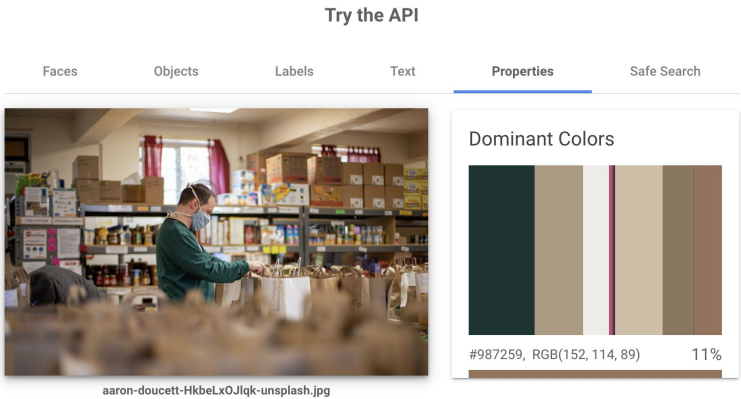
Data Visualization Framework

Bill Qin, Lucy Hu, Evelyn Li
bogdan-jonathan-fan-club

Domain - Image Analysis

The framework performs image analysis on images from different sources provided by data plugins and shows results in different ways using visualization plugins. Potential image analysis includes object recognition, label generation, emotion recognition for images containing faces, dominant color identification, etc. The framework performs these image analysis itself using various Google's Cloud Vision APIs, thus providing the benefits for reuse.



Domain - Data Plugins

Data plugins could provide images from various sources, potential data plugins include:

- A web scraper plugin that takes in an url and grabs the images that website contains
- A unsplash plugin that can get a random photo or a list of images given a topic using the unsplash API
- A local image upload plugin that can allow users to upload a local image
- Many other interesting image and photo related APIs to serve as potential data plugins (DALL·E, Cat API)

Domain - Visualization Plugins

Visualization plugins could include:

- Word cloud containing objects recognized, labels generated, emotions, etc.
- Bar graph displaying the objects, labels, emotions and their corresponding score
- Pie chart showing the color distribution
- Many other potential possibilities!

Generality vs Specificity

The framework possesses an abstraction for getting images, extracting information, and generating visualizations. Such an abstraction allows different implementations for loading images from different APIs, extracting different types of data for a given image, and generating different types of visualizations.

```
/**
 * Grab the images from some source of data.
 */
ArrayList<BufferedImage> getImage(Source imageSource);

/**
 * Get data (ex: text, color, etc.) from a image.
 */
Data getData(BufferedImage image);

/**
 * Generate a visualization from data.
 */
VisData generateVisualization(Data);
```

Reuse

- The data and display plugins are consistent with their interface, supporting the reuse of code that interacts with the interface.

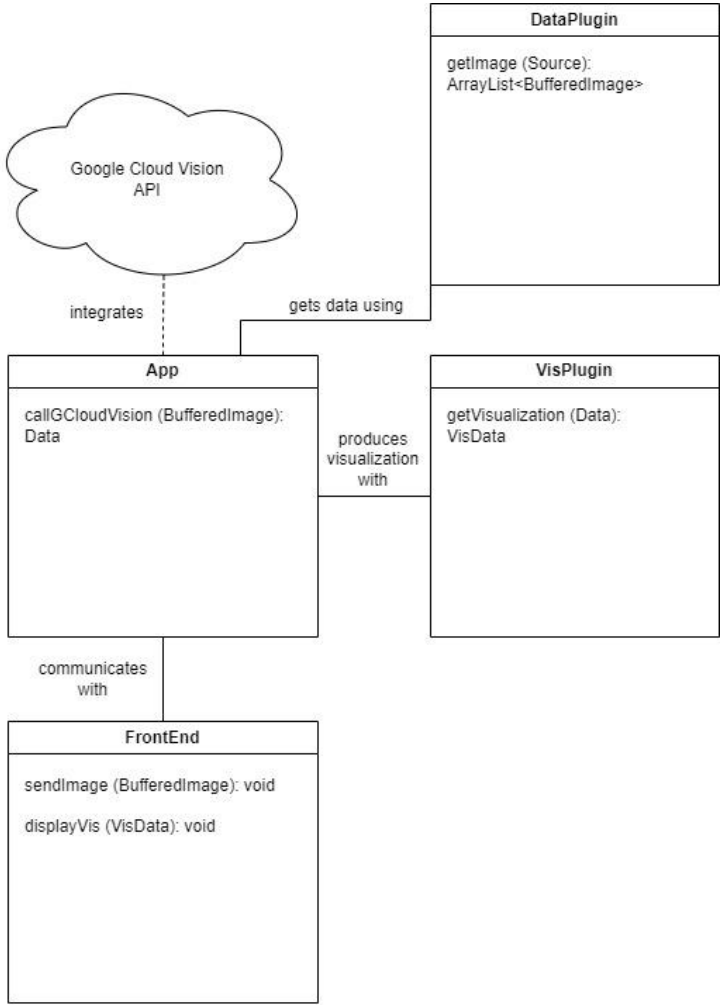
- The core framework is separated from the data and display plugins to allow adding or removing new plugins without much changes to the core framework code.

```
/**
 * Grab the images from some source of data.
 */
ArrayList<BufferedImage> getImage(Source imageSource);

/**
 * Get data (ex: text, color, etc.) from a image.
 */
Data getData(BufferedImage image);

/**
 * Generate a visualization from data.
 */
VisData generateVisualization(Data);
```

UML Diagram


DataPlugin
getImage (Source): ArrayList<BufferedImage>
Google Cloud Vision API
integrates
gets data using
App
callGCloudVision (BufferedImage): Data
produces visualization with
VisPlugin
getVisualization (Data): VisData
communicates with
FrontEnd
sendImage (BufferedImage): void
displayVis (VisData): void

Data Plugin Interface

```
/**
* Grab the images from some source of data.
*/
ArrayList<BufferedImage> getImage(Source imageSource);
```

Back-end

The Source datatype is part of the plugin, which contains information necessary to retrieve the images. For example, if the plugin is an API accessor, the Source class may include information such as the JSON of the API request, where as if it is a local upload, it may simply be the path to the local location of the file.

This will result in a list of `BufferedImage`s, which can be presented to the user through the UI, and then data can be retrieved using `getData()` as described above.

Data Plugin Interface

Front-end

The front-end will likely require some form of a `render()` depending on how the image source is retrieved. If it is a local upload, it will need a UI to select images. If it is an API accessor, it could instead be text boxes storing an API key, an identifier, or other information needed to create a successful API request.

Visualization Plugin Interface

Back-end

Much like `Source`, `VisData` is likely to also be implemented by the plugin, depending on the components required for the visualization.

```
/**
* Generate a visualization from data.
*/
VisData generateVisualization(Data);
```

Front-end

The front-end will once again require some form of a `render()`, based on the `VisData` class.