Interactive Visualization Spring 2020 (RPI)

Photospheres

Frederick Choi, Dev Rangarajan

April 30, 2020

1 Introduction

A common issue that novice designers run into is trying to look for the right image that should go on a poster or graphic. The first step here is often to go on Google and search for an image. But perhaps simply a search for "beach party" is not enough, and the designer wants something that fits better aesthetically. However, a novice designer may have a lack of vocabulary to describe image they want. Perhaps Google images does not have the exact tags they are looking for. The question becomes: how can we search a large image database without the use of text or tags?

In this case, some way of searching for images that is based purely on visual attributes can be of help. In this paper we introduce Photospheres, a text-free image searching tool that organizes images into hierarchical clusters for fast browsing.

Some other applications can include browsing personal pictures to build a scrapbook, slideshow, or collage. As a commercial tool, it can be integrated into OneDrive or Google Photos.

It could also be used to find a good desktop background or theme. It could especially be help for searching type fonts, since finding a type font of a specific aesthetically quality is difficult since there are no semantic tags (unlike searching for a "beach party" image), and many of the distinguishing descriptors (kern, serifs, etc.) may require specialized knowledge of type fonts to understand.

2 Background & Related Works

other tools, how do we compare, where did get our inspiration from, cover all the references we have

As mentioned above, Google Photos is a common first step for most people looking for an image. Adobe Stock is another tool created specifically for designers that can be used to find images. However, both of these tools require the user to use keywords or some refine based on categories/subcategories with text. This project investigates the viability of an image search algorithm that does not rely on text.

The approach outlines in this paper focuses mostly on clustering. Clustering is used in many disciplines for reducing the complexity of a data set as well as finding specific datum. For example, Yin et al.[1] discuss clustering student programming assignments to reduce the load on graders, since similar assignments should get similar grades and probably similar feedback. In that paper they discuss using t-SNE by Maaten et al[2] to visualize clusters, which is used in this paper.

Some hierarchical clustering techniques were borrowed from Gifford.[3]

3 Algorithm

3.1 Overview

Design Goals. Photospheres is designed around the following goals:

1. The user should be able to find a more satisfactory image than with just text-based searching/refinement
2. The user should be able to find a satisfactory image quickly

In addition, the tool should work with any image set, and not be specialized to any single image set or category of images. Overall satisfaction with the tool itself is important. The tool should not be frustrating to use or difficult to learn how to use. While "satisfactory" is not a concrete quantity, a user test subject should be able to provide some feedback on how satisfied they are with an image they found. A scale of 1-10 could suffice for a rough estimate. Meanwhile, the time it takes to find a satisfactory image can be directly measured.

The hypothesis is that hierarchical clustering paired with fast navigation and informative previews can rival image searching with text.

The visualizations in this paper are implemented with d3.[4]

Clustering. A hierarchical clustering of the images is created in a top-down recursive manner. The images are clustered using the algorithms described below, and images within each cluster are further separated into sub-clusters recursively until the depth limit is reached or the cluster is too small.

The idea is to break down a large set of images into a very small (7 in this prototype) number of clusters. Ideally, the user would which one or two clusters might contain the image they are looking for, which cuts down the search space by an order of magnitude. Then, within these clusters, the user should be able to again identify which clusters are relevant and refine the search further.

An analogy is looking through folders on a filesystem to find a specific file. Folders within folders organize the data into groups, and the user identifies based on the name of the folder and a preview of its contents (on some operating systems) which groups may contain the desired file. In that way, a relevant file can be found very quickly without even using a search bar. For the case of image searching, a good clustering algorithm must be used to organize the images in to sensible groups, and a good preview to efficiently communicate to the user the images within that group.

Visualization. These clusters are visualized using a couple techniques that embody a few design principles. The clusters are clearly distinguishable, since they are the primary method of navigation through the image database. Clicking on a cluster reveals in more detail its sub-clusters. This is fast in order to minimize the time lost and mental cost of entering a wrong cluster. This is also done in a way to ensure coherence between the two views and to minimize disorientation. Hovering over a cluster pops up a preview window that to conveys to the user an idea of what images are contained in a cluster before they click into it.

3.2 Clustering

The core of the tool presented in this paper is hierarchical clustering. A few different methods of clustering, including different feature transforms and clustering algorithms were investigated. The techniques presented here should be extendable to any image set.

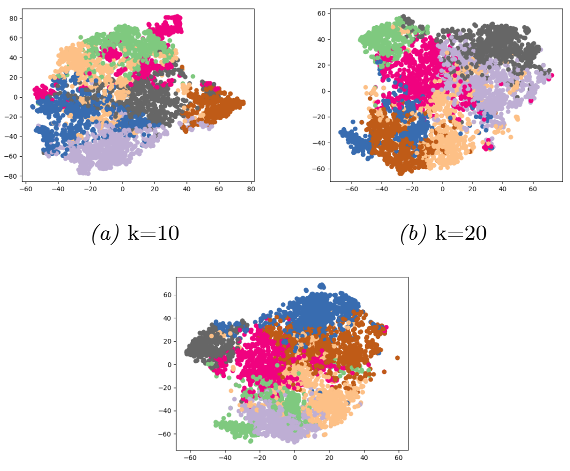
3.3 Feature Transform

Principle Component Analysis. PCA can used to extract a basic feature set from the images. After converting the images into vectors where each component is a single RGB channel of a single pixel, PCA can be used to extract the pixels and color channels which explain the most variance in the images. The primary advantage here is its simplicity. However, this method suffers from its high dependence on locality. For example, a house on the left side of an image will produce a vector that is dissimilar to a vector produced by an image with a house on the right side.

The prototype presented in this paper uses an implementation of PCA from the Scikit Learn[5] package for Python that uses the probabilistic PCA model from Tipping et al.[6] The number of components constrained by computation time and refined through rough visual analysis/agreement with the *t*-SNE based visualization. However, more robust methods such as the elbow method[7] are likely to yield better results.

Figure 1 shows comparisons of *k*-means clusters and *t*-SNE. From these plots, it seems that the clusters assigned by k-means with a 20-component PCA agrees reasonably well with the embedding from *t*-SNE. Points that belong to the same cluster are embedded close to each other. Having fewer components also makes for faster computation time. Thus for the purposes of the prototype presented in this paper, a 20-component PCA was used as a feature transform.

Image Hashing. Image hashing is a hashing algorithm that takes a raw image, and outputs a hashed value for that image. Image hashing is a technique originally developed for copyright detection.[8] The hashing algorithm differs from a typical hashing algorithm; its goal is to produce similar hashes for similar images. Our hypothesis was that we could cluster using the hamming distance (the number of different characters between two strings) to cluster similar hashes and thus similar images together. However, k-means does not work with non euclidean distance metrics, so we would need to implement a different clustering algorithm. Agglomerative clustering is a more generic clustering algorithm which works with hamming distances. Agglomerative clustering works by putting all elements into clusters, and then grouping the closest clusters together until the all elements are contained in the same super cluster.



(a) k=10 *(b)* k=20 *(c)* k=40

Figure 1: Comparison of k-means clustering t-SNE in 2 dimensions. Each point represents an image. The top k eigenvectors from the image set were used to compute the clustering and t-SNE. The color represents cluster membership while points area plotted according to t-SNE.

Keras. Keras is a Python API for deep learning. It allows for simplified use of other neural network packages such as TensorFlow (used for this project). Keras has some pre-trained neural networks for image classification. The one we used in this project was VGG-16 (Visual Geometry Group 16 Layer Network[9]). The entire dataset was run through this model, and then a feature set was returned. We then used k-means to cluster the features together and get the final output. These features are determined by the neural network, and are represented as vectors. This has less of a dependence on the position of objects within an image, and should thus be better suited to this application.

3.4 Recursive Clustering

Let I be a set of images. Using one of the feature transforms discussed above, a feature vector X is computed from the images. A hierarchy of clusters is formed based on these transformed values are follows.

Let a cluster be denoted by (I, C) where I is the set of images within that cluster and C is the set of sub-clusters/child clusters. Algorithm 1 outlines the procedure for computing a recursive k-means clustering, though k-means can be substituted for any clustering algorithm.

Algorithm 1 Recursive K-means

```
$X \leftarrow$ Feature-transformed input data
$I \leftarrow$ Raw images
$k \leftarrow$ # clusters per branch
$split\_threshold$, $depth\_limit \leftarrow$ Recursion limits
1: function CLUSTER($X, I$)
2:     export mean of $I$ as preview
3:     if $depth\_limit \leq 0$ or # points in $X < split\_threshold$ then
4:         return cluster $(I, \varnothing)$
5:     else
6:         Run k-means to partition $X$ into clusters $X'_i$ for $0 \leq i < k$
7:         Partition $I$ into $I'_i$ corresponding to $X'_i$
8:         C $\leftarrow$ { CLUSTER($X'_i$, $I'_i$, $split\_threshold$, $depth\_limit - 1$) for $0 \leq i < k$ }
9:         return $(I, C)$
```

3.5 Visualization

Once the images have been processed into clusters, they must be presented to the user in a useful way. The following principles guided the design of the interface:

1. Navigation through the clusters should be fast to minimize the cost of a user navigating to the "wrong" cluster
2. Clusters should be sensible to humans
3. Previews should be representative and the user should be able to guess what images a cluster would contain

Interactions. Keeping the design goals in mind, the visualizations presented in this paper implement the following interactions.

1. Clicking on a cluster reveals more detail of the cluster and its sub-clusters while diverting the attention from other clusters
2. Hovering over a cluster shows a preview of the cluster, indicating what images the cluster likely contains.

In addition, the clusters are clearly distinguishable, since they are the primary method of navigation through the image database. Any transitions are animated since two views of the same data can change drastically which is potentially disorienting if a sudden transitions were used instead. Hovering over a cluster pops up a relatively small preview window that is close to the mouse cursor in order to minimize

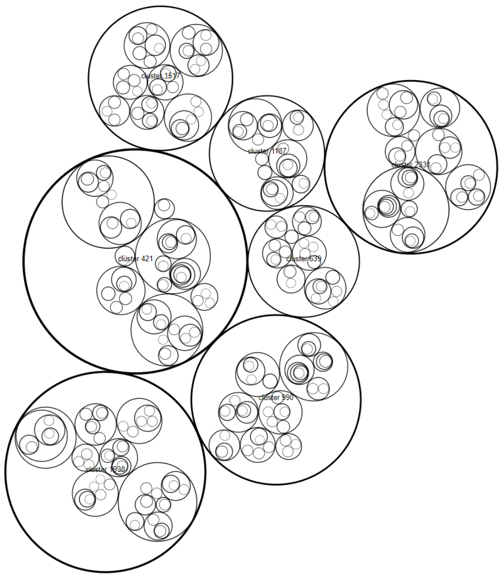
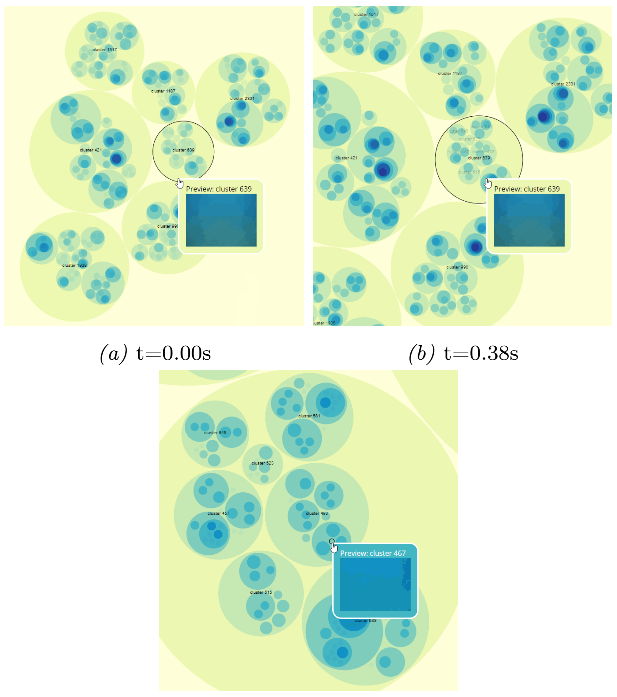


Figure 2: A screenshot of the circle packing visualization with only outlines rendered to accentuate the tightly packed nature of the circles



(a) t=0.00s *(b)* t=0.38s

(c) t=0.75s

Figure 3: Frames of the cluster zooming animation for the circle packing visualization

the area of the base visualization it occludes while placing it close to where the user's gaze is likely already focused.

3.6 Circle Packing

In this visualization, all the clusters and sub-clusters are represented as circles. Sub-clusters are packed within its parent cluster such that it fits entirely within its parent cluster while remaining tangent to its sibling clusters and/or its parent cluster. Figure 2 illustrates this tightly packed layout of the circles.

The "empty" space within a cluster but outside its sub-clusters opens up an opportunity for more interaction. Hovering over this empty space pops up a small preview modal near the user's cursor containing a preview of that cluster. The preview is simply a pixel-to-pixel average of the images within the cluster, but better summarizing techniques can be investigated. After all, the point is to convey to the user what images are likely contained within the cluster before they have to search through it.

Figure 3 contains a screenshot of the preview modal as the user hovers over a cluster. It also illustrates the zooming feature. Clicking a cluster will trigger an animation that focuses in on that cluster as illustrated. Note that the sub-clusters come into view with more detail in frame (c), whereas some of the finer details are absent in frame (a). The other clusters are almost entirely cropped out. This keeps with the idea of diverting the attention away from the other clusters and focusing on the one the user selects.

Figure 4 displays a typical color scheme used in the prototype presented in this paper. The clusters are colored by depth, with darker circles corresponding to deeper sub-clusters. The color scheme in this figure is a sequential color scheme from Color Brewer.[10] At the lowest level, the images are represented as white circles to contrast from the saturated cluster colors, and previews simply show the image itself as seen in figure 5. Additionally, the name of the image appears on the preview for retrieval outside the tool.

3.7 *t*-SNE based layout

t-SNE as described by Maatan et al.[2] is an algorithm for embedding high dimensional data into low dimensional space. However, this algorithm runs slowly on high dimensional input data, so features are extracted first extracted from the images (recall X from section 3.3). For the prototype in this paper, an implementation from Scikit Learn[5] was used.

In contrast to the circle-packing layout, no clusters are drawn directly. Instead, all the images are repre-

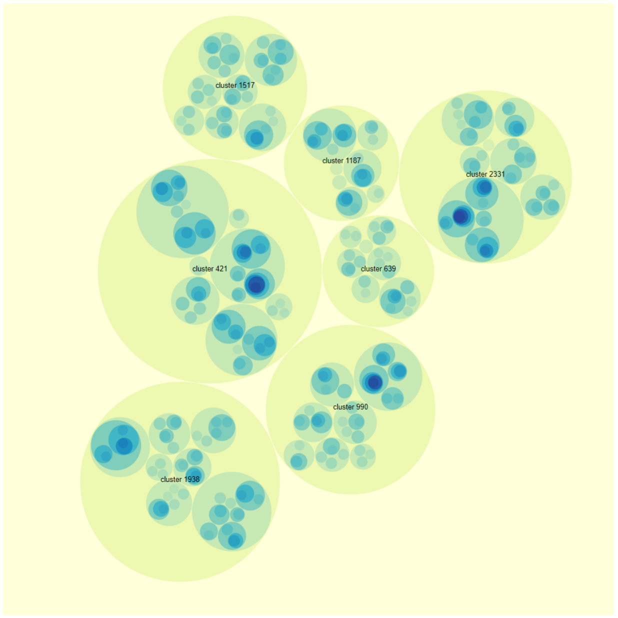


Figure 4: A screenshot of the circle packing visualization with colors

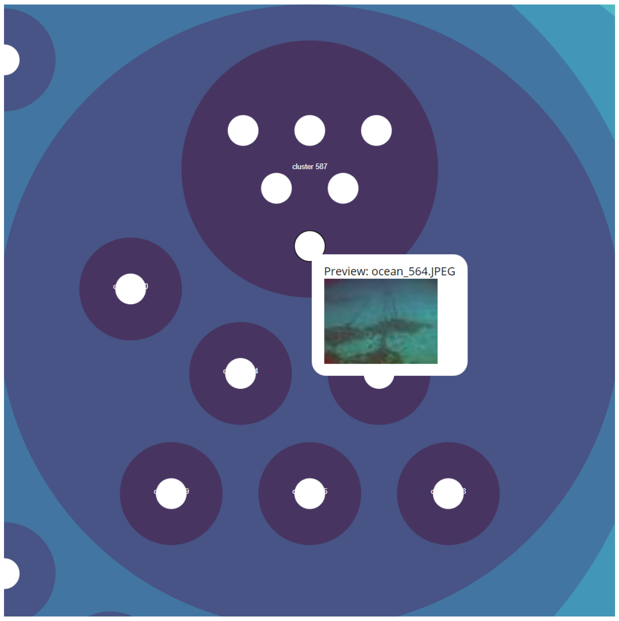


Figure 5: A screenshot of the circle packing visualization at the image level

Algorithm 2 Recursive K-means with t-SNE

```
$X \leftarrow$ Feature-transformed input data
$I \leftarrow$ Raw images
$k \leftarrow$ # clusters per branch
$split\_threshold$, $depth\_limit \leftarrow$ Recursion limits
1:  function CLUSTER($X, I$)
2:      export mean of $I$ as preview
3:      if $depth\_limit \leq 0$ or # points in $X < split\_threshold$ then
4:          return cluster ($I$, $\varnothing$)
5:      else
6:          Run k-means to partition $X$ into clusters $X'_i$ for $0 \leq i < k$
7:          Partition $I$ into $I'_i$ corresponding to $X'_i$
8:          C $\leftarrow$ { CLUSTER($X'_i$, $I'_i$, $split\_threshold$, $depth\_limit - 1$) for $0 \leq i < k$ }
9:          E $\leftarrow$ t-SNE embedded coordinates of points in $X$
10:         return $(I, E, C)$
```

sented by relatively small circles whose positions are determined by *t*-SNE and are colored by cluster (see figure 6). *t*-SNE positions are computed within each cluster, since each cluster represents a smaller data set. *t*-SNE on a smaller data set can yield better results as fewer points means it is easier to keep dissimilar points far apart. Algorithm 2 illustrates how this fits into the implementation.

Note that we lose the information about the structure of the hierarchy, and we lose the ability to "skip" steps and preview a deeper sub-cluster. However, we gain the ability to judge the spread of a cluster based on how spread out it is in the visualization. In addition, points that are close by are likely to represent related/visually similar images.

When the user hovers over a circle, the preview shows both the image the circle represents as well as a preview of the cluster. This is illustrated in figure 7. When the user clicks on a circle, the cluster it belongs to comes into focus. The circles move to new positions as determined by *t*-SNE within that cluster, and are recolored according to subcluster. The other circles fade out. Figure 8 contains frames of this interaction.

4 Results

At the beginning of this project, we defined a set of principles that our visualization would follow, and also a set of potential tasks that it would need to han-

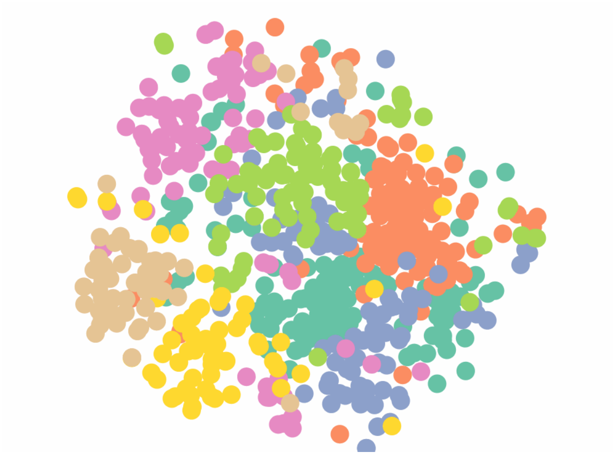


Figure 6: A screenshot of the t-SNE based visualization

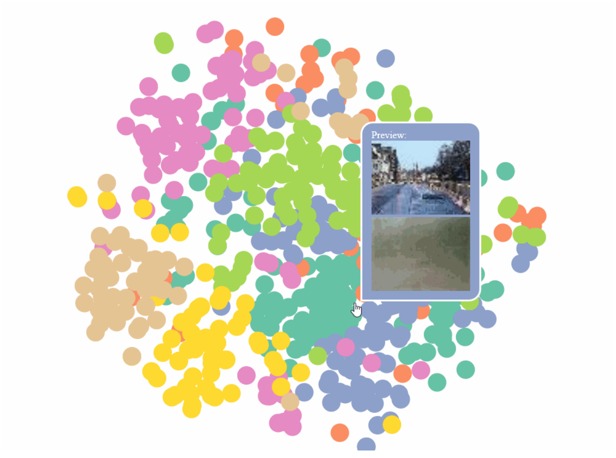
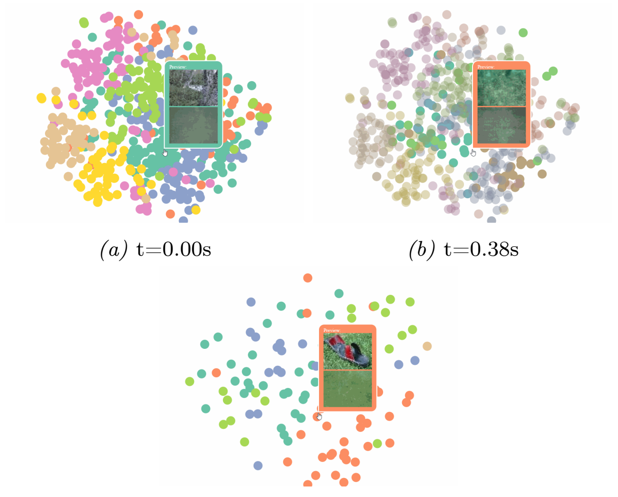


Figure 7: A screenshot of the hovering interaction in the t-SNE based visualization. The upper image in the preview is the image corresponding to the circle under the cursor. The lower image is a preview of the cluster (in this case, the teal cluster).



(a) t=0.00s *(b)* t=0.38s *(c)* t=0.75s

Figure 8: Frames of the cluster zooming animation for the circle packing visualization

dle. These are stated in section 3.1 and 3.5 of this paper. The first principle about quick cluster navigation is accomplished with both visualization, and is mostly a function of using d3. The second principle about cluster sensibility is addressed by the specific preprocessing results described later in this section. The final principle about representative previews is the same for all visualizations/algorithms. Currently, all previews are the pixel by pixel mean of the images in the cluster. This is good at showing the potential colors, but bad at showing the potential features. It is also useless after a certain number of clusters; it tends to converge.

Due to a lack of resources, a formal user study has not been performed. As such, we can only make hypotheses about the accomplishment of our design goals. Photospheres is an inherently subjective tool, meant to help people search for things without words, so success can only be determined if the user finds an image that they feel fits their needs. The goal is not to be objectively perfect, but to allow for lateral discovery and expose users to images they may not have originally thought would work for their purpose. We can conclude that Photospheres is capable of this, in that it is possible to find a wide variety of photos, however it is not easy to find a specific photo. The original matching test where a user is given a photo and told to find it in the interface is not practical at this time. We believe that the current implementation of photospheres meets design goal 2 for some users. Design goal 1 is both hard to achieve and hard to prove, especially because our target audience is used to the text based search paradigm. At this time we are unable to make a strong conclusion about whether or not we met this goal.

Both 20-PCA and Keras based K-means clustering showed a lot of initial promise. These preprocessing steps result in clusters that largely abide by the principles laid out at the start of the project. It is difficult to quantify their success, because if we had a metric for how much sense the cluster makes to a human we would be able to use that to create perfect clusters. As such, it is impossible to prove that all clusters are good and that the visualization could not be better. However, we have not seen any clusters that are indisputable violations of the principles we outlaid at the start. Further work and optimization can definitely be done with these algorithms, but we think that they are the clear way forward.

On the other hand, the implementations of image hashing we tried were unsuccessful. The core problem is that image hashing is a technique that's mostly used for detecting duplicate images, or incredibly similar images (think watermarks, single pixel changes, scaling up or down, etc.) Most image hashing algorithms start by converting the image to binary or gray-scale, and then performing the hash. This increases performance and robustness for the intended use of image hashing, but doesn't work for clustering in the way we hoped it would. When run on this large set of low resolution images, image hashing puts texturally similar images together, even if they are completely different. There are some more advanced algorithms that claim to take color into account, but we did not have the time to implement them.

5 Discussion & Future Work

From our presentation it seems that our tool has promise. People found it unique and interesting, and some found it fun to use. As an image searching tool, much more work needs to be done before it becomes viable, let alone compete with Google Images.

Regarding cluster previews, simply taking the pixel-to-pixel average yields poor previews. This average usually turns out "muddy," and does not actually represent an actual image contained within the cluster. Better summarizing techniques, possibly with multiple preview images, can be investigated and are likely to improve the tool.

In terms of efficiency, this prototype can do better. It currently takes 2min to run the clustering on 4000 images on a 3Ghz processor. The ability to support a large number of images is important, especially since an image set as small as 4000 is unlikely to contain anything relevant. It would be interesting to adapt online clustering algorithms to this purpose. That way the tool could be run as a service, with images continually being added as new ones become available online.

The framerate of the visualization also drops as low as 4 frames per second for the largest clusters. This is primarily due to the large number of shapes that have to be animated and rendered. Perhaps instead of using d3, it would be worth drawing through the GPU instead using a library such as three.js.

Author Credits. Choi: Provided initial project (circle packing, k-means). Investigated t-SNE based visualization

Rangarajan: Investigated Image Hash based clustering and Keras based clustering.

6 References

[1] H. Yin, J. Moghadam, and A. Fox, "Clustering student programming assignments to multiply instructor leverage," in *Proceedings of the Second (2015) ACM Conference on Learning@ Scale*, pp. 367–372, 2015.

[2] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[3] H. Gifford, "Hierarchical k-means for unsupervised learning," 2016.

[4] M. Bostock, V. Ogievetsky, and J. Heer, "D3 data-driven documents," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, p. 2301–2309, Dec. 2011.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[6] M. E. Tipping and C. M. Bishop, "Mixtures of probabilistic principal component analyzers," *Neural computation*, vol. 11, no. 2, pp. 443–482, 1999.

[7] P. Bholowalia and A. Kumar, "Ebk-means: A clustering technique based on elbow method and

k-means in wsn," *International Journal of Computer Applications*, vol. 105, no. 9, 2014.

[8] R. Venkatesan, S.-M. Koon, M. Jakubowski, and P. Moulin, "Robust image hashing," vol. 3, pp. 664 – 666 vol.3, 02 2000.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[10] M. H. Cynthia Brewer, "Colorbrewer 2.0." http://colorbrewer2.org/ (Accessed 12 February. 2020).