Deep learning

9.2. Looking at activations

François Fleuret

https://fleuret.org/dlc/

UNIVERSITÉ DE GENÈVE

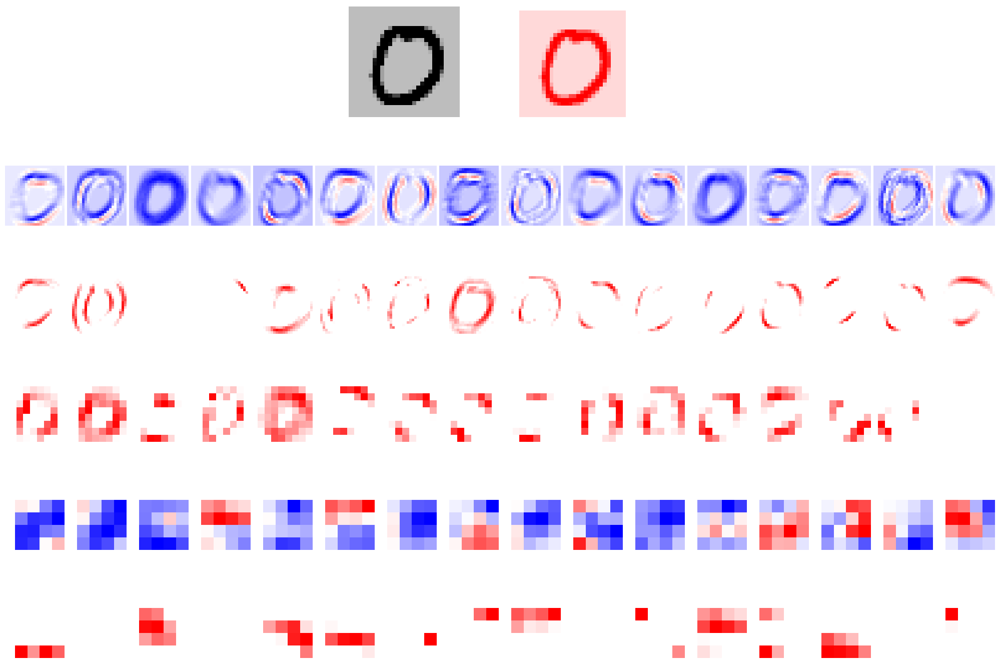
An alternative approach is to look at the activations themselves.

Since the convolutional layers maintain the 2d structure of the signal, the activations can be visualized as images, where the local coding at any location of an activation map is associated to the original content at that same location.

Given the large number of channels, we have to pick a few at random.

Since the representation is distributed across multiple channels, individual channel have usually no clear semantic.

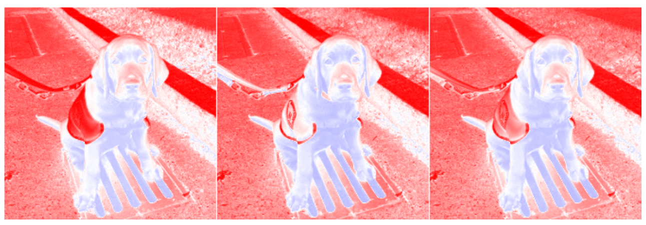
A MNIST character with LeNet (LeCun et al., 1998).

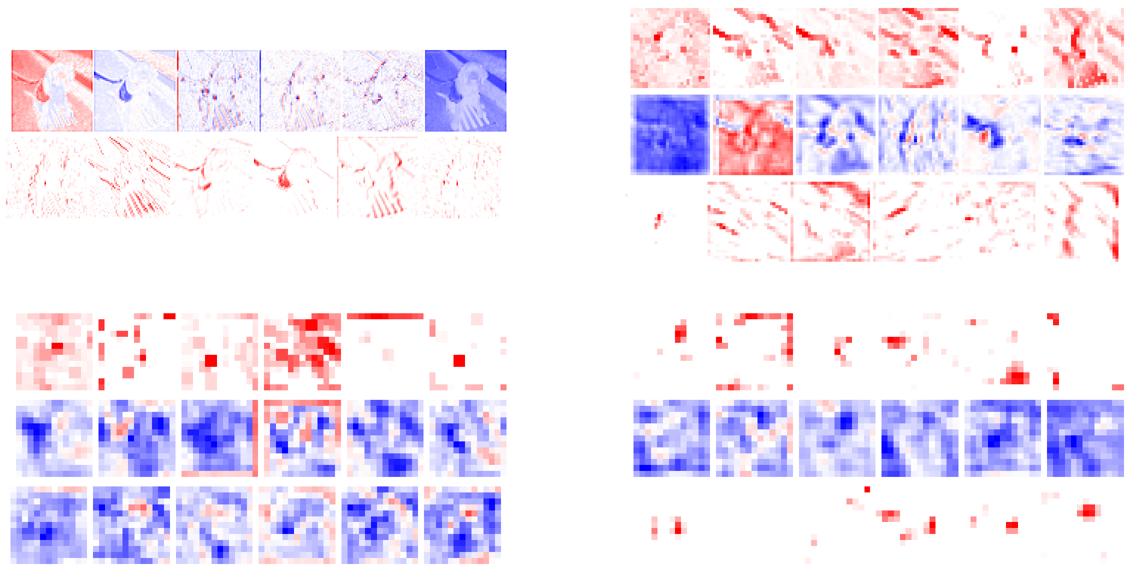


Notes

These images show the activation maps in the successive layers. We use red for positive values, blue for negatives ones, and white for zero.
The top-left grayscale image is the input signal, and top-right red one is its signed version, which has only positive values.
The first row shows the output of the first convolutional layer, which exhibits nice edge detections.
The second row shows activations after the first ReLU. This is why there only have positive values.
The third row shows activations after the first max pooling layer, which reduces the resolution. This is why the images are smaller and more pixelated.
The fourth row is the activations after the second convolution, and the fifth row after the second ReLU. These two last rows are harder to interpret.

An RGB image with AlexNet (Krizhevsky et al., 2012).

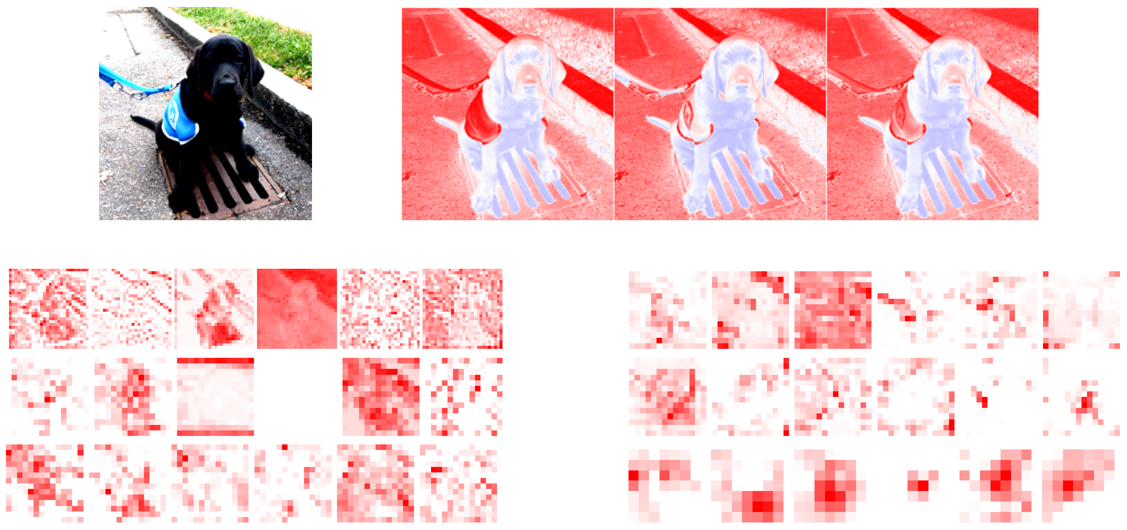




Notes

These images shows activations of AlexNet when trained for image classification on ImageNet.
The first row shows the original image on the left, and its three channels after normalization on the right. The normalization subtracts the mean and divide by the standard deviation, hence may produce negative values.
Although it is quite hard to precisely tell the role of the filters, we can see some edges detectors and some dark area detectors. The deeper we go in the network, the more difficult it becomes to understand what is going on.

ILSVRC12 with ResNet152 (He et al., 2015).



Notes

As on the previous slide, the first row shows the original image on the left, and its three channels after normalization on the right.
The next rows show the activations after the ReLU following a ResNet block (this is why we only see positive values). ResNet reduces the activation maps less than AlexNet and we obtain larger images.
Yet again it is hard to understand what is really encoded, although the activation maps do not show random noise at all, rather blobs that can be interpreted as a part of an object being detected.

Yosinski et al. (2015) developed analysis tools to visit a network and look at the internal activations for a given input signal.

This allowed them in particular to find units with a clear semantic in an AlexNet-like network trained on ImageNet.

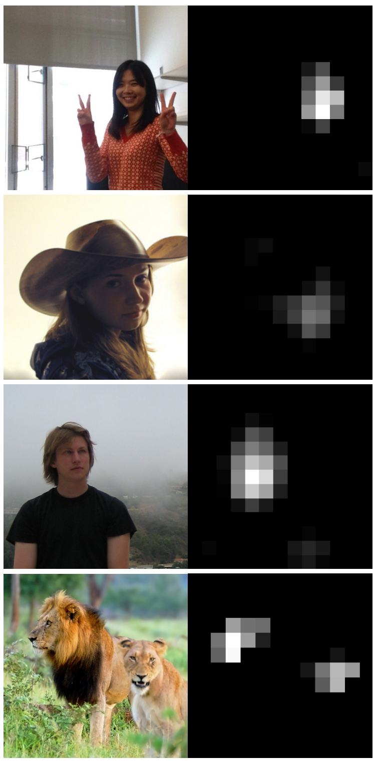


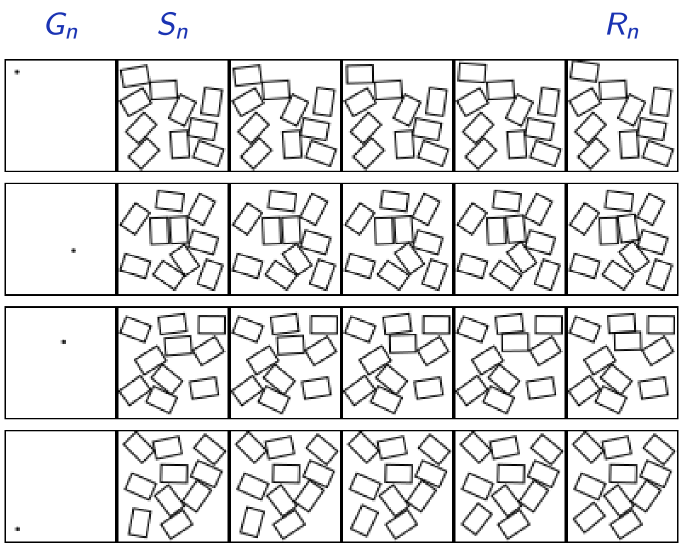
Figure 2. A view of the 13×13 activations of the 151st channel on the conv5 layer of a deep neural network trained on ImageNet, a dataset that does not contain a face class, but does contain many images with faces. The channel responds to human and animal faces and is robust to changes in scale, pose, lighting, and context, which can be discerned by a user by actively changing the scene in front of a webcam or by loading static images (e.g. of the lions) and seeing the corresponding response of the unit. Photo of lions via Flickr user arnolouise, licensed under CC BY-NC-SA 2.0.

(Yosinski et al., 2015)

Notes

This illustration shows the activations (right column) of the 151st channel of the conv5 layer of AlexNet for the input images (left column).
The activation maps are of size 13×13 and have been rescaled to the original image size to match with the receptive fields.
This particular channel has a high response for face pattern of animals and humans. This is very interesting because when the network was trained, the notion of face was never provided as is, but just image classes (lion, tiger, human, etc.). And even if the class were provided, AlexNet is a classification network for which no location is provided.
It is very satisfying to see that a clear part-like and semantic-like detector is emerging in the network, even though the information about the location of these specific parts are never provided, but only indirectly available through the labels at the full image level. It is also interesting to see that this structure appears in different classes: the network has “factorized” the detection of this structure to perform the classification efficiently.

Prediction of 2d dynamics with a 18 layer residual network.



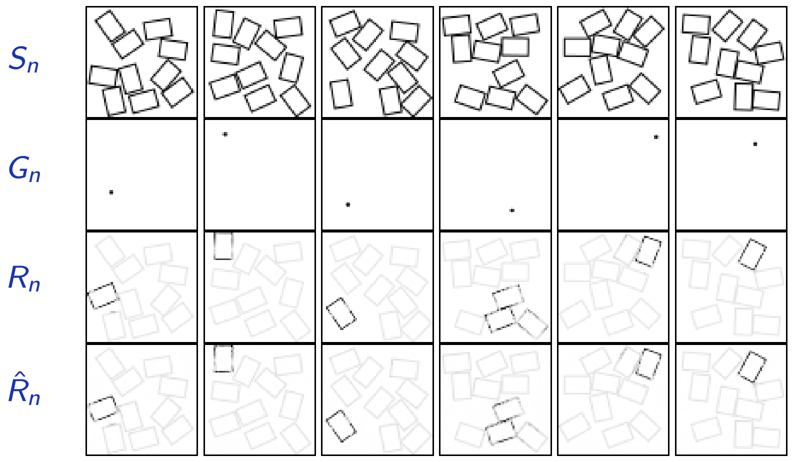
(Fleuret, 2016)

Notes

Here a ResNet is trained to predict the dynamics of rigid 2d rectangles.

- S_n is an image of rectangles dispatched in 2d randomly so that they do not overlap.
- G_n is the location of a "grabbing point" selected at random in the rectangles' interior.
- R_n is the resulting configuration after simulating several time steps during which a force was applied upward at the grabbing point. Images between G_n and R_n are the intermediate steps.

The simulator takes into account the physics and dynamics, torque, inertia, collisions.
For instance, on the first row, the upper left rectangle is selected (roughly at its top left corner) and we can see that it is moved upward in the next images on the right. Since there is no rectangle above, only this one is displaced. On the second row, one rectangle in the middle is pulled upwards, and after four steps, has pushed on two rectangles: one above and one on its right.
We have trained a network which takes as input the two images G_n and S_n as a $2 \times H \times W$ tensor, and should predict the final configuration R_n as an image.
Note that the rectangle positions are not provided as-is, rather the network has to figure out the composition structure of the image and the underlying geometric constraints from the pixels alone.



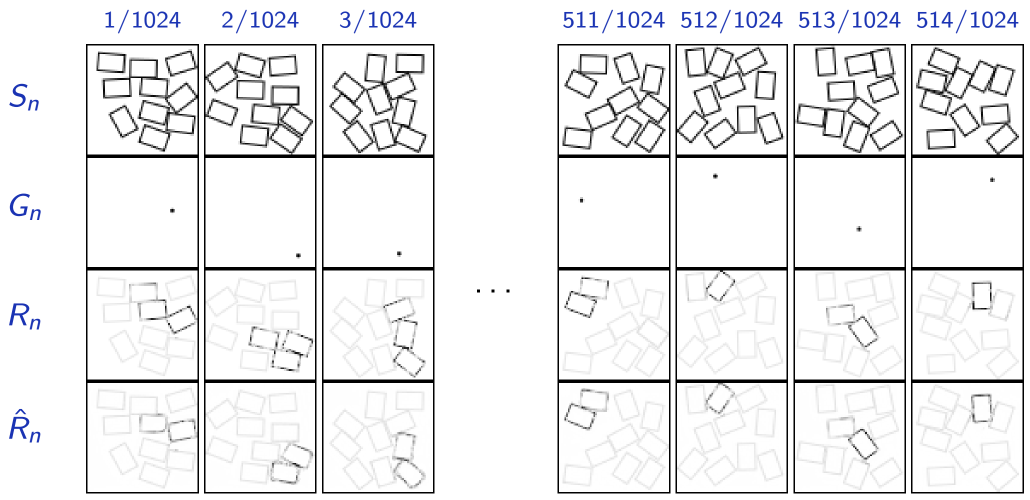
(Fleuret, 2016)

Notes

The network can successfully be trained for the task.
The images show for test samples (one column per test sample):

- S_n: the initial configuration,
- G_n: the grabbing point at which the constant force is applied upwards,
- R_n: the ground simulated truth, in which the difference with S_n is highlighted in black,
- $\hat{R}_n$: the prediction of the network after training, similarly highlighted.

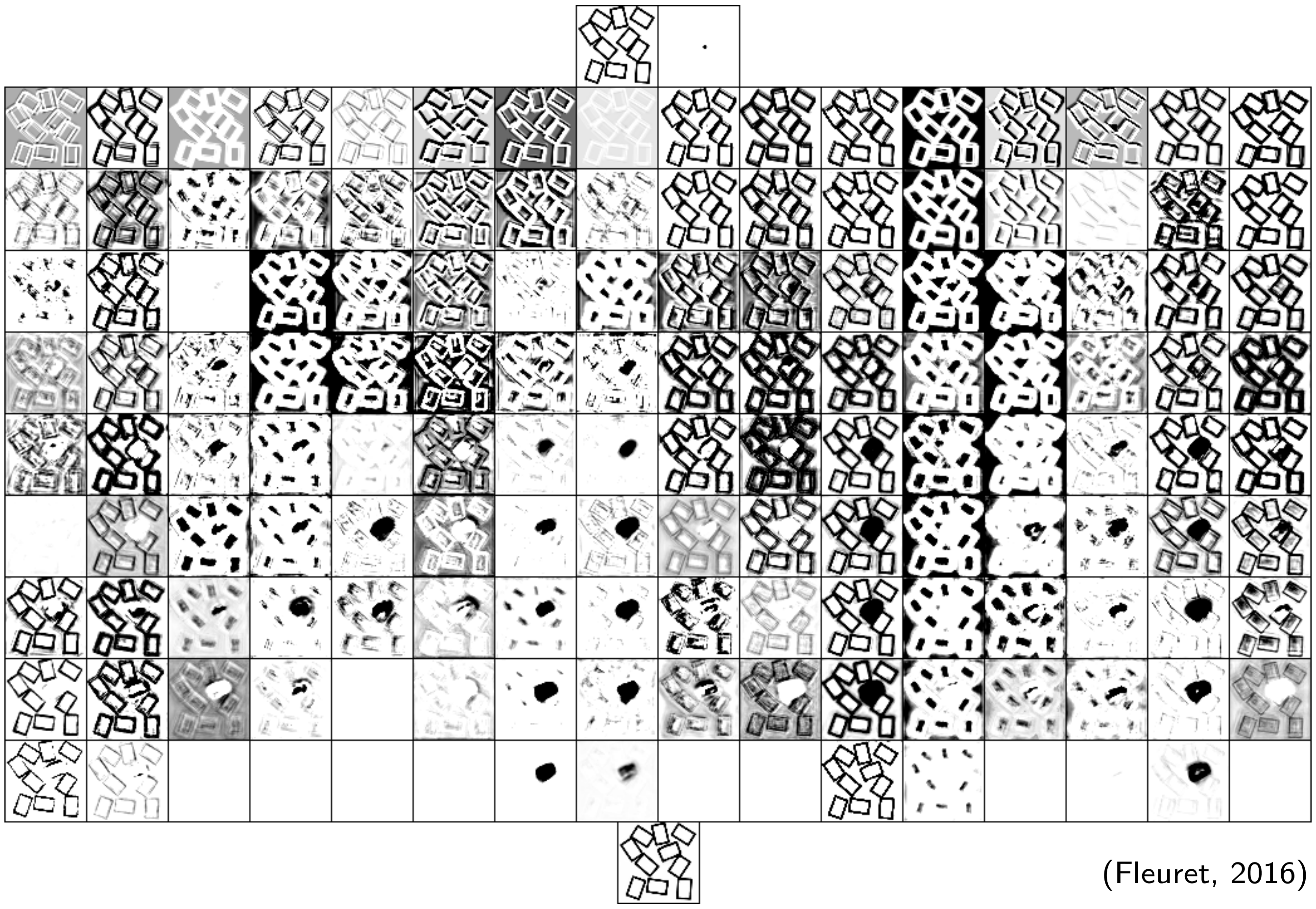
Visually, the network correctly predicts the outcome of the displacement as $\hat{R}_n \simeq R_n$ look similar.



(Fleuret, 2016)

Notes

To have a better understanding of how well it works, we have ranked 1024 test images by decreasing prediction error. The images on the left show the worst predictions, while the images on the right are around the median.
The worst predictions correspond to difficult situations where the network fails at predicting a collision and the resulting chain of collision and subsequent displacement of neighbor boxes.
These results are satisfactory: with an image as input, the network was able to understand the notion of solid entity delimited by edges which should move in a consistent manner, maintaining its shape while translating and rotating. It additionally modeled properly the dynamics, torque, and collisions.



(Fleuret, 2016)

Notes

The activation maps after each residual block can be visualized as images. The pair of images at the top is the input while the next nine rows show each the sixteen activation maps of each of the nine residual blocks. The image at the bottom is the output.
Once again, it is very hard to have a precise understanding of what is going on.
Some channels are able to identify the part which should be moved by having a strong response on it, and some other channels are able to remove this part. This is indeed important to identify what rectangles should not take part in the motion to leave the proper parts of the image unchanged and synthetize images without the moving parts.

Layers as embeddings

In the classification case, the network can be seen as a series of processings aiming as disentangling classes to make them easily separable for the final decision.

In this perspective, it makes sense to look at how the samples are distributed spatially after each layer.

The main issue to do so is the dimensionality of the signal. If we look at the total number of dimensions in each layer:

- A MNIST sample in a LeNet goes from 784 to up to 18k dimensions,
- A ILSVRC12 sample in ResNet152 goes from 150k to up to 800k dimensions.

This requires a mean to project a [very] high dimension point cloud into a 2d or 3d “human-brain accessible” representation

We have already seen PCA and k-means as two standard methods for dimension reduction, but they poorly convey the structure of a smooth low-dimension and non-flat manifold.

It exists a plethora of methods that aim at reflecting in low-dimension the structure of data points in high dimension.

Notes

k-means is a good methods when we have clusters, but not when having smooth and continuous manifold,
When the data is distributed along a curved manifold, PCA "wastes" dimensions to capture the curvature, even if its intrinsic dimension is small.

Given data-points in high dimension

$$\mathscr{D} = \left\{ x_n \in \mathbb{R}^D, \ n = 1, \dots, N \right\}$$

the objective of data-visualization is to find a set of corresponding low-dimension points

$$\mathscr{E} = \left\{ y_n \in \mathbb{R}^C, \ n = 1, \dots, N \right\}$$

such that the positions of the ys "reflect" that of the xs.

Notes

To make the representation meaningful to the human eye, C should be 2 or 3.
The projection should provide a good sense in terms of both metrics and geometrical (grouping, topology, connexity) to get an understanding of where the x_ns are dispatched in the high dimensional space.

The **t-Distributed Stochastic Neighbor Embedding** (t-SNE) proposed by van der Maaten and Hinton (2008) optimizes with SGD the y_is so that the distributions of distances to close neighbors of each point are preserved.

It actually matches for $\mathbb{D}_{\mathsf{KL}}$ two distance-dependent distributions: Gaussian in the original space, and Student t-distribution in the low-dimension one.

The scikit-learn toolbox

http://scikit-learn.org/

is built around SciPy, and provides many machine learning algorithms, in particular embeddings, among which an implementation of t-SNE.

The only catch to use it in PyTorch is the conversions to and from numpy arrays.

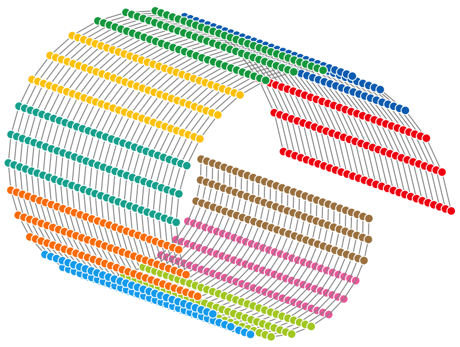
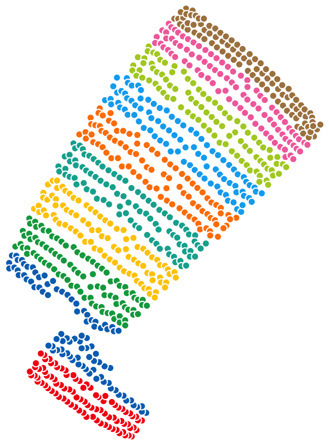
```
from sklearn.manifold import TSNE

# x is the array of the original high-dimension points
x_np = x.numpy()
y_np = TSNE(n_components = 2, perplexity = 50).fit_transform(x_np)
# y is the array of corresponding low-dimension points
y = torch.from_numpy(y_np)
```

`n_components` specifies the embedding dimension and `perplexity` states [crudely] how many points are considered neighbors of each point.

Notes

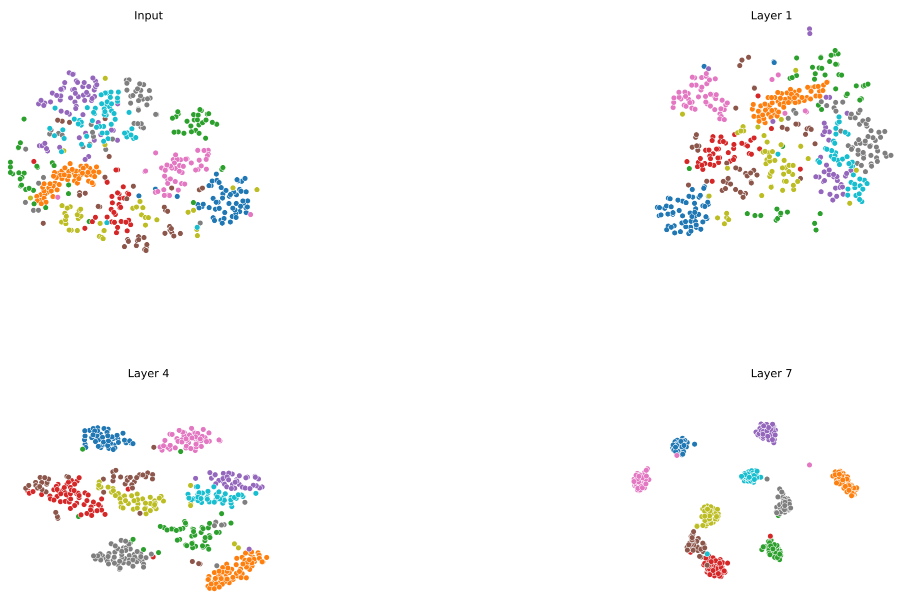
The perplexity can be interpreted as the "number of points" we consider neighbors of x_n.

t-SNE unrolling of the swiss roll (with one noise dimension)

Notes

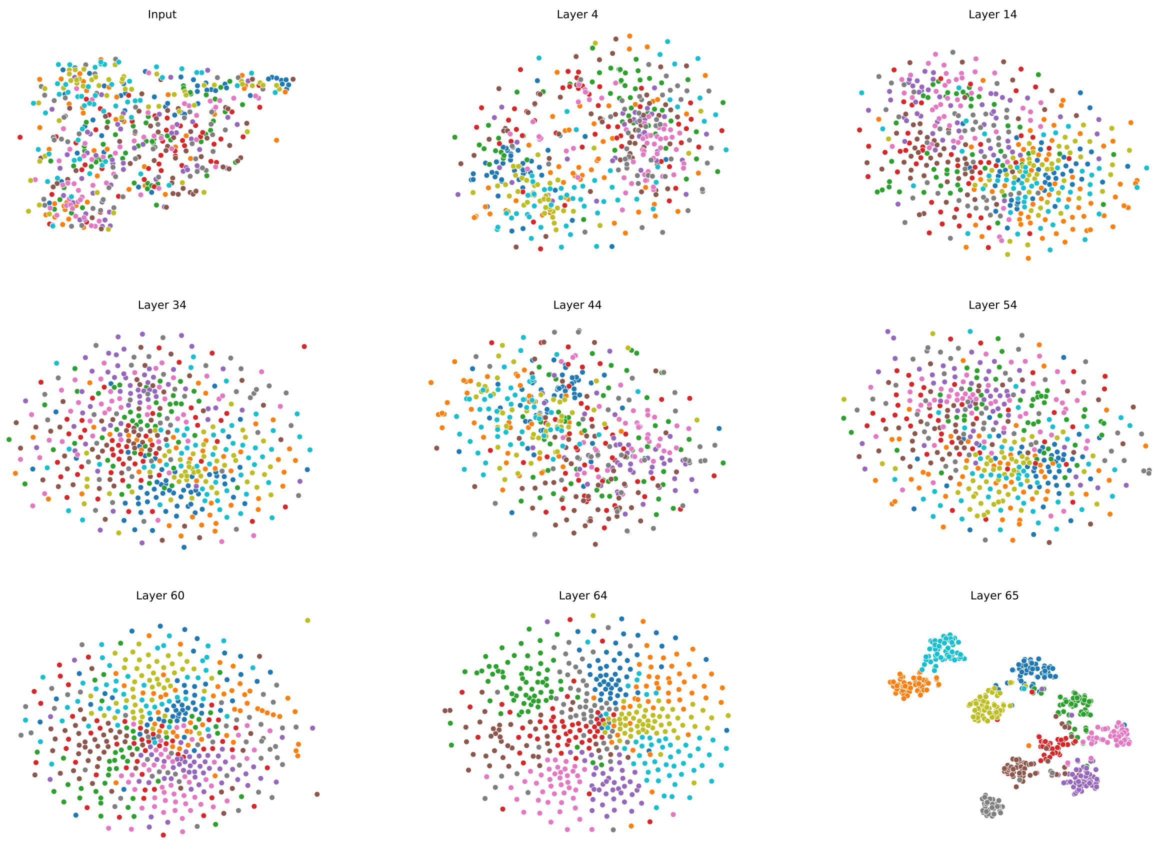
We consider a simple example with data points from $\mathbb{R}^4$ where the first three dimensions are dispatched on a rolled regular grid (a "Swiss roll") and the last dimension is pure Gaussian noise. The left picture shows the projection of the three first coordinates, where the points are colored by row.
The right picture shows the result obtained with t-SNE to project the 4d points in 2d.
Although the projection is not perfect, it captured a lot of the structure. Since t-SNE optimizes the location of the y_n, it may have gotten stuck in a local minimum, and could not "fix" some shearing. But overall, many neighborhoods are maintained.



t-SNE for LeNet on MNIST

Notes

We apply t-SNE on a subset of the MNIST data set as encoded in successive layers of a LeNet model.
Each point corresponds to a sample and each color corresponds to a class.
When we apply t-SNE directly on the input images of dimension 784, we obtain some clusters of classes: the morphology of the distribution of MNIST is such that a simple t-SNE on the raw data make it easy to do the classification.
As t-SNE is applied on the activation maps produced by deeper layers, we can see that classes get pushed away from each other.



t-SNE for a home-baked ResNet (no pooling, 66 layers) CIFAR10

Notes

When we apply the exact same process a ResNet trained on CIFAR10, we see that the task is more difficult.
The result of t-SNE on the input data shows that the classes are deeply mixed. No cluster appear in the low dimensional space.
When going though layers, the population get more and more separated. Layer 64 is the last layer before the final global average pooling, and layer 65 is the last layer before the final decision. We can see that layers are pushing points in the high dimensional space far from each other and reconfigure the topology of the space so that populations that have to be processed the same way will be at the same location in a Euclidean sense.

References

F. Fleuret. **Predicting the dynamics of 2d objects with a deep residual network**. CoRR, abs/1610.04032, 2016.

K. He, X. Zhang, S. Ren, and J. Sun. **Deep residual learning for image recognition**. CoRR, abs/1512.03385, 2015.

A. Krizhevsky, I. Sutskever, and G. Hinton. **Imagenet classification with deep convolutional neural networks**. In Neural Information Processing Systems (NIPS), 2012.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. **Gradient-based learning applied to document recognition**. Proceedings of the IEEE, 86(11):2278–2324, 1998.

L. van der Maaten and G. Hinton. **Visualizing high-dimensional data using t-SNE**. Journal of Machine Learning Research (JMLR), 9:2579–2605, 2008.

J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. **Understanding neural networks through deep visualization**. In Deep Learning Workshop, International Conference on Machine Learning (WS/ICML), 2015.