Explainability & Vulnerability

Explainability

Complex topic with many different approaches.

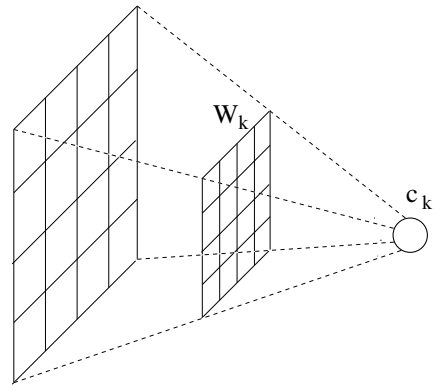
We shall merely focus on a major technique, based on **gredient ascent on input**.

The technique can be used to

- understand how CNNs "see the world"
- visualize portions of the input causing specific activations
- inspect the internal representation
- inject "style"
- explore adversarial attacks

Visualization of hidden layers

Goal: find a way to visualize the kind of patterns a specific neuron gets activated by.



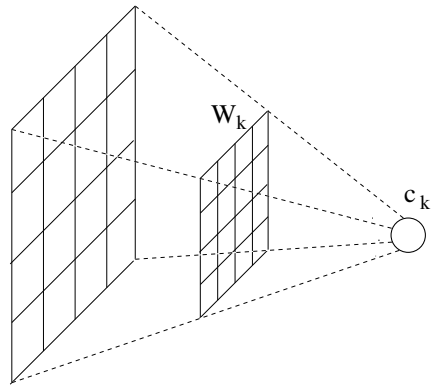
The loss function $\mathcal{L}(\theta, x)$ of a NN depends on the parameters θ and the input x.

During training, we fix x and compute the partial derivative of $\mathcal{L}(\theta, x)$ w.r.t the parameters θ to adjust them in order to decrease the loss.

In the same way, we can fix θ and use partial derivatives w.r.t. input pixels in order to syntehsize images minimizing the loss.

Visualization of hidden layers

Goal: find a way to visualize the kind of patterns a specific neuron gets activated by.



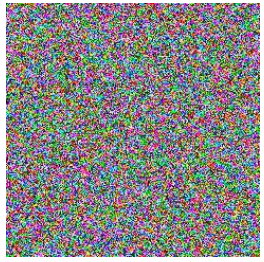
The loss function $\mathcal{L}(\theta, x)$ of a NN depends on the parameters θ and the input x.

During training, we fix x and compute the partial derivative of $\mathcal{L}(\theta, x)$ w.r.t the parameters θ to adjust them in order to decrease the loss.

In the same way, we can fix θ and use partial derivatives w.r.t. input pixels in order to syntehsize images minimizing the loss.

The "gradient ascent" technique

Start with a random image, e.g.



- do a forward pass using this image x as input to the network to compute the activation $a_i(x)$ caused by x at some neuron (or at a whole layer)
- do a backward pass to compute the gradient of $\partial a_i(x)/\partial x$ of $a_i(x)$ with respect to each pixel of the input image
- modify the image adding a small percentage of the gradient $\partial a_i(x)/\partial x$ and repeat the process until we get a sufficiently high activation of the neuron

First layers of Alex Net

Visualizations for the first two layers of AlexNet (Understanding Neural Networks Through Deep Visualization by A.Nguyen et al., 2015)

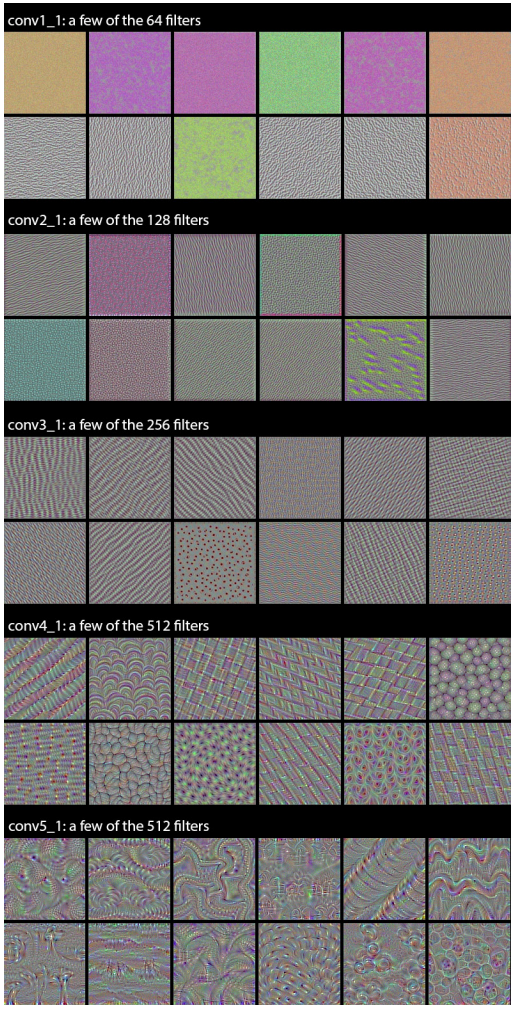
In the following pictures the size of the image is equal to the receptive field of the neurons of the given layer.

Since generation is non deterministic, 4 samples are provided for each feature map.

First maps are very simple:

At second layer they are already much more complex:

Visualizations for VGG and ResNet



An exploration of convnet filter with keras

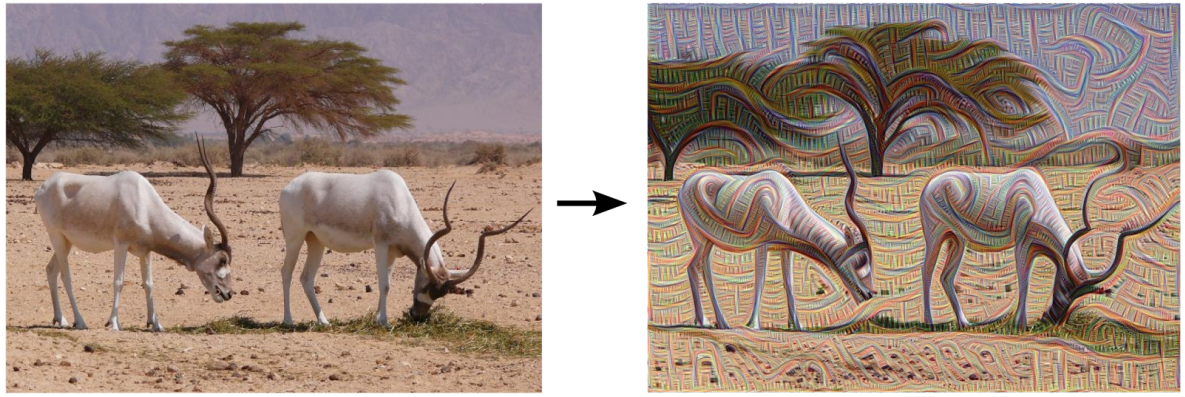
For this visualization images of a fixed size (e.g. 180) are genrated.

The "pattern" recognized by the kernel is the pattern repeated inside the generated image.

Going down towards deeper layer the size and complexity of patterns augment and their frequency decrease.

Enhancing content

Instead of prescribing which feature we want to amplify, we can also fix a layer and enhance whatever it detected.



Each layer of the network deals with features at a different level of abstraction. This was deeply explored in a technique called **inceptionism**.

Lower layers will produce strokes or simple ornament-like patterns, because those layers are sensitive to basic features such as edges and their orientations.

A different usage

What caused the activation of this neuron **in this image**?

Emphasize what matters

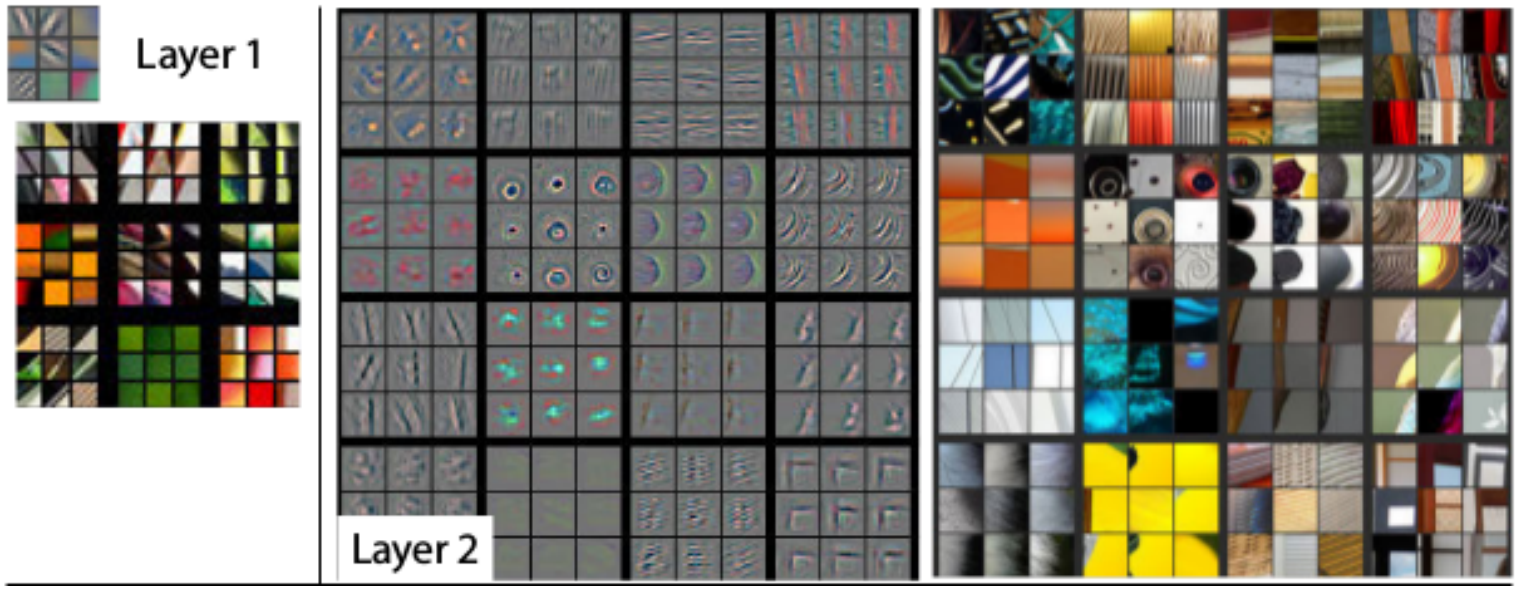
Instead of syntesizing an image maximing the activation of a neuron, emphasize in real images what caused a specific activation.

Visualizing and Understanding Convolutional Networks Matthew D Zeiler, Rob Fergus (2013)

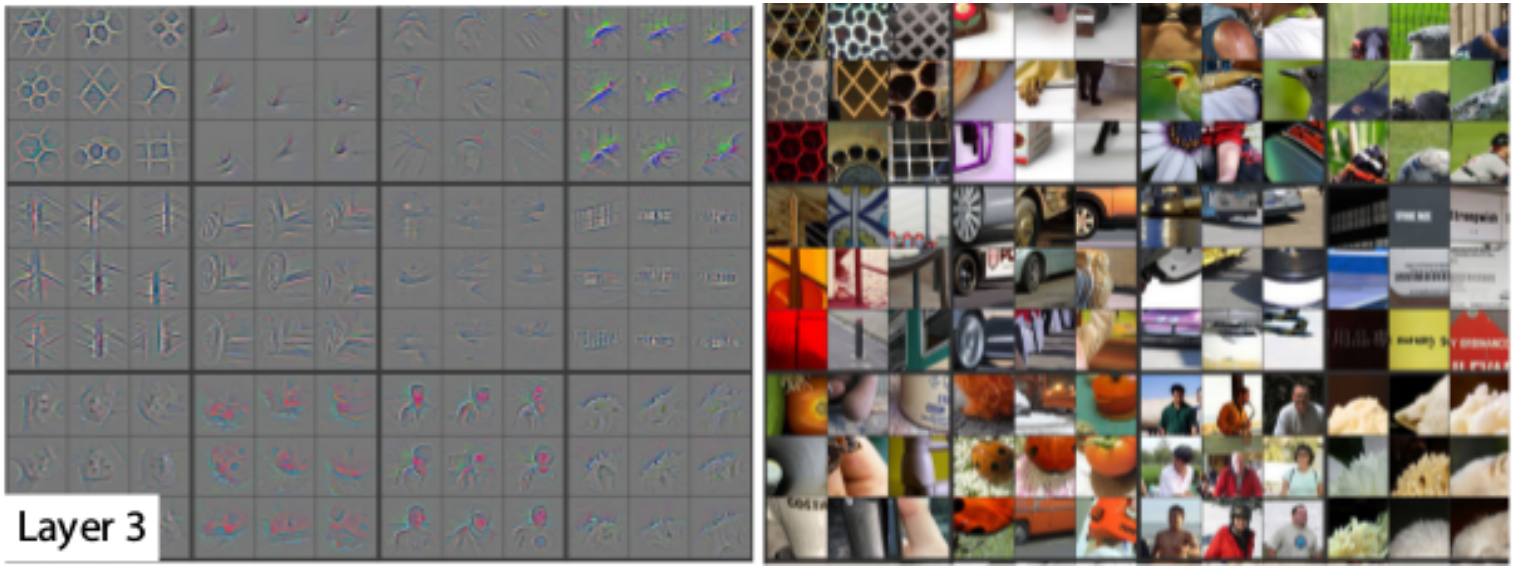
results - layers 1 and 2

For each layer we see:

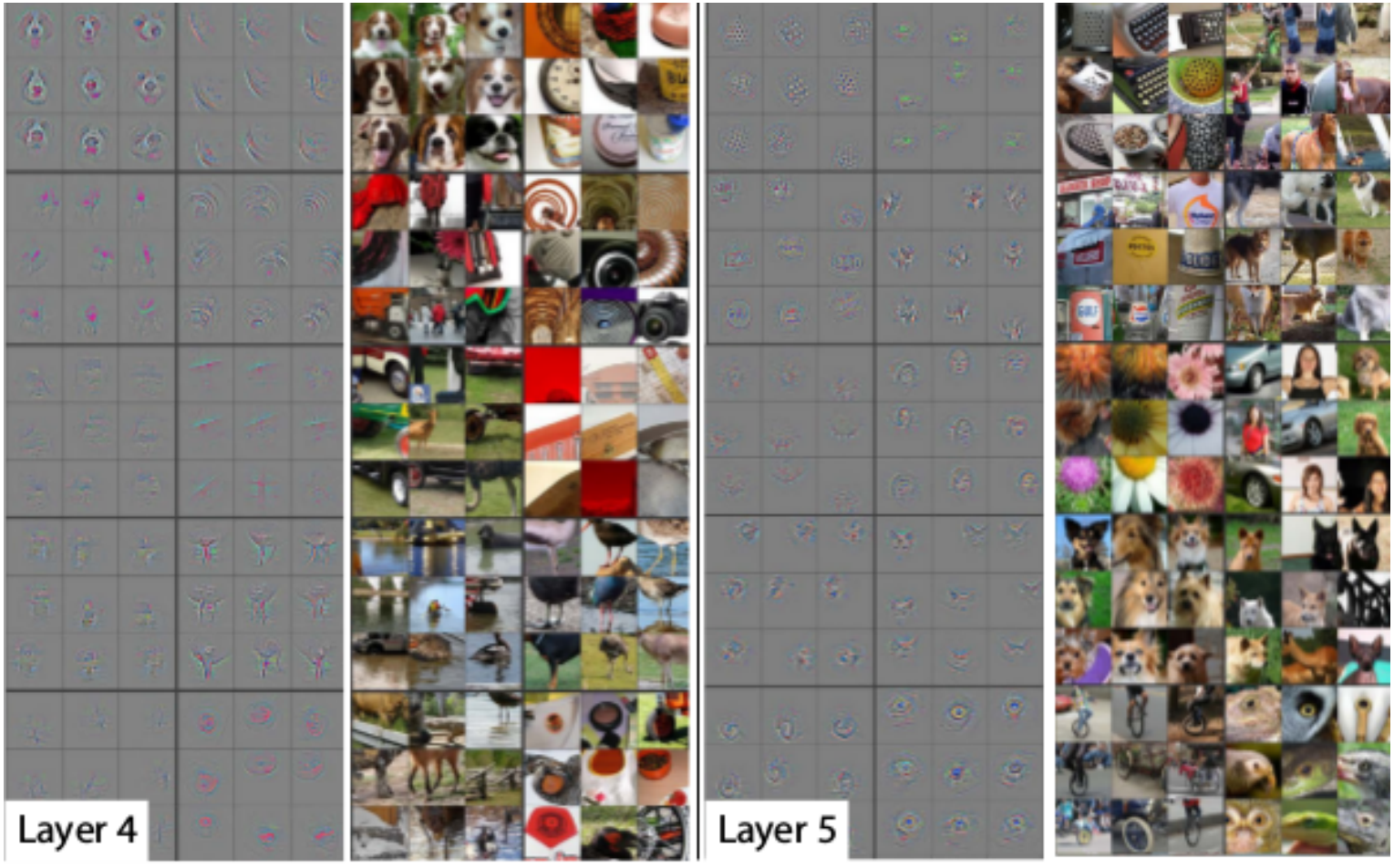
- 9 recostructed patterns relative to strong activations for a subset of feature maps of a given layer
- corresponding image patches



results - layer 3



results - layers 4 and 5



Discussion

Moving towards higher levels we observe

- growing structural complexity:
  oriented lines, colors $\rightarrow$ angles, arcs $\rightarrow$ textures
- more semantical grouping
- greater invariance to scale and rotation

Still another approach

Try to understand the inner representation at some layer by generating an image indistinguishable from the original one.

"DeConv nets look at how certain network outputs are obtained, whereas we look for what information is preserved by the network output – Mahendran, Vevaldi 2014"

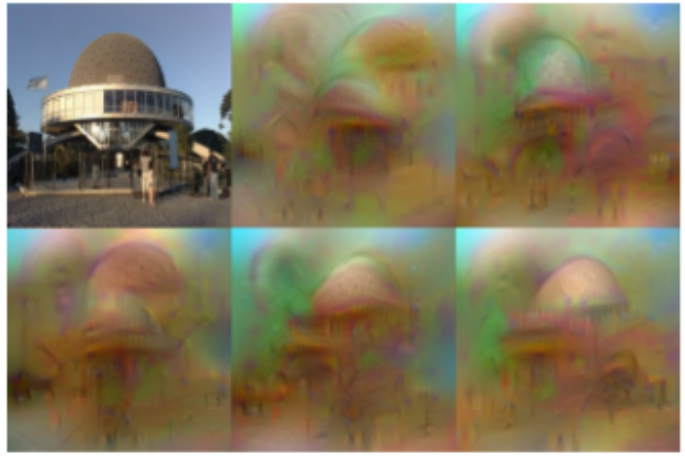


Figure 1. **What is encoded by a CNN?** The figure shows five possible reconstructions of the reference image obtained from the 1,000-dimensional code extracted at the penultimate layer of a reference CNN[13] (before the softmax is applied) trained on the ImageNet data. From the viewpoint of the model, all these images are practically equivalent. This image is best viewed in color/screen.

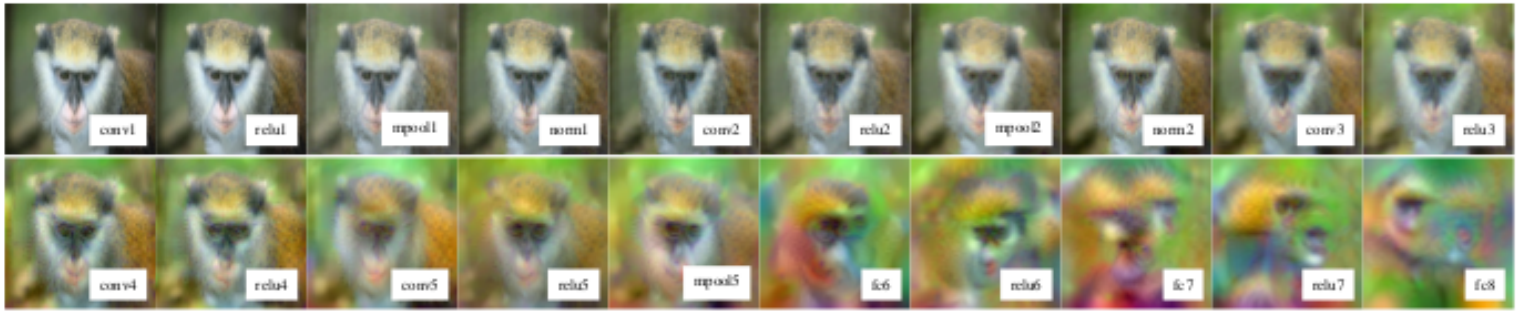
Understanding Deep Image Representations by Inverting Them
A. Mahendran, A. Vedaldi (2014)

Comparison to other works

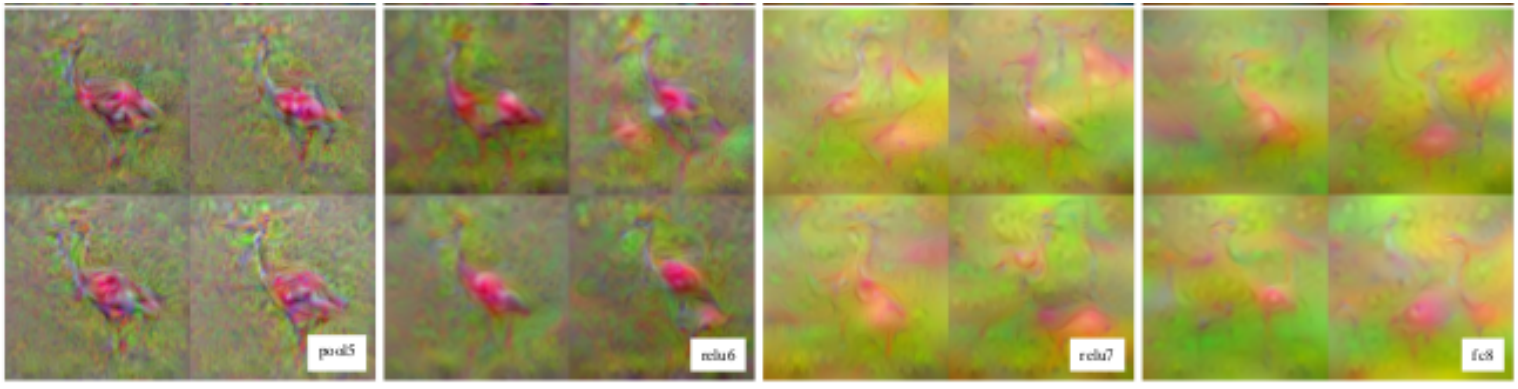
- objective similar to autoencoding, but
  1. it works with an arbitrary network (typically, a classifier), not an autoencoder
  2. we are not training the network: backpropagation is used to reconstruct the image
  3. loss is measured on the internal representation
- we progressively adjust the source image by gradient ascent, but instead of optimizing towards a given category, the goal is to minimize the distance from an internal encoding $\Theta_0 = \Theta(x_0)$ of a given image x_0

$$\underset{x \in R^{H \times W \times C}}{\operatorname{argmin}} \underbrace{\ell(\Theta(x), \Theta_0)}_{\textit{loss}} + \underbrace{\lambda \mathcal{R}(x)}_{\textit{regularizer}}$$

Results



First layers are an invertible code, progressively becoming fuzzier.



Last layers invert back to multiple copies of object's parts at different positions and scales.

These details must be relevant for discriminative classification.

A recent application: inverting CLIP

What do we learn from inverting CLIP models?

Embed a text and reconstruct an image through inversion.

Usually CLIP is used to "guide" generation of generative models (either at training or sampling time).

"Gradient ascent" gives a more direct grasp of CLIP's internal representation, not mediated by the generator.

Injecting Style

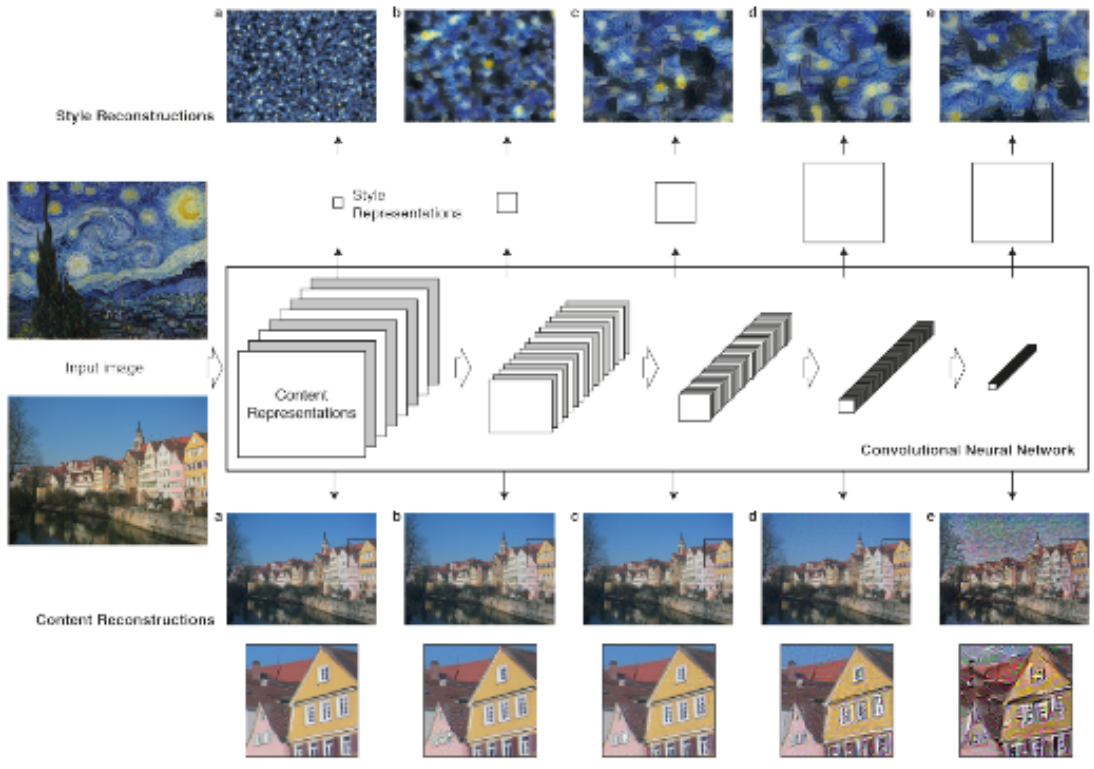
Mimicking artistic style

Gradient ascent was on of the first techniques used to mimic (incept) artistic style inside images

A neural algorithm of artistic style L.A.Gatys, A.S.Ecker, M.Bethge

Capturing style

Add a feature space on top of the original CNN representations computing correlations between the different features maps (channels) at each given layer.
A technique already used to compute image textures.



The Gram Matrix

At layer ℓ, an image is encoded with distinct feature maps F_d^ℓ for $d \in D^\ell$, the Depth (number of channels) at ℓ.

The correlation between the feature maps is another feature that seems to capture the **texture** of the image.

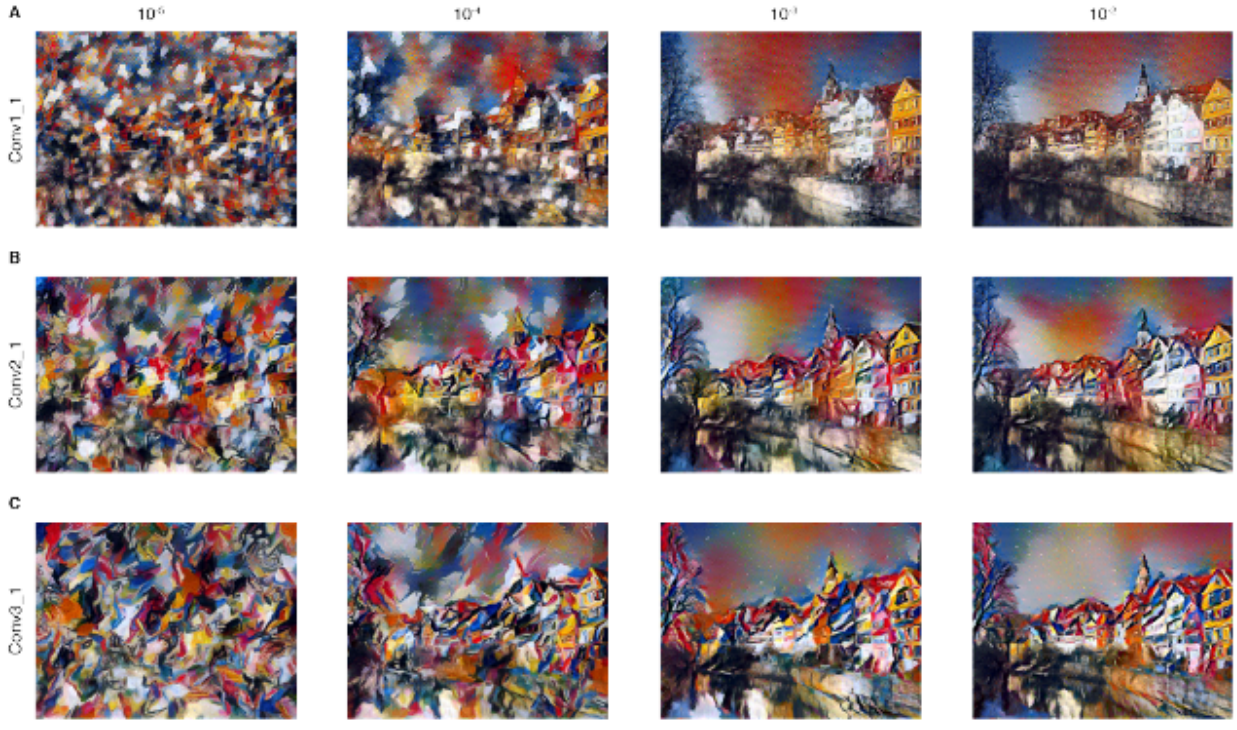
Feature correlations for the given image are given by the Gram matrix $G^\ell \in \mathcal{R}^{D^\ell \times D^\ell}$, where $G^\ell_{d_1,d_2}$ is the dot product between the feature maps $F^\ell_{d_1}$ and $F^\ell_{d_2}$ at layer ℓ:

$$G^\ell_{d_1,d_2} = F^\ell_{d_1} \cdot F^\ell_{d_2} = \sum_k F^\ell_{d_1,k} \cdot F^\ell_{d_2,k}$$

The Gram Matrix is just yet another internal representation of the image, useful to encode style.

Combine style and content

Content and style are separable:



Different combinations varying the reconstrution layer (rows) and the relevance ratio between style and content (columns).

Other approaches to Style Tranfer

Many different techniques have been proposed over the year to address style transfer or to mimic style.

The modern approach looks at style transfer as a particular form of conditioning along image generation.

Vulnerability

bus

ostrich

Fooling neural networks

We already know that we may use gradient ascent to to modify an input image to emphasize the activation of a given neuron.

In an image classification framework, we can use this technique to increase the score of whatever class we want.

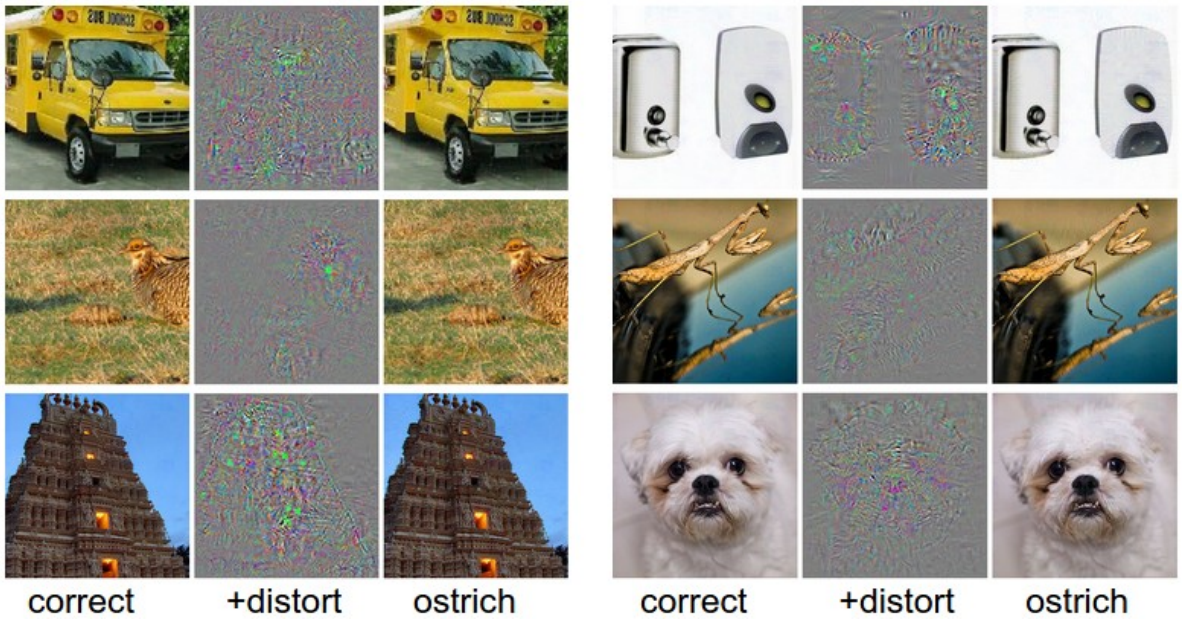
So, in principle, we should be able to automatically syntesize images of any of the categories under consideration.

Let us try this

Demo!

Fooling Neural Networks

Since we have many pixels, a tiny (imperceptible to humans!), consistent perturbation of all of them is enough to fool the classifier.



Intriguing Properties of Neural Networks, C. Szegedy et al., 2013

A different technique

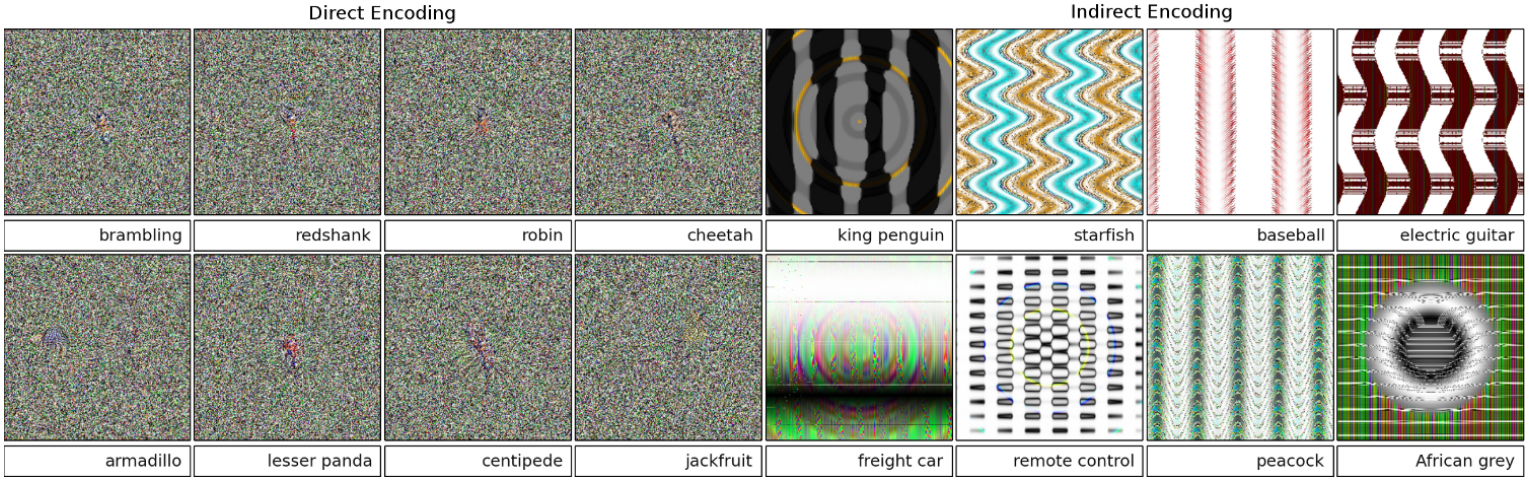
The previous technique, being based on gradient ascent, requires the knowledge of the neural net in order to fool it.

It is possible to obtain a similar result using the network as a black box?

DNNs are easily fooled

In "Deep Neural Networks are easily fooled", Nguyen et al. proved that it was easy to produce adversarial examples using an evolutionary technique

In particular, they were able to produce not only "noisy" adversarial images, but also geometrical examples with high regularities (images meaningful for humans).



Evolutionary approach

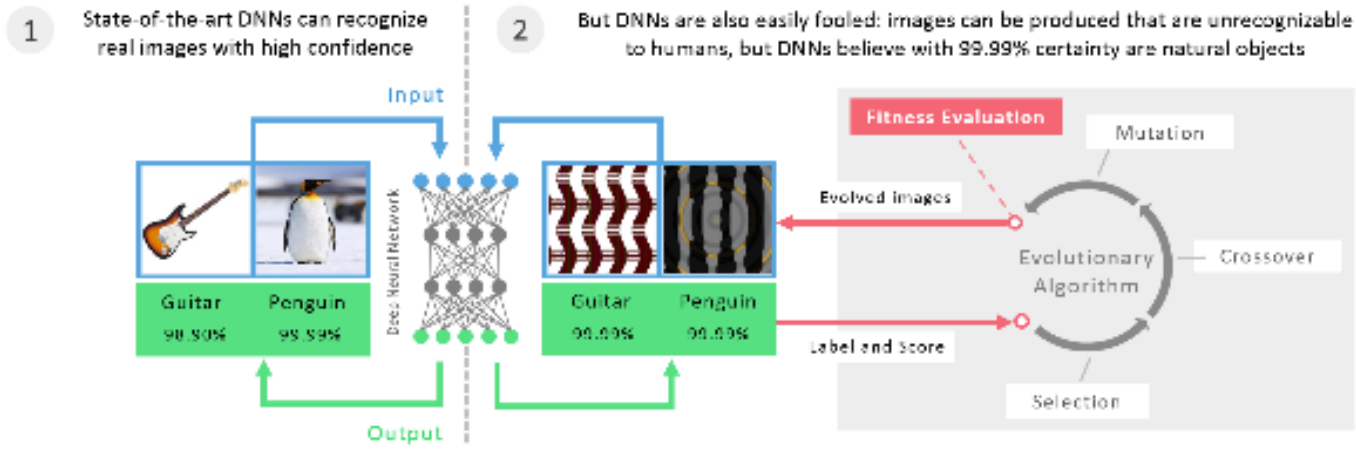
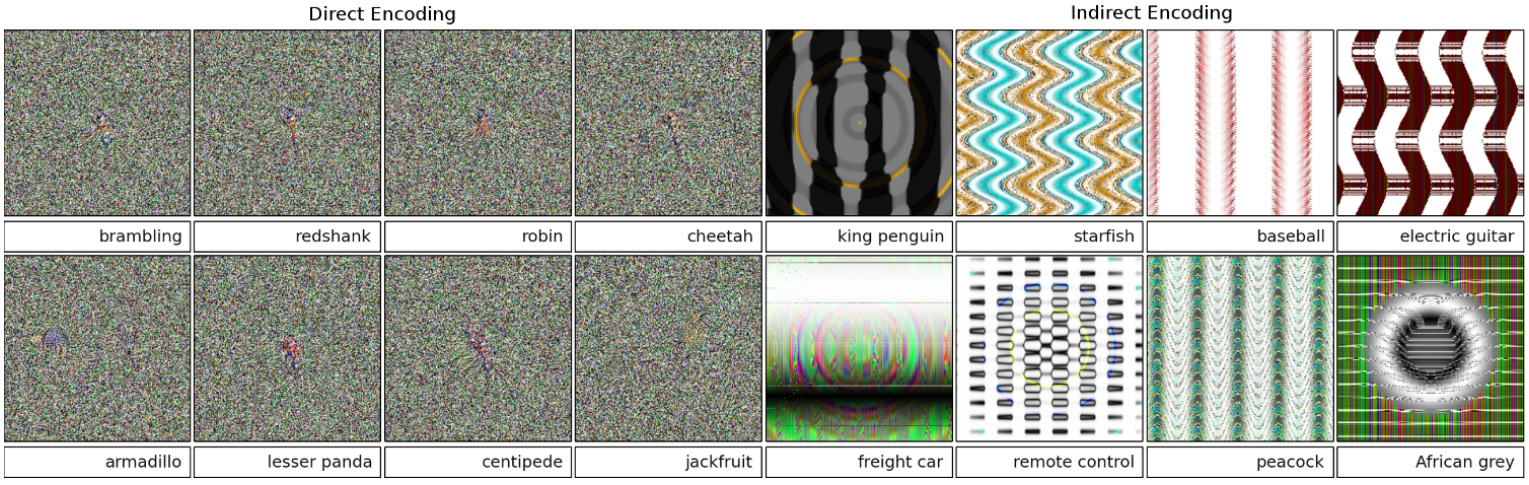


Figure 2. Although state-of-the-art deep neural networks can increasingly recognize natural images (*left panel*), they also are easily fooled into declaring with near-certainty that unrecognizable images are familiar objects (*center*). Images that fool DNNs are produced by evolutionary algorithms (*right panel*) that optimize images to generate high-confidence DNN predictions for each class in the dataset the DNN is trained on (here, ImageNet).

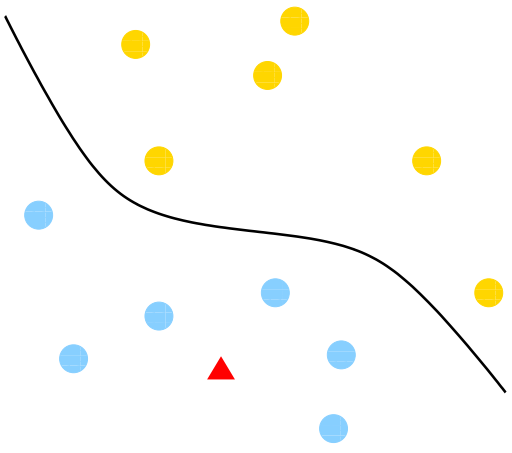
- start with a random population of images
- alternately apply selection (keep best) and mutation (random perturbation/crossover)

Image encoding



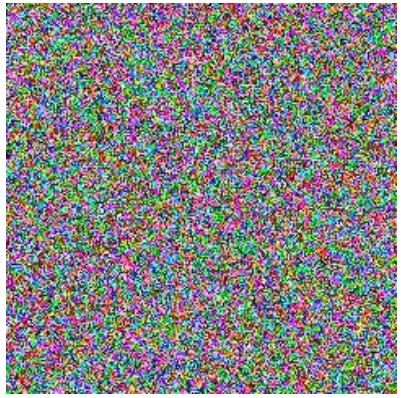
- **direct encoding**: explicit representation of the image as an array of pixels
- **indirect encoding**: implicit representation of the image as a composition of regular functions: sine, gaussian, sigmoid, linear, etc.

Why classification techniques are vulnerable



Dimensionality of data vs their feature space

If we generate an image at random, it looks like noise.
The probability of randomly generating an image having "a meaning" **is null**.



This means that "natural images" occupy a portion of the features space of almost **no dimension**.

Moreover, due their regularities, we expect them to be organized along some **disciplined, smooth-runnning** regions.

This is the so called **manifold** of data (in the features space).

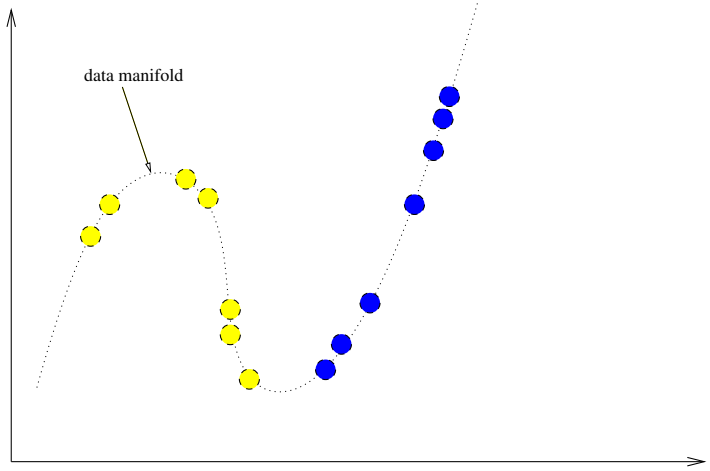
Data Manifold

Manifold: a collection of points forming a certain kind of set, such as those of a topologically closed surface

- Natural images have manifolds with low dimensionality (that is why we can compress them)
- in high dimensional space, easy to move away from the manifold of a given category (with modifications imperceptible to the human eye)

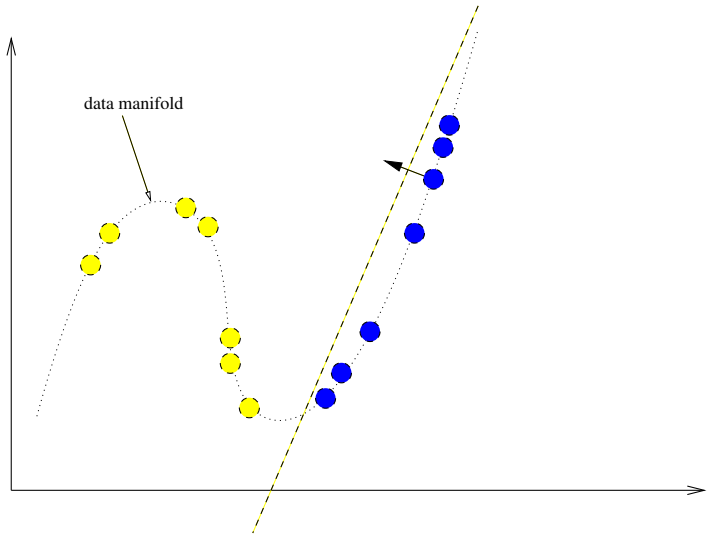
The Manifold issue

Suppose we have a space with two features, and our data occupy the manyfold of dimension 1, along the dotted line described in the following picture.



Suppose moreover that our data are splitted in two categories (yellow and blue) and we want to perform their classification

The Manifold issue



We have little knowledge of where the classifier will draw the boundary.
A tiny change in the data features may easily result in a miss-classification.

Now imagine the possibilities in a space with hundreads or thousands of dimensions.

Observe that we are **escaping** from the actual data manifold.

A remark on inversions

When we tried to transform an elephant into a tiger shark via gradient ascent we failed.

In some other cases, e.g, inverting CLIP, we obtained acceptable results.

- synthesis through gradient ascent is always difficult
- synthesis from discriminative features is particularly complex.
  - good inversion from first layers of a CNNs for image classification (general features, little loss of information)
  - bad inversion from categories: difficult to syntesize a "cat"
- moving in the direction of gradient we easily quit the data manifold.

A remark on inversion

The complexity of inversion through gradient descent (provide to start from informative features) consists in modifying an image **remaining inside the expected data manifold**.

This is difficult, since we have little knowldge about the actual data distribution in the feature space.

To this aim, deepdream generator exploits **regularization** techniques (smoothing, texture similarities, etc.) trying to obtain images similar (in statistical terms) to those in the training set.