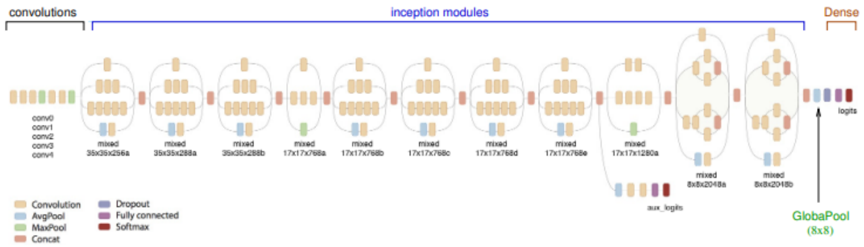
Data Manifold

Why classification techniques are vulnerable

Every single classification technique, not just NN, is vulnerable. - discriminating between domains does not give us much knowledge about those domains. Difference between *generative* and *discriminative* approaches in machine learning.
- In general, in *generative techniques* we try to discriminate by taking account how the data that I have is distributed and what features does it have. - Instead, the *discriminative techniques* we try to discriminate by only looking at those features of a sample which allow us to solve our task.
- if data occupies a low-dimensional portion in the feature space, it is easy to *modify features*, to pass the borders of our discrimination boundaries.

Let's consider this neural network, Inception V3, which is used for image process-



ing:

This kind of network has - 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 - 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).

What if we reverse the representation of this kind of data? 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)

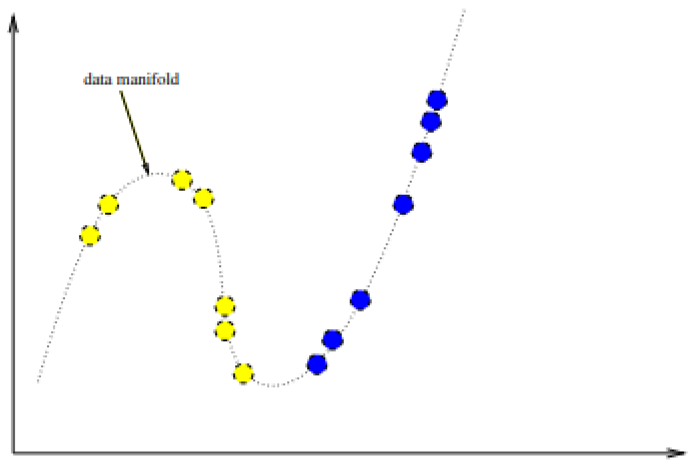
Data Manifold

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).

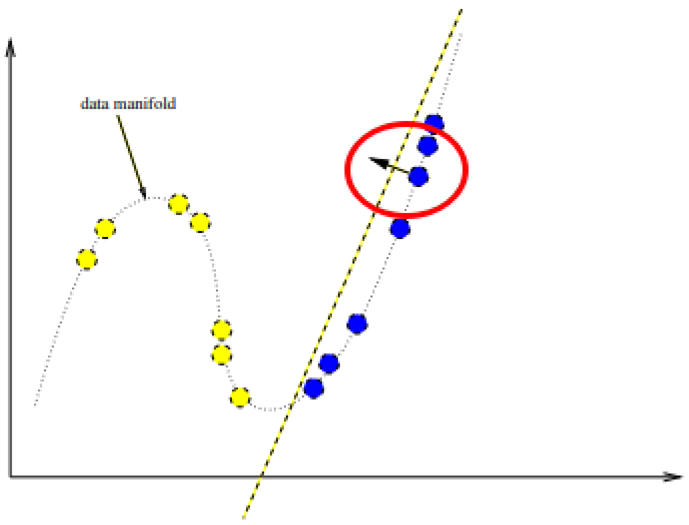
> [!DEFINITION] A manifold is a set of values in which the all neighbours are homeomorphic. - The data manifold is a continue -> if we interpolate 2 image that are not noise, we may get an image that is senseless, but that it still not noise at the very least.

The Manifold issue

- Suppose we have a space with *two features* (a cartesian plane), and our data occupy the *manifold of dimension* 1 (the line in the picture), 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



We have little knowledge of where the classifier will draw the boundary. A *tiny change* in the *data features* may easily result in a *missclassification*. Observe that we are escaping from the actual data manifold..



As we can see, it is very easy to move the circled sample *outside of the boundary*, such that it is misclassified.

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

So, essentially, when we are adding the noise that we saw in the previous lesson, we *are escaping the manifold*, and doing so in a way that allows us to go out of the boundary of our class.

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*.

A remark on manifold So, what if, when we have a classification problem, we introduce another class that essentially captures all the data that is not on the manifold (in Asperti's words: that is bullshit). Would something like this be feasible? - i.e. We have 3 classes: cats, dogs and other. In short: no, ==it would not be feasible==, since our *manifold* (which in this case is our dataset) is an extremely small set in the feature space. So, the “other” class would just be too huge. Thus, when building a classifier, we should only discriminate between the classes that we're interested in.

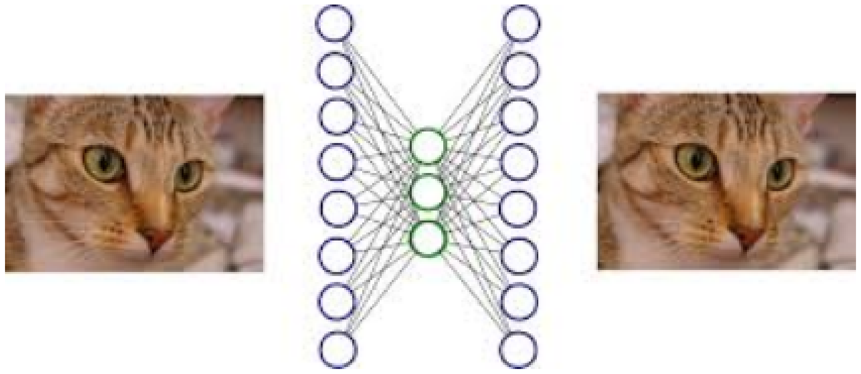
Autoencoders

Two natural questions about the data manifold: 1. We said that (in almost all 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**.

What is an autoencoder?

An *autoencoder* is a net trained to reconstruct input data out of a *learned internal representation.* Usually, the internal representation has ==lower dimension-



allity== w.r.t. the input.
Why is data compression possible, in general? Because we *exploit regularities* (correlations) in the *features* describing input data. Thus, If the input has a random structure (high entropy) no compression is possible. - random, lawless, uncompressible, high entropy - ordered, lawfull, compressible, low entropy

Are autoencoder *really* feasible as a data compression method? Well, in a strict computer science sense, not so much. Let's consider why. If (as usual) the internal layer has fewer units of the input, autoencoders can be seen as a form data compression. This 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 (unsupervised, and the data itself is used as both training data and the target).

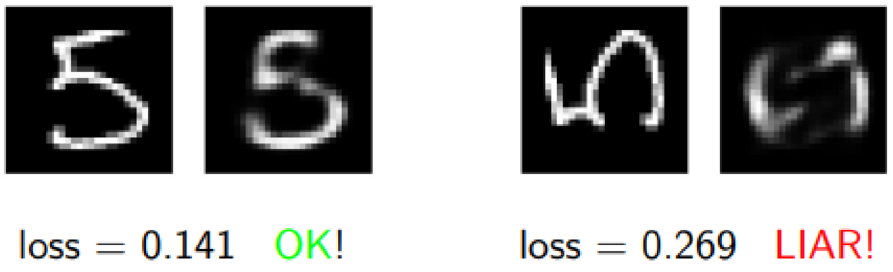
What are they good for?

Not 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.

Autoencoders for anomaly detection

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.

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.*


loss = 0.141 OK!
loss = 0.269 LIAR!