Lesson 6

- adversarial attacks
- the data manifold
- autoencoders

How to fool a Neural Networks

Reducing distance from a target category

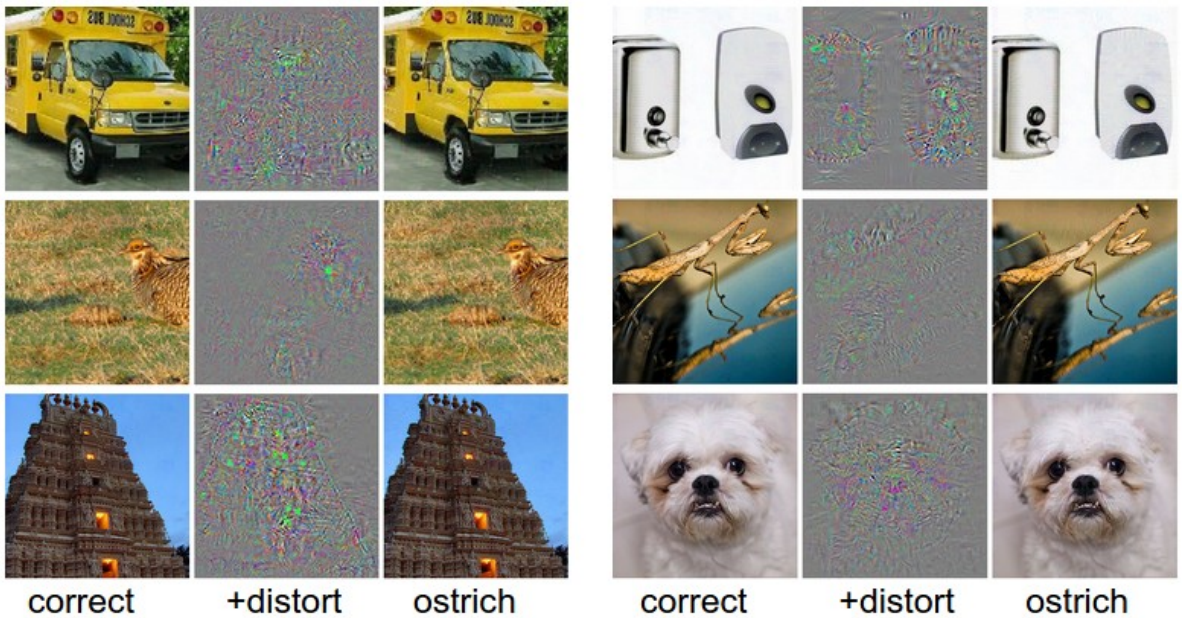
In an image classification framework, we can use the **gradient ascent technique** to increase, starting from noise or any given picture, the score of whatever class we want.

Demo!

We shall try to transform an elephant into a tigershark.

Fooling Neural Networks

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



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

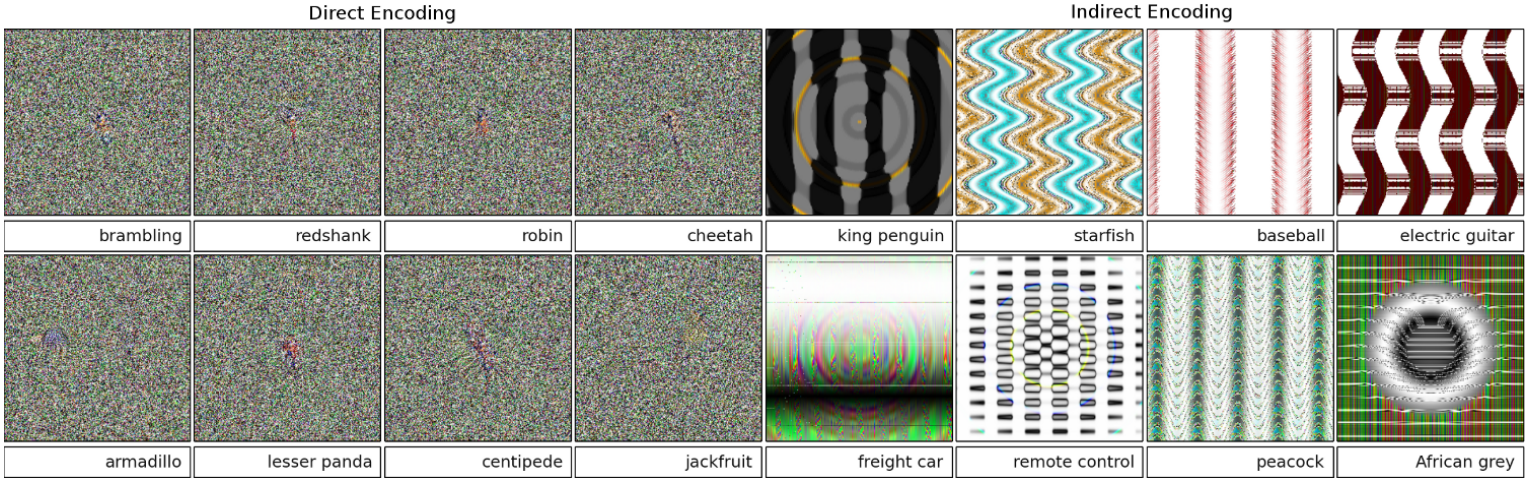
A different technique

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

We can do something similar using the network as a **black box**, for instance by means of evolutionary techniques

- Nguyen A, Yosinski J, Clune J. Deep Neural Networks are Easily Fooled. In Computer Vision and Pattern Recognition (CVPR '15), IEEE, 2015.

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



Evolutionary approach

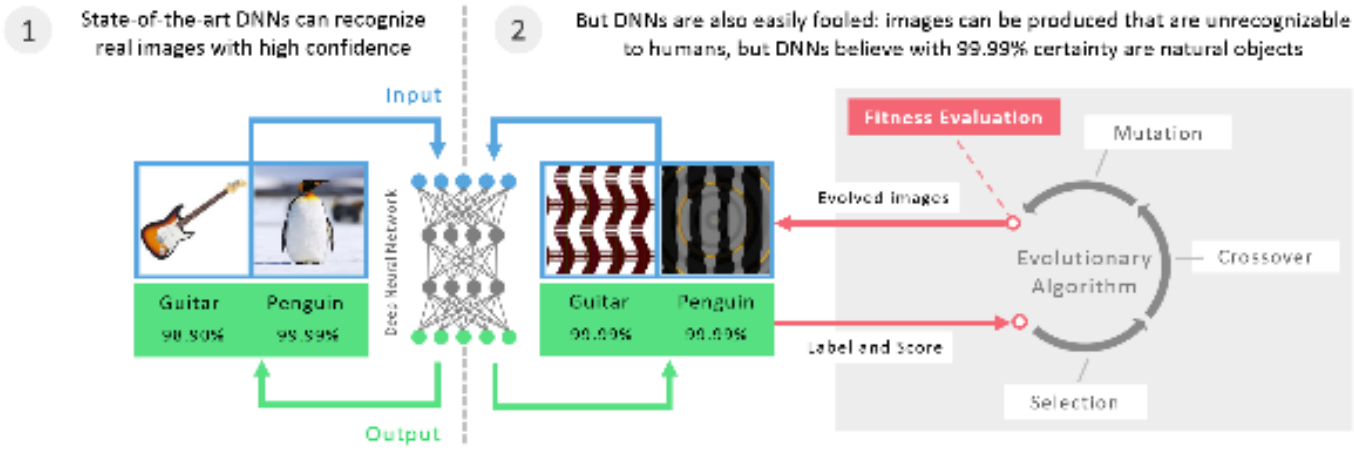
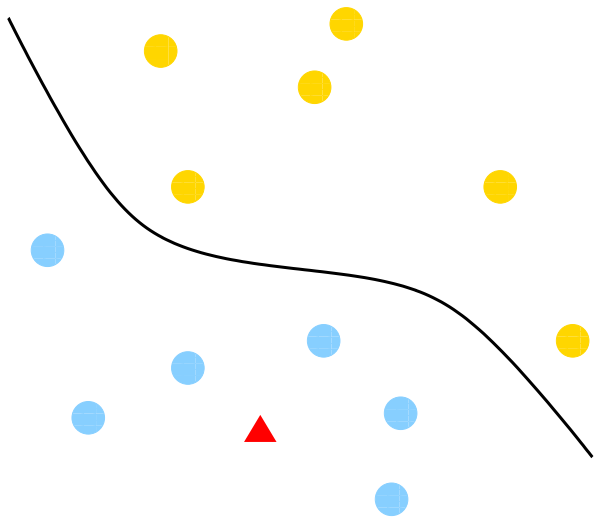


Figure 2. Although state-of-the-art deep neural networks can increasingly recognize natural images (*left panel*), they also are easily fooled into declaring with near-certainty that unrecognizable images are familiar objects (*center*). Images that fool DNNs are produced by evolutionary algorithms (*right 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)

Why classification techniques are vulnerable



Why classification techniques are vulnerable

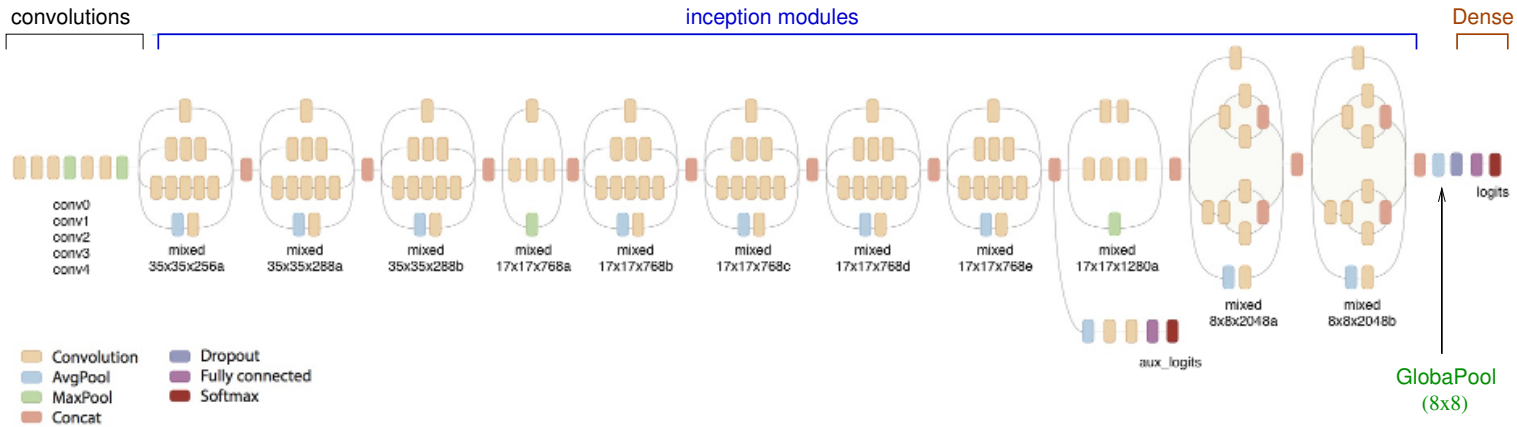
- **discriminating** between domains does not give us much knowledge about those domains.

  Difference between **generative** and **discriminative** approaches in machine learning.
- if data occupy a **low-dimensional** portion in the feature space, it is easy to modify features, to pass the borders of our discrimination boundaries.

In the next slides we try to better understand these concepts.

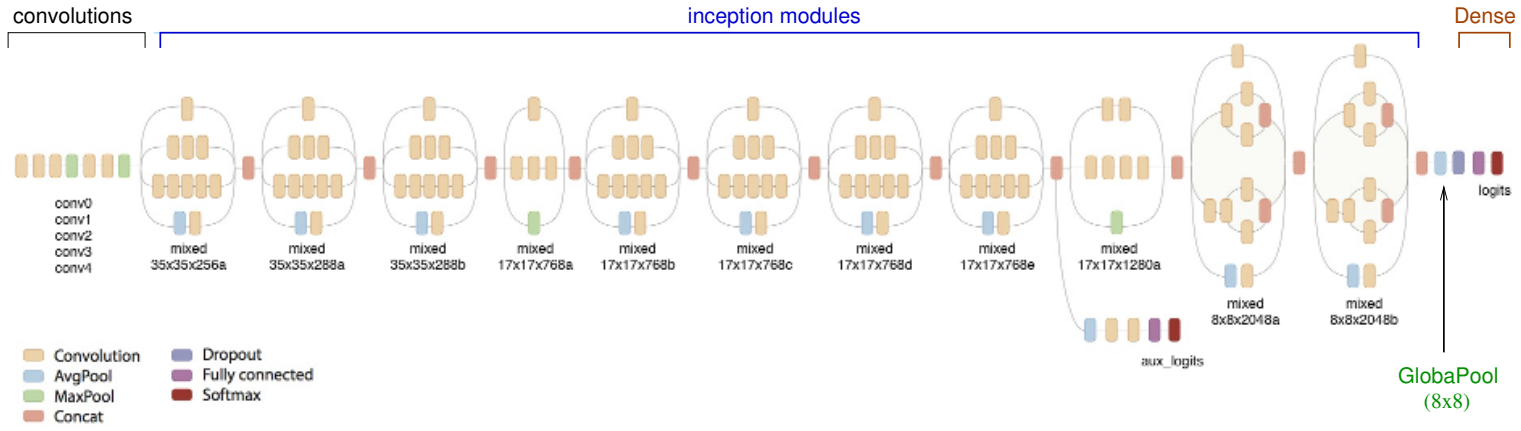
NN for image processing

A Neural Network for image processing has a structure of the following kind (this is Inception V3):



- a **long** sequence of convolutional layers, possibly organized in suitable modules (e.g. inception modules)
- a **short** (2 or 3) final sequence of dense layers

Feature extraction and exploitation



- with convolutional layers we **extract** interesting features from the input image, generating a different internal representation of data in terms of these features
- with the dense layers, we **exploit** these features in view of the particular problem we are aiming to solve (e.g. classification).

Reversing the representation

Reversing the representation of data makes sense as far as we are in extraction phase

- **we can syntesize interesting patterns** recognized by neurons of convolutional layers

We cannot expect to derive interesting information about categories from the information that the network uses to discriminate among them.

- **we cannot automatically synthesize a "cat"** (starting from a classifier; we shall see specific generative techniques in the next lesson)

Fooling networks

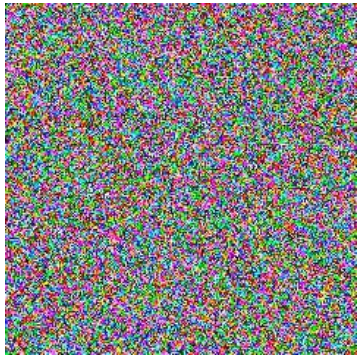
The previous point explains why we should not expect to be able to syntesize images of high level categories starting from a classifier.

But it does not explain why an almost imperceptible modification of an image is enough to fool the classifier.

To answer to this question we must discuss the dimensionality of the data manifold.

Dimensionality of data and feature space

If we generate an image at random, it looks like noise.
The probability of randomly generating an image having some sense for us **is null**.



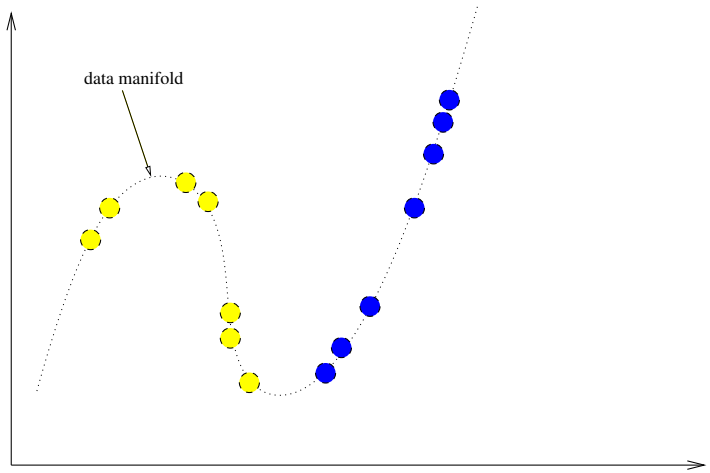
This means that "natural images" occupy a portion of the features space of almost **no dimension**.

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

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

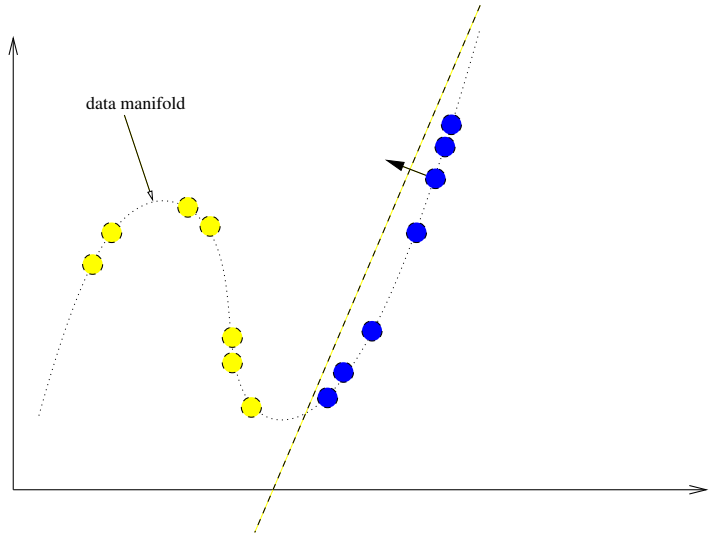
The Manifold issue

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.

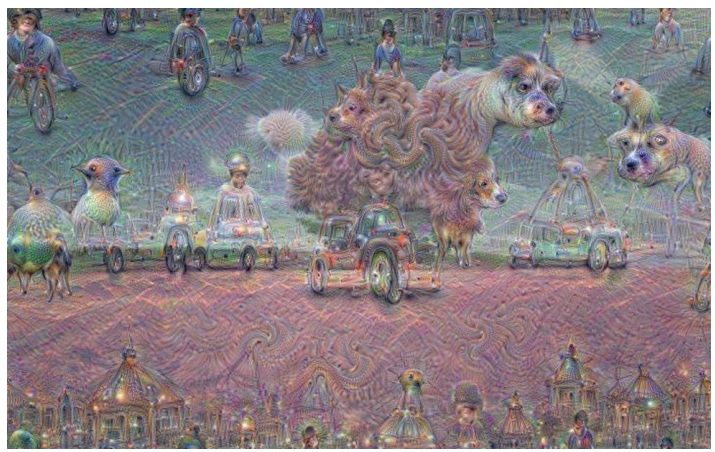
Observe that we are **escaping** from the actual data manifold.

A remark on inceptionism

The complexity of inceptionims 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.



Autoencoders

Introduction

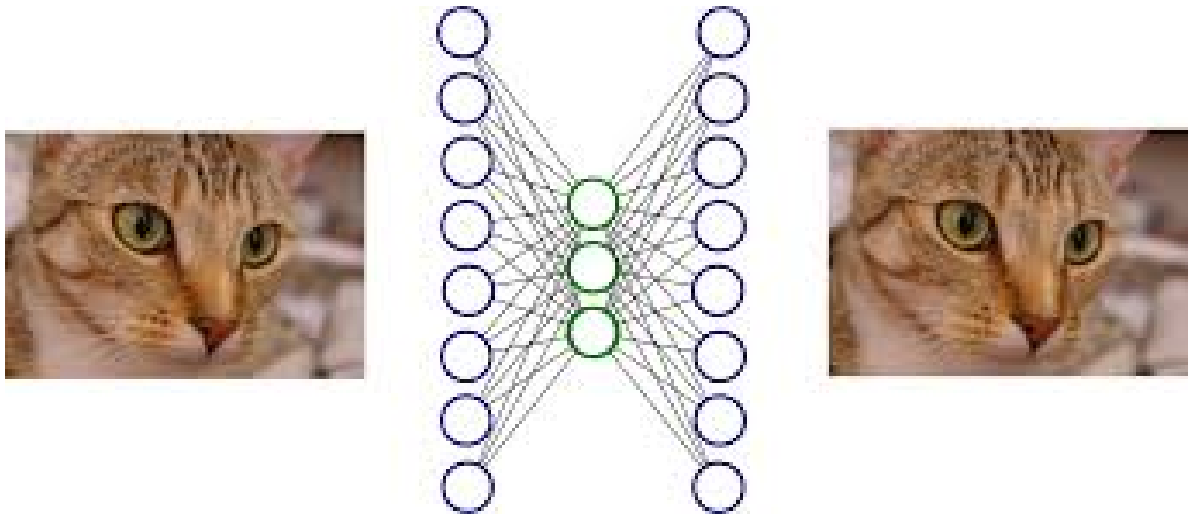
Two natural questions about the data manifold:

1. We said that (in frequent cases) the actual dimensionality of the data manifold is low in comparison with the latent space.
   Can we experimentally confirm this claim?
   Can we **compress** data?
2. For fooling networks, we synthesized new samples **outside** the actual data manifold.
   Is it possible to **automatically detect** this kind of **anomalies**?

To answer to these kind of questions it is worth to have a look at particular neural network models called **autoencoders**.

Autoencoders

An autoencoder is a net trained to reconstruct input data out of a learned internal representation



Usually, the internal representation has lower dimensionallity w.r.t. the input

Compression

Why is data compression possible, in general?

Because we exploit **regularities** (correlations) in the features describing input data.

If the input has a random structure (high **entropy**) no compression is possible

- random, lawless, uncompressible, high entropy
- ordered, lawfull, compressible, low entropy

Compression

Why is data compression possible, in general?

Because we exploit **regularities** (correlations) in the features describing input data.

If the input has a random structure (high **entropy**) no compression is possible

- random, lawless, uncompressible, high entropy
- ordered, lawfull, compressible, low entropy

A form of data compression

If (as usual) the internal layer has fewer units of the input, autoencoders can be seen as a form data compression. The compression is

- **data-specific**: it only works well on data with strong correlations (e.g. digits, faces, etc.) This is different from traditional data compression algorithms
- **lossy**: the output is degraded with respect to the input. This is different from textual compression algorithms, such as gzip
- **directly trained** on unlabeled data samples. We usually talk of self-supervised training

What are they good for?

Non so good for data compression, due to the lossy nature.

Applications to

- data denoising
- anomaly detection
- feature extraction (generalization of PCA)
- generative models (VAE)

Especially, an amusing and simple to understand topic.

Demo: autoencoders in Keras

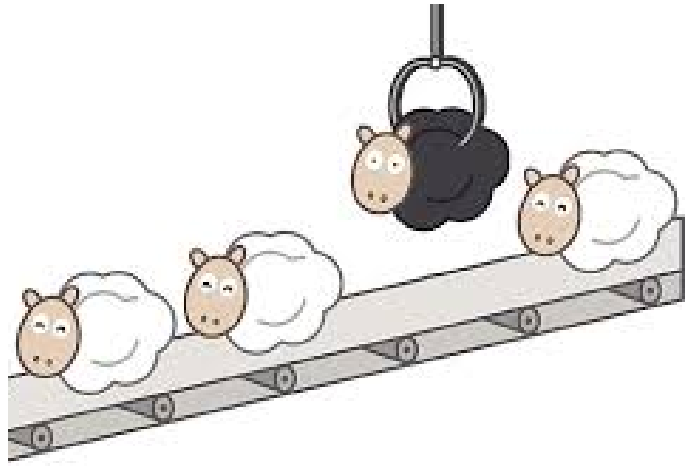
See Building autoencoders in Keras, on the Keras blog

Image denoising

Artificially add noise and train to reconstruct the original image

Denoising autoencoders inspired dropout

Anomaly detection



The latent encoding

The latent encoding of data is meant to capture the **main components** of the data features.

We can hence expect to **easily detect anomalies** by looking at points with **abnormal latent values**.

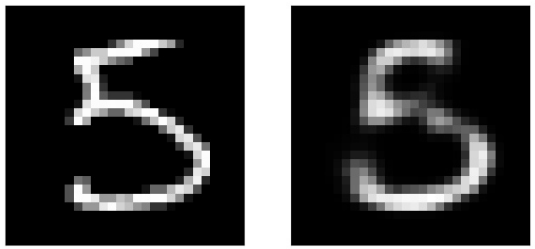
Equivalently, we may look at points with a **reconstruction below the expected quality**.

Anomaly detection

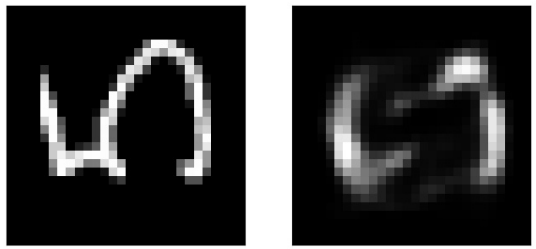
Autoencoding is **data specific**: the autoencoder works well on data similar to those it was trained on.

If applied on different data (anomaly), it will perform poorly.

Example on mnist data with the autoencoder of the previous demo.
mean loss = 0.105, std: 0.036



loss = 0.141 OK!



loss = 0.269 LIAR!

Suggested reading: Anomaly Detection with Autoencoders Made Easy