First layer of ConvNet:

↳ Visualizing the learned filters / weights

↳ Rescaled to 0-255

Q. Why use filters for visualizing?

↳ Because the thing that causes the inner product to excite maximally is a copy of the thing you are taking an inner product with.

Second layer:

↳ Second layer recieves these 16 7x7 input & does 7x7 conv with 20 filters
↳ We can't intercept these because these are NOT connected to the input image but rather output of first layer.
↳ Also, CL learns the relationship in first layer that make more complex features

Last layer:

↳ if we use nearest neighbour but instead using pixels from the last layer
↳ The feature map of two differently oriented images end up close to each other.

t-distributed stochastic neighbour embeddings:

↳ Non linear dimension reduction method
↳ Use 4096 from layer & apply t-SNE

↳ We have pixels of the image
↳ We have the 4096 d vector
↳ We use t-SNE to get a 2 dimensional coordinate from 4096 vector

NOTE:
1. A nueron corresponds to a single value that corresponds to a patch of the image
2. We use multiple filters to look at a multiple features

↳ We place the image at the coordinate

Maximally Activating patches:

↳ Pick a layer & a channel. Eg: conv5 is 128 x 13 x 13, pick a channel

↳ Choose the patch of image that maximally activate it.

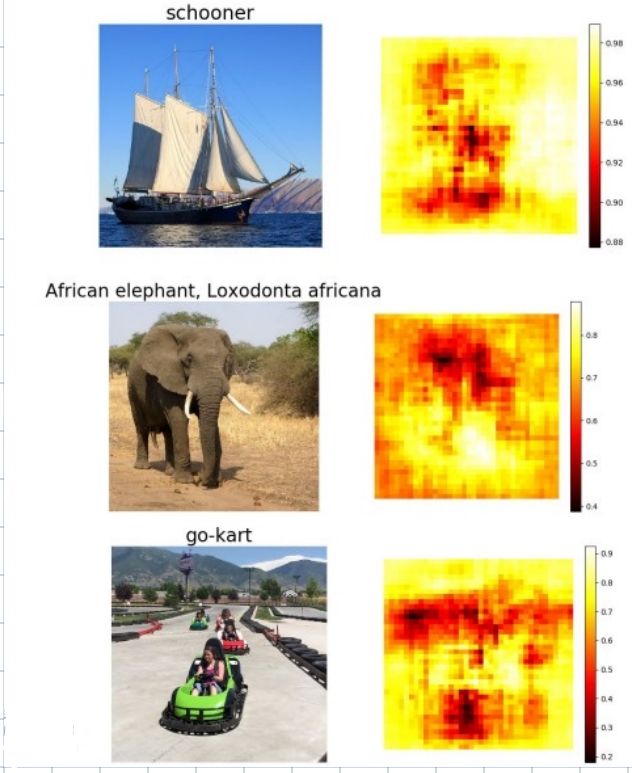
↳ This gives us a sense of what type of features that nueron is looking for

Occulation experiment:

↳ Which parts of the input image leads the network classify it

1. Take an image
2. Block a region of the image
3. Replace it with mean pixel image
4. Go through the network
5. Go to step 2

↳ If a blocked patch of the image caused the image to change drastically then that patch was really imp for the classification



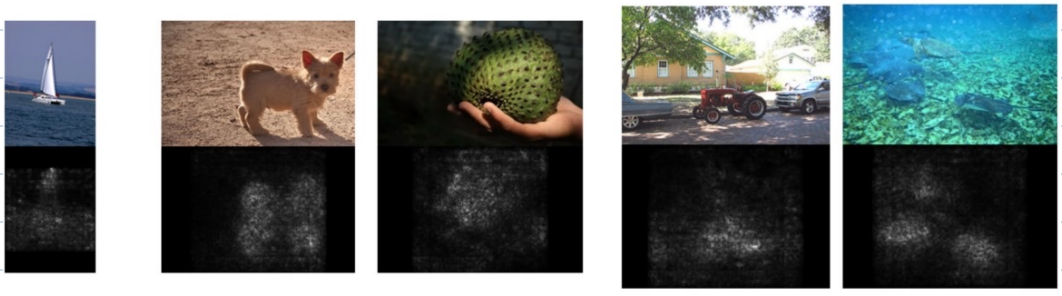
high prob score corresponding to a blocked patch

low prob score corresponding to a blocked patch

Saliency Maps:

↳ How to tell which pixel matters for classification?

↳ Take grad of predicted class score w.r.t the pixels of input image



Saliency maps for semantic segmention:

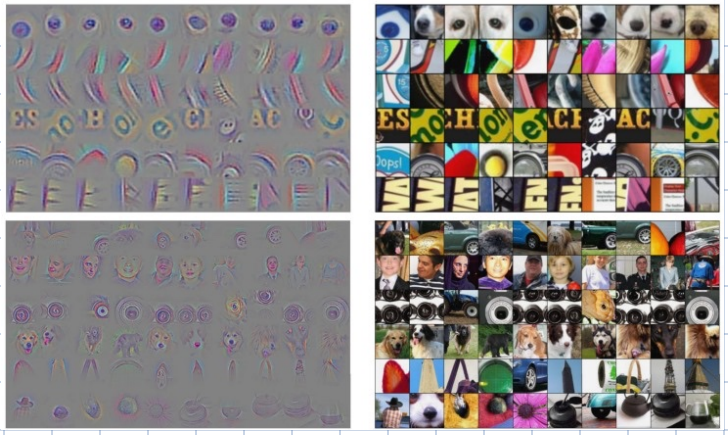
↳ Grabcut segmentation algorithm + saliency map ends up segmenting the image

↳ NOT very reliable

Intermediate features via (guided) backprop

↳ Take grad of nueron w.r.t the pixels of input image

↳ Images come out nicer if you only backprop only through tev grads thraigh each ReLU



Gradient Ascent:

↳ Weights remain same

↳ Change pixels to maximize the nueron activations

$I^* = \arg\max_I f(I) + R(I)$

↳ Natural image regularizer

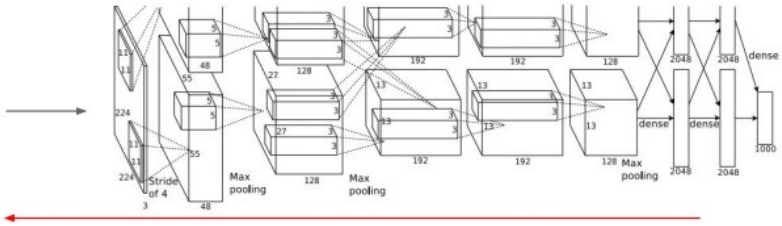
Neuron value

L2 norm of gen image

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

score for class c (before Softmax)

1. Initialize image to zeros



Repeat:
2. Forward image to compute current scores
3. Backprop to get gradient of neuron value with respect to image pixels
4. Make a small update to the image

↳ Instead of L2 norm, we add

↳ haussian blur image
↳ Clip pixels with small values to 0
↳ Clip pixels with small gradients to 0

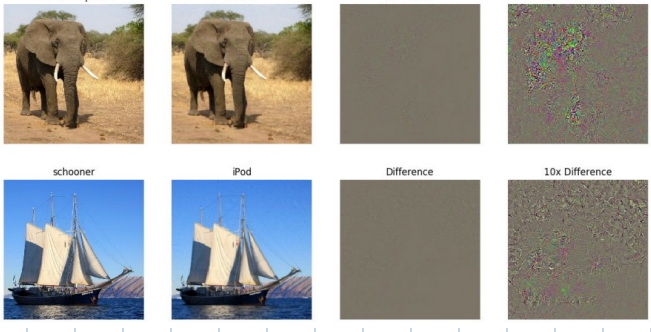
Another way:

↳ Maximize FC6 representation of an image rather than score

Fooling Images/Adversial Examples

↳ Start from an asbitrary images → Elephant image
↳ Pick an asbitrary image → koala bear
↳ Modify the image to maximize the class
↳ Repeat until the image is fooled

↳ Elephant would morph into koala bear but it does not?



DeepDream:

Choose an image & a layer in CNN:

↳ Foward: Compute activations at chosen layer
↳ Set gradient of chosen layer equal to it's activation
↳ Backward: Compute gradient on image
↳ Update image

↳ Amplify existing features

Feature inversion:

↳ Run image through the net
↳ Record feature vector of that image
↳ Recreate image using that feature representation

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

Given feature vector

Features of new image

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

→ minimize the dist b/w gen feature & saved feature

$$\mathcal{R}_{V^\beta}(\mathbf{x}) = \sum_{i,j} \left((x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2 \right)^{\frac{\beta}{2}}$$

Total Variation regularizer
(encourages spatial smoothness)

Texture synthesis:

↳ Given a small patch of some texture, generate larger image of same texture.

Generate pixels one at a time in scanline order; form neighbourhood of already generated pixels and copy nearest neighbour from input

↳ Works well for simple patterns

Neural texture Synthesis:

↳ Pass image through the network
↳ Take the activation volume
↳ Take two C-dimensional vectors
↳ Take outer product will give CxC

↳ Do for all
↳ Avg over HxW & get gram matrix

↳ Captures co-occurrence. i.e i & j is large, the ij will be larger

↳ Recreate the Gram matrix

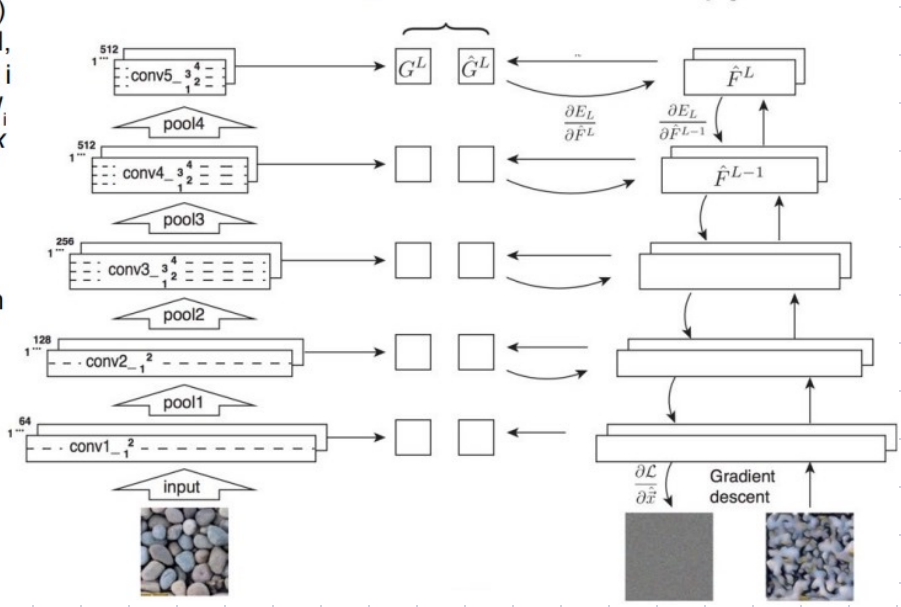
Neural Texture Synthesis

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - \hat{G}_{ij}^l\right)^2 \qquad \mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^{L} w_l E_l$$

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the *Gram matrix* giving outer product of features:

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l \text{ (shape } C_i \times C_i\text{)}$$

4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5



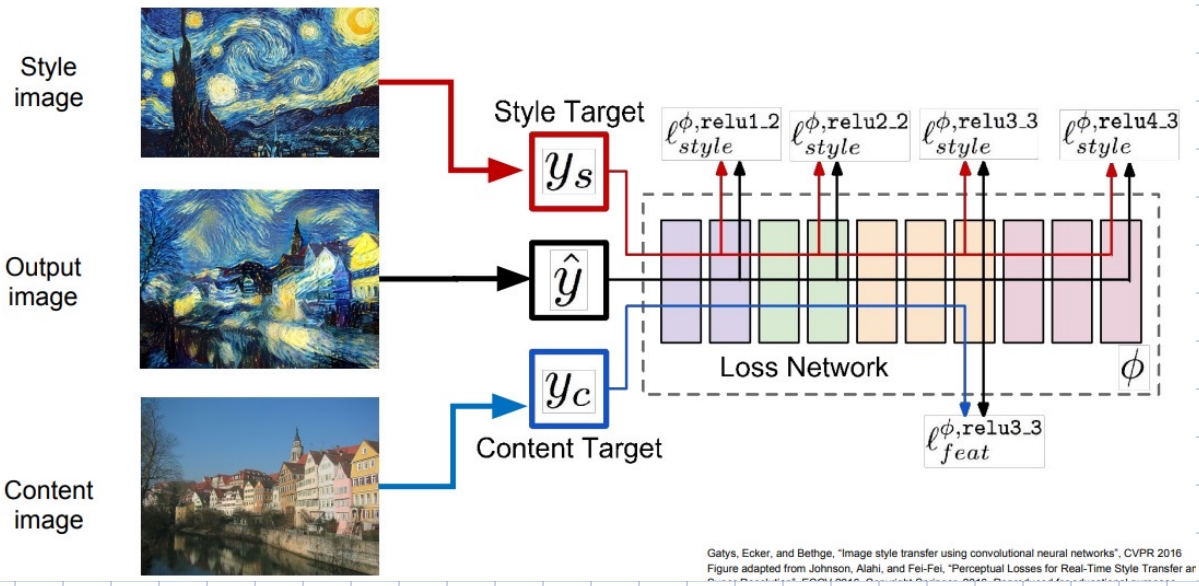
Style transfer:

Content image + Style image = style transfer

↳ What output will look like

↳ Texture

↳ Minimize feature represent & Gram matrix loss



Gatys, Ecker, and Bethge, "Image style transfer using convolutional neural networks", CVPR 2016
Figure adapted from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer ar