DATA ANALYTICS USING DEEP LEARNING

GT 8803 // FALL 2019 // JOY ARULRAJ

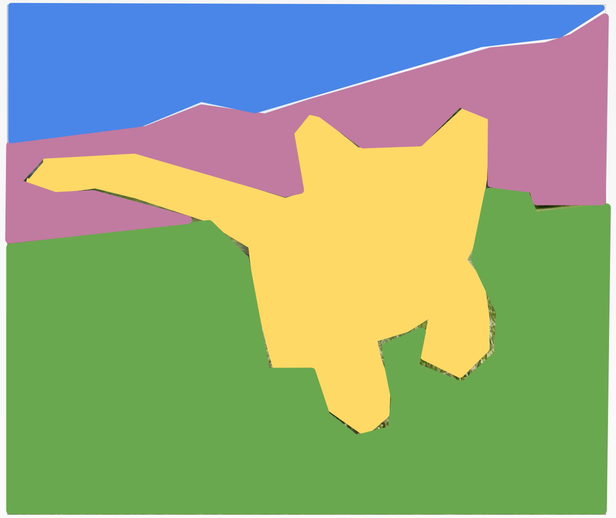
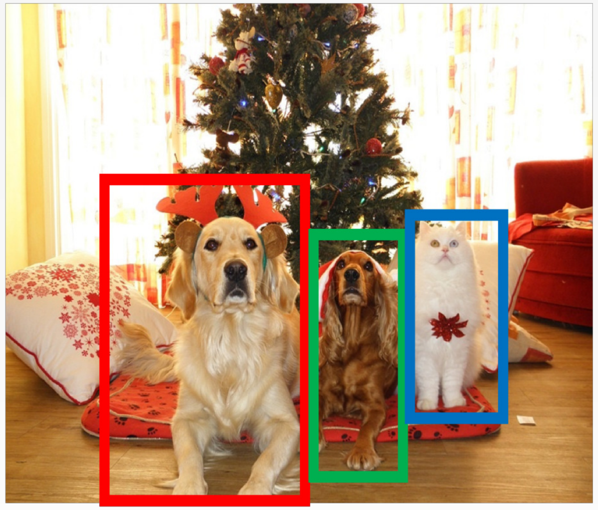
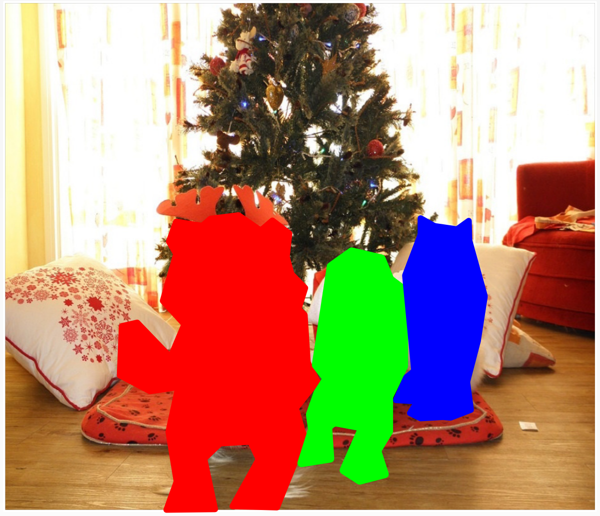
LECTURE #18: VISUALIZING & UNDERSTANDING CONVOLUTIONAL NETWORKS

CREATING THE NEXT®

ADMINISTRIVIA

- Reminders
  - Code reviews due on Nov 9
  - Team member contribution analyses will be anonymous
  - Grades for project checkpoint #1 released
  - Assignment 3 released

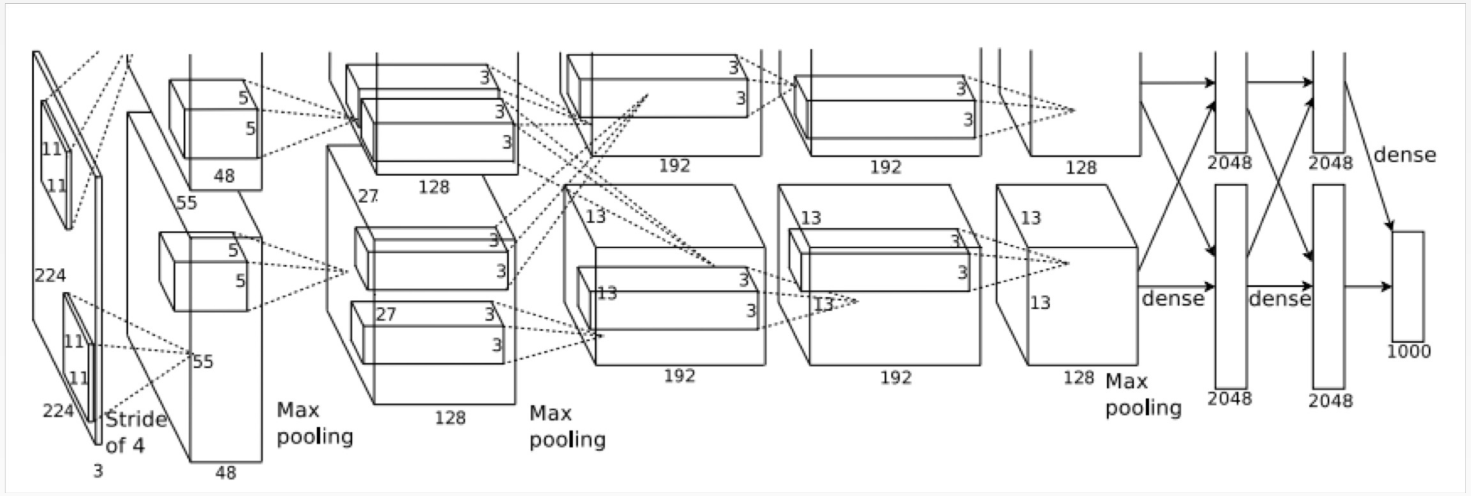
LAST TIME: LOTS OF COMPUTER VISION TASKS

Classification	Semantic Segmentation	Object Detection	Instance Segmentation
			
CAT	GRASS, CAT, TREE, SKY	DOG, DOG, CAT	DOG, DOG, CAT
No spatial extent	No objects, just pixels	Multiple Object	

Georgia Tech

WHAT'S GOING ON INSIDE CONVNETS?

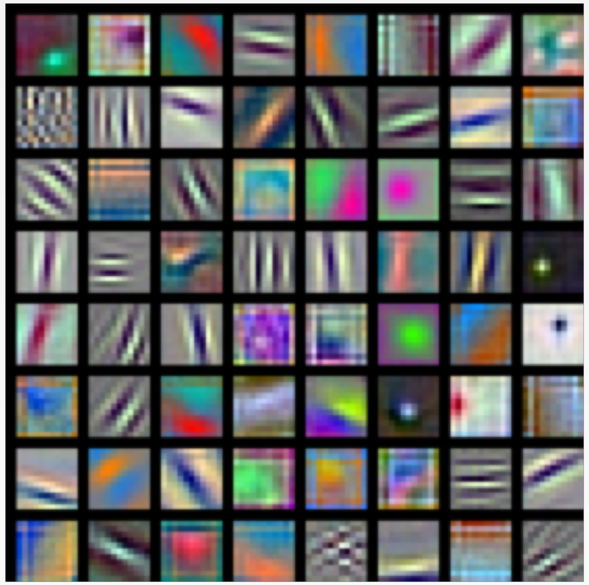
Input Image:
3 x 224 x 224



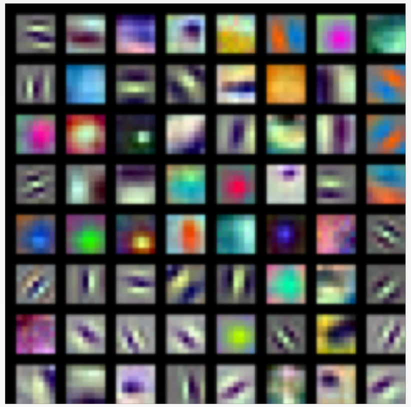
Class Scores:
1000 numbers

What are the intermediate features looking for?

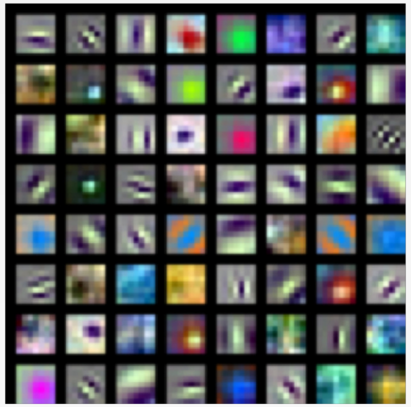
FIRST LAYER: VISUALIZE FILTERS



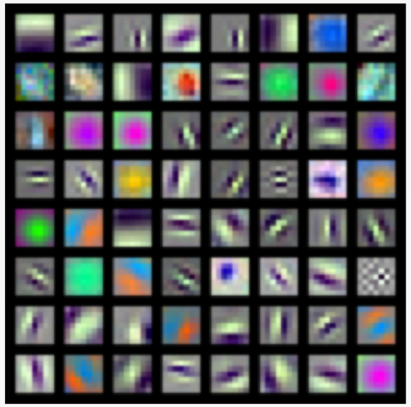
AlexNet:
64 x 3 x 11 x 11



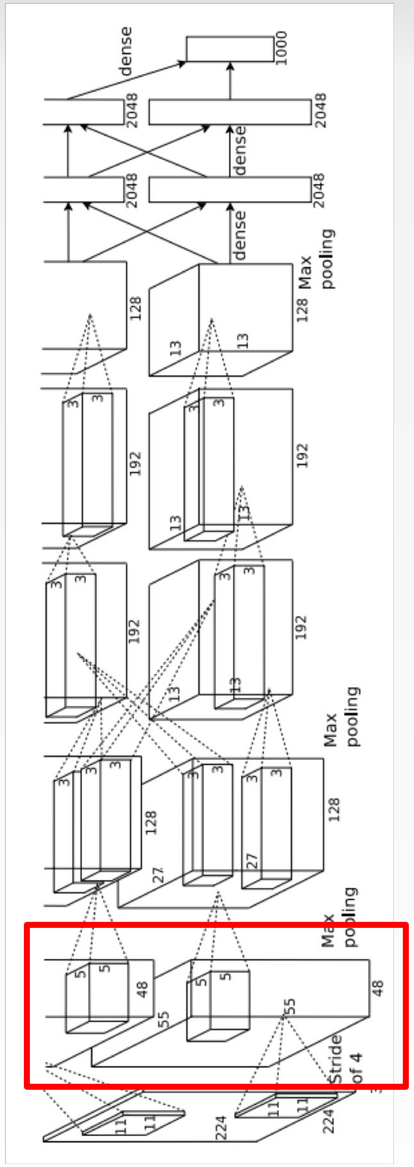
ResNet-18:
64 x 3 x 7 x 7



ResNet-101:
64 x 3 x 7 x 7

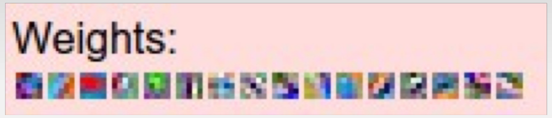


DenseNet-121:
64 x 3 x 7 x 7



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

VISUALIZE THE FILTERS/KERNELS (RAW WEIGHTS)



layer 1 weights

16 x 3 x 7 x 7

layer 2 weights

20 x 16 x 7 x 7

layer 3 weights

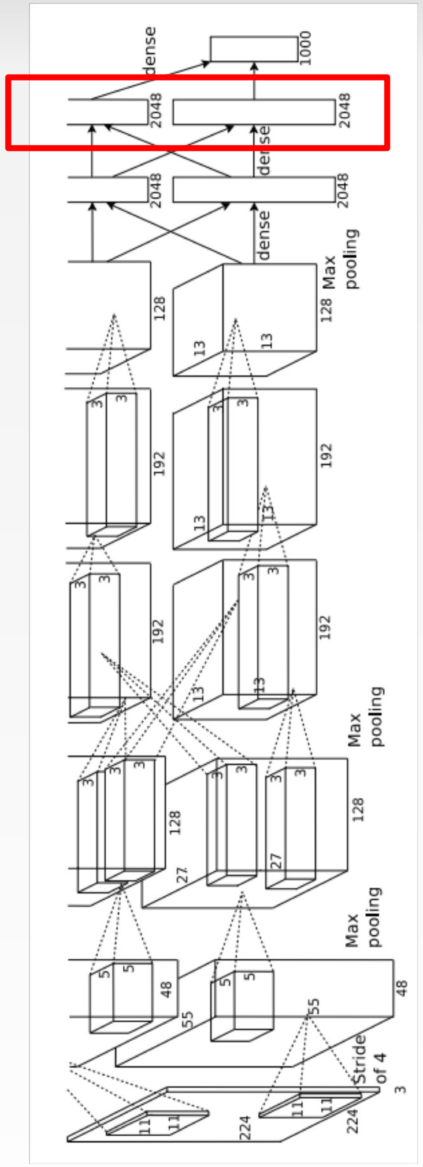
20 x 20 x 7 x 7

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

(these are taken from ConvNetJS CIFAR-10 demo)

LAST LAYER

FC7 layer



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

Run the network on many images, collect the feature vectors

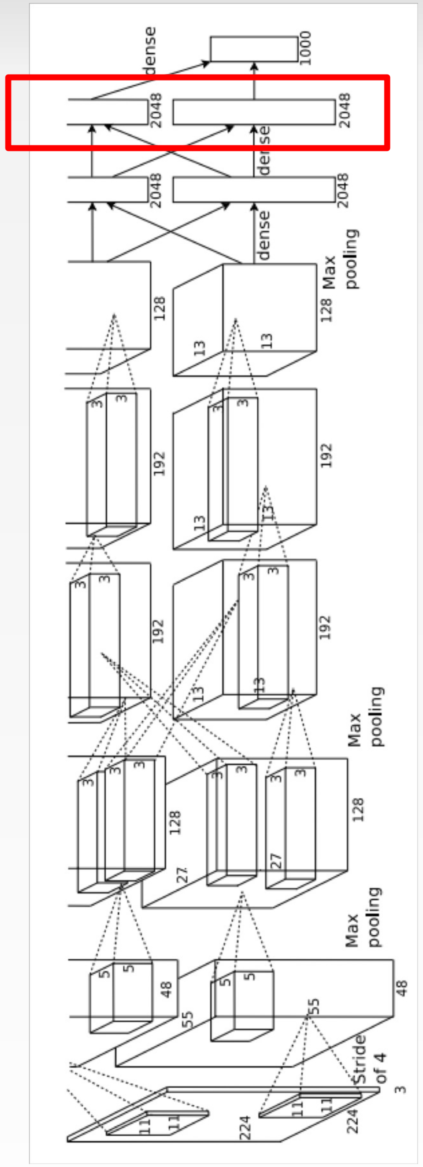
LAST LAYER: NEAREST NEIGHBORS

Recall: Nearest neighbors in pixel space

Test image

L2 Nearest neighbors in feature space

4096-dim vector



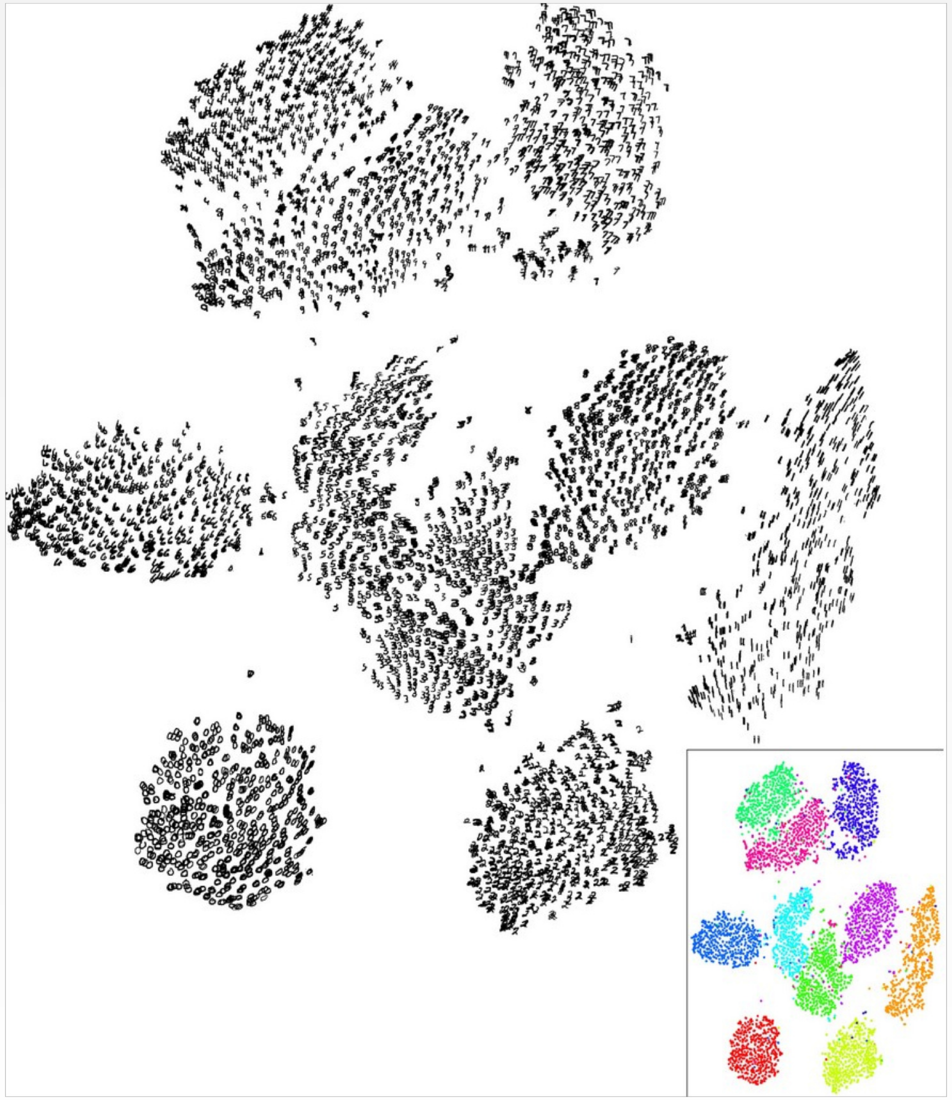
Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012.
Figures reproduced with permission.

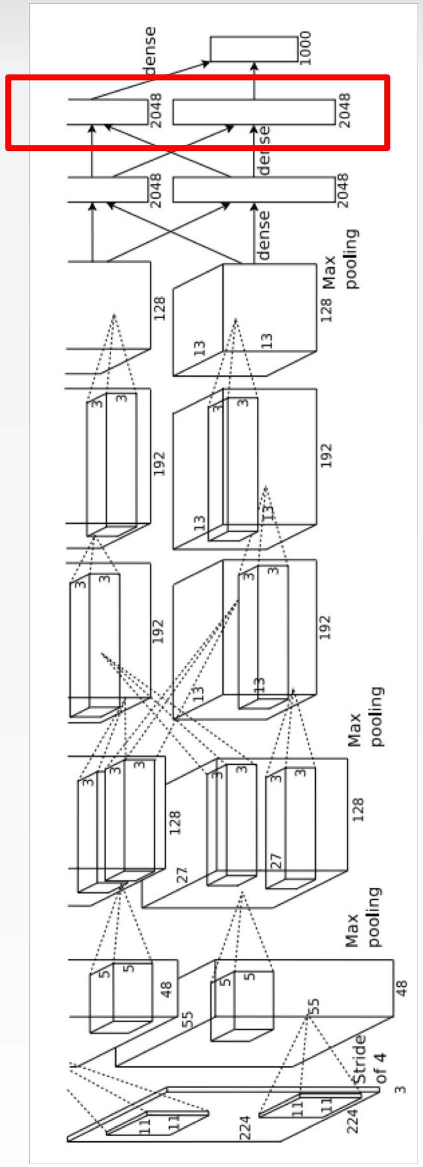
LAST LAYER: DIMENSIONALITY REDUCTION

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

Simple algorithm:
Principal Component Analysis (PCA)

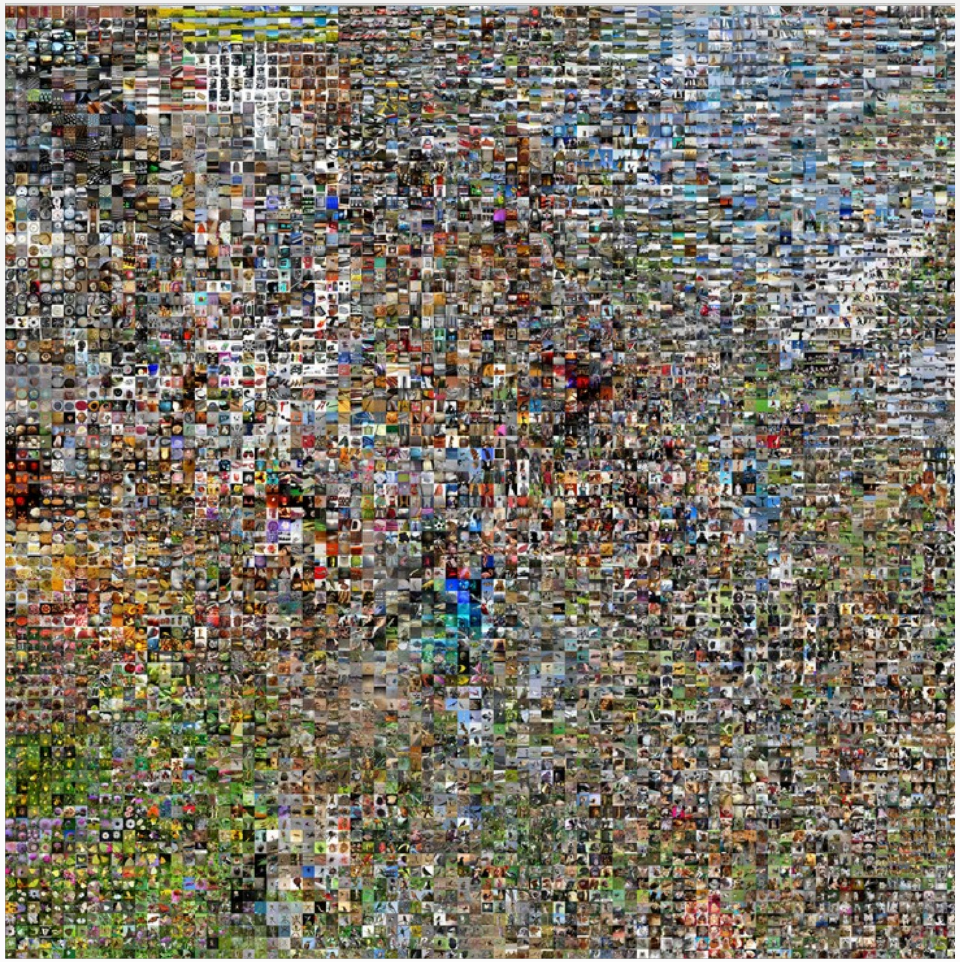
More complex: **t-SNE**

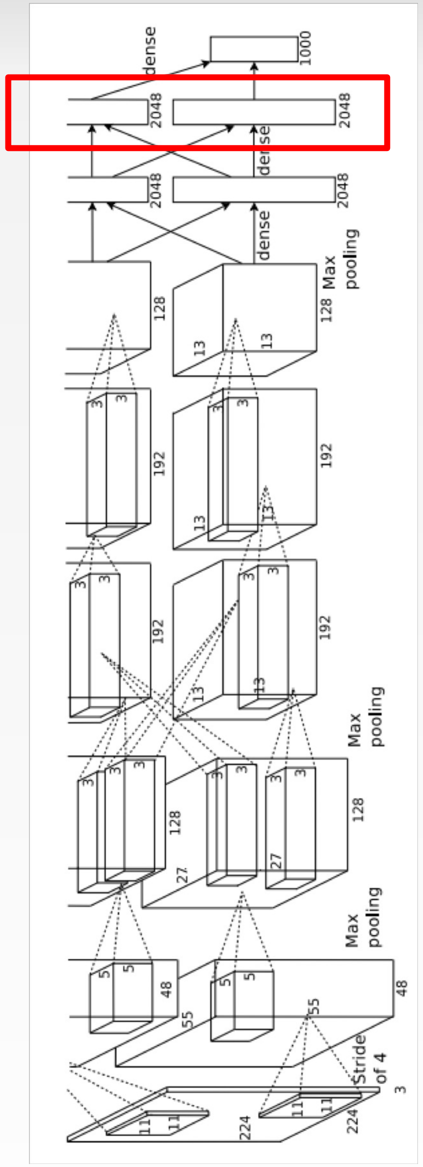




Van der Maaten and Hinton, "Visualizing Data using t-SNE", JMLR 2008

LAST LAYER: DIMENSIONALITY REDUCTION



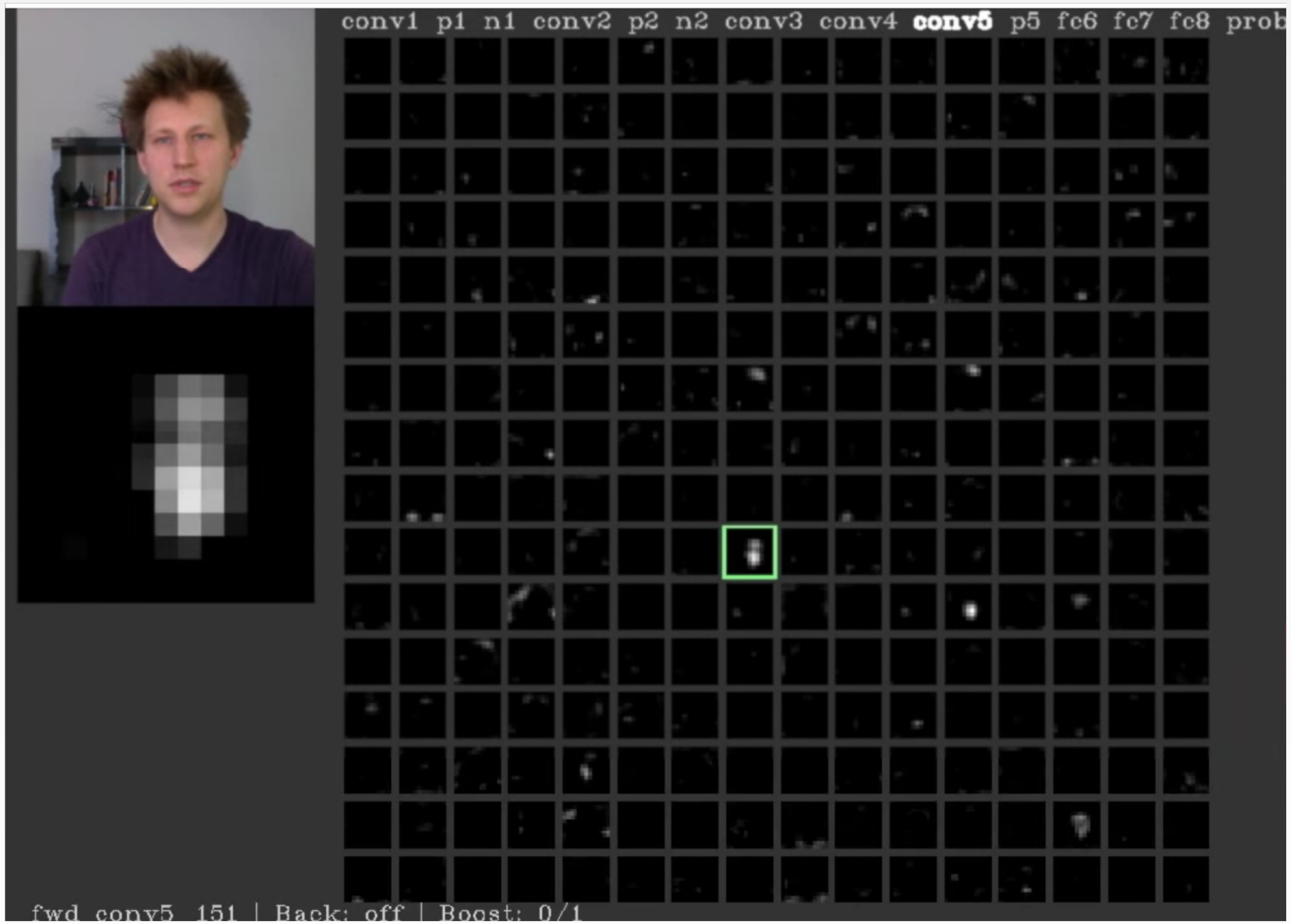


Van der Maaten and Hinton, "Visualizing Data using t-SNE", JMLR 2008
Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012.
Figure reproduced with permission.

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

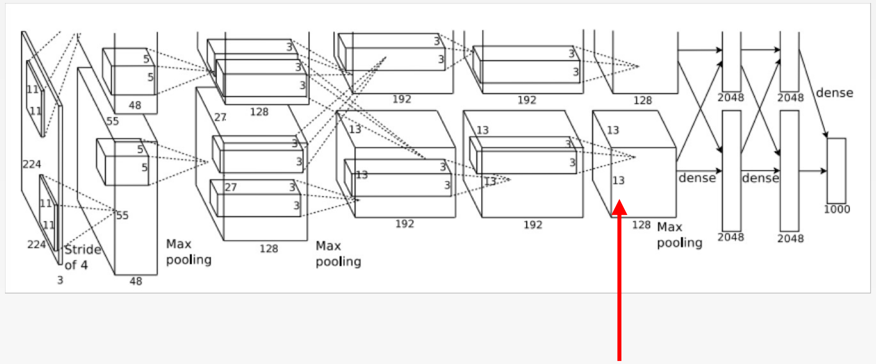
VISUALIZING ACTIVATIONS

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



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

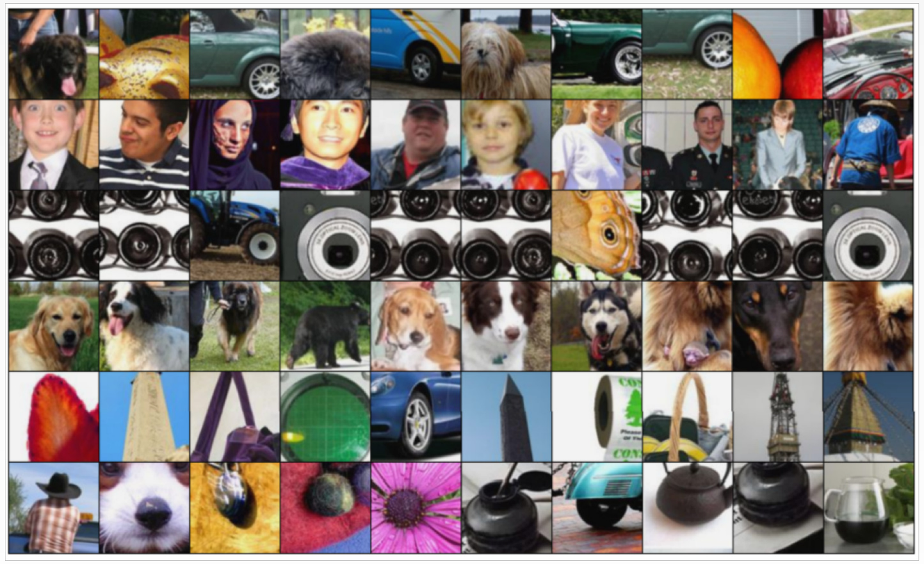
MAXIMALLY ACTIVATING PATCHES



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

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

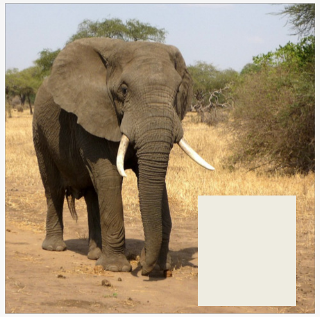
Visualize image patches that correspond to maximal activations

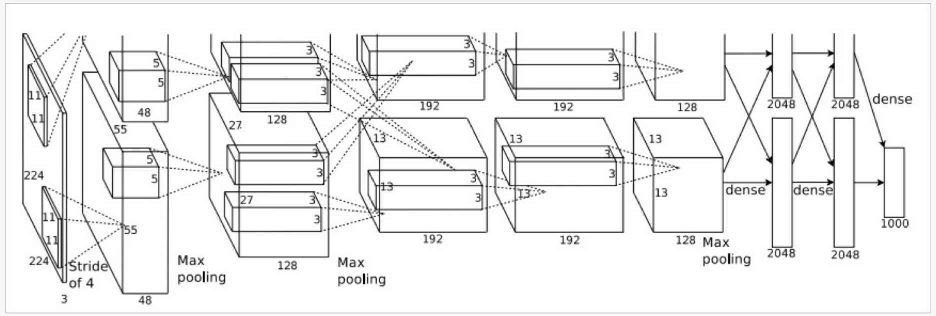


Springenberg et al, “Striving for Simplicity: The All Convolutional Net”, ICLR Workshop 2015

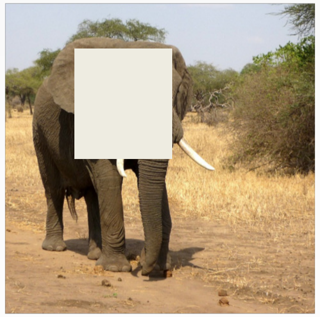
WHICH PIXELS MATTER: SALIENCY VIA OCCLUSION

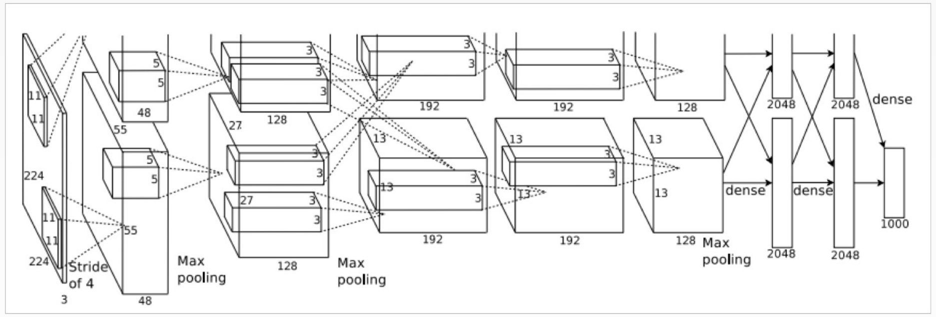
Mask part of the image before feeding to CNN, check how much predicted probabilities change





P(elephant) = 0.95



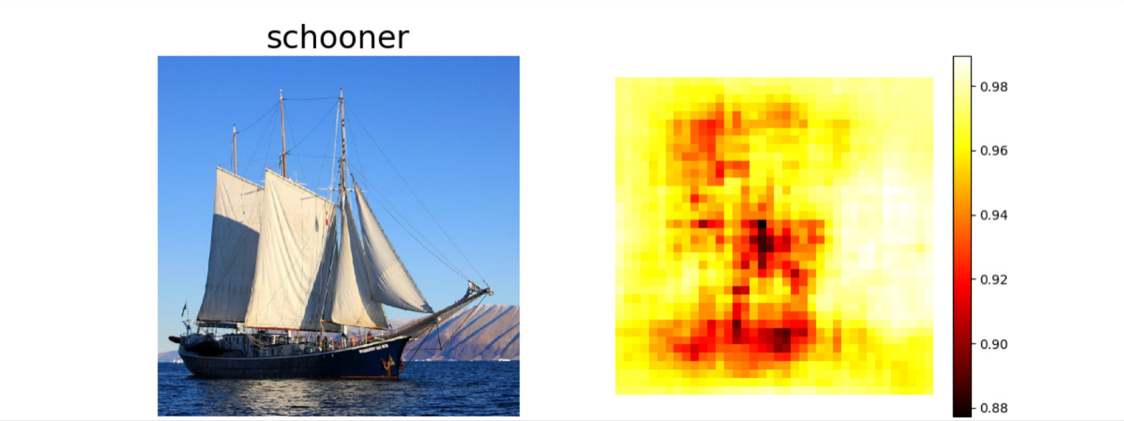


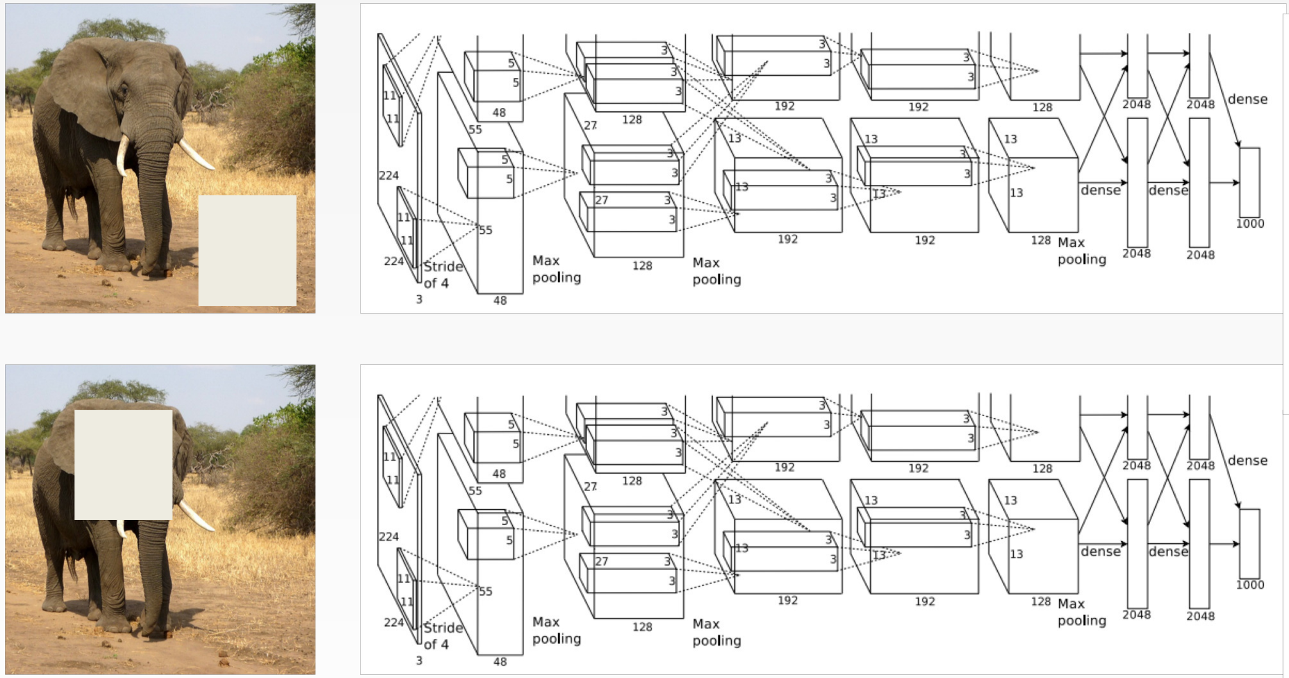
P(elephant) = 0.75

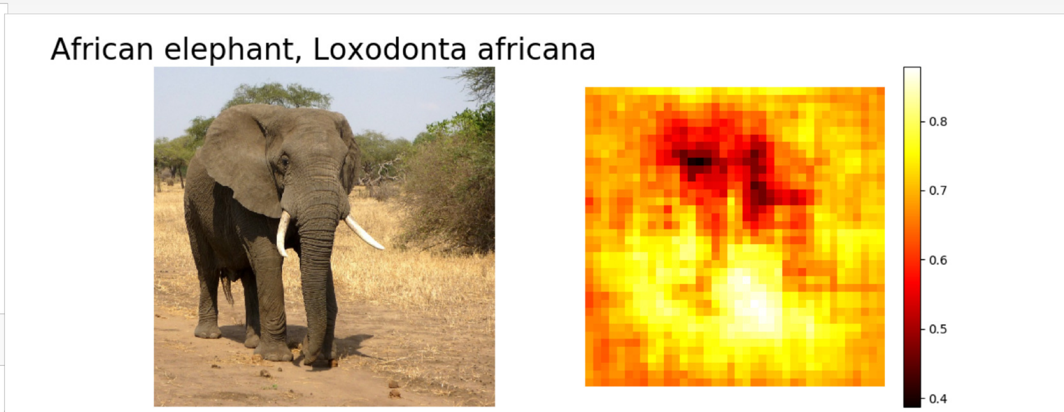
Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

WHICH PIXELS MATTER: SALIENCY VIA OCCLUSION

Mask part of the image before feeding to CNN, check how much predicted probabilities change





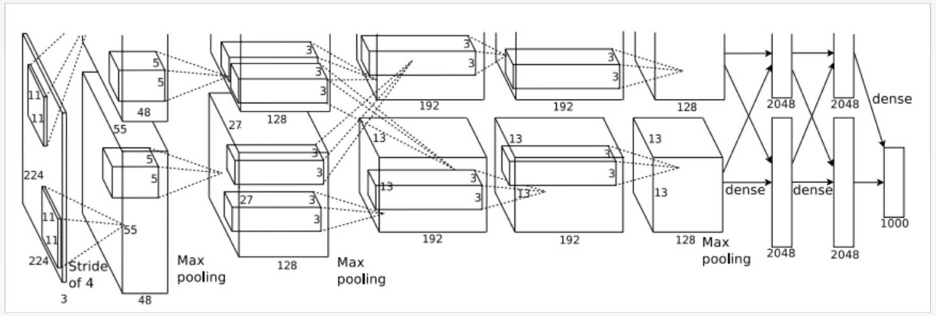


Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

Georgia Tech

WHICH PIXELS MATTER: SALIENCY VIA BACKPROP

Forward pass: Compute probabilities

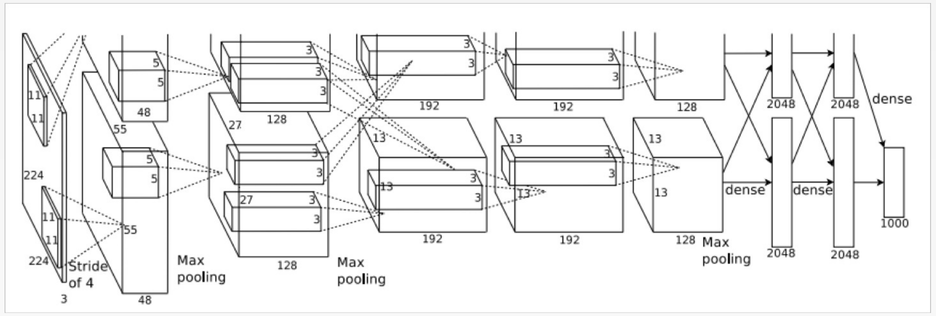


Dog

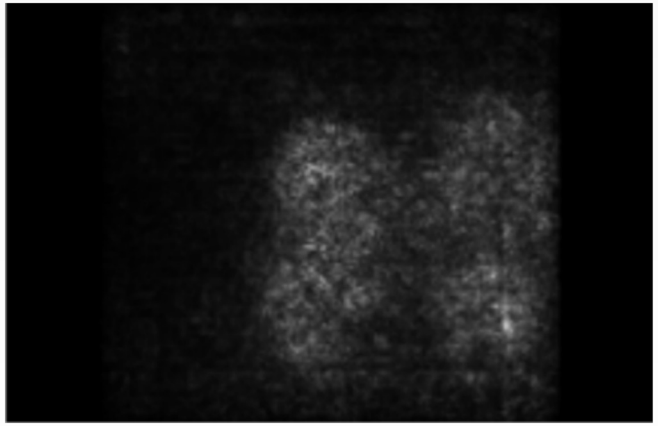
Simonyan, Vedaldi, and Zisserman, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”, ICLR Workshop 2014.

WHICH PIXELS MATTER: SALIENCY VIA BACKPROP

Forward pass: Compute probabilities



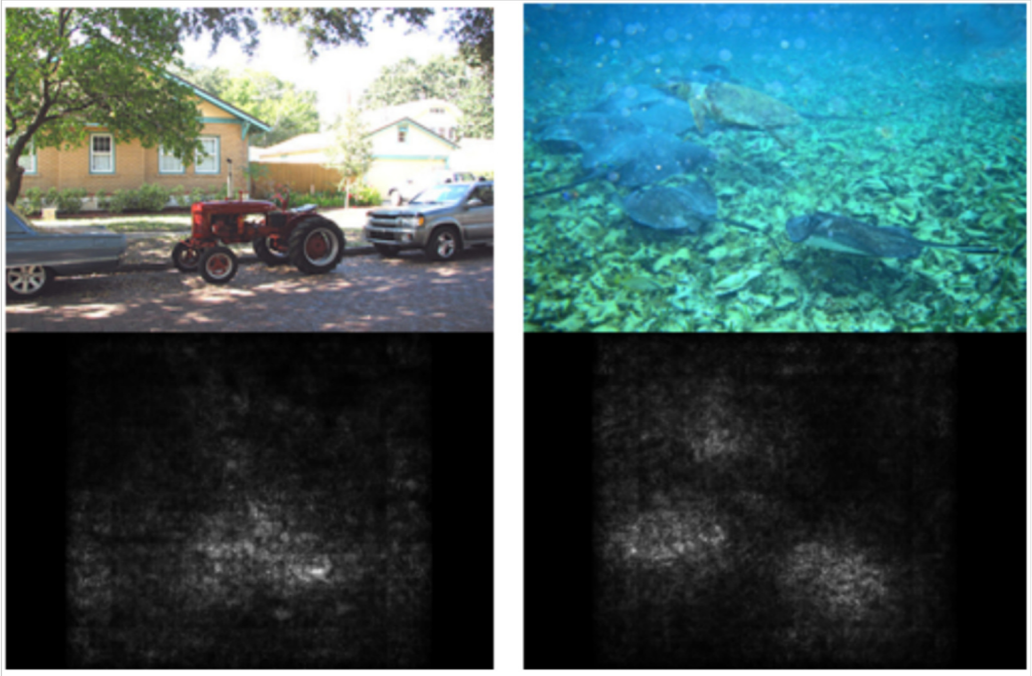
Dog



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

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

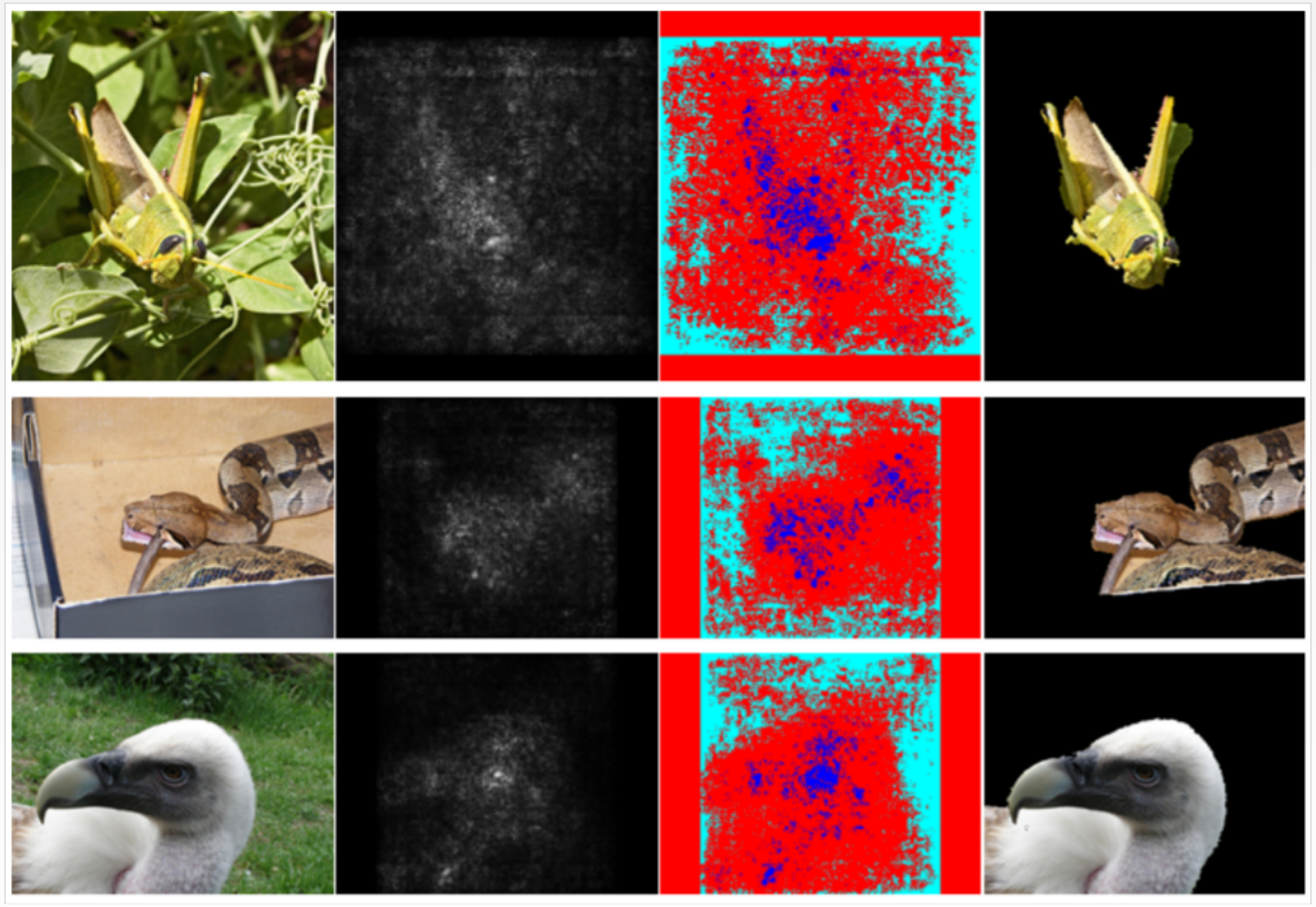
SALIENCY MAPS



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

SALIENCY MAPS: SEGMENTATION WITHOUT SUPERVISION

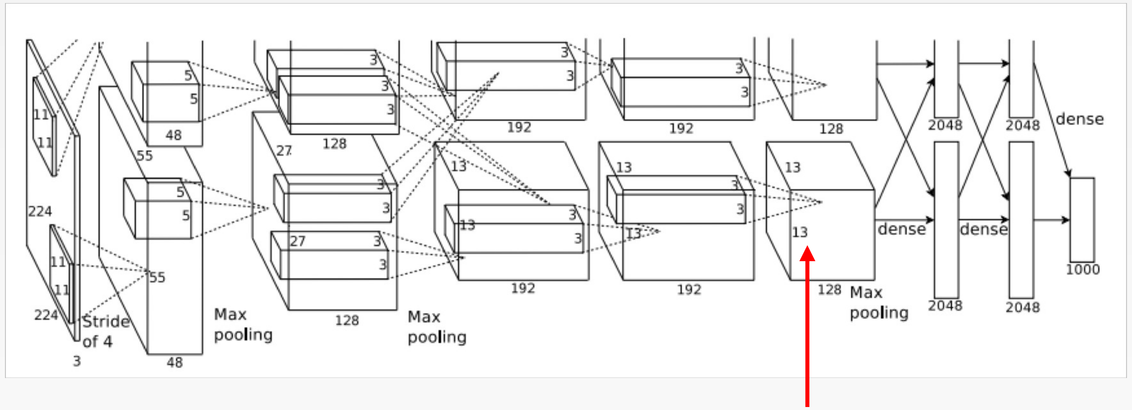
Use GrabCut on saliency map



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

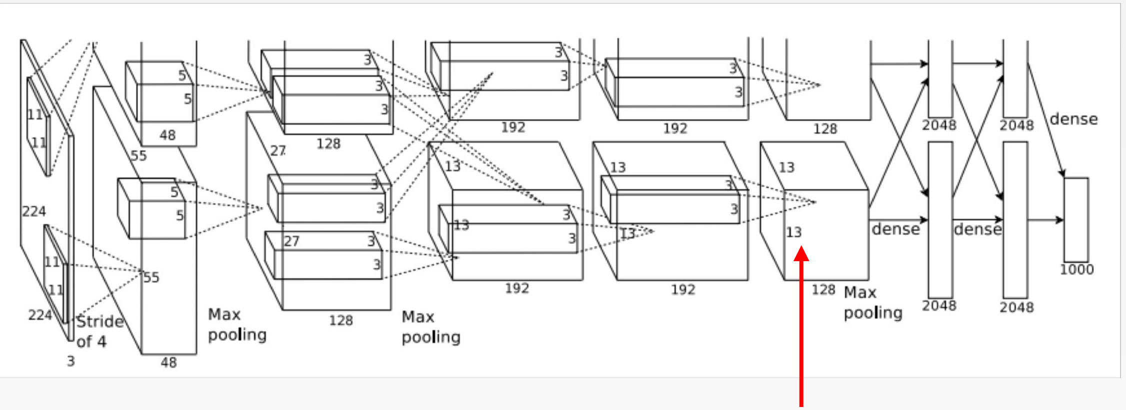


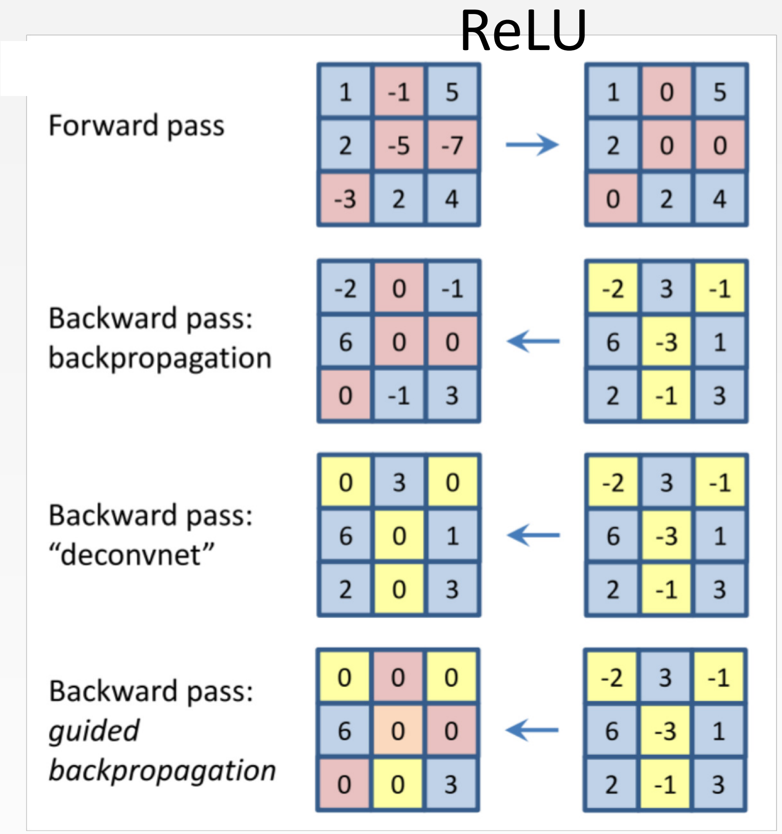
Pick a single intermediate neuron, e.g. one value in 128 x 13 x 13 conv5 feature map

Compute gradient of neuron value with respect to image pixels

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

INTERMEDIATE FEATURES VIA (GUIDED) BACKPROP





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

Compute gradient of neuron value with respect to image pixels

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

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

Maximally activating patches
(Each row is a different neuron)

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

VISUALIZING CNN FEATURES: GRADIENT ASCENT

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

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

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

Neuron value

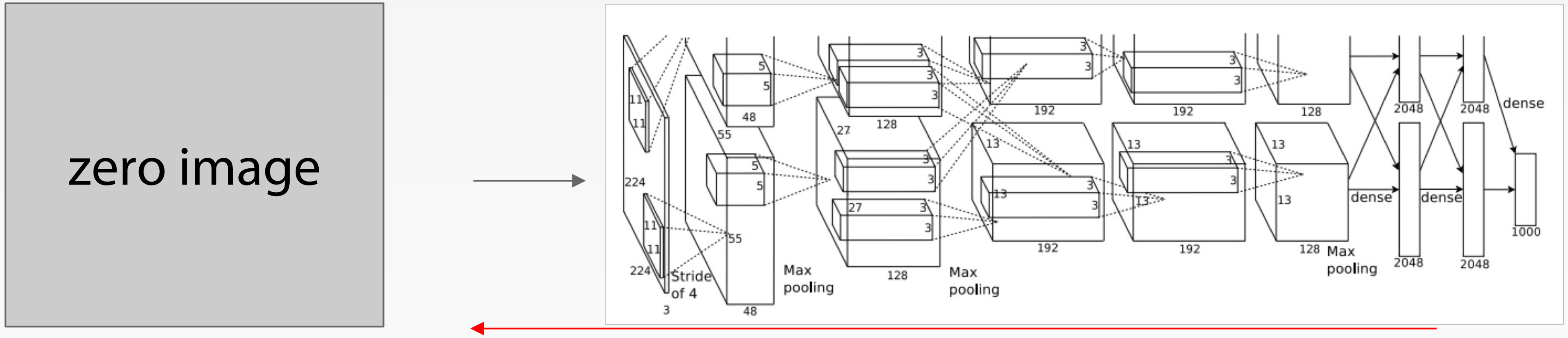
Natural image regularizer

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



Repeat:

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

VISUALIZING CNN FEATURES: GRADIENT ASCENT

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

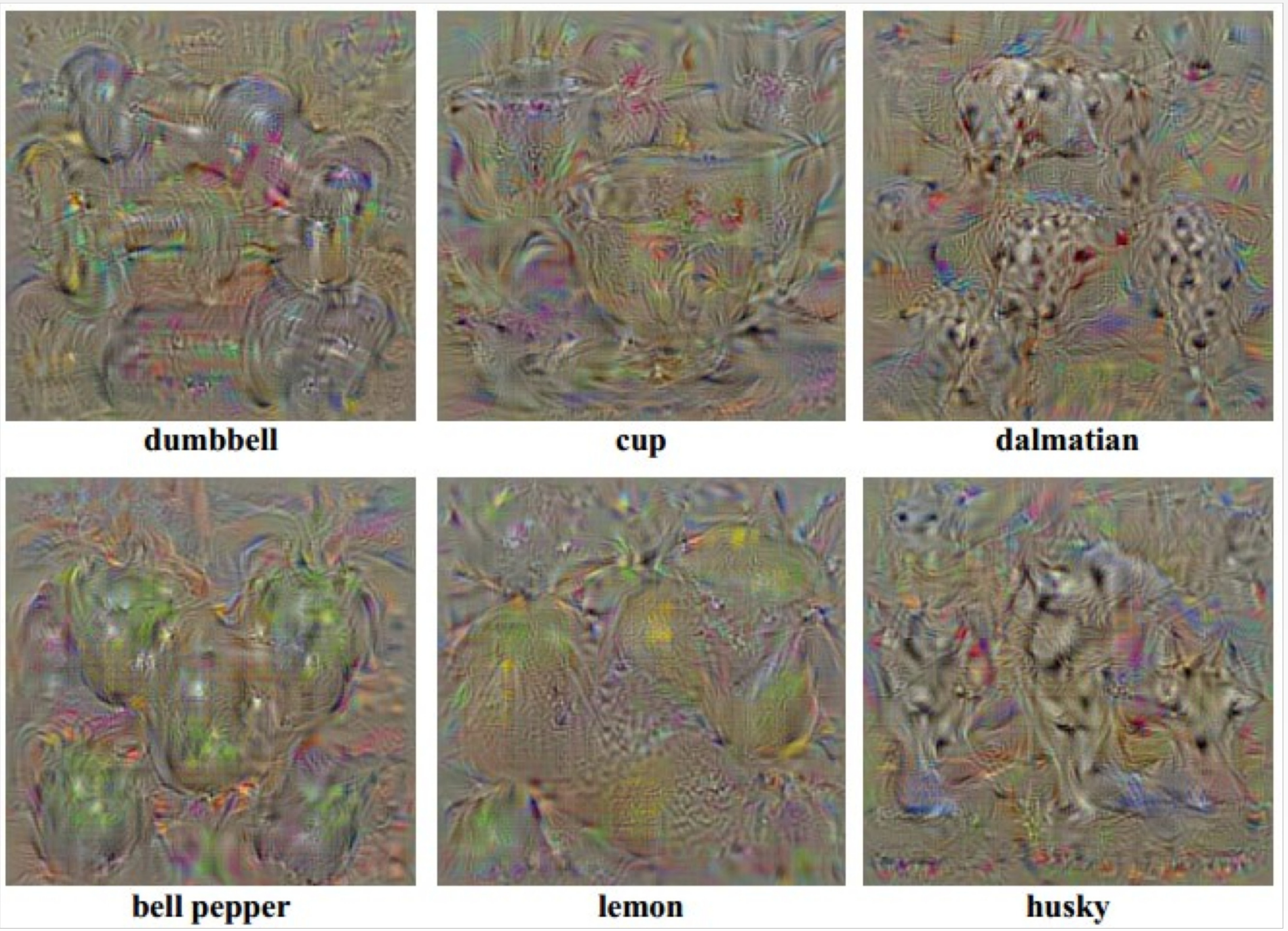
Simple regularizer: Penalize
L2 norm of generated image

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

VISUALIZING CNN FEATURES: GRADIENT ASCENT

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

Simple regularizer: Penalize L2 norm of generated image

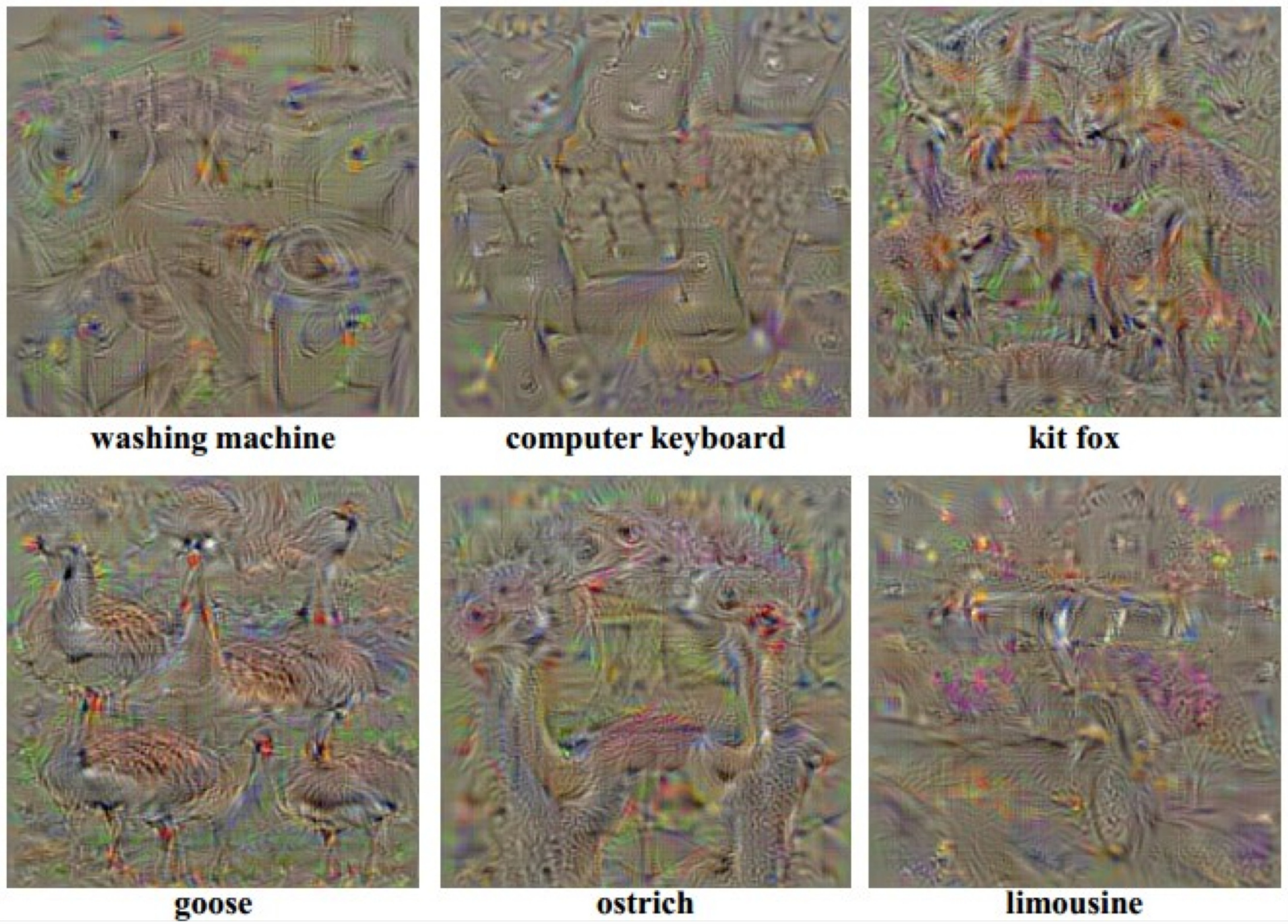


Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.

VISUALIZING CNN FEATURES: GRADIENT ASCENT

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

Simple regularizer: Penalize L2 norm of generated image



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

VISUALIZING CNN FEATURES: GRADIENT ASCENT

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

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

- Gaussian blur image
- Clip pixels with small values to 0
- Clip pixels with small gradients to 0

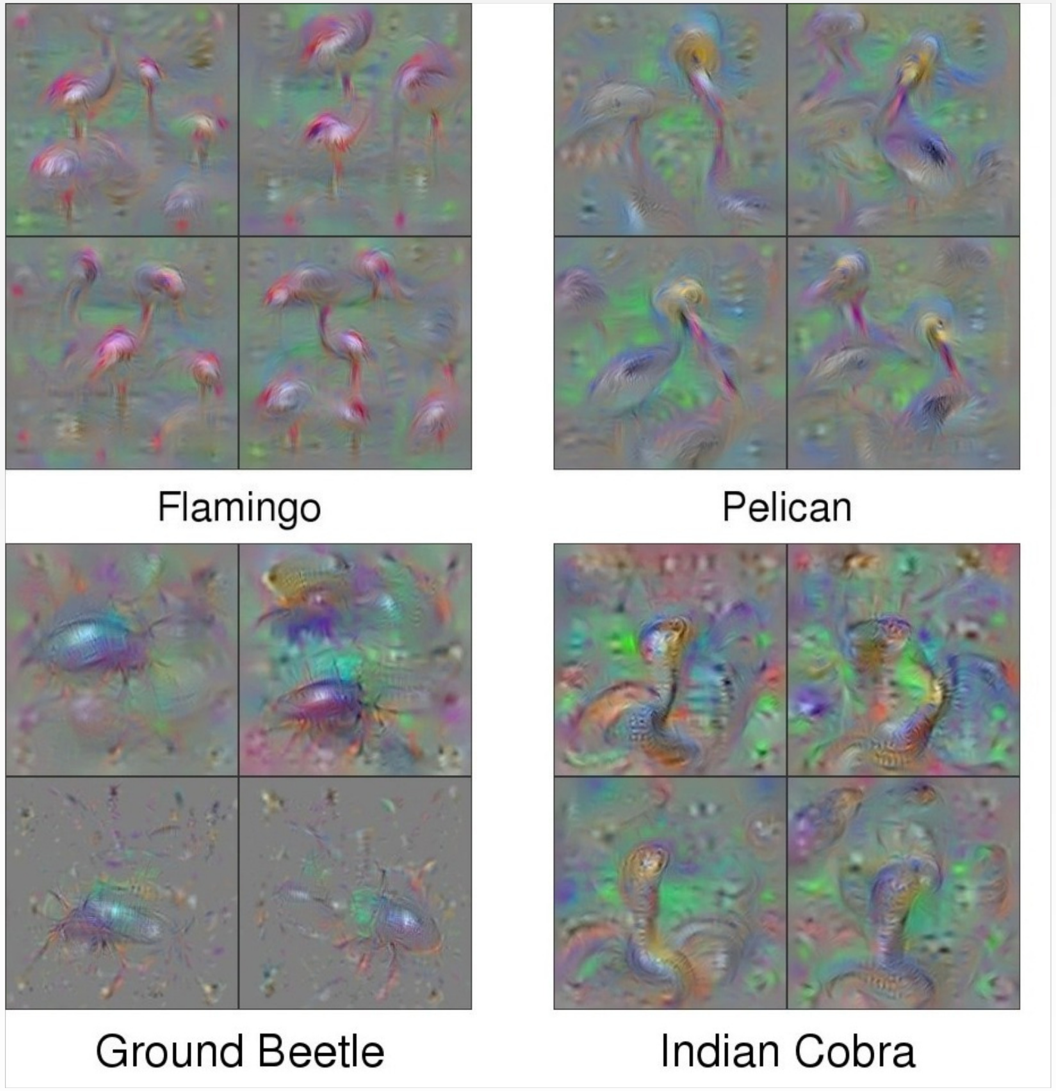
Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.

VISUALIZING CNN FEATURES: GRADIENT ASCENT

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

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

- Gaussian blur image
- Clip pixels with small values to 0
- Clip pixels with small gradients to 0



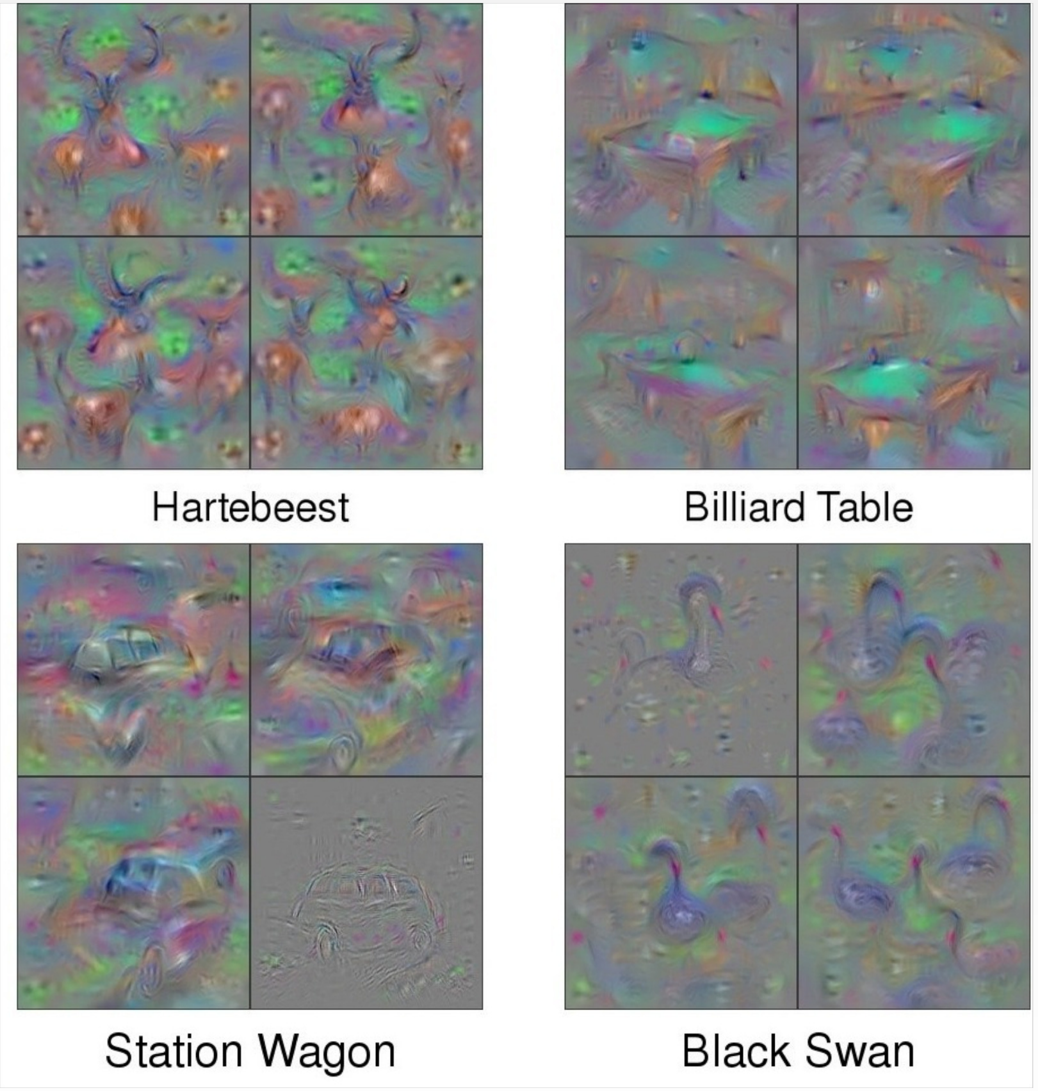
Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.

VISUALIZING CNN FEATURES: GRADIENT ASCENT

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

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

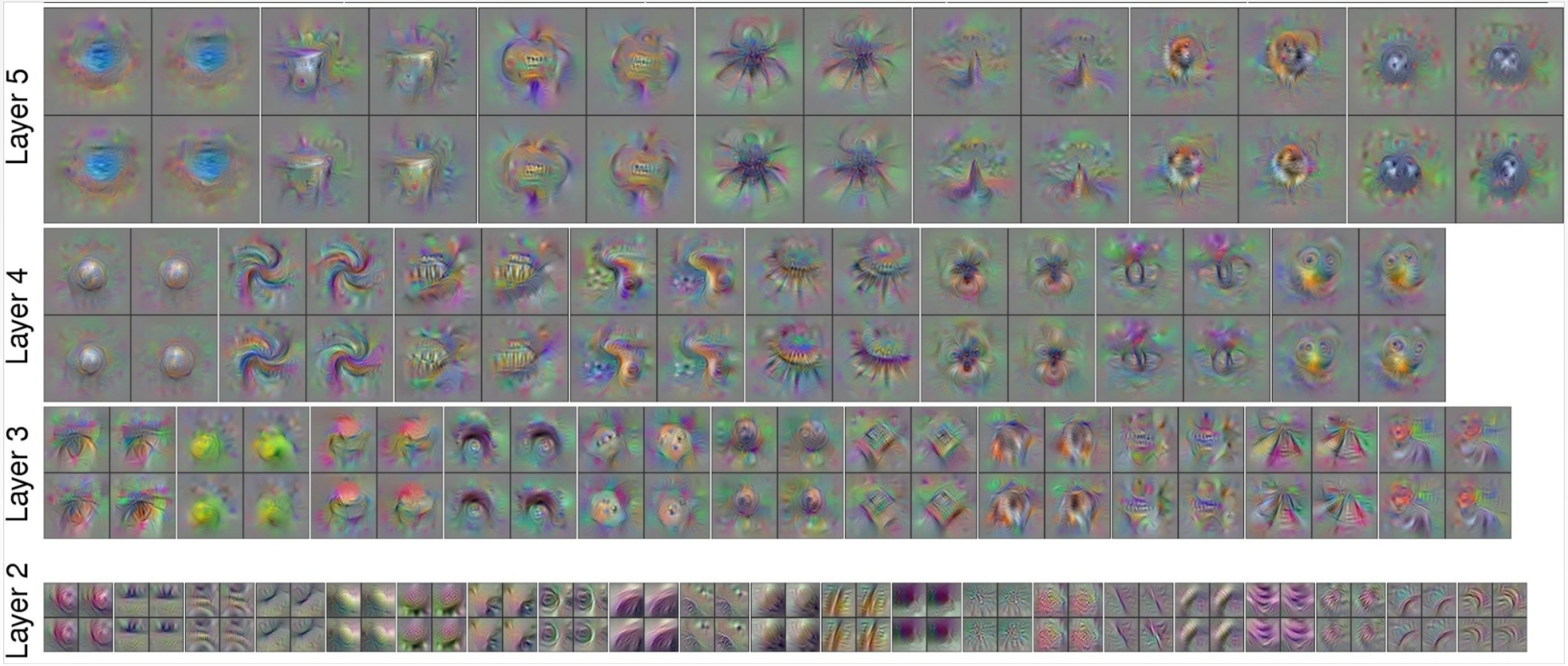
- Gaussian blur image
- Clip pixels with small values to 0
- Clip pixels with small gradients to 0



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

VISUALIZING CNN FEATURES: GRADIENT ASCENT

Use the same approach to visualize intermediate features



Yosinski 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

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

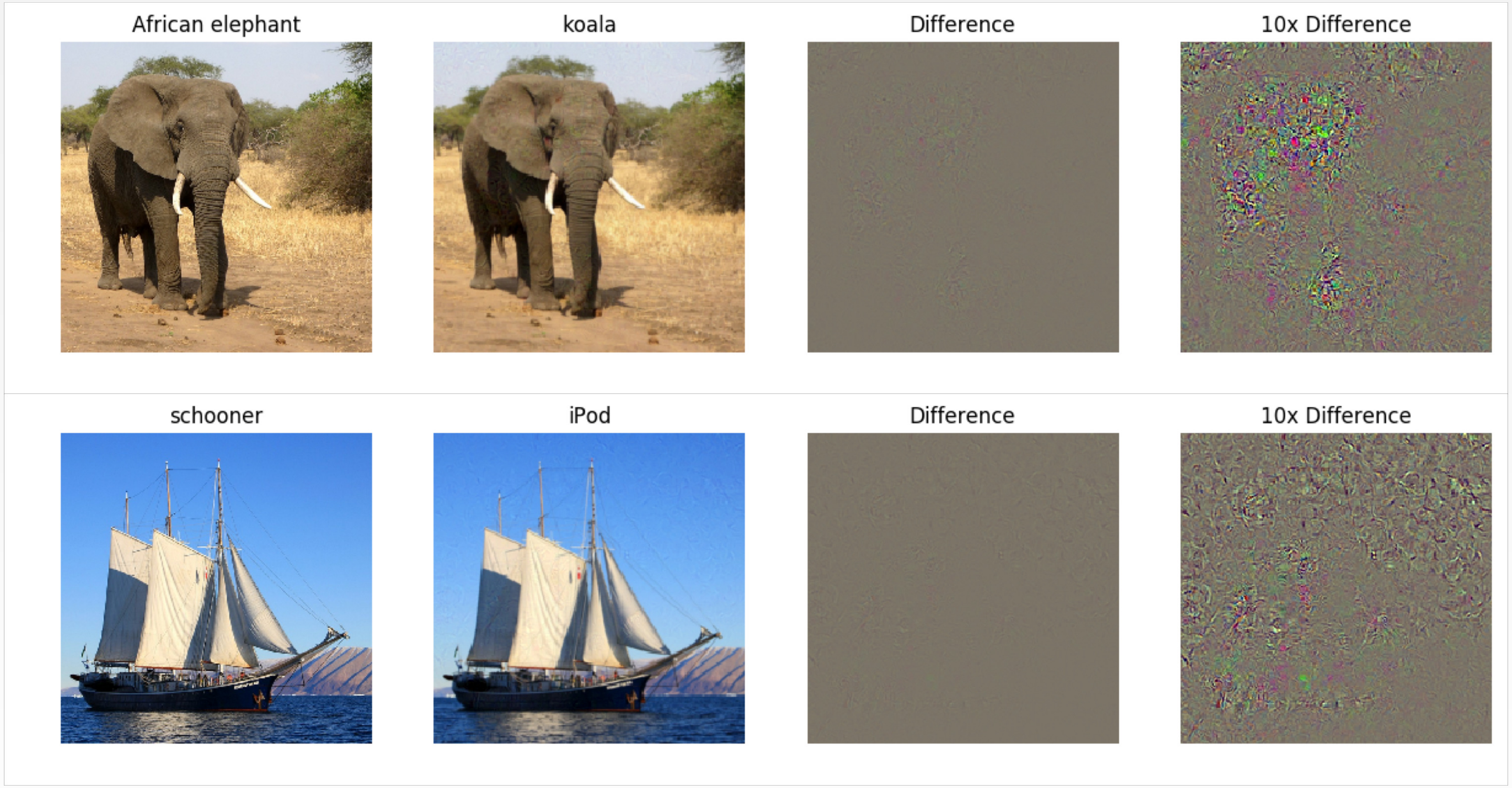
Optimize in FC6 latent space instead of pixel space:

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

FOOLING IMAGES / ADVERSARIAL EXAMPLES

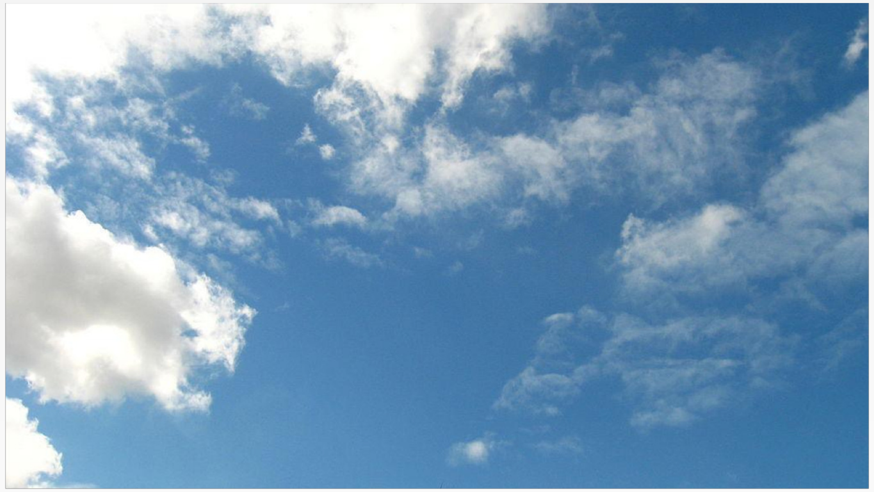
(1) Start from an arbitrary image
(2) Pick an arbitrary class
(3) Modify the image to maximize the class
(4) Repeat until network is fooled

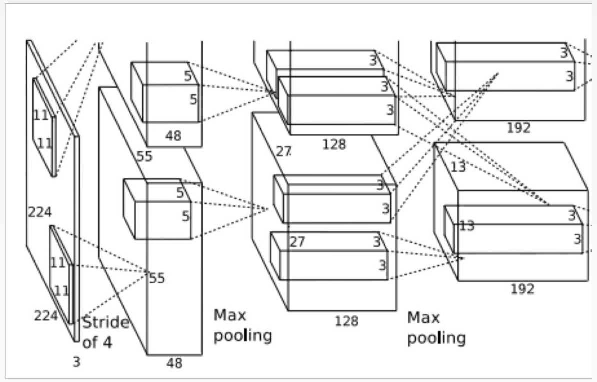
FOOLING IMAGES / ADVERSARIAL EXAMPLES



DEEPDREAM: AMPLIFY EXISTING FEATURES

Rather than synthesizing an image to maximize a specific neuron, instead try to **amplify** the neuron activations at some layer in the network





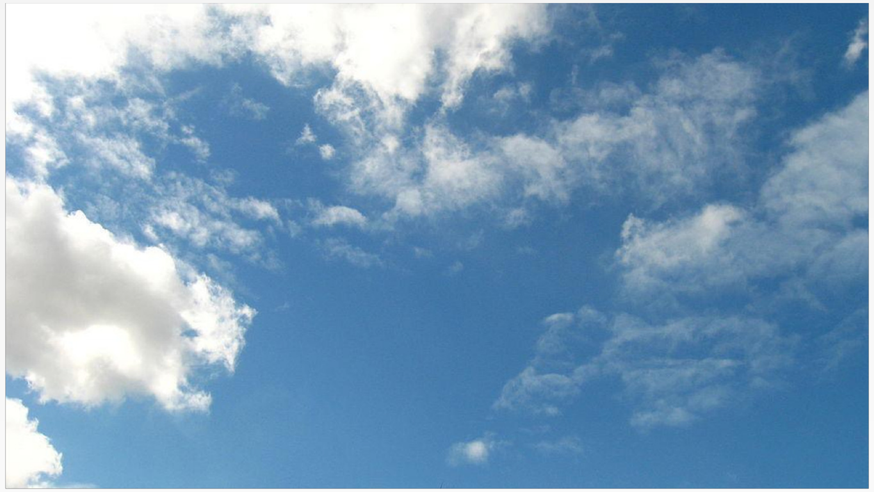
Choose an image and a layer in a CNN; repeat:

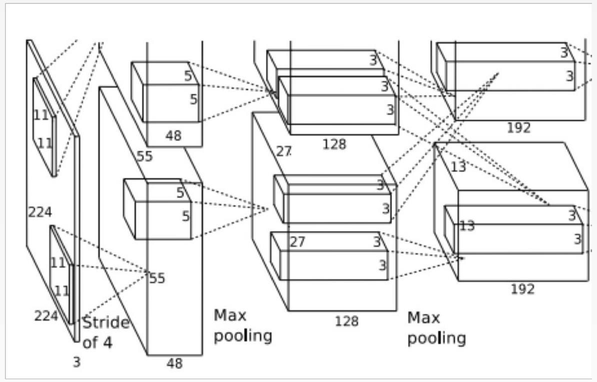
1. Forward: compute activations at chosen layer
2. Set gradient of chosen layer *equal to its activation*
3. Backward: Compute gradient on image
4. Update image

Mordvintsev, Olah, and Tyka, "Inceptionism: Going Deeper into Neural Networks", Google Research Blog.

DEEPDREAM: AMPLIFY EXISTING FEATURES

Rather than synthesizing an image to maximize a specific neuron, instead try to **amplify** the neuron activations at some layer in the network





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

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

Equivalent to:

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

Mordvintsev, Olah, and Tyka, "Inceptionism: Going Deeper into Neural Networks", Google Research Blog. Images are licensed under CC-BY 4.0

DEEPDREAM: AMPLIFY EXISTING FEATURES

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

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

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

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

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

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

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

Code is very simple but it uses a couple tricks:

(Code is licensed under Apache 2.0)

DEEPDREAM: AMPLIFY EXISTING FEATURES

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

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

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

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

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

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

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

Code is very simple but it uses a couple tricks:

(Code is licensed under Apache 2.0)

Jitter image

DEEPDREAM: AMPLIFY EXISTING FEATURES

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

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

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

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

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

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

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

Code is very simple but it uses a couple tricks:

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:

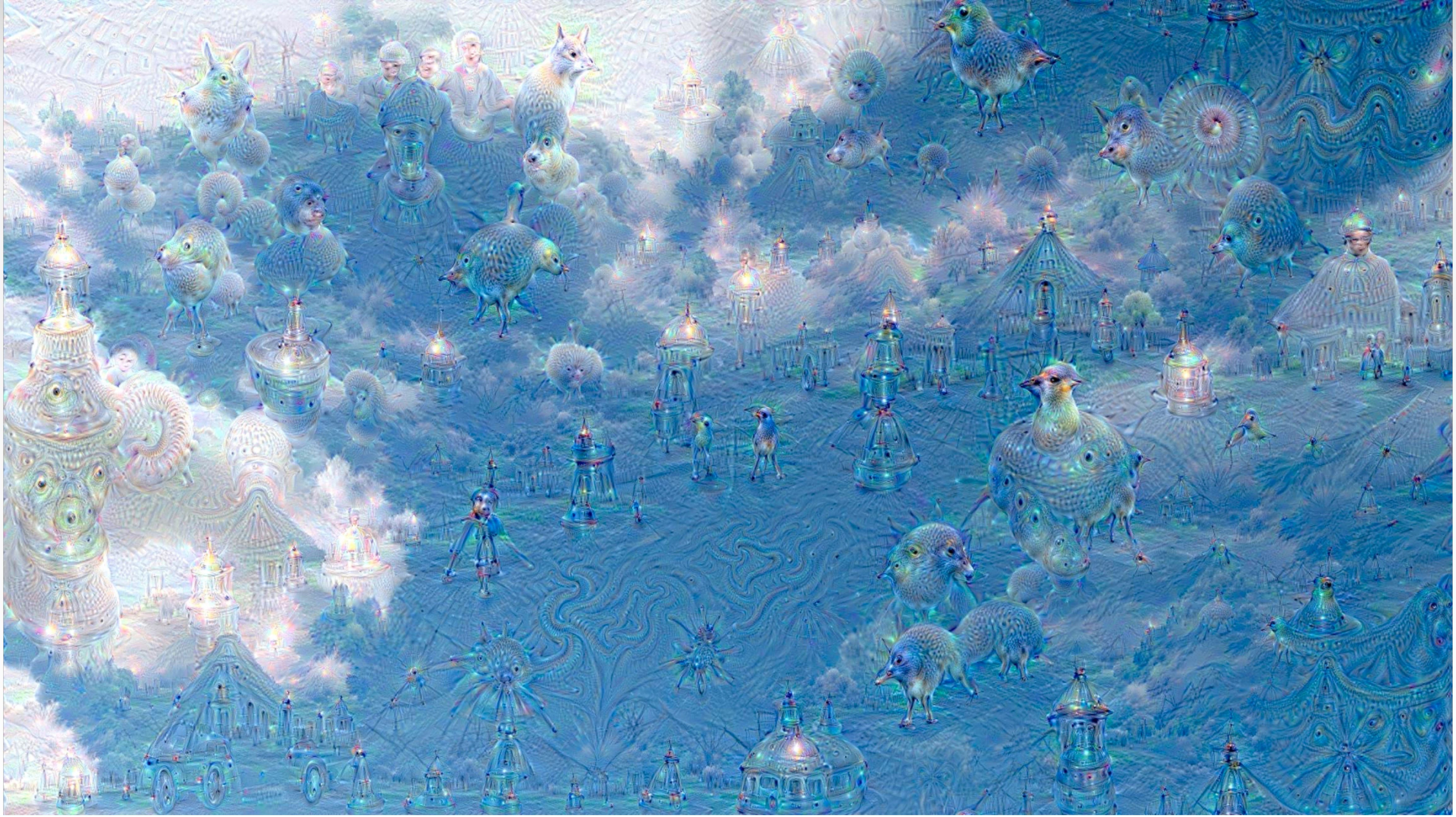
(Code is licensed under Apache 2.0)

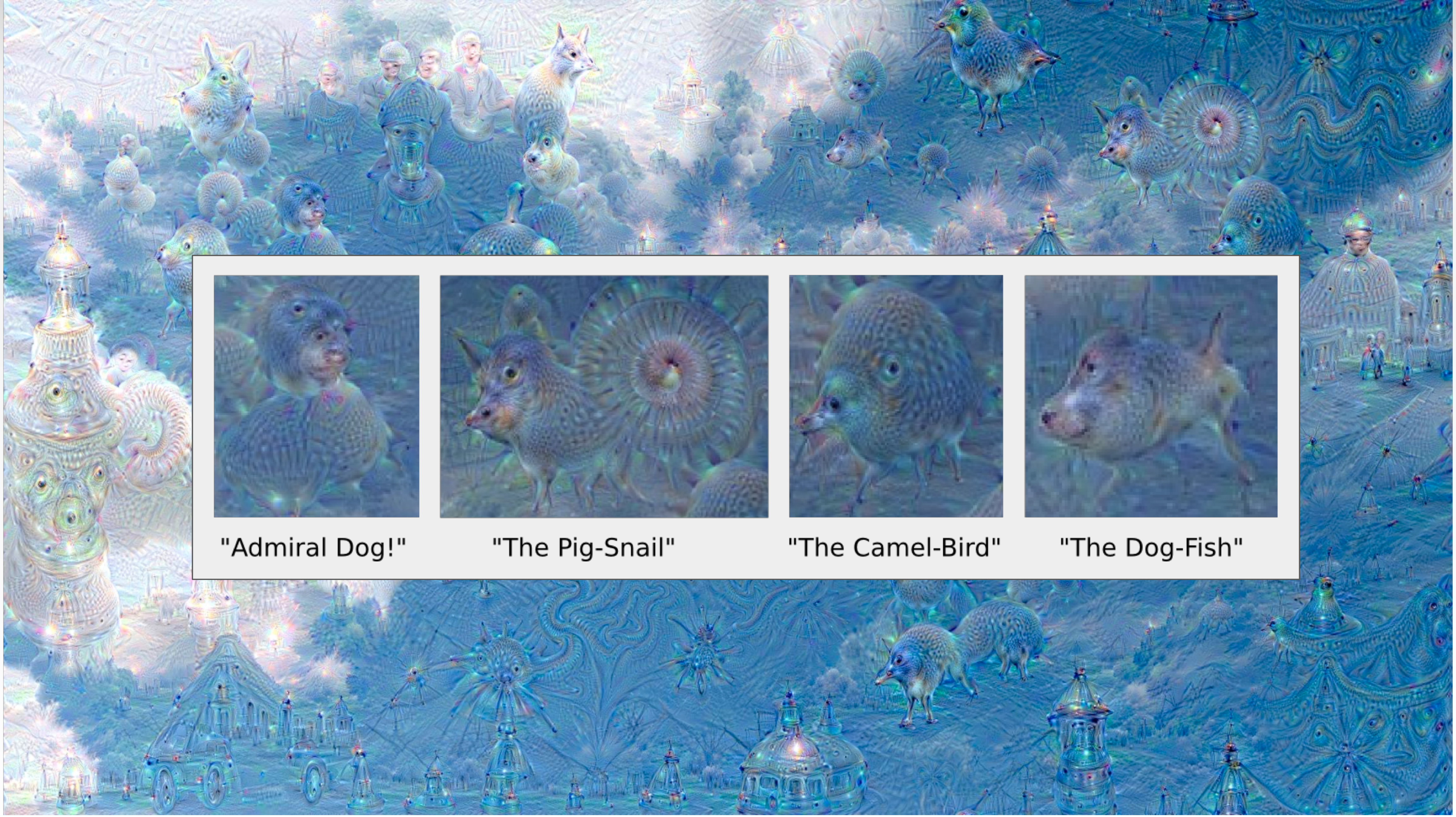
Jitter image

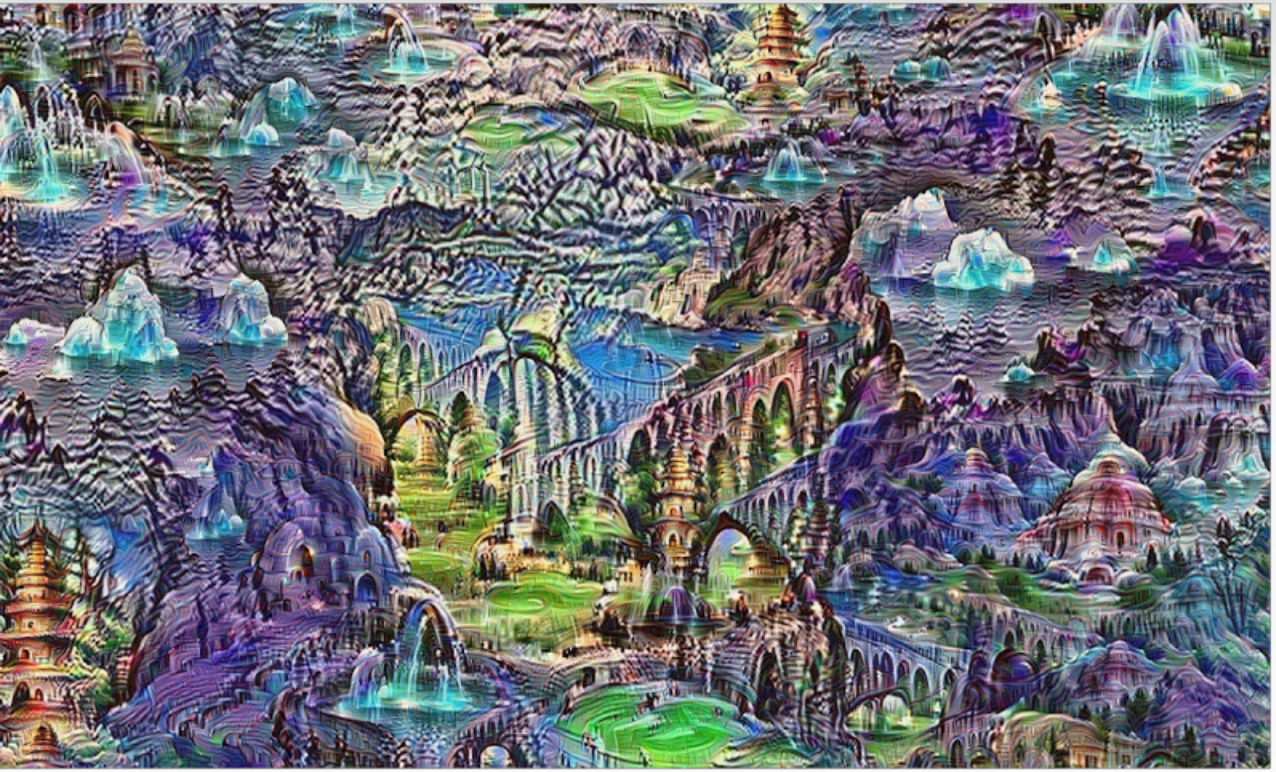
L1 Normalize gradients

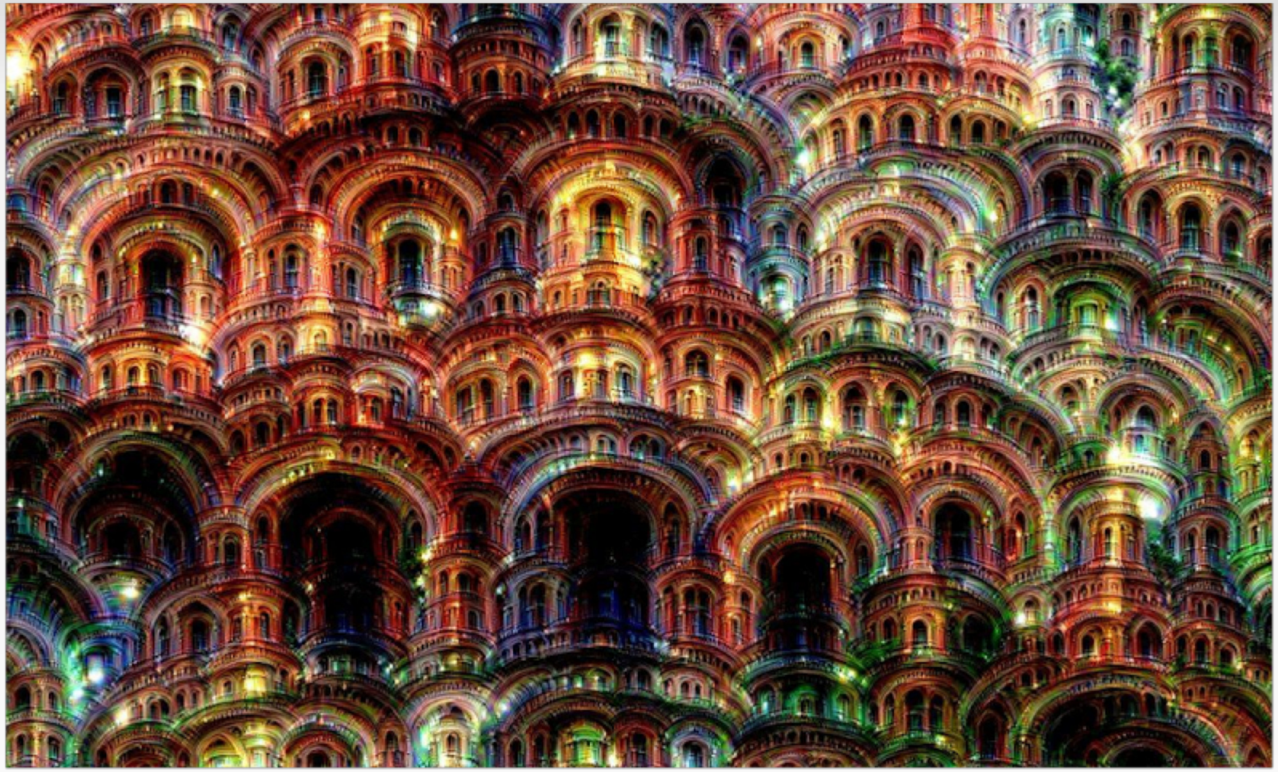
Clip pixel values

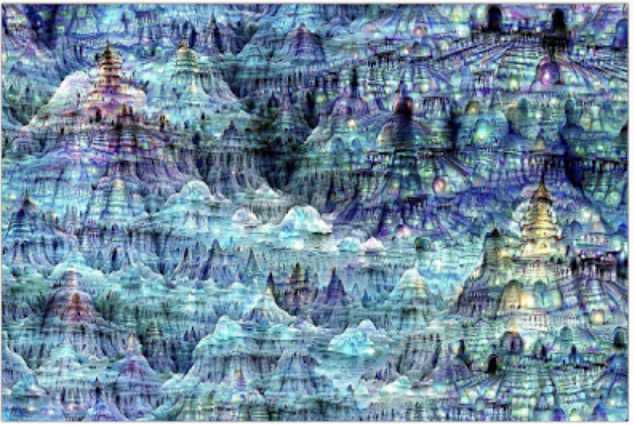
Also uses multiscale processing for a fractal effect (not shown)

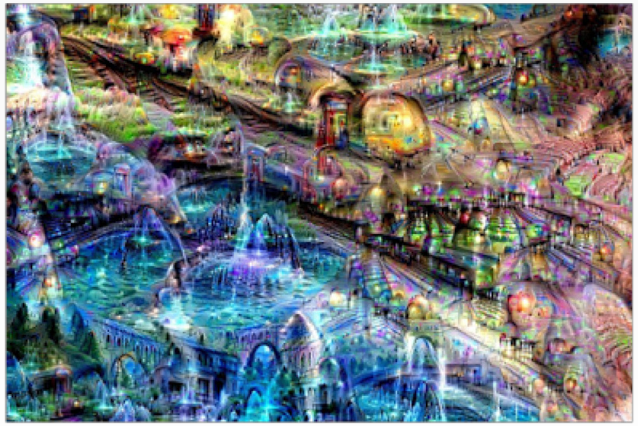


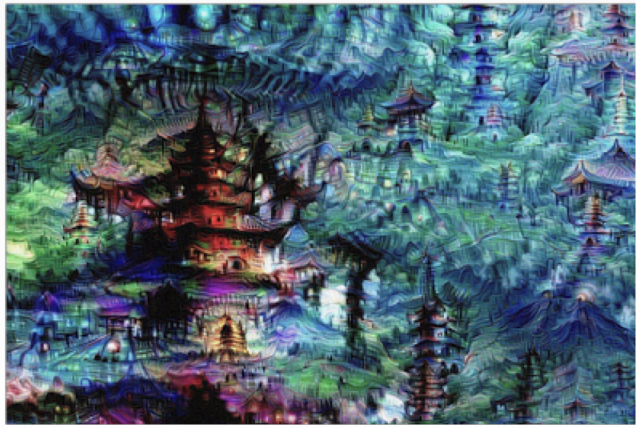

"Admiral Dog!"
"The Pig-Snail"
"The Camel-Bird"
"The Dog-Fish"











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

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

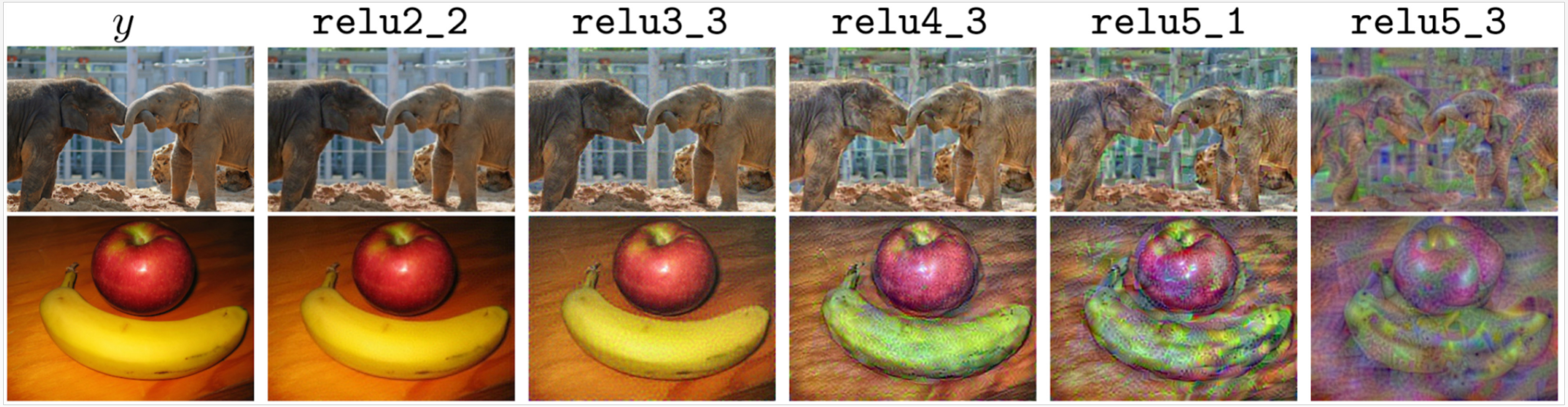
$$\mathcal{R}_{V^\beta}(\mathbf{x}) = \sum_{i,j}\left(\left(x_{i,j+1} - x_{ij}\right)^2 + \left(x_{i+1,j} - x_{ij}\right)^2\right)^{\frac{\beta}{2}}$$

Total Variation regularizer (encourages spatial smoothness)

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

FEATURE INVERSION

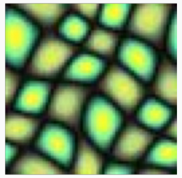
Reconstructing from different layers of VGG-16



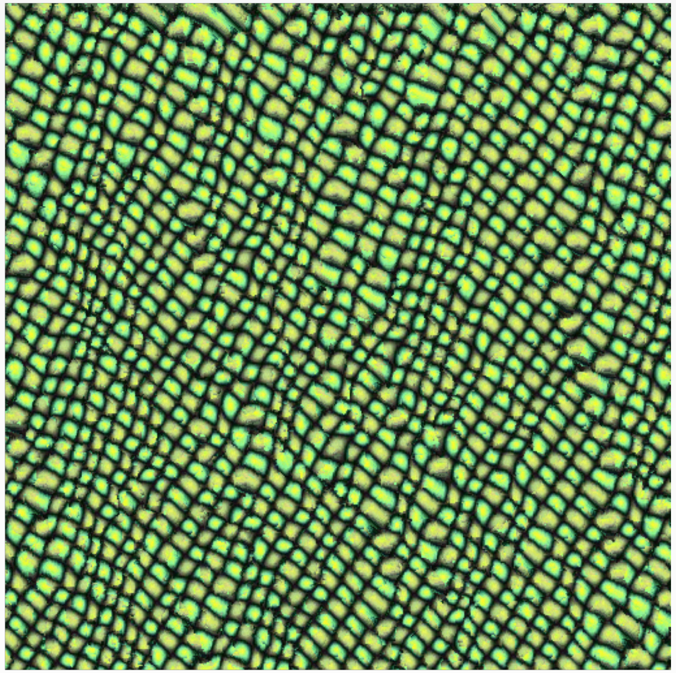
Mahendran and Vedaldi, "Understanding Deep Image Representations by Inverting Them", CVPR 2015
Figure from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016. Copyright Springer, 2016. Reproduced for educational purposes.

TEXTURE SYNTHESIS

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



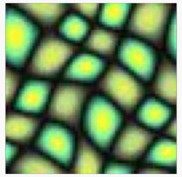
Input

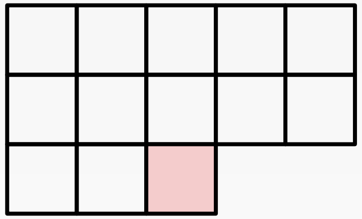


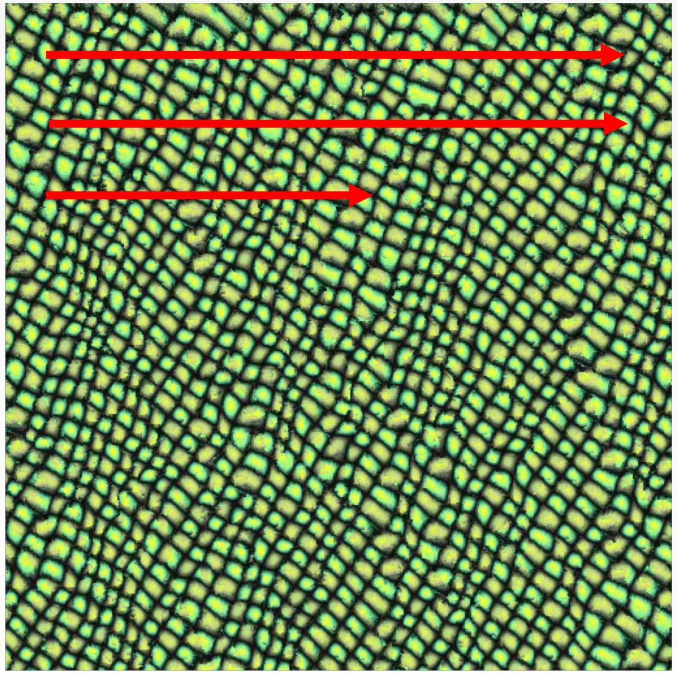
Output

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

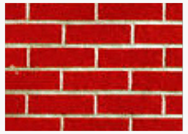


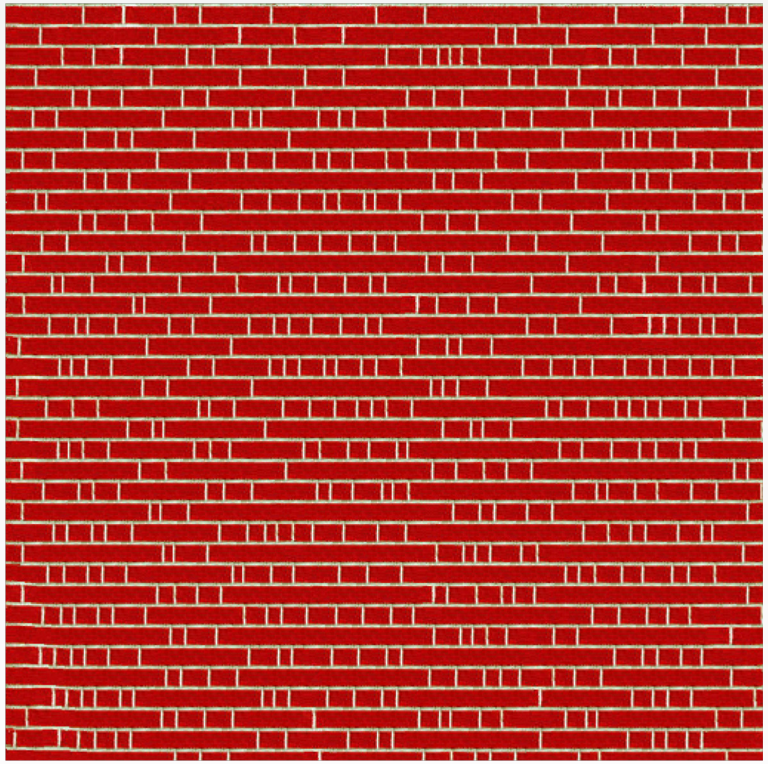




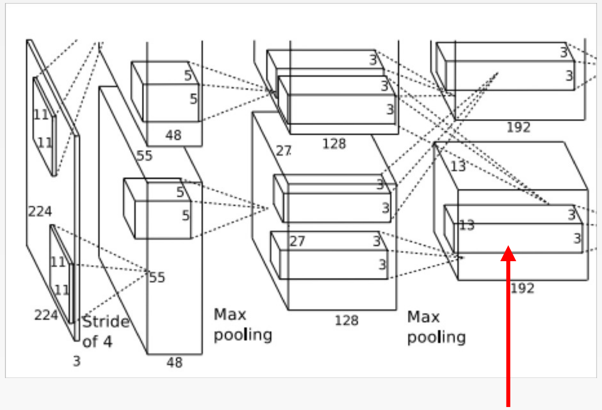
Wei and Levoy, "Fast Texture Synthesis using Tree-structured Vector Quantization", SIGGRAPH 2000
Efros and Leung, "Texture Synthesis by Non-parametric Sampling", ICCV 1999

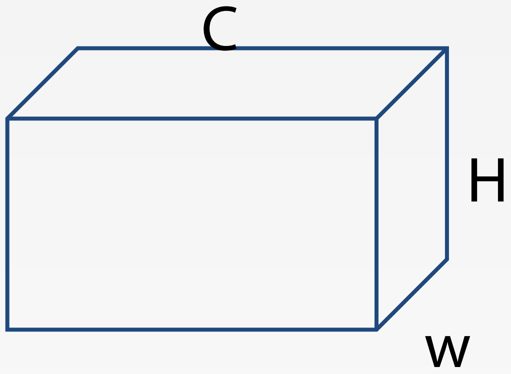
TEXTURE SYNTHESIS: NEAREST NEIGHBOR





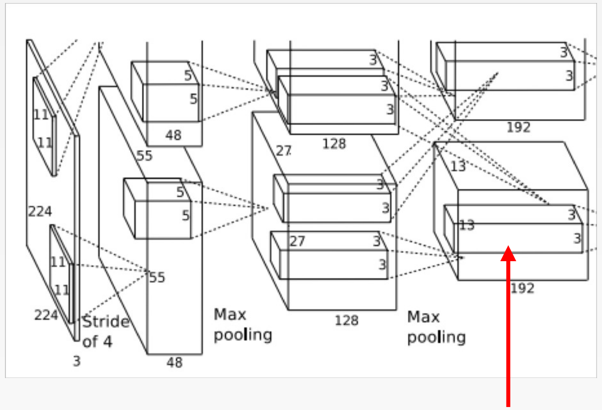
NEURAL TEXTURE SYNTHESIS: GRAM MATRIX

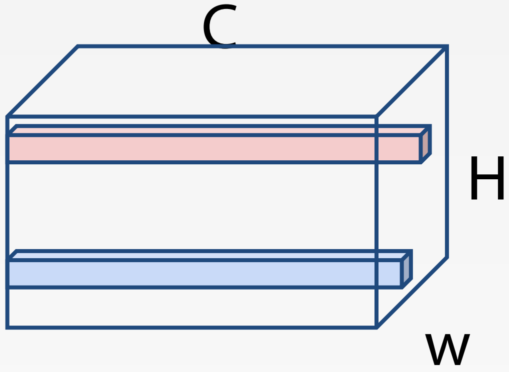


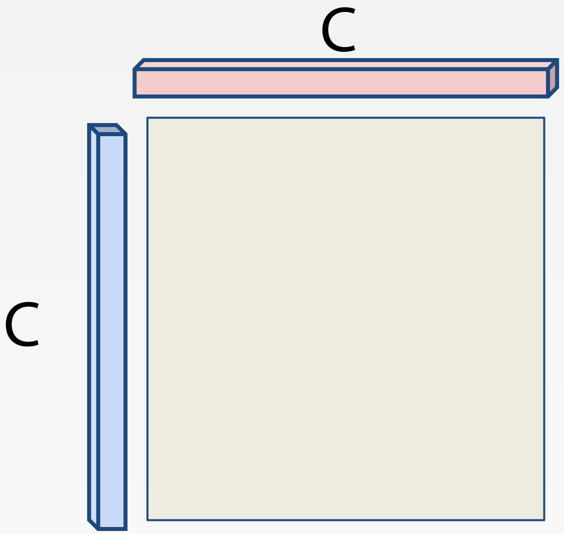


Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

NEURAL TEXTURE SYNTHESIS: GRAM MATRIX



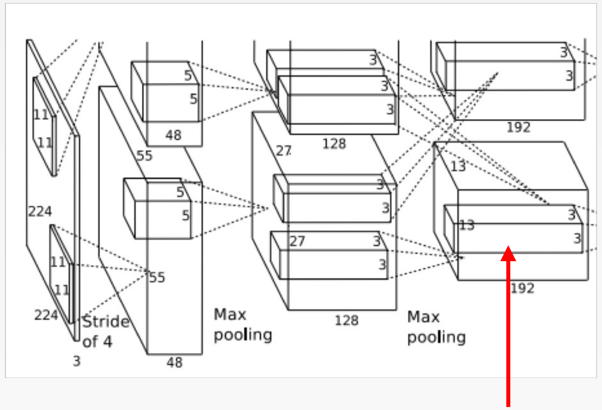


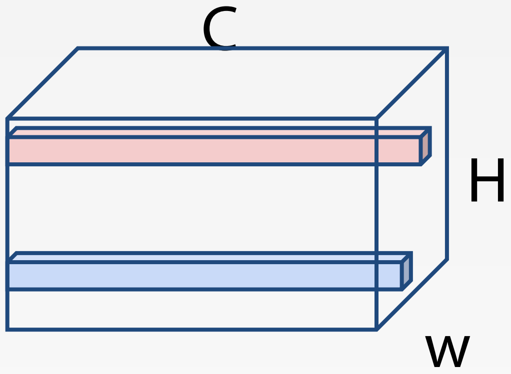


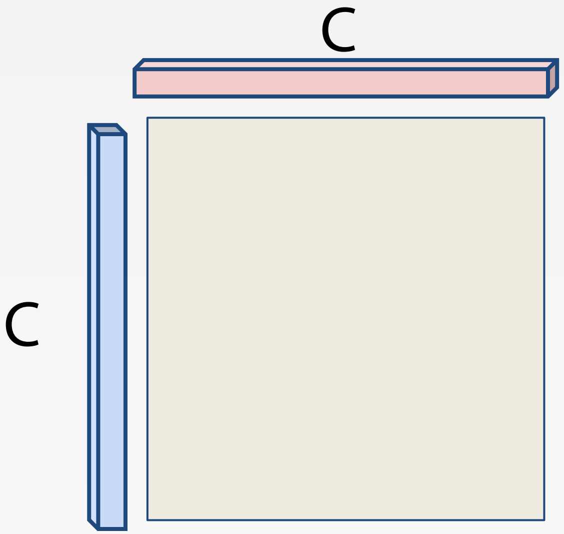
Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

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

NEURAL TEXTURE SYNTHESIS: GRAM MATRIX







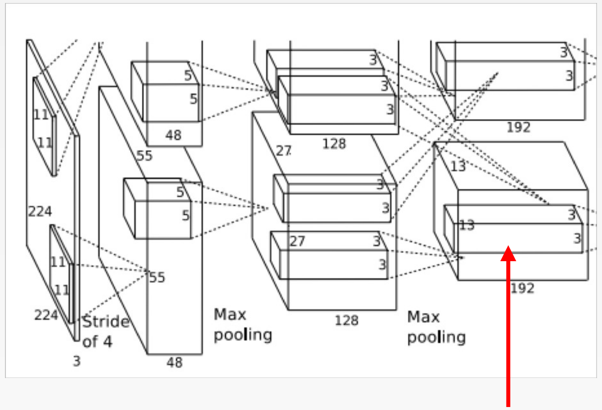
Gram Matrix

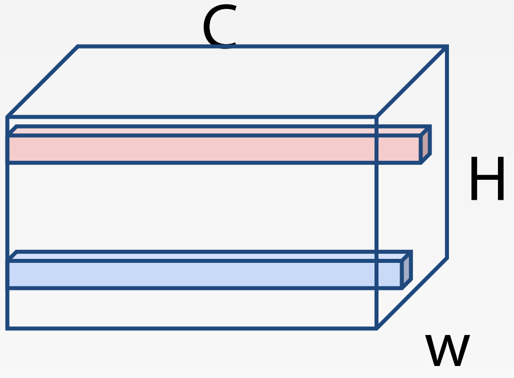
Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

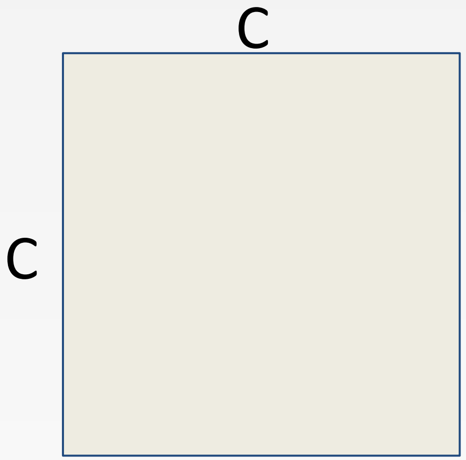
Outer product of two C-dimensional vectors gives C x C matrix measuring co-occurrence

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

NEURAL TEXTURE SYNTHESIS: GRAM MATRIX







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

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

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

Efficient to compute; reshape features from

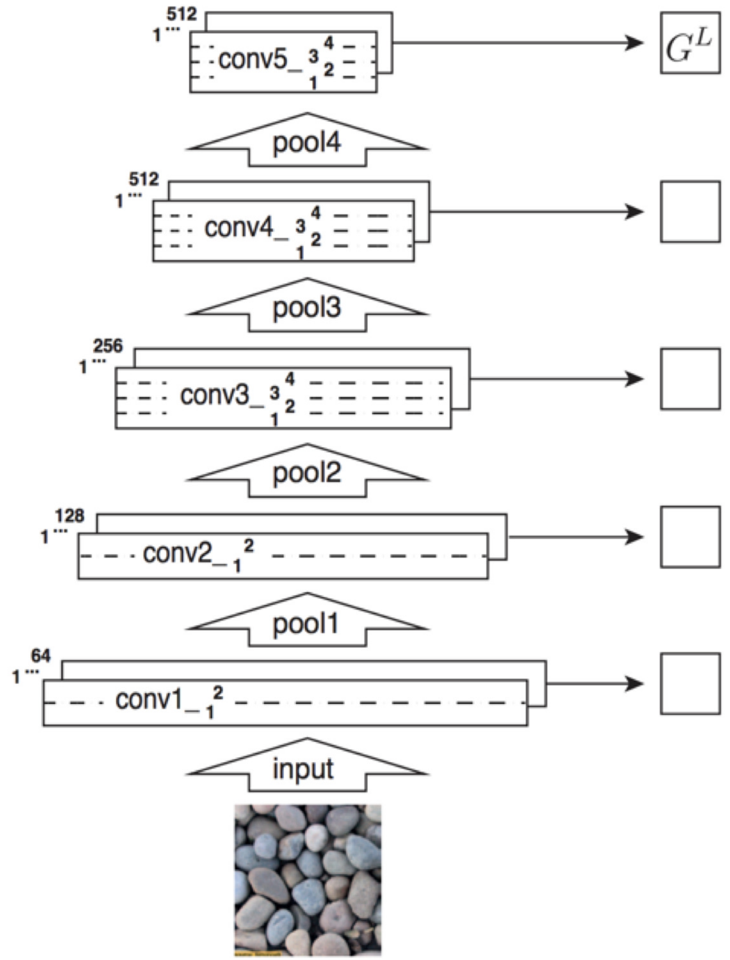
C x H x W to =C x HW

then compute $G = FF^T$

NEURAL TEXTURE SYNTHESIS

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

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



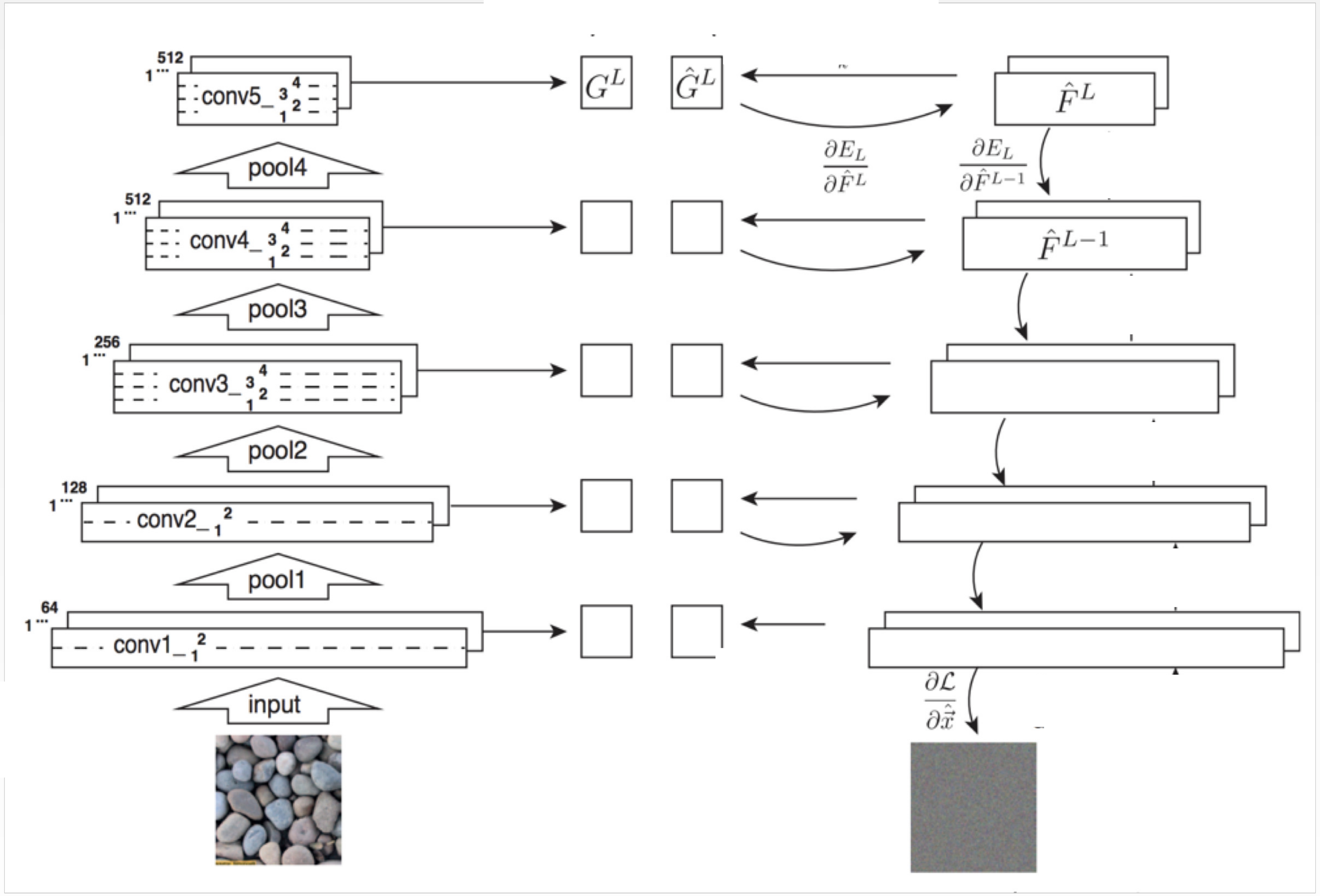
Gatys, Ecker, and Bethge, "Texture Synthesis Using Convolutional Neural Networks", NIPS 2015

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:

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

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



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

NEURAL TEXTURE SYNTHESIS

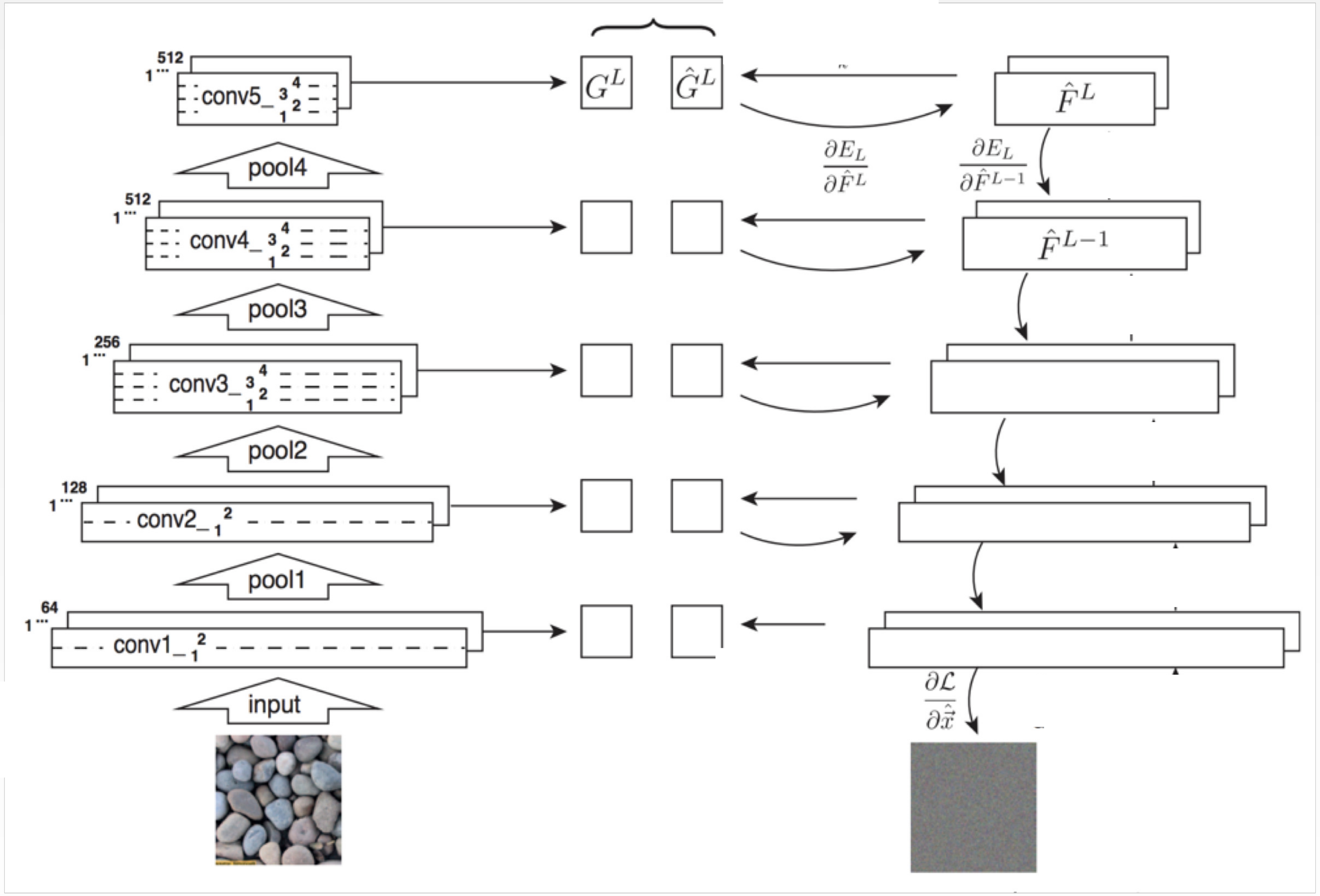
$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - \hat{G}_{ij}^l \right)^2$$

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

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

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

4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices



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

NEURAL TEXTURE SYNTHESIS

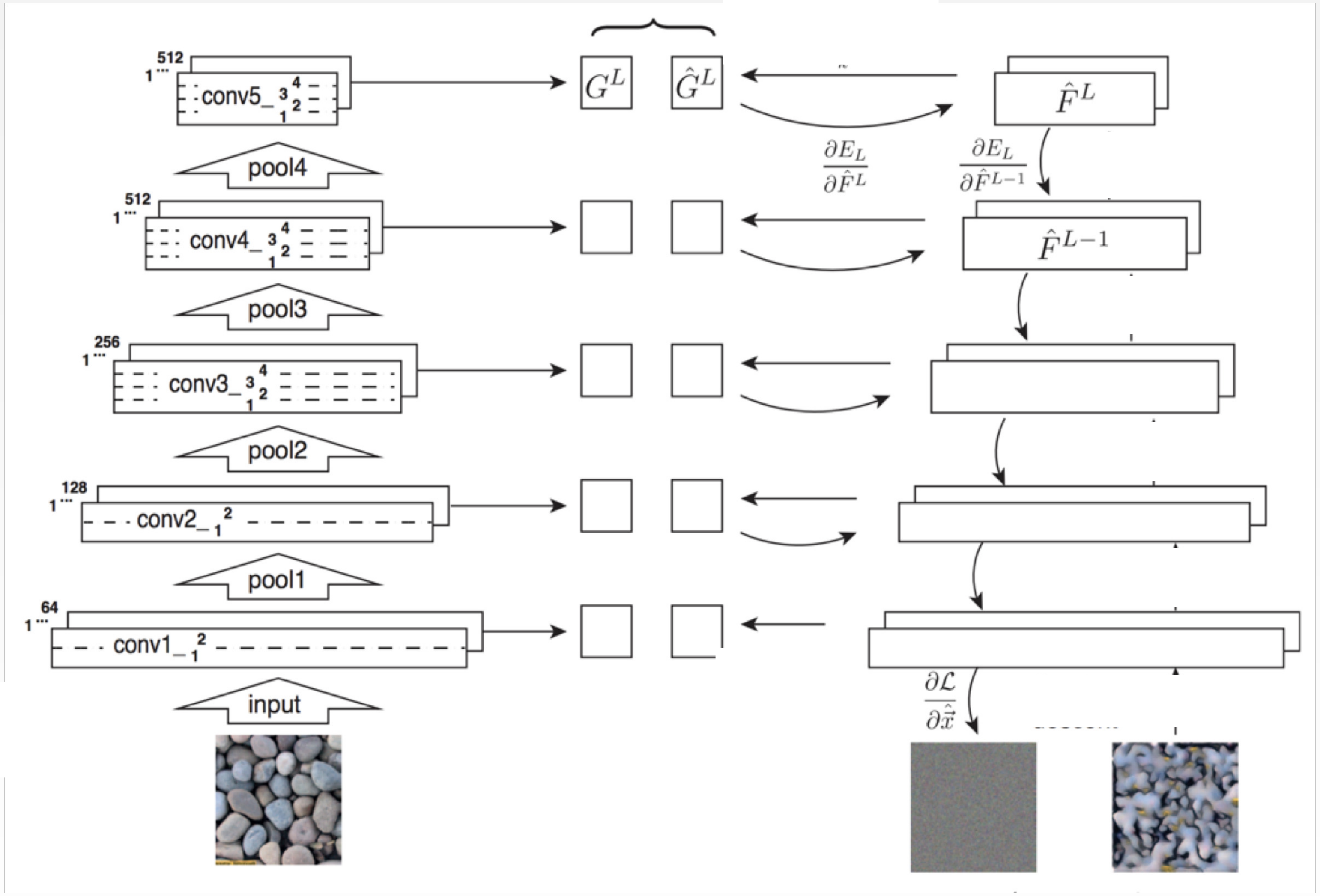
$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G^l_{ij} - \hat{G}^l_{ij} \right)^2$$

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

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:

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

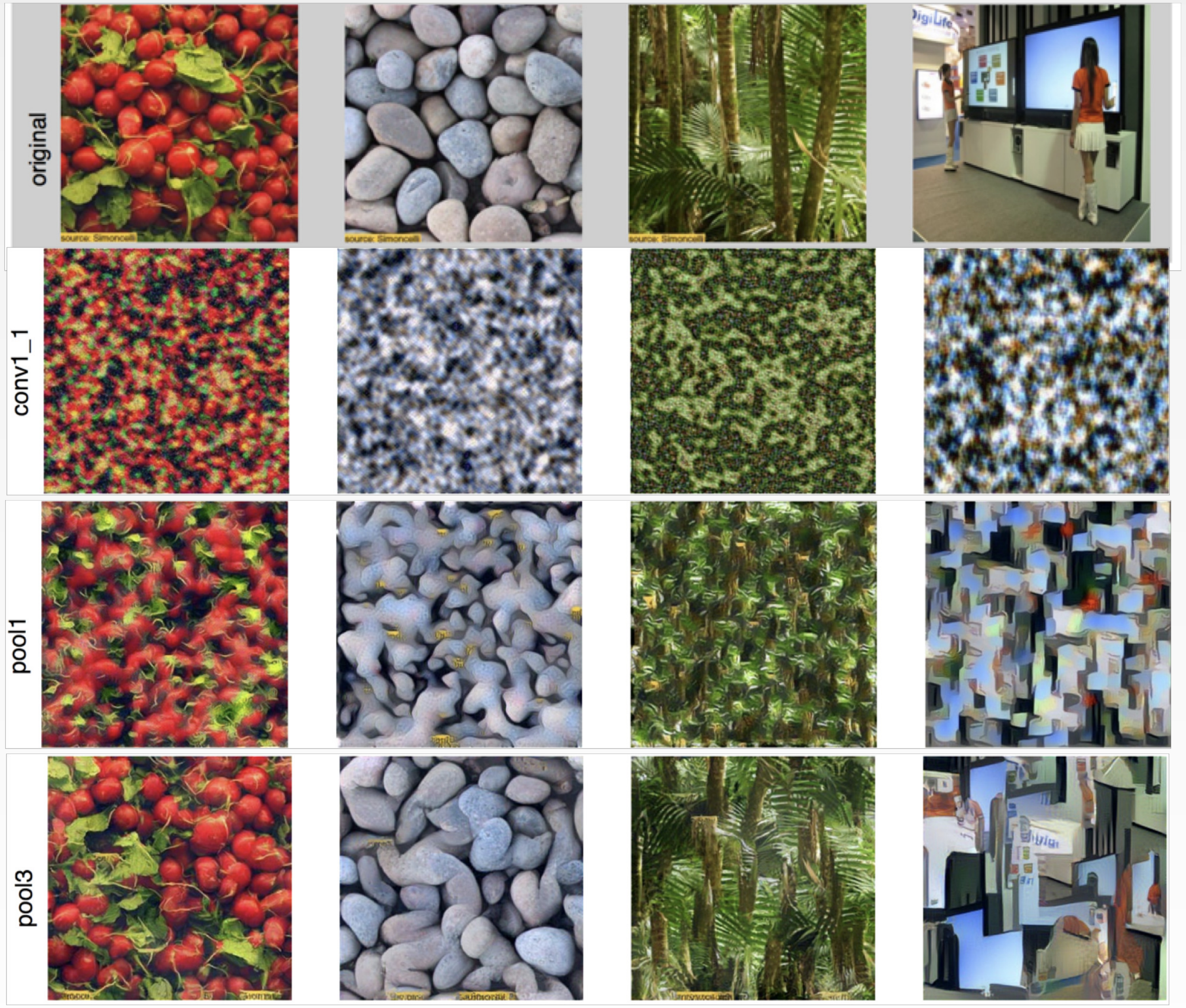
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5



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

NEURAL TEXTURE SYNTHESIS

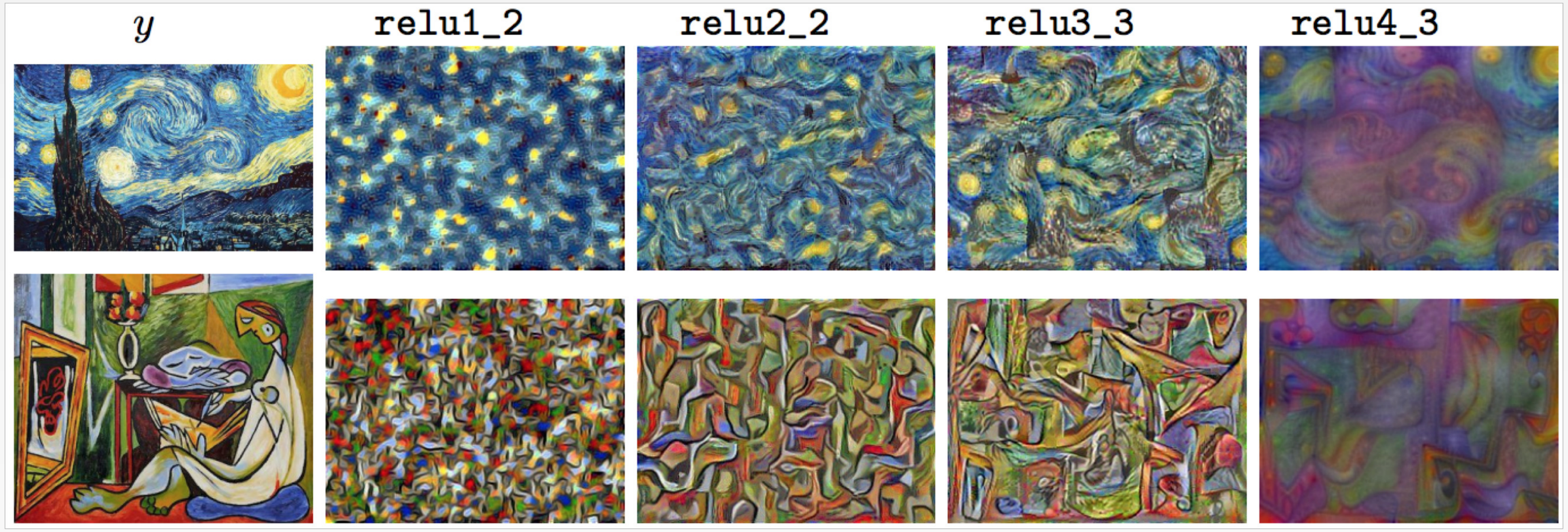
Reconstructing texture from higher layers recovers larger features from the input texture



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

NEURAL TEXTURE SYNTHESIS

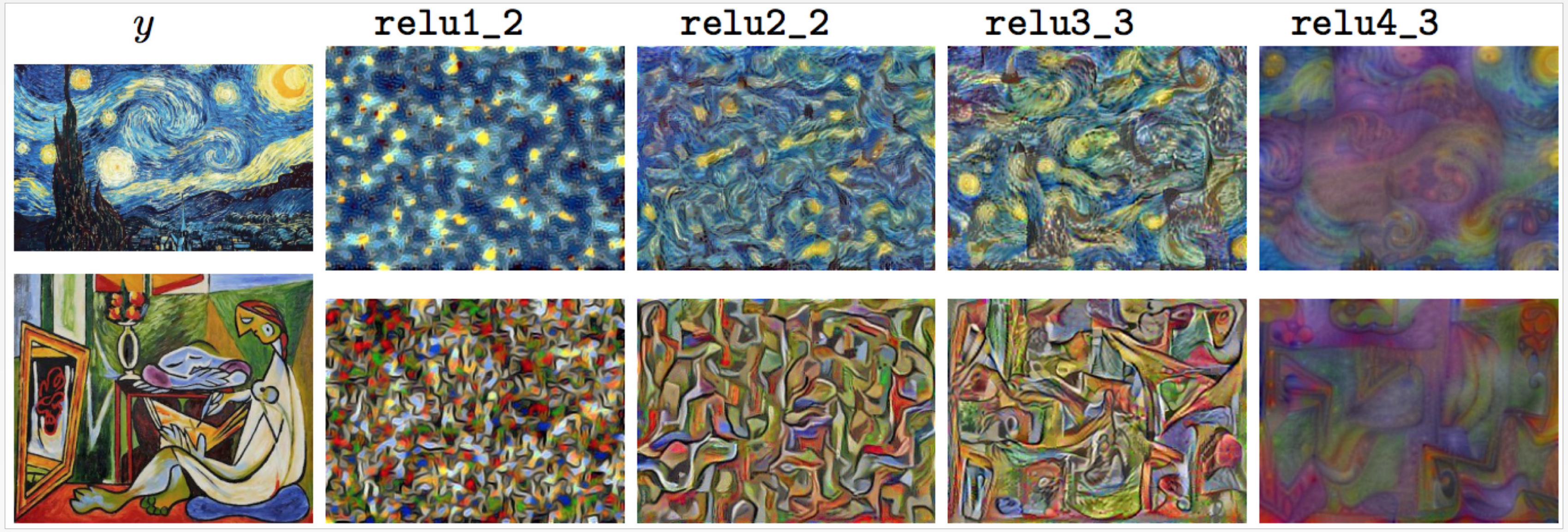
Texture synthesis (Gram reconstruction)



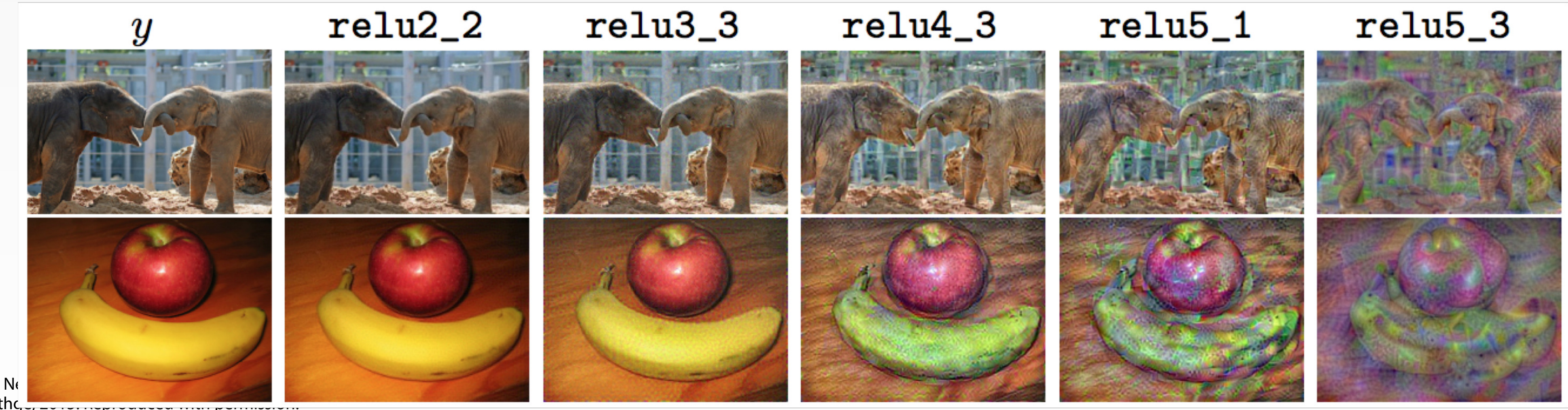
Gatys, Ecker, and Bethge, "Texture Synthesis Using Convolutional Neural Networks", NIPS 2015

NEURAL TEXTURE SYNTHESIS

Texture synthesis (Gram reconstruction)



Feature reconstruction



Gatys, Ecker, and Bethge, “Texture Synthesis Using Convolutional Ne
Figure copyright Leon Gatys, Alexander S. Ecker, and Matthias Bethge, 2015. Reproduced with permission.

NEURAL STYLE TRANSFER

Content Image

+

Style Image

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

NEURAL STYLE TRANSFER

Content Image

+

Style Image

=

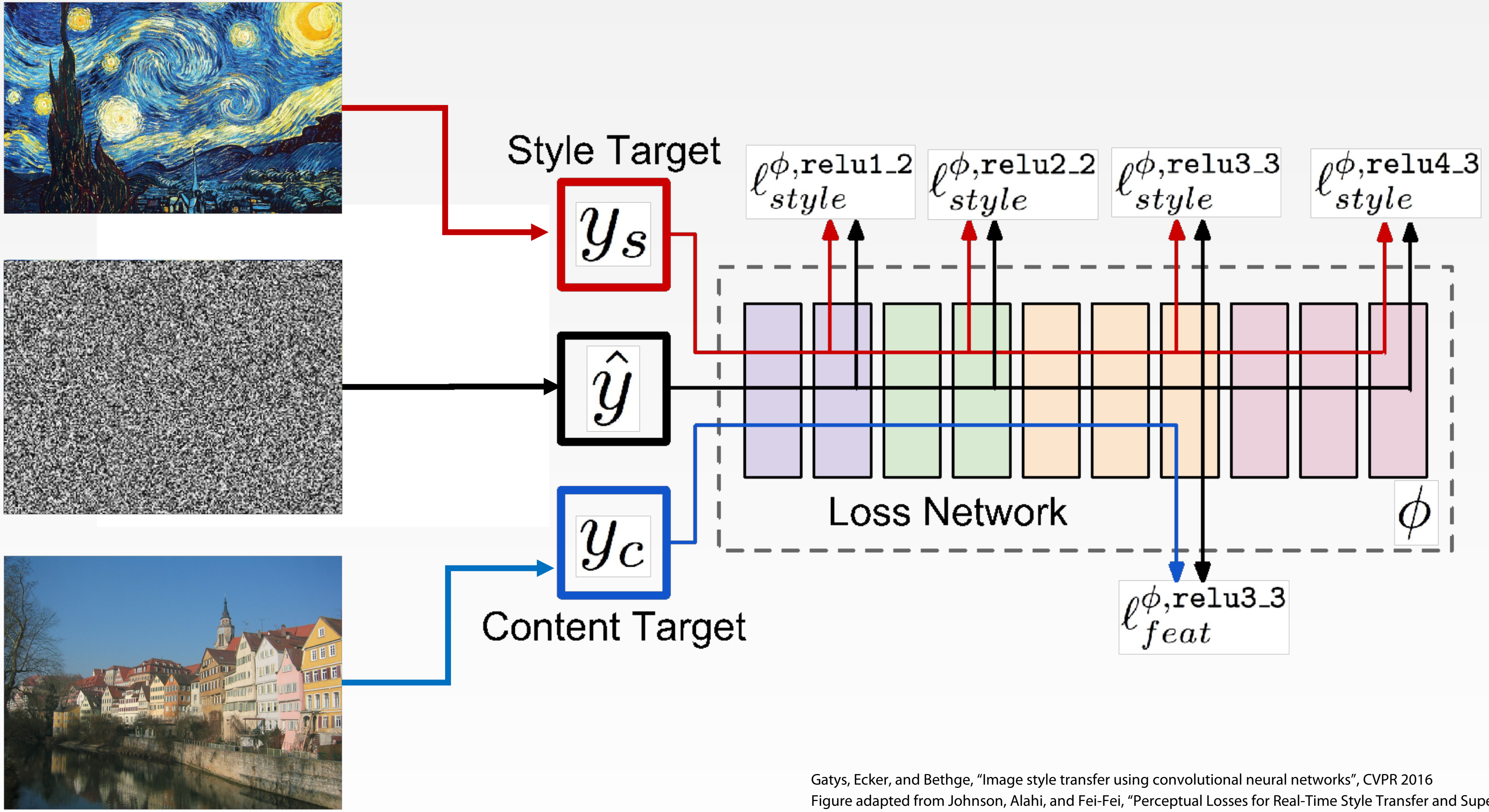
Style Transfer!

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

Style image

Output image (Start with noise)

Content image

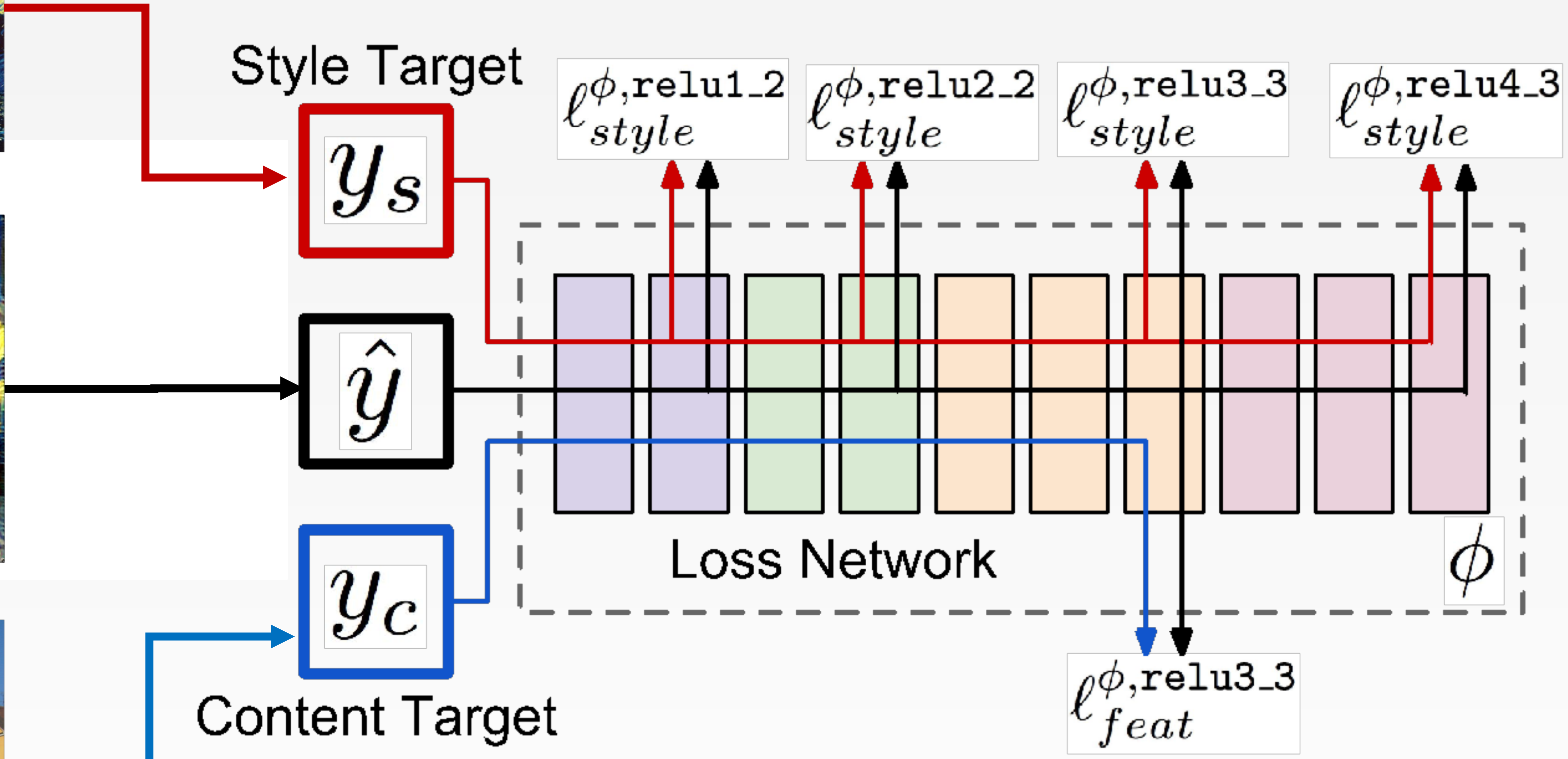


Gatys, Ecker, and Bethge, "Image style transfer using convolutional neural networks", CVPR 2016
Figure adapted from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016. Copyright Springer, 2016. Reproduced for educational purposes.

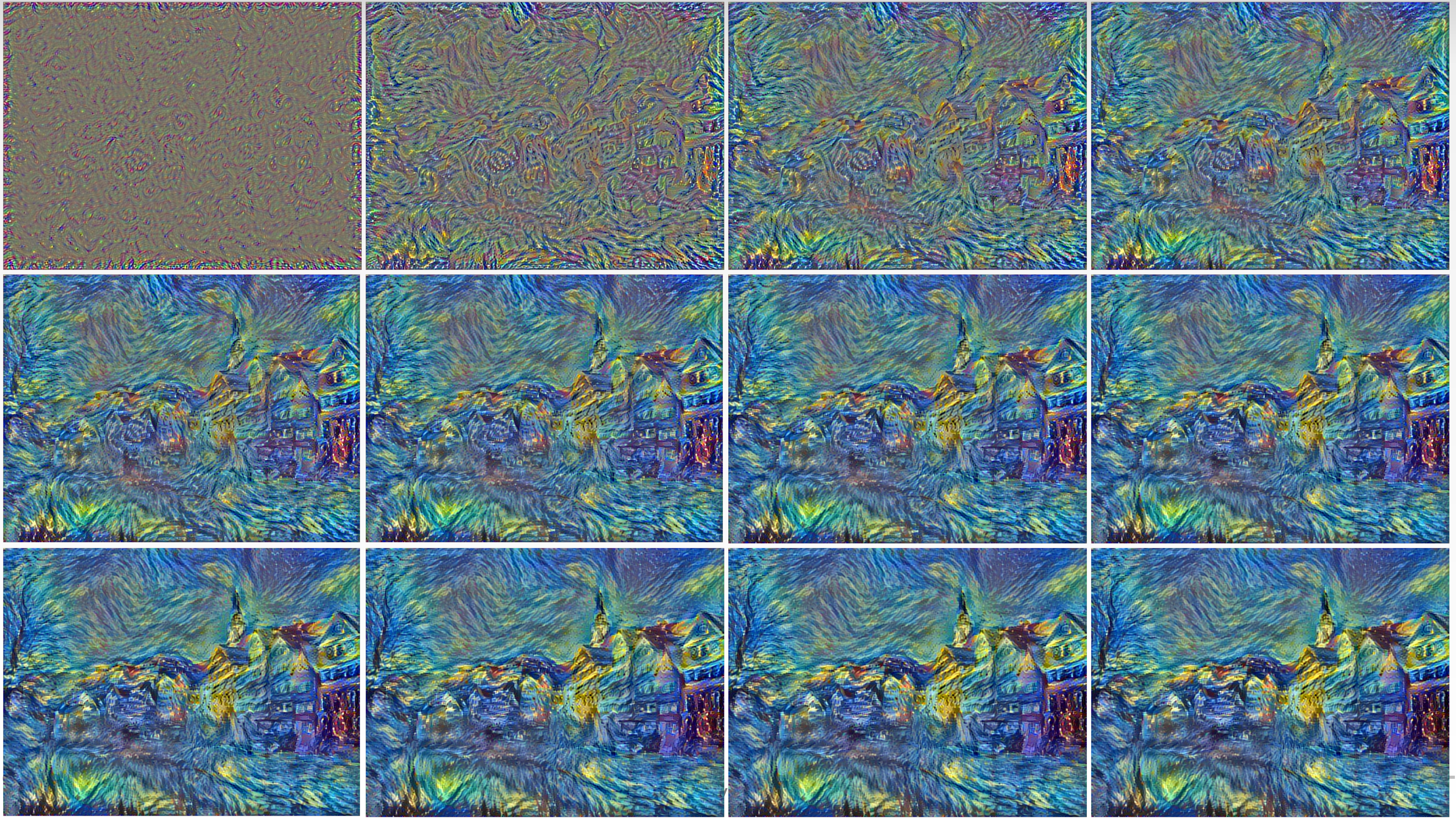
Style image

Output image (Start with noise)

Content image



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



NEURAL STYLE TRANSFER

Example outputs from 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

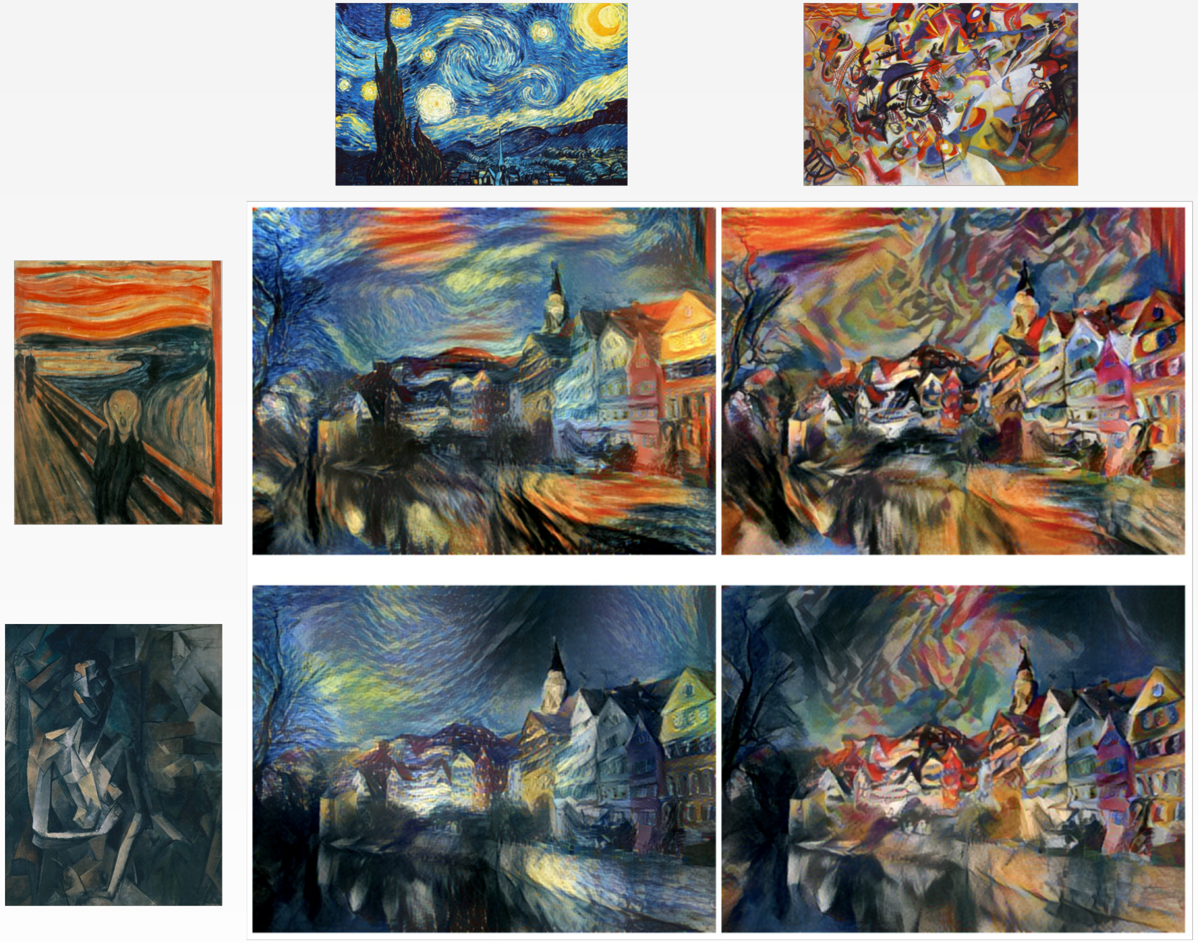
Resizing style image before running style transfer algorithm can transfer different types of features

Larger style image ⟷ Smaller style image

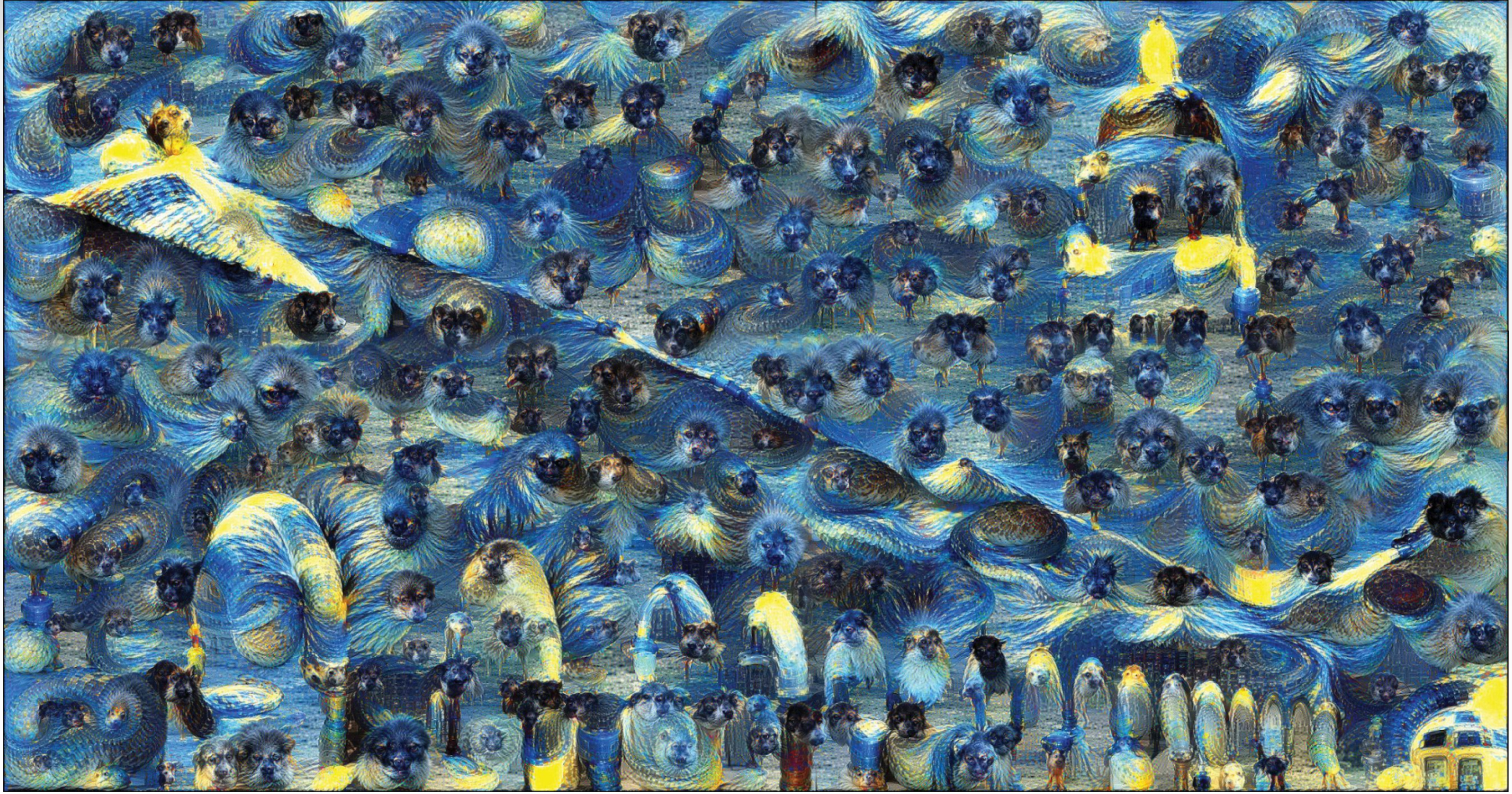
Gatys, Ecker, and Bethge, "Image style transfer using convolutional neural networks", CVPR 2016

NEURAL STYLE TRANSFER: MULTIPLE STYLE IMAGES

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



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



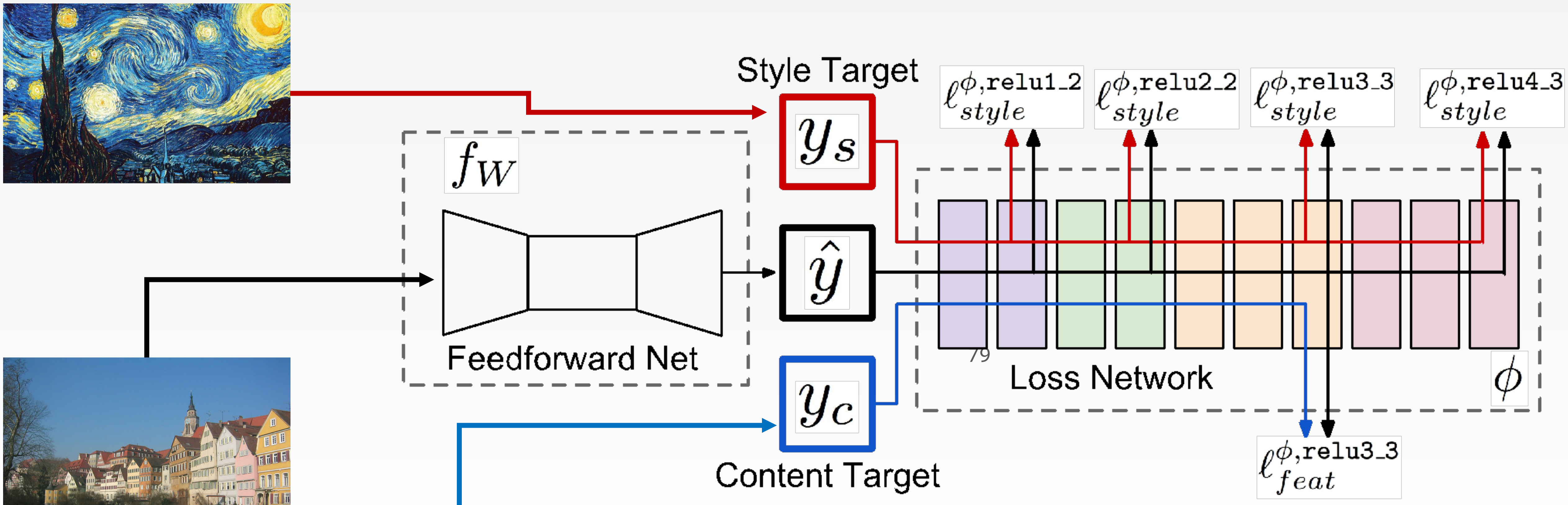
Problem: Style transfer requires many forward / backward passes through VGG; very slow!

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

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

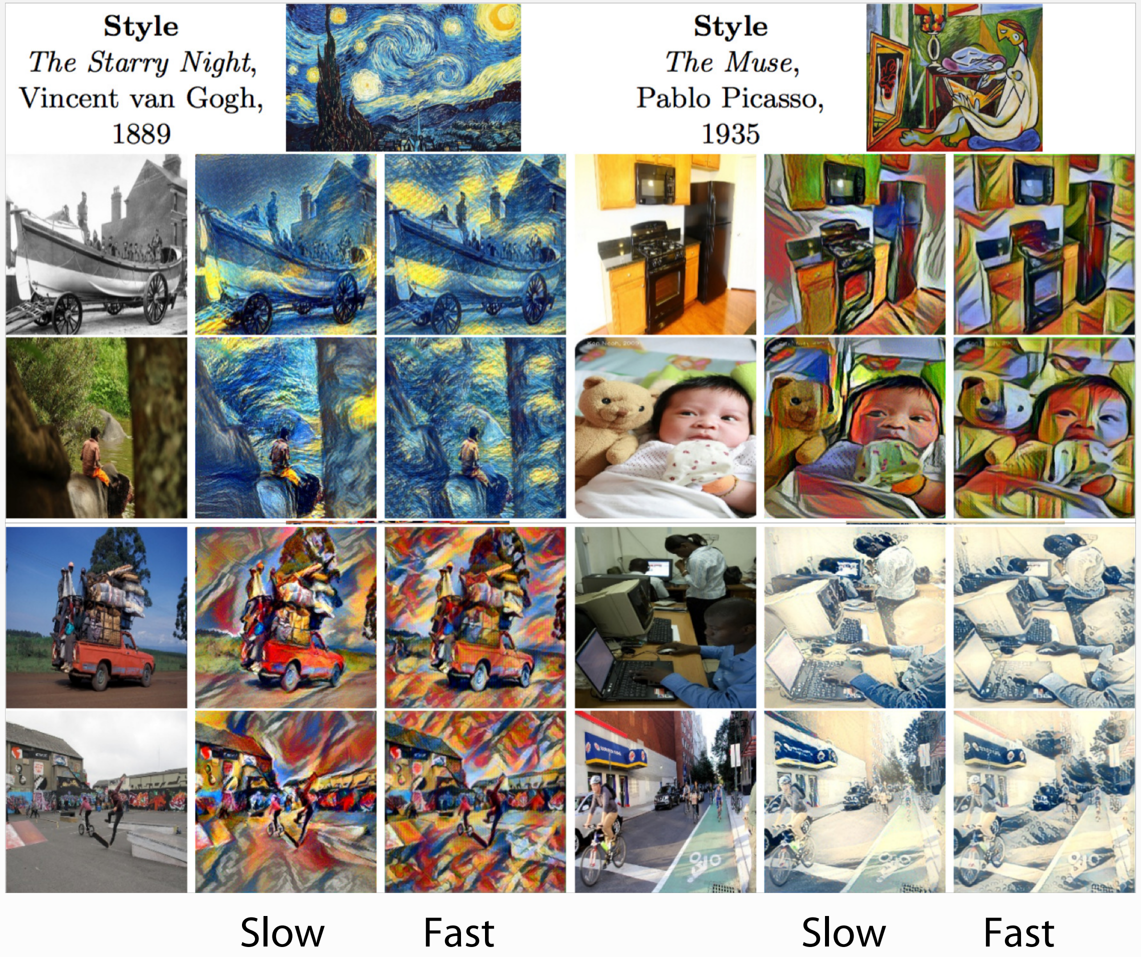
FAST STYLE TRANSFER

(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

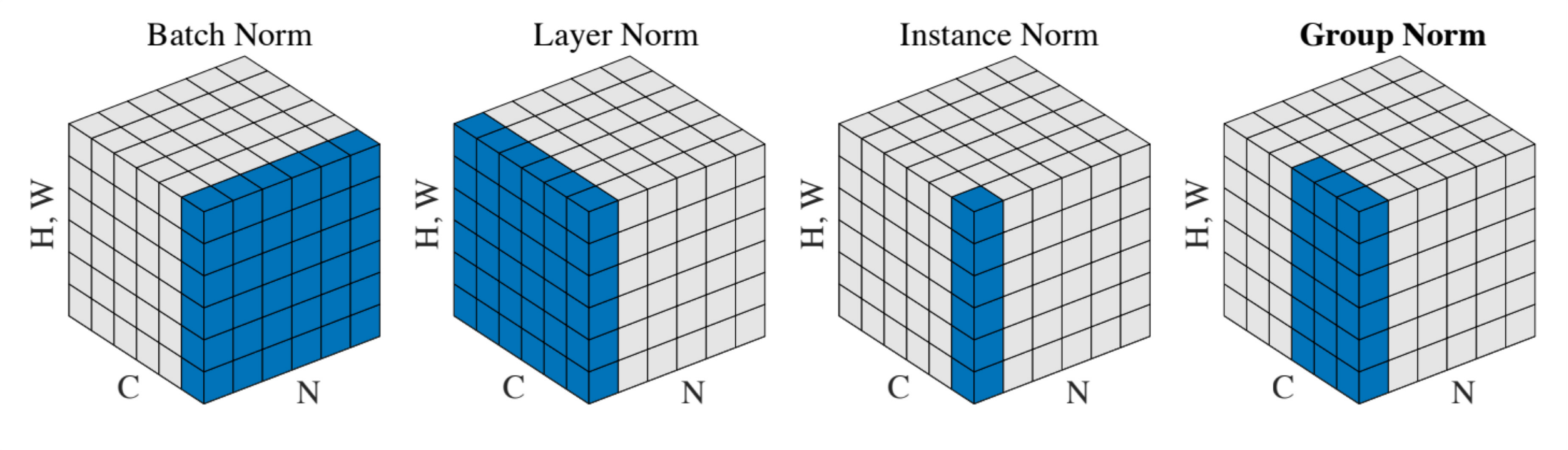
FAST STYLE TRANSFER



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

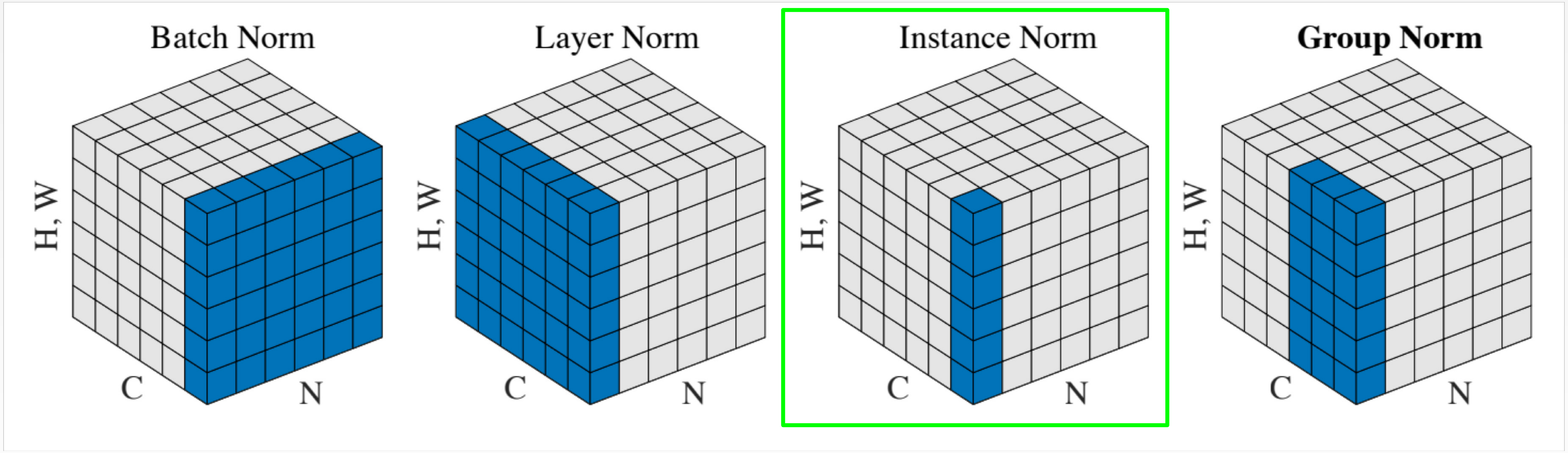
https://github.com/jcjohnson/fast-neural-style

REMEMBER NORMALIZATION METHODS?



REMEMBER NORMALIZATION METHODS?

Instance Normalization was developed for style transfer!

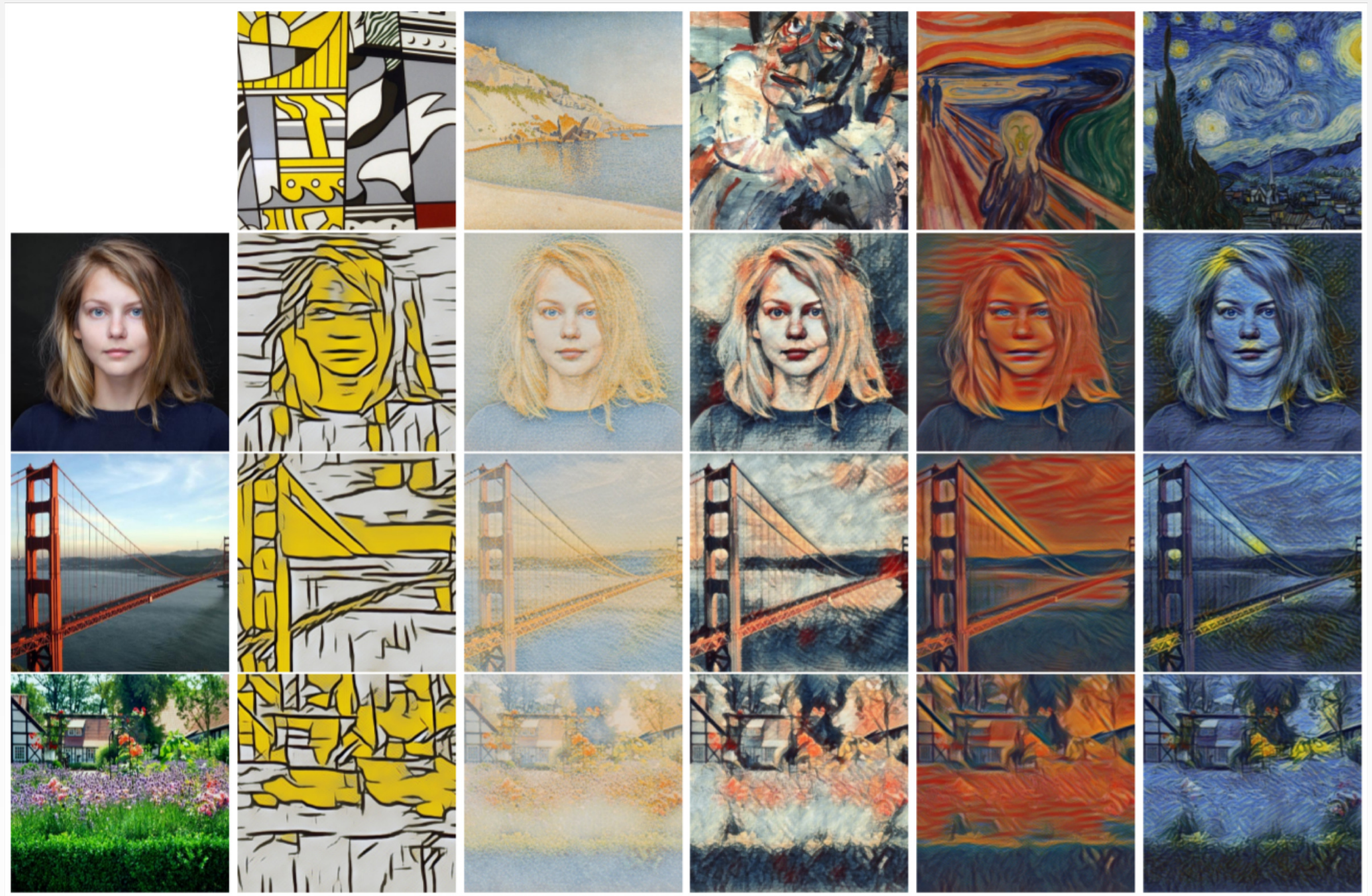


FAST STYLE TRANSFER

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

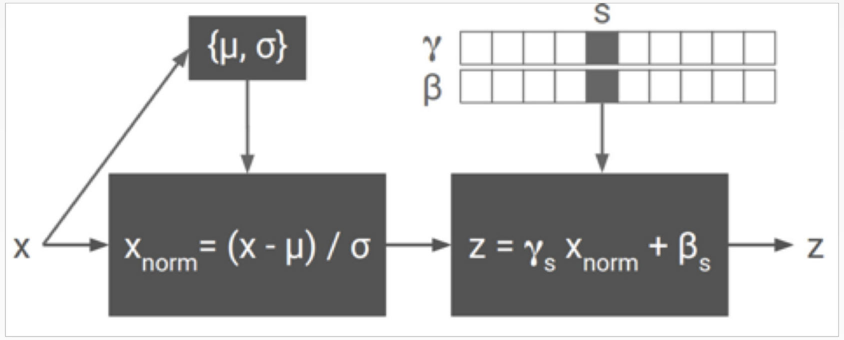
ONE NETWORK, MANY STYLES

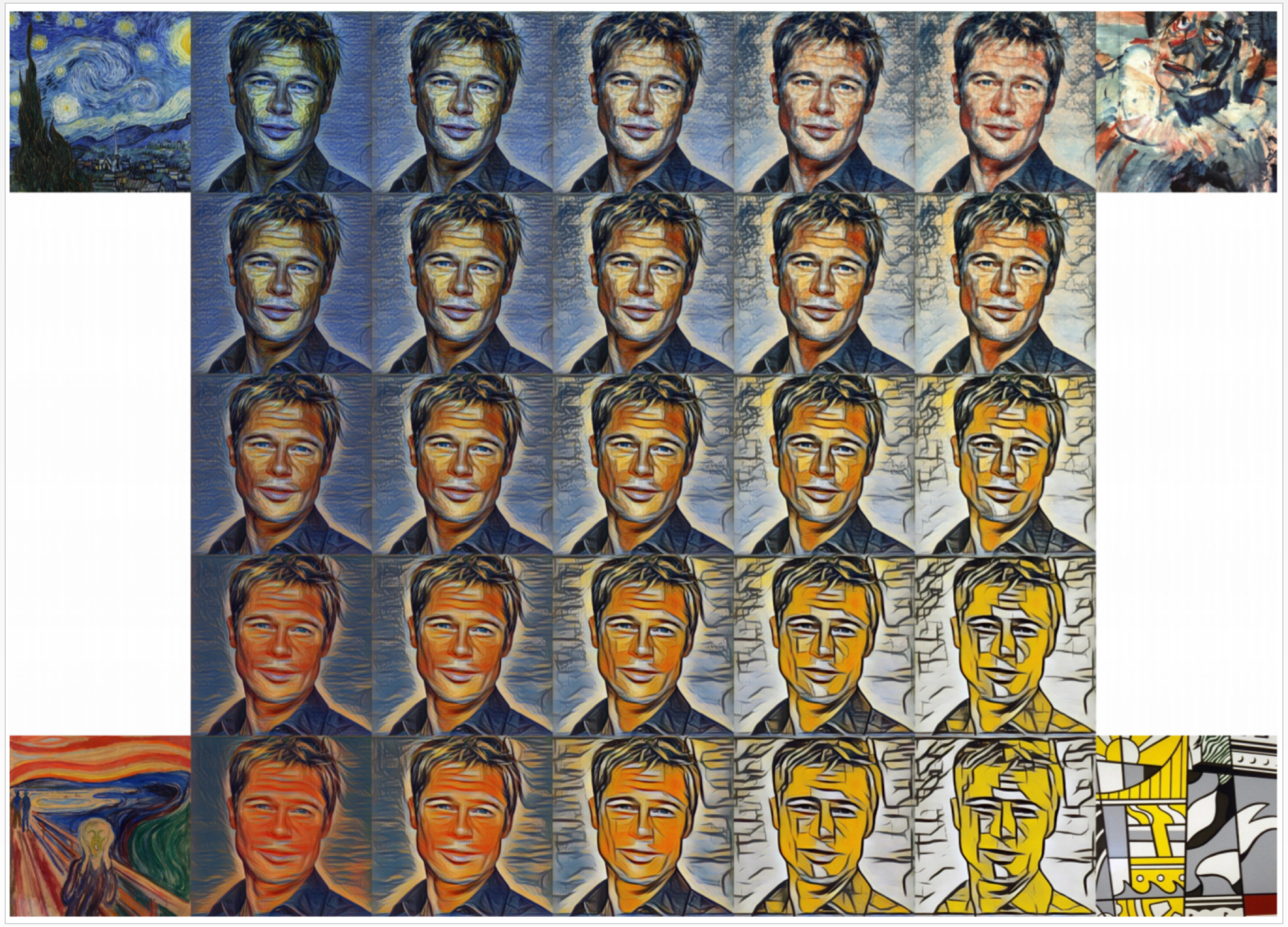


Dumoulin, Shlens, and Kudlur, "A Learned Representation for Artistic Style", ICLR 2017.

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
- **Activation-based Methods:** Nearest neighbors, Dimensionality reduction, maximal patches, occlusion
- **Gradient-based Methods**: Saliency maps, class visualization, fooling images, feature inversion
- **Fun**: DeepDream, Style Transfer.

NEXT TIME: (DEEP) REINFORCEMENT LEARNING