



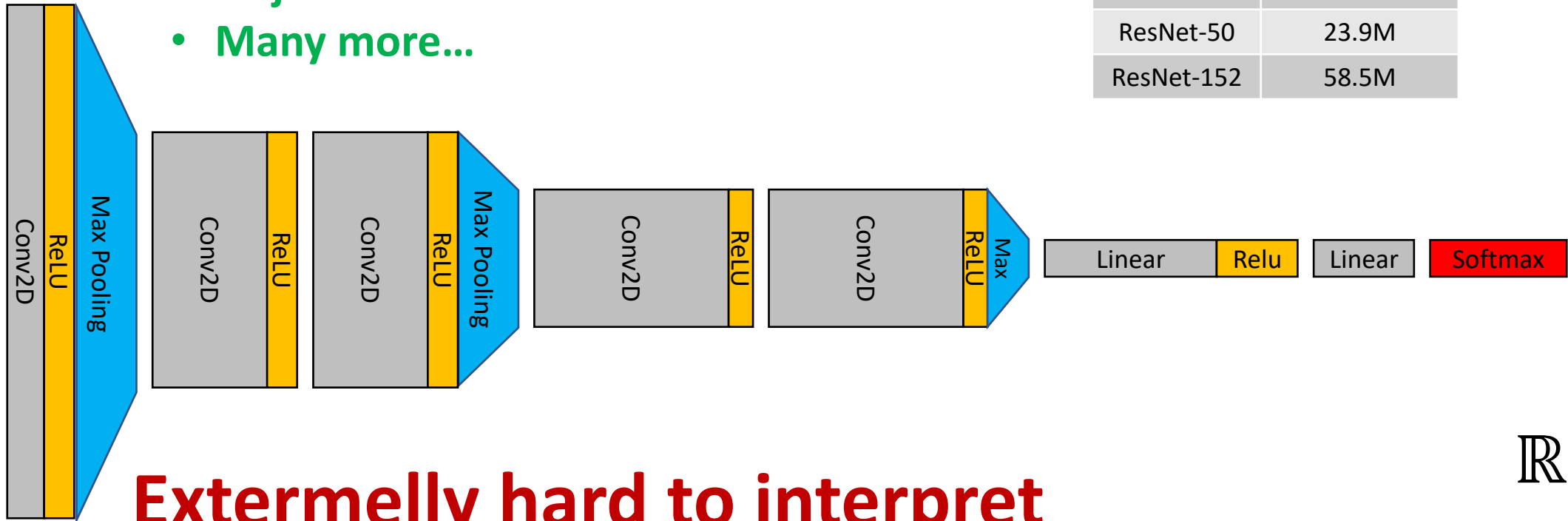
Lecture 6: Visualizing and Understanding Neural Networks

Nov 22nd, 2021

Tali Dekel

State-of-the-art performance:

- Image classification
- Segmentation
- Object detection
- Many more...

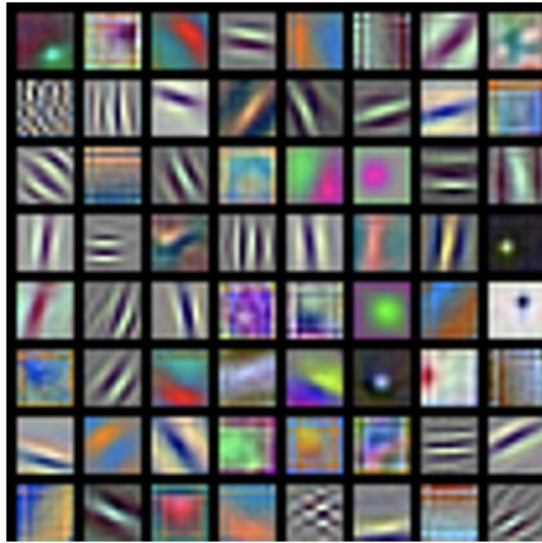


Extermelly hard to interpret

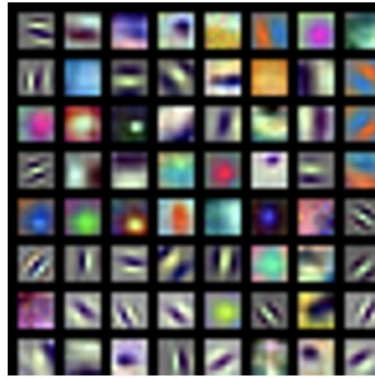
Model	# parameters
VGG-16	134.7M
ResNet-18	11.4M
ResNet-34	21.5M
ResNet-50	23.9M
ResNet-152	58.5M

Visualizing the First Layer's Filters

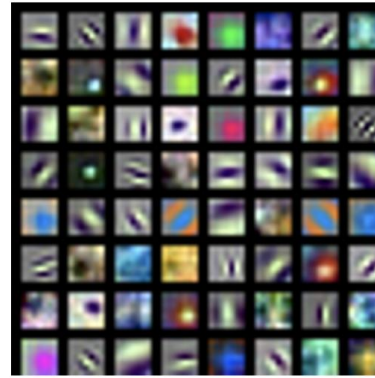
Visualize the linear weights in the **input-to-first layer**



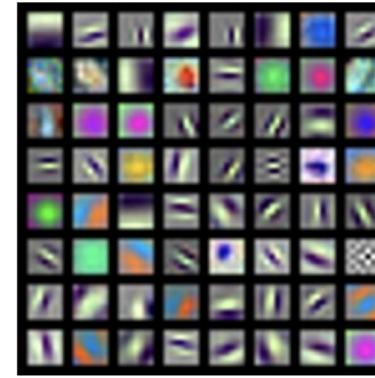
AlexNet:
 $64 \times 3 \times 11 \times 11$



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

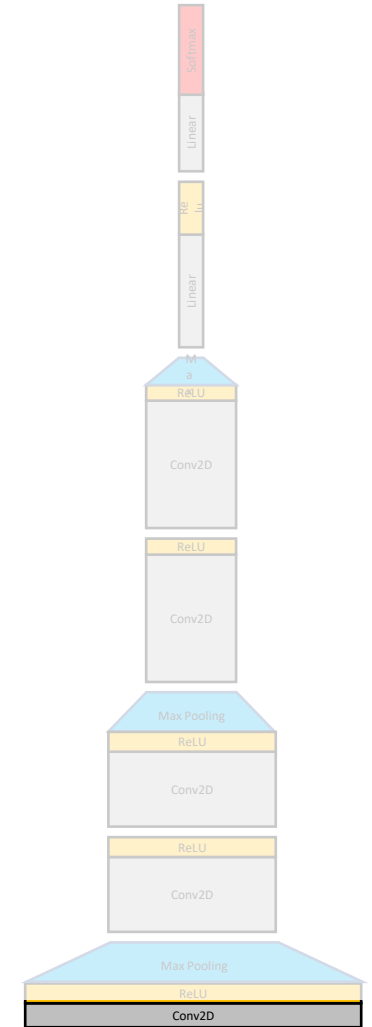


ResNet-101:
 $64 \times 3 \times 7 \times 7$



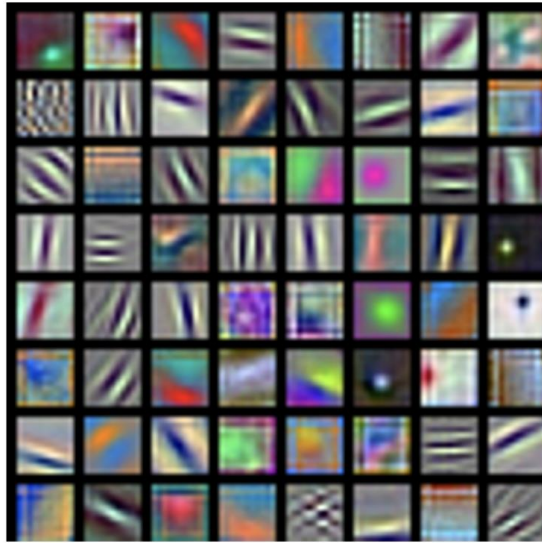
DenseNet-121:
 $64 \times 3 \times 7 \times 7$

<https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>



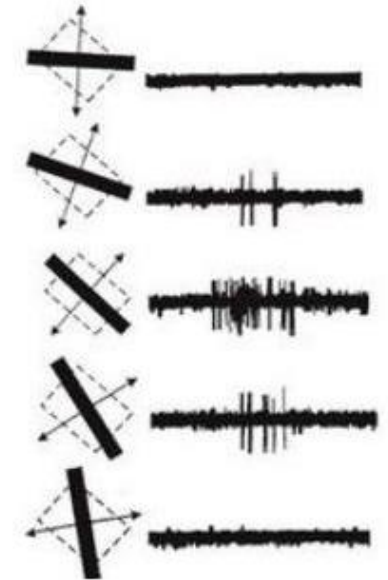
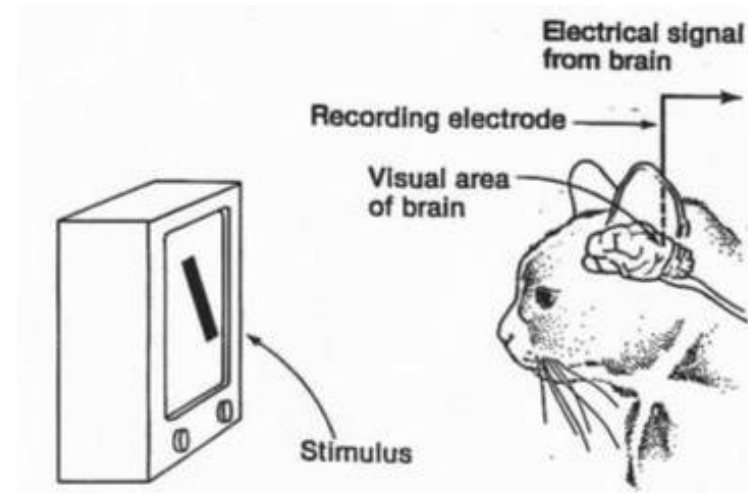
Visualizing the First Layer's Filters

Visualize the linear weights in the **input-to-first layer**



AlexNet:
64 x 3 x 11 x 11

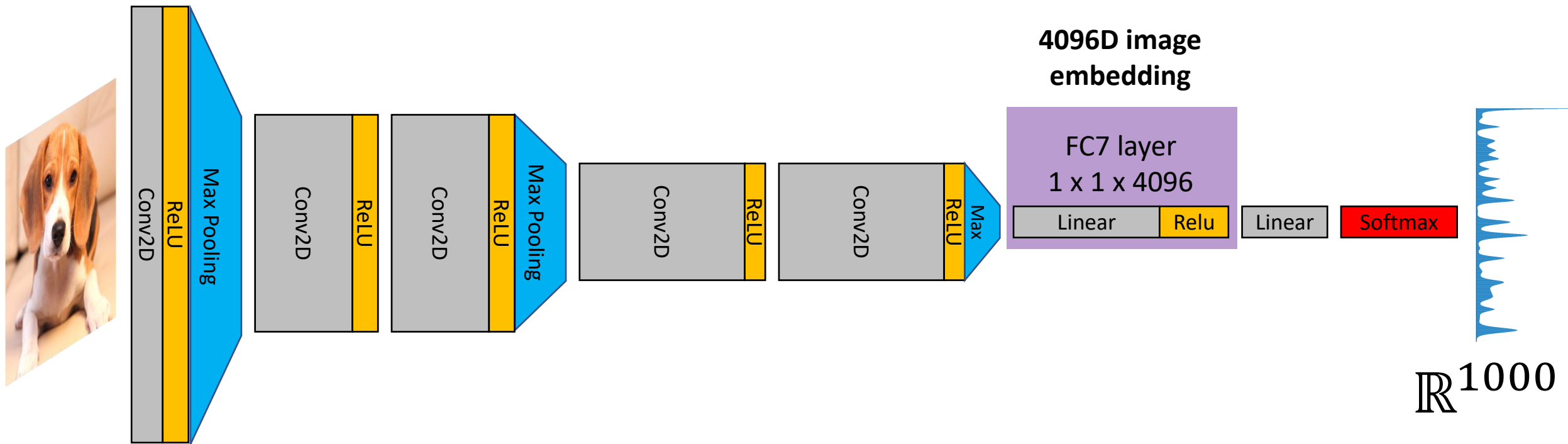
Hubel & Wiesel, 1959



<https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>

Last (fully connected) Layer Feature

- Use the feature activation of the last layer as an image embedding (feature)



Last (fully connected) Layer Feature: Nearest Neighbors (NNs)

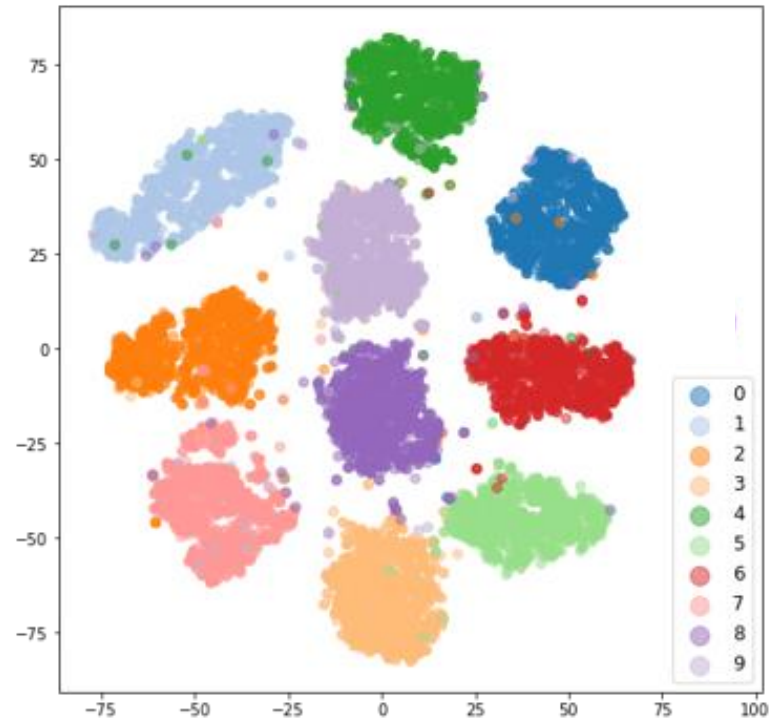
- (1) Run many images in the network and record their features
- (2) Compute similarity of a query image to the rest; retrieve nearest neighbours

Query



Deep Features as Image Embedding

- (1) Feed many images and record their features
- (2) Reduce feature dimensionality to 2D (or 3D):
 - Linear: Principle Component Anyalsis (PCA)
 - Non-linear: T-distributed stochastic neighbor (t-SNE)



MNIST t-SNE embedding

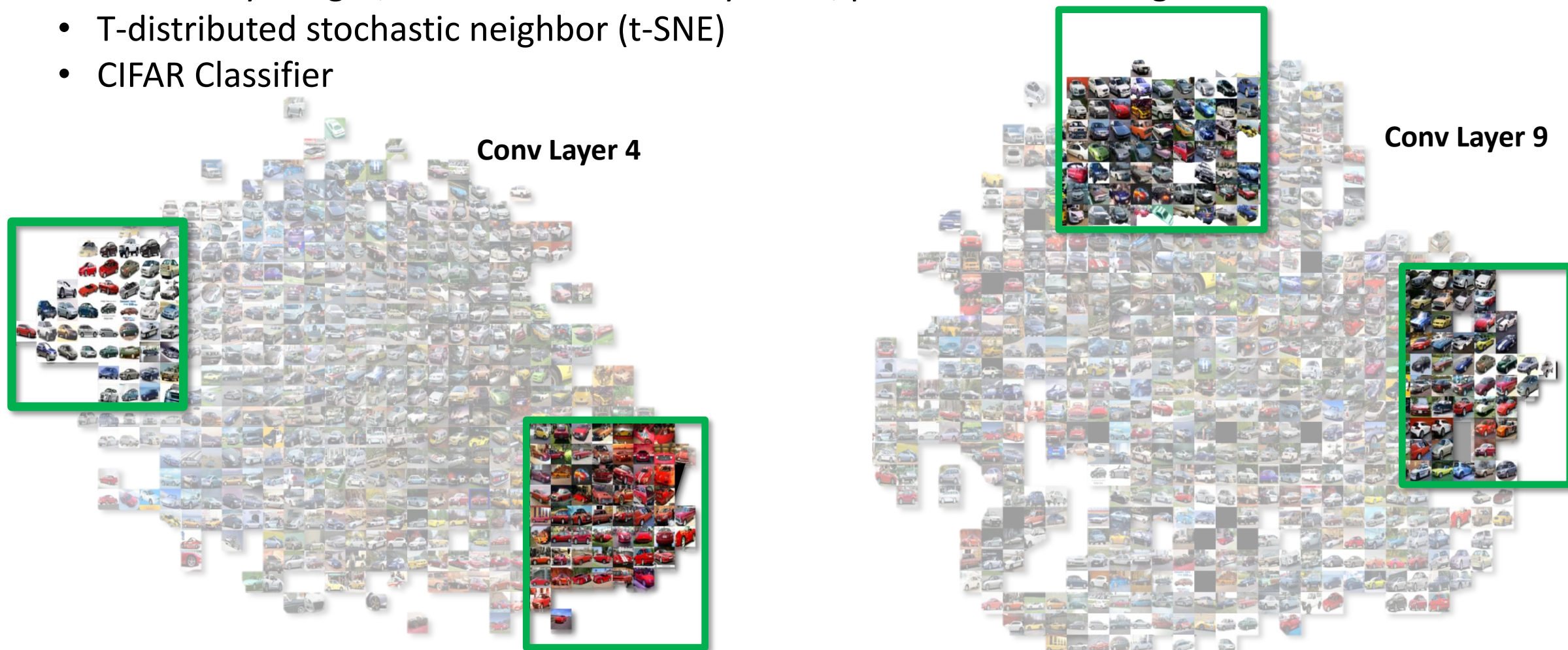
Plot from:

<https://towardsdatascience.com/visualizing-feature-vectors-embeddings-using-pca-and-t-sne-ef157cea3a42>

Deep Features as Image Embedding

Embed many images, reduce dimensionality to 2D, plot the source images:

- T-distributed stochastic neighbor (t-SNE)
- CIFAR Classifier



See more in: http://people.csail.mit.edu/talidekel/private/2D_feature_embedding_cifar_html/index.html

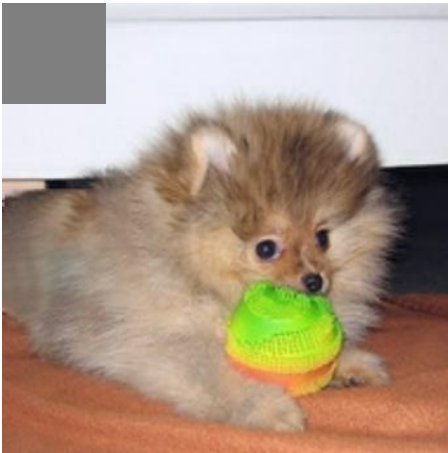
<https://cs.stanford.edu/people/karpathy/cnnembed/>

Saliency via occlusion

Which pixels matter?

- Mask an image region
- Feed masked image to the classifier
- Record the prediction of the original class

True Label: Pomeranian



Probability of Pomeranian

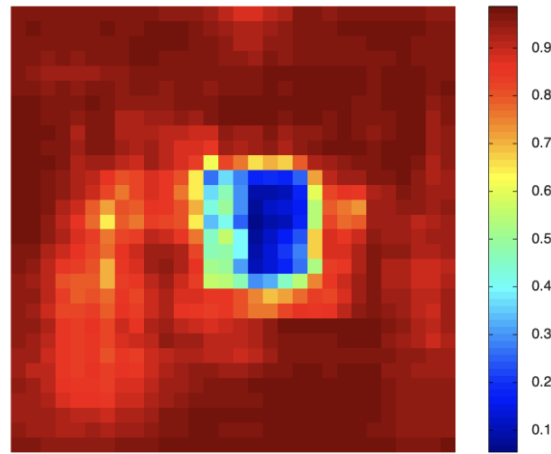
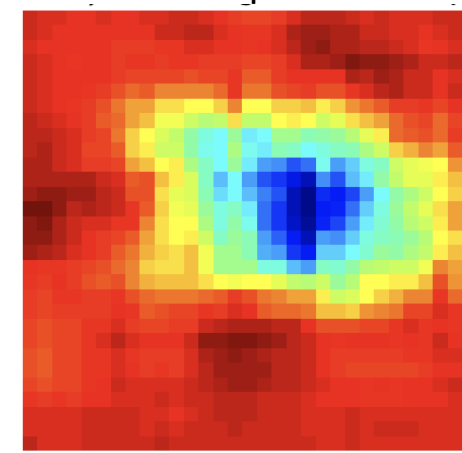


Image overlay

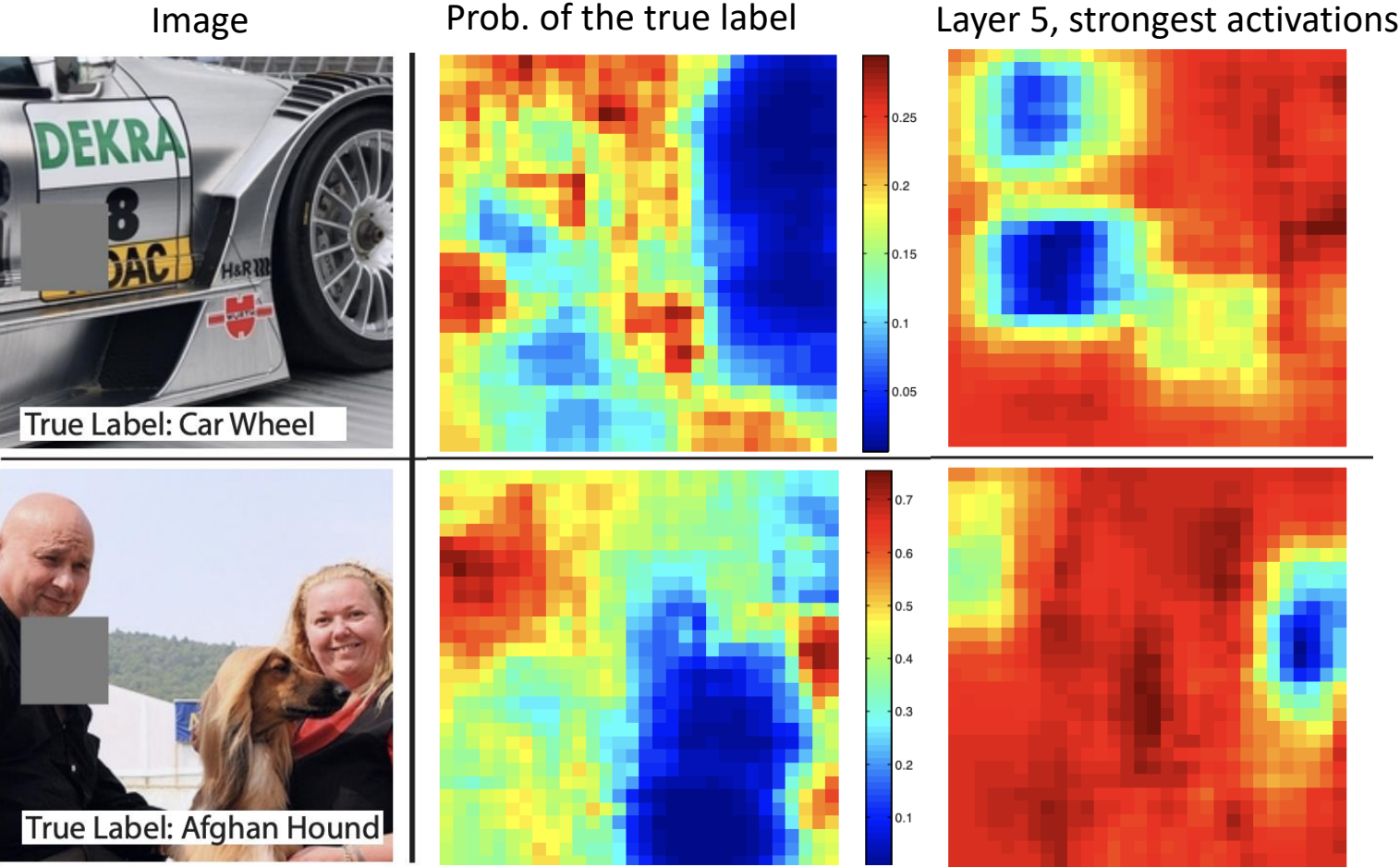


Layer 5, strongest activations
(summed over spatial locations)



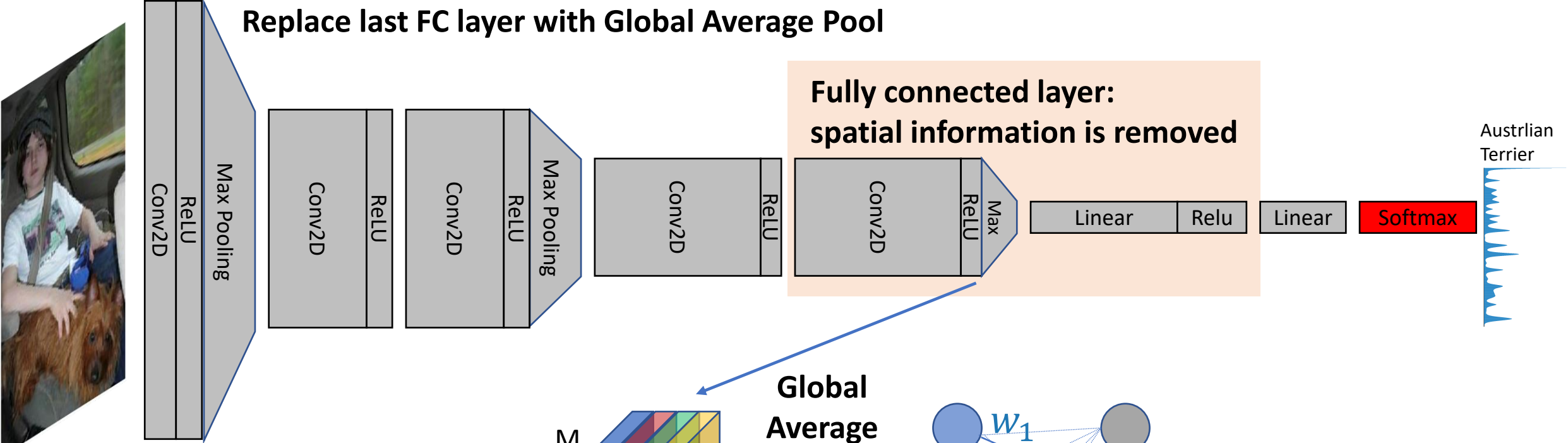
Saliency via occlusion

Which parts of the image are important for classification?



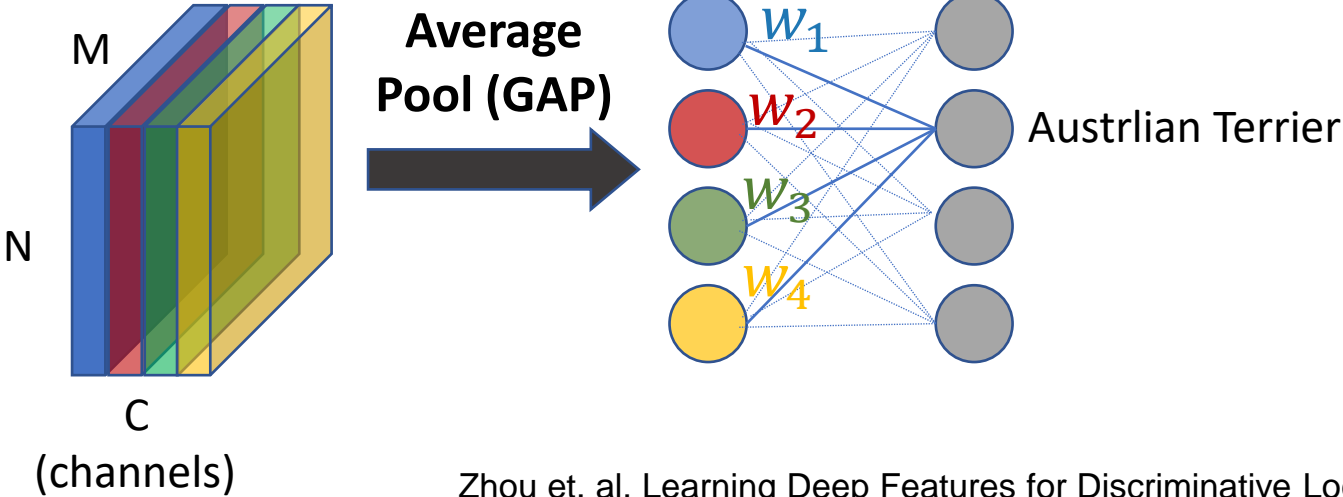
Class Activation Maps (CAM)

Replace last FC layer with Global Average Pool

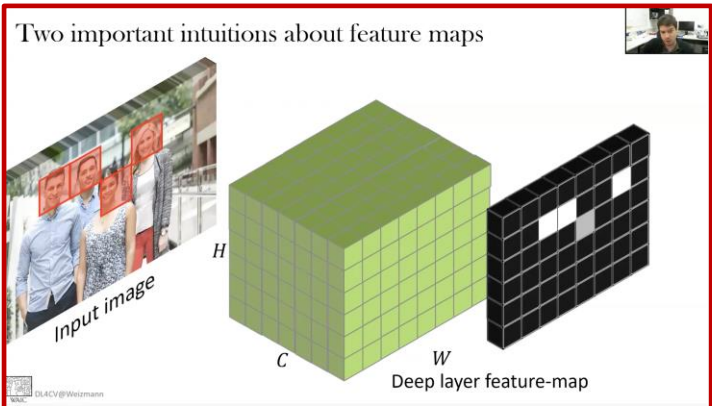


Global

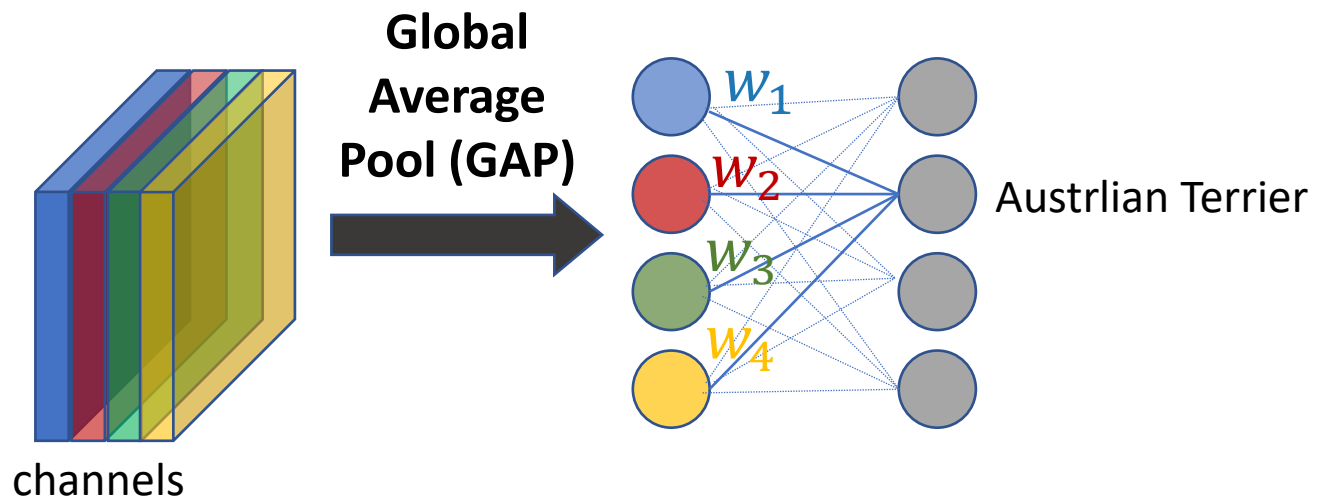
Average
Pool (GAP)



Two important intuitions about feature maps



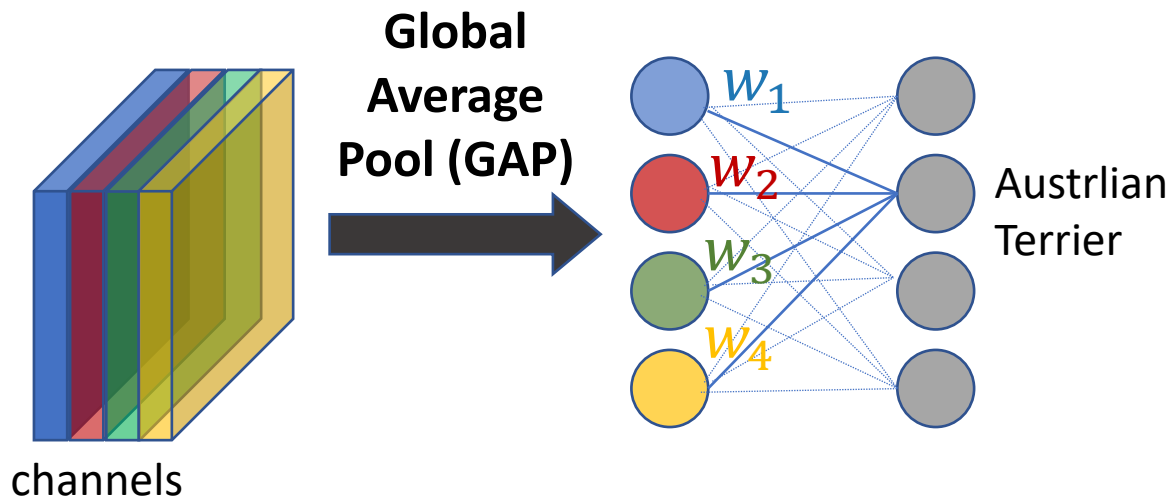
Class Activation Maps (CAM)



$$W_1 * \text{Feature Map}_1 + W_2 * \text{Feature Map}_2 + \dots + W_n * \text{Feature Map}_n = \text{Class Activation Map (Australiian Terrier)}$$

Class Activation Map (Australiian Terrier)

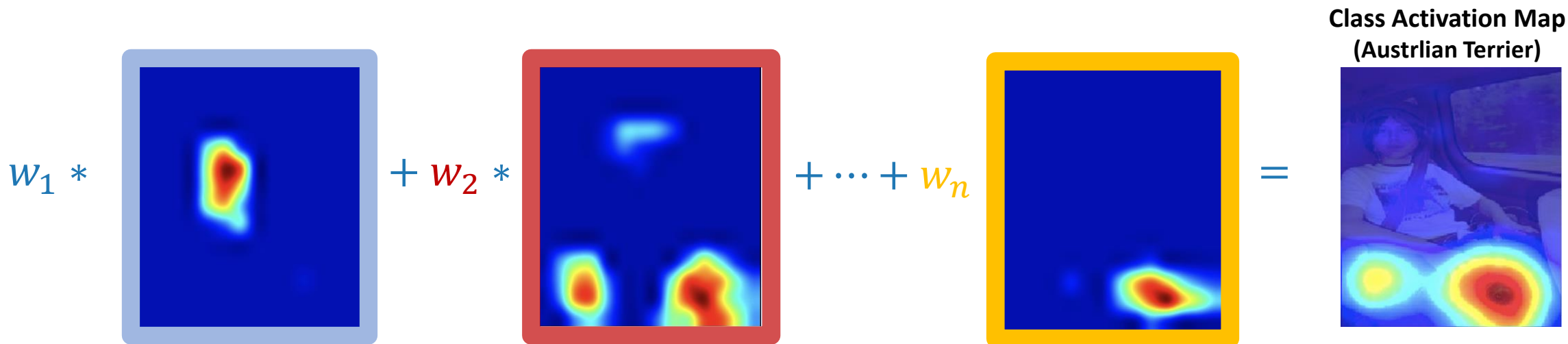
Class Activation Maps (CAM)



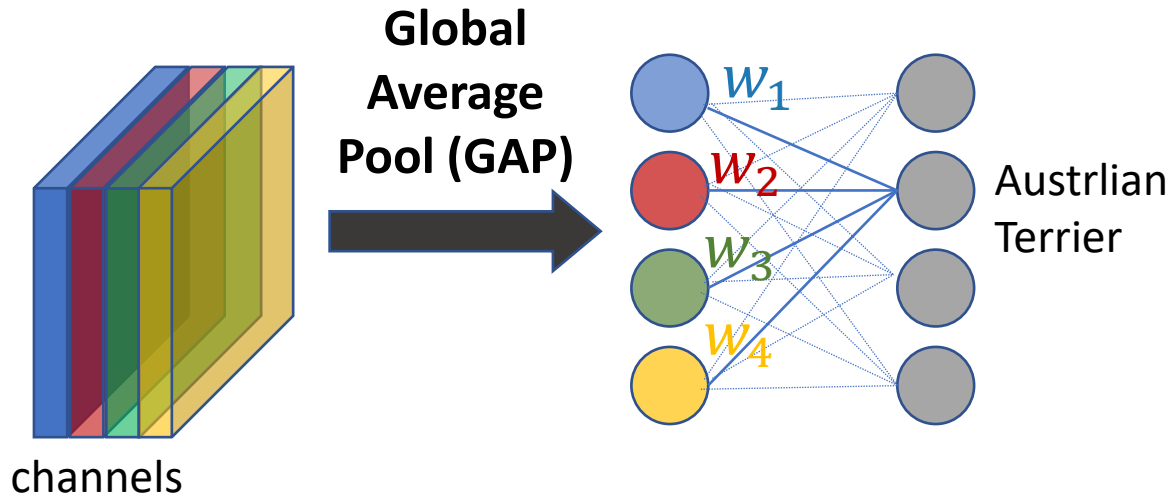
$f_k(x, y)$ - the k th feature map

$F_k = \sum_{x,y} f_k(x, y)$ - the k th averaged pooled value

Score for class "c"

$$S_c = \sum_k w_k F_k$$


Class Activation Maps (CAM)



$f_k(x, y)$ - the k th feature map

$F_k = \sum_{x,y} f_k(x, y)$ - the k th averaged pooled value

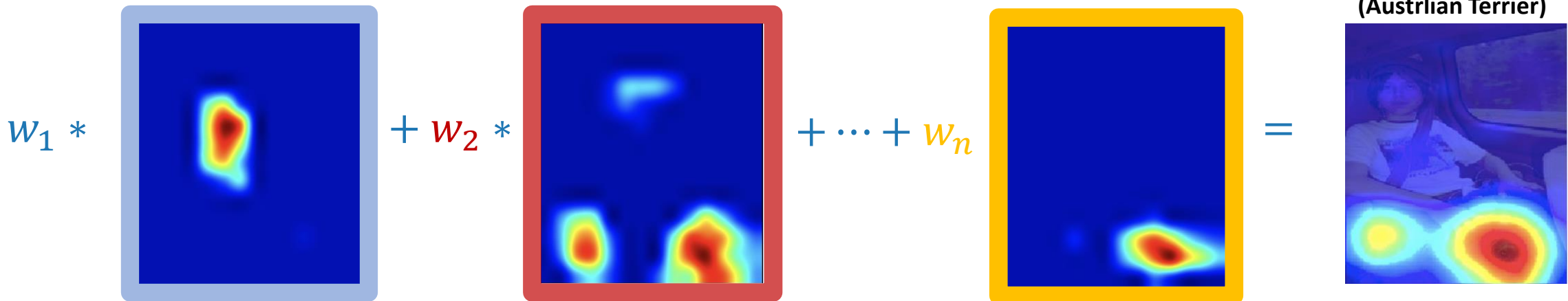
Score for class "c"

$$S_c = \sum_k w_k F_k = \sum_k w_k \sum_{x,y} f_k(x, y)$$

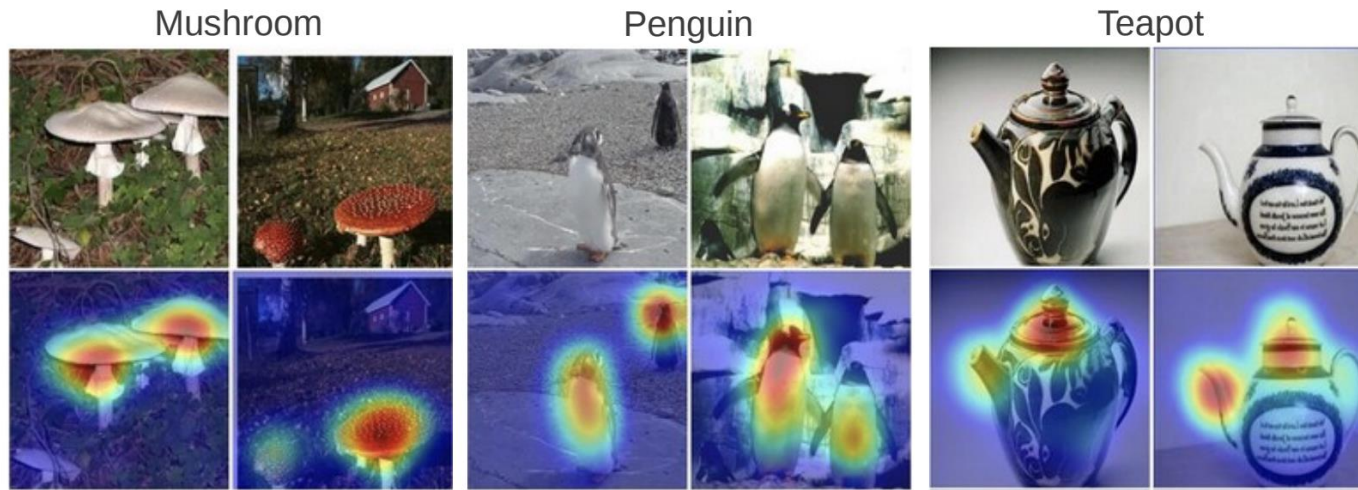
$$= \sum_{x,y} \sum_k w_k f_k(x, y)$$

M_{CAM}^c

Class Activation Map (Austrian Terrier)



Class Activation Maps (CAM)



Caltech256



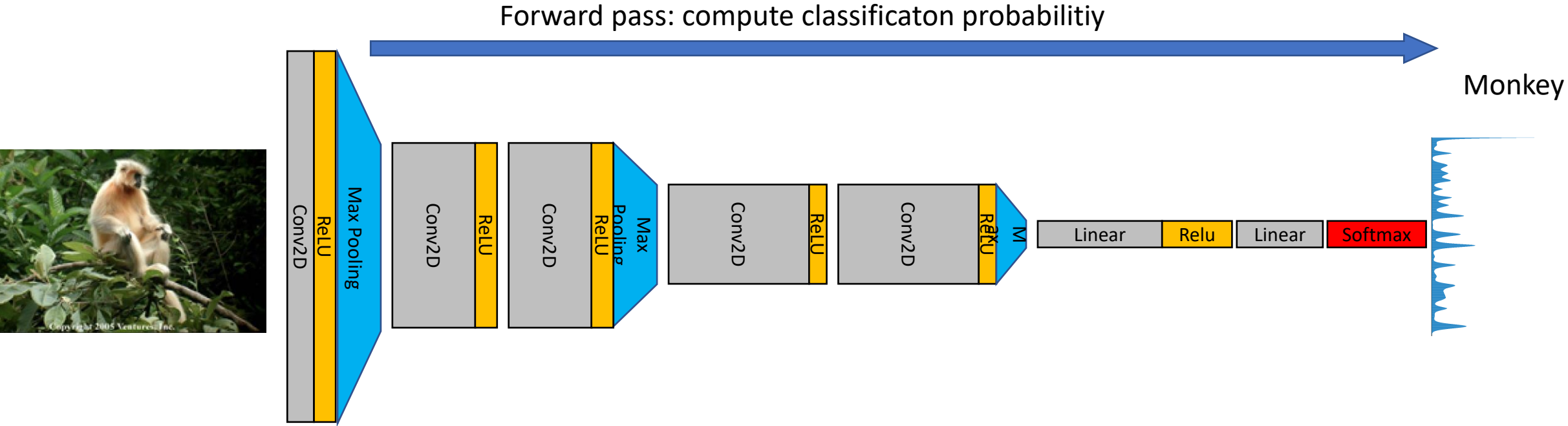
UIUC Event8

But...

- CAM can be applied only on the last layer
- CAM requires GAP and cannot be applied to arbitrary models

Saliency via Backpropogation

Which parts of the image are important for classification?

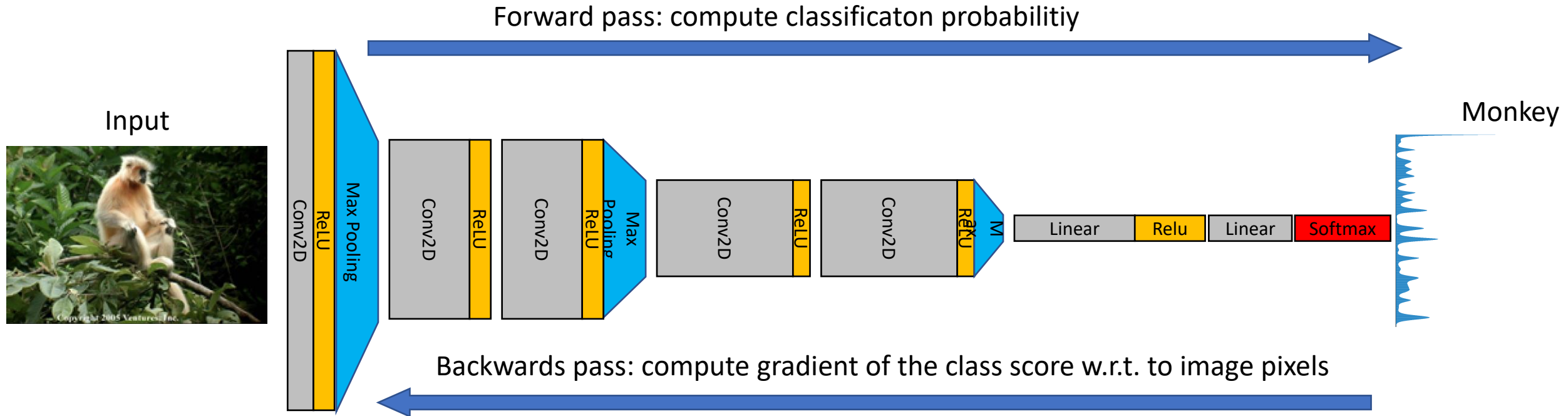


Simonyan, Vedaldi, and Zisserman, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”, ICLR Workshop 2014.

Saliency via Backpropogation

Which parts of the image are important for classification?

Compute the gradients of the target class w.r.t. the input image



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

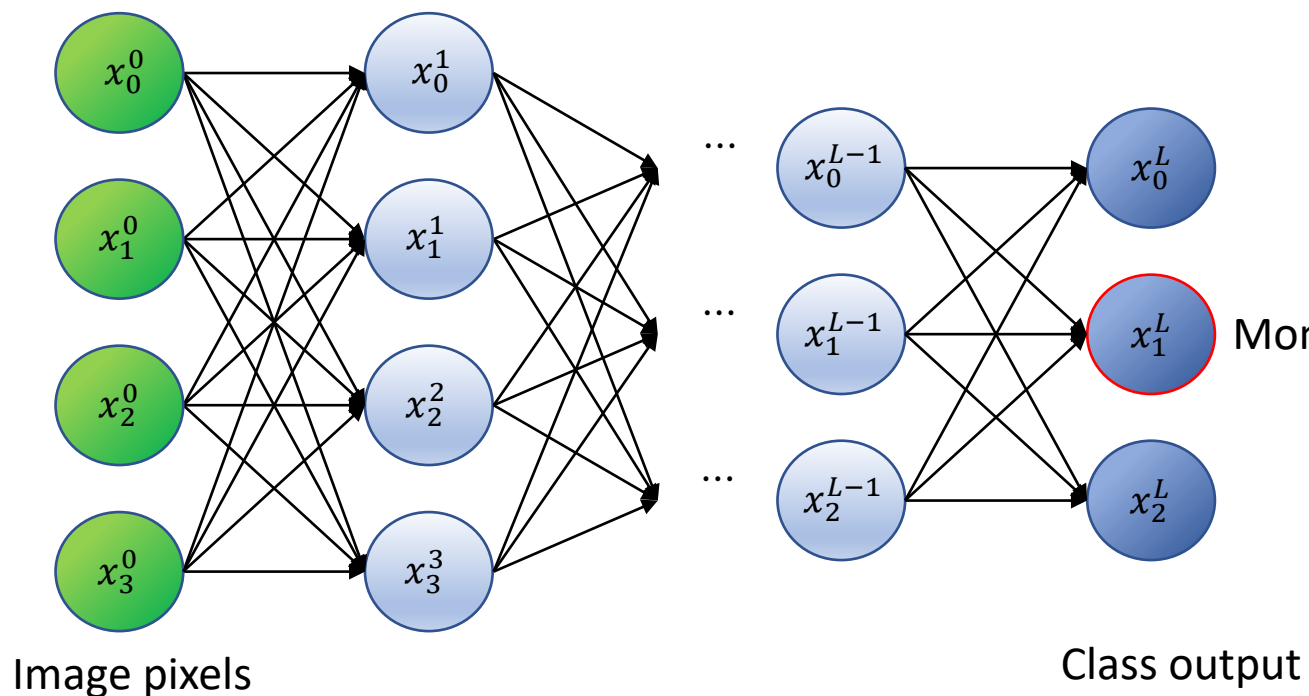
Saliency via Backpropogation

Which parts of the image are important for classification?

Compute the gradients of the target class w.r.t. the input image

$$\frac{\partial x_1^L}{\partial x_i^0}$$

Probability of the true class
Image pixel

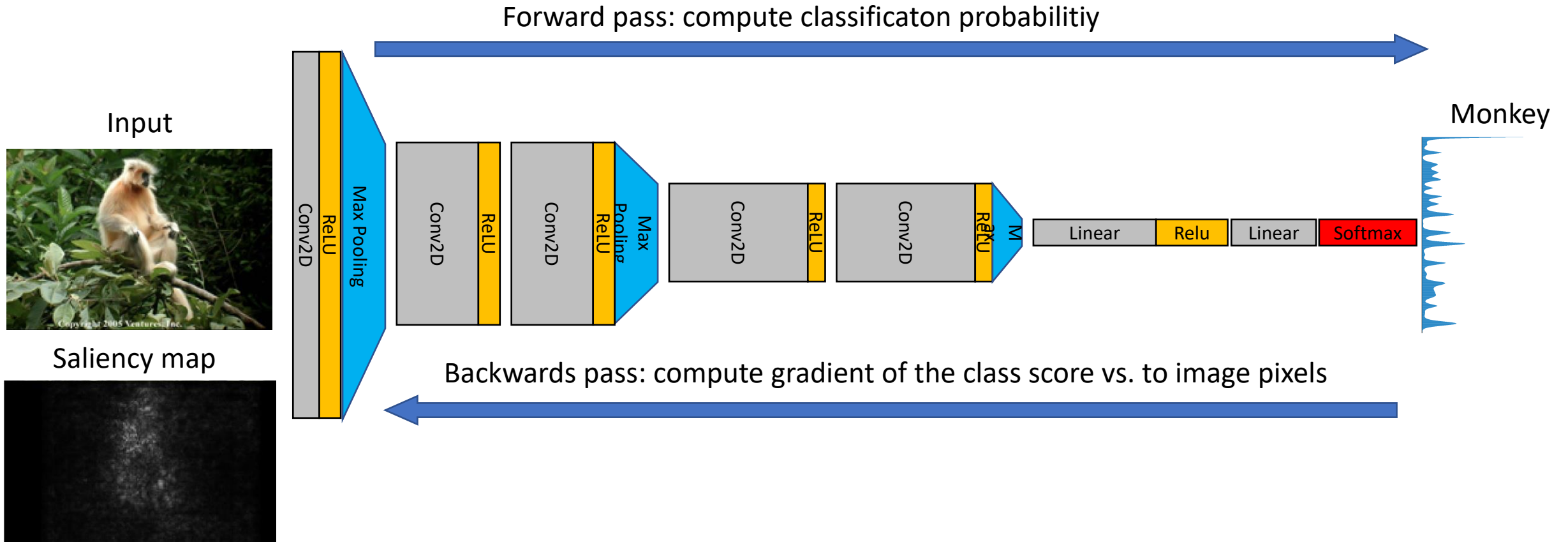


*Reminder! Class #2:
Computed the gradient of the
loss w.r.t. network weights*

Saliency via Backpropogation

Which parts of the image are important for classification?
Compute the gradients of the target class w.r.t. the input image

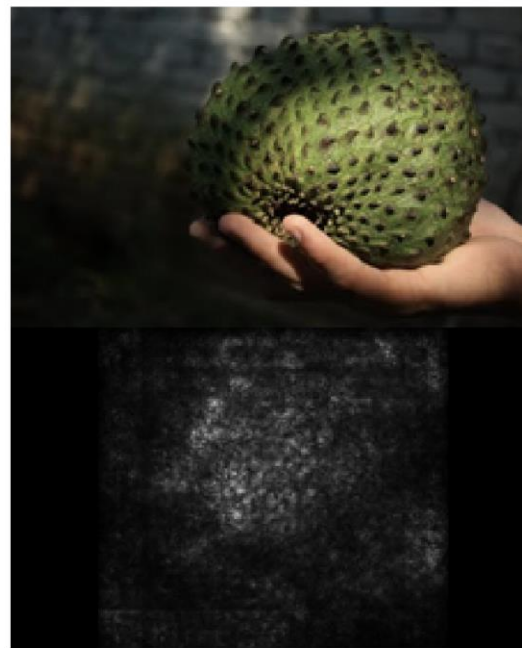
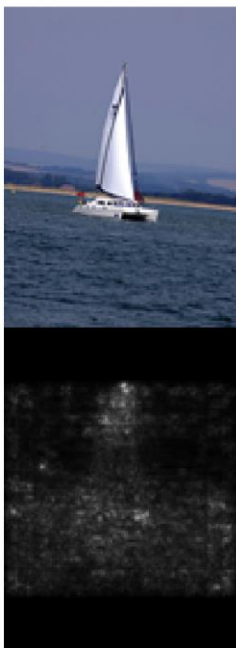
$$\frac{\partial x_1^L}{\partial x_i^0}$$



Saliency via Backpropagation

Compute the gradient of the true class w.r.t. input image

$$\frac{\partial x_1^L}{\partial x_i^0}$$



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

Saliency via Backpropogation



- Can be applied to existing models
- Highlight fine-grained details
- **Not discriminative**

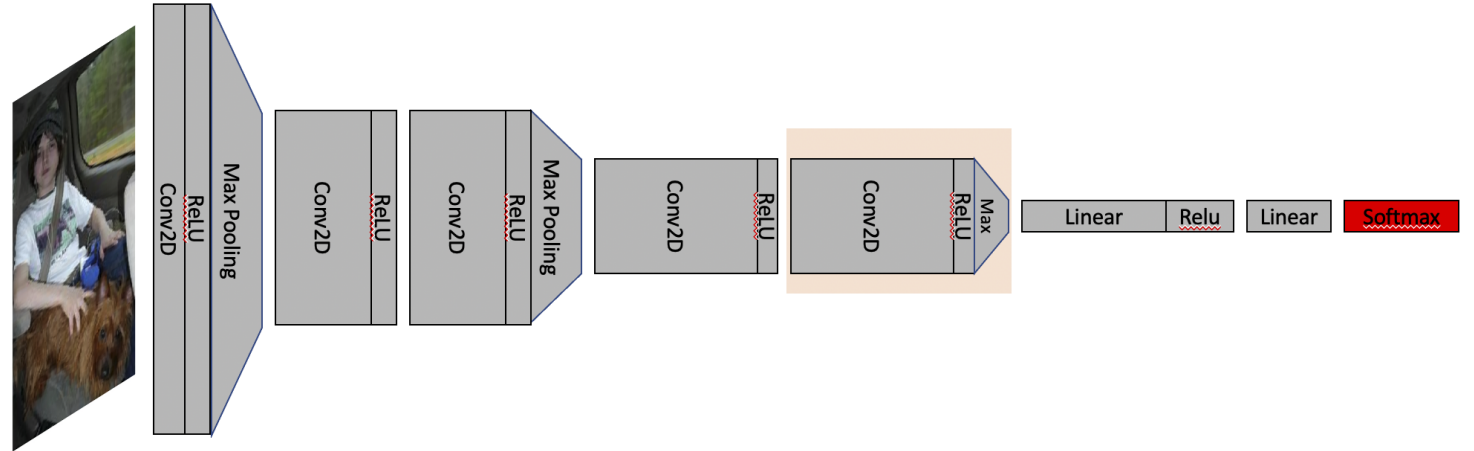
Class Activation Maps



- **Discriminative**
- **Can be applied only on the last layer**
- **Cannot be applied to arbitrary models**

Gradient-Weighted Class Activation Mapping (Grad-CAM)

Pixel-space gradient + CAM

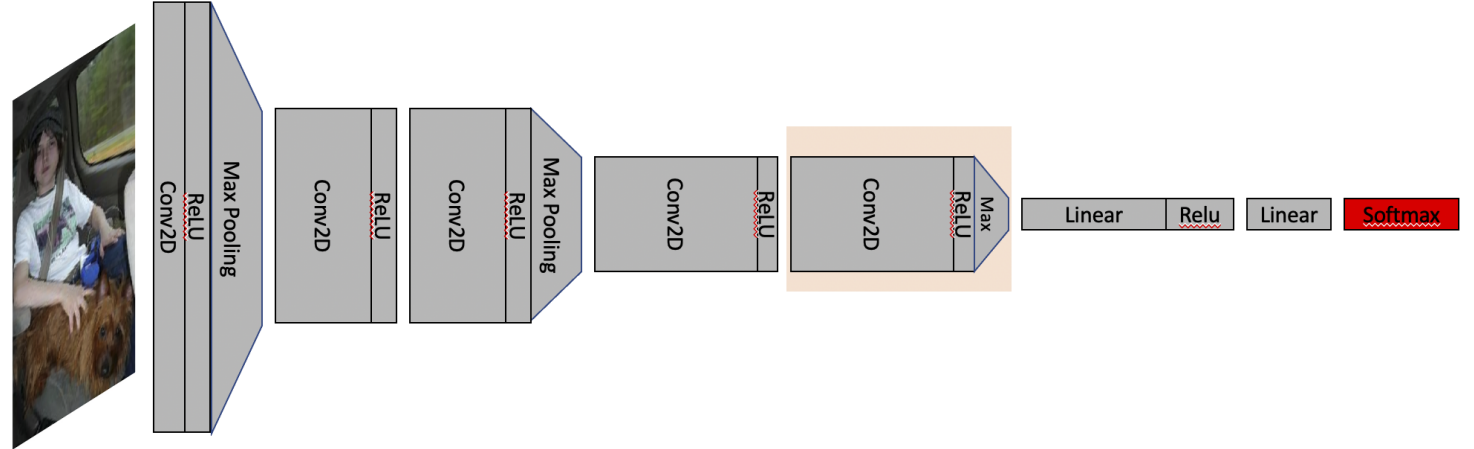


1. Compute the gradient of the target class score S_c w.r.t. the feature map $f_k(x, y) \in \mathbb{R}^{u \times v}$:

$$\underbrace{\frac{\partial S_c}{\partial f_k(x, y)}}_{\text{gradients via backprop}}$$

Gradient-Weighted Class Activation Mapping (Grad-CAM)

Pixel-space gradient + CAM



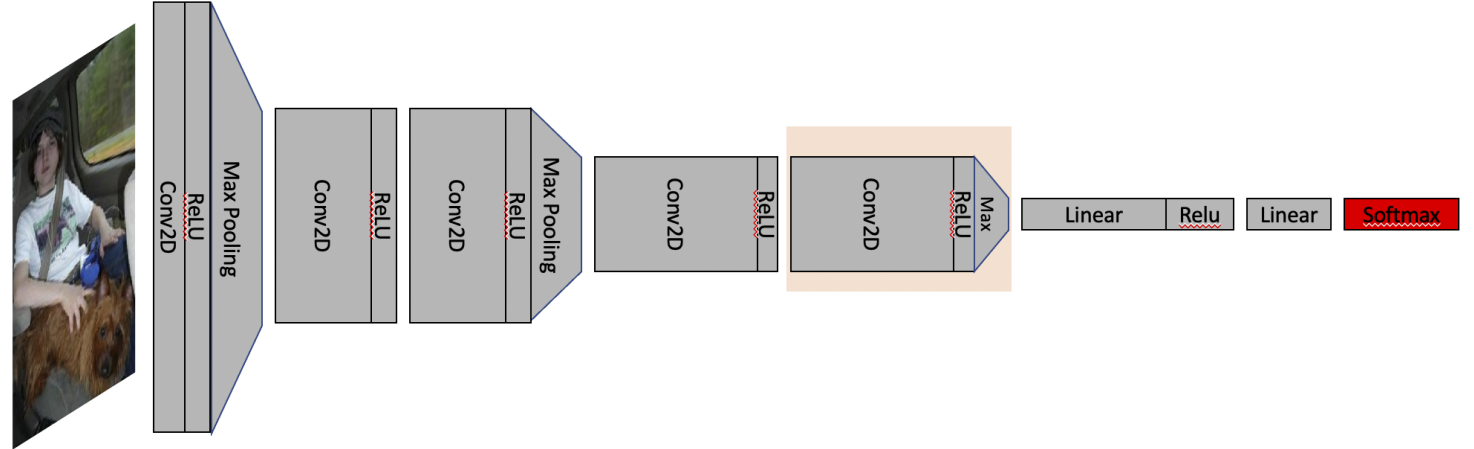
1. Compute the gradient of the target class score S_c w.r.t. the feature map $f_k(x, y) \in \mathbb{R}^{u \times v}$:
2. Global Average Pool the gradients:

'importance' weights of feature map k

$$\alpha_k = \frac{1}{Z} \sum_{x,y} \underbrace{\frac{\partial S_c}{\partial f_k(x, y)}}_{\text{gradients via backprop}}$$

Gradient-Weighted Class Activation Mapping (Grad-CAM)

Pixel-space gradient + CAM



1. Compute the gradient of the target class score S_c w.r.t. the feature map $f_k(x, y) \in \mathbb{R}^{u \times v}$:
2. Global Average Pool the gradients:

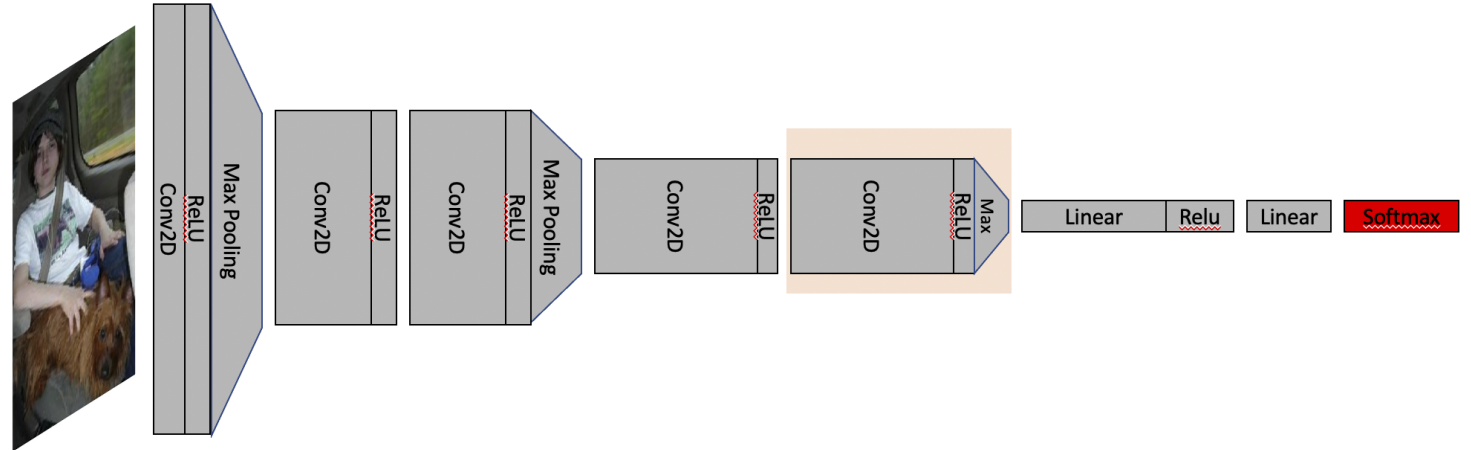
$$\text{'importance' weights of feature map } k \quad \alpha_k = \frac{1}{Z} \sum_{x,y} \underbrace{\frac{\partial S_c}{\partial f_k(x, y)}}_{\text{gradients via backprop}}$$

3. Compute a linear combination of the activations + ReLU:

$$M_{\text{Grad-CAM}}^c = \text{ReLU} \left(\sum_k \alpha_k f_k \right)$$

Gradient-Weighted Class Activation Mapping (Grad-CAM)

Pixel-space gradient + CAM



1. Compute the gradient of the target class score S_c w.r.t. the feature map $f_k(x, y) \in \mathbb{R}^{u \times v}$:
2. Global Average Pool the gradients:

$$\text{'importance' weights of feature map } k \quad \alpha_k = \frac{1}{Z} \sum_{x,y} \underbrace{\frac{\partial S_c}{\partial f_k(x, y)}}_{\text{gradients via backprop}}$$

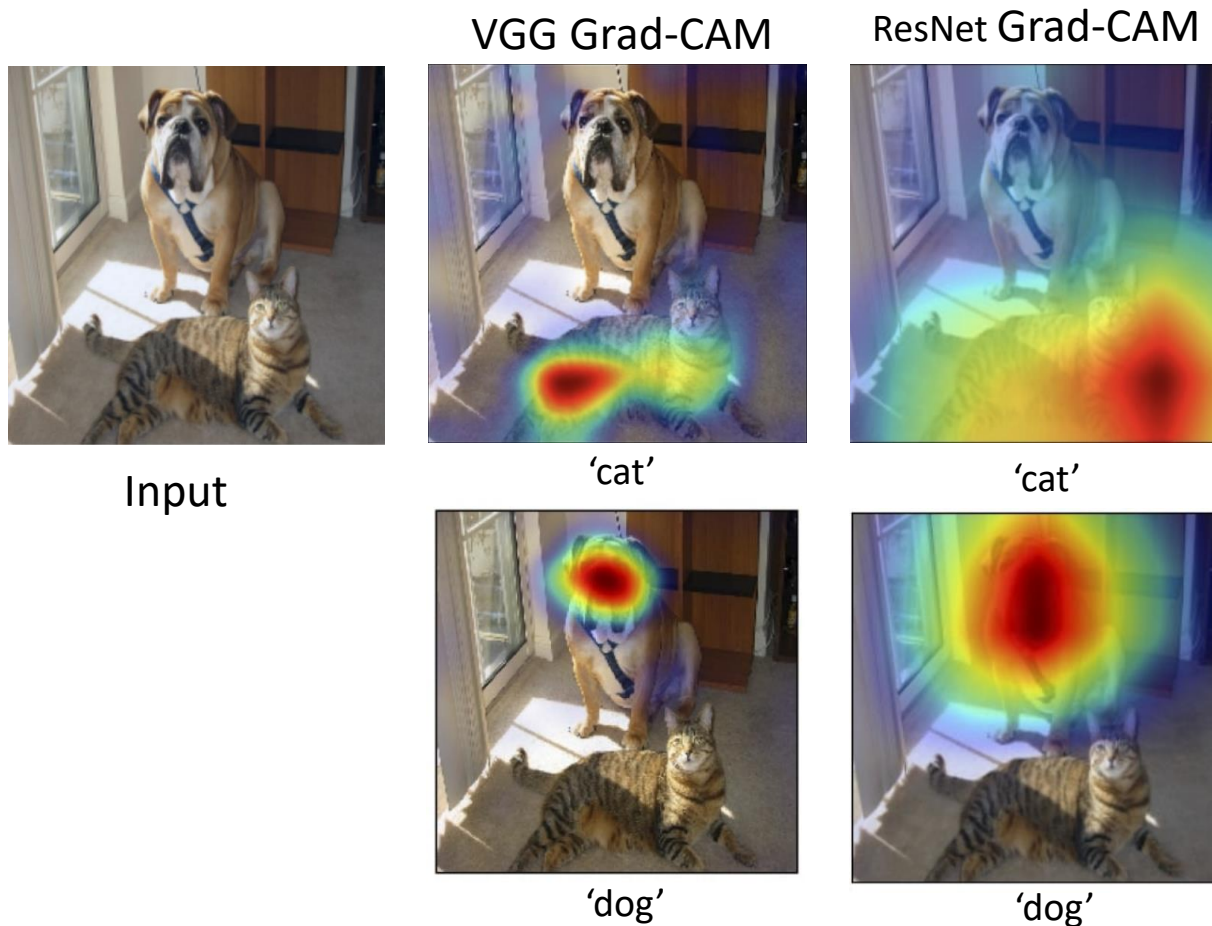
3. Compute a linear combination of the activations + ReLU:

$$M_{\text{CAM}}^c = \sum_k w_k f_k \quad \text{vs.} \quad M_{\text{Grad-CAM}}^c = \text{ReLU} \left(\sum_k \alpha_k f_k \right)$$

$w_k = \alpha_k$
See proof in the paper

Gradient-Weighted Class Activation Mapping (Grad-CAM)

- Grad-CAM can be applied to arbitrary layers
- Grad-CAM can be applied to existing state-of-the-art models



A man is sitting at a table with a pizza

Feature Visualization via Image Synthesis

What kind of input would maximize the activation of a certain neuron?

Solve for an image I that maximizes the objective:

$$\arg \max_I S_j^l(I) - \lambda R(I)$$

Neuron activation Natural image prior

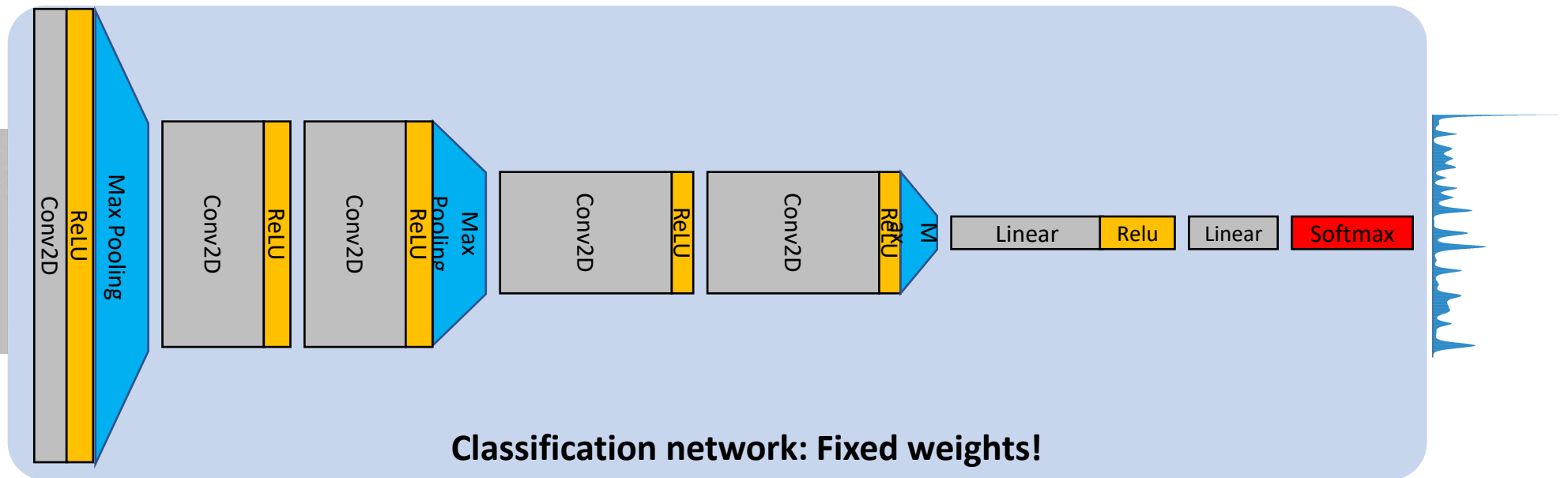
No image prior



The variables of optimization are the image values



Unknown Image I



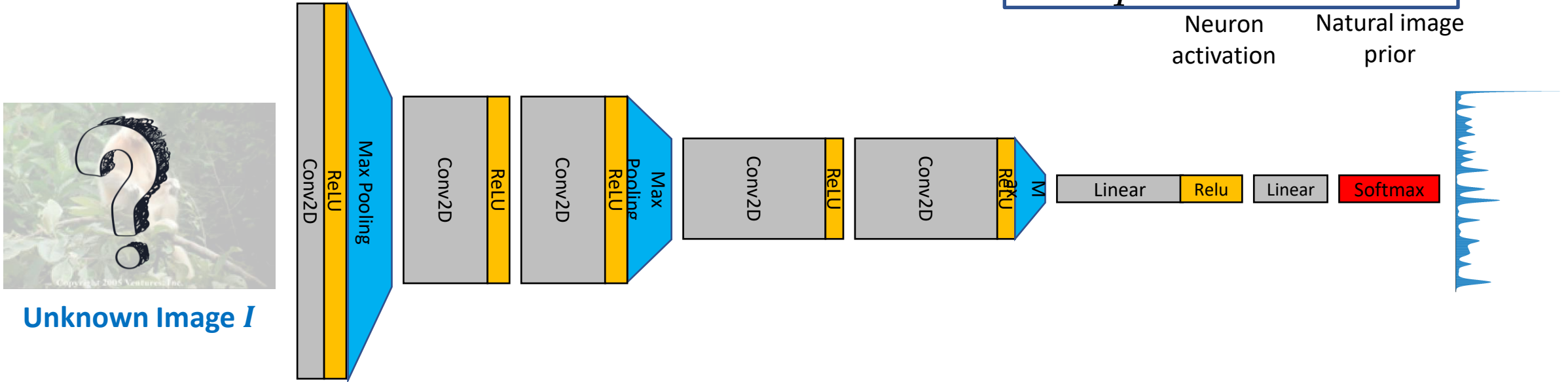
Classification network: Fixed weights!

Feature Visualization via Image Synthesis

Solve for an image I that maximizes the objective:

$$\arg \max_I S_j^l(I) - \lambda R(I)$$

Neuron activation Natural image prior



Initlize the image (zeros / mean test image / random noise)

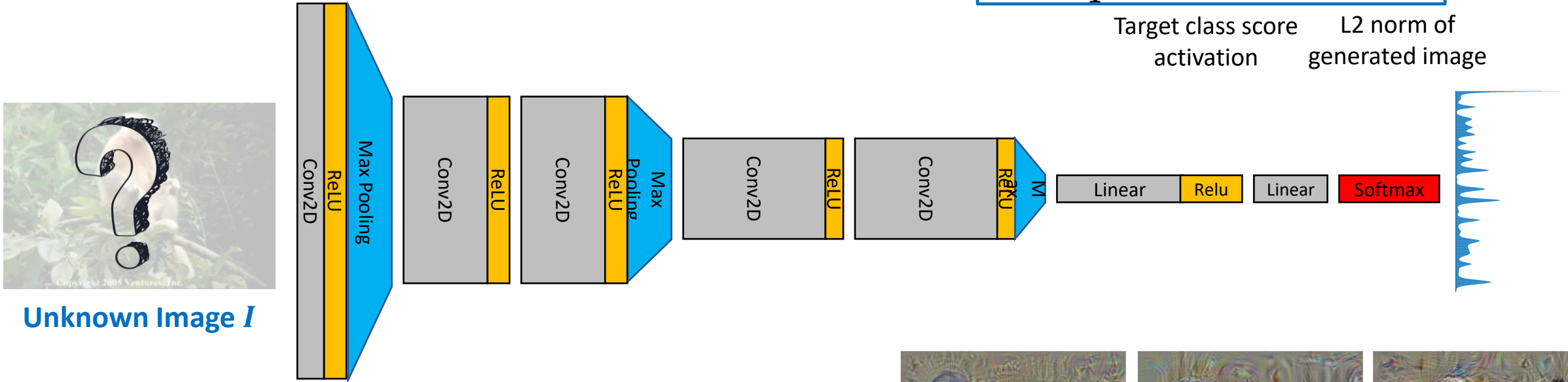
Repeat:

1. Feed into the network to compute $S_j^l(I)$
2. Compute gradients: $\partial S_j^l / \partial I$ via backproption
3. Update the image

Feature Visualization via Image Synthesis

Solve for an image I that maximizes the objective: $\arg \max_I S_c(I) - \lambda ||I||_2^2$

Target class score L2 norm of
activation generated image

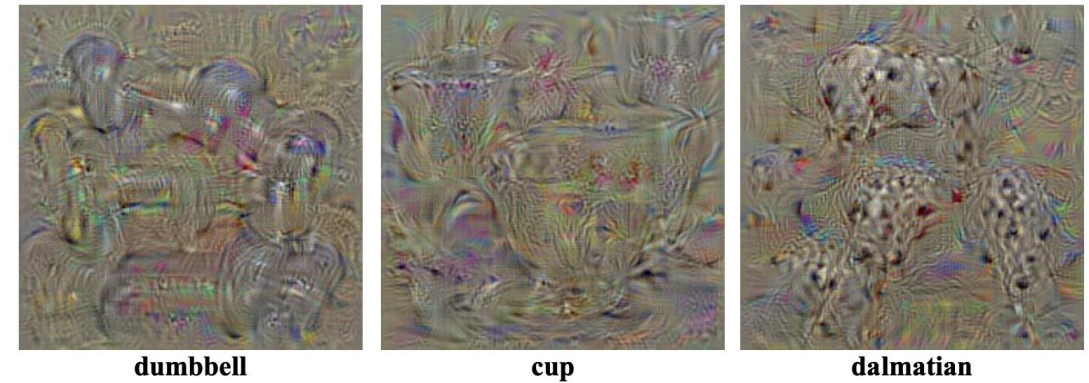


Unknown Image I

Initlize the image (zeros / mean test image / random noise)

Repeat:

1. Feed into the network to compute $S_j^l(I)$
2. Compute gradients: $\frac{\partial S_j^l}{\partial I}$ via backproption
3. Update the image



Feature Visualization via Image Synthesis

$$\arg \max_I S_j^l(I) - \lambda R(I)$$

Neuron activation Natural image prior



Image from: <https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>

Feature Visualization via Image Synthesis

$$\arg \max_I S_j^l(I) - \lambda R(I)$$

Neuron activation
Natural image prior


Can we design a more “fancy” natural image prior?

No regularization




Learned priors
(e.g., GANs)


Image naturalness

- 


Erhan, et al., 2009 [3]
Introduced core idea. Minimal regularization.

- 


Szegedy, et al., 2013 [11]
Adversarial examples. Visualizes with dataset examples.

- 


Mahendran & Vedaldi, 2015 [7]
Introduces total variation regularizer. Reconstructs input from representation.

- 


Nguyen, et al., 2015 [14]
Explores counterexamples. Introduces image blurring.

- 


Mordvintsev, et al., 2015 [4]
Introduced jitter & multi-scale. Explored GMM priors for classes.

- 


Øygaard, et al., 2015 [15]
Introduces gradient blurring. (Also uses jitter.)

- 


Tyka, et al., 2016 [16]
Regularizes with bilateral filters. (Also uses jitter.)

- 

Mordvintsev, et al., 2016 [17]
Normalizes gradient frequencies. (Also uses jitter.)

- 

Nguyen, et al., 2016 [18]
Parameterizes images with GAN generator.

- 

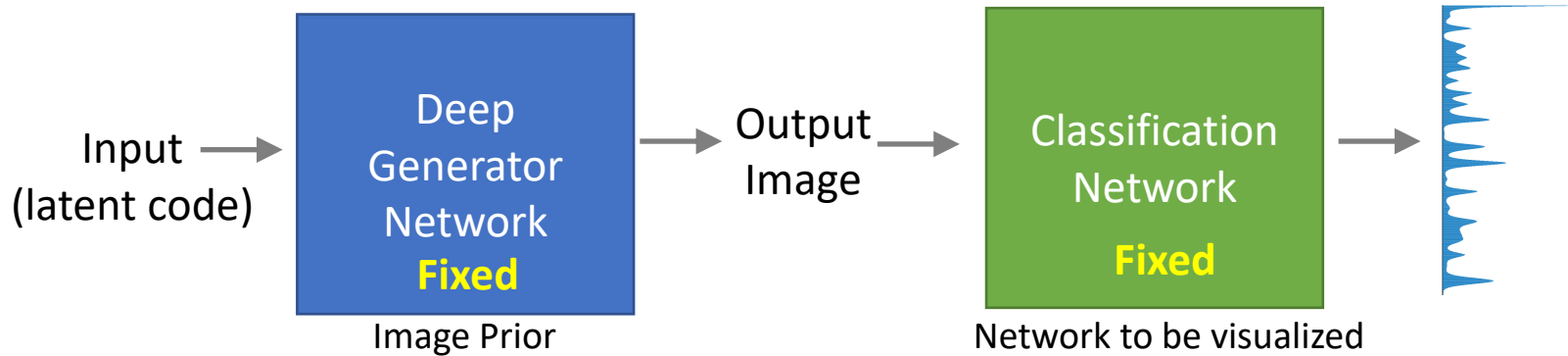
Nguyen, et al., 2016 [10]
Uses denoising autoencoder prior to make a generative model.

<https://distill.pub/2017/feature-visualization/#learned-priors>

Feature Visualization via Image Synthesis

$$\arg \max_I S_j^l(I) - \lambda R(I)$$

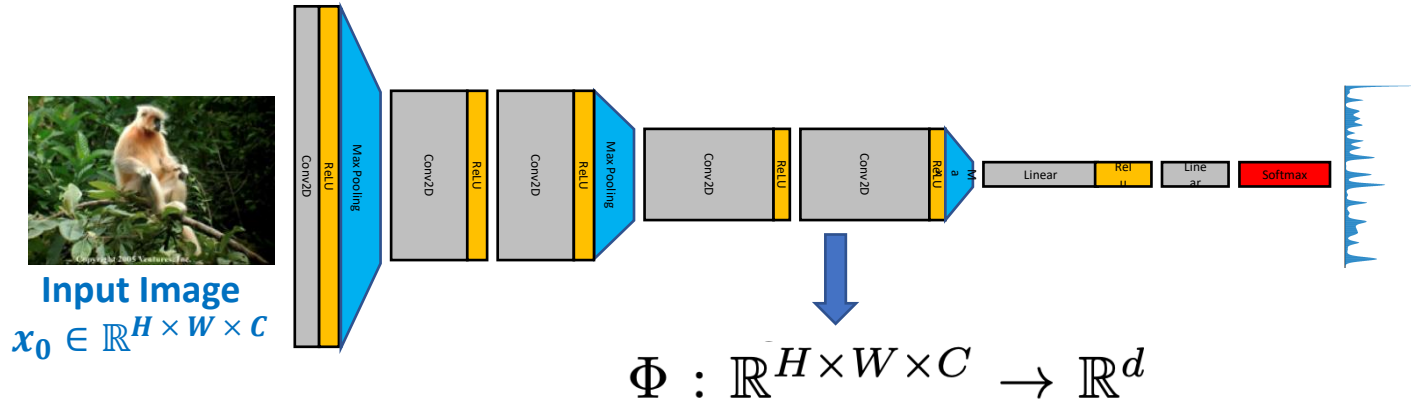
Neuron activation
Natural image prior



Feature Inversion

Input: Target feature vector at some layer/s (computed from a given image)

Output: new “natural” image that matches the target feature



$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathbb{R}^{H \times W \times C}}{\operatorname{argmin}} \underbrace{\ell(\Phi(\mathbf{x}), \Phi_0)}_{\text{Feature data term}} + \underbrace{\lambda \mathcal{R}(\mathbf{x})}_{\text{Natural image prior}}$$

Euclidian distance between the features:

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

Total Variation (TV) regularizer (encourages spatial smoothness)

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

Feature Inversion

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

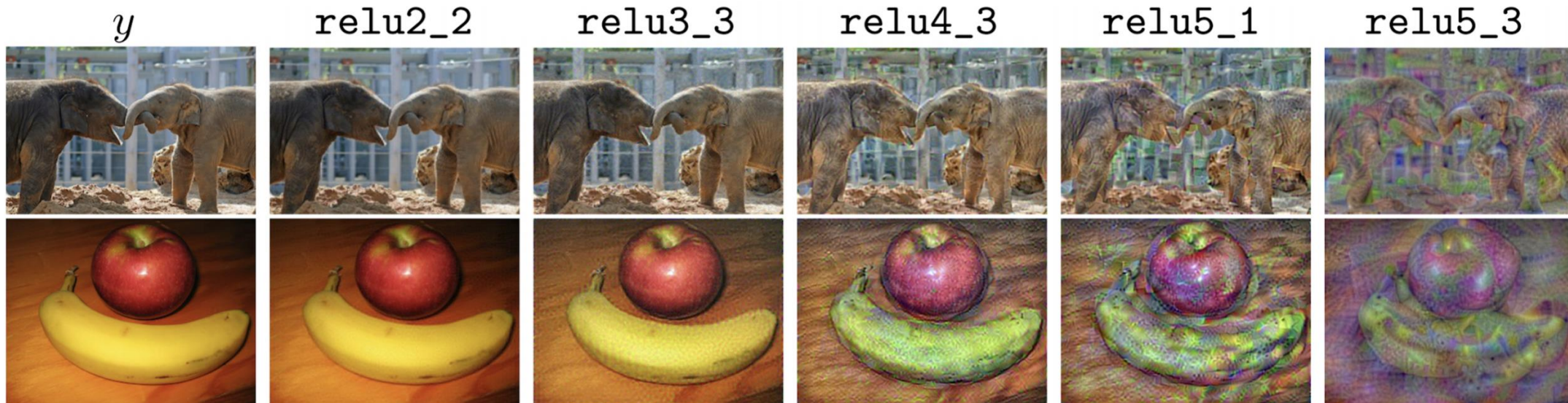
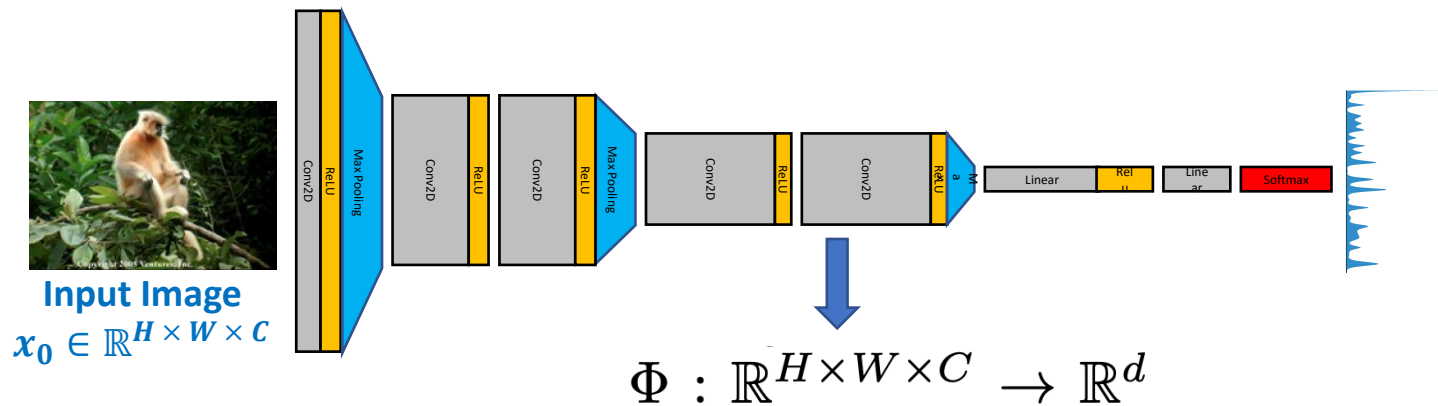
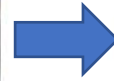


Figure from: "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016

Neural Texture Synthesis and Style Transfer



Original
Texture image



Output
New texture image



Content Image A



Style Image B



Stylized image
(content A + Style B)

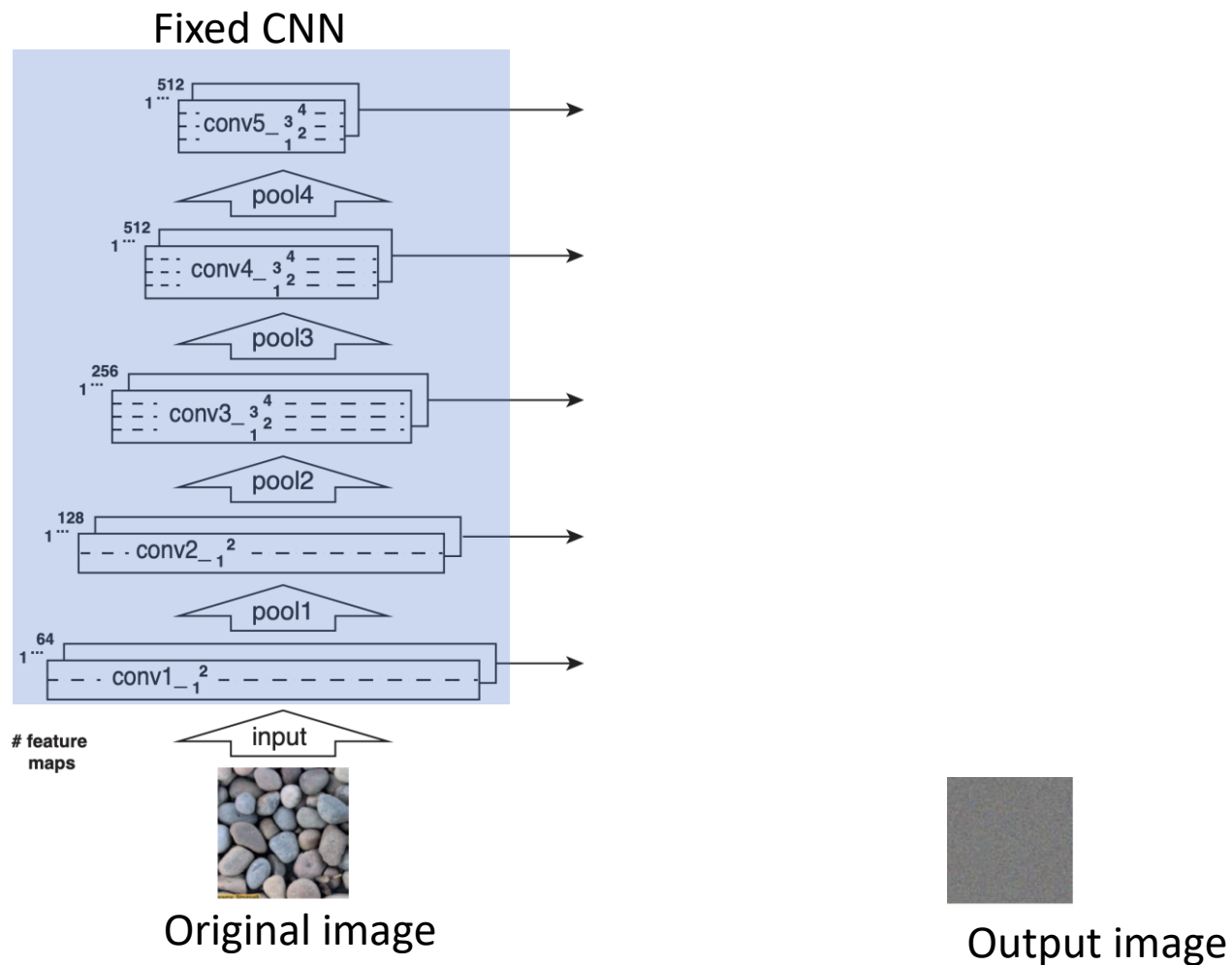
Neural Texture Synthesis

Loss that captures texture similarity

$$I^* = \arg \min_I \sum_l E_l$$

Input: texture image, pre-trained CNN

- * Feed original image to get feature activations
- * Initialize the output image (random noise)



Neural Texture Synthesis

Input: texture image, pre-trained CNN

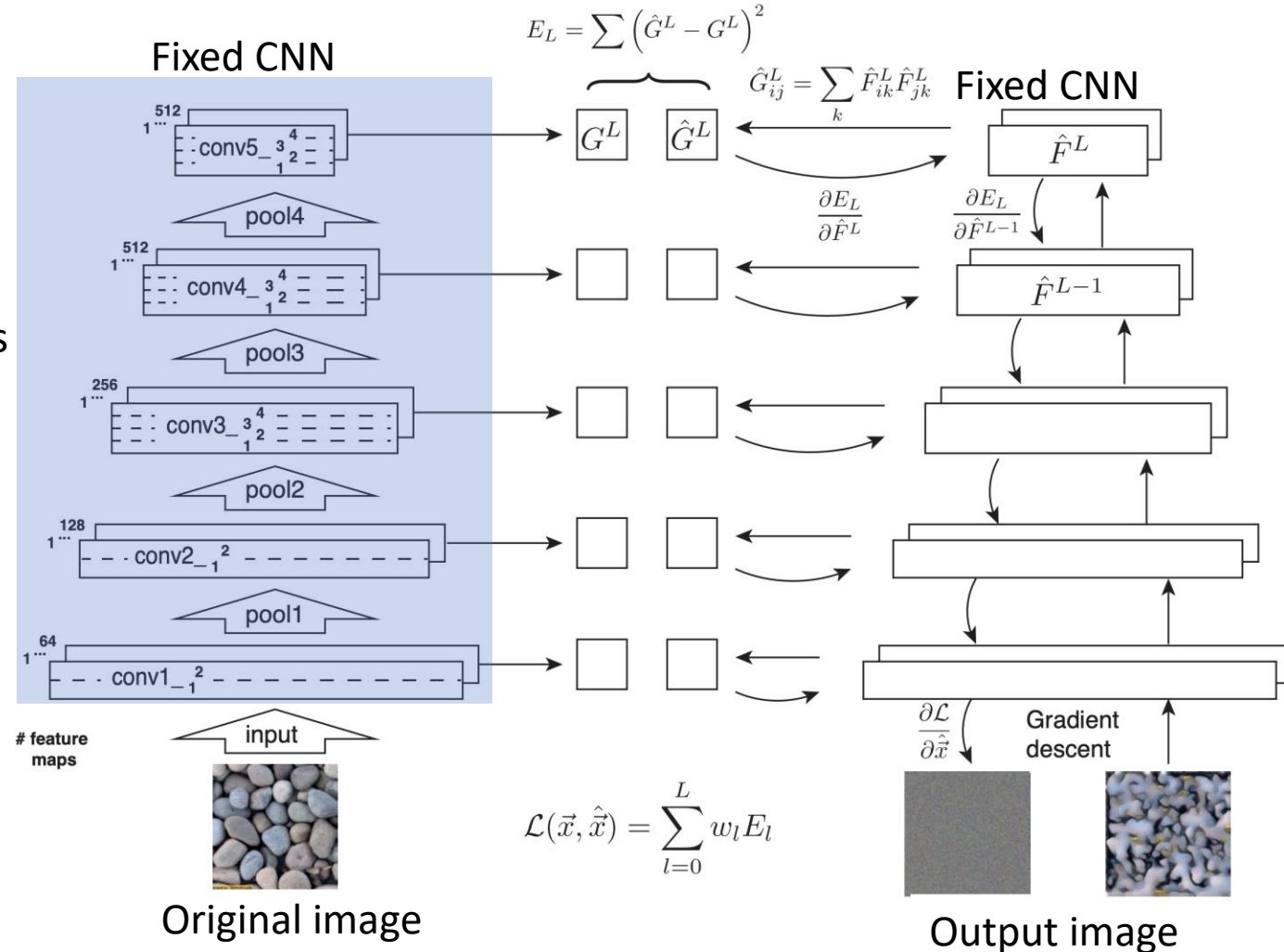
- * Initlize the output image (random noise)
- * Feed original image to get feature activations

Repeat:

1. Feed output image to compute output activations
2. Compute gradients of the loss using backprop.
3. Update the image

Loss that captures texture similarity

$$I^* = \arg \min_I \sum_l w_l E_l$$



Neural Texture Synthesis

Loss that captures
texture similarity

$$I^* = \arg \min_I \sum_l w_l E_l$$

A Parametric Texture Model Based on Joint Statistics of Complex Wavelet Coefficients

JAVIER PORTILLA AND EERO P. SIMONCELLI

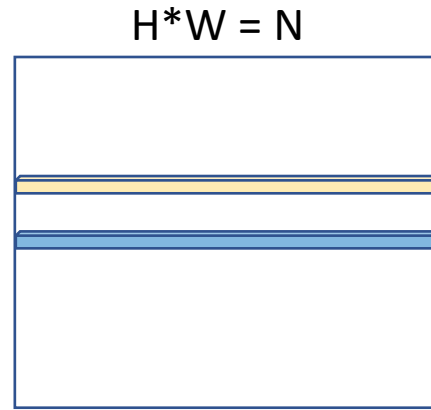
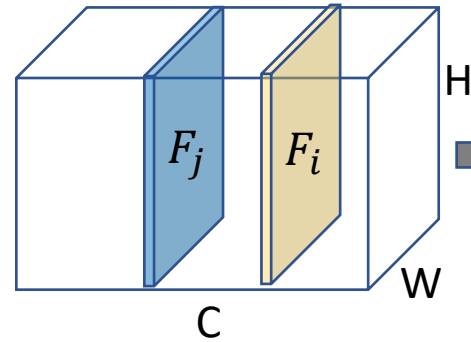
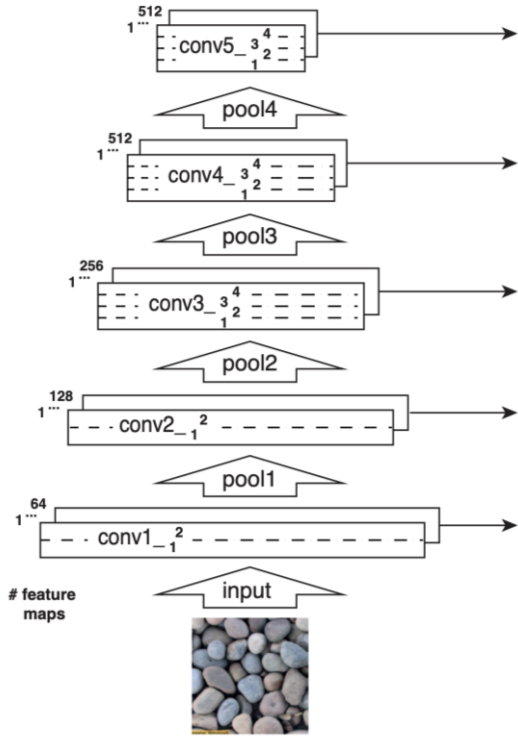
*Center for Neural Science, and Courant Institute of Mathematical Sciences, New York University,
New York, NY 10003, USA*

Received November 12, 1999; Revised June 9, 2000

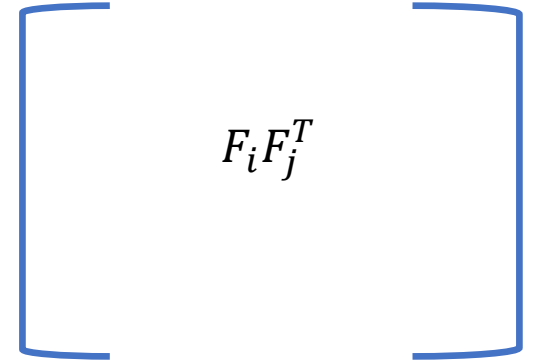


Neural Texture Synthesis: Gram Matrix

Each layer is a $C \times H \times W$ tensor of features;
 We have C feature maps (each of resolution $H \times W$)



$F_i \in \mathcal{R}^N$
 $F_j \in \mathcal{R}^N$



$C \times C$
 Gram Matrix

Loss that captures
 texture similarity

$$I^* = \arg \min_I \sum_l w_l E_l$$

Each element is the correlation
 between two features

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

Neural Texture Synthesis

Input: texture image, pre-trained CNN

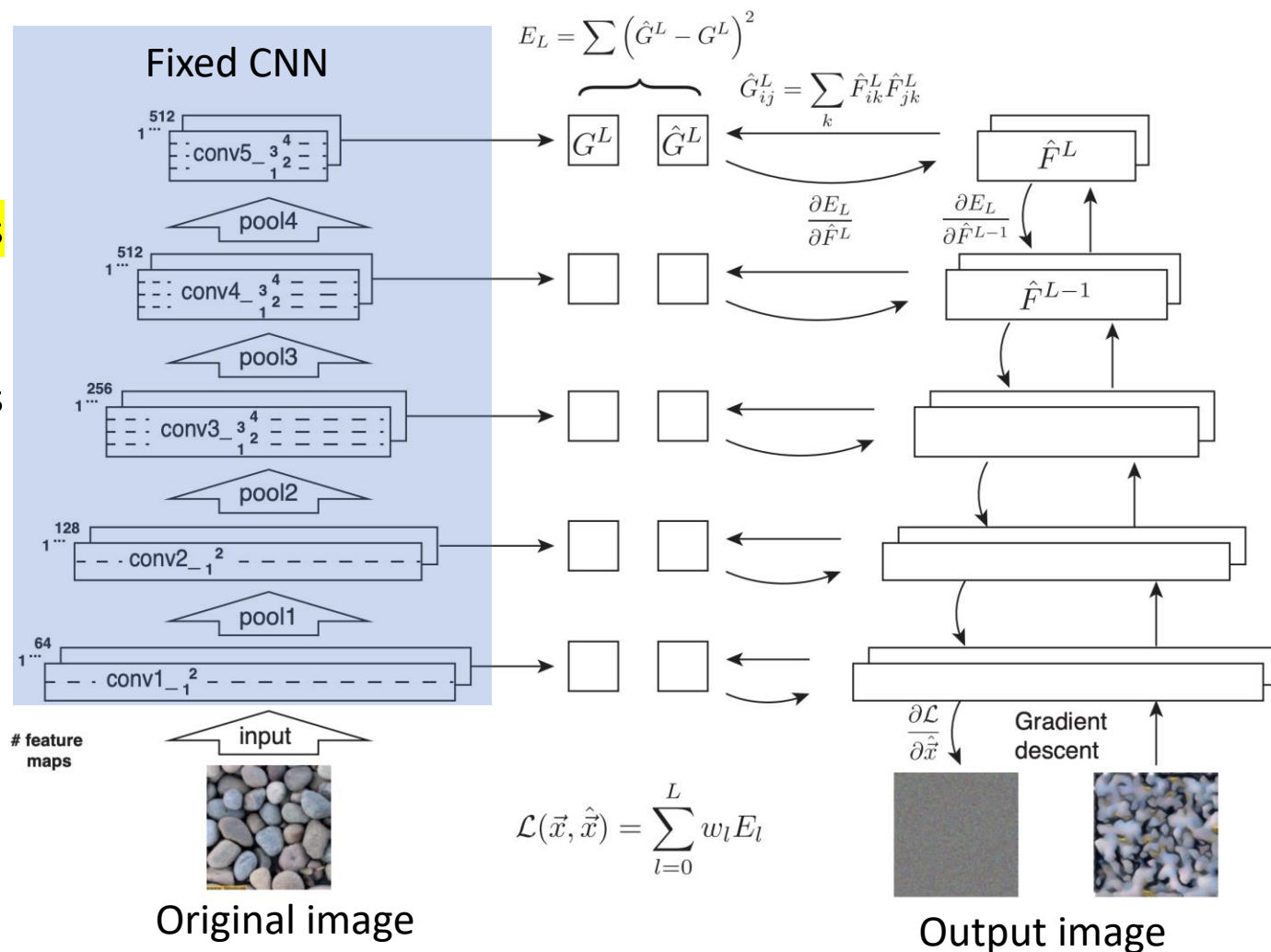
- * Initlize the output image (random noise)
- * Feed original image to get feature activations
- * Compute original Gram matrices for different layers

Repeat:

1. Feed output image to compute output activations
2. Compute Gram-matrices for different layers
3. Compute gradients of the loss using backprop.
4. Update the image

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

$$I^* = \arg \min_I \sum_l w_l E_l$$



Neural Texture Synthesis

Patterns/structures size increases as we use deeper layers



Neural Style Transfer



Content Image A



Style/Texture Image B



Stylized image
(content A + Style B)

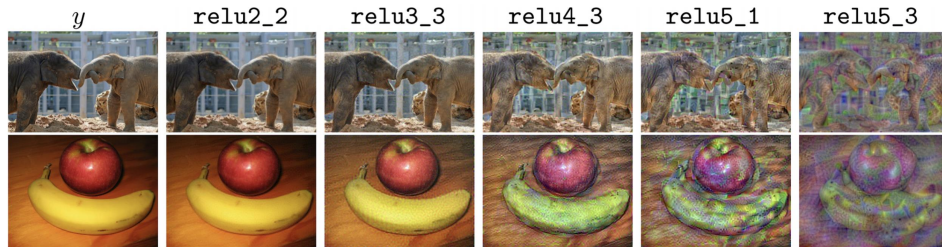
Match features of the
content image



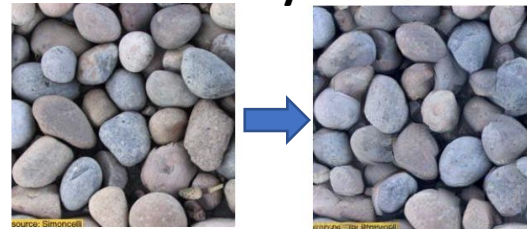
Match Gram matrices
of the style image



Feature Inversion (reconstruction)



Texture Synthesis



Original
Texture image

Output
New texture image

$$\mathcal{L}_{\text{content}}(A, y) + \alpha \mathcal{L}_{\text{style}}(B, y)$$

Neural Style Transfer

$$\mathcal{L}_{\text{content}}(A, y) + \alpha \mathcal{L}_{\text{style}}(B, y)$$

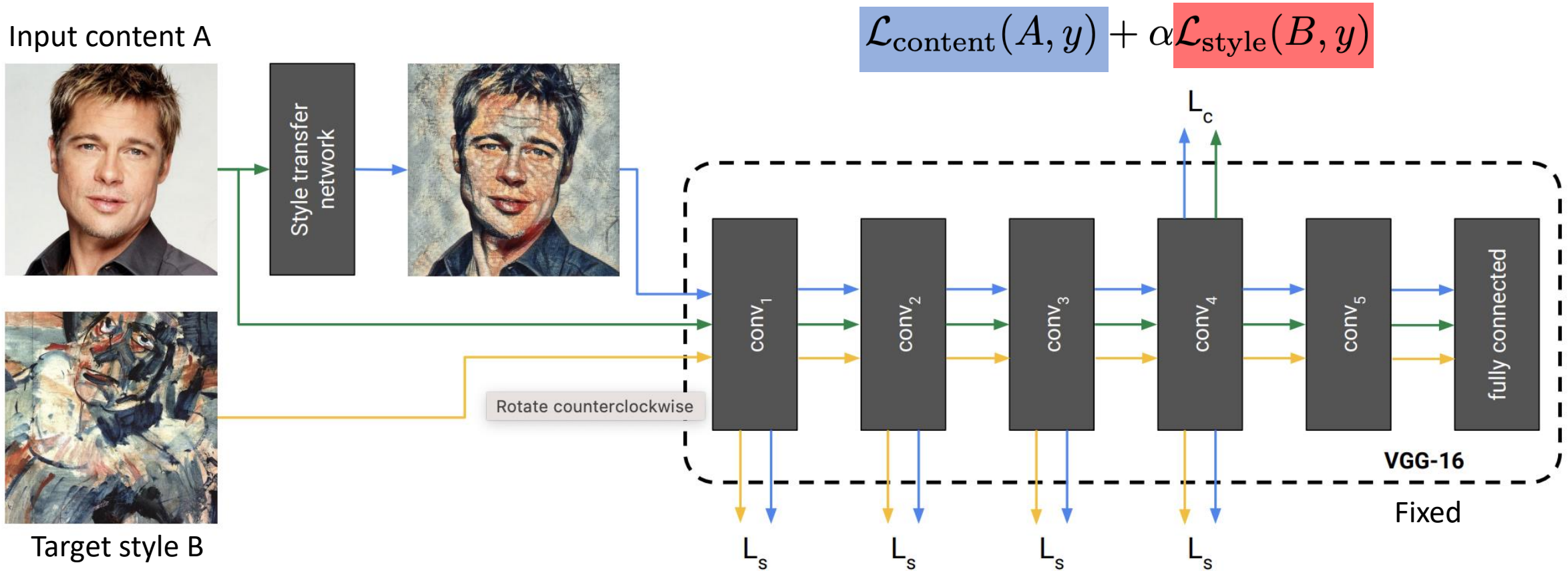


Figure from: <https://github.com/jcjohnson/neural-style>

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

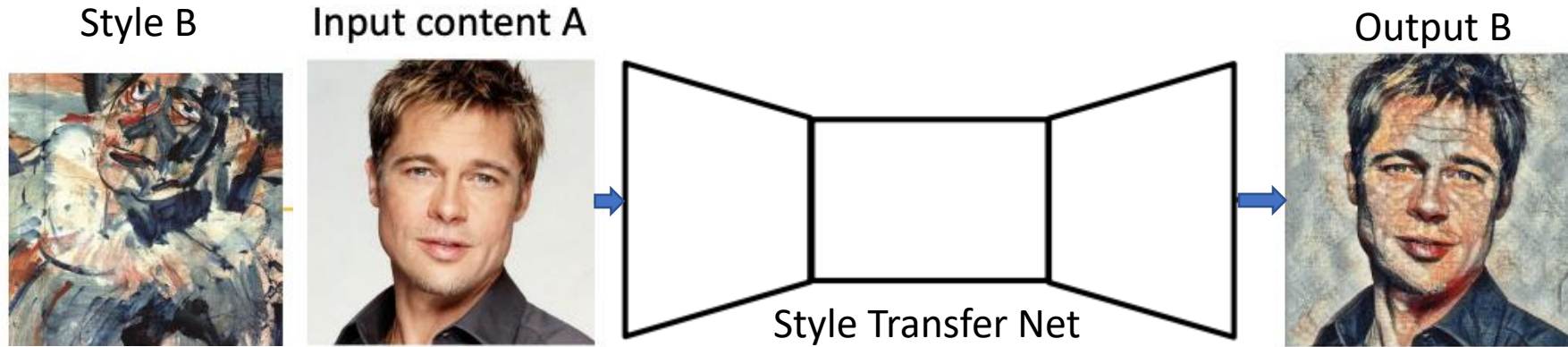
Feed Forward Style Transfer (per style)

Train a feedforward network for each style, use **perceptual loss** for training



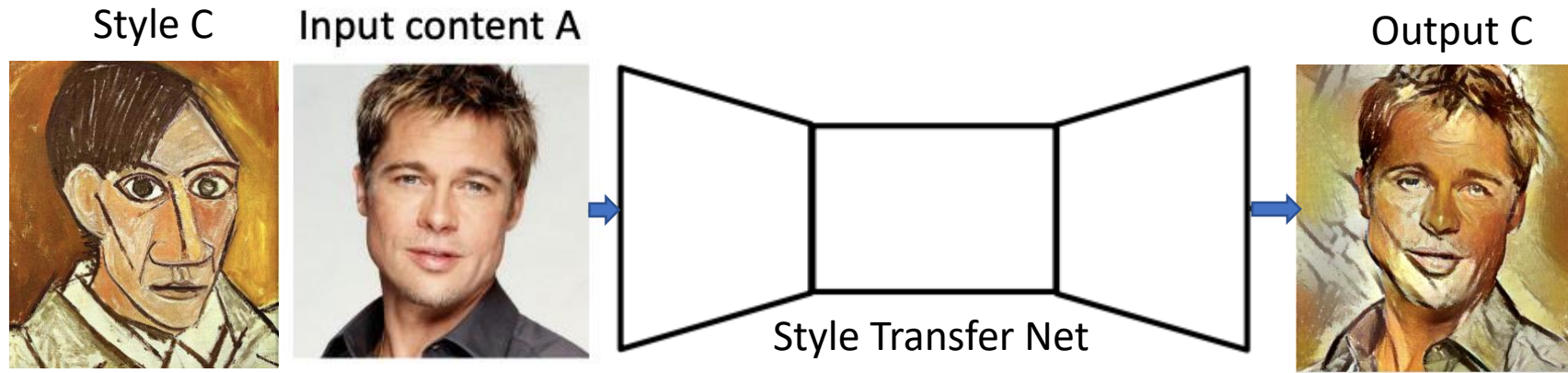
Feed Forward Style Transfer (many styles)

Train a feedforward network for **multiple** styles



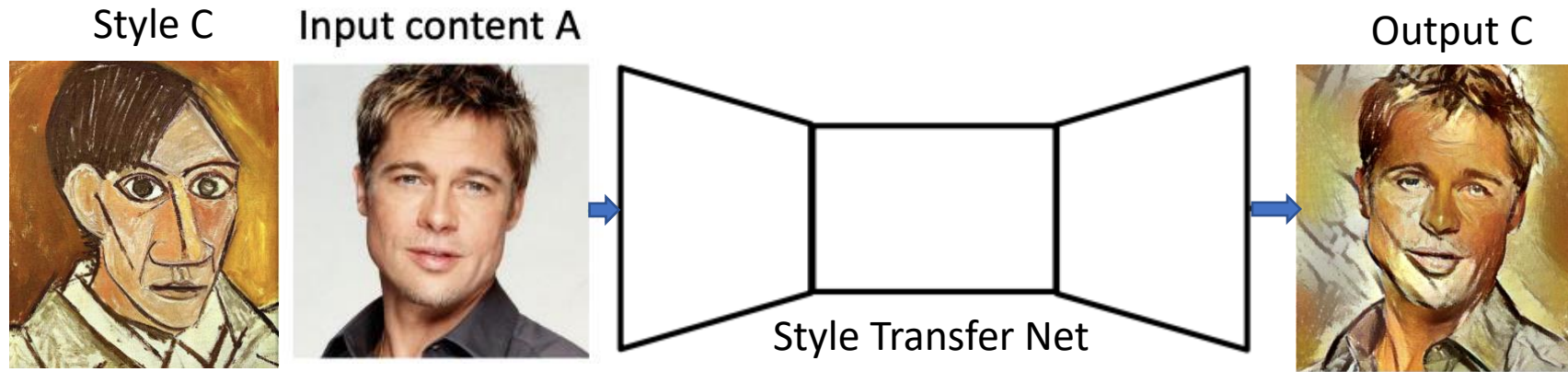
Feed Forward Style Transfer (many styles)

Train a feedforward network for **multiple** styles



Feed Forward Style Transfer (many styles)

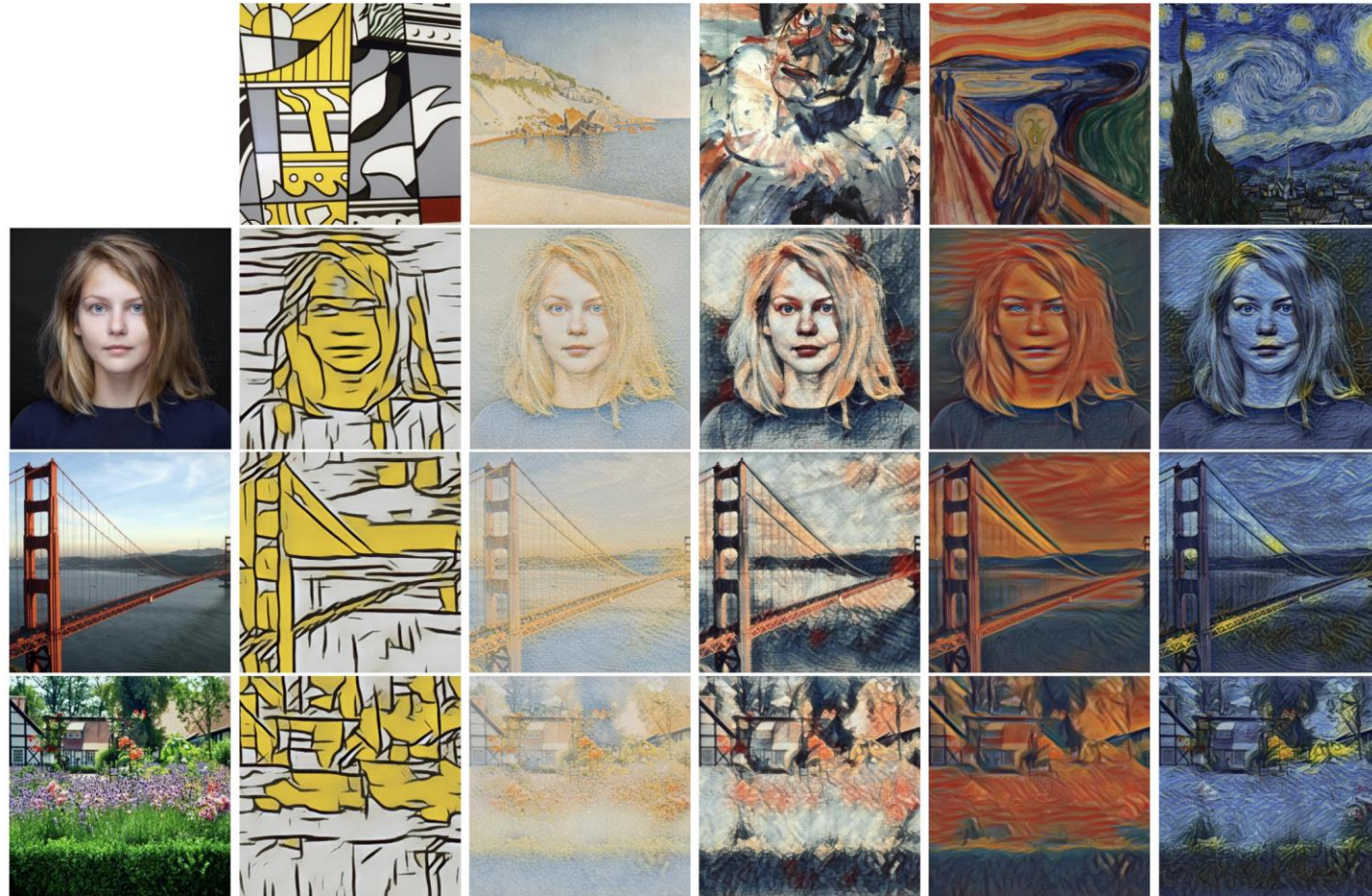
Train a feedforward network for **multiple** styles, using conditional **Instance Normalization**:



After each CONV layer in the Style Transfer Net, features are normalized, then scaled + shifted:

Feed Forward Style Transfer (many styles)

Train a feedforward network for **multiple** styles, using conditional **Instance Normalization**:

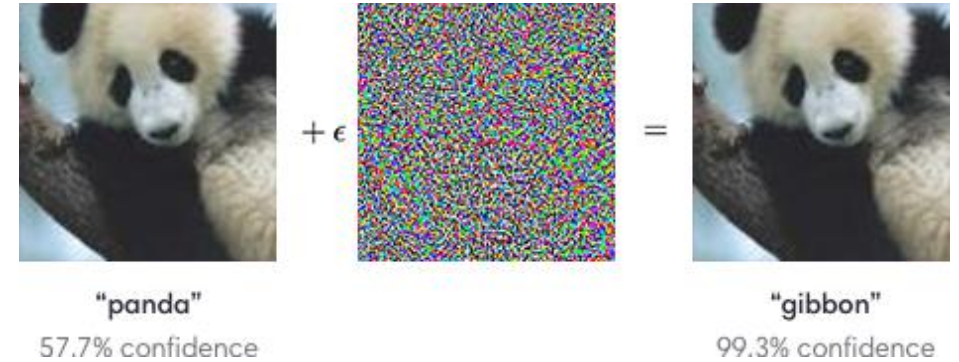


Next tutorial:

“Adversarial Examples”



Niv Haim



Next class:

“Sequences” (RNN, Attention, ViT)



Shai Bagon