

# Lecture 12: Visualizing and Understanding

# Administrative

Milestones due tonight on Canvas, 11:59pm

Midterm grades released on Gradescope this week

A3 due next Friday, 5/26

HyperQuest deadline extended to Sunday 5/21, 11:59pm

Poster session is June 6

# Last Time: Lots of Computer Vision Tasks

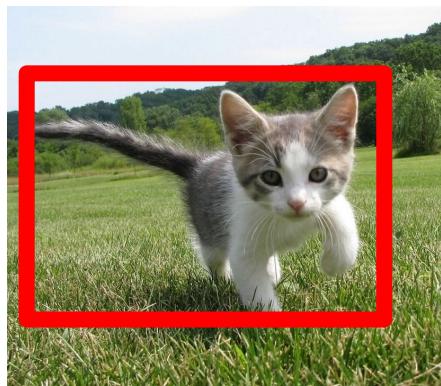
## Semantic Segmentation



GRASS, CAT,  
TREE, SKY

No objects, just pixels

## Classification + Localization



CAT

Single Object

主要区别还是事先  
不知道图像中包含  
的种类数

## Detection



DOG, DOG, CAT

Multiple Object

图像分割+目标检测

## Instance Segmentation

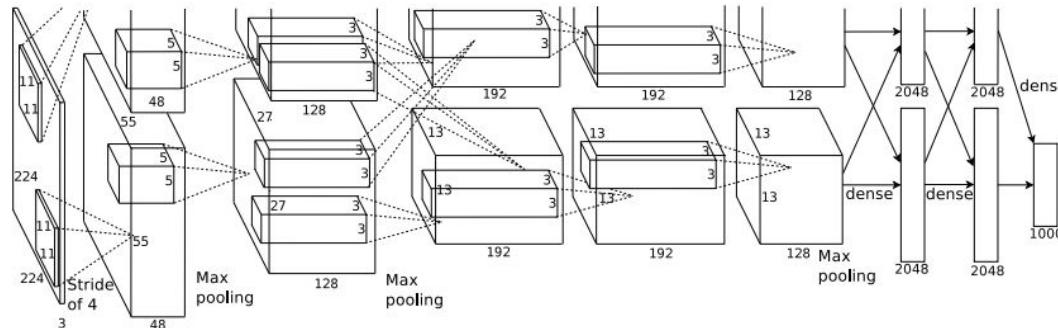


DOG, DOG, CAT

This image is CC0 public domain

# What's going on inside ConvNets?

This image is CC0 public domain



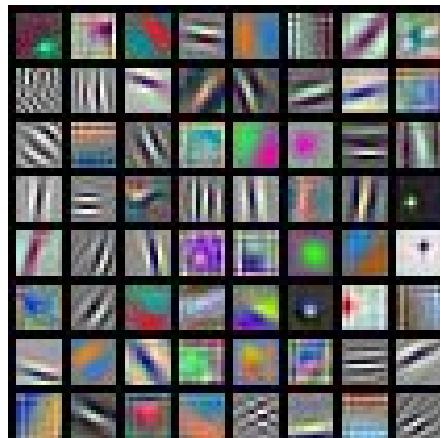
Input Image:  
3 x 224 x 224

What are the intermediate features looking for?

Class Scores:  
1000 numbers

Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012.  
Figure reproduced with permission. [▲](#)

# First Layer: Visualize Filters



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



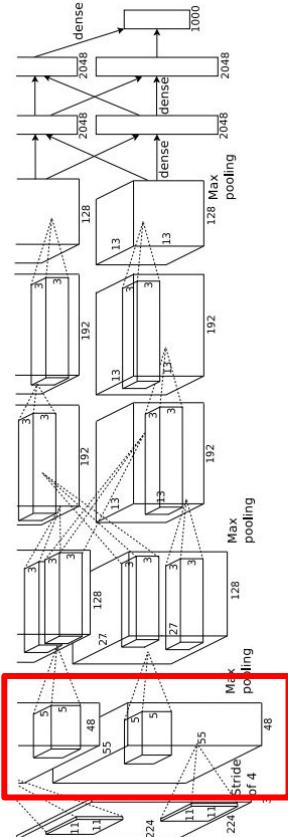
ResNet-18:  
 $64 \times 3 \times 7 \times 7$



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



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



卷积核类似于模板，就是你要寻找的东西，当input与你的模板相同时，此时内积最大

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

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

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

# Visualize the filters/kernels (raw weights)

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

(these are taken from ConvNetJS  
CIFAR-10 demo)

Weights:  


layer 1 weights

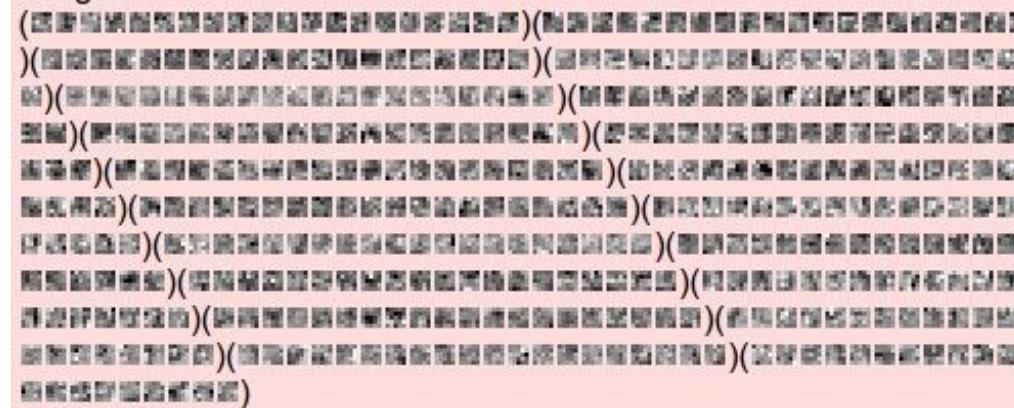
$16 \times 3 \times 7 \times 7$

Weights:  


layer 2 weights

$20 \times 16 \times 7 \times 7$

以 7x7 的灰度图显示

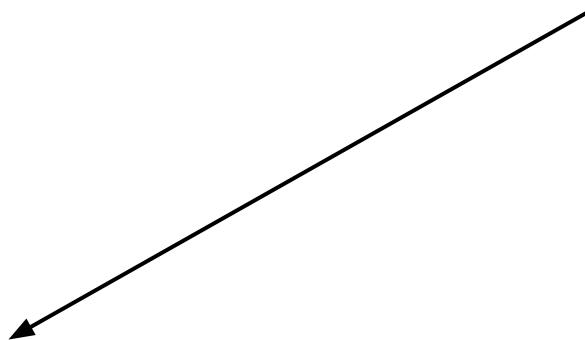
Weights:  


layer 3 weights

$20 \times 20 \times 7 \times 7$

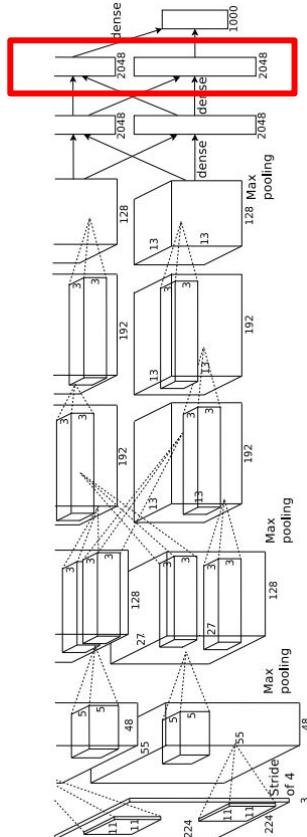
# Last Layer

FC7 layer



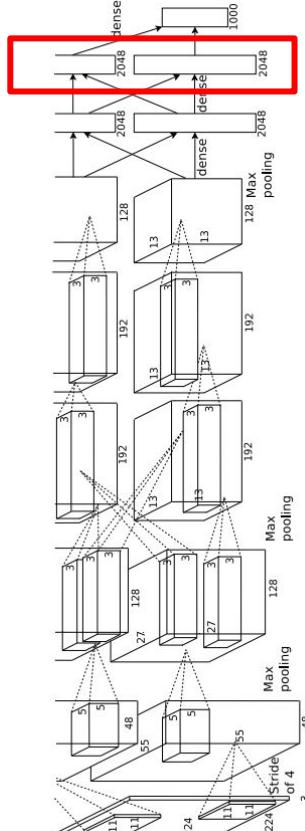
4096-dimensional feature vector for an image  
(layer immediately before the classifier)

Run the network on many images, collect the  
feature vectors



# Last Layer: Nearest Neighbors

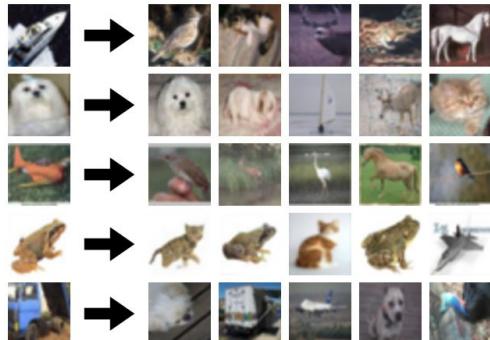
4096-dim vector



Test image L2 Nearest neighbors in feature space



**Recall:** Nearest neighbors  
in pixel space



Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012.

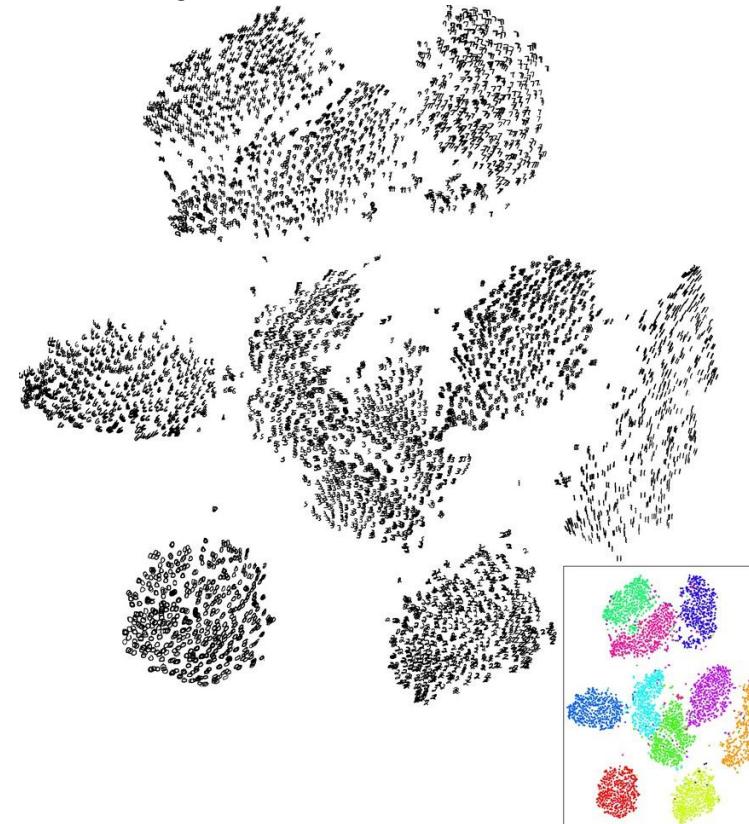
Figures reproduced with permission.

# Last Layer: Dimensionality Reduction

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

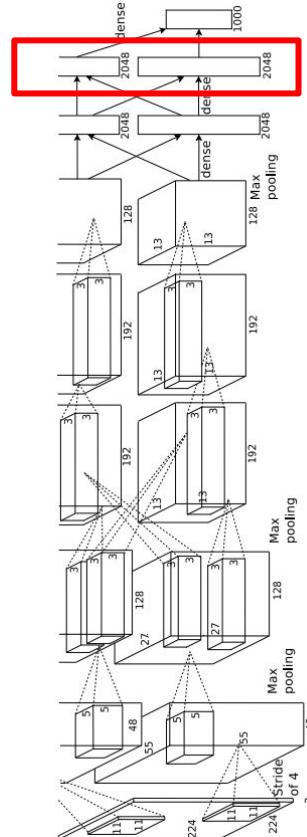
Simple algorithm: Principle  
Component Analysis (PCA)

More complex: **t-SNE**

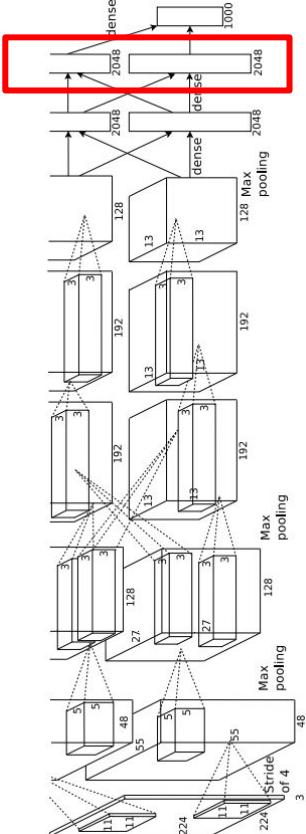
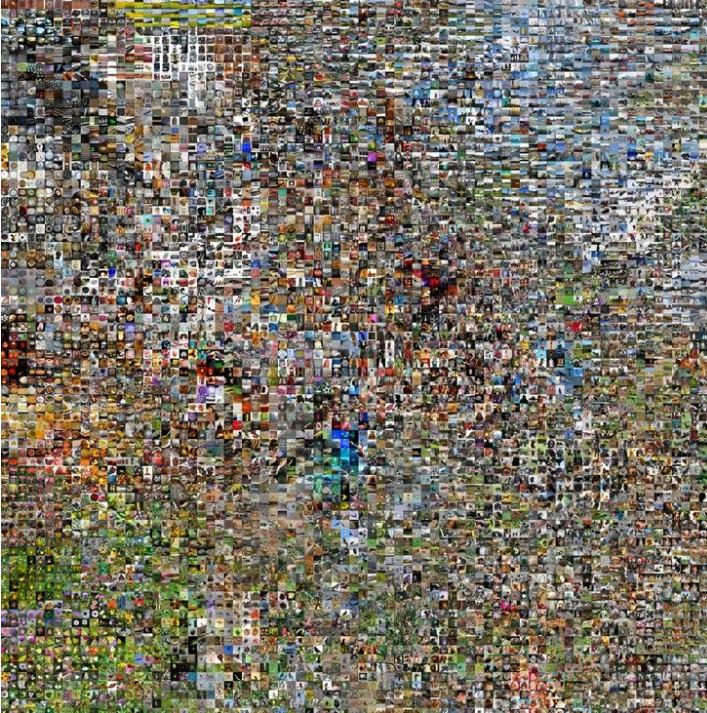
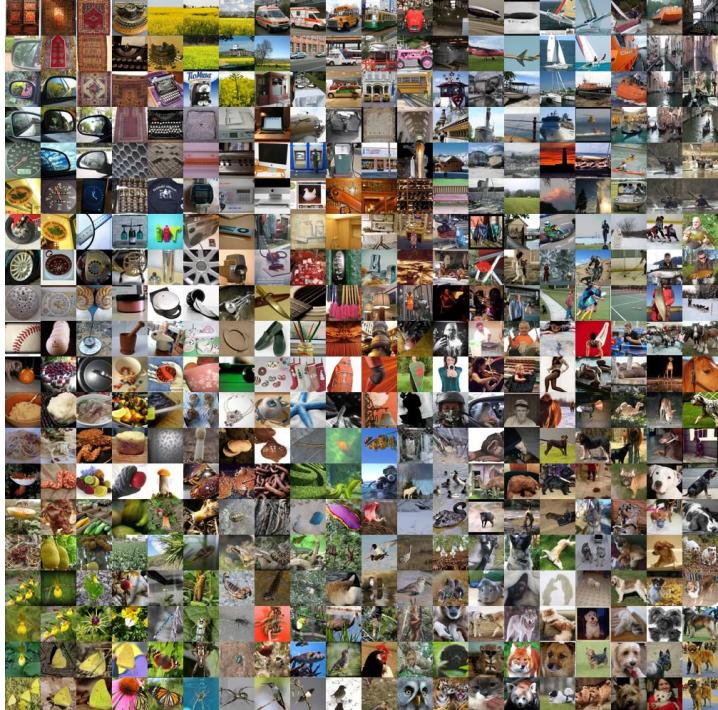


然后把原始图像放置到对应的二维坐标上

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



# Last Layer: Dimensionality Reduction



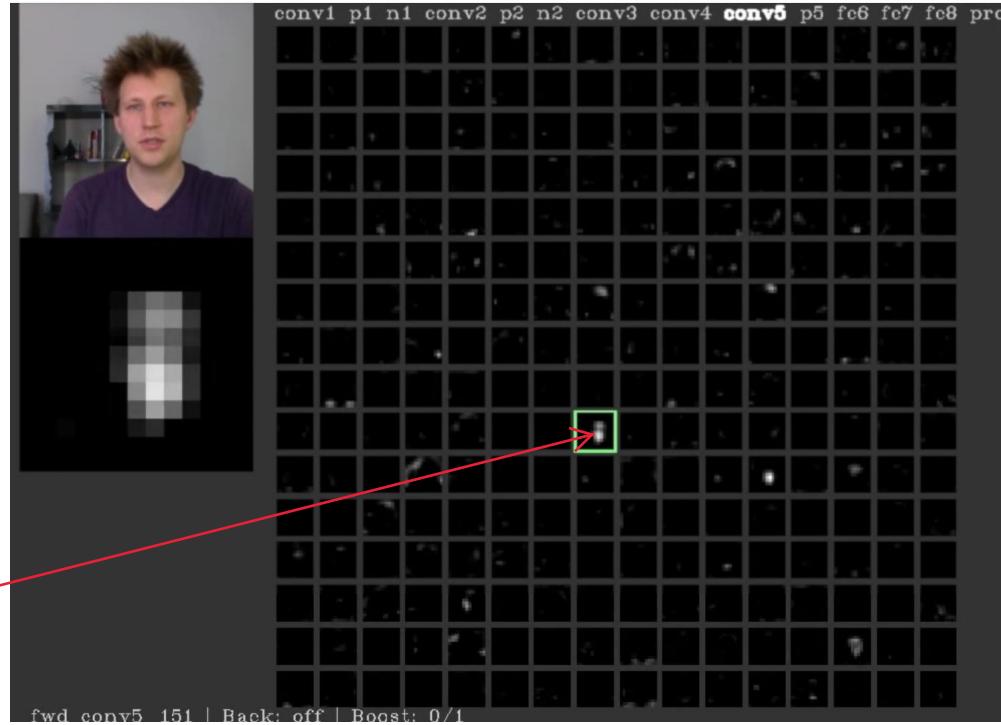
Van der Maaten and Hinton, "Visualizing Data using t-SNE", JMLR 2008  
Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012.  
Figure reproduced with permission.

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

# Visualizing Activations

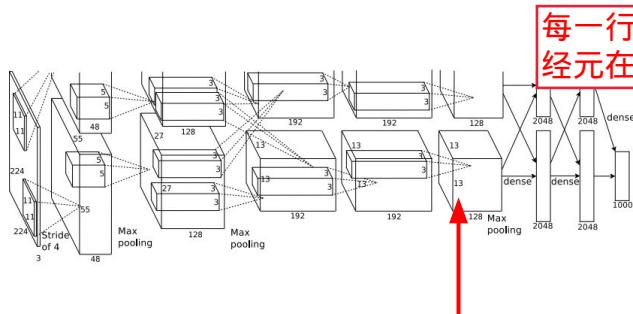
conv5 feature map is  
128x13x13; visualize  
as 128 13x13  
grayscale images

这个比较突出，貌似在被激活，看看他在找什么？



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

# Maximally Activating Patches



每一行代表每个神经元在寻找的特征

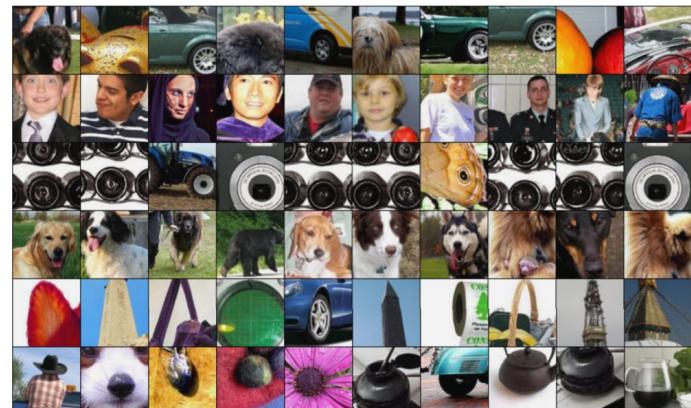


Pick a layer and a channel; e.g. conv5 is  $128 \times 13 \times 13$ , pick channel 17/128

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

Visualize image patches that correspond  
to maximal activations

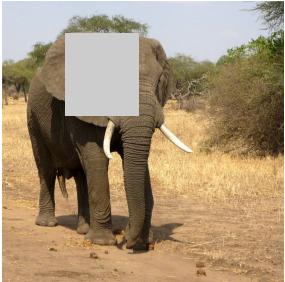
在庞大的数据集中，可视化特定层的特定特征  
的最大激活的图像块



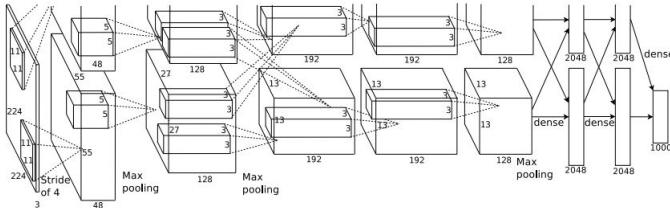
Springenberg et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015  
Figure copyright Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, 2015;  
reproduced with permission.

# Occlusion Experiments

Mask part of the image before feeding to CNN, draw heatmap of probability at each mask location

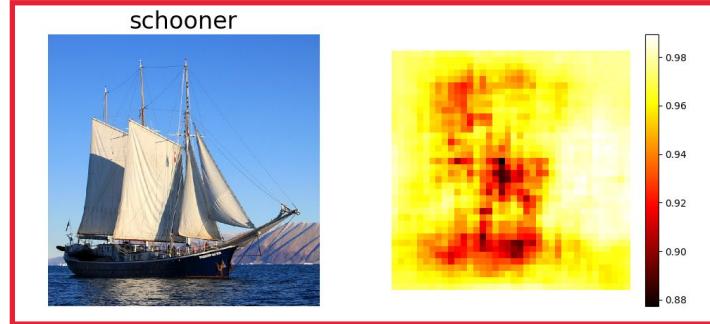


分别遮挡不同部分，看哪些部分受遮挡后输出分值会急剧变化

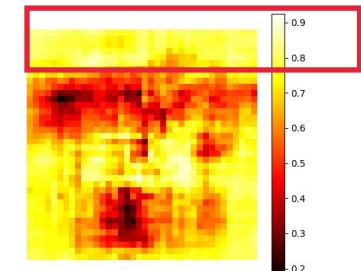
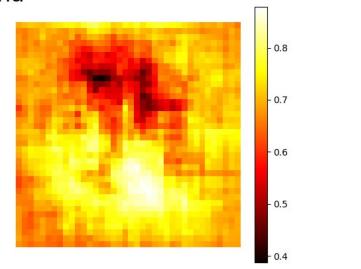


Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

Boat image is CC0 public domain  
Elephant image is CC0 public domain  
Go-Karts image is CC0 public domain

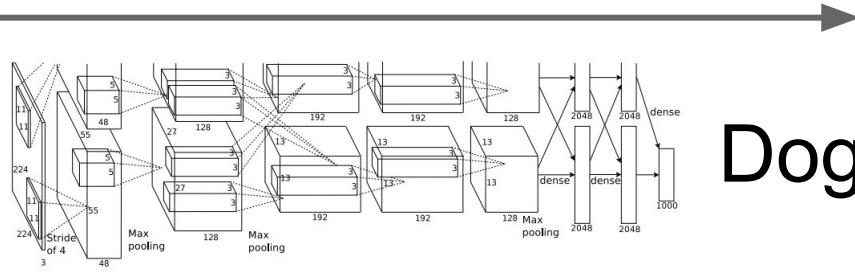


African elephant, Loxodonta africana



# Saliency Maps

How to tell which pixels matter for classification?



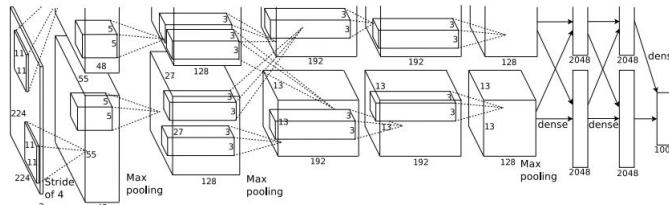
Dog

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

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

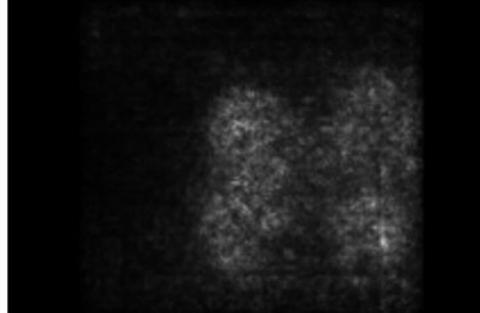
# Saliency Maps

How to tell which pixels matter for classification?



Dog

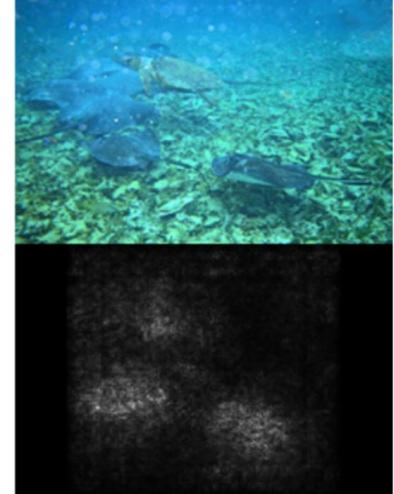
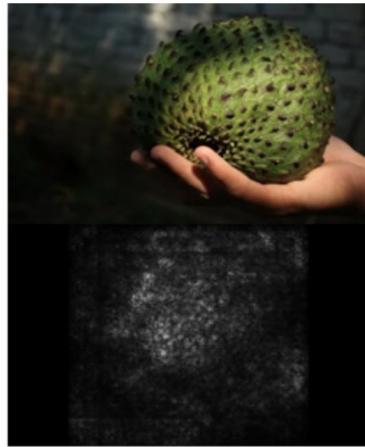
Compute gradient of (unnormalized) class score with respect to image pixels, take absolute value and max over RGB channels



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

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

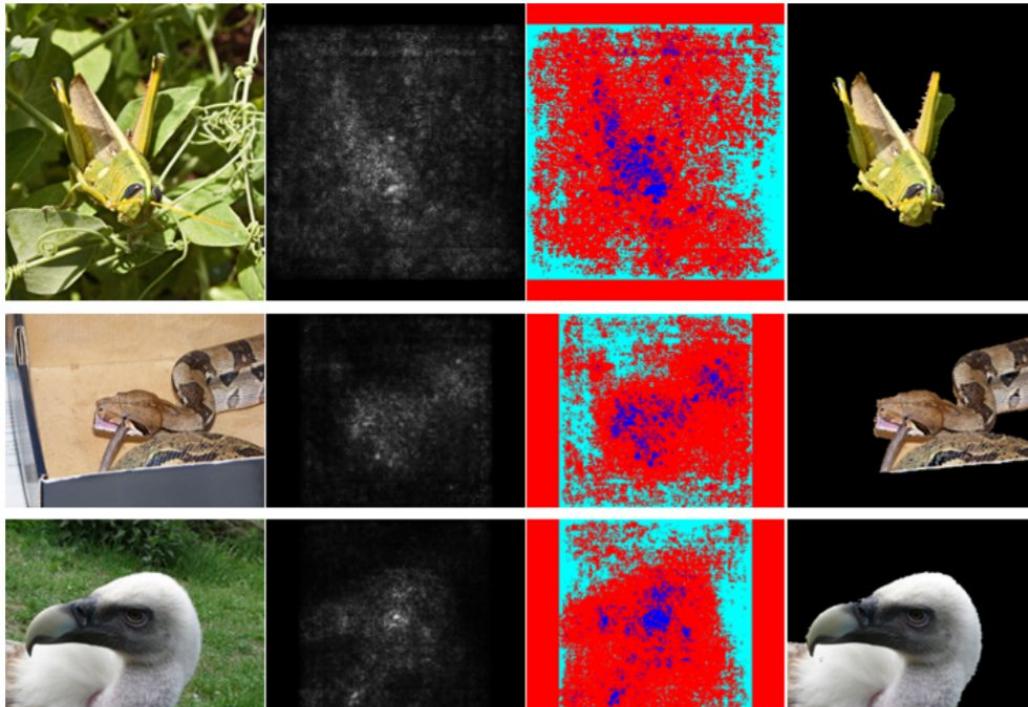
# Saliency Maps



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

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

# Saliency Maps: Segmentation without supervision



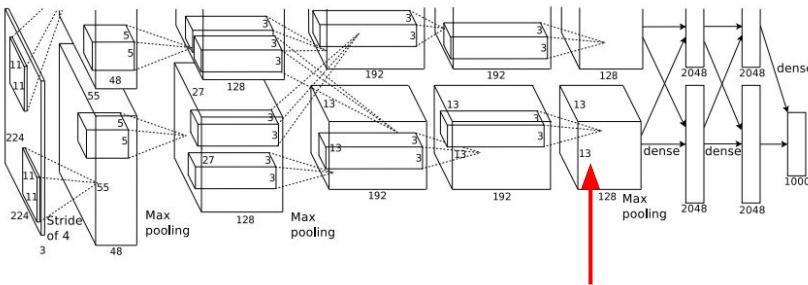
Use GrabCut on  
saliency map

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

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

Rother et al, "Grabcut: Interactive foreground extraction using iterated graph cuts", ACM TOG 2004

# Intermediate Features via (guided) backprop

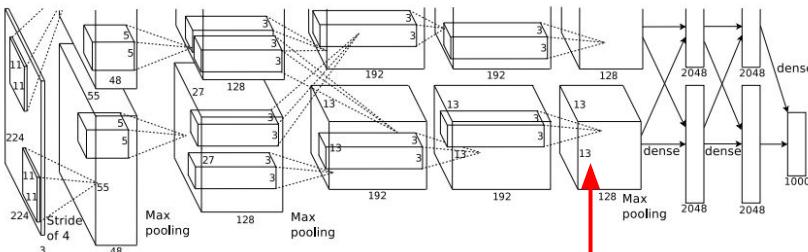


Pick a single intermediate neuron, e.g. one value in  $128 \times 13 \times 13$  conv5 feature map

Compute gradient of neuron value with respect to image pixels

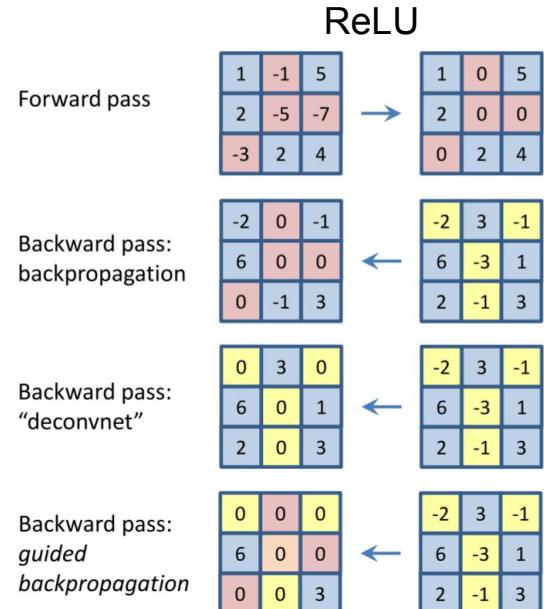
Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014  
Springenberg et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015

# Intermediate features via (guided) backprop



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

Compute gradient of neuron value with respect to image pixels

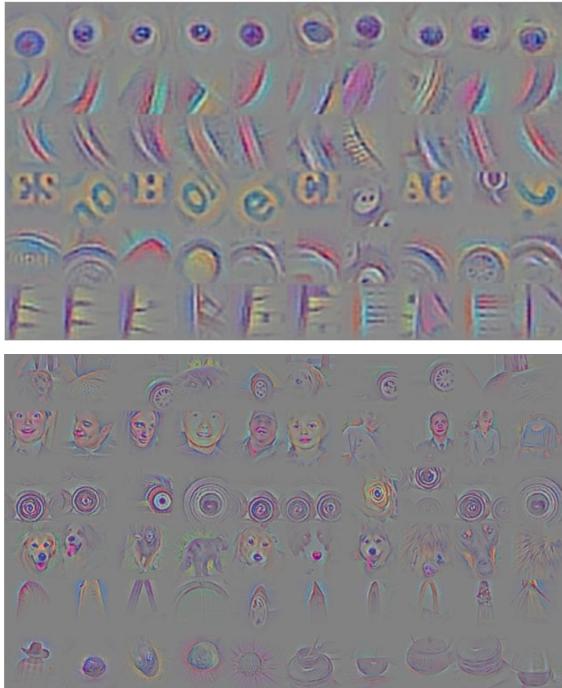


Images come out nicer if you only backprop positive gradients through each ReLU (guided backprop)

Figure copyright Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, 2015; reproduced with permission.

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

# Intermediate features via (guided) backprop



Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

Springenberg et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015

Figure copyright Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, 2015; reproduced with permission.

# Visualizing CNN features: Gradient Ascent

**(Guided) backprop:**

Find the part of an image that a neuron responds to

**Gradient ascent:**

Generate a synthetic image that maximally activates a neuron

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

在图像像素上执行梯度上升来合成图像，尝试最大化中间某些神经元和类的分值

Neuron value

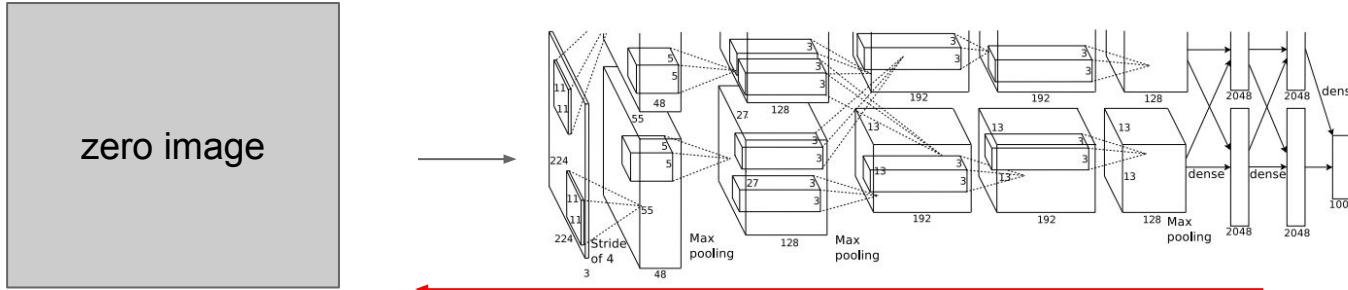
Natural image regularizer

# Visualizing CNN features: Gradient Ascent

1. Initialize image to zeros

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

score for class c (before Softmax)



Repeat:

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

# Visualizing CNN features: Gradient Ascent

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

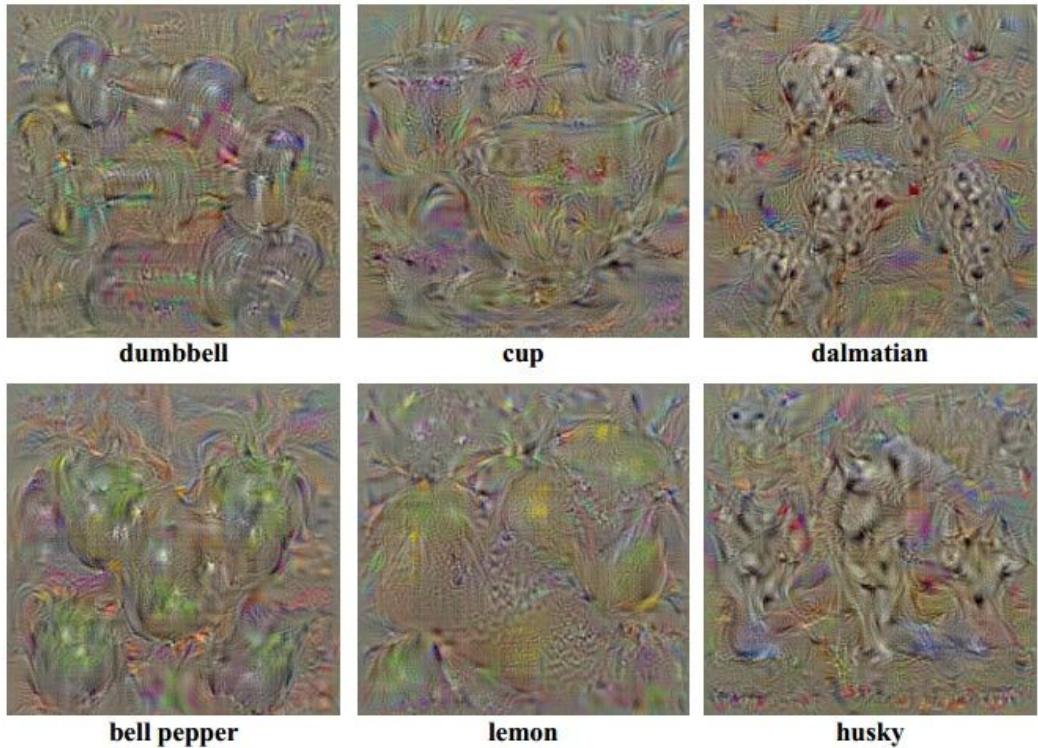
Simple regularizer: Penalize L2  
norm of generated image

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

# Visualizing CNN features: Gradient Ascent

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

Simple regularizer: Penalize L2 norm of generated image



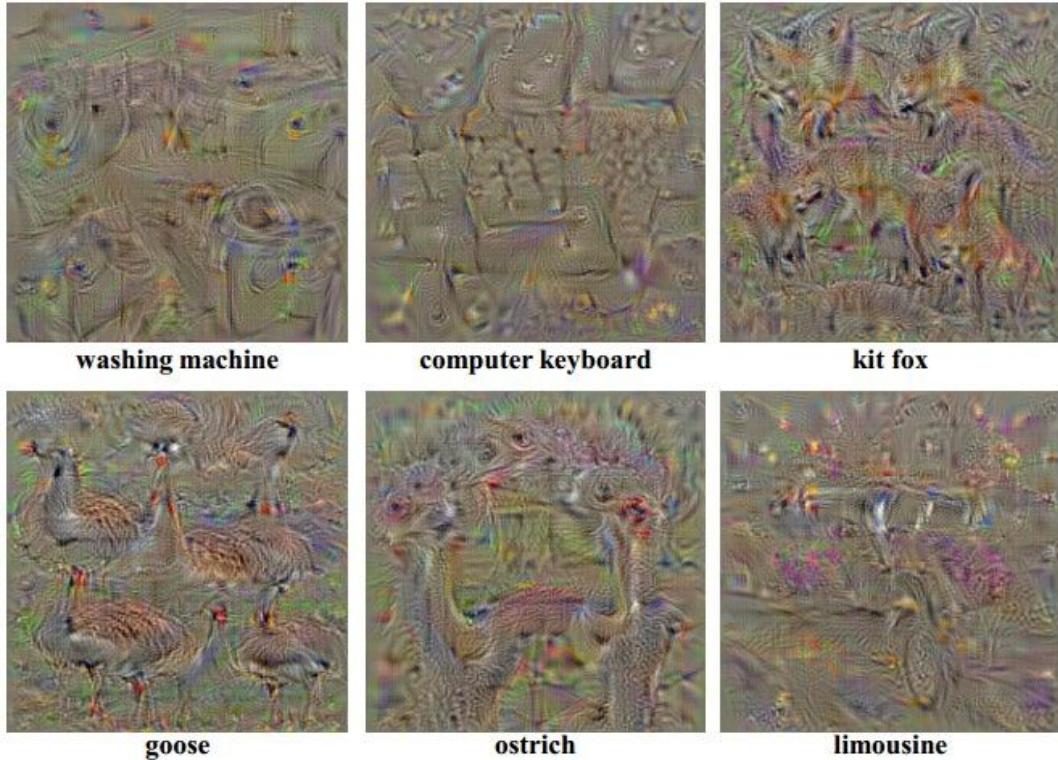
Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.

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

# Visualizing CNN features: Gradient Ascent

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

Simple regularizer: Penalize L2  
norm of generated image



Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.  
Figure copyright Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson, 2014.  
Reproduced with permission.

# Visualizing CNN features: Gradient Ascent

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

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

- (1) Gaussian blur image
- (2) Clip pixels with small values to 0
- (3) Clip pixels with small gradients to 0

# Visualizing CNN features: Gradient Ascent

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

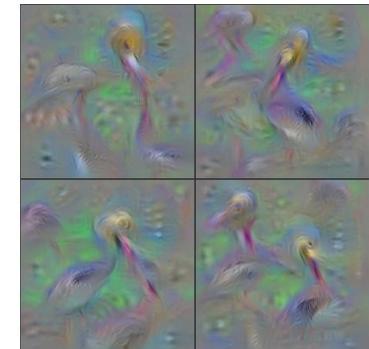
Better regularizer: Penalize L2 norm of image; also during optimization periodically

- (1) Gaussian blur image
- (2) Clip pixels with small values to 0
- (3) Clip pixels with small gradients to 0

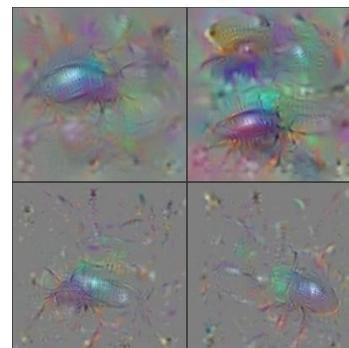
刚开始输入随机初始化图像?



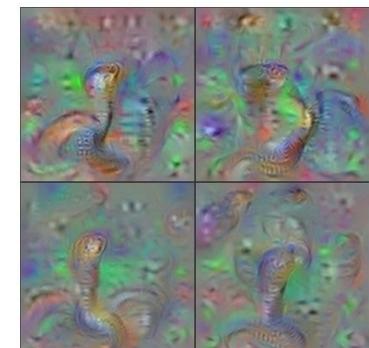
Flamingo



Pelican



Ground Beetle



Indian Cobra

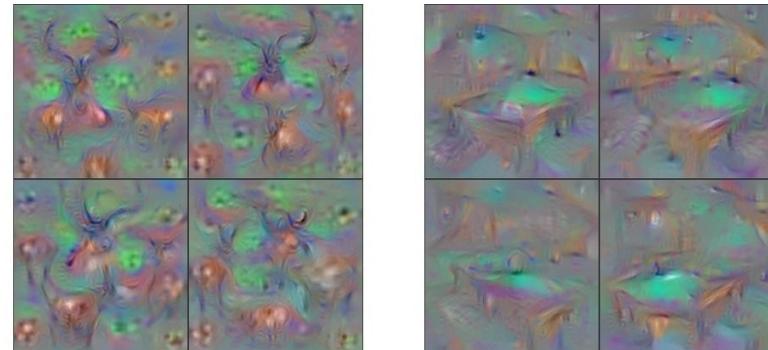
Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.  
Figure copyright Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson, 2014. Reproduced with permission.

# Visualizing CNN features: Gradient Ascent

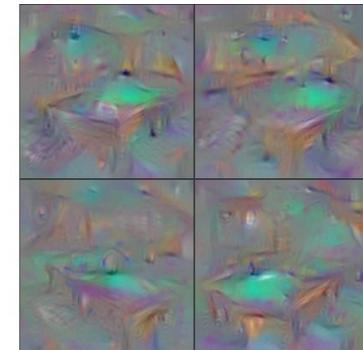
$$\arg \max_I S_c(I) - \lambda \|I\|_2^2$$

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

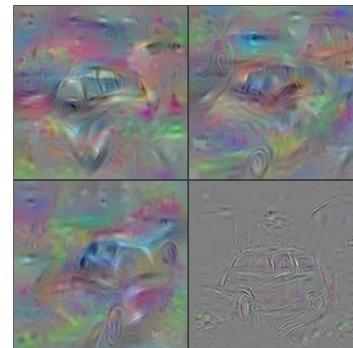
- (1) Gaussian blur image
- (2) Clip pixels with small values to 0
- (3) Clip pixels with small gradients to 0



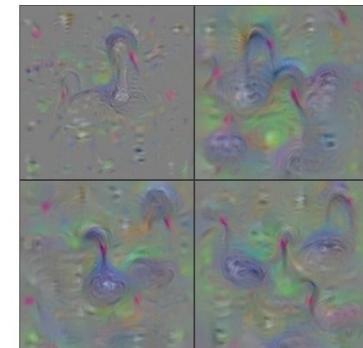
Hartebeest



Billiard Table



Station Wagon

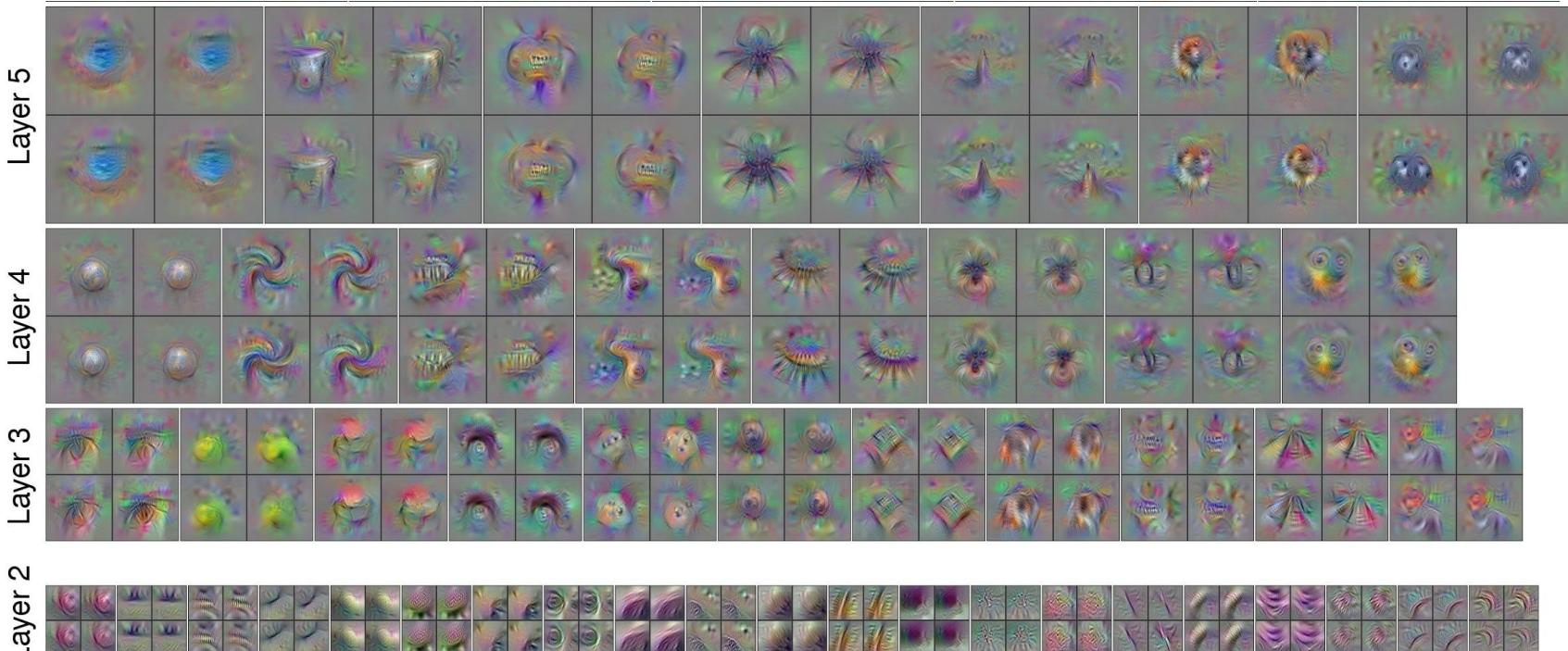


Black Swan

Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.  
Figure copyright Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson, 2014. Reproduced with permission.

# Visualizing CNN features: Gradient Ascent

Use the same approach to visualize intermediate features

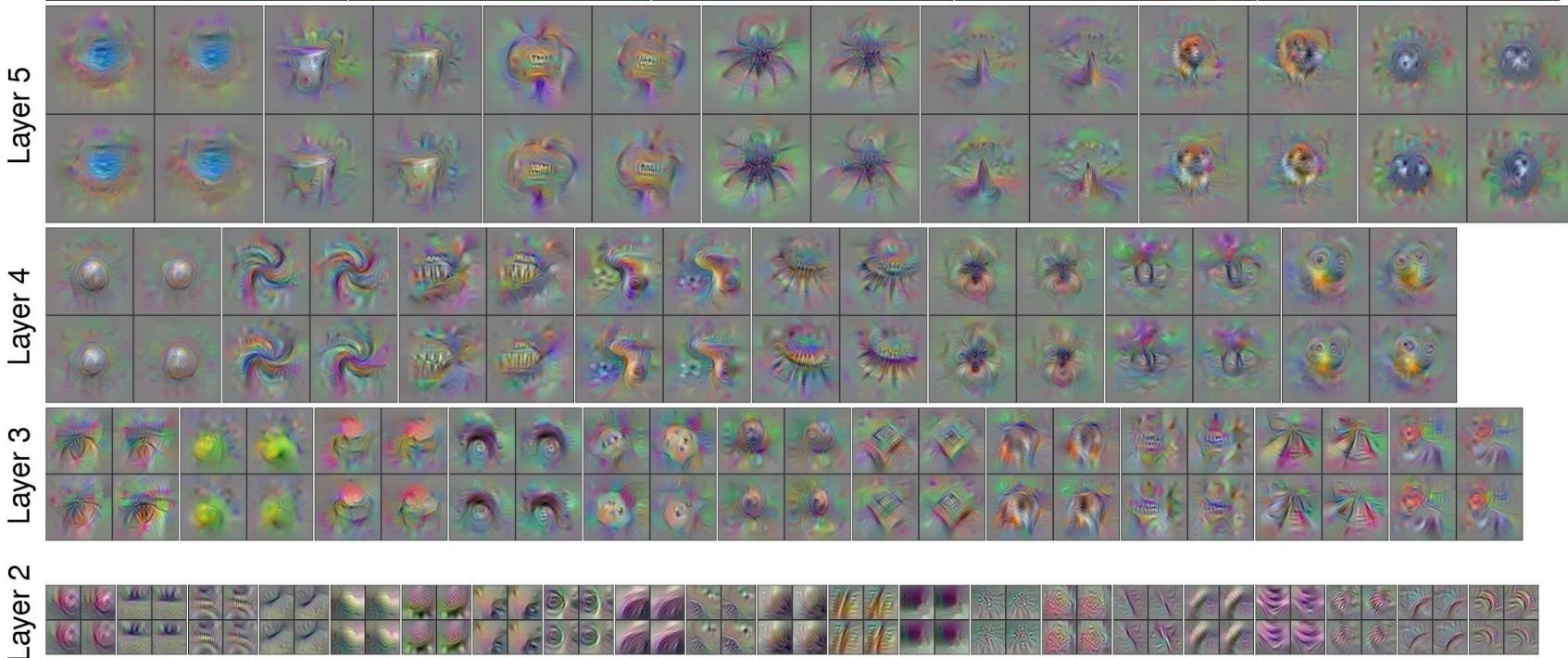


Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.

Figure copyright Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson, 2014. Reproduced with permission.

# Visualizing CNN features: Gradient Ascent

Use the same approach to visualize intermediate features



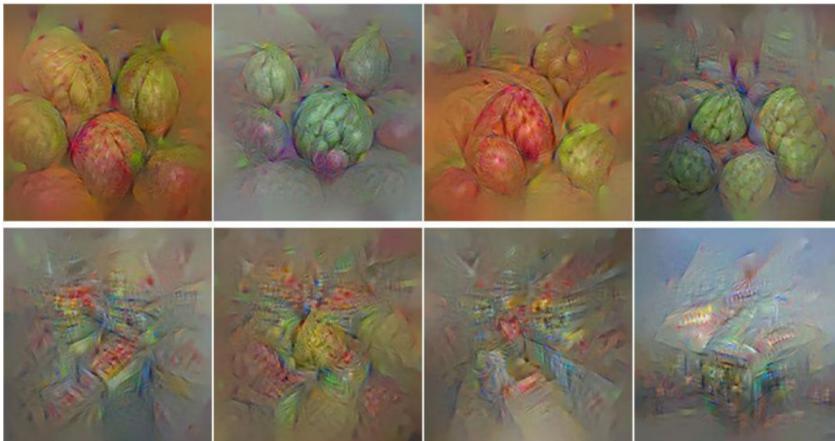
Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.

Figure copyright Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson, 2014. Reproduced with permission.

# Visualizing CNN features: Gradient Ascent

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

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



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



Nguyen et al., “Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks”, ICML Visualization for Deep Learning Workshop 2016.  
Figures copyright Anh Nguyen, Jason Yosinski, and Jeff Clune, 2016; reproduced with permission.

# Visualizing CNN features: Gradient Ascent



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

Figures copyright Anh Nguyen, Jason Yosinski, and Jeff Clune, 2016; reproduced with permission.

# Visualizing CNN features: Gradient Ascent

Optimize in FC6 latent space instead of pixel space:



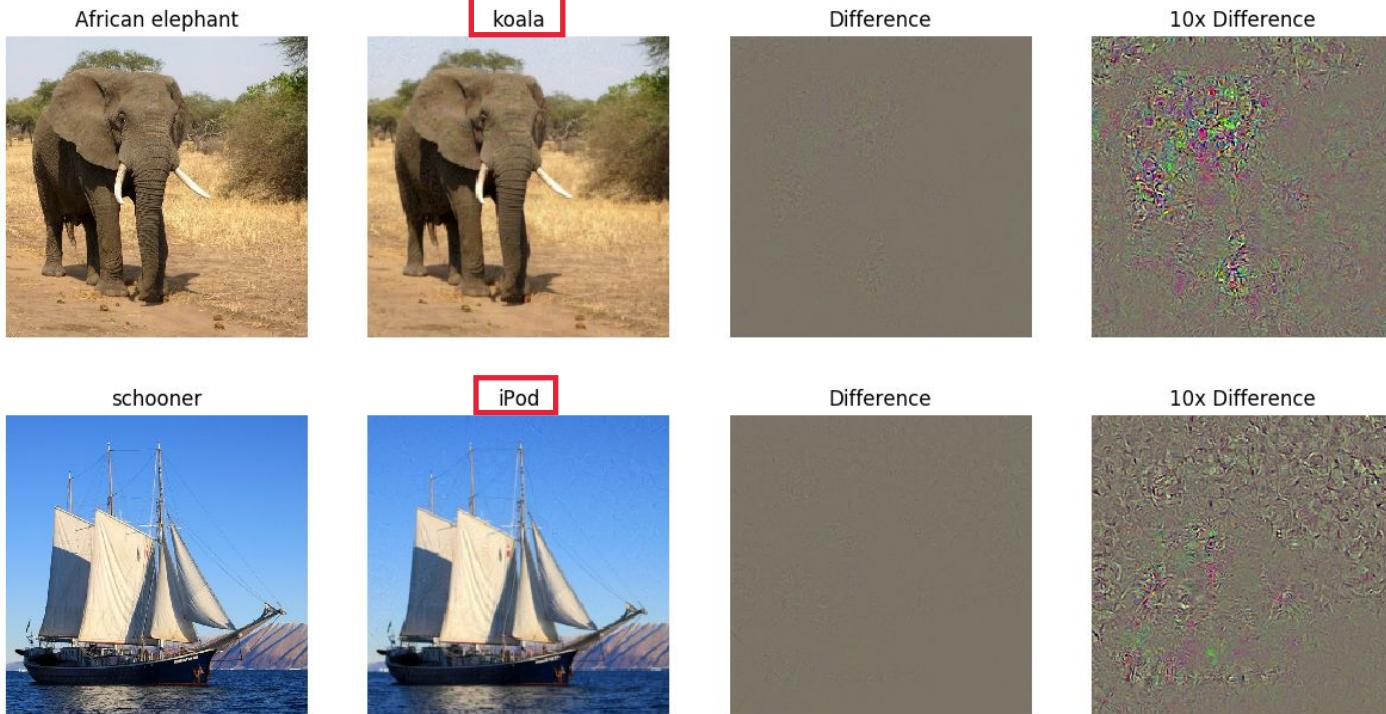
Nguyen et al., "Synthesizing the preferred inputs for neurons in neural networks via deep generator networks," NIPS 2016

Figure copyright Nguyen et al, 2016; reproduced with permission.

# Fooling Images / Adversarial Examples

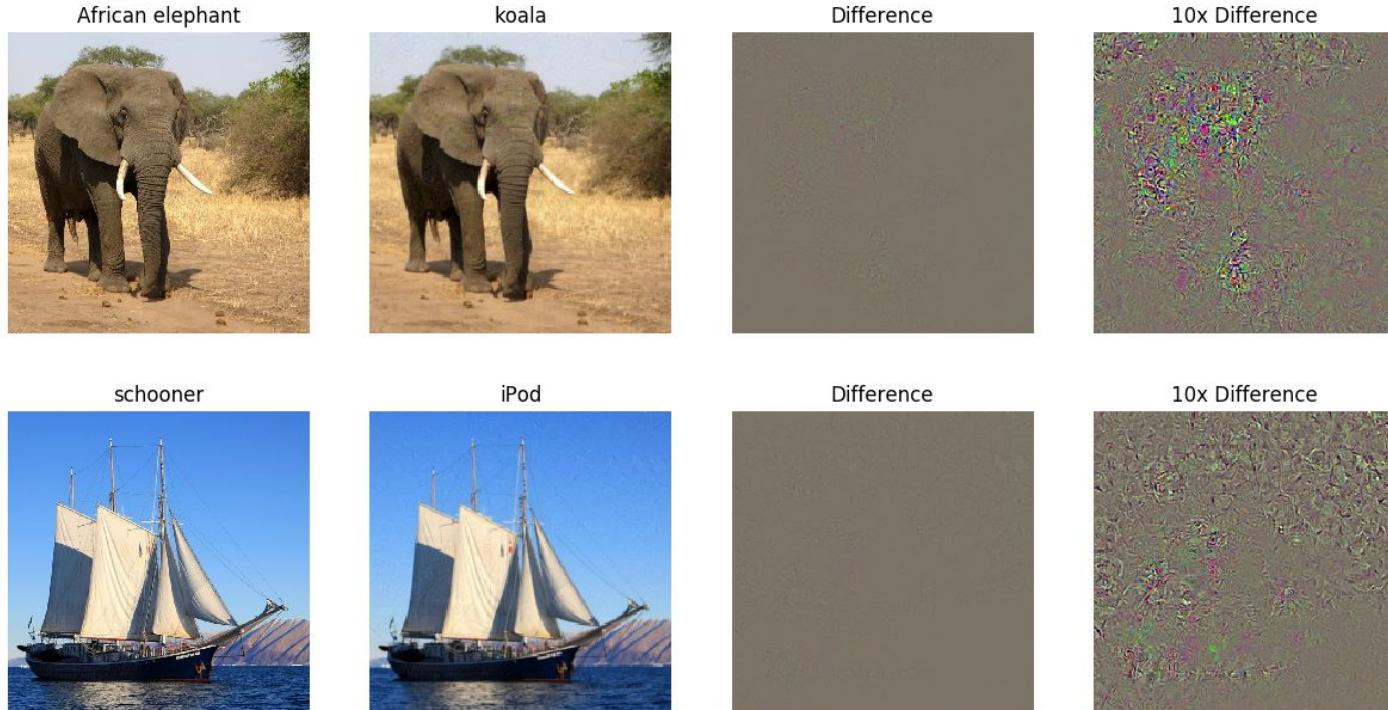
- (1) Start from an arbitrary image
- (2) Pick an arbitrary class
- (3) Modify the image to maximize the class
- (4) Repeat until network is fooled

# Fooling Images / Adversarial Examples



[Boat image is CC0 public domain](#)  
[Elephant image is CC0 public domain](#)

# Fooling Images / Adversarial Examples

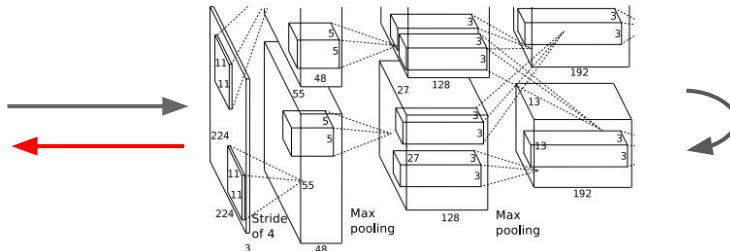


What is going on? Ian Goodfellow will explain

Boat image is CC0 public domain  
Elephant image is CC0 public domain

# 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



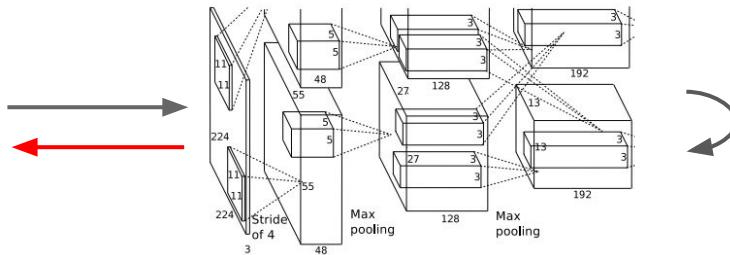
Choose an image and a layer in a CNN; repeat:

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

```
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](#))

# 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

为了使图像变得更平滑

# 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)



Sky image is licensed under CC-BY SA 3.0

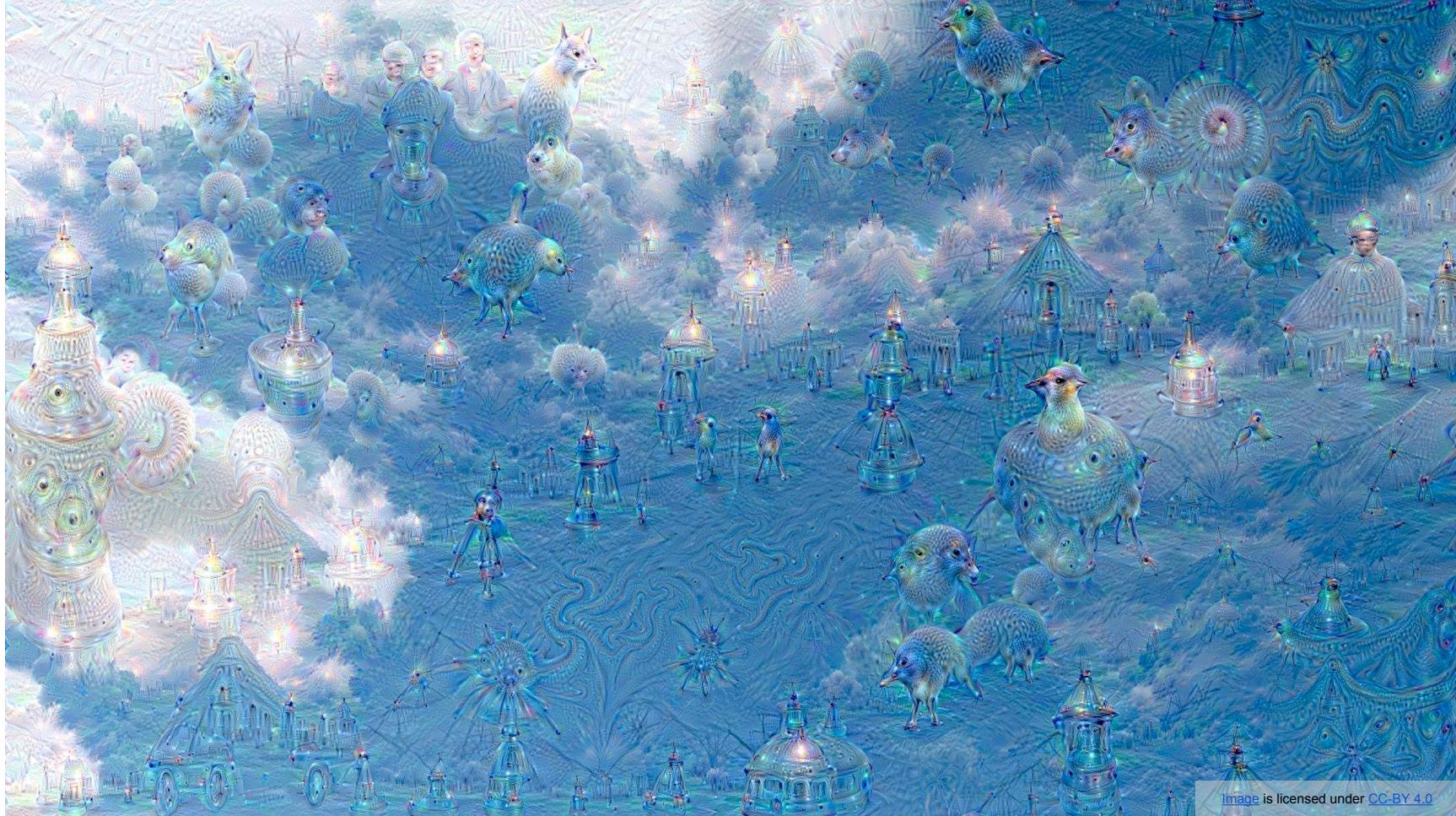
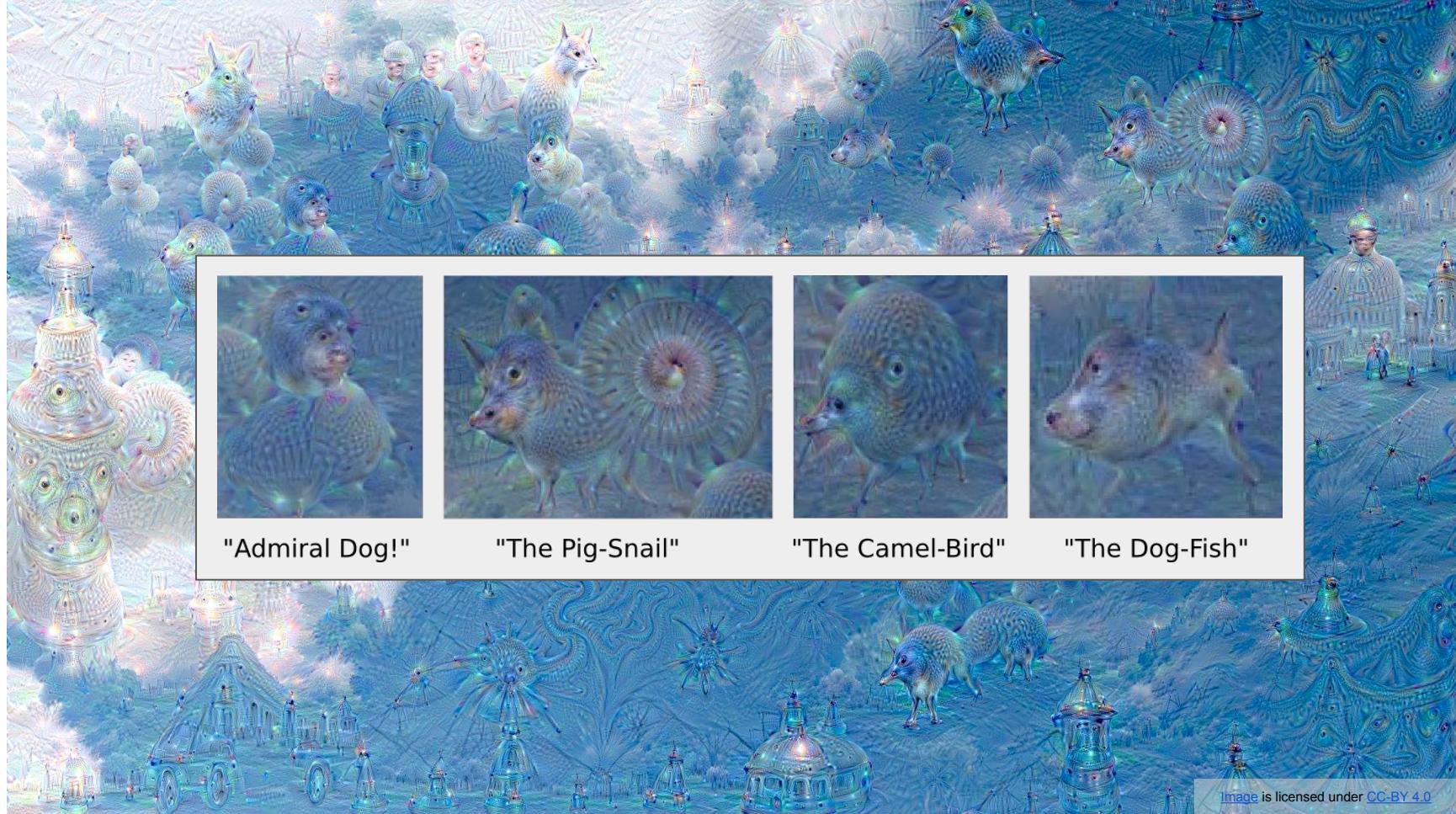


Image is licensed under CC-BY 4.0



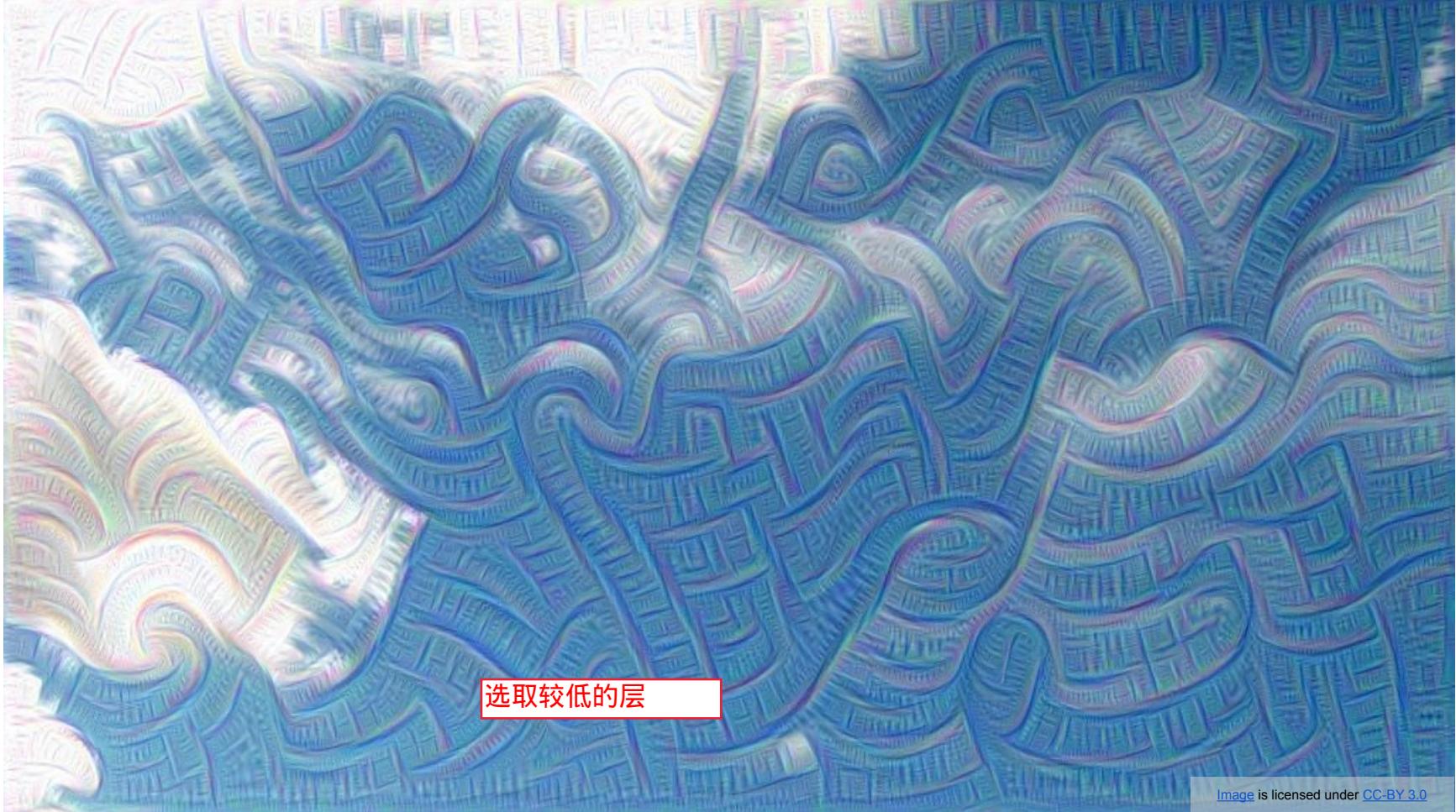
"Admiral Dog!"

"The Pig-Snail"

"The Camel-Bird"

"The Dog-Fish"

Image is licensed under CC-BY 4.0



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



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

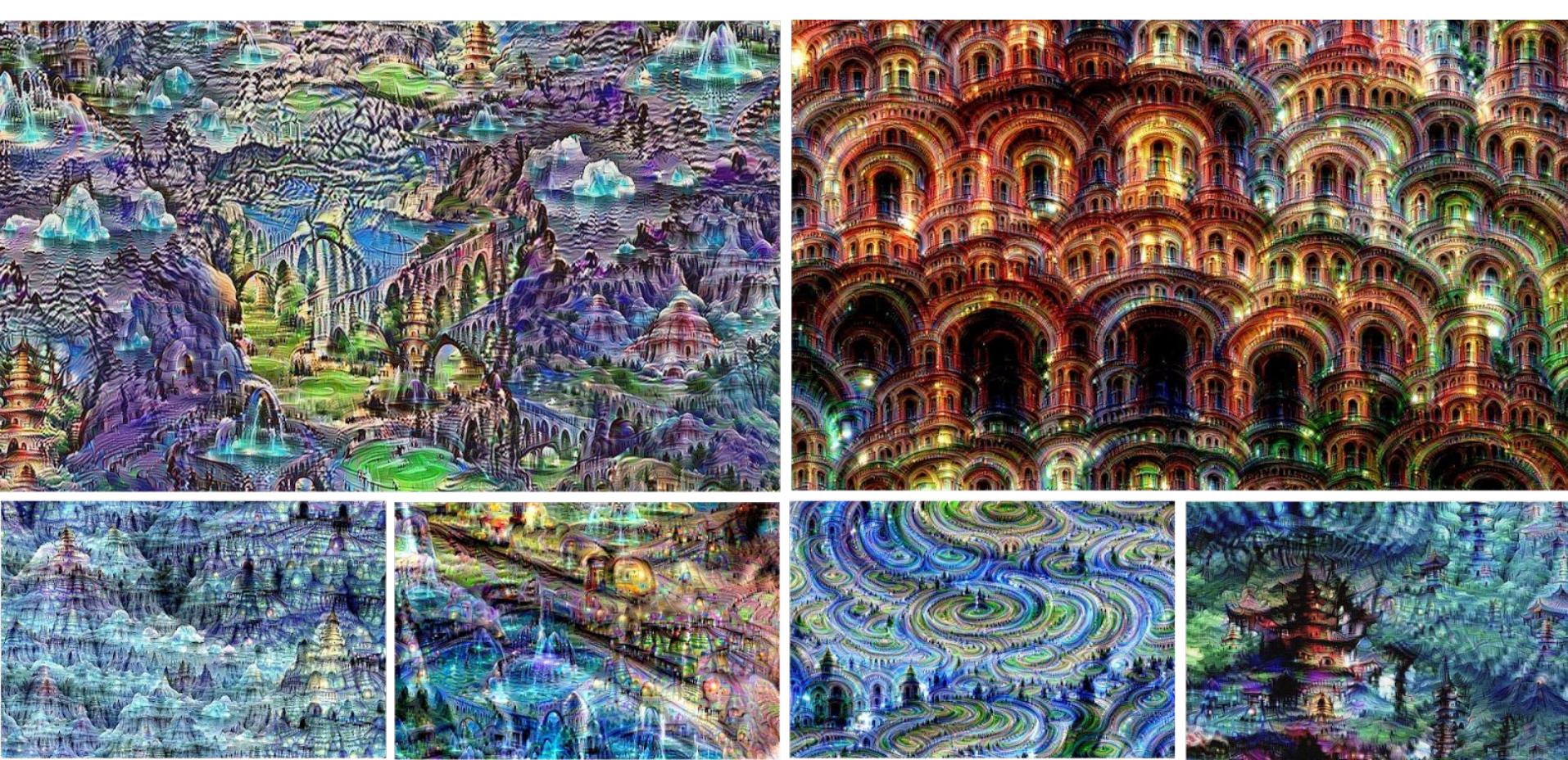


Image is licensed under CC-BY 4.0

# 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

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

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

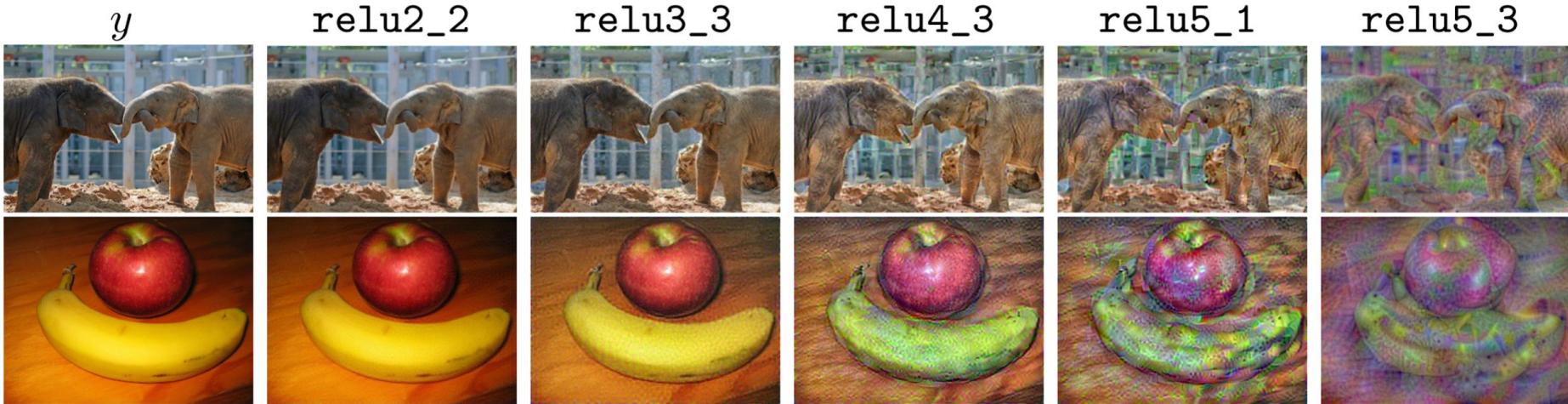
Total Variation regularizer  
(encourages spatial smoothness)

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

# Feature Inversion

越往后，图的空间结构会被保留，但低层次的细节会丢失

Reconstructing from different layers of VGG-16



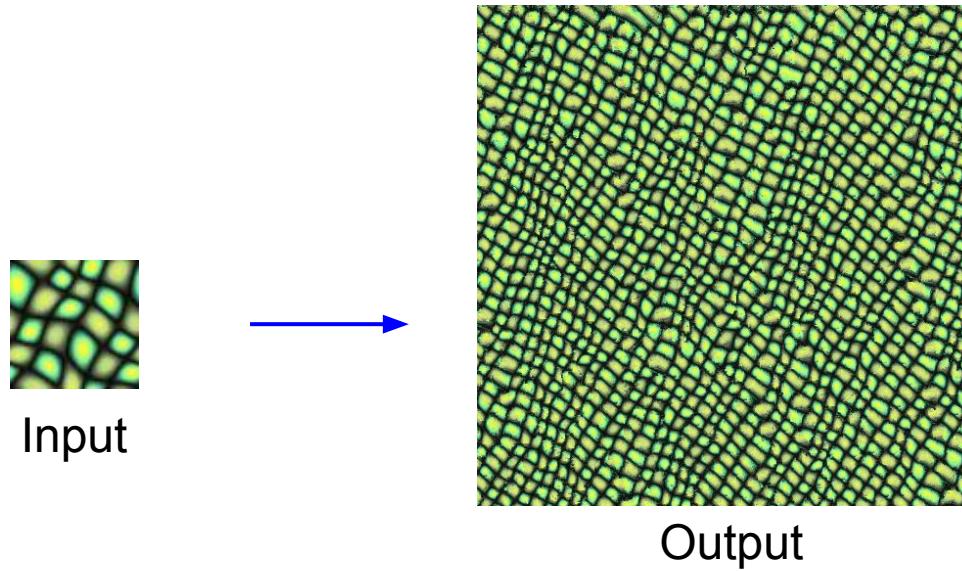
Mahendran and Vedaldi, "Understanding Deep Image Representations by Inverting Them", CVPR 2015

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.

# Texture Synthesis

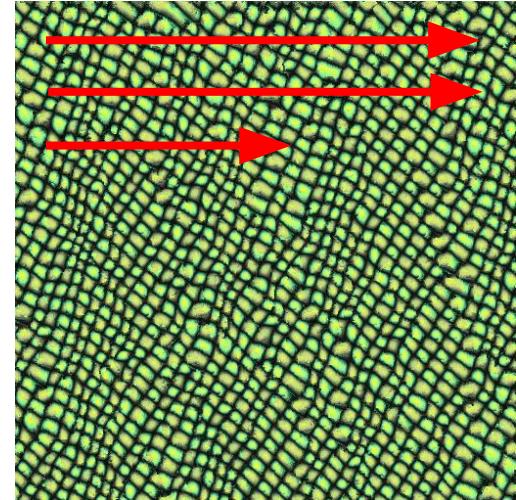
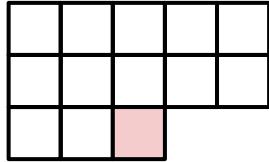
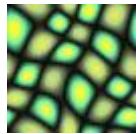
Given a sample patch of some texture, can we generate a bigger image of the same texture?



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

# Texture Synthesis: Nearest Neighbor

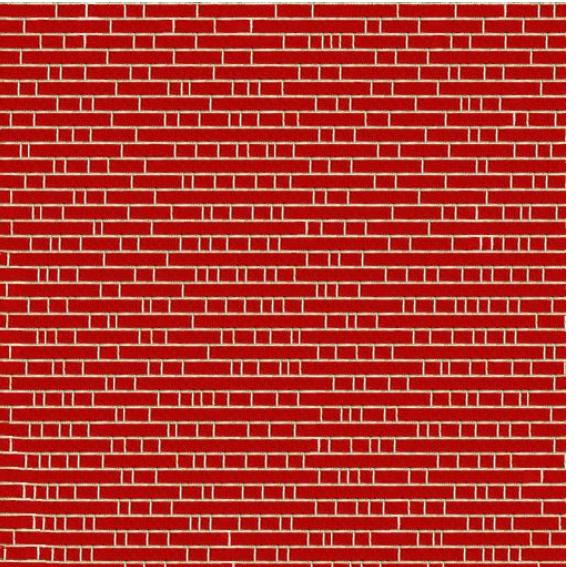
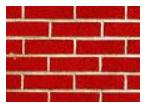
Generate pixels one at a time in scanline order; form neighborhood of already generated pixels and copy nearest neighbor from input



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

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

# Texture Synthesis: Nearest Neighbor



itndwars ouene Tring rooms," as Heff te fast of th  
ira dat noemt certes rihet tlastet hert bedan Al I  
conceal Honts, as Farre, as do Lewin, as  
Man, as Leving, as Lest, as Lest, as Lest. He  
is dan Al last ful council liew at "this dayss dyl  
einfachl. Hoerwing rooms," as House De fale dyl  
und tical conuenerched tlast fall. He fall. Heff  
is oround it at the left a right, quanta Lewin,  
jers, as a stary of Mous, Lewin, as  
a Thar Fing, as a stary of Mous, Lewin, as  
houster of Mie lufit. A list fast of the lauerd car Heff  
id it yip? Tafouself, a ringind tmonedt a ring  
artical cois ore years of Moung fall. He rob Mous  
years, as a stary of Mous, Lewin, as a stary of Mous  
as Tapp. Political connelis. Al he few, ring  
politicl. Con repes of the stoxess of tis. Fratva li  
ses Lewin le rume. He has questring of, albeo

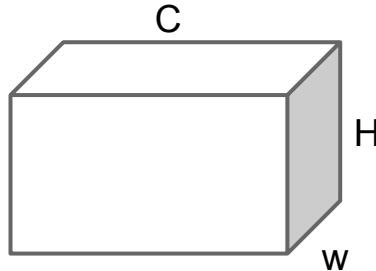
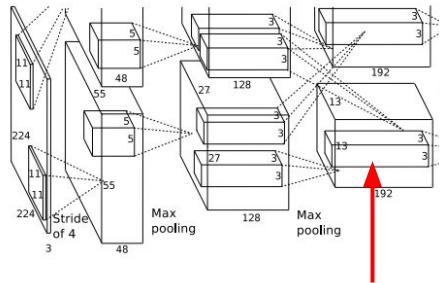


itndwars ouene Tring rooms," as Heff te fast of th  
ira dat noemt certes rihet tlastet hert bedan Al I  
conceal Honts, as Farre, as do Lewin, as  
Man, as Leving, as Lest, as Lest, as Lest. He  
is dan Al last ful council liew at "this dayss dyl  
einfachl. Hoerwing rooms," as House De fale dyl  
und tical conuenerched tlast fall. He fall. Heff  
is oround it at the left a right, quanta Lewin,  
jers, as a stary of Mous, Lewin, as  
a Thar Fing, as a stary of Mous, Lewin, as  
houster of Mie lufit. A list fast of the lauerd car Heff  
id it yip? Tafouself, a ringind tmonedt a ring  
artical cois ore years of Moung fall. He rob Mous  
years, as a stary of Mous, Lewin, as a stary of Mous  
as Tapp. Political connelis. Al he few, ring  
politicl. Con repes of the stoxess of tis. Fratva li  
ses Lewin le rume. He has questring of, albeo

# Neural Texture Synthesis: Gram Matrix



This image is in the public domain.

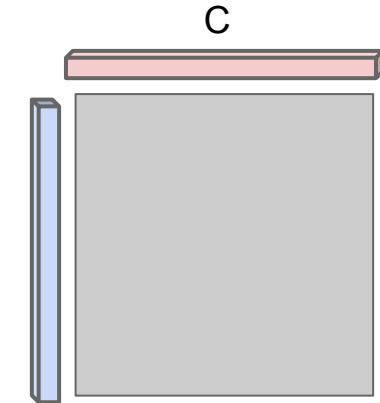
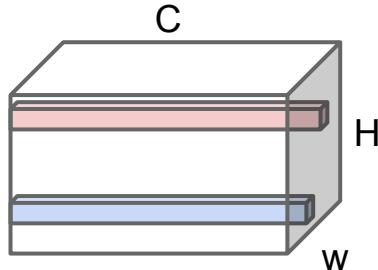
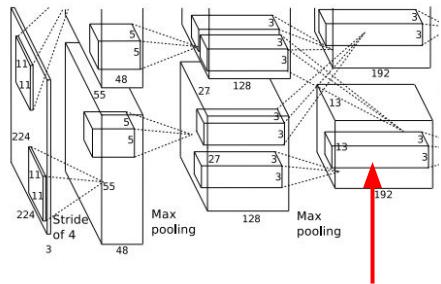


Each layer of CNN gives  $C \times H \times W$  tensor of features;  $H \times W$  grid of  $C$ -dimensional vectors

# Neural Texture Synthesis: Gram Matrix



This image is in the public domain.



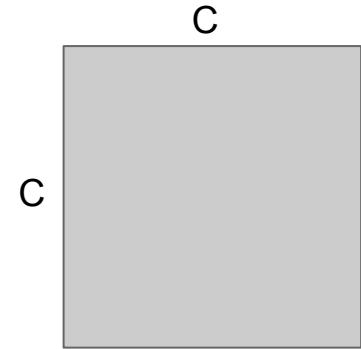
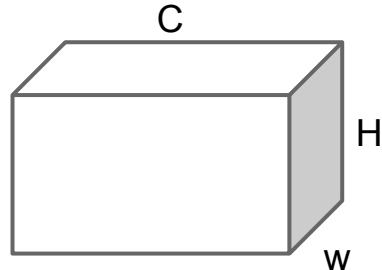
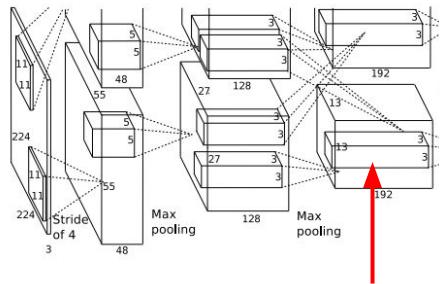
Each layer of CNN gives  $C \times H \times W$  tensor of features;  $H \times W$  grid of  $C$ -dimensional vectors

Outer product of two  $C$ -dimensional vectors gives  $C \times C$  matrix measuring co-occurrence

# Neural Texture Synthesis: Gram Matrix



This image is in the public domain.



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

Outer product of two  $C$ -dimensional vectors gives  $C \times C$  matrix measuring co-occurrence

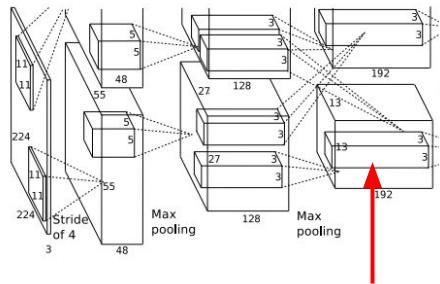
Average over all **HW pairs** of vectors, giving **Gram matrix** of shape  $C \times C$

Gram Matrix

# Neural Texture Synthesis: Gram Matrix



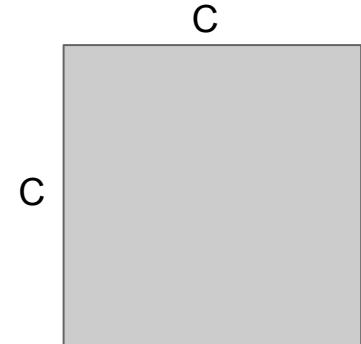
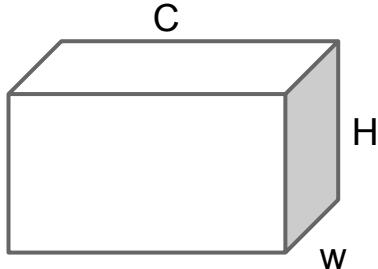
This image is in the public domain.



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

Outer product of two  $C$ -dimensional vectors gives  $C \times C$  matrix measuring co-occurrence

Average over all  $HW$  pairs of vectors, giving **Gram matrix** of shape  $C \times C$



Efficient to compute; reshape features from

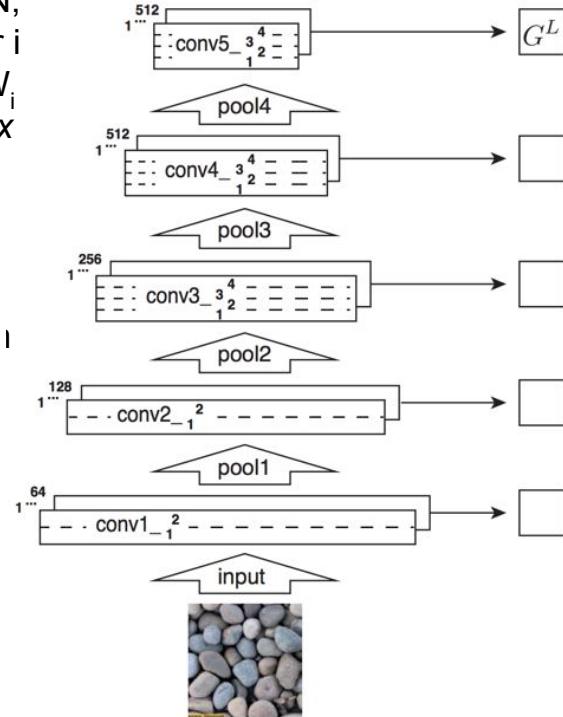
$C \times H \times W$  to  $=C \times HW$

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:

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



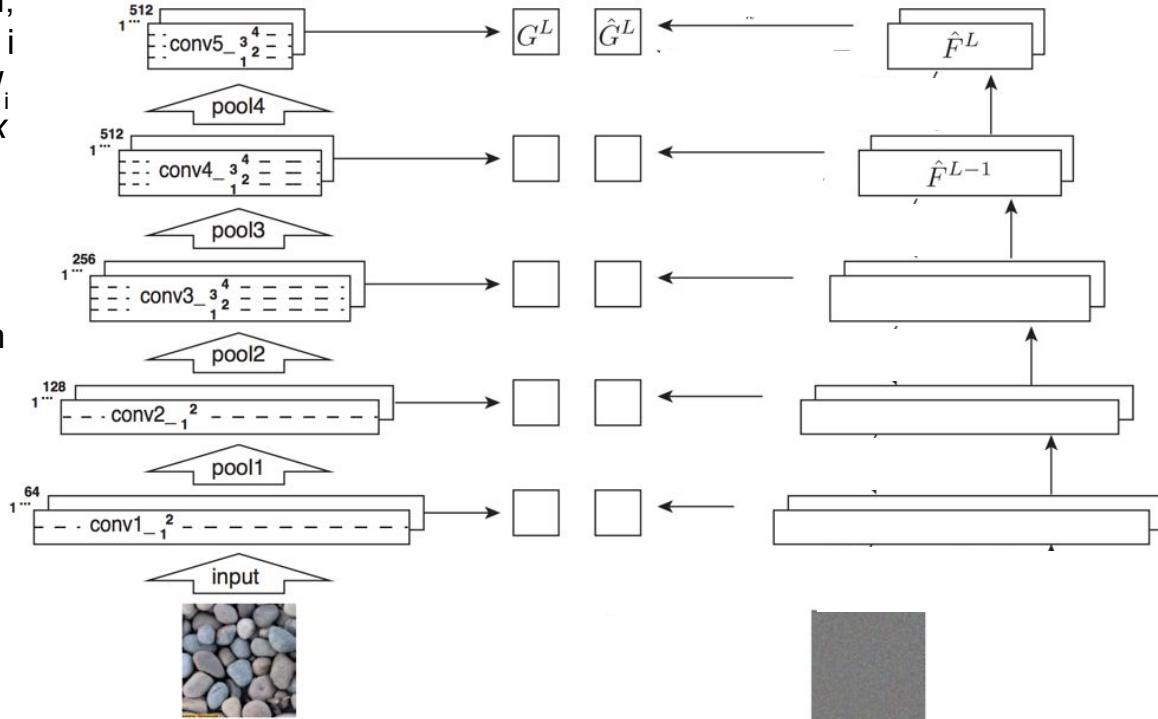
Gatys, Ecker, and Bethge, "Texture Synthesis Using Convolutional Neural Networks", NIPS 2015  
Figure copyright Leon Gatys, Alexander S. Ecker, and Matthias Bethge, 2015. Reproduced with permission.

# 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:

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

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



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

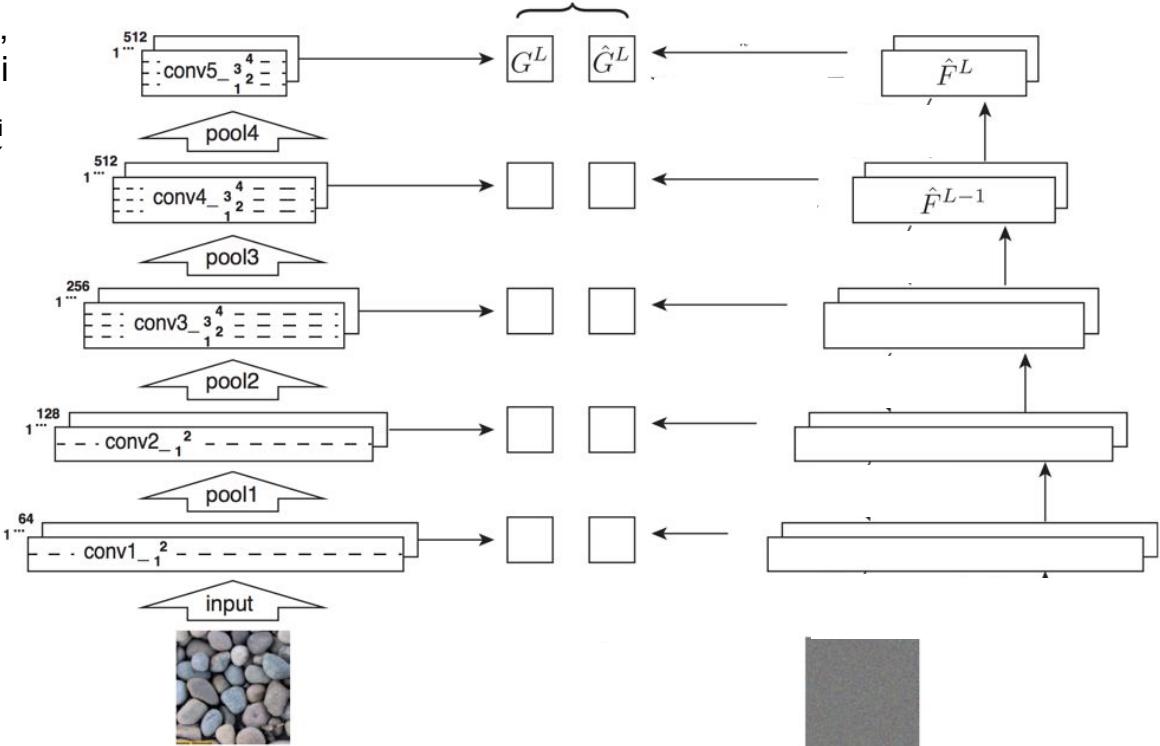
1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer  $i$  gives feature map of shape  $C_i \times H_i \times W_i$
3. At each layer compute the *Gram matrix* giving outer product of features:

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

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

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

$$\mathcal{L}(\vec{x}, \hat{x}) = \sum_{l=0}^L w_l E_l$$



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

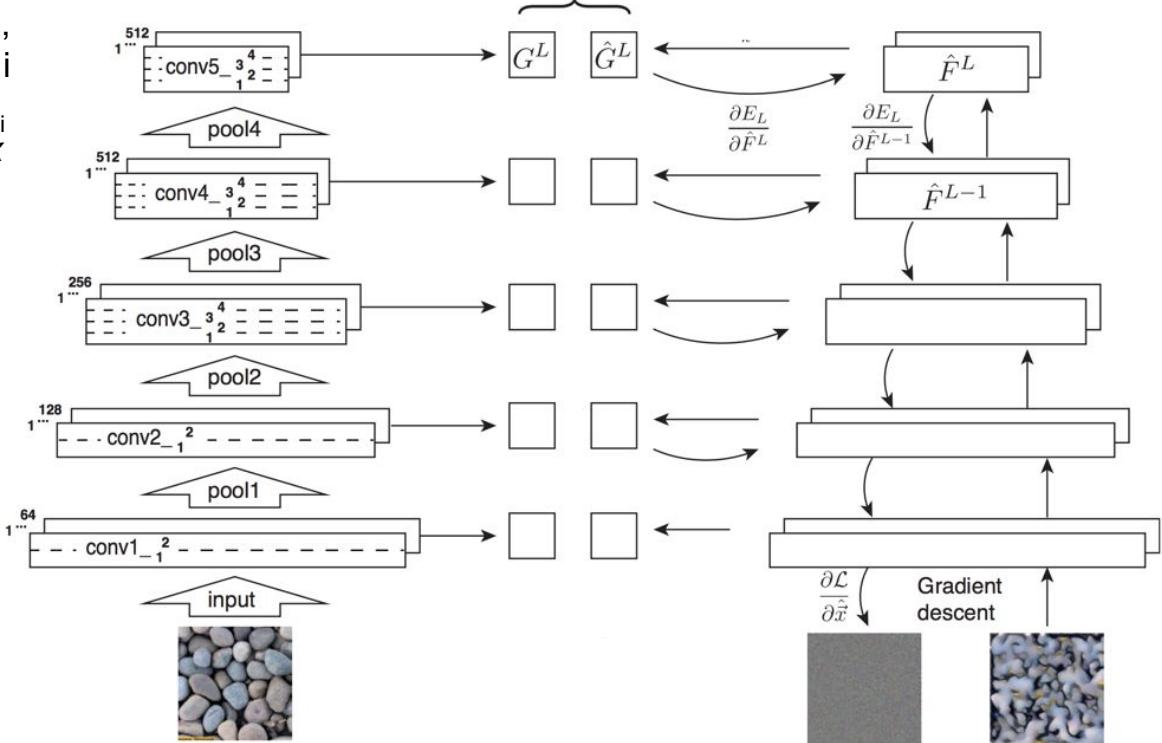
1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer  $i$  gives feature map of shape  $C_i \times H_i \times W_i$
3. At each layer compute the *Gram matrix* giving outer product of features:

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

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

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

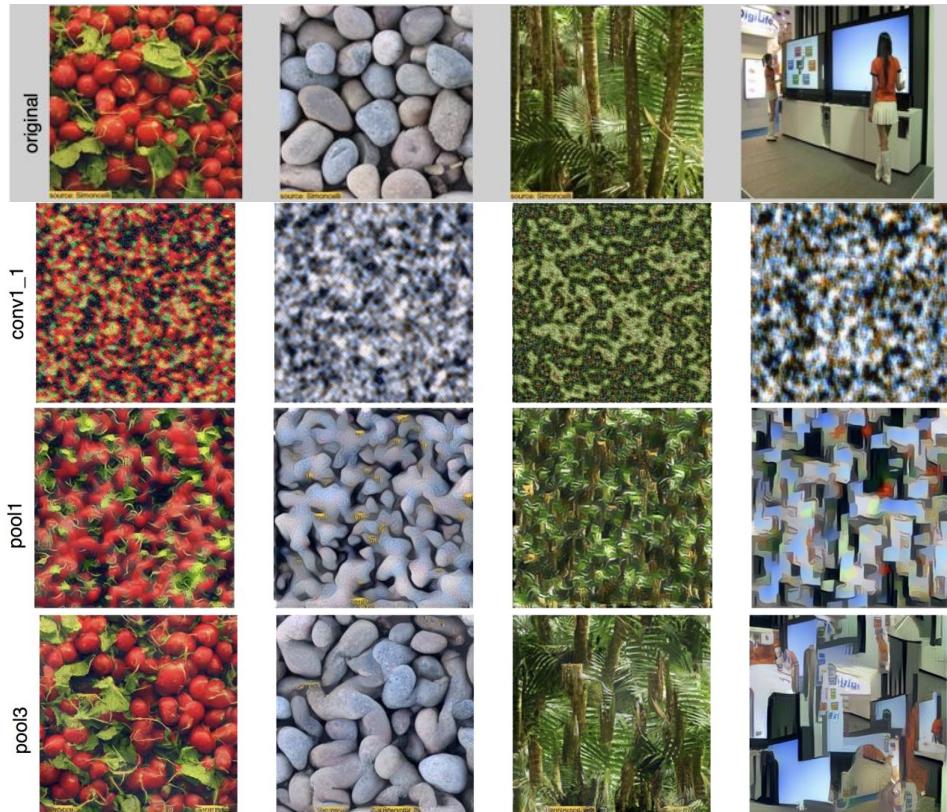
$$\mathcal{L}(\vec{x}, \hat{x}) = \sum_{l=0}^L w_l E_l$$



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)

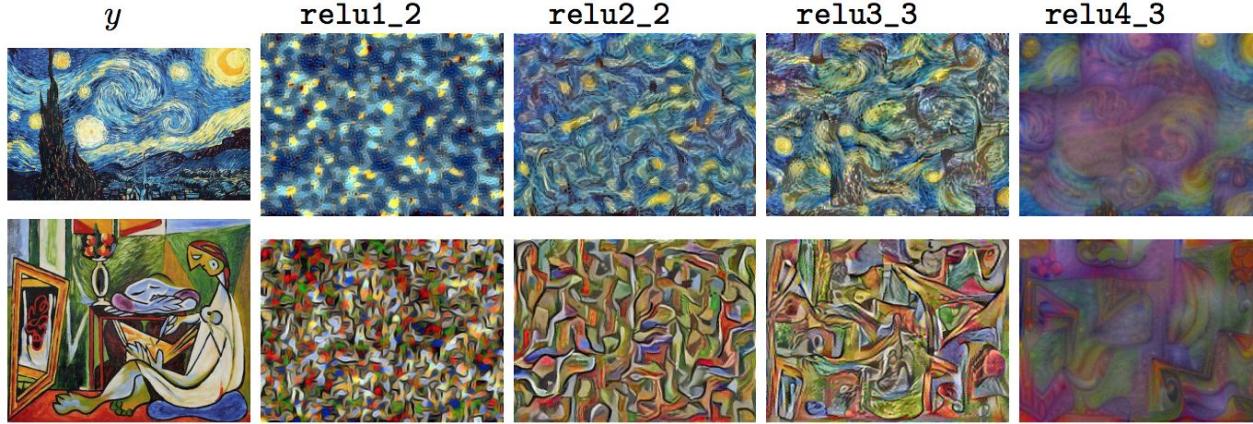
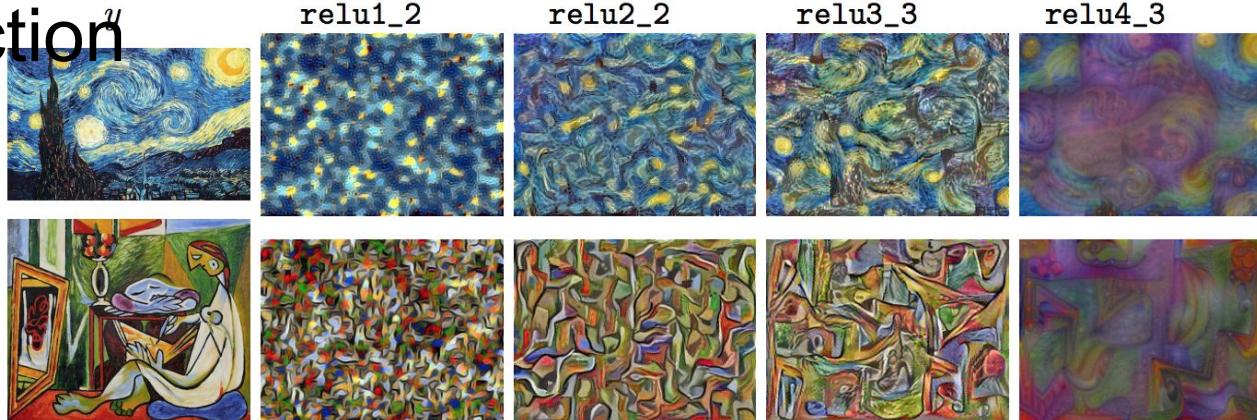


Figure from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016. Copyright Springer, 2016.  
Reproduced for educational purposes.

# Neural Style Transfer: Feature + Gram

## Reconstruction

Texture synthesis  
(Gram  
reconstruction)



Feature  
reconstruction

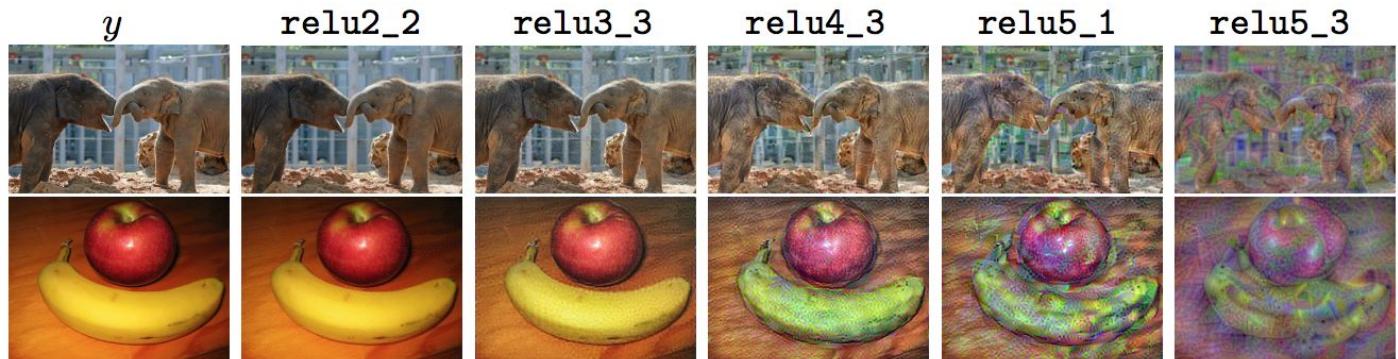


Figure from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016. Copyright Springer, 2016. Reproduced for educational purposes.

# 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



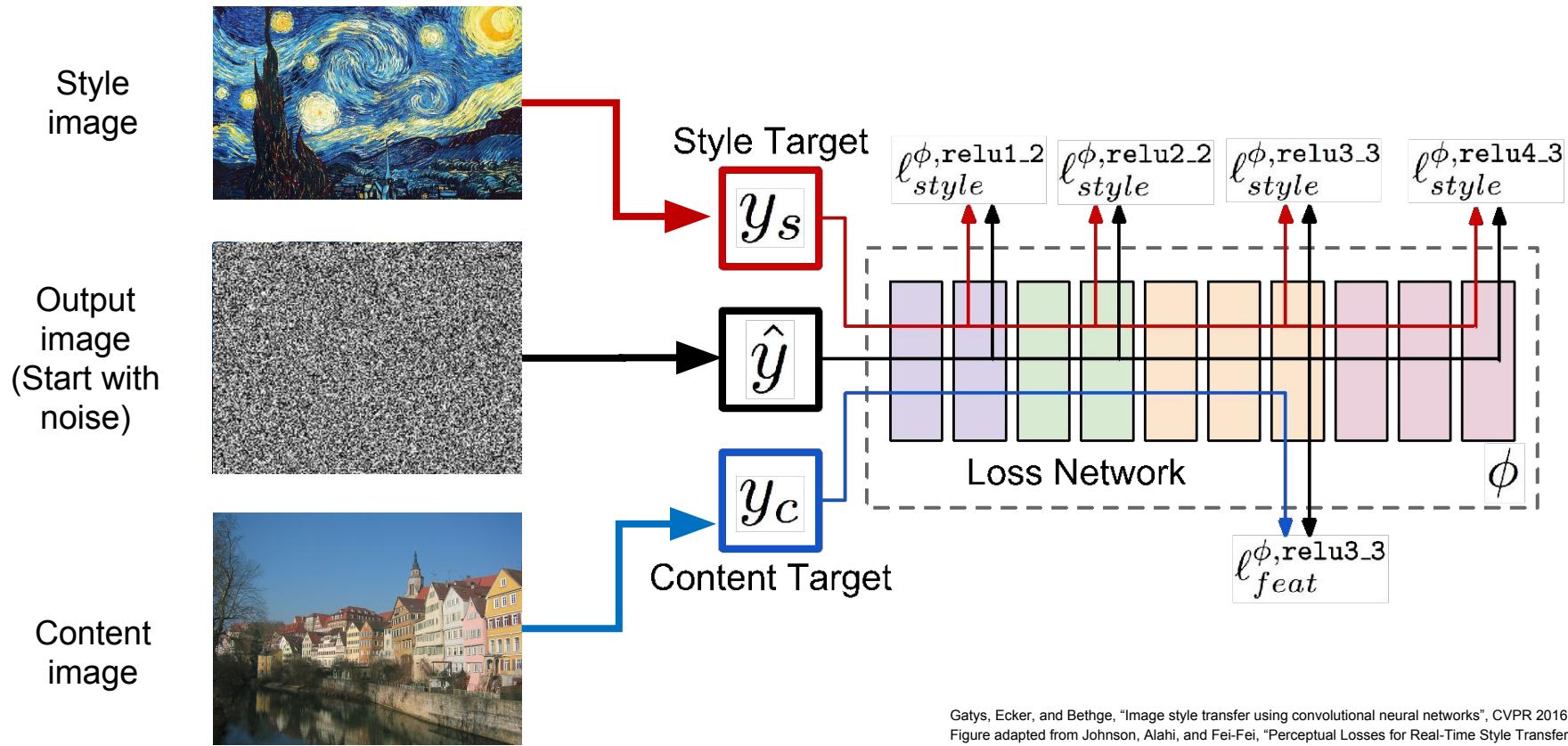
[Starry Night](#) by Van Gogh is in the public domain

=

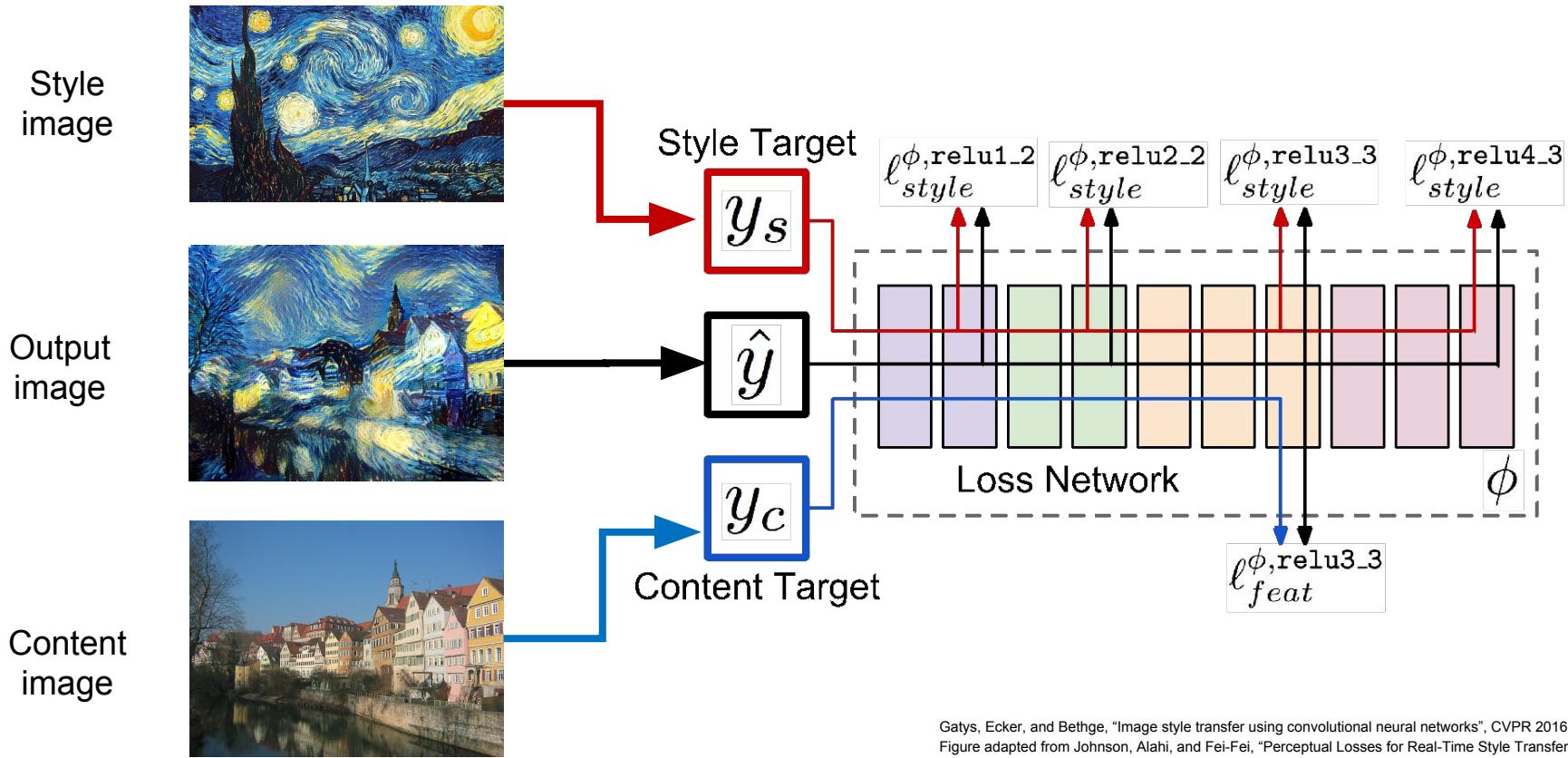


[This image](#) copyright Justin Johnson, 2015. Reproduced with permission.

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



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.



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  
[my 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

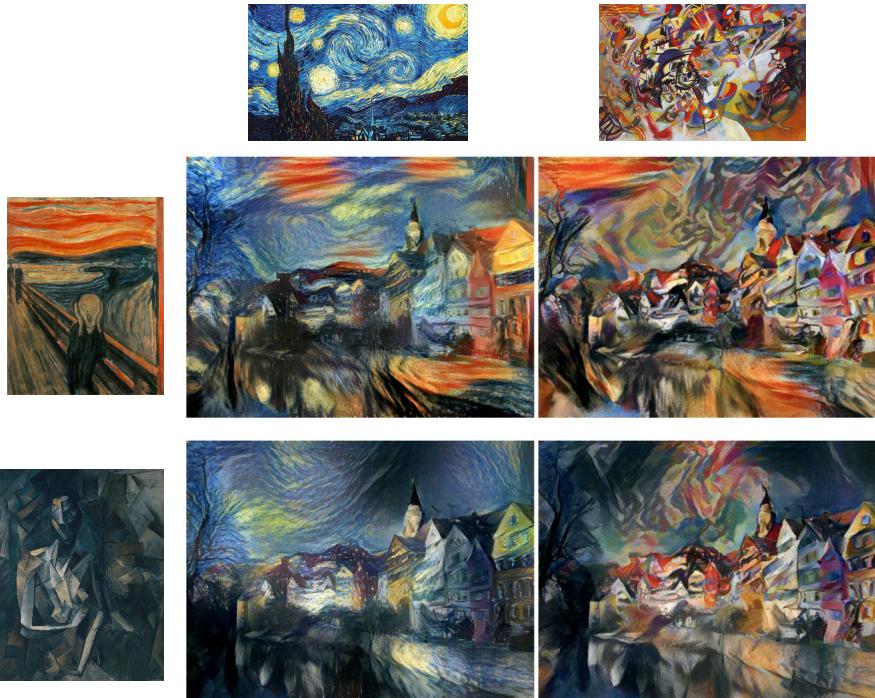


Smaller style  
image

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.





Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 11 - 74 May 10, 2017



Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 11 - 75 May 10, 2017

# Neural Style Transfer

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

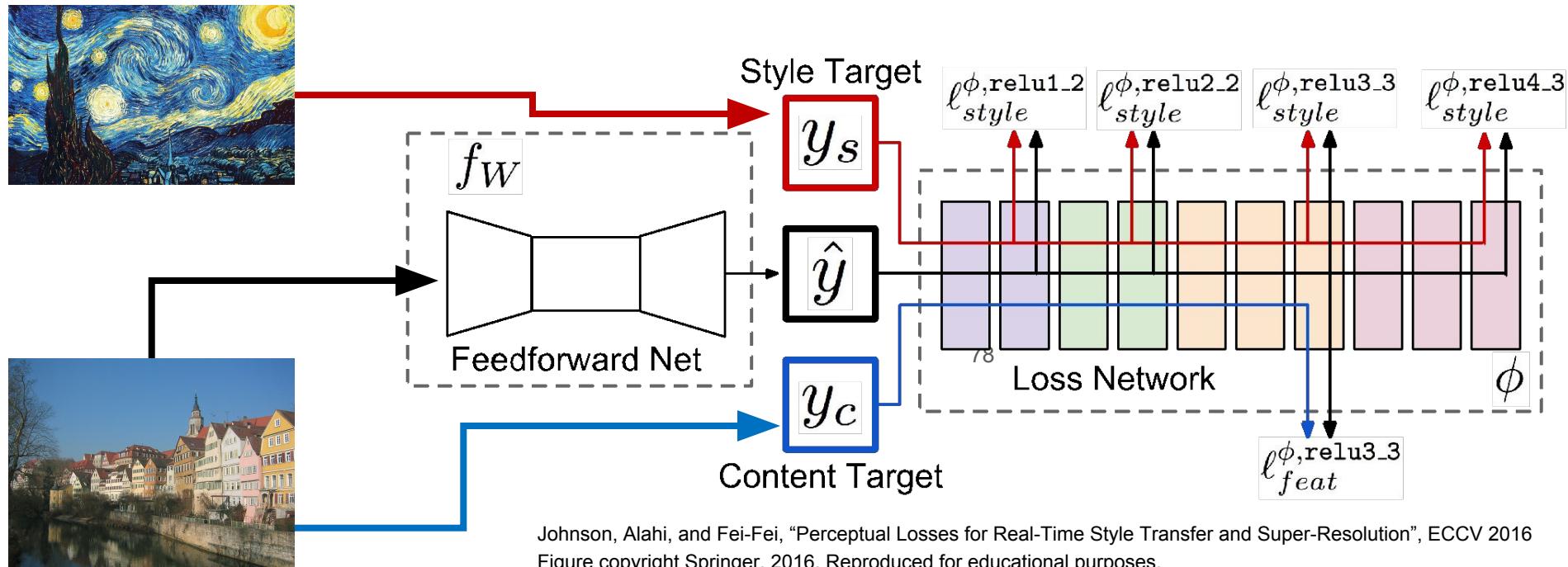
# Neural Style Transfer

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

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

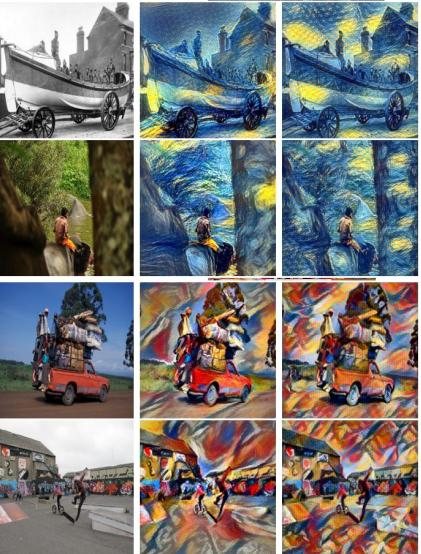
# Fast Style Transfer

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



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

# Fast Style Transfer



Slow

Fast



Slow

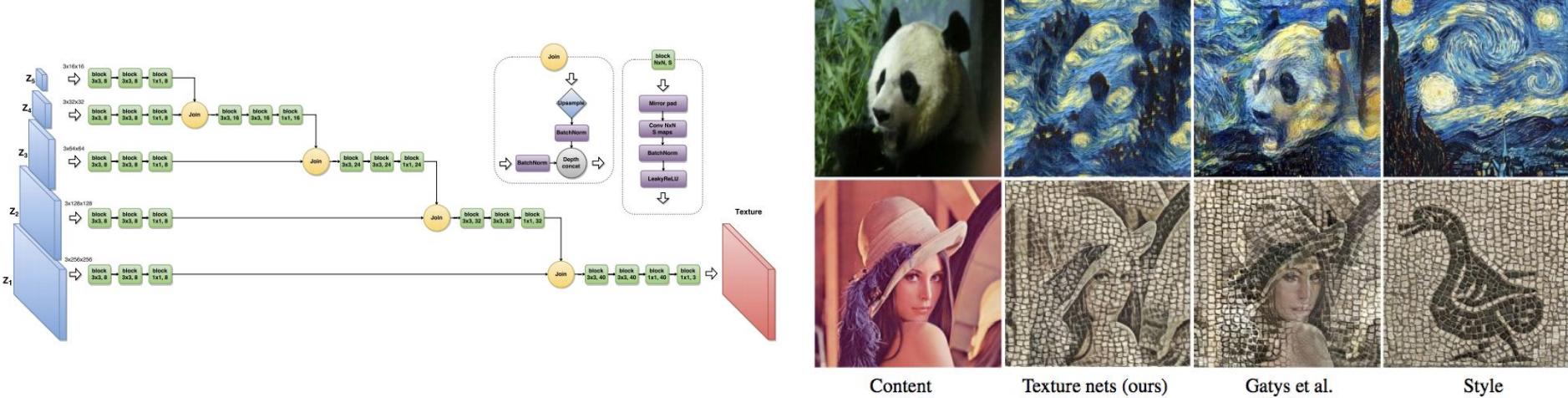
Fast



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

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

# Fast Style Transfer



Concurrent work from Ulyanov et al, comparable results

Ulyanov et al, "Texture Networks: Feed-forward Synthesis of Textures and Stylized Images", ICML 2016

Ulyanov et al, "Instance Normalization: The Missing Ingredient for Fast Stylization", arXiv 2016

Figures copyright Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor Lempitsky, 2016. Reproduced with permission.

# Fast Style Transfer



Replacing batch normalization with Instance Normalization improves results

Ulyanov et al. "Texture Networks: Feed-forward Synthesis of Textures and Stylized Images", ICML 2016

Ulyanov et al. "Instance Normalization: The Missing Ingredient for Fast Stylization", arXiv 2016

Figures copyright Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor Lempitsky, 2016. Reproduced with permission.

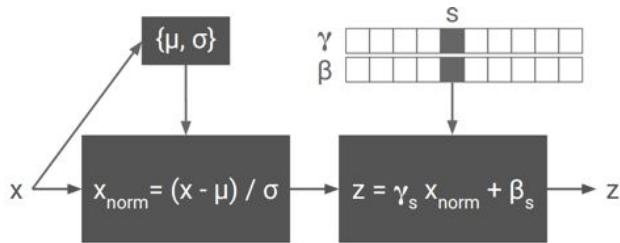
# One Network, Many Styles



Dumoulin, Shlens, and Kudlur, "A Learned Representation for Artistic Style", ICLR 2017.  
Figure copyright Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur, 2016; reproduced with permission.

# One Network, Many Styles

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



Single network can blend styles after training

Dumoulin, Shlens, and Kudlur, "A Learned Representation for Artistic Style", ICLR 2017.  
Figure copyright Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur, 2016; reproduced with permission.

# Summary

Many methods for understanding CNN representations

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

**Gradients:** Saliency maps, class visualization, fooling images, feature inversion

**Fun:** DeepDream, Style Transfer.

Next time: **Unsupervised Learning**  
Autoencoders  
Variational Autoencoders  
Generative Adversarial Networks