Logo Similarity Project

Context

Logos are instrumental for a company's identity – they're the symbol that customers use to recognize your brand. Ideally, you'll want people to instantly connect the sight of your logo with the memory of what your company does – and, more importantly, how it makes them feel.

Solution parts

The solution to this problem can be broken down into multiple parts:

- Scraping the logos from the domain list
- Filtering them because on some sites our program possibly could have scraped something else (another photo for example)
- Group the logos using multiple criterias
-

Objective / Ideas

Group the logos of companies (obtained from their domain) by similarity from multiple angles and do this with a scalable approach.
A logo's identity and recognizability is based on 2 main things:

- The way it looks overall
- The logo's color palette

So having this in mind I chose to pursue these 2 directions. Grouping the logos based on color and overall look.

Scraping

My first approach was to look at the websites of some domains to understand their structure so i would be able to code up a script that scrapes the logos from them. This solution would be changed iteratively based on what websites it would fail until it is getting a good enough score (over 95% for example), it obtained a good score in the end, but it was quite slow, so i found an api that does exactly this and it runs faster: clearbit api. This api obtained 84% success rate, which was quite low, so i implemented a combination of my solution and this to obtain over 98% success rate on a list of unique domains (the given one had duplicates).
Now, i tried to get an even better success rate by looking at the fails. There were multiple possible reasons for the fails:

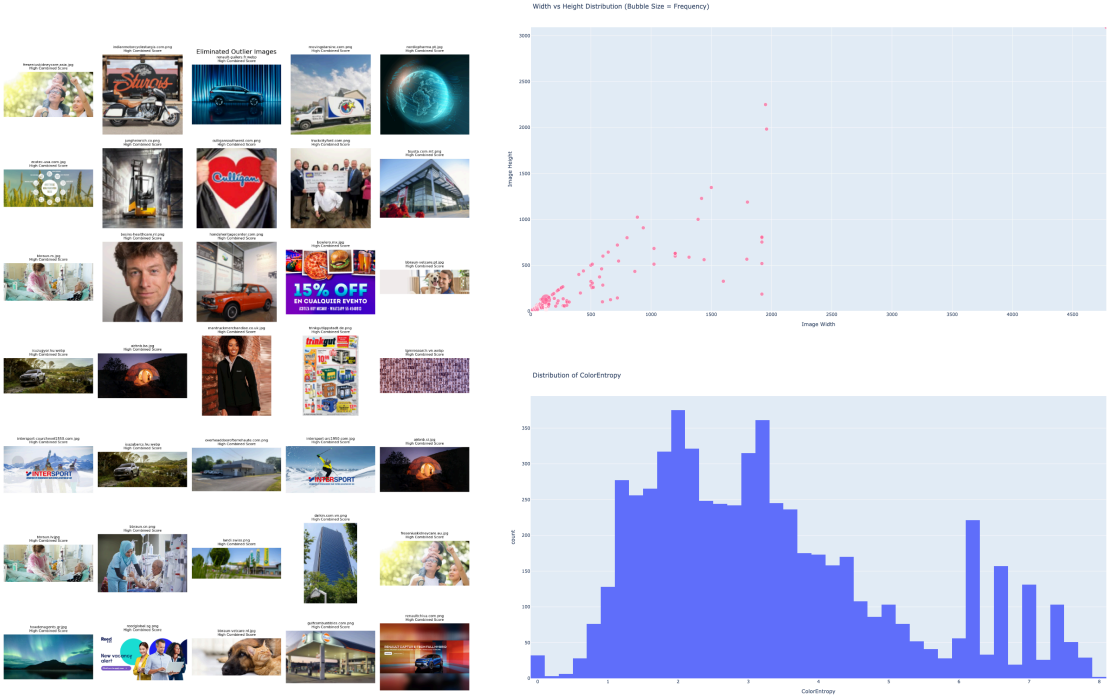
- The website was not working, it did not exist etc..
- It had bot protection

- I was blocked for some time because of too many requests😁

Next i tried to somehow bypass the bot protection and extract the logo using selenium. This worked quite well, managing to extract the majority of the fails from before, but i figured this was not a scalable / good automated solution because it was really slow and it opened up a browser before scraping the logo.
How to make the current scraping method faster? Multithreading. This was an obvious solution for speeding up the scraping process and I applied it.

Finding scraping fails in the extracted logos

Rarely the scraper fails to find the correct logo because of website structure diversity (the scraper could have been further improved, but this would take some time). For this case we would need an automated method to find the outliers. After analyzing the logos a little bit, I found some relevant metrics that would suggest normal images, not logos. I tried multiple metrics like size, edge detection metrics like edge sobel or canny, color entropy. From these size and color entropy seemed the most relevant. Below there are some graphs that show the distribution of the 2 (dimensions up and color entropy the one below it).



The color entropy was more relevant, but the area was also relevant, so i combined the 2 creating a unified score by giving each metric a coefficient (0.1 and 0.9).
Based on the obtained scores, on the left there are the eliminated “logos”. The results of this score are quite impressive, this method only

eliminated 2 or 3 valid logos (of which 2 are really not ordinary), while eliminating multiple outliers. Another possible method for eliminating outliers would be a ML classification model for logos, but this method would need manual labeling and the current method obtains quite good results.

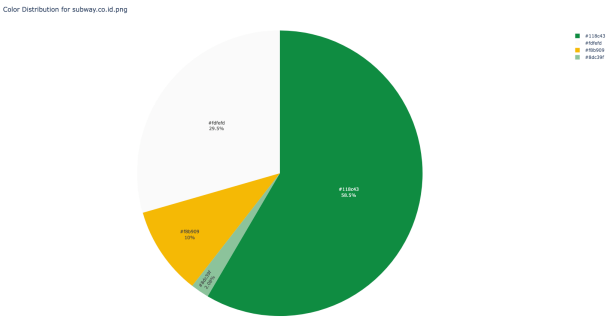
Standardising the logo images

To extract some features, first we need to standardise the logo images. The logos, of course have different formats and dimensions and we want a standard one for every picture. The majority format is png so i chose to convert everything to that, and also the majority dimensions are 128x128 as we could see in the graph above, so i resized everything to that while keeping the proportions because they are quite important to each logo look / identity. I added transparent padding to every logo that had a different aspect ratio than a square.

Feature extraction

I needed to extract features from logos to group them afterwards by color palette and overall look. The features that were extracted were:

1. **Colors and their dominance in the logo**
   This was done by running the kmeans algorithm for each image to find the x most dominant colors. Obviously the color palette for logos in general is made up of a variable number of colors, but for now i chose 4 as the number of colors in the palette because this suited the majority of the list quite well. This number was chosen by trying different K values and looking at the extracted colors. Above is an example of visualization for the subway logo. This logo has 3 basic colors, but as it can be seen the 4th color in the pie chart is within the color theme, so this is ok.



2. **Features extracted using a convnet**
   The overall look features can be extracted using a convolutional network that was pretrained on image classification. These networks aren't perfect for this task, however they can put into numbers quite well the features of the logo images. A better approach could have been extracting the features of the logos with a conv net trained on classifying logos (trained on augmented images of logos with many classes), because this would recognize better

what makes each logo unique, but this would take some time and it is a taks of its own.

3. **The perceptual hash of an image**
   Another method of extracting the overall look of an image into features is perceptual hashing. Perceptual hashing is useful for logo feature extraction because it converts an image into a compact binary signature that captures its essential visual features. This method is robust against small variations like scaling, rotation, or color adjustments, which could be present in our list.

Grouping algorithms

For the grouping I tried multiple algorithms and methods. Here are some of them and the conclusions based on the results:

1. **Kmeans**. This was an obvious trial because this is a clustering problem. However this has some problems that you can spot even before trying it. This ML algorithm requires a predefined K, and you can't try every number K possible until you can visually see the groups are what you want because this is computationally expensive and also it takes a lot of human time for observing the groups. And also, it is not possible to scale this because every time you add a new logo to your list, the process of choosing the right K needs to be repeated. But with all these problems, this algorithm can be used to quickly see some logo groups by color palette or overall look. Here are some results (on the right) of the grouping based on color palette. As you can see this looks quite good overall, but there are some clusters that are not so good, like Cluster 20 which contains multiple color palettes that are sufficiently different.
2. **DBSCAN**. This is definitely the worst clustering algorithm because of how it groups elements. For example, if we have points that form a line in a multidimensional space this will group them as one, but this is not what we want in this case. Let's take for example the color palette classification, many logos have red and blue as the base colors and there are many variations of that. Because of dbscan grouping, all

variations of these colors will be in the same group because there are red logos, more red and some blue logos, half and half and so on, so in the same group we will have all red logos as well as all blue ones. This is not what we want and this also translates to the overall look of the logos, there are little variations in a chain and this breaks the success rate of the dbscan grouping. On the right there are group visualizations of the dbscan. The 1, 2, 3 groups look really good, but Cluster 0 illustrates exactly what i discussed above, variations of the use of blue and red will be in the same group.

3. **Graph based grouping**. Although this is not a natural conclusion given the task, **graph based grouping seems to be the best approach for the logo clustering challenge for color palette and also overall looks**. First we compute the similarity of each logo with each other. This can be done in multiple ways depending on the features extracted:
    - For the image hash the similarity is calculated using hamming distance
    - For the colors and cnn features it is calculated using cosine similarity

   After building the similarity matrix, basically we have some weights for edges in a graph. Then we can use a greedy approach to make the groups, we take a logo and if all elements from a previous group are close enough to it (< threshold) we put it in the same group, if not, we create a new one. There are more precise solutions for this (ex: Bron-Kerbosch algorithm for finding all maximal cliques), but this balances results and computation power needed, also other solutions would not be scalable for a big number of domains. This method is also very versatile: because we calculate a similarity matrix, we can dynamically add more logos and the regrouping is not computationally expensive. We can also make optimisations for this algorithm like removing edges with low similarity to more efficiently store the edges or we can adjust the algorithm with nondeterministic methods to boost efficiency. This method is also modular because we can easily switch or combine metrics for calculating similarity scores.

4. **Hierarchical clustering**. This method can break the logos into a tree of groups, this is an

interesting approach because we can see the structure of the groups better.

5. **Self organizing map**. This method provides a very interesting visualization for the color palette of the logos. However it does not provide good clustering of the logos because it has a predefined number of groups based on the number of cells of the map. A visualization of this is above, on the right and you can observe that the logos are layed out based on colors.

Other Experiments

For this task the most important thing to get a good result is experimentation, so here are some other experiments i made:

- varied the number of colors extracted for the color palette or tried classification based only on top X from a larger number of colors. This gives interesting insights because some groups seem to be better defined than before. On the right are some examples from the groups formed this way.
- applied kmeans or dbscan based on the similarity matrix. This more versatile than the standard version because you can change the similarity_score function
- tried using deltaE_ciede2000() function for similarity. This should give better results than the other similarity metrics for color palette comparison as this function is made to measure the difference of the color for the human eye, but it needs quite large computation resources compared to the euclidean distance metric.
- i also tried many other variations but most are not worth mentioning

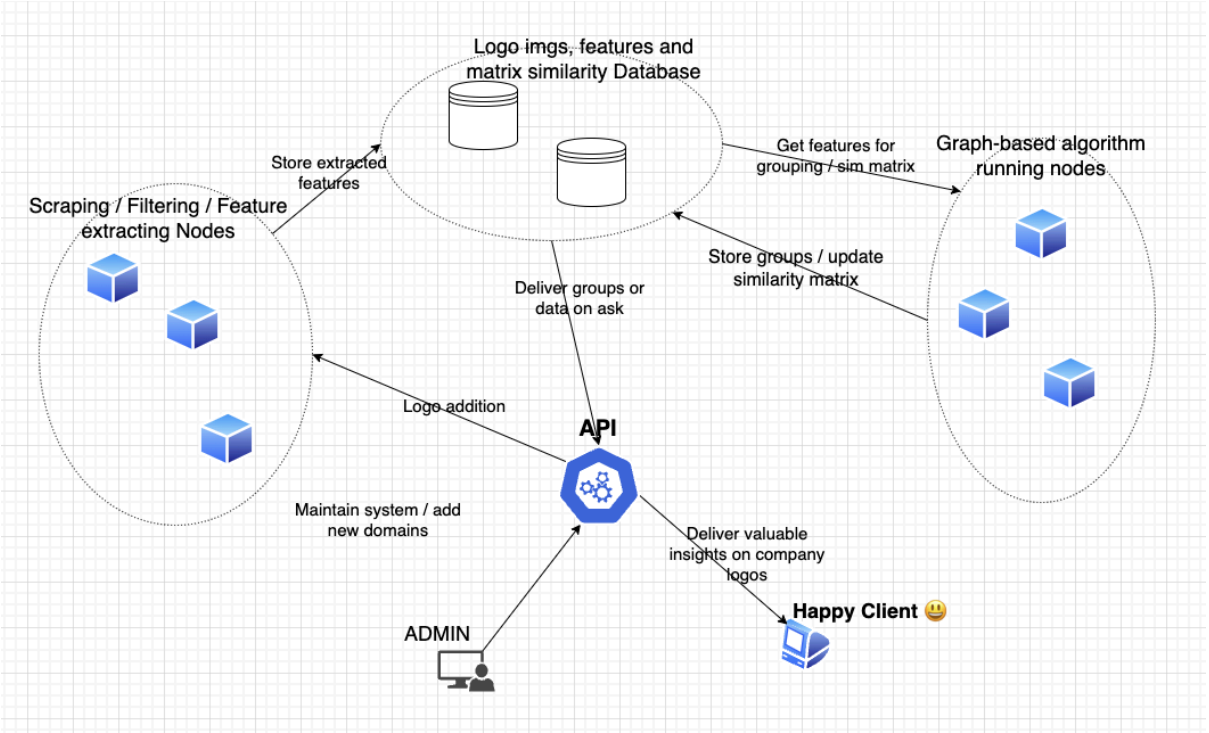
Result project

As the final result, the grouping algorithm is basically an api because this can be easily used further in production and it is easy to scale. It is versatile and modular and it implements most of the ideas / experiments presented above. And it is also very easily configurable, as it can do many types of grouping.

Scalability?

The project is easily scalable as the scraping, filtering and feature extraction can be parallelised for each separate domain, so this type of work can be distributed easily to worker nodes and / or multiple threads. Because of the similarity matrix the grouping does not need to be done on every domain addition, so this computation can be scaled to many more domains. The project architecture can be also easily

integrated into other projects. Here is a proposed architecture for a production implementation of this project.



Other possible additions / Future work

With this project design it is easy to add new features. Here are some ideas:

- adding logos using different methods: giving domain name, just giving any format image, giving a link etc.
- delivering the client a logo report. The client posts their logo into the system and it gives a similarity score to others, delivers valuable insights, gives examples of similar logos and domains
- improving the grouping algorithm
- offering more visualisations