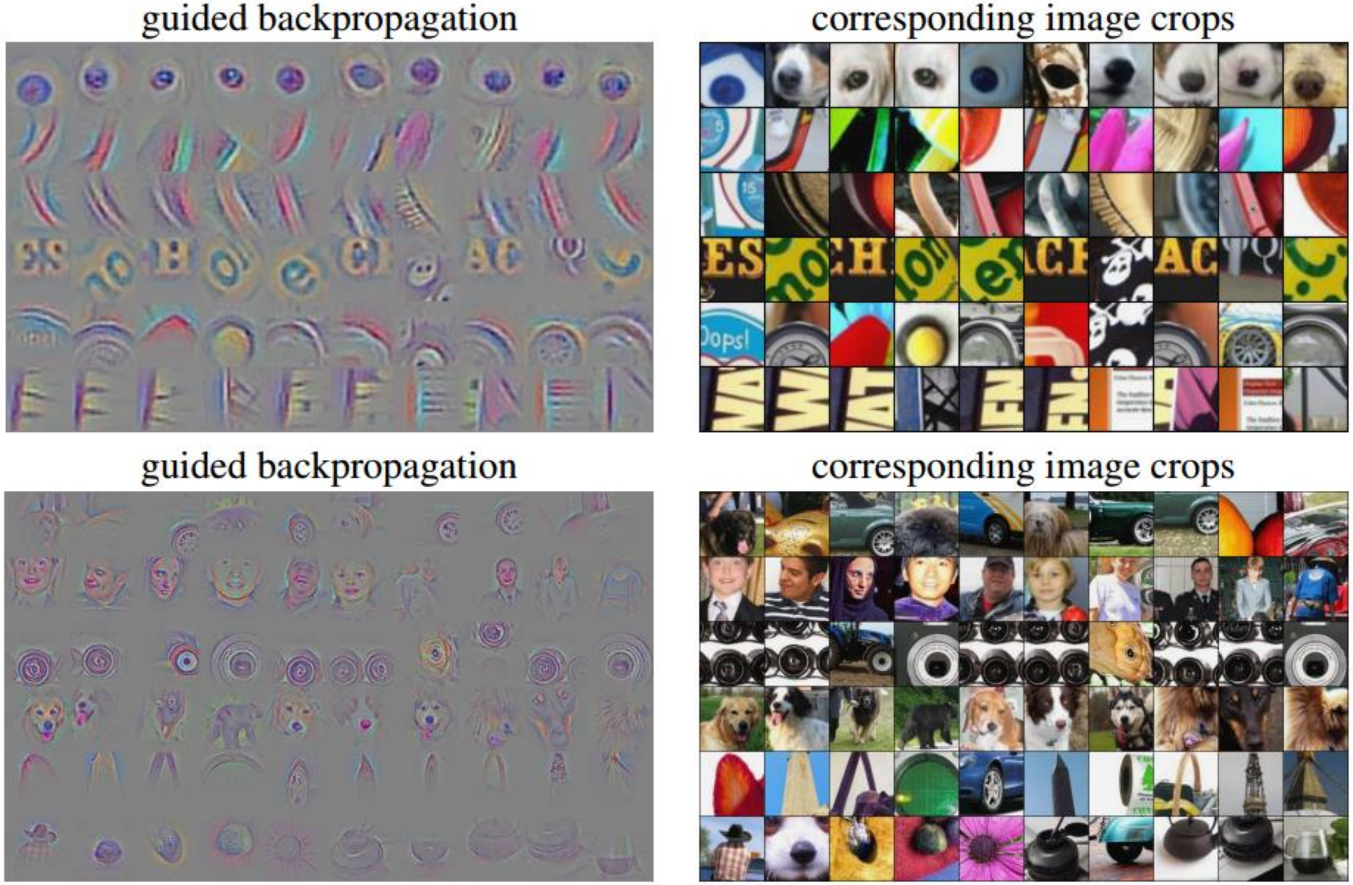
Understanding How ConvNets See



Springerberg et al, Striving for Simplicity: The All Convolutional Net (ICLR 2015 workshops)

Key ideas

- Can understand the mechanism of how the network computes an output by thinking about how individual neurons are computed
- The gradient of the cost with respect to a variable indicates how important the variable is
- Can differentiate a cost with respect to the *input* rather than the weights, and then change the *input*

Prerequisites

- Understand what a gradient does
- Understand how backpropagation works
- Convolutional layers and ReLU

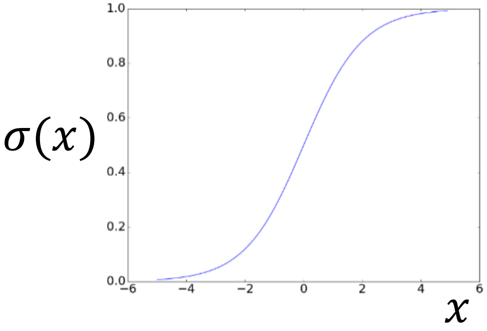
Sample task

- Training set: 6 actors, with 100 64×64 photos of faces for each
- Test set: photos of faces of the same 6 actors
- Want to classify each face as one of ['Fran Drescher', 'America Ferrera', 'Kristin Chenoweth', 'Alec Baldwin', 'Bill Hader', 'Steve Carell']

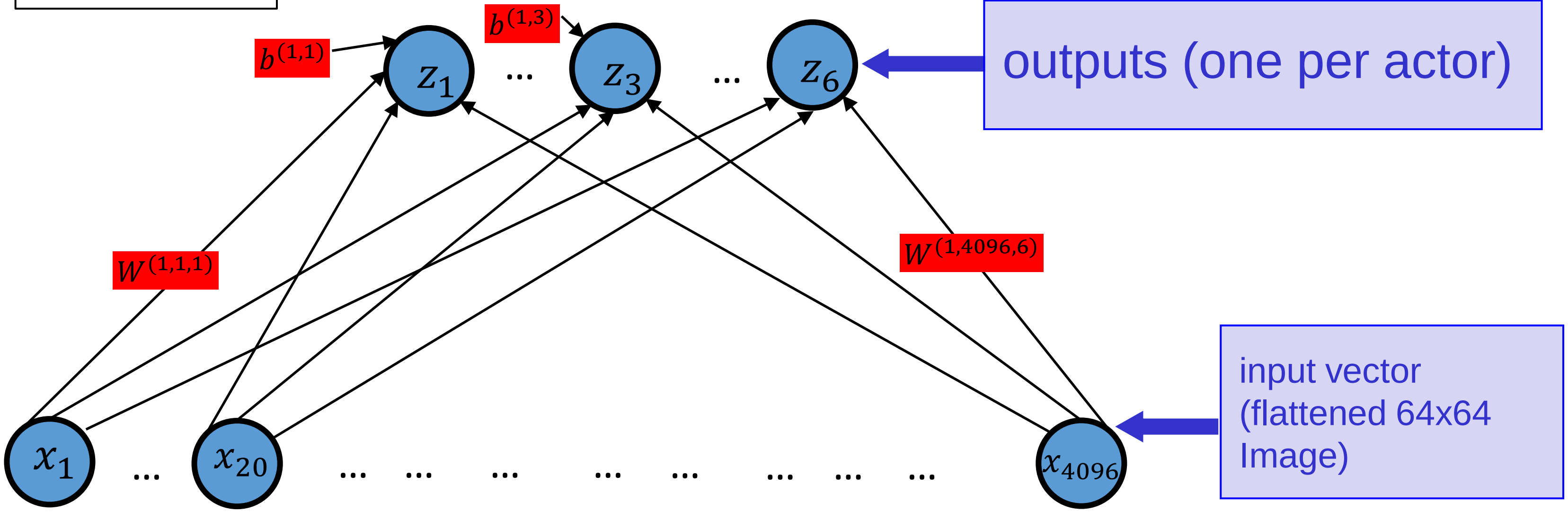
The Simplest Possible Neural Network for Face Recognition

$$z_k = \sigma\left(\sum_{j=1}^{4096} W^{(1,j,k)}\, x_j + b^{(1,k)}\right)$$

$$= \sigma\left(W^{(1,*,k)} \cdot x + b^{(1,k)}\right)$$



$$\boxed{h_\theta = h_{W,b}}$$



outputs (one per actor)

input vector
(flattened 64x64
Image)

An interpretation

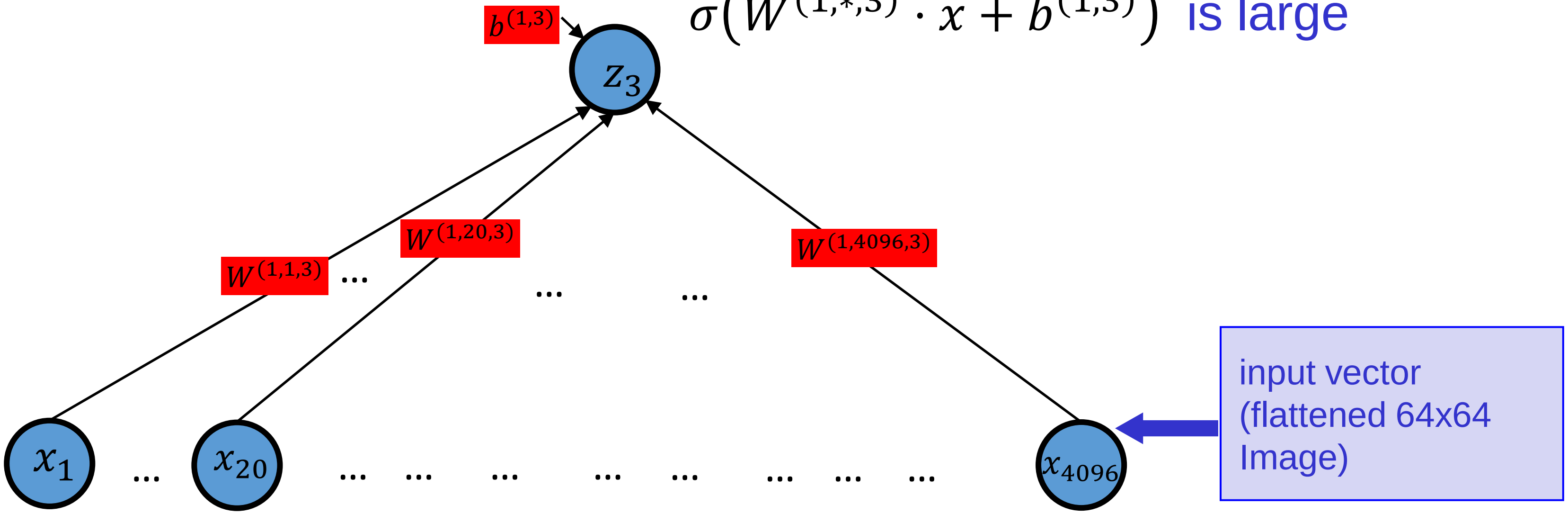
z_1 is large if $W^{(1,*,1)} \cdot x$ is large
z_2 is large if $W^{(1,*,2)} \cdot x$ is large

z_3 is large if $W^{(1,*,3)} \cdot x$ is large

….

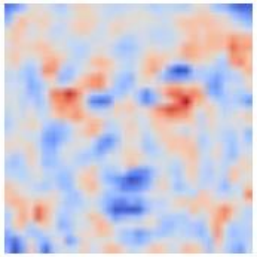
$W^{(1,*,1)}$, $W^{(1,*,2)}$, …, $W^{(1,*,6)}$ are *templates* for the faces of actor 1, actor 2, …, actor 6

Actor 3 neuron activated:
$\sigma\left(W^{(1,*,3)} \cdot x + b^{(1,3)}\right)$ is large

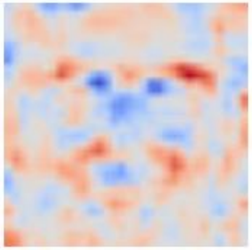


input vector
(flattened 64x64
Image)

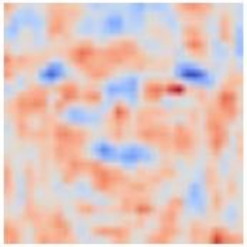
Visualizing the parameters W



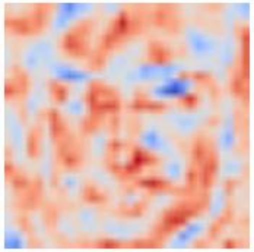
Baldwin
$W^{(1,*,1)}$



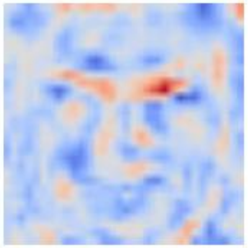
Carrel
$W^{(1,*,2)}$



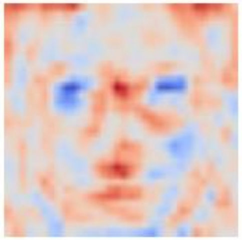
Hader
$W^{(1,*,3)}$



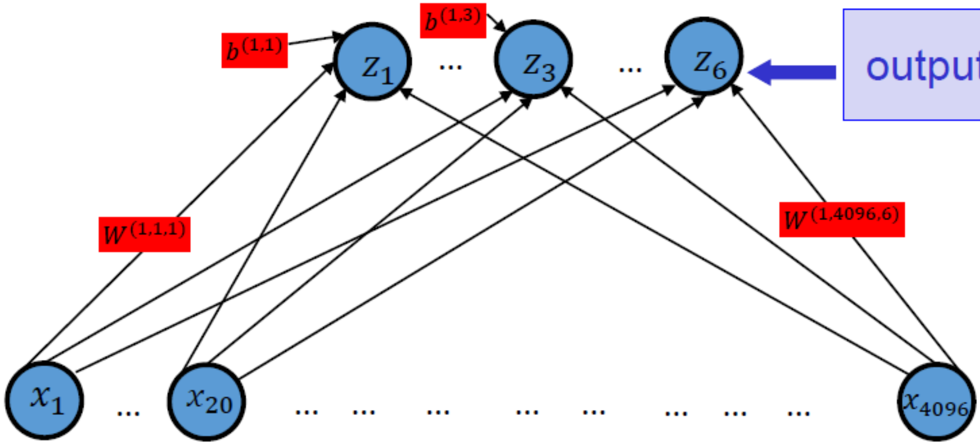
Ferrera
$W^{(1,*,4)}$



Drescher
$W^{(1,*,5)}$



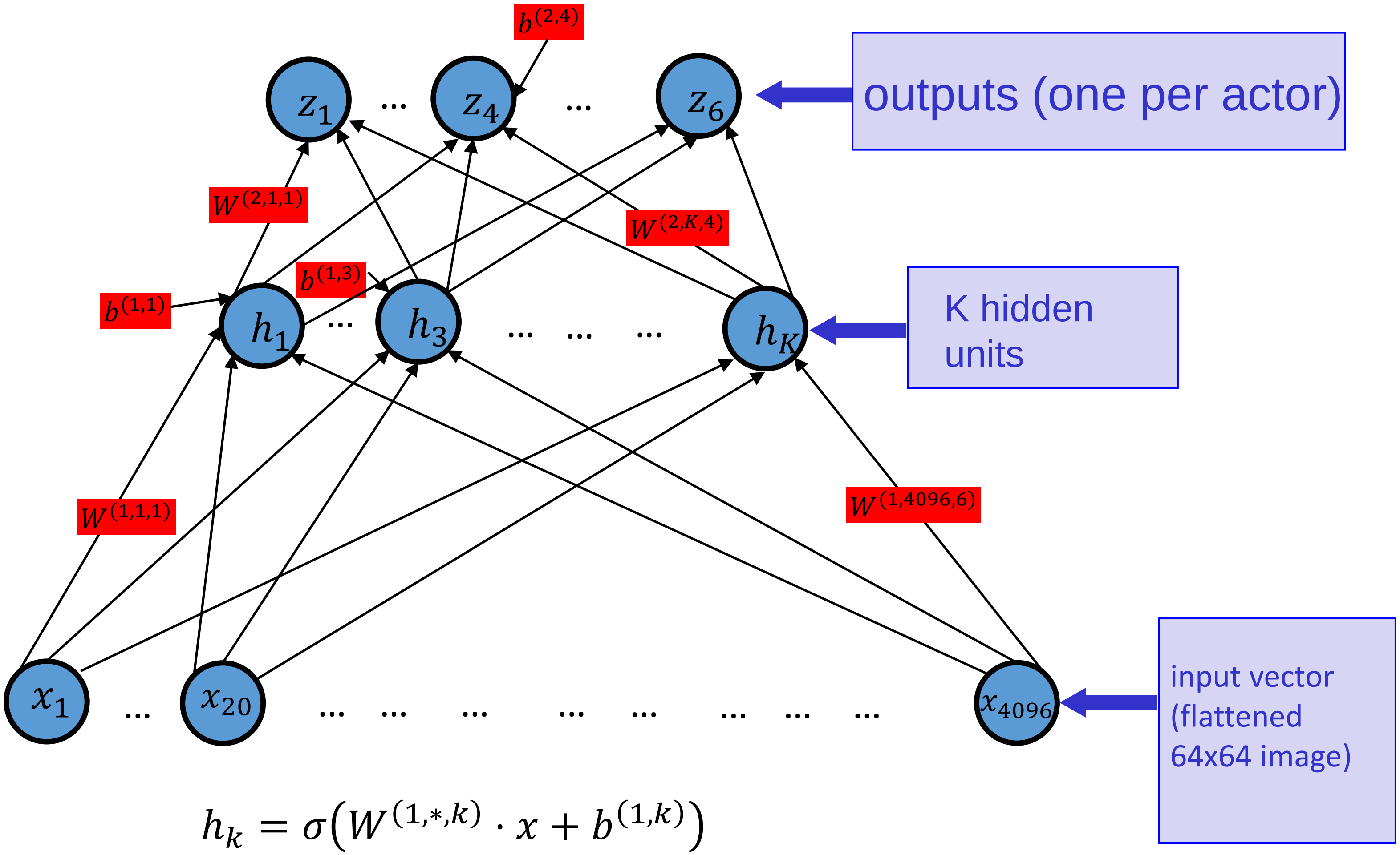
Chenoweth
$W^{(1,*,6)}$



outputs (one per actor)

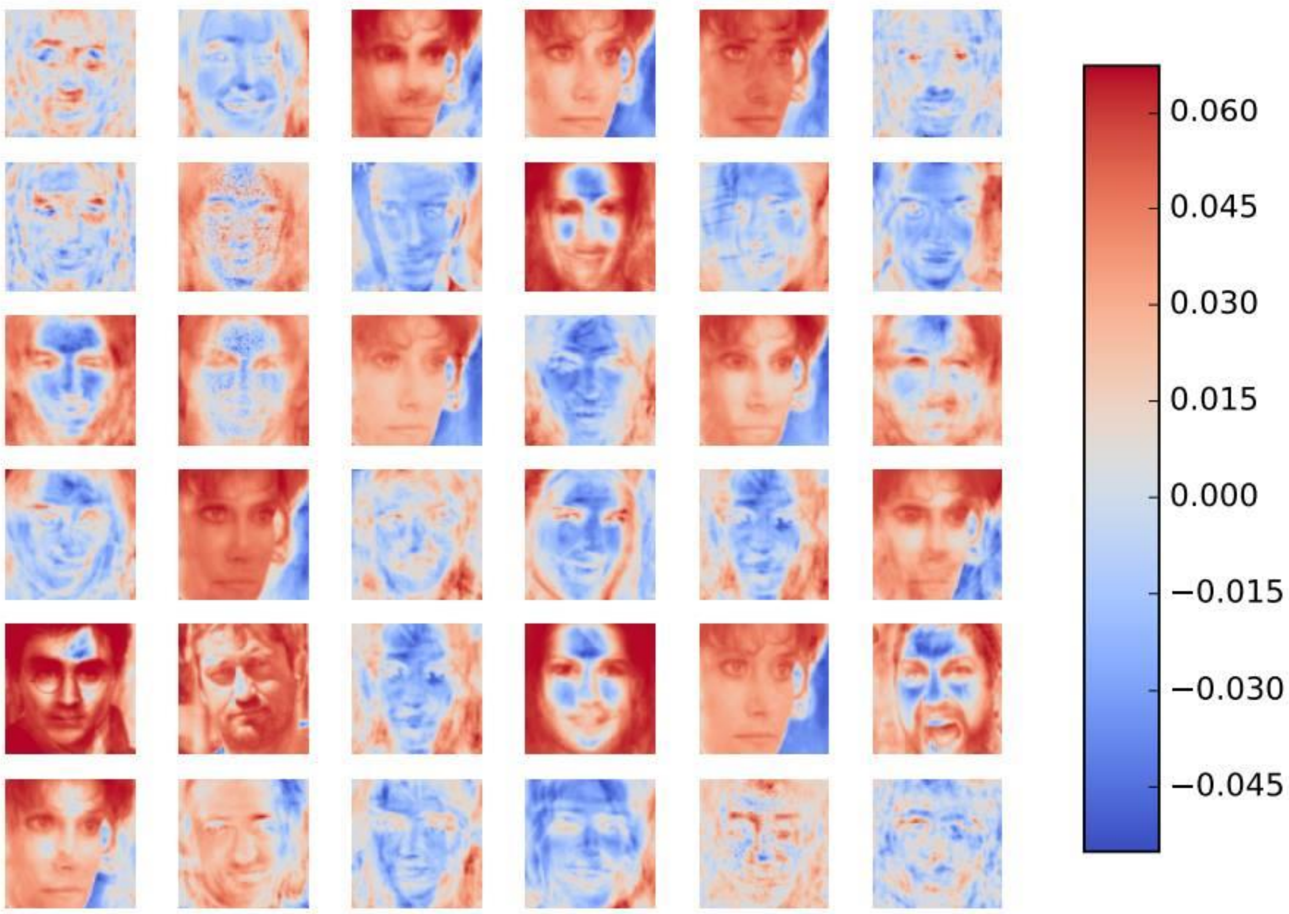
input vector
(flattened 64x64 image)

One hidden layer



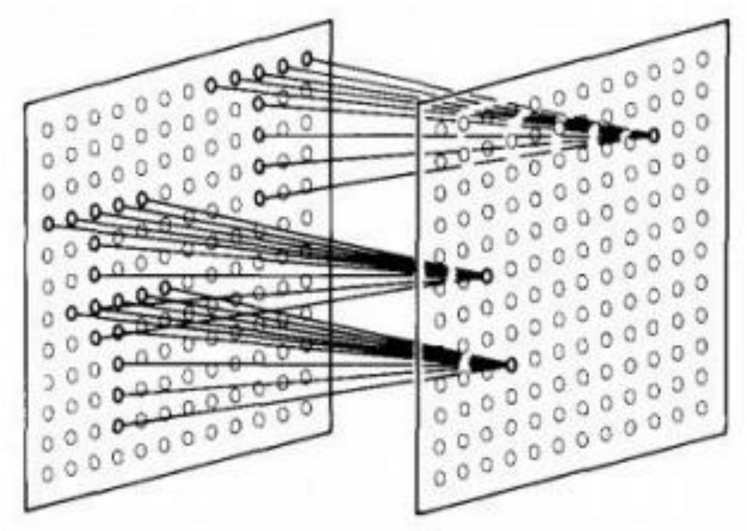
$$h_k = \sigma\left(W^{(1,*,k)} \cdot x + b^{(1,k)}\right)$$
$$z_m = \sigma(W^{(2,*,m)} \cdot h + b^{(2,m)})$$

Visualizing a One-Hidden-Layer NN



What Does a Neuron Do in a ConvNet? (1)

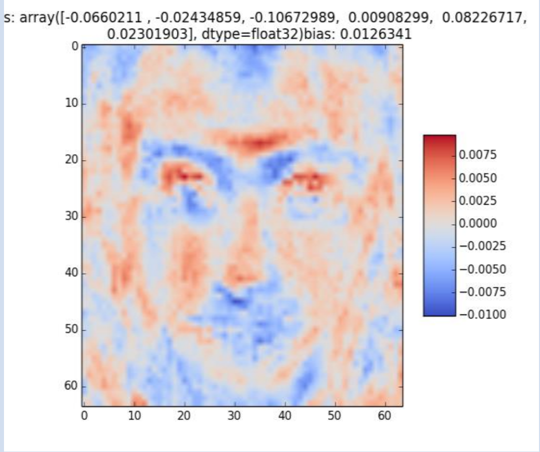
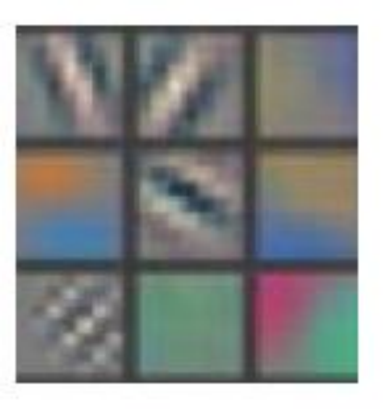
- A neuron in the first hidden layer computes a weighted sum of pixels in a patch of the image for which it is responsible



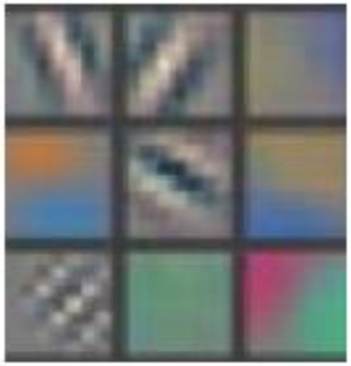
K. Fukushima, “Neurocognitron: A self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position” (Biol. Cybernetics 1980)

What Does a Neuron Do in a ConvNet? (2)

- For neurons in the first hidden layer, we can visualize the weights.

Example weights for fully-connected single-hidden layer network for faces, for one neuron	**Weights for 9 features in the first convolutional layer of a layer for classifying ImageNet images**
 	 Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”

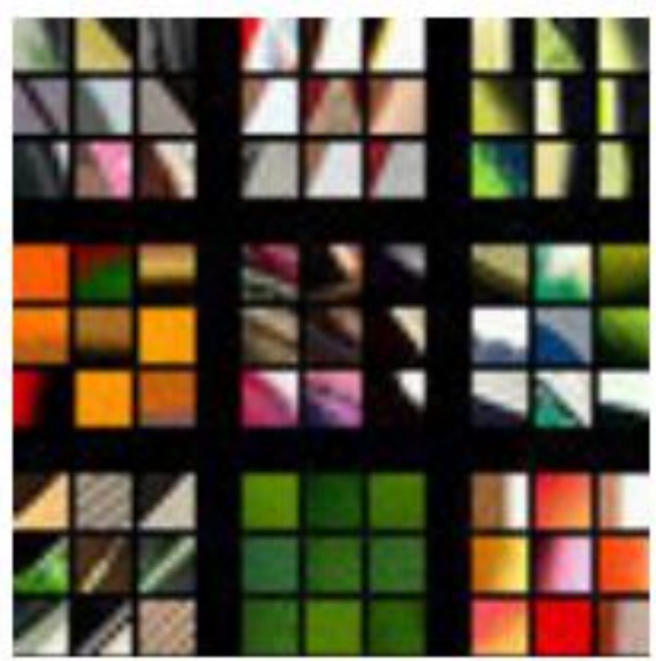
What Does a Neuron Do in a ConvNet? (3)



- **The neuron would be activated the most if the input looks like the weight matrix**
- These are called “Gabor-like filters”
- The colour is due to the input being 3D. We visualize the strength of the weight going from each of the R, G, and B components

What Does a Neuron Do in a ConvNet (4)

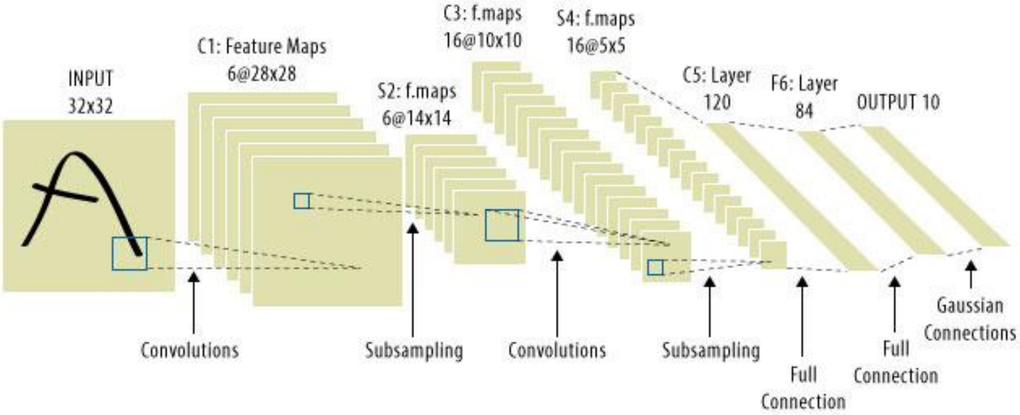
- Another to figuring out what kind of images active the neuron: just try lots of images in a dataset, and see which ones activate the neuron the most



For each feature, fine the 9 images that produce the highest activations for the neuron, and crop out the relevant patch

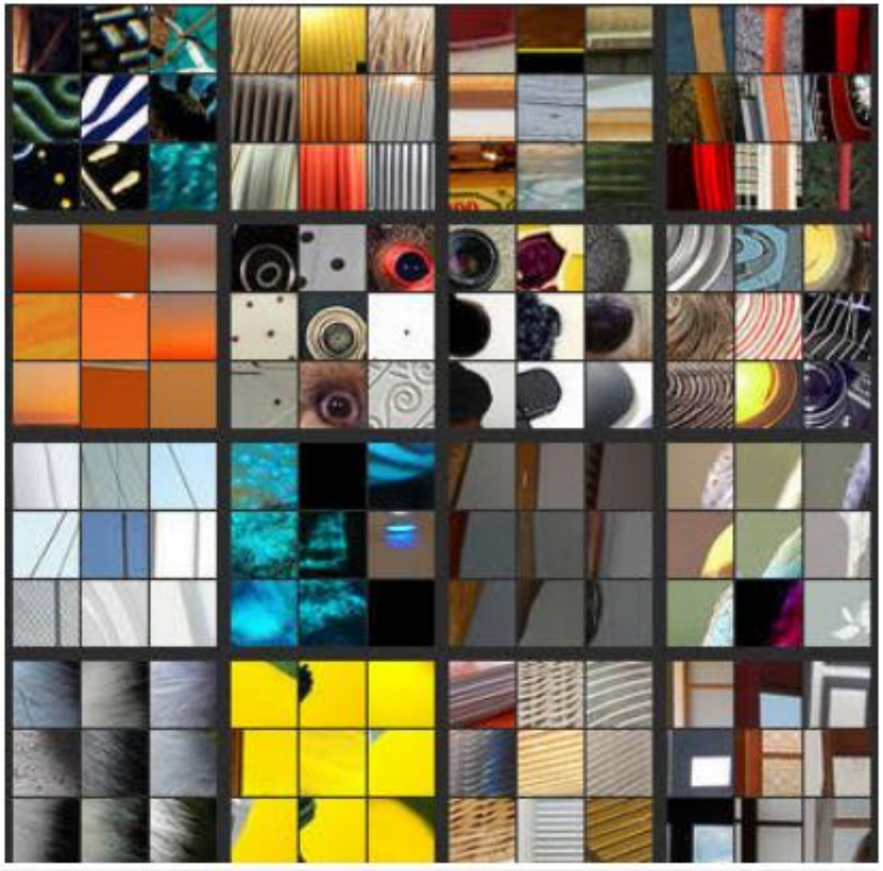
Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks"

Aside: Relevant Patch?

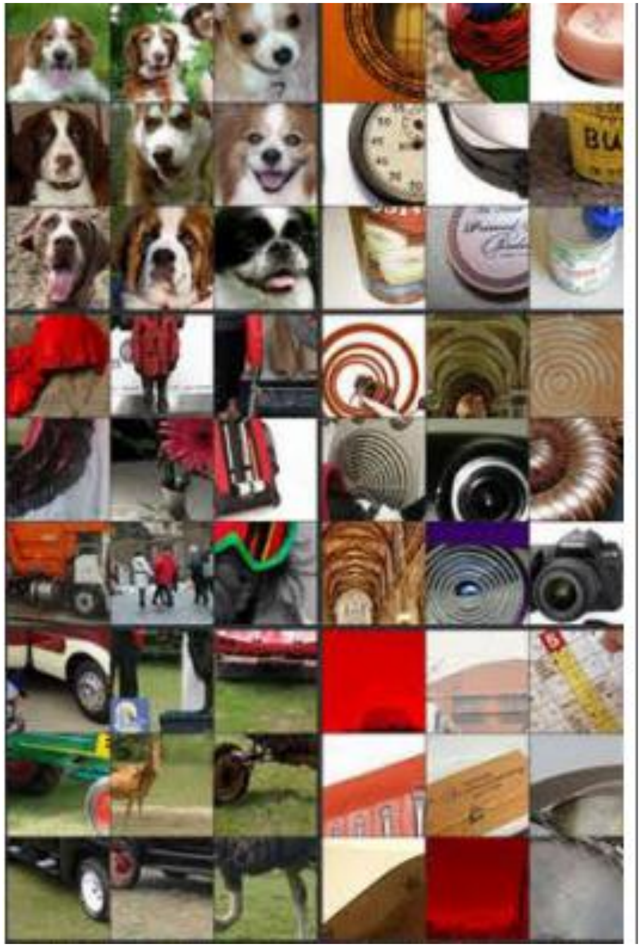


- Each neuron is affected by some small patch in the layer below
- Can recursively figure out what patch in the input layer each neuron is affected
- Neurons in the top layers are affected by (almost) the entire image

This allows us to look at layers besides the first one: layer 3



Layer 4



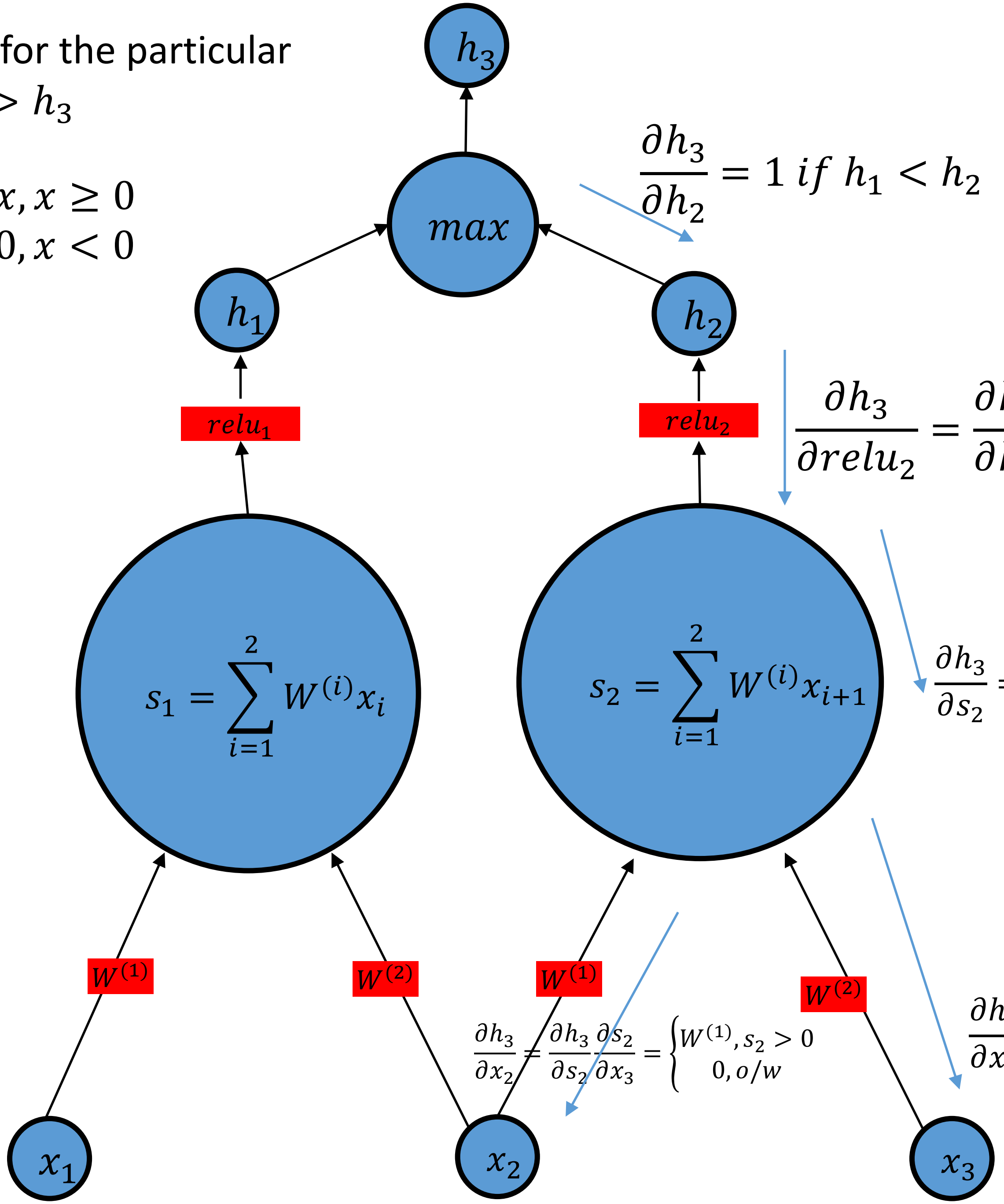
Layer 5

Which Pixels in the Input Affect the Neuron the Most?

- Rephrased: which pixels would make the neuron not turn on if they had been different?
- In other words, for which inputs is $\frac{\partial neuron}{\partial x_i}$ large?

Assume that for the particular image x, $h_2 > h_3$

$$relu(x) = \begin{cases} x, x \geq 0 \\ 0, x < 0 \end{cases}$$



$$\frac{\partial h_3}{\partial h_2} = 1 \; if \; h_1 < h_2$$

$$\frac{\partial h_3}{\partial relu_2} = \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial relu_2} = \begin{cases} 1, s_2 > 0 \\ 0, o/w \end{cases}$$

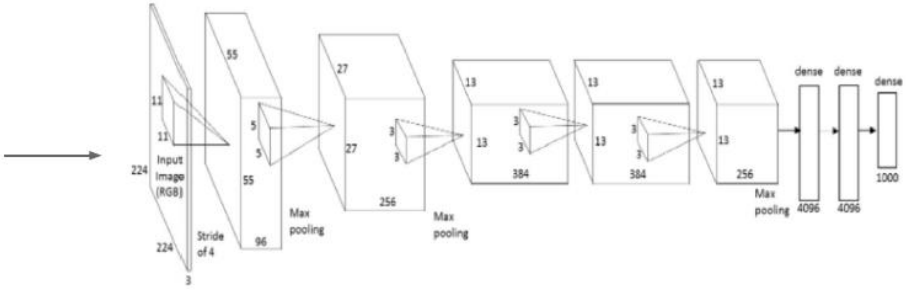
$$\frac{\partial h_3}{\partial s_2} = \frac{\partial h_3}{\partial relu_2} \frac{\partial relu_2}{\partial s_2} = \begin{cases} 1, s_2 > 0 \\ 0, o/w \end{cases}$$

$$\frac{\partial h_3}{\partial x_2} = \frac{\partial h_3}{\partial s_2} \frac{\partial s_2}{\partial x_3} = \begin{cases} W^{(1)}, s_2 > 0 \\ 0, o/w \end{cases}$$

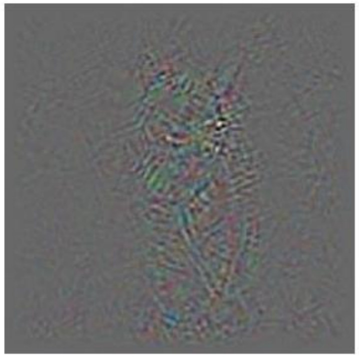
$$\frac{\partial h_3}{\partial x_3} = \frac{\partial h_3}{\partial s_2} \frac{\partial s_2}{\partial x_3} = \begin{cases} W^{(2)}, s_2 > 0 \\ 0, o/w \end{cases}$$

Typical Gradient of a Neuron

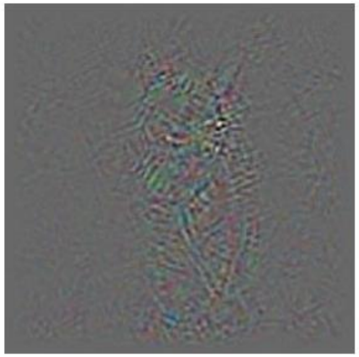
- Visualize the gradient of a particular neuron with respect to the input *x*
- Do a forward pass:



- Compute the gradient of a particular neuron using backprop:

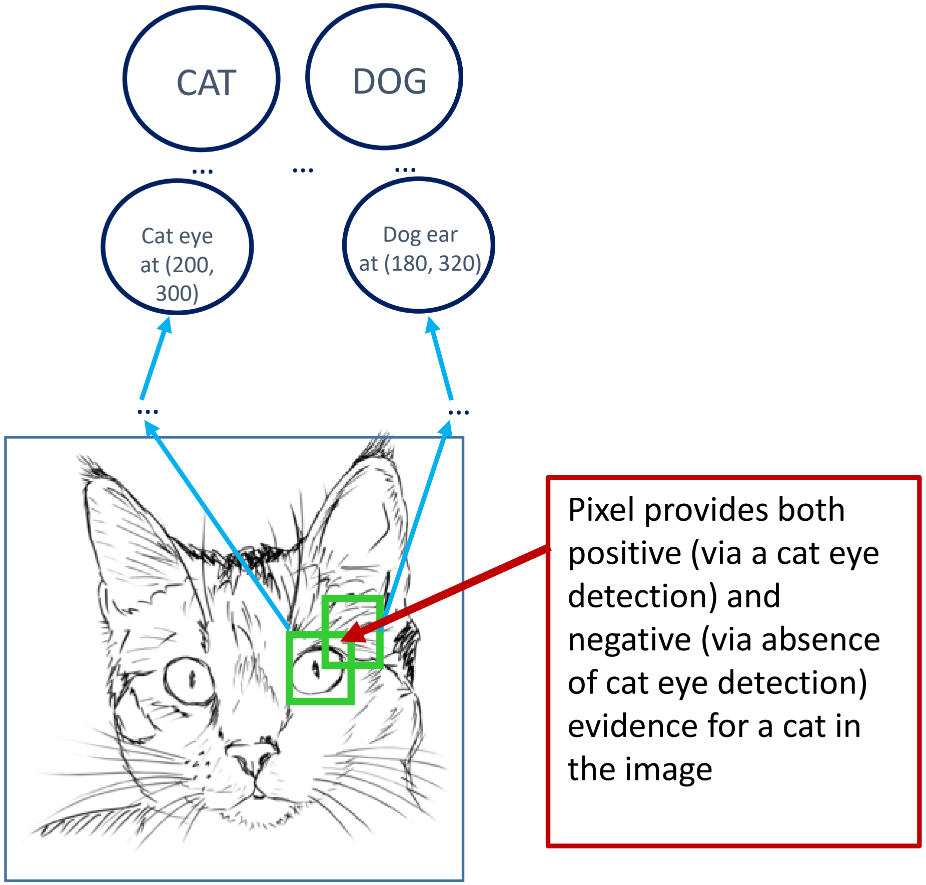


Typical Gradient of a Neuron



- Mostly zero away from the object, but the results are not very satisfying
- Every pixel influences the neuron via multiple hidden neurons.
  The network is trying to detect kittens everywhere, and the same pixel could fit a kitten in one location but not another, leading to its overall effect on the kitten neuron to be 0

Typical Gradient of a Neuron



Pixel provides both positive (via a cat eye detection) and negative (via absence of cat eye detection) evidence for a cat in the image

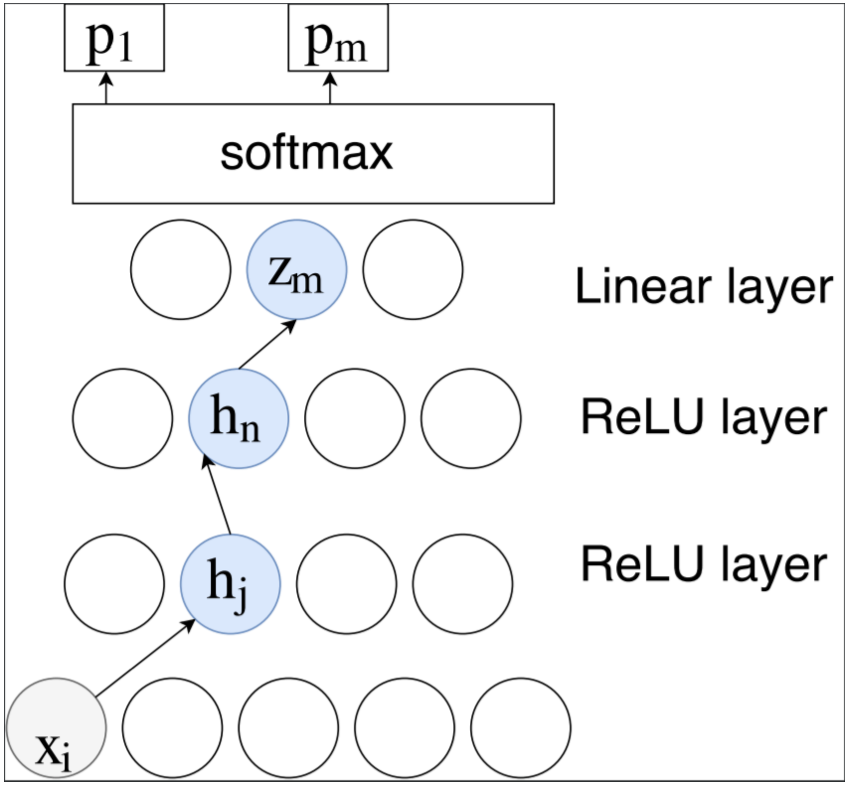
Guided Backpropagation

- Idea: neurons act like detectors of particular image features
- We are only interested in what image features the neuron detects, not in what kind of stuff it *doesn't* detect

Guided Backpropagation

- Instead of computing $\frac{\partial p_m}{\partial x}$, only consider paths from x to p_m where the weights are positive and all the units are positive (and greater than 0). Compute this modified version of $\frac{\partial p_m}{\partial x}$

- Only consider evidence for neurons being active, discard evidence for neurons having to be not active



Guided Backpropagation: Computation

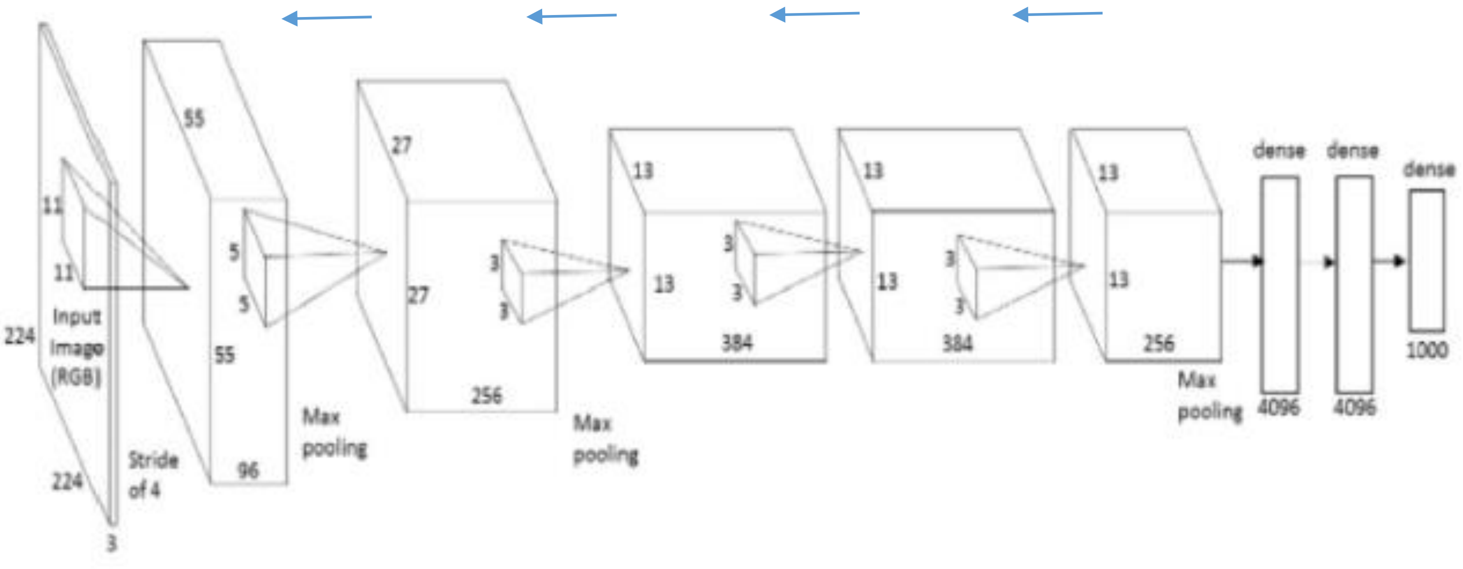
- When performing the backward pass we already know "$\frac{\partial neuron}{\partial h^{(l,i)}}$" for every i
- If $\frac{\partial h^{(l,i)}}{\partial h^{(l,-1\ j)}}$ < 0, set it to 0
- Compute "$\frac{\partial neuron}{\partial h^{(l-1,j)}}$" = $= \sum_i$ "$\frac{\partial neuron}{\partial h^{(l,i)}}$""$\frac{\partial h^{(l,i)}}{\partial h^{(l,-1\ j)}}$"
- Repeat
- If a path contains negative weights, it will be ignored, since a negative weight corresponds to a negative $\frac{\partial h^{(l,i)}}{\partial h^{(l,-1\ j)}}$

Guided Backpropagation

Compute gradient,
zero out negatives,
backpropagate

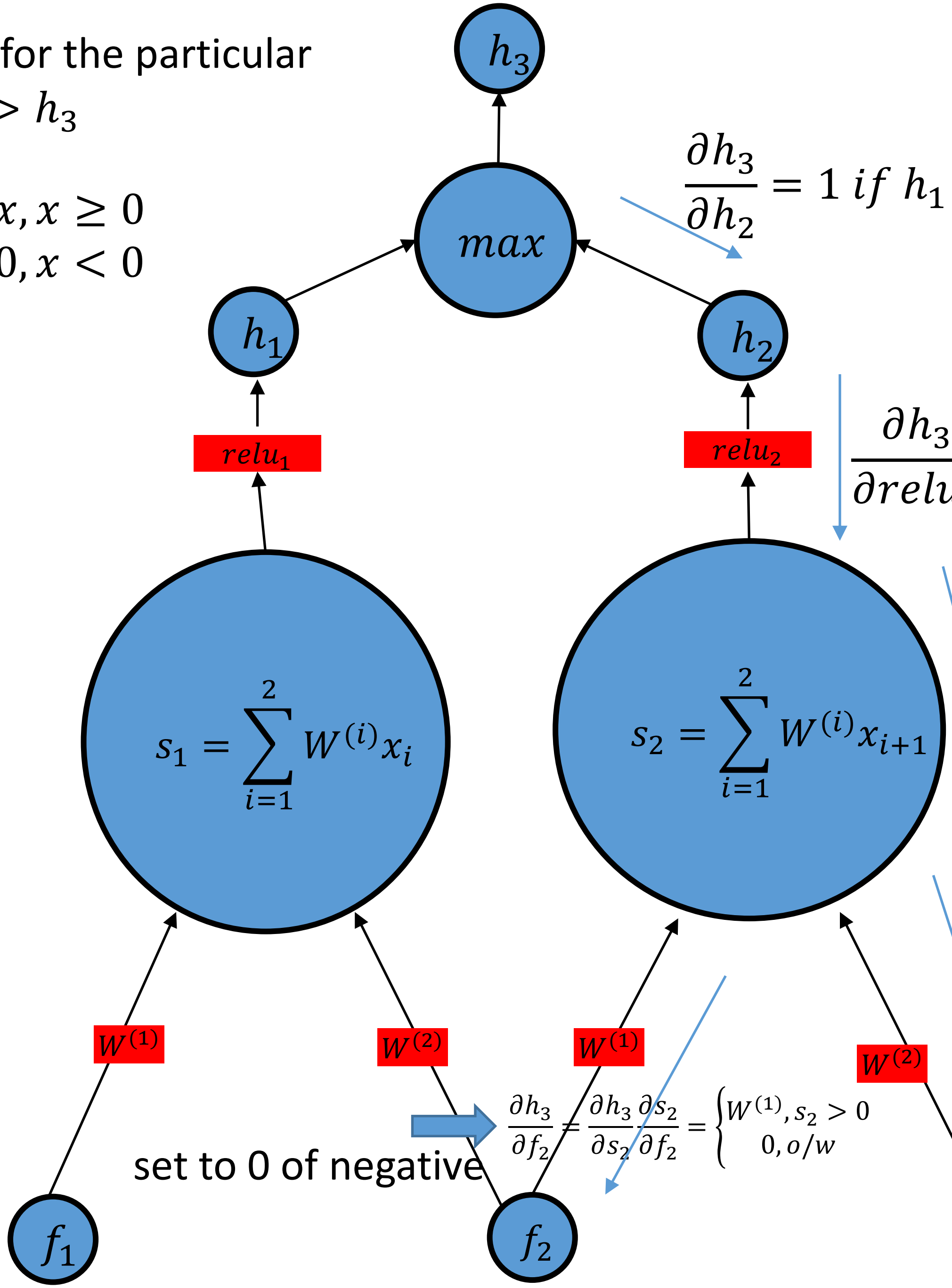
Compute gradient,
zero out negatives,
backpropagate

Compute gradient,
zero out negatives,
backpropagate



Assume that for the particular image x, $h_2 > h_3$

$$relu(x) = \begin{cases} x, x \geq 0 \\ 0, x < 0 \end{cases}$$



$$\frac{\partial h_3}{\partial h_2} = 1\ if\ h_1 < h_2$$

$$\frac{\partial h_3}{\partial relu_2} = \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial relu_2} = \begin{cases} 1, s_2 > 0 \\ 0, o/w \end{cases}$$

$$\frac{\partial h_3}{\partial s_2} = \frac{\partial h_3}{\partial relu_2} \frac{\partial relu_2}{\partial s_2} = \begin{cases} 1, s_2 > 0 \\ 0, o/w \end{cases}$$

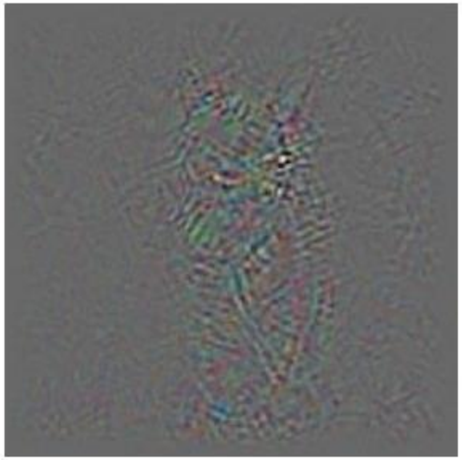
set to 0 if negative

$$\frac{\partial h_3}{\partial f_3} = \frac{\partial h_3}{\partial s_2} \frac{\partial s_2}{\partial f_3} = \begin{cases} W^{(2)}, s_2 > 0 \\ 0, o/w \end{cases}$$

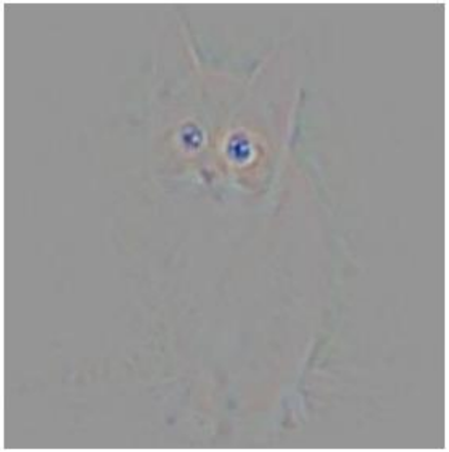
set to 0 of negative

$$\frac{\partial h_3}{\partial f_2} = \frac{\partial h_3}{\partial s_2} \frac{\partial s_2}{\partial f_2} = \begin{cases} W^{(1)}, s_2 > 0 \\ 0, o/w \end{cases}$$

Guided Backpropagation

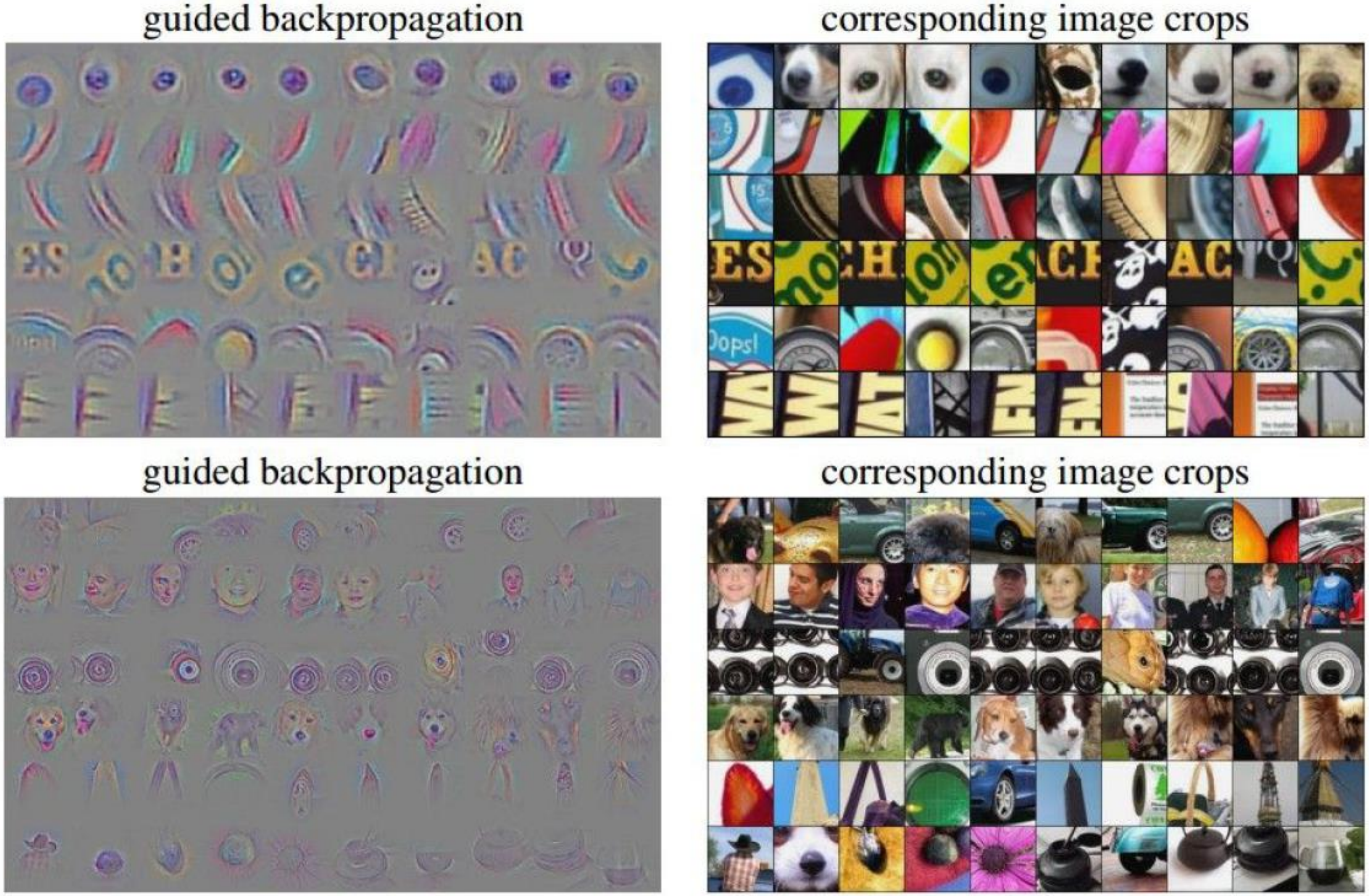


Backprop



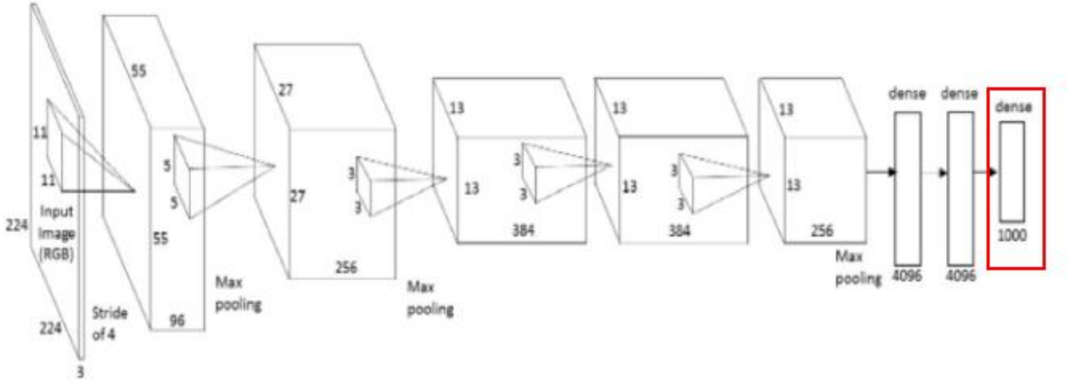
Guided Backprop

Guided Backpropagation

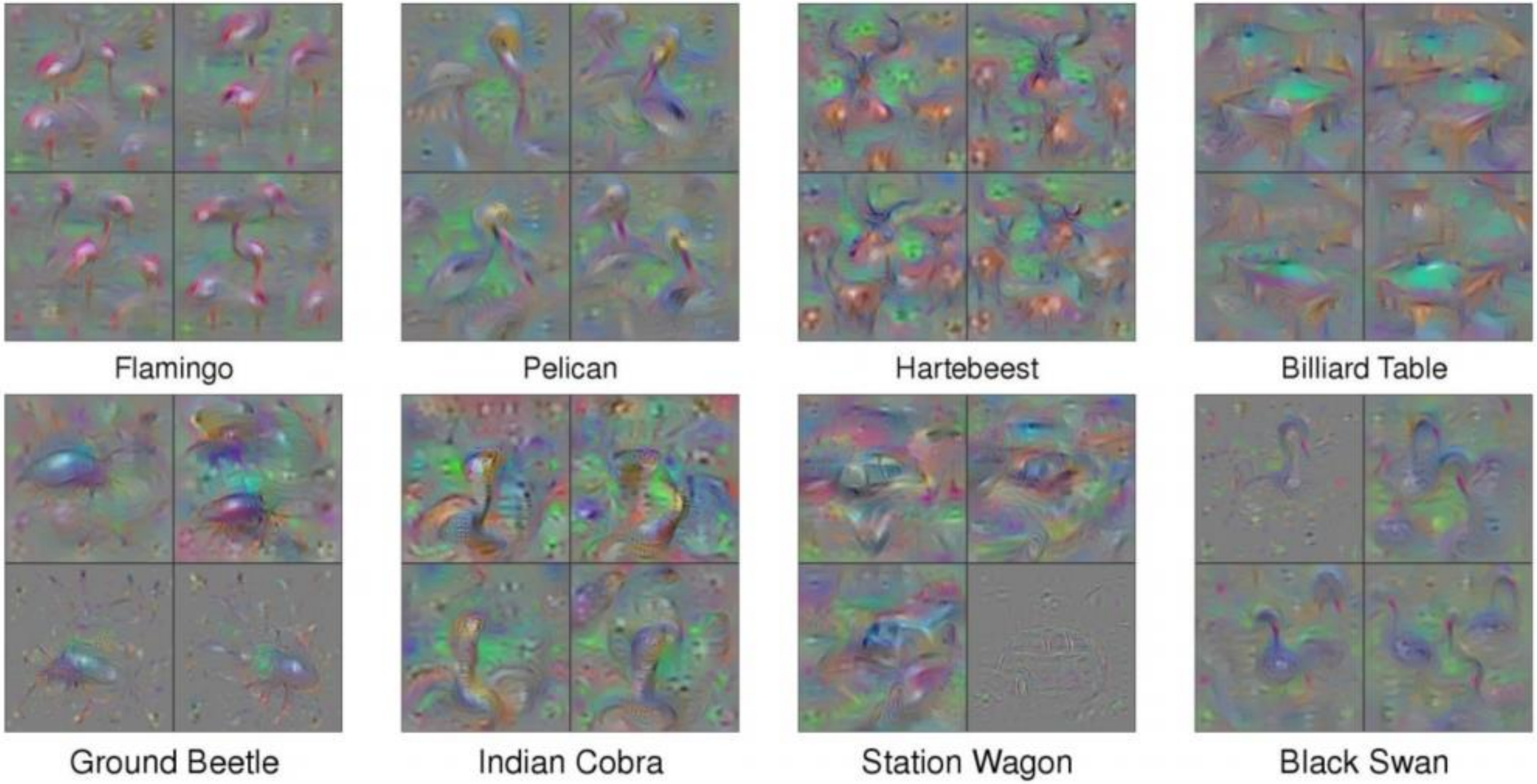


Springerberg et al, Striving for Simplicity: The All Convolutional Net (ICLR 2015 workshops)

What About Doing Gradient Ascent?



- Want to maximize the i-th output of the softmax
- Can compute the gradient of the i-th output of the softmax with respect to the *input x* (the W's and b's are fixed to make classification as good as possible)
- Perform gradient ascent on the *input*



Flamingo

Pelican

Hartebeest

Billiard Table

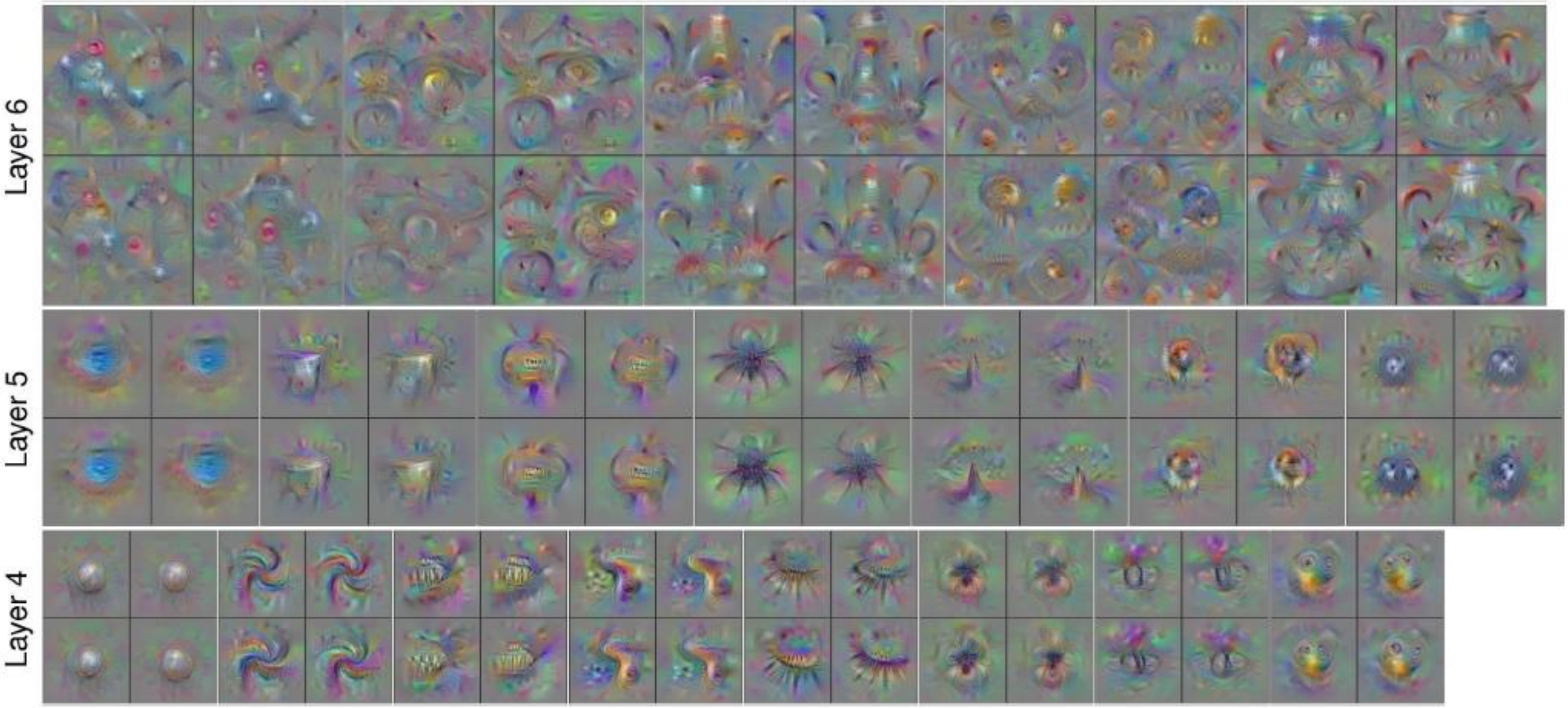
Ground Beetle

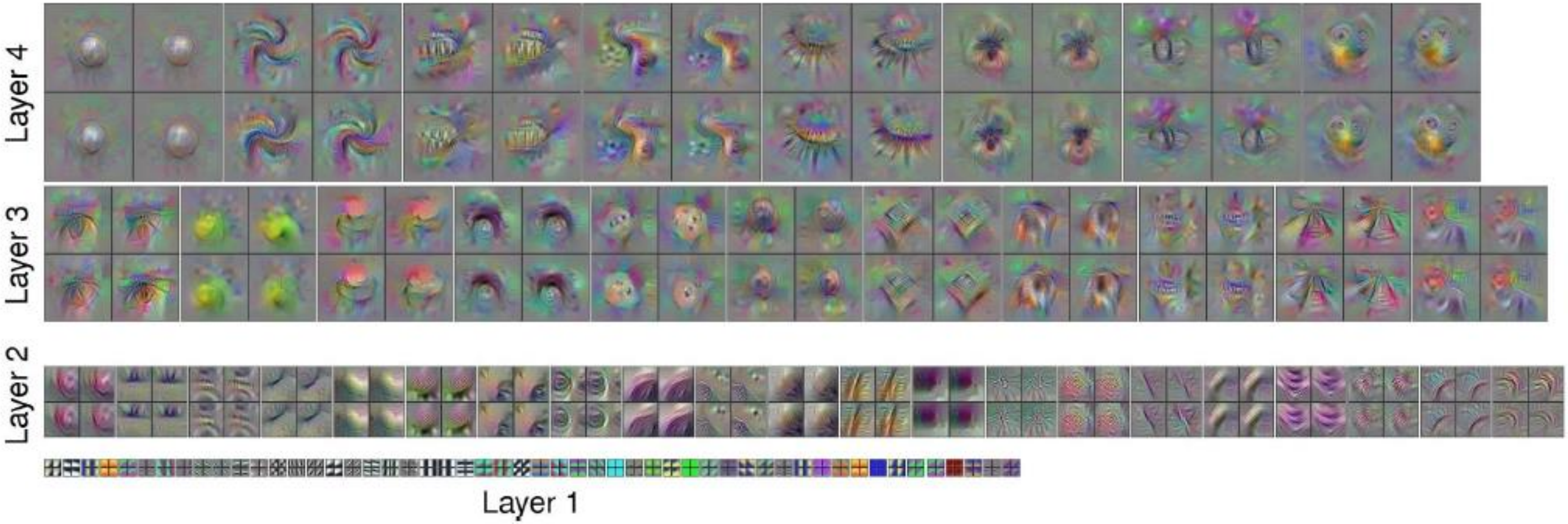
Indian Cobra

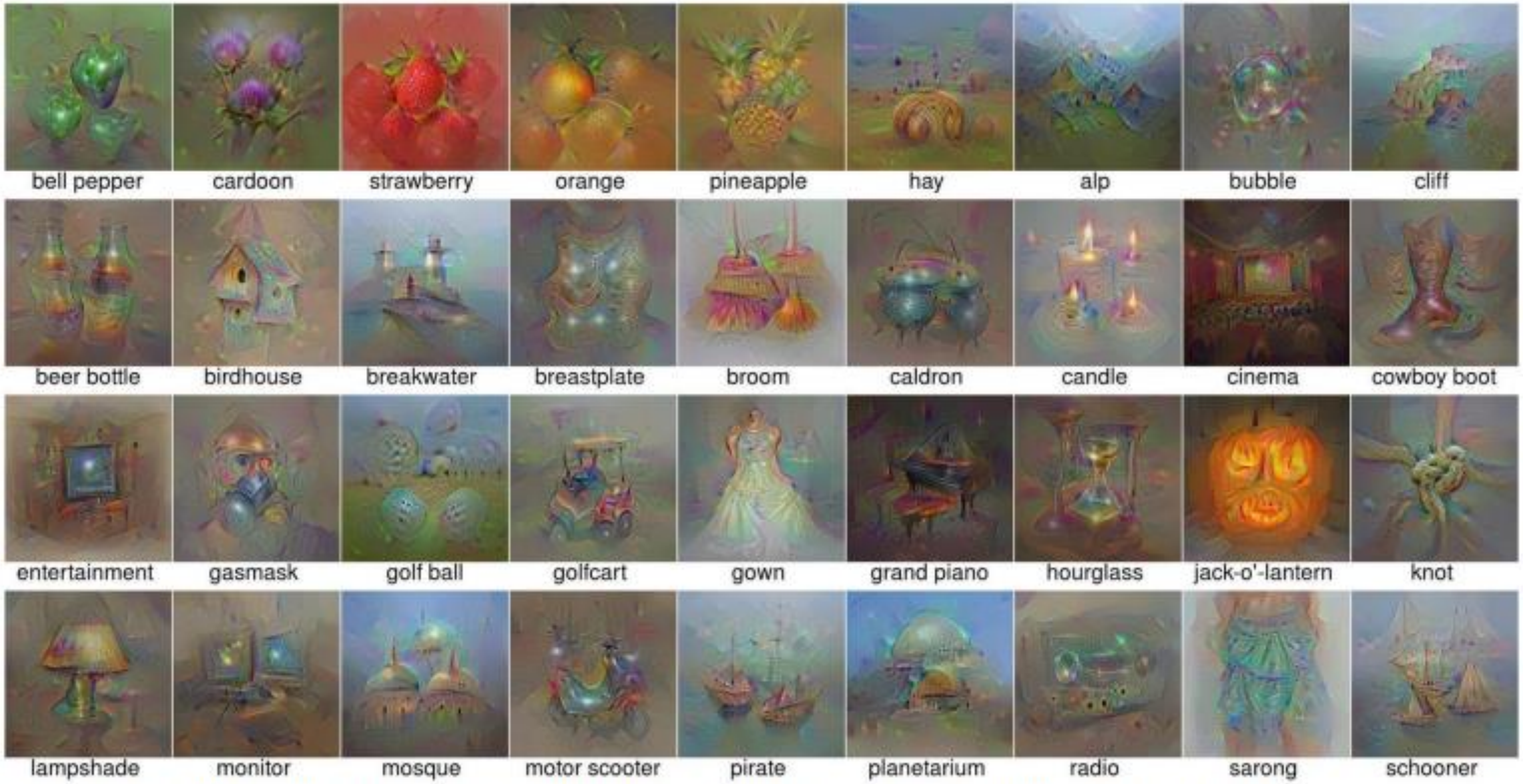
Station Wagon

Black Swan

Yosinski et al, Understanding Neural Networks Through Deep Visualization (ICML 2015)


Layer 6
Layer 5
Layer 4

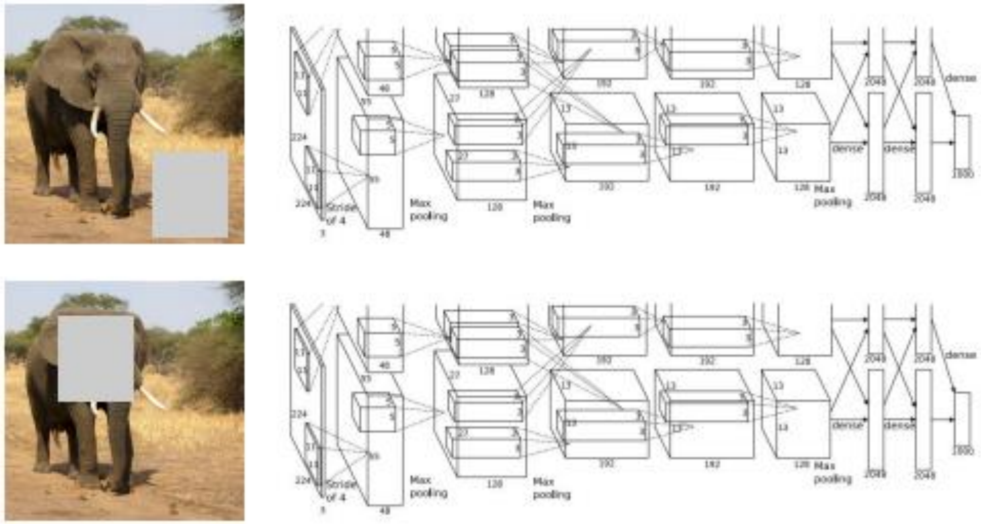

Layer 4
Layer 3
Layer 2
Layer 1

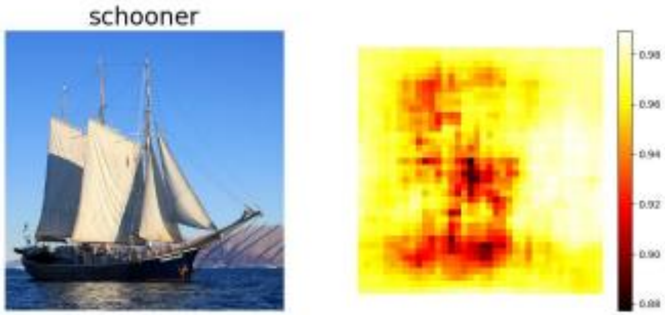


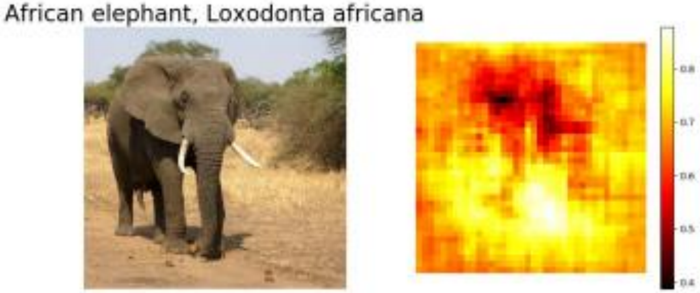
Nguyen et al, "Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks", ICML Visualization for Deep Learning Workshop 2016.
Figures copyright Anh Nguyen, Jason Yosinski, and Jeff Clune, 2016; reproduced with permission.

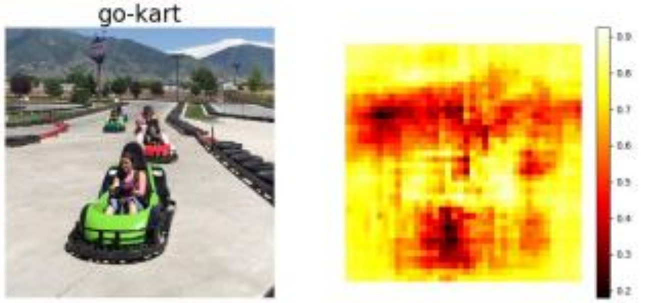
Which pixels matter: Saliency via Occlusion

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





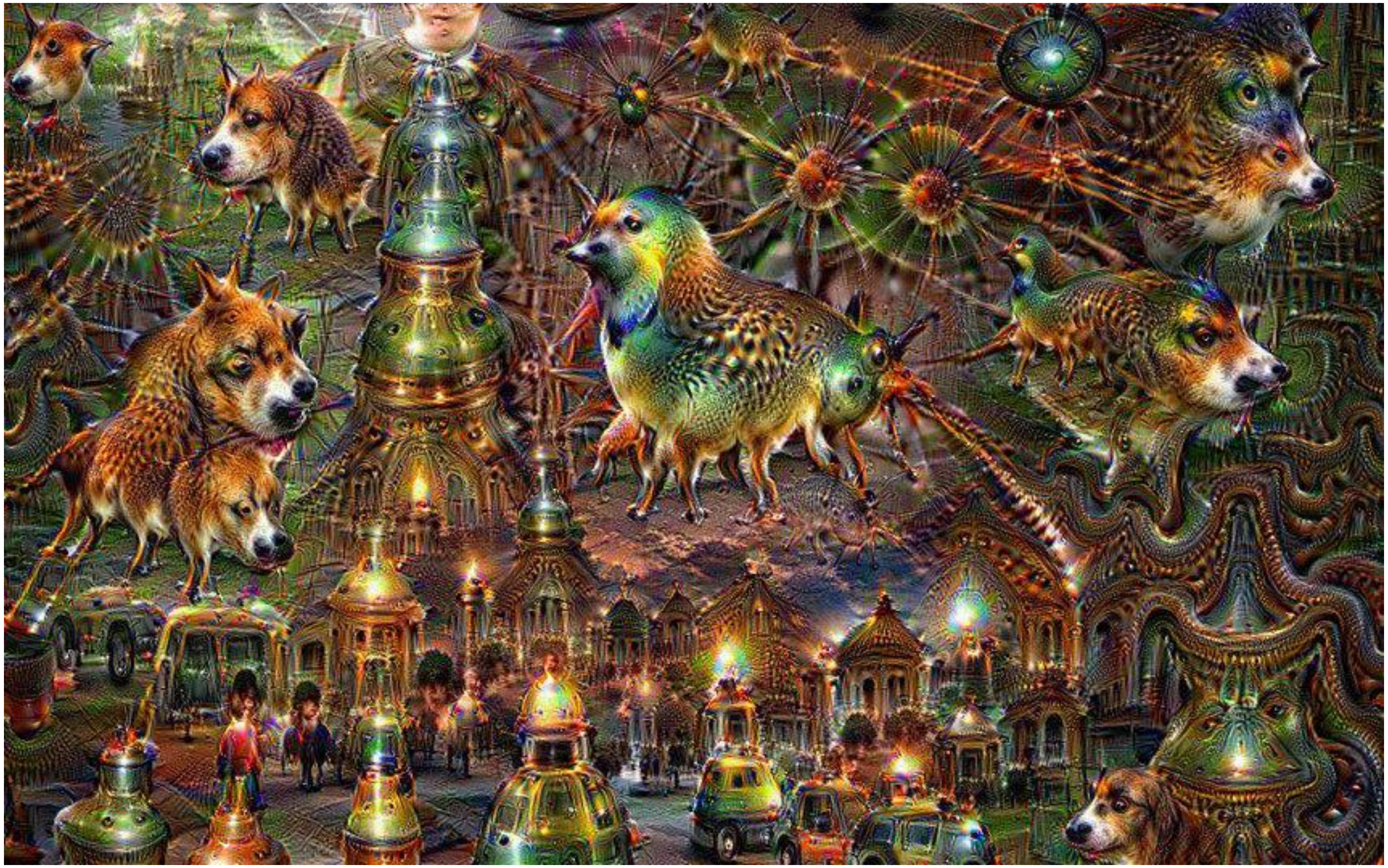




Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

Boat image is CC0 public domain
Elephant image is CC0 public domain
Go-Karts image is CC0 public domain

Deep Dream



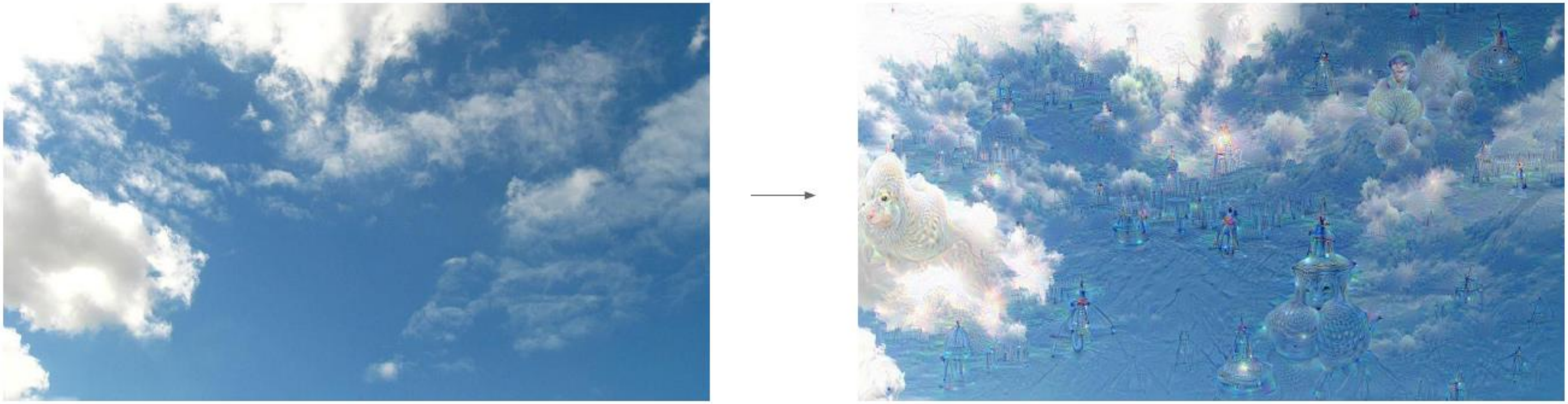
Deep Dream

- Idea: want to exaggerate details in the image that look a little bit like recognizable objects
- Overview
  - Pick a layer in the ConvNet
  - It will have some neurons that are highly activated
  - There is a trade-off here: we can’t make *all* of them be even more highly activated simultaneously
  - Idea: make the rich get richer. Change the input x with the most highly activated neurons influencing the change in the input the most

Change the input x with the most highly activated neurons influencing the change in the input the most

- Set the gradient at the layer that we picked to be *equal to the activation at that layer*
  - A hack: this is not the gradient at all
- Backpropagate the gradient to figure out how much to change the input
- Repeat

- Result: a feedback loop where the image looks more and more like the objects that were kind of detected at first



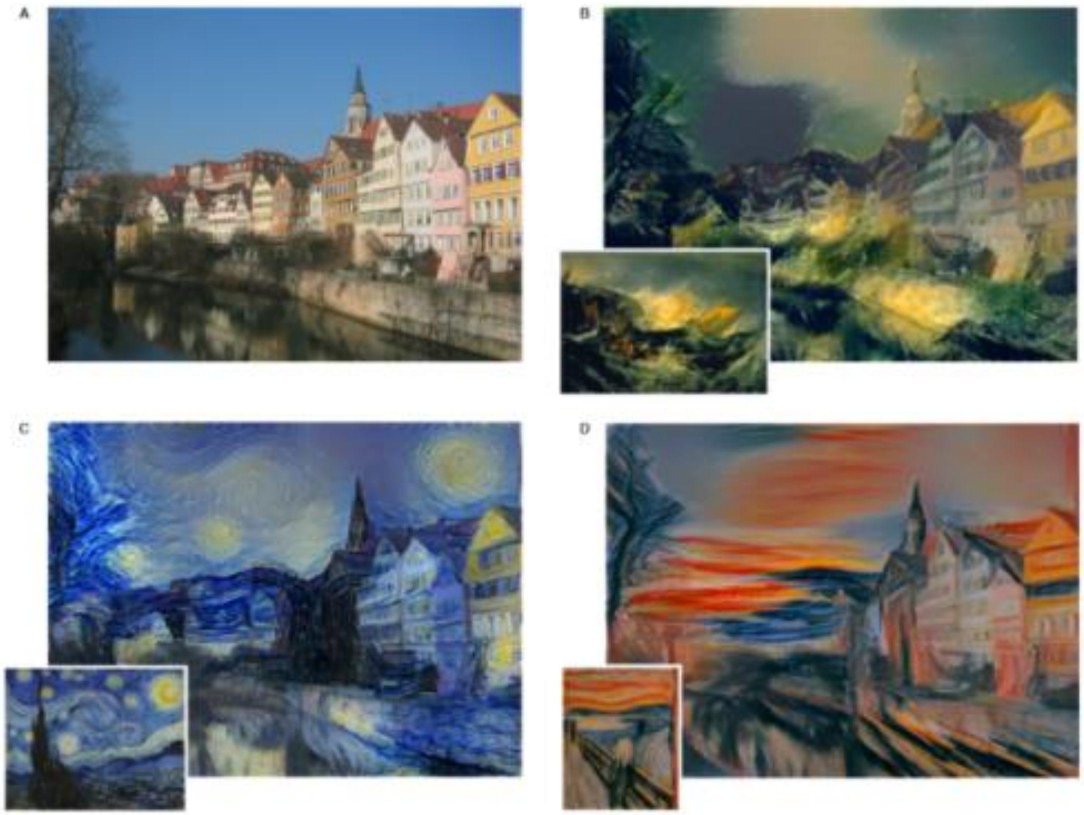
"Admiral Dog!"

"The Pig-Snail"

"The Camel-Bird"

"The Dog-Fish"

Neural Style Transfer



Gatys, Ecker, Bethge. “A Neural Algorithm of Artistic Style”
(http://arxiv.org/abs/1508.06576)

Neural Style

- Task: given an input photo I and a painting P, produce a photo with the same contents as I, but with the style P
- Idea: use gradient descent again to change the input with the weights of the ConvNet constant, with a cost function that keeps x close to I (i.e., preserve the content) and makes the style of x close to the style of P
- Use a ConvNet that works for image classification
    - It knows how to represent images well

Cost function: content

- To keep the content close to the input I, make sure that $|x - I|^2$ stays small
  - Not the best idea! The pixels might be a completely different colour in the x, but x and I can still be similar in content
- Even better: make sure that all the activations in all the different layers for the original image and for x stay the same:
- Make sure that $\mathrm{L}_{content}(x, I) = \sum_{l,i,j}(F(x)^{l}_{i,j} - F(I)^{l}_{i,j})^2$ is small
  - $F(y)^{l}_{(i,j)}$ is the activation at layer *l* at location *j* in feature map *i*, for input *y*
- Will make sure that the high-level features in the image stay the same, too

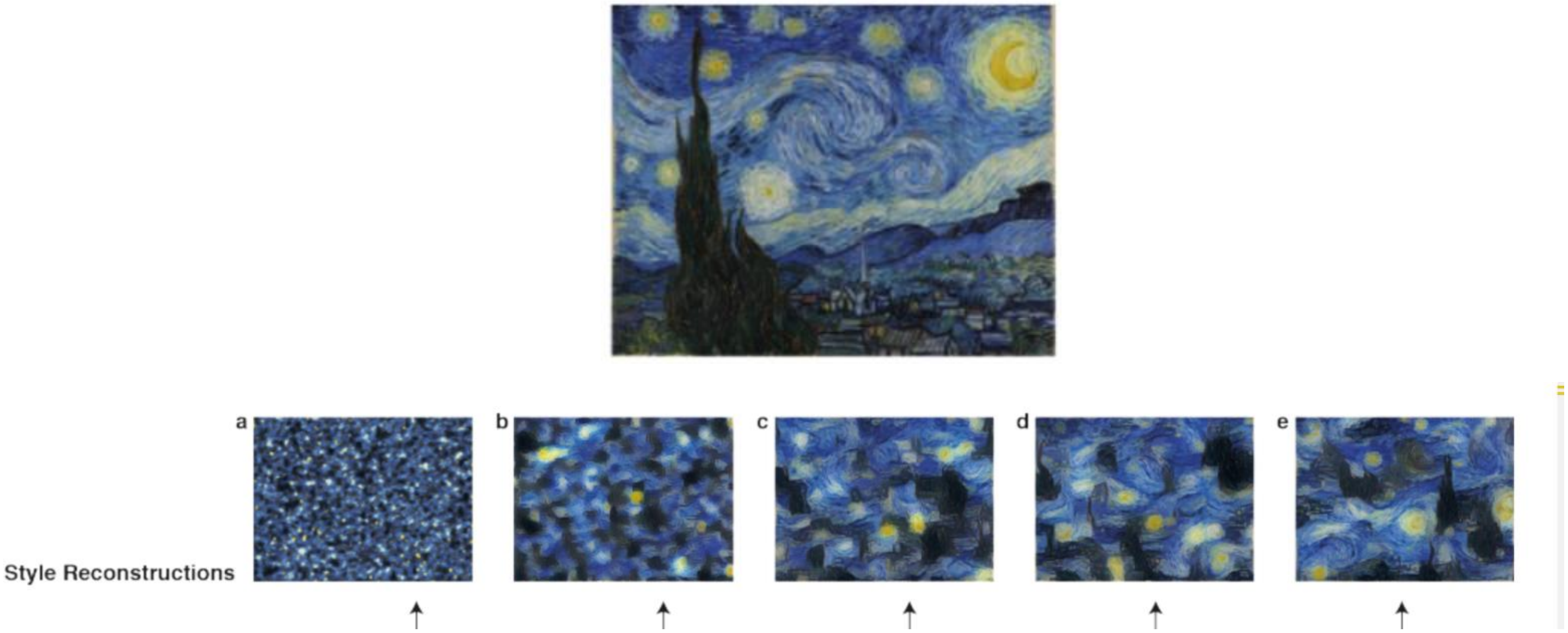
Style

- Define the Gram matrix at layer las

$$G_{ij}^{l}(y) = \sum_{k} F(y)^{l}{}_{ik} F(y)^{l}{}_{jk}$$

- Discovery: the Gram matrix represents the style
  - $G_{ij}^{l}(y)$ is large if at layer *l,* the i-th feature and the j-th feature tend to be discovered together a lot
    - (i.e., a lot of k’s for which the product of both activations is large)

Style Reconstructions



- “Blue strokes at 40 degrees co-occur with blue strokes at 45 degrees?”

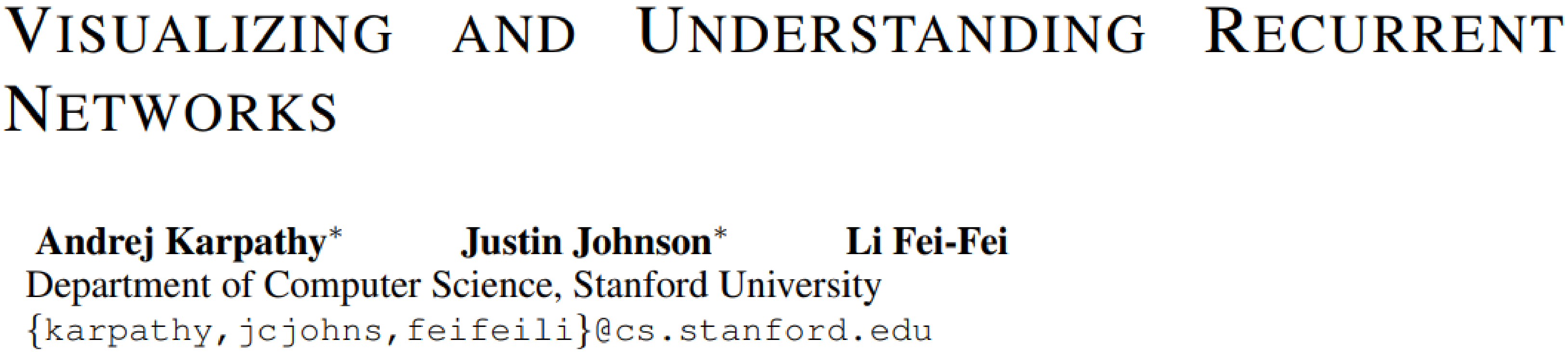
Cost function: style

Divide by the size of the layer to make sure every layer influences things equally for now

- $E_l(x,P) = \frac{1}{4N_l^2M_l^2}\sum_{i,j}(G_{ij}^l(x) - G_{ij}^l(P))^2$
- $L_{style}(x,P) = \sum_l w_l E_l(x,P)$

Cost function: overall

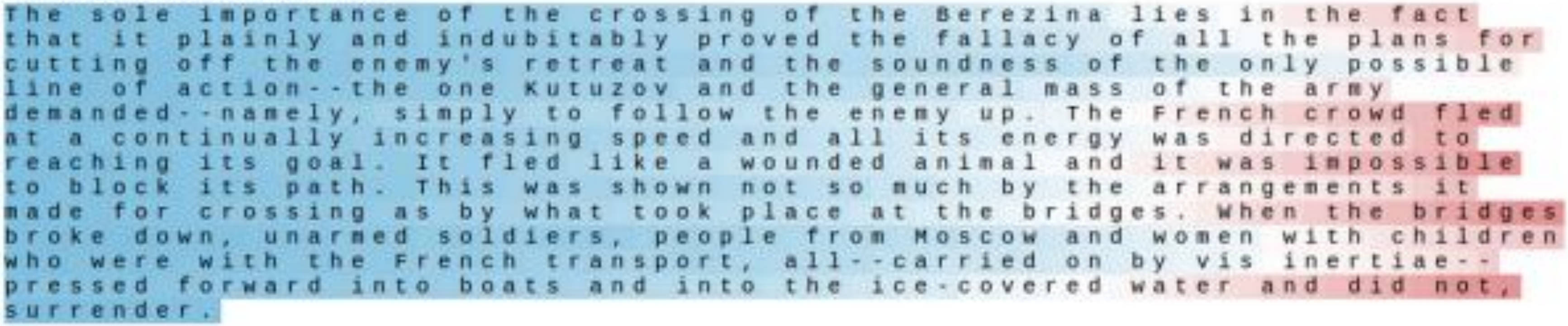
- $cost(x, I, P) = \alpha L_{style}(x, P) + \beta L_{content}(x, I)$



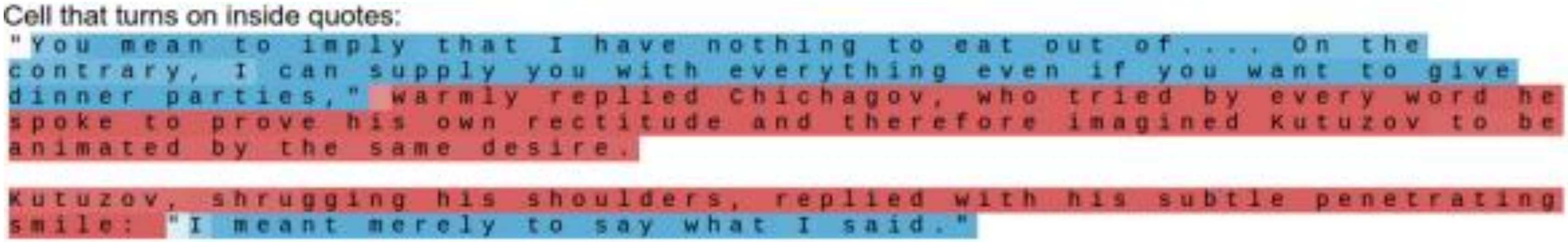
VISUALIZING AND UNDERSTANDING RECURRENT NETWORKS

Andrej Karpathy* **Justin Johnson*** **Li Fei-Fei**
Department of Computer Science, Stanford University
{karpathy,jcjohns,feifeili}@cs.stanford.edu

Cell sensitive to position in line:



The sole importance of the crossing of the Berezina lies in the fact
that it plainly and indubitably proved the fallacy of all the plans for
cutting off the enemy's retreat and the soundness of the only possible
line of action--the one Kutuzov and the general mass of the army
demanded--namely, simply to follow the enemy up. The French crowd fled
at a continually increasing speed and all its energy was directed to
reaching its goal. It fled like a wounded animal and it was impossible
to block its path. This was shown not so much by the arrangements it
made for crossing as by what took place at the bridges. When the bridges
broke down, unarmed soldiers, people from Moscow and women with children
who were with the French transport, all--carried on by vis inertiae--
pressed forward into boats and into the ice-covered water and did not,
surrender.

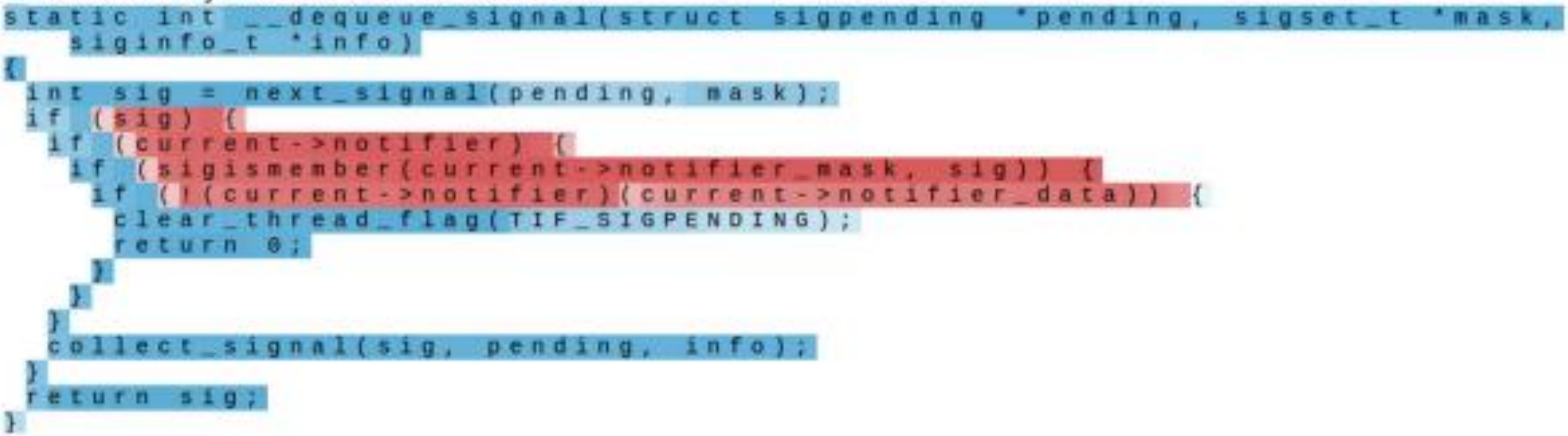


Cell that turns on inside quotes:

"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

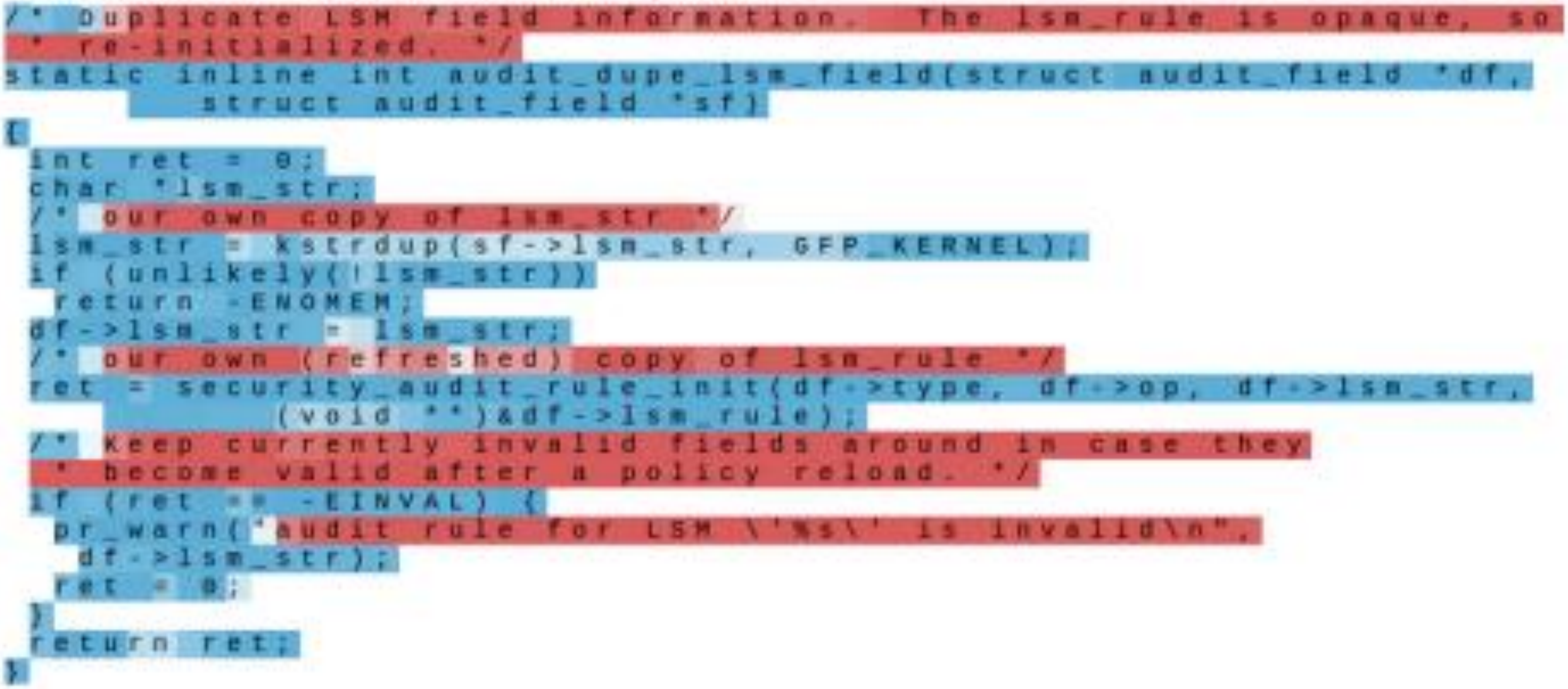
Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

Cell that robustly activates inside if statements:



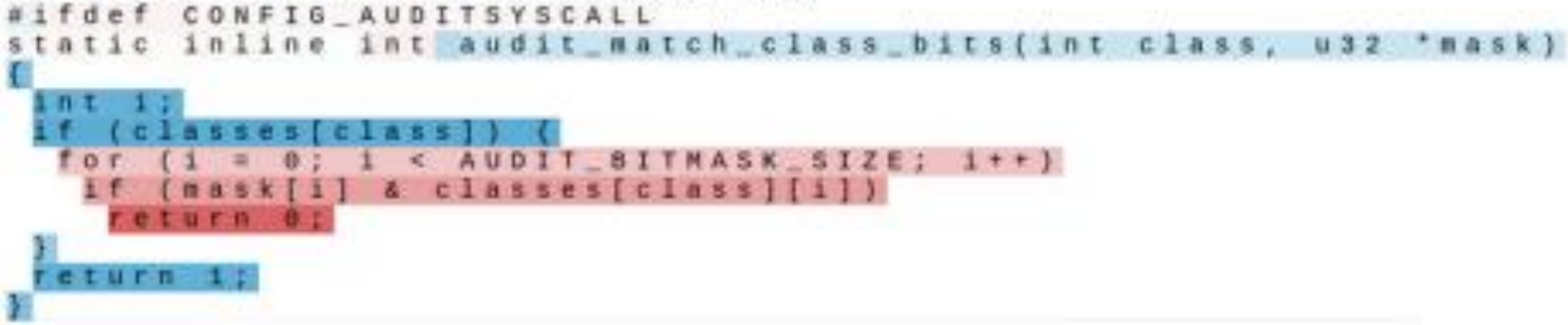
```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
    siginfo_t *info)
{
 int sig = next_signal(pending, mask);
 if (sig) {
  if (current->notifier) {
   if (sigismember(current->notifier_mask, sig)) {
    if (!(current->notifier)(current->notifier_data)) {
     clear_thread_flag(TIF_SIGPENDING);
     return 0;
    }
   }
  }
  collect_signal(sig, pending, info);
 }
 return sig;
}
```

Cell that turns on inside comments and quotes:



```
/* Duplicate LSM field information.  The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
        struct audit_field *sf)
{
 int ret = 0;
 char *lsm_str;
 /* our own copy of lsm_str */
 lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
 if (unlikely(!lsm_str))
  return -ENOMEM;
 df->lsm_str = lsm_str;
 /* our own (refreshed) copy of lsm_rule */
 ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
          (void **)&df->lsm_rule);
 /* Keep currently invalid fields around in case they
  * become valid after a policy reload. */
 if (ret == -EINVAL) {
  pr_warn("audit rule for LSM \'%s\' is invalid\n",
   df->lsm_str);
  ret = 0;
 }
 return ret;
}
```

Cell that is sensitive to the depth of an expression:



```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
 int i;
 if (classes[class]) {
  for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
   if (mask[i] & classes[class][i])
    return 0;
 }
 return 1;
}
```