9.2. Looking at activations
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.
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).
An RGB image with AlexNet (Krizhevsky et al., 2012).
An RGB image with AlexNet (Krizhevsky et al., 2012).
An RGB image with AlexNet (Krizhevsky et al., 2012).
An RGB image with AlexNet (Krizhevsky et al., 2012).
An RGB image with AlexNet (Krizhevsky et al., 2012).
ILSVRC12 with ResNet152 (He et al., 2015).
ILSVRC12 with ResNet152 (He et al., 2015).
ILSVRC12 with ResNet152 (He et al., 2015).
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.

Although the last three layers are sensitive to small input changes, much of the lower layer computation is more robust. For example, when visualizing the conv5 layer, one can find many invariant detectors for faces, shoulders, text, etc. by moving oneself or objects in front of the camera. Even though the 1000 classes contain no explicitly labeled faces or text, the network learns to identify these concepts simply because they represent useful partial information for making a later classification decision. One face detector, denoted conv5_{151} (channel number 151 on conv5), is shown in Figure 2 activating for human and lion faces and in Figure 1 activating for a cat face. Zhou et al. (2014) recently observed a similar effect where convnets trained only to recognize different scene types — playgrounds, restaurant patios, living rooms, etc. — learn object detectors (e.g. for chairs, books, and sofas) on intermediate layers. The reader is encouraged to try this visualization tool out for him or herself. The code, together with pre-trained models and images synthesized by gradient ascent, can be downloaded at http://yosinski.com/deepvis.

3. Visualizing via Regularized Optimization

The second contribution of this work is introducing several regularization methods to bias images found via optimization toward more visually interpretable examples. While each of these regularization methods helps on its own, in combination they are even more effective. We found useful combinations via a random hyperparameter search, as discussed below.

Formally, consider an image $x \in \mathbb{R}^{C \times H \times W}$, where $C = 3$ color channels and the height ($H$) and width ($W$) are both 227 pixels. When this image is presented to a neural network, it causes an activation $a_i(x)$ for some unit $i$, where for simplicity $i$ is an index that runs over all units on all layers. We also define a parameterized regularization function $R_\theta(x)$ that penalizes images in various ways.

Our network was trained on ImageNet by first subtracting the per-pixel mean of examples in ImageNet before inputting training examples to the network. Thus, the direct input to the network, $x$, can be thought of as a zero-centered input. We may pose the optimization problem as finding an image $x^*$ where

$$x^* = \arg \max_x (a_i(x) - R_\theta(x))$$

In practice, we use a slightly different formulation. Because we search for $x^*$ by starting at some $x_0$ and taking gradient steps, we instead define the regularization via an operator $r_\theta(\cdot)$ that maps $x$ to a slightly more regularized version of itself. This latter definition is strictly more expressive, allowing regularization operators $r_\theta$ that are not

(Yosinski et al., 2015)
Prediction of 2d dynamics with a 18 layer residual network.

(Fleuret, 2016)
(Fleuret, 2016)
(Fleuret, 2016)
(Fleuret, 2016)
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.
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.
Given data-points in high dimension

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

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

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

such that the positions of the $y$s “reflect” that of the $x$s.
The **t-Distributed Stochastic Neighbor Embedding** (t-SNE) proposed by van der Maaten and Hinton (2008) optimizes with SGD the $y_i$s so that the distances to close neighbors of each point are preserved.
The t-Distributed Stochastic Neighbor Embedding (t-SNE) proposed by van der Maaten and Hinton (2008) optimizes with SGD the $y_i$’s so that the distances to close neighbors of each point are preserved.

It actually matches for $D_{KL}$ 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.
t-SNE unrolling of the swiss roll (with one noise dimension)
t-SNE unrolling of the swiss roll (with one noise dimension)
t-SNE for LeNet on MNIST
t-SNE for LeNet on MNIST
Layer #4

t-SNE for LeNet on MNIST
Layer #7

t-SNE for LeNet on MNIST
t-SNE for a home-baked resnet (no pooling, 66 layers) CIFAR10
t-SNE for a home-baked resnet (no pooling, 66 layers) CIFAR10
Layer #14

t-SNE for a home-baked resnet (no pooling, 66 layers) CIFAR10
t-SNE for a home-baked resnet (no pooling, 66 layers) CIFAR10
t-SNE for a home-baked resnet (no pooling, 66 layers) CIFAR10
t-SNE for a home-baked resnet (no pooling, 66 layers) CIFAR10
t-SNE for a home-baked resnet (no pooling, 66 layers) CIFAR10
t-SNE for a home-baked resnet (no pooling, 66 layers) CIFAR10
t-SNE for a home-baked resnet (no pooling, 66 layers) CIFAR10
t-SNE for a home-baked resnet (no pooling, 66 layers) CIFAR10
t-SNE for a home-baked resnet (no pooling, 66 layers) CIFAR10
t-SNE for a home-baked resnet (no pooling, 66 layers) CIFAR10
t-SNE for a home-baked resnet (no pooling, 66 layers) CIFAR10
The end
References


