Stage 8: Image Analysis

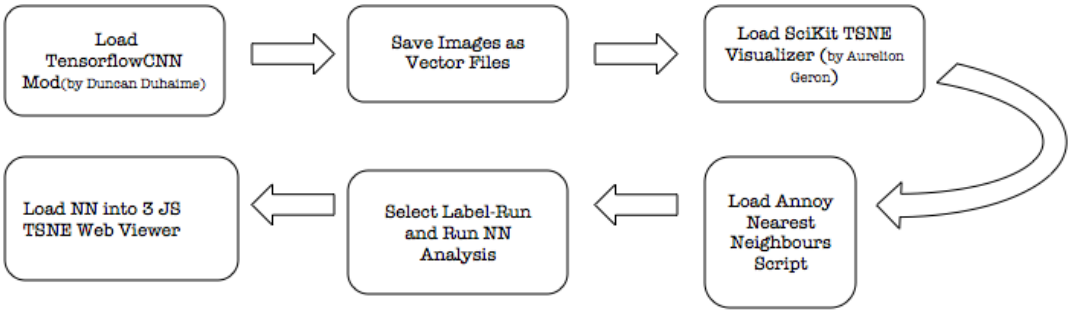


Fig 8.1 *– Workflow Stage Overview*

8.1 Stage Overview

- Exploring London Image Data through Embeddings using the Vector Representations generated in the CNN Classification Process

Having settled on a model from the label classification task we can proceed with searching for properties in the labels which might inform us further as to the visual nature of our image data population.

This part of the study was inspired by the Google Arts project and the work of Douglas Duhaime from Yale Digital Humanities lab using Tensorflow CNNs and TSNE Clustering methods to categorize images based on their visual similarity.

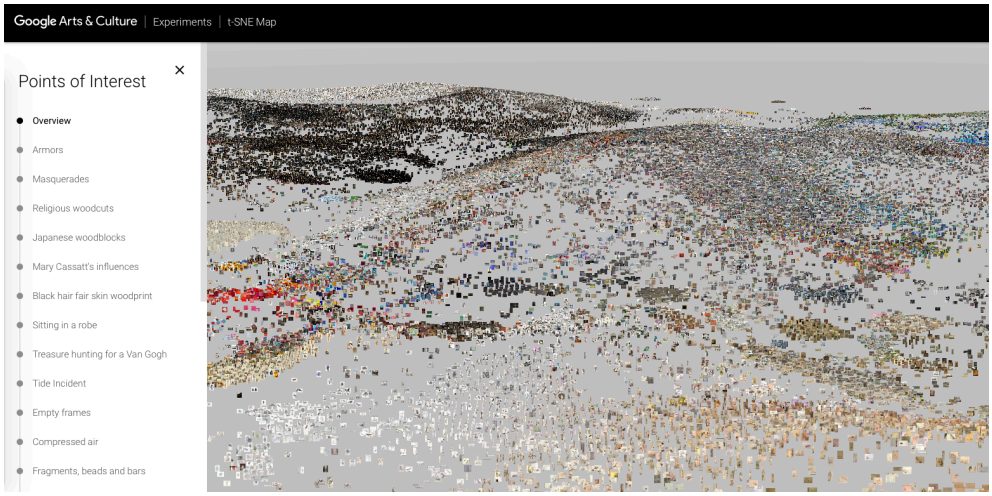


Fig 8.2 *– Google Arts Project*

8.2 Image Similar Concept and Value

A core concept in Computer Vision is that of Image Similarity Analysis whereby a vector space model is used for similarity search between one or more objects. This approach can be used to derive insight of a visually semantic nature into our visual feature space.

Goal: Looking for typologically defining features in our CNN labeled London Building Data Set and to explore

Step 1 – ReName/ReLabel Candidate Images¶
Step 2 - Convert Images into Vectors
Step 3 - Create Labels File and Fit TSNE Model
Step 4 - Plot and Explore TSNE and Dimensionality Reduction Methods

An embedding is a mapping of a discrete, categorical variable to a vector of continuous numbers. In the context of neural networks, embeddings are low-dimensional, learned continuous vector representations of discrete variables. Neural network embeddings are useful because they can reduce the dimensionality of categorical variables and meaningfully represent catagories in the transformed space.

Neural network embeddings have 3 primary purposes:

1 - Finding nearest neighbors in the embedding space. These can be used to make recommendations based on user interests or cluster categories.
2- As input to a machine learning model for a supervised task.
3 - For visualization of concepts and relations between categories.

8.3-Vector Extraction

For similarity tasks, it's generally better to work with float point vectors rather than categorical labels, as vectors capture more of the original object's signal. We can obtain vector representations of images by only slightly modifying the classify_image.py script. In essence, instead of asking the last (softmax) layer of the neural network for the text classifications of input images, we'll ask the penultimate layer of the neural network for the internal model weights for a given image, and we'll store those weights as a vector representation of the input image. This will allow us to perform traditional vector analysis using images (Duhaime - 2017).

8.4 Nearest Neighbors

The extracted layers can be used for processes such as Nearest Neighbor Analysis. A nice way to achieve this functionality is to leverage Erik Bern's Approximate Nearest Neighbors Oh Yeah library(in use by the Spotify application and available from their github page) to identify the approximate nearest neighbors for each image.

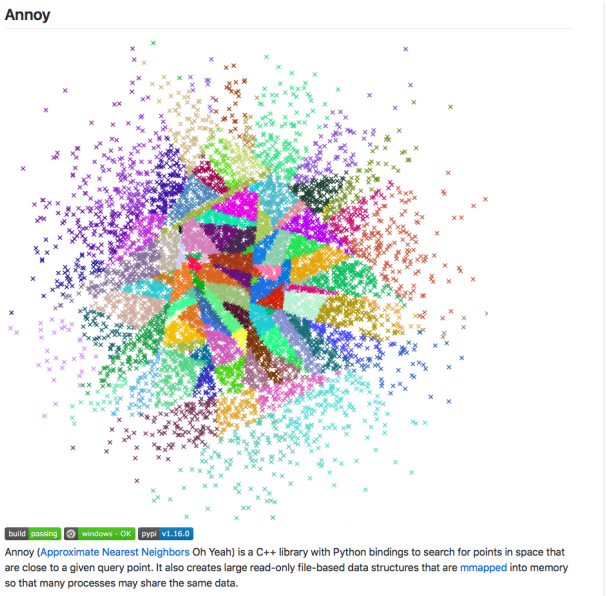


Fig 8.2 *– Annoy NN algorithm*

Using random projections and by building up a tree. At every intermediate node in the tree, a random hyperplane is chosen, which divides the space into two subspaces. This hyperplane is chosen by sampling two points from the subset and taking the hyperplane equidistant from them.

We do this k times so that we get a forest of trees. k has to be tuned to your need, by looking at what tradeoff you have between precision and performance.

Hamming distance (contributed by Martin Aumüller) packs the data into 64-bit integers under the hood and uses built-in bit count primitives so it could be quite fast. All splits are axis-aligned.

Dot Product distance (contributed by Peter Sobot) reduces the provided vectors from dot (or "inner-product") space to a more query-friendly cosine space using a method by Bachrach et al., at Microsoft Research, published in 2014 (Bern).

The Tables below depict the top 8 Nearest Neighbors for 2 images examples from the 4 Property Types of that particular label Run. The image number range reflects whether there are more or less images from originating labeled category. We can see that the from an nearest neighbor analysis attribute characteristics are not highly evident on their own.

Property Type = INDUSTRIAL	Images 200 to 261
Filename=200.jpg	Filename=201.jpg
Nearest Neighbour: 281.jpg	Nearest Neighbour: 202.jpg
Similarity Measure=1.0	Similarity Measure=1.0
Nearest Neighbour: 305.jpg	Nearest Neighbour: 316.jpg
Similarity Measure=0.8073	Similarity Measure=0.8517
Nearest Neighbour: 361.jpg	Nearest Neighbour: 178.jpg
Similarity Measure=0.8071	Similarity Measure=0.8457
Nearest Neighbour: 289.jpg	Nearest Neighbour: 302.jpg
Similarity Measure=0.8055	Similarity Measure=0.8289
Nearest Neighbour: 143.jpg	Nearest Neighbour: 319.jpg
Similarity Measure=0.8045	Similarity Measure=0.8283
Nearest Neighbour: 308.jpg	Nearest Neighbour: 8.jpg
Similarity Measure=0.7998	Similarity Measure=0.8258
Nearest Neighbour: 178.jpg	Nearest Neighbour: 360.jpg
Similarity Measure=0.7967	Similarity Measure=0.815
Nearest Neighbour: 275.jpg	Nearest Neighbour: 45.jpg
Similarity Measure=0.7891	Similarity Measure=0.8143

Property Type = OFFICE	Images 261 to 359
Filename=300.jpg	Filename=301.jpg
Nearest Neighbour: 46.jpg	Nearest Neighbour: 47.jpg
Similarity Measure=1.0	Similarity Measure=1.0
Nearest Neighbour: 160.jpg	Nearest Neighbour: 22.jpg
Similarity Measure=0.8222	Similarity Measure=0.7982
Nearest Neighbour: 18.jpg	Nearest Neighbour: 141.jpg
Similarity Measure=0.8123	Similarity Measure=0.7751
Nearest Neighbour: 56.jpg	Nearest Neighbour: 23.jpg
Similarity Measure=0.8075	Similarity Measure=0.7675
Nearest Neighbour: 44.jpg	Nearest Neighbour: 25.jpg
Similarity Measure=0.803	Similarity Measure=0.7668
Nearest Neighbour: 185.jpg	Nearest Neighbour: 255.jpg
Similarity Measure=0.803	Similarity Measure=0.7647
Nearest Neighbour: 79.jpg	Nearest Neighbour: 41.jpg
Similarity Measure=0.8019	Similarity Measure=0.7615
Nearest Neighbour: 209.jpg	Nearest Neighbour: 103.jpg
Similarity Measure=0.796	Similarity Measure=0.7615

Property Type = FLAT	Images 0 to 100
Filename=0.jpg	Filename=1.jpg
Nearest Neighbour: 0.jpg	Nearest Neighbour: 1.jpg
Similarity Measure=1.0	Similarity Measure=1.0
Nearest Neighbour: 33.jpg	Nearest Neighbour: 153.jpg
Similarity Measure=0.8438	Similarity Measure=0.8103
Nearest Neighbour: 82.jpg	Nearest Neighbour: 211.jpg
Similarity Measure=0.8373	Similarity Measure=0.8072
Nearest Neighbour: 15.jpg	Nearest Neighbour: 168.jpg
Similarity Measure=0.8288	Similarity Measure=0.8026
Nearest Neighbour: 84.jpg	Nearest Neighbour: 23.jpg
Similarity Measure=0.8194	Similarity Measure=0.8019
Nearest Neighbour: 160.jpg	Nearest Neighbour: 213.jpg
Similarity Measure=0.8173	Similarity Measure=0.8014
Nearest Neighbour: 127.jpg	Nearest Neighbour: 10.jpg
Similarity Measure=0.8165	Similarity Measure=0.8
Nearest Neighbour: 18.jpg	Nearest Neighbour: 141.jpg
Similarity Measure=0.809	Similarity Measure=0.7975

Property Type = HOUSE	Images 100 to 200
Filename=100.jpg	Filename=101.jpg
Nearest Neighbour: 19.jpg	Nearest Neighbour: 190.jpg
Similarity Measure=1.0	Similarity Measure=1.0
Nearest Neighbour: 166.jpg	Nearest Neighbour: 199.jpg
Similarity Measure=0.8132	Similarity Measure=0.8
Nearest Neighbour: 141.jpg	Nearest Neighbour: 185.jpg
Similarity Measure=0.8054	Similarity Measure=0.7867
Nearest Neighbour: 96.jpg	Nearest Neighbour: 182.jpg
Similarity Measure=0.7855	Similarity Measure=0.777
Nearest Neighbour: 28.jpg	Nearest Neighbour: 138.jpg
Similarity Measure=0.7798	Similarity Measure=0.7749
Nearest Neighbour: 137.jpg	Nearest Neighbour: 144.jpg
Similarity Measure=0.7761	Similarity Measure=0.7738
Nearest Neighbour: 153.jpg	Nearest Neighbour: 111.jpg
Similarity Measure=0.7736	Similarity Measure=0.7682
Nearest Neighbour: 133.jpg	Nearest Neighbour: 195.jpg
Similarity Measure=0.7732	Similarity Measure=0.7637

Fig 8.3 *– Nearest Neighbours XML Results output rendered as Excel Table*

8.4 Dimensionality Reduction

CNNs can be interpreted as gradually transforming the images into a representation in which the classes are separable by a linear classifier. We can get a rough idea about the topology of this space by embedding images into two dimensions so that their low-dimensional representation has approximately equal distances than their high-dimensional representation. There are many embedding methods that have been developed with the intuition of embedding high-dimensional vectors in a low-dimensional space while preserving the pairwise distances of the points. Among these, TSNE is one of the best-known methods that consistently produces visually pleasing results.

8.5 TSNE (T-Distributed Stochastic Neighbor Embedding)

TSNE is a tool to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. t-SNE has a cost function that is not convex, i.e. with different initializations we can get different results.
It is highly recommended to use another dimensionality reduction method (e.g. PCA for dense data or TruncatedSVD for sparse data) to reduce the number of dimensions to a reasonable amount (e.g. 50) if the number of features is very high. This will suppress some noise and speed up the computation of pairwise distances between samples (van der Maaten's - 2017)

The perplexity is related to the number of nearest neighbors that is used in other manifold learning algorithms. Larger datasets usually require a larger perplexity. The literature recommends we select a value between 5 and 50. Different values can result in significantly different results.

This method notes that the tensor pool_3:0 contains the weights for the penultimate layer of the network. These weights form a 2048 dimensional vector (or list of 2048 numeric units) that's perfect for image similarity computations.

Using t-distributed Stochastic Neighbor Embeddings(TSNE) we can reduce the dimensionality of the feature space, and attempt to identify similarity between classes. Transfer Learning allows us to remove the last softmax layer of the CNN train process, and to use the vectors as feature descriptors. We can apply TSNE which is well integrated with Tensorflow and lends itself to strong visualization possibilities of the reduced feature space.

Dimensionality reduction on the full 60,000 images takes a very long time

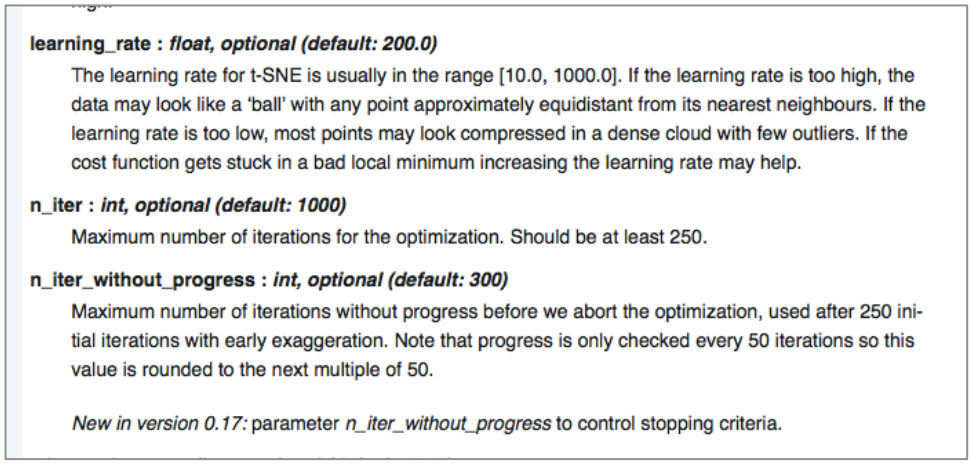
t-SNE to reduce dimensionality down to 2D so we can plot the dataset: Matplotlib's scatter() function to plot a scatterplot, using a different color for each digit:

this plot tells us which numbers are easily distinguishable from the others (e.g., 0s, 6s, and most 8s are rather well separated clusters), and it also tells us which numbers are often hard to distinguish

8.6 TSNE Method Summary

- Represent 3 Dimensional Image Arrary on 2 Dimensional plane
- Important: Perplexity Variable and Iteration Count
- Misreading Pitfalls: https://distill.pub/2016/misread-tsne/
- Quickest and Most Accurate(on our datasets) = PCA & TSNE Pipeline¶
- Different data sets can require different numbers of iterations to converge.
- The most important thing is to iterate until reaching a stable configuration(e.g. 1000 iterations, 5000 = better).
- with perplexity values in the range (5 - 50) suggested by van der Maaten & Hinton, the diagrams show strong clusters

- You cannot see relative sizes of clusters in a t-SNE plot.
- From now on, unless otherwise stated, we'll show results from 5,000 iterations. That's generally enough for convergence in the (relatively small) examples in this essay.



learning_rate : *float, optional (default: 200.0)*

The learning rate for t-SNE is usually in the range [10.0, 1000.0]. If the learning rate is too high, the data may look like a 'ball' with any point approximately equidistant from its nearest neighbours. If the learning rate is too low, most points may look compressed in a dense cloud with few outliers. If the cost function gets stuck in a bad local minimum increasing the learning rate may help.

n_iter : *int, optional (default: 1000)*

Maximum number of iterations for the optimization. Should be at least 250.

n_iter_without_progress : *int, optional (default: 300)*

Maximum number of iterations without progress before we abort the optimization, used after 250 initial iterations with early exaggeration. Note that progress is only checked every 50 iterations so this value is rounded to the next multiple of 50.

New in version 0.17: parameter *n_iter_without_progress* to control stopping criteria.

Fig 8.4 – *Relevant SciKIt Learn TSNE Documentation*

8.5 TSNE Analysis of the LDD CNN Classification Model

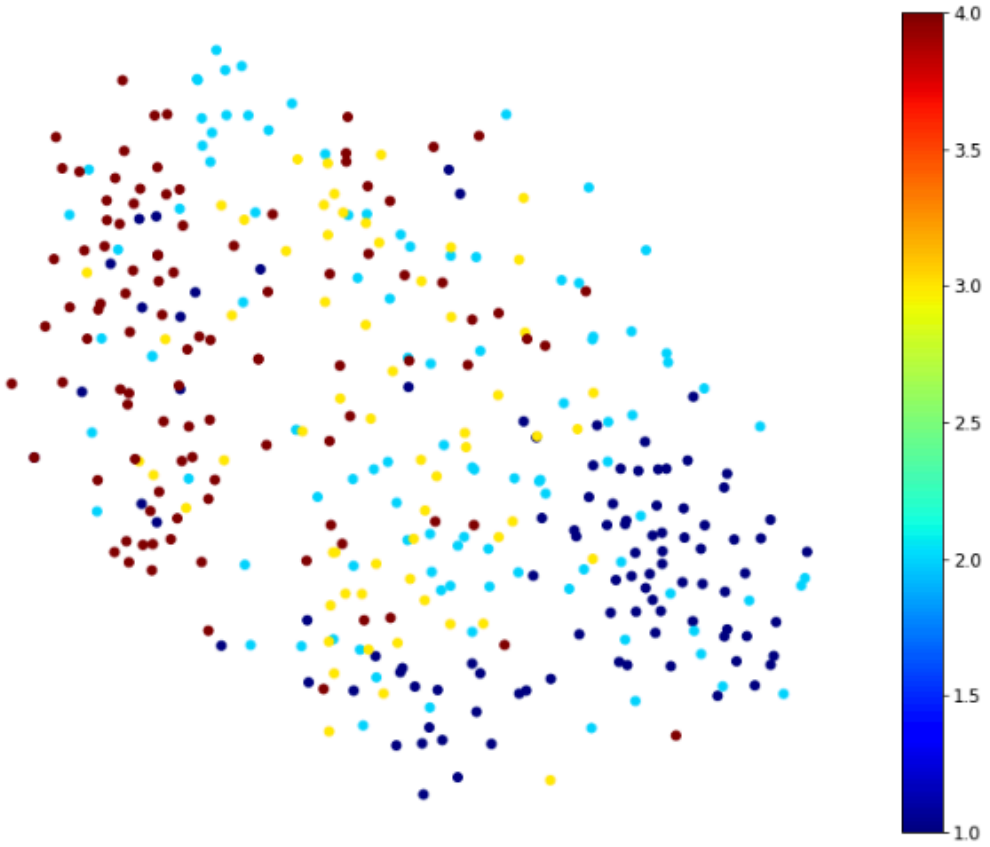


Fig 8.5 *– Using Image Vectors from our best of class n= 4 Label run, we can see some class differneation in labels 1 and 4*

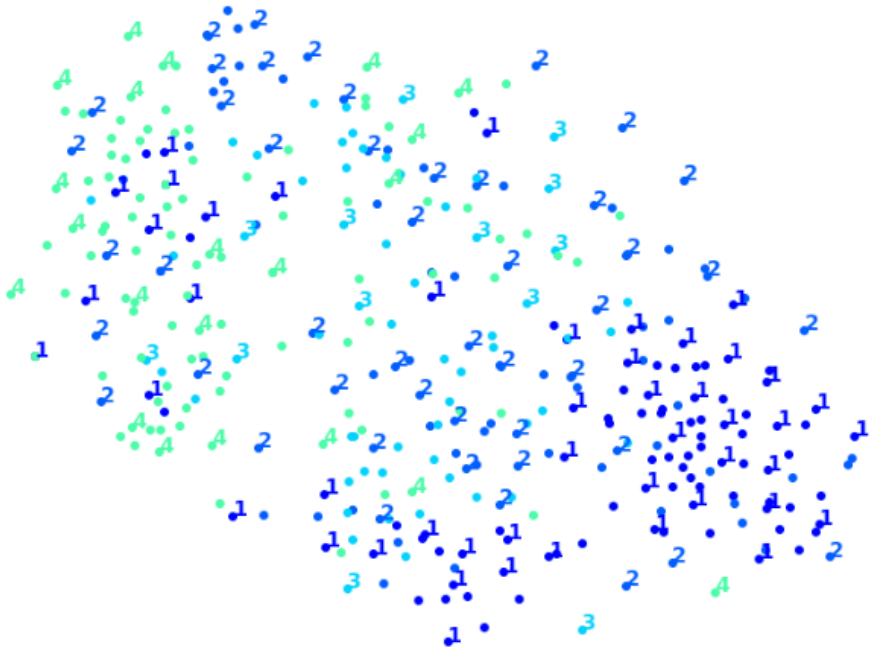


Fig 8.6 – *Subset TSNE Attempt to Clean Up Weaker Clusters(oiginally 2, 3 = House and Industrial)*

Where digit images are provided, they are plotted instead. This implementation was inspired from one of Scikit-Learn's examples (plot_lle_digits, based on a different digit dataset).

```
plot_digits(X_reduced, y, images=img_np, figsize=(35, 25))
```

Fig 8.7 *– Class labels replace with Photographic thumbnails for each Image Vectors*

Fig 8.8 *– Varying Image Size for the Embeddings Plot*

With the benefit of Large screen to view the plot in detail we can see some visual distinction emerging to the left of the plot. Here Modern High-Rise Flat developments appear to be responding to an Unsupervised Image Classification task.

```
plot_digits(X_subset_reduced, y_subset, images=img_np, figsize=(22, 22))
```

Fig 8.9 *– Varying Image Size for the Embeddings Plot on additional Label Runs*

8.6 Building an interactive TSNE Cadastral Explorer Interface with Three JS

Our image plots can become cluttered and hard to assimilate quite quickly. Using the Web GL enabled functionality of three JS we can port the same functionality into an interactive application that allows greater user control to explore and understand the spatial relationship between the image similar feature rendition. For this study we use the git hub ready module developed by Yale University Digital Humanities Lab ‘s Douglas Duhaime

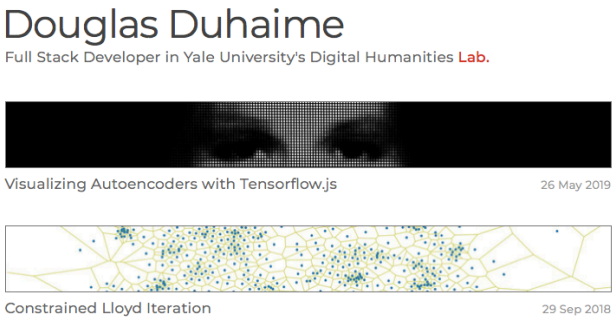


Fig 8.10 *– Duhaime Digital Humanities Code Repository*

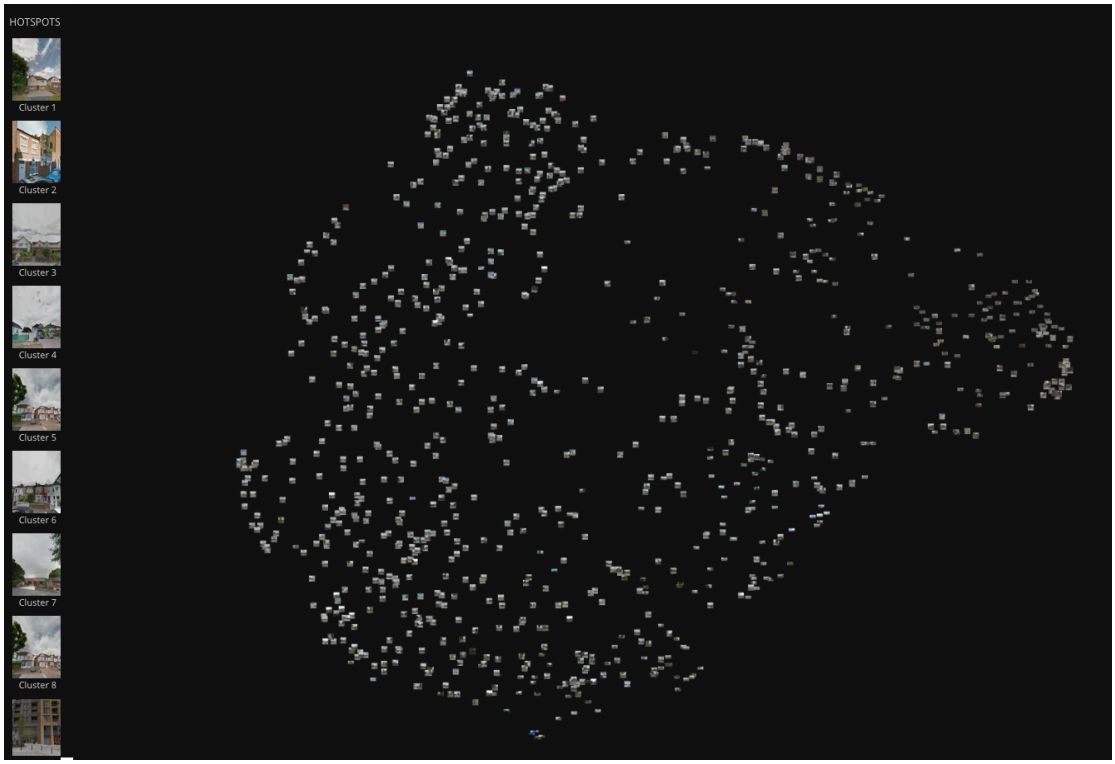
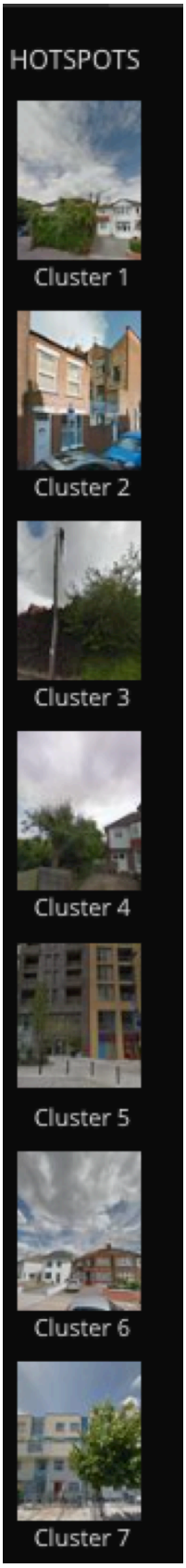


Fig 8.11 *– The study’s CNN Classification Task Model of the London Developmnet Database rendered as an interactive Three JS Image Viewer.*



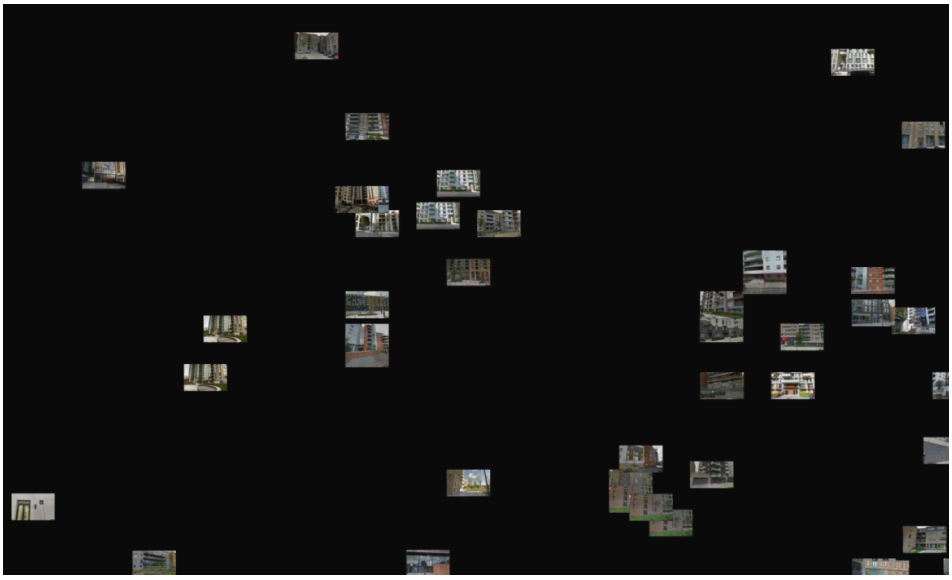


Fig 8.12 *– The TSNE Analysis as applied here identifies 9 Clusters. At first Glance it is not apparent how strong our Unsupervised Label Groupings have been..*

As we scroll and zoom into the identified cluster patches, the class similarity of building form and grouping of architectural feature and detail becomes apparent.

Cluster 9 (seen in detail and located the far right to the overall group conglomeration) can be seen to group high density London New Build Property types.

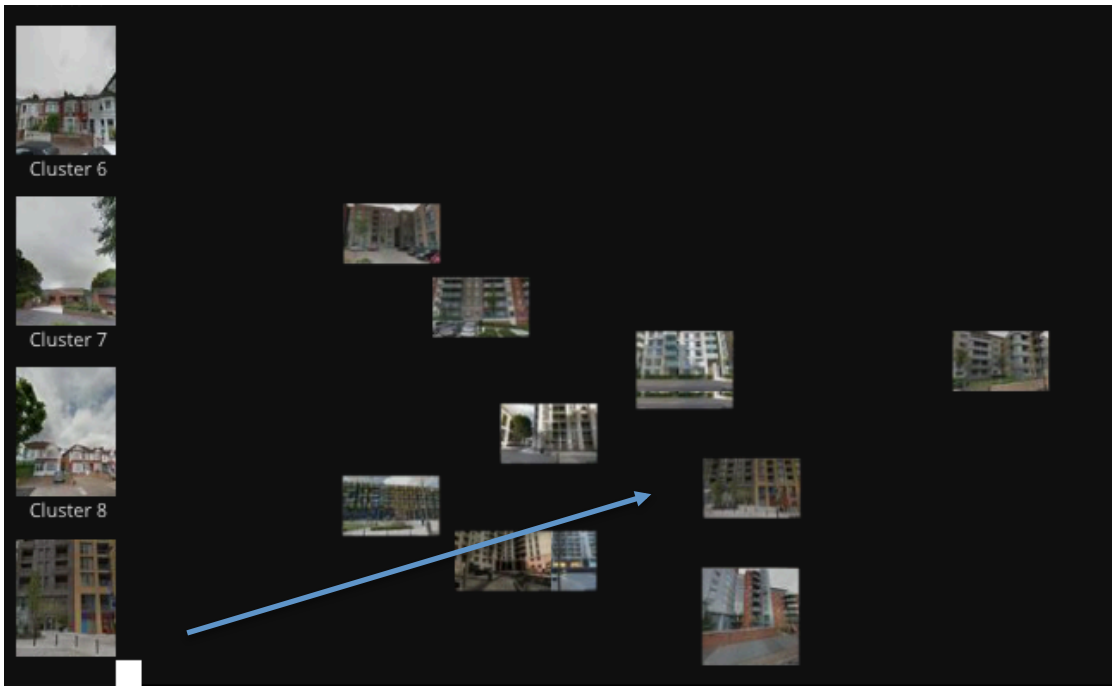


Fig 8.13 *– Cluster 9 Close Up*

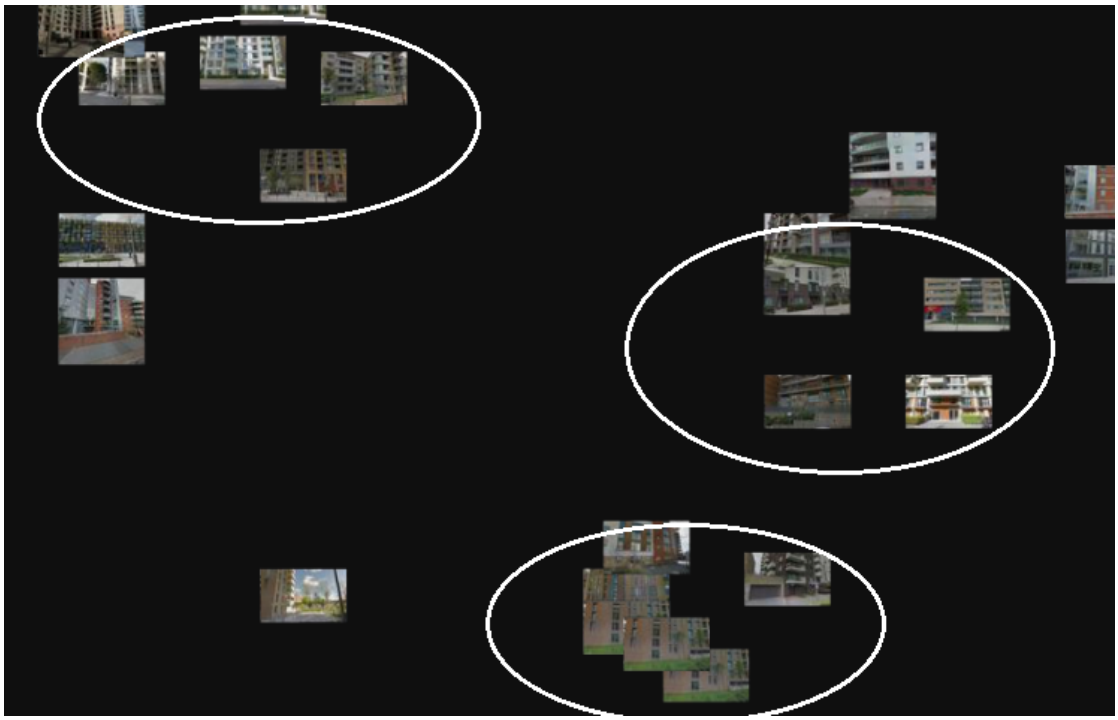


Fig 8.14 *– Here we identify 3 Clusters for further examination. These are shown in detail overleaf.*

Fig 8.15 *– Moving anti clockwise though the clusters idnetified in fig 8.14: Further examination reveals self similar propertiesin t the groupings but also highlights the presence of duplicate images reflecting parent address references*

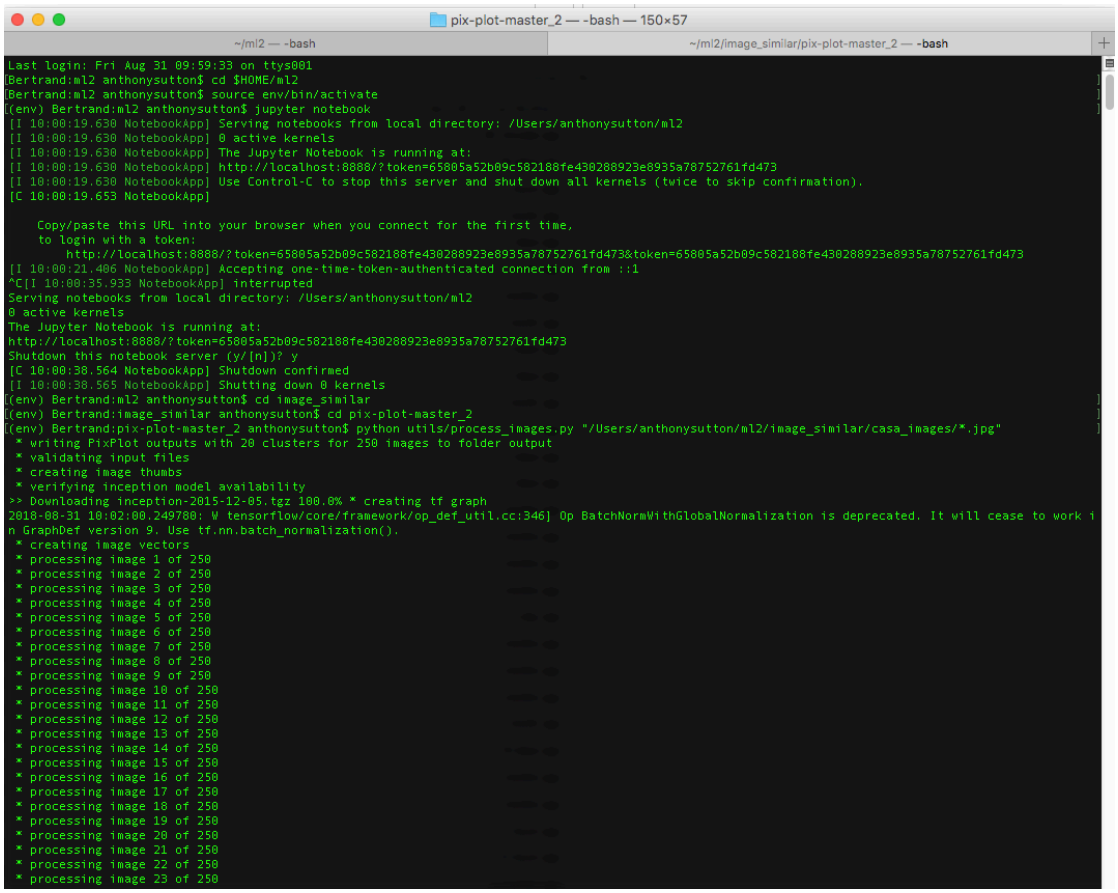


Fig 8.16 *– Genreration of the Image Vectors to be reused*

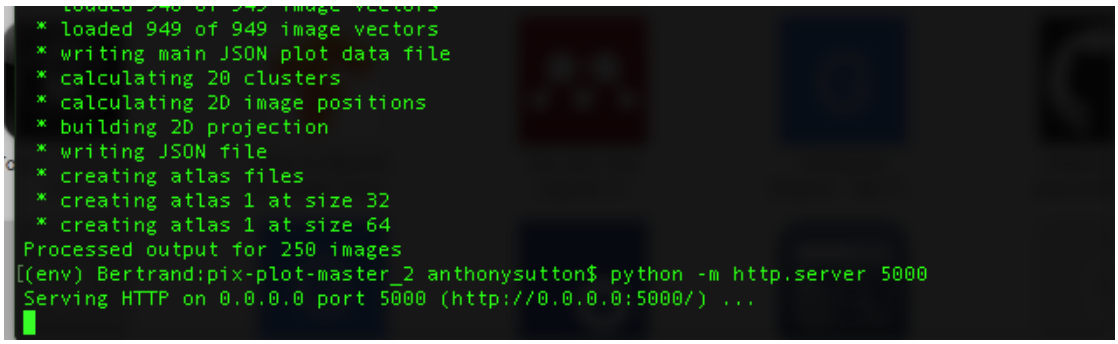


Fig 8.17 *– Running a local web server to host the Three JS Web GL Package.*

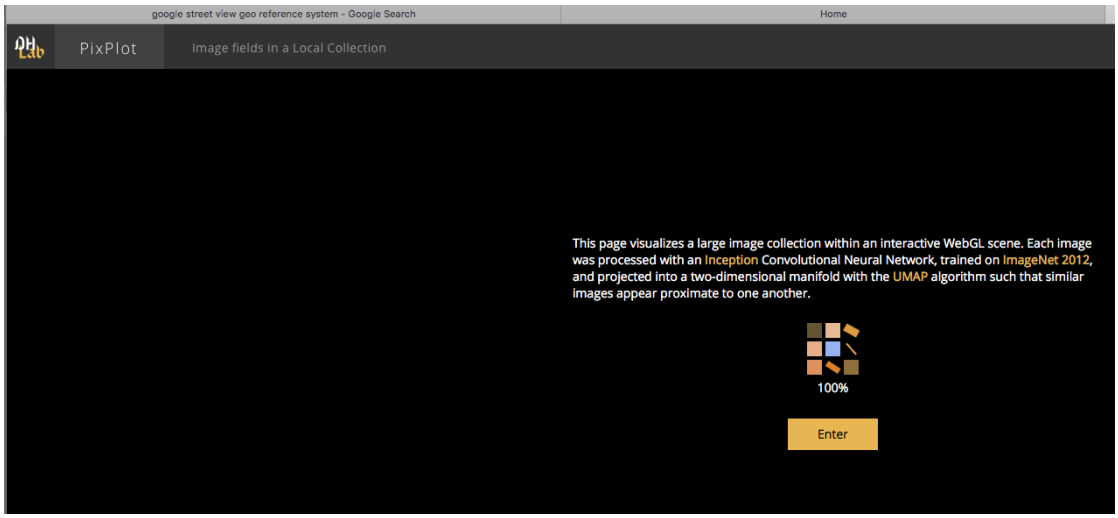


Fig 8.18 *– The Splash Page greets the Viewer while the TSNE Cluster analysis runs in the backgground*

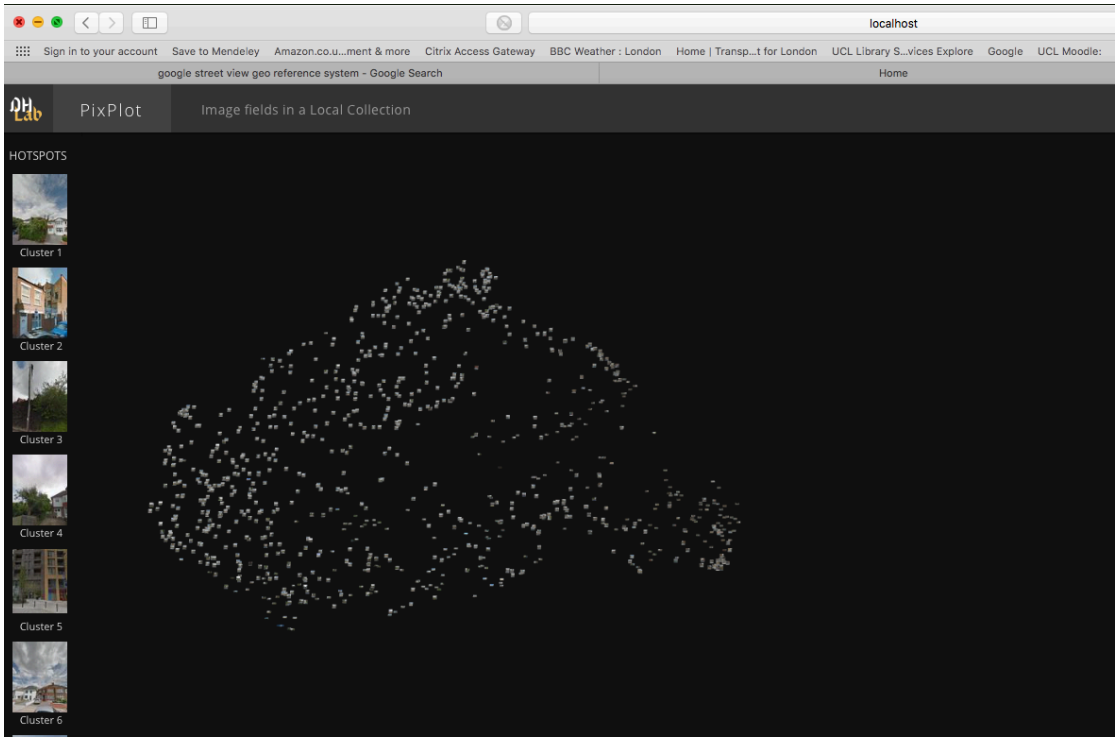


Fig 8.19*– The Complete Label Run, fully zoomed out*

Image Analysis Workflow Summary

As the above walk through demonstrates Nearest Neighbour analysis, in this instance, provides us with poor results whilst the TSNE Image Similar pipeline produces a strong demarcation of the Modern Flat Building Type from our Classification Model

The accompanying building types from our chosen sets of Label Runs had less success in being clearly identified. This again confirms our findings from the previous workflow stages, and shows the model to comprise of a predominance mix of opaque building types from a Machine Learning perspective .

Whilst this workflow stage provides us with more robust(at least in terms of not suffering from the model overfit issue) and more visually satisfying result than the initial supervised CNN Training and Classification Stages, the analytical results of this workflow are still inconclusive in what observations relating to building form we can derive from the unsupervised categories that we are looking at.

It is therefore more practical to focus on the complementary value we can derive and apply to the exploratory themes of our study. This workflow stage demonstrates the value the TSNE embeddings exploration might provide us as an immersive visual aid to the wider technological narrative of imagining real time data driven marketplaces and the increased control, public ownership and cognitive assimilation of large scale aggregate data and the frequently ephemeral and complex entities that they represent, that this approach promises..

From a Design Thinking point of view, we also see the emergence of the possibility or idea of aesthetic architectural and urban design decisions that could be made from within a larger feature space. If we can expand the boundaries that currently curtail our feelings of what

This idea of architectural and urban design decisions made not just within the local o but firmly on a global scale resonates with Christopher Alexander's Pattern Design Thinking, where a visionary good urban design is made in a hierarchical and connected context, and encourages the urban design dialogue toward further public enfranchisement beyond the merely physical inanimate impact of buildings, cities and place.