

Lecture 21: Visualizing Models and Generating Images

Reminder: A5

Recurrent networks, attention, Transformers

We released a minor revision to the starter code today;
only fixes typos, no functional changes

Autograder will be up today

Due on **Tuesday 4/12**, 11:59pm ET

A3 Grades

Released last night

Post regrade requests on Piazza until Monday 4/11

Last Time: Generative Models

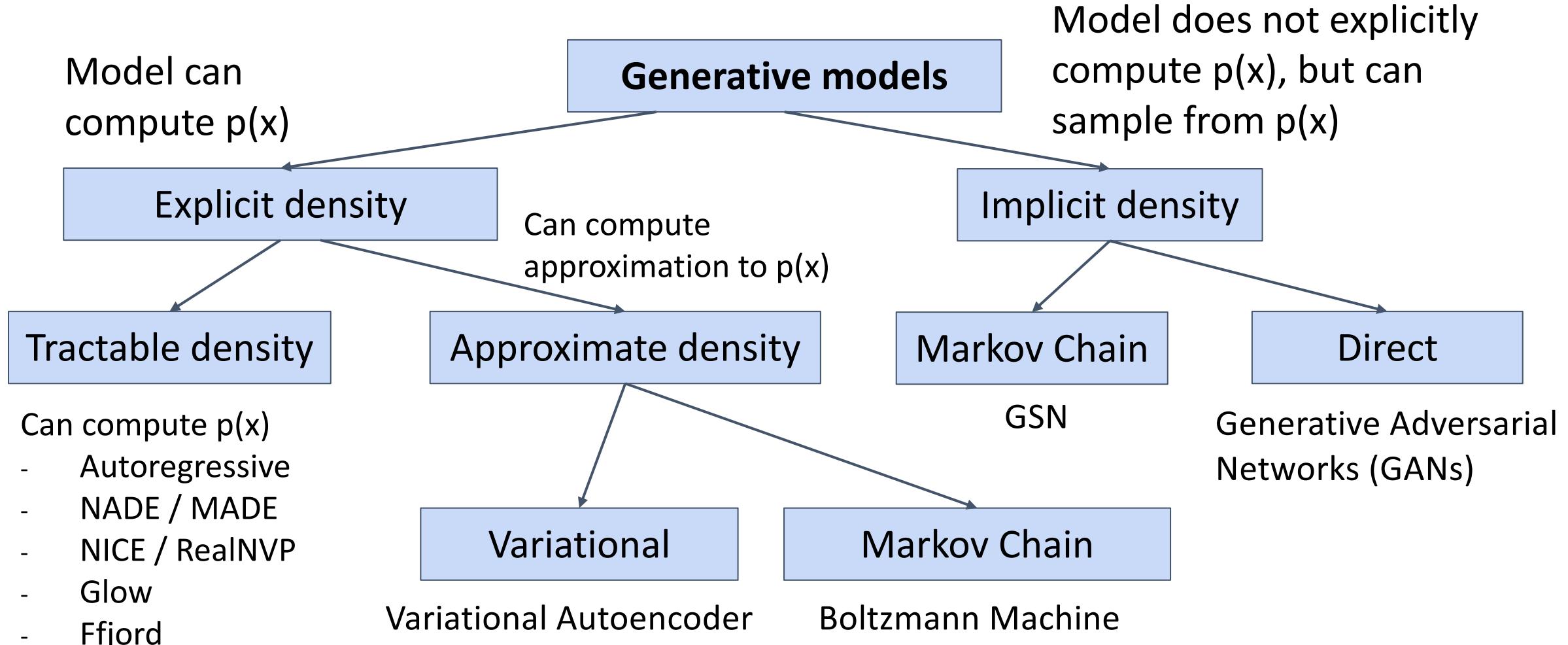


Figure adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Last Time: Generative Models

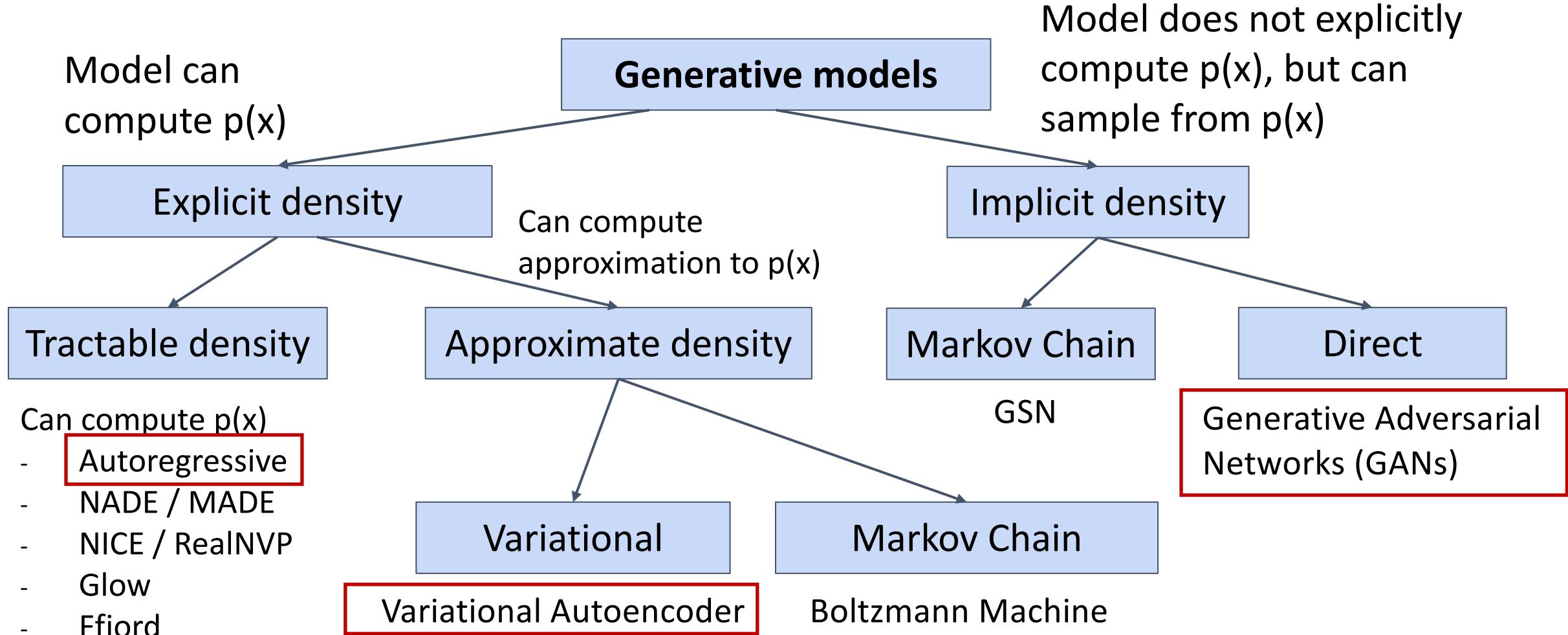
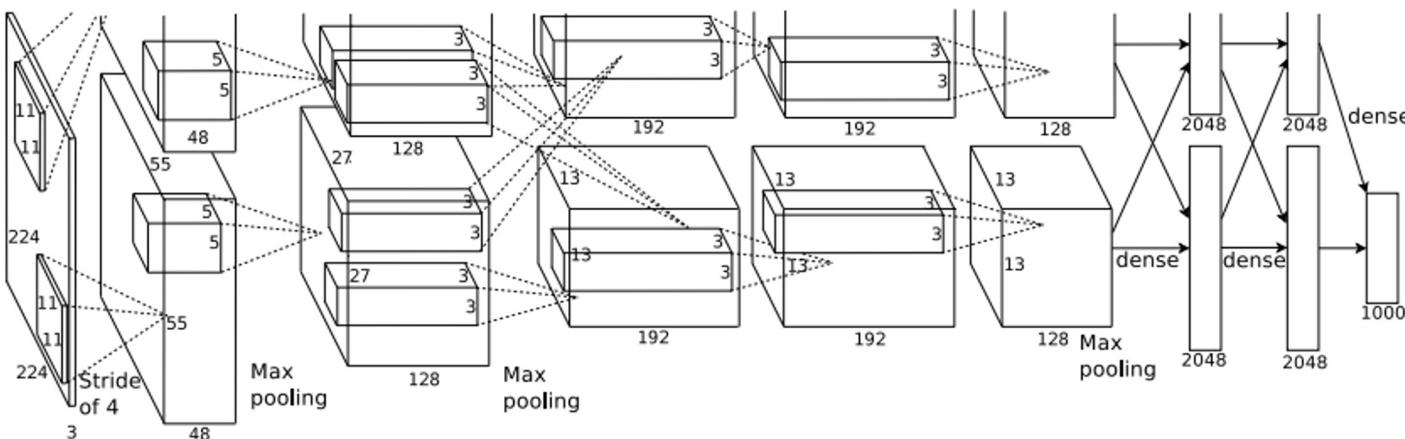


Figure adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Today: Visualizing Networks and Generating Images

What's going on inside Convolutional Networks?

This image is CC0 public domain

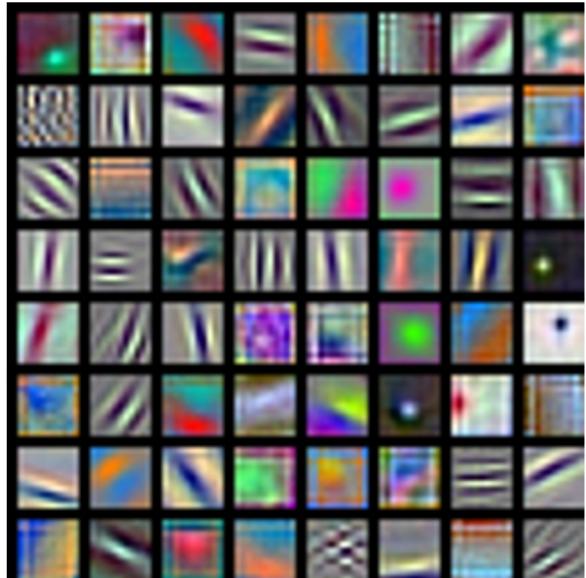


Input Image:
3 x 224 x 224

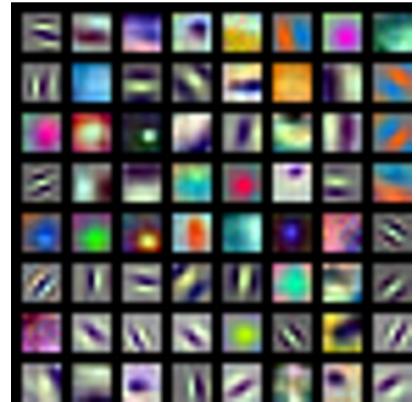
What are the intermediate features looking for?

Class Scores:
1000 numbers

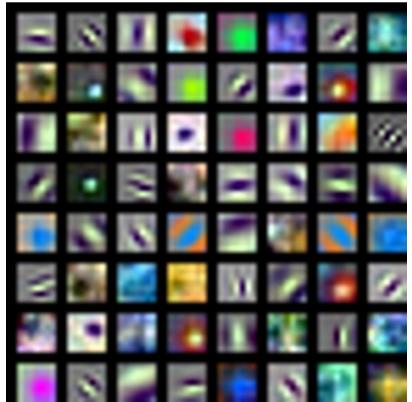
First Layer: Visualize Filters



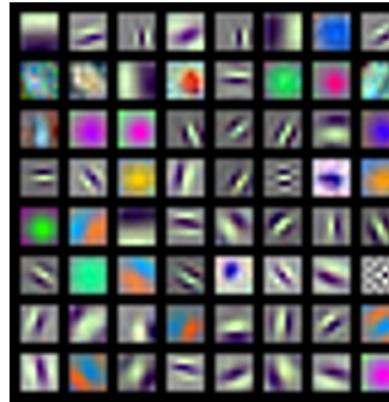
AlexNet:
 $64 \times 3 \times 11 \times 11$



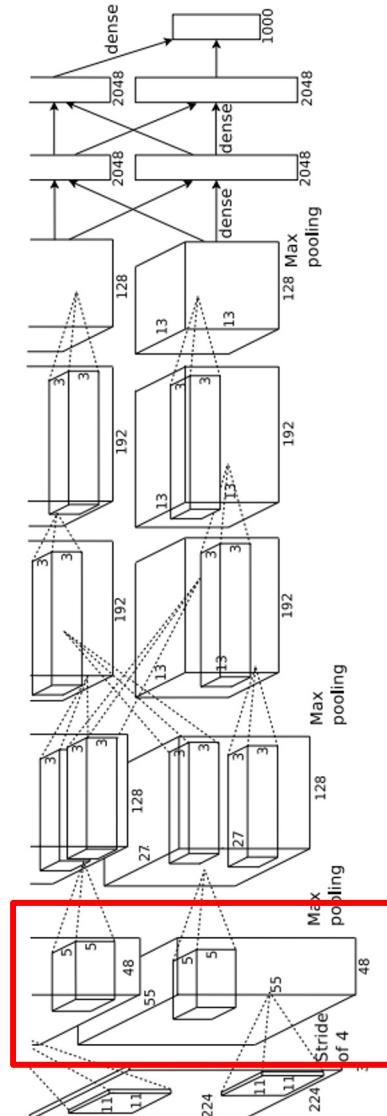
ResNet-18:
 $64 \times 3 \times 7 \times 7$



ResNet-101:
 $64 \times 3 \times 7 \times 7$



DenseNet-121:
 $64 \times 3 \times 7 \times 7$



Krizhevsky, "One weird trick for parallelizing convolutional neural networks", arXiv 2014

He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

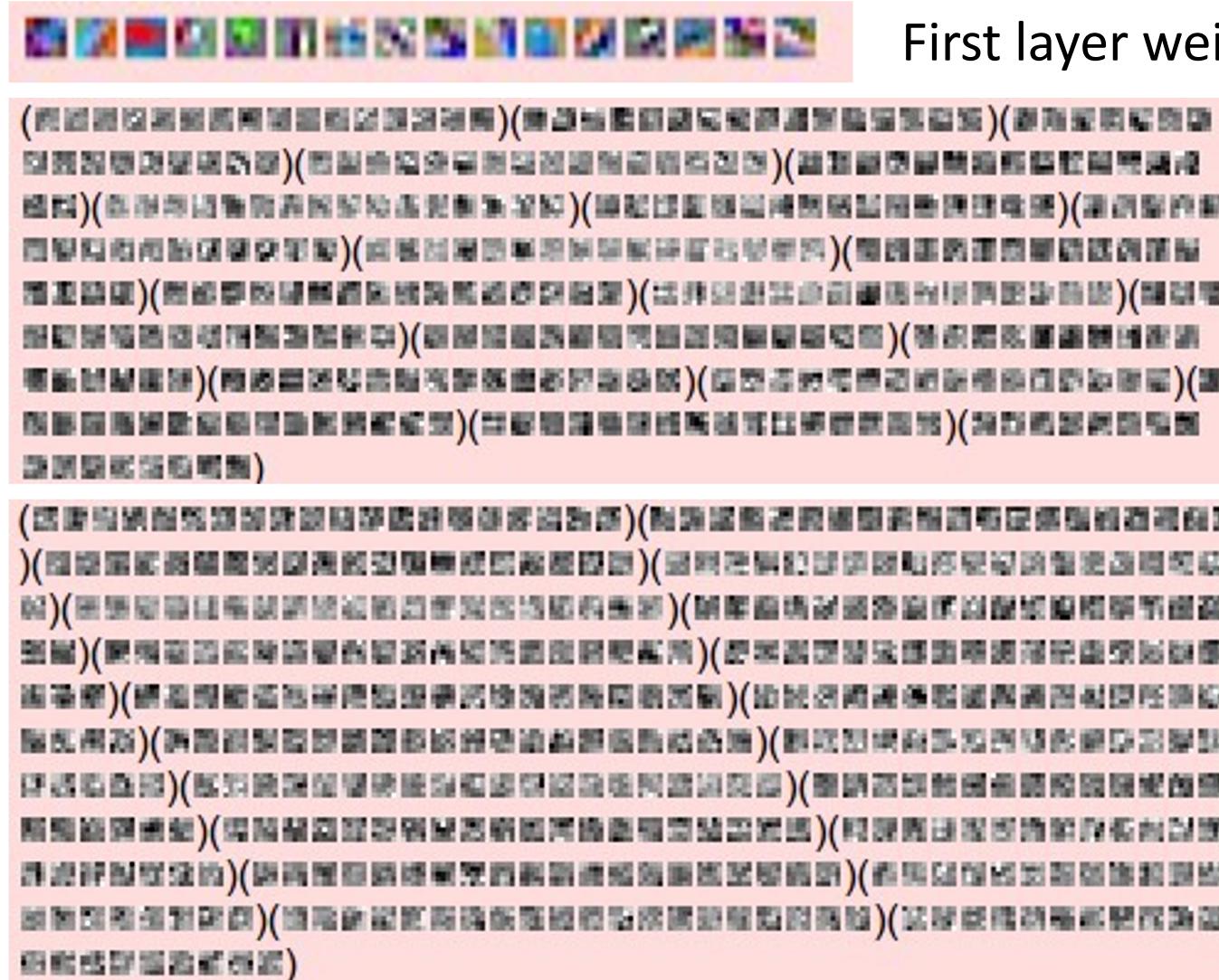
Huang et al, "Densely Connected Convolutional Networks", CVPR 2017

Higher Layers: Visualize Filters

We can visualize filters at higher layers, but not that interesting

Source: ConvNetJS
CIFAR-10 example

<https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>



First layer weights: $16 \times 3 \times 7 \times 7$

Second layer weights:
 $20 \times 16 \times 7 \times 7$

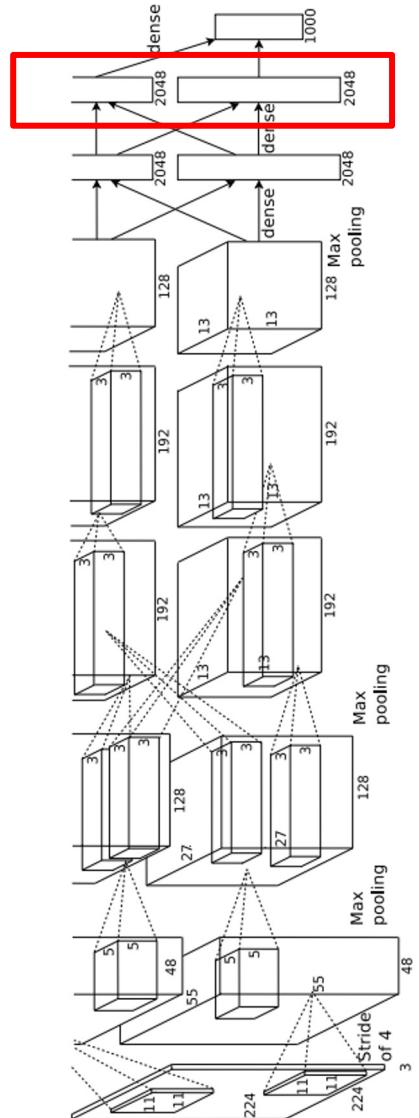
Third layer weights:
 $20 \times 20 \times 7 \times 7$

Last Layer

FC7 layer

4096-dimensional feature vector for an image
(layer immediately before the classifier)

Run the network on many images, collect the feature vectors

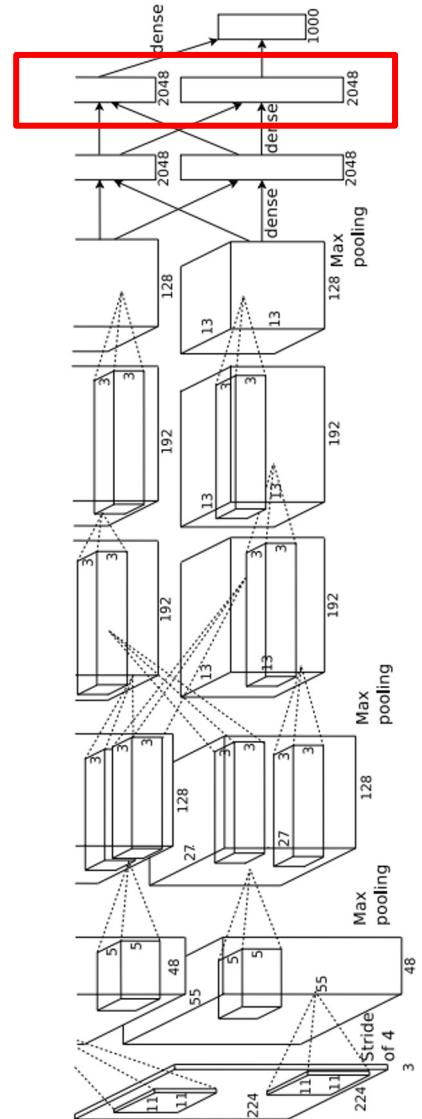
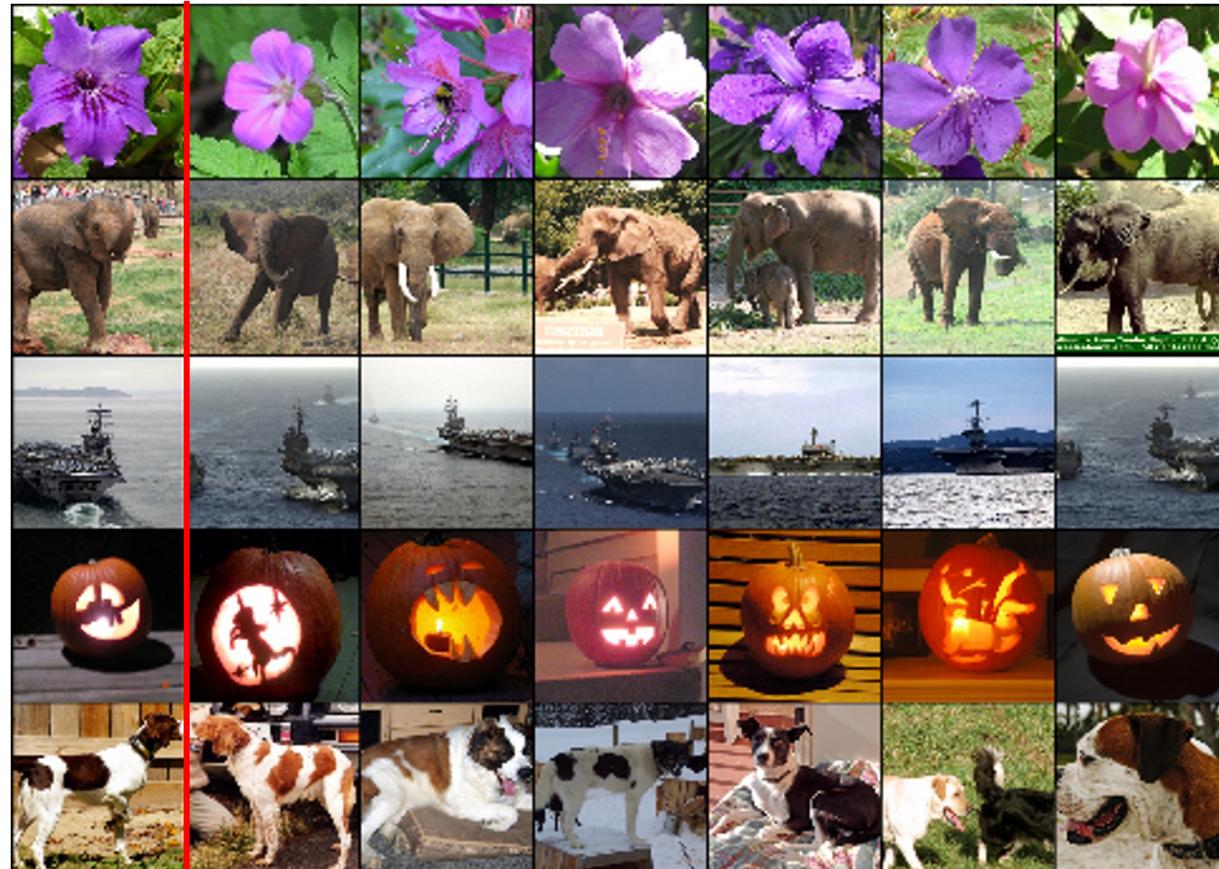


Last Layer: Nearest Neighbors

Recall: Nearest neighbors in pixel space



Test
image L2 Nearest neighbors in feature space



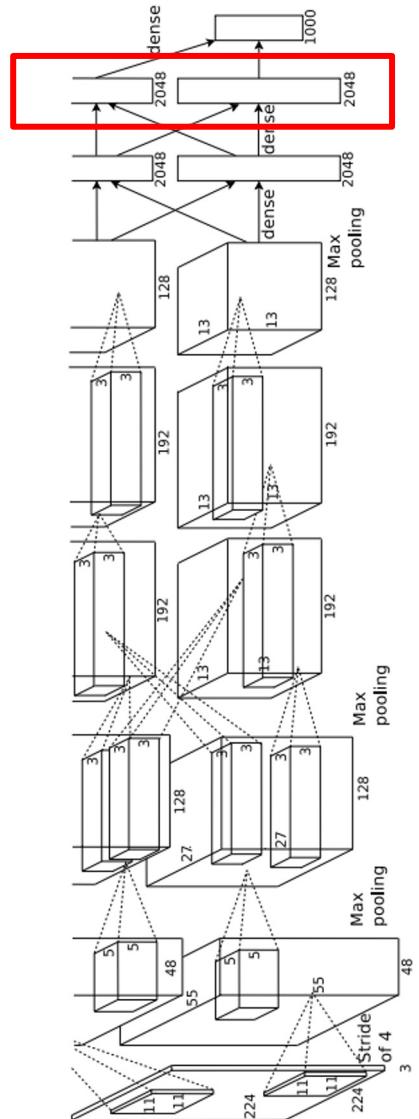
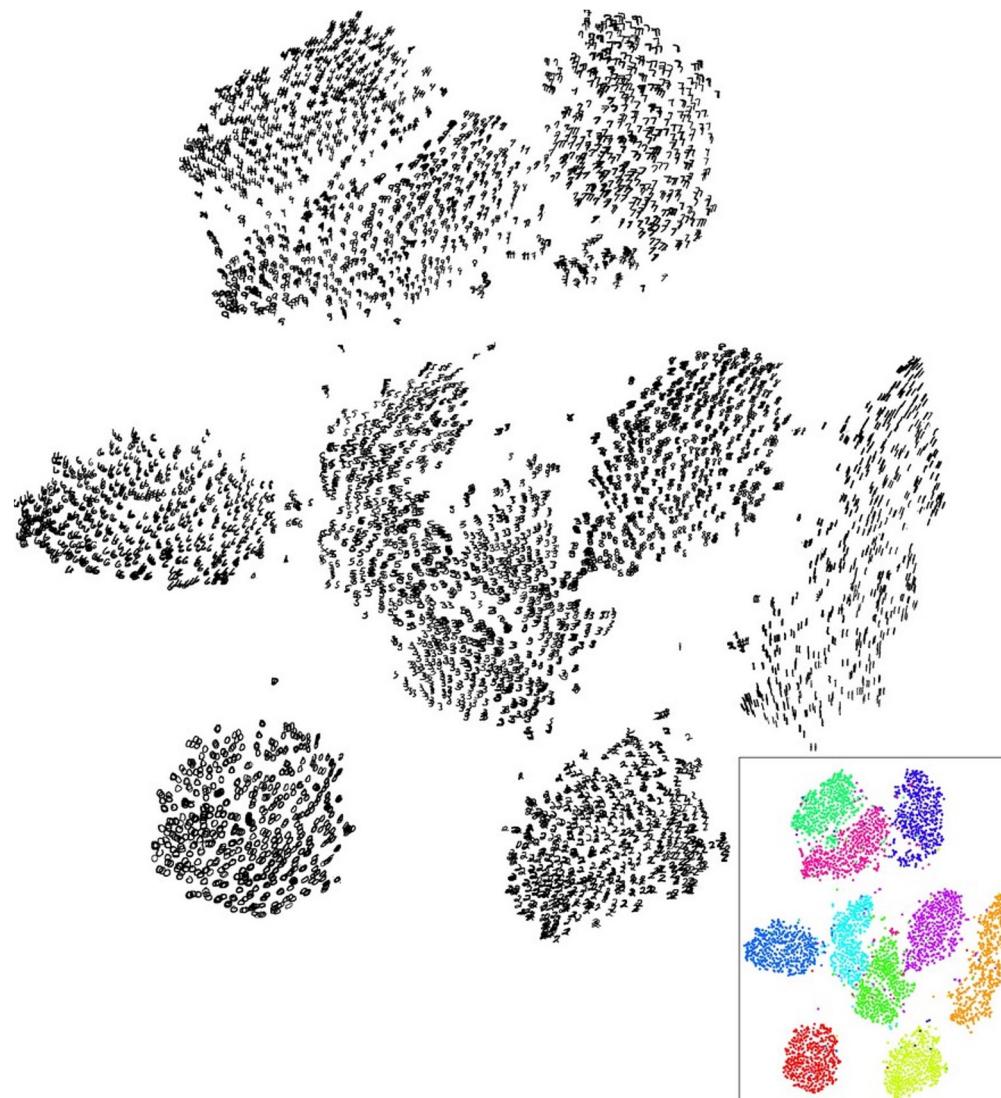
Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NeurIPS 2012.
Figures reproduced with permission.

Last Layer: Dimensionality Reduction

Visualize the “space” of FC7
feature vectors by reducing
dimensionality of vectors from
4096 to 2 dimensions

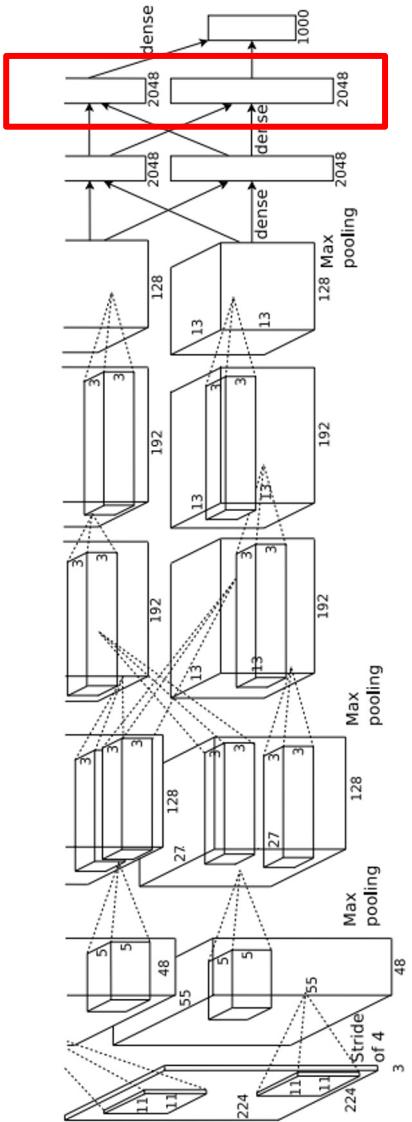
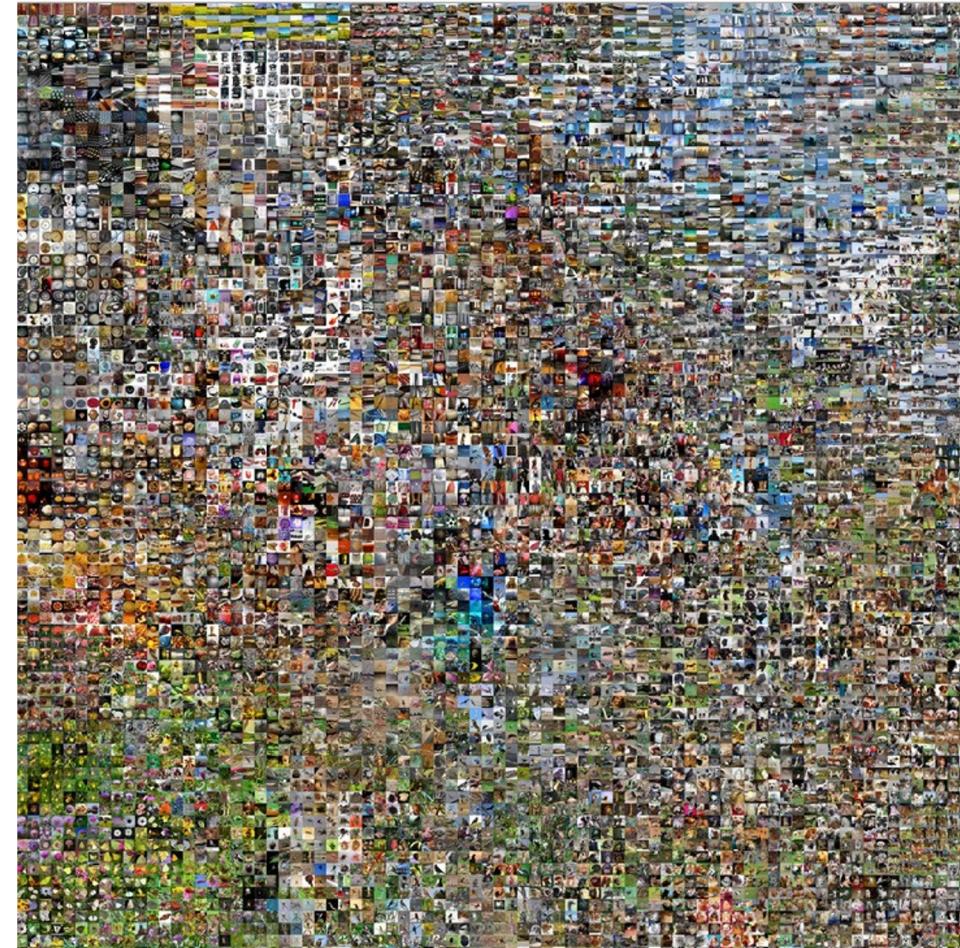
Simple algorithm: Principal
Component Analysis (PCA)

More complex: t-SNE



Van der Maaten and Hinton, “Visualizing Data using t-SNE”, JMLR 2008
Figure copyright Laurens van der Maaten and Geoff Hinton, 2008. Reproduced with permission.

Last Layer: Dimensionality Reduction

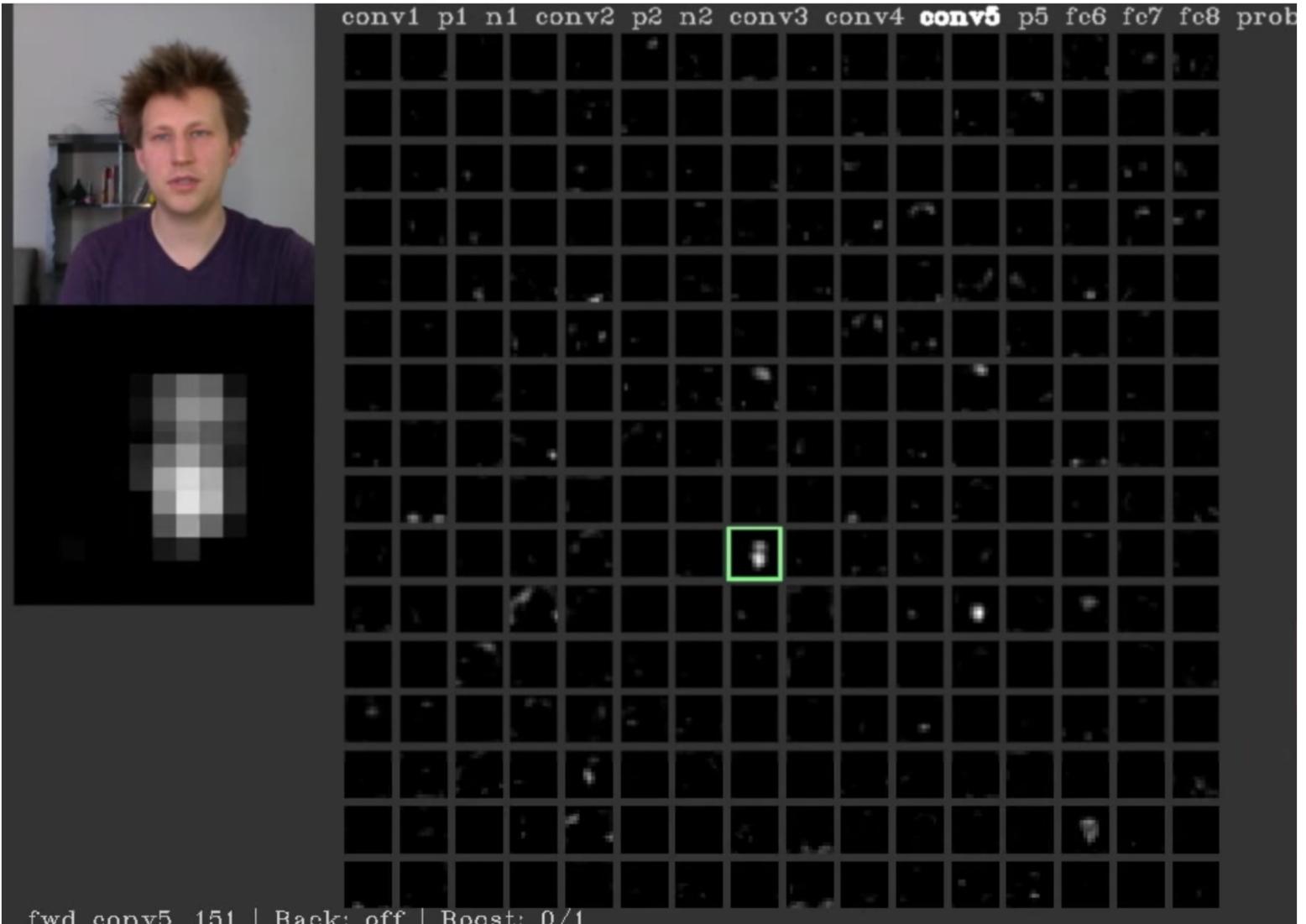


See high-resolution versions at
<http://cs.stanford.edu/people/karpathy/cnnembed/>

Van der Maaten and Hinton, "Visualizing Data using t-SNE", JMLR 2008
Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012.
Figure reproduced with permission.

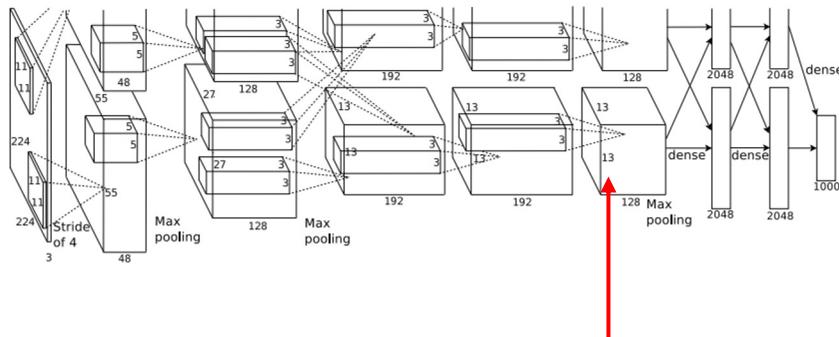
Visualizing Activations

conv5 feature map is
128x13x13; visualize as
128 13x13 grayscale
images



Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.
Figure copyright Jason Yosinski, 2014. Reproduced with permission.

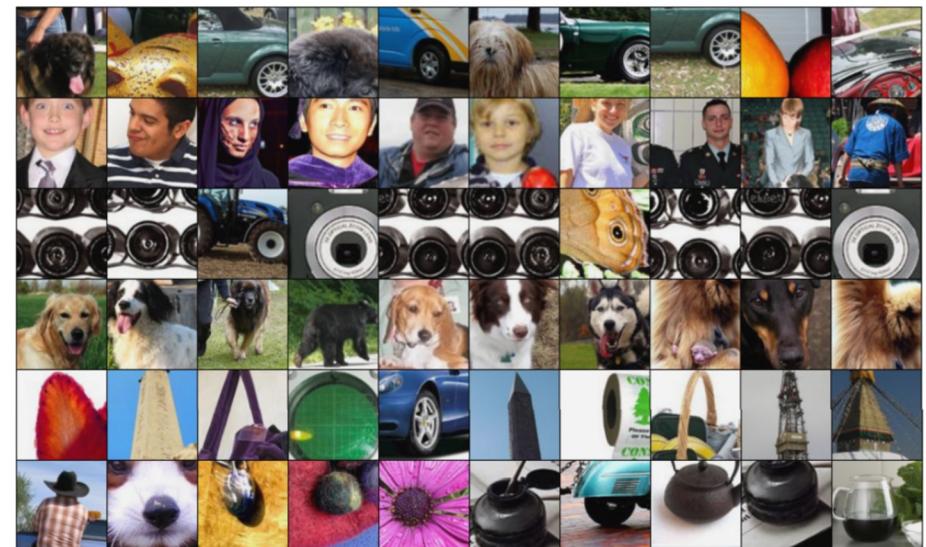
Maximally Activating Patches



Pick a layer and a channel; e.g. conv5 is $128 \times 13 \times 13$, pick channel 17/128

Run many images through the network,
record values of chosen channel

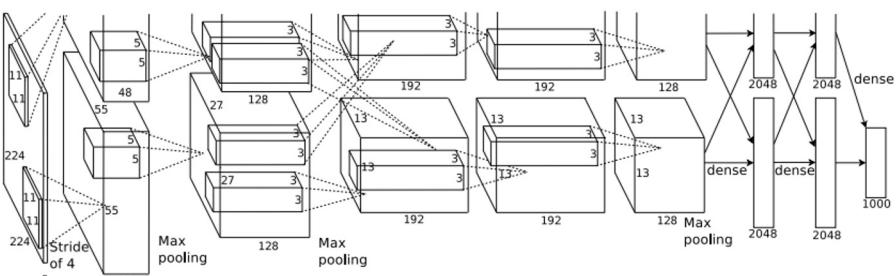
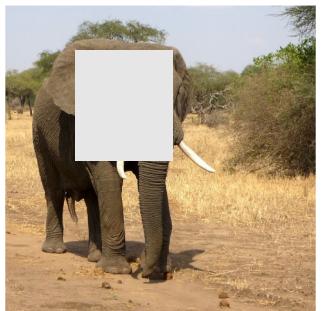
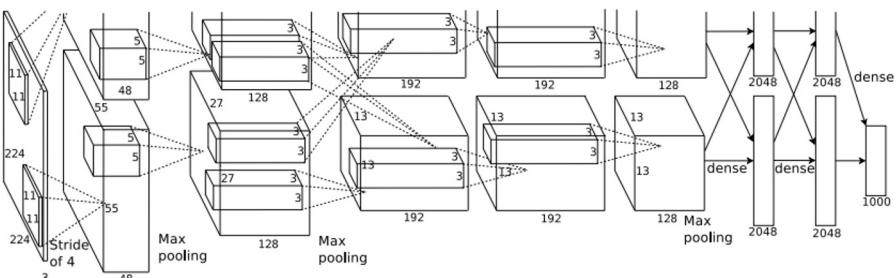
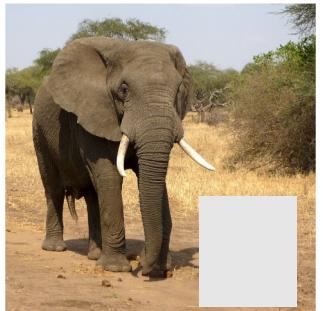
Visualize image patches that correspond to
maximal activations



Springenberg et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015
Figure copyright Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, 2015; reproduced with permission.

Which Pixels Matter? Saliency via Occlusion

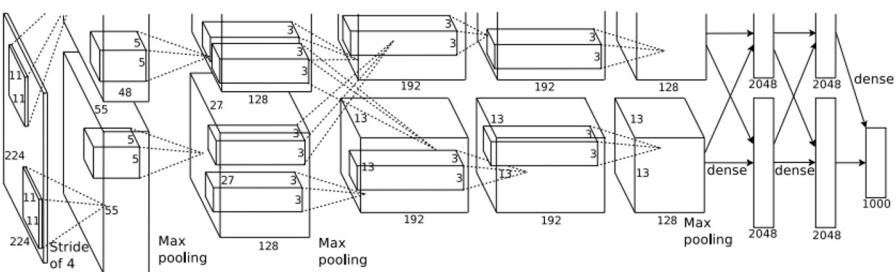
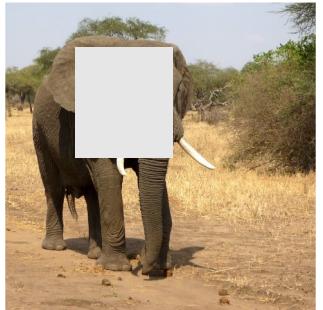
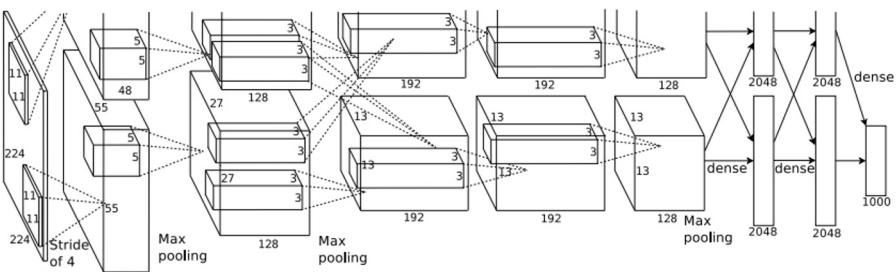
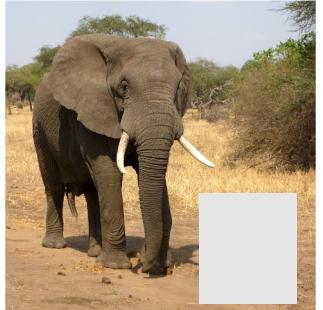
Mask part of the image before feeding to CNN,
check how much predicted probabilities change



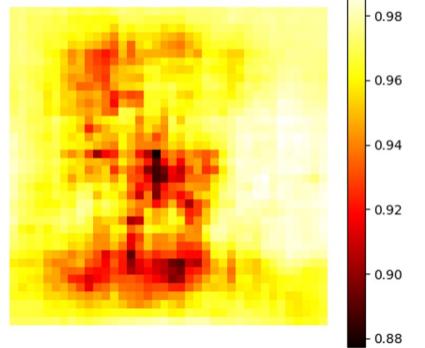
[Boat image](#) is CCO public domain
[Elephant image](#) is CCO public domain
[Go-Karts image](#) is CCO public domain

Which Pixels Matter? Saliency via Occlusion

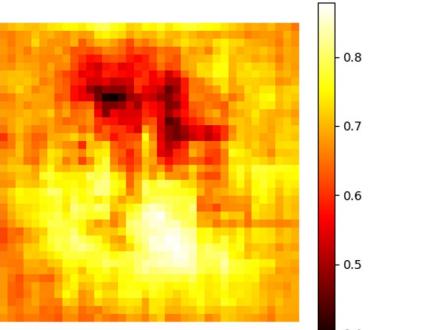
Mask part of the image before feeding to CNN,
check how much predicted probabilities change



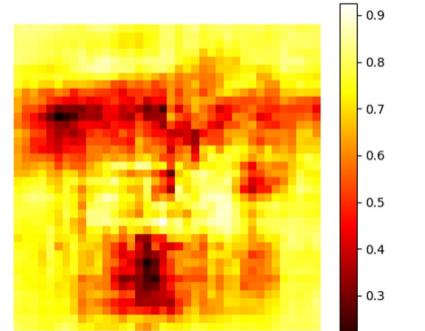
[Boat image](#) is CCO public domain
[Elephant image](#) is CCO public domain
[Go-Karts image](#) is CCO public domain



African elephant, *Loxodonta africana*

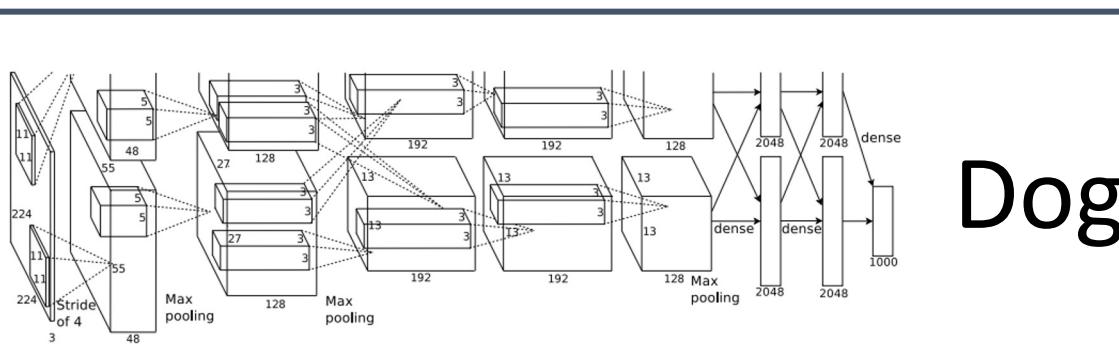


go-kart



Which pixels matter? Saliency via Backprop

Forward pass: Compute probabilities

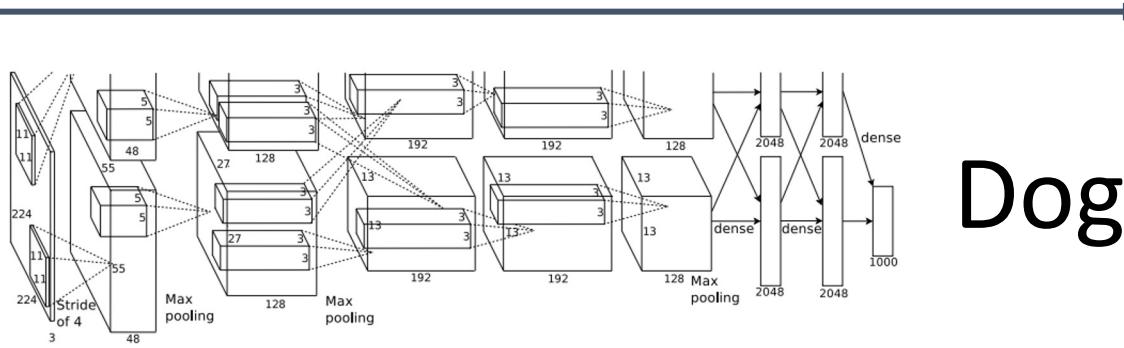


Dog

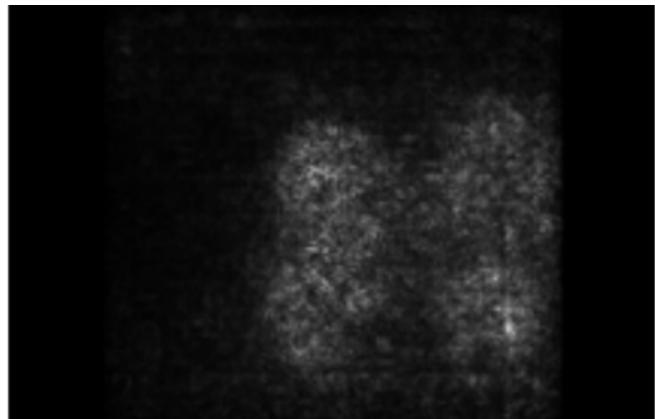
Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.
Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.

Which pixels matter? Saliency via Backprop

Forward pass: Compute probabilities

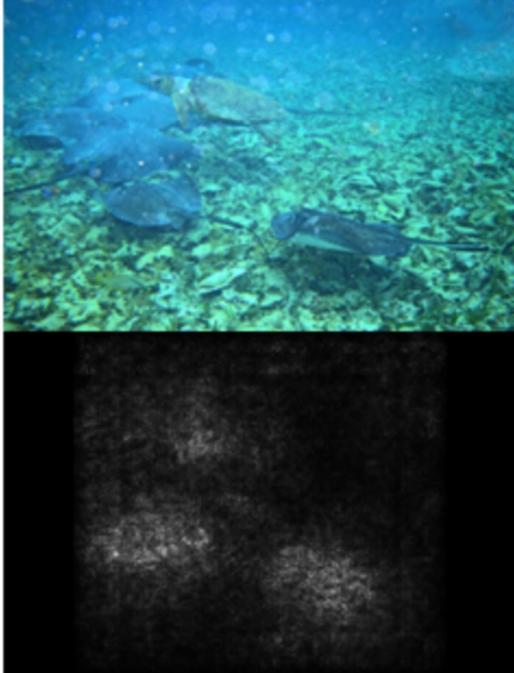
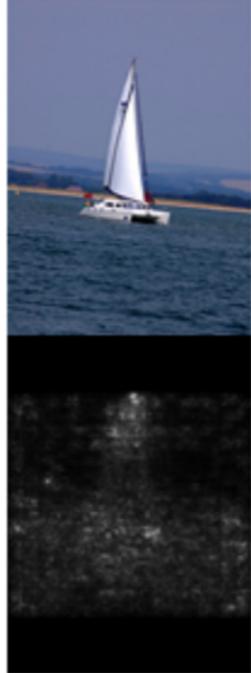


Compute gradient of (unnormalized) class score with respect to image pixels, take absolute value and max over RGB channels



Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.
Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.

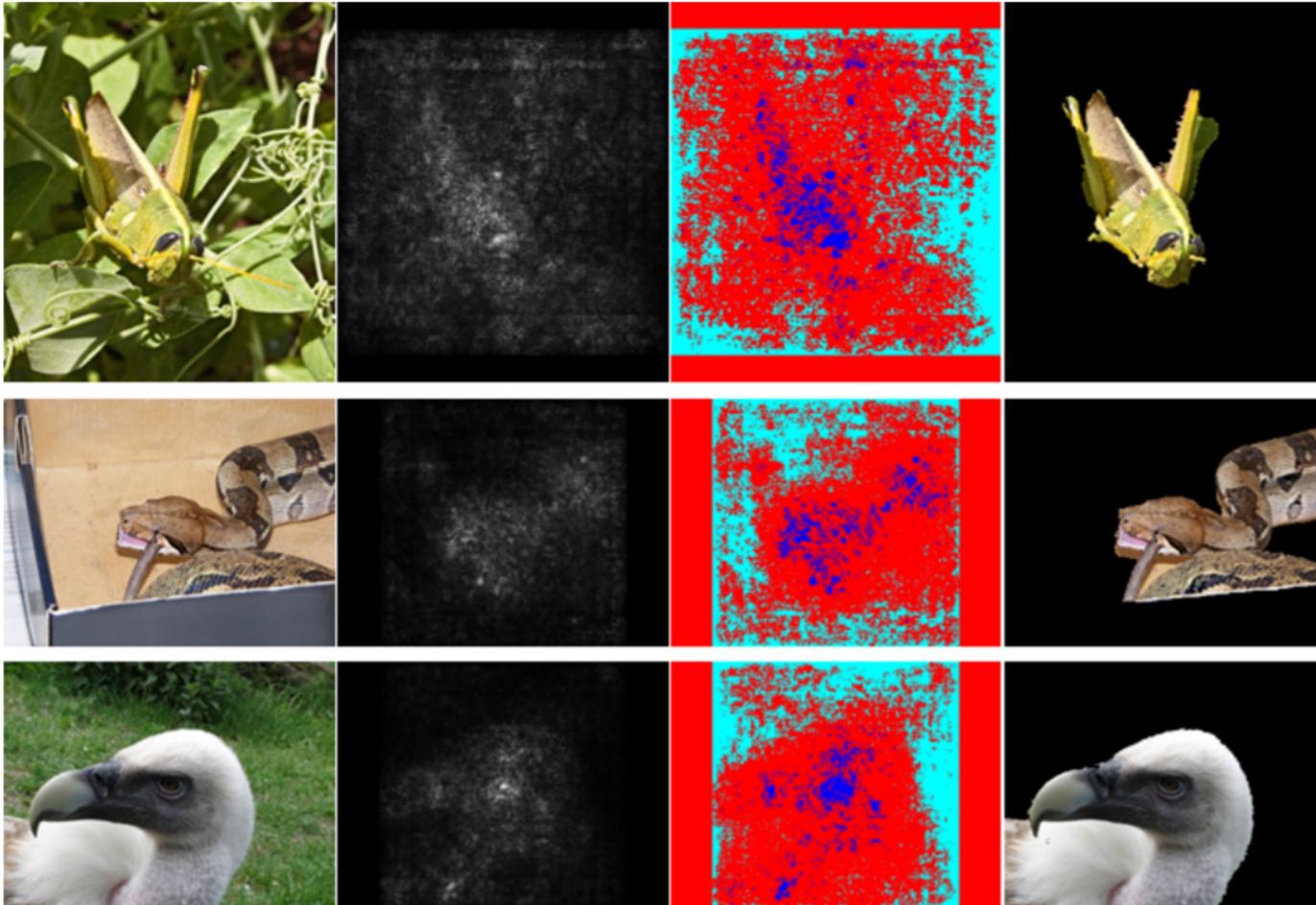
Which pixels matter? Saliency via Backprop



Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.
Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.

Saliency Maps: Segmentation without Supervision

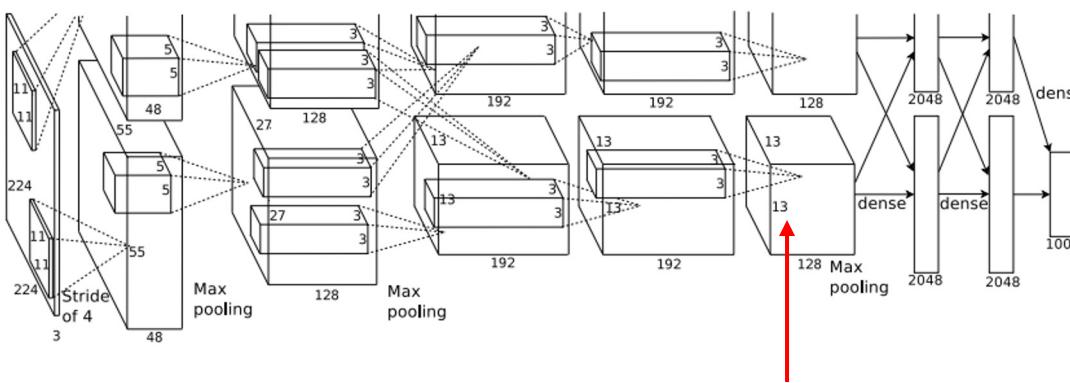
Use GrabCut on
saliency map



Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.

Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.
Rother et al, "Grabcut: Interactive foreground extraction using iterated graph cuts", ACM TOG 2004

Intermediate Features via (guided) backprop

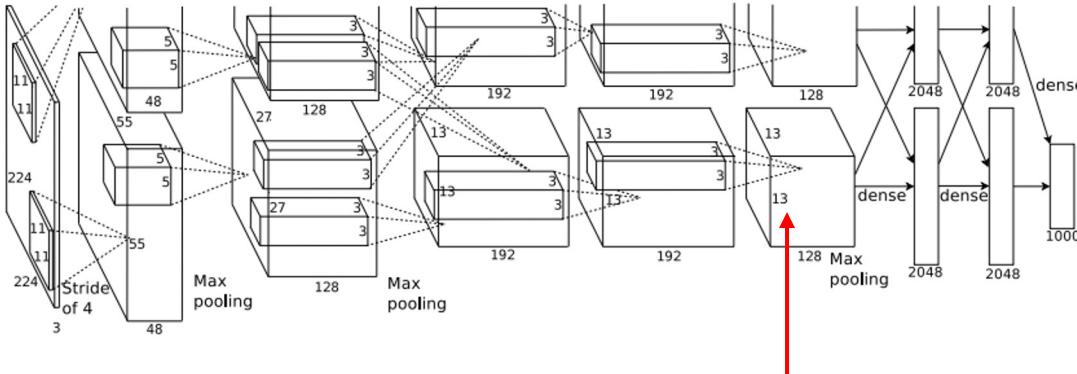


Pick a single intermediate neuron, e.g.
one value in $128 \times 13 \times 13$ conv5
feature map

Compute gradient of neuron value with
respect to image pixels

Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014
Springenber et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015

Intermediate Features via (guided) backprop



Pick a single intermediate neuron, e.g.
one value in $128 \times 13 \times 13$ conv5
feature map

Compute gradient of neuron value with
respect to image pixels

Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014
Springenber et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015

b)
Forward pass

$$\text{ReLU}$$

1	-1	5
2	-5	-7
-3	2	4

→

1	0	5
2	0	0
0	2	4

Backward pass:
backpropagation

-2	0	-1
6	0	0
0	-1	3

←

-2	3	-1
6	-3	1
2	-1	3

Backward pass:
"deconvnet"

0	3	0
6	0	1
2	0	3

←

-2	3	-1
6	-3	1
2	-1	3

Backward pass:
guided
backpropagation

0	0	0
6	0	0
0	0	3

←

-2	3	-1
6	-3	1
2	-1	3

Images come out nicer if you only
backprop positive gradients through
each ReLU (guided backprop)

Intermediate Features via (guided) backprop



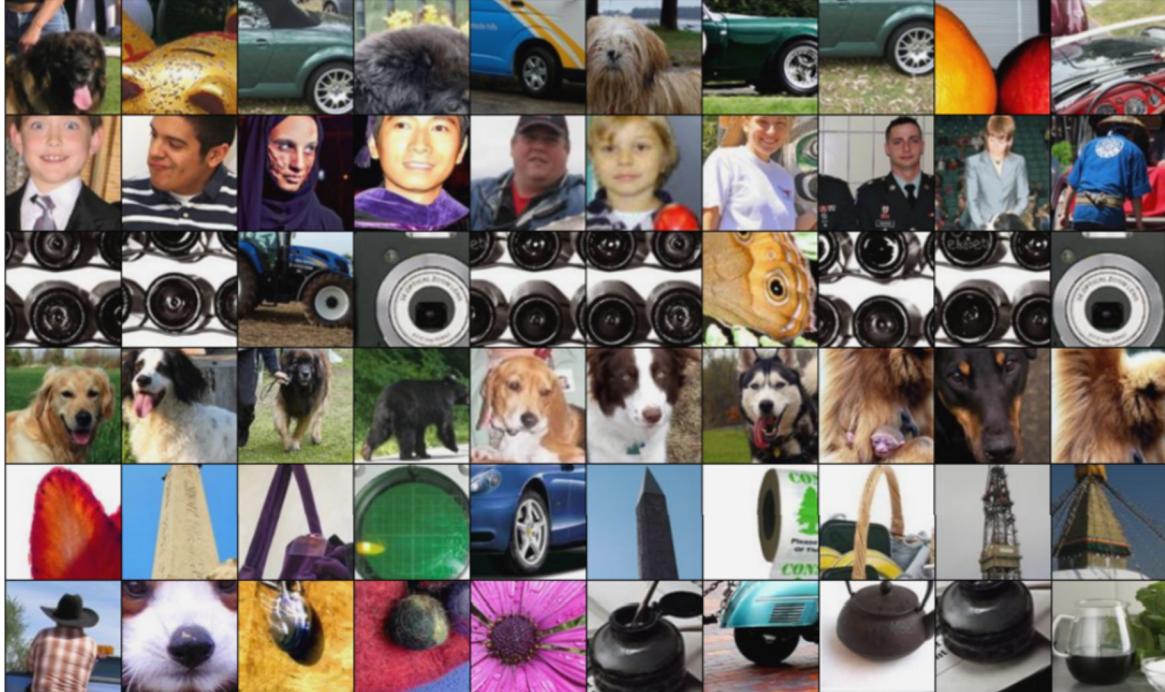
Maximally activating patches
(Each row is a different neuron)



Guided Backprop

Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014
Springenberg et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015
Figure copyright Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, 2015; reproduced with permission.

Intermediate Features via (guided) backprop



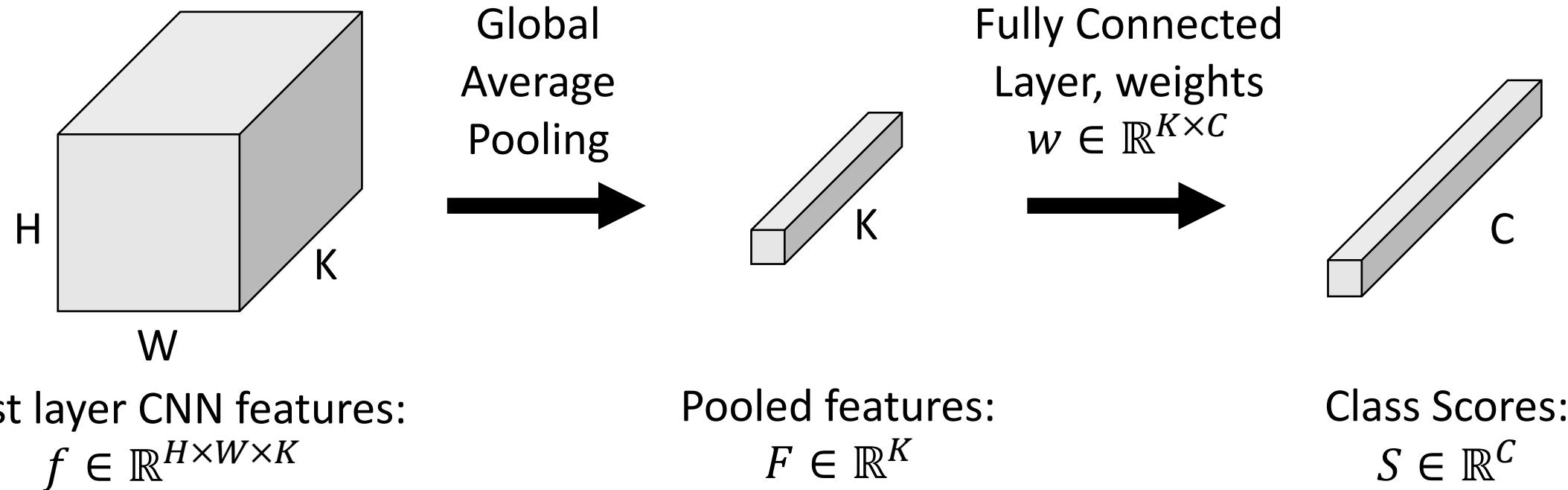
Maximally activating patches
(Each row is a different neuron)



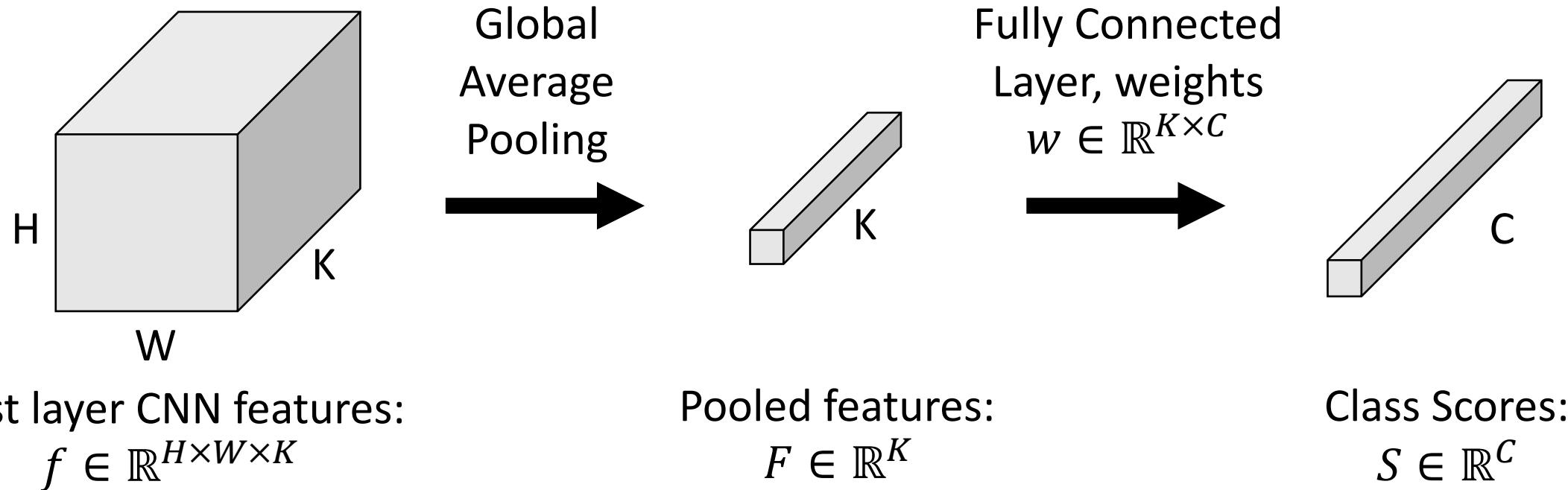
Guided Backprop

Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014
Springenberg et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015
Figure copyright Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, 2015; reproduced with permission.

Class Activation Mapping (CAM)



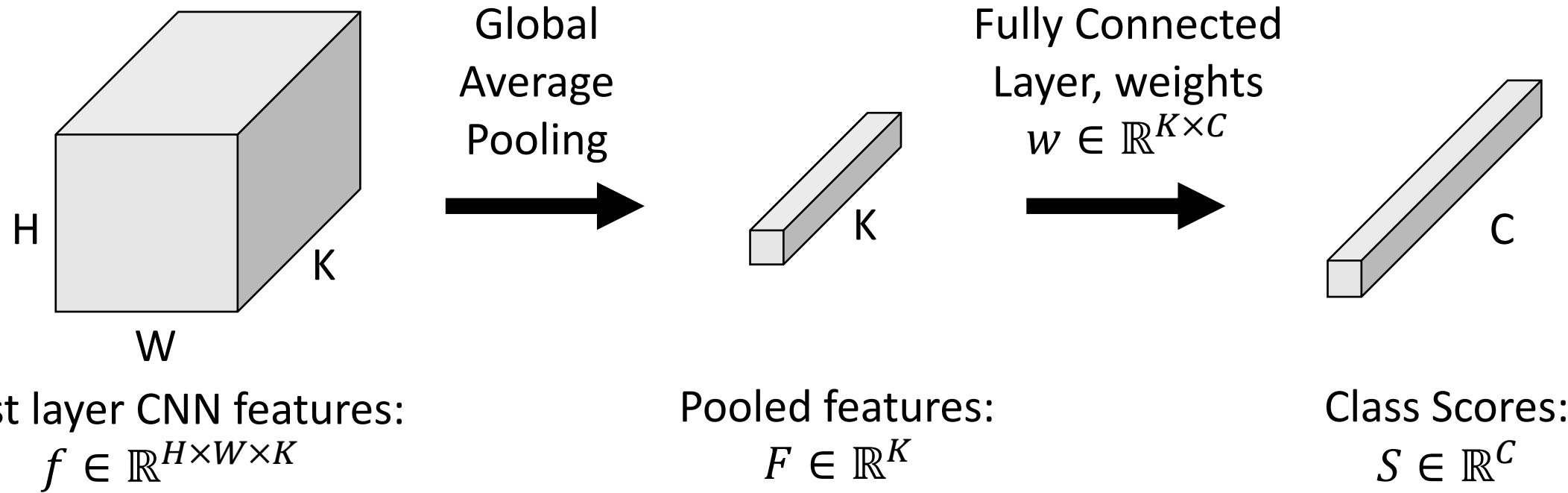
Class Activation Mapping (CAM)



$$F_k = \frac{1}{HW} \sum_{h,w} f_{h,w,k}$$

Zhou et al, "Learning Deep Features for Discriminative Localization", CVPR 2016

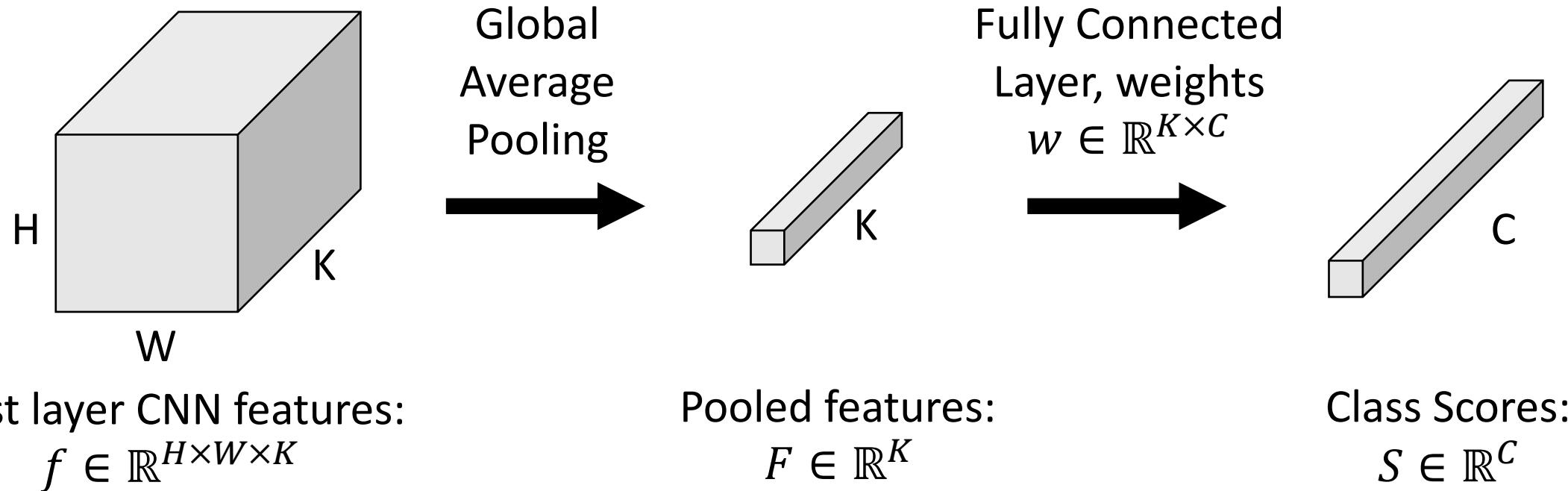
Class Activation Mapping (CAM)



$$F_k = \frac{1}{HW} \sum_{h,w} f_{h,w,k} \quad S_c = \sum_k w_{k,c} F_k$$

Zhou et al, "Learning Deep Features for Discriminative Localization", CVPR 2016

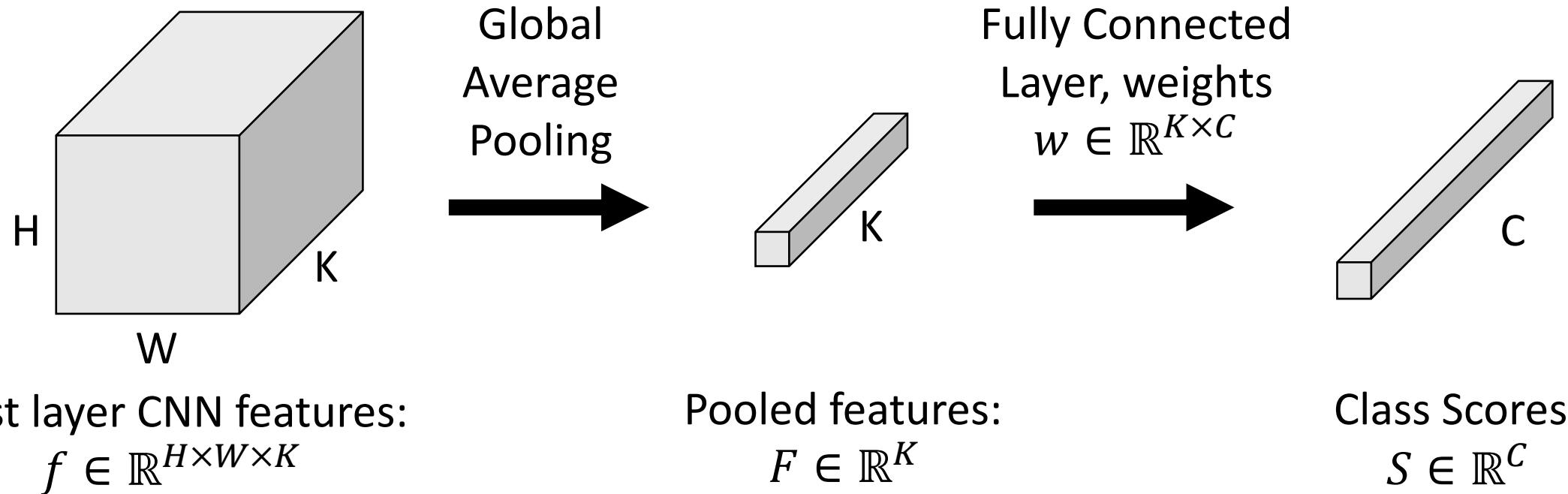
Class Activation Mapping (CAM)



$$F_k = \frac{1}{HW} \sum_{h,w} f_{h,w,k} \quad S_c = \sum_k w_{k,c} F_k = \frac{1}{HW} \sum_k w_{k,c} \sum_{h,w} f_{h,w,k}$$

Zhou et al, "Learning Deep Features for Discriminative Localization", CVPR 2016

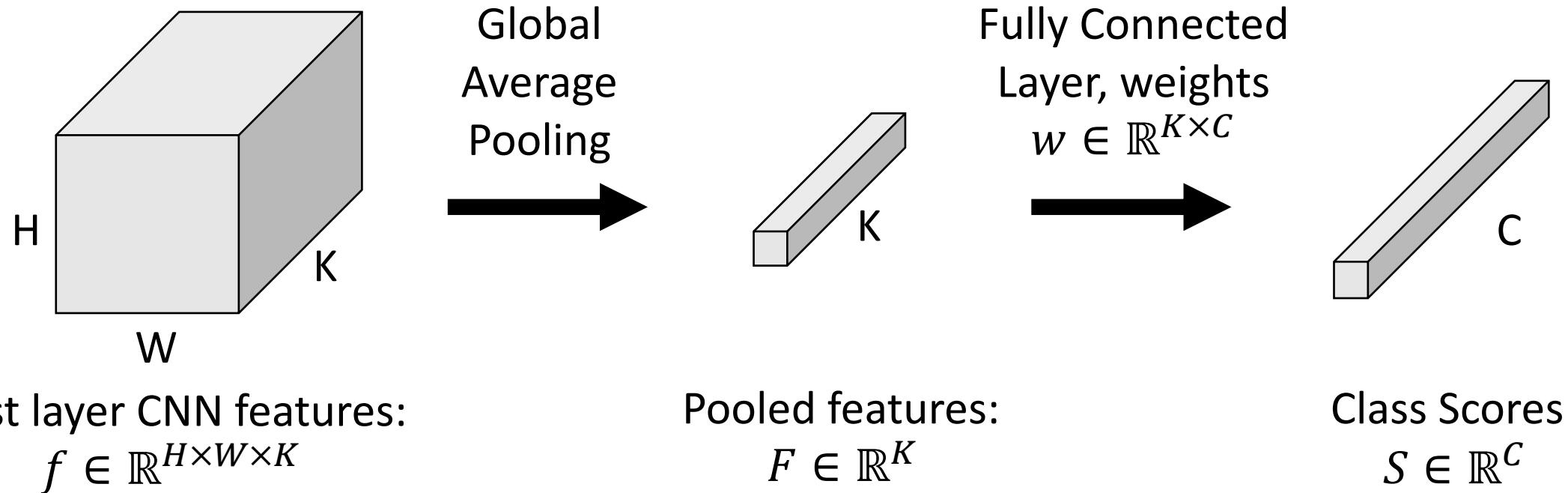
Class Activation Mapping (CAM)



$$F_k = \frac{1}{HW} \sum_{h,w} f_{h,w,k} \quad S_c = \sum_k w_{k,c} F_k = \frac{1}{HW} \sum_k w_{k,c} \sum_{h,w} f_{h,w,k} \\ = \frac{1}{HW} \sum_{h,w} \sum_k w_{k,c} f_{h,w,k}$$

Zhou et al, "Learning Deep Features for Discriminative Localization", CVPR 2016

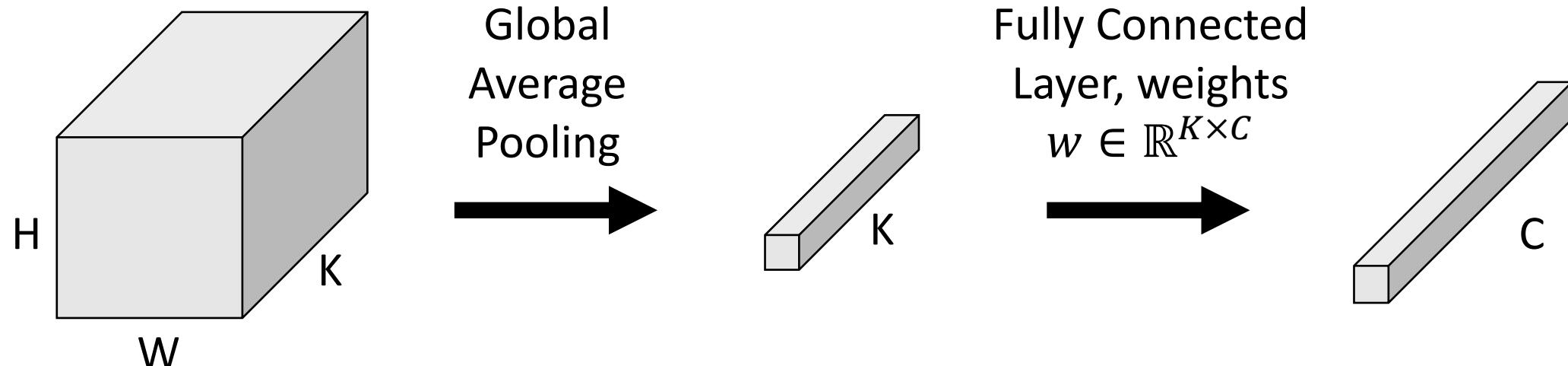
Class Activation Mapping (CAM)



$$F_k = \frac{1}{HW} \sum_{h,w} f_{h,w,k} \quad S_c = \sum_k w_{k,c} F_k = \frac{1}{HW} \sum_k w_{k,c} \sum_{h,w} f_{h,w,k}$$
$$= \frac{1}{HW} \sum_{h,w} \sum_k \mathbf{w}_{k,c} \mathbf{f}_{h,w,k}$$

Zhou et al, "Learning Deep Features for Discriminative Localization", CVPR 2016

Class Activation Mapping (CAM)



Last layer CNN features:
 $f \in \mathbb{R}^{H \times W \times K}$

Pooled features:
 $F \in \mathbb{R}^K$

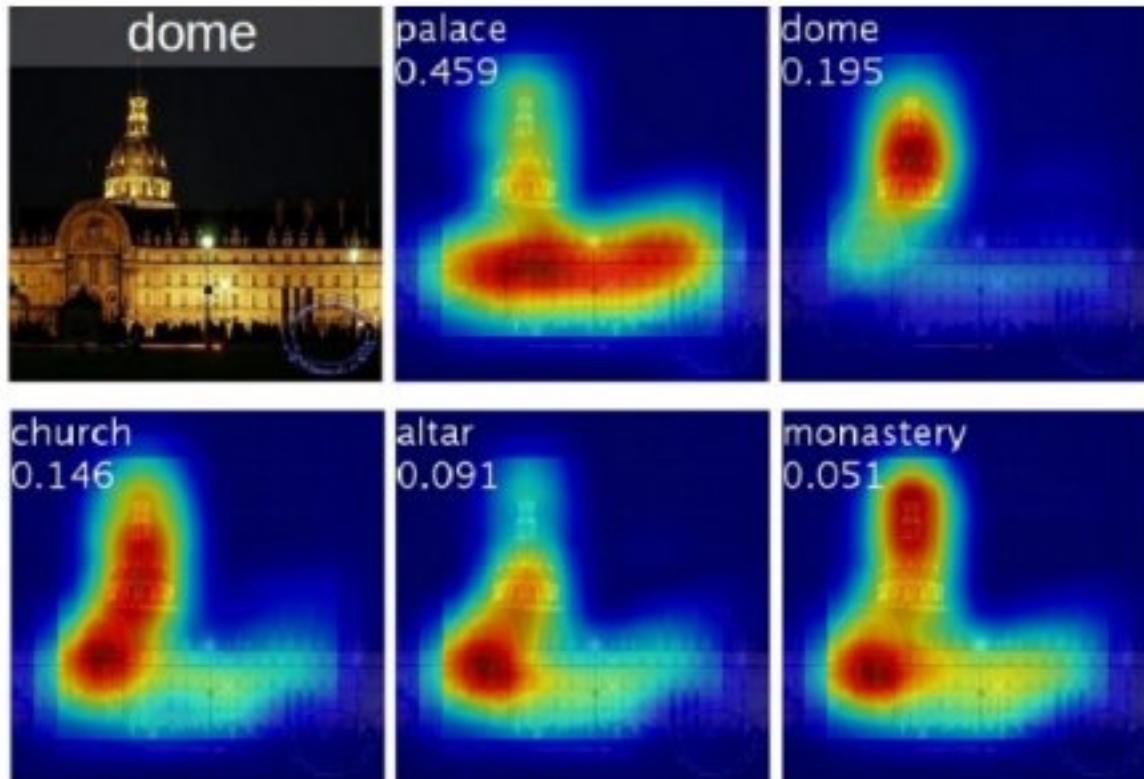
Class Scores:
 $S \in \mathbb{R}^C$

$$F_k = \frac{1}{HW} \sum_{h,w} f_{h,w,k} \quad S_c = \sum_k w_{k,c} F_k = \frac{1}{HW} \sum_k w_{k,c} \sum_{h,w} f_{h,w,k} \\ = \frac{1}{HW} \sum_{h,w} \sum_k w_{k,c} f_{h,w,k}$$

Class Activation Maps:
 $M \in \mathbb{R}^{C,H,W}$
 $M_{c,h,w} = \sum_k w_{k,c} f_{h,w,k}$

Zhou et al, "Learning Deep Features for Discriminative Localization", CVPR 2016

Class Activation Mapping (CAM)



Class activation maps of top 5 predictions

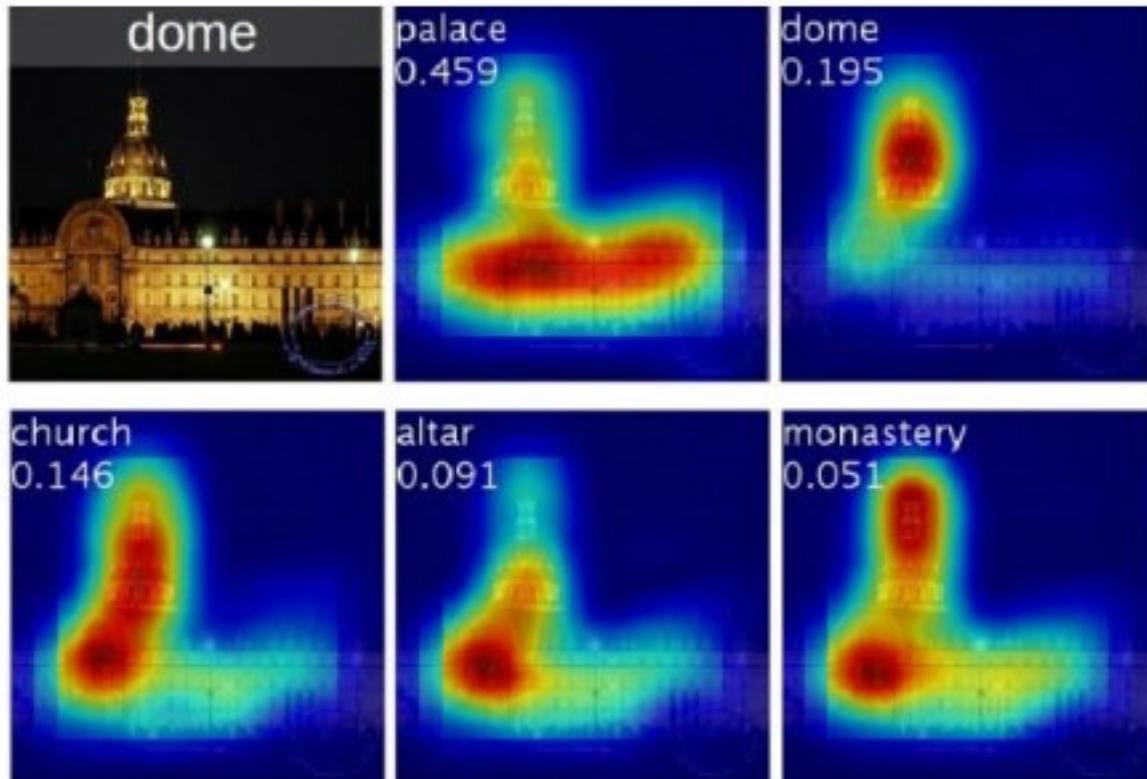


Class activation maps for one object class

Zhou et al, "Learning Deep Features for Discriminative Localization", CVPR 2016

Class Activation Mapping (CAM)

Problem: Can only apply to last conv layer



Class activation maps of top 5 predictions



Class activation maps for one object class

Gradient-Weighted Class Activation Mapping (Grad-CAM)

1. Pick any layer, with activations $A \in \mathbb{R}^{H \times W \times K}$

Gradient-Weighted Class Activation Mapping (Grad-CAM)

1. Pick any layer, with activations $A \in \mathbb{R}^{H \times W \times K}$
2. Compute gradient of class score S_c with respect to A:

$$\frac{\partial S_c}{\partial A} \in \mathbb{R}^{H \times W \times K}$$

Gradient-Weighted Class Activation Mapping (Grad-CAM)

1. Pick any layer, with activations $A \in \mathbb{R}^{H \times W \times K}$
2. Compute gradient of class score S_c with respect to A:

$$\frac{\partial S_c}{\partial A} \in \mathbb{R}^{H \times W \times K}$$

3. Global Average Pool the gradients to get weights $\alpha \in \mathbb{R}^K$:

$$\alpha_k = \frac{1}{HW} \sum_{h,w} \frac{\partial S_c}{\partial A_{h,w,k}}$$

Gradient-Weighted Class Activation Mapping (Grad-CAM)

1. Pick any layer, with activations $A \in \mathbb{R}^{H \times W \times K}$
2. Compute gradient of class score S_c with respect to A:

$$\frac{\partial S_c}{\partial A} \in \mathbb{R}^{H \times W \times K}$$

3. Global Average Pool the gradients to get weights $\alpha \in \mathbb{R}^K$:

$$\alpha_k = \frac{1}{HW} \sum_{h,w} \frac{\partial S_c}{\partial A_{h,w,k}}$$

4. Compute activation map $M^c \in \mathbb{R}^{H,W}$:

$$M_{h,w}^c = \text{ReLU} \left(\sum_k \alpha_k A_{h,w,k} \right)$$

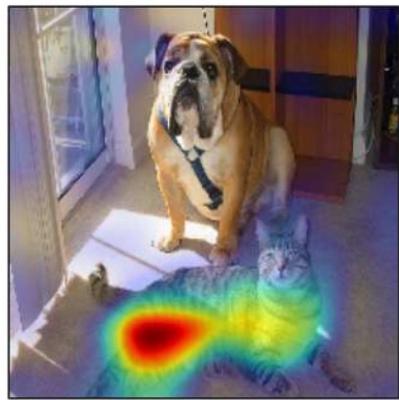
Gradient-Weighted Class Activation Mapping (Grad-CAM)



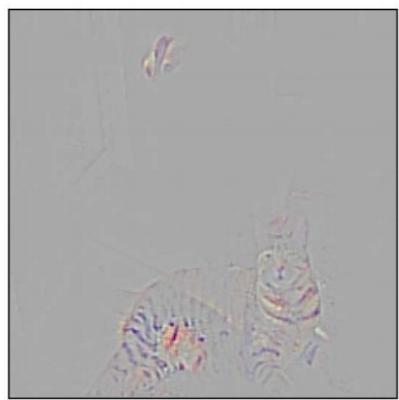
(a) Original Image



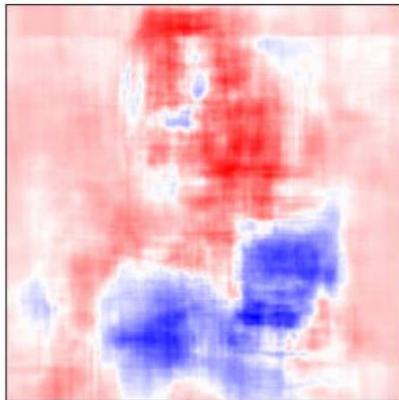
(b) Guided Backprop ‘Cat’



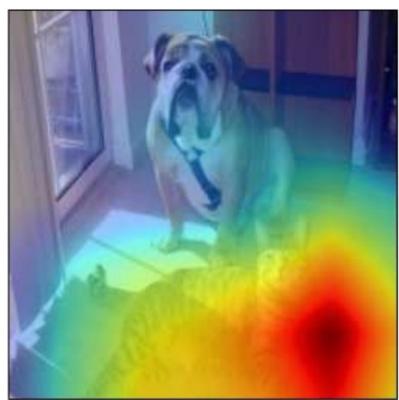
(c) Grad-CAM ‘Cat’



(d) Guided Grad-CAM ‘Cat’



(e) Occlusion map for ‘Cat’



(f) ResNet Grad-CAM ‘Cat’



(g) Original Image



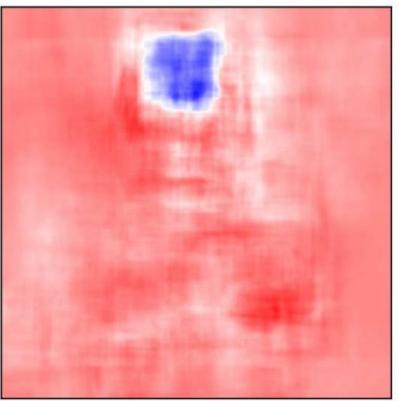
(h) Guided Backprop ‘Dog’



(i) Grad-CAM ‘Dog’



(j) Guided Grad-CAM ‘Dog’



(k) Occlusion map for ‘Dog’



(l) ResNet Grad-CAM ‘Dog’

Selvaraju et al, “Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization”, CVPR 2017

Gradient-Weighted Class Activation Mapping (Grad-CAM)

Can also be applied beyond classification models, e.g. image captioning



A group of people flying kites on a beach

A man is sitting at a table with a pizza

Selvaraju et al, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization", CVPR 2017

Visualizing CNN Features: Gradient Ascent

(Guided) backprop:

Find the part of an image that a neuron responds to

Gradient ascent:

Generate a synthetic image that maximally activates a neuron

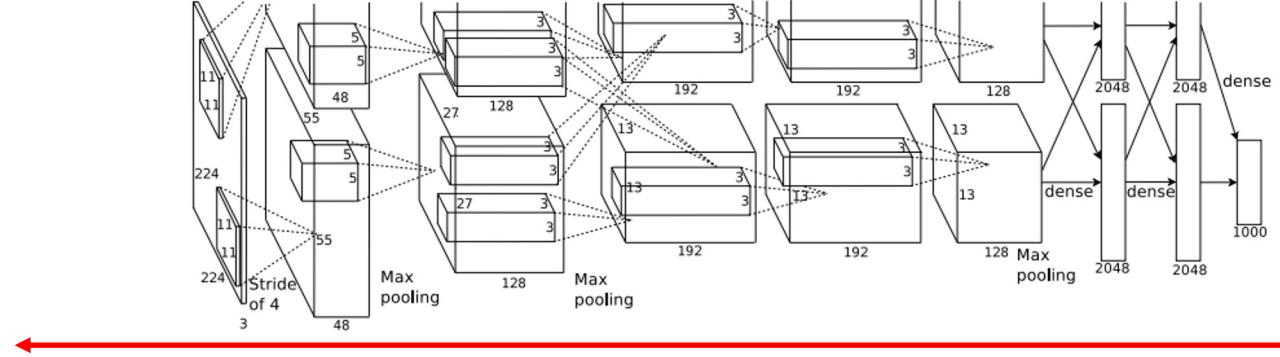
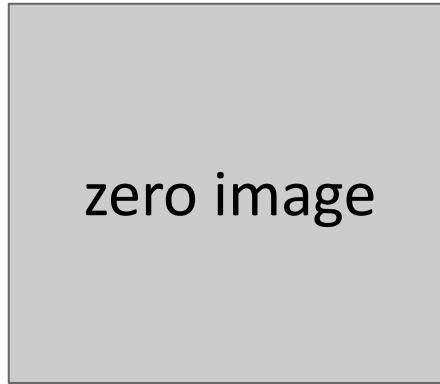
$$I^* = \arg \max_I f(I) + R(I)$$

Neuron value

Natural image regularizer

Visualizing CNN Features: Gradient Ascent

1. Initialize image to zeros



Repeat:

2. Forward image to compute current scores
3. Backprop to get gradient of neuron value with respect to image pixels
4. Make a small update to the image

$$\arg \max_I S_c(I) - \lambda \|I\|_2^2$$

score for class c (before Softmax)

Visualizing CNN Features: Gradient Ascent

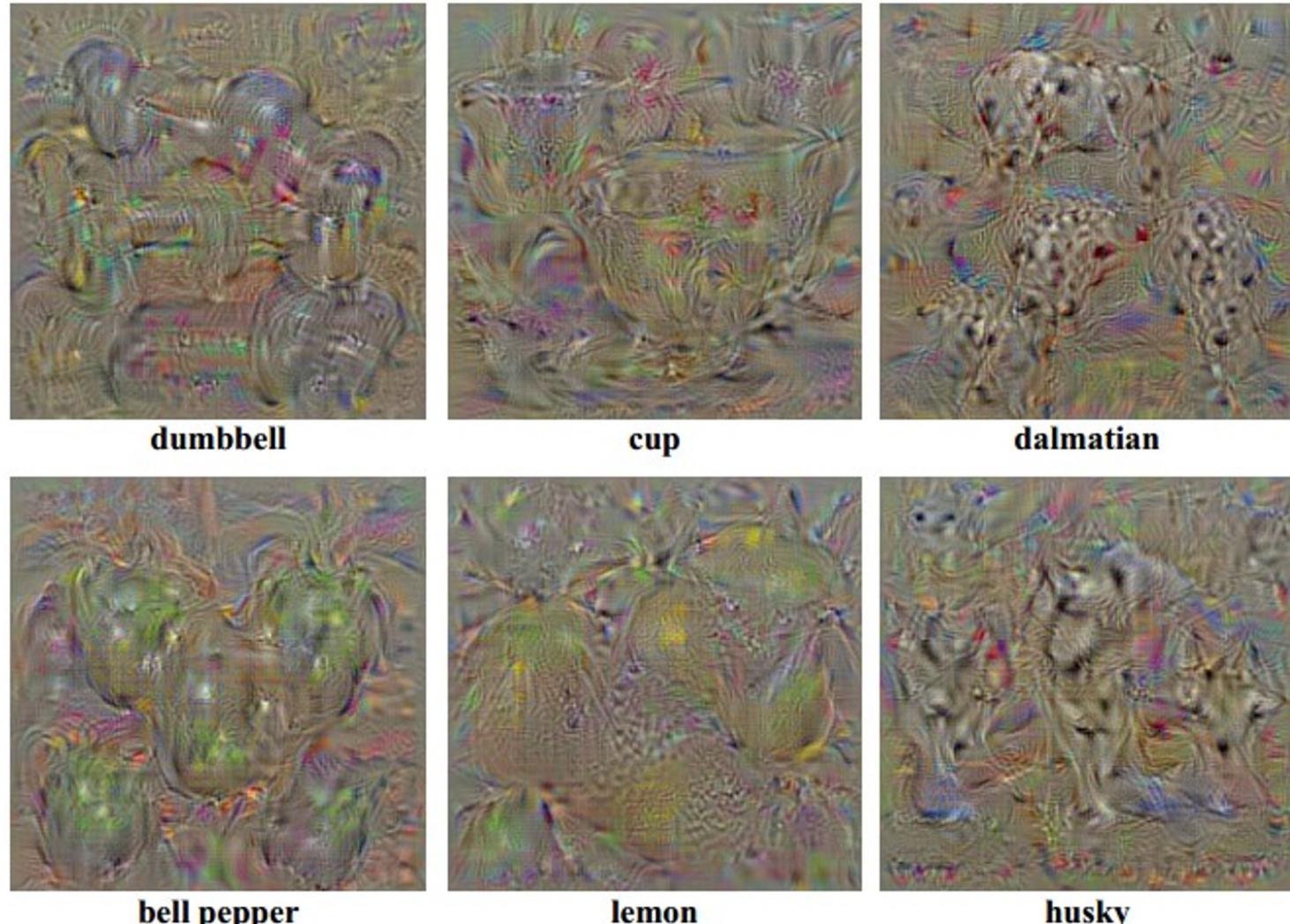
$$\arg \max_I S_c(I) - \boxed{\lambda \|I\|_2^2}$$

Simple regularizer: Penalize
L2 norm of generated image

Visualizing CNN Features: Gradient Ascent

$$\arg \max_I S_c(I) - \boxed{\lambda \|I\|_2^2}$$

Simple regularizer: Penalize L2 norm of generated image



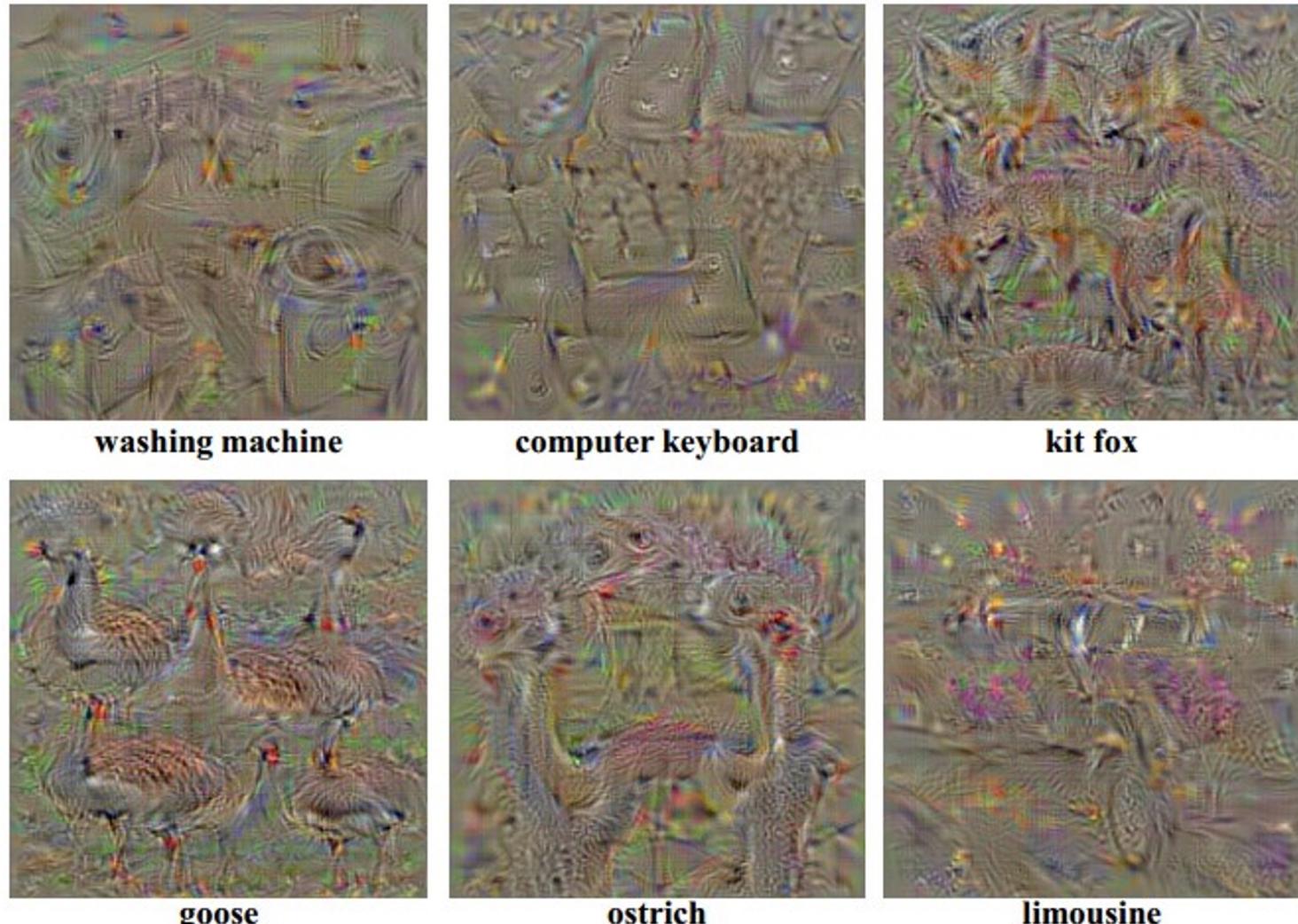
Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.

Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.

Visualizing CNN Features: Gradient Ascent

$$\arg \max_I S_c(I) - \boxed{\lambda \|I\|_2^2}$$

Simple regularizer: Penalize L2 norm of generated image



Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.

Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.

Visualizing CNN Features: Gradient Ascent

$$\arg \max_I S_c(I) - \lambda \|I\|_2^2$$

Better regularizer: Penalize L2 norm of image; also during optimization periodically

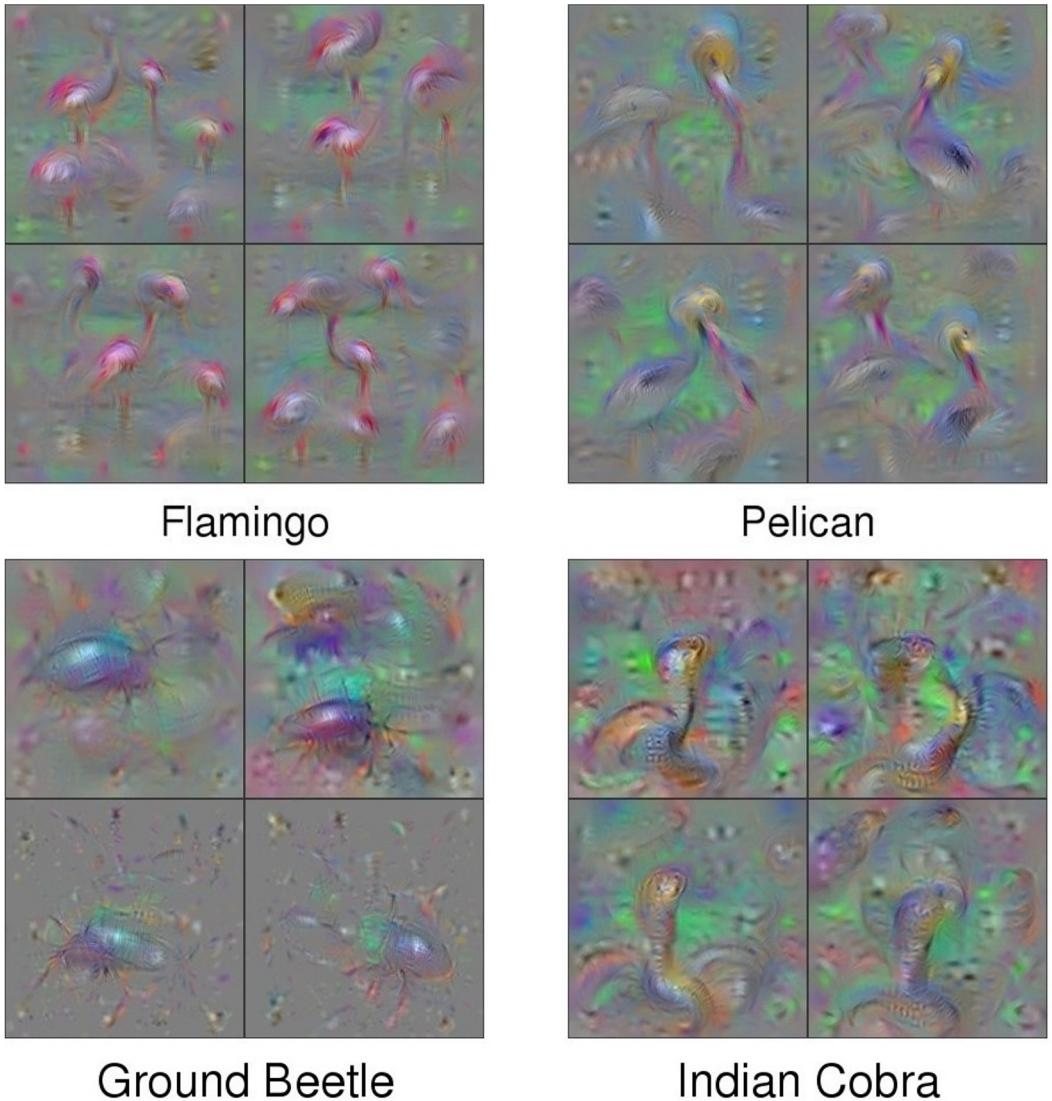
1. Gaussian blur image
2. Clip pixels with small values to 0
3. Clip pixels with small gradients to 0

Visualizing CNN Features: Gradient Ascent

$$\arg \max_I S_c(I) - \lambda \|I\|_2^2$$

Better regularizer: Penalize L2 norm of image; also during optimization periodically

1. Gaussian blur image
2. Clip pixels with small values to 0
3. Clip pixels with small gradients to 0

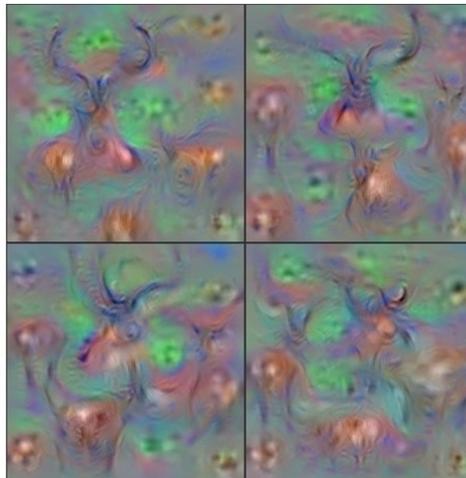


Visualizing CNN Features: Gradient Ascent

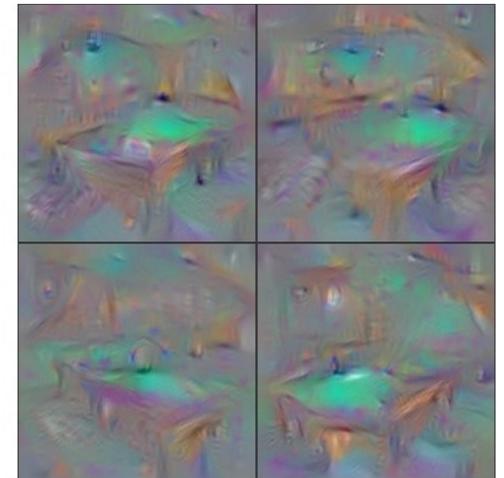
$$\arg \max_I S_c(I) - \lambda \|I\|_2^2$$

Better regularizer: Penalize L2 norm of image; also during optimization periodically

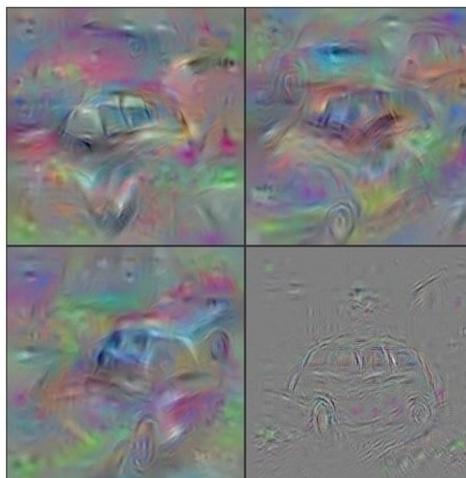
1. Gaussian blur image
2. Clip pixels with small values to 0
3. Clip pixels with small gradients to 0



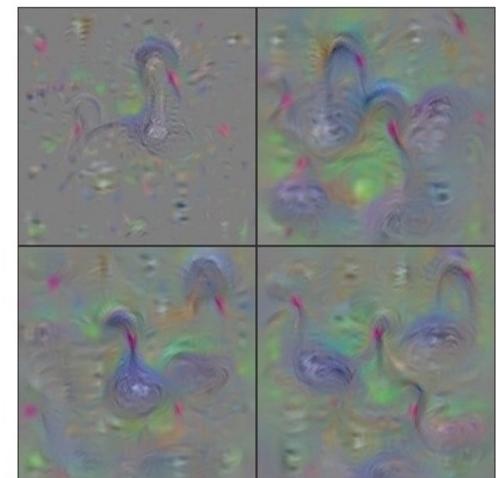
Hartebeest



Billiard Table



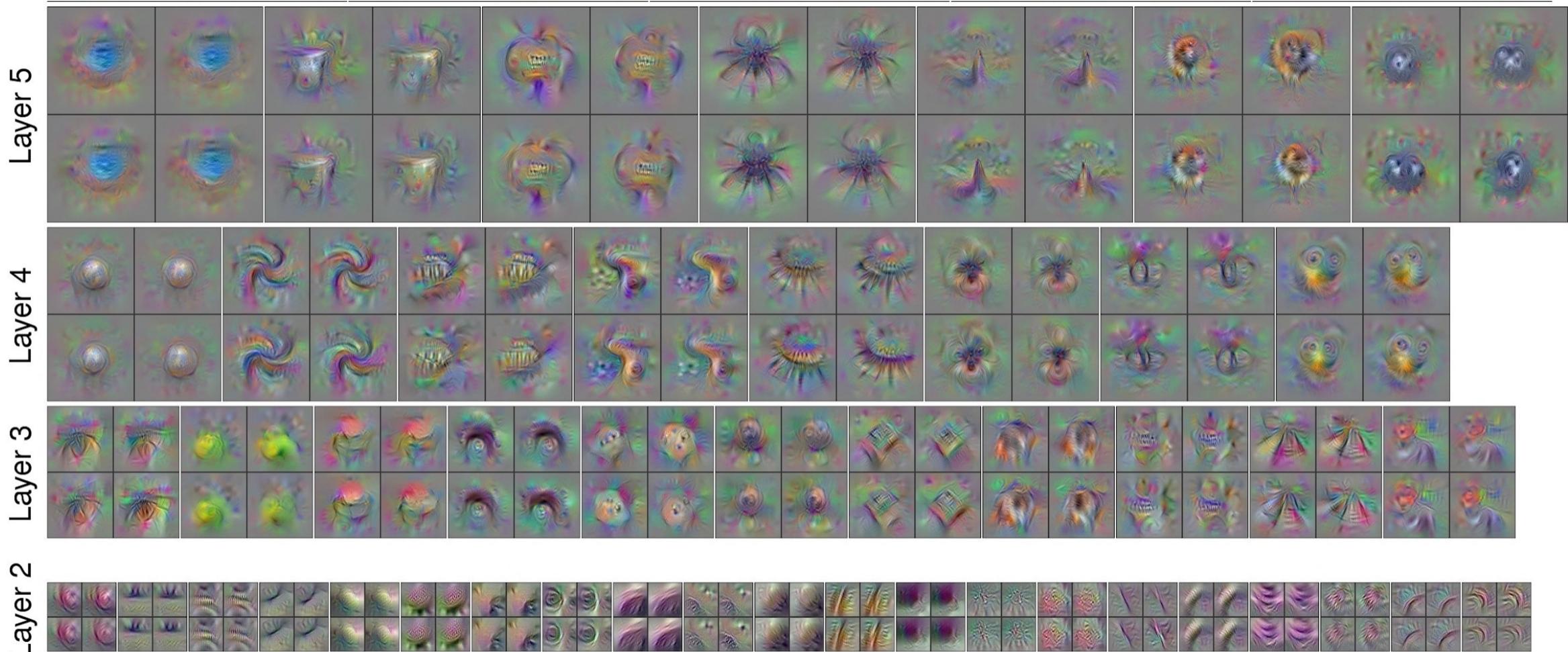
Station Wagon



Black Swan

Visualizing CNN Features: Gradient Ascent

Use the same approach to visualize intermediate features



Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.

Visualizing CNN Features: Gradient Ascent

Adding “multi-faceted” visualization gives even nicer results:
(Plus more careful regularization, center-bias)

Reconstructions of multiple feature types (facets) recognized by the same “grocery store” neuron



Corresponding example training set images recognized by the same neuron as in the “grocery store” class



Nguyen et al, “Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks”, ICML Visualization for Deep Learning Workshop 2016.

Visualizing CNN Features: Gradient Ascent



Nguyen et al, "Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks", ICML Visualization for Deep Learning Workshop 2016.

Visualizing CNN Features: Gradient Ascent



Nguyen et al, "Synthesizing the preferred inputs for neurons in neural networks via deep generator networks," NIPS 2016

Adversarial Examples

1. Start from an arbitrary image
2. Pick an arbitrary category
3. Modify the image (via gradient ascent)
to maximize the class score
4. Stop when the network is fooled

Szegedy et al, "Intriguing properties of neural networks", 2013

Adversarial Examples

African elephant



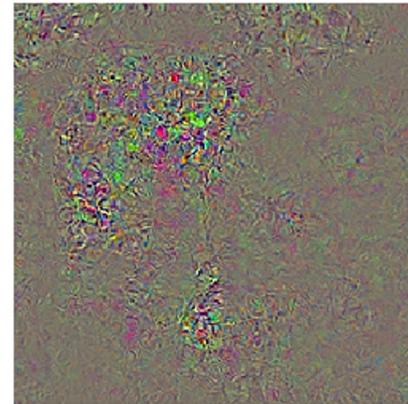
koala



Difference



10x Difference



schooner



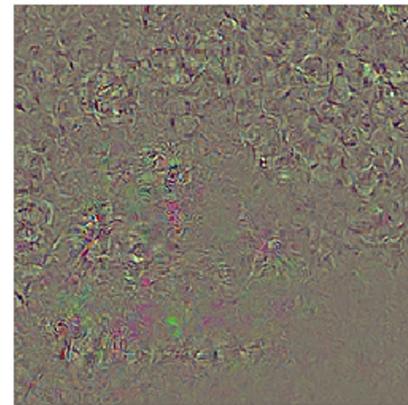
iPod



Difference



10x Difference



Boat image is [CC0 public domain](#)

Elephant image is [CC0 public domain](#)

Adversarial Attacks and Defense

Adversarial Attack: Method for generating adversarial examples for a network

Adversarial Defense: Change to network architecture, training, etc that make it harder to attack

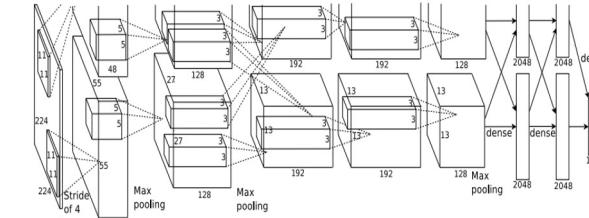
Adversarial Attacks and Defense

Adversarial Attack: Method for generating adversarial examples for a network – **Easy**

Adversarial Defense: Change to network architecture, training, etc that make it harder to attack – **Hard**

Adversarial Attacks

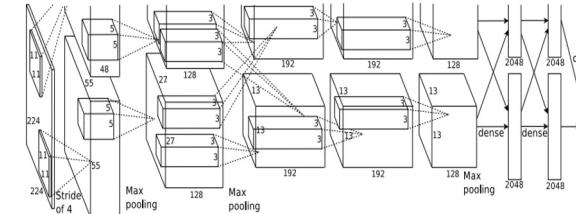
White-box attack: We have access to the network architecture and weights. Can get outputs, gradients for arbitrary input images.



P(elephant) = 0.9
P(cat) = 0.05
...

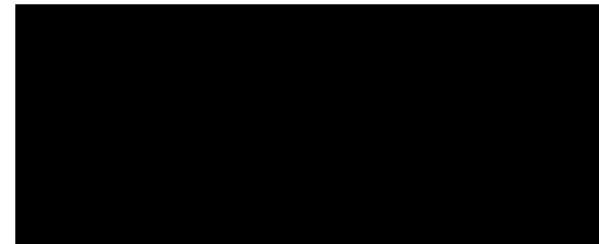
Adversarial Attacks

White-box attack: We have access to the network architecture and weights. Can get outputs, gradients for arbitrary input images.



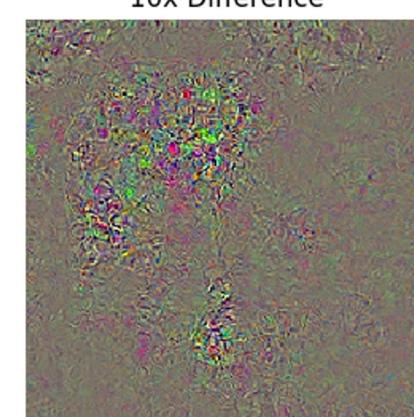
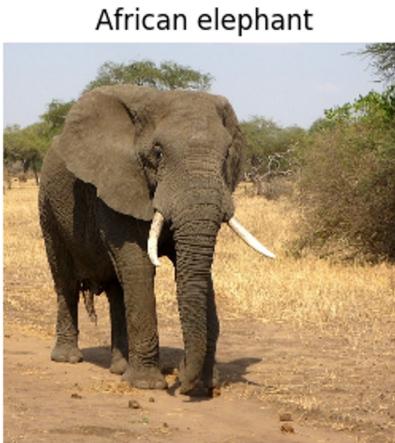
$P(\text{elephant}) = 0.9$
 $P(\text{cat}) = 0.05$
...

Black-box attack: We don't know network architecture or weights; can only get network predictions for arbitrary input images



$P(\text{elephant}) = 0.9$
 $P(\text{cat}) = 0.05$
...

Adversarial Examples



Huge area of research!

Security concern for networks deployed in the wild

Feature Inversion

Given a CNN feature vector for an image, find a new image that:

- Matches the given feature vector
- “looks natural” (image prior regularization)

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathbb{R}^{H \times W \times C}}{\operatorname{argmin}} \ell(\Phi(\mathbf{x}), \Phi_0) + \lambda \mathcal{R}(\mathbf{x})$$

Given feature vector

$$\ell(\Phi(\mathbf{x}), \Phi_0) = \|\Phi(\mathbf{x}) - \Phi_0\|^2$$

Features of new image
$$\mathcal{R}_{V^\beta}(\mathbf{x}) = \sum_{i,j} \left((x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2 \right)^{\frac{\beta}{2}}$$

Total Variation regularizer (encourages spatial smoothness)

Mahendran and Vedaldi, “Understanding Deep Image Representations by Inverting Them”, CVPR 2015

Feature Inversion

Reconstructing from different layers of VGG-16

y

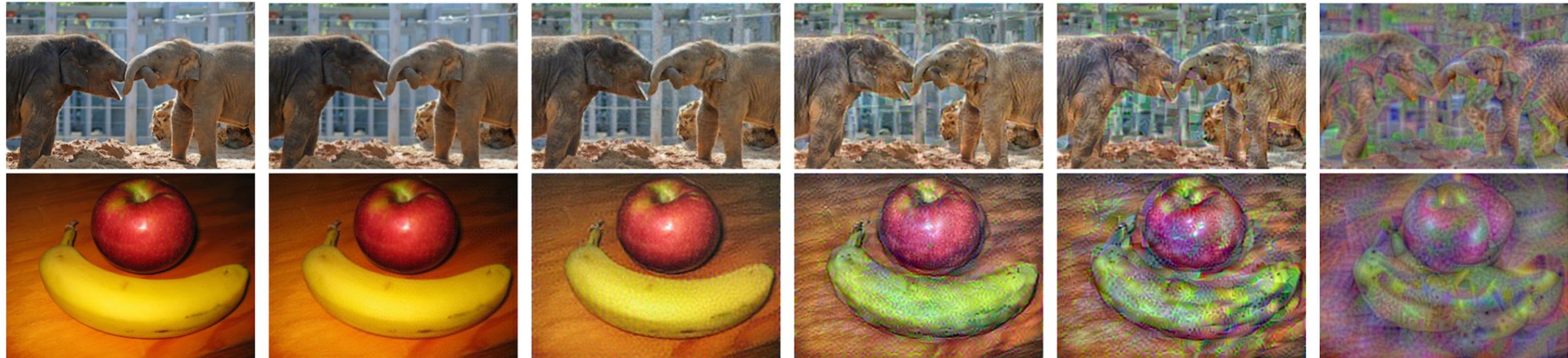
`relu2_2`

`relu3_3`

`relu4_3`

`relu5_1`

`relu5_3`

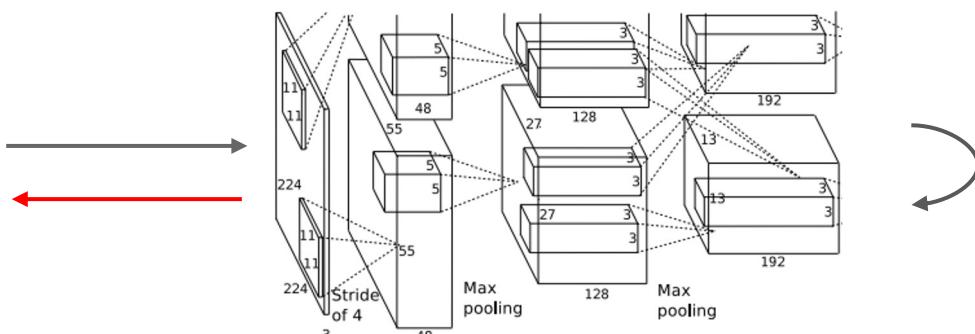
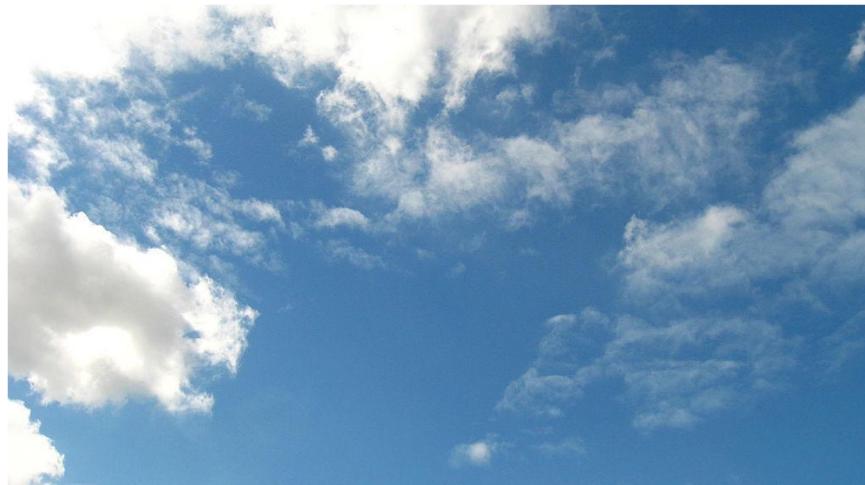


Mahendran and Vedaldi, "Understanding Deep Image Representations by Inverting Them", CVPR 2015

Figure from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016.

DeepDream: Amplify Existing Features

Rather than synthesizing an image to maximize a specific neuron, instead try to **amplify** the neuron activations at some layer in the network



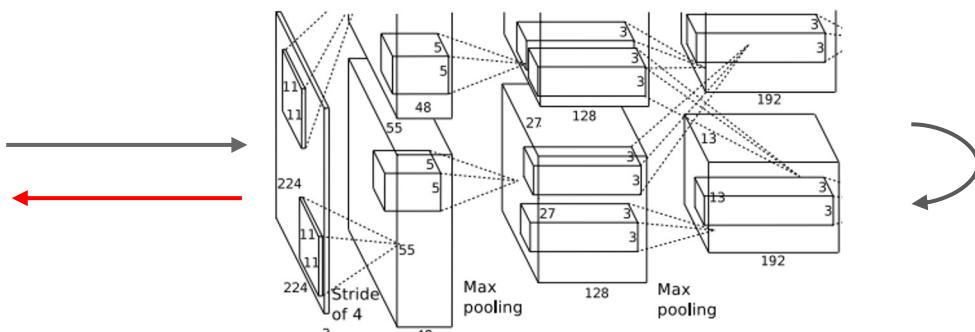
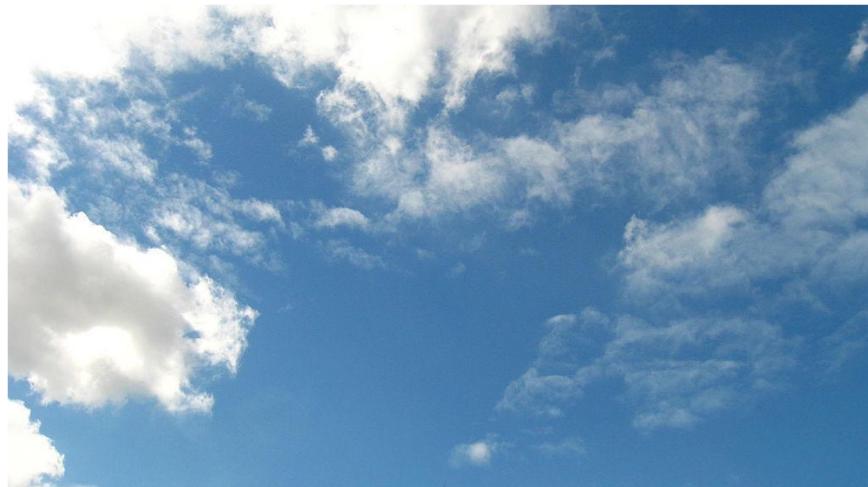
Choose an image and a layer in a CNN; repeat:

1. Forward: compute activations at chosen layer
2. Set gradient of chosen layer *equal to its activation*
3. Backward: Compute gradient on image
4. Update image

Mordvintsev, Olah, and Tyka, "Inceptionism: Going Deeper into Neural Networks", [Google Research Blog](#). Images are licensed under [CC-BY 4.0](#).

DeepDream: Amplify Existing Features

Rather than synthesizing an image to maximize a specific neuron, instead try to **amplify** the neuron activations at some layer in the network



Choose an image and a layer in a CNN; repeat:

1. Forward: compute activations at chosen layer
2. Set gradient of chosen layer *equal to its activation*
3. Backward: Compute gradient on image
4. Update image

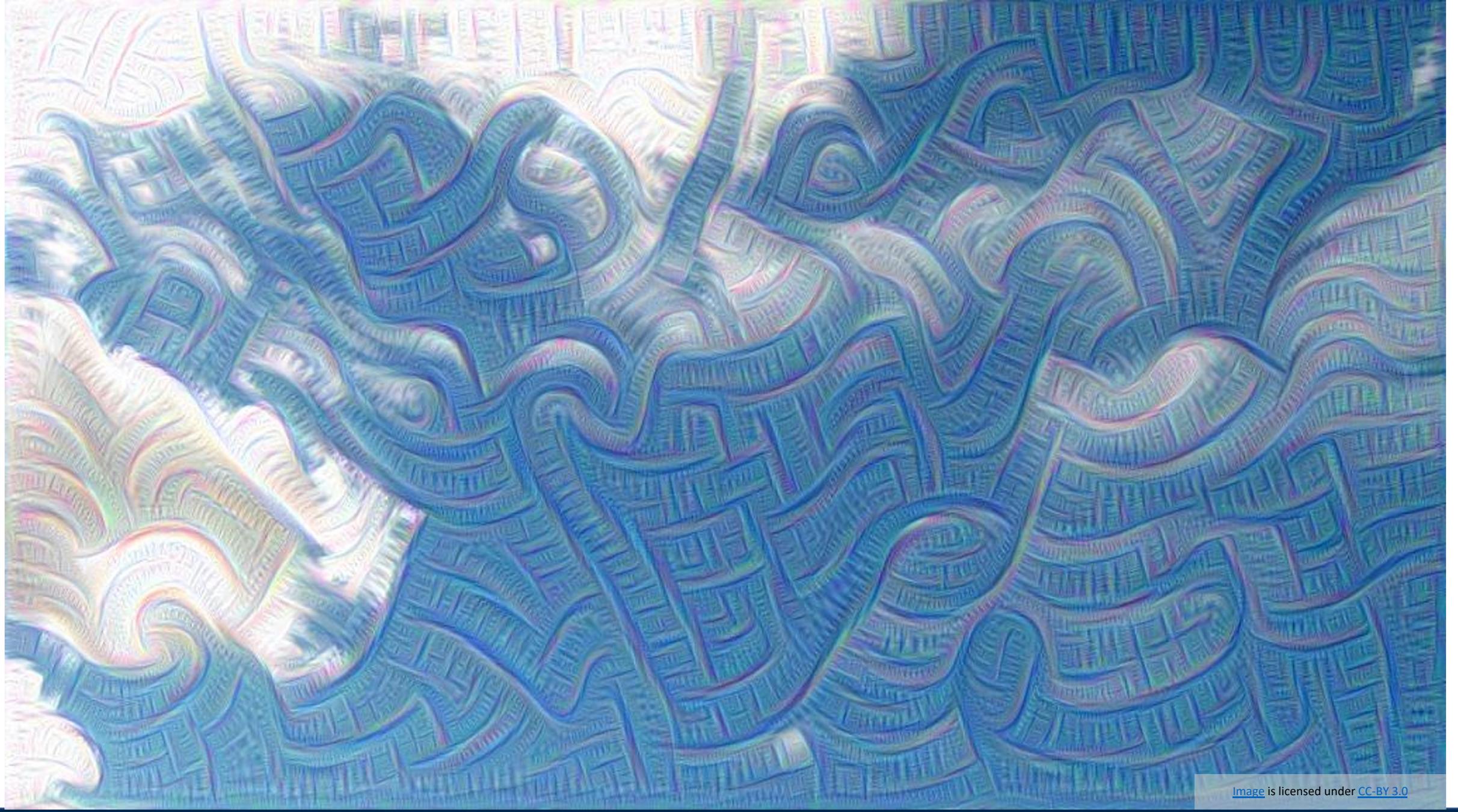
Equivalent to:

$$I^* = \arg \max_I \sum_i f_i(I)^2$$

Mordvintsev, Olah, and Tyka, "Inceptionism: Going Deeper into Neural Networks", [Google Research Blog](#). Images are licensed under [CC-BY 4.0](#).



[Sky image](#) is licensed under [CC-BY SA 3.0](#)



[Image](#) is licensed under [CC-BY 3.0](#)

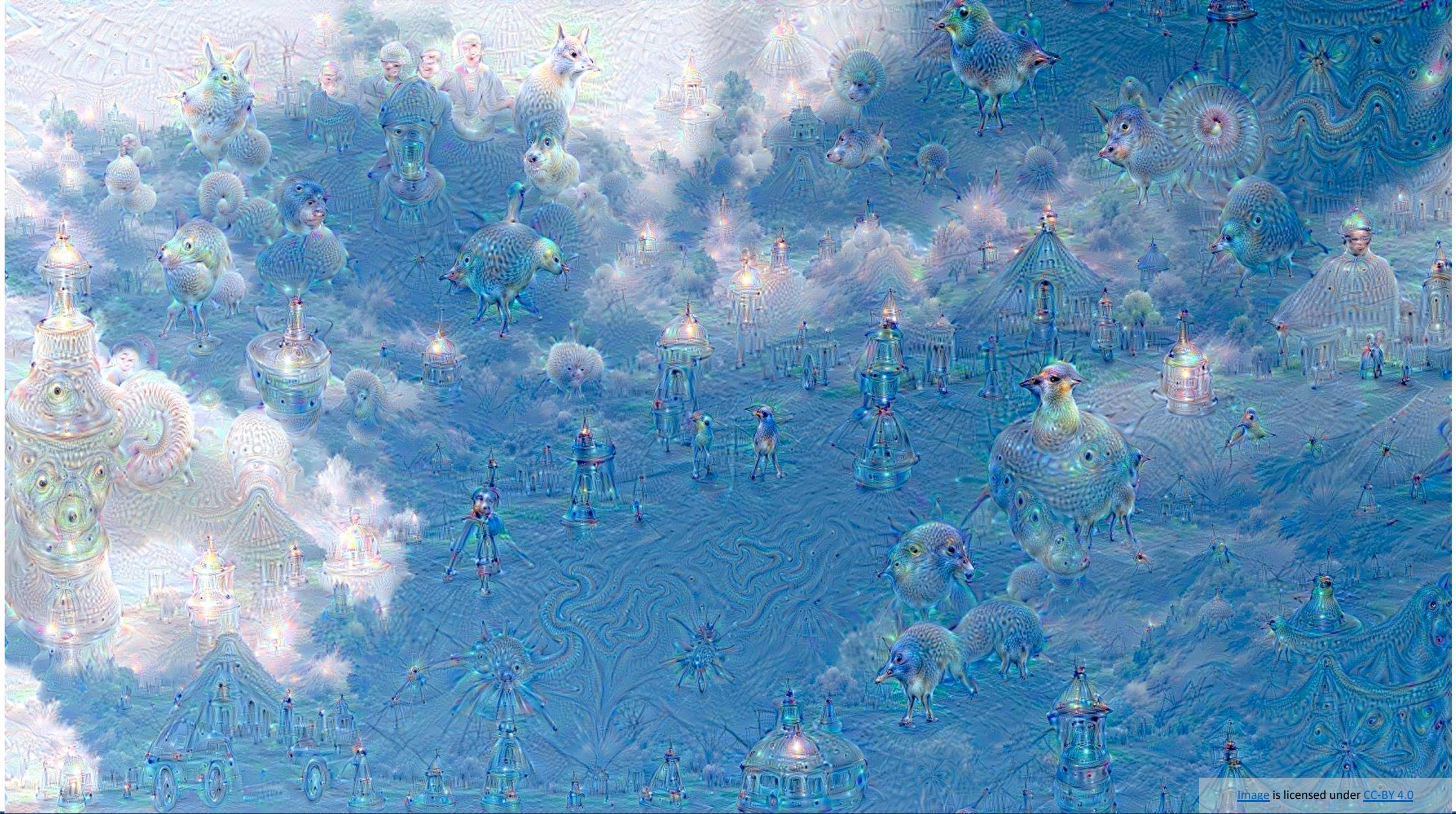
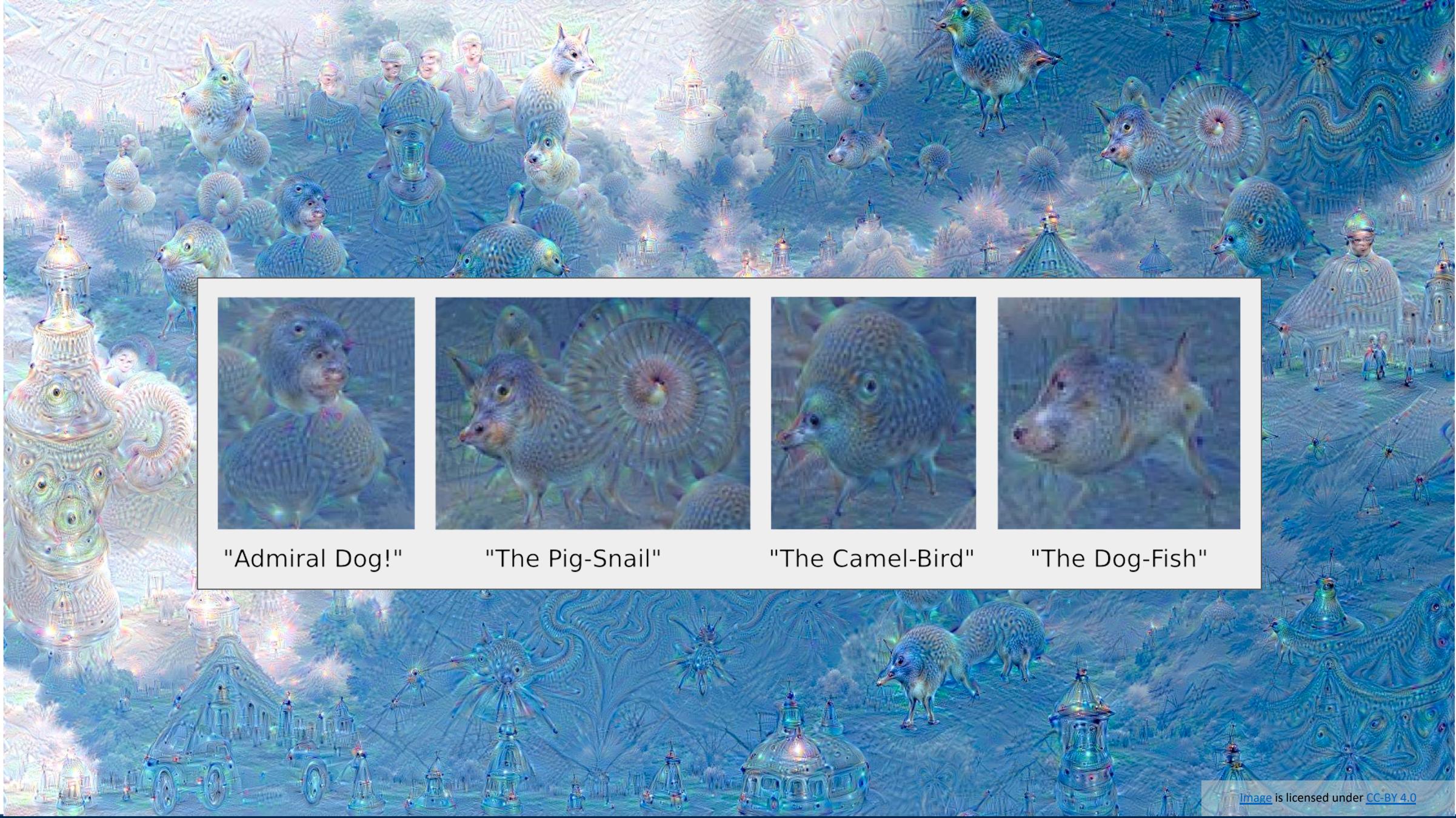


Image is licensed under CC-BY 4.0



"Admiral Dog!"



"The Pig-Snail"



"The Camel-Bird"

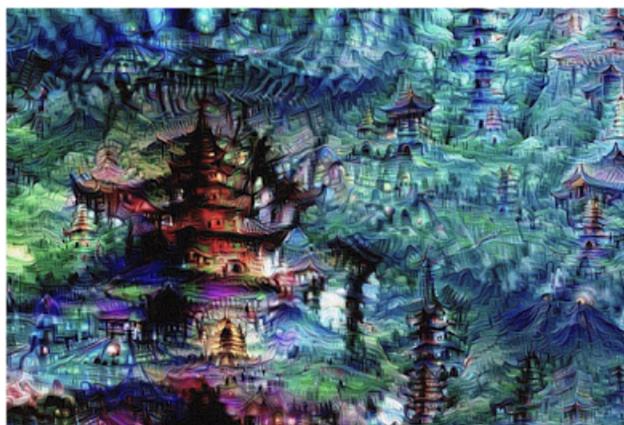
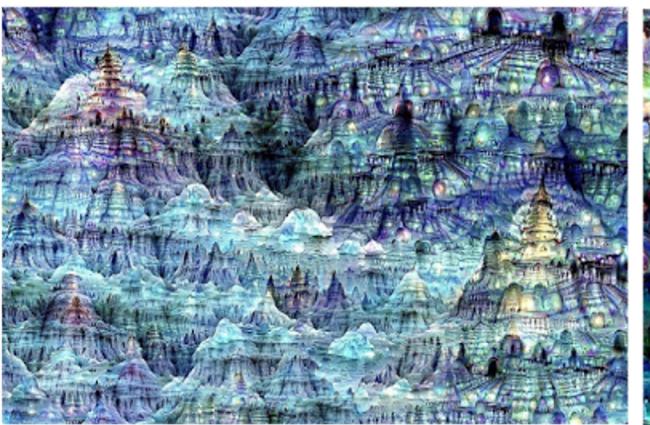
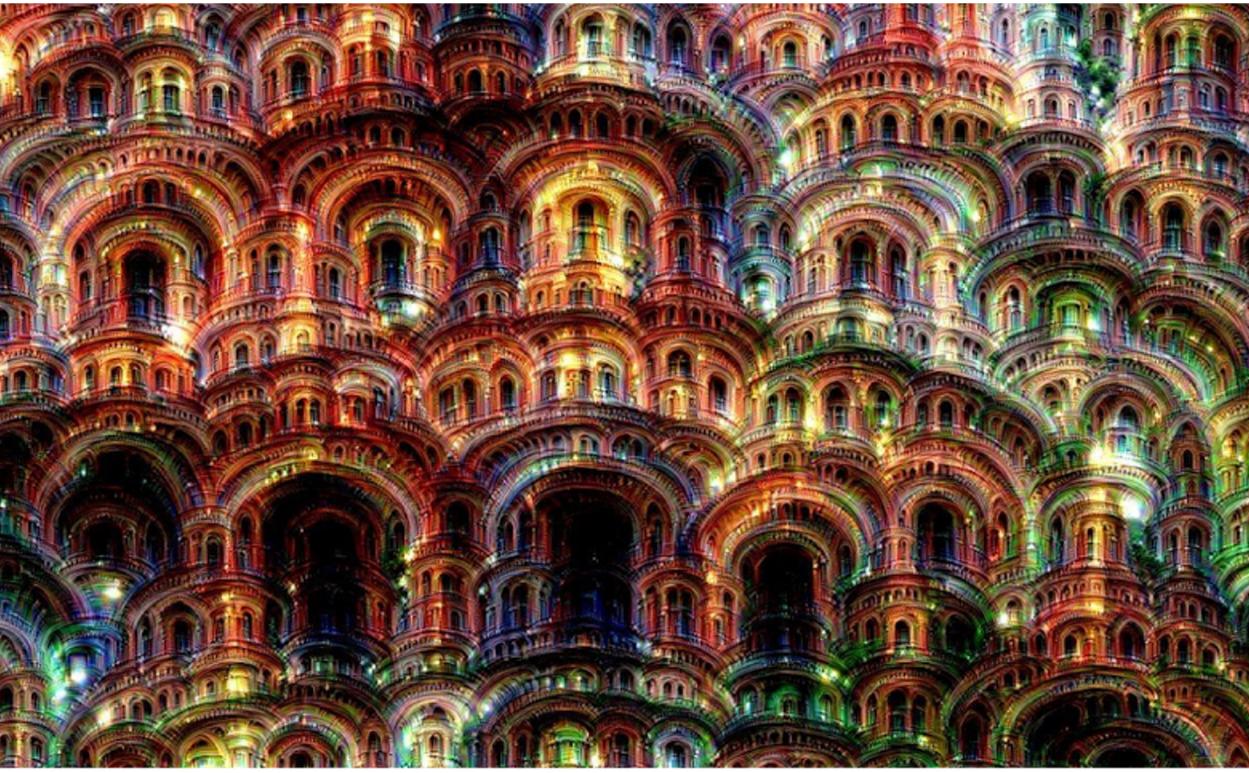
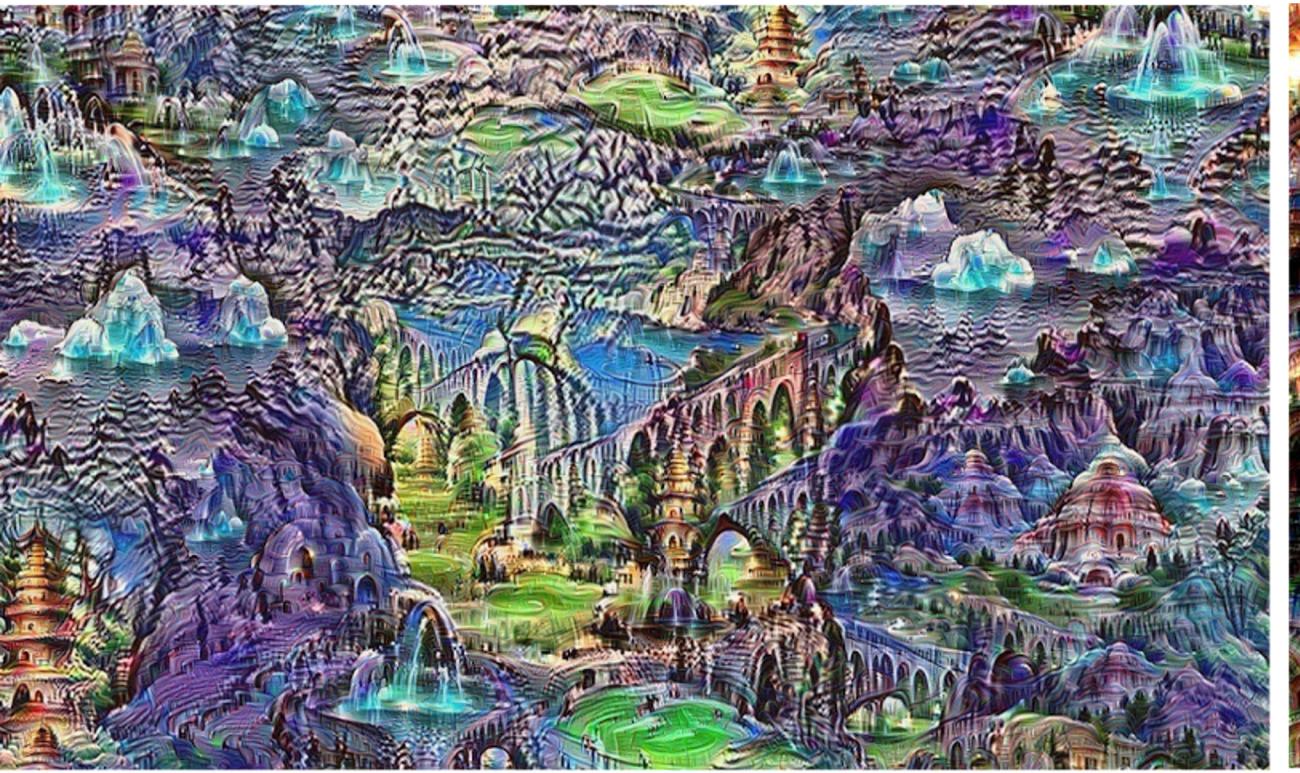


"The Dog-Fish"

[Image](#) is licensed under [CC-BY 4.0](#)



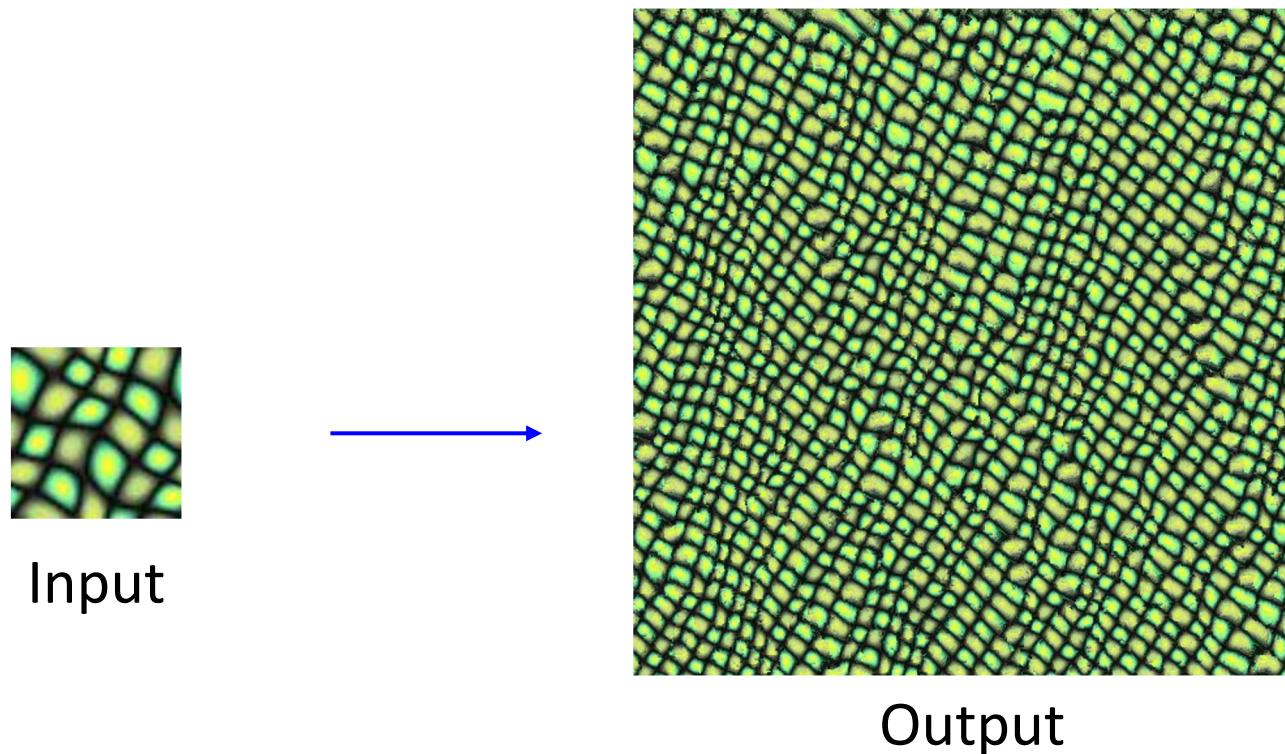
[Image](#) is licensed under [CC-BY 3.0](#)



[Image](#) is licensed under [CC-BY 4.0](#)

Texture Synthesis

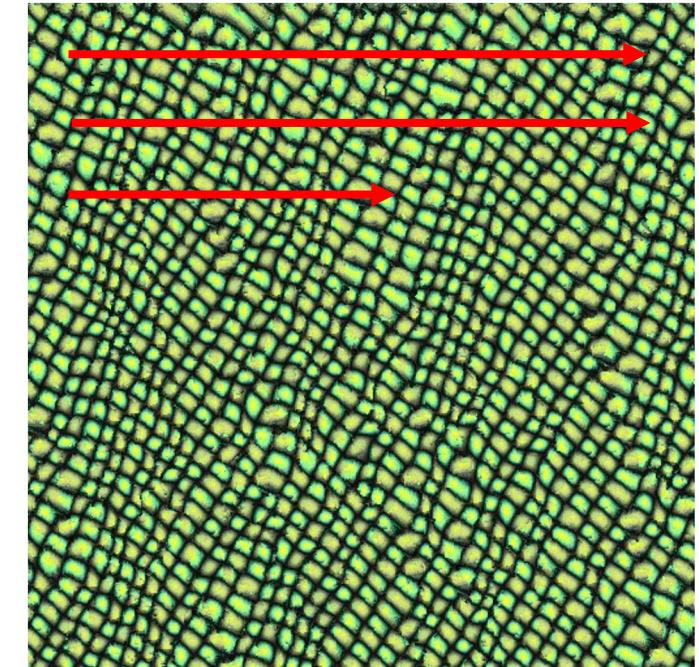
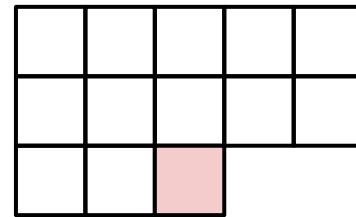
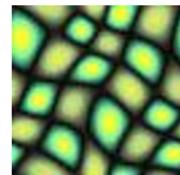
Given a sample patch of some texture, can we generate a bigger image of the same texture?



[Output image](#) is licensed under the [MIT license](#)

Texture Synthesis: Nearest Neighbor

Generate pixels one at a time in scanline order;
form neighborhood of already generated pixels
and copy nearest neighbor from input

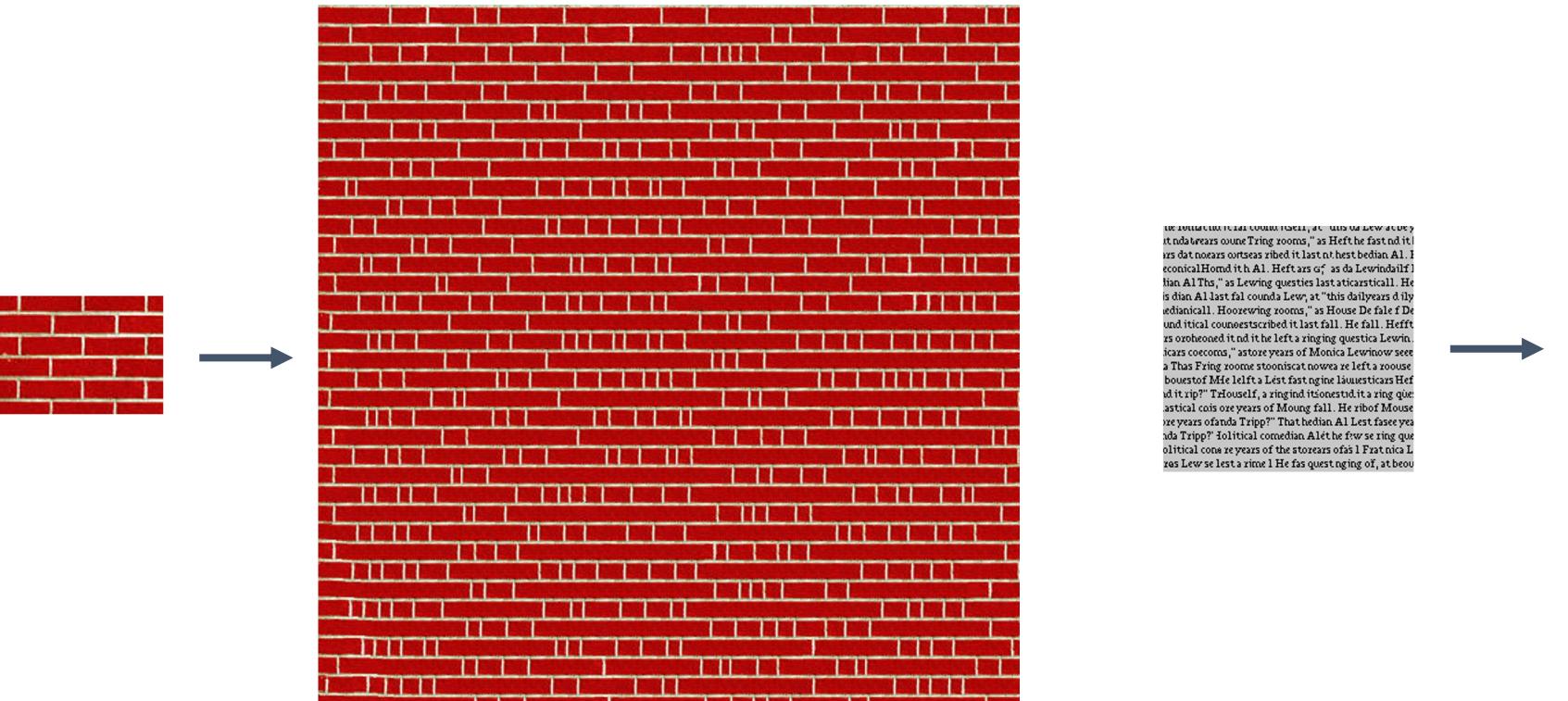


Wei and Levoy, "Fast Texture Synthesis using Tree-structured Vector Quantization", SIGGRAPH 2000

Efros and Leung, "Texture Synthesis by Non-parametric Sampling", ICCV 1999

[Output image](#) is licensed under the [MIT license](#)

Texture Synthesis: Nearest Neighbor

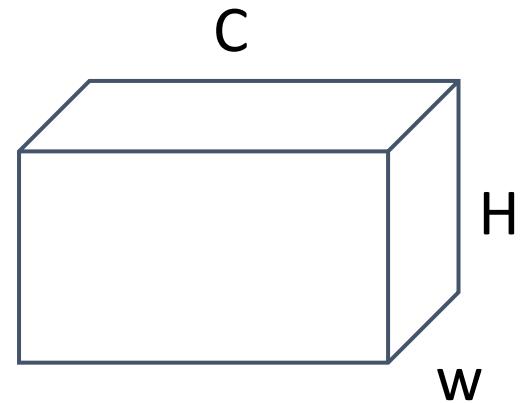
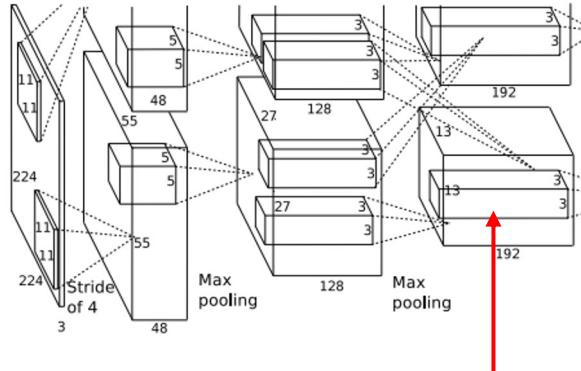


[Images](#) licensed under the [MIT license](#)

Texture Synthesis with Neural Networks: Gram Matrix



[This image](#) is in the public domain.



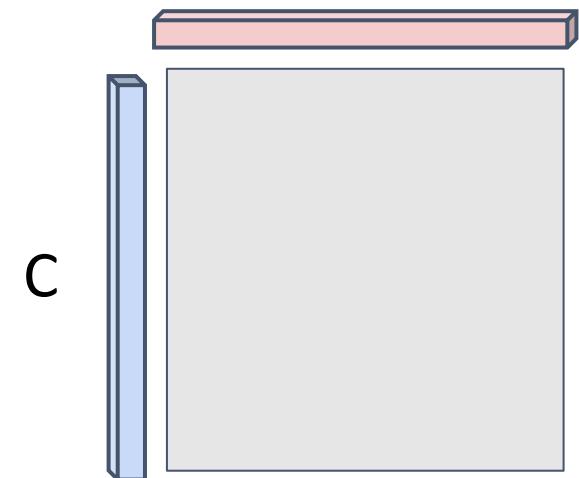
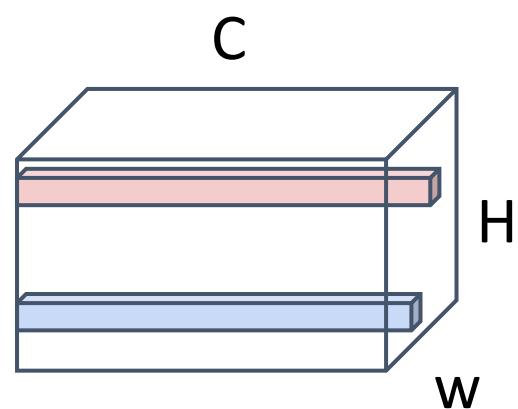
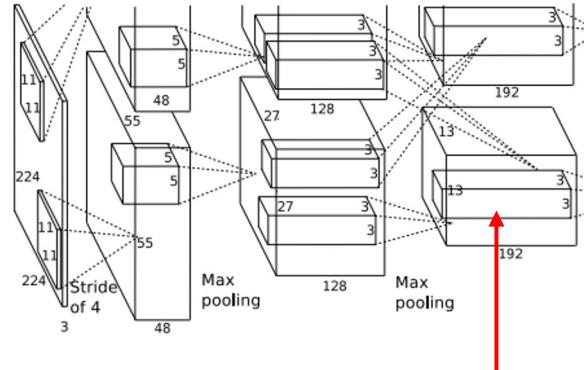
Each layer of CNN gives $C \times H \times W$ tensor of features; $H \times W$ grid of C -dimensional vectors

$$F^\ell \in \mathbb{R}^{C \times H \times W}$$

Texture Synthesis with Neural Networks: Gram Matrix



[This image](#) is in the public domain.



Each layer of CNN gives $C \times H \times W$ tensor of features; $H \times W$ grid of C -dimensional vectors

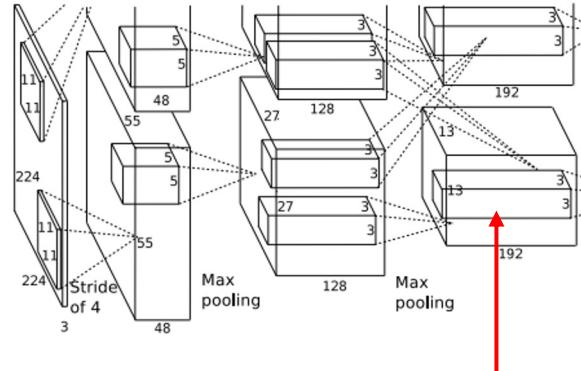
$$F^\ell \in \mathbb{R}^{C \times H \times W}$$

Outer product of two C -dimensional vectors gives $C \times C$ matrix of elementwise products

Texture Synthesis with Neural Networks: Gram Matrix



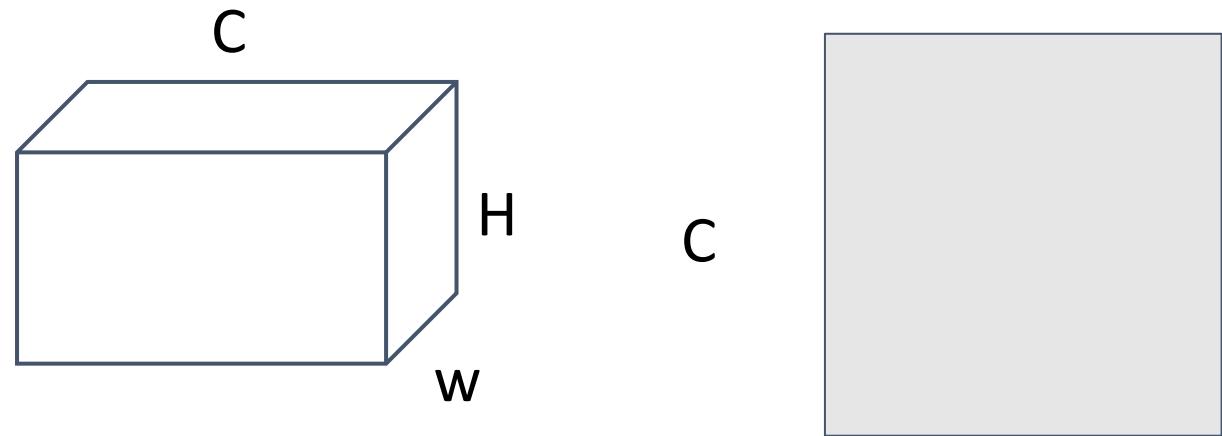
[This image](#) is in the public domain.



Each layer of CNN gives $C \times H \times W$ tensor of features; $H \times W$ grid of C -dimensional vectors

Outer product of two C -dimensional vectors gives $C \times C$ matrix of elementwise products

Average over all HW pairs gives **Gram Matrix** of shape $C \times C$ giving unnormalized covariance



$$F^\ell \in \mathbb{R}^{C \times H \times W}$$

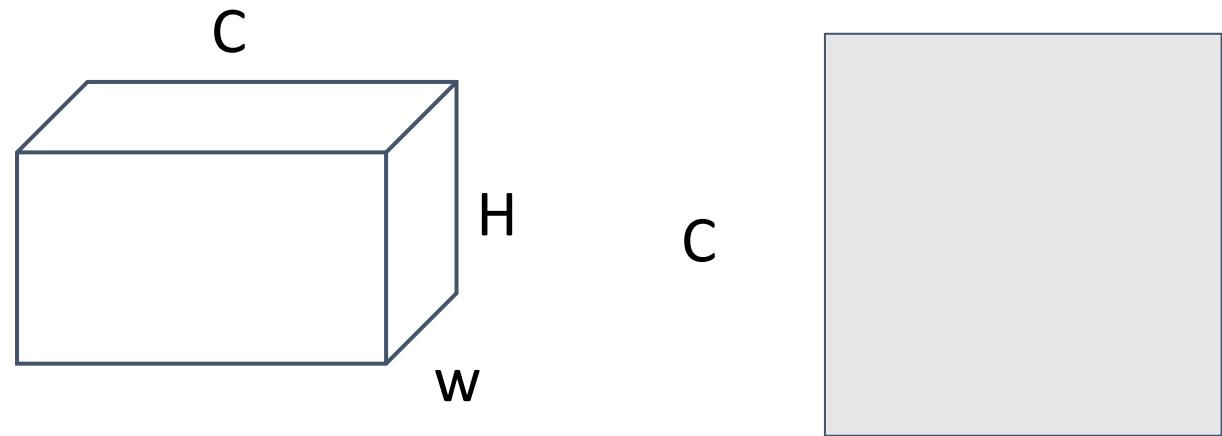
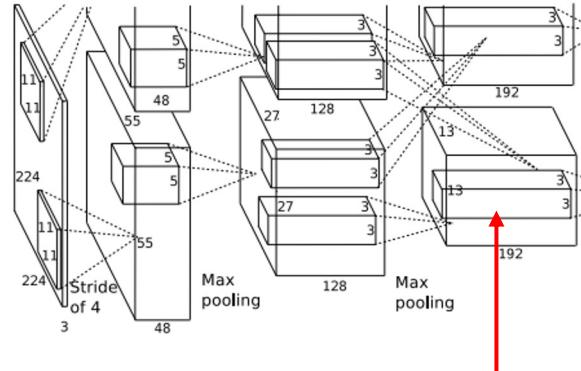
$$G^\ell \in \mathbb{R}^{C \times C}$$

$$G_{c,c'}^\ell = \sum_{h,w} F_{c,h,w}^\ell F_{c',h,w}^\ell$$

Texture Synthesis with Neural Networks: Gram Matrix



[This image](#) is in the public domain.



Each layer of CNN gives $C \times H \times W$ tensor of features; $H \times W$ grid of C -dimensional vectors

Efficient to compute;
reshape features from

Outer product of two C -dimensional vectors
gives $C \times C$ matrix of elementwise products

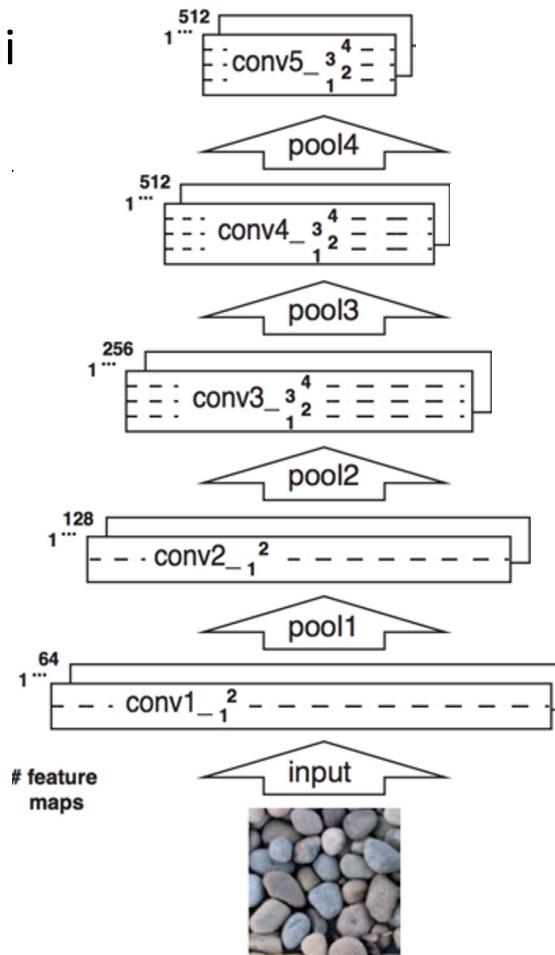
$C \times H \times W$ to $F = C \times HW$

Average over all HW pairs gives **Gram Matrix**
of shape $C \times C$ giving unnormalized covariance

then compute $G = FFT$

Neural Texture Synthesis

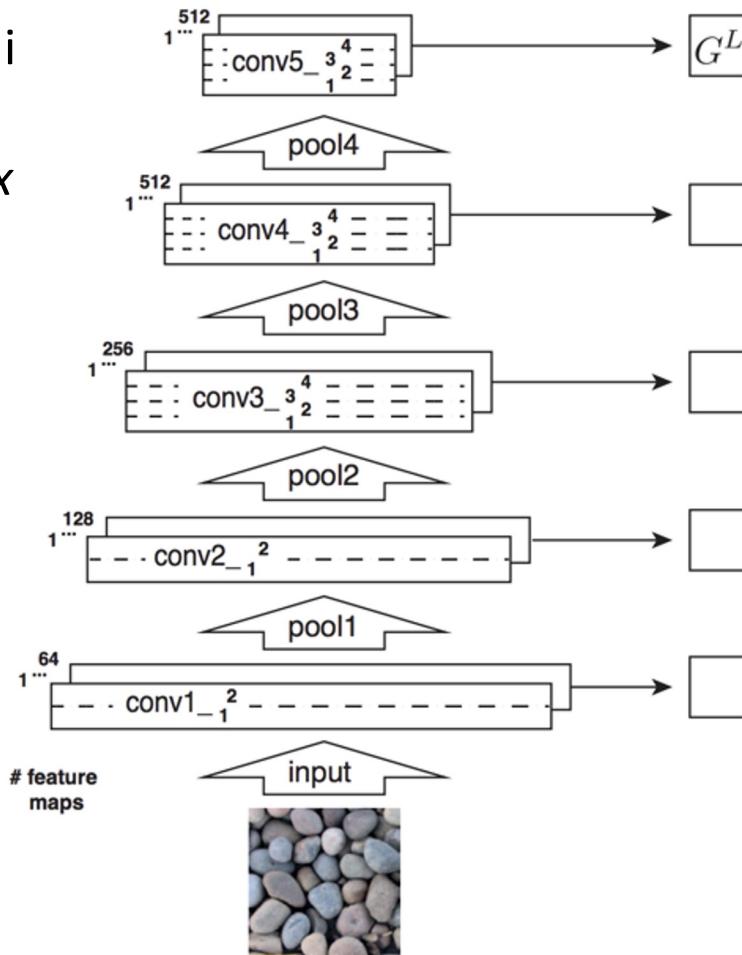
1. Pretrain a CNN on ImageNet (VGG-19)
2. Run texture forward through CNN,
record activations on every layer; layer i
gives features $F^\ell \in \mathbb{R}^{C_i \times H_i \times W_i}$



Neural Texture Synthesis

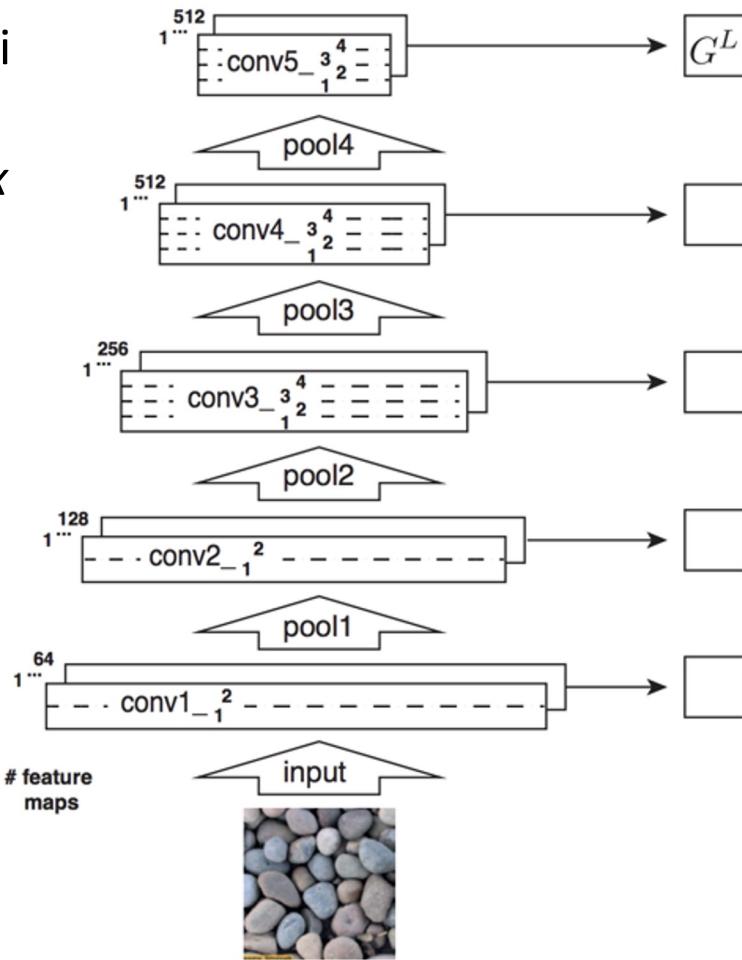
1. Pretrain a CNN on ImageNet (VGG-19)
2. Run texture forward through CNN,
record activations on every layer; layer i
gives features $F^\ell \in \mathbb{R}^{C_i \times H_i \times W_i}$
3. At each layer compute the *Gram matrix*
giving outer product of features:

$$G_{c,c'}^\ell = \sum_{h,w} F_{c,h,w}^\ell F_{c',h,w}^\ell \in \mathbb{R}^{C_\ell \times C_\ell}$$



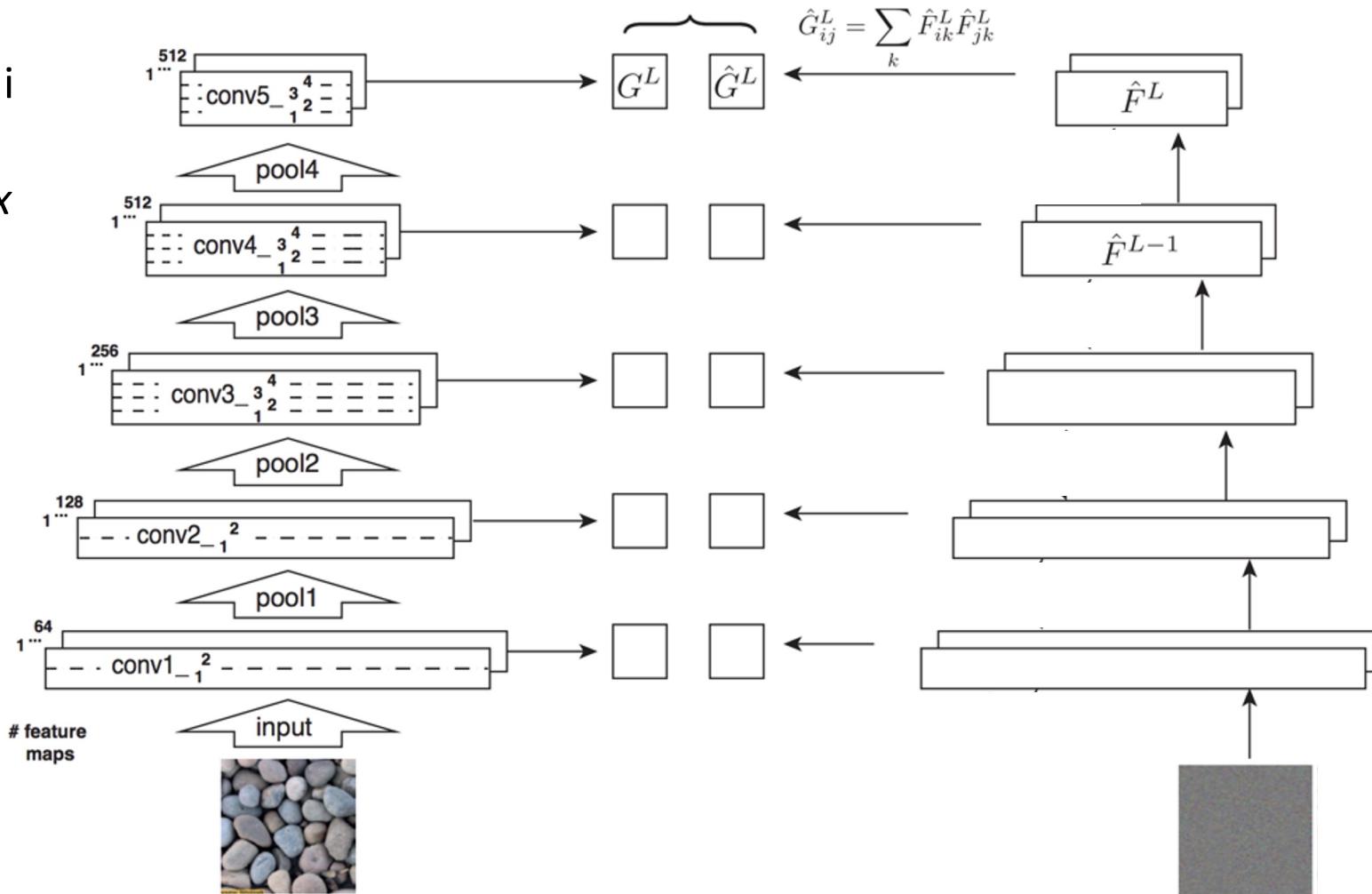
Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run texture forward through CNN, record activations on every layer; layer i gives features $F^\ell \in \mathbb{R}^{C_i \times H_i \times W_i}$
3. At each layer compute the *Gram matrix* giving outer product of features:
$$G_{c,c'}^\ell = \sum_{h,w} F_{c,h,w}^\ell F_{c',h,w}^\ell \in \mathbb{R}^{C_\ell \times C_\ell}$$
4. Initialize generated image from random noise



Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run texture forward through CNN, record activations on every layer; layer i gives features $F^{\ell} \in \mathbb{R}^{C_i \times H_i \times W_i}$
3. At each layer compute the *Gram matrix* giving outer product of features:
$$G_{c,c'}^{\ell} = \sum_{h,w} F_{c,h,w}^{\ell} F_{c',h,w}^{\ell} \in \mathbb{R}^{C_{\ell} \times C_{\ell}}$$
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer



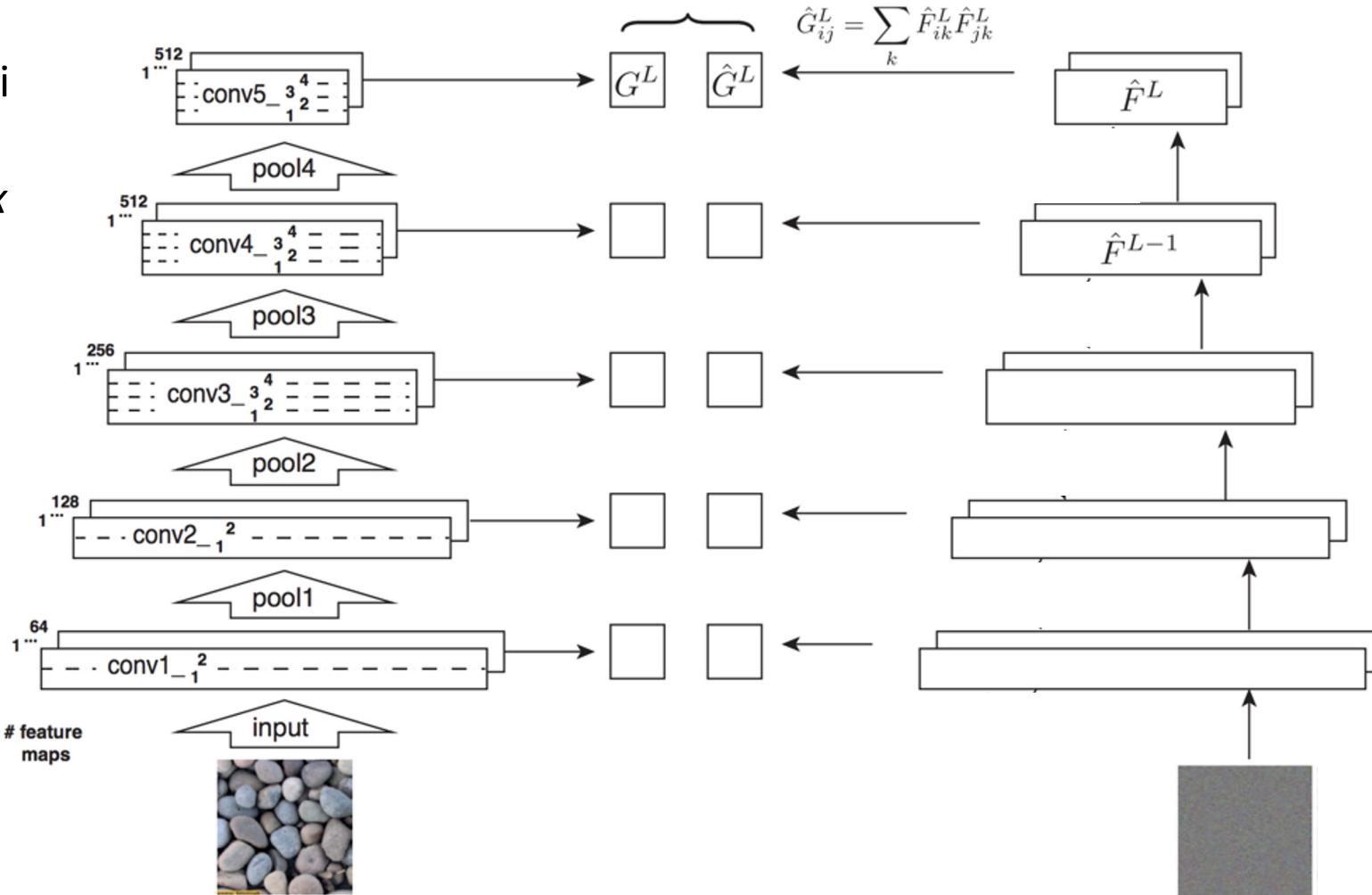
Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run texture forward through CNN, record activations on every layer; layer i gives features $F^\ell \in \mathbb{R}^{C_i \times H_i \times W_i}$
3. At each layer compute the *Gram matrix* giving outer product of features:

$$G_{c,c'}^\ell = \sum_{h,w} F_{c,h,w}^\ell F_{c',h,w}^\ell \in \mathbb{R}^{C_\ell \times C_\ell}$$
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices

$$E_\ell = \frac{1}{4N_\ell^2 M_\ell^2} \sum_{c,c'} (G_{c,c'}^\ell - \hat{G}_{c,c'}^\ell)^2$$

$$L = \sum_{\ell=0}^L w_\ell E_\ell$$



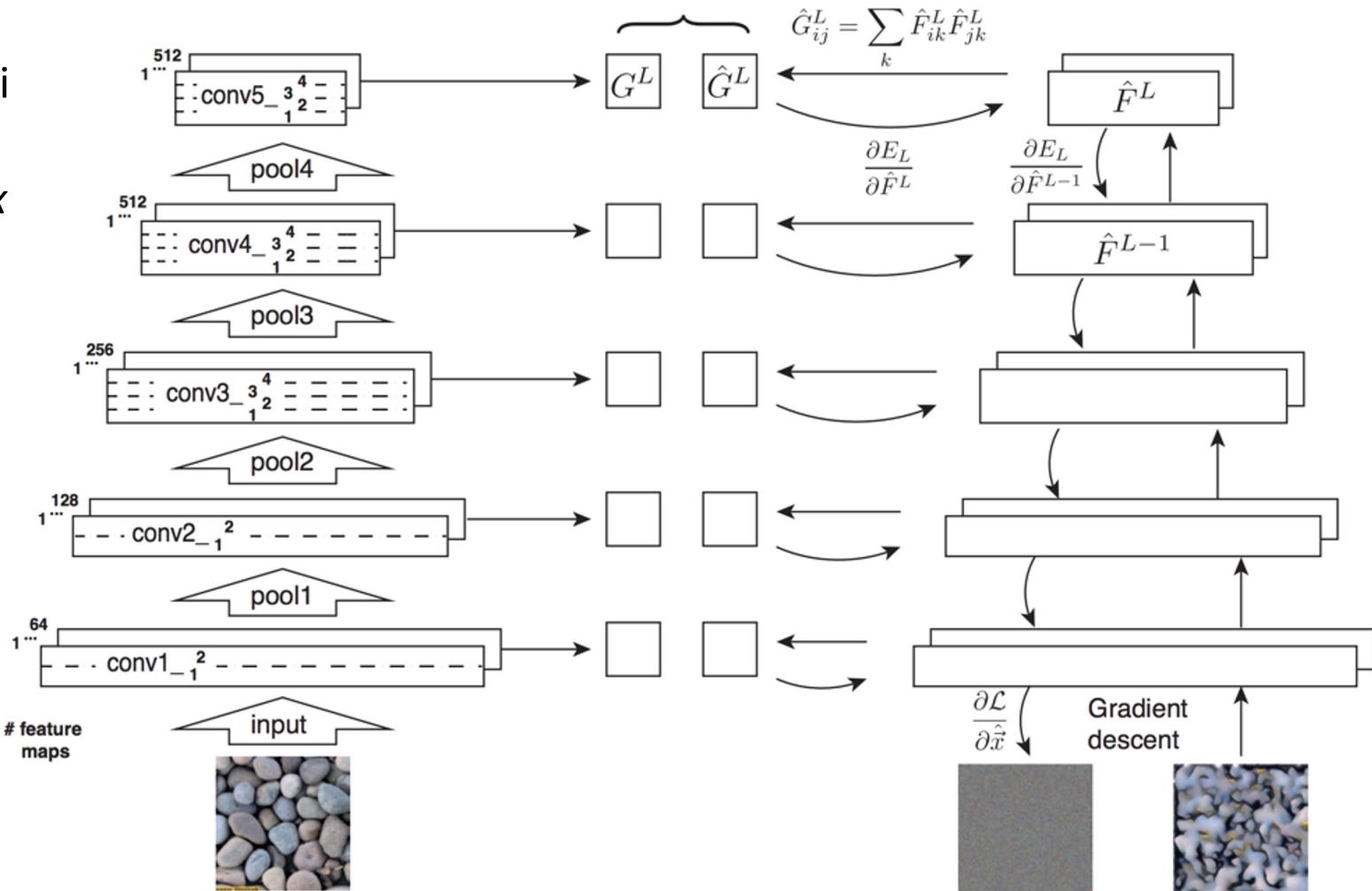
Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run texture forward through CNN, record activations on every layer; layer i gives features $F^\ell \in \mathbb{R}^{C_i \times H_i \times W_i}$
3. At each layer compute the *Gram matrix* giving outer product of features:

$$G_{c,c'}^\ell = \sum_{h,w} F_{c,h,w}^\ell F_{c',h,w}^\ell \in \mathbb{R}^{C_\ell \times C_\ell}$$
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5

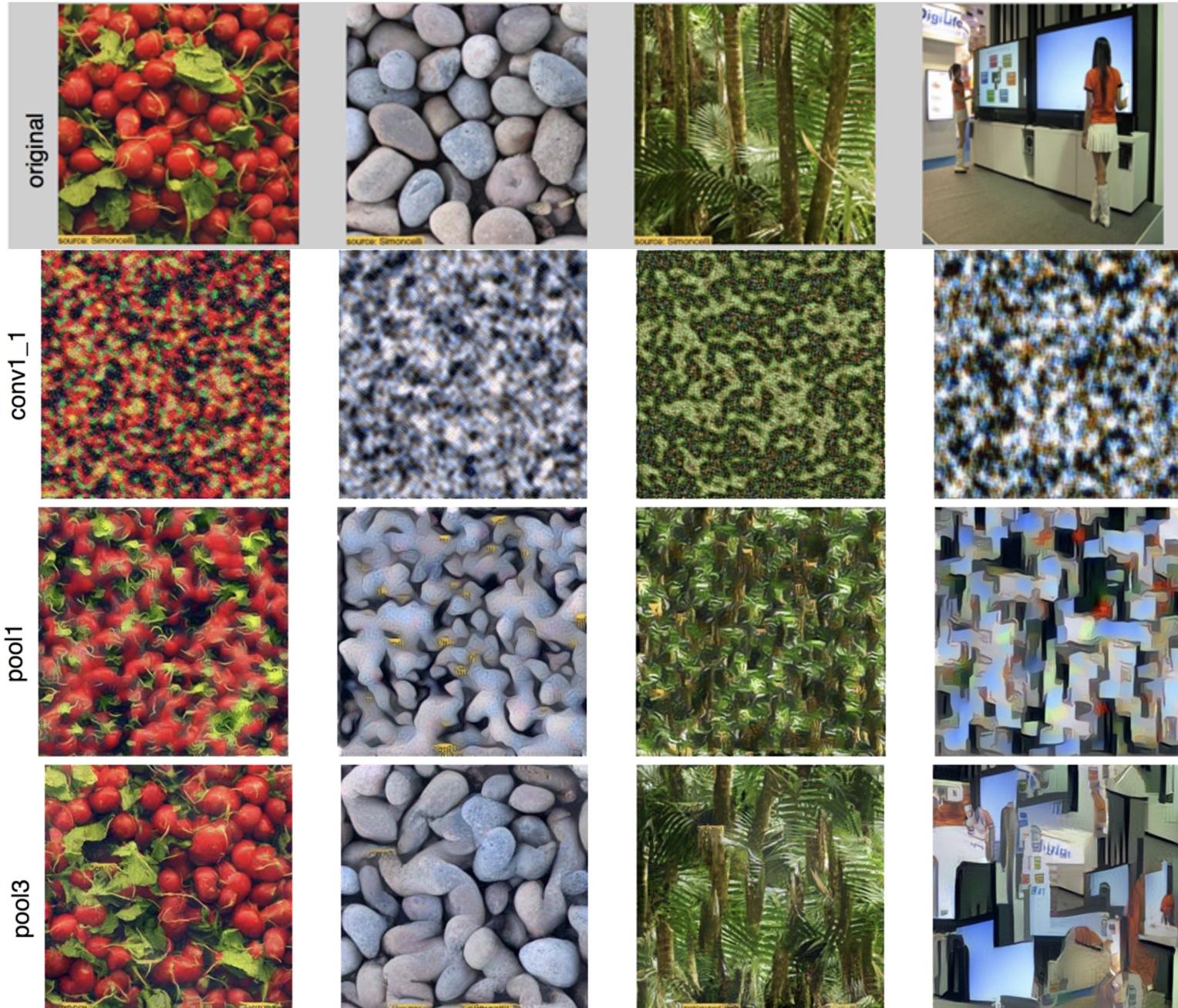
$$E_\ell = \frac{1}{4N_\ell^2 M_\ell^2} \sum_{c,c'} (G_{c,c'}^\ell - \hat{G}_{c,c'}^\ell)^2$$

$$L = \sum_{\ell=0}^L w_\ell E_\ell$$



Neural Texture Synthesis

Reconstructing texture
from higher layers
recovers larger features
from the input texture



Gatys, Ecker, and Bethge, "Texture Synthesis Using Convolutional Neural Networks", NIPS 2015

Neural Texture Synthesis: Texture = Artwork

Texture
synthesis (Gram
reconstruction)

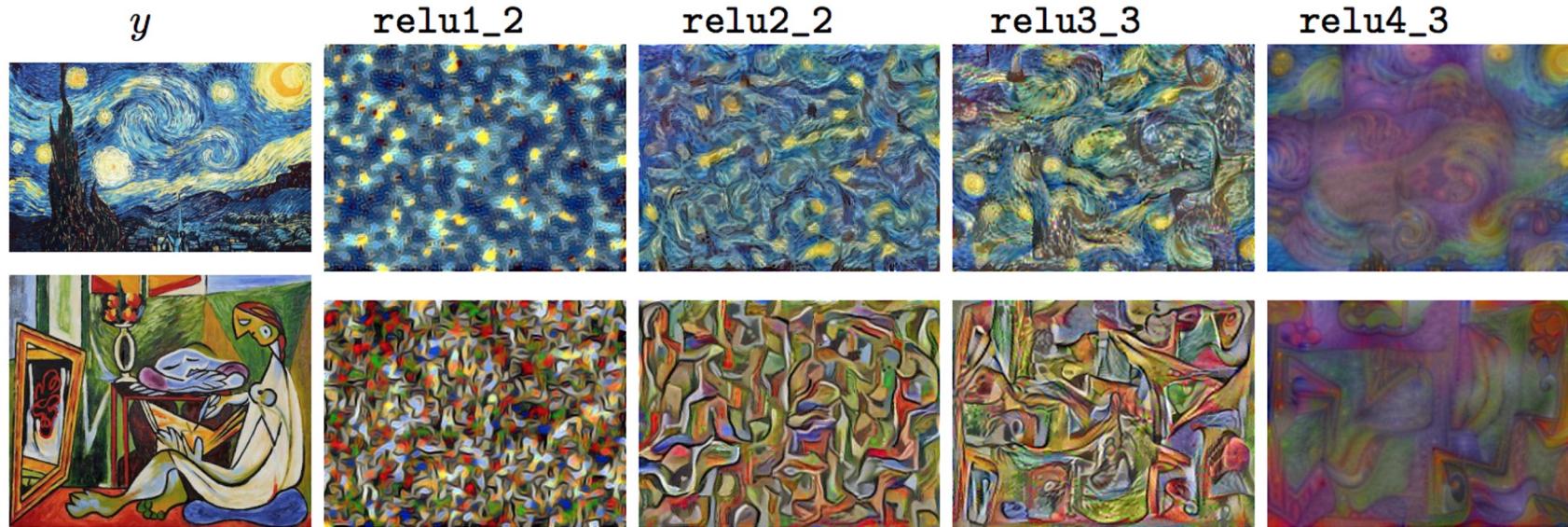
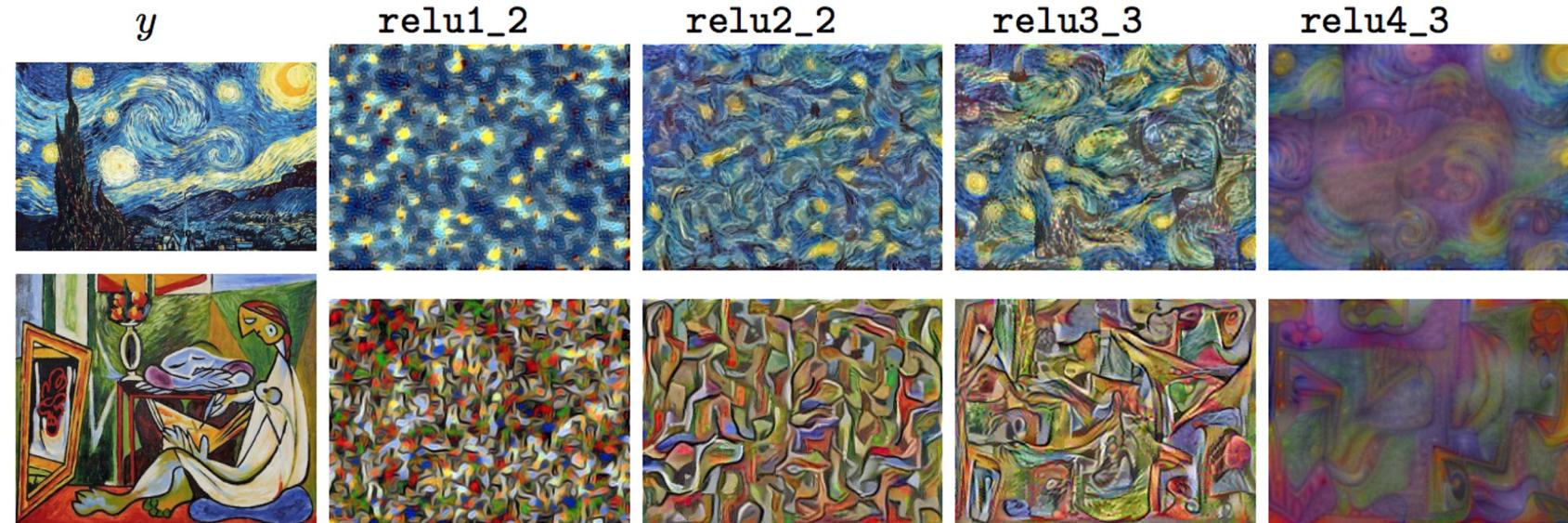


Figure from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016.

Neural Style Transfer: Feature + Gram Reconstruction

Texture
synthesis (Gram
reconstruction)



Feature
reconstruction

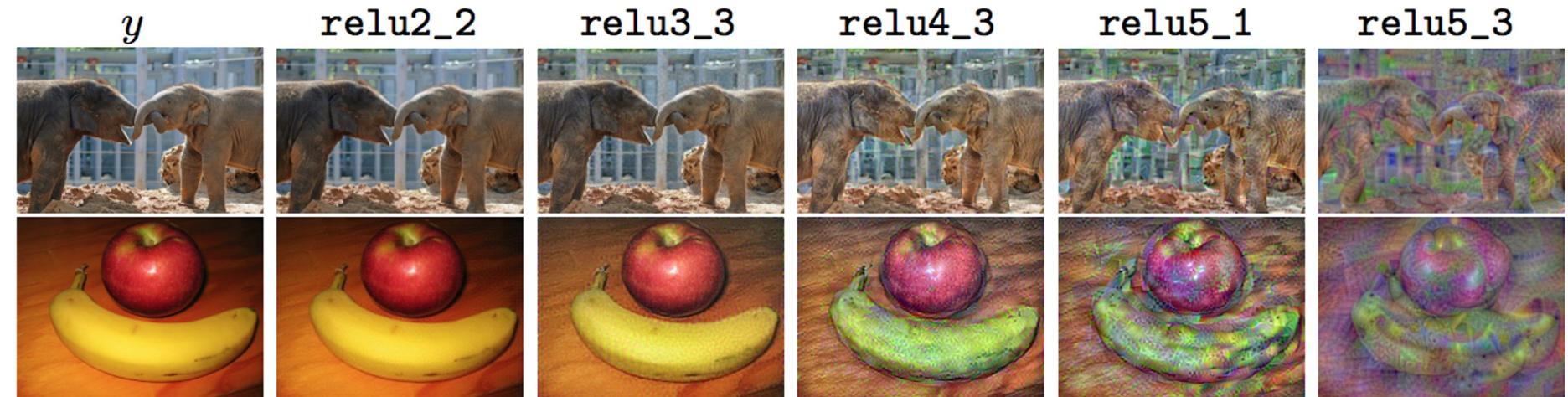


Figure from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016.

Neural Style Transfer

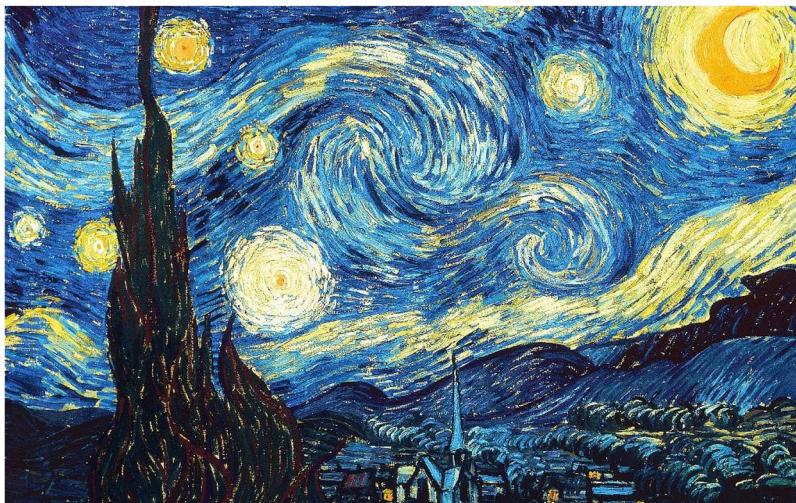
Content Image



[This image](#) is licensed under [CC-BY 3.0](#)

+

Style Image



[Starry Night](#) by Van Gogh is in the public domain

=

Output Image

Match features
from content
image and Gram
matrices from
style image

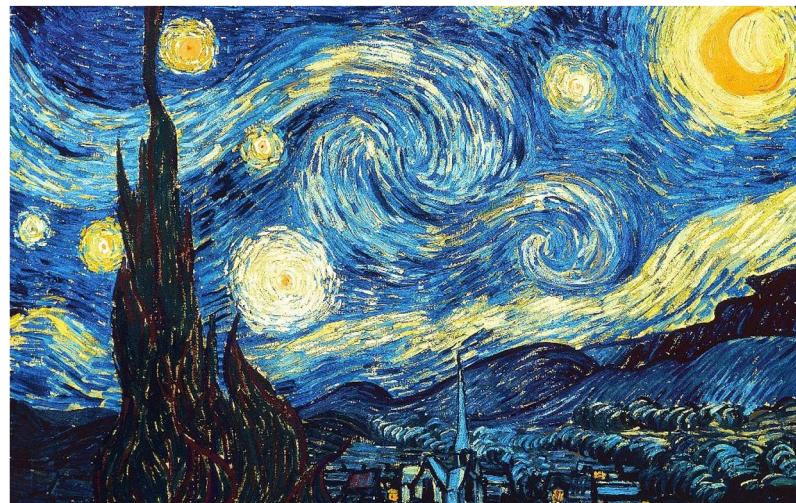
Neural Style Transfer

Content Image



+

Style Image



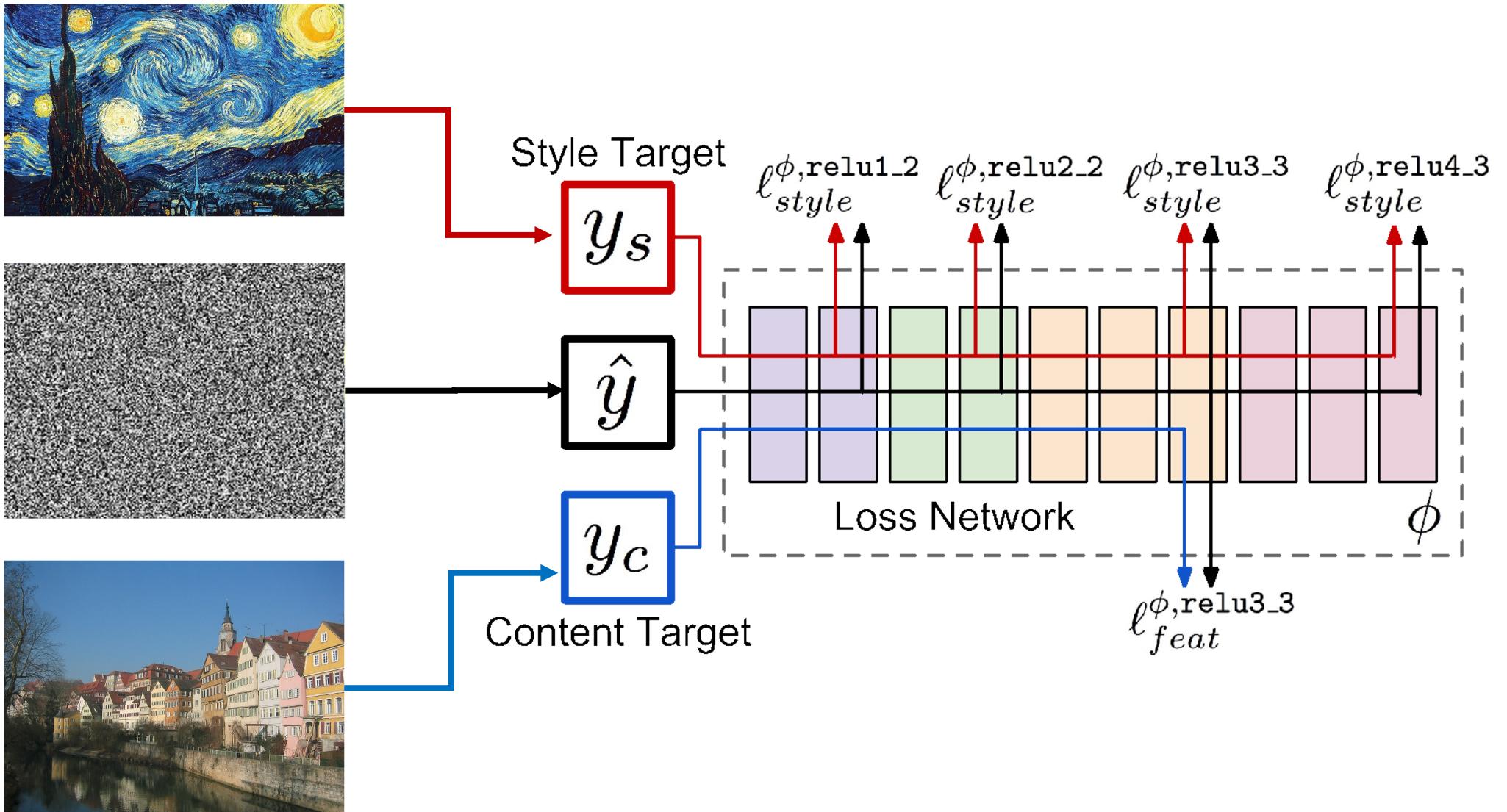
[Starry Night](#) by Van Gogh is in the public domain

=



[This image](#) copyright Justin Johnson, 2015. Reproduced with permission.

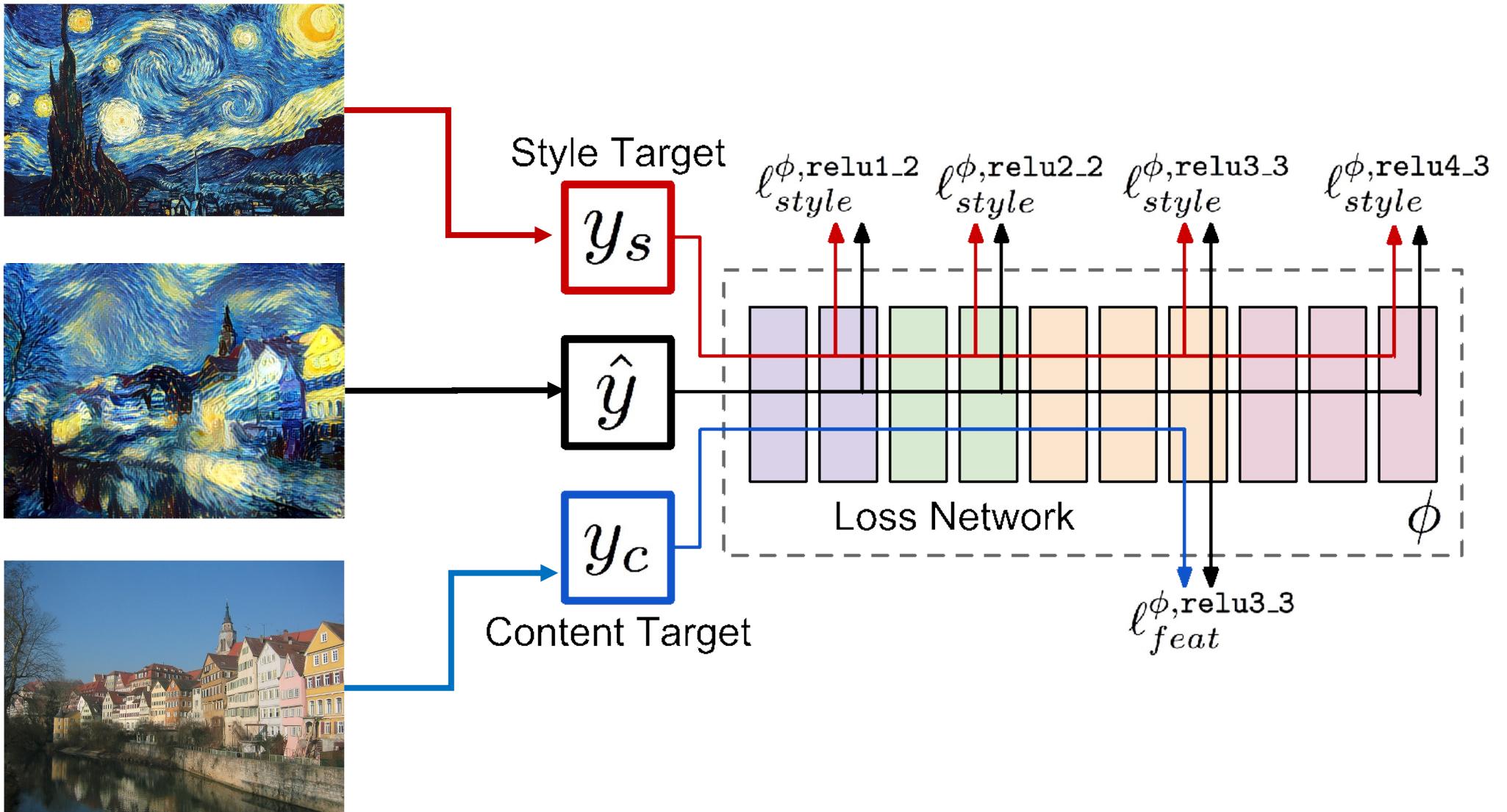
Style image
Output image
(Start with noise)
Content image



Gatys, Ecker, and Bethge, "Image style transfer using convolutional neural networks", CVPR 2016

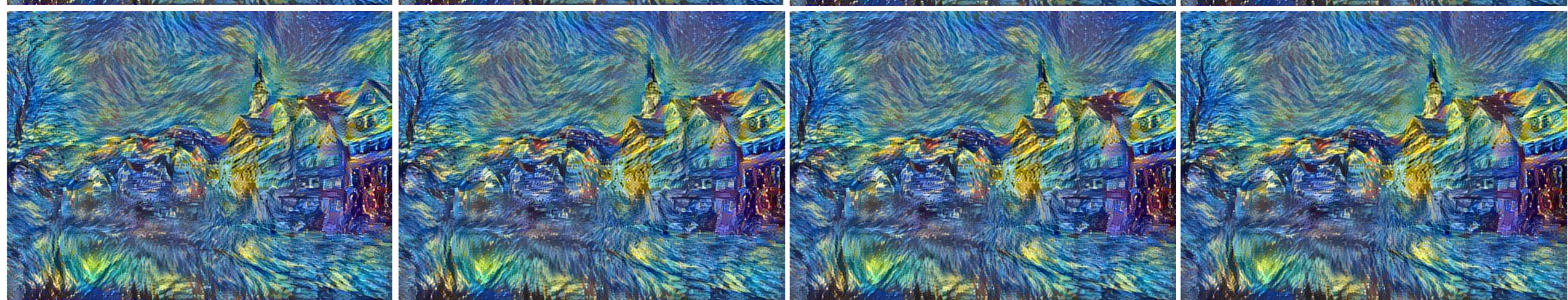
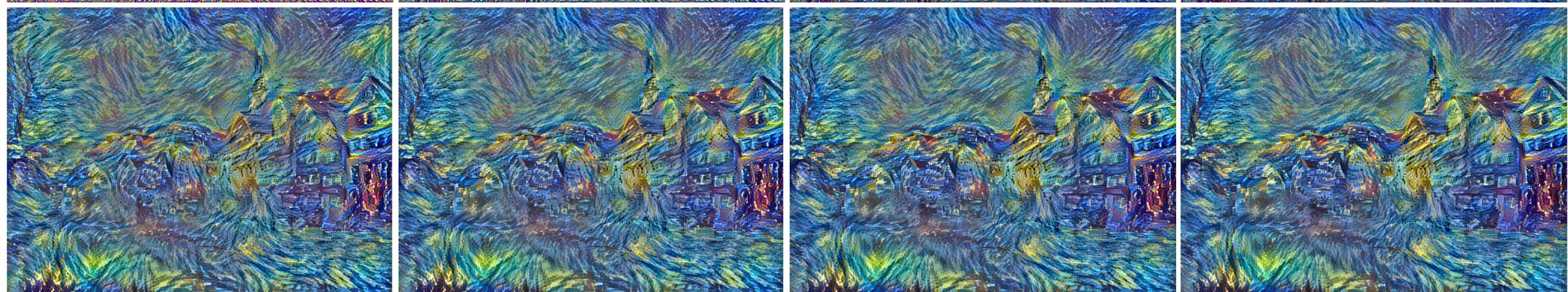
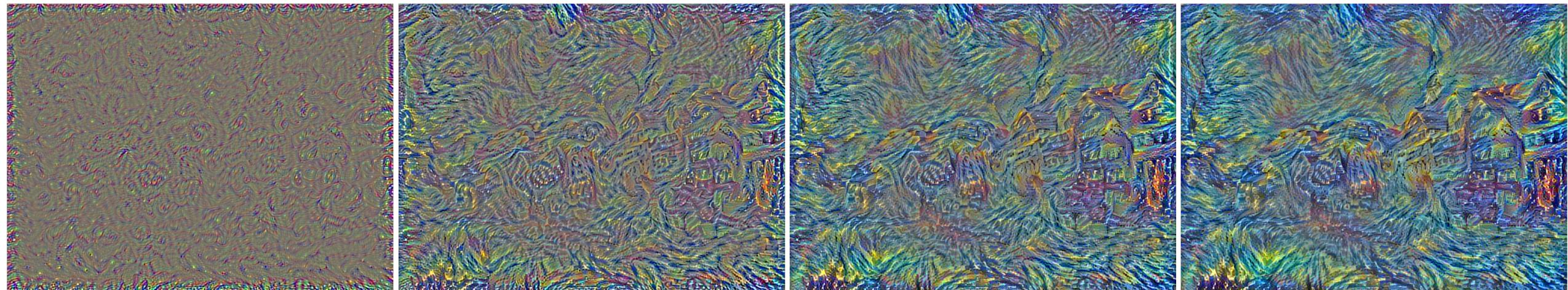
Figure adapted from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016.

Style image
Output image
Content image



Gatys, Ecker, and Bethge, "Image style transfer using convolutional neural networks", CVPR 2016

Figure adapted from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016.



Neural Style Transfer

Example outputs
from [my
implementation](#)
(in Lua Torch)



Gatys, Ecker, and Bethge, "Image style transfer using convolutional neural networks", CVPR 2016
Figure copyright Justin Johnson, 2015.

Neural Style Transfer



More weight to
content loss

More weight to
style loss

Neural Style Transfer

Resizing style image before running style transfer algorithm can transfer different types of features



Larger style image

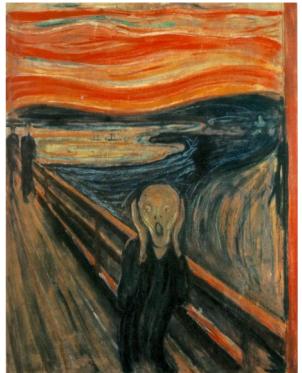


Smaller style image

Gatys, Ecker, and Bethge, "Image style transfer using convolutional neural networks", CVPR 2016
Figure copyright Justin Johnson, 2015.

Neural Style Transfer: Multiple Style Images

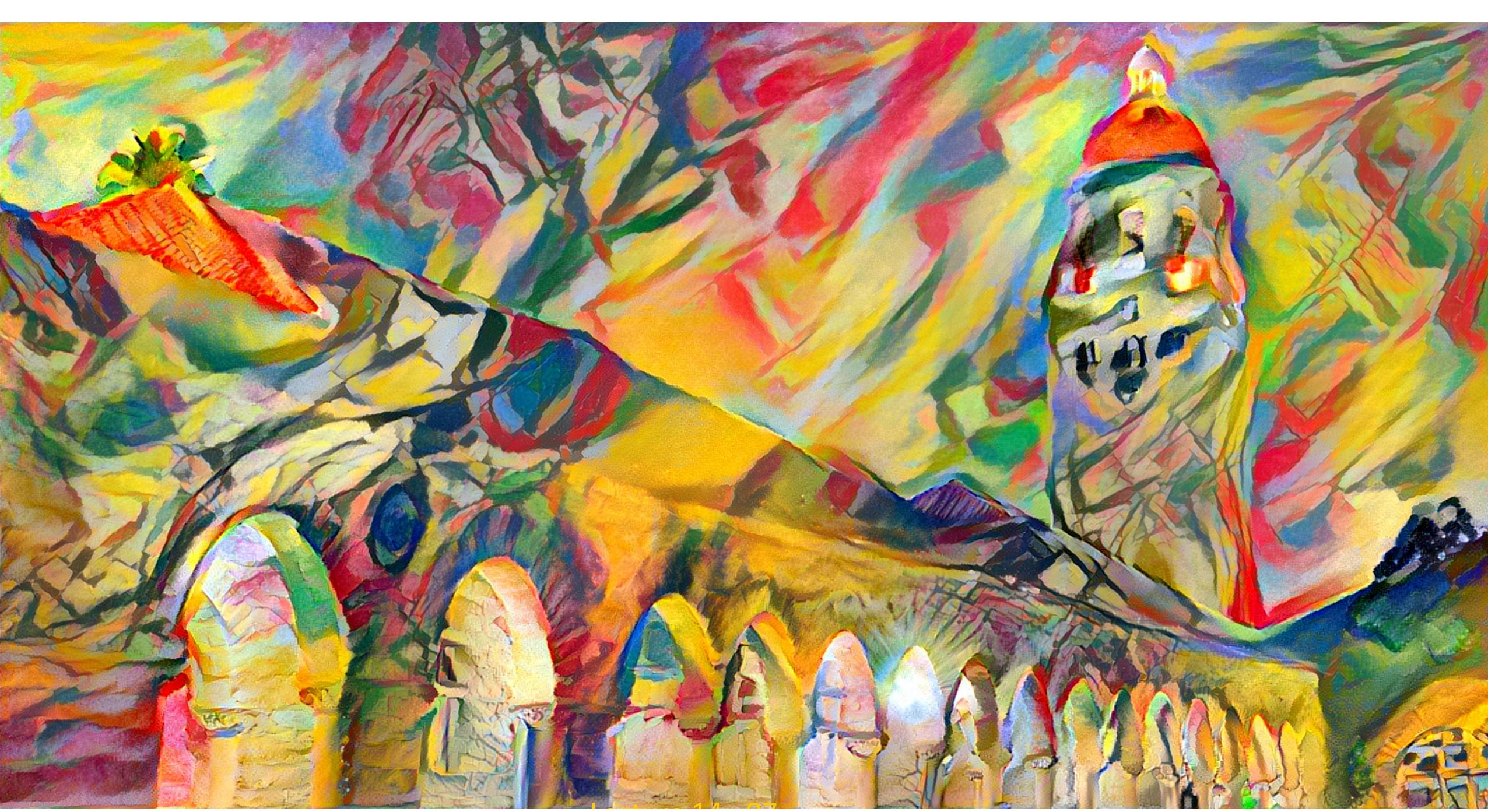
Mix style from
multiple images by
taking a weighted
average of Gram
matrices



Gatys, Ecker, and Bethge, "Image style transfer using convolutional neural networks", CVPR 2016
Figure copyright Justin Johnson, 2015.









Neural Style Transfer

Problem: Style transfer requires many forward / backward passes through VGG; very slow!

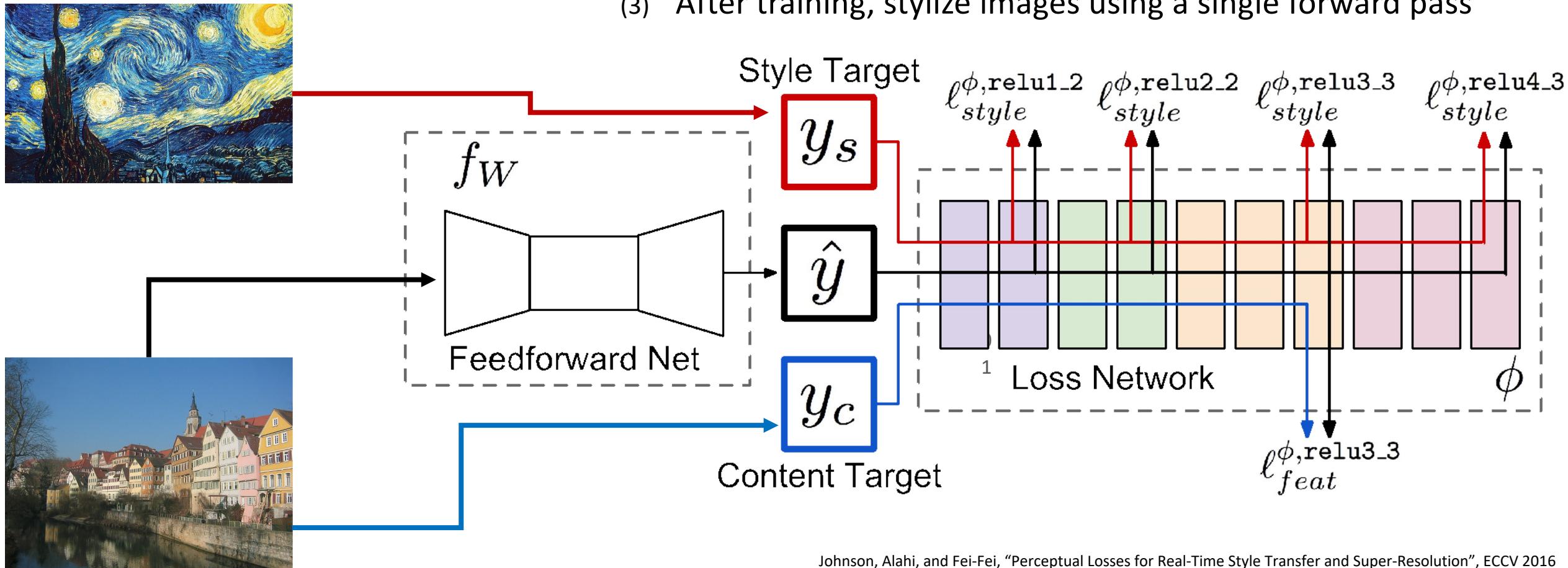
Neural Style Transfer

Problem: Style transfer requires many forward / backward passes through VGG; very slow!

Solution: Train another neural network to perform style transfer for us!

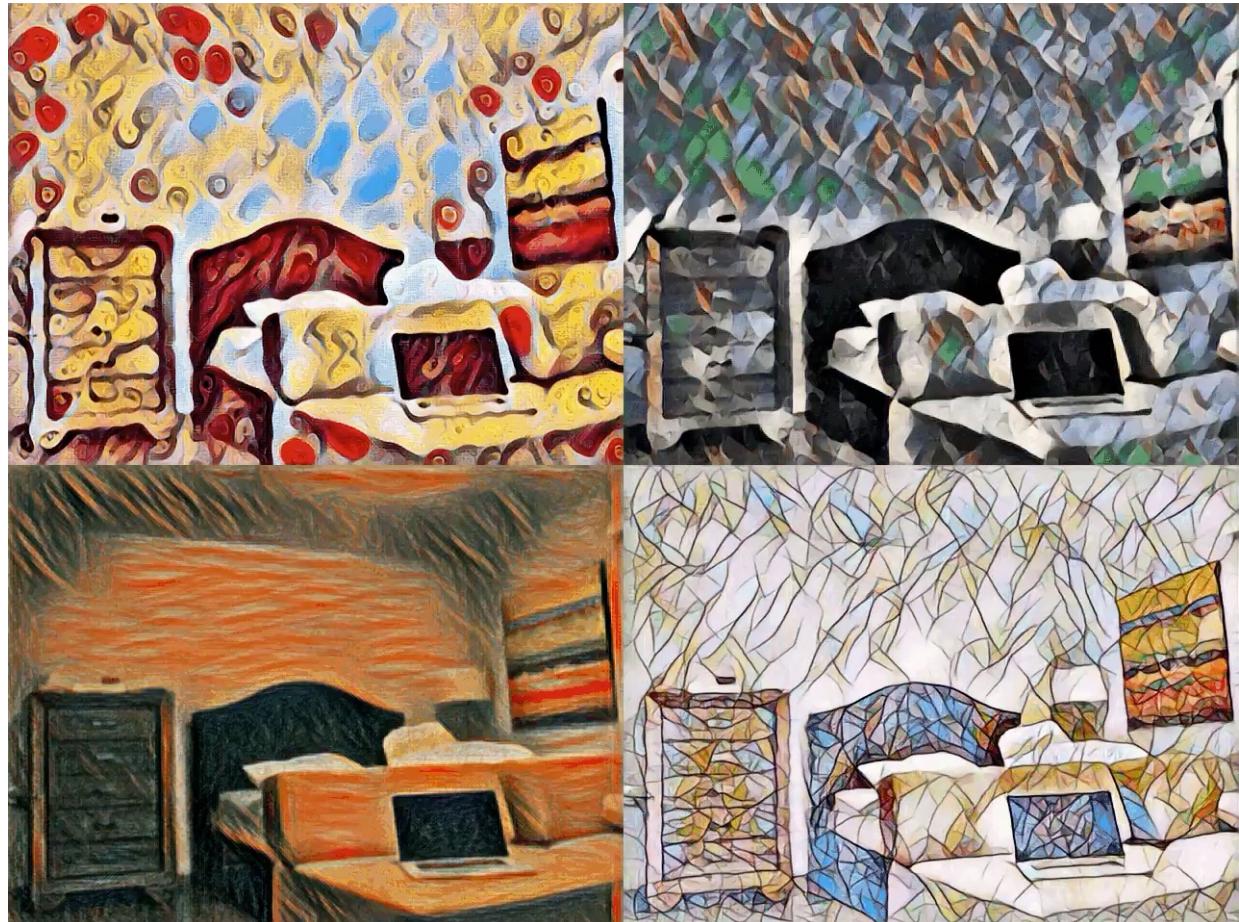
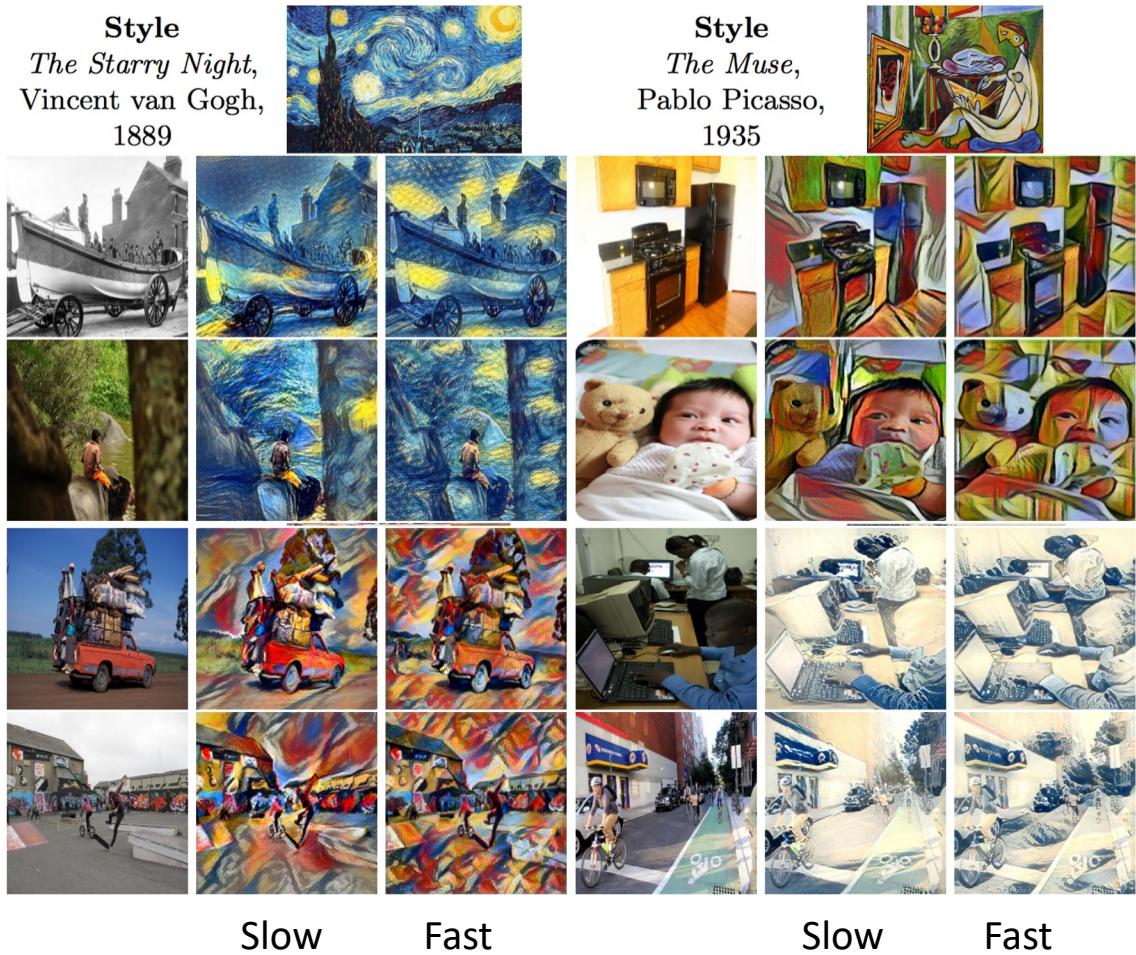
Fast Neural Style Transfer

- (1) Train a feedforward network for each style
- (2) Use pretrained CNN to compute same losses as before
- (3) After training, stylize images using a single forward pass



Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016

Fast Neural Style Transfer

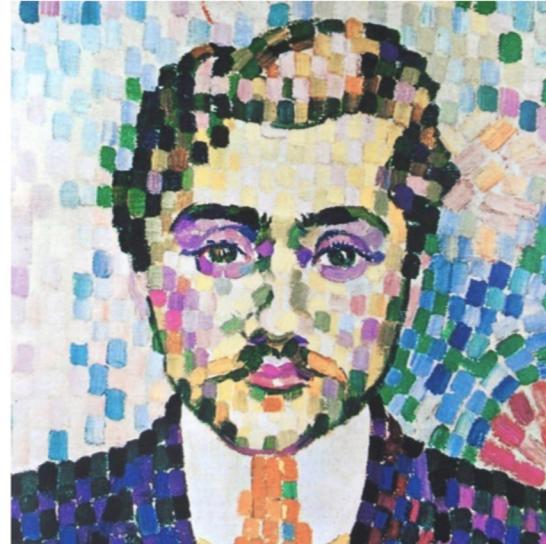
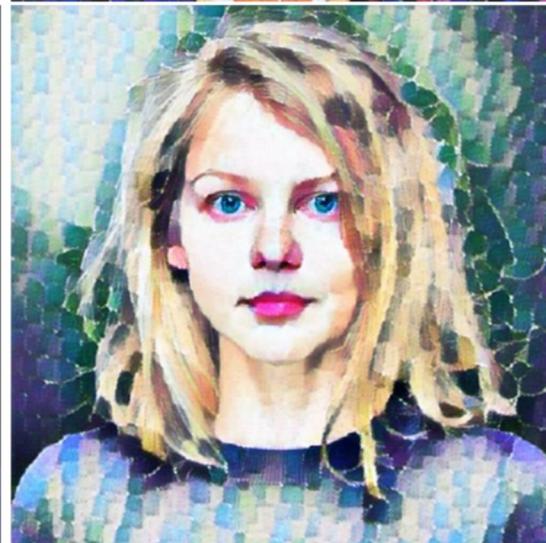


<https://github.com/jcjohnson/fast-neural-style>

Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016

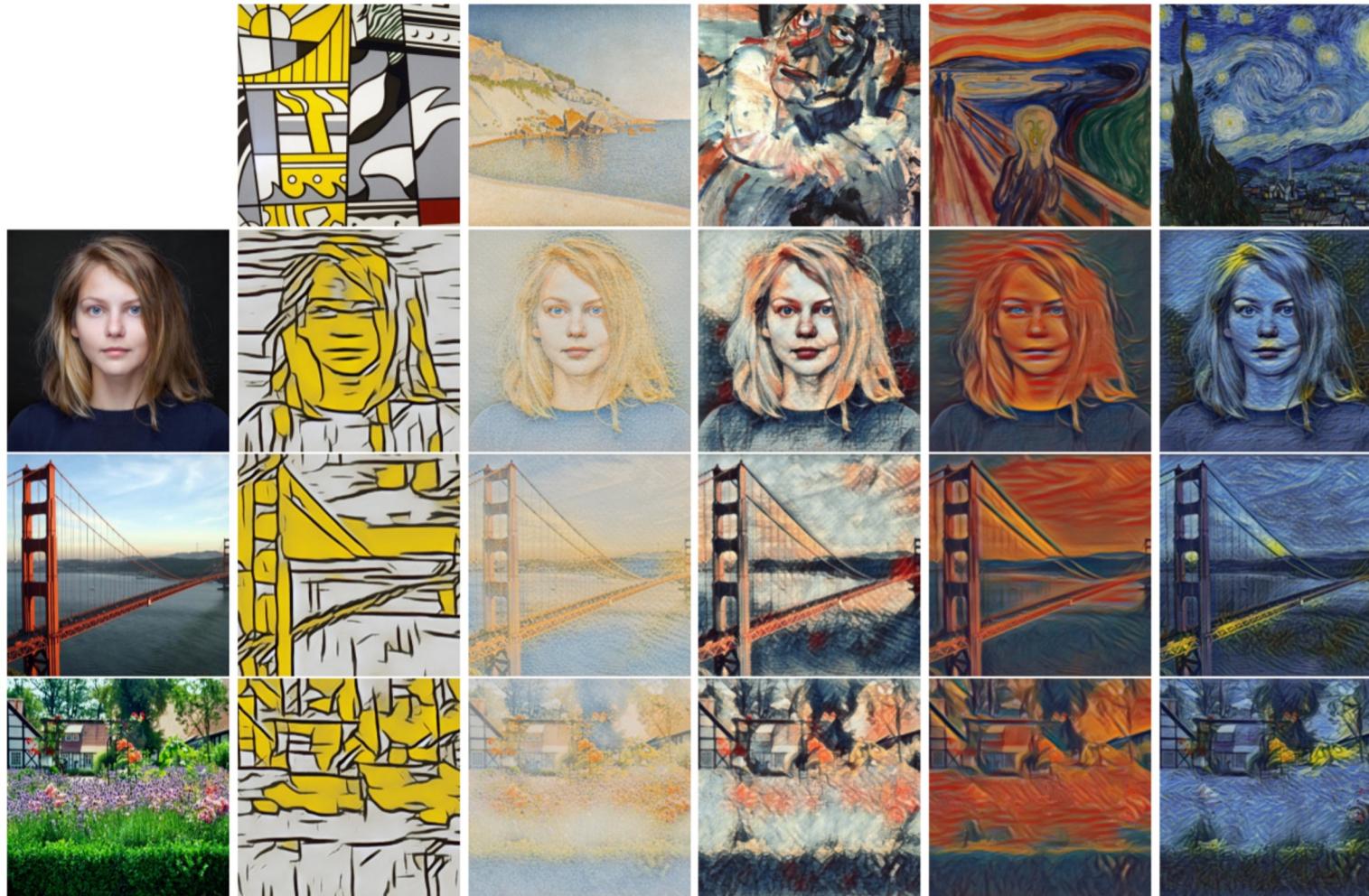
Fast Neural Style Transfer

Replacing batch
normalization with
Instance Normalization
improves results



Ulyanov et al, "Texture Networks: Feed-forward Synthesis of Textures and Stylized Images", ICML 2016
Ulyanov et al, "Instance Normalization: The Missing Ingredient for Fast Stylization", arXiv 2016

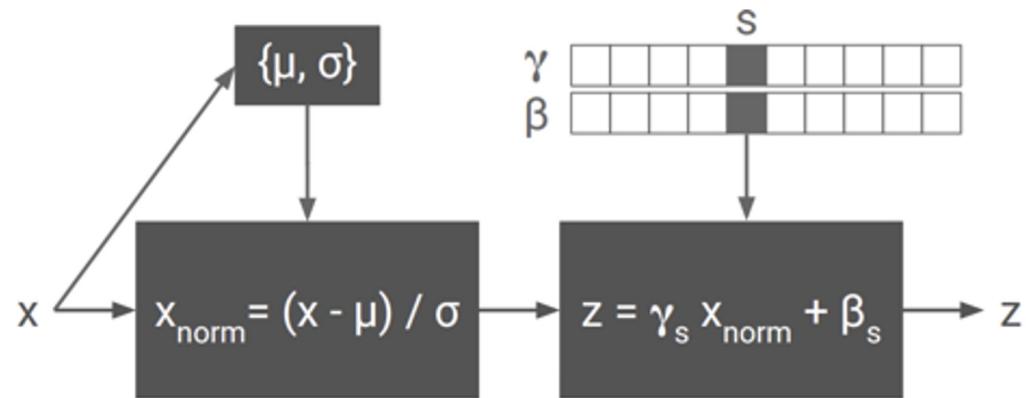
One Network, Many Styles



Dumoulin, Shlens, and Kudlur, "A Learned Representation for Artistic Style", ICLR 2017.

One Network, Many Styles

Use the same network for multiple styles using conditional instance normalization: learn separate scale and shift parameters per style



Single network can blend styles after training

Dumoulin, Shlens, and Kudlur, "A Learned Representation for Artistic Style", ICLR 2017.

Summary

Many methods for understanding CNN representations

Activations: Nearest neighbors, Dimensionality reduction, maximal patches, occlusion, CAM

Gradients: Grad-CAM, Saliency maps, class visualization, adversarial examples, feature inversion

Fun: DeepDream, Style Transfer.

Next Time: Self-Supervised Learning