



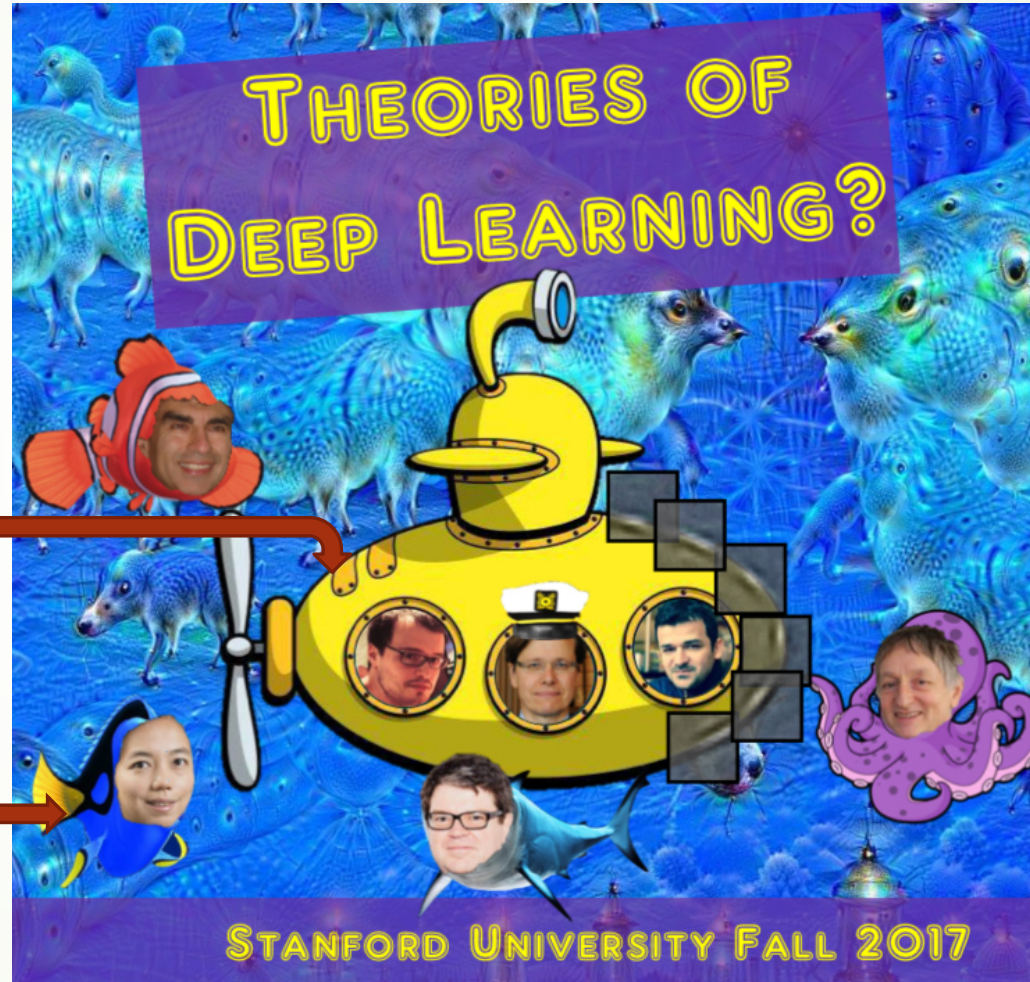
# Topics on CNN: Visualization, Transfer Learning, Neural Style, and Adversarial Examples

1

Yuan YAO

HKUST

# Acknowledgement



<https://stats385.github.io/>

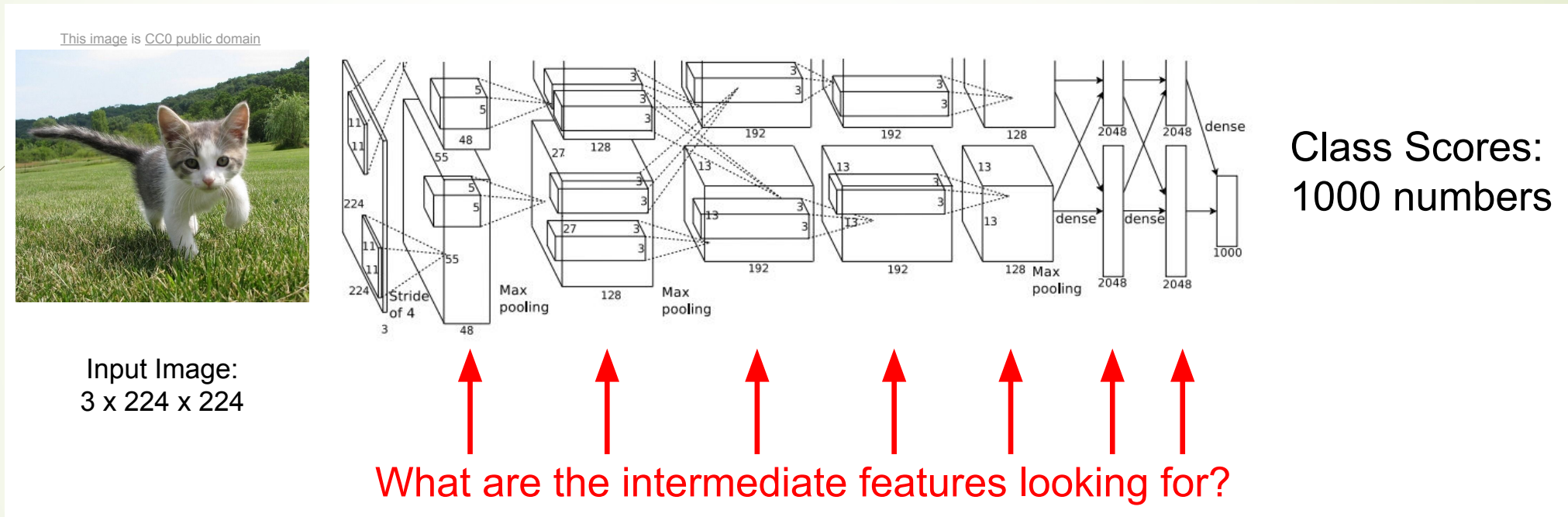
<http://cs231n.github.io/>

A following-up course at HKUST: <https://deeplearning-math.github.io/>



# Visualizing Convolutional Networks

# Understanding intermediate neurons?



# Visualizing CNN Features: Gradient Ascent

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

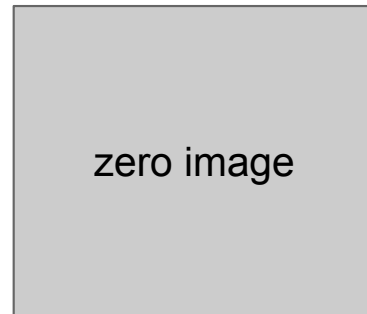
$$I^* = \arg \max_I f(I) + R(I)$$

Neuron value

Natural image regularizer

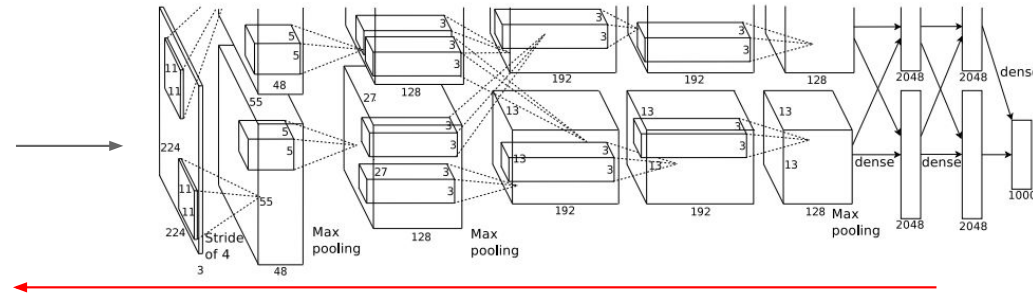
# Visualizing CNN Features: Gradient Ascent

1. Initialize image to zeros



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

score for class  $c$  (before Softmax)



Repeat:

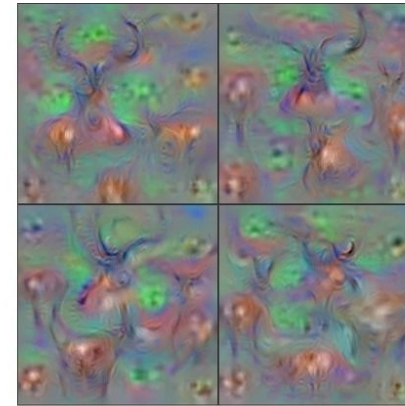
2. Forward image to compute current scores
3. Backprop to get gradient of neuron value with respect to image pixels
4. Make a small update to the image

# Visualizing CNN Features: Gradient Ascent

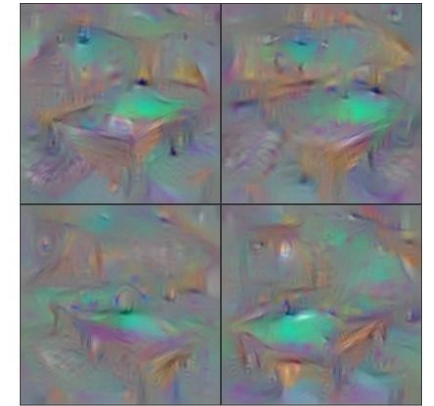
$$\arg \max_I S_c(I) - \lambda \|I\|_2^2$$

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

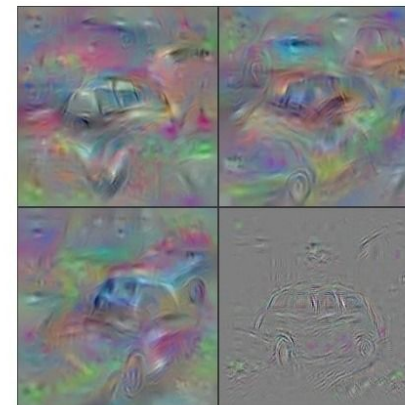
- (1) Gaussian blur image
- (2) Clip pixels with small values to 0
- (3) Clip pixels with small gradients to 0



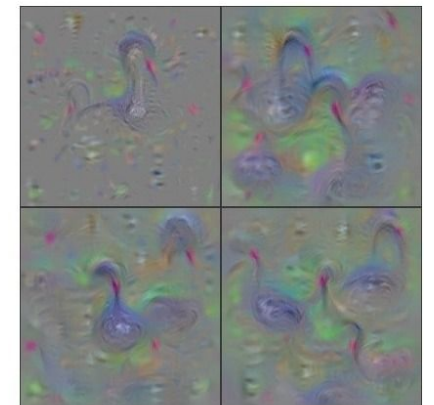
Hartebeest



Billiard Table



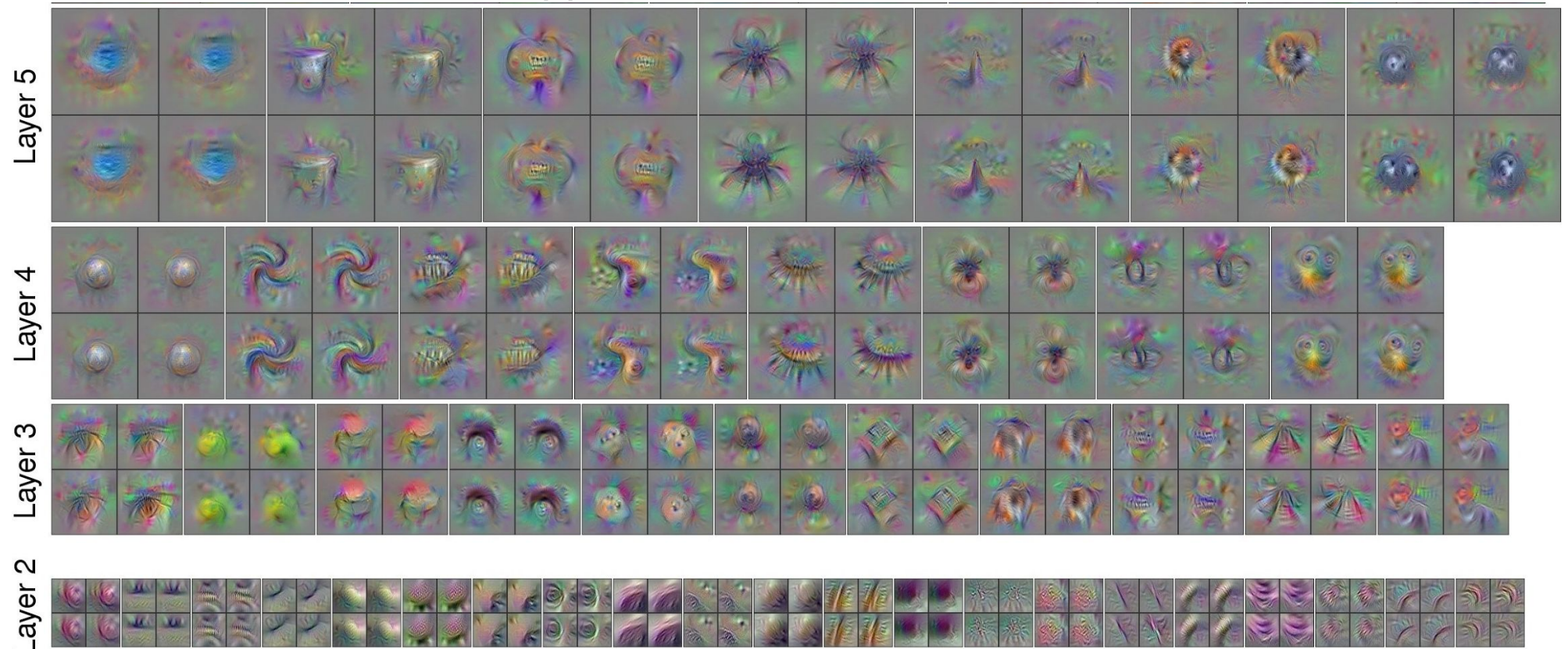
Station Wagon



Black Swan

# Visualizing CNN Features: Gradient Ascent

Use the same approach to visualize intermediate features

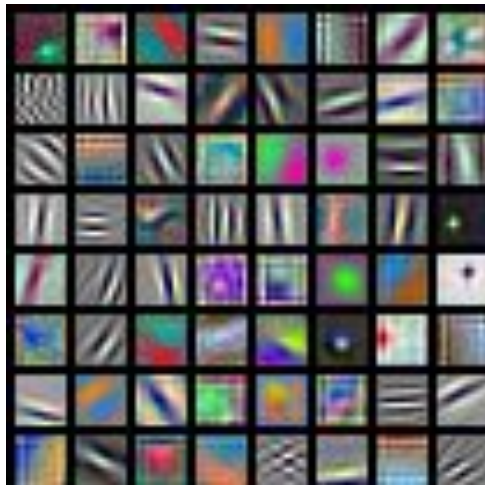


Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.  
Figure copyright Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson, 2014. Reproduced with permission.



# It's easy to visualize early layers

## First Layer: Visualize Filters



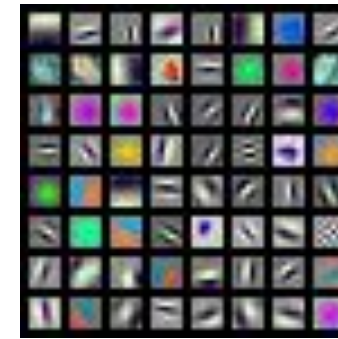
AlexNet:  
 $64 \times 3 \times 11 \times 11$



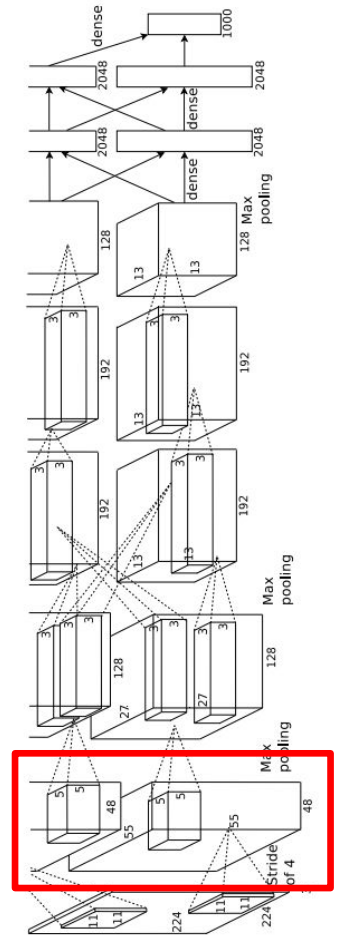
ResNet-18:  
 $64 \times 3 \times 7 \times 7$



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



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



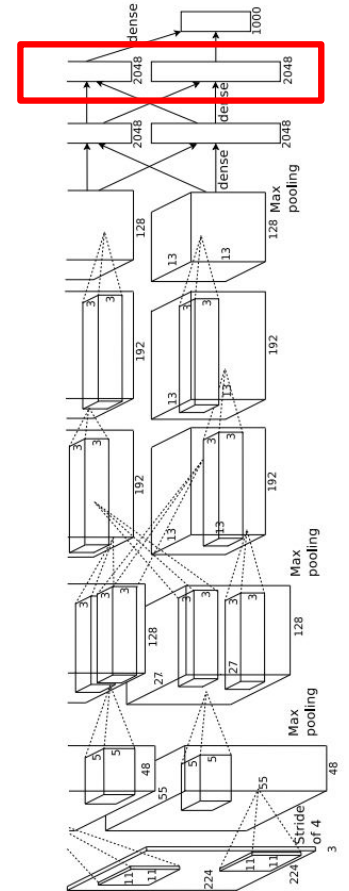
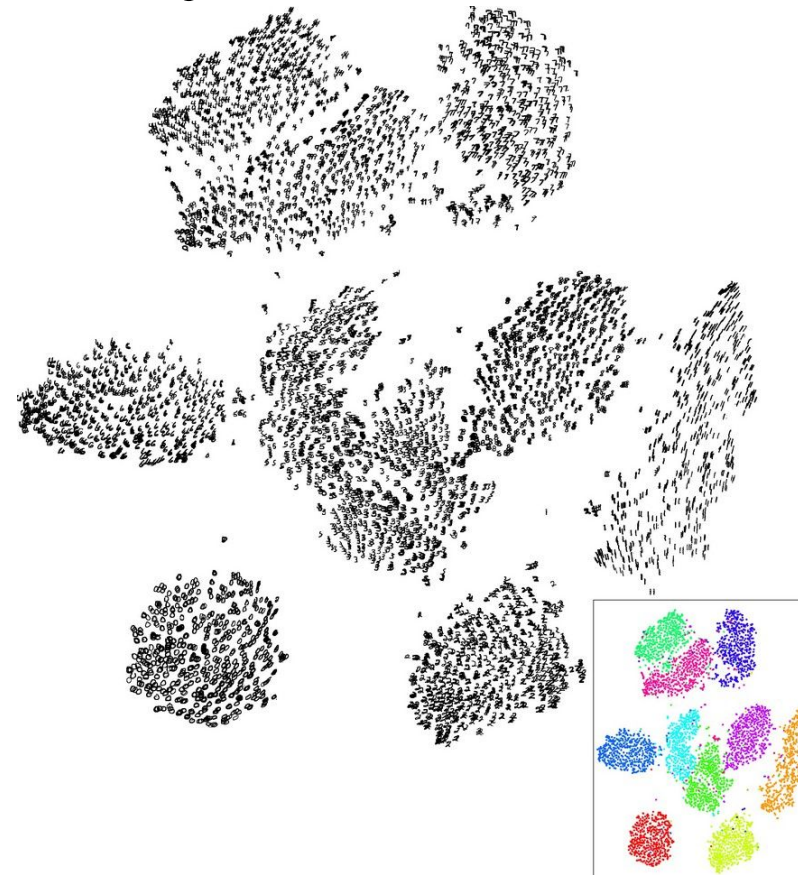
# Last layers are hard to visualize

## Last Layer: Dimensionality Reduction

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

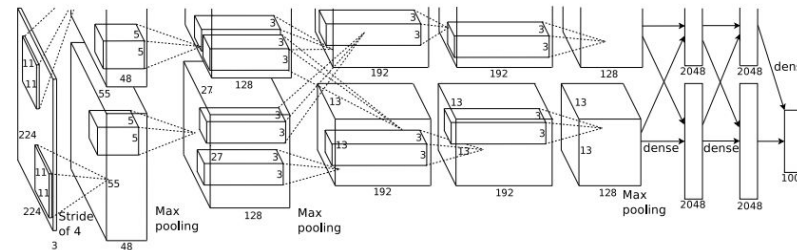
Simple algorithm: Principle Component Analysis (PCA)

More complex: **t-SNE**



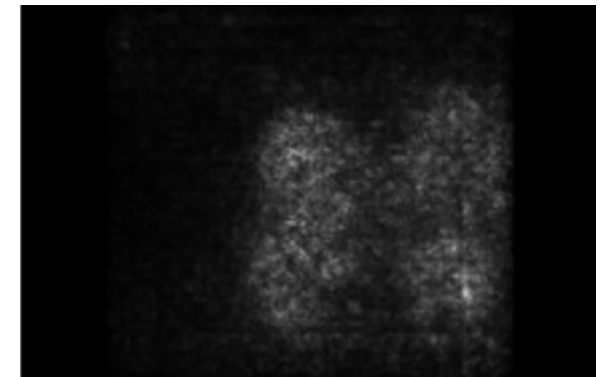
# Saliency Maps

How to tell which pixels matter for classification?



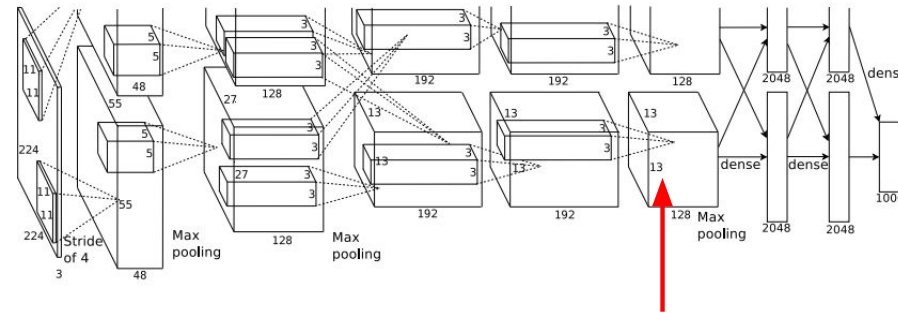
Dog

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



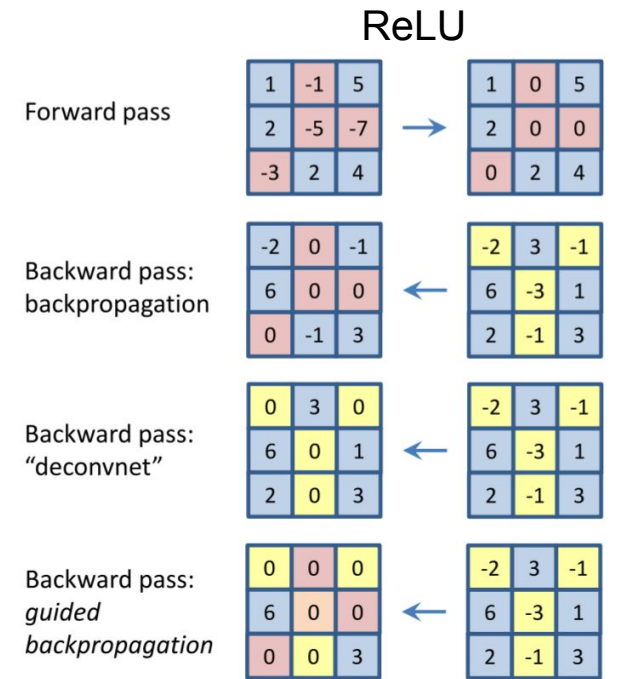
# Guided BP

## Intermediate features via (guided) backprop



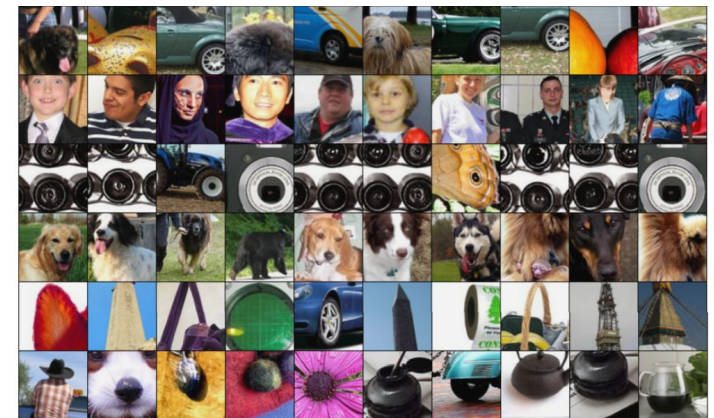
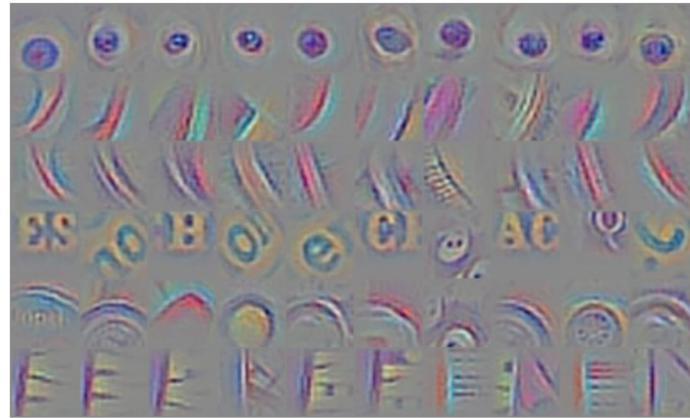
Pick a single intermediate neuron, e.g. one value in 128 x 13 x 13 conv5 feature map

Compute gradient of neuron value with respect to image pixels



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

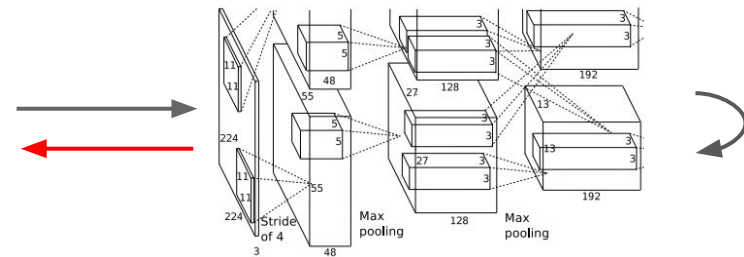
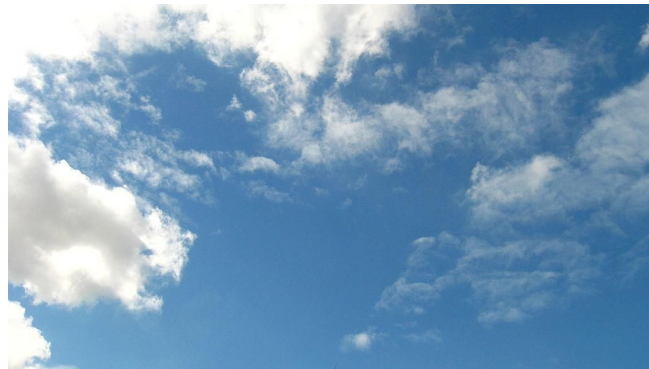
# Intermediate features via Guided BP



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.

# DeepDream: amplifying features

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



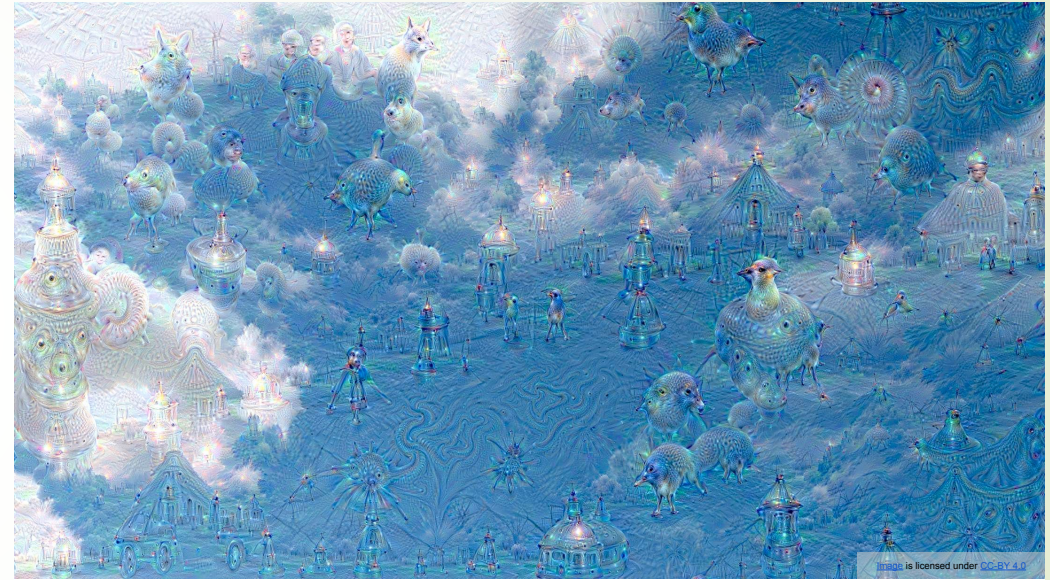
Choose an image and a layer in a CNN; repeat:

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:

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

# Example: DeepDream of Sky



"Admiral Dog!"



"The Pig-Snail"

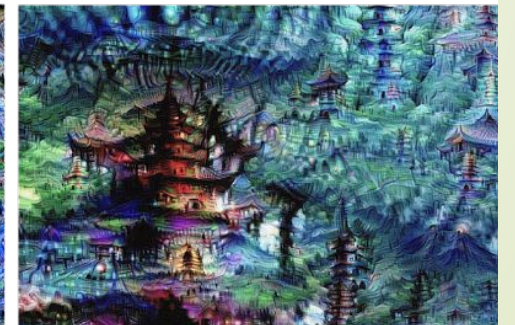
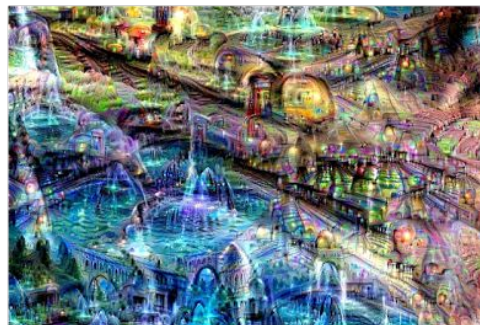
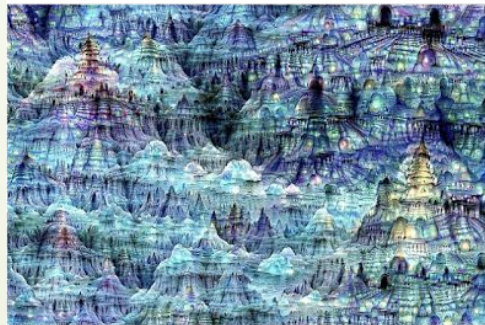


"The Camel-Bird"



"The Dog-Fish"

# More Examples







# Python Notebooks

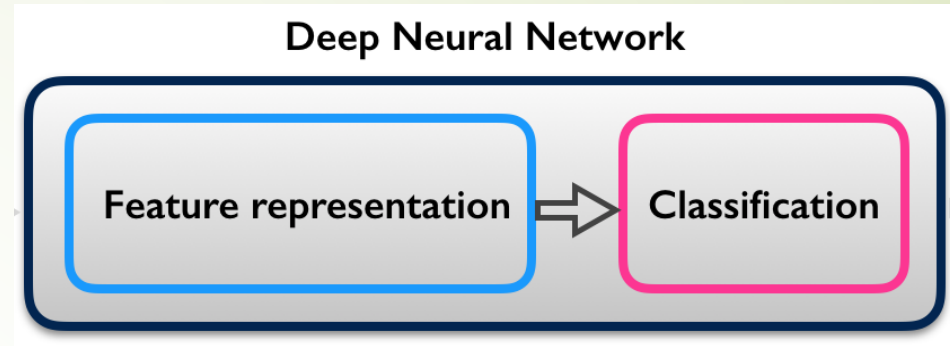


- ▶ An interesting Pytorch Implementation of these visualizatoion methods
  - ▶ <https://github.com/utkuozbulak/pytorch-cnn-visualizations>
- ▶ Some examples demo:
  - ▶ <https://github.com/aifin-hkust/aifin-hkust.github.io/blob/master/2020/notebook/vgg16-visualization.ipynb>
  - ▶ <https://github.com/aifin-hkust/aifin-hkust.github.io/blob/master/2020/notebook/vgg16-heatmap.ipynb>



# Transfer Learning: Feature Extraction and Fine Tuning

# Transfer Learning?



- Filters learned in first layers of a network are transferable from one task to another
- When solving another problem, no need to retrain the lower layers, just fine tune upper ones
- Is this simply due to the large amount of images in ImageNet?
- Does solving many classification problems simultaneously result in features that are more easily transferable?
- Does this imply filters can be learned in unsupervised manner?
- Can we characterize filters mathematically?

# Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014  
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

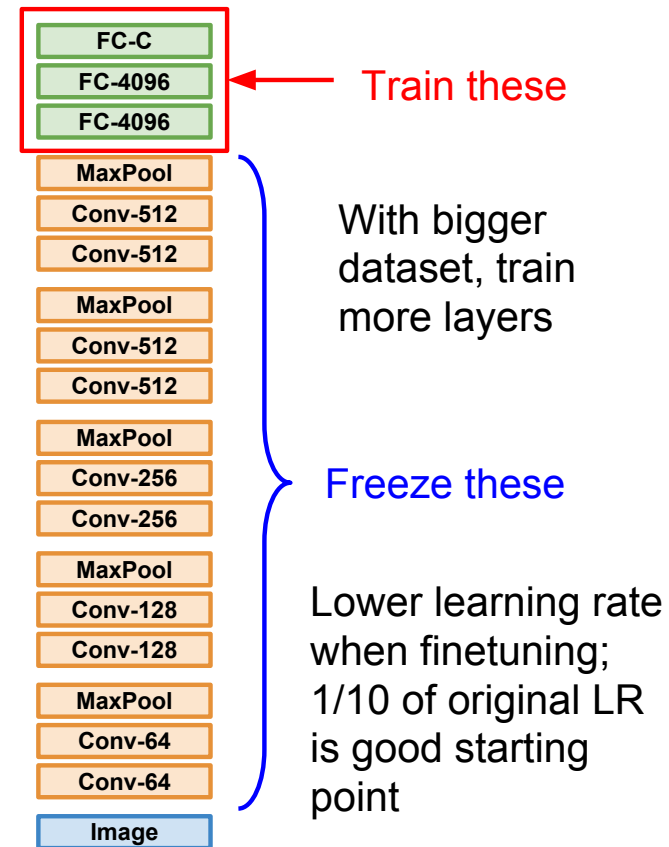
## 1. Train on Imagenet

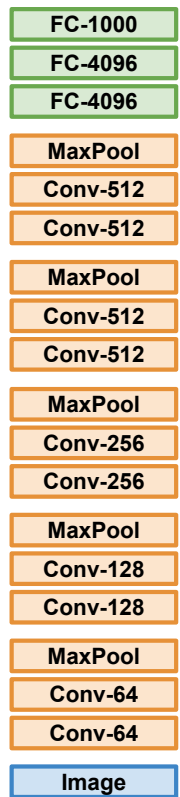


## 2. Small Dataset (C classes)



## 3. Bigger dataset





More specific

More generic

	<b>very similar dataset</b>	<b>very different dataset</b>
<b>very little data</b>	Use Linear Classifier on top layer	You're in trouble... Try linear classifier from different stages
<b>quite a lot of data</b>	Finetune a few layers	Finetune a larger number of layers



# Summary



- ▶ Feature Extraction vs. Fine-Tuning:
  - ▶ Feature extraction usually refers to freeze the bottom (early layers) and retrain the top (last) layer
  - ▶ Fine-Tuning usually refers to retrain the last few layers or the whole network initialized from pretrained parameters
  - ▶ They are both called transfer learning
- ▶ Jupyter notebook examples with pytorch:
  - ▶ [https://github.com/aifin-hkust/aifin-hkust.github.io/blob/master/2020/notebook/finetuning\\_resnet.ipynb](https://github.com/aifin-hkust/aifin-hkust.github.io/blob/master/2020/notebook/finetuning_resnet.ipynb)

A decorative graphic on the left side of the slide. It features a solid red arrow pointing to the right, positioned horizontally. Behind the arrow and extending upwards and to the right are several thin, dark grey, curved lines that resemble stylized grass or reeds. The background is a light, neutral color with a subtle gradient.

# Neural Style

# Example: The Noname Lake in PKU

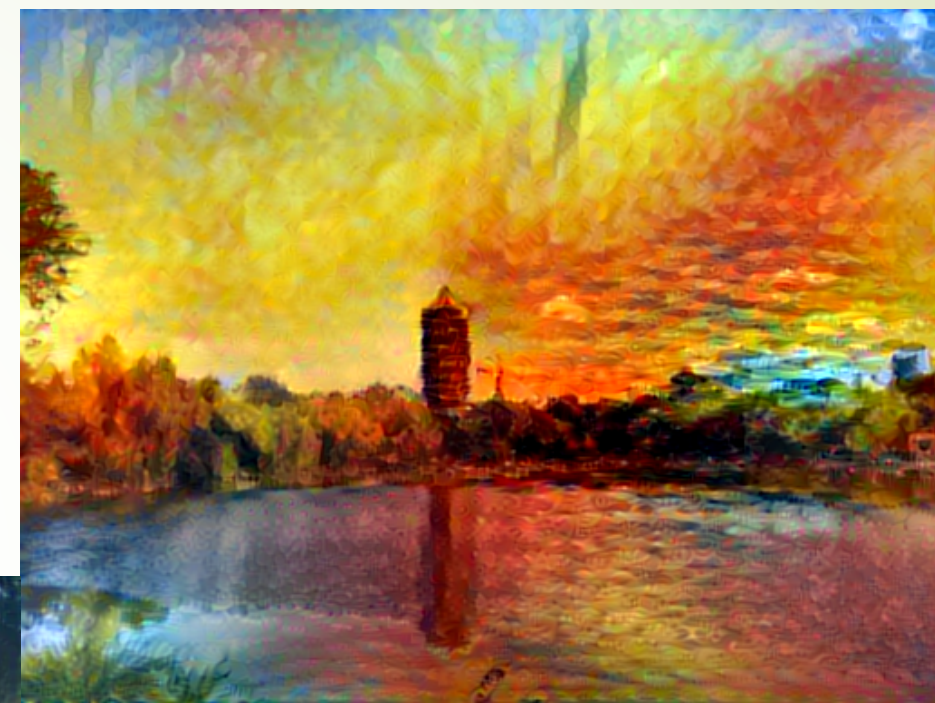
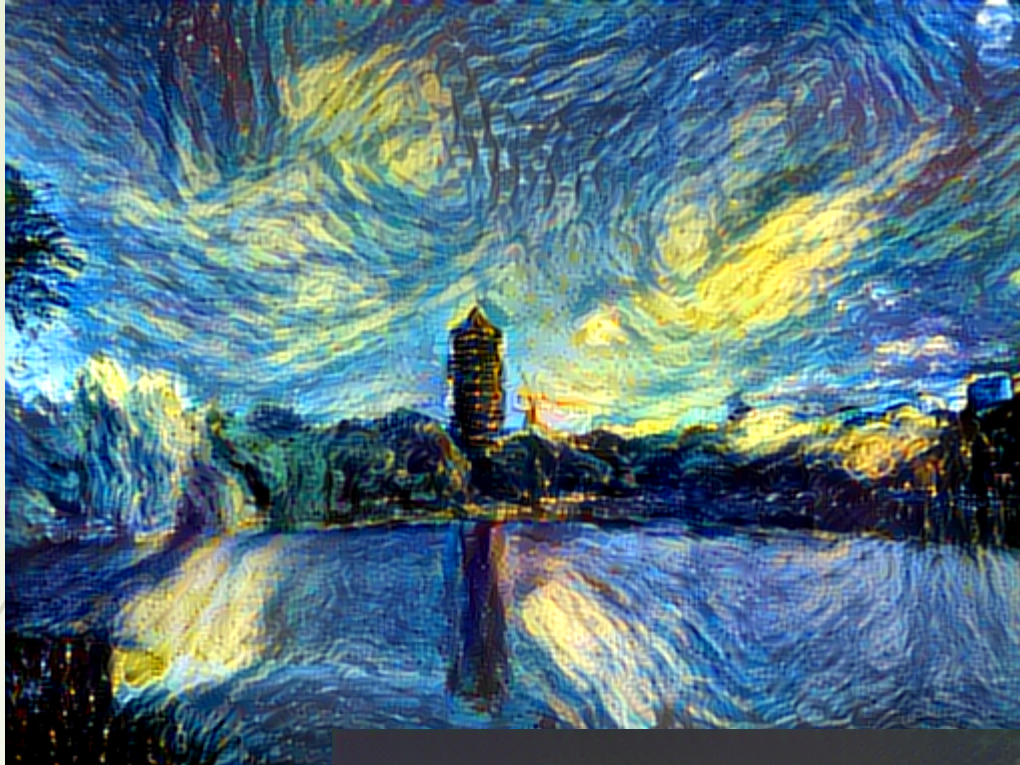




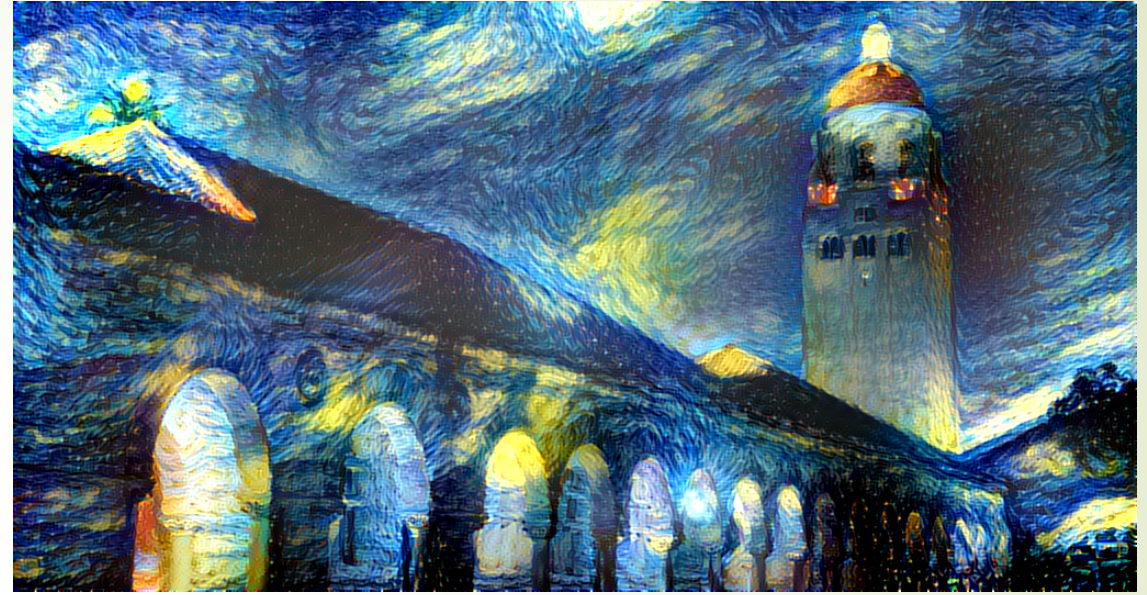


Left: Vincent Van Gogh, *Starry Night*  
Right: Claude Monet, *Twilight Venice*  
Bottom: William Turner, *Ship Wreck*





Application of Deep Learning:  
Content-Style synthetic  
pictures  
By "neural-style"

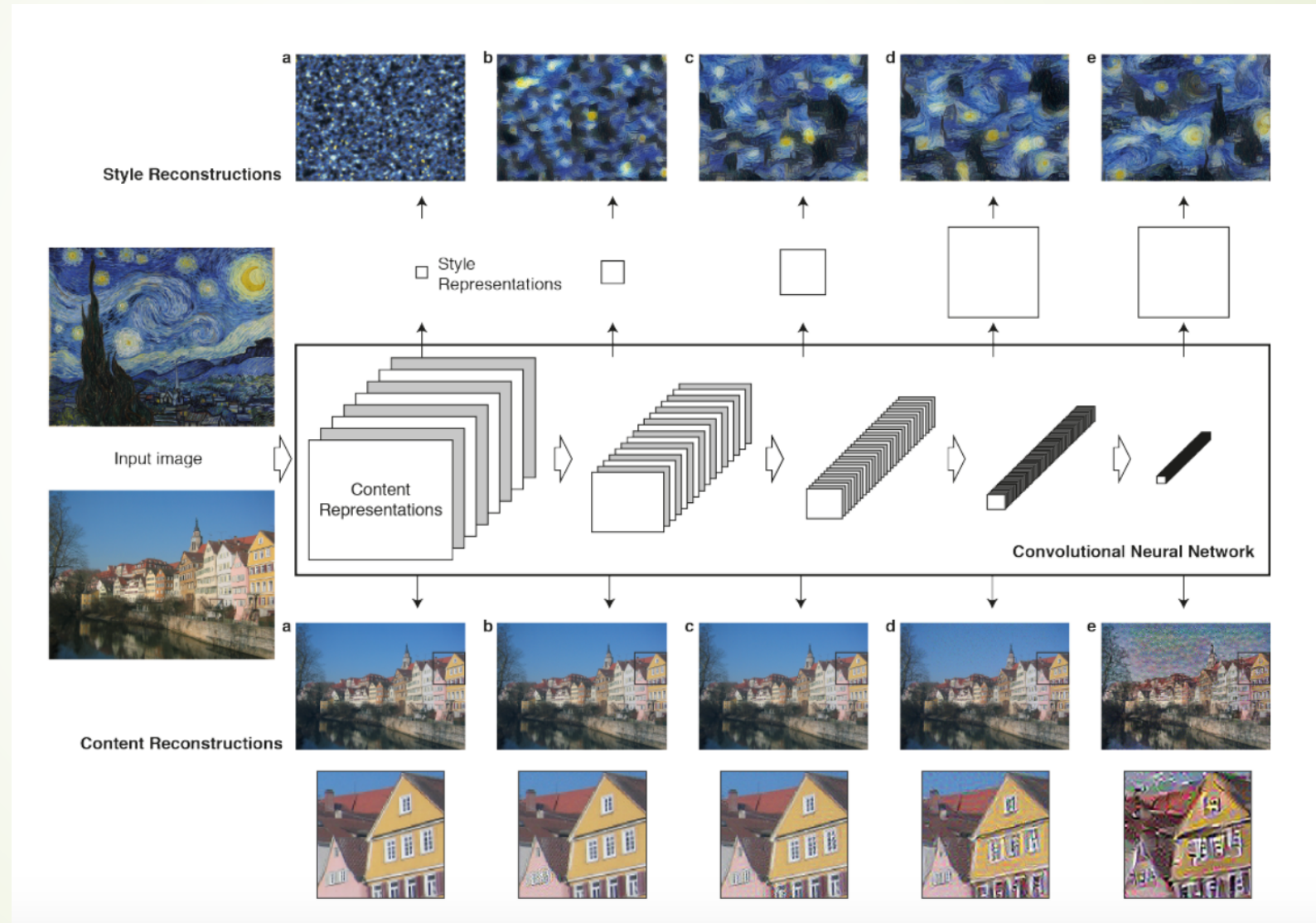




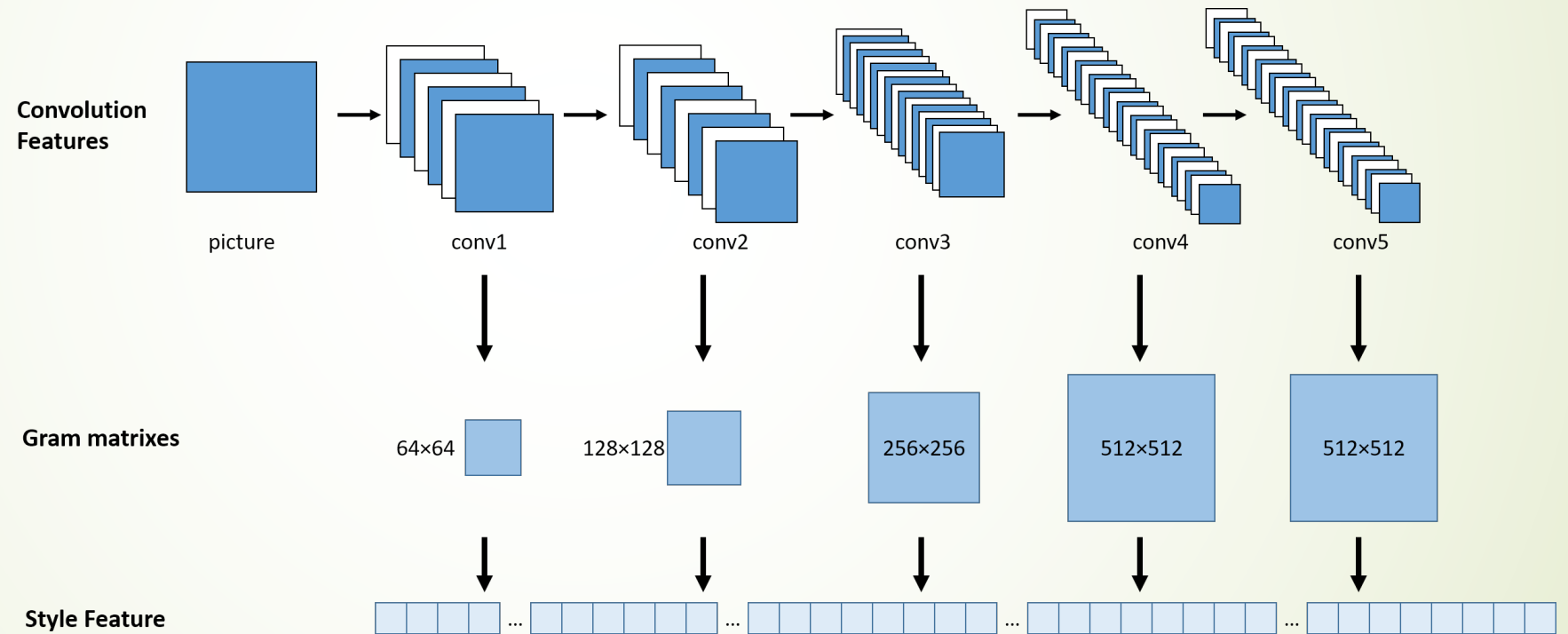
# Neural Style

- ▶ J C Johnson's Website: <https://github.com/jcjohnson/neural-style>
- ▶ A torch implementation of the paper
  - ▶ *A Neural Algorithm of Artistic Style*,
  - ▶ by Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge.
  - ▶ <http://arxiv.org/abs/1508.06576>

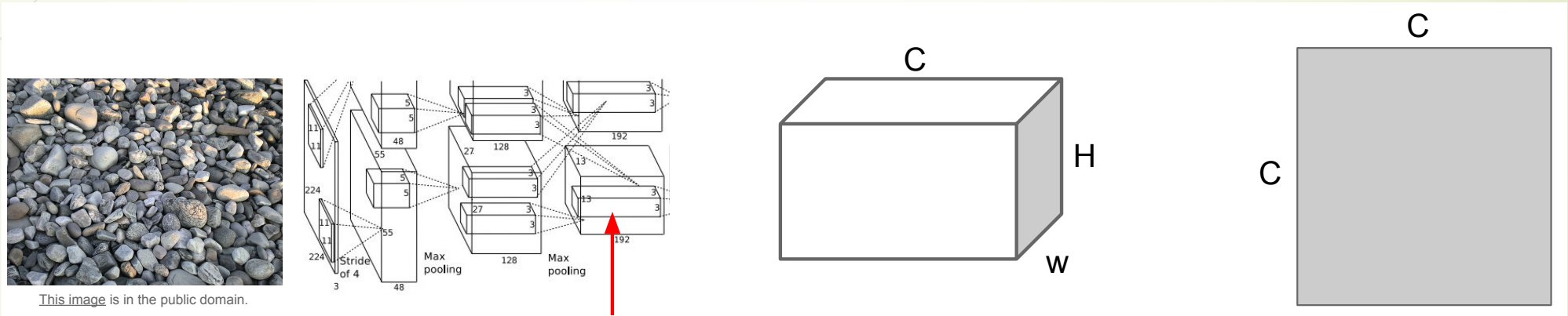
# Style-Content Feature Extraction



# Style Features as Second Order Statistics



# Neural Texture Synthesis



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 measuring co-occurrence

Average over all  $HW$  pairs of vectors, giving **Gram matrix** of shape  $C \times C$

Efficient to compute; reshape features from

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

then compute  $G = FF^T$

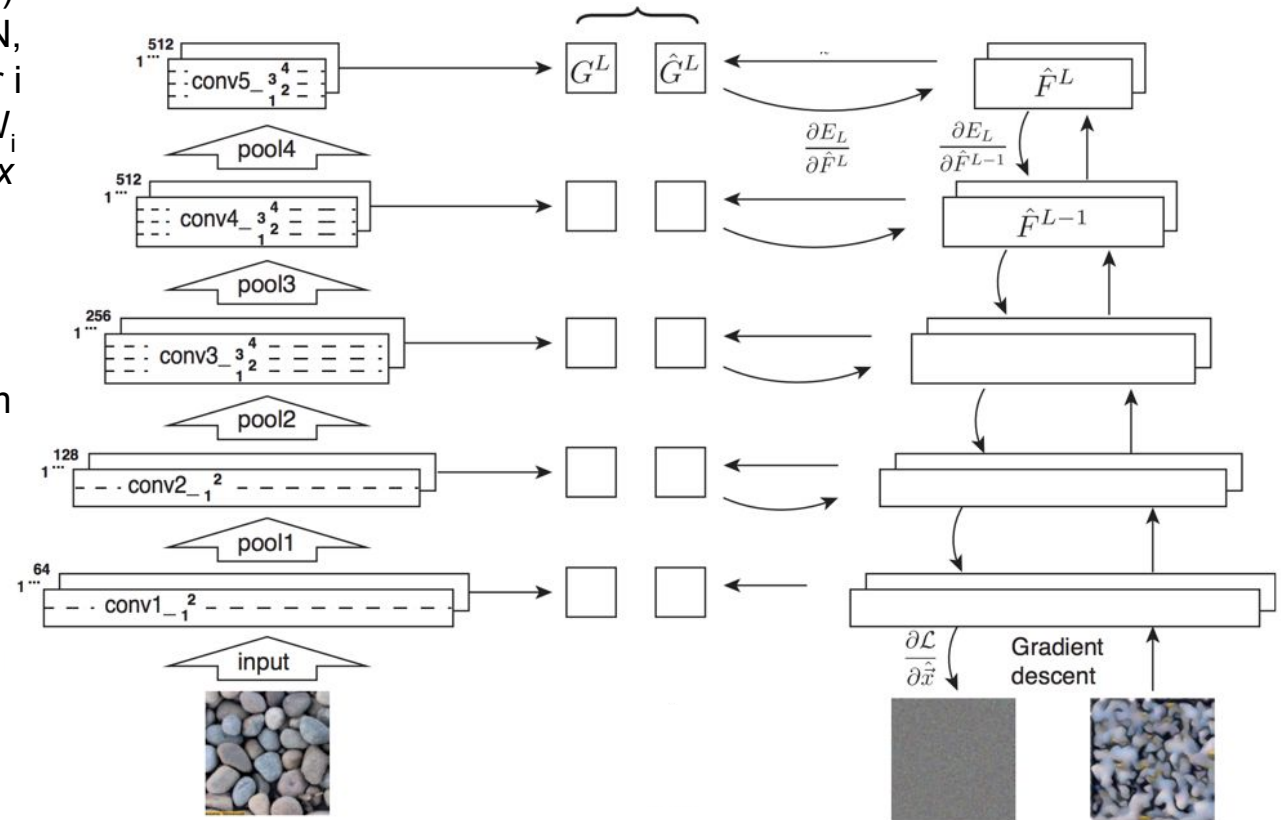
# Neural Texture Synthesis

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

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

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
8. Make gradient step on image
9. GOTO 5

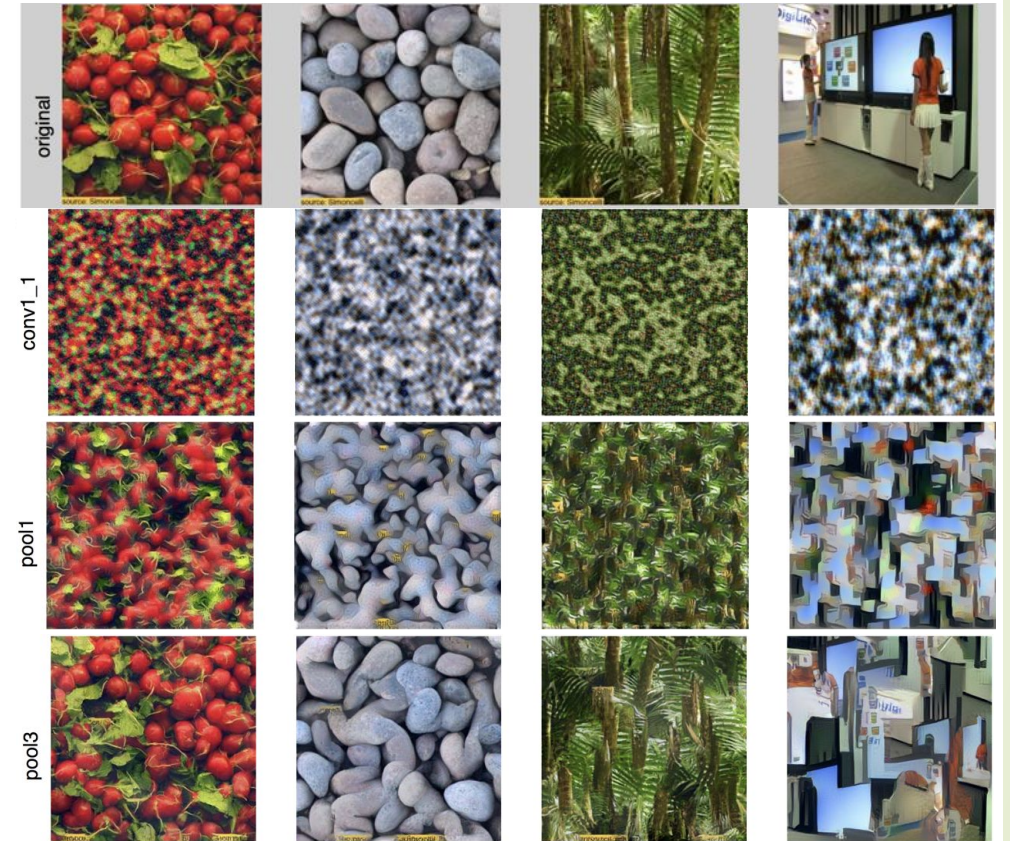
$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - \hat{G}_{ij}^l)^2 \quad \mathcal{L}(\vec{x}, \hat{x}) = \sum_{l=0}^L w_l E_l$$





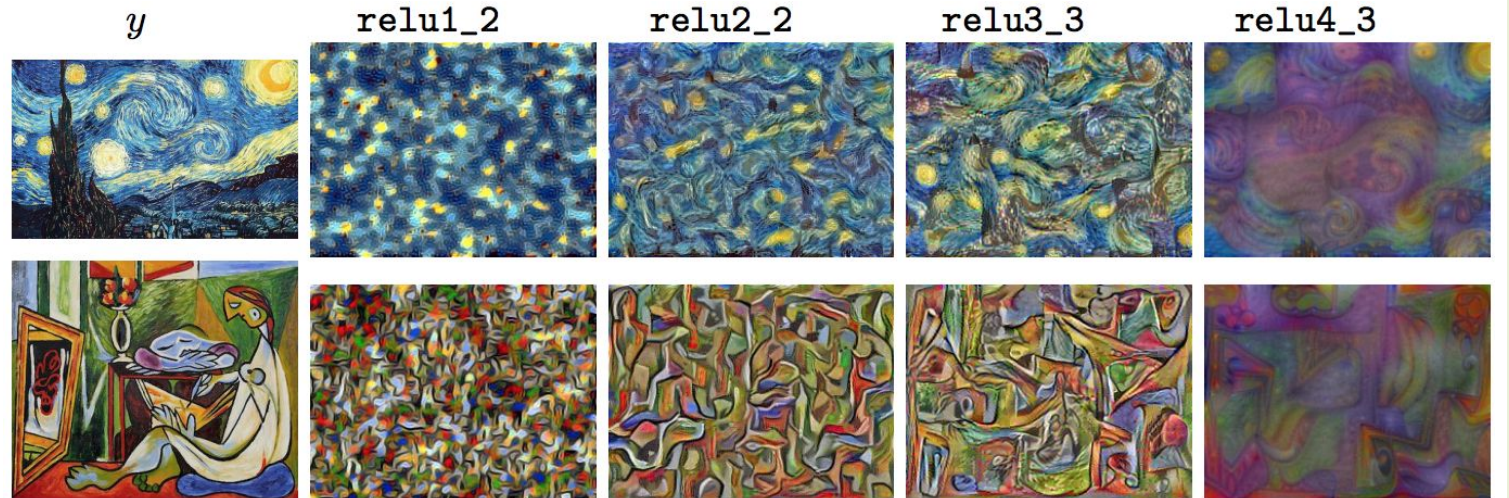
# Neural Texture Synthesis

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



# Neural Texture Synthesis: Gram Reconstruction

Texture synthesis  
(Gram  
reconstruction)



# 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)

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

Given feature vector

Features of new image

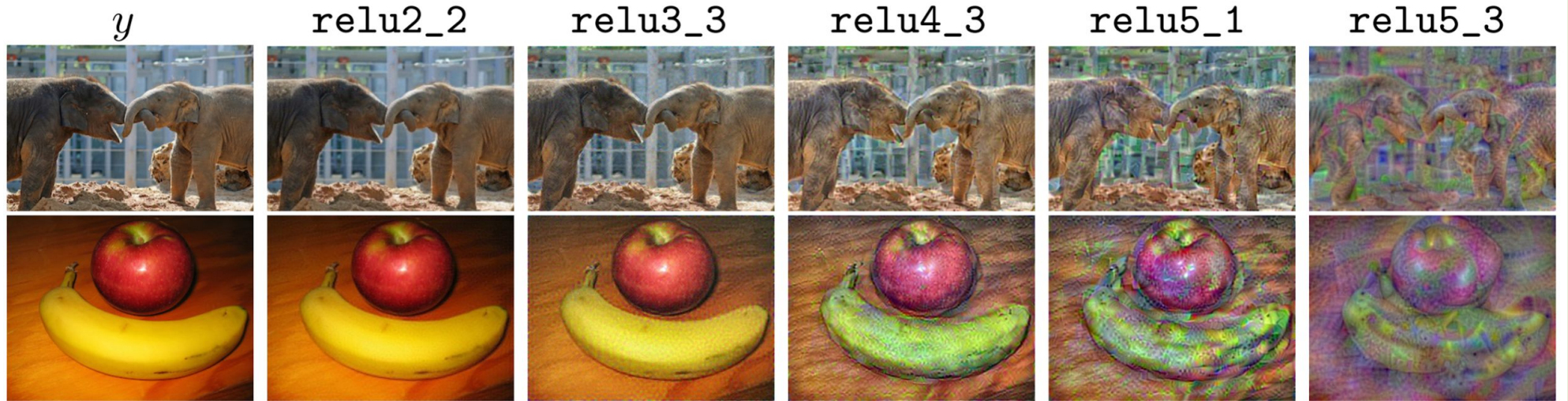
$$\ell(\Phi(\mathbf{x}), \Phi_0) = \|\Phi(\mathbf{x}) - \Phi_0\|^2$$

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

Total Variation regularizer  
(encourages spatial smoothness)

# Feature Inversion

Reconstructing from different layers of VGG-16



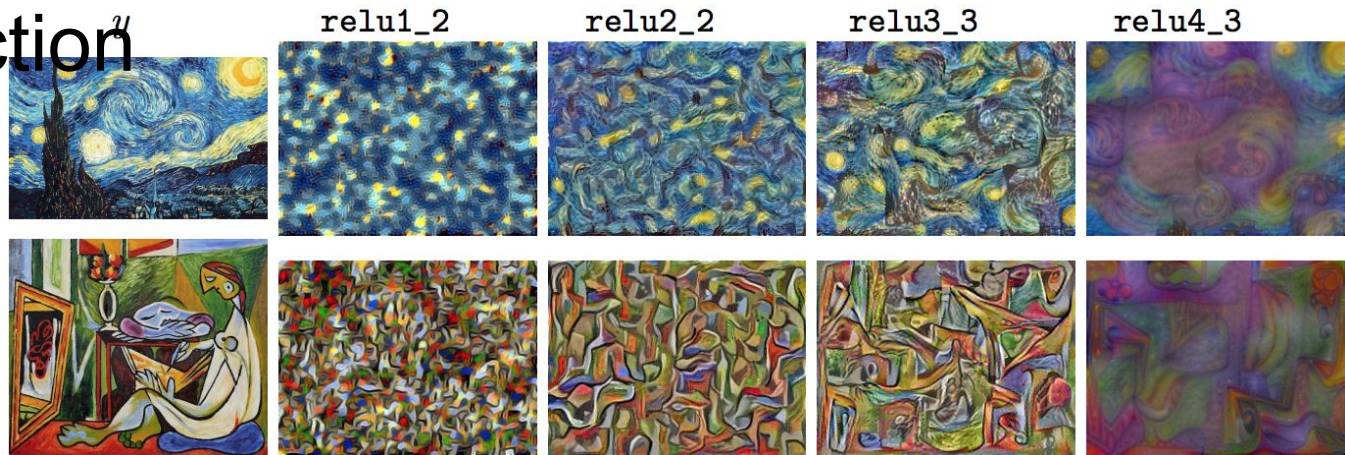
Mahendran and Vedaldi, "Understanding Deep Image Representations by Inverting Them", CVPR 2015

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

Reproduced for educational purposes.

# Neural Style Transfer: Feature + Gram Reconstruction

Texture synthesis  
(Gram reconstruction)



Feature reconstruction

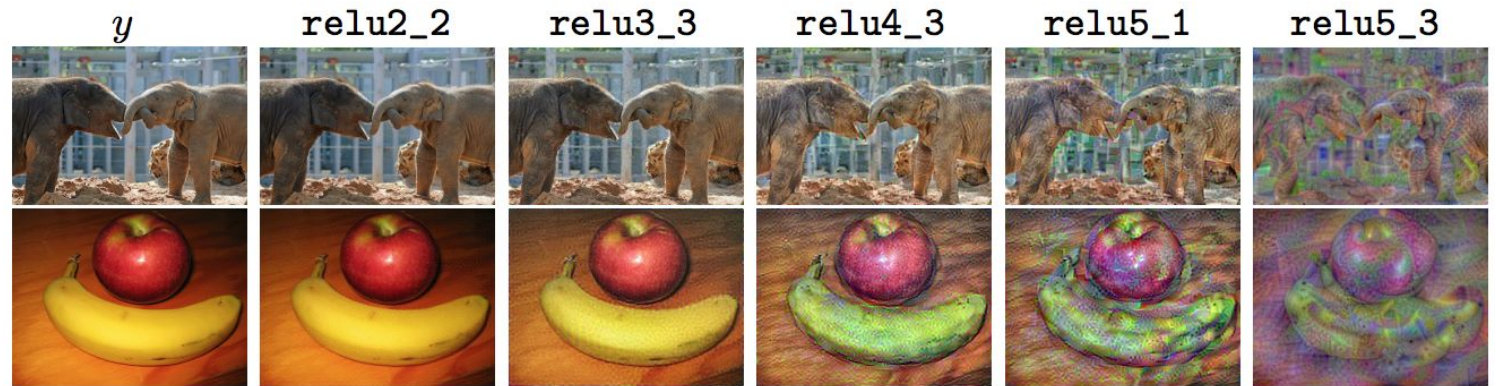


Figure from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016. Copyright Springer, 2016. Reproduced for educational purposes.

# Combined Loss for both Content (1<sup>st</sup> order statistics) and Style (2<sup>nd</sup> order statistics: Gram)

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2 .$$

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

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$$

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l .$$

# Neural Style Transfer

Content Image



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

+

Style Image



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

=

Style Transfer!



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

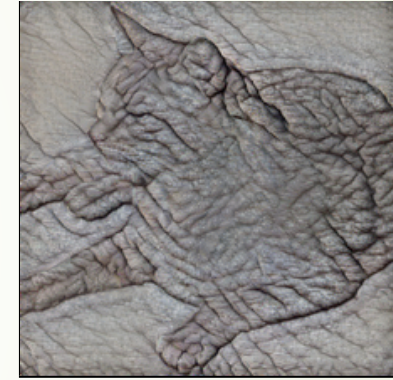
# CNN learns **texture** features, not **shapes**!



(a) Texture image  
81.4% **Indian elephant**  
10.3% indri  
8.2% black swan



(b) Content image  
71.1% **tabby cat**  
17.3% grey fox  
3.3% Siamese cat



(c) Texture-shape cue conflict  
63.9% **Indian elephant**  
26.4% indri  
9.6% black swan

Geirhos et al. ICLR 2019

<https://videoken.com/embed/W2HvLBMhCJQ?tocitem=46>





# Examples

- ▶ Jupyter Notebook Demo
- 



# Adversarial Examples and Robustness

# Deep Learning may be fragile: adversarial examples

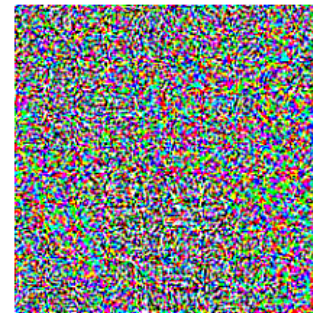


$x$

“panda”

57.7% confidence

+ .007 ×



$\text{sign}(\nabla_x J(\theta, x, y))$

“nematode”

8.2% confidence

=



$x +$

$\epsilon \text{sign}(\nabla_x J(\theta, x, y))$

“gibbon”


99.3 % confidence

[Goodfellow et al., 2014]

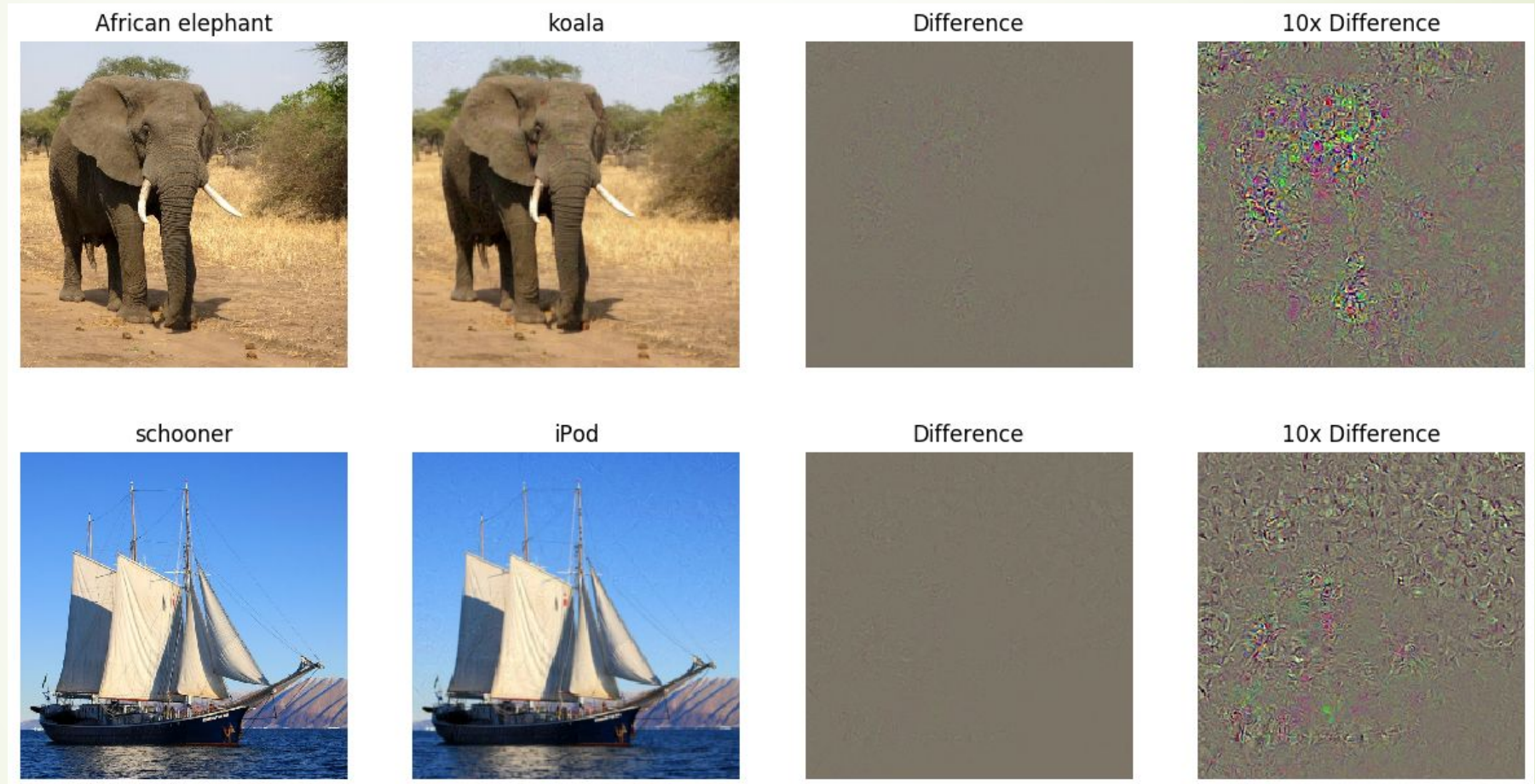
- Small but malicious perturbations can result in severe misclassification
- Malicious examples generalize across different architectures
- What is source of instability?
- Can we robustify network?



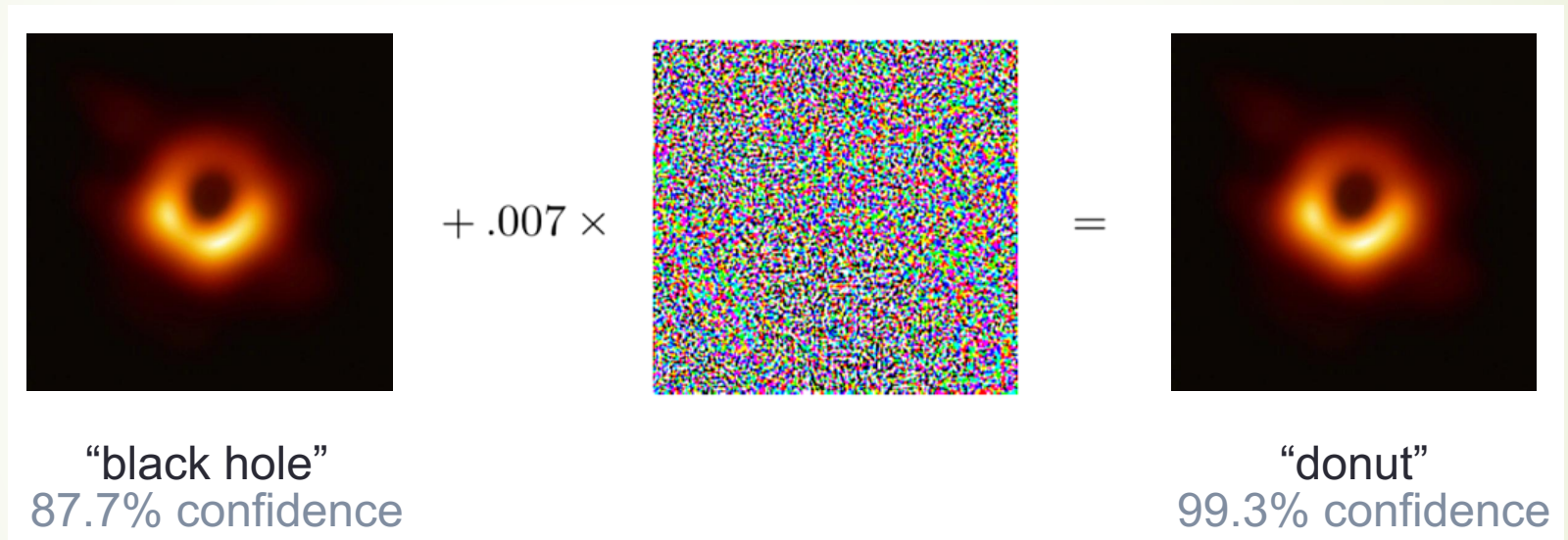
# Adversarial Examples: Fooling Images

- Start from an arbitrary image
  - Pick an arbitrary class
  - Modify the image to maximize the class
  - Repeat until network is fooled
- 

# Fooling Images/Adversarial Examples



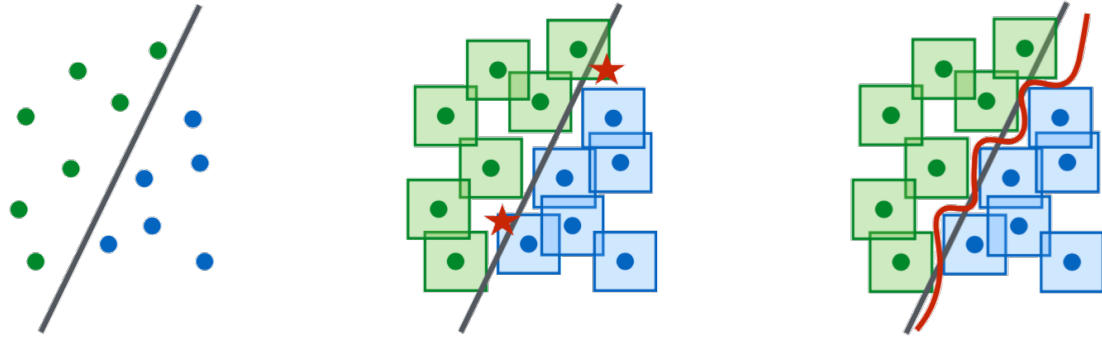
# Convolutional Networks lack Robustness



Courtesy of Dr. Hongyang ZHANG.



# Adversarial Robust Training



- Traditional training:

$$\min_{\theta} J_n(\theta, \mathbf{z} = (x_i, y_i)_{i=1}^n)$$

- e.g. square or cross-entropy loss as negative log-likelihood of logit models

- Robust optimization (Madry et al. ICLR'2018):

$$\min_{\theta} \max_{\|\epsilon_i\| \leq \delta} J_n(\theta, \mathbf{z} = (x_i + \epsilon_i, y_i)_{i=1}^n)$$

- robust to any distributions, yet computationally hard

Extended by **Hongyang ZHANG** et al. by TRADES, 2019.

Thank you!

