Lecture 12
Visualizing and Understanding

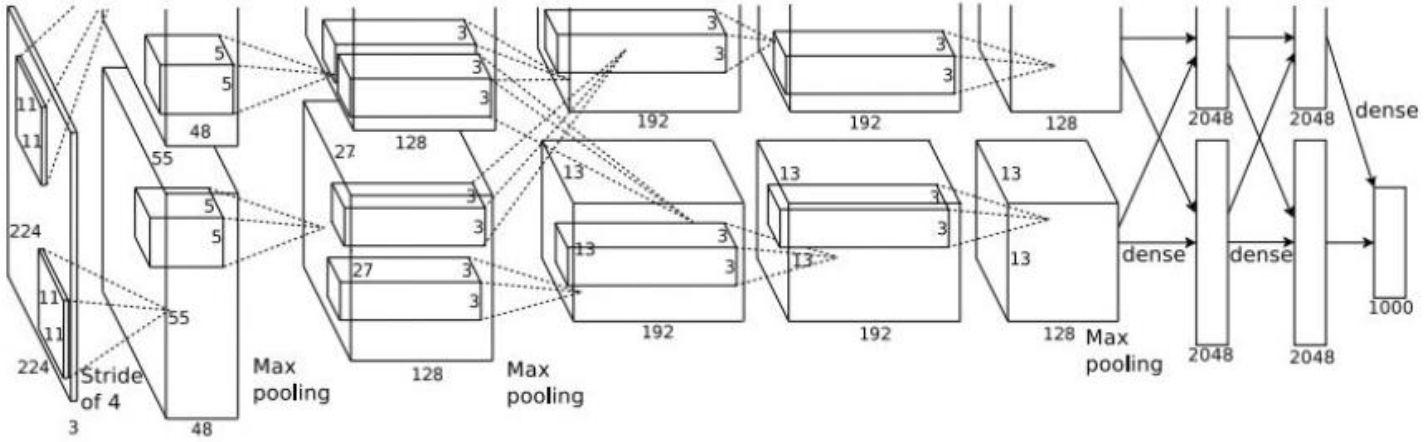
Introduction

- Many methods for understanding CNN representation
  - Activations
    - Nearest neighbors, Dimensionality reduction, maximal patches, occlusion
  - Gradients
    - Saliency maps, class visualization, fooling images, feature inversion
  - Fun
    - DeepDream, Style transfer

1. Activations

What's going on inside ConvNets?

This image is CC0 public domain



Class Scores:
1000 numbers

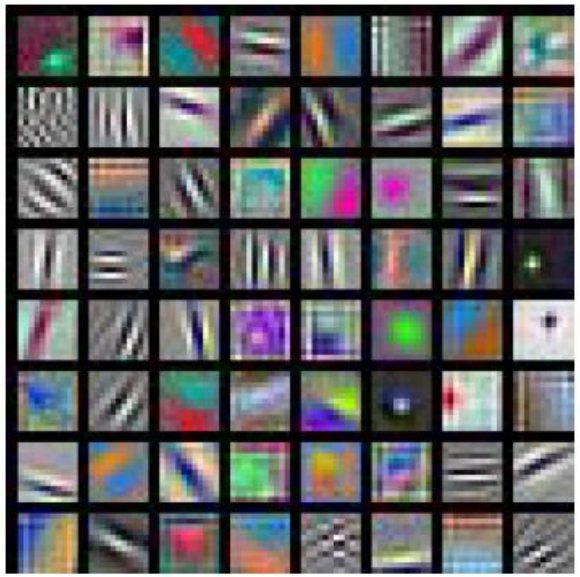
Input Image:
3 x 224 x 224

What are the intermediate features looking for?

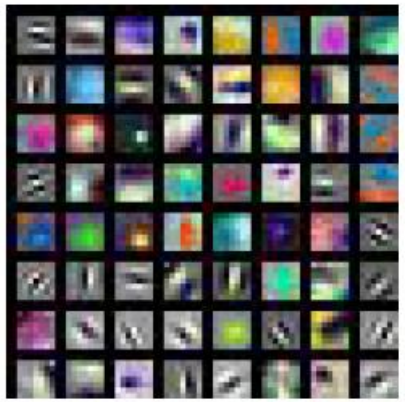
Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012.
Figure reproduced with permission.

1.1. Visualize Filters

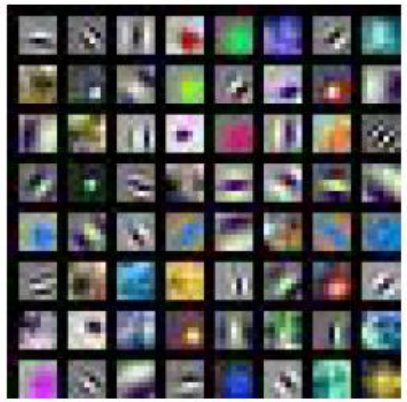
First Layer : Visualize Filters



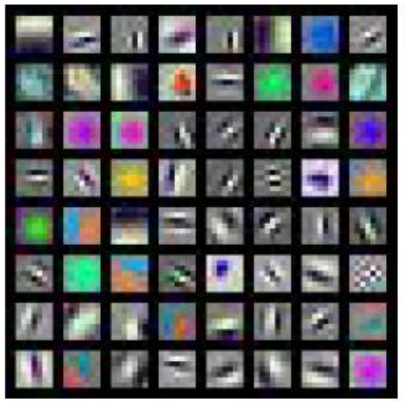
AlexNet:
64 x 3 x 11 x 11



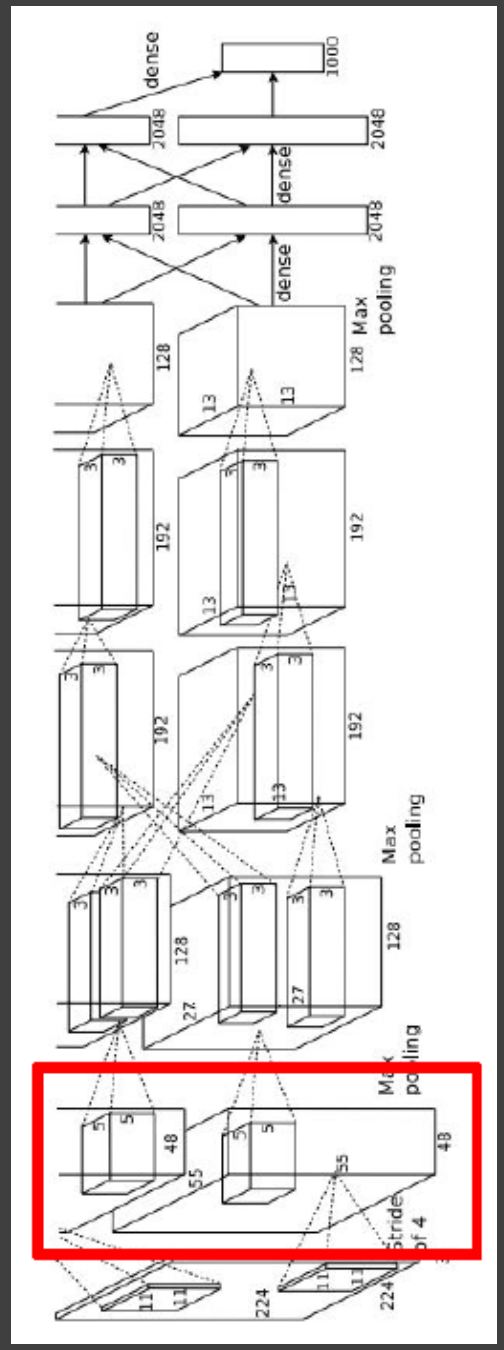
ResNet-18:
64 x 3 x 7 x 7



ResNet-101:
64 x 3 x 7 x 7



DenseNet-121:
64 x 3 x 7 x 7



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

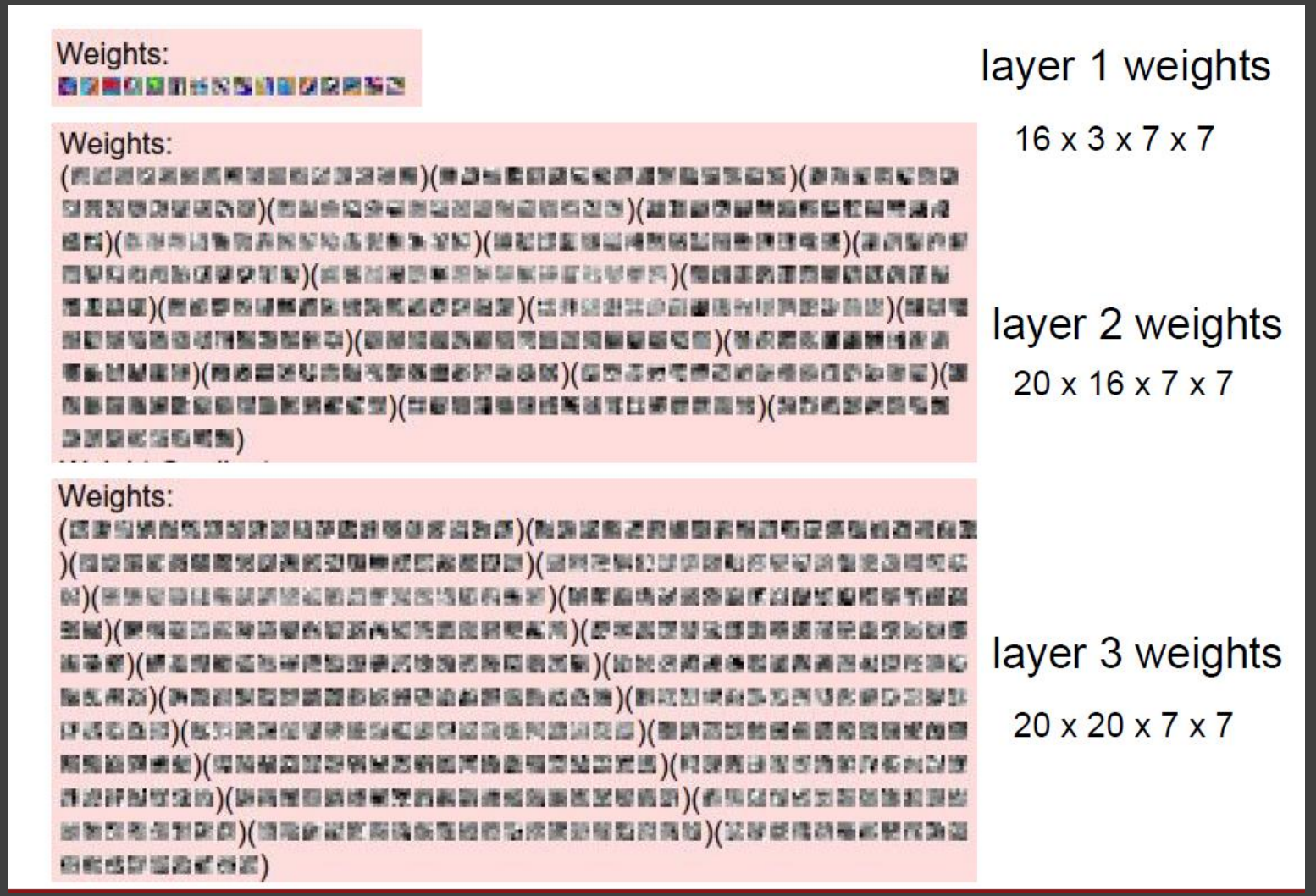
First layer 에서 Convolutional Layer 는 입력 이미지와 직접 내적을 수행한다 → 이 필터를 단순히 시각화 시키는 것만으로 이 필터가 이미지에서 무엇을 찾고 있는지 알아낼 수 있다.

AlexNet:
64 x 3 x 11 x 11

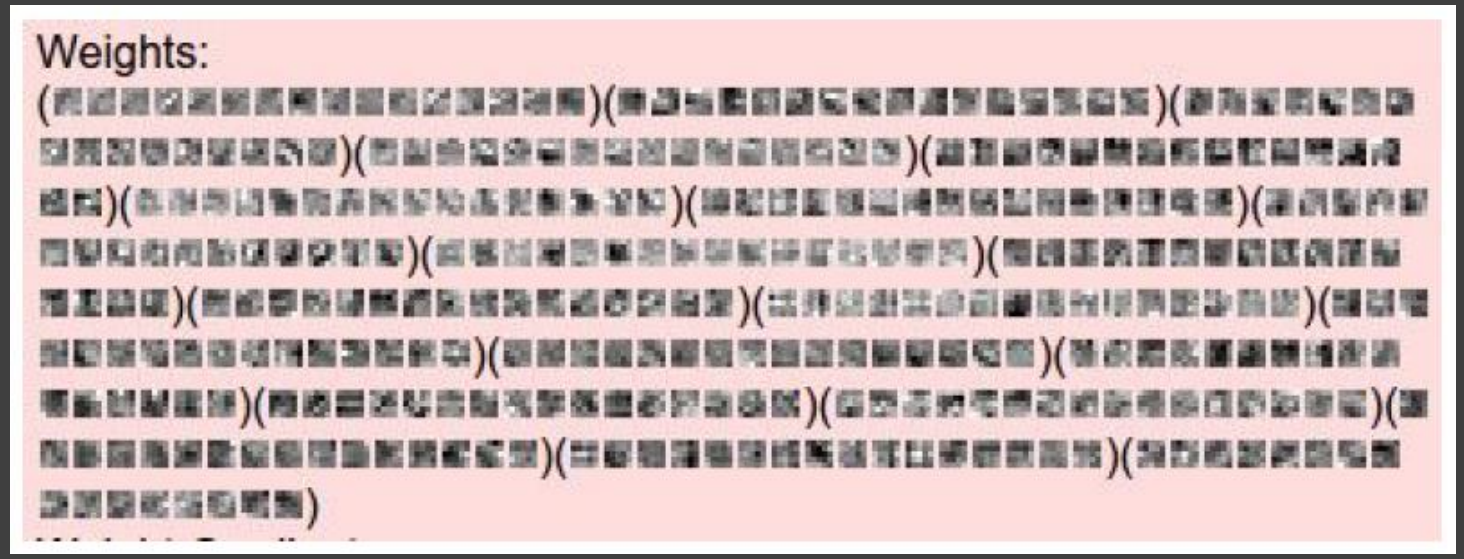
다양한 각도와 위치에서의 보색이 보인다.
(초록&분홍 / 주황 &파랑)

가장 많이 찾는 것이 Edge 성분

Visualize the filters/kernels (raw weights)



(these are taken from ConvNetJS CIFAR-10 demo)



layer 2 weights

20 x 16 x 7 x 7

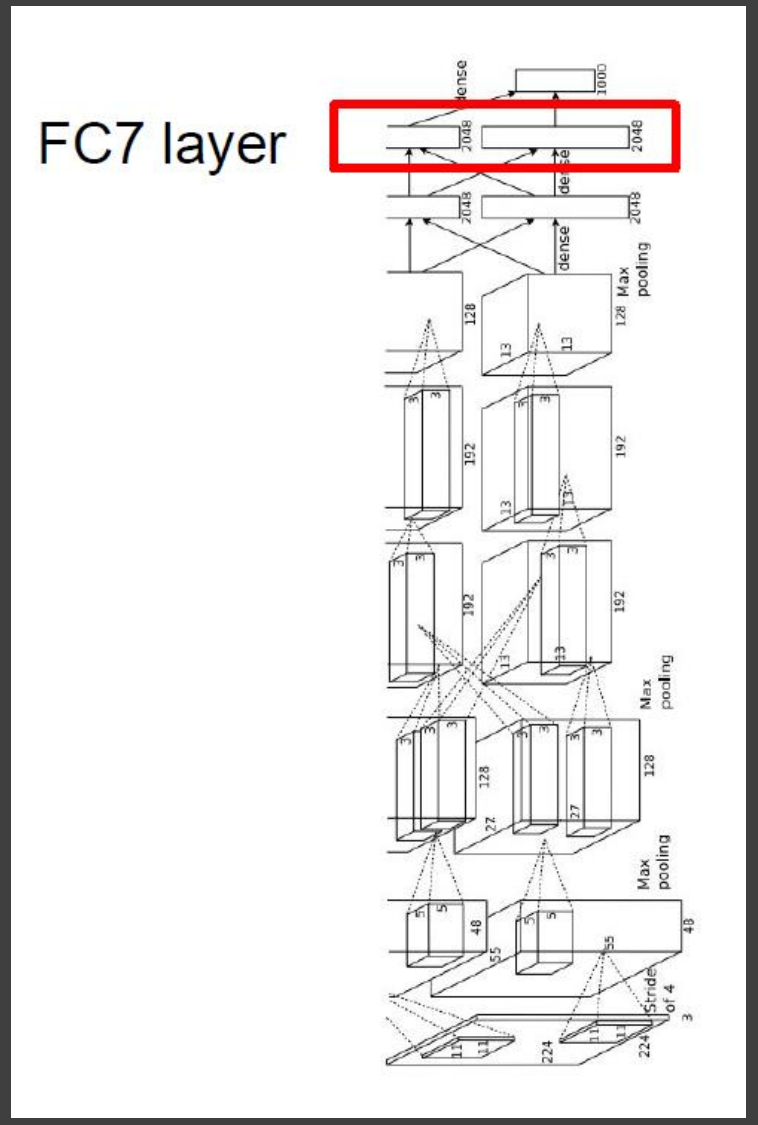
- 이 필터들은 이미지와 직접 연결되어 있지 않다. 그래서 이 필터가 무엇을 찾는지 알 수 없다.
- 따라서 우리가 시각화한 내용은 “두 번째 레이어의 결과를 최대화 시키는 첫 번째 레이어의 출력 패턴이 무엇인지“ 이다.
- 하지만 이미지의 관점에서 첫 번째 레이어의 출력이 어떻게 생겼는지 감을 잡고 해석하기란 쉽지 않다.

1.2. Visualize Feature maps

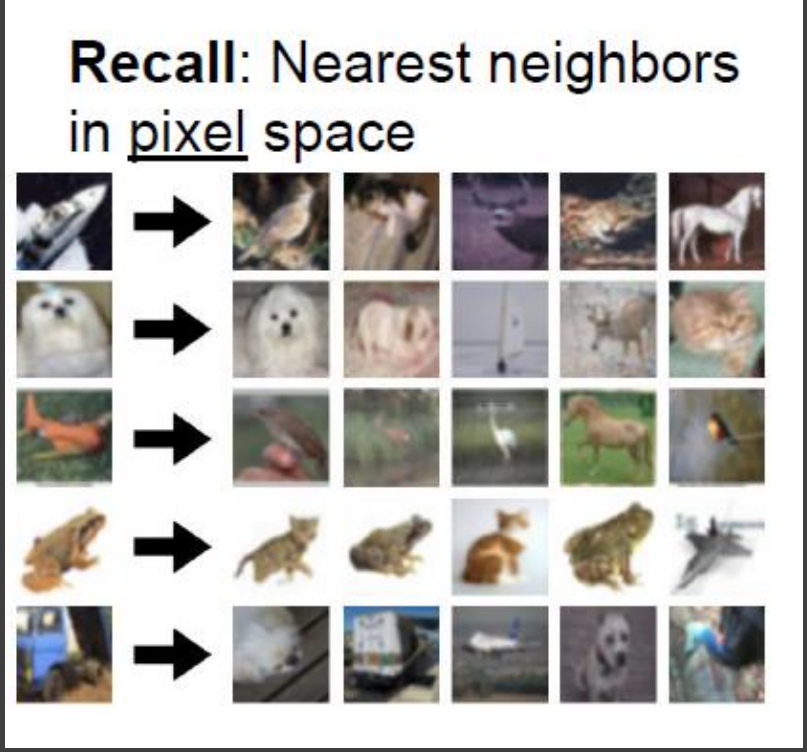
Last Layer

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

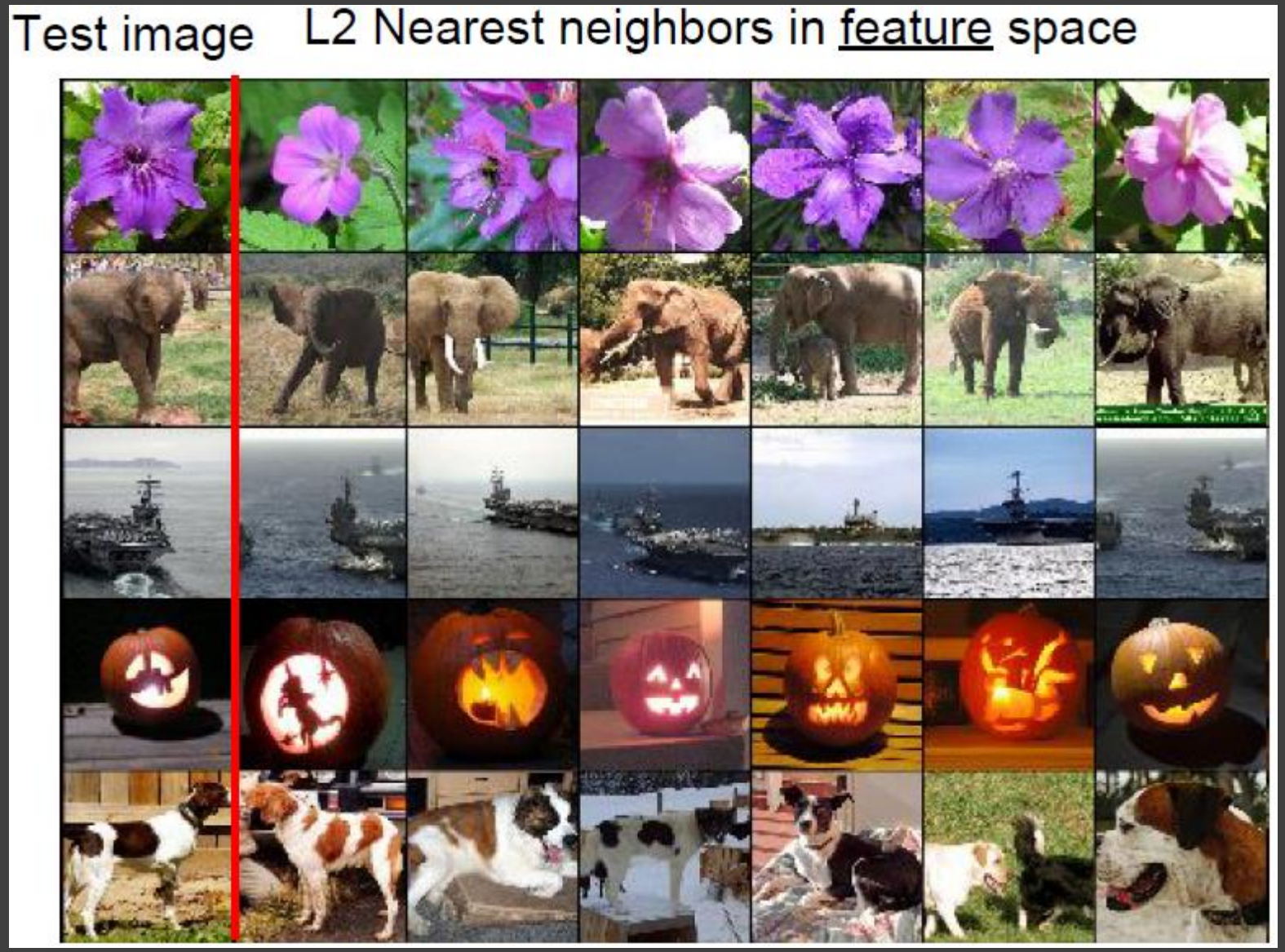
Run the network on many images, collect the feature vectors



Last Layer : Nearest Neighbors



- Lecture 2, CIFAR-10 데이터들의 “ image pixel space“ 에서의 Nearest Neighbors
- 이와 같이 거리가 가까운 이미지들을 시각화해보는 방법도 시각화 기법이 될 수 있다.

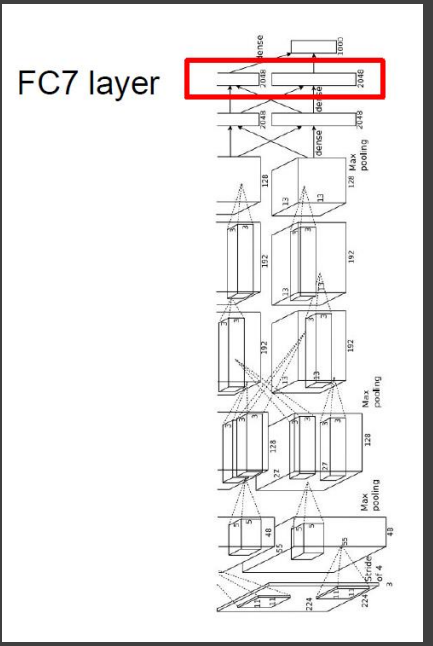


- CNN 에서 나온 4096-dimentional feature vector space 에서 nearest neighbors 를 계산한 것이다.

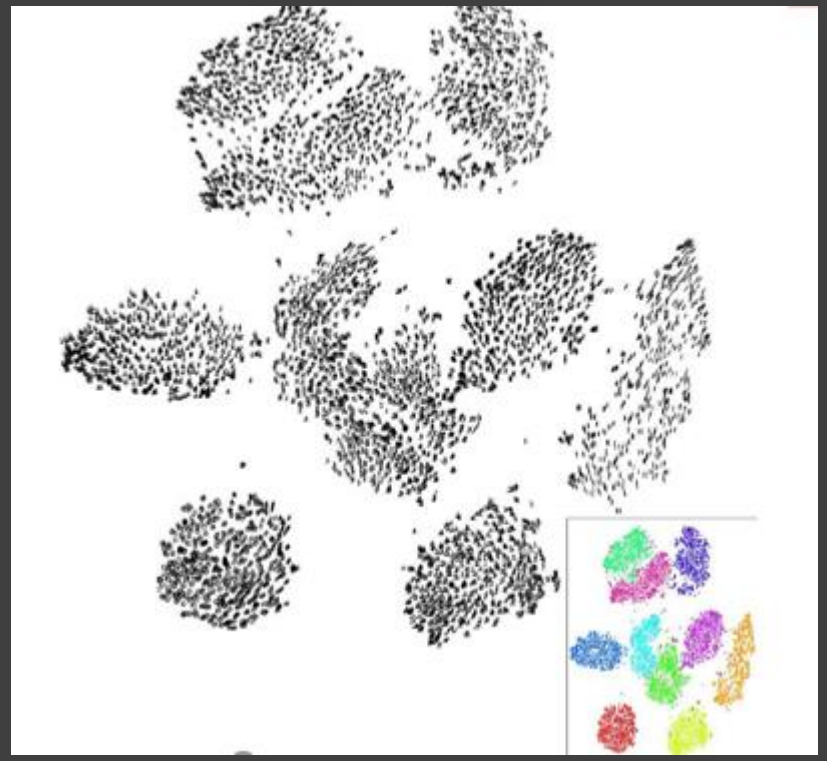
Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012.
Figures reproduced with permission.

Last Layer : Dimensionality Reduction

- 4096 차원의 FC7 feature vectors 를 2차원으로 차원 축소한 것을 시각화 한다.
- 차원을 축소하기 위해 사용하는 알고리즘
  - PCA
    - 4096-dim 과 같은 고차원 특징 벡터들을 2-dim 으로 압축시키는 기법
    - 이 방법을 통해서 특징 공간을 조금 더 직접적으로 시각화 시킬 수 있다.
  - t-SNE(t-distributed stochastic neighbor embeddings)
    - PCA 보다 더 강력한 알고리즘



t-SNE 예시 : MNIST 를 t-SNE 를 통해 시각화 한 모습



- MNIST
  - 0부터 9까지로 이루어진 손 글씨 숫자 데이터셋
  - 각 이미지는 gray scale 28x28 이미지
- Process
  - t-SNE 가 MNIST 의 28 x 28-dim 데이터를 입력으로 받고, 2-dim 으로 압축한다.
  - 2-dim 으로 압축한 결과를 통해 MNIST 를 시각화 시킨다.

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

이것을 ImageNet 을 분류하려고 학습시킨 네트워크의 마지막 레이어에도 적용해볼 수 있다.

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/

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.

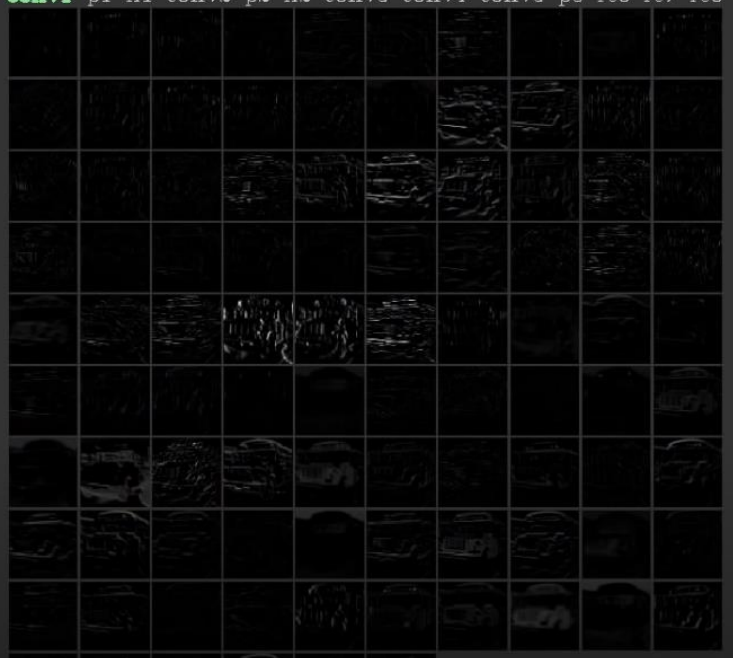
Visualizing Activations

- 중간 레이어의 가중치를 시각화 시켜도 해석하기 쉽지 않다 → Activation map 을 시각화 시키면 일부 해석할 수 있는 것들이 있다.

Jason Yosinski 의 Tool?

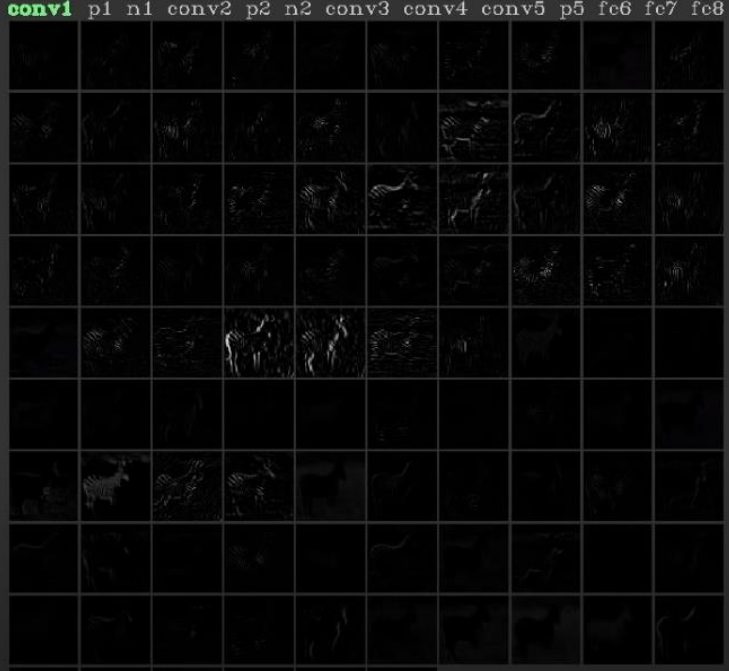
1.00 school bus
0.00 minibus
0.00 passenger car
0.00 trolleybus
0.00 moving van

conv1 p1 n1 conv2 p2 n2 conv3 conv4 conv5 p5 fc6 fc7 fc8 prob



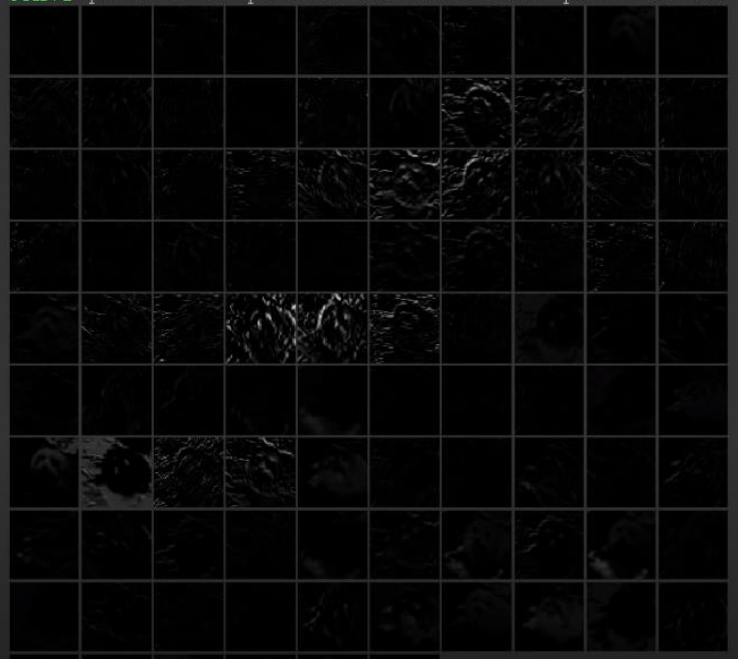
1.00 zebra
0.00 prairie chicken
0.00 tiger
0.00 leopard
0.00 tiger cat

conv1 p1 n1 conv2 p2 n2 conv3 conv4 conv5 p5 fc6 fc7 fc8 prob



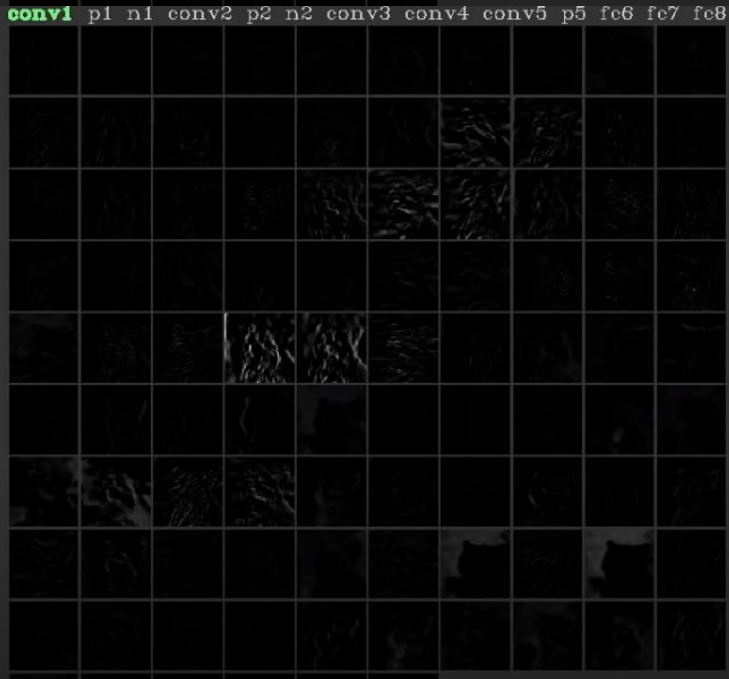
0.76 Maltese dog
0.09 toy poodle
0.05 miniature pooc
0.05 Lhasa
0.02 Tibetan terrier

conv1 p1 n1 conv2 p2 n2 conv3 conv4 conv5 p5 fc6 fc7 fc8 prob



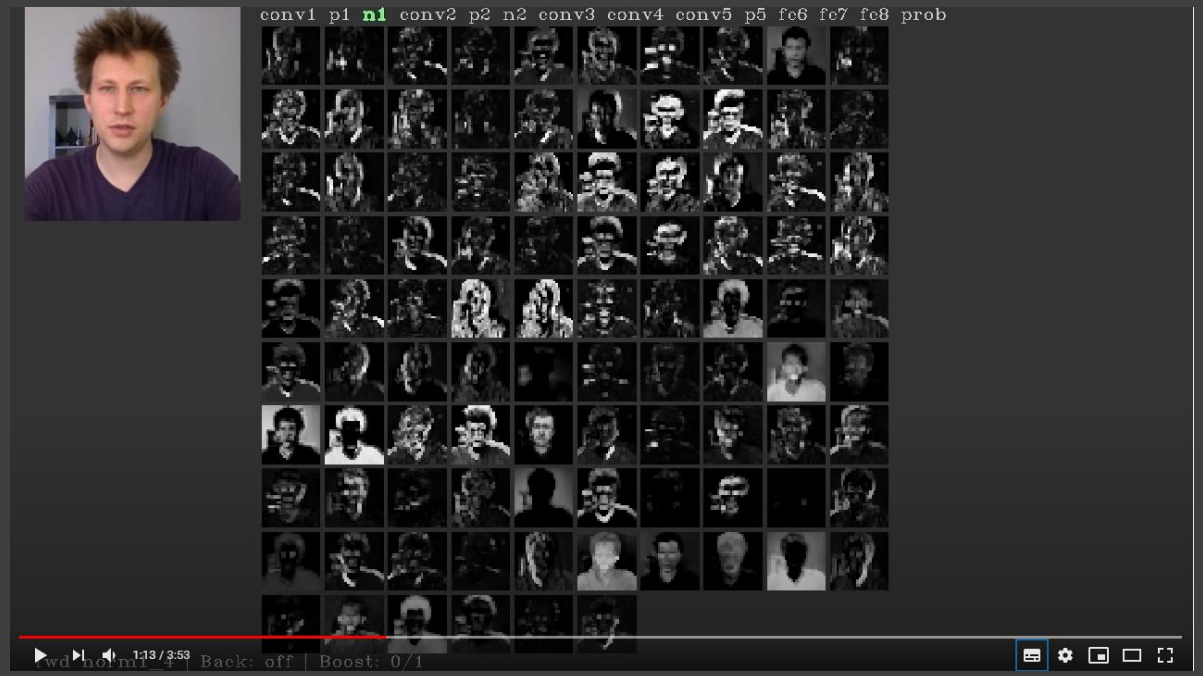
0.89 Yorkshire terri
0.05 silky terrier
0.05 Australian terr
0.00 Norwich terrier
0.00 Norfolk terrier

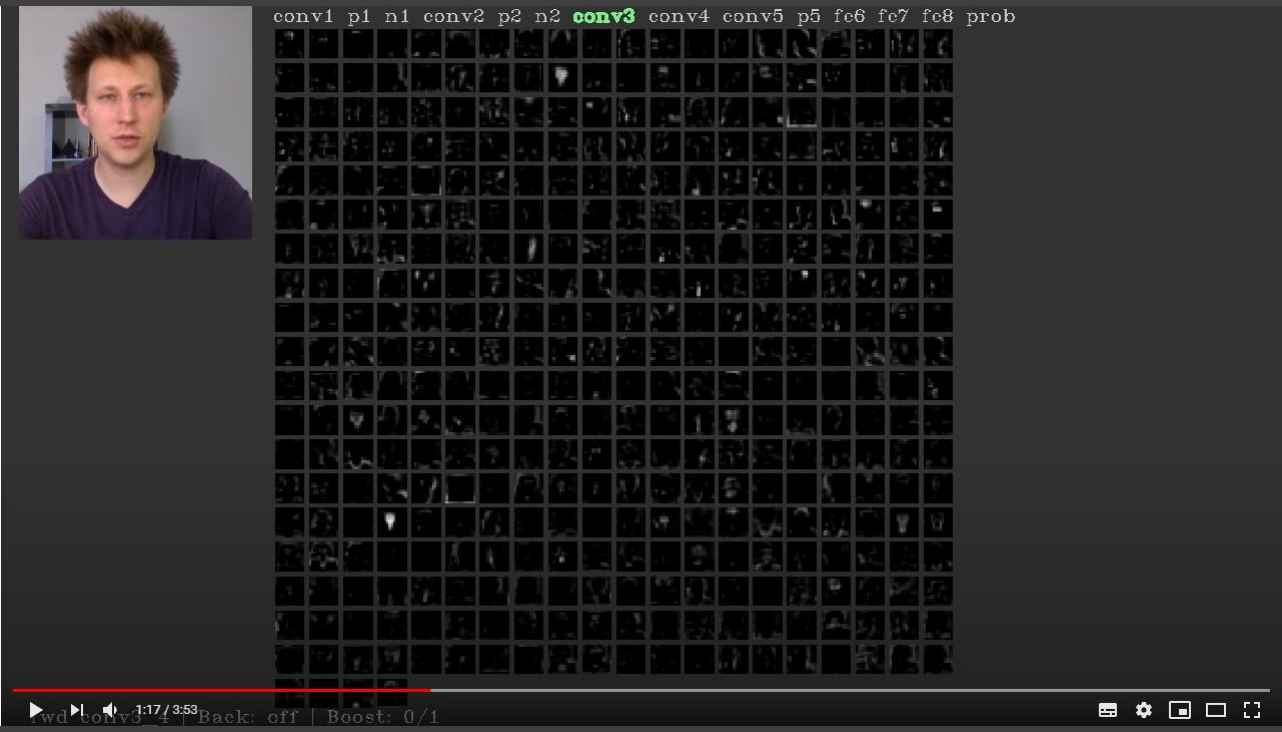
conv1 p1 n1 conv2 p2 n2 conv3 conv4 conv5 p5 fc6 fc7 fc8 prob




conv1 p1 n1 conv2 p2 n2 conv3 conv4 conv5 p5 fc6 fc7 fc8 prob
1:11 / 3:53
Back: off | Boost: 0/1


conv1 p1 n1 conv2 p2 n2 conv3 conv4 conv5 p5 fc6 fc7 fc8 prob
1:12 / 3:53
Back: off | Boost: 0/1

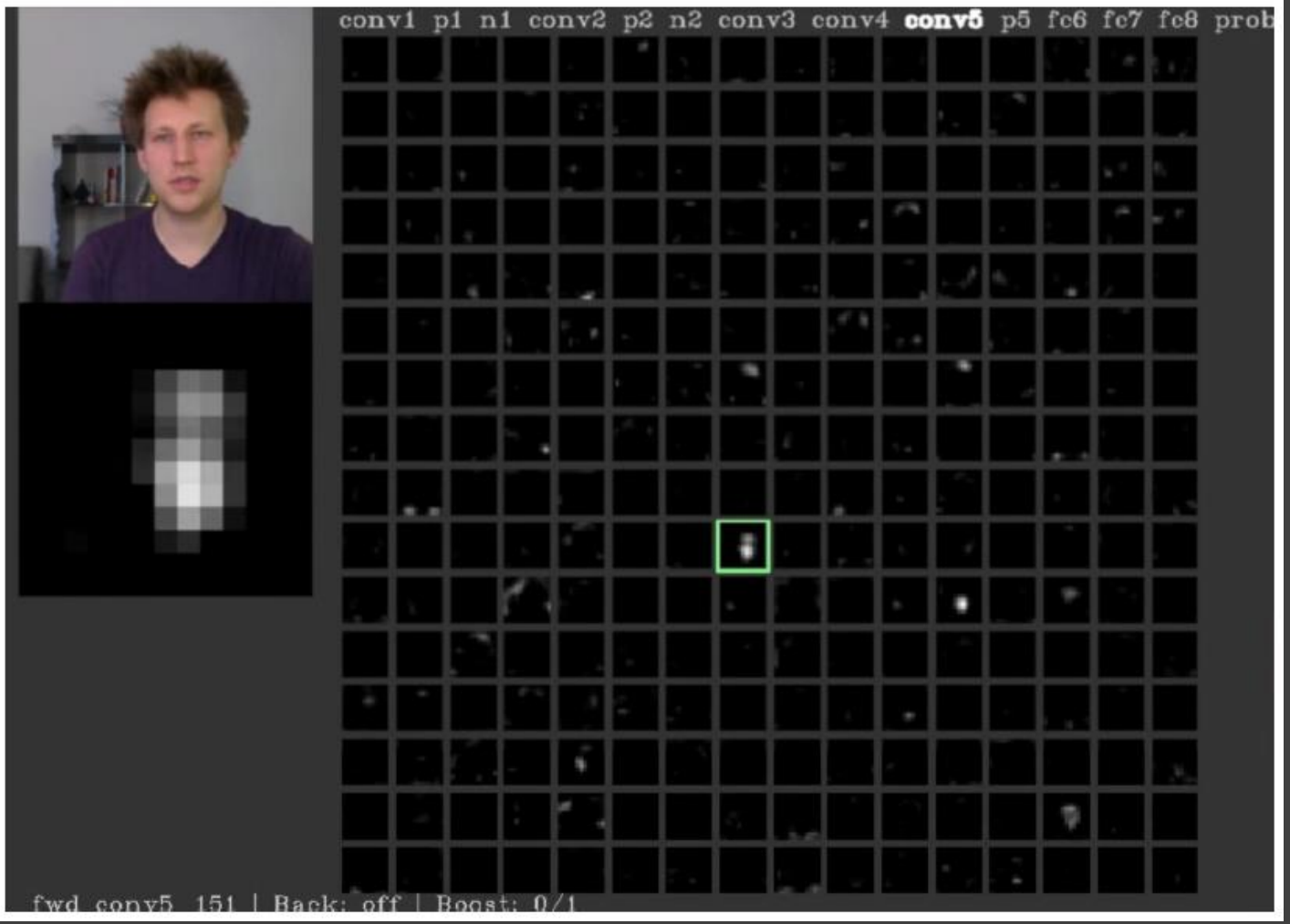

conv1 p1 n1 conv2 p2 n2 conv3 conv4 conv5 p5 fc6 fc7 fc8 prob
1:13 / 3:53
Back: off | Boost: 0/1


conv1 p1 n1 conv2 p2 n2 conv3 conv4 conv5 p5 fc6 fc7 fc8 prob
1:17 / 3:53
Back: off | Boost: 0/1


conv1 p1 n1 conv2 p2 n2 conv3 conv4 conv5 p5 fc6 fc7 fc8 prob
1:19 / 3:53
Back: off | Boost: 0/1

Example of AlexNet

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



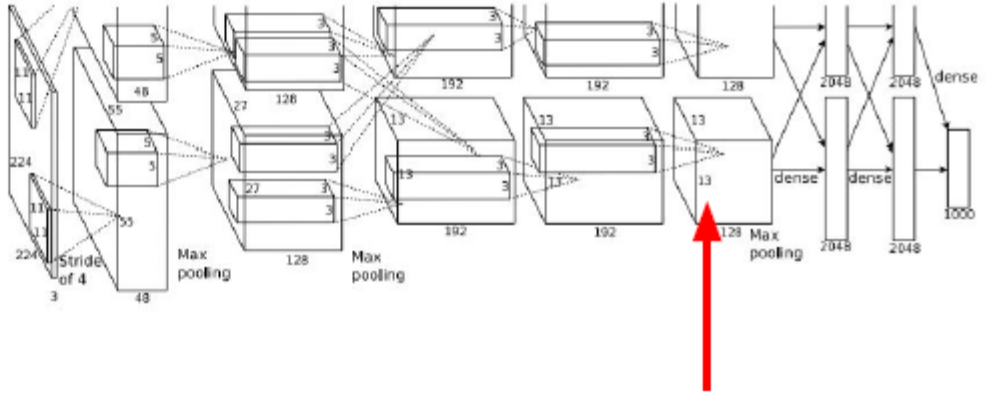
Convolutional layer 가 입력에서 어떤 특징을 찾고 있는지 짐작할 수 있다.

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

1.3. Others

Maximally Activating Patches

어떠한 이미지가 들어와야 각 뉴런들의 활성이 최대화되는지 알아보기 위해 시각화 해보자

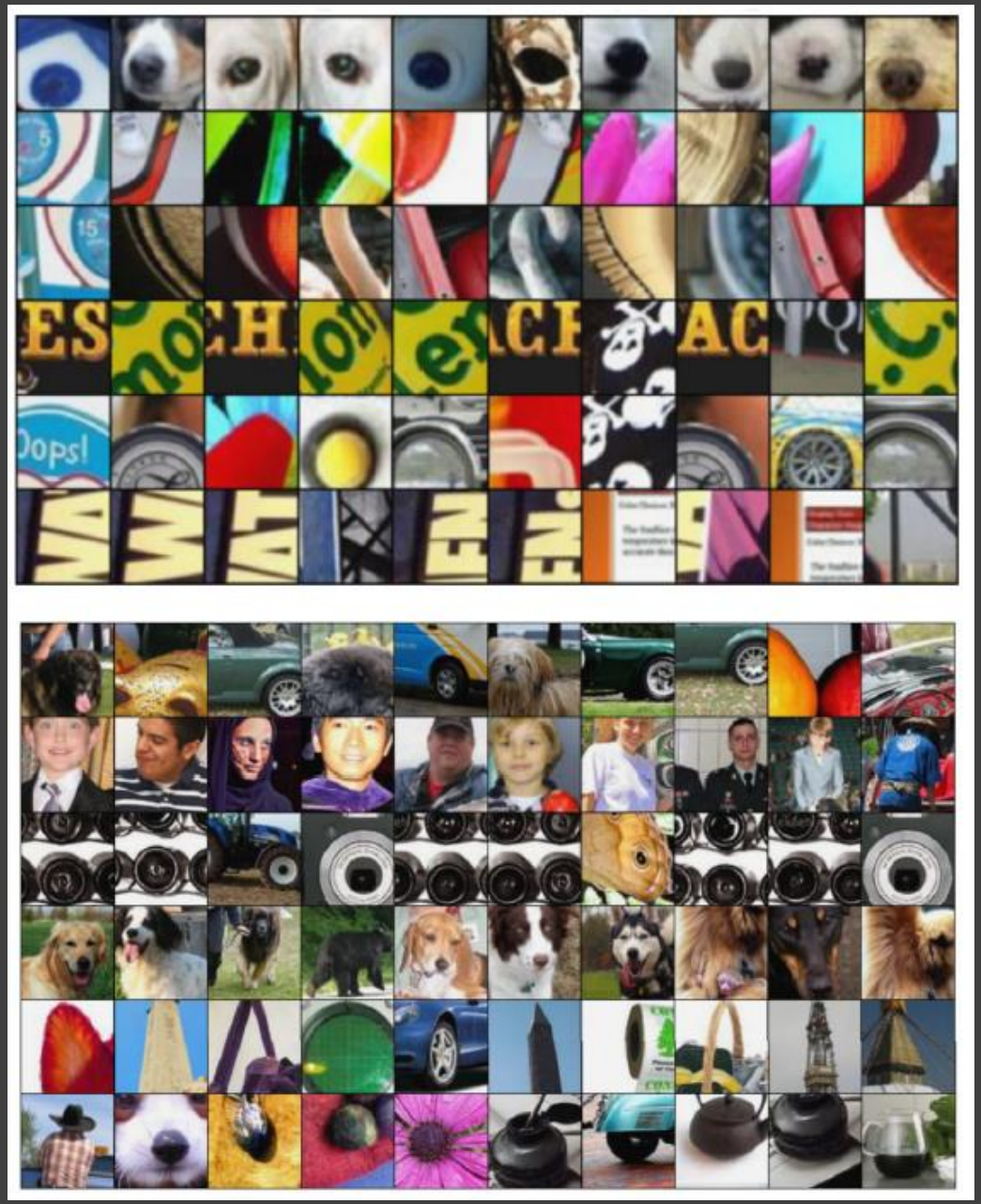


Pick a layer and a channel; e.g. conv5 is 128 x 13 x 13, pick channel 17/128

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

Visualize image patches that correspond to maximal activations

- Patch 들의 특징을 통해서 해당 뉴런이 무엇을 찾고 있는지 짐작해 볼 수 있다.

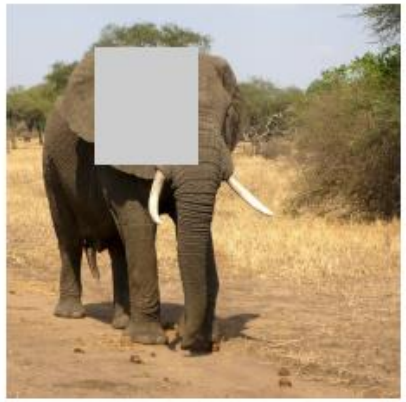


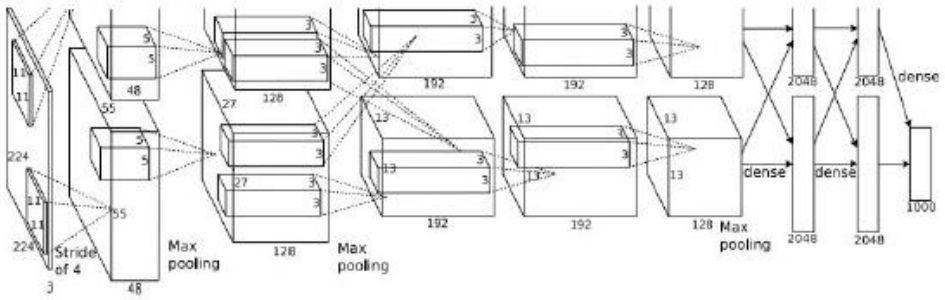
Springenberg et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015
Figure copyright Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, 2015; reproduced with permission.

Occlusion Experiments

입력의 어떤 부분이 분류를 결정짓는 근거가 되는지에 관한 연구이다.

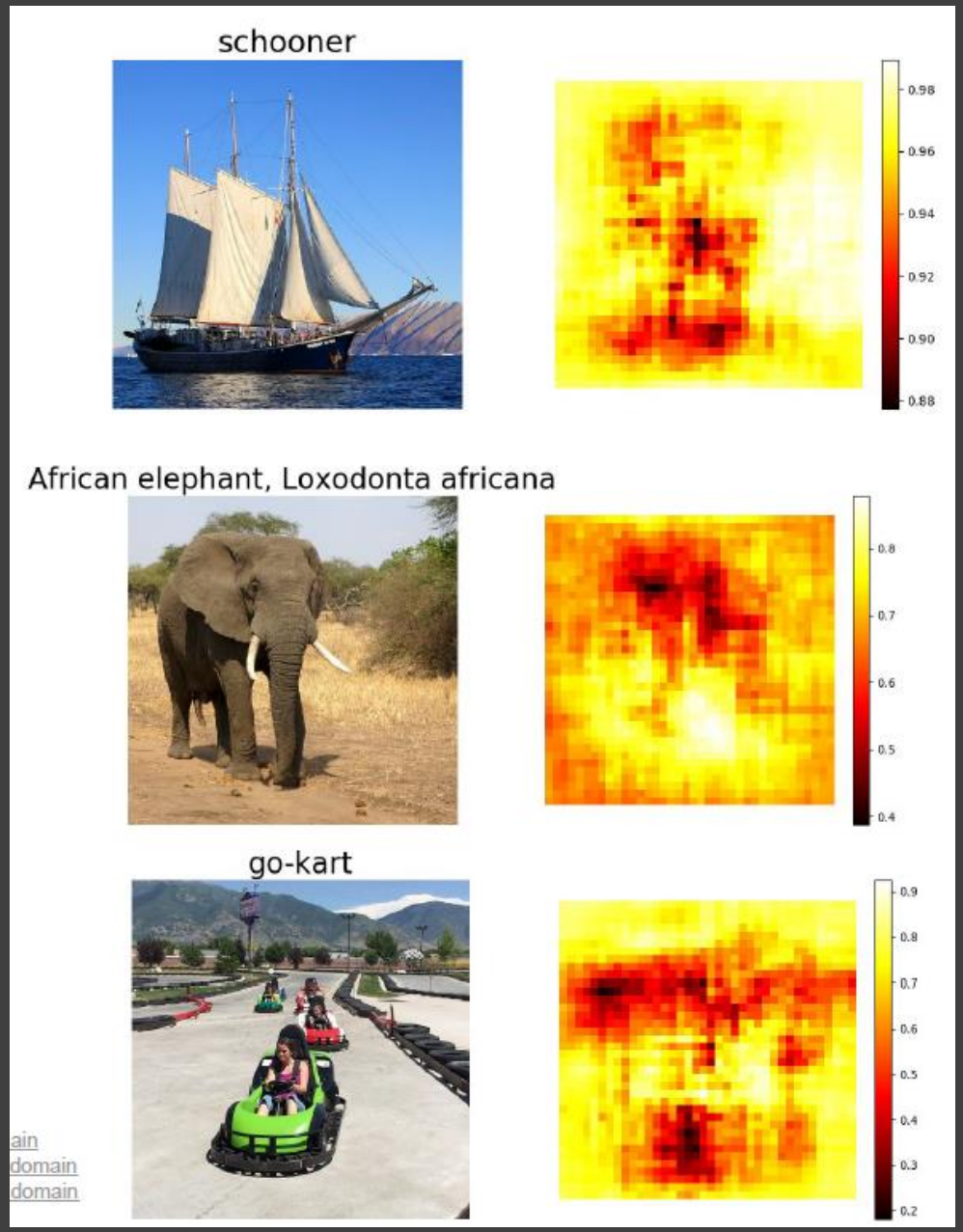
Mask part of the image before feeding to CNN, draw heatmap of probability at each mask location



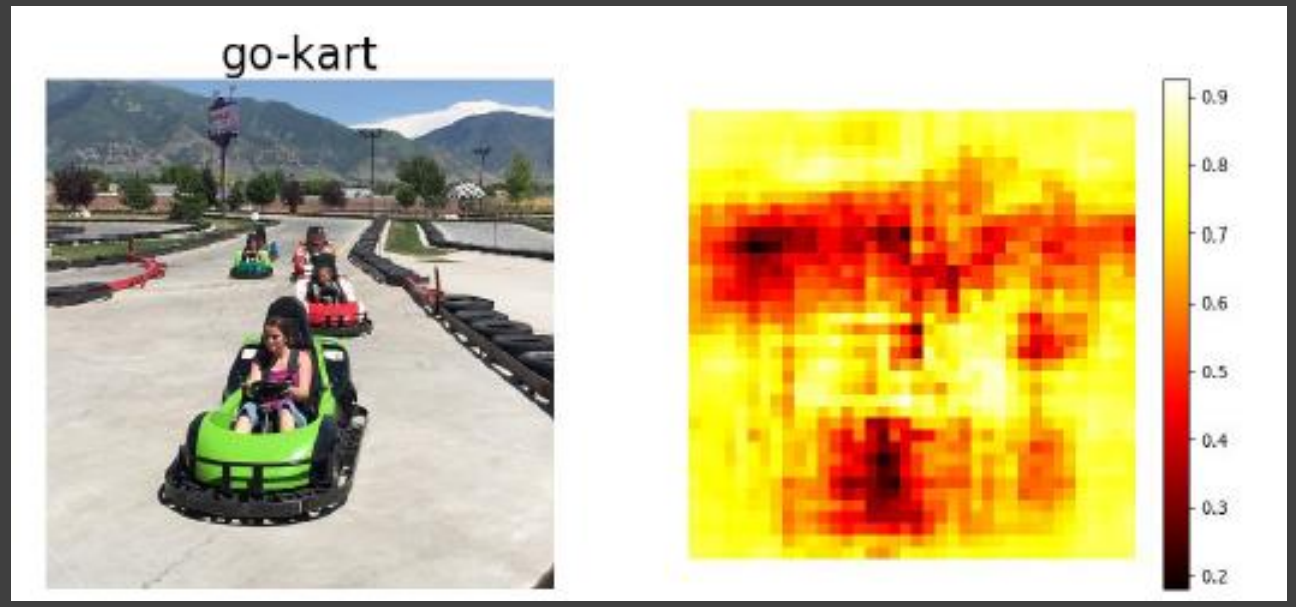


- 이미지의 일부를 가리고, 가린 부분을 데이터셋의 평균 값으로 채워버린다.
- 가려진 이미지를 네트워크에 통과시키고 네트워크가 이 이미지를 예측한 확률을 heatmap 으로 기록한다.
- 이 occlusion patch 를 전체 이미지에 대해 slide 하면서 같은 과정을 반복한다.

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



- Heat map 은 이미지를 가린 patch 의 위치에 따른 네트워크의 예측 확률의 변화를 의미한다.
- 만약 이미지의 일부를 가렸는데 네트워크의 스코어의 변화가 크게 발생한다면 가려진 바로 그 부분이 분류를 결정짓는데 아주 중요한 부분이었다는 사실을 짐작할 수 있다.

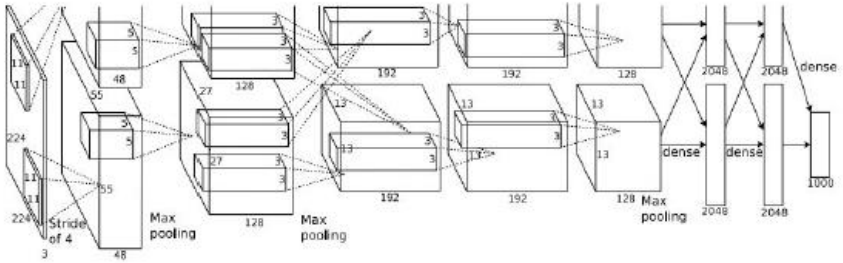


- Heat map 에서 빨간색 지역은 확률 값이 낮고, 노란색 지역은 확률 값이 높음을 의미한다.
- 앞쪽의 go-kart 를 가렸을 때 go-kart 에 대한 확률이 아주 많이 감소함을 볼 수 있다.
- 이를 통해 네트워크가 분류를 결정할 때 실제로 go-kart 를 아주 많이 고려한다는 사실을 알 수 있다.

2. Gradients

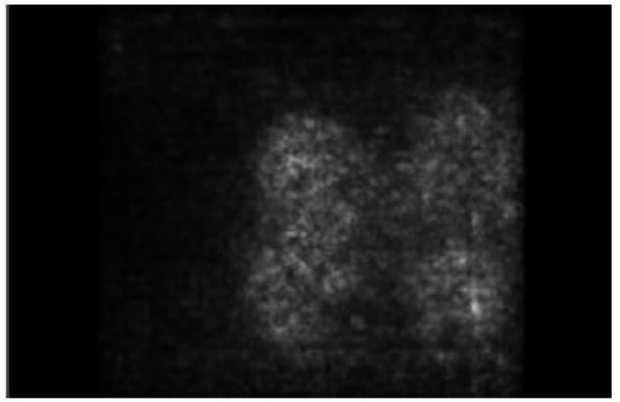
Saliency Maps 입력 이미지의 어떤 픽셀이 영향력 있는지 알려준다.

How to tell which pixels matter for classification?



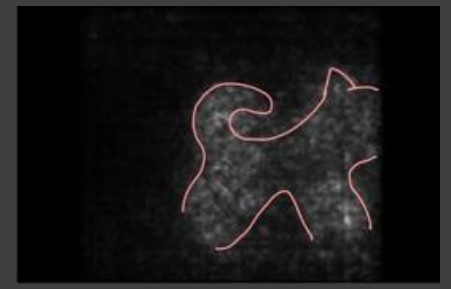
Dog

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



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

"개 " 를 분류하는데 있어서 어떤 픽셀들이 필요한지 알 수 있다.

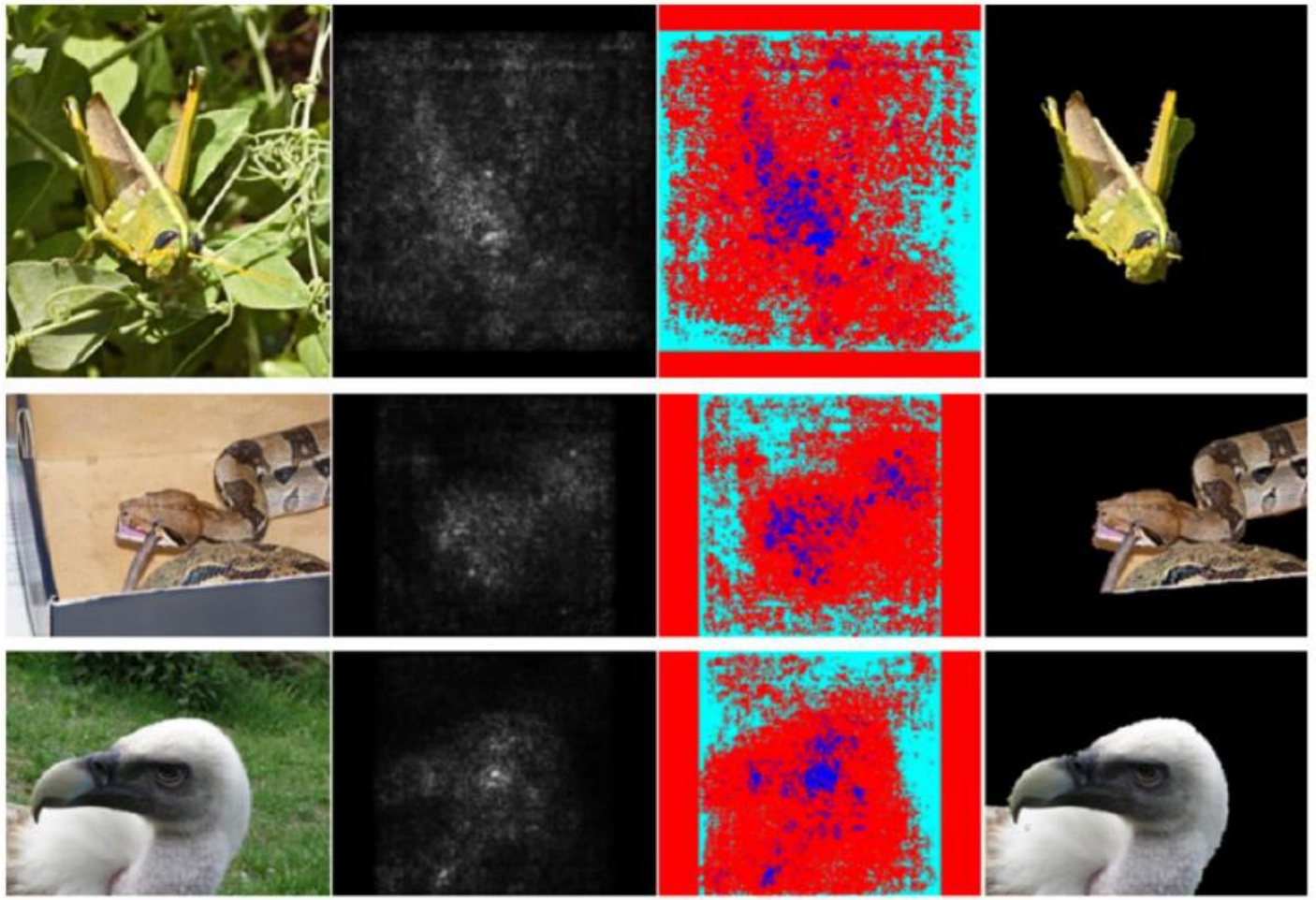


Saliency map 을 보면 "개" 의 윤곽이 나타난다.

Saliency Maps

Saliency Maps : segmentation without supervision

Use GrabCut on saliency map

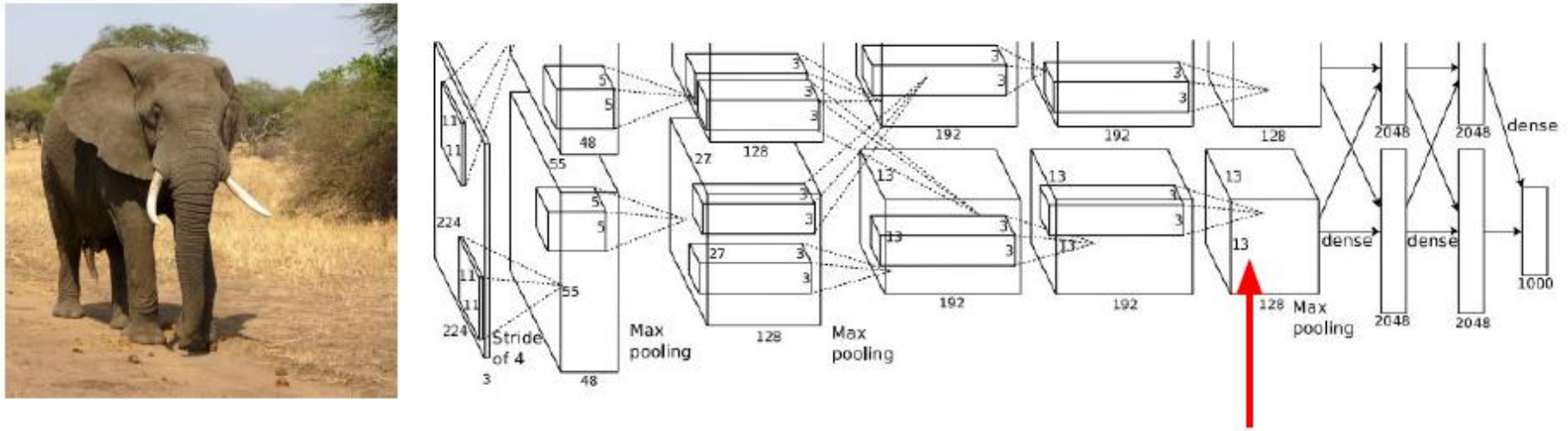


Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.

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

Intermediate Features via (guided) backprop

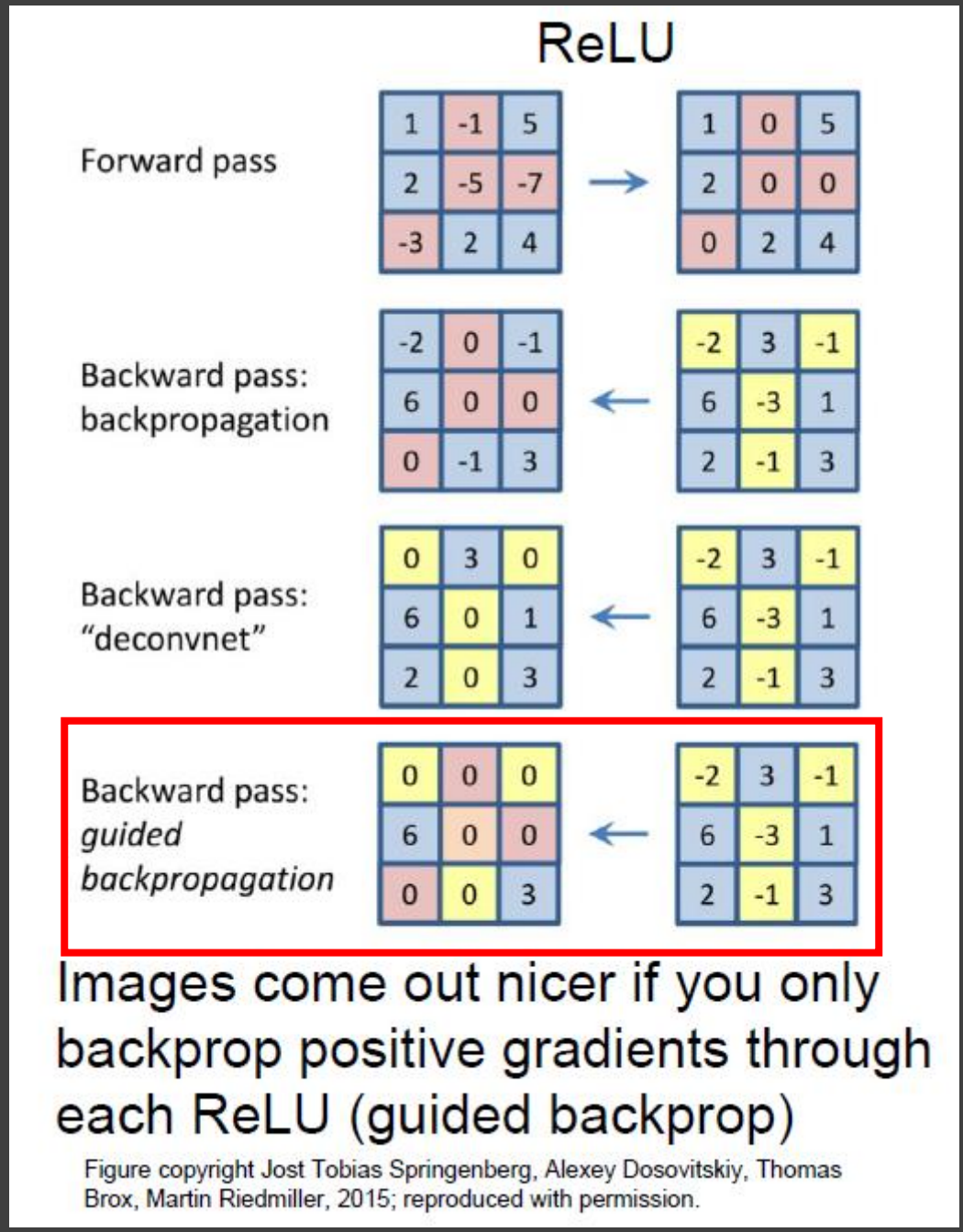
입력 이미지의 어떤 부분이, 내가 선택한 중간 뉴런의 값에 영향을 주는지 찾는 것이다.



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

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

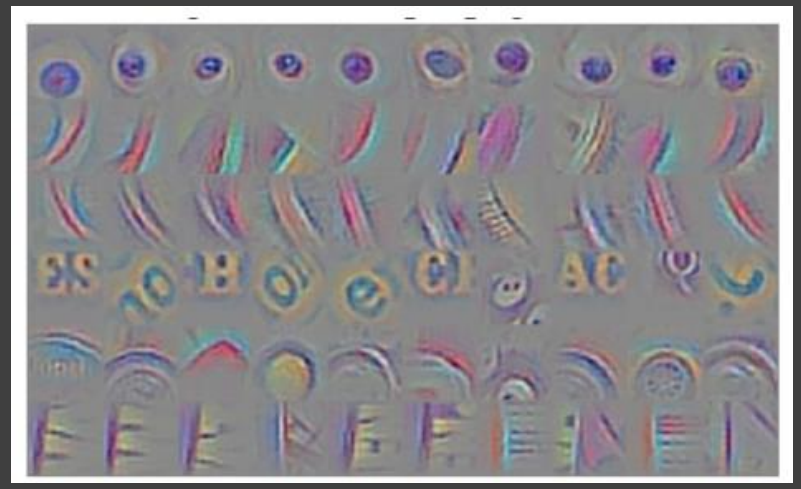


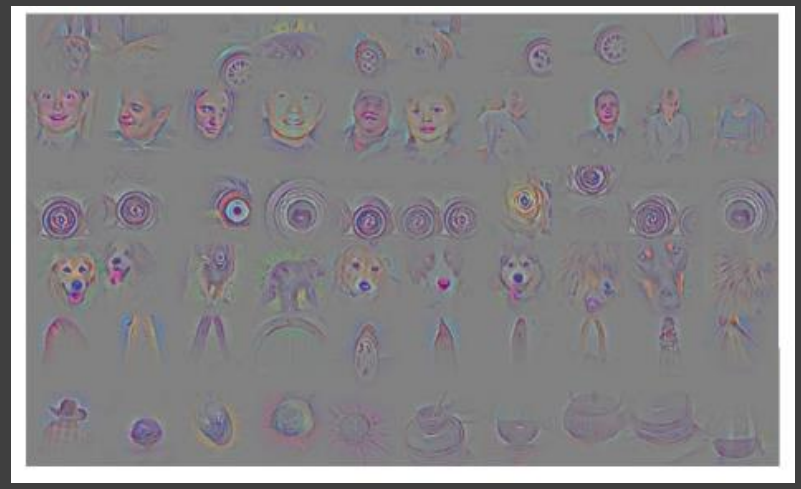
Images come out nicer if you only backprop positive gradients through each ReLU (guided backprop)

Intermediate Features via (guided) backprop

Guided Backprop

Maximally Activating Patches





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

- 입력 이미지에 의존적이지 않은 방법이 있을까? 해당 뉴런을 활성화시킬 수 있는 어떤 “일반적인“ 입력 이미지가 있을까?
- 이에 대한 질문은 “Gradient ascent” 라는 방법이 해답을 줄 수 있다.

Visualizing CNN features : Gradient Ascent

(Guided) backprop: Find the part of an image that a neuron responds to

뉴런이 반응하는 이미지의 일부를 찾는다.

Gradient ascent: Generate a synthetic image that maximally activates a neuron

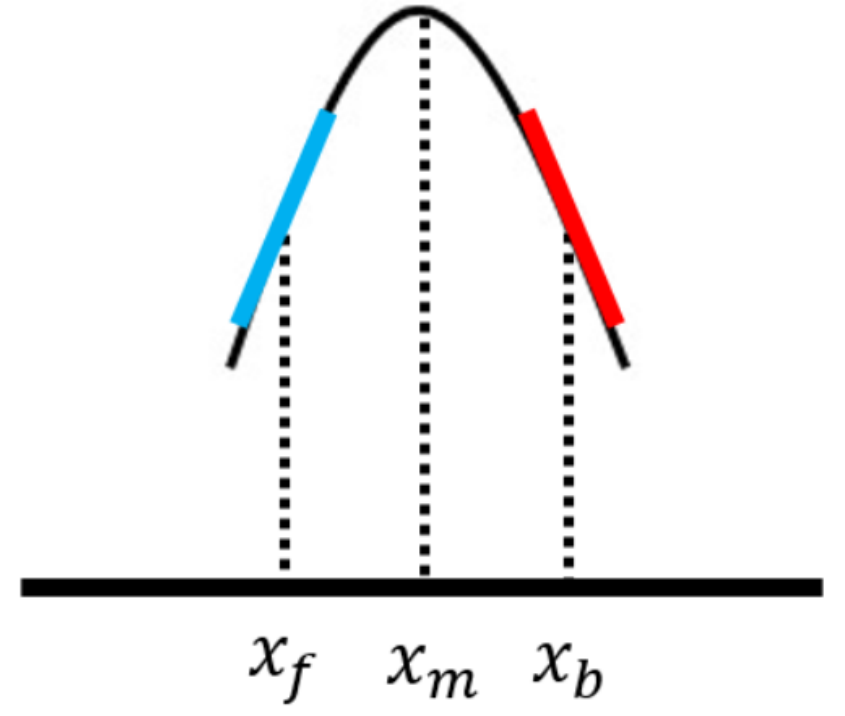
뉴런을 최대로 활성화 시키는 synthetic image 를 만든다.

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

Neuron value

Natural image regularizer

Gradient Ascent 란?



Gradient Ascent 는 말 그대로 해당 모델에서 **Loss가 최대가 되는 Parameter를 찾는 방법**이다.

Gradient Descent 가 Loss 가 최소가 되는 Parameter 를 찾는 방법인 것으로 보았을 때 이 둘을 서로 상반되는 개념으로 보아도 무리가 없는 것처럼 보인다.

그러나 Visualizing CNN Filter를 위해 사용되는 개념과는 조금 차이가 있다. (Backprop 과정에서 Gradient Ascent를 사용한다는 의미가 아니다.)

- Gradient Ascent 가 Deep Learning Visualization 에서 중요한 역할을 차지하는 이유 : Gradient Ascent 를 통해 **중요한 Feature 들을 부각하여 보여줄 수 있기 때문**이다.

- Gradient Ascent를 통해 **뉴런을 최대로 활성화 시키는 합성 이미지**를 만들어낼 수 있고, 이것이 해당 레이어에서 일어나는 일을 시각화 하는 데에 단서를 제공하게 된다.

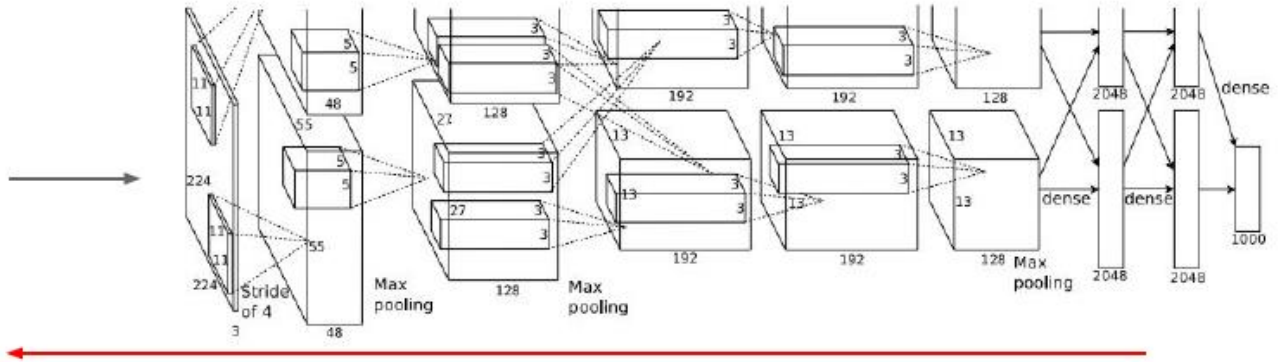
Visualizing CNN features : Gradient Ascent

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

score for class c (before Softmax)

1. Initialize image to zeros

zero image



Repeat:

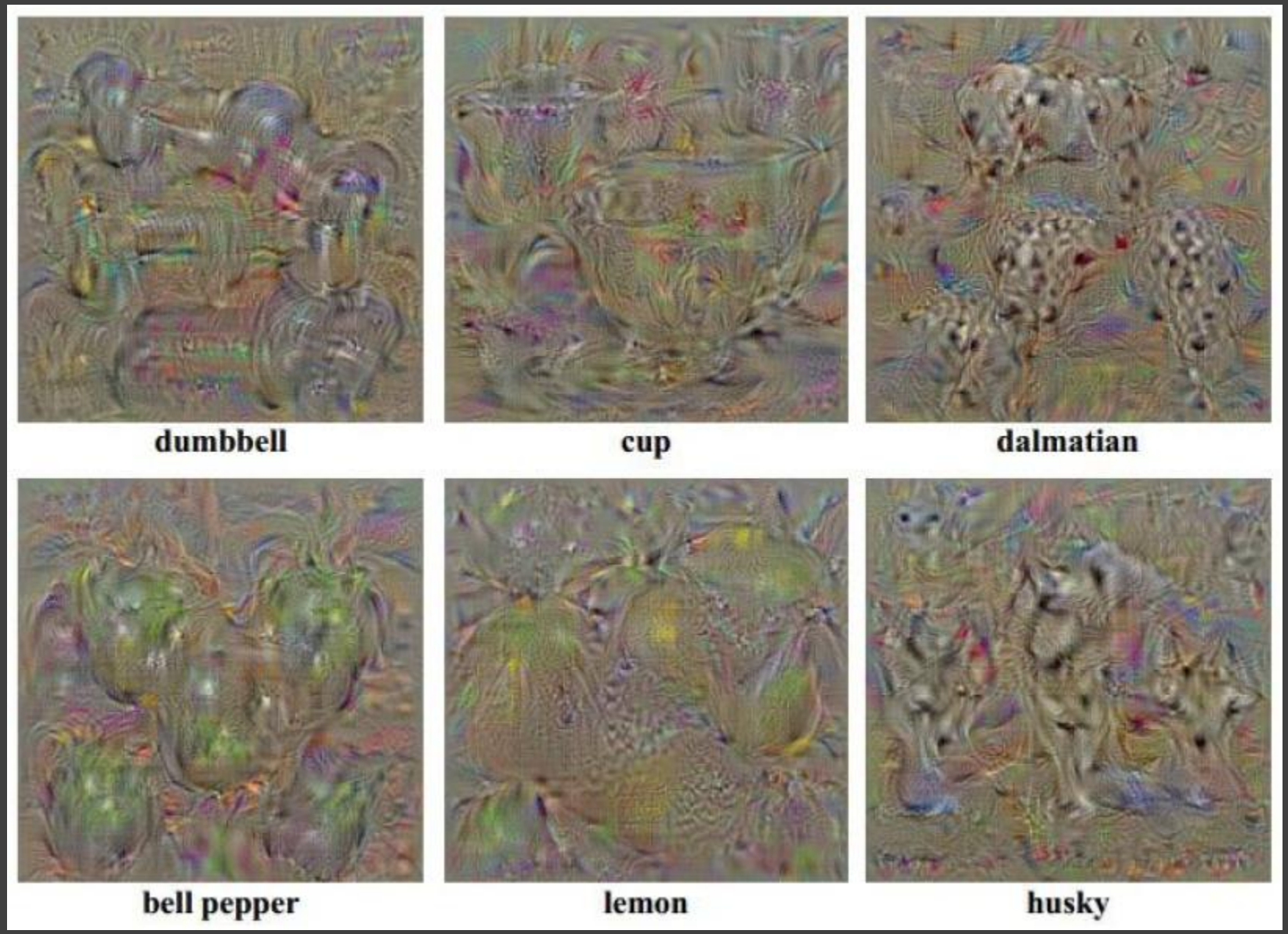
2. Forward image to compute current scores
3. Backprop to get gradient of neuron value with respect to image pixels
4. Make a small update to the image

Visualizing CNN features : Gradient Ascent

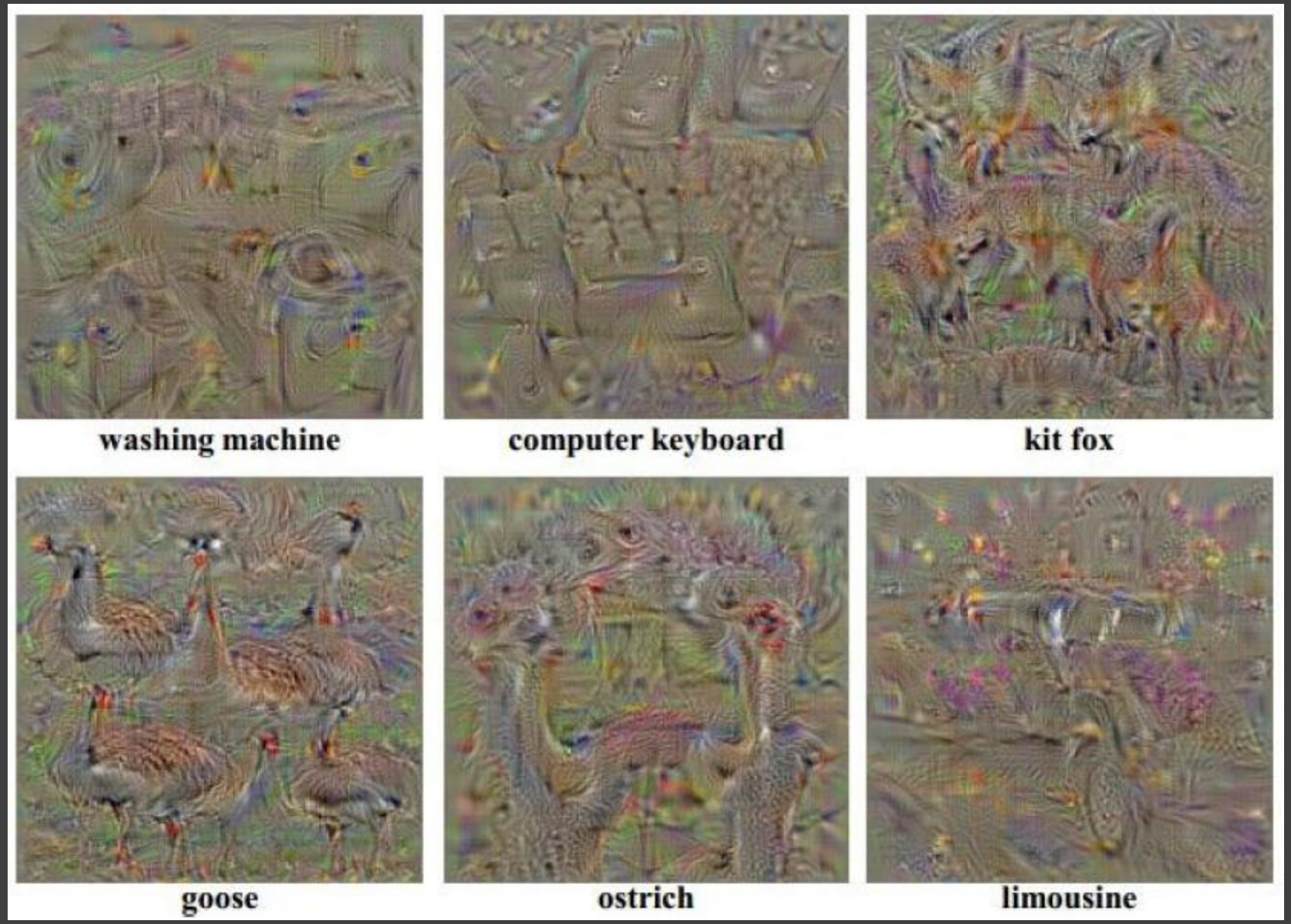
$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

Simple regularizer: Penalize L2 norm of generated image

여기에서는 단순하게 생성된 이미지에 대한 L2 norm 을 계산해서 더해준다. (왜 L2 norm 사용하는가에 대해 큰 의미는 없다.)



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



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

Visualizing CNN features : Gradient Ascent

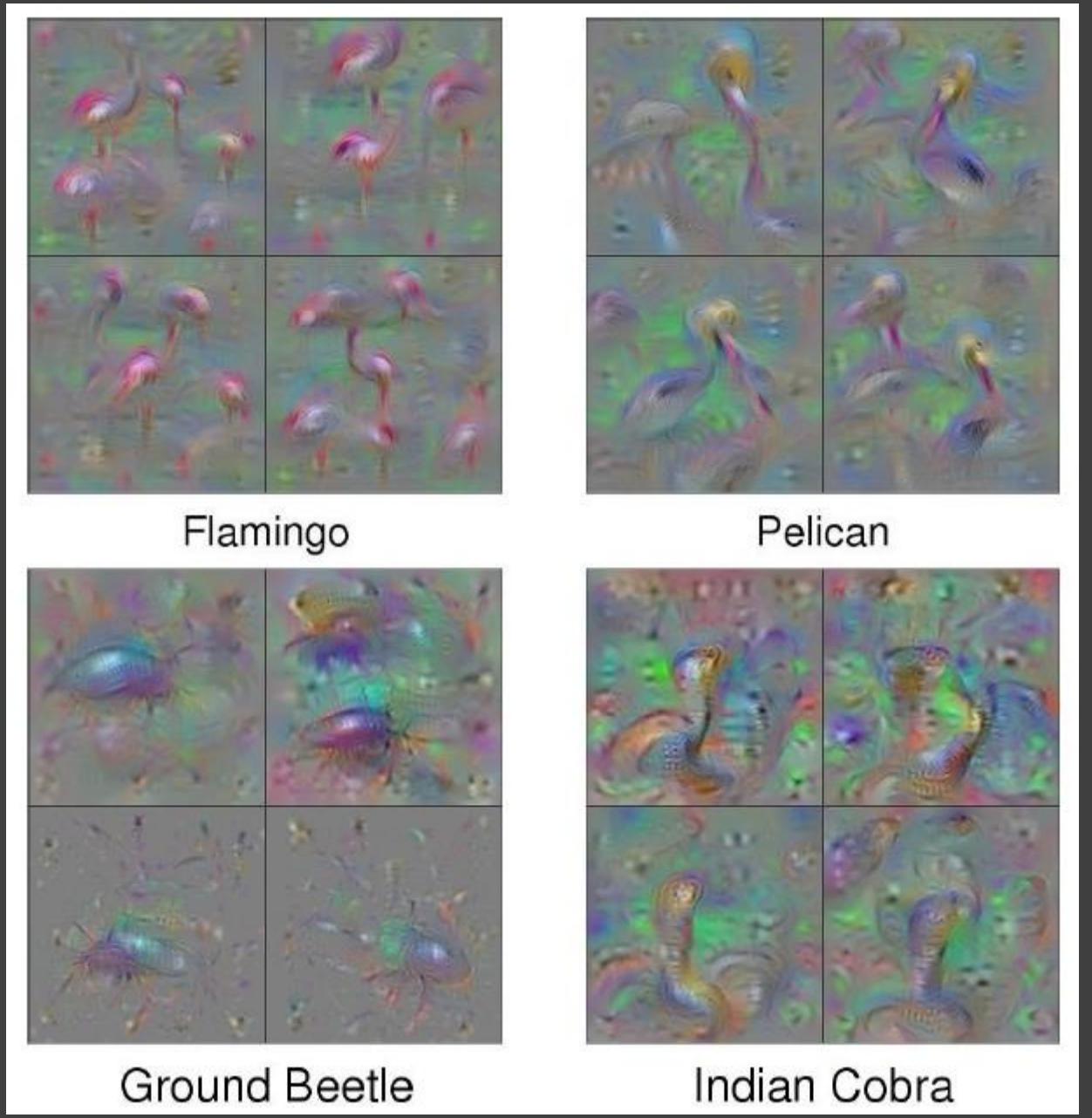
$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

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

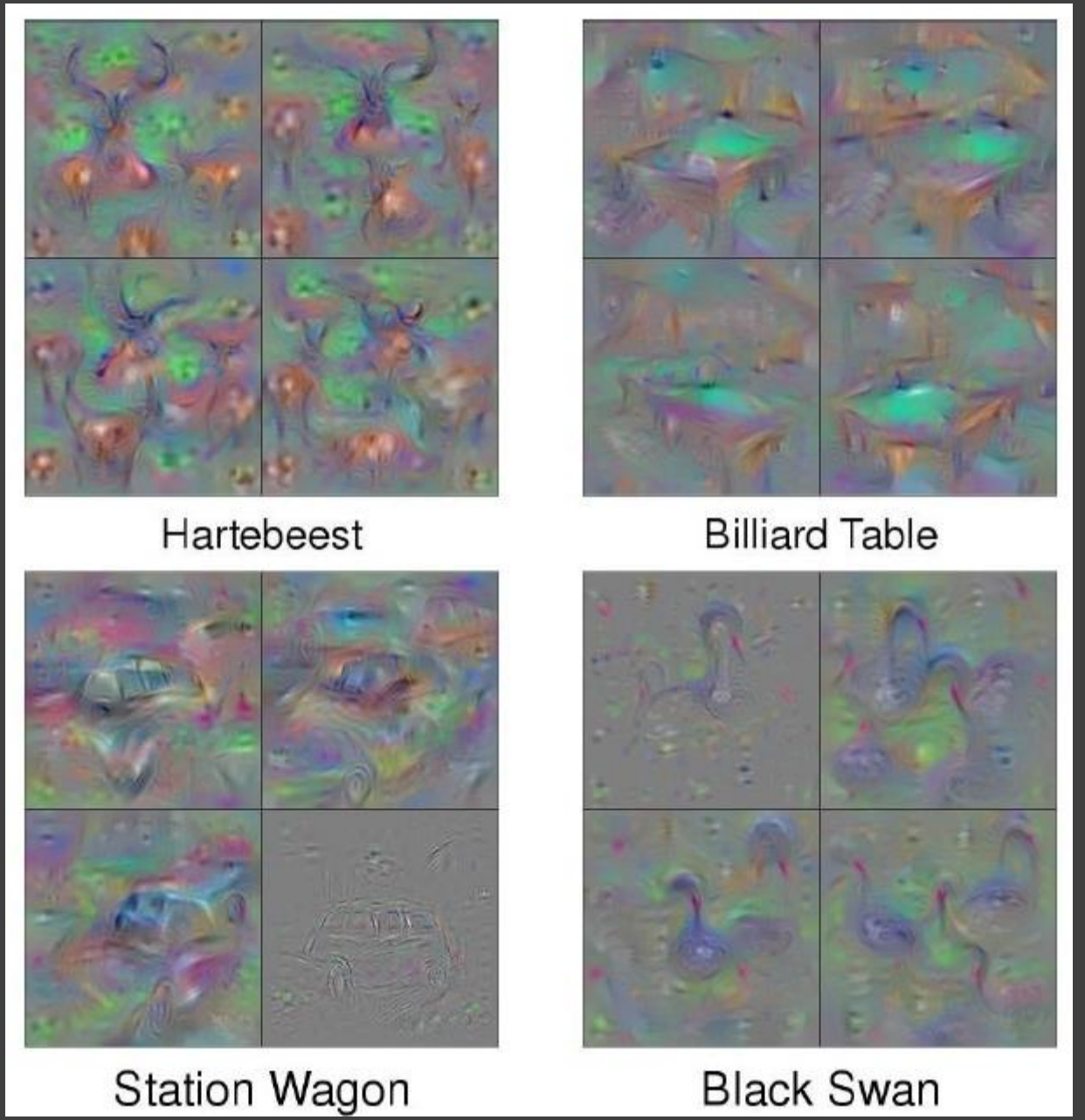
(1) Gaussian blur image
(2) Clip pixels with small values to 0
(3) Clip pixels with small gradients to 0

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

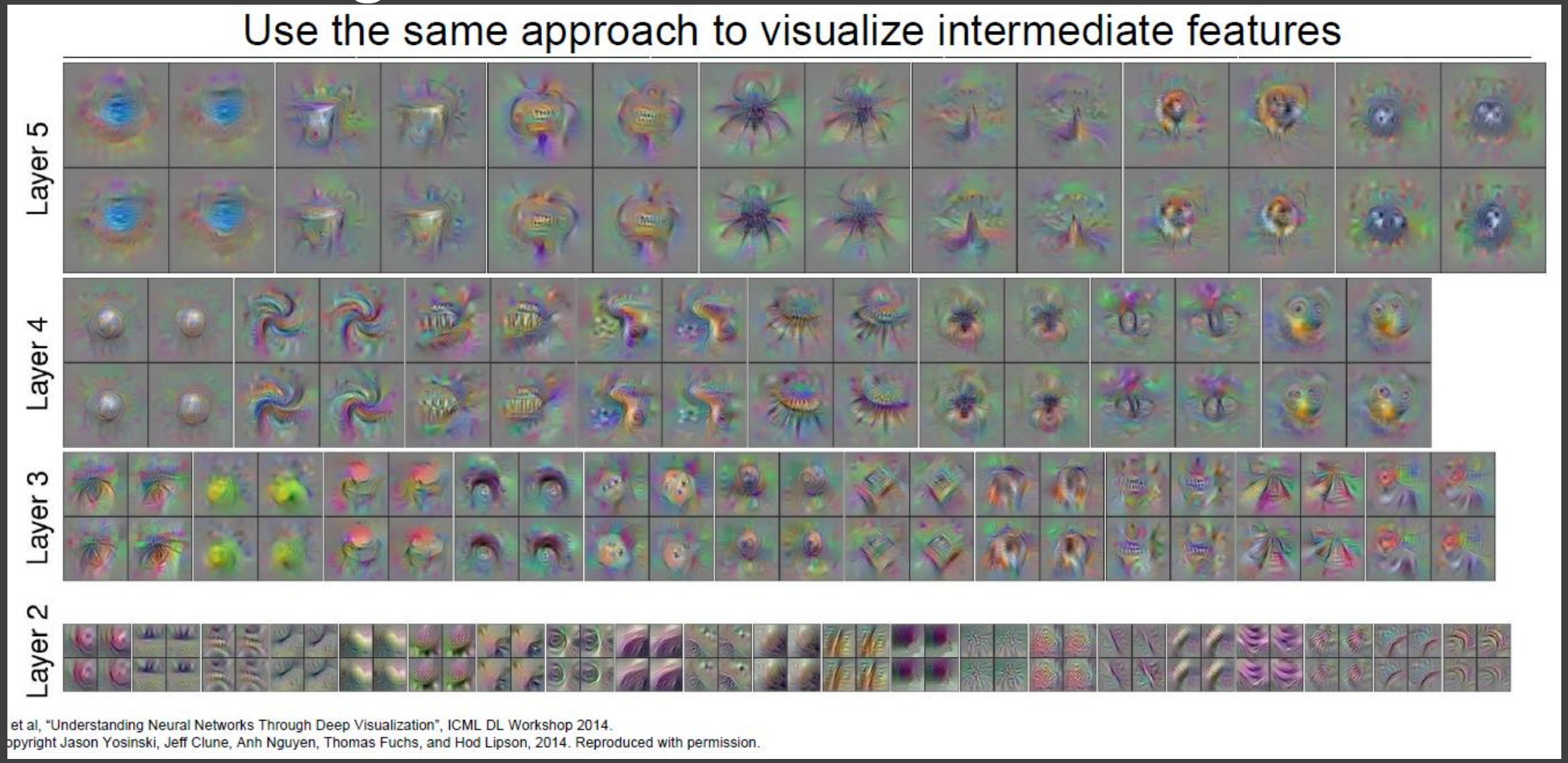
With Better regularizer



With Better regularizer



Visualizing CNN features : Gradient Ascent

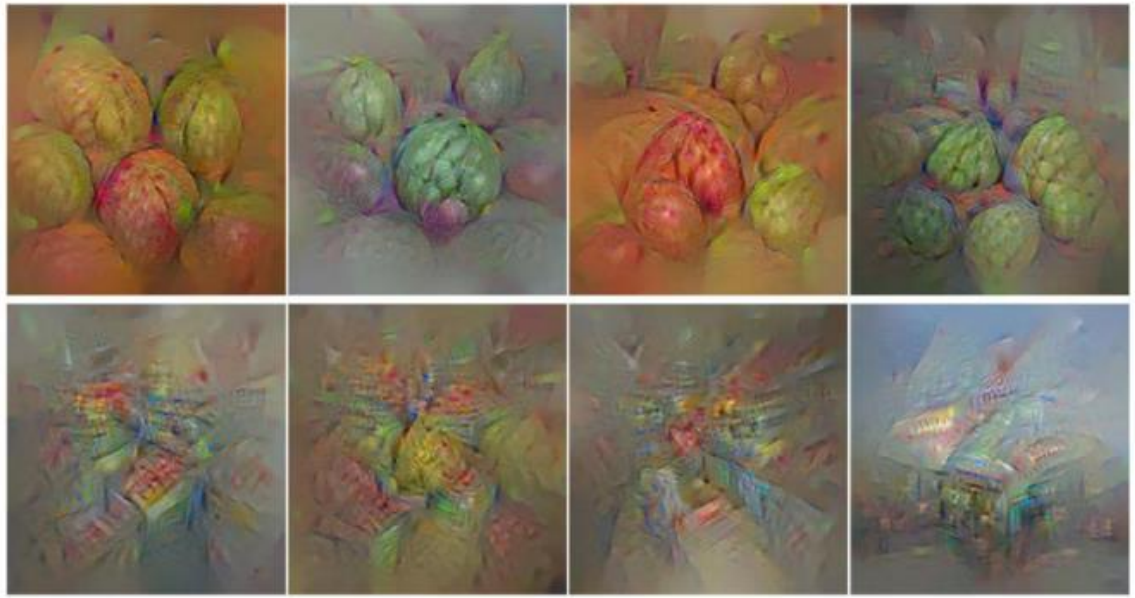


et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.

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

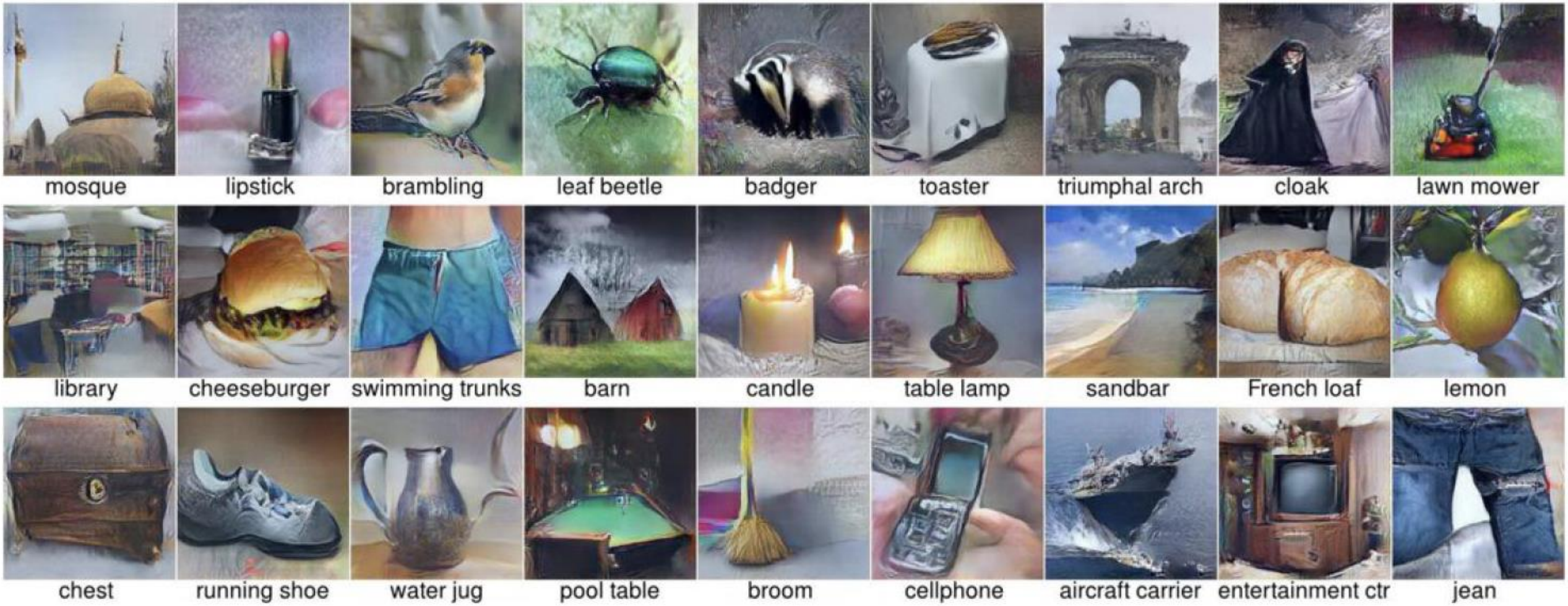
Nguyen et al, “Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks”, ICML Visualization for Deep Learning Workshop 2016.

Visualizing CNN features : Gradient Ascent

aceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks", ICML Visualization for Deep Learning Workshop 2016.
h Nguyen, Jason Yosinski, and Jeff Clune, 2016; reproduced with permission.

Visualizing CNN features : Gradient Ascent

Optimize in FC6 latent space instead of pixel space:

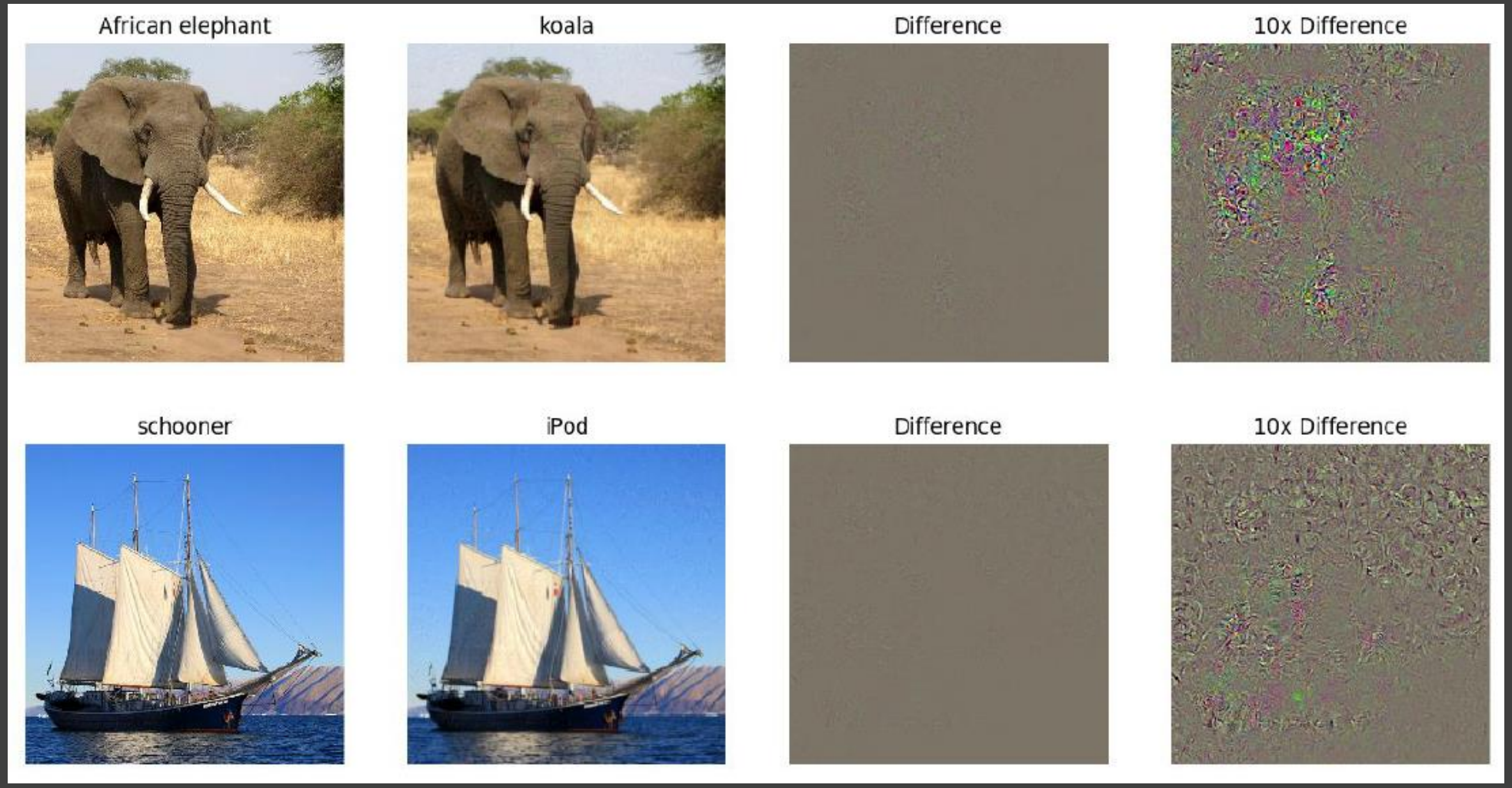


Nguyen et al, "Synthesizing the preferred inputs for neurons in neural networks via deep generator networks," NIPS 2016

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

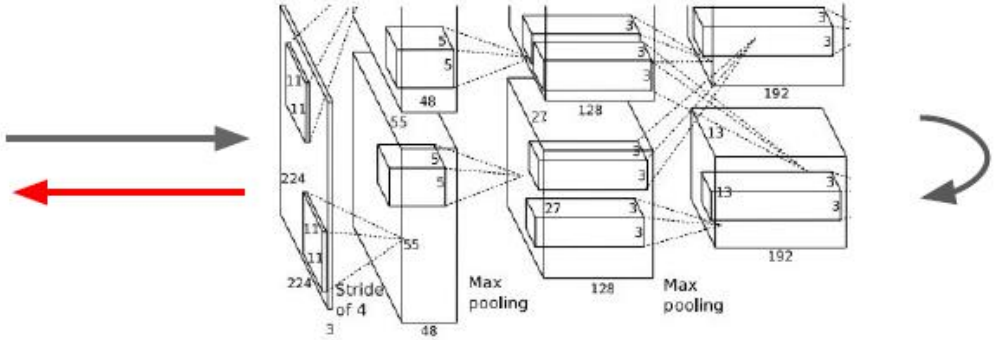
Fooling Images / Adversarial Examples



3. Fun

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:

$$I^* = \arg\max_I \sum_i f_i(I)^2$$

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

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

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

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

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

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

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

Code is very simple but it uses a couple tricks:

Jitter image

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

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

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

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

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

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

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

Code is very simple but it uses a couple tricks:

Jitter image

L1 Normalize gradients

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

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

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

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

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

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

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

Code is very simple but it uses a couple tricks:

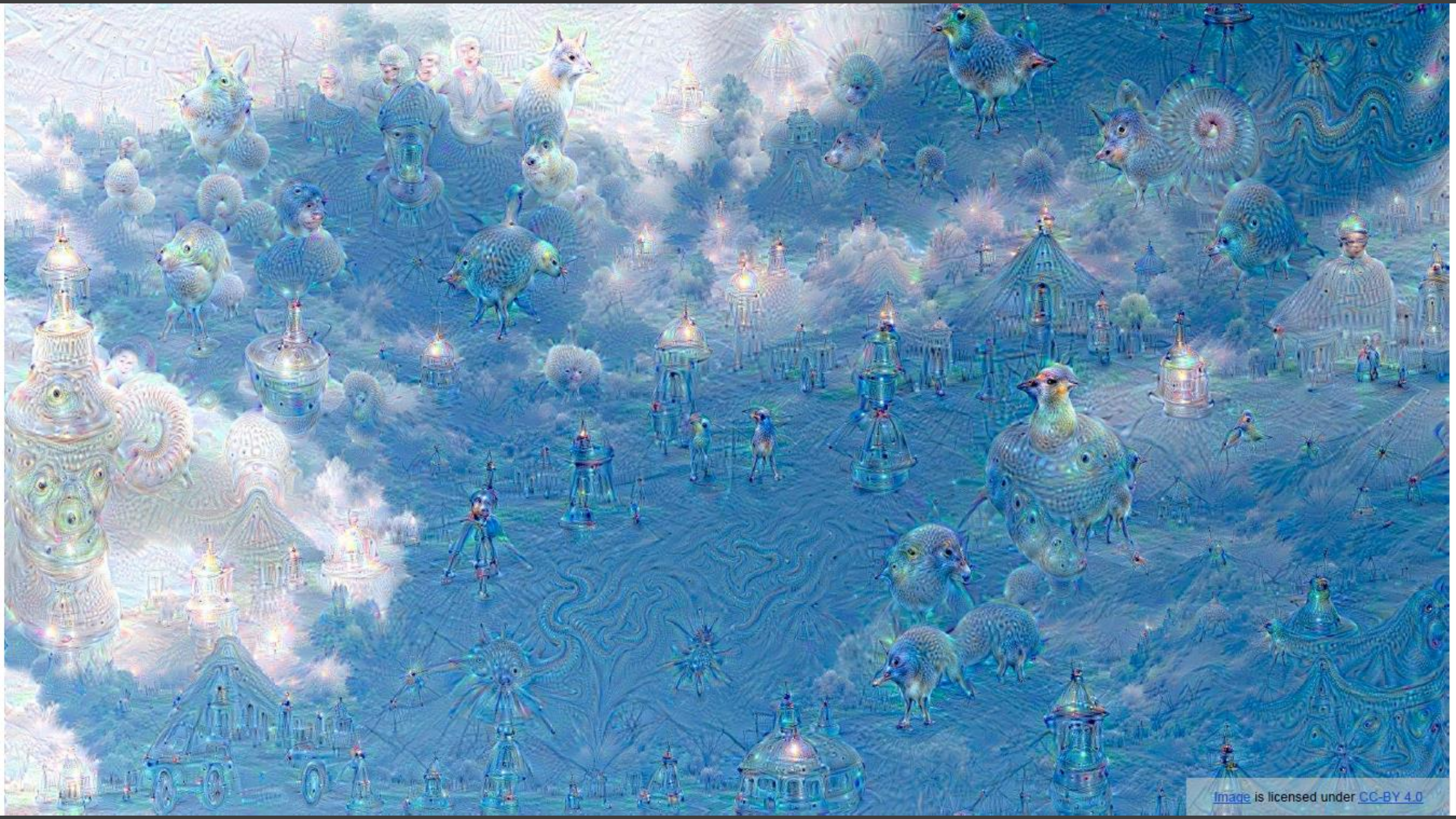
Jitter image

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



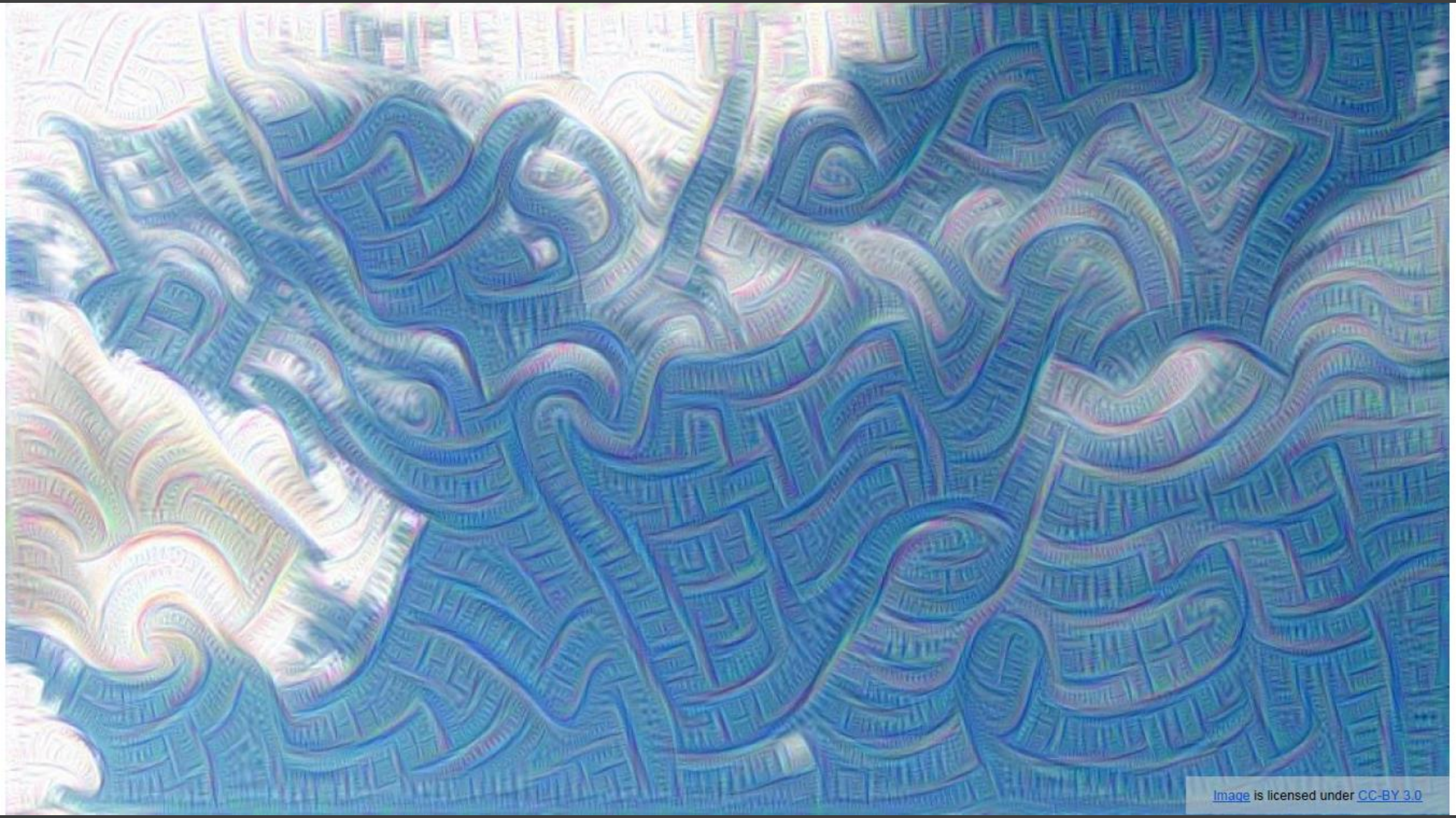
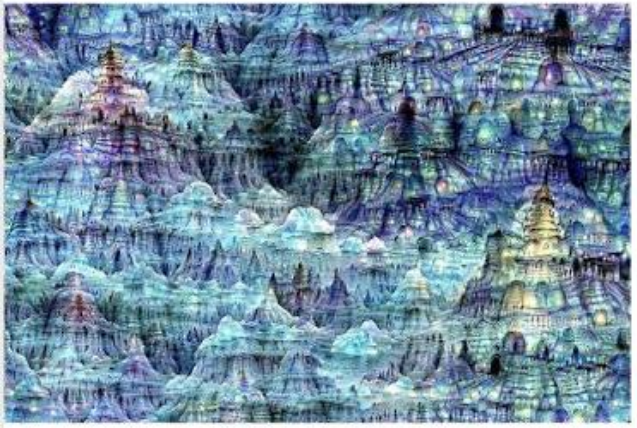
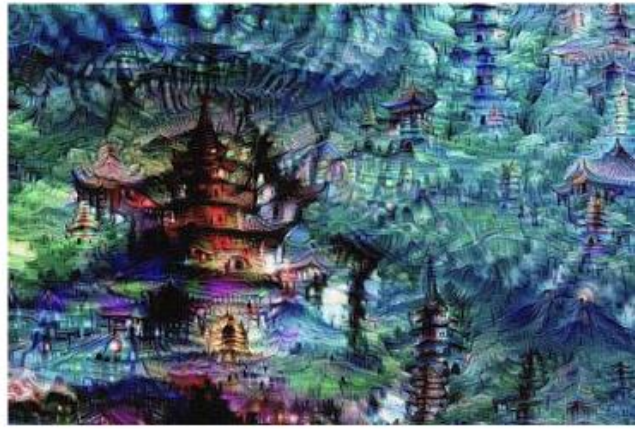



Image is licensed under CC-BY 3.0





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

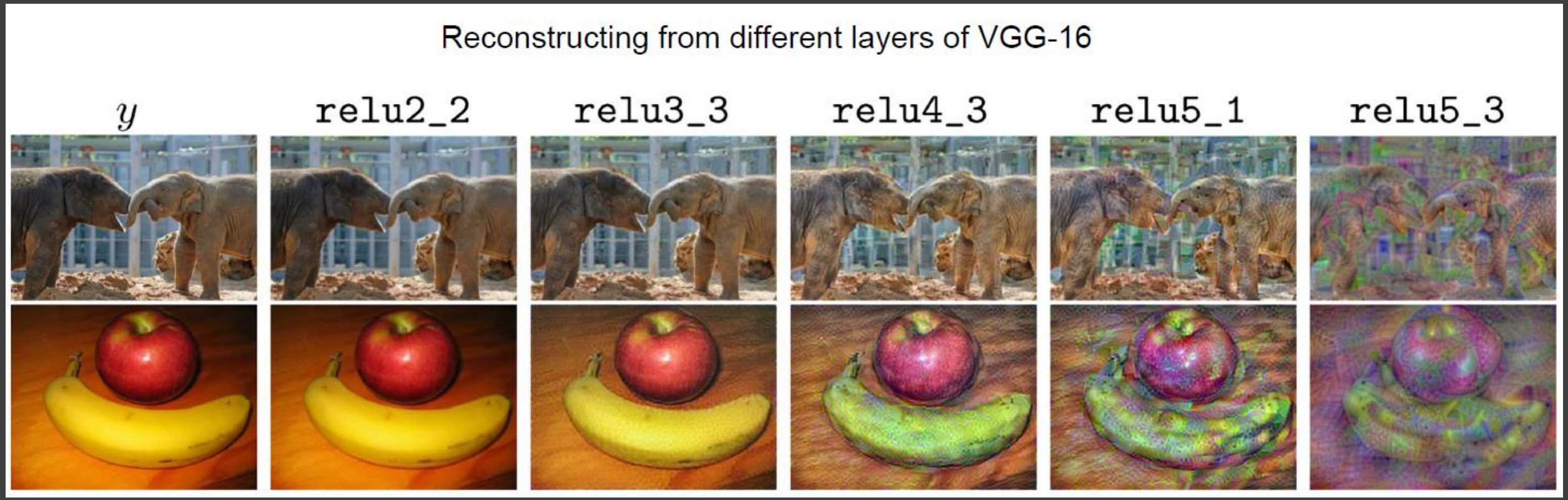
Features of new image

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

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

Total Variation regularizer
(encourages spatial smoothness)

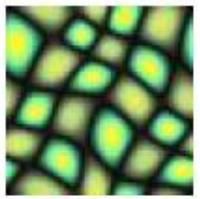
Feature Inversion



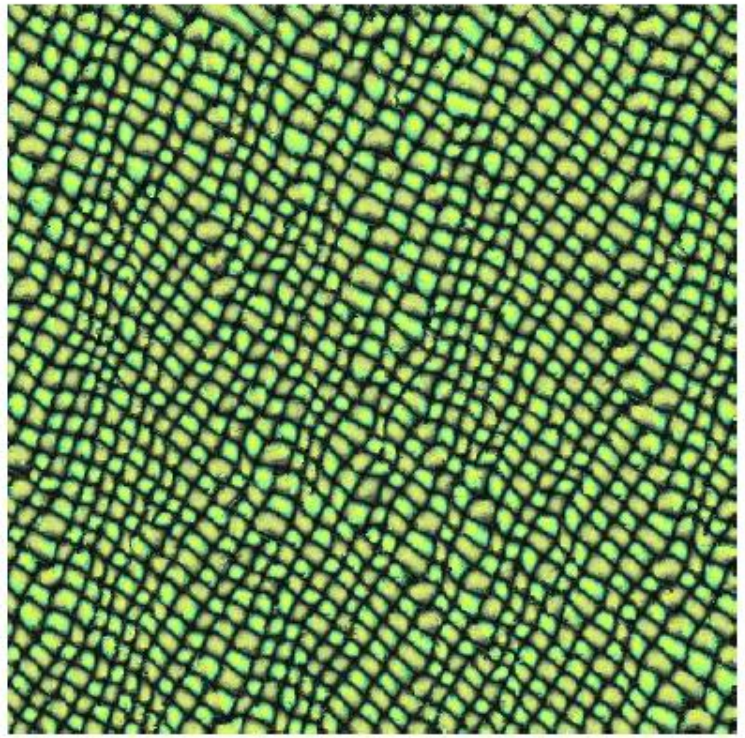
Mahendran and Vedaldi, "Understanding Deep Image Representations by Inverting Them", CVPR 2015
Figure from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016. Copyright Springer, 2016.
Reproduced for educational purposes.

Texture Synthesis

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



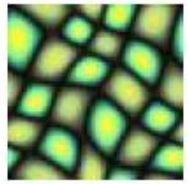
Input

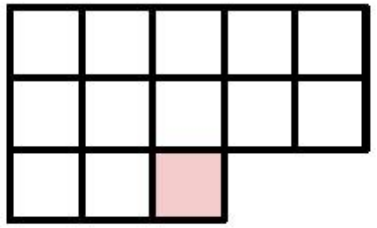


Output

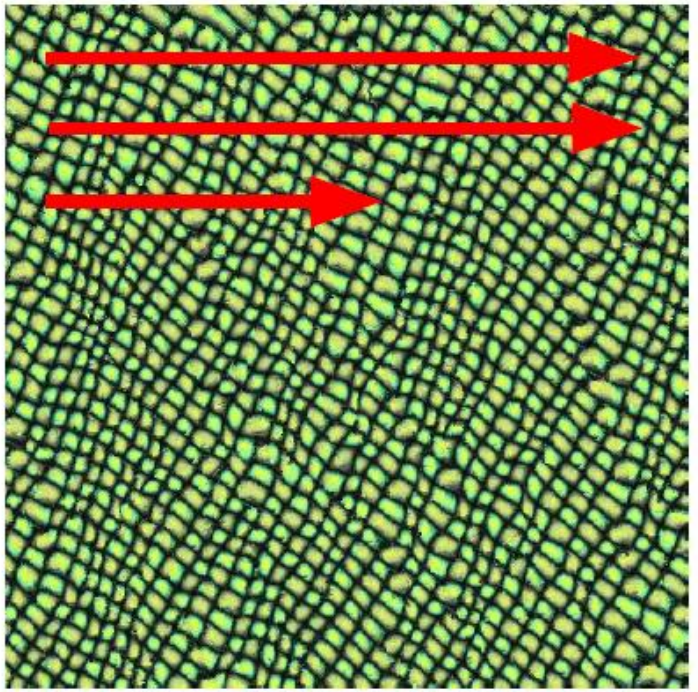
Wei and Levoy, "Fast Texture Synthesis using Tree-structured Vector Quantization", SIGGRAPH 2000
Efros and Leung, "Texture Synthesis by Non-parametric Sampling", ICCV 1999

Texture Synthesis : Nearest Neighbor

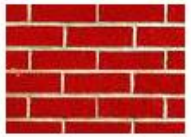


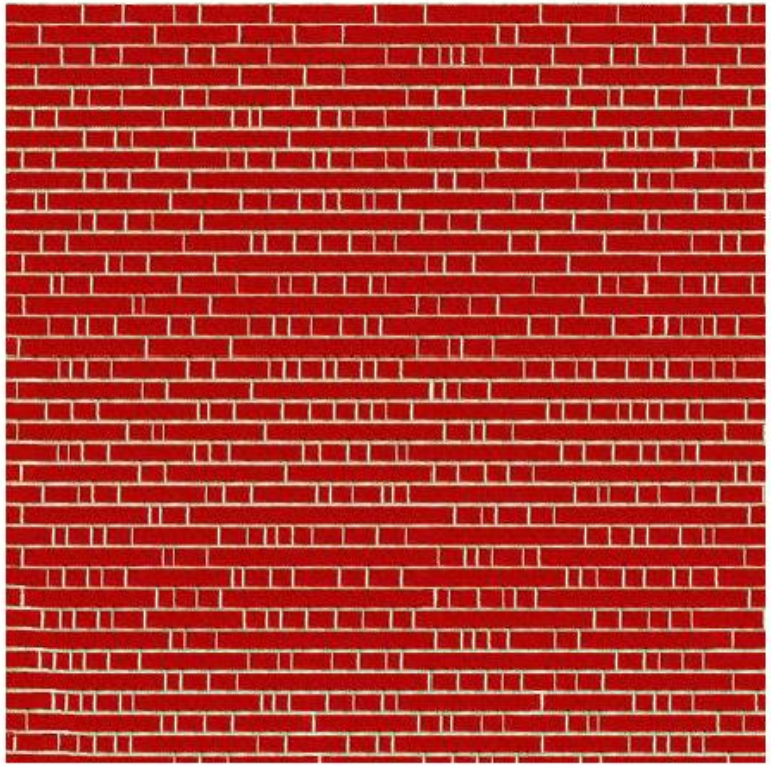


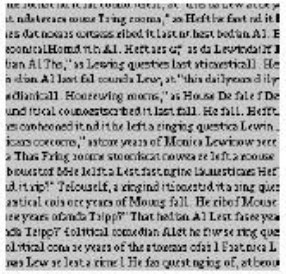
Generate pixels one at a time in scanline order; form neighborhood of already generated pixels and copy nearest neighbor from input

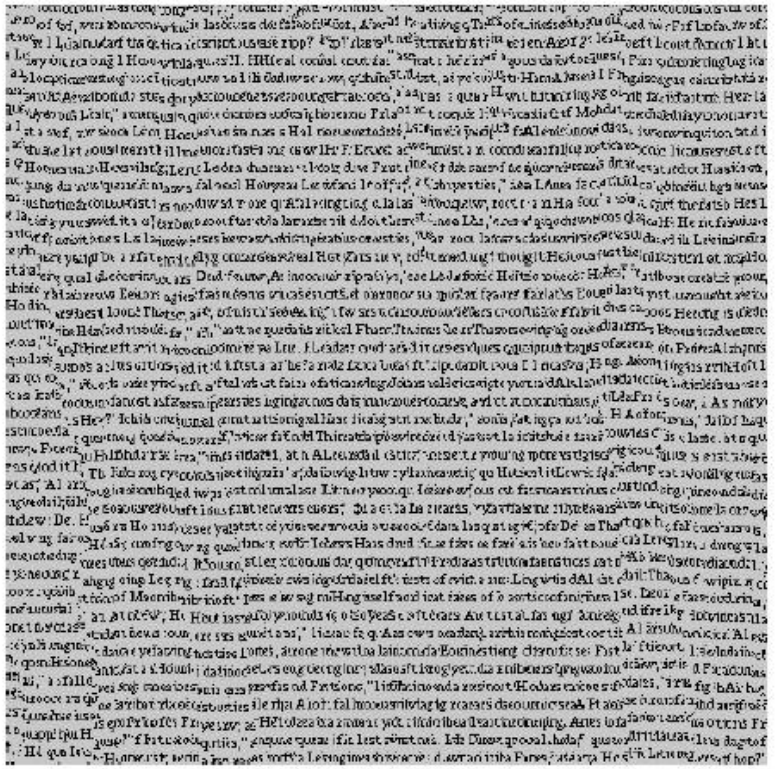


Texture Synthesis : Nearest Neighbor



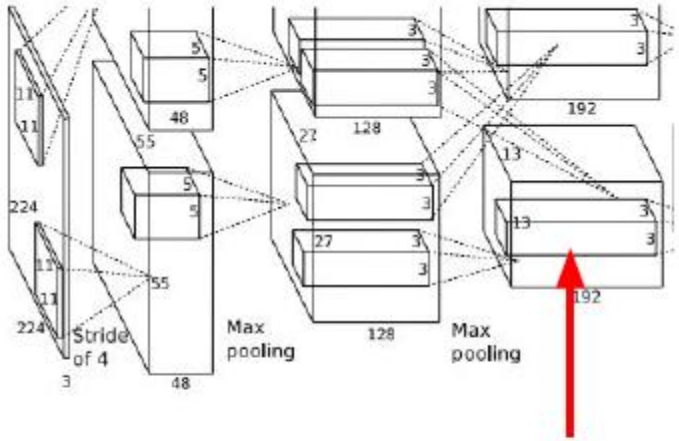


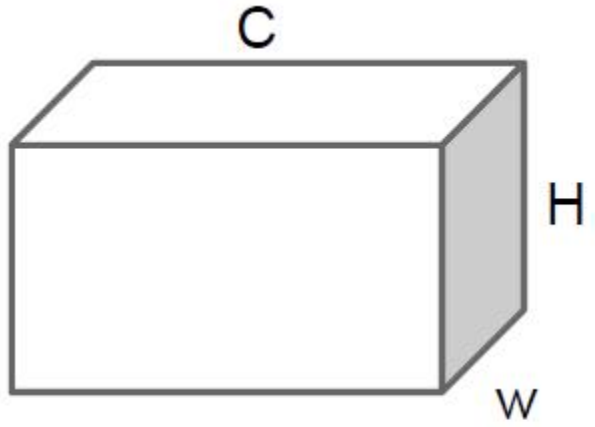




Neural Texture Synthesis : Gram Matrix

This image is in the public domain.

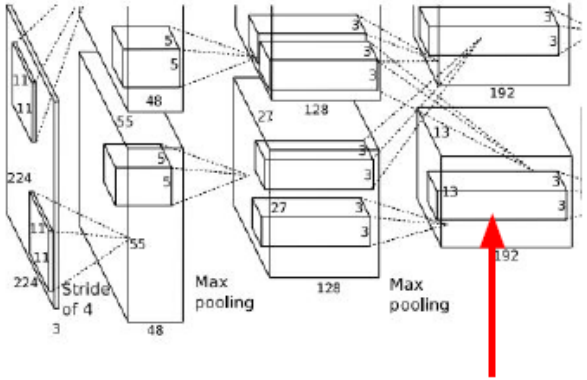


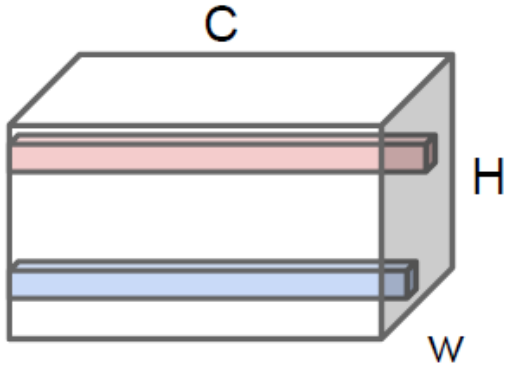


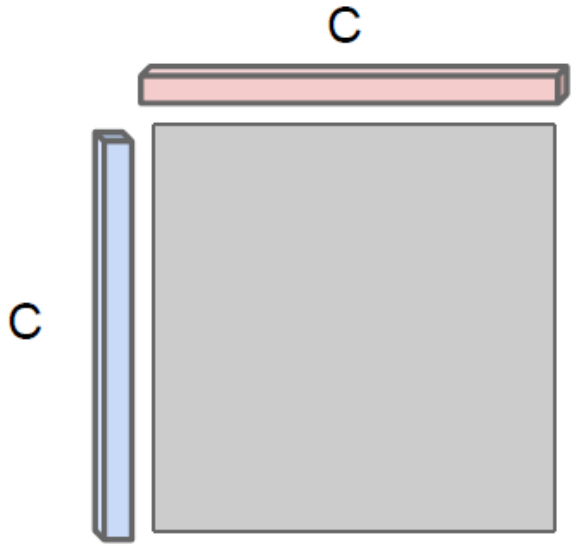
Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

Neural Texture Synthesis : Gram Matrix

This image is in the public domain.





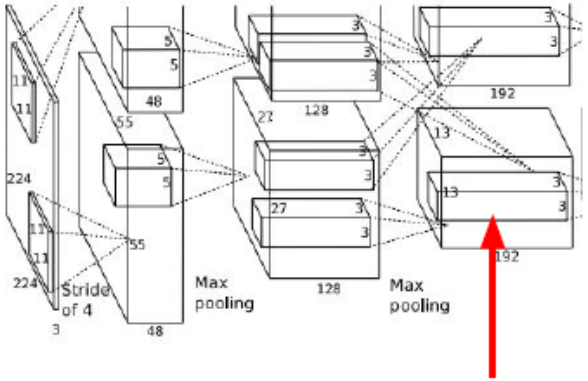


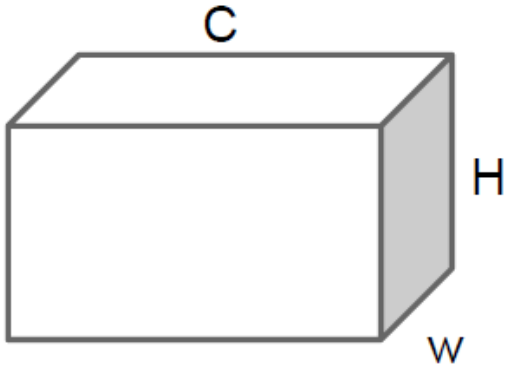
Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

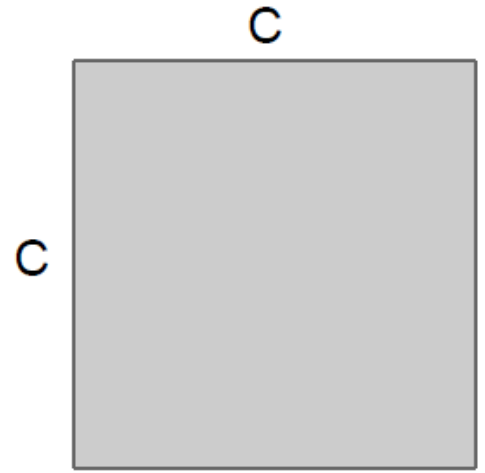
Outer product of two C-dimensional vectors gives C x C matrix measuring co-occurrence

Neural Texture Synthesis : Gram Matrix

This image is in the public domain.







Gram Matrix

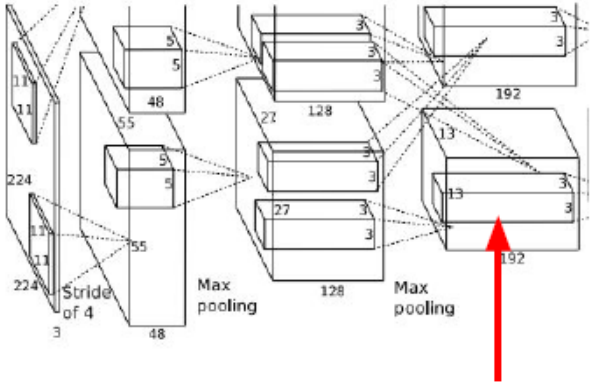
Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

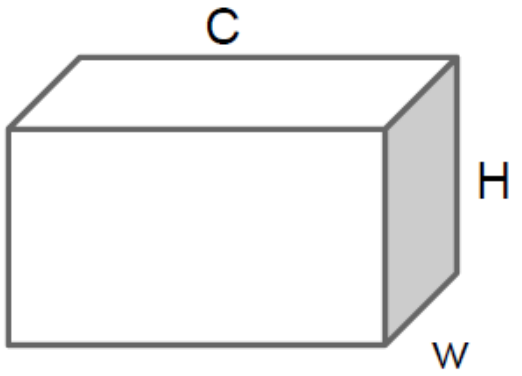
Outer product of two C-dimensional vectors gives C x C matrix measuring co-occurrence

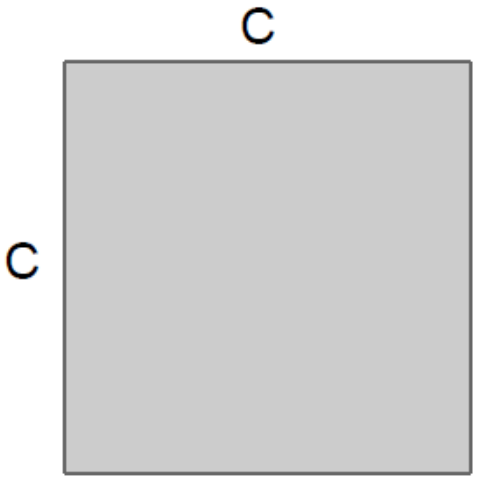
Average over all HW pairs of vectors, giving **Gram matrix** of shape C x C

Neural Texture Synthesis : Gram Matrix

This image is in the public domain.







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

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

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

Efficient to compute; reshape features from

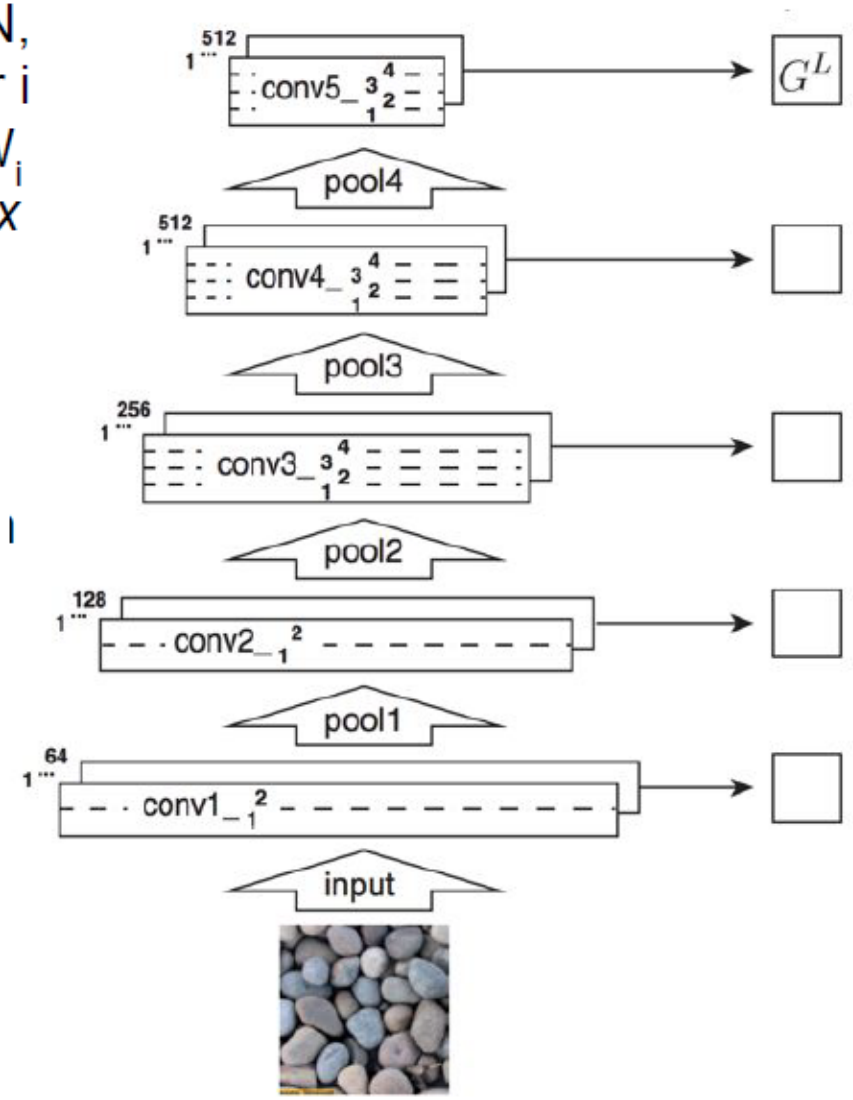
C x H x W to =C x HW

then compute $G = FF^T$

Neural Texture Synthesis : Gram Matrix

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

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



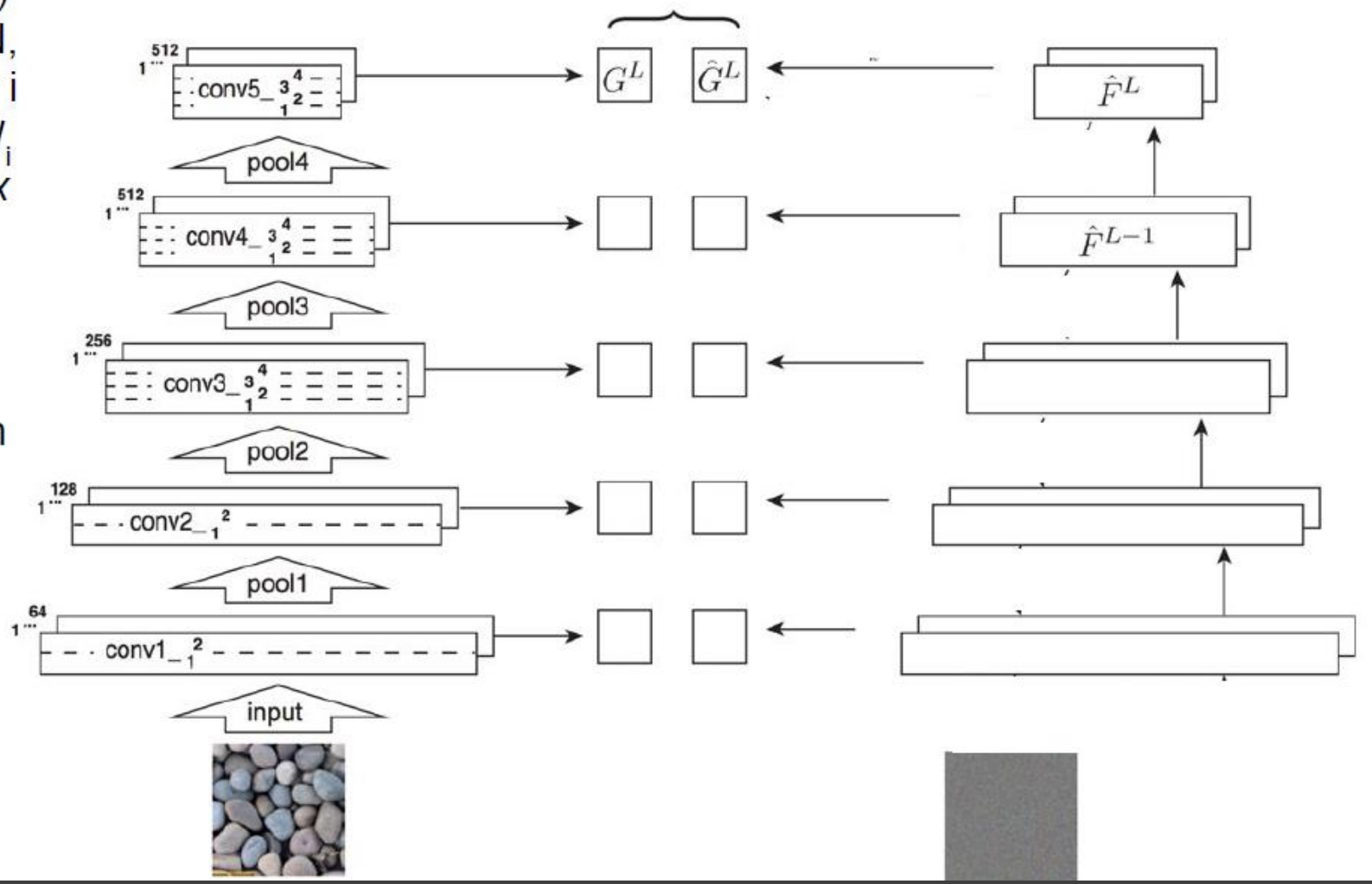
Gatys, Ecker, and Bethge, "Texture Synthesis Using Convolutional Neural Networks", NIPS 2015
Figure copyright Leon Gatys, Alexander S. Ecker, and Matthias Bethge, 2015. Reproduced with permission.

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
6. Compute loss: weighted sum of L2 distance between Gram matrices

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

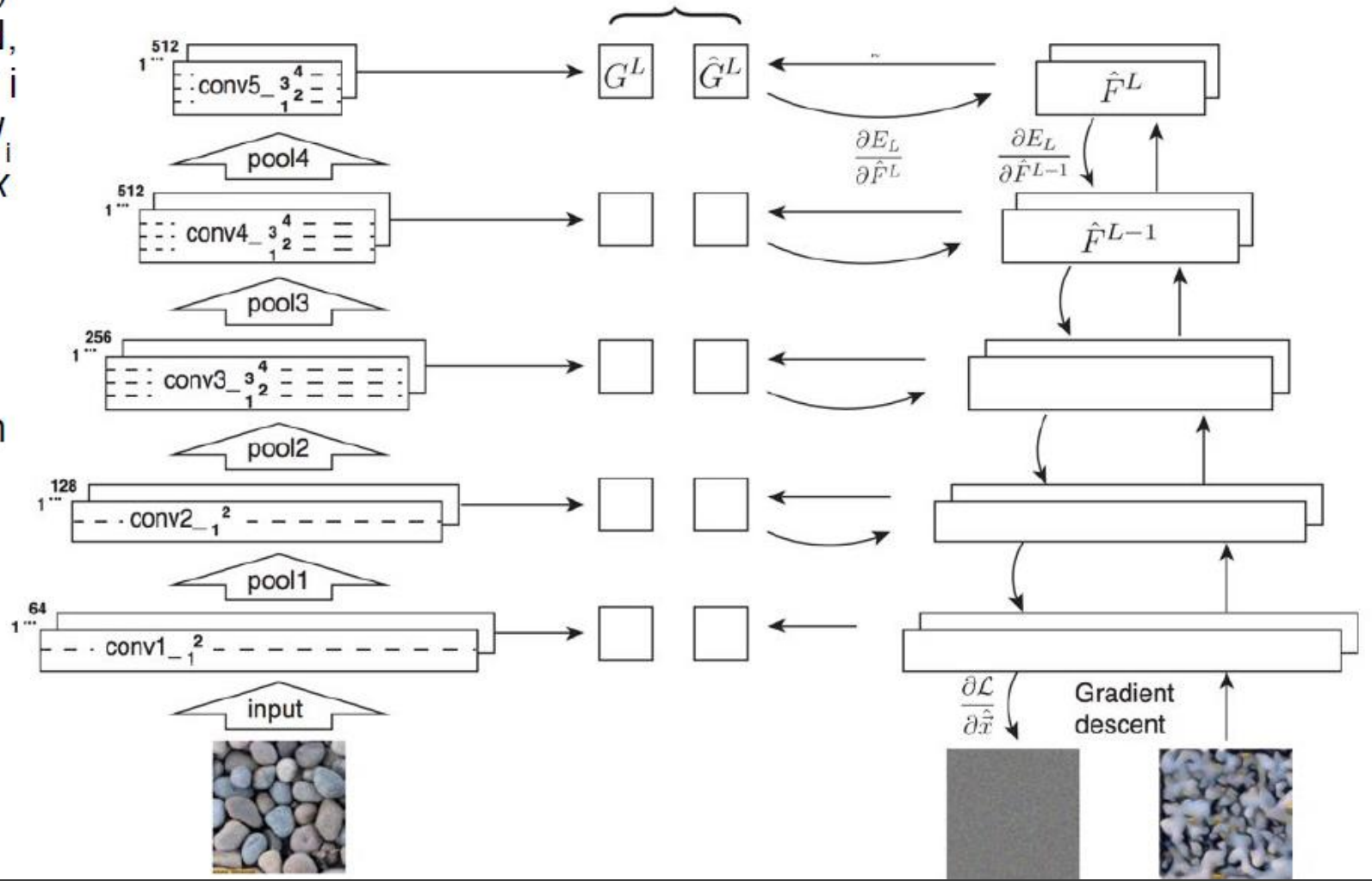


1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the *Gram matrix* giving outer product of features:

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

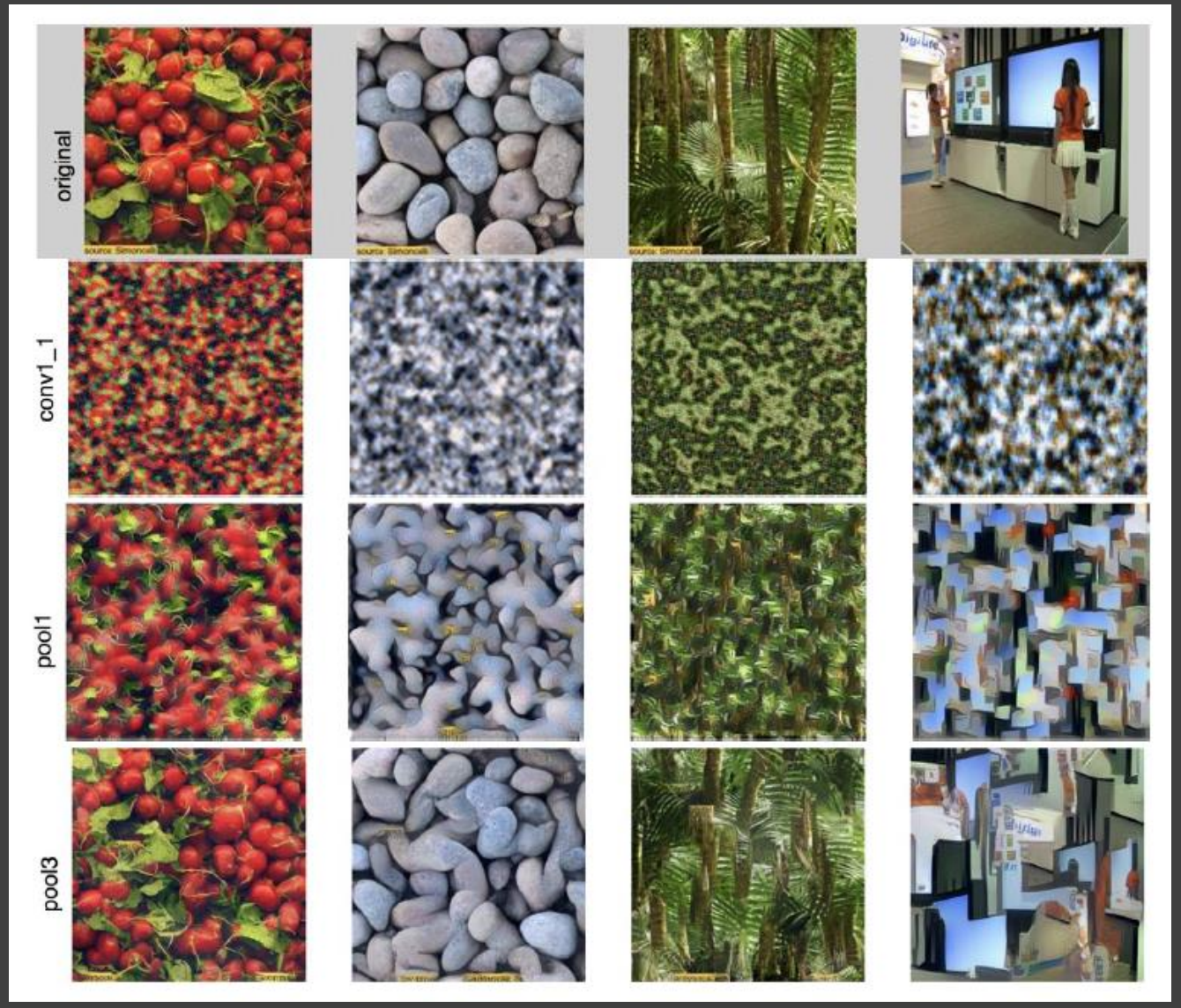
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5

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



Neural Texture Synthesis

Reconstructing texture from higher layers recovers larger features from the input texture



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

Neural Texture Synthesis : Texture = Artwork

Texture synthesis (Gram reconstruction)

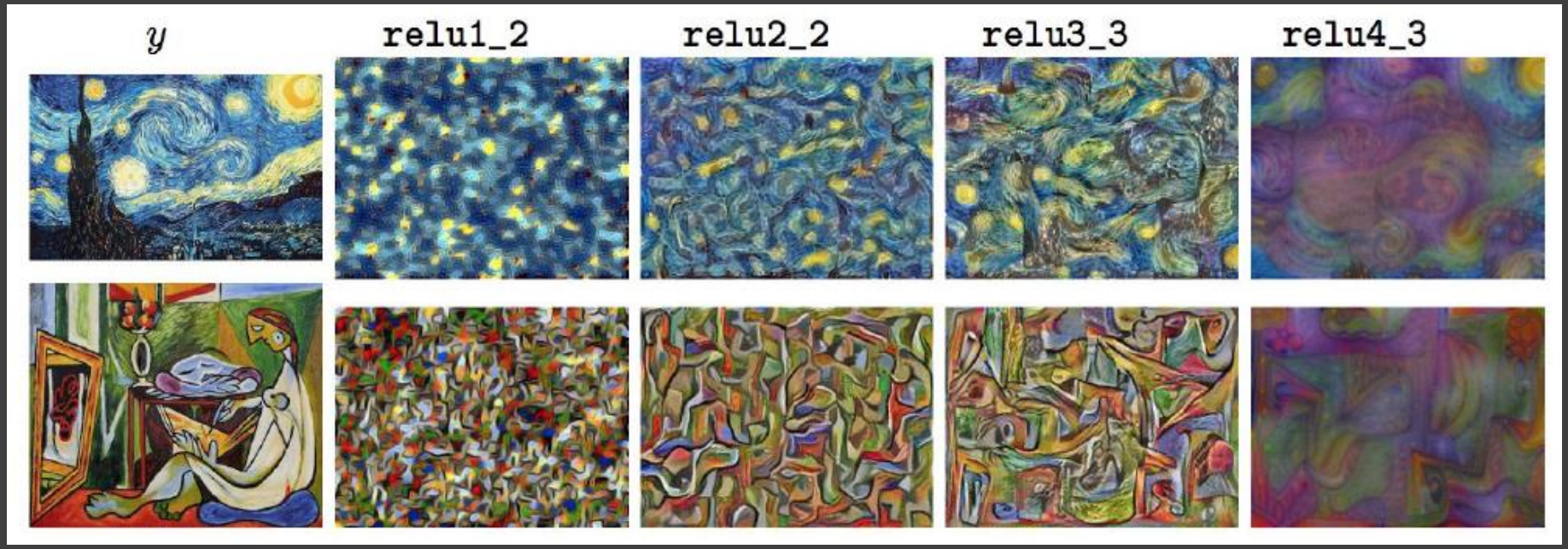
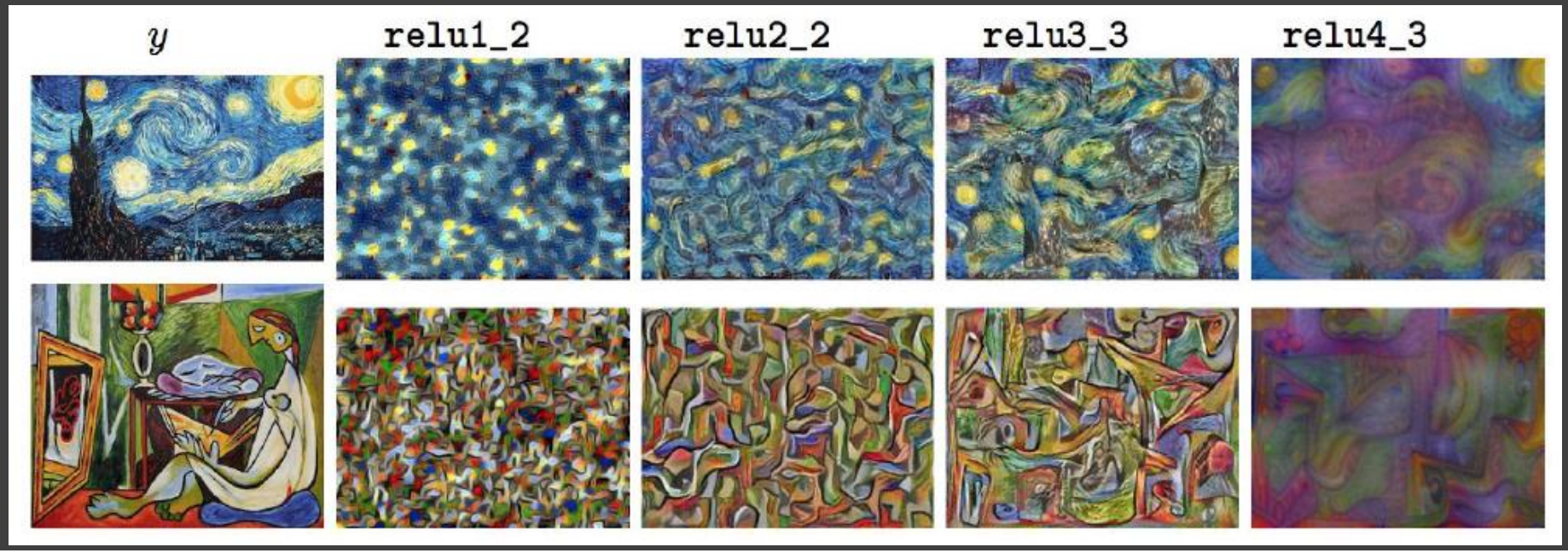


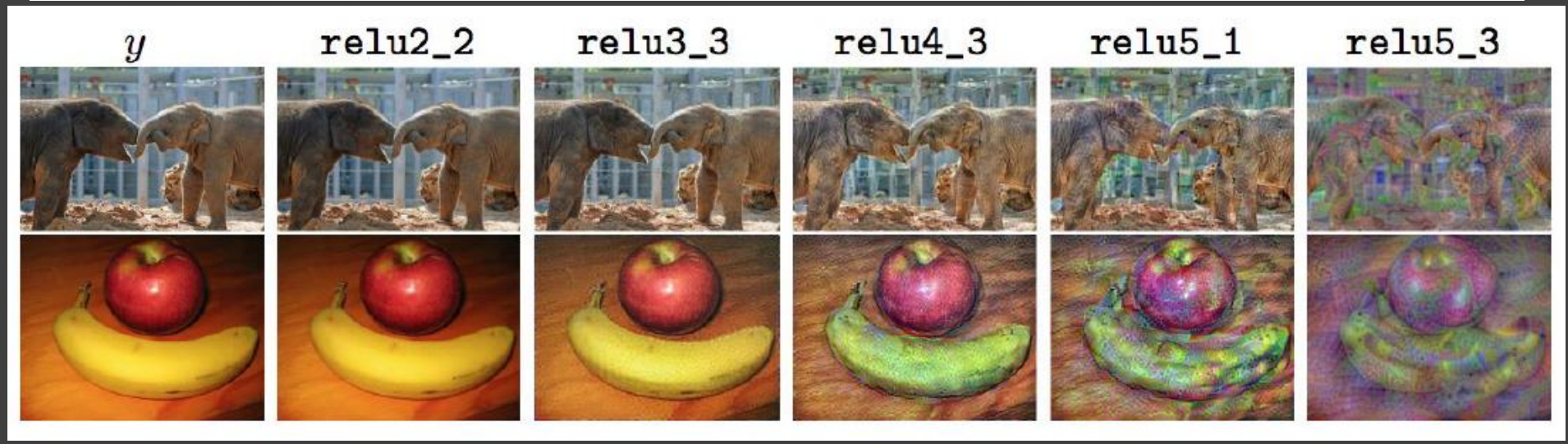
Figure from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016. Copyright Springer, 2016. Reproduced for educational purposes.

Neural Style Transfer : Feature + Gram Reconstruction

Texture synthesis (Gram reconstruction)



Feature reconstruction



Neural Style Transfer

Content Image

This image is licensed under CC-BY 3.0

+

Style Image

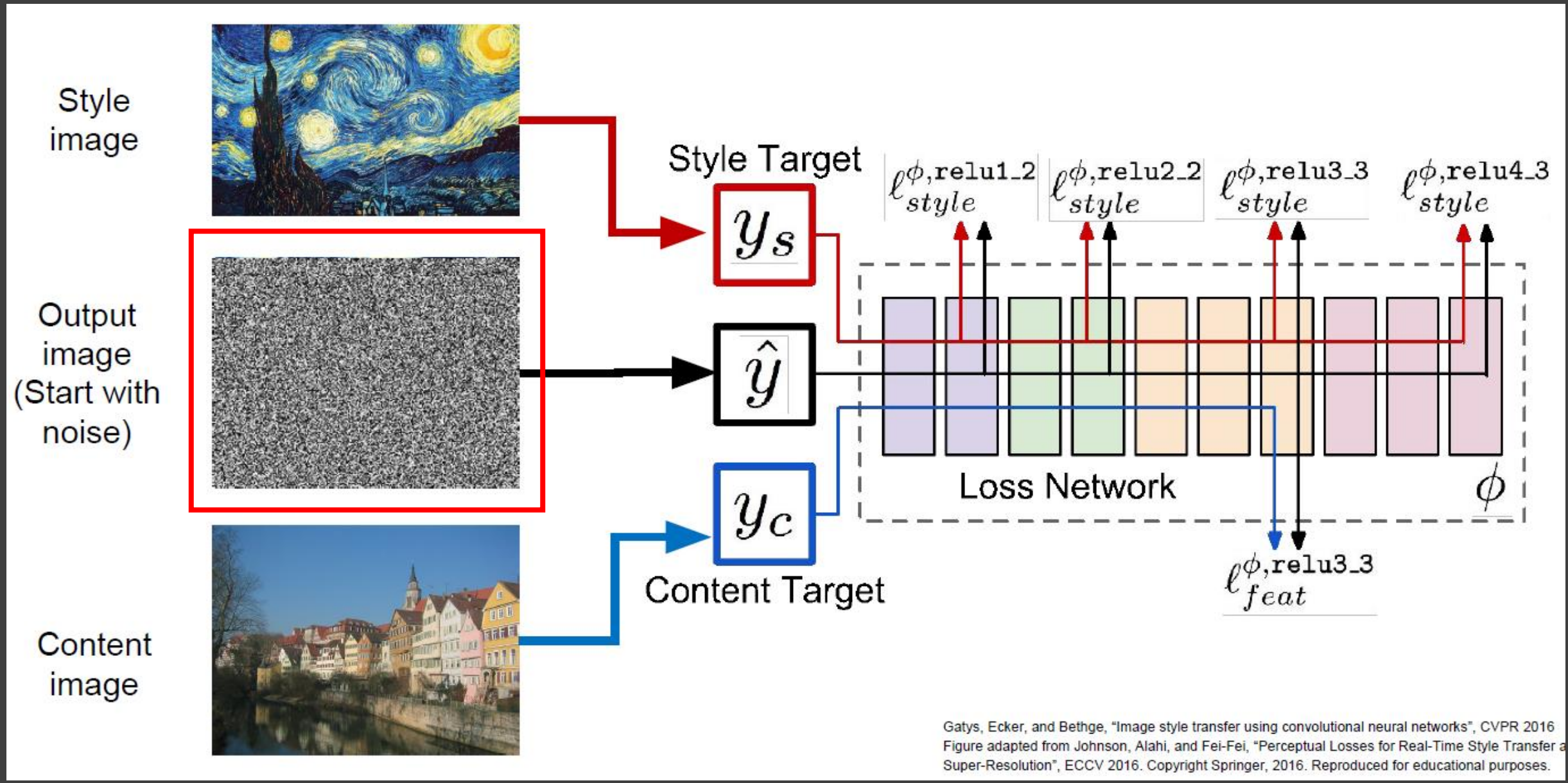
Starry Night by Van Gogh is in the public domain

=

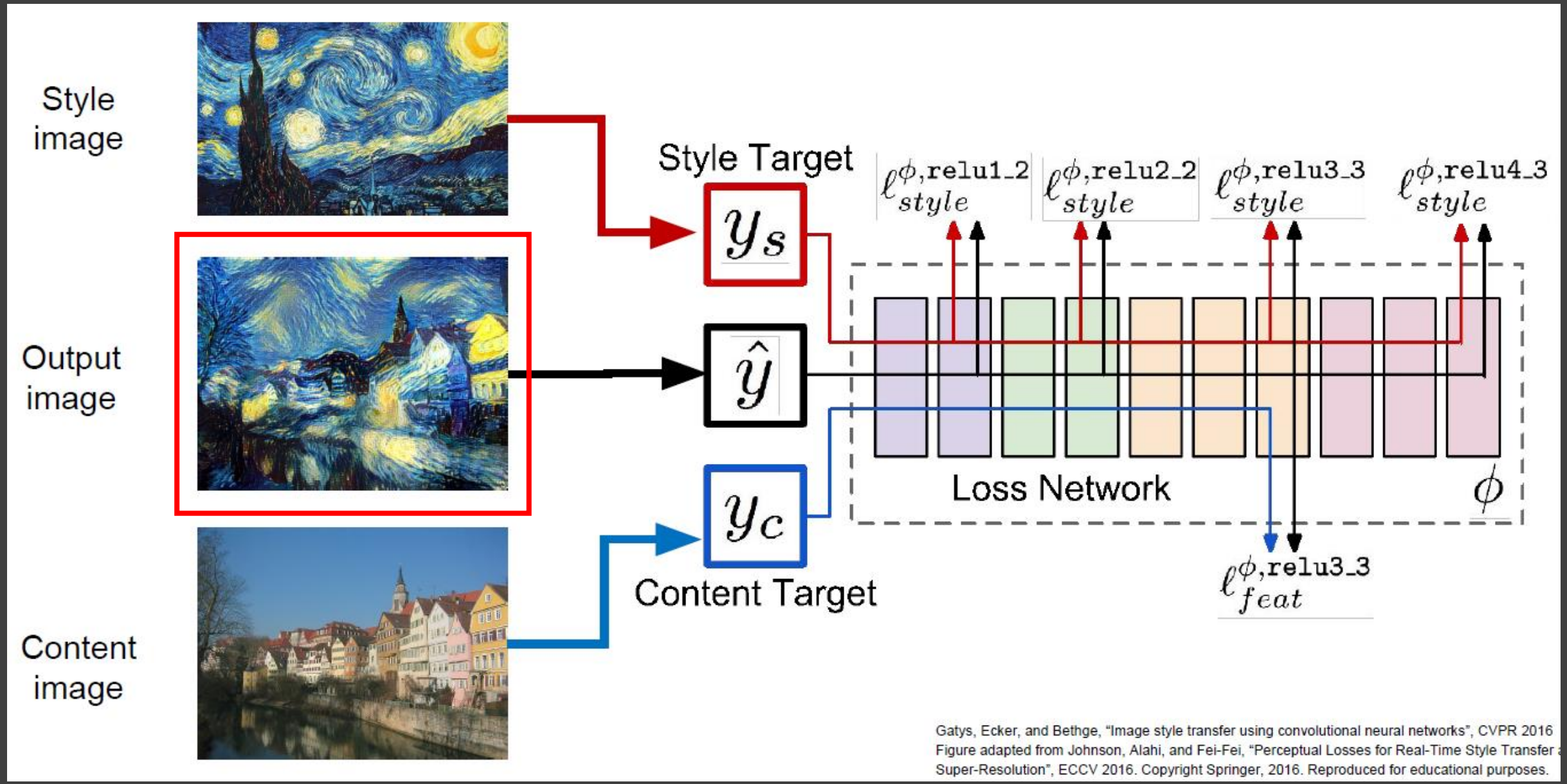
Style Transfer!

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

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



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 a
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
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

Neural Style Transfer

Content loss 에 초점을 맞춤

Style loss 에 초점을 맞춤

Neural Style Transfer

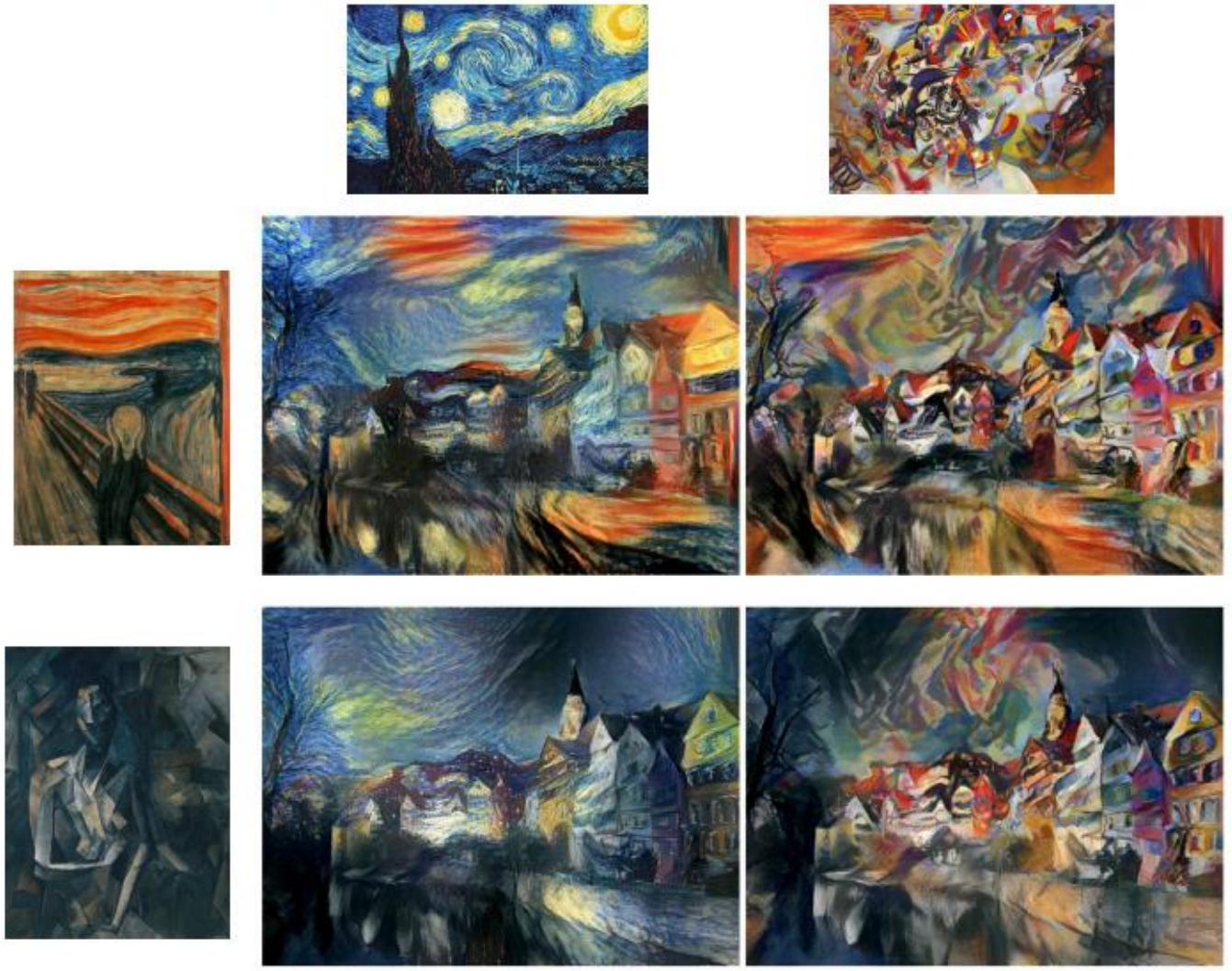
Resizing style image before running style transfer algorithm can transfer different types of features

Larger style image ⟷ Smaller style image

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

Neural Style Transfer : Multiple Style Images

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

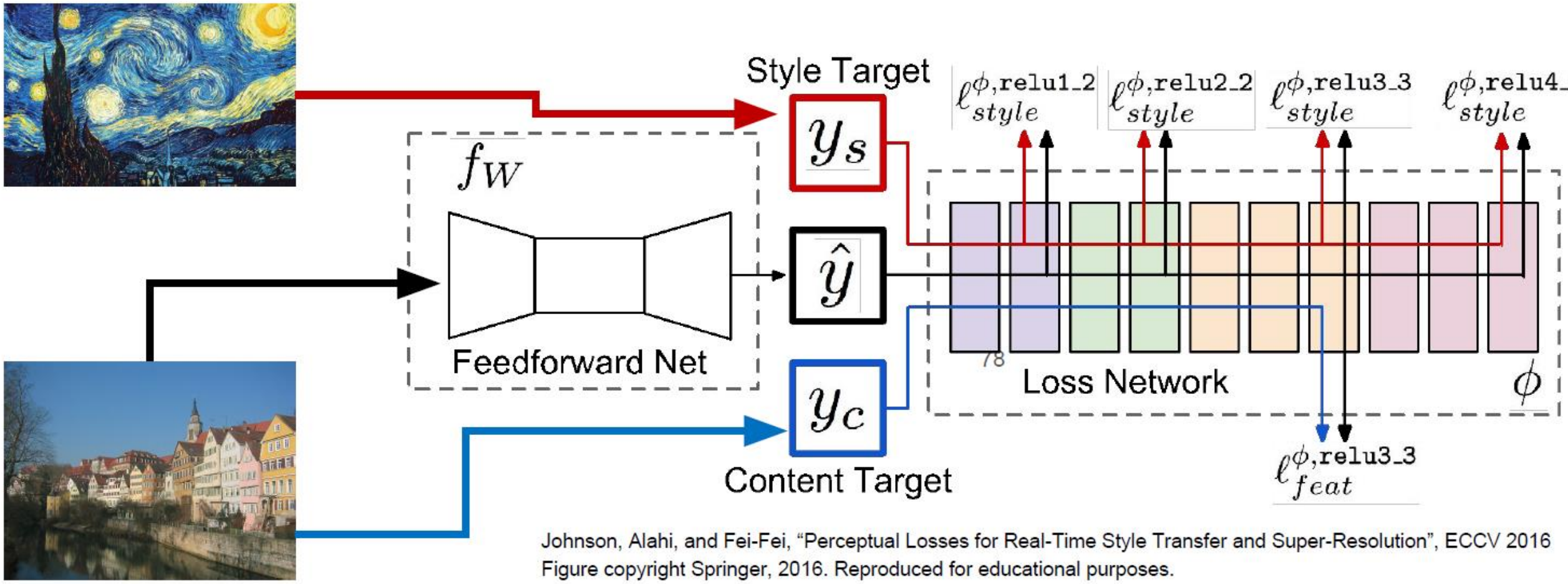


Neural Style Transfer

- Problem
  - Style transfer requires many forward / backward passes through VGG
  - Very slow!
- Solution
  - Train another neural network to perform style transfer for us!

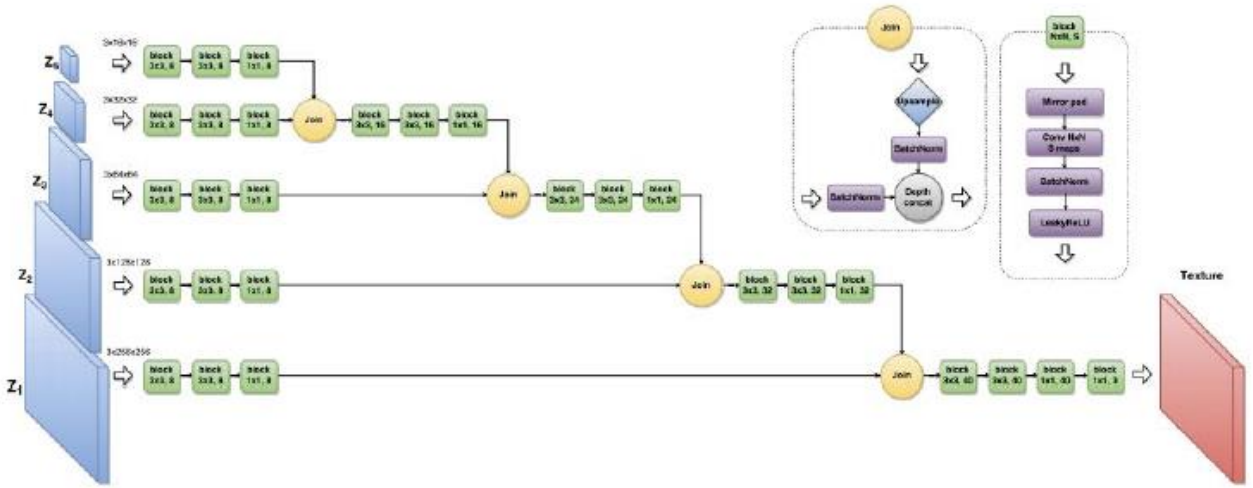
Fast Style Transfer

(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

Fast Style Transfer

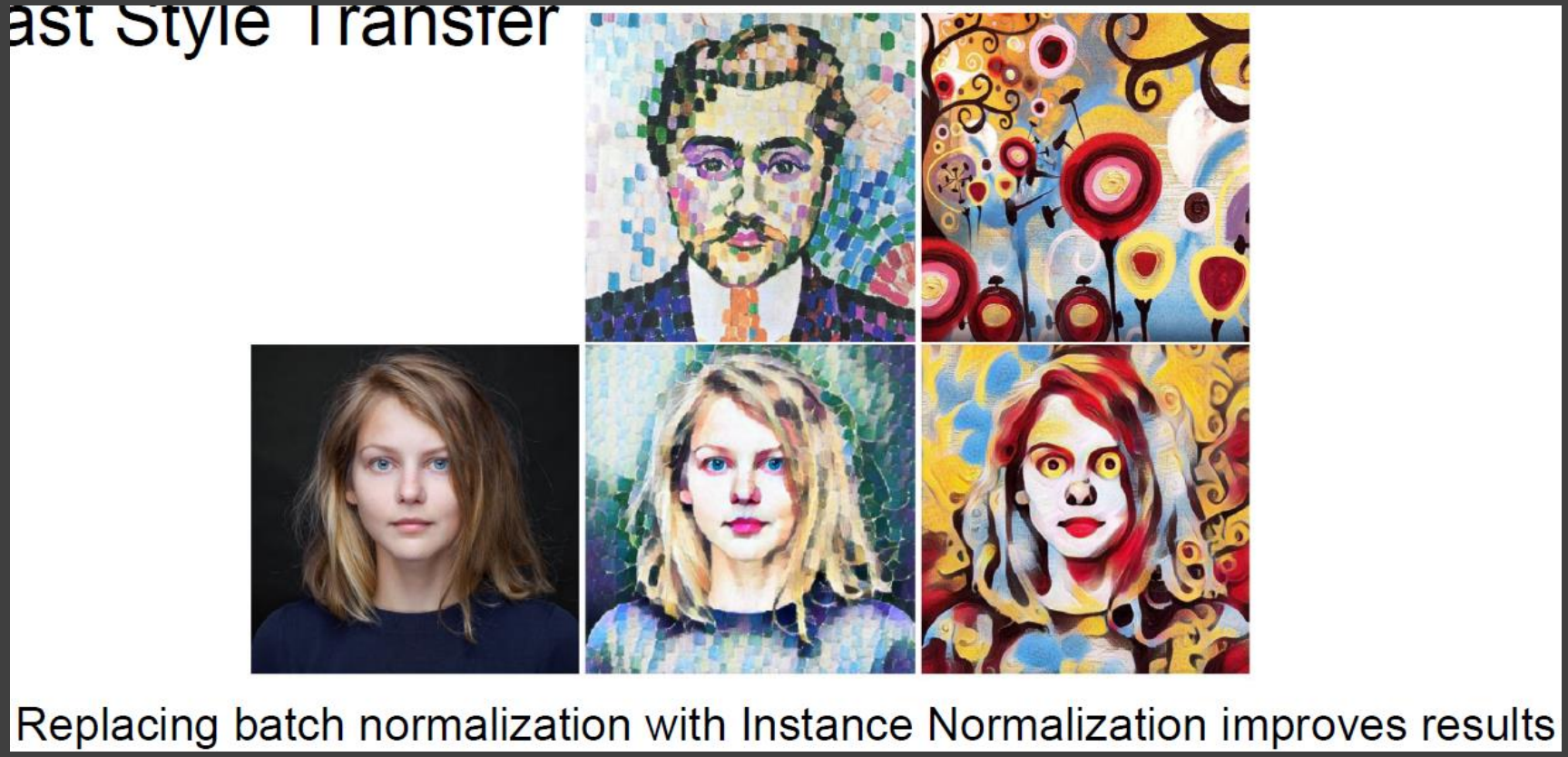


Content | Texture nets (ours) | Gatys et al. | Style

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

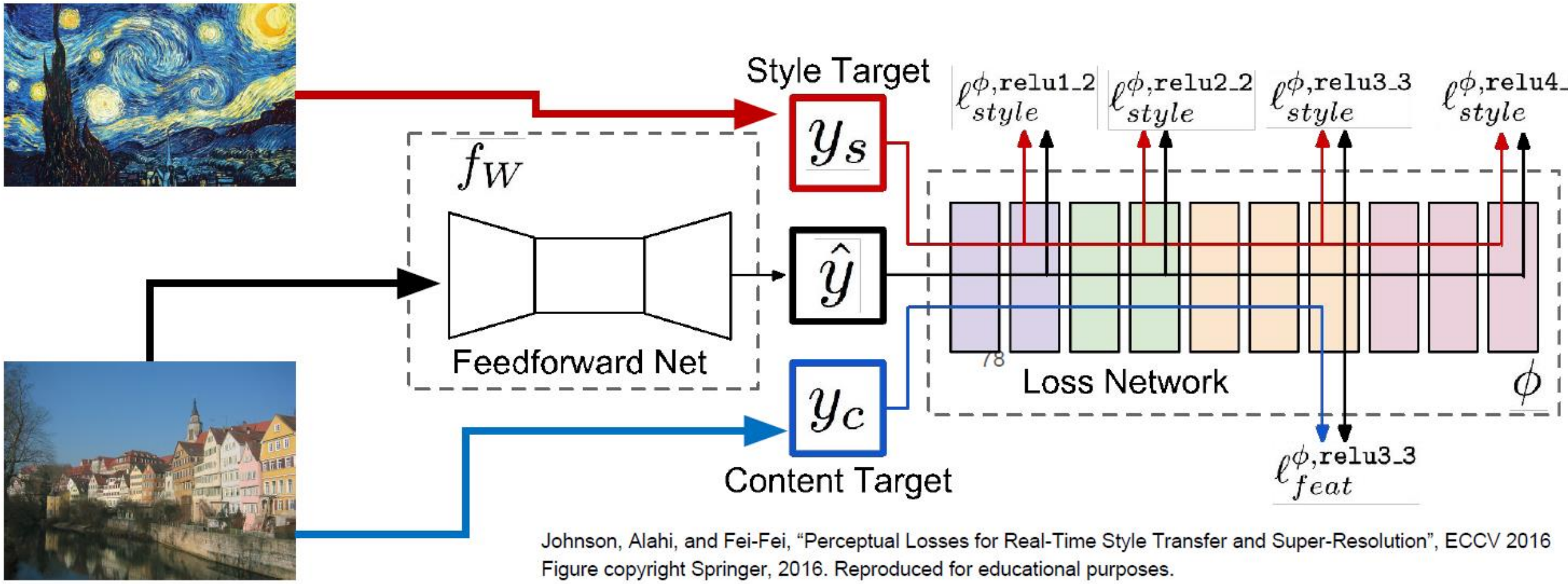
Fast Style Transfer



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

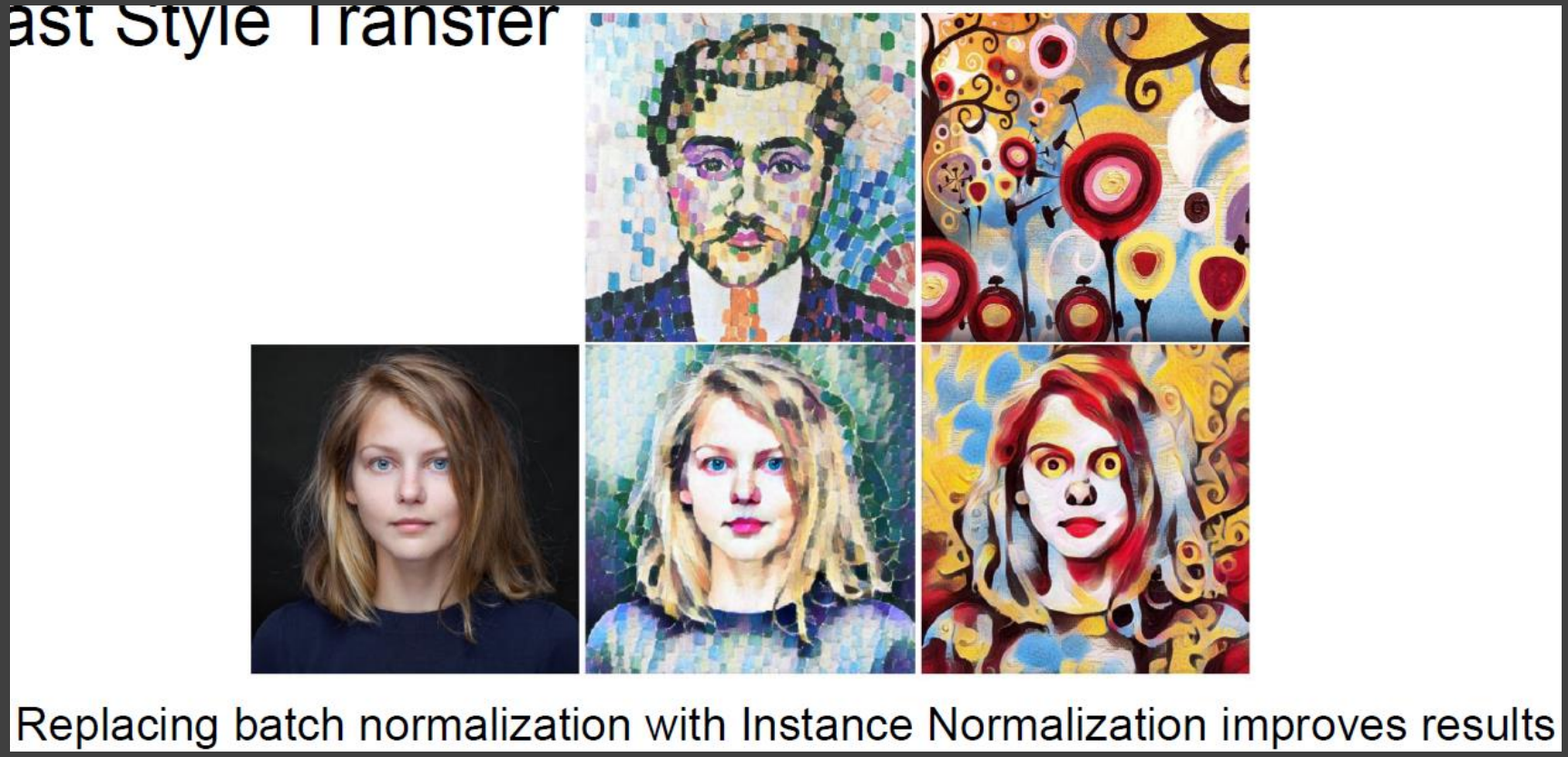
Fast Style Transfer

(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



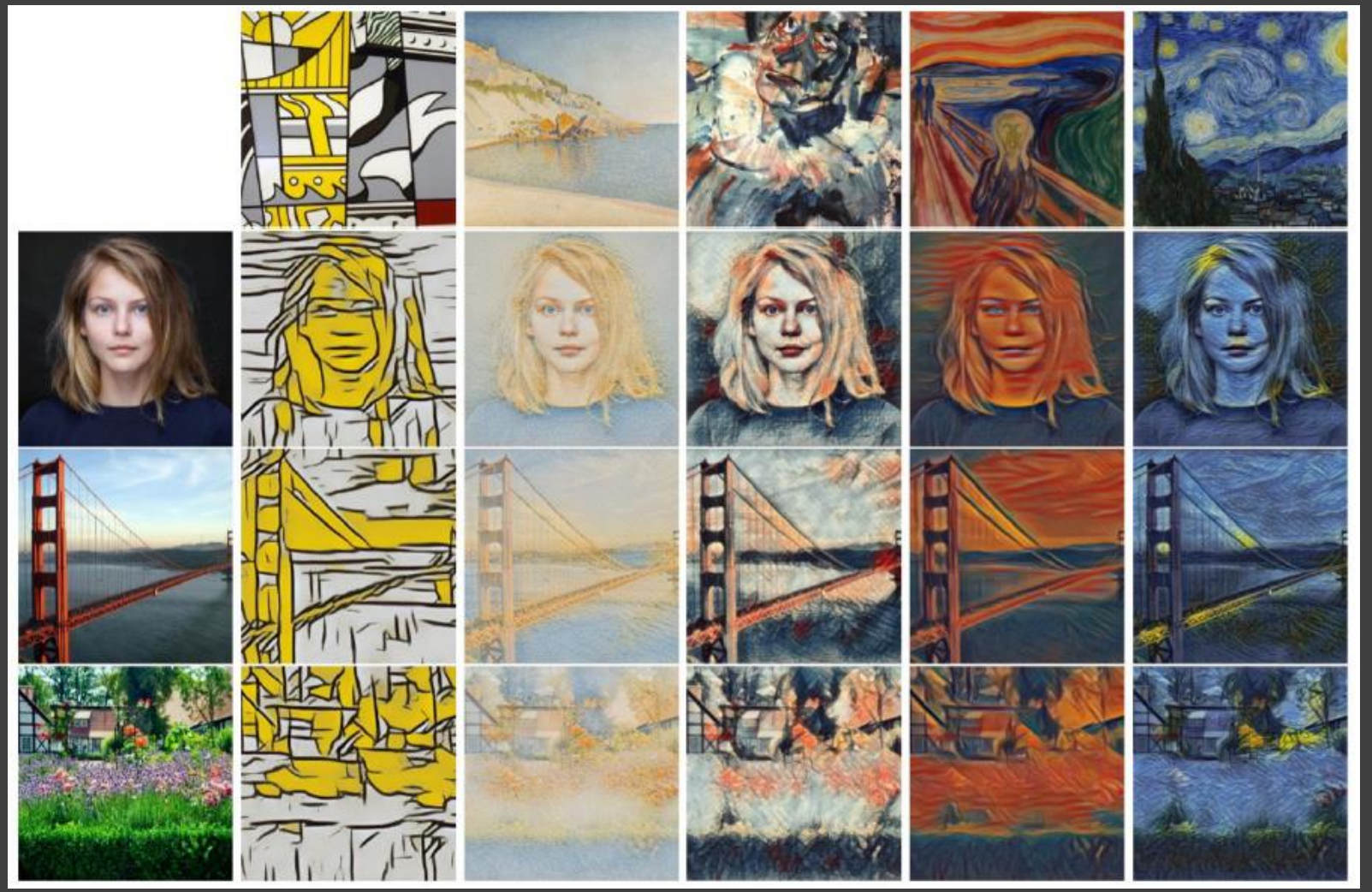
Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016

Fast Style Transfer



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

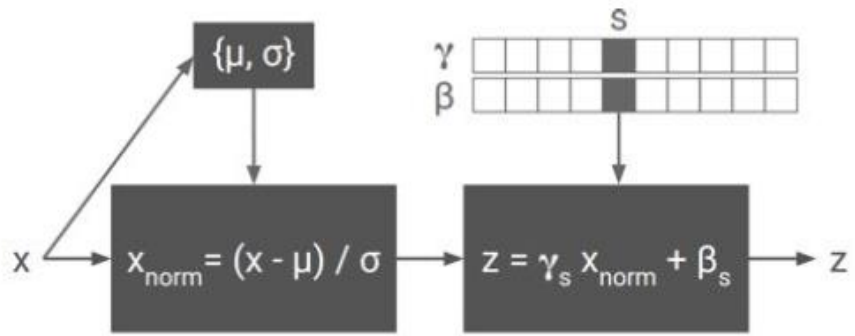
One Network, Many Styles

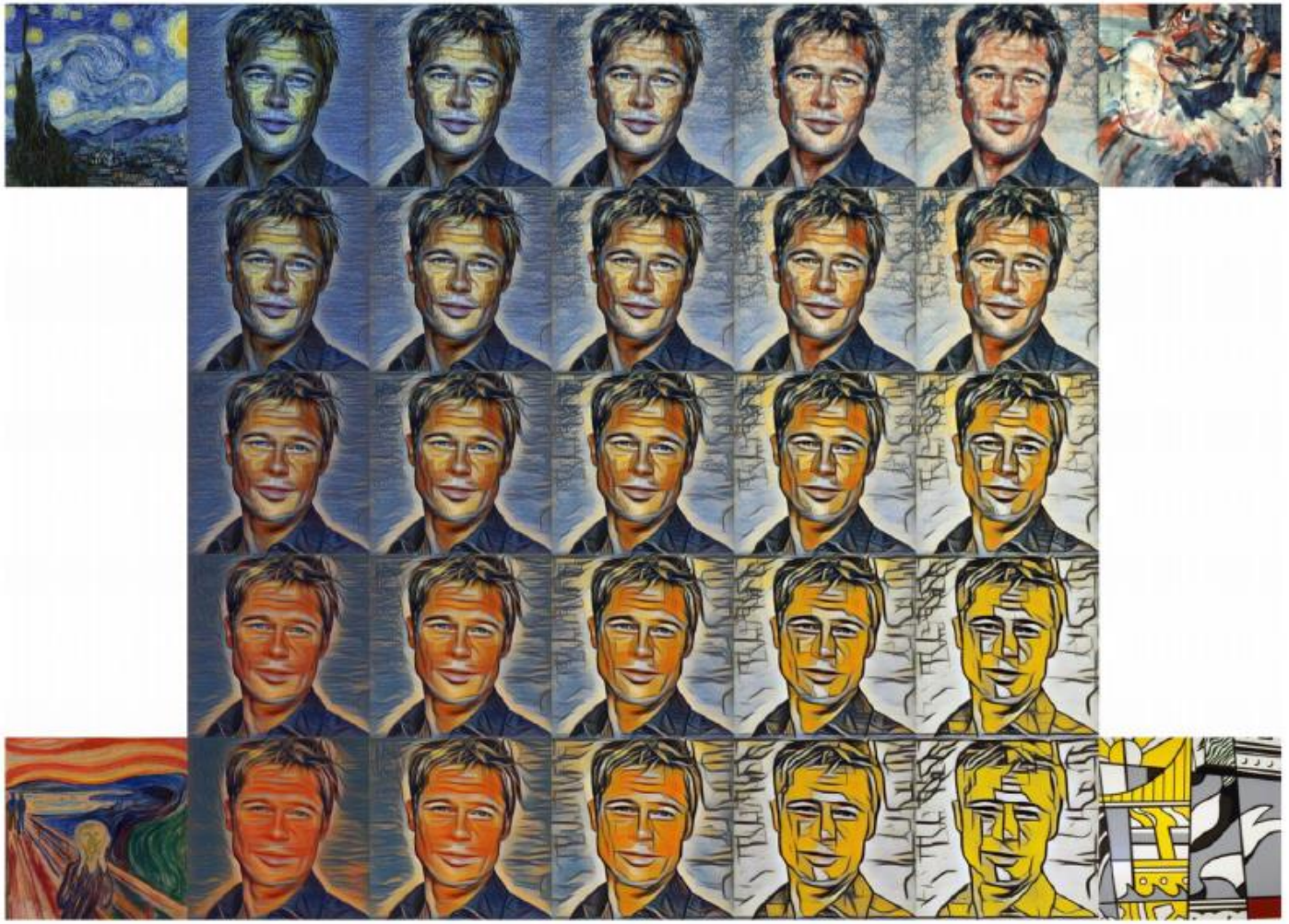


Dumoulin, Shlens, and Kudlur, "A Learned Representation for Artistic Style", ICLR 2017.

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 representation
  - Activations
    - Nearest neighbors, Dimensionality reduction, maximal patches, occlusion
  - Gradients
    - Saliency maps, class visualization, fooling images, feature inversion
  - Fun
    - DeepDream, Style transfer