SUPA Experiments: Embedding Visualization

The Problem

We want to address our client's needs to build an embedding visualization tool that allows them to clean/process their image data. The pain points mentioned are as such:

1. Loading times and Slow Processes
2. Limits to the amount of points that can be displayed simultaneously
3. A good GUI

Competitor Analysis: Aquarium's Illume

Their platform allows users to:

1. Have access to an embedding model that is maintained by Illume for embedding generation
2. Process these embeddings
3. Query the embeddings

This is essentially an end-to-end solution:

data ingestion → embedding generation → embedding processing → query embeddings

Aquarium Platform Overview

Based on the video linked above, it would seem that Aquarium is capable of handling extremely large image datasets. By summing up the numbers in the confusion matrix in the video, I estimate roughly 600k images being processed and visualized via Illume.

This sets a benchmark for what we have to hit in terms of performance and data capacity. It's important to note that since Illume is closing, we were unable to get access to a demo. Based on the platform overview video, we only know it supports 2D visualizations and no further comments can be made about whether or not 3D visualizations are supported.

Our Focus: How can we create a visualization tool, that can at minimum visualize more than 600K in either 2D or 3D?

Visualization Tool Analysis

To analyze the different tools that exist out there, we'll be looking at the following criteria:

UI/UX → Tool is easy to use
Data Capacity → Tool allows visualizing 600K points or more
Responsiveness → Tool does not slow down at 600K points or more
Interactivity → Tool allows users to view and manipulate the data in various ways
Customization → Tool allows developers to customize styles and functionality

2D SCATTER PLOT TEST

Tool	UI/UX	Data Capacity	Responsiveness	Interactivity	Customizability
Tensorflow Projector	Good UI/UX Overall	10K points Maximum	Unable to test	Many Data Analysis Tools	Rigid with minimal room for customization
Voxel51	Good UI/UX Overall	Able to handle up to 1.2 million points	Good responsiveness even at high data point numbers	Many Data Analysis Tools	Constraint to using 51 App Instance
Plotly Dash	Depends on our design	Can handle around 1M points	Good Performance above 600K points, which slows down around 1M points	Many Data Analysis Tools	High level of customizability from app style and utilizing event listeners click/lasso/box-select

Figure 1: Matrix comparing different visualization tools and their capabilities in terms of producing 2D plots

3D SCATTER PLOT TEST

Tool	UI/UX	Data Capacity	Responsiveness	Interactivity	Customizability
Tensorflow Projector	Good UI/UX Overall	10K points Maximum	Unable to test	Many Data Analysis Tools	Rigid with minimal room for customization
Voxel51*	Unable to test	Unable to test	Unable to test	Unable to test	Constraint to using 51 App Instance
Plotly Dash	Depends on our design	~ 500K points Maximum	Good performance below 400K points, which slows down around 500K points	Many Data Analysis Tools	High level of customizability from app style and utlizing event listeners click/lasso/box-select

Figure 2: Matrix comparing different visualization tools and their capabilities in terms of producing 3D plots

**Voxel51 does not seem to have an option for visualizing embeddings in 3D, hence why I was unable to test it further*

For 3D, it would seem that **web-based solutions like Plotly Dash and TensorFlow projector are unable to handle huge amounts of data points** due to browser memory limitations and rendering performance.

Prototype Details → Build

Given the confusion matrix above, I opted to build a prototype based on the Plotly Dash Open Source library, see the demo here!

Embedding Generation

For the prototype, I generated embeddings via a scheduled Kaggle notebook run. I did this with Kaggle's T4 GPU and was able to generate a maximum of 20k embeddings for diamond images. It's important to note that I've intentionally cherry-picked an image of resolution 3x224x224 as this fits the input dimensions for Google's ViT model.

Once the embeddings had been generated, I applied PCA to 3 components, and saved these derived coordinates into a pandas DataFrame with columns:

- "x" → float
- "y" → float
- "z" → float
- "image_path" → string
- "label"→ string

The resulting pandas dataframe is then saved to CSV format and downloaded manually for visualization and analysis.

Embedding Visualization

For plotting the data, the CSV file is loaded as a pandas DataFrame and is passed in as an argument into Plotly's scatter plot function. Plot interactivity comes pre-packaged with Plotly, and functionality can be modified by introducing callbacks. This allowed me to:

- Render images on click, lasso/box select, and cherry-pick
- Allow pagination for rendering 12 images per page
- Disable pagination buttons based on the number of data points selected

It's important to note that for demonstration purposes, the 2D scatter plot uses the same DataFrame, but with a dropped "z" column. I want to emphasize that in practice this is

not recommended and we should be computing PCA on the embeddings with 2 principle components in mind from the get-go.

Page styles are edited in the headers.css file which must be stored in the assets directory for Plotly to automatically fetch and load the stylesheet. I've also added reduced opacity and different shapes to minimize point occlusion, a common impact of overplotting. Finally, Users can also double-click on the plot legend to isolate a class for viewing and click on the legend to hide a class.

Dependency Management & Deployment

The prototype mainly uses poetry as the main dependency management tool. Bla bla bla, deployment details

Recommendation

We can immediately scrap the idea of using Tensorflow Projector since the data capacity is so much lower than Voxel51 and Plotly Dash at the time of writing this document.

Voxel51 has the main visualization and analysis tools with the data capacity to boot. However, usage of their features is restricted to the 51 app which have to be launched via the web browser or through the desktop app. This makes it difficult for SUPA developers to customize functionality and style which may be problematic. On top of that, they don't seem to have an option for 3D visualization in their embedding analysis tool

Plotly Dash has the visualization and basic analysis tools pre-packaged, with a lot of freedom for styling the Dash web app as well as customizing interactivity based on the event listeners they have (onClick, onZoom, etc.)

This would make it easier for SUPA developers to engineer our unique solution as opposed to Voxel51. For a 2D scatter plot, we're able to get great performance even after 600K points with diminishing performance around the 1M points mark. For 3D however, it seems to only be able to render around 500K points before performance diminishes.

My final recommendation in actionable steps is as such:

1. Talk to prospective clients and discuss with them to get the following information:
    a. What's the scale of their data?
    b. What features are they looking for? Only visualization or a full end-to-end solution?

2. Get in touch with the Aquarium and their Illume team to discuss how they've been able to build a visualization tool for huge data points.
    a. They've expressed their openness to this here
    b. You can contact them here

3. If the client expects an end-to-end solution, more research is required on the data ingestion and embedding generation side of things.
    a. My existing research on this is attached in "Appendix" for you to build off

4. Based on feedback from the client and Aquarium's resources, we can begin building the application.
    a. A minimal Dash App setup, with onClick events and 2D/3D plots rendered with WebGL, can be found here

Appendix

Data Ingestion - source

We can store the data in the ".parquet" format based on the Apache Arrow format for huge datasets with rows in the millions. Parquet files employ a lazyframe so that you can load the data only when you need it utilizing lazy loading. An example for using and reading in Parquet files in Plotly here

Embedding Generation

At the initial stages of my research, I focused on the end-to-end process, beginning with tracking the embedding generation times. While this is not the main focus of the research, below are my findings anyway for the following configurations:

- Model: Google ViT Based-sized model
- Data: Diamond Images
- Clustering Algorithm: Principle Component Analysis (PCA)
- Embedding Generation Time Avg (CPU): 0.768s / embedding
- Embedding Generation Time Avg (GPU): 0.01896s / embedding
- Dimensionality Reduction Time Avg: 0.007s / embedding

Here is also a public repo I created to demo the embedding generation process and the visualization process via Plotly's Dash Web App. The actual embedding generation however is done via this kaggle notebook due to my limited computing resources.

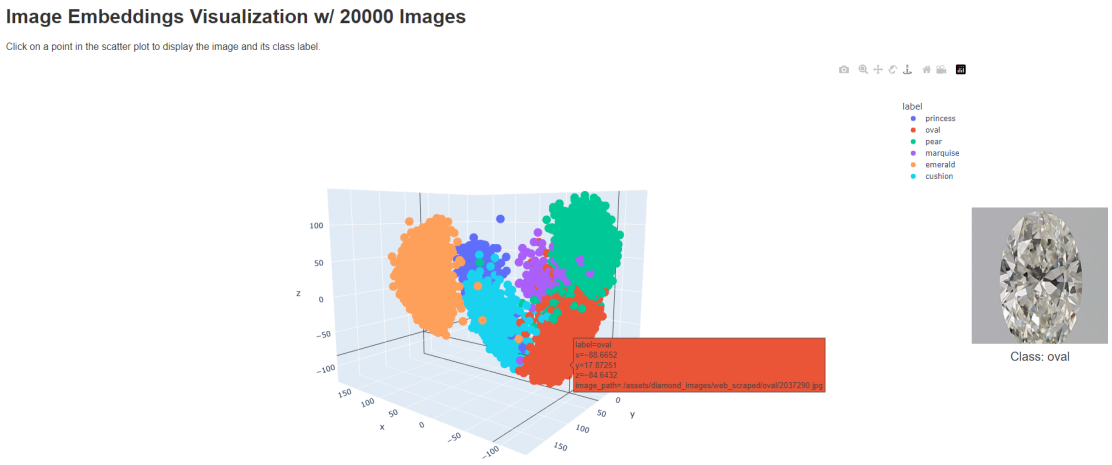

Figure 3: Embedding Visualization w/ 20K images

I only generated 20k embeddings with the T4 GPUs as I constantly ran into out-of-memory issues despite implementing batching for the embedding generation. Since this was out of scope for what is required in this project, I opted to focus my time on the visualization aspect.

Honorable Mentions

In my research for a visualization tool, I also came across the Datashader Library which implements rasterization to be able to plot millions of data points however the caveat is that this only works for 2D plots and not 3D.

I've also been able to push performance pass the 500K data points mark with Plotly by using Holoview but only up to 600K data points. Even then, performance was poor making the app practically unusable.

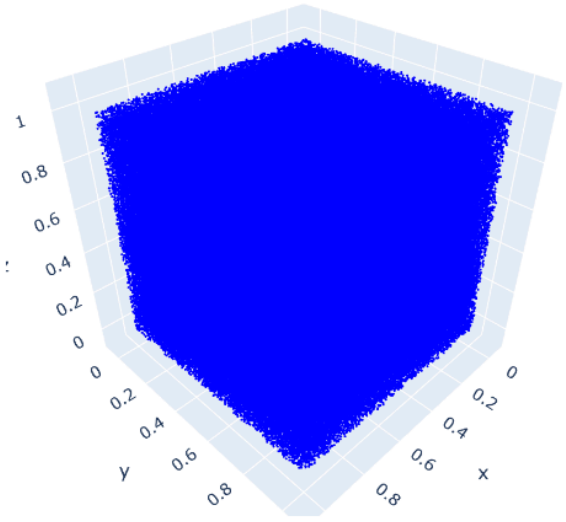

Figure 4: 600k Points Plotted With Plotly and Holoview

It should also be noted that Plotly has good performance with plotting 1 million data points in 2D, however at that point, you would most definitely suffer from overplotting which would make analyzing these embeddings a pain for our clients instead of solving the problem for them.

Figure 5: 1 Million points plotted with Plotly Dash in 2D