CS194/294-129: Designing, Visualizing and Understanding Deep Neural Networks

John Canny

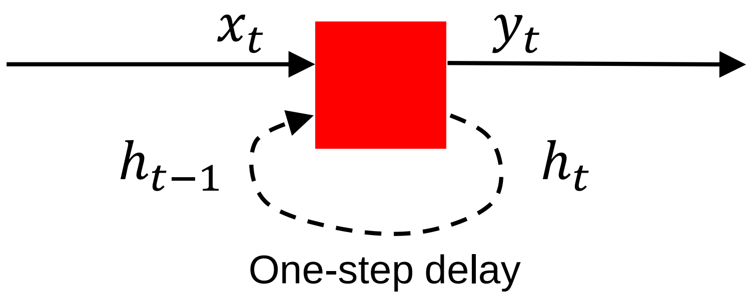
Spring 2018

Lecture 10: Visualization

Based on Notes by Andrej Karpathy

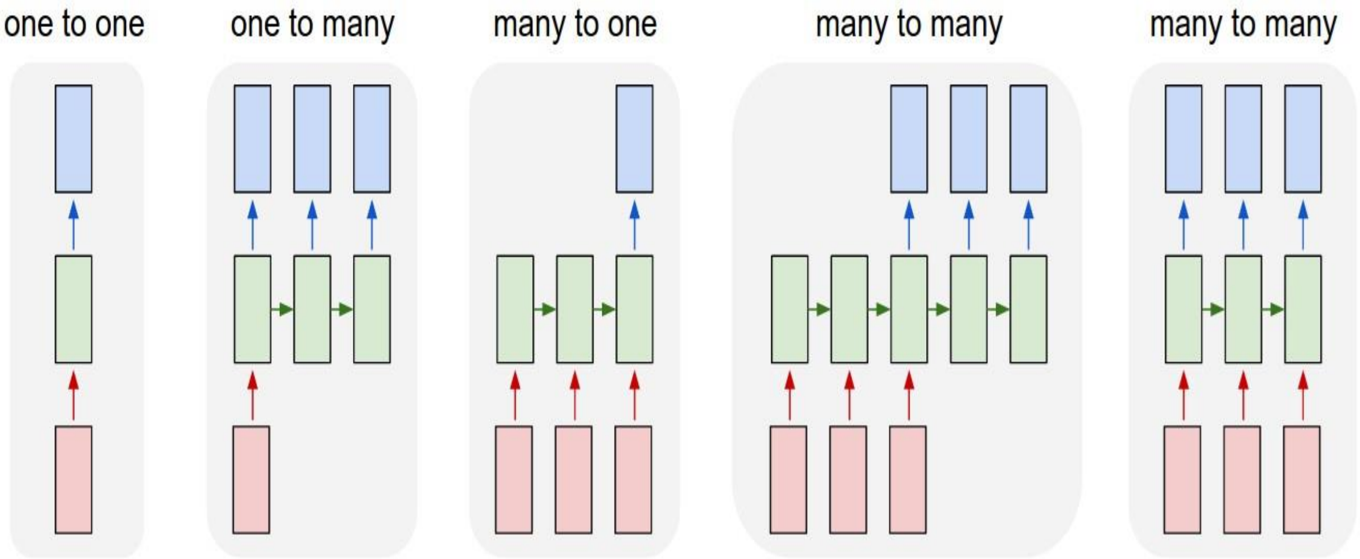
Last Time: Recurrent Neural Networks (RNNs)

Recurrent networks introduce cycles and a notion of time.

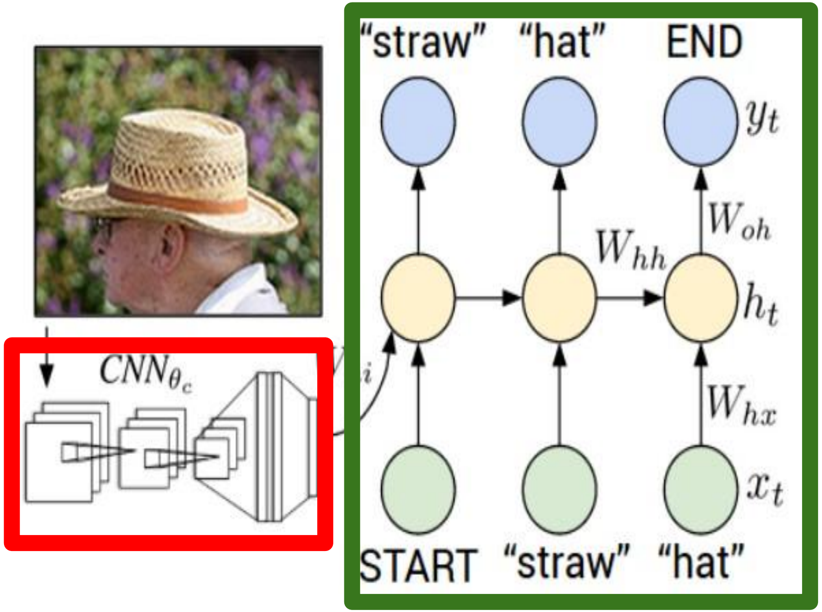


- They are designed to process sequences of data $x_1, \ldots, x_n$ and can produce sequences of outputs $y_1, \ldots, y_m$.

Last Time: Recurrent Designs:



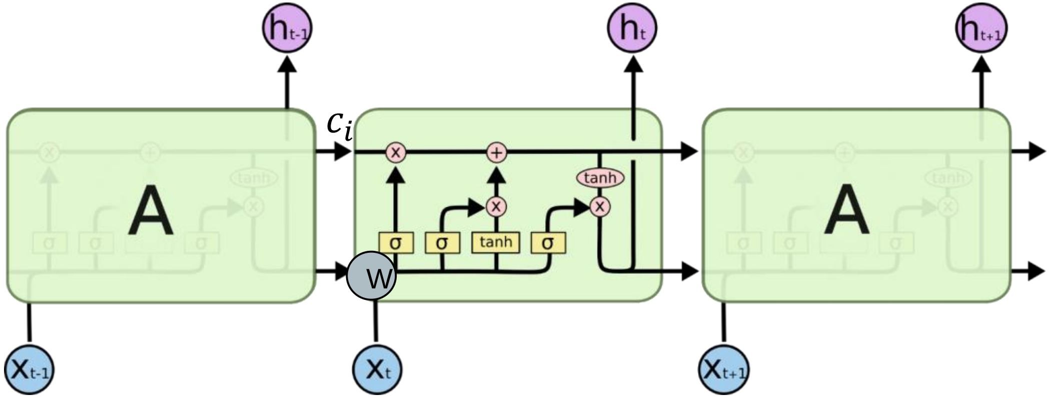
Last Time: Recurrent Neural Network Captioning



Convolutional Neural Network

Last Time: LSTMs

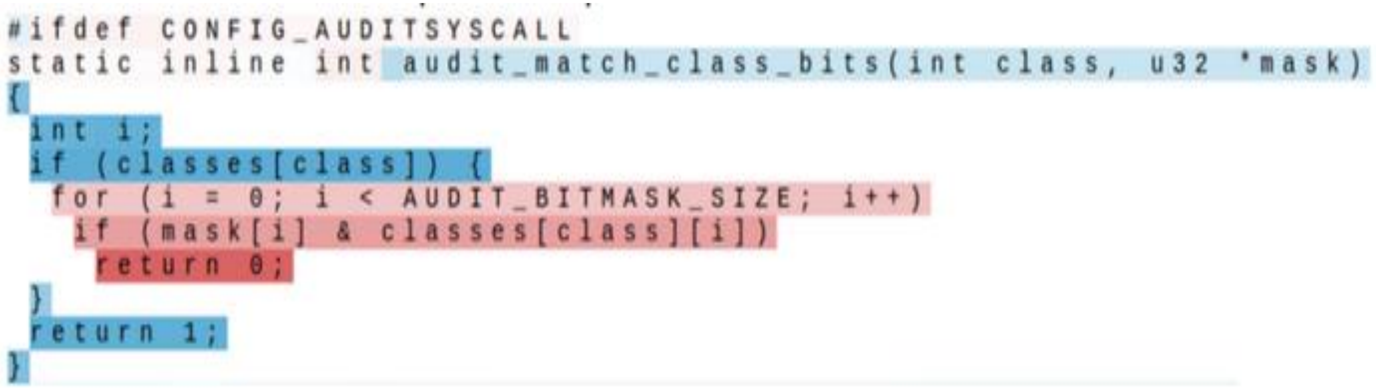
There are two recurrent nodes, c_i and h_i.
h_i plays the role of the output in the simple RNN, and is recurrent.
The the cell state c_i is the cell's *memory*, it undergoes no transform.
When we compose LSTMs into arrays, they look like this:



For stacked arrays, the hidden layers (h_i's) become the inputs (x_i's) for the layer above.

Figure courtesy Chris Olah http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Last Time: Interpreting LSTM cells



code depth cell

Based on cs231n by Fei-Fei Li & Andrej Karpathy & Justin Johnson

Midterm

Monday 5pm-6:30pm (Berkeley time)

- CS194-129: All in 105 Northgate (Here)
- CS294-129: Last names A-K in 105 Northgate
- CS294-129: Last names L-Z in 400 Cory (Hughes Room)

Closed-book, one double-sided sheet of notes

Visualizing Representations

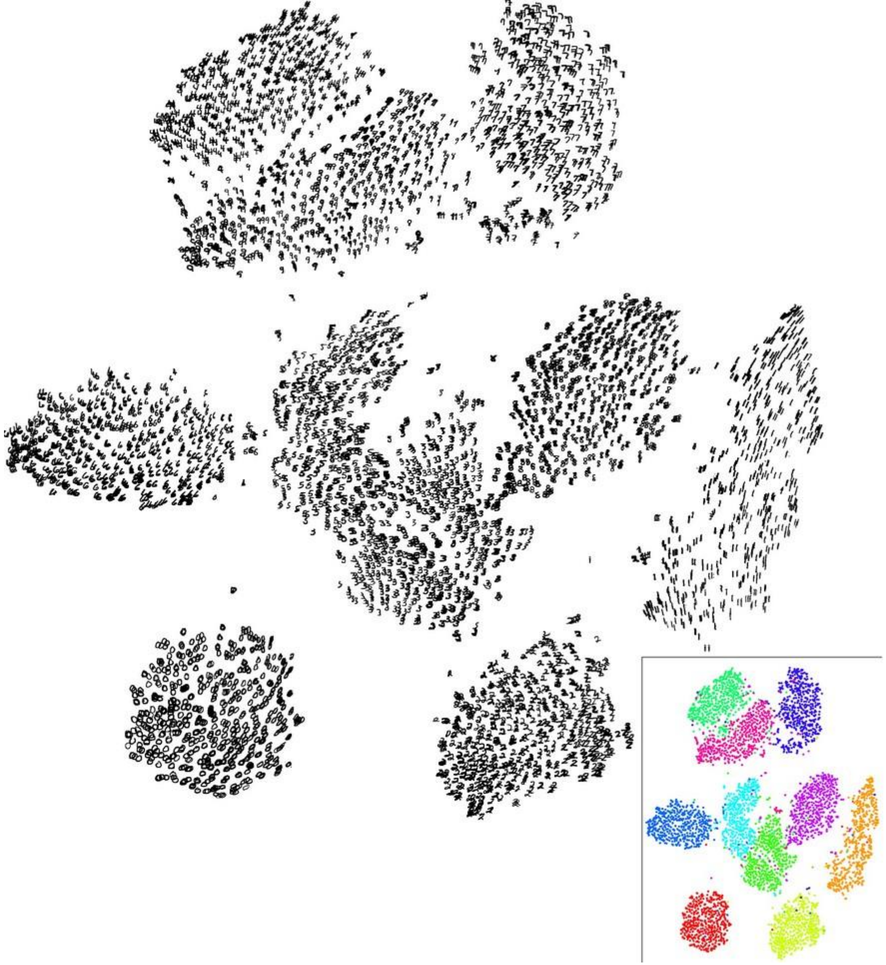
t-SNE visualization

[van der Maaten & Hinton]

Stochastic Neighbor Embedding:
Embed high-dimensional points so that locally, pairwise distances are conserved

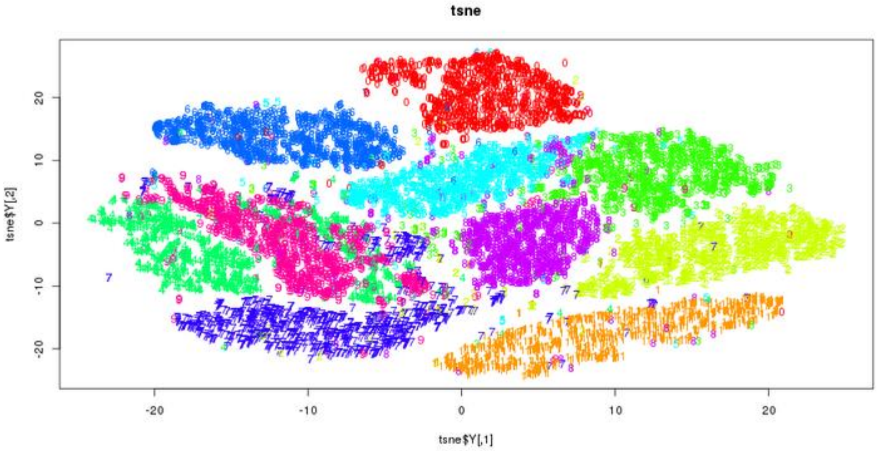
i.e. similar things end up in similar places. dissimilar things end up wherever

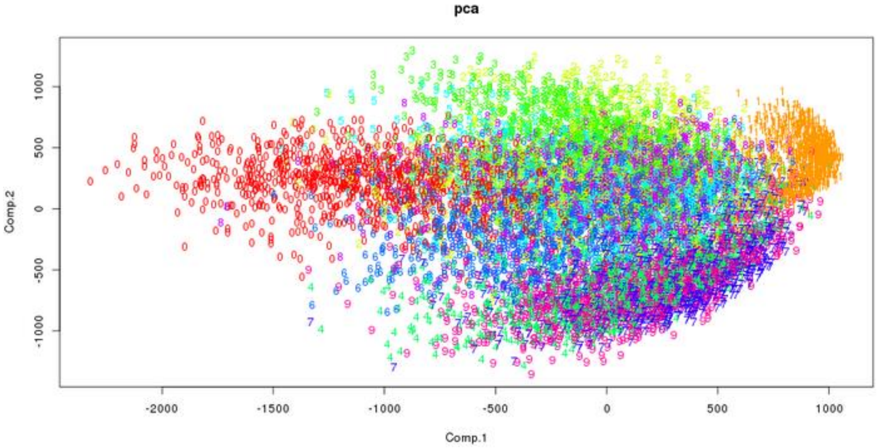
Right: Example embedding of MNIST digit images (0-9) in 2D



t-SNE Embeddings

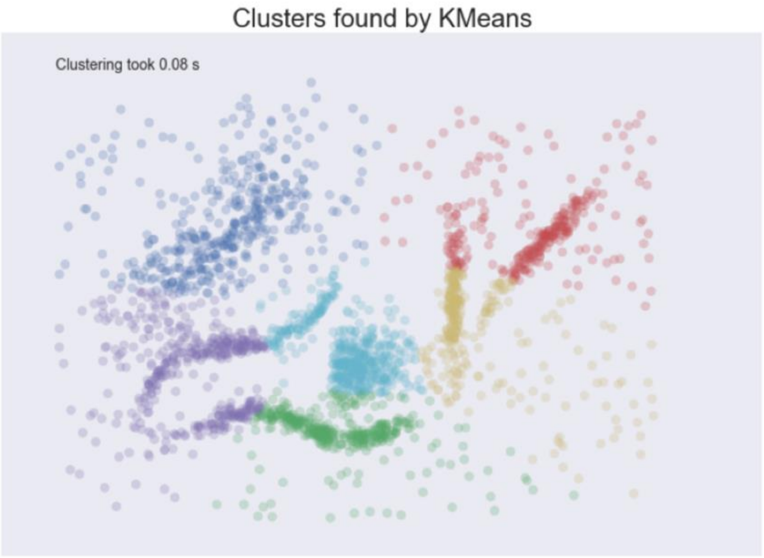
Generally does a better job of separating classes compared to PCA:

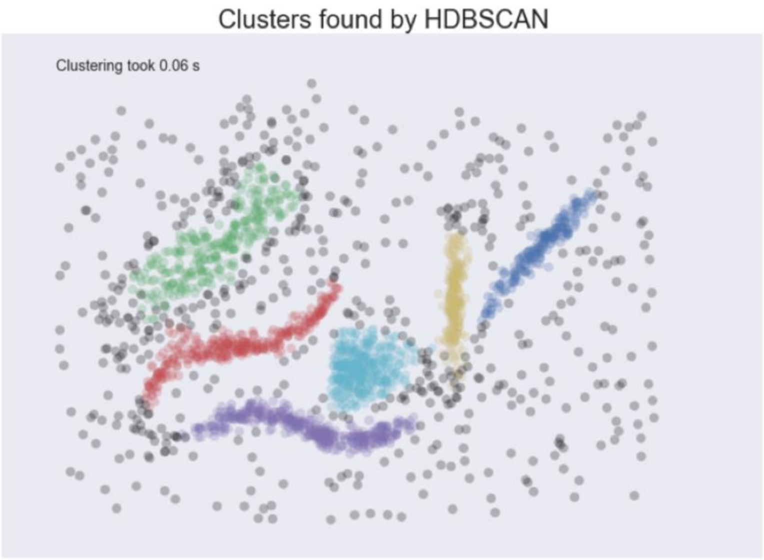




t-SNE Embeddings

A t-SNE embedding puts similar items close to each other in 2 or 3-D.
Its often useful to cluster the data in the embedding space.
Since the clusters are not “compact” (sphere-like), its best to use a density-based clustering like DBSCAN or HDBSCAN



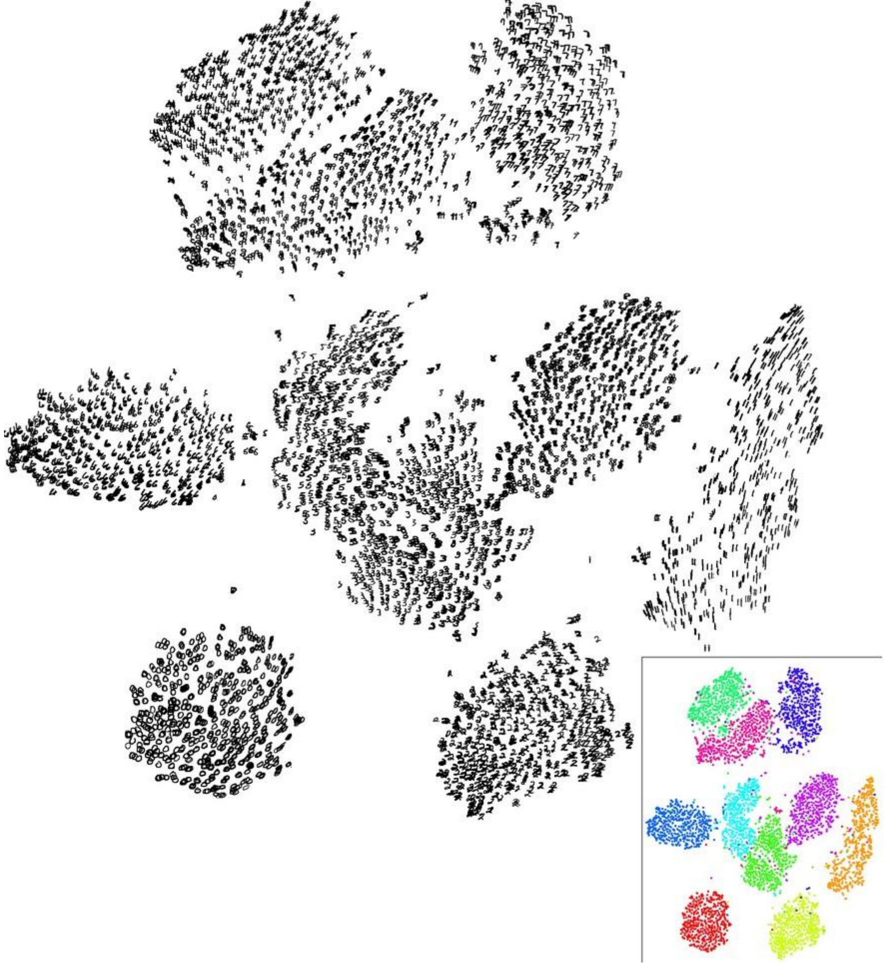


t-SNE Implementation

Fast Stochastic Neighbor Embedding
[van der Maaten 2013]

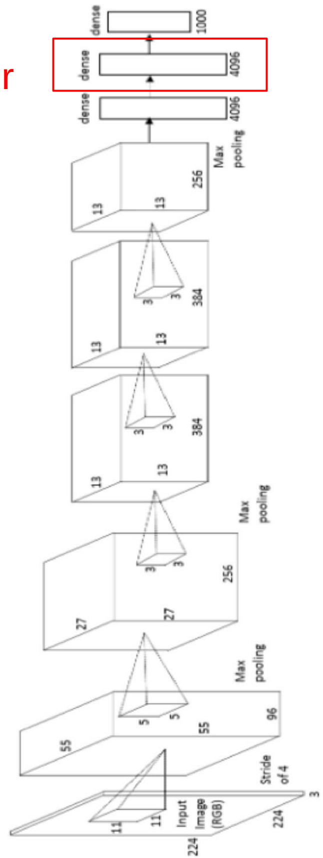
Aside: t-SNE is an iterative algorithm and expensive $O(N^2)$ for datasets with N points.

Its common to use an approximation (Barnes-Hut-SNE) which is $O(N \log N)$ and which can manage millions of points.



Visualizing Representations

fc7 layer



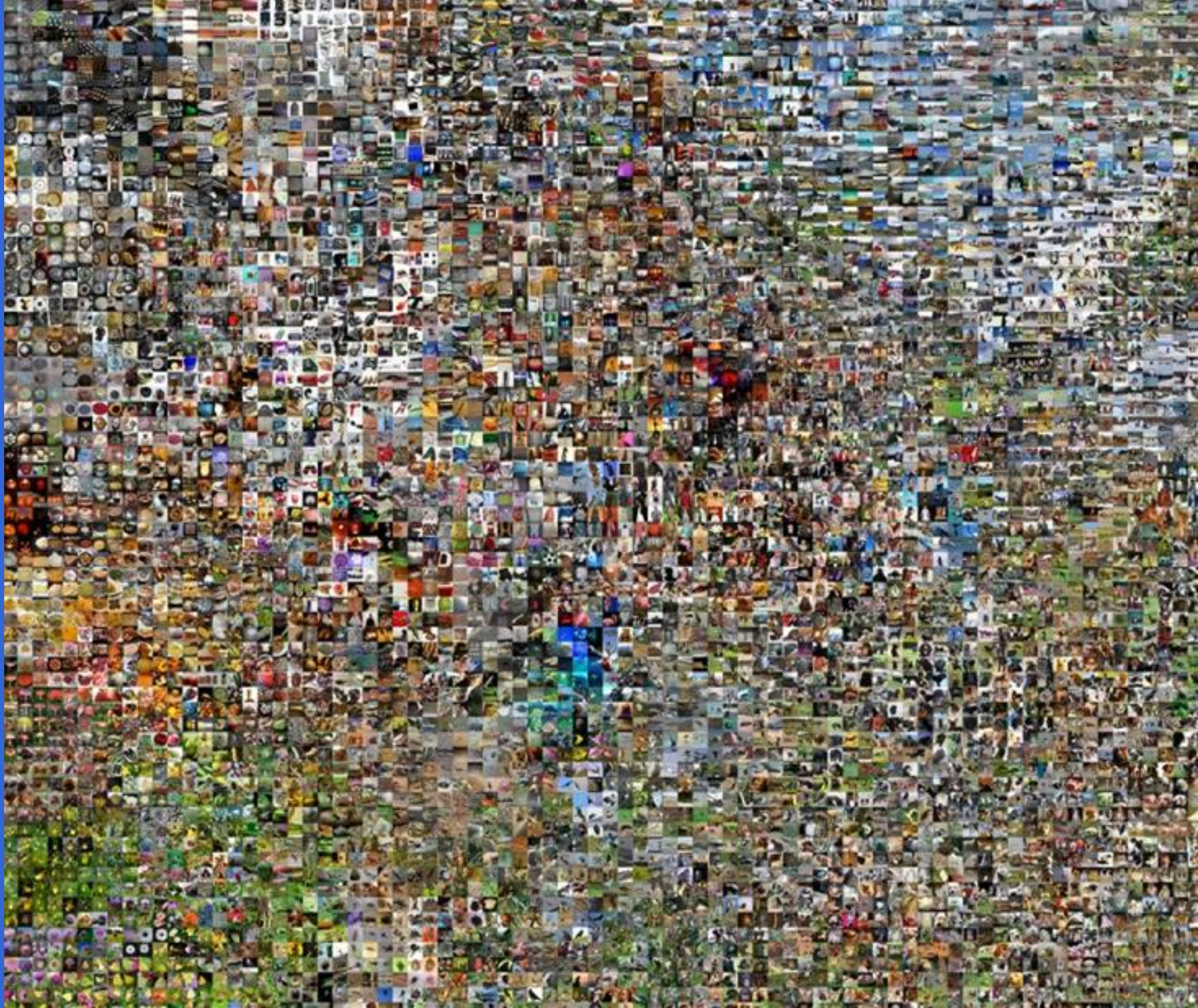
4096-dimensional “code” for an image
(layer immediately before the classifier)

can collect the code for many images

t-SNE visualization:

two images are placed nearby if their CNN codes are close. See more:

http://cs.stanford.edu/people/karpathy/cnnembed/

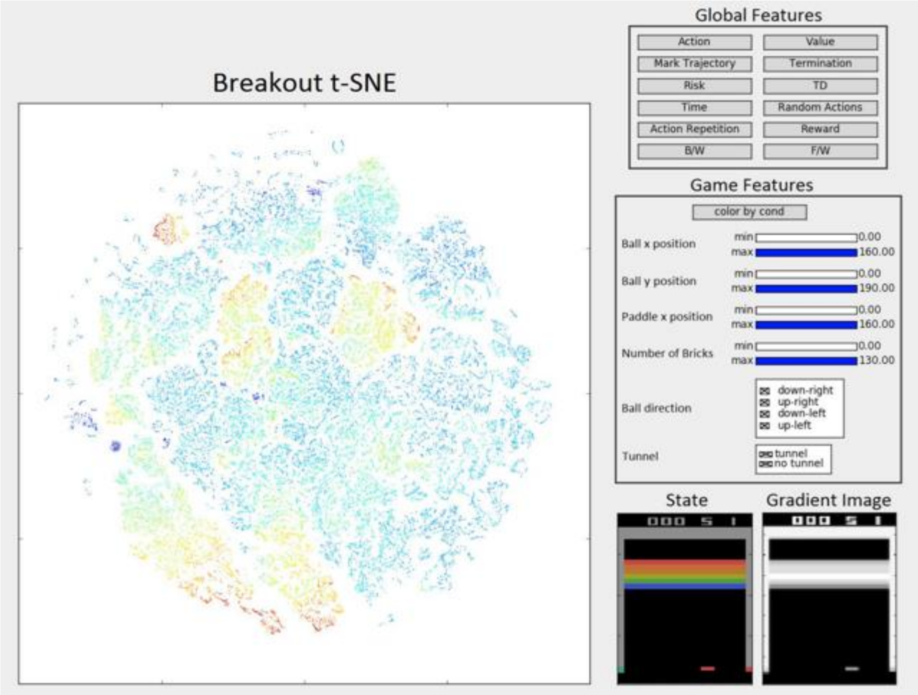


Graying the black box: Understanding DQNs

Zahavy, Zrihem, Mannor 2016

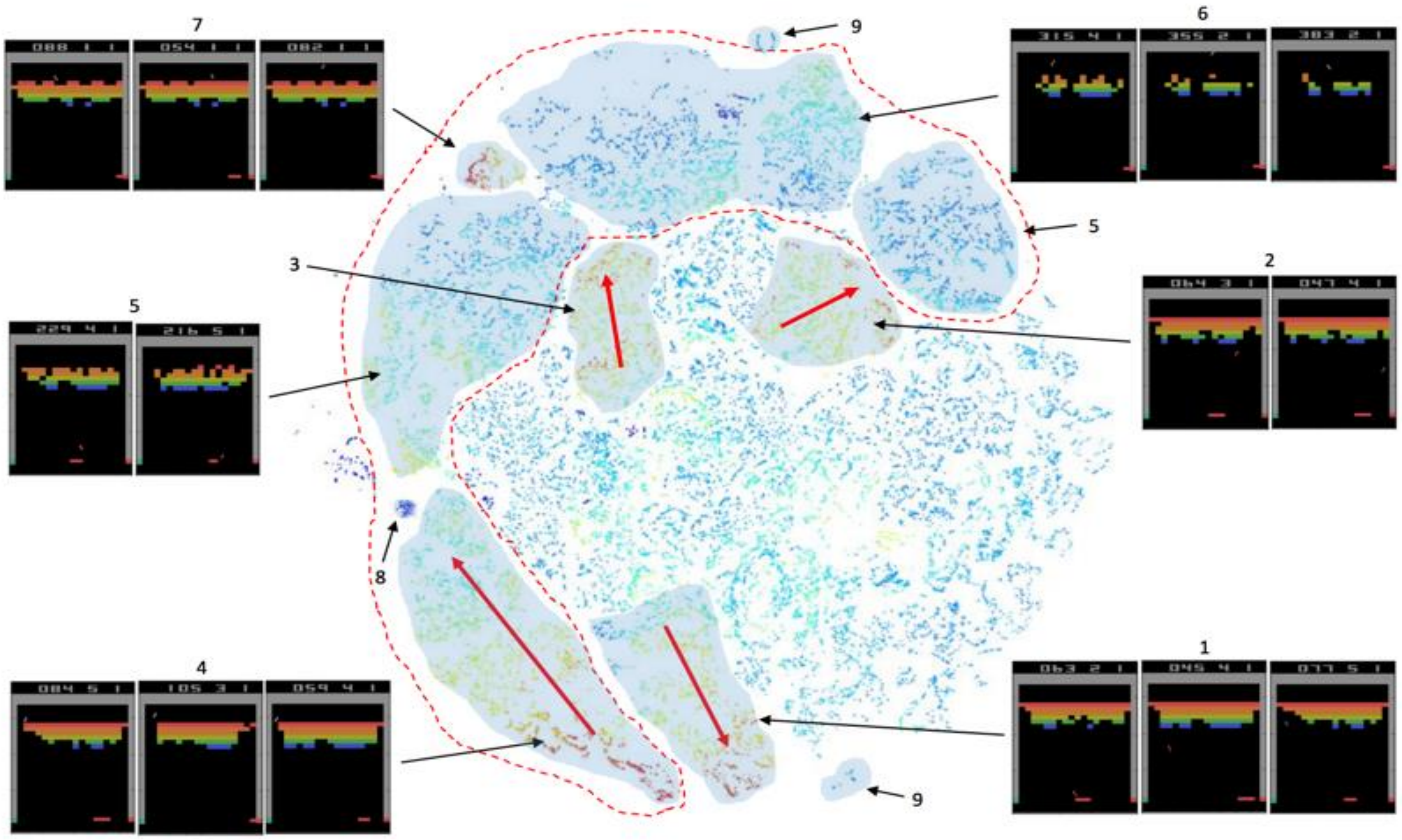
Playing Atari with Deep Reinforcement Learning, Mnih et al. 2013

Graying the black box: Understanding DQNs
Zahavy, Zrihem, Mannor 2016



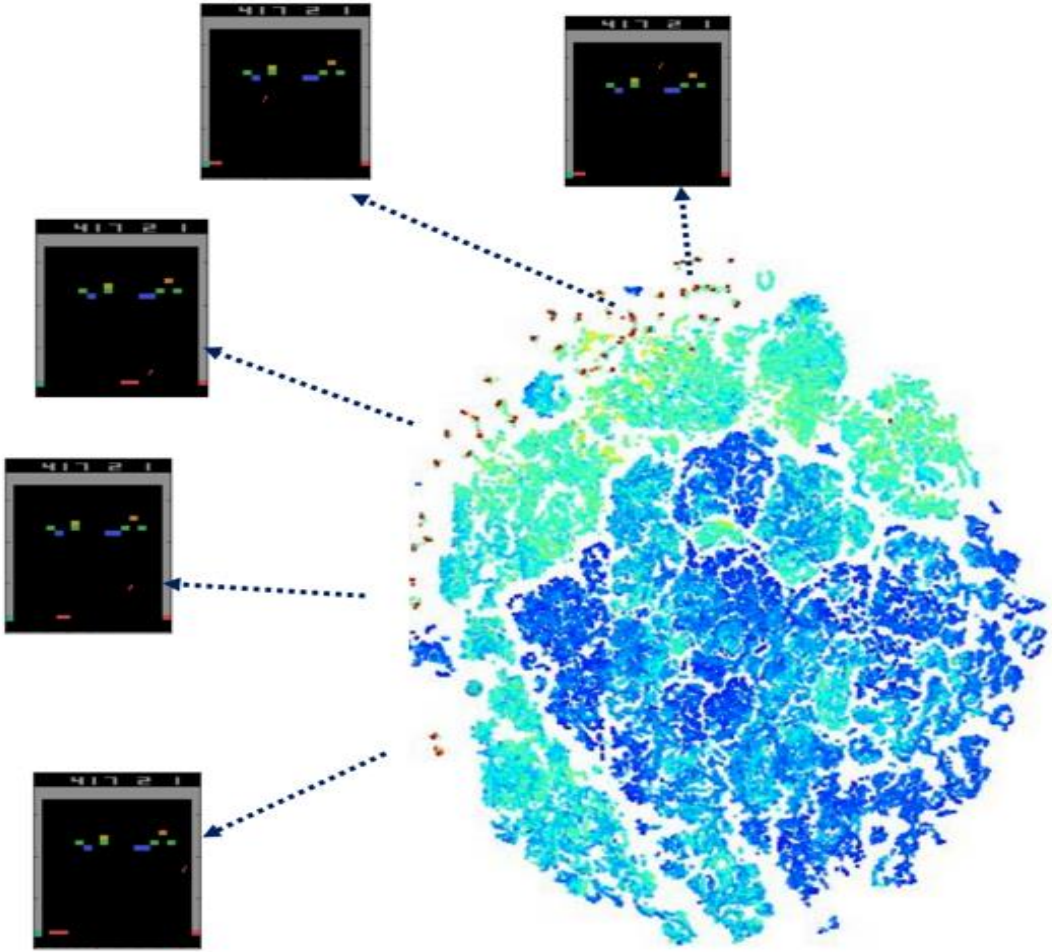
The embedding shows clustering of the *activations of the agent's policy network* for different frames of breakout.

Visualizing the network's "strategy"



Identify “policy bugs”: state clusters where the agent spends a very long time

(e.g. failing to hit the last few blocks over and over again for a very long time)



Understanding Learned CNN Models

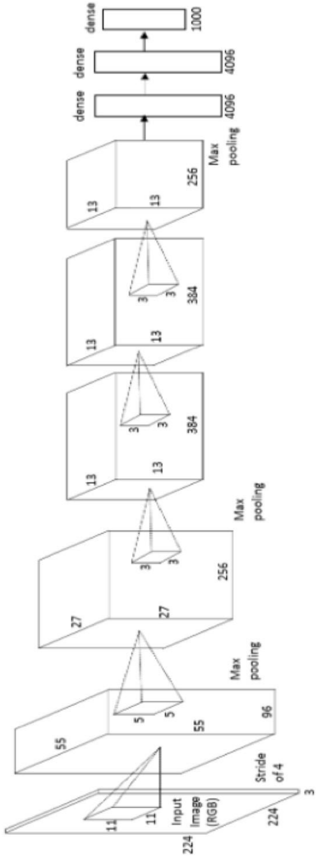
What do ConvNets learn?

Multiple lines of attack:

- Visualize patches that maximally activate neurons
- Visualize the weights
- Visualize the representation space (e.g. with t-SNE)
- Occlusion experiments
- Deconv approaches (single backward pass)
- Optimization over image approaches (optimization)

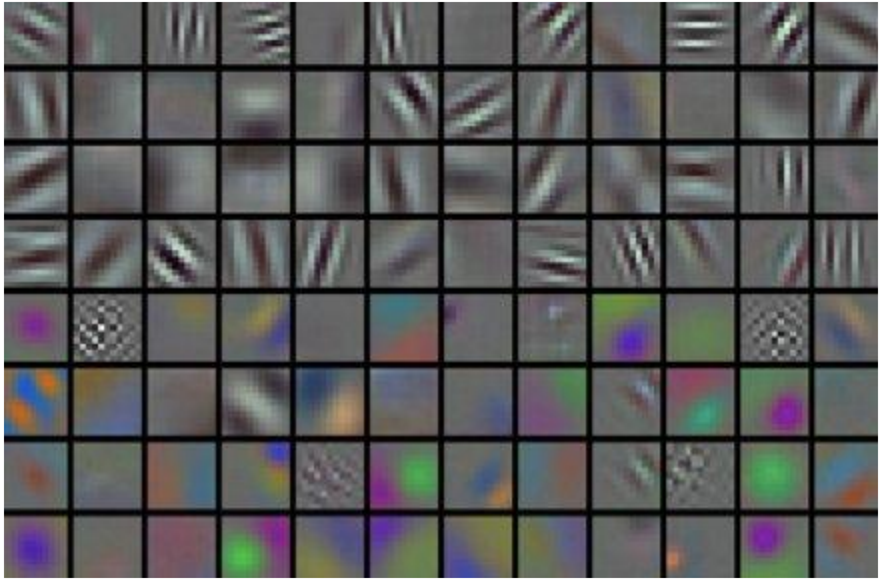
AlexNet

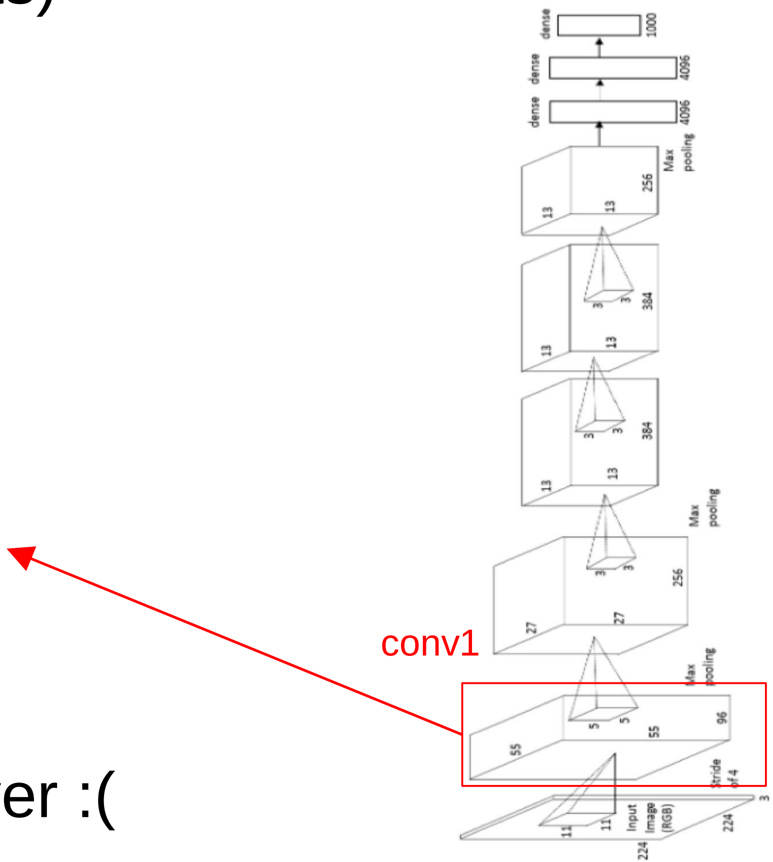
Figure 4: Top regions for six pool$_5$ units. Receptive fields and activation values are drawn in white. Some units are aligned to concepts, such as people (row 1) or text (4). Other units capture texture and material properties, such as dot arrays (2) and specular reflections (6).



Rich feature hierarchies for accurate object detection and semantic segmentation – (R-CNN paper)
[Girshick, Donahue, Darrell, Malik, 2014]

Visualize the filters/kernels (raw weights)





only interpretable on the first layer :(

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)

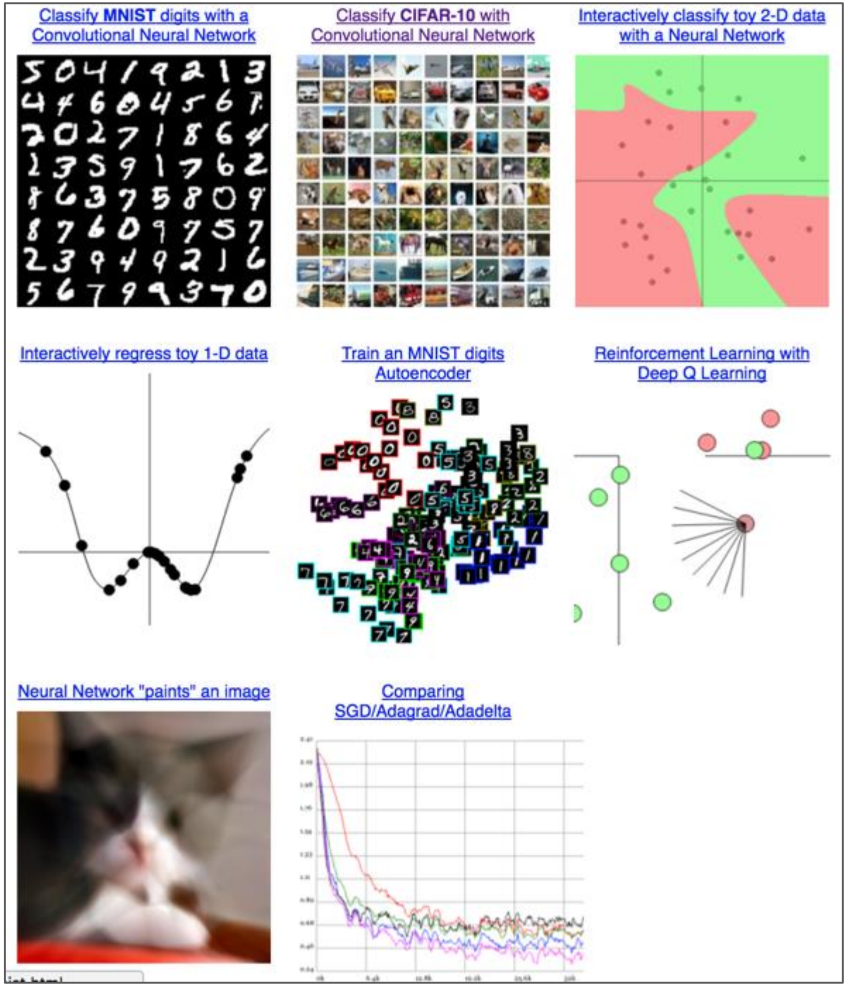
layer 1 weights

layer 2 weights

layer 3 weights

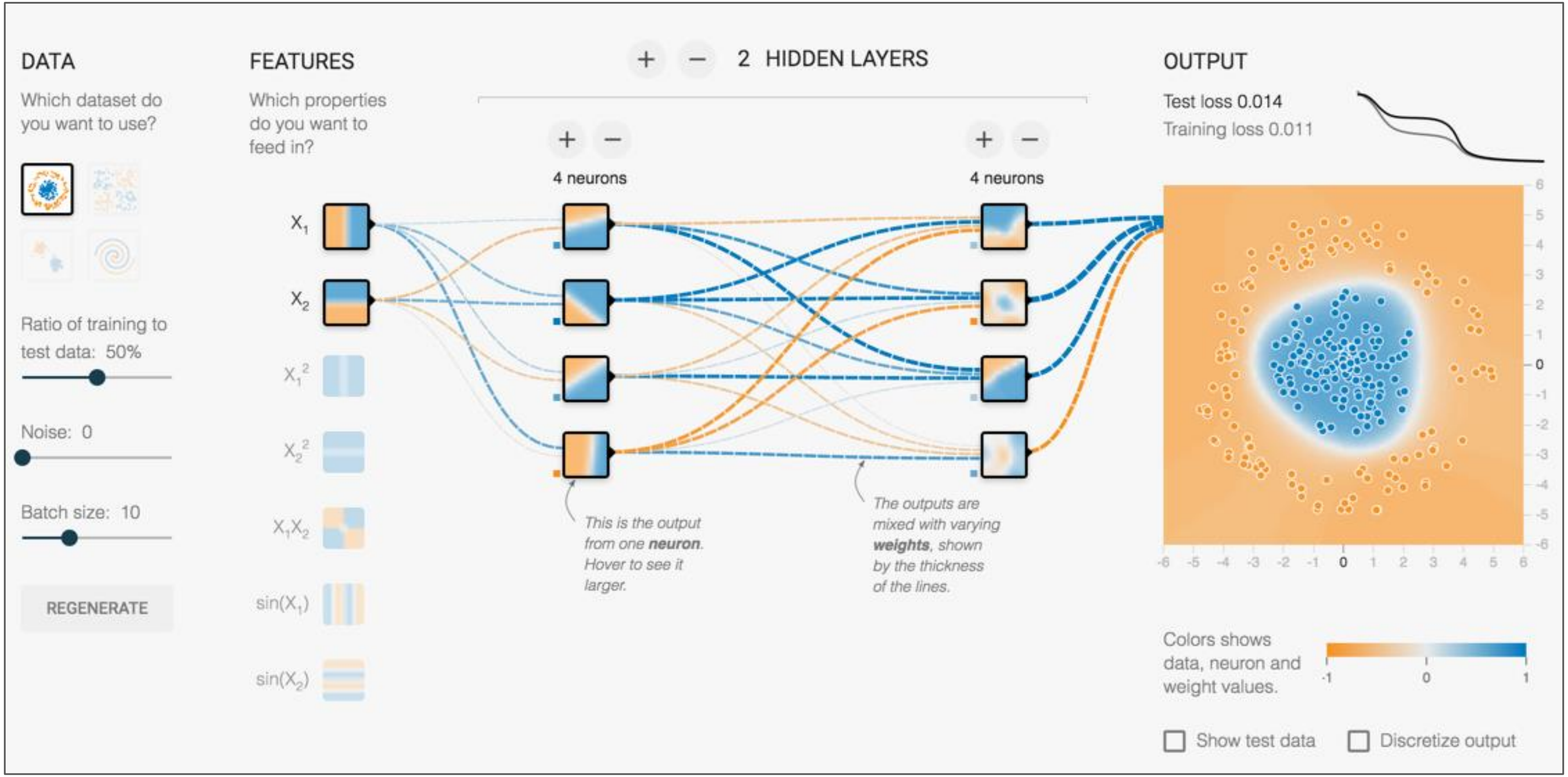
Javascript demos: ConvNetJS

https://cs.stanford.edu/people/karpathy/convnetjs/



Javascript demos: TensorFlow Playground

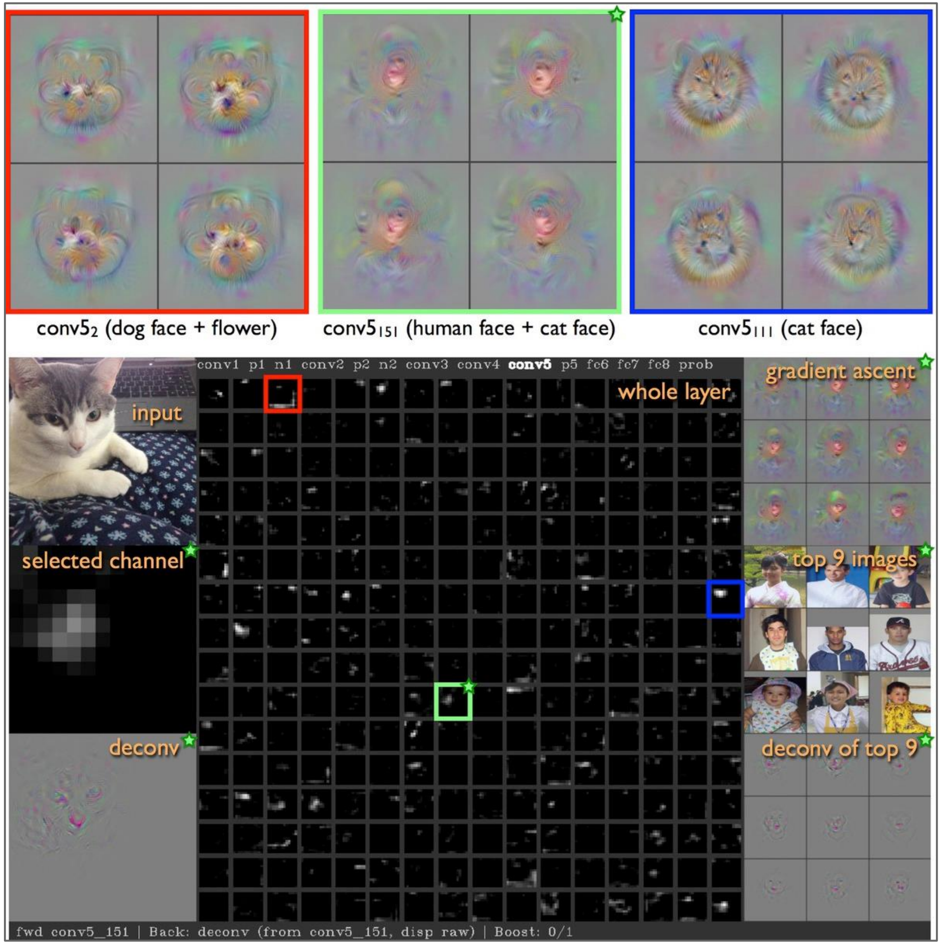
http://playground.tensorflow.org/



ConvNet DeepVis

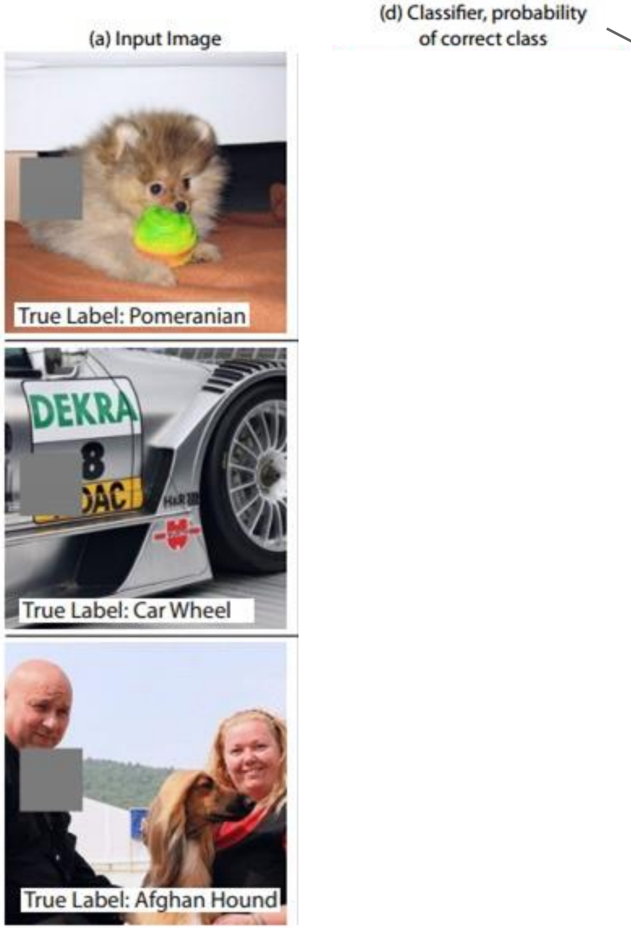
http://yosinski.com/deepvis

Not Javascript :(



YouTube video
https://www.youtube.com/watch?v=AgkfIQ4IGaM
(4min)

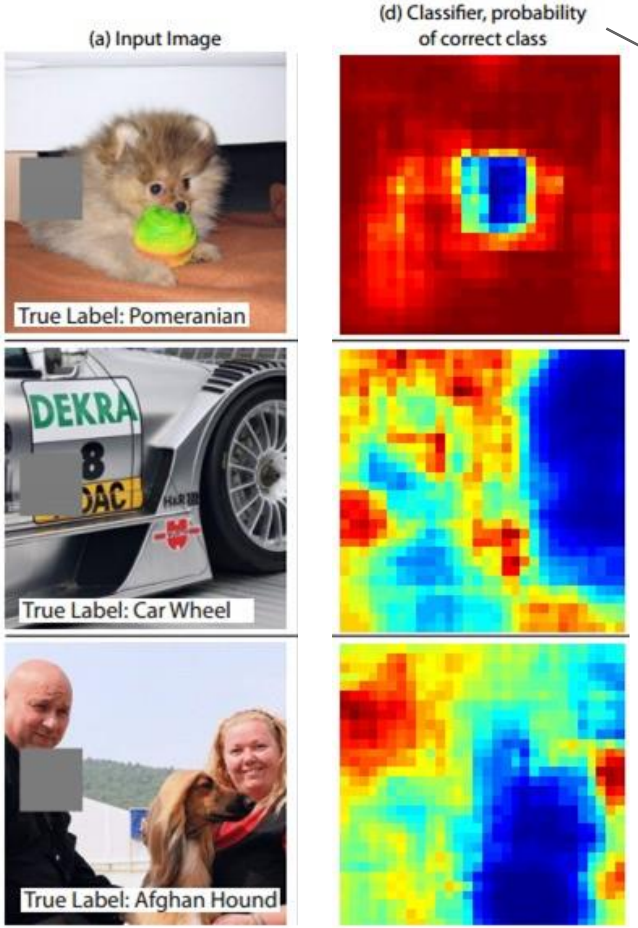
Occlusion experiments
[Zeiler & Fergus 2013]



(as a function of the position of the gray square in the original image)

Occlusion experiments

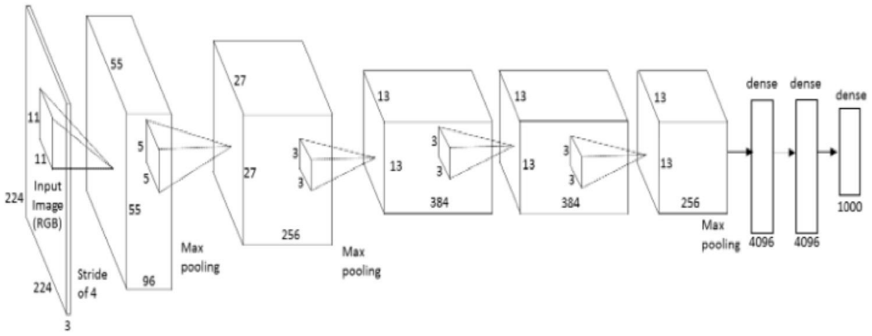
[Zeiler & Fergus 2013]



(as a function of the position of the gray square in the original image)

Saliency approaches (Network Centric)

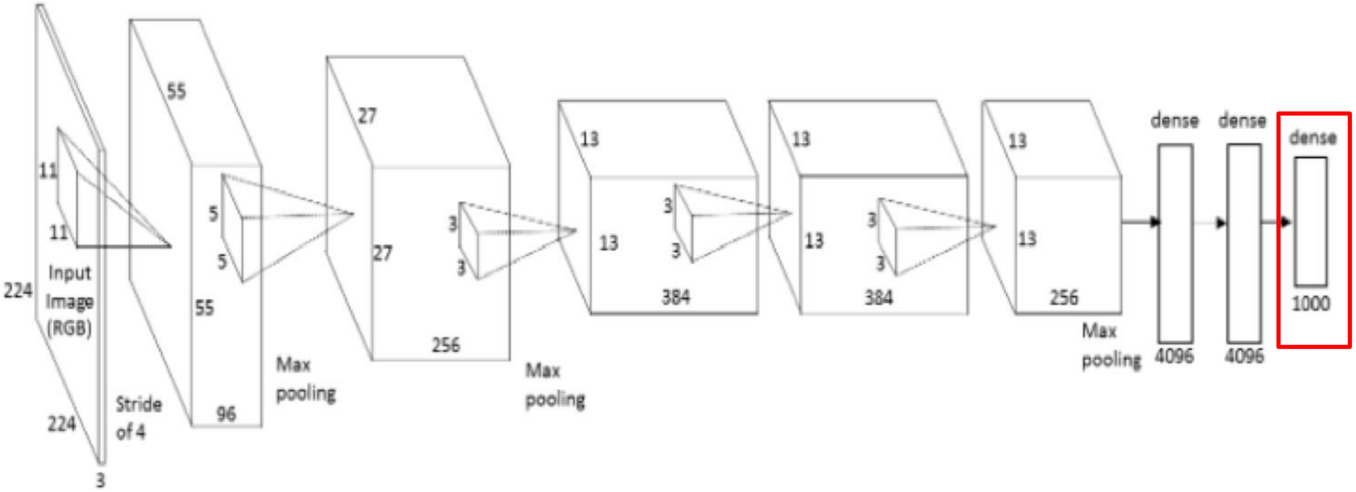
?



Q: how can we compute the pixels that have the most effect on the class score?

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

Optimization to Image

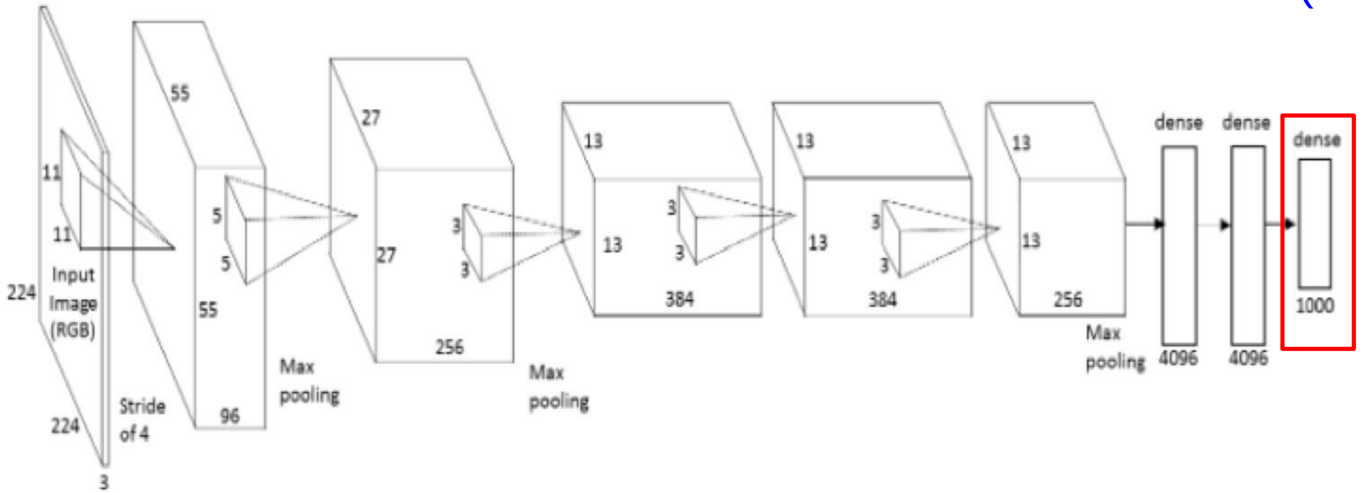


i.e. generate an image that maximizes the class score

Optimization to Image

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

score for class c (before Softmax)

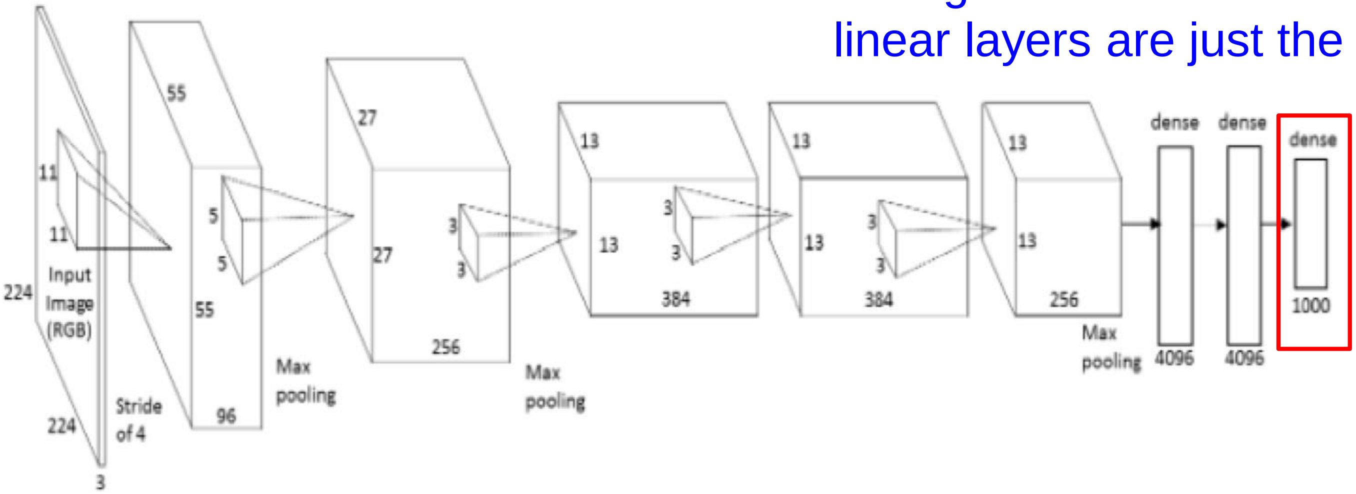


Generate an image that maximizes the class score

Optimization to Image

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

L2 regularization: Saliency maps for linear layers are just the layer weights.

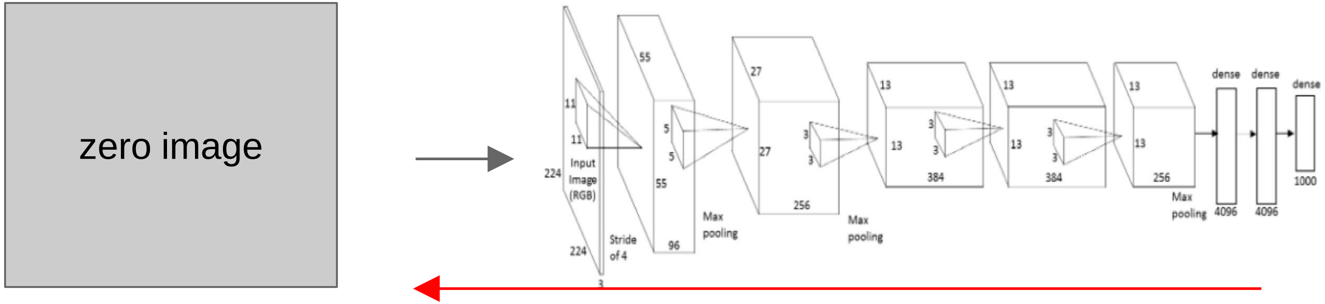


Generate image that maximizes some class score?

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

Optimization to Image

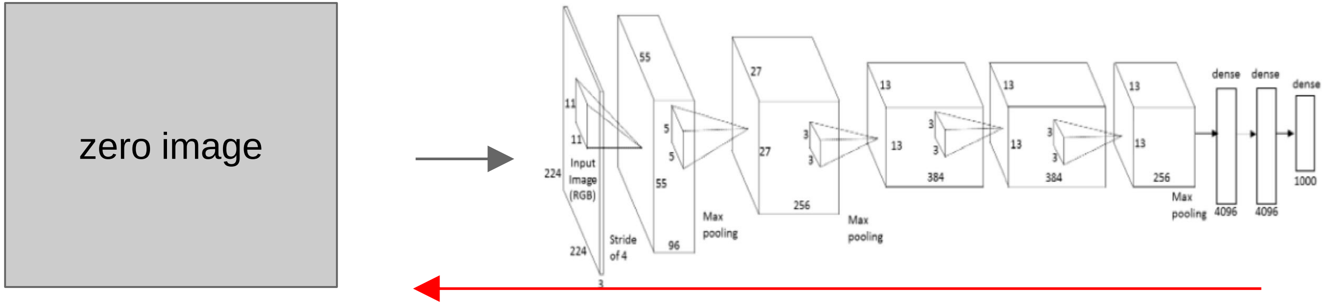
1. feed in zeros.



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

Optimization to Image

1. feed in zeros.



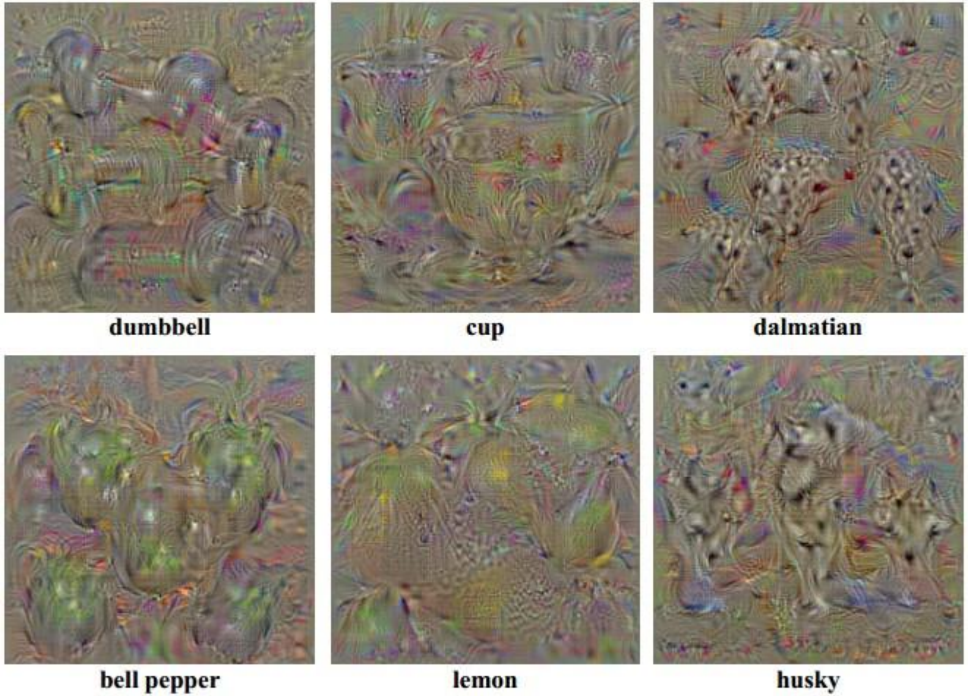
2. set the gradient of the scores vector to be [0,0,....1,....,0], then backprop to image
3. do a small “image update”
4. forward the image through the network.
5. go back to 2.

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

score for class c (before Softmax)

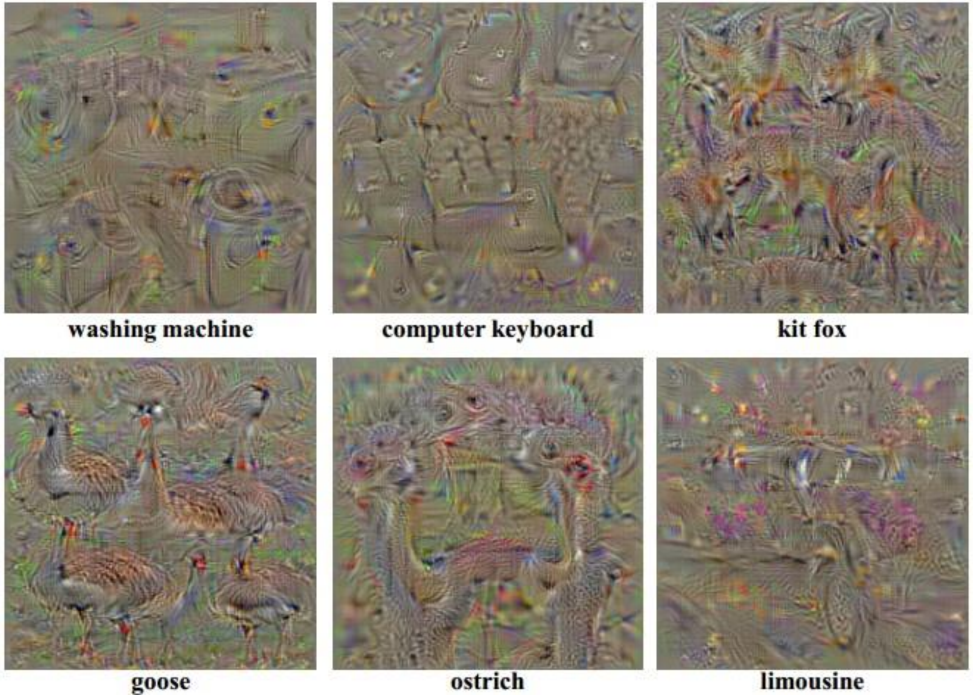
Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps
Karen Simonyan, Andrea Vedaldi, Andrew Zisserman, 2014

1. Generate 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. Generate images that maximize some class score:



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

Proposed 3 new regularizers beyond L_2

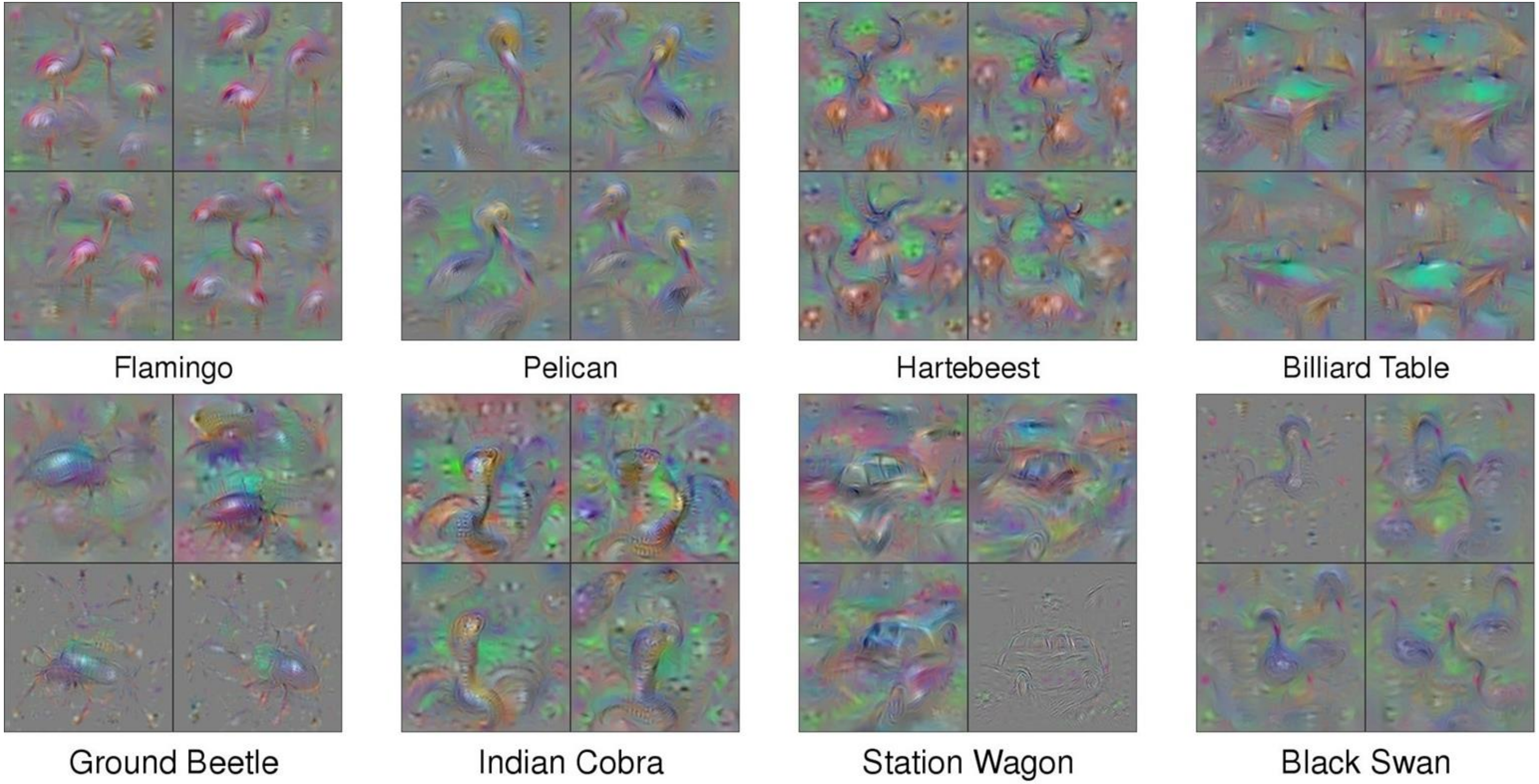
$$\arg\max_I S_c(I) - \boxed{\lambda \|I\|_2^2}$$

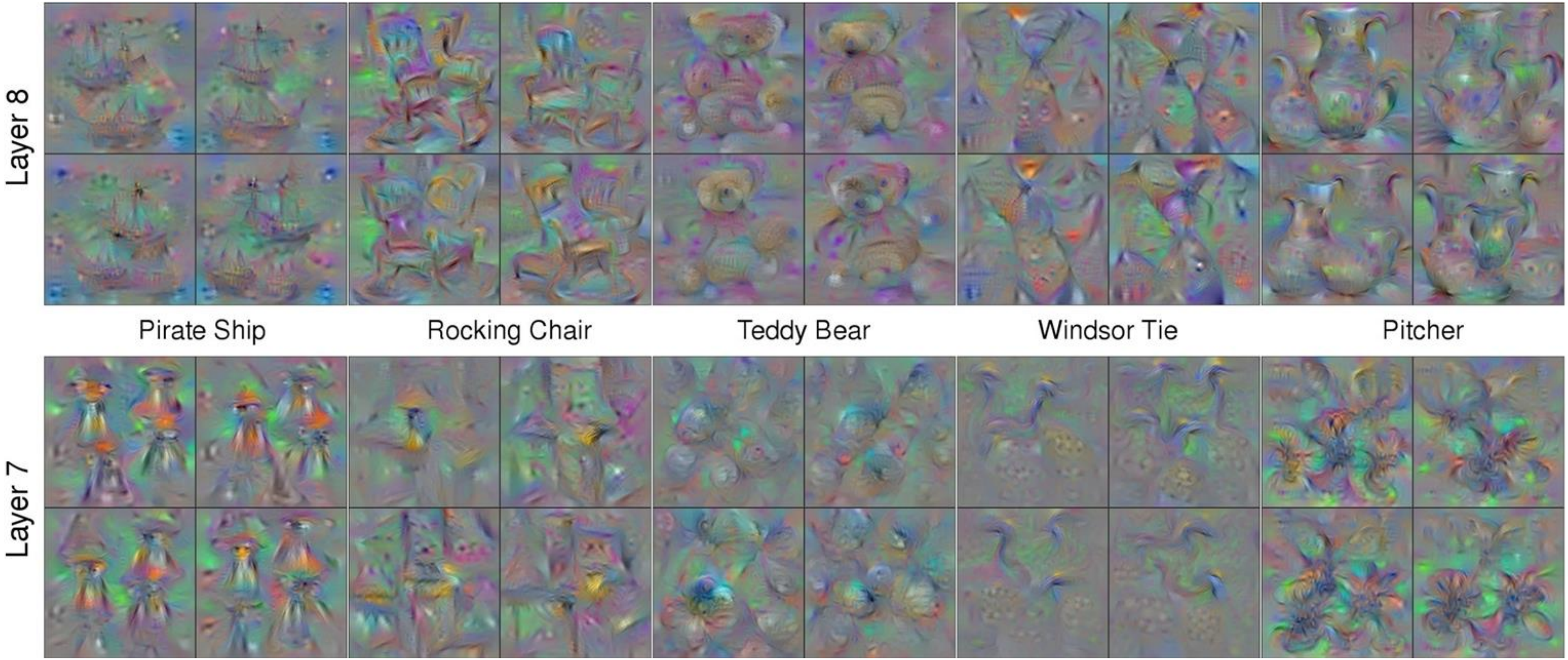
New Regularizers:

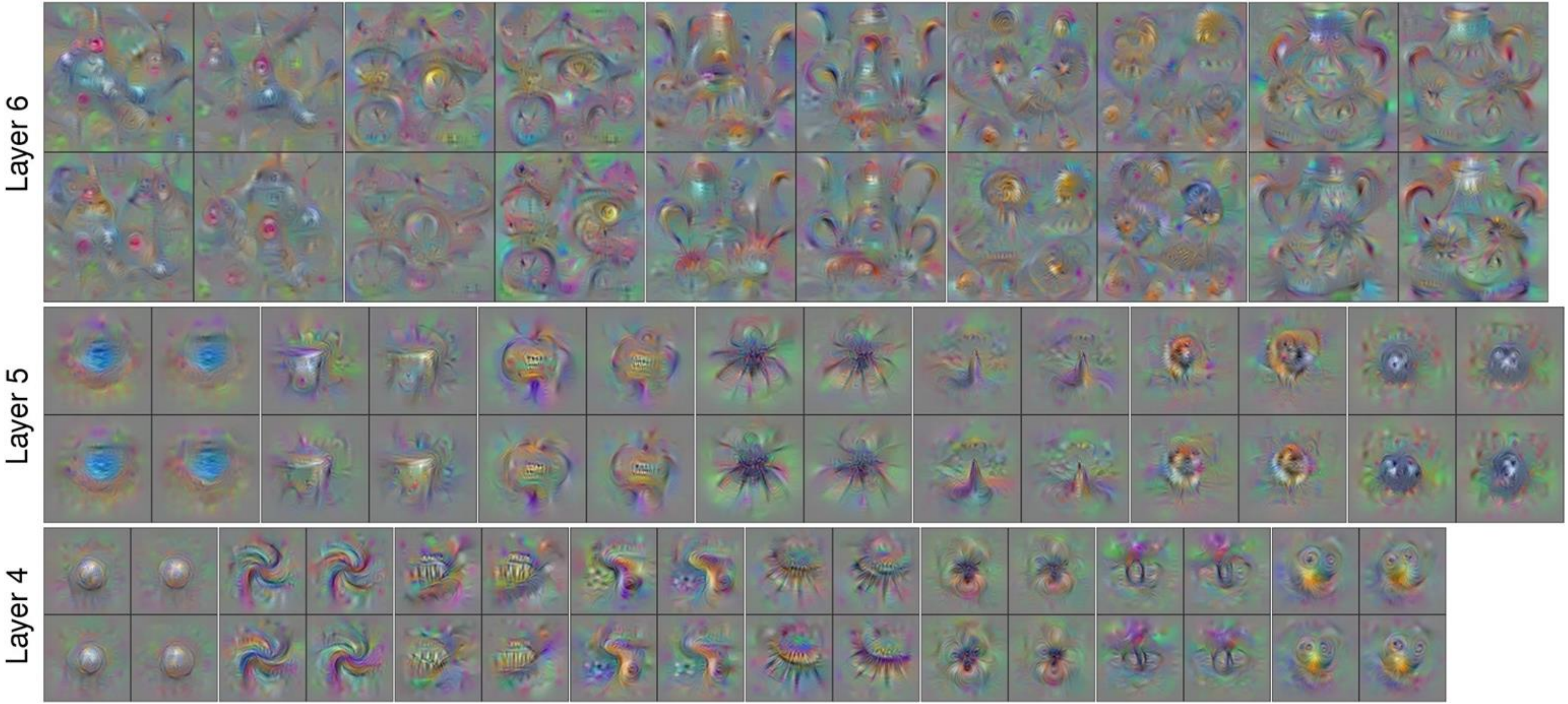
- Penalize high frequencies: Apply gaussian blur.
- Clip (zero) pixels with small norm using a threshold.
- Clip pixels with small contribution. Do this by ***ablation***: set pixel activation to zero, measure change in output. Zero pixels if change is small.

[Understanding Neural Networks Through Deep Visualization, Yosinski et al. , 2015] (AlexNet network)

http://yosinski.com/deepvis




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


Layer 6
Layer 5
Layer 4

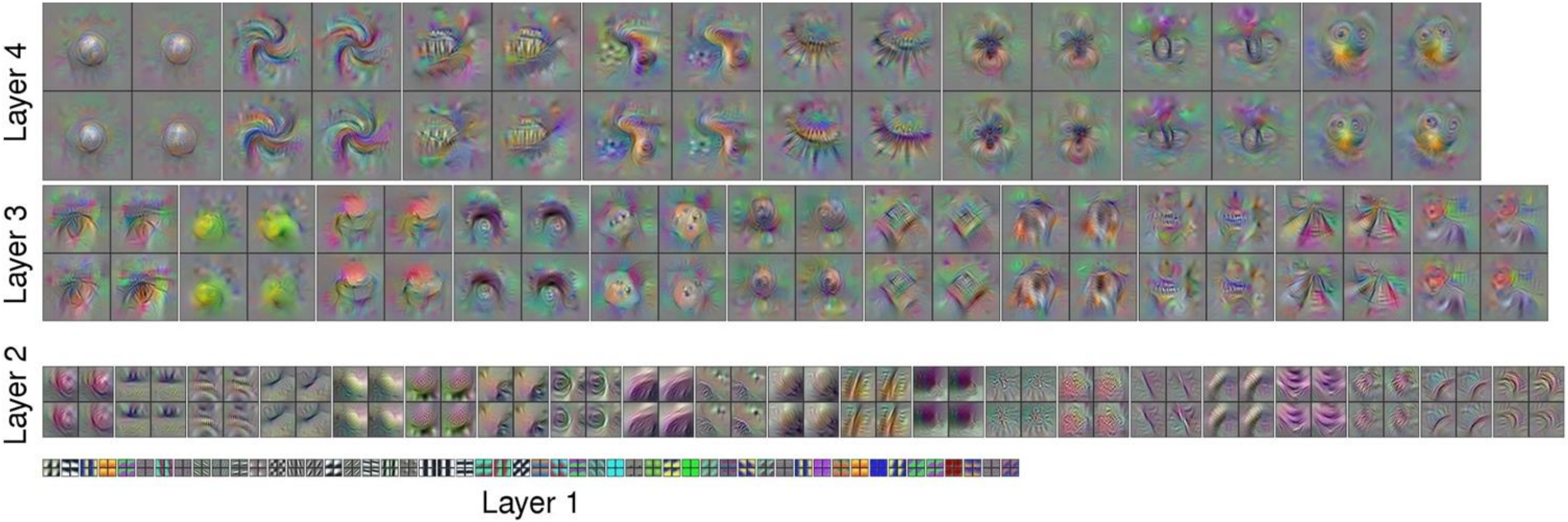
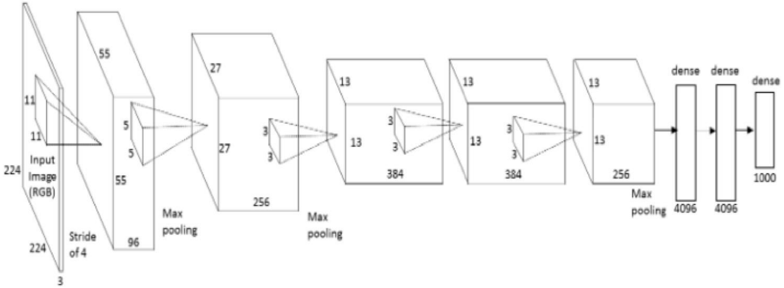

Layer 4
Layer 3
Layer 2
Layer 1

Image-Centric Approaches

1. Feed image into net



Q: Which pixels are most important for the class output on a particular image?

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)

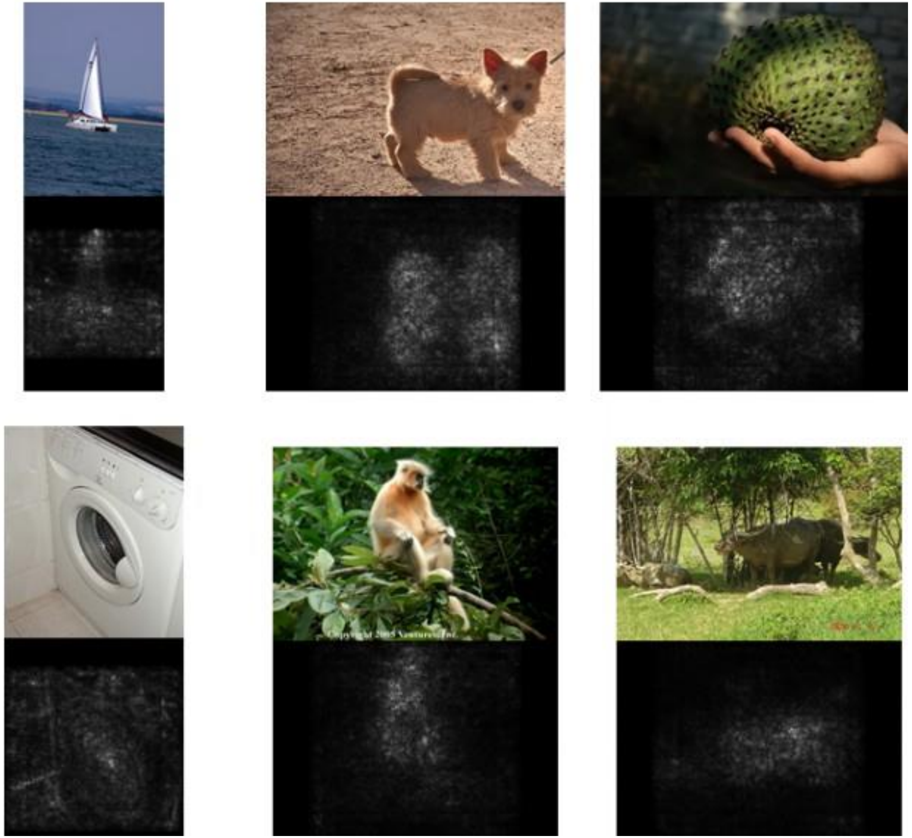
Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps
Karen Simonyan, Andrea Vedaldi, Andrew Zisserman, 2014

2. Visualize the Data gradient:

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

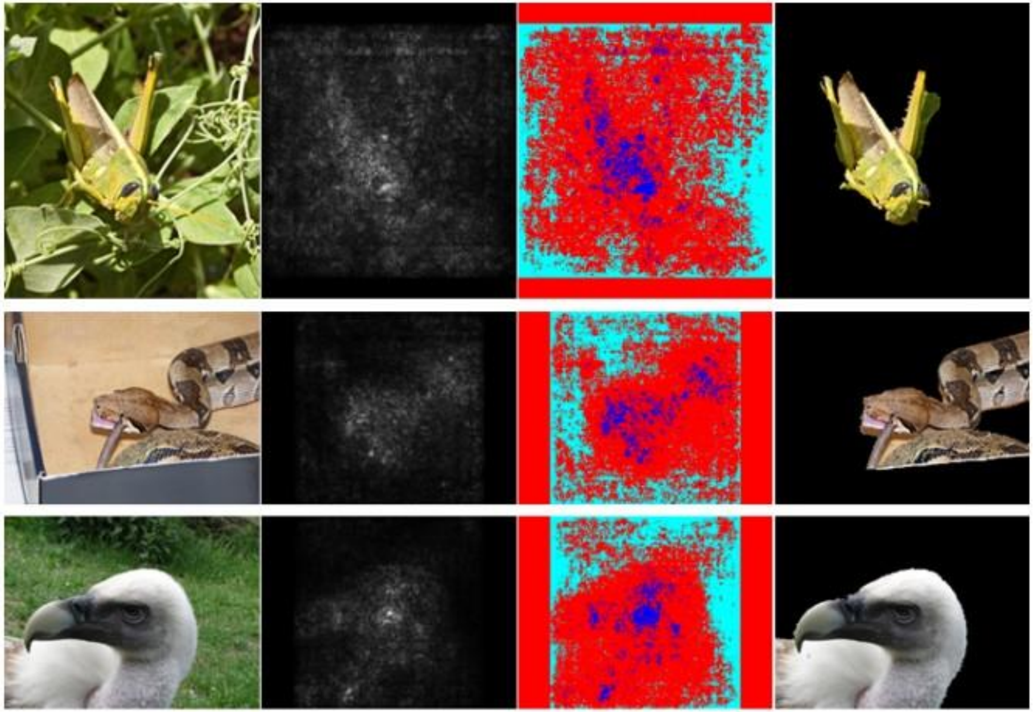
$$M_{ij} = \max_c |w_{h(i,j,c)}|$$

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



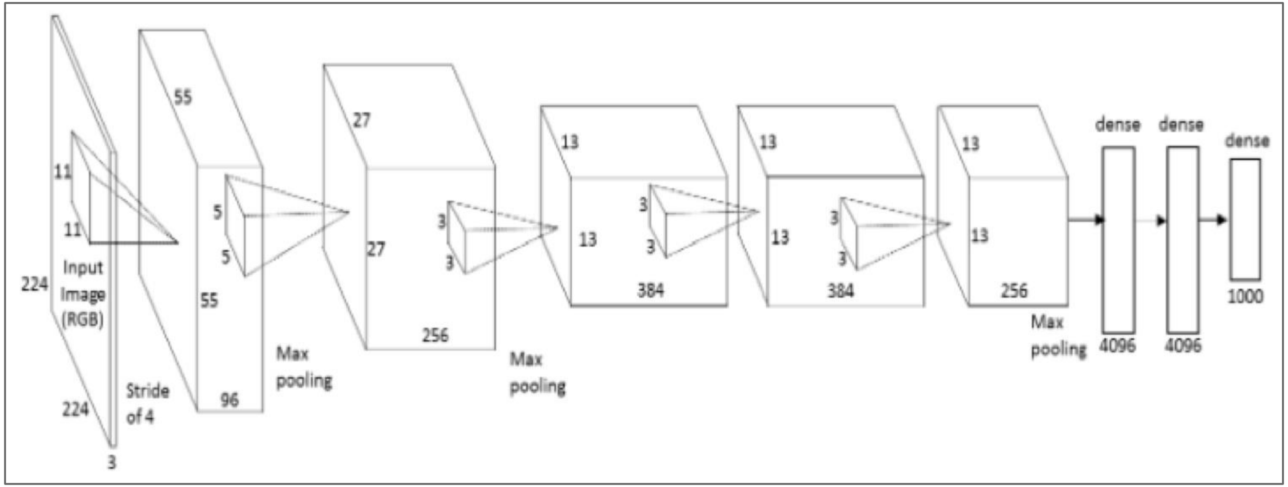
Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps
Karen Simonyan, Andrea Vedaldi, Andrew Zisserman, 2014

Use **grabcut** for segmentation:
- Use a box to select the object
- Compute the max class score
- Construct an saliency map for the class
- Segment the saliency map



Finds the activating object in the image

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

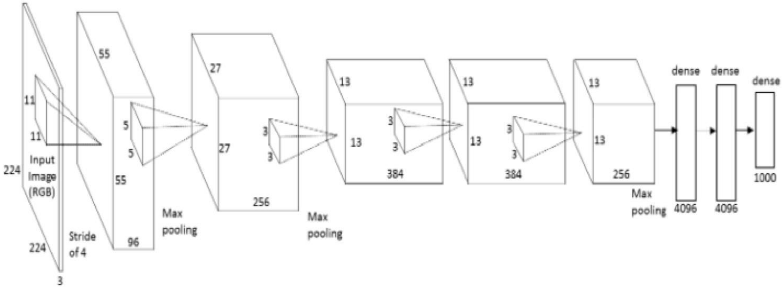


Repeat:

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”

Deconv approaches

1. Feed image into net



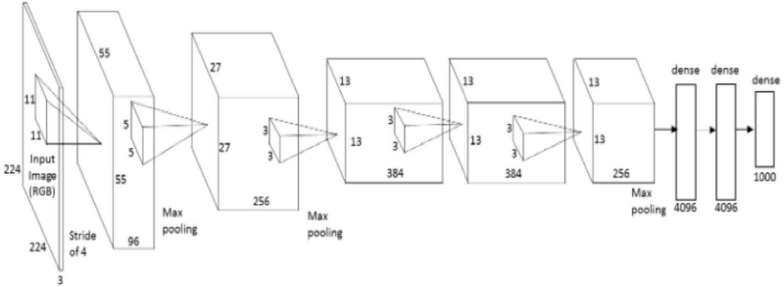
2. Propagate activations backwards using a “deconvnet”:

“A deconvnet can be thought of as a convnet model that uses the same components (filtering, pooling) but in reverse, so instead of mapping pixels to features does the opposite”

[Visualizing and Understanding Convolutional Networks, Zeiler and Fergus 2013]

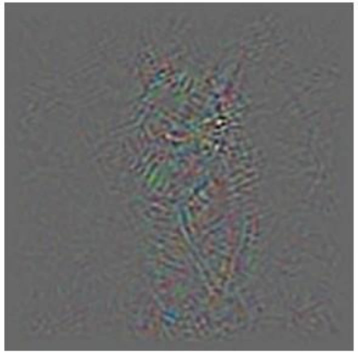
Deconv approaches

1. Feed image into net



2. Pick a layer, start from a given neuron, and propagate backwards:

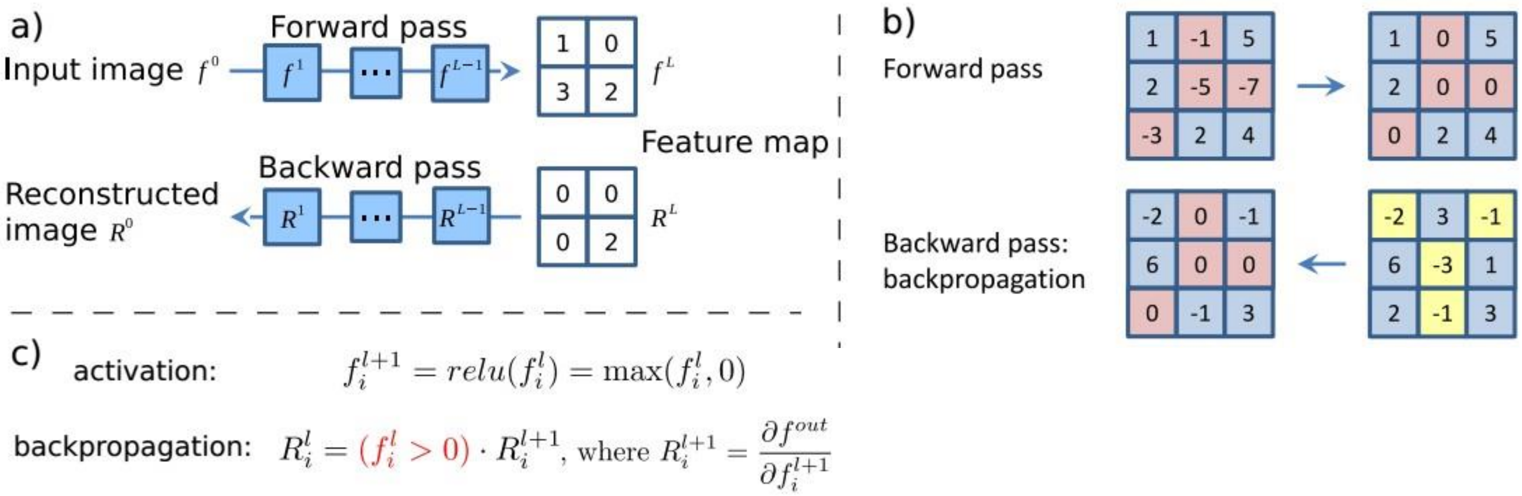
3. Deconv:



“Guided backpropagation:” instead

Deconv approaches

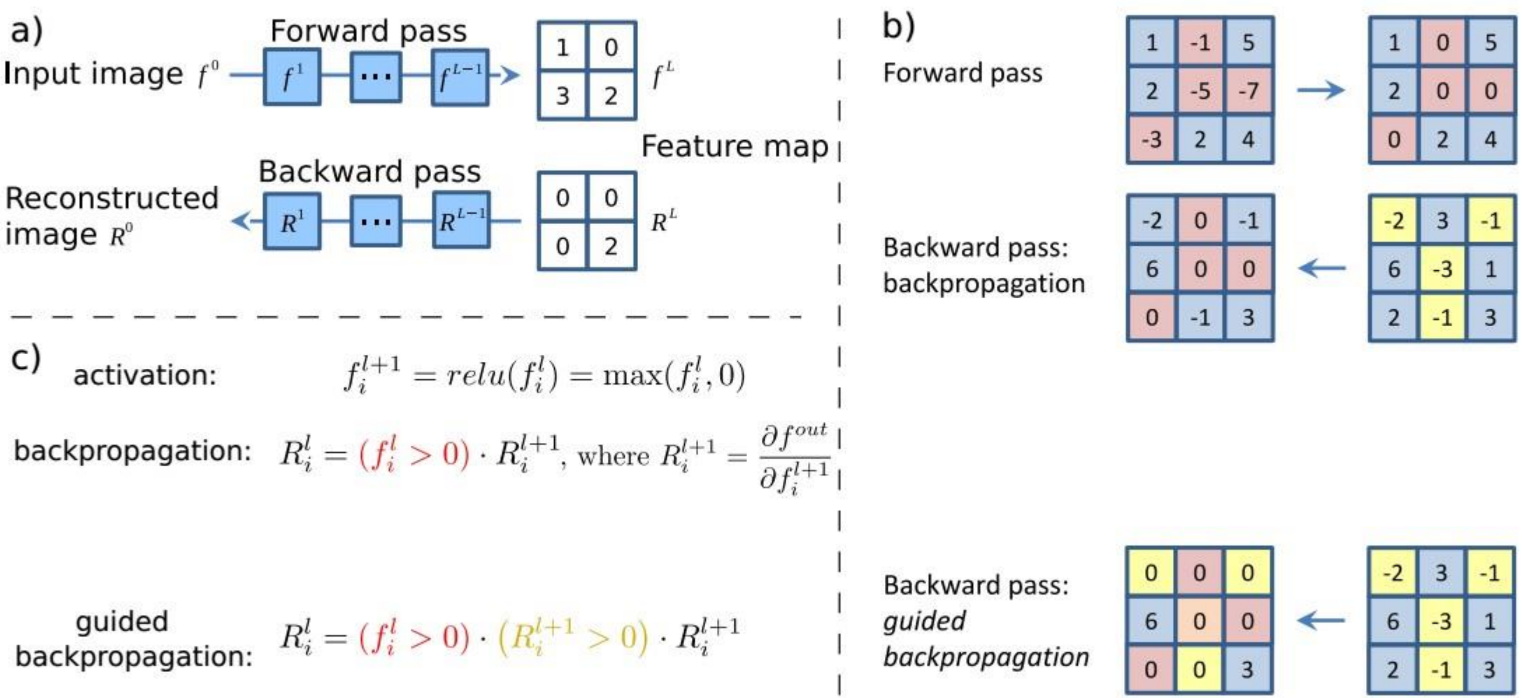
[Visualizing and Understanding Convolutional Networks, Zeiler and Fergus 2013]
[Striving for Simplicity: The all convolutional net, Springenberg, Dosovitskiy, et al., 2015]



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

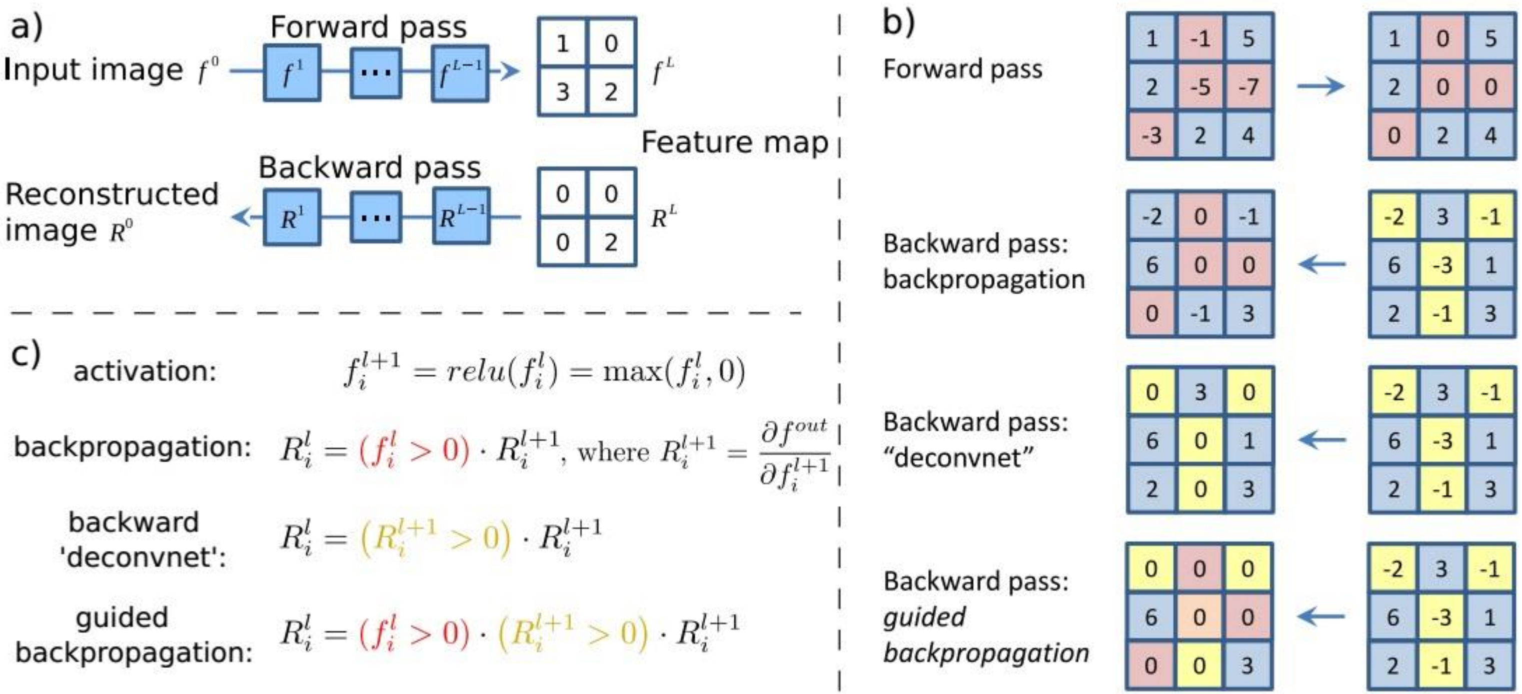
Deconv approaches

[Visualizing and Understanding Convolutional Networks, Zeiler and Fergus 2013]
[Striving for Simplicity: The all convolutional net, Springenberg, Dosovitskiy, et al., 2015]



Deconv approaches

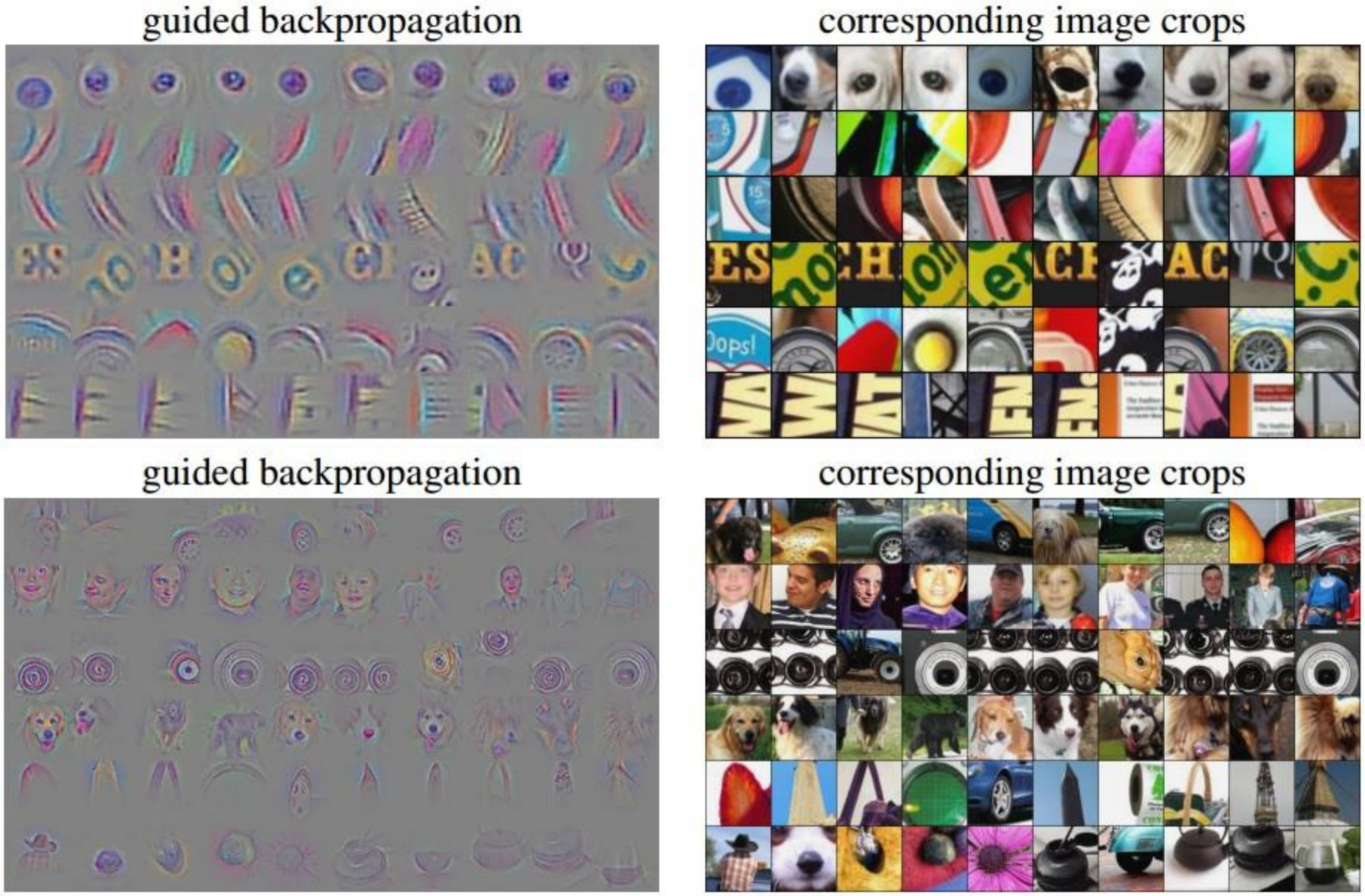
[Visualizing and Understanding Convolutional Networks, Zeiler and Fergus 2013]
[Striving for Simplicity: The all convolutional net, Springenberg, Dosovitskiy, et al., 2015]



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.

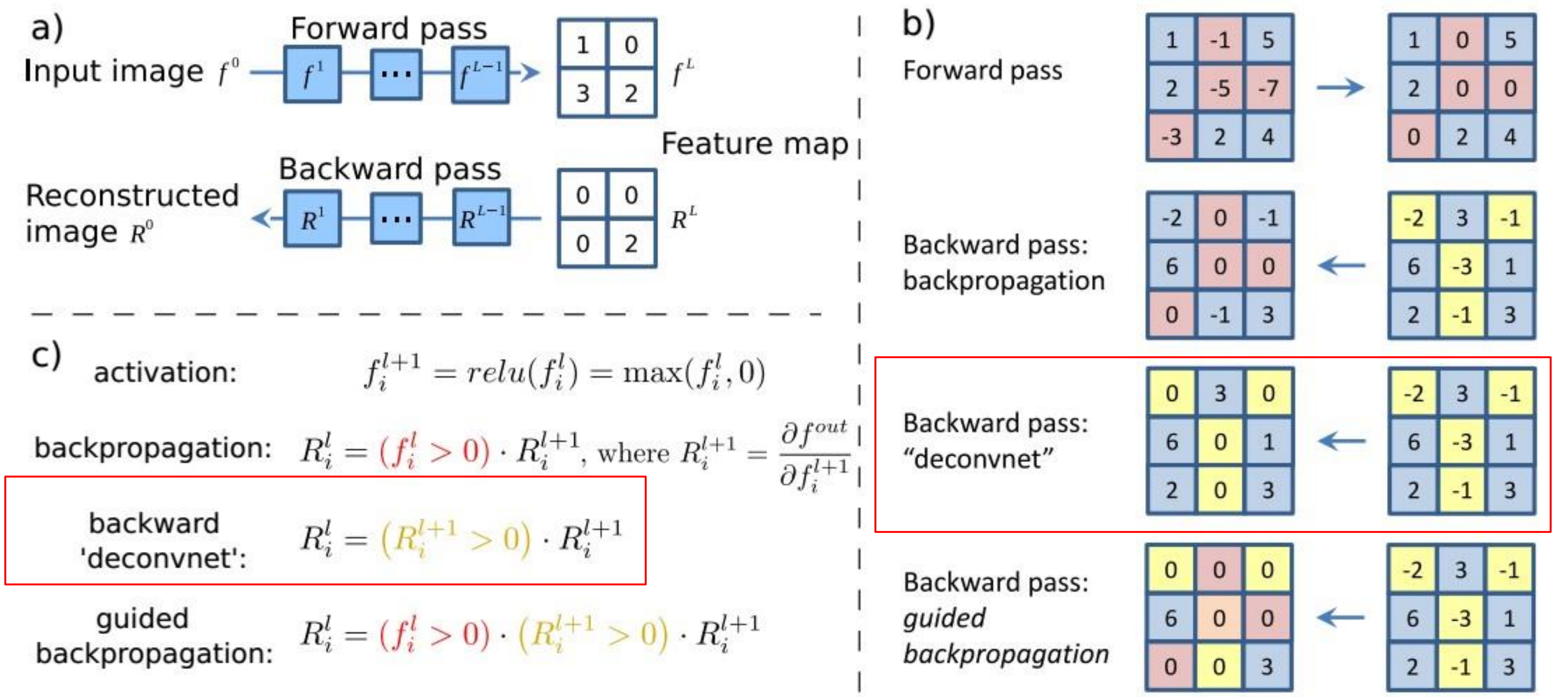
The visualization using “guided backpropagation” is based on the top 10 image patches activating this filter taken from the ImageNet dataset.



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

Deconv approaches

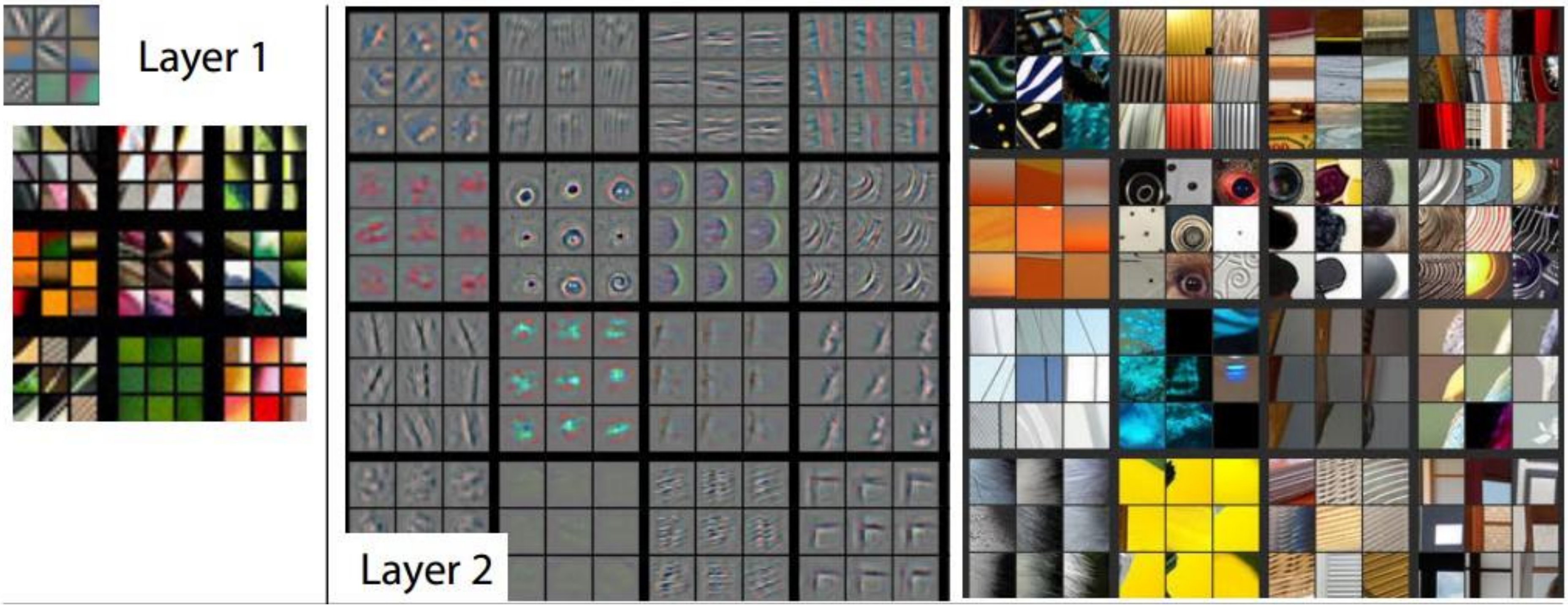
[Visualizing and Understanding Convolutional Networks, Zeiler and Fergus 2013]
[Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, Simonyan et al., 2014]
[Striving for Simplicity: The all convolutional net, Springenberg, Dosovitskiy, et al., 2015]



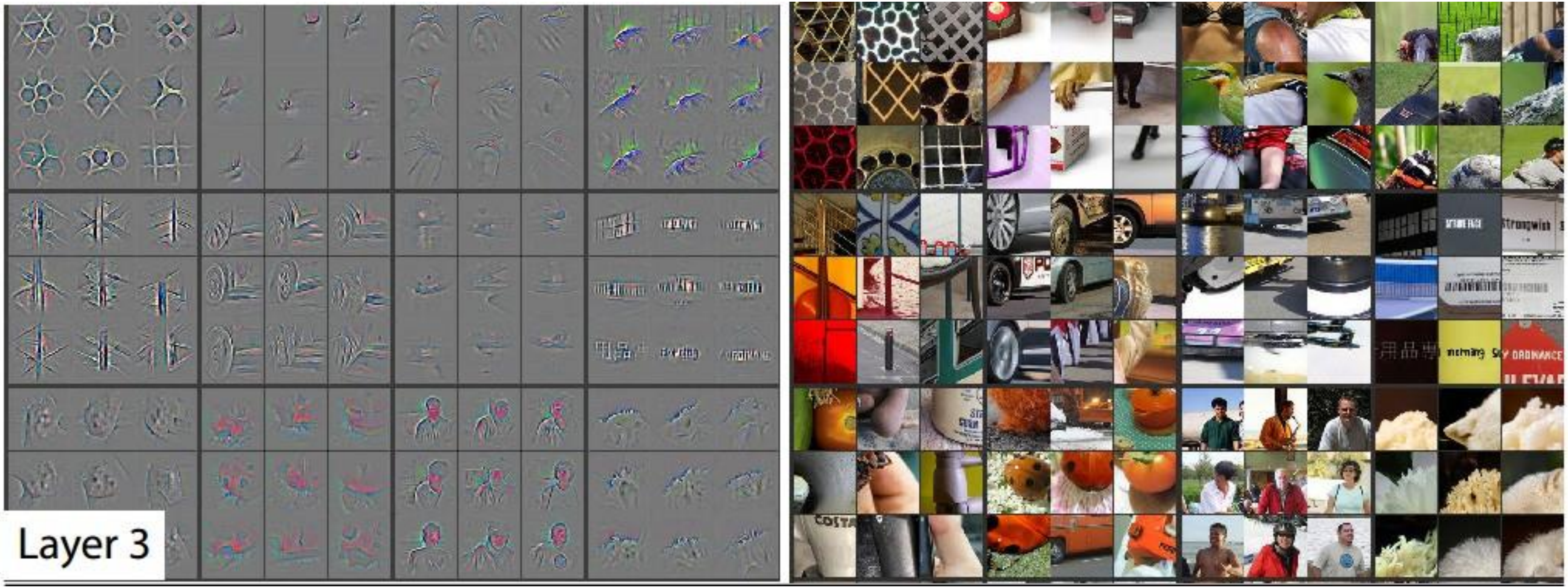
Intuition: Feature maps shouldn't be conditioned on images

Visualizing and Understanding Convolutional Networks
Zeiler & Fergus, 2013

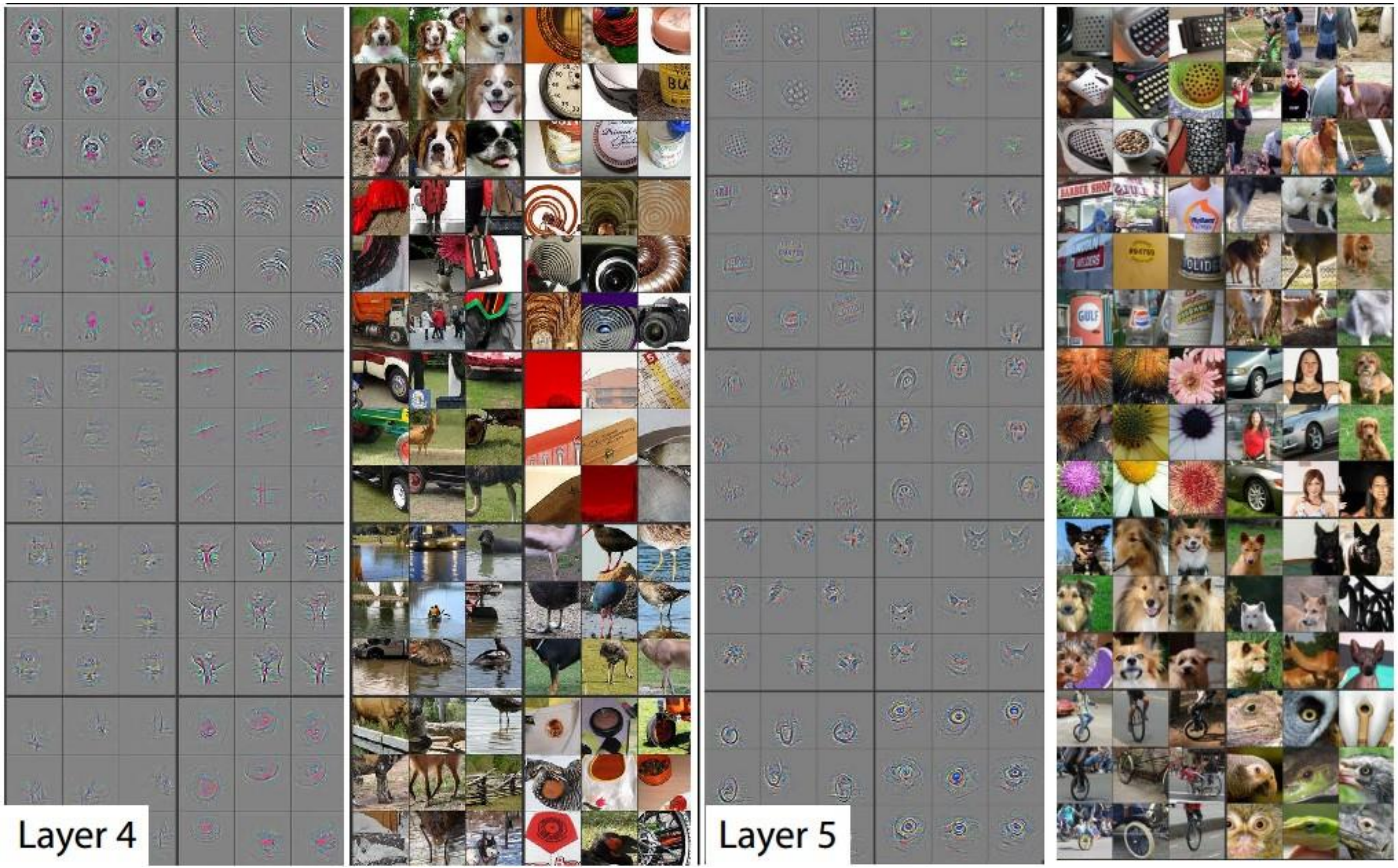
Visualizing arbitrary neurons along the way to the top...



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



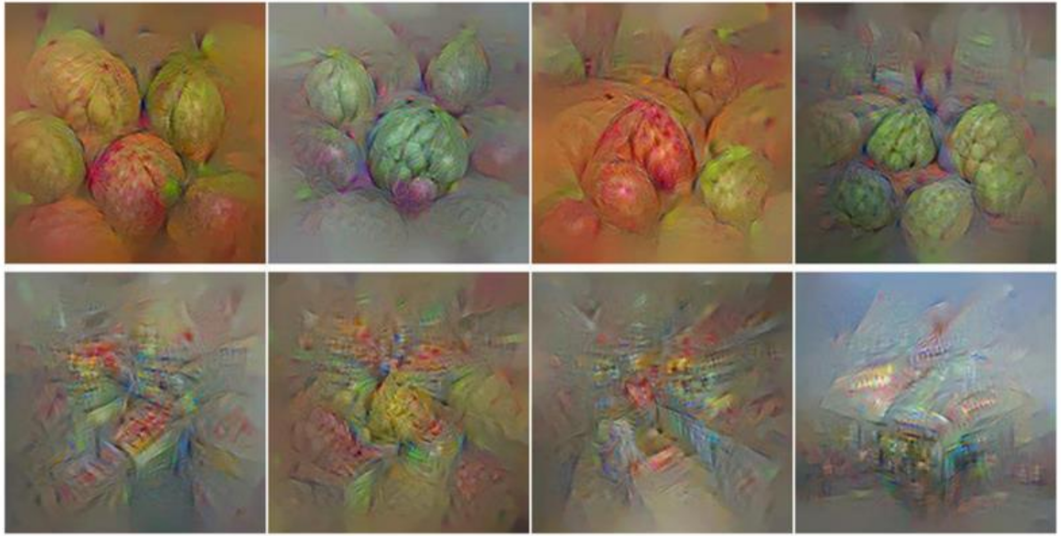
Visualizing arbitrary neurons along the way to the top...



More pretty pictures

[Nguyen et al 2016 Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks]

Reconstructions of multiple feature types (facets) recognized by the same “grocery store” neuron



Change initialization: project the training set images that maximally activate a neuron into a low-dimensional space (here, a 2D space via t-SNE), cluster the images via k-means, and average the n (here, 15) closest images to each cluster centroid to produce the initial image.

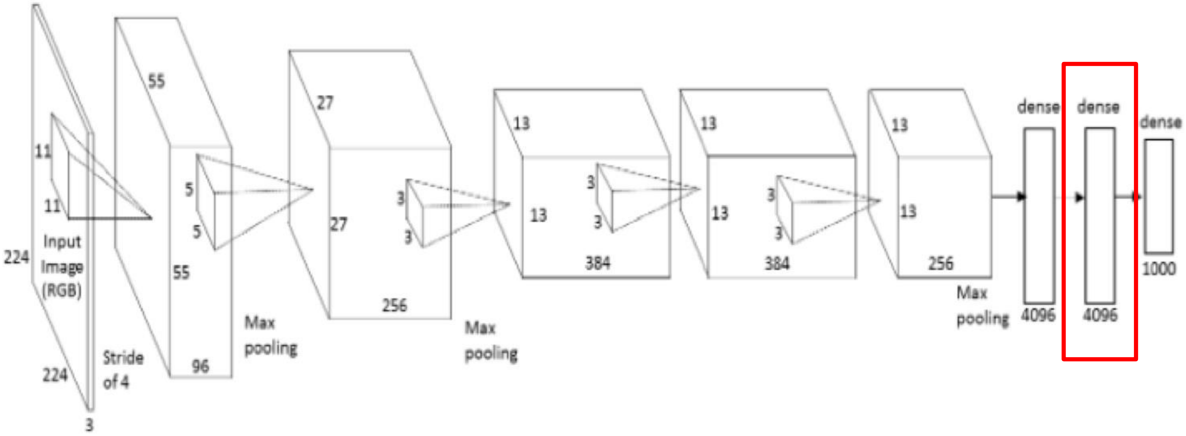
More pretty pictures

Nguyen et al 2016

[Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks]

pretty!

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



Find an image such that:

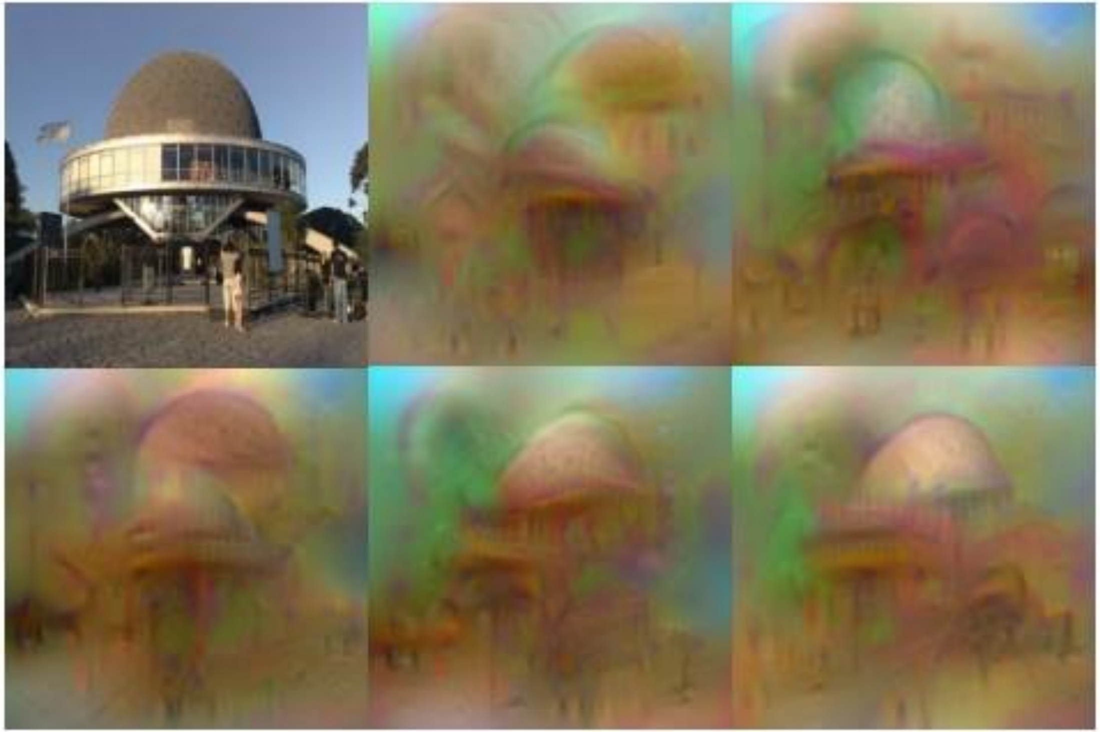
- Its code is similar to a given code
- It “looks natural” (image prior regularization)

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

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

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

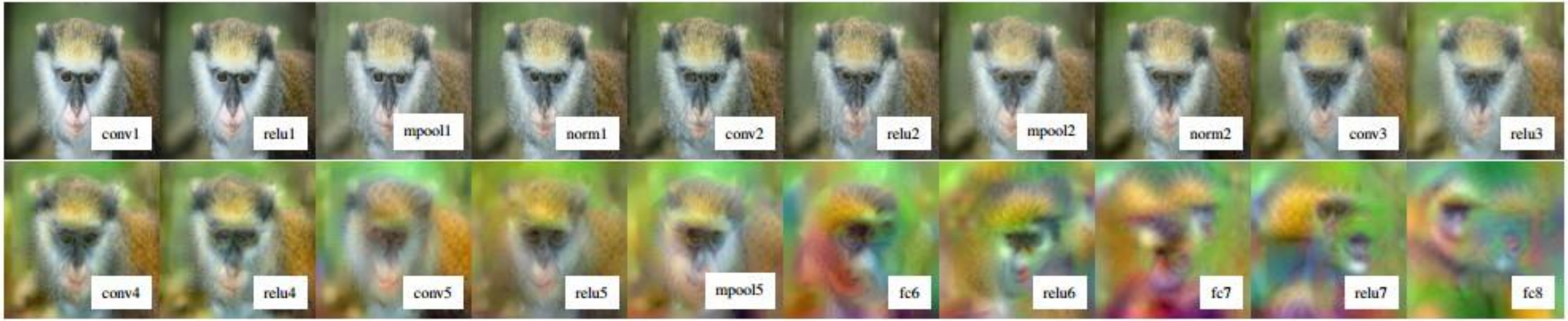
original image



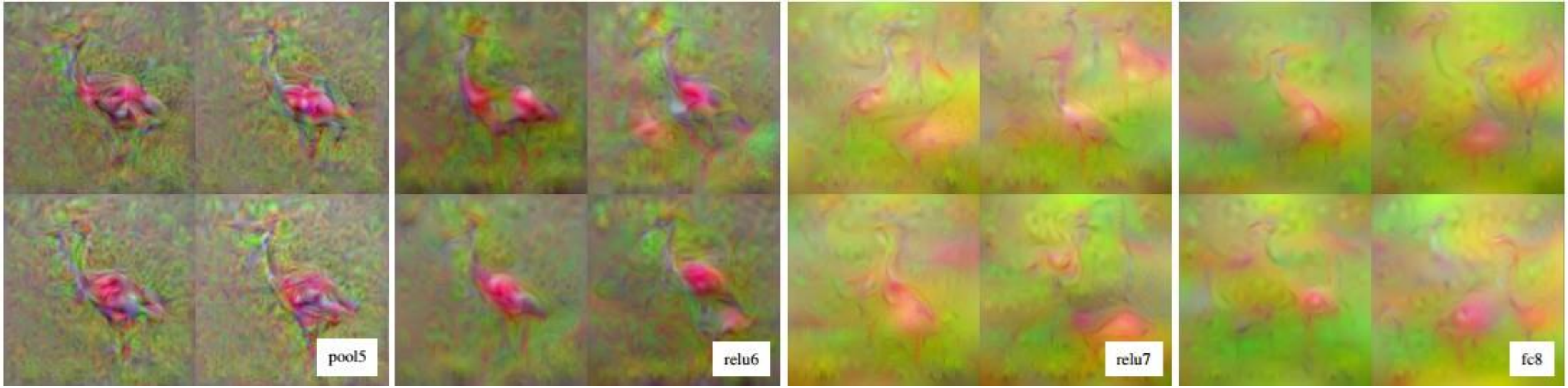
reconstructions from the 1000 log probabilities for ImageNet (ILSVRC) classes

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

Reconstructions from intermediate layers



Multiple reconstructions. Images in quadrants all “look” the same to the CNN (same code)



Inverting Visual Representations with Convolutional Networks

(Another **code inversion** approach from [Dosovitskiy and Brox 2015])

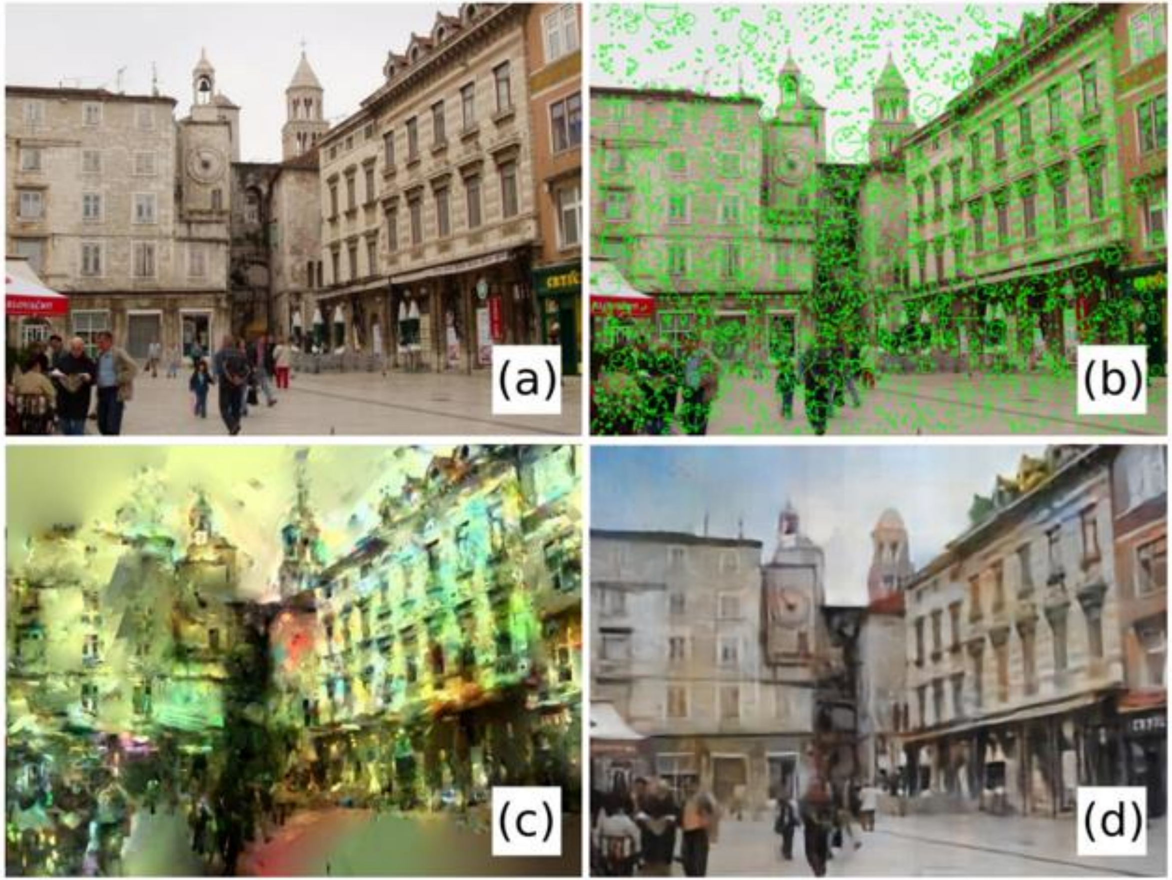
- Requires no optimization “at test time”, directly trains the image reconstructor with Euclidean loss to the original true image.

$$\boxed{W^* = \arg\min_W \sum_i ||x_i - f(\Phi(x_i), W)||_2^2}$$

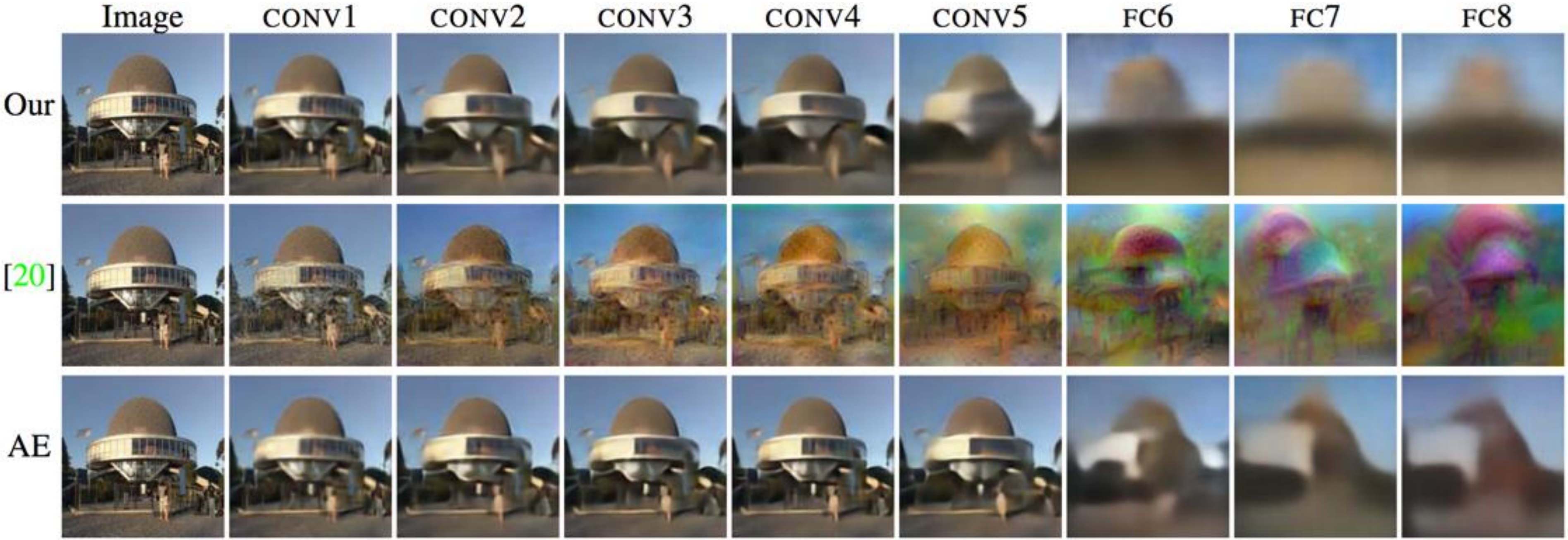
i.e. directly train a network for the mapping: features -> image.

[Dosovitskiy and Brox 2015]
Inverting SIFT features:

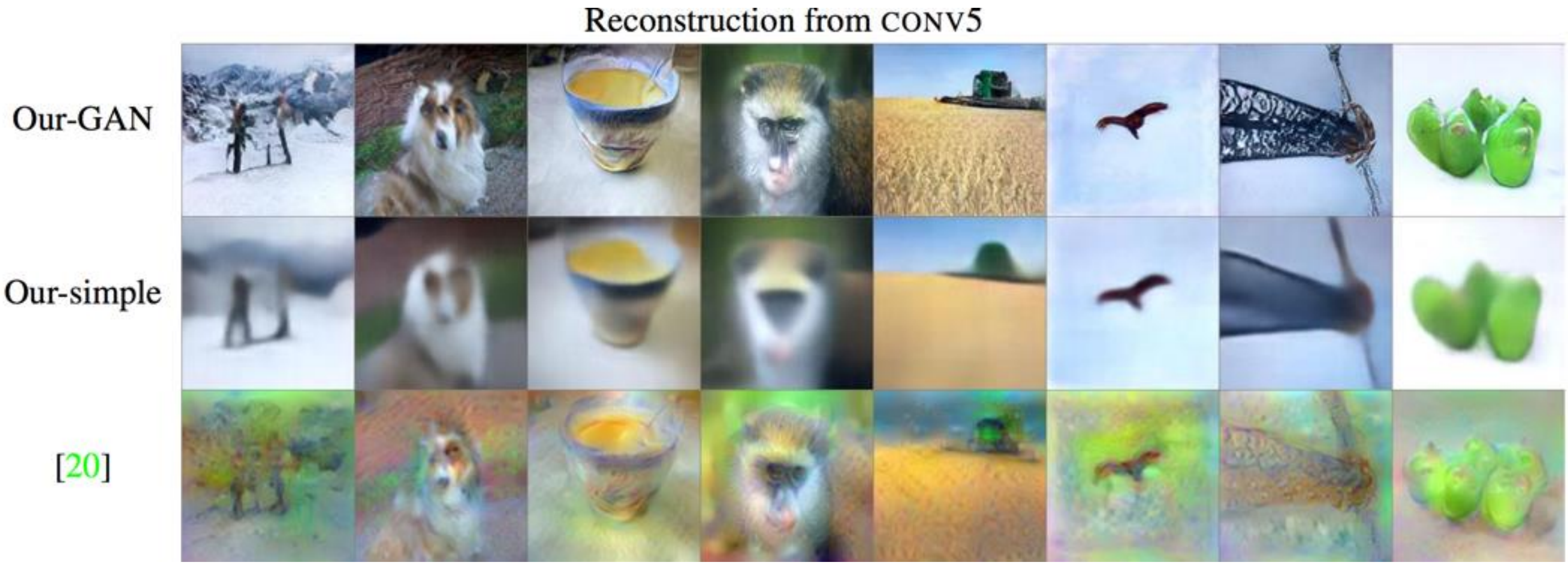
Previous work



[Dosovitskiy and Brox 2015]



[Dosovitskiy and Brox 2015]

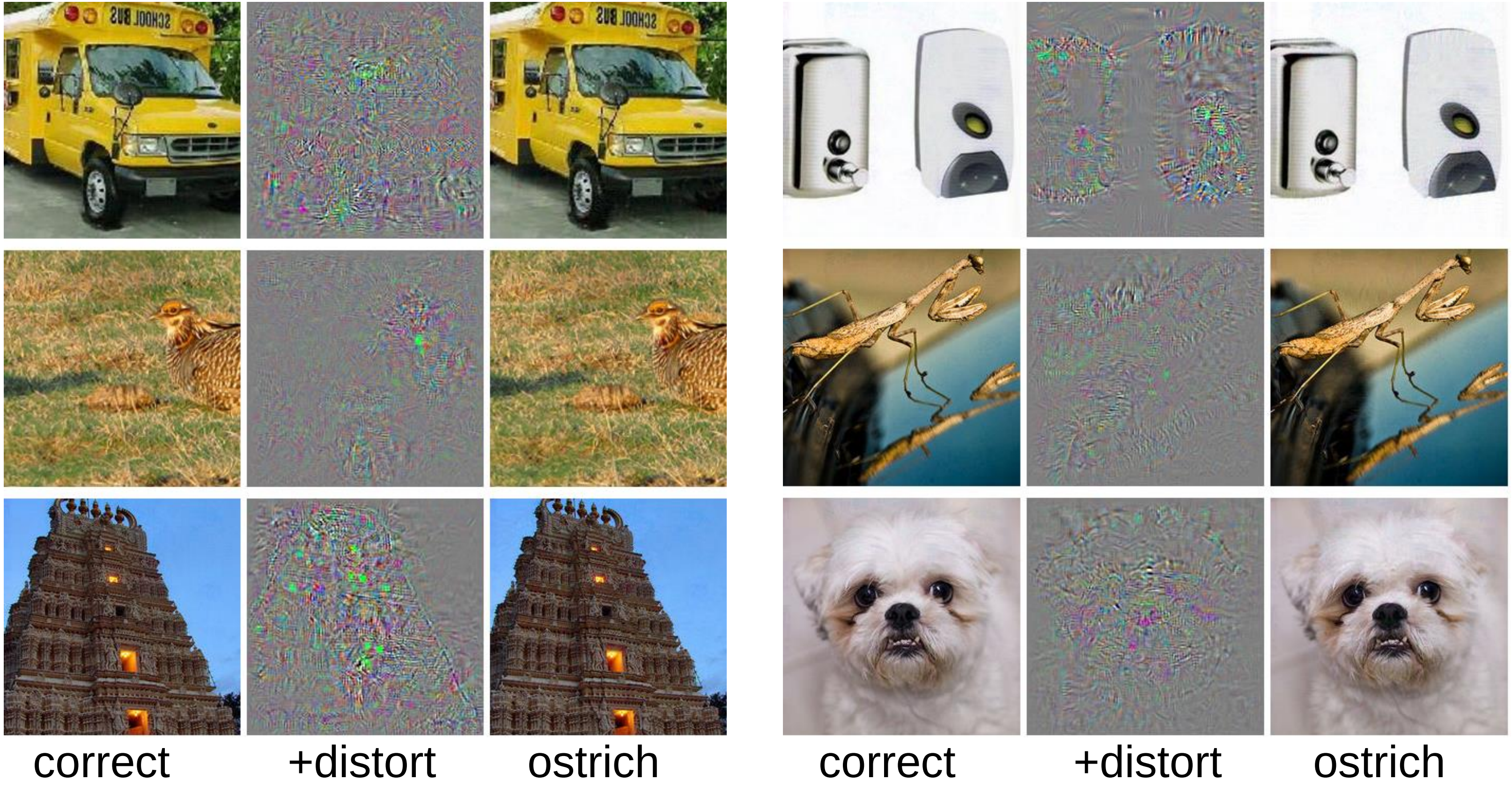


We can pose an optimization over the input image to maximize any class score.
That seems useful.

Question: Can we use this to “fool” ConvNets?

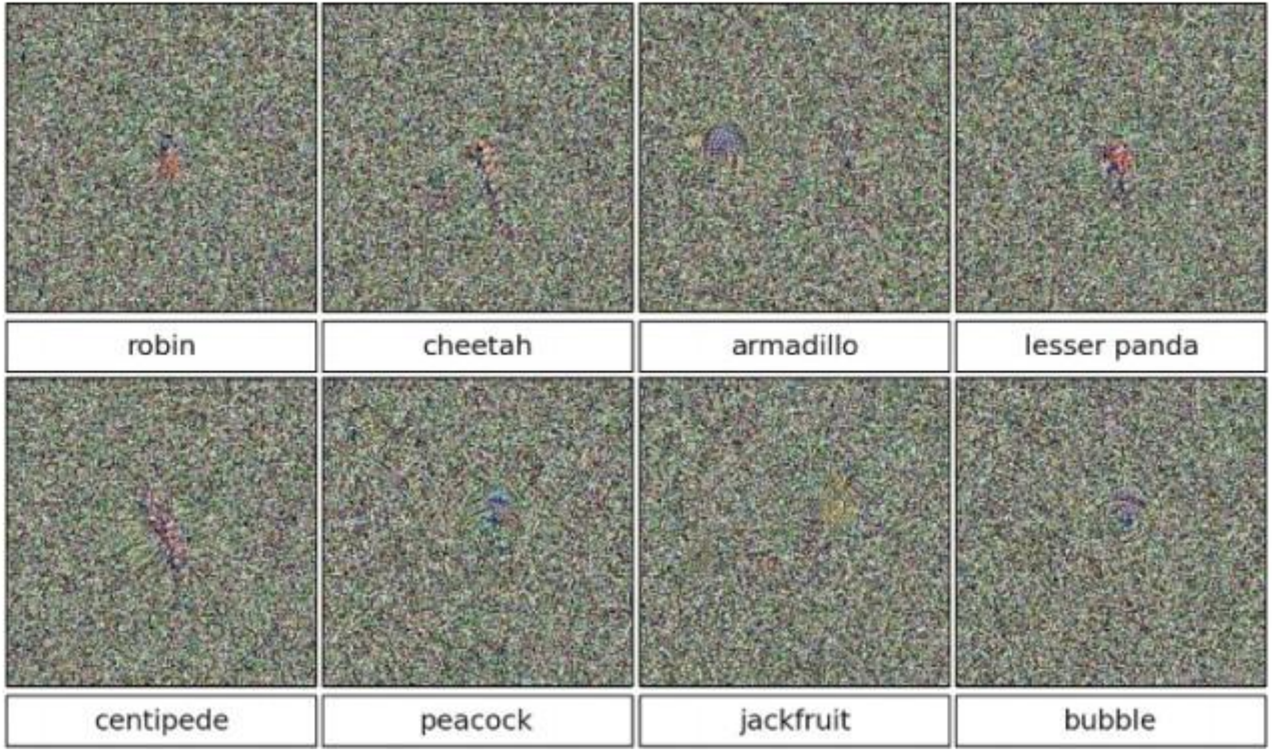
spoiler alert: yeah

[Intriguing properties of neural networks, Szegedy et al., 2013]



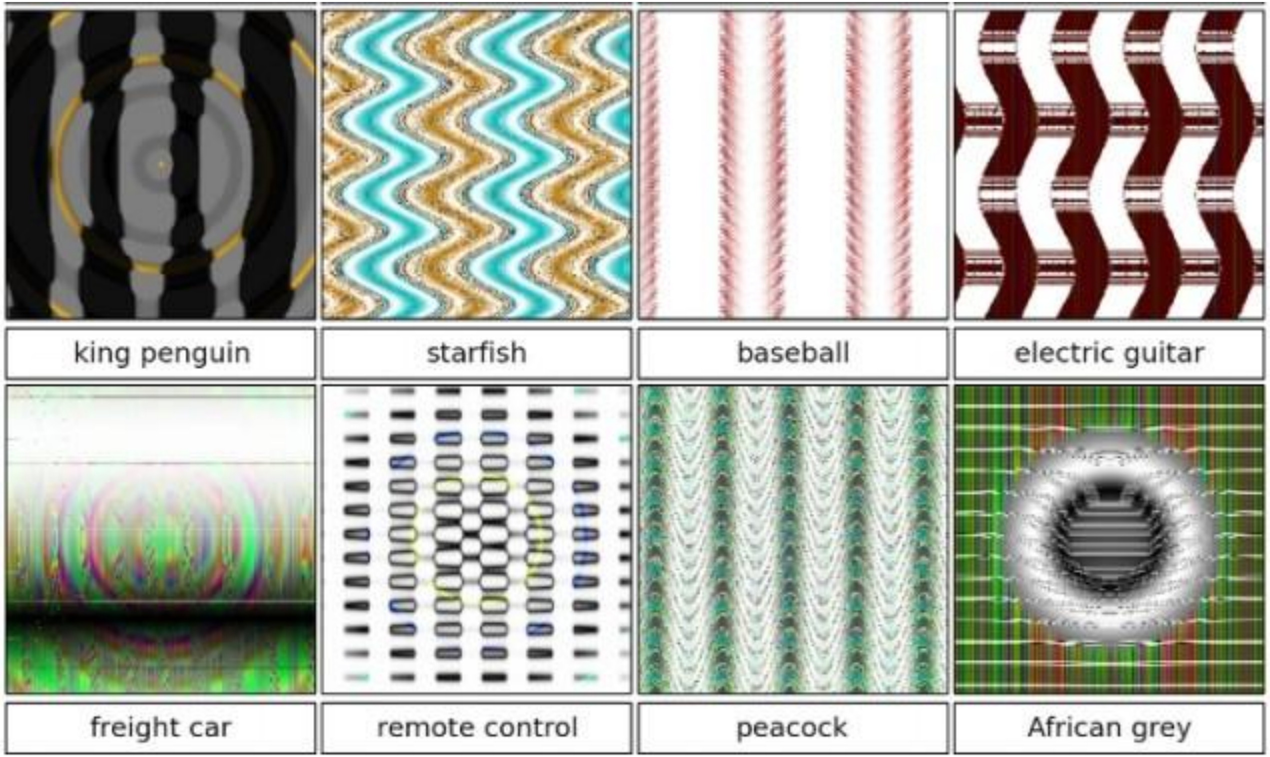
[Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images Nguyen, Yosinski, Clune, 2014]

>99.6% confidences



[Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images Nguyen, Yosinski, Clune, 2014]

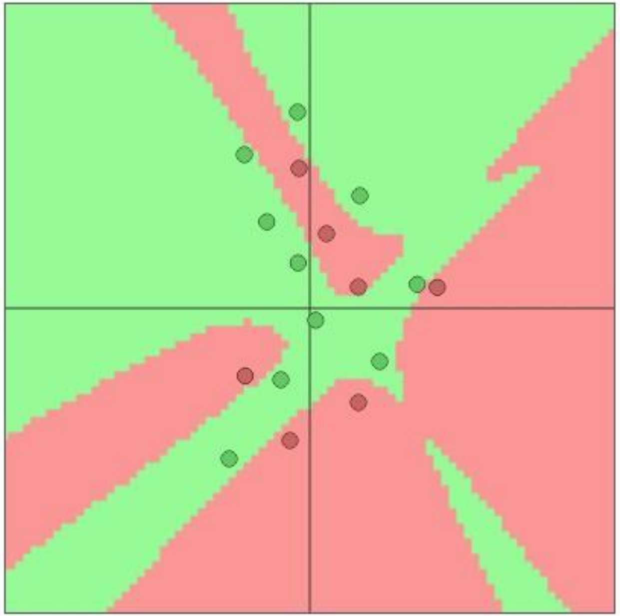
>99.6%
confidences

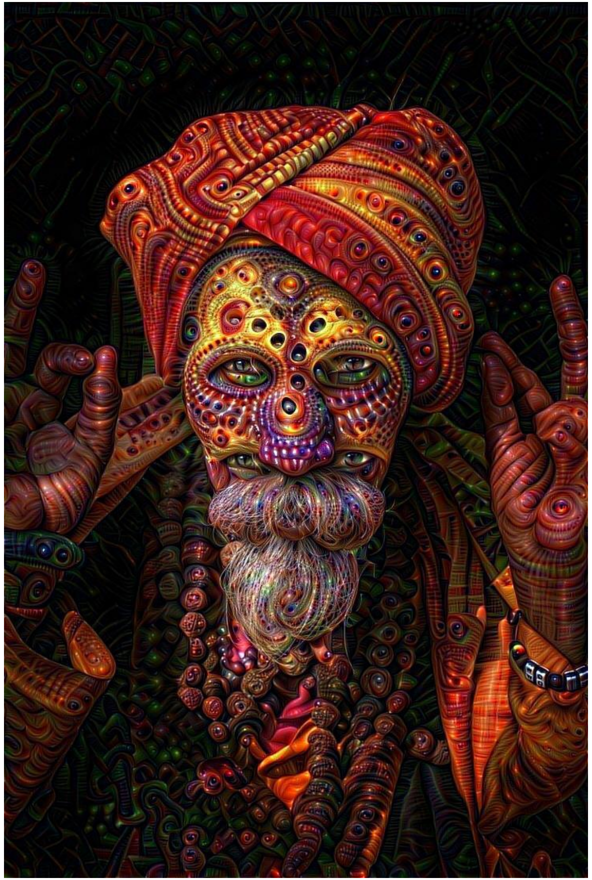


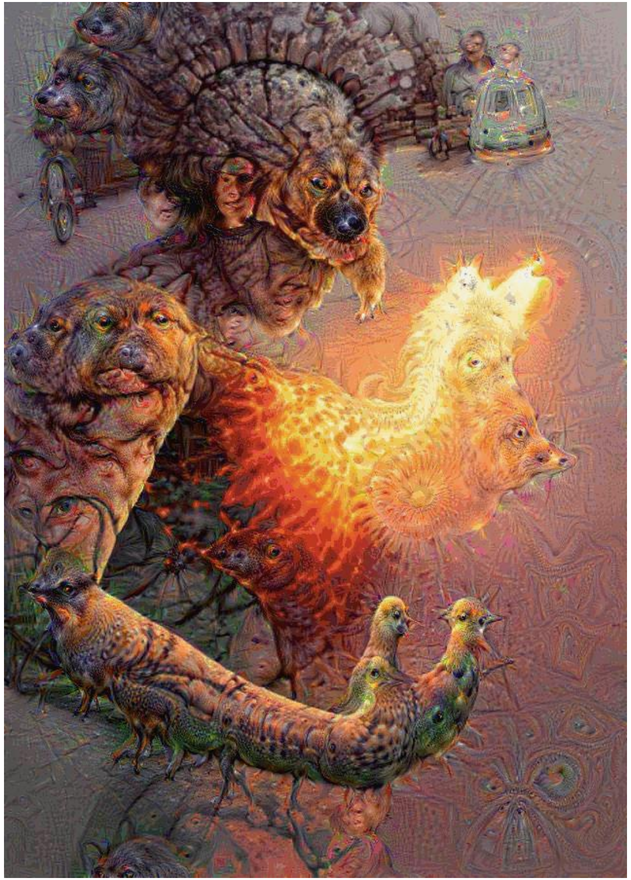
EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES
[Goodfellow, Shlens & Szegedy, 2014]

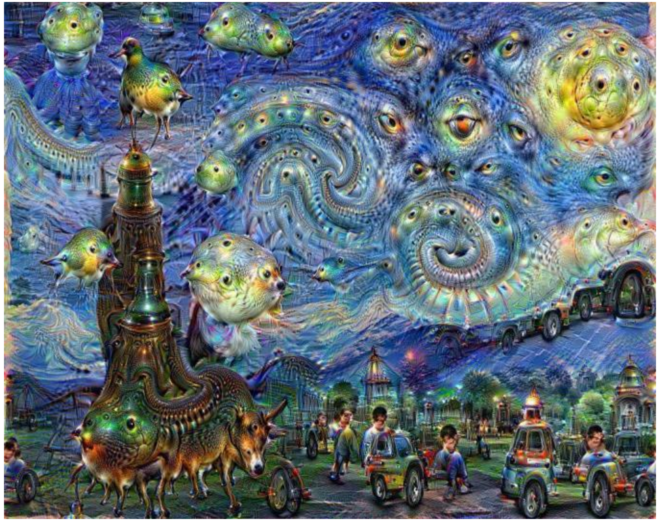
“primary cause of neural networks’ vulnerability to adversarial perturbation is their **linear nature**“

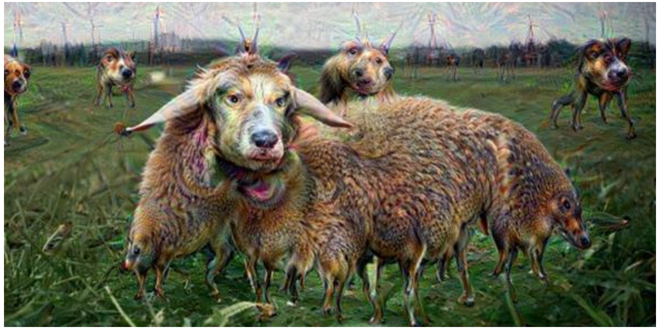
In particular, this is not a problem with Deep Learning, and has little to do with ConvNets specifically. Same issue would come up with Neural Nets in any other modalities.











DeepDream https://github.com/google/deepdream

```python
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)
```

```python
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)
```

DeepDream: set dx = x :)

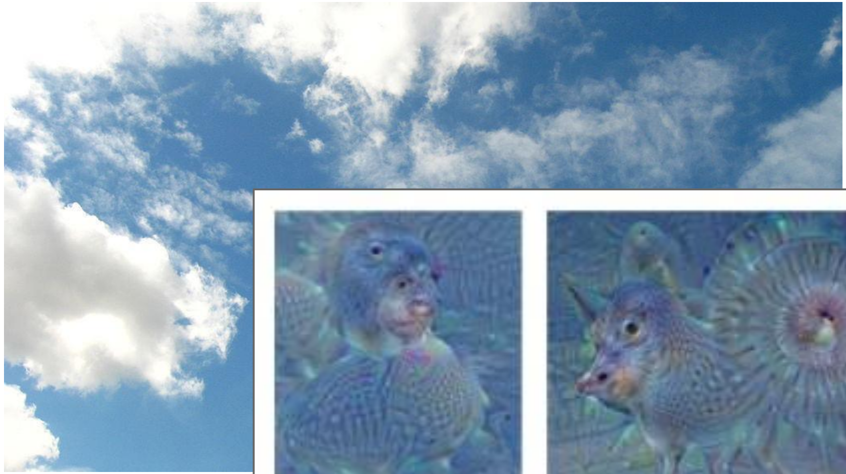
jitter regularizer

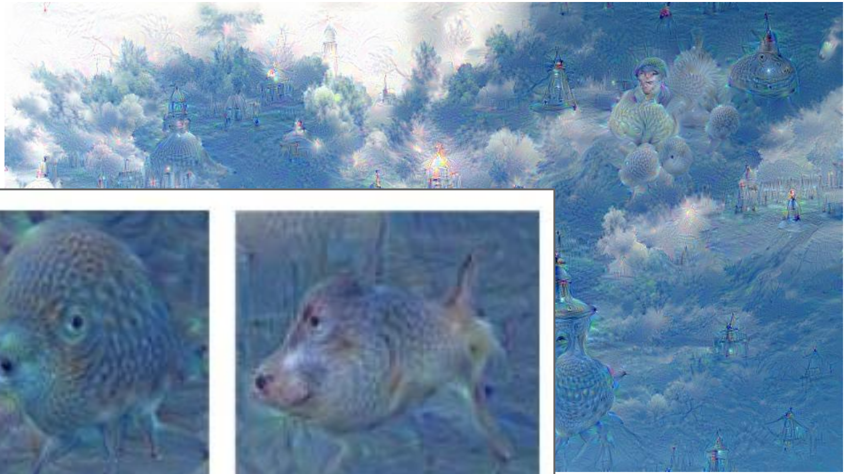
“image update”

inception_4c/output

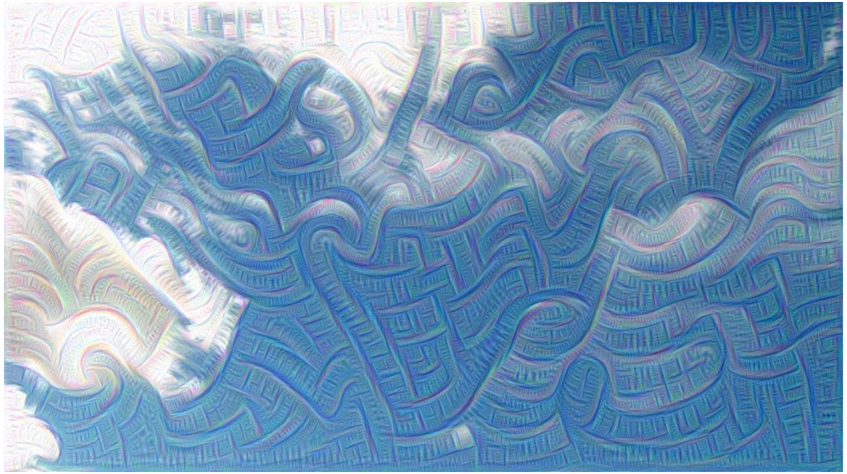
DeepDream modifies the image in a way that “boosts” all activations, at any layer

this creates a feedback loop: e.g. any slightly detected dog face will be made more and more dog like over time inception_4c/output





DeepDream modifies the image in a way that "boosts" all activations, at any layer inception_3b/5x5_reduce



DeepDream modifies the image in a way that “boosts” all activations, at any layer

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

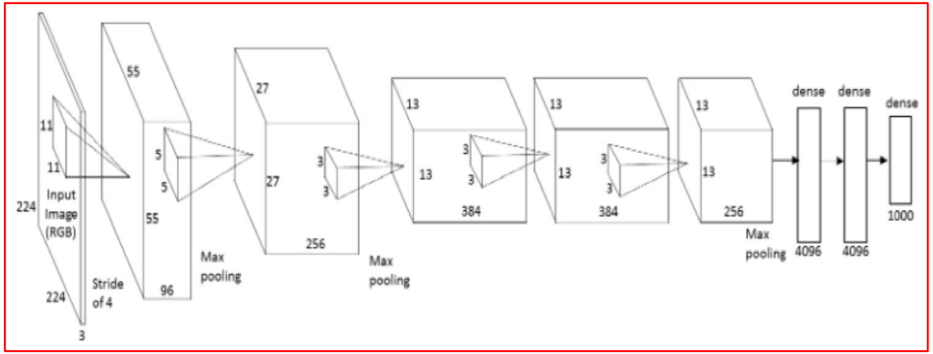
NeuralStyle

[A Neural Algorithm of Artistic Style by Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge, 2015]

good implementation by Justin Johnson in Torch:
https://github.com/jcjohnson/neural-style

make your own easily on deepart.io

Step 1: Extract **content targets** (ConvNet activations of all layers for the given content image)

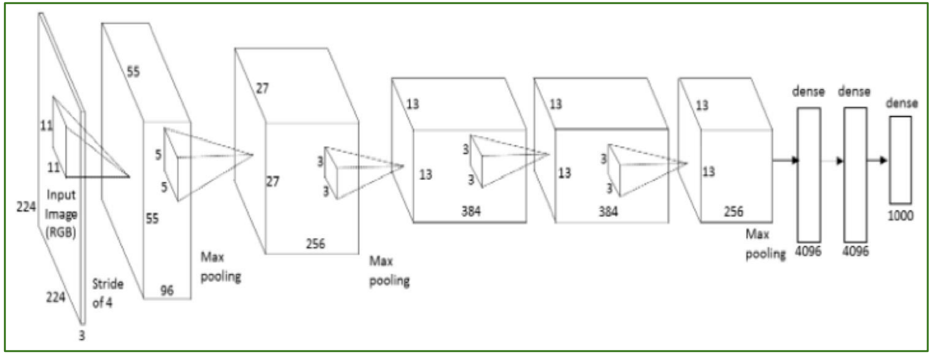


content activations

e.g.

at CONV5_1 layer we would have a [14x14x512] array of target activations

Step 2: Extract **style targets** (Gram matrices of ConvNet activations of all layers for the given style image)



style gram matrices $G = V^T V$

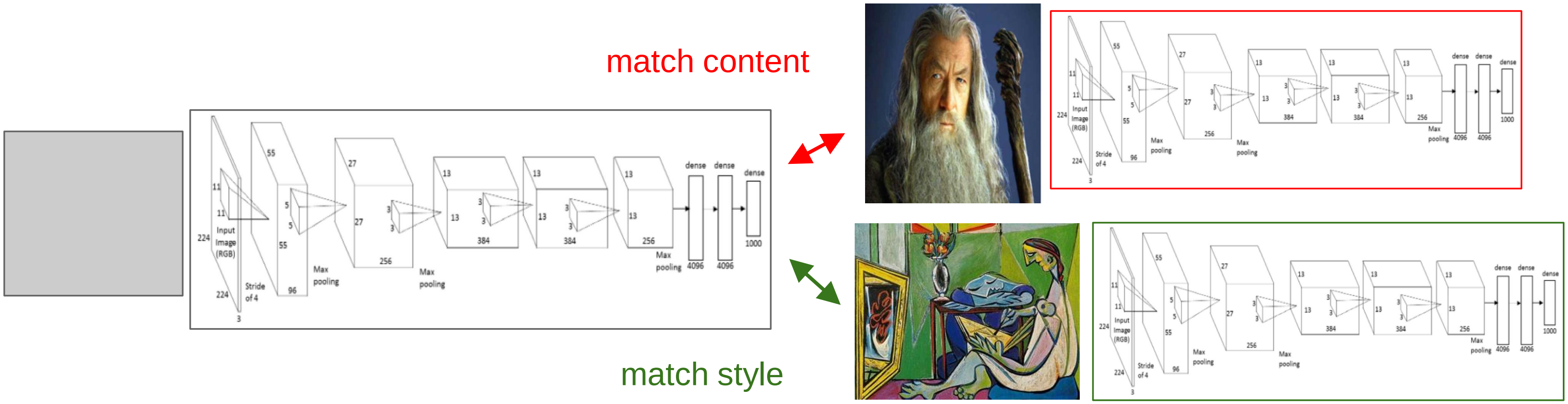
e.g.
at CONV1 layer (with [224x224x64] activations) would give a [64x64] Gram matrix of all pairwise activation covariances (summed across spatial locations)

Step 3: Optimize over image to have:

- The **content** of the content image (activations match content)
- The **style** of the style image (Gram matrices of activations match style)

$$\mathcal{L}_{total}(\vec{p},\vec{a},\vec{x}) = \alpha\mathcal{L}_{content}(\vec{p},\vec{x}) + \beta\mathcal{L}_{style}(\vec{a},\vec{x})$$

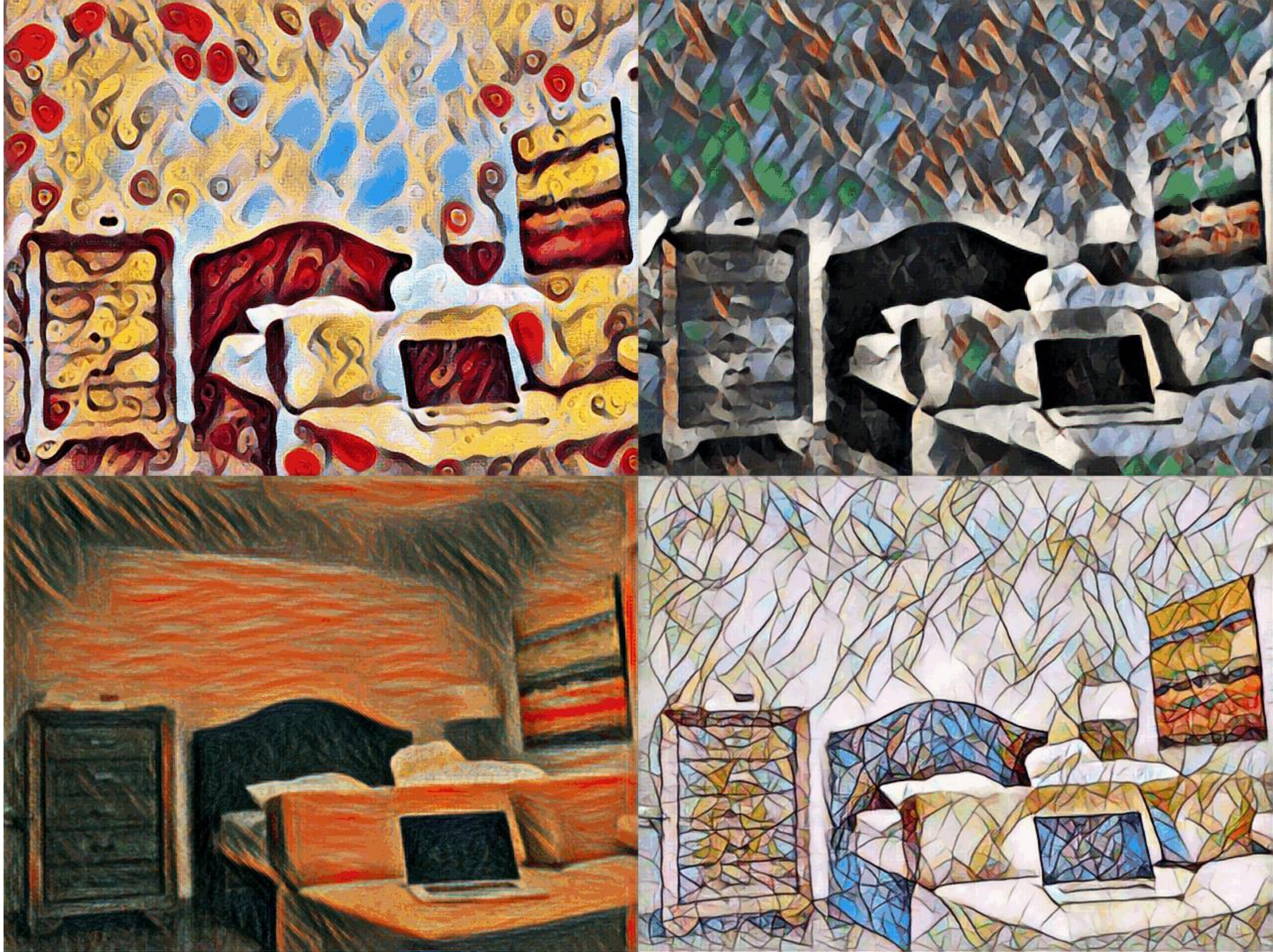
(+Total Variation regularization (maybe))



FAST neural-style

run a webcam demo in real time:

https://github.com/jcjohnson/fast-neural-style



Fast Neural Style

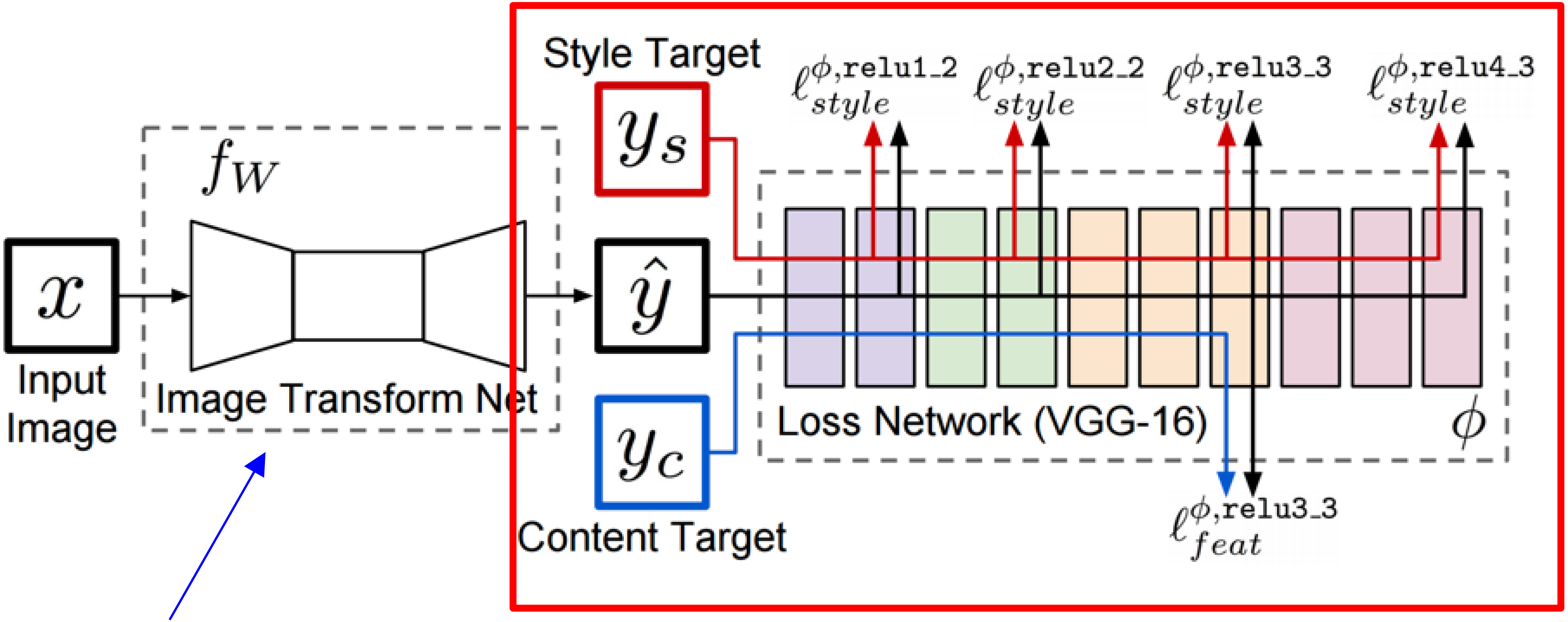
Recall for **image code inversion**:

- Mahendran and Vedaldi 2014:
  optimize over the image such that the compute code matches a target code.
- Dosovitskiy and Brox 2015:
  train a new “inversion” network from code to image using image-image loss (e.g. L2)

Use the same idea to transform Neural Style to Fast Neural Style!

Fast Neural Style Johnson et al. 2016

Can also think of this as a fixed discriminator in GAN that doesn't get trained...

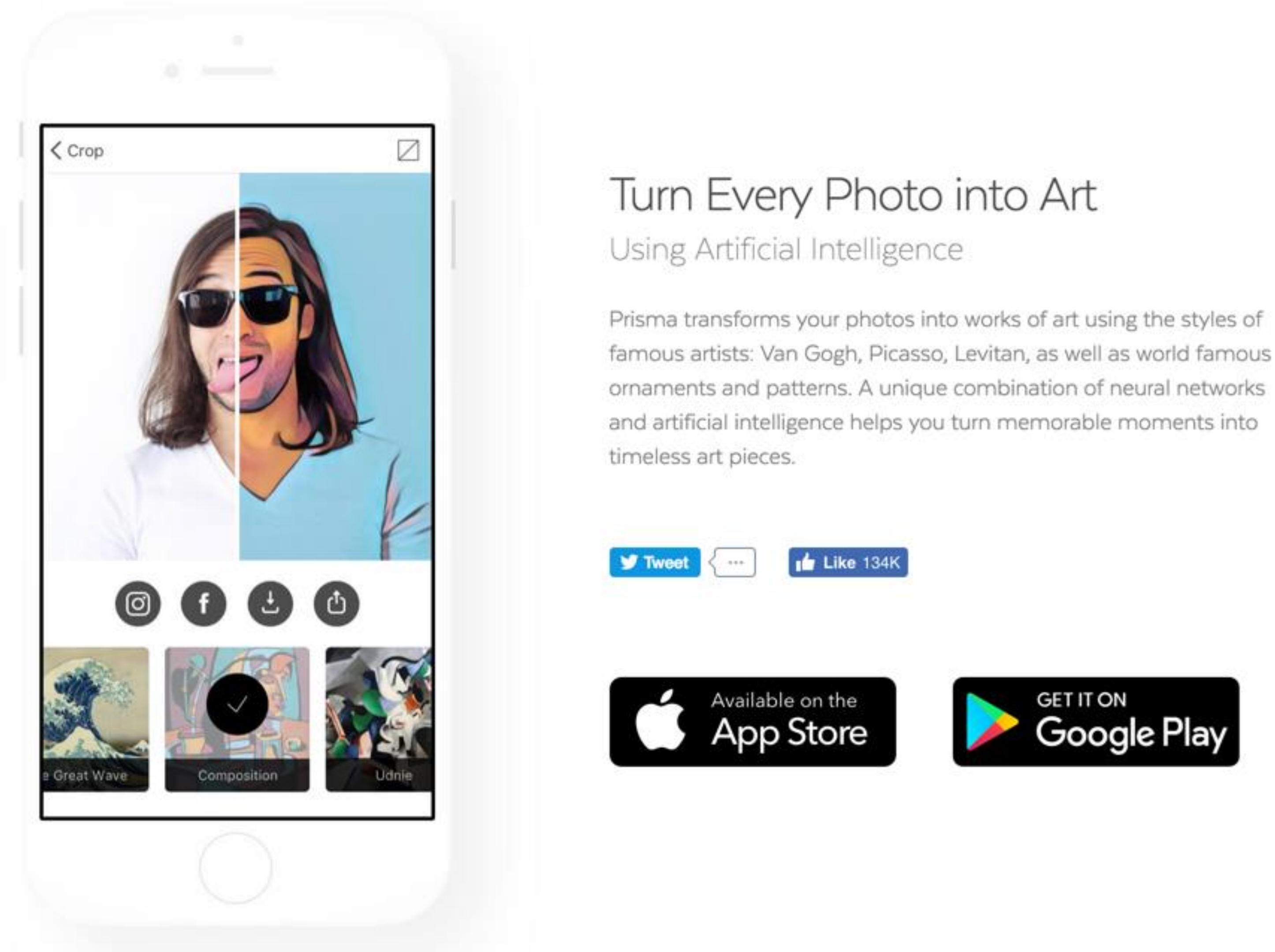


Train this network

All of this is just "neural style loss"

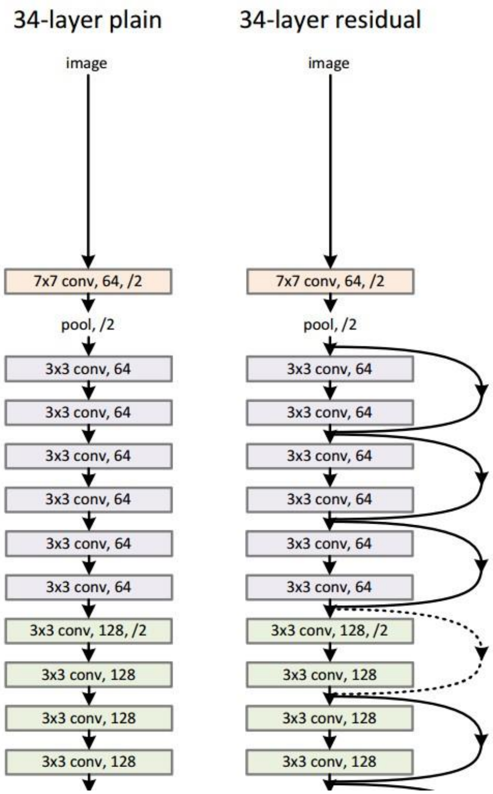
Pros: SUPER FAST (no optimization); **Cons**: network is style-specific :(

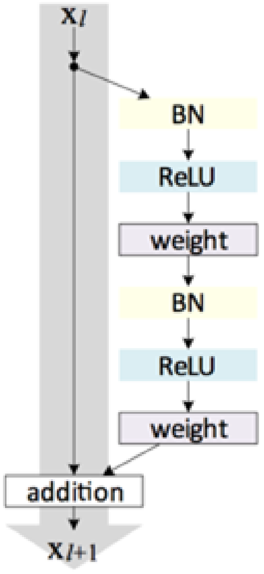
or get the
PRISMA app:



Challenge for the future: Understanding ResNets

Identity Mappings in Deep Residual Networks, He et al. 2016





(b) proposed

Summary

Visualize representations (e.g. t-SNE), use activations as feature vectors

CNNs:

- Visualize weights – easy
- Network-Centric Visualization
- Optimize class score over the input – importance to regularize
- Image-Centric Visualization
- Much easier to interpret – "localizes" activations to a real image
- Deconv approaches - guided backprop, use more info when you have it
- Initialization matters ! Can get good images from good initializations
- Full layers (codes) contain much more information and allow better image reconstruction.

Summary

Deep Dream: Feed back activations as gradients: hallucinate

Optimization for class output can be used to create “fooling images”

Neural Style transfer:

- Learn a classifer
- Take outputs from a given layer to evaluate content (code)
- Computer gram matrices from that layer to evaluate style
- Optimize images to match code and gram matrix