



(/blog/)

## Visualizing with t-SNE

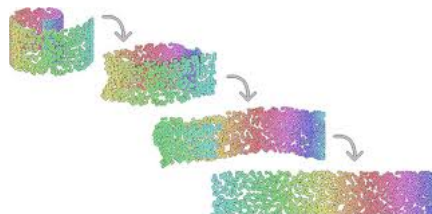
Luke Metz - August 25, 2015

---

### Data visualization

A big part of working with data is getting intuition on what those data show. Staring at raw data points, especially when there are many of them, is almost never the correct way to tackle a problem. Low dimensional data are easy to visually inspect. You can simply pick pairs of dimensions and plot them against each other. Say, for example, I wanted to see how distance to a subway stop is correlated to house price — I'd make a plot. If the points were seemingly random, then there is no correlation. If the points follow some pattern (such as a line) then there is a correlation. However, as the number of dimensions increases, this technique makes less sense because 3D plots are a lot harder to read than 2D plots. Trying to compare four dimensions of data against each other is nearly impossible. So, what can we do?

One strategy is to embed the high dimensional data into a smaller dimensional space. For visualization purposes, we generally try to reduce the number of dimensions to two. To build intuition, consider the following toy problem that has three dimensions. The input is points arranged in a curl. As humans, we can see that there is structure in this 3D representation, and we can use that structure to effectively unroll the 3D space and flatten it to a 2D space.



(<https://indico.io/blog/wp-content/uploads/2015/08/embedding.jpg>)

Visual non-linear dimensionality reduction.

Source:

<http://axon.cs.byu.edu/papers/gashler2011smc.pdf>

(<http://axon.cs.byu.edu/papers/gashler2011smc.pdf>)

This is exactly what we want to do, except instead of using human intuition, we will use the power of machine learning (<https://indico.io/blog/the-founders-guide-to-machine-learning-why-you-shouldnt-build-your-own-models/>). This style of operation is commonly called **nonlinear dimensionality reduction**, or manifold learning. For more information I would recommend reading the Nonlinear Dimensionality Reduction ([https://en.wikipedia.org/wiki/Nonlinear\\_dimensionality\\_reduction](https://en.wikipedia.org/wiki/Nonlinear_dimensionality_reduction)) Wikipedia page. In this article, I'm going to focus on one my favorite kinds of embedding algorithms: t-SNE.

dimensional visualization problems. Instead of trying to preserve the global structure like many dimensionality reduction techniques, t-SNE tries to group local data points closer to each other, which (in my experience) is a better match for building human reasoning. Part of the power of this technique is that it can normally be treated as a black box, and I'll do just that for the remainder of this blog post. For those interested, I would highly recommend this fantastic Google Tech Talk (<https://www.youtube.com/watch?v=RJVL80Gg3IA>) which goes into a lot more detail.

## Easy to use

In most cases, t-SNE just works. It's implemented in many different computer languages — you can check which ones on Laurens van der Maaten's site (<http://lvdmaaten.github.io/tsne/#implementations>). For Python users, there is a PyPI package called `tsne`. You can install it easily with `pip install tsne`.

To make use of this, we first need a dataset of some kind to try to visualize. For simplicity, let's use MNIST, a dataset of handwritten digits. The code below uses `skdata` to load up `mnist`, converts the data to a suitable format and size, runs `bh_tsne`, and then plots the results.

```
import numpy as np
from skdata.mnist.views import OfficialImageClassification
from matplotlib import pyplot as plt
from tsne import bh_sne

# load up data
data = OfficialImageClassification(x_dtype="float32")
x_data = data.all_images
y_data = data.all_labels

# convert image data to float64 matrix. float64 is need for bh_sne
x_data = np.asarray(x_data).astype('float64')
x_data = x_data.reshape((x_data.shape[0], -1))

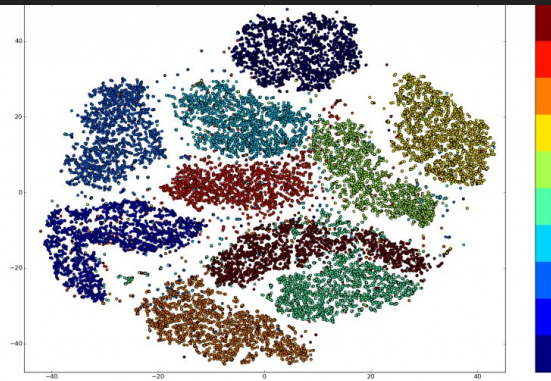
# For speed of computation, only run on a subset
n = 20000
x_data = x_data[:n]
y_data = y_data[:n]

# perform t-SNE embedding
vis_data = bh_sne(x_data)

# plot the result
vis_x = vis_data[:, 0]
vis_y = vis_data[:, 1]

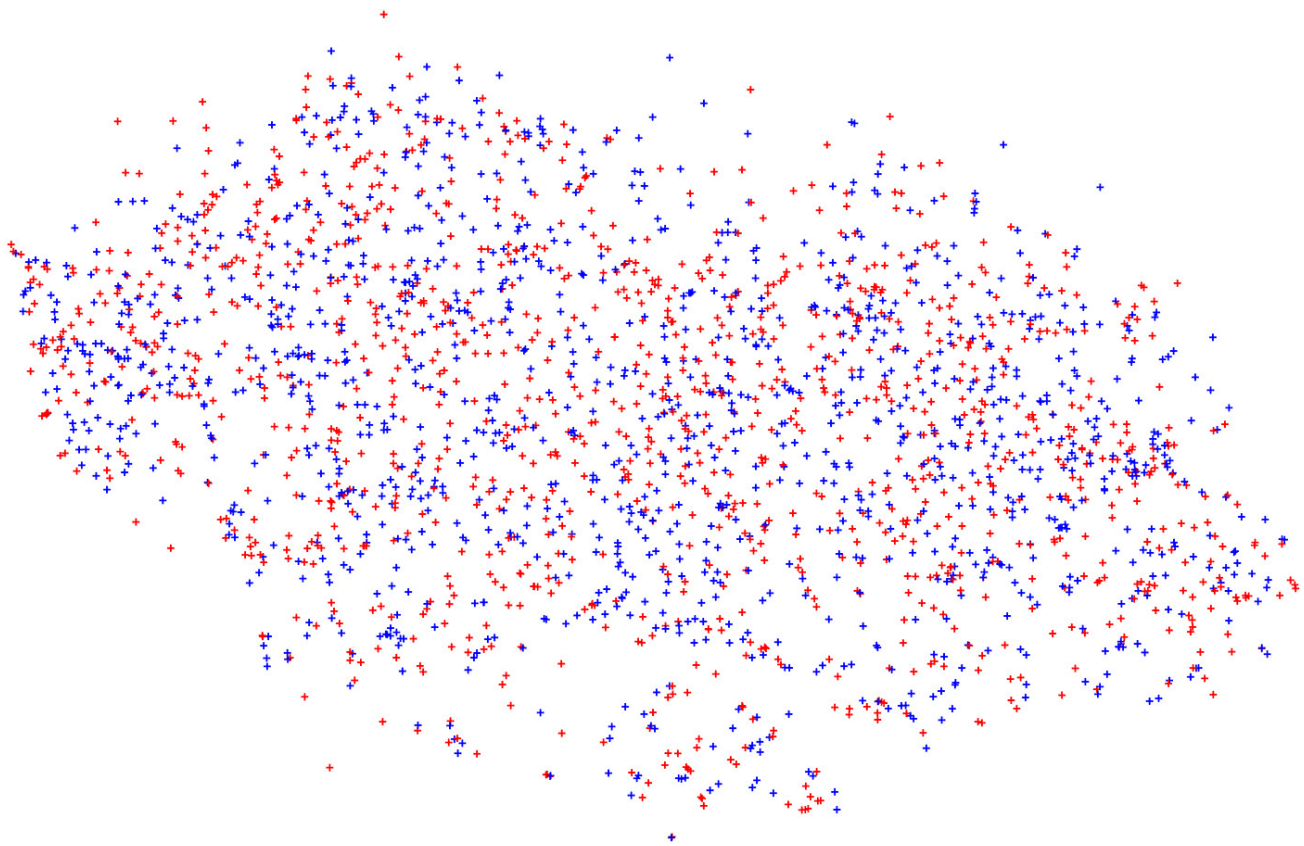
plt.scatter(vis_x, vis_y, c=y_data, cmap=plt.cm.get_cmap("jet", 10))
plt.colorbar(ticks=range(10))
plt.clim(-0.5, 9.5)
plt.show()
```

After about two minutes of execution, the result looks something like this:



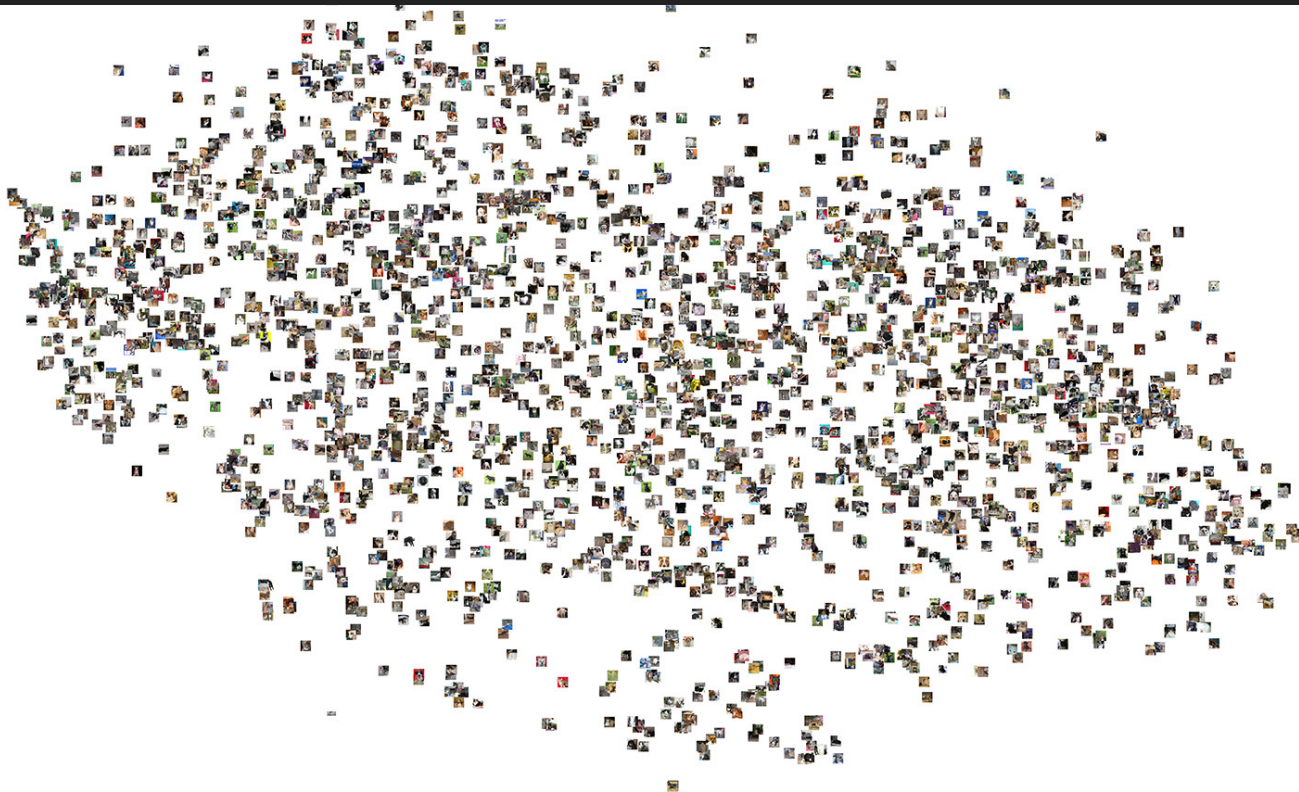
(<https://indico.io/blog/wp-content/uploads/2015/08/mnist.jpg>) While MNIST is a really great dataset for testing and evaluating ideas, this is a rather easy task. Let's try t-SNE on something a little bit harder! Recently, Kaggle hosted an image processing competition titled Dogs vs. Cats (<https://www.kaggle.com/c/dogs-vs-cats>).

Since these images are much more detailed, I would recommend clicking the images to download a higher resolution version for further exploration.



([https://indico.io/blog/wp-content/uploads/2015/08/raw\\_input64\\_dot\\_large.png](https://indico.io/blog/wp-content/uploads/2015/08/raw_input64_dot_large.png))

High Resolution ([https://indico.io/blog/wp-content/uploads/2015/08/raw\\_input64\\_dot\\_large.png](https://indico.io/blog/wp-content/uploads/2015/08/raw_input64_dot_large.png))



(<http://i.imgur.com/EV8Wzq8.jpg>)

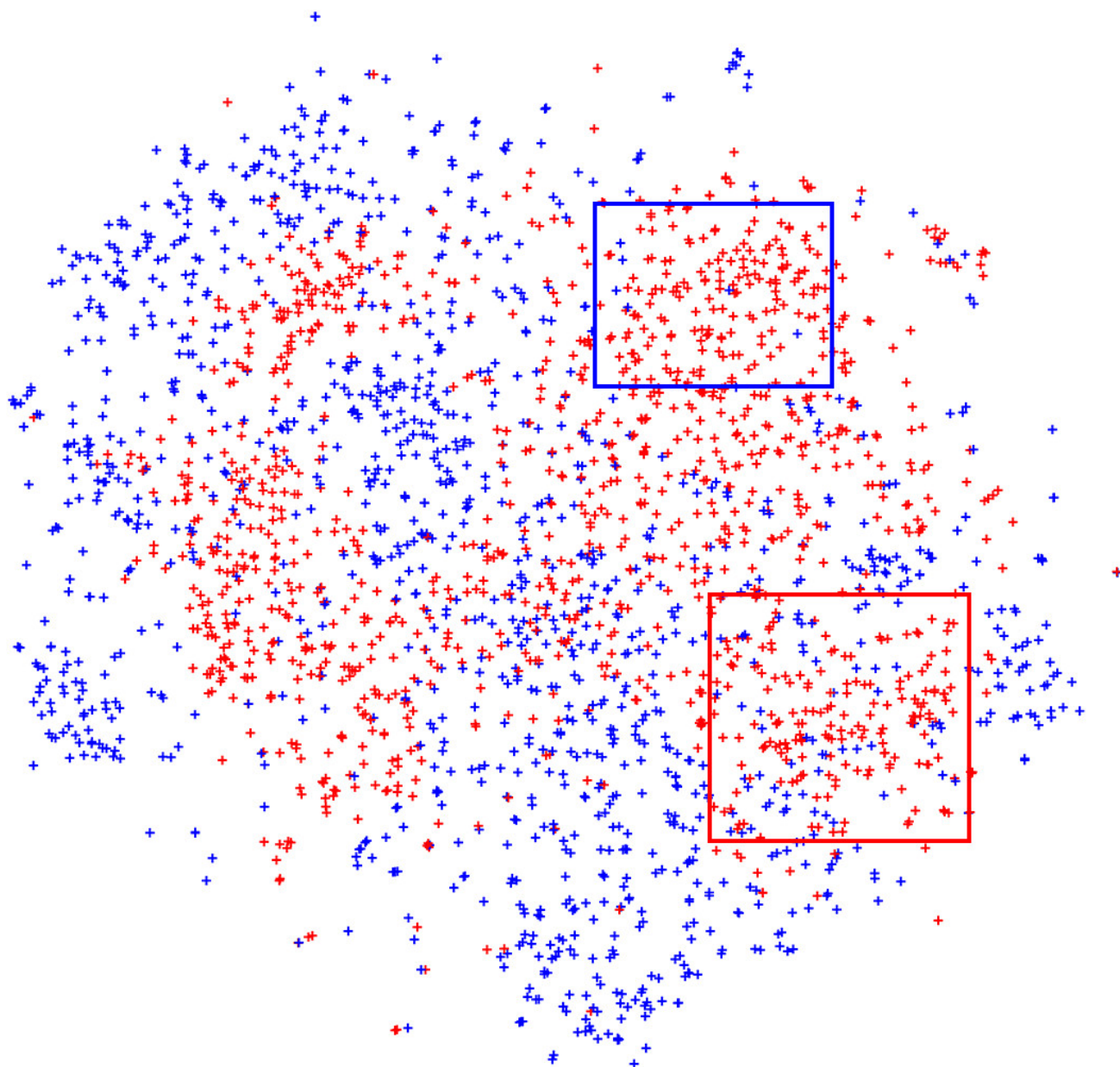
High Resolution (<http://i.imgur.com/EV8Wzq8.jpg>).

Now, these embeddings don't look nearly as good, because they're working on raw pixels values. In general, when we look at images, we look to patterns and objects, not pixel intensities at pixels. It's asking too much of our dimensionality reduction algorithm to get all of that from just pixel values. So, how can we resolve this? Converting images into some representation that preserves the content of the image is a hard problem. These features strive to be invariant (they do not change) under certain constraints. For example, is a cat still a cat depending on the quality of a photo? Does the location of the cat in the image matter? A zoomed in version of a cat is still a cat. The raw pixel values may change drastically, but the subject stays the same.

There are a number of techniques used to make sense of images, such as SIFT ([https://en.wikipedia.org/wiki/Scale-invariant\\_feature\\_transform](https://en.wikipedia.org/wiki/Scale-invariant_feature_transform)) or HOG ([https://en.wikipedia.org/wiki/Histogram\\_of\\_oriented\\_gradients](https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients)), but computer vision research has been moving more towards using convolutional neural networks to create these features. The general idea is to train a very large and very deep neural network on an image classification task to differentiate between many different classes of images. In the processes of learning to classify, the model learns useful feature extractors that can then be used for other tasks. Sadly, these techniques require large amounts of data, computation, and infrastructure to use. Models can take weeks to train and need expensive graphics cards and millions of labeled images. Although it's highly rewarding when you get them to work, sometimes creating these models is just too time intensive.

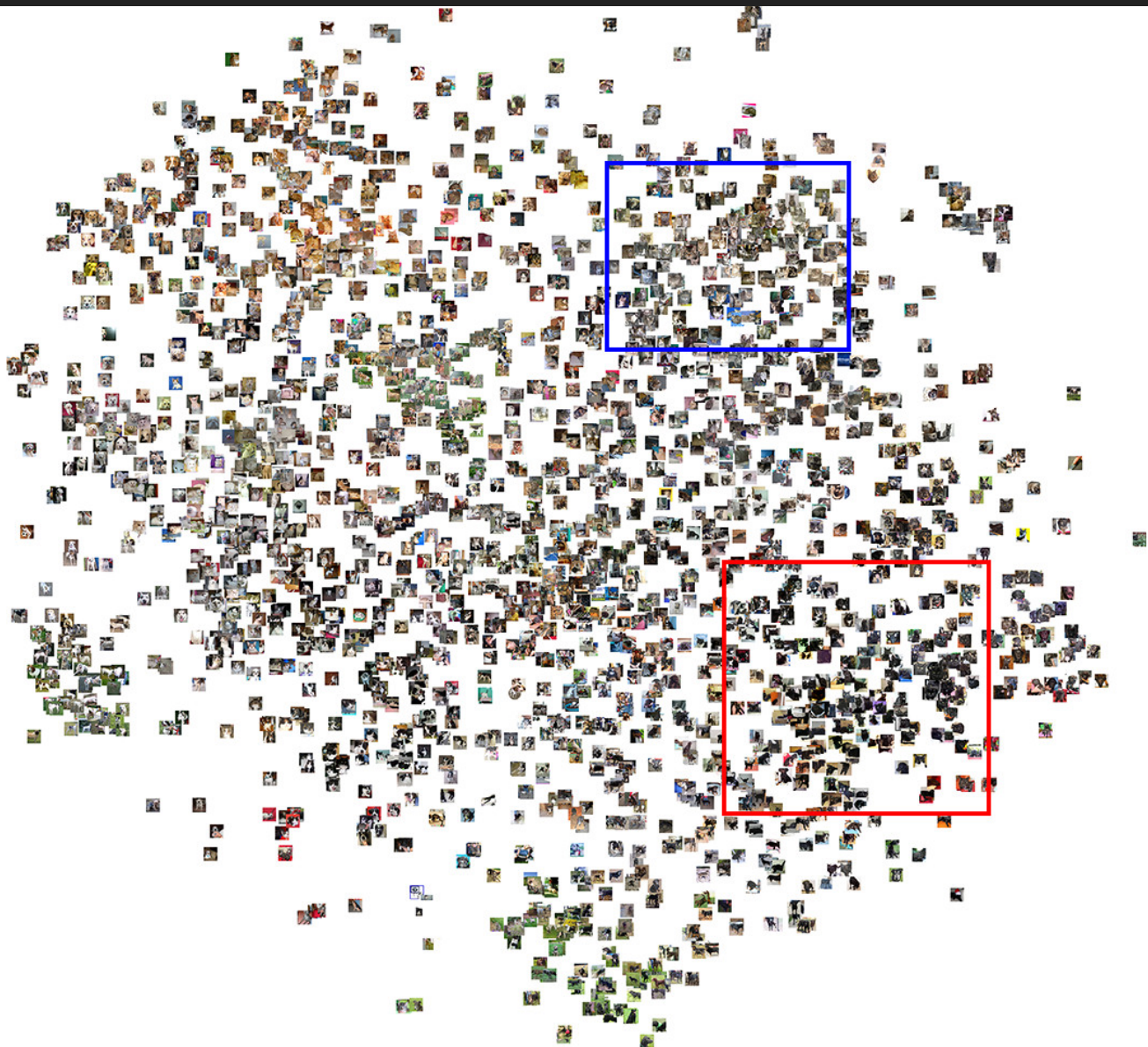
indico (<http://indico.io>) provides a feature extractor with its Image Features API ([https://indico.io/docs#image\\_features](https://indico.io/docs#image_features)), which is built using the same technique I described above: a stack of convolution layers trained on a 1000-way image classification task. To see the effect, we can apply the Image Features API to this dataset and then t-SNE the result to see how it performs against raw pixels.





(<http://i.imgur.com/uCMcy3R.jpg>)

High Resolution (<http://i.imgur.com/uCMcy3R.jpg>)



(<http://i.imgur.com/m8Lhmlo.jpg>)

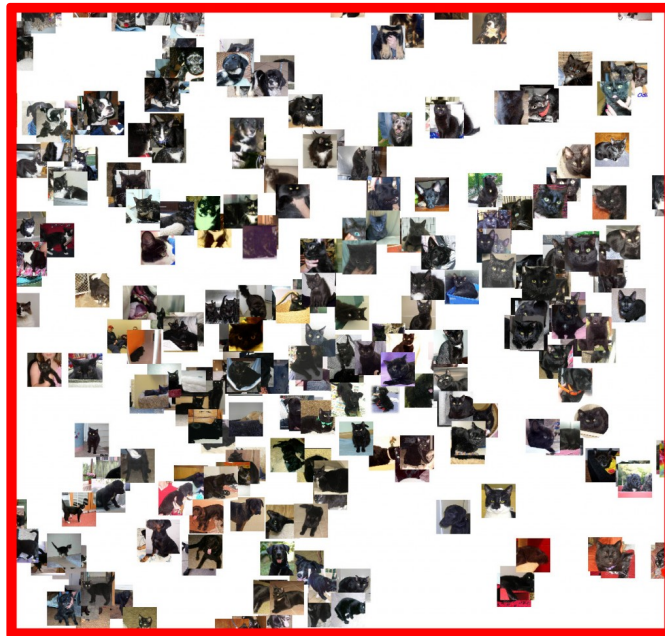
High Resolution (<http://i.imgur.com/m8Lhmlo.jpg>)

Right off the bat we can see that there is much more clustering based off of what is in the image. The features don't just tell us which images are cats and which images are dogs, but instead group them and provide more information. Take the red and blue clusters. Both contain cats, but blue contains grey cats with stripes, while red contains black cats.



([https://indico.io/blog/wp-content/uploads/2015/08/indico\\_features\\_callout\\_blue.jpg](https://indico.io/blog/wp-content/uploads/2015/08/indico_features_callout_blue.jpg))

Cluster containing grey cats.



([https://indico.io/blog/wp-content/uploads/2015/08/indico\\_features\\_callout\\_red.jpg](https://indico.io/blog/wp-content/uploads/2015/08/indico_features_callout_red.jpg))

Cluster containing black cats.

38  
Shares

25

By working in a space that is better attuned for images, we can really start exploring and picking out interesting things in data!

## Conclusion

The applications of t-SNE are limitless. It can be applied anytime there is a high dimensional dataset — it has been applied to text and natural language processing, (<http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/>)speech (<https://cbmm.mit.edu/sites/default/files/publications/CBMM-Memo-022-1406.3884v1.pdf>), and even to visualize Atari game states (<http://rdcu.be/cdlg>).



demystified/". In most applications, these "deep" models can be boiled down to the composition of simple functions that embed from one high dimensional space to another. At first glance, these spaces might seem too large to think about or visualize, but techniques such as t-SNE allow us to start to understand what's going on inside the black box. Now, instead of treating these models as black boxes, we can start to visualize and understand them.

I hope this technique proves useful to you in your work. In my experience, time spent making visualizations such as these are always helpful in determining next steps and understanding my data.

Questions or comments? Feel free to discuss on this post's r/machinelearning thread ([https://www.reddit.com/r/MachineLearning/comments/3id7yn/visualizing\\_with\\_tsne/](https://www.reddit.com/r/MachineLearning/comments/3id7yn/visualizing_with_tsne/))!

### Suggested Posts

Techstars Boston Demo Day Pitch (<https://indico.io/blog/techstars-boston-demo-day-pitch-2/>)

General Sequence Learning Using Recurrent Neural Nets (<https://indico.io/blog/general-sequence-learning-using-recurrent-neural-nets/>)

Technology That Can Adapt to Us: A Look at Prosthetic Advancement (<https://indico.io/blog/technology-that-can-adapt-to-us-a-look-at-prosthetic-advancement-2/>)

Done reading and ready to build?

**GET STARTED ([HTTPS://INDICO.IO/PLANS](https://indico.io/plans))**



Get the latest tutorials, tips, and articles from indico.

[Never miss a post >>](#)



[Home \(/\)](#)

[Hack indico \(/hack\)](#)

[Gallery \(/gallery/\)](#)

[Press \(/blog/press\)](#)

[Blog \(/blog/\)](#)

[RSS Feed \(https://indico.io/blog/feed/\)](https://indico.io/blog/feed/)

[Careers \(https://jobs.lever.co/indico\)](https://jobs.lever.co/indico)

[Docs \(/docs\)](#)

[Team \(/team\)](#)

[Terms of Service \(/terms\)](#)


[Privacy \(/terms#privacy\)](#)

[Contact \(/contact\)](#)

 (<https://github.com/IndicoDataSolutions>)

 (<https://www.facebook.com/IndicoDataSolutions>)  (<https://twitter.com/indicodata>)

 (<https://www.youtube.com/channel/UCGuUwm6PaPkeftGFmNOtTHw>)

 (<https://instagram.com/indicodata/>)



<https://mixpanel.com/f/partner>