Visualizing and Understanding Neural Networks

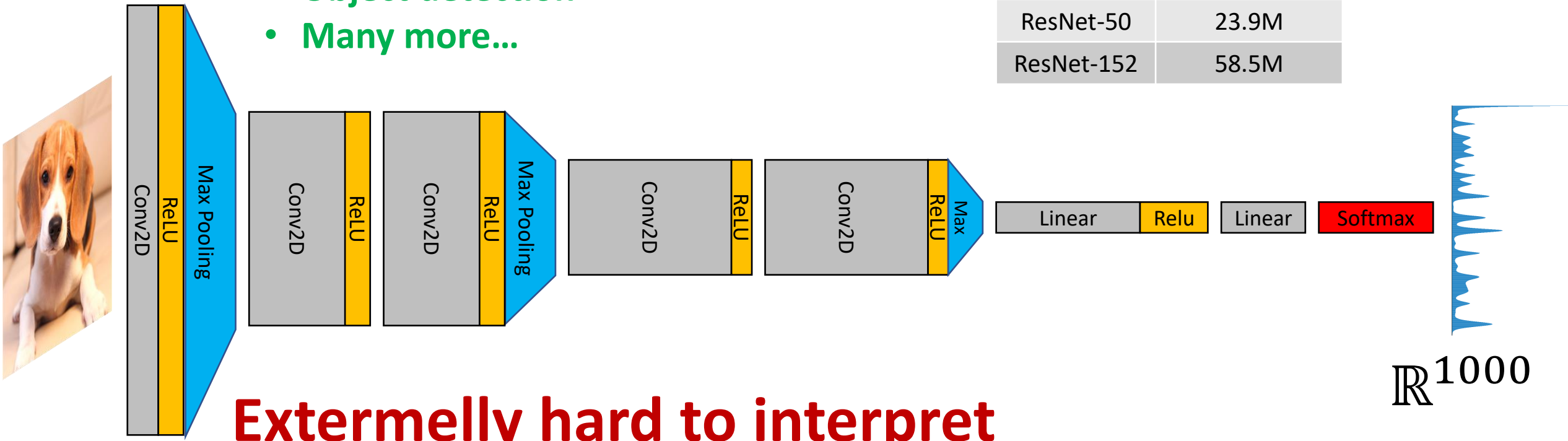
Jan 8th, 2024

Tali Dekel

State-of-the-art performance:

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

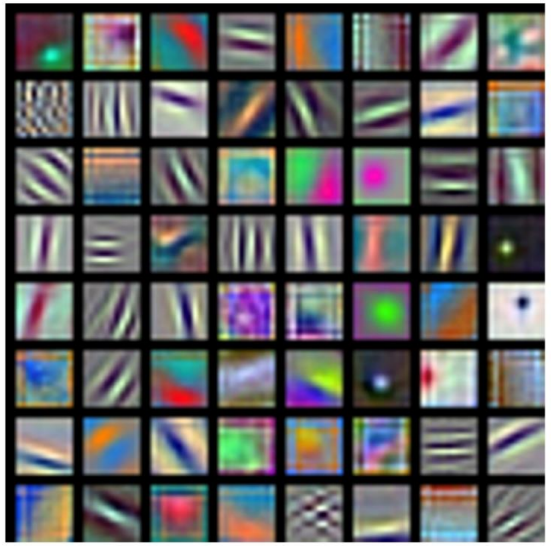
Model	# parameters
VGG-16	134.7M
ResNet-18	11.4M
ResNet-34	21.5M
ResNet-50	23.9M
ResNet-152	58.5M



Extermelly hard to interpret

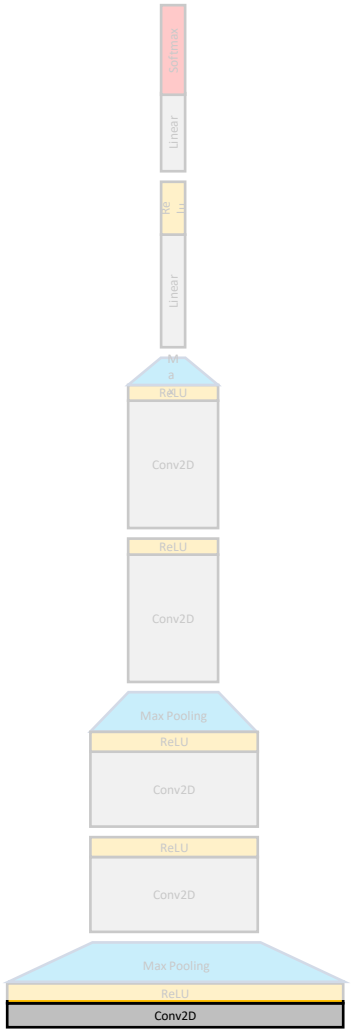
Visualizing the First Layer's Filters

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



AlexNet:
64 x 3 x 11 x 11

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

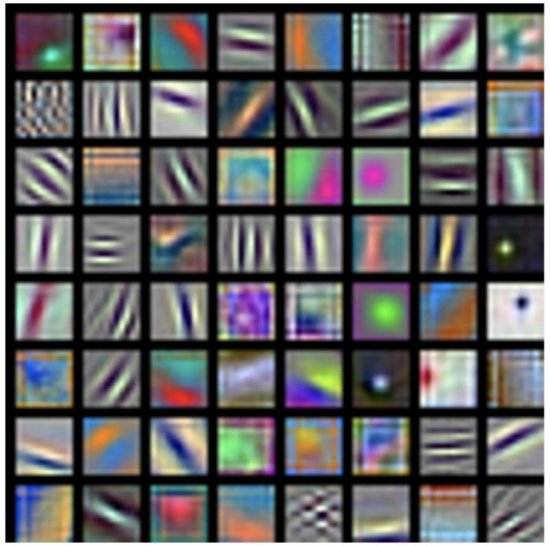


Slide from: Justin Jonshon, EECS498/CS231

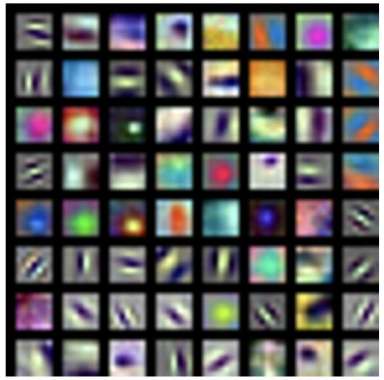
Krizhevsky, "One weird trick for parallelizing convolutional neural networks", arXiv 2014 He et al, "
Deep Residual Learning for Image Recognition", CVPR 2016 Huang et al,
"Densely Connected Convolutional Networks", CVPR 2017

Visualizing the First Layer's Filters

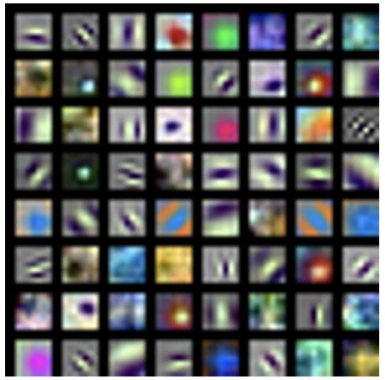
Visualize the linear weights in the **input-to-first layer**



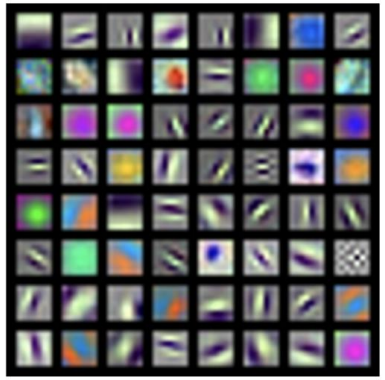
AlexNet:
64 x 3 x 11 x 11



ResNet-18:
64 x 3 x 7 x 7

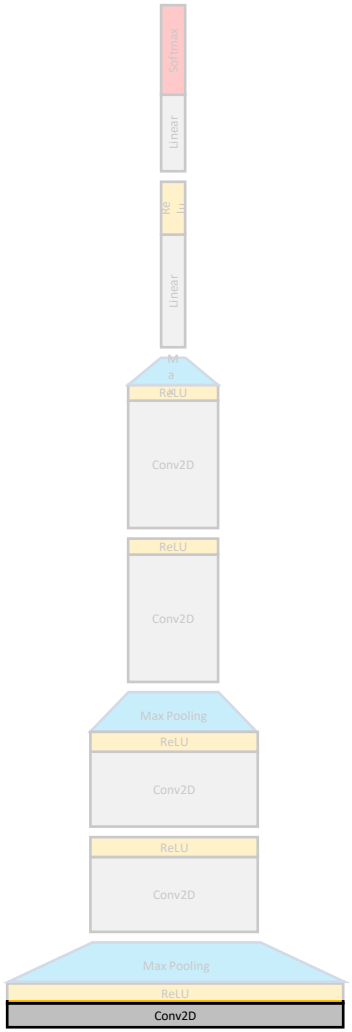


ResNet-101:
64 x 3 x 7 x 7



DenseNet-121:
64 x 3 x 7 x 7

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

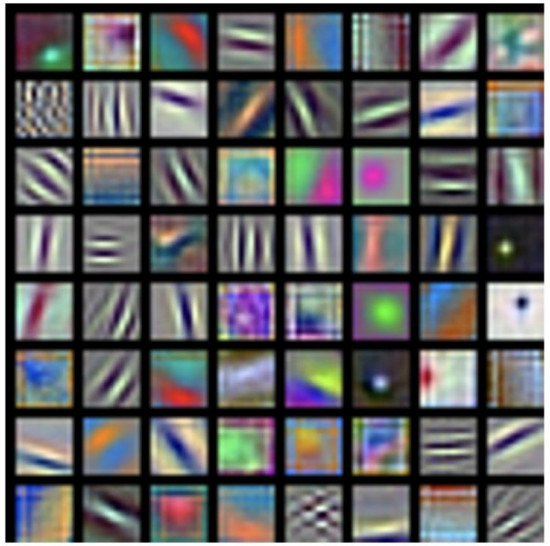


Slide from: Justin Jonshon, EECS498/CS231

Krizhevsky, "One weird trick for parallelizing convolutional neural networks", arXiv 2014 He et al, "
Deep Residual Learning for Image Recognition", CVPR 2016 Huang et al,
"Densely Connected Convolutional Networks", CVPR 2017

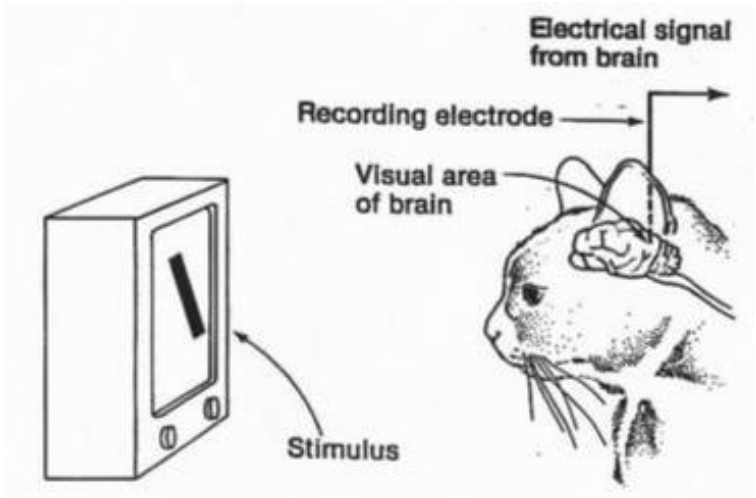
Visualizing the First Layer's Filters

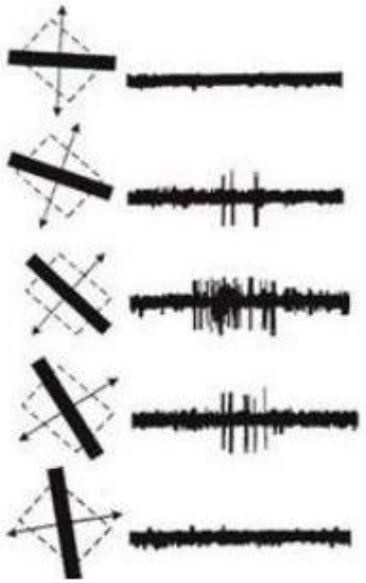
Visualize the linear weights in the **input-to-first layer**



AlexNet:
64 x 3 x 11 x 11

Hubel & Wiesel, 1959





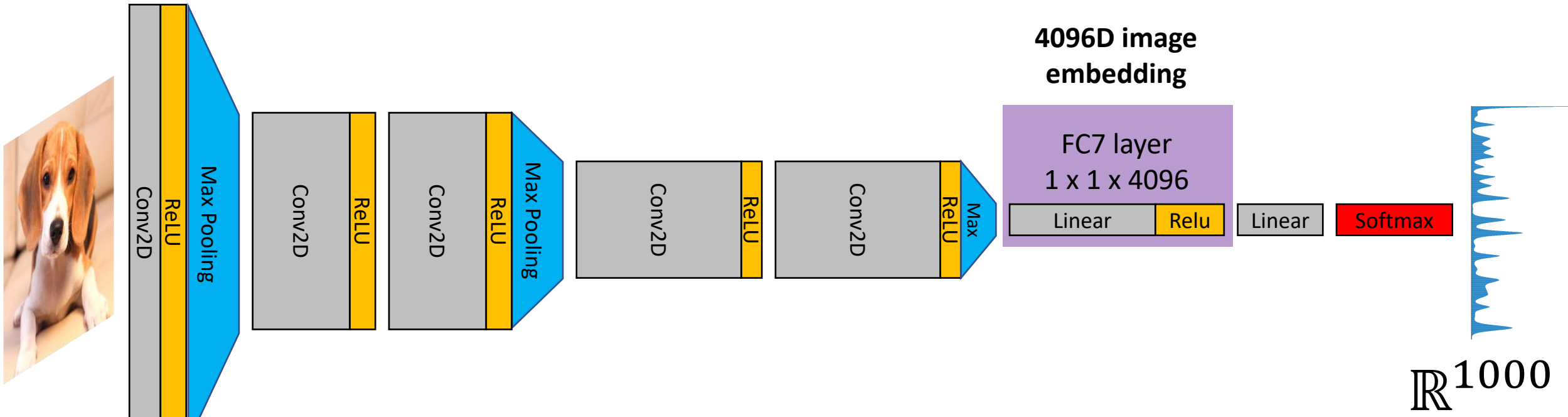
https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

Slide from: Justin Jonshon, EECS498/CS231

Krizhevsky, "One weird trick for parallelizing convolutional neural networks", arXiv 2014 He et al, "
Deep Residual Learning for Image Recognition", CVPR 2016 Huang et al,
"Densely Connected Convolutional Networks", CVPR 2017

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)

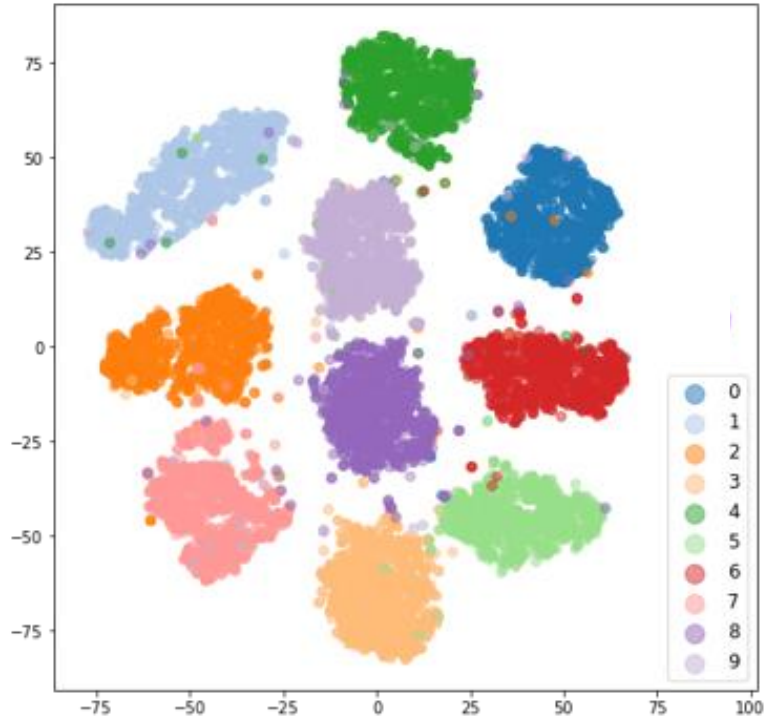
(1) Run many images in the network and record their features
(2) Compute similarity of a query image to the rest; retrieve nearest neighbours

Query

Krizhevsky et al, “ImageNet Classification with Deep Convolutional Neural Networks”, NeurIPS 2012

Deep Features as Image Embedding

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



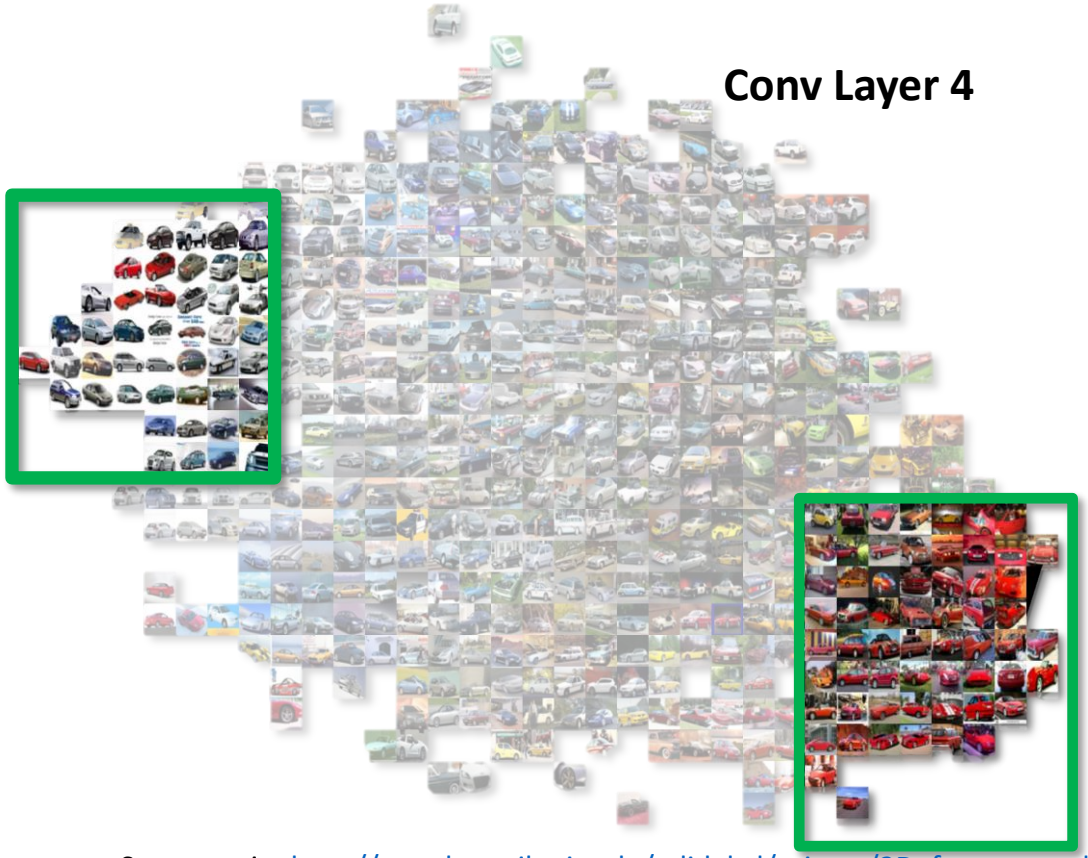
MNIST t-SNE embedding

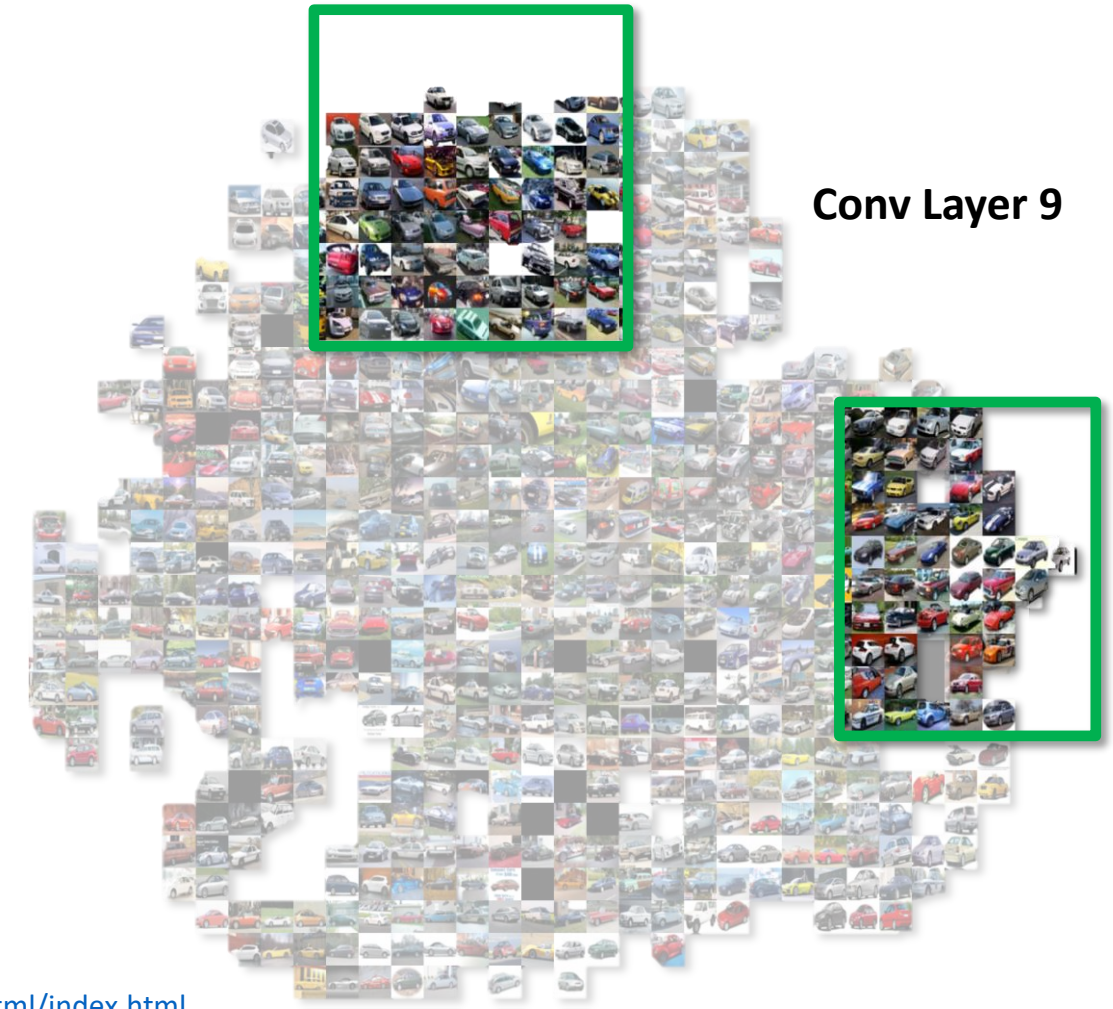
Plot from:
https://towardsdatascience.com/visualizing-feature-vectors-embeddings-using-pca-and-t-sne-ef157cea3a42

Deep Features as Image Embedding

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

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





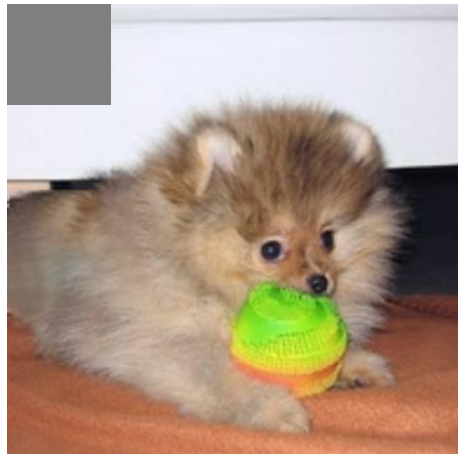
See more in: http://people.csail.mit.edu/talidekel/private/2D_feature_embedding_cifar_html/index.html
https://cs.stanford.edu/people/karpathy/cnnembed/

Saliency via occlusion

Which pixels matter?

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

True Label: Pomeranian



Probability of Pomeranian

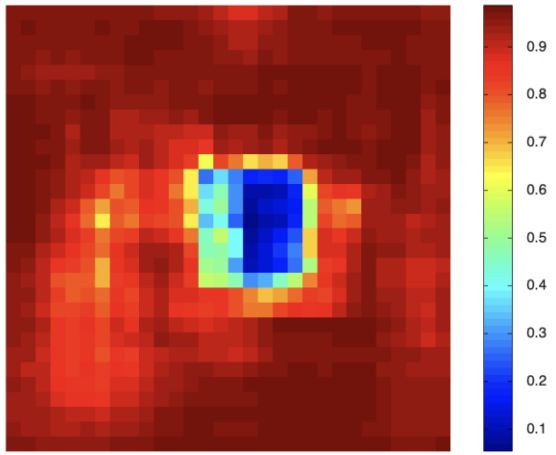
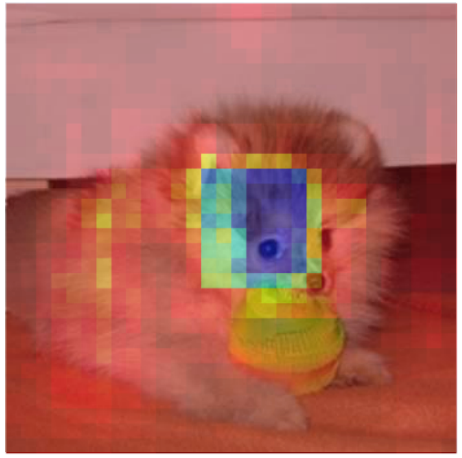
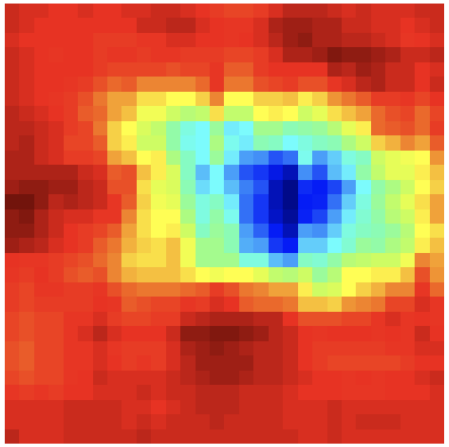


Image overlay



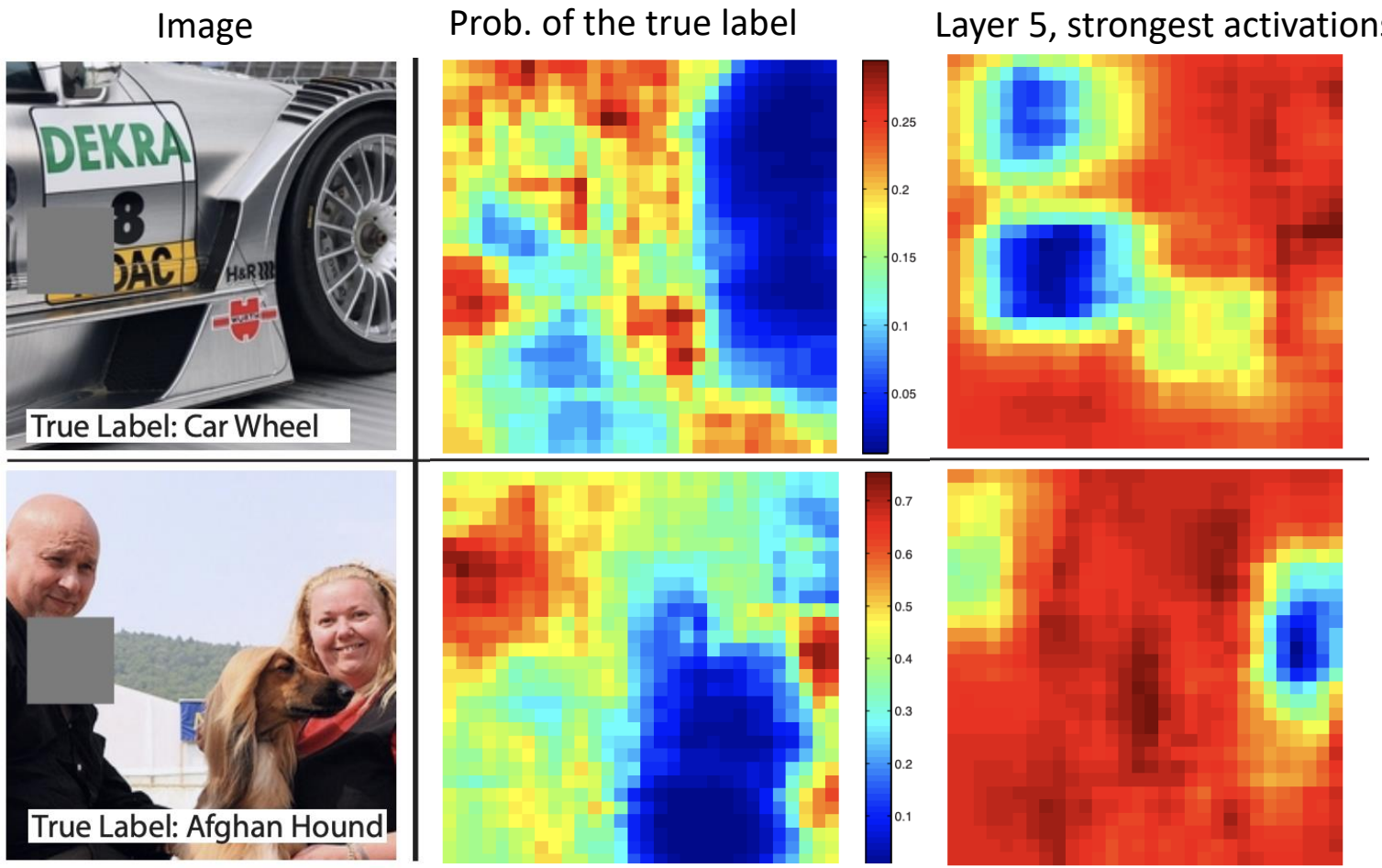
Layer 5, strongest activations (summed over spatial locations)



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

Saliency via occlusion

Which parts of the image are important for classification?



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

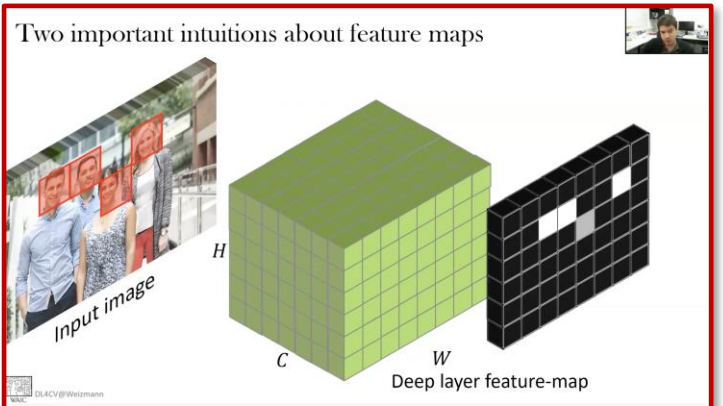
Class Activation Maps (CAM)

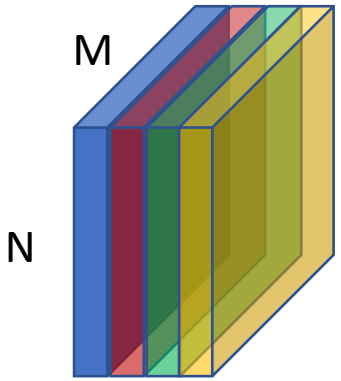
Replace last FC layer with Global Average Pool

Conv2D → ReLU → Conv2D → Max Pooling → Conv2D → ReLU → Conv2D → ReLU → Max Pooling → Conv2D → ReLU → Max

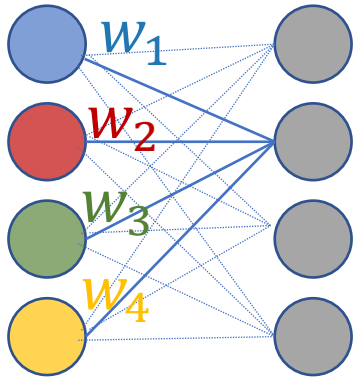
Fully connected layer: spatial information is removed

Linear → Relu → Linear → Softmax → Austrlian Terrier





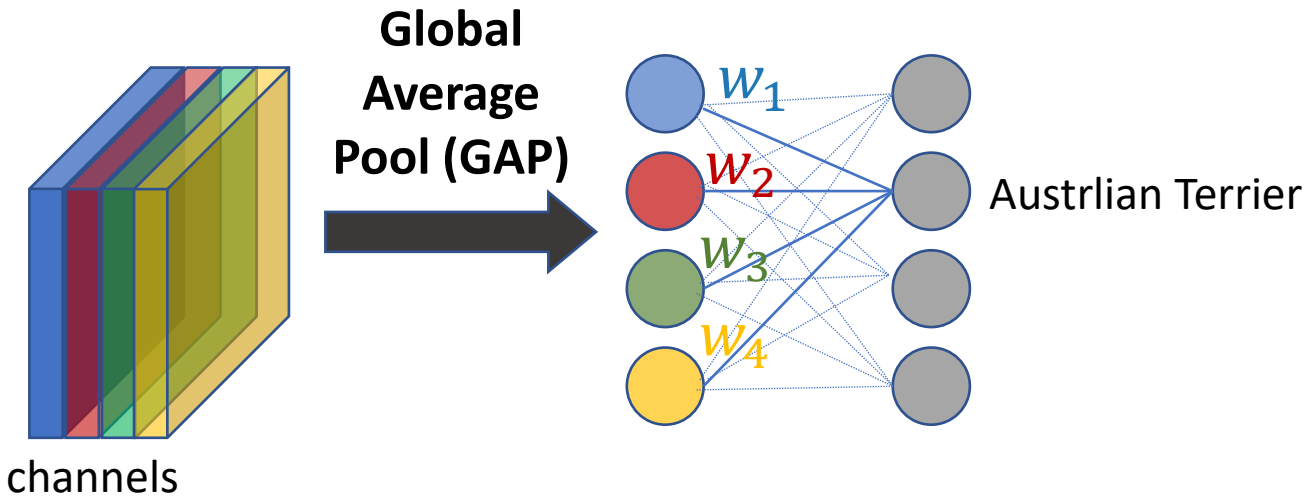
Global Average Pool (GAP)

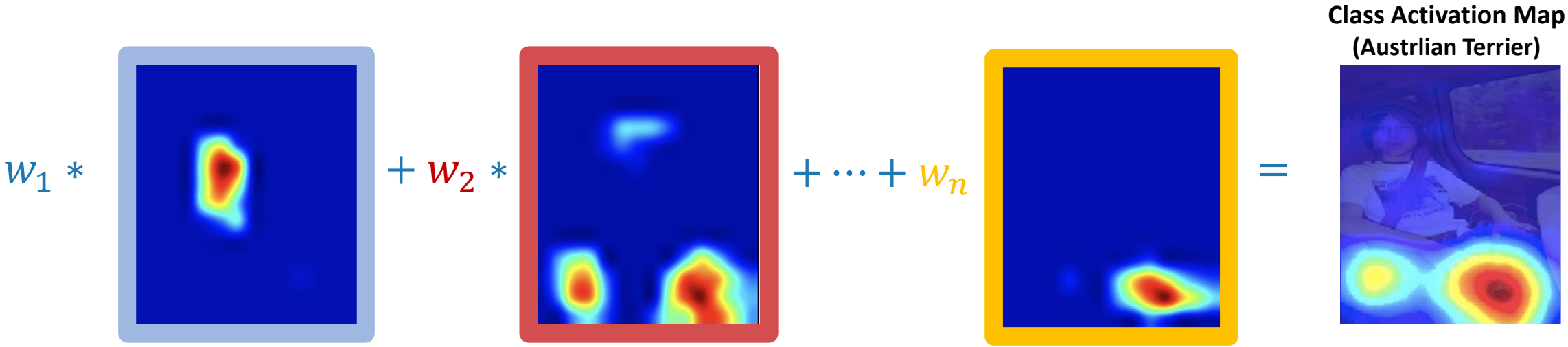


Austrlian Terrier

Zhou et. al, Learning Deep Features for Discriminative Localization, CVPR'16

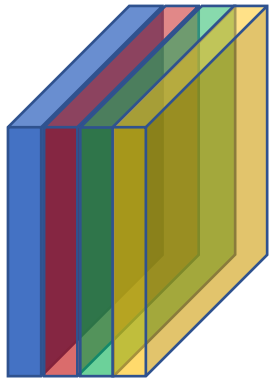
Class Activation Maps (CAM)





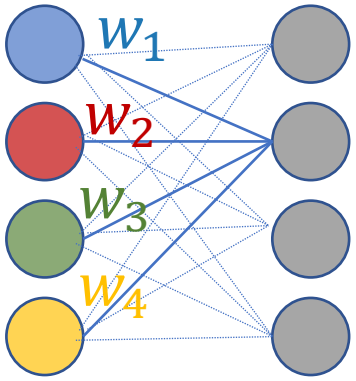
Zhou et. al, Learning Deep Features for Discriminative Localization, CVPR'16

Class Activation Maps (CAM)



channels

Global Average Pool (GAP)



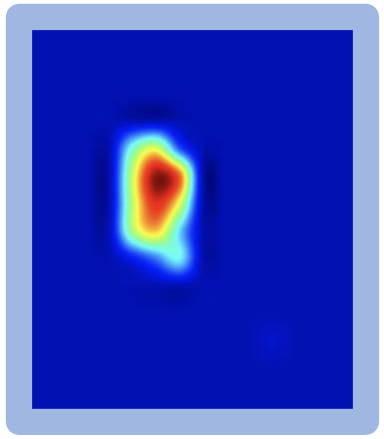
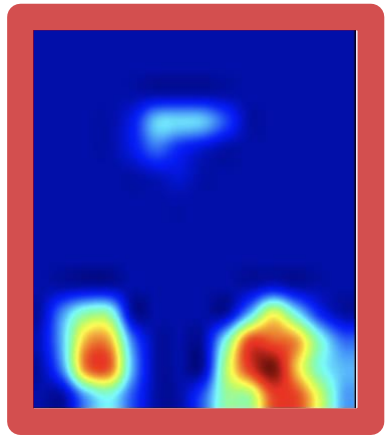
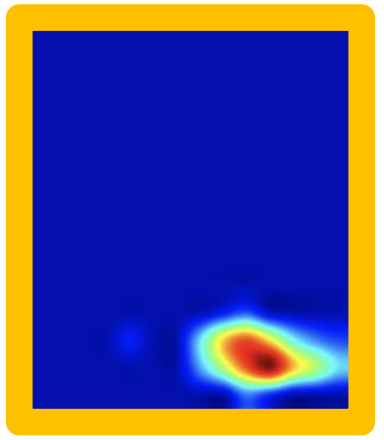
Austrlian Terrier

$f_k(x, y)$ - the kth feature map

$F_k = \sum_{x,y} f_k(x, y)$ - the kth averged pooled value

$S_c = \sum_k w_k F_k$

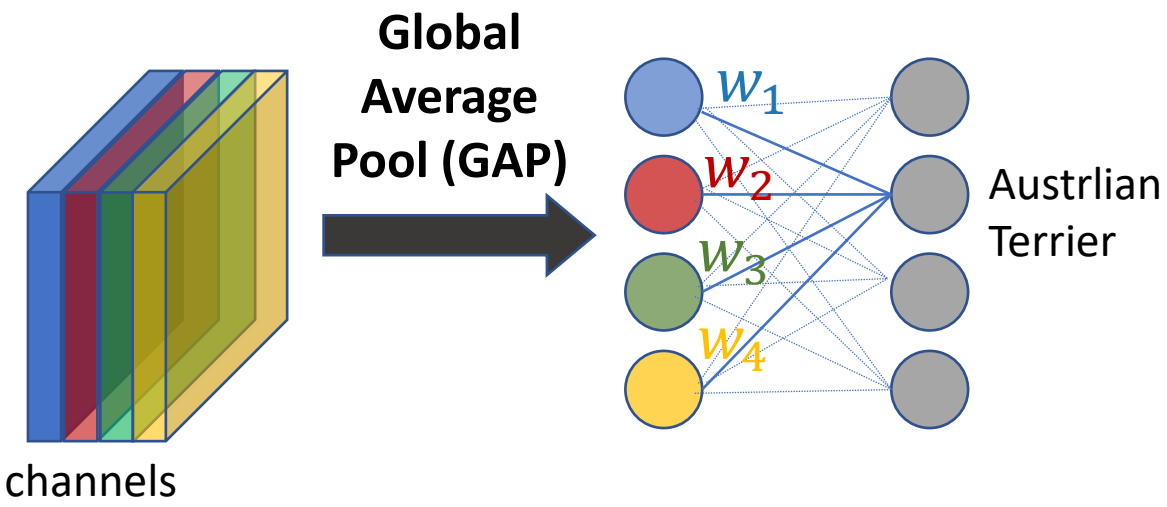
Score for class "c"

$w_1 *$  $+ w_2 *$  $+ \cdots + w_n$  $=$

Class Activation Map (Austrlian Terrier)

Zhou et. al, Learning Deep Features for Discriminative Localization, CVPR'16

Class Activation Maps (CAM)



$f_k(x, y)$ - the kth feature map

$$F_k = \sum_{x,y} f_k(x, y)$$ - the kth averged pooled value

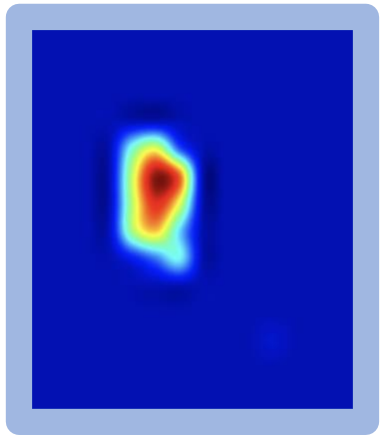
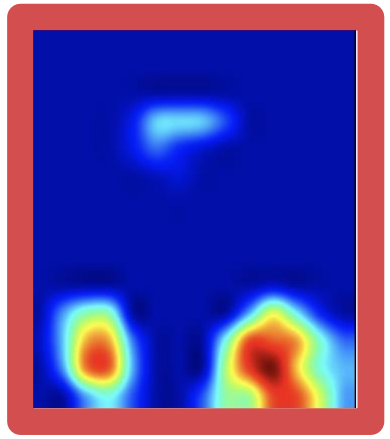
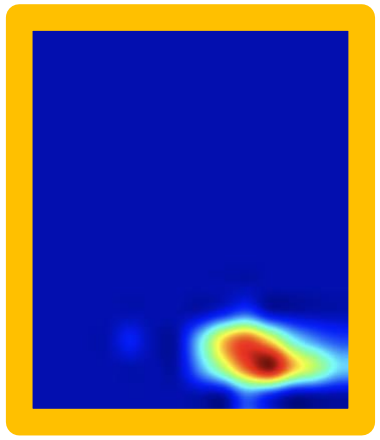
$$S_c = \sum_k w_k F_k = \sum_k w_k \sum_{x,y} f_k(x, y)$$

Score for class "c"

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

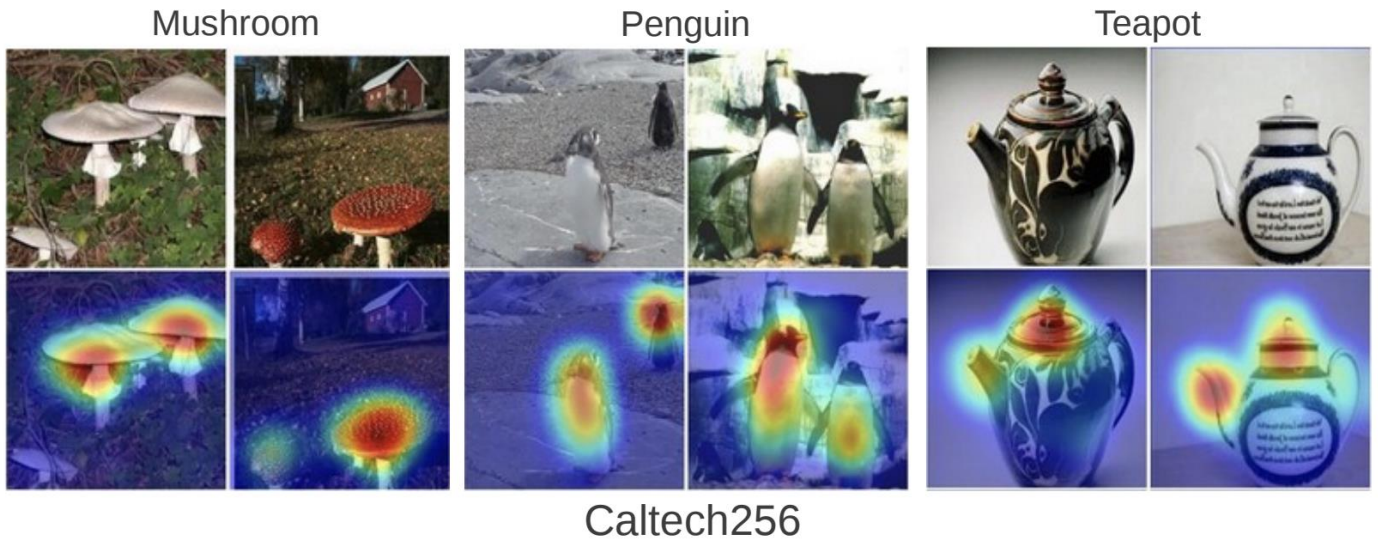
$$M^c_{\text{CAM}}$$

Class Activation Map (Austrlian Terrier)

$w_1 *$  $+ w_2 *$  $+ \cdots + w_n$  $=$ 

Zhou et. al, Learning Deep Features for Discriminative Localization, CVPR'16

Class Activation Maps (CAM)



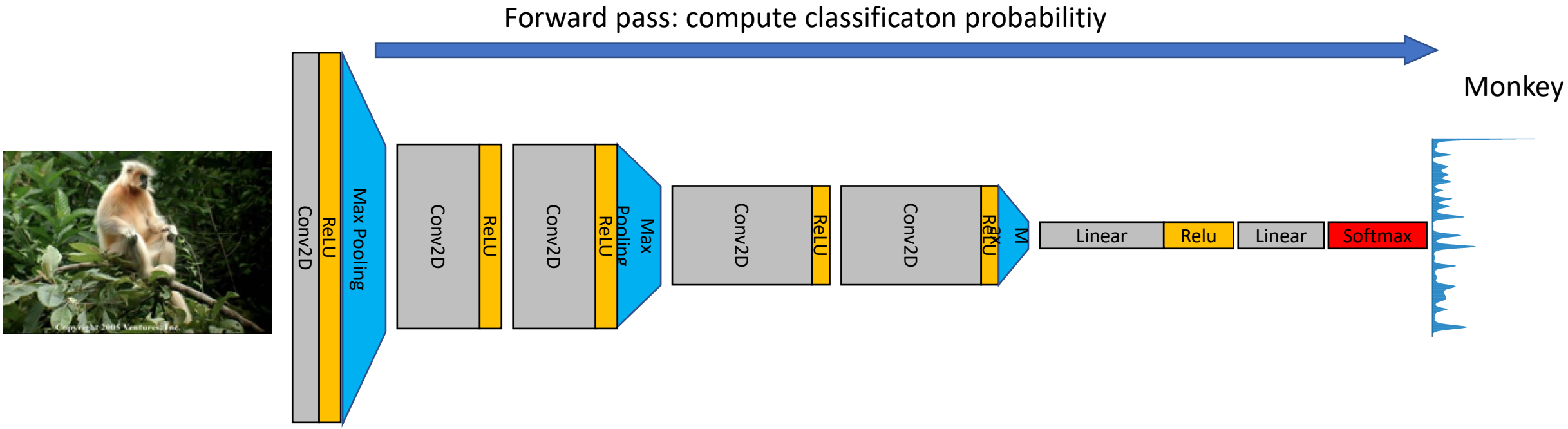
But…

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

Zhou et. al, Learning Deep Features for Discriminative Localization, CVPR'16

Saliency via Backpropogation

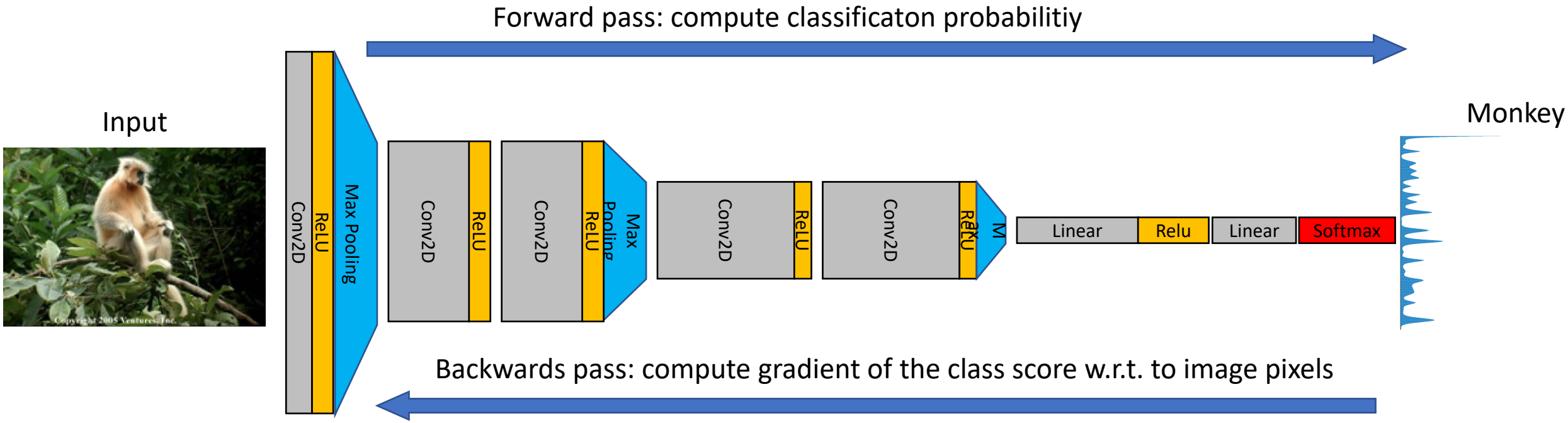
Which parts of the image are important for classification?



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

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

Probablity of the true class

Image pixel

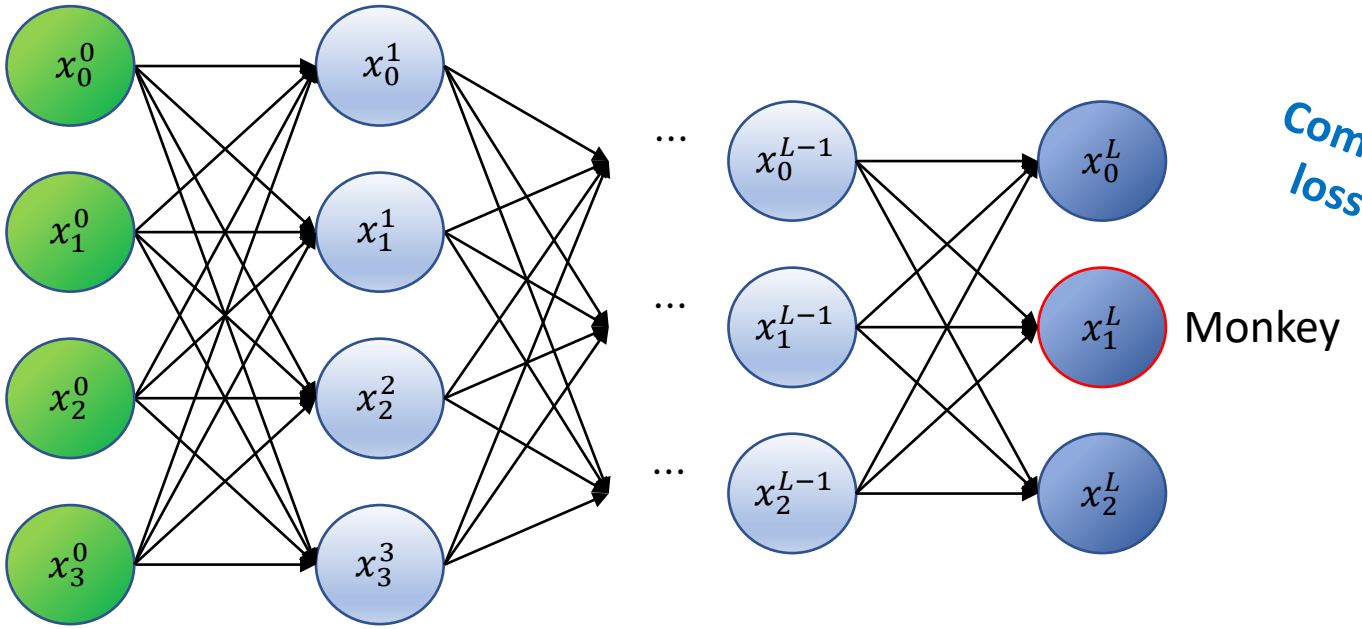


Image pixels

Class output

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

Saliency via Backpropogation

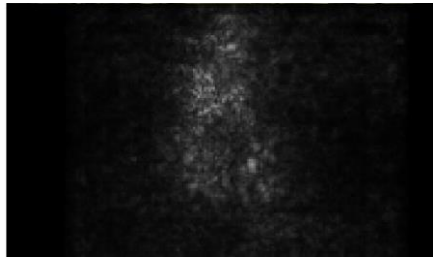
Which parts of the image are important for classification?
Compute the gradients of the target class w.r.t. the input image

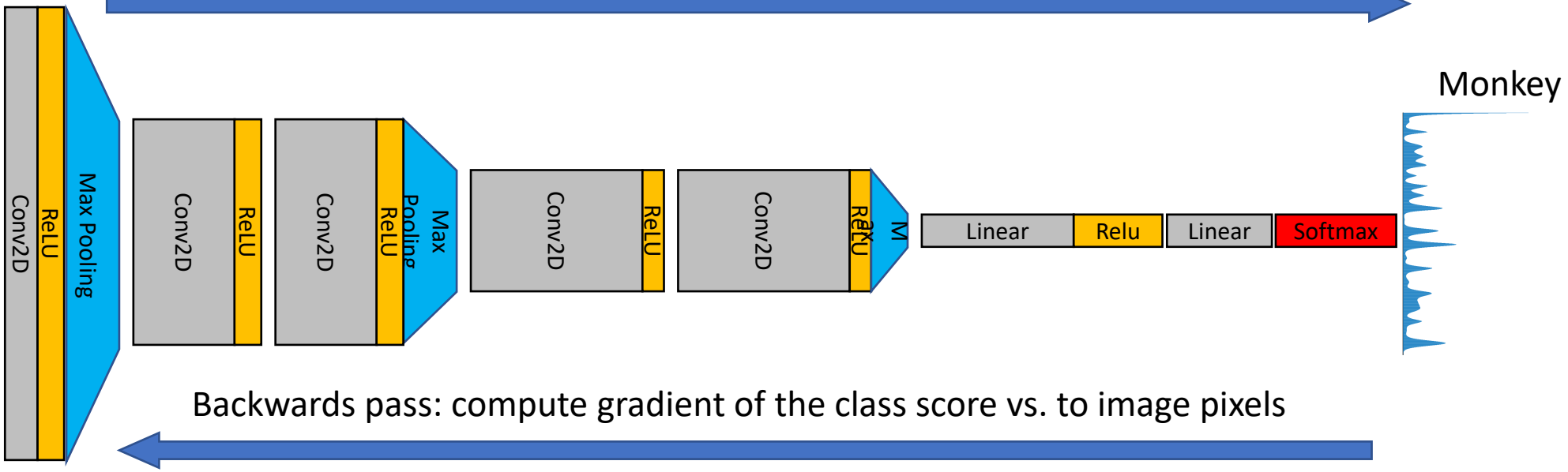
$$\frac{\partial x_1^L}{\partial x_i^0}$$

Forward pass: compute classificaton probabilitiy

Input

Saliency map





Backwards pass: compute gradient of the class score vs. to image pixels

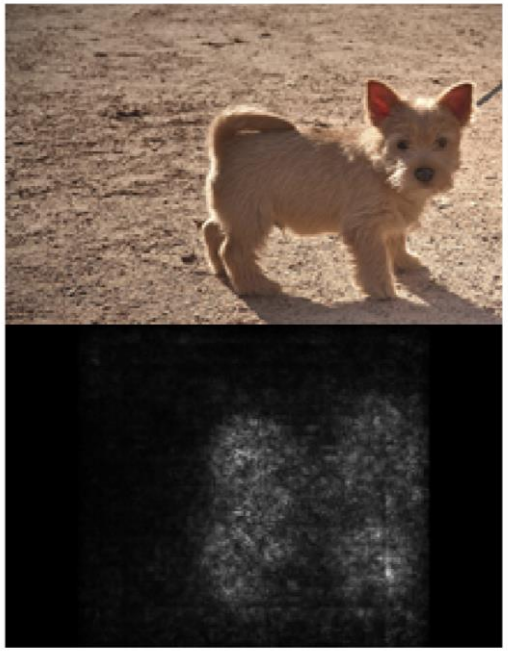
Simonyan, Vedaldi, and Zisserman, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”, ICLR Workshop 2014.

Saliency via Backpropogation

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



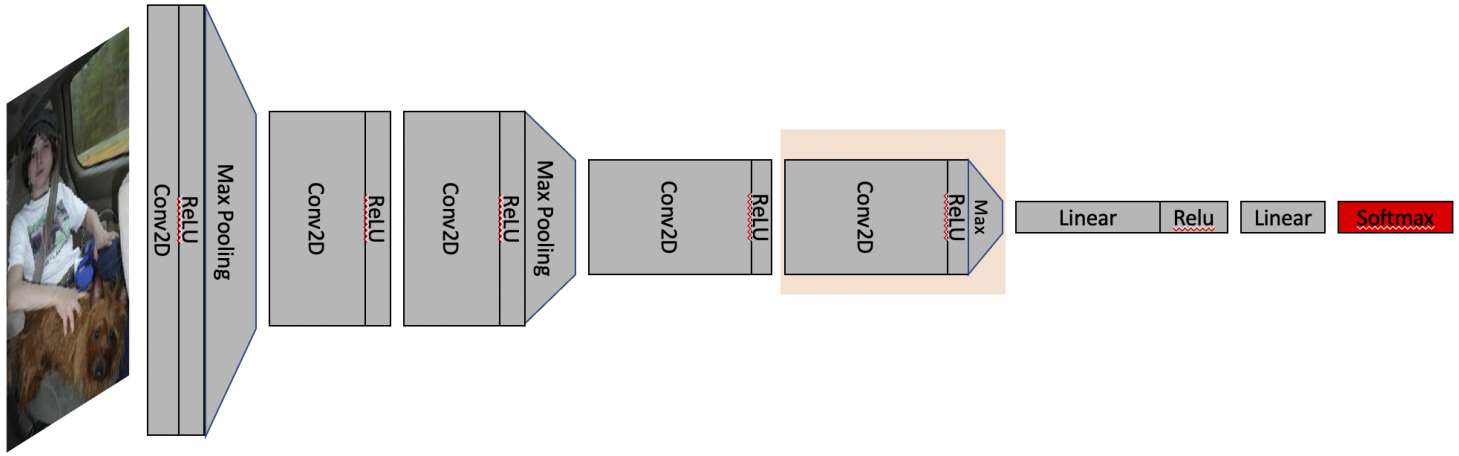
- Can be applied to existing models
- Highlight fine-grained details
- Not discriminative

Class Activation Maps

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

Gradient-Weighted Class Activation Mapping (Grad-CAM)

Pixel-space gradient + CAM



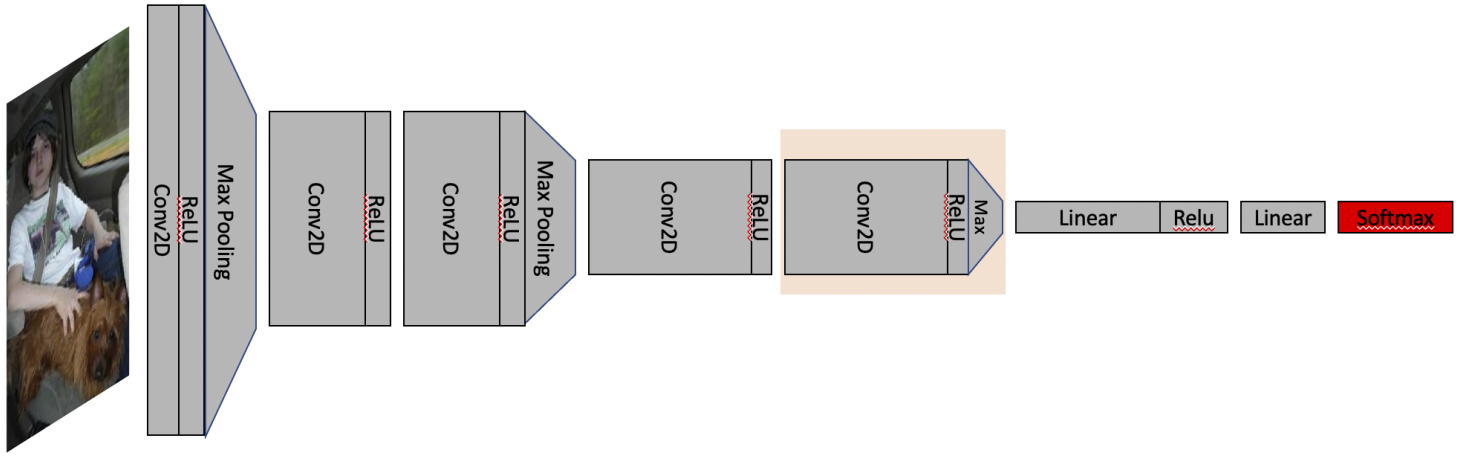
1. Compute the gradient of the target class score S_c w.r.t. the feature map $f_k(x, y) \in \mathbb{R}^{u \, x \, v}$:

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

Selvaraju et al, “Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization”, CVPR 2017

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:

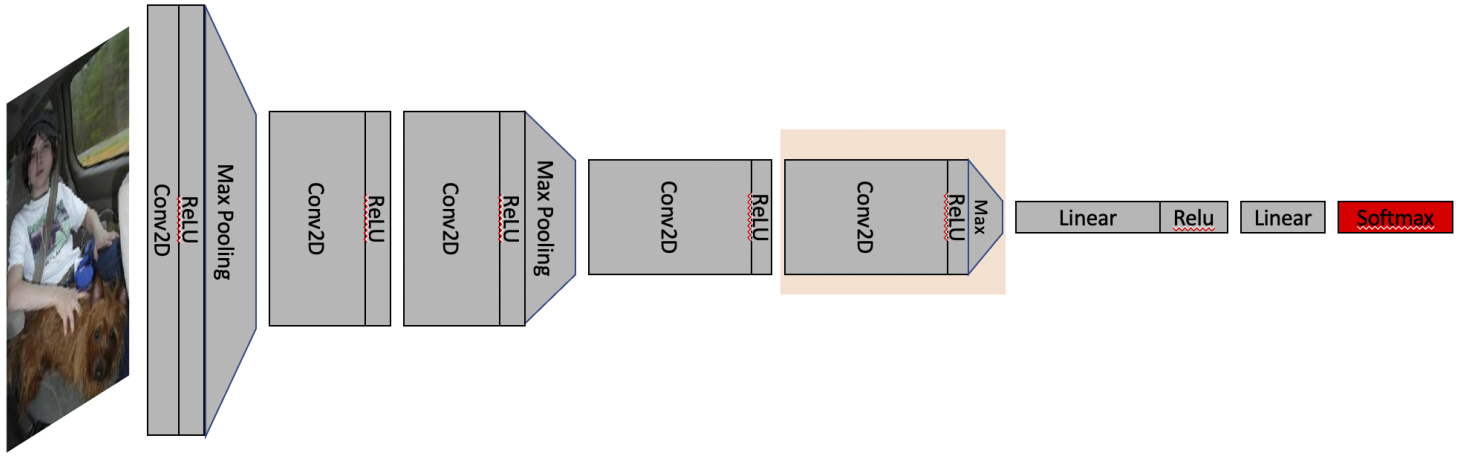
'importance' weights of feature map k

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

Selvaraju et al, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization", CVPR 2017

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

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

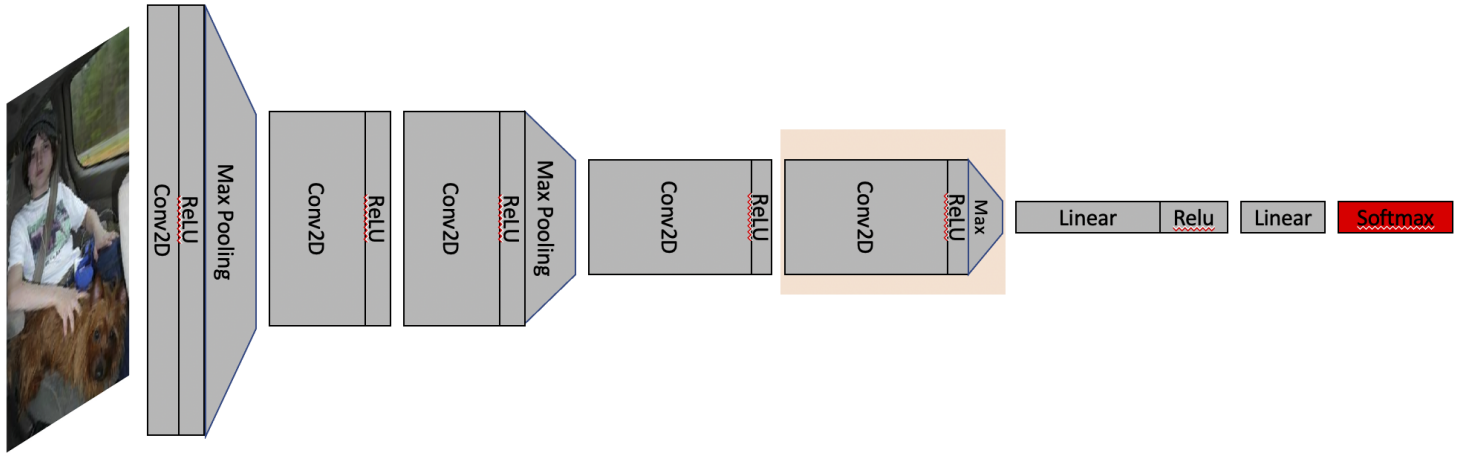
3. Compute a linear combination of the activations + ReLU:

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

Selvaraju et al, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization", CVPR 2017

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

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

$w_k = \alpha_k$

See proof in the paper

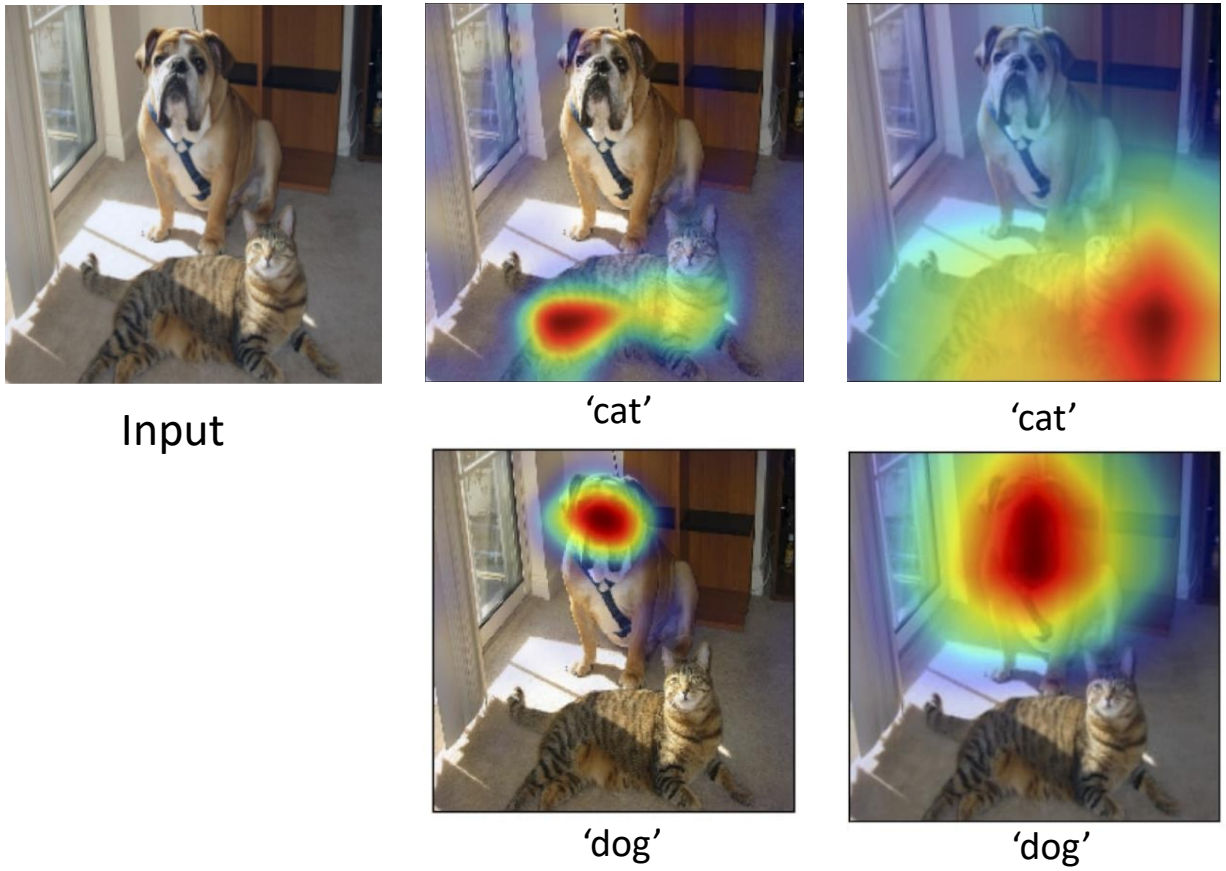
3. Compute a linear combination of the activations + ReLU:

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

Selvaraju et al, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization", CVPR 2017

Gradient-Weighted Class Activation Mapping (Grad-CAM)

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



Selvaraju et al, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization", CVPR 2017

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:

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

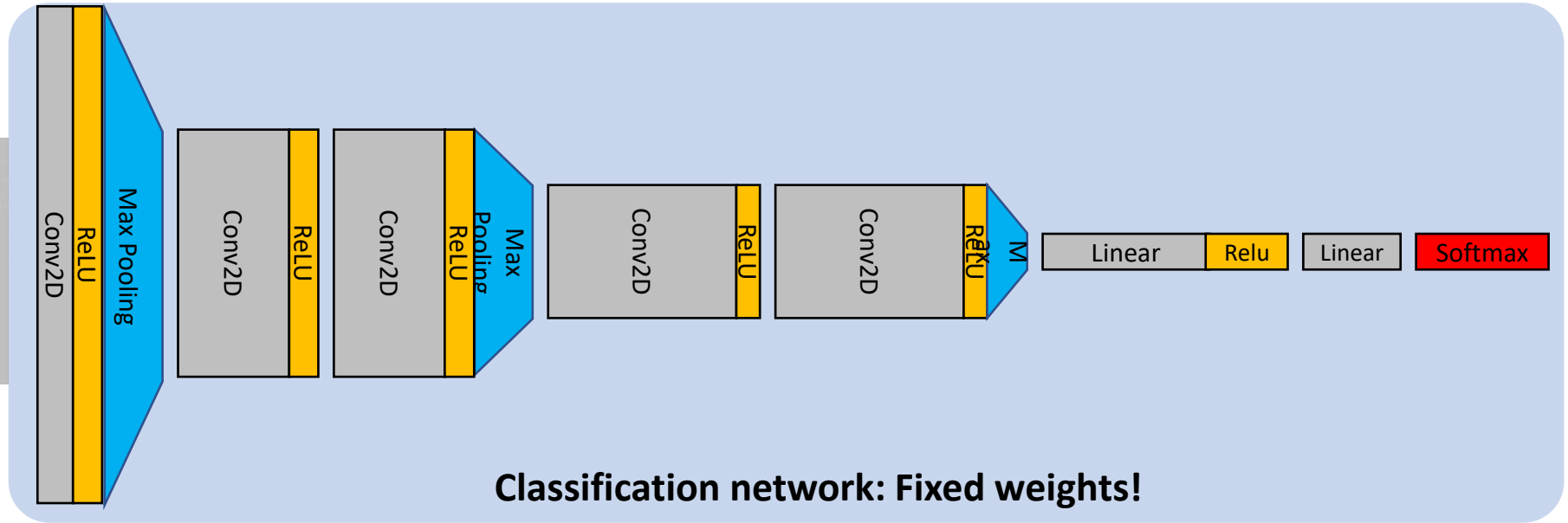
Neuron activation

Natural image prior

No image prior

The variables of optimization are the image values

Unknown Image *I*



Classification network: Fixed weights!

Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.
https://distill.pub/2017/feature-visualization/

Feature Visualization via Image Synthesis

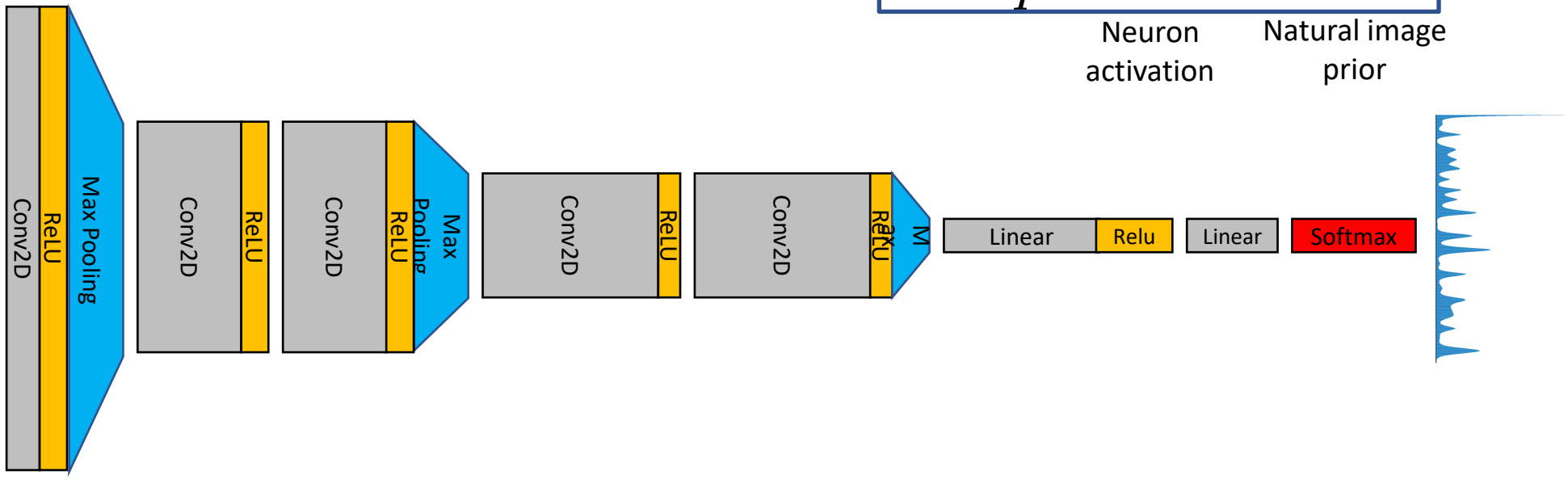
Solve for an image I that maximizes the objective:

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

Neuron activation

Natural image prior

Unknown Image I



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

Rpeat:

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

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

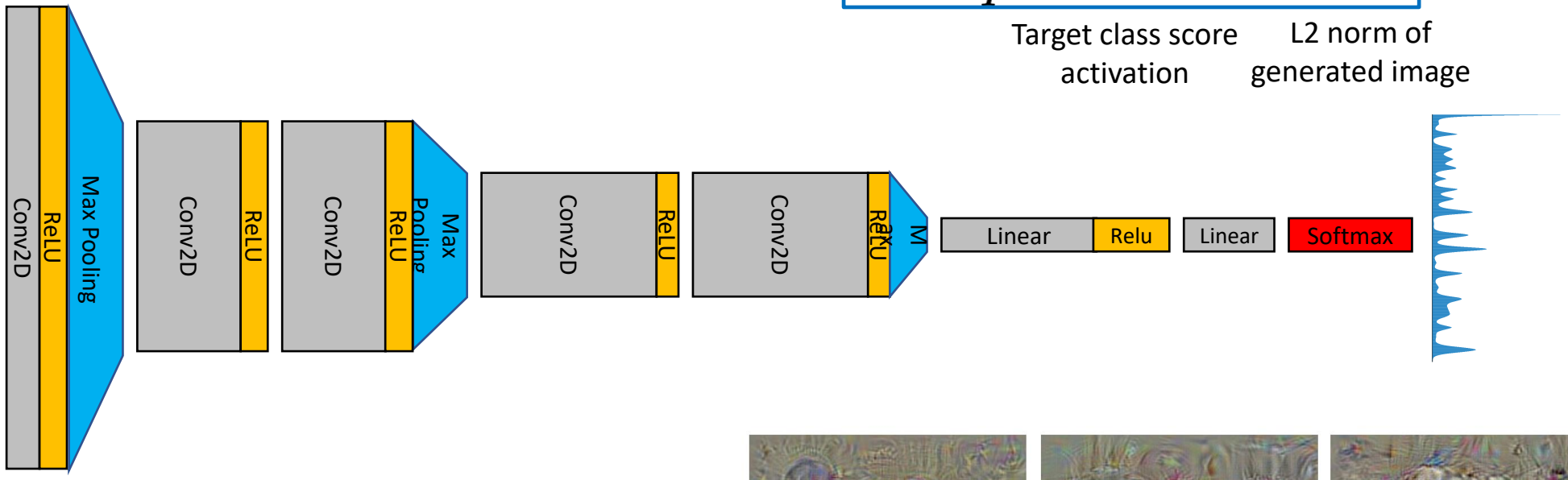
Feature Visualization via Image Synthesis

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

Target class score activation

L2 norm of generated image

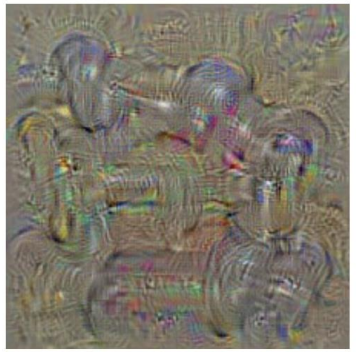
Unknown Image I



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

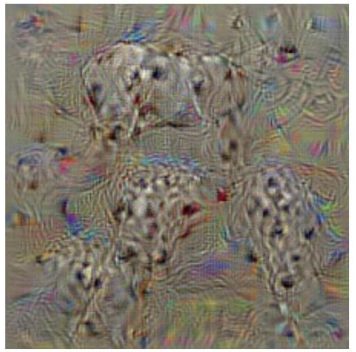
Rpeat:

1. **Feed into the nework to compute $S_j^l(I)$**
2. **Compute gradients: $\partial S_j^l / \partial I$ via backproption**
3. **Apply a small update to the image**



dumbbell

cup



dalmatian

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

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

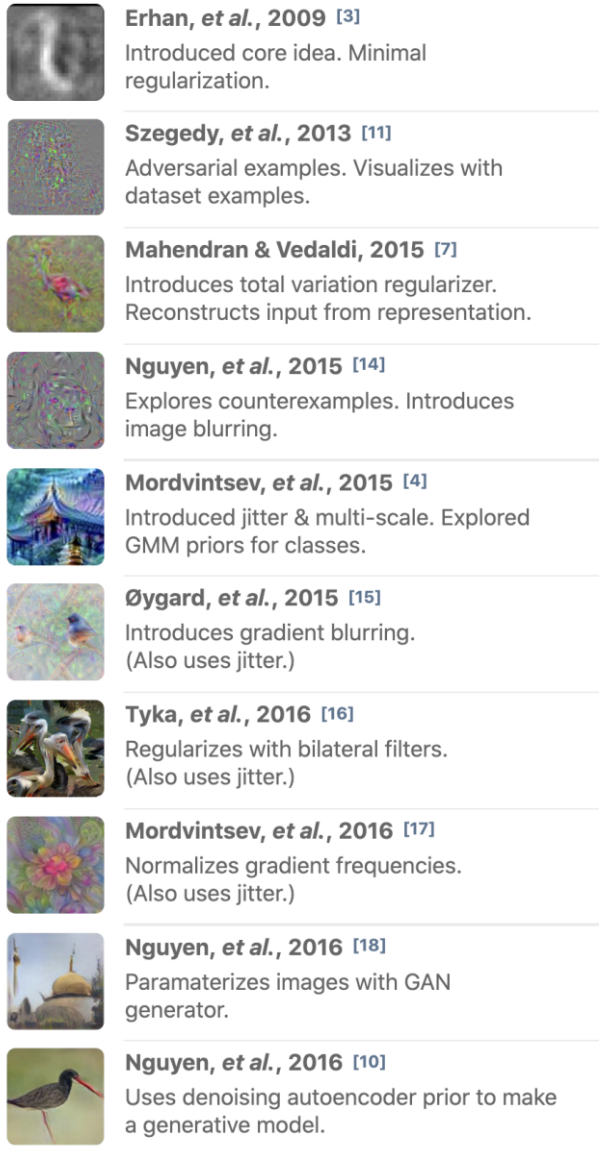
Can we design a more “fancy” natural image prior?

No regularization

Learned priors (e.g., GANs)

Image naturalness

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



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.

Øygard, *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.)

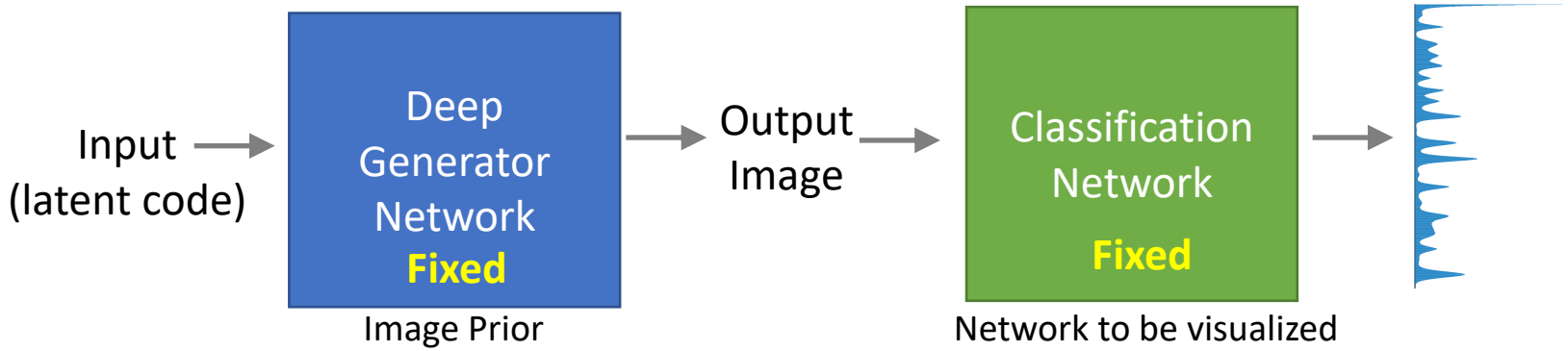
Nguyen, *et al.*, 2016 [18]
Paramaterizes images with GAN generator.

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

Feature Visualization via Image Synthesis

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

Neuron activation — Natural image prior

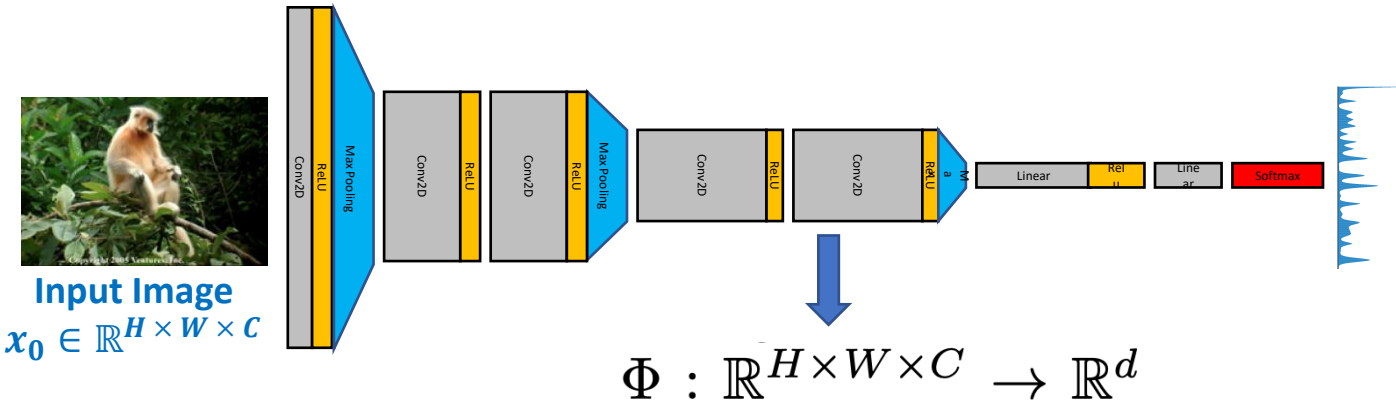


Nguyen et al, "Synthesizing the preferred inputs for neurons in neural networks via deep generator networks," NIPS 2016

Feature Inversion

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

Output: new "natural" image that matches the target feature



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

Feature data term

Natural image prior

Euclidian distance between the features:

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

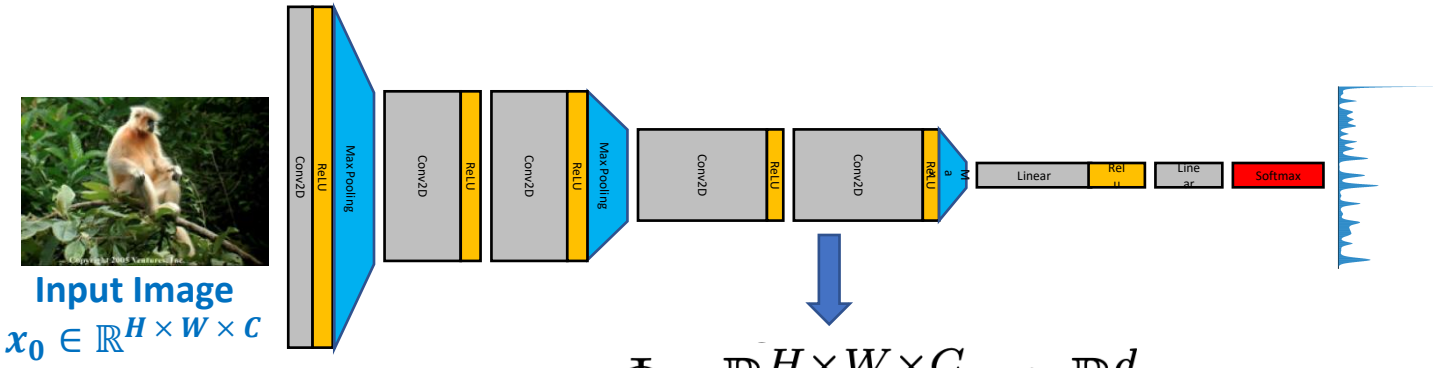
Total Variation (TV) regularizer (encourages spatial smoothness)

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

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

Feature Inversion

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



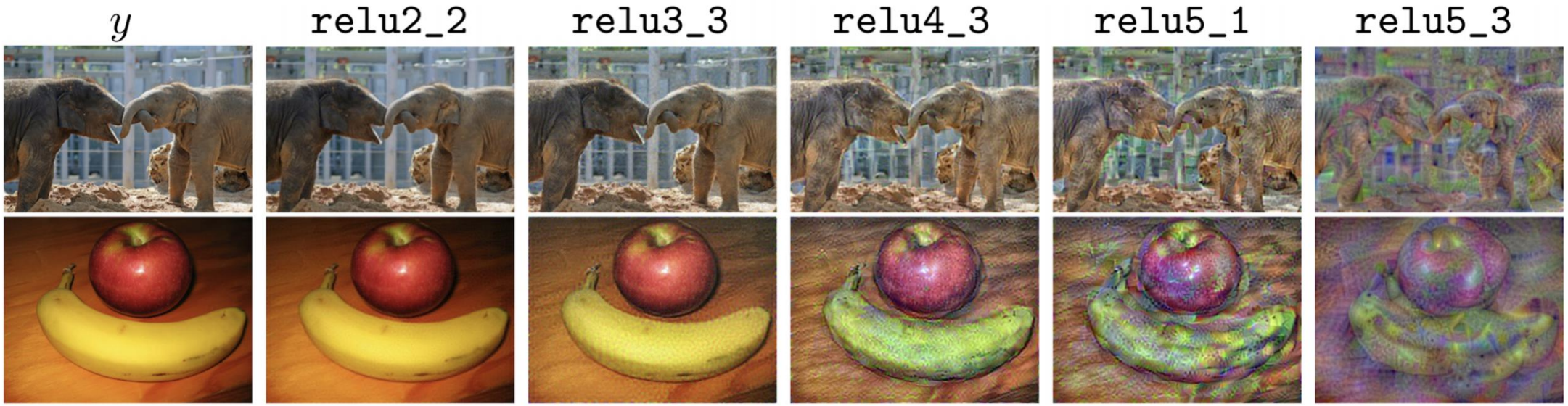
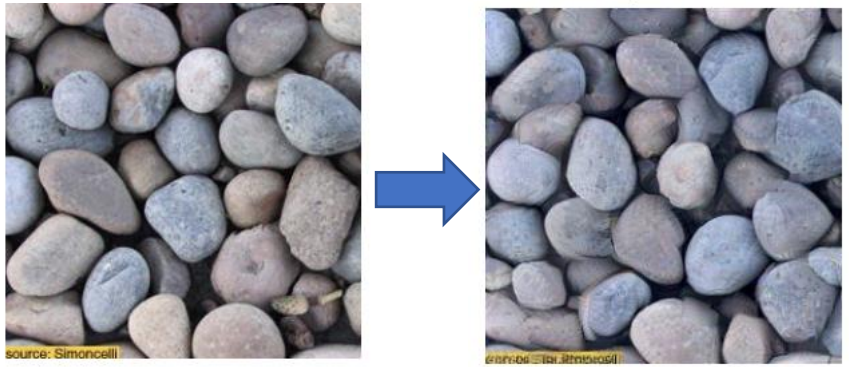


Figure from: “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016

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)

Gatys et. al, Texture Synthesis Using Convolutional Neural Networks, 2015
Gatys, Ecker, and Bethge, “Image style transfer using convolutional neural networks”, CVPR 2016

Neural Texture Synthesis

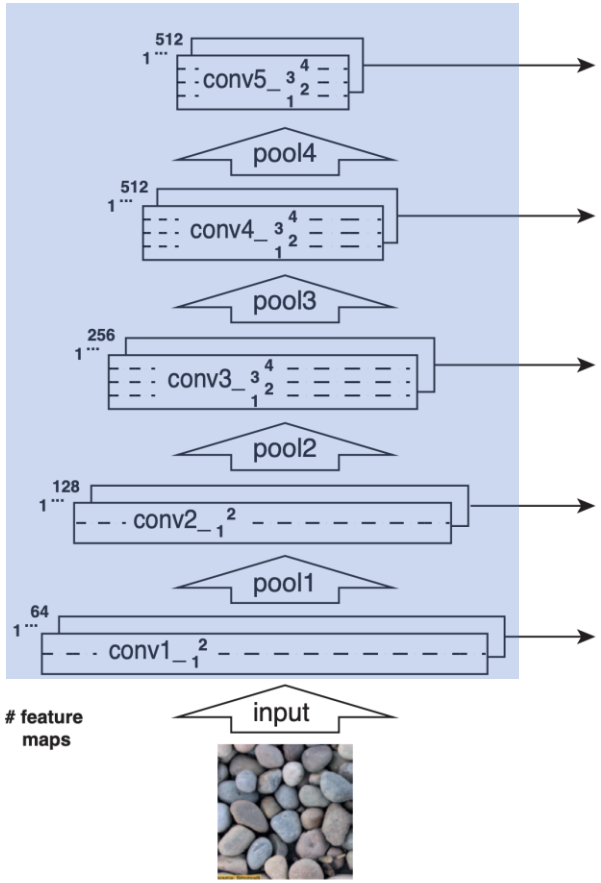
Loss that captures texture similarity

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

Input: texture image, pre-trained CNN

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

Fixed CNN



Original image

Output image

Gatys et. al, Texture Synthesis Using Convolutional Neural Networks, 2015

Neural Texture Synthesis

Loss that captures texture similarity

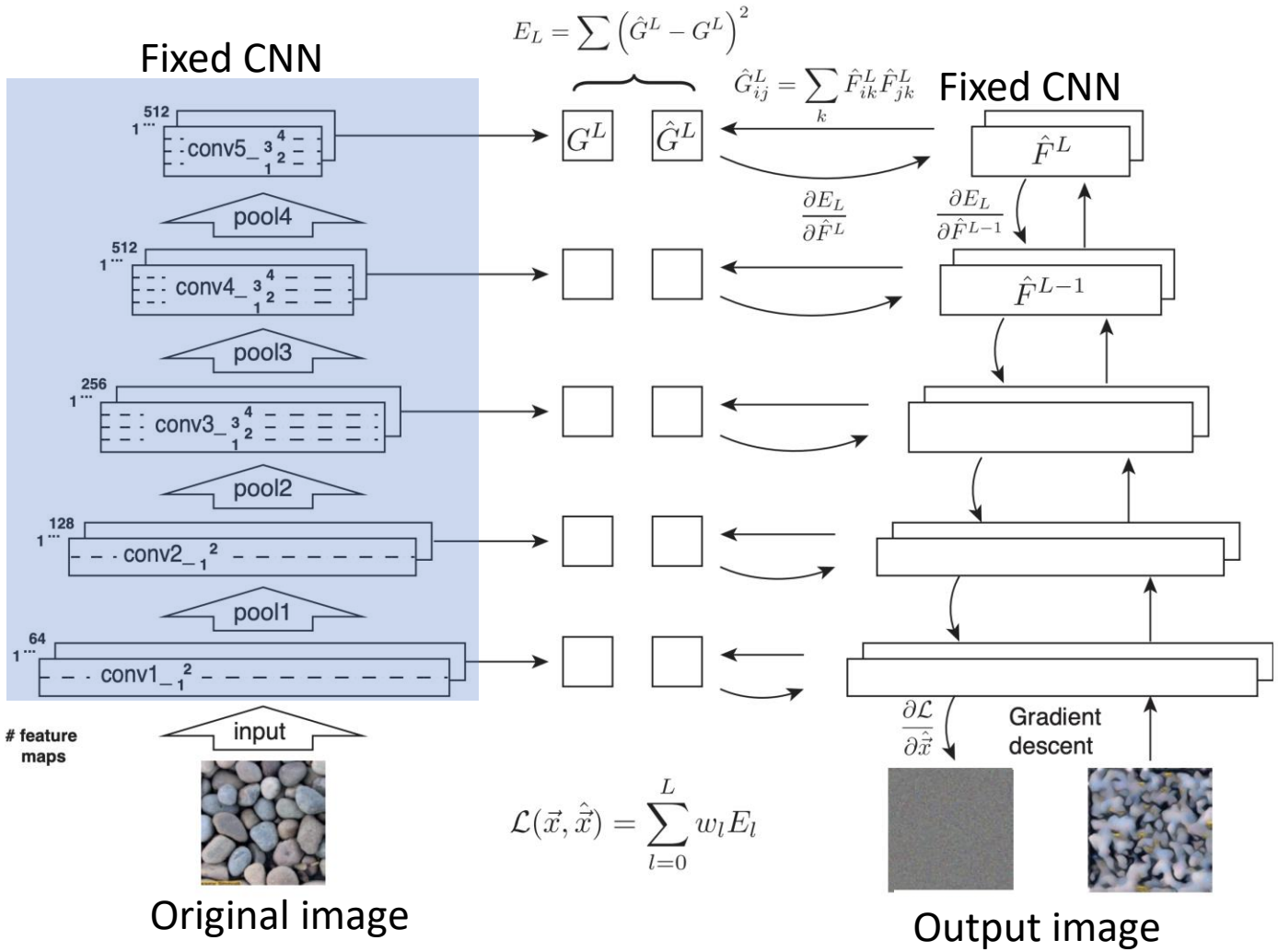
$$I^* = \arg\min_I \sum_l w_l E_l$$

Input: texture image, pre-trained CNN

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

Rpeat:

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



Gatys et. al, Texture Synthesis Using Convolutional Neural Networks, 2015

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

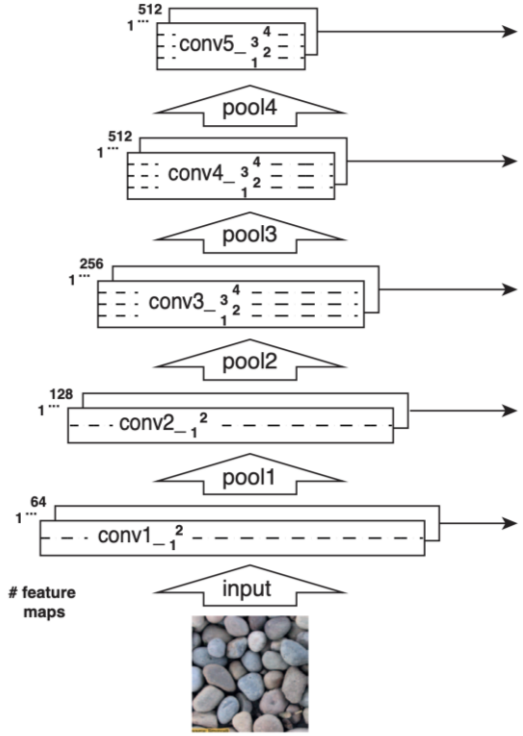
Gatys et. al, Texture Synthesis Using Convolutional Neural Networks, 2015

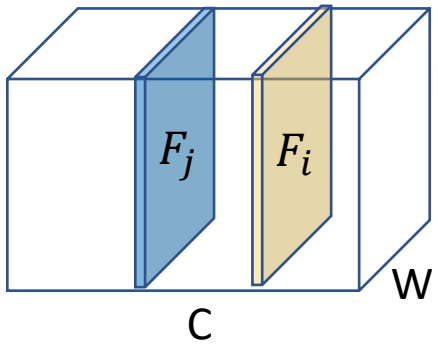
Neural Texture Synthesis: Gram Matrix

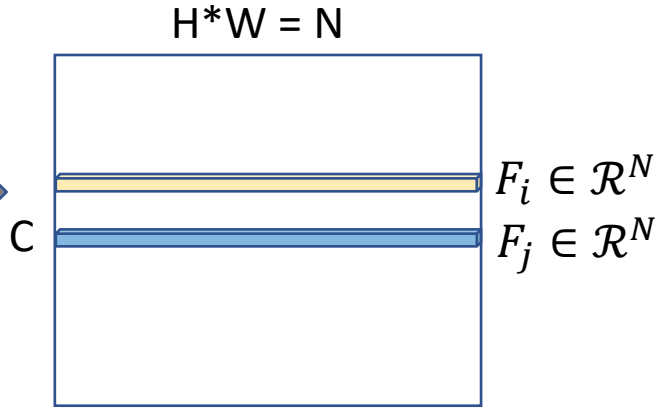
Each layer is a C x H x W tensor of features;
We have C feature maps (each of resolution H x W)

Loss that captures texture similarity

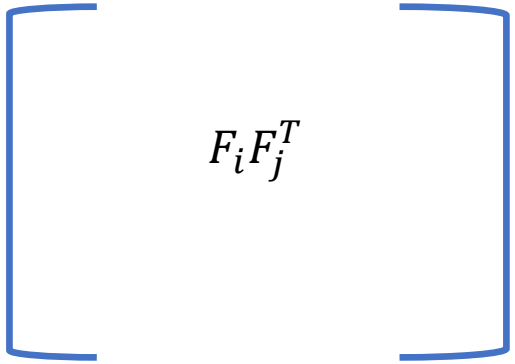
$$I^* = \arg\min_I \sum_l w_l E_l$$







Each element is the correlation between two features



C x C
Gram Matrix

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

Gatys et. al, Texture Synthesis Using Convolutional Neural Networks, 2015

Neural Texture Synthesis

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

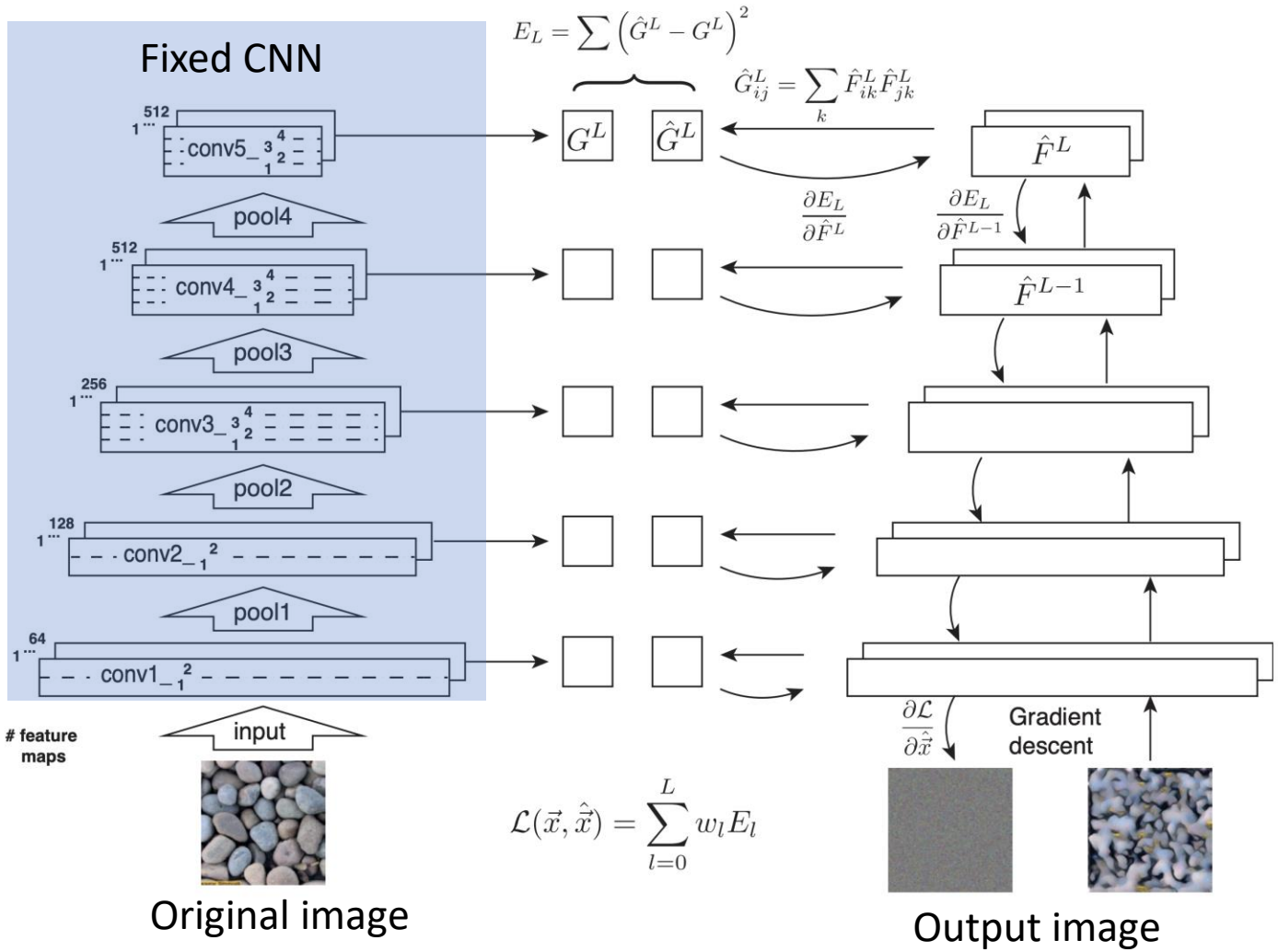
Input: texture image, pre-trained CNN

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

Rpeat:

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

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



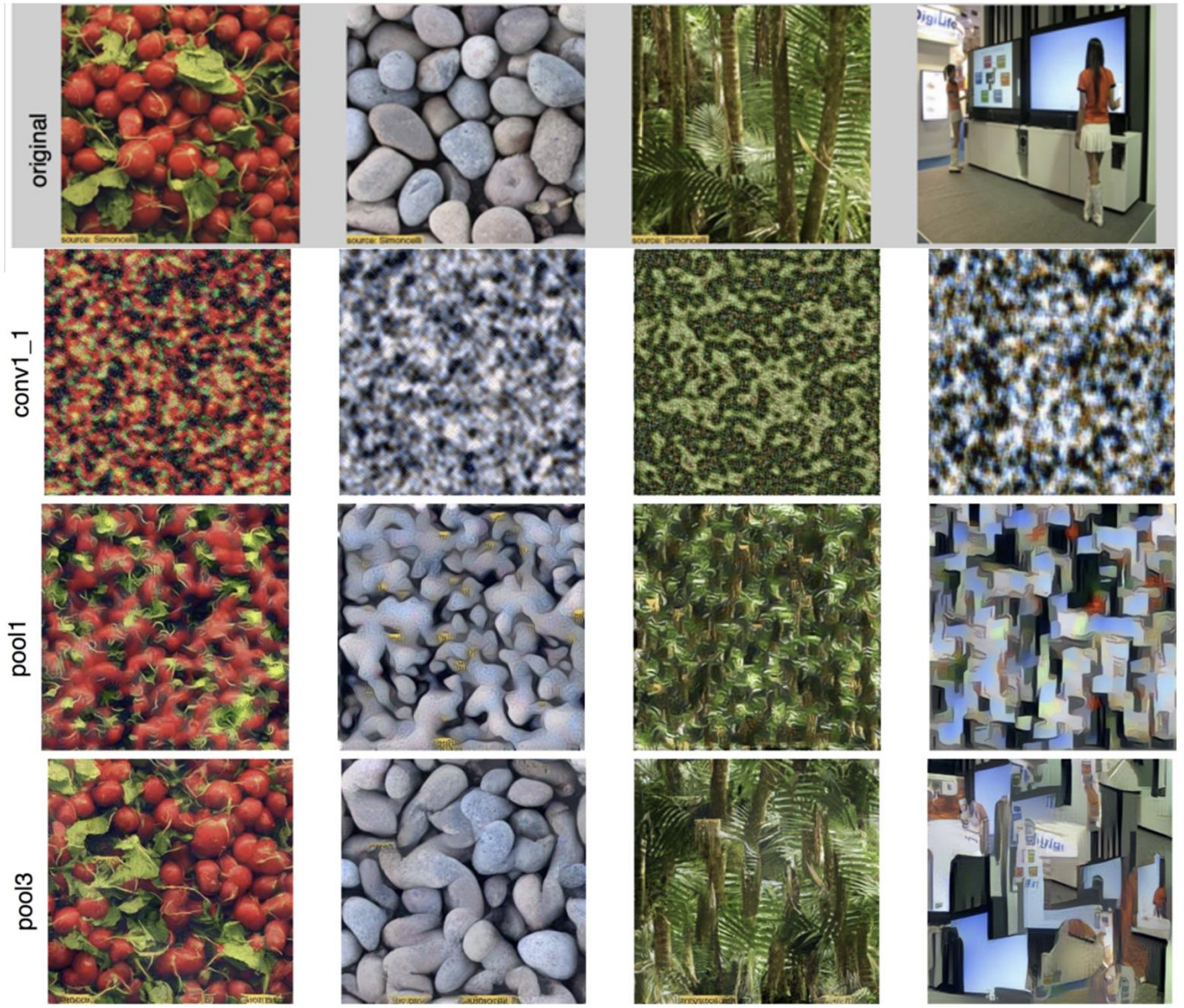
Original image

Output image

Gatys et. al, Texture Synthesis Using Convolutional Neural Networks, 2015

Neural Texture Synthesis

Patterns/structures size increasesas we use deeper layers



Gatys et. al, Texture Synthesis Using Convolutional Neural Networks, 2015

Neural Style Transfer

Content Image A

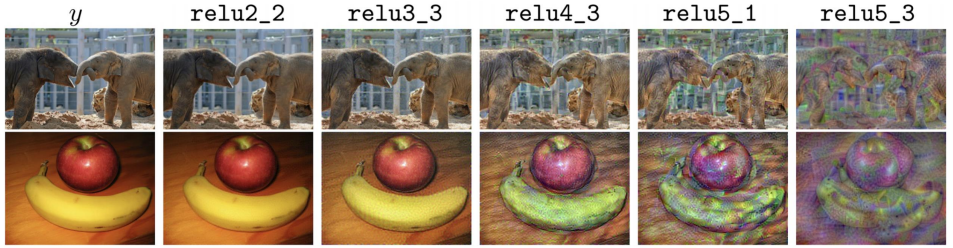
Style/Texture Image B

Stylized image
(content A + Style B)

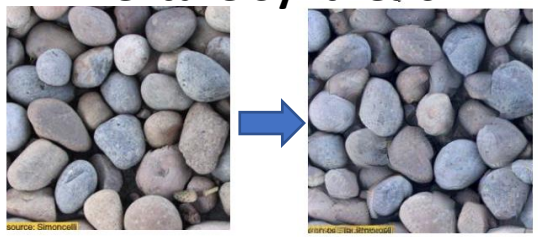
Match features of the conent image

Match Gram matrices of the style image

Feature Inverstion (reconstruction)



Texture Synthesis



Original
Texture image

Output
New texture image

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

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

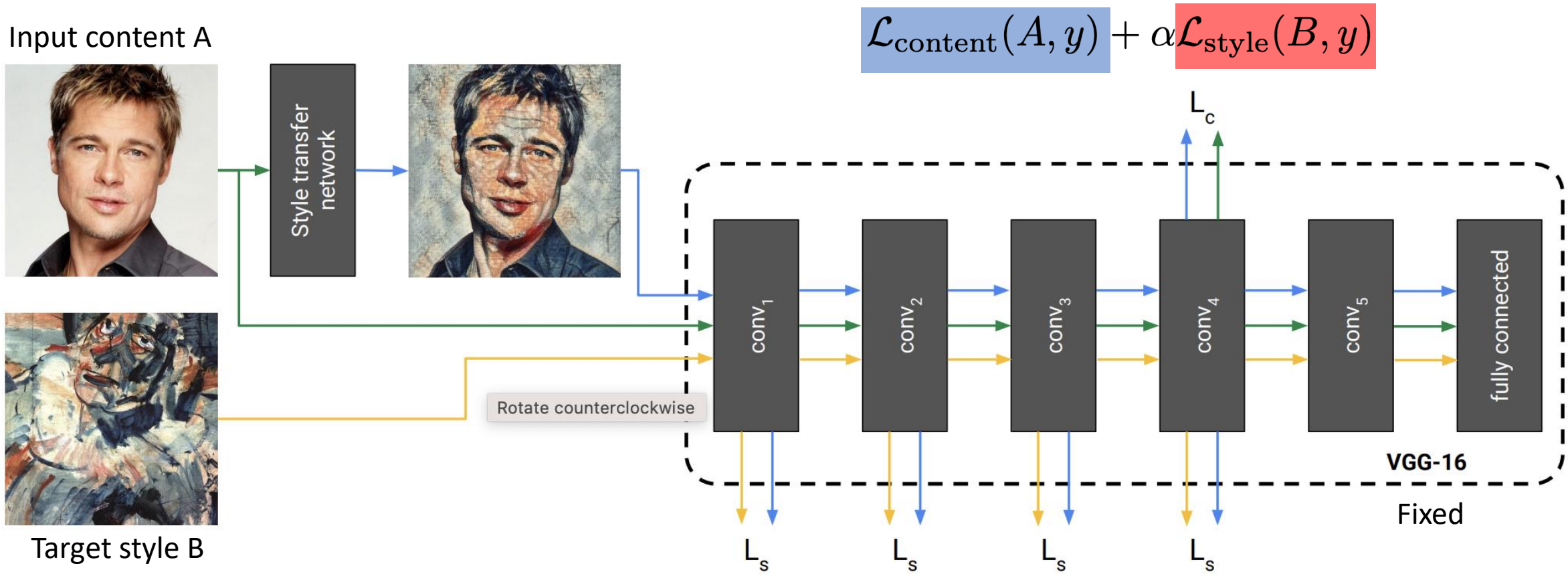
Neural Style Transfer

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

Figure from: https://github.com/jcjohnson/neural-style
Gatys, Ecker, and Bethge, “Image style transfer using convolutional neural networks”, CVPR 2016

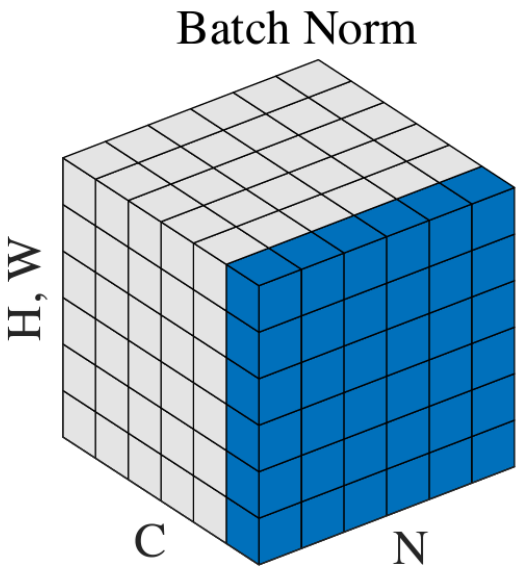
Feed Forward Style Transfer (per style)

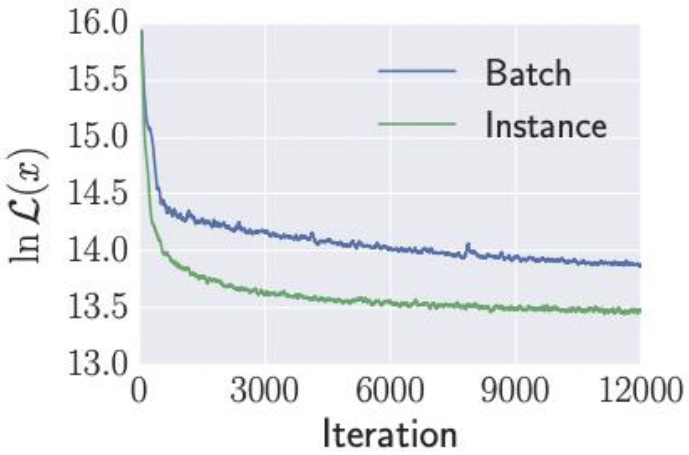
Train a feedforward network for each style, use **perceptual loss** for training

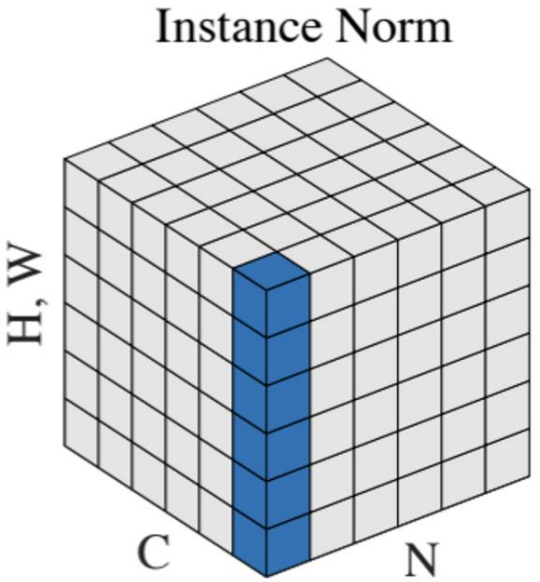


Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016

Normalization Layers







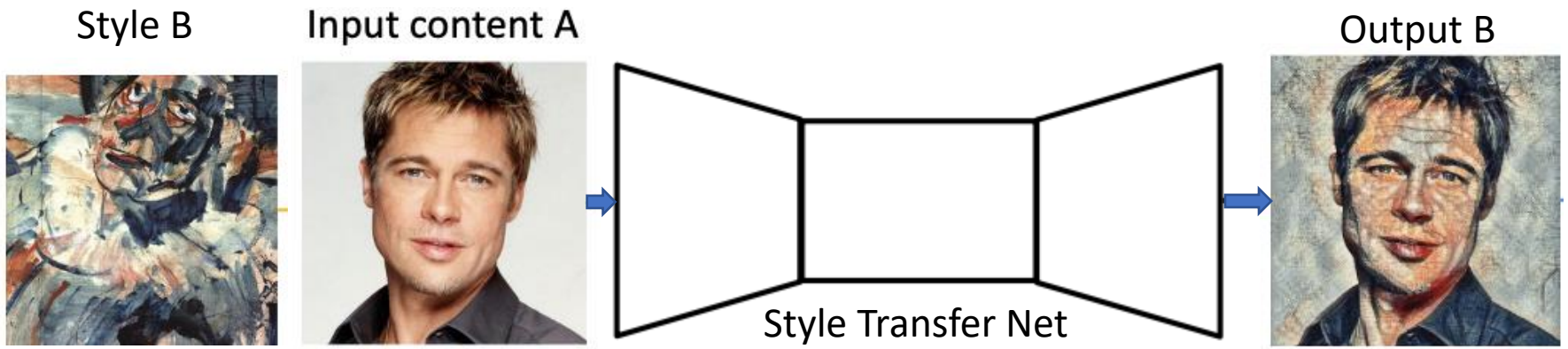
$\{\mu, \sigma\}$ – mean and std over spatial dimentions

$$y = \frac{x - \mu}{\sigma}$$

Ulyanov et. al, "Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis", CVPR'17

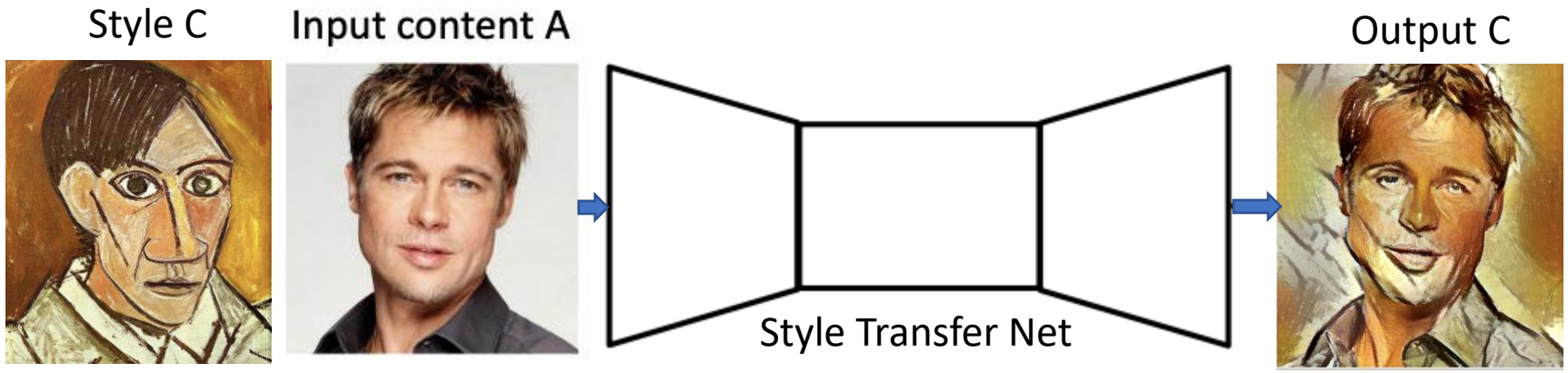
Feed Forward Style Transfer (many styles)

Train a feedforward network for **multiple** styles



Feed Forward Style Transfer (many styles)

Train a feedforward network for **multiple** styles



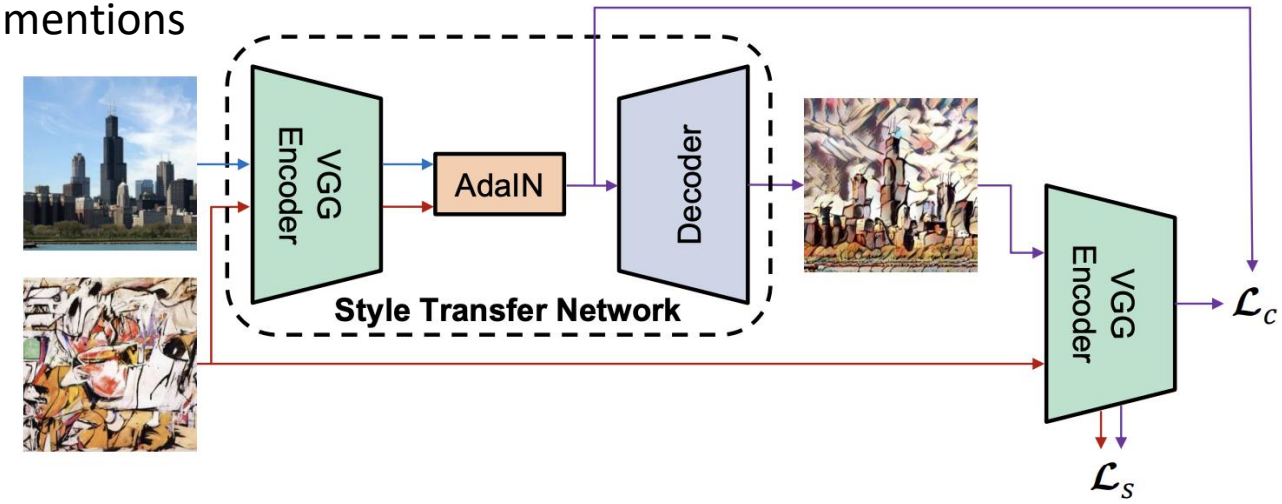
Feed Forward Style Transfer (many styles)

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

Content features are normalized, then scaled + shifted according to the style statistics:

$$\text{AdaIN}(x, y) = \sigma(y)\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \mu(y)$$

x – Content feature map C x H x W
y – Style feature map C x H x W
$\{\mu, \sigma\}$ – mean and std over spatial dimentions



Dumoulin et. al, "A Learned Representation for Artistic Style ", ICLR 2017
Huang & Belongie, "Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization", CVPR 2017

Feed Forward Style Transfer (many styles)

Train a feedforward network for **arbitrary** styles, using **AdaIN:**

Next class/tutorial:
“Advanced Pytorch”

Rafail Fridman

Niv Haim

“Adversarial Examples”