


FACULTY OF MATHEMATICS,
PHYSICS AND INFORMATICS
Comenius University
Bratislava

Neural Networks for Computer Vision

Lecture 12: CNN Visualization

Ing. Viktor Kocur, PhD., RNDr. Zuzana Černeková, PhD.

8.12.2021

Contents

Acknowledgment

Most of the slides are directly adopted from slides for CS231n[1] course at Stanford University!

[1] Fei-Fei Li, Ranjay Krishna, and Danfei Xu. *Stanford CS231n lecture slides*. http://cs231n.stanford.edu/slides/

Visualizing Neural Networks

With deep learning we are able to obtain a neural network which represents a complex function to perform some tasks.

Visualizing Neural Networks

With deep learning we are able to obtain a neural network which represents a complex function to perform some tasks.

In this lecture we will cover some methods which help us visualize how the neural network works!

Filters in the initial layer

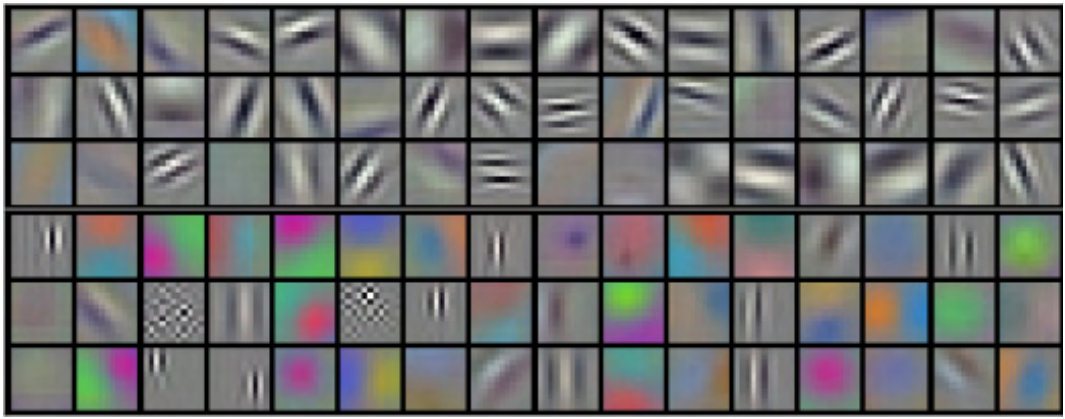


Image adopted from: Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks.” In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105

Nearest neighbors in feature space

Image adopted from: Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks.” In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105

t-SNE

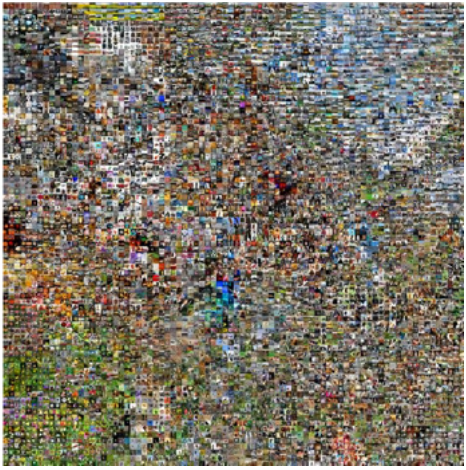
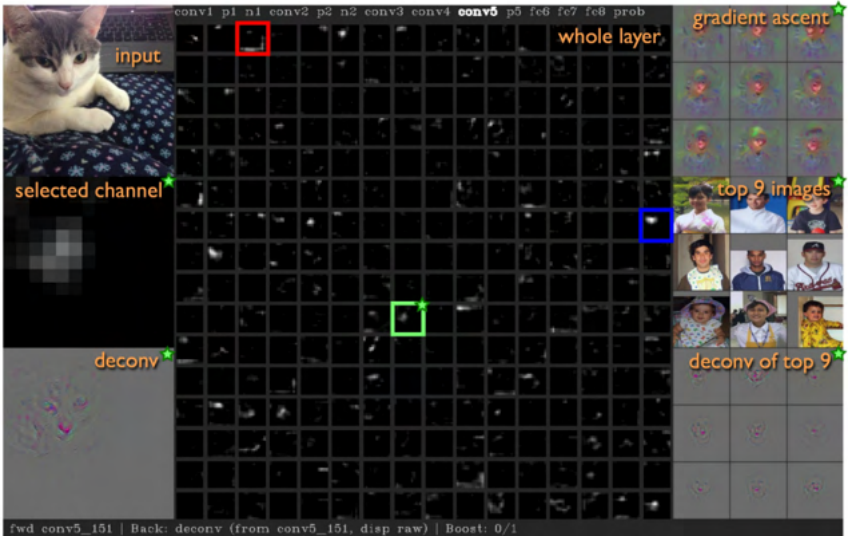


Image adopted from: Andrej Karpathy. *T-SNE visualization of CNN Codes - Stanford Computer Science*. URL: https://cs.stanford.edu/people/karpathy/cnnembed/

Understanding activations



Finding images that activate cetrain neurons

Normally we optimize the weights of layers to decrease a loss function. In a similar way we can optimize the image pixels to maximize the image.

$$I_f = \arg\max_I A_f(I) - R(I)$$

To produce nicer images we may choose to use a regularization term R such as L2 regularization.

Optimal images for some classes

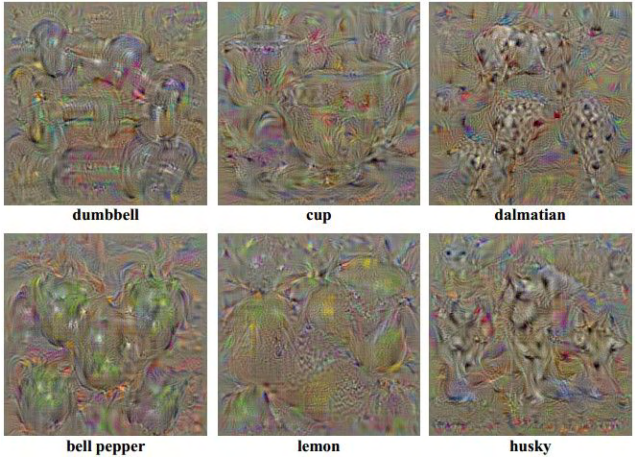


Image adopted from: Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep inside convolutional networks: Visualising image classification models and saliency maps.” In: *arXiv preprint arXiv:1312.6034* (2013)

Optimal images for some classes

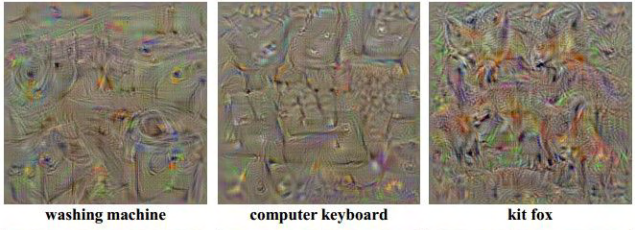


Image adopted from: Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep inside convolutional networks: Visualising image classification models and saliency maps.” In: *arXiv preprint arXiv:1312.6034* (2013)

Features in a network

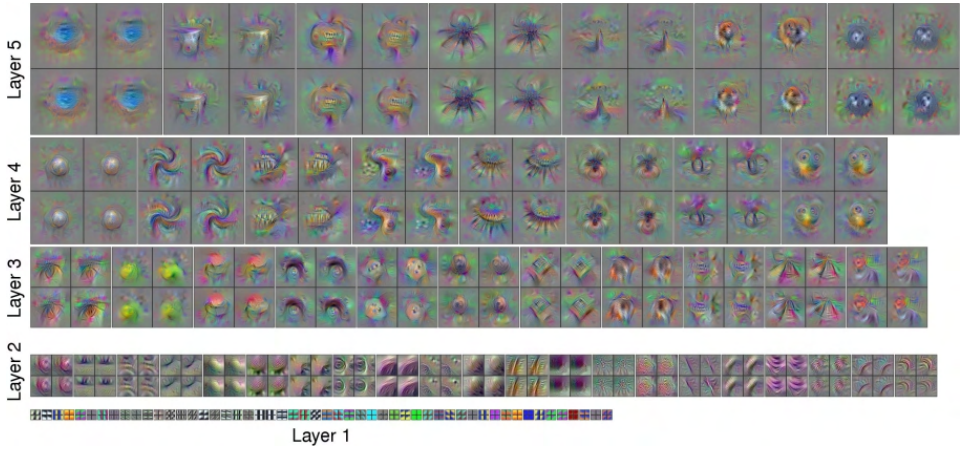


Image adopted from: Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep inside convolutional networks: Visualising image classification models and saliency maps.” In: *arXiv preprint arXiv:1312.6034* (2013)

Features in a network

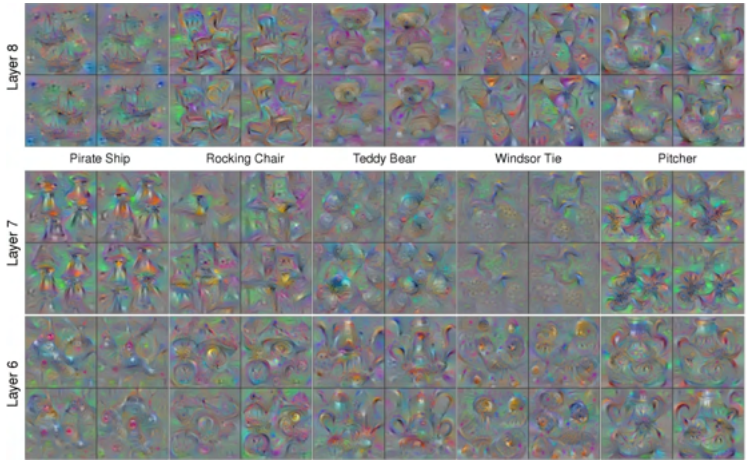


Image adopted from: Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep inside convolutional networks: Visualising image classification models and saliency maps.” In: *arXiv preprint arXiv:1312.6034* (2013)

Deconvolution

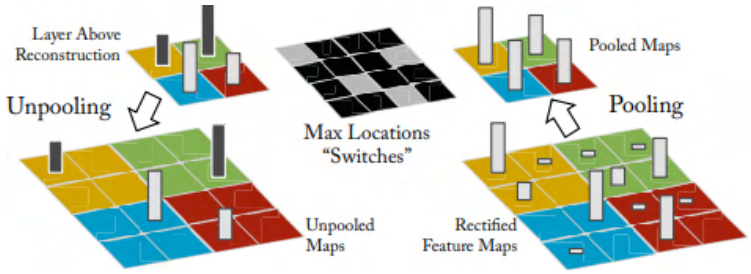


Image adopted from: Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks." In: *European conference on computer vision*. Springer. 2014, pp. 818–833

Activation maps using patches

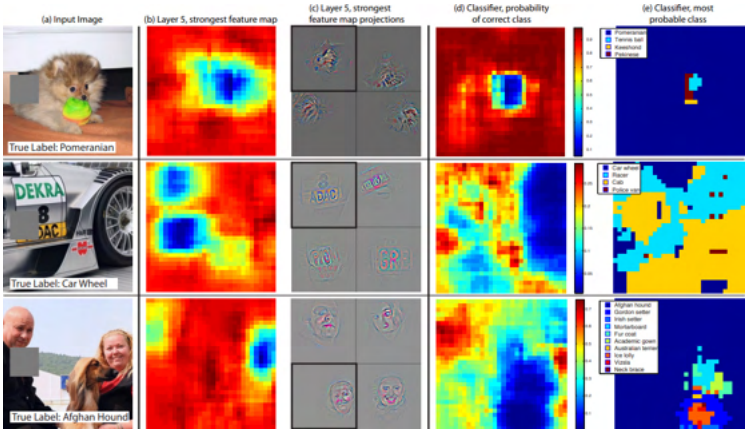


Image adopted from: Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks.” In: *European conference on computer vision*. Springer. 2014, pp. 818–833

Features and maximally activating patches

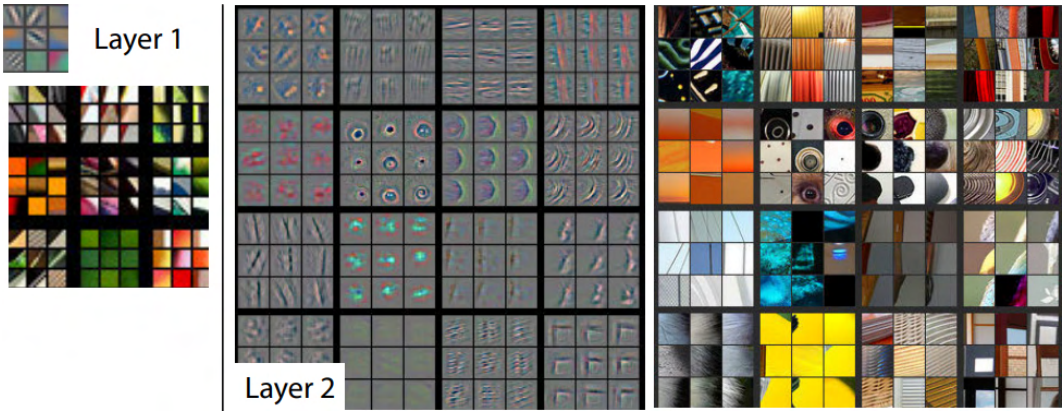


Image adopted from: Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks.” In: *European conference on computer vision*. Springer. 2014, pp. 818–833

Features and maximally activating patches

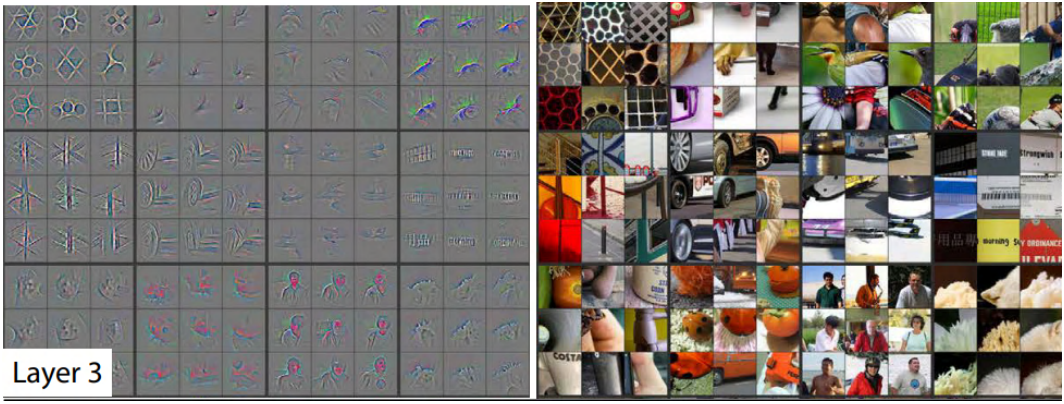


Image adopted from: Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks.” In: *European conference on computer vision*. Springer. 2014, pp. 818–833

Features and maximally activating patches

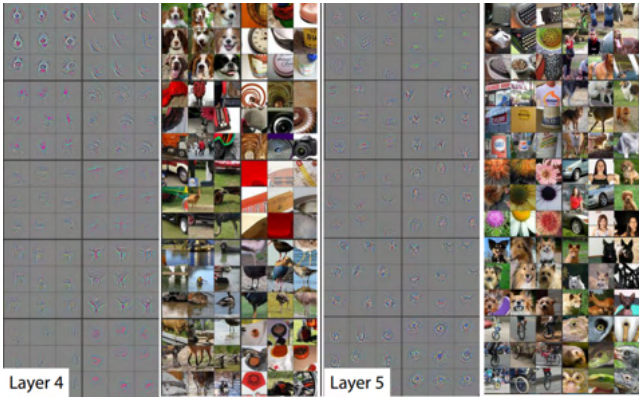


Image adopted from: Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks.” In: *European conference on computer vision*. Springer. 2014, pp. 818–833

Image crops

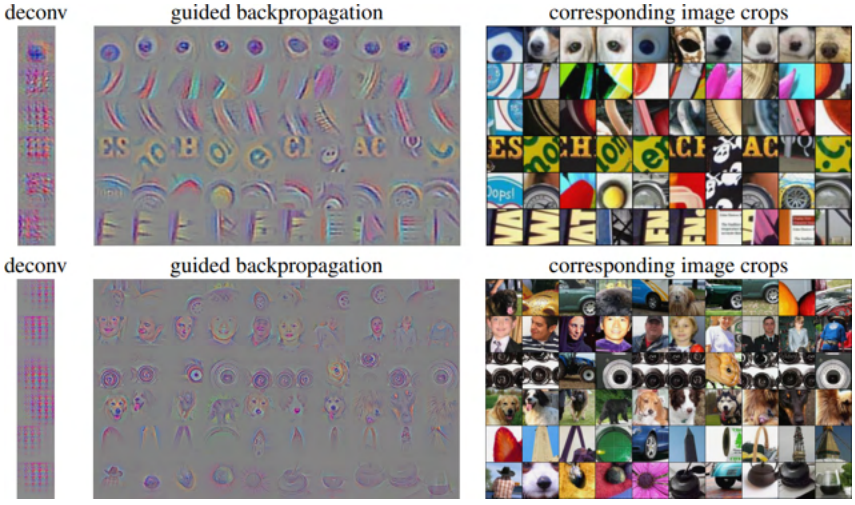


Image adopted from: Jost Tobias Springenberg et al. “Striving for simplicity: The all convolutional net.” In: *arXiv preprint arXiv:1412.6806* (2014)

Class activation maps

Image adopted from: Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep inside convolutional networks: Visualising image classification models and saliency maps.” In: *arXiv preprint arXiv:1312.6034* (2013)

Fooling a DNN is easy

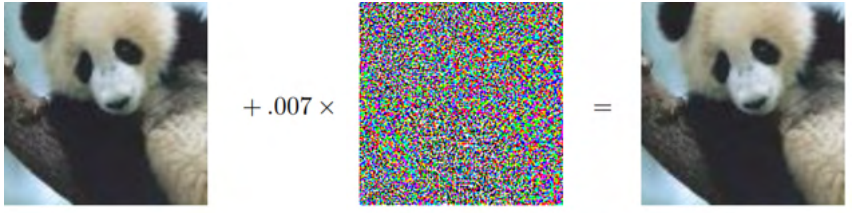


Image adopted from: Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." In: *arXiv preprint arXiv:1412.6572* (2014)

Adversarial sticker

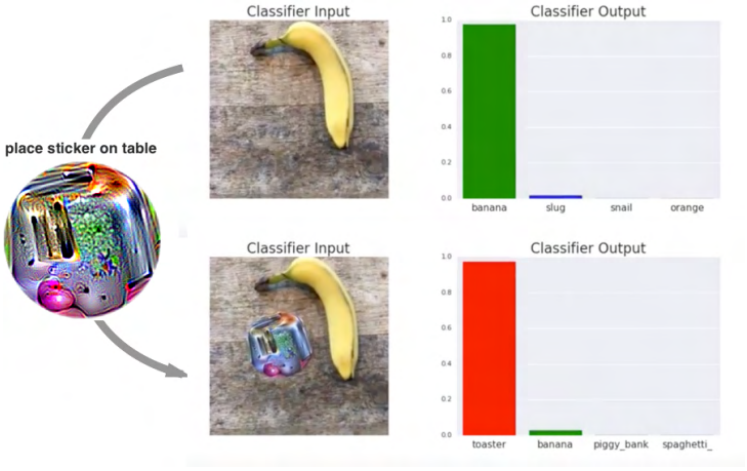


Image adopted from: Tom B Brown et al. “Adversarial patch.” In: *arXiv preprint arXiv:1712.09665* (2017)

Fooling an object detector

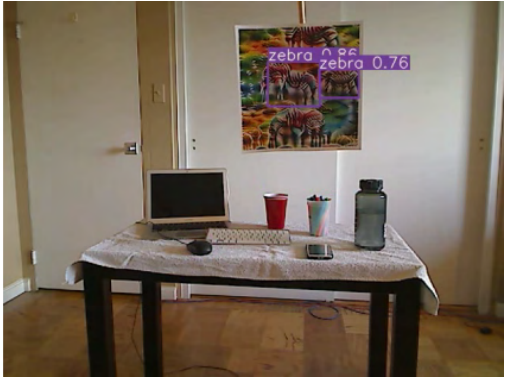


Image adopted from: Mark Lee and Zico Kolter. “On physical adversarial patches for object detection.” In: *arXiv preprint arXiv:1906.11897* (2019)

Universal attacks

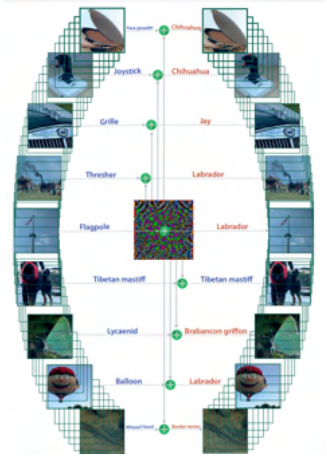
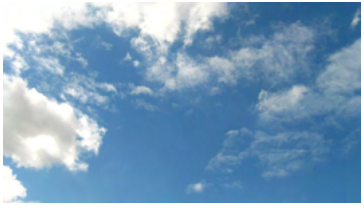
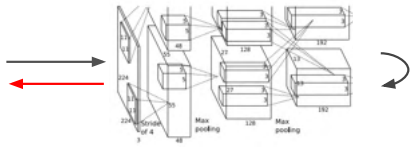


Image adopted from: Seyed-Mohsen Moosavi-Dezfooli et al. “Universal adversarial perturbations.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1765–1773

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





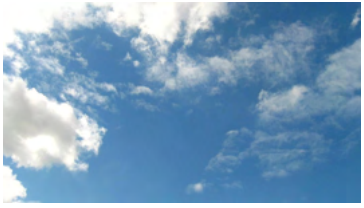
Choose an image and a layer in a CNN; repeat:

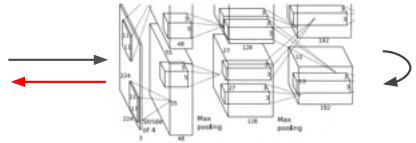
1. Forward: compute activations at chosen layer
2. Set gradient of chosen layer *equal to its activation*
3. Backward: Compute gradient on image
4. Update image

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

DeepDream: Amplify existing features

```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)
```

Code is very simple but it uses a couple tricks:

DeepDream: Amplify existing features

```
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)
```

Code is very simple but it uses a couple tricks:

Jitter image

DeepDream: Amplify existing features

```
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)
```

Code is very simple but it uses a couple tricks:

(Code is licensed under Apache 2.0)

Jitter image

L1 Normalize gradients

DeepDream: Amplify existing features

```
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)
```

Code is very simple but it uses a couple tricks:

(Code is licensed under Apache 2.0)

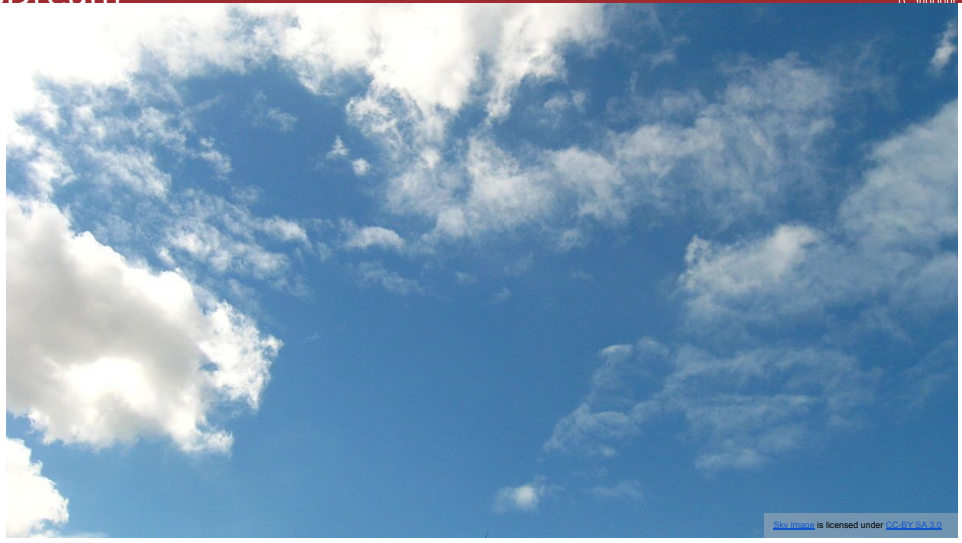
Jitter image

L1 Normalize gradients

Clip pixel values

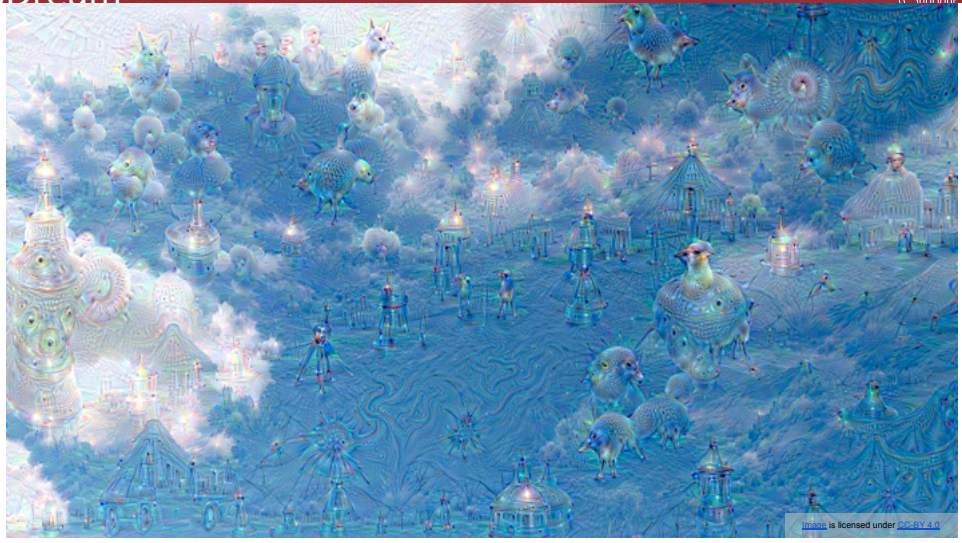
Also uses multiscale processing for a fractal effect (not shown)

DeepDream

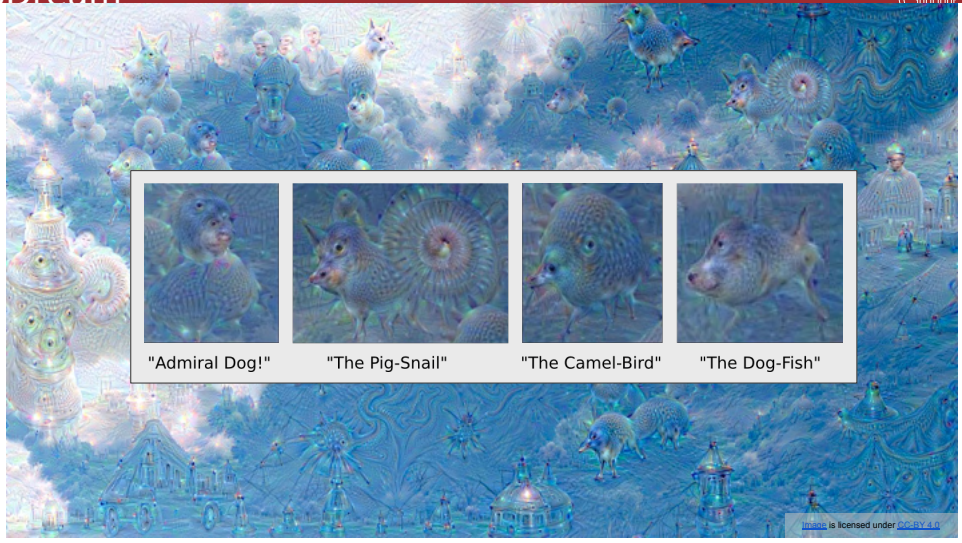


DeepDream

DeepDream



DeepDream


"Admiral Dog!"
"The Pig-Snail"
"The Camel-Bird"
"The Dog-Fish"
Image is licensed under CC-BY 4.0

DeepDream

DeepDream

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

Features of new image

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

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

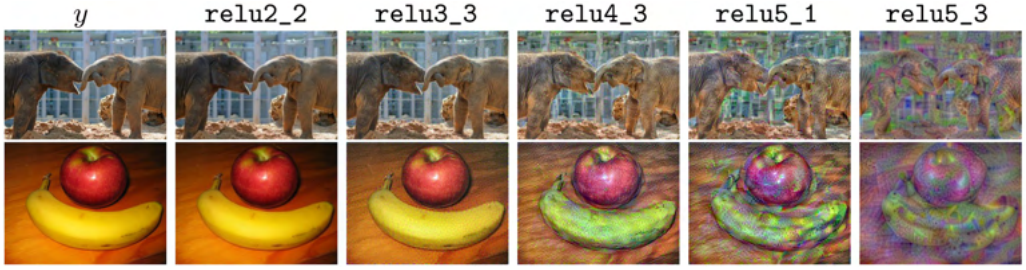
Total Variation regularizer
(encourages spatial smoothness)

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

Aravindh Mahendran and Andrea Vedaldi. “Understanding deep image representations by inverting them.” In: *Proceedings of the IEEE*

Feature Inversion

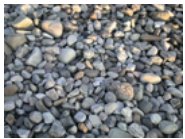
Reconstructing from different layers of VGG-16



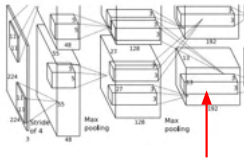
Mahendran and Vedaldi, "Understanding Deep Image Representations by Inverting Them", CVPR 2015
Figure from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016. Copyright Springer, 2016.
Reproduced for educational purposes.

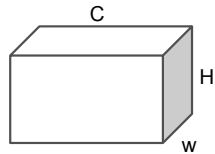
Aravindh Mahendran and Andrea Vedaldi. "Understanding deep image representations by inverting them." In: *Proceedings of the IEEE*

Neural Texture Synthesis: Gram Matrix



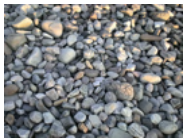
This image is in the public domain.



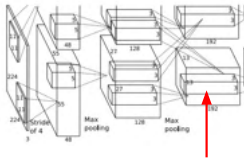


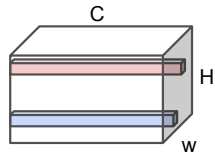
Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

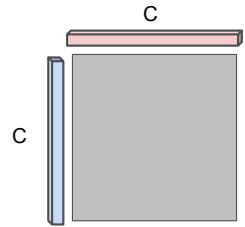
Neural Texture Synthesis: Gram Matrix



This image is in the public domain.



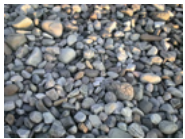




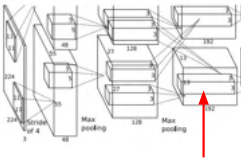
Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

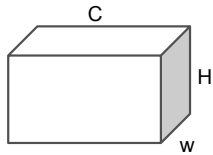
Outer product of two C-dimensional vectors gives C x C matrix measuring co-occurrence

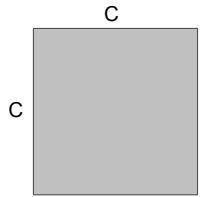
Neural Texture Synthesis: Gram Matrix



This image is in the public domain.







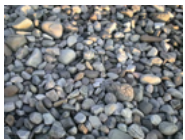
Gram Matrix

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

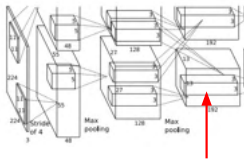
Outer product of two C-dimensional vectors gives C x C matrix measuring co-occurrence

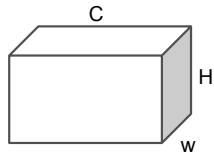
Average over all HW pairs of vectors, giving **Gram matrix** of shape C x C

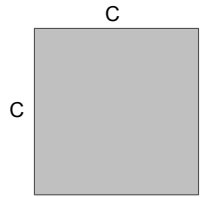
Neural Texture Synthesis: Gram Matrix



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 two C-dimensional vectors gives C x C matrix measuring co-occurrence

Average over all HW pairs of vectors, giving **Gram matrix** of shape C x C

Efficient to compute; reshape features from

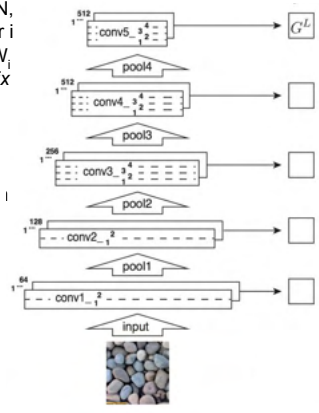
C x H x W to =C x HW

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:

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



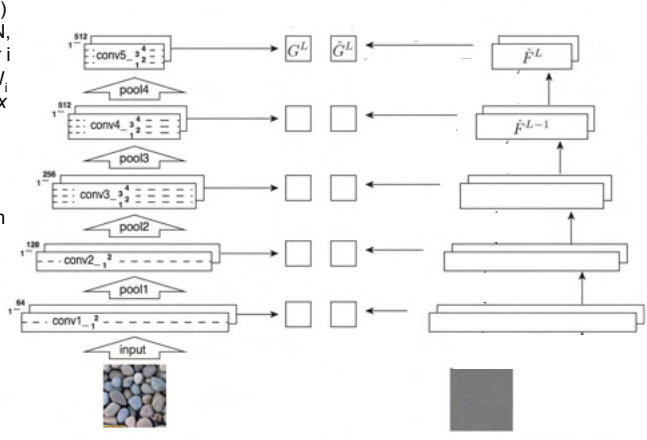
Gatys, Ecker, and Bethge, "Texture Synthesis Using Convolutional Neural Networks", NIPS 2015
Figure copyright Leon Gatys, Alexander S. Ecker, and Matthias Bethge, 2015. Reproduced with permission.

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:

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

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



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.

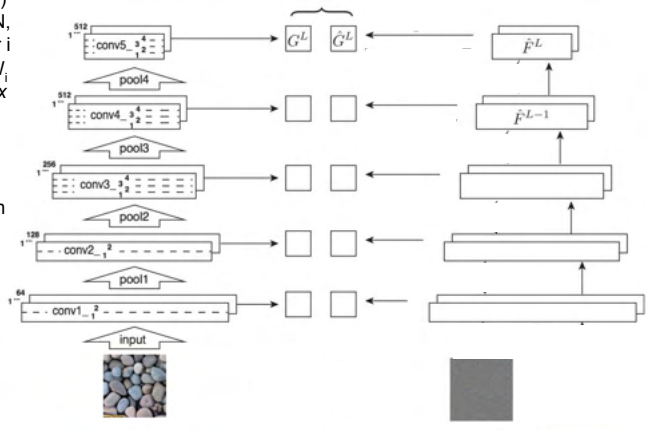
Neural Texture Synthesis

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

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.

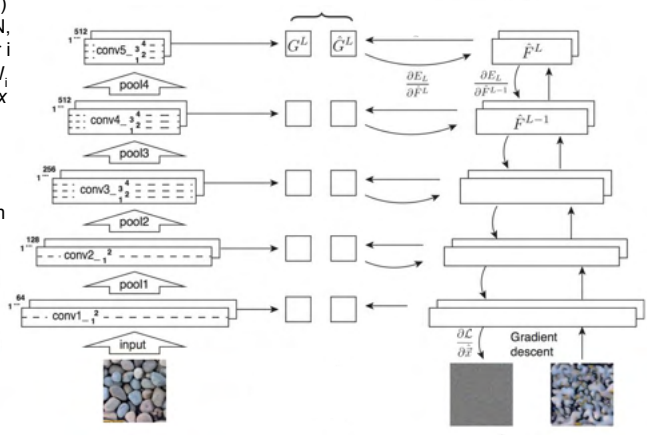
Neural Texture Synthesis

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

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:

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

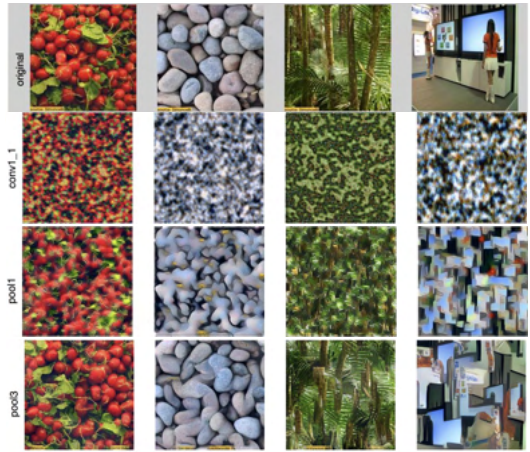
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5



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)

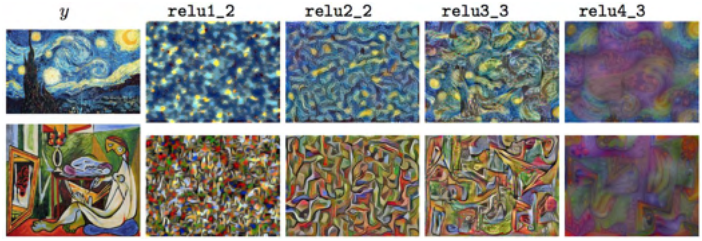
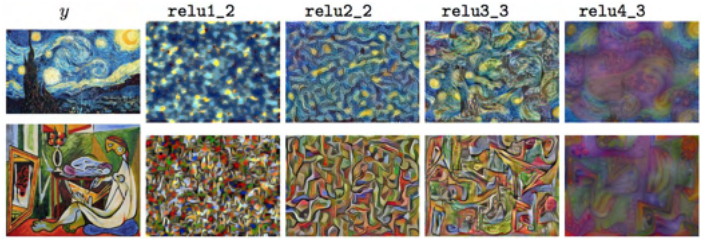


Figure from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016. Copyright Springer, 2016. Reproduced for educational purposes.

Neural Style Transfer: Feature + Gram Reconstruction

Texture synthesis (Gram reconstruction)



Feature reconstruction

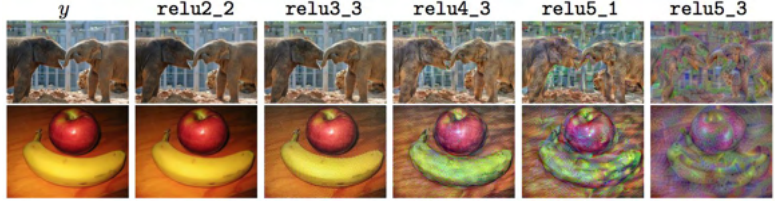


Figure from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016. Copyright Springer, 2016. Reproduced for educational purposes.

Neural Style Transfer

Content Image

+

Style Image

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

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. "Image style transfer using convolutional neural networks." In *Proceedings of the IEEE*

Neural Style Transfer

Content Image

This image is licensed under CC-BY 3.0

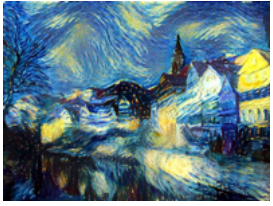
+

Style Image

Starry Night by Van Gogh is in the public domain

=

Style Transfer!

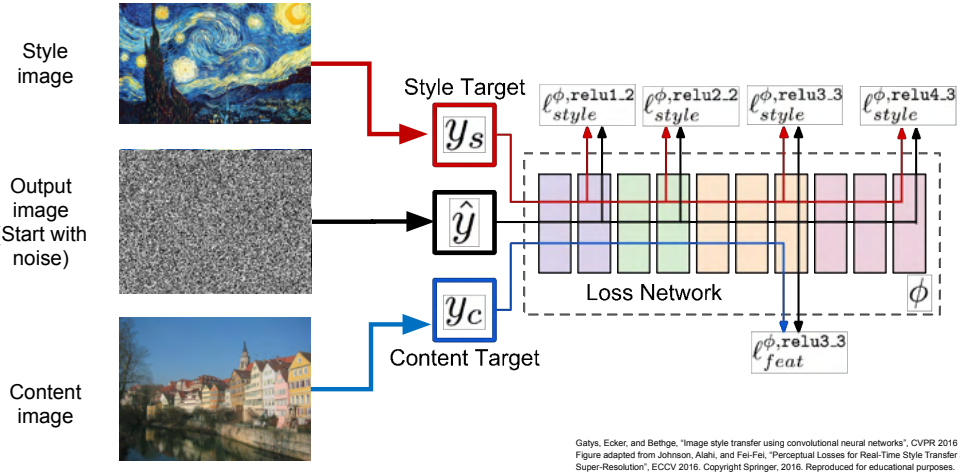


This image copyright Justin Johnson, 2015. Reproduced with permission.

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

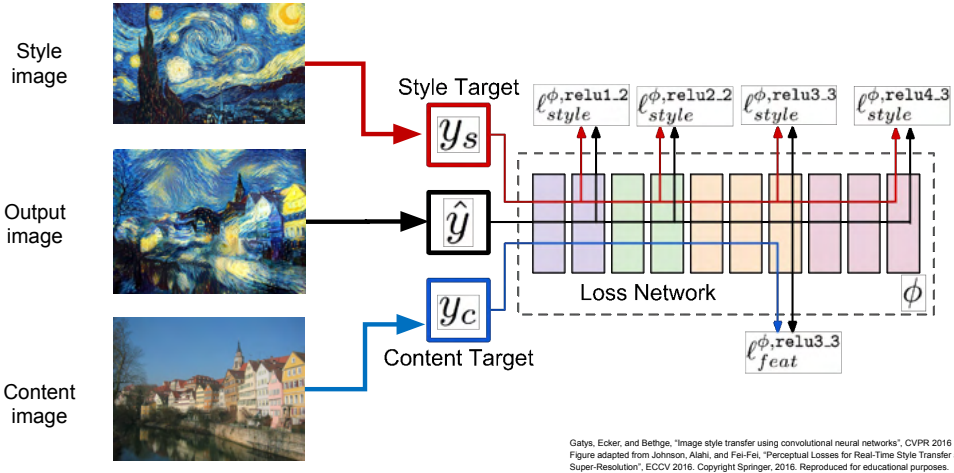
Leon A Gatys, Alexander S Ecker, and Matthias Bethge. "Image style transfer using convolutional neural networks." In *Proceedings of the IEEE*

Neural style transfer



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

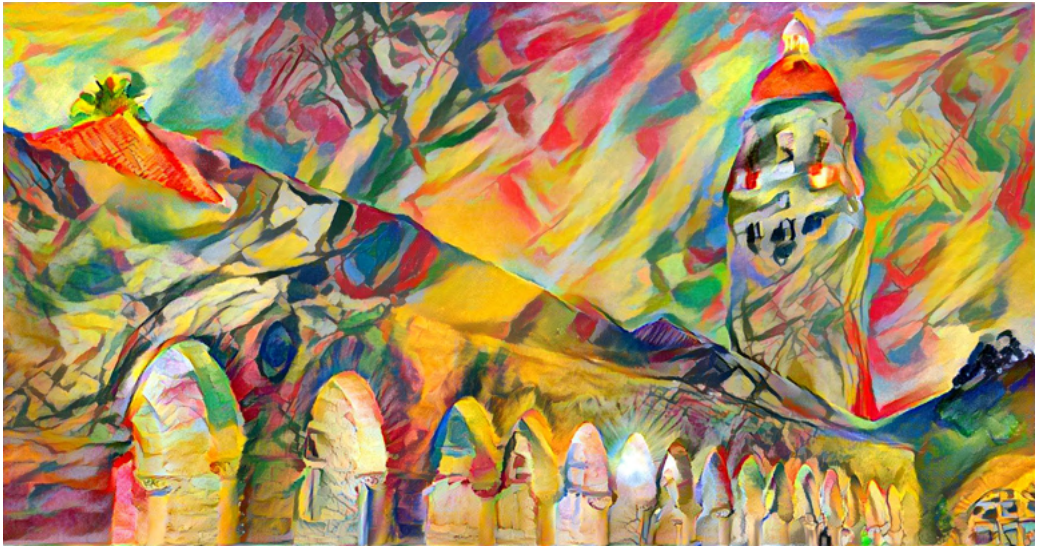


Gatys, Ecker, and Bethge, "Image style transfer using convolutional neural networks", CVPR 2016
Figure adapted from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016. Copyright Springer, 2016. Reproduced for educational purposes.

Neural style transfer

Neural style transfer

Neural style transfer



Neural style transfer