Deep Learning for Visual Computing (COMP0169)

Convolutional Neural Networks

Niloy Mitra

Tobias Ritschel

Introduction

- Idea
- Evolution

Where do filters come from?

- So far we said we will code those filters ourselves
- Which is the right filter for a task?

Quiz

I say "

Input

Output

"?

You say (1, 1, 1)/3

More quiz

I say “   ”? You say (-1,0,1)/2

Input

Output

More quiz, computer helping

I say “

Input

Output

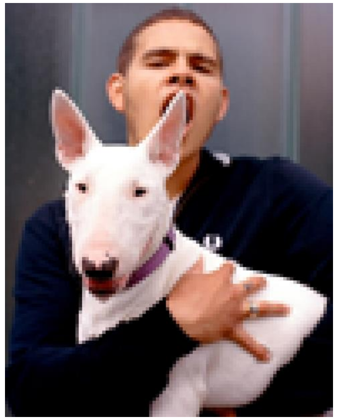
“?

Computer says ... a lot of numbers

Finding a filter by optimization

- Optimize the filter kernel to produce a desired result

$$\arg\min_\theta ||f \odot g_\theta - h||_2$$



kernel

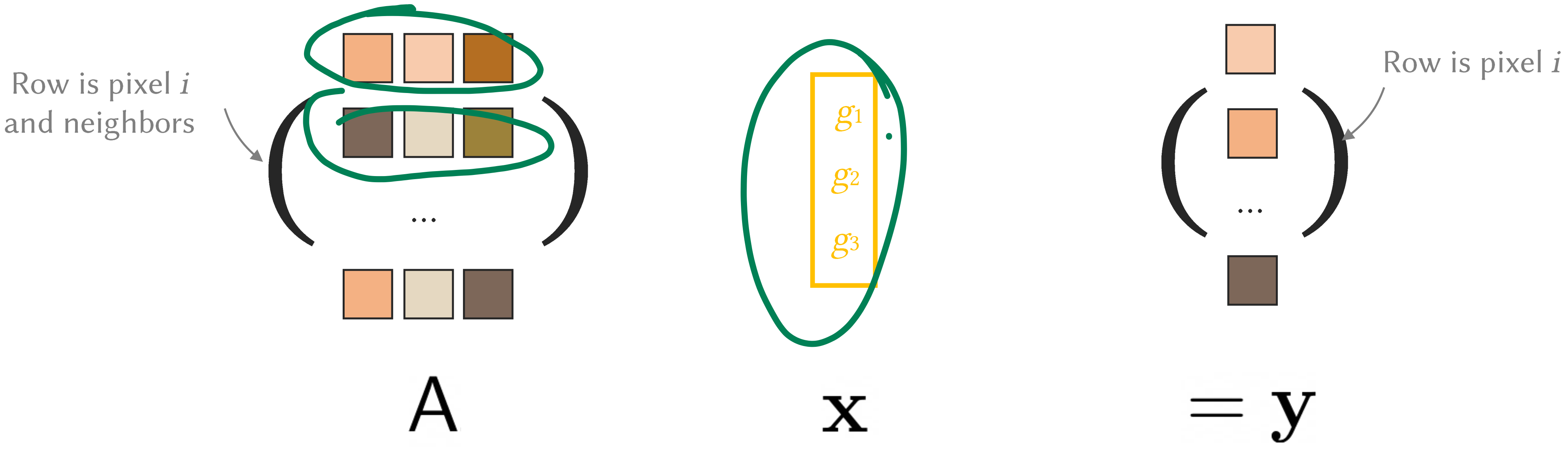
$$\begin{bmatrix} g_{11} & g_{12} & g_{13} \\ g_{21} & g_{22} & g_{23} \\ g_{31} & g_{32} & g_{33} \end{bmatrix}$$

$$\theta \in \mathbb{R}^{3\times 3} = \{g_{11}, \ldots, g_{33}\}$$

Linear filtering

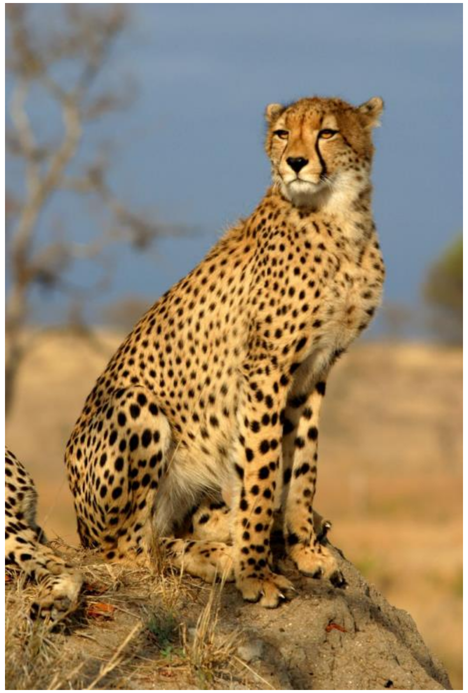
- Convolution is a linear operation, so this can be solved quite easily

$$\arg\min_\theta ||f \odot g_\theta - h||_2$$



The cheetah and the zebra

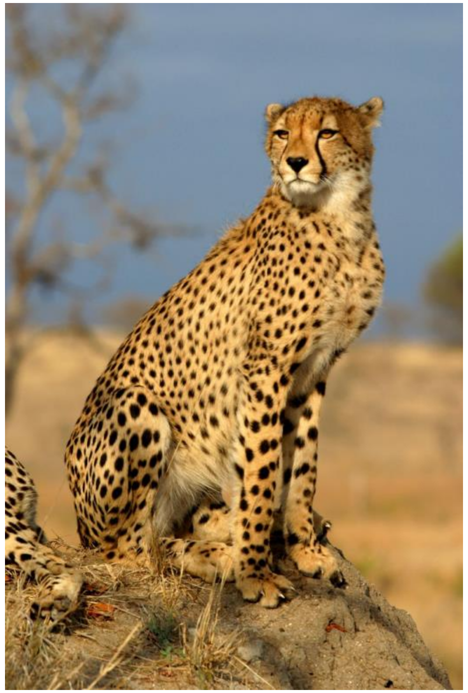
- Could we use this to classify images?



cheetah

zebra

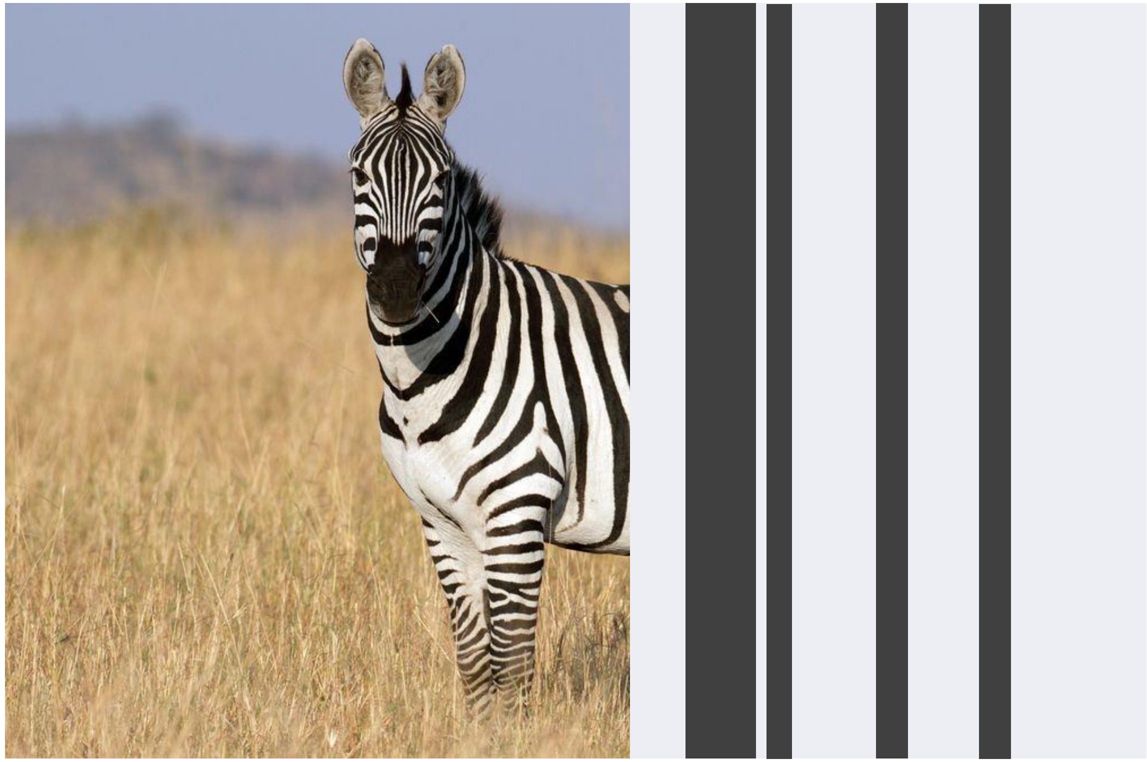
Probably relevant



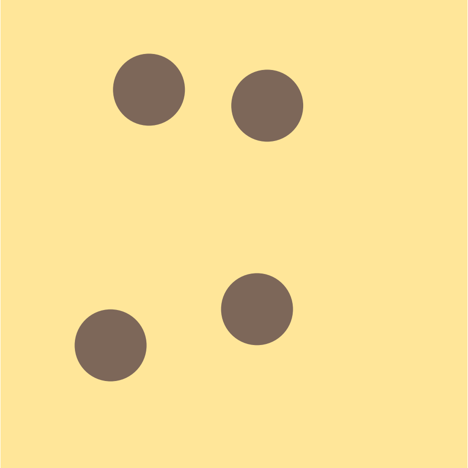
cheetah

zebra

The cheetah and the zebra



Points, edges, non-points, non-edges



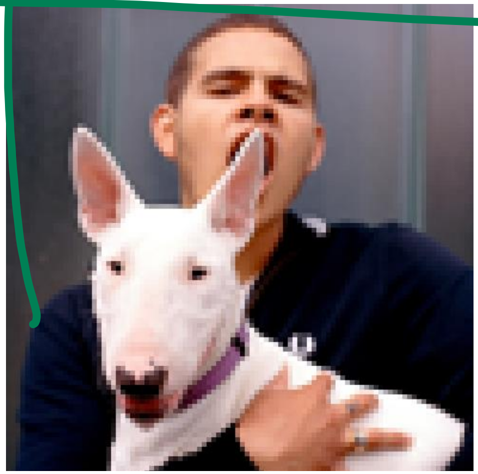
$\odot f_{\theta_1} = 1$

是 cheetah

$\odot f_{\theta_2} = 1$

不是 cheetah

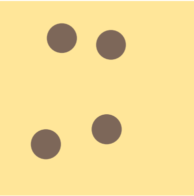
$\odot f_{\theta_1} = -1$



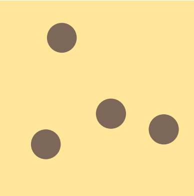
$\odot f_{\theta_2} = -1$

Example solution

$\mathbb{E}$



$\odot g_{\theta,1} = 1$

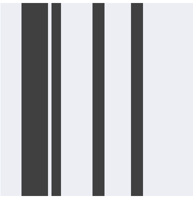


$\odot g_{\theta,1} = 1$

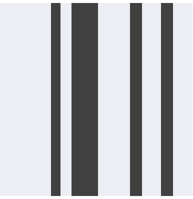
Expectation

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$\mathbb{E}$



$\odot g_{\theta,2} = 1$

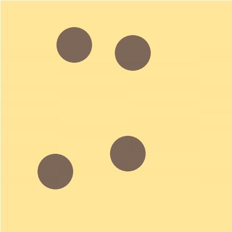
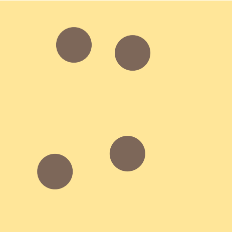


$\odot g_{\theta,2} = 1$

$$\begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$

Non-linearity

- As we saw in the classification and NN lecture, **non-linearity** is important

$\mathbb{E}[\phi($  $\odot g_\theta)] \neq \phi(\mathbb{E}[$  $\odot g_\theta])$

Non-linearity

Non-linearity

- And after that no-linearity, we could again do a convolution
- We call this **layers**

$$\mathbb{E}[\phi_1($$

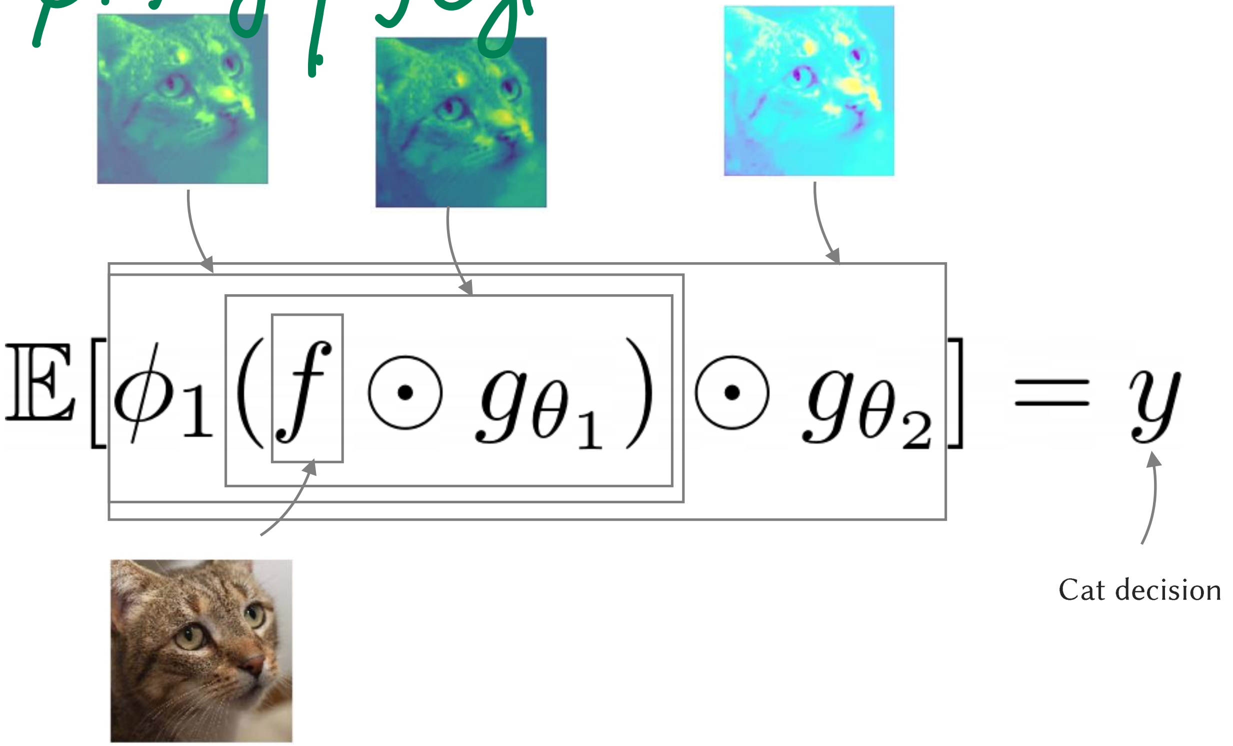
$$\odot g_{\theta_1}) \odot g_{\theta_2}] \neq$$

$$\mathbb{E}[\phi_1($$

$$\odot (g_{\theta_1} \odot g_{\theta_2}))]$$

Depth + non-linearity Φ

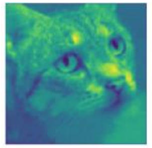
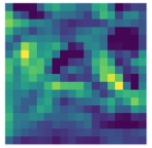
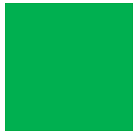
- What this looks like $\phi_1(f \odot g_{\theta_1})$ $f \odot g_{\theta_1}$



$$\mathbb{E}[\phi_1(\boxed{f} \odot g_{\theta_1}) \odot g_{\theta_2}] = y$$

Multi-resolution

- We recall that finding edges and any feature is best done on a pyramid
- CNN so far one resolution
- Solution: insert **pooling** reducing resolution after each convolution

256x256

$$\mathbb{E}[\texttt{pool}(\phi_1(f \odot g_{\theta_1})) \odot g_{\theta_2}]$$

32x32

1x1

提取这区域的主要特征

Fully-connected layer

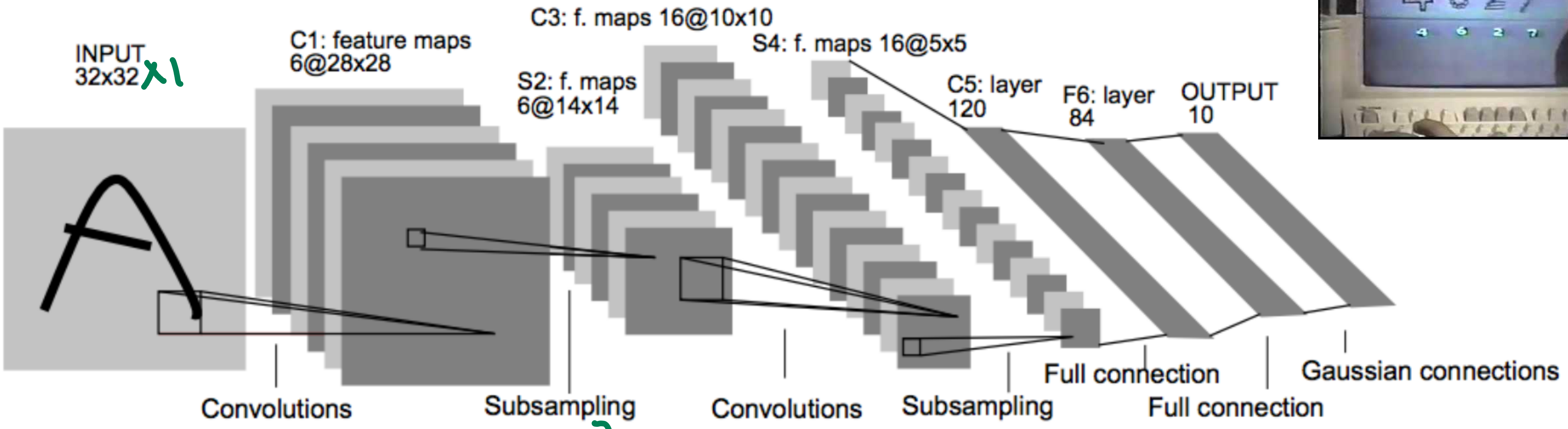
- Instead of expectation, take all pixels and feed them to a classic NN

Map c-times-n vector to number

$$\psi(\mathtt{stack}(\overbrace{\mathtt{pool}(\phi_1(f \odot g_{\theta_1})) \odot g_{\theta_2}}^{32\times 32})) = y$$

Make c-times-n-vector from all n pixels and all c chans

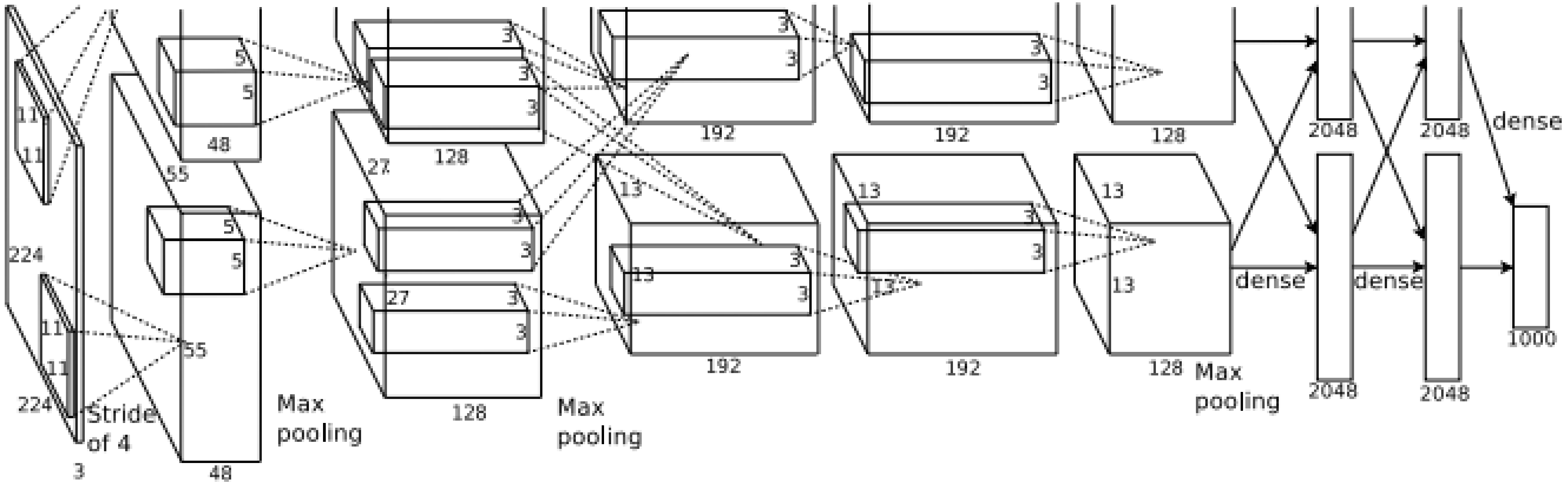
Handwriting recognition: LeNet



就是1个[5x5 kernel 扫描一遍

LeCun et al, 1990: Handwritten digit recognition with a back-propagation network

Image classification: AlexNet



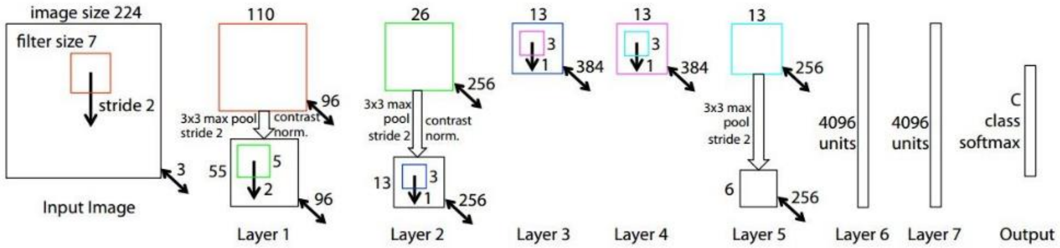
Krizhevsky et al, 2012: *ImageNet Classification with Deep Convolutional Neural Networks*

What happened between 1992 and 2012?

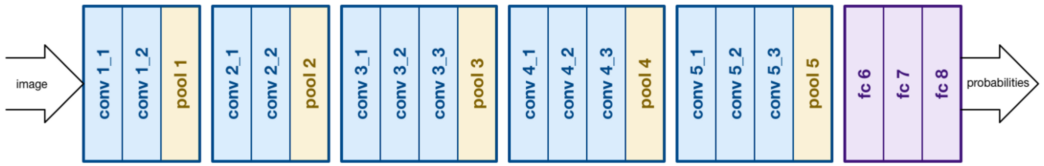
Image data	GPUs	Scripting
60k vs 60M	Pentium II 0.2 GFLOPS Nvidia GTX 680 2600 GFLOPS	C++ vs CUDA/Python

More architectures

- Not always relevant or clear what happened



GoogLeNet/Inception (2014)

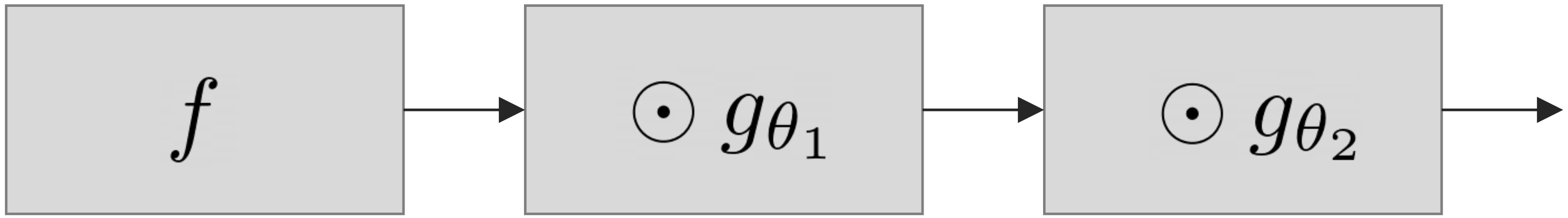


VGG Net (2014)

ResNet, 2015

- Usual sequence makes propagating gradients hard

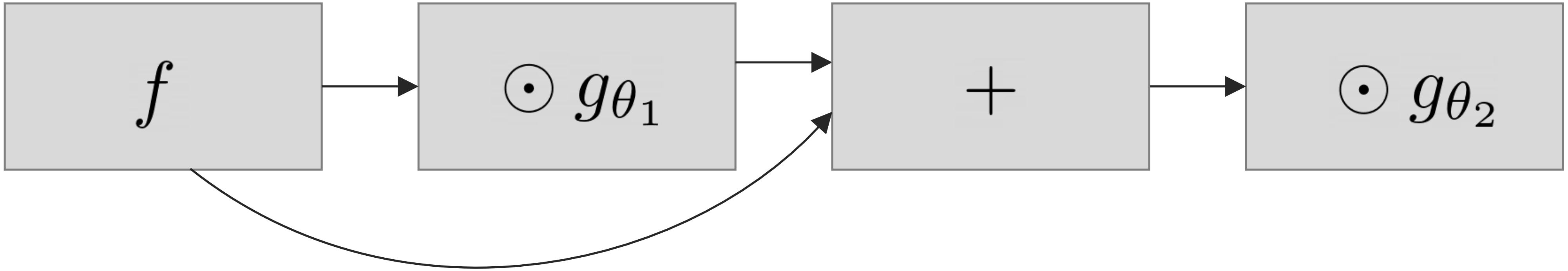
$$\psi(\phi_1(f \odot g_{\theta_1}) \odot g_{\theta_2})$$



ResNet, 2015

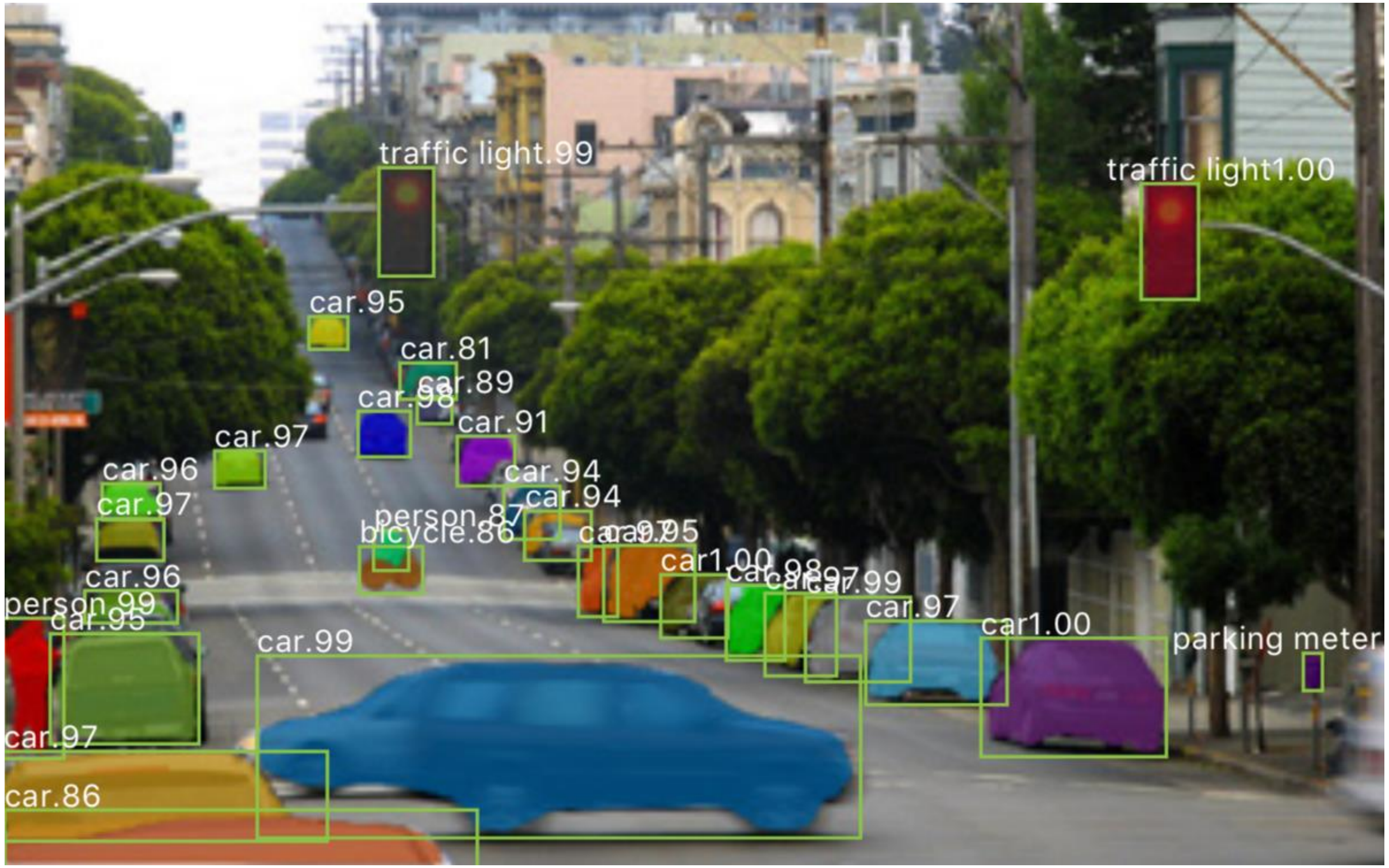
- Usual sequence makes propagating gradients hard

$$\psi((\phi_1(f \odot g_{\theta_1}) + f) \odot g_{\theta_2})$$



Object detection task

- Map image to bounding boxes
- Challenge: Number **unknown**



3D Object detection

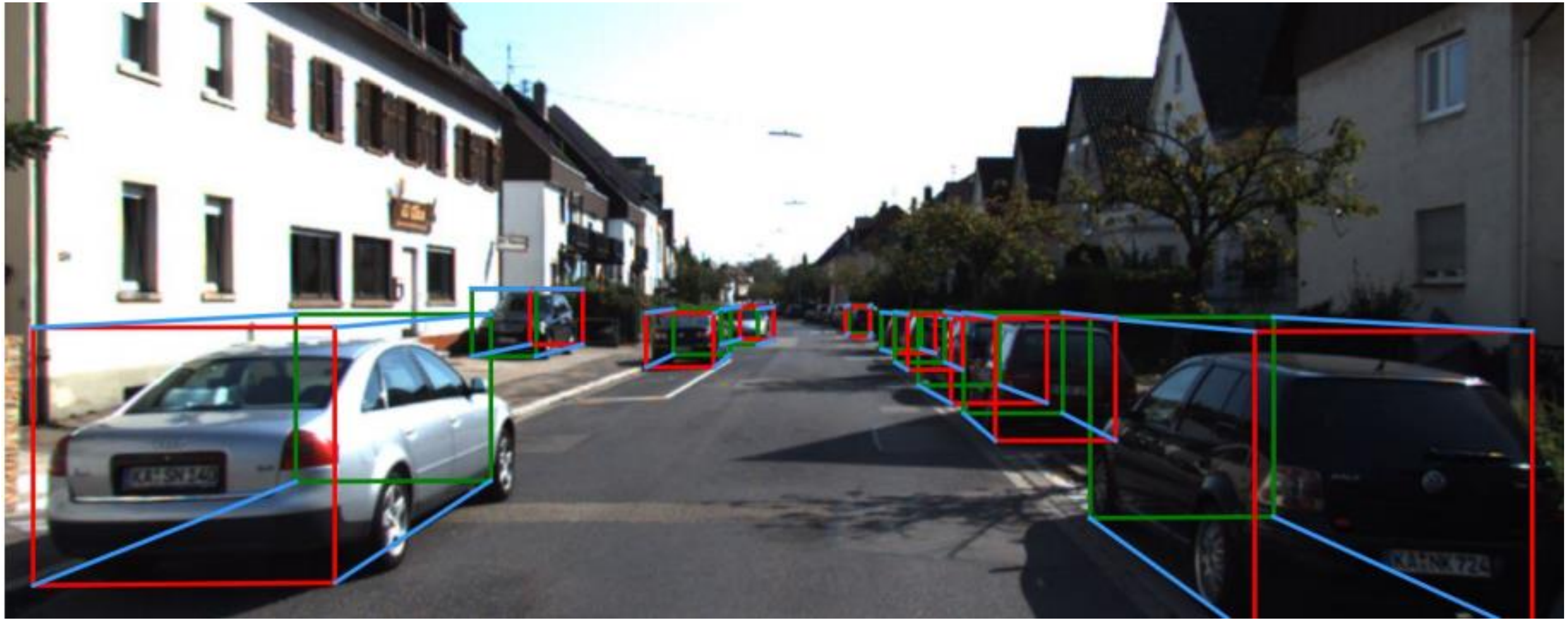
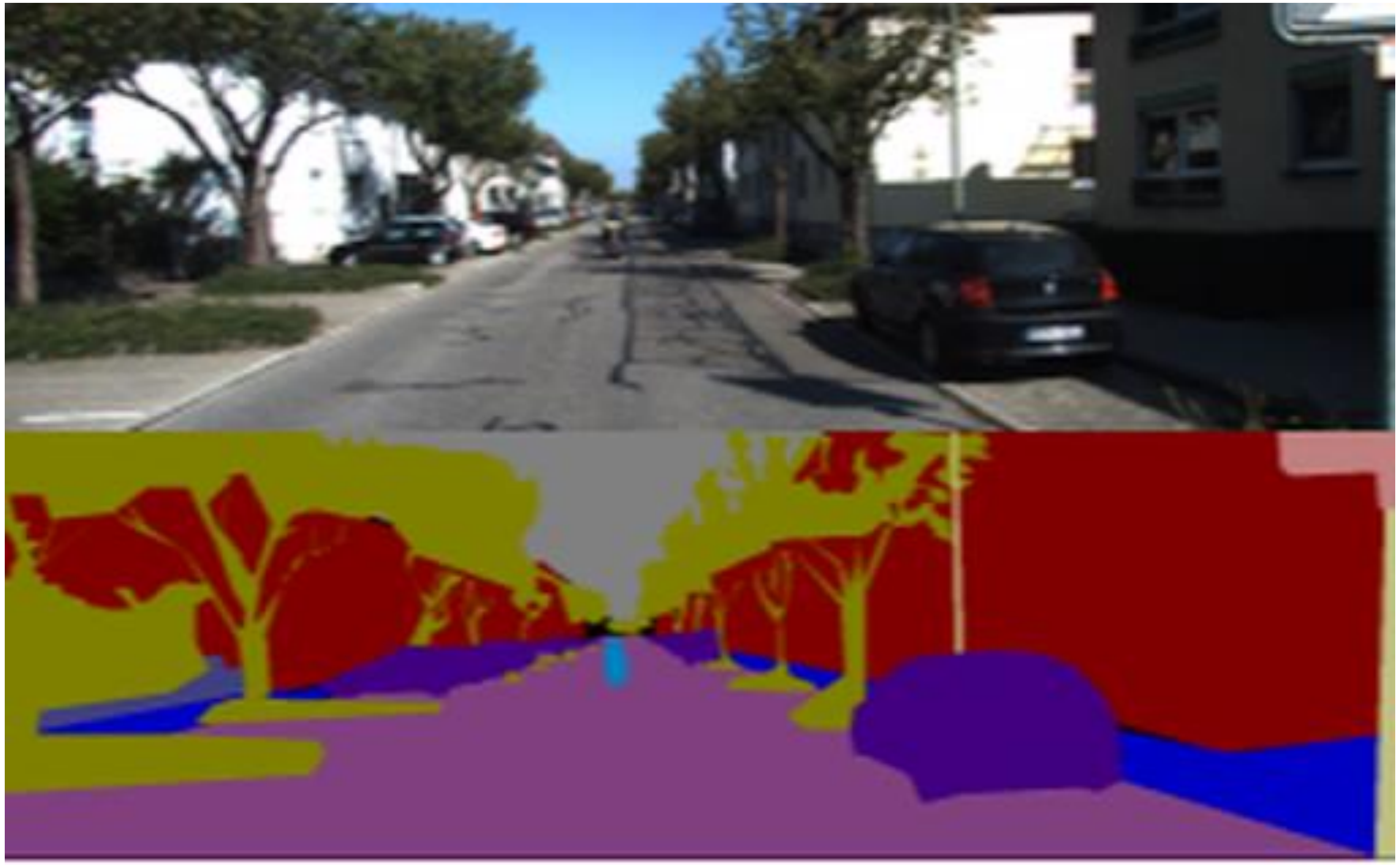


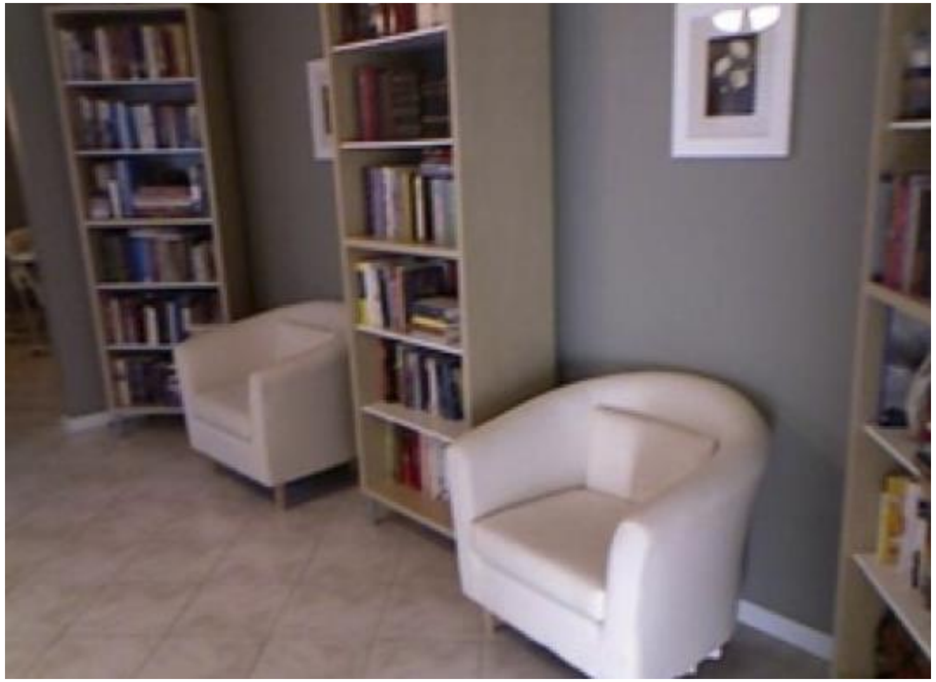
Image-to-image: Segmentation

RGB

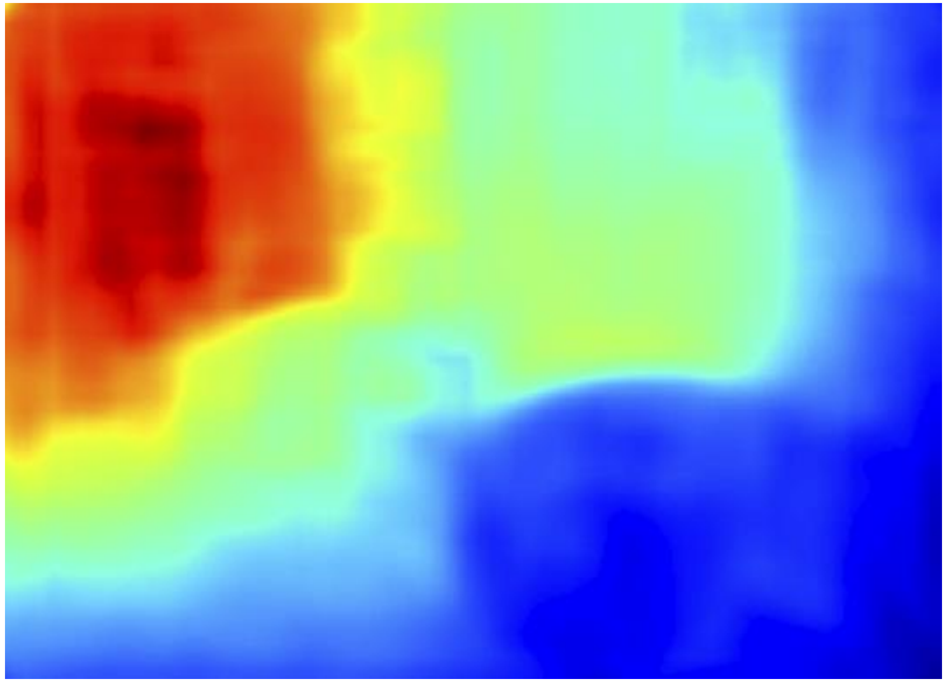


Semantic
segments

Image-to-image



RGB



Depth

Image-to-image

- Image in, number out: Encoder-decoder
- How about image out?

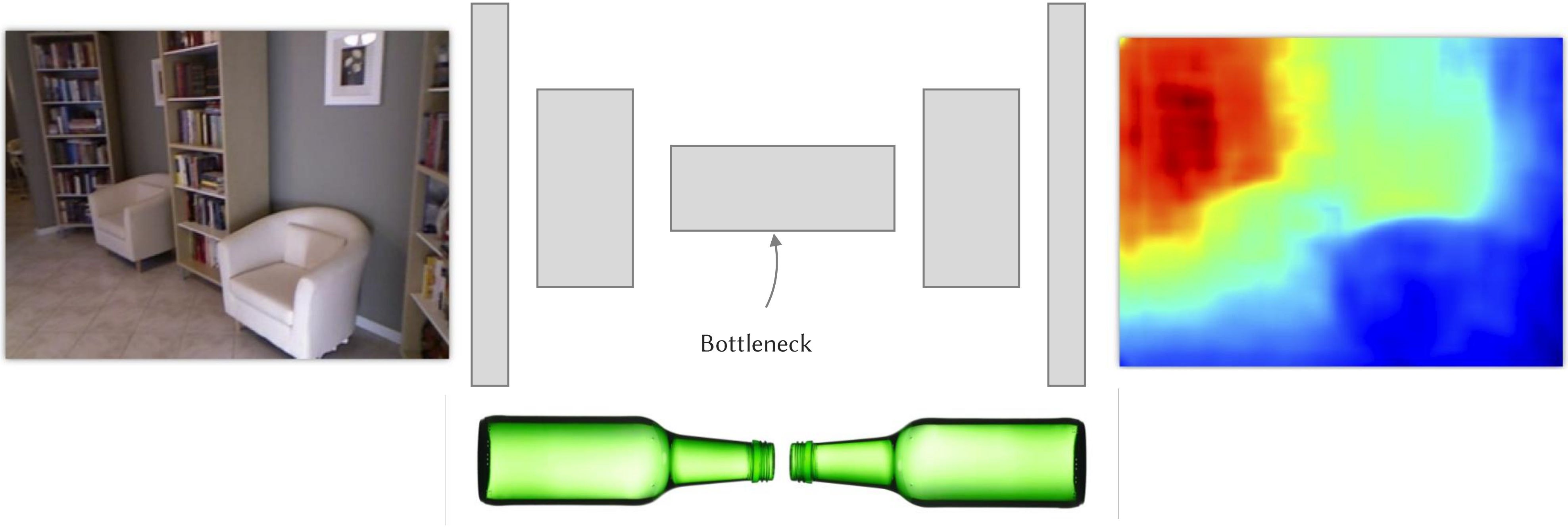
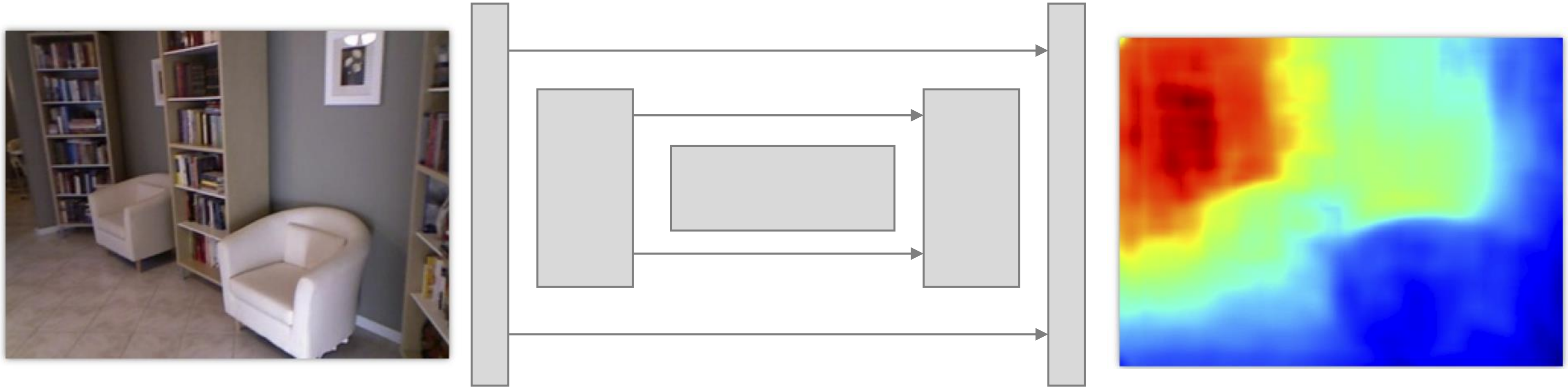
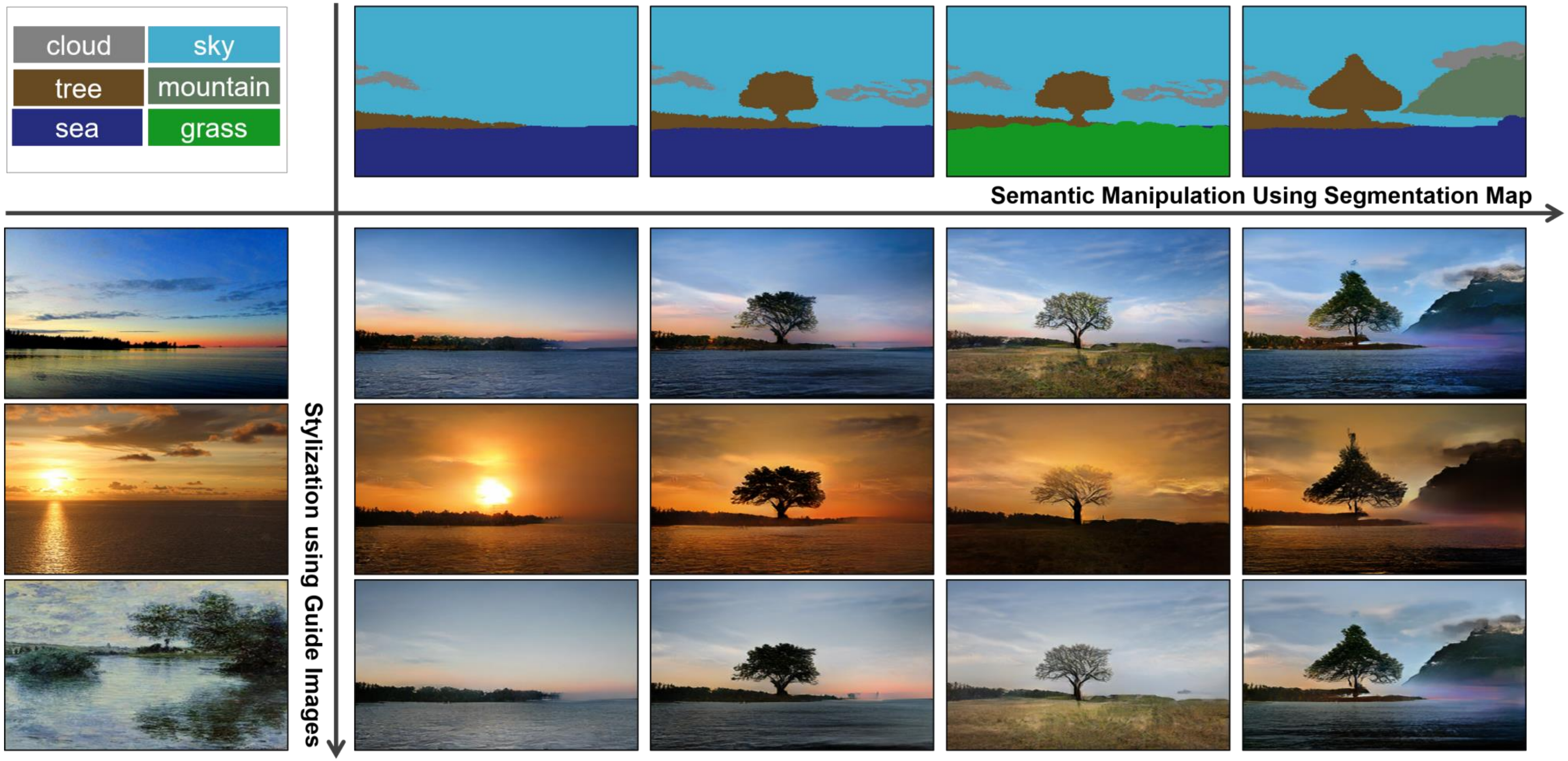


Image-to-image

- No chance to get back details
- Idea: **skip connections,** allow decoder to access encoder state (Unet)



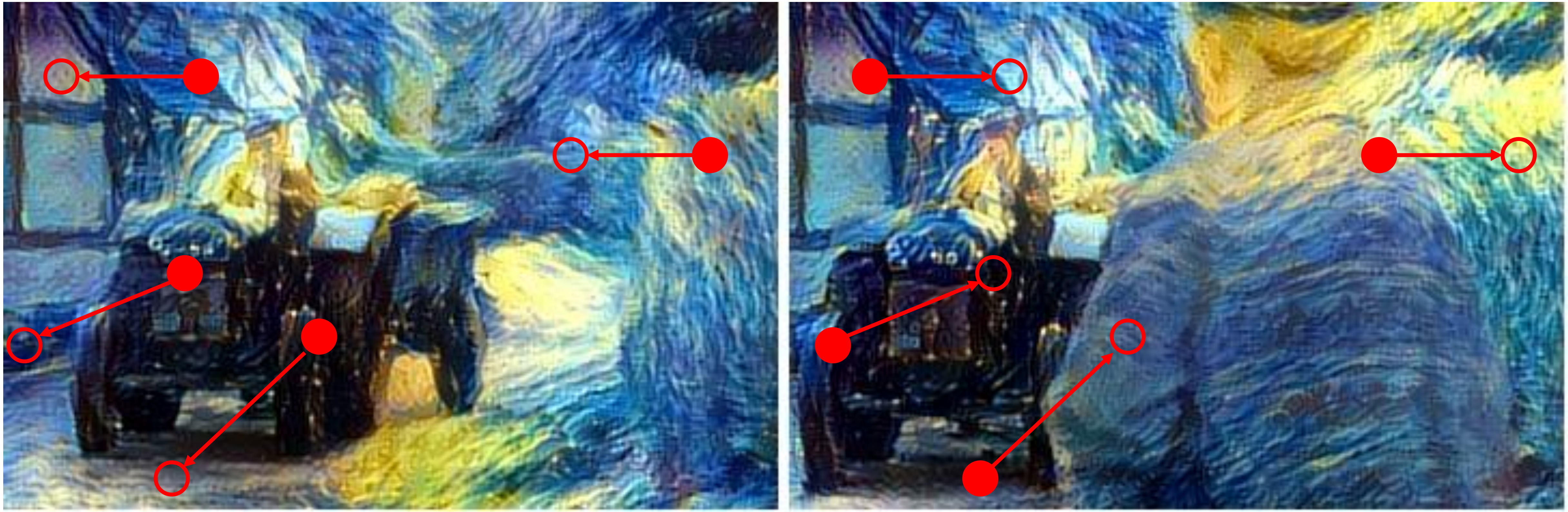
Semantic image synthesis



Style transfer

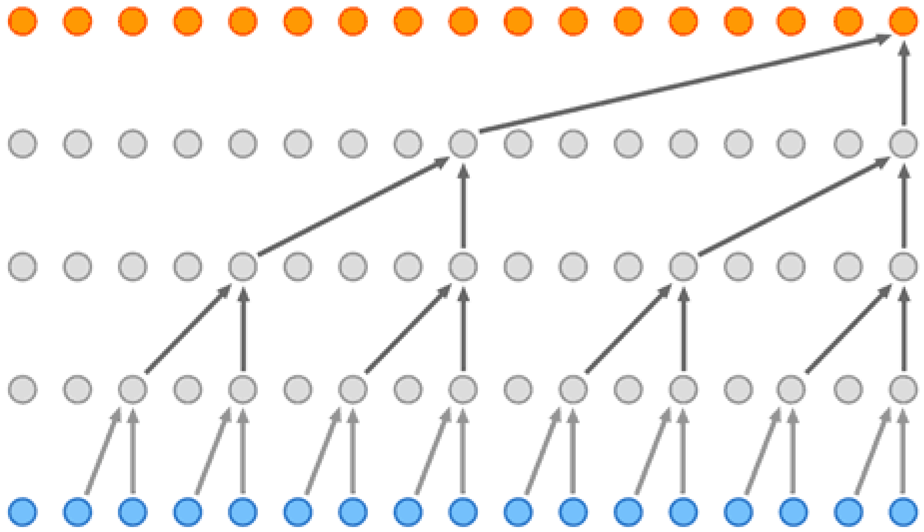
Encoder-decoder for video

- Insert flow-compensated difference into loss



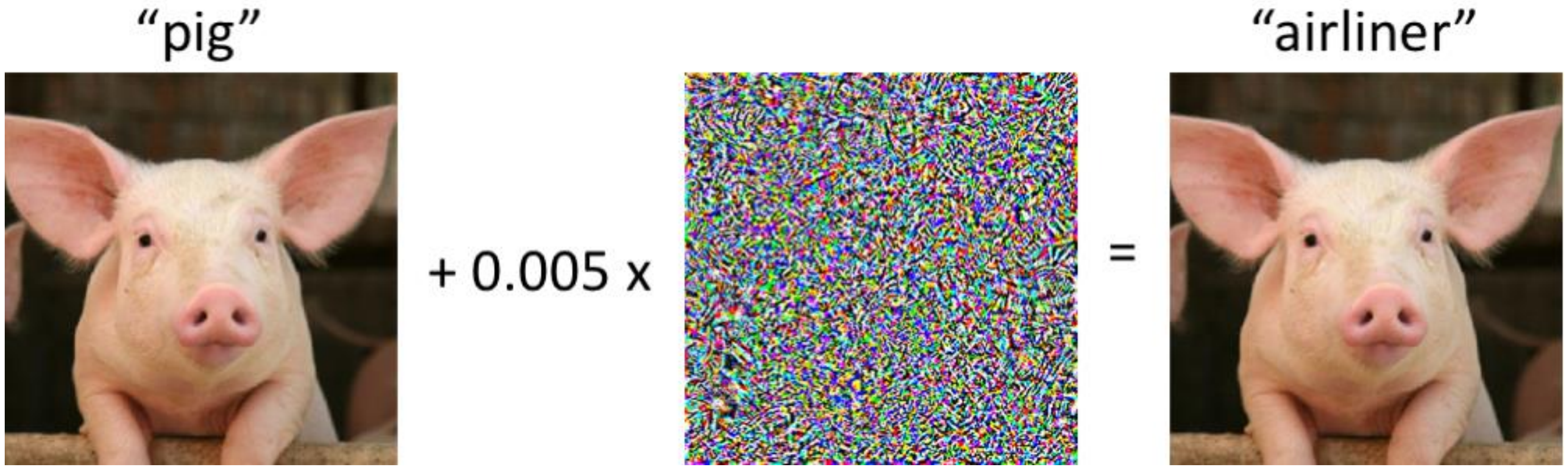
Encoder-Decoder for music

- Wavenet
- Like U-Net, just 1D



Failure

- Adversarial examples
- Small perturbation put detection off
- Found by optimizing for it



Dreaming

Conclusion

- Don’t write filters yourself
- Let the computer find them
- Stack them with non-linearities
- Add a few tricks (skip, res)
- And it does many things