# Lecture 15:

## Adversarial examples and style transfer

# Announcements

Happy halloween!

Homework 3 released

Attend MLFL on Thursday (12-1pm) instead of lecture

James Tompkin

Assistant Professor, Brown University

**When:** Thursday, 11/2/2023, 12pm-1pm
**Where:** CS 150/151 **(pizzas available)**, Zoom
**Title:** More Cameras and Better Cameras for Scene Reconstruction

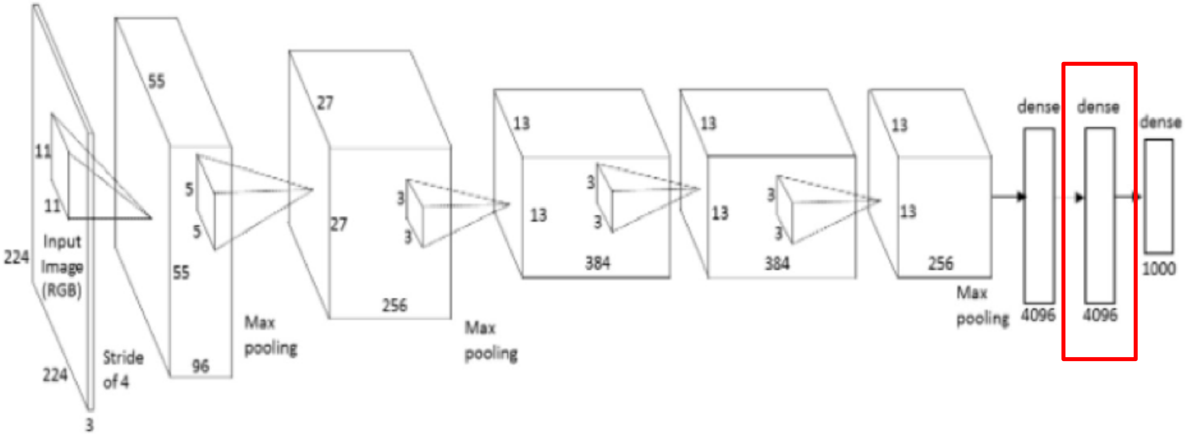
# Agenda

Visualizing CNNs via inversion (wrap up)

Adversarial examples

Style transfer

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



Find an image such that:

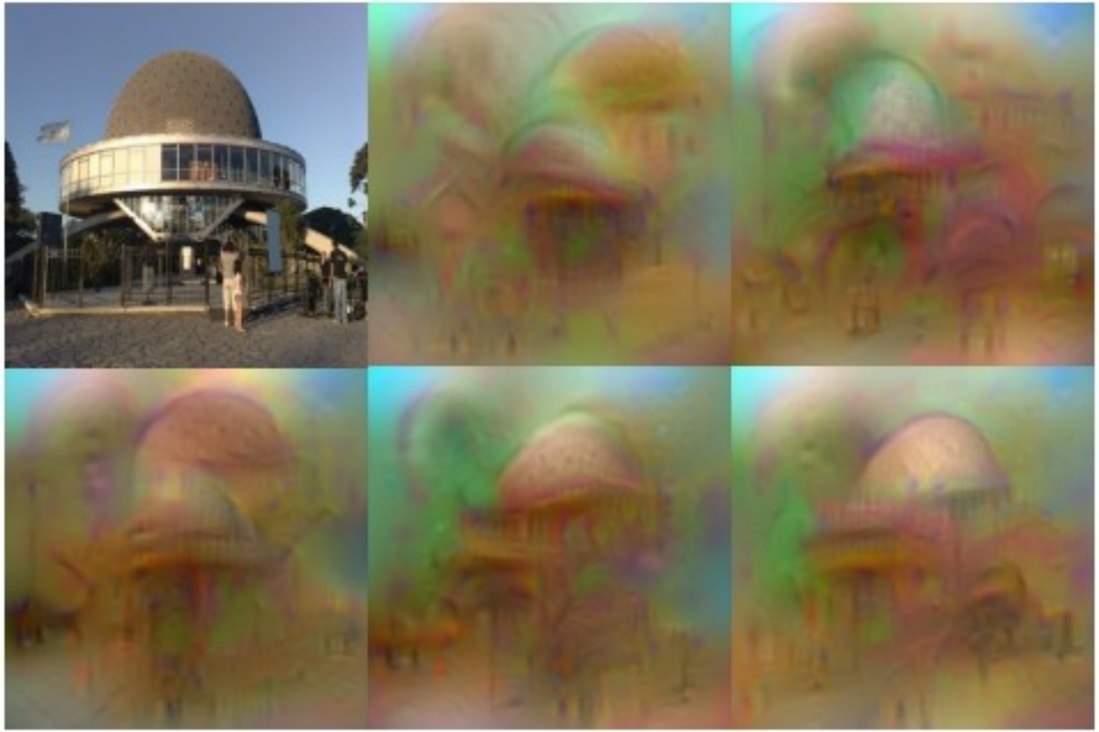
- Its code is similar to a given code
- It "looks natural" (image prior regularization)

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

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

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

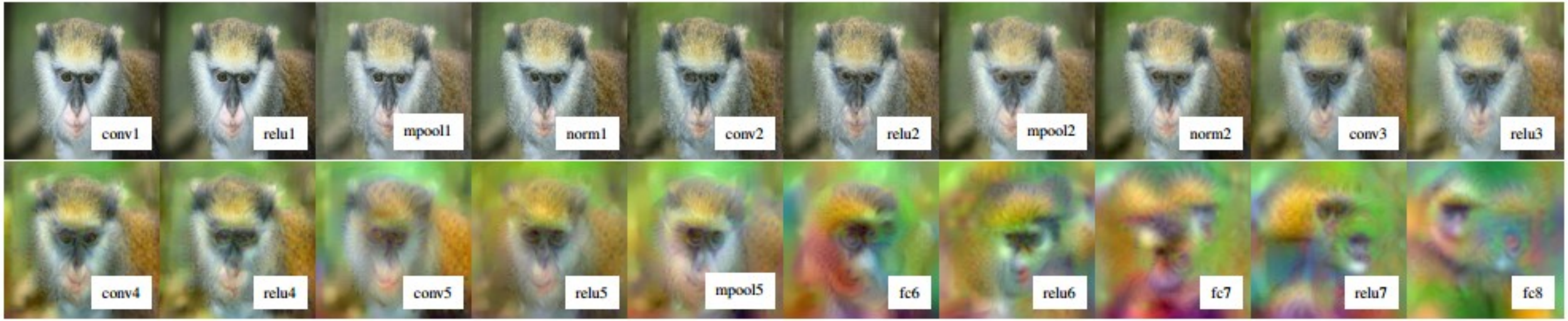
original image

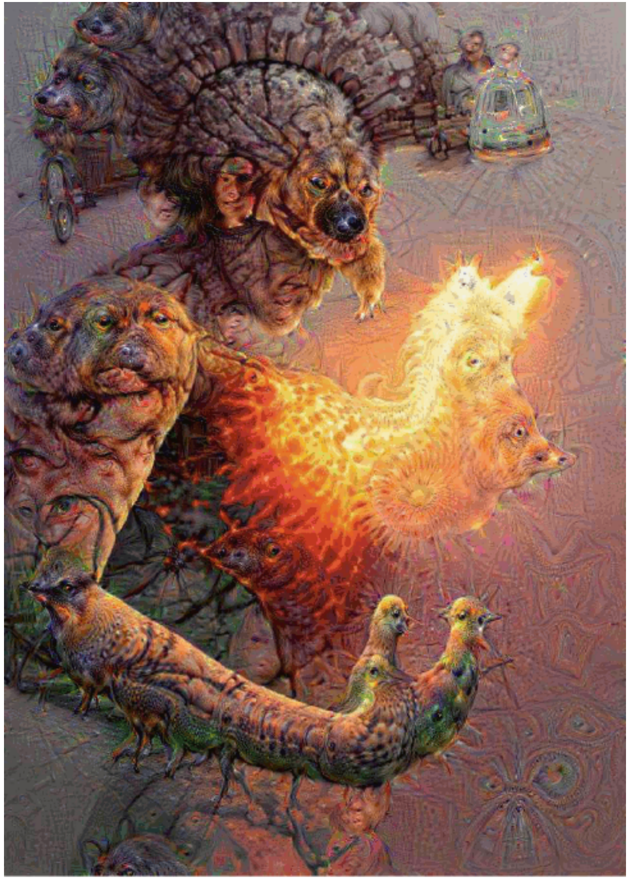


reconstructions from the 1000 log probabilities for ImageNet (ILSVRC) classes

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

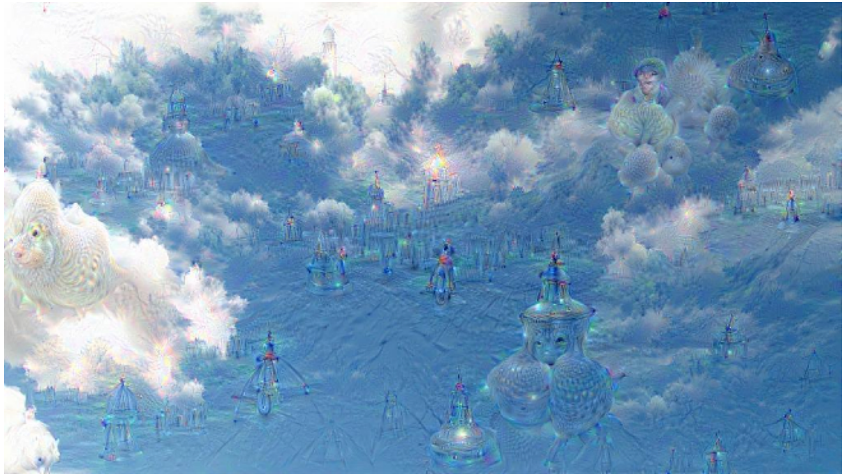

# Reconstructions from intermediate layers



DeepDream https://github.com/google/deepdream

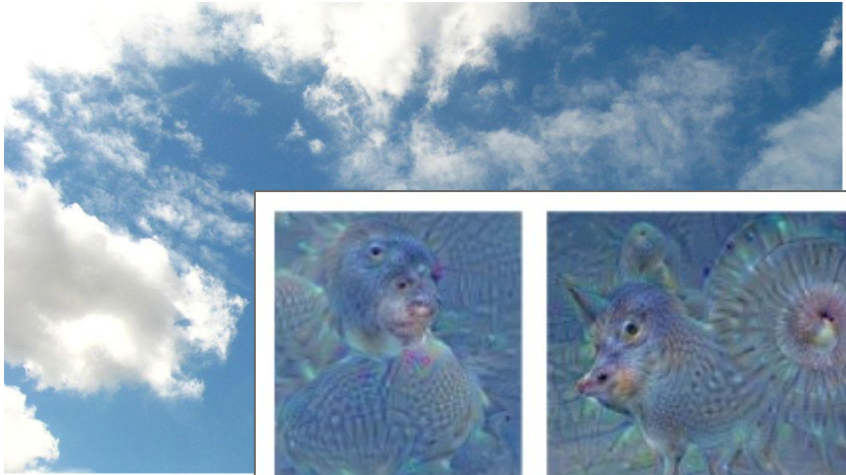
inception_4c/output

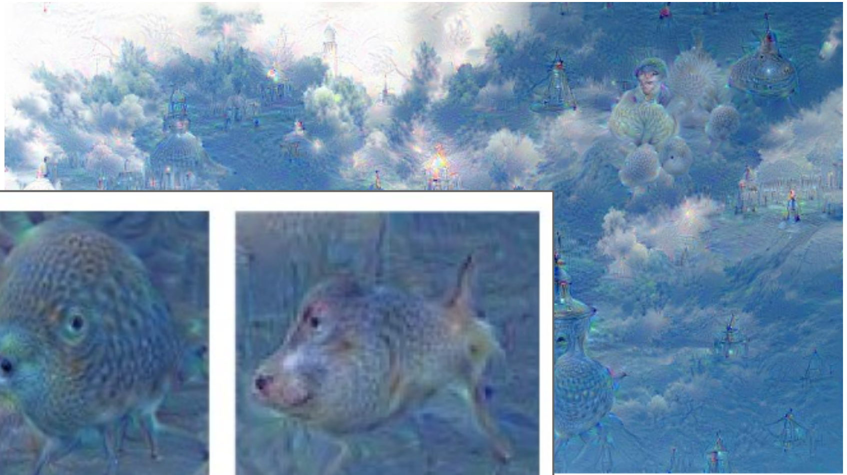


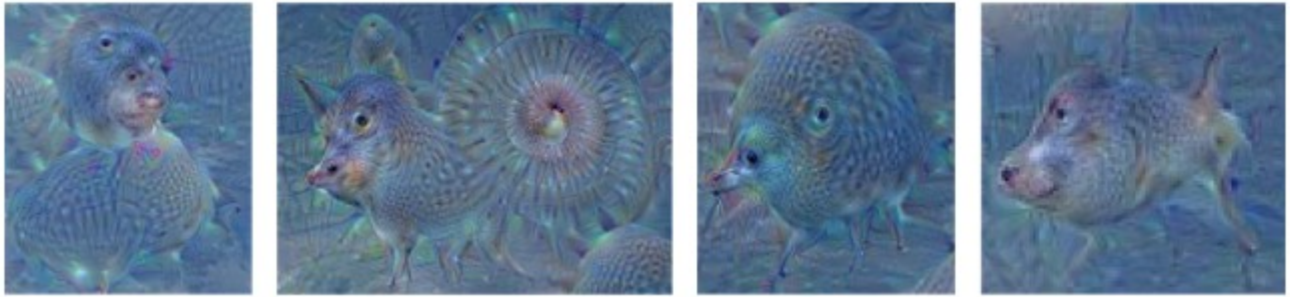
DeepDream modifies the image in a way that "boosts" all activations, at any layer

this creates a feedback loop: e.g. any slightly detected dog face will be made more and more dog like over time

inception_4c/output

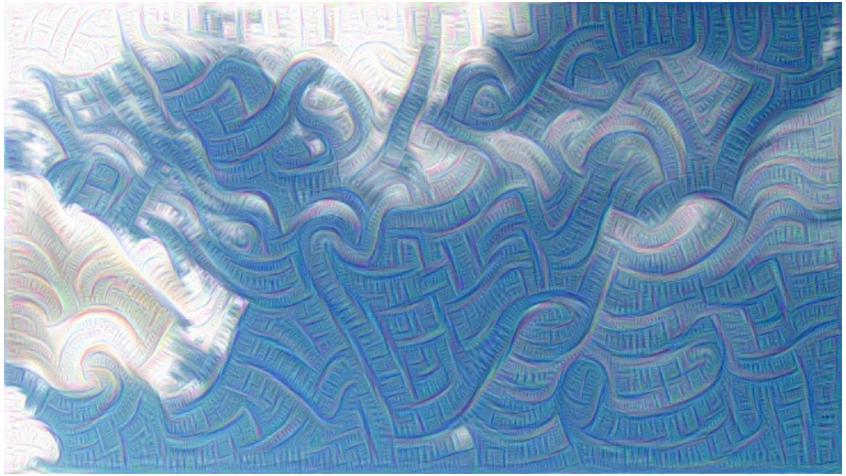






"Admiral Dog!" "The Pig-Snail" "The Camel-Bird" "The Dog-Fish"

DeepDream modifies the image in a way that "boosts" all activations, at any layer inception_3b/5x5_reduce



DeepDream modifies the image in a way that “boosts” all activations, at any layer

```python
def objective_L2(dst):
    dst.diff[:] = dst.data

def make_step(net, step_size=1.5, end='inception_4c/output',
              jitter=32, clip=True, objective=objective_L2):
    '''Basic gradient ascent step.'''

    src = net.blobs['data'] # input image is stored in Net's 'data' blob
    dst = net.blobs[end]

    ox, oy = np.random.randint(-jitter, jitter+1, 2)
    src.data[0] = np.roll(np.roll(src.data[0], ox, -1), oy, -2) # apply jitter shift

    net.forward(end=end)
    objective(dst)  # specify the optimization objective
    net.backward(start=end)
    g = src.diff[0]
    # apply normalized ascent step to the input image
    src.data[:] += step_size/np.abs(g).mean() * g

    src.data[0] = np.roll(np.roll(src.data[0], -ox, -1), -oy, -2) # unshift image

    if clip:
        bias = net.transformer.mean['data']
        src.data[:] = np.clip(src.data, -bias, 255-bias)
```

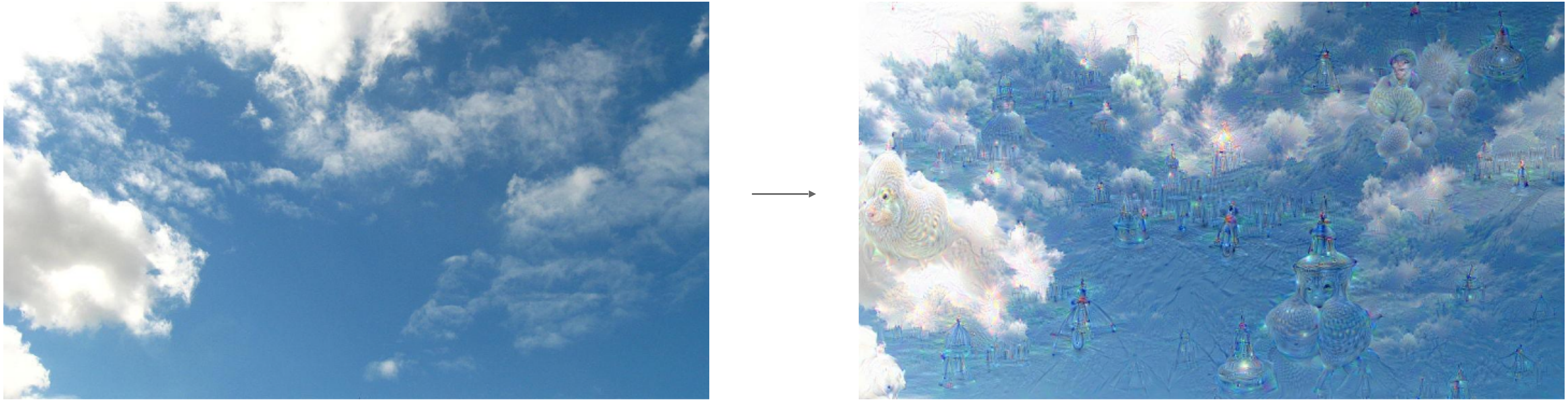
```
def objective_L2(dst):
    dst.diff[:] = dst.data

def make_step(net, step_size=1.5, end='inception_4c/output',
              jitter=32, clip=True, objective=objective_L2):
    '''Basic gradient ascent step.'''

    src = net.blobs['data'] # input image is stored in Net's 'data' blob
    dst = net.blobs[end]

    ox, oy = np.random.randint(-jitter, jitter+1, 2)
    src.data[0] = np.roll(np.roll(src.data[0], ox, -1), oy, -2) # apply jitter shift

    net.forward(end=end)
    objective(dst)  # specify the optimization objective
    net.backward(start=end)
    g = src.diff[0]
    # apply normalized ascent step to the input image
    src.data[:] += step_size/np.abs(g).mean() * g

    src.data[0] = np.roll(np.roll(src.data[0], -ox, -1), -oy, -2) # unshift image

    if clip:
        bias = net.transformer.mean['data']
        src.data[:] = np.clip(src.data, -bias, 255-bias)
```

DeepDream: set dx = x :)

jitter regularizer

“image update”

inception_4c/output



DeepDream modifies the image in a way that “boosts” all activations, at any layer

this creates a <u>feedback loop</u>: e.g. any slightly detected dog face will be made more and more dog like over time

# Bonus videos

Deep Dream Grocery Trip
https://www.youtube.com/watch?v=DgPaCWJL7XI

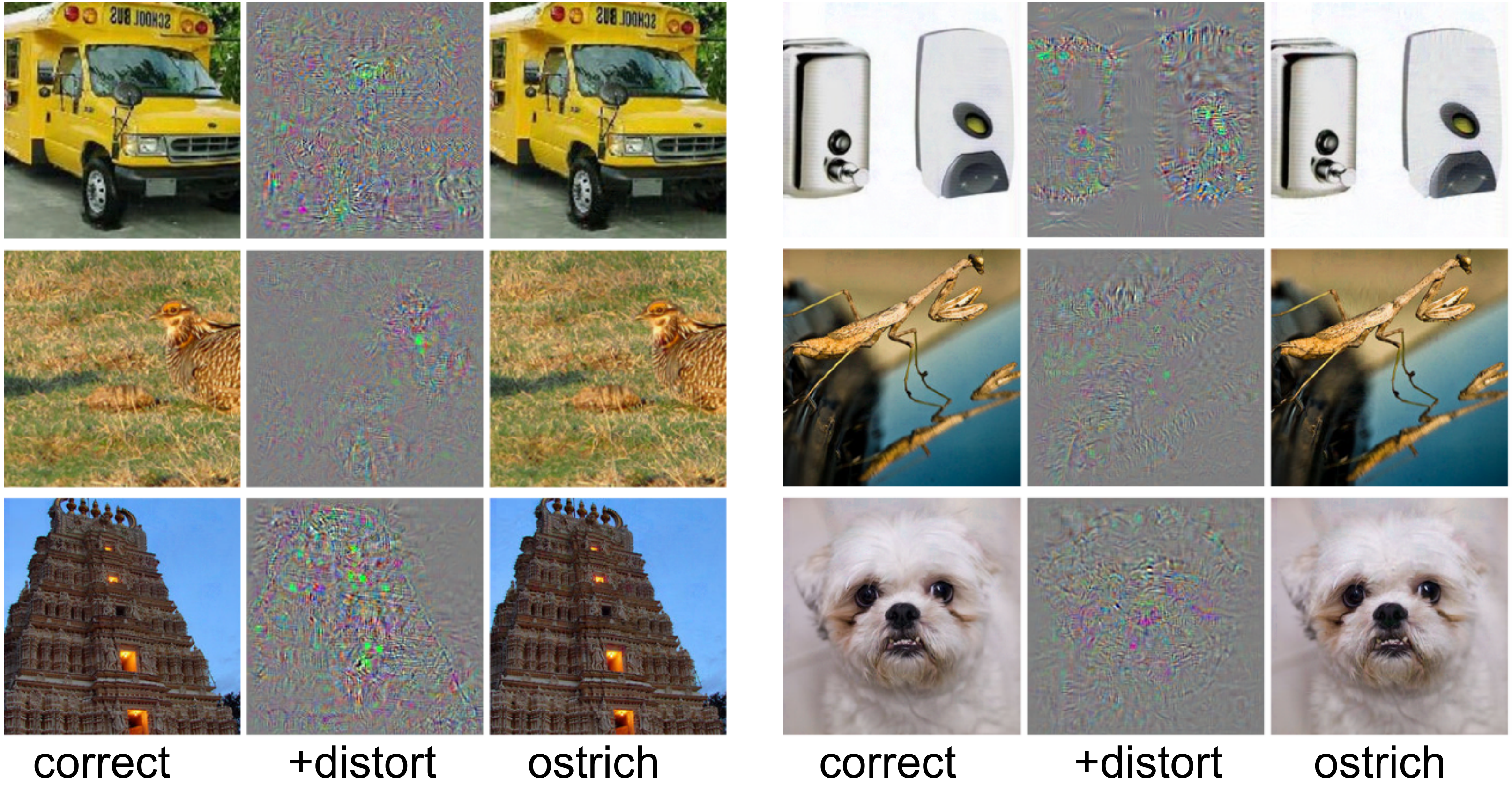
Deep Dreaming Fear & Loathing in Las Vegas: the Great San Francisco Acid Wave
https://www.youtube.com/watch?v=oyxSerkkP4o

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?

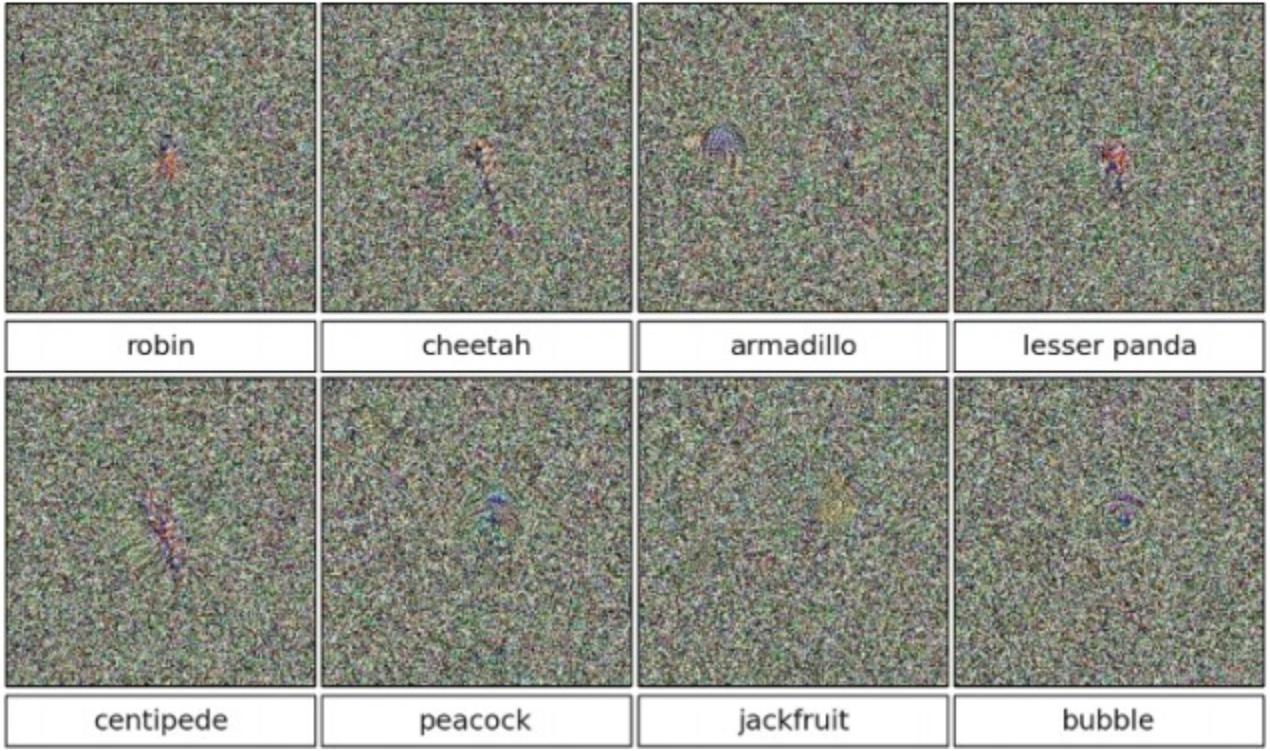
spoiler alert: yeah

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



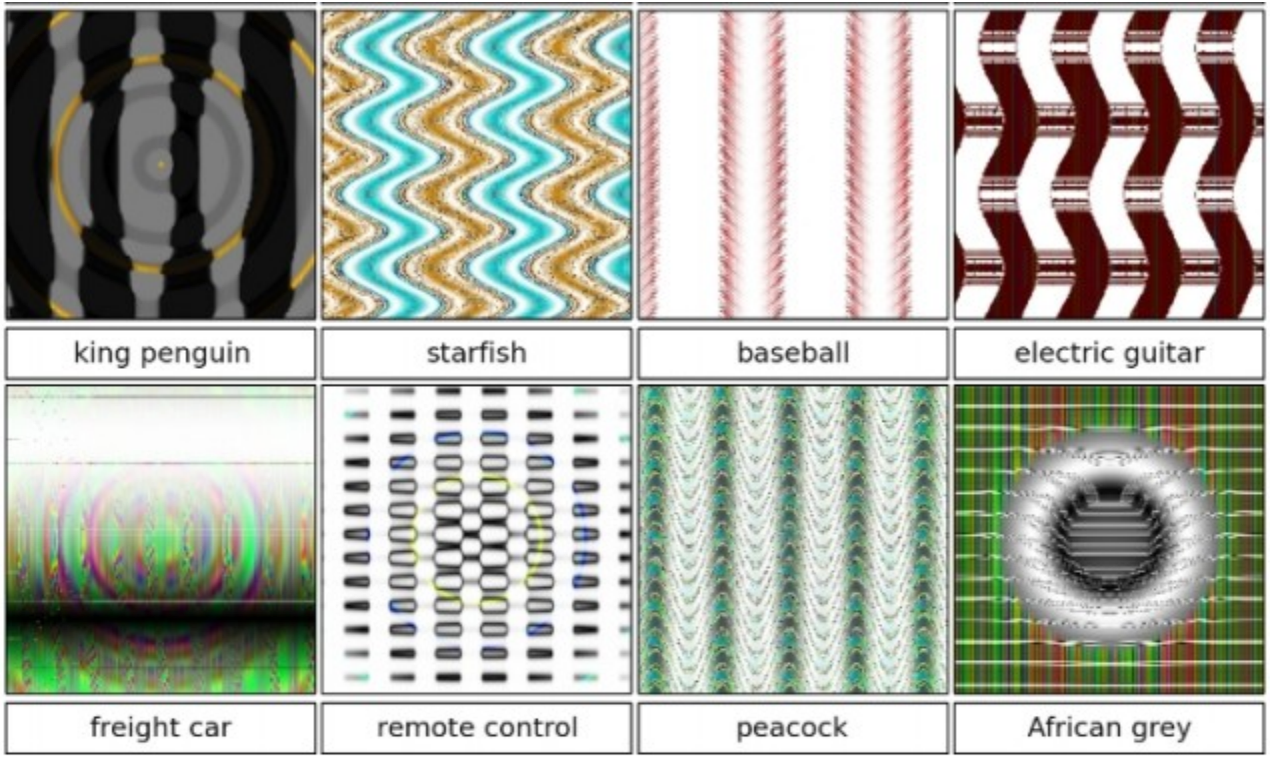
[Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images
Nguyen, Yosinski, Clune, 2014]

>99.6%
confidences



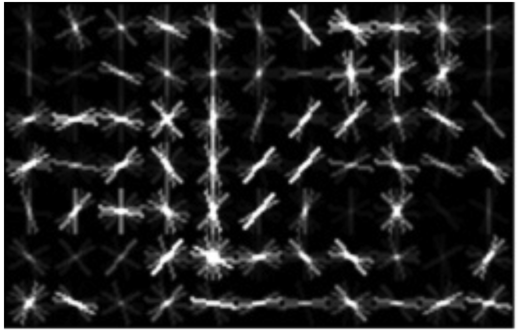
*[Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images Nguyen, Yosinski, Clune, 2014]*

>99.6% confidences



# These kinds of results were around even before ConvNets…

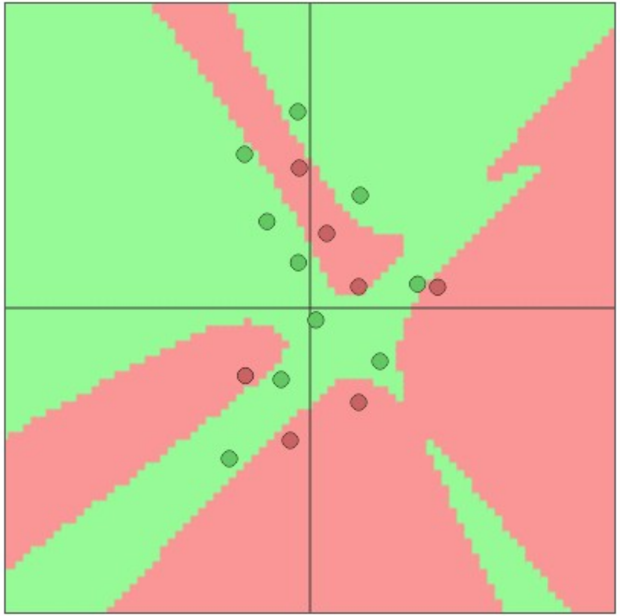
*[Exploring the Representation Capabilities of the HOG Descriptor, Tatu et al., 2011]*



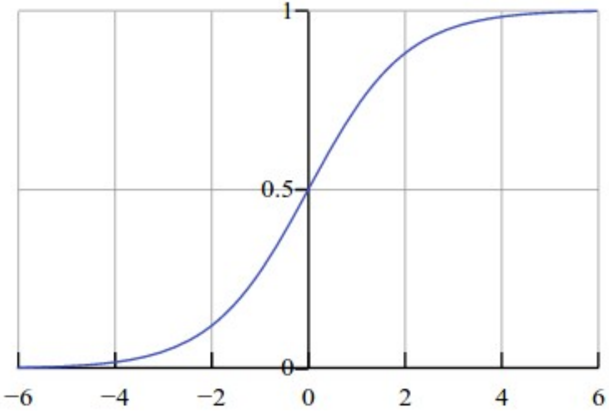
Identical HOG represention

*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 | ← input example |
| w | -1 | -1 | 1 | -1 | 1 | -1 | 1 | 1 | -1 | 1 | ← weights |
| adversarial x | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | |

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%**

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

# 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

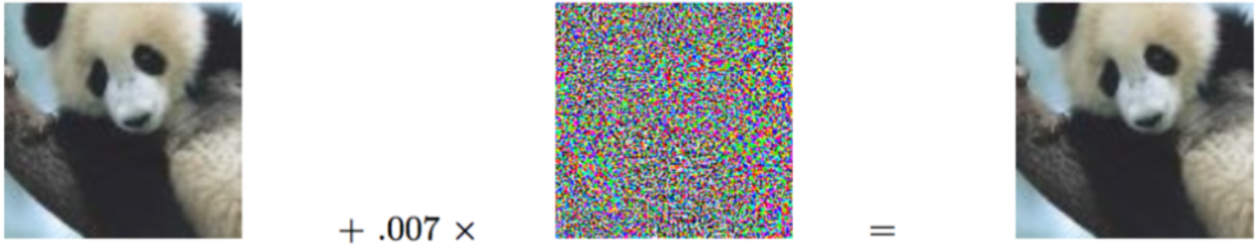
**i.e. we improved the class 1 probability from 5% to 88%**

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)

# Adversarial perturbations

Adding adversarial noise can cause the network to predict a wrong label, even though the change is imperceptible to us



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

| $\boldsymbol{x}$ | $\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$ | $\boldsymbol{x} + \epsilon \cdot \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$ |
|---|---|---|
| $y =$"panda" | "nematode" | "gibbon" |
| w/ 57.7% confidence | w/ 8.2% confidence | w/ 99.3 % confidence |

**Explaining and Harnessing Adversarial Examples**

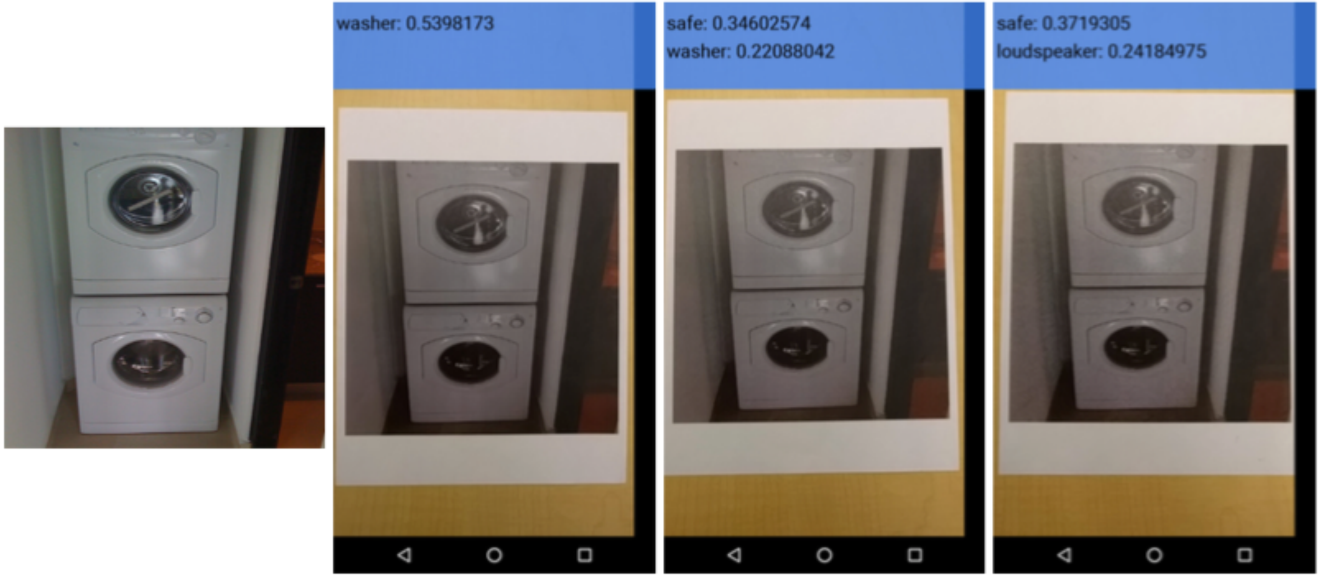
Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy

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

correct +distort ostrich correct +distort ostrich

# Adversarial perturbations

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

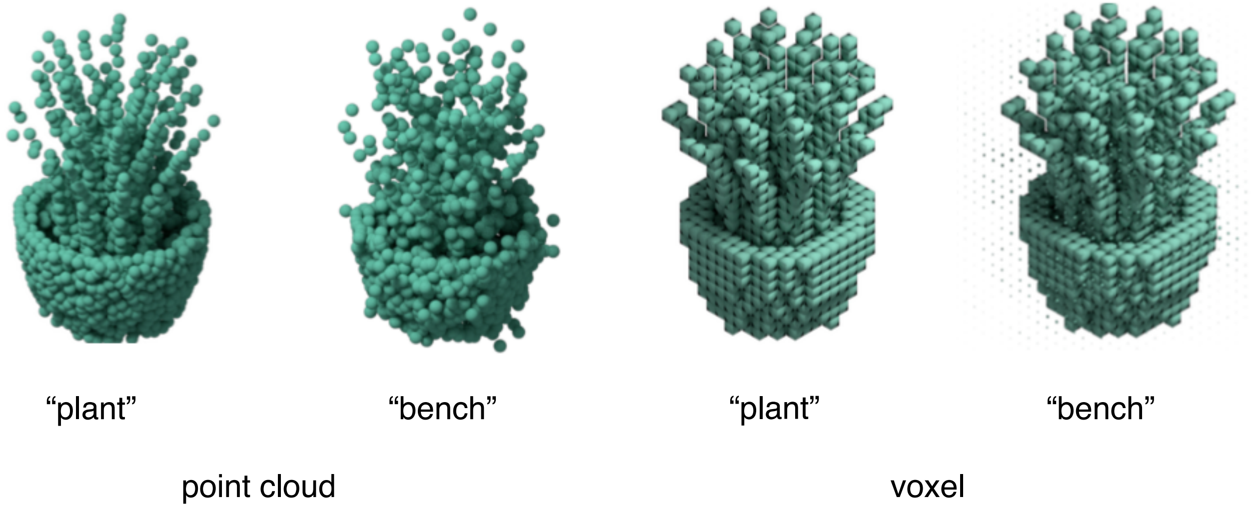



(a) Image from dataset (b) Clean image (c) Adv. image, $\epsilon = 4$ (d) Adv. image, $\epsilon = 8$

# Adversarial perturbations

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

Su et al., ECCV 2018



# Neural Style Transfer

Content Image



+

Style Image



=

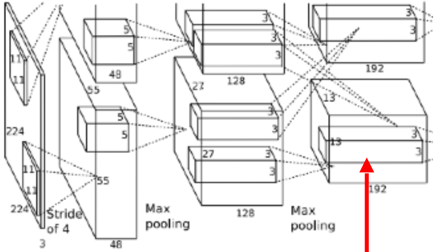
Style Transfer!

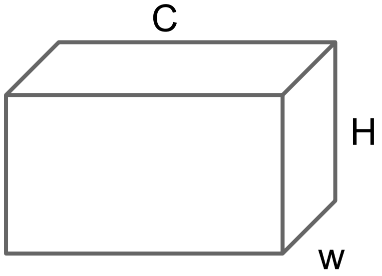


Gatys, Ecker, and Bethge, “Image style transfer using convolutional neural networks”, CVPR 2016

# Neural Texture Synthesis: Gram Matrix

This image is in the public domain.

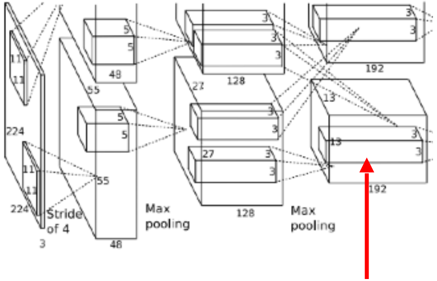


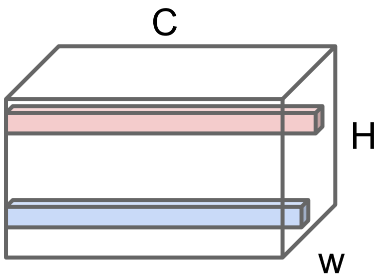



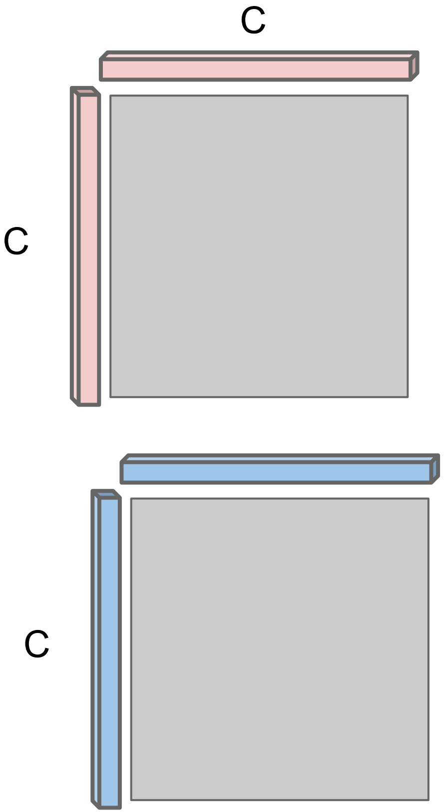

Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

# Neural Texture Synthesis: Gram Matrix

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



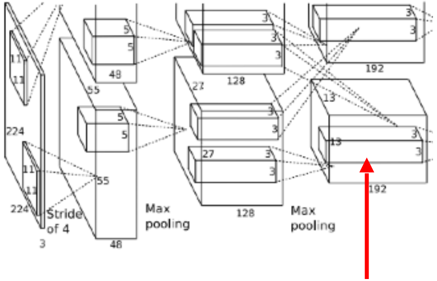



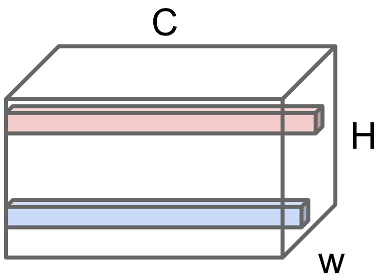

Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

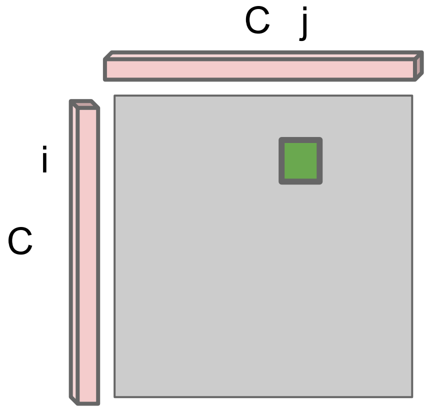
Outer product of C-dimensional vector with itself gives C x C matrix measuring co-occurrence

# Neural Texture Synthesis: Gram Matrix

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



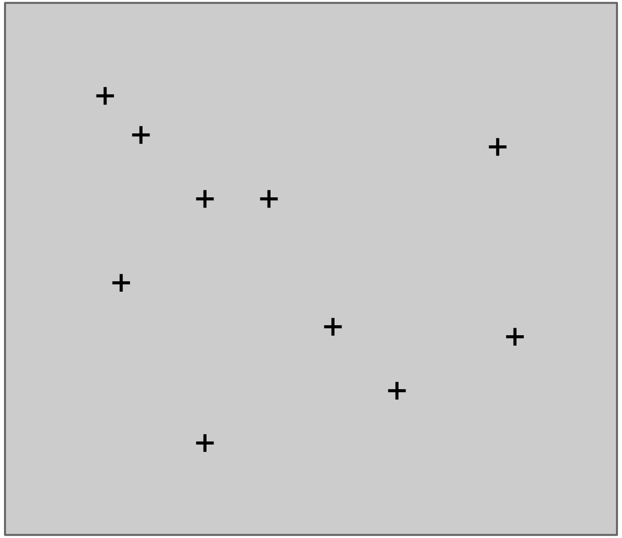

Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

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

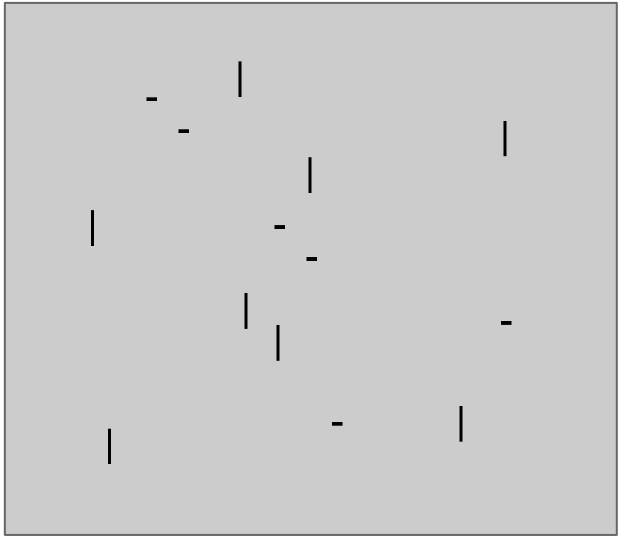
The green box G(i,j) represents the AVERAGE, over image positions in the image, of the product of features i and j.

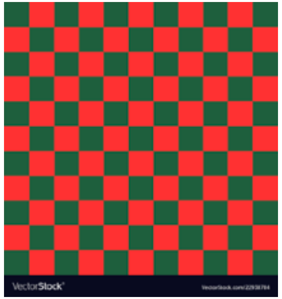
Suppose "i" represents horizontal lines and "j" represents vertical lines.

Vertical and horizontal co-occur



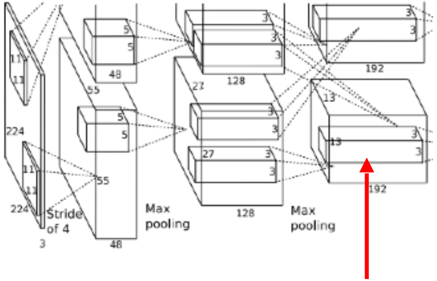
Vertical and horizontal DO NOT co-occur

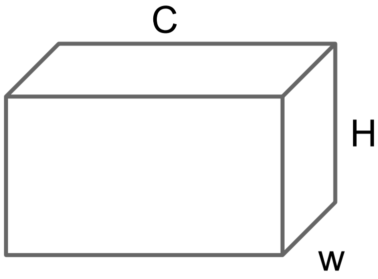


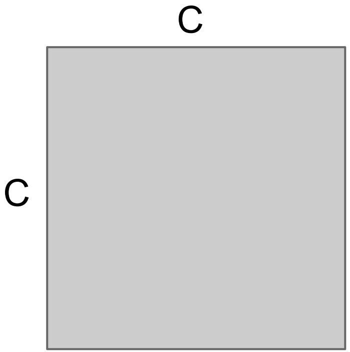

VectorStock

# Neural Texture Synthesis: Gram Matrix

This image is in the public domain.




Gram Matrix

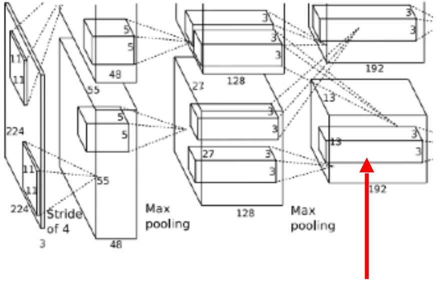
Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

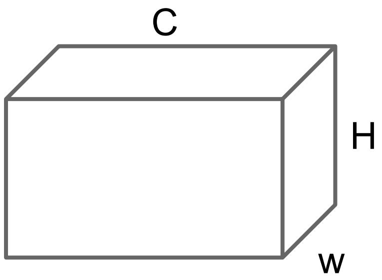
Outer product of C-dimensional vector with itself gives C x C matrix measuring co-occurrence

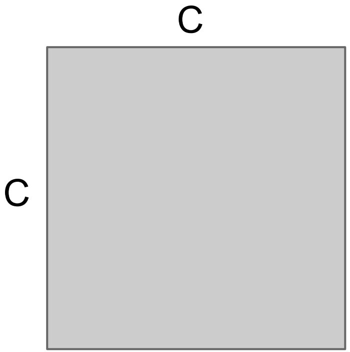
Average over all HW outer products, giving **Gram matrix** of shape C x C

# 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

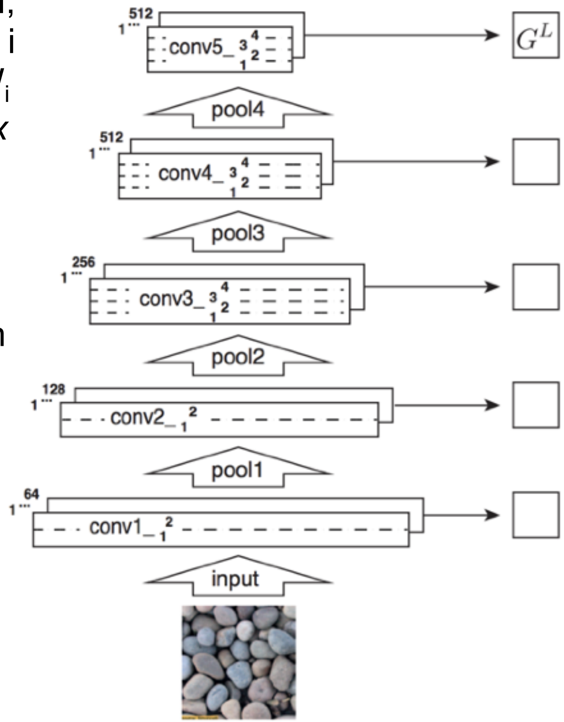
C x H x W to =C x HW

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:

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



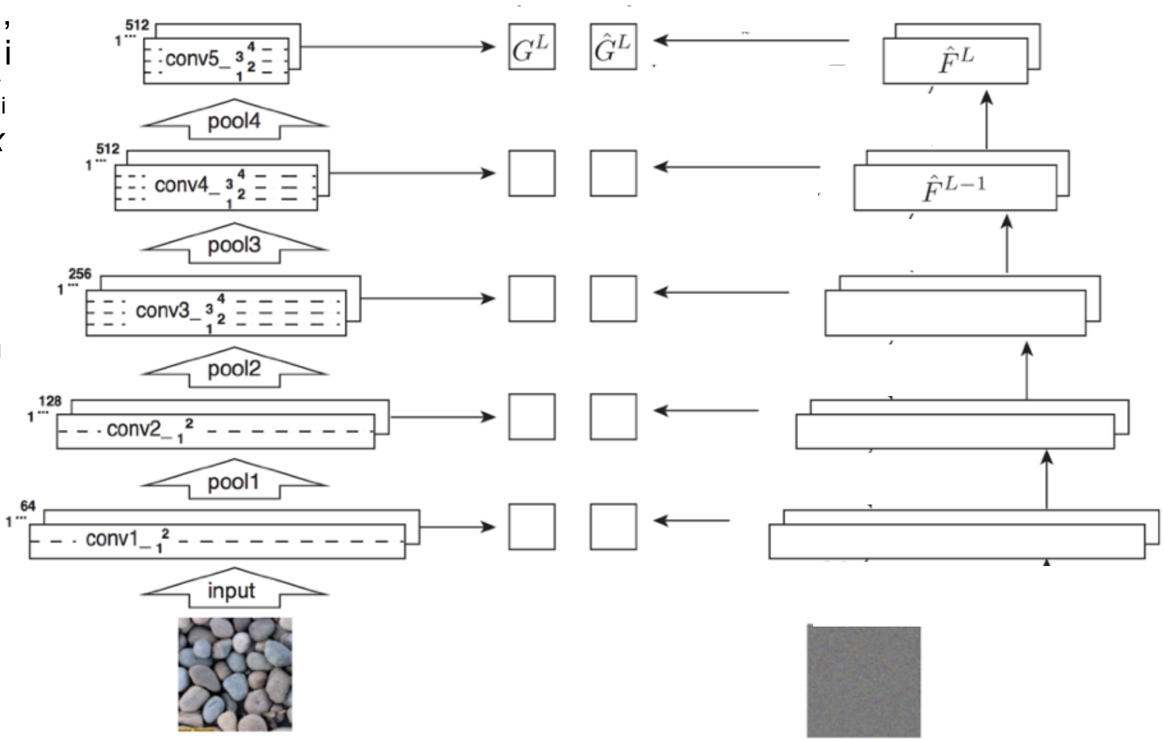

Gatys, Ecker, and Bethge, “Texture Synthesis Using Convolutional Neural Networks”, NIPS 2015

# 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:

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

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




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

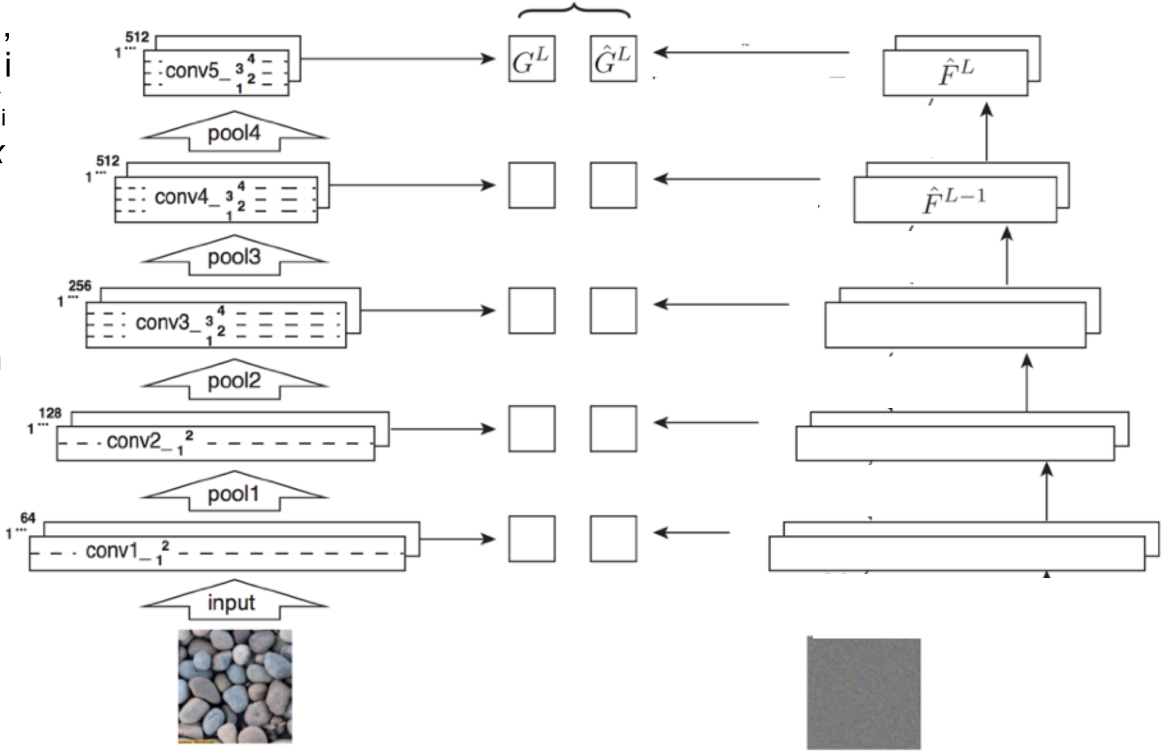
# Neural Texture Synthesis

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

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:

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

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




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

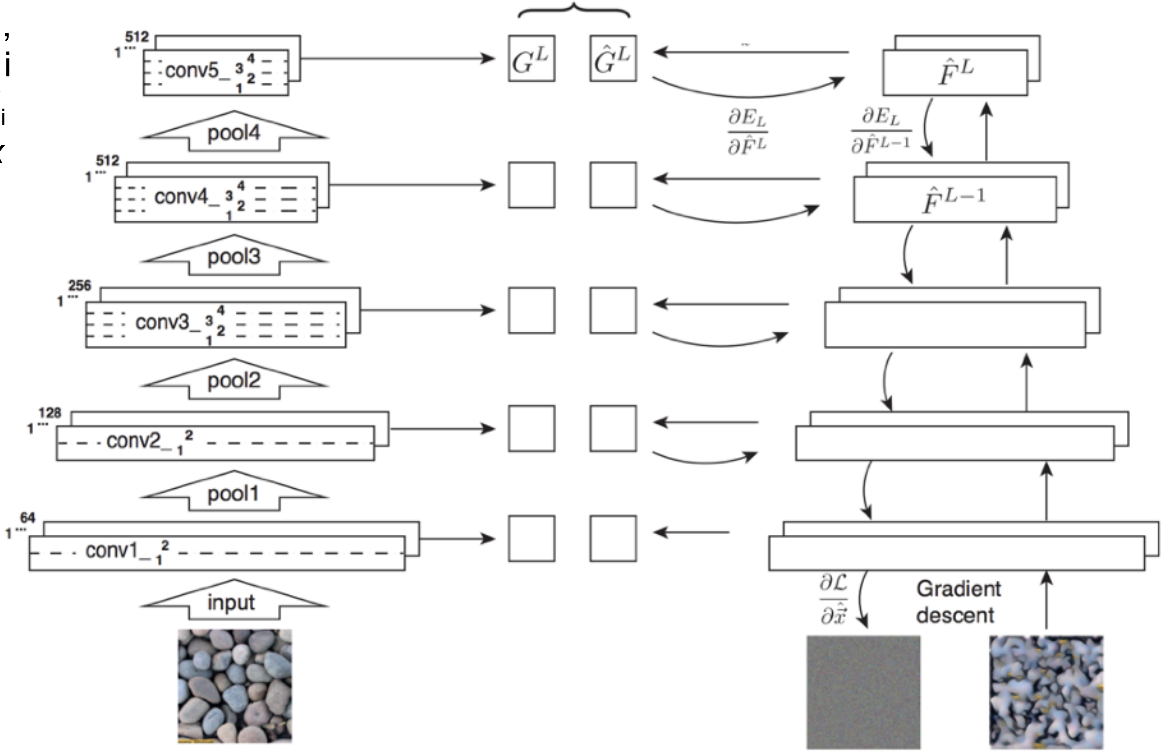
# Neural Texture Synthesis

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

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:

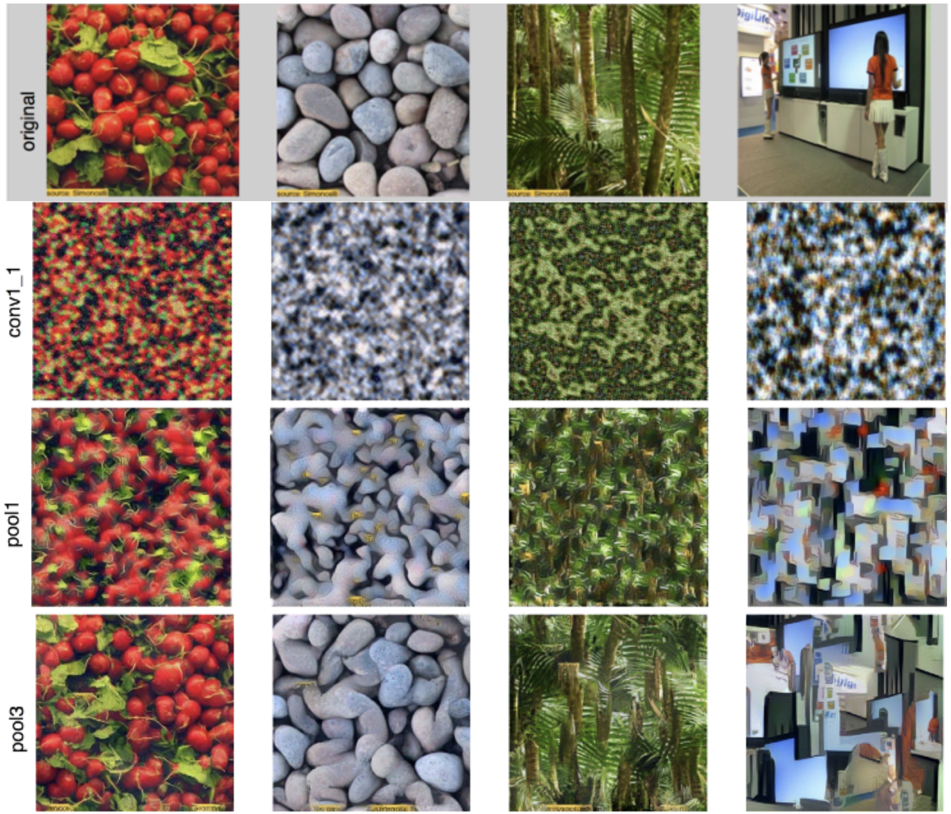
$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l \text{ (shape } C_i \times C_i\text{)}$$

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

# 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)

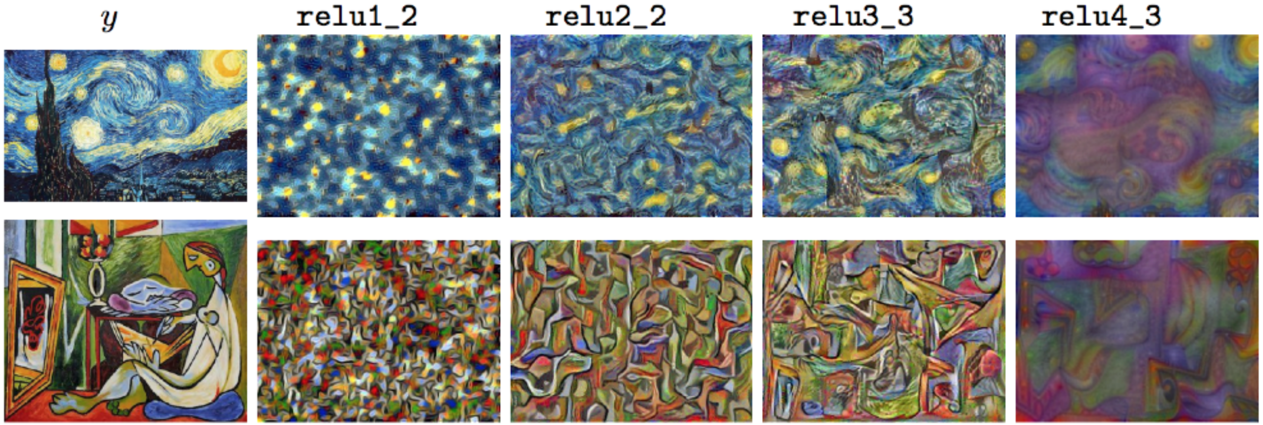
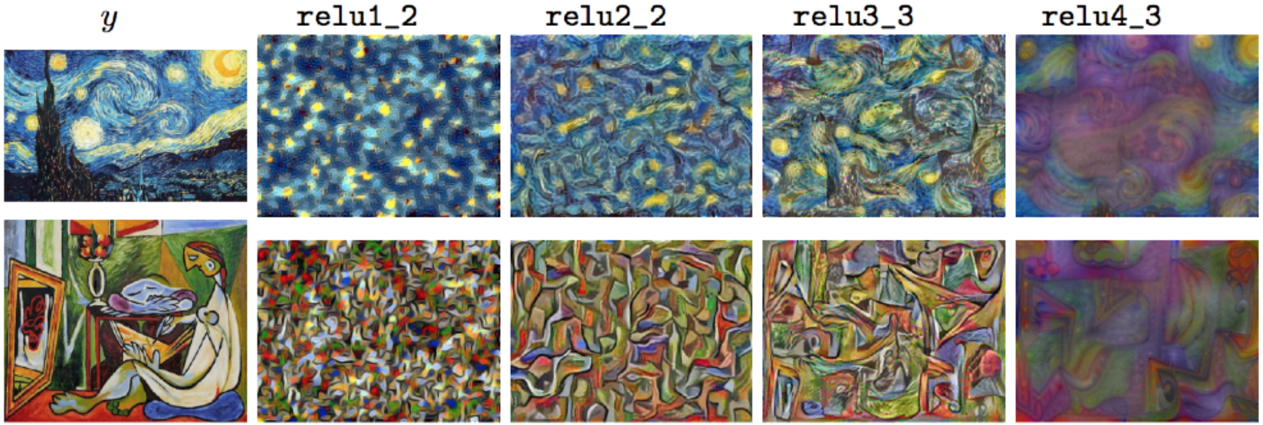



Figure from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016. Copyright Springer, 2016. Reproduced for educational purposes.

# Neural Style Transfer: Feature + Gram Reconstruction

Texture synthesis (Gram reconstruction)




Feature reconstruction

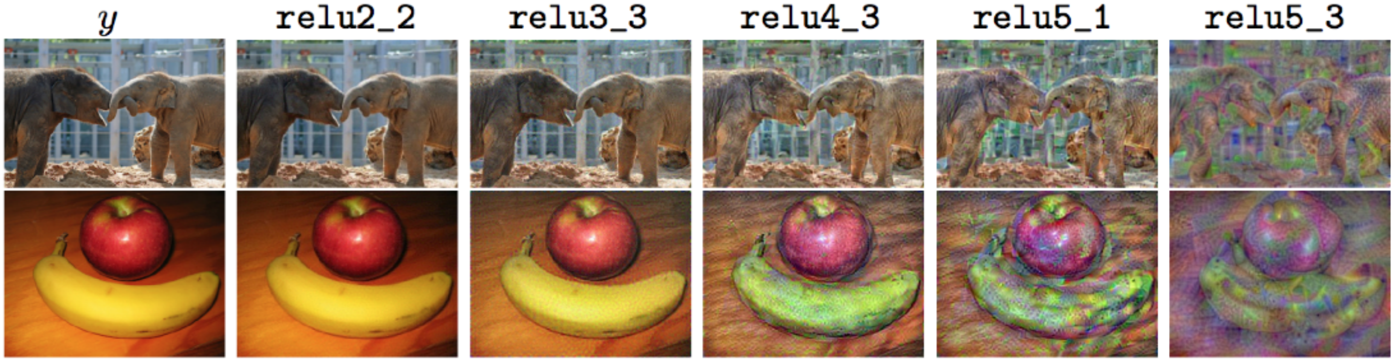



Figure from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016. Copyright Springer, 2016. Reproduced for educational purposes.

# Neural Style Transfer

Content Image



+

Style Image



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

# Neural Style Transfer

Content Image



+

Style Image



=

Style Transfer!

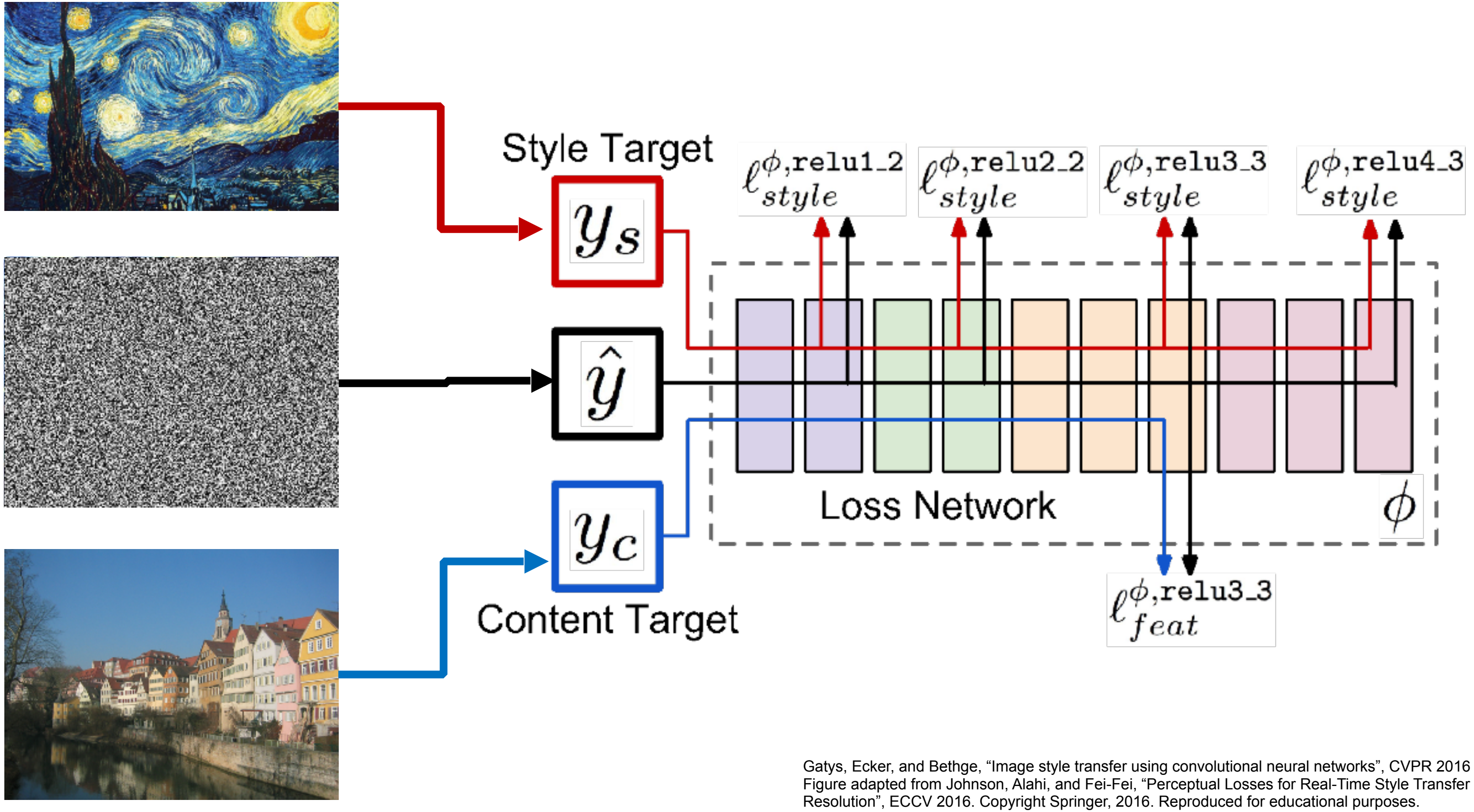


Gatys, Ecker, and Bethge, “Image style transfer using convolutional neural networks”, CVPR 2016

Style image

Output image (Start with noise)

Content image

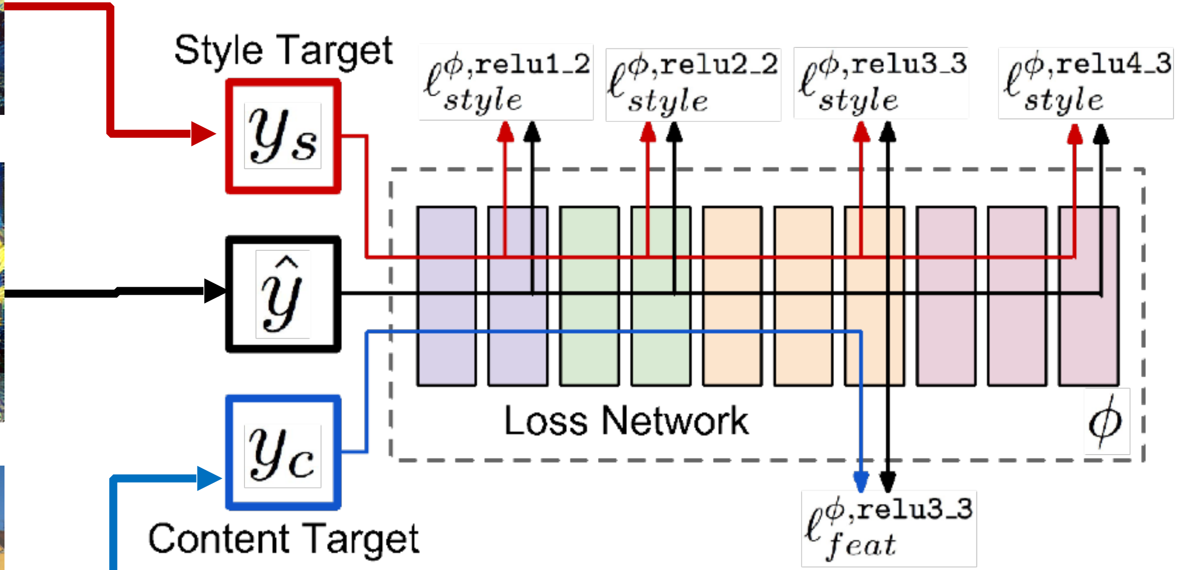



Gatys, Ecker, and Bethge, “Image style transfer using convolutional neural networks”, CVPR 2016
Figure adapted from Johnson, Alahi, and Fei-Fei, “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016. Copyright Springer, 2016. Reproduced for educational purposes.

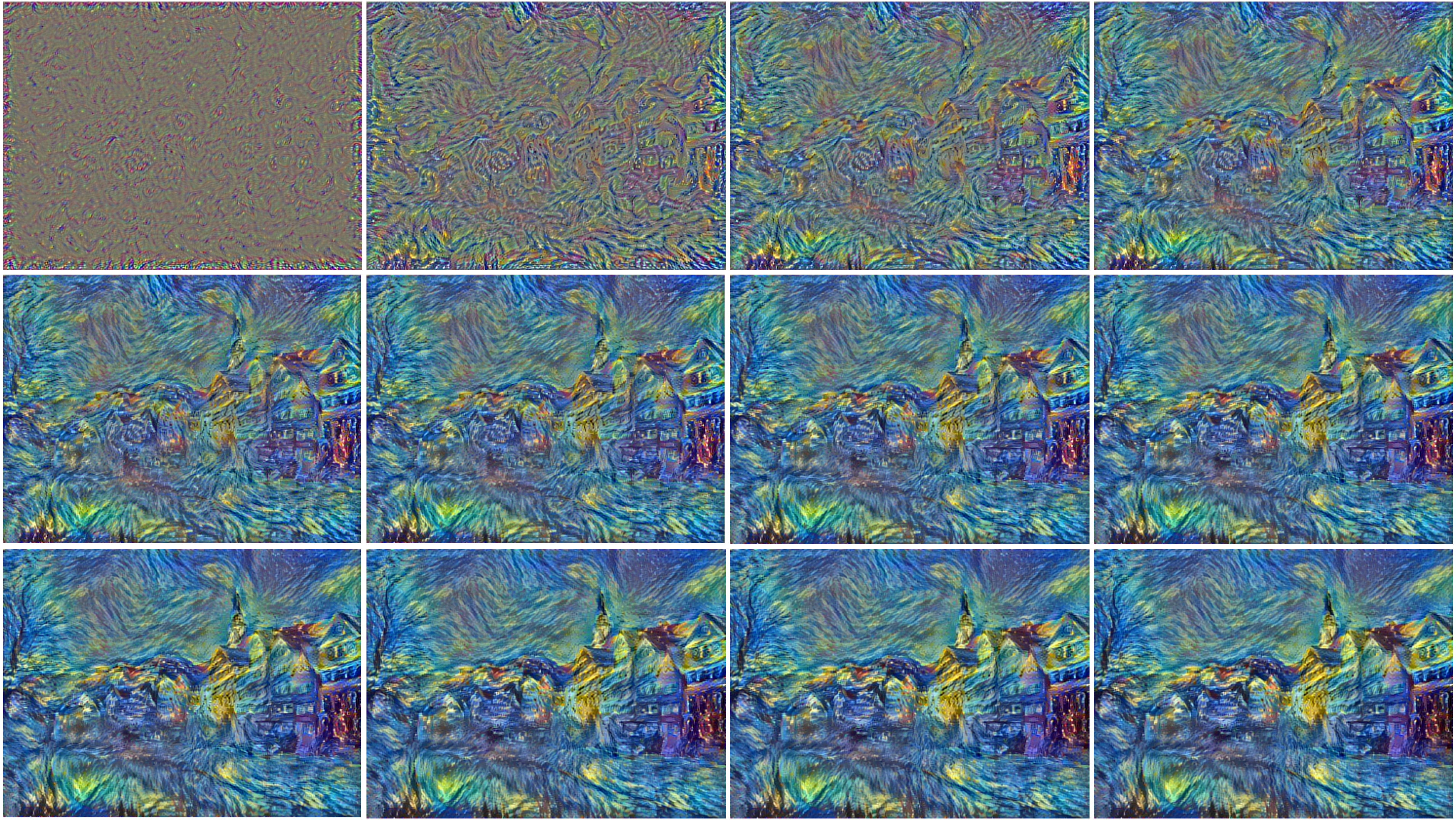
Style image

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. Copyright Springer, 2016. Reproduced for educational purposes.



# Neural Style Transfer

Example outputs from [implementation](implementation) (in Torch)

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

# Neural Style Transfer

More weight to content loss

More weight to style loss

# Neural Style Transfer

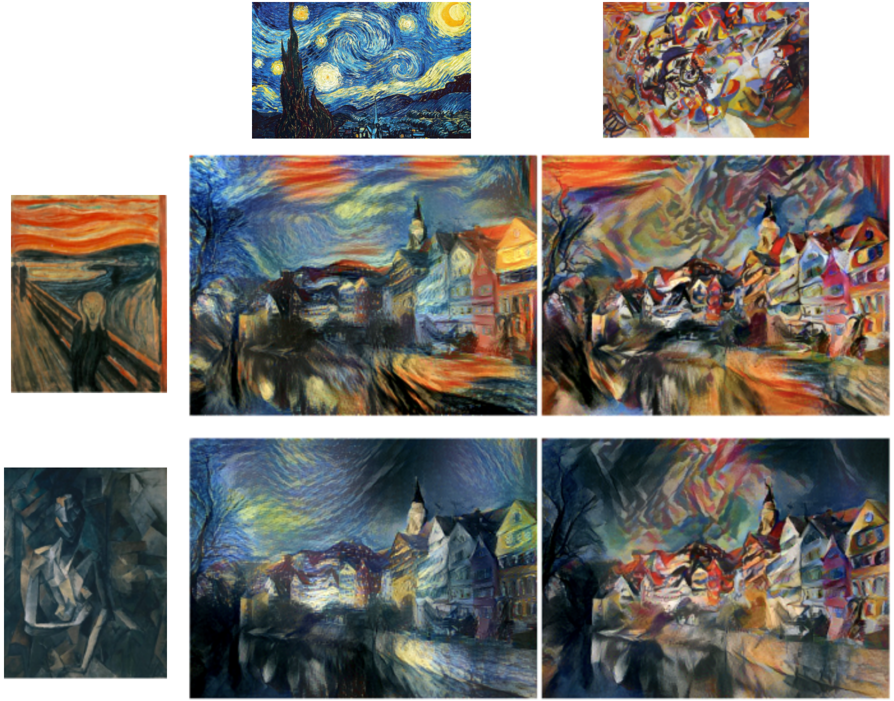
Resizing style image before running style transfer algorithm can transfer different types of features

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.

# How important are learned representations?

For synthesis even with randomly initialized neural networks produce good results

Experiment with Random filters vs. VGG-16 filters

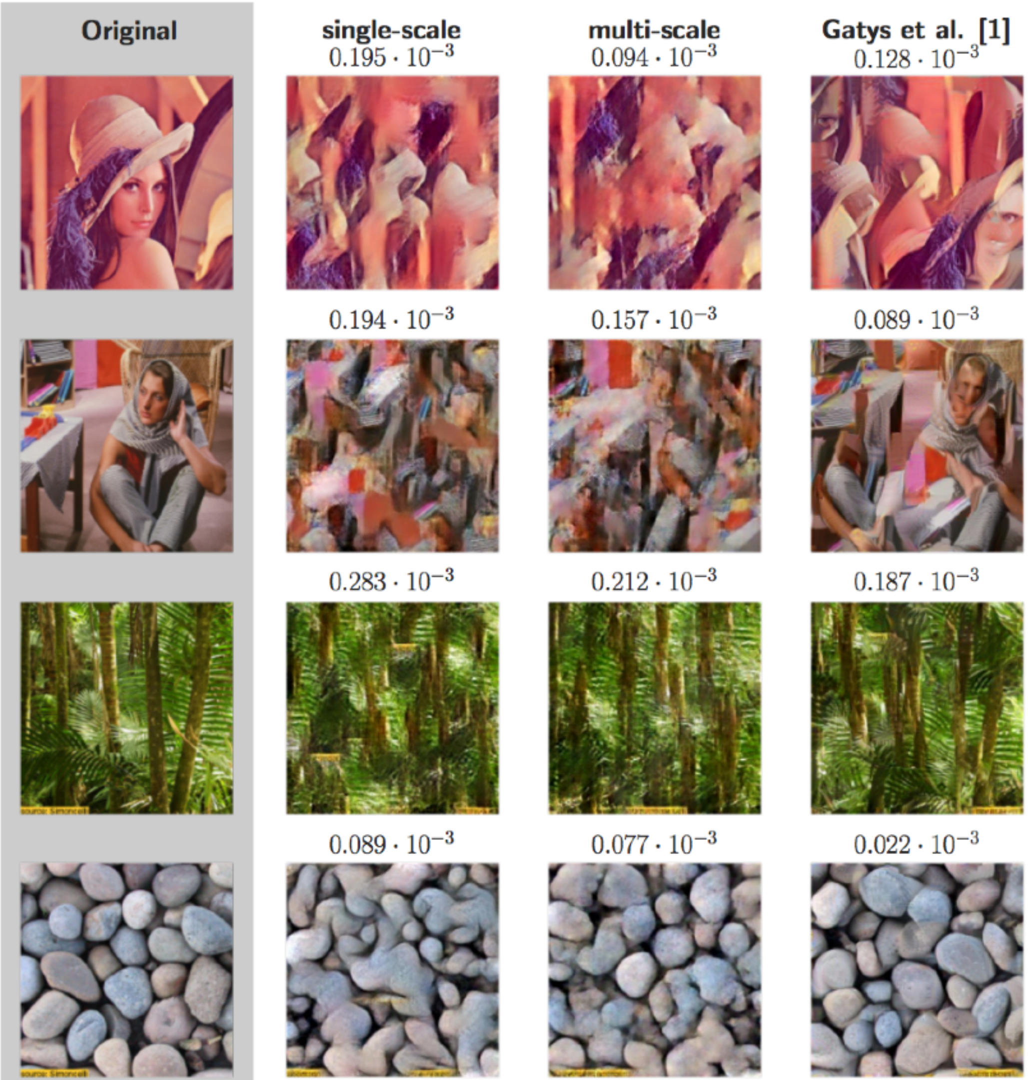
- Single-scale — 1 layer network with random filters [11x11] + ReLU
- Multi-scale — 1 layer network with random filters of different scales [3x3, 5x5, …, 128x128] + ReLU

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

# Summary

Many methods for understanding CNN representations

**Activations**: Nearest neighbors, Dimensionality reduction, maximal patches, occlusion
**Gradients**: Saliency maps, class visualization, feature inversion
**Fun**: DeepDream, Style Transfer.