Lecture 14:

Understanding and Visualizing Convolutional Networks

Administrivia

Project milestone due 10/31 (see the piazza note)

Administrivia

Thursday (10/31) there will be no class. Instead attend the MLFL seminar:

who: Boqing Gong (http://boqinggong.info/)
when: Thursday, 10/31/2024, 12pm-1pm
where: CS 150/151 (pizzas available), Zoom
food: Pizza and drinks

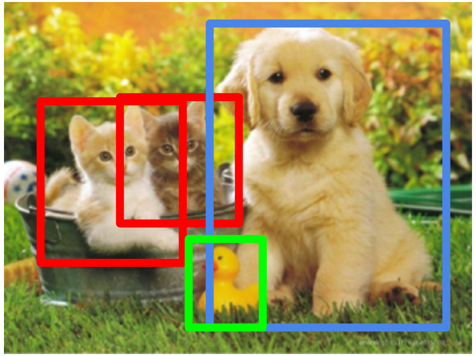
Title: From Domain Adaptation to VideoPrism: A Decade-Long Quest for Out-of-Domain Visual Generalization

Abstract:
This talk explores the challenges of out-of-domain (OOD) generalization in computer vision, encompassing tasks like domain adaptation, webly-supervised learning, and long-tailed recognition. I will review some principles and techniques underlying the seemingly diverse tasks and then connect them to the recent development of generalist vision systems, showcasing VideoPrism --- a state-of-the-art generalist video encoding model --- and ongoing research into image and video generation models.

Bio:
Boqing Gong is a computer science faculty member at Boston University and a part-time research scientist at Google DeepMind. His research focuses on AI models' generalization and efficiency and the visual analytics of objects, scenes, human activities, and their interactions.

https://www.cics.umass.edu/category/machine-learning-and-friends-lunch

Previously: Computer Vision Tasks

Classification	Classification + Localization	Object Detection	Instance Segmentation
			
CAT	CAT	CAT, DOG, DUCK	CAT, DOG, DUCK

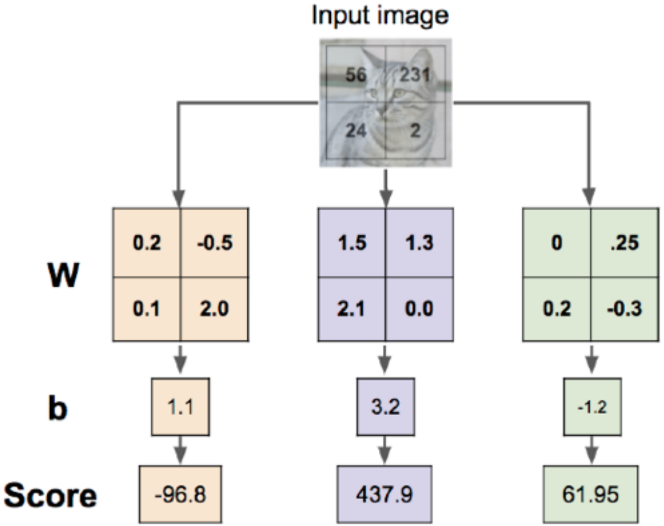
Single object

Multiple objects

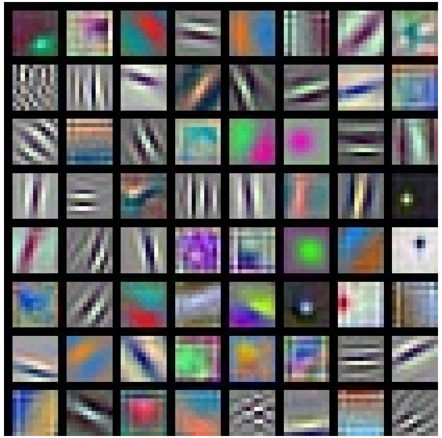
Today: Understanding ConvNets

- Visualize the weights
- Visualize the last layer (via t-SNE)
- Visualize patches that maximally activate neurons
- Occlusion experiments
- Deconv approaches (single backward pass)
- Optimization over image approaches (optimization)

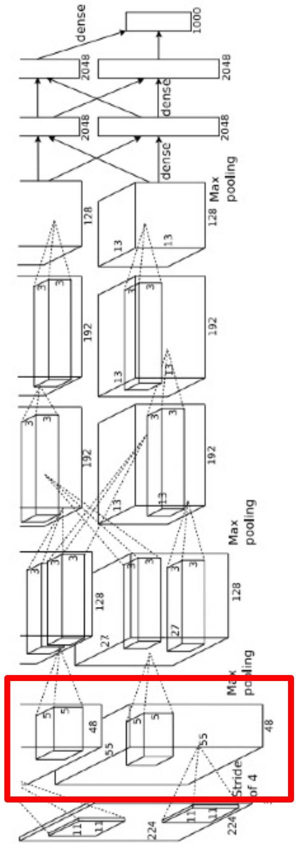
Interpreting a Linear Classifier: Visual Viewpoint



First Layer: Visualize Filters

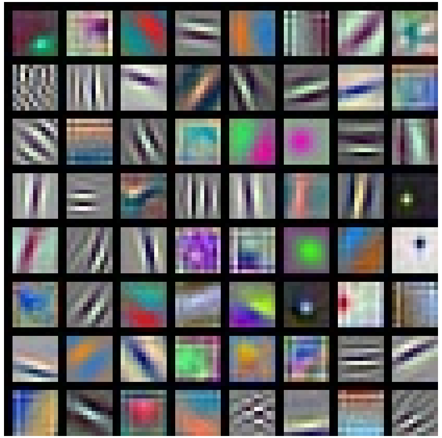


AlexNet:
64 x 3 x 11 x 11

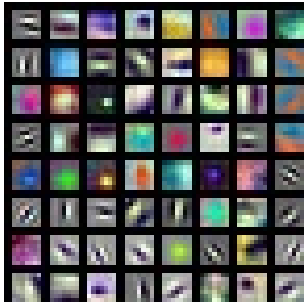


Krizhevsky, “One weird trick for parallelizing convolutional neural networks”, arXiv 2014
He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016
Huang et al, “Densely Connected Convolutional Networks”, CVPR 2017

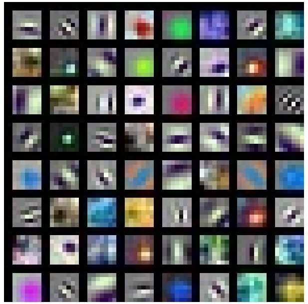
First Layer: Visualize Filters



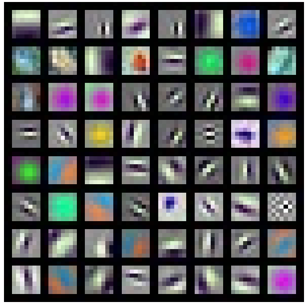
AlexNet:
64 x 3 x 11 x 11



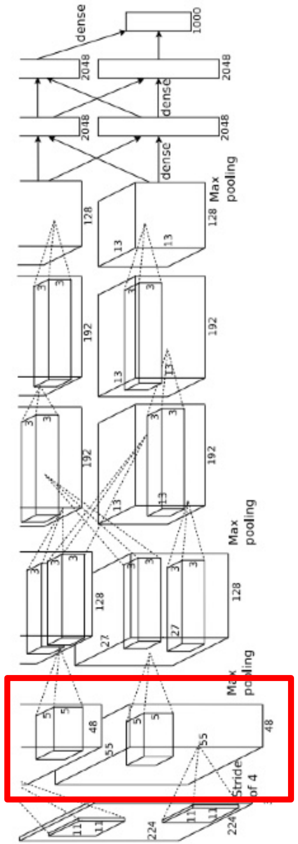
ResNet-18:
64 x 3 x 7 x 7



ResNet-101:
64 x 3 x 7 x 7



DenseNet-121:
64 x 3 x 7 x 7

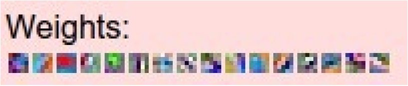


Krizhevsky, “One weird trick for parallelizing convolutional neural networks”, arXiv 2014
He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016
Huang et al, “Densely Connected Convolutional Networks”, CVPR 2017

Visualize the filters/kernels (raw weights)

you can still do it for higher layers, it's just not that interesting

(these are taken from ConvNetJS CIFAR-10 demo)



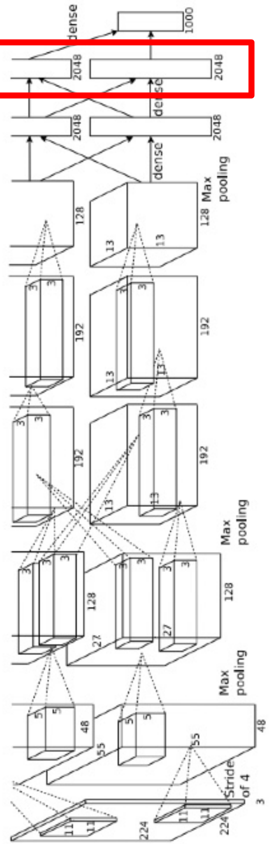
layer 1 weights

layer 2 weights

layer 3 weights

Last Layer

FC7 layer



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

Run the network on many images, collect the feature vectors

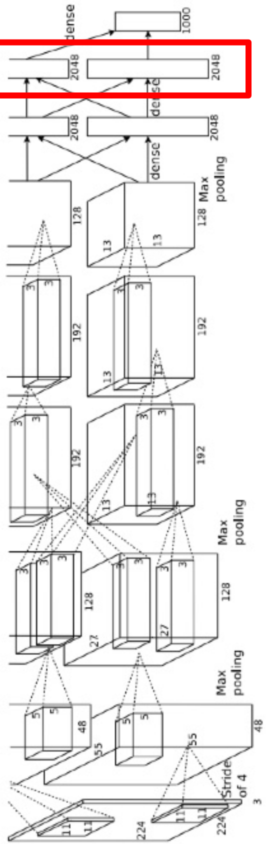
Last Layer: Nearest Neighbors

Recall: Nearest neighbors in pixel space

Krizhevsky et al, “ImageNet Classification with Deep Convolutional Neural Networks”, NIPS 2012.
Figures reproduced with permission.

Last Layer: Nearest Neighbors

4096-dim vector



Test image   L2 Nearest neighbors in feature space

Recall: Nearest neighbors in pixel space

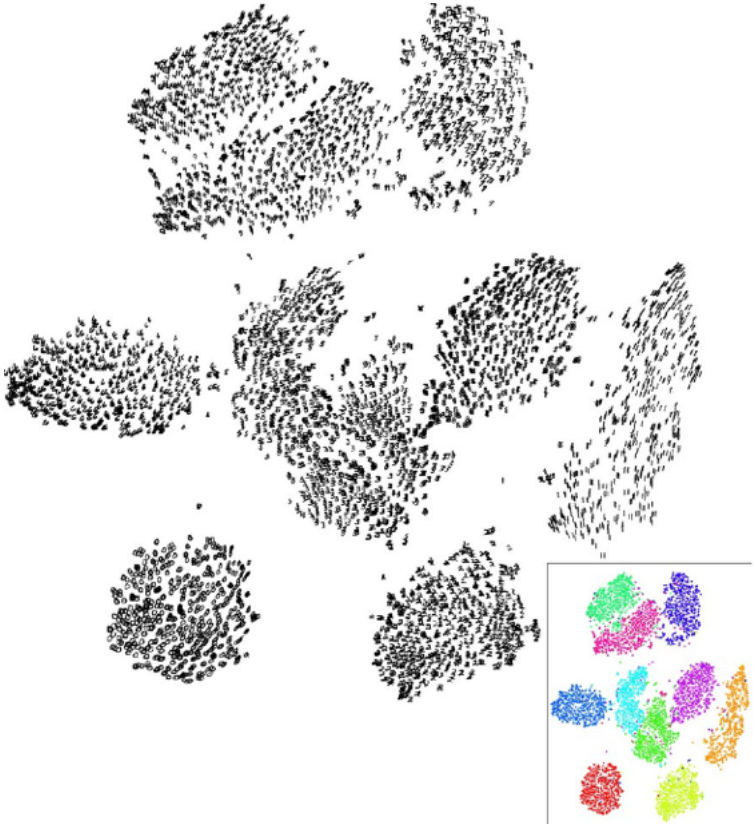
Krizhevsky et al, “ImageNet Classification with Deep Convolutional Neural Networks”, NIPS 2012.
Figures reproduced with permission.

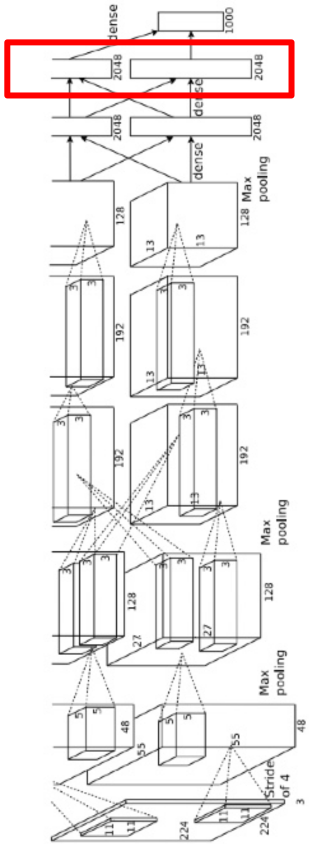
Last Layer: Dimensionality Reduction

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

Simple algorithm: Principal Component Analysis (PCA)

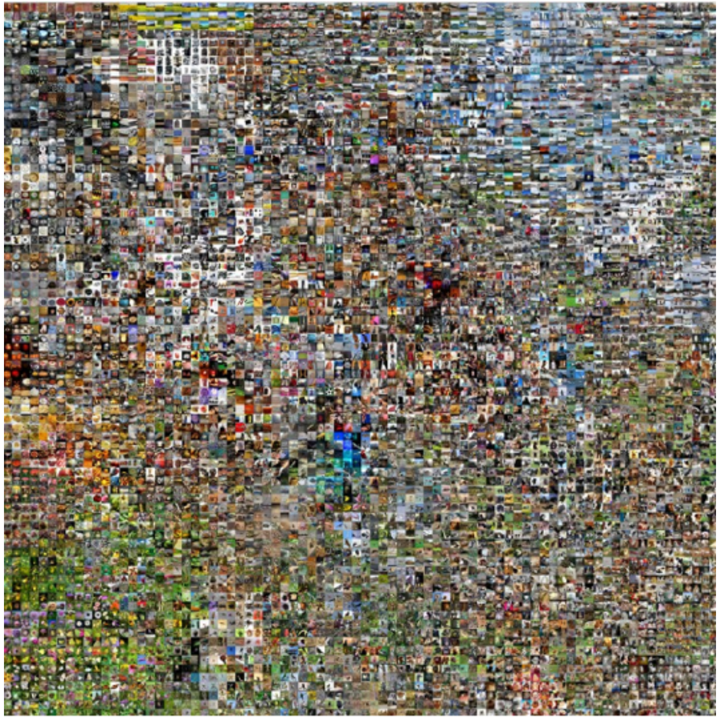
More complex: **t-SNE**

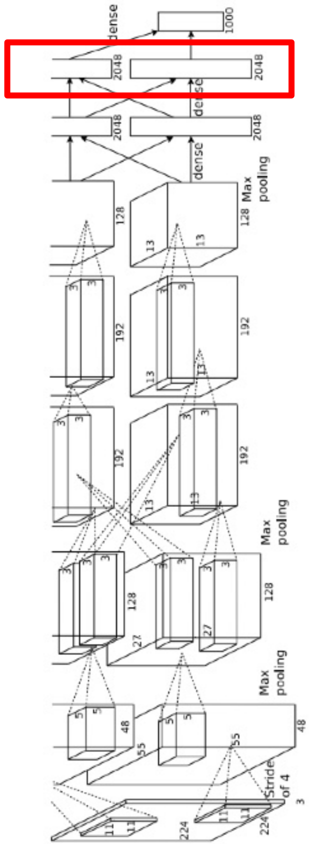




Van der Maaten and Hinton, "Visualizing Data using t-SNE", JMLR 2008

Last Layer: Dimensionality Reduction





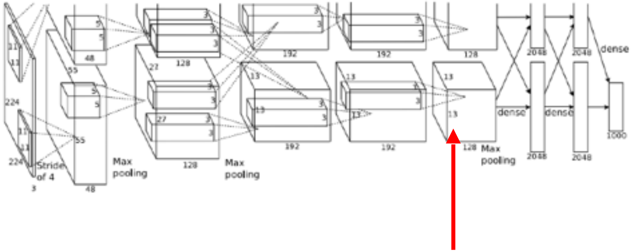
Van der Maaten and Hinton, “Visualizing Data using t-SNE”, JMLR 2008
Krizhevsky et al, “ImageNet Classification with Deep Convolutional Neural Networks”, NIPS 2012.
Figure reproduced with permission.

See high-resolution versions at
http://cs.stanford.edu/people/karpathy/cnnembed/

Today: Understanding ConvNets

- Visualize the weights
- Visualize the last layer (via t-SNE)
- Visualize patches that maximally activate neurons
- Occlusion experiments
- Deconv approaches (single backward pass)
- Optimization over image approaches (optimization)

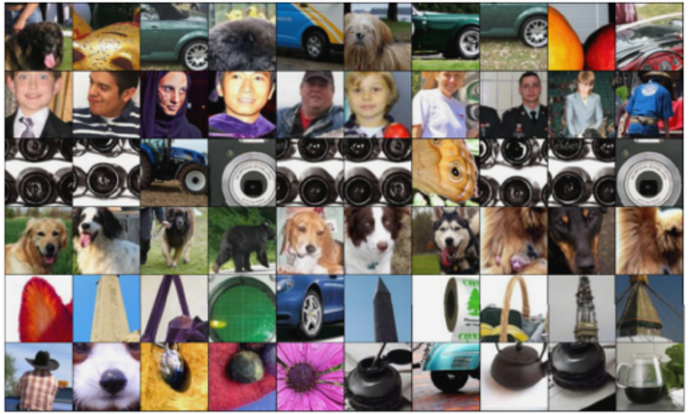
Maximally Activating Patches



Pick a layer and a channel; e.g. conv5 is 128 x 13 x 13, pick channel 17/128

Run many images through the network, record values of chosen channel

Visualize image patches that correspond to maximal activations

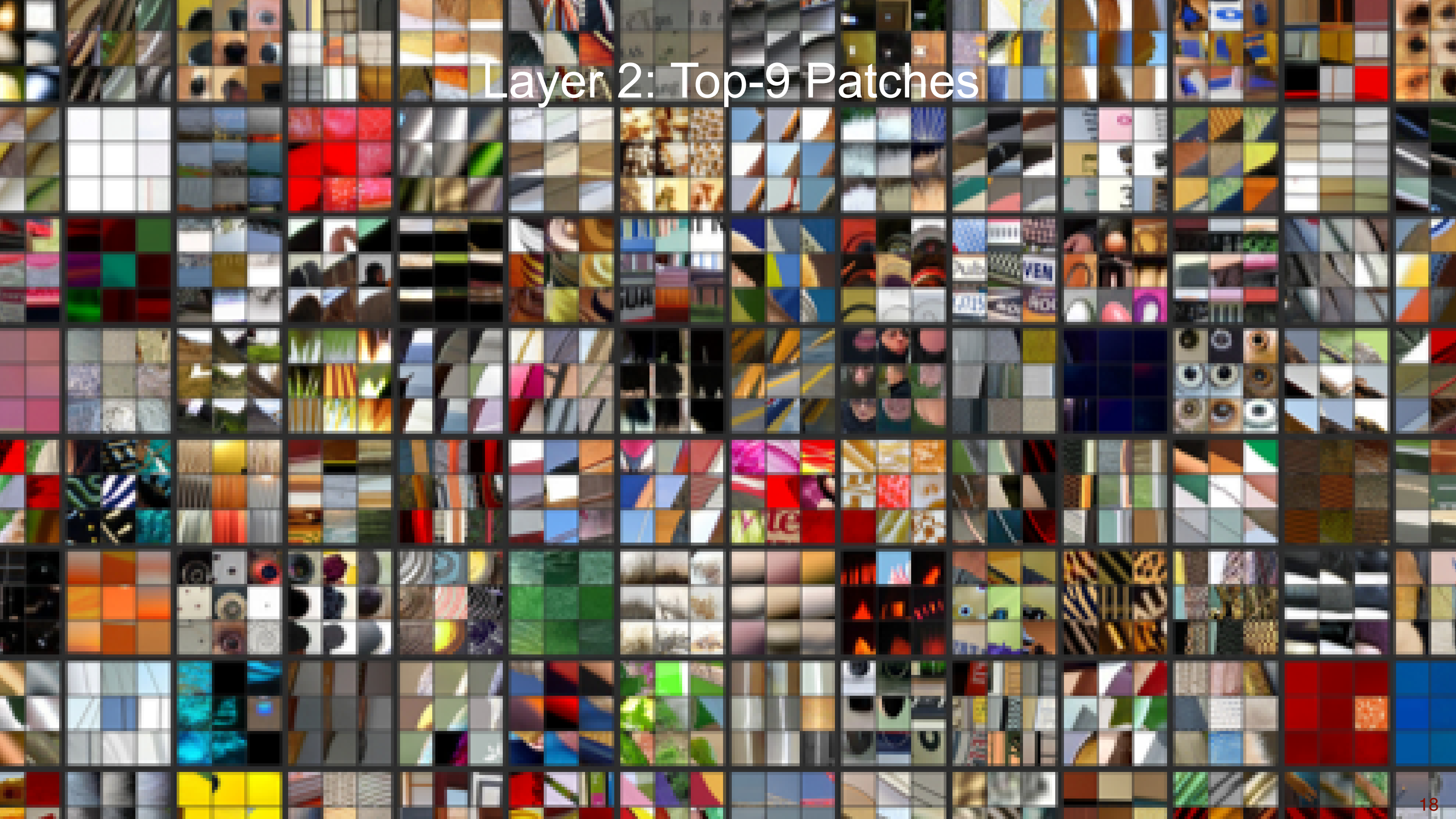


Springenberg et al, “Striving for Simplicity: The All Convolutional Net”, ICLR Workshop 2015

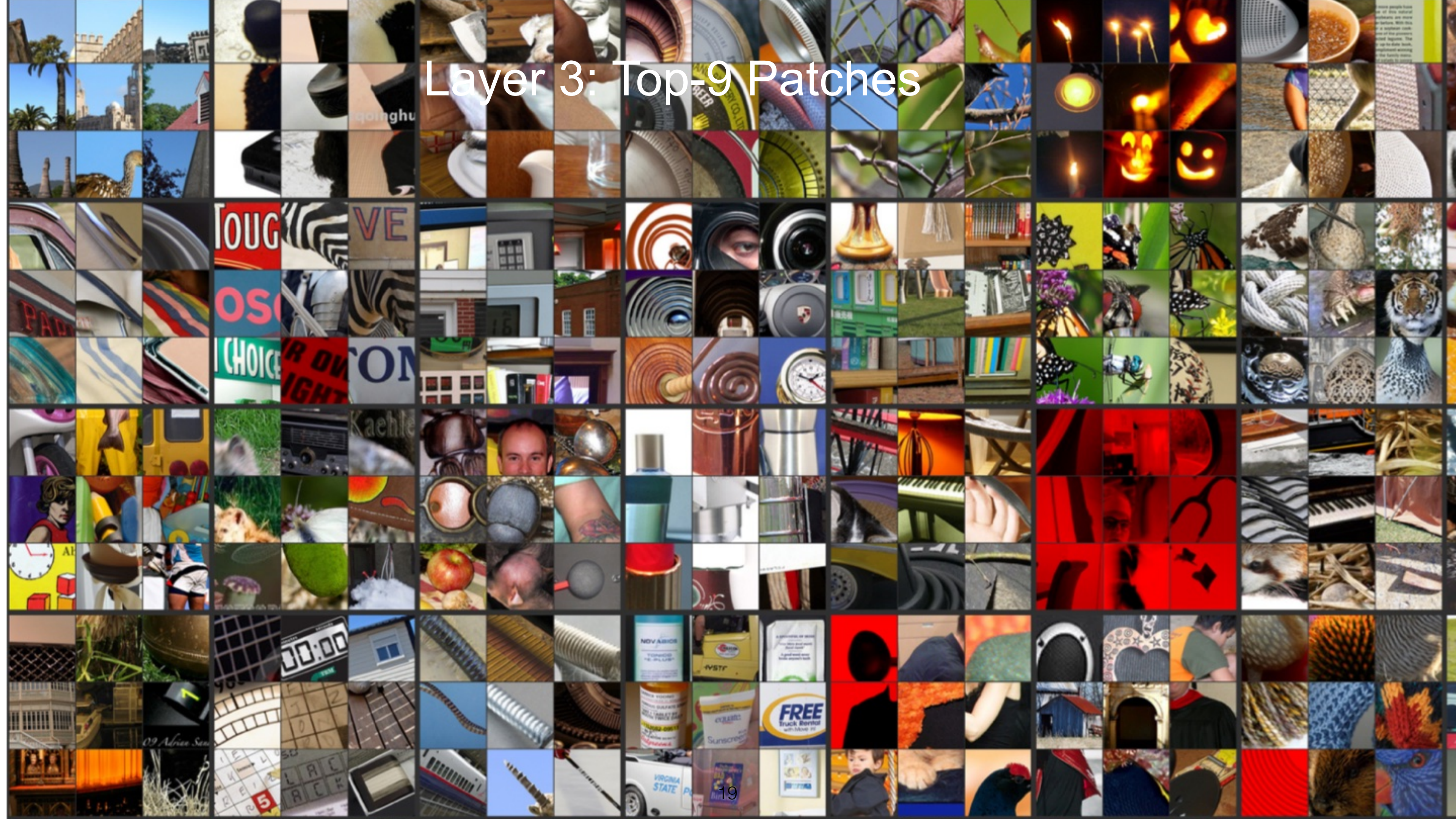
Layer 1: Top-9 Patches

- **Patches from validation images that give maximal activation of a given feature map**

Layer 2: Top-9 Patches



Layer 3: Top-9 Patches

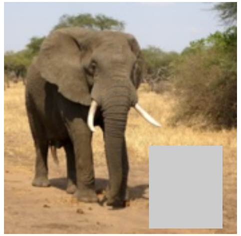
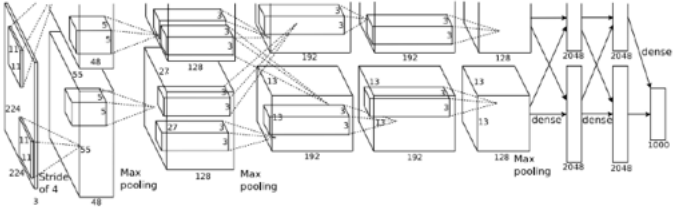


Layer 4: Top-9 Patches

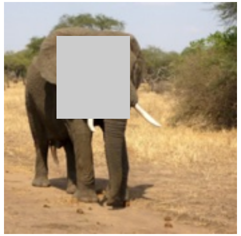
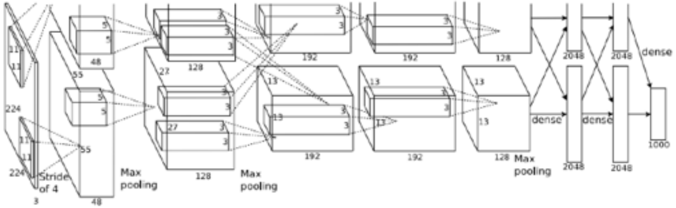

Layer 5: Top-9 Patches

Which pixels matter:

Mask part of the image before feeding to CNN, check how much predicted probabilities change

P(elephant) = 0.95

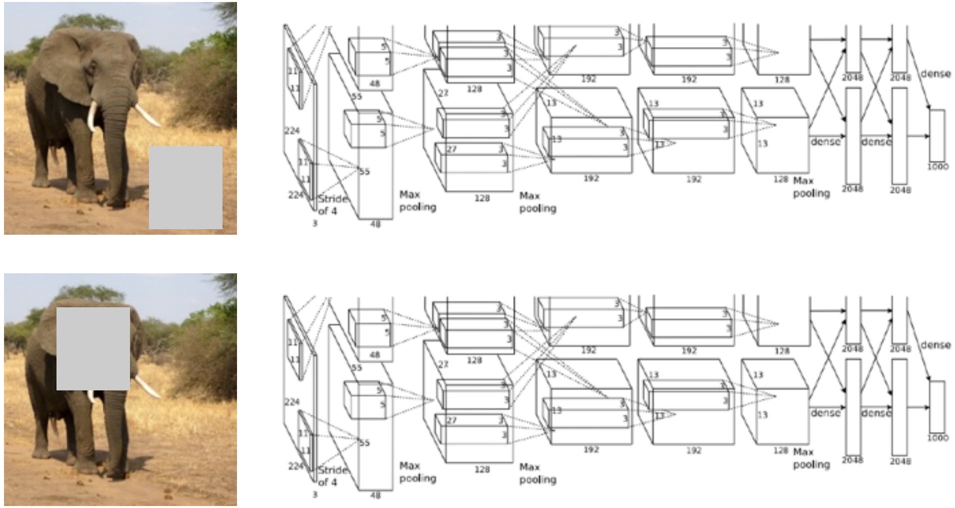



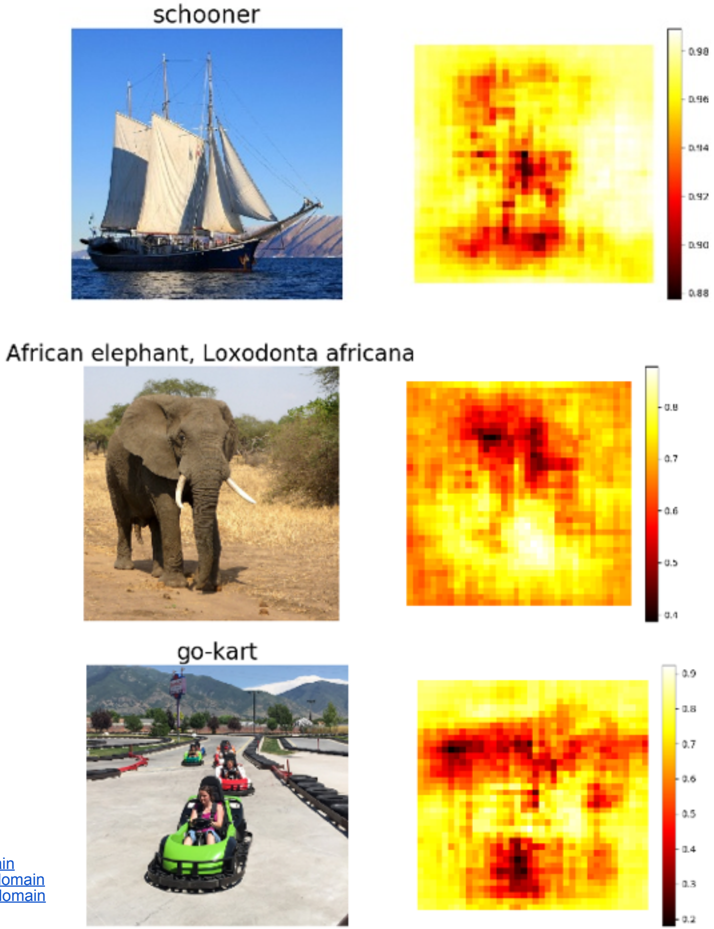
P(elephant) = 0.75

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

Which pixels matter:

Mask part of the image before feeding to CNN,
check how much predicted probabilities change





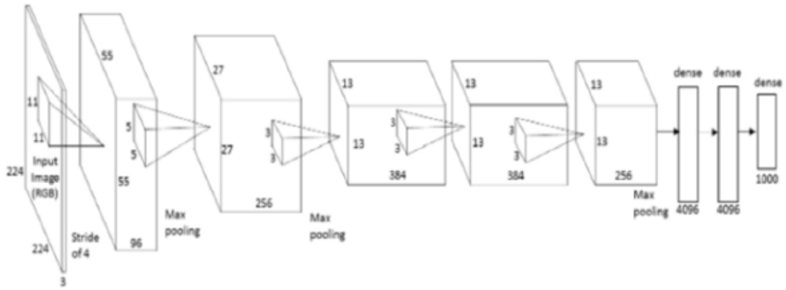
Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

Today: Understanding ConvNets

- Visualize the weights
- Visualize the last layer (via t-SNE)
- Visualize patches that maximally activate neurons
- Occlusion experiments
- Deconv approaches (single backward pass)
- Optimization over image approaches (optimization)

Intermediate features visualization via backprop

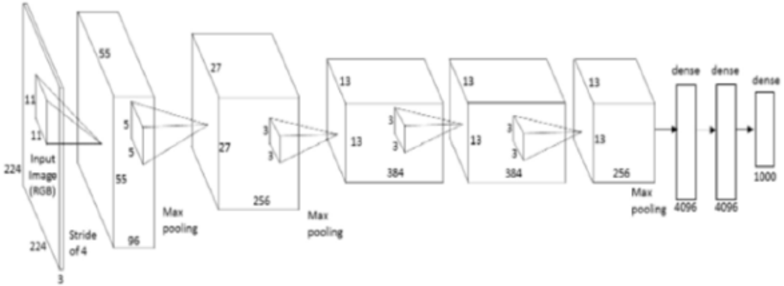
1. Feed image into net

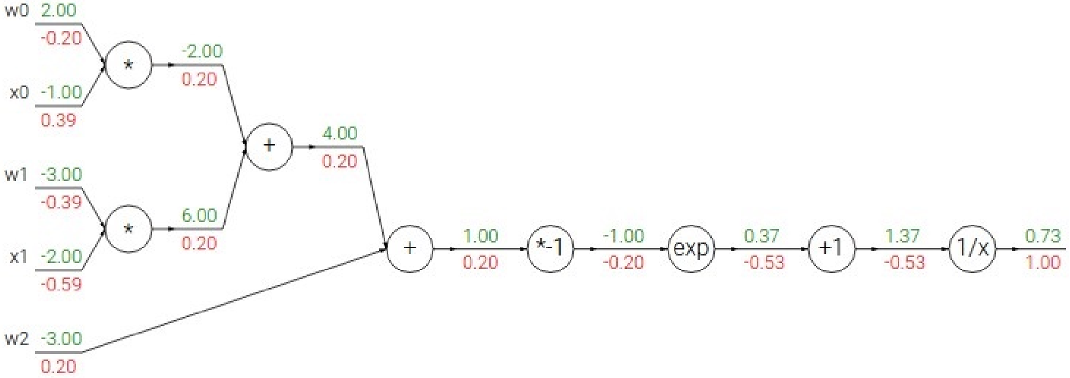


Q: how can we compute the gradient of any arbitrary neuron in the network w.r.t. the image?

Intermediate features visualization via backprop

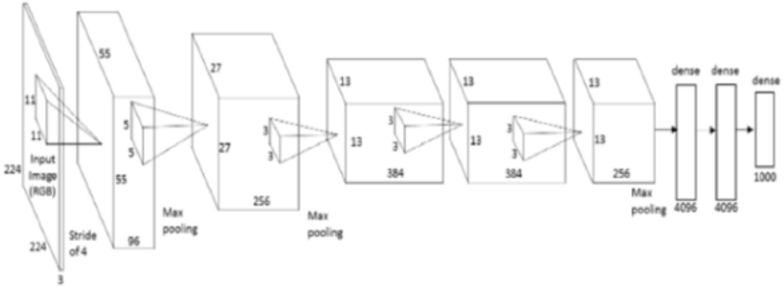
1. Feed image into net



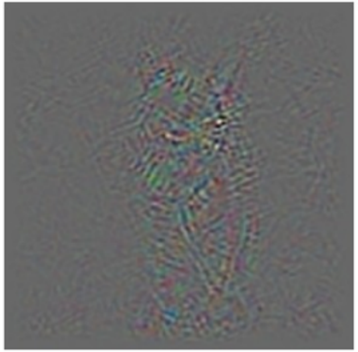


Intermediate features visualization via backprop

1. Feed image into net

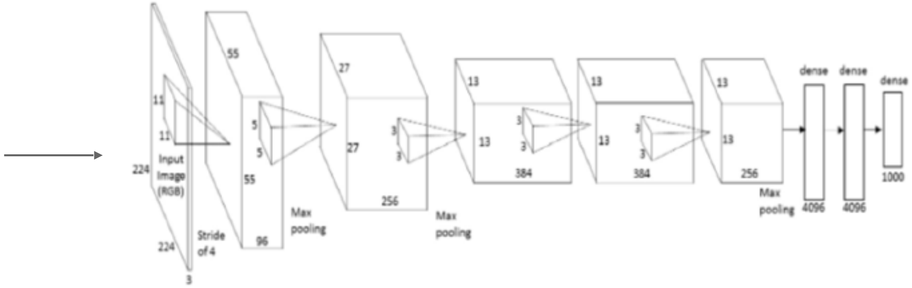


2. Pick a layer, set the gradient there to be all zero except for one 1 for some neuron of interest
3. Backprop to image:

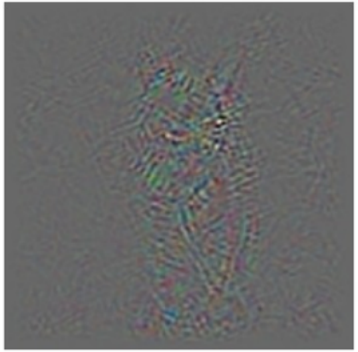


Intermediate features visualization via backprop

1. Feed image into net



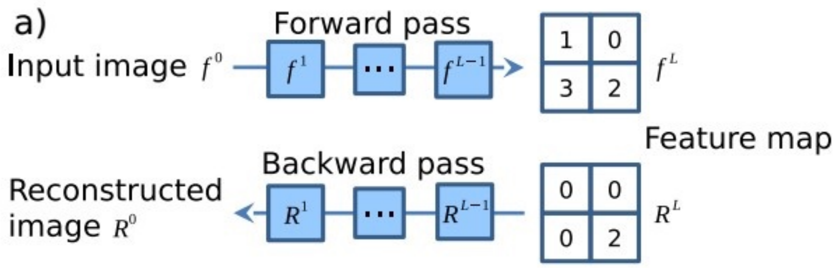
2. Pick a layer, set the gradient there to be all zero except for one 1 for some neuron of interest
3. Backprop to image:



"Guided backpropagation:" instead

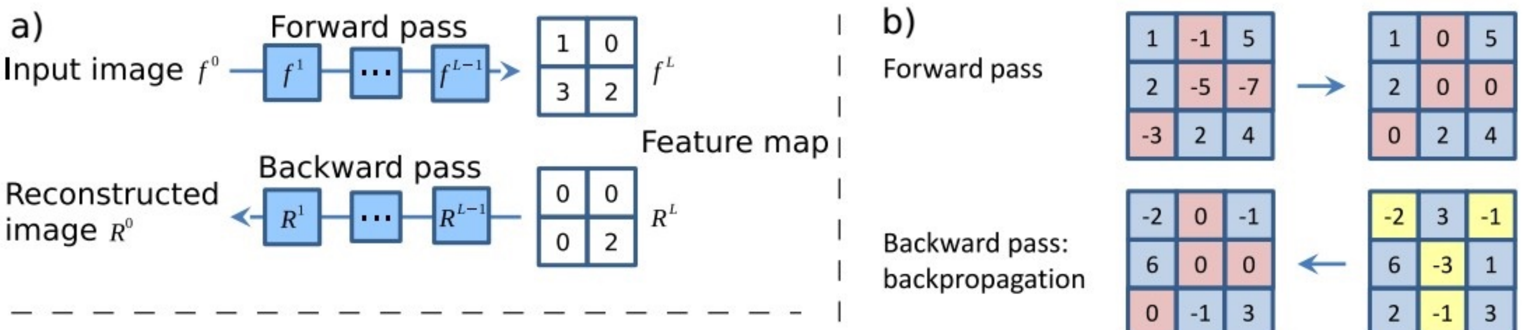
Intermediate features visualization via backprop

[Visualizing and Understanding Convolutional Networks, Zeiler and Fergus 2013]
[Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, Simonyan et al., 2014]
[Striving for Simplicity: The all convolutional net, Springenberg, Dosovitskiy, et al., 2015]



Intermediate features visualization via backprop

[Visualizing and Understanding Convolutional Networks, Zeiler and Fergus 2013]
[Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, Simonyan et al., 2014]
[Striving for Simplicity: The all convolutional net, Springenberg, Dosovitskiy, et al., 2015]



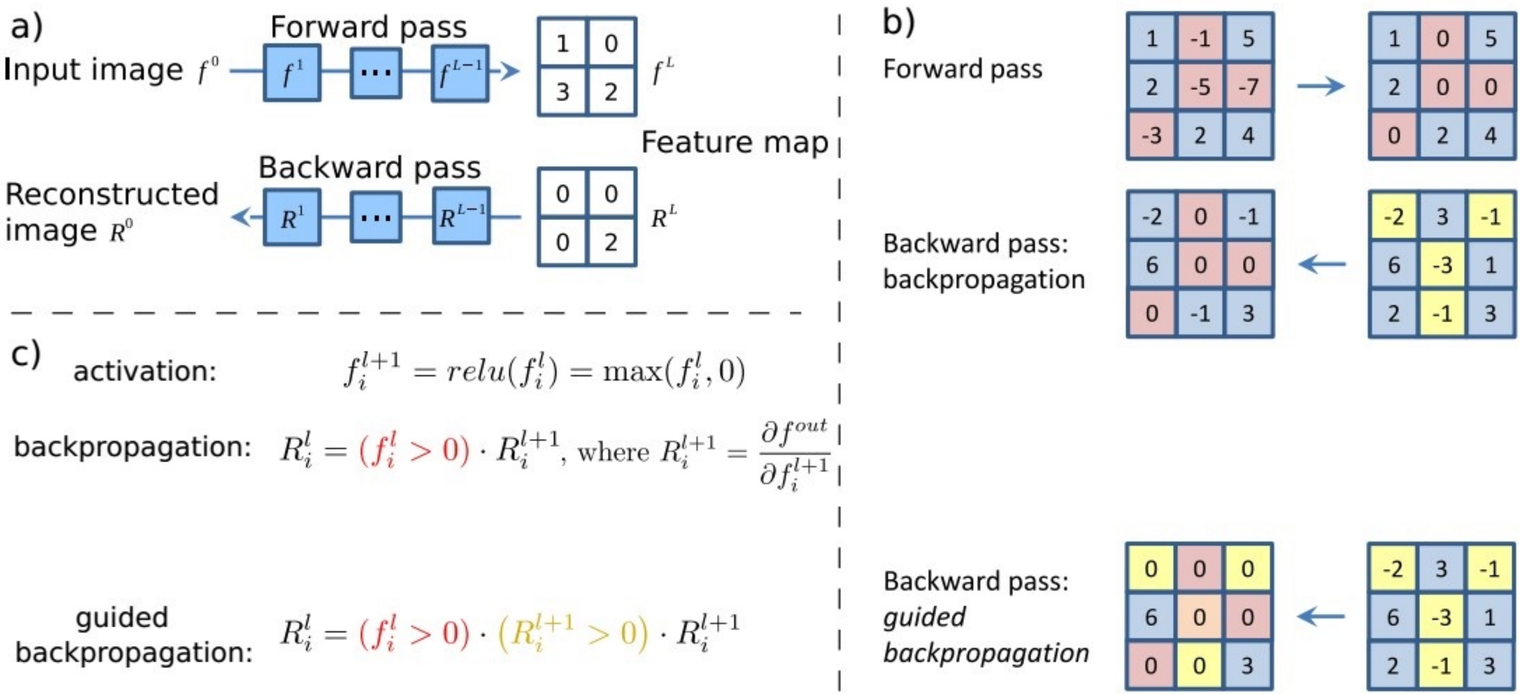
c) activation: $f_i^{l+1} = relu(f_i^l) = \max(f_i^l, 0)$

backpropagation: $R_i^l = (f_i^l > 0) \cdot R_i^{l+1}$, where $R_i^{l+1} = \frac{\partial f^{out}}{\partial f_i^{l+1}}$

Backward pass for a ReLU (will be changed in Guided Backprop)

Intermediate features visualization via backprop

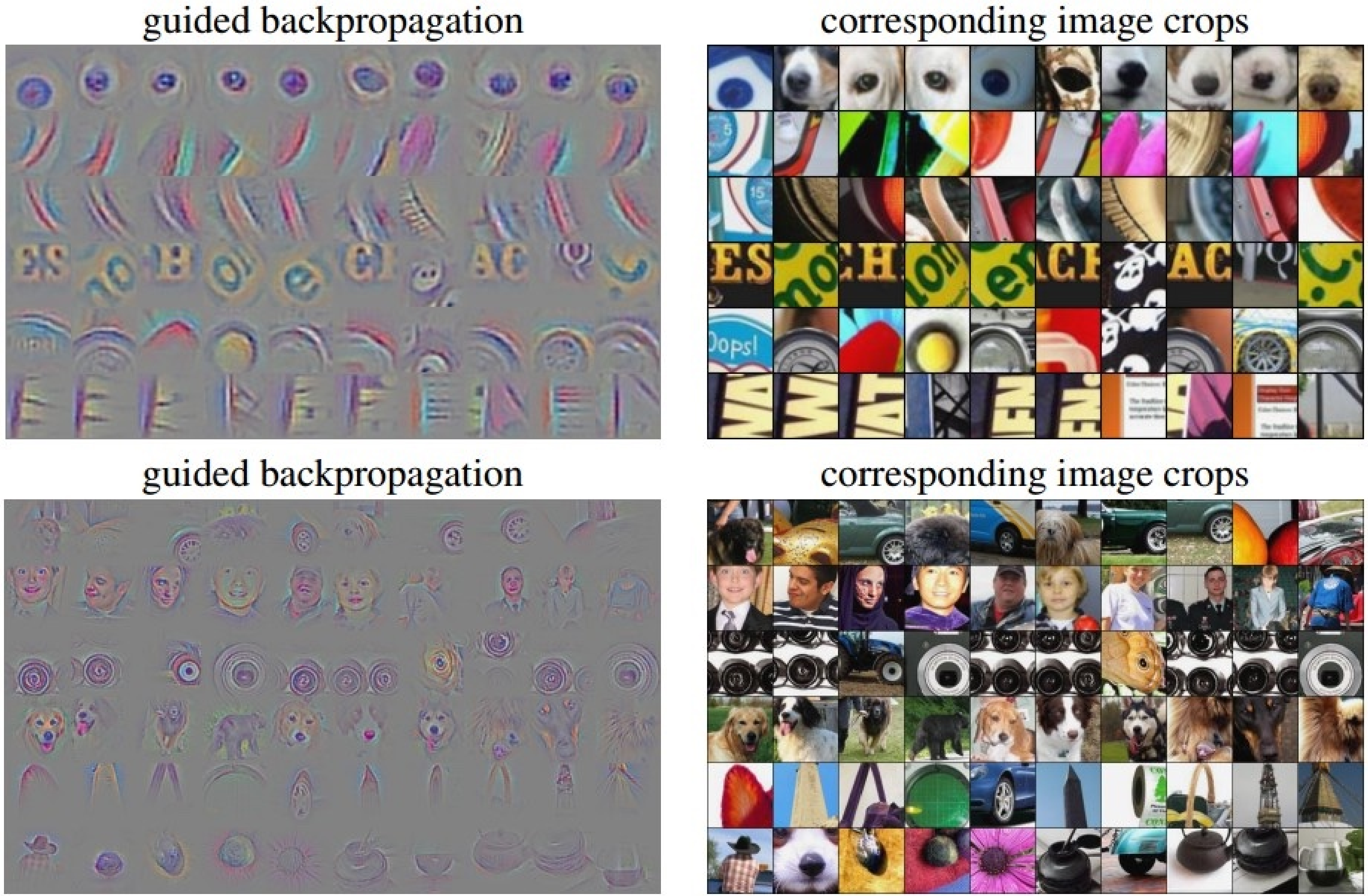
[Visualizing and Understanding Convolutional Networks, Zeiler and Fergus 2013]
[Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, Simonyan et al., 2014]
[Striving for Simplicity: The all convolutional net, Springenberg, Dosovitskiy, et al., 2015]



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

Each row corresponds to one filter.

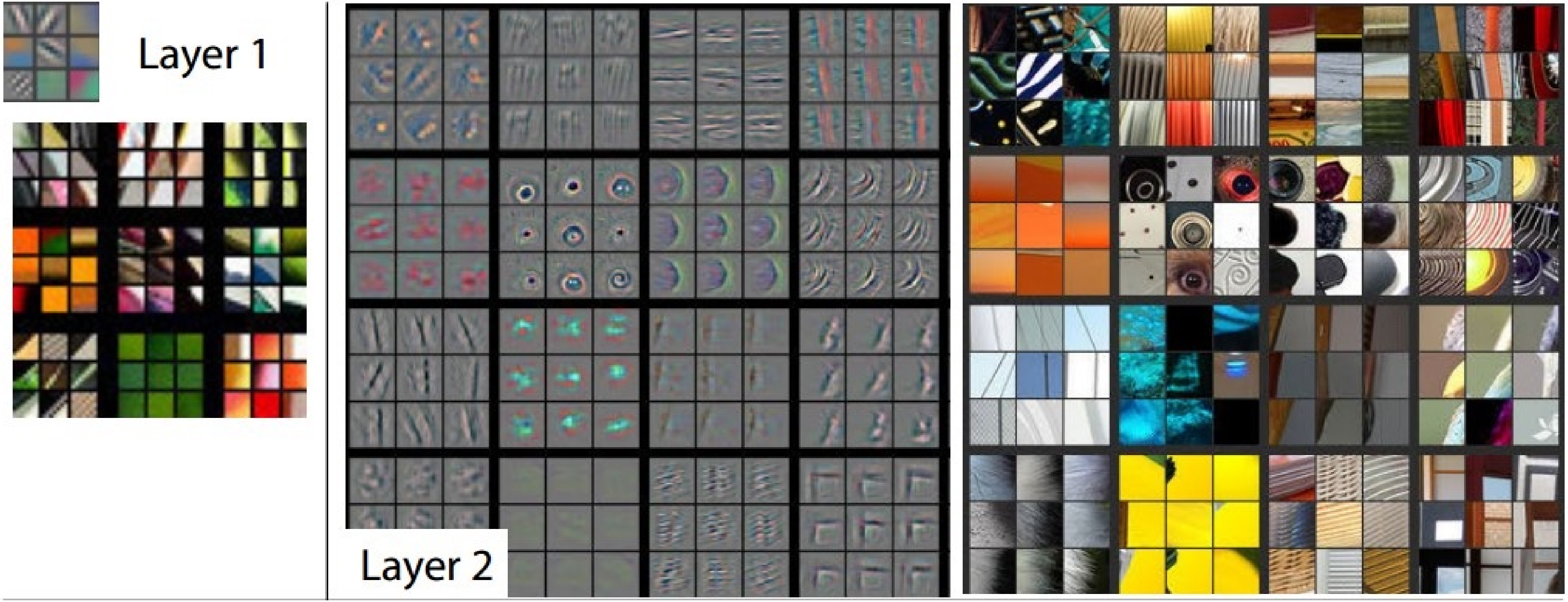
The visualization using “guided backpropagation” is based on the top 10 image patches activating this filter taken from the ImageNet dataset.



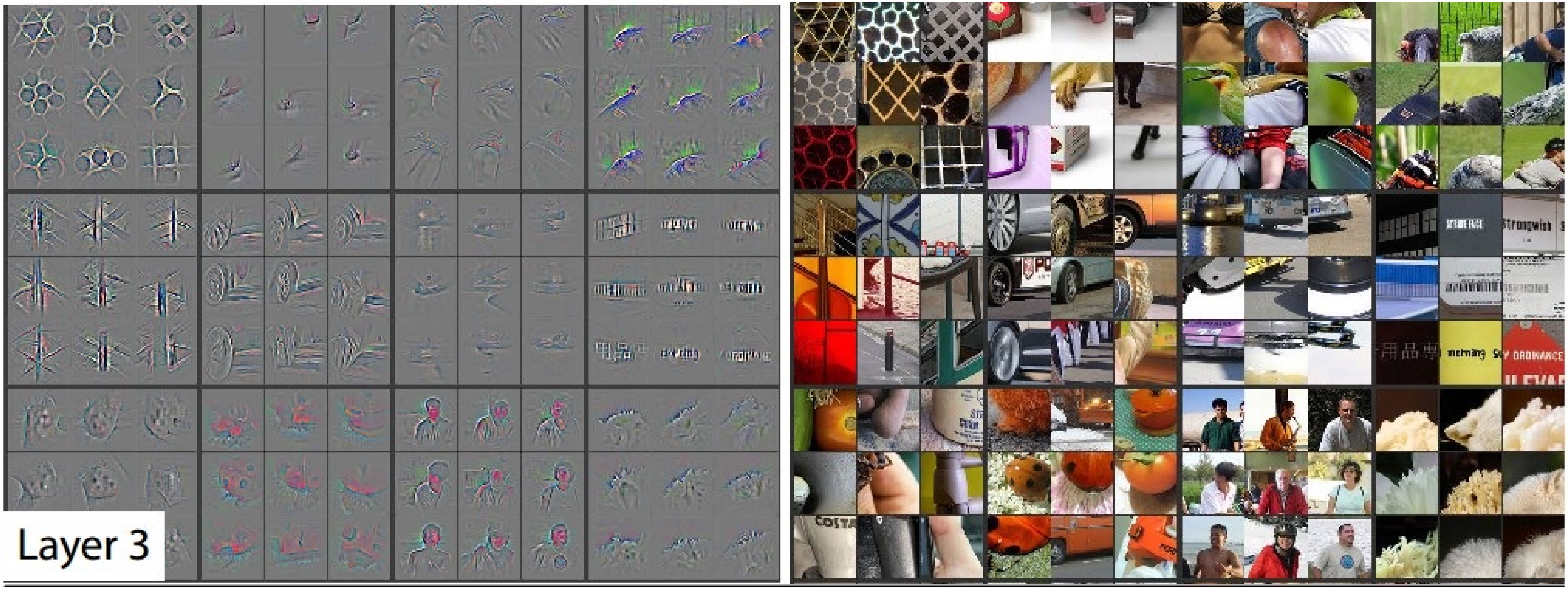
[Striving for Simplicity: The all convolutional net, Springenberg, Dosovitskiy, et al., 2015]

Visualizing and Understanding Convolutional Networks
Zeiler & Fergus, 2013

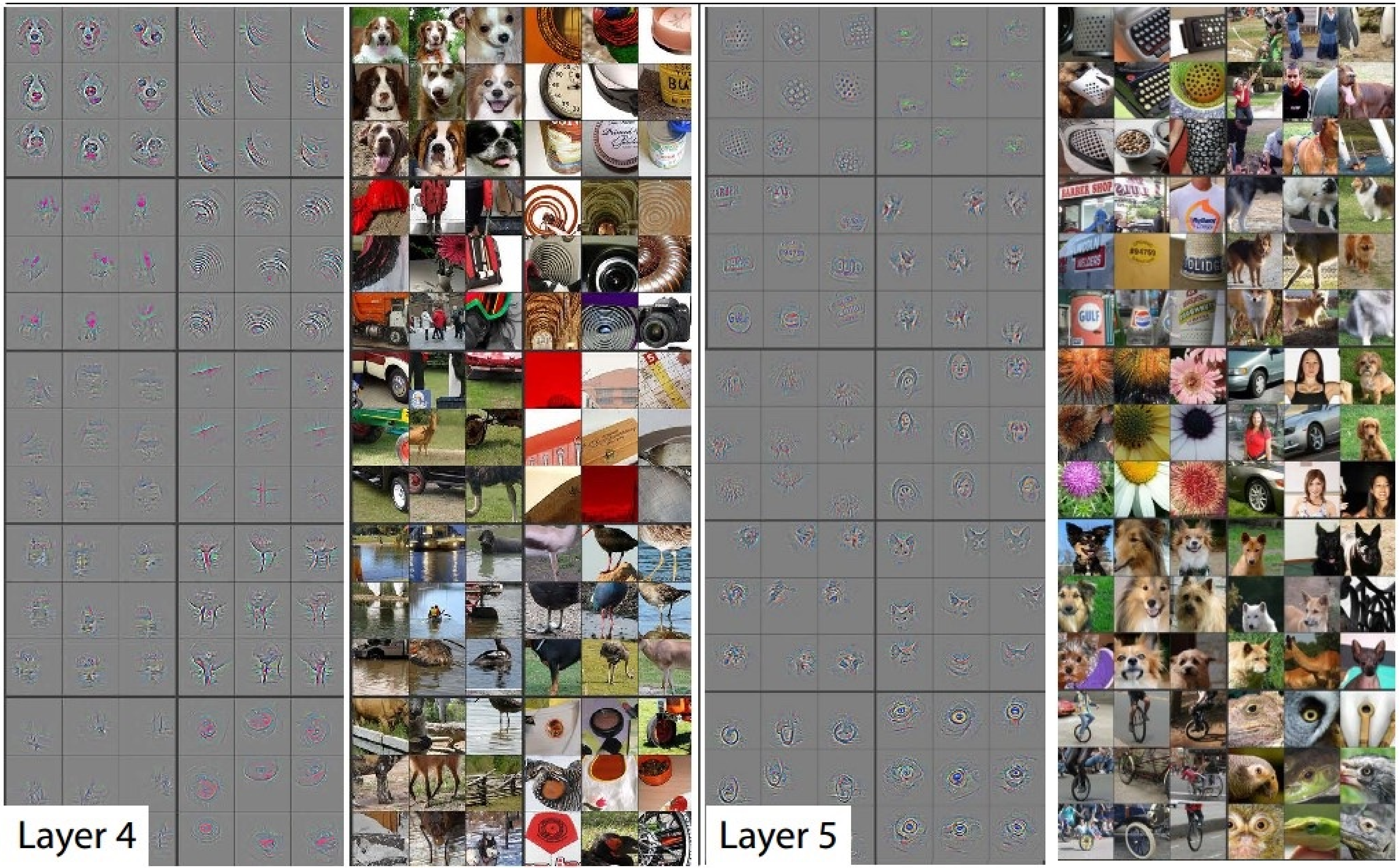
Visualizing arbitrary neurons along the way to the top...



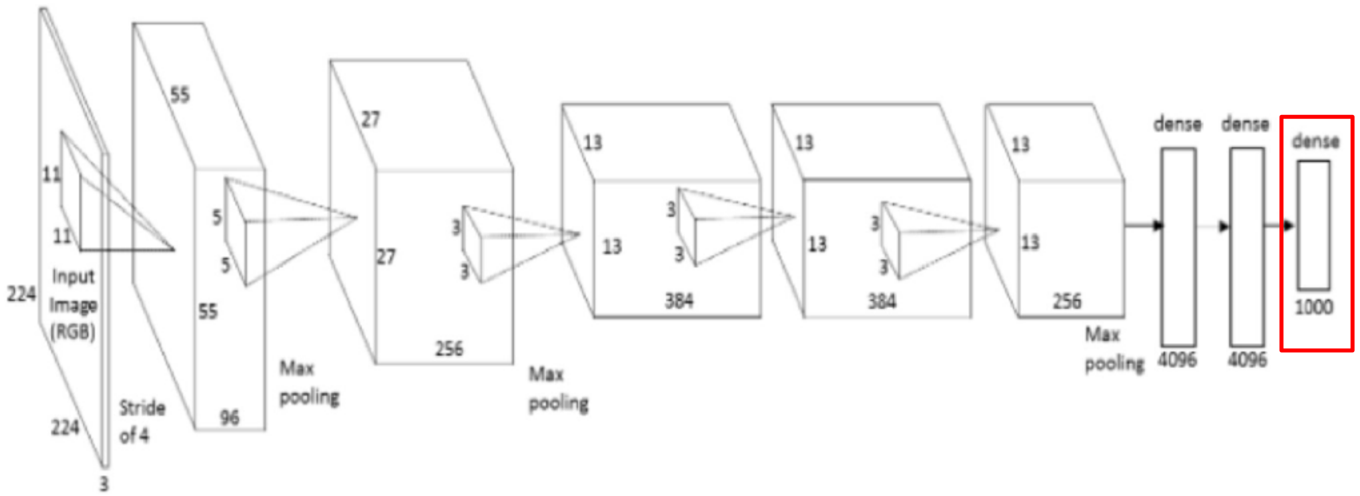
Visualizing arbitrary neurons along the way to the top...



Visualizing arbitrary neurons along the way to the top...



Optimization to Image

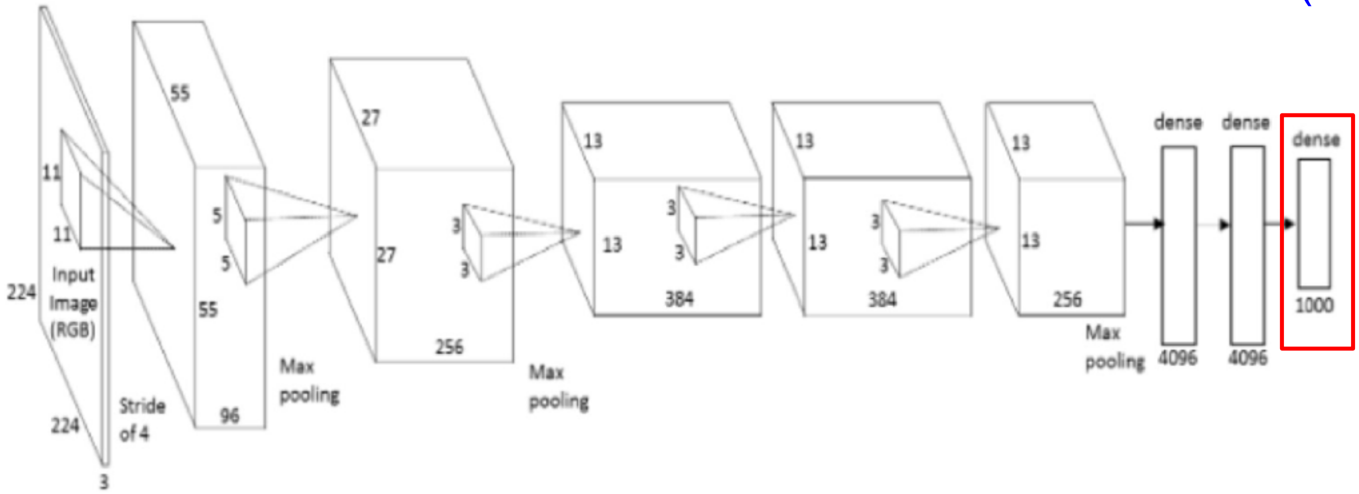


Q: can we find an image that maximizes some class score?

Optimization to Image

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

score for class c (before Softmax)

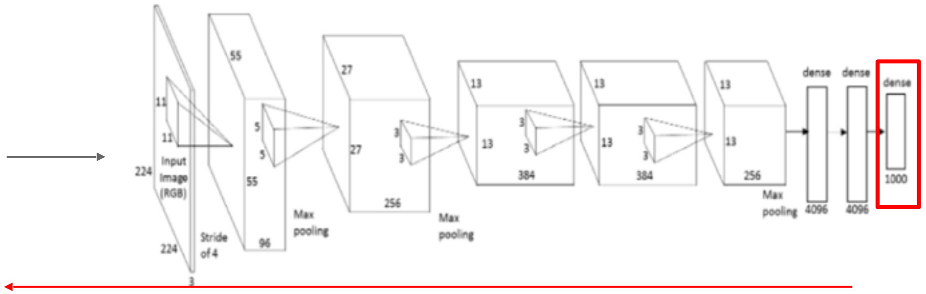


Q: can we find an image that maximizes some class score?

Optimization to Image

1. feed in zeros.

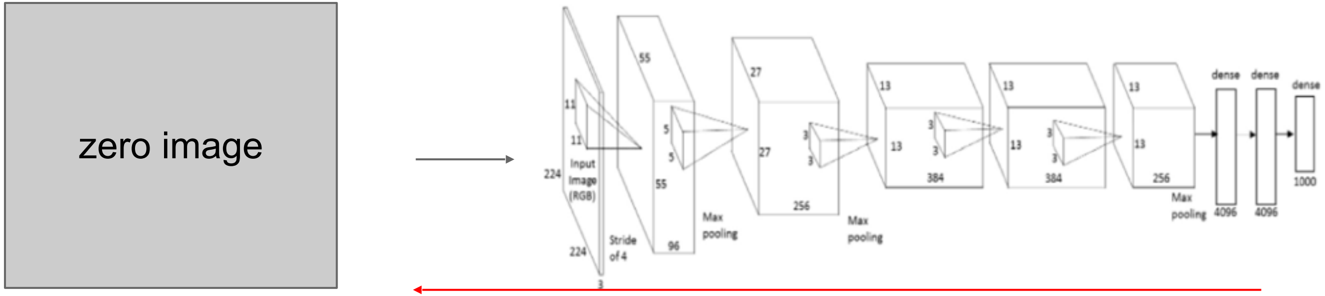
zero image



2. set the gradient of the scores vector to be [0,0,....1,....,0], then backprop to image

Optimization to Image

1. feed in zeros.



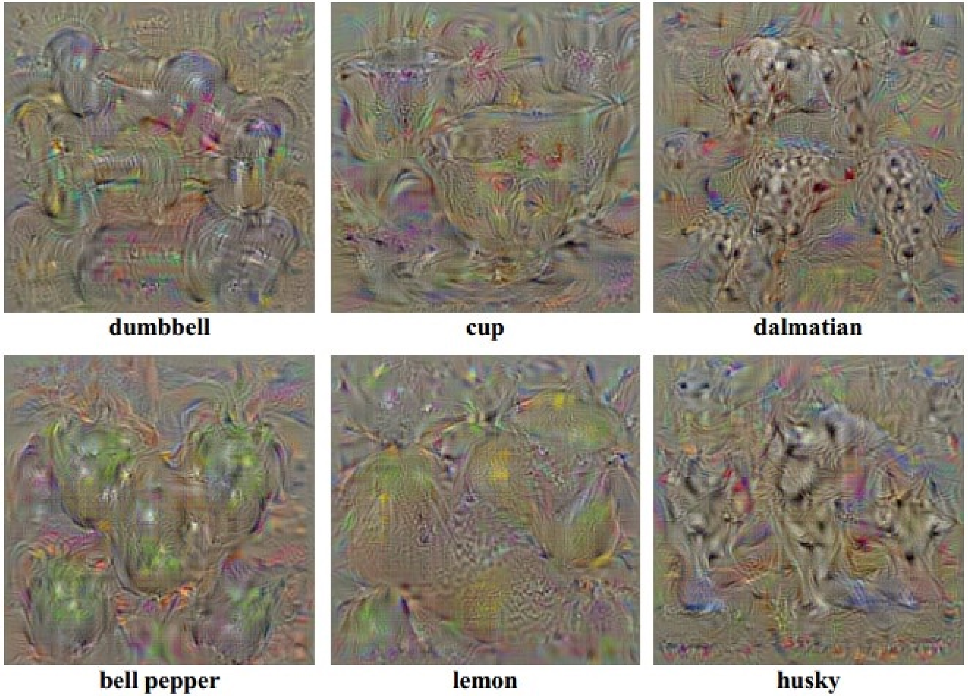
2. set the gradient of the scores vector to be [0,0,....1,....,0], then backprop to image
3. do a small “image update”
4. forward the image through the network.
5. go back to 2.

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

score for class c (before Softmax)

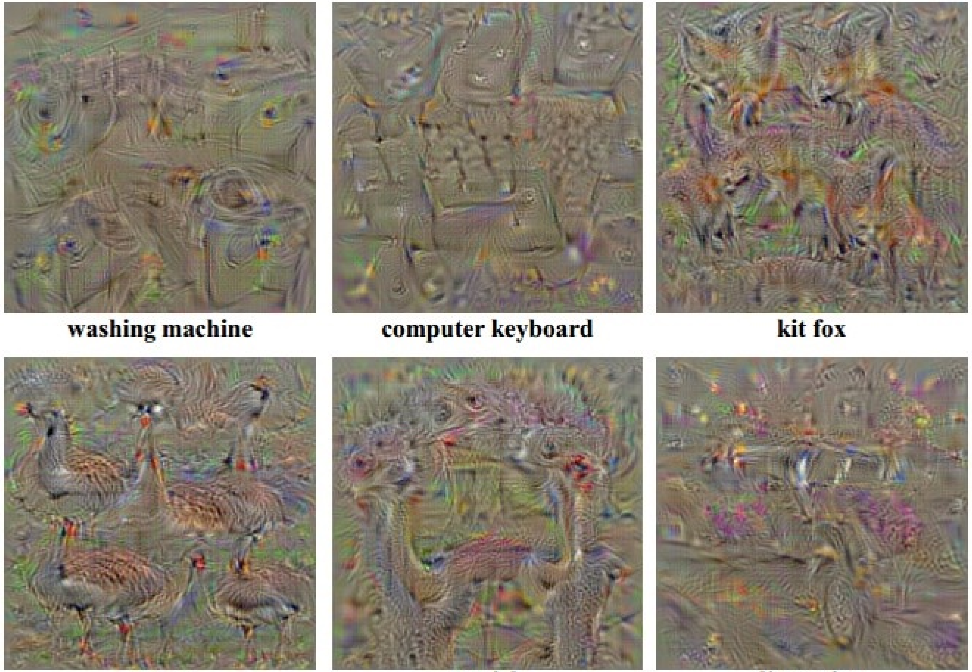
Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps
Karen Simonyan, Andrea Vedaldi, Andrew Zisserman, 2014

1. Find images that maximize some class score:



Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps
Karen Simonyan, Andrea Vedaldi, Andrew Zisserman, 2014

1. Find images that maximize some class score:



washing machine, computer keyboard, kit fox, goose, ostrich, limousine

2. Visualize the Data gradient:

M = ?

(note that the gradient on data has three channels. Here they visualize M, s.t.:

$$M_{ij} = \max_c |w_{h(i,j,c)}|$$

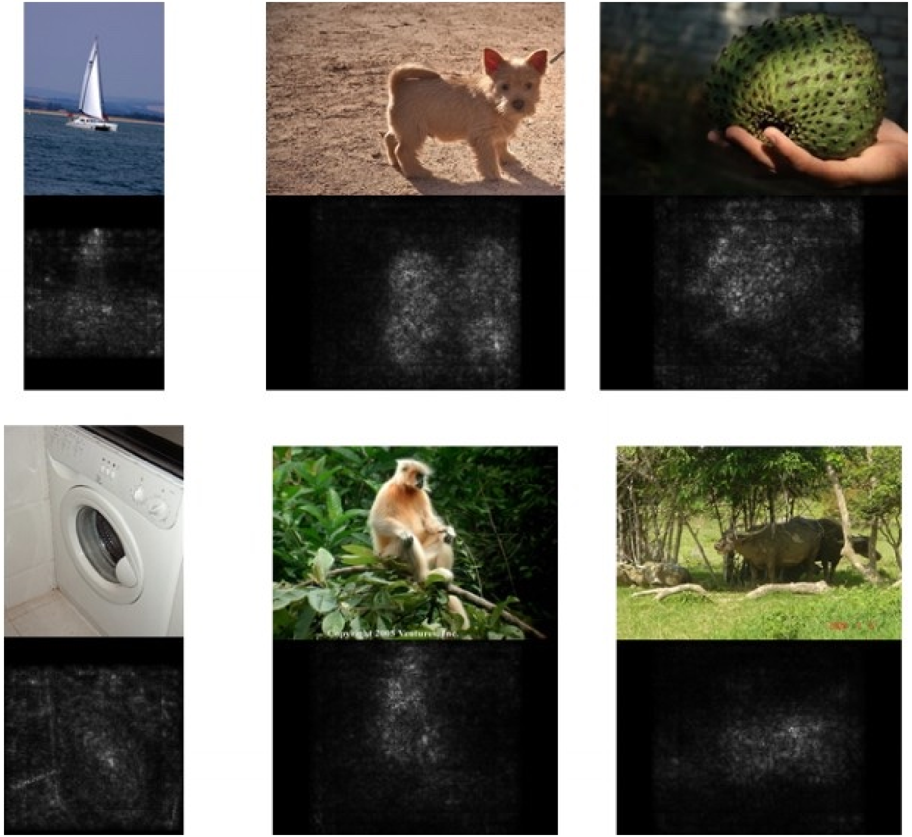
(at each pixel take abs val, and max over channels)

2. Visualize the Data gradient:

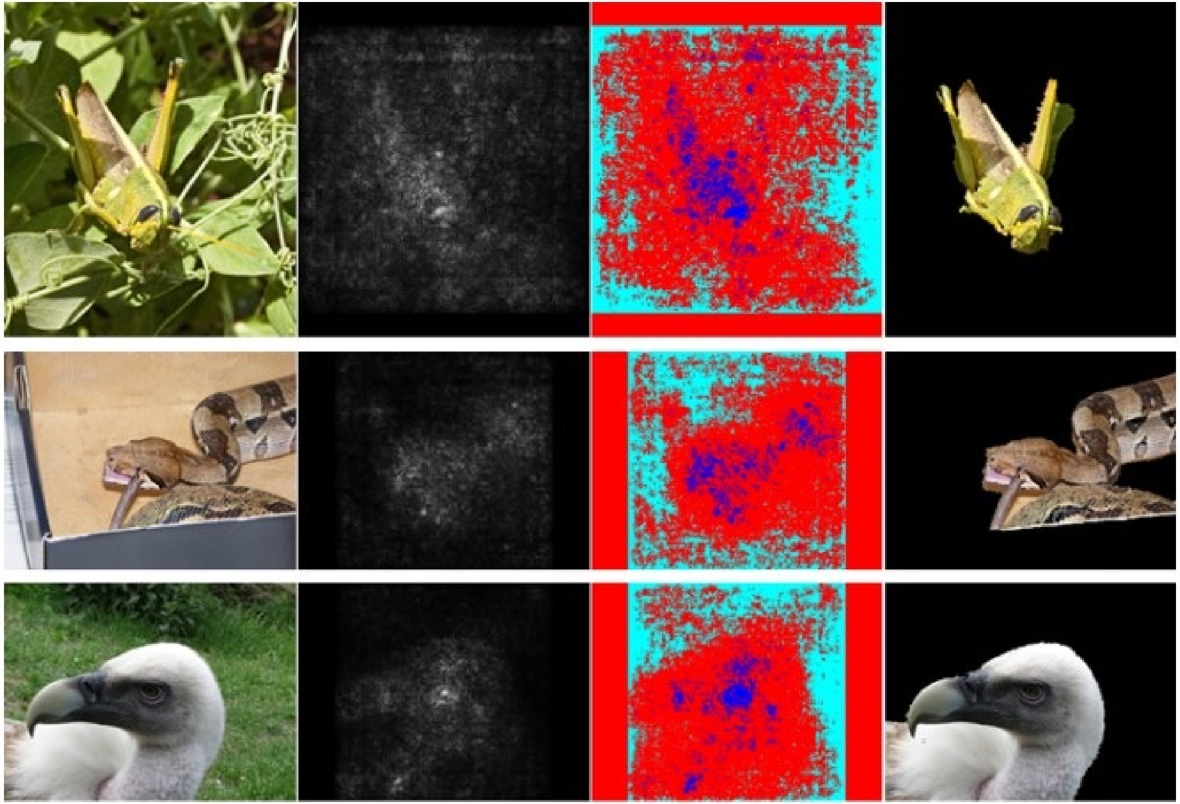
(note that the gradient on data has three channels. Here they visualize M, s.t.:

$$M_{ij} = \max_c |w_{h(i,j,c)}|$$

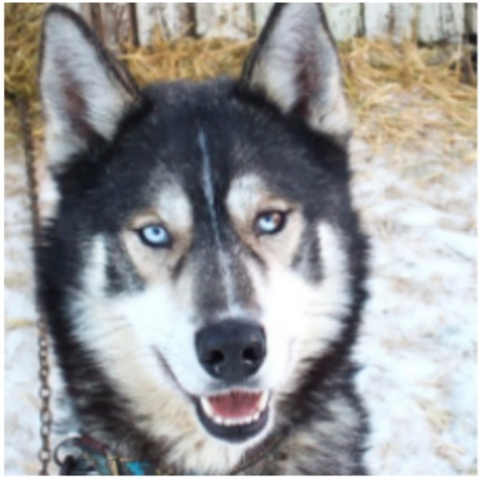
(at each pixel take abs val, and max over channels)



- Use **grabcut** for segmentation



Saliency maps: Uncovers biases

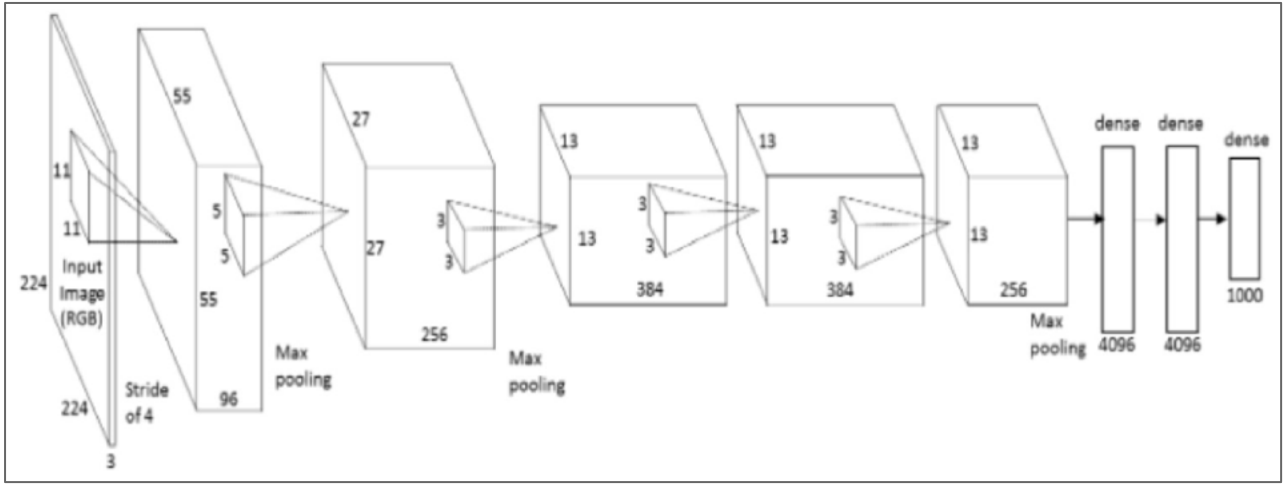


(a) Husky classified as wolf

(b) Explanation

Ribeiro et al, ““Why Should I Trust You?” Explaining the Predictions of Any Classifier”, ACM KDD 2016

We can in fact do this for arbitrary neurons along the ConvNet



Repeat:

1. Forward an image
2. Set activations in layer of interest to all zero, except for a 1.0 for a neuron of interest
3. Backprop to image
4. Do an "image update"

[Understanding Neural Networks Through Deep Visualization, Yosinski et al. , 2015]

Proposed a different form of regularizing the image

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

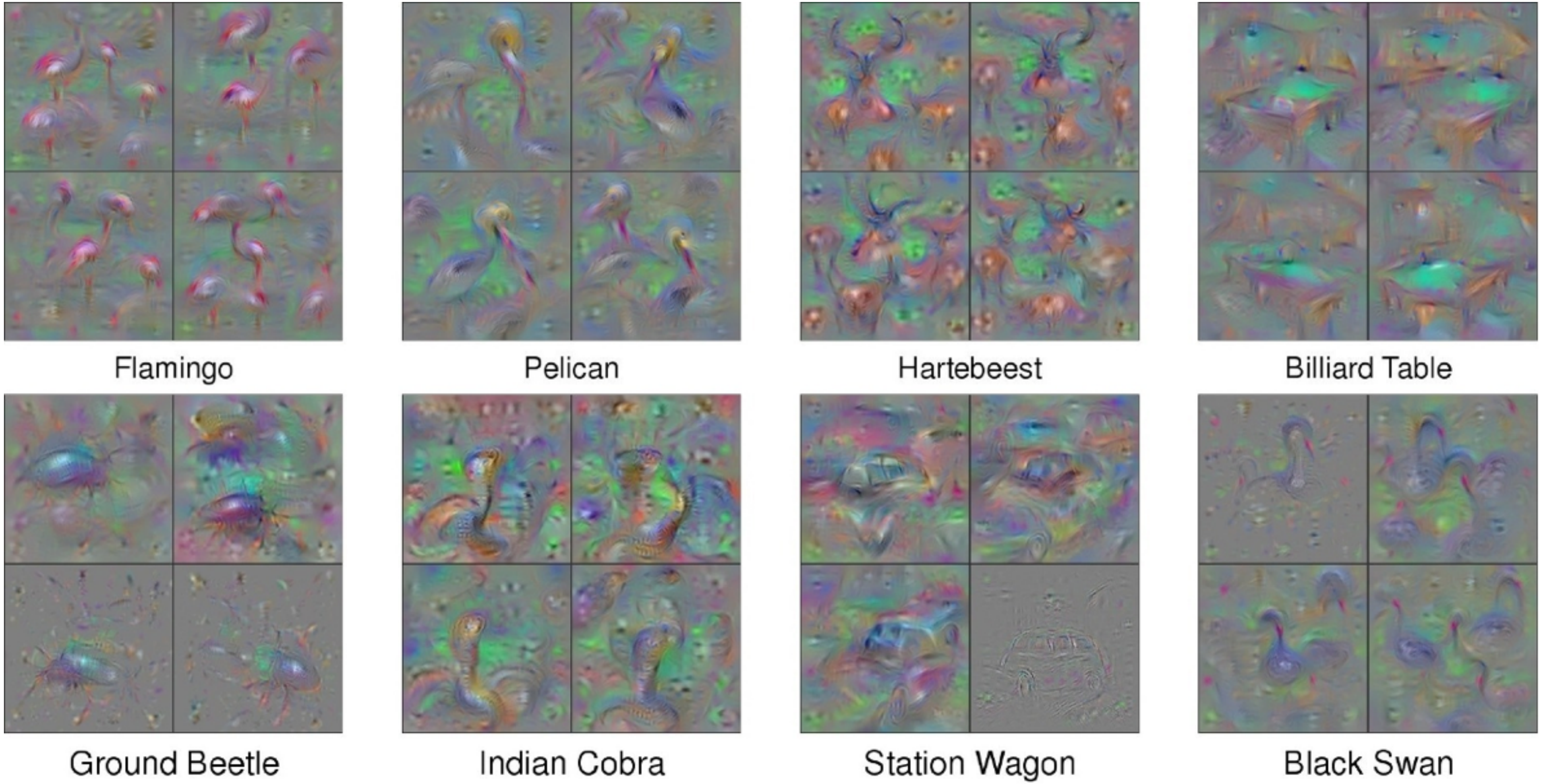
More explicit scheme:

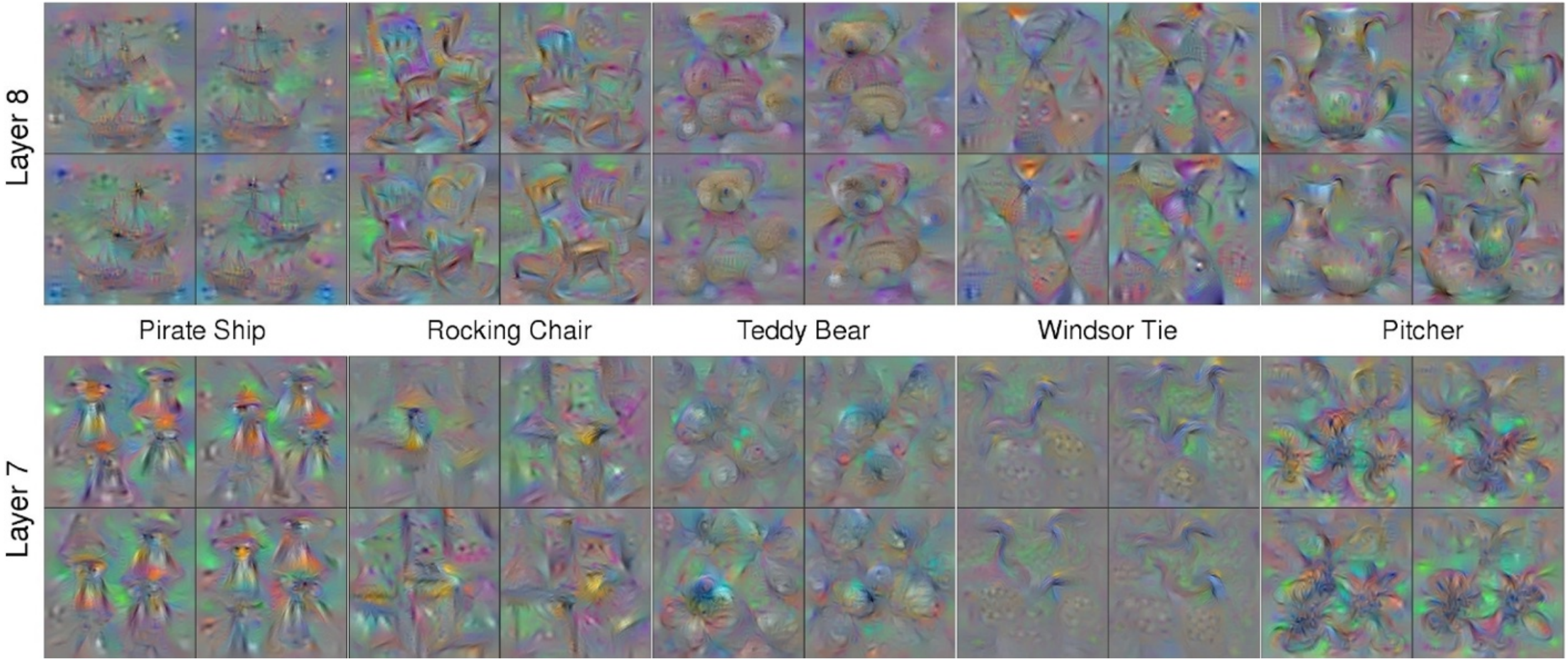
Repeat:

- Update the image ***I*** with gradient from some unit of interest
- Blur ***I*** a bit
- Take any pixel with small norm to zero (to encourage sparsity)

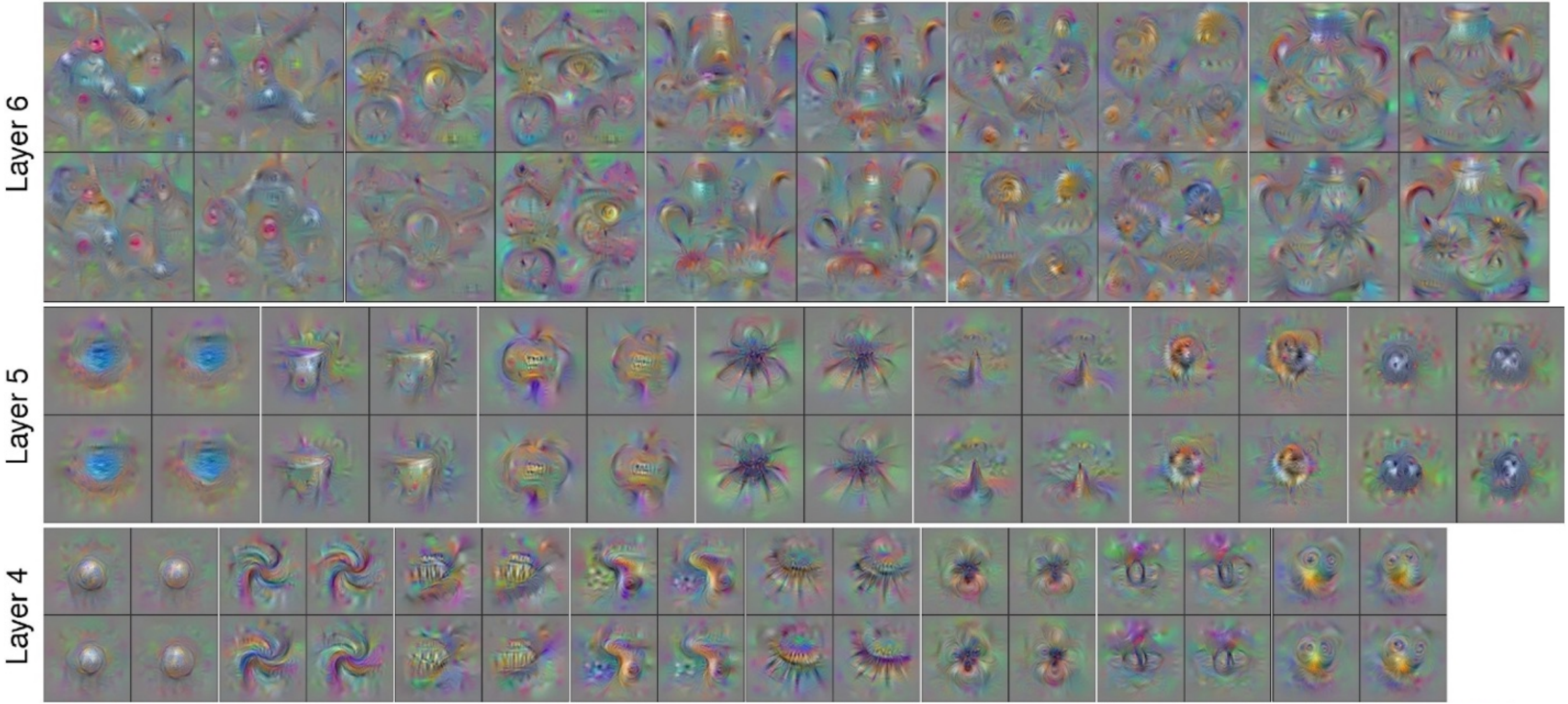
[Understanding Neural Networks Through Deep Visualization, Yosinski et al. , 2015]

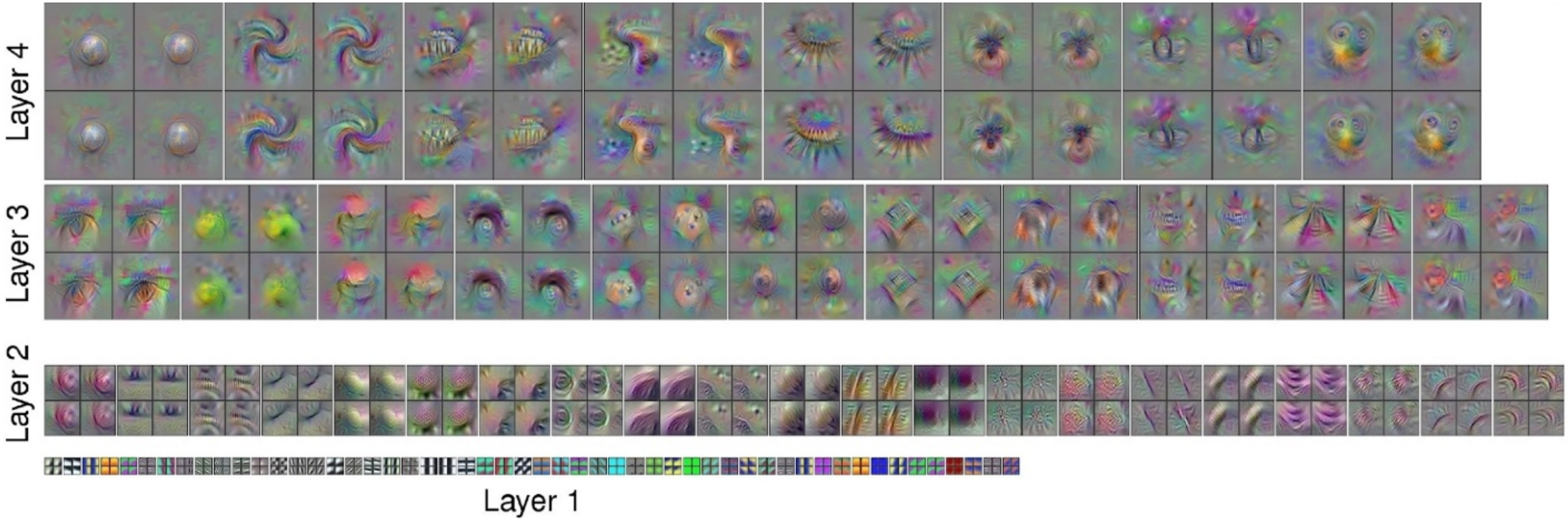
http://yosinski.com/deepvis



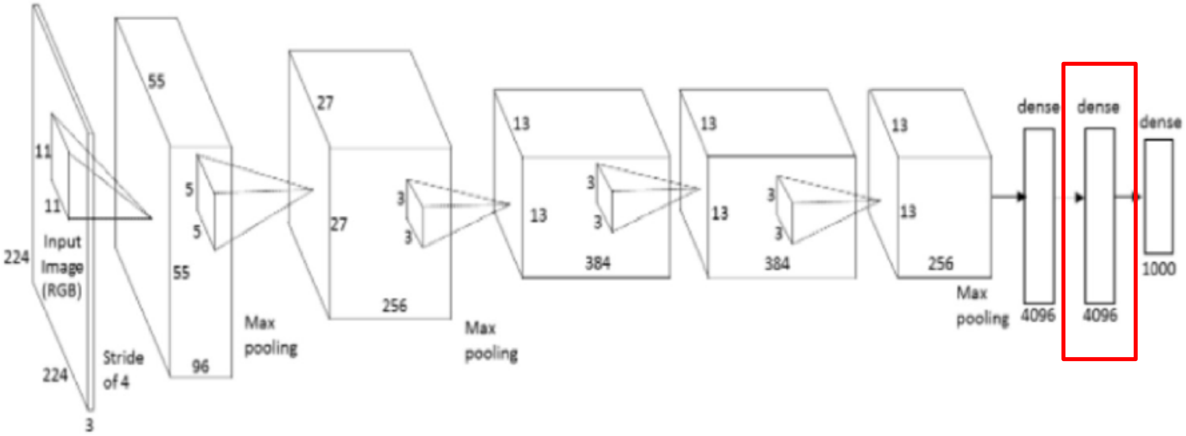


Layer 8
Pirate Ship
Rocking Chair
Teddy Bear
Windsor Tie
Pitcher
Layer 7


Layer 6
Layer 5
Layer 4


Layer 4
Layer 3
Layer 2
Layer 1

Question: Given a CNN code, is it possible to reconstruct the original image?



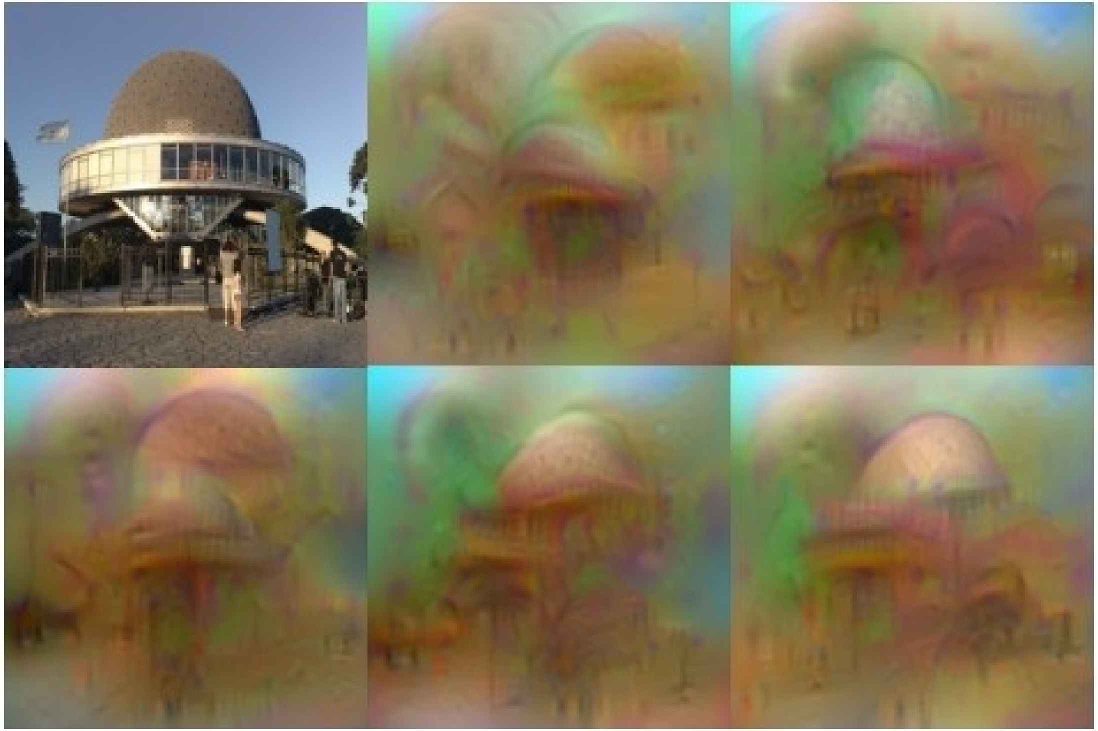
Find an image such that:
- Its code is similar to a given code
- It "looks natural" (image prior regularization)

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

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

Understanding Deep Image Representations by Inverting Them [Mahendran and Vedaldi, 2014]

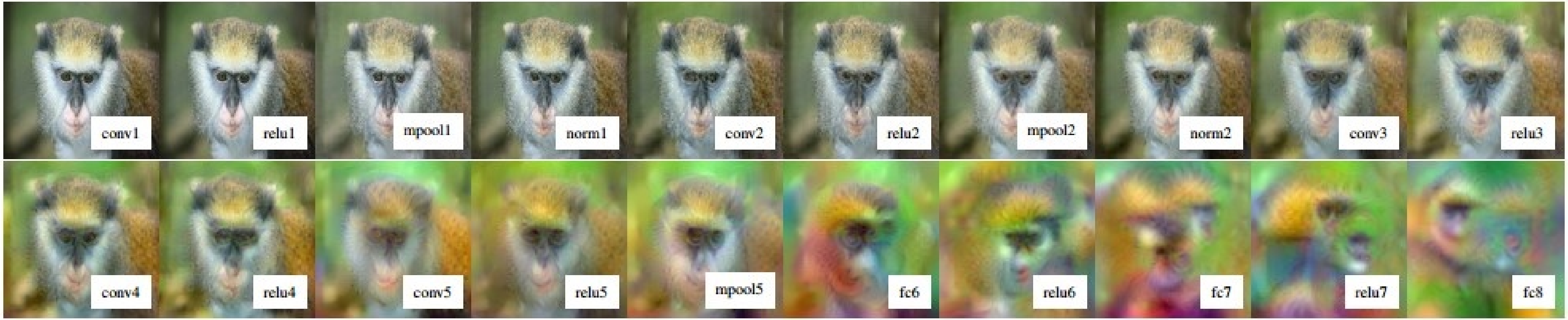
original image



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

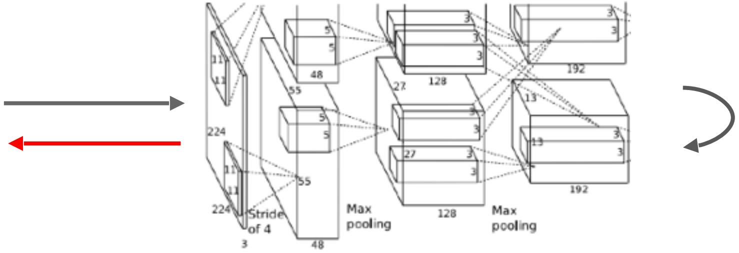
Reconstructions from the representation after last last pooling layer (immediately before the first Fully Connected layer)

Reconstructions from intermediate layers



DeepDream: Amplify existing features

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

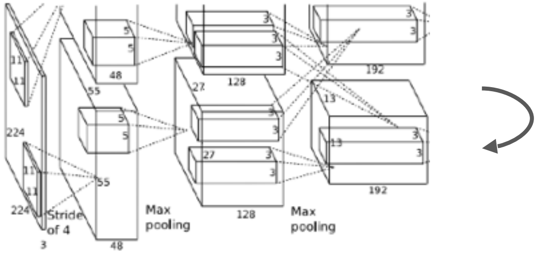


Choose an image and a layer in a CNN; repeat:
1. Forward: compute activations at chosen layer
2. Set gradient of chosen layer *equal to its activation*
3. Backward: Compute gradient on image
4. Update image

Mordvintsev, Olah, and Tyka, "Inceptionism: Going Deeper into Neural Networks", Google Research Blog.

DeepDream: Amplify existing 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:

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

Mordvintsev, Olah, and Tyka, "Inceptionism: Going Deeper into Neural Networks", Google Research Blog. Images are licensed under CC-BY 4.0

DeepDream: Amplify existing features

```
def objective_L2(dst):
    dst.diff[:] = dst.data

def make_step(net, step_size=1.5, end='inception_4c/output',
              jitter=32, clip=True, objective=objective_L2):
    '''Basic gradient ascent step.'''

    src = net.blobs['data'] # input image is stored in Net's 'data' blob
    dst = net.blobs[end]

    ox, oy = np.random.randint(-jitter, jitter+1, 2)
    src.data[0] = np.roll(np.roll(src.data[0], ox, -1), oy, -2) # apply jitter shift

    net.forward(end=end)
    objective(dst)  # specify the optimization objective
    net.backward(start=end)
    g = src.diff[0]
    # apply normalized ascent step to the input image
    src.data[:] += step_size/np.abs(g).mean() * g

    src.data[0] = np.roll(np.roll(src.data[0], -ox, -1), -oy, -2) # unshift image

    if clip:
        bias = net.transformer.mean['data']
        src.data[:] = np.clip(src.data, -bias, 255-bias)
```

[Code](#) is very simple but it uses a couple tricks:

DeepDream: Amplify existing features

```
def objective_L2(dst):
    dst.diff[:] = dst.data

def make_step(net, step_size=1.5, end='inception_4c/output',
              jitter=32, clip=True, objective=objective_L2):
    '''Basic gradient ascent step.'''

    src = net.blobs['data'] # input image is stored in Net's 'data' blob
    dst = net.blobs[end]

    ox, oy = np.random.randint(-jitter, jitter+1, 2)
    src.data[0] = np.roll(np.roll(src.data[0], ox, -1), oy, -2) # apply jitter shift

    net.forward(end=end)
    objective(dst)  # specify the optimization objective
    net.backward(start=end)
    g = src.diff[0]
    # apply normalized ascent step to the input image
    src.data[:] += step_size/np.abs(g).mean() * g

    src.data[0] = np.roll(np.roll(src.data[0], -ox, -1), -oy, -2) # unshift image

    if clip:
        bias = net.transformer.mean['data']
        src.data[:] = np.clip(src.data, -bias, 255-bias)
```

Code is very simple but it uses a couple tricks:

Jitter image

DeepDream: Amplify existing features

```
def objective_L2(dst):
    dst.diff[:] = dst.data

def make_step(net, step_size=1.5, end='inception_4c/output',
              jitter=32, clip=True, objective=objective_L2):
    '''Basic gradient ascent step.'''

    src = net.blobs['data'] # input image is stored in Net's 'data' blob
    dst = net.blobs[end]

    ox, oy = np.random.randint(-jitter, jitter+1, 2)
    src.data[0] = np.roll(np.roll(src.data[0], ox, -1), oy, -2) # apply jitter shift

    net.forward(end=end)
    objective(dst)  # specify the optimization objective
    net.backward(start=end)
    g = src.diff[0]
    # apply normalized ascent step to the input image
    src.data[:] += step_size/np.abs(g).mean() * g

    src.data[0] = np.roll(np.roll(src.data[0], -ox, -1), -oy, -2) # unshift image

    if clip:
        bias = net.transformer.mean['data']
        src.data[:] = np.clip(src.data, -bias, 255-bias)
```

Code is very simple but it uses a couple tricks:

Jitter image

L1 Normalize gradients

DeepDream: Amplify existing features

```
def objective_L2(dst):
    dst.diff[:] = dst.data

def make_step(net, step_size=1.5, end='inception_4c/output',
              jitter=32, clip=True, objective=objective_L2):
    '''Basic gradient ascent step.'''

    src = net.blobs['data'] # input image is stored in Net's 'data' blob
    dst = net.blobs[end]

    ox, oy = np.random.randint(-jitter, jitter+1, 2)
    src.data[0] = np.roll(np.roll(src.data[0], ox, -1), oy, -2) # apply jitter shift

    net.forward(end=end)
    objective(dst)  # specify the optimization objective
    net.backward(start=end)
    g = src.diff[0]
    # apply normalized ascent step to the input image
    src.data[:] += step_size/np.abs(g).mean() * g

    src.data[0] = np.roll(np.roll(src.data[0], -ox, -1), -oy, -2) # unshift image

    if clip:
        bias = net.transformer.mean['data']
        src.data[:] = np.clip(src.data, -bias, 255-bias)
```

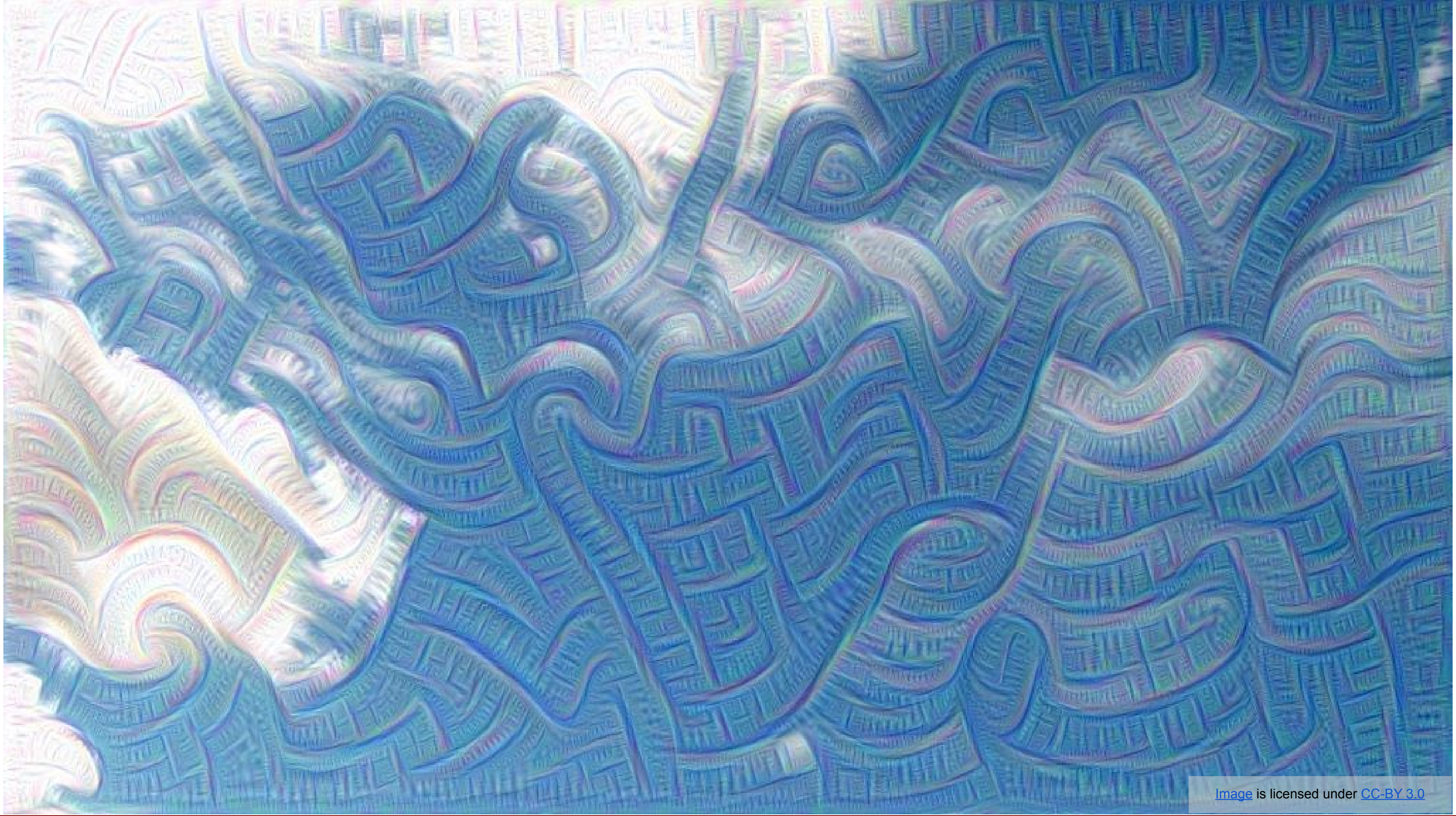
Code is very simple but it uses a couple tricks:

Jitter image

L1 Normalize gradients

Clip pixel values

Also uses multiscale processing for a fractal effect (not shown)



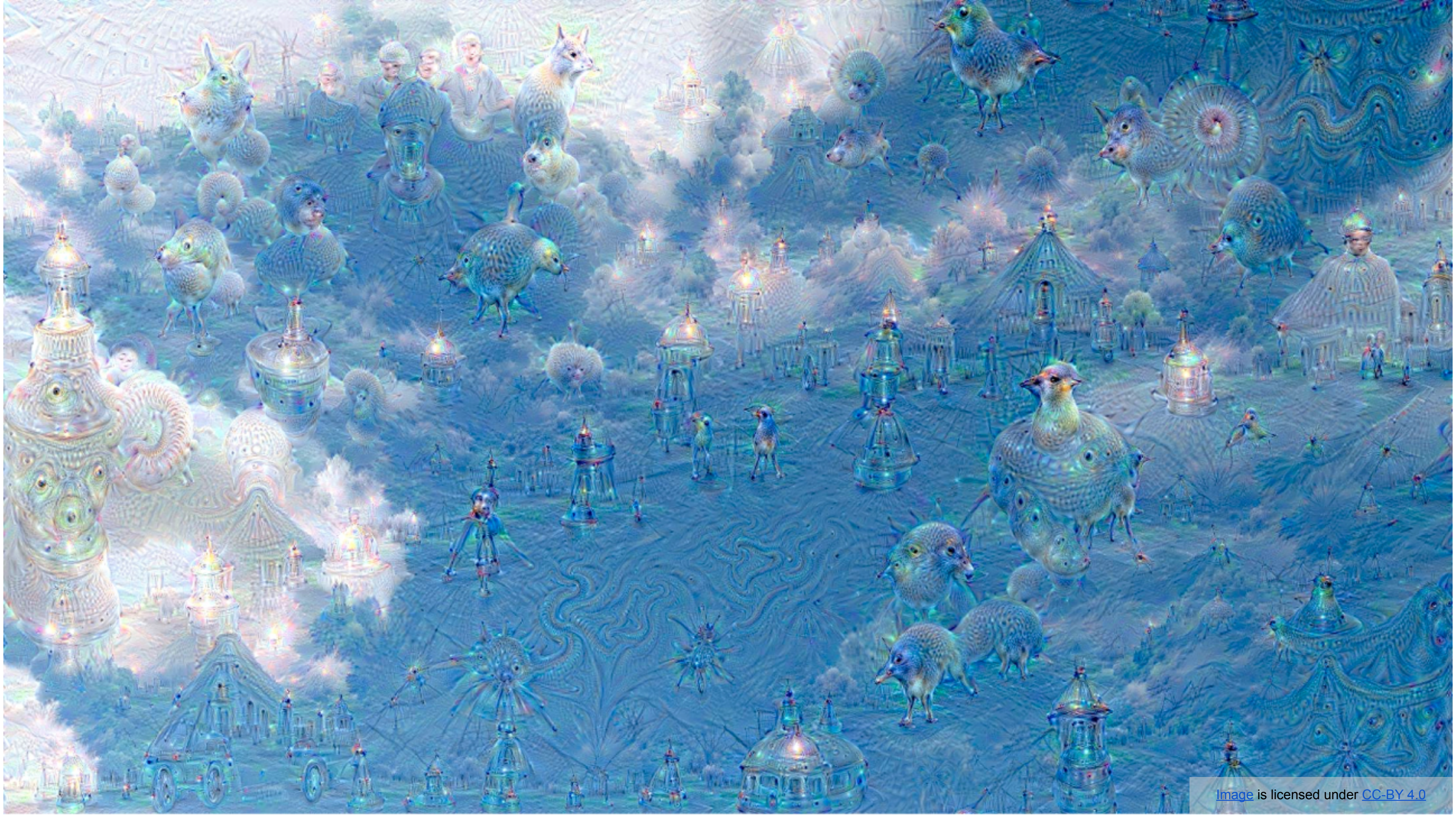
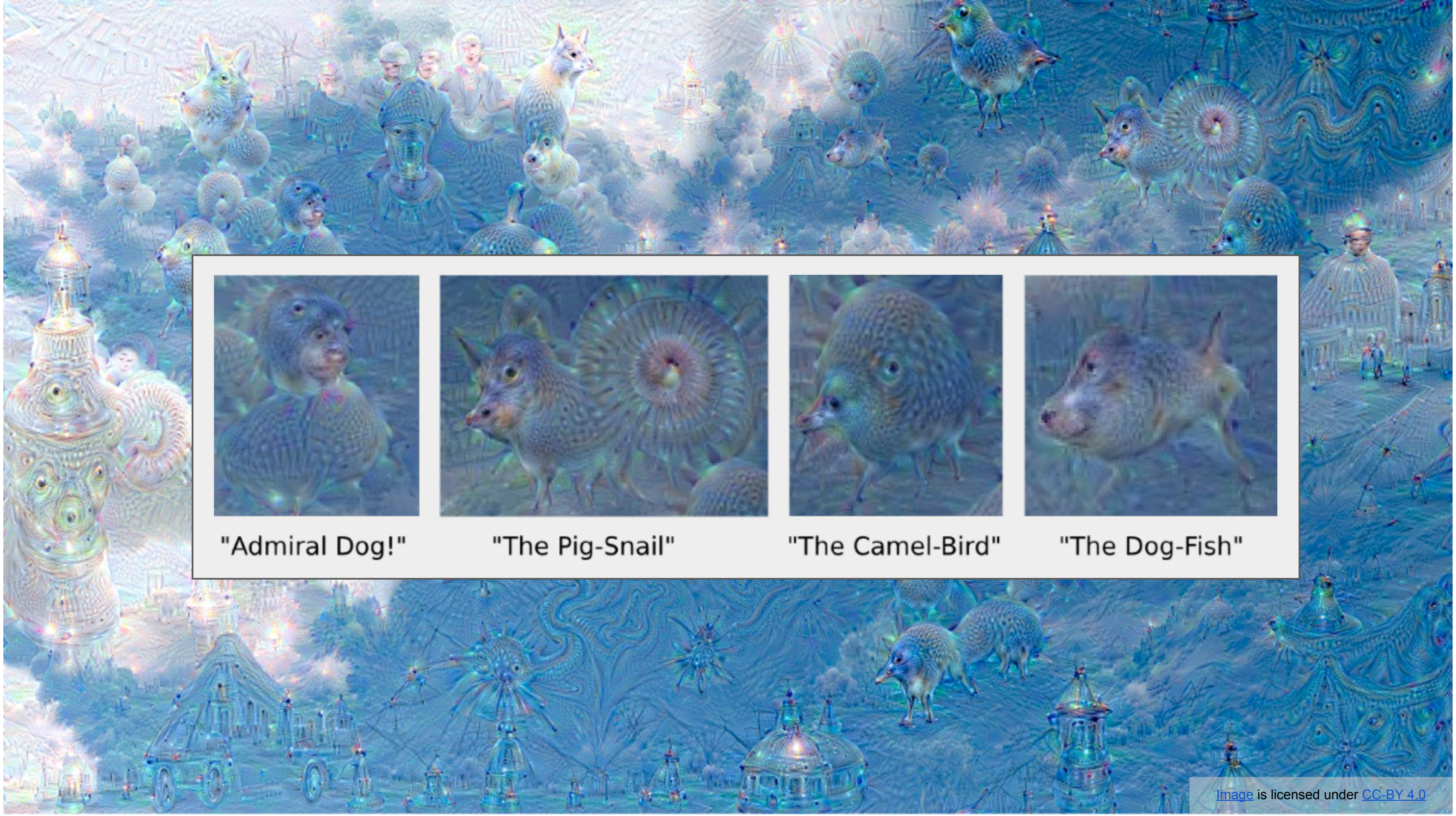


Image is licensed under CC-BY 4.0


"Admiral Dog!"
"The Pig-Snail"
"The Camel-Bird"
"The Dog-Fish"

Bonus videos

Deep Dream Grocery Trip
https://www.youtube.com/watch?v=DgPaCWJL7XI

Deep Dreaming Fear & Loathing in Las Vegas: the Great San Francisco Acid Wave
https://www.youtube.com/watch?v=oyxSerkkP4o