

Lecture 14: Visualizing and Understanding

Reminder: A4

A4 due Wednesday, November 13, 11:59pm

A4 covers:

- PyTorch autograd
- Residual networks
- Recurrent neural networks
- Attention
- Feature visualization
- Style transfer
- Adversarial examples

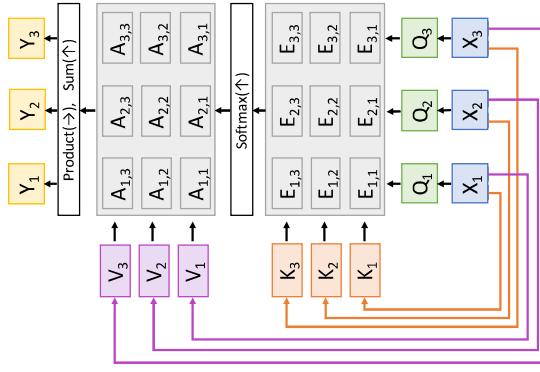
Last Time: Attention

Adding Attention to RNN
models lets them look at
different parts of the
input at each timestep

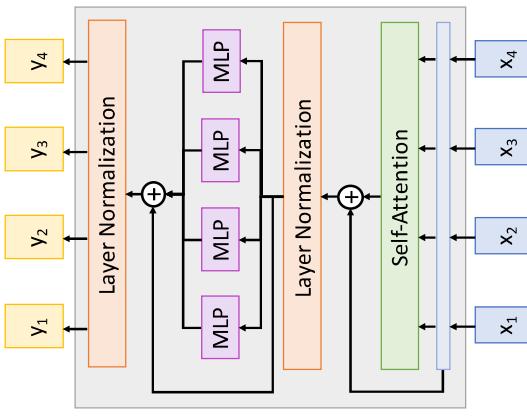


A dog is standing on a hardwood floor.

Generalized Self-Attention
is new, powerful neural
network primitive



Transformers are a new
neural network model
that only uses attention



Last week: Guest Lectures



Monday 10/28
Luowei Zhou
Vision and Language



Wednesday 10/30
Prof. Atul Prakash
Adversarial Machine Learning

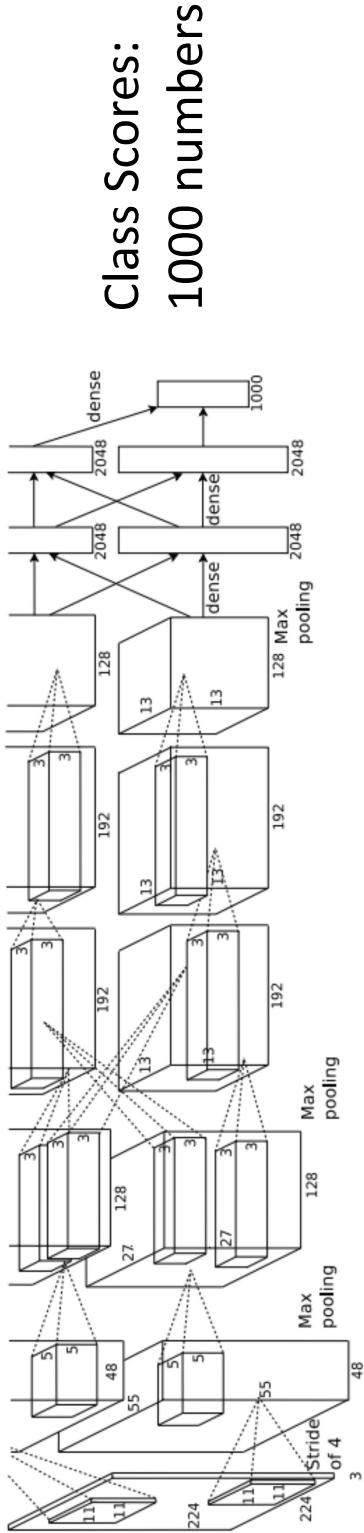
Justin Johnson

Lecture 14 - 4

November 4, 2019

What's going on inside Convolutional Networks?

This image is CC0 public domain



Input Image:
3 x 224 x 224

What are the intermediate features looking for?

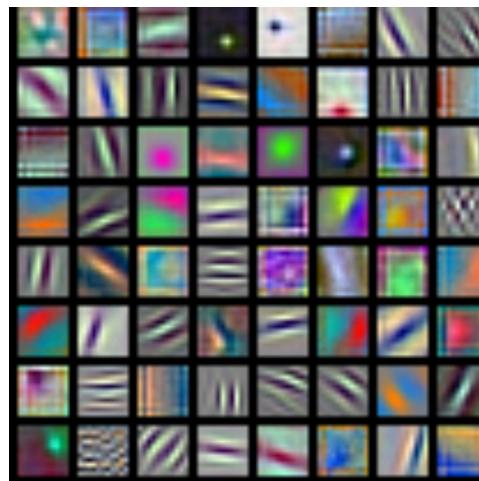
Krizhevsky et al., "ImageNet Classification with Deep Convolutional Neural Networks", NeurIPS 2012.

Justin Johnson

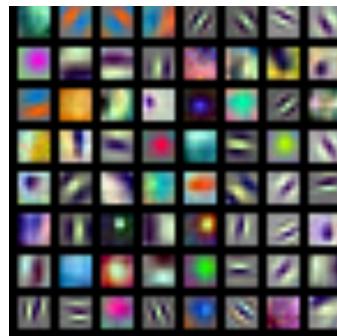
Lecture 14 - 5

November 4, 2019

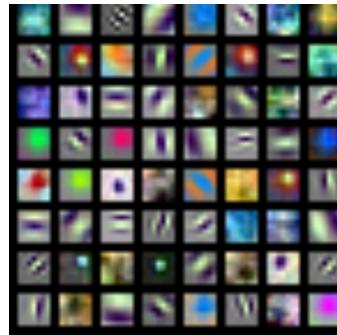
First Layer: Visualize Filters



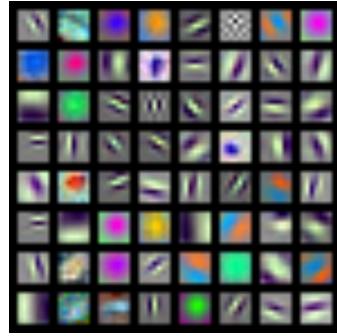
AlexNet:
 $64 \times 3 \times 11 \times 11$



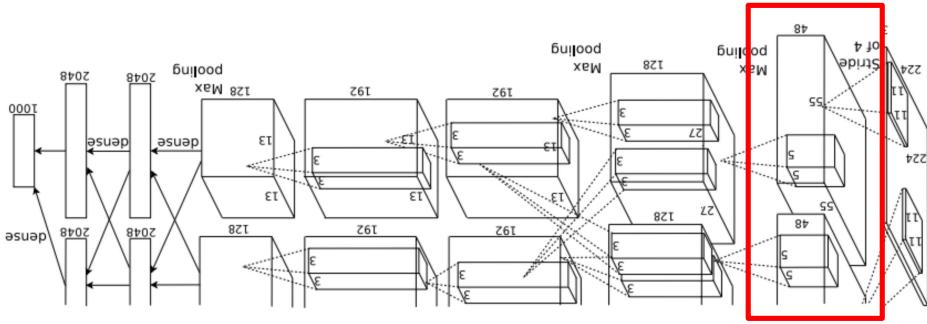
ResNet-18:
 $64 \times 3 \times 7 \times 7$



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



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



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

Higher Layers: Visualize Filters

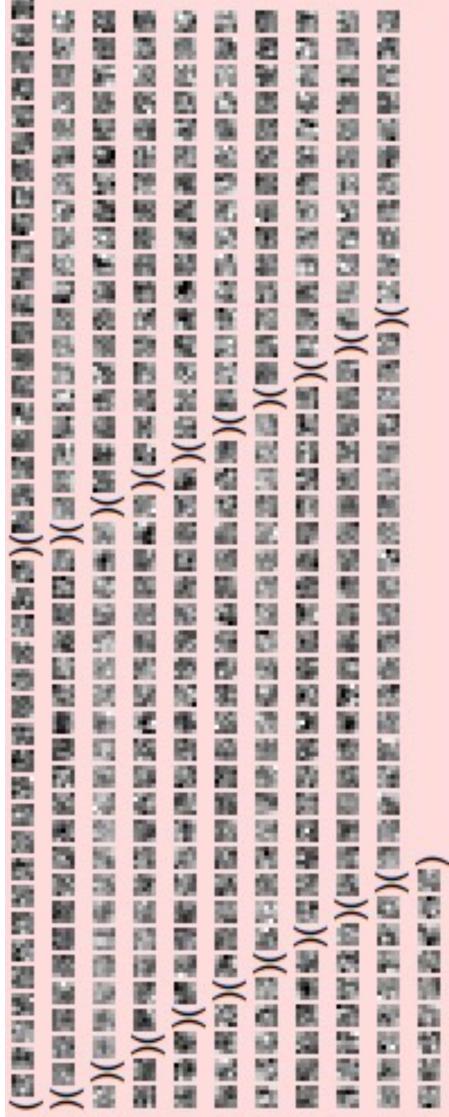


First layer weights: $16 \times 3 \times 7 \times 7$



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

Second layer weights:
 $20 \times 16 \times 7 \times 7$

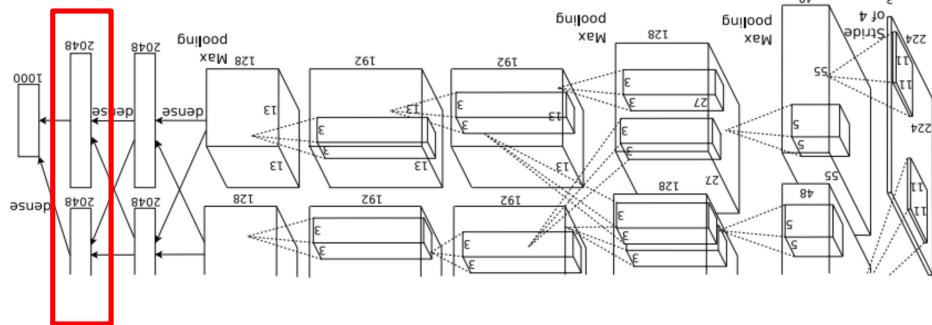


Third layer weights:
 $20 \times 20 \times 7 \times 7$

Source: ConvNetJS
CIFAR-10 example
<https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>

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

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

Justin Johnson

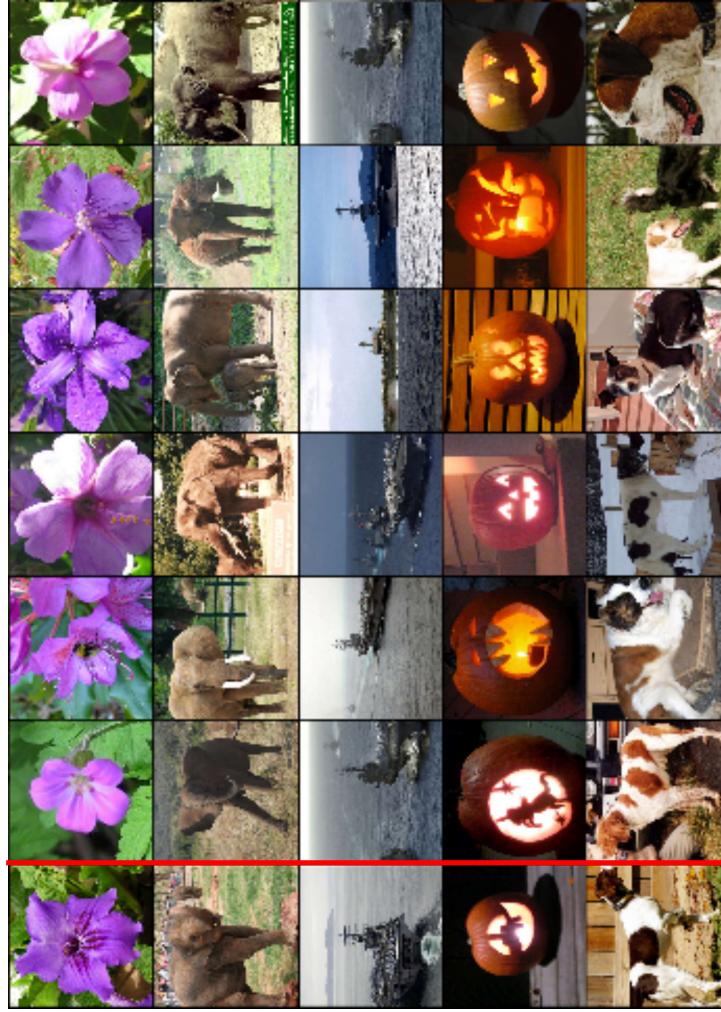
Lecture 14 - 8

November 4, 2019

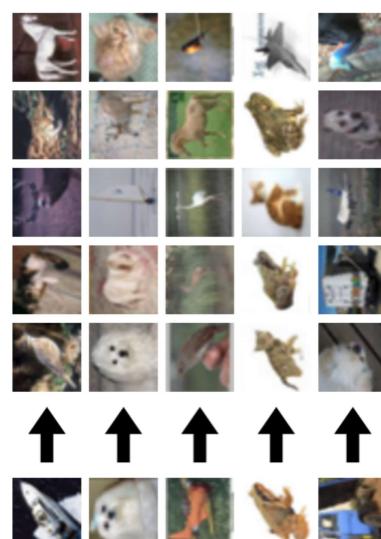
Last Layer: Nearest Neighbors

Test

image L2 Nearest neighbors in feature space



Recall: Nearest
neighbors in pixel space

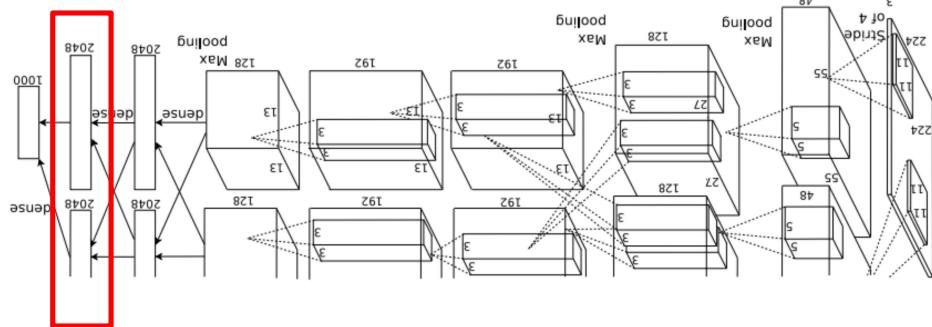


Krizhevsky et al., "ImageNet Classification with Deep Convolutional Neural Networks", NeurIPS 2012.
Figures reproduced with permission.

Justin Johnson

Lecture 14 - 9

November 4, 2019

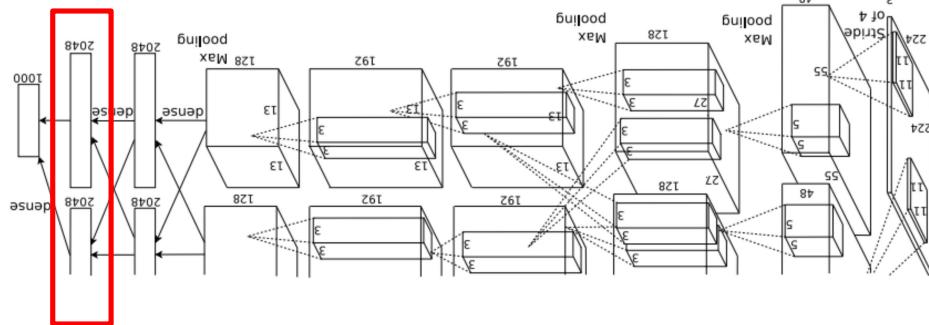
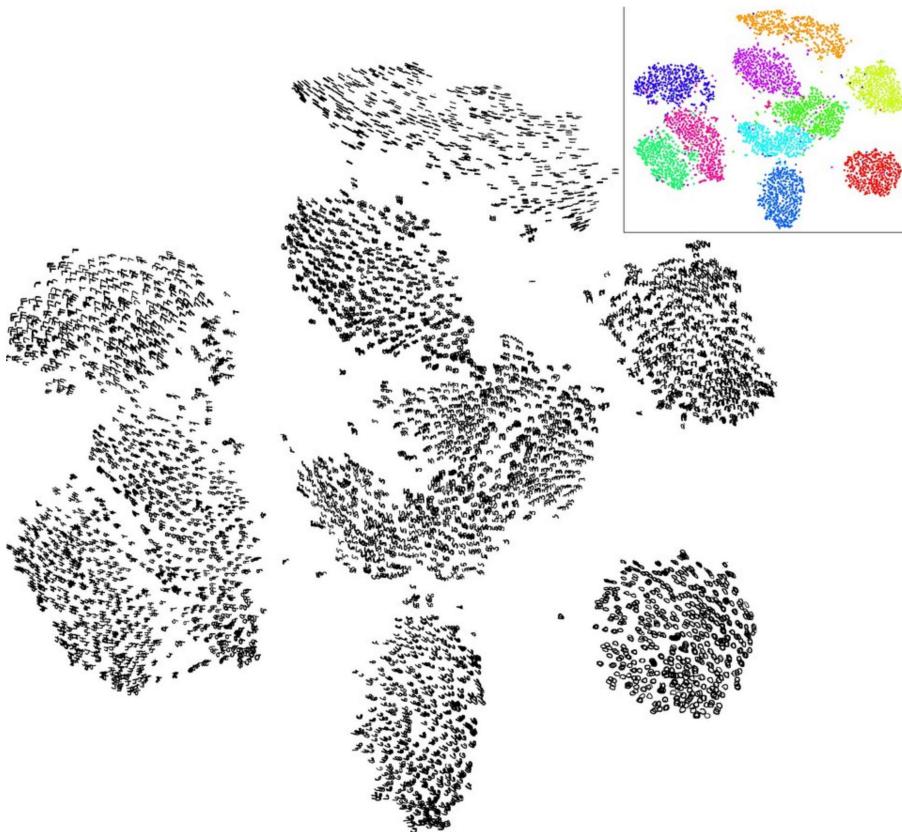


Last Layer: Dimensionality Reduction

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

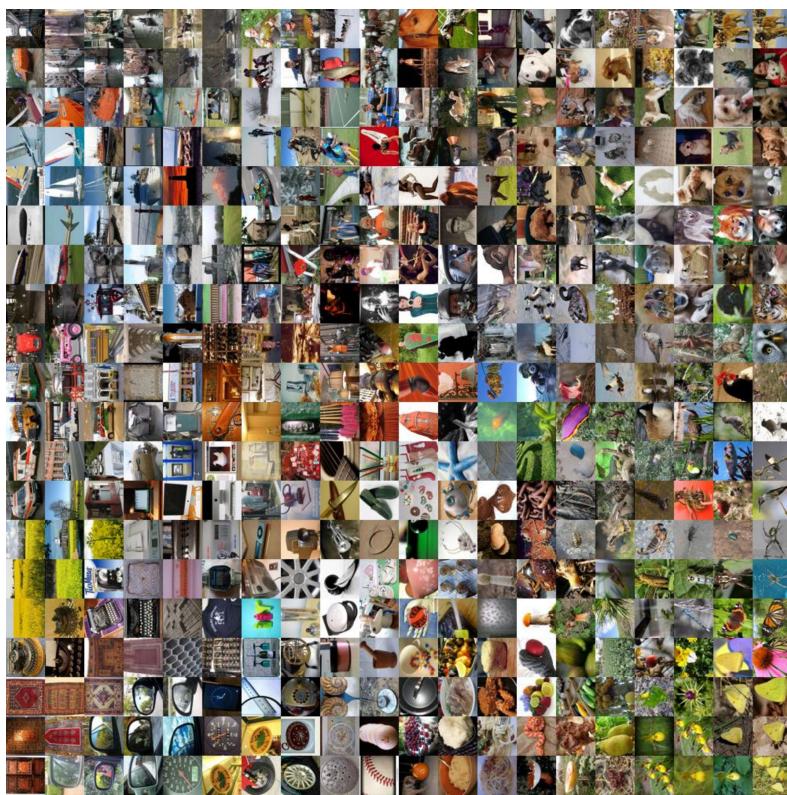
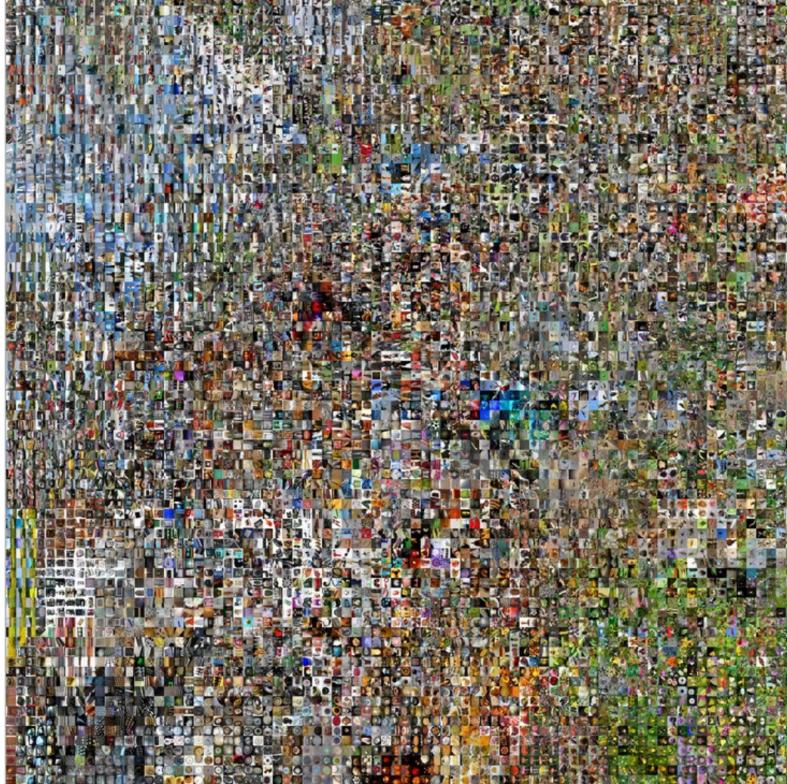
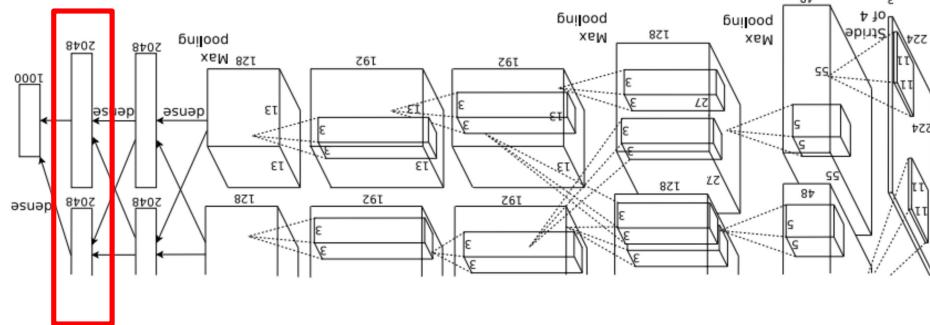
Simple algorithm: Principal
Component Analysis (PCA)

More complex: t-SNE



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

Last Layer: Dimensionality Reduction



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

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

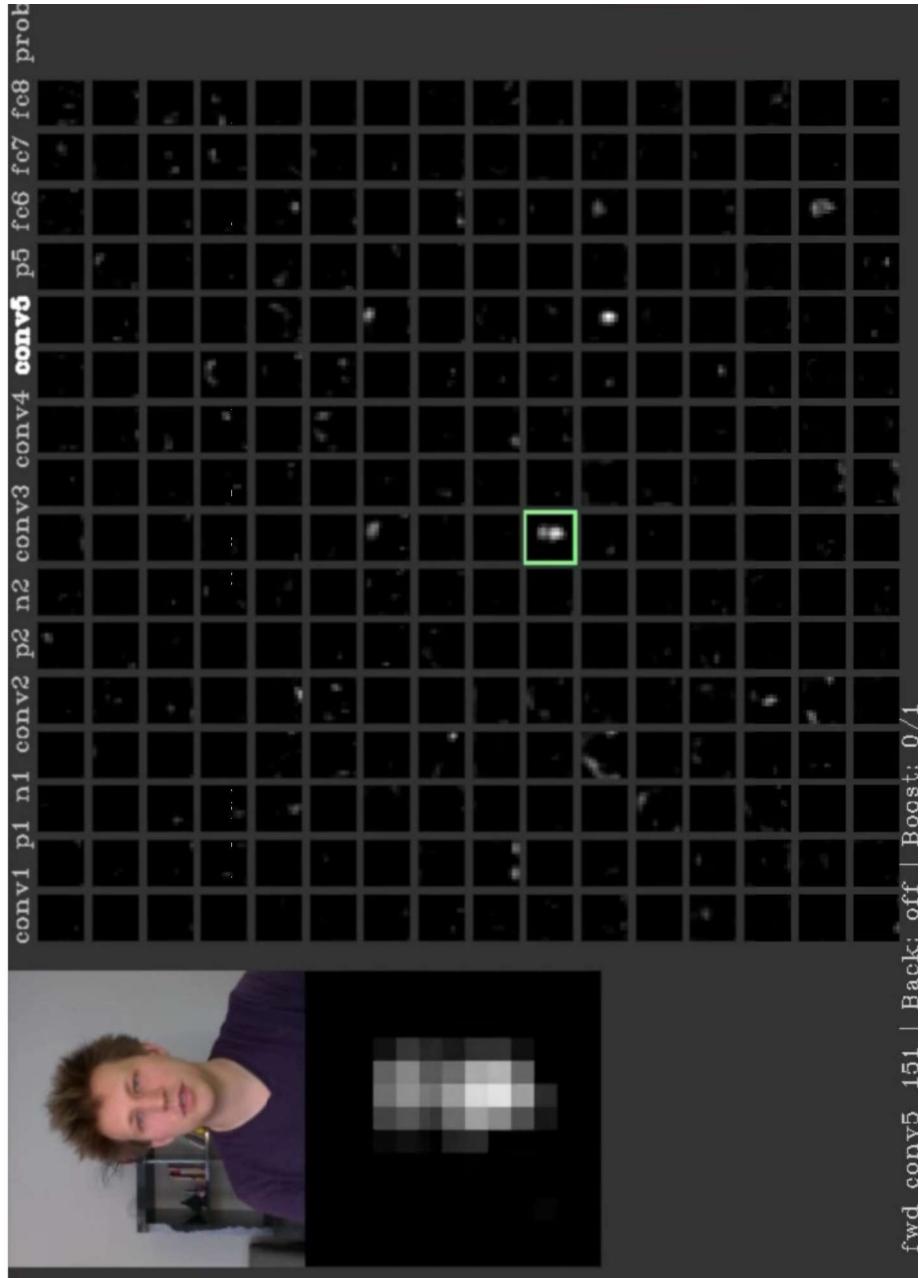
Justin Johnson

Lecture 14 - 11

November 4, 2019

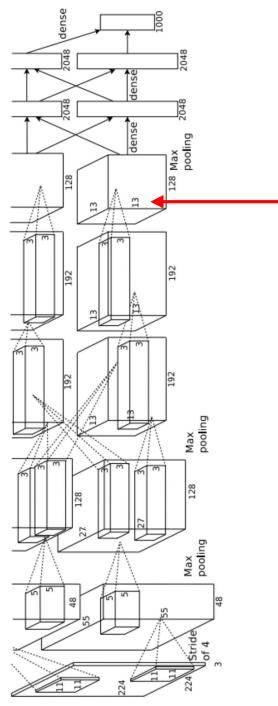
Visualizing Activations

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



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

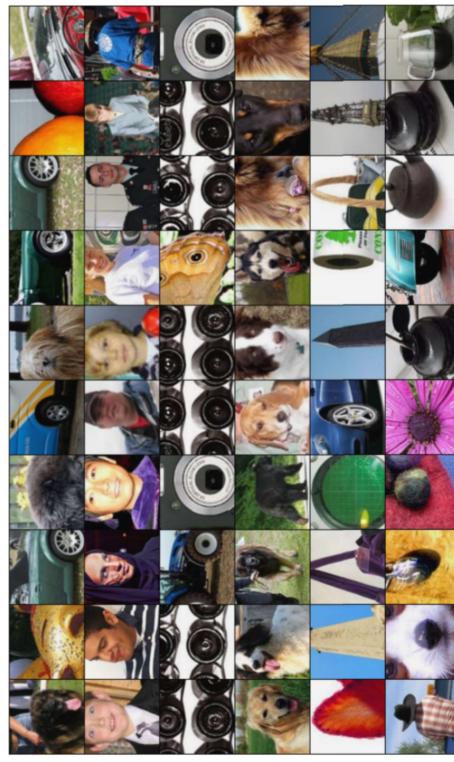
Maximally Activating Patches



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

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

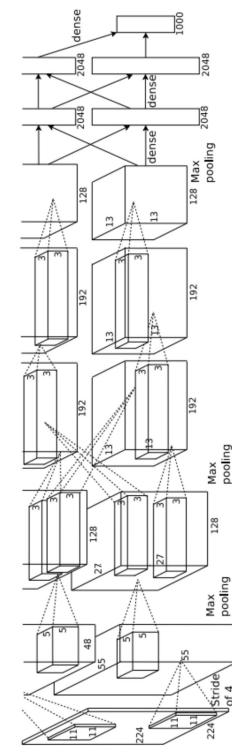
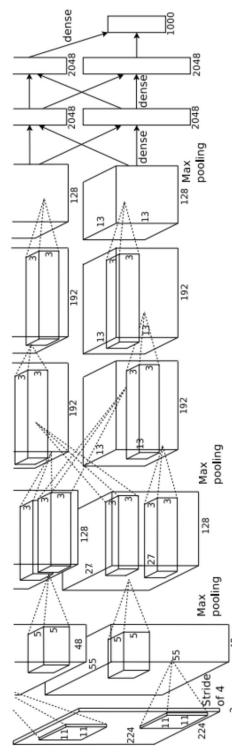
Visualize image patches that correspond to
maximal activations



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.

Which Pixels Matter? Saliency via Occlusion

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



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

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

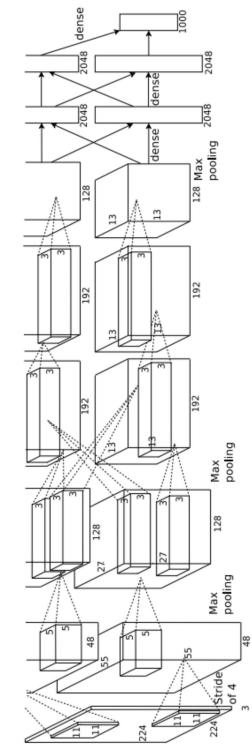
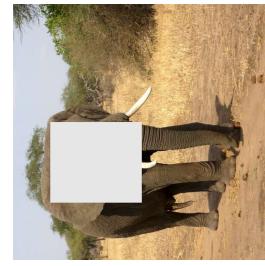
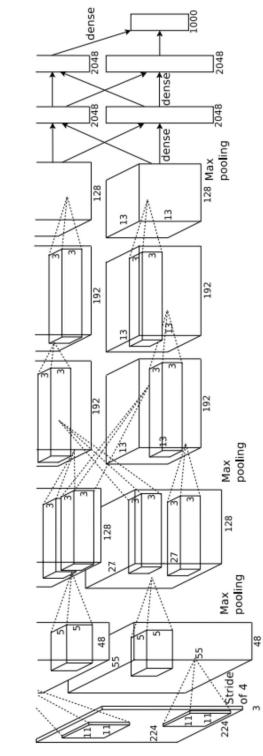
Justin Johnson

Lecture 14 - 14

November 4, 2019

Which Pixels Matter? Saliency via Occlusion

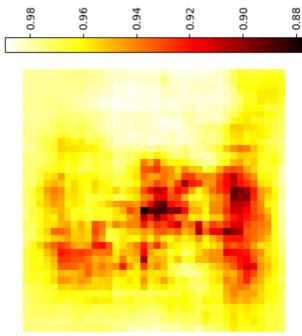
Mask part of the image before feeding to CNN,
check how much predicted probabilities change



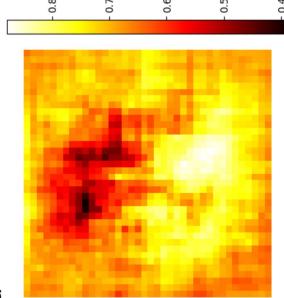
Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

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

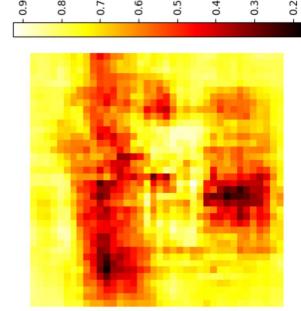
schooner



African elephant, Loxodonta africana



go-kart



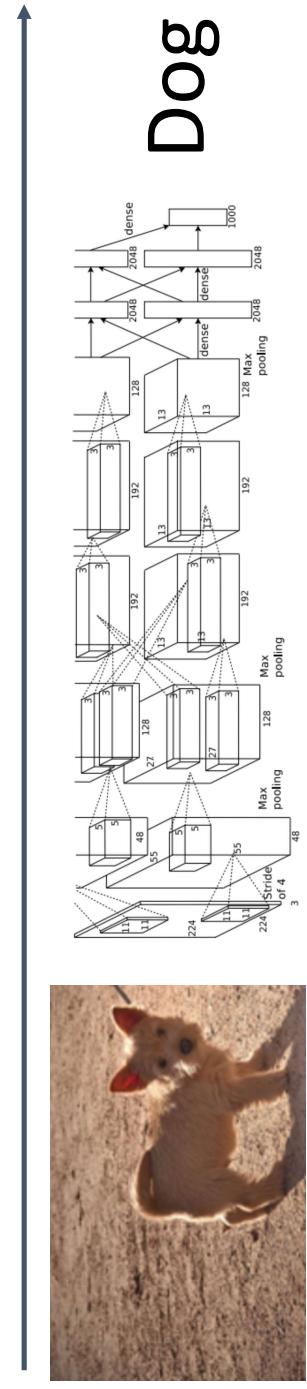
Justin Johnson

Lecture 14 - 15

November 4, 2019

Which pixels matter? Saliency via Backprop

Forward pass: Compute probabilities



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

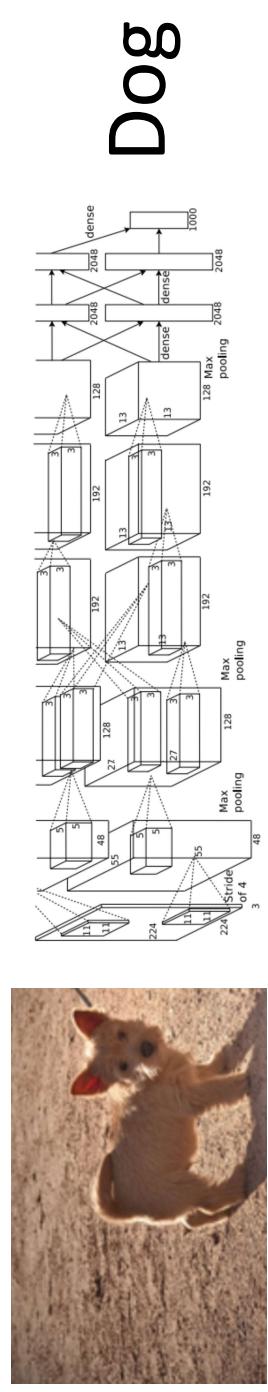
Justin Johnson

Lecture 14 - 16

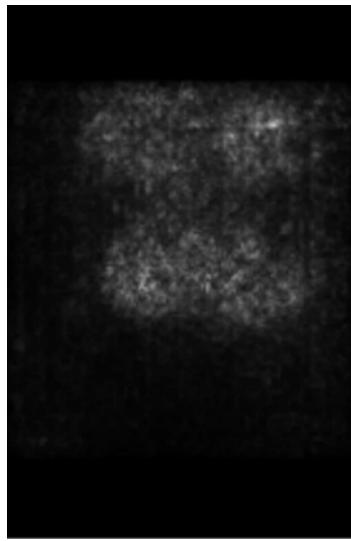
November 4, 2019

Which pixels matter? Saliency via Backprop

Forward pass: Compute probabilities



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



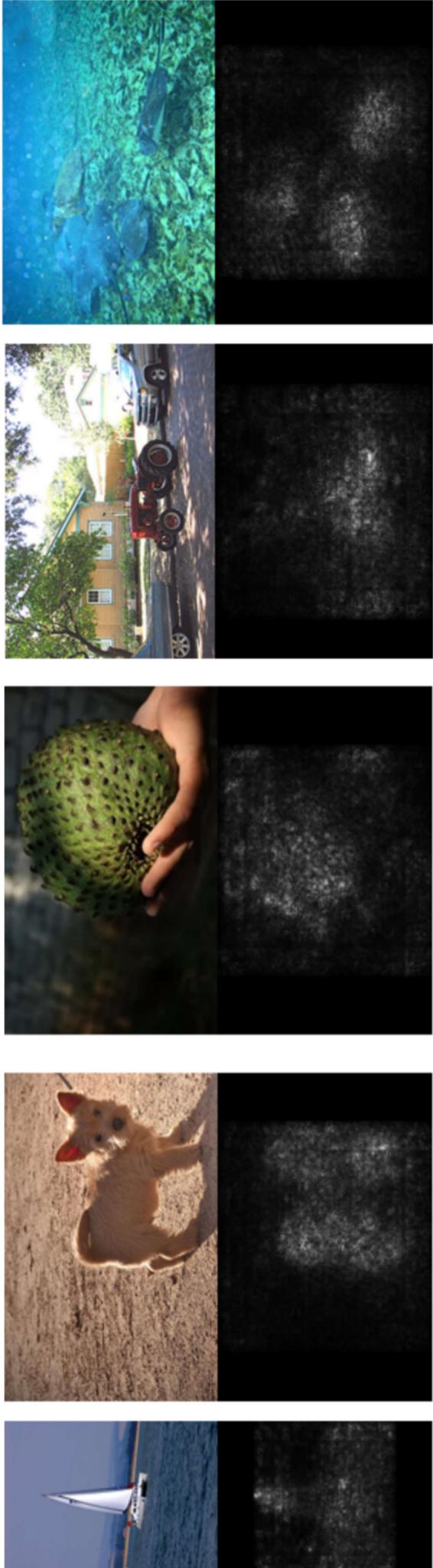
Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.
Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.

Justin Johnson

Lecture 14 - 17

November 4, 2019

Which pixels matter? Saliency via Backprop



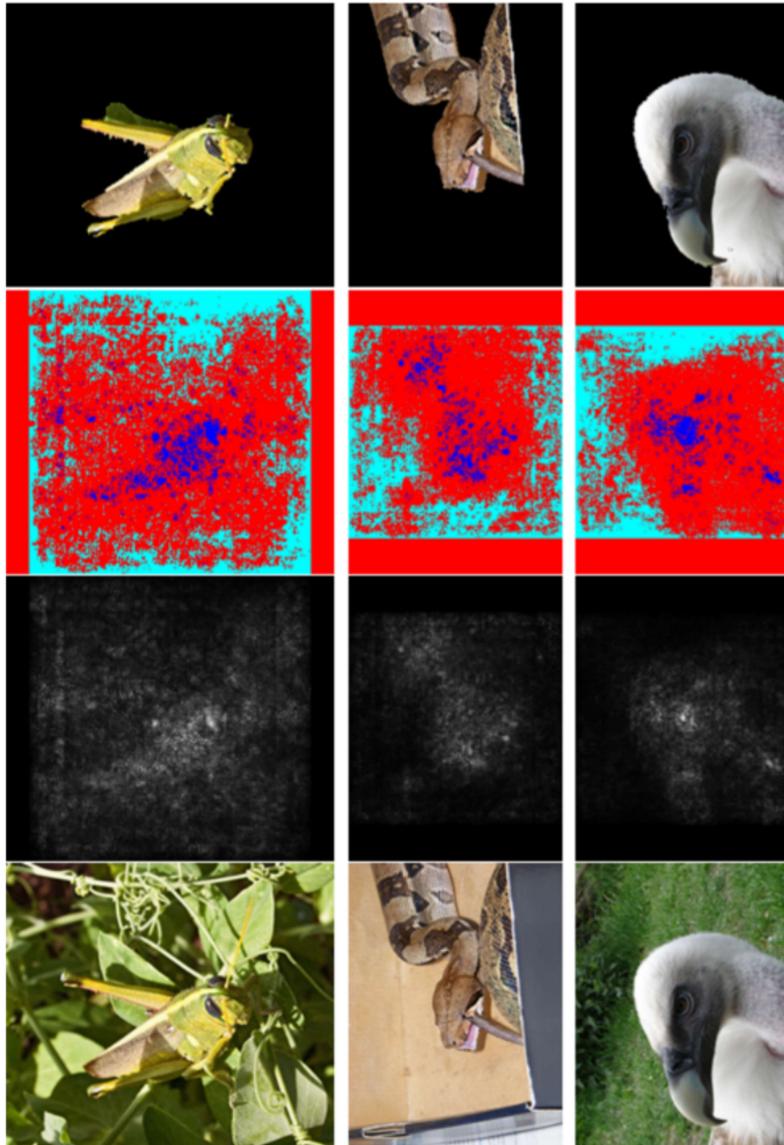
Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.
Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.

Justin Johnson

Lecture 14 - 18

November 4, 2019

Saliency Maps: Segmentation without Supervision



Use GrabCut on
saliency map

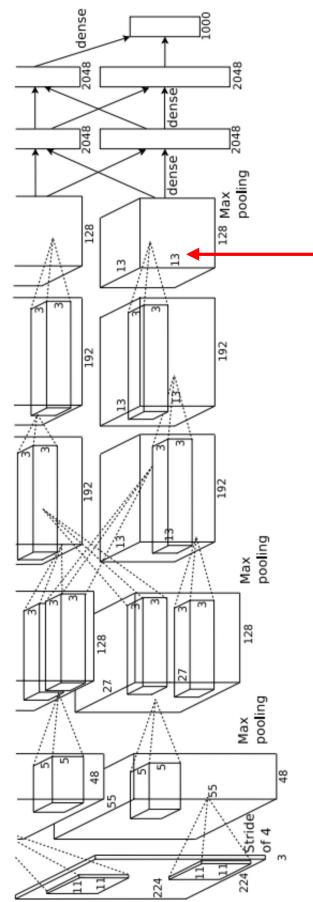
Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.
Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.
Rother et al, 'Grabcut: Interactive foreground extraction using iterated graph cuts', ACM TOG 2004

Justin Johnson

Lecture 14 - 19

November 4, 2019

Intermediate Features via (guided) backprop

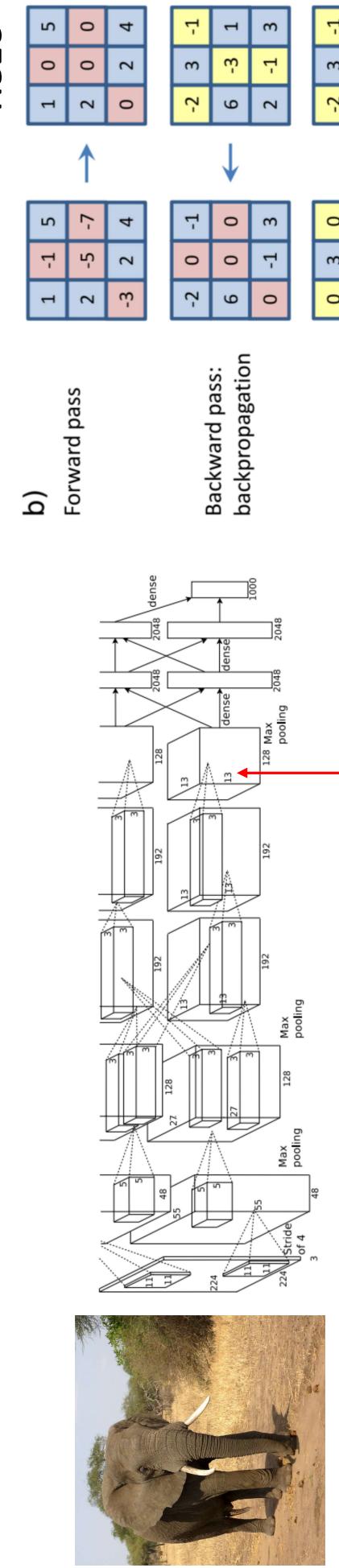


Pick a single intermediate neuron, e.g.
one value in $128 \times 13 \times 13$ conv5
feature map

Compute gradient of neuron value with
respect to image pixels

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

Intermediate Features via (guided) backprop



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

Compute gradient of neuron value with
respect to image pixels

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

Intermediate Features via (guided) backprop



Maximally activating patches
(Each row is a different neuron)

Guided Backprop



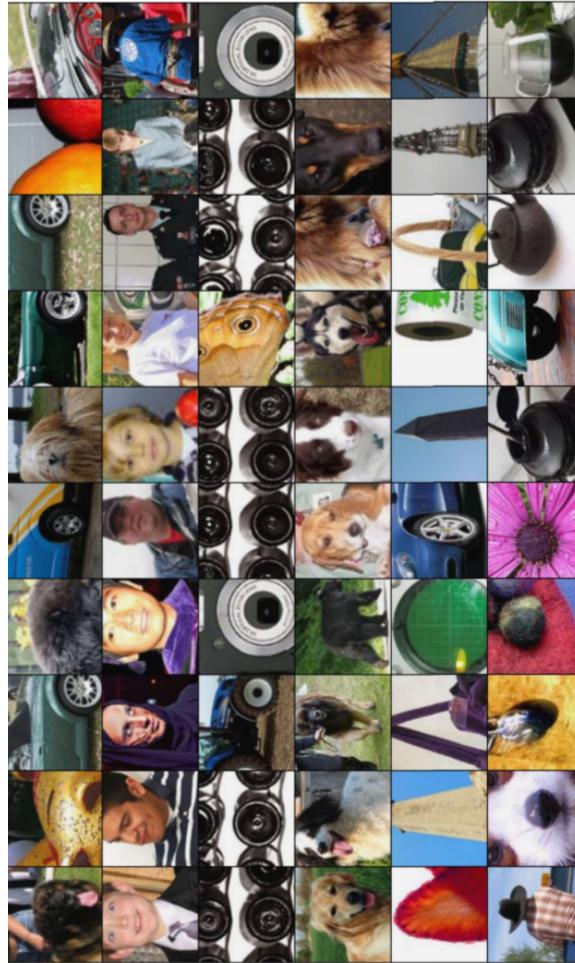
Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014
Springenberg et al, 'Striving for Simplicity: The All Convolutional Net', ICLR Workshop 2015
Figure copyright Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, 2015; reproduced with permission.

Justin Johnson

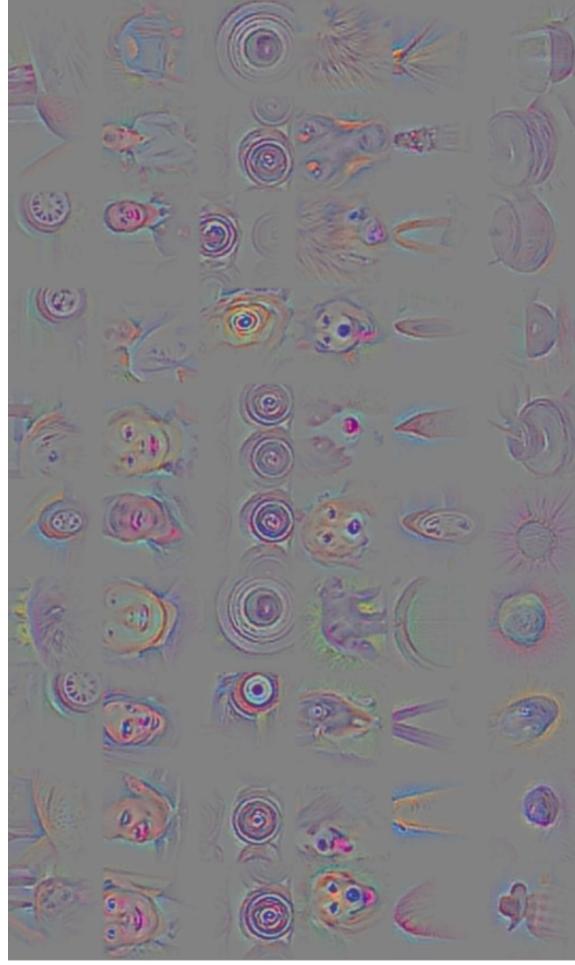
Lecture 14 - 22

November 4, 2019

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

Visualizing CNN Features: Gradient Ascent

(Guided) backprop:

Find the part of an image that a neuron responds to

Gradient ascent:

Generate a synthetic image that maximally activates a neuron

$$l^* = \arg \max_l f(l) + R(l)$$

Neuron value

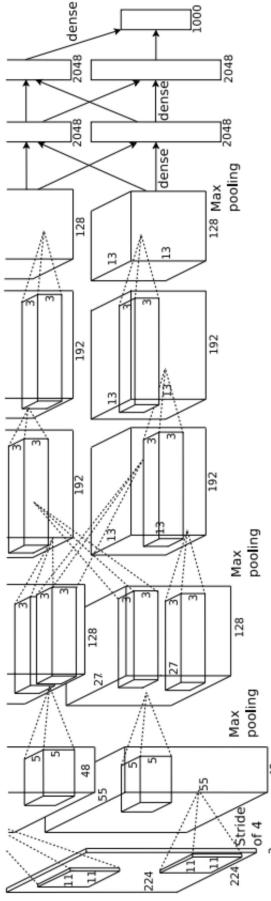
Natural image regularizer

Visualizing CNN Features: Gradient Ascent

1. Initialize image to zeros

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

score for class c (before Softmax)



Repeat:

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

Visualizing CNN Features: Gradient Ascent

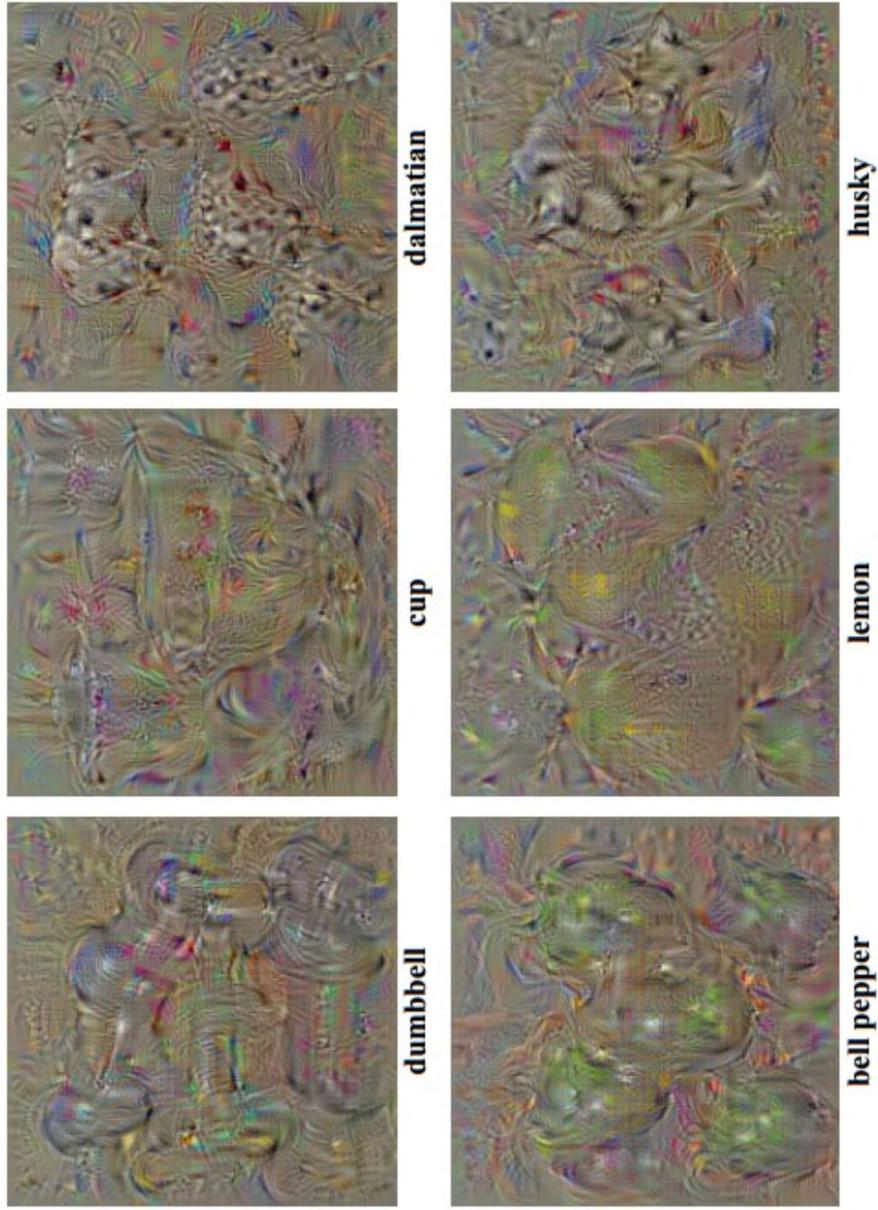
$$\arg \max_I S_c(I) - \lambda \|I\|_2^2$$

Simple regularizer: Penalize
L2 norm of generated image

Visualizing CNN Features: Gradient Ascent

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

Simple regularizer: Penalize
L2 norm of generated image

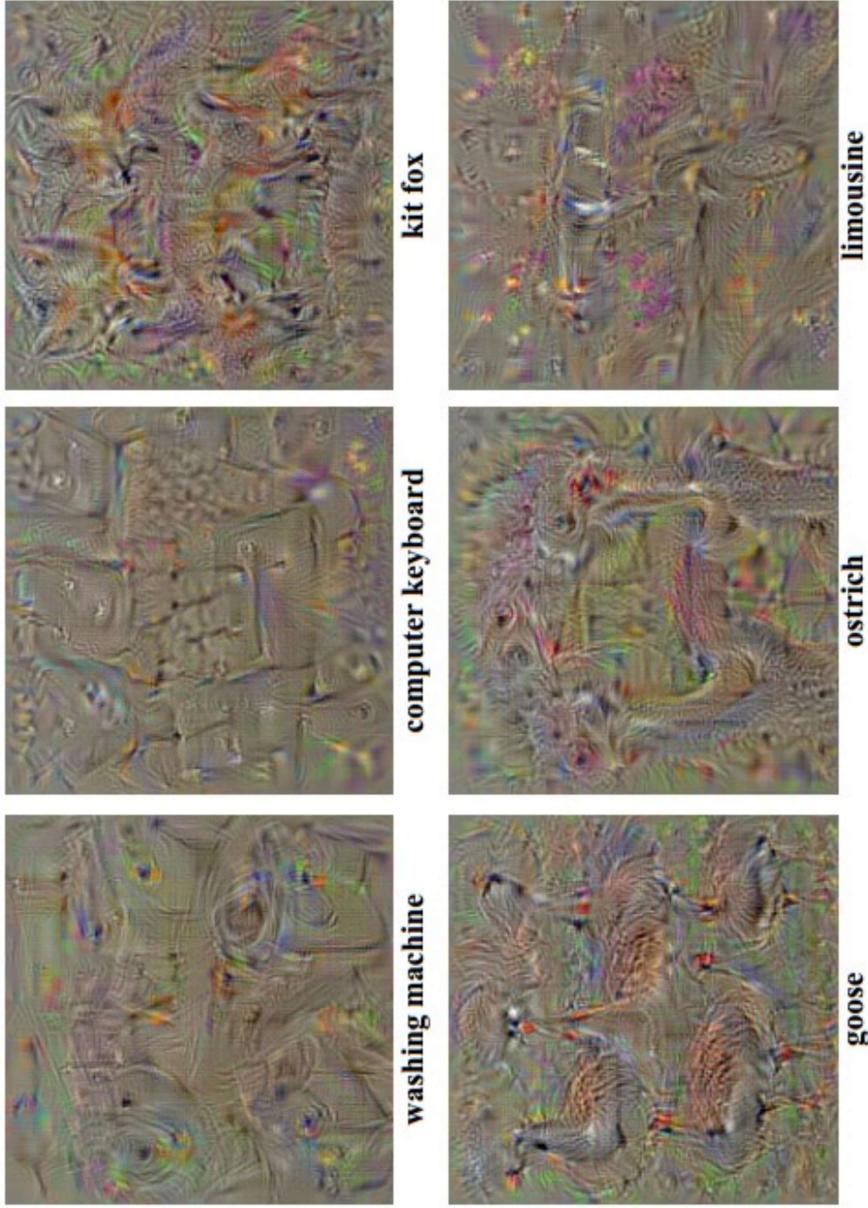


Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.
Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014, reproduced with permission.

Visualizing CNN Features: Gradient Ascent

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

Simple regularizer: Penalize
L2 norm of generated image



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

Visualizing CNN Features: Gradient Ascent

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

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

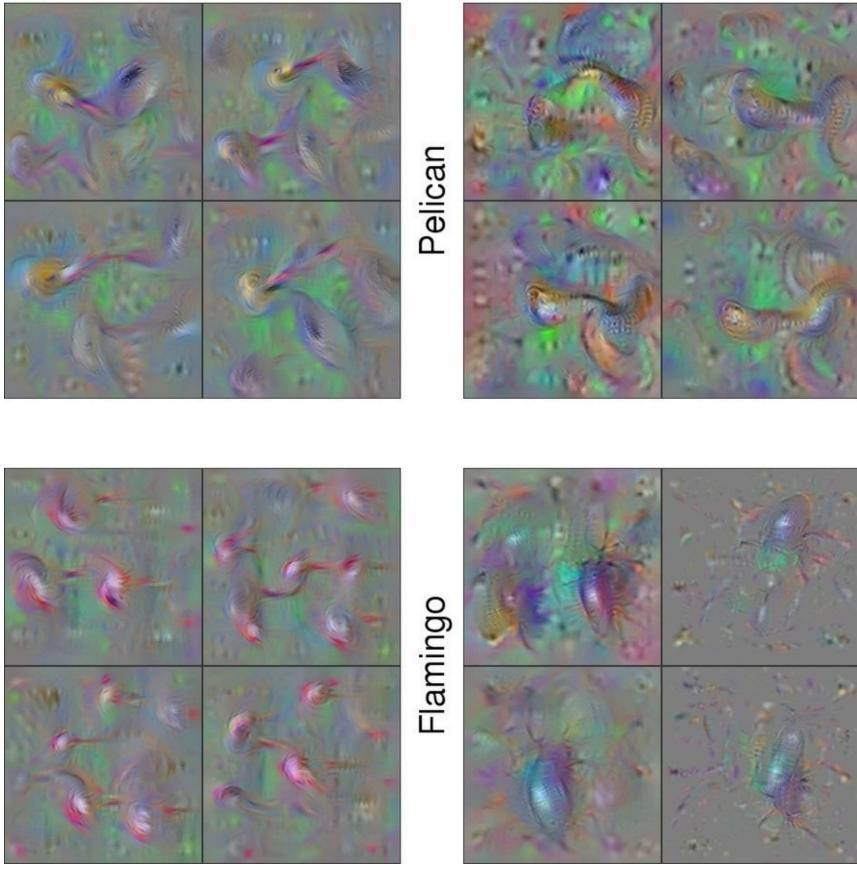
1. Gaussian blur image
2. Clip pixels with small values to 0
3. Clip pixels with small gradients to 0

Visualizing CNN Features: Gradient Ascent

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

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

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



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

Justin Johnson

Lecture 14 - 30

November 4, 2019

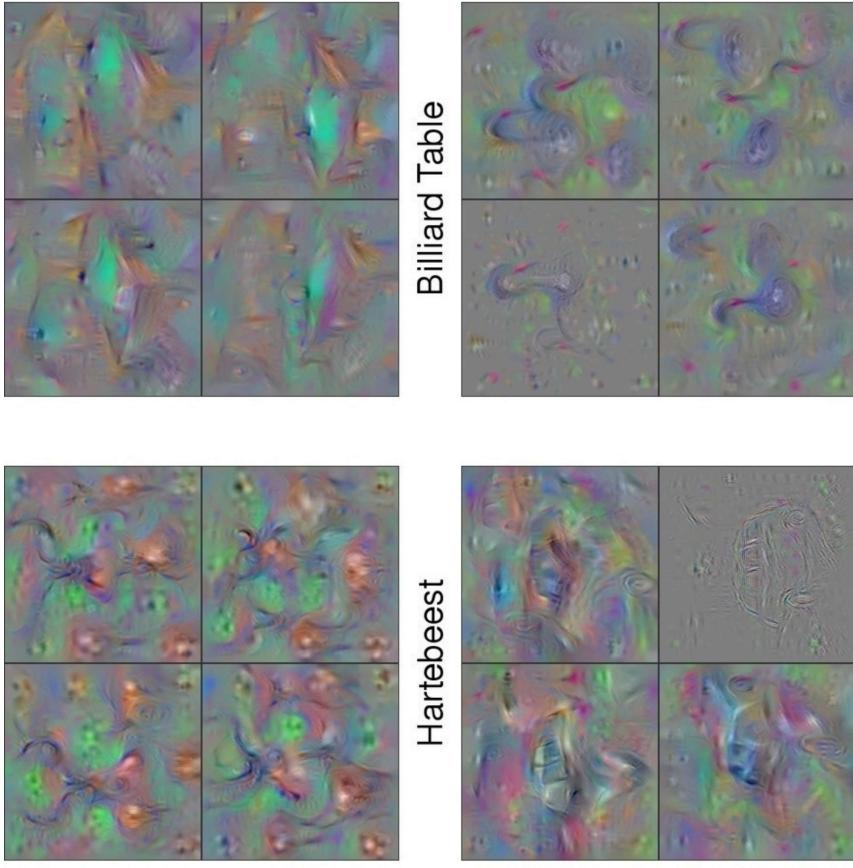
Indian Cobra

Visualizing CNN Features: Gradient Ascent

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

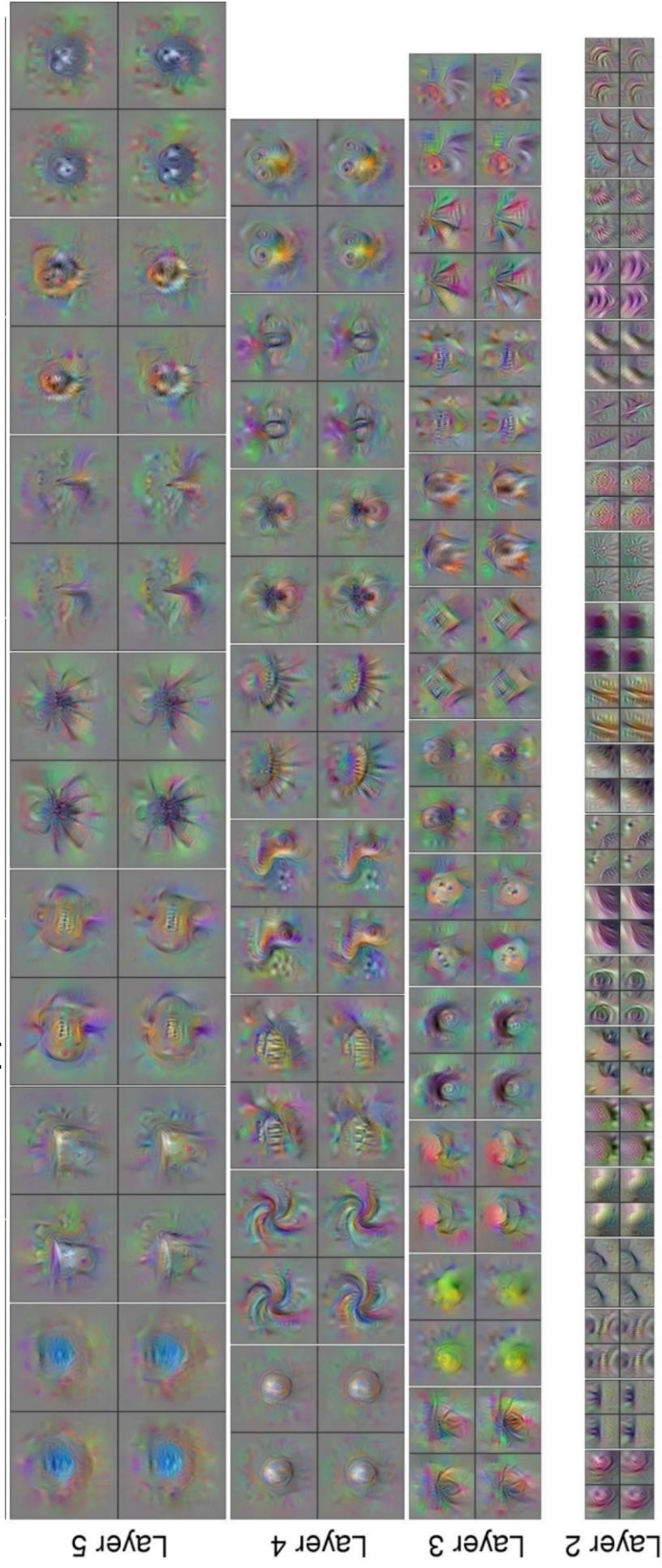
Better regularizer: Penalize L2 norm of image; also during optimization periodically

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



Visualizing CNN Features: Gradient Ascent

Use the same approach to visualize intermediate features



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

Justin Johnson

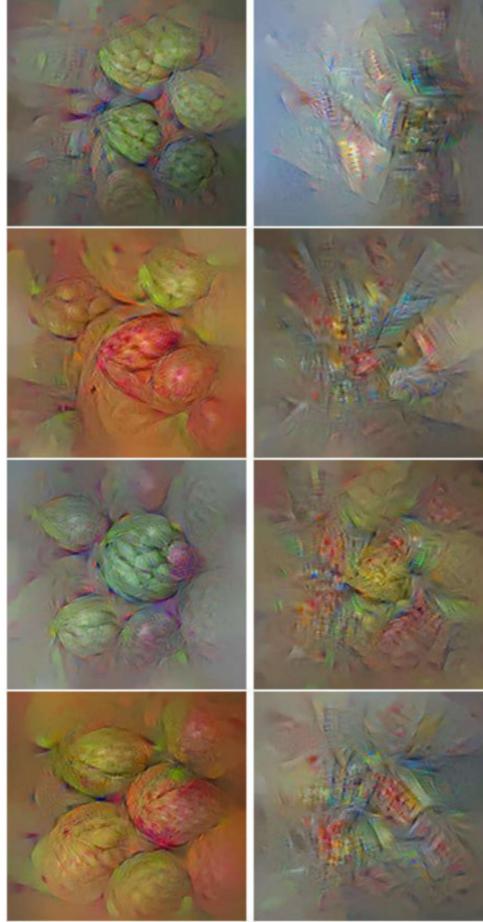
Lecture 14 - 32

November 4, 2019

Visualizing CNN Features: Gradient Ascent

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

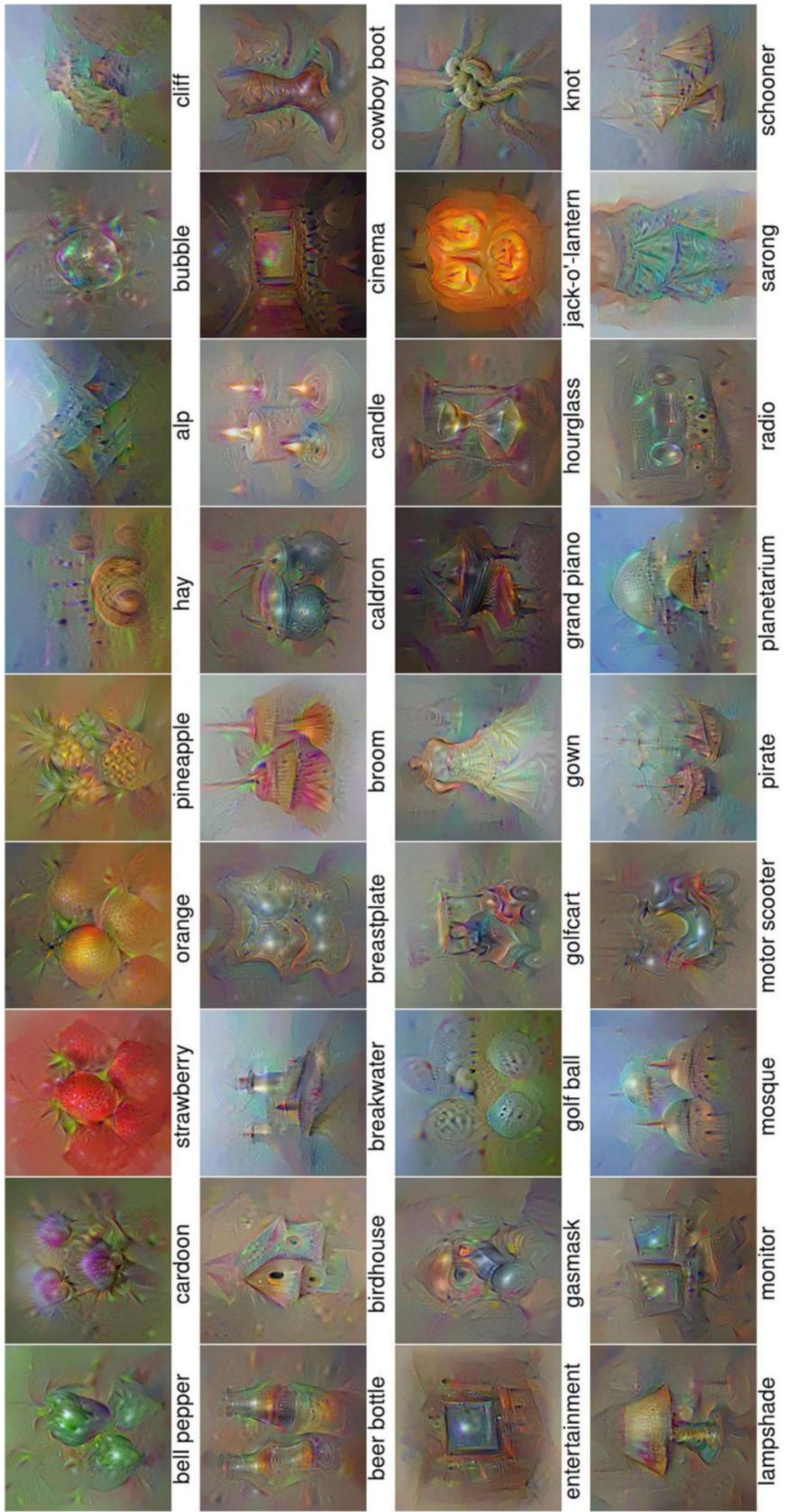
Reconstructions of multiple feature types (facets) recognized by the same “grocery store” neuron



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



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

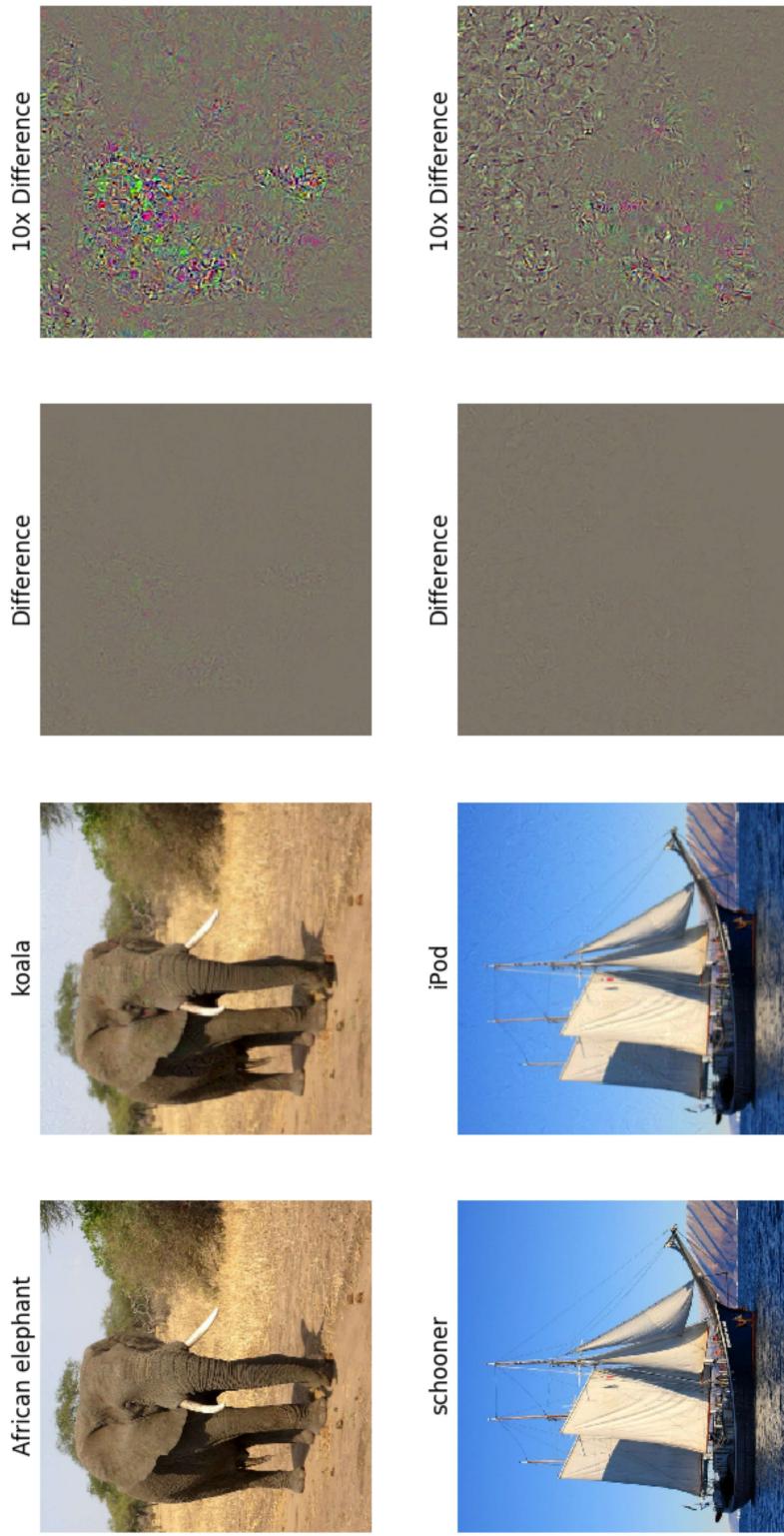


Nguyen et al., "Synthesizing the preferred inputs for neurons in neural networks via deep generator networks," NIPS 2016

Adversarial Examples

1. Start from an arbitrary image
2. Pick an arbitrary category
3. Modify the image (via gradient ascent)
to maximize the class score
4. Stop when the network is fooled

Adversarial Examples



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

Justin Johnson

Lecture 14 - 37

November 4, 2019

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

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

Features of new image

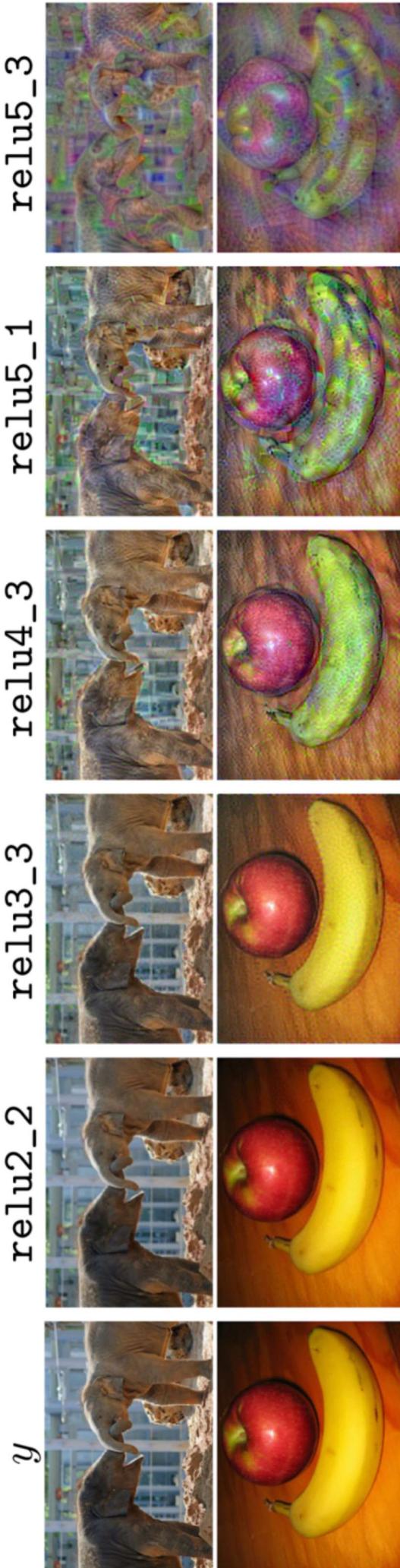
$$\mathcal{R}_{V^\beta}(\mathbf{x}) = \sum_{i,j} \left((x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2 \right)^{\frac{\beta}{2}}$$

Total Variation regularizer (encourages spatial smoothness)

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

Feature Inversion

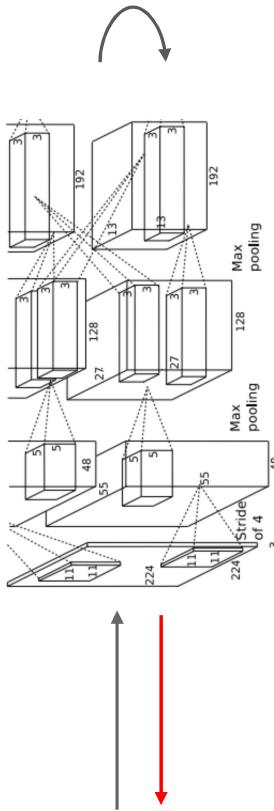
Reconstructing from different layers of VGG-16



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.

DeepDream: Amplify Existing Features

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



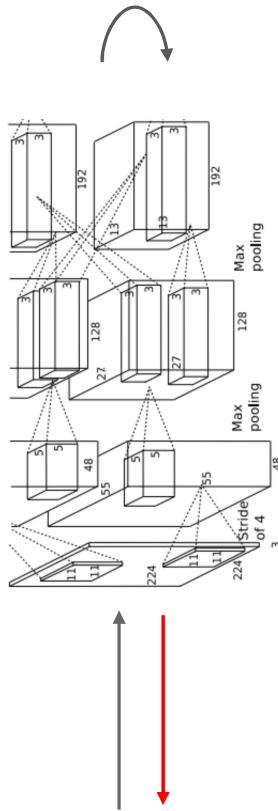
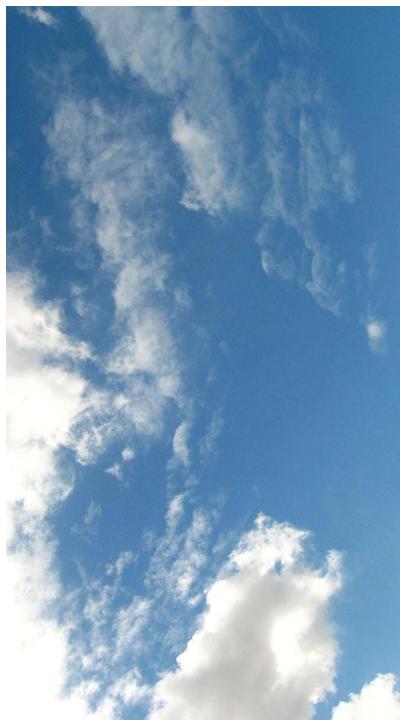
Choose an image and a layer in a CNN; repeat:

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

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

DeepDream: Amplify Existing Features

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



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

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

Equivalent to:

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

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

DeepDream: Amplify Existing Features

Code is very simple but it uses a couple tricks:

(Code is licensed under Apache 2.0)

```
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)

q = src.diff[0]
# apply normalized ascent step to the input image
src.data[:] += step_size/np.abs(q).mean() * q

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)
```

Forward / backward

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

DeepDream: Amplify Existing Features

Code is very simple but it uses a couple tricks:

(Code is licensed under Apache 2.0)

```
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)

q = src.diff[0]
# apply normalized ascent step to the input image
src.data[:] += step_size/np.abs(q).mean() * q

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)
```



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

DeepDream: Amplify Existing Features

Code is very simple but it uses a couple tricks:

(Code is licensed under Apache 2.0)

```
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

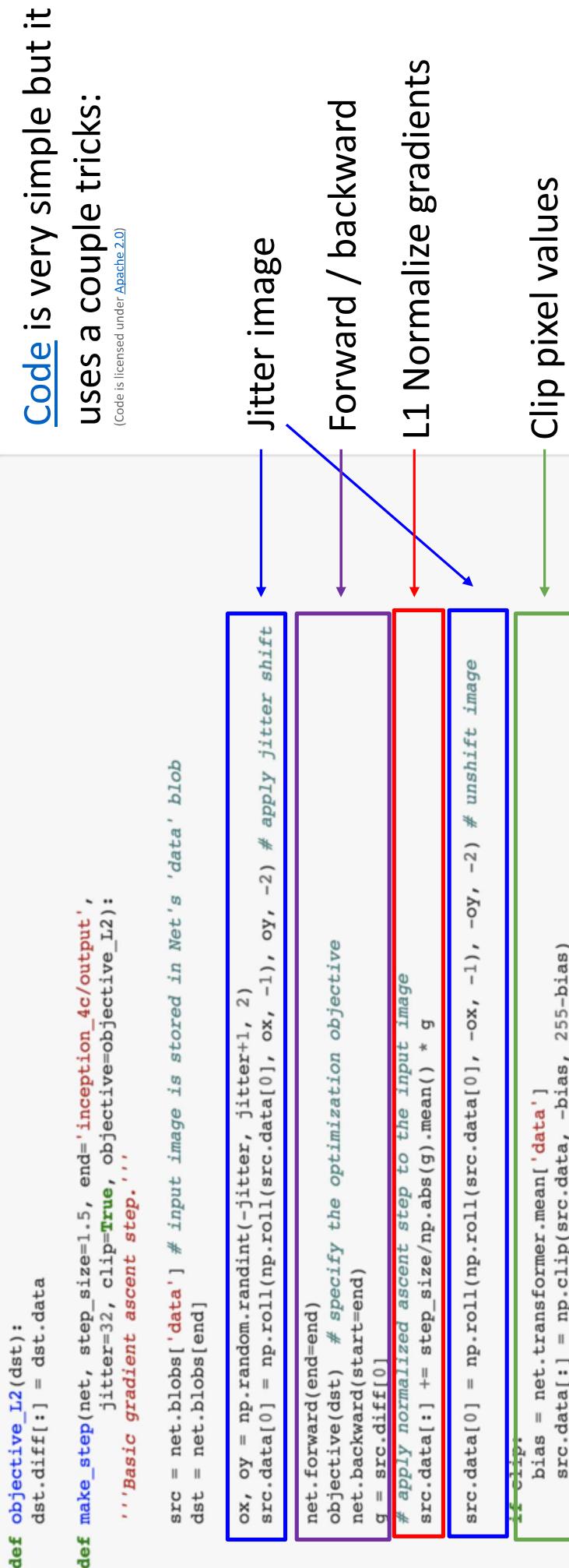
net.forward(end=end)
objective(dst) # specify the optimization objective
net.backward(start=end)
q = src.diff[0]
# apply normalized ascent step to the input image
src.data[:] += step_size/np.abs(q).mean() * q
src.data[0] = np.roll(np.roll(src.data[0], -ox, -1), -oy, -2) # unshift image

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

Jitter image → Forward / backward → L1 Normalize gradients

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

DeepDream: Amplify Existing Features



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

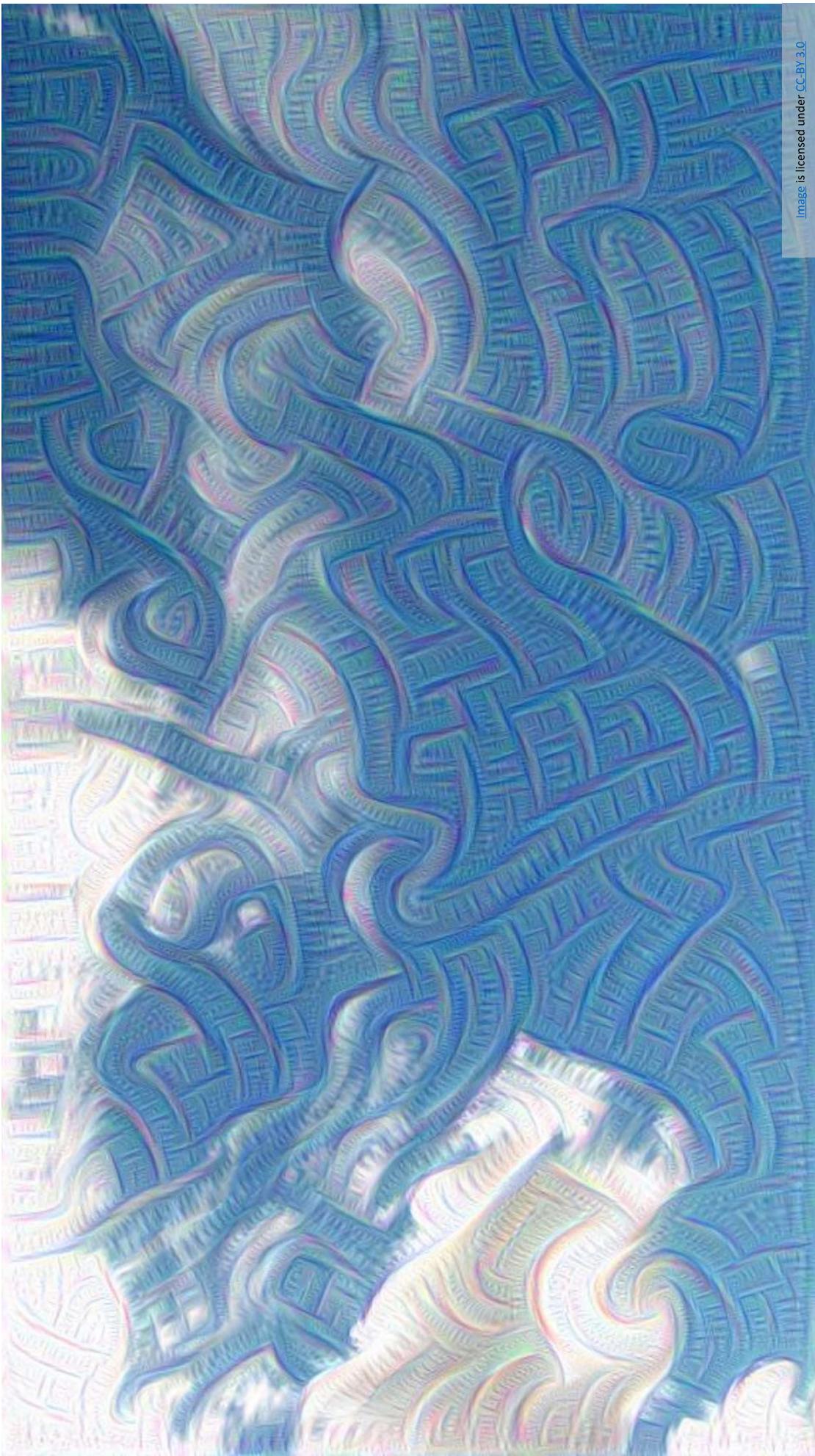
Justin Johnson

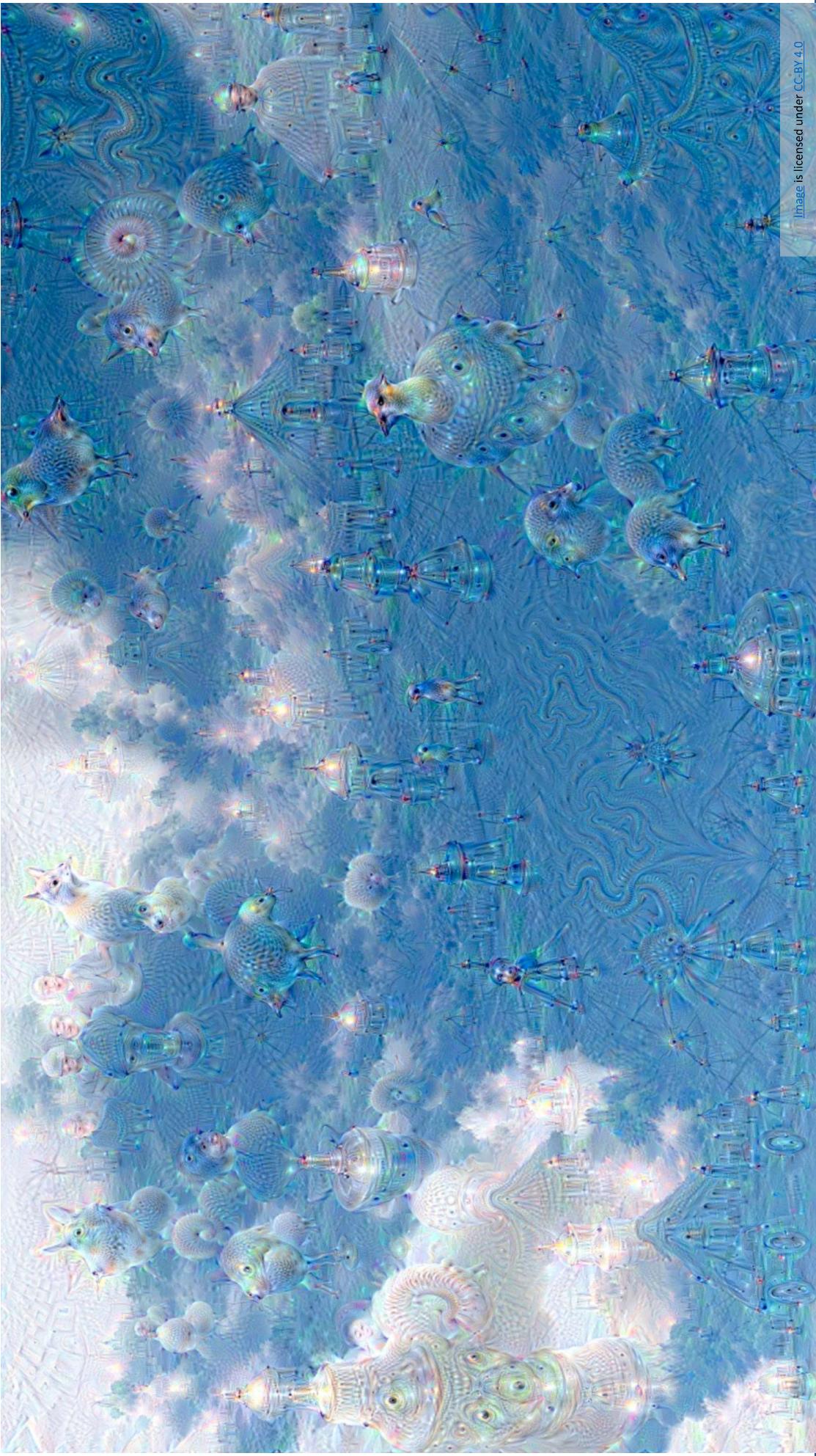
Lecture 14 - 46

November 4, 2019

Sky image is licensed under CC-BY-SA 3.0







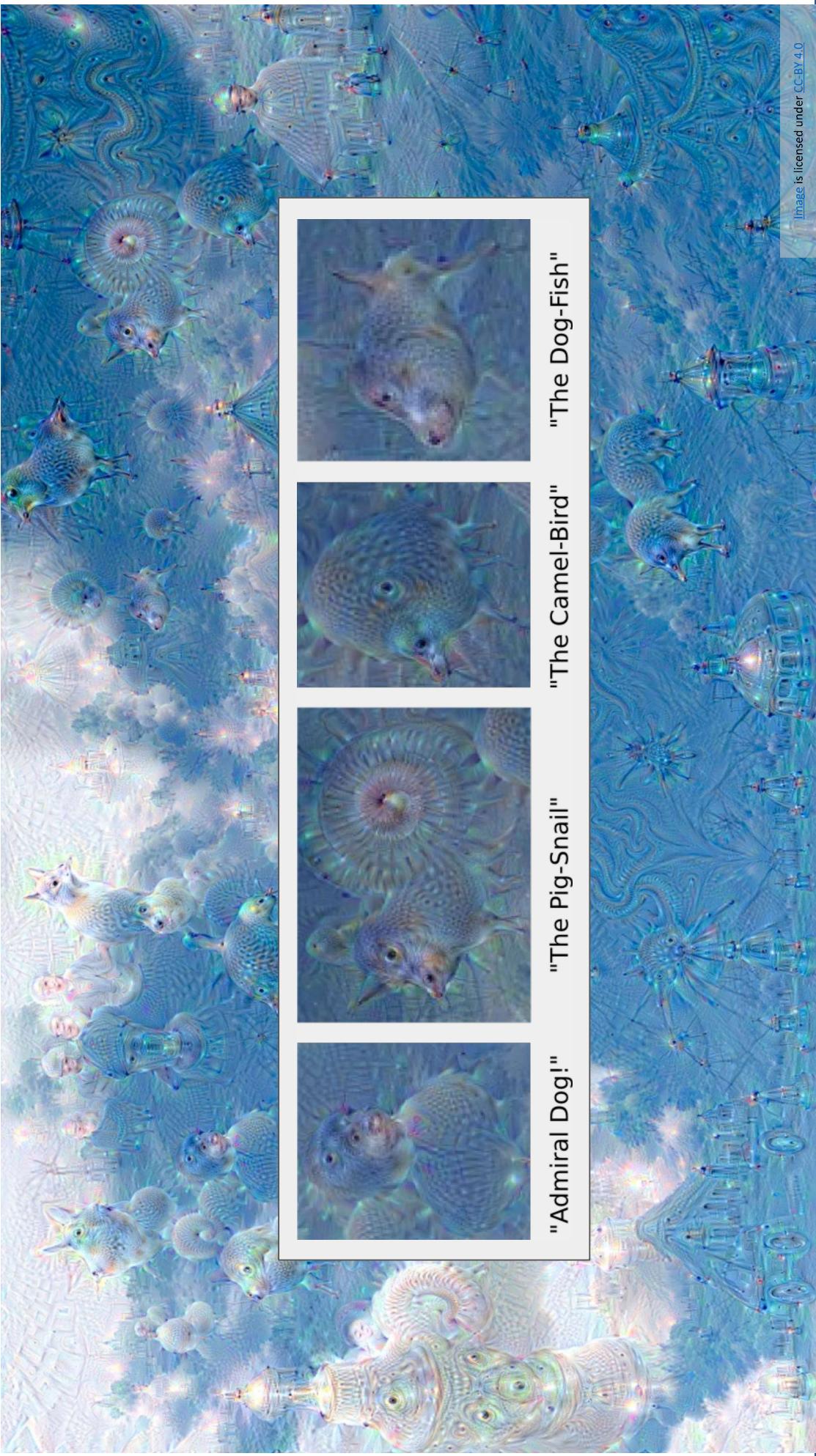
November 4, 2019

Lecture 14 - 49

Justin Johnson

Image is licensed under CC-BY 4.0

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

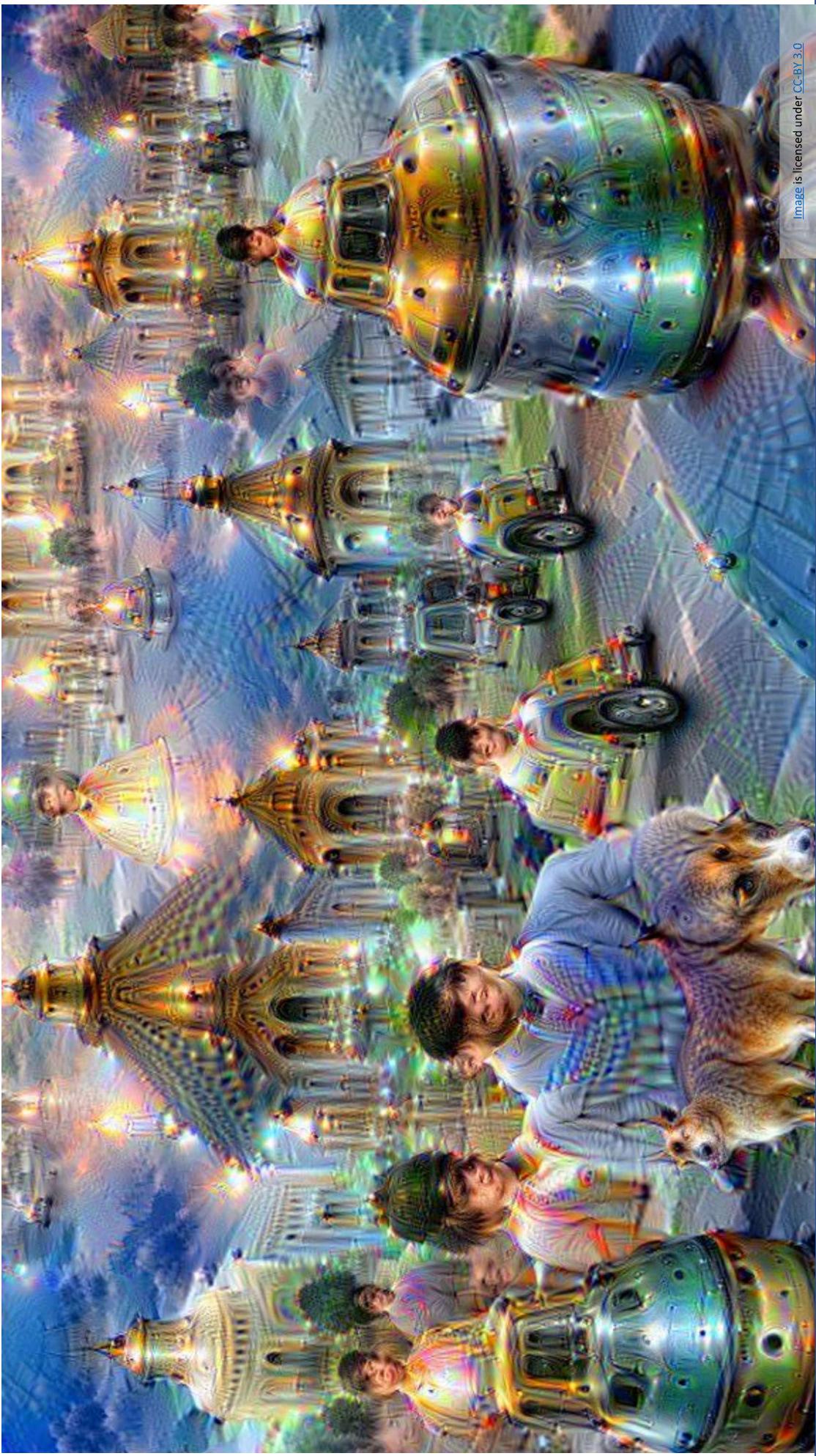


November 4, 2019

Lecture 14 - 50

Justin Johnson

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

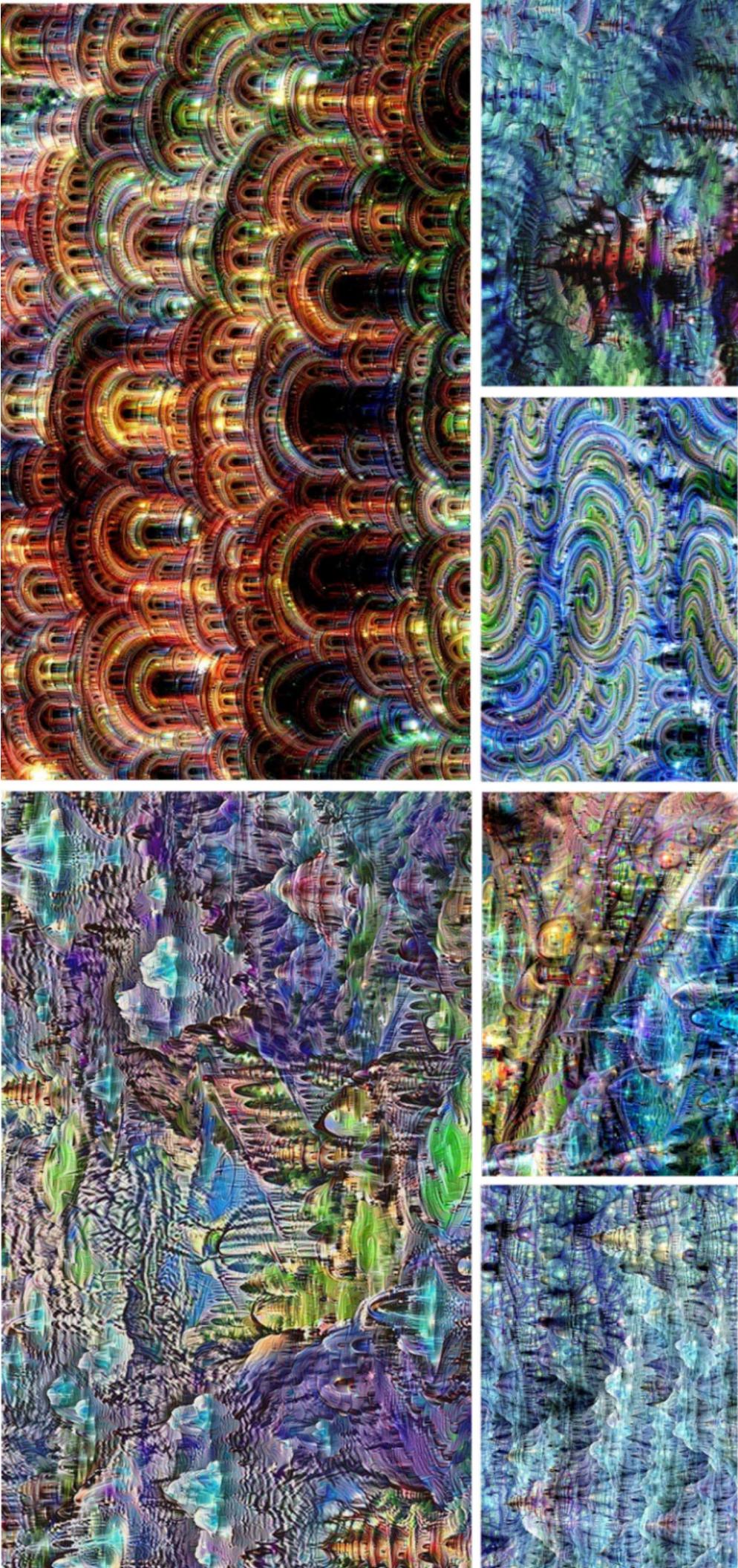


November 4, 2019

Lecture 14 - 51

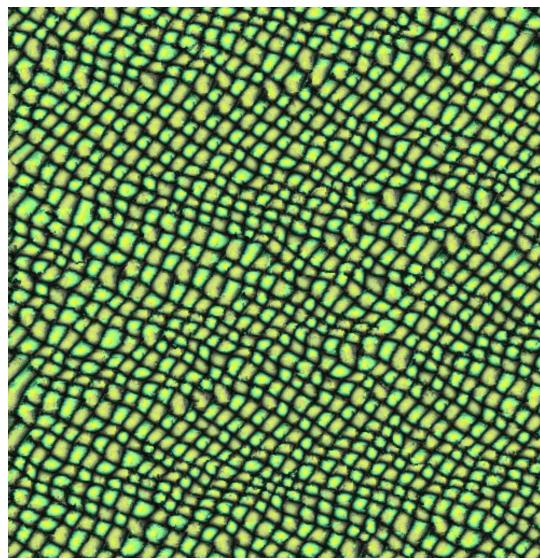
Justin Johnson

Image is licensed under CC-BY 4.0

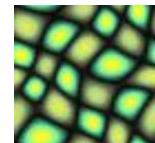


Texture Synthesis

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



Output



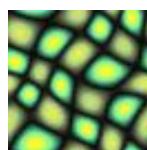
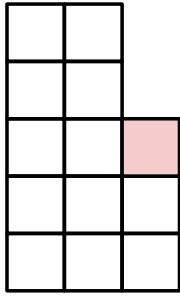
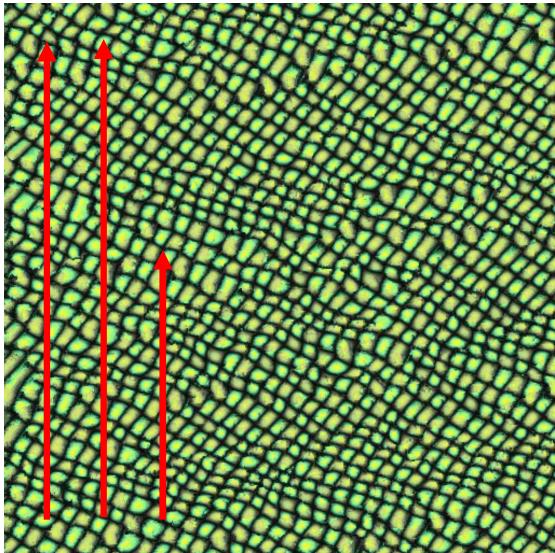
Input



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

Texture Synthesis: Nearest Neighbor

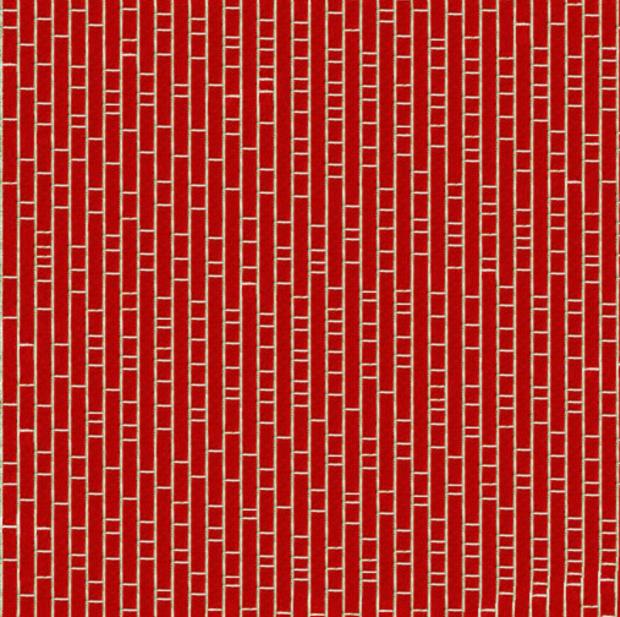
Generate pixels one at a time in scanline order;
form neighborhood of already generated pixels
and copy nearest neighbor from input



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

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

Texture Synthesis: Nearest Neighbor



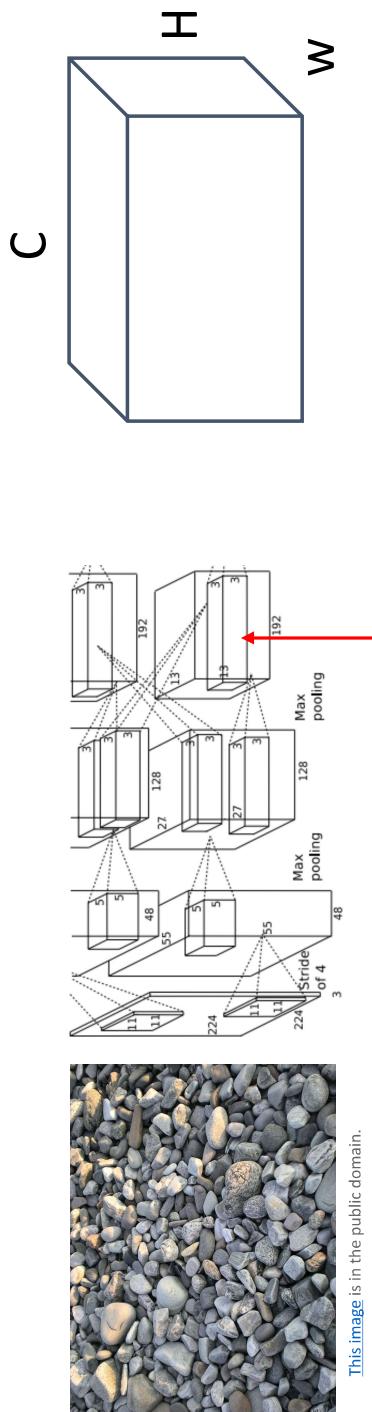
Images licensed under the MIT license

Justin Johnson

Lecture 14 - 54

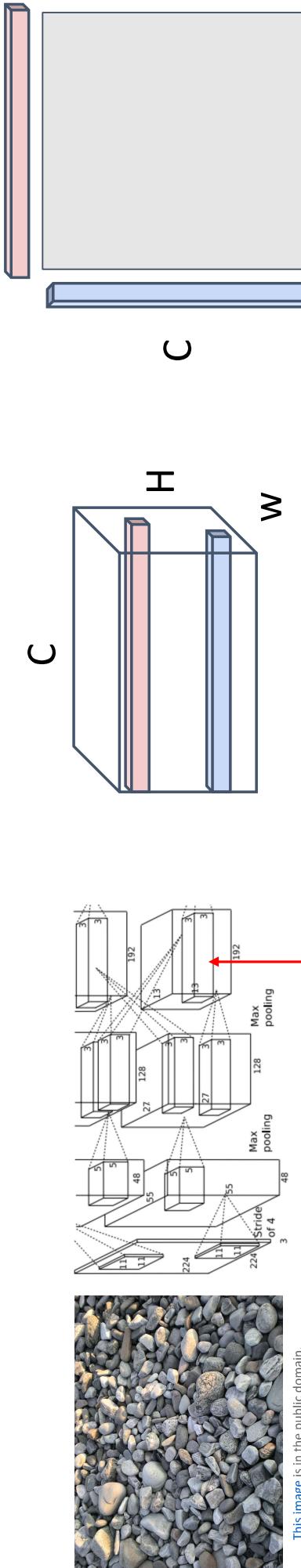
November 4, 2019

Texture Synthesis with Neural Networks: Gram Matrix



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

Texture Synthesis with Neural Networks: Gram Matrix

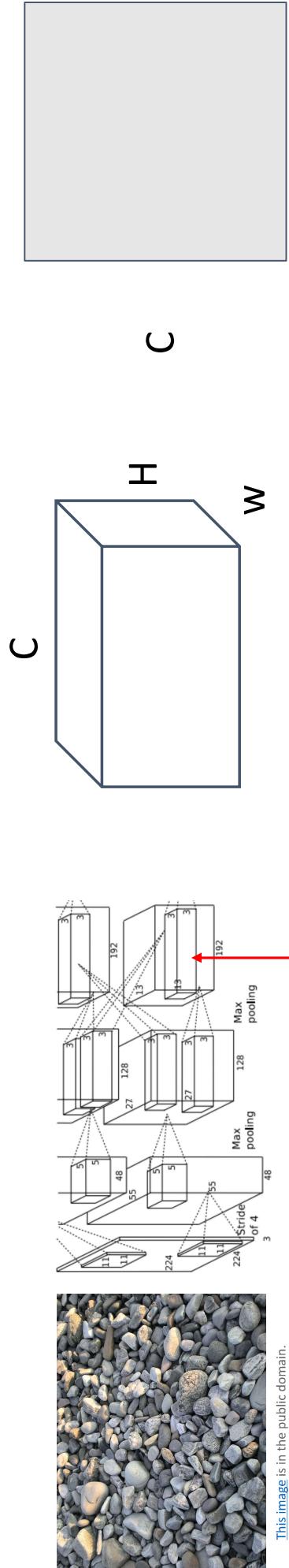


This image is in the public domain.

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

Outer product of two C -dimensional vectors gives $C \times C$ matrix of elementwise products

Texture Synthesis with Neural Networks: Gram Matrix

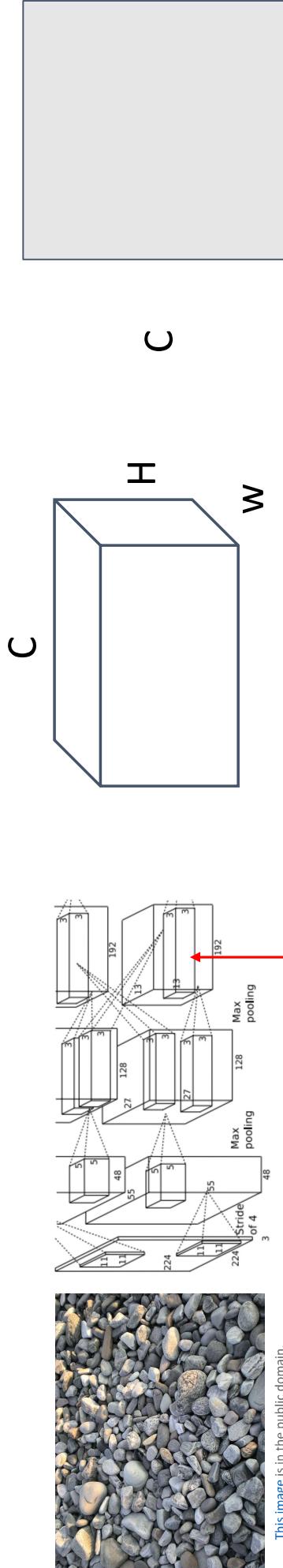


Each layer of CNN gives $C \times H \times W$ tensor of features; $H \times W$ grid of C -dimensional vectors

Outer product of two C -dimensional vectors gives $C \times C$ matrix of elementwise products

Average over all HW pairs gives **Gram Matrix** of shape $C \times C$ giving unnormalized covariance

Texture Synthesis with Neural Networks: Gram Matrix



This image is in the public domain.

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

Efficient to compute;
reshape features from

Outer product of two C -dimensional vectors
gives $C \times C$ matrix of elementwise products

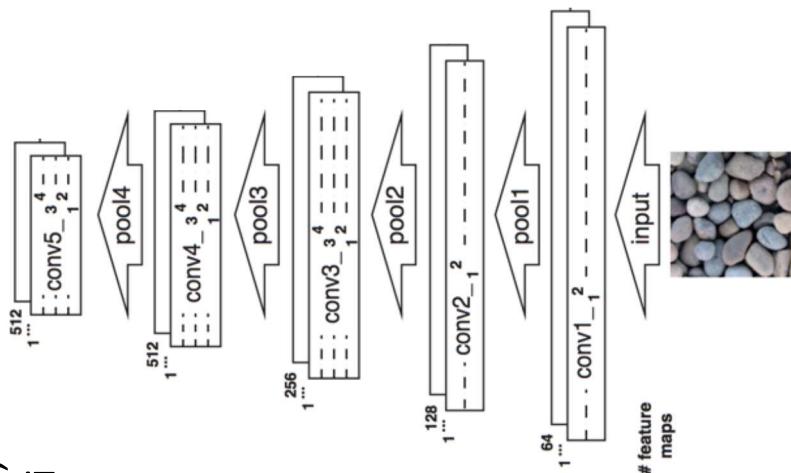
$C \times H \times W$ to $F = C \times HW$

Average over all HW pairs gives **Gram Matrix**
of shape $C \times C$ giving unnormalized covariance

then compute $G = FF^\top$

Neural Texture Synthesis

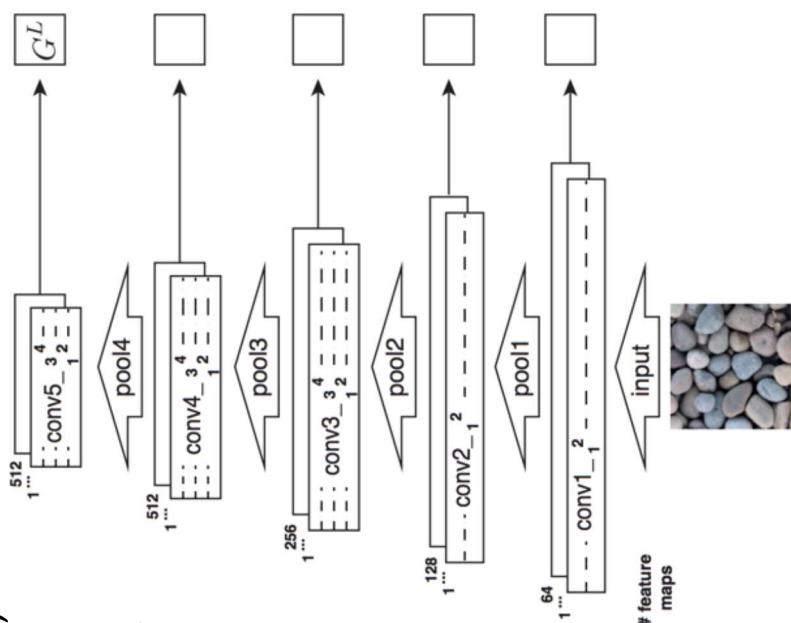
1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$



Neural Texture Synthesis

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

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

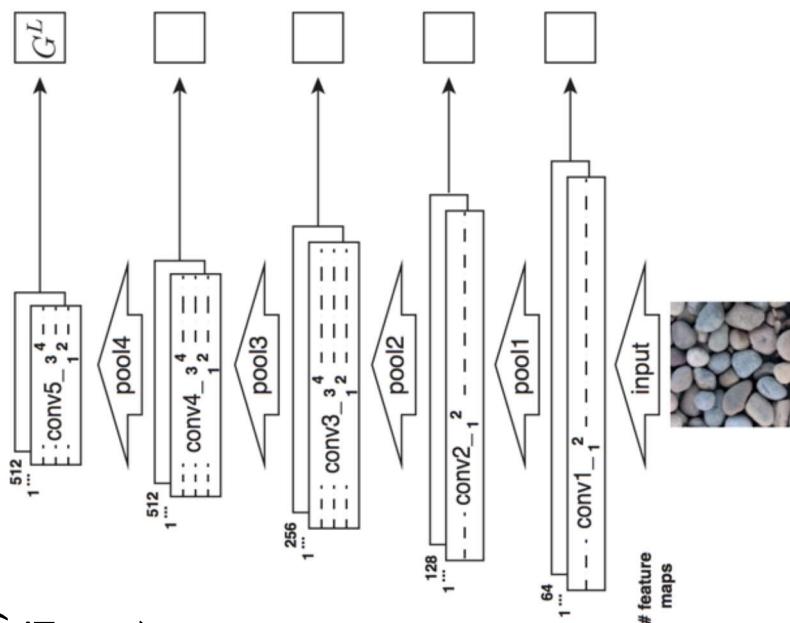


Neural Texture Synthesis

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

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

4. Initialize generated image from random noise

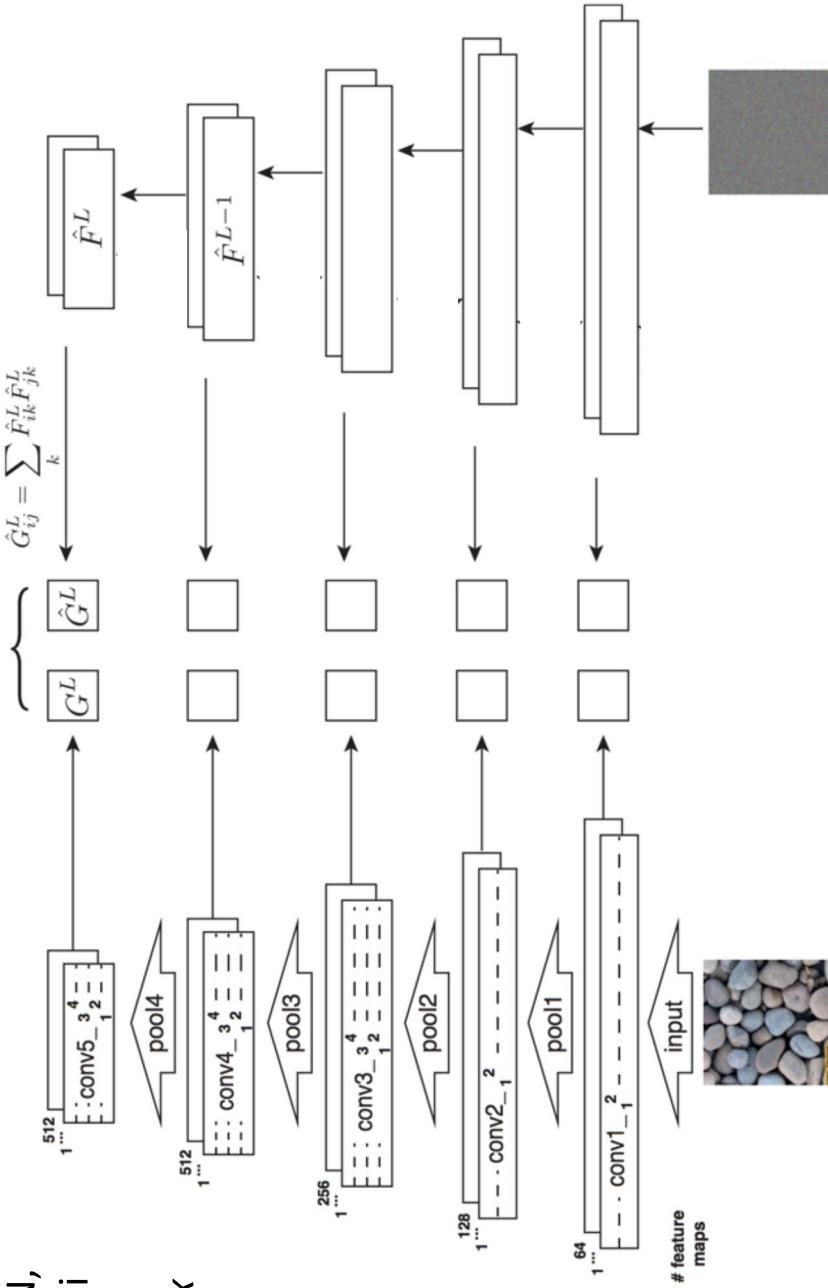


Neural Texture Synthesis

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

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

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



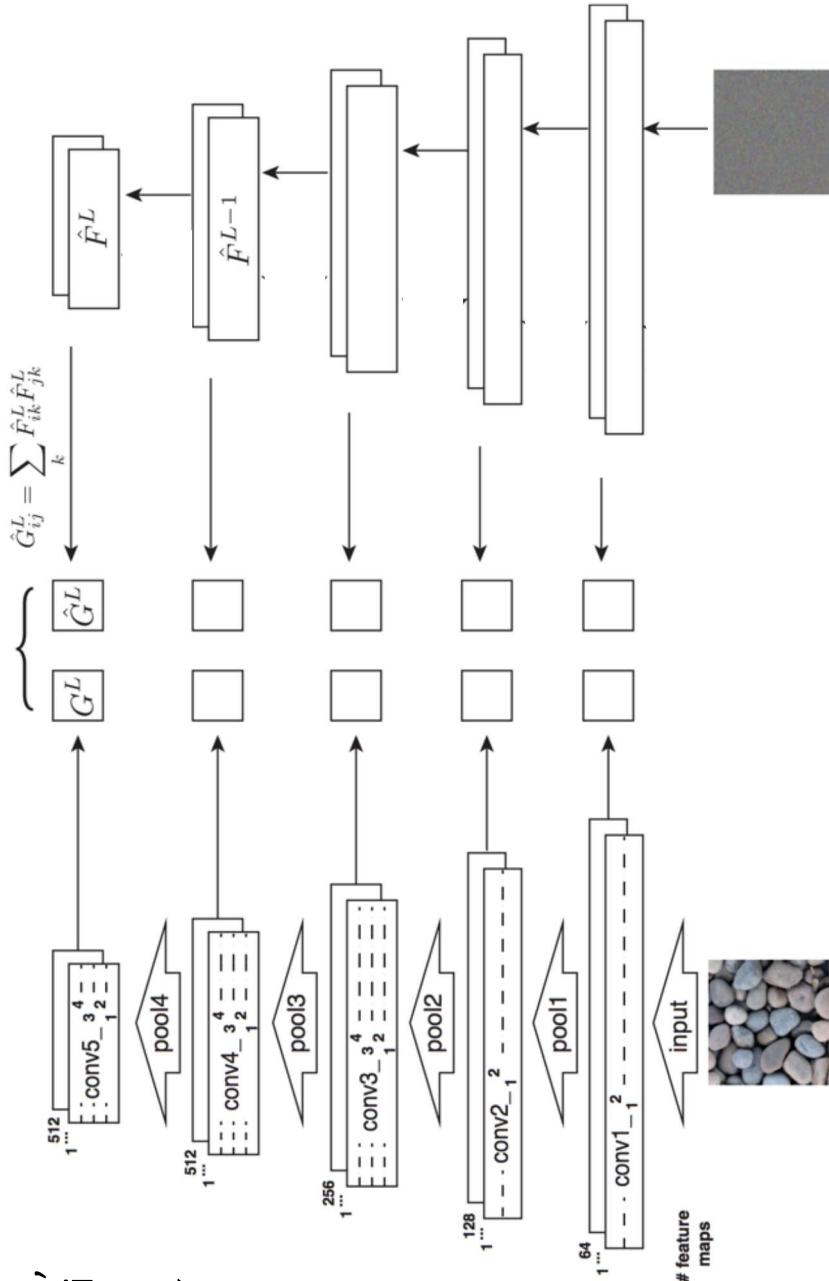
Neural Texture Synthesis

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

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

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

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



Neural Texture Synthesis

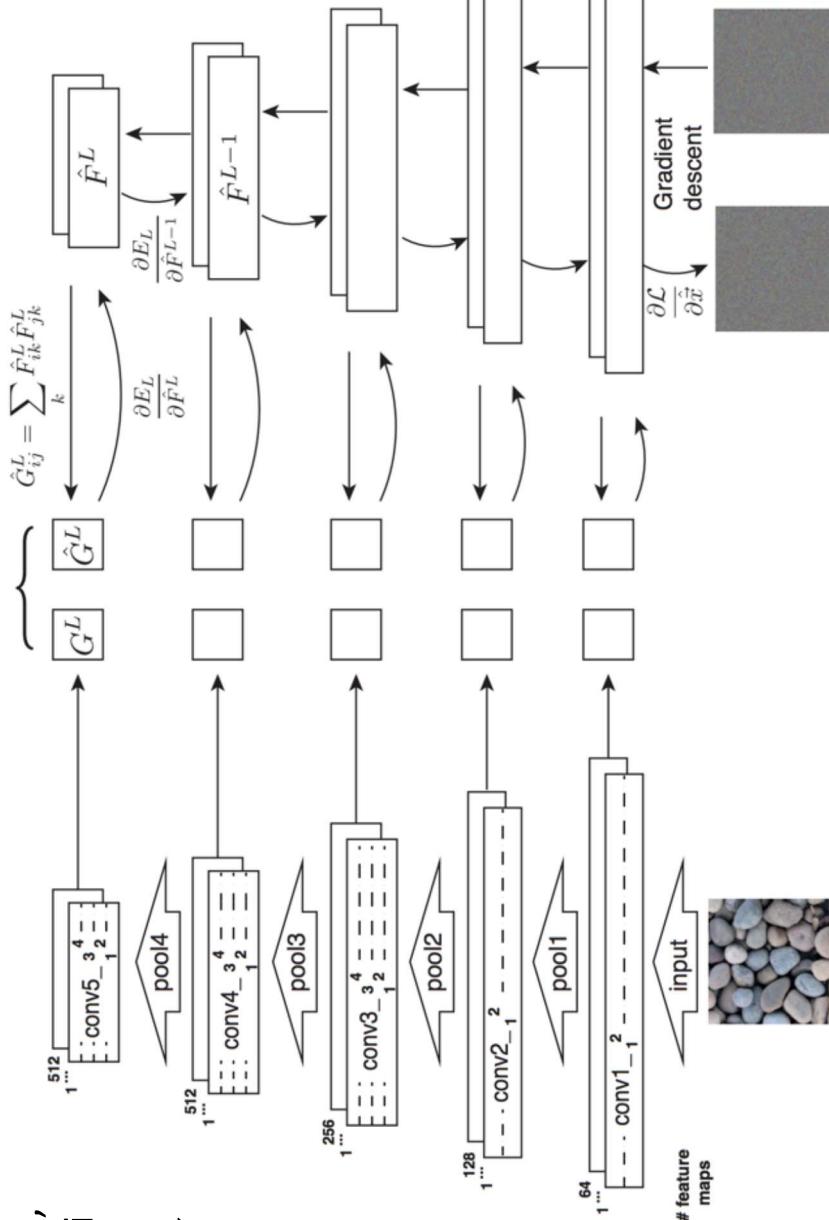
1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the Gram matrix giving outer product of features:

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

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

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



Neural Texture Synthesis

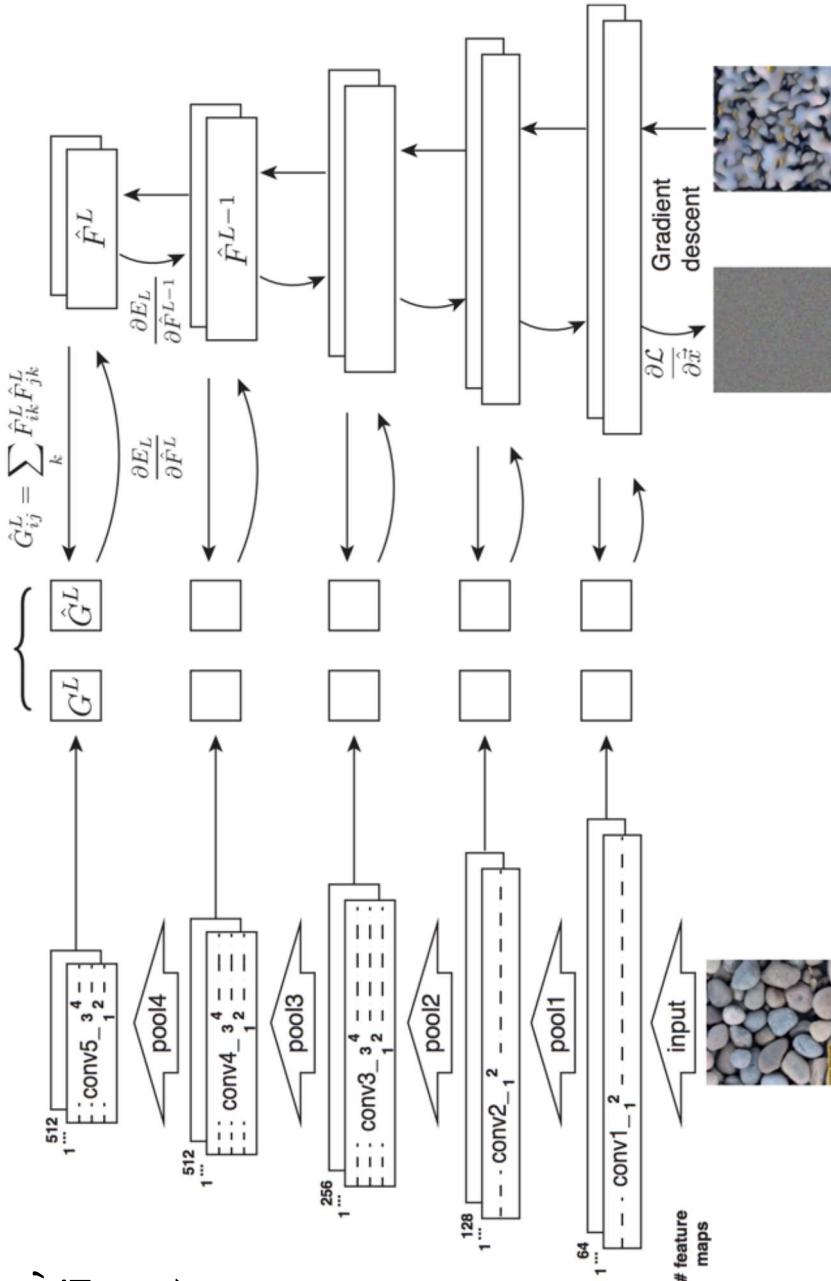
1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the Gram matrix giving outer product of features:

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

4. Initialize generated image from random noise

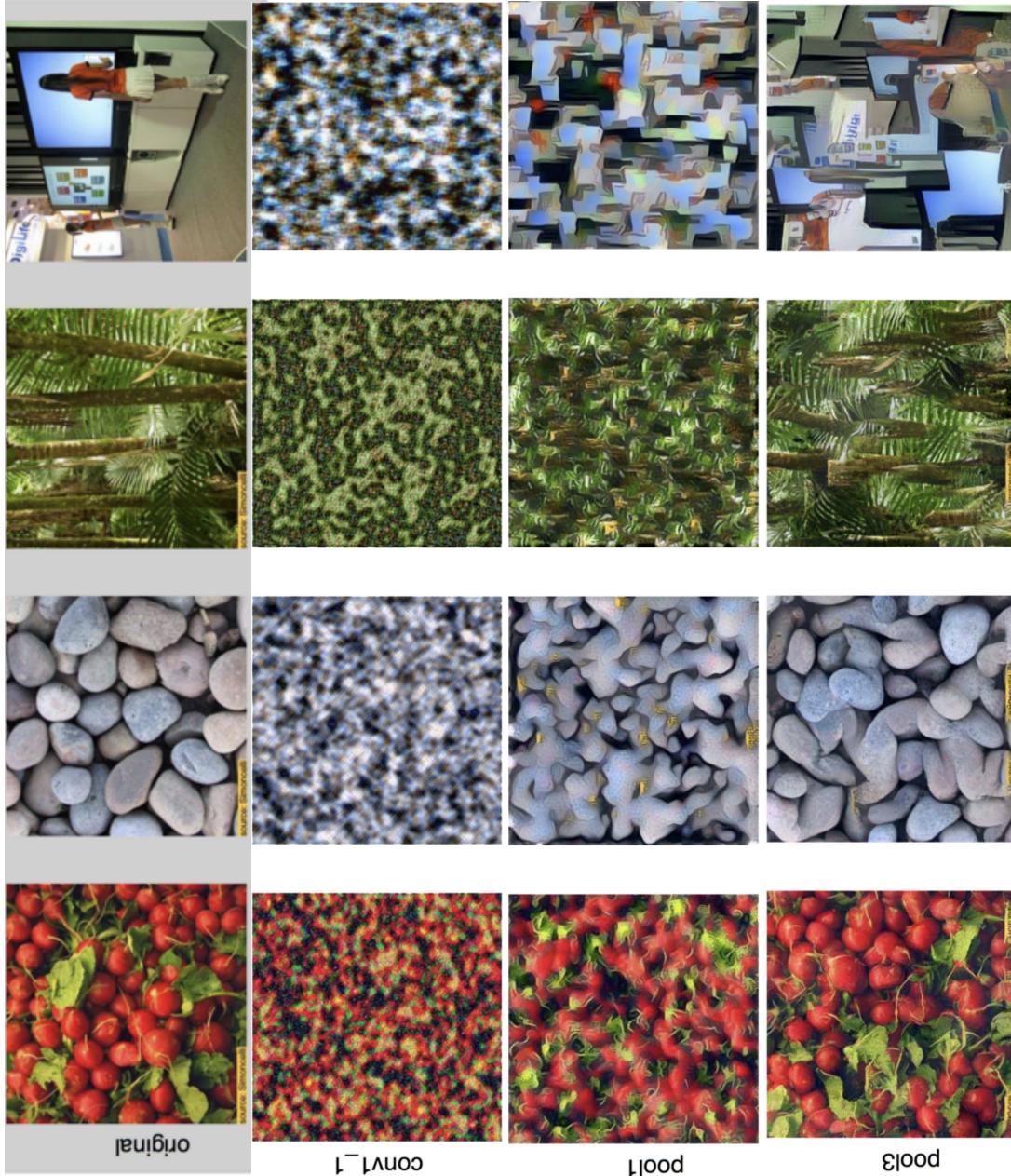
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5

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



Neural Texture Synthesis

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



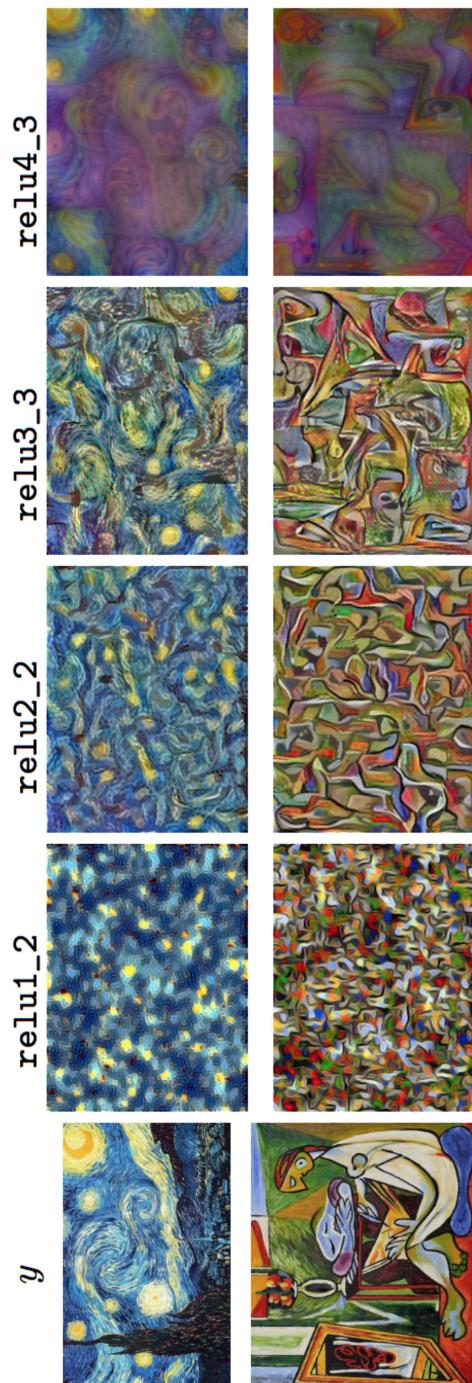
Gatys, Ecker, and Bethge, "Texture Synthesis Using Convolutional Neural Networks", NIPS 2015

Justin Johnson

Lecture 14 - 66

November 4, 2019

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.

Neural Style Transfer: Feature + Gram Reconstruction

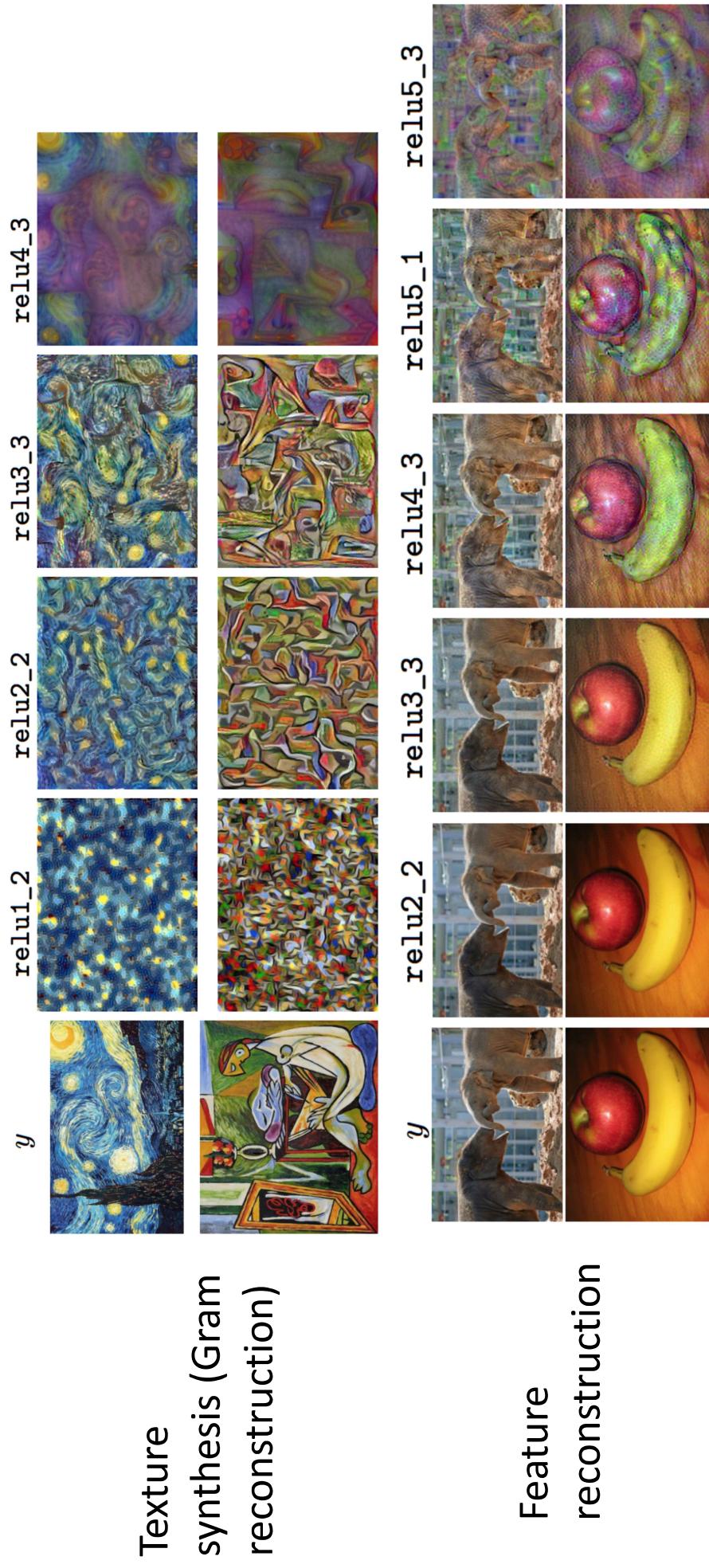


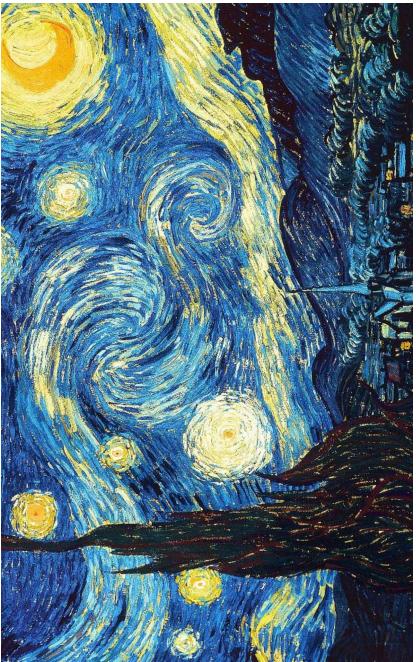
Figure from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016.

Neural Style Transfer

Content Image



Style Image



Output Image

Match features
from content
image and Gram
matrices from
style image

=

This image is licensed under CC-BY 3.0

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

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

Justin Johnson

Lecture 14 - 69

November 4, 2019

Neural Style Transfer

Content Image



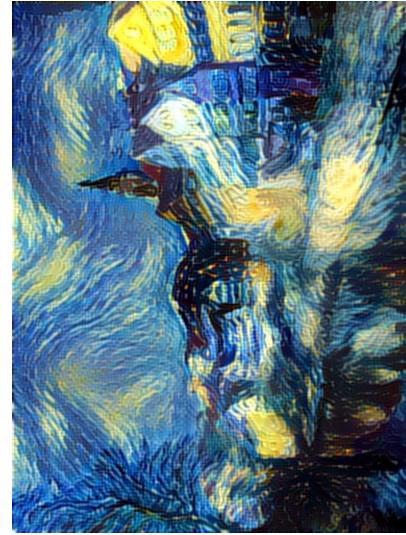
This image is licensed under CC-BY 3.0

Style Image



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

Output Image



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

=

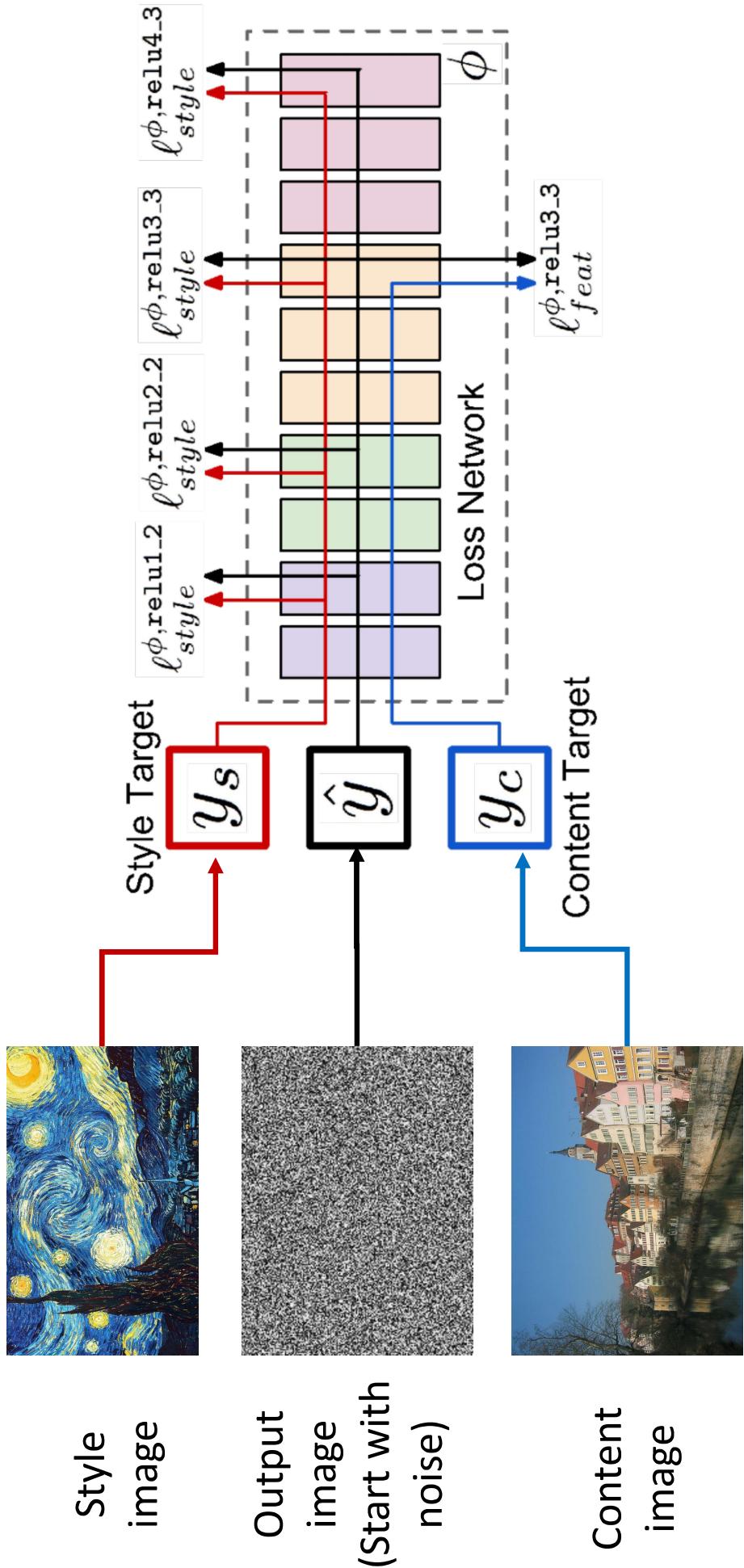
+

Justin Johnson

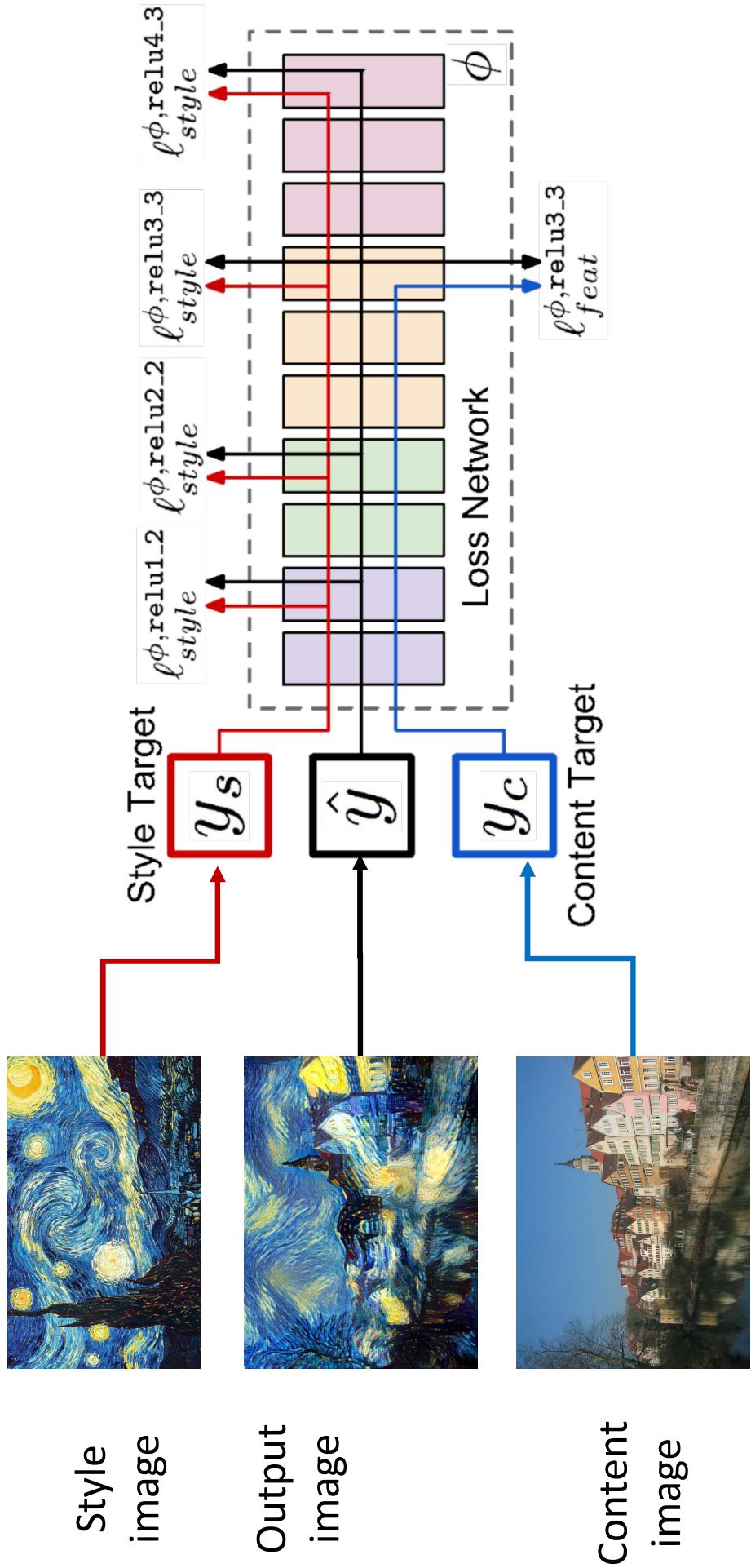
Lecture 14 - 70

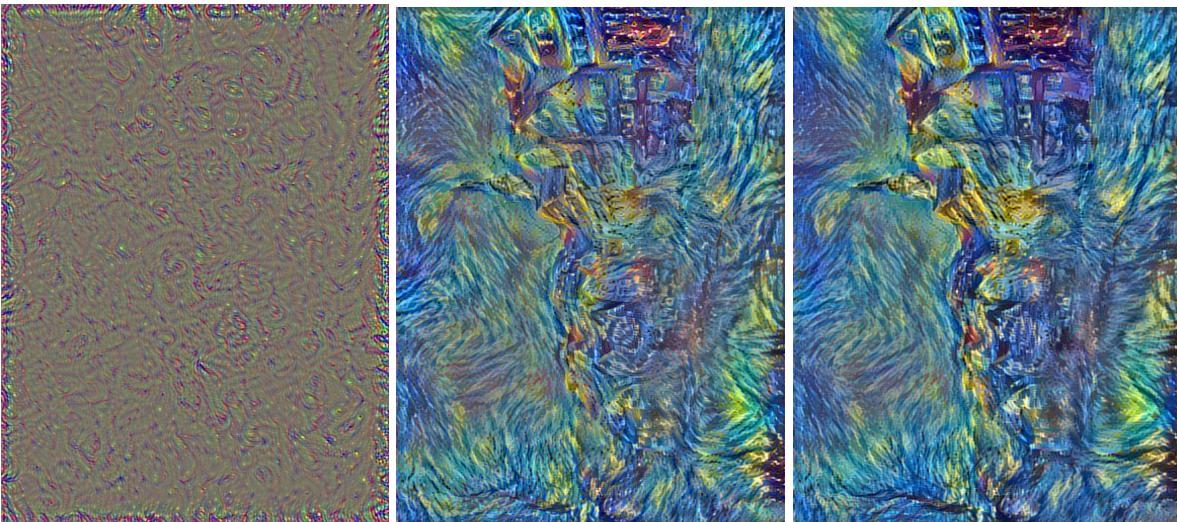
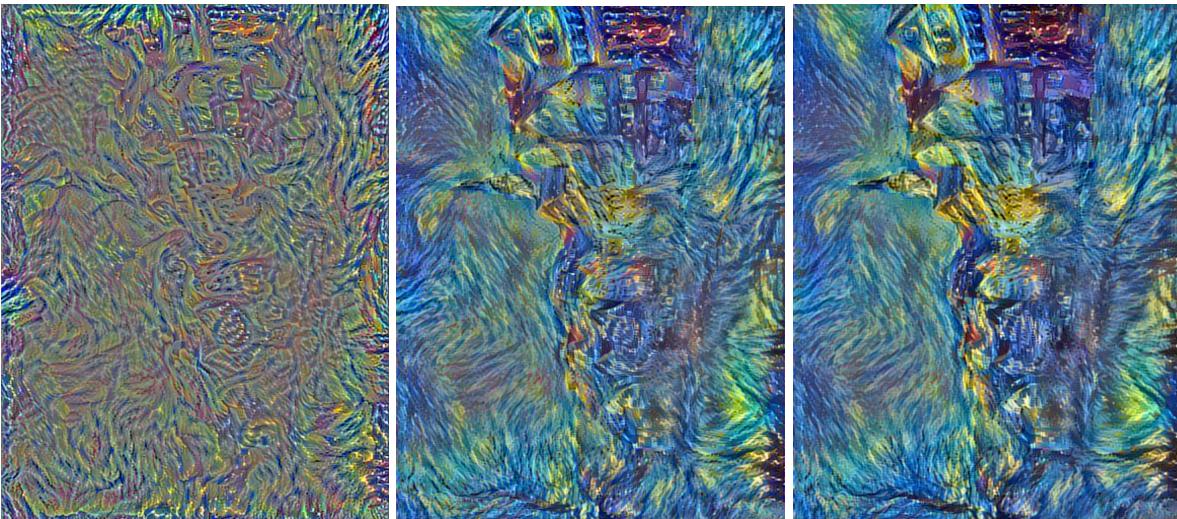
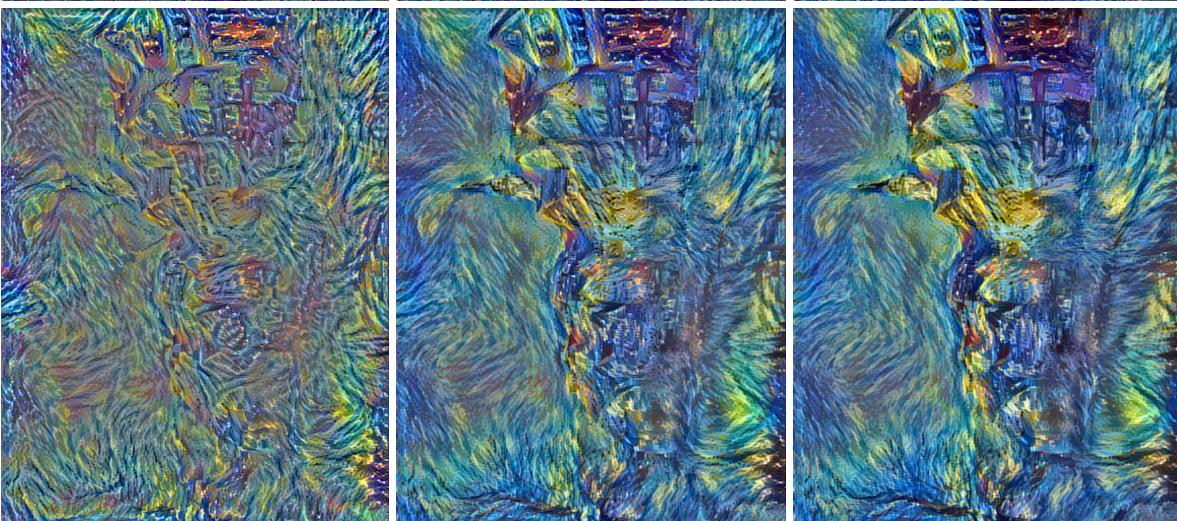
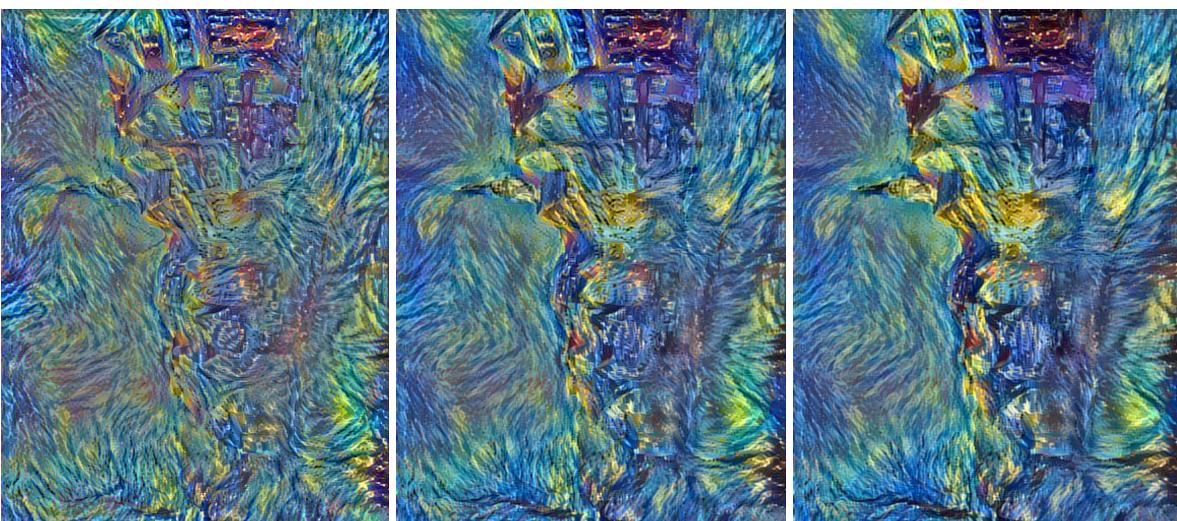
November 4, 2019

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

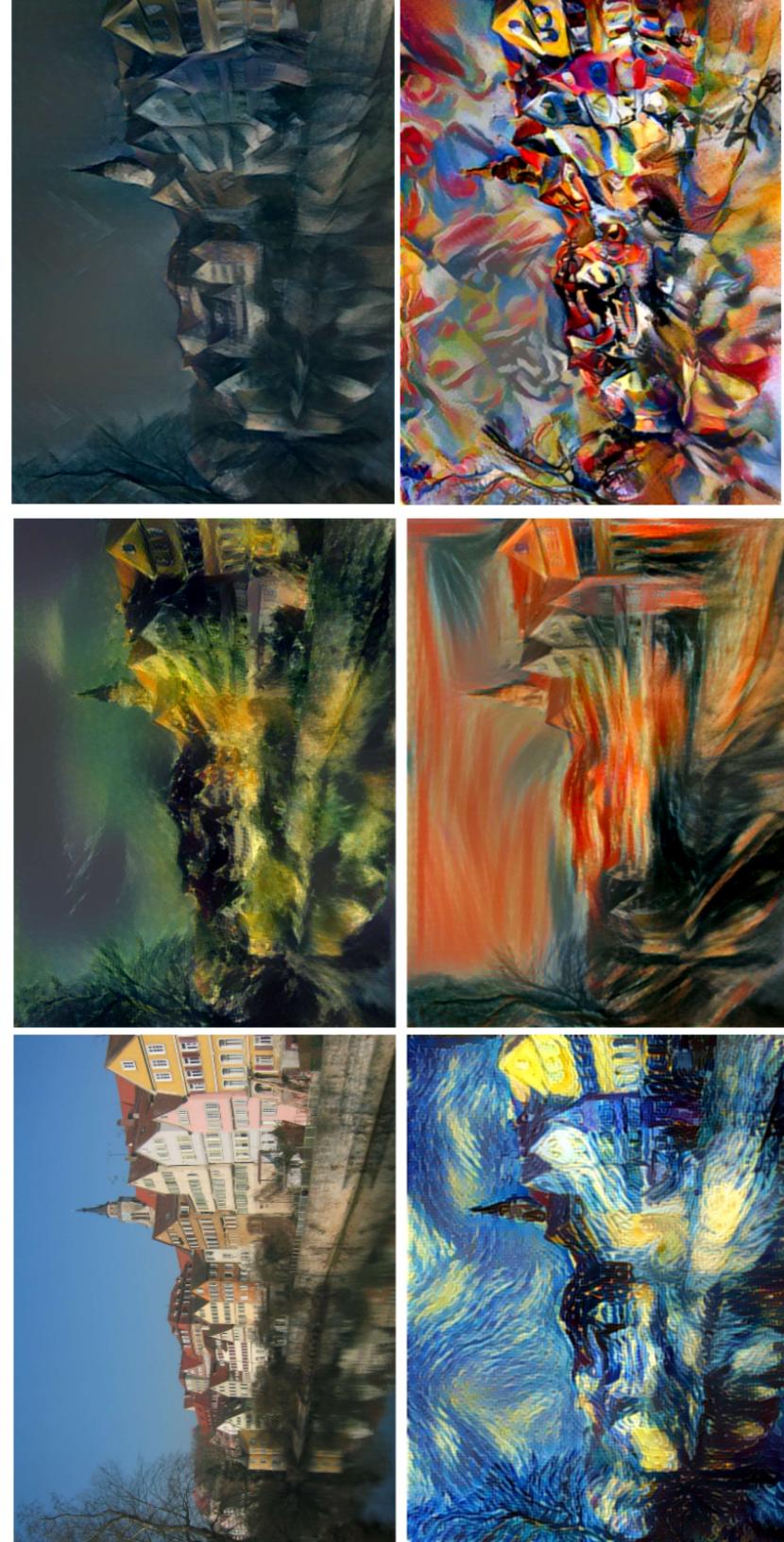


Gatys, Ecker, and Bethge, "Image style transfer using convolutional neural networks", CVPR 2016
 Figure adapted from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016.





Neural Style Transfer



Example outputs
from my
implementation
(in Lua Torch)

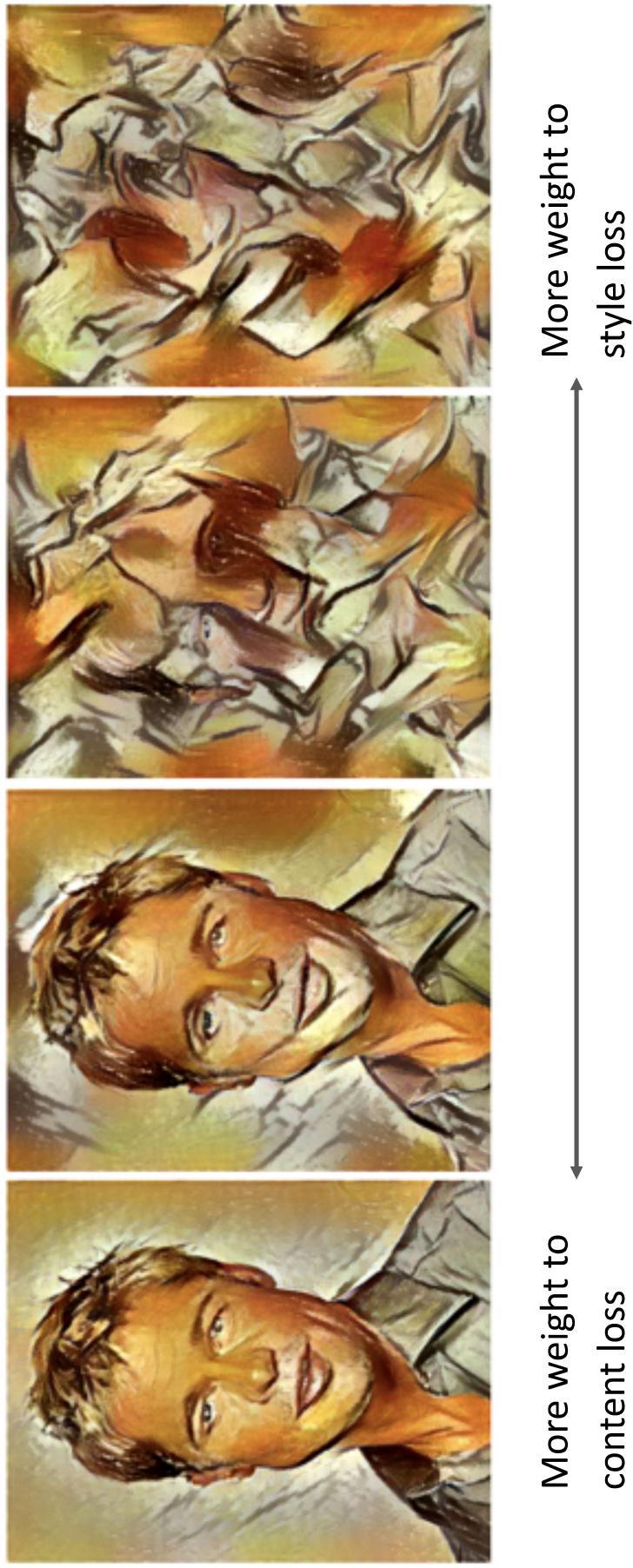
Gatys, Ecker, and Bethge, "Image style transfer using convolutional neural networks", CVPR 2016.
Figure copyright Justin Johnson, 2015.

Justin Johnson

Lecture 14 - 74

November 4, 2019

Neural Style Transfer



Neural Style Transfer

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



Larger style image

Smaller style image

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

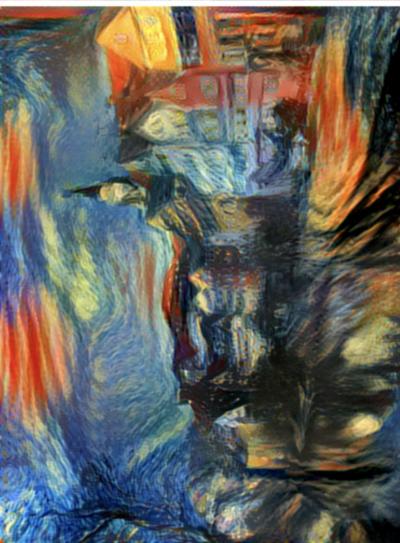
Justin Johnson

Lecture 14 - 76

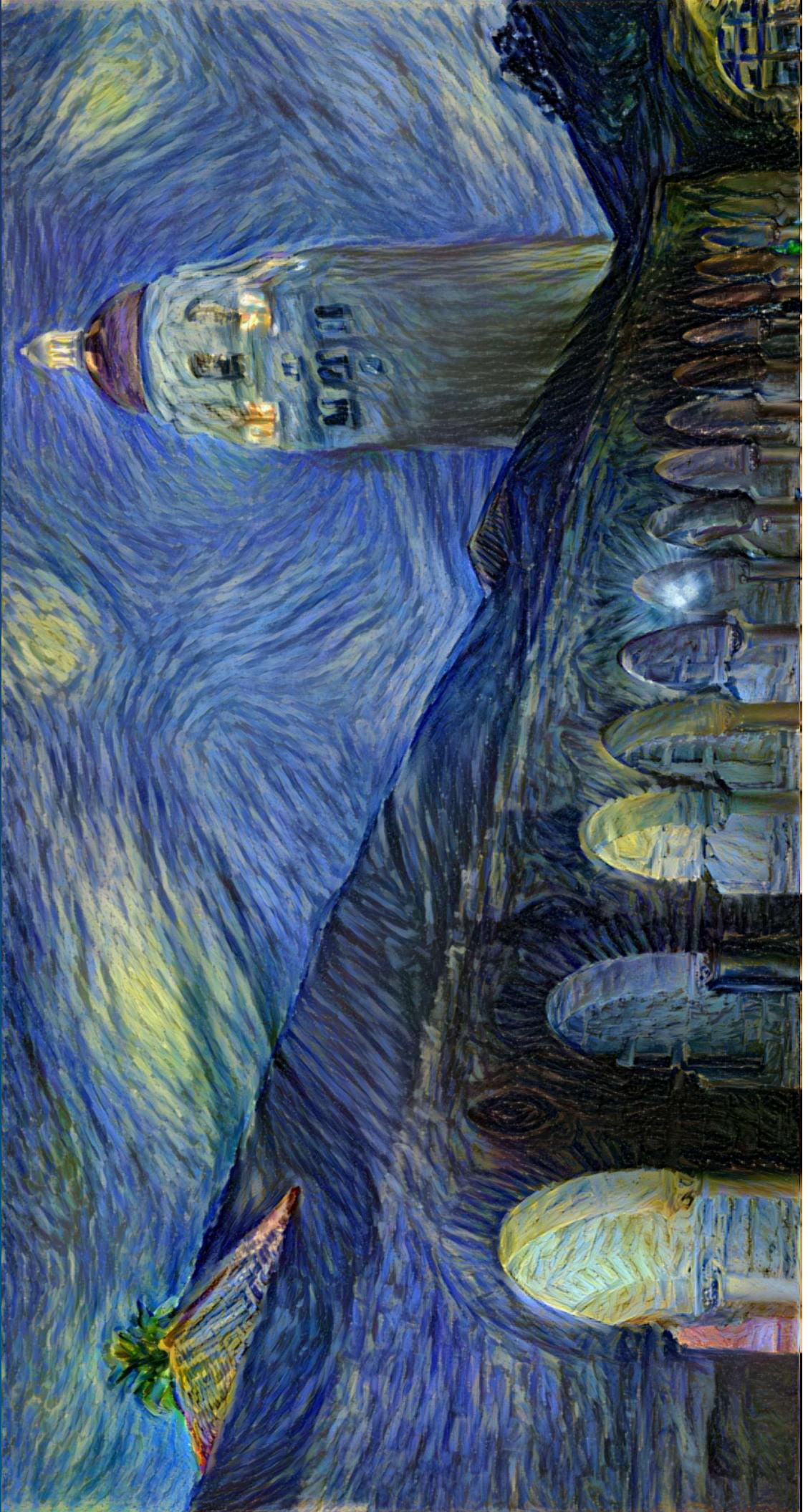
November 4, 2019

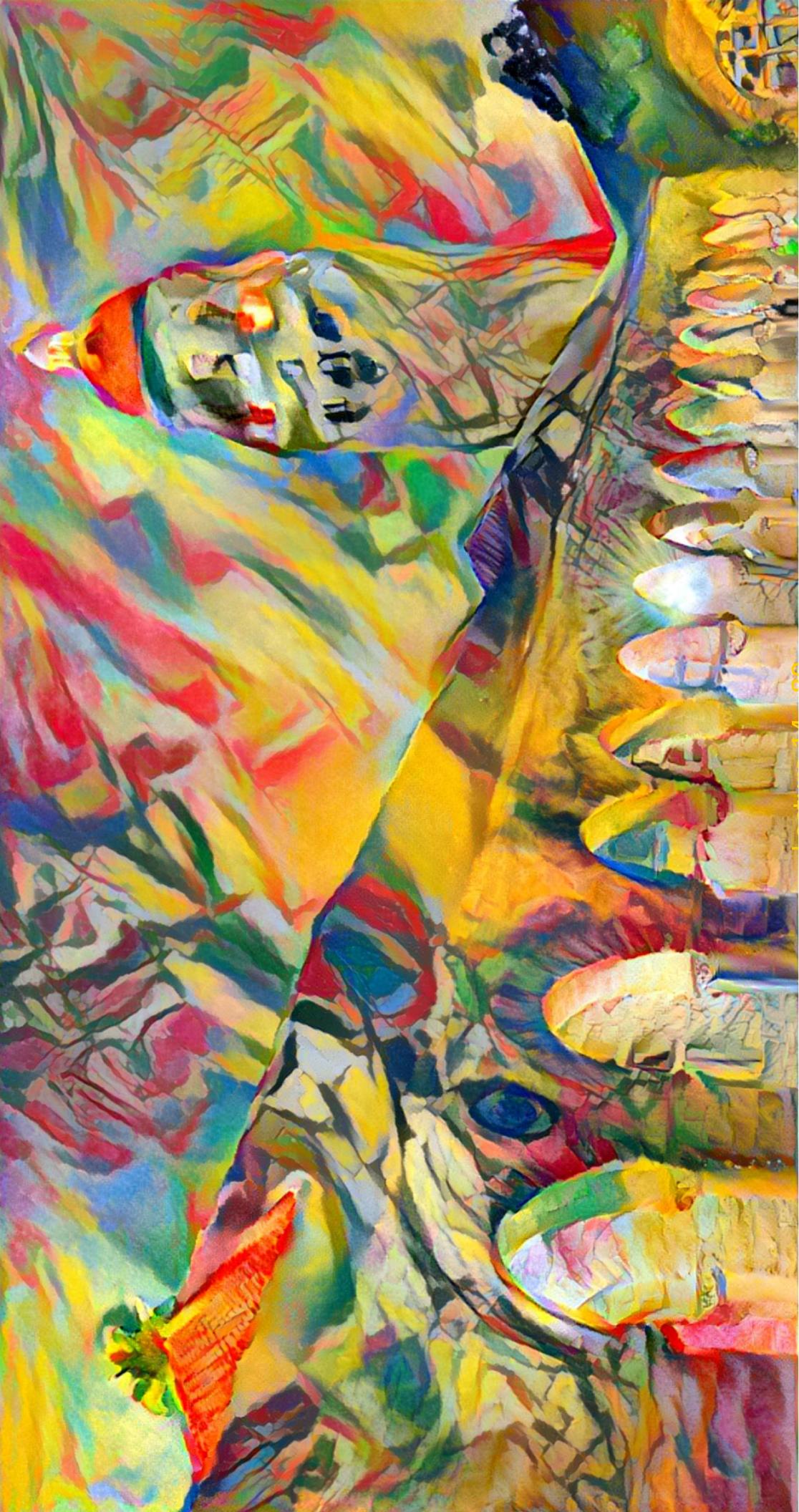
Neural Style Transfer: Multiple Style Images

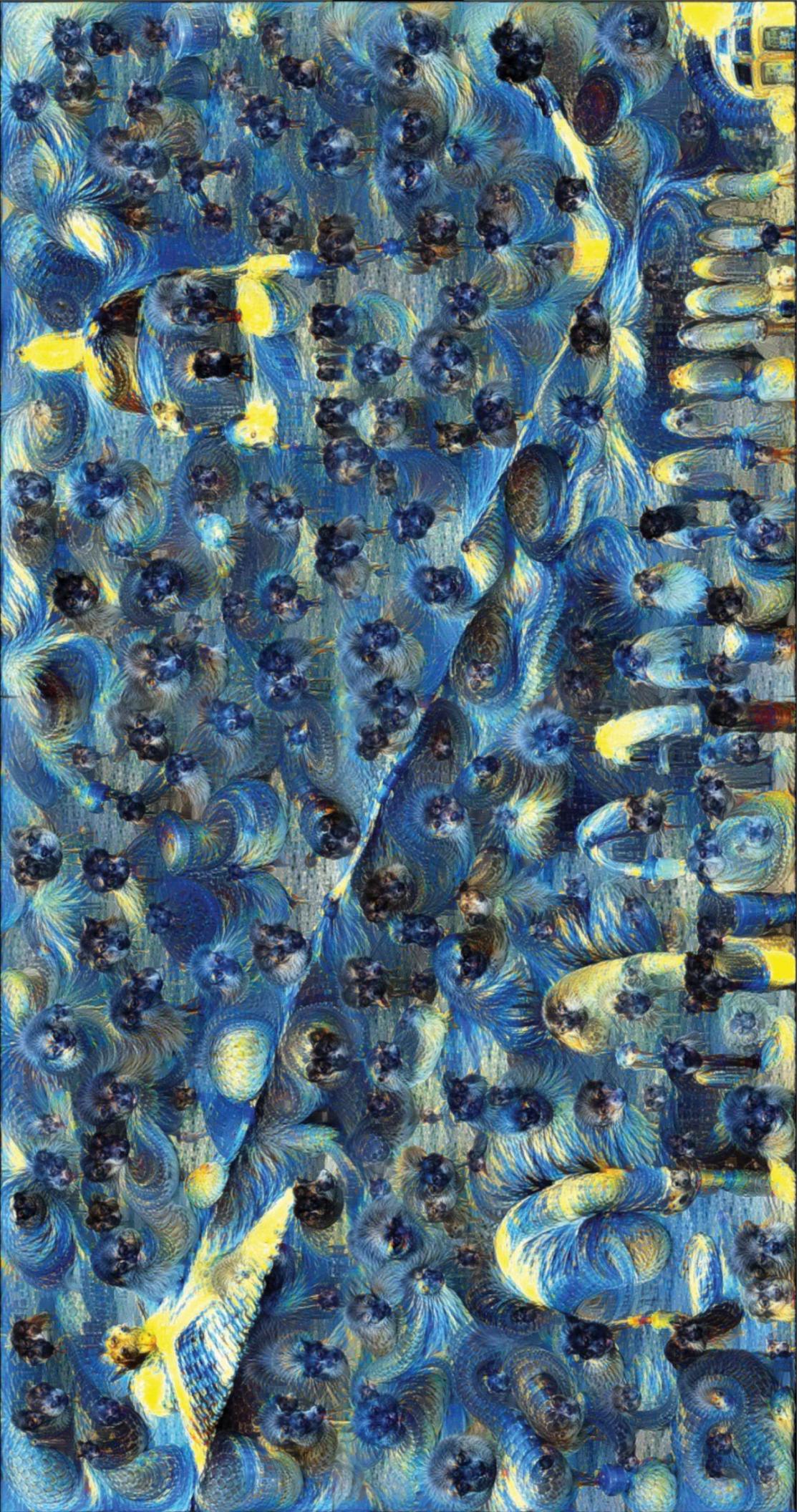
Mix style from
multiple images by
taking a weighted
average of Gram
matrices











Neural Style Transfer

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

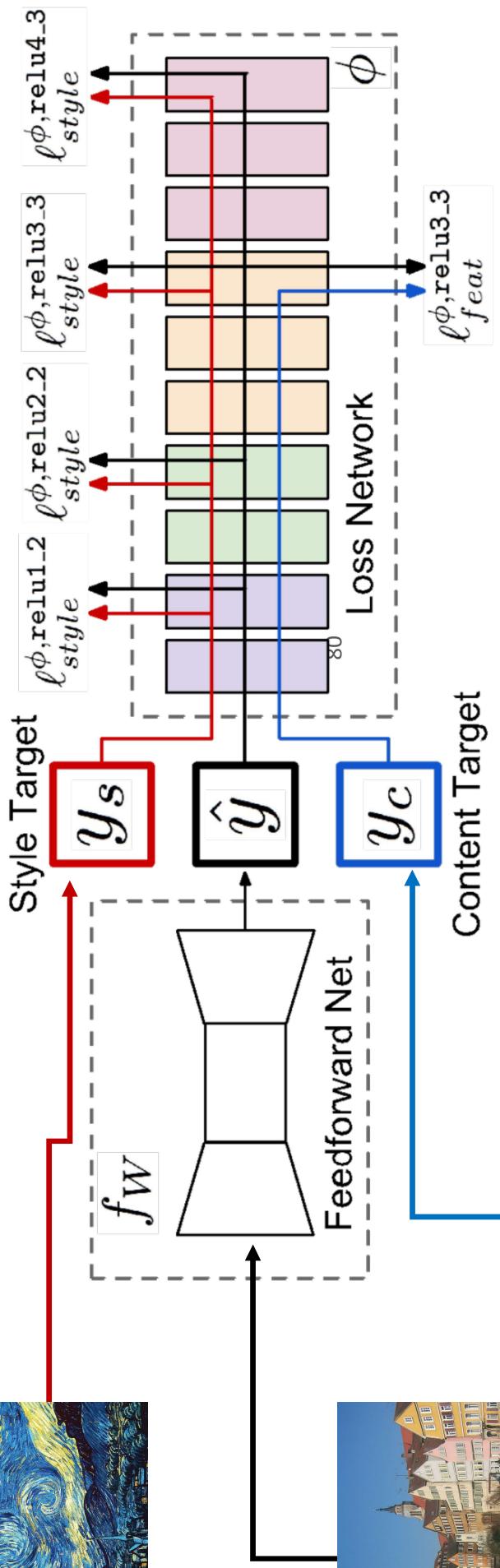
Neural Style Transfer

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

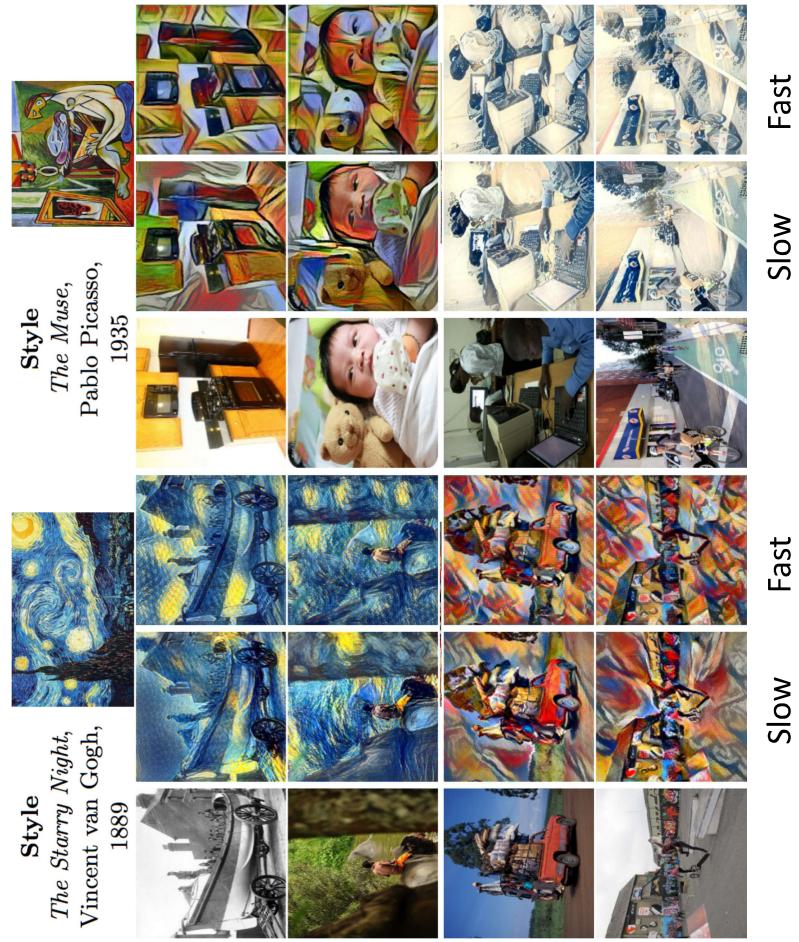
Solution: Train another neural network to perform style transfer for us!

Fast Neural 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



Fast Neural Style Transfer



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

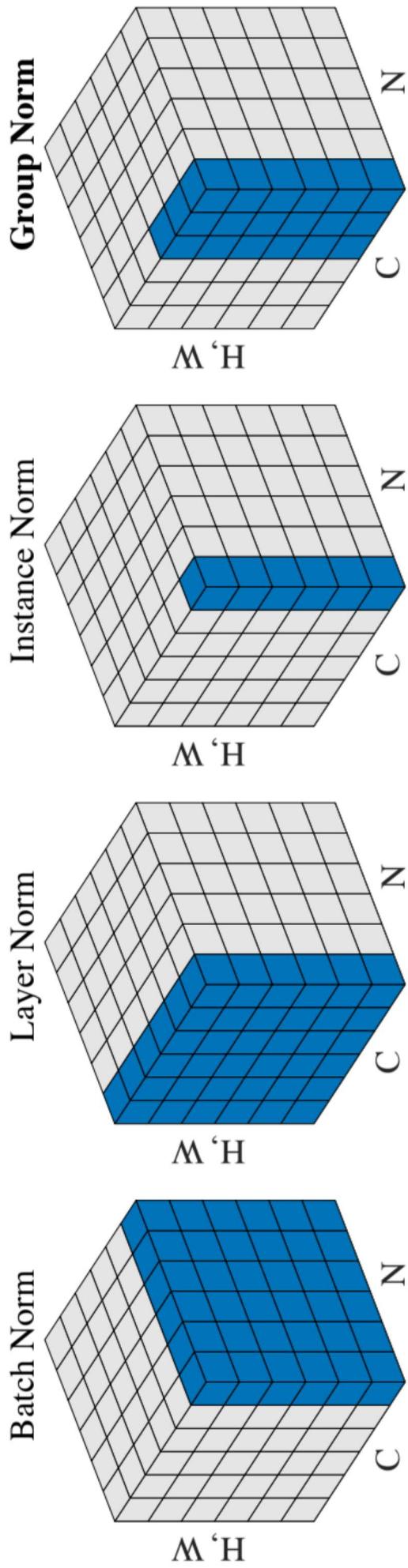
<https://github.com/jcjohnson/fast-neural-style>

Justin Johnson

Lecture 14 - 85

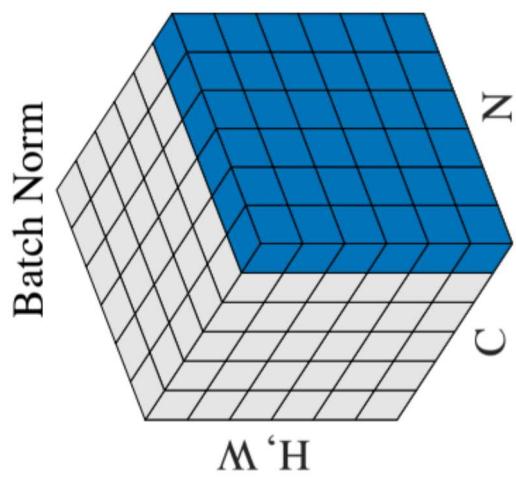
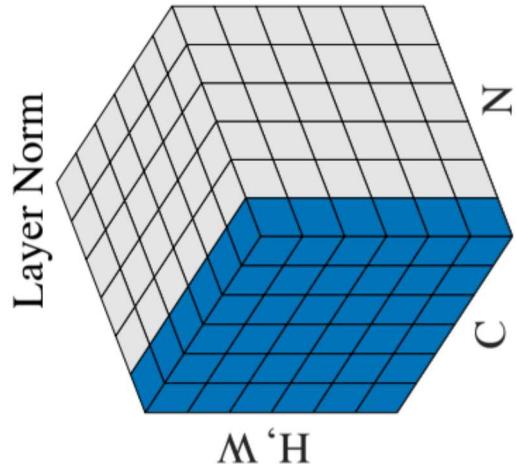
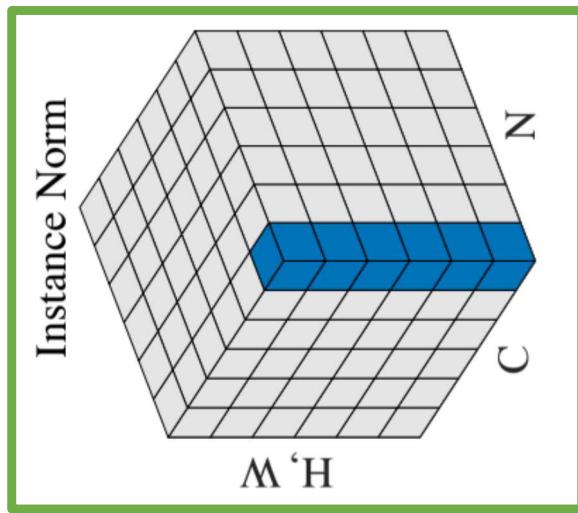
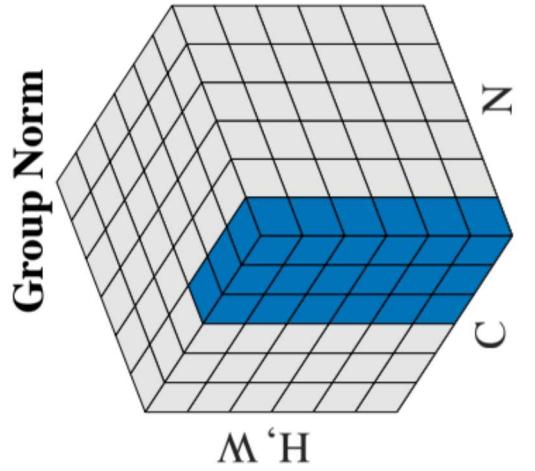
November 4, 2019

Recall Normalization Methods?



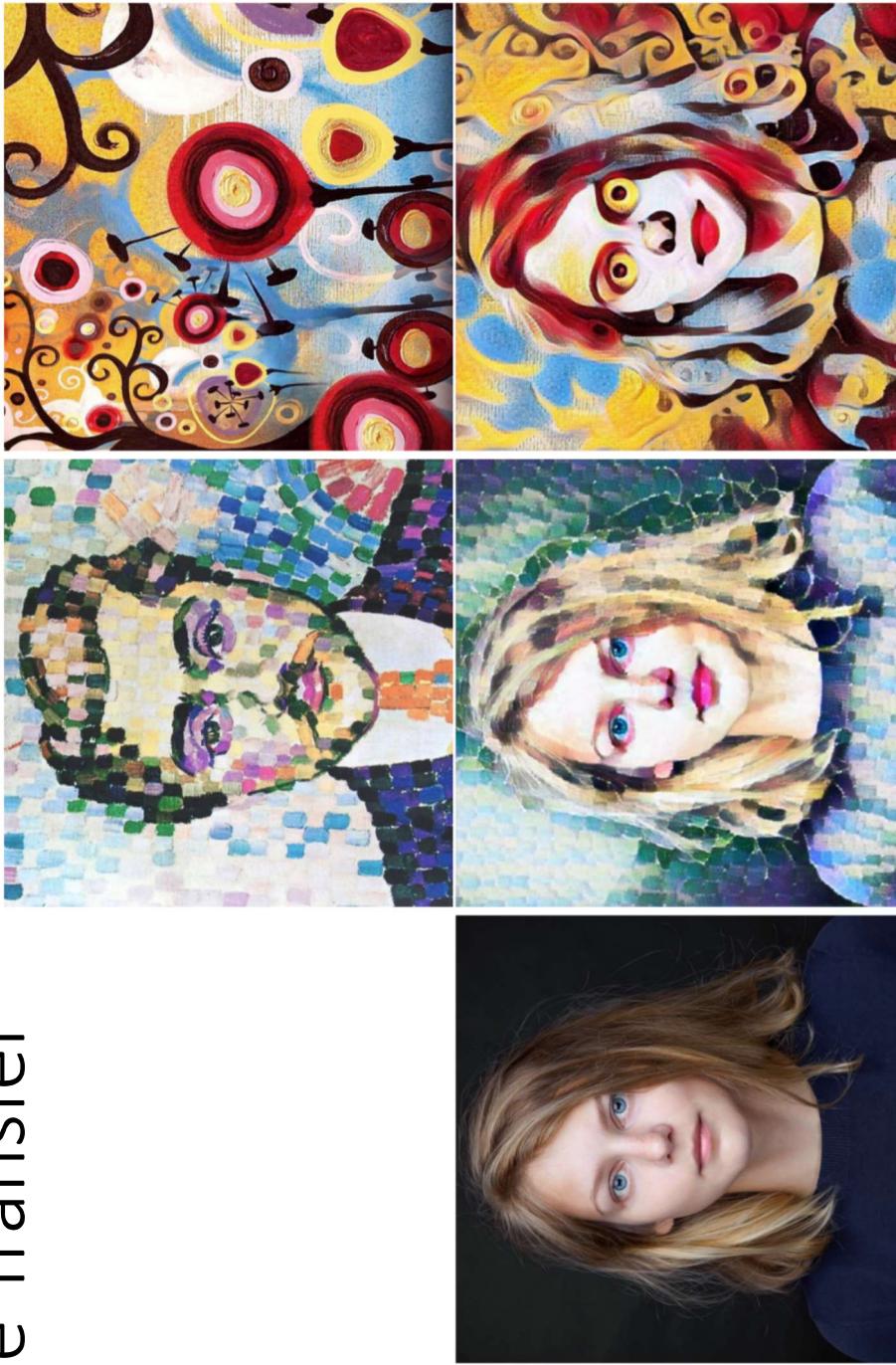
Recall Normalization Methods?

Instance Normalization was developed for style transfer!



Fast Neural Style Transfer

Replacing batch
normalization with
Instance Normalization
improves results



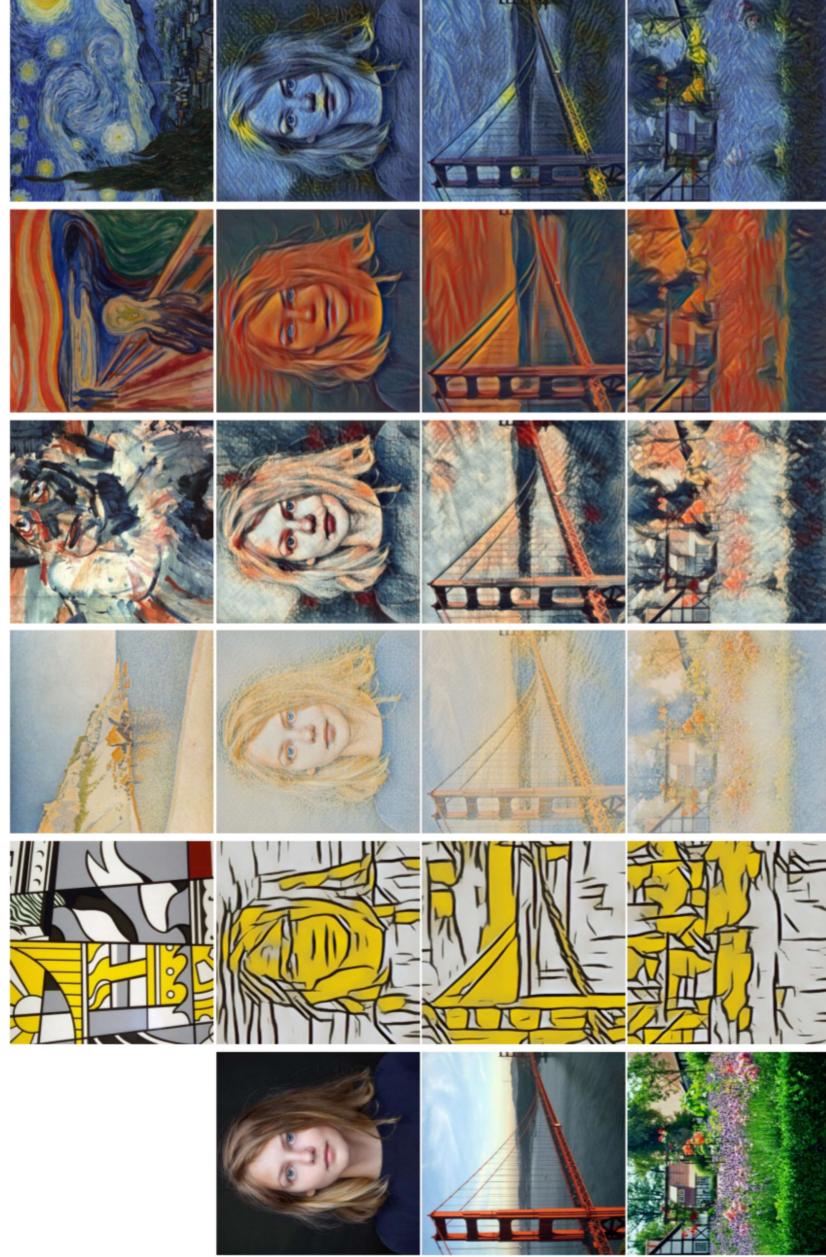
Ulyanov et al, "Texture Networks: Feed-forward Synthesis of Textures and Stylized Images", ICML 2016
Ulyanov et al, "Instance Normalization: The Missing Ingredient for Fast Stylization", arXiv 2016

Justin Johnson

Lecture 14 - 88

November 4, 2019

One Network, Many Styles



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

Justin Johnson

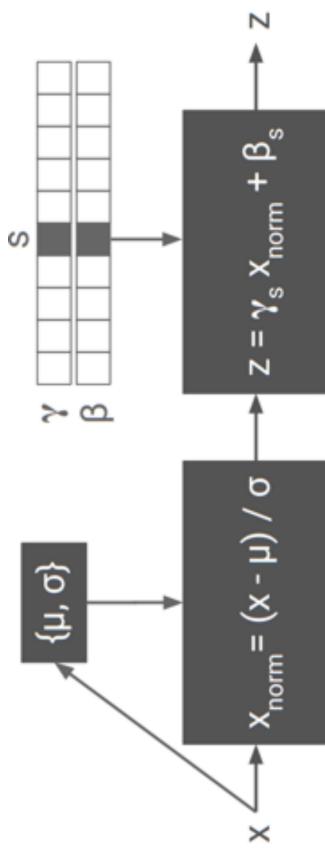
Lecture 14 - 89

November 4, 2019

One Network, Many Styles



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



Single network can blend styles after training

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

Summary

Many methods for understanding CNN representations

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

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

Fun: DeepDream, Style Transfer.

Next Time: Object Detection