Convolutional Neural Networks + Neural Style Transfer

Justin Johnson

2/1/2017

Outline

- Convolutional Neural Networks
  - Convolution
  - Pooling
  - Feature Visualization
- Neural Style Transfer
  - Feature Inversion
  - Texture Synthesis
  - Style Transfer

Convolutional Neural Networks: Deep Learning with Images

IMAGENET Large Scale Visual Recognition Challenge

Steel drum

The Image Classification Challenge:

1,000 object classes

1,431,167 images

Output:
Scale
T-shirt
<u>Steel drum</u>
Drumstick
Mud turtle

✔

Output:
Scale
T-shirt
Giant panda
Drumstick
Mud turtle

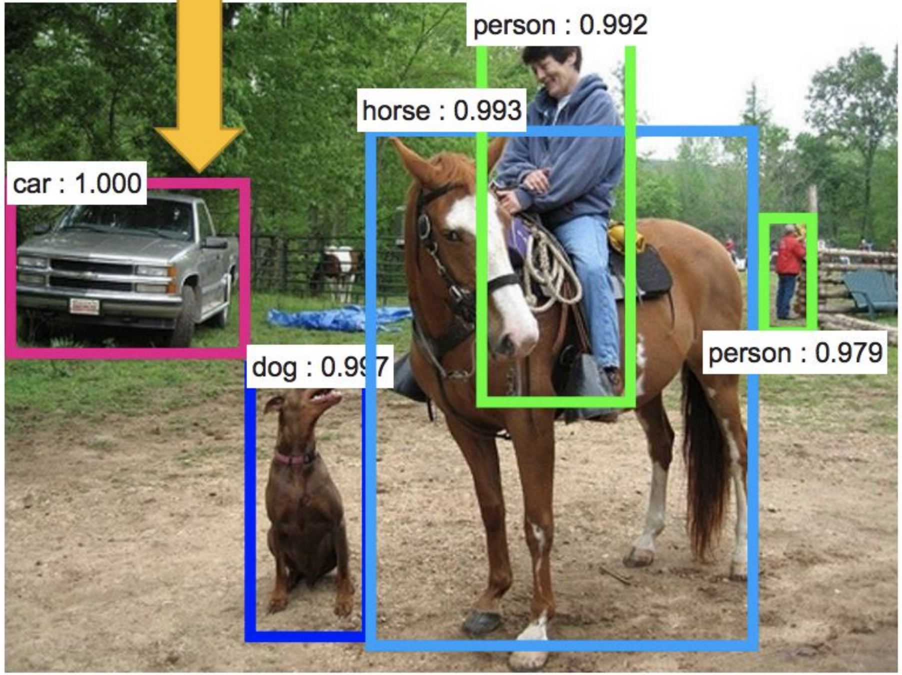
✗

Russakovsky et al. arXiv, 2014

Object Detection = What, and Where

Localization
Where?

Recognition
What?



Slide credit: Kaiming He, ICCV 2015

Object segmentation

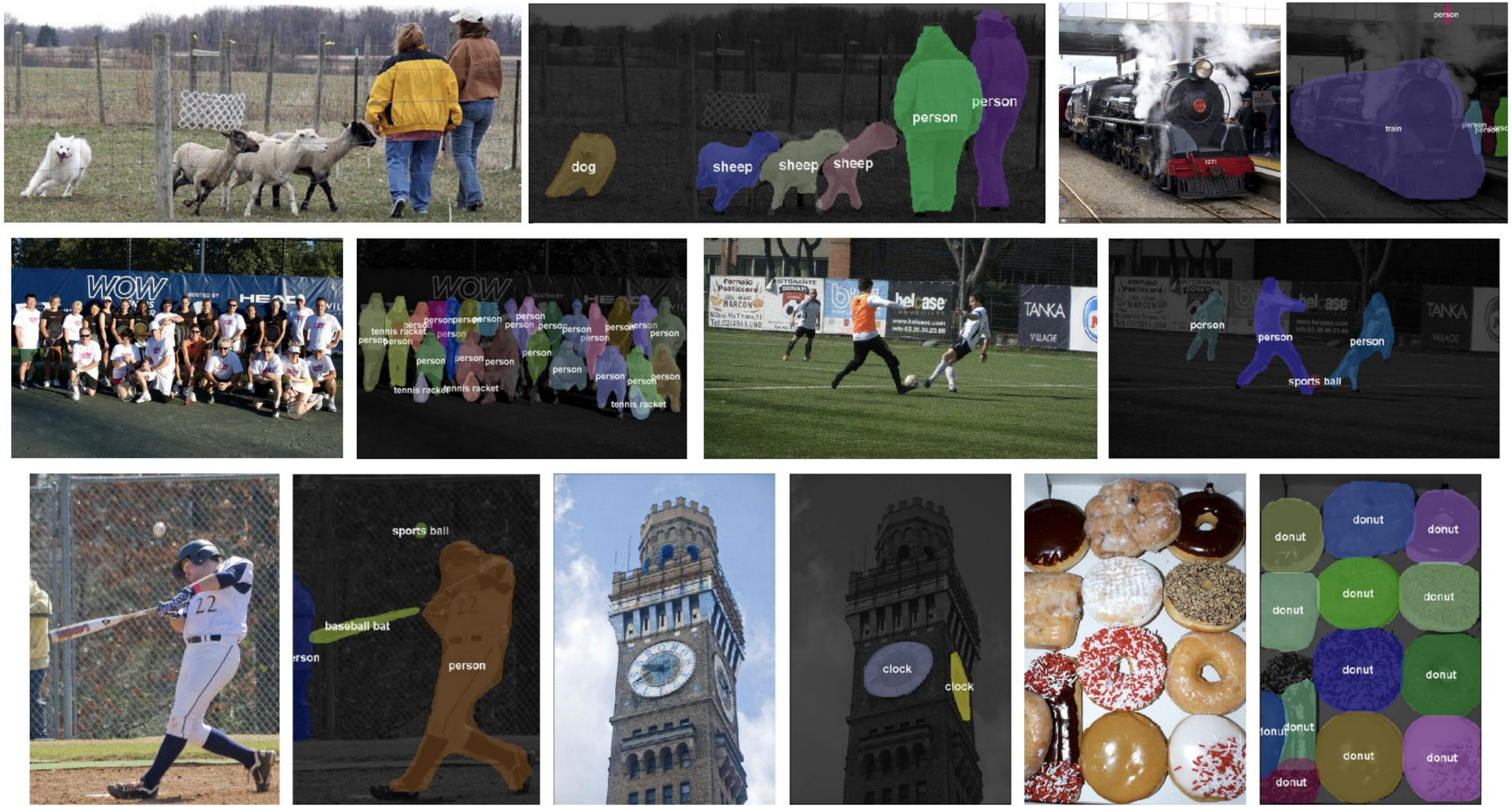
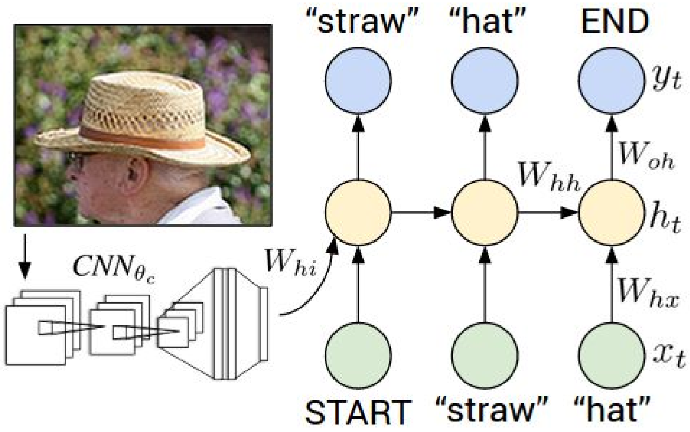


Figure credit: Dai, He, and Sun, “Instance-aware Semantic Segmentation via Multi-task Network Cascades”, CVPR 2016

Pose Estimation

Figure credit: Cao et al, “Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields”, arXiv 2016

Image Captioning




"man in black shirt is playing guitar."


"construction worker in orange safety vest is working on road."


"two young girls are playing with lego toy."


"boy is doing backflip on wakeboard."


"girl in pink dress is jumping in air."


"black and white dog jumps over bar."


"young girl in pink shirt is swinging on swing."


"man in blue wetsuit is surfing on wave."

Figure credit: Karpathy and Fei-Fei, “Deep Visual-Semantic Alignments for Generating Image Descriptions”, CVPR 2015

Dense Image Captioning

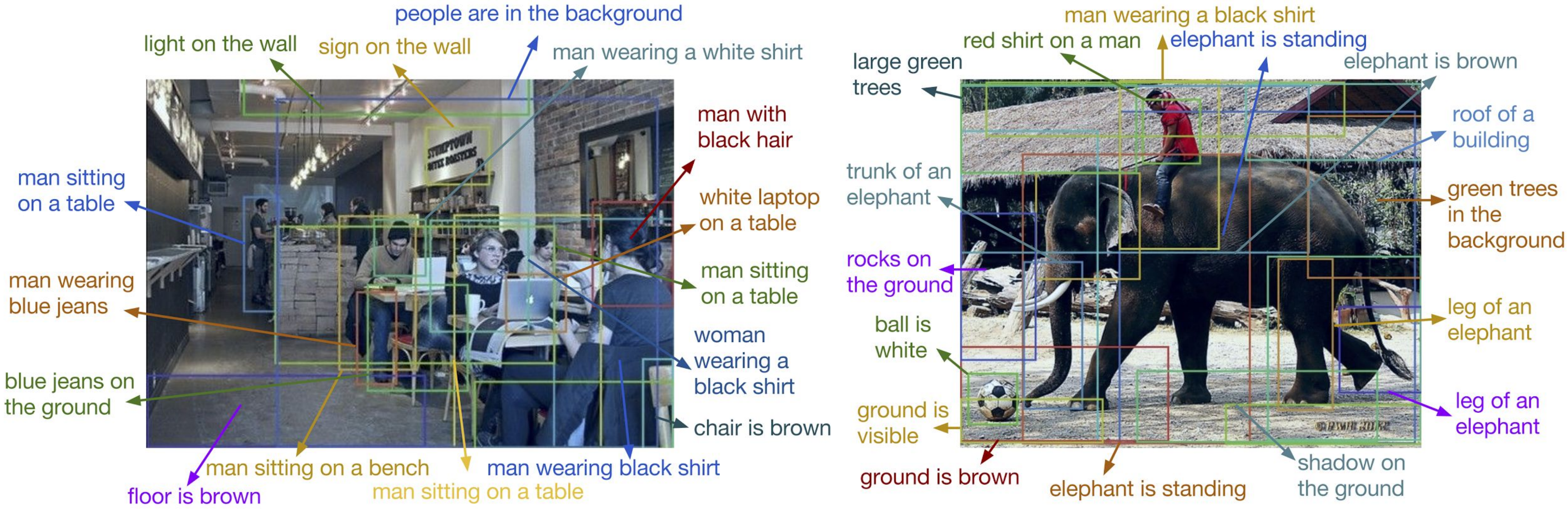


Figure credit: Johnson*, Karpathy*, and Fei-Fei, “DenseCap: Fully Convolutional Localization Networks for Dense Captioning”, CVPR 2016

Visual Question Answering

Figure credit: Agrawal et al, “VQA: Visual Question Answering”, ICCV 2015

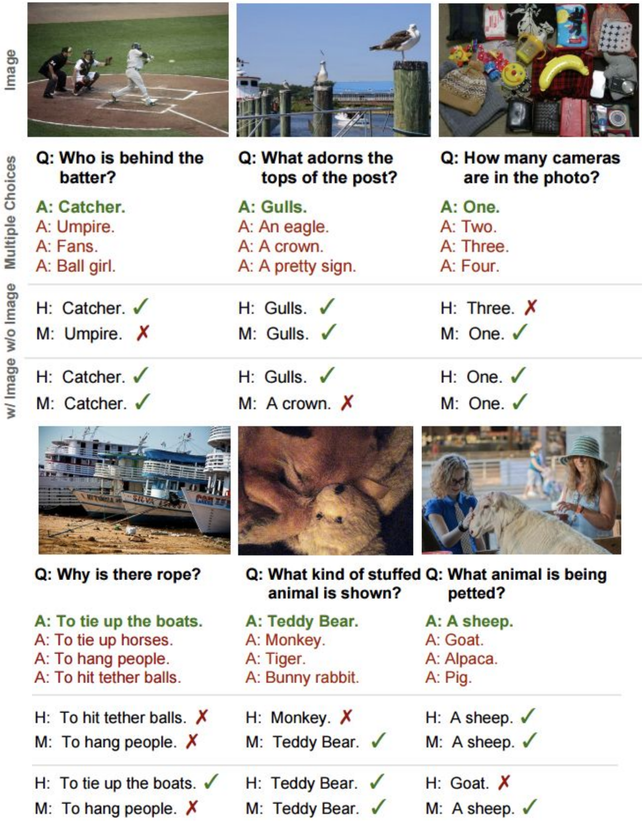


Figure credit: Zhu et al, “Visual7W: Grounded Question Answering in Images”, CVPR 2016

Image Super-Resolution

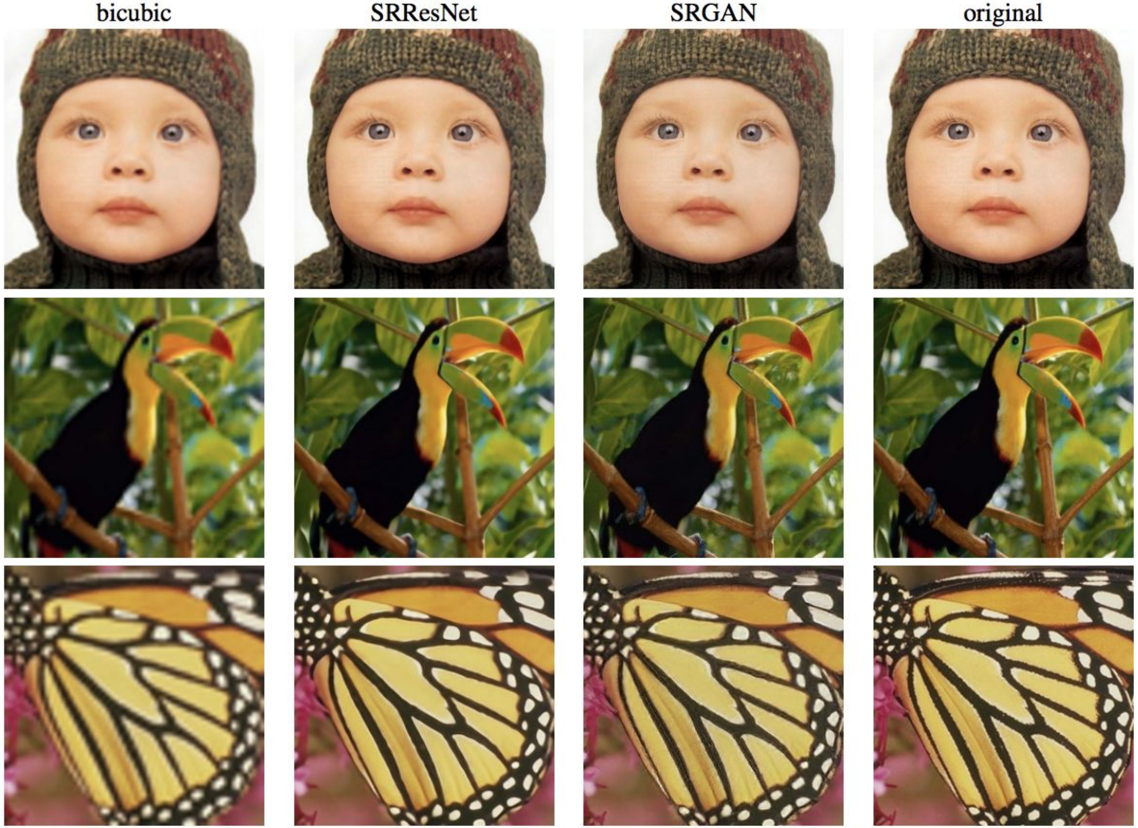


Figure credit: Ledig et al, “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network”, arXiv 2016

Generating Art

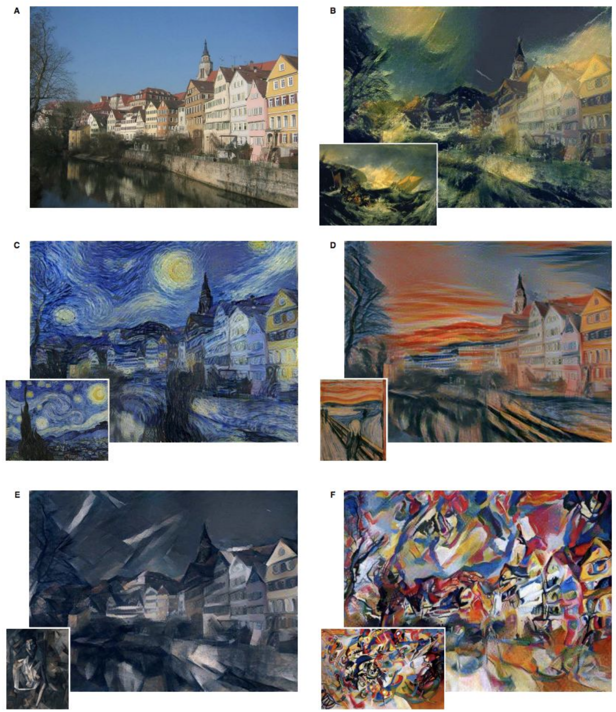


Figure credit: Gatys, Ecker, and Bethge, “Image Style Transfer using Convolutional Neural Networks”, CVPR 2016

Figure credit: Mordvintsev, Olah, and Tyka, “Inceptionism: Going Deeper into Neural Networks”, https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html

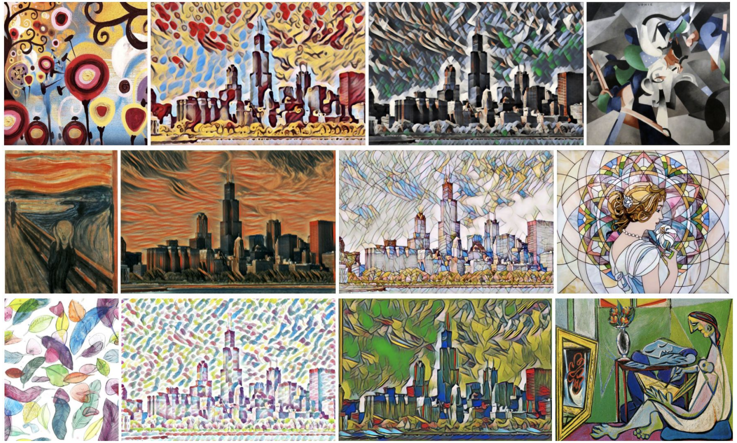
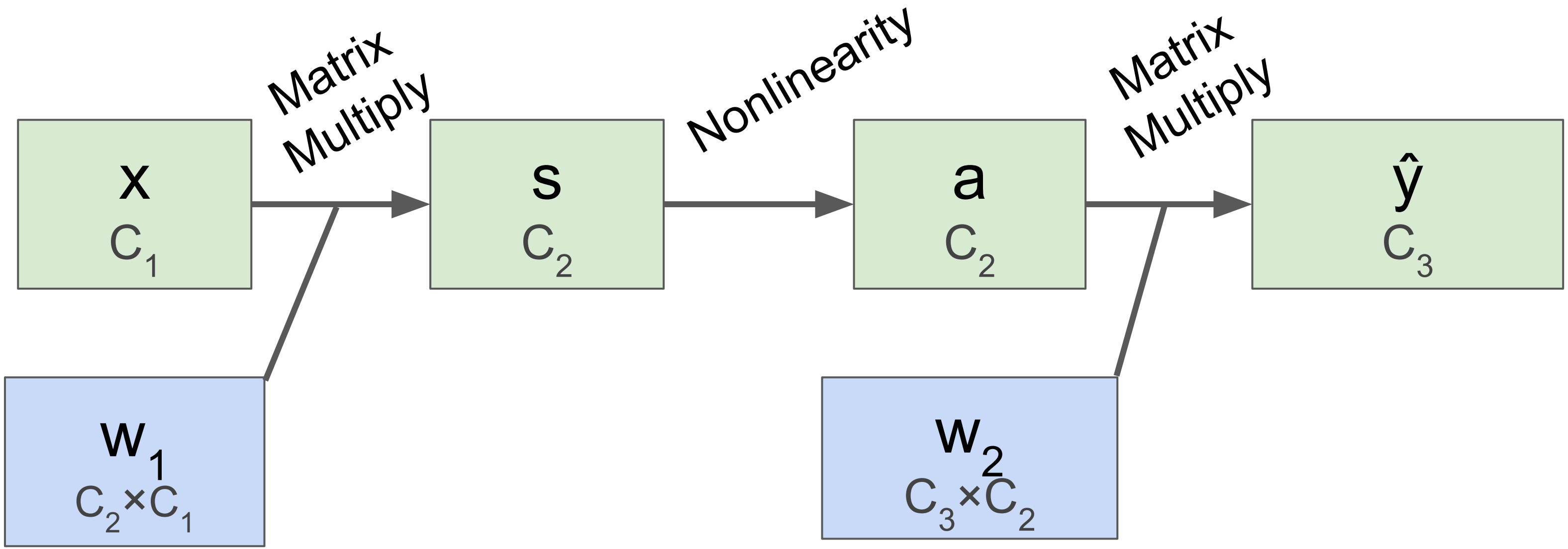


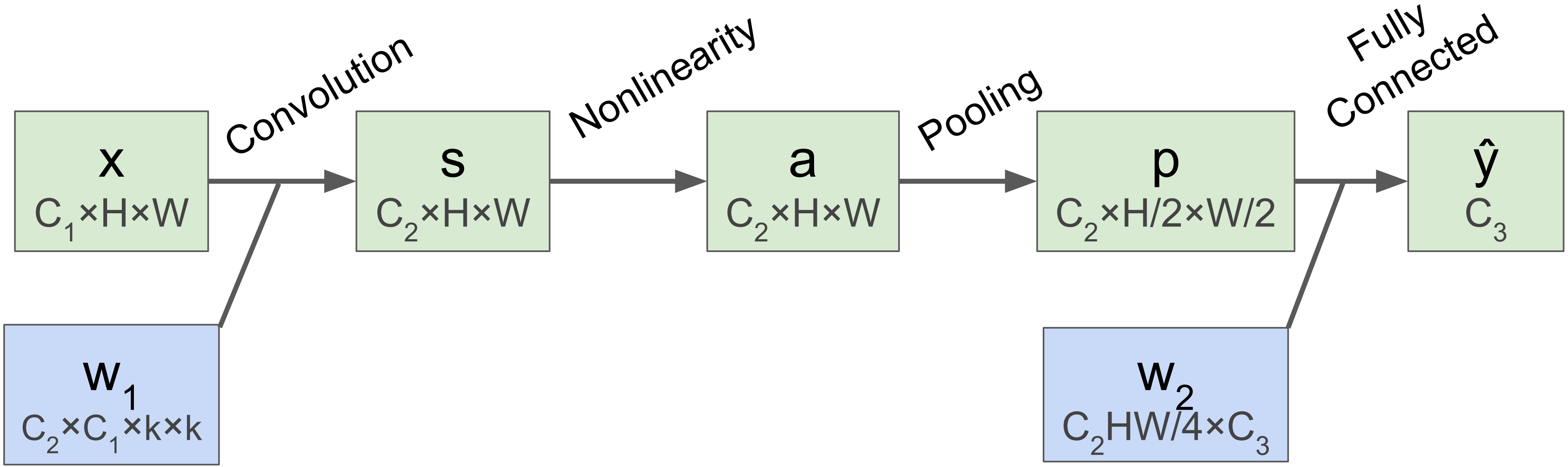
Figure credit: Johnson, Alahi, and Fei-Fei: “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016, https://github.com/jcjohnson/fast-neural-style

What is a Convolutional Neural Net?

Fully-Connected Neural Network

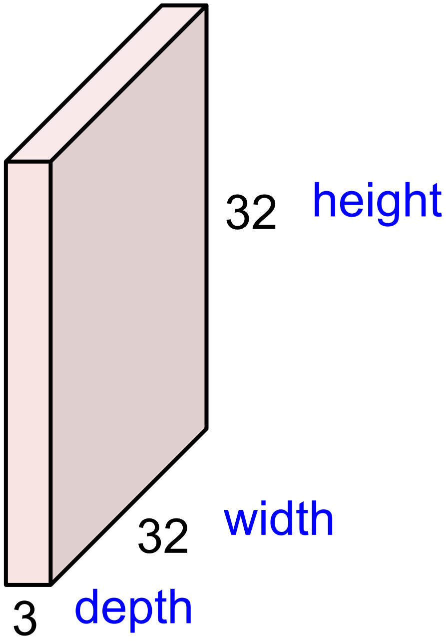


Convolutional Neural Network



Convolution Layer

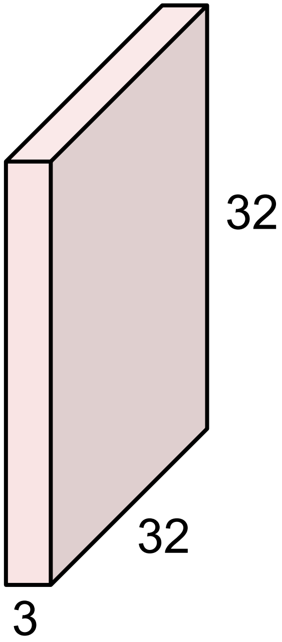
32x32x3 image



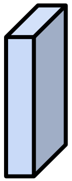
Slide credit: CS231n Lecture 7

Convolution Layer

32x32x3 image



5x5x3 filter



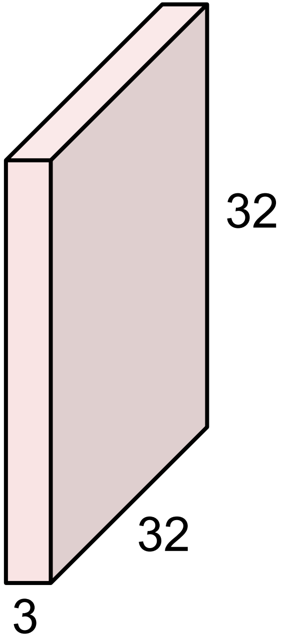
Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Slide credit: CS231n Lecture 7

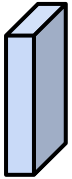
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



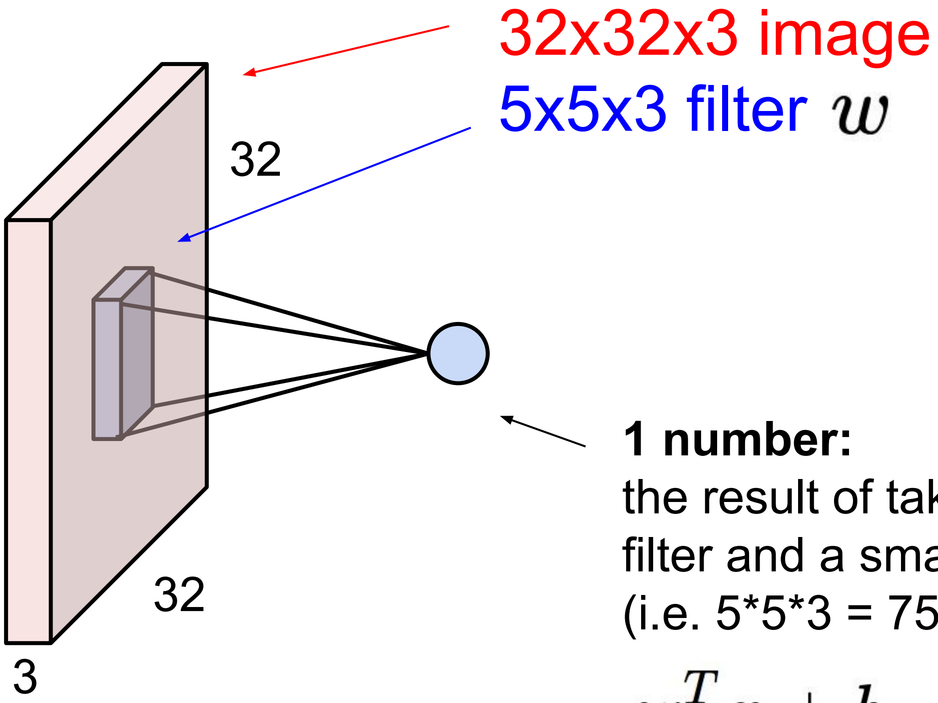
5x5x3 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Slide credit: CS231n Lecture 7

Convolution Layer

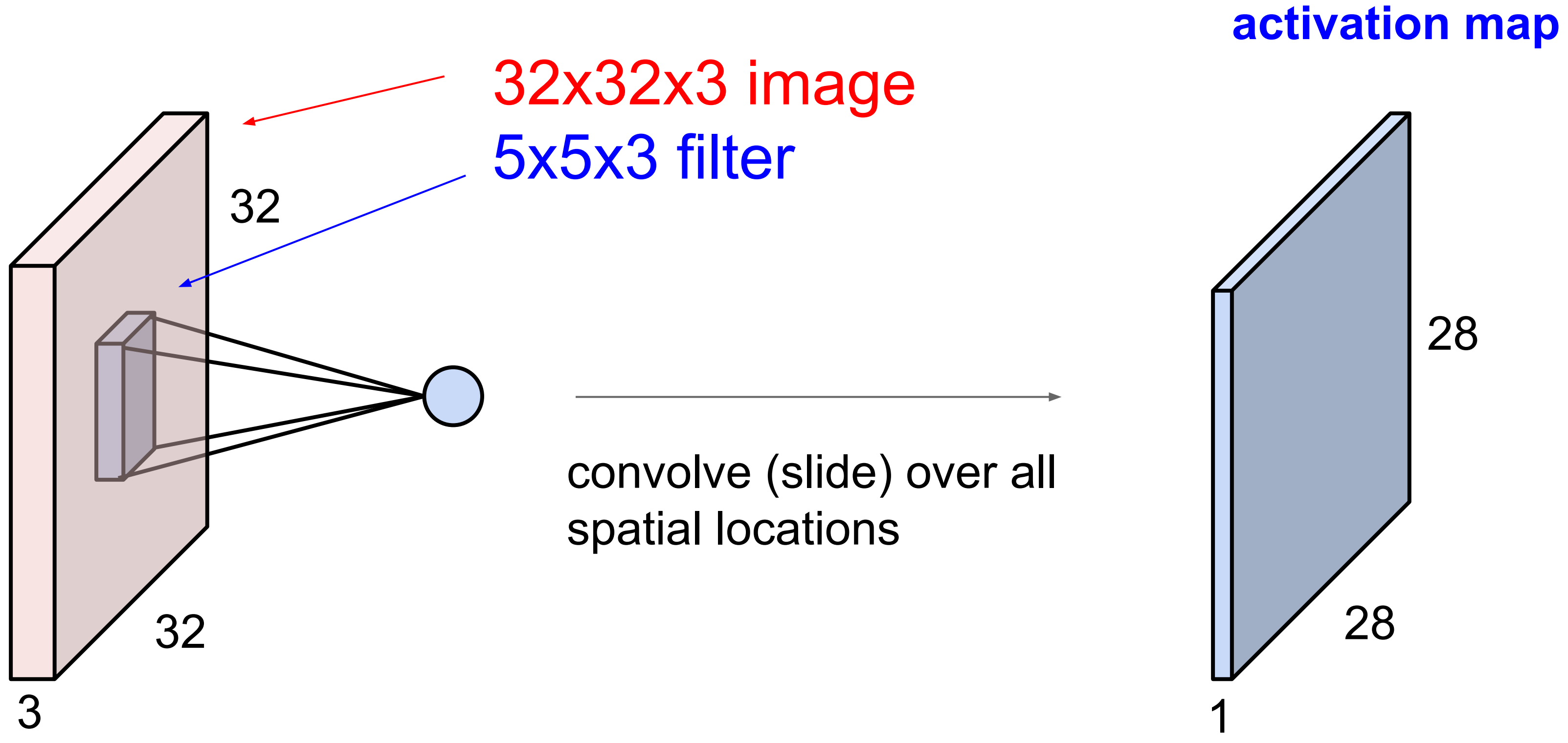


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Slide credit: CS231n Lecture 7

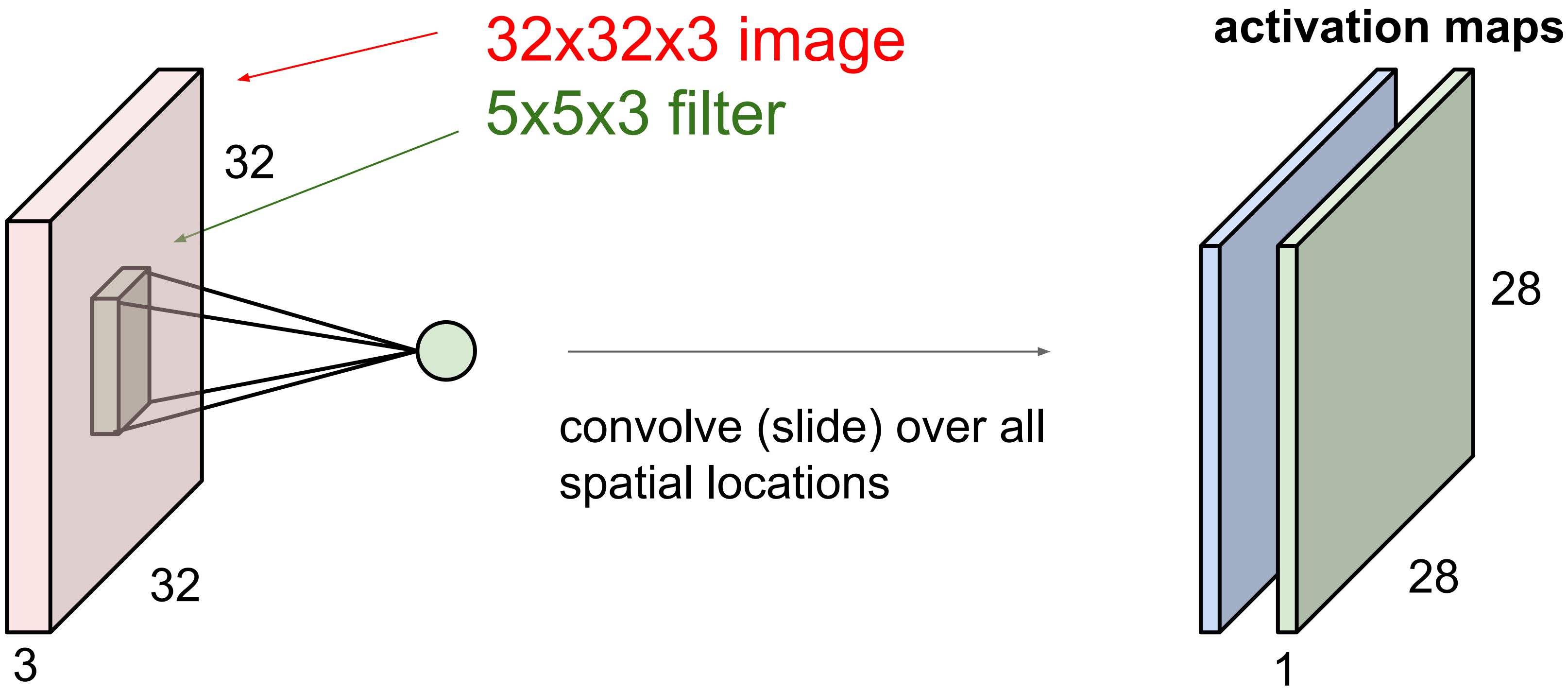
Convolution Layer



Slide credit: CS231n Lecture 7

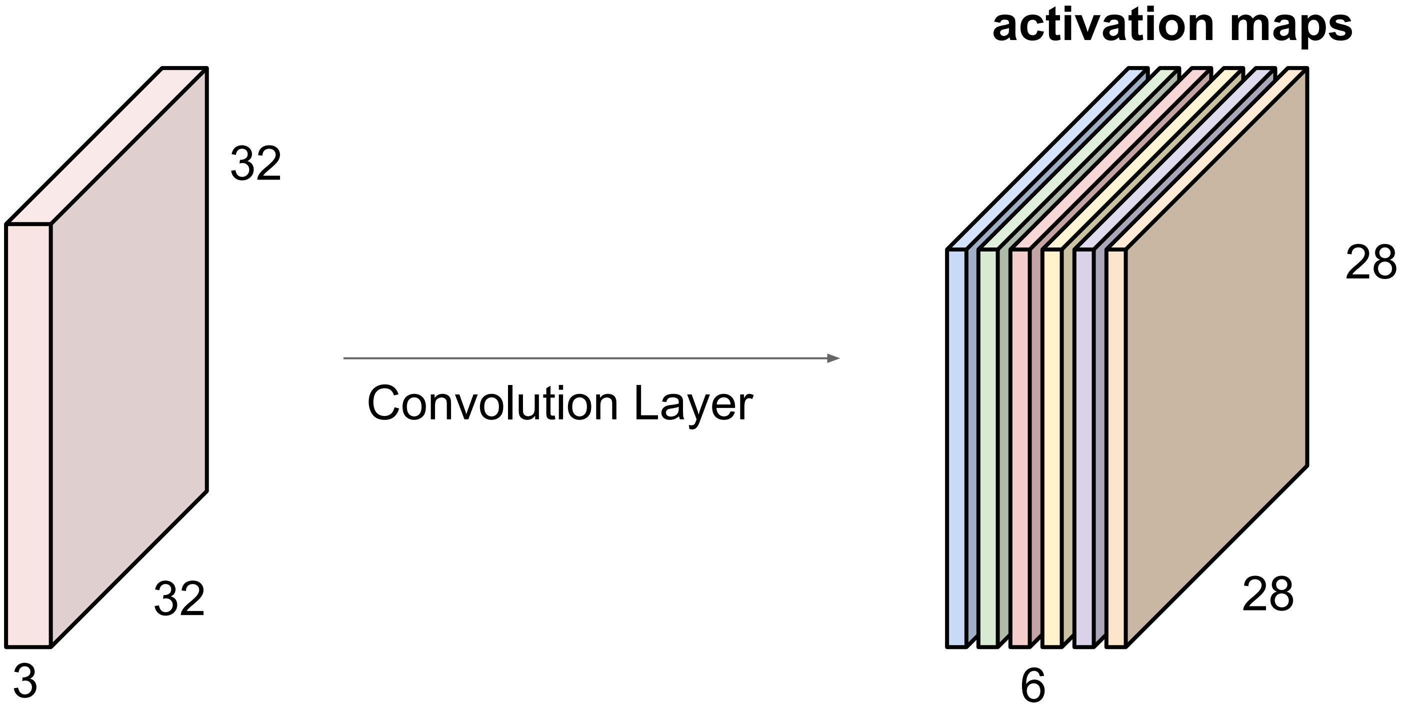
Convolution Layer

consider a second, green filter



Slide credit: CS231n Lecture 7

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

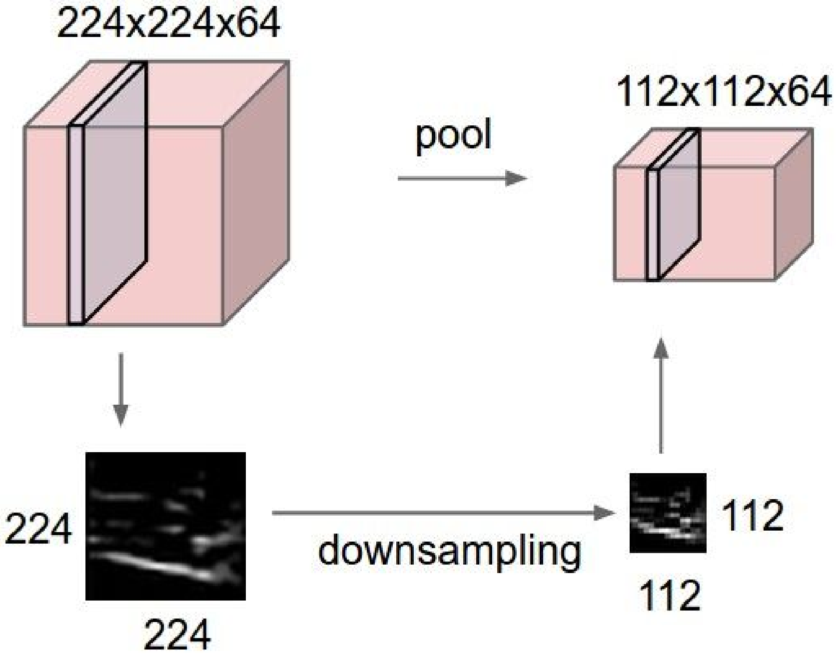


We stack these up to get a "new image" of size 28x28x6!

Slide credit: CS231n Lecture 7

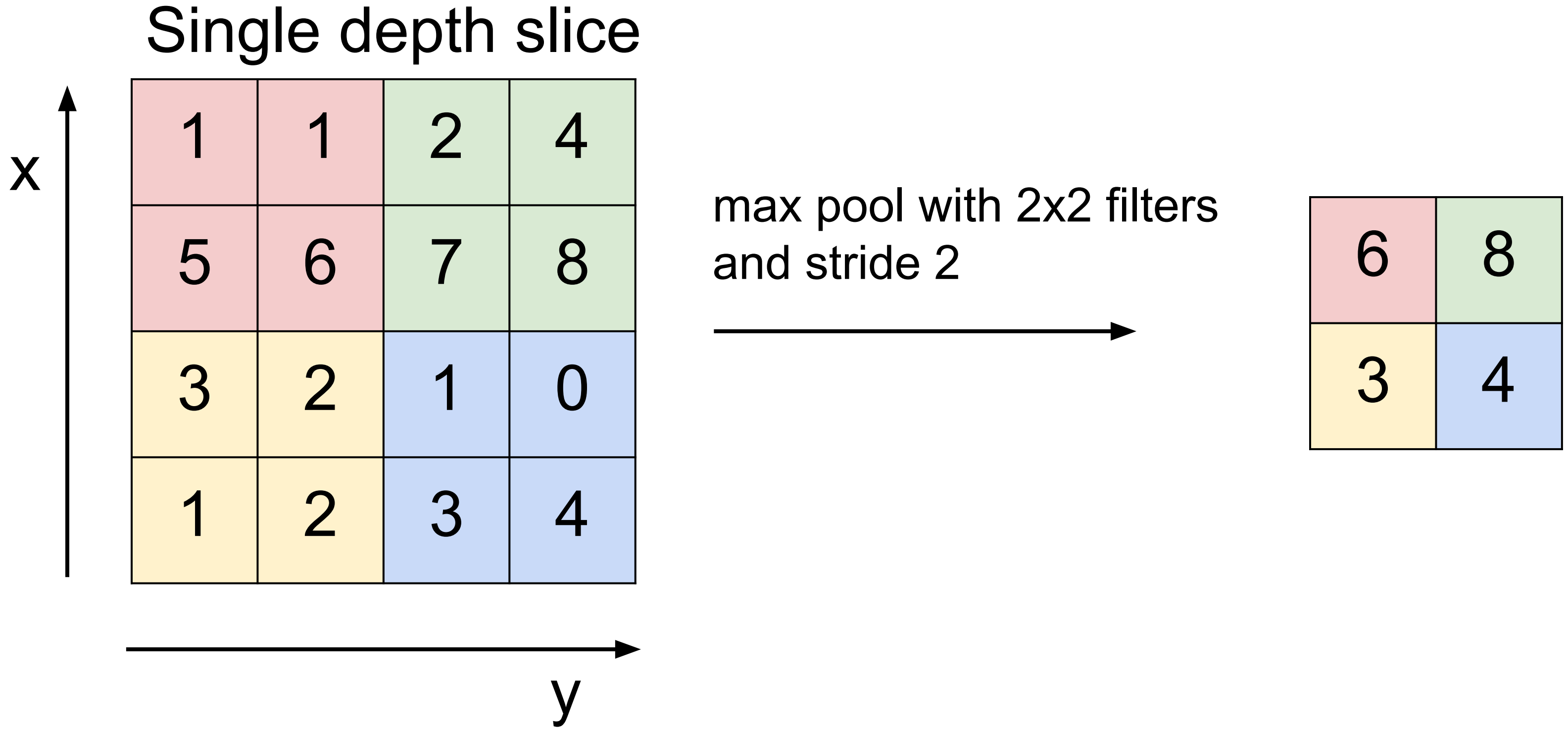
Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



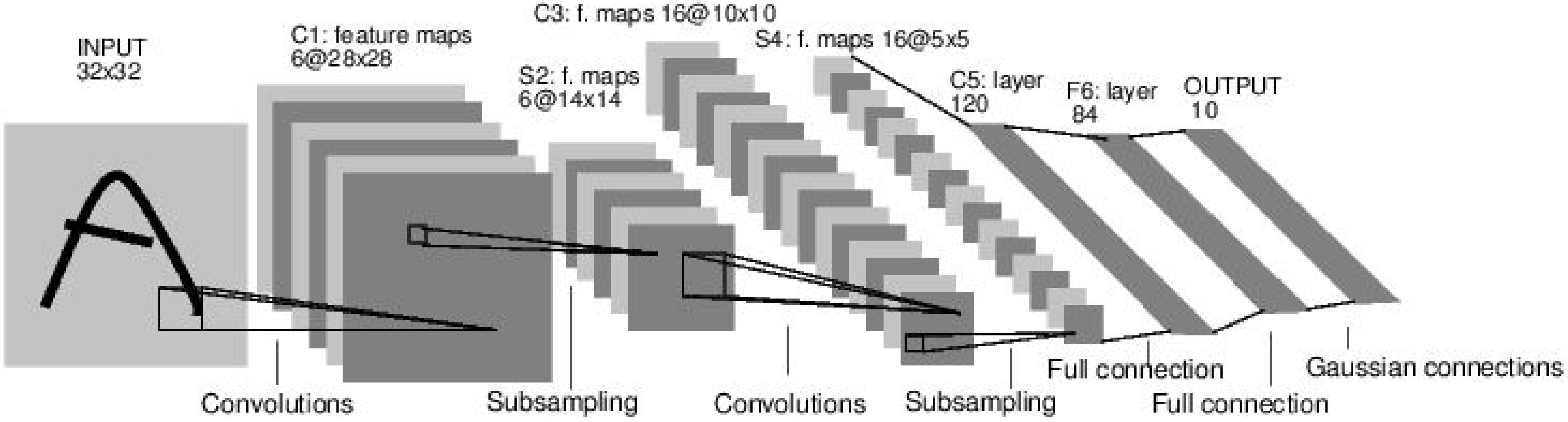
Slide credit: CS231n Lecture 7

MAX POOLING


Single depth slice
x
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
y
max pool with 2x2 filters
and stride 2
6 8
3 4

Case Study: LeNet-5

[LeCun et al., 1998]

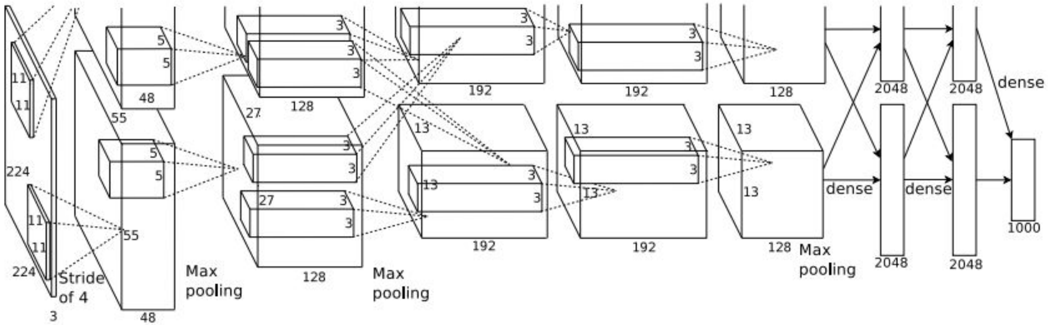


Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Slide credit: CS231n Lecture 7

Case Study: AlexNet

[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Slide credit: CS231n Lecture 7

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

best model

11.2% top 5 error in ILSVRC 2013
->
7.3% top 5 error

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

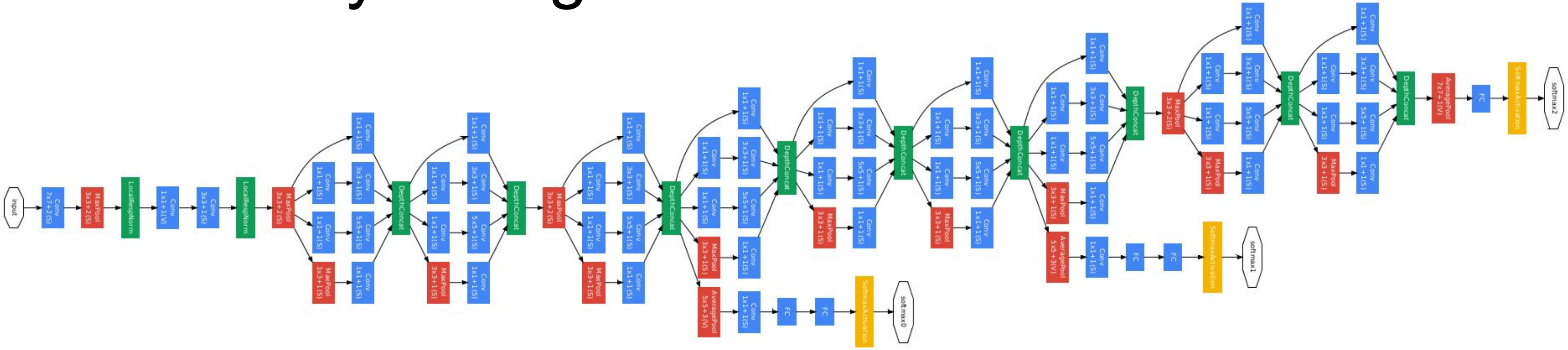
Table 2: **Number of parameters** (in millions).

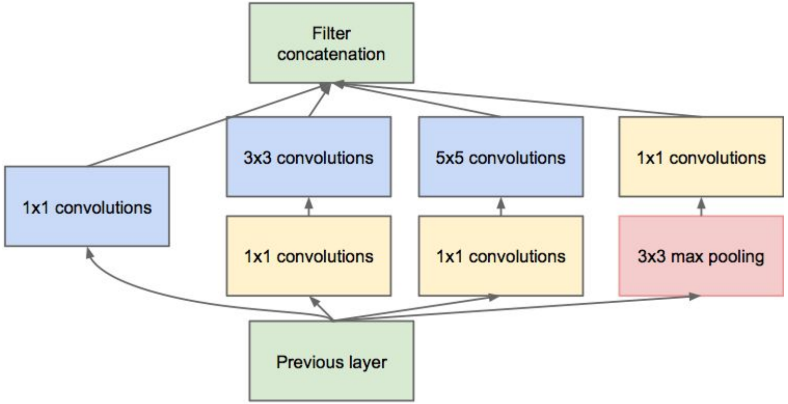
Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Slide credit: CS231n Lecture 7

Case Study: GoogLeNet

[Szegedy et al., 2014]





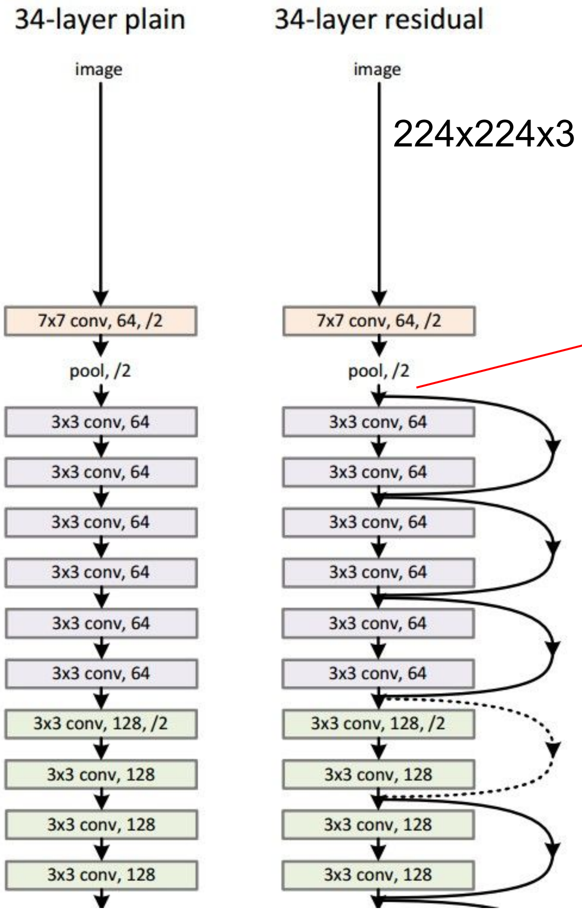
Inception module

ILSVRC 2014 winner (6.7% top 5 error)

Slide credit: CS231n Lecture 7

Case Study: ResNet

[He et al., 2015]

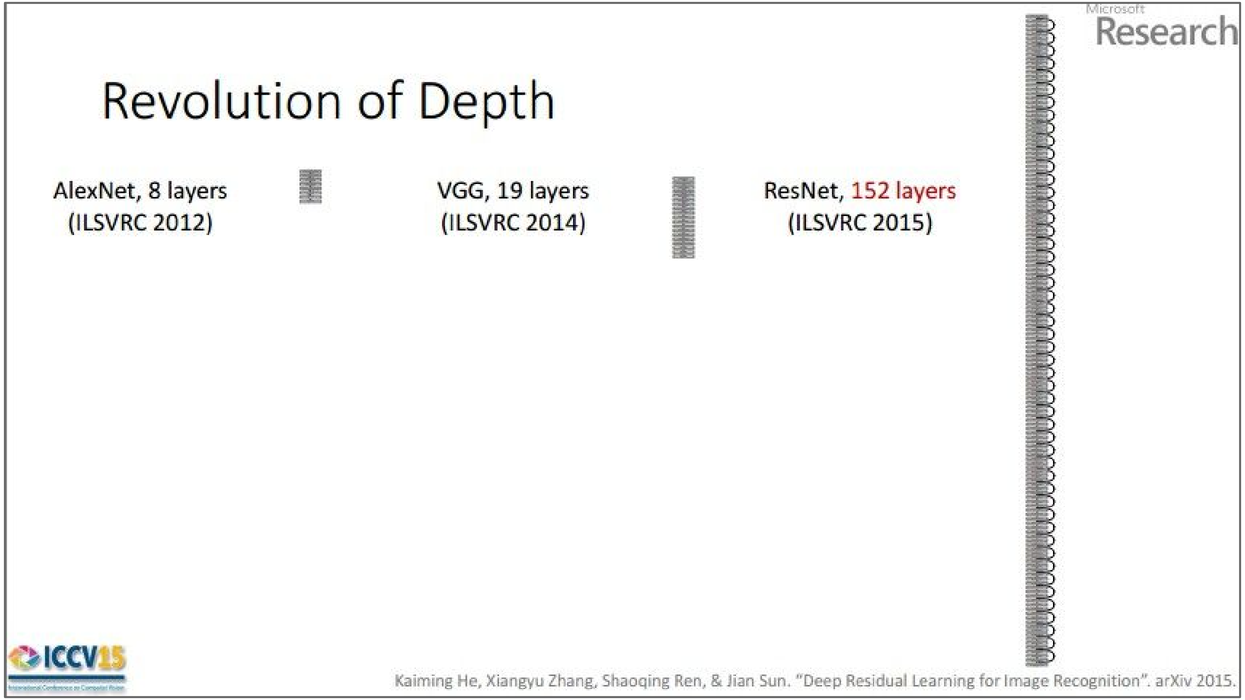


spatial dimension
only 56x56!

Slide credit: CS231n Lecture 7

Case Study: ResNet
[He et al., 2015]

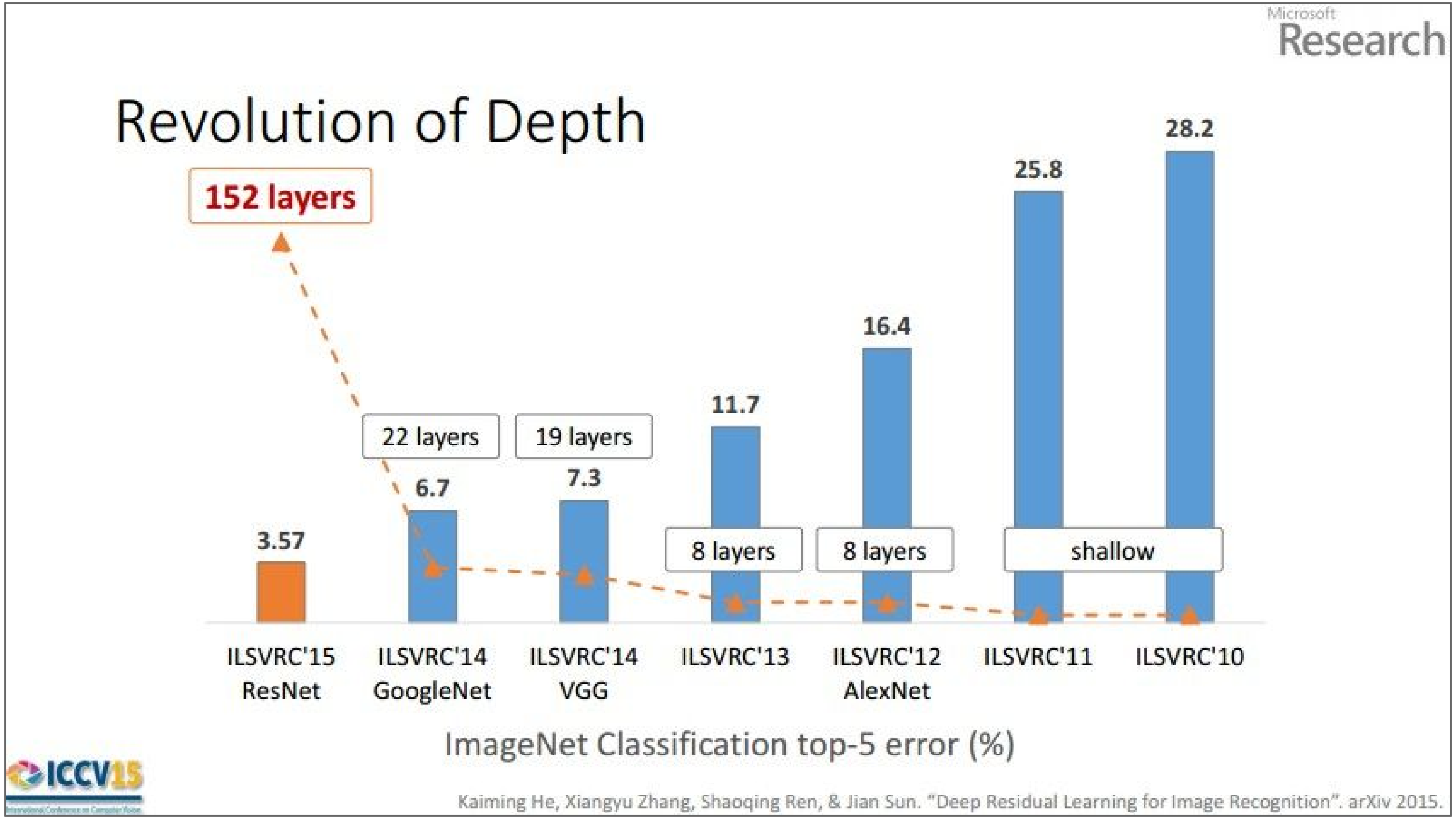
ILSVRC 2015 winner (3.6% top 5 error)



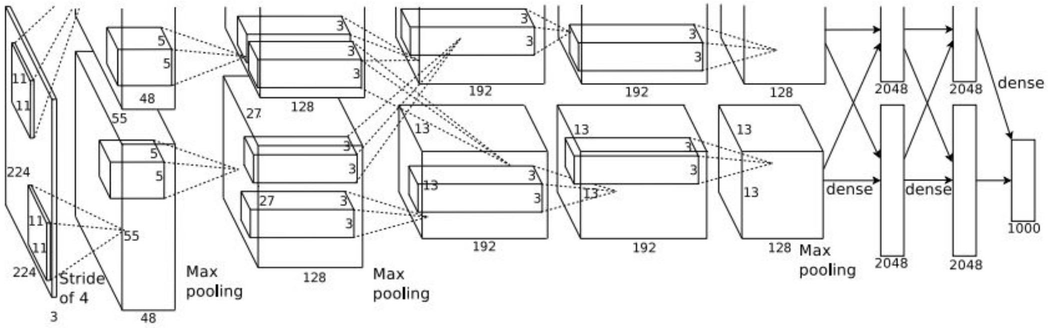
(slide from Kaiming He's ICCV 2015 presentation)

2-3 weeks of training on 8 GPU machine

at runtime: faster than a VGGNet! (even though it has 8x more layers)

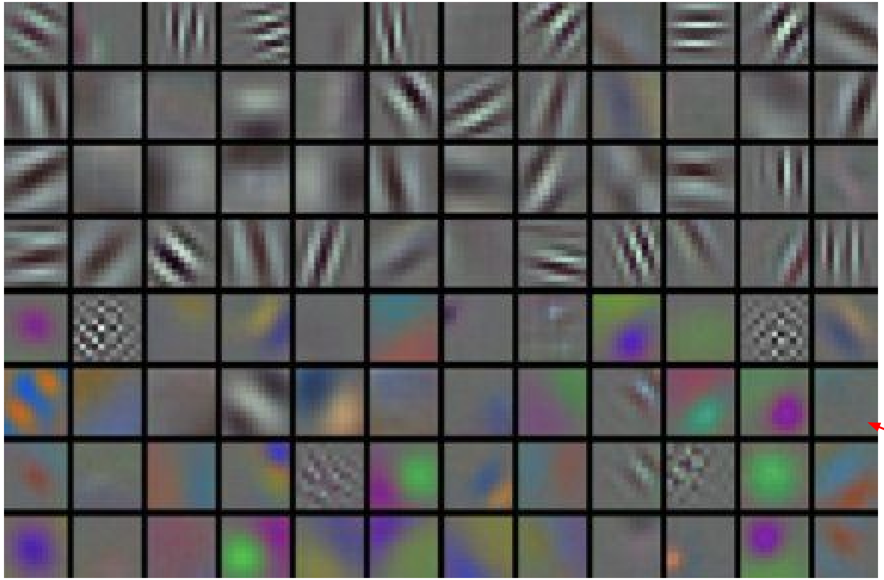


(slide from Kaiming He’s ICCV 2015 presentation)

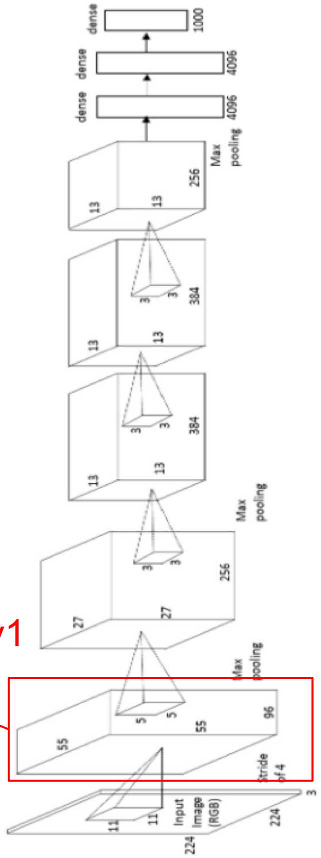


Visualizing ConvNet Features

Visualizing CNN features: Look at filters



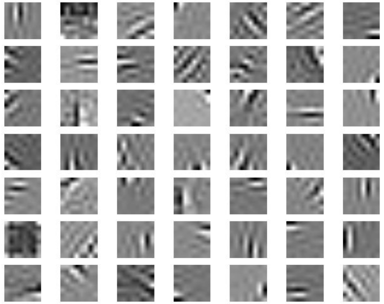
AlexNet

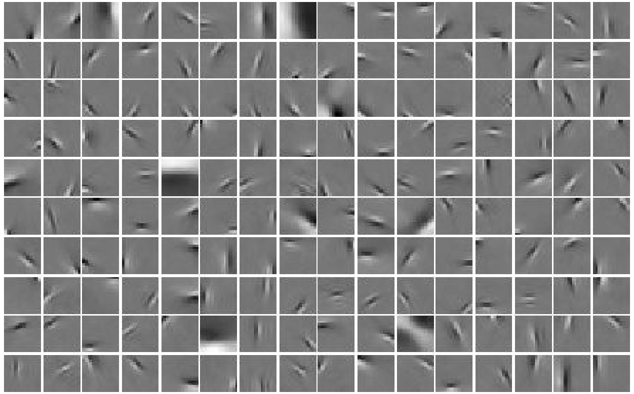


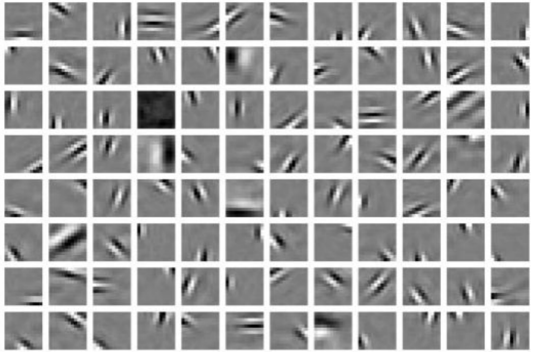
conv1

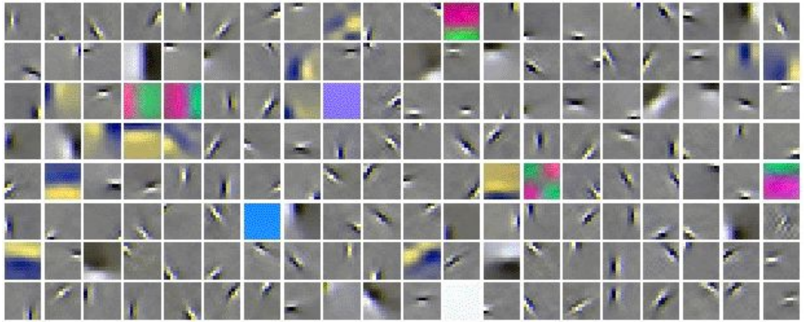
Slide credit: CS231n Lecture 9

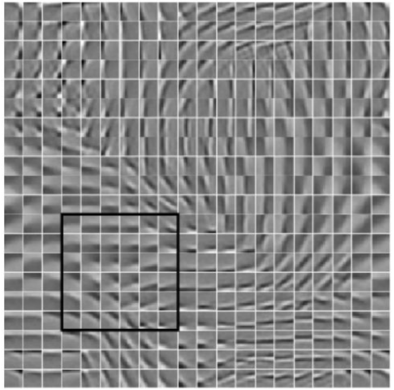
Many networks learn similar filters

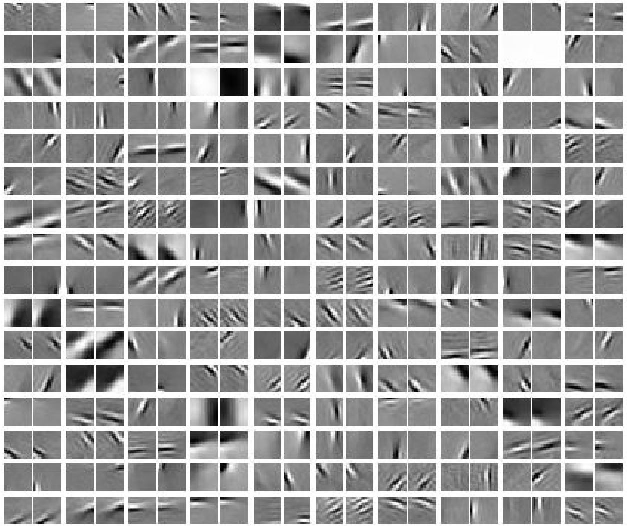






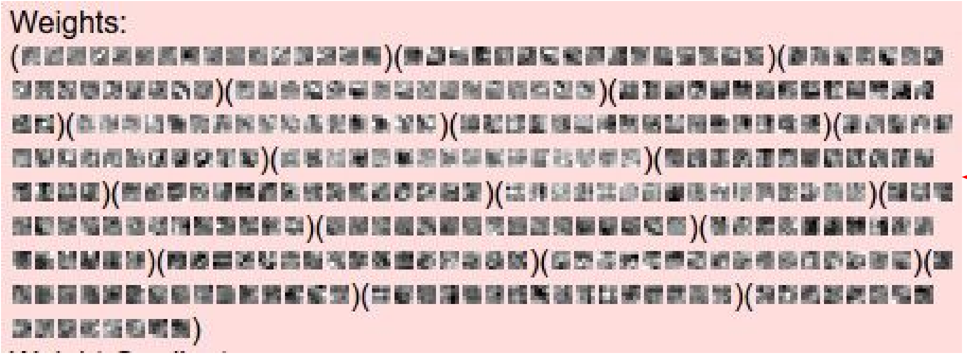


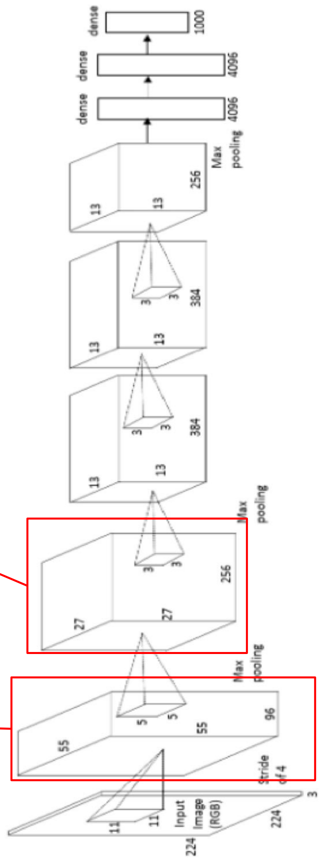




Slide credit: CS231n Lecture 9

Visualizing CNN features: Look at filters





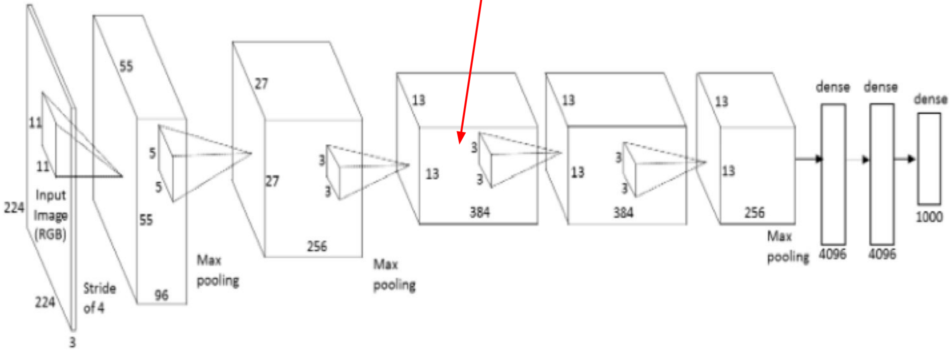
Filters from higher layers don't make much sense

Image credit: CS231n Lecture 9;
Filters from ConvNetJS CIFAR-10 model

Visualizing CNN features: (Guided) Backprop

Choose an image

Choose a layer and a neuron in a CNN

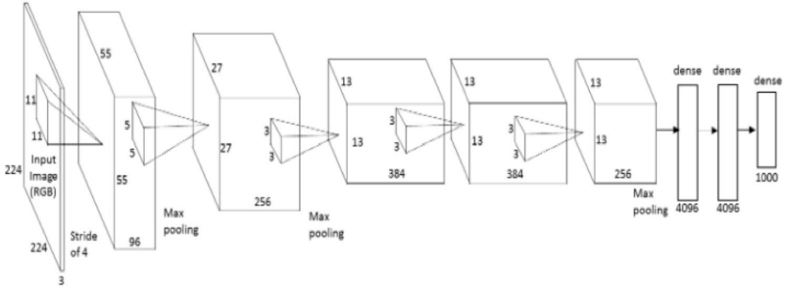


Question:
How does the chosen neuron respond to the image?

Slide credit: CS231n Lecture 9

Visualizing CNN features: (Guided) Backprop

1. Feed image into net



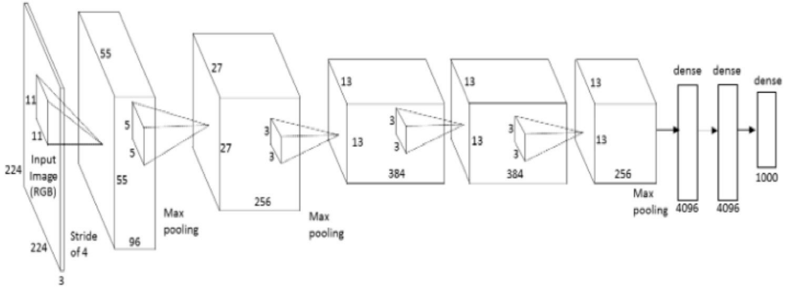
Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

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

Slide credit: CS231n Lecture 9

Visualizing CNN features: (Guided) Backprop

1. Feed image into net



2. Set gradient of chosen layer to all zero, except 1 for the chosen neuron

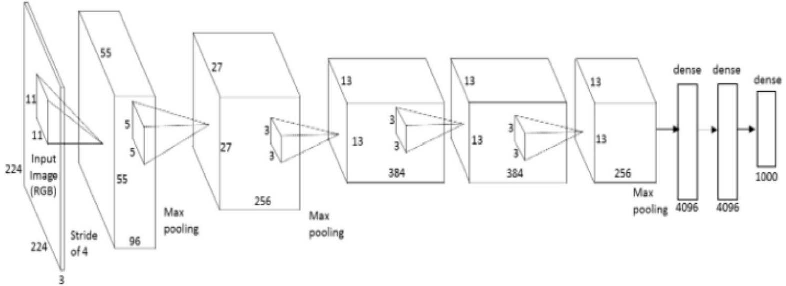
Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

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

Slide credit: CS231n Lecture 9

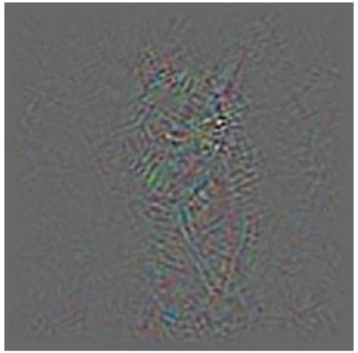
Visualizing CNN features: (Guided) Backprop

1. Feed image into net



2. Set gradient of chosen layer to all zero, except 1 for the chosen neuron

3. Backprop to image:



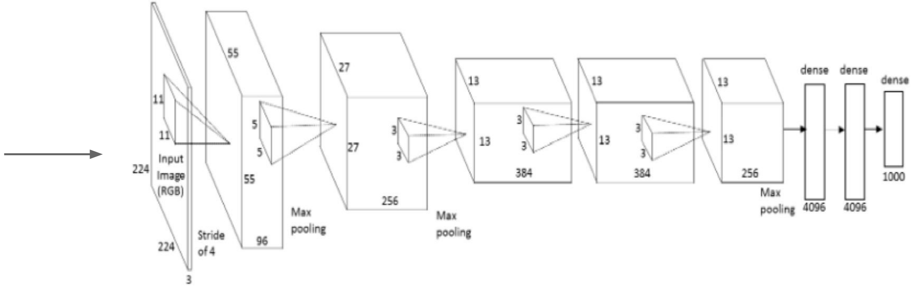
Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

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

Slide credit: CS231n Lecture 9

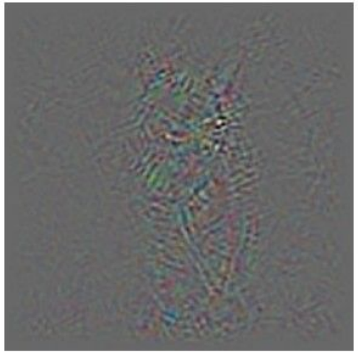
Visualizing CNN features: (Guided) Backprop

1. Feed image into net



2. Set gradient of chosen layer to all zero, except 1 for the chosen neuron

3. Backprop to image:



Guided backpropagation: instead

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

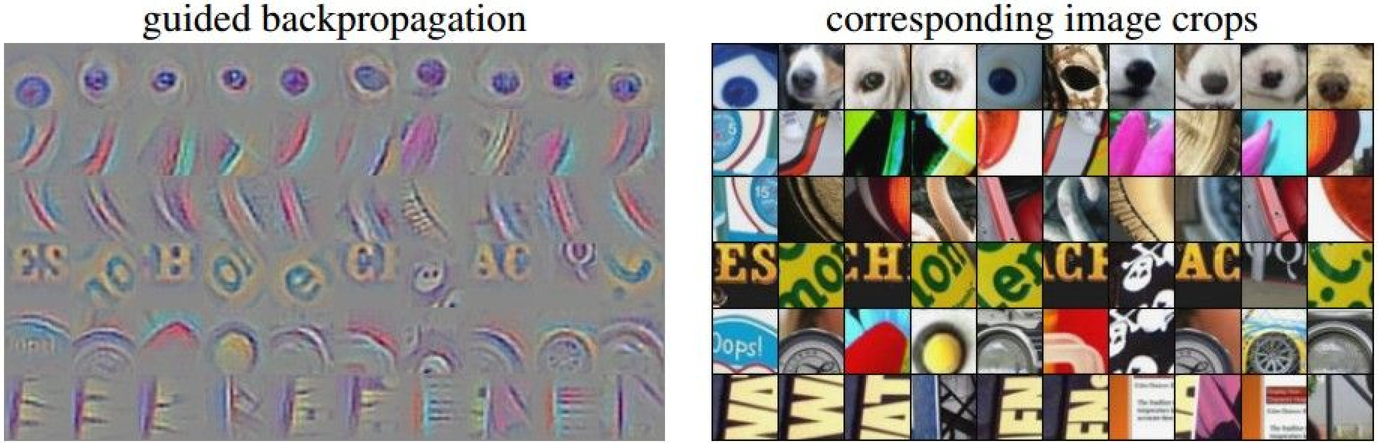
Dosovitskiy et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015

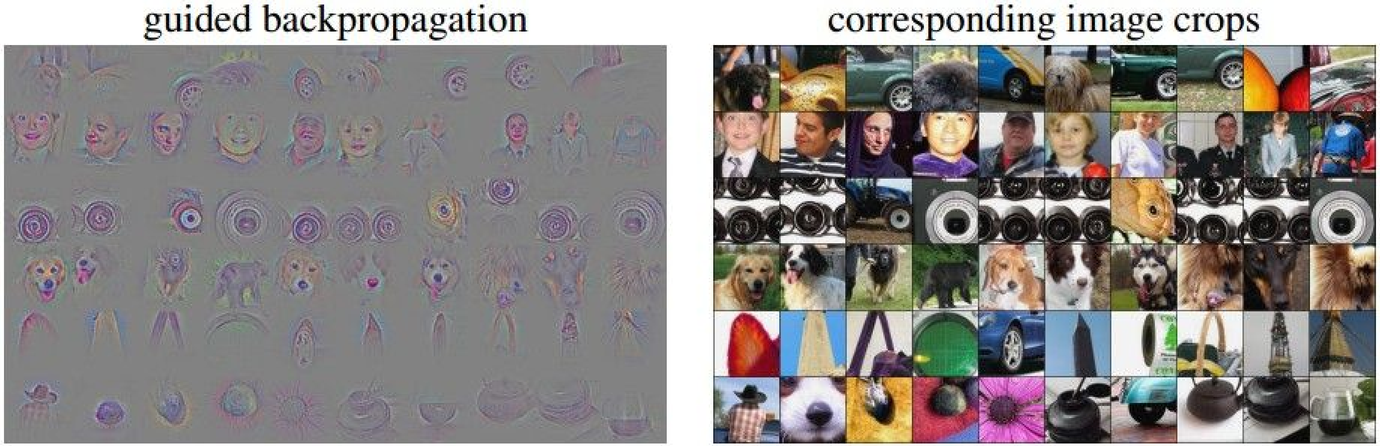
Slide credit: CS231n Lecture 9

Visualization of patterns learned by the layer **conv6** (top) and layer **conv9** (bottom) of the network trained on ImageNet.

Each row corresponds to one filter.

The visualization using “guided backpropagation” is based on the top 10 image patches activating this filter taken from the ImageNet dataset.





Dosovitskiy 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

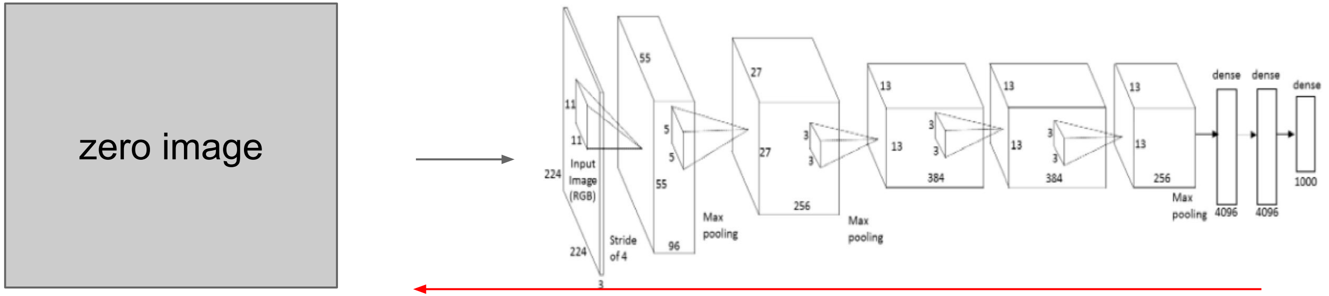
$$I^* = \arg\max_I \underbrace{\boxed{f(I)}}_{\text{Neuron value}} + \underbrace{\boxed{R(I)}}_{\text{Natural image regularizer}}$$

Visualizing CNN features: Gradient Ascent

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

score for class c (before Softmax)

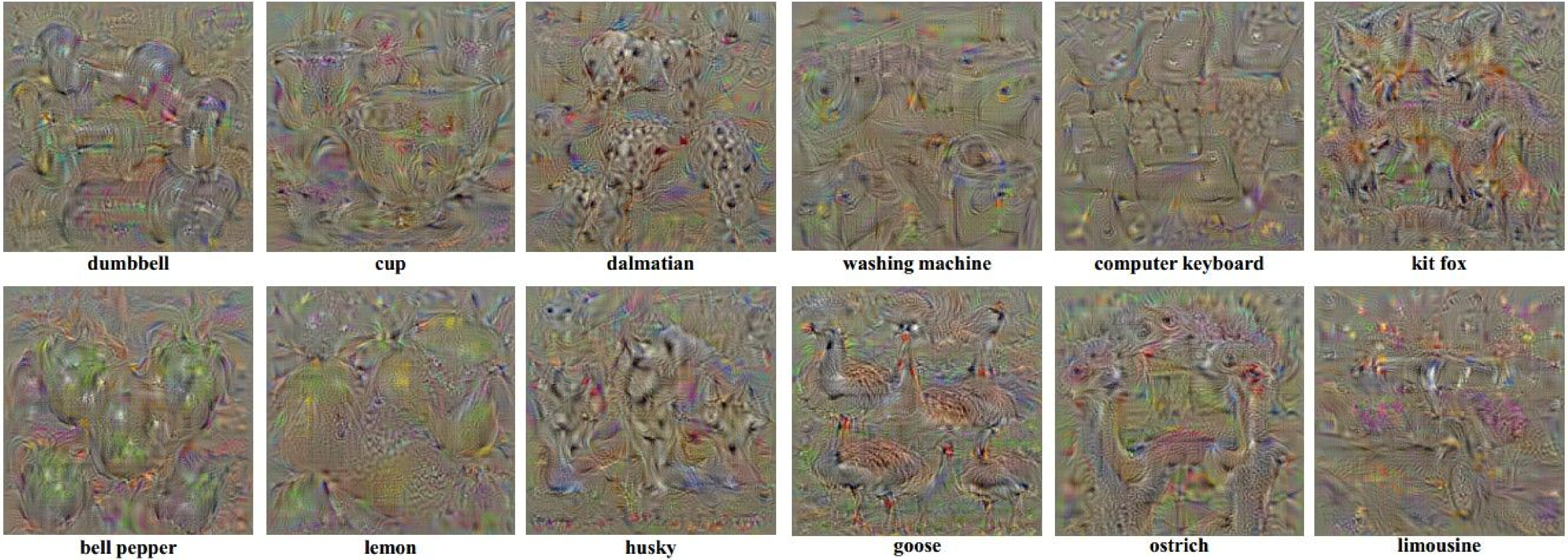
1. Initialize image to zeros



Repeat:
2. Forward image to compute current scores
3. Set gradient of scores to be 1 for target class, 0 for others
4. Backprop to get gradient on image
5. Make a small update to the image

Simonyan et al, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”, ICLR Workshop 2014

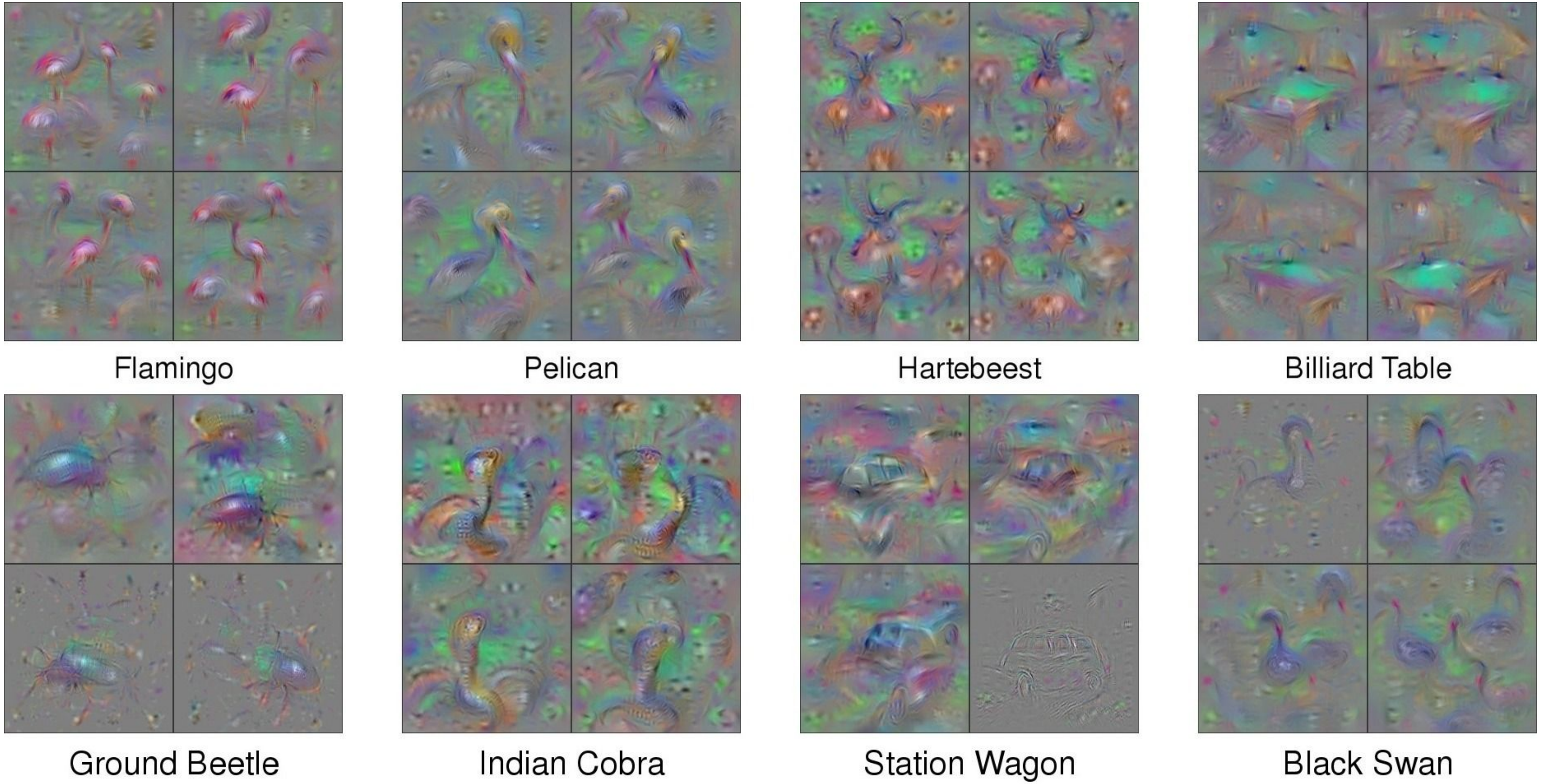
Visualizing CNN features: Gradient Ascent



Simonyan et al, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”, ICLR Workshop 2014

Visualizing CNN features: Gradient Ascent

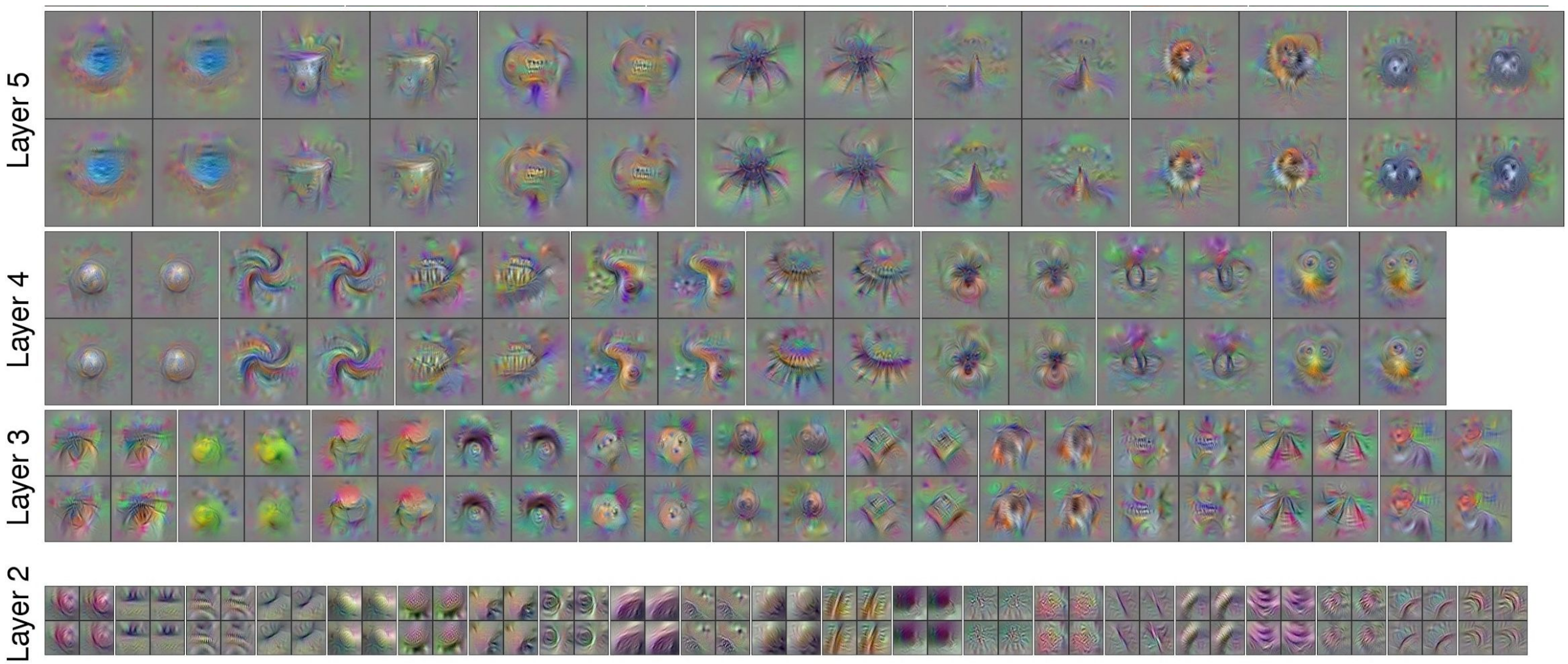
Better image regularizers give prettier results:



Yosinski et al, “Understanding Neural Networks Through Deep Visualization”, ICML DL Workshop 2015

Visualizing CNN features: Gradient Ascent

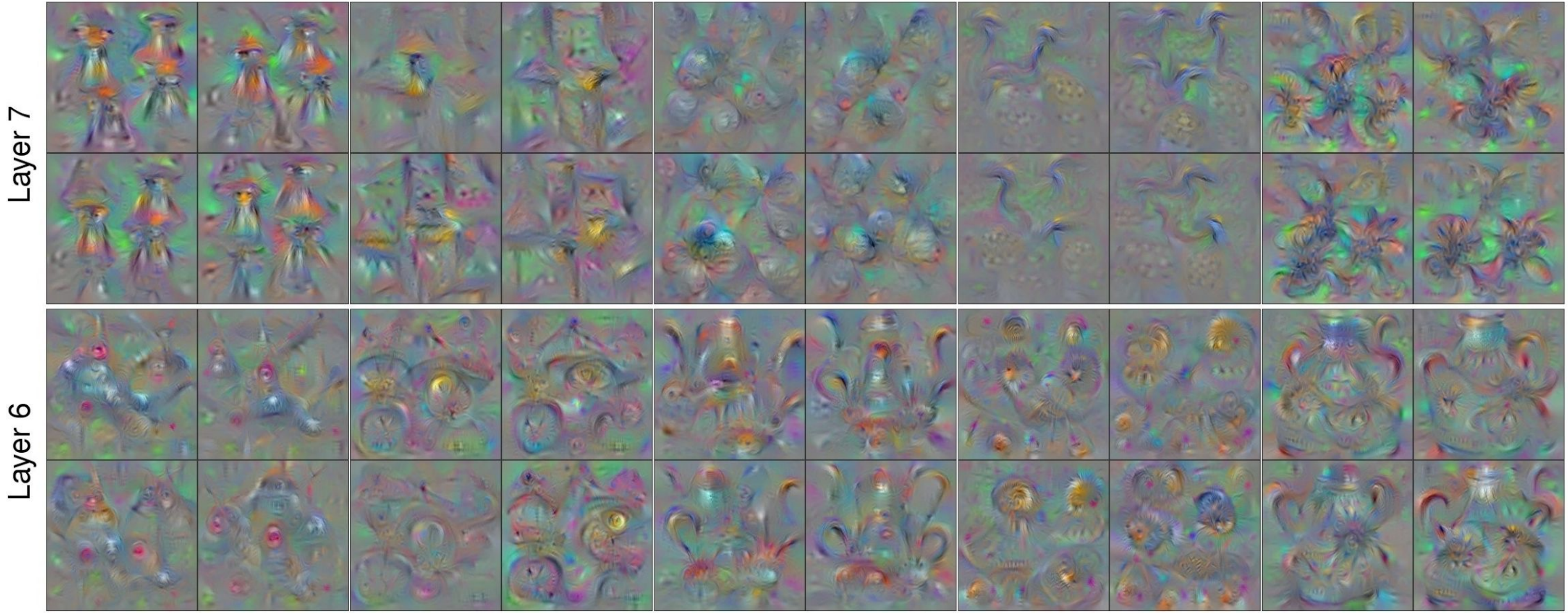
Use the same approach to visualize intermediate features



Yosinski et al, “Understanding Neural Networks Through Deep Visualization”, ICML DL Workshop 2015

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 2015

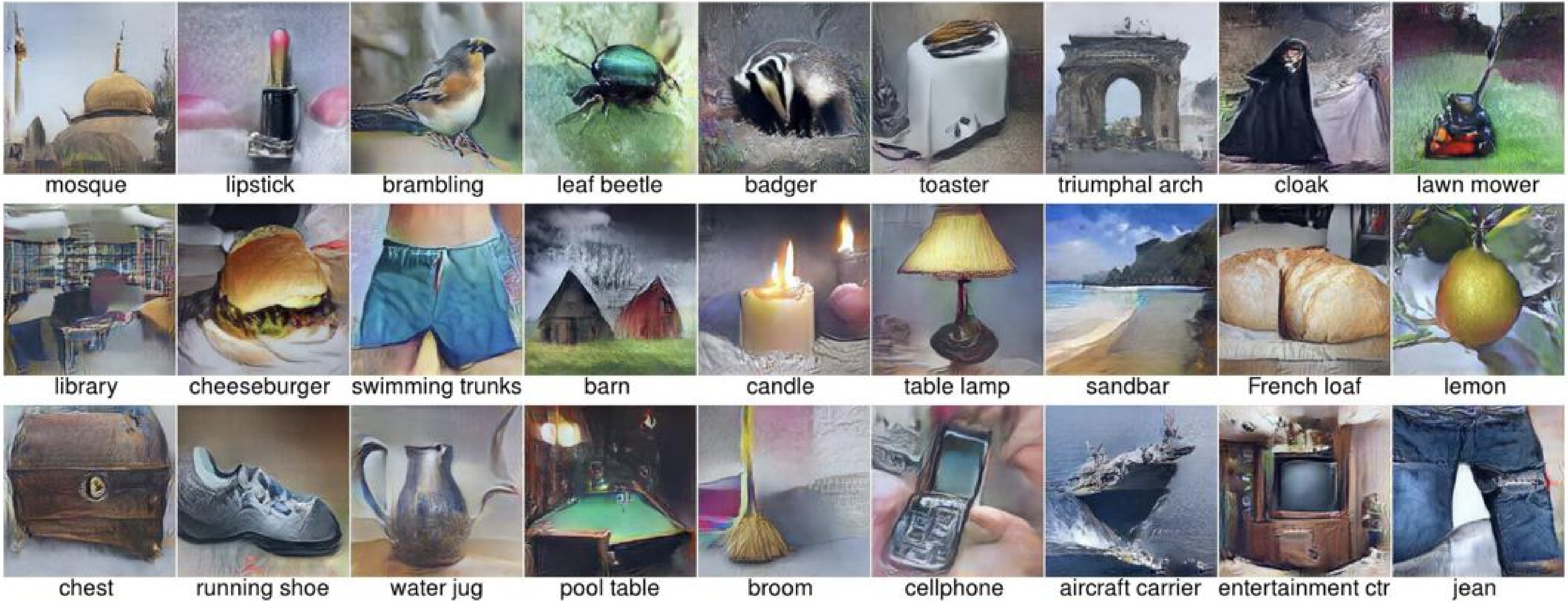
Visualizing CNN features: Gradient Ascent

You can add even more tricks to get nicer results:

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

GAN image priors give amazing results:



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

Feature Inversion

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

- Its features are similar to the given features
- It “looks natural” (image prior regularization)

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

Feature Inversion

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

- Its features are similar to the given features
- It "looks natural" (image prior regularization)

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

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

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

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

Feature Inversion

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

- Its features are similar to the given features
- It "looks natural" (image prior regularization)

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

Given feature vector

Features of new image

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

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

Total Variation regularizer (encourages spatial smoothness)

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

Feature Inversion

original image

Reconstructions from the 1000 log probabilities for ImageNet (ILSVRC) classes

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

Feature Inversion

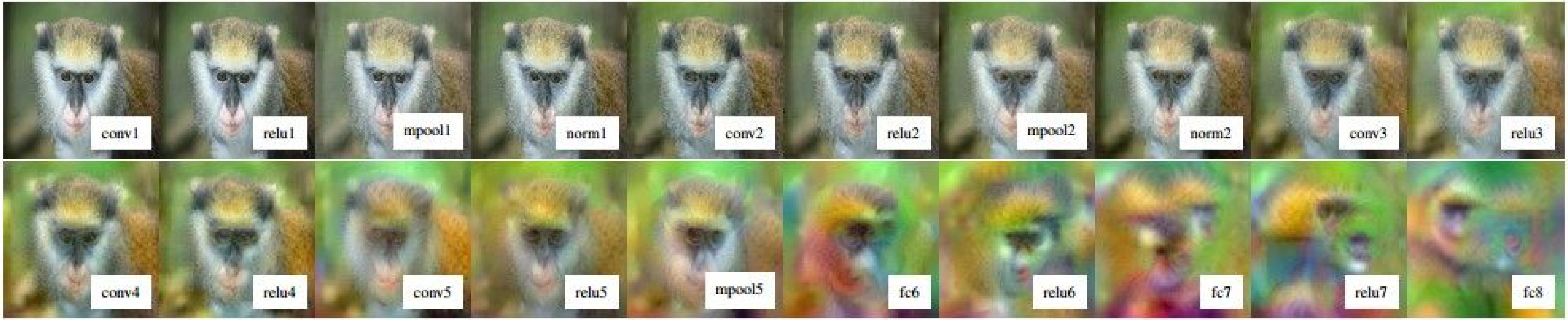
Reconstructions from the representation after last last pooling layer (immediately before the first Fully Connected layer)

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

Feature Inversion

Reconstructions from intermediate layers

Higher layers are less sensitive to changes in color, texture, and shape

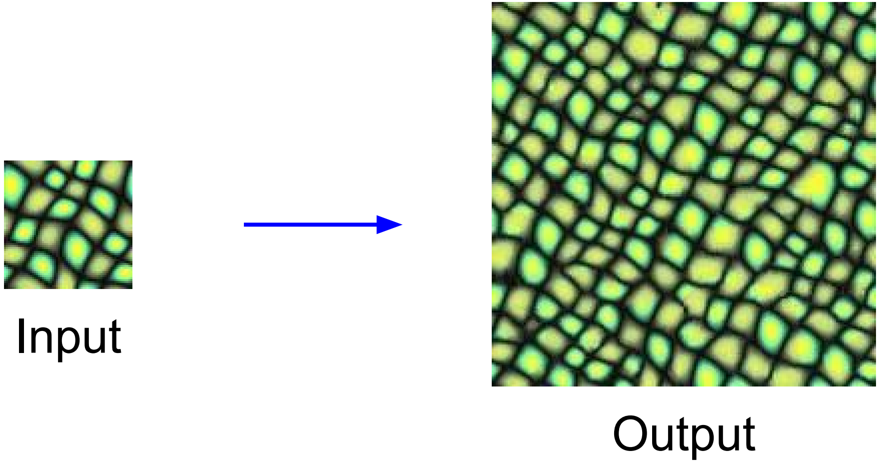


Mahendran and Vedaldi, “Understanding Deep Image Representations by Inverting Them”, CVPR 2015

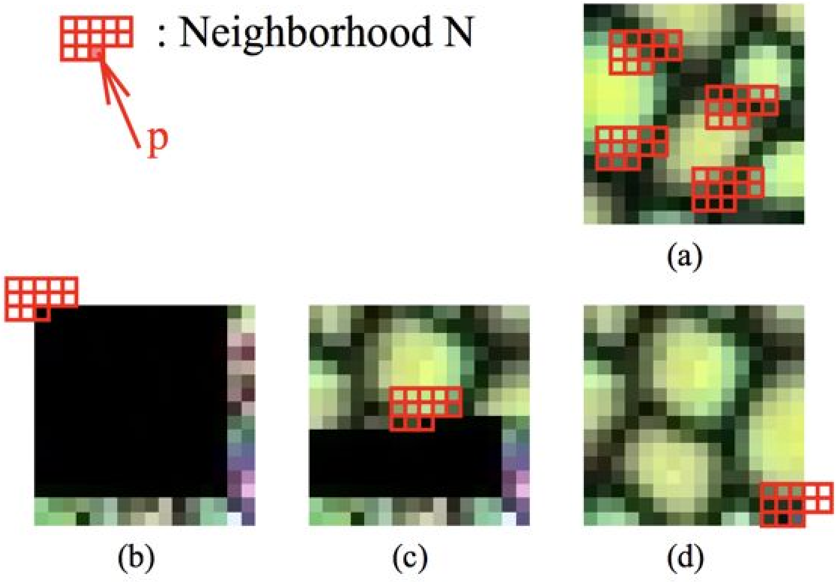
(Neural) Texture Synthesis

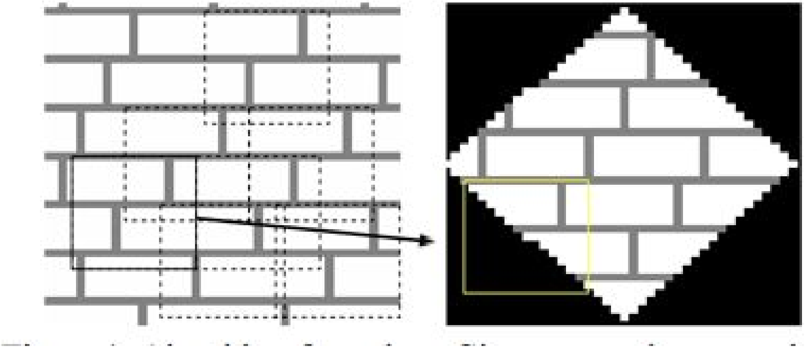
Texture Synthesis

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



Texture Synthesis

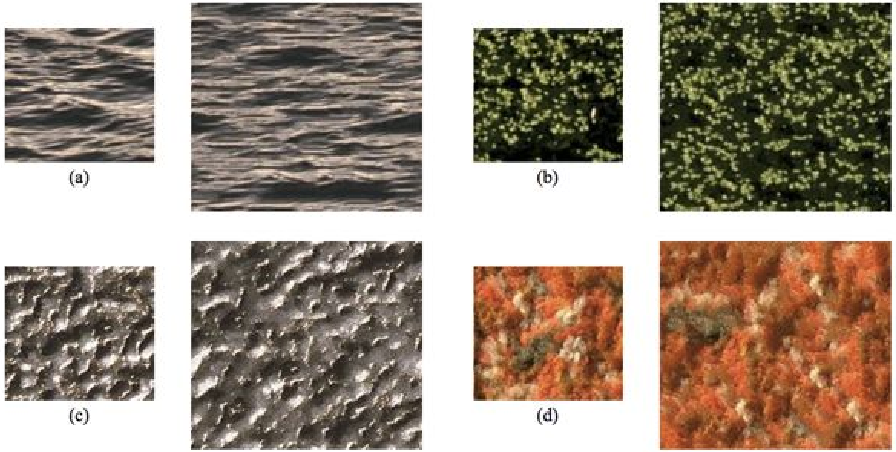




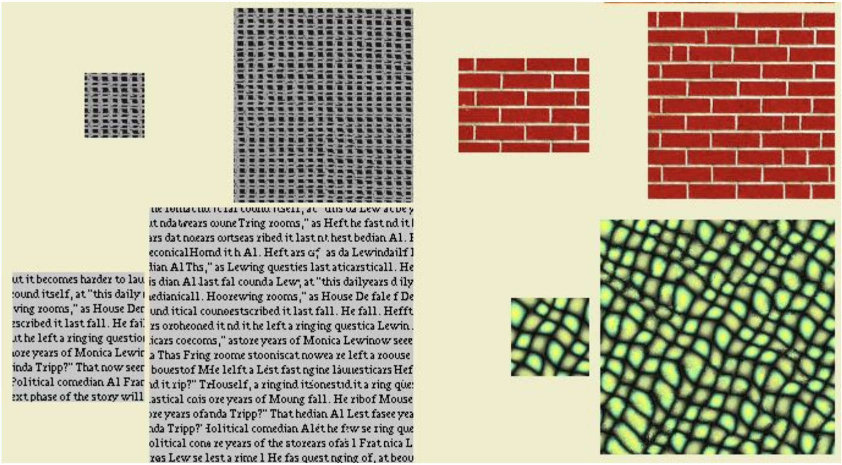
Wei and Levoy, “Fast Texture Synthesis using Tree-structured Vector Quantization”, SIGGRAPH 2000

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

Texture Synthesis



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

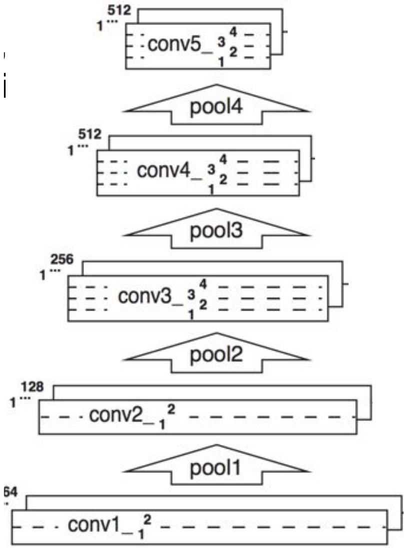


Efros and Leung, “Texture Synthesis by Non-parametric Sampling”, ICCV 1999

I have a Torch implementation here:
https://github.com/jcjohnson/texture-synthesis

Neural Texture Synthesis

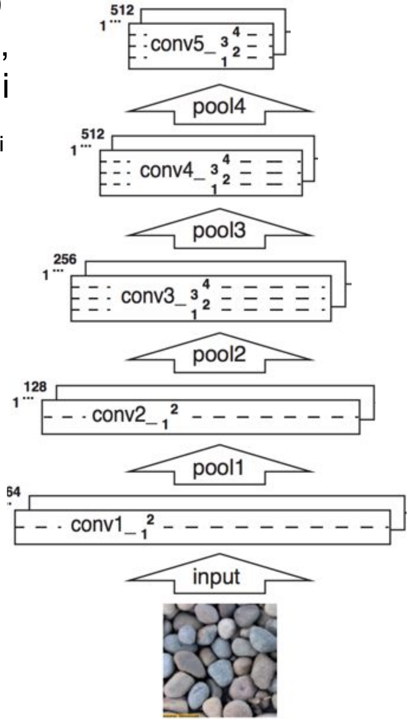
1. Pretrain a CNN on ImageNet (VGG-19)



Gatys et al, “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 $C_i \times H_i \times W_i$

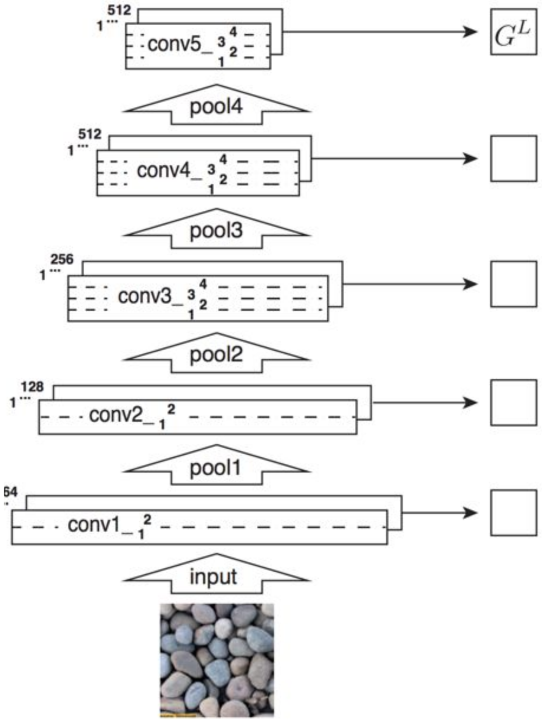


Gatys et al, “Texture Synthesis using Convolutional Neural Networks”, NIPS 2015

Neural Texture Synthesis

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

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



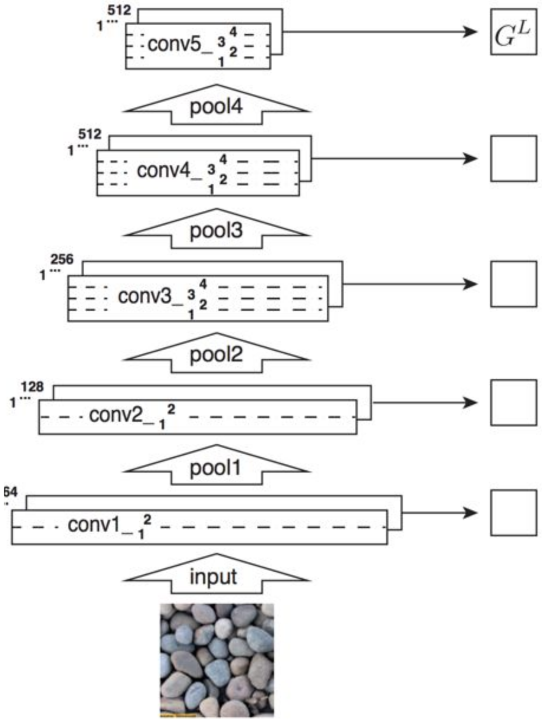
Gatys et al, “Texture Synthesis using Convolutional Neural Networks”, NIPS 2015

Neural Texture Synthesis

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

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

4. Initialize generated image from random noise



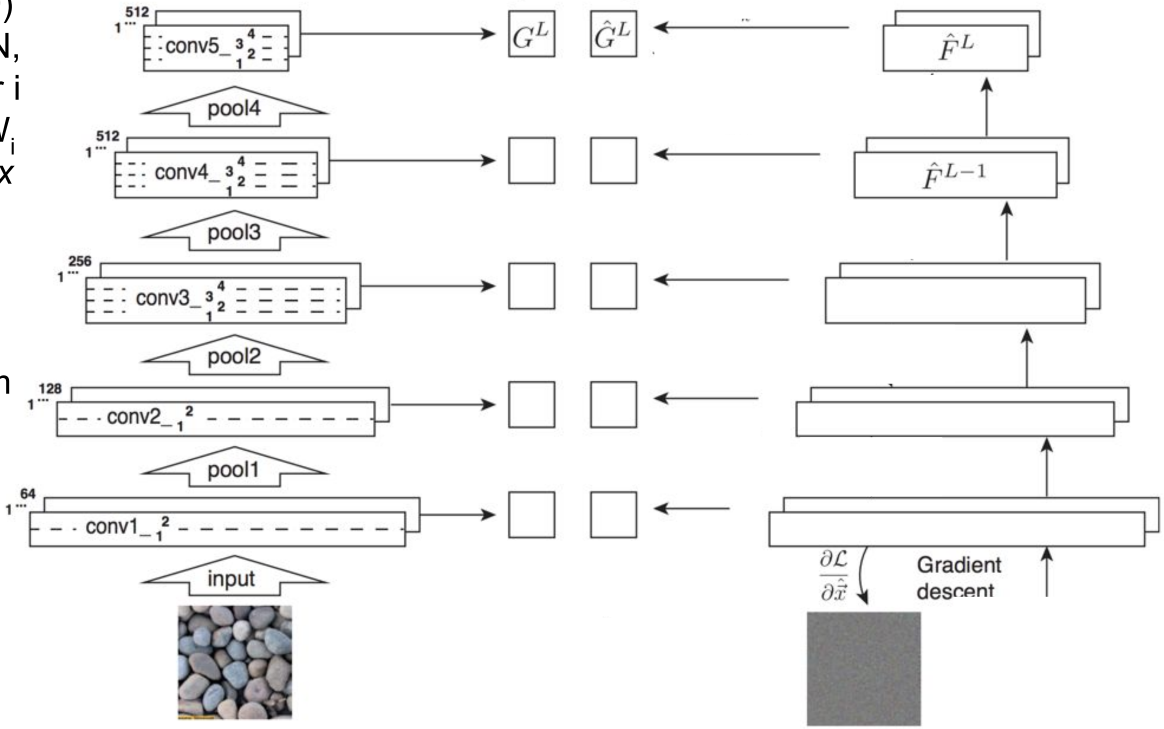
Gatys et al, "Texture Synthesis using Convolutional Neural Networks", NIPS 2015

Neural Texture Synthesis

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

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

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



Gatys et al, "Texture Synthesis using Convolutional Neural Networks", NIPS 2015

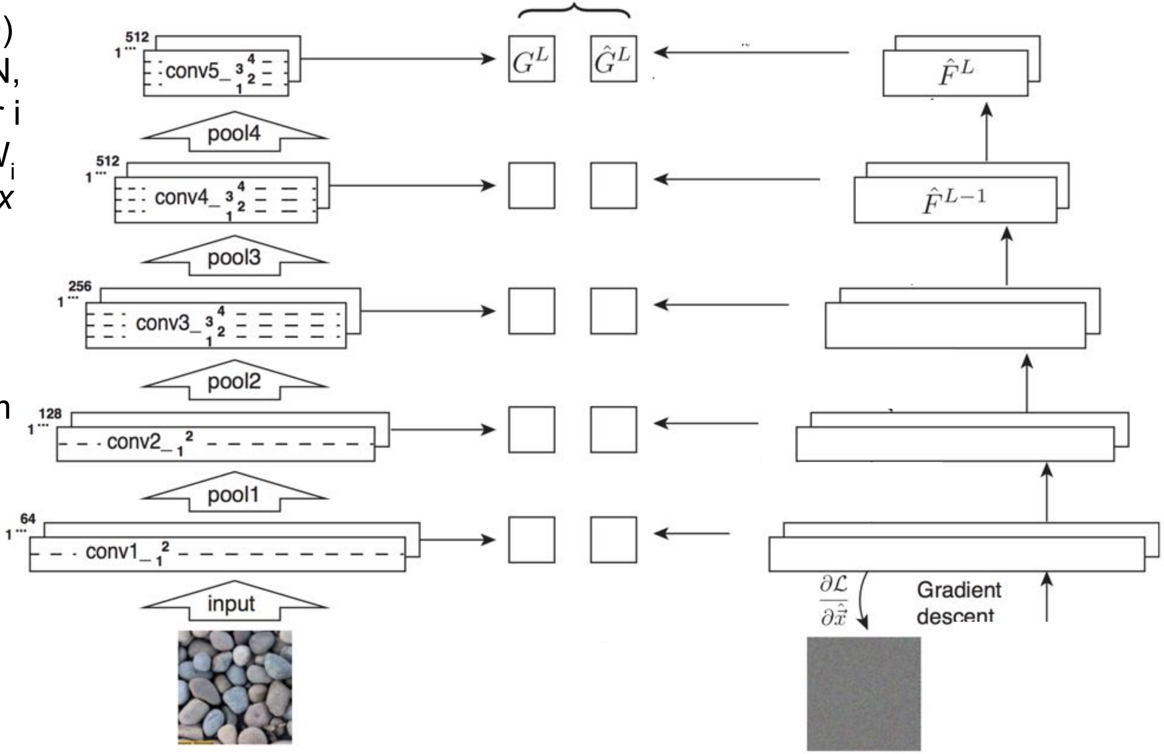
Neural Texture Synthesis

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

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

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

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



Gatys et al, "Texture Synthesis using Convolutional Neural Networks", NIPS 2015

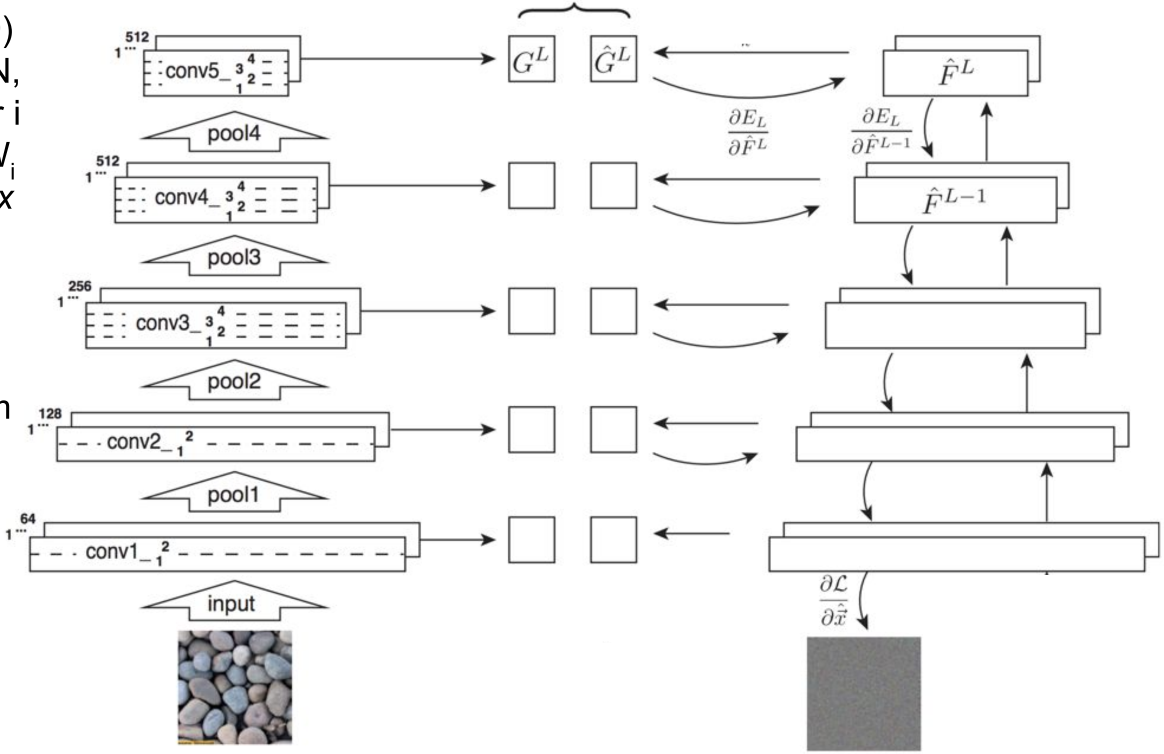
Neural Texture Synthesis

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

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

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

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



Gatys et al, "Texture Synthesis using Convolutional Neural Networks", NIPS 2015

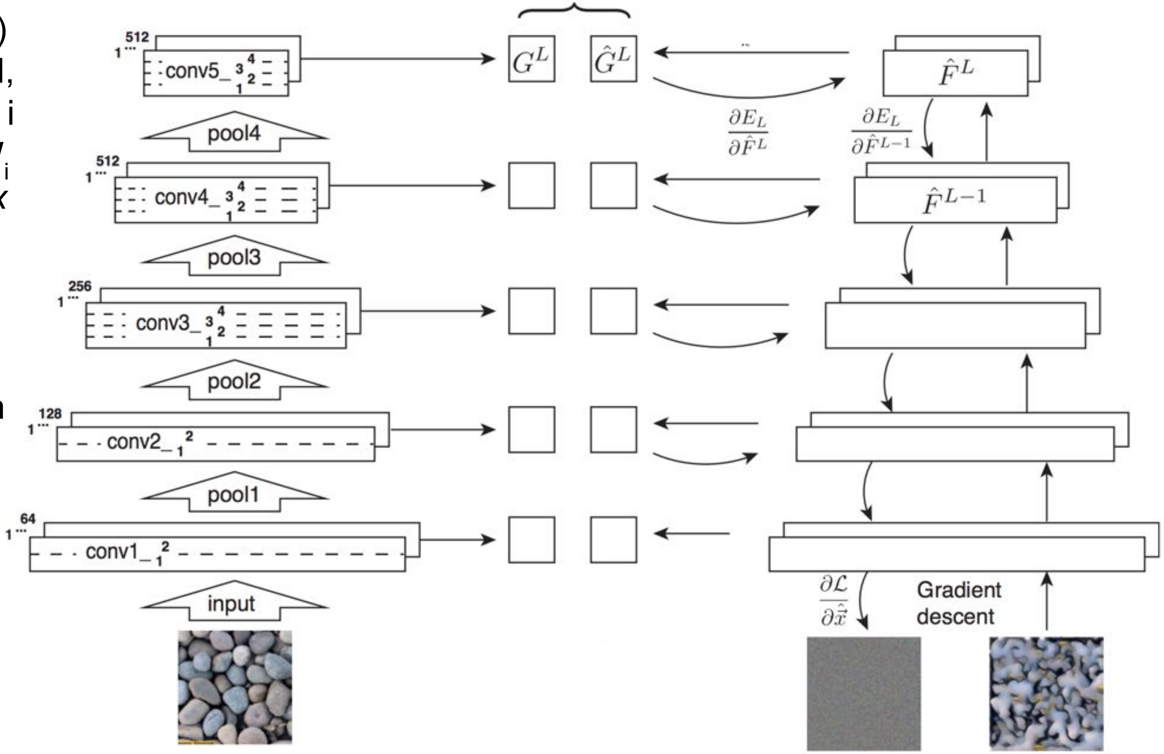
Neural Texture Synthesis

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

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

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

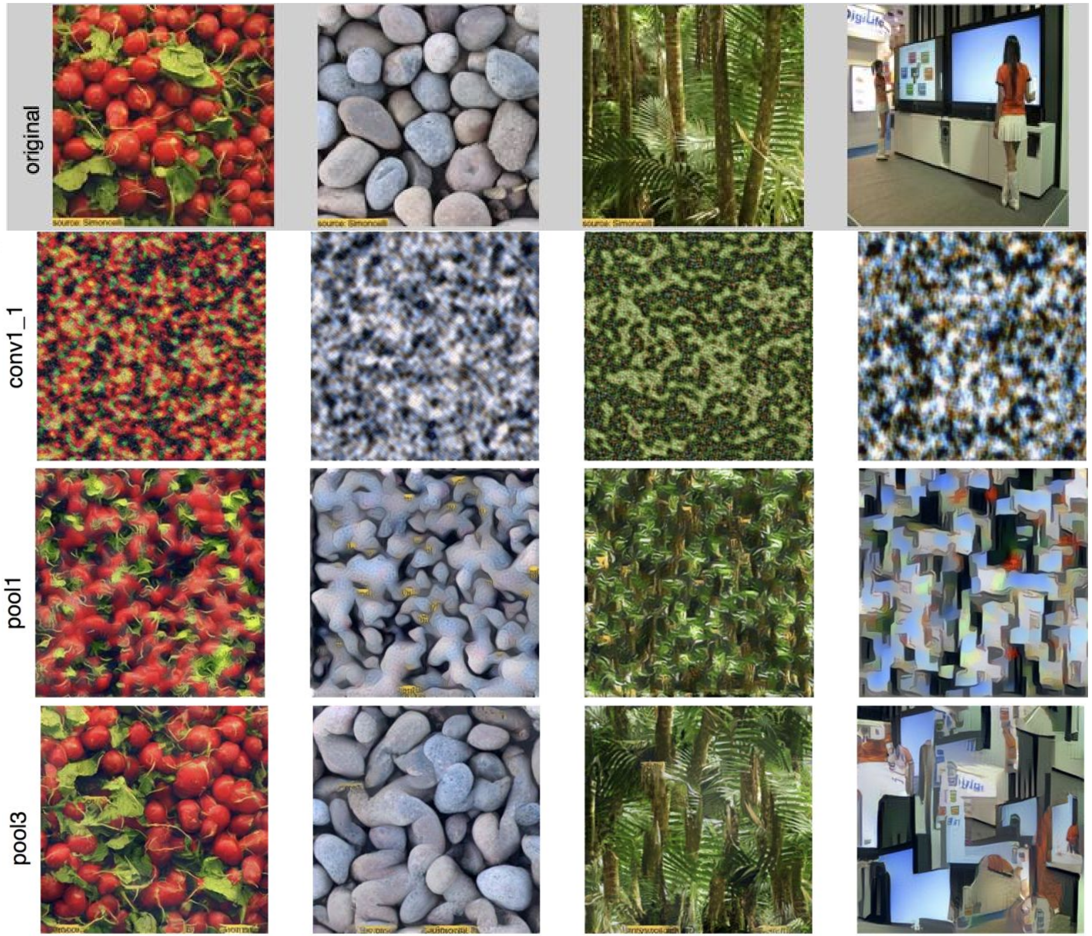
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5



Gatys et al, “Texture Synthesis using Convolutional Neural Networks”, NIPS 2015

Neural Texture Synthesis

Reconstructing from higher layers recovers larger features from the input texture

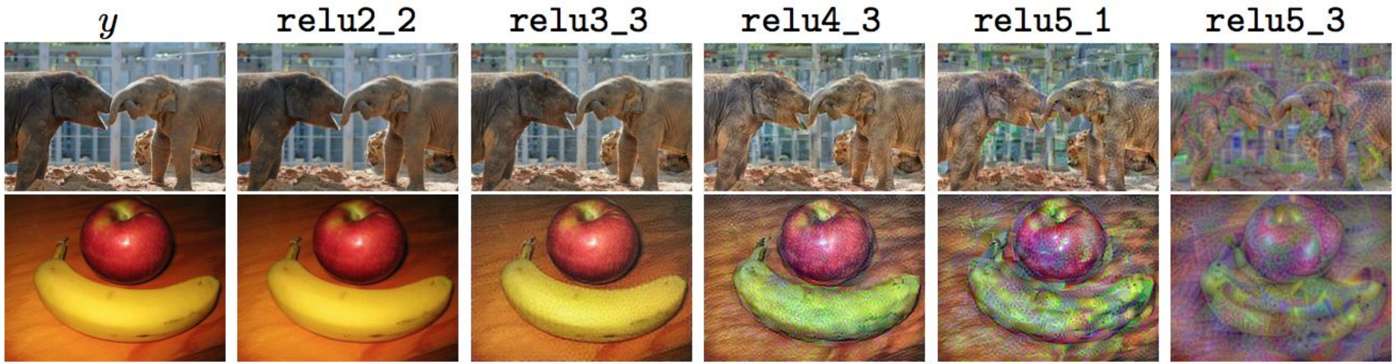


Gatys et al, “Texture Synthesis using Convolutional Neural Networks”, NIPS 2015

Style Transfer:
Feature Inversion + Texture Synthesis

Neural Style Transfer: Feature + Gram reconstruction

Feature reconstruction



Texture synthesis (Gram reconstruction)

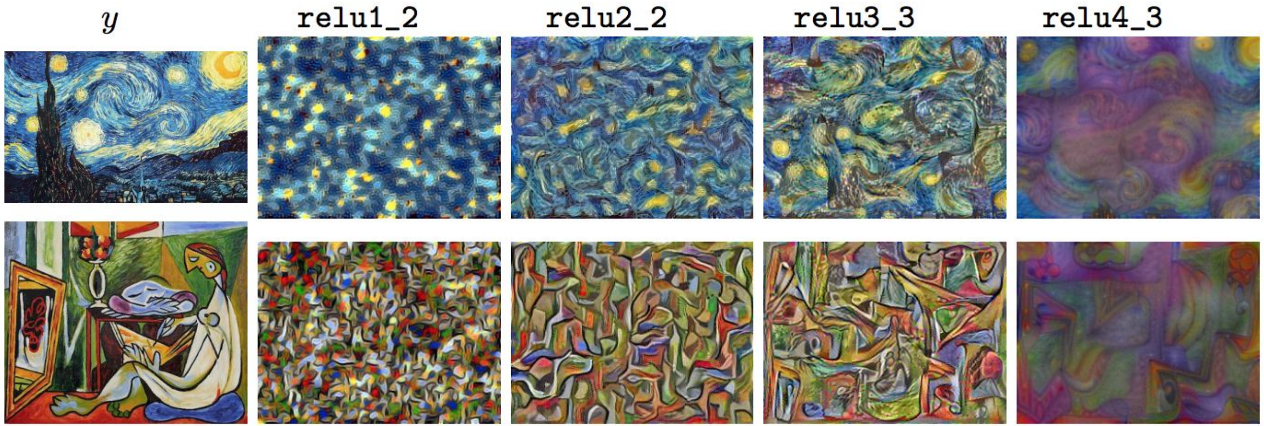


Figure credit: Johnson et al, “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016

Neural Style Transfer

Given a **content image** and a **style image**, find a new image that

- Matches the CNN features of the content image (feature reconstruction)
- Matches the Gram matrices of the style image (texture synthesis)

Combine feature reconstruction from Mahendran et al with Neural Texture Synthesis from Gatys et al, using the same CNN!

Content Image

+

Style Image

Gatys et al, “A Neural Algorithm of Artistic Style”, arXiv 2015
Gatys et al, “Image Style Transfer using Convolutional Neural Networks”, CVPR 2016

Neural Style Transfer

Given a **content image** and a **style image**, find a new image that

- Matches the CNN features of the content image (feature reconstruction)
- Matches the Gram matrices of the style image (texture synthesis)

Combine feature reconstruction from Mahendran et al with Neural Texture Synthesis from Gatys et al, using the same CNN!

Content Image

+

Style Image

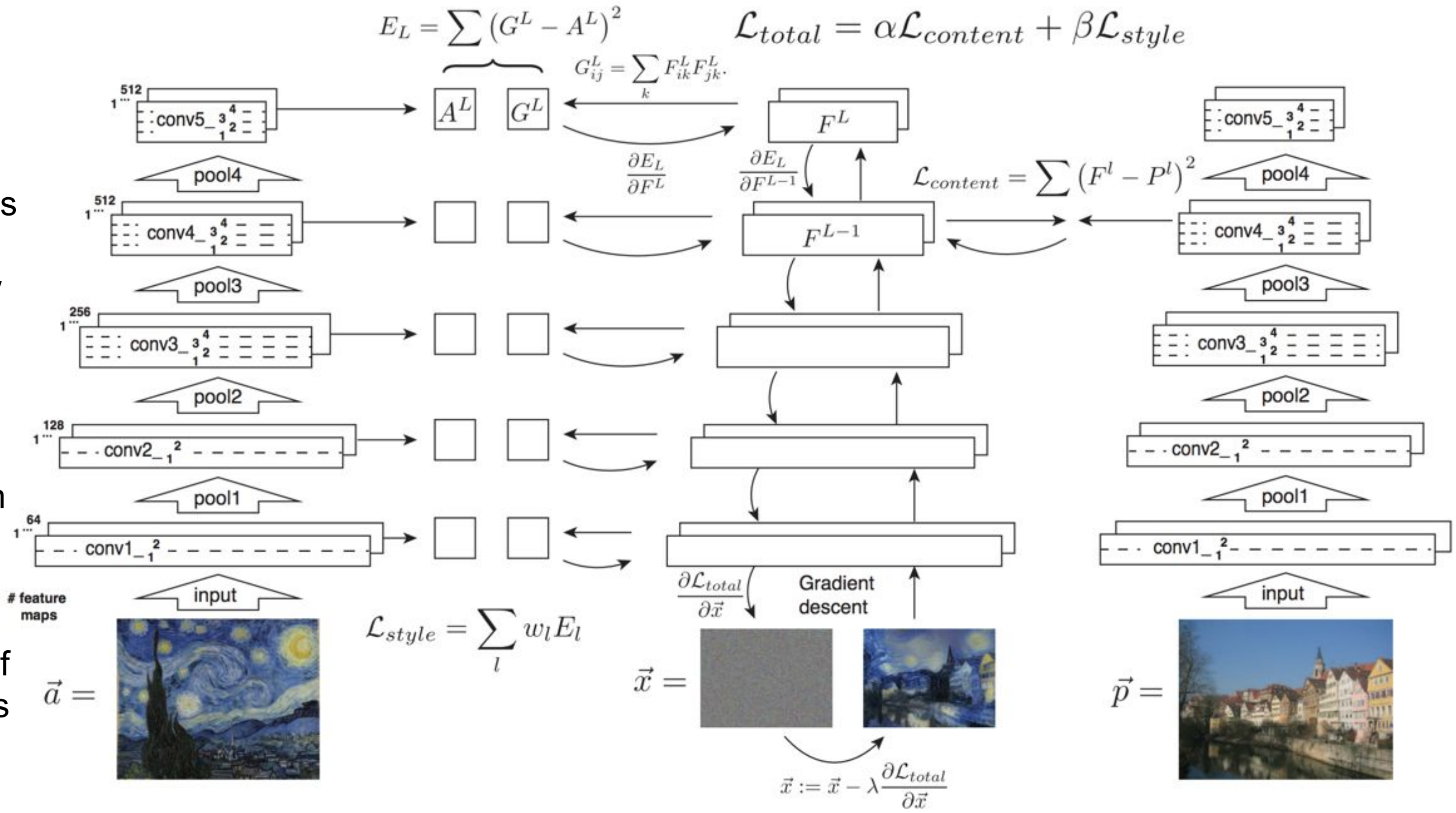
=

Stylized Result

Gatys et al, “A Neural Algorithm of Artistic Style”, arXiv 2015
Gatys et al, “Image Style Transfer using Convolutional Neural Networks”, CVPR 2016

Neural Style Transfer

1. Pretrain CNN
2. Compute features for content image
3. Compute Gram matrices for style image
4. Randomly initialize new image
5. Forward new image through CNN
6. Compute style loss (L2 distance between Gram matrices) and content loss (L2 distance between features)
7. Loss is weighted sum of style and content losses
8. Backprop to image
9. Take a gradient step
10. GOTO 5

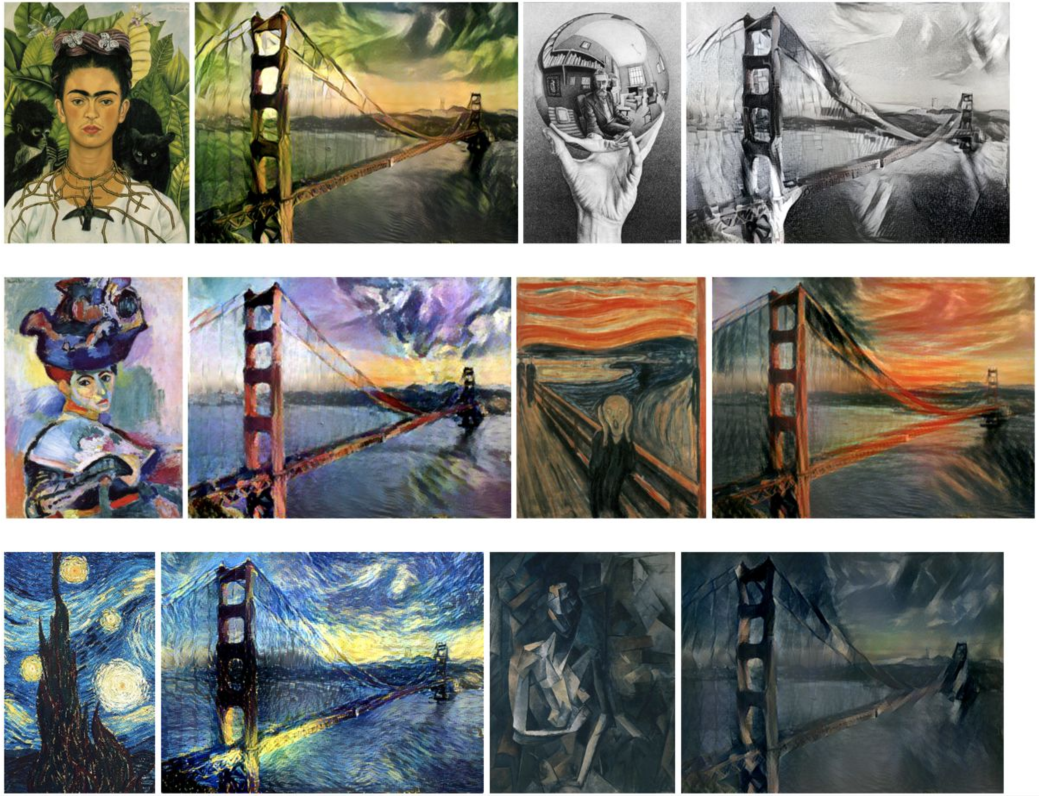


Gatys et al, “Image Style Transfer using Convolutional Neural Networks”, CVPR 2016

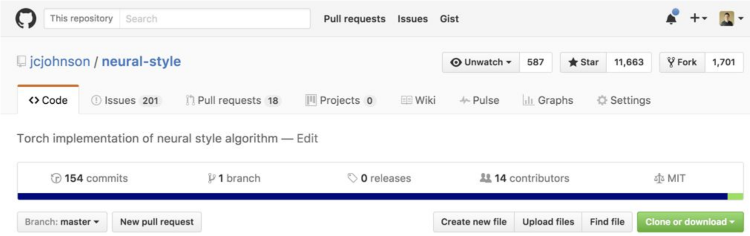
Neural Style Transfer

Gatys et al, “Image Style Transfer using Convolutional Neural Networks”, CVPR 2016

Neural Style Transfer



From my implementation on GitHub:
https://github.com/jcjohnson/neural-style



Gatys et al, “Image Style Transfer using Convolutional Neural Networks”, CVPR 2016

Neural Style Transfer: Style / Content Tradeoff

More weight to content loss ⟷ More weight to style loss

Justin Johnson, “neural-style”, https://github.com/jcjohnson/neural-style

Neural Style Transfer: Style Scale

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

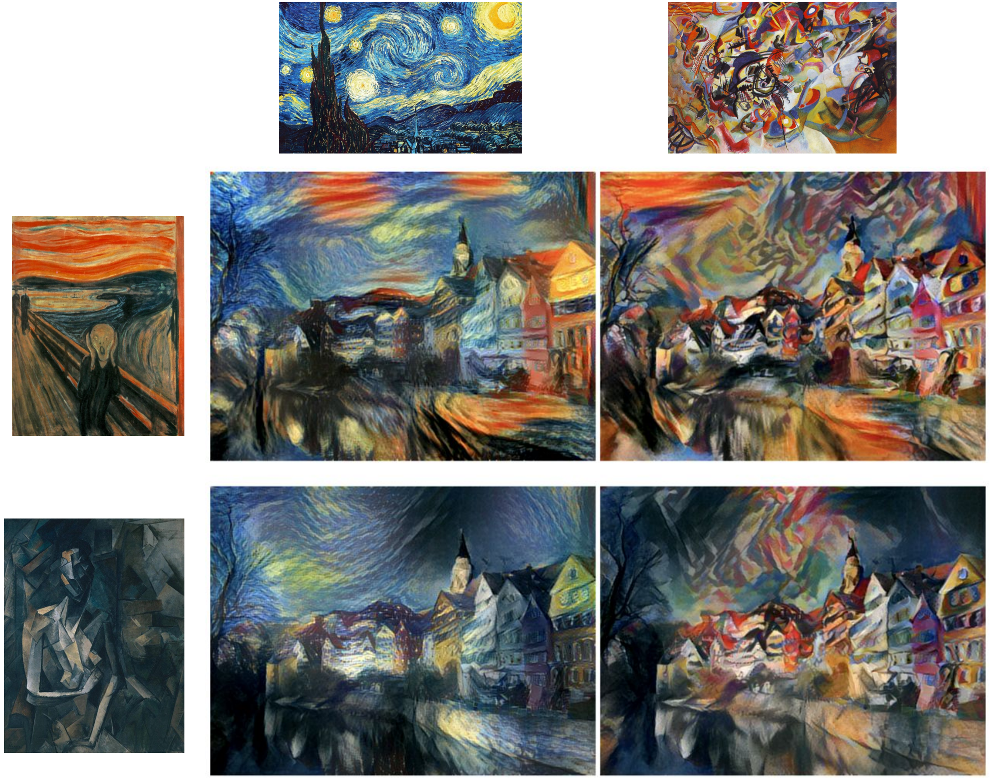
Larger style image

Smaller style image

Justin Johnson, “neural-style”, https://github.com/jcjohnson/neural-style

Neural Style Transfer: Multiple Style Images

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



Justin Johnson, “neural-style”, https://github.com/jcjohnson/neural-style

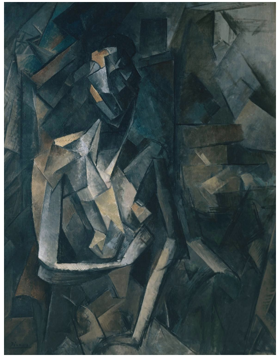
Neural Style Transfer: Multiple Style Images

More “Scream” ⟷ More “Starry Night”

Justin Johnson, “neural-style”, https://github.com/jcjohnson/neural-style

Neural Style Transfer: Preserve colors

Style



Content

Perform style transfer only on the luminance channel
(eg Y in YUV colorspace);
Copy colors from content image

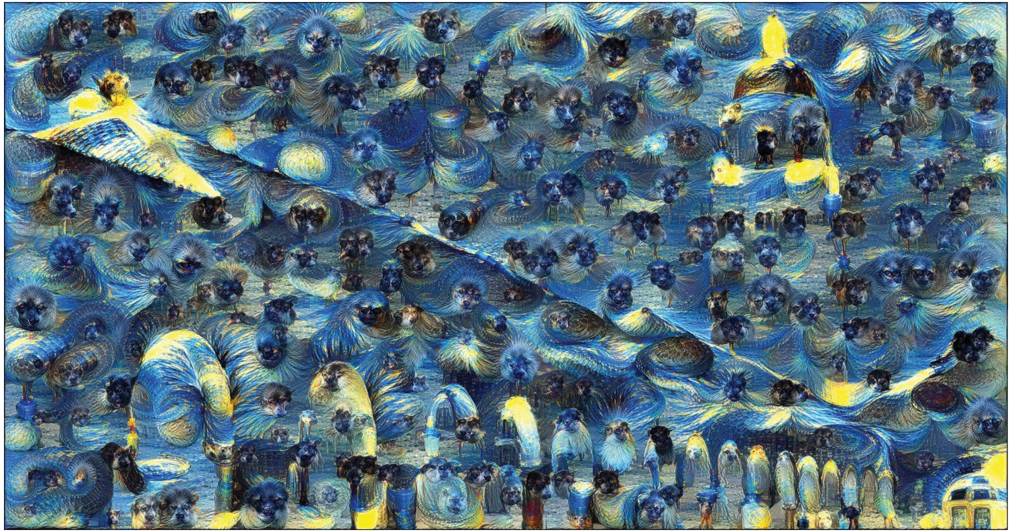
Normal style transfer

Color-preserving style transfer

http://blog.deepart.io/2016/06/04/color-independent-style-transfer/
Gatys et al, "Preserving Color in Neural Artistic Style Transfer", arXiv 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", arXiv 2016

Simultaneous DeepDream and Style Transfer!

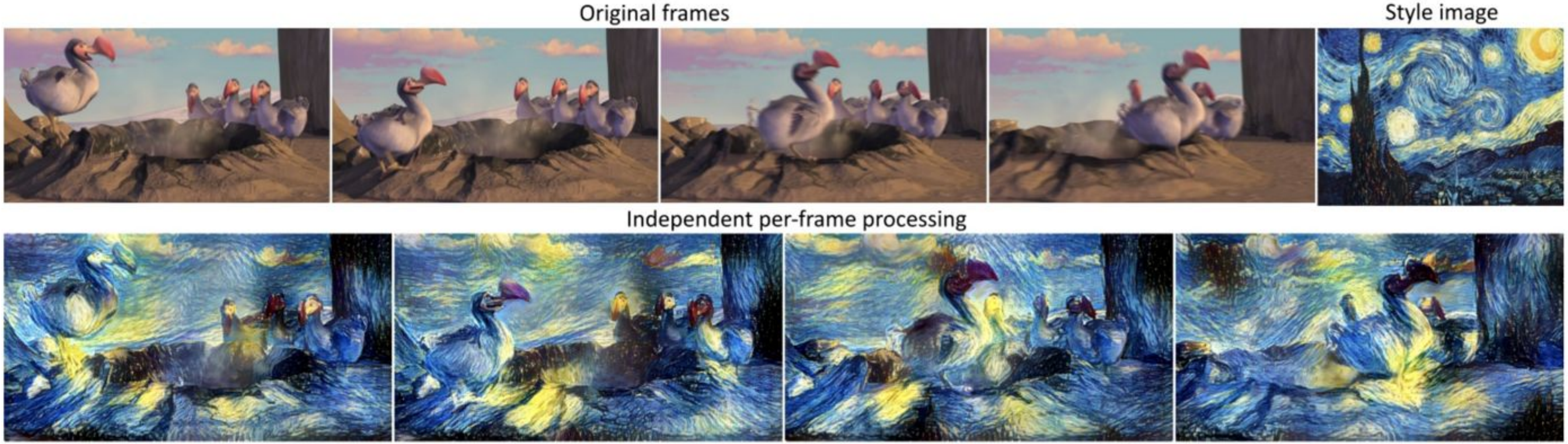
Jointly minimize feature reconstruction loss, style reconstruction loss, and maximize DeepDream feature amplification loss!



https://github.com/jcjohnson/fast-neural-style/issues/5

Style Transfer on Video

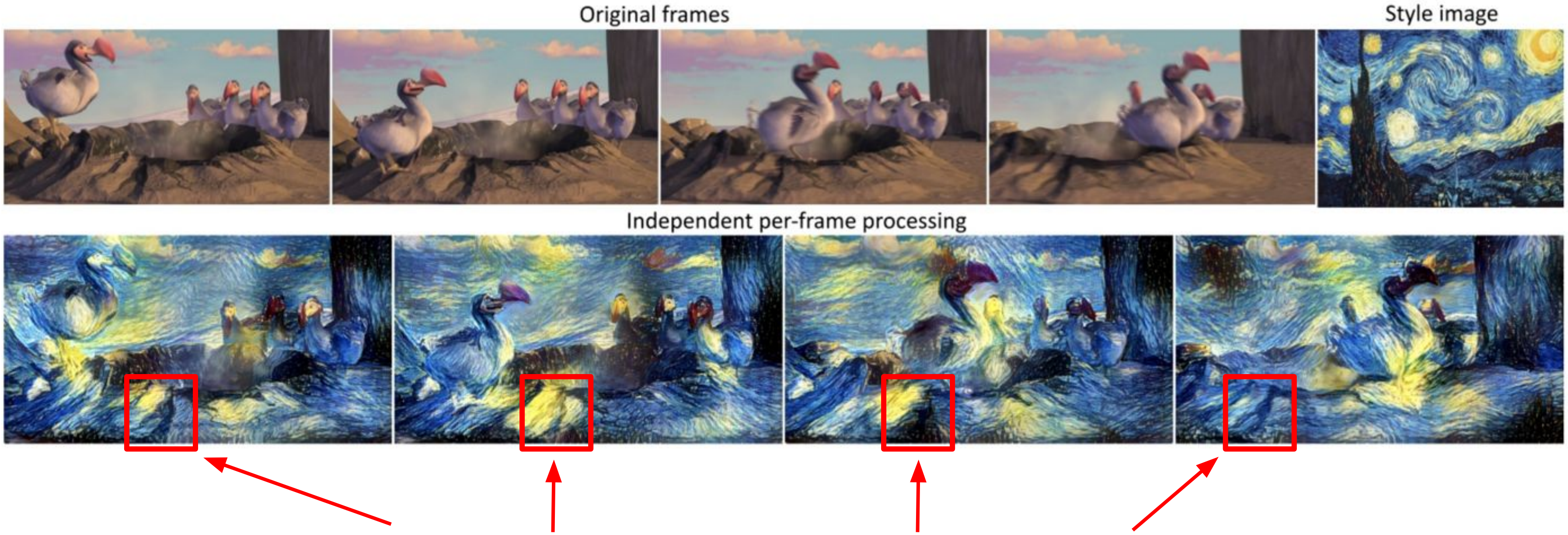
Running style transfer independently on each video frame results in poor per-frame consistency:



Ruder et al, “Artistic style transfer for videos”, arXiv 2016

Style Transfer on Video

Running style transfer independently on each video frame results in poor per-frame consistency:



Appearance of the rock formation different in each frame!

Ruder et al, “Artistic style transfer for videos”, arXiv 2016

Style Transfer on Video

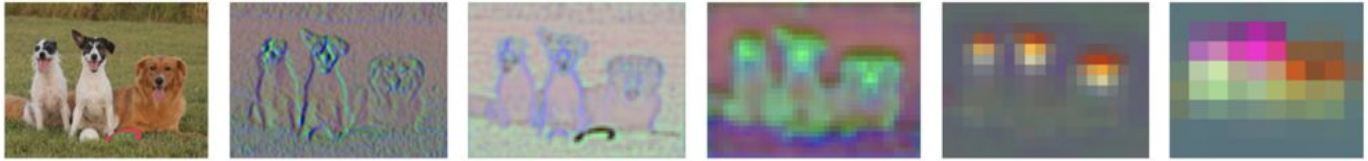
Tricks for video style transfer:
- **Initialization:** Initialize frame t+1 with a warped version of the stylized result at frame t (using optical flow)
- **Short-term temporal consistency**: warped forward optical flow should be opposite of backward optical flow
- **Long-term temporal consistency**: When a region is occluded then visible again, it should look the same
- **Multipass processing**: Make multiple forward and backward passes over the video with few iterations per pass

Ruder et al, "Artistic style transfer for videos", arXiv 2016
https://github.com/manuelruder/artistic-videos

Beyond Gram Matrices: CNNMRF

Idea: Use patch matching like classic texture synthesis,
but match patches in CNN feature space rather than pixel space!

Neural patches at different layers of VGG19:



input image relu2_1 relu3_1 relu4_1 relu5_1 pool5

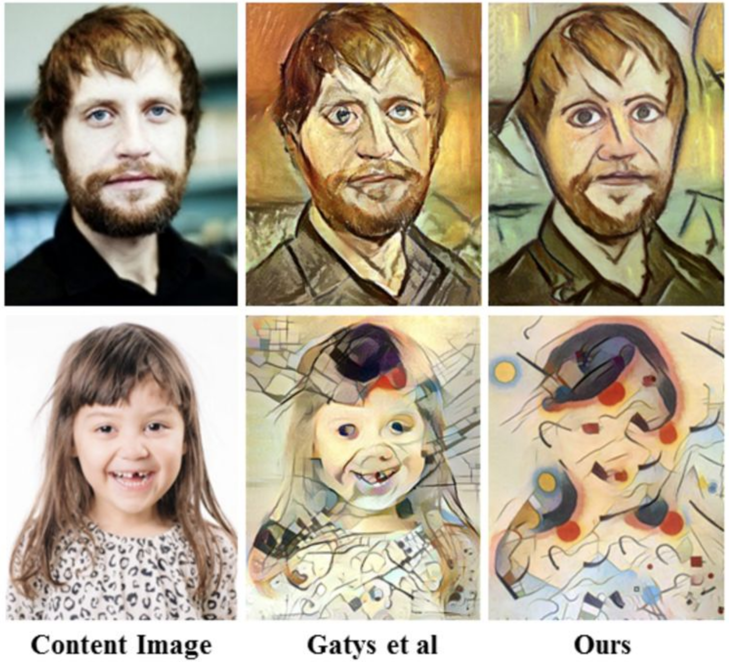
$$E_s(\Phi(\mathbf{x}), \Phi(\mathbf{x}_s)) = \sum_{i=1}^{m} ||\Psi_i(\Phi(\mathbf{x})) - \Psi_{NN(i)}(\Phi(\mathbf{x}_s))||^2 \quad (2)$$

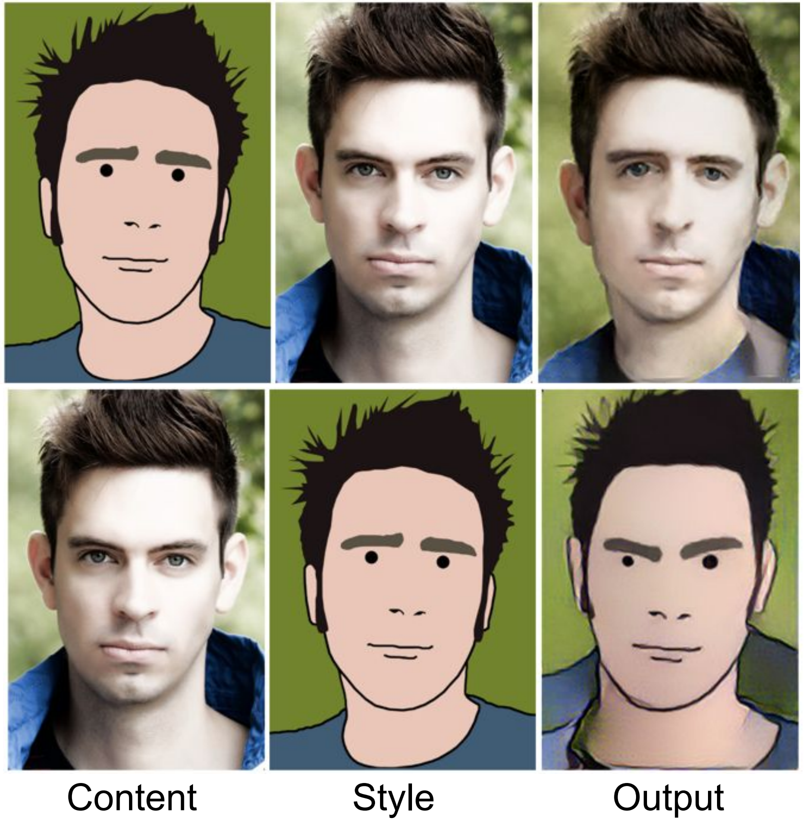
$$NN(i) := \underset{j=1,\ldots,m_s}{\arg\min} \frac{\Psi_i(\Phi(\mathbf{x})) \cdot \Psi_j(\Phi(\mathbf{x}_s))}{|\Psi_i(\Phi(\mathbf{x}))| \cdot |\Psi_j(\Phi(\mathbf{x}_s))|}$$

For each neural patch in generated image, find nearest-neighbor
neural patch in style image; minimize distance between patches

Li and Wand, "Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis", CVPR 2016
https://github.com/chuanli11/CNNMRF

Beyond Gram Matrices: CNNMRF





Li and Wand, “Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis”, CVPR 2016
https://github.com/chuanli11/CNNMRF

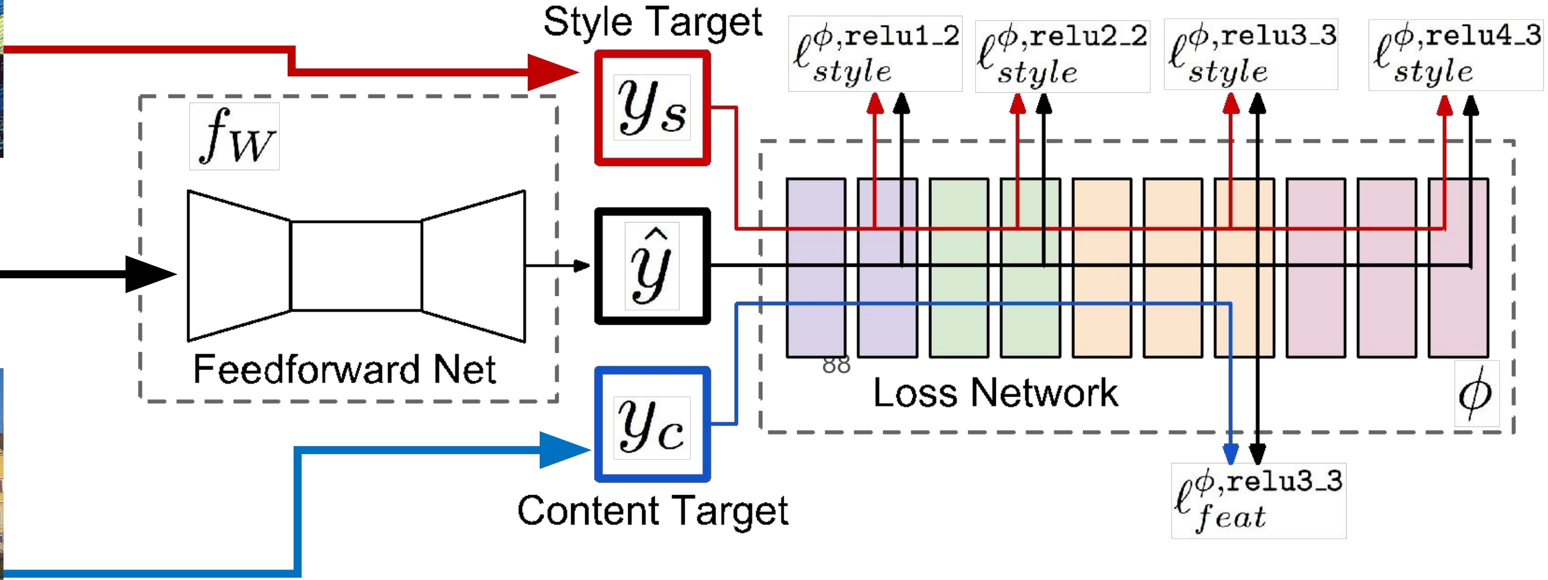
Fast Style Transfer

Problem: Style transfer is slow; need hundreds of forward + backward passes of VGG

Solution: Train a feedforward network to perform style transfer!

Fast Style Transfer

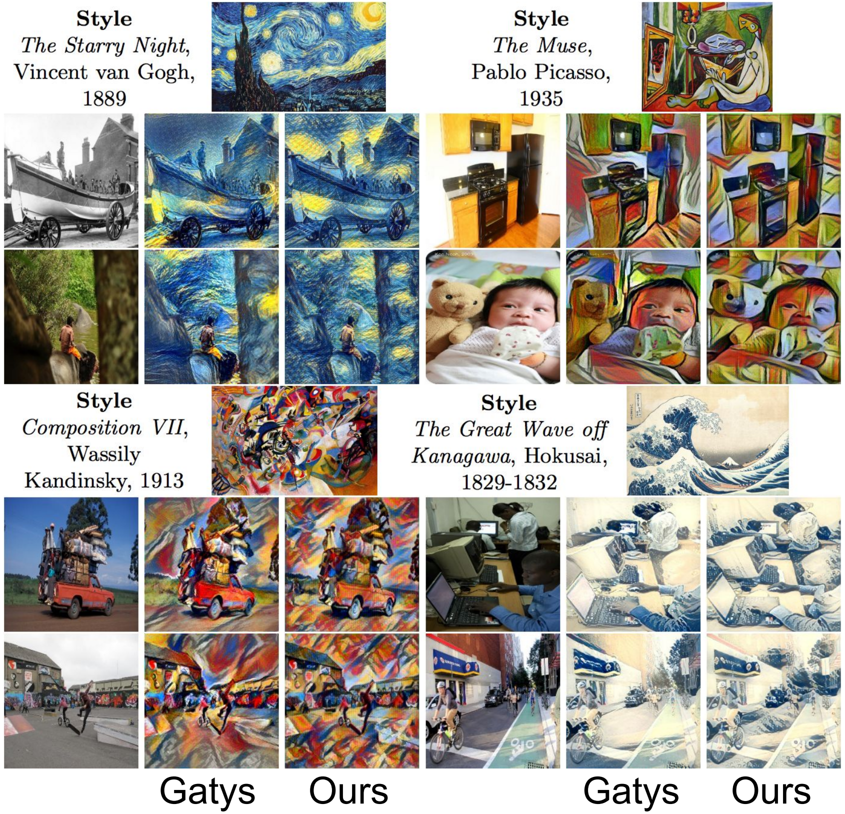
(1) Train a feedforward network for each style
(2) Use pretrained CNN to compute same losses as before
(3) After training, stylize images using a single forward pass



Works real-time at test-time!

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

Fast Style Transfer

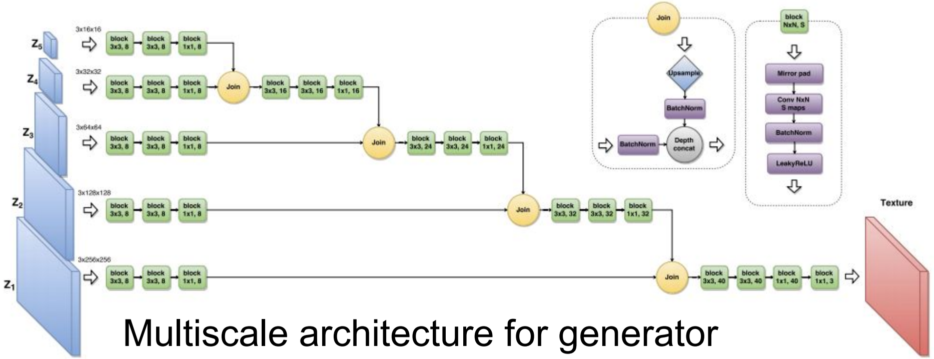


Johnson et al, “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016
https://github.com/jcjohnson/fast-neural-style

Works real-time on video!

Fast Style Transfer: Texture Networks

Concurrent work with mine
with comparable results



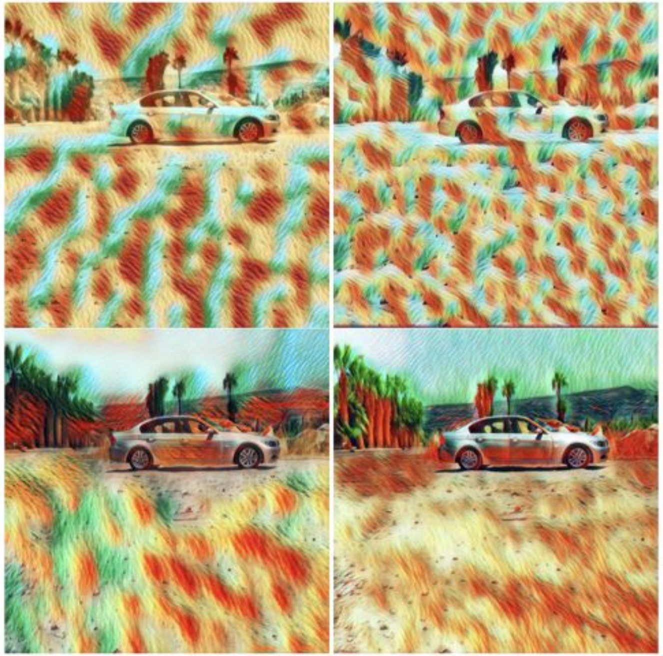
Multiscale architecture for generator

Ulyanov et al, "Texture Networks: Feed-forward Synthesis of Textures and Stylized Images", ICML 2016
https://github.com/DmitryUlyanov/texture_nets

Fast Style Transfer: Instance Normalization

A minor tweak to the architecture of the generator significantly improves results

Ulyanov et al Johnson et al



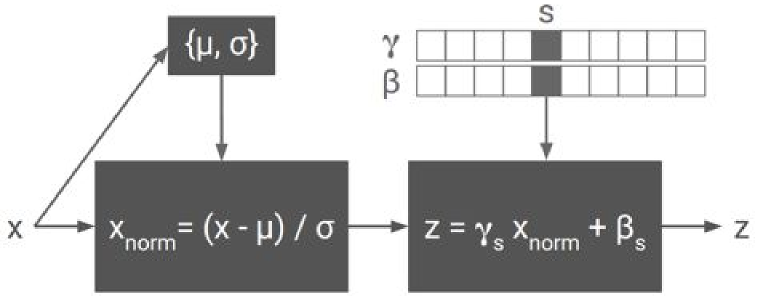
Batch Normalization

Instance Normalization

Ulyanov et al, “Instance Normalization: The Missing Ingredient for Fast Stylization”, ICML 2016

Fast Style Transfer: Multiple styles with one network

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



At test-time, blend scale and shift parameters for realtime style blending!

Dumoulin et al, "A Learned Representation for Artistic Style", arXiv 2016
https://research.googleblog.com/2016/10/supercharging-style-transfer.html

Fast Style Transfer: Multiple styles with one network

Dumoulin et al, “A Learned Representation for Artistic Style”, arXiv 2016
https://research.googleblog.com/2016/10/supercharging-style-transfer.html

For more details on CNNs,
take CS 231n in Spring!