Lecture 12:
Visualizing and Understanding

Administrative

Milestones due tonight on Canvas, 11:59pm

Midterm grades released on Gradescope this week

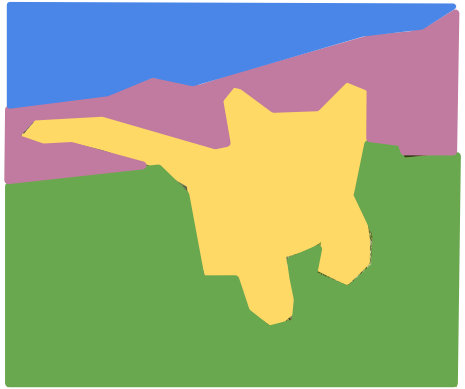
A3 due next Friday, 5/26

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

Poster session is June 6

Last Time: Lots of Computer Vision Tasks

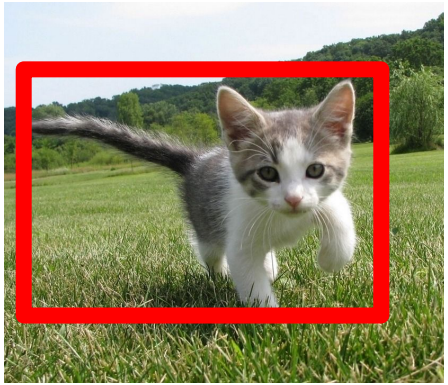
Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

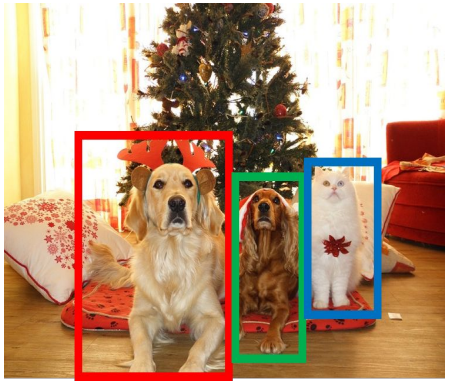
Classification + Localization



CAT

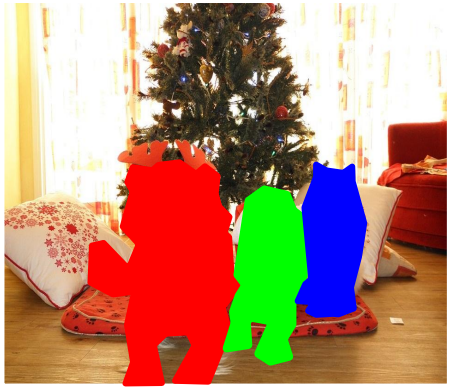
Single Object

Object Detection



DOG, DOG, CAT

Instance Segmentation

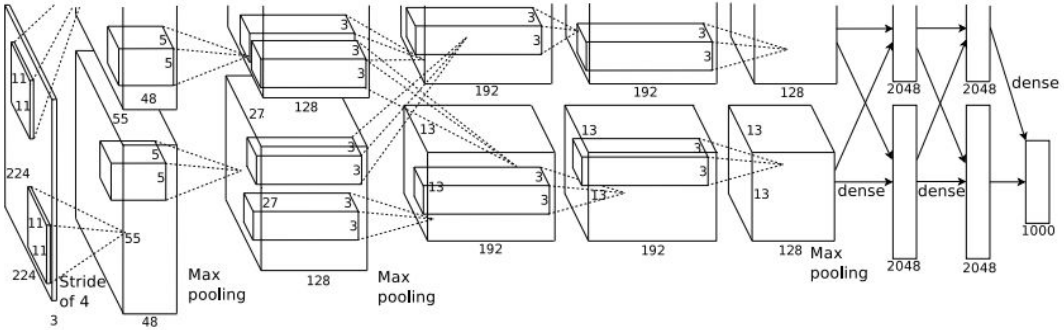


DOG, DOG, CAT

Multiple Object

What's going on inside ConvNets?

여태까지는 ConvNet이 black box로 작동했다.



Input Image:
3 x 224 x 224

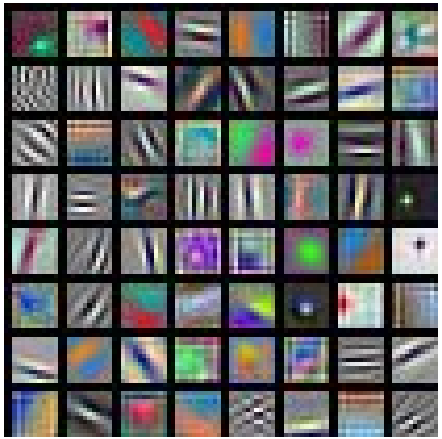
Class Scores:
1000 numbers

What are the intermediate features looking for?

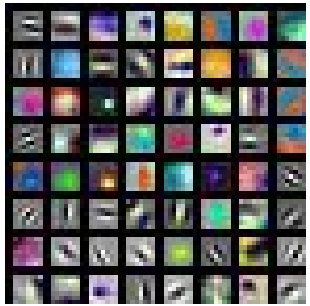
Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012.
Figure reproduced with permission.

First Layer: Visualize Filters

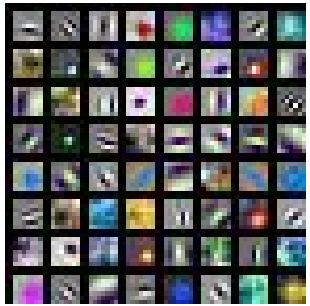
convolutional layer



AlexNet:
64 x 3 x 11 x 11



ResNet-18:
64 x 3 x 7 x 7



ResNet-101:
64 x 3 x 7 x 7

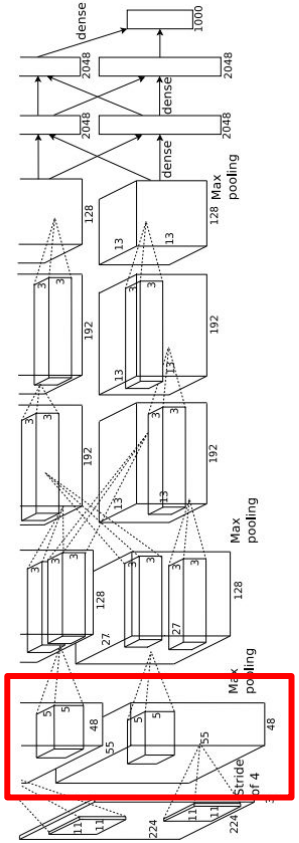
DenseNet-121:
64 x 3 x 7 x 7

첫 번째 Layer
- Convolutional Layer
- 첫 번째 Layer을 시각화 했다.

1) edge(oriented)
2) color(opposing)

Q. 왜 weight를 visualization 하는 것이 filter가 무엇인지 알려주는가.

- inner product의 성질이다.
- maximum이 되는 것은 같은 것끼리 곱하는 것이다.



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

- Hubel이 한 것과 비슷하다.

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:

less interpretable
> 16 channel input
> 바로 image로 나타낼 수 없다.

layer 1 weights

16 x 3 x 7 x 7

Weights:

0 ~ 255 rescale로 했다.

layer 2 weights

20 x 16 x 7 x 7

7x7 16 channel
- gray scale image
- 20 *(16 * (7 x 7))
- input image와 연결된 것이 아니라 first layer와 연결될 수 없다.

Weights:

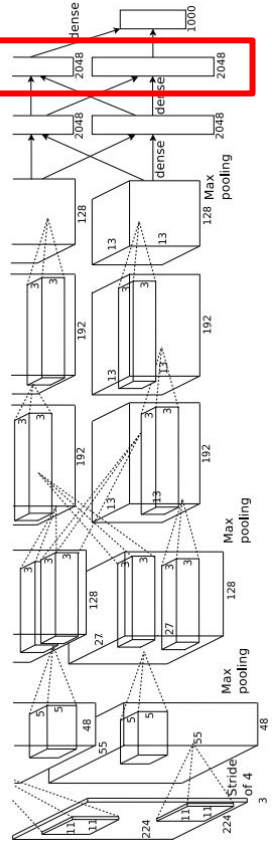
layer 3 weights

20 x 20 x 7 x 7

Last Layer

마지막 layer

FC7 layer



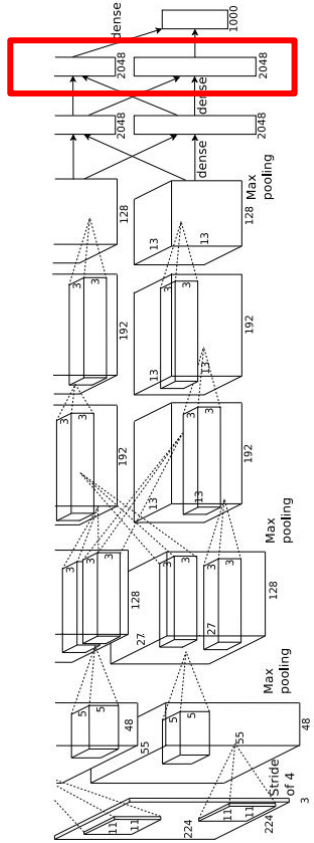
4096-dimensional feature vector for an image (layer immediately before the classifier)

Run the network on many images, collect the feature vectors

Last Layer: Nearest Neighbors

4096개의 feature space에서 nearest neighbor를 계산한다.

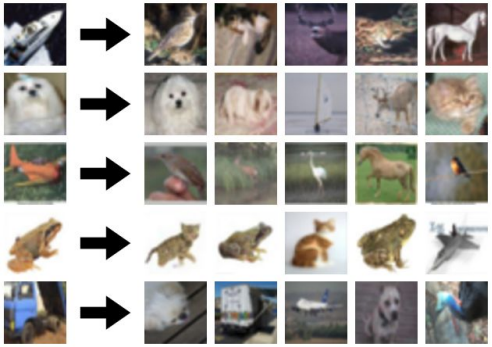
4096-dim vector



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.

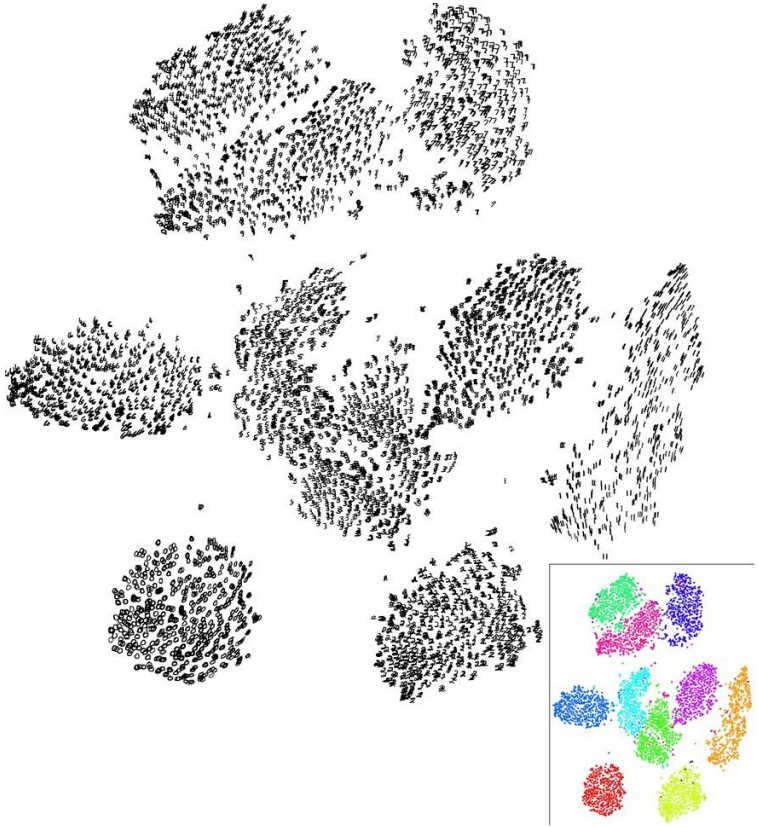
Last Layer: Dimensionality Reduction

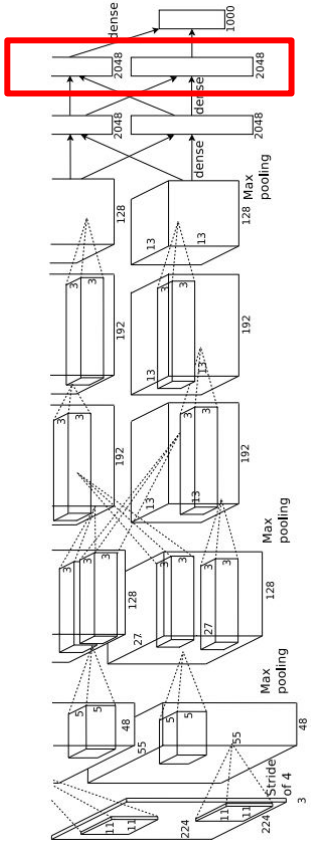
1. dimensionality reduction
2. t-SNE(nonlinear)

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**



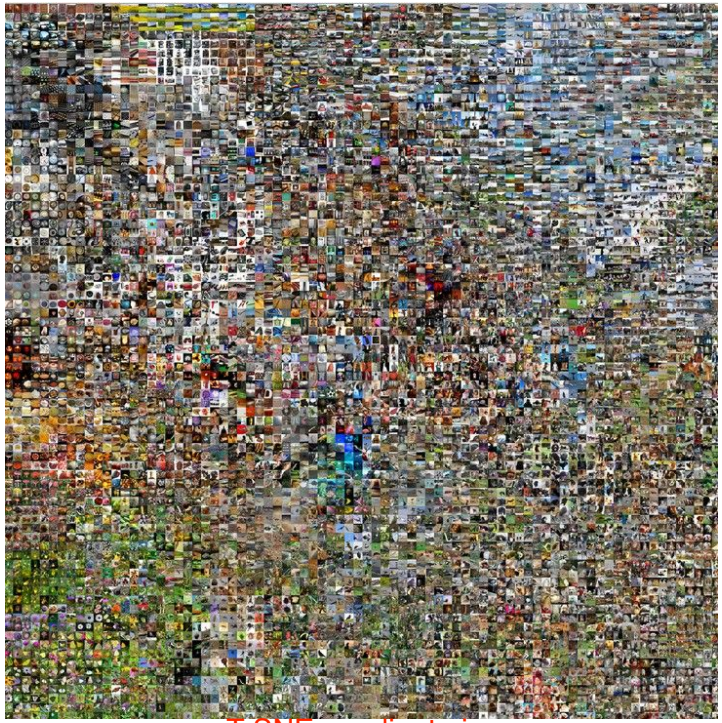


Van der Maaten and Hinton, “Visualizing Data using t-SNE”, JMLR 2008

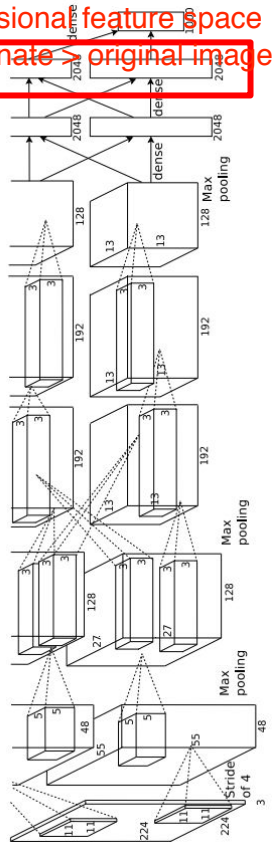
Last Layer: Dimensionality Reduction

T-SNE
4096 > 2 dimensional feature space
4096 > 2 coordinate > original image

original image



T-SNEcoordinate image



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/

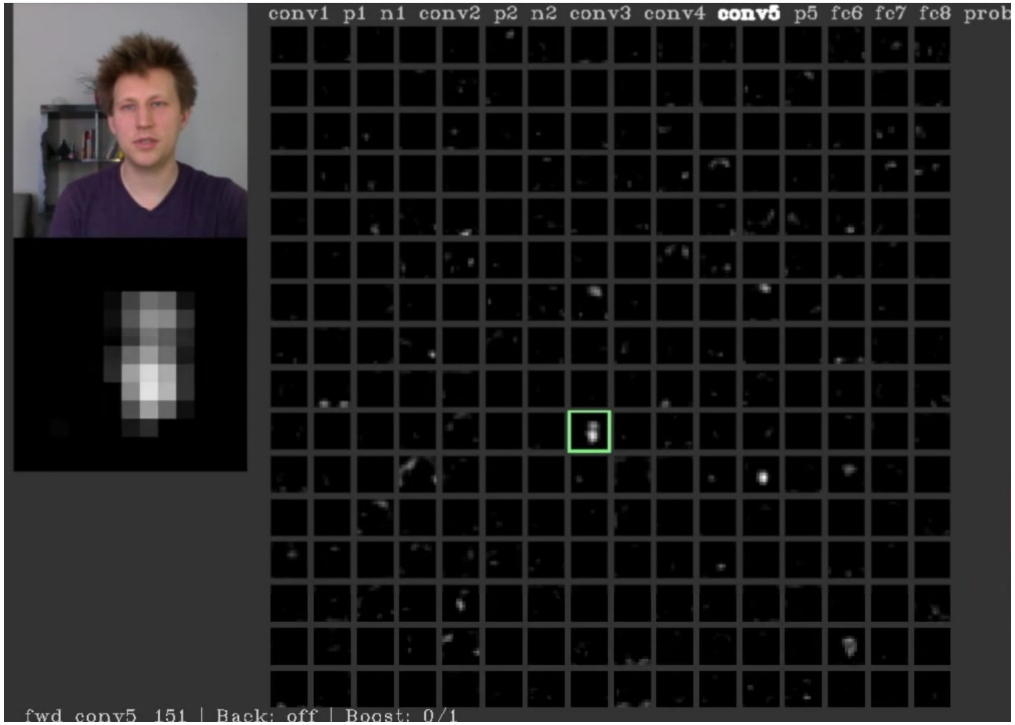
Visualizing Activations

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

128 different 13 x 13 2D grid -> human face

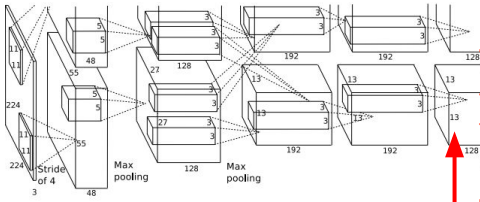
dead relu: 모든 training data 에서 죽는 것이다.

not active for this particular input



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

Maximally Activating Patches



conv5
- pick 17 channel

- 계속 데이터를 흘려보고 가장 activation이 큰 이미지를 본다.

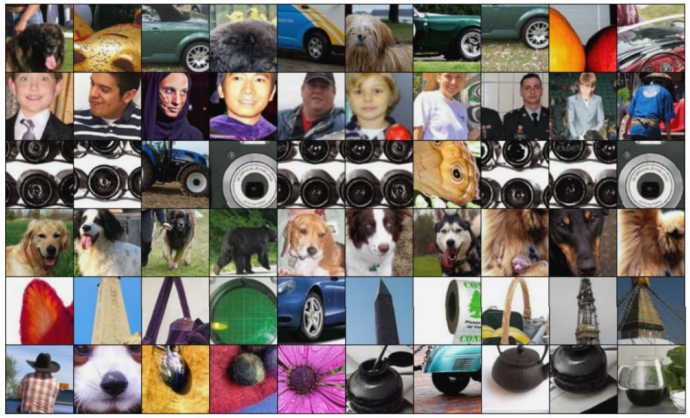
- 각각의 이미지는 전체가 아니라 subset of image를 본다.

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

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

Visualize image patches that correspond to maximal activations

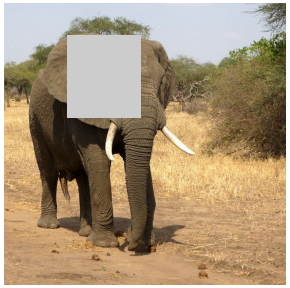
높은 layer은 더 넓은 receptive field를 가지기 때문에 더 전반적인 특성을 잡아낸다.

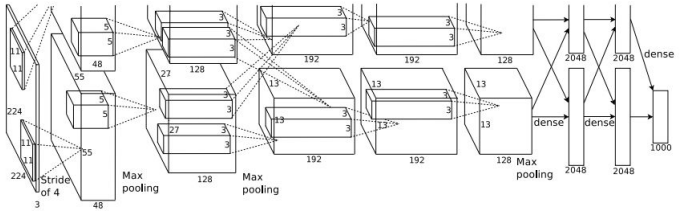


Springenberg et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015
Figure copyright Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, 2015; reproduced with permission.

Occlusion Experiments

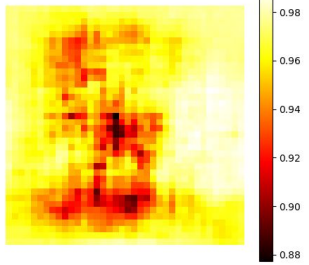
Mask part of the image before feeding to CNN, draw heatmap of probability at each mask location



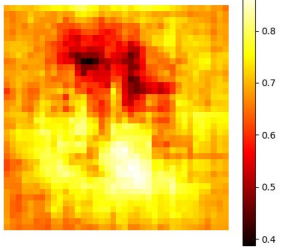


mask를 움직이면서 probability 변화를 본다.
> block out을 하면 probability가 많이 변하는 것은 중요한 feature라는 것이다.

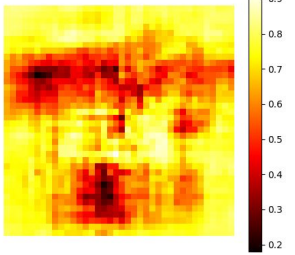
schooner



African elephant, Loxodonta africana



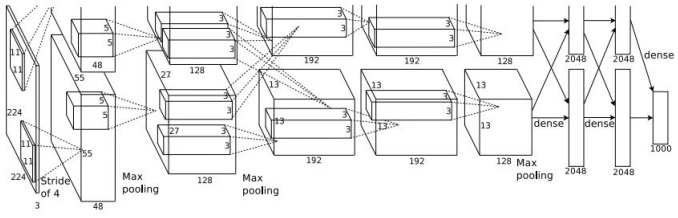
go-kart



Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

Saliency Maps

How to tell which pixels matter for classification?

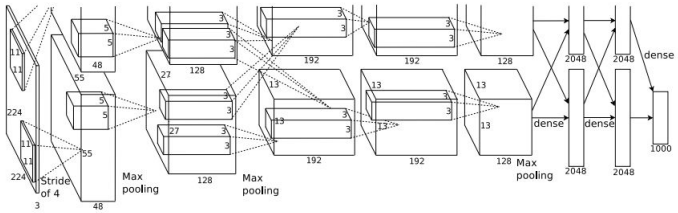


Dog

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

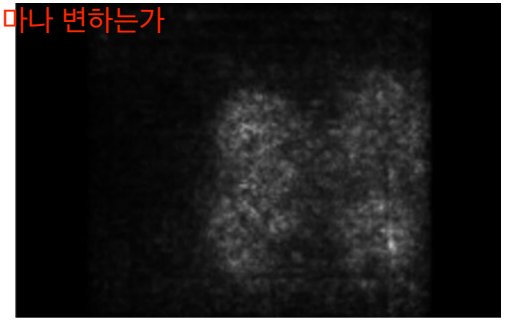
How to tell which pixels matter for classification?



Dog

Input image의 pixel에 관해서 기울기를 바탕으로 한다.
> 각각의 input image의 pixel을 wiggle하면 classification score이 얼마나 변하는가
> which pixel important for the class.

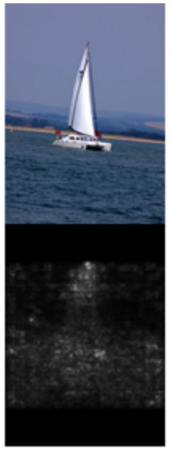
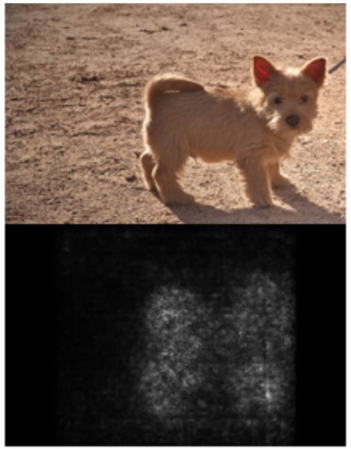
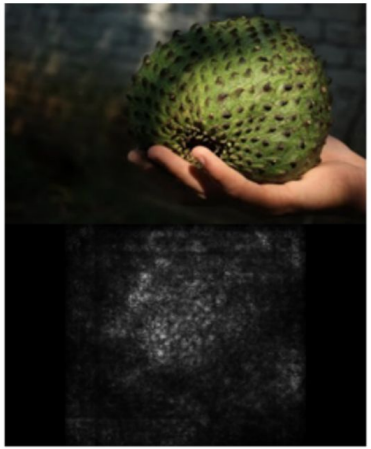
Compute gradient of (unnormalized) class score with respect to image pixels, take absolute value and max over RGB channels

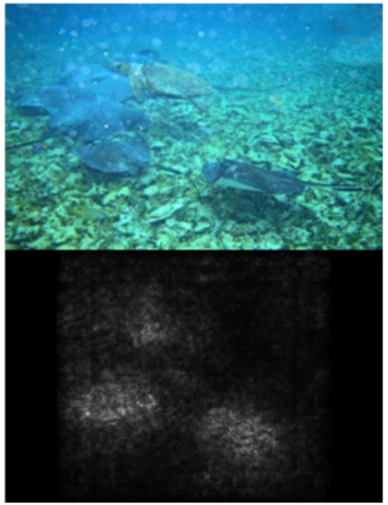


Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.
Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.

Saliency Maps

semantic segmentation에 Saliency map으로 사용할 수 있다.

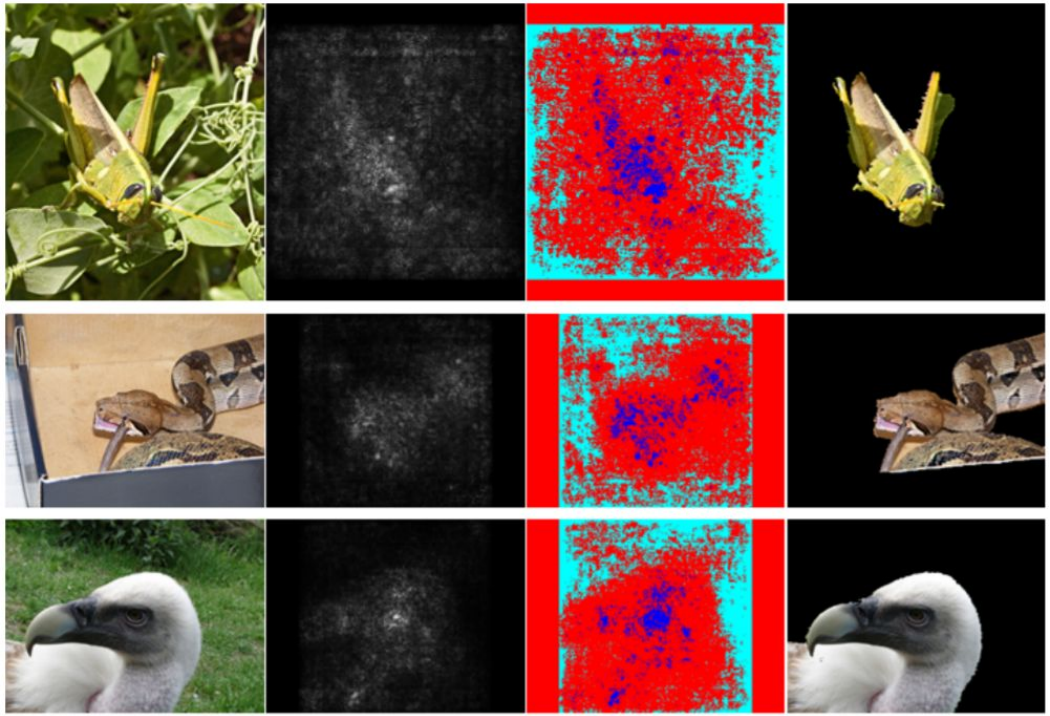






Simonyan, Vedaldi, and Zisserman, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”, ICLR Workshop 2014.
Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.

Saliency Maps: Segmentation without supervision

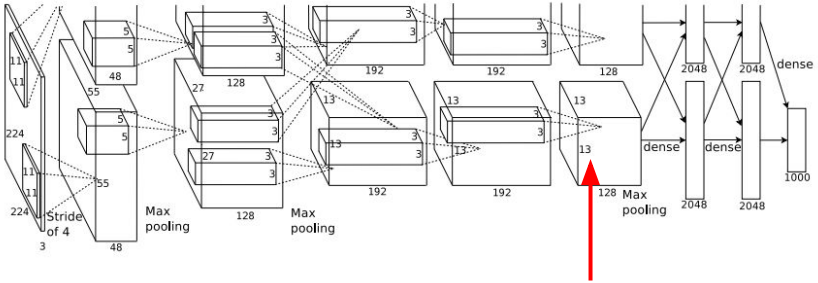
Use GrabCut on saliency map



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

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

Intermediate Features via (guided) backprop



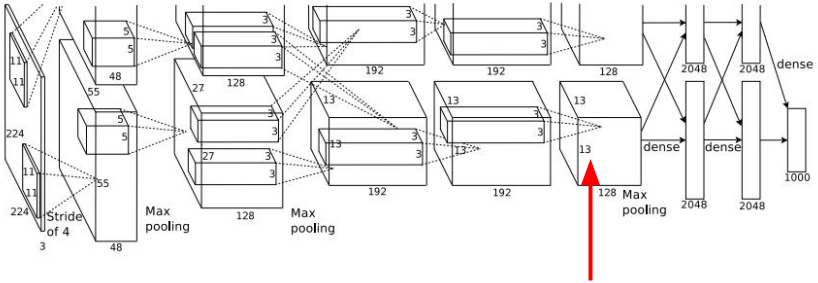
Pick a single intermediate neuron, e.g. one value in 128 x 13 x 13 conv5 feature map

Compute gradient of neuron value with respect to image pixels

어떤 input image part가 중요한가
> rather than computing the gradient for the class score
> 중간의 뉴런에 대해서 gradient를 구할 수 있다.

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

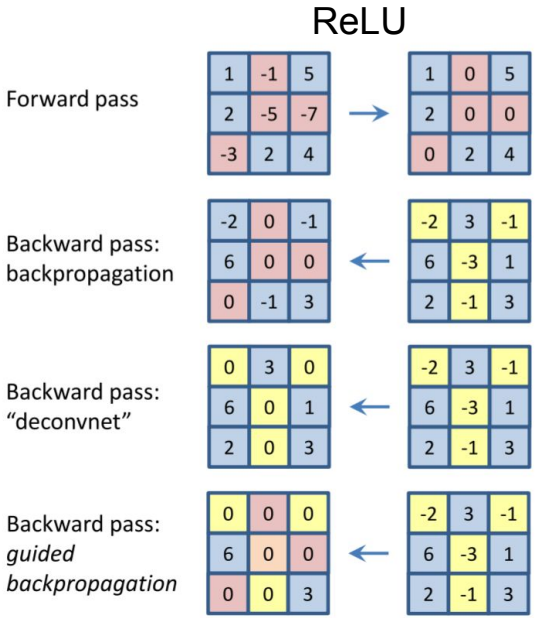
Intermediate features via (guided) backprop 질문



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

Compute gradient of neuron value with respect to image pixels

오직 backpropagate through positive relu
> positive influences



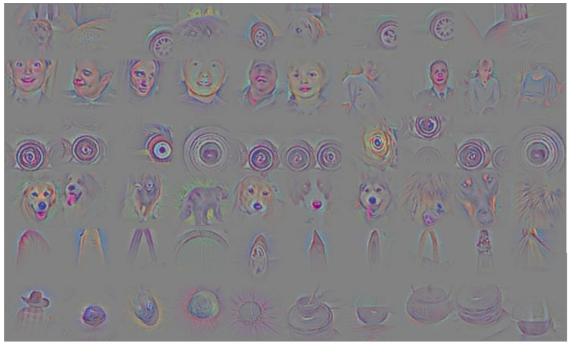
Images come out nicer if you only backprop positive gradients through each ReLU (guided backprop)

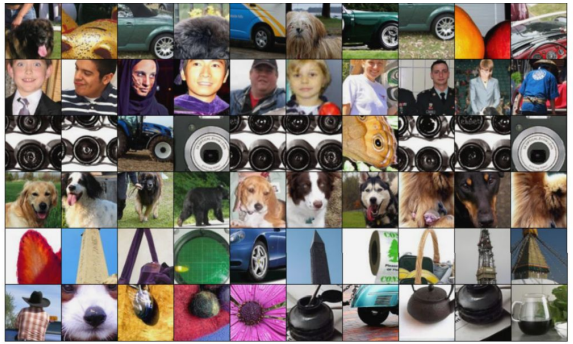
Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014
Springenberg et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015

Intermediate features via (guided) backprop

guided backpropagation
> 더 잘 보이게 한다.

input patch part가 영향을 주는가
> maximally activate patch
> guided backpropagation





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

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

1) fix the weights of our network.

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

regularization

2) synthesize image by performing gradient ascent on the pixels of the image to try and maximize score of some intermediate neuron or of some class.

- Gradient Ascent
- Change of pixel for image to maximize nueron or class score

Neuron value

Natural image regularizer

look naturally

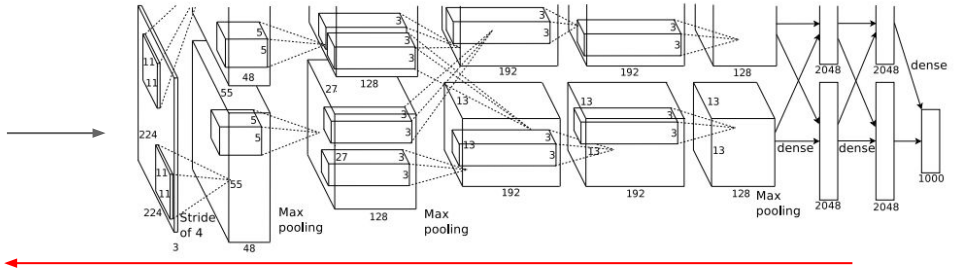
Visualizing CNN features: Gradient Ascent

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

score for class c (before Softmax)

1. Initialize image to zeros

zero image



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

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

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.

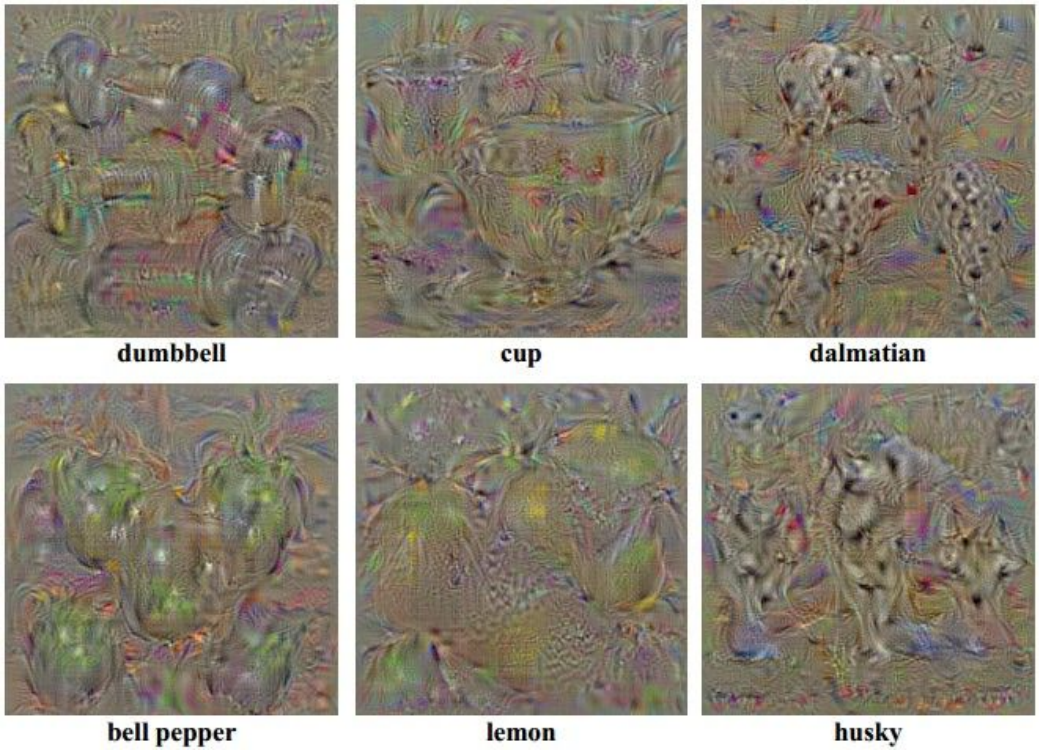
Visualizing CNN features: Gradient Ascent

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

regularizer

Simple regularizer: Penalize L2 norm of generated image

target: dumbell



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

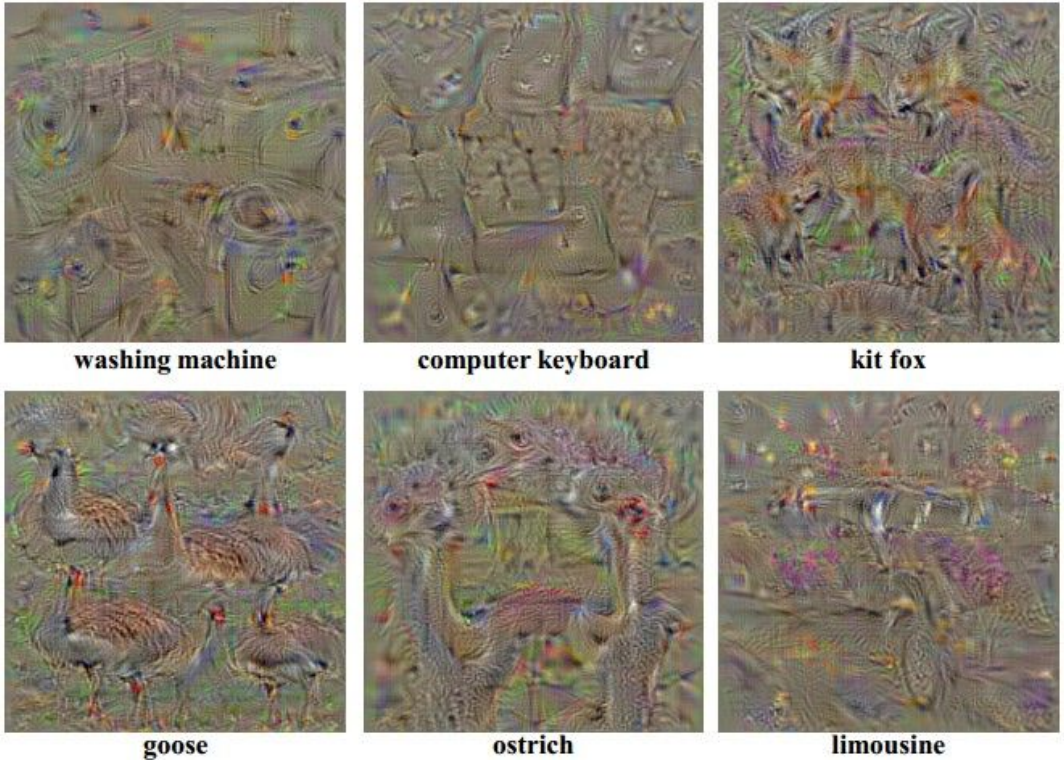
Visualizing CNN features: Gradient Ascent

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

Simple regularizer: Penalize L2 norm of generated image

Q1. why rainbow color?
> 보통 image는 [0, 255]
> 여기에서는 unconstrained
> 그냥 심각하게 볼 건 없다.

Q2. regularizer 없다면
> maximize score.
> doesn't look like 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

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

Visualizing CNN features: Gradient Ascent

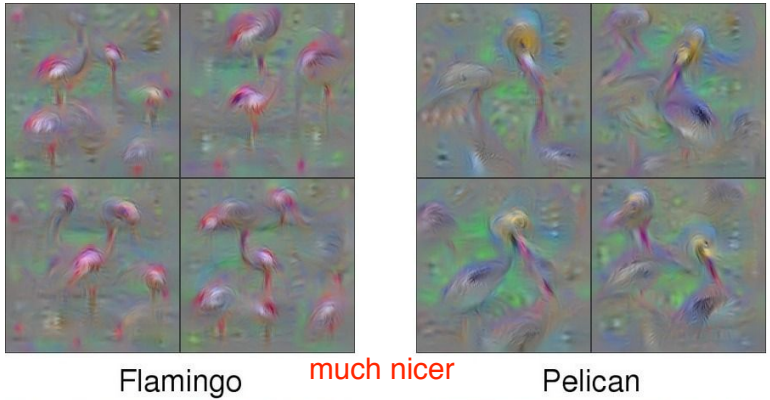
random initializer

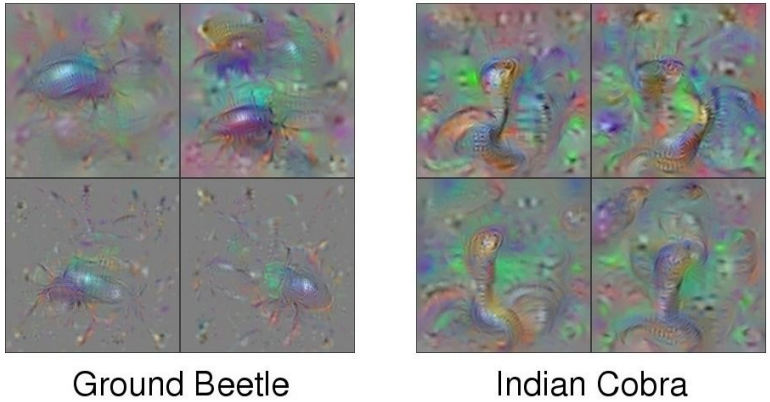
$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

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

periodically

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





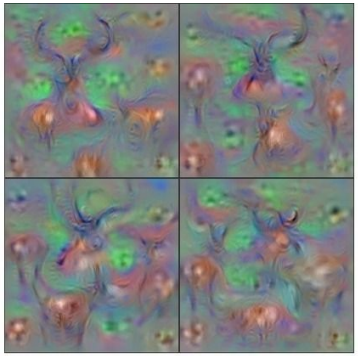
Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.
Figure copyright Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson, 2014. Reproduced with permission.

Visualizing CNN features: Gradient Ascent

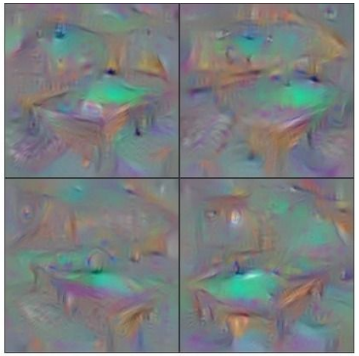
$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

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

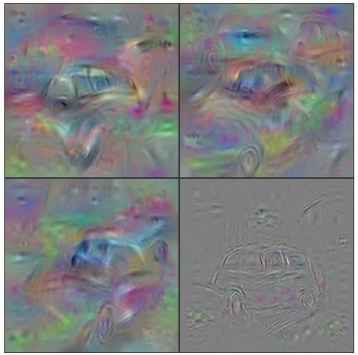
(1) Gaussian blur image
(2) Clip pixels with small values to 0
(3) Clip pixels with small gradients to 0



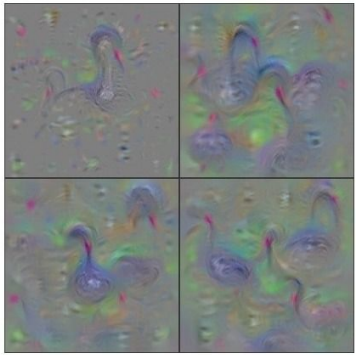
Hartebeest



Billiard Table



Station Wagon



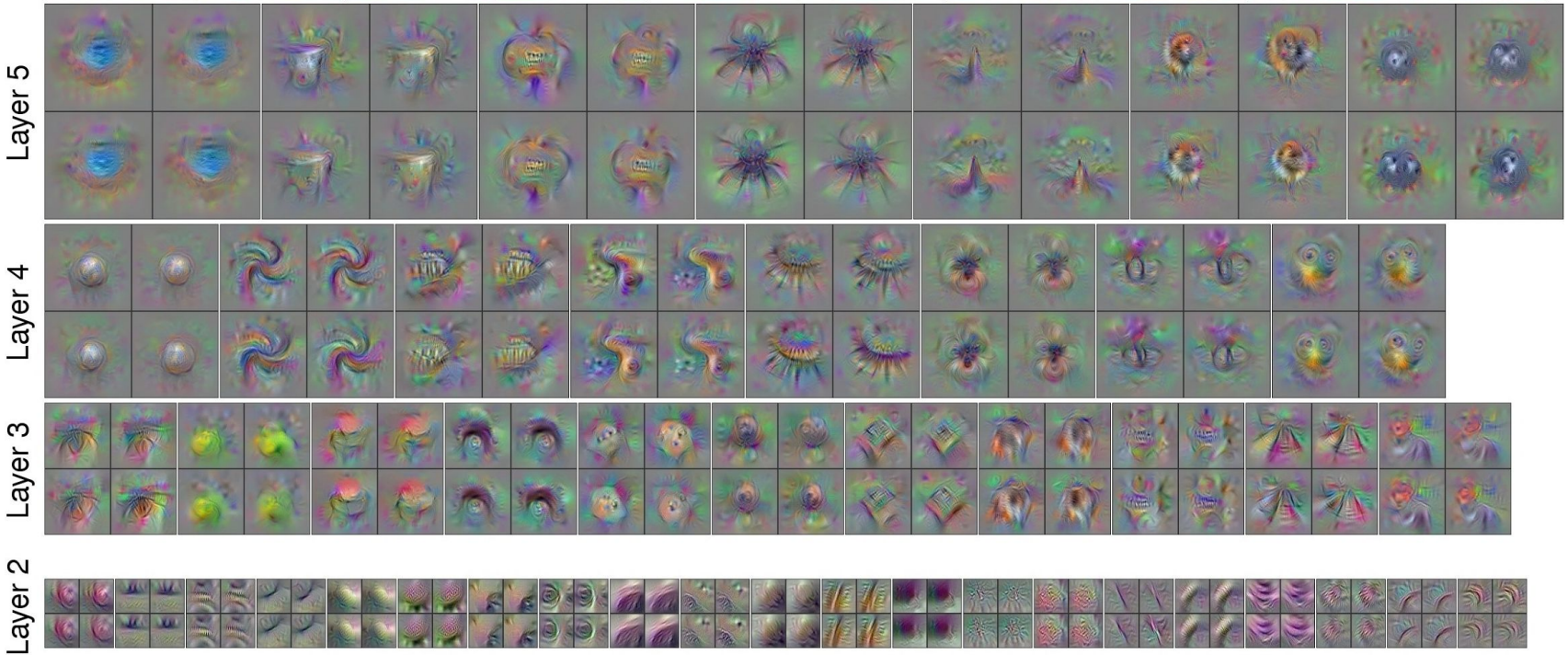
Black Swan

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

Visualizing CNN features: Gradient Ascent

intermediate layer score로 할 수도 있다.

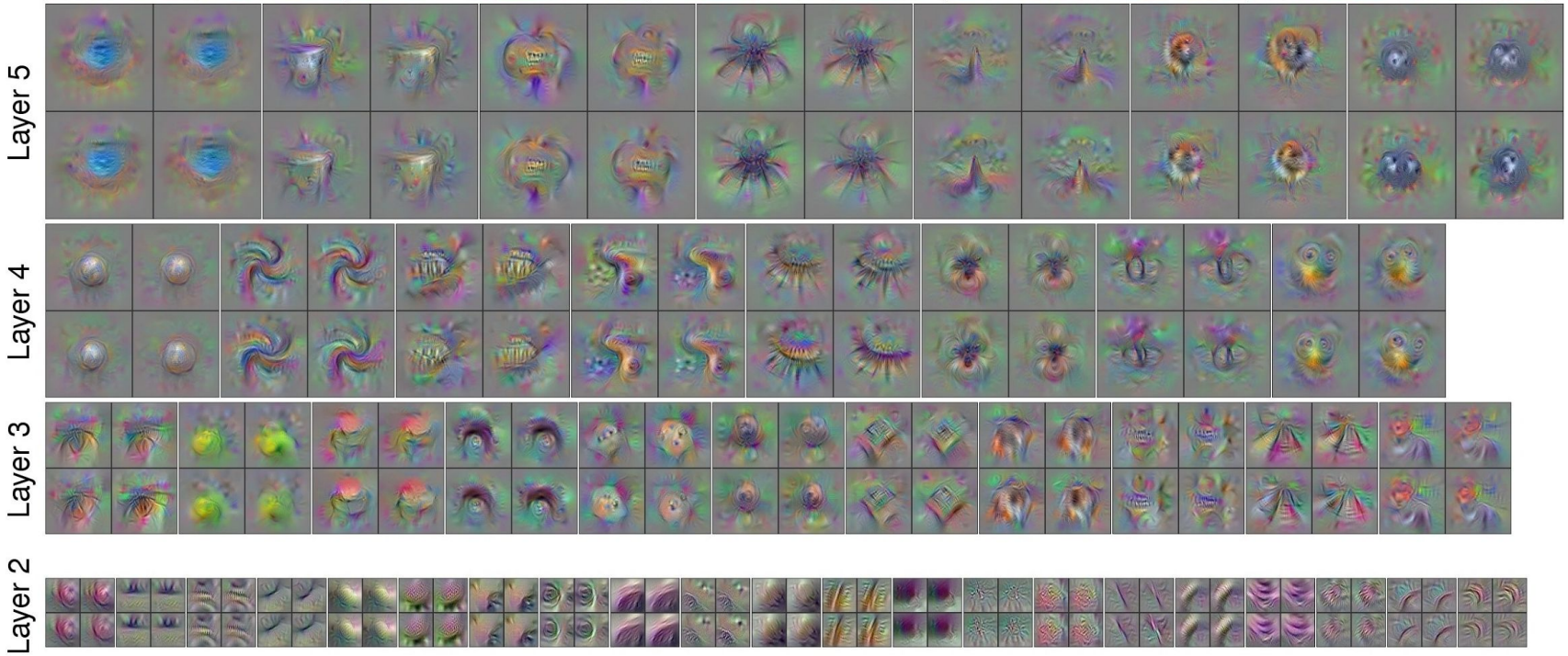
Use the same approach to visualize intermediate features



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)

clustering algorithm
> different mode

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

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

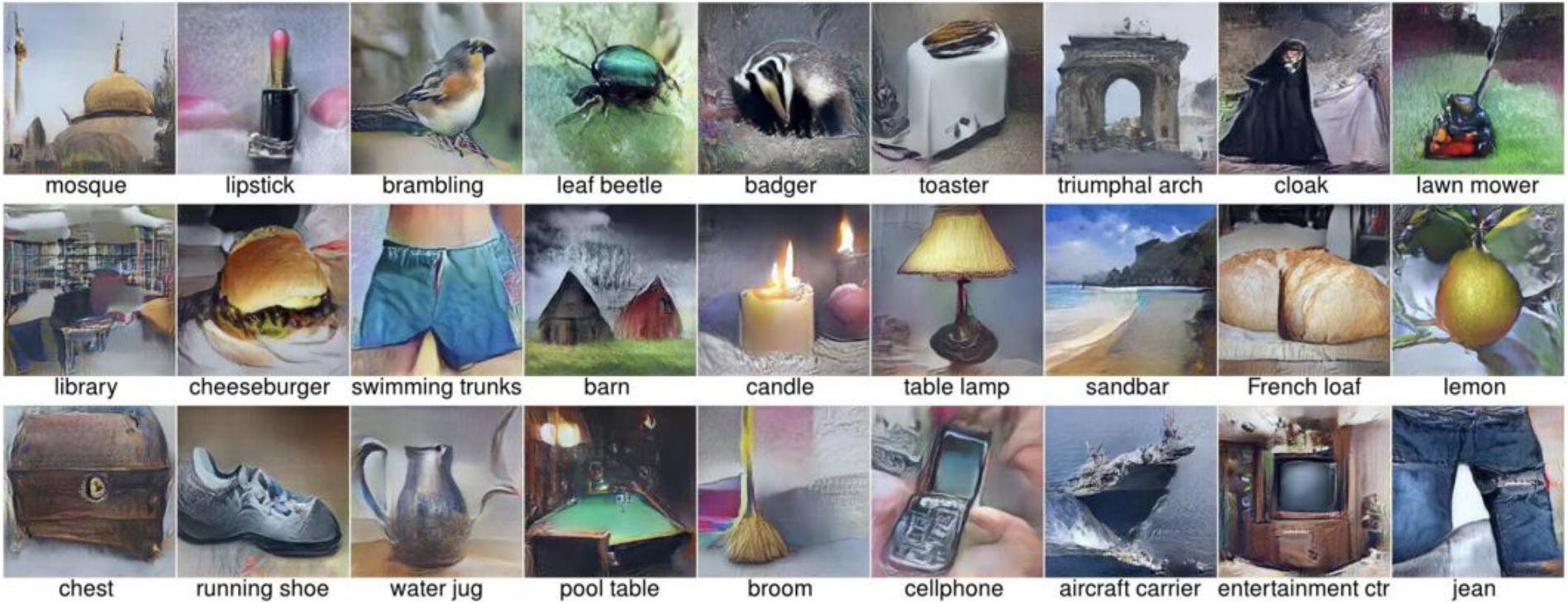
Nguyen et al, “Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks”, ICML Visualization for Deep Learning Workshop 2016.

Visualizing CNN features: Gradient Ascent

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

Visualizing CNN features: Gradient Ascent

Optimize in FC6 latent space instead of pixel space:



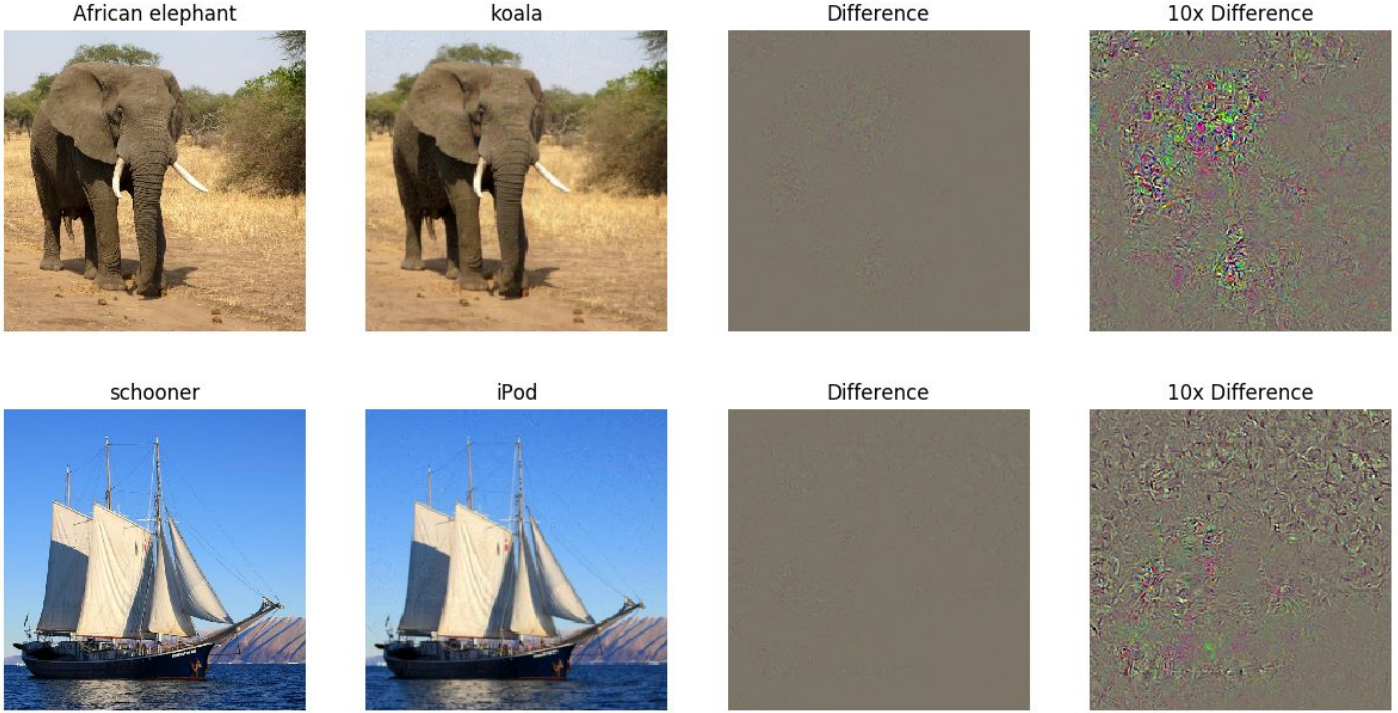
Nguyen et al, "Synthesizing the preferred inputs for neurons in neural networks via deep generator networks," NIPS 2016

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

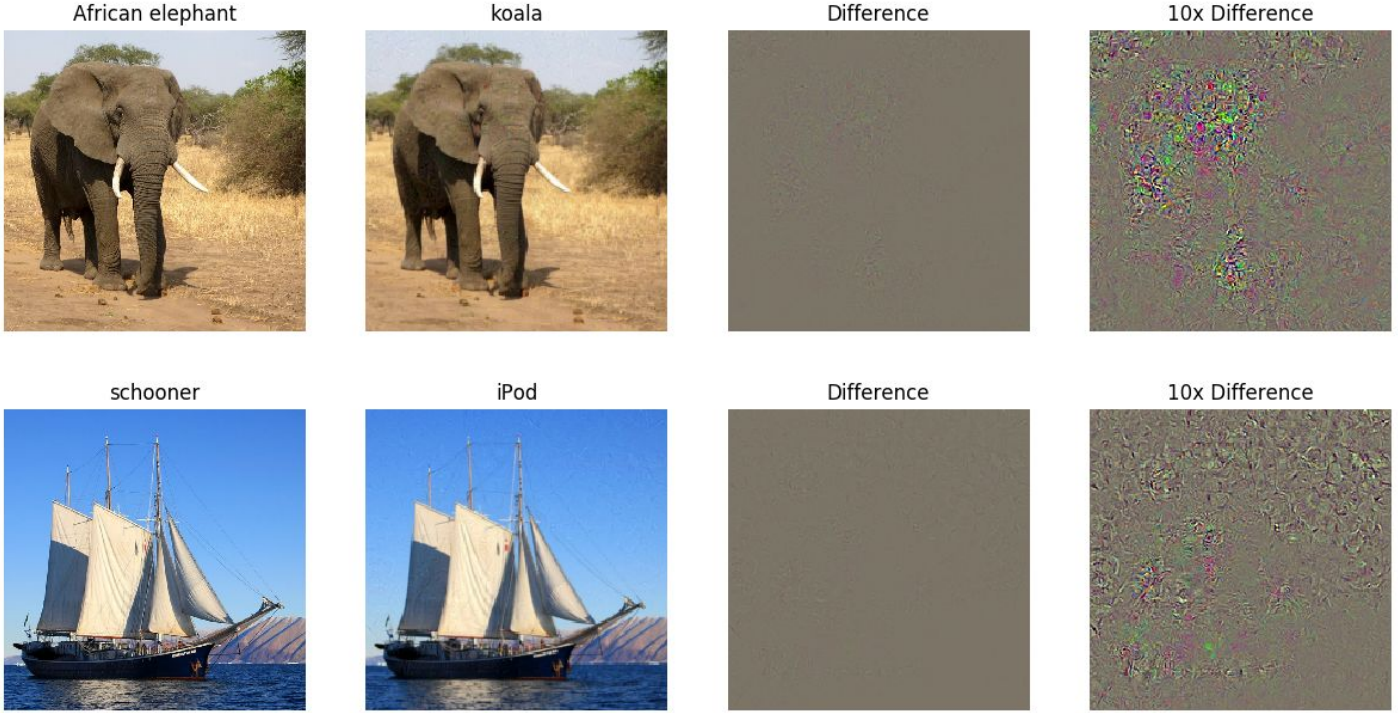
change elephant image into koala bear

Fooling Images / Adversarial Examples



common criticism > visualization can be good understanding > interesting interpretable things

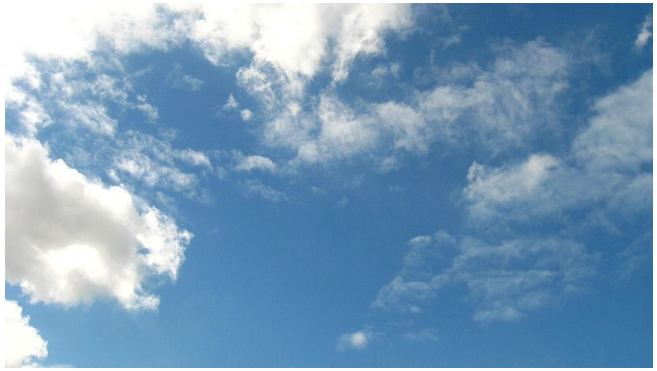
Fooling Images / Adversarial Examples

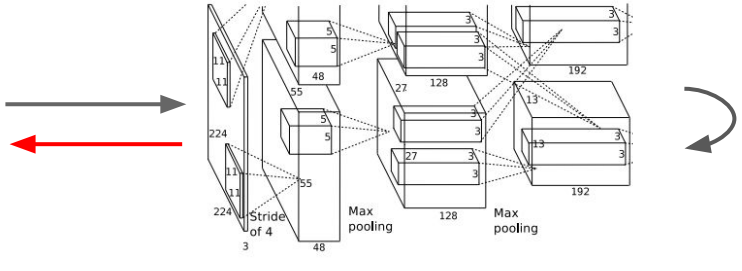


What is going on? Ian Goodfellow will explain

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





질문

Choose an image and a layer in a CNN; repeat: up to some layer

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

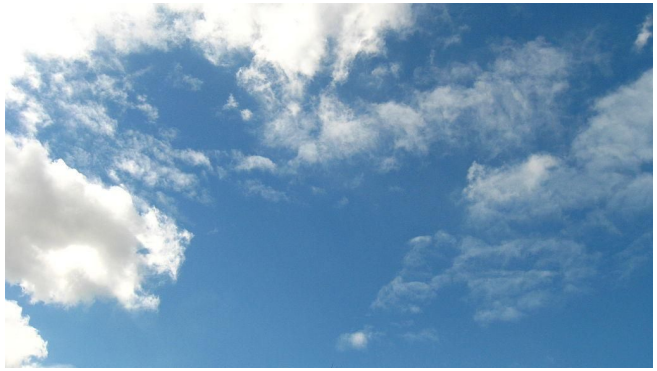
backpropagate
> gradient value = activation

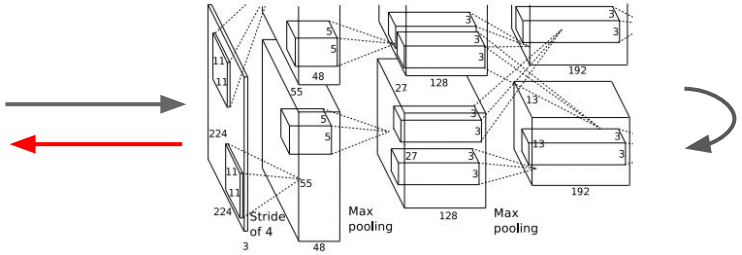
amplify existing feature
> gradient equal to the feature

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:

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

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)

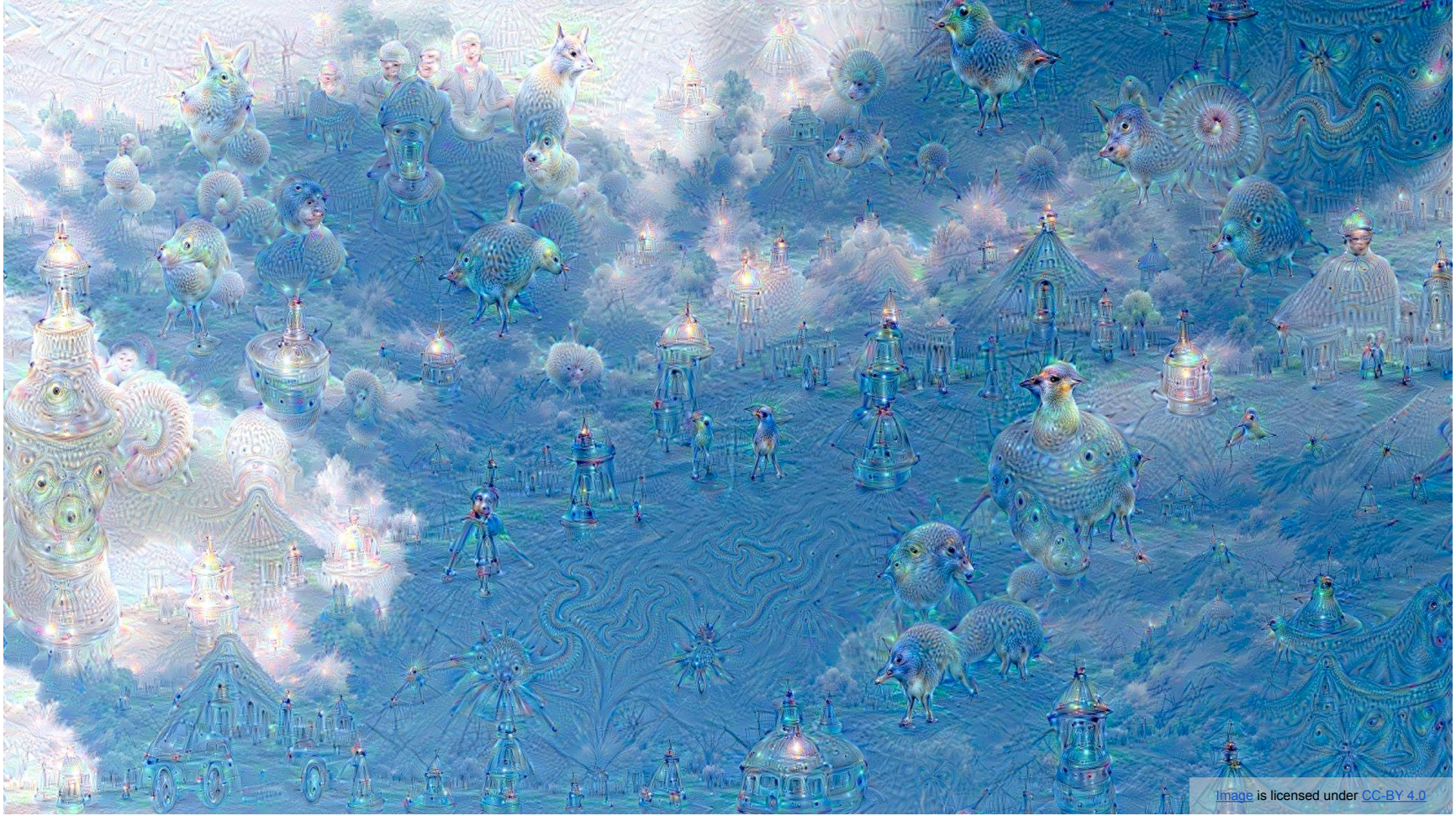
Jitter image

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



high layer

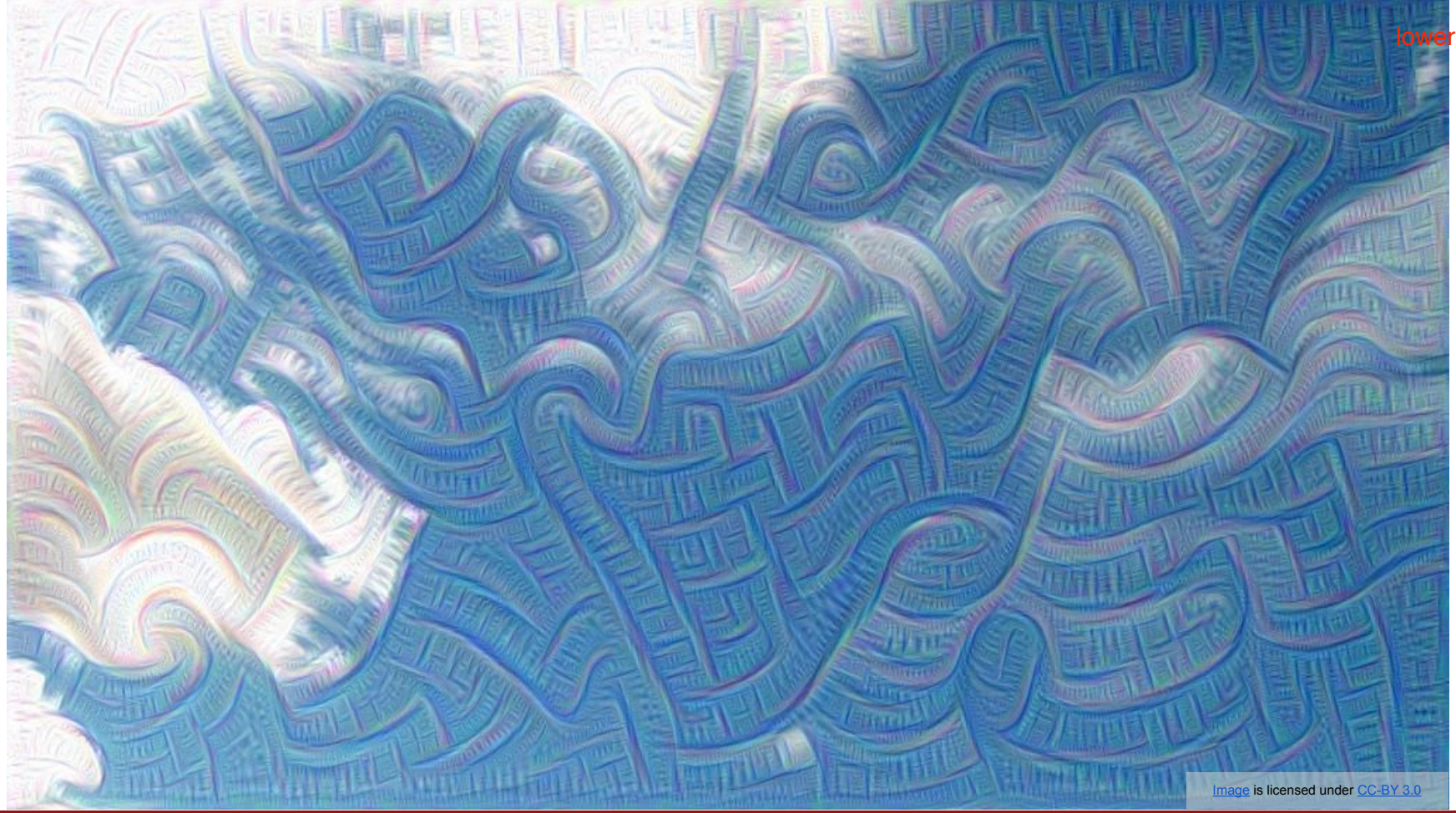
"Admiral Dog!"

"The Pig-Snail"

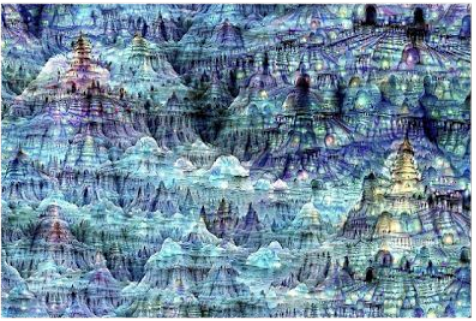
"The Camel-Bird"

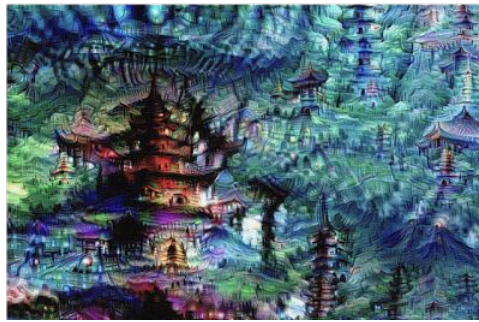
"The Dog-Fish"

200 hundreds of. category is. dog







Feature Inversion

feature representation > reconstructed image

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

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

image > network > record feature value > reconstruction > image
(what is in the feature vector?)

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

Given feature vector

Features of new image

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

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

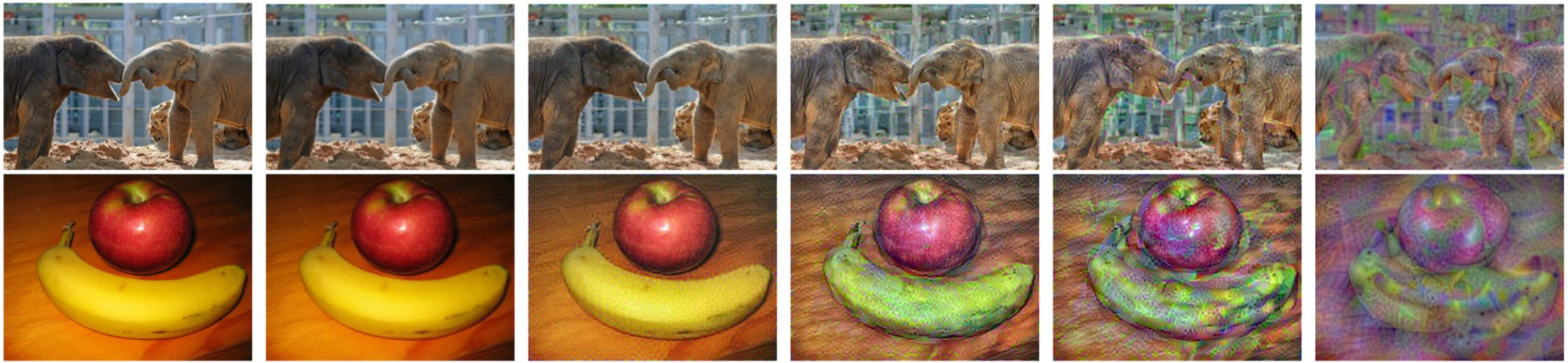
Total Variation regularizer
(encourages spatial smoothness)

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

Feature Inversion

Reconstructing from different layers of VGG-16

original image



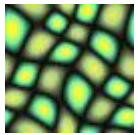
stored
> we're not throwing

deeper layer
> higher layer
> network로 갈수록 pixel 정보보다는
> semantic 정보를 유지한다.

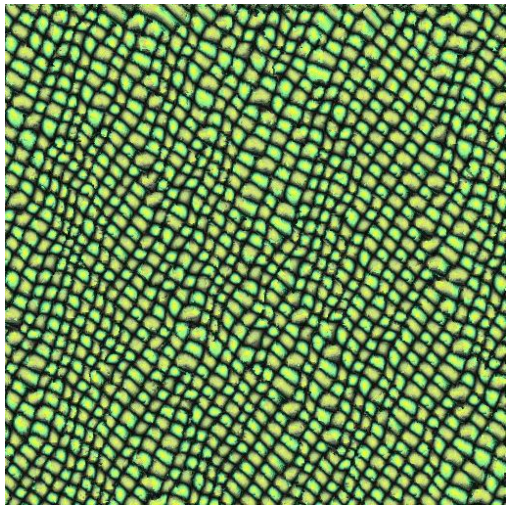
Mahendran and Vedaldi, "Understanding Deep Image Representations by Inverting Them", CVPR 2015
Figure from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016. Copyright Springer, 2016.
Reproduced for educational purposes.

Texture Synthesis

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

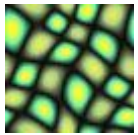


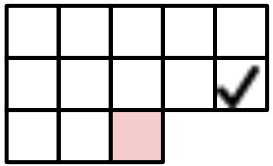
Input



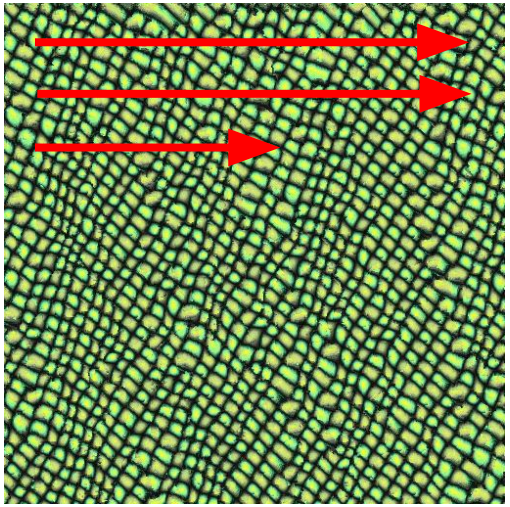
Output

Texture Synthesis: Nearest Neighbor





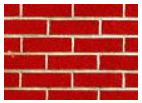
Generate pixels one at a time in scanline order; form neighborhood of already generated pixels and copy nearest neighbor from input

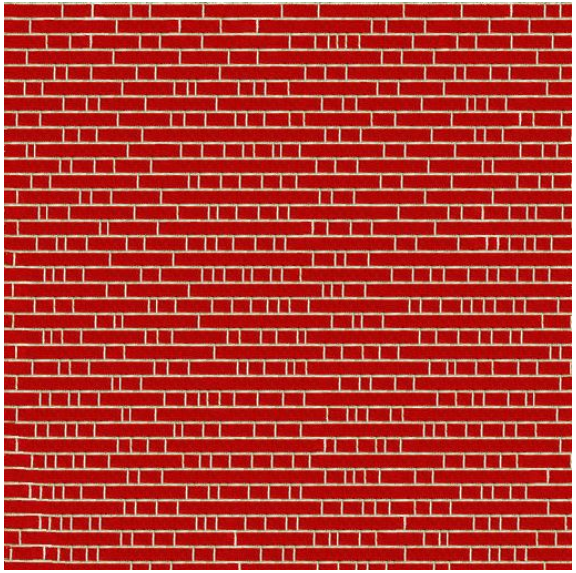


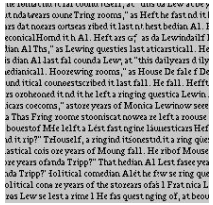
Wei and Levoy, “Fast Texture Synthesis using Tree-structured Vector Quantization”, SIGGRAPH 2000
Efros and Leung, “Texture Synthesis by Non-parametric Sampling”, ICCV 1999

Texture Synthesis: Nearest Neighbor

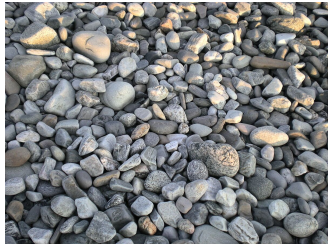
complex texture
> neural network



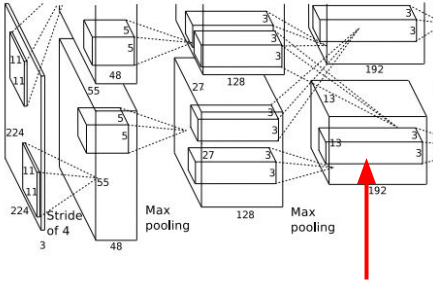


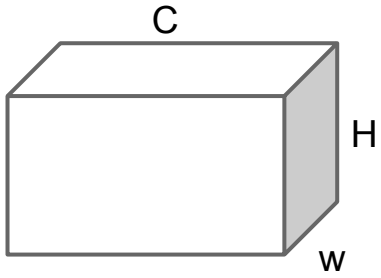


Neural Texture Synthesis: Gram Matrix



This image is in the public domain.



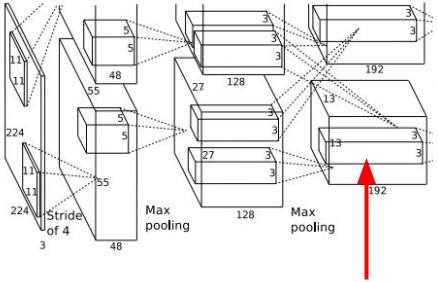


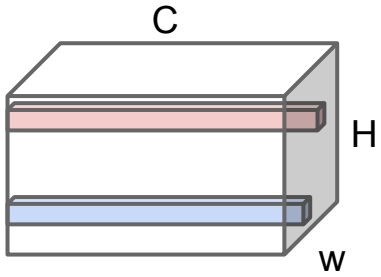
Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

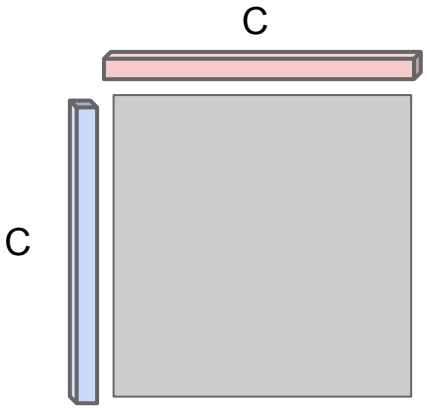
Neural Texture Synthesis: Gram Matrix

input texture > cnn > convolutional feature > H * W * C(channel) feature

This image is in the public domain.





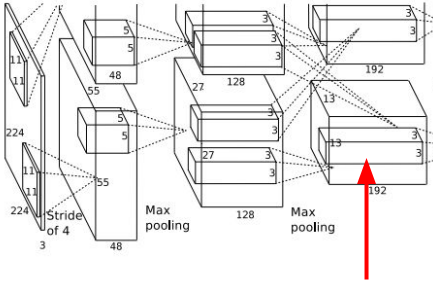


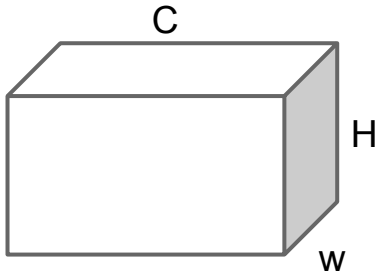
Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

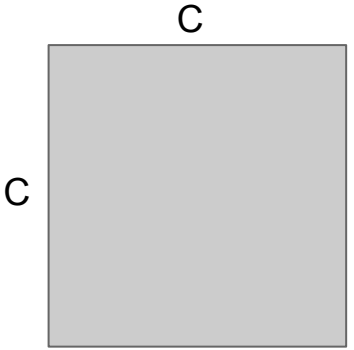
Outer product of two C-dimensional vectors gives C x C matrix measuring co-occurrence

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

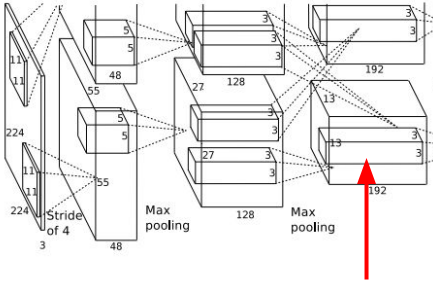
Outer product of two C-dimensional vectors gives C x C matrix measuring co-occurrence

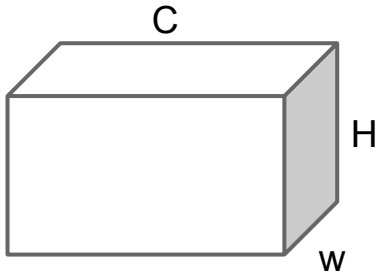
Average over all HW pairs of vectors, giving **Gram matrix** of shape C x C

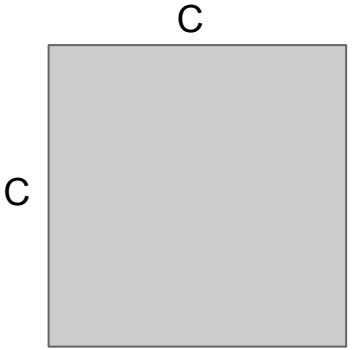
gram matrix라고 하는 것은 feature map 의 channel간 correlation 정보를 가지고 있다.

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

공간적인 정보는 잃는다.

Efficient to compute; reshape features from

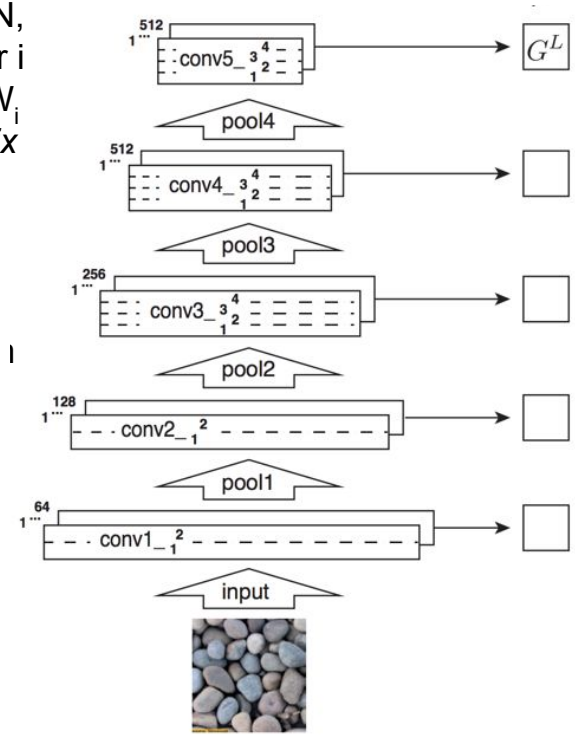
C x H x W to =C x HW

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 $C_i \times C_i$)



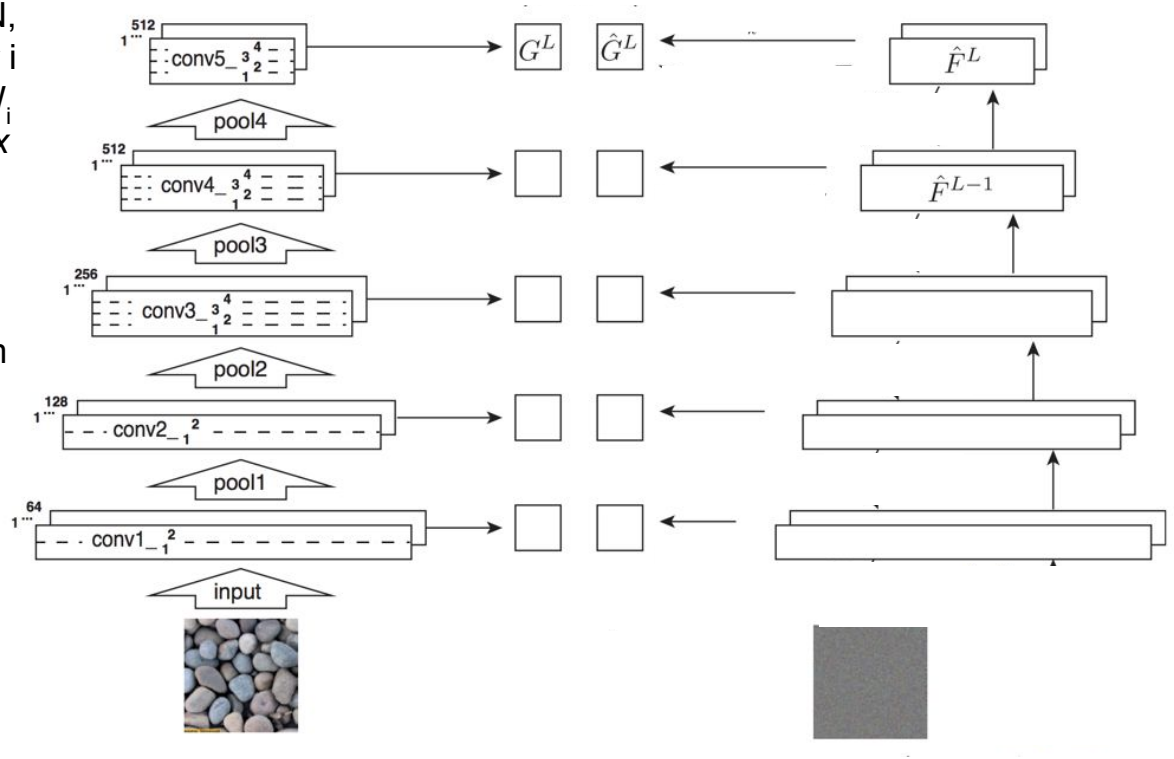
Gatys, Ecker, and Bethge, "Texture Synthesis Using Convolutional Neural Networks", NIPS 2015

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:

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

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

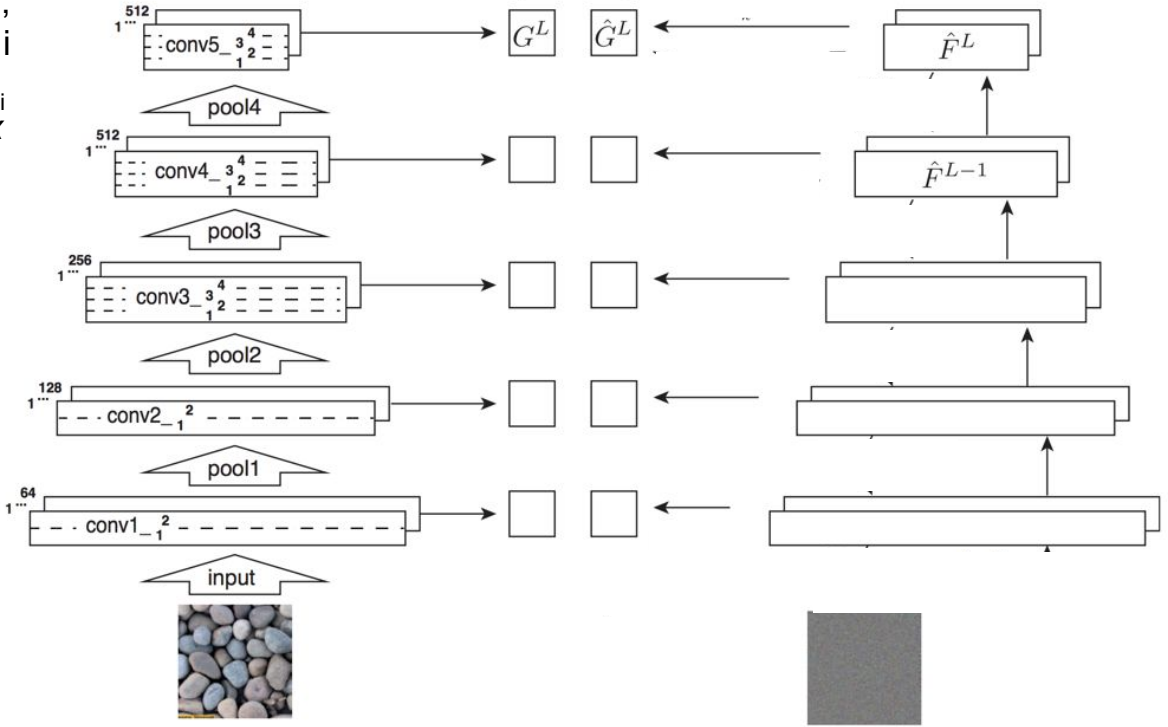
$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{ij} \left(G_{ij}^l - \hat{G}_{ij}^l\right)^2 \qquad \mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^{L} w_l E_l$$

gram matrix를 작게 한다는 것은 각각의 feature map 간의 관계를 비슷하게 만들어준다는 의미이다.

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:

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

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.

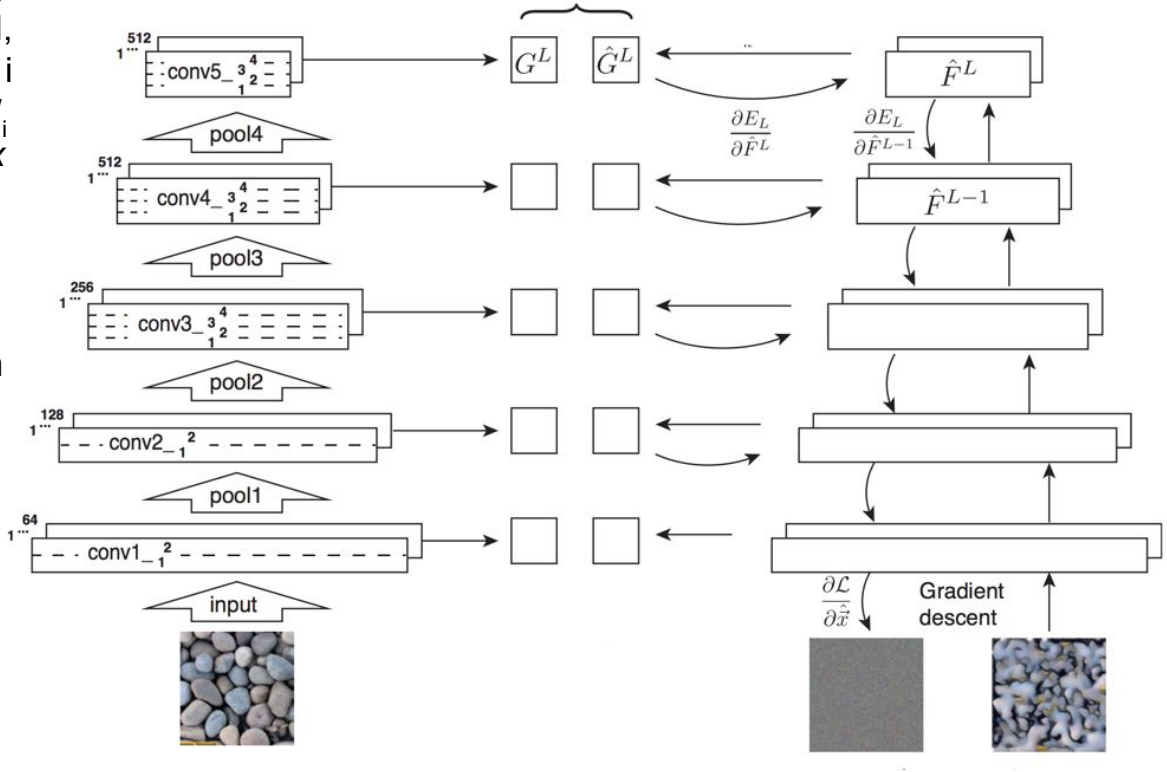
Neural Texture Synthesis

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

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:

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

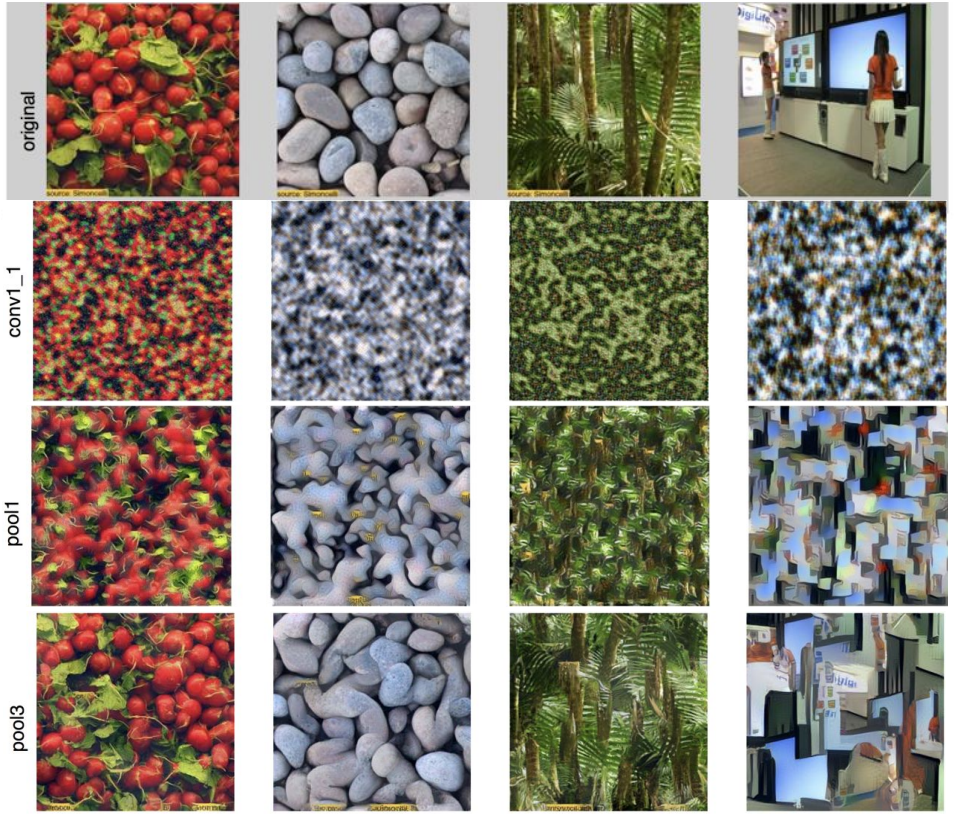
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5



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

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)

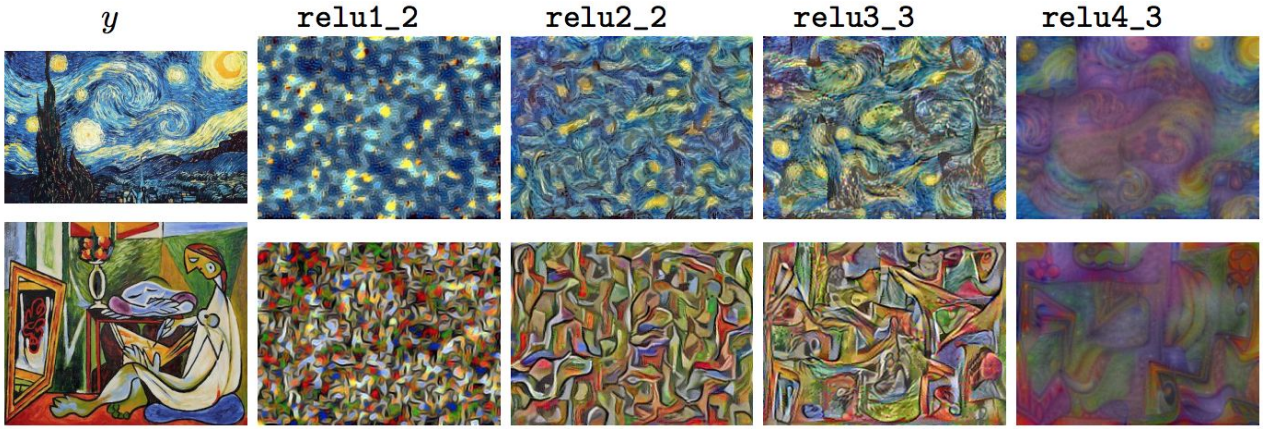


Figure from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016. Copyright Springer, 2016. Reproduced for educational purposes.

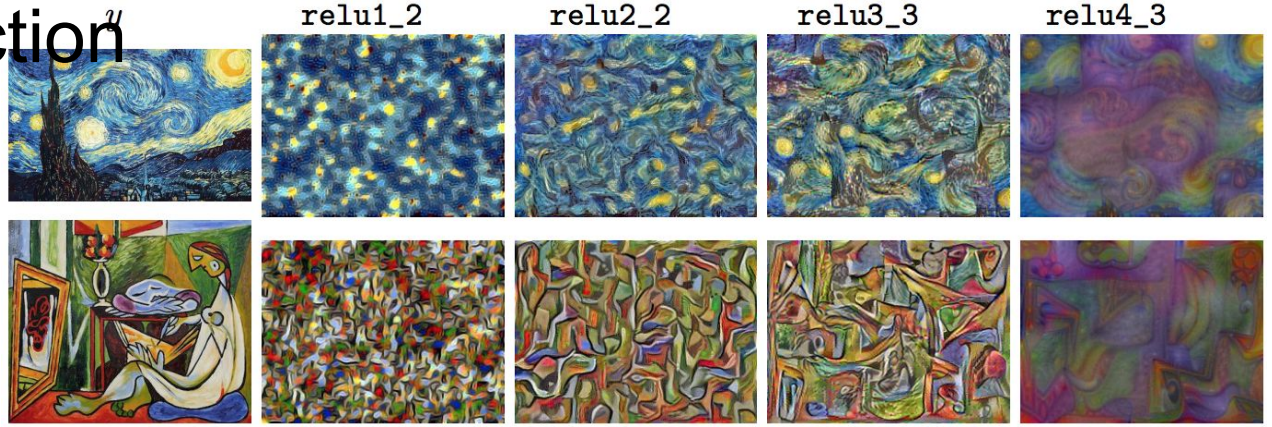
Neural Style Transfer: Feature + Gram Reconstruction

Texture synthesis (Gram reconstruction)

gram reconstruction

Feature reconstruction

feature reconstruction



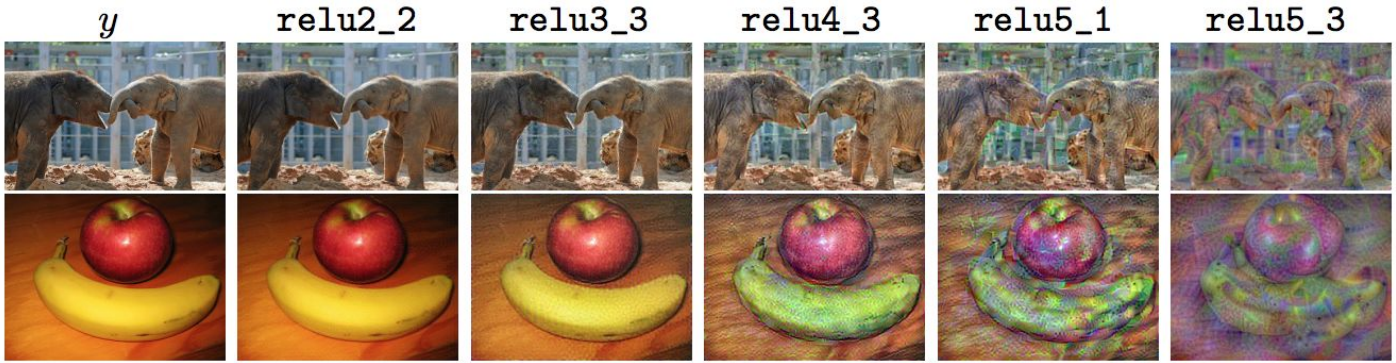


Figure from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016. Copyright Springer, 2016. Reproduced for educational purposes.

Neural Style Transfer

참고: https://blog.lunit.io/2017/04/27/style-transfer/

Content Image

This image is licensed under CC-BY 3.0

+

Style Image

Starry Night by Van Gogh is in the public domain

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

Neural Style Transfer

Content Image

This image is licensed under CC-BY 3.0

+

Style Image

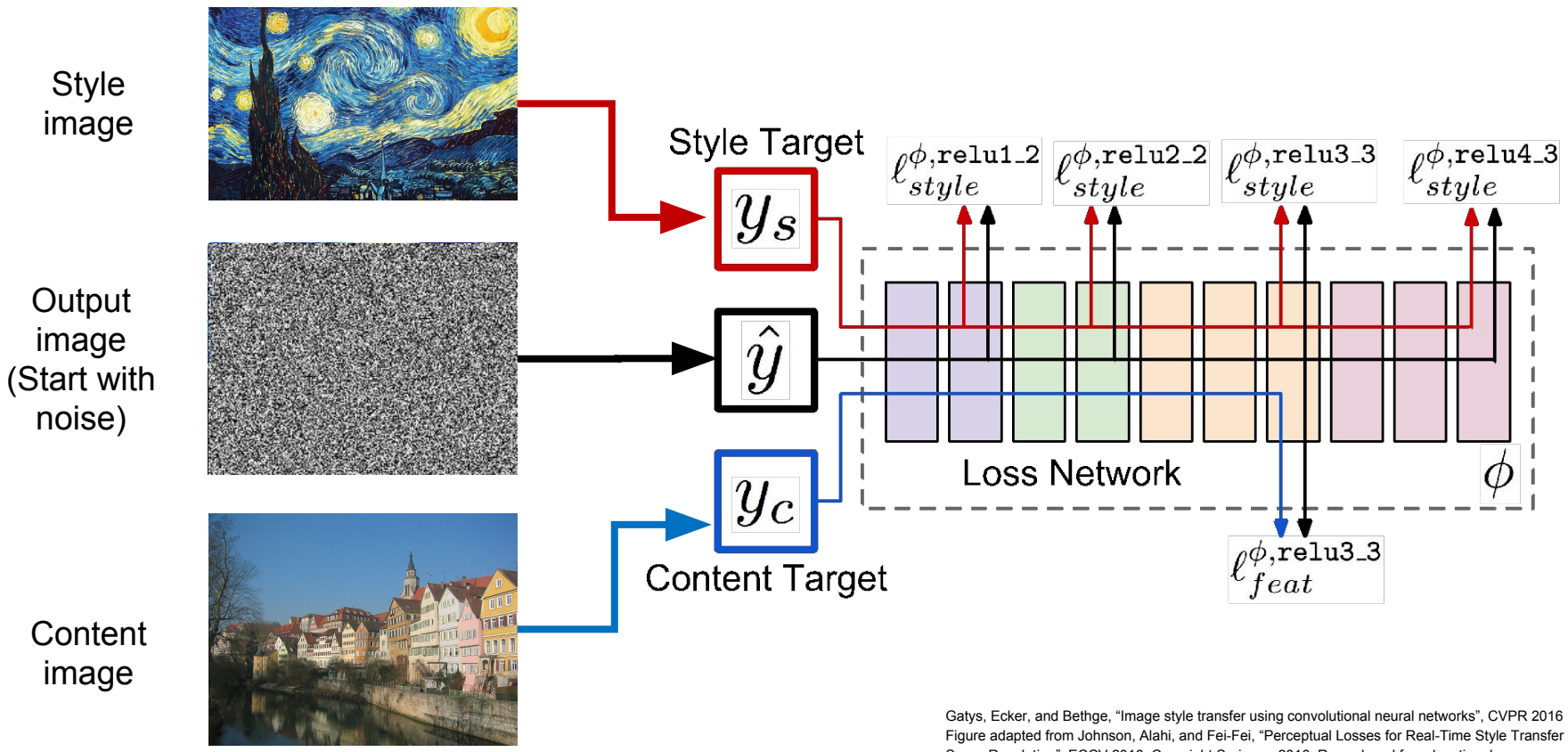
Starry Night by Van Gogh is in the public domain

=

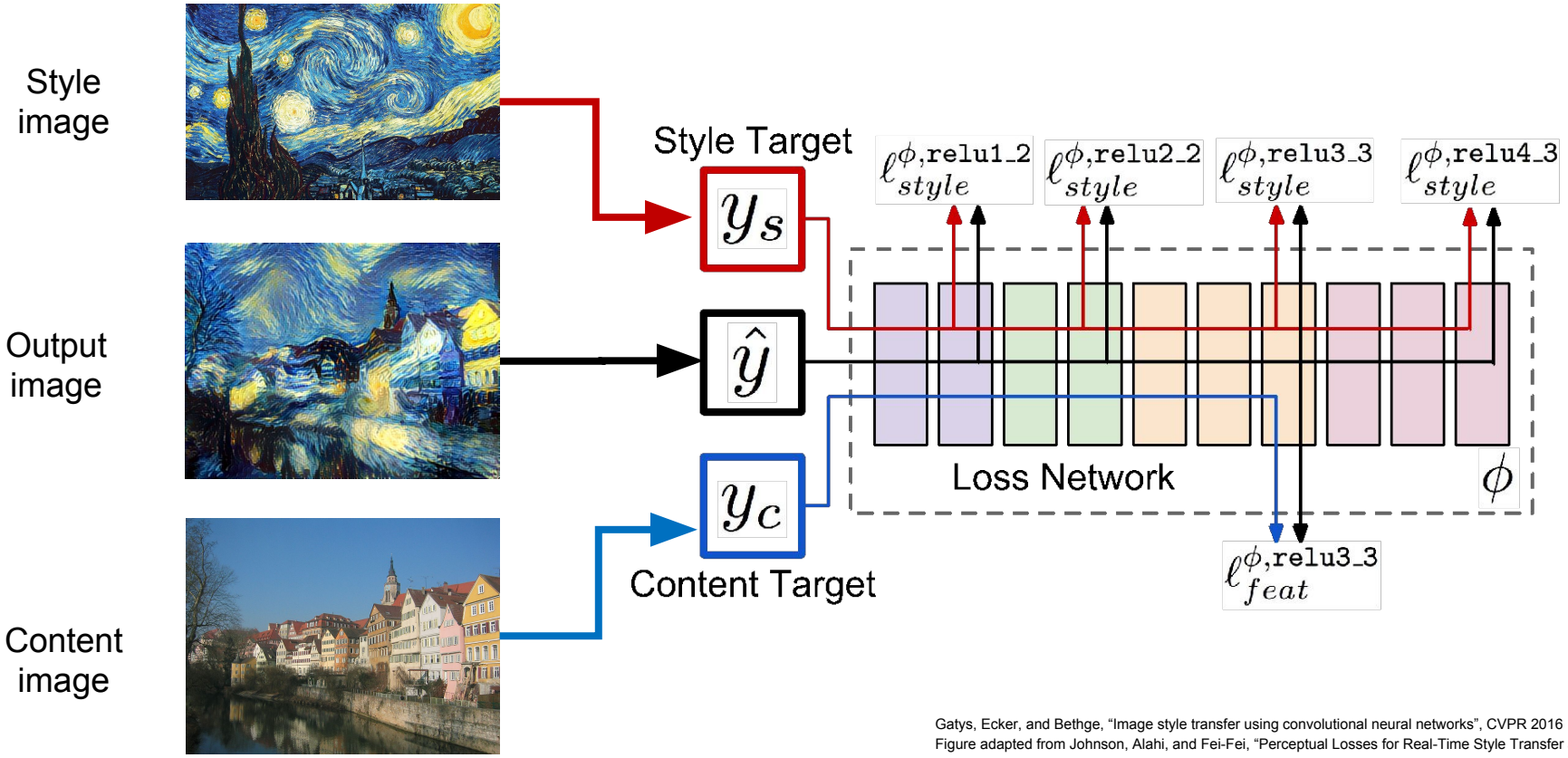
Style Transfer!

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

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



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

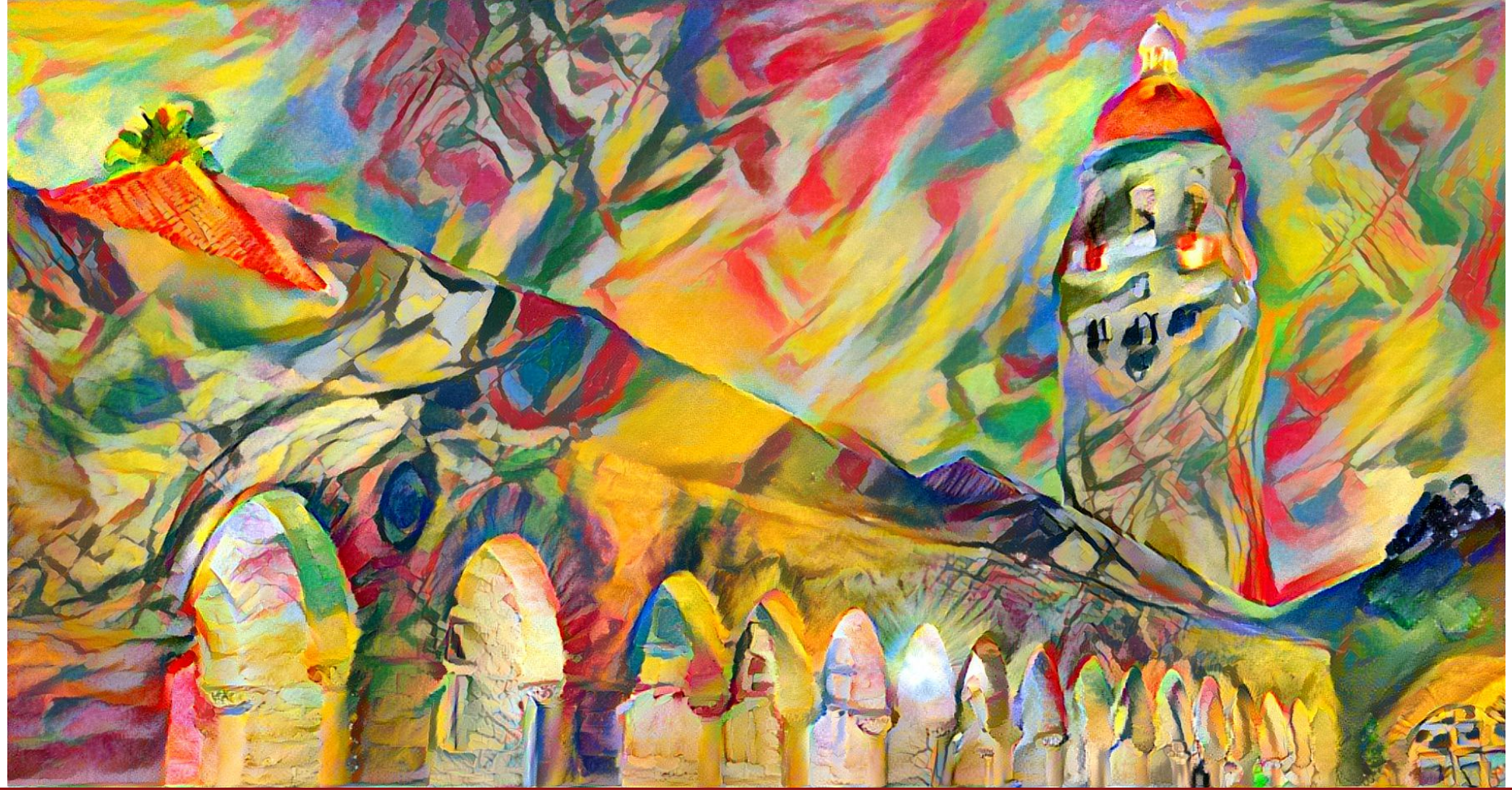
Gatys, Ecker, and Bethge, “Image style transfer using convolutional neural networks”, CVPR 2016
Figure copyright Justin Johnson, 2015.

Neural Style Transfer: Multiple Style Images

multiple gram matrix

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

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



Neural Style Transfer

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

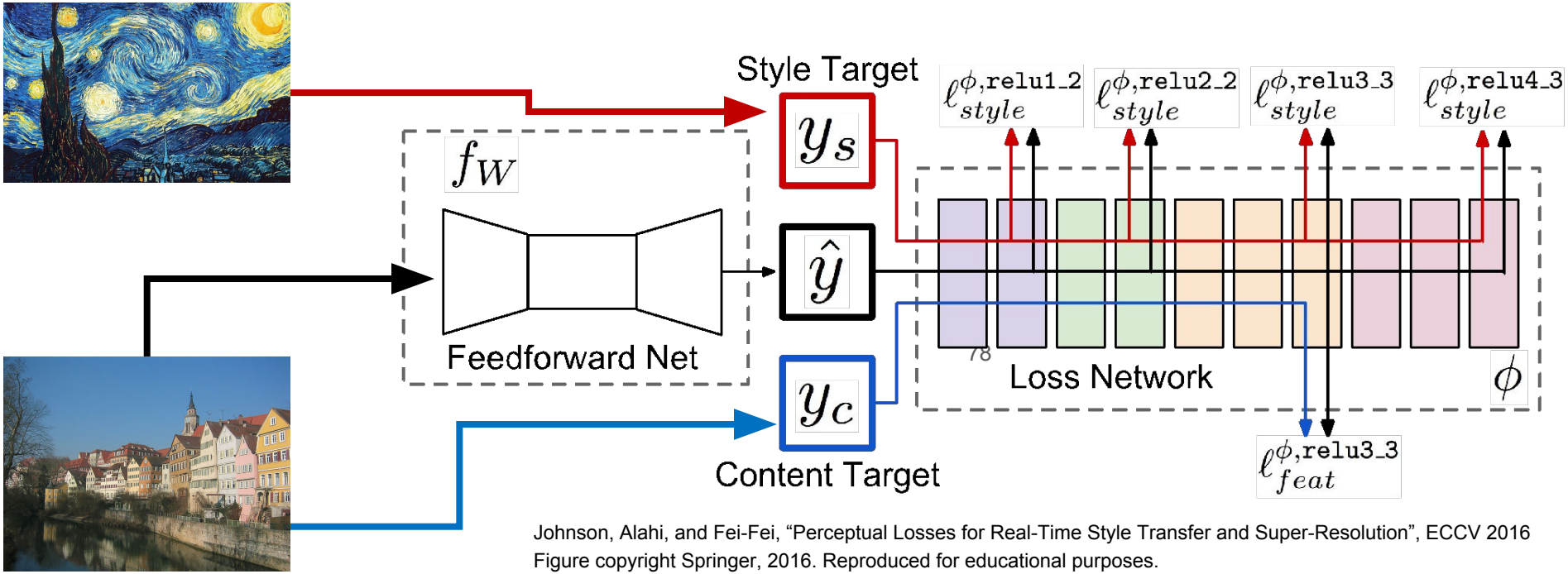
Neural Style Transfer

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

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

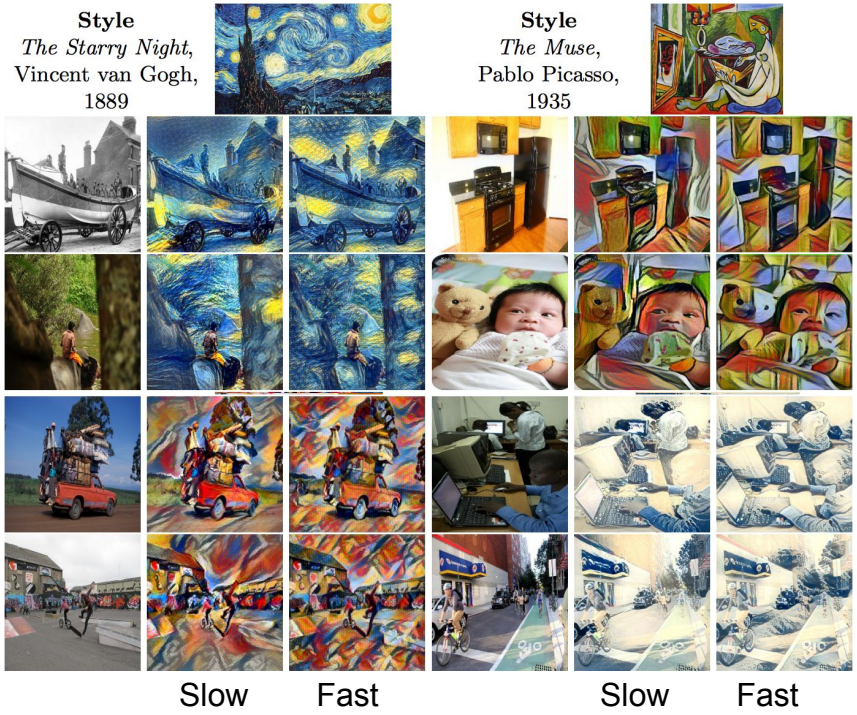
Fast Style Transfer

(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.

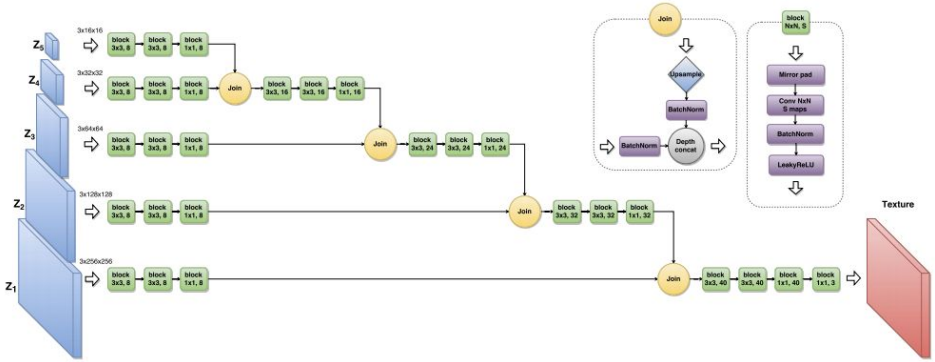
Fast Style Transfer



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.

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

Fast Style Transfer



Concurrent work from Ulyanov et al, comparable results

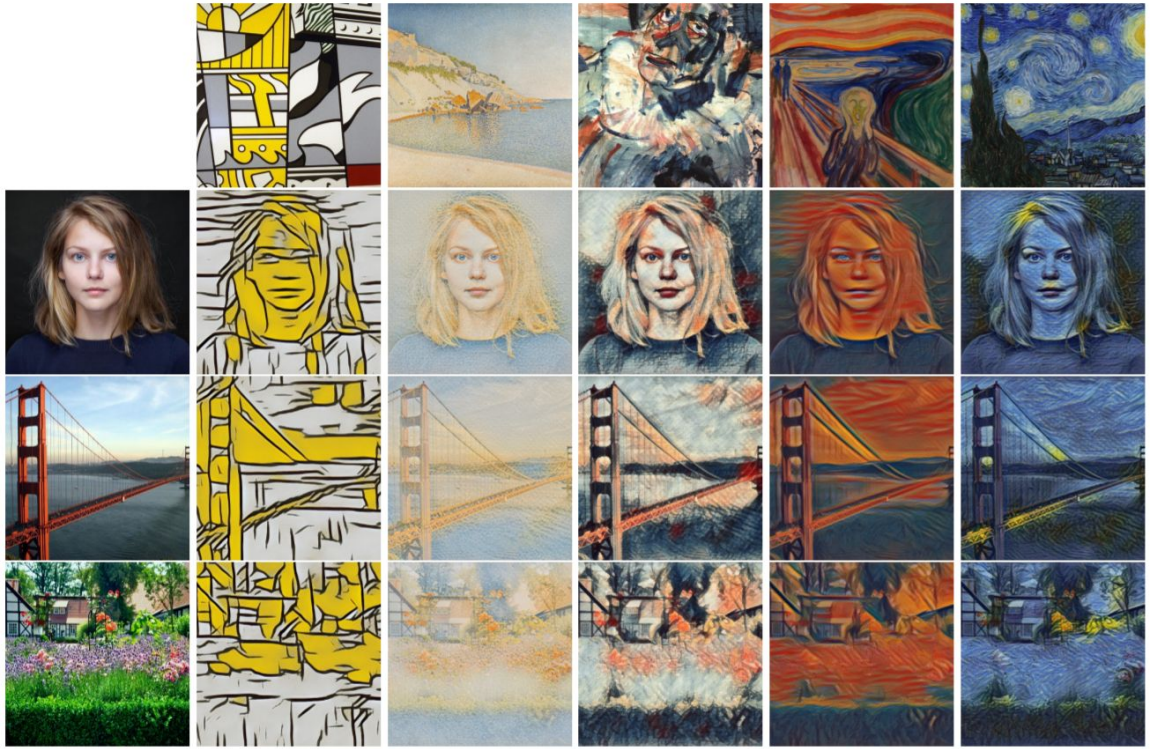
Ulyanov et al, “Texture Networks: Feed-forward Synthesis of Textures and Stylized Images”, ICML 2016
Ulyanov et al, “Instance Normalization: The Missing Ingredient for Fast Stylization”, arXiv 2016

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

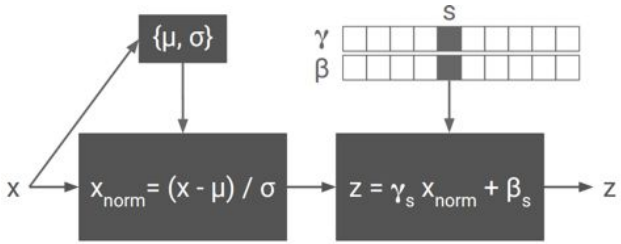
One Network, Many Styles



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

One Network, Many Styles

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



Single network can blend styles after training

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

Summary

Many methods for understanding CNN representations

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

Next time: **Unsupervised Learning**

Autoencoders

Variational Autoencoders

Generative Adversarial Networks