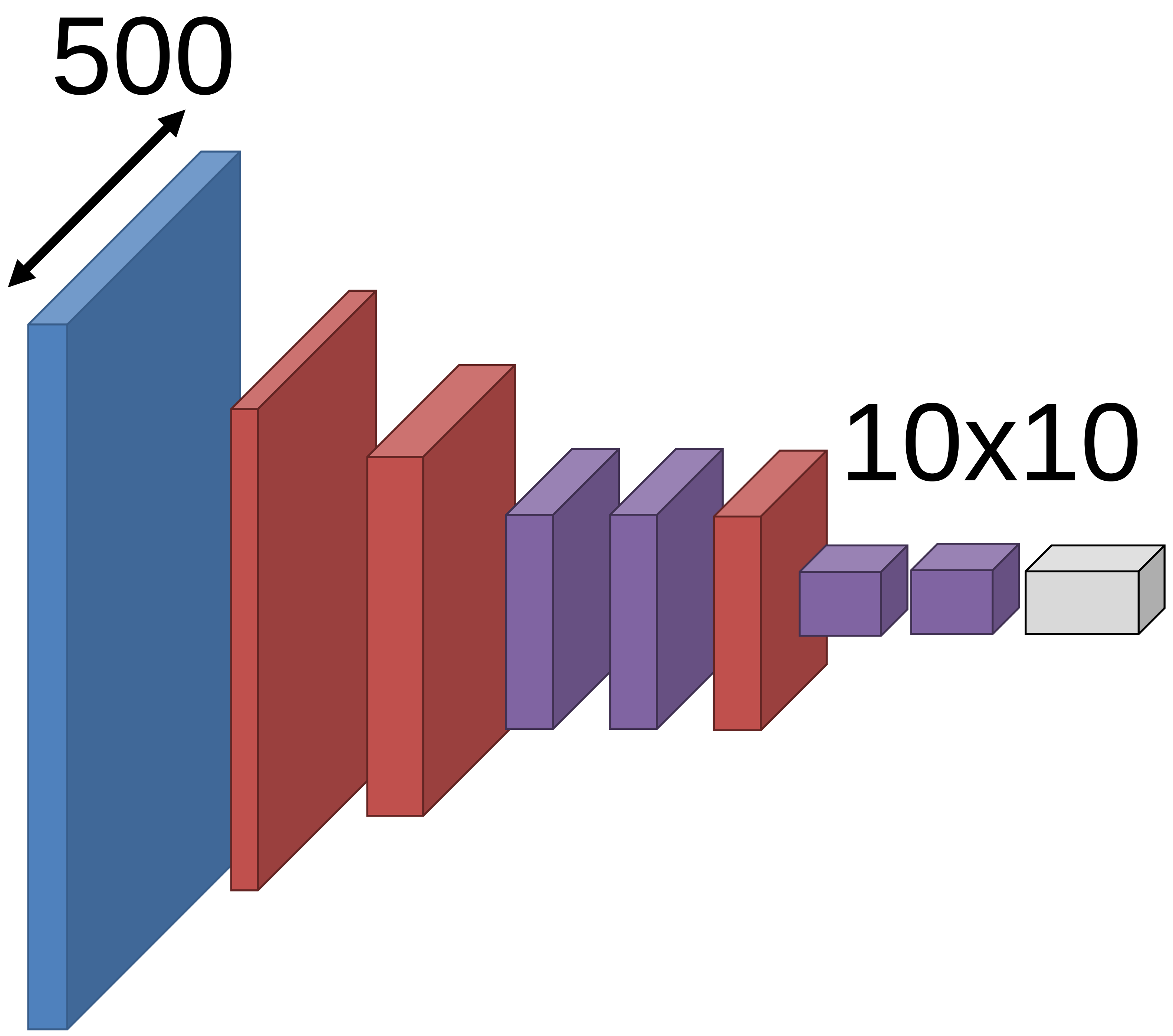
What if we make the input bigger?

Where Do We Get Parameters?

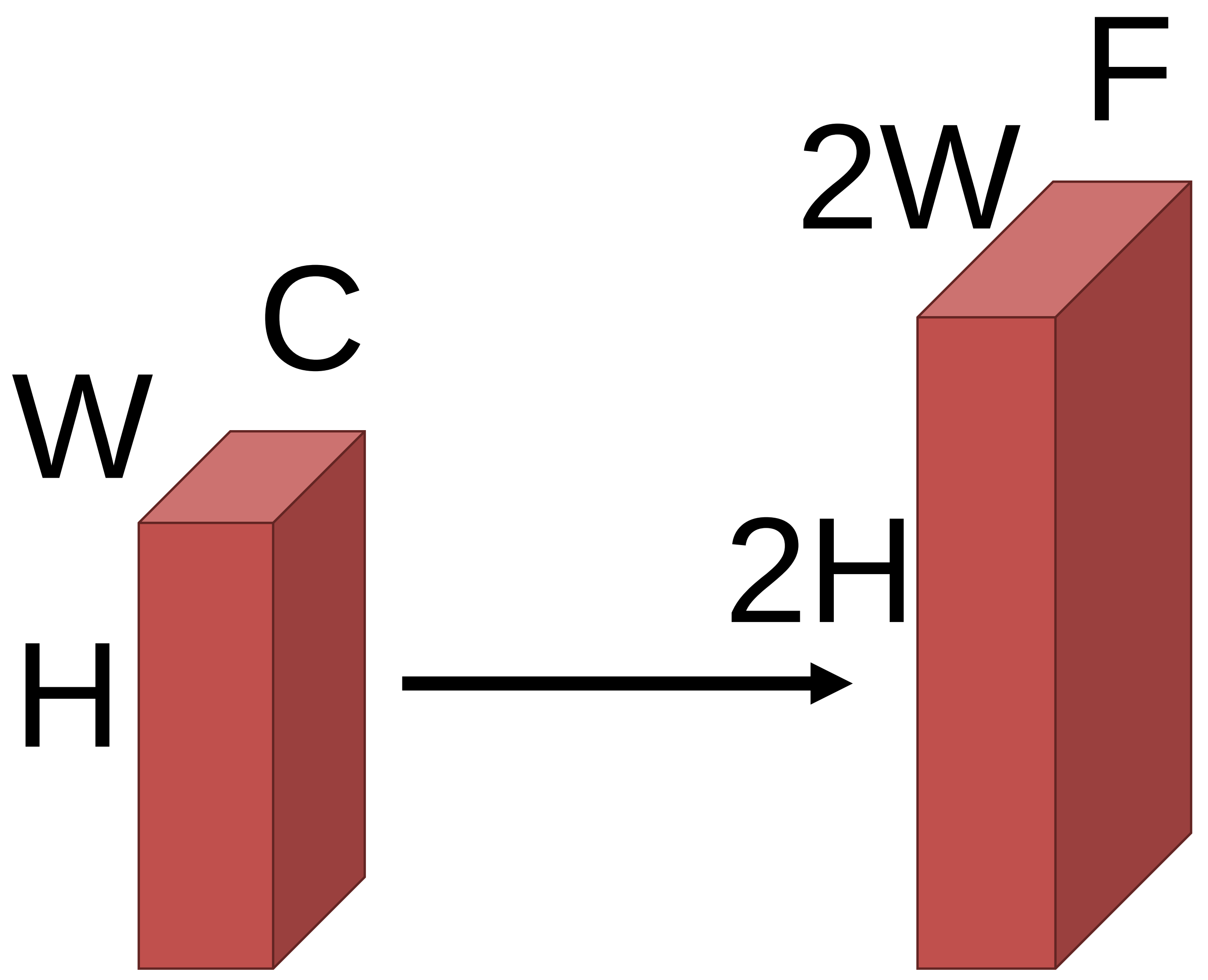
Convnet that maps
images to vectors



Convnet that maps
images to images



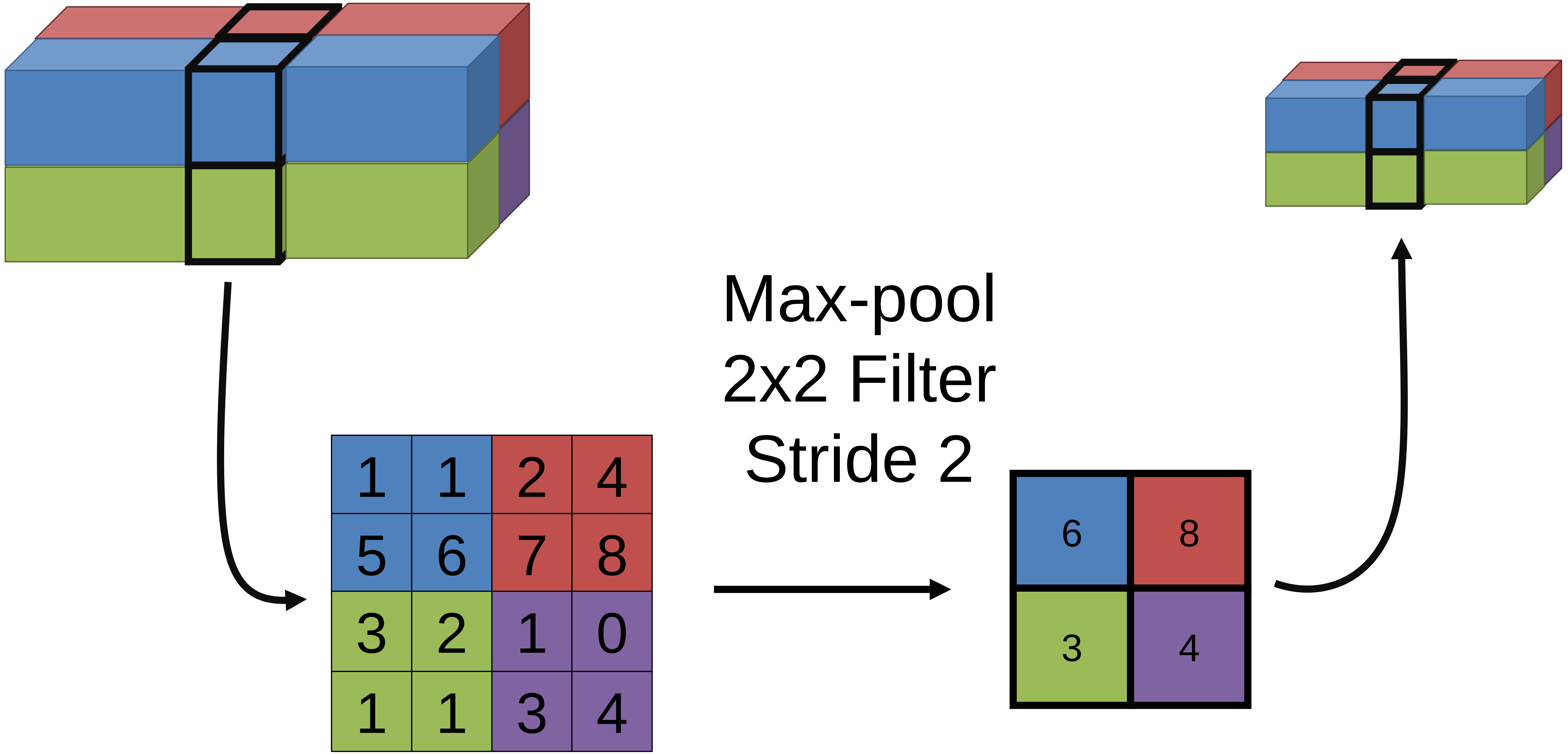
How Do We Upsample?



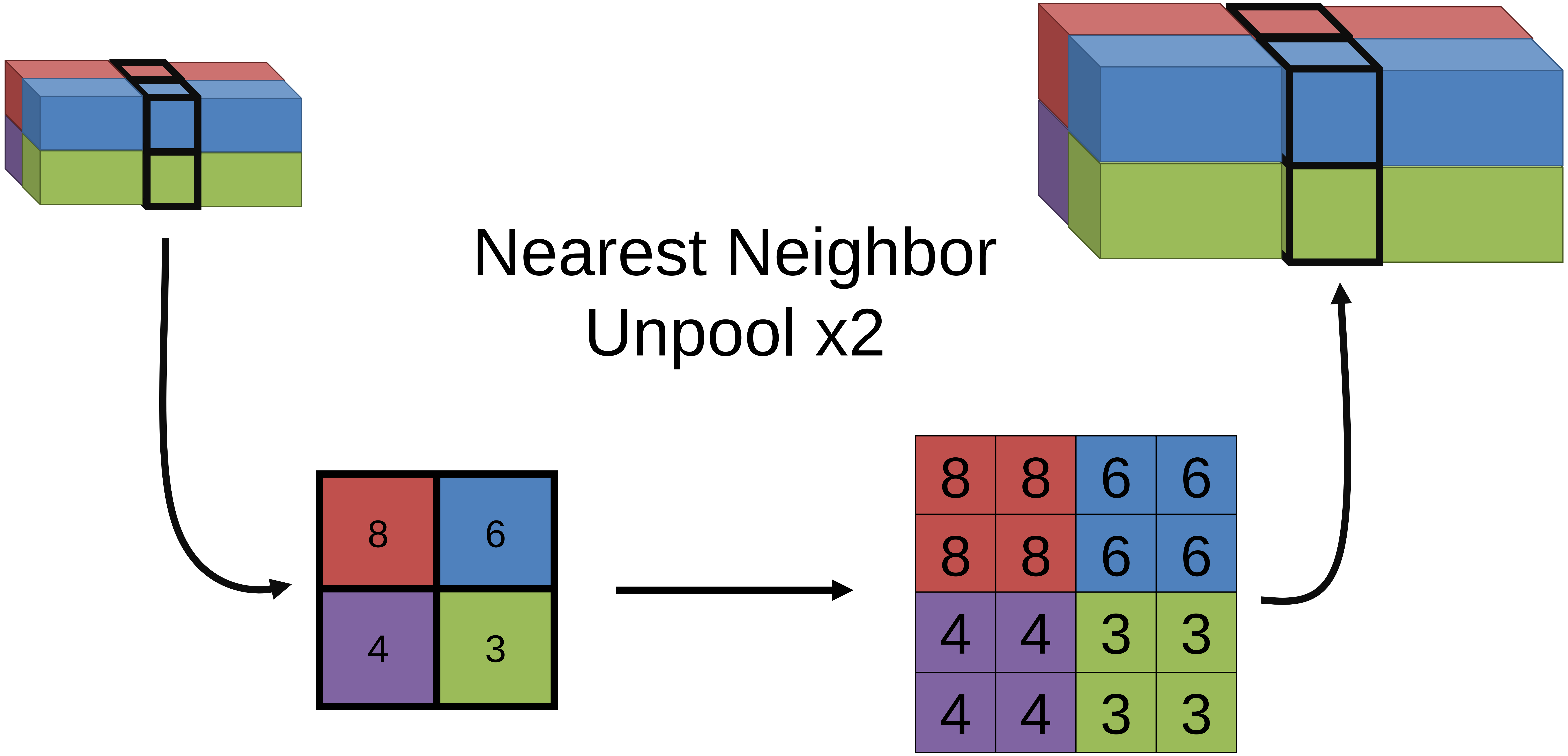
Do the opposite of how we downsample:

1. Pooling → “Unpooling”
2. Convolution → “Transpose Convolution”

Recall: Pooling

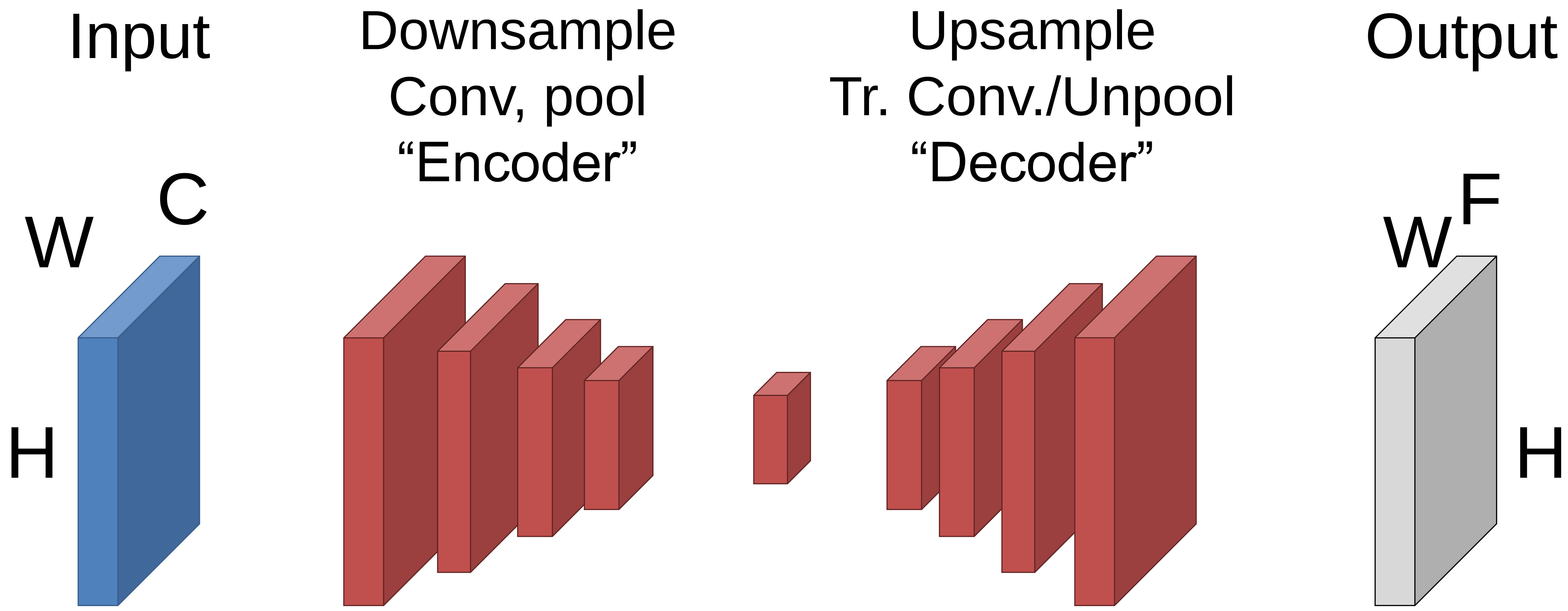


Now: Unpooling



Putting it Together

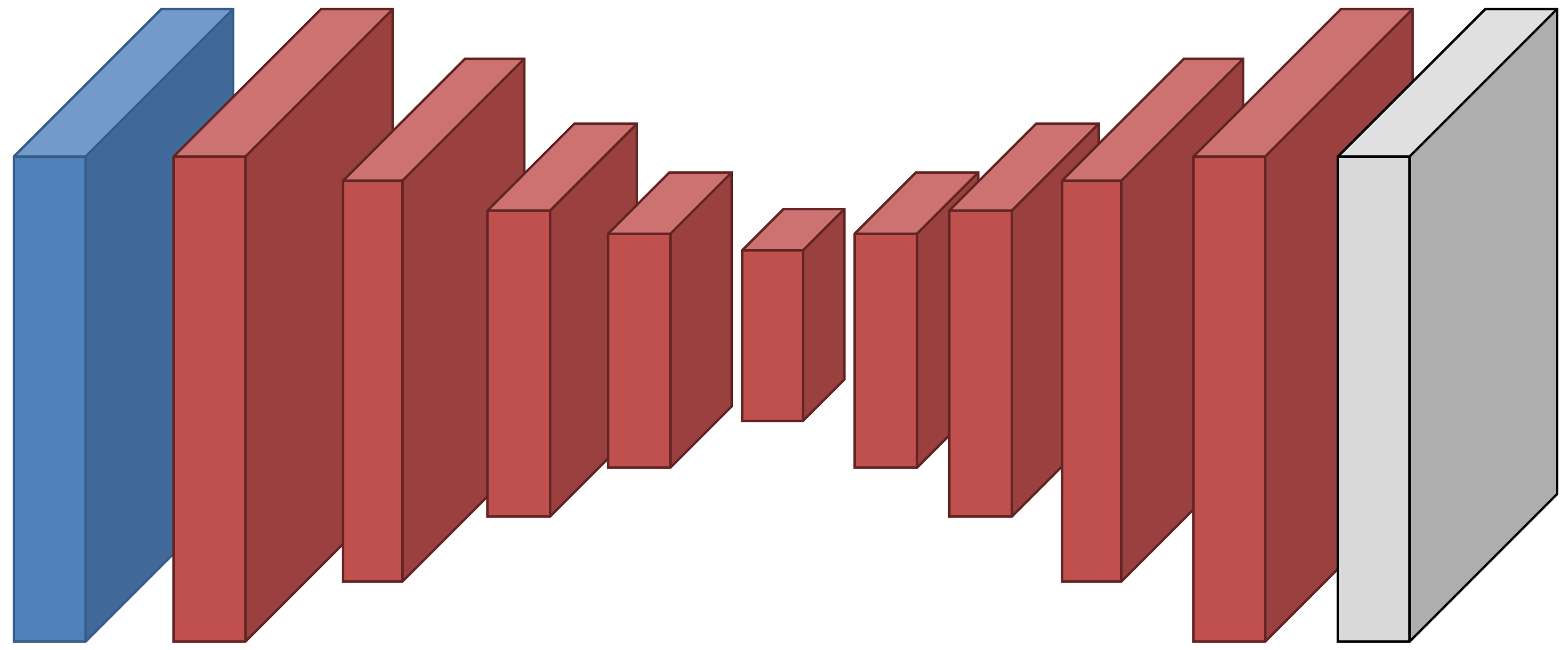
Convolutions + pooling downsample/compress/encode
Transpose convs./unpoolings upsample/uncompress/decode



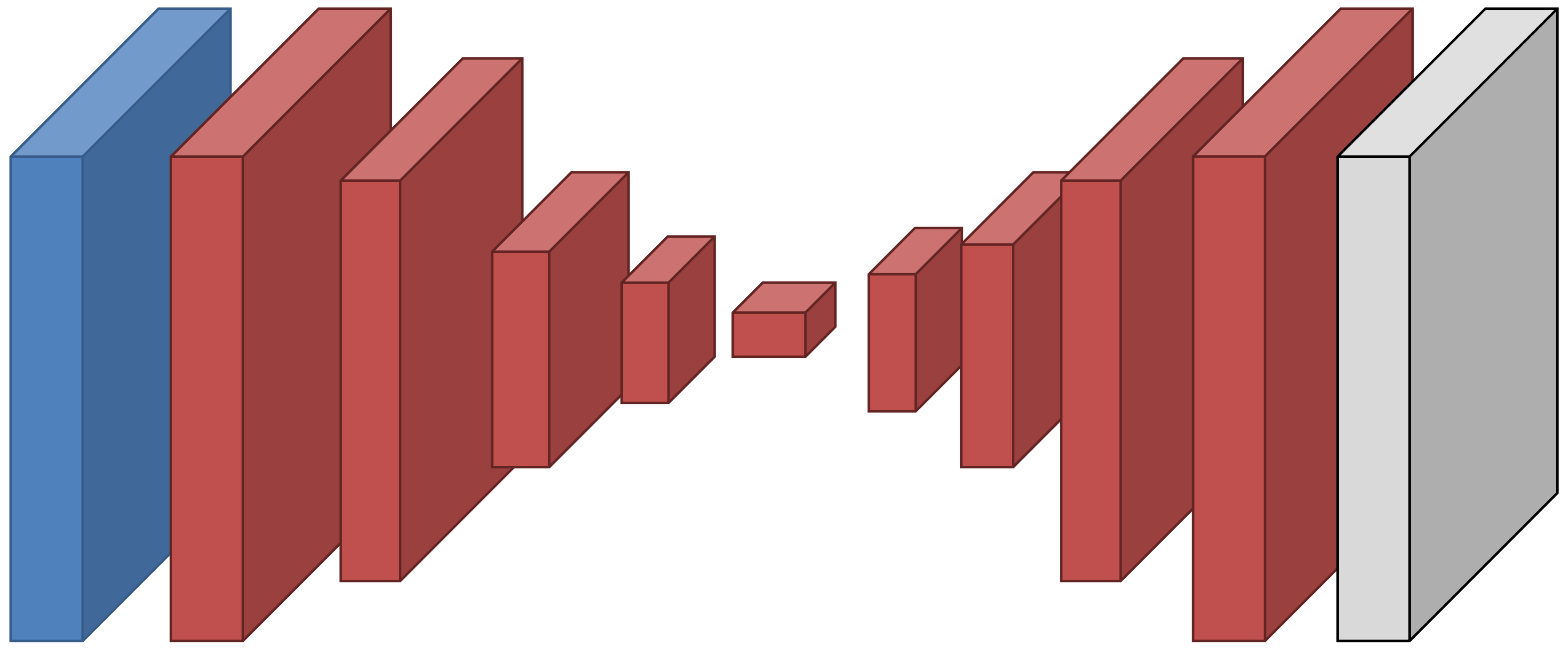
Putting It Together – Block Sizes

- Networks come in lots of forms
- **Don't take any block sizes literally.**
- Often (not always) keep some spatial resolution

Encode to spatially smaller tensor, then decode.



Encode to 1D vector then decode

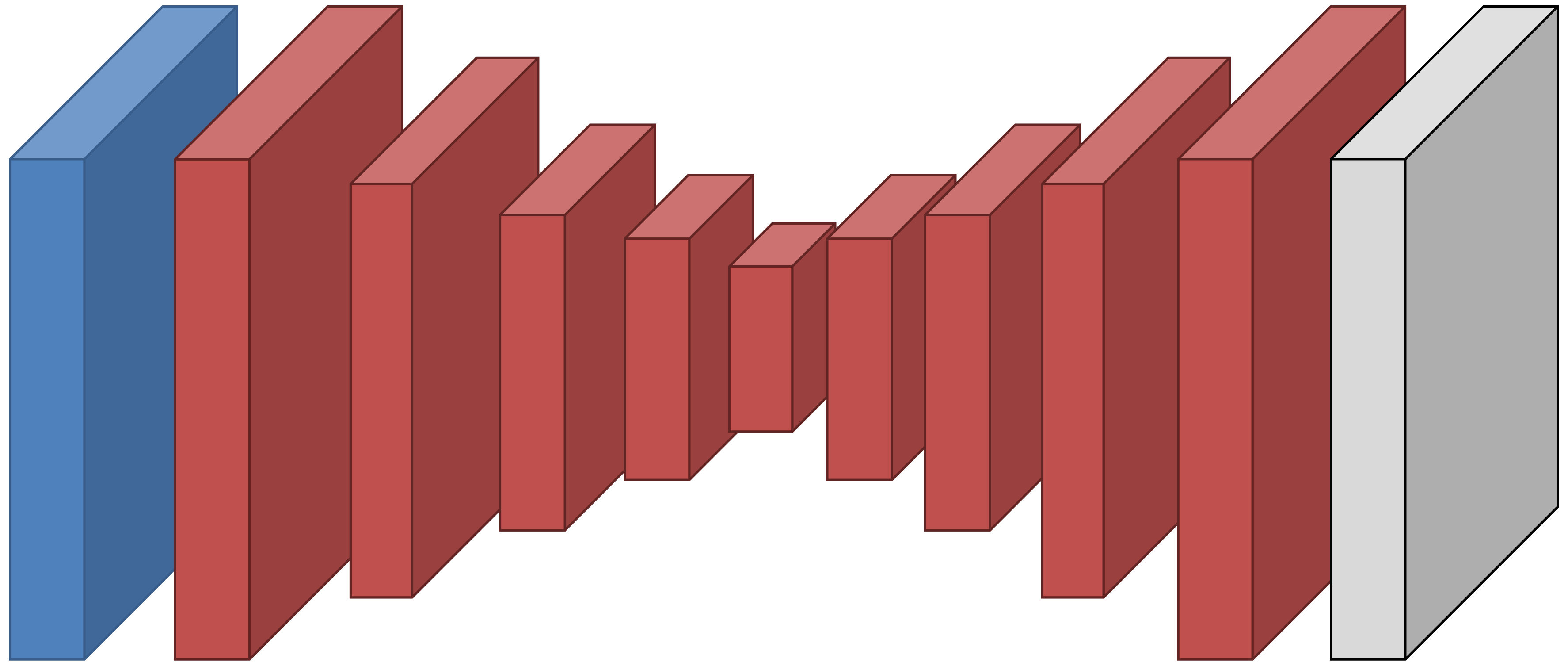


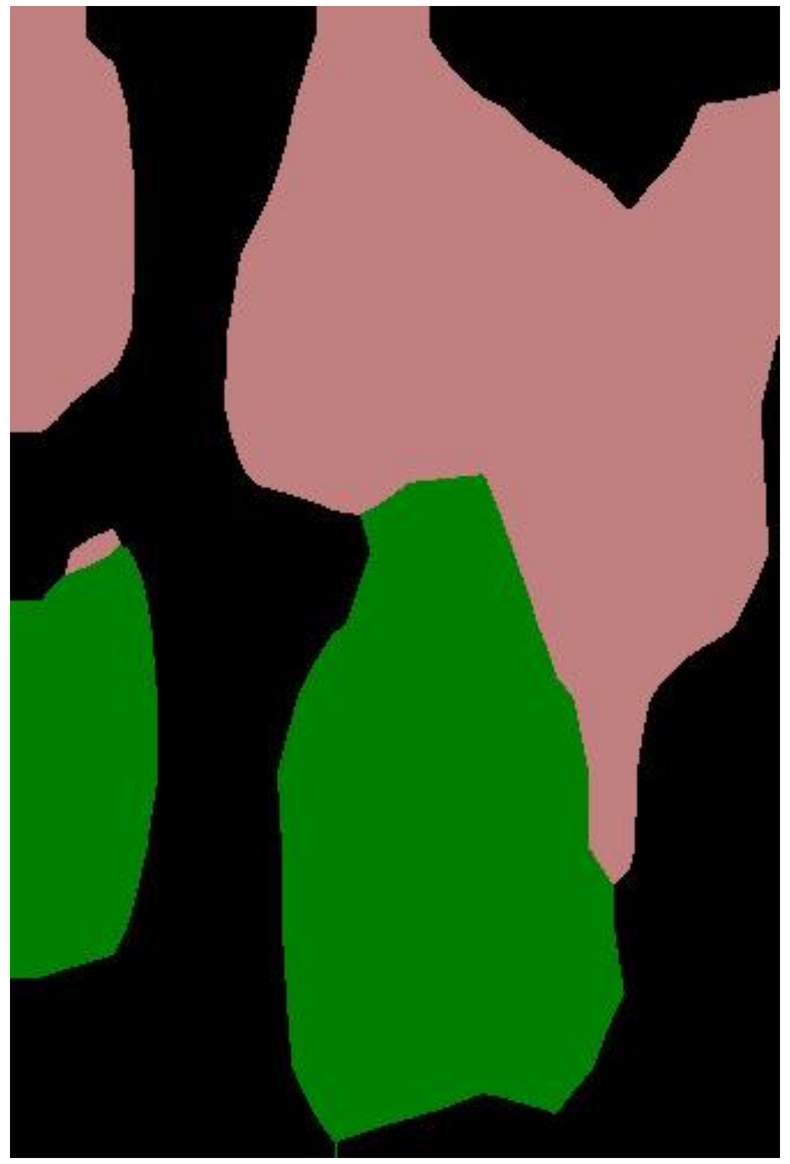
Missing Details

While the output *is* HxW, just upsampling often produces results without details/not aligned with the image.

Why?

Information about details lost when downsampling!

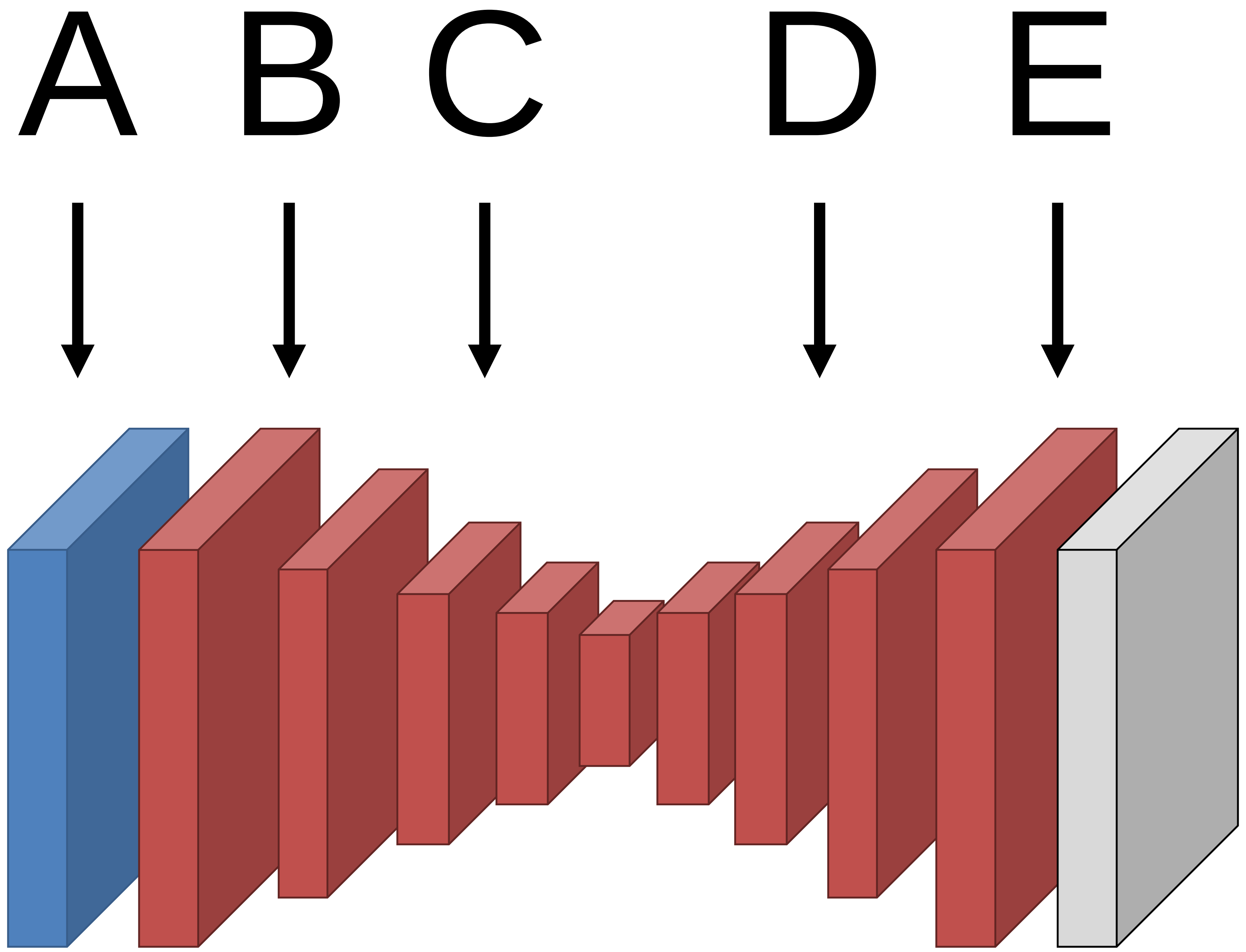


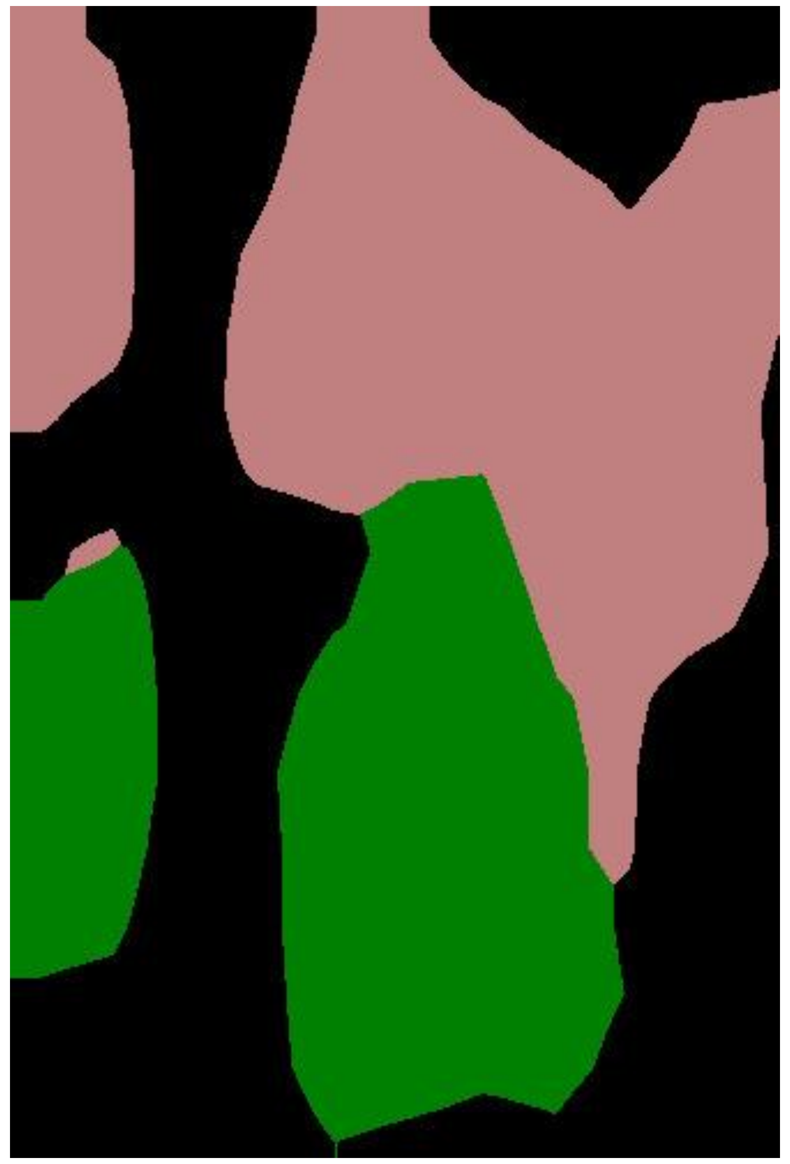


Result from Long et al. *Fully Convolutional Networks For Semantic Segmentation*. CVPR 2014

Missing Details

Where is the useful information about the high-frequency details of the image?





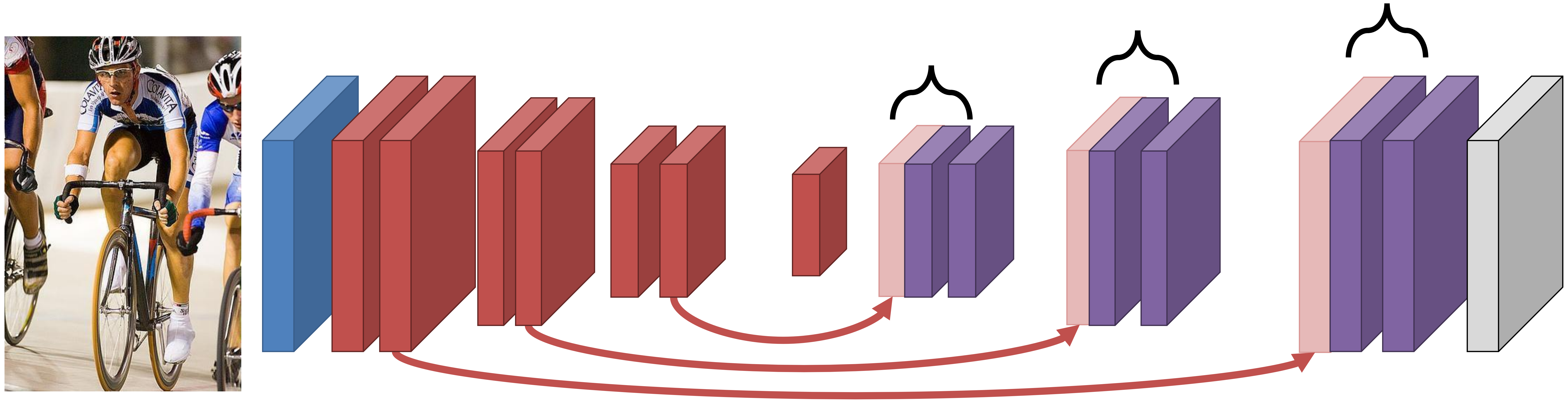
Result from Long et al. *Fully Convolutional Networks For Semantic Segmentation*. CVPR 2014

Missing Details

How do you send details forward in the network?

You copy the activations forward.

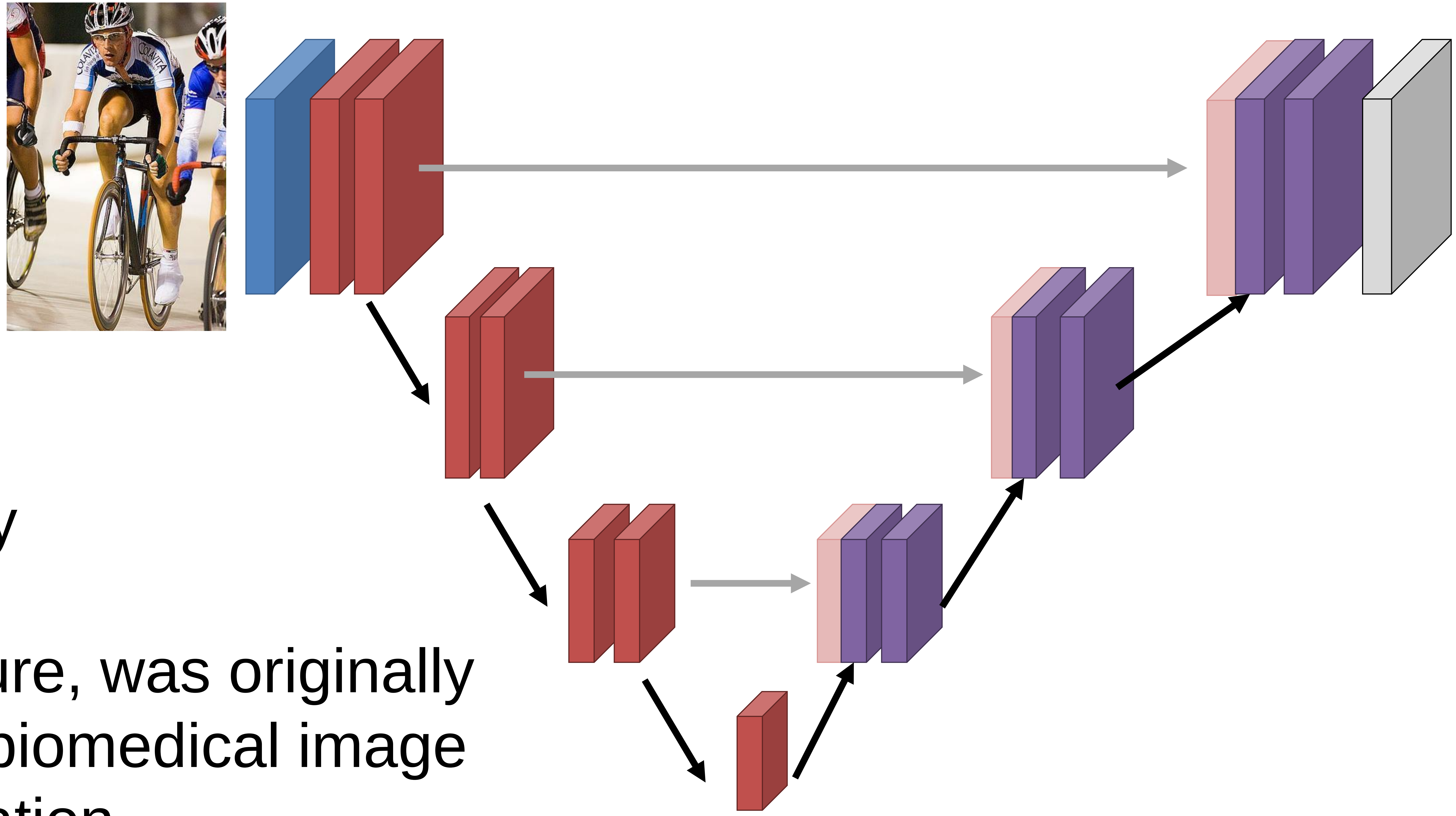
Subsequent layers at the same resolution figure out how to fuse things.



Copy

Result from Long et al. *Fully Convolutional Networks For Semantic Segmentation*. CVPR 2014

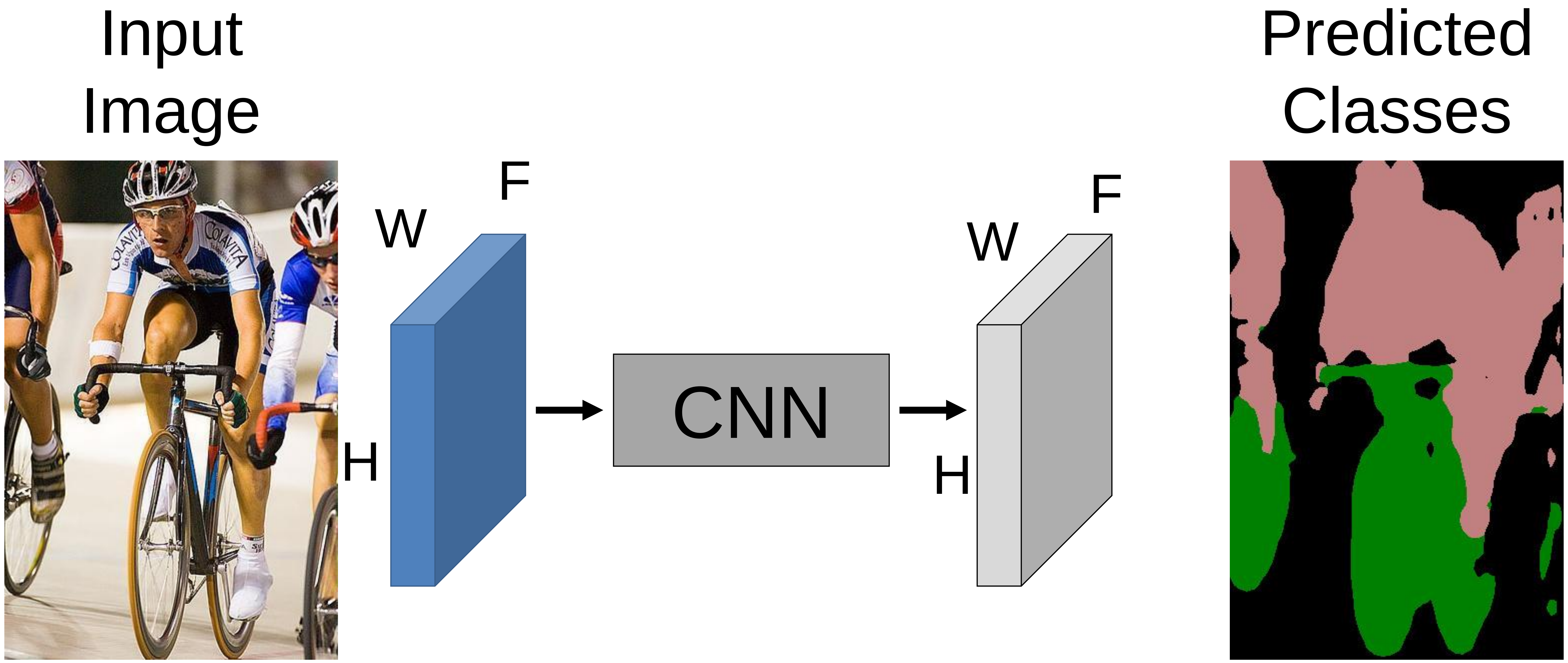
U-Net



Extremely popular architecture, was originally used for biomedical image segmentation.

Ronneberger et al. “U-Net: Convolutional Networks for Biomedical Image Segmentation. MICCAI 2015

Evaluating Pixel Labels



How do we convert final HxWxF into labels?

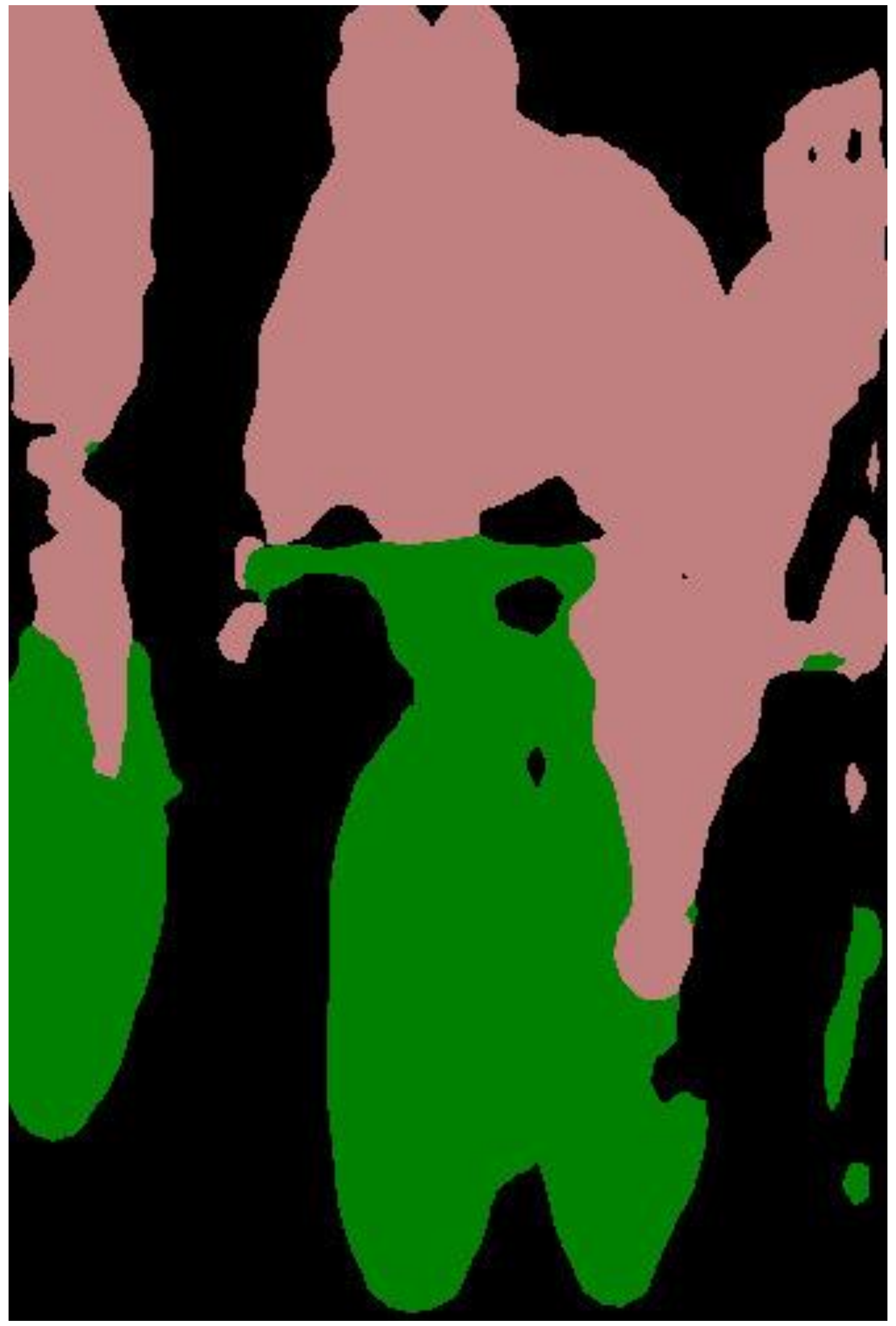
argmax over labels

Evaluating Semantic Segmentation

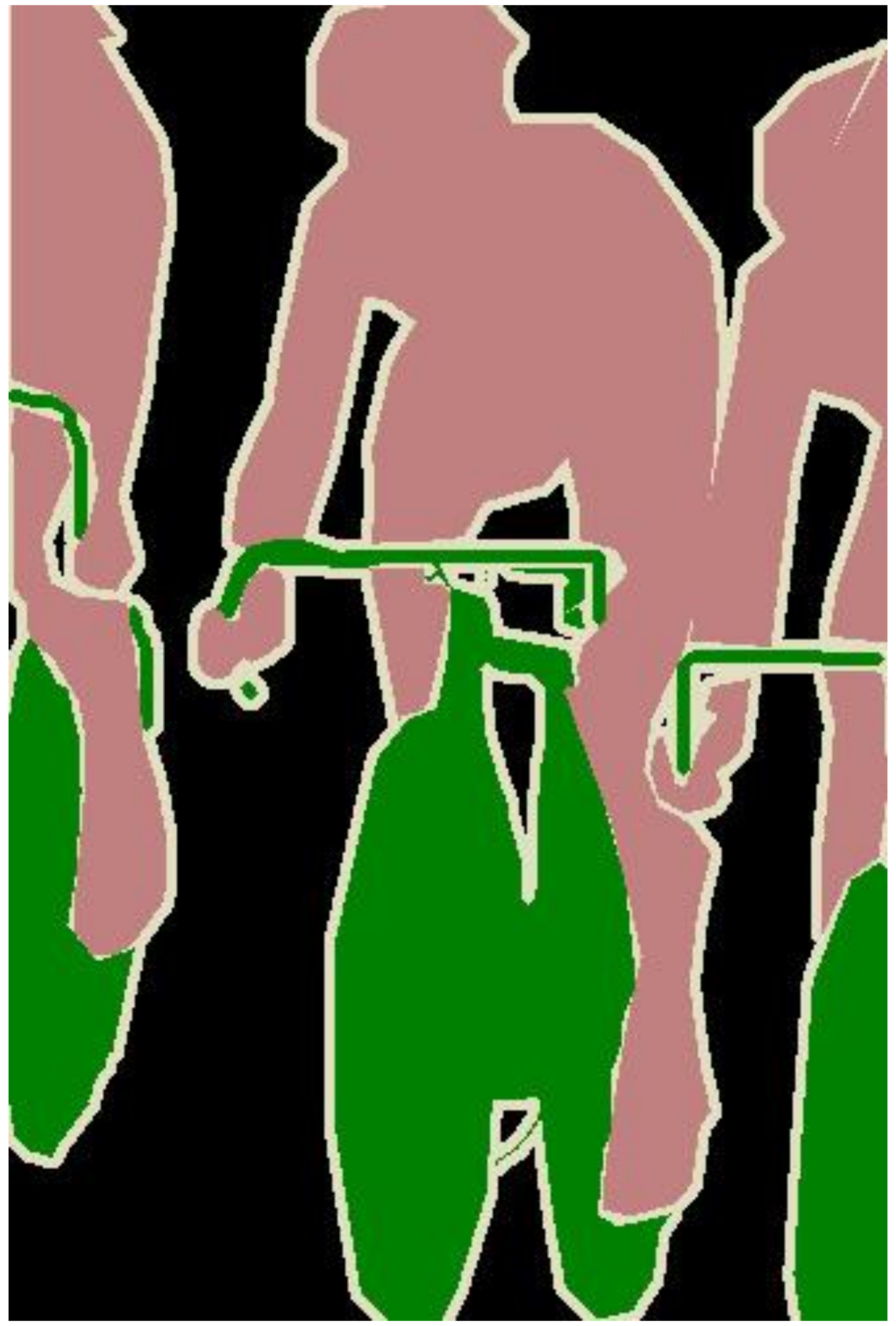
Given predictions, how well did we do?

Input

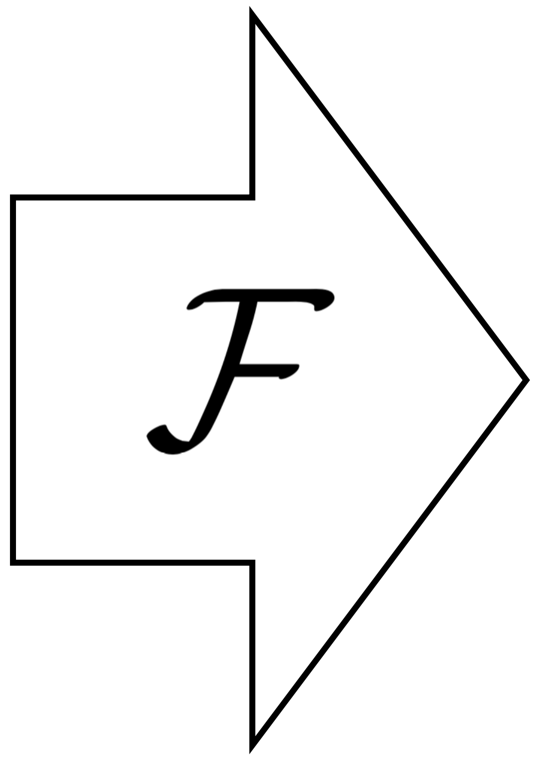
Prediction ($\hat{y}$)



Ground-Truth (y)



What about continuous labels?

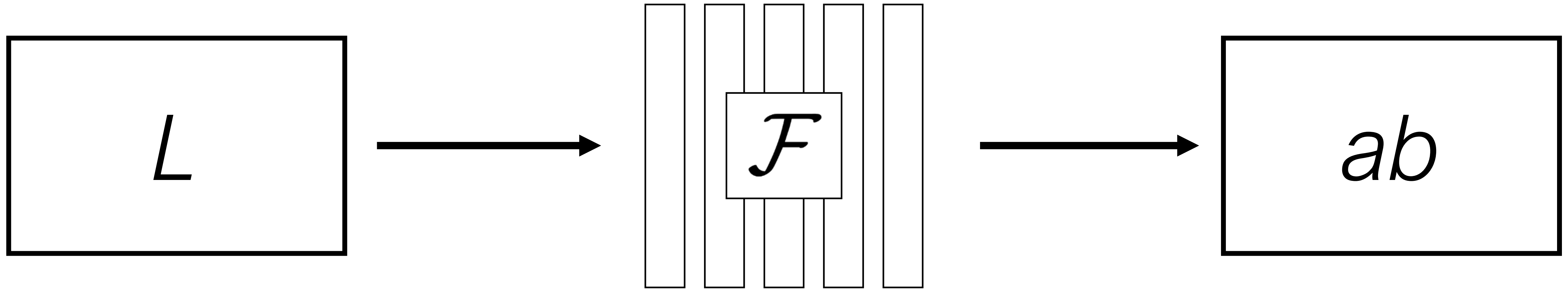


Grayscale image: *L* channel

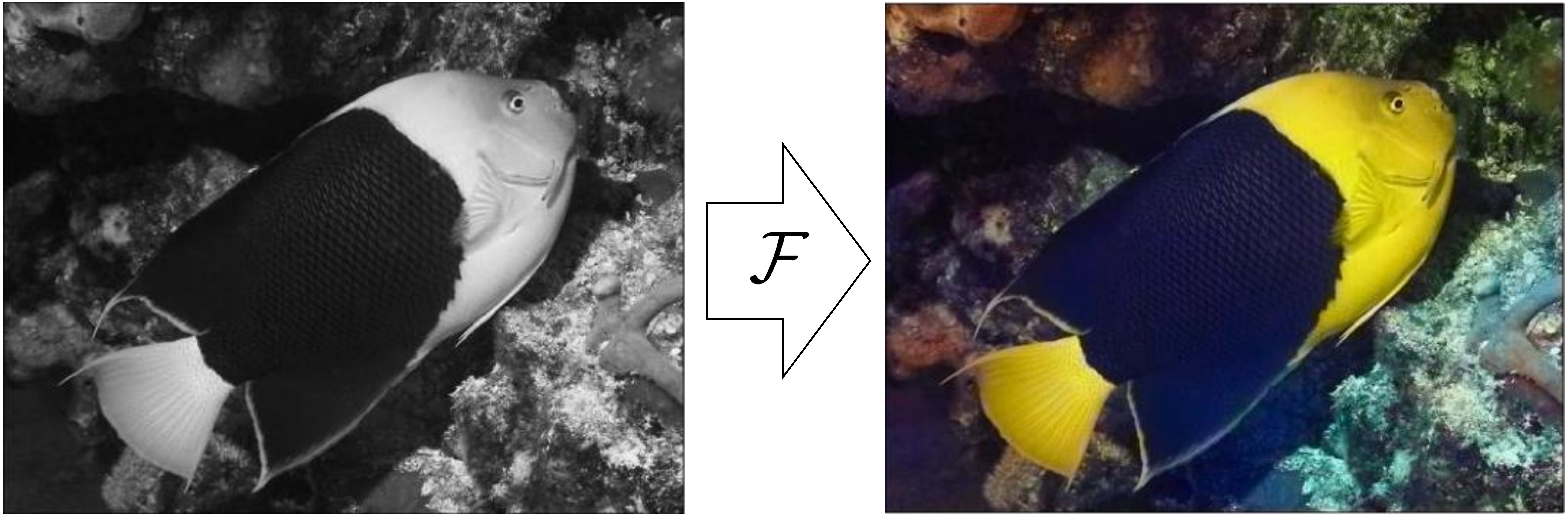
$$\mathbf{X} \in \mathbb{R}^{H \times W \times 1}$$

Color information: *ab* channels

$$\widehat{\mathbf{Y}} \in \mathbb{R}^{H \times W \times 2}$$



Zhang, Isola, Efros. *Colorful Image Colorization.* In *ECCV*, 2016.

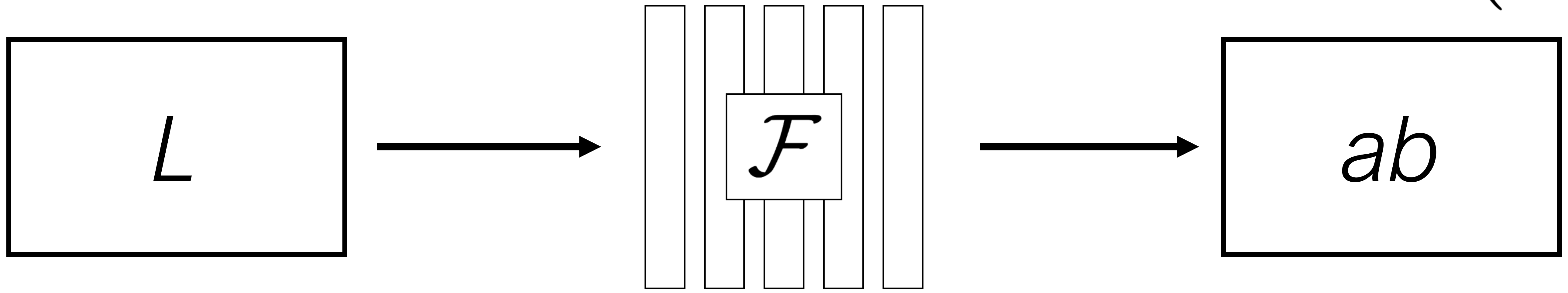


Grayscale image: *L* channel

$$\mathbf{X} \in \mathbb{R}^{H \times W \times 1}$$

Concatenate (*L*,*ab*) channels

$$(\mathbf{X}, \widehat{\mathbf{Y}})$$



Zhang, Isola, Efros. ***Colorful Image Colorization.*** In *ECCV*, 2016.

Regressing to pixel values doesn't work ☹

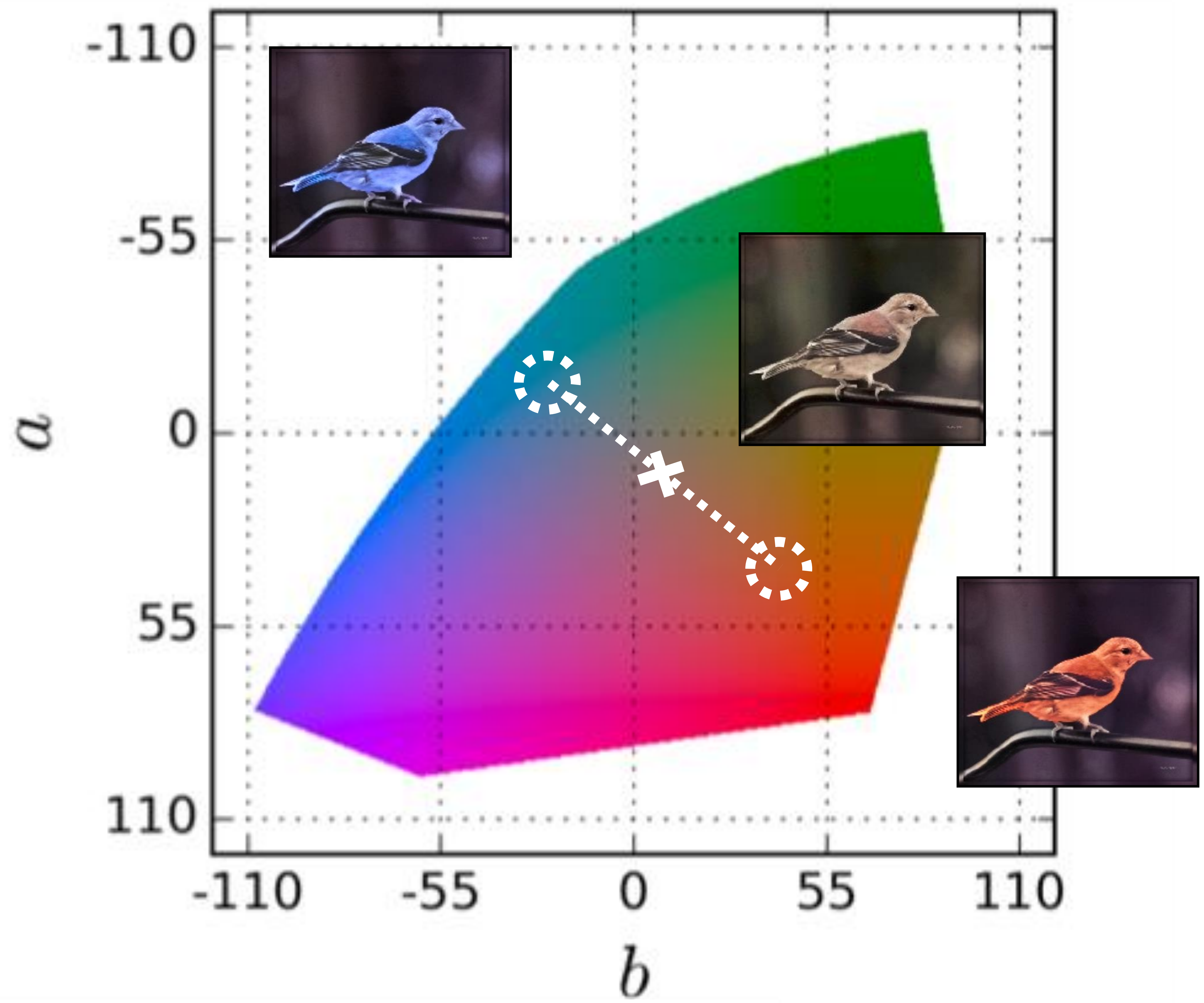
Input

Output

Ground truth

$$L_2(\widehat{\mathbf{Y}}, \mathbf{Y}) = \frac{1}{2}\sum_{h,w} \|\mathbf{Y}_{h,w} - \widehat{\mathbf{Y}}_{h,w}\|_2^2$$

Slide by Richard Zhang



$$\mathrm{L}_2(\widehat{\mathbf{Y}}, \mathbf{Y}) = \frac{1}{2} \sum_{h,w} \| \mathbf{Y}_{h,w} - \widehat{\mathbf{Y}}_{h,w} \|_2^2$$

Slide by Richard Zhang

Better Loss Function

$$\theta^* = \arg\min_\theta \ell(\mathcal{F}_\theta(\mathbf{X}), \mathbf{Y})$$

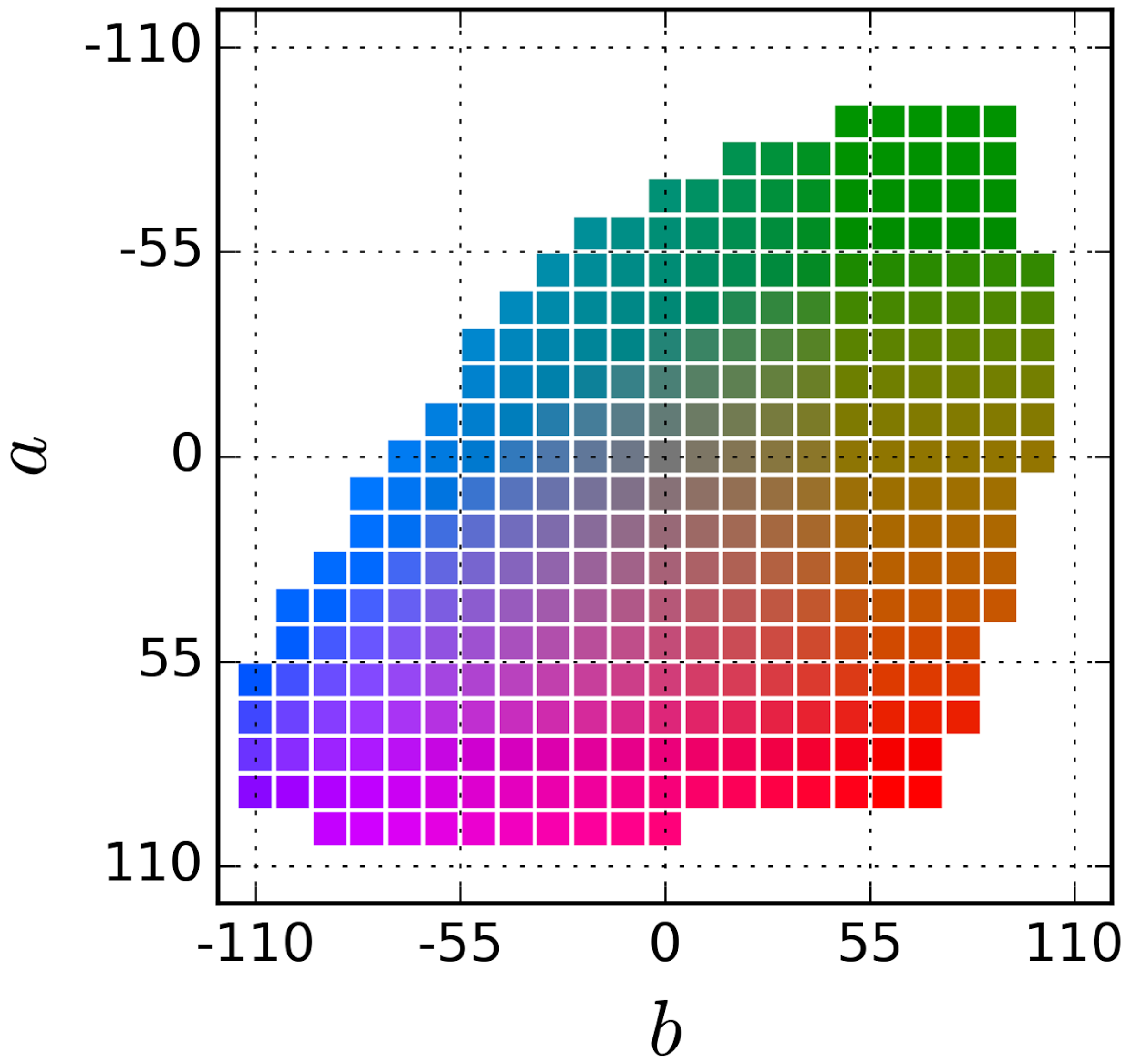
- Regression with L2 loss inadequate

$$L_2(\widehat{\mathbf{Y}}, \mathbf{Y}) = \frac{1}{2}\sum_{h,w} \|\mathbf{Y}_{h,w} - \widehat{\mathbf{Y}}_{h,w}\|_2^2$$

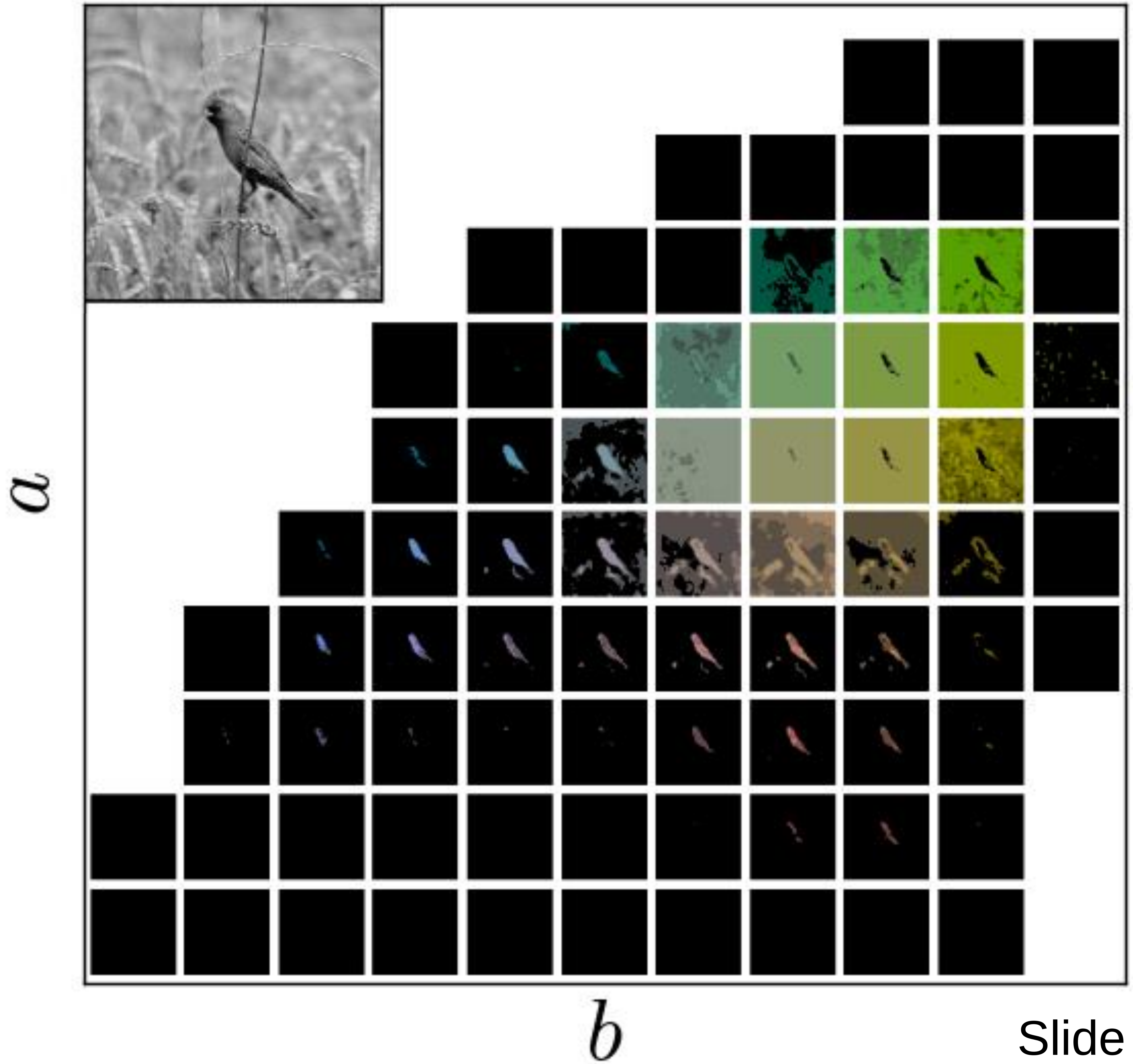
- Use per-pixel multinomial classification

$$L(\widehat{\mathbf{Z}}, \mathbf{Z}) = -\frac{1}{HW}\sum_{h,w}\sum_q \mathbf{Z}_{h,w,q} \log(\widehat{\mathbf{Z}}_{h,w,q})$$

Colors in *ab* space
(discrete)



Slide by Richard Zhang



Slide by Richard Zhang

Designing pixel loss functions

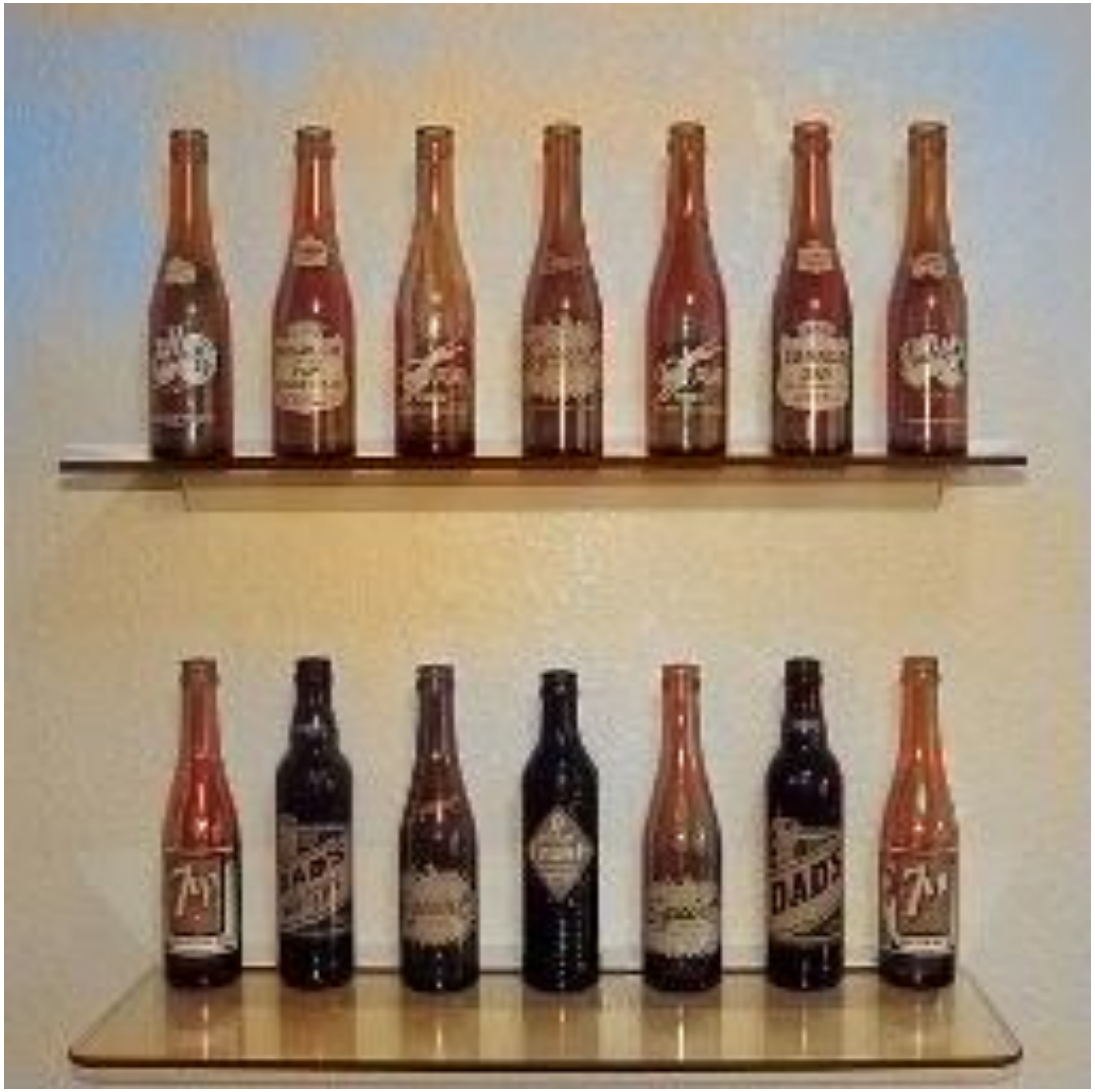
Input

Zhang et al. 2016

Ground truth

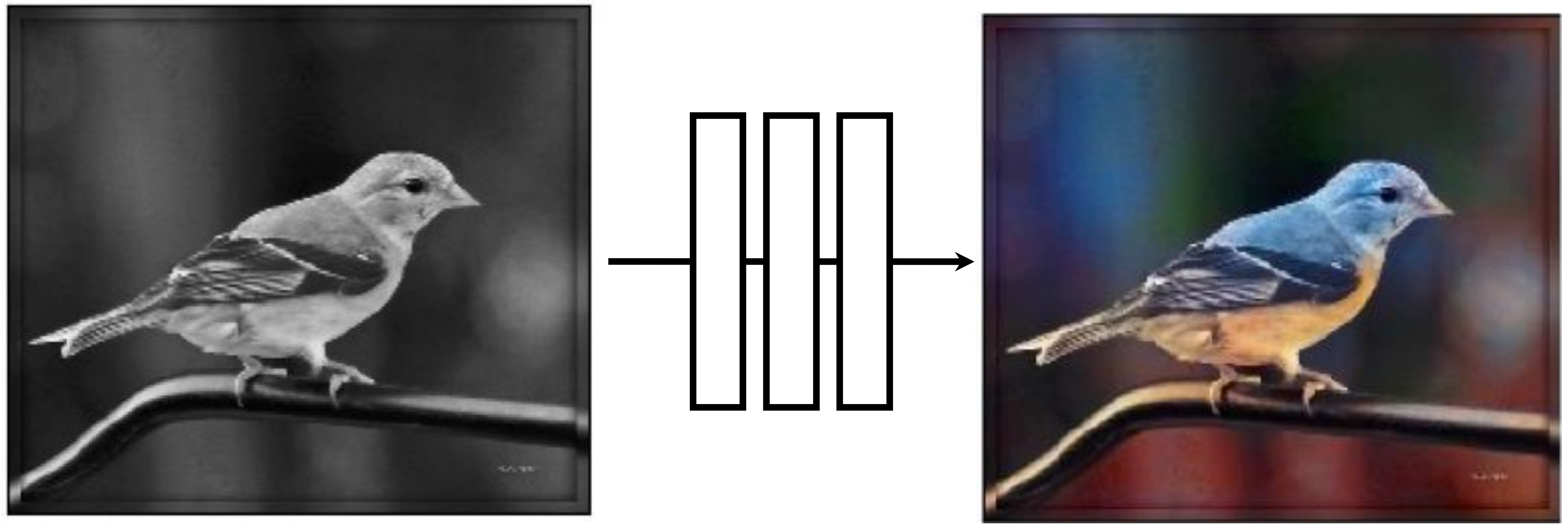
Color distribution cross-entropy loss with colorfulness enhancing term.

[Zhang, Isola, Efros, ECCV 2016]


DADS

Designing pixel loss functions

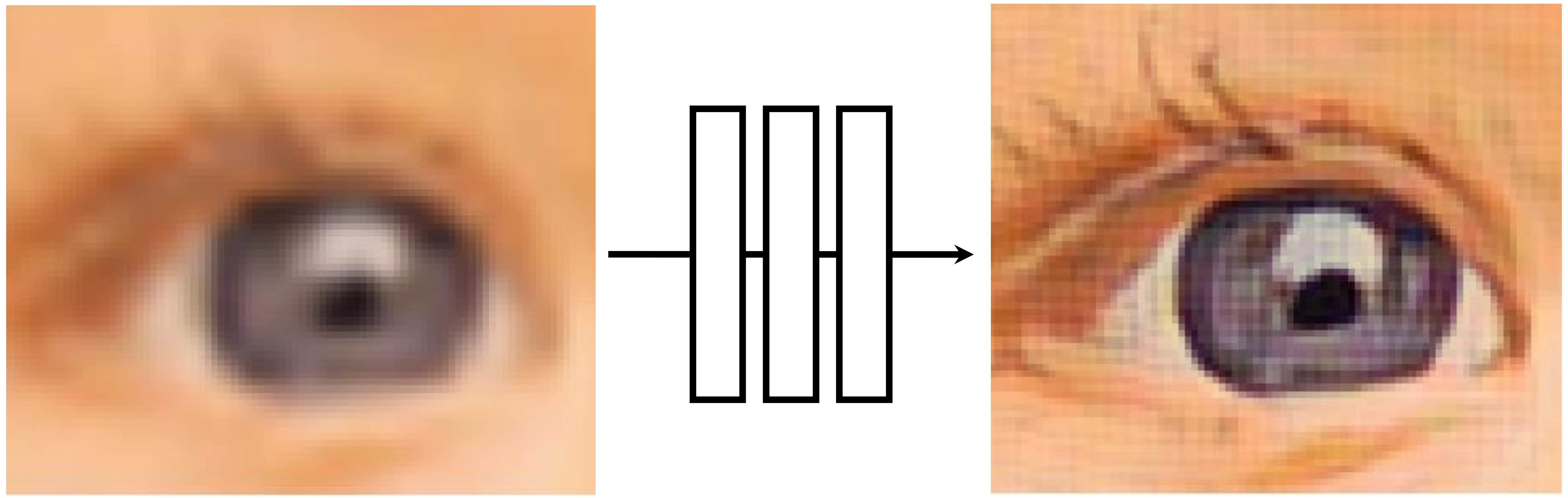
Image colorization



[Zhang et al. 2016]

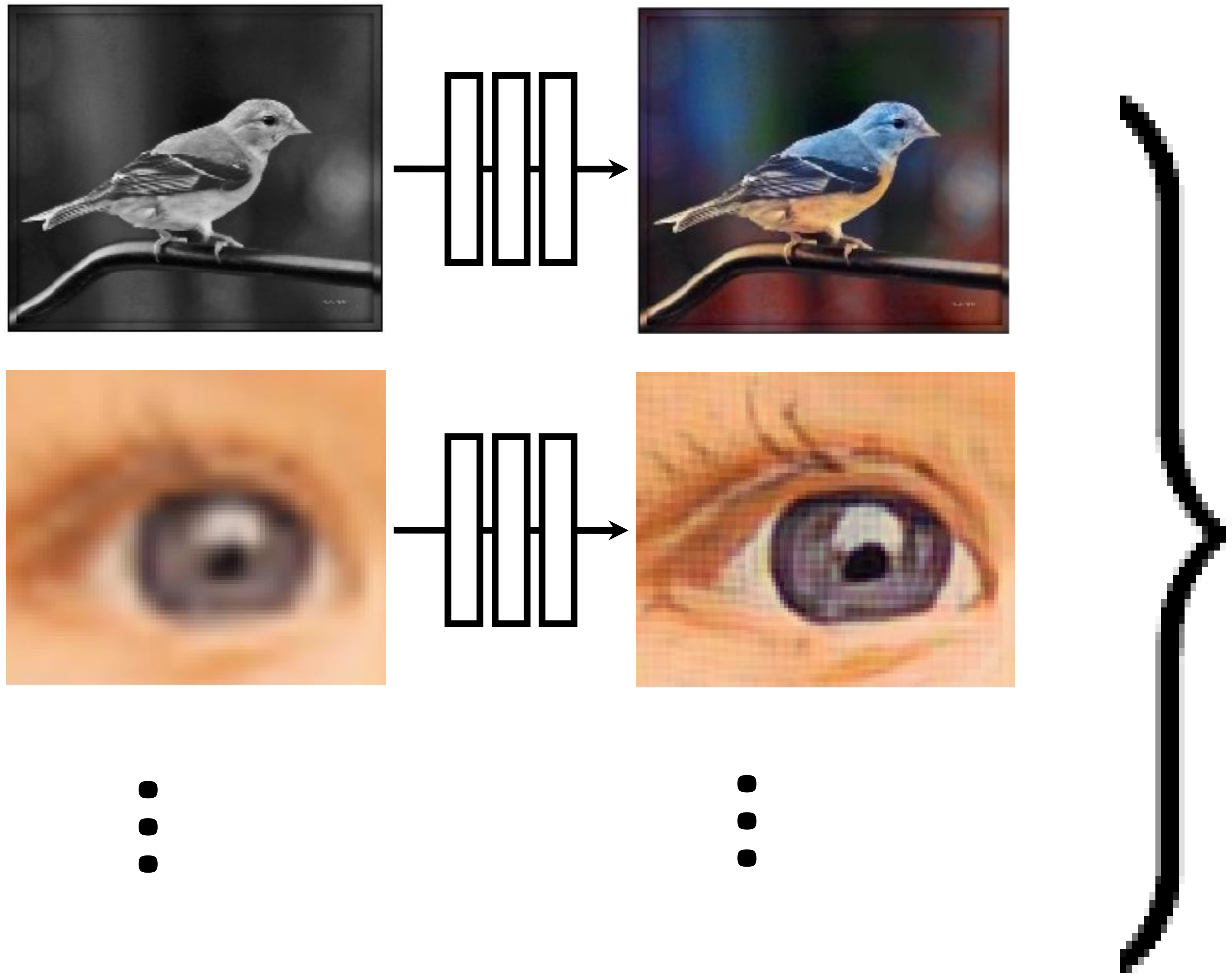
Cross entropy loss, with colorfulness term

Super-resolution



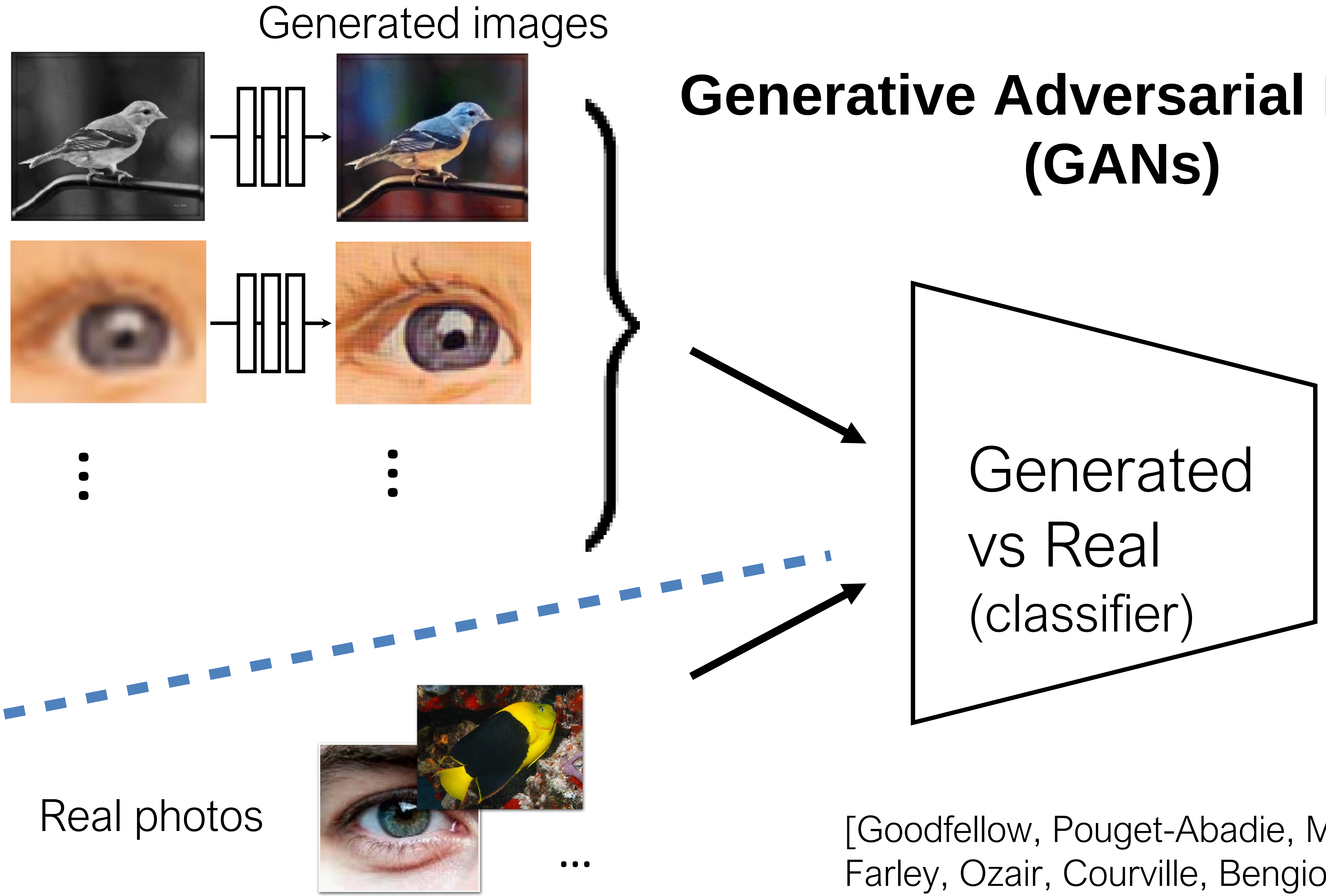
[Johnson et al. 2016]

"semantic feature loss" (VGG feature covariance matching objective)

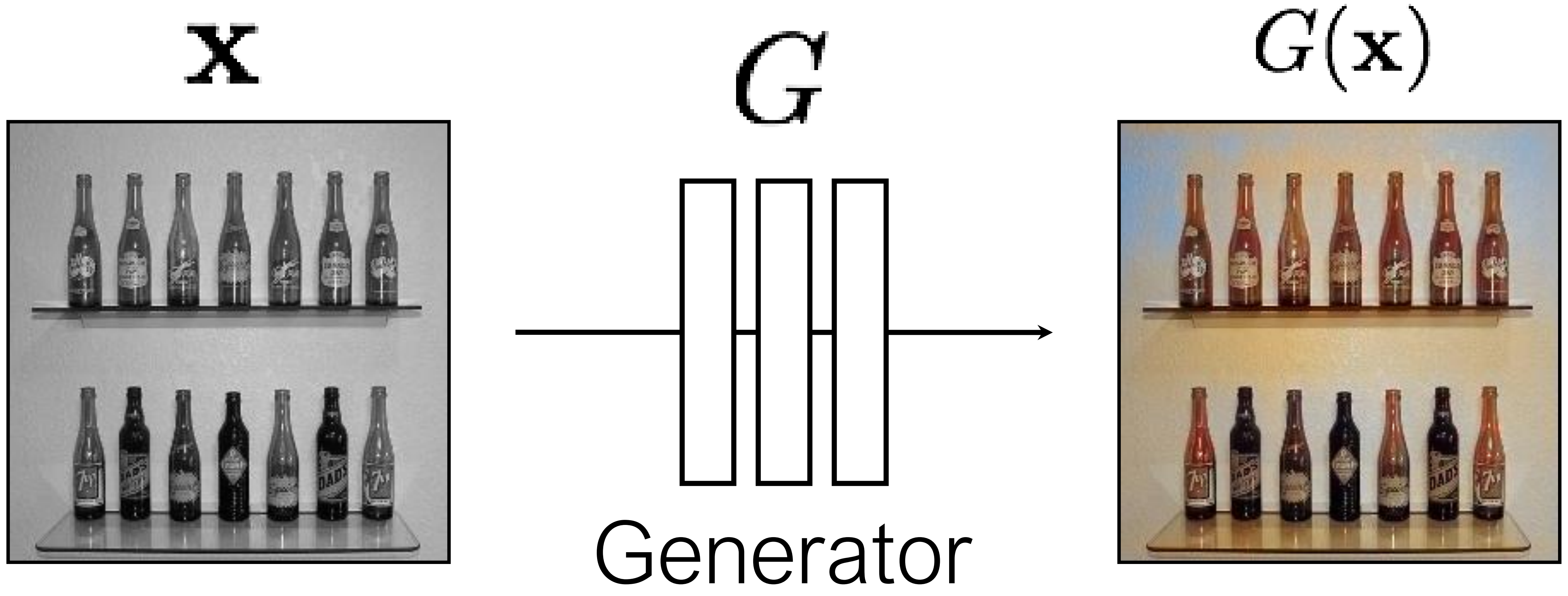


Universal loss?

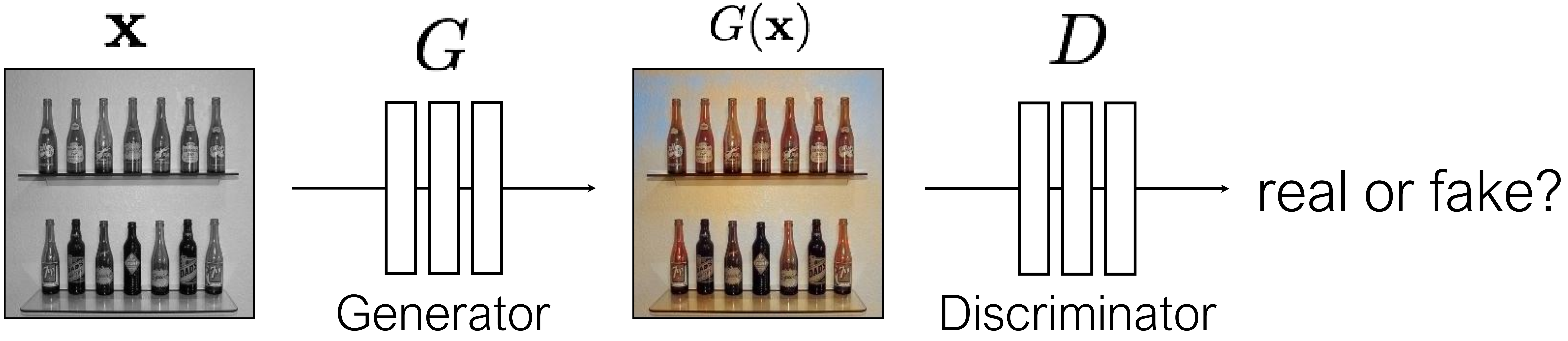
Generative Adversarial Network (GANs)



[Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville, Bengio 2014]



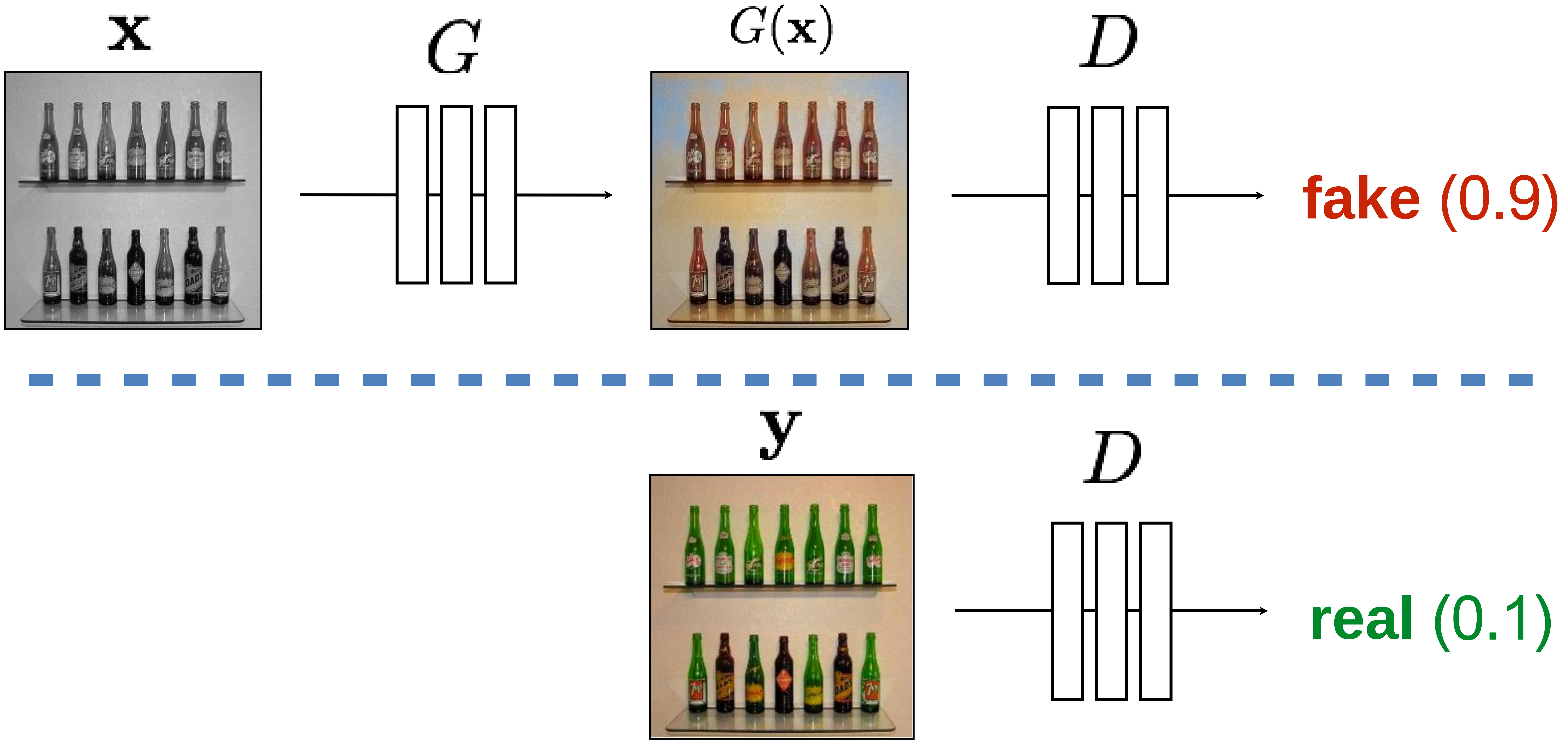
[Goodfellow et al., 2014]



G tries to synthesize fake images that fool **D**

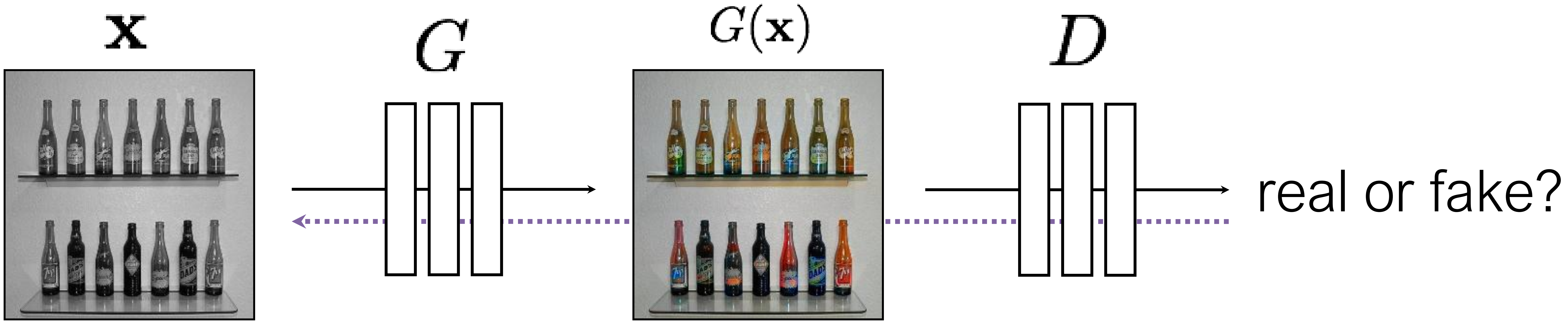
D tries to identify the fakes

[Goodfellow et al., 2014]



$$\arg\max_{D}\ \mathbb{E}_{\mathbf{x},\mathbf{y}}[\ \boxed{\log D(G(\mathbf{x}))}\ +\ \boxed{\log(1-D(\mathbf{y}))}\ ]$$

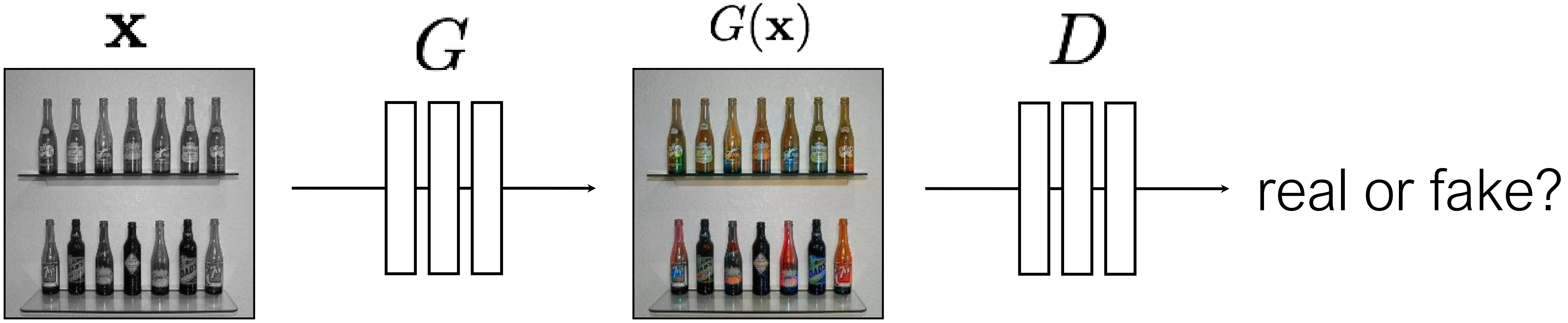
[Goodfellow et al., 2014]



G tries to synthesize fake images that ***fool*** **D**:

$$\arg \boxed{\min_{G}} \; \mathbb{E}_{\mathbf{x},\mathbf{y}}[\; \log D(G(\mathbf{x})) \;+\; \log(1 - D(\mathbf{y})) \;]$$

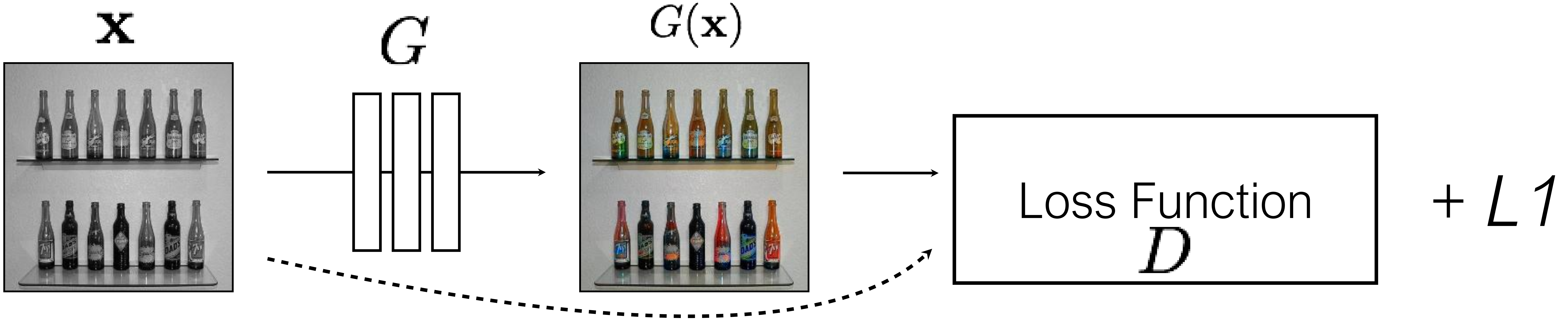
[Goodfellow et al., 2014]



G tries to synthesize fake images that *fool* the ***best*** **D**:

$$\arg \boxed{\min_{G}} \boxed{\max_{D}} \; \mathbb{E}_{\mathbf{x},\mathbf{y}}[\; \log D(G(\mathbf{x})) \;+\; \log(1 - D(\mathbf{y})) \;]$$

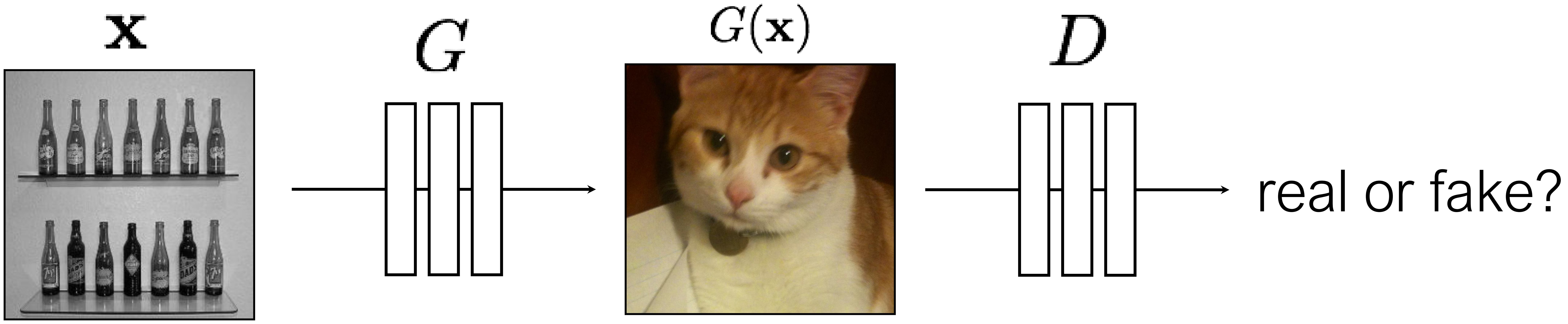
[Goodfellow et al., 2014]



G’s perspective: **D** is a loss function.

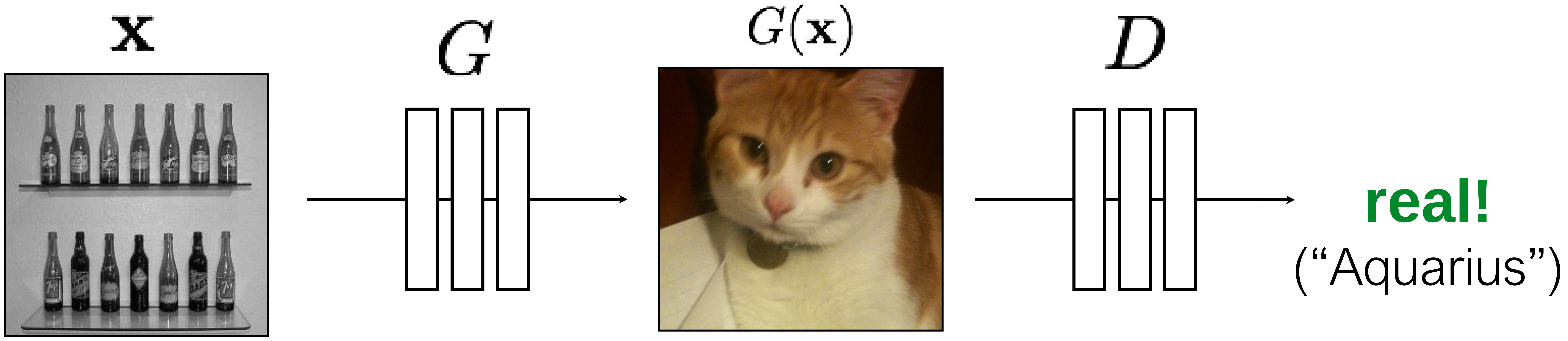
Rather than being hand-designed, it is *learned*.

[Goodfellow et al., 2014]
[Isola et al., 2017]



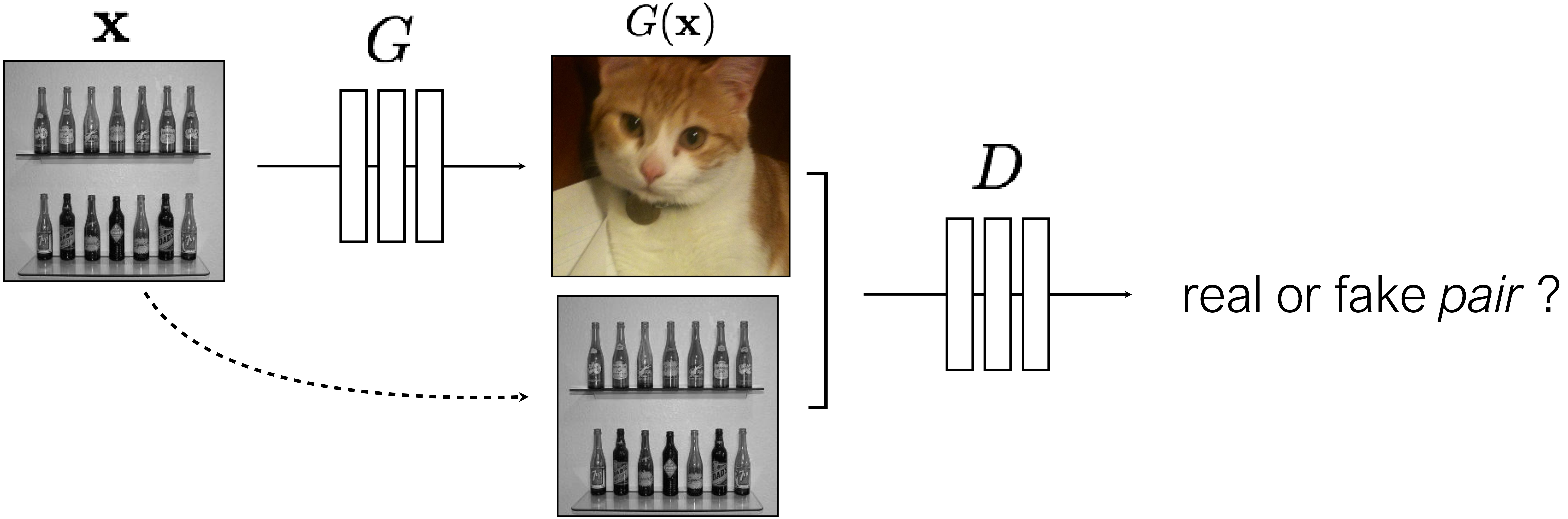
$$\arg\min_{G}\max_{D}\ \mathbb{E}_{\mathbf{x},\mathbf{y}}[\ \log D(G(\mathbf{x})) \quad + \quad \log(1 - D(\mathbf{y}))\ \ ]$$

[Goodfellow et al., 2014]



$$\arg\min_{G}\max_{D}\ \mathbb{E}_{\mathbf{x},\mathbf{y}}[\ \log D(G(\mathbf{x})) \ + \ \log(1 - D(\mathbf{y}))\ ]$$

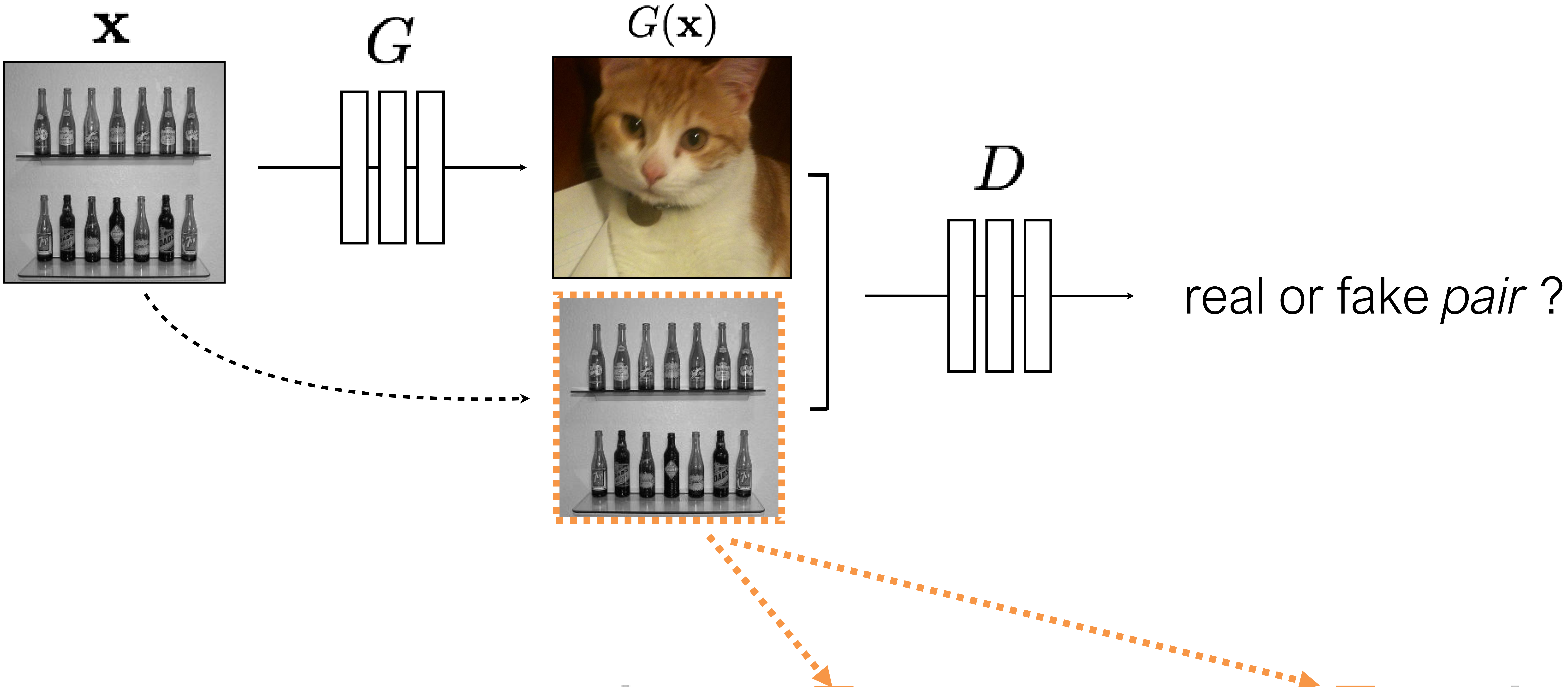
[Goodfellow et al., 2014]



$$\arg\min_G \max_D \; \mathbb{E}_{\mathbf{x},\mathbf{y}}[\; \log D(G(\mathbf{x})) \quad + \quad \log(1 - D(\mathbf{y})) \;]$$

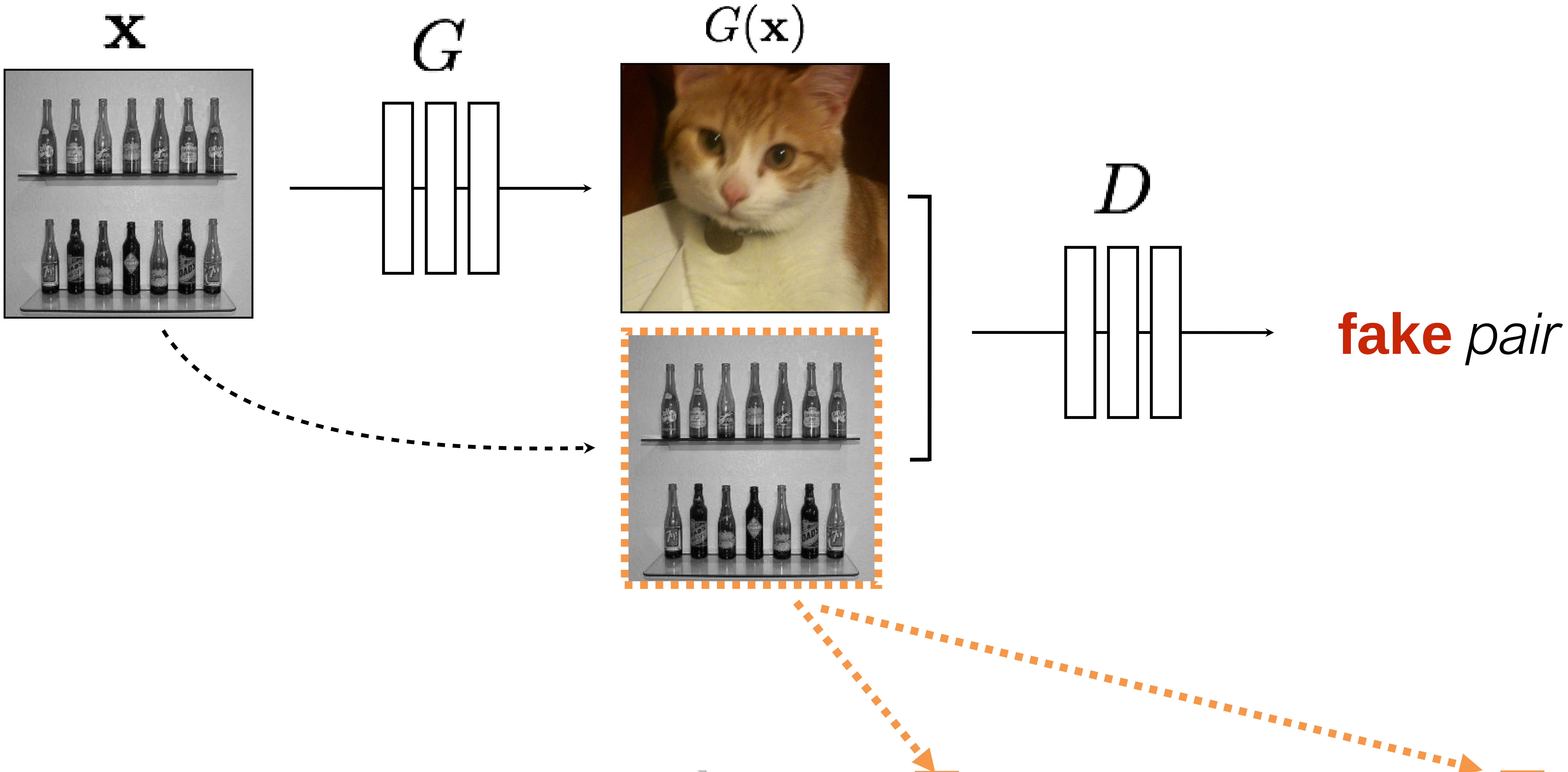
[Goodfellow et al., 2014]

[Isola et al., 2017]



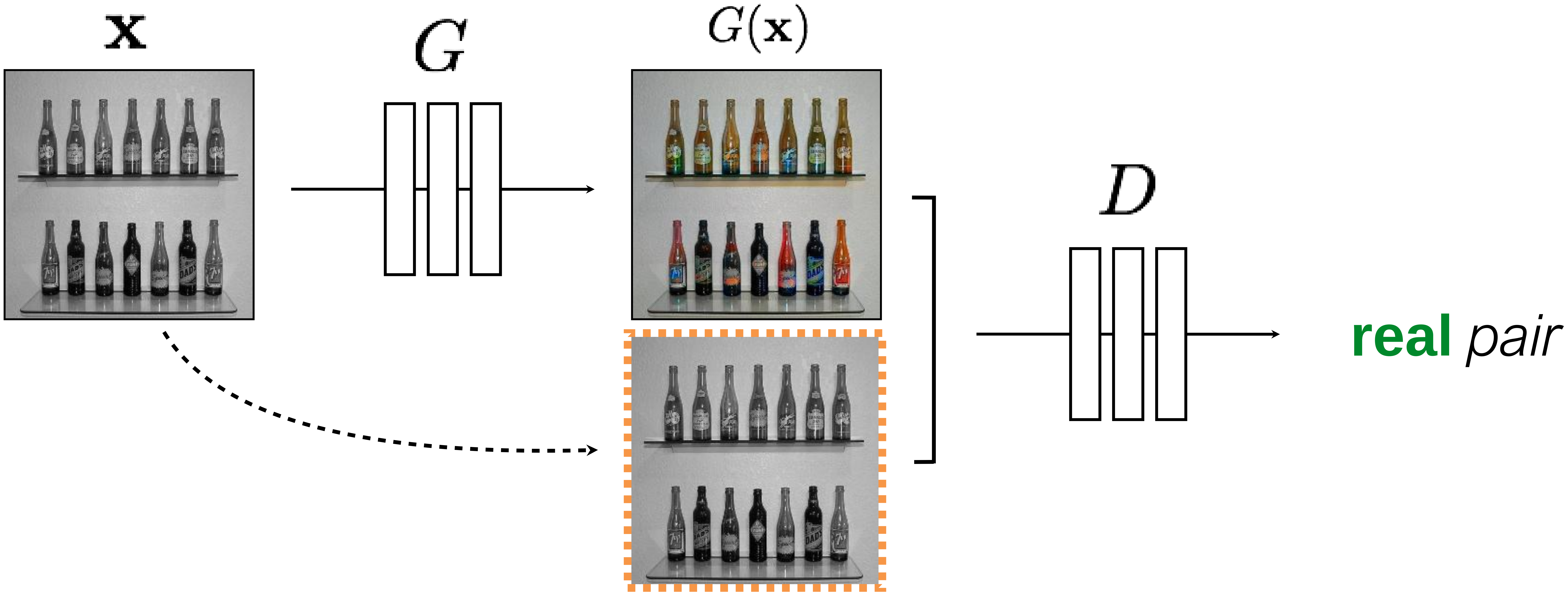
$$\arg\min_{G}\max_{D}\ \mathbb{E}_{\mathbf{x},\mathbf{y}}\left[\ \log D(\boxed{\mathbf{x}}, G(\mathbf{x})) + \log(1 - D(\boxed{\mathbf{x}}, \mathbf{y}))\ \right]$$

[Goodfellow et al., 2014]
[Isola et al., 2017]



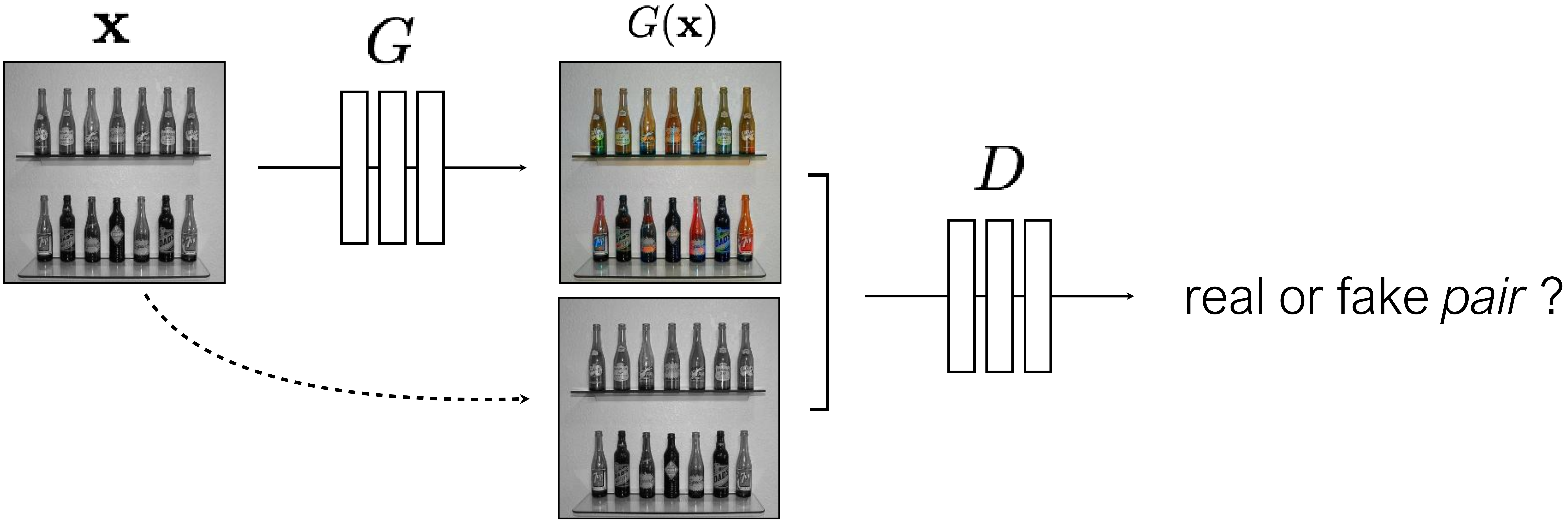
$$\arg\min_G \max_D \; \mathbb{E}_{\mathbf{x},\mathbf{y}}\left[\; \log D(\boxed{\mathbf{x}}, G(\mathbf{x})) + \log(1 - D(\boxed{\mathbf{x}}, \mathbf{y})) \;\right]$$

[Goodfellow et al., 2014]
[Isola et al., 2017]



$$\arg\min_{G}\max_{D}\ \mathbb{E}_{\mathbf{x},\mathbf{y}}[\ \log D(\boxed{\mathbf{x}}, G(\mathbf{x})) + \log(1 - D(\boxed{\mathbf{x}}, \mathbf{y}))\ ]$$

[Goodfellow et al., 2014]
[Isola et al., 2017]



$$\arg\min_{G}\max_{D}\ \mathbb{E}_{\mathbf{x},\mathbf{y}}[\ \log D(\mathbf{x},G(\mathbf{x})) + \log(1 - D(\mathbf{x},\mathbf{y}))\ ]$$

[Goodfellow et al., 2014]
[Isola et al., 2017]

BW → Color






Data from [Russakovsky et al. 2015]

BW → Color






Data from [Russakovsky et al. 2015]

Input

Output

Groundtruth

Data from [maps.google.com]

Input

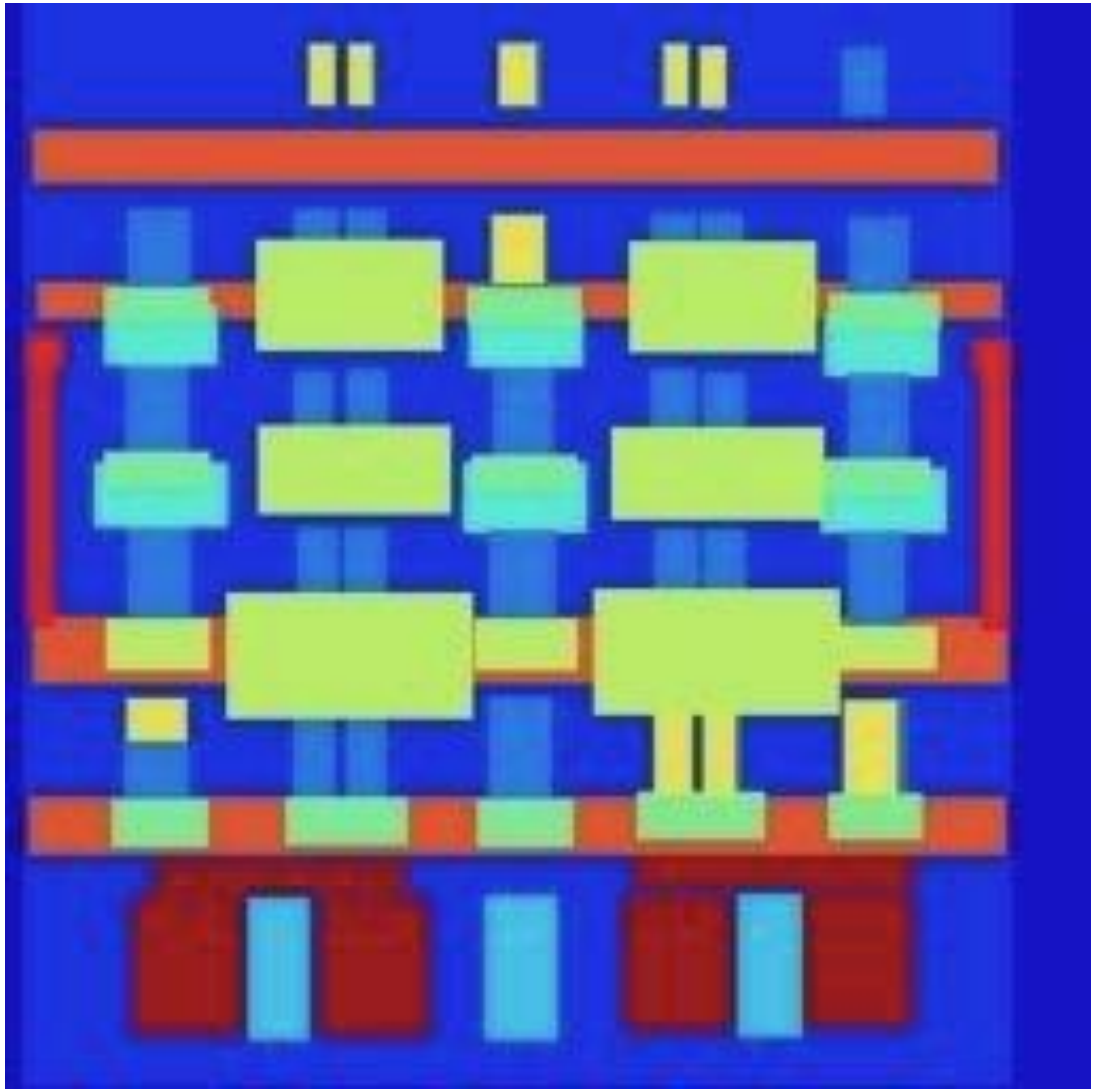
Output

Groundtruth

Data from [maps.google.

Labels → Facades

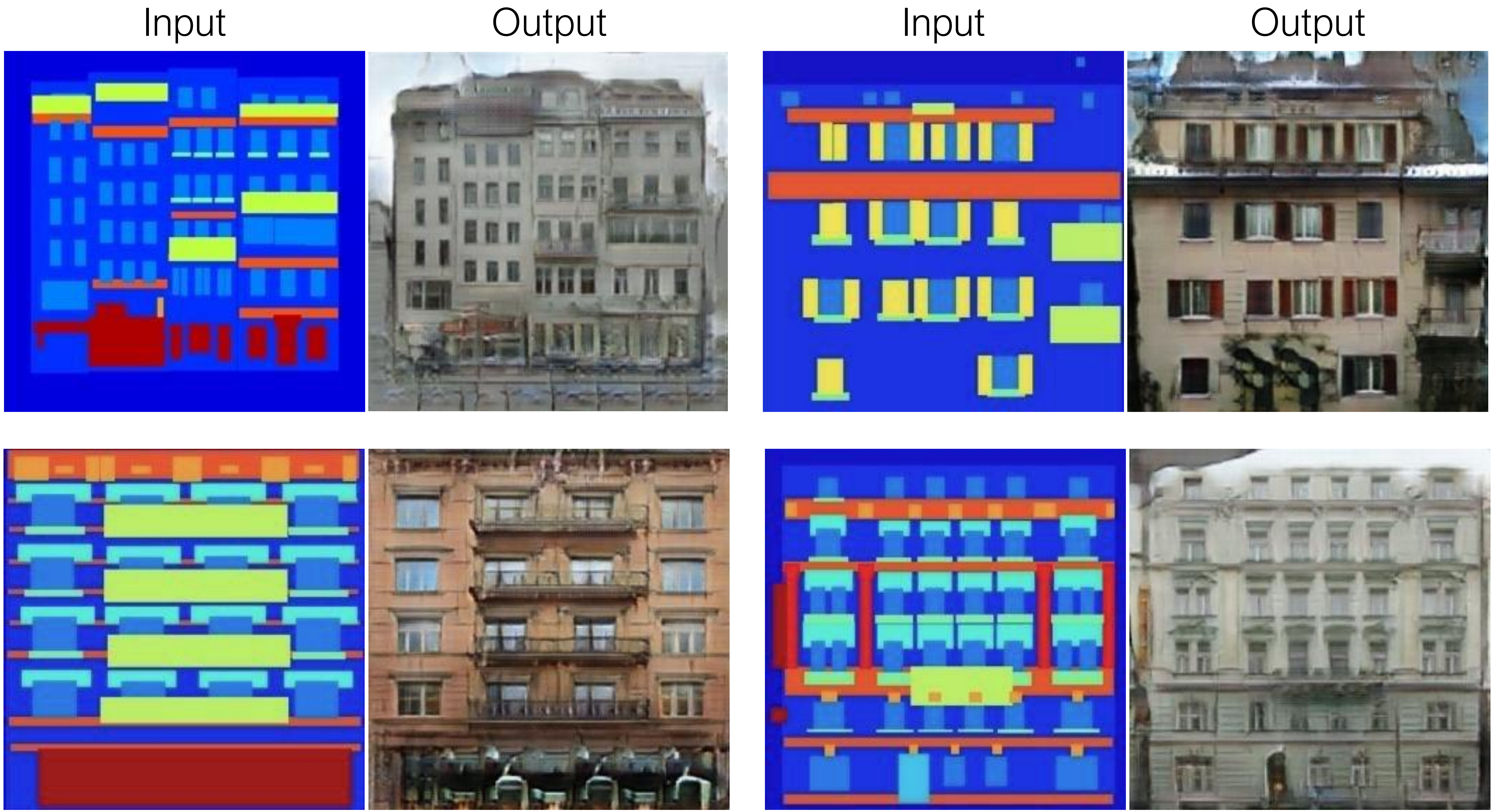
Input



Output

Data from [Tylecek, 2013]

Labels → Facades



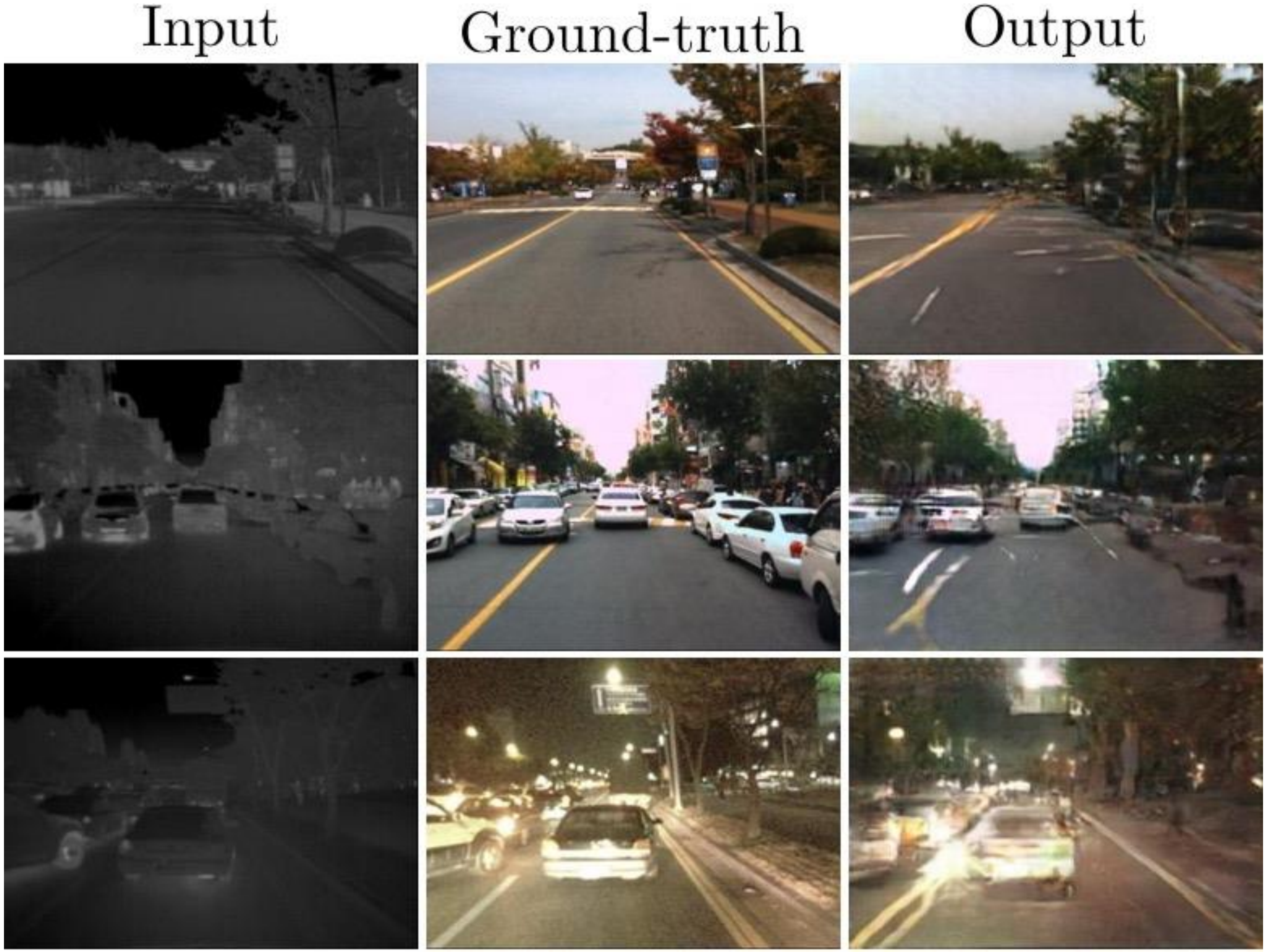
Data from [Tylecek, 2013]

Day → Night

Input Output Input Output Input Output

Data from [Laffont et al., 2014]

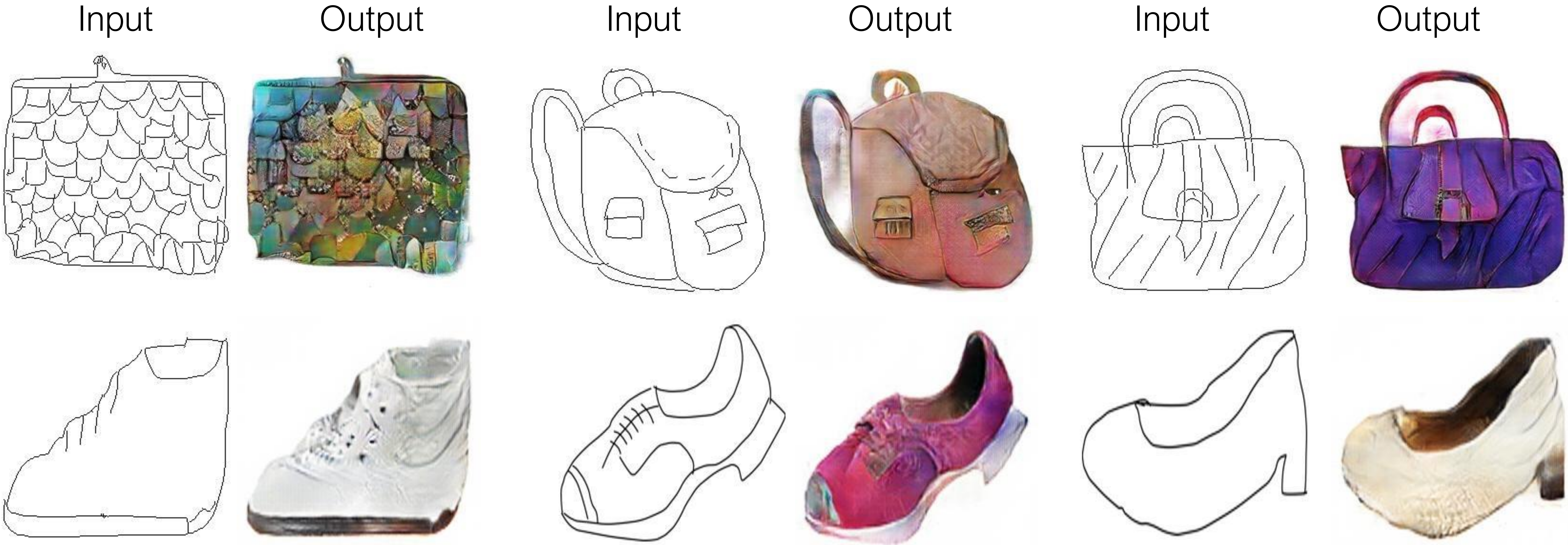
Thermal → RGB



Edges → Images

Edges from [Xie & Tu, 2015]

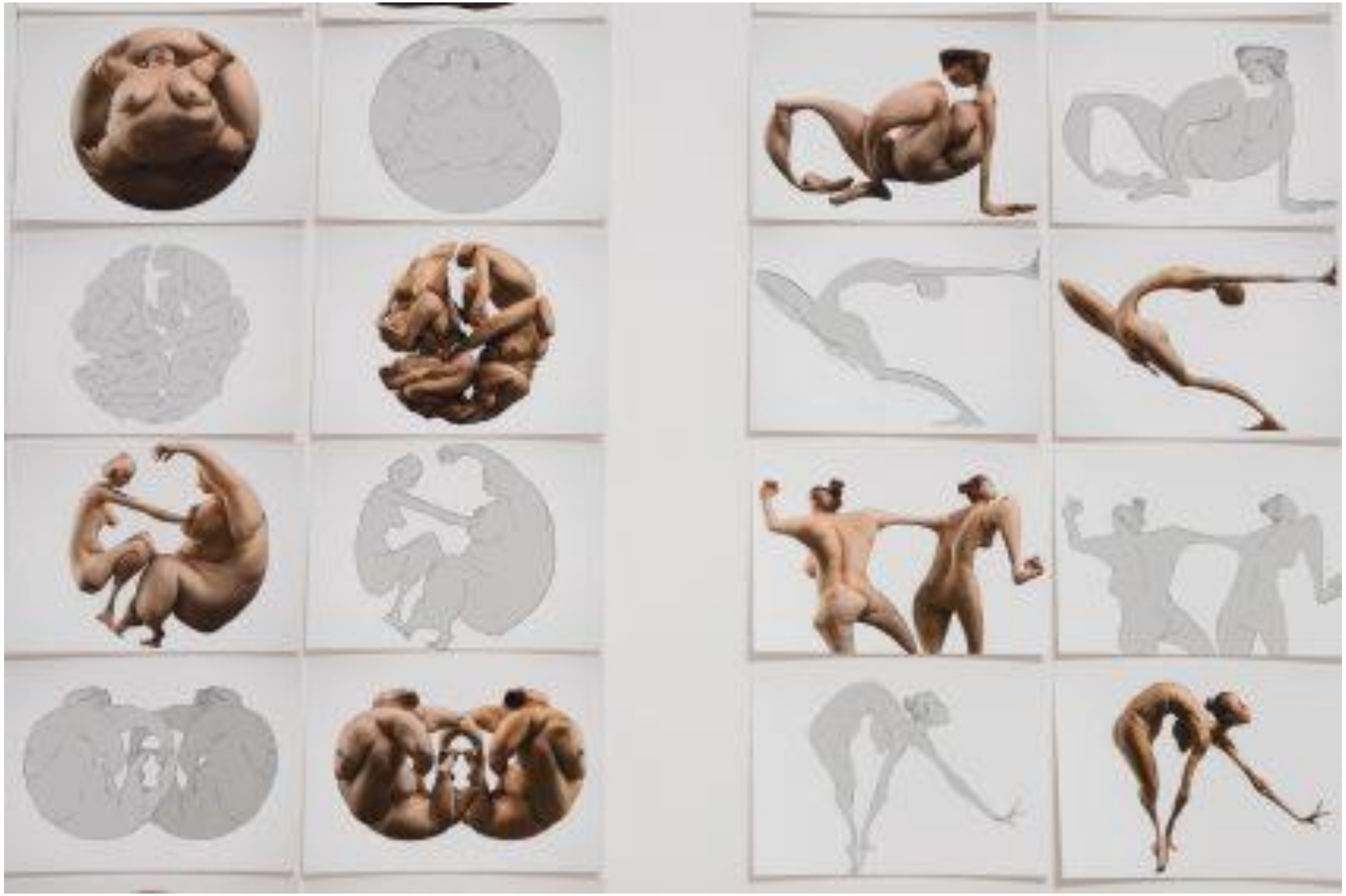
Sketches → Images



Trained on Edges → Images

Data from [Eitz, Hays, Alexa, 2012]

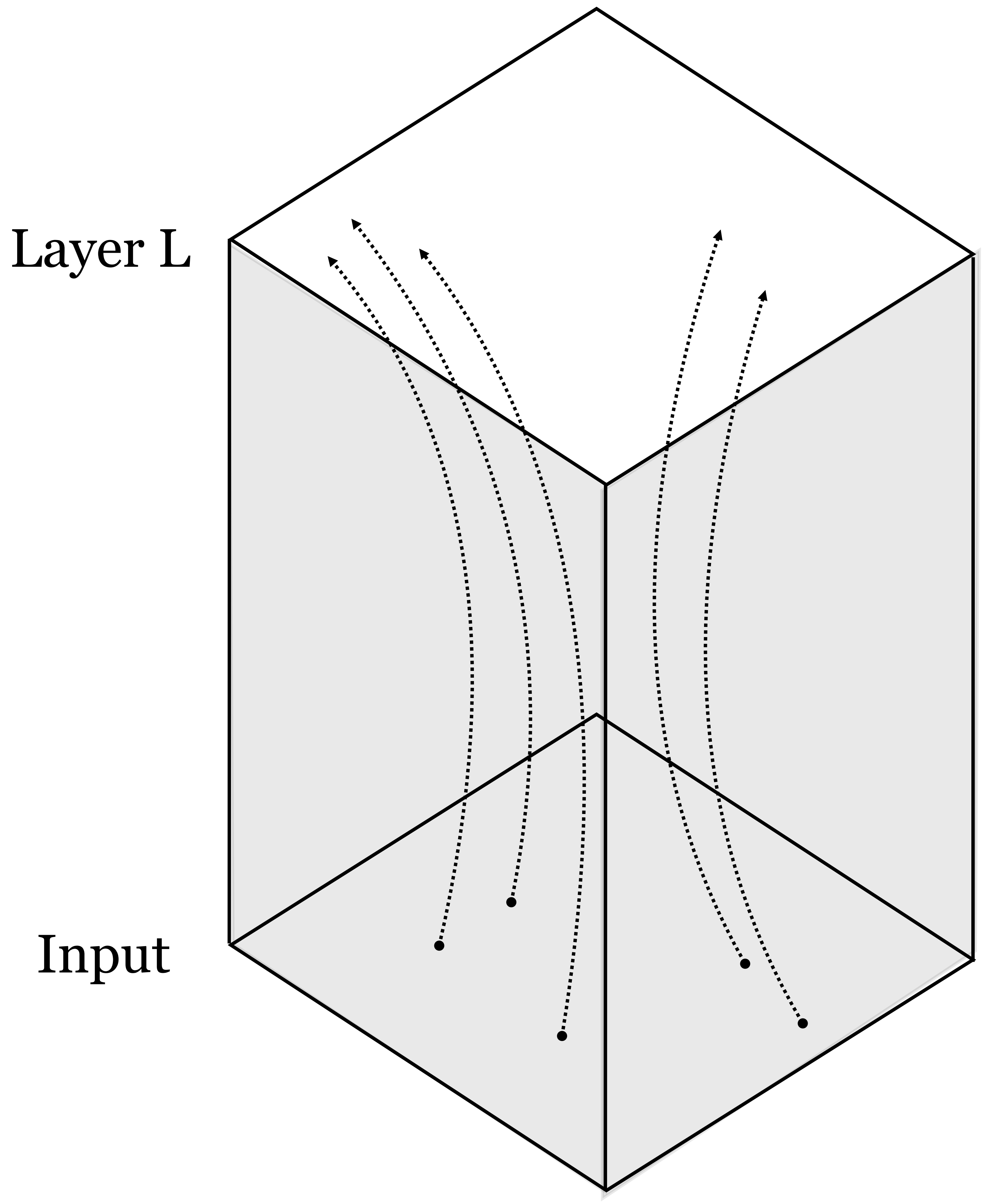
Scott Eaton (http://www.scott-eaton.com/)



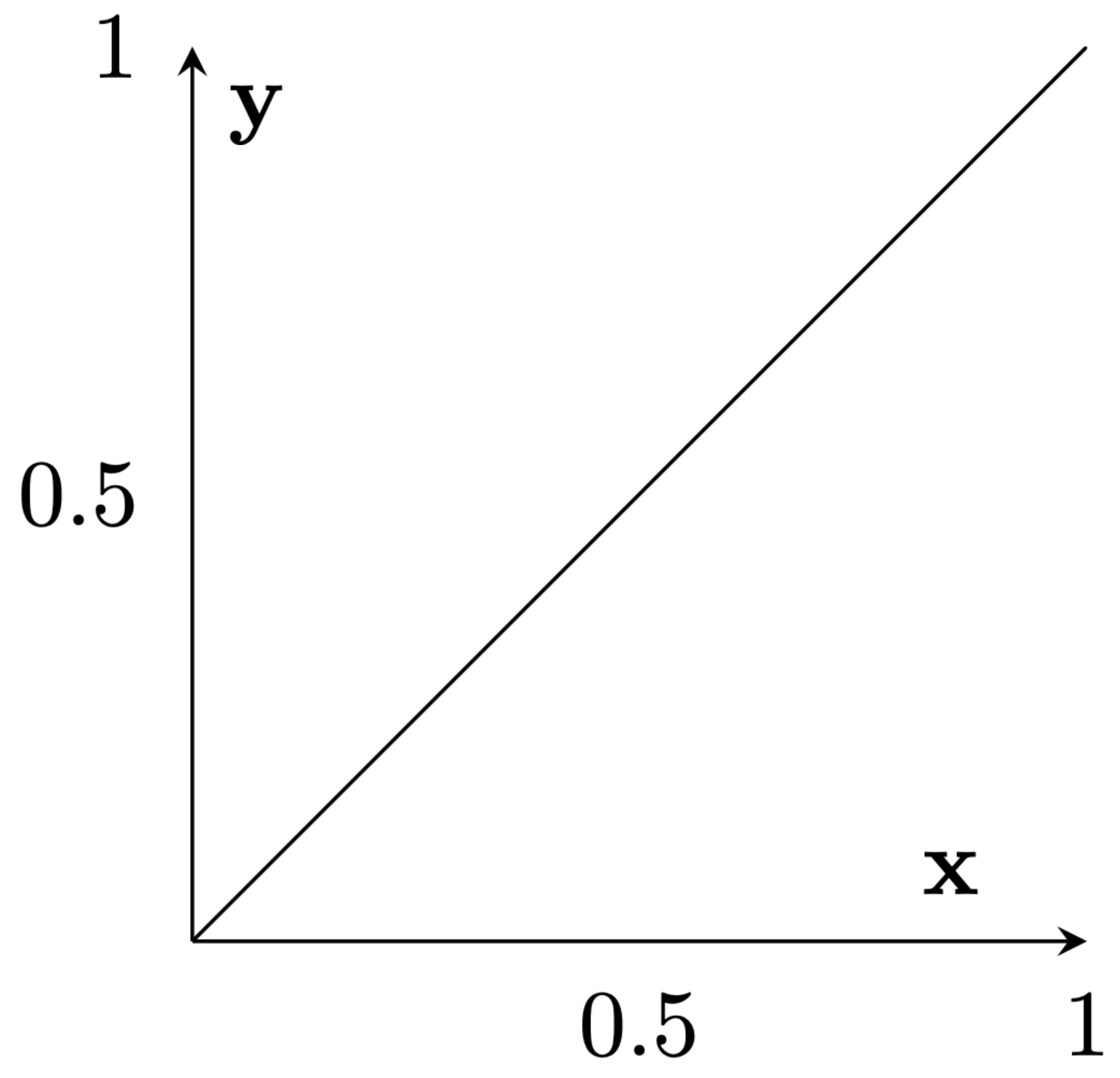
Capturing Statistics: Fast Activations and Slow Parameters

Deep nets are data transformers

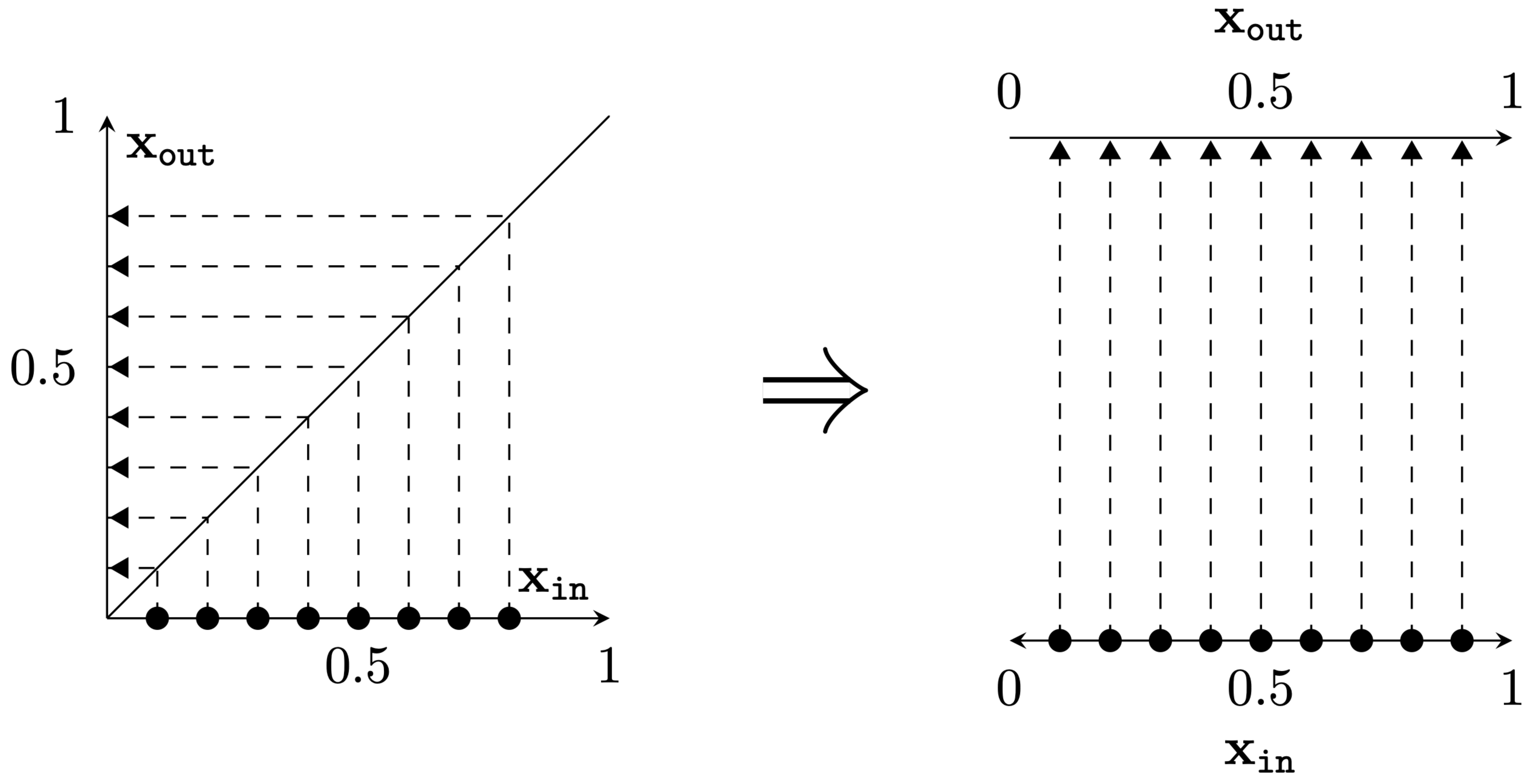
- Deep nets transform datapoints, layer by layer
- Each layer is a different representation of the data
- We call these representations embeddings



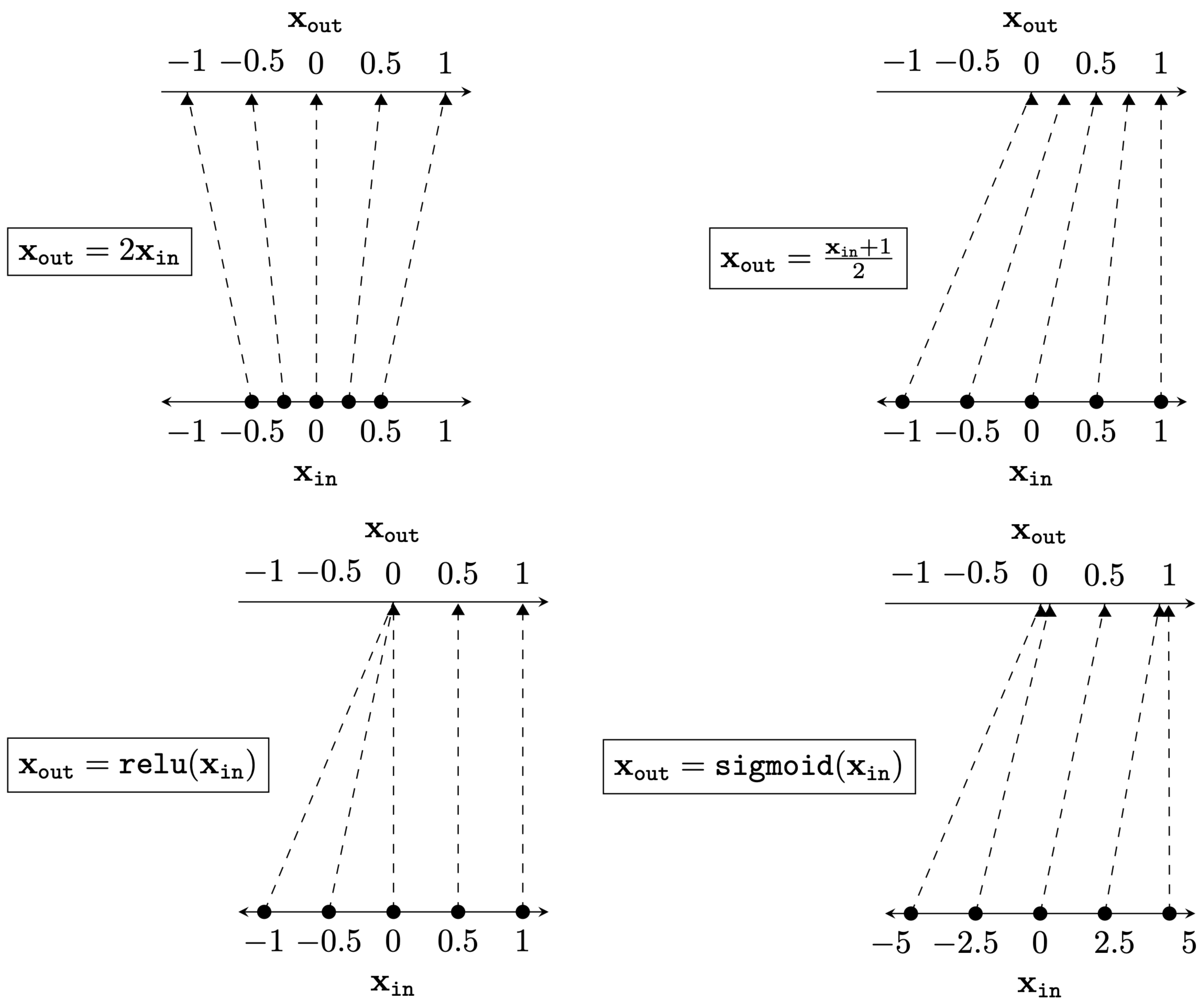
Two different ways to represent a function



Two different ways to represent a function



Data transformations for a variety of neural net layers

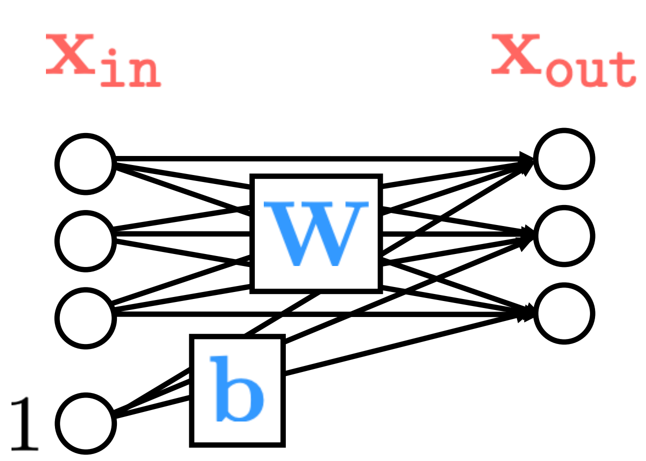


Activations

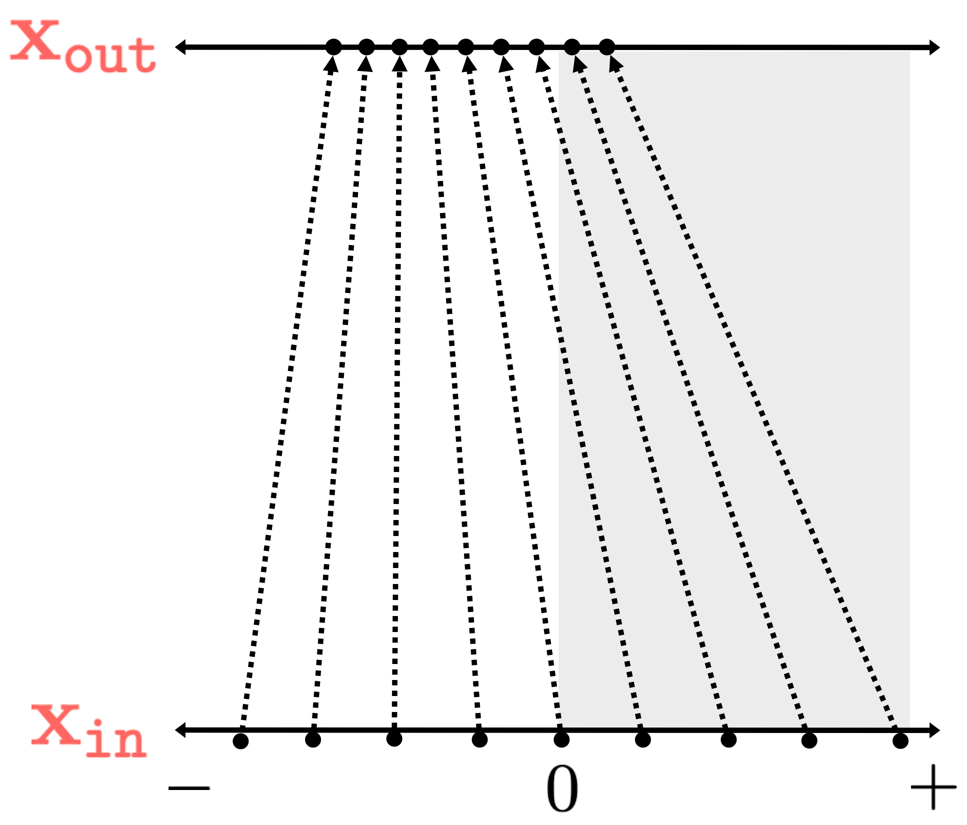
Parameters

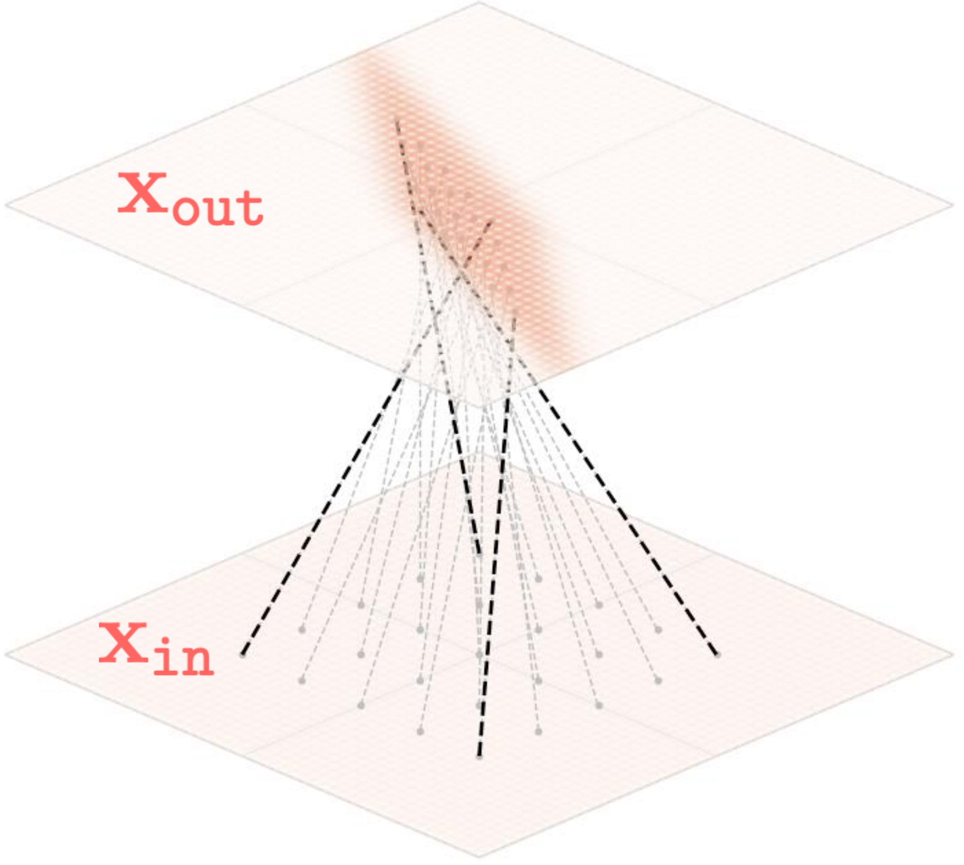
Wiring graph	Equation	Mapping 1D	Mapping 2D

`linear`

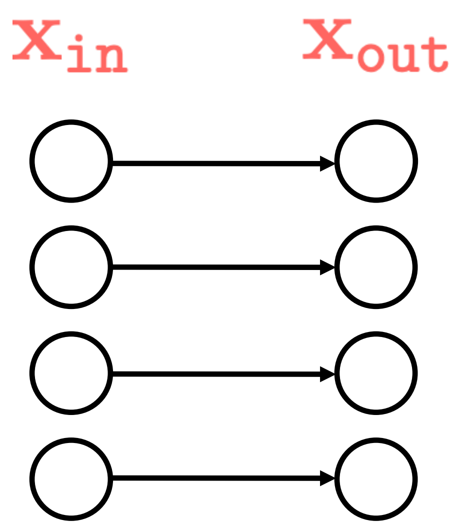


$$\mathbf{x}_{\texttt{out}} = \mathbf{W}\mathbf{x}_{\texttt{in}} + \mathbf{b}$$

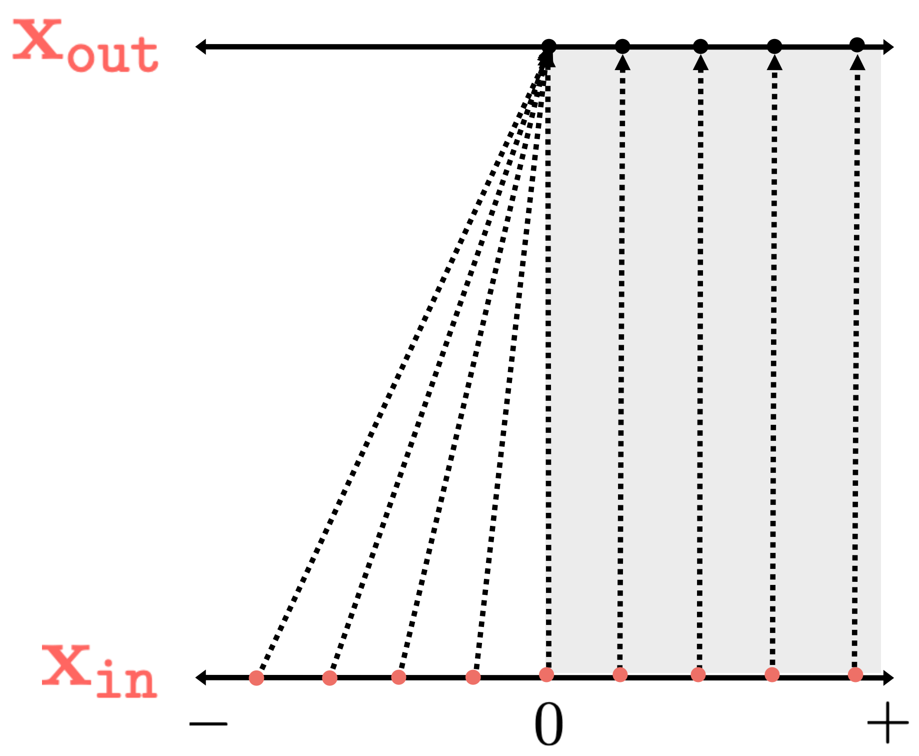


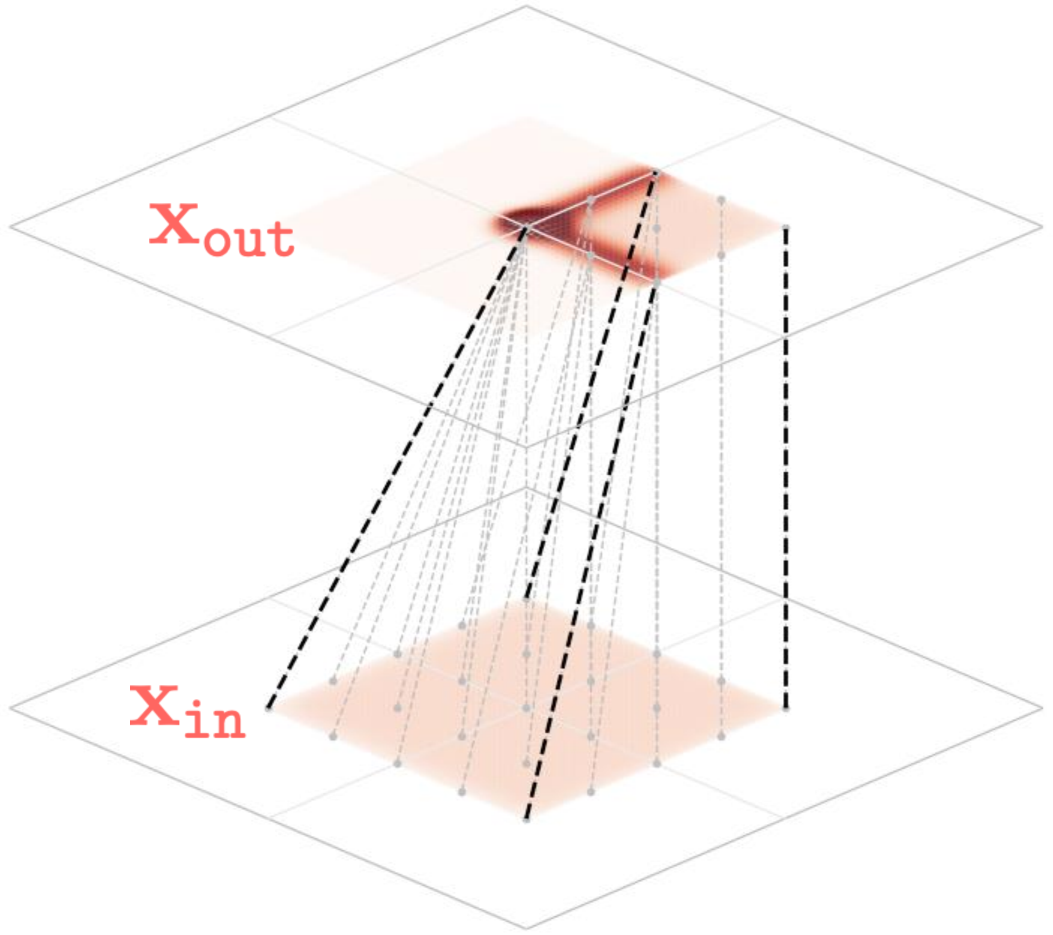


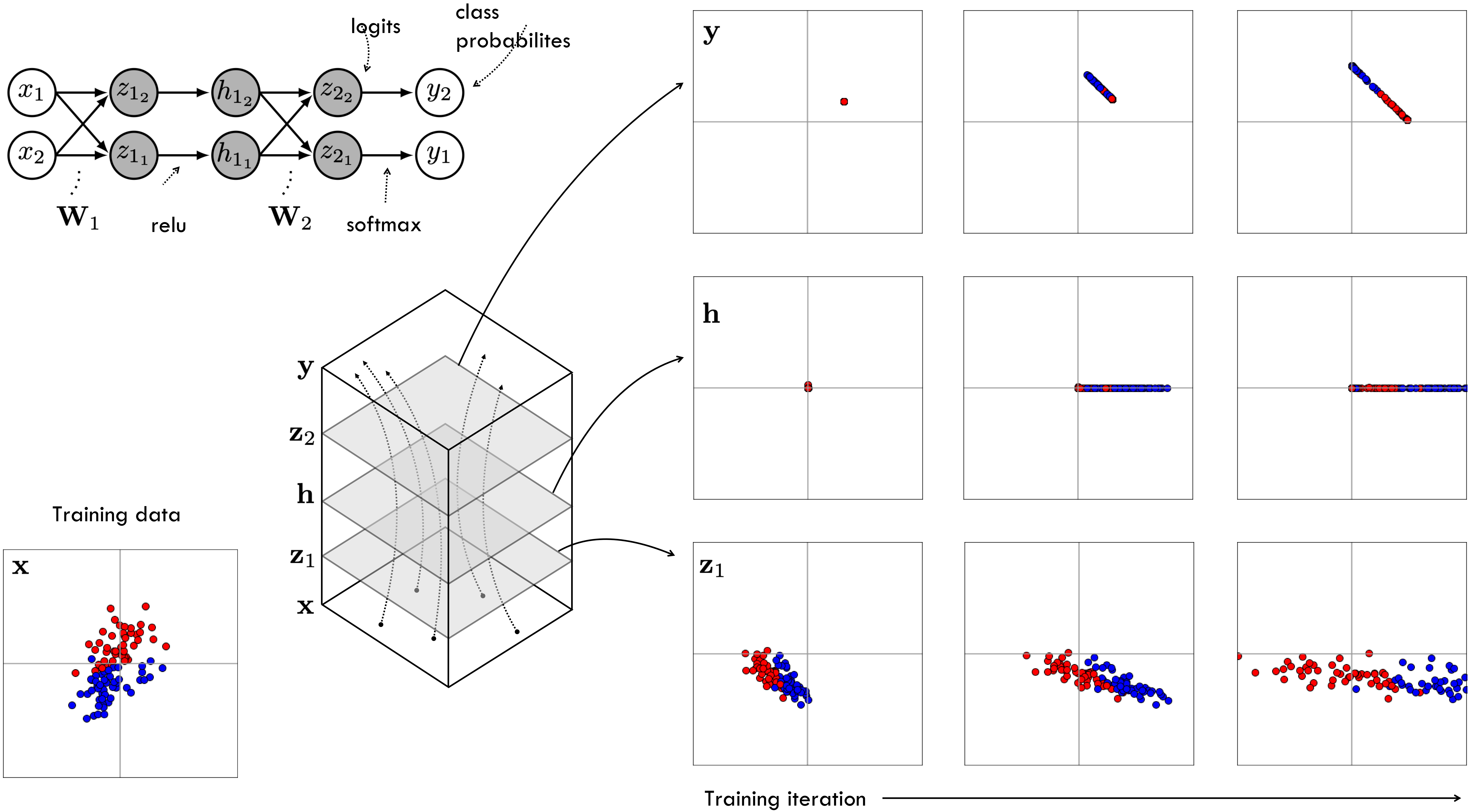
`relu`

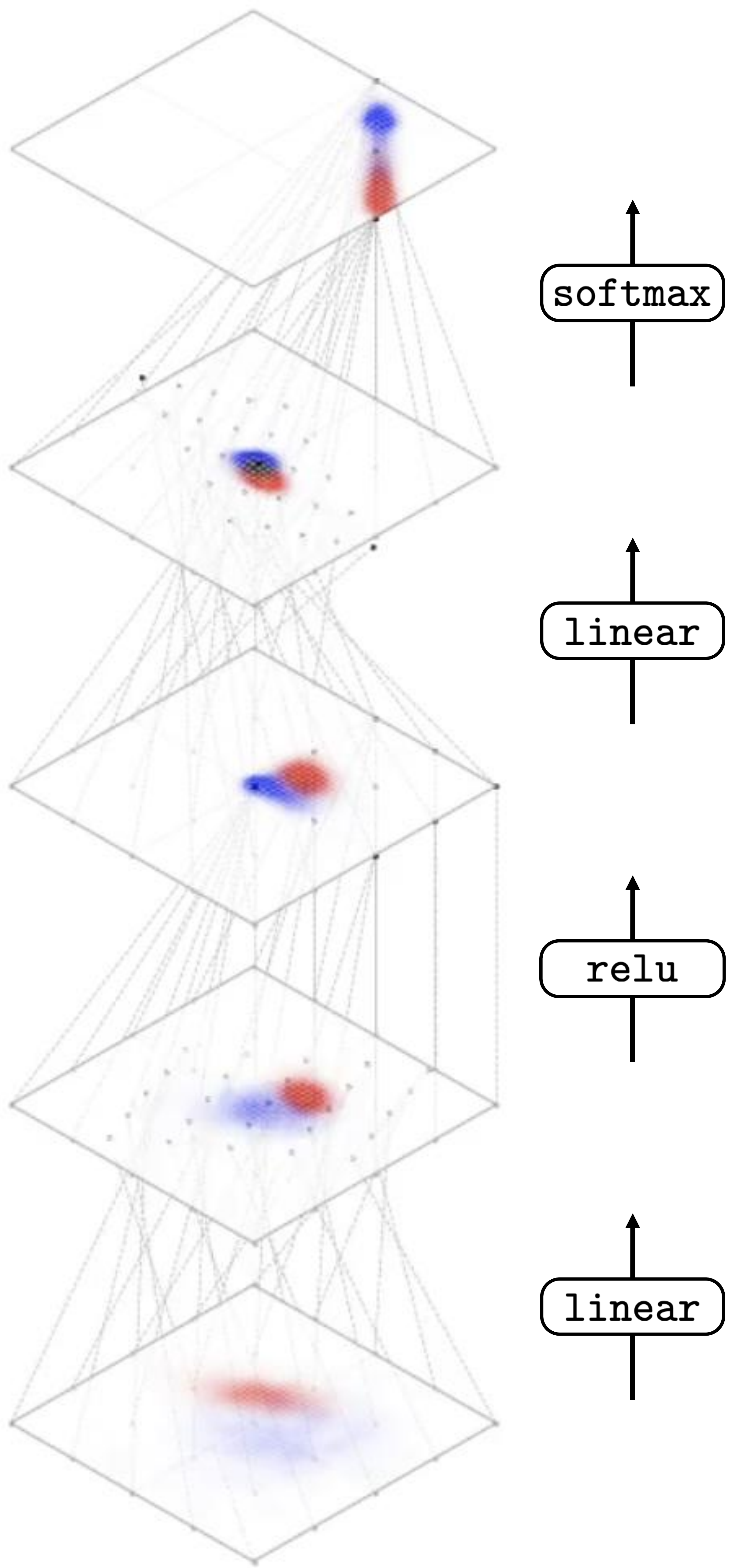


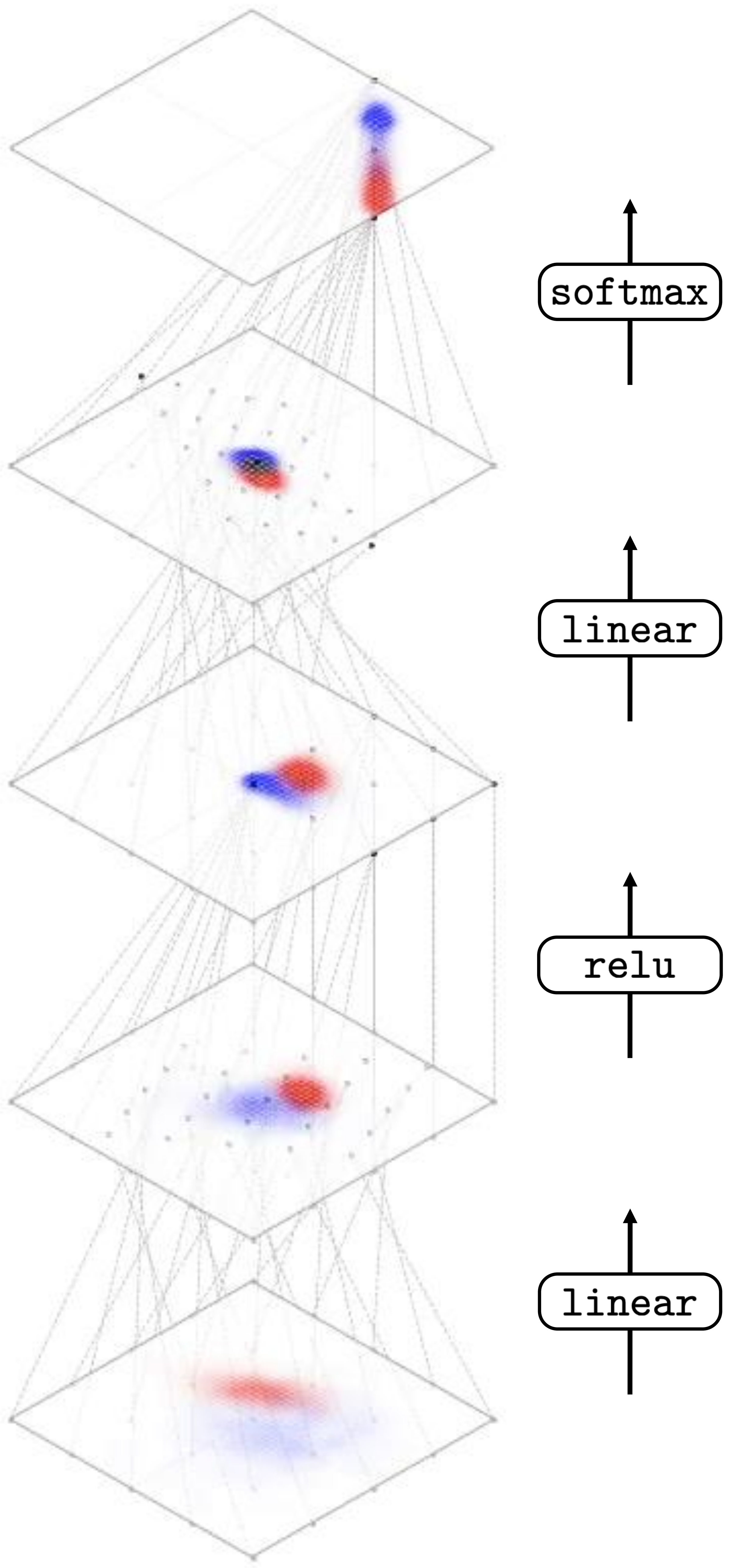
$$x_{\texttt{out}_i} = \max(x_{\texttt{in}_i}, 0)$$

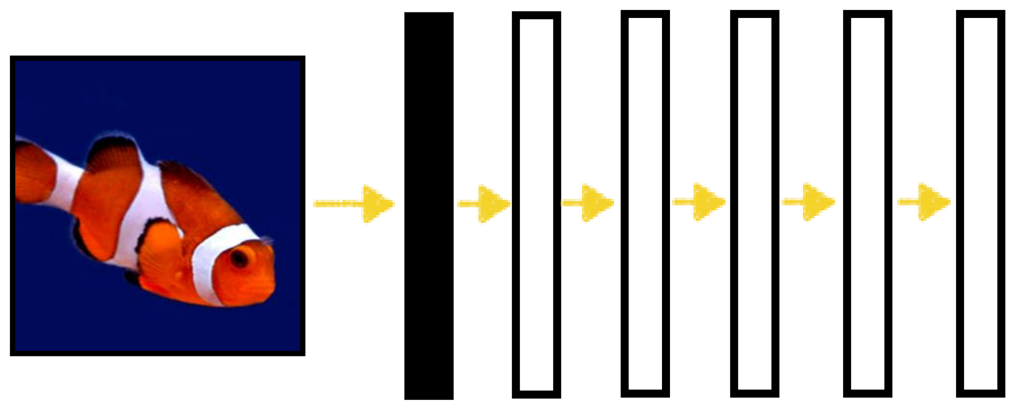




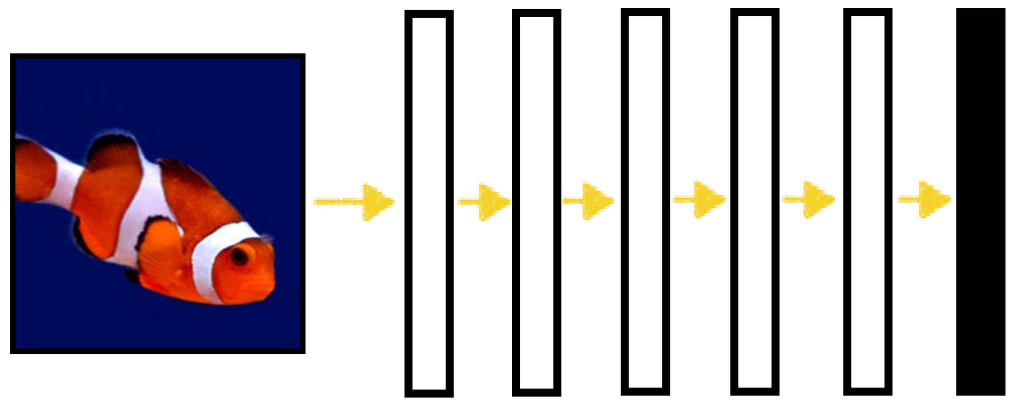

logits
class probabilites
x_1
x_2
z_{1_2}
z_{1_1}
h_{1_2}
h_{1_1}
z_{2_2}
z_{2_1}
y_2
y_1
$\mathbf{W}_1$
relu
$\mathbf{W}_2$
softmax
$\mathbf{y}$
$\mathbf{z}_2$
$\mathbf{h}$
$\mathbf{z}_1$
$\mathbf{x}$
Training data
Training iteration


softmax
linear
relu
linear

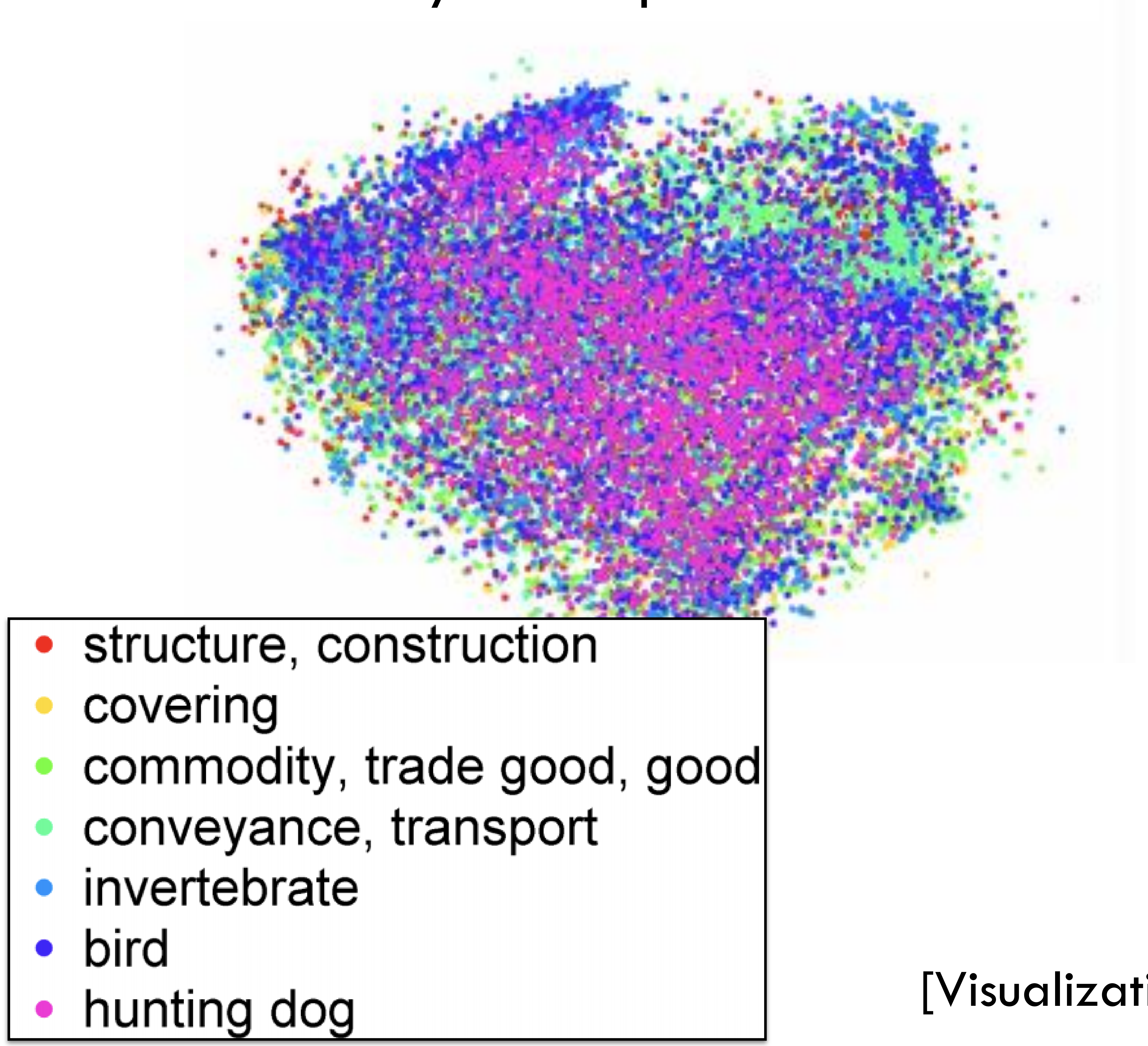

softmax
linear
relu
linear

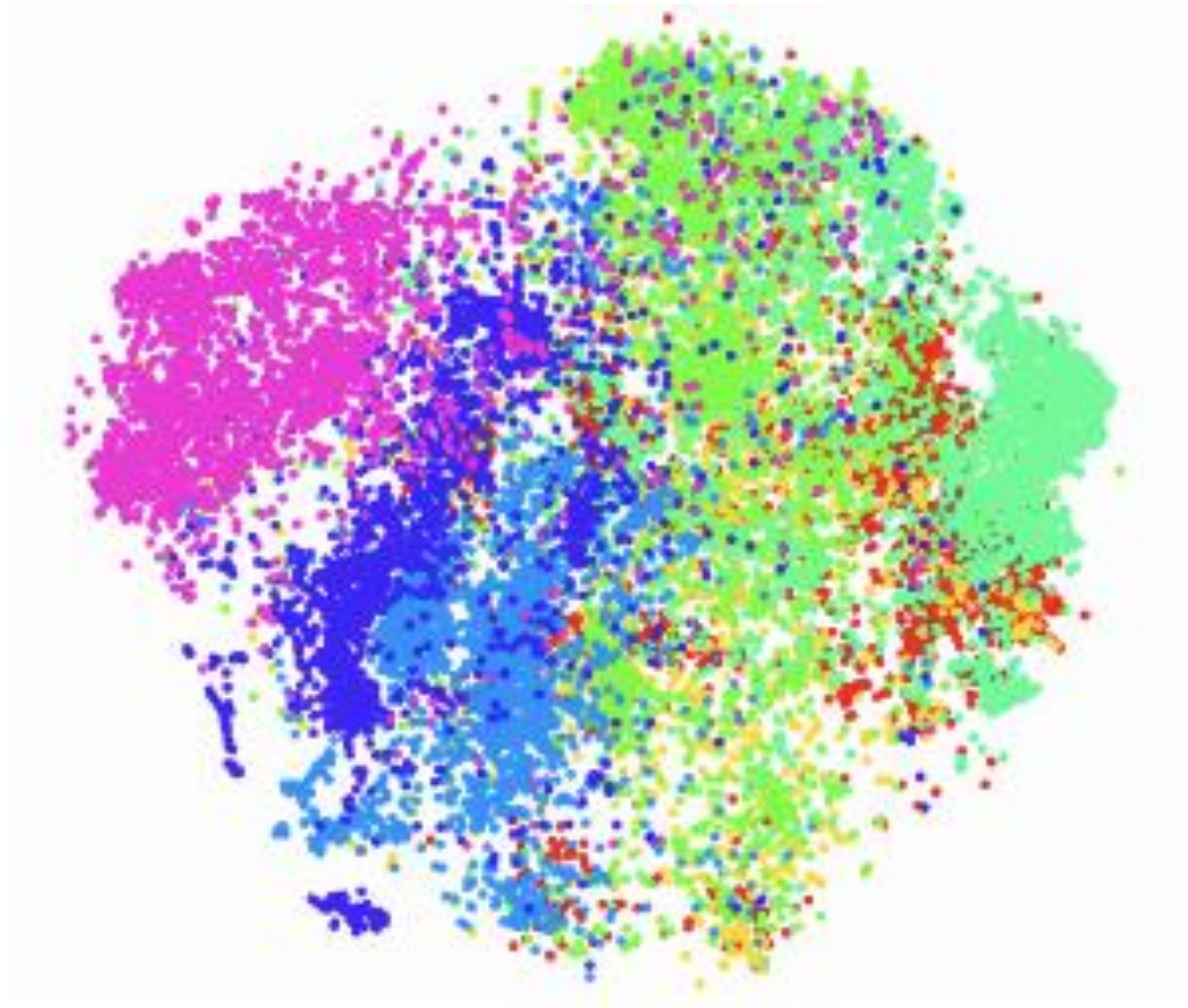


Layer 1 representation



Layer 6 representation





[DeCAF, Donahue, Jia, et al. 2013]

[Visualization technique : t-sne, van der Maaten & Hinton, 2008]

Can be used as a generic feature

("CNN code" = 4096-D vector before classifier)

query image

nearest neighbors in the "code" space