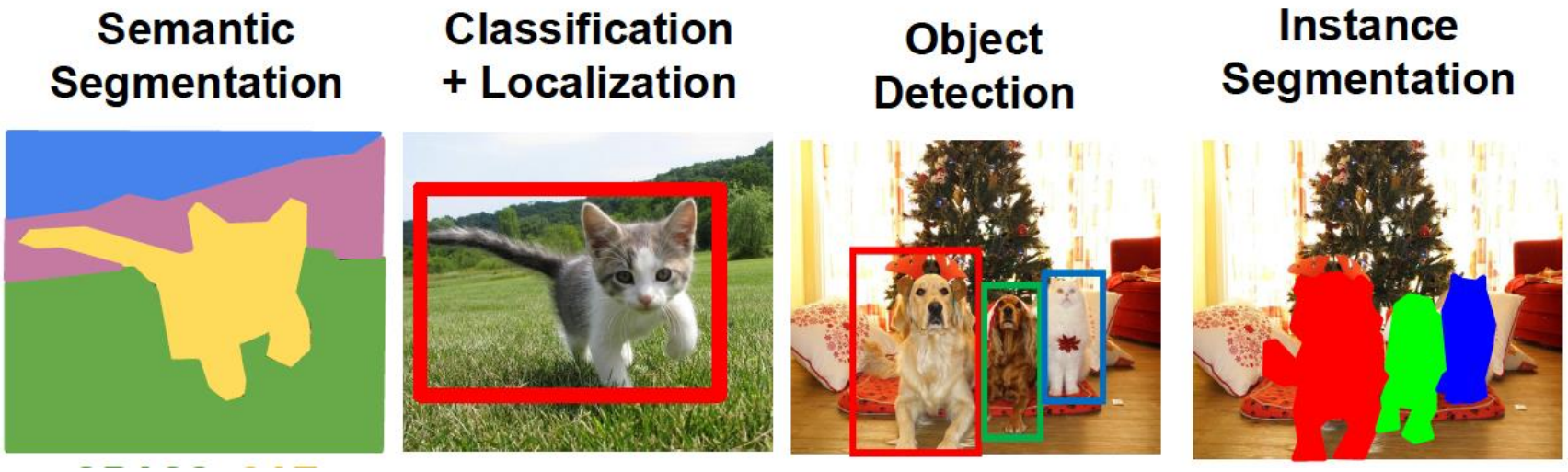
Lecture 11: CNNs – Visualizing and Understanding

Xuming He
SIST, ShanghaiTech
Fall, 2019

Previously on CNNs

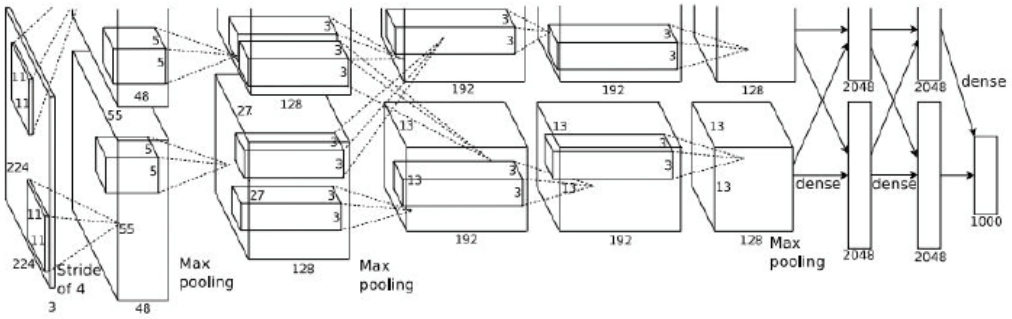
- CNNs in computer vision
  - Localization, Detection
  - Semantic segmentation
  - Instance segmentation



What CNNs have learned?

- What representations are used by CNNs?
  - Our goal: hierarchical feature learning
  - Can we look into the model to understand it?

Input Image:
3 x 224 x 224



Class Scores:
1000 numbers

What are the intermediate features looking for?

Outline

- Understanding CNN through visualization
  - Visualizing filters: Network weights
  - Visualizing neural activations: Network outputs
  - Visualizing sensitivities: Network inputs
- Case studies
  - Adversarial examples
  - Neural texture synthesis and style transfer

Acknowledgement: Roger Grosse's CSC231 and Feifei Li et al's cs231n notes

I. Visualizing filters

- We can understand the first-layer features by visualizing the weights
  - The better the input matches these weights, the more the neurons activate

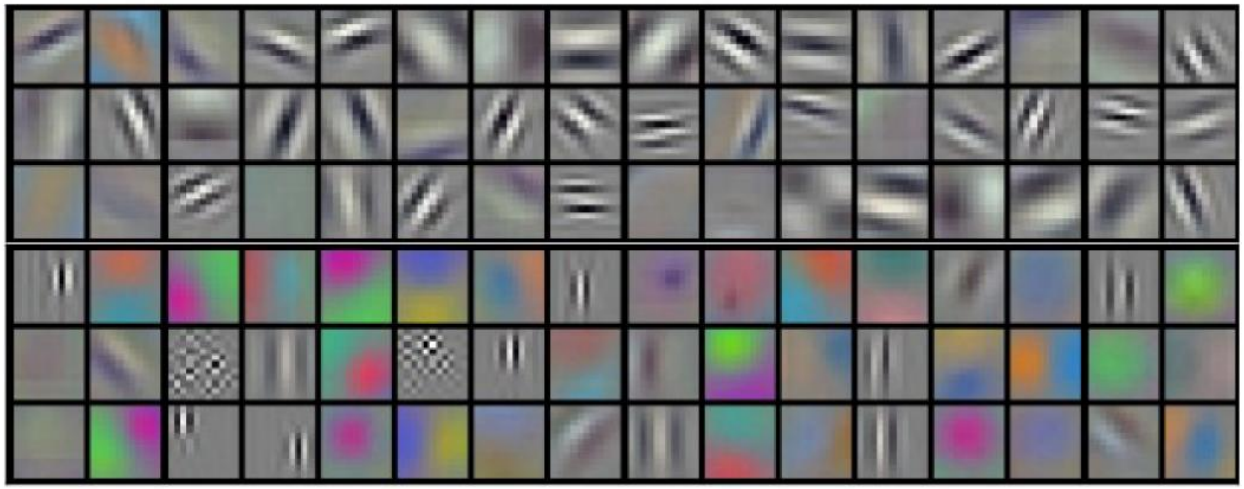
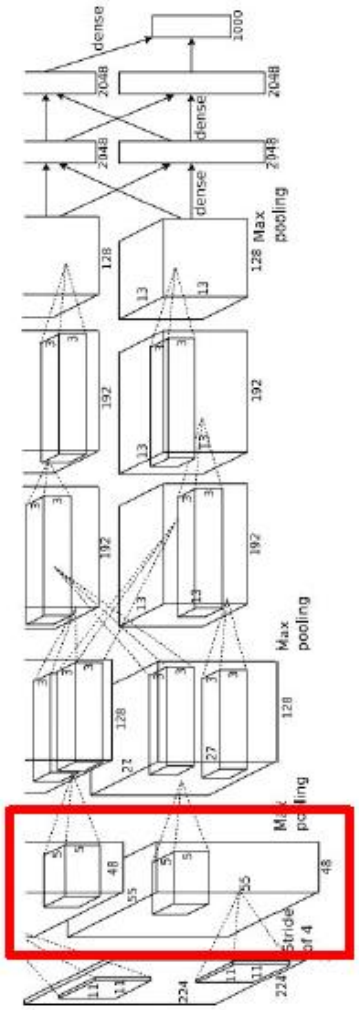


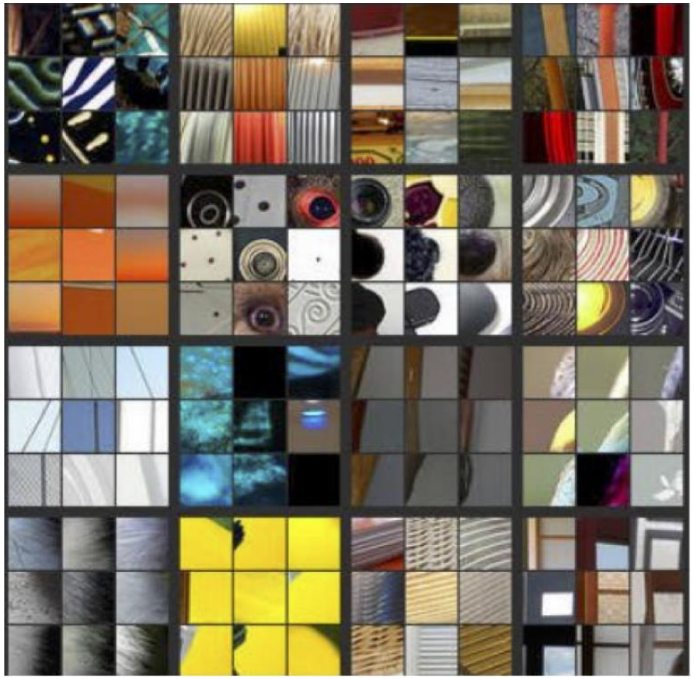
Figure : Filters in the first convolutional layer of Krizhevsky et al



I. Visualizing filters

- Higher-level weights are hard to interpret
  - Visualizing higher-level features by seeing what inputs activate them
  - Pick the images in the training set which active a unit most strongly

Layer 1



Layer 3

I. Visualizing filters

- Higher-level weights are hard to interpret
  - Visualizing higher-level features by seeing what inputs activate them
  - Pick the images in the training set which active a unit most strongly

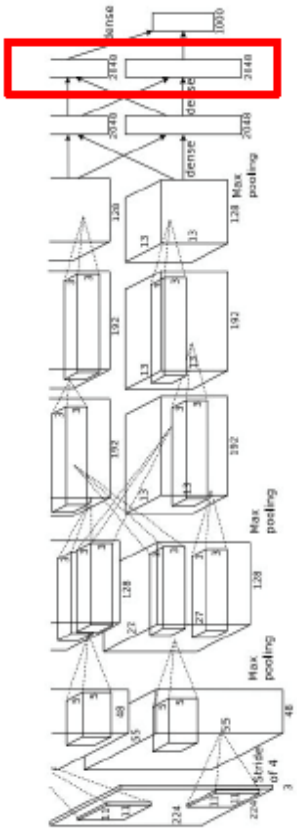
Layer 4

Layer 5

II. Visualizing neural activations

- Understand the neural coding by looking into images

FC7 layer

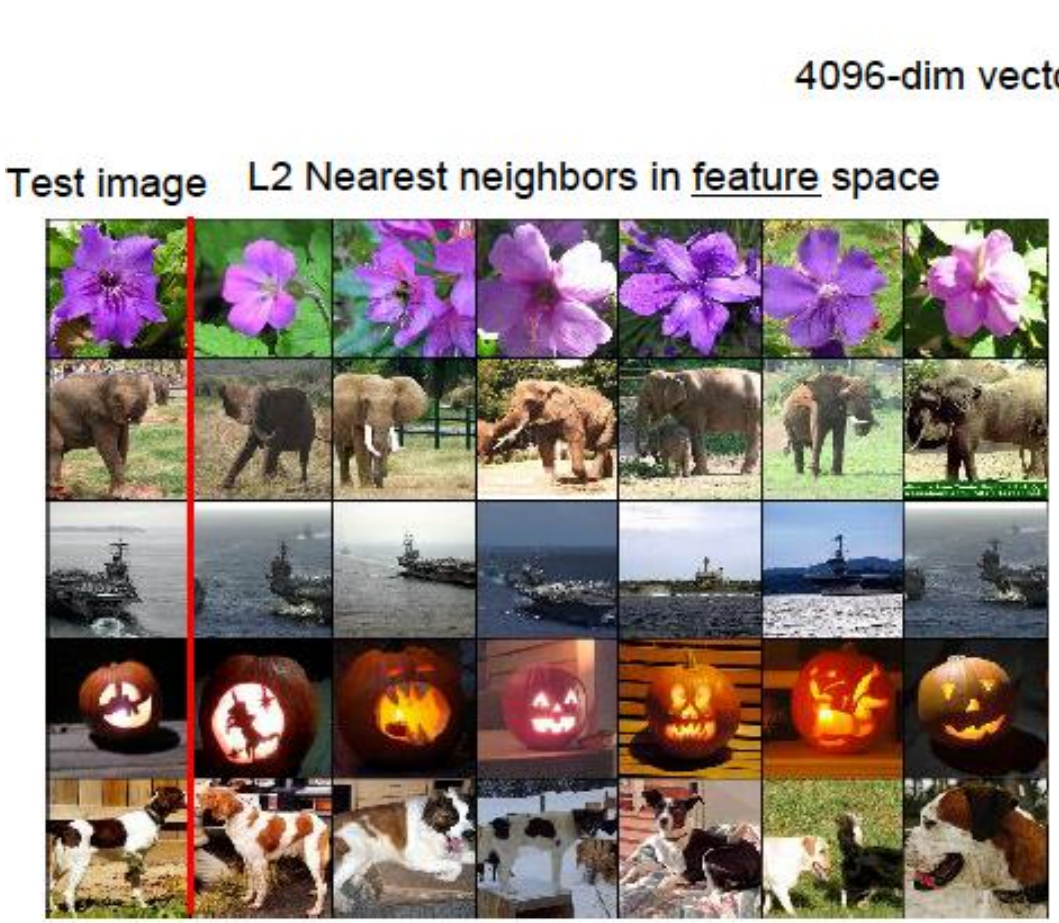


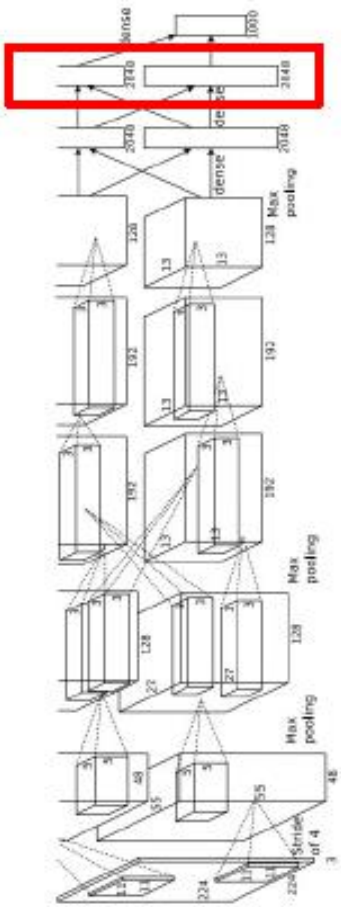
4096-dimensional feature vector for an image (layer immediately before the classifier)

Run the network on many images, collect the feature vectors

II. Visualizing neural activations

- Check the k-nearest neighbors in the feature space

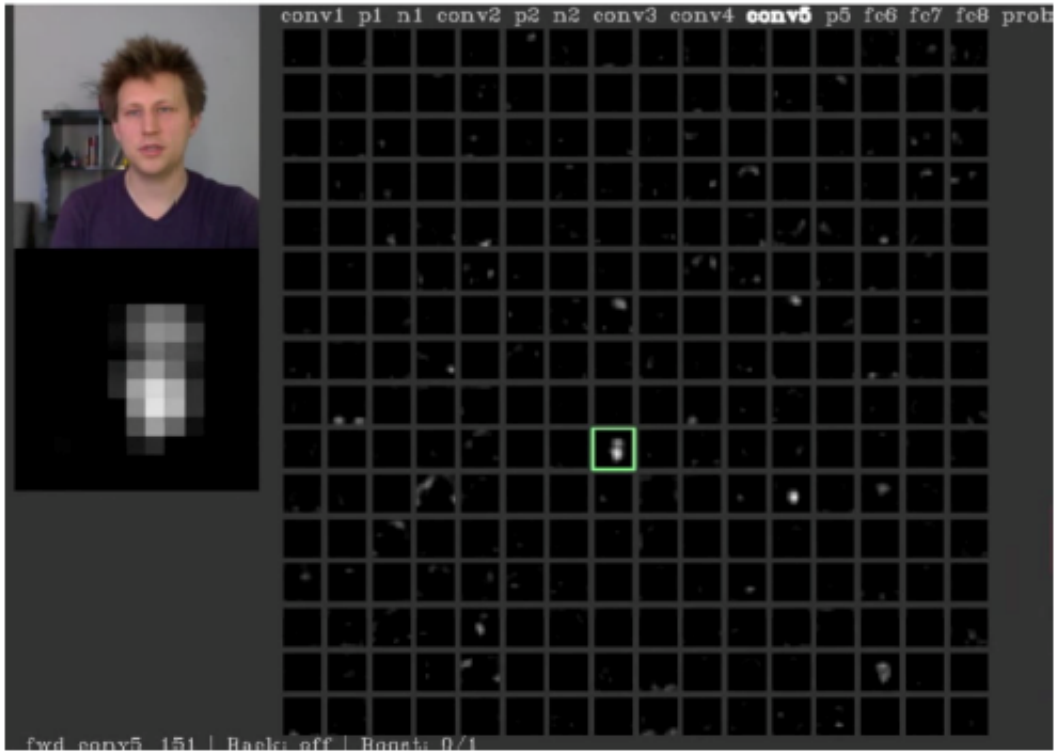




II. Visualizing neural activations

- Feature response maps

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



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

Visualizing image patches

- Higher layers seem to capture more abstract, semantic information.
- But problems?
  - Can’t tell what the neuron is actually responding to in the image
  - Reading too much into results
  - Any patterns outside of training data?
- Alternatively, we can use the sensitivity analysis
  - Use input gradients to diagnose what the neuron responds to
  - I. See how to change an image to increase a unit's activation
  - II. Optimize an image from scratch to increase a unit's activation

Outline

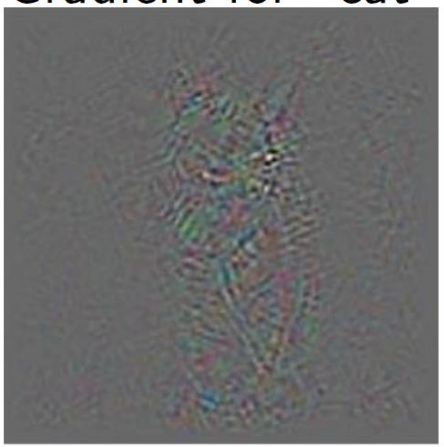
- Understanding CNN through visualization
  - Visualizing filters: Network weights
  - Visualizing neural activations: Network outputs
  - Visualizing sensitivities: Network inputs
- Case studies
  - Adversarial examples
  - Neural texture synthesis and style transfer

I. Visualizing input gradient

- Take a trained object classification network (AlexNet) and compute the gradient of $\log P(y = \text{“}cat\text{”} | \mathbf{x})$

Original image

Gradient for “cat”



- Input gradients can be hard to interpret
  - Part of it is that the network tries to detect cats everywhere; a pixel may be consistent with cats in one location, but inconsistent with cats in other locations.

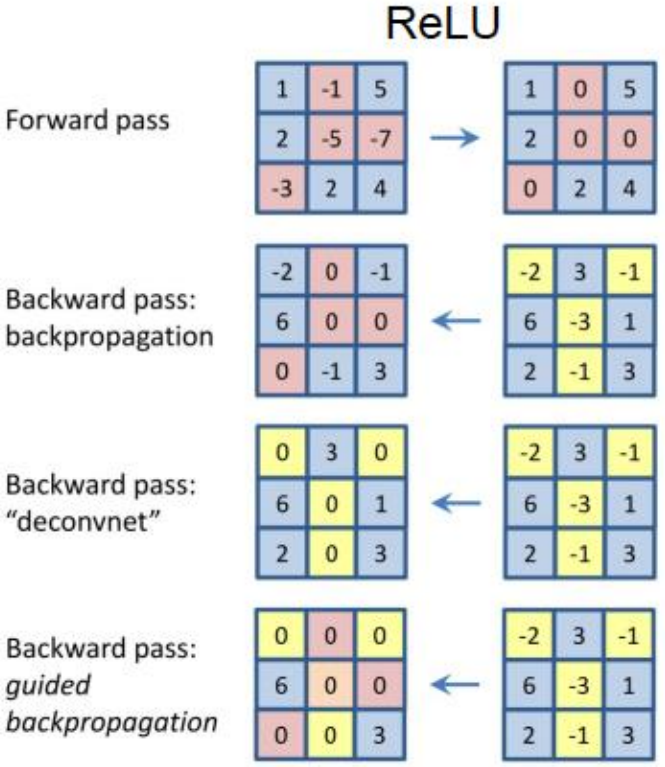
I. Visualizing input gradient

- Guided backprop is a total hack to prevent this cancellation
- Do the backward pass as normal, but apply the ReLU nonlinearity to all the activation error signals

$$y = \mathrm{ReLU}(z) \qquad \bar{z} = \begin{cases} \bar{y} & \text{if } z > 0 \text{ and } \bar{y} > 0 \\ 0 & \text{otherwise} \end{cases}$$

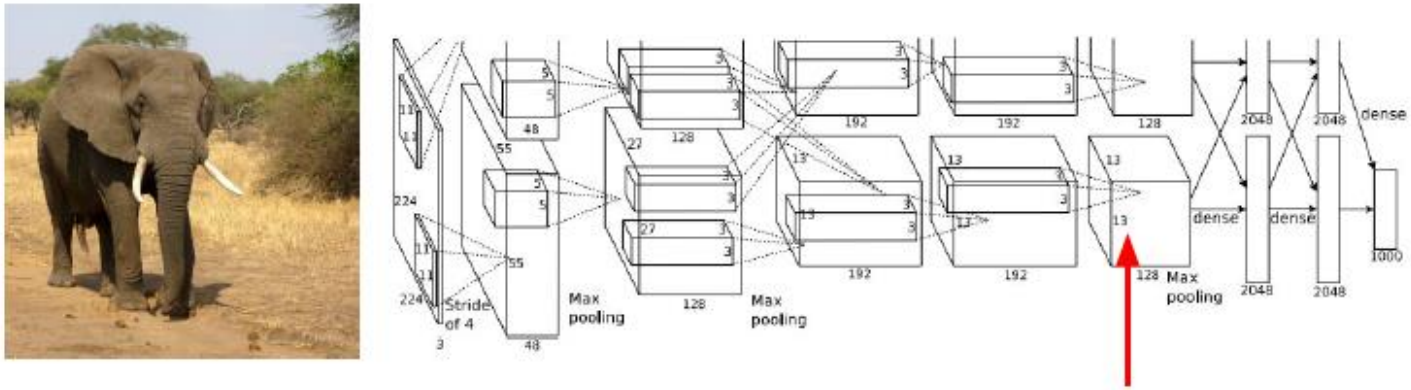
  - What excites a given neuron

Guided Backprop



I. Visualizing input gradient

- Intermediate features via Backprop



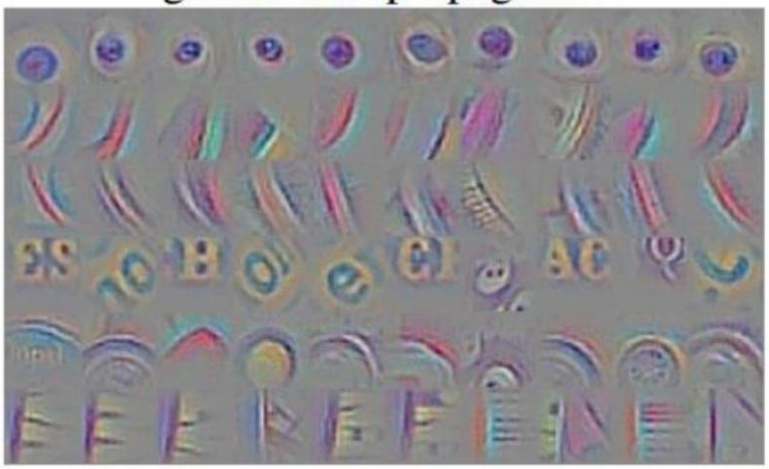
Pick a single intermediate neuron, e.g. one value in 128 x 13 x 13 conv5 feature map

Compute gradient of neuron value with respect to image pixels

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

I. Visualizing input gradient

guided backpropagation



corresponding image crops

guided backpropagation

corresponding image crops

Springerberg et al, Striving for Simplicity: The All Convolutional Net (ICLR 2015 workshops)

I. Visualizing input gradient

- Object detectors in CNNs [Zhou et al, ICLR 2015]
  - Visualizing actual receptive fields

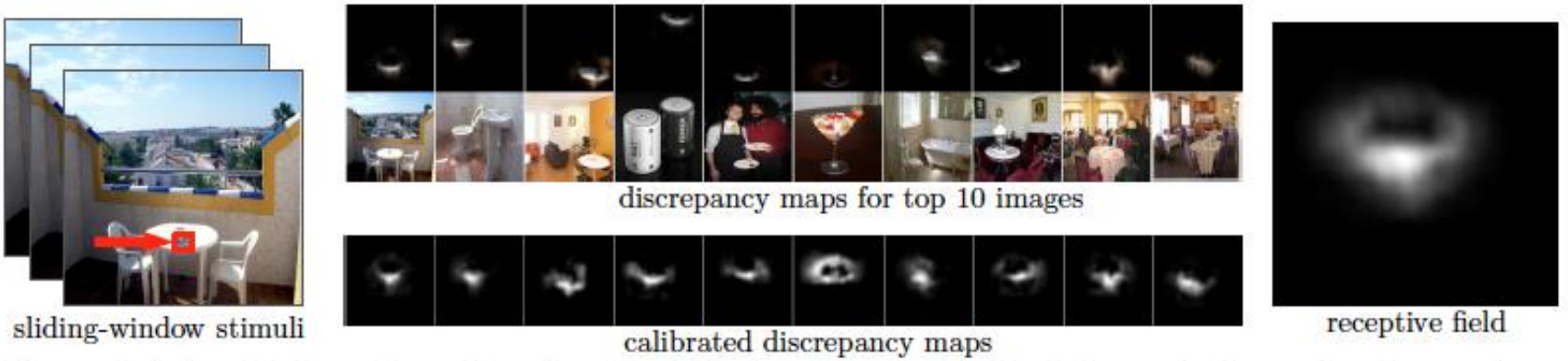


Figure 3: The pipeline for estimating the RF of each unit. Each sliding-window stimuli contains a small randomized patch (example indicated by red arrow) at different spatial locations. By comparing the activation response of the sliding-window stimuli with the activation response of the original image, we obtain a discrepancy map for each image (middle top). By summing up the calibrated discrepancy maps (middle bottom) for the top ranked images, we obtain the actual RF of that unit (right).

II. Synthesizing input images

- Gradient ascent on an image to maximize the activation of a given neuron

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

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

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

Neuron value

Natural image regularizer

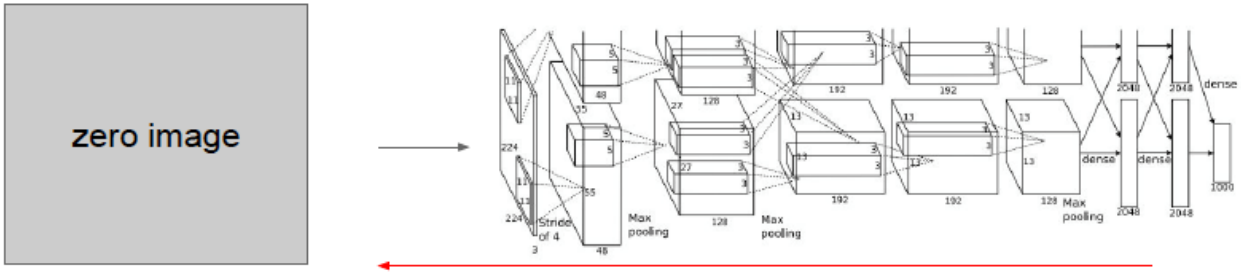
II. Synthesizing input images

- Gradient ascent: requires a few tricks
  - https://distill.pub/2017/feature-visualization/
  - Example for the output neurons

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

score for class c (before Softmax)

1. Initialize image to zeros



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

II. Synthesizing input images

- Dataset examples vs. optimized input

Baseball—or stripes?
mixed4a, Unit 6

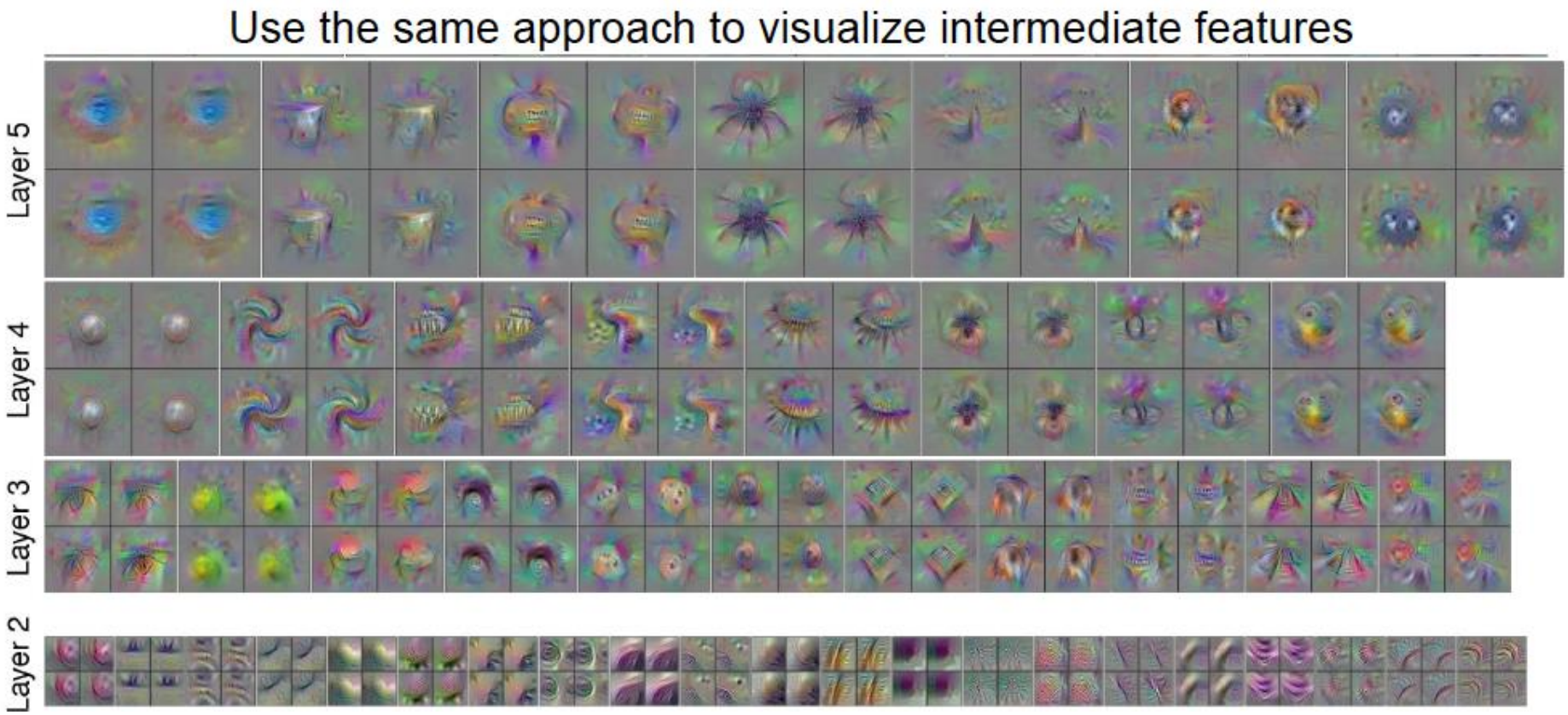
Animal faces—or snouts?
mixed4a, Unit 240

Clouds—or fluffiness?
mixed4a, Unit 453

Buildings—or sky?
mixed4a, Unit 492

II. Synthesizing input images

- Higher layers in the network often learn higher-level, more interpretable representations



Deep Dream

- Start with an image, and run a ConvNet on it.
- Pick a layer in the network.
- Change the image such that units which were already highly activated get activated even more strongly. “Rich get richer.“
- Repeat
- This will accentuate whatever features of an image already kind of resemble the object.

Deep Dream

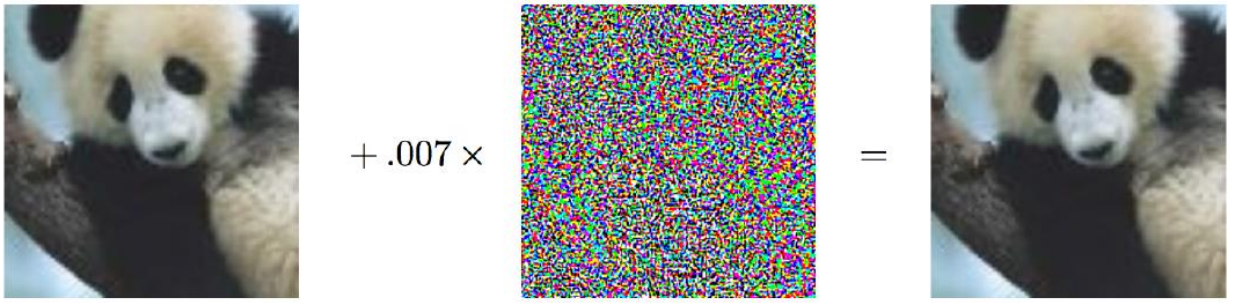
Deep Dream

Outline

- Understanding CNN through visualization
  - Visualizing filters: Network weights
  - Visualizing neural activations: Network outputs
  - Visualizing sensitivities: Network inputs
- Case studies
  - Adversarial examples
  - Neural texture synthesis and style transfer

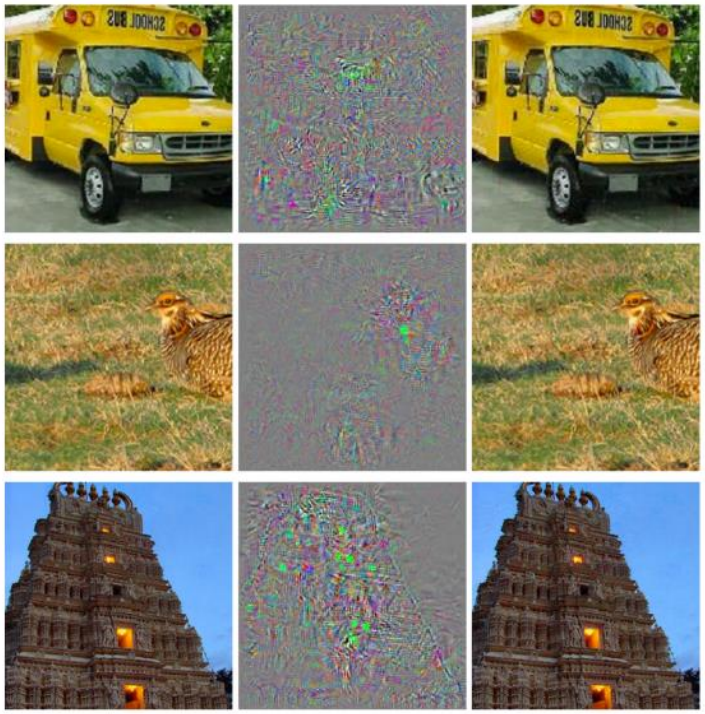
Adversarial Examples

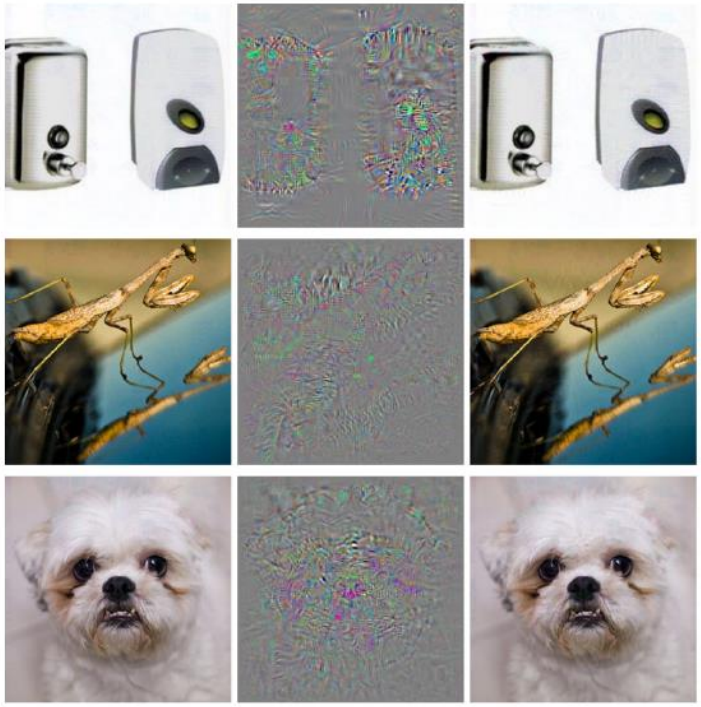
- Surprising findings: adversarial inputs
  - Inputs optimized to fool an algorithm
- Given an image for one category (e.g., “cat”), compute the input gradient to maximize the network’s output for a different category (e.g., “dog”)
  - Perturb the image very slightly in this direction and the network will change its prediction
  - Fast gradient sign method: take the sign of the entries in the gradient



Adversarial Examples

- The following adversarial examples are misclassified as ostriches (Middle = perturbation x 10)





Adversarial Examples

- 2013: ha ha, how cute!
  - “Intriguing Properties of Neural Networks”
- 2018: serious security threat
  - Nobody has found a reliable method yet to defend against them
    - 7 of 8 proposed defenses accepted to ICLR 2018 were cracked within days
  - Adversarial examples transfer to different networks trained on a totally separate training set
  - You don’t need access to the original network; you can train up a new network to match its predictions, and then construct adversarial examples for that
    - Attack carried out against proprietary classification networks accessed using prediction APIs (MetaMind, Amazon, Google)

Adversarial Examples

- 2013: ha ha, how cute!
  - “Intriguing Properties of Neural Networks”
- 2018: serious security threat
  - Nobody has found a reliable method yet to defend against them
    - 7 of 8 proposed defenses accepted to ICLR 2018 were cracked within days
  - Adversarial examples transfer to different networks trained on a totally separate training set
  - You don’t need access to the original network; you can train up a new network to match its predictions, and then construct adversarial examples for that
    - Attack carried out against proprietary classification networks accessed using prediction APIs (MetaMind, Amazon, Google)

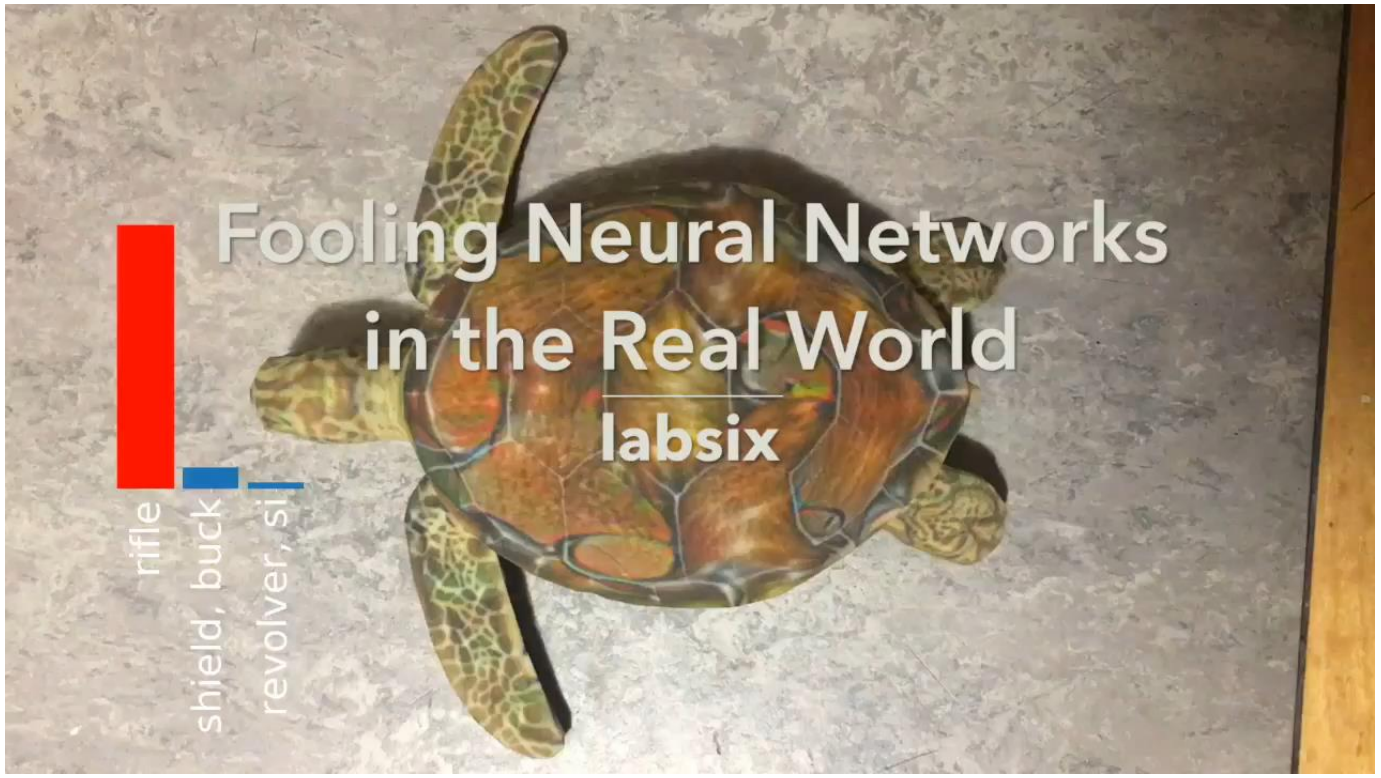
Adversarial Examples

- You can print out an adversarial image and take a picture of it, and it still works!

(a) Printout (b) Photo of printout (c) Cropped image

Adversarial Examples

- An adversarial example in the physical world
  - Network thinks it is a gun from a variety of viewing angles



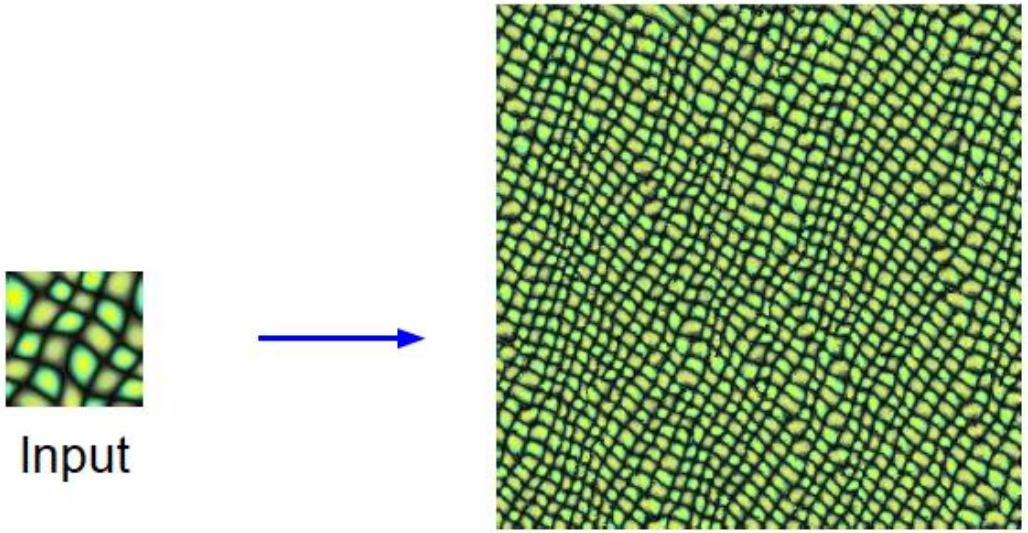
Outline

- Understanding CNN through visualization
  - Visualizing filters: Network weights
  - Visualizing neural activations: Network outputs
  - Visualizing sensitivities: Network inputs
- Case studies
  - Adversarial examples
  - Neural texture synthesis and style transfer

Texture Synthesis

- Problem setup

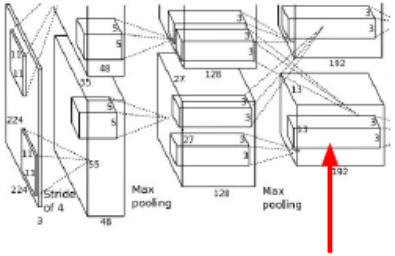
Given a sample patch of some texture, can we generate a bigger image of the same texture?

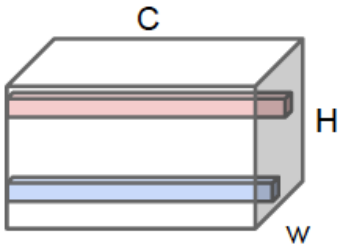


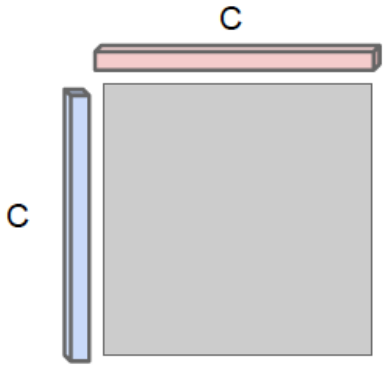
Texture Synthesis

- CNN-based modeling of image statistics

This image is in the public domain.







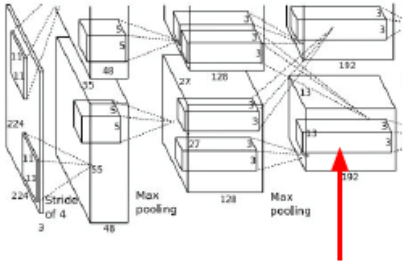
Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

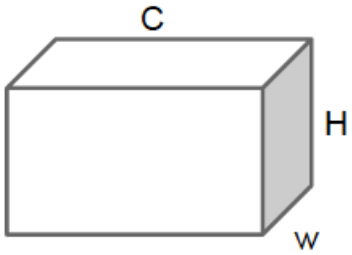
Outer product of two C-dimensional vectors gives C x C matrix measuring co-occurrence

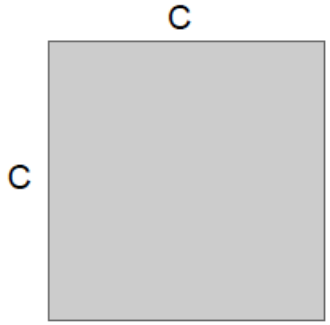
Texture Synthesis

- CNN-based modeling of image statistics

This image is in the public domain.







Gram Matrix

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

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

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

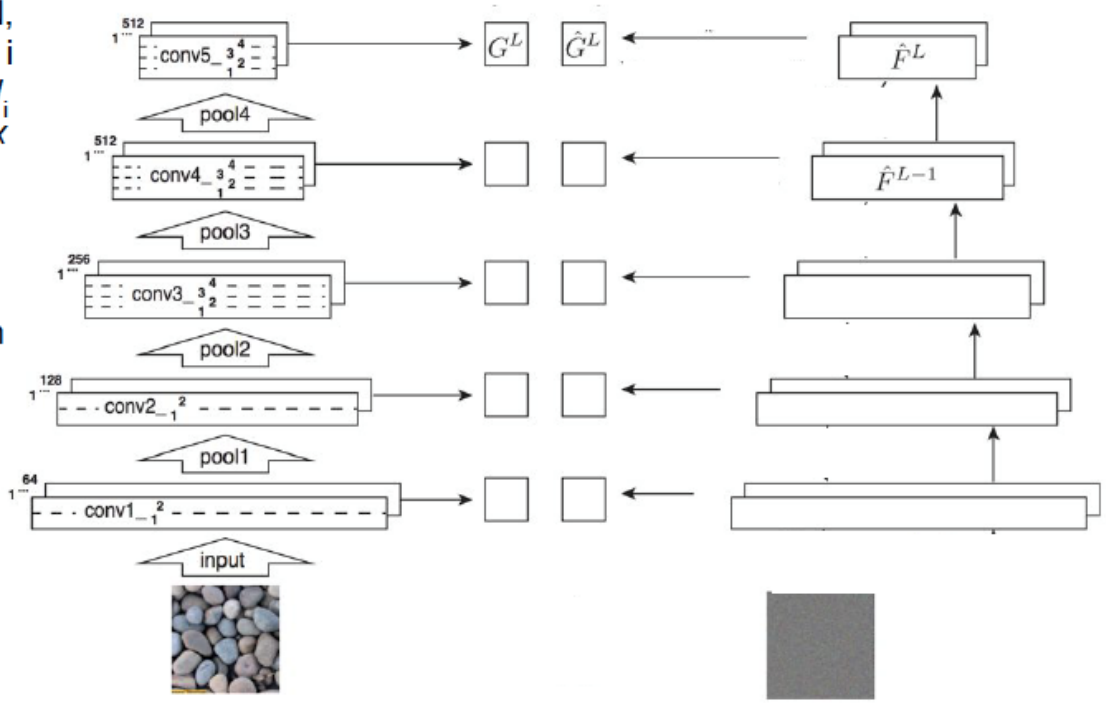
Texture Synthesis

- Neural texture synthesis

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

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

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



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

Texture Synthesis

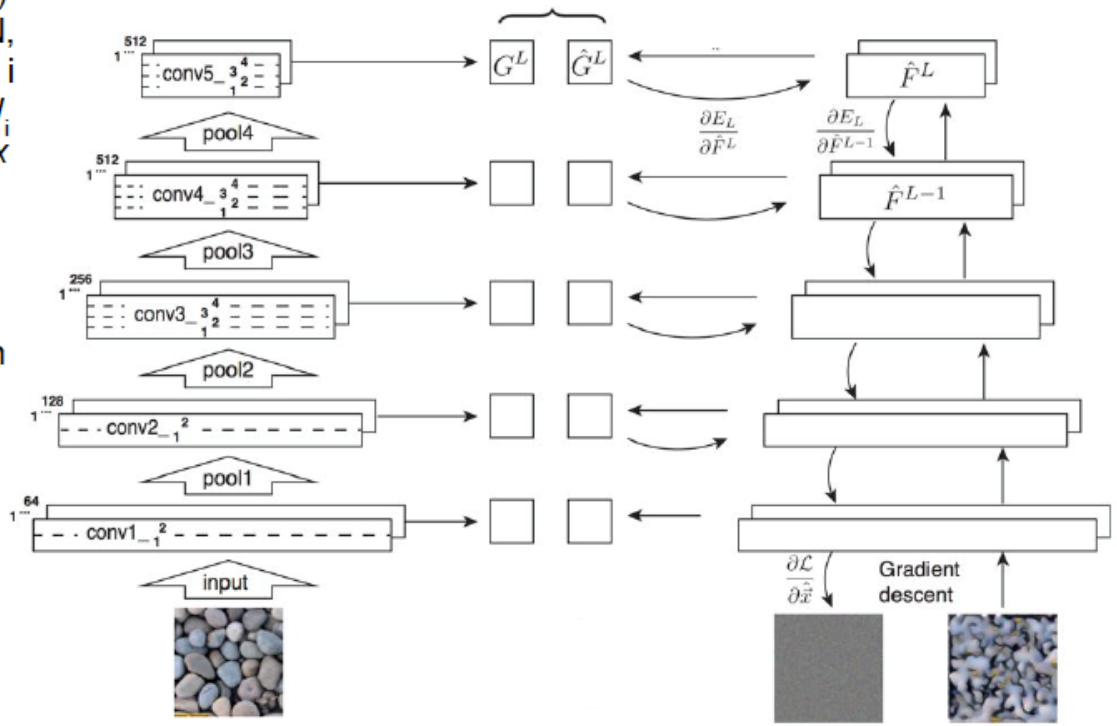
- Neural texture synthesis

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

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

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

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



Neural Style Transfer

- Problem setup

Content Image

This image is licensed under CC-BY 3.0

+

Style Image

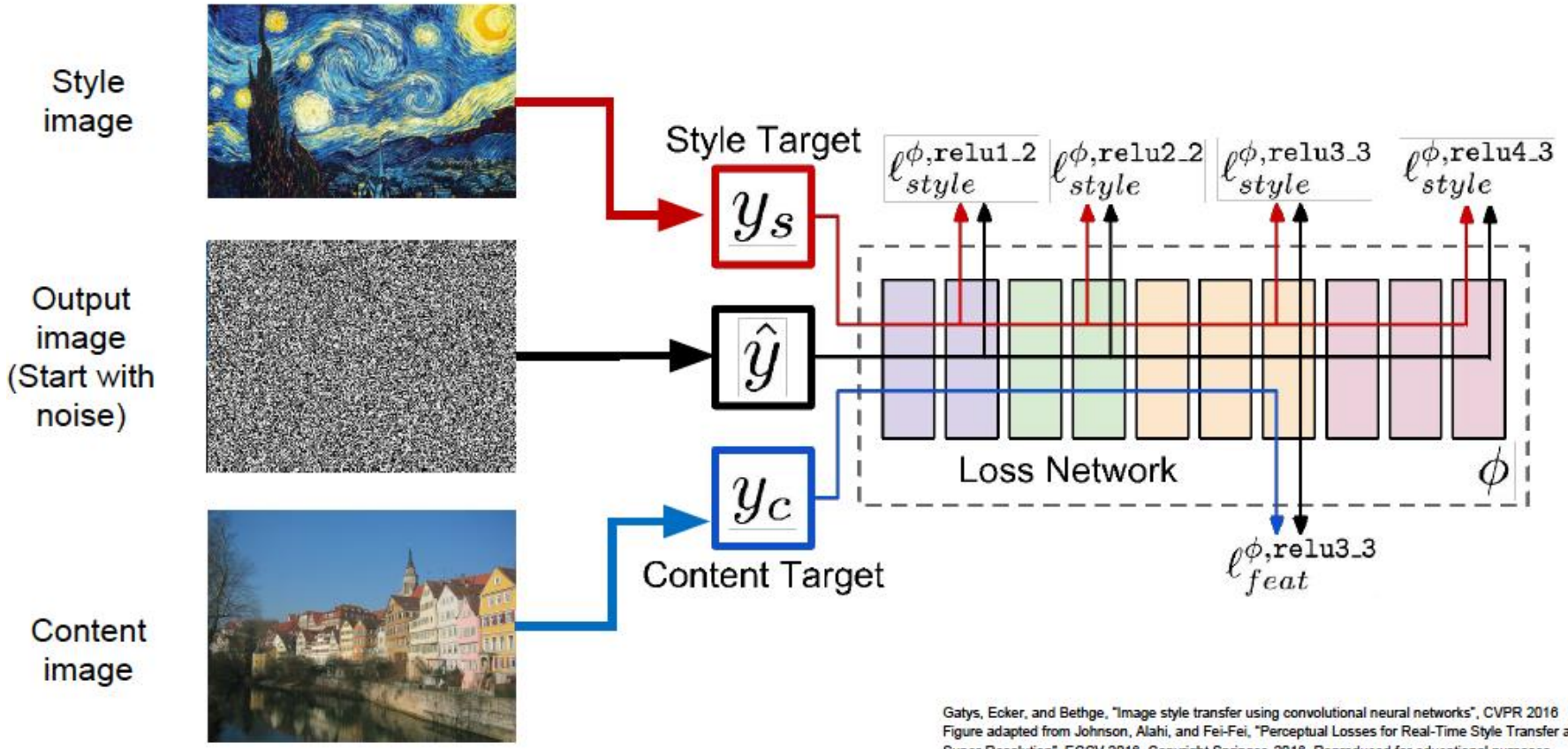
Starry Night by Van Gogh is in the public domain

=

Style Transfer!

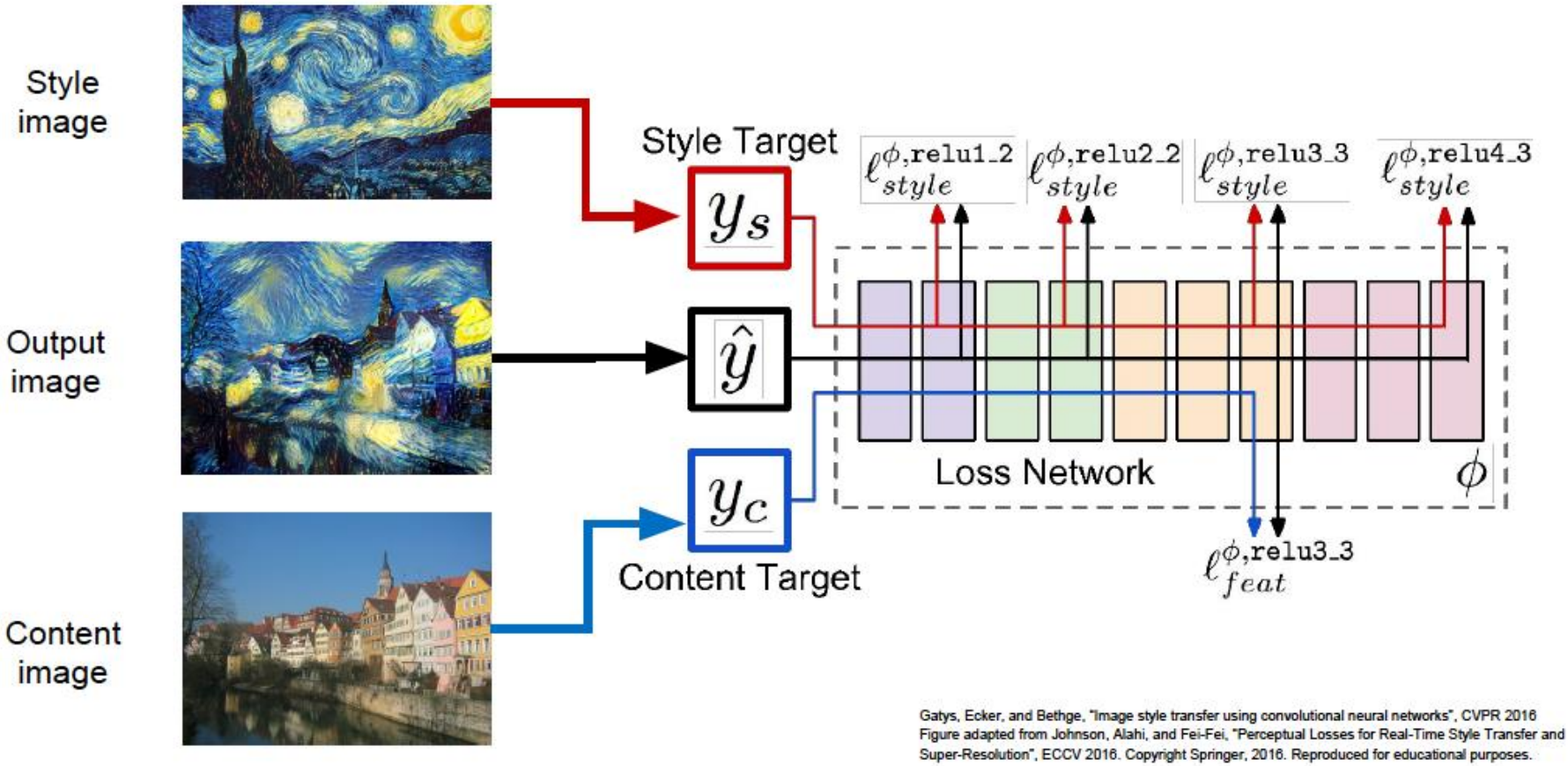
This image copyright Justin Johnson, 2015. Reproduced with permission.

Neural Style Transfer



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

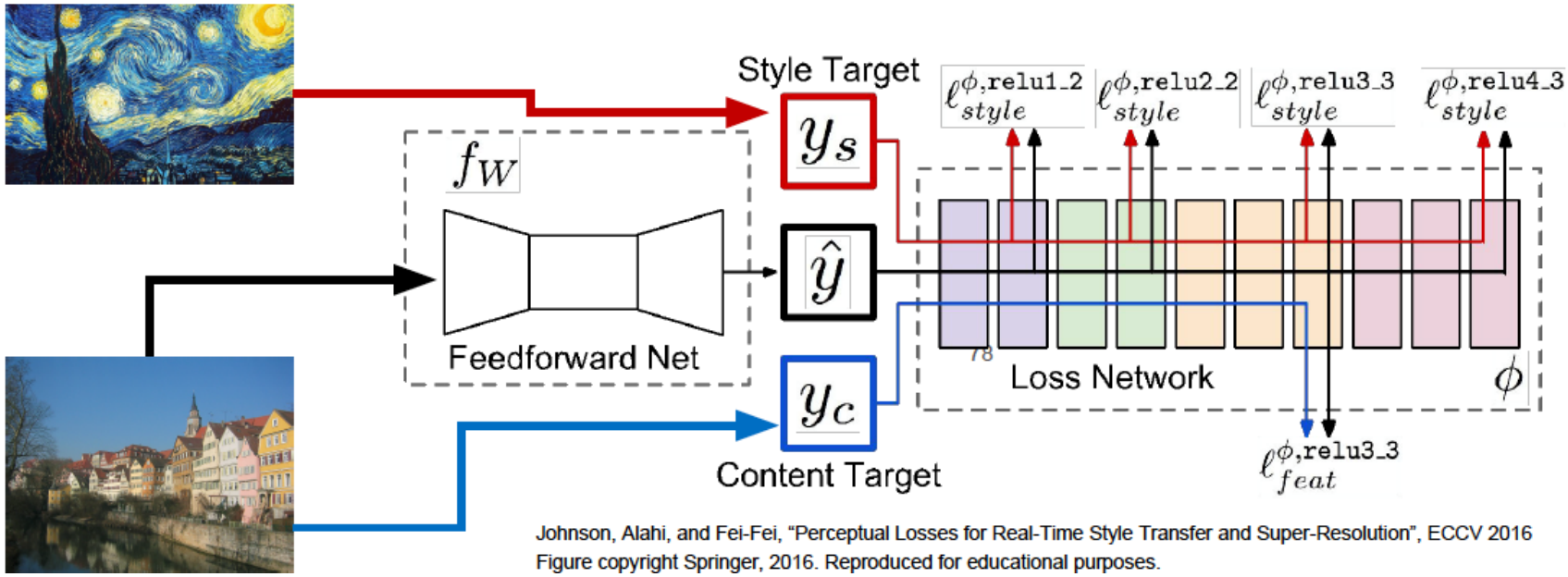
Neural Style Transfer



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

Fast Style Transfer

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



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

Summary

- CNNs in computer vision
  - Many and more applications
  - Still lack of deep understanding

- Next time:
  - Recurrent Neural Networks