Lecture 15:

Adversarial examples, texture synthesis, and style transfer

Agenda

Recap

Adversarial examples

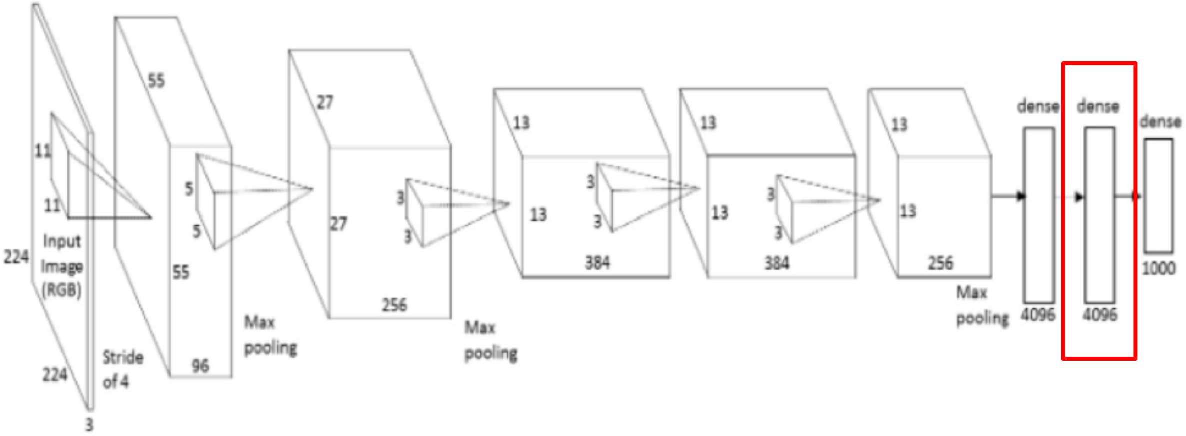
Texture synthesis and style transfer

Bonus

Last lecture: Understanding ConvNets

- Visualize the weights
- Visualize the last layer (via t-SNE)
- Visualize patches that maximally activate neurons
- Occlusion experiments
- Deconv approaches (single backward pass)
- Optimization over image approaches (optimization)

Question: Given a CNN code, is it possible to reconstruct the original image?



Find an image such that:

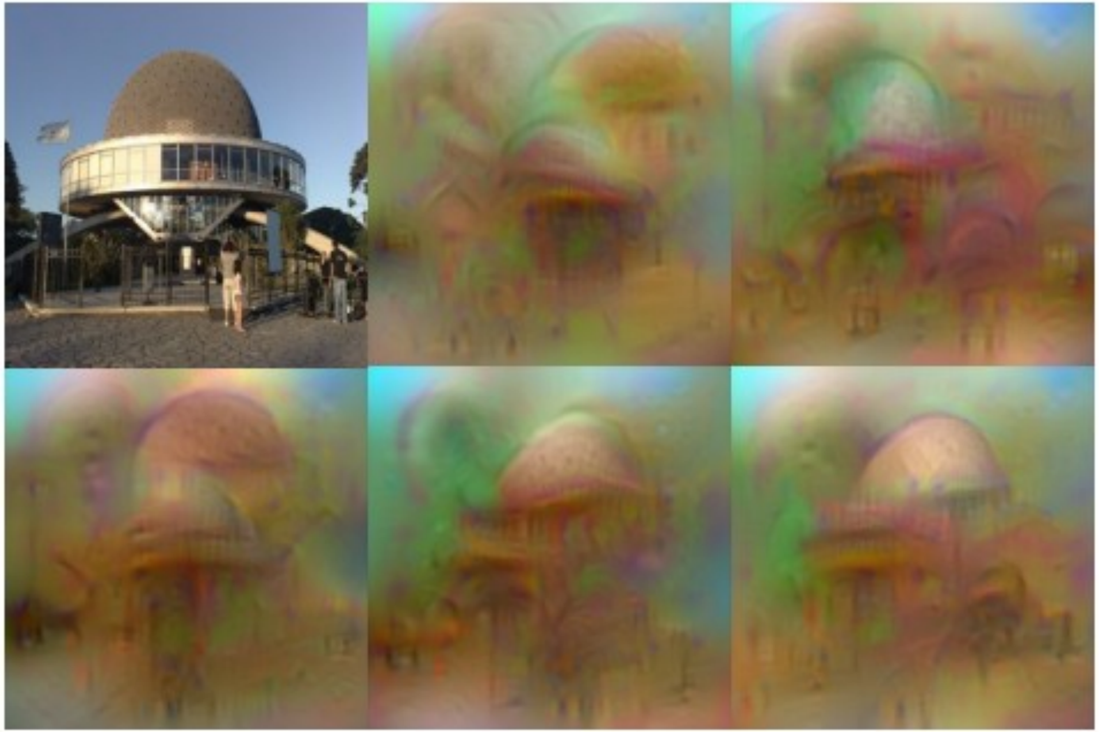
- Its code is similar to a given code
- It “looks natural” (image prior regularization)

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

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

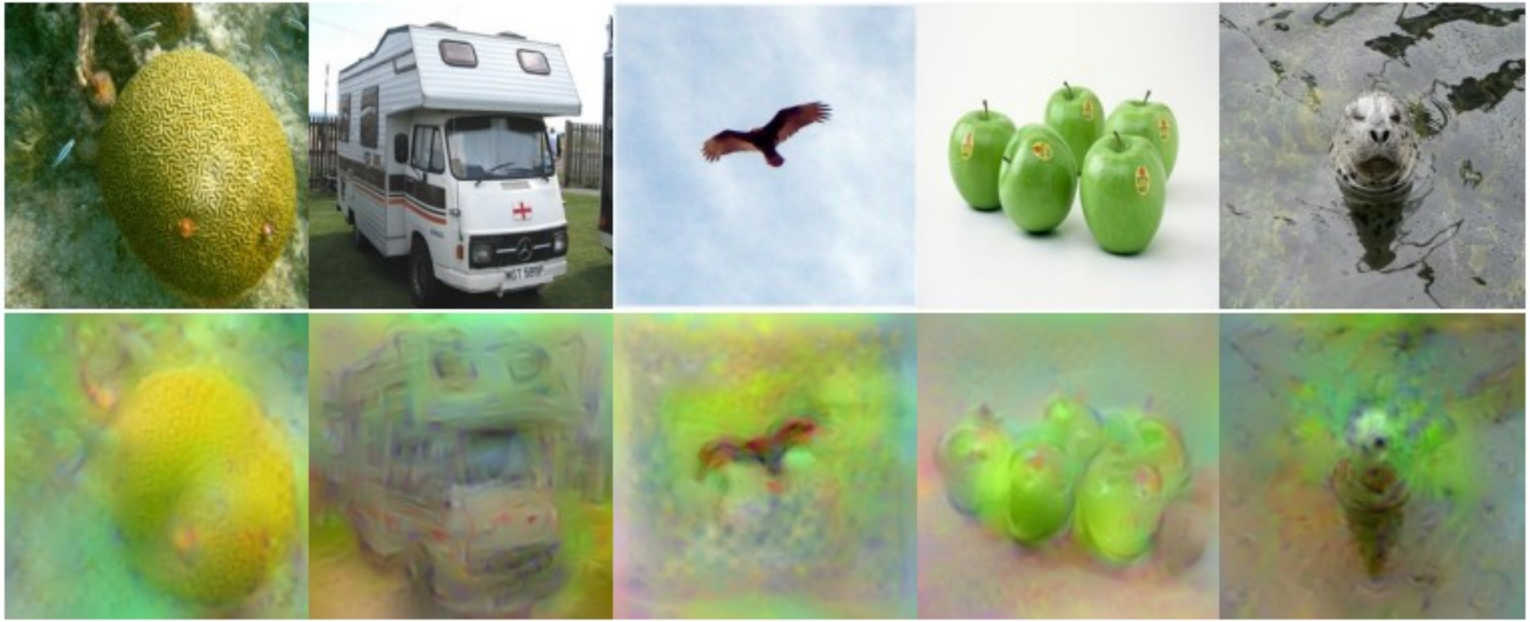
Understanding Deep Image Representations by Inverting Them [Mahendran and Vedaldi, 2014]

original image

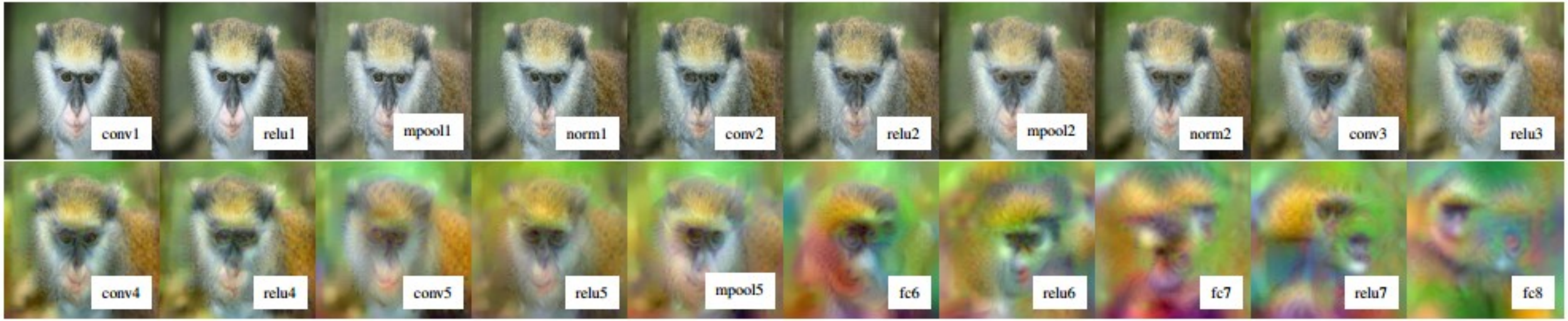


reconstructions from the 1000 log probabilities for ImageNet (ILSVRC) classes

Reconstructions from the representation after last last pooling layer (immediately before the first Fully Connected layer)



Reconstructions from intermediate layers



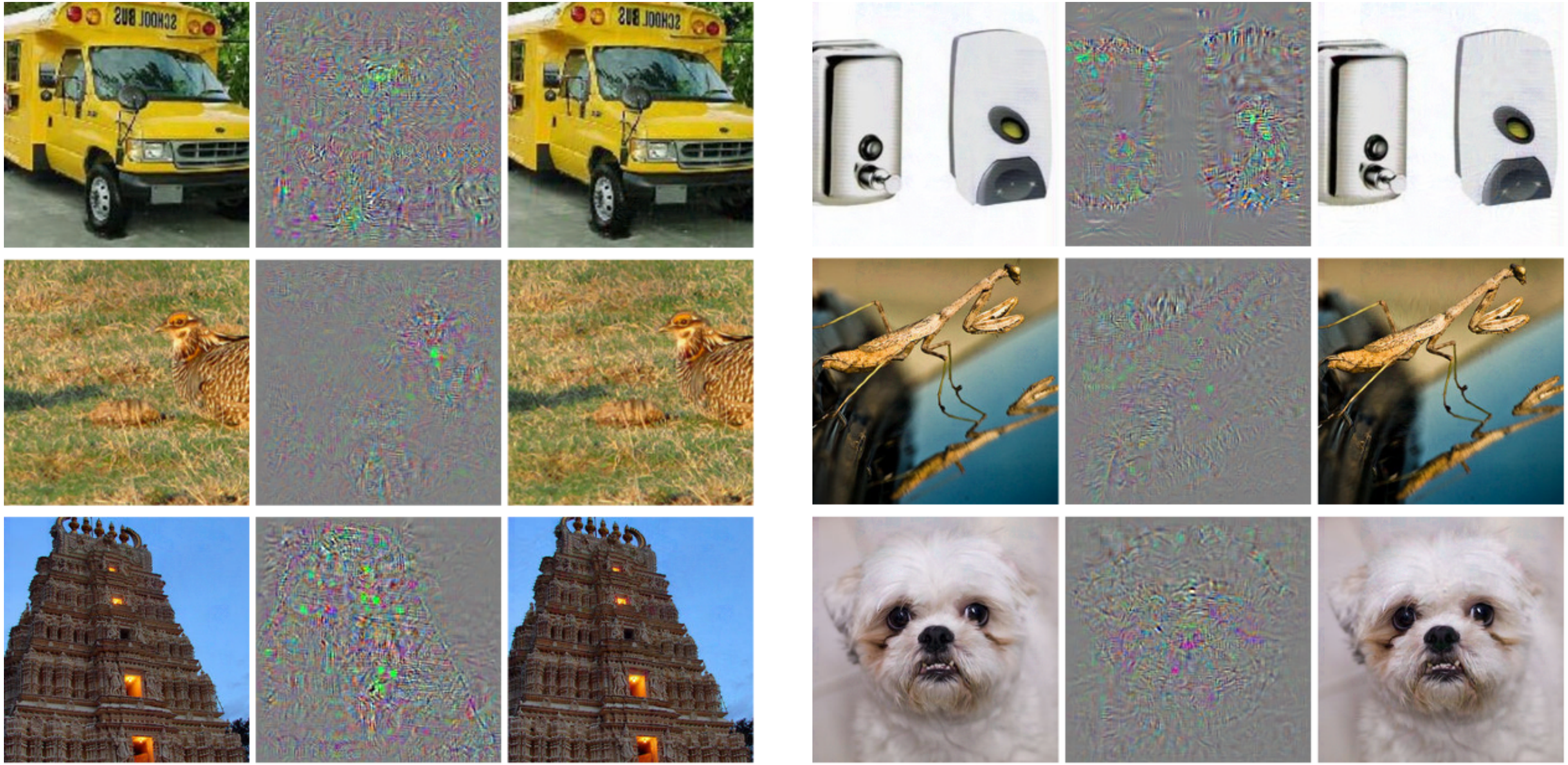
We can pose an optimization over the input image to maximize any class score.
That seems useful.

Question: Can we use this to “fool” ConvNets?

spoiler alert: yeah

(1) Start from an arbitrary image
(2) Pick an arbitrary class
(3) Modify the image to maximize the class
(4) Repeat until network is fooled

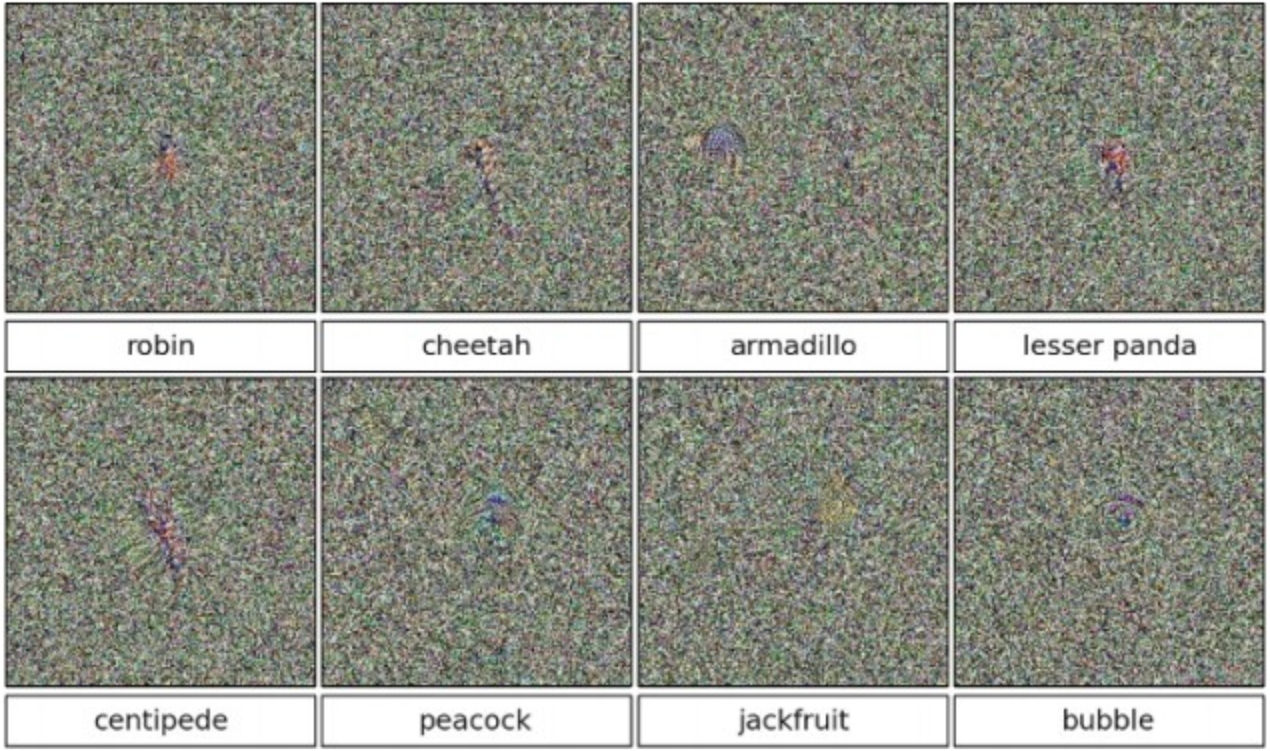
[Intriguing properties of neural networks, Szegedy et al., 2013]



correct +distort ostrich correct +distort ostrich

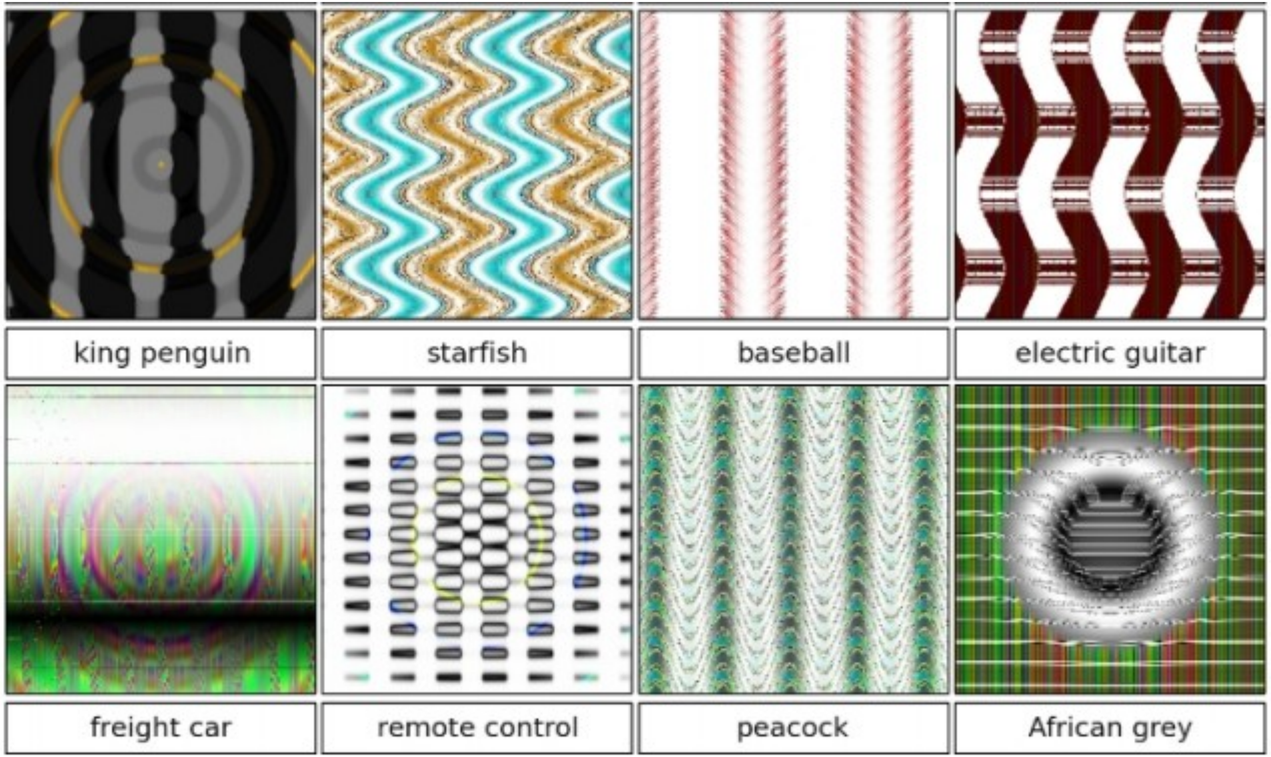
[Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images Nguyen, Yosinski, Clune, 2014]

>99.6% confidences



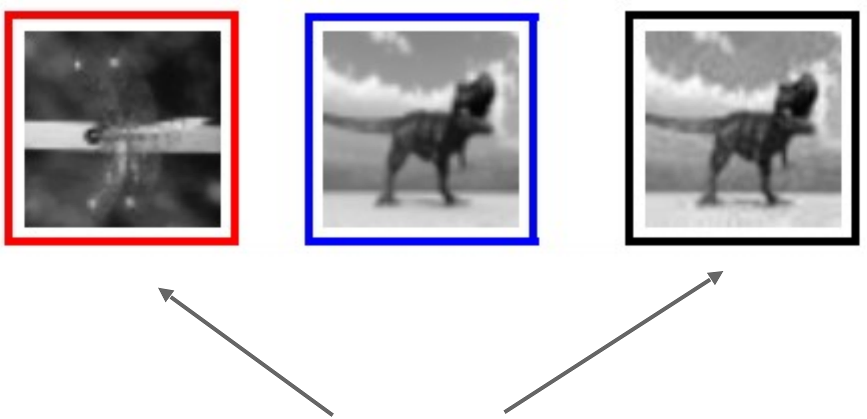
[Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images Nguyen, Yosinski, Clune, 2014]

>99.6% confidences

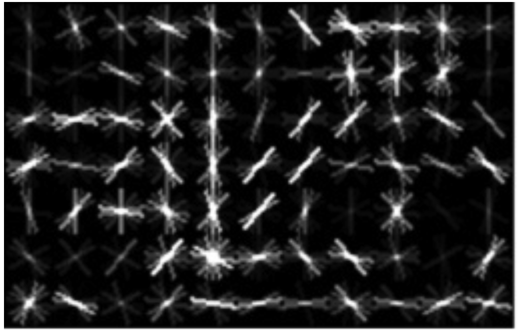


These kinds of results were around even before ConvNets…

[Exploring the Representation Capabilities of the HOG Descriptor, Tatu et al., 2011]

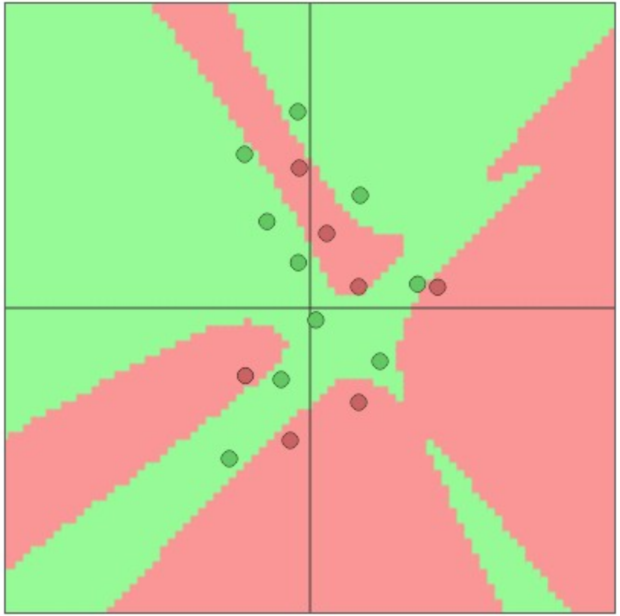


Identical HOG represention



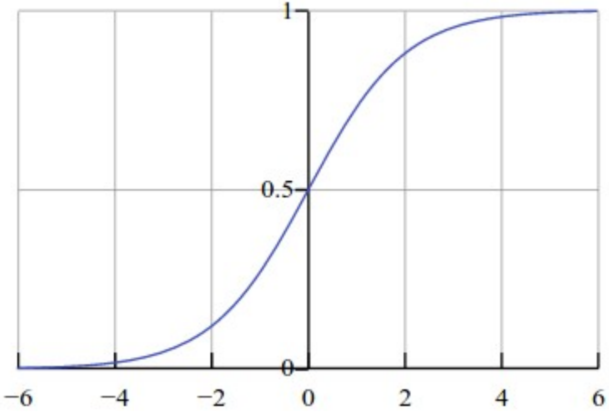
EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES
[Goodfellow, Shlens & Szegedy, 2014]

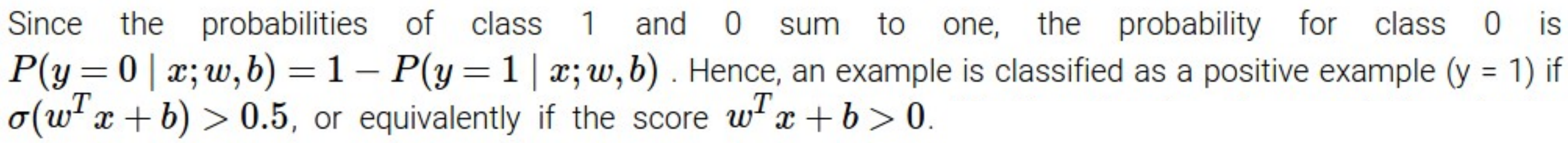
“primary cause of neural networks’ vulnerability to adversarial perturbation is their **linear nature**“



Lets fool a binary linear classifier: (logistic regression)

$$P(y=1 \mid x; w, b) = \frac{1}{1+e^{-(w^T x + b)}} = \sigma(w^T x + b)$$





Since the probabilities of class 1 and 0 sum to one, the probability for class 0 is $P(y=0 \mid x; w, b) = 1 - P(y=1 \mid x; w, b)$. Hence, an example is classified as a positive example (y = 1) if $\sigma(w^T x + b) > 0.5$, or equivalently if the score $w^T x + b > 0$.

Lets fool a binary linear classifier:

x	2	-1	3	-2	2	2	1	-4	5	1	← input example
w	-1	-1	1	-1	1	-1	1	1	-1	1	← weights

$$P(y=1 \mid x; w, b) = \frac{1}{1+e^{-(w^T x + b)}} = \sigma(w^T x + b)$$

Lets fool a binary linear classifier:

x	2	-1	3	-2	2	2	1	-4	5	1	← input example
w	-1	-1	1	-1	1	-1	1	1	-1	1	← weights

class 1 score = dot product:
= -2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3
=> probability of class 1 is 1/(1+e^(-(-3))) = 0.0474
i.e. the classifier is **95%** certain that this is class 0 example.

$$P(y=1 \mid x; w, b) = \frac{1}{1+e^{-(w^T x+b)}} = \sigma(w^T x + b)$$

Lets fool a binary linear classifier:

x	2	-1	3	-2	2	2	1	-4	5	1
w	-1	-1	1	-1	1	-1	1	1	-1	1
adversarial x	?	?	?	?	?	?	?	?	?	?

← input example

← weights

class 1 score = dot product:
= -2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3
=> probability of class 1 is 1/(1+e^(-(-3))) = 0.0474
i.e. the classifier is **95%** certain that this is class 0 example.

$$P(y=1 \mid x; w, b) = \frac{1}{1+e^{-(w^T x+b)}} = \sigma(w^T x + b)$$

Lets fool a binary linear classifier:

x	2	-1	3	-2	2	2	1	-4	5	1	← input example
w	-1	-1	1	-1	1	-1	1	1	-1	1	← weights
adversarial x	1.5	-1.5	3.5	-2.5	2.5	1.5	1.5	-3.5	4.5	1.5	

class 1 score before:
-2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3
=> probability of class 1 is 1/(1+e^(-(-3))) = 0.0474

-1.5+1.5+3.5+2.5+2.5-1.5+1.5-3.5-4.5+1.5 = 2
=> probability of class 1 is now 1/(1+e^(-(2))) = 0.88
i.e. we improved the class 1 probability from 5% to 88%

$$P(y=1\mid x;w,b)=\frac{1}{1+e^{-(w^Tx+b)}}=\sigma(w^Tx+b)$$

Lets fool a binary linear classifier:

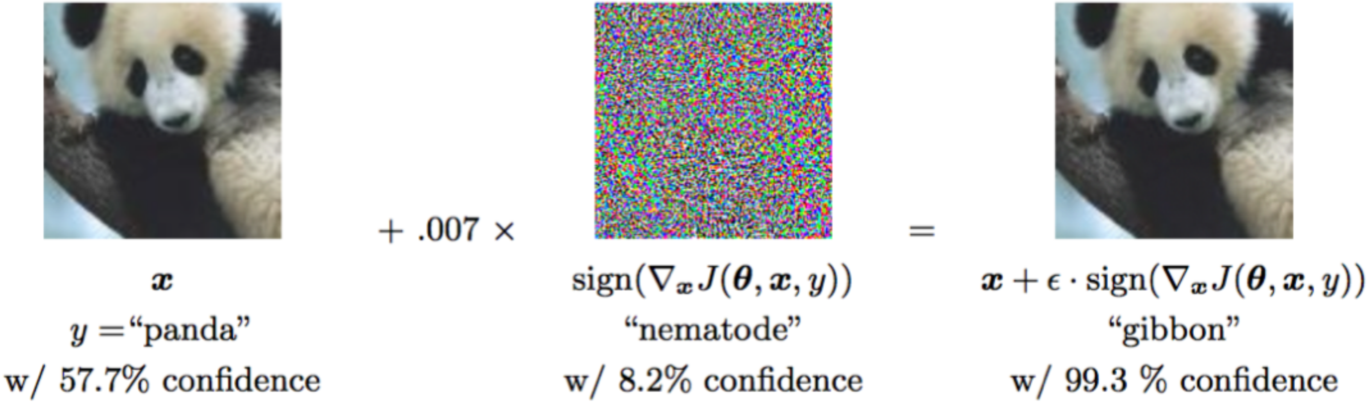
x	2	-1	3	-2	2	2	1	-4	5	1	← input example
w	-1	-1	1	-1	1	-1	1	1	-1	1	← weights
adversarial x	1.5	-1.5	3.5	-2.5	2.5	1.5	1.5	-3.5	4.5	1.5	

class 1 score before:
-2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3
=> probability of class 1 is 1/(1+e^(-(-3))) = 0.0474

-1.5+1.5+3.5+2.5+2.5-1.5+1.5-3.5-4.5+1.5 = 2
=> probability of class 1 is now 1/(1+e^(-(2))) = 0.88
i.e. we improved the class 1 probability from 5% to 88%

This was only with 10 input dimensions. A 224x224 input image has 150,528.

(It’s significantly easier with more numbers, need smaller nudge for each)



$$x + .007 \times \text{sign}(\nabla_{x} J(\theta, x, y)) = x + \epsilon \cdot \text{sign}(\nabla_{x} J(\theta, x, y))$$

$y =$"panda"
w/ 57.7% confidence

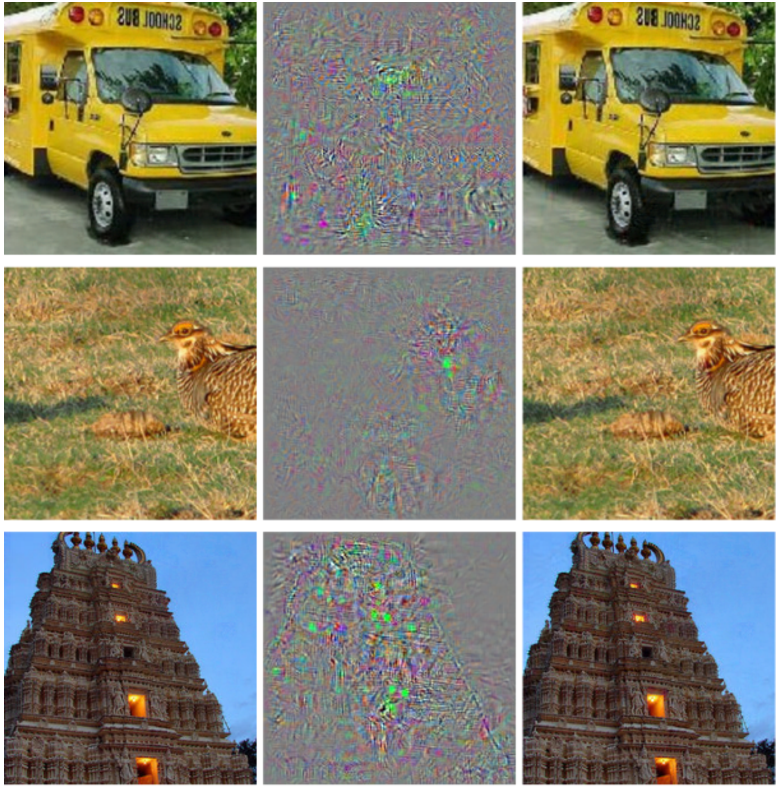
"nematode"
w/ 8.2% confidence

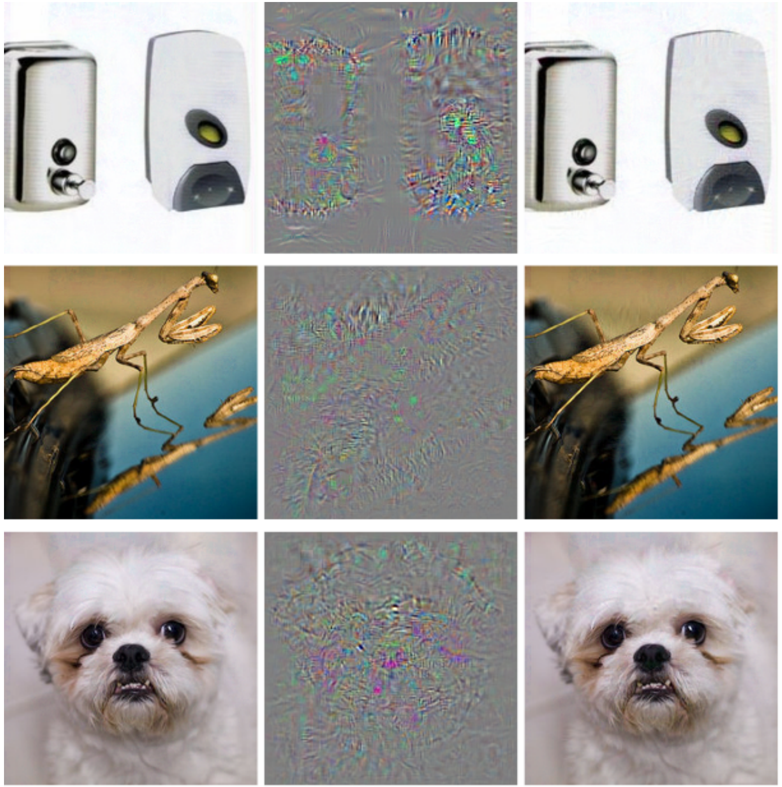
"gibbon"
w/ 99.3 % confidence

Explaining and Harnessing Adversarial Examples

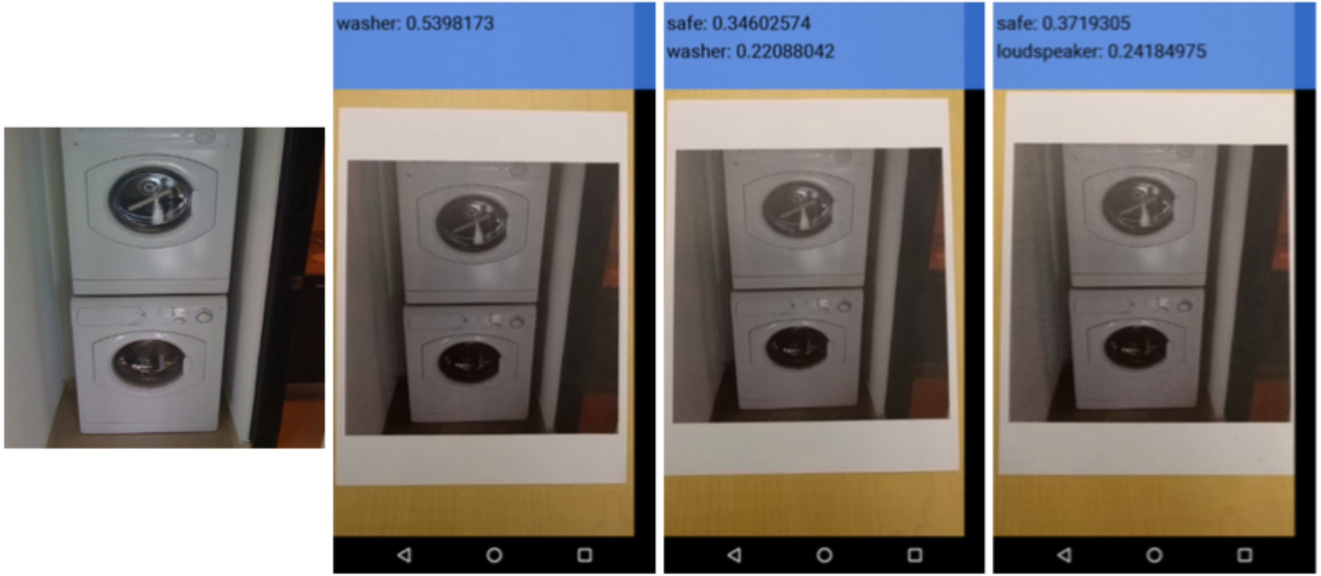
Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy

[Intriguing properties of neural networks, Szegedy et al., 2013]





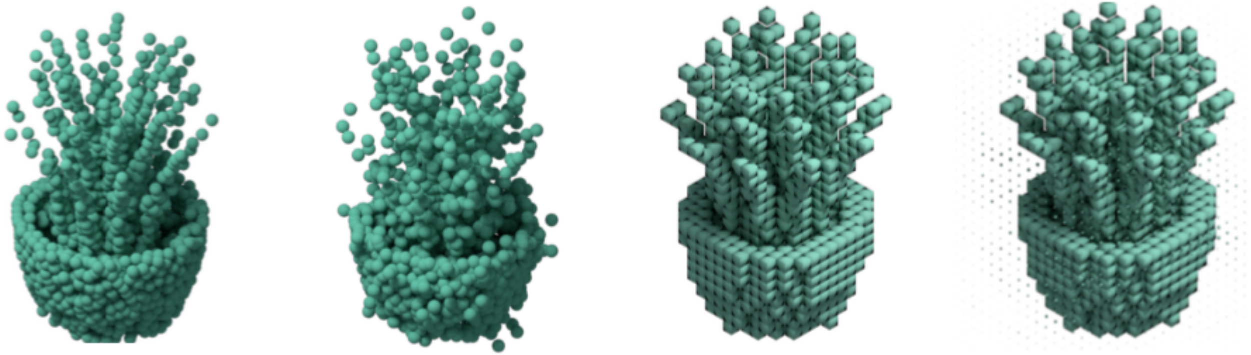
Can be printed on paper! Kurakin et al., 17



(a) Image from dataset (b) Clean image (c) Adv. image, $\epsilon = 4$ (d) Adv. image, $\epsilon = 8$

Also works for 3D models!
(though a little harder for point clouds)

Su et al., ECCV 2018

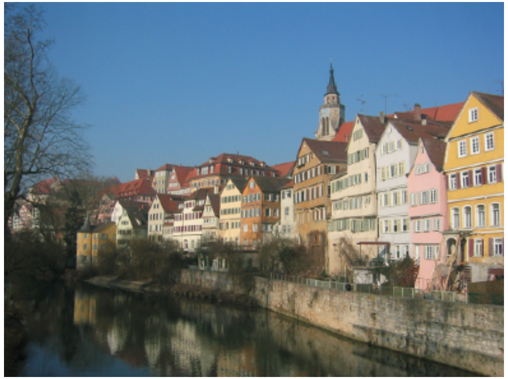


“plant” “bench” “plant” “bench”

point cloud voxel

Neural Style Transfer and Texture Synthesis

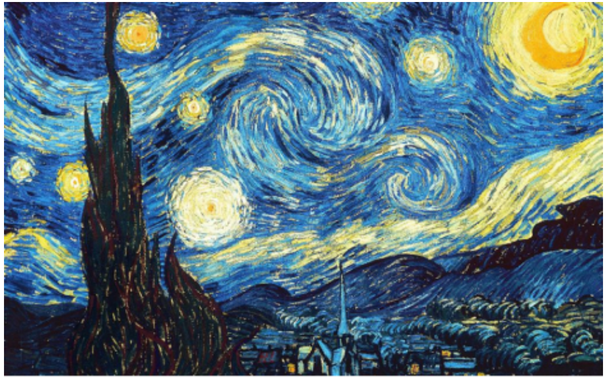
Content Image



This image is licensed under CC-BY 3.0

+

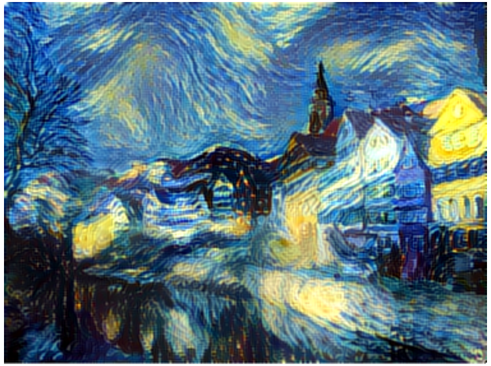
Style Image



Starry Night by Van Gogh is in the public domain

=

Style Transfer!

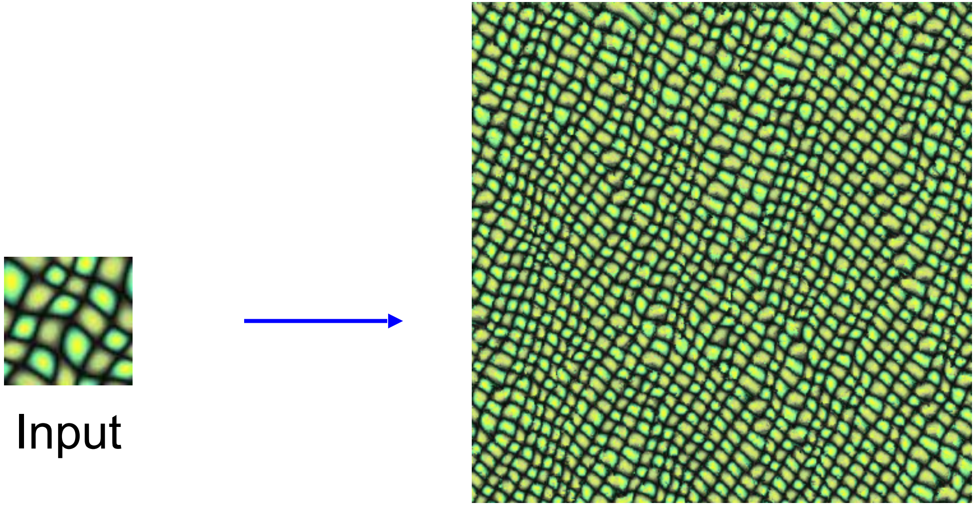


This image copyright Justin Johnson, 2015. Reproduced with permission.

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

Texture Synthesis

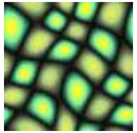
Given a sample patch of some texture, can we generate a bigger image of the same texture?

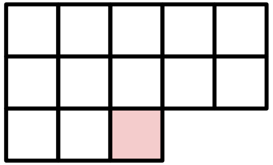


Input

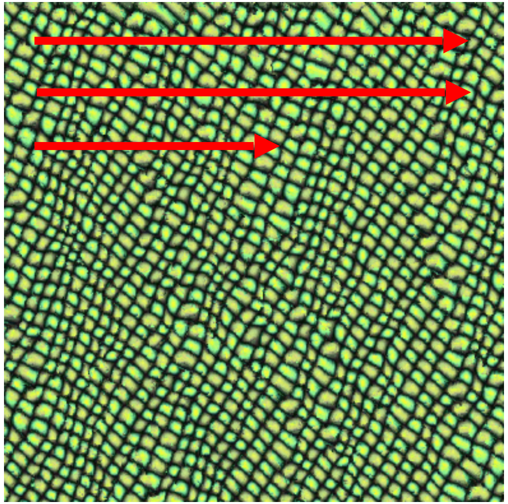
Output

Texture Synthesis: Nearest Neighbor



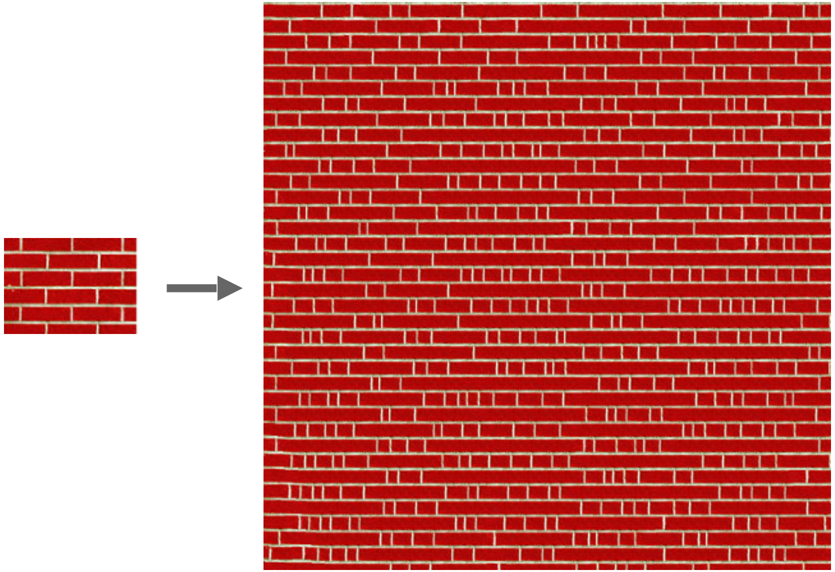


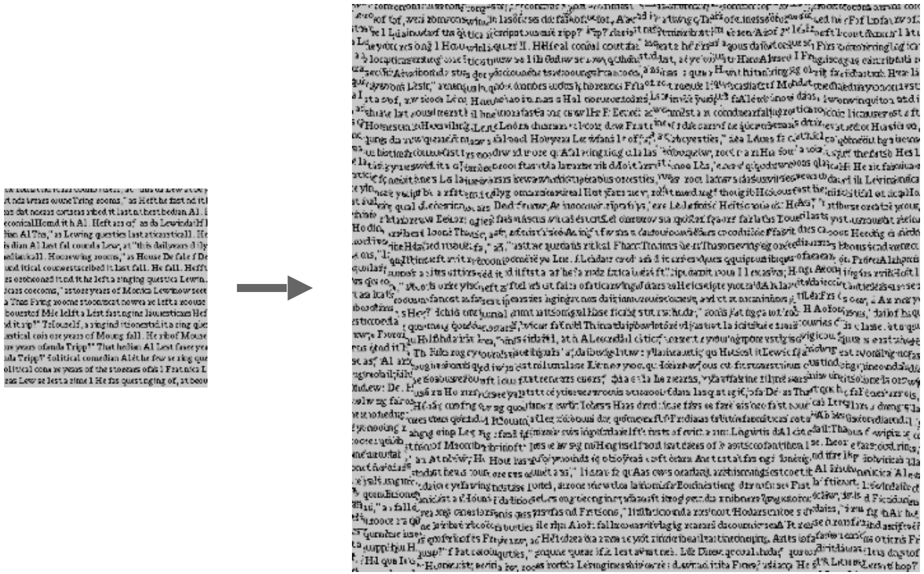
Generate pixels one at a time in scanline order; form neighborhood of already generated pixels and copy nearest neighbor from input



Wei and Levoy, “Fast Texture Synthesis using Tree-structured Vector Quantization”, SIGGRAPH 2000
Efros and Leung, “Texture Synthesis by Non-parametric Sampling”, ICCV 1999

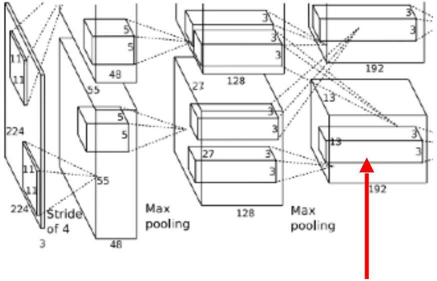
Texture Synthesis: Nearest Neighbor

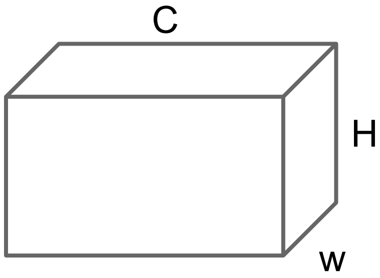




Neural Texture Synthesis: Gram Matrix

This image is in the public domain.

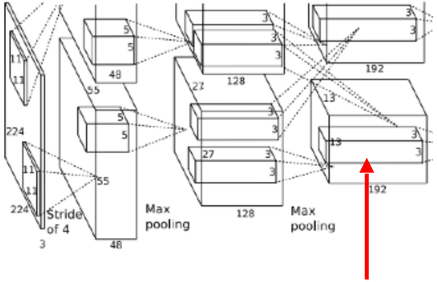


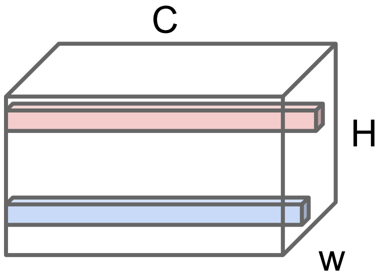


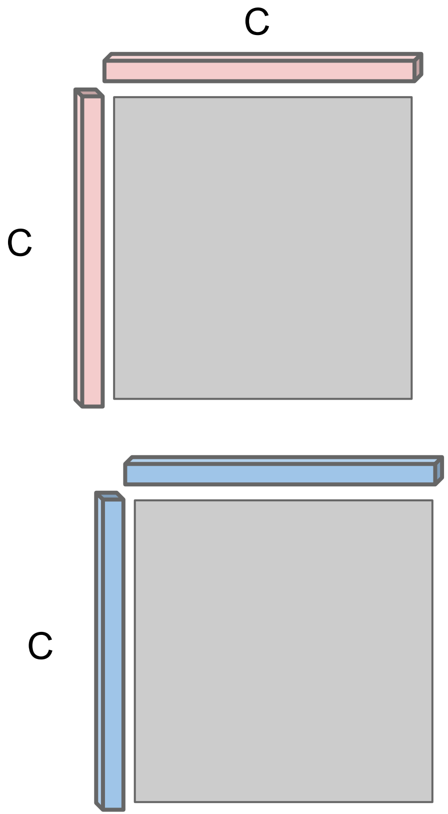
Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

Neural Texture Synthesis: Gram Matrix

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





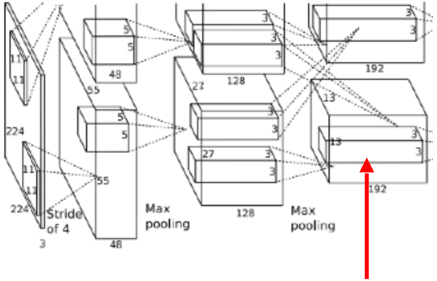


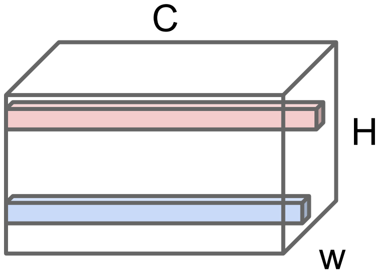
Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

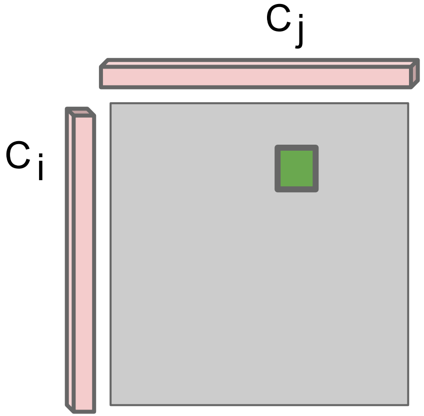
Outer product of C-dimensional vector with itself gives C x C matrix measuring co-occurrence

Neural Texture Synthesis: Gram Matrix

This image is in the public domain.







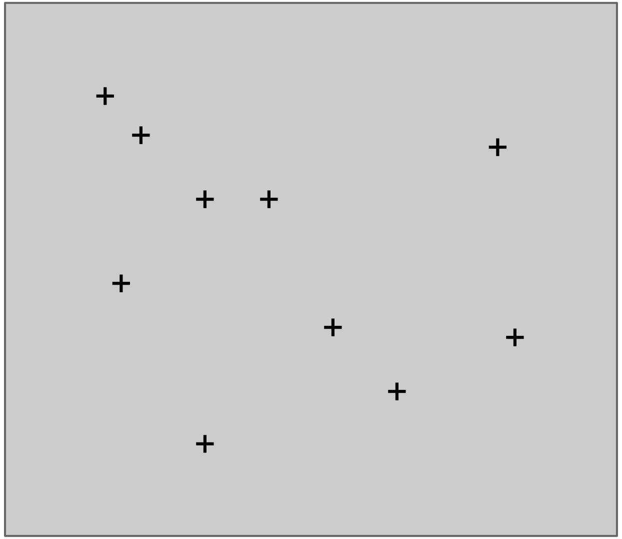
Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

Outer product of C-dimensional vector with itself gives C x C matrix measuring co-occurrence

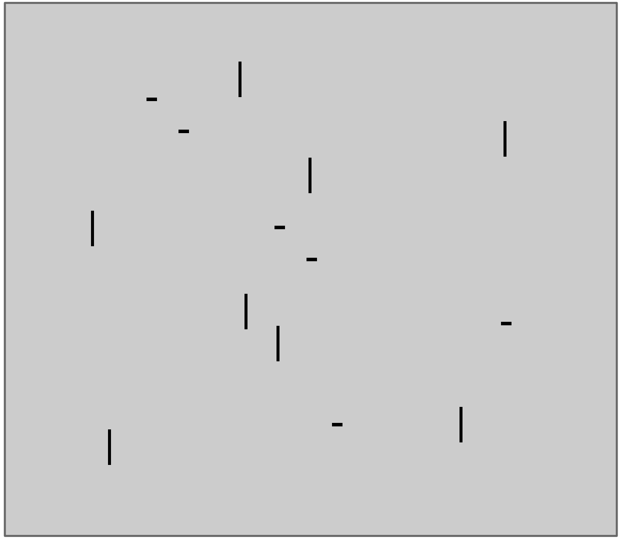
🟩 The green box G(i,j) represents the AVERAGE, over image positions in the image, of the product of features i and j.

Suppose “i” represents horizontal lines and “j” represents vertical lines.

Vertical and horizontal co-occur

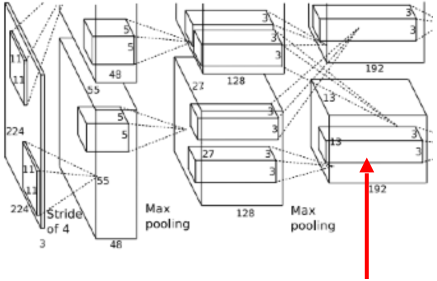


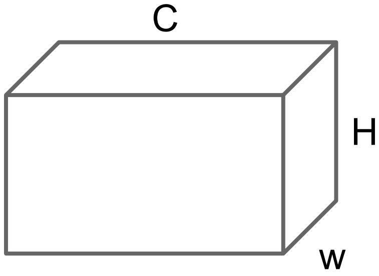
Vertical and horizontal DO NOT co-occur

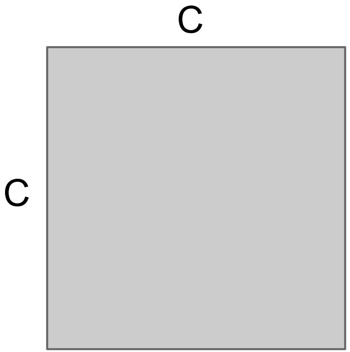


Neural Texture Synthesis: Gram Matrix

This image is in the public domain.







Gram Matrix

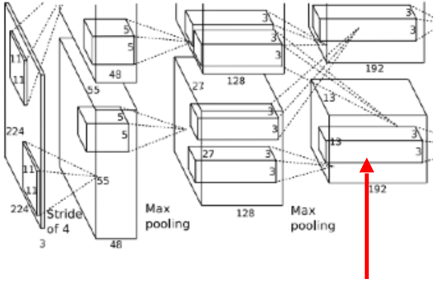
Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

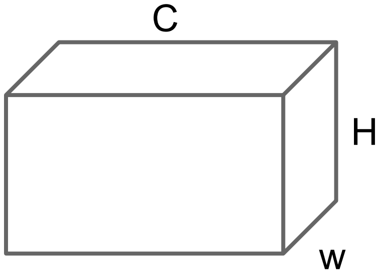
Outer product of C-dimensional vector with itself gives C x C matrix measuring co-occurrence

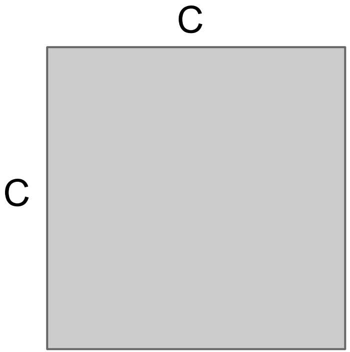
Average over all HW outer products, giving **Gram matrix** of shape C x C

Neural Texture Synthesis: Gram Matrix

This image is in the public domain.







Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

Outer product of C-dimensional vector with itself gives C x C matrix measuring co-occurrence

Average over all HW outer products, giving **Gram matrix** of shape C x C

Efficient to compute; reshape features from

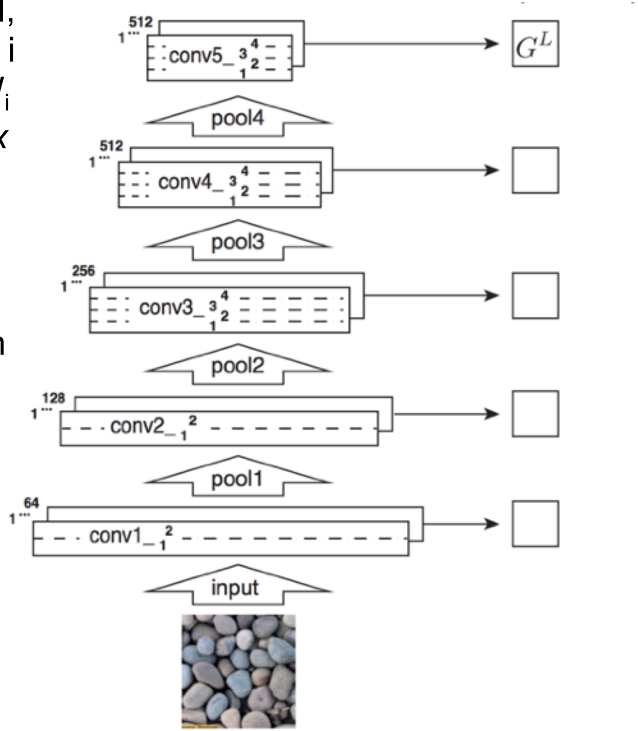
C x H x W to =C x HW

then compute $G = FF^T$

Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the *Gram matrix* giving outer product of features:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk} \quad \text{(shape } C_i \times C_i\text{)}$$



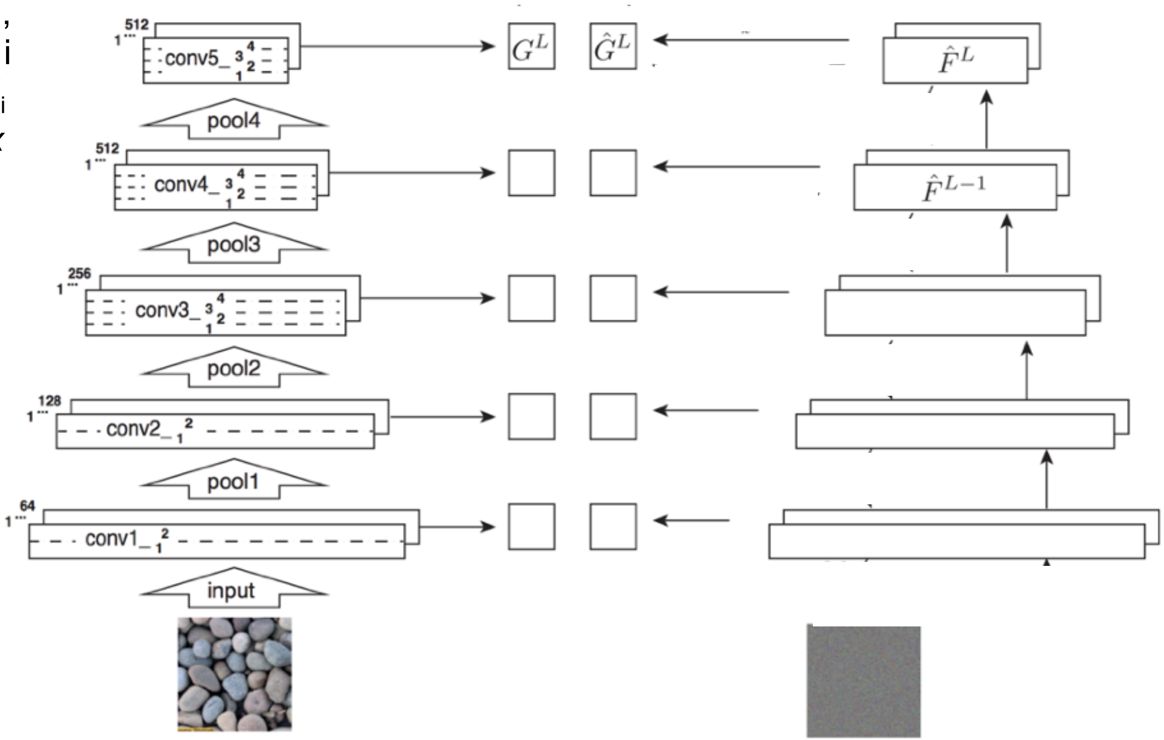
Gatys, Ecker, and Bethge, "Texture Synthesis Using Convolutional Neural Networks", NIPS 2015
Figure copyright Leon Gatys, Alexander S. Ecker, and Matthias Bethge, 2015. Reproduced with permission.

Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the *Gram matrix* giving outer product of features:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk} \quad \text{(shape } C_i \times C_i\text{)}$$

4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer



Gatys, Ecker, and Bethge, “Texture Synthesis Using Convolutional Neural Networks”, NIPS 2015
Figure copyright Leon Gatys, Alexander S. Ecker, and Matthias Bethge, 2015. Reproduced with permission.

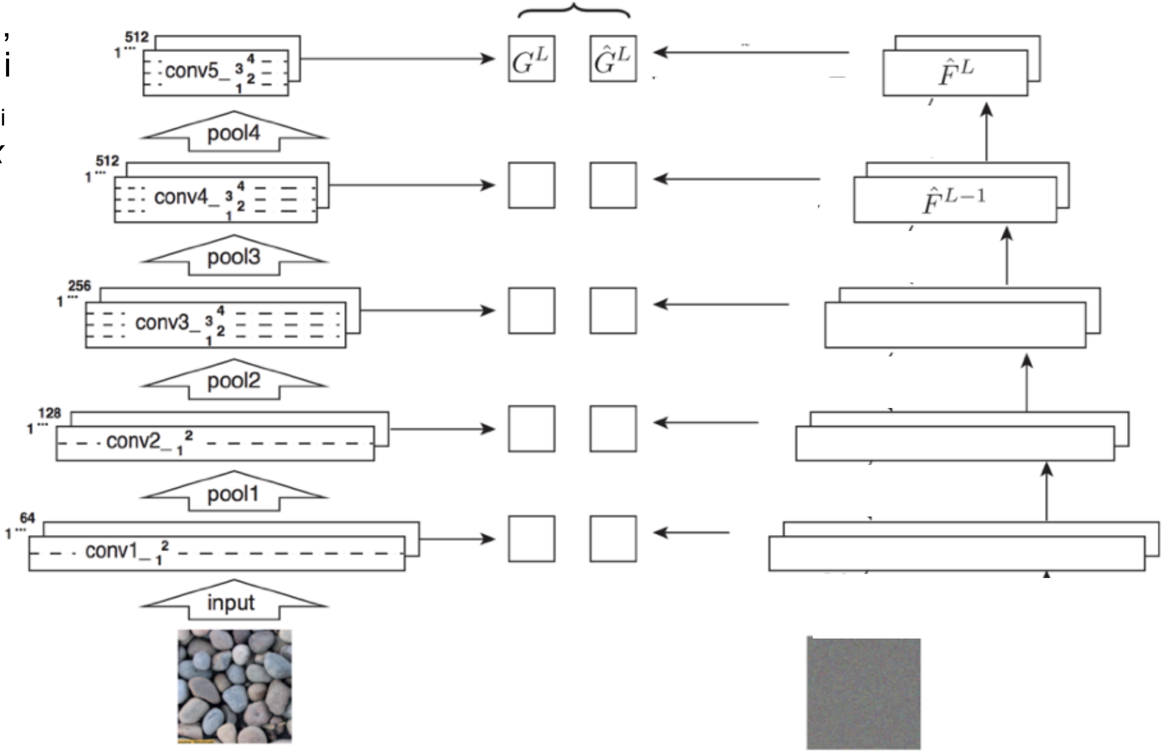
Neural Texture Synthesis

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - \hat{G}_{ij}^l \right)^2 \qquad \mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^{L} w_l E_l$$

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the *Gram matrix* giving outer product of features:

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l \text{ (shape } C_i \times C_i\text{)}$$

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



Gatys, Ecker, and Bethge, "Texture Synthesis Using Convolutional Neural Networks", NIPS 2015
Figure copyright Leon Gatys, Alexander S. Ecker, and Matthias Bethge, 2015. Reproduced with permission.

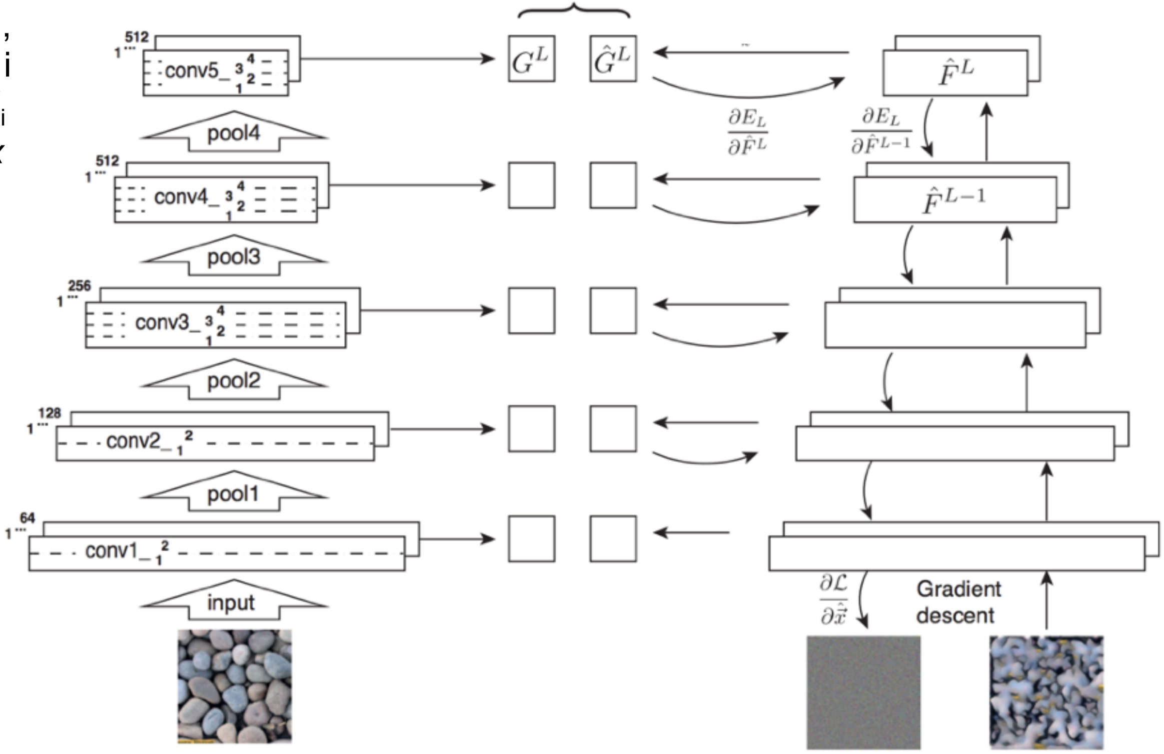
Neural Texture Synthesis

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - \hat{G}_{ij}^l \right)^2 \qquad \mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^{L} w_l E_l$$

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the *Gram matrix* giving outer product of features:

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l \text{ (shape } C_i \times C_i\text{)}$$

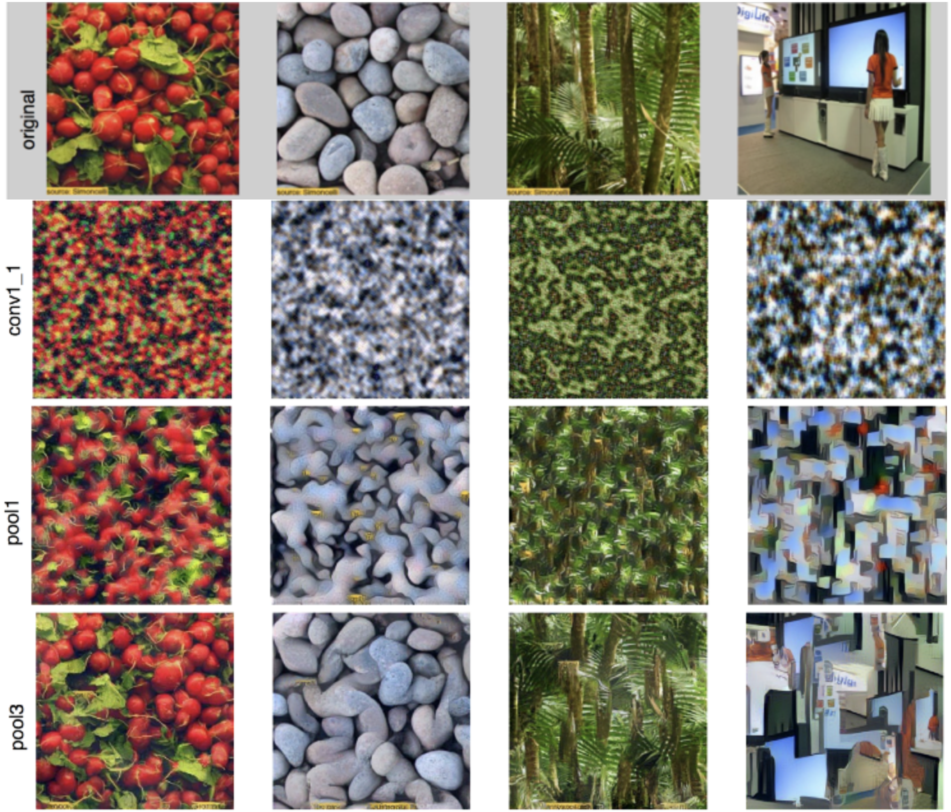
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5



Gatys, Ecker, and Bethge, “Texture Synthesis Using Convolutional Neural Networks”, NIPS 2015
Figure copyright Leon Gatys, Alexander S. Ecker, and Matthias Bethge, 2015. Reproduced with permission.

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
Figure copyright Leon Gatys, Alexander S. Ecker, and Matthias Bethge, 2015. Reproduced with permission.

Neural Texture Synthesis: Texture = Artwork

Texture synthesis (Gram reconstruction)

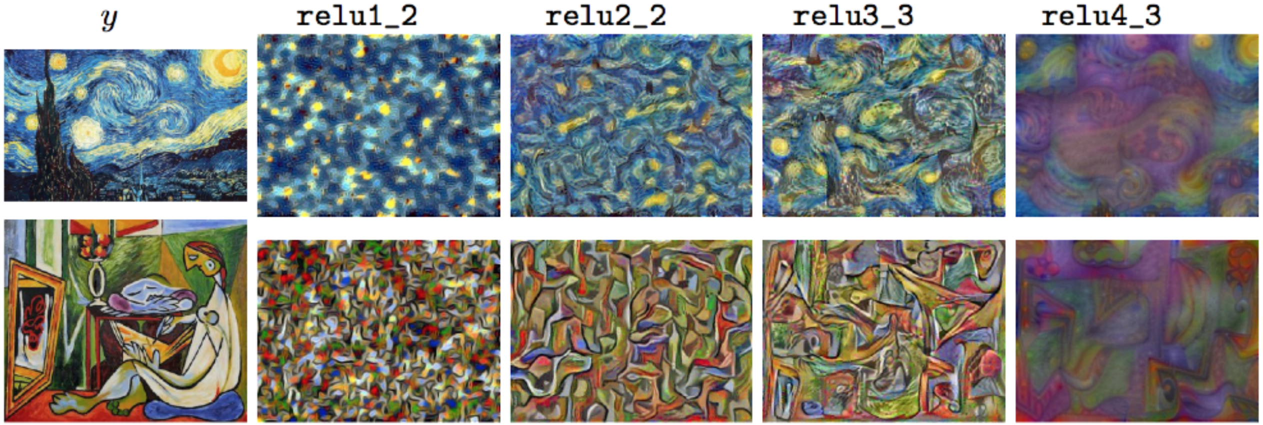
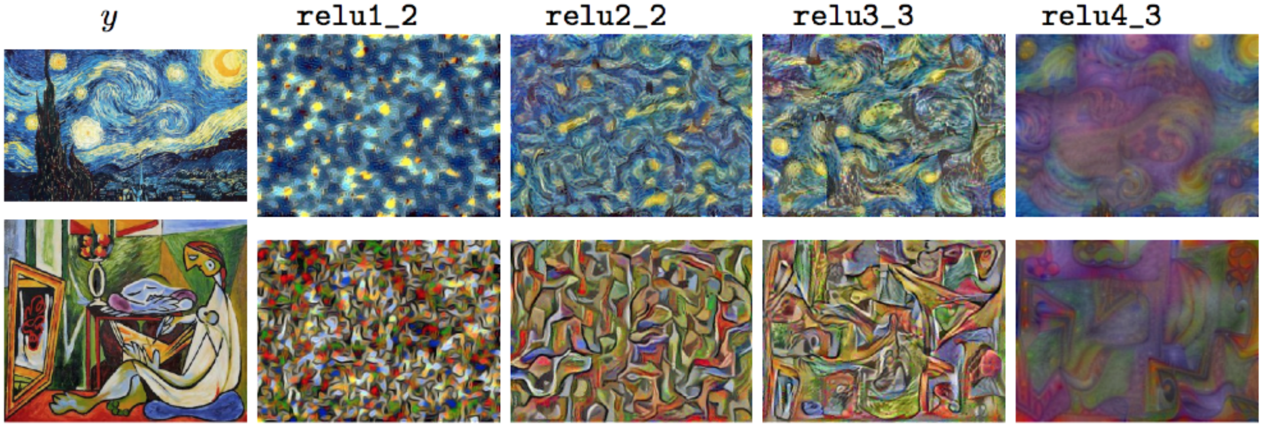


Figure from Johnson, Alahi, and Fei-Fei, “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016. Copyright Springer, 2016. Reproduced for educational purposes.

Neural Style Transfer: Feature + Gram Reconstruction

Texture synthesis (Gram reconstruction)



Feature reconstruction

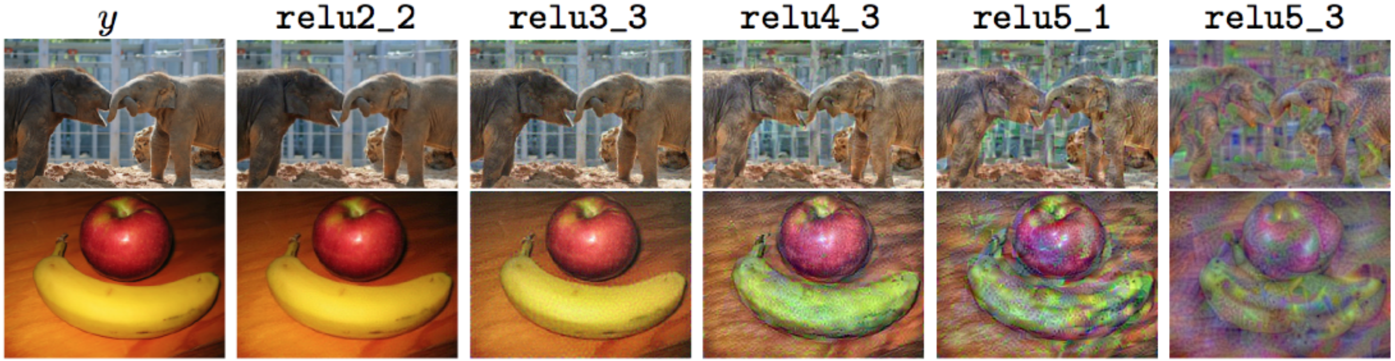


Figure from Johnson, Alahi, and Fei-Fei, “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016. Copyright Springer, 2016. Reproduced for educational purposes.

Neural Style Transfer

Content Image

This image is licensed under CC-BY 3.0

+

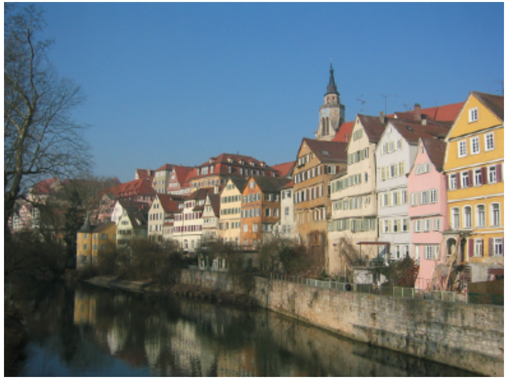
Style Image

Starry Night by Van Gogh is in the public domain

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

Neural Style Transfer

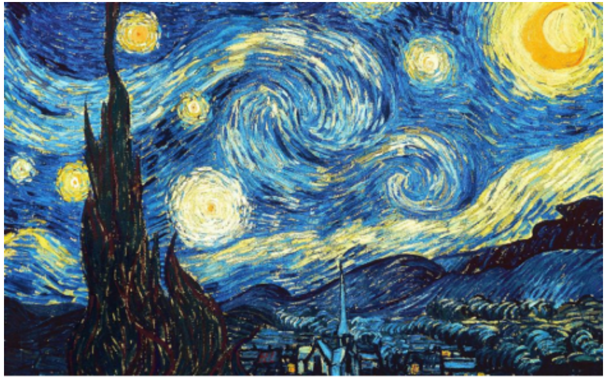
Content Image



This image is licensed under CC-BY 3.0

+

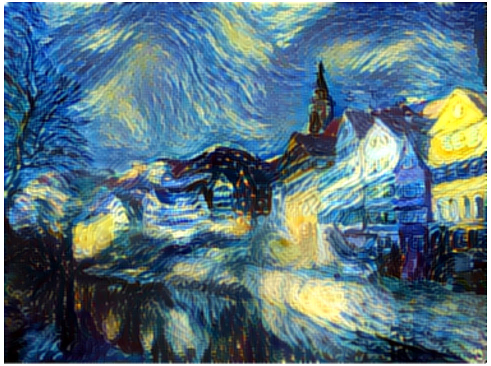
Style Image



Starry Night by Van Gogh is in the public domain

=

Style Transfer!



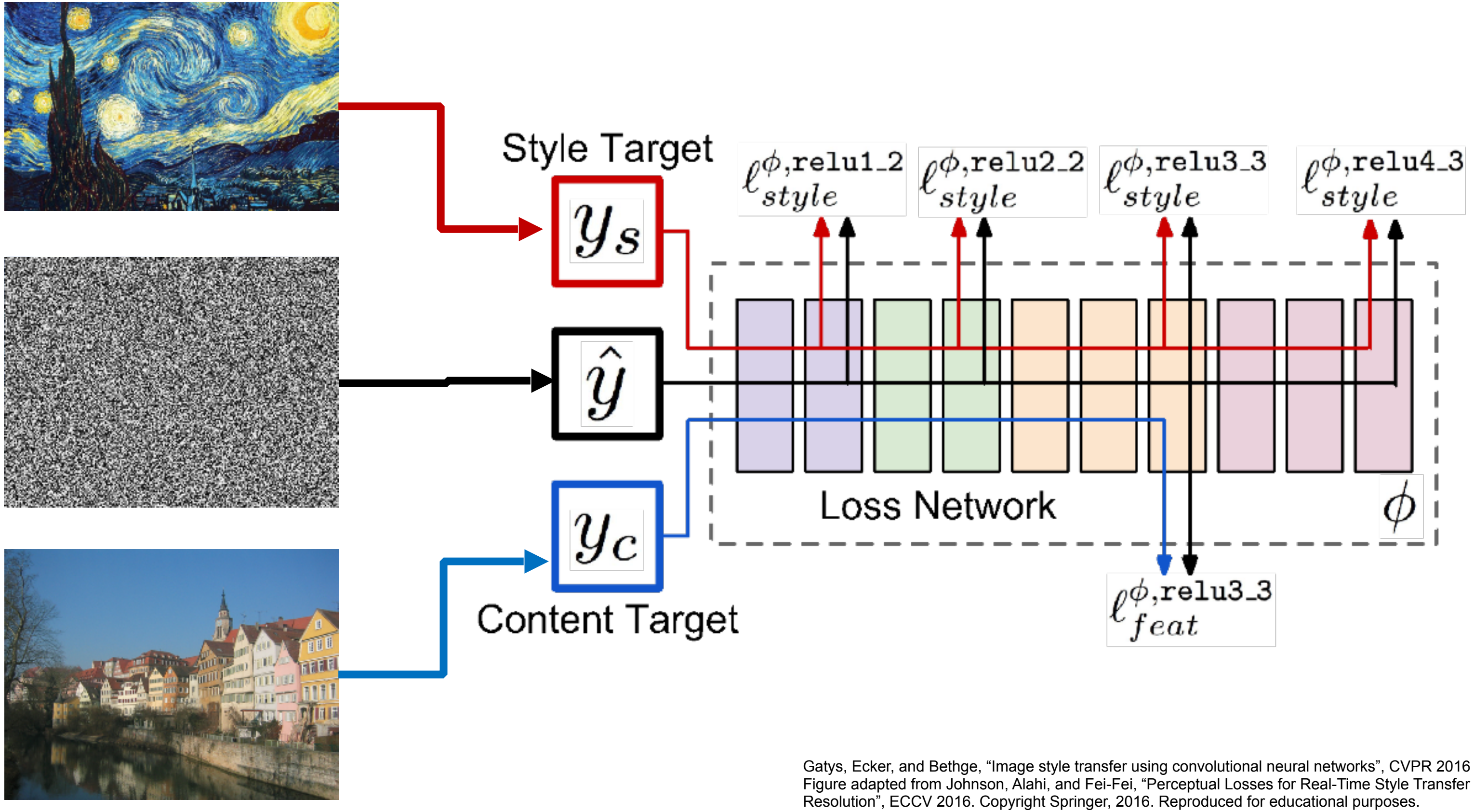
This image copyright Justin Johnson, 2015. Reproduced with permission.

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

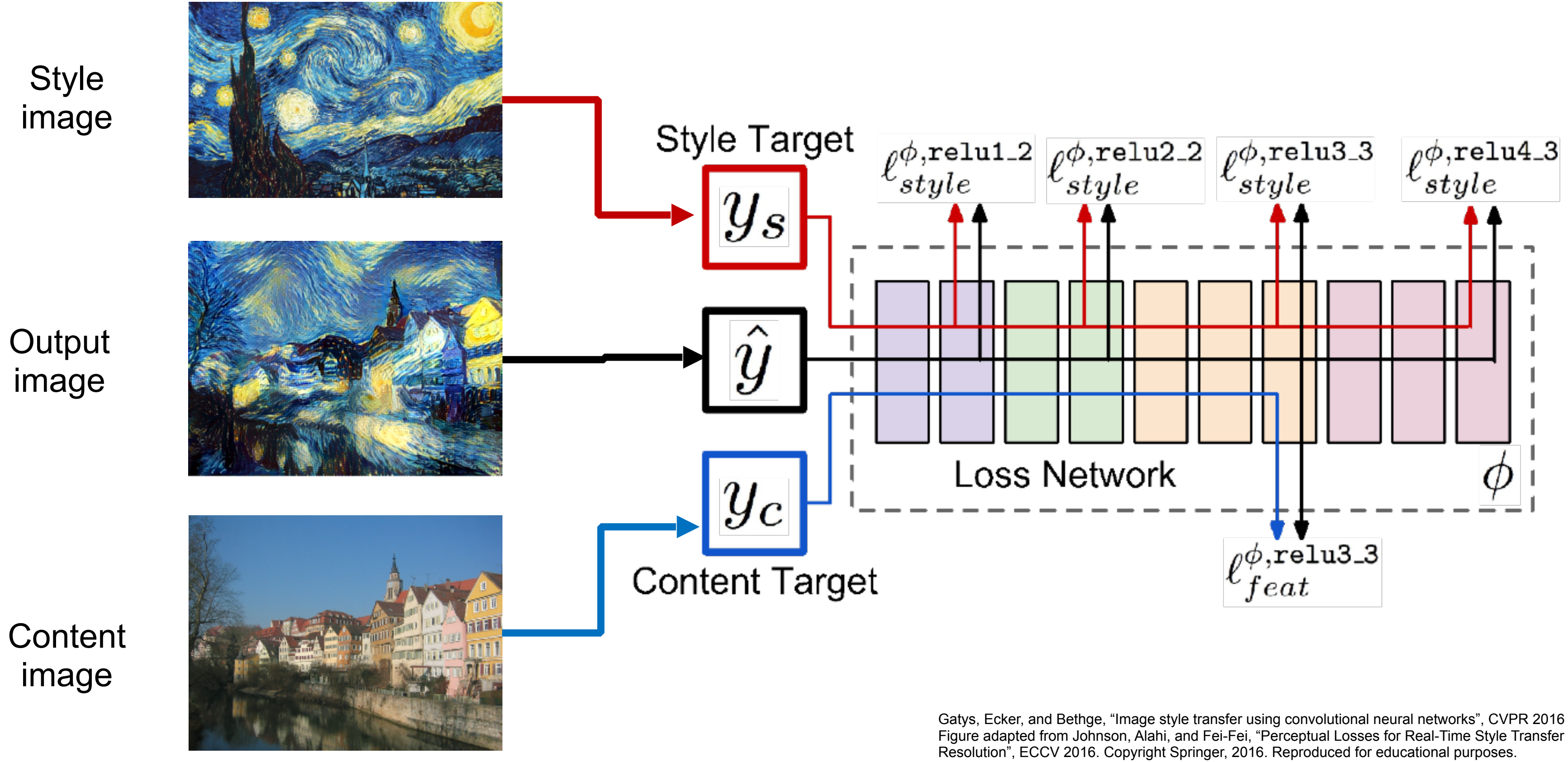
Style image

Output image (Start with noise)

Content image


Style Target
y_s
$\ell_{style}^{\phi,relu1\_2}$
$\ell_{style}^{\phi,relu2\_2}$
$\ell_{style}^{\phi,relu3\_3}$
$\ell_{style}^{\phi,relu4\_3}$
$\hat{y}$
y_c
Content Target
Loss Network
ϕ
$\ell_{feat}^{\phi,relu3\_3}$

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. Copyright Springer, 2016. Reproduced for educational purposes.



Style image

Output image

Content image

Style Target

y_s

$\hat{y}$

y_c

Content Target

$\ell_{style}^{\phi,relu1\_2}$

$\ell_{style}^{\phi,relu2\_2}$

$\ell_{style}^{\phi,relu3\_3}$

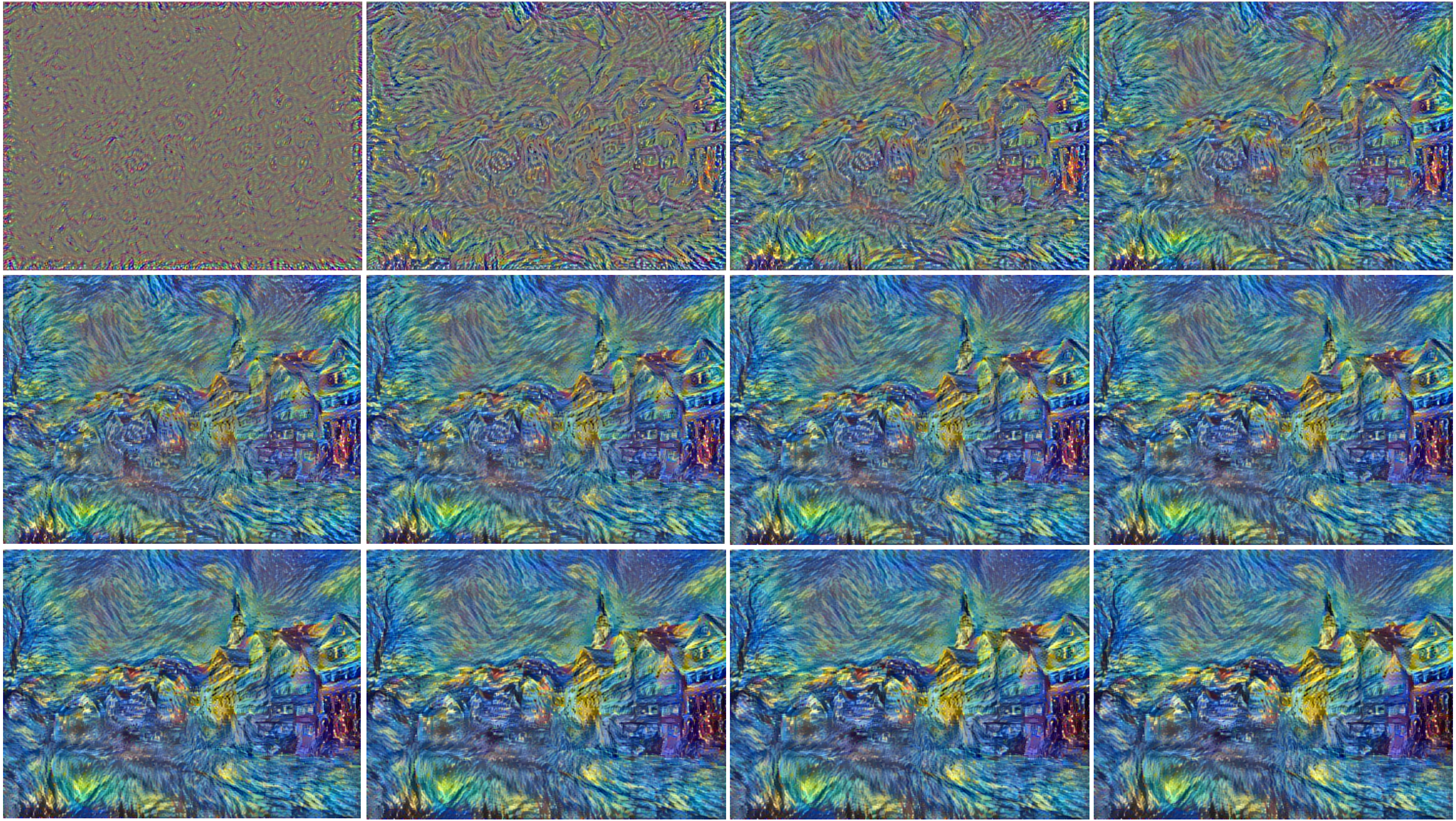
$\ell_{style}^{\phi,relu4\_3}$

Loss Network

ϕ

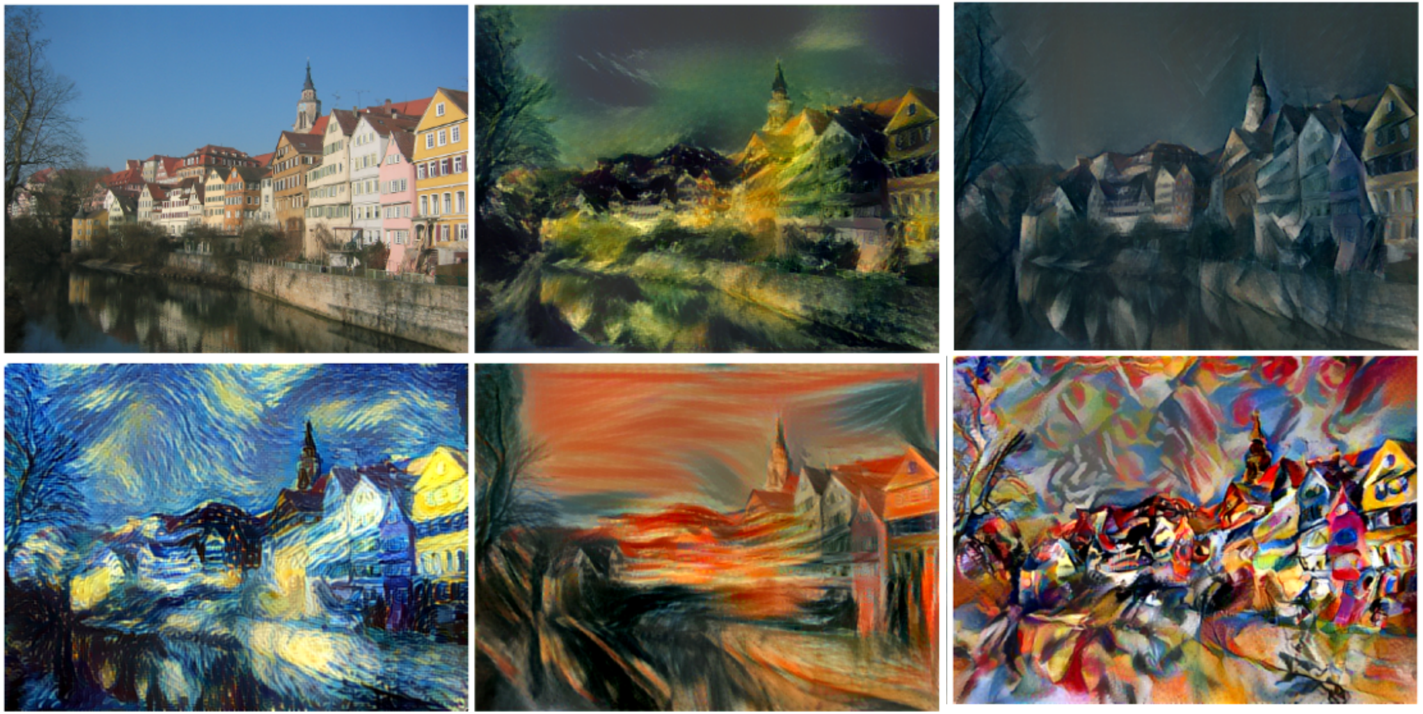
$\ell_{feat}^{\phi,relu3\_3}$

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. Copyright Springer, 2016. Reproduced for educational purposes.



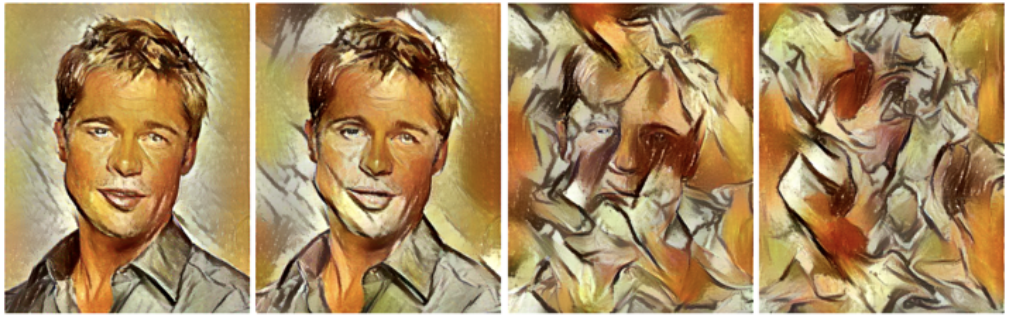
Neural Style Transfer

Example outputs from implementation (in Torch)



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

Neural Style Transfer

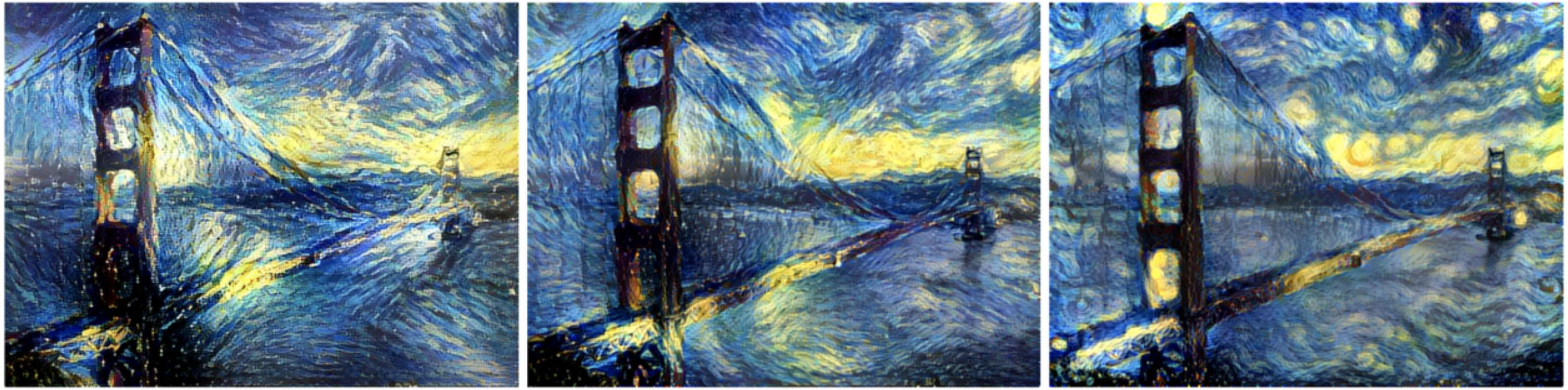


More weight to content loss

More weight to style loss

Neural Style Transfer

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

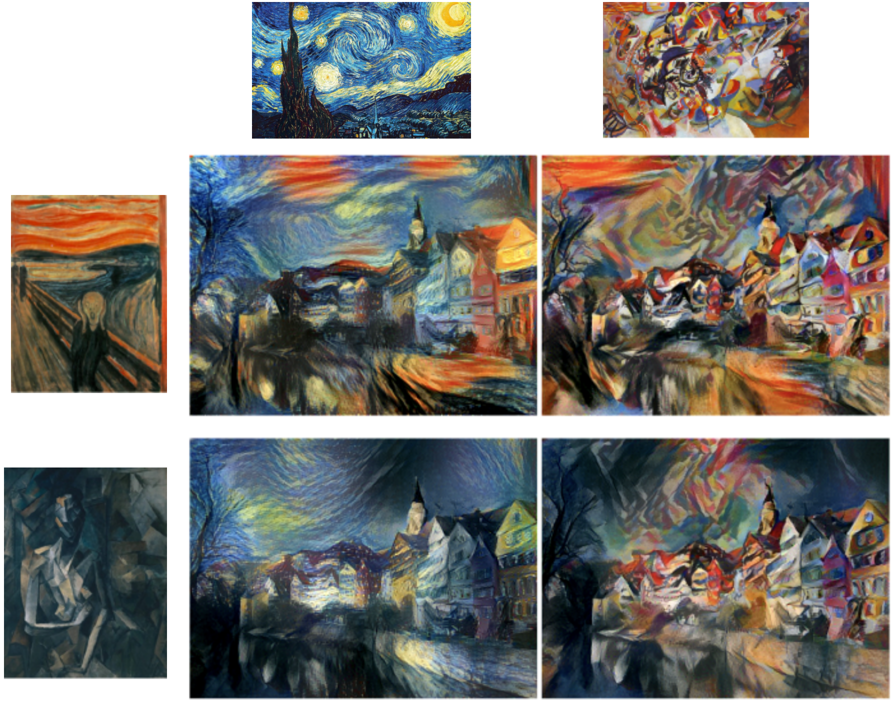


Larger style image ⟷ Smaller style image

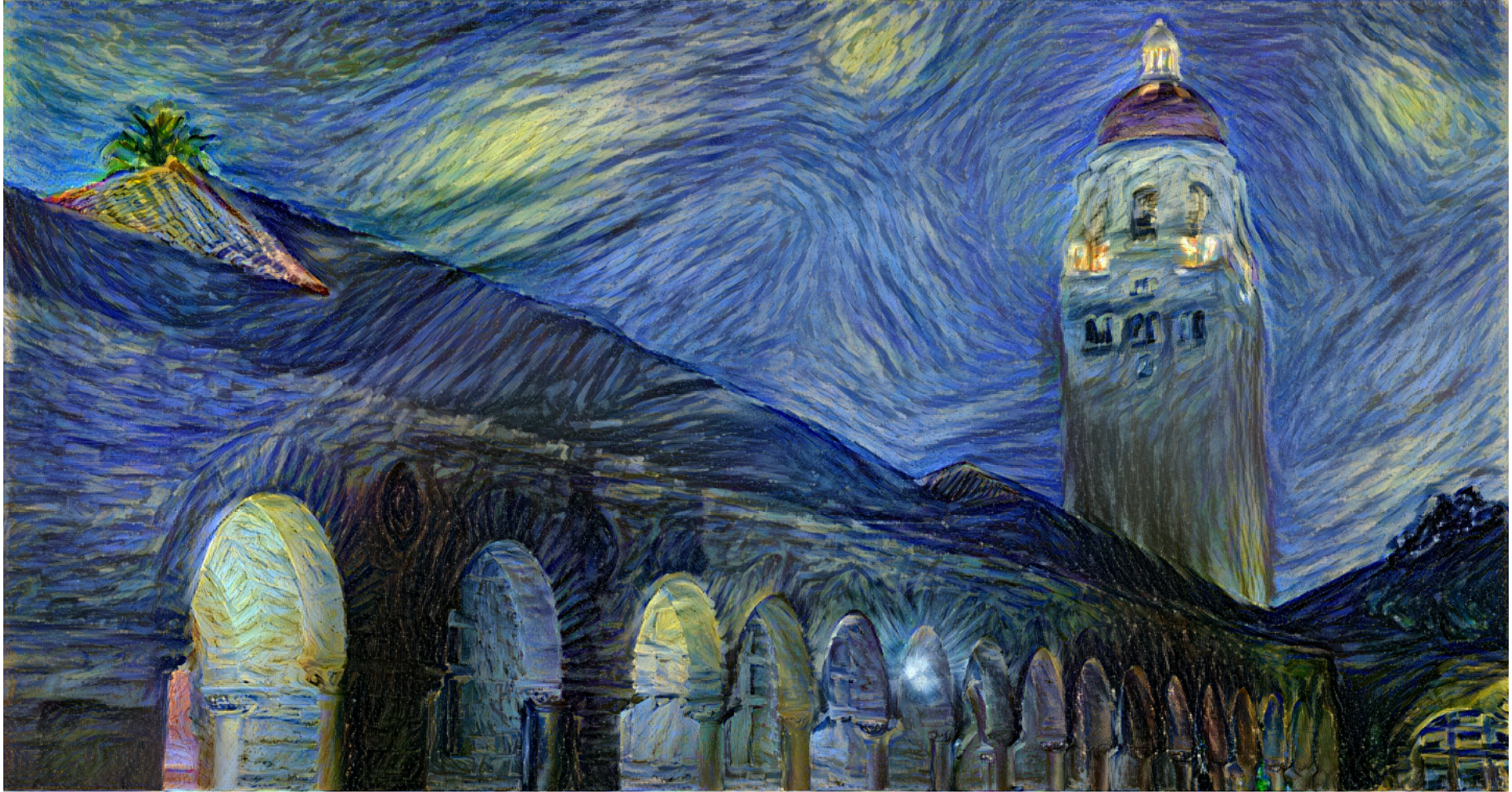
Gatys, Ecker, and Bethge, “Image style transfer using convolutional neural networks”, CVPR 2016
Figure copyright Justin Johnson, 2015.

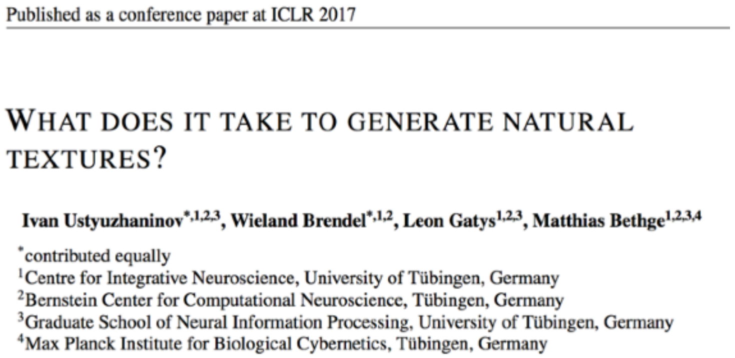
Neural Style Transfer: Multiple Style Images

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.



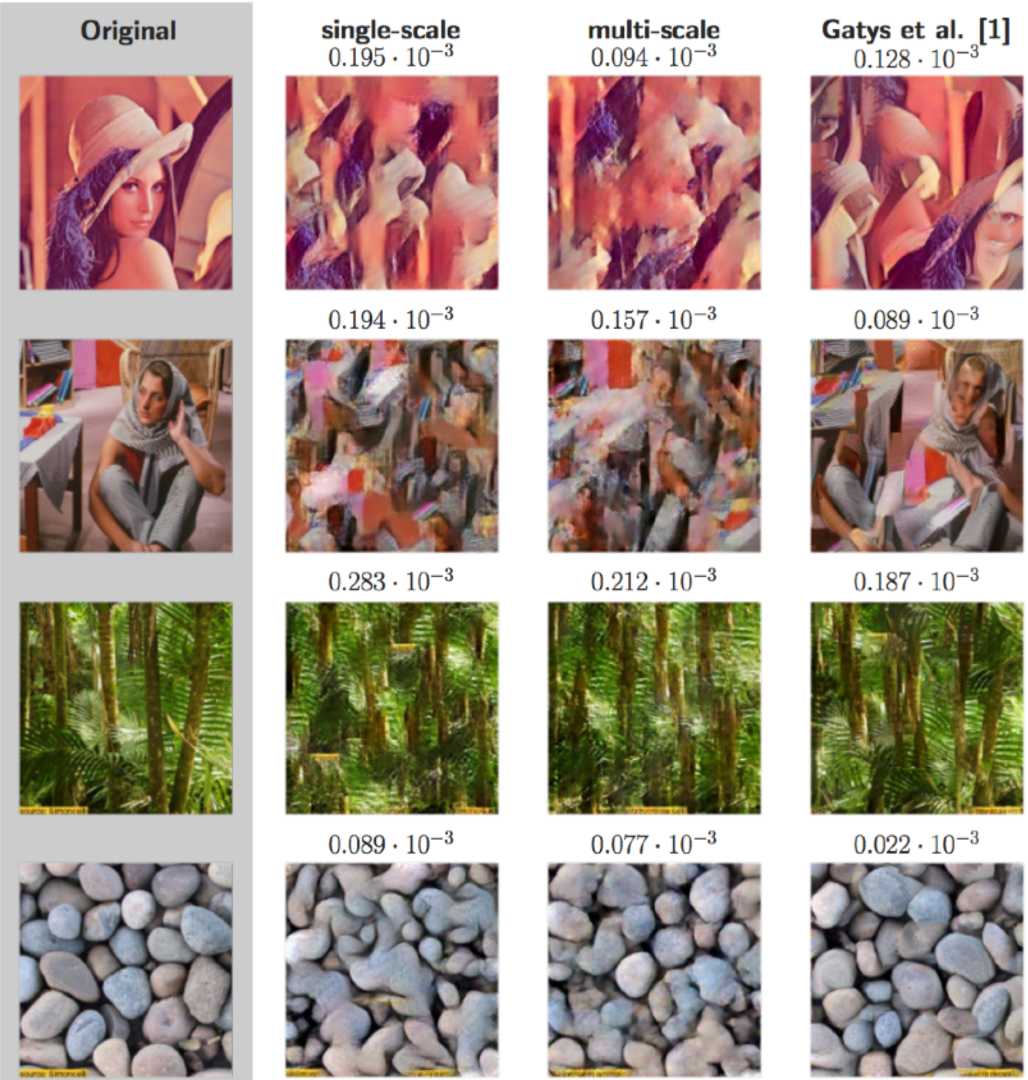


Published as a conference paper at ICLR 2017

WHAT DOES IT TAKE TO GENERATE NATURAL TEXTURES?

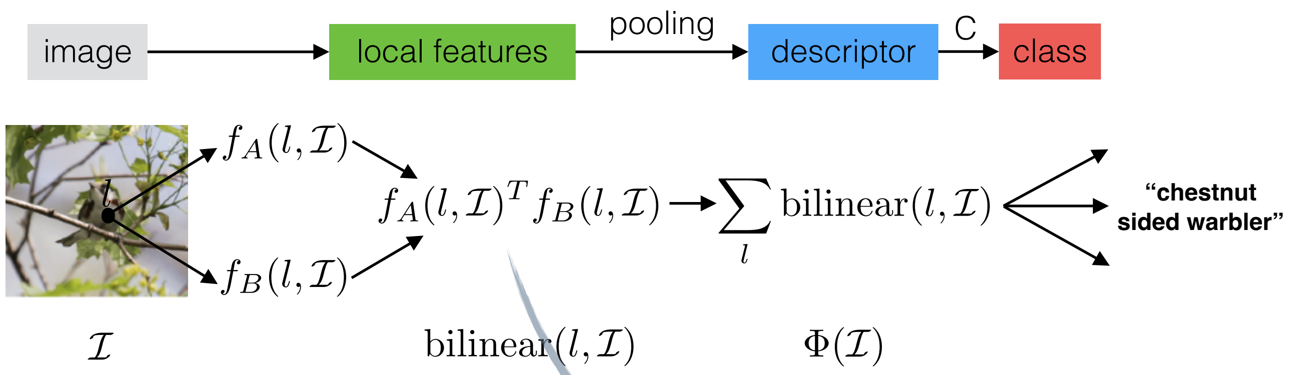
Ivan Ustyuzhaninov[*,1,2,3], **Wieland Brendel**[*,1,2], **Leon Gatys**[1,2,3], **Matthias Bethge**[1,2,3,4]

[*] contributed equally
[1] Centre for Integrative Neuroscience, University of Tübingen, Germany
[2] Bernstein Center for Computational Neuroscience, Tübingen, Germany
[3] Graduate School of Neural Information Processing, University of Tübingen, Germany
[4] Max Planck Institute for Biological Cybernetics, Tübingen, Germany


Original
single-scale
$0.195 \cdot 10^{-3}$
multi-scale
$0.094 \cdot 10^{-3}$
Gatys et al. [1]
$0.128 \cdot 10^{-3}$
$0.194 \cdot 10^{-3}$
$0.157 \cdot 10^{-3}$
$0.089 \cdot 10^{-3}$
$0.283 \cdot 10^{-3}$
$0.212 \cdot 10^{-3}$
$0.187 \cdot 10^{-3}$
$0.089 \cdot 10^{-3}$
$0.077 \cdot 10^{-3}$
$0.022 \cdot 10^{-3}$

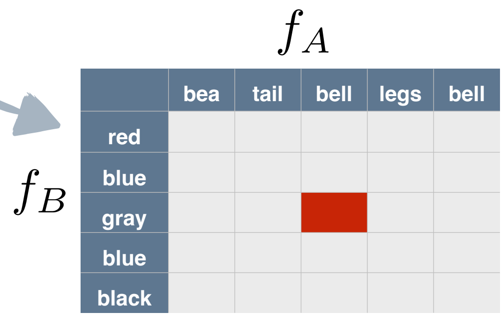
Bilinear (second-order) pooling

CNN activations pooled after outer-product encoding



Generalizes texture encoders

- Fisher vectors, Bag of Visual Words, VLAD
- Gram-matrix (when f_A=f_B)
- Excellent transfer from ImageNet to fine-grained domains (e.g., birds, cars, airplanes)



	bea	tail	bell	legs	bell
red					
blue					
gray			🟥		
blue					
black					

"gray belly"

Lin et al., Bilinear CNNs for Fine-grained Visual Recognition, ICCV 15, PAMI 17

Bilinear (second-order) pooling

Fine-grained classification (VGG-D + VGG-M networks)

CUB 200-2011
200 species, 11,788 images

FGVC Aircraft
100 variants, 10,000 images

Stanford cars
196 models, 16,185 images

Method	Birds	Aircraft	Cars
Fully connected [D]	70.4	76.6	79.8
Fisher vector [D]	74.7	78.7	85.7
Bilinear [D+D]	**84.0**	**83.9**	**90.6**
Bilinear [D+M]	**84.1**	**84.5**	**91.3**
Previous work	**84.1** [1]	**80.7** [2]	**92.6** [3]

Method	NABirds
Inception-BN	**73.1** [4]
B-CNN [D+M]	**79.4**

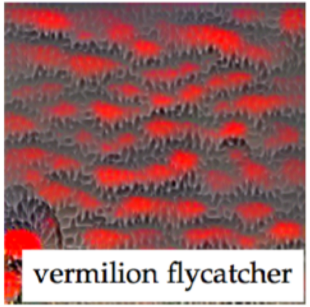
48,562 images of 555 categories

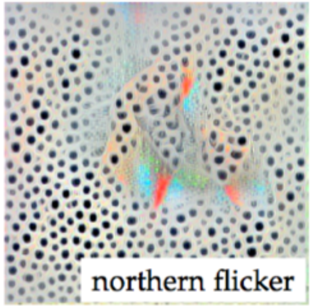
[1] Spatial Transformer Networks, Jaderberg et al., NIPS 15
[2] Revisiting the Fisher vector for Fine-grained Classification, Gosselin et al., PR Letters 14
[3] Fine-Grained Rec. w/o Part Annotations, Krause et al., CVPR 15
[4] Batch-normalized Inception Architectures, Szegedy et al., CVPR 15

Visualizing deep networks

“inverse” images for bilinear CNNs

Maximal images: $$\arg\max_{\mathcal{I}} \log P(c|\mathcal{I}, \mathbf{W}) + \log \Gamma(\mathcal{I})$$

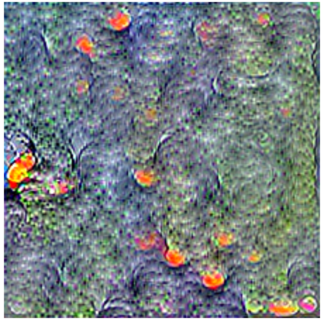




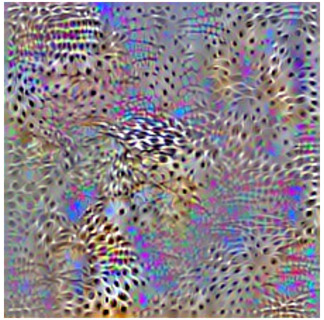
Lin and Maji, Visualizing and Understanding Deep Texture Representations, CVPR 16

Visualizing deep networks

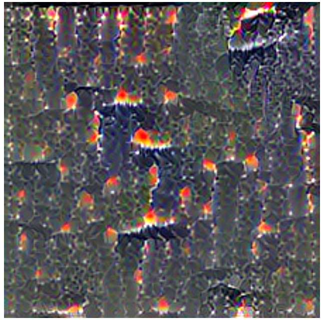
What texture are birds?



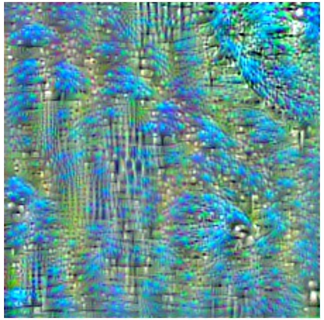
Crested aucket



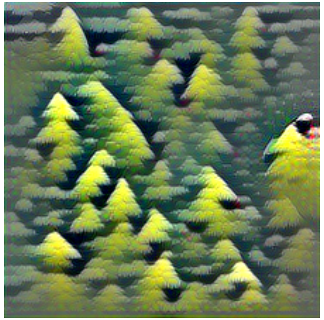
Cactus wren



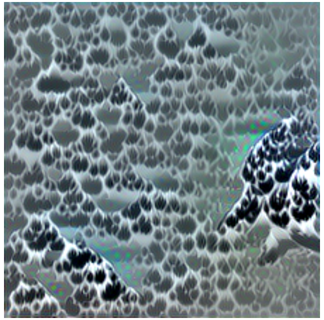
Red winged blackbird



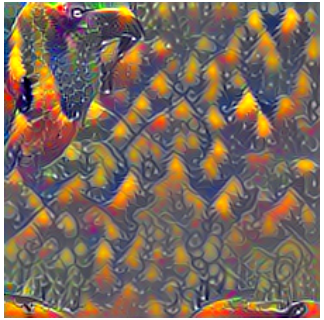
Indigo bunting



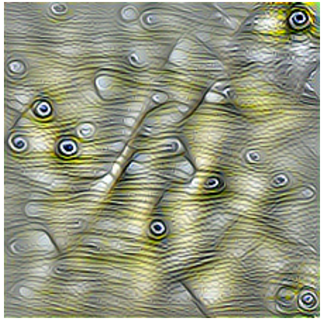
American goldfinch



Pied kingfisher



Hooded oriole

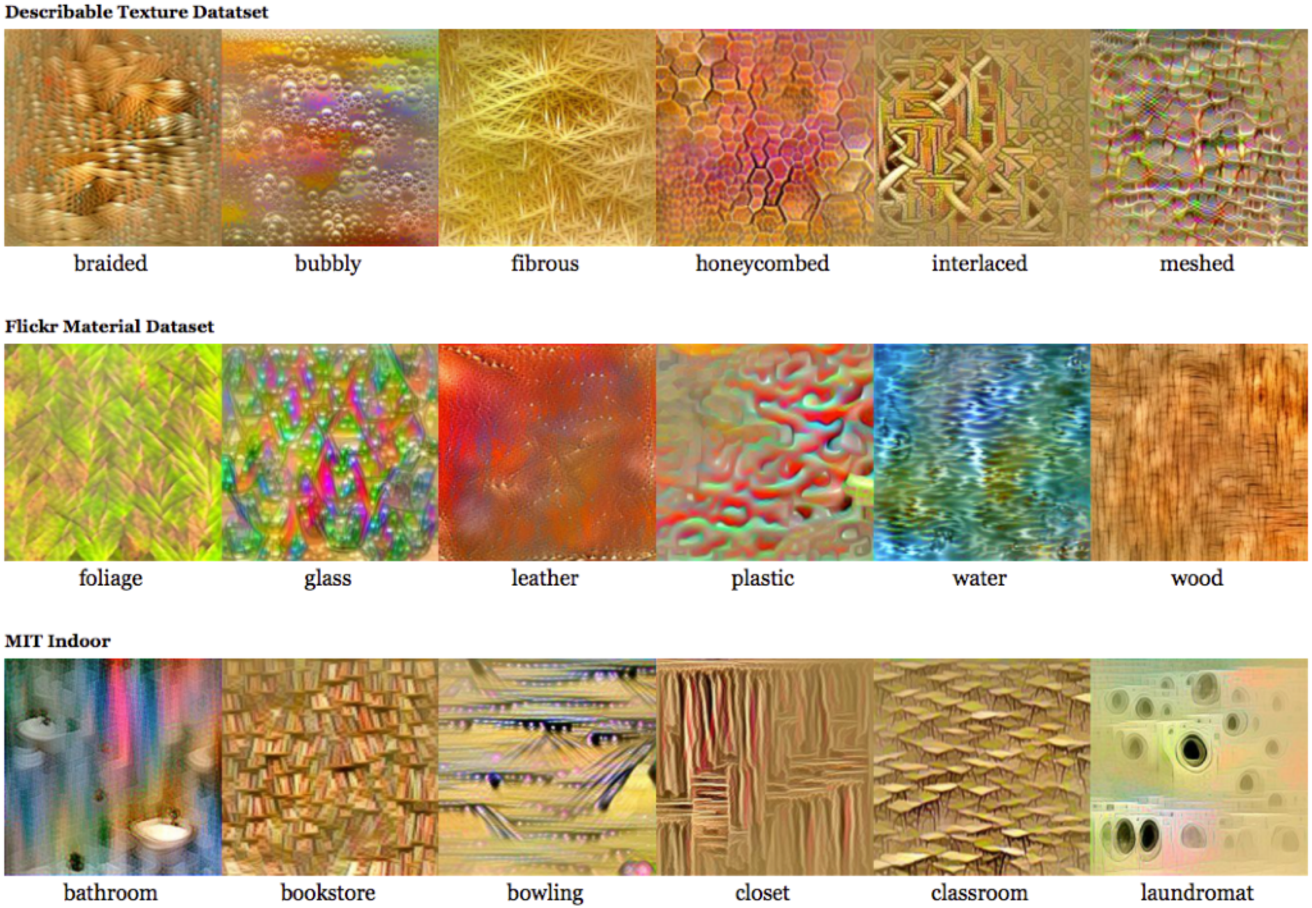


White eyed vireo

Lin and Maji, Visualizing and Understanding Deep Texture Representations, CVPR 16

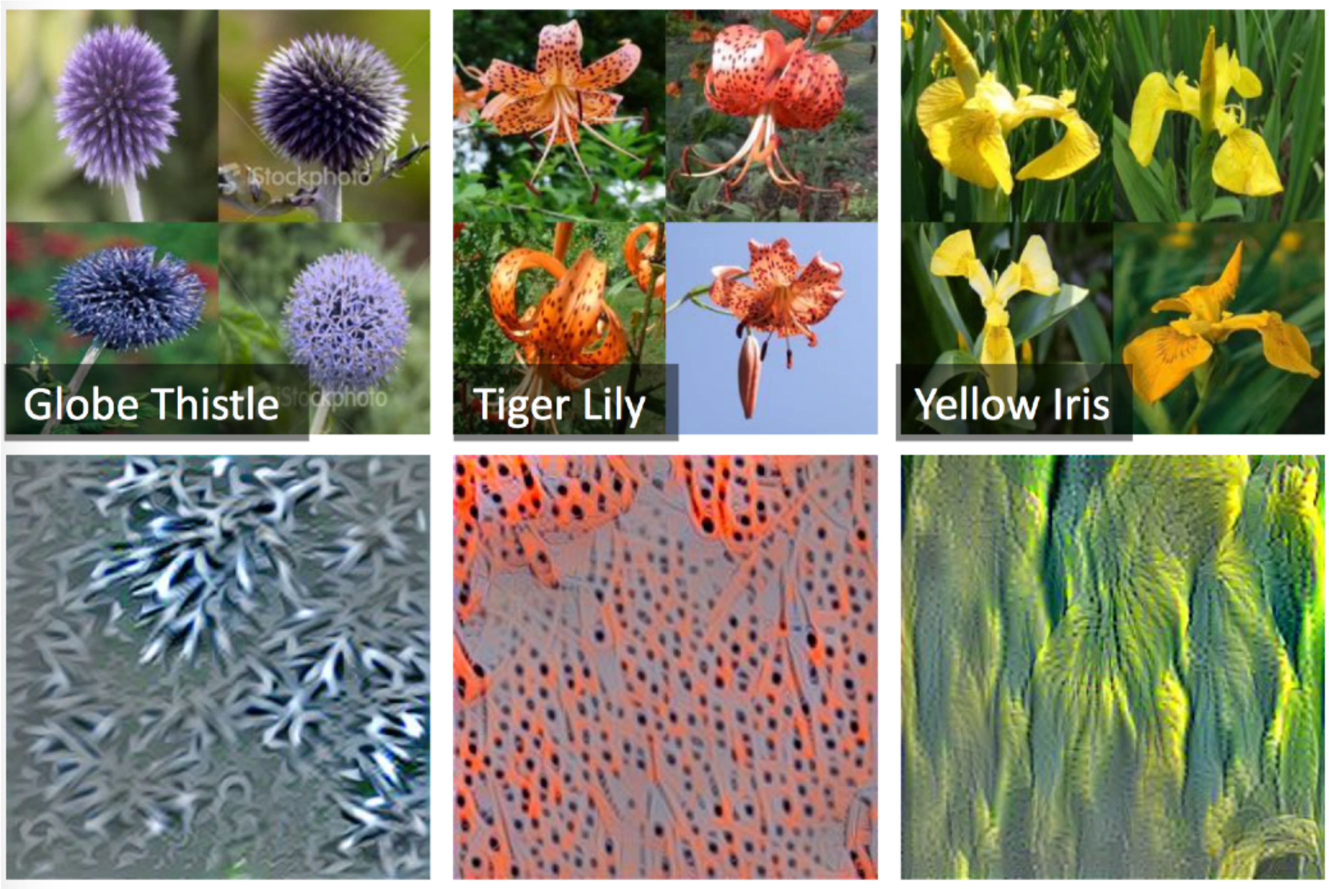
Visualizing deep networks

What texture are bookstores?



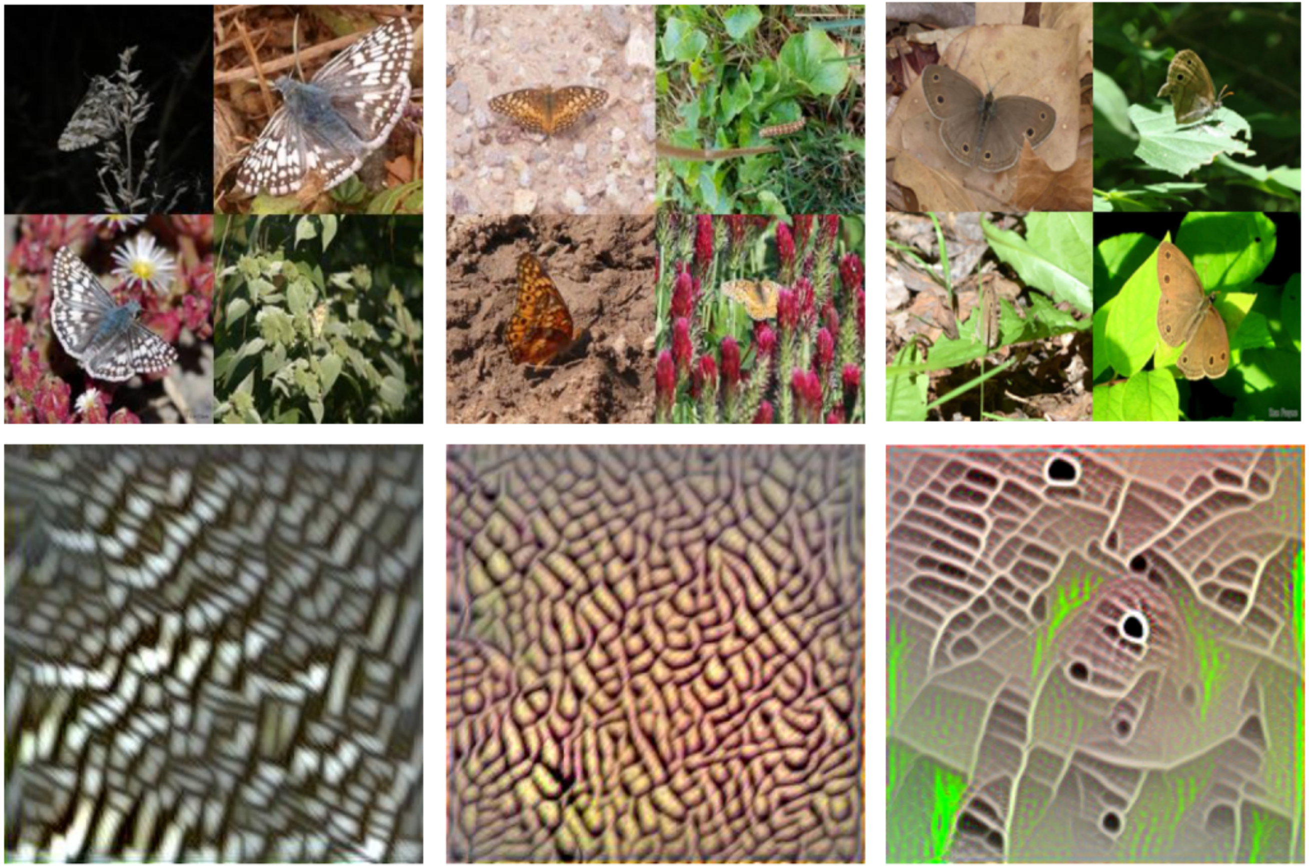
Visualizing deep networks

Oxford flowers



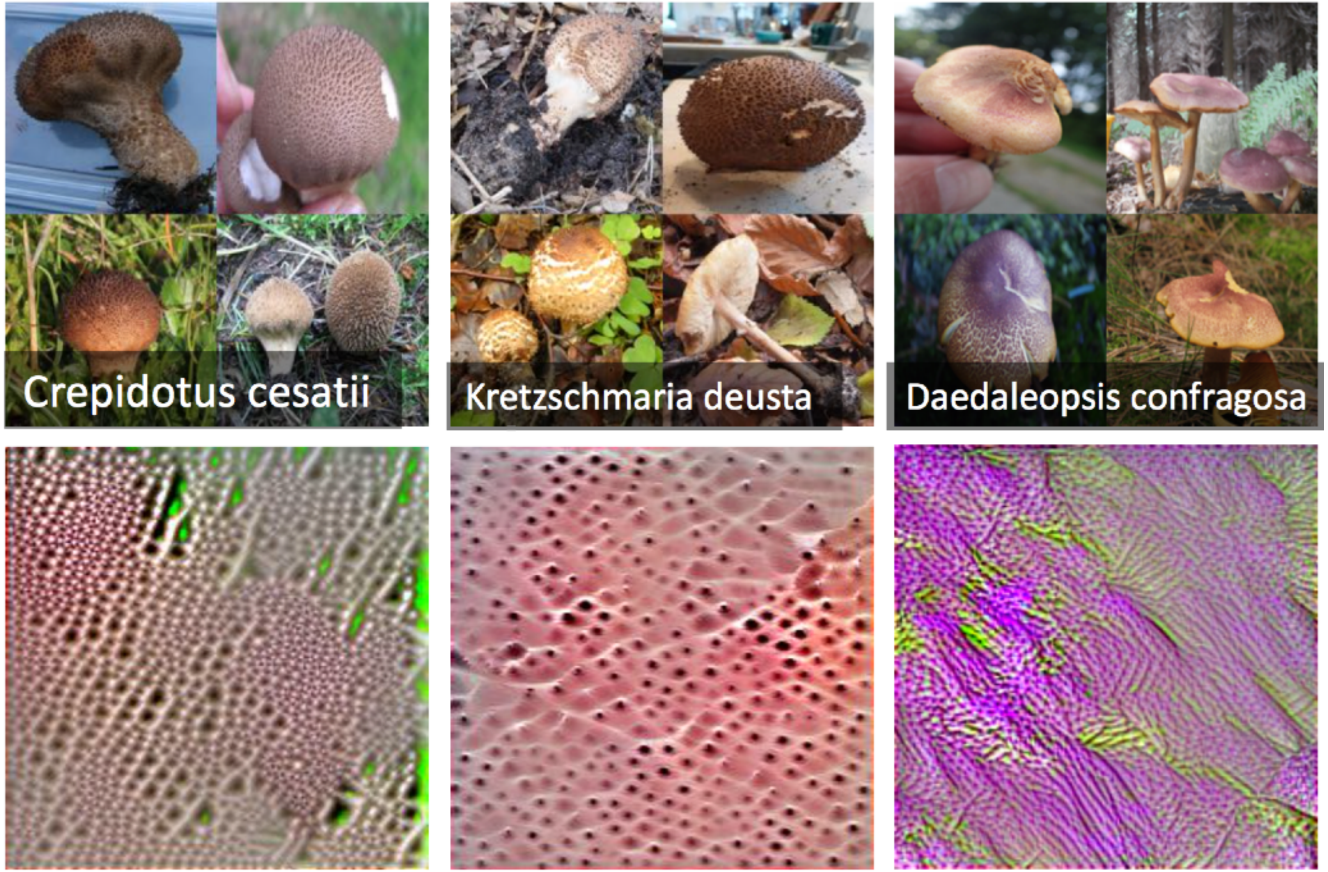
Visualizing deep networks

FGVC butterflies and moths



Visualizing deep networks

FGVC fungi



Summary

Many methods for understanding CNN representations

Activations: Nearest neighbors, Dimensionality reduction, maximal patches, occlusion
Gradients: Saliency maps, class visualization, feature inversion
Fun: DeepDream, Texture Synthesis, Style Transfer
Bonus: Works for fine-grained categorization too!