Course Notes: Deep Learning for Visual Computing

Peter Wonka

August 30, 2021

Contents

1 Visualization

1.1 Literature

- distill.pub article on Feature Visualization - amazing polish and visual quality
- Deep Visualization Toolbox Code Repository - try out things yourself

1.2 Fancy 3D renderings

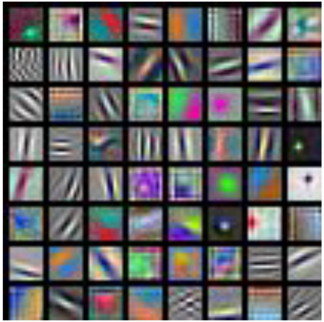
- Beautiful 3D visualization by Denis Dmitriev
- Looks great
- Hard to see what is going on

1.3 Tone Mapping and Range Mapping

- Visualizing arrays of values in the range $[min, max]$ needs **Tone Mapping** or **Range Mapping**
  - Mapping min to 0 and max to 1 (popular)
  - Map values < 0 to 0 and value > 1 to 1
- Warning: results can look quite different when the range mapping changes
  - Example: values are in a small range, e.g. $[-\epsilon, +\epsilon]$, but values are mapped to the complete visible range $[0, 1]$
  - Example: values are far outside the visible range, e.g. $[-100, 100]$ but are mapped inside the visible range

1.4 Visualizing Filters Directly as RGB images

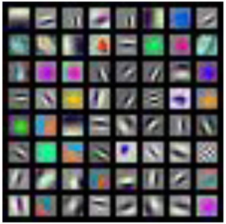
- Works great for filters with size $W \times H \times 3$
  - typically first layer CONV filters
- Does not work great for small filters (e.g. 3×3)
- Does not work great for filters in later layers
- Does not work great for filters with many channels (e.g. $3 \times 3 \times 512$)



AlexNet:
64 x 3 x 11 x 11

ResNet-18:
64 x 3 x 7 x 7

ResNet-101:
64 x 3 x 7 x 7



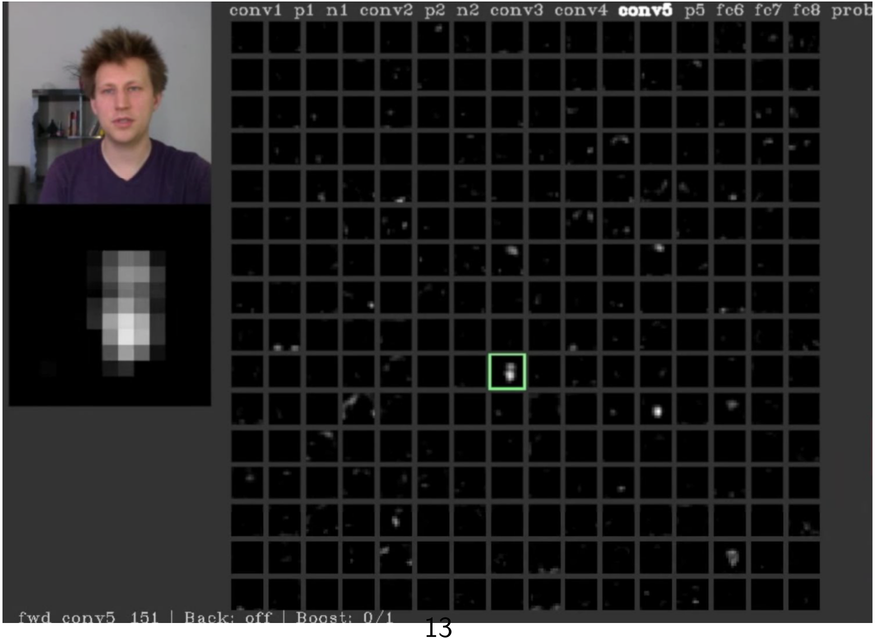
DenseNet-121:
64 x 3 x 7 x 7

1.5 Visualizing Filters as Collection of Grayscale Images

- Filters of size $W \times H \times C_{in}$ with $C_{in} > 3$ can be visualized as C_{in} grayscale images
- For one layer there will be $C_{out} \times C_{in}$ grayscale images
- Discussion: hard to interpret
- Example visualization for one layer of a smaller network:
  - 20 filters or size $7 \times 7 \times 16$

1.6 Visualizing Activation Tensors

- Each channel of a tensor (activation map) can be shown as grayscale image
- A $W \times H \times C$ tensor gives C grayscale images of size $W \times H$.

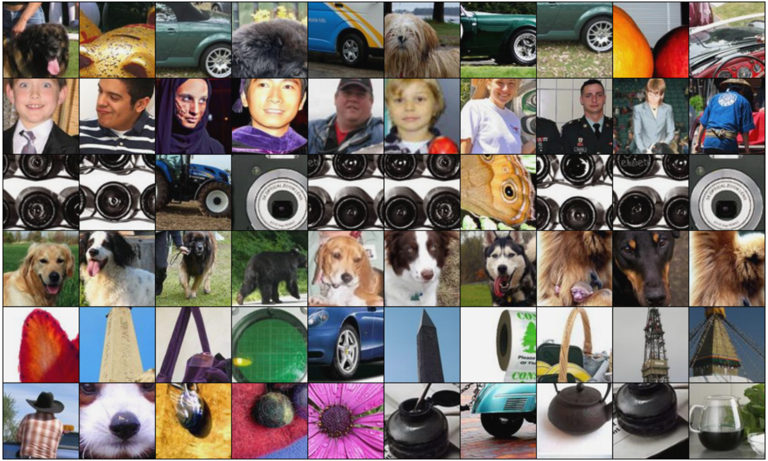

conv1 p1 n1 conv2 p2 n2 conv3 conv4 conv5 p5 fc6 fc7 fc8 prob
fwd conv5 151 | Back: off | Boost: 0/1

- Example shows an activation map that seems to correspond to faces
- Video: youtube: Deep Visualization Toolbox, 4 min

1.7 Maximally Activating Patches

- For a channel of a tensor, find the k (e.g. 9) image patches that produce the maximal response (at any pixel).
- e.g. conv5 is AlexNet is $13 \times 13 \times 128$, pick one of the 128 channels
- Run many images through the network and find the k largest activation values
- Crop the patches from the input images that correspond to the receptive field / field of view of largest value

5
15
ES
mo
CH
lon
en
CH
AC
ops!

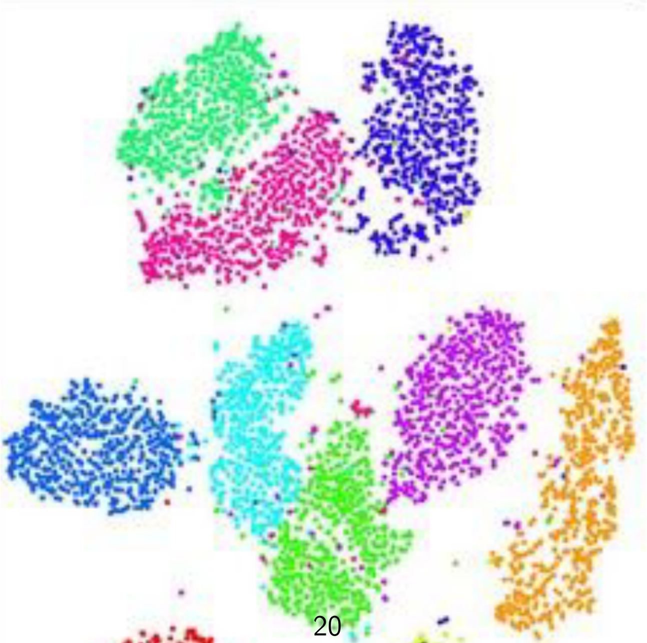


- Upper examples are from an earlier layer in the network
- Results enable a very good semantic interpretation

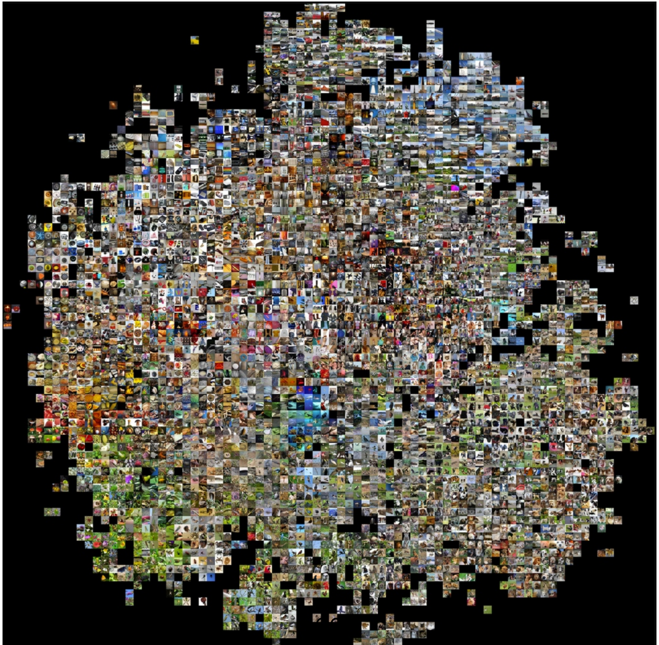
- Problem: activations are also sensitive to patches very different from the training data

1.8 Embedding Images into 2D

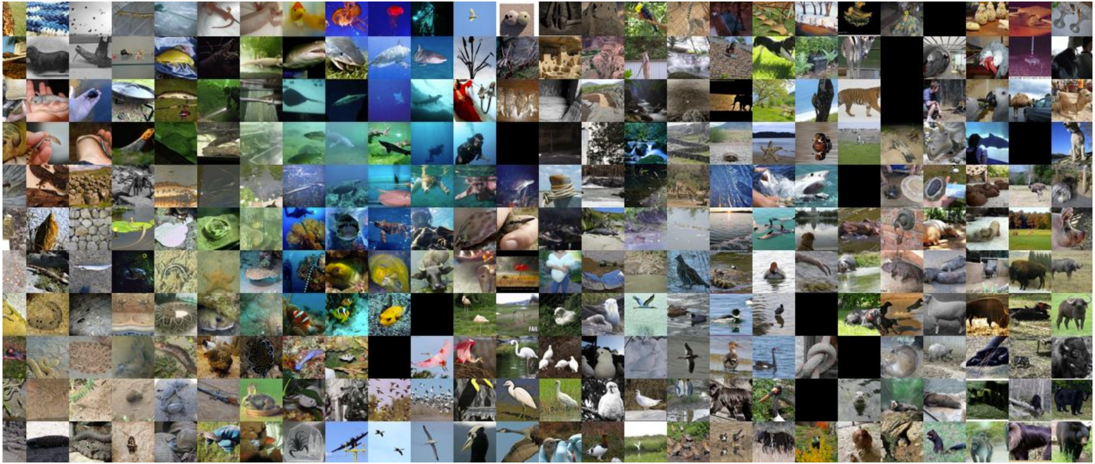
- Activations can be seen as features
- We can pick a layer, typically a late layer to embed input images in $2D$
- Example: pick an FC $1 \times 1 \times 4096$ dimensional layer
- Use dimension reduction to map D dimensions to 2 dimensions:
- Simple dimension reduction: PCA
- More popular for visualization: t-SNE



- Visualization of MNIST images
- Each image is mapped to a point and color coded according to its class
- We can observe a nice separation of 10 classes



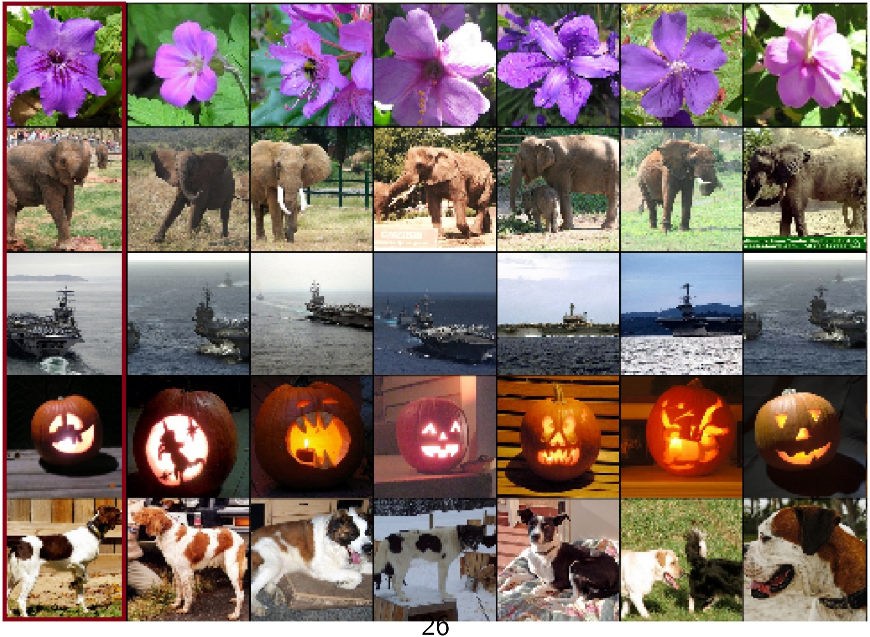
- Visualization example from Kaparthy
- Each image is visualized as downsampled image
- Needs processing to map images to grid cells, resize images, and figure out occlusion
- Hard to see much, so we also show a zoomed in version



- Note that semantically similar images are closeby, e.g. marine animals

1.9 Nearest Neighbors in Feature Space

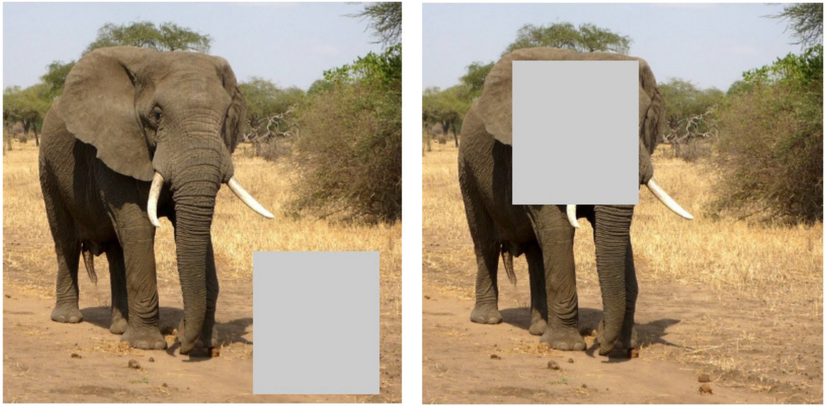
- Visualize nearest neighbors in feature space, e.g. a late FC layer



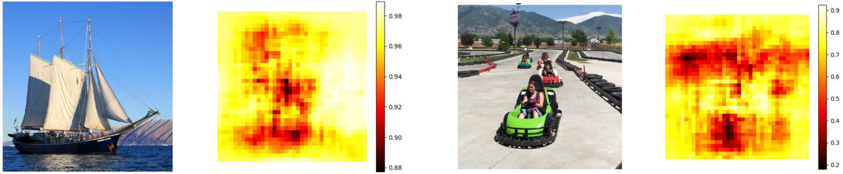
- Five imagenet query images in the first column.
- The remaining columns show six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the query image.

1.10 Saliency Maps via Occlusion

- Mask part of the image with a rectangle
    - Set all values inside the rectangle to the average pixel value
    - typically 0 in transformed space, but some gray value in regular RGB space
- Slide rectangle over the image and record predicted probability of the correct class by the CNN



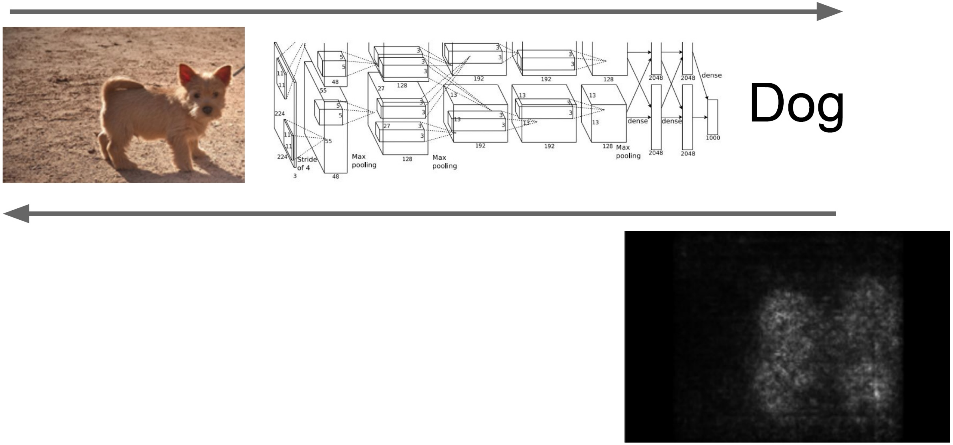
- Intuition: class of elephant should drop when mask moves over elephant face



- Example shows saliency maps for schooner left and go-kart right
- Intuition: locations where class probability drops the most are most important

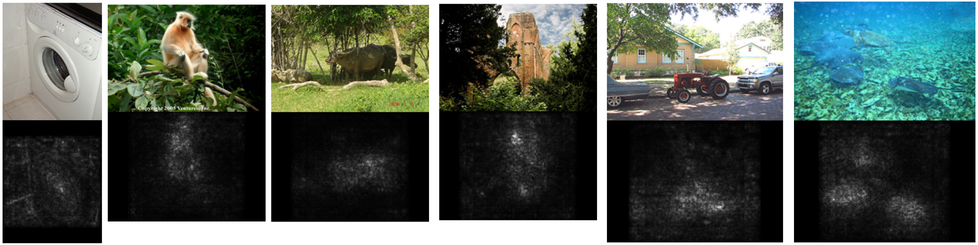
1.11 Saliency via Gradients

- What pixels in the input image should change to improve the class score?
- Forward pass: compute class score
- Backward pass:
  - Compute gradient of class score with respect to image pixels
  - Take absolute value and max over RGB channels



Dog

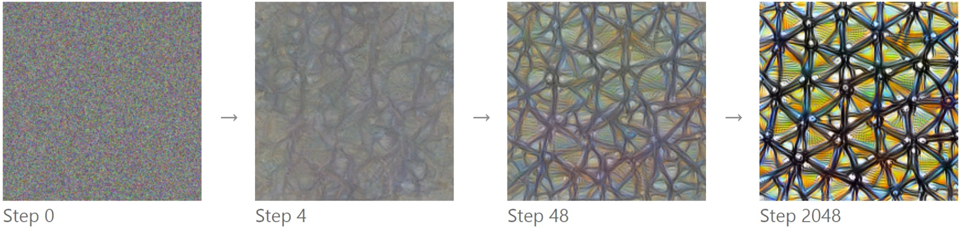
- Intuition is that high valued pixels contribute more to the dog class prediction



- Various other examples
- Problem: salient regions are somewhat visible, but hard to interpret the details

1.12 Visualizing using Gradient Ascent

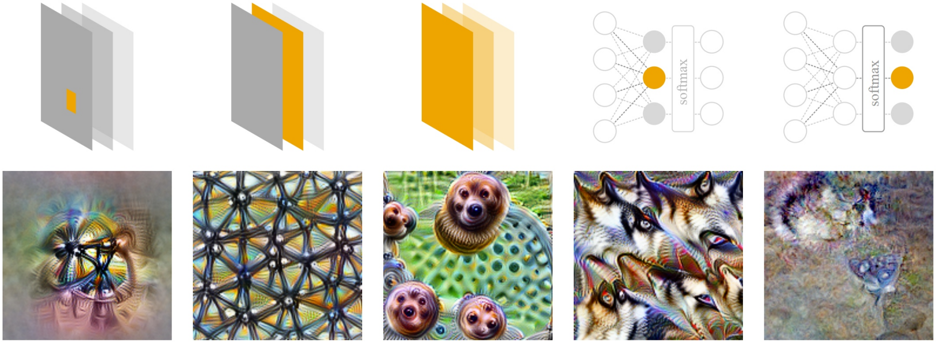
- General Idea: use optimization to increase / maximize a particular network response
  - e.g. use gradient ascent



Step 0 → Step 4 → Step 48 → Step 2048

- Design choice 1: What objective function / activation to maximize?
  - single neuron: single value in a $(W \times H \times C)$ activation tensor
  - channel, activation map: one channel in a $(W \times H \times C)$ activation tensor

- activation tensor: all values in a $(W \times H \times C)$ activation tensor
- class logits: class scores before the softmax activation function
- class probabilities: class probabilities after the softmax activation function



Neuron layer$_n$[x,y,z]

Channel layer$_n$[:,:,z]

Layer/DeepDream layer$_n$[:,:,:]^2

Class Logits pre_softmax[k]

Class Probability softmax[k]

- Design choice 2: start from random image vs. zero image vs. start from a given image
- Design choice 3: how to regularize the optimization
- Compute

$$I^{viz} = \arg\max_{I} f(I) + R(I) \tag{1.1}$$

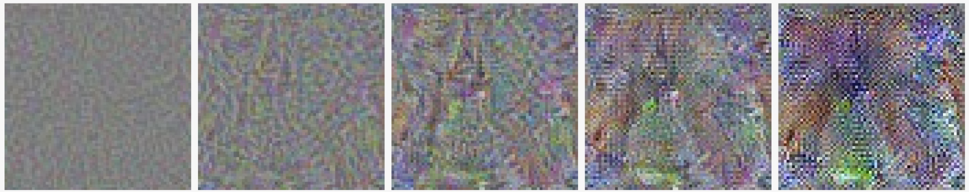
  - $f(I)$ is a function on tensor values computed by a forward pass of I through the network
  - $R(I)$ is a regularizer to generate more natural images

1.13 Problem with Class Probabilities

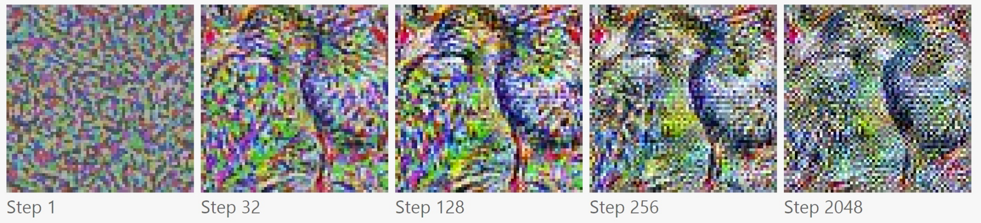
- Should we use **class probabilities** or **raw class scores** (transformed by softmax into probabilities)
- Class probabilities are generally not used
- Conjecture: to maximize a given class probability it's easier to reduce other raw class scores than increase the raw desired score before softmax
- Conjecture: Softmax makes things harder to optimize in general

1.14 Optimizing without Regularizer

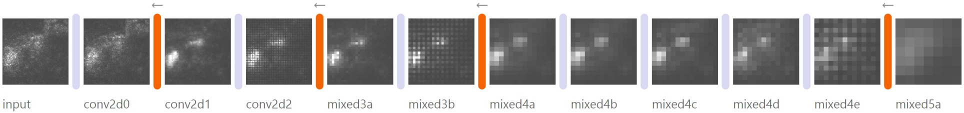
- Gives sometimes reasonable result when optimizing for a long time
- Often the results look completely random
- First example has a lower learning rate, second example has a higher learning rate
- Note the checkerboard artifacts in the results
- Can we really see a pattern?



Step 1 Step 32 Step 128 Step 256 Step 2048

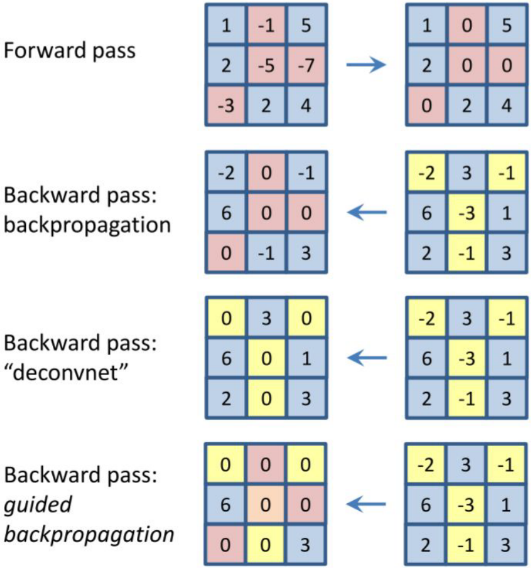


- Problem strided convolution and pooling layers create checkerboard patterns in the gradient:



1.15 Image Features via Guided Backpropagation

- Pick an intermediate activation tensor (e.g. conv5 $13 \times 13 \times 128$), pick channel, pick a very high activation
- Compute modified gradient of neuron with respect to input image pixels
- Modify backpropagation by only looking at positive gradients at each ReLU



Forward pass

1	-1	5
2	-5	-7
-3	2	4

→

1	0	5
2	0	0
0	2	4

Backward pass: backpropagation

-2	0	-1
6	0	0
0	-1	3

←

-2	3	-1
6	-3	1
2	-1	3

Backward pass: "deconvnet"

0	3	0
6	0	1
2	0	3

←

-2	3	-1
6	-3	1
2	-1	3

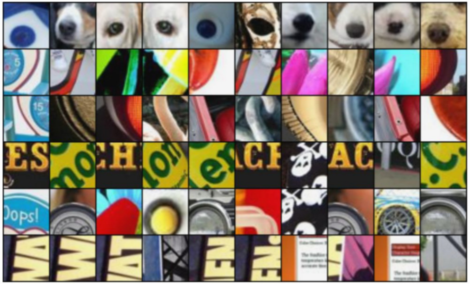
Backward pass: *guided backpropagation*

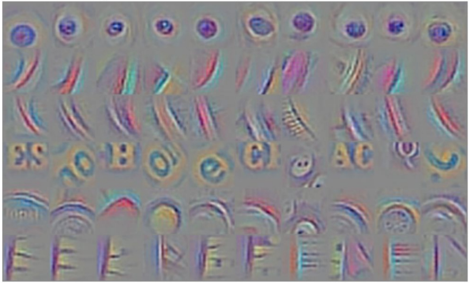
0	0	0
6	0	0
0	0	3

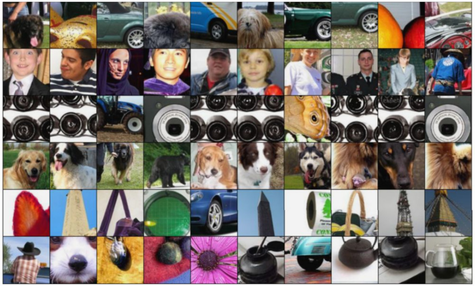
←

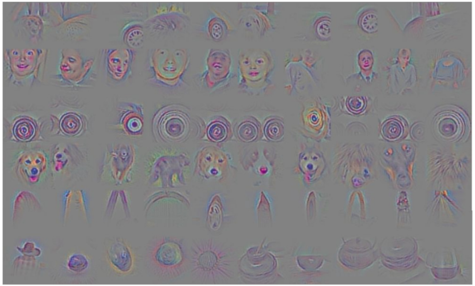
-2	3	-1
6	-3	1
2	-1	3

- Example visualizations below









- Left are the maximally activating patches, right are the results using guided backprop
- Interesting Note: the idea of modified backpropagation was introduced by Zeiler and Fergus as deconvolutional networks.
    - Their deconvolution layer for a conv layer is exactly the same as gradient computation
    - Their max unpooling is the same as gradient computation

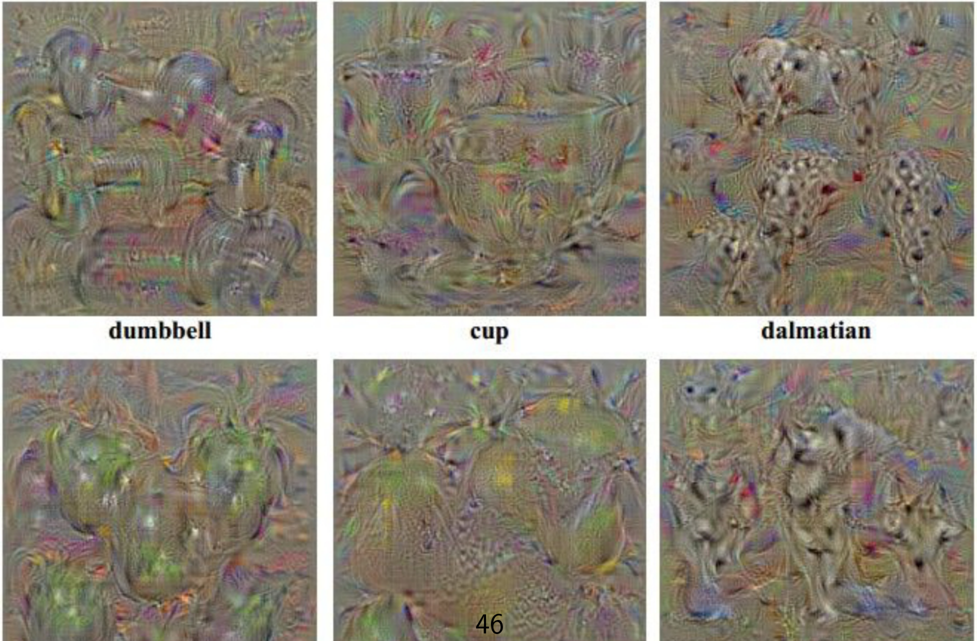
  - Their ReLU solution is slightly different (see above)
- Literature:
  - Springenberg et al., Striving for Simplicity: The All Convolutional Net, 2015

1.16 Maximizing Class Scores with L2

- Initialize Image I^{viz} to zeroes / random noise
- Repeat:
  - Forward pass through the network to compute tensors / loss function
  - Backprop to get gradient of loss values with respect to image pixels
  - Make a small change to the image
- Optimize:

$$I^{viz} = \underset{I}{\arg\max} \, S_C(I) - \lambda |I|_2^2 \tag{1.2}$$

  - S_C is the score for class C before softmax
  - squared two norm prevents pixels from getting too small or too large



dumbbell

cup

dalmatian

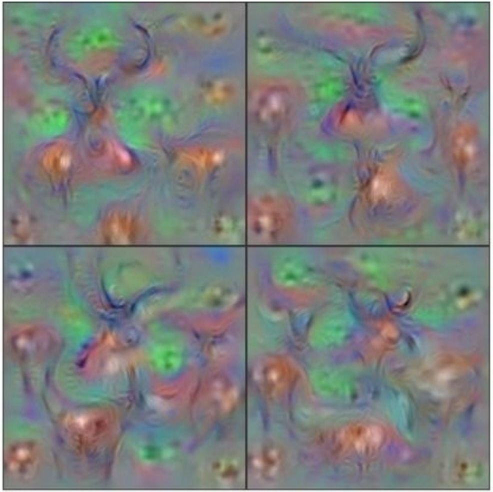
1.17 Maximizing Class Scores with L2++

- Literature: Yosinski et al., Understanding Neural Networks Through Deep Visualization, 2015
- Optimize as before:

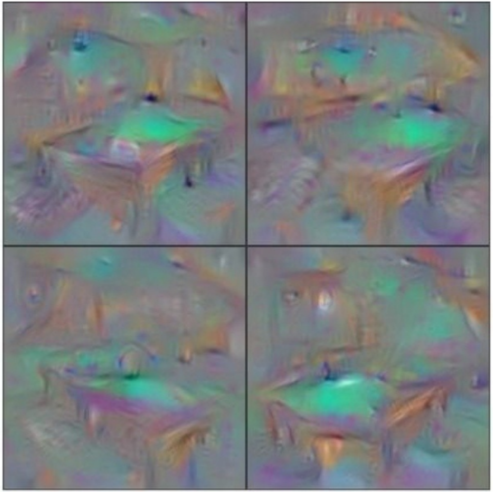
$$I^{viz} = \arg\max_{I} S_C(I) - \lambda |I|_2^2 \tag{1.3}$$

- In addition during iterations:
  - Blur the image using Gaussian kernels
    - only every few iterations to make it faster
  - Clip pixels with small values to 0
    - create empty regions rather than regions with small random patterns influencing the result
  - Compute $val =$ inner product of absolute value of gradient and pixel

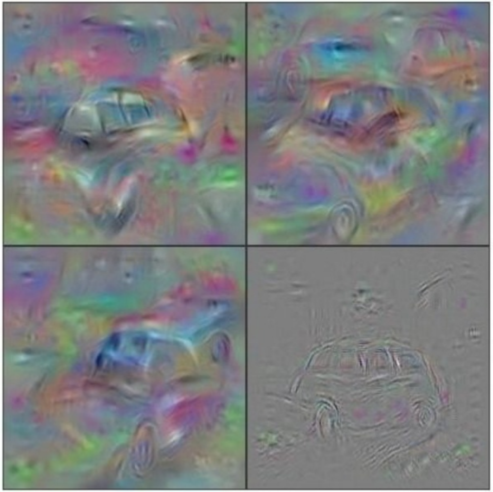
- Set pixel to 0 if val smaller than threshold
- Idea: clip pixels that do not contribute much



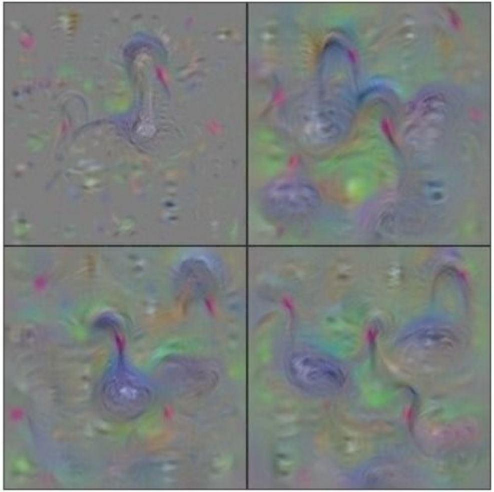
Hartebeest



Billiard Table



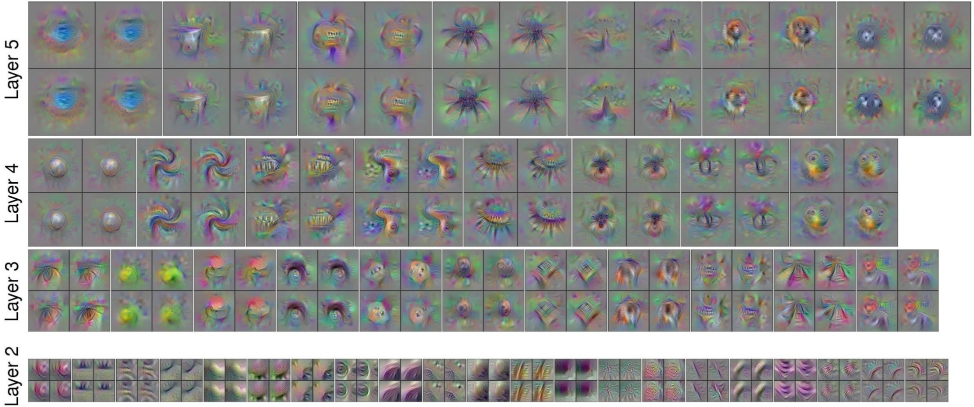
Station Wagon



Black Swan

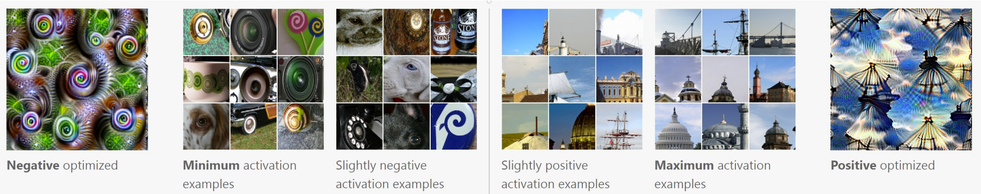
1.18 Maximizing Activations with L2++

- Do same computation as before, but maximize selected activations rather than class scores



1.19 Diversity

- Diversity problem: there can be multiple clusters of images that maximize the objective function
  - How can we find all representative images that maximize the objective function?



Negative optimized

Minimum activation examples

Slightly negative activation examples

Slightly positive activation examples

Maximum activation examples

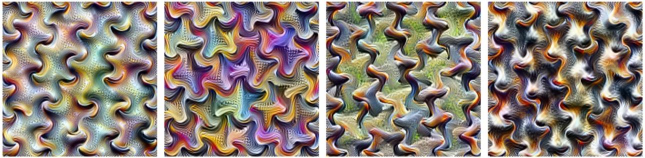
Positive optimized

- This neuron shows blue sky, truss structures, and architecture in representative patches
- What is the neuron really looking for?
  - Only sky, all three, certain grid pattern?

1.20 Diversity Examples

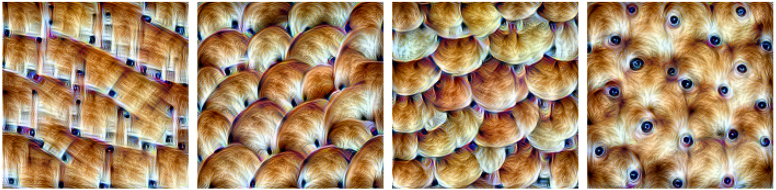
- Diversity optimization includes a term that makes new solutions different from existing solutions.
  - In these examples the diversity term is computed using the Gram matrix, similar to style transfer

Simple Optimization



Optimization with diversity reveals four different, curvy facets. *Layer mixed4a, Unit 97*

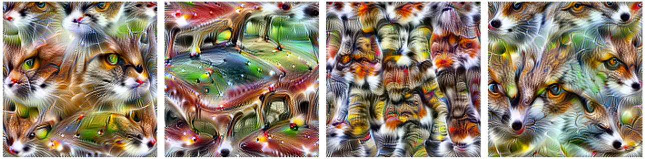
Dataset examples



Optimization with diversity. *Layer mixed4a, Unit 143*

Dataset examples

Simple Optimization



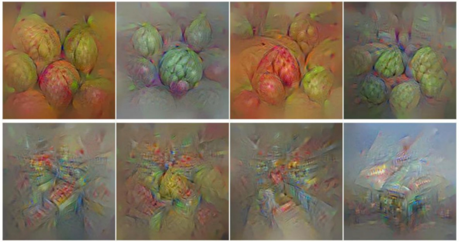
Optimization with diversity show cats, foxes, but also cars. *Layer mixed4e, Unit 55*

Dataset examples

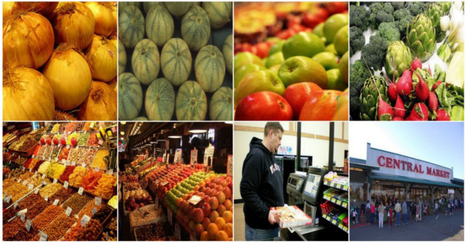
1.21 Multi-faceted Visualizations

- Some classes can be recognized due to multiple different types of features
- Multi-facated feature visualization generates different images leading to high activations
- Visualizations for the grocery store neuron and example training images recognized by the neuron as grocery store

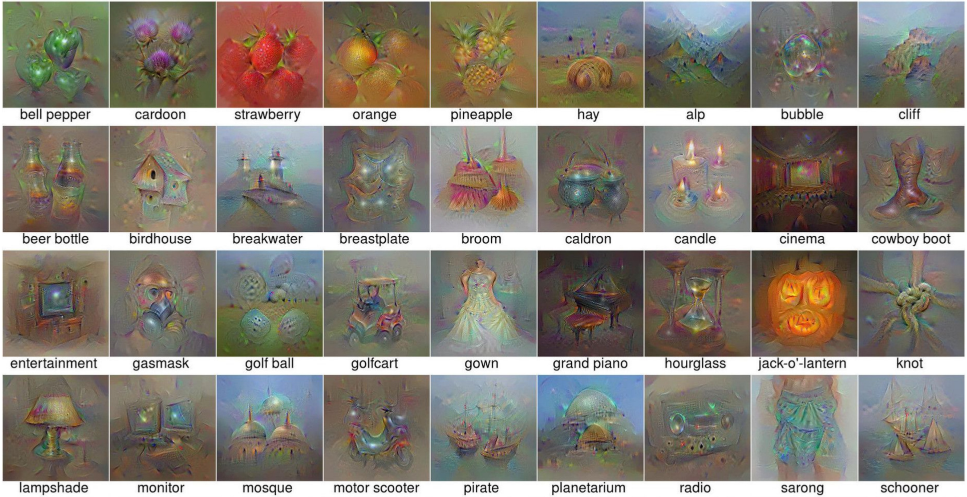
Reconstructions of multiple feature types (facets) recognized by the same “grocery store” neuron



Corresponding example training set images recognized by the same neuron as in the "grocery store" class



1.22 Multi-faceted Visualizations Gallery



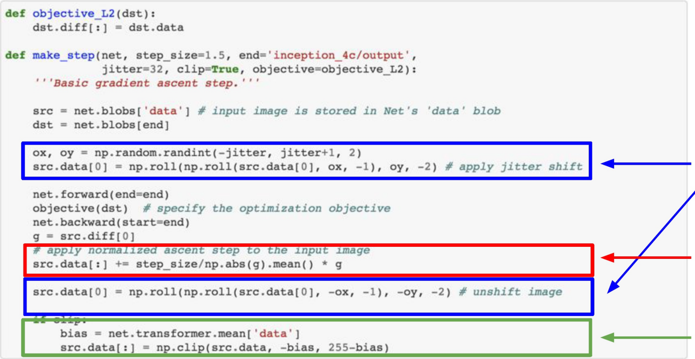
1.23 DeepDream

- Amplify activations at some layer in the network
  - Instead of maximizing activations
- Choose an image I and a layer l in the CNN; repeat
  - computate activations in l by a forward pass using I as input
  - Set gradient of chosen layer equal to its activations
  - Compute gradients of the image by backpropagation
  - Update I
- Equivalent to

$$I^{viz} = \arg\max_{I} \sum_{i} f_i(I)^2 \tag{1.4}$$

1.24 DeepDream Code

- Code is simple but uses a few tricks:
  - Jitter the image
  - L1 normalization of gradients
  - Clipping pixel values
  - Multi-scale processing



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

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

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

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

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

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

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

Code is very simple but it uses a couple tricks:

(Code is licensed under Apache 2.0)

Jitter image

L1 Normalize gradients

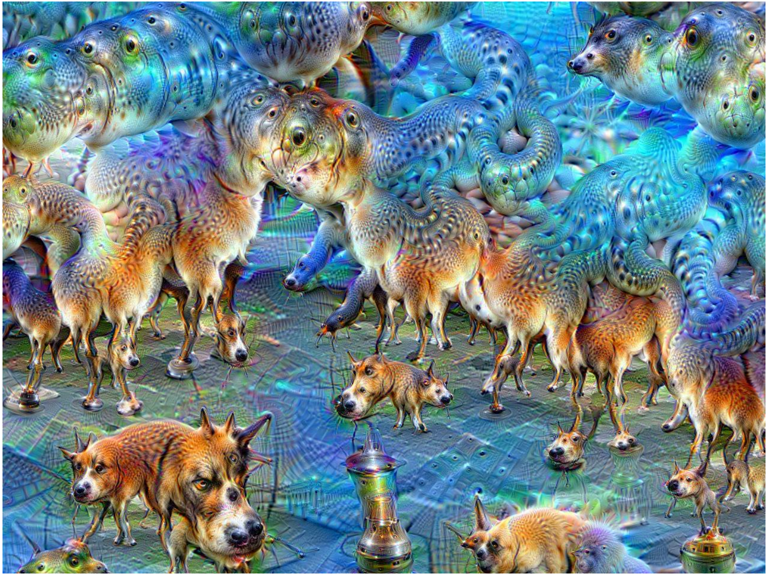
Clip pixel values

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

1.25 DeepDream Example

- Input Image:

- DeepDream on later layers



- DeepDream using data other than imagenet (with more architectural examples)