Understanding CNNs using visualisation and transformation analysis

Andrea Vedaldi

Medical Imaging Summer School

August 2016

UNIVERSITY OF OXFORD

Image representations

$\mathbf{x}$

encoder Φ

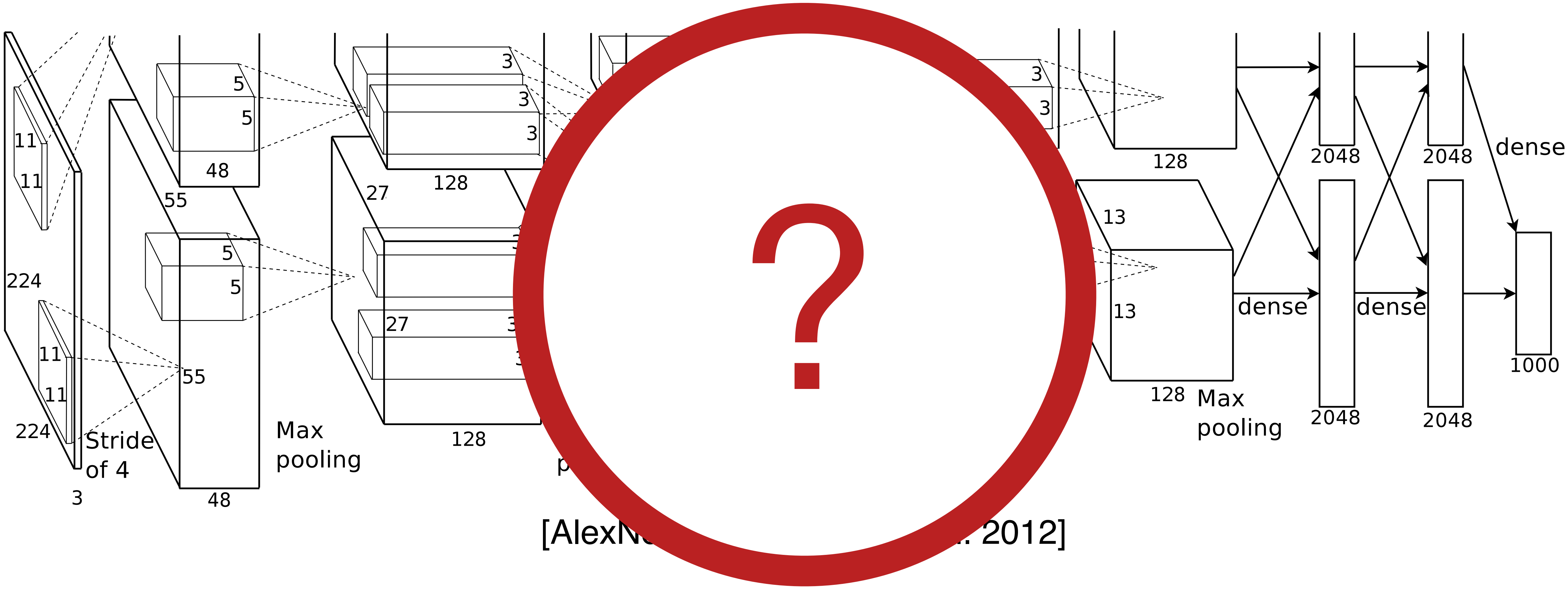
representation

$\Phi(\mathbf{x}) \in \mathbb{R}^d$

An **encoder** maps the data into a **vectorial representation**

Facilitate labelling of images, text, sound, videos, …

Modern convolutional nets



[AlexNe... 2012]

Excellent **performance** in image understanding tasks

Millions of parameters learned from data

Learn a sequence of **general-purpose representations**

The "**meaning**" of the representation is **unclear**

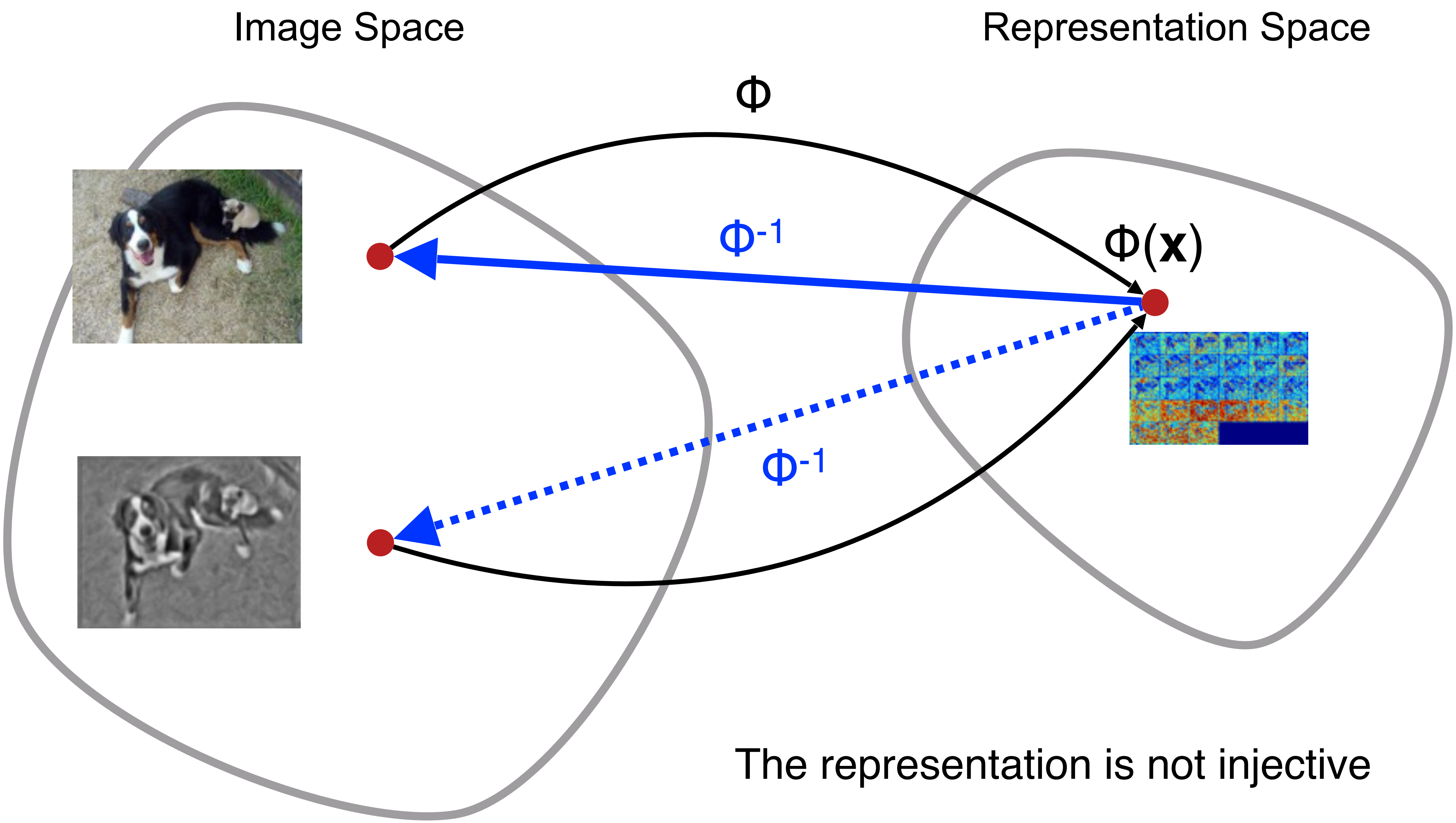
Understanding visual representations

Visualizing representations

Backpropagation networks and “deconvolution”

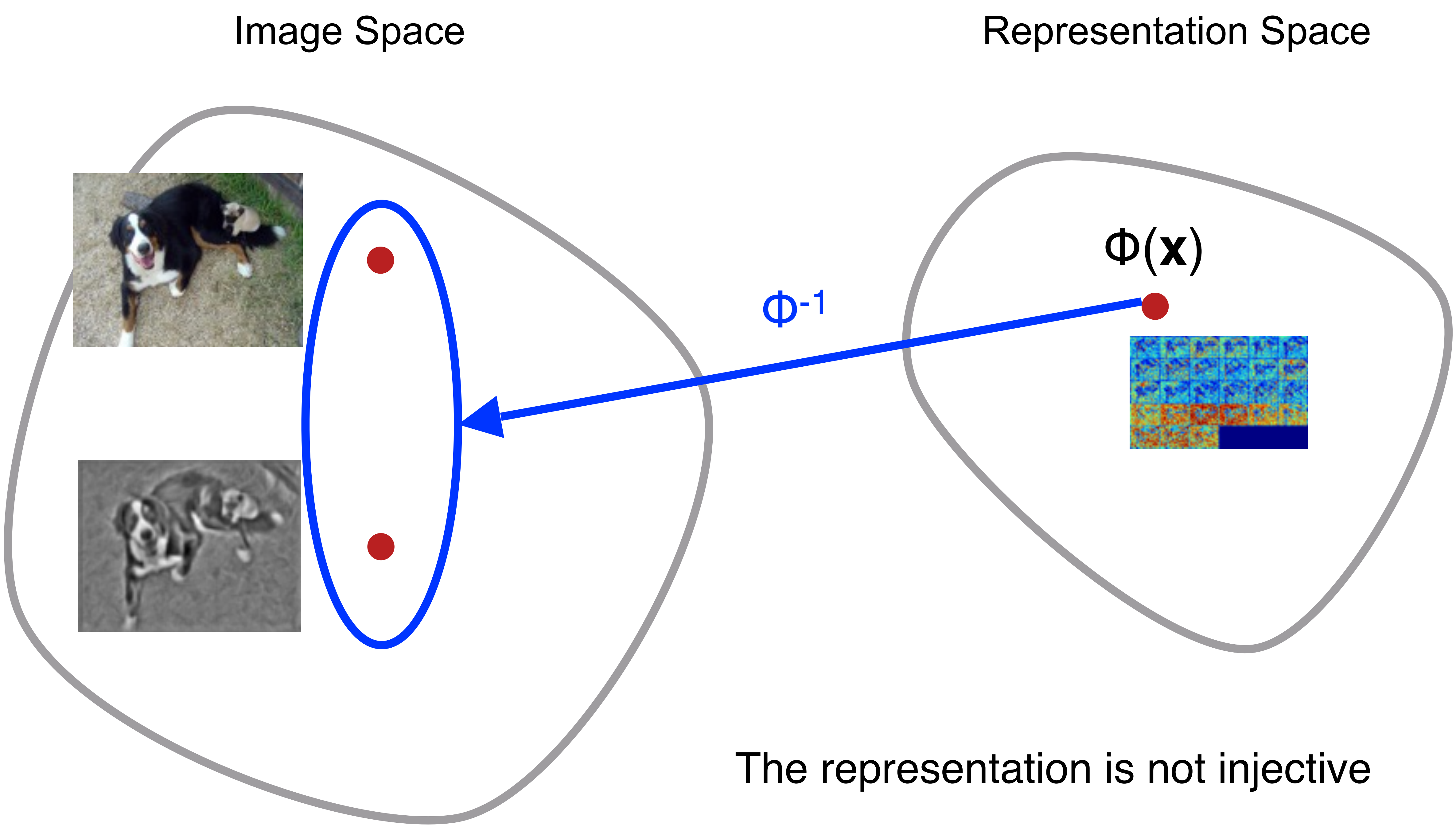
Representations: equivalence & transformations

Visualization: Pre-Image



The representation is not injective

Visualization: Pre-Image



The representation is not injective

The reconstruction ambiguity **provides useful information about the representation**

Finding a Pre-Image

A simple yet general and effective method

$$\min_{\mathbf{x}} \|\Phi(\mathbf{x}) - \Phi_0\|_2^2$$

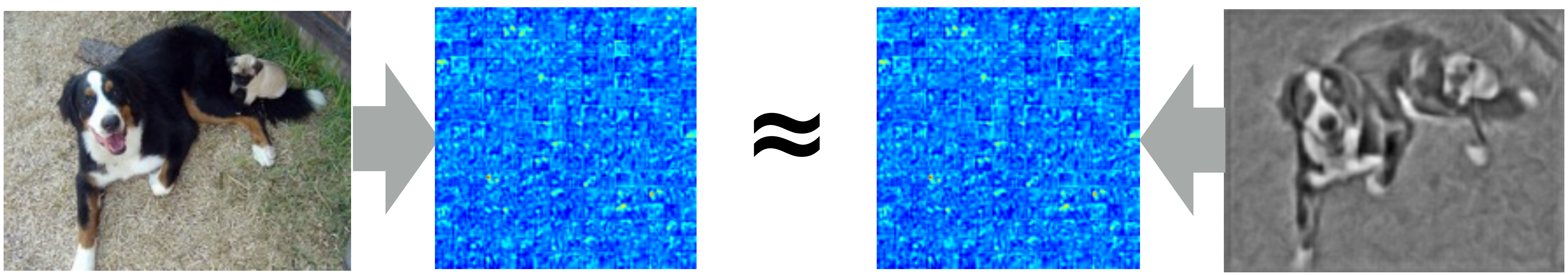


Image Representation Reconstruction Pre-Image

Start from **random noise**

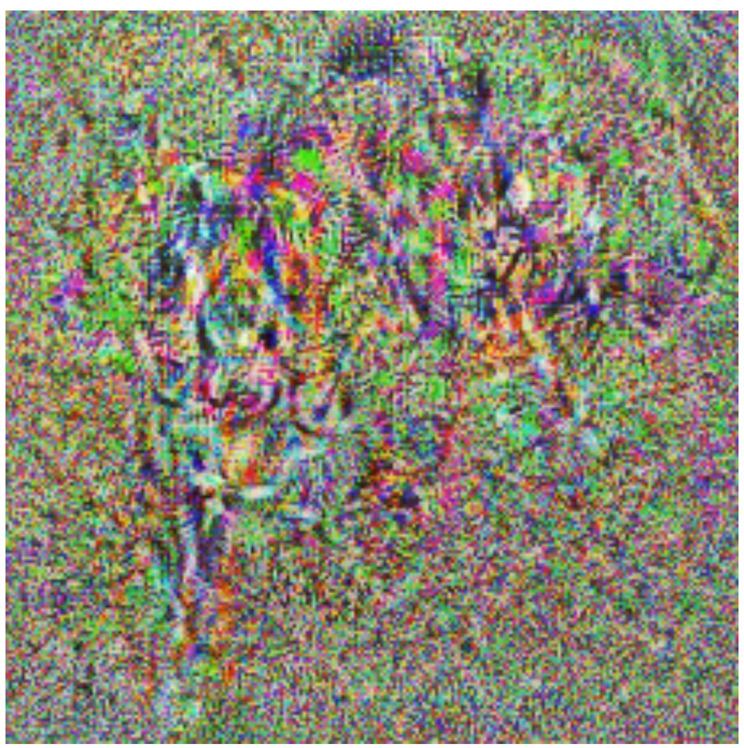
Optimize using stochastic **gradient descent**

Finding a Pre-Image

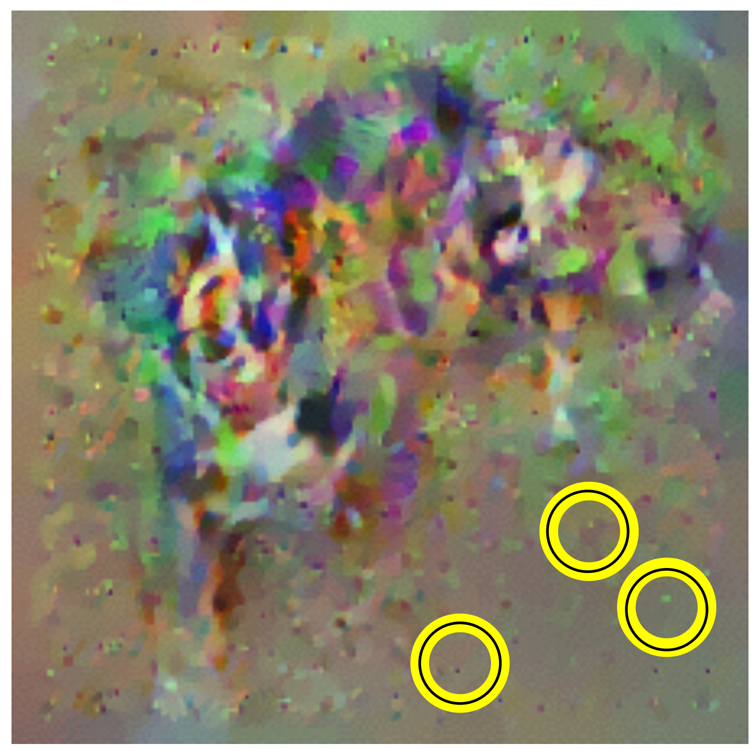
A simple yet general and effective method

$$\min_{\mathbf{x}} \|\Phi(\mathbf{x}) - \Phi_0\|_2^2$$

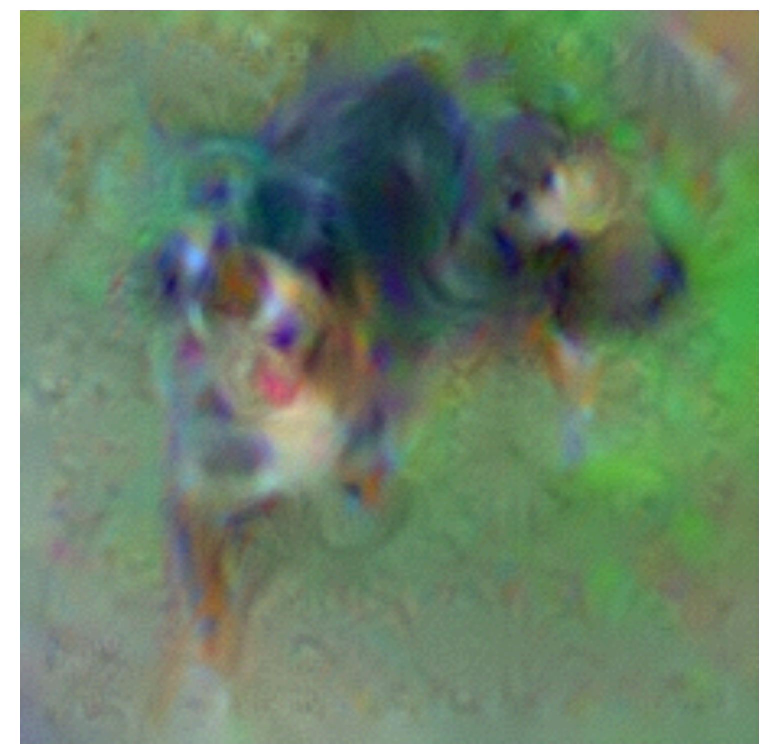
No prior



TV-norm β = 1



TV-norm β = 2



Related Work

Analysis tools

Visualizing higher-layer features of a deep network
Ethan et al. 2009
[intermediate features]

Deep inside convolutional networks
Simonyan et al. 2014
[deepest features, aka "deep dreams"]

DeConvNets
Zeiler et al. In ECCV, 2014
[intermediate features]

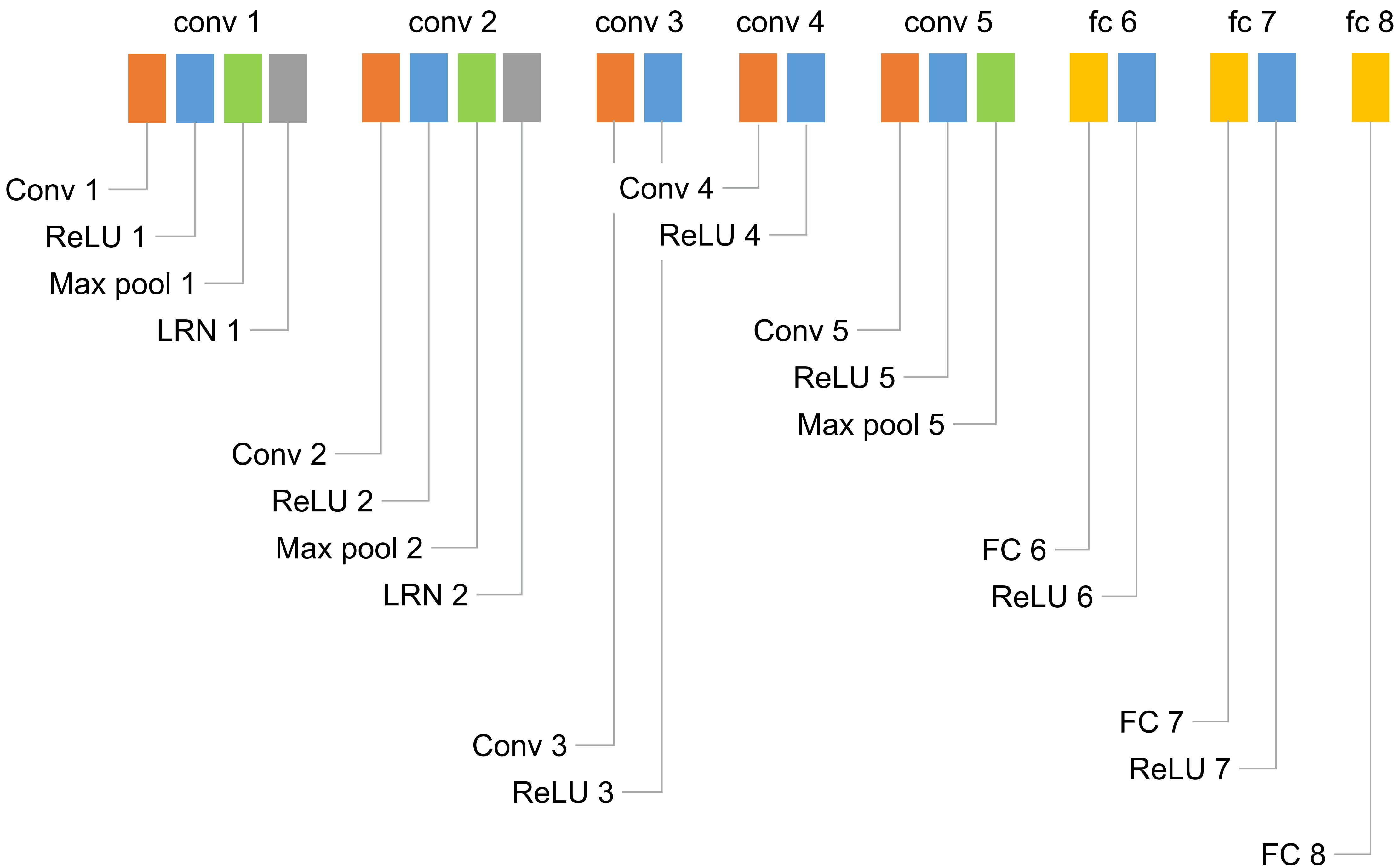
Understanding neural networks through deep visualisation
Yosinksi et al. 2015
[intermediate features]

Artistic tools

Google's "inceptionsm"
Mordvintsev et al. 2015

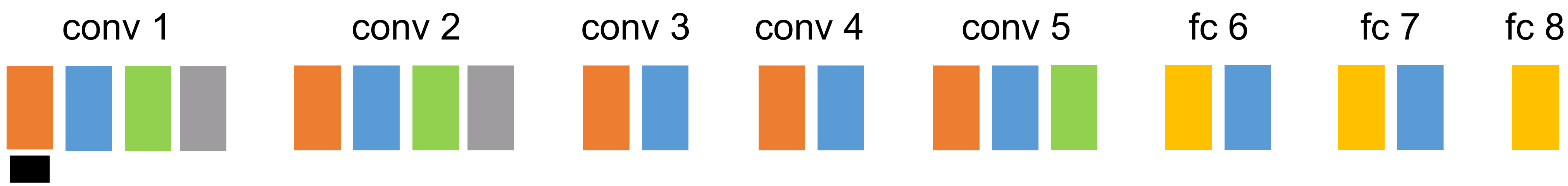
Style synthesis and transfer
Gatys et al. 2015

Inversion



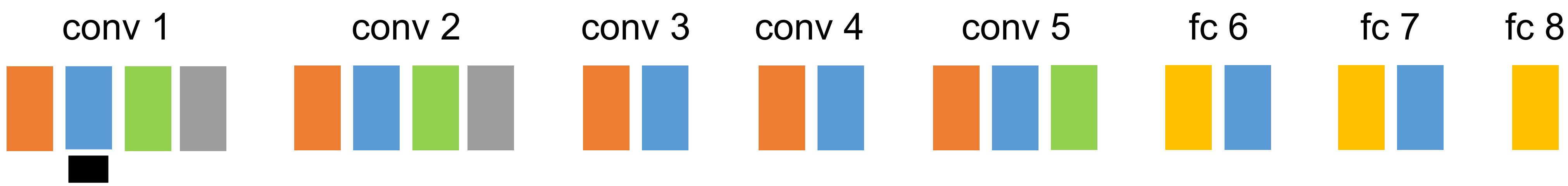
AlexNet
[Krizhevsky et al. 2012]

Inversion



Original Image

Inversion

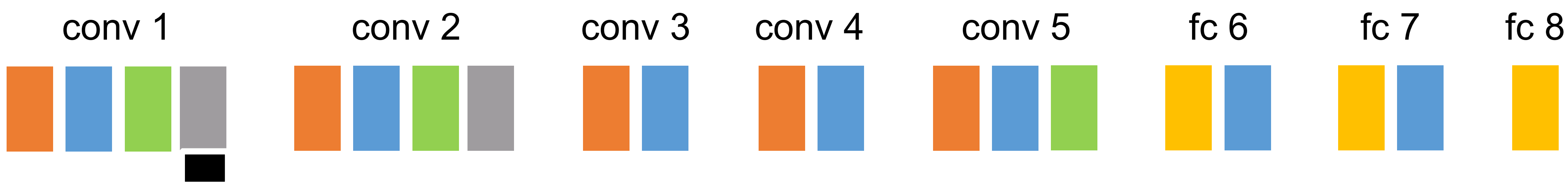


Original Image

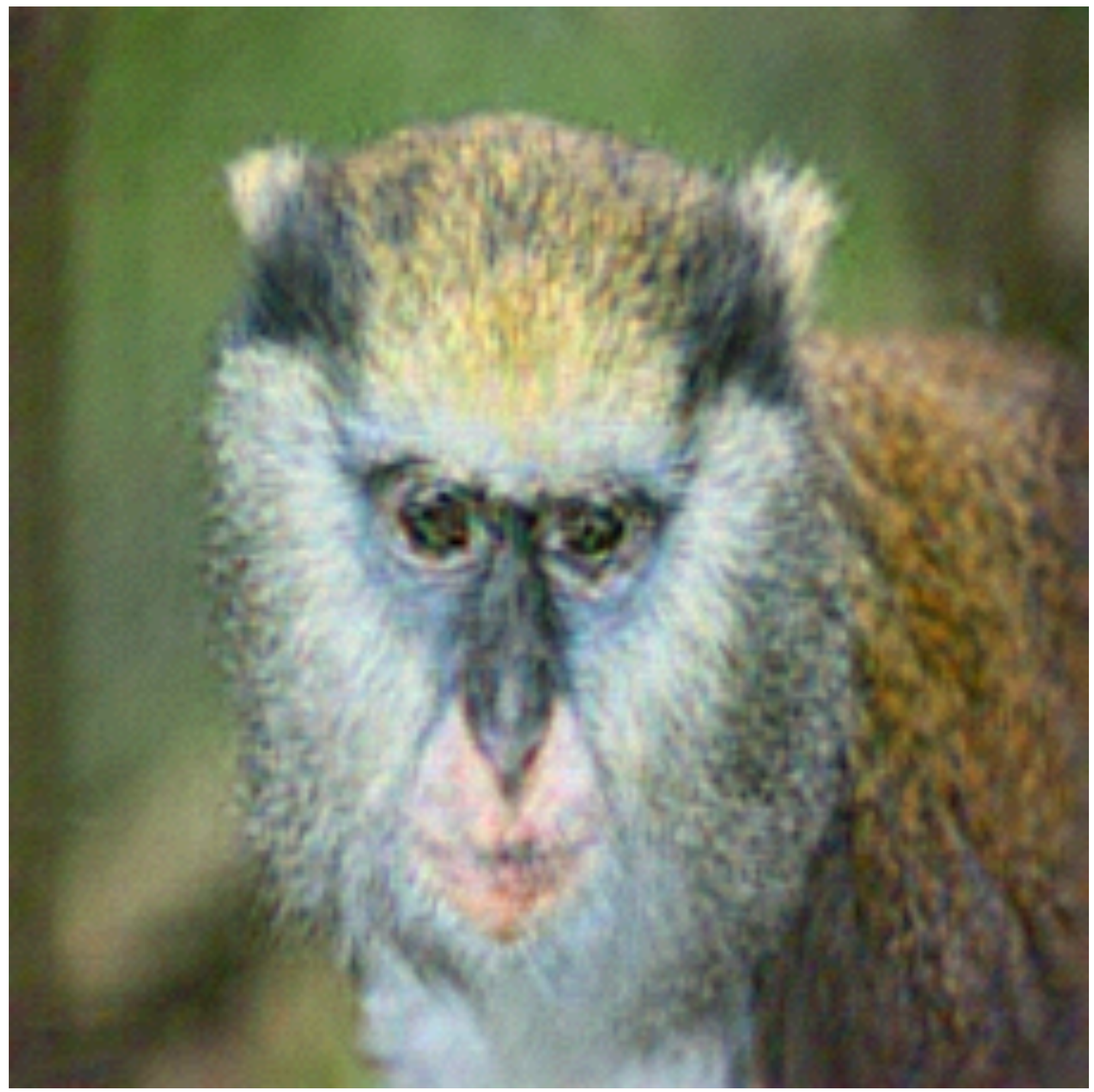
Inversion


Original Image

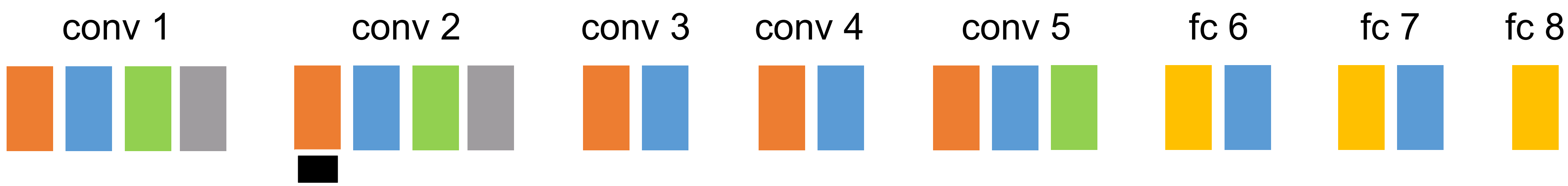
Inversion



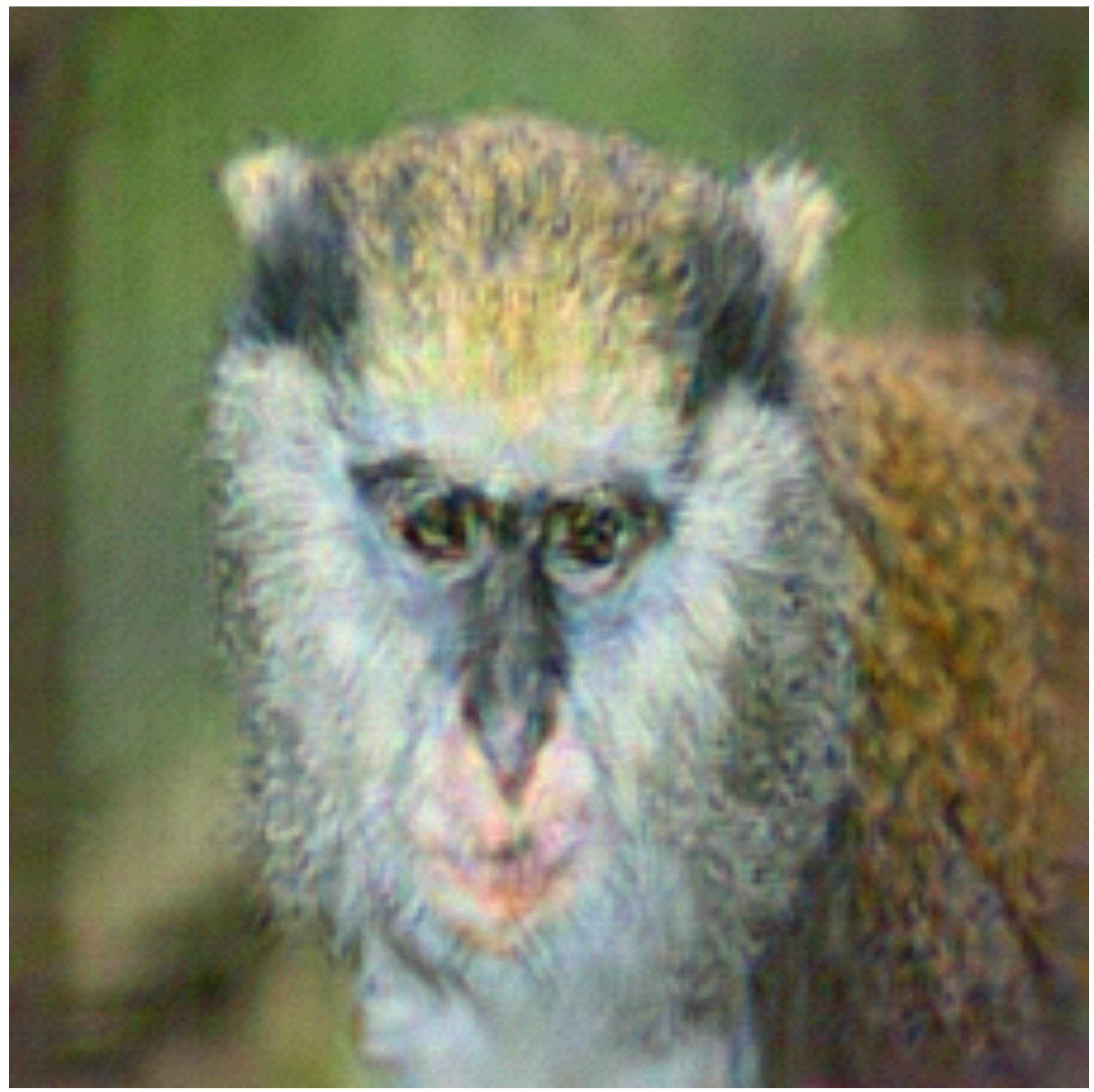
Original Image



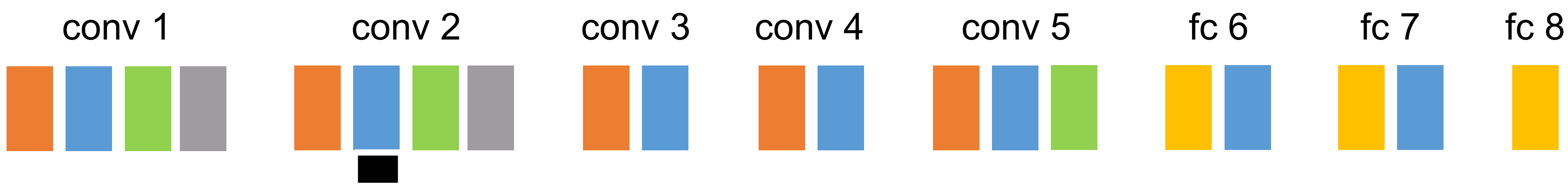
Inversion



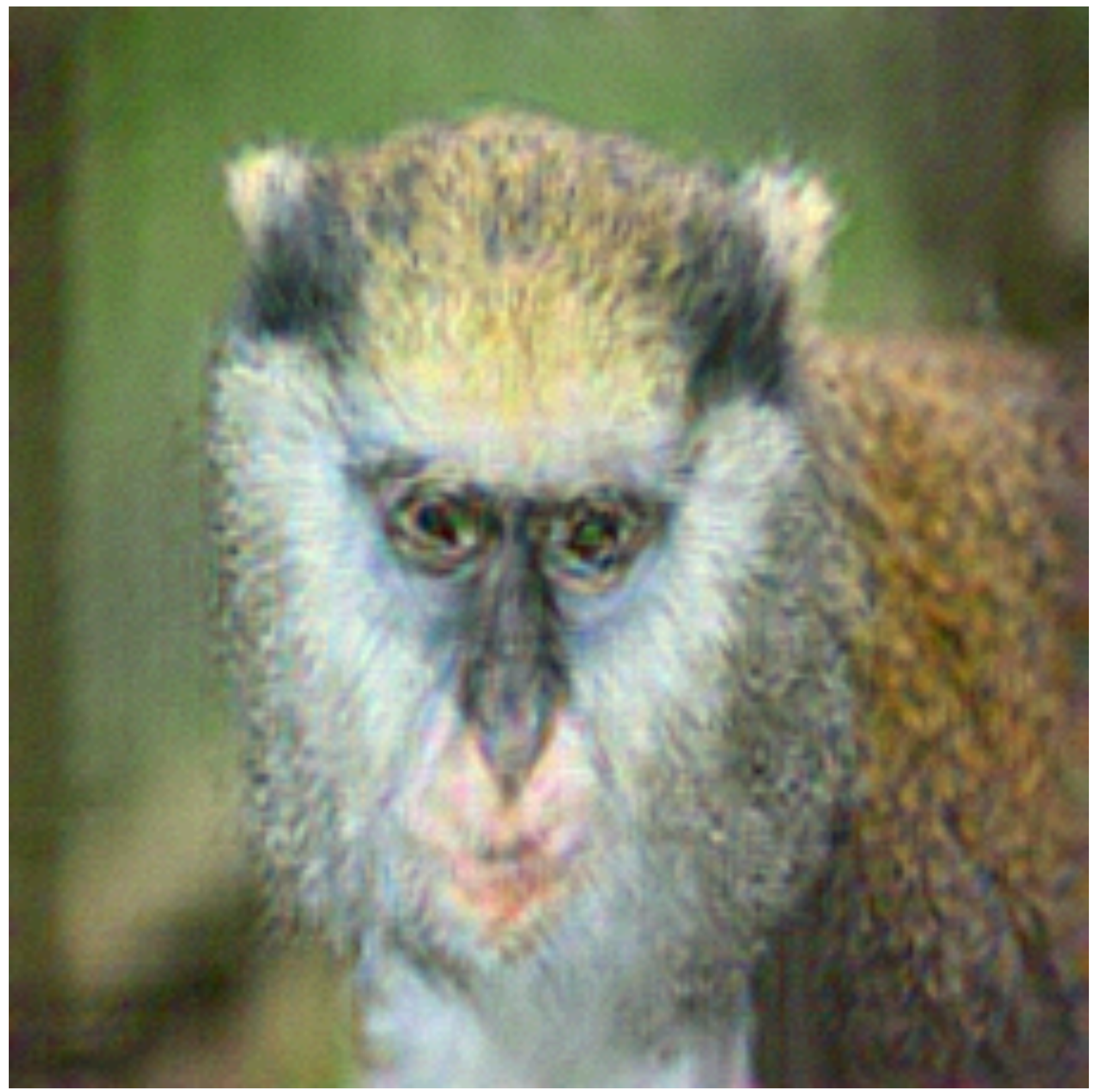
Original Image



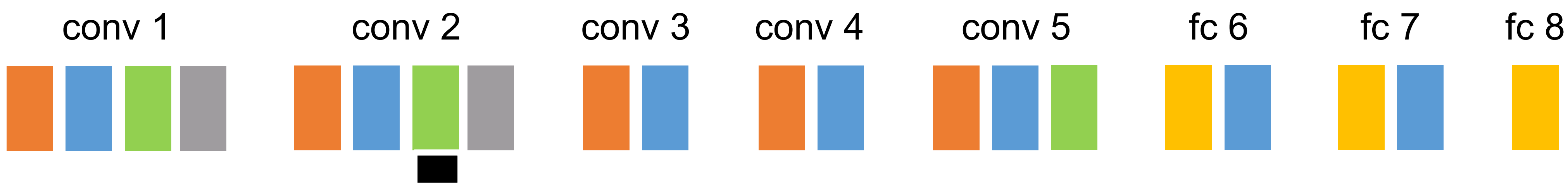
Inversion



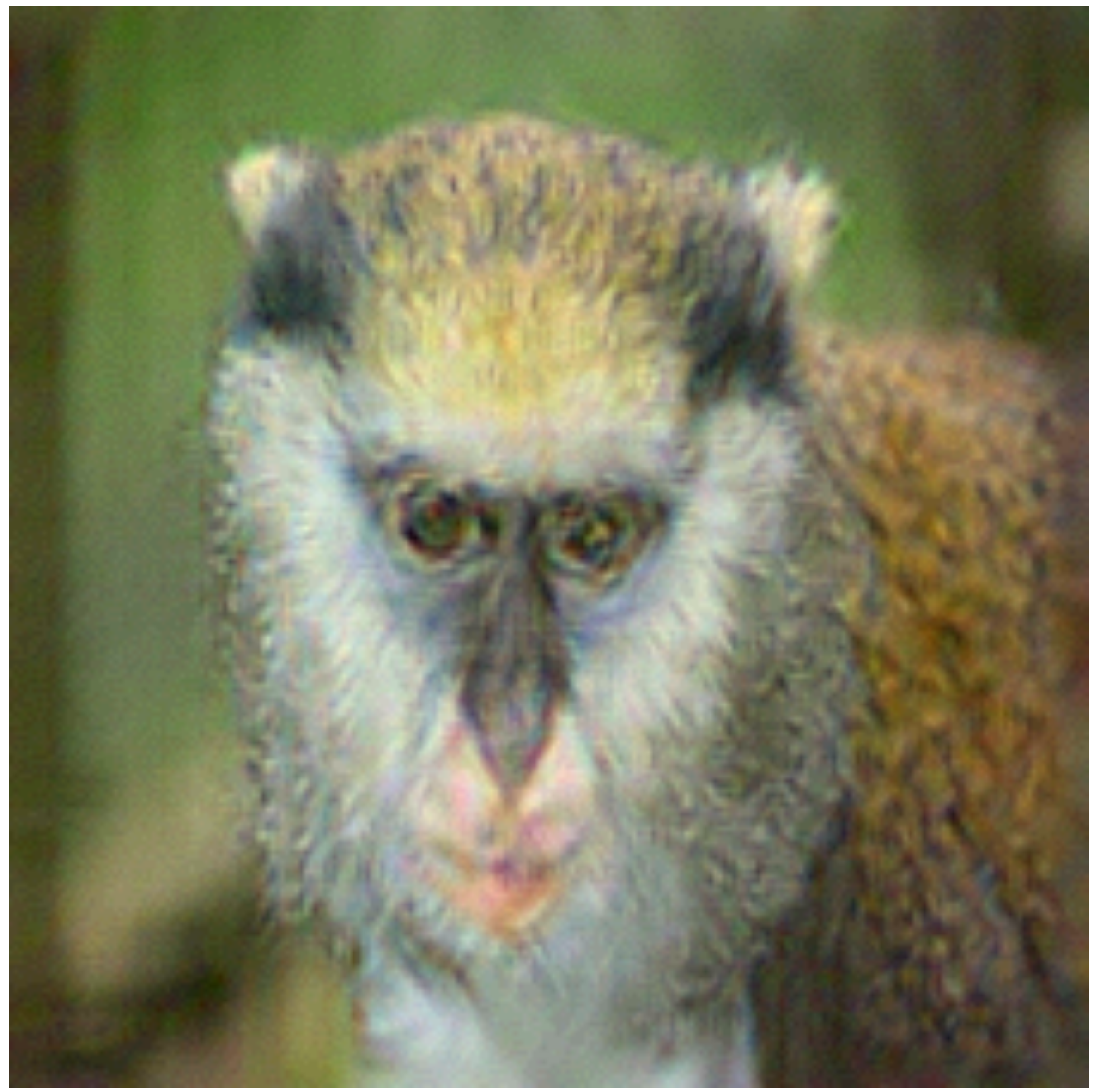
Original
Image



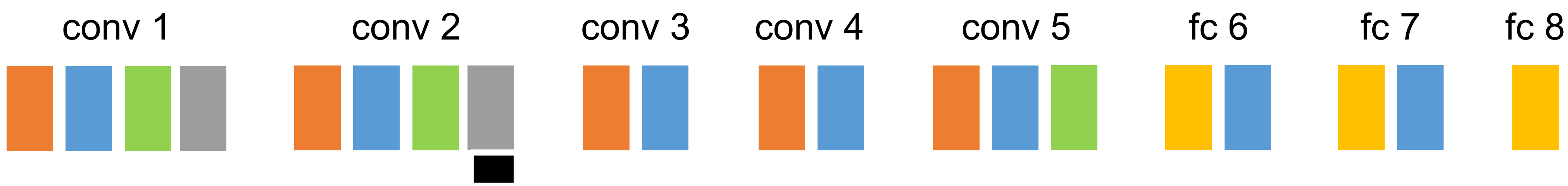
Inversion



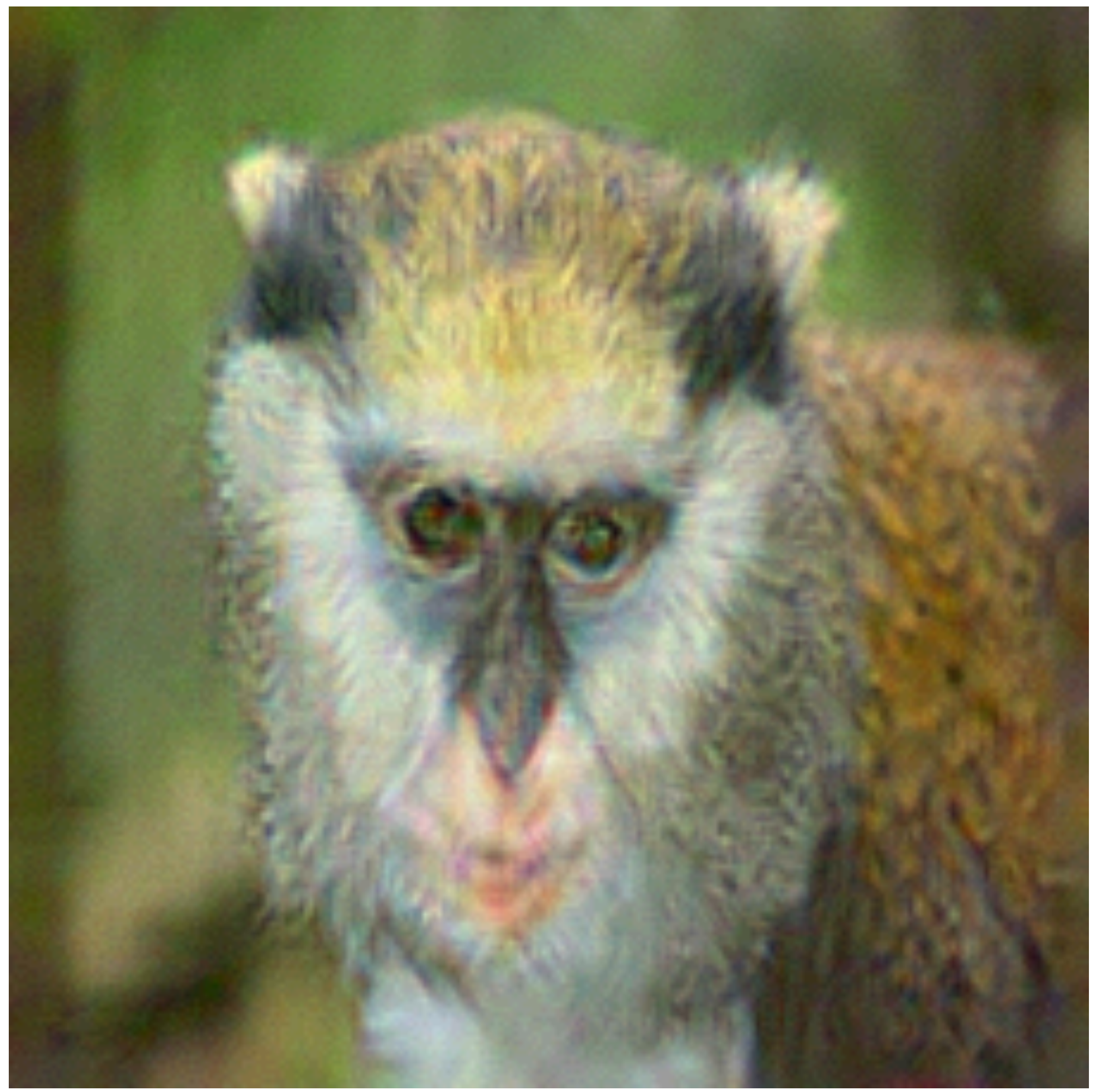
Original Image



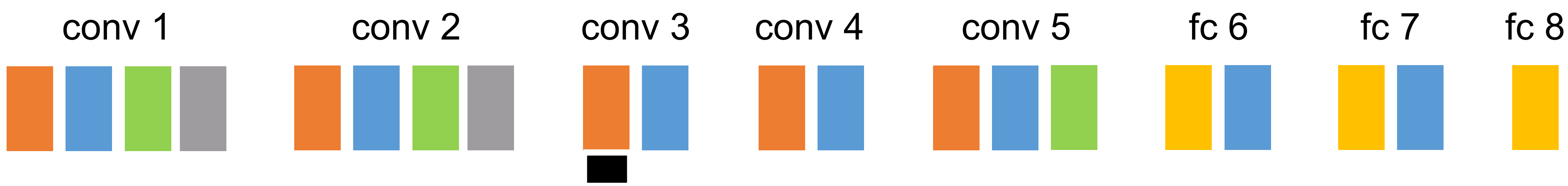
Inversion



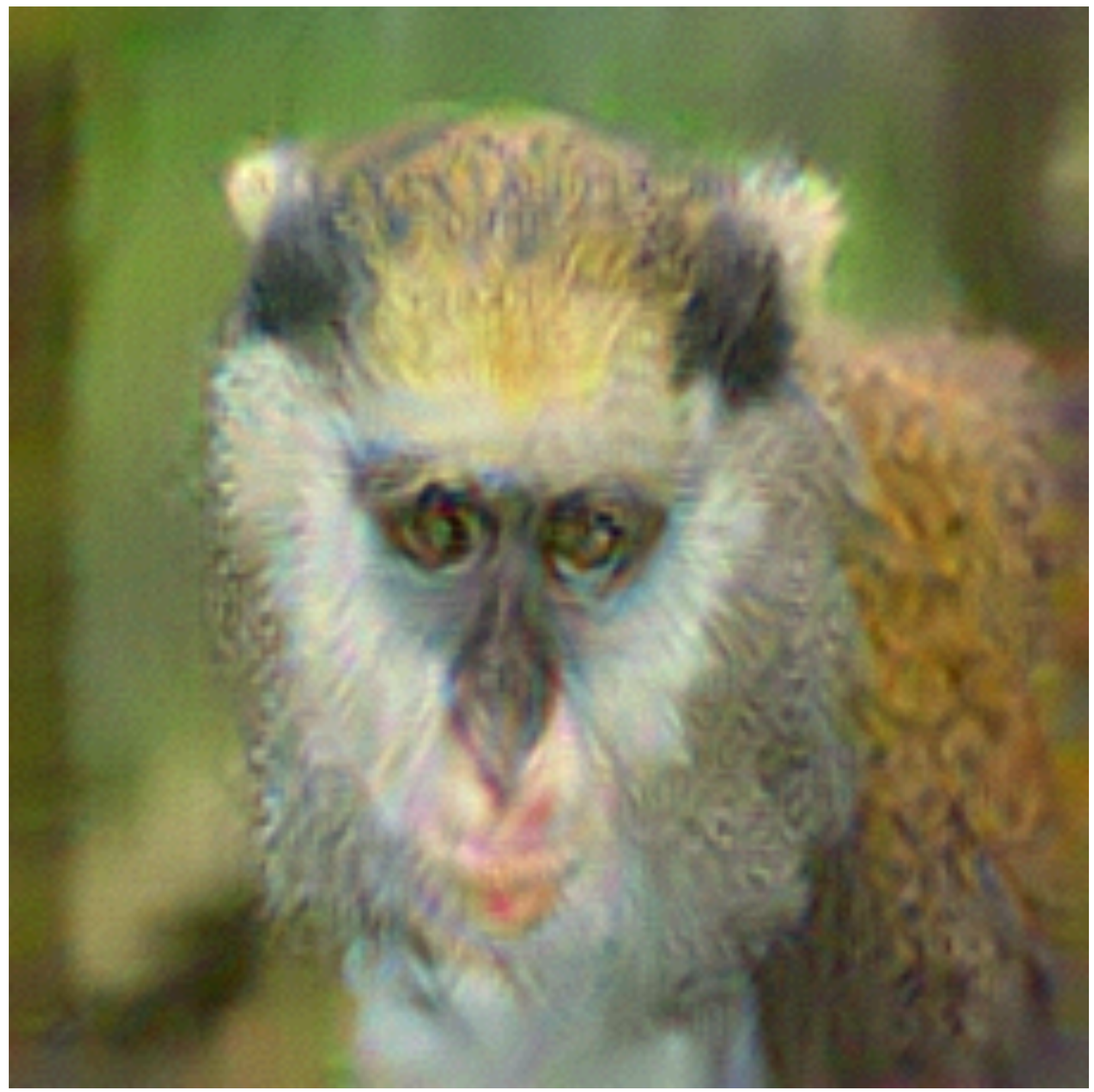
Original Image



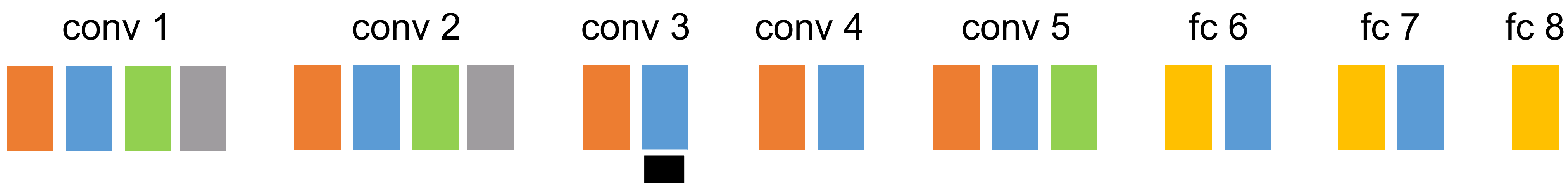
Inversion



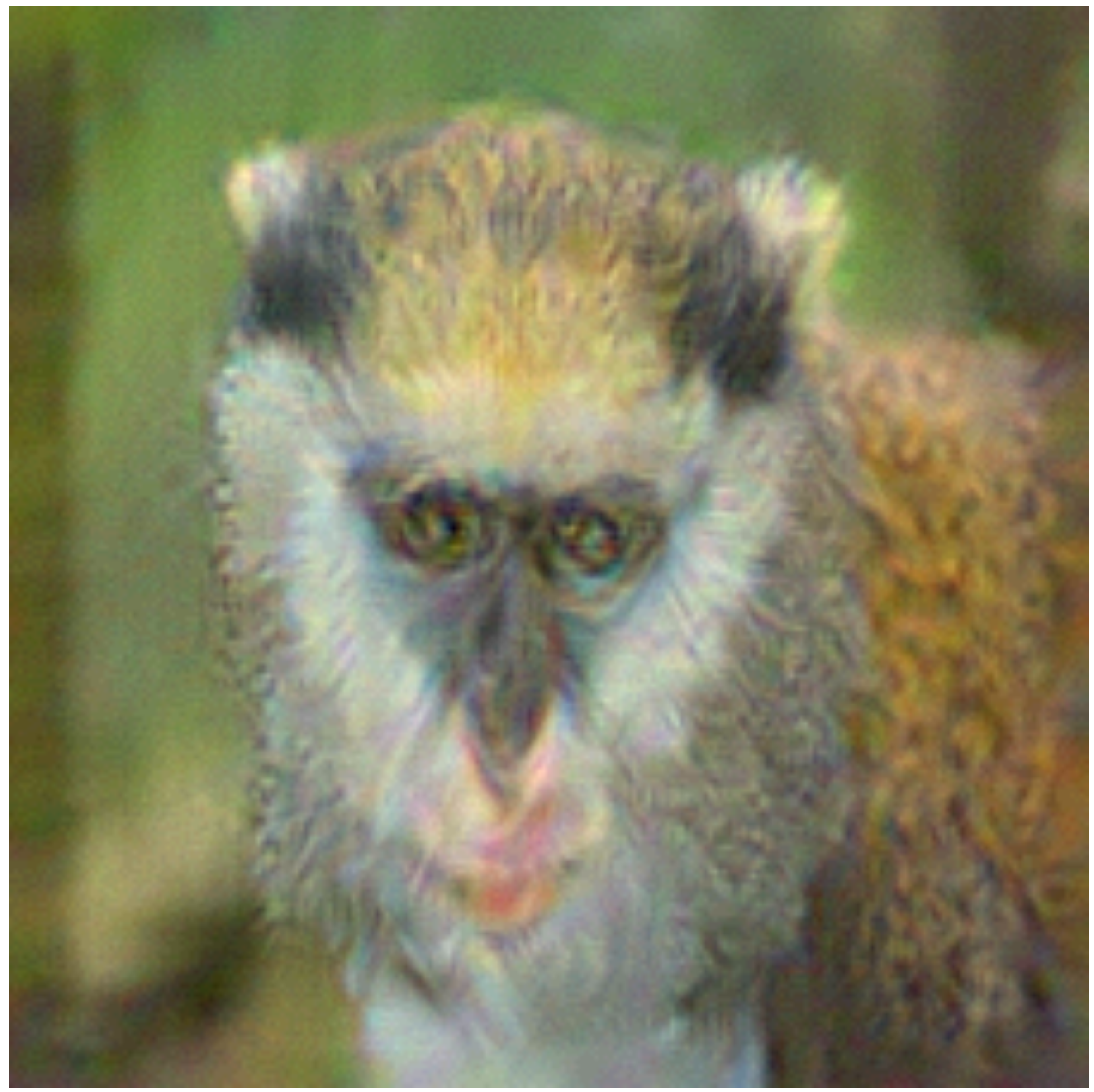

Original Image



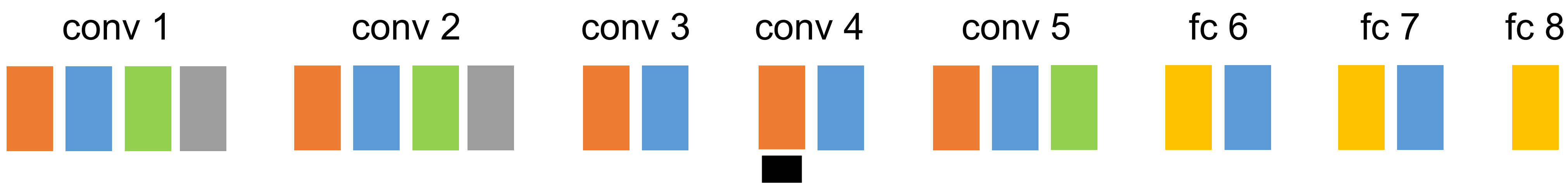
Inversion



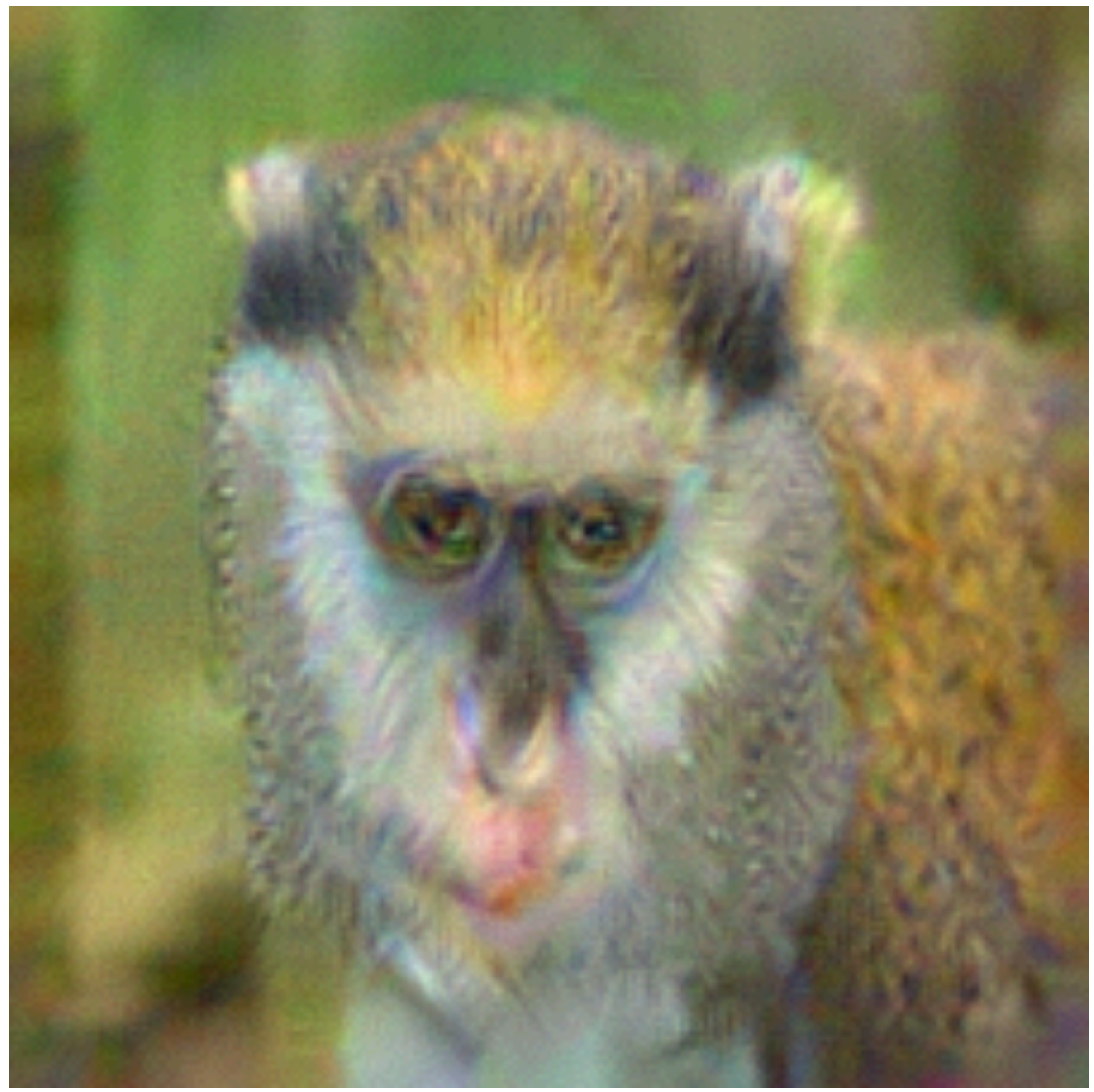
Original Image



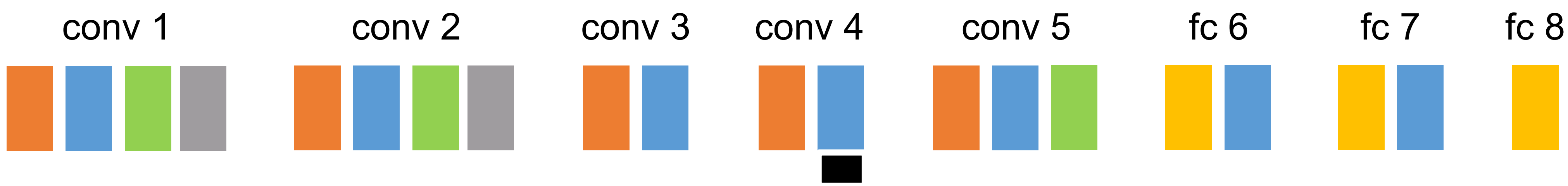
Inversion



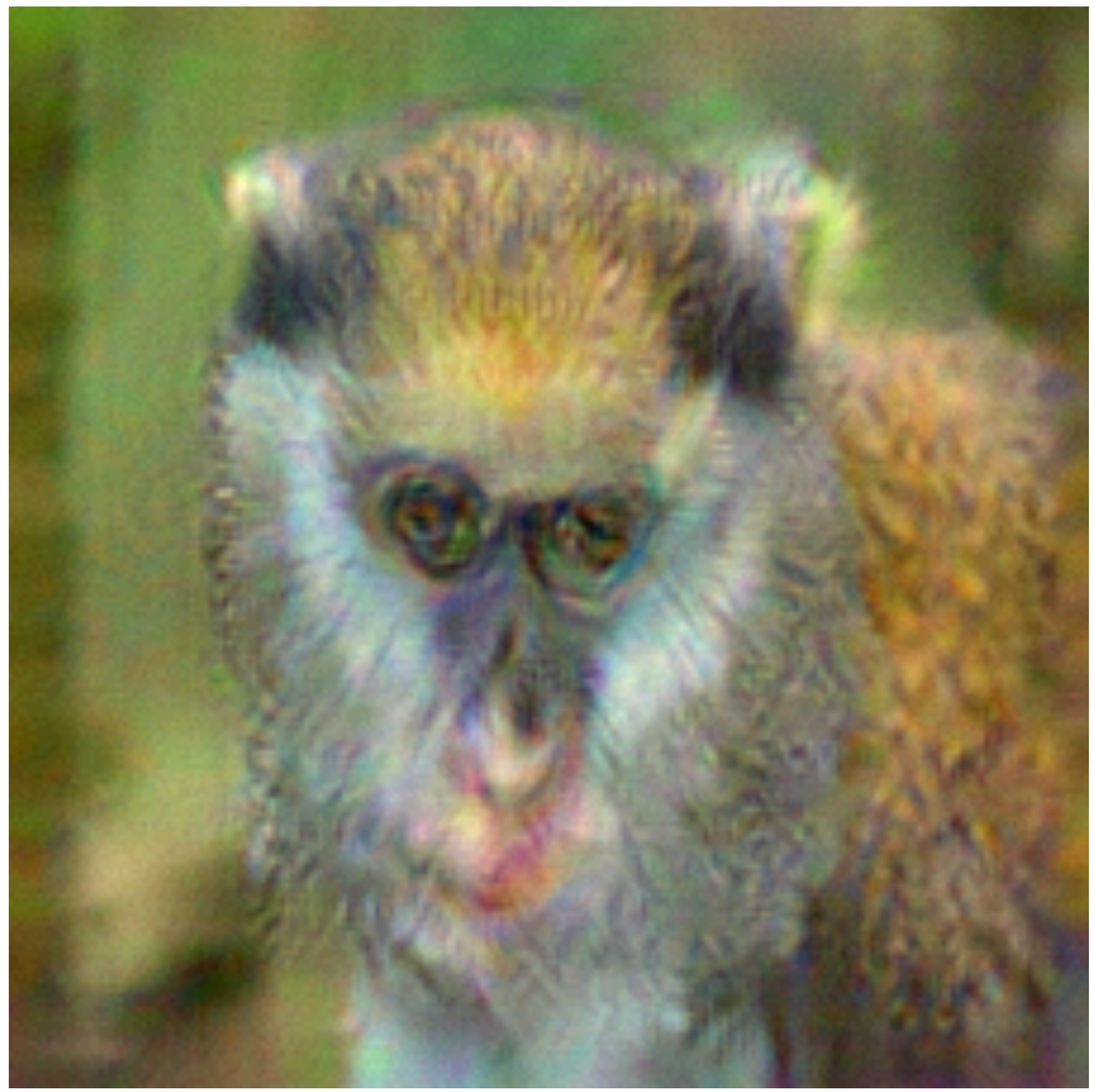

Original Image



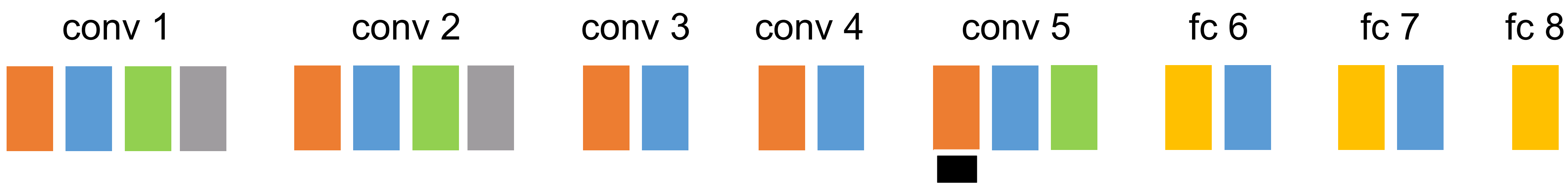
Inversion



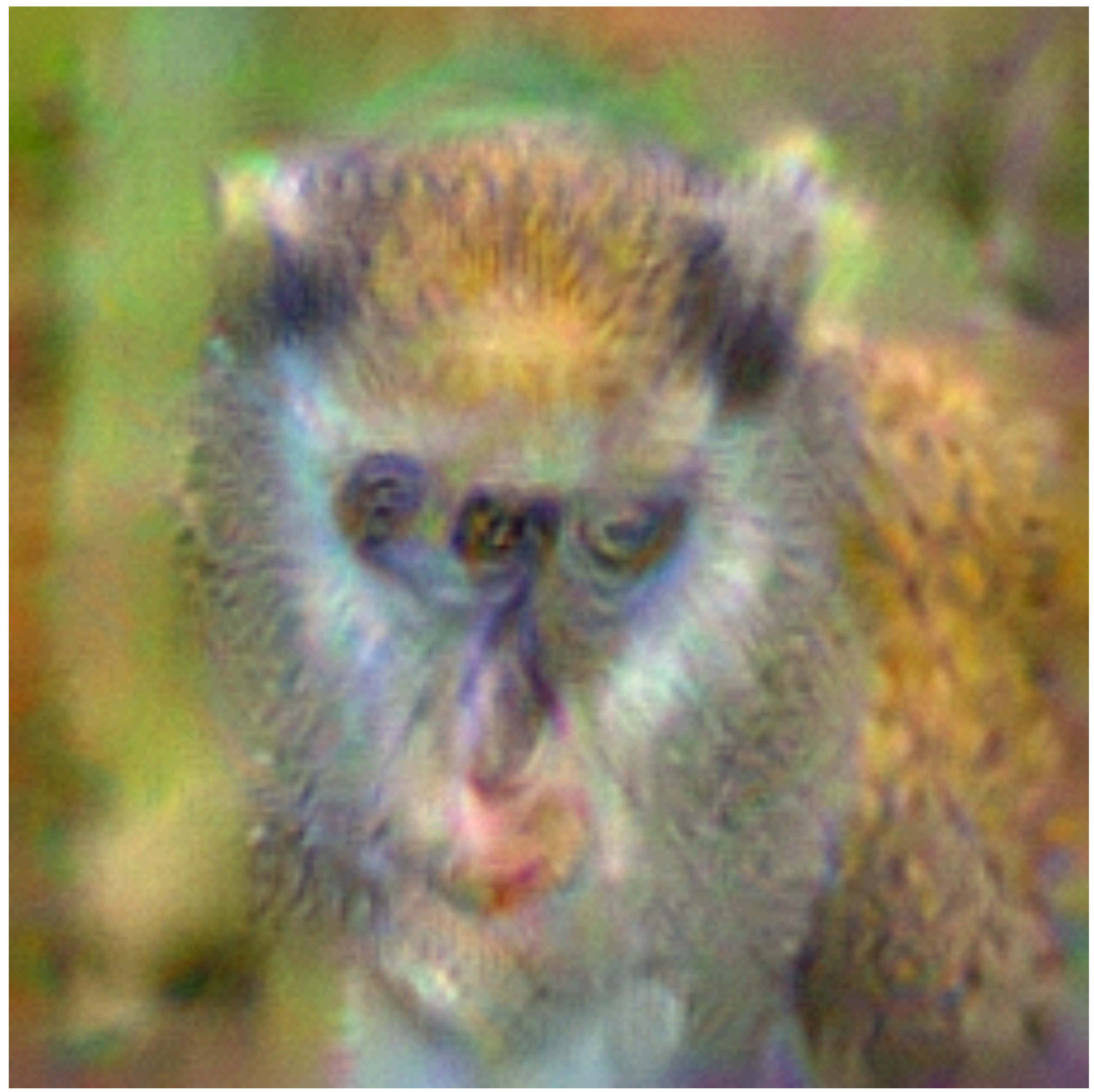
Original Image



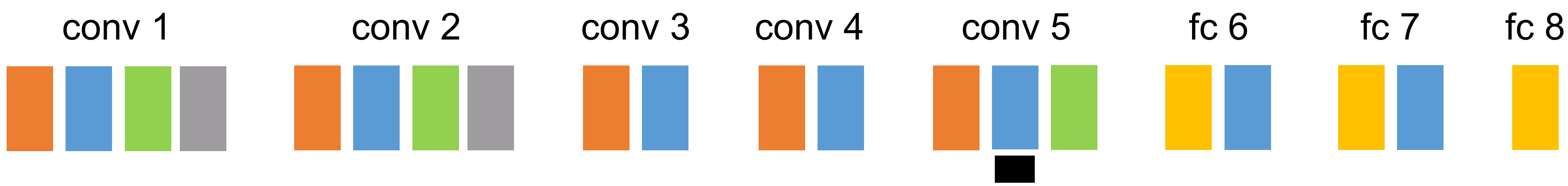
Inversion



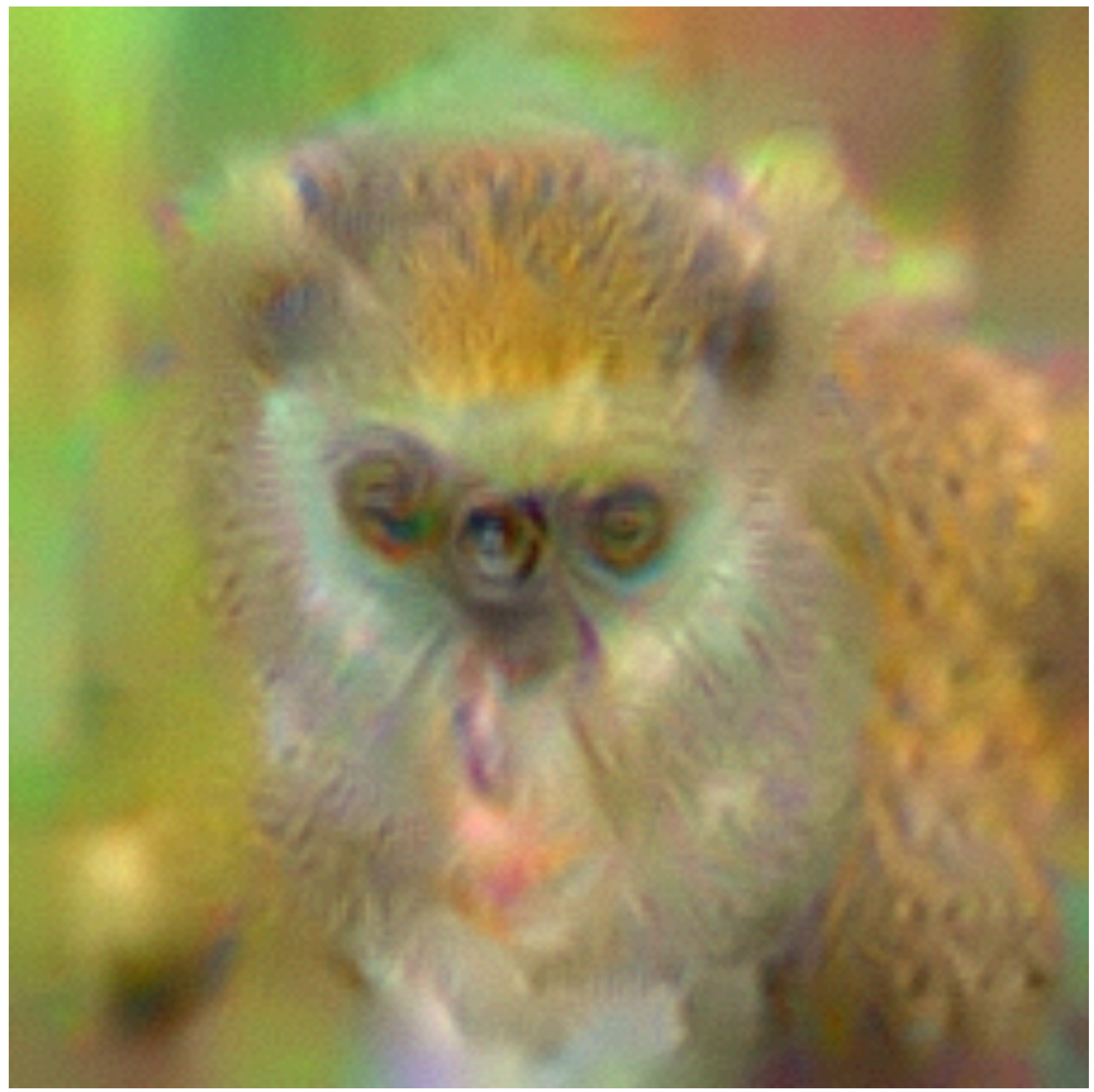
Original Image



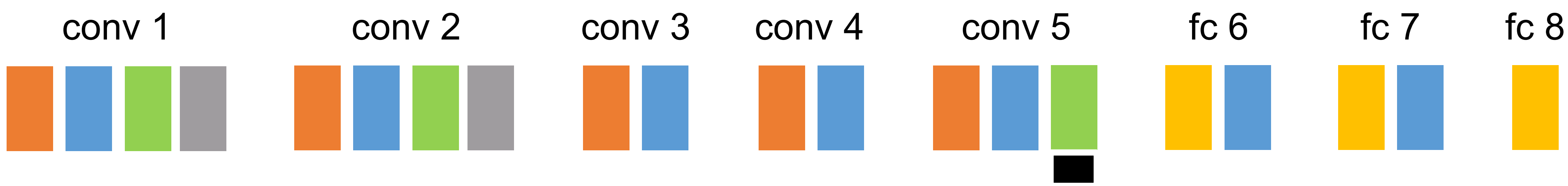
Inversion



Original Image

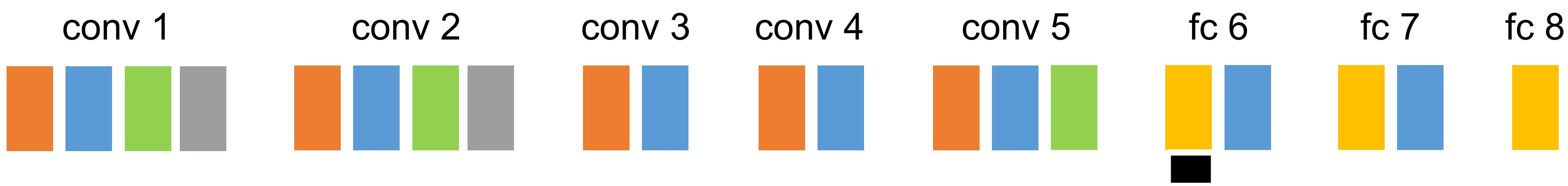


Inversion

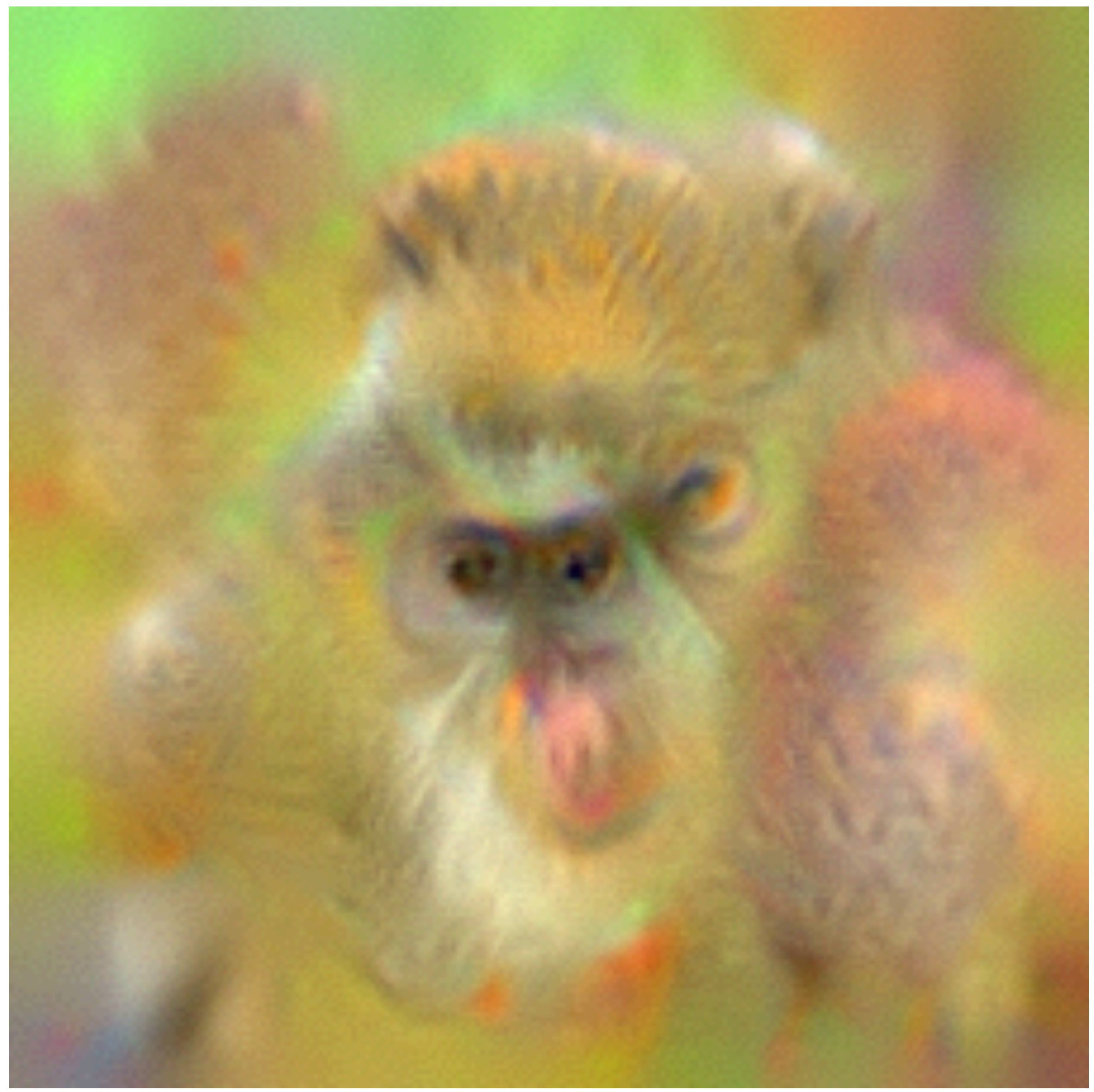


Original Image

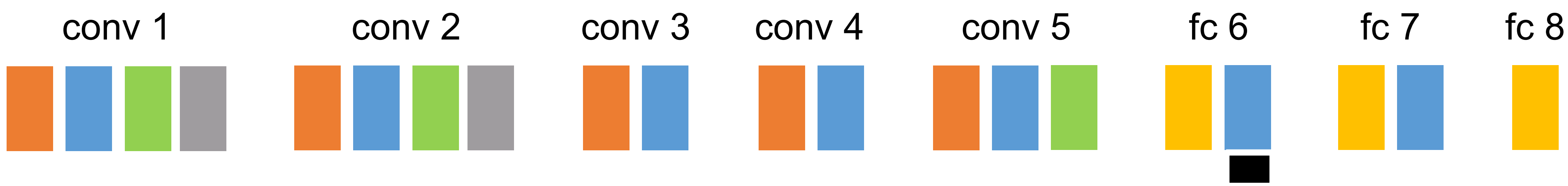
Inversion



Original Image

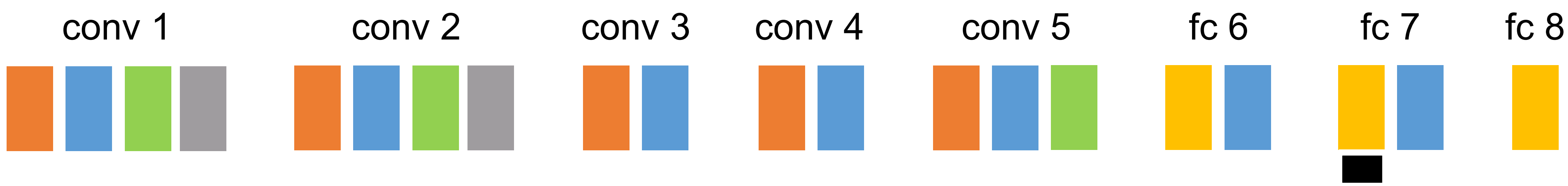


Inversion

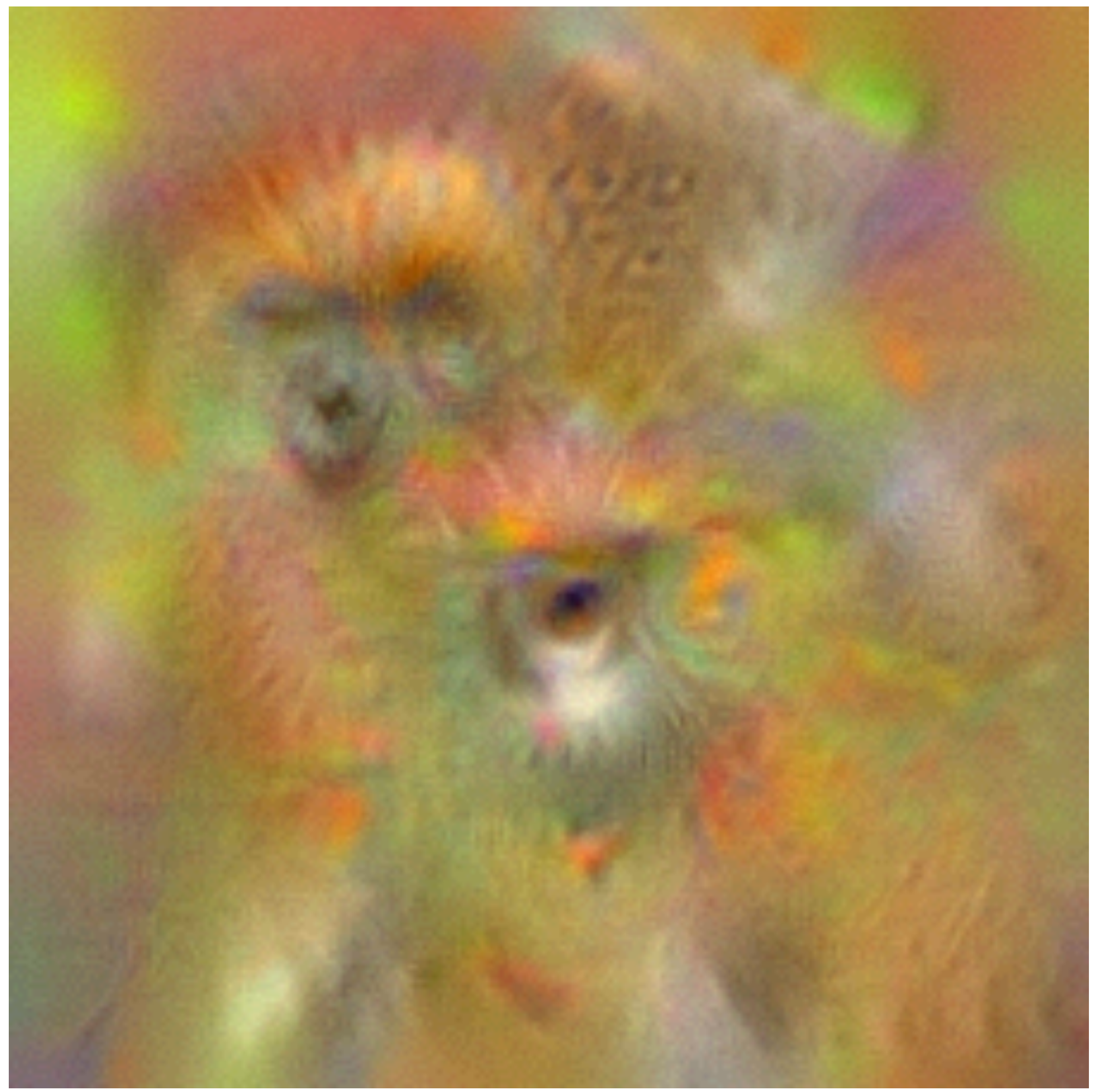


Original Image

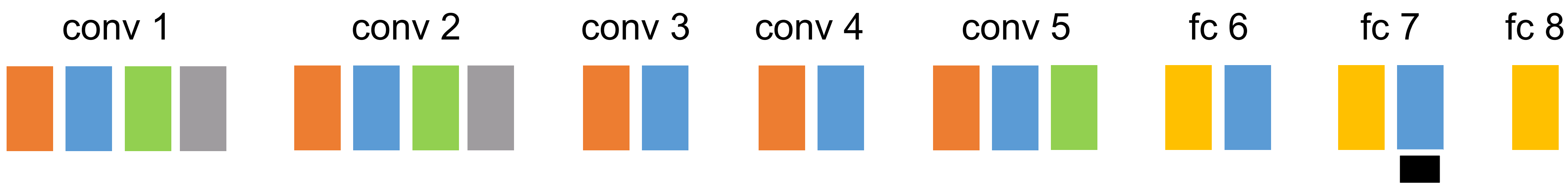
Inversion



Original Image

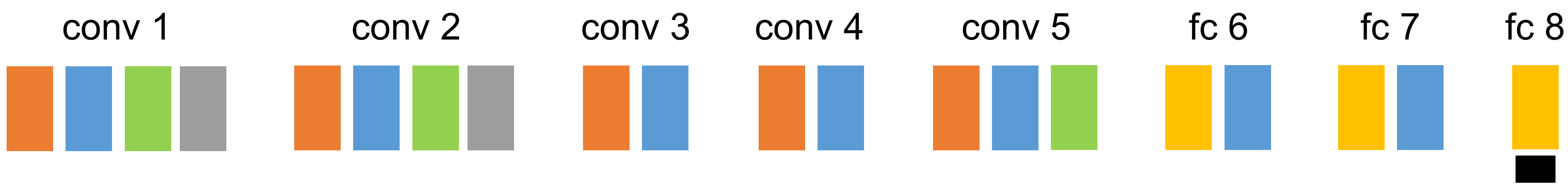


Inversion



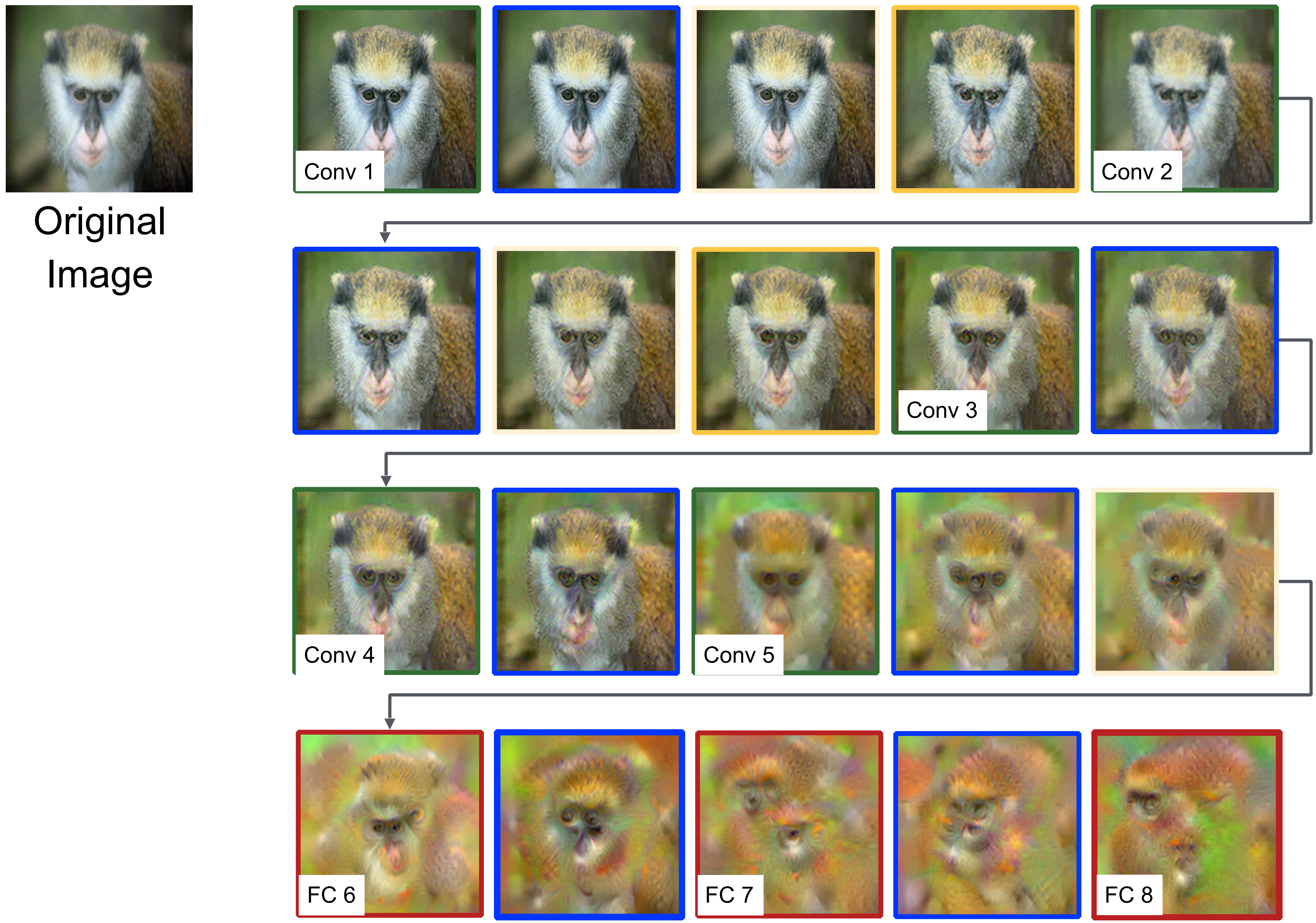
Original Image

Inversion



Original Image

Inverting a Deep CNN

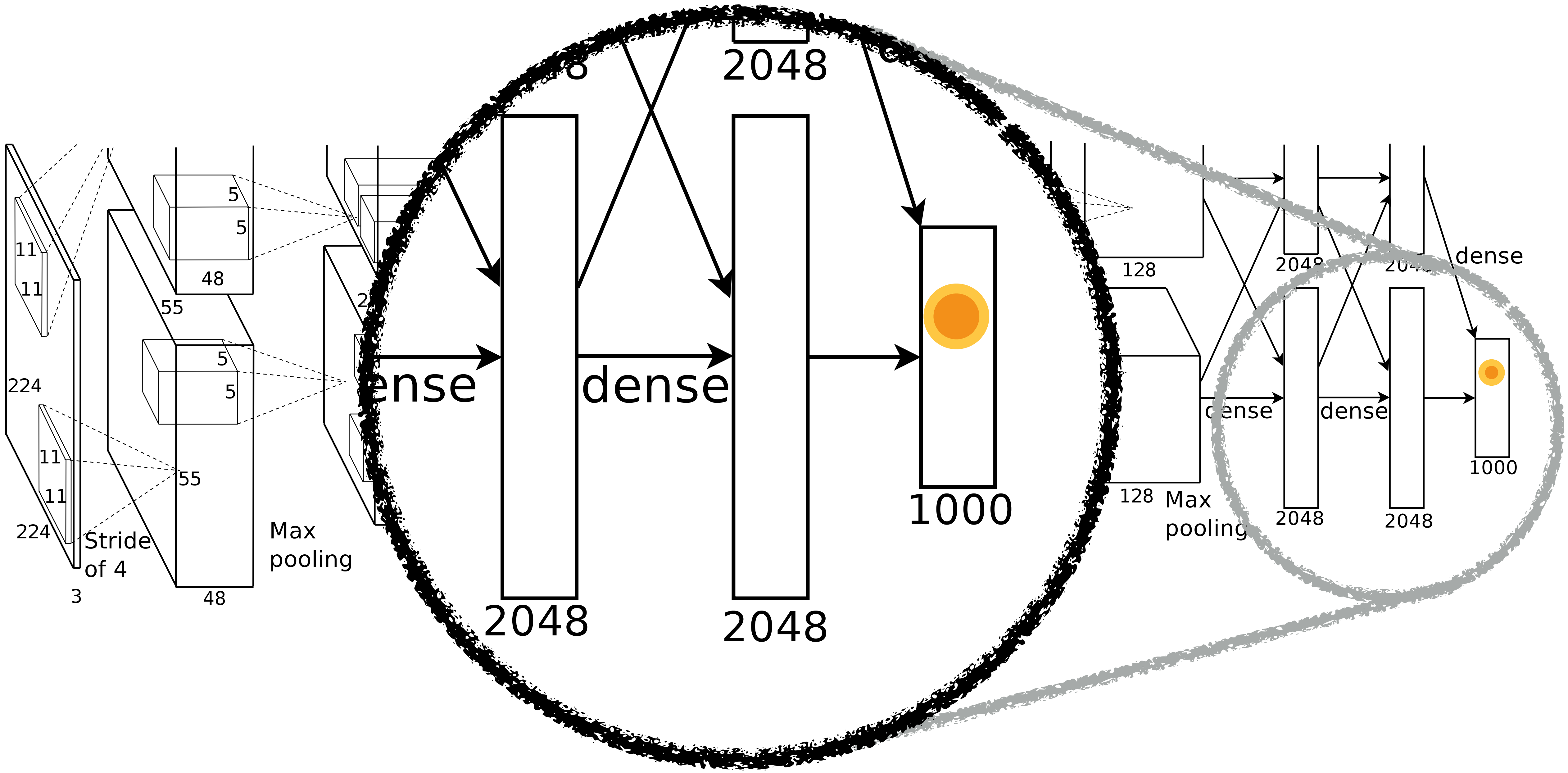


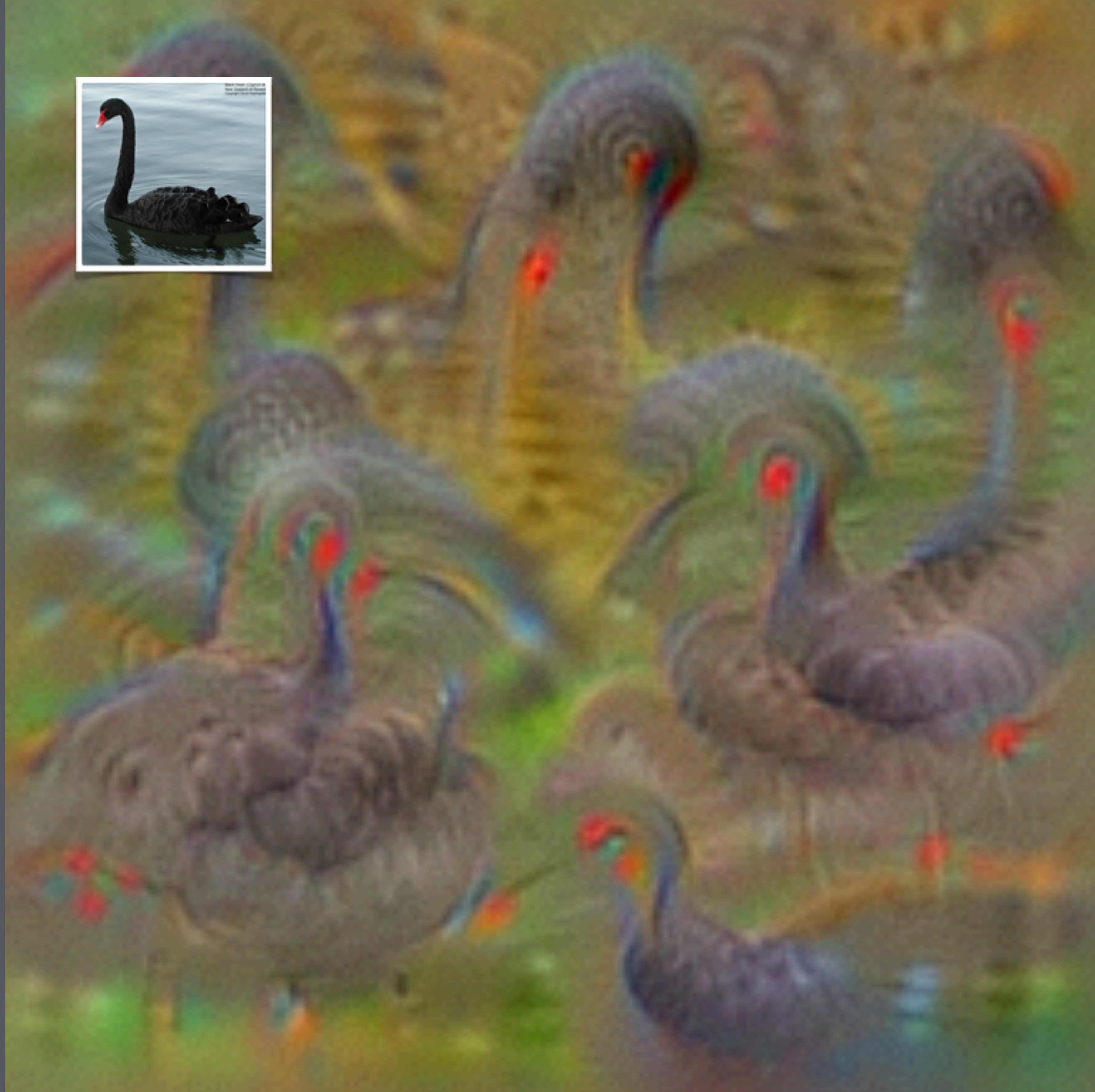
CNNs = visual codes?

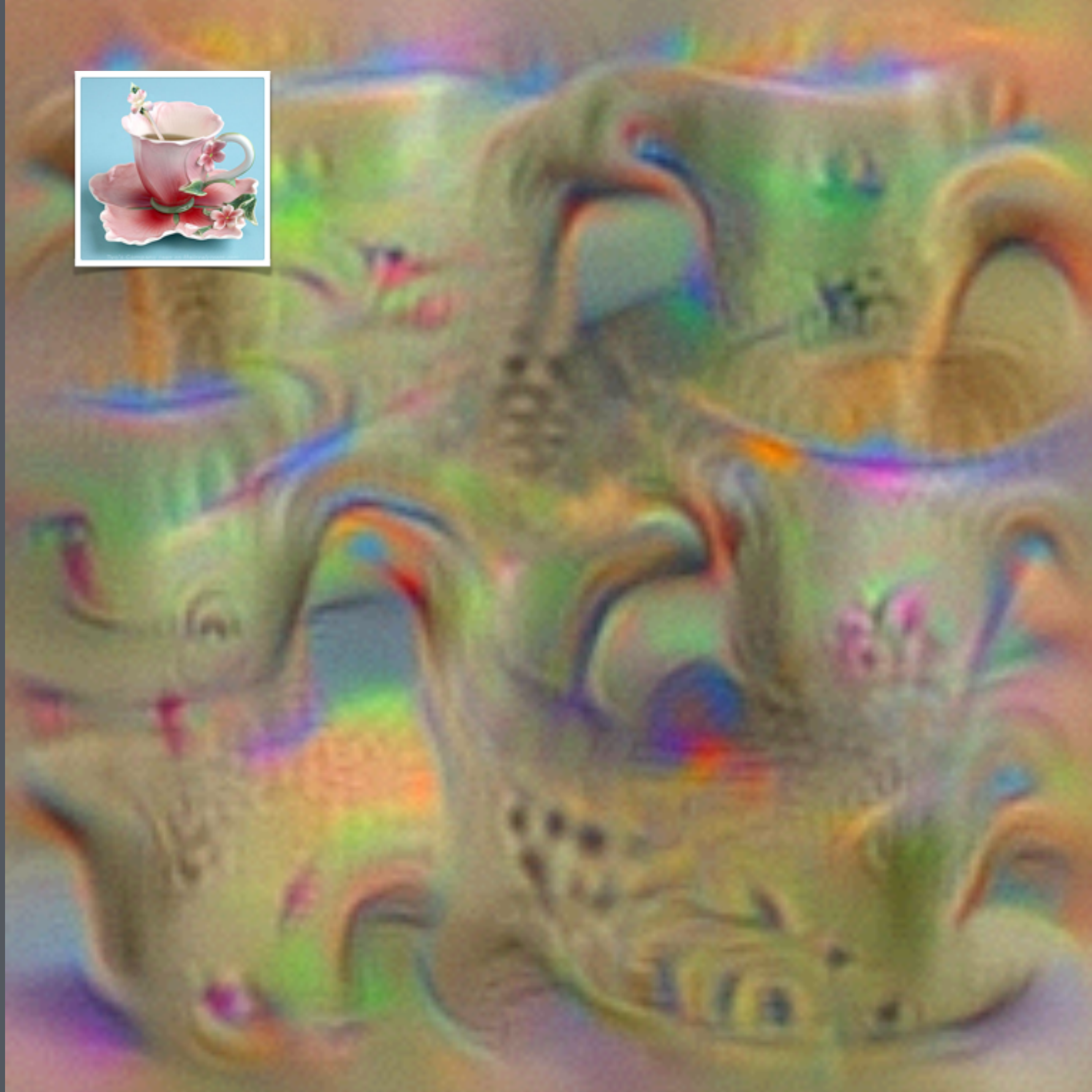
Activation Maximization

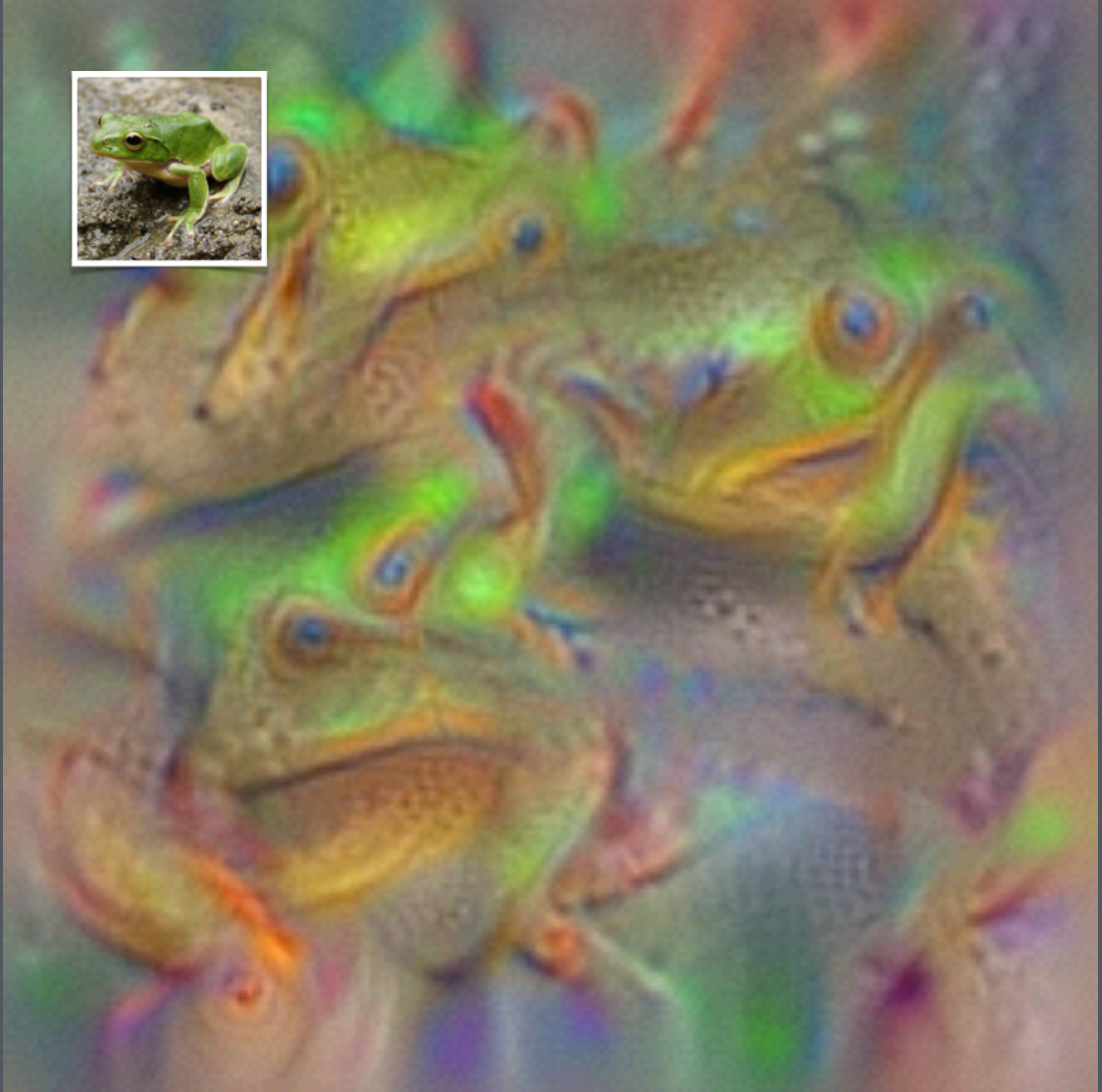
Look for an image that maximally activates a **specific feature component**

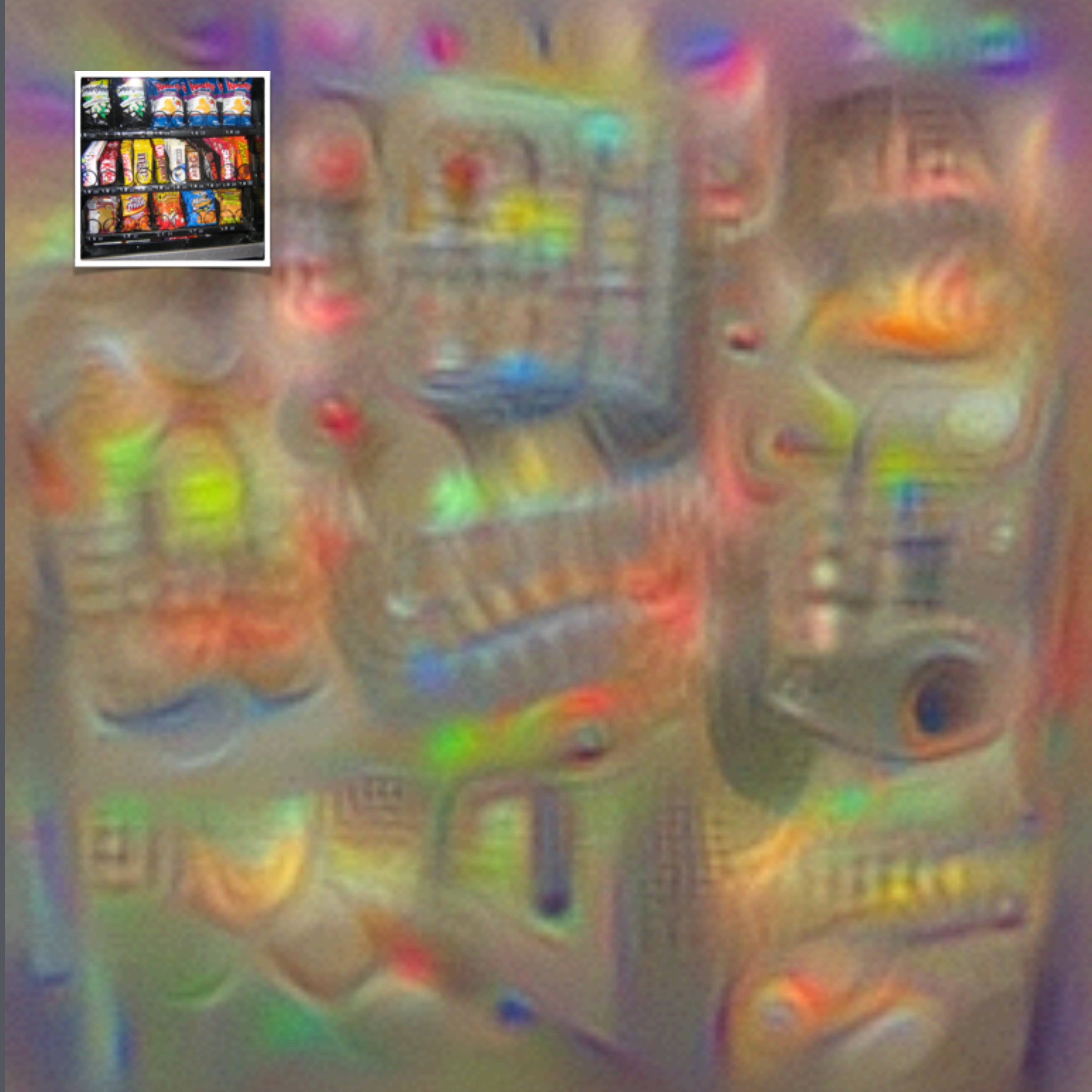
$$\min_{\mathbf{x}} -\langle \mathbf{e}_k, \Phi(\mathbf{x})\rangle + R_{TV}(\mathbf{x}) + R_\alpha(\mathbf{x})$$





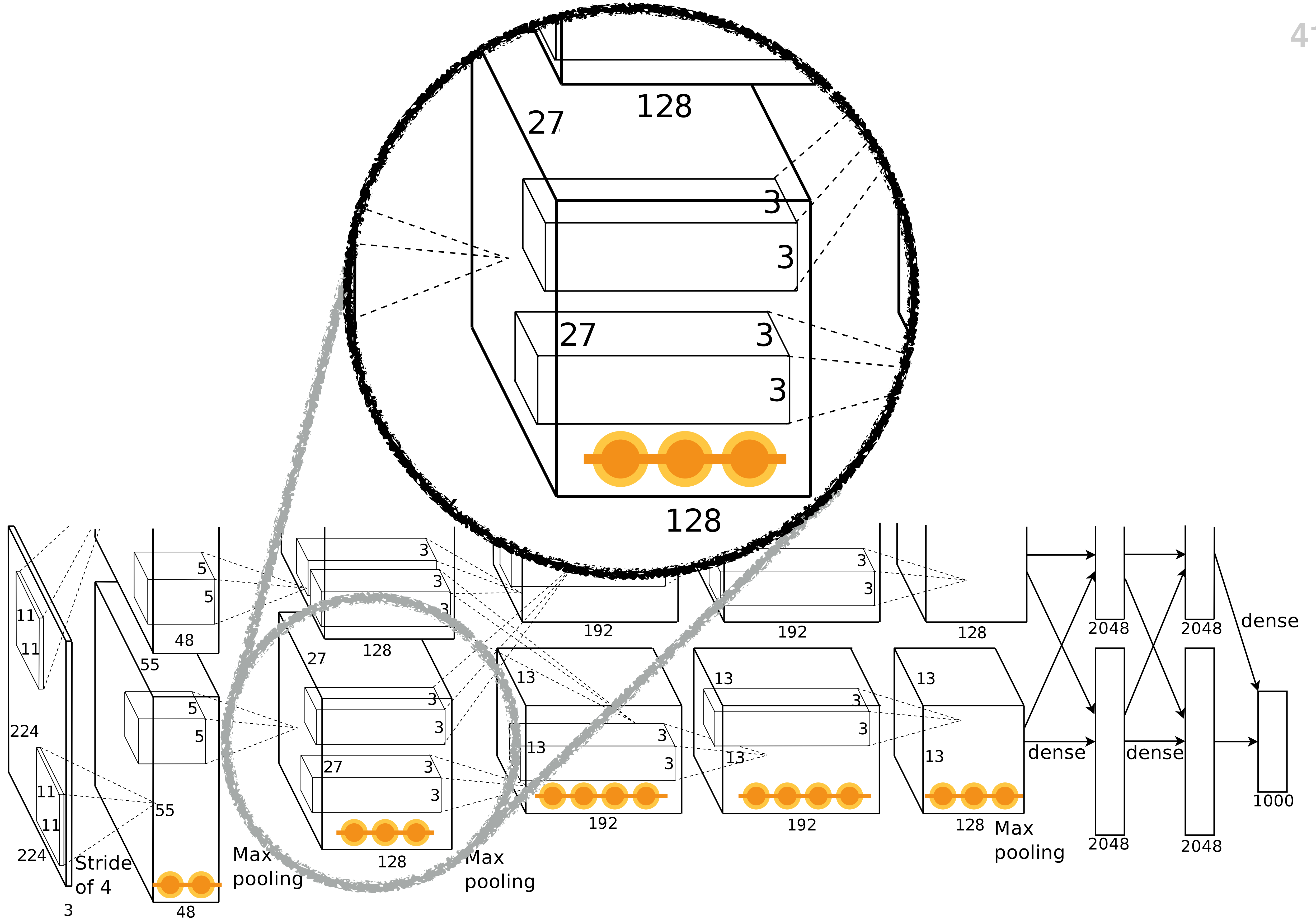






Remember: the starting point is white noise

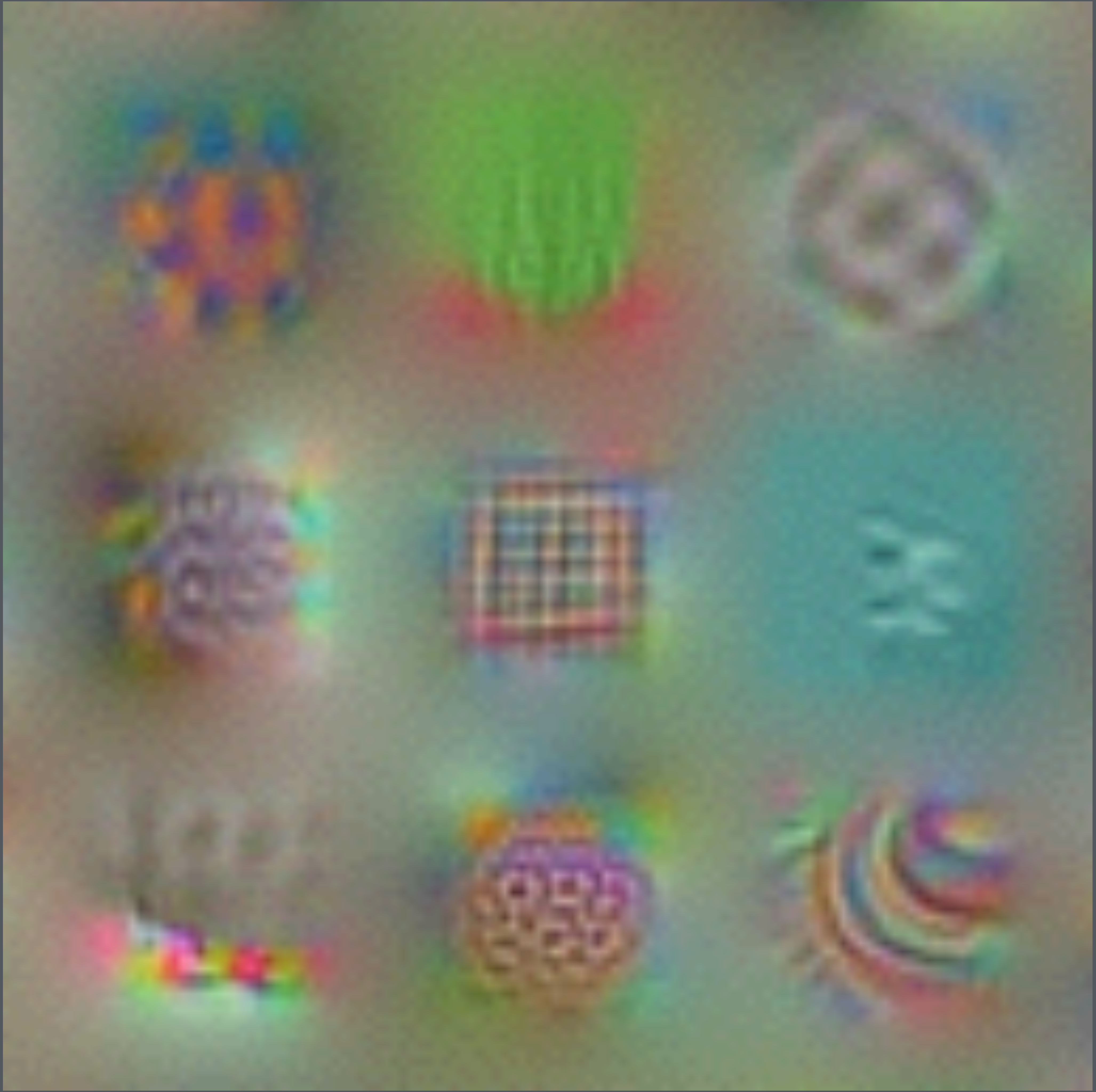
Not an image!



128
27
3
3
27
3
3
128
11
11
224
224
3
Stride of 4
5
5
48
55
5
5
55
48
Max pooling
27
128
27
3
3
27
3
3
128
Max pooling
3
3
3
13
192
13
3
3
13
192
3
3
192
13
3
3
13
192
128
13
13
128
Max pooling
dense
dense
2048
2048
2048
2048
dense
1000

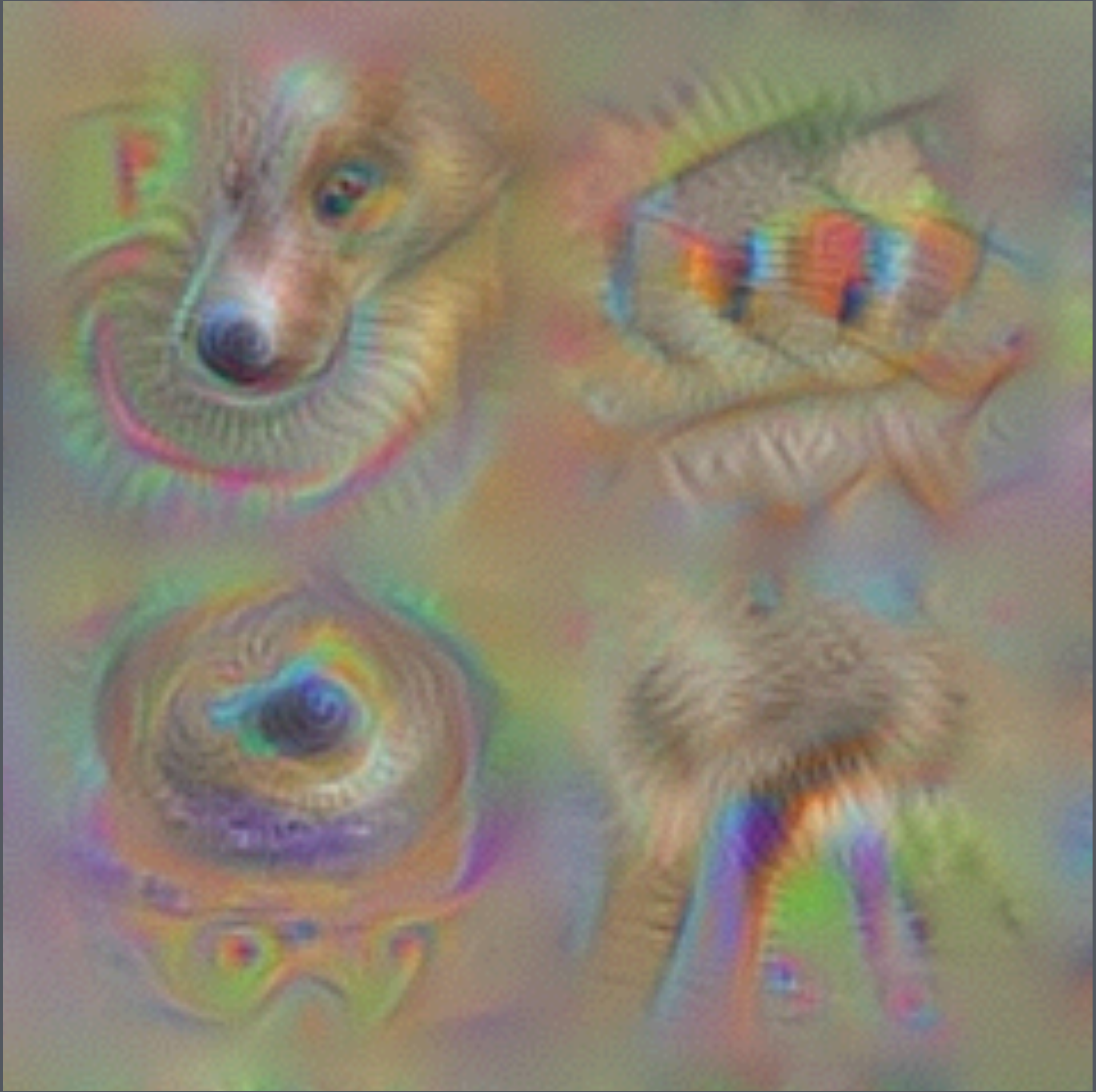
conv1

conv2

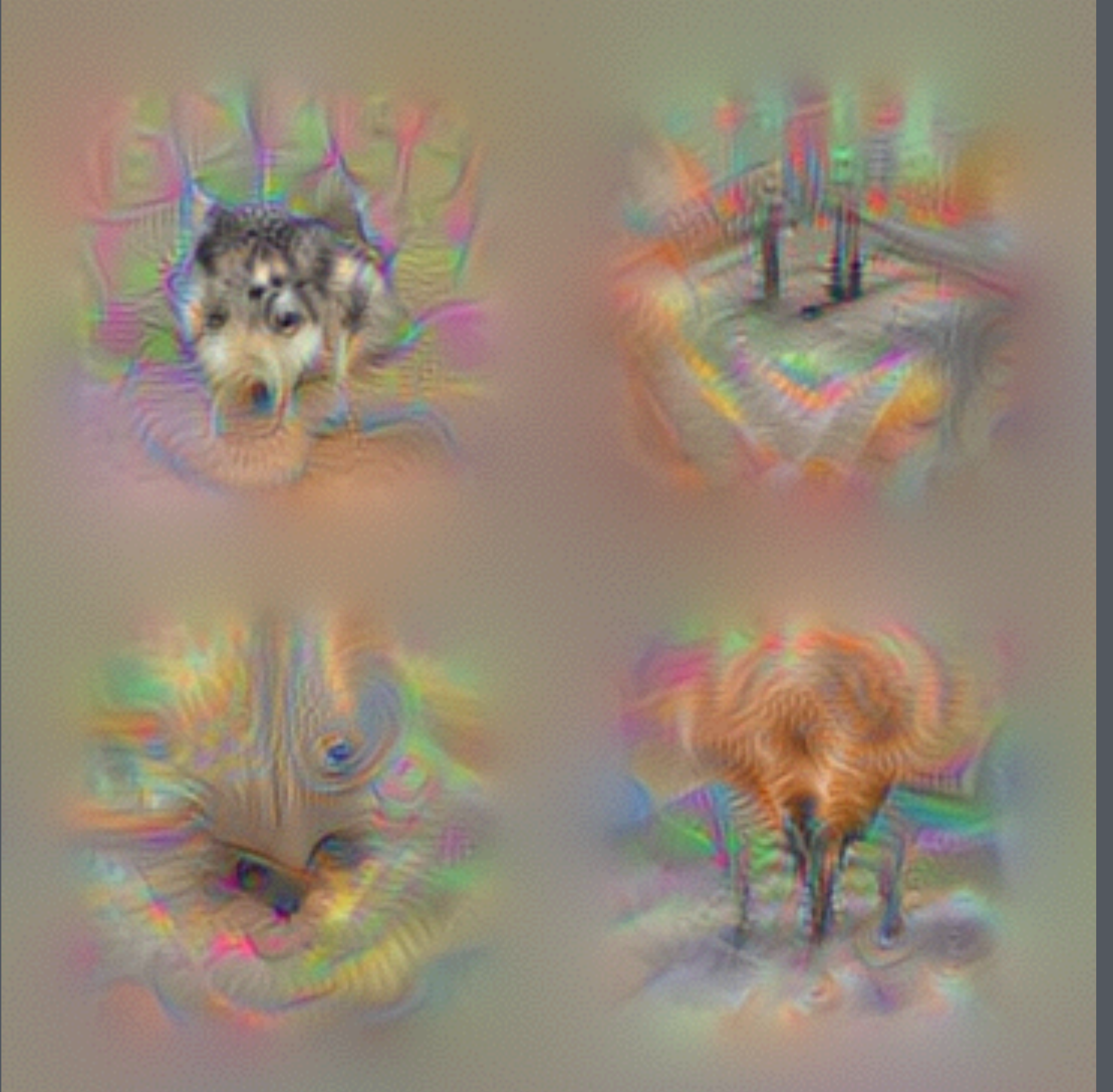


conv3

conv4



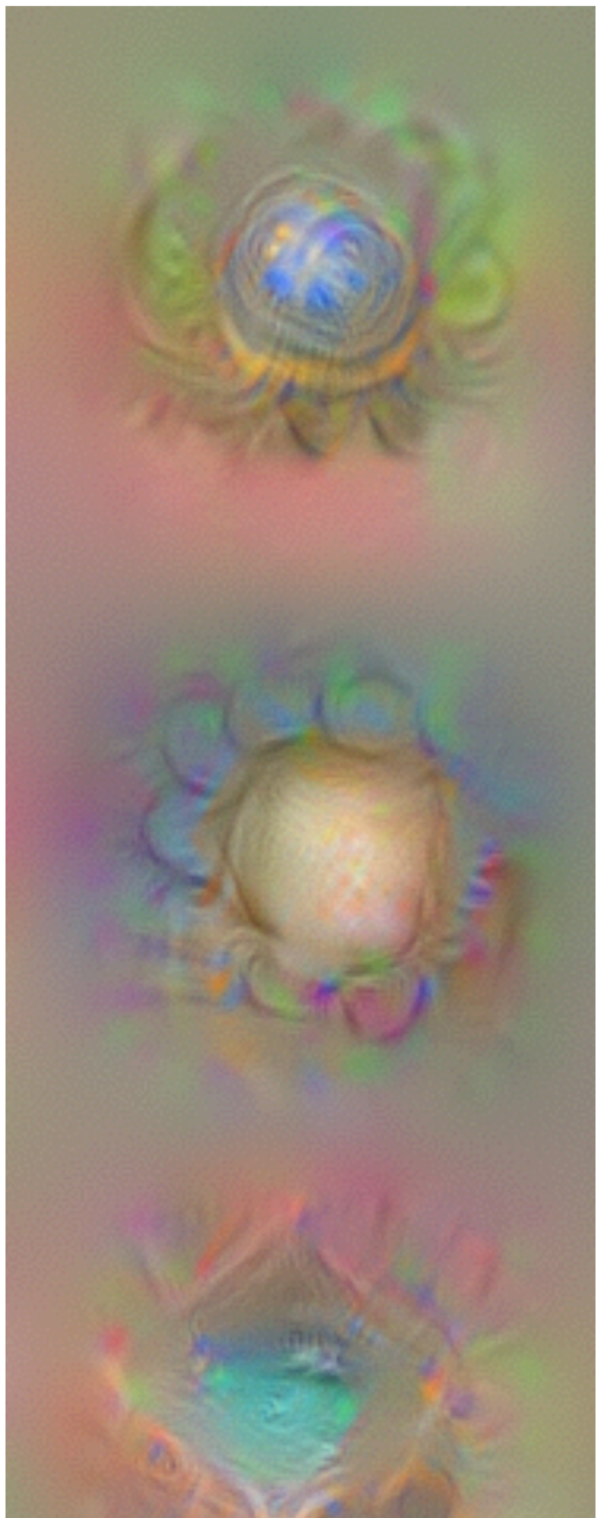
conv5



Network comparison

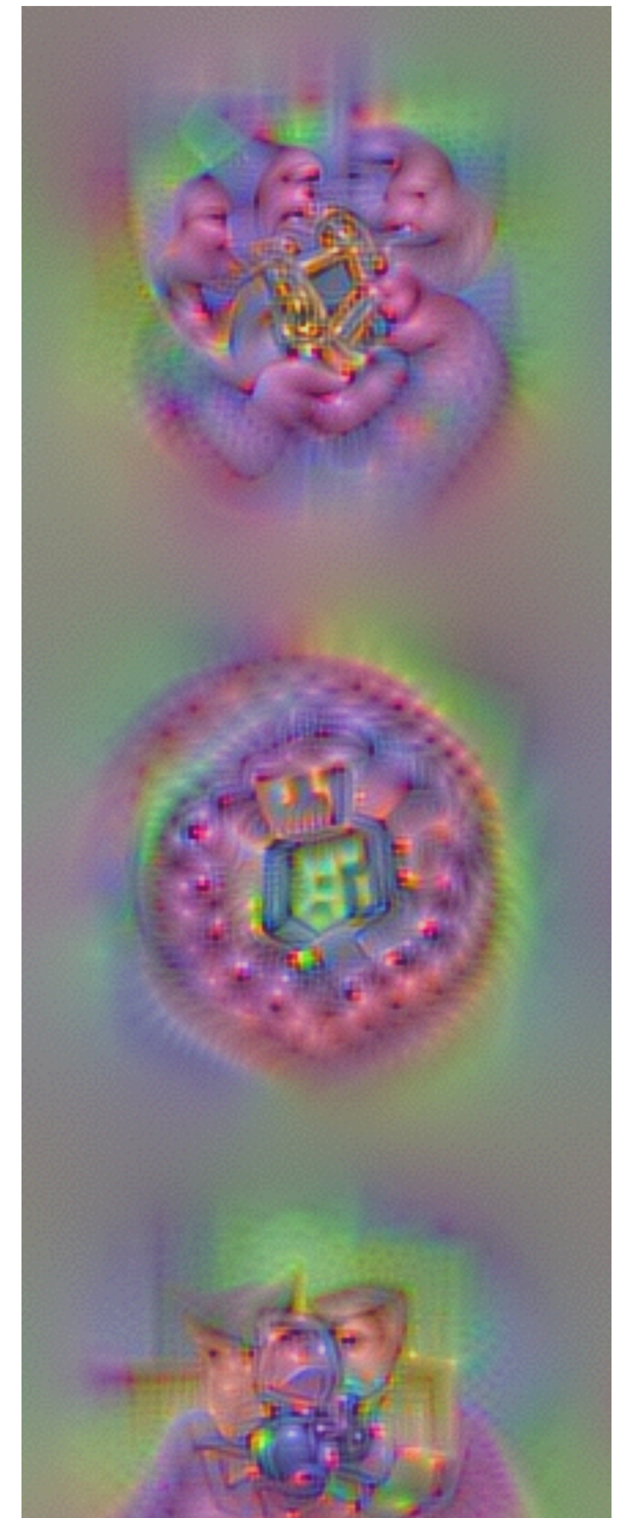
“conv5” features

AlexNet



VGG-M

VGG-VD



Caricaturization

[Google Inceptionism 2015, Mahendran et al. 2016]

Emphasise patterns that are detected by a certain representation

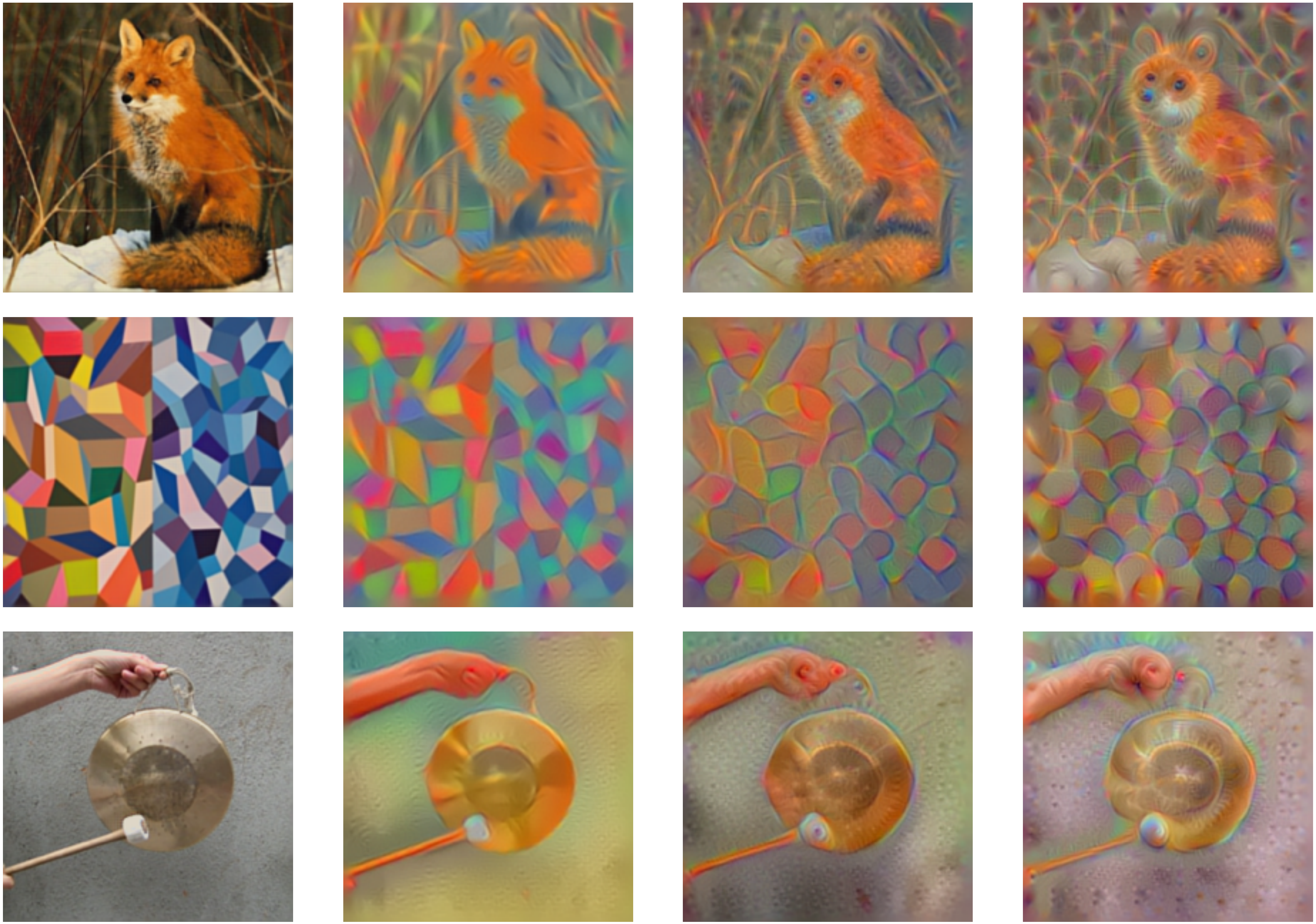
$$\boxed{\min_{\mathbf{x}} -\langle \Phi(\mathbf{x}_0), \Phi(\mathbf{x}) \rangle + R_{TV}(\mathbf{x}) + R_{\alpha}(\mathbf{x})}$$

Key differences:

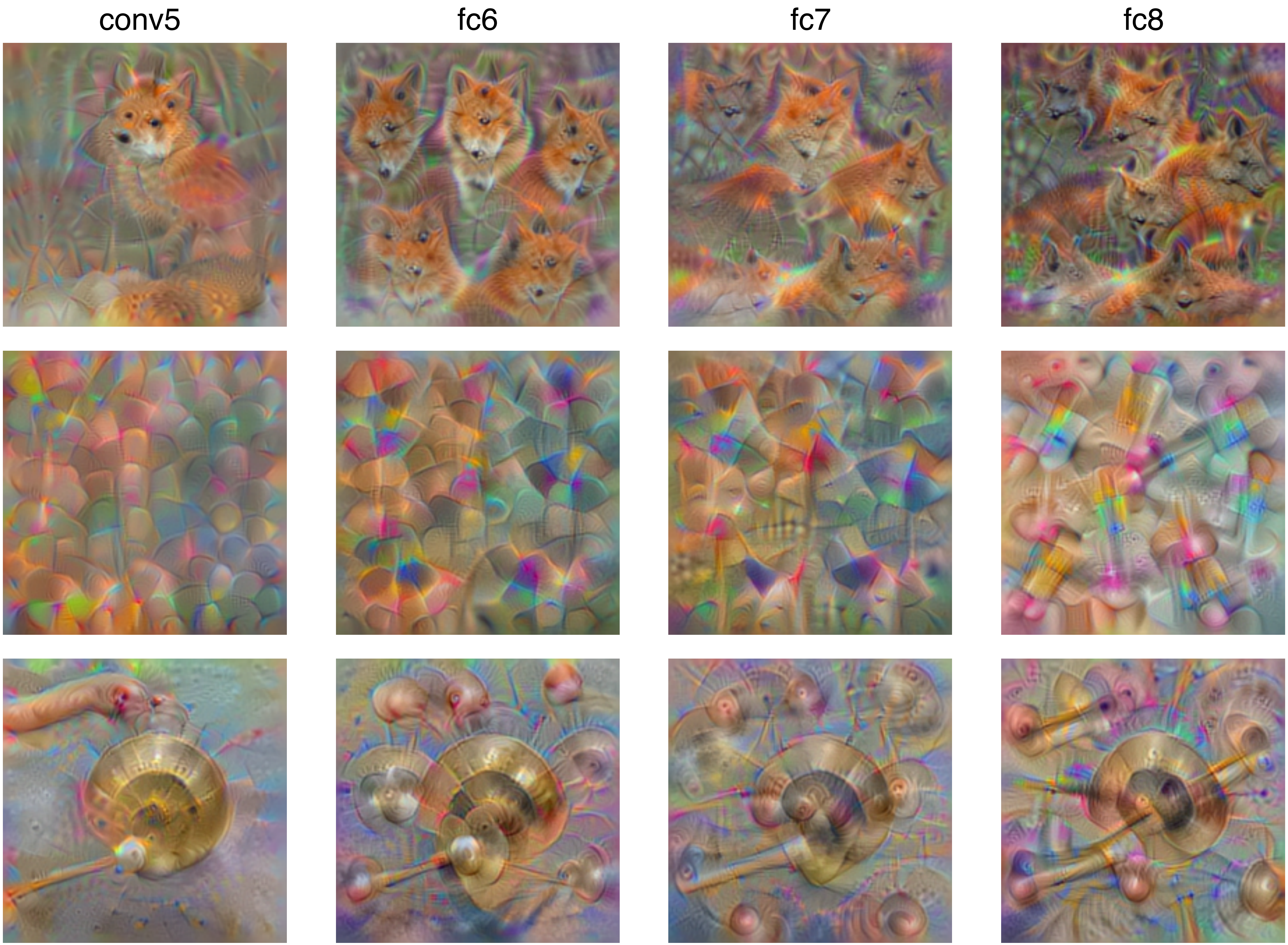
- the starting point **is** the image $\mathbf{x}_0$
- particular configurations of features are emphasized, not individual features

Caricaturization (VGG-M)

input conv2 conv3 conv4



Caricaturization (VGG-M)



Interlude: neural art

Surprisingly, the filters learned by discriminative neural networks capture well the "style" of an image.

This can be used to transfer the style of an image (e.g. a painting) to any other.

Optimisation based

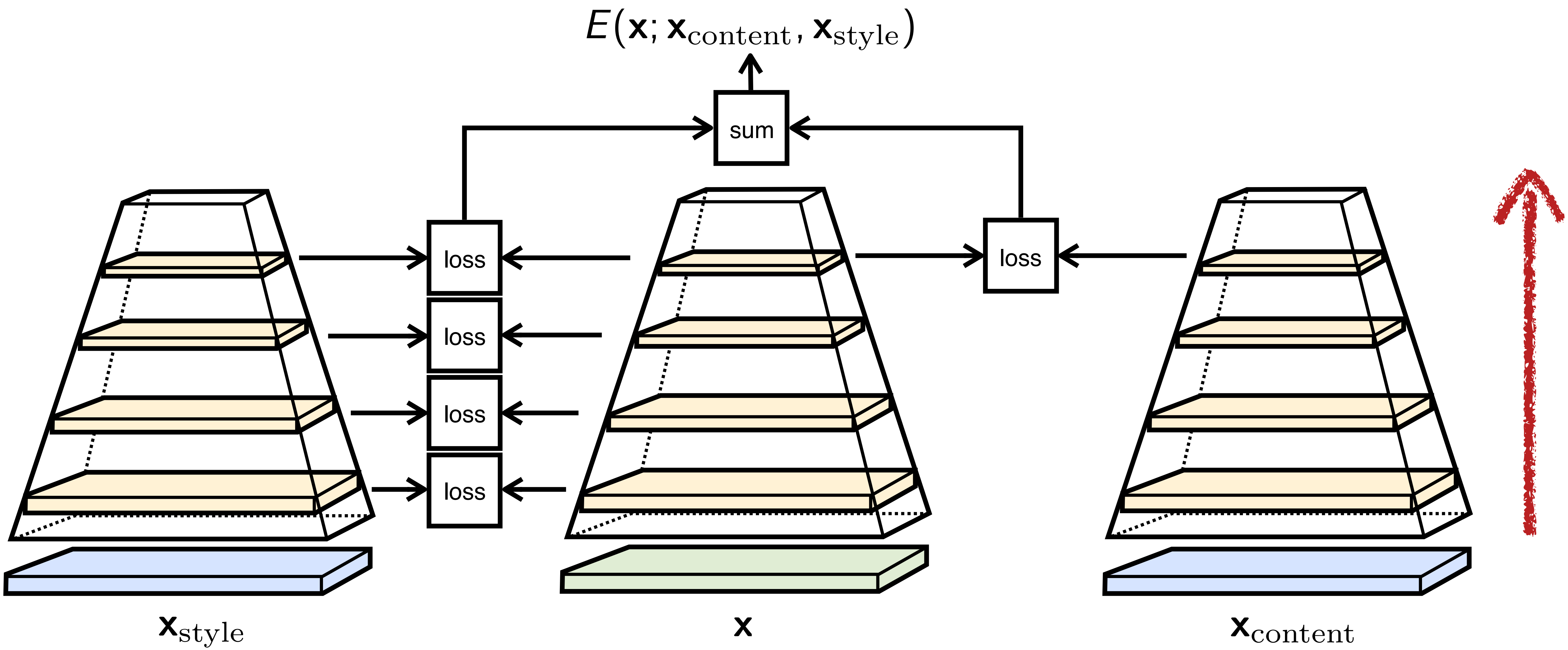
L. A. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis and the controlled generation of natural stimuli using convolutional neural networks. In Proc. NIPS, 2015.

Feed-forward neural network equivalents

D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. Proc. ICML, 2016.

J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In Proc. ECCV, 2016.

Generation by moment matching

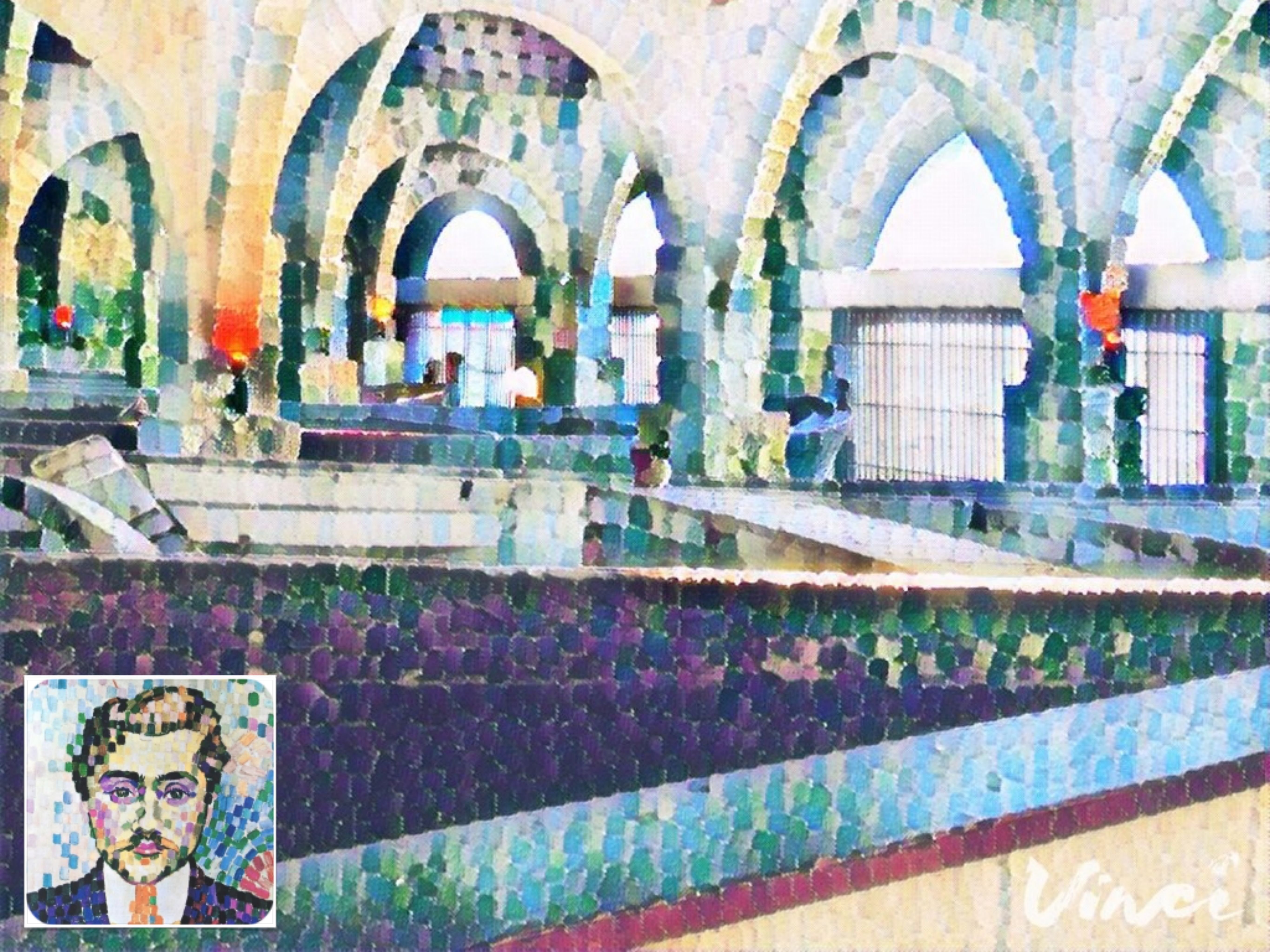


Moment matching

- Content statistics: same as inversion
- Style statistics: cross-channel correlations

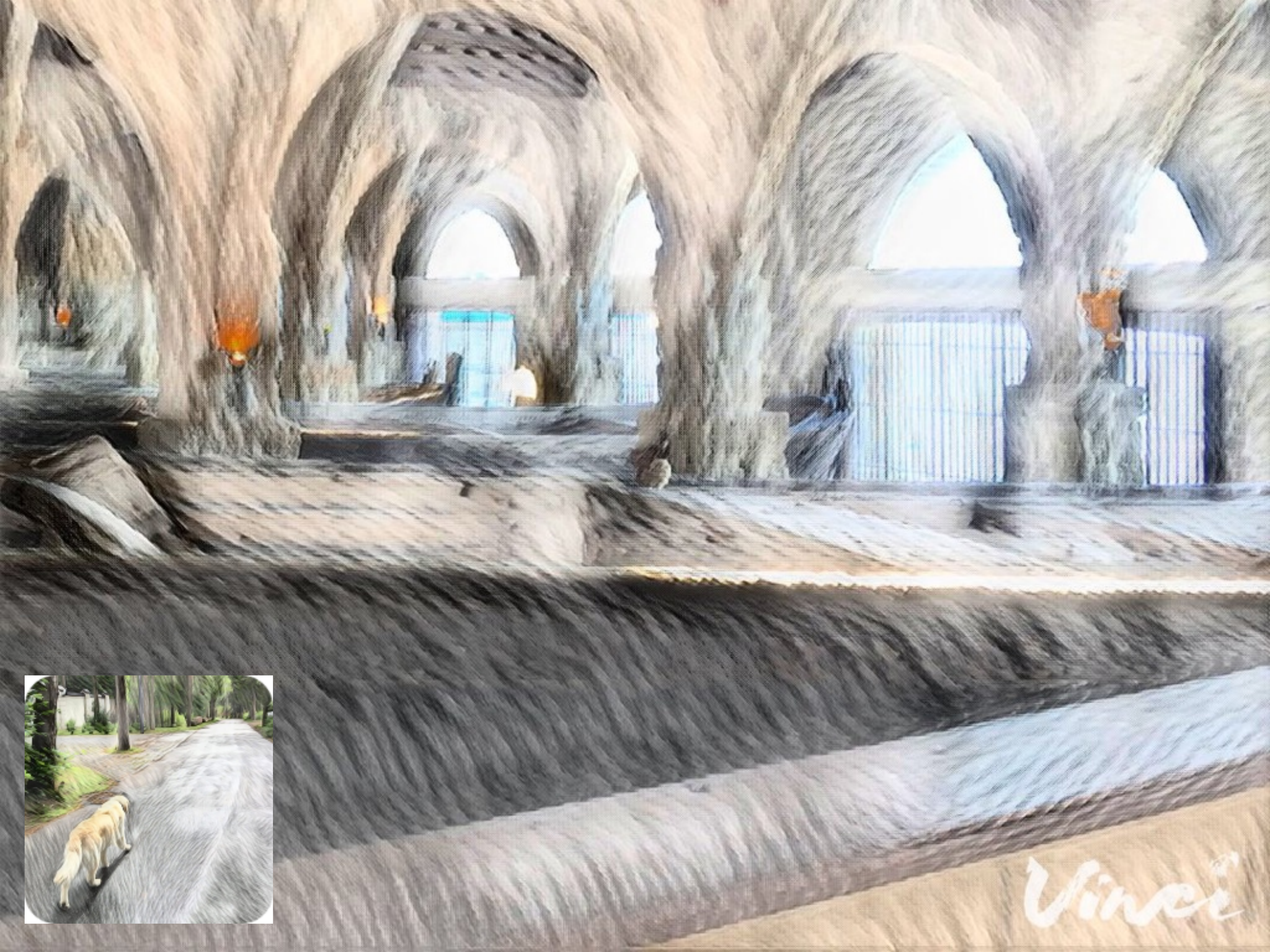
$$\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}} E(\mathbf{x}; \mathbf{x}_{\text{content}}, \mathbf{x}_{\text{style}})$$


Vinci


Vinci


Vinci


Vinci

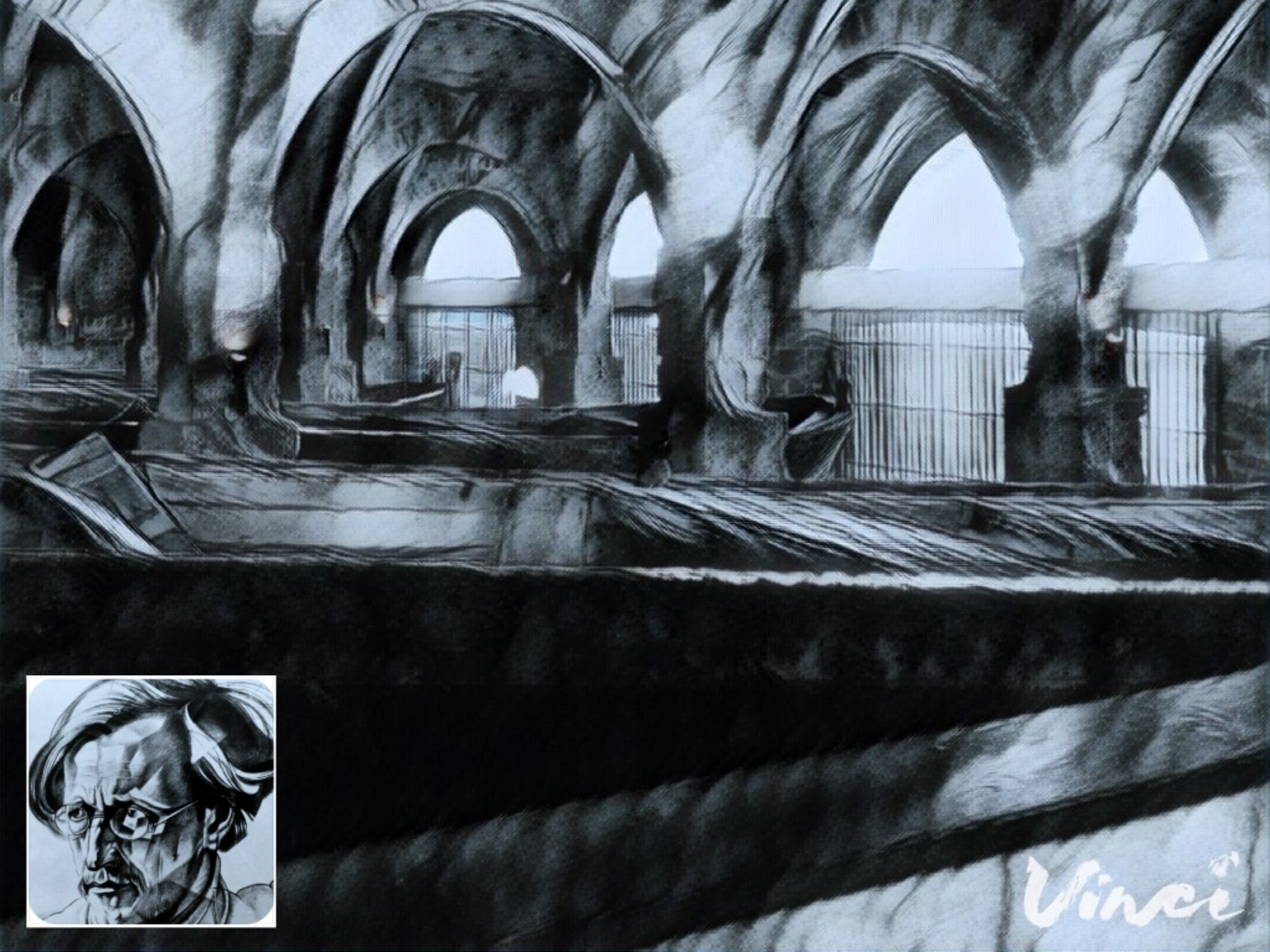

Vinci


Vinci


Vinci


Vinci


Vinci


Vinci


Vinci


Vinci


Vinci


Vinci

Understanding visual representations

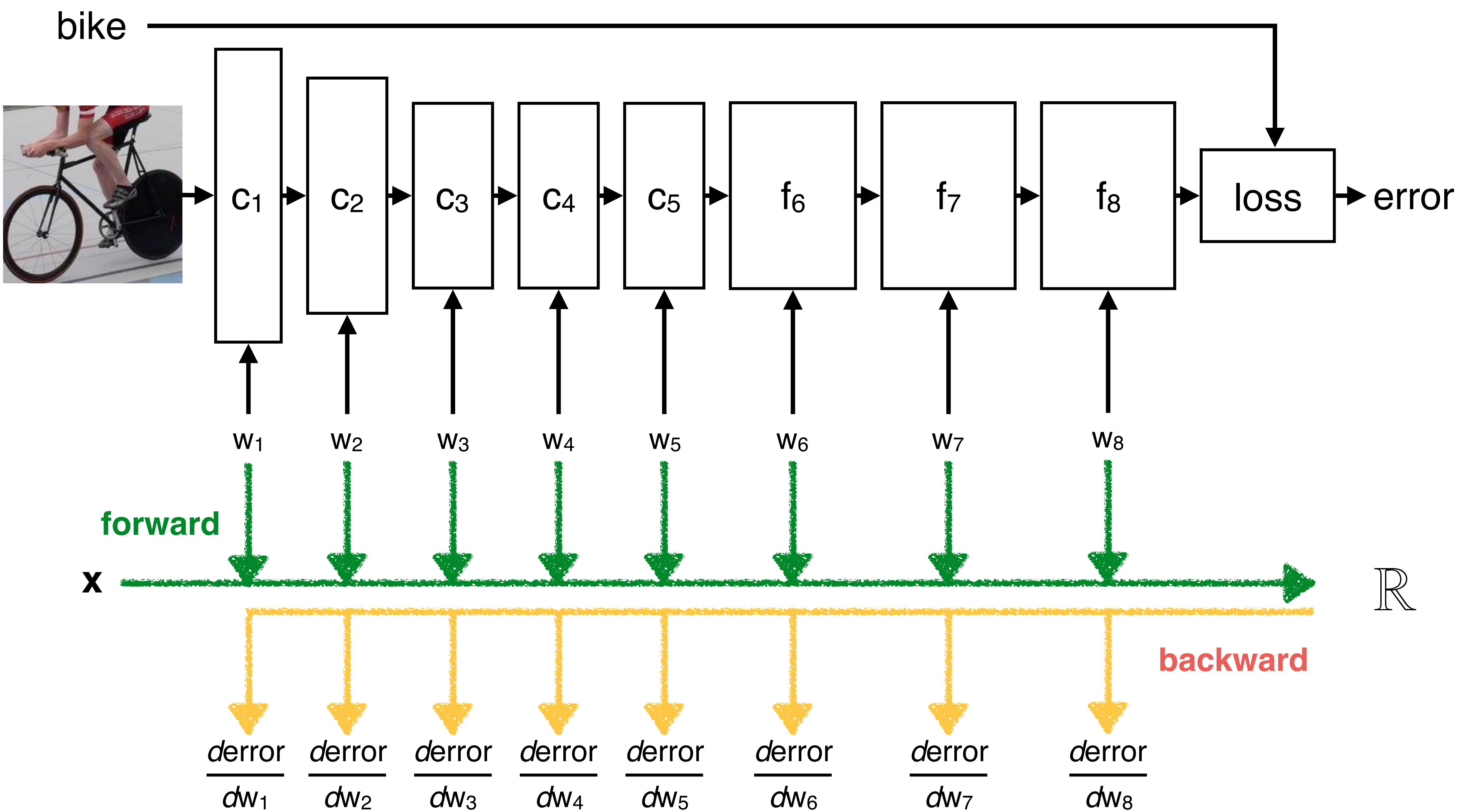
Visualizing representations

$\boxed{\text{Backpropagation networks and “deconvolution”}}$

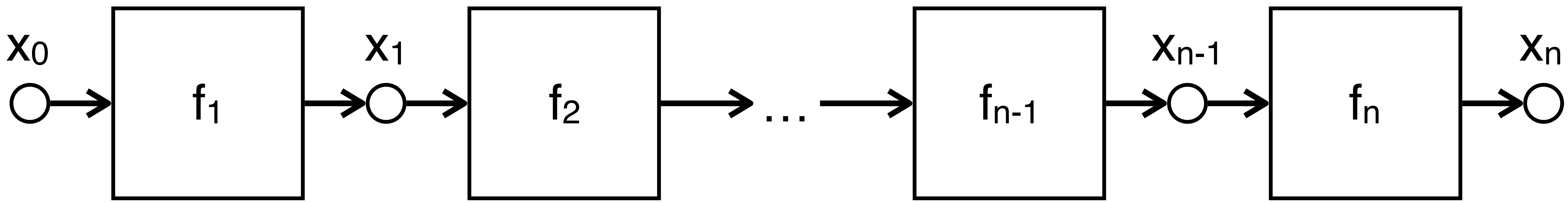
Representations: equivalence & transformations

Backpropagation

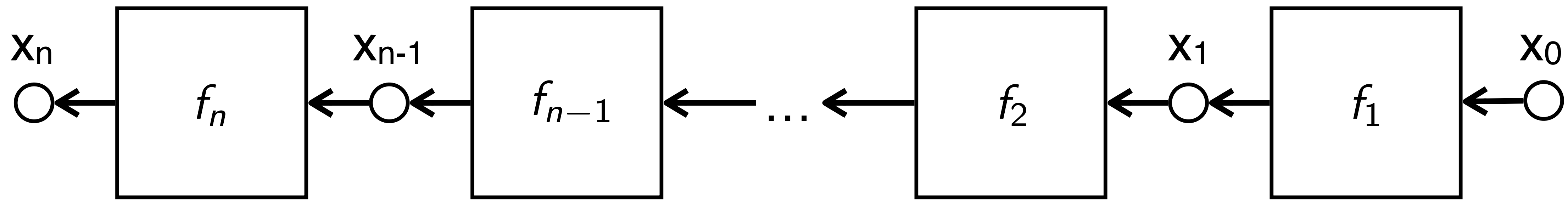
Compute derivatives using the chain rule



Chain rule: scalar version



Chain rule: scalar version



A composition of *n* functions

$$\mathbf{x}_n = \left(f_n \circ f_{n-1} \circ \ldots \circ f_2 \circ f_1 \right) (\mathbf{x}_0)$$

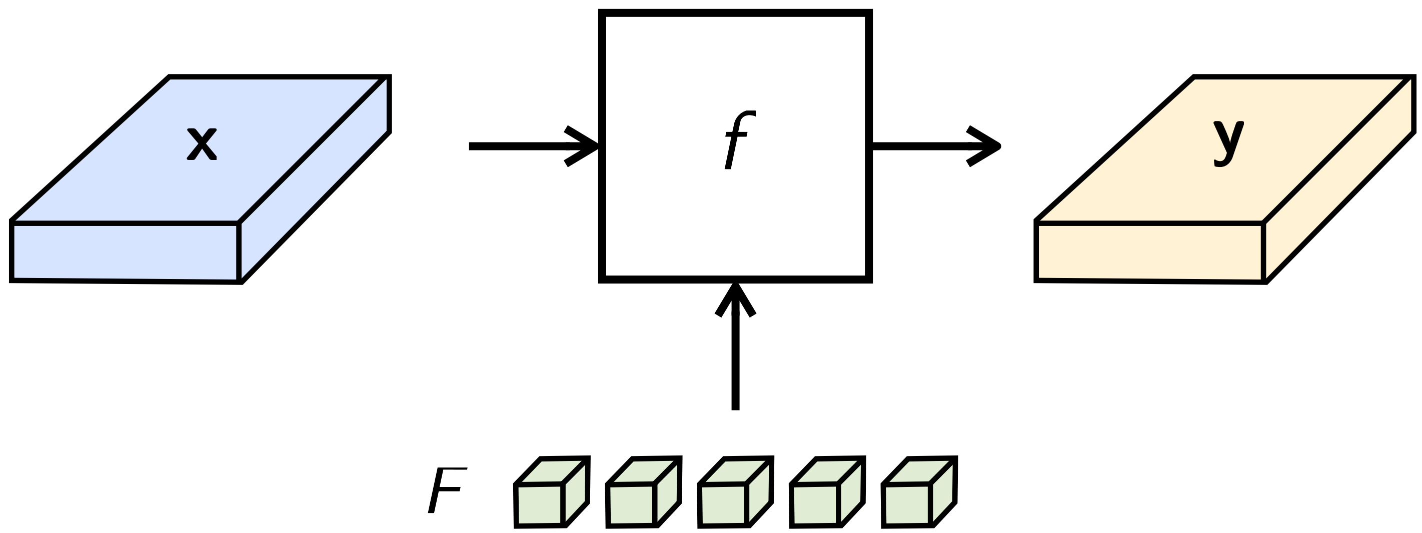
$$\frac{dx_n}{dx_0} = \frac{df_n}{dx_{n-1}} \times \frac{df_{n-1}}{dx_{n-2}} \times \ldots \times \frac{df_2}{dx_1} \times \frac{df_1}{dx_0}$$

Derivative ← chain rule

Tensor-valued functions

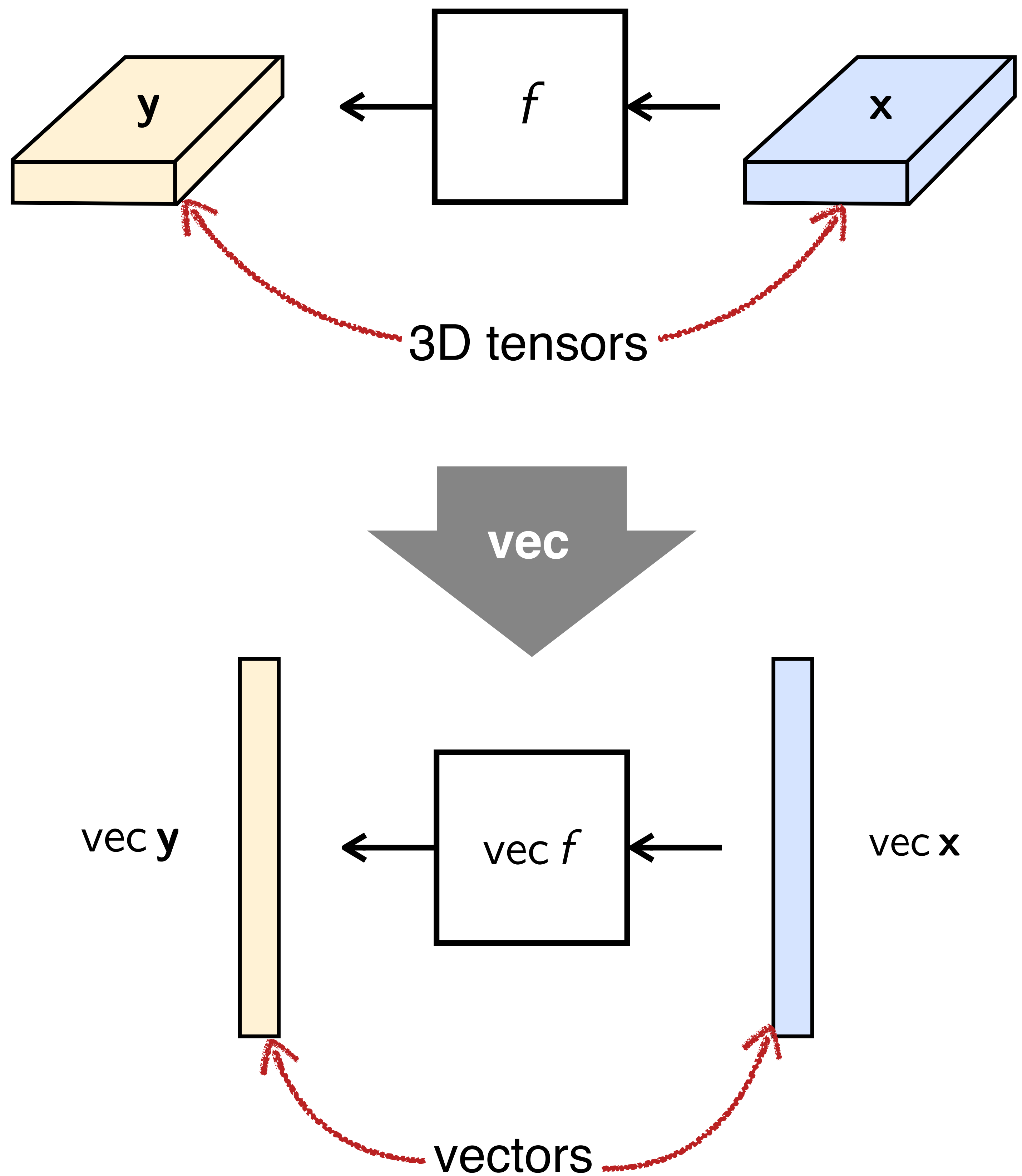
E.g. linear convolution = bank of 3D filters

$$\boxed{\mathbf{y} = F * \mathbf{x} + b}$$

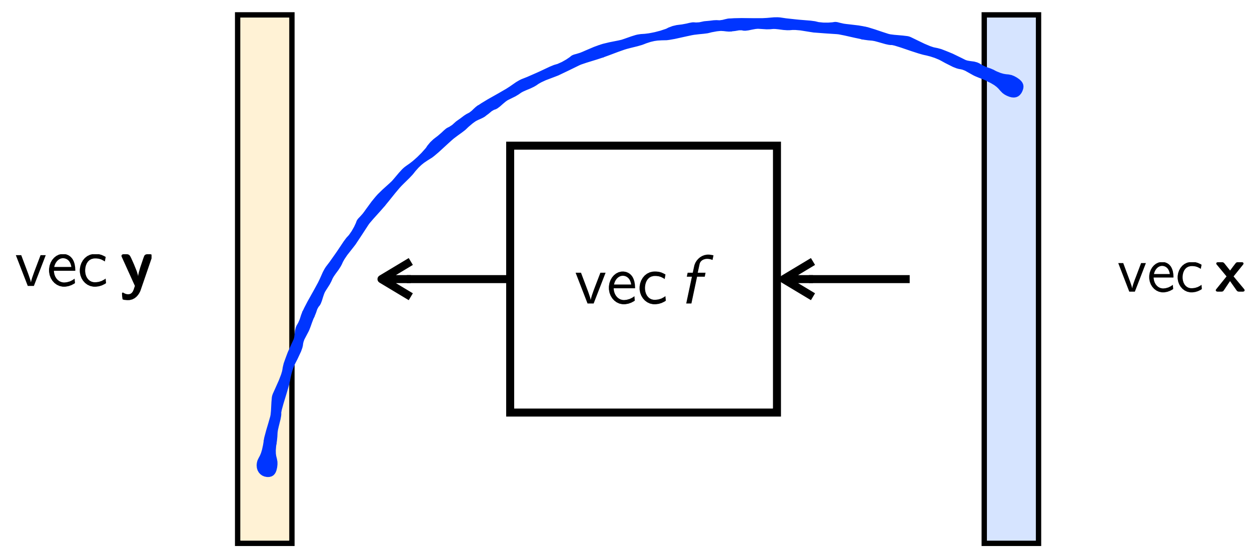


	height	width	channels	instances
input **x**	H	W	C	1 or N
filters F	H_f	W_f	C	K
output **y**	$H - H_f + 1$	$W - W_f + 1$	K	1 or N

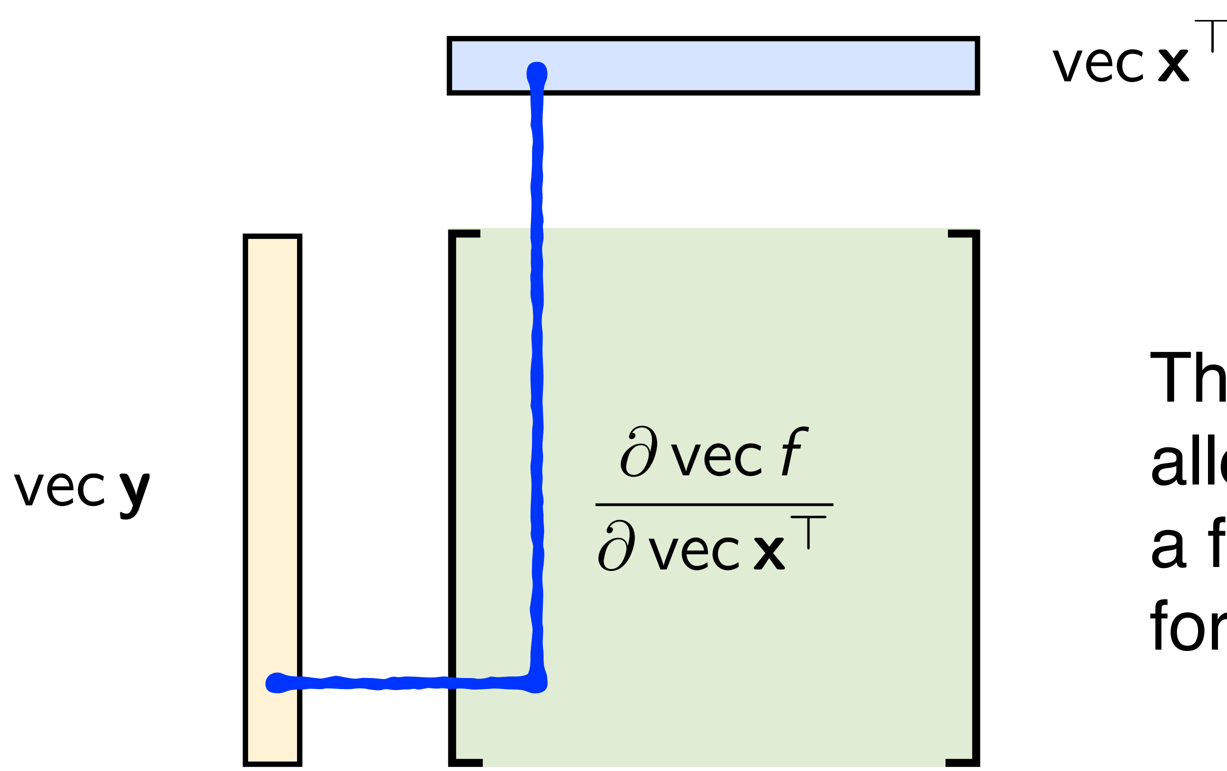
Vector representation



Derivative of tensor-valued functions



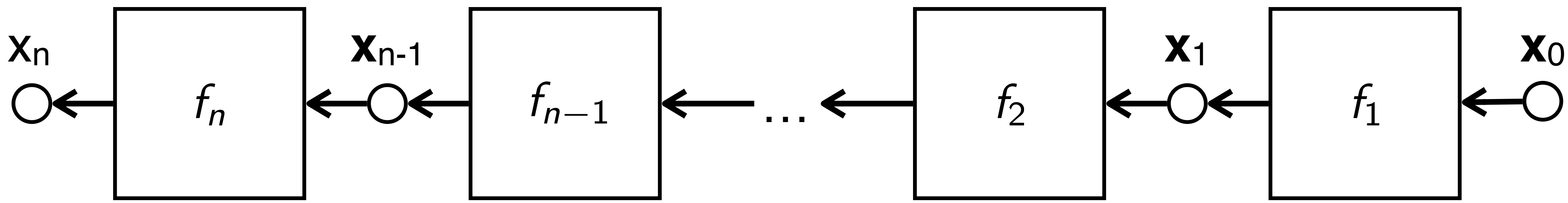
Derivative (Jacobian): every output element w.r.t. every input element!



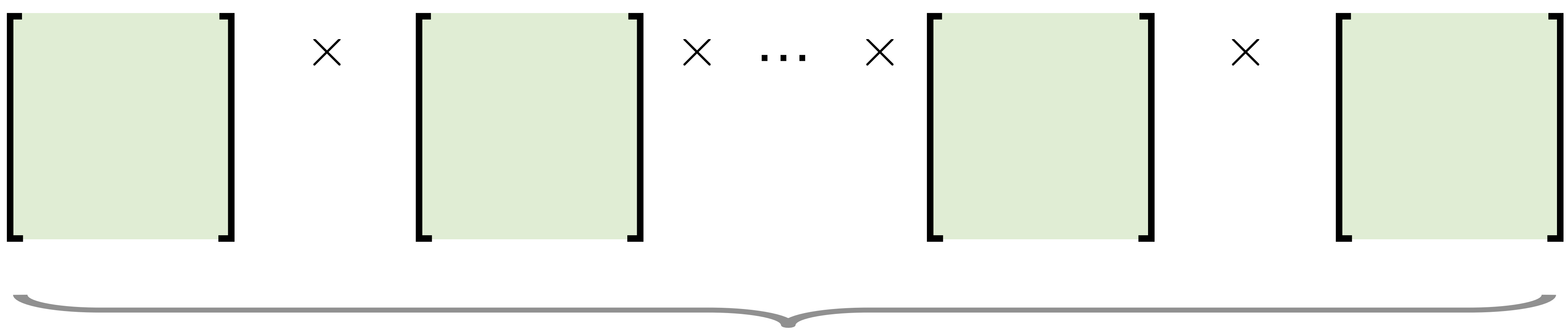
The vec operator allows us to use a familiar matrix notation for the derivatives

Chain rule: tensor version

Using vec() and matrix notation

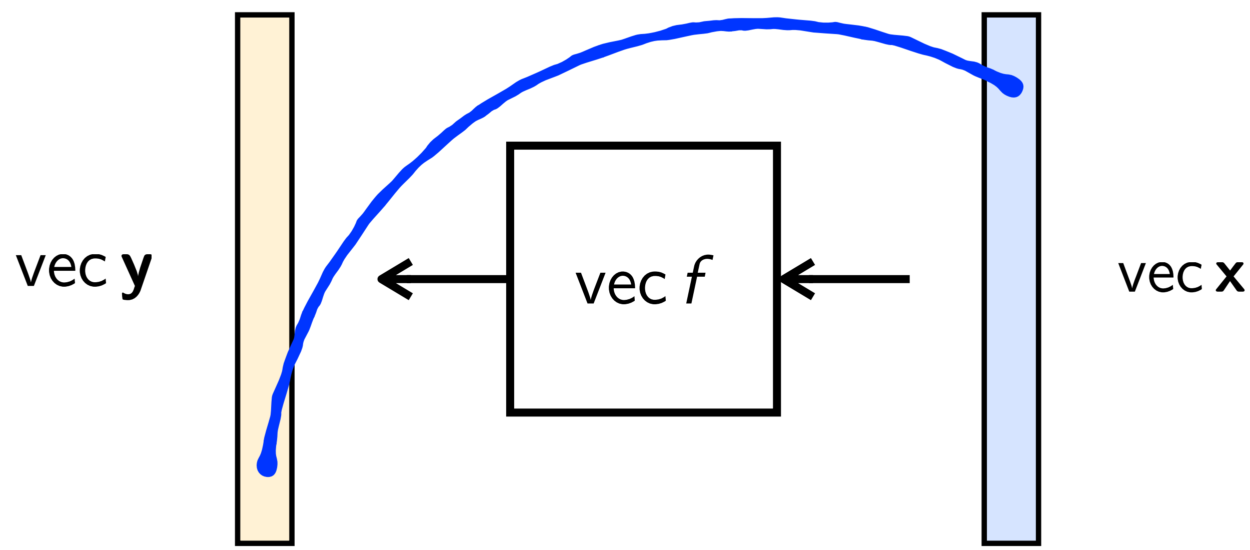


$$\frac{d \operatorname{vec} f_n}{d \operatorname{vec} \mathbf{x}_{n-1}^\top} \times \frac{d \operatorname{vec} f_{n-1}}{d \operatorname{vec} \mathbf{x}_{n-2}^\top} \times \cdots \times \frac{d \operatorname{vec} f_2}{d \operatorname{vec} \mathbf{x}_1^\top} \times \frac{d \operatorname{vec} f_1}{d \operatorname{vec} \mathbf{x}_0^\top}$$

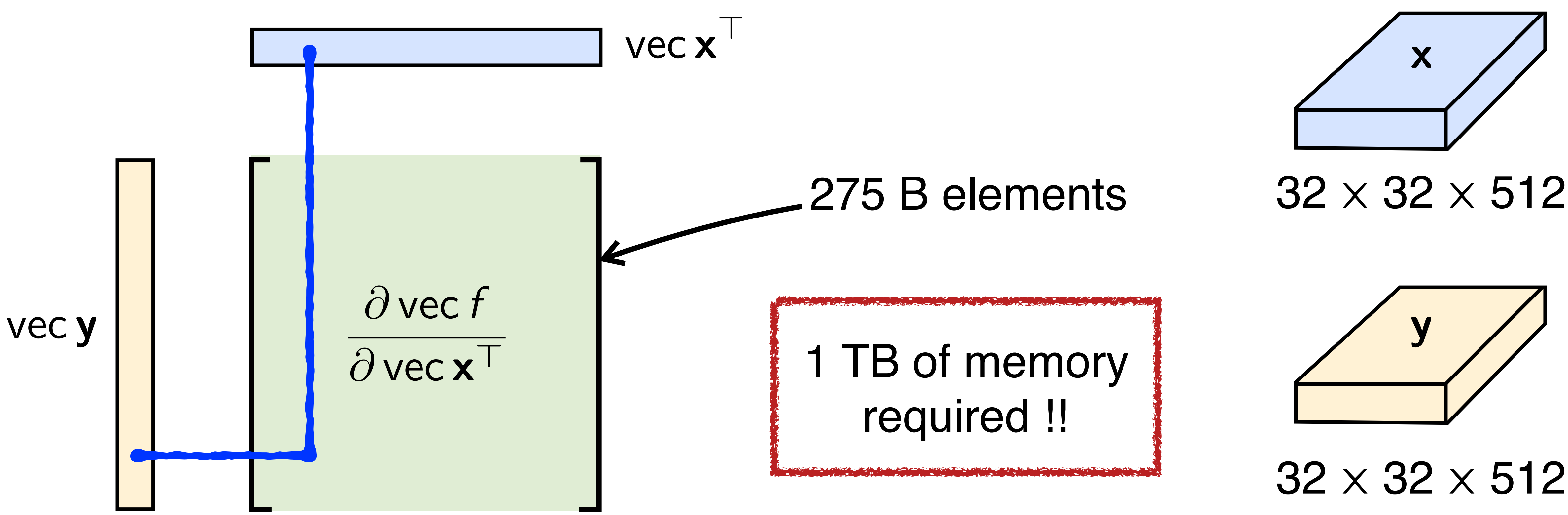


$$\frac{d \operatorname{vec} f}{d \operatorname{vec} \mathbf{x}_0^\top}$$

The (unbearable) size of tensor derivatives



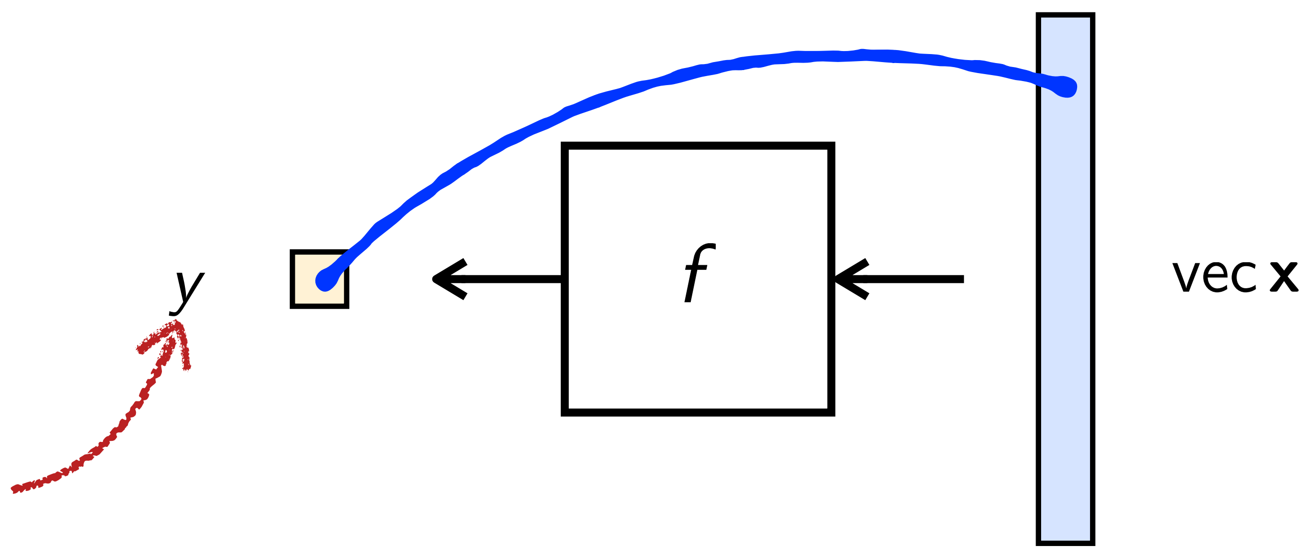
The size of these Jacobian matrices is **huge**. Example:



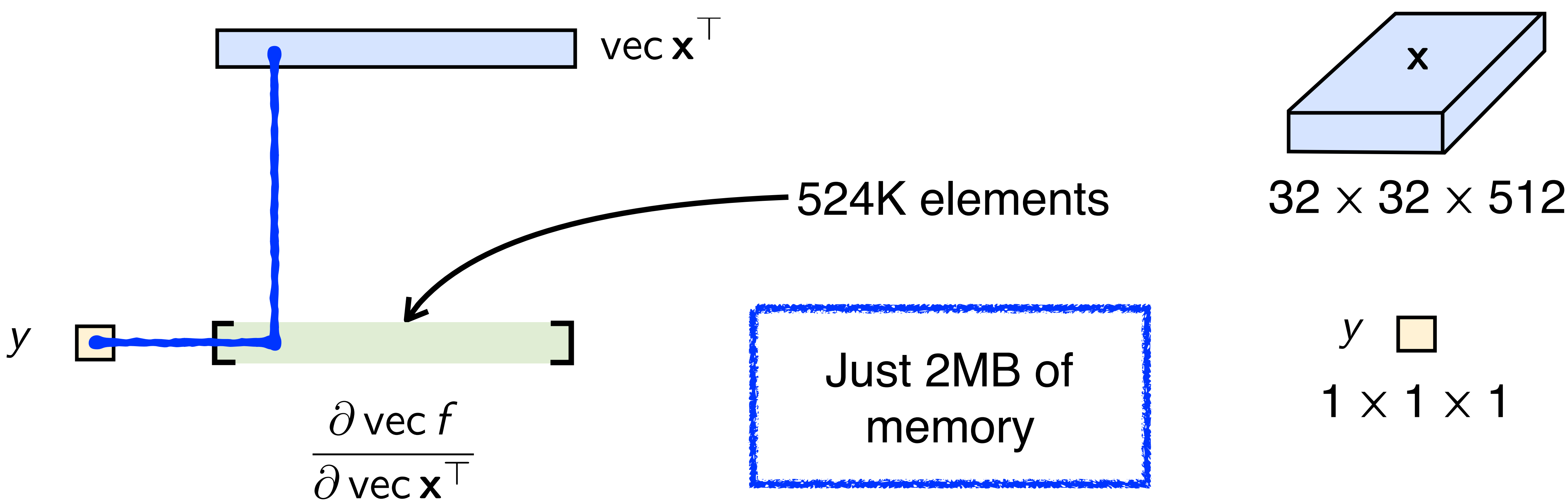
Unless the output is a scalar

Scalar

This is always the case
if the last layer
is the **loss function**

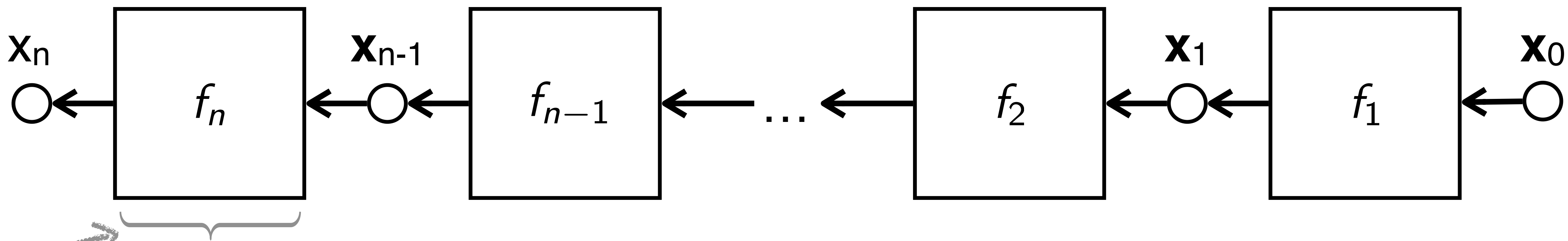


Now the Jacobian has the same size as **x**. Example:

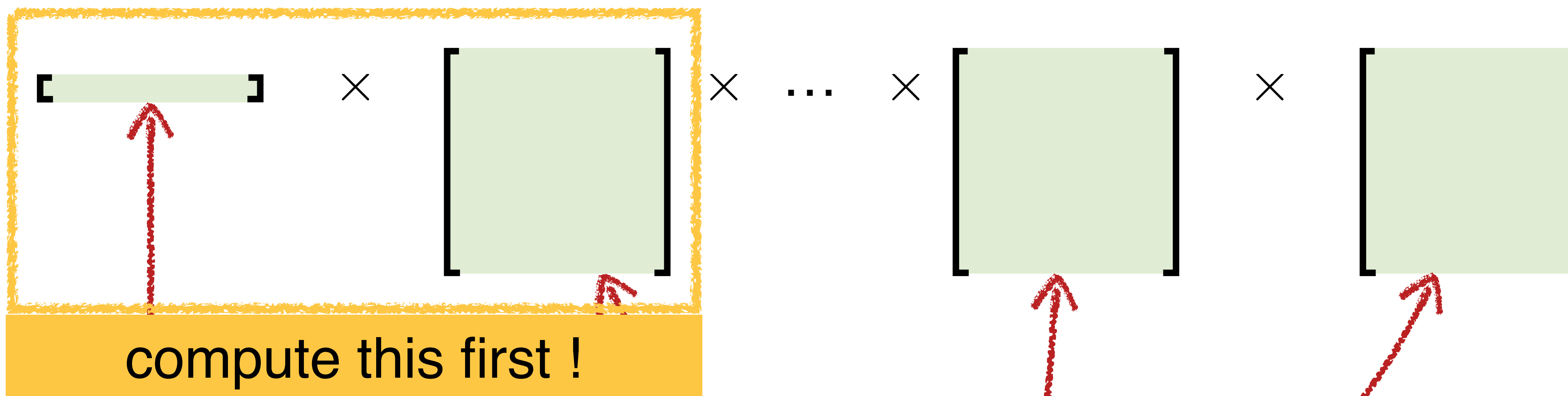


Backpropagation

Assume that x_n is a scalar (e.g. loss)



$$\frac{df_n}{d\,\mathrm{vec}\,\mathbf{x}_{n-1}^\top} \times \frac{d\,\mathrm{vec}\,f_{n-1}}{d\,\mathrm{vec}\,\mathbf{x}_{n-2}^\top} \times \cdots \times \frac{d\,\mathrm{vec}\,f_2}{d\,\mathrm{vec}\,\mathbf{x}_1^\top} \times \frac{d\,\mathrm{vec}\,f_1}{d\,\mathrm{vec}\,\mathbf{x}_0^\top}$$

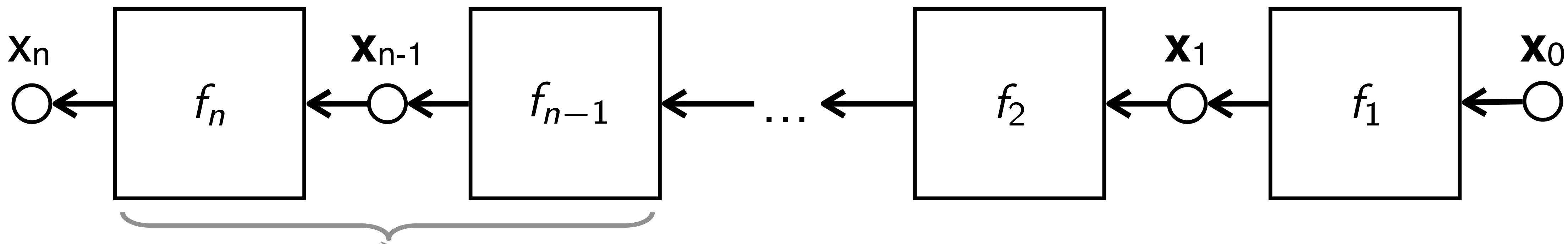


small
explicitly compute

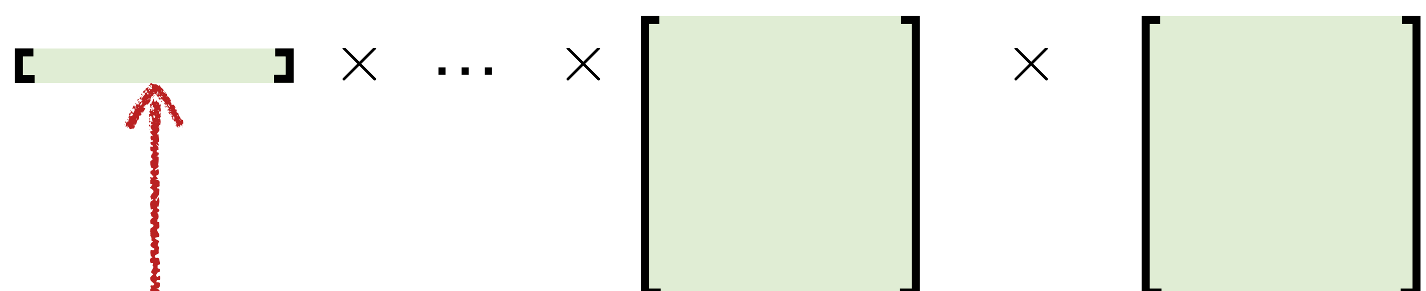
uber matrices
do **not** explicitly compute

Backpropagation

Assume that x_n is a scalar (e.g. loss)



$$\frac{d f_n \circ f_{n-1}}{d \operatorname{vec} \mathbf{x}_{n-2}^\top} \times \cdots \times \frac{d \operatorname{vec} f_2}{d \operatorname{vec} \mathbf{x}_1^\top} \times \frac{d \operatorname{vec} f_1}{d \operatorname{vec} \mathbf{x}_0^\top}$$

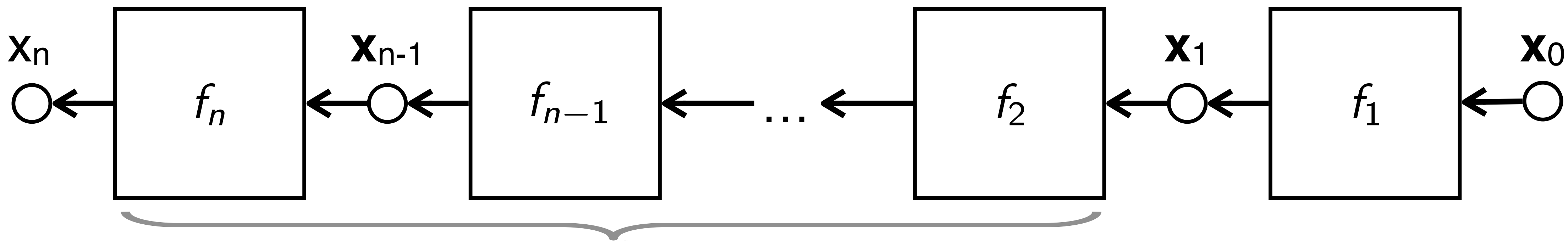


small
explicitly compute

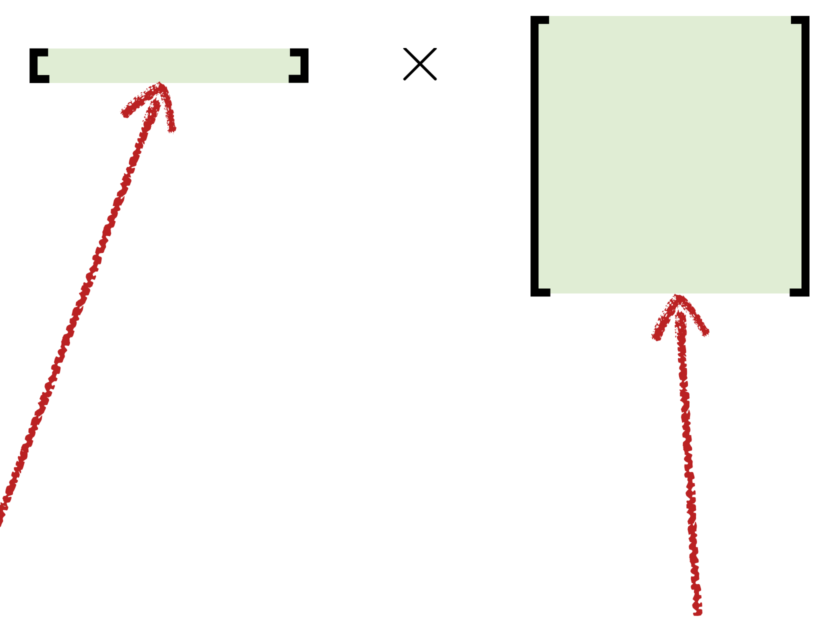
uber matrices
do **not** explicitly compute

Backpropagation

Assume that x_n is a scalar (e.g. loss)



$$\frac{d f_n \circ \cdots \circ f_2}{d \operatorname{vec} \mathbf{x}_1^\top} \times \frac{d \operatorname{vec} f_1}{d \operatorname{vec} \mathbf{x}_0^\top}$$

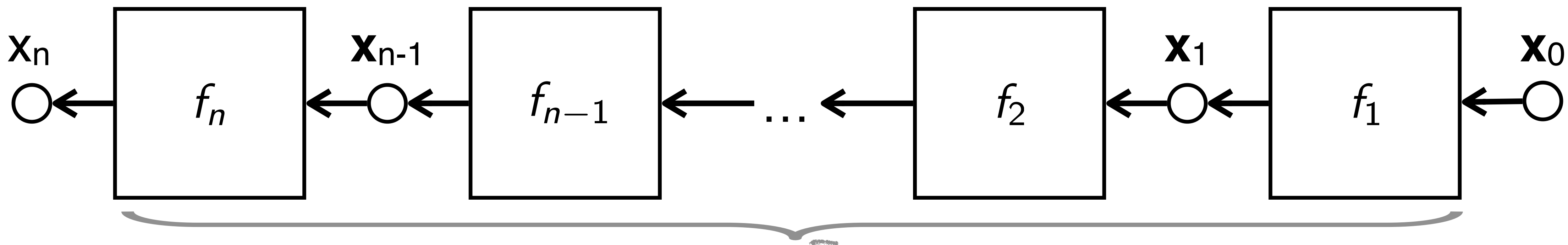


small
explicitly compute

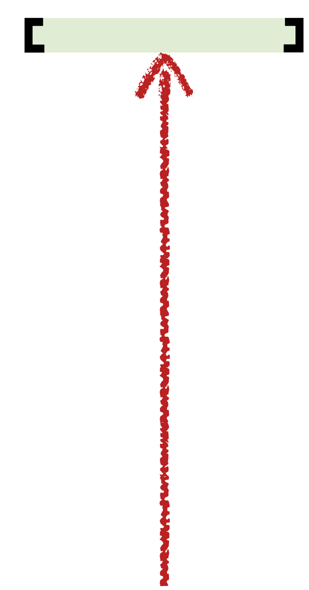
uber matrix
do **not** explicitly compute

Backpropagation

Assume that x_n is a scalar (e.g. loss)



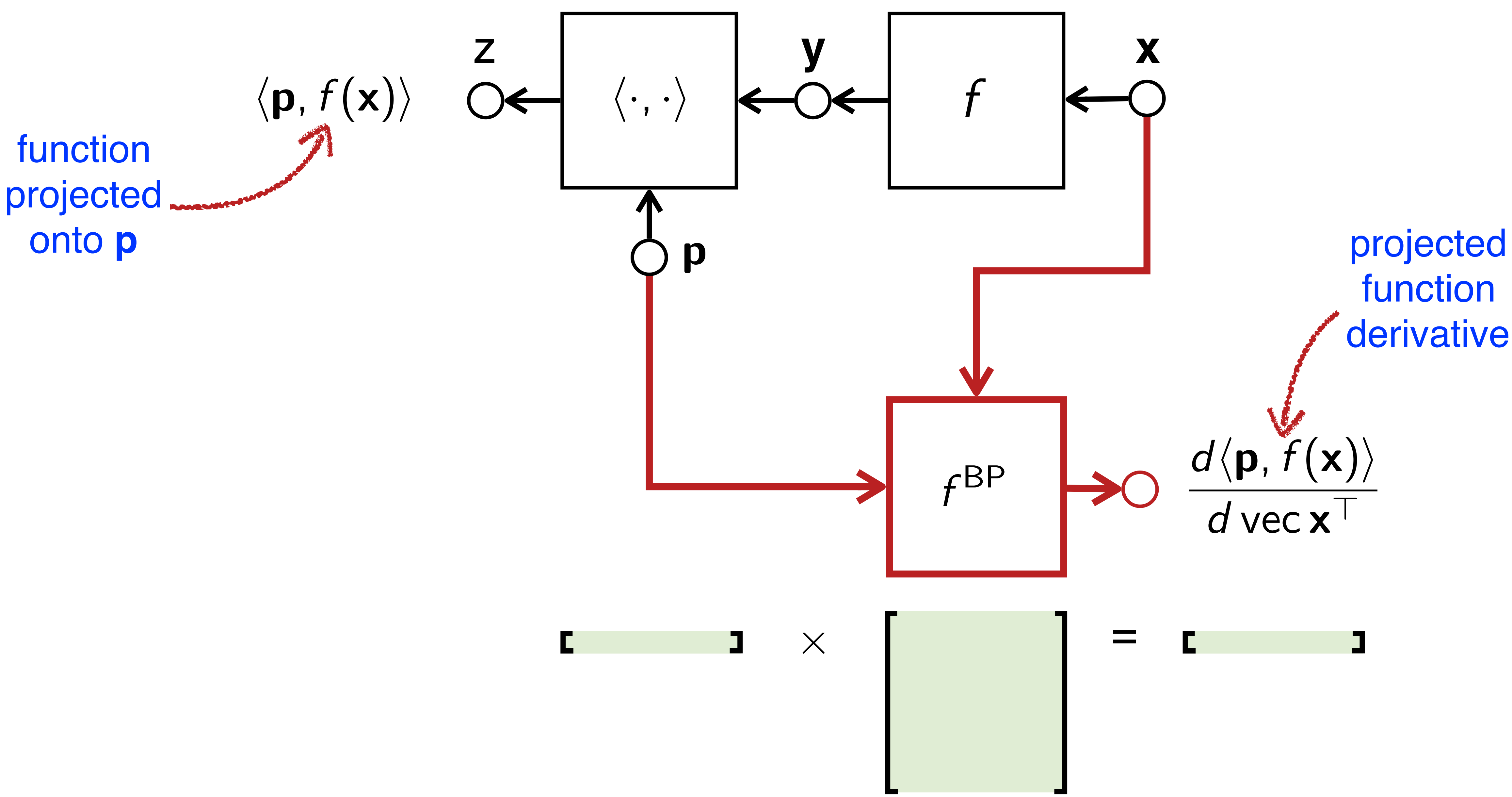
$$\frac{d f_n \circ \cdots \circ f_1}{d \operatorname{vec} \mathbf{x}_0^\top}$$



small
explicitly compute

Projected function derivative

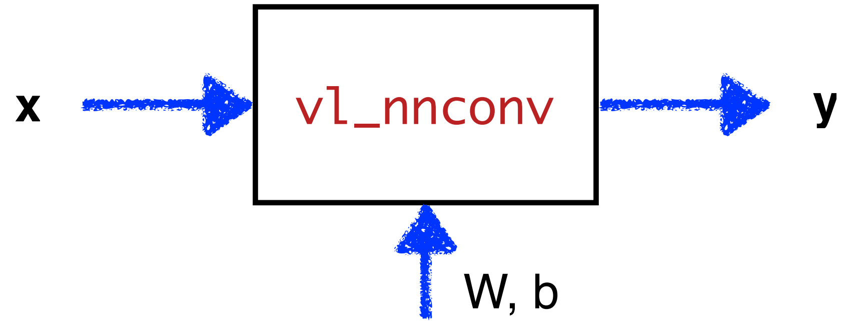
The “BP-reversed” layer



An “equivalent circuit” is obtained by introducing a transposed function f^T

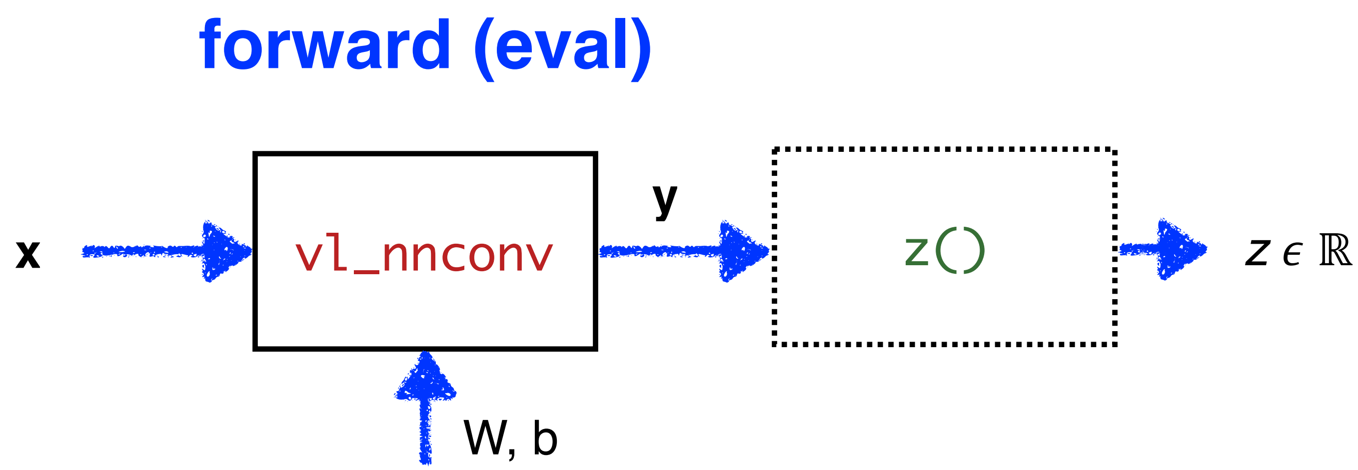
Anatomy of a building block

forward (eval)




```
y = vl_nnconv(x, W, b)
```

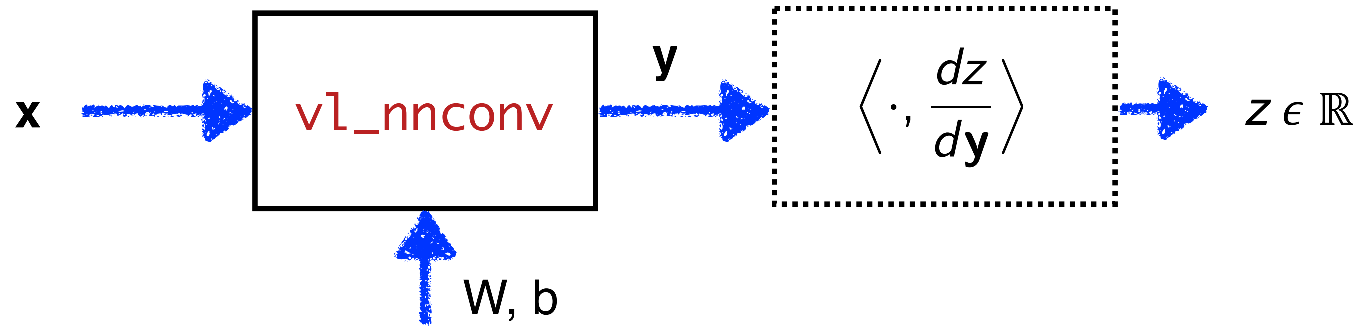
Anatomy of a building block




```
y = vl_nnconv(x, W, b)
```

Anatomy of a building block

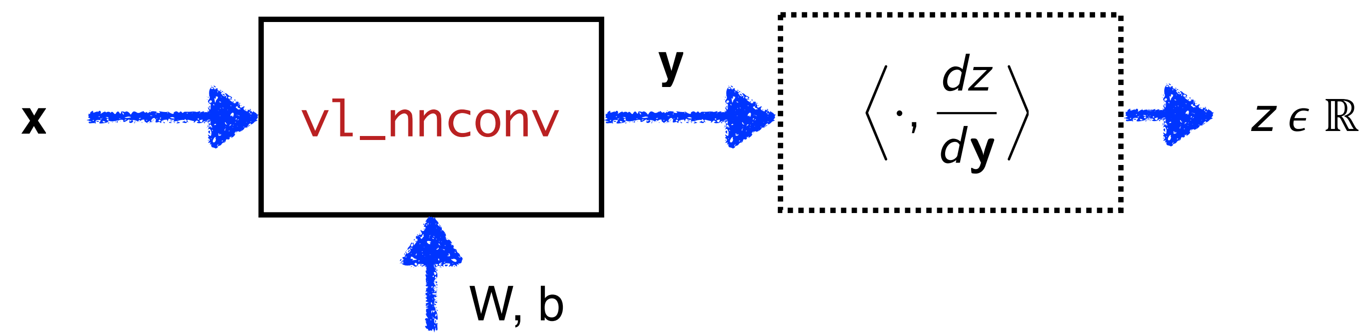
forward (eval)




```
y = vl_nnconv(x, W, b)
```

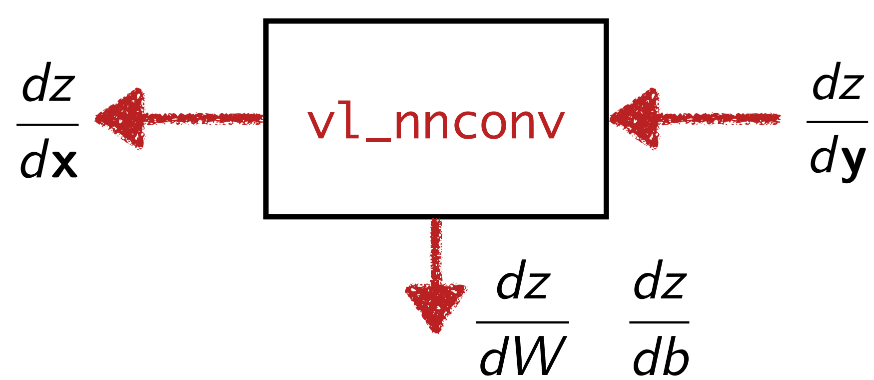
Anatomy of a building block

forward (eval)




```
y = vl_nnconv(x, W, b)
```

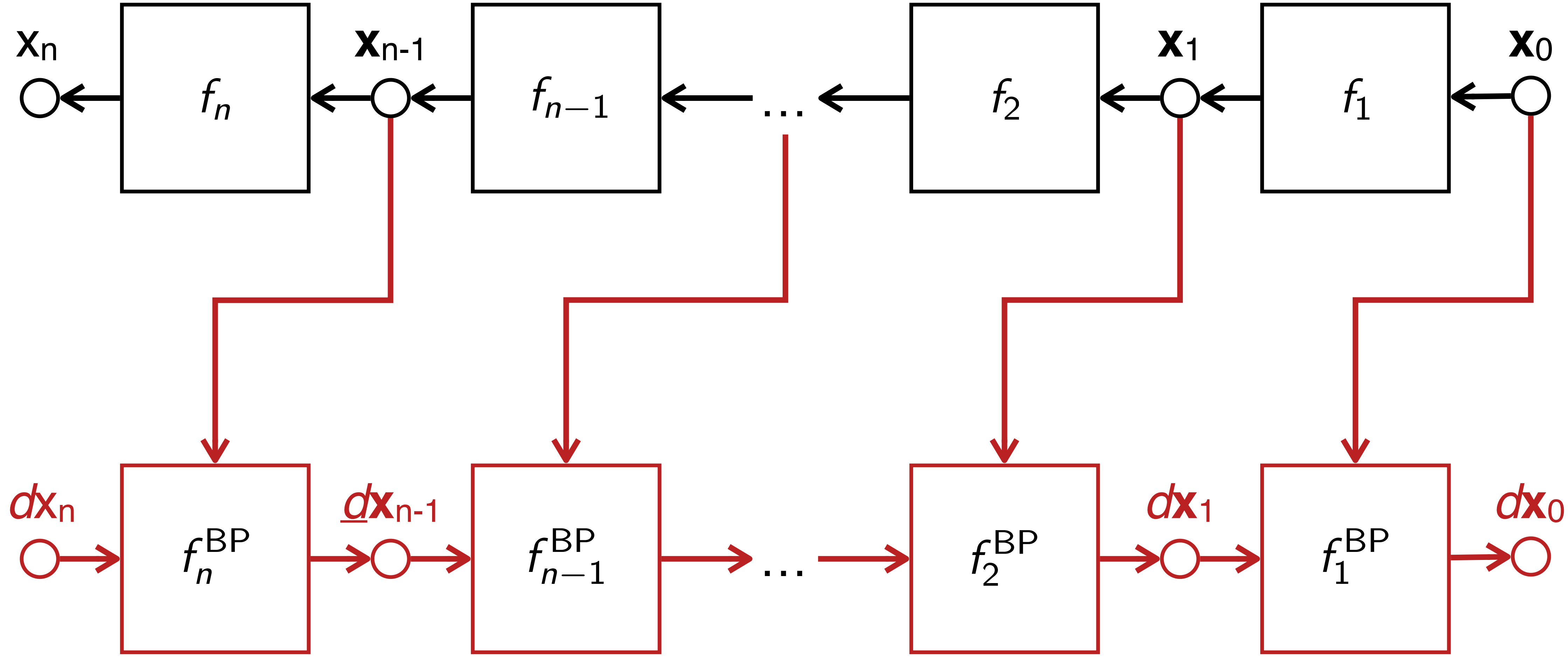
backward (backprop)




```
dzdx = vl_nnconv(x, W, b, dzdy)
```

Backpropagation network

BP induces a “reversed” network

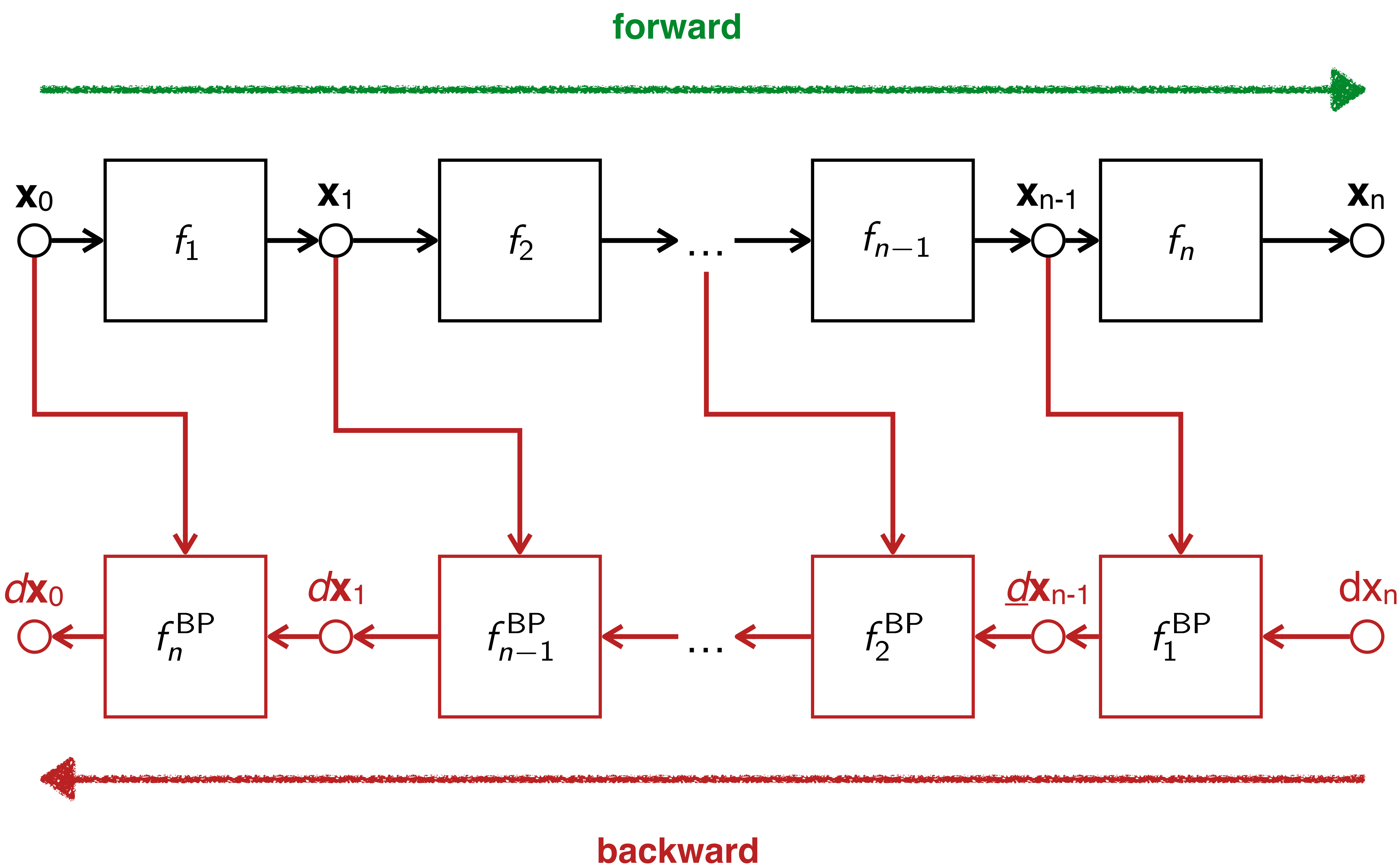


where $$d\mathbf{x}_i = \frac{df_n \circ \cdots \circ f_{i+1}}{d \operatorname{vec} \mathbf{x}_i}$$

Note: the BP network is linear in $d\mathbf{x}_1$, …, $d\mathbf{x}_{n-1}$, $d\mathbf{x}_n$. Why?

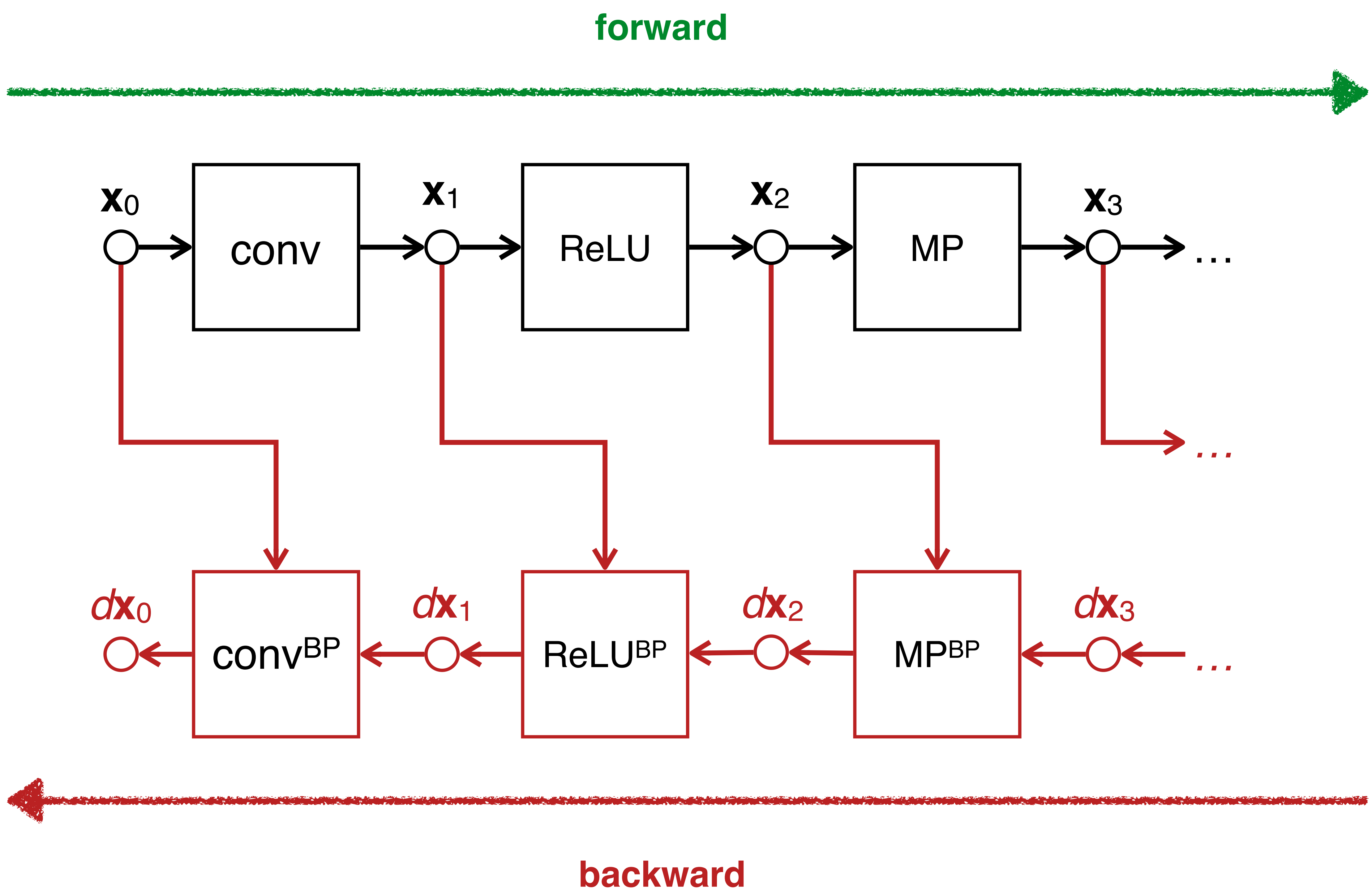
Backpropagation network

BP induces a “transposed” network



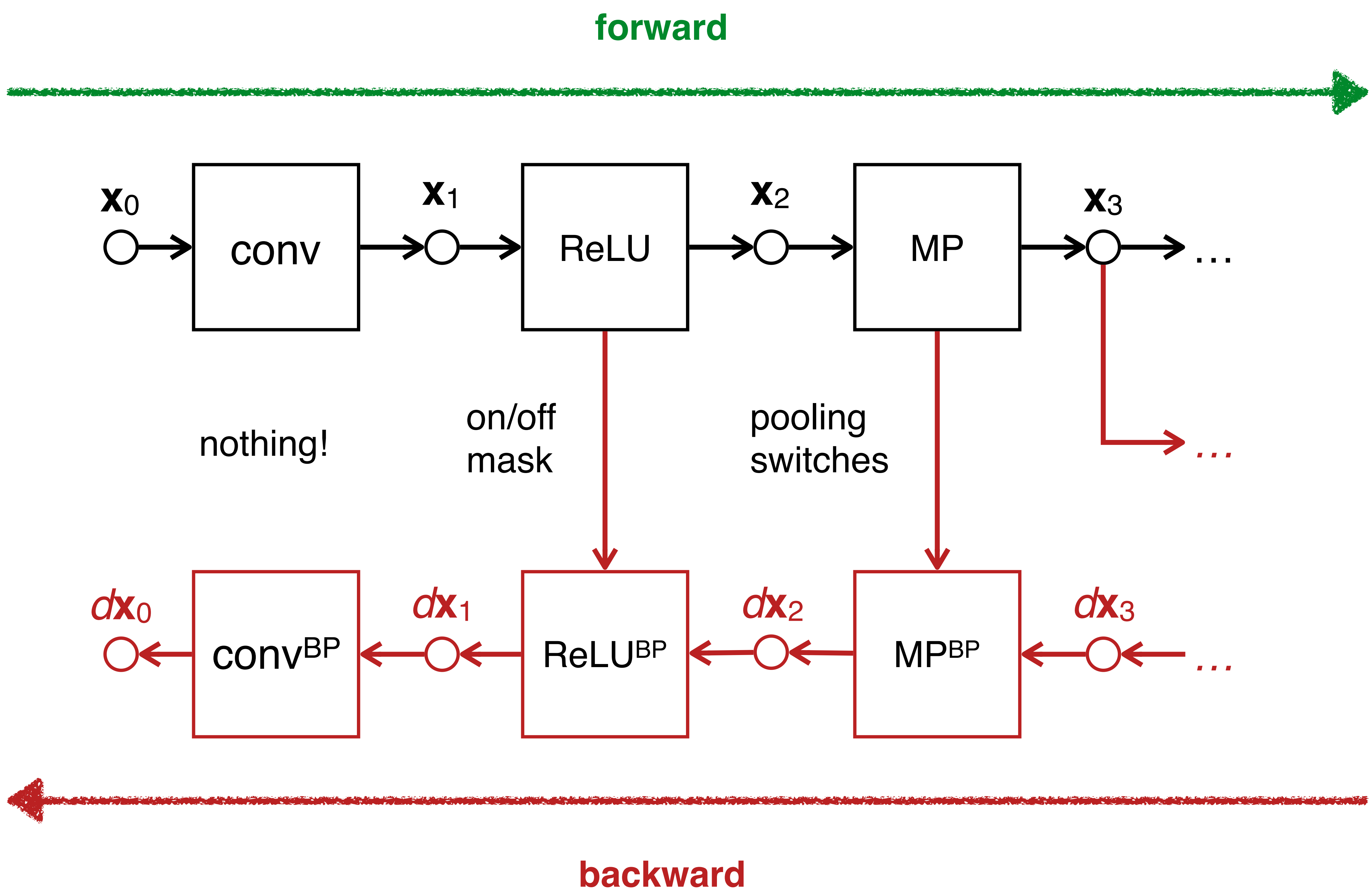
Backpropagation network

Conv, ReLU, MP and their transposed blocks



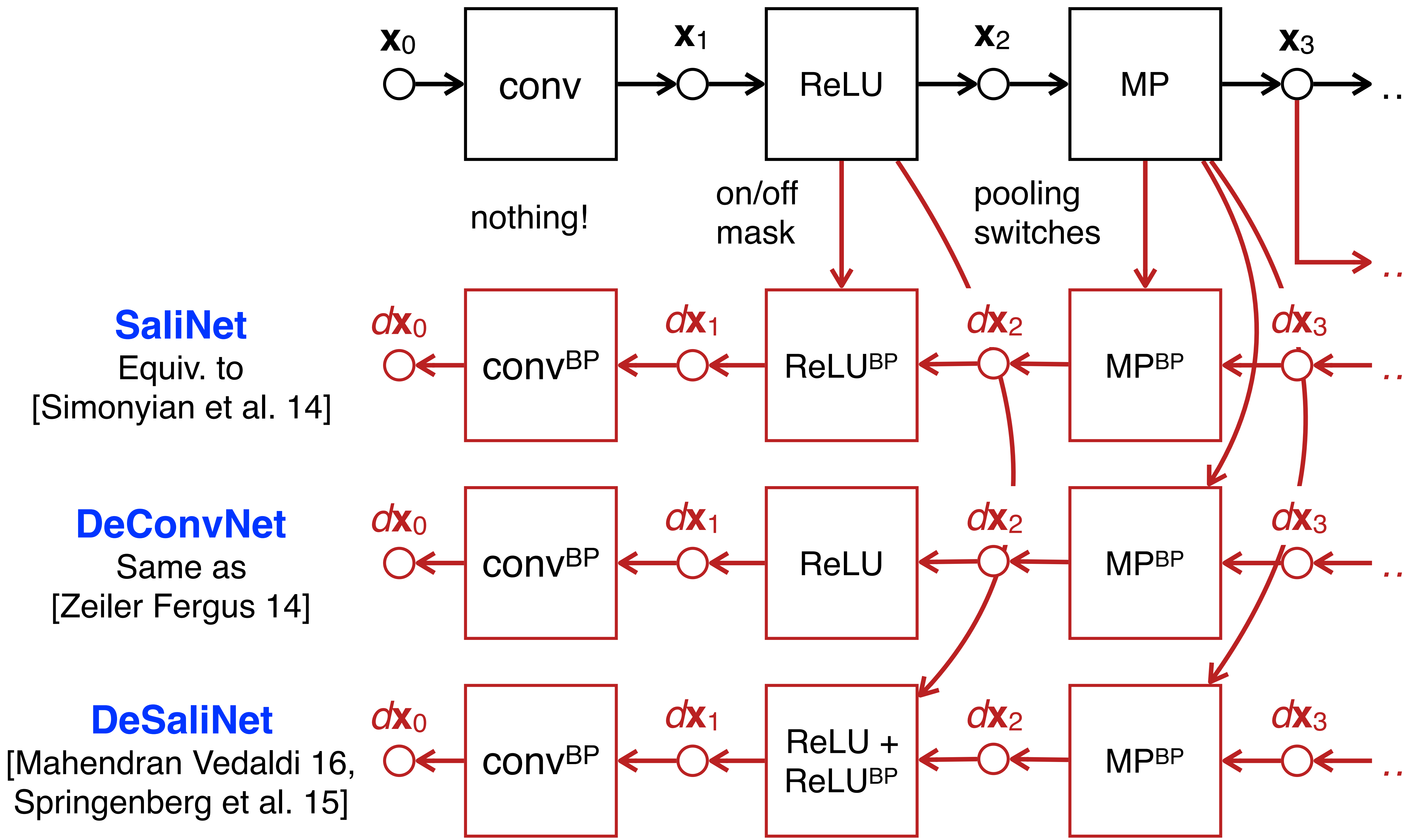
Sufficient statistics and bottlenecks

Usually much less information is needed

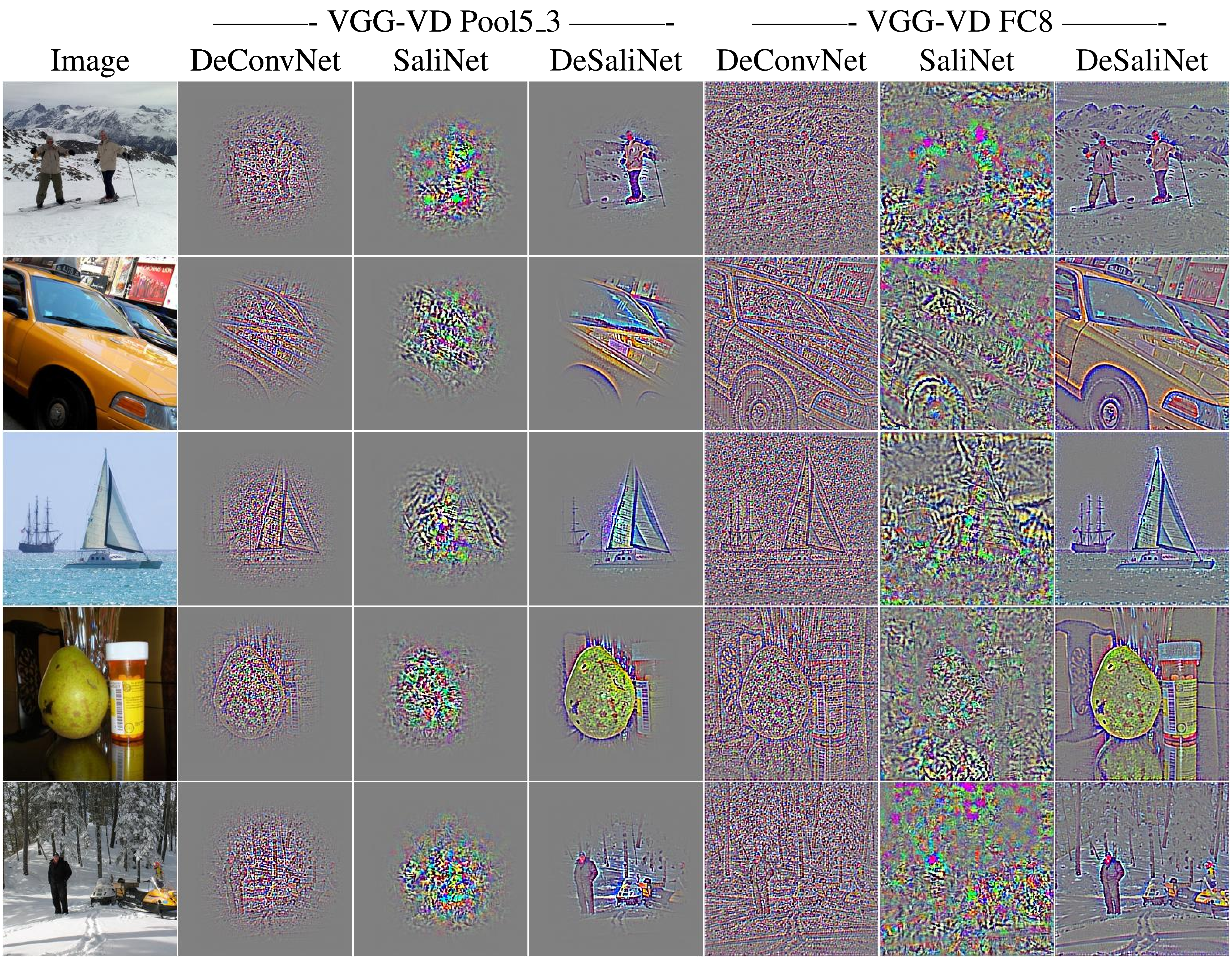


Three visualisation techniques

Modified backpropagation networks



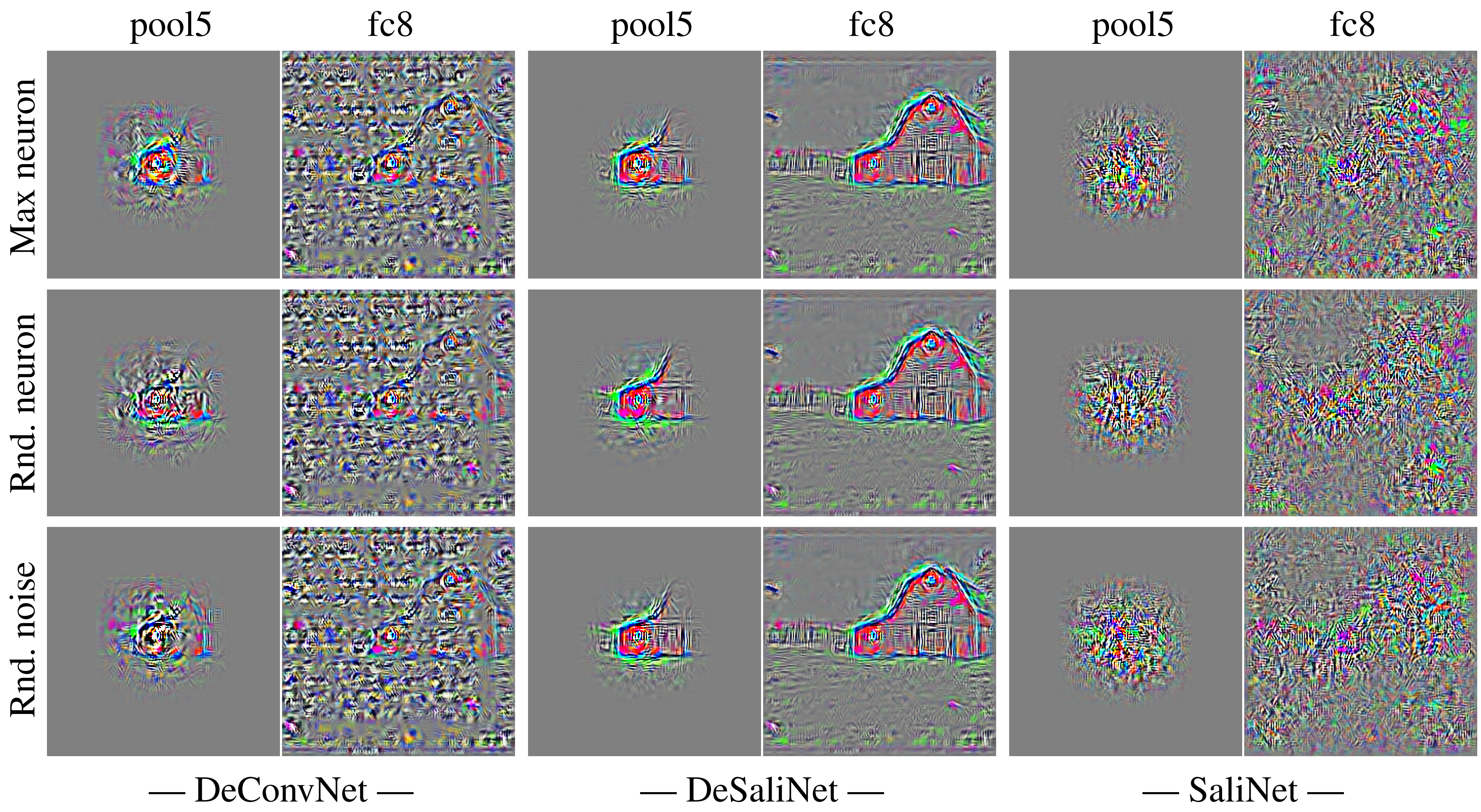
Results



VGG-VD Pool5_3
VGG-VD FC8
Image
DeConvNet
SaliNet
DeSaliNet
DeConvNet
SaliNet
DeSaliNet

Key limitation of DeConvNets and similar

They are largely *not* neuron selective



Good for saliency

Bad for studying individual neurons
(use inversion, act. max., etc. instead)

Understanding visual representations

Visualizing representations

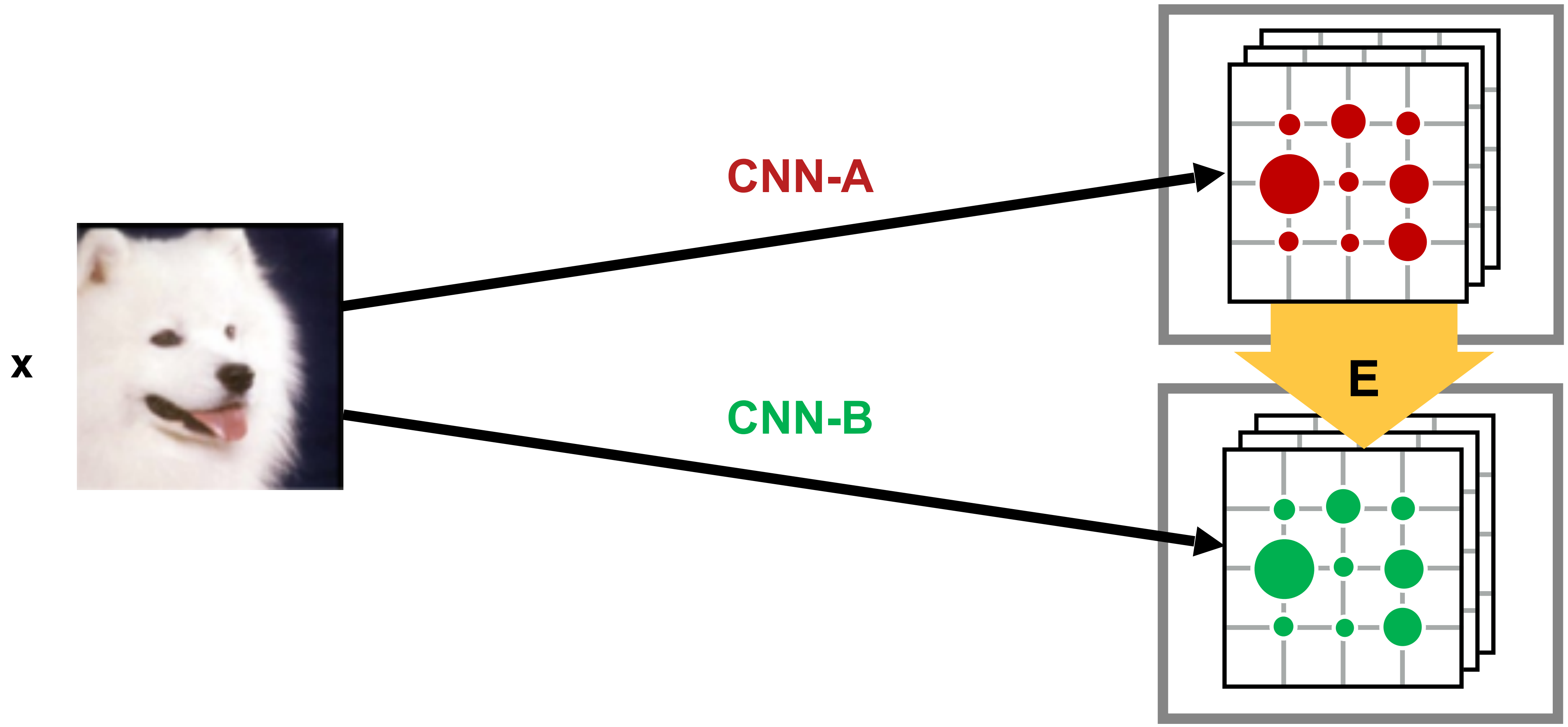
Backpropagation networks and “deconvolution”

Representations: equivalence & transformations

When are two representations the same?

Learning representations means that there is an endless number of them

Variants obtained by learning on different datasets, or different local optima

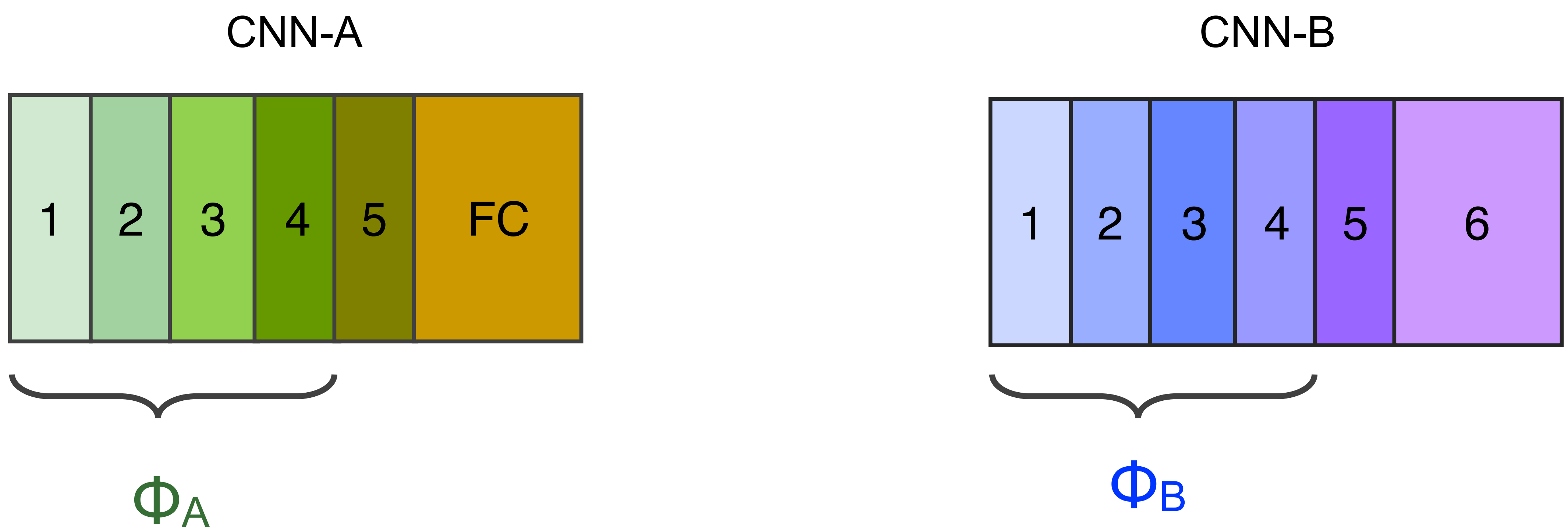


Equivalence

$$\Phi_B(\mathbf{x}) = E\ \Phi_A(\mathbf{x})$$

Equivalence

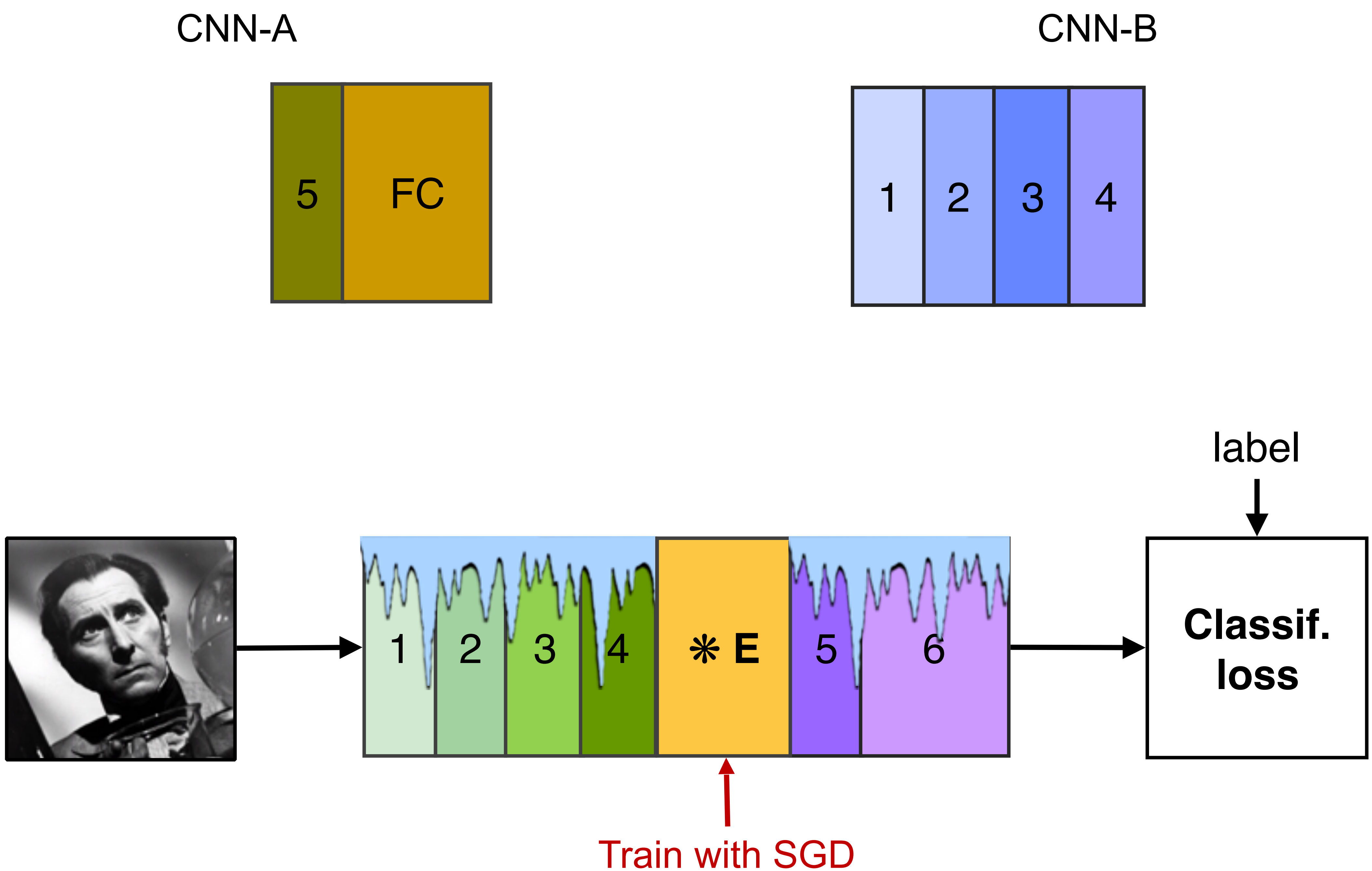
AlexNet, same training data, different parametrization:



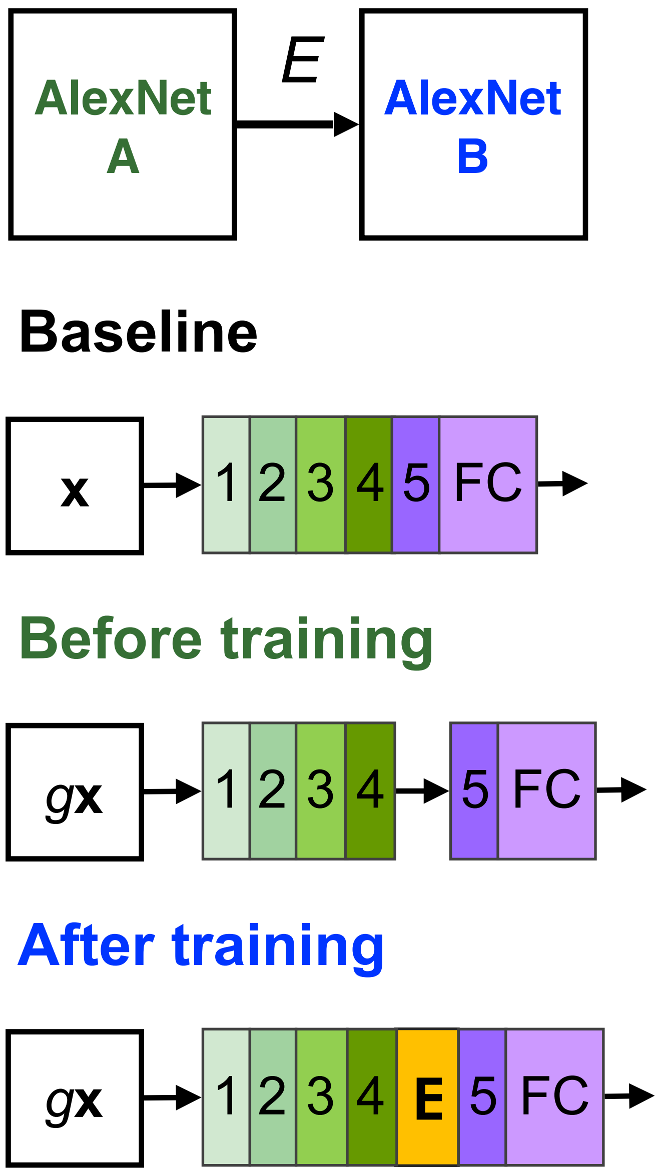
Are Φ_A and Φ_B equivalent ?

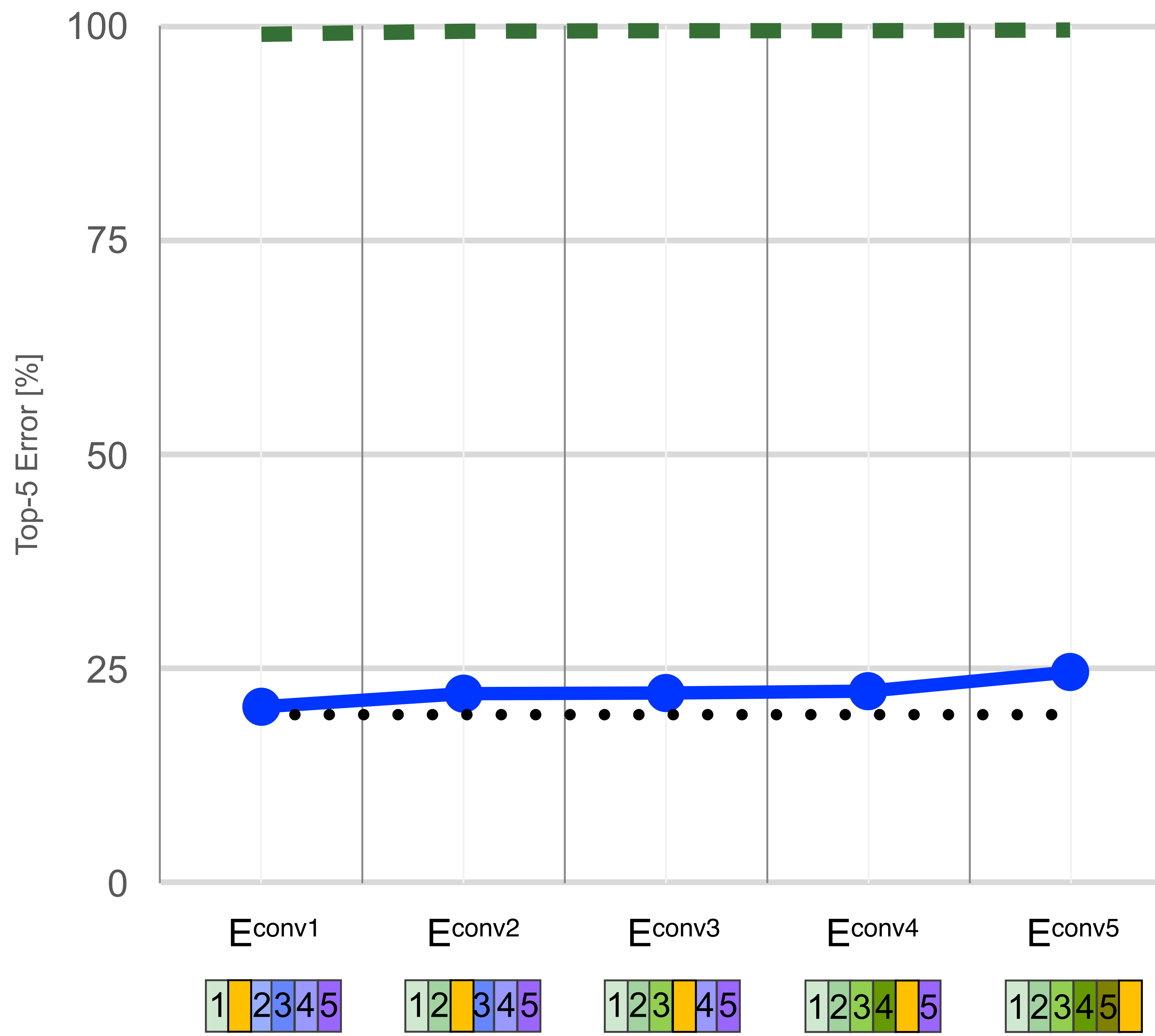
Equivalence

AlexNet, same training data, different parametrization:



Equivalence with different random seeds

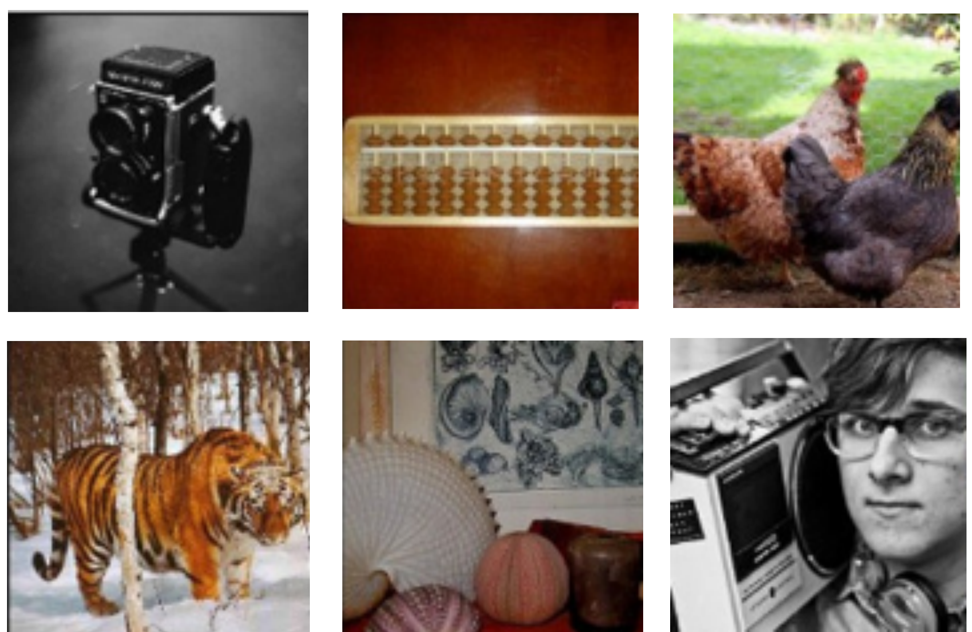




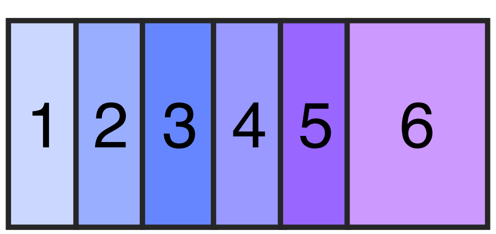
Equivalence of similar architecture

Train on two different datasets

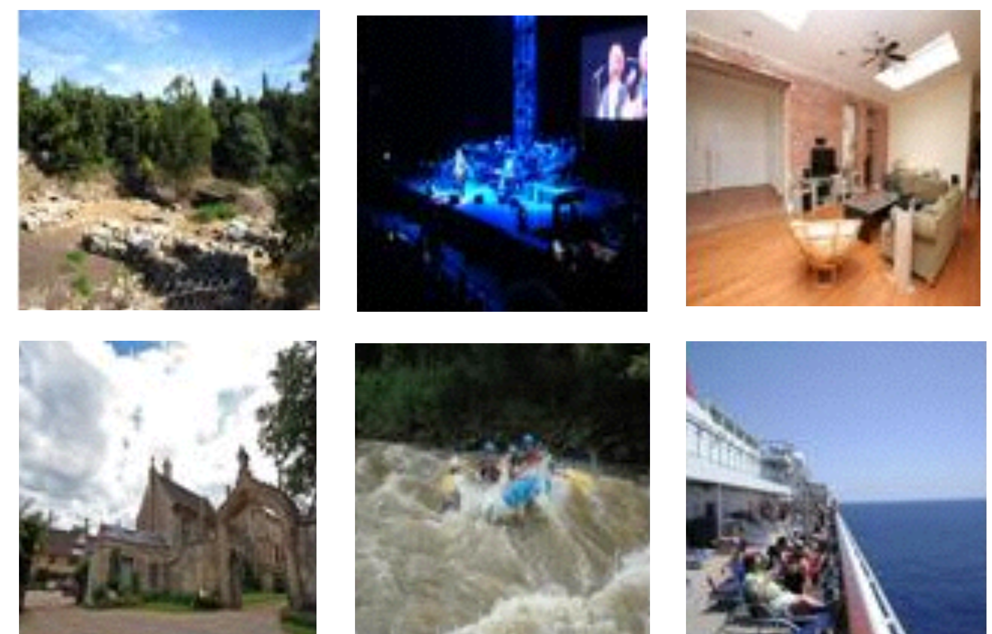
ILSVRC12 dataset



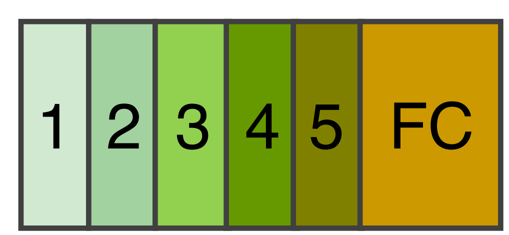
CNN-IMNET



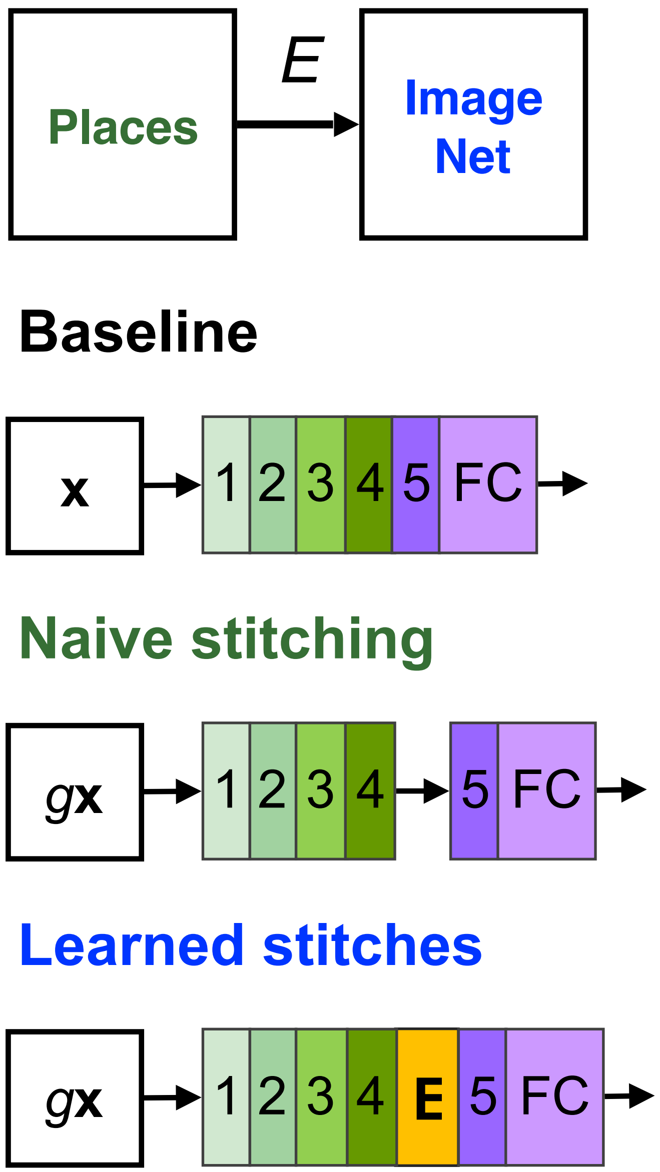
Places dataset

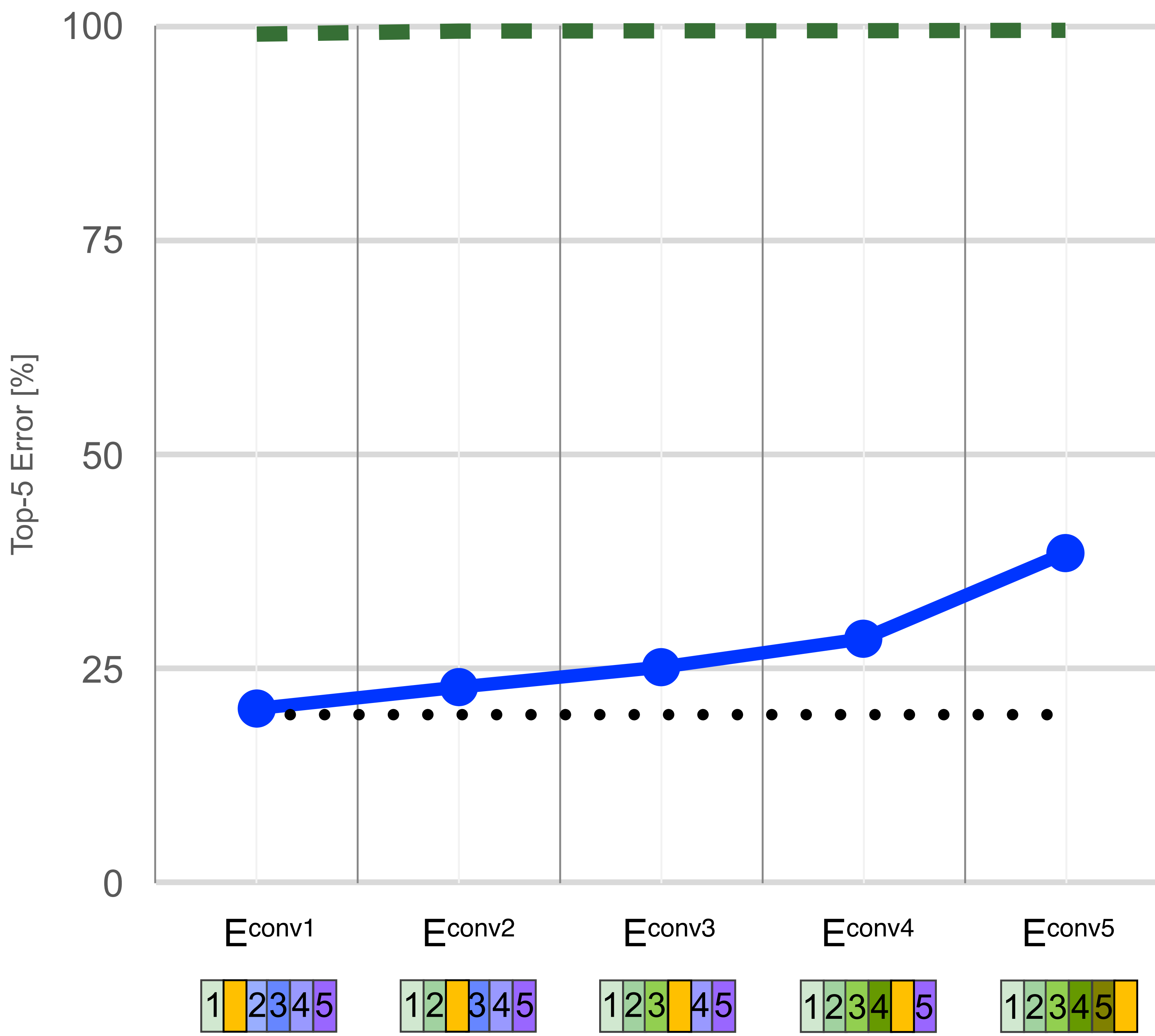


CNN-PLACES



Equivalence with different training data

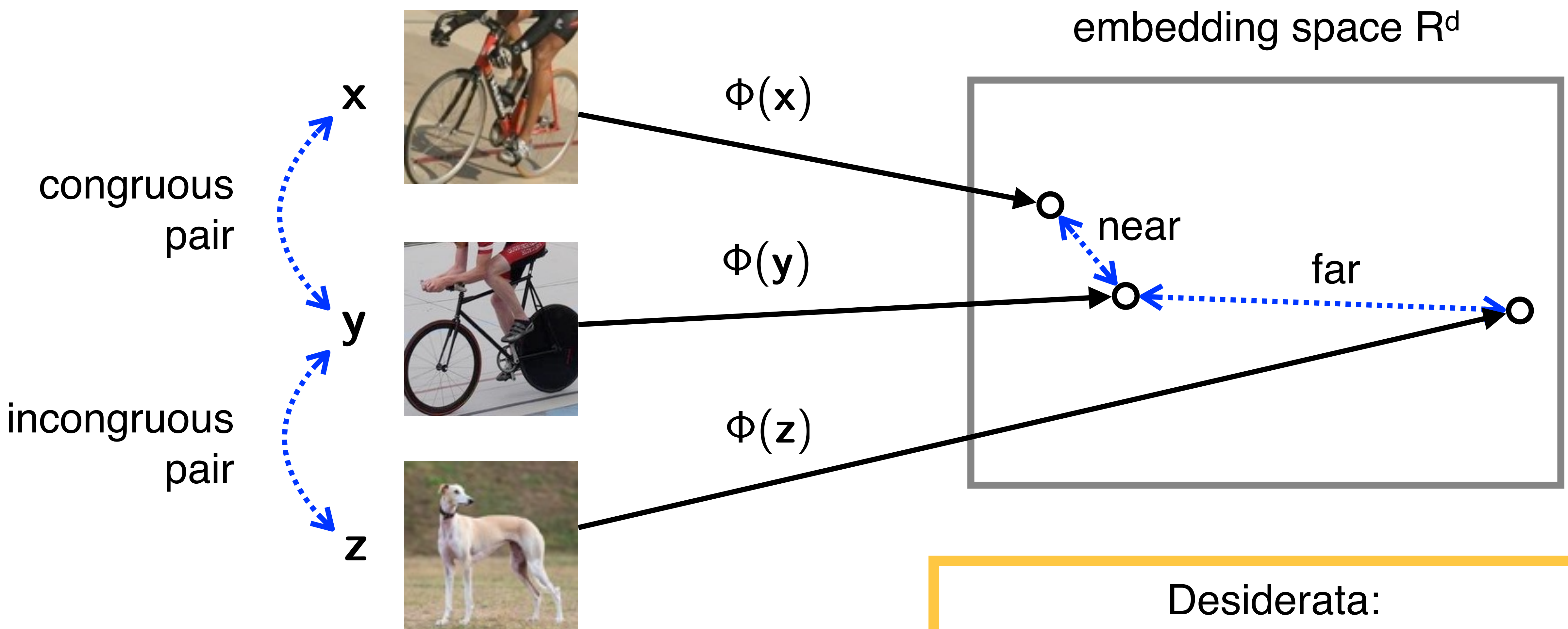




Meaningful representations

representation

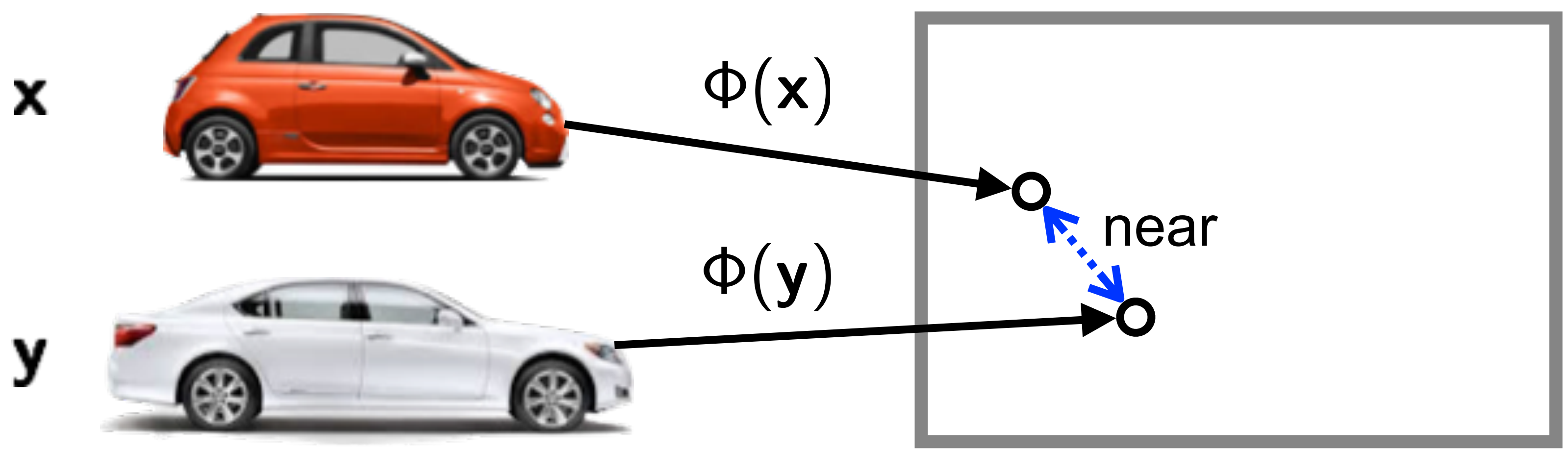
Semantic similarity → Vector similarity (distance)



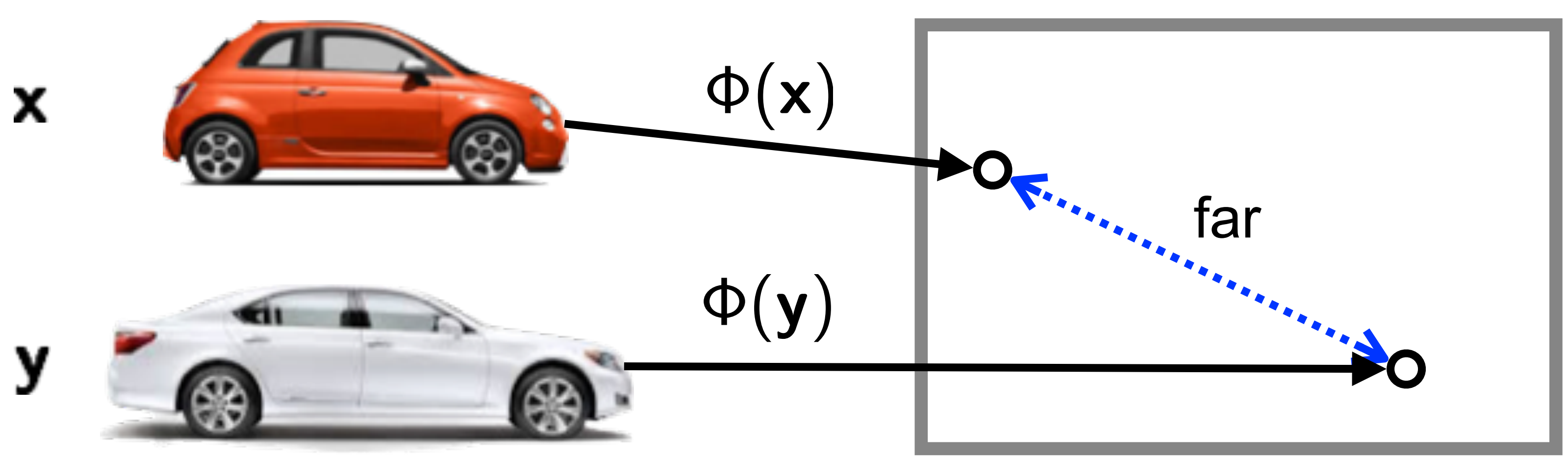
Desiderata:
invariance and **distinctiveness**

Invariance is task dependent

Are **x** and **y** the same category?



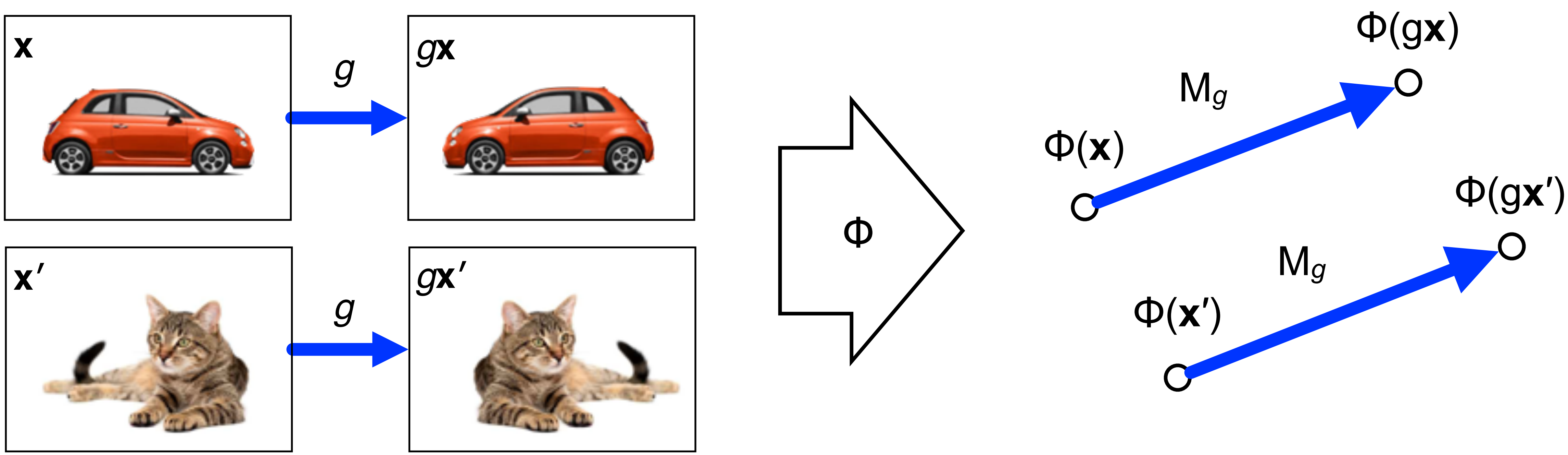
Are **x** and **y** the same colour?



Equivariance

image space

feature space

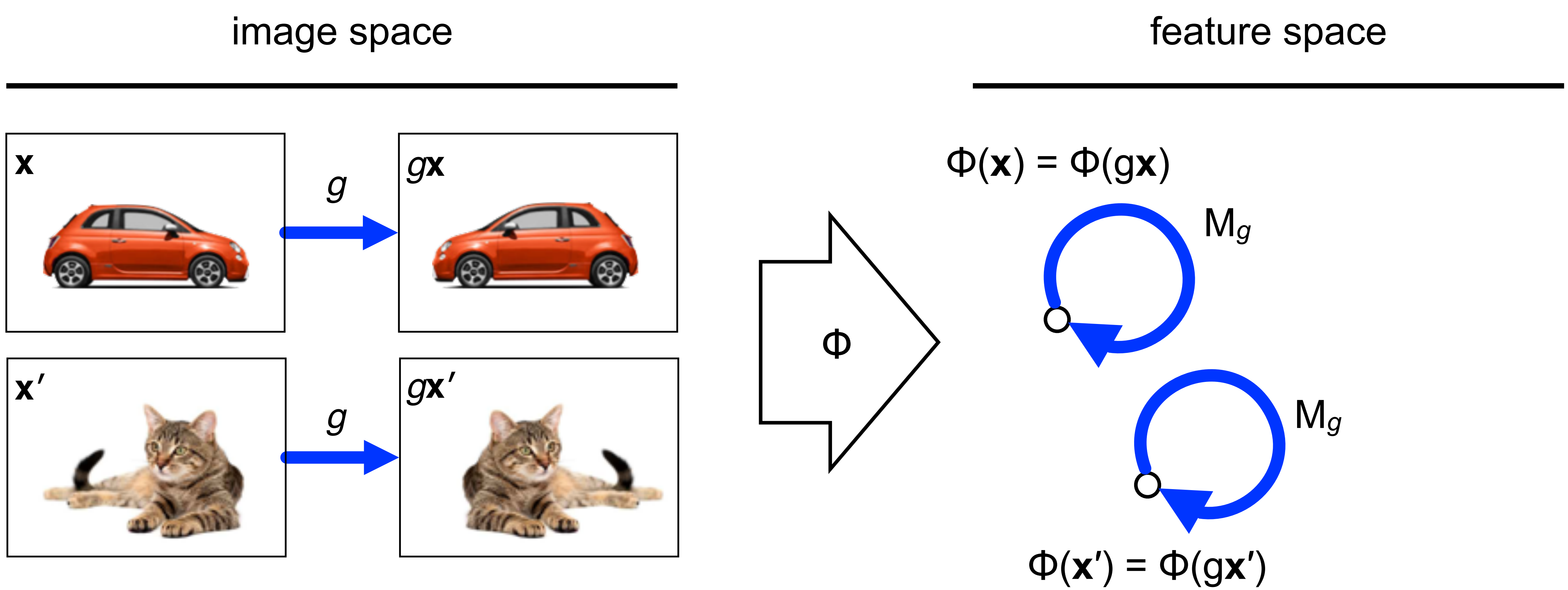


g is an image transformation

g is the corresponding feature transformation

$$\Phi(g\mathbf{x}) = M_g\, \Phi(\mathbf{x})$$

Invariance

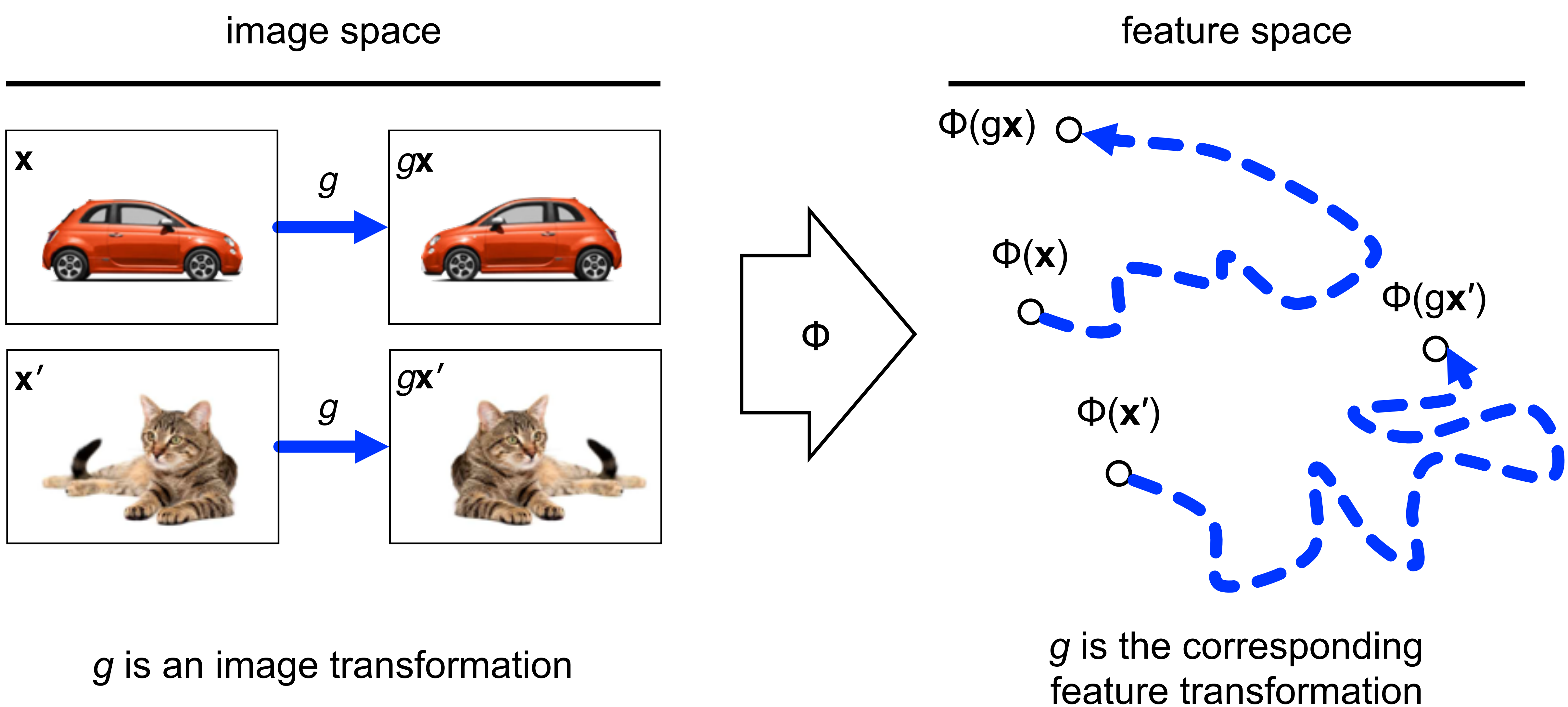


g is an image transformation

g is the corresponding feature transformation

the map M_g is the identity

No equivariance



the map M_g does not
exists (or is intractable)

Representations and transformations

image space

g is an image transformation

feature space

Equivariance
There is a (simple) M_g

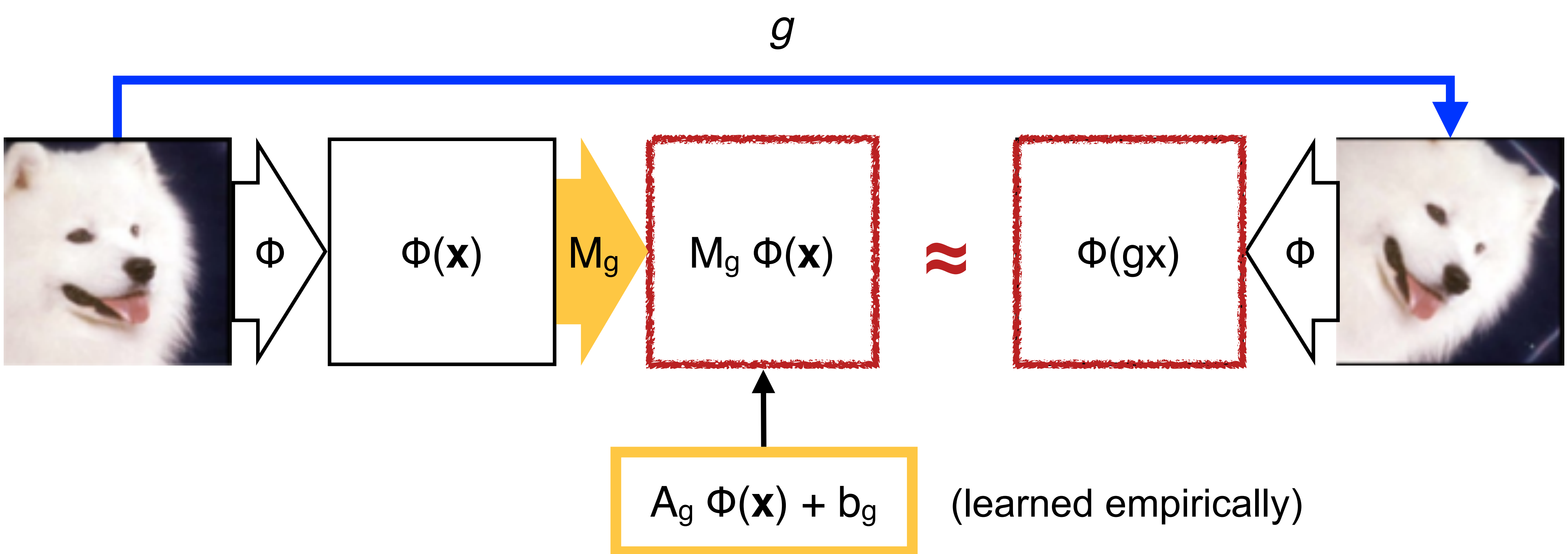
Invariance
M_g is the identity

Neither
There is no (simple) M_g

$$\Phi(g\mathbf{x}) = M_g\, \Phi(\mathbf{x})$$

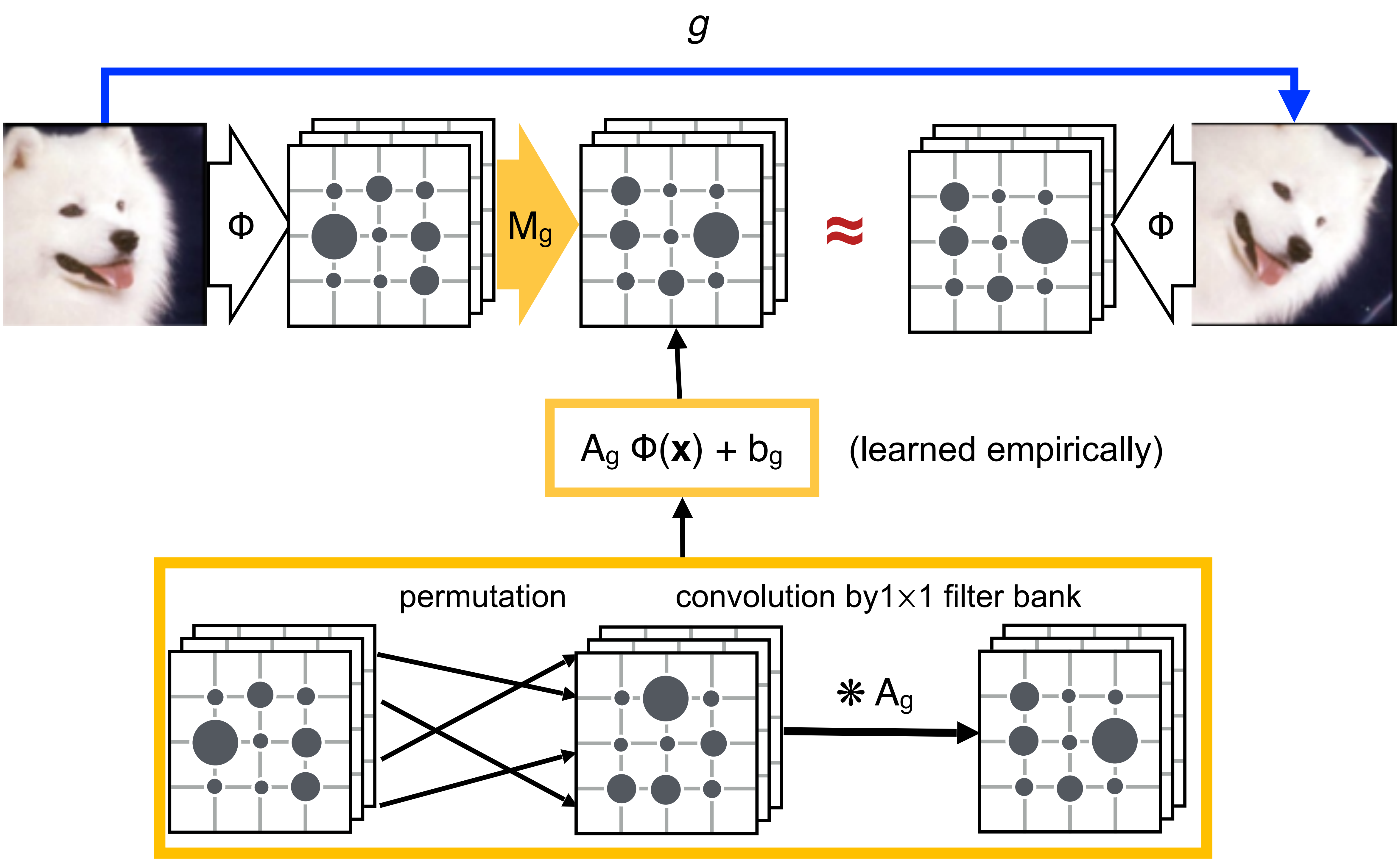
An empirical test of equivariance

Regularized linear regression



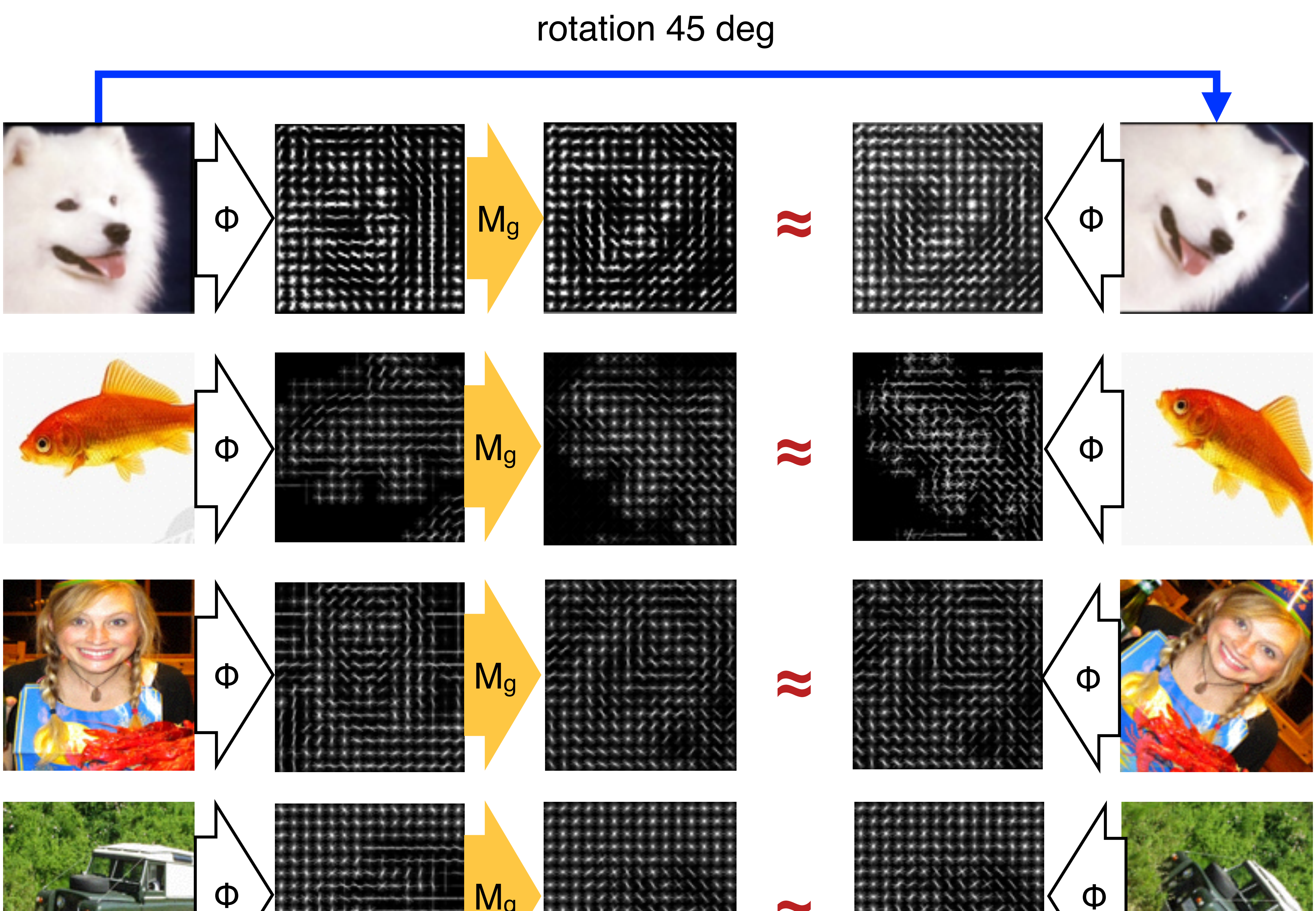
An empirical test of equivariance

Regularized linear regression



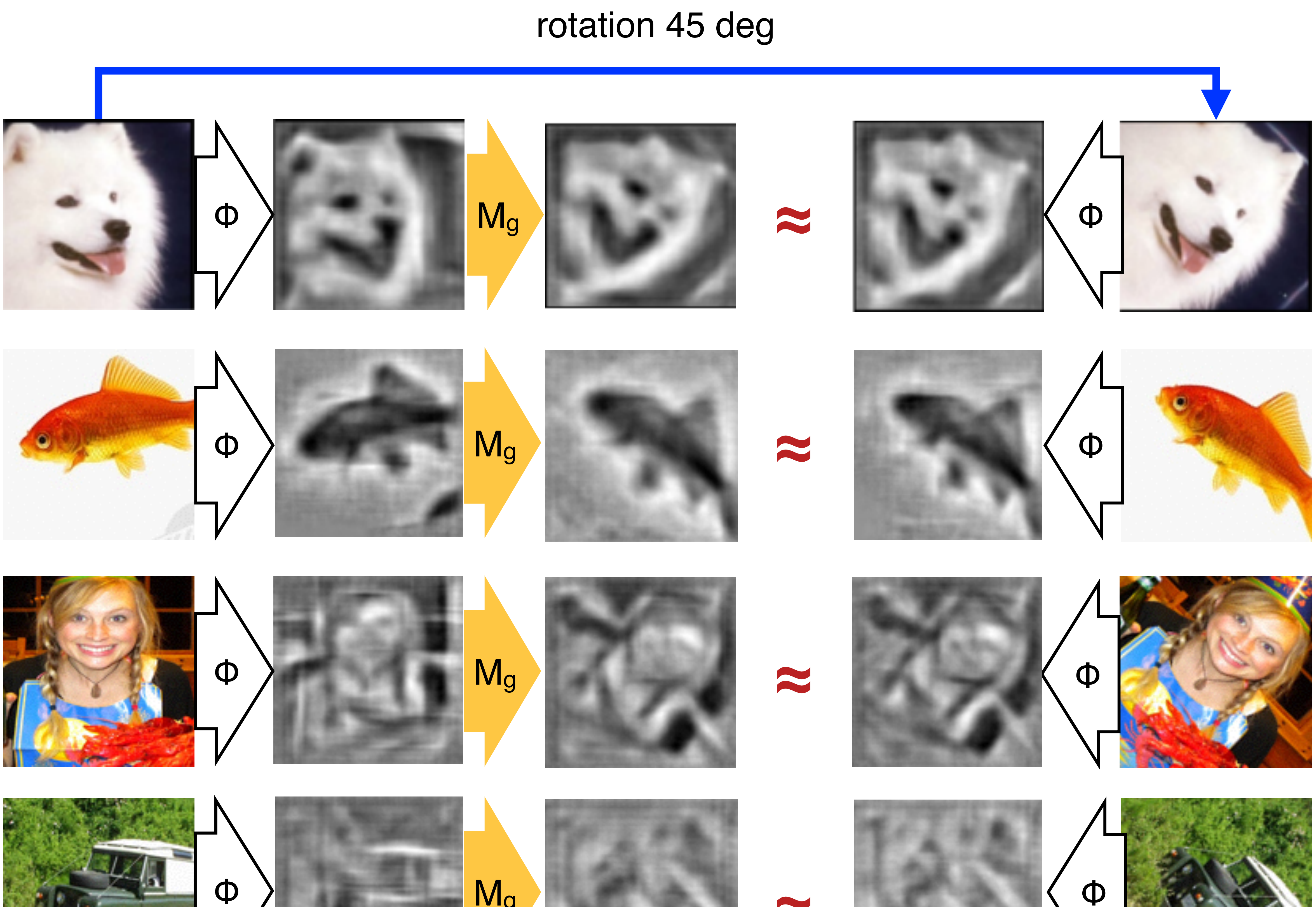
An empirical test of equivariance

HOG features

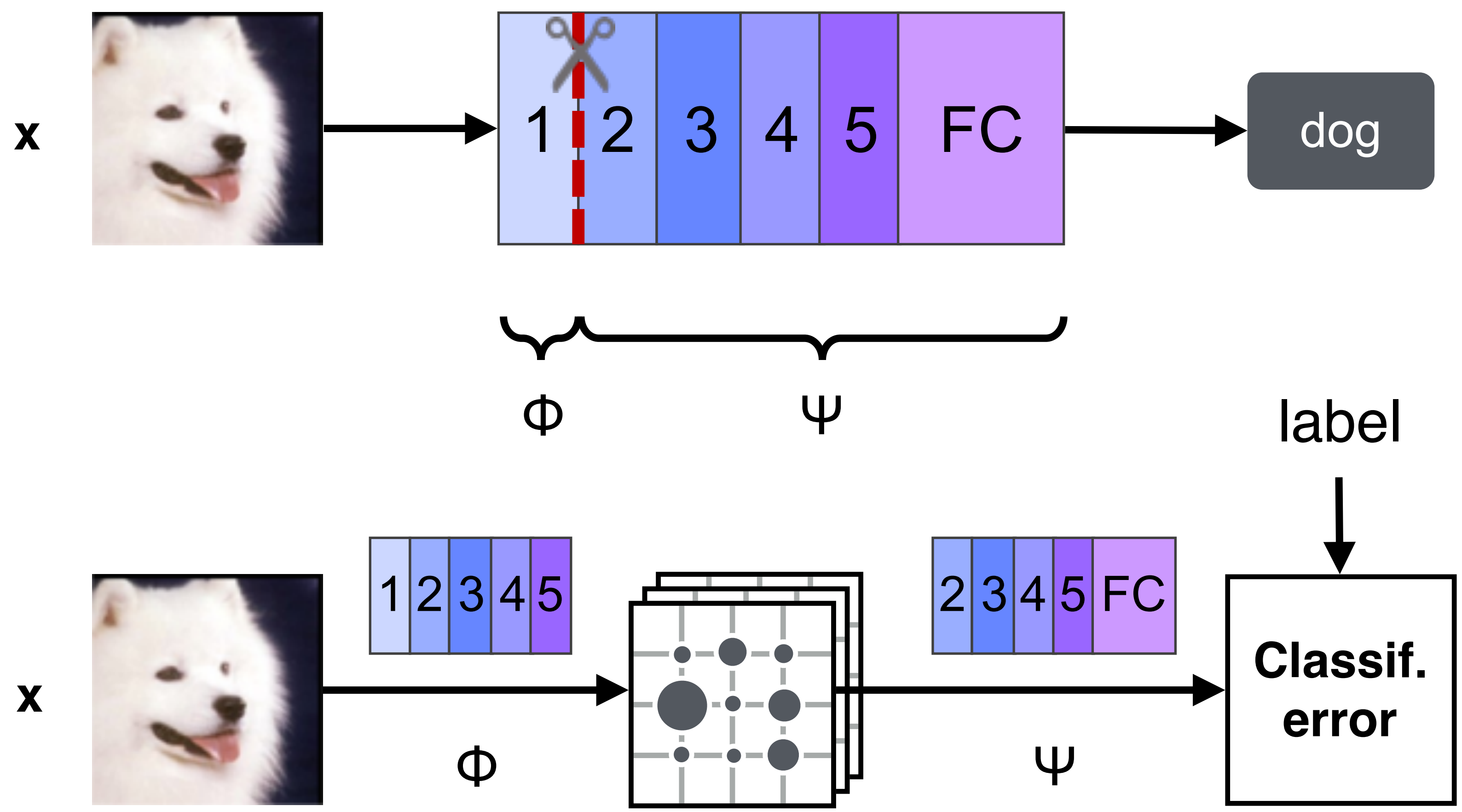


An empirical test of equivariance

HOG features



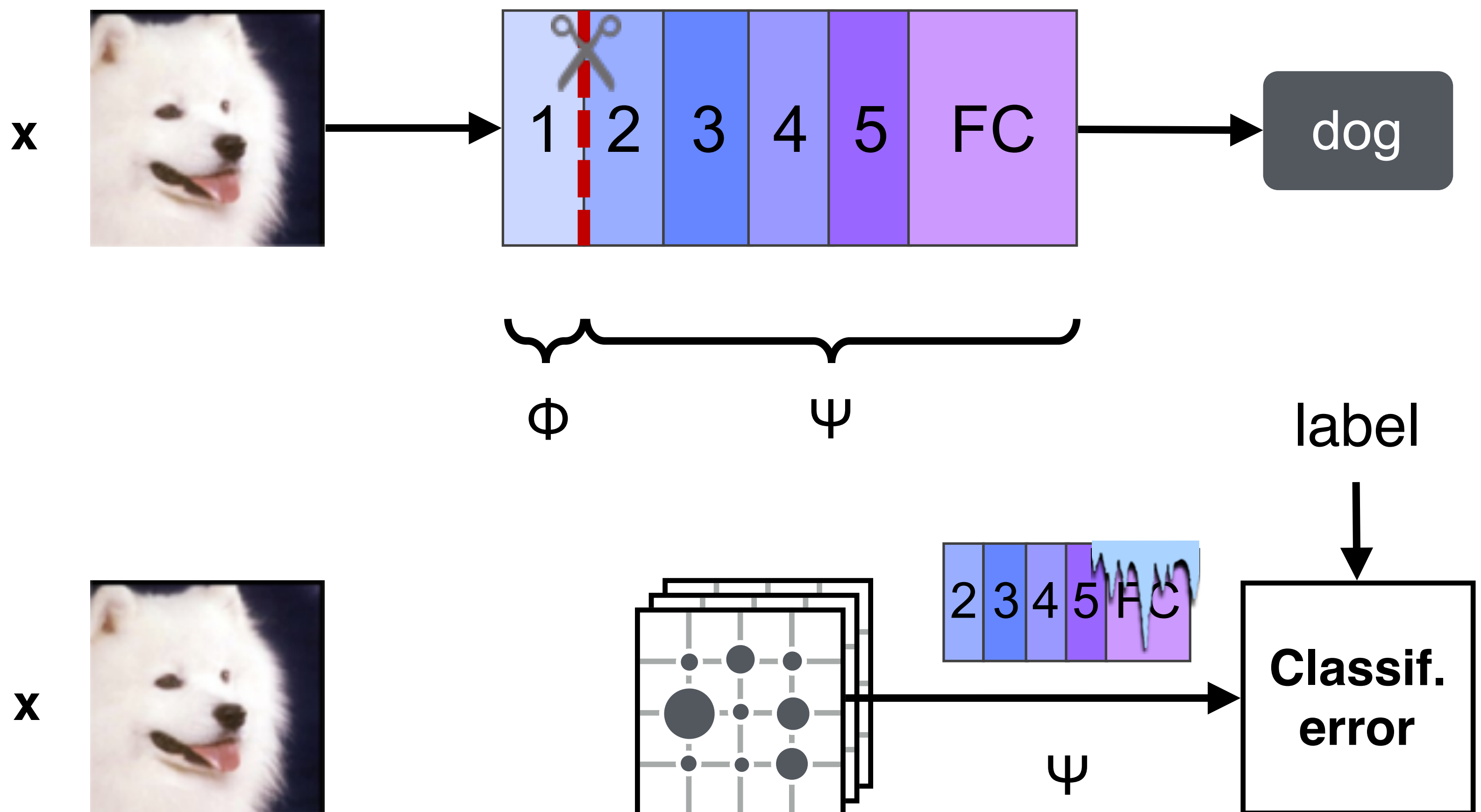
CNN: a sequence of representations

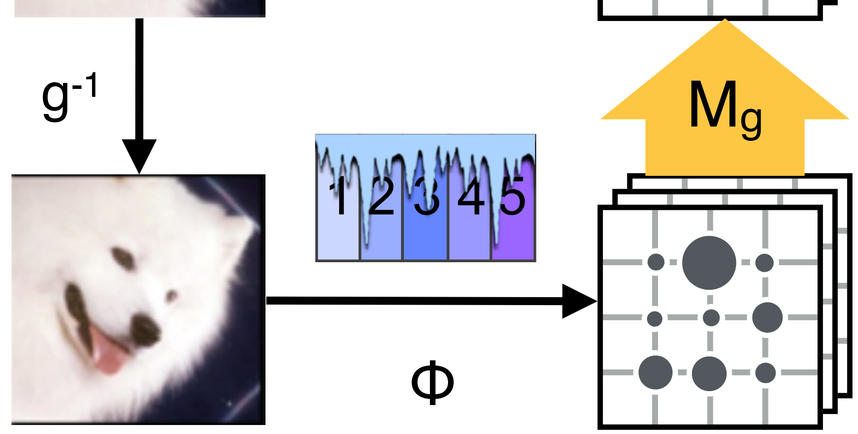


We run the same analysis on a typical CNN architecture

- AlexNet [Krizevsky et al. 12]
- 5 convolutional layers + fully-connected layers
- Trained on ImageNet ILSVRC

A discriminative goal to learn equivariance

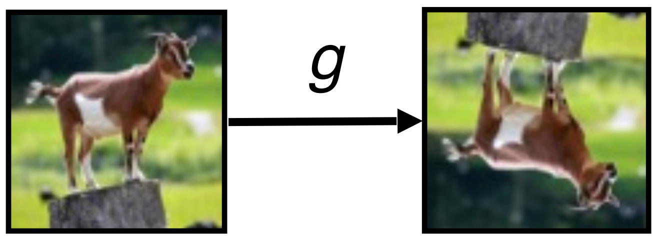




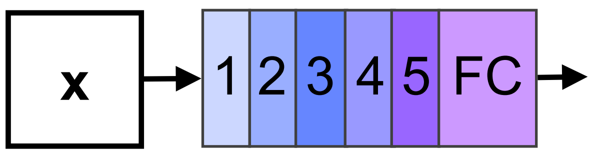
M_g is learned empirically

All the other layers (representation and classifier) are frozen

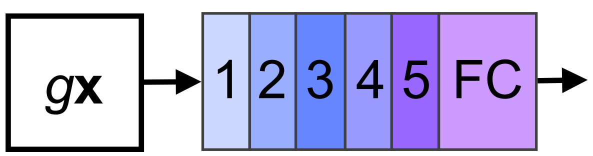
Vertical flips



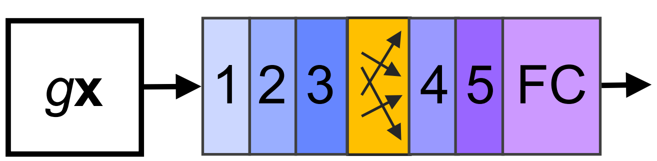
Uncompensated, no TF



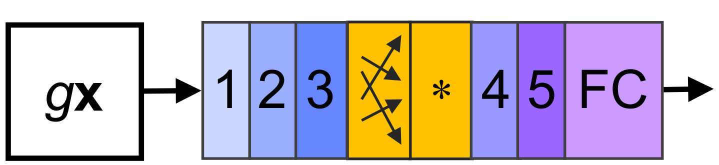
Uncompensated, TF

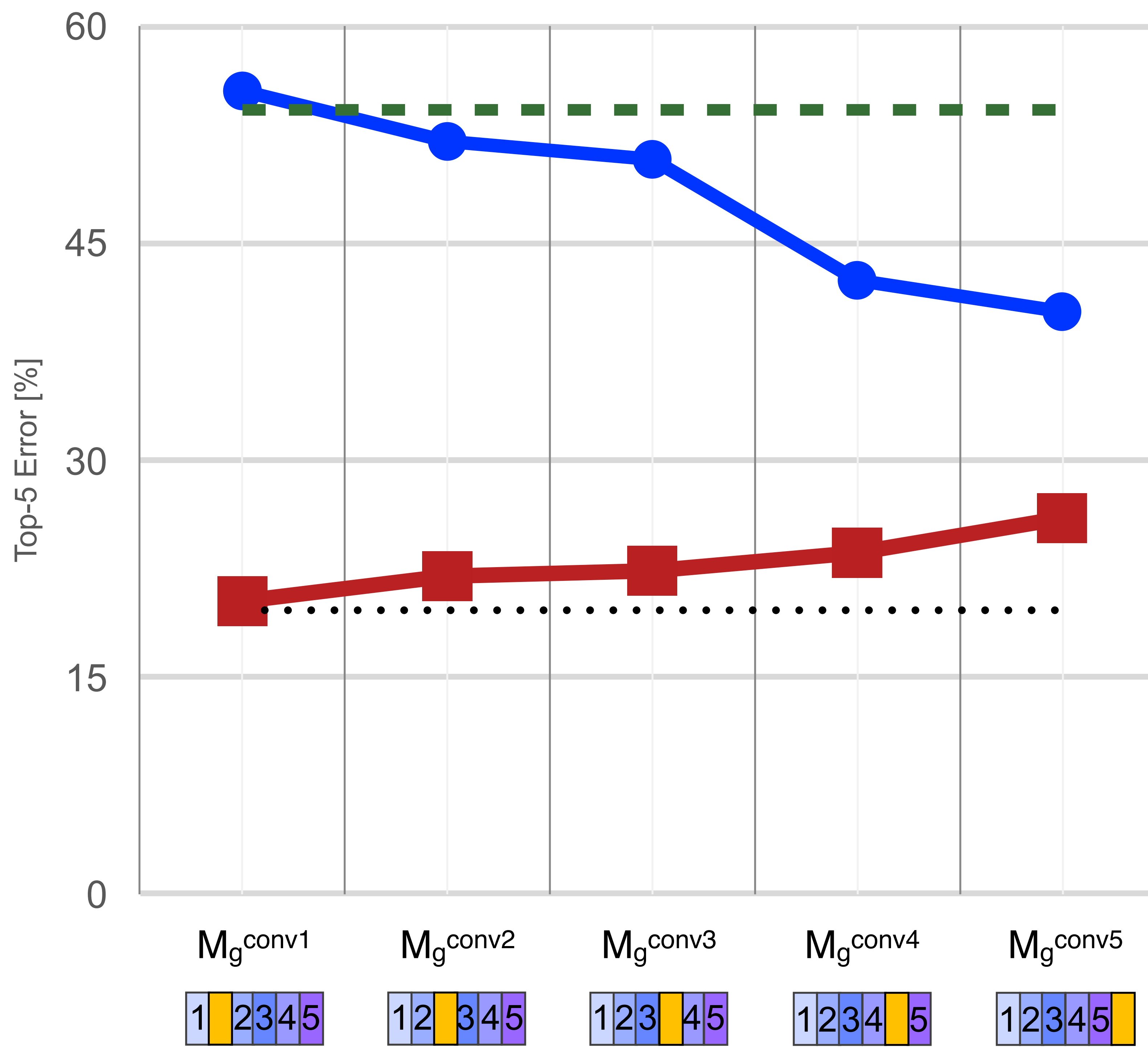


Compensated TF w perm.



Compensated TF, learned





Summary

Modern CNNs are still "new" technology

- Expect bigger and meaner CNNs to further improve performance
- CNNs are more than big balls of parameters
- We do not really understand what they do

Understanding deep nets

- What do deep networks do?
- Are there computational / statistical principles to be learned?
- How much do deep nets learn about the visual world?

Possible angles

- Visualisations
- Probing statistical properties