

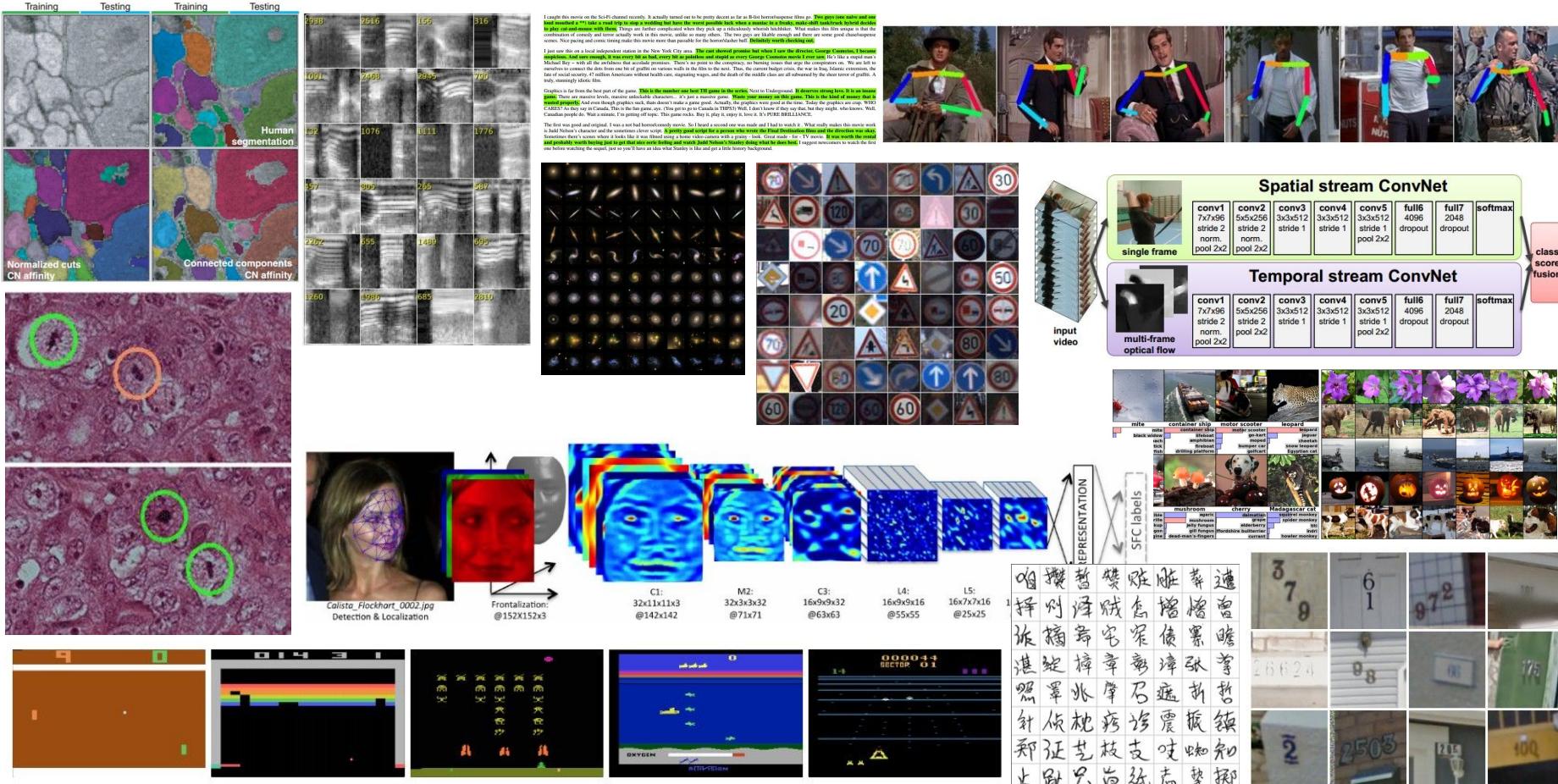
Lecture 9:

Understanding and Visualizing
Convolutional Neural Networks

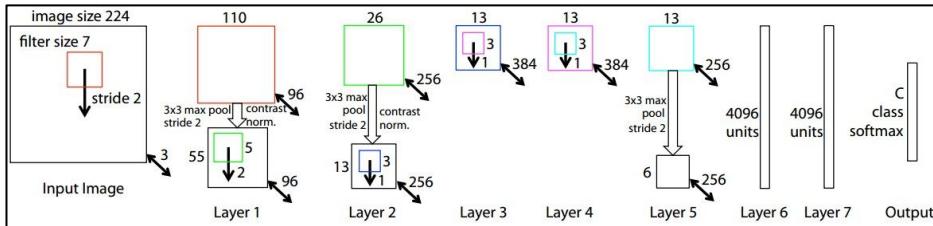
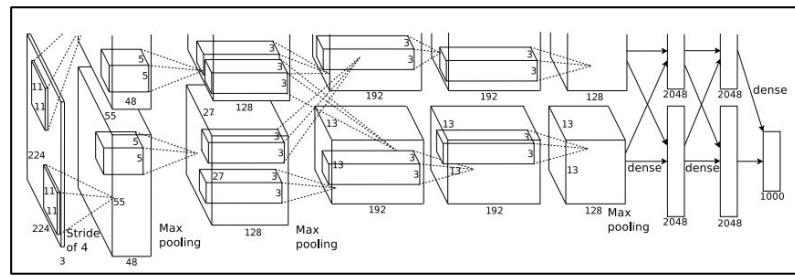
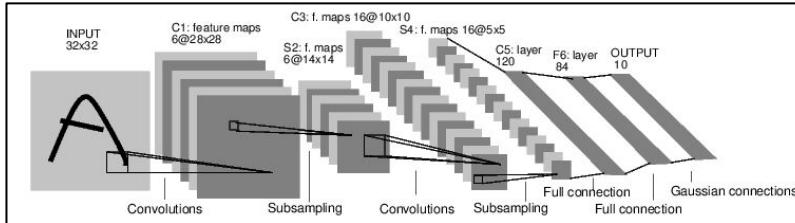
Administrative

- A1 is graded. We'll send out grades tonight (or so)
- A2 is due Feb 5 (this Friday!): **submit in Assignments tab** on CourseWork (not Dropbox)
- Midterm is Feb 10 (next Wednesday)
- Oh and pretrained ResNets were released today (152-layer ILSVRC 2015 winning ConvNets)

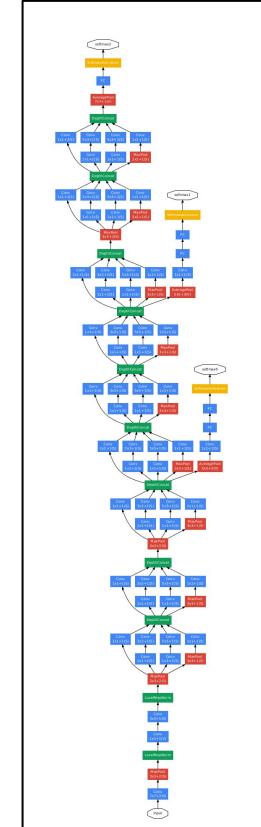
<https://github.com/KaimingHe/deep-residual-networks>



ConvNets



D	E
16 weight layers	19 weight layers
conv3-64	conv3-64
conv3-64	conv3-64
conv3-128	conv3-128
conv3-128	conv3-128
conv3-256	conv3-256
conv3-256	conv3-256
conv3-256	conv3-256
conv3-512	conv3-512
conv3-512	conv3-512
conv3-512	conv3-512
conv3-512	conv3-512
conv3-512	conv3-512
conv3-512	conv3-512
maxpool	
FC-4096	
FC-4096	
FC-1000	
soft-max	



Computer Vision Tasks

Classification



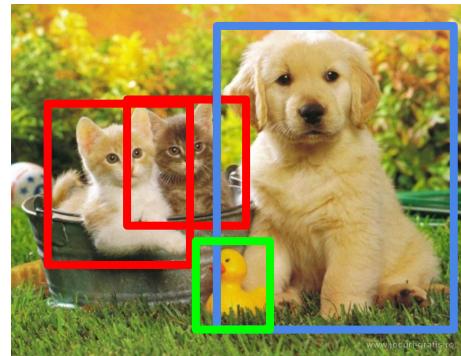
CAT

Classification + Localization



CAT

Object Detection



CAT, DOG, DUCK

Instance Segmentation



CAT, DOG, DUCK

Single object

Multiple objects

Understanding ConvNets

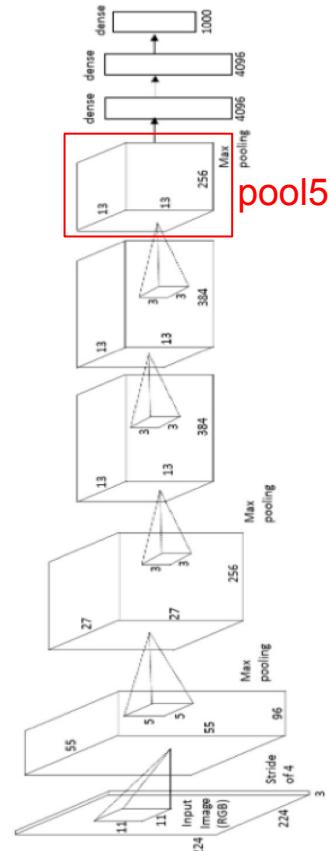
- Visualize patches that maximally activate neurons
- Visualize the weights
- Visualize the representation space (e.g. with t-SNE)
- Occlusion experiments
- Human experiment comparisons
- Deconv approaches (single backward pass)
- Optimization over image approaches (optimization)

Visualize patches that maximally activate neurons

one-stream AlexNet



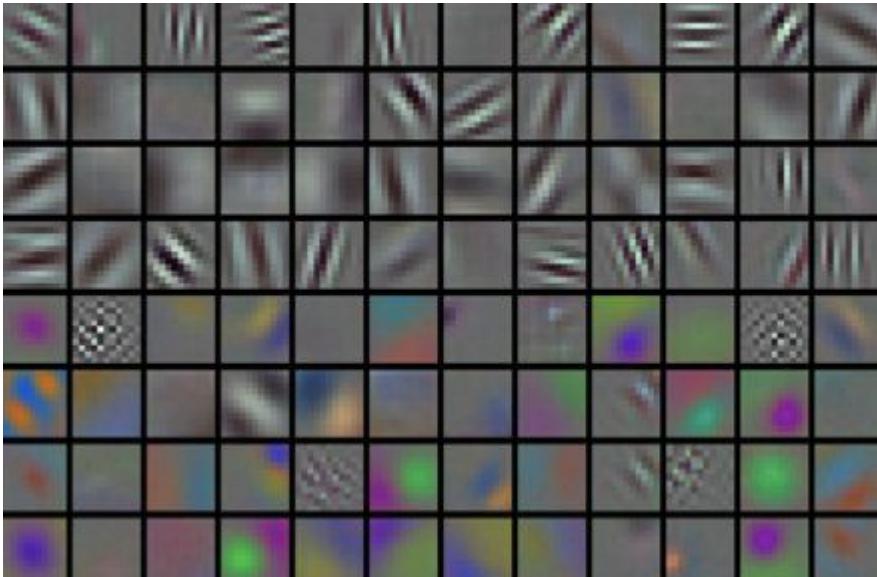
Figure 4: Top regions for six pool_5 units. Receptive fields and activation values are drawn in white. Some units are aligned to concepts, such as people (row 1) or text (4). Other units capture texture and material properties, such as dot arrays (2) and specular reflections (6).



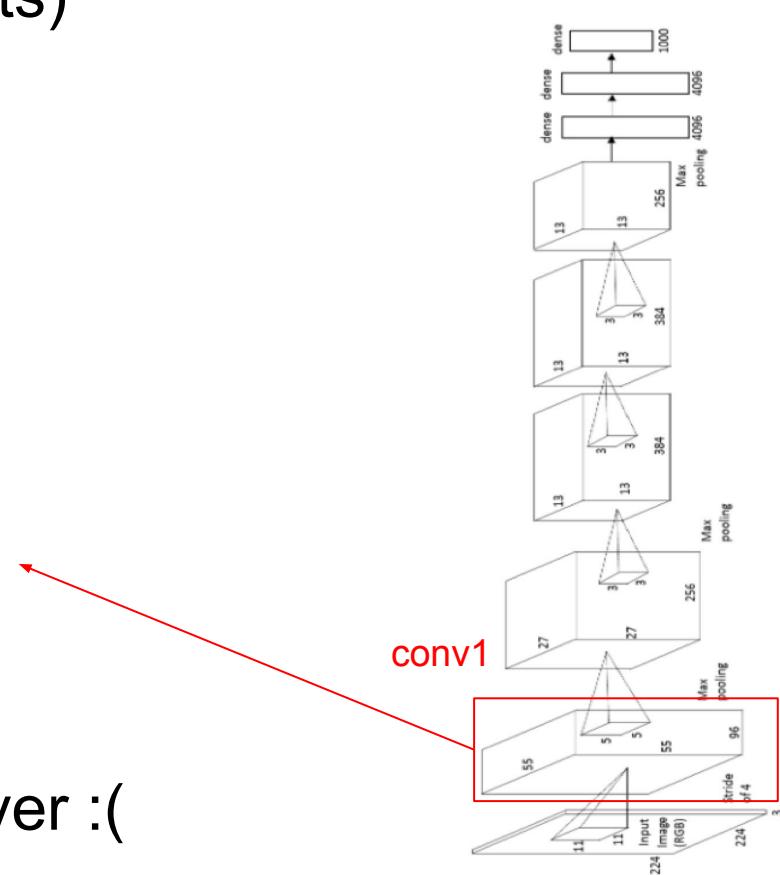
*Rich feature hierarchies for accurate object detection and semantic segmentation
[Girshick, Donahue, Darrell, Malik]*

Visualize the filters/kernels (raw weights)

one-stream AlexNet



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)

Weights:


layer 1 weights

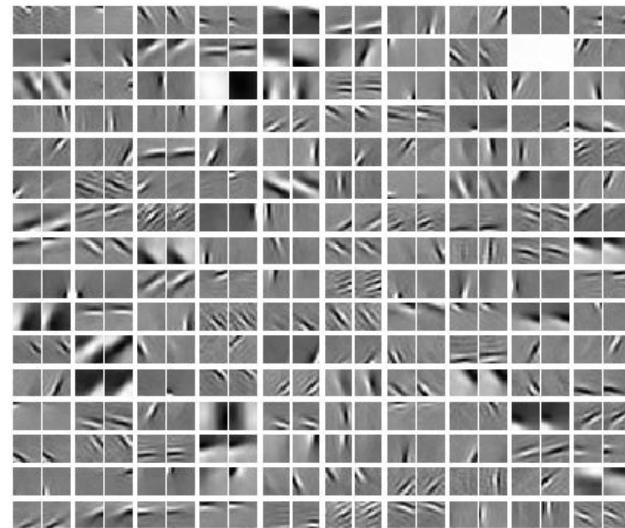
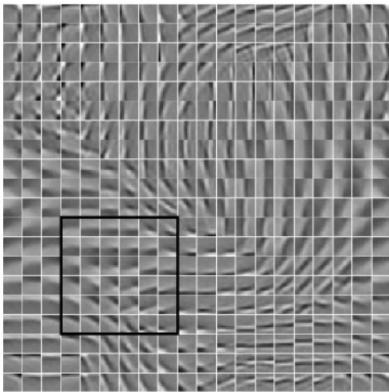
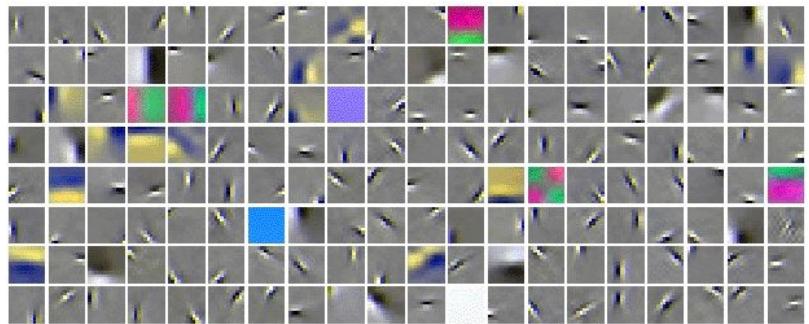
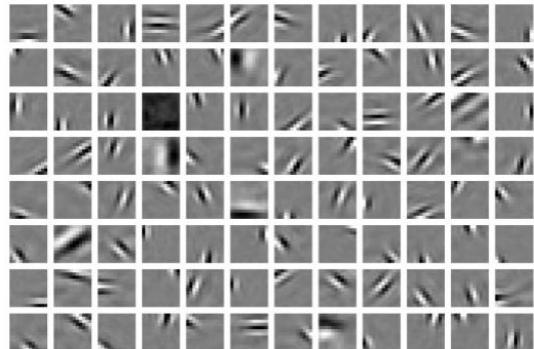
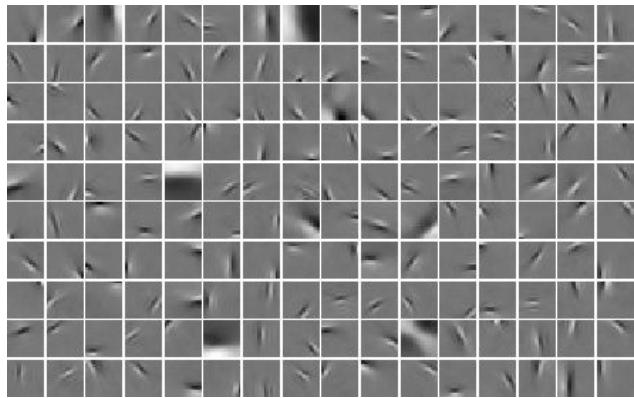
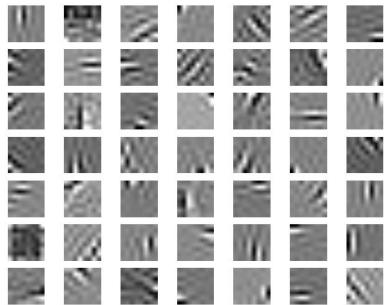
Weights:


layer 2 weights

Weights:


layer 3 weights

The gabor-like filters fatigue

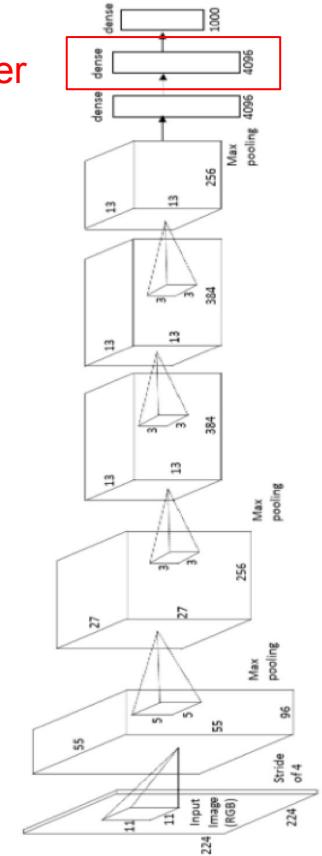


Visualizing the representation

fc7 layer

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

can collect the code for many images



Visualizing the representation

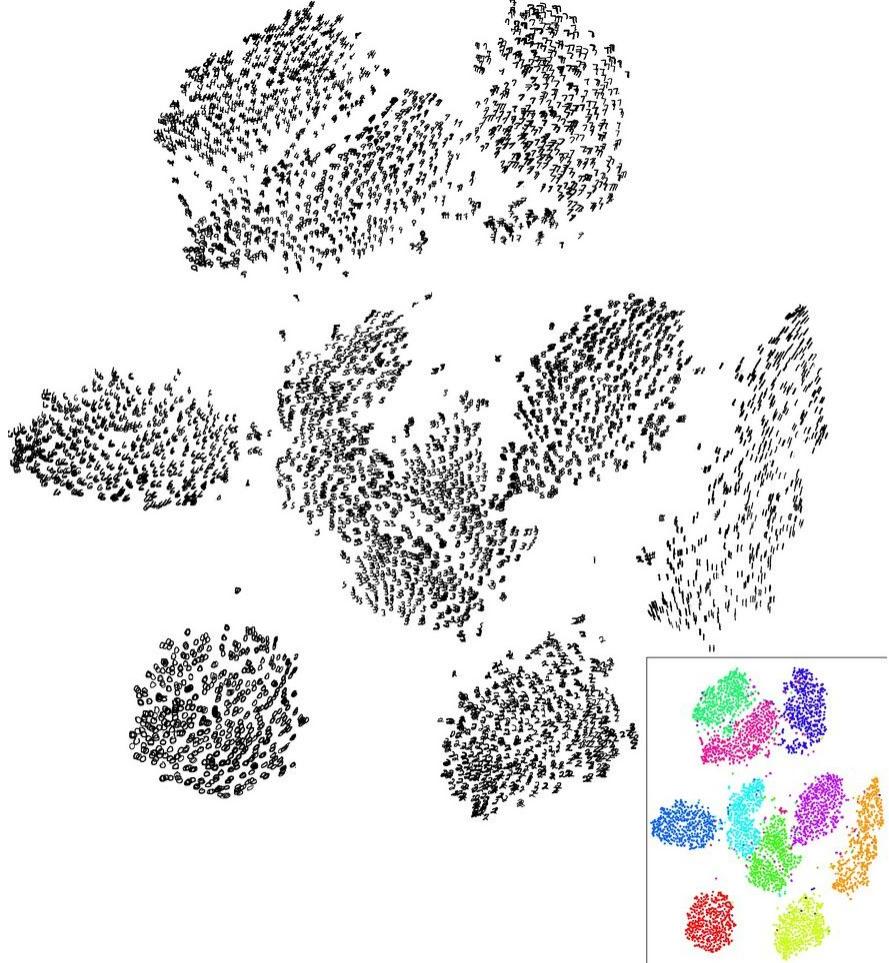
t-SNE visualization

[van der Maaten & Hinton]

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 digits
(0-9) in 2D



t-SNE visualization:

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

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



Occlusion experiments

[Zeiler & Fergus 2013]

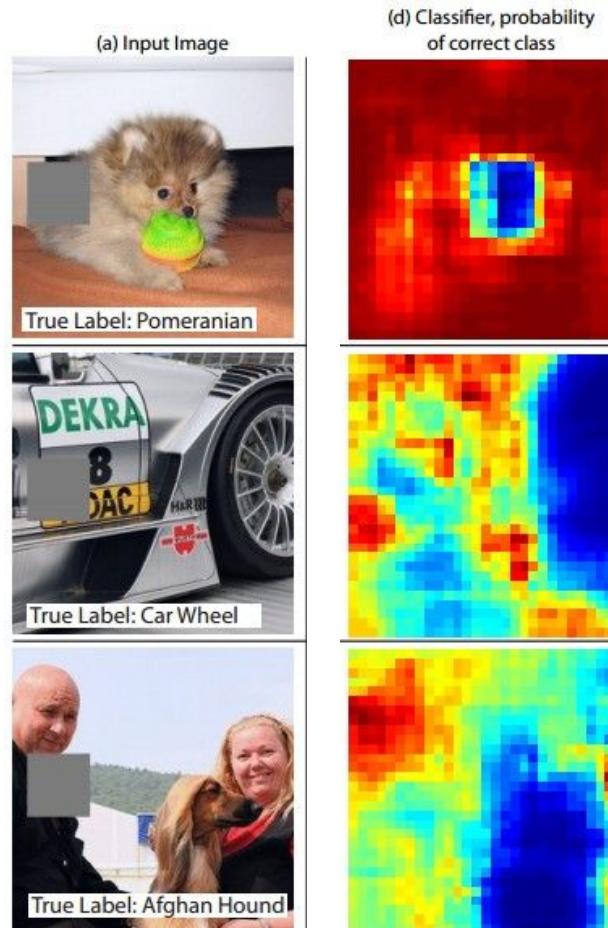


(d) Classifier, probability
of correct class

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

Occlusion experiments

[Zeiler & Fergus 2013]



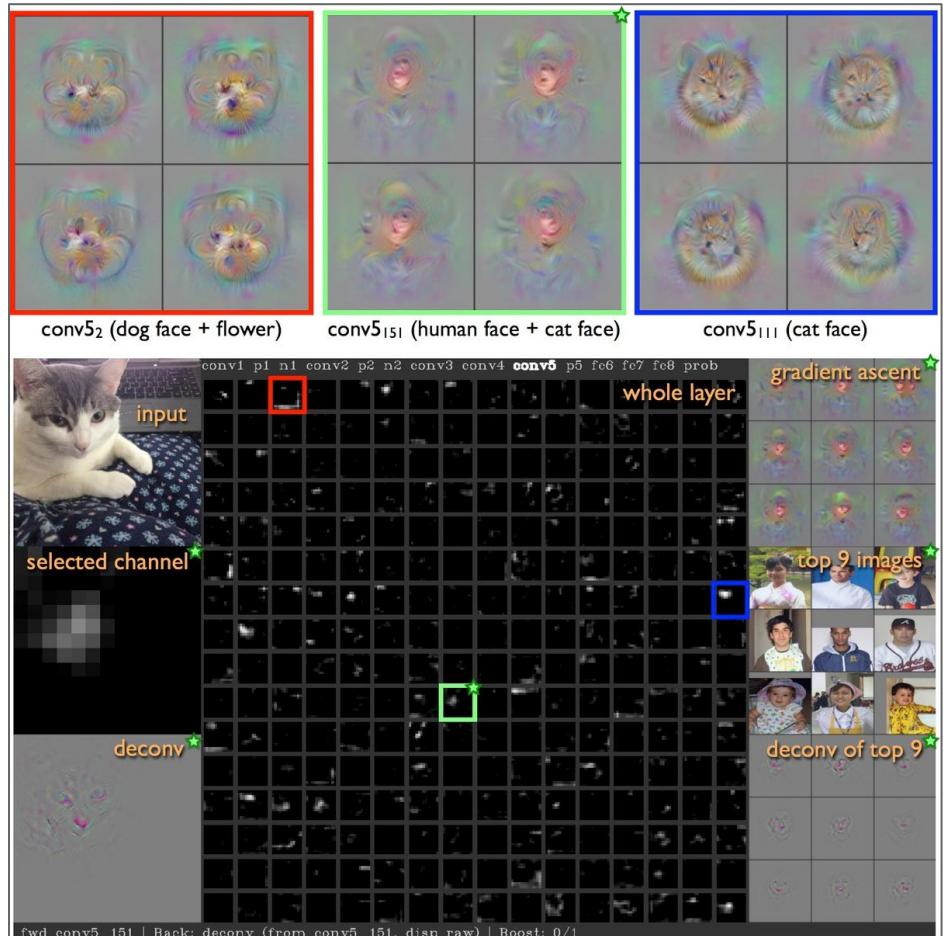
(as a function of the position of the square of zeros in the original image)

Visualizing Activations

<http://yosinski.com/deepvis>

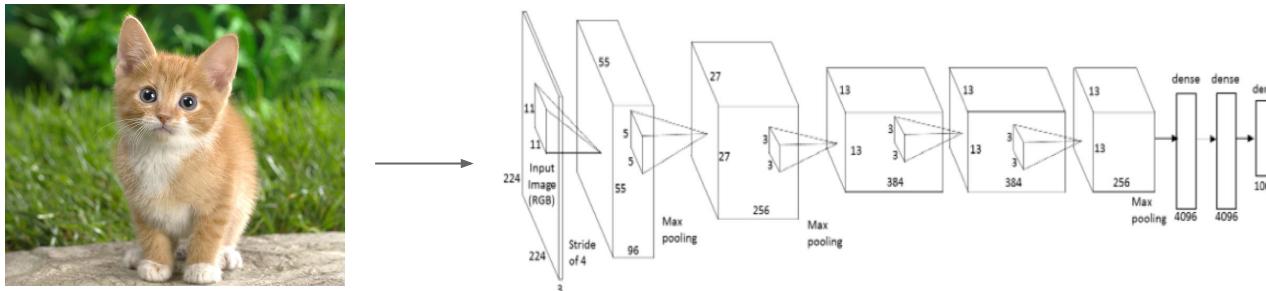
YouTube video

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



Deconv approaches

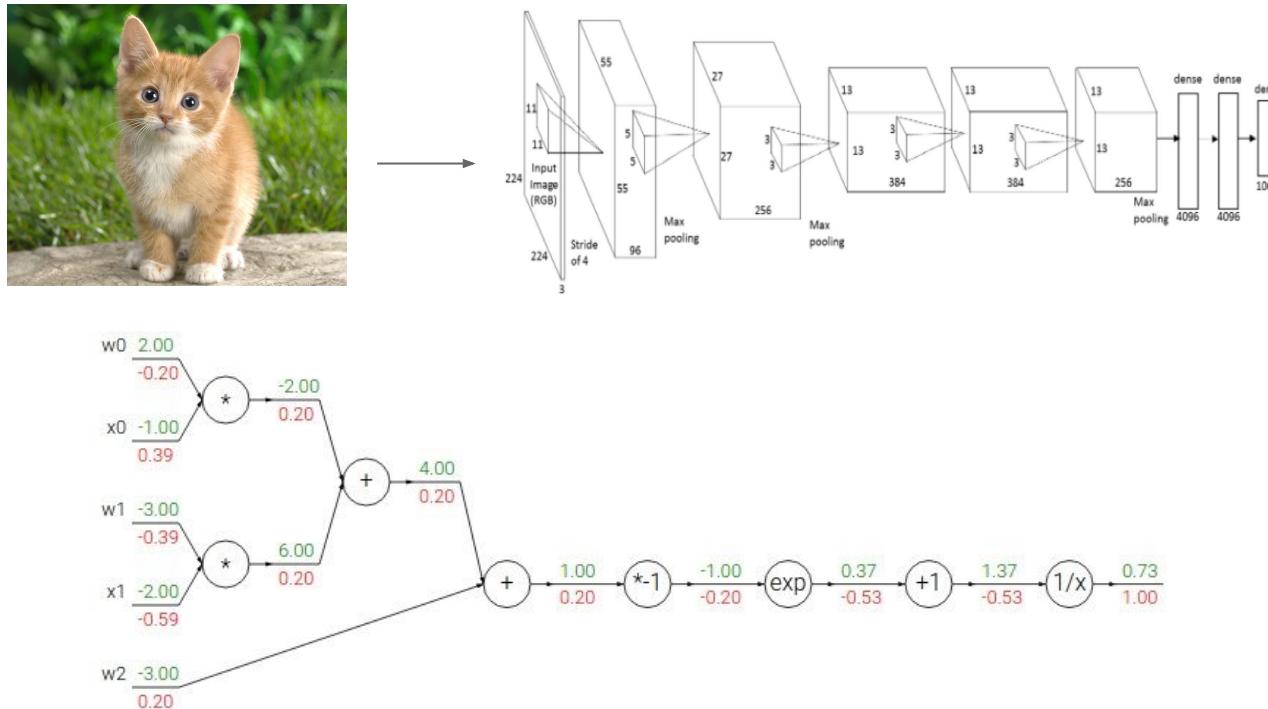
1. Feed image into net



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

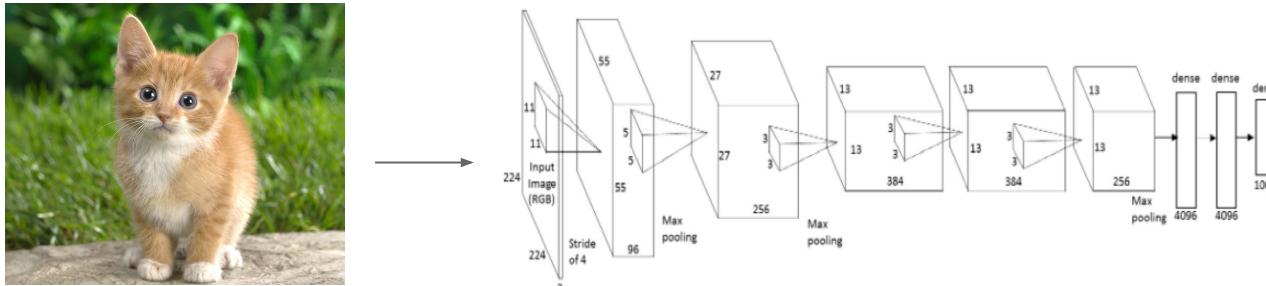
Deconv approaches

1. Feed image into net

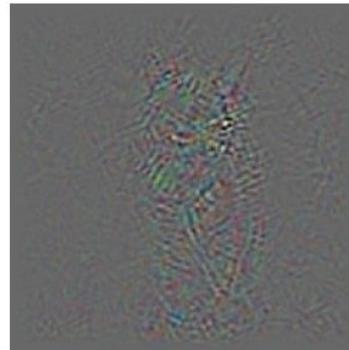


Deconv approaches

1. Feed image into net

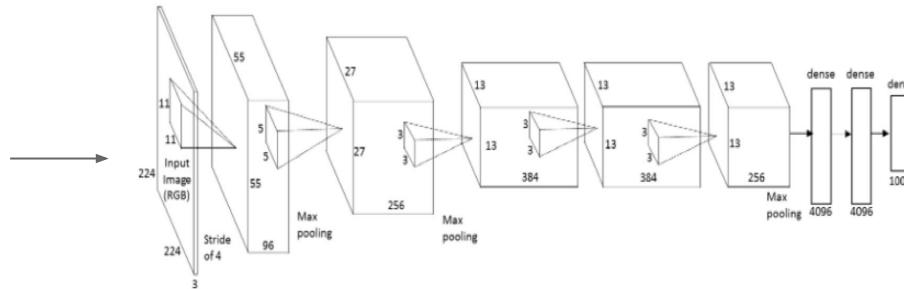


2. Pick a layer, set the gradient there to be all zero except for one 1 for some neuron of interest
3. Backprop to image:



Deconv approaches

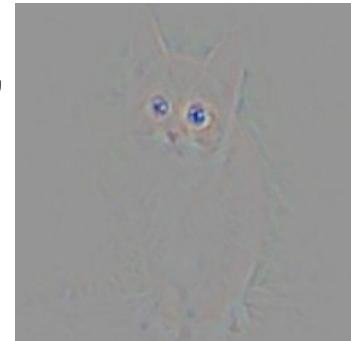
1. Feed image into net



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

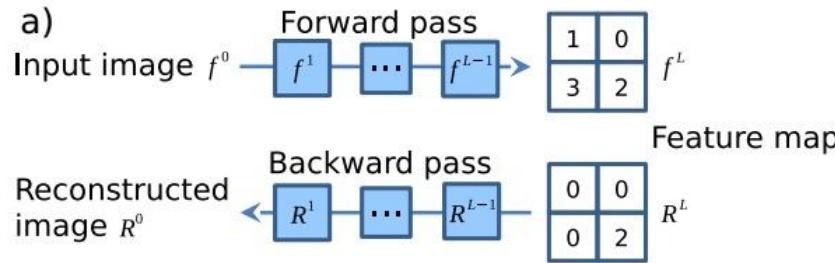


Deconv approaches

[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]

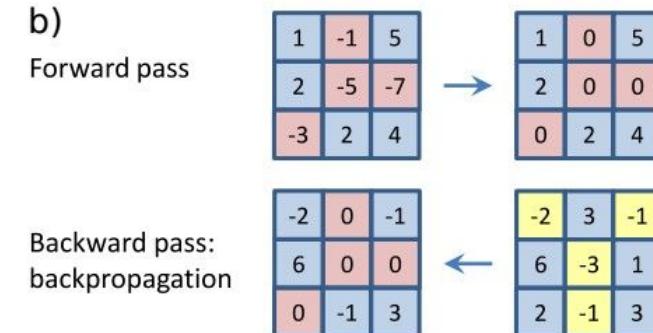
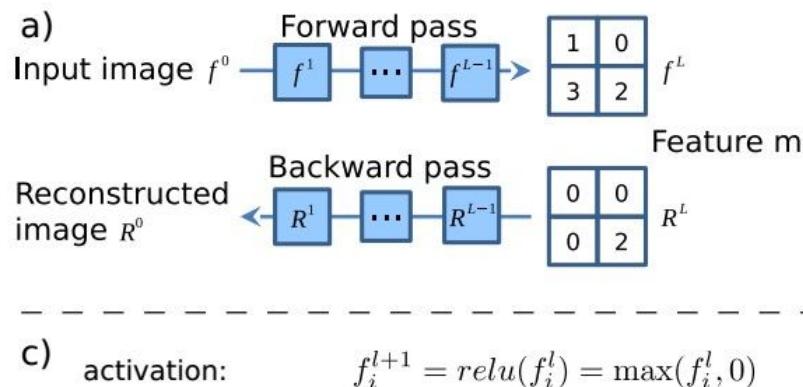


Deconv approaches

[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]



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

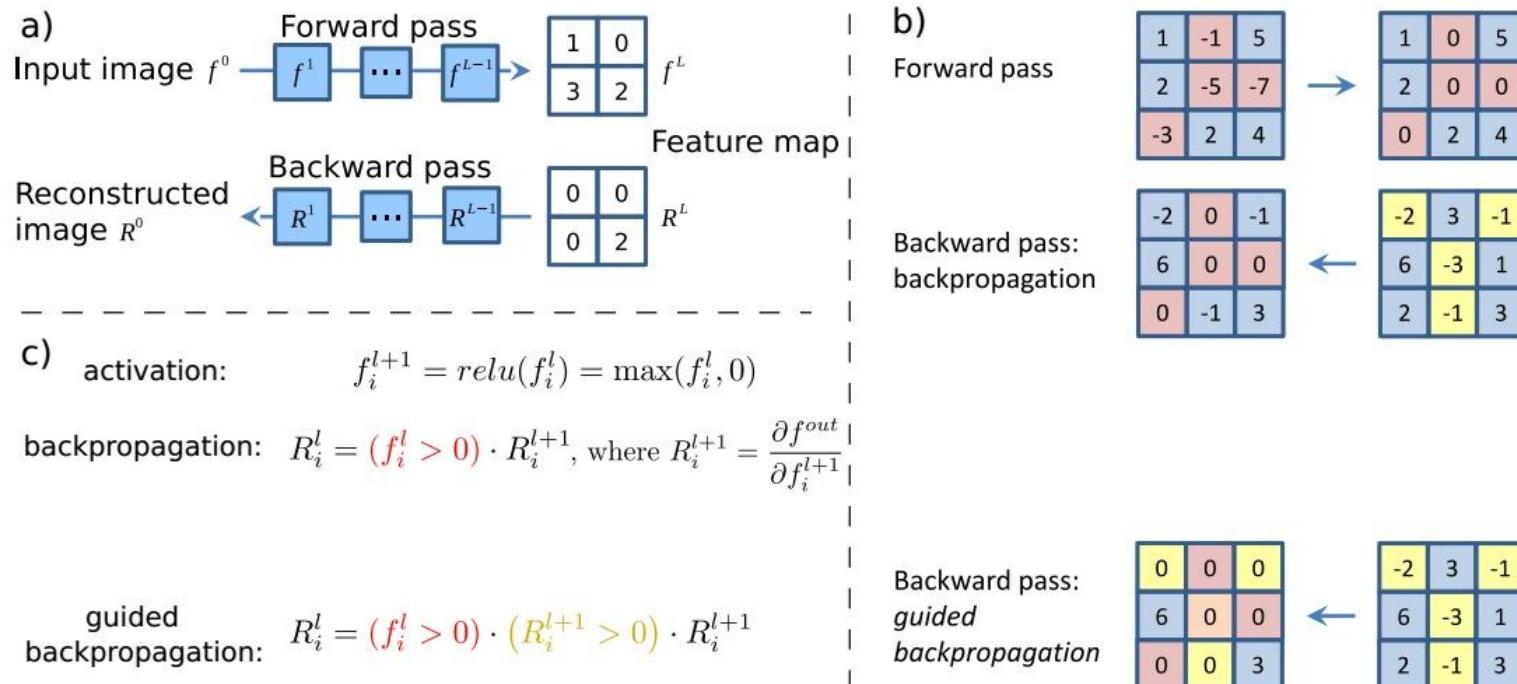
Backward pass for a ReLU (will be changed in Guided Backprop)

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]

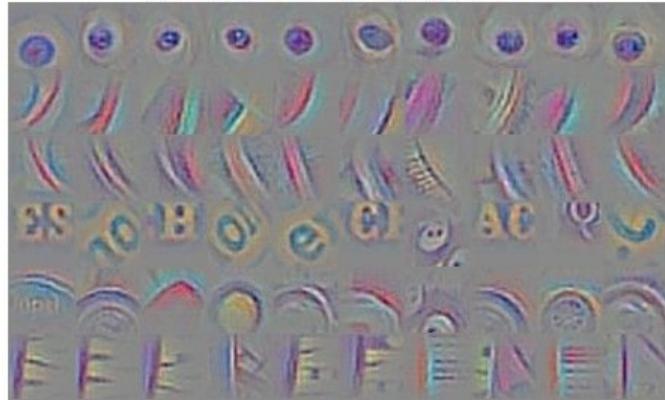


Visualization of patterns learned by the layer **conv6** (top) and layer **conv9** (bottom) of the network trained on ImageNet.

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.

guided backpropagation



guided backpropagation



corresponding image crops



corresponding image crops



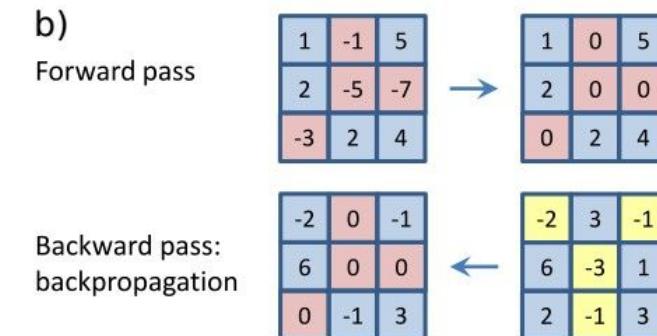
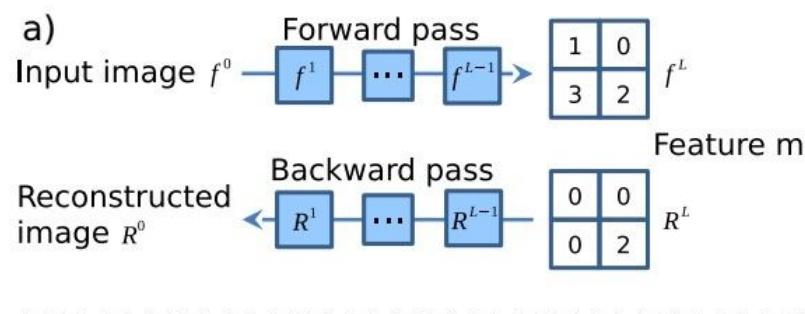
[*Striving for Simplicity: The all convolutional net*, Springenberg, Dosovitskiy, et al., 2015]

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]



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

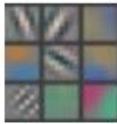
backpropagation: $R_i^l = (\mathbf{f}_i^l > 0) \cdot R_i^{l+1}$, where $R_i^{l+1} = \frac{\partial f^{out}}{\partial f_i^{l+1}}$

backward 'deconvnet': $R_i^l = (\mathbf{R}_i^{l+1} > 0) \cdot R_i^{l+1}$

guided backpropagation: $R_i^l = (\mathbf{f}_i^l > 0) \cdot (\mathbf{R}_i^{l+1} > 0) \cdot R_i^{l+1}$



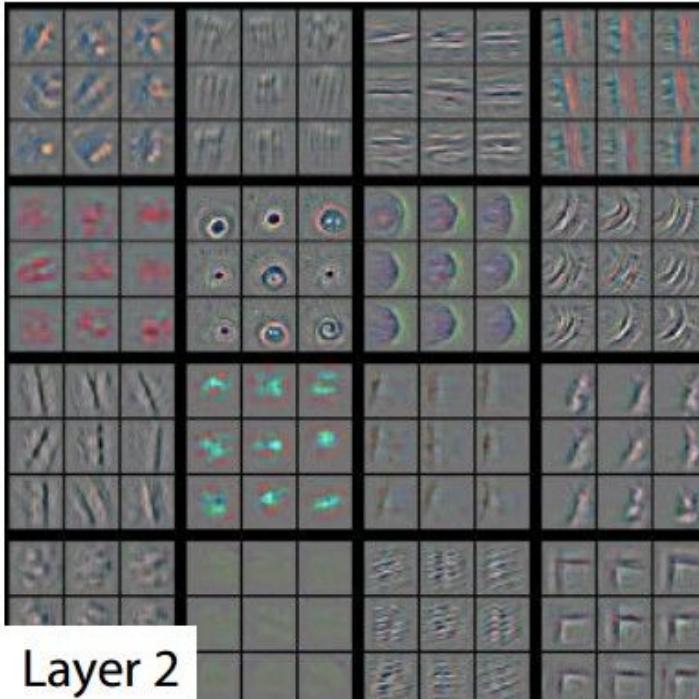
Visualizing arbitrary neurons along the way to the top...



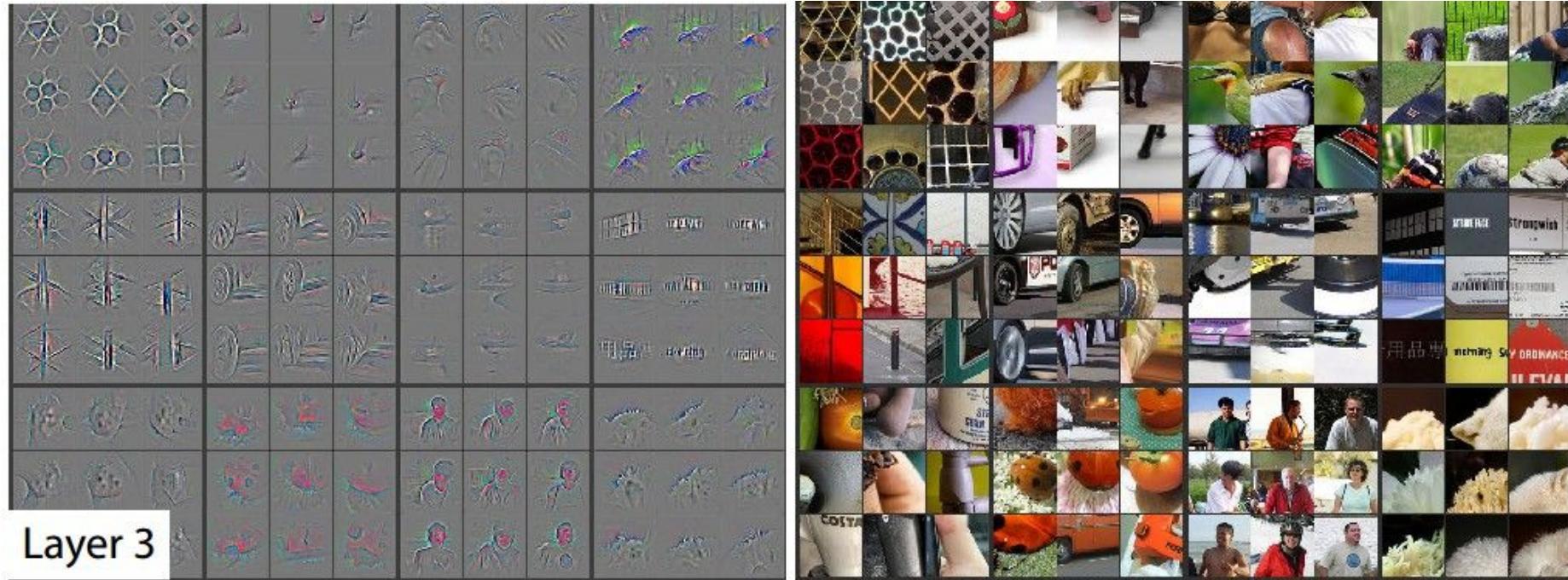
Layer 1



Layer 2



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



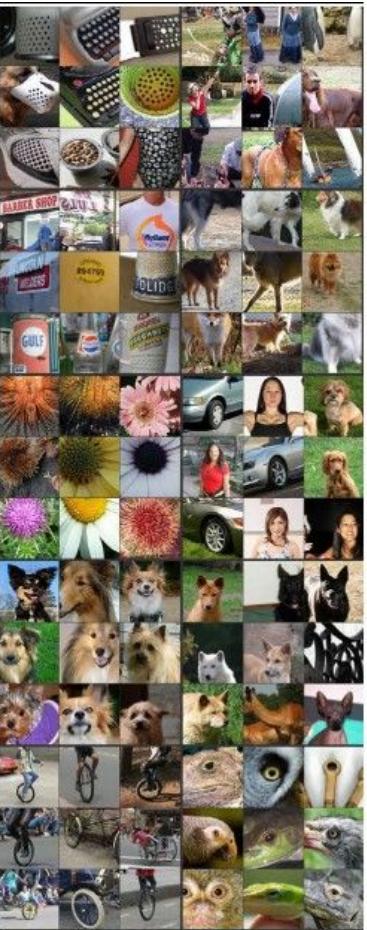
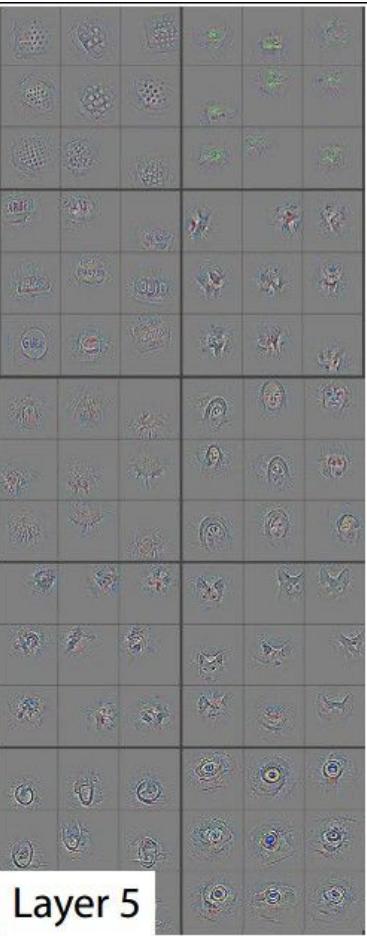
Visualizing
arbitrary
neurons along
the way to the
top...



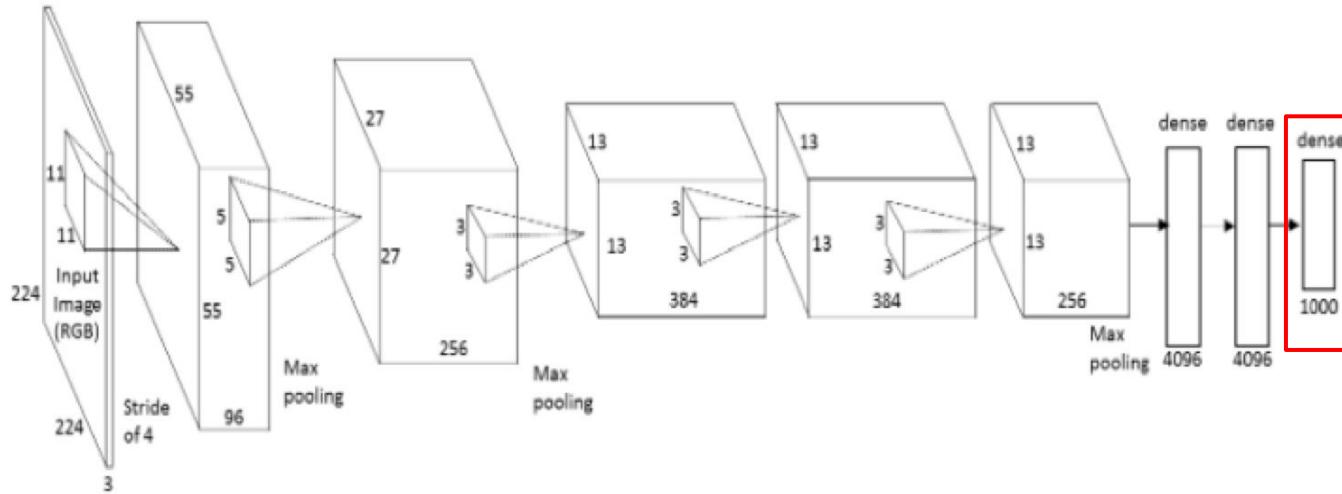
Layer 4



Layer 5



Optimization to Image

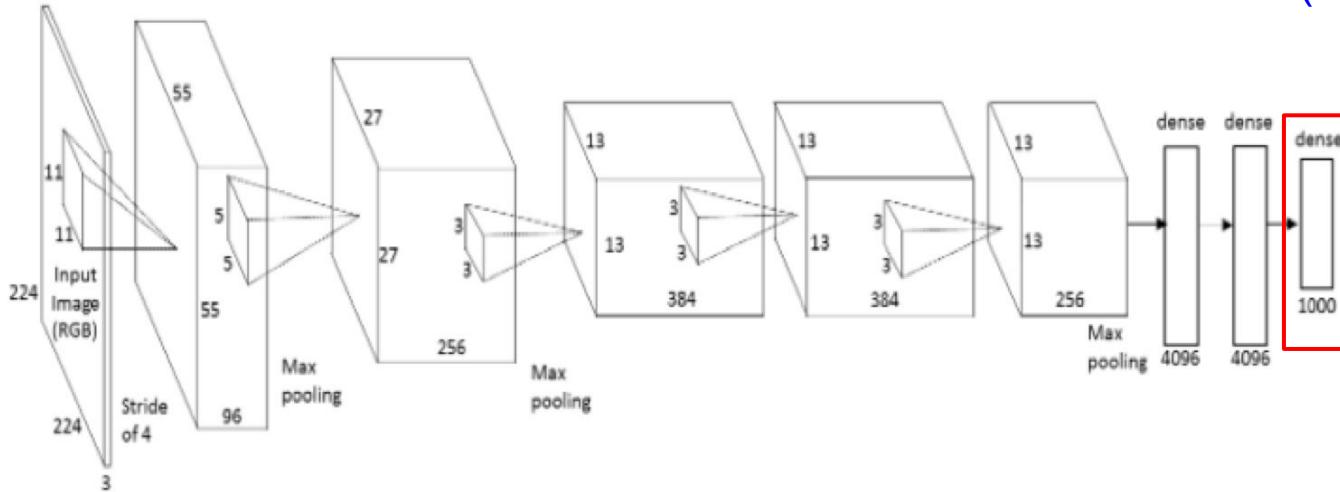


Q: can we find an image that maximizes some class score?

Optimization to Image

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

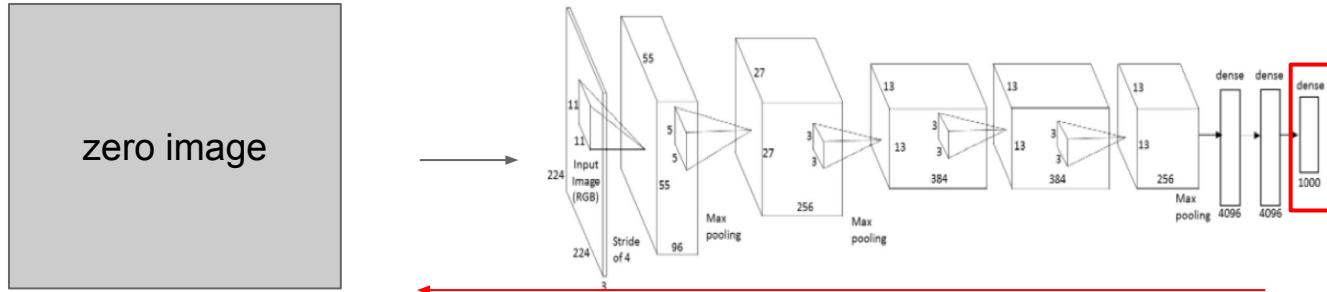
score for class c (before Softmax)



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

Optimization to Image

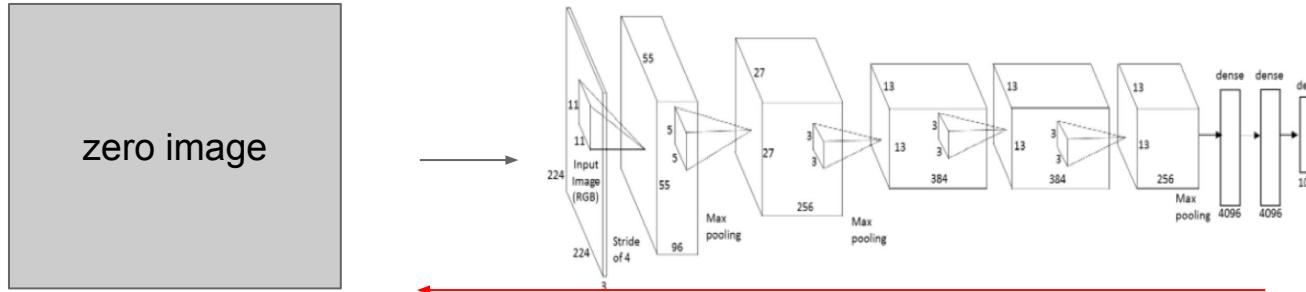
1. feed in
zeros.



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

Optimization to Image

1. feed in zeros.



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

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

score for class c (before Softmax)

1. Find images that maximize some class score:



dumbbell



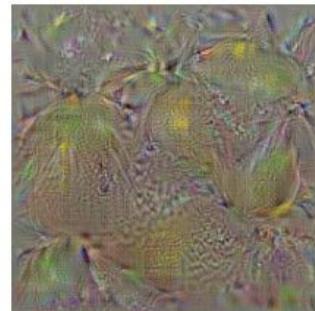
cup



dalmatian



bell pepper

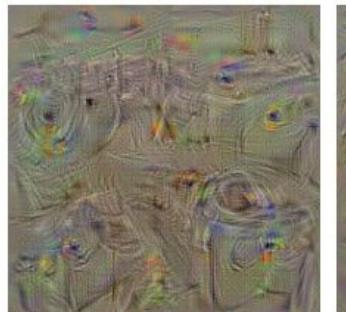


lemon

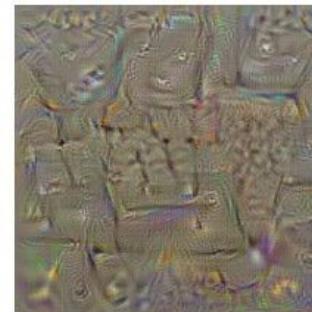


husky

1. Find images that maximize some class score:



washing machine



computer keyboard



kit fox



goose



ostrich



limousine

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)



M = ?

2. Visualize the Data gradient:

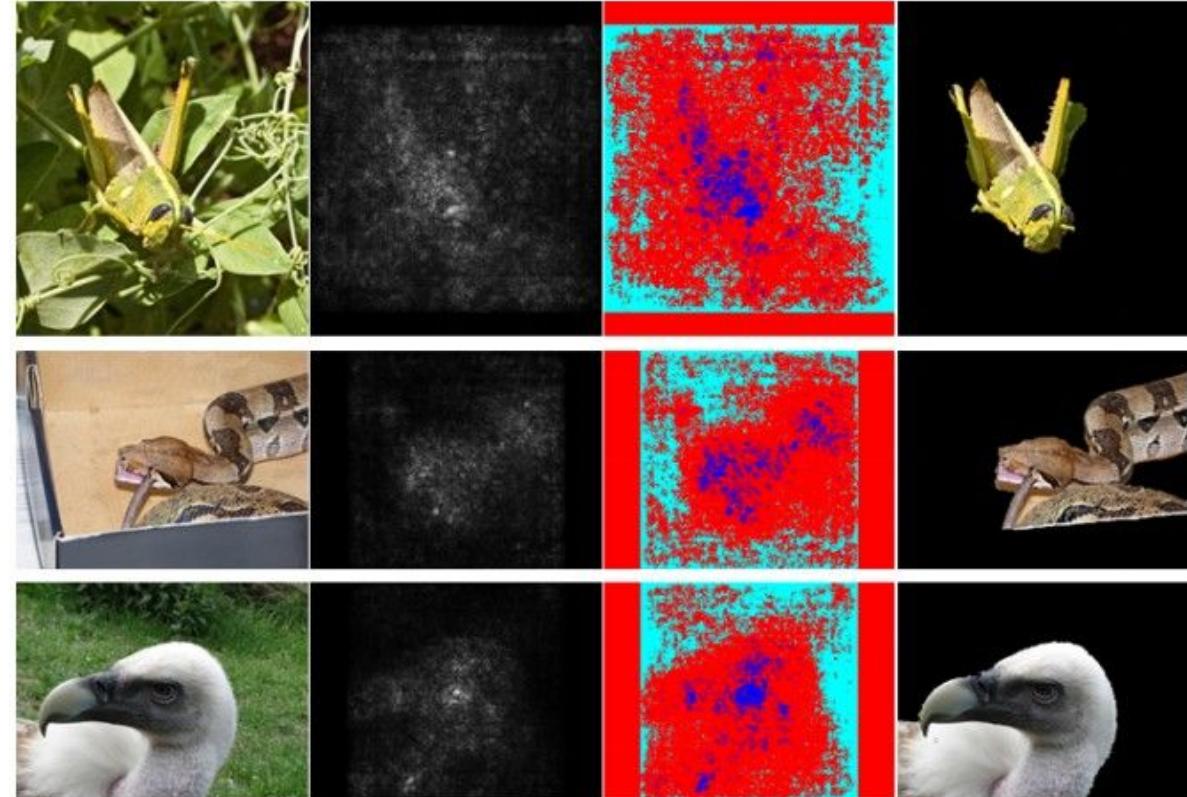
(note that the gradient on data has three channels.
Here they visualize M, s.t.:

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

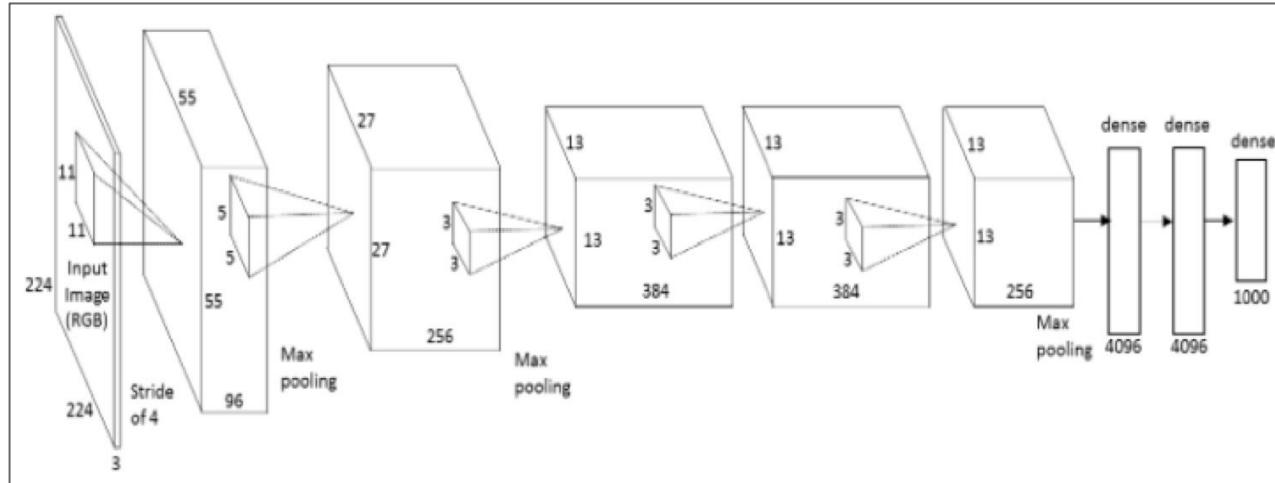
(at each pixel take abs val, and max over channels)



- Use **grabcut** for segmentation



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”

Proposed a different form of regularizing the image

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



More explicit scheme:

Repeat:

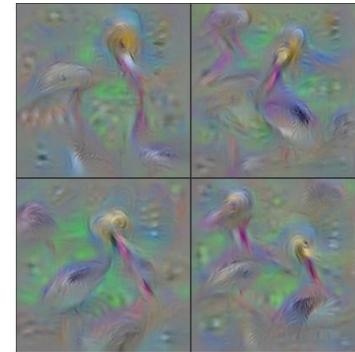
- Update the image \mathbf{x} with gradient from some unit of interest
- Blur \mathbf{x} a bit
- Take any pixel with small norm to zero (to encourage sparsity)

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

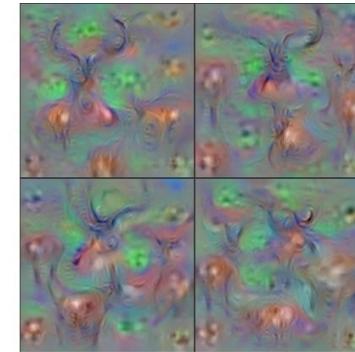
<http://yosinski.com/deepvis>



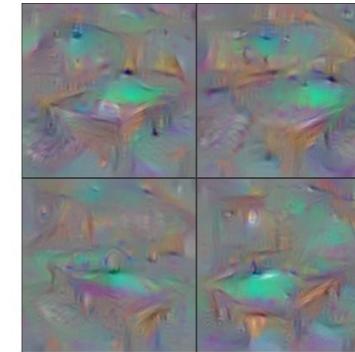
Flamingo



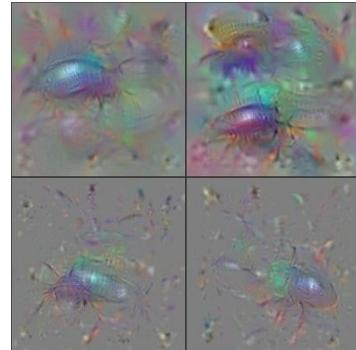
Pelican



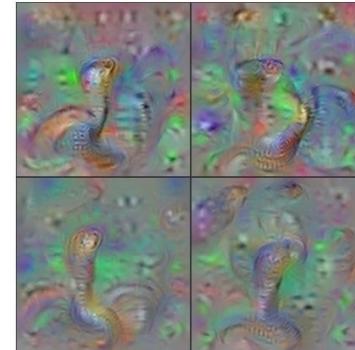
Hartebeest



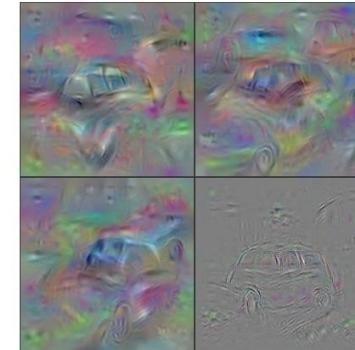
Billiard Table



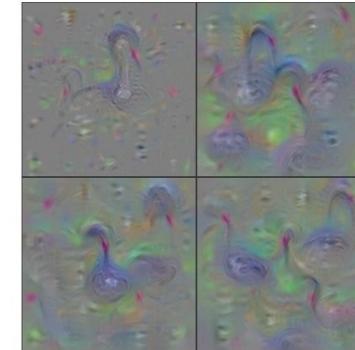
Ground Beetle



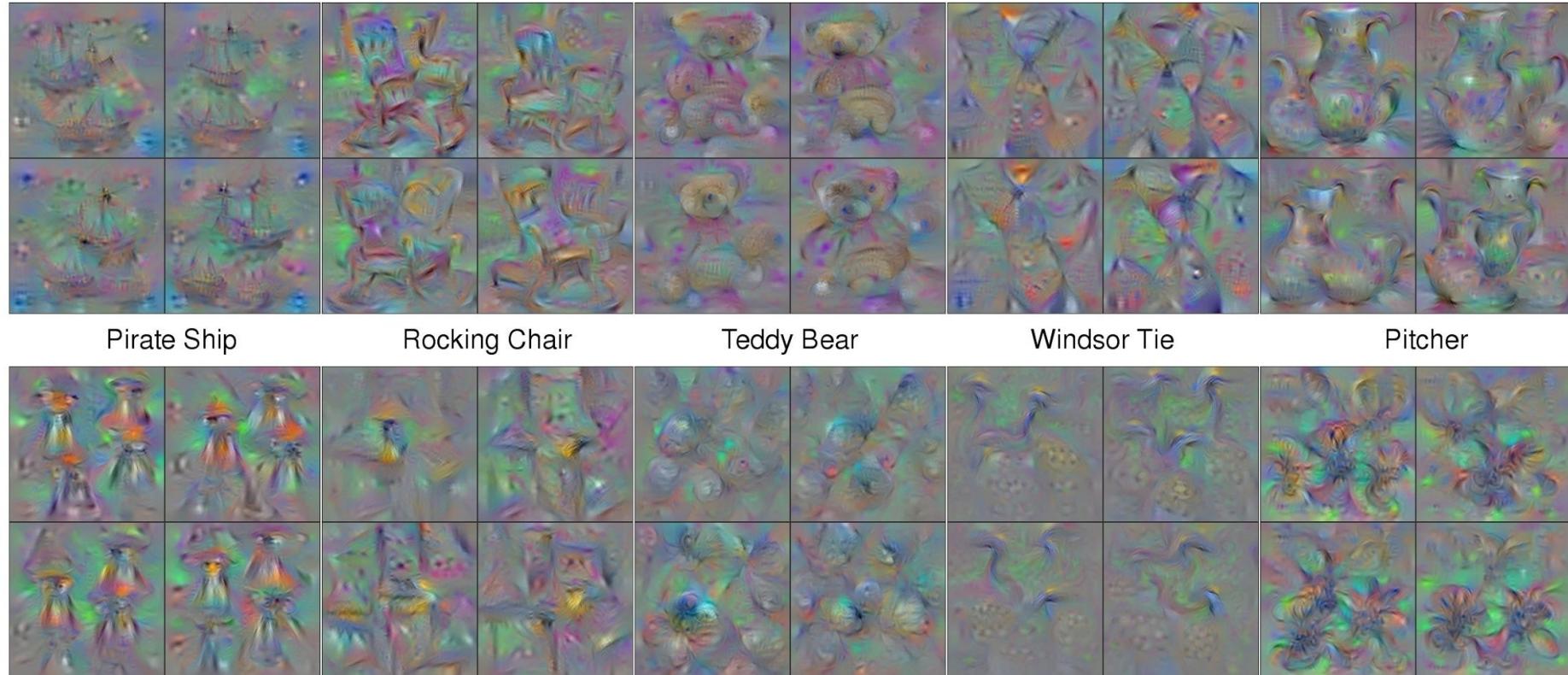
Indian Cobra

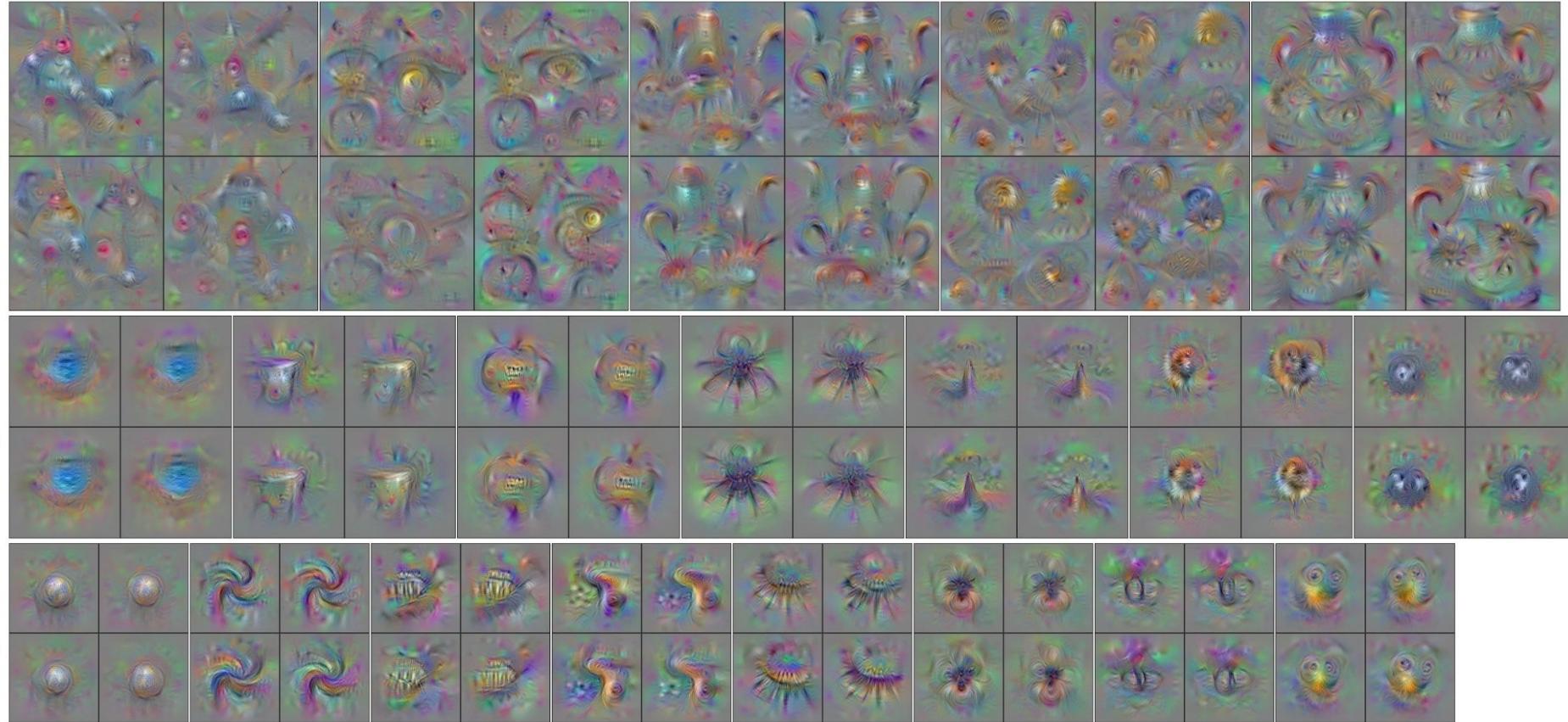


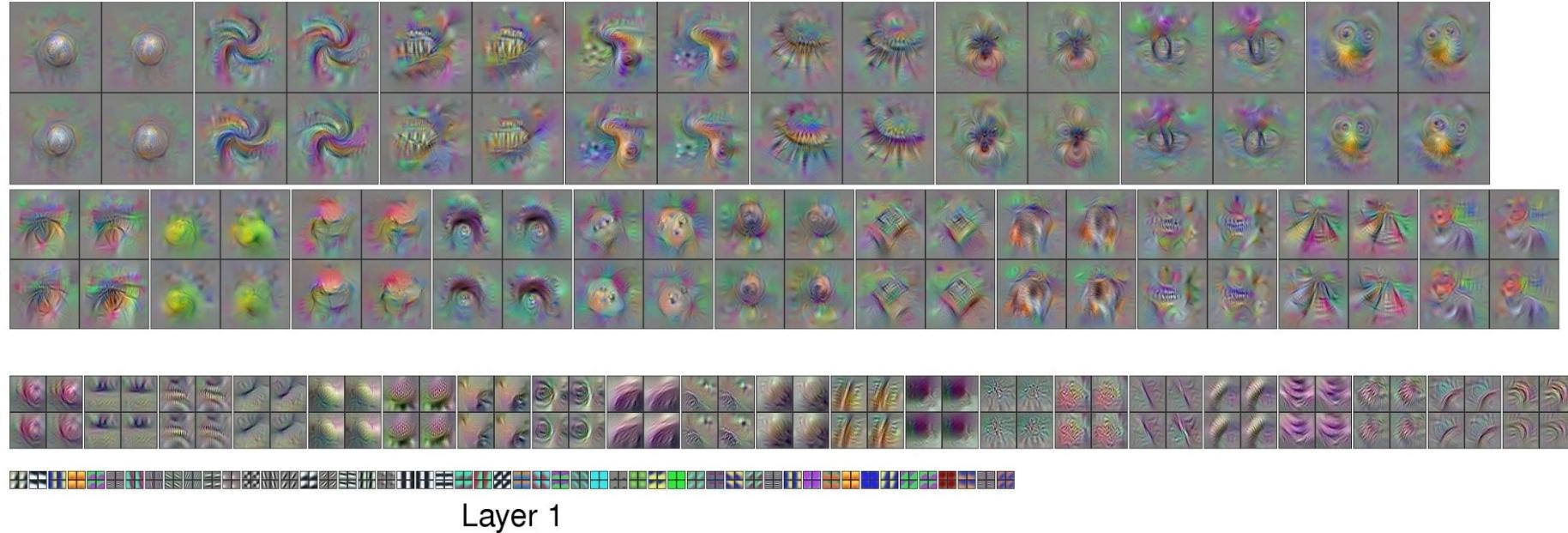
Station Wagon



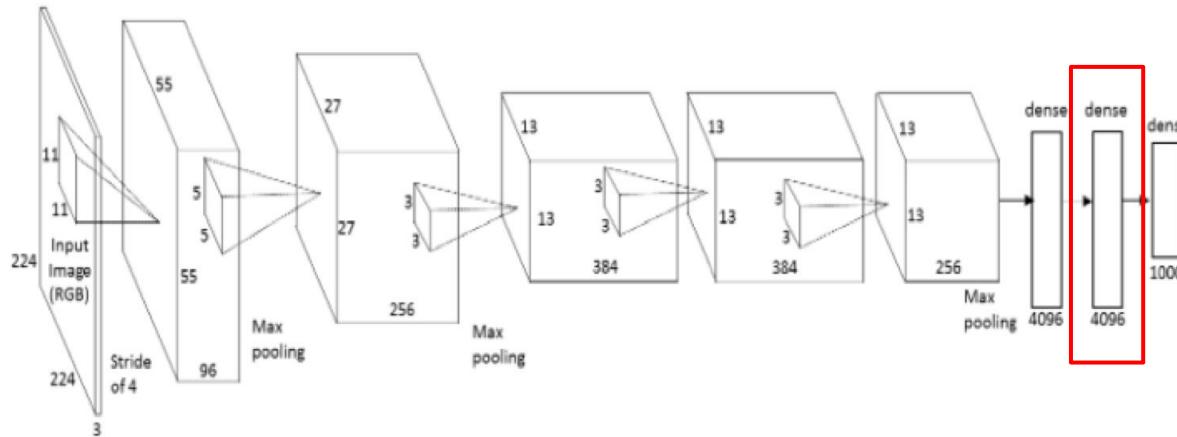
Black Swan







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



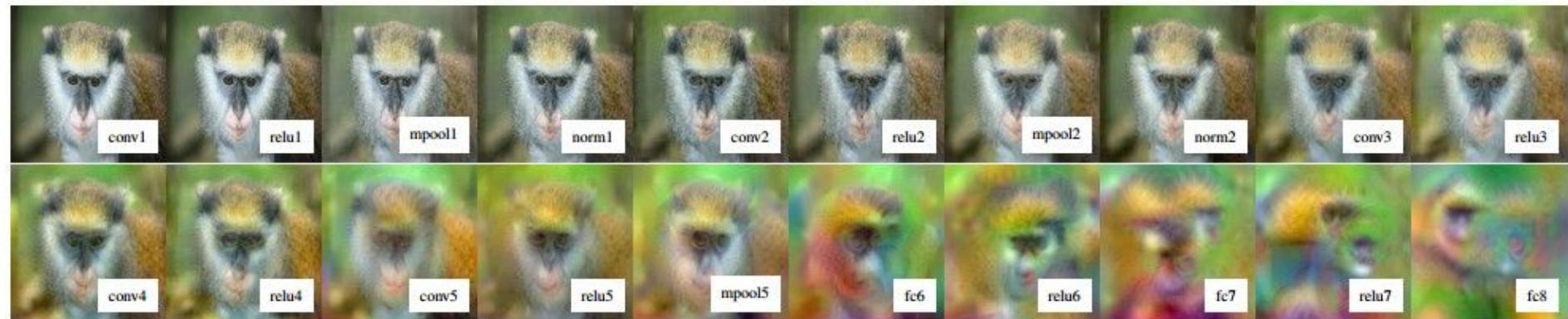
reconstructions
from the 1000
log probabilities
for ImageNet
(ILSVRC)
classes

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



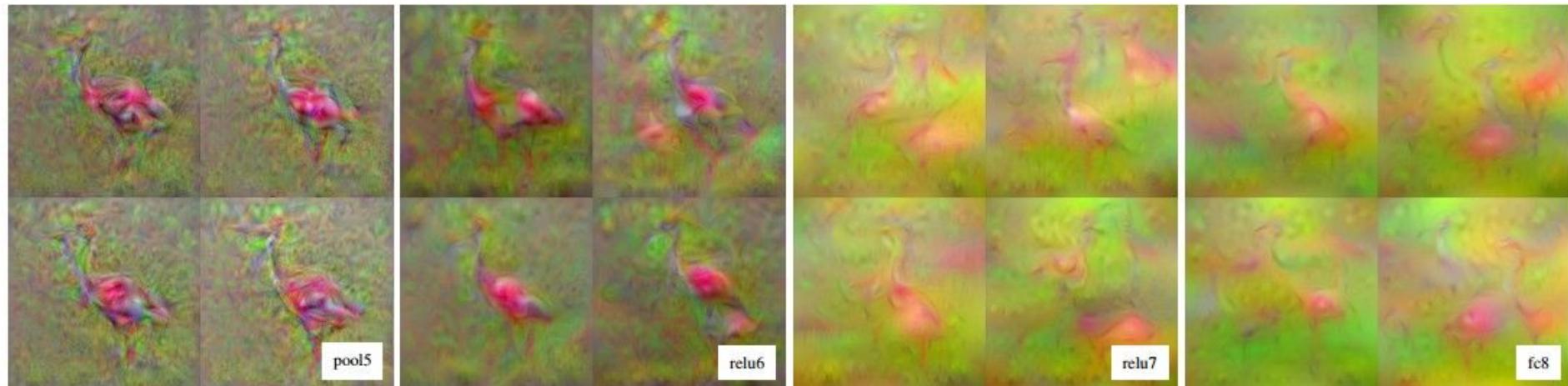


Reconstructions from intermediate layers





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





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

```

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)

```

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

DeepDream: set $dx = x$:)

```
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
```

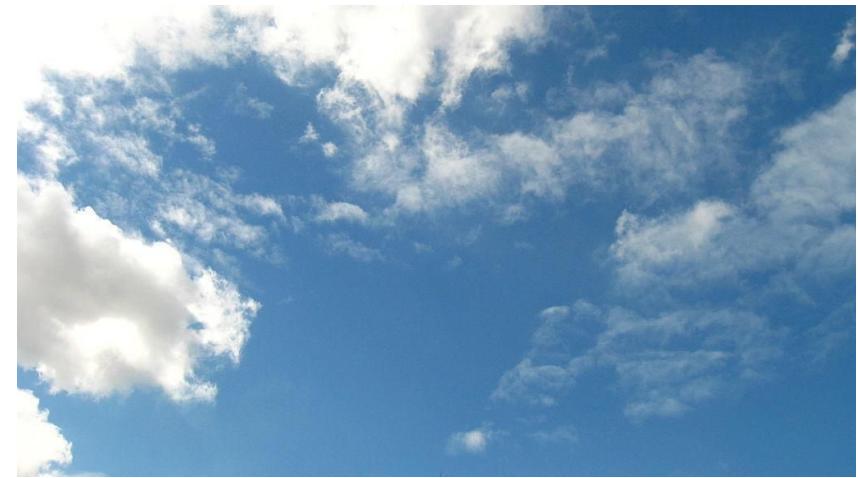
```
src.data[0] = np.roll(np.roll(src.data[0], -ox, -1), -oy, -2) # unshift image
```

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

jitter regularizer

“image update”

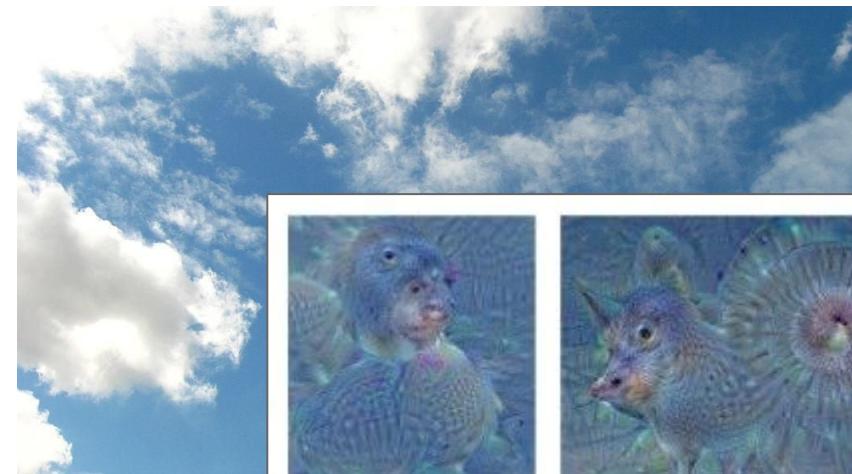
inception_4c/output



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



"Admiral Dog!"



"The Pig-Snail"



"The Camel-Bird"



"The Dog-Fish"



DeepDream modulates 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

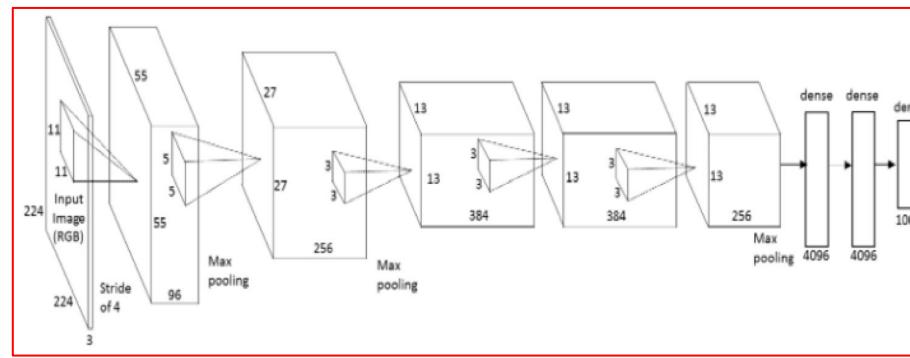
[*A Neural Algorithm of Artistic Style* by Leon A. Gatys,
Alexander S. Ecker, and Matthias Bethge, 2015]
good implementation by Justin in Torch:
<https://github.com/jcjohnson/neural-style>





make your own easily on deepart.io

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

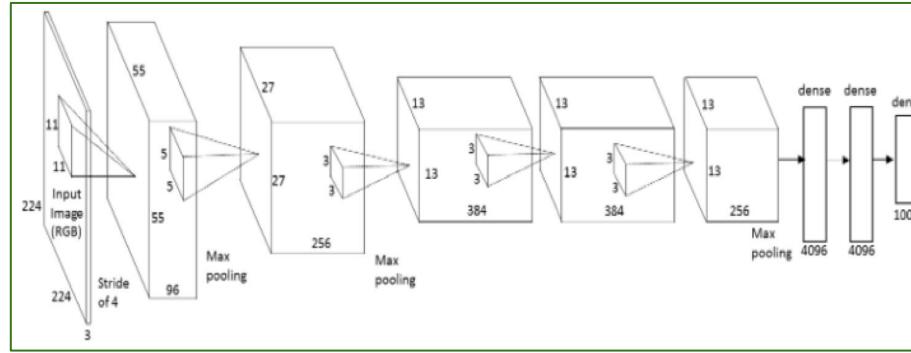


content activations

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

e.g.

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

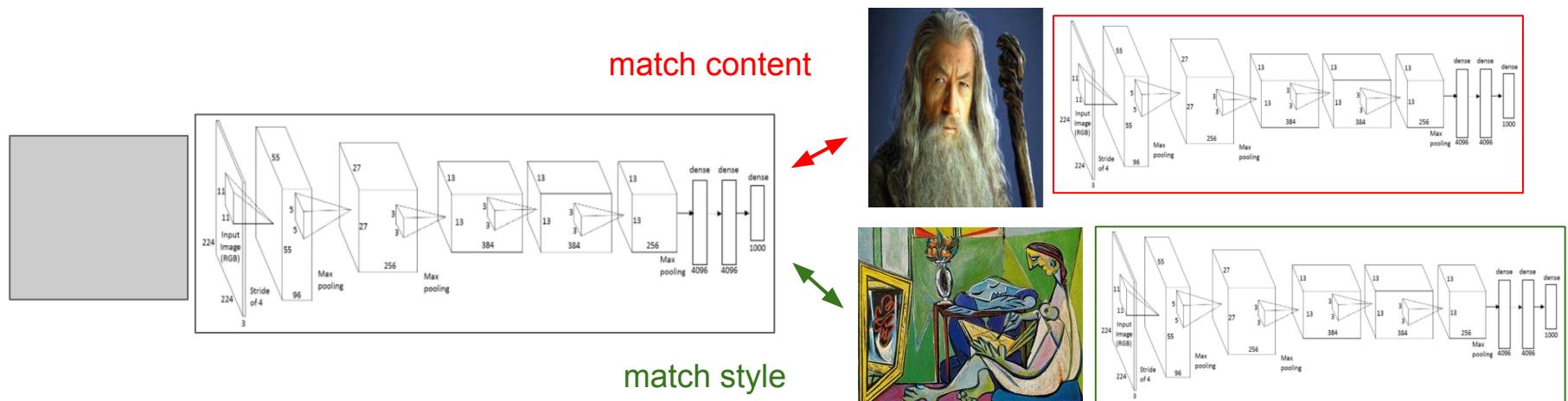
$$G = V^T V$$

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))

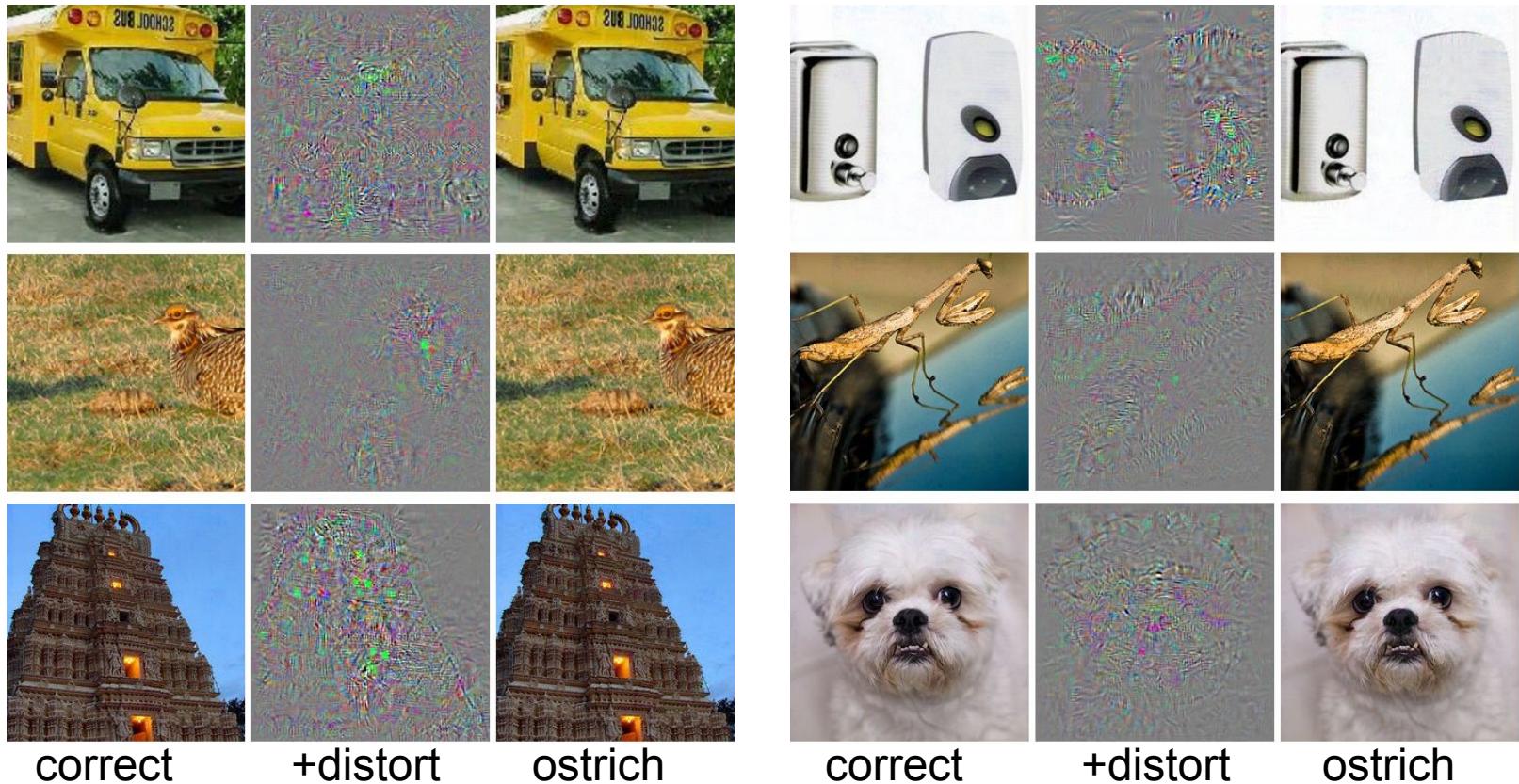


We can pose an optimization over the input image to maximize any class score.
That seems useful.

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

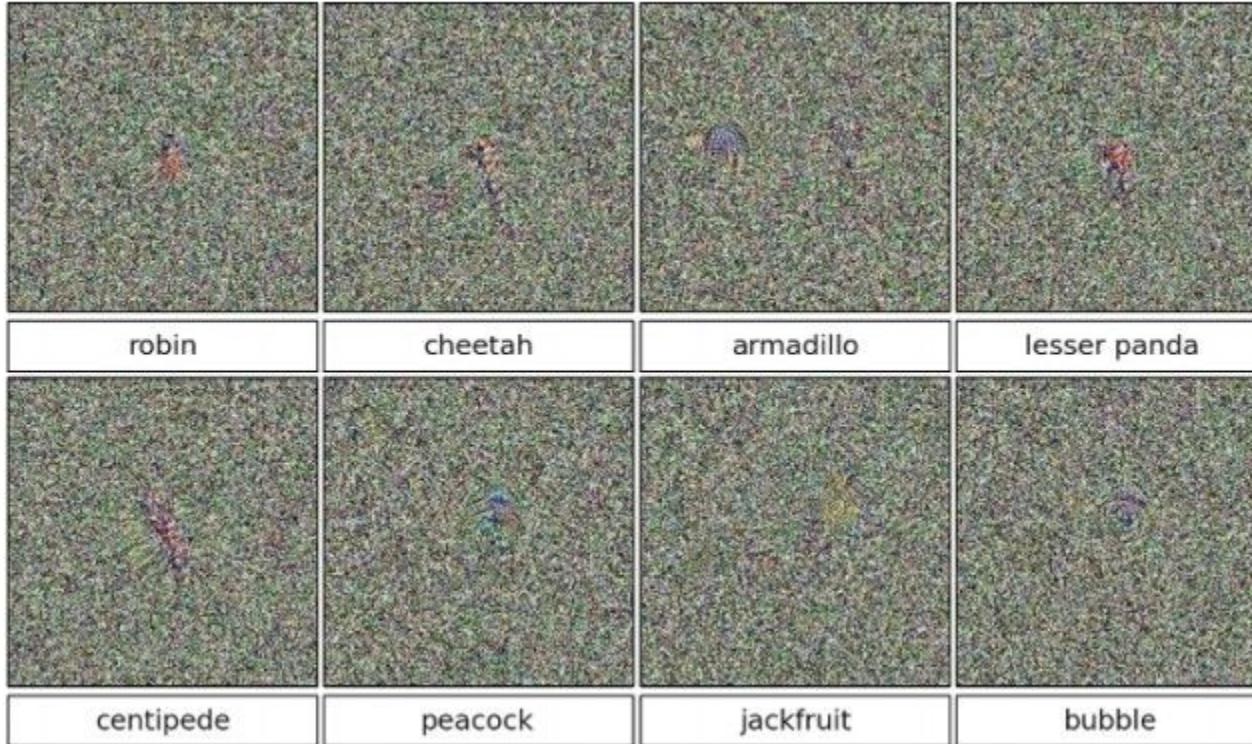
spoiler alert: yeah

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



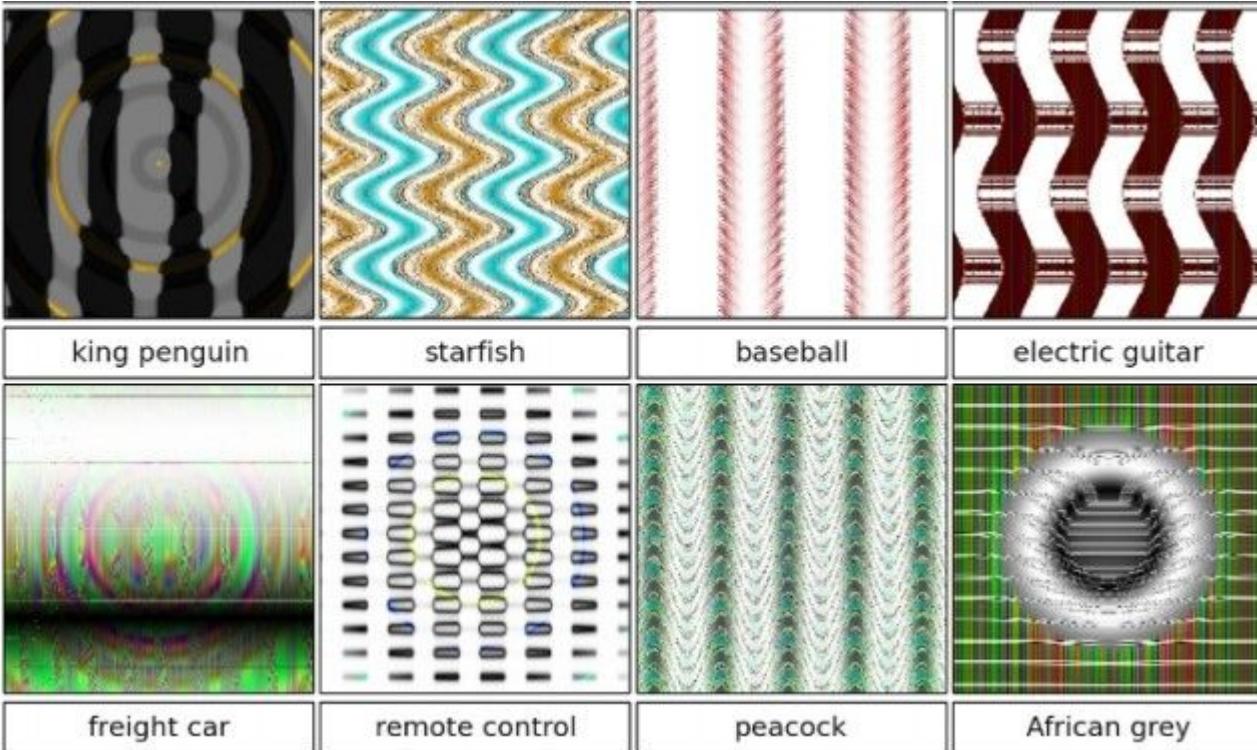
[Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images
Nguyen, Yosinski, Clune, 2014]

>99.6%
confidences

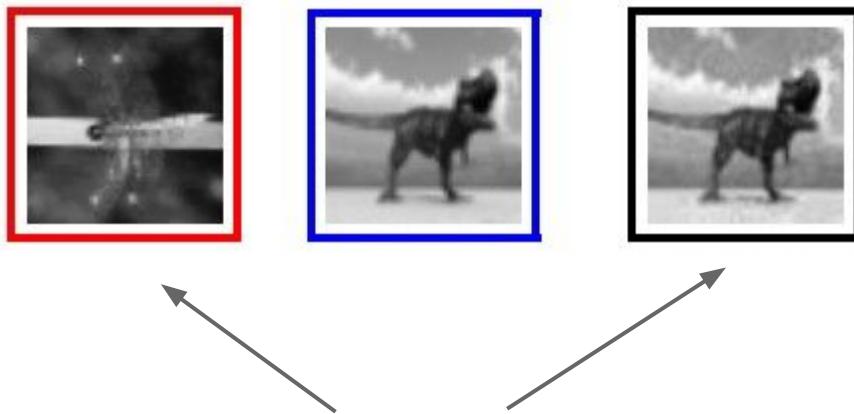


[Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images
Nguyen, Yosinski, Clune, 2014]

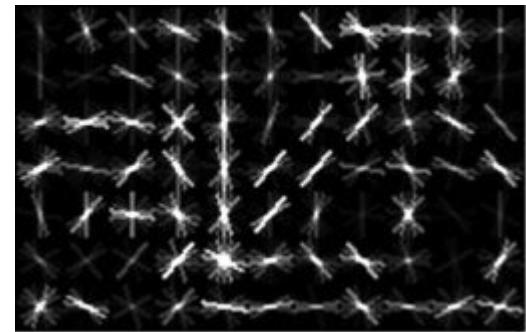
>99.6%
confidences



These kinds of results were around even before ConvNets...
[Exploring the Representation Capabilities of the HOG Descriptor, Tatu et al., 2011]



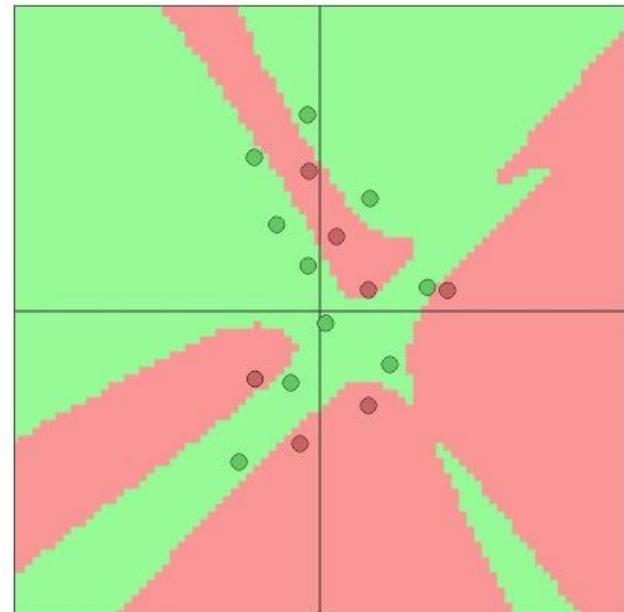
Identical HOG representation



EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES

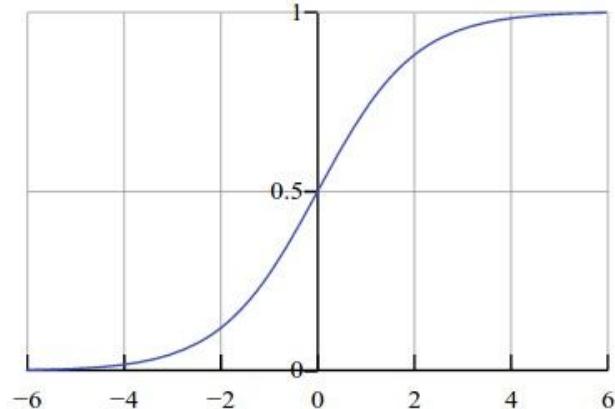
[Goodfellow, Shlens & Szegedy, 2014]

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



Lets fool a binary linear classifier: (logistic regression)

$$P(y = 1 | x; w, b) = \frac{1}{1 + e^{-(w^T x + b)}} = \sigma(w^T x + b)$$



Since the probabilities of class 1 and 0 sum to one, the probability for class 0 is $P(y = 0 | x; w, b) = 1 - P(y = 1 | x; w, b)$. Hence, an example is classified as a positive example ($y = 1$) if $\sigma(w^T x + b) > 0.5$, or equivalently if the score $w^T x + b > 0$.

Lets fool a binary linear classifier:

X	2	-1	3	-2	2	2	1	-4	5	1	← input example
W	-1	-1	1	-1	1	-1	1	1	-1	1	← weights

$$P(y = 1 | x; w, b) = \frac{1}{1 + e^{-(w^T x + b)}} = \sigma(w^T x + b)$$

Lets fool a binary linear classifier:

X	2	-1	3	-2	2	2	1	-4	5	1	← input example
W	-1	-1	1	-1	1	-1	1	1	-1	1	← weights

class 1 score = dot product:

$$= -2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3$$

$$\Rightarrow \text{probability of class 1 is } 1/(1+e^{-(-3)}) = 0.0474$$

i.e. the classifier is **95%** certain that this is class 0 example.

$$P(y = 1 | x; w, b) = \frac{1}{1 + e^{-(w^T x + b)}} = \sigma(w^T x + b)$$

Lets fool a binary linear classifier:

X	2	-1	3	-2	2	2	1	-4	5	1	← input example
W	-1	-1	1	-1	1	-1	1	1	-1	1	← weights
adversarial x	?	?	?	?	?	?	?	?	?	?	

class 1 score = dot product:

$$= -2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3$$

$$\Rightarrow \text{probability of class 1 is } 1/(1+e^{-(-3)}) = 0.0474$$

i.e. the classifier is **95%** certain that this is class 0 example.

$$P(y = 1 | x; w, b) = \frac{1}{1 + e^{-(w^T x + b)}} = \sigma(w^T x + b)$$

Lets fool a binary linear classifier:

X	2	-1	3	-2	2	2	1	-4	5	1	← input example
W	-1	-1	1	-1	1	-1	1	1	-1	1	← weights
adversarial x	1.5	-1.5	3.5	-2.5	2.5	1.5	1.5	-3.5	4.5	1.5	

class 1 score before:

$$-2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3$$

=> probability of class 1 is $1/(1+e^{-(-3)}) = 0.0474$

$-1.5+1.5+3.5+2.5-1.5+1.5-3.5-4.5+1.5 = 2$

=> probability of class 1 is now $1/(1+e^{-(-2)}) = 0.88$

i.e. we improved the class 1 probability from 5% to 88%

$$P(y=1 | x; w, b) = \frac{1}{1 + e^{-(w^T x + b)}} = \sigma(w^T x + b)$$

Lets fool a binary linear classifier:

X	2	-1	3	-2	2	2	1	-4	5	1	← input example
W	-1	-1	1	-1	1	-1	1	1	-1	1	← weights
adversarial x	1.5	-1.5	3.5	-2.5	2.5	1.5	1.5	-3.5	4.5	1.5	

class 1 score before:

$$-2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3$$

=> probability of class 1 is $1/(1+e^{(-(-3))}) = 0.0474$

$$\textcolor{red}{-1.5+1.5+3.5+2.5-1.5+1.5-3.5-4.5+1.5 = 2}$$

=> probability of class 1 is now $1/(1+e^{(-(2)}) = 0.88$

i.e. we improved the class 1 probability from 5% to 88%

This was only with 10 input dimensions. A 224x224 input image has 150,528.

(It's significantly easier with more numbers, need smaller nudge for each)

Blog post: Breaking Linear Classifiers on ImageNet

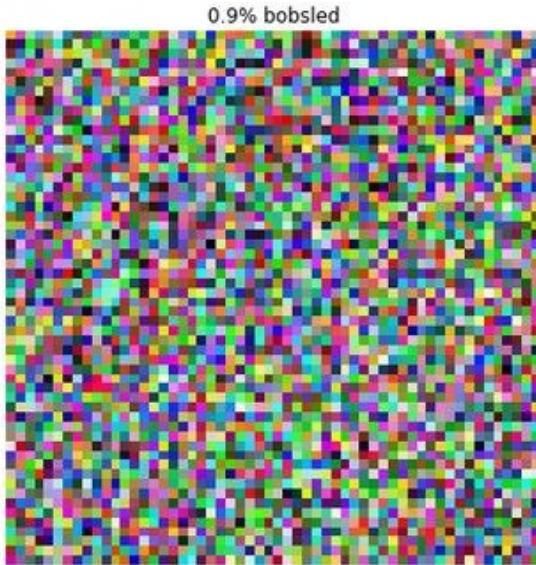
Recall CIFAR-10 linear classifiers:



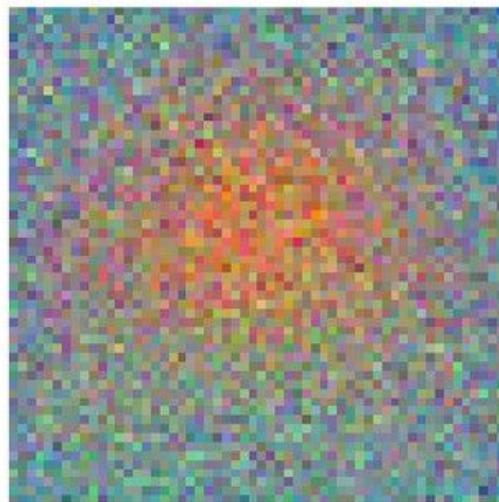
ImageNet classifiers:



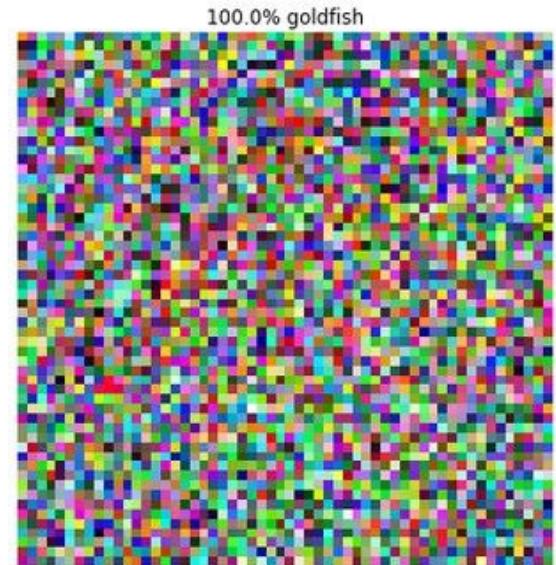
mix in a tiny bit of
Goldfish classifier weights



+



=

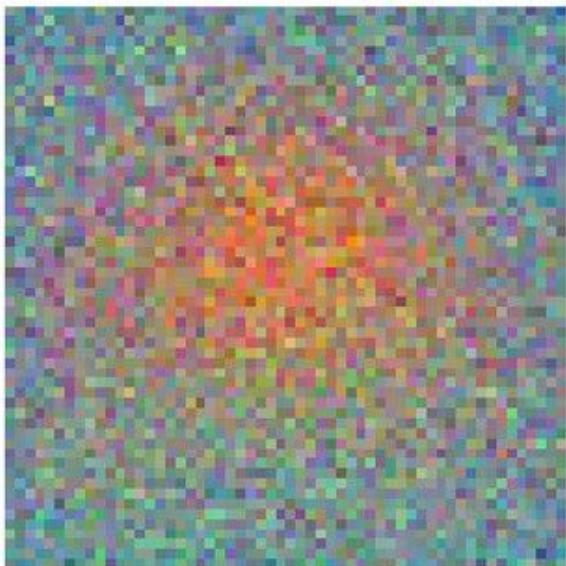


100% Goldfish

1.0% kit fox



8.0% goldfish



1.0% kit fox



3.9% school bus



8.3% goldfish

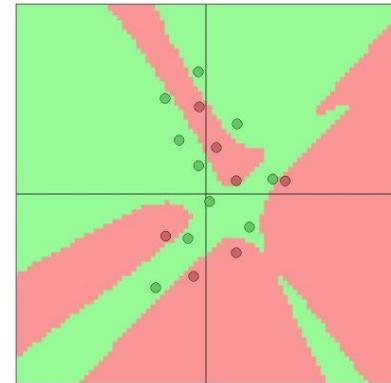


12.5% daisy



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

“primary cause of neural networks’ vulnerability to adversarial perturbation is their **linear nature**“
(and very high-dimensional, sparsely-populated input spaces)



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.

Summary

Backpropping to the image is powerful. It can be used for:

- **Understanding** (e.g. visualize optimal stimuli for arbitrary neurons)
- **Segmenting** objects in the image (kind of)
- **Inverting** codes and introducing privacy concerns
- **Fun** (NeuralStyle/DeepDream)
- **Confusion and chaos** (Adversarial examples)

Next lecture:

Image Captioning

Recurrent Neural Networks

RNN Language Models



"man in black shirt is playing guitar."



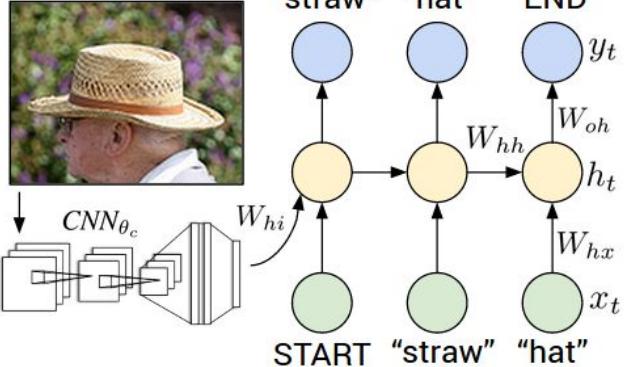
"construction worker in orange safety vest is working on road."

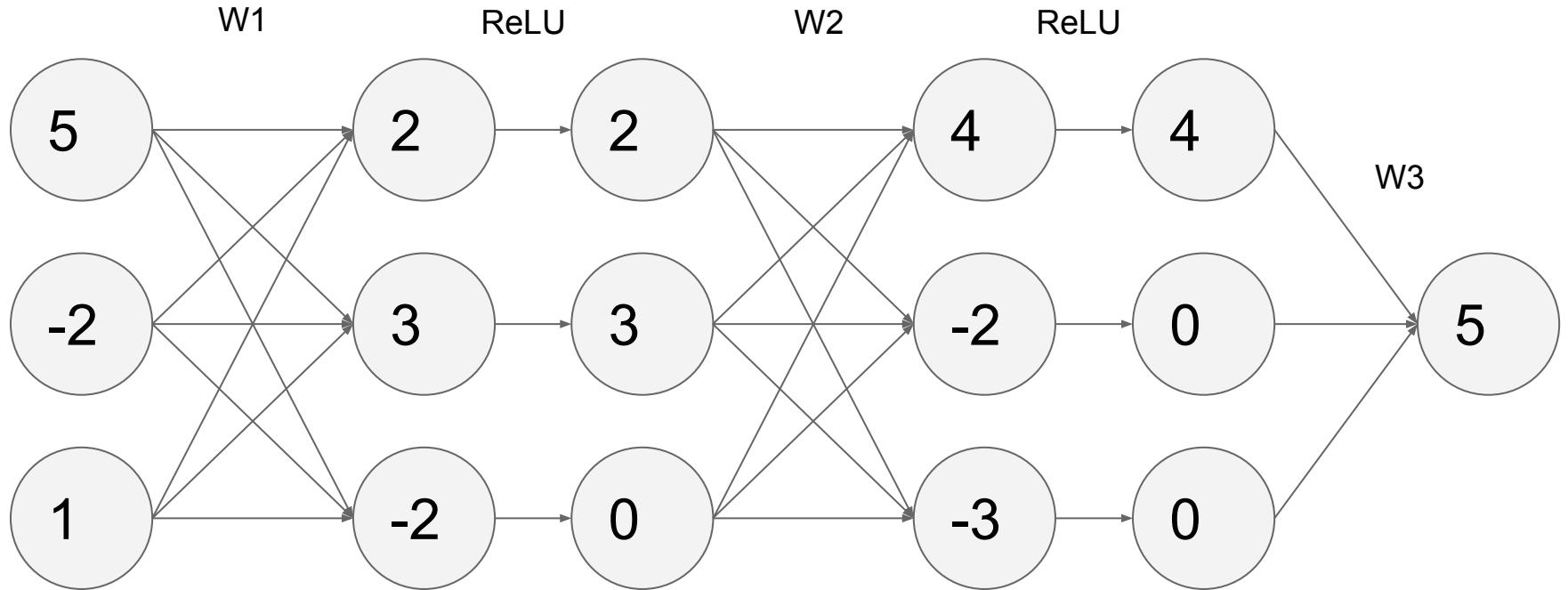


"two young girls are playing with lego toy."

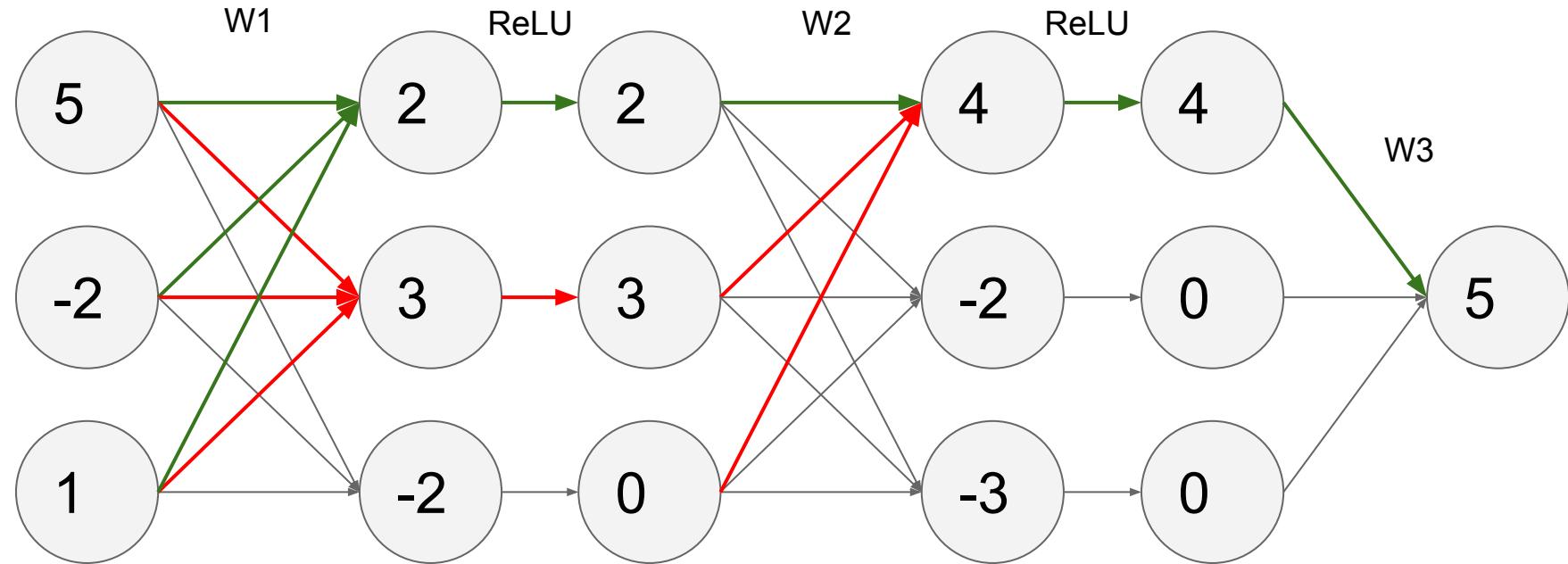


"boy is doing backflip on wakeboard."



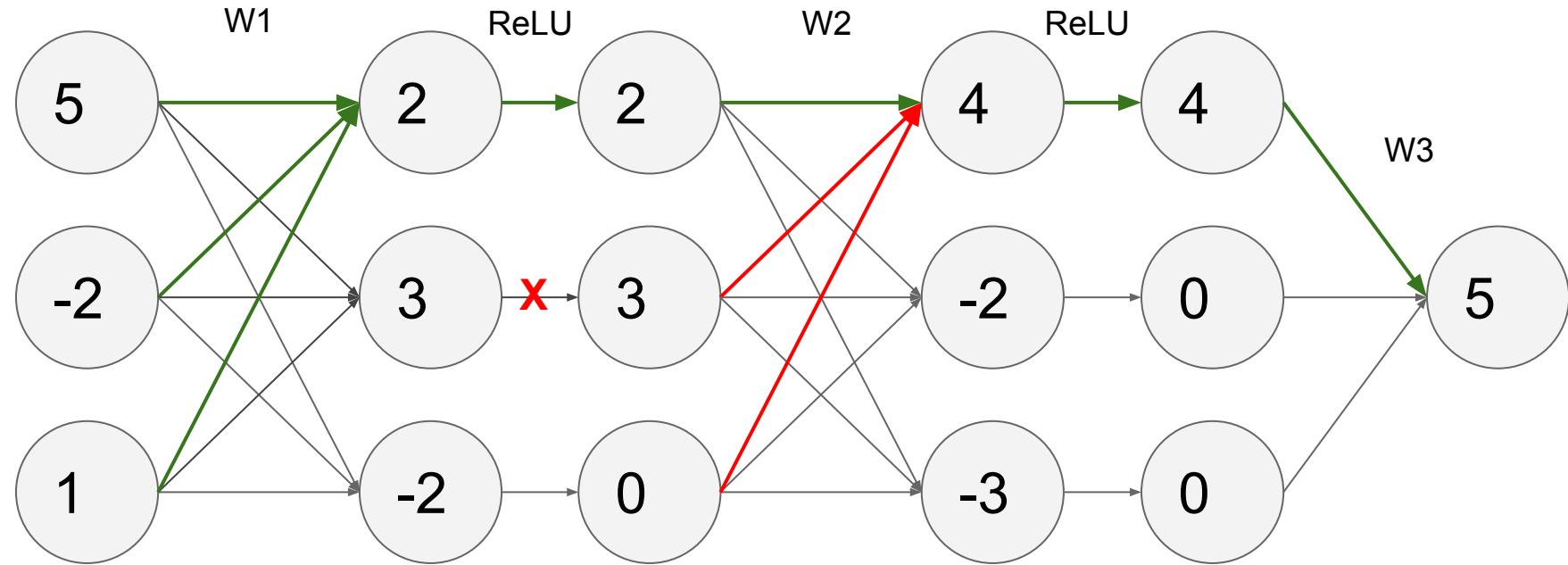


In backprop: all +ve and -ve paths of influence through the graph interfere



positive gradient, negative gradient, zero gradient

In guided backprop: cancel out -ve paths of influence at each step
(i.e. we only keep positive paths of influence)



positive gradient, negative gradient, zero gradient