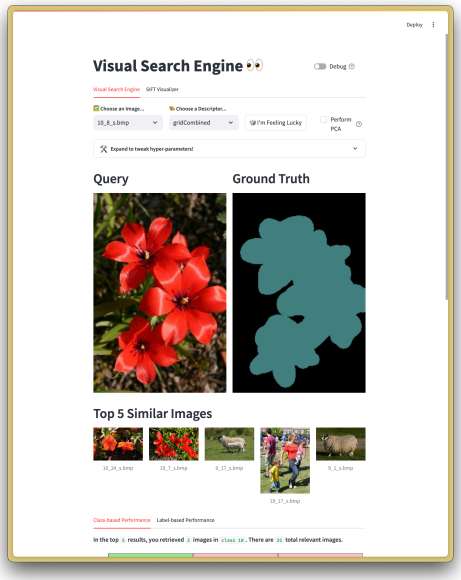
Introduction to Visual Search System (PDF Version)

▼ **Table of Contents**

Introduction 👋



The Visual Search Engine GUI

Before we start, I encourage cloning the repository and reference the code here [1], as well as reading the report from the notion website here.

A streamlined version of this project is currently live on Streamlit at **visual-search.streamlit.app**. I'll publish the complete version after the assignment deadline to comply with the University of Surrey's academic integrity policy. I'm a big fan of Grant Sanderson's emphasis on visual engagement and intuition in learning, so I hope this project becomes a valuable learning resource and reference for future students.

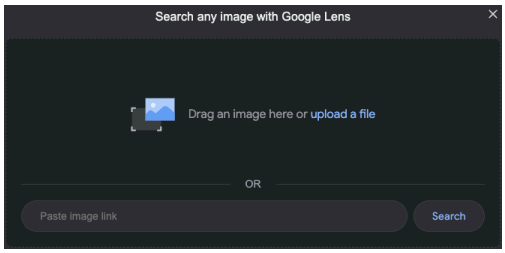
If you're curious about how I built this project, I'll be releasing a separate guide and working with Surrey CompSoc to host a workshop.

For the graders, bare with me and I'll try best to make this lengthy report as painless to read as possible. Also, for the code, you'll also find comprehensive unit tests here [1].

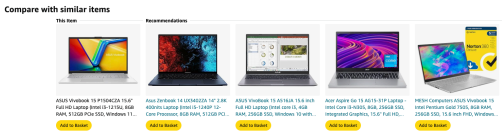
Abstract 💭

Digital image collections are traditionally searched using textual queries (e.g., Google). However, searching based on visual appearance is often desirable (e.g., Google Lens)—for instance, when recommending visually-similar products in online shopping. Visual search systems address this need by allowing users to search for images based on their visual characteristics rather than relying solely on textual descriptions.

In the context of image search, we must **first define what makes two images "similar."** Images are considered similar based on a combination of visual content and semantic content.**Visual content** of an image includes color, texture, and shape, whereas **semantic content** focuses on objects, context, and the relationships between these objects.



Google's `Search by image` option

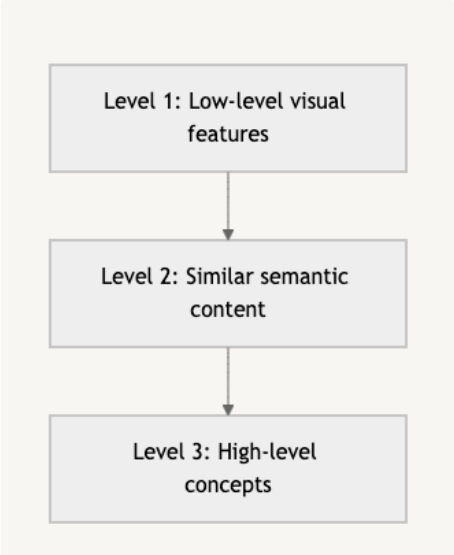


Amazon's `Similar items` recommendation

More specifically, we have established three levels of image similarity in lecture 7 [2]:

- **Level 1**: Low-level visual features such as **texture** and **color**
- **Level 2**: **Similar semantic content** such as an object or an activity
- **Level 3**: **High-level concepts** such as war, comedy, news, etc.

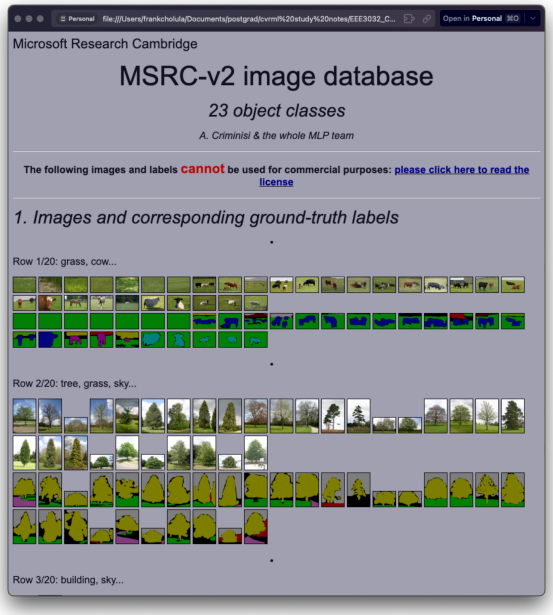
As we progress through these levels, we observe an increase in semantic content that is, in essence, **more human-like**. The **semantic gap** refers to the discrepancy between the low-level features extractable through computer vision and the high-level similarity definitions desired in content-based image retrieval.



Increasing semantic content as we go down the hierarchy

In this report, we'll progress from extracting low-level visual features to exploring more complex semantic content. We'll then use these extracted descriptors to construct a visual search system, comparing various distance metrics. Finally, we'll evaluate our system's performance using the results of the the output images. This approach aims to bridge the semantic gap between computer and human perception of image similarity, offering insights into how we can improve our visual search system.

About the Dataset



`ClickMe.html` description of the Image Database

object class	R	G	B	Colour
void	0	0	0	
building	128	0	0	
grass	0	128	0	
tree	128	128	0	
cow	0	0	128	
horse	128	0	128	
sheep	0	128	128	
sky	128	128	128	
mountain	64	0	0	
aeroplane	192	0	0	
water	64	128	0	
face	192	128	0	
car	64	0	128	
bicycle	192	0	128	
flower	64	128	128	
sign	192	128	128	
bird	0	64	0	
book	128	64	0	
chair	0	192	0	
road	128	64	128	
cat	0	192	128	
dog	128	192	128	
body	64	64	0	
boat	192	64	0	

The Provided Legend from the MSRC V2 dataset

We'll be using the MSRC-v2 Image Database, provided by Microsoft Research Cambridge [6]. This dataset contains **591 images across 20 classes**. While each image belongs to a single class, it can feature multiple

objects. Although **the dataset doesn't come with pre-assigned labels**, we can generate them using the provided legend and ground truth images, which are properly **segmented** versions of the original images.

Ground truth image of image `16_14_s.bmp`. It belongs to class `16` with the labels `["dog", "road"]`

A more complicated Image `20_14_s.bmp` of class `20` with the labels `["water", "boat", "tree", "sky", "building", "face", "body"]`

Conveniently, I've created a `labels.json` file containing each image's **class** and **labels**. This was done by detecting pixel colors in the corresponding ground truth images. This file will be useful in evaluating our visual search system's performance. You can reference the code here [1]. The gist of the logic is implemented in the `ImageLabeler` class here [1].

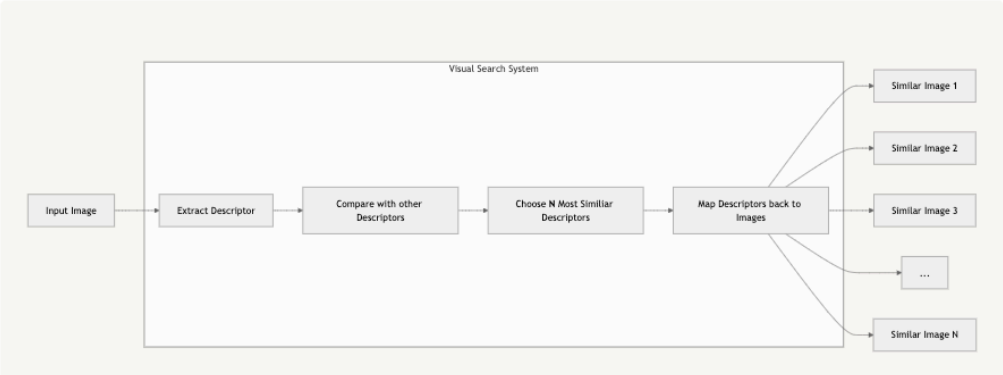
```
{
    "17_15_s.bmp": {
        "labels": [
            "building",
            "road",
            "sky"
        ],
        "class": "17"
    }
}
```

The resulting JSON representation of one image is shown on the left.

For instance, the image `17_15_s.bmp` belongs to class `17` and contains the labels `[building, road, sky]`. Essentially, you can think of **a class as a superset of the labels.**

If you'd like to examine the full `labels.json` file, you can find it in my Github repository.

Defining the Evaluation Methodology 🔬



Pipeline architecture of our visual search system

As illustrated in the pipeline above, the Visual Search System (VSS) accepts an input image and generates `N` similar images as output.

We can evaluate our system by examining either the **classes** or the **labels** of the retrieved images. **While we'll primarily assess the system's performance using a class-based approach, it's important to also consider the label-based method**. The label-based approach provides more in-depth information, especially given that the class designations are somewhat arbitrary for the MSRC image set we're using (as you will see in the callout in the conclu).

1. In the **class-based approach**, the goal is to retrieve images that **share the same class** as the input image.
2. In the **label-based approach**, we want to retrieve images that share **at least one label** with the input image.

We're first going to define our **precision** as:

`Of the top 10 retrieved images, how many share the same class or relevant labels as the input image?`

and **recall** as:

`Of all the images that could be relevant (i.e., share one or more labels or belong to the same class), how many did the system retrieve?`

As a result, our **threshold** will be defined as:

`The rank at which we stop considering retrieved items (e.g., after the top 1, top 2, top 3, …, top N images).`

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Each evaluation will involve defining True Positives (TP), False Positives (FP), and False Negatives (FN). We'll use these metrics to calculate precision and recall using the formulas on the left. In the following sections, I'll delve into the details of each evaluation approach.

Class-Based Evaluation

Class-based evaluation is relatively straightforward. We aim to retrieve images that are in the same class as the input image.

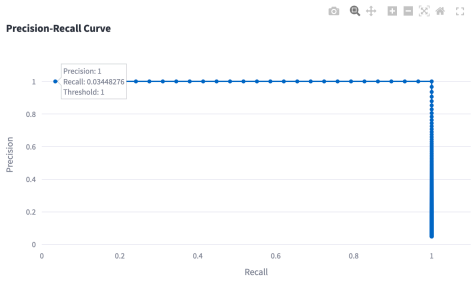
✨ **Defining TP, FP, FN, TN for class-based Evaluation**

	Positive	Negative
True	A retrieved image is **in the same class** as in the input image (correctly retrieved).	A retrieved image is **not in the same class** as the input image (Incorrectly retrieved).
False	An image that **belongs to the same class** as the query image **was not retrieved.**	An image that **belongs to a different class was not retrieved**.

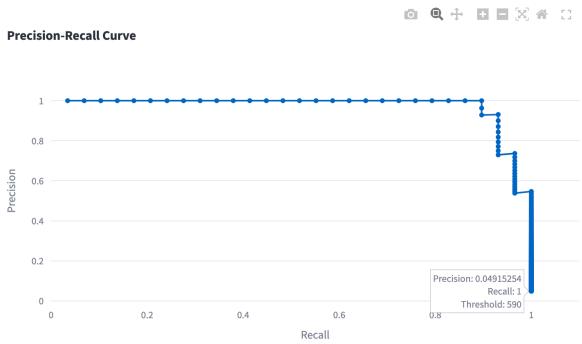
Defining Total Relevant Images

The total relevant images are those in the dataset that belong to the same class as the input query image, excluding the input image itself. In this dataset, each class typically contains 29 relevant images (30 minus the input image).

It's worth noting that the precision and recall **will never be exactly zero for a well-functioning system** based on our definition. This is because **we've excluded the input image** from our search. I mean, an visual search engine that returns the exact match of an input image will be quite useless.



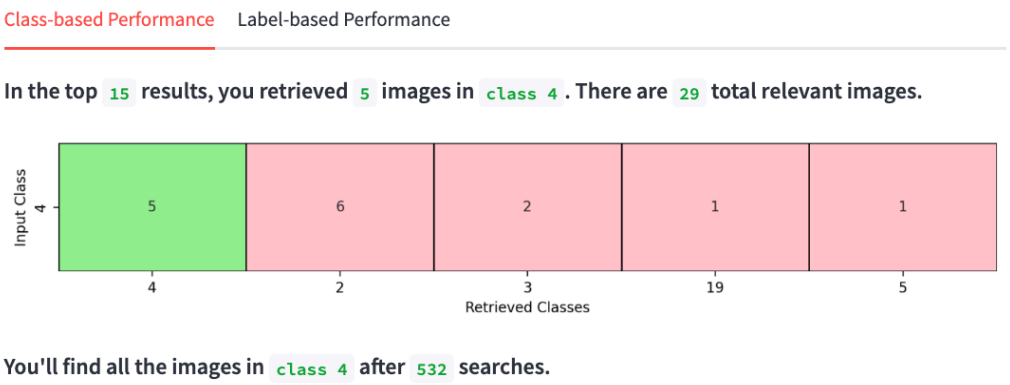
example of recall at threshold 1



example of precision at threshold 590

To illustrate, at threshold 1, if we make a correct retrieval, our **recall** will be `1/29`. At threshold 590, our **precision** will be `29/590`.

For visualization purposes, I've created a matrix showcasing the retrieval results. The y-axis represents the input class, while the x-axis shows the retrieved classes. Green cells indicate correct retrievals, and red cells represent retrievals of incorrect classes. You'll be able to reference the code for generating these plots and matrices here [1]. At the bottom, a message indicates the threshold values at which we successfully retrieve all relevant class images.



Label-Based Evaluation

Recall that (no pun intended) in the label-based evaluation, we need to retrieve images that share **at least one label** with the input image. This approach is more nuanced because an image might have multiple associated labels. I've thought about two approaches.

1. **Instance-level Evaluation (Micro averaging)**: a relaxed approach **where we evaluate the entire image** as a whole. In other words, if any label matches between the input image and retrieved image, that image counts as a true positive.
2. **Label-level Evaluation(Macro averaging)**: a more stringent approach **where we evaluate each label individually**. Each label in a retrieved image can be treated as a separate evaluation point. This adds more granularity in the expense of added complexity.

Instance-level Evaluation

Due to the time constraint of this assignment, I will be focusing on the **Instance-level evaluation** and define my **TP**, **FP**, **FN**, **TN,** and the **Total Relevance** accordingly.

✨ **Defining TP, FP, FN, TN for label-based, instance-level evaluation**

	Positive	Negative
True	A retrieved image that shares **at least one** label with the input image.	A retrieved image **that does not share any labels** with the input image.
False	A relevant image in the dataset that (i.e., an image that shares at least one label with the input image) **that was not retrieved.**	An image in the dataset **that was not retrieved** and **does not share any labels** with the input image.

Defining Total Relevant Images

This refers to the total number of images in the dataset that have at least one label in common with the input image. For instance, if the input image has the labels `["building", "grass", "tree"]`, the total relevant instances would be the count of images in the dataset that contain **any of these labels**.

Since we have multiple labels associated with each image, so we need a way to aggregate our calculated precision and recall. We are going with the micro averaging approach and calculate the **mean average precision and recall (mAP and mAR)** for each threshold. Here are the specific steps:

1. For each threshold, compute TP, FP, and FN counts for labels and compute precision, recall.
2. Repeat step one for the next threshold, then calculate the average precision and average recall across the thresholds.

I have implemented this in the `calculate_cumlative_preicions_recall_f1` function in the `LabelBaesdEvaluator` class here [1].

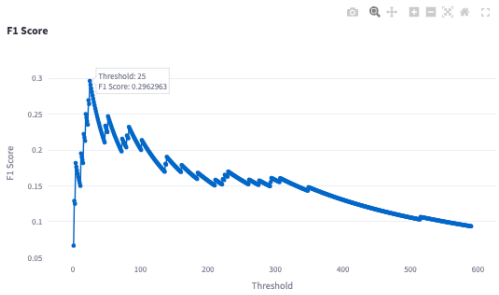
Similar to the class-based evaluation, we can generate a visualization to show the labels we retrieved correctly. The y-axis displays all the retrieved labels, regardless of their accuracy. The x-axis represents different images retrieved at various thresholds. Green cells indicate correct labels, while red cells represent incorrectly retrieved labels.

Other Evaluation Methodologies

Though we won't be using these in our experiments, I've also generated F1 score for class-based evaluation and label-based evaluation, combining both the precision and recall into a single metric.

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

You can see the generated F1 score vs. threshold plot on the right.



F1 score of class-based evaluation using `17_15_s.bmp`

The F1-score balances precision and recall. It's particularly valuable **when comparing different thresholds to find the optimal trade-off between these two metrics.**

In a discussion with fellow student Can Ali Yarman, we conceived a more sophisticated approach: evaluating the percentage of label content in each image. For example, if an image contains labels `['tree', 'sky', 'building']`, we could quantify the percentage of the image occupied by each label by counting pixels in the segmented ground truth image.

While this approach is intriguing, it's too complex and beyond the scope of this assignment. Therefore, I won't discuss it further in this documentation.

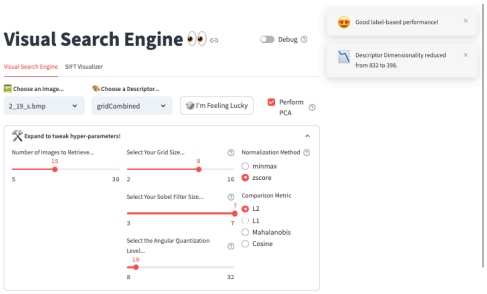
Implementation of Visual Search Techniques 🛠️

Distance Measures for Descriptors

Before delving into the descriptor extraction, let's examine the similarity metrics used to compare two descriptors. I will be focusing on 4 distance metrics specifically:

1. **L1 Norm (Manhattan Distance)**
2. **L2 Norm (Euclidean Distance)**
3. **Mahalanobis Distance**
4. **Cosine Similarity**

you can find the metric code here under `retrievers.py`.



Control panel of our visual search system [1]

For each similarity measure, I will briefly explain the **intuition behind these distance metrics** and speculate how they might perform best for different types of images. Then, to corroborate our speculation, I'll conduct an experiment for each distance measure using **one image**, `2_19_s.bmp`, using the `gridCombined` descriptor with the following hyper-parameters set in the control panel image.

🔔 **Experiment Success Indicator**

To visually represent **the success of each experiment** in this report, I'll use the following emoji system in the experiment headings.

- 🟢 **Excellent**: More than 10 out of 15 relevant images retrieved
- 🟡 **Moderate**: 5-10 relevant images retrieved
- 🔴 **Poor**: 0-5 relevant images retrieved

To represent the **most successful distance metric**, I'll use the following emoji system in the experiment headings.

- 🚕 **L1 Norm**
- 📏 **L2 Norm**
- 🌭 **Mahalanobis Distance**
- 📐 **Cosine Similarity**

This quick visual guide allows for easy assessment of each experiment's performance.

For the hyper-parameters of the distance metric experiment:

- **Grid Size is set to 8**. The image is divided into 64 squares.
- **Sobel Filter Size**: 7
- **Angular quantization level is set to 10.**This means the 180-degree range of orientations will be divided into 10 equal bins.
- **Z-score** normalization is used when combining the grid color histograms.
- **Perform PCA** is set to true to reduce the descriptor dimension from a whopping 832 to 398.

Both class-based and label-based PR curves and the retrieval matrices will be provided as well in each experimental result section.

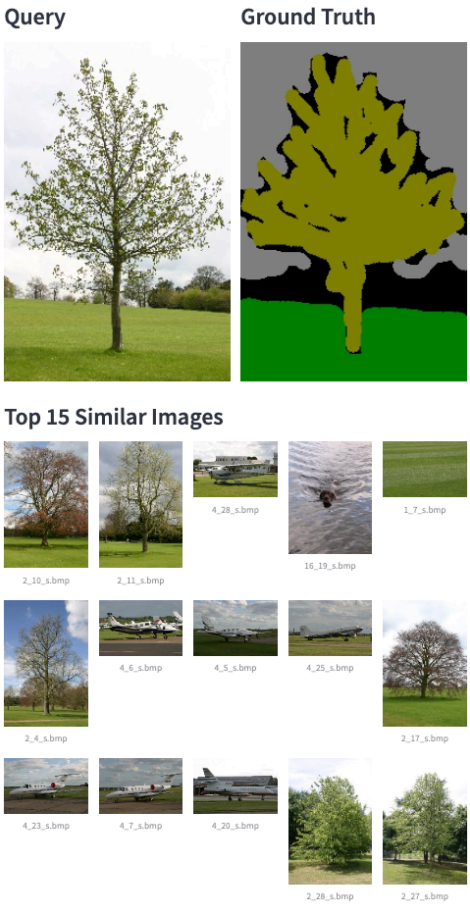
L1 Norm (Manhattan Distance) 🚕

$$L1(x,y) = \sum_{i=1}^{n} |x_i - y_i|$$

The L1 norm computes the sum of absolute differences between the descriptors.

This is, in a way, the simplest distance metric. Intuitively, it should work well with **sparse descriptors** (zero minus zero is zero) and doesn't exaggerate larger differences compared to L2. We expect it to perform effectively with simple features such as color intensity or basic shapes, including color histograms and texture histograms.

▼ **Experimental Results using L1** 🟡



Retrieval results using L1

The L1 performance is decent. It effectively captures fundamental differences in edge orientations (such as tree branches versus background) and color distributions (predominantly green images).

Class-based performance using L1 [1]

Label-based performance using L1 [1]

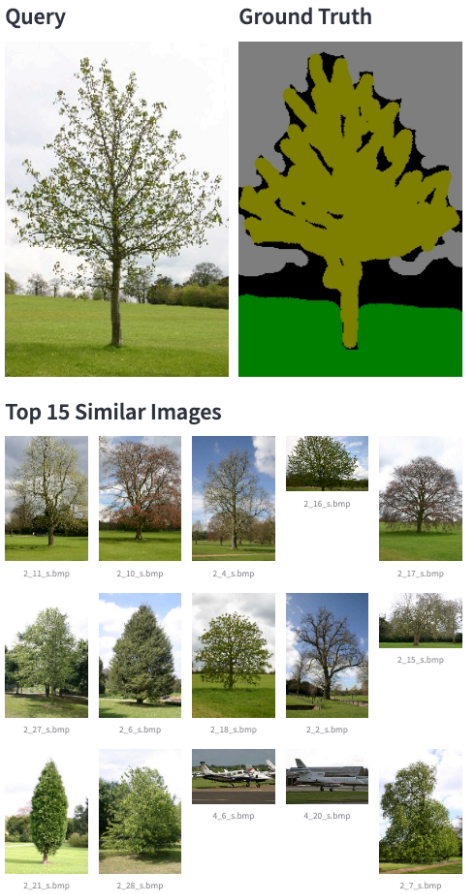
L2 Norm (Euclidean Distance) 📏

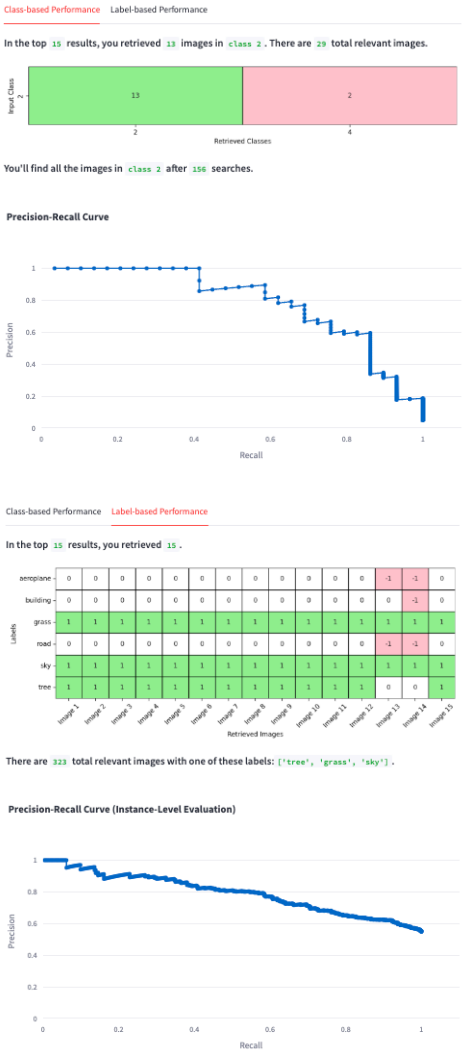
$$L2(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

The L2 norm computes the square root of the sum of squared differences.

L2 norm is sensitive to large differences due to squaring, making it effective for fine-grained features that depend on spatial relationships. It's particularly useful when every dimension of the image descriptor carries meaningful information. The squaring also makes L2 well-suited for dense descriptors. Consequently, we expect key-point based descriptors like SIFT, and spatial gradients like HoG, to perform relatively well with L2 norm.

▼ **Experimental Results using L2** 🟢





The L2 performance surpasses that of L1. It emphasizes large deviations in edges and captures the intricate differences in tree branches more effectively.

Mahalanobis Distance 🌭

$$D_M(x, y) = \sqrt{(x - y)^T \Sigma^{-1} (x - y)}$$

The Mahalanobis distance accounts for correlation between dimensions and scales the distance by the covariance of the data, where Where Σ is the covariance matrix of the feature vectors.

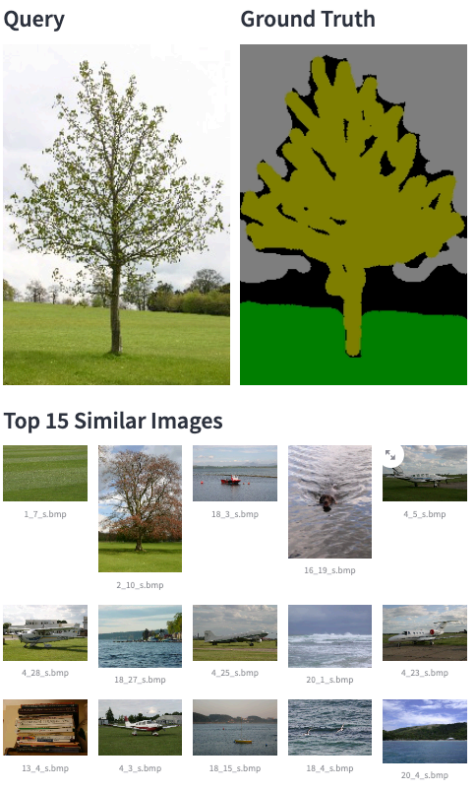
> "You're not going to fit a ⭕ over a 🌭, are you?" - Professor Miroslaw Boeber [2]

Ah, the sweet sweet sausage, as Professor Boeber affectionately called it. Since we are taking into the account of the correlation and variances of our dataset, Mahalanobis distance should account for feature scaling. Unlike L1 and L2 norms, which treat all dimensions equally, Mahalanobis distance accounts for feature variance, giving less weight to dimensions with high variability. This should work well with our PCA-reduced features and texture features such as the edge orientation histograms.

In practice, computing the covariance matrix can be computationally expensive for high-dimensional features. I observed this firsthand when reloading the Distance Metrics in the VSS control panel. Moreover, due to floating-point rounding errors and potential matrix singularity issues, the Mahalanobis distance calculation can sometimes produce inaccurate or unstable results. Thanks to Mohammad Nezami, I learned the nifty trick of adding the covariance matrix to its transpose and averaging the two. This ensures the covariance matrix is symmetric and invertible. Here's the code snippet:

```
cov_matrix = np.cov(all_descriptors.T)
# enforcing symmetry here
cov_matrix = (cov_matrix + cov_matrix.T) / 2
# ensuring invertibility in case of singular matrices
cov_matrix += np.eye(cov_matrix.shape[0]) * 1e-6
cov_matrix_inv = np.linalg.inv(cov_matrix)
```

▼ **Experimental Results using Mahalanobis Distance 🔴 s**



The Mahalanobis distance using the `gridCombined` option performs poorly. Despite PCA reducing the descriptor dimension to 398, it's still relatively high. Our covariance matrix might be ill-conditioned since the number of samples isn't significantly larger than the number of dimensions. We might also be giving more weight to the variance in grass backgrounds—which are

abundant in our dataset—rather than the distinctive tree edges we're trying to capture.

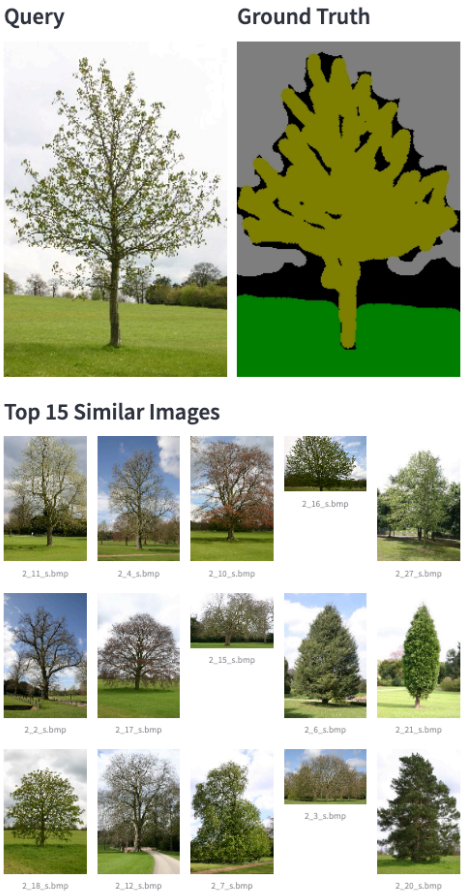
Cosine Similarity 📐

$$\frac{x \cdot y}{\|x\|\|y\|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$$

The cosine similarity measures the cosine of the angle between two vectors, with an emphasis on the direction rather the magnitude of two descriptors.

Cosine similarity has a **scale-invariant** property, as it focuses on the direction rather than the magnitude of vectors. This makes it particularly effective for normalized, high-dimensional descriptors and global features, where the relative proportions of features are crucial. It's likely to excel with deep learning embeddings but may struggle with correlated features. The strength of cosine similarity lies in its ability to capture the essence of feature relationships, regardless of their absolute sizes.

▼ **Experimental Results using Cosine Similarity** 🟢



The z-score normalization boosts cosine similarity's performance. This metric handles variations in edge strengths and color intensities, giving equal weight to both edge orientation histograms (EOH) and color histograms. Our descriptor zeroes in on the **directional alignment of edge and color patterns**—precisely what cosine similarity measures.

Feature Extraction 🍊

Now that we've covered the comparison of descriptors, let's move on to the feature extraction process. A descriptor is a feature vector that contains information about an image. I like to think of it as a "juice concentrate" of the image. As we "distill" the image into a numerical representation, we capture the essential characteristics that make the image unique and recognizable. To extend the analogy further, the way we "juice" an image can give us some descriptors "*with pulp*" (**retaining detailed, localized information**), while others are **more abstract and global**, **representing the overall structure or pattern** ("pulp-free")

iStock image of me juicing an image for a descriptor [8].

In the following sections, I will discuss the juicy 🧃 implementation of basic descriptors before progressing to more advanced techniques. These include quantized color histograms and spatial grids, and in the end, Bag of Visual Words along with dimension reduction using PCA.

For each section, I'll present experimental results using **3 different images with a top 15 image threshold, while fixing the best-performing distance metric and hyper-parameters**. This approach is necessary because the effectiveness of a distance metric heavily depends on both the model and the specific image chosen.

These three carefully chosen images should showcase the strengths and shortcomings of each descriptor.

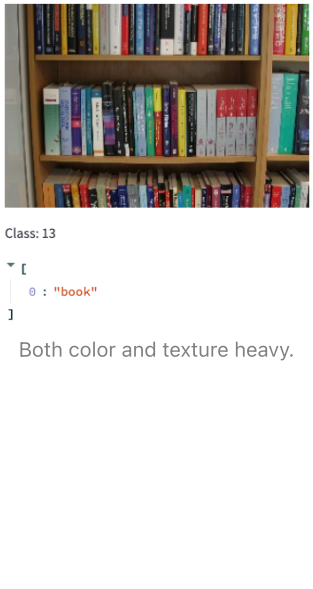
`10_8_s.bmp`

Color heavy image

`8_14_s.bmp`

Texture heavy image

`13_7_s.bmp`



Both color and texture heavy.

Each section will feature an **Experimental Results** subsection. Each subsection will contain:

1. The **hyper-parameters** chosen for the model
2. A set of **15 retrieved images**. Moreover, the subsection will contain
3. **PR curves** and **retrieval matrices** to demonstrate both the class-based and label-based performance.

You can find the specific descriptor code in the `Extractor` class within `descriptors.py` here[1].

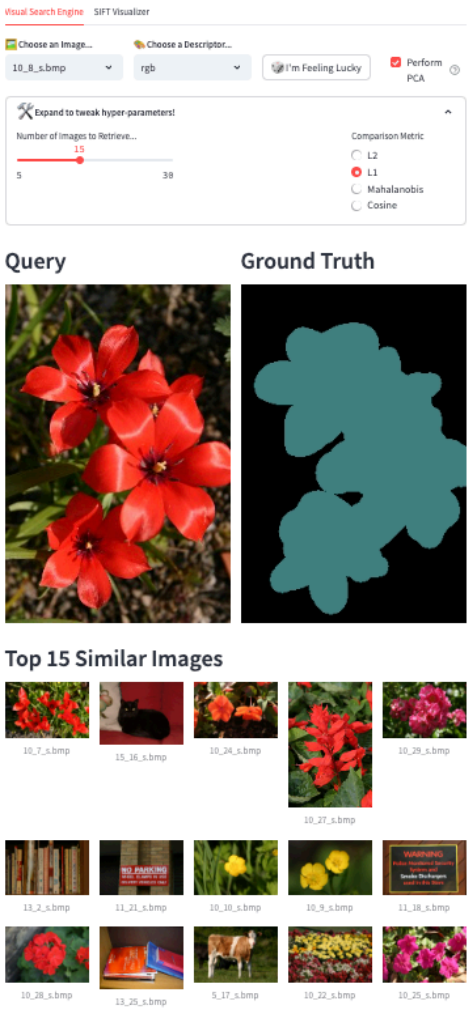
Color Descriptor 🎨

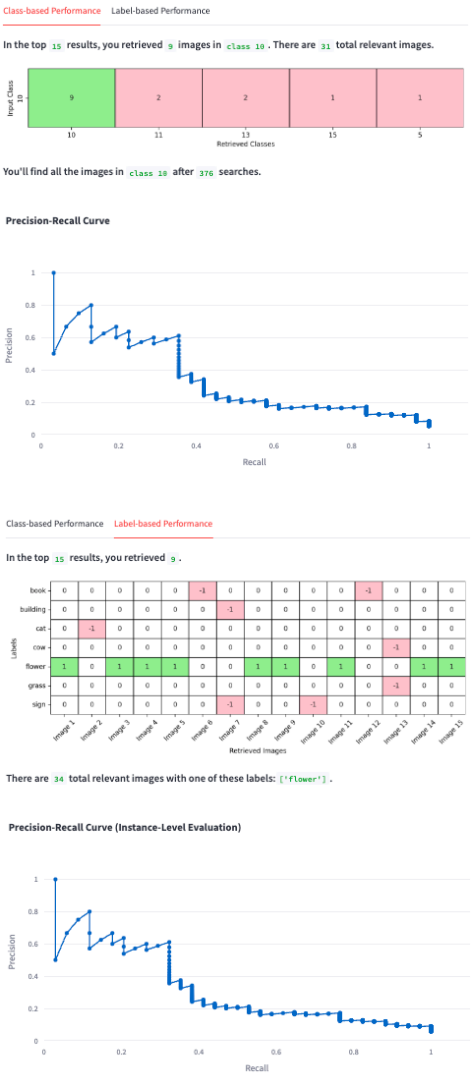
If this descriptor is a drink, it'd be a pumpkin spice latte 🎃. It's the **most basic descriptor with absolutely no depth**. However, compared to a descriptor that just generates an array of random values, it is reliable. Sometimes, simplicity is all you need.

We read the image, we normalize the pixels, and then we calculate the RGB channel means. You can find the code in the `extract_rgb` function here.

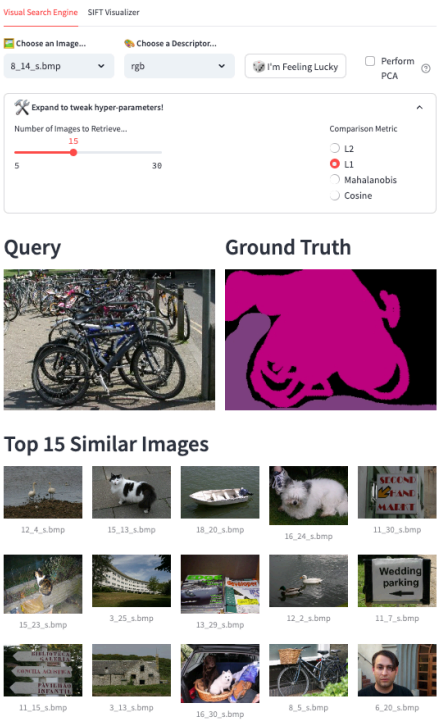
▼ **Experimental Results for Color Descriptor**

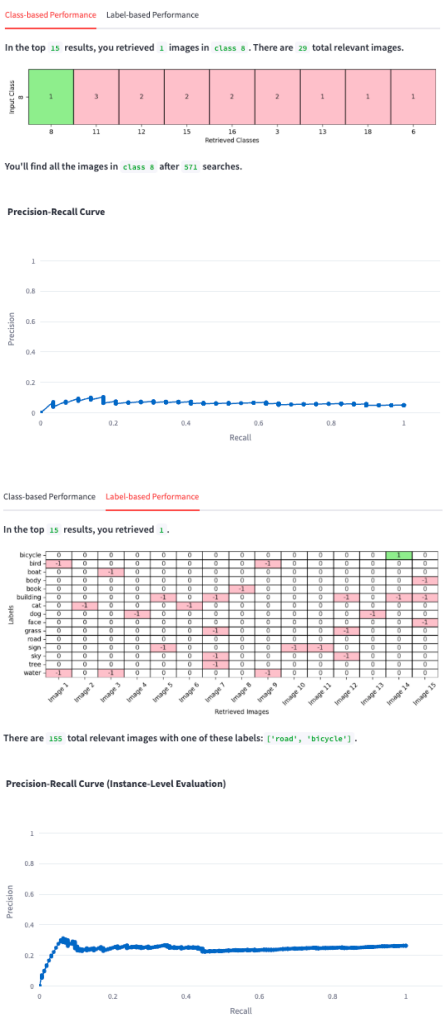
▼ **10_8_s.bmp** (🟡 with 🚕)



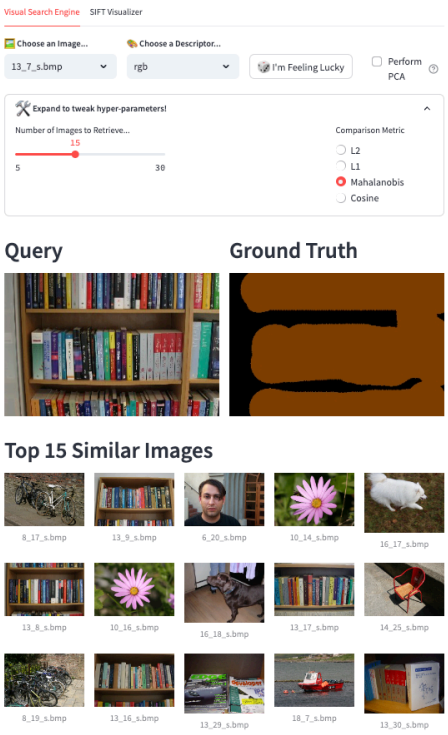


▼ **8_14_s.bmp** (🔴 with 🚕)





▼ **13_7_s.bmp** (🟡 with 🌭)



Using the L1 distance metric for the flower image, the RGB descriptor effectively captures the red hue of the flower and the green and brown tones of the background. This straightforward color-based approach provides a quick and effective way to group visually similar images, especially **when color is a dominant feature**. The use of L1 is also quite obvious as it works well with simple descriptors.

Again, using the best-performing distance metric L1, we still see poor retrieval performance for the bike image. This is primarily due to the abundance of edge features in the stack of bicycles and the relatively uniform pixel colors throughout the image. The simple RGB descriptor struggles to capture the complex **textures** in this scene.

As expected, when both edges and colors are present, we score decently. For image 3, the Mahalanobis distance is the best-performing metric here. This can be explained by the structured and repeated patterns of bookshelves in terms of orientations and colors. Mahalanobis distance leverages these correlations effectively.

Color Histogram 📊

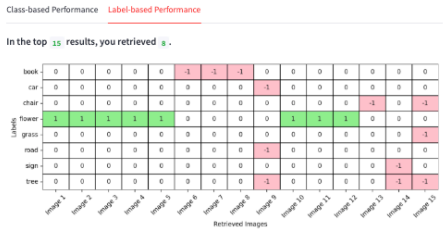
This is an extension on top of the color descriptor above. I have implemented two versions: `extract_globalRGBhisto` and `extract_globalRGBhisto_quant`. You can find the links to them here. The `extract_globalRGBhisto` simply returns the the **standard RGB histogram** for an image by dividing each color channel into bins and calculating the frequency of pixel intensities within each bin. This method is more granular compared to the quantized method.

`extract_globalRBhisto_quant` takes the extra step of reducing the precision of RGB values and mapping them into discrete **quantized bins**. It returns a quantized RGB histogram as covered in lecture. This quantization helps in grouping similar colors and reducing descriptors dimensionality. In a way, it captures the joint color relationship and can offer better robustness to noise such as lighting variations. It's more generalizable.

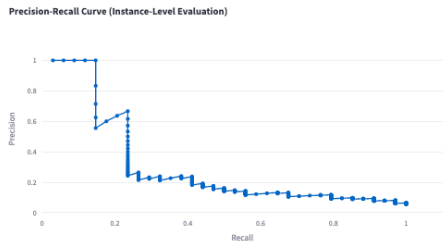
In our experiments, I found that the quantized version generally performed better for our dataset so I'll be focusing on the results of the quantized version.

▼ **Experimental Results for Quantized Color Histogram**

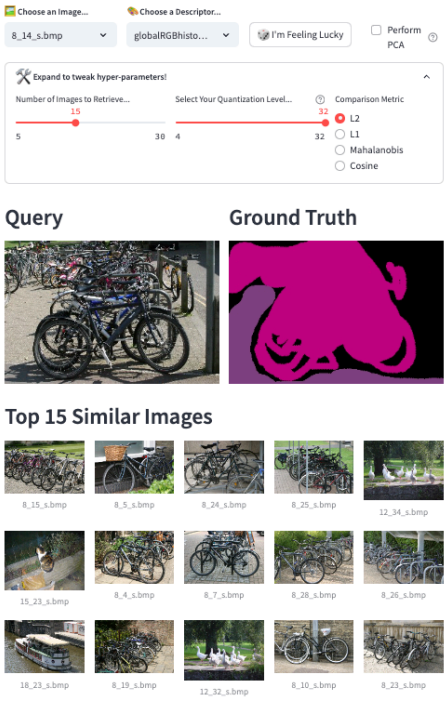
▼ **10_8_s.bmp** (🟡 with 🔪)

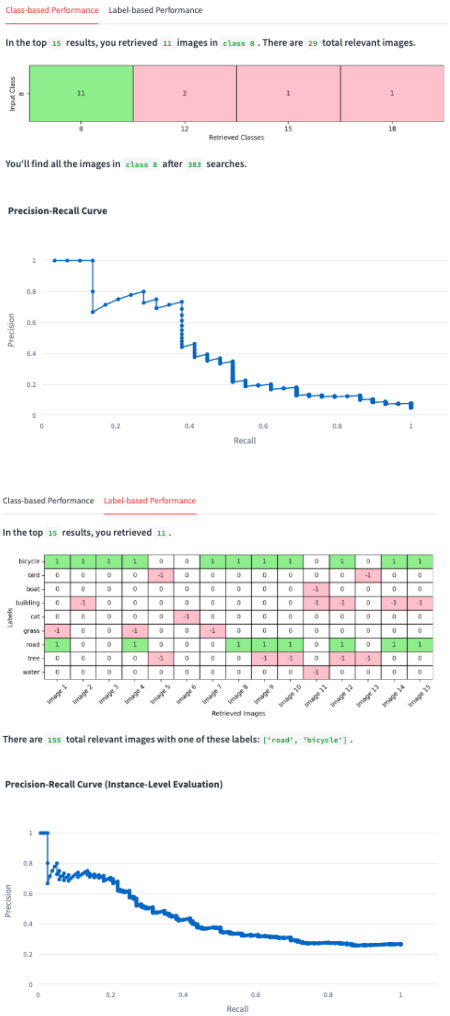


There are 34 total relevant images with one of these labels: `['flower']`.

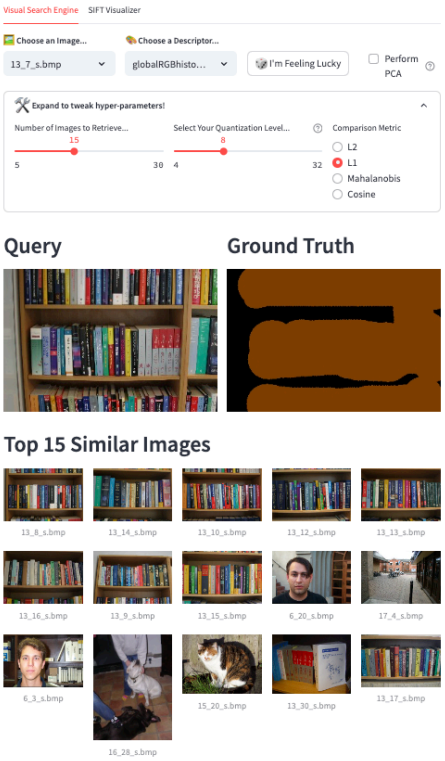


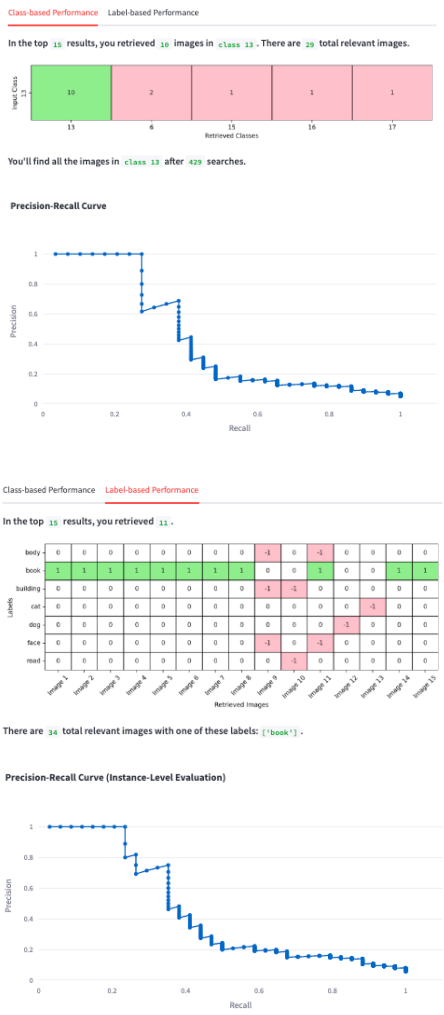
▼ **8_14_s.bmp** (🟢 with 🔪)





▼ **13_7_s.bmp** (🟢 with 🚙)





For the first image, this approach retrieves fewer true positives than the simple color descriptor at first glance. Nonetheless, the quality of the retrieved images improves—we're seeing more red flowers. We're using a quantization level of 8 to minimize noise from other colors. L2 distance performs best here, likely because it emphasizes both the vibrant red flowers and the green background.

Surprisingly, a high quantization level for the color histogram actually helps a lot on the second image. The increased resolution captures more subtle color variations (like the different shades of the bikes) and helps differentiate bikes with similar but not identical color palettes. This is in contrast to lower quantization levels, where these subtle differences might all fall into the same quantized bins.

For the bookshelf image, we see a significant improvement over the color descriptor. By using a quantization level of 8 and L1 distance, we've achieved a **fine-grained yet manageable resolution**. This approach captures subtle differences between book spines without overfitting. The label-based performance is particularly impressive.

Spatial Grid (Color and Texture) ■

I have implemented two descriptors here: `extract_gridRGB` and `extract_gridEOhisto`. These descriptors are designed to encode color information and edge orientation information, respectively. Additionally, each descriptor **incorporates spatial structure through grid-based positioning of the image**. This means the image is divided into a grid of smaller cells, with each cell analyzed separately.

Despite the **local information** being captured in each grid cell, it's important to note that at the end, we concatenate features from all cells, creating **a single descriptor for the entire image**. We do not add them together. This effectively **summarizes the image's overall color or structure.** By tweaking the grid size, we are controlling the balance between local and global information. A **fine grid** captures more detailed information but may lose efficiency and become sensitive to noise; a **coarse grid** captures broader patterns but may miss subtle variations within the image.

Regarding the edge orientation, we apply the Sobel filter, which calculates the image gradient. This gradient distribution helps us detect edges and their orientations in the x and y directions.

A size 3x3 Sobel filter for the x-direction

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

And for the y-direction:

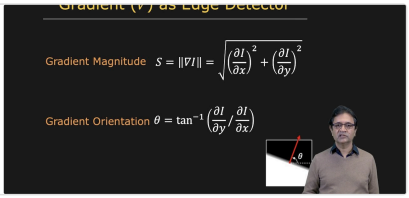
$$S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

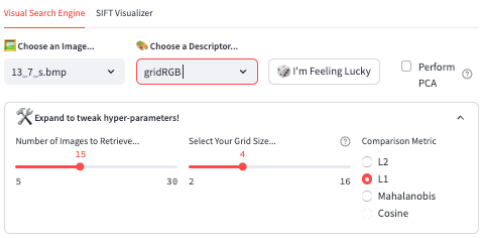
For a detailed explanation of edge detection using different gradient operators, including the Sobel filter, I highly recommend the First Principles of Computer Vision[3] video series. It does an excellent job breaking down the concept, so I'll skip the nitty-gritty details here.

Edge Detection Using Gradients | Edge Detection

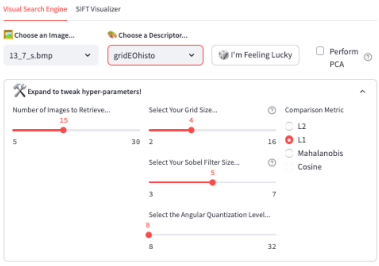
First Principles of Computer Vision is a lecture series presented by Shree Nayar who is faculty in the Computer Science Department, School of Engineering and Applied Sciences, Columbia University. Computer Vision is the enterprise of building machines

https://youtu.be/lOEBsQodtEQ?si=3Wt-VnKm6vz41nl6





Hyper-parameters for `extract_gridRGB` [1]



hyper-parameters for `extract_gridEOhist` [1]

The grid-based color approach doesn't introduce any novel concepts beyond incorporating the grid-based method, so the only parameter we need to tune is the grid size. Note that I implemented the grid size as the number by which the image will be divided both horizontally and vertically. For example, a grid size of 8 means the image will be divided into 64 equal sections (8x8).

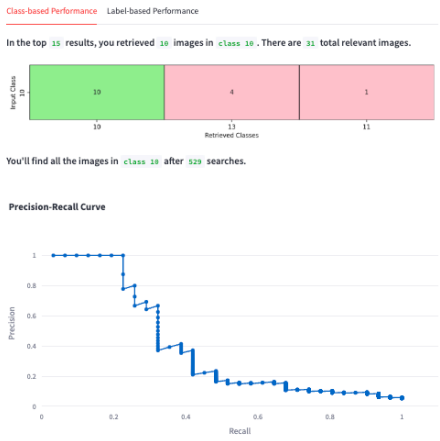
For the edge orientation approach, there are additional parameters to adjust beyond just the grid size. I've incorporated different Sobel filter sizes (3, 5, and 7) as well as angular quantization levels ranging from 0 to 32.

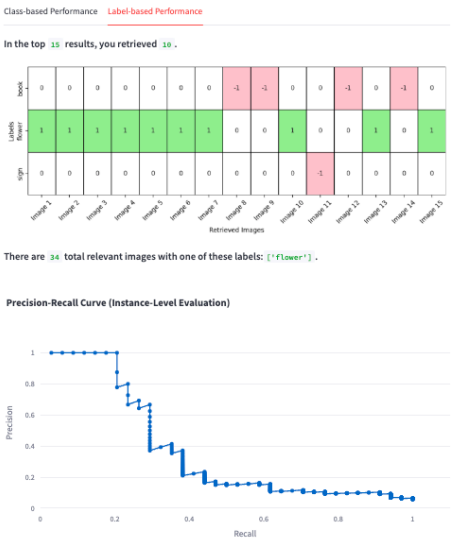
The key insight in angular quantization is that we only need to **consider 180 degrees**, **as edge orientations are symmetric**. An edge at 0 degrees is identical to one at -180 degrees. This symmetry allows us to represent all edge orientations within a 180-degree range, simplifying calculations and reducing the descriptor's dimensionality—all without sacrificing important information. Here's the code snippet:

```
orientation = (
    np.arctan2(sobely, sobelx) * 180 / np.pi
)  # convert to degrees
# Normalize orientation to [0, 180] because direction of
# an angle is ambiguous up to 180.
norm_orientation = np.mod(orientation + 180, 180)
bin_width = 180 / ang_quant_lvl
```

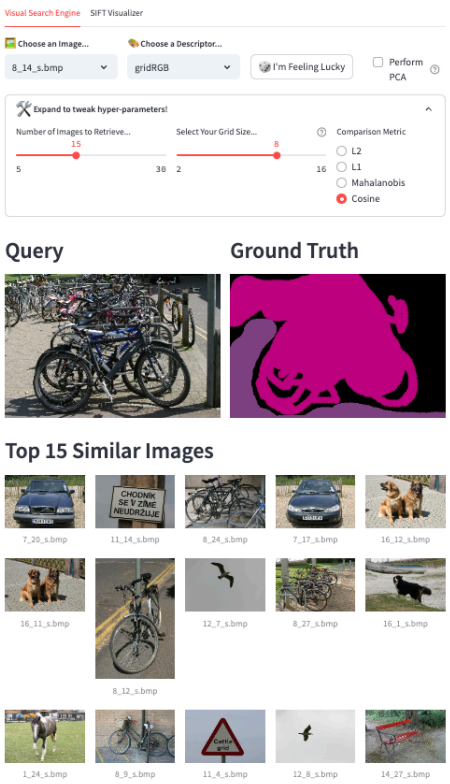
▼ **Experimental Results for Grid-based Color Descriptor**

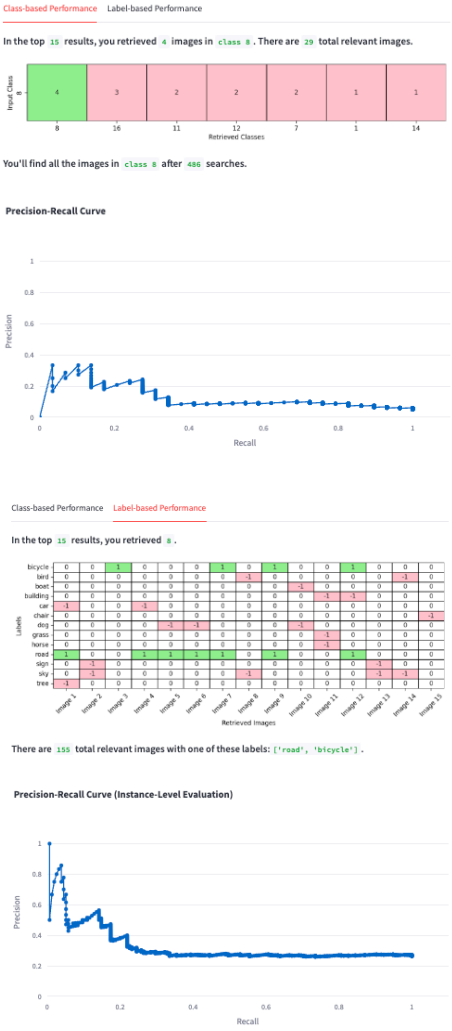
▼ **10_8_s.bmp** (🟢 with 📐)



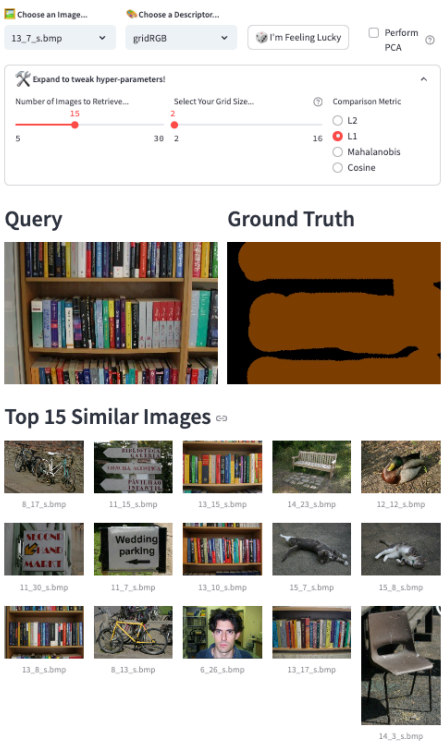


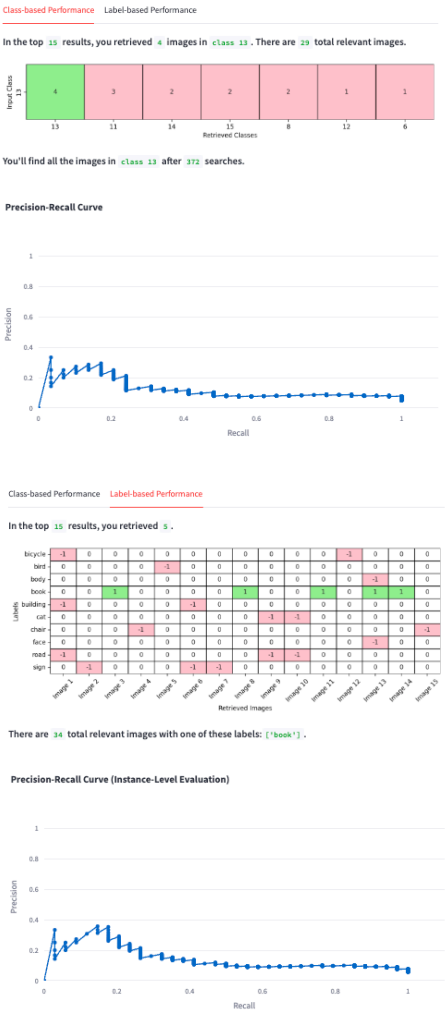
▼ **8_14_s.bmp** (🔴 with 📐)





▼ **13_7_s.bmp** (🔴 with 🚙)

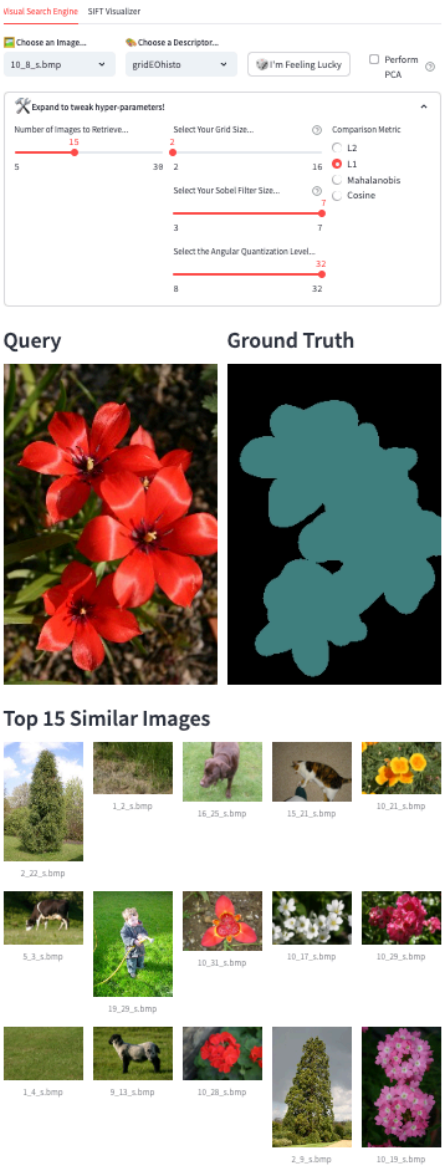


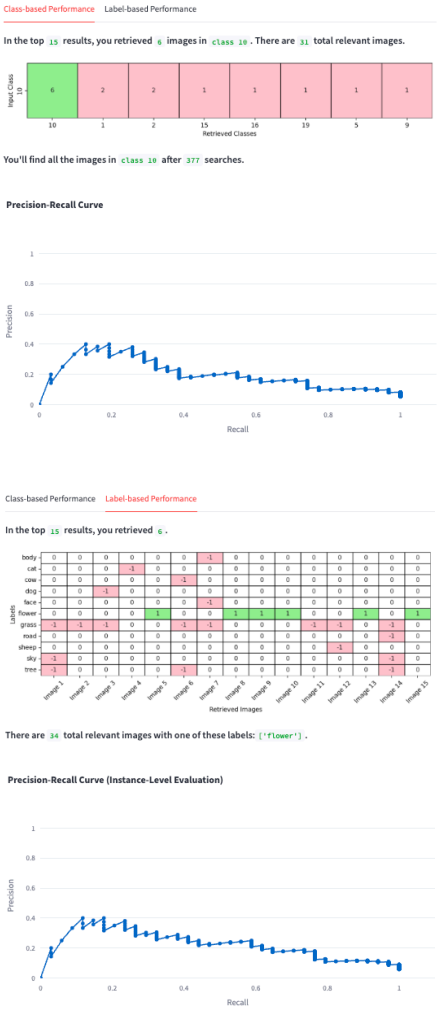


Compared to the color histogram and the generic color descriptors, we're already seeing significant improvements with the grid-based color approach. The interesting thing to note here is that **cosine similarity actually yields us the best retrieval results** for the flower image. This is most likely due to its ability to capture the overall color distribution pattern, regardless of intensity variations. In images with varying lighting conditions or global intensity shifts, absolute color values may differ, but the relative color proportions often remain consistent. Cosine similarity focuses on the angle between vectors, which effectively compares the proportions of colors rather than their absolute values, making it more robust to lighting changes and global intensity shifts. Finally, for the textbook and bike images, it's not surprising that color doesn't perform well because it's heavily textured.

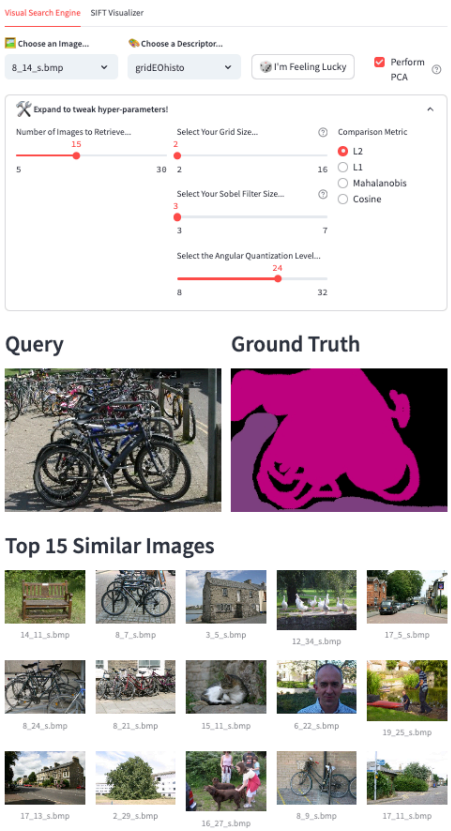
▼ **Experimental Results for Grid-based Edge Orientation Histogram**

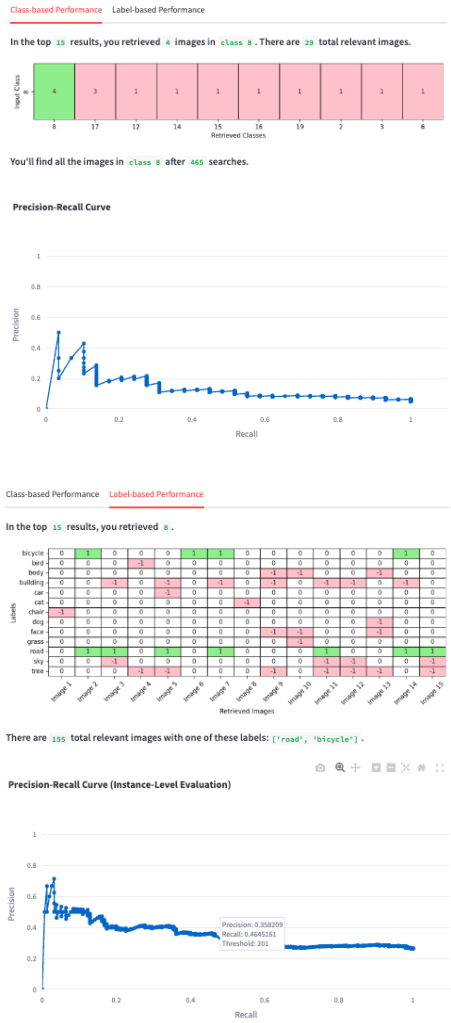
▼ **10_8_s.bmp** (🟡 with 🚙)


Visual Search Engine
SIFT Visualizer
Choose an Image...
10_8_s.bmp
Choose a Descriptor...
gridEOhisto
I'm Feeling Lucky
Perform PCA
Expand to tweak hyper-parameters!
Number of Images to Retrieve...
15
5
30
Select Your Grid Size...
2
2
16
Comparison Metric
L2
L1
Mahalanobis
Cosine
Select Your Sobel Filter Size...
7
3
7
Select the Angular Quantization Level...
32
8
32
Query
Ground Truth
Top 15 Similar Images
2_22_s.bmp
1_2_s.bmp
16_25_s.bmp
15_21_s.bmp
10_21_s.bmp
5_3_s.bmp
19_29_s.bmp
10_31_s.bmp
10_17_s.bmp
10_29_s.bmp
1_4_s.bmp
9_13_s.bmp
10_28_s.bmp
2_9_s.bmp
10_19_s.bmp

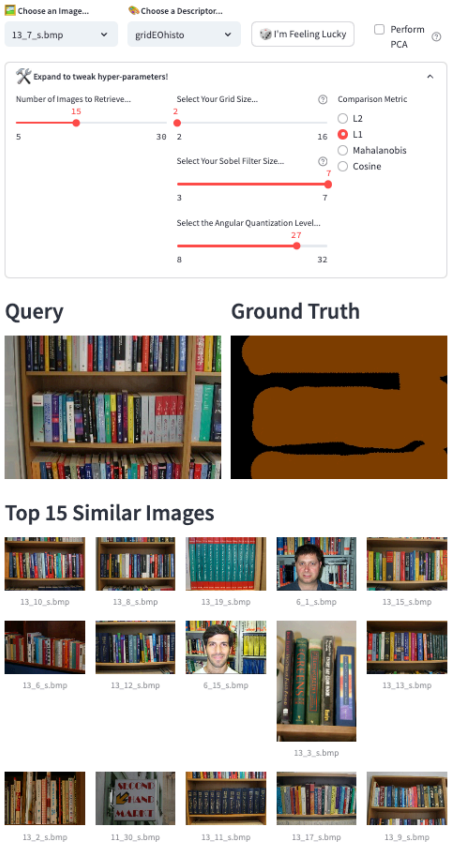

Class-based Performance
Label-based Performance
In the top 15 results, you retrieved 6 images in class 10. There are 31 total relevant images.
Input Class 10
Retrieved Classes
You'll find all the images in class 10 after 377 searches.
Precision-Recall Curve
Precision
Recall
Class-based Performance
Label-based Performance
In the top 15 results, you retrieved 6.
Labels
Retrieved Images
There are 34 total relevant images with one of these labels: ['flower'].
Precision-Recall Curve (Instance-Level Evaluation)
Precision
Recall

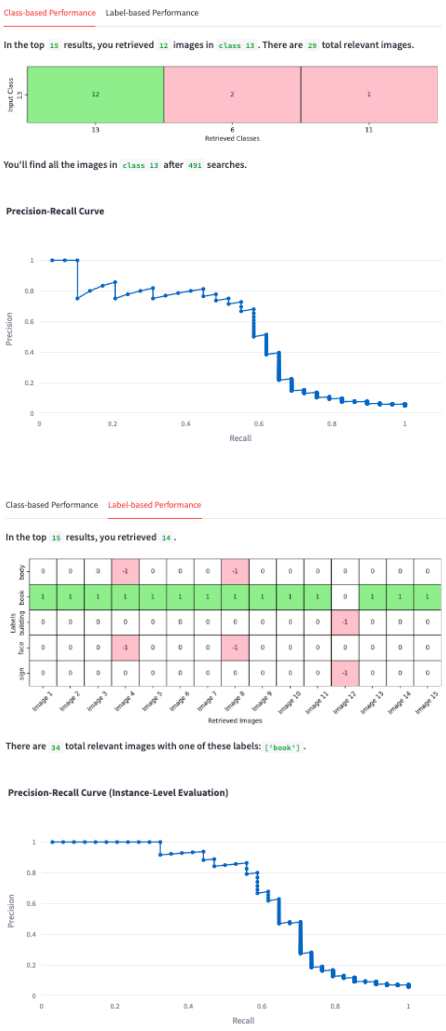
▼ **8_14_s.bmp** (🔴 wth 🔪)





▼ **13_7_s.bmp** (🟢 with 🚕)





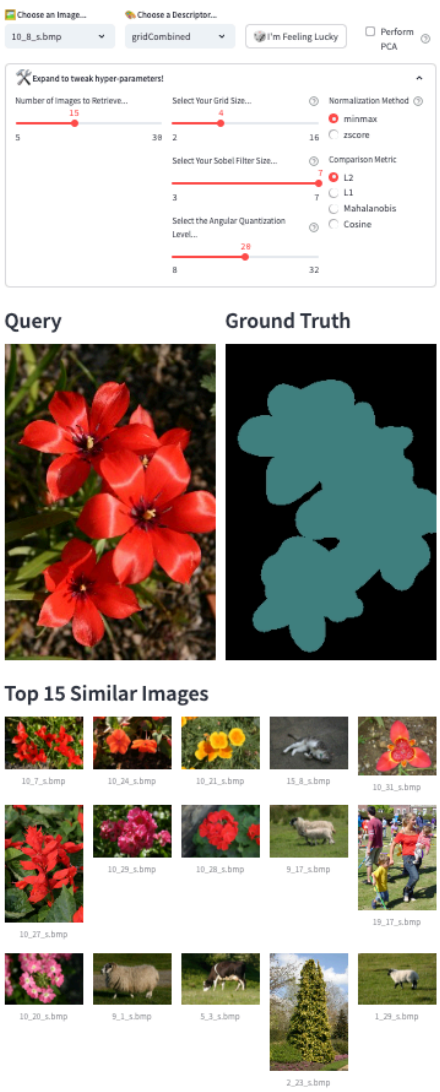
It's expected for our edge descriptors to have terrible retrieval for images with vibrant color. Hence the poor performance in the flower image. However, I am a bit surprised by the poor performance of the bike image as well. I think this is due to the cluttered bike rack and overlapping bikes obscuring the clear edge patterns. This descriptor redeems itself with the last image retrieval of bookshelves. This probably attributes to the structural patterns that align perfectly with what the edge orientation histogram captures.

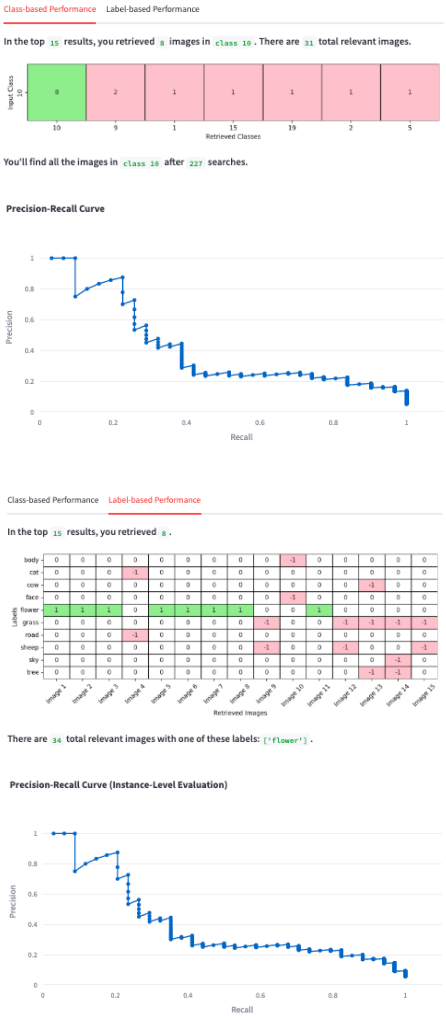
Combining Grid-based EOH with Grid-based Color ➕

These two descriptors can be combined to create a hybrid descriptor that captures both color and structural information. I've implemented this hybrid approach in the `extract_gridCombined` function, which you can find here and you can find the experimental results below.

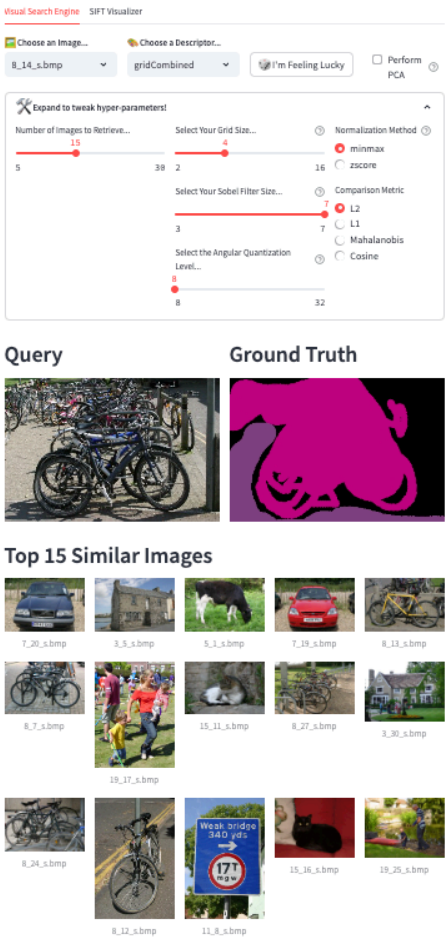
▼ **Experimental Results for Combined**

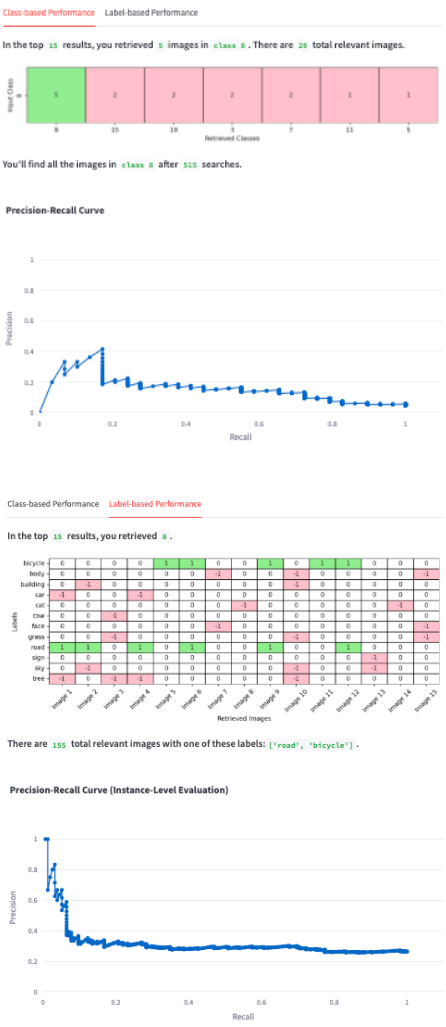
▼ **10_8_s.bmp** (🟢 with 🔪)



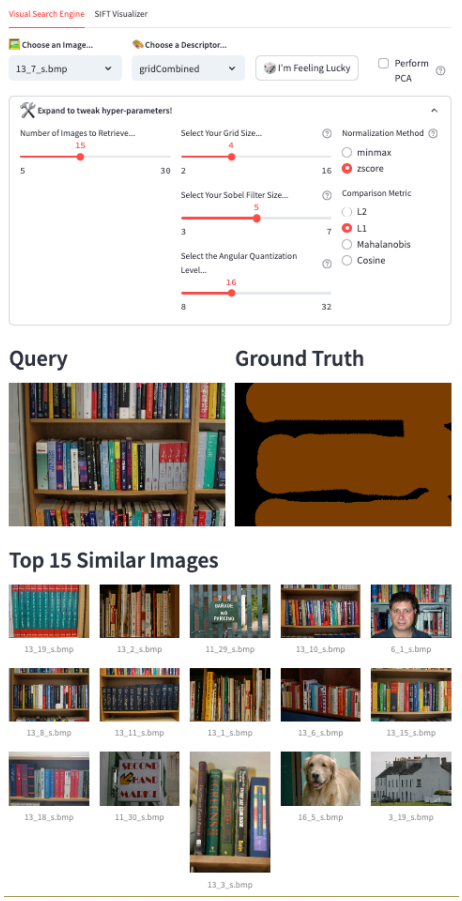


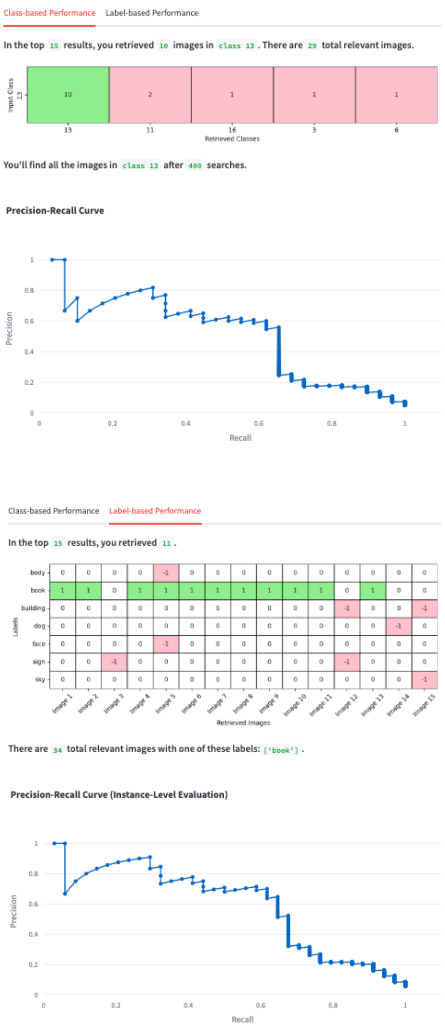
▼ **8_14_s.bmp** (🟡 with 🔪)





▼ **13_7_s.bmp** (🟢 with 🚓)

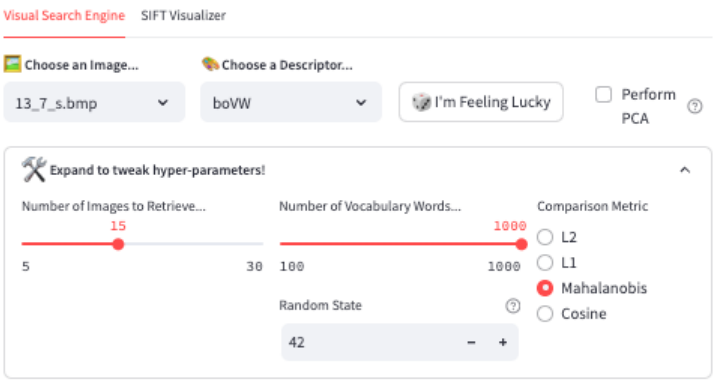




As expected, the combined approach shows improvement across the board for all test images because it combines the textual details and the color gradients from the two descriptors above. Another interesting thing to point out through all three experiments is that **L1 distance still dominates with our bookshelf pictures.** This is probably because bookshelves often produce **sparse feature vectors**, especially with edge orientation histograms, with a heavy focus on vertical edges and horizontal edges (spines and shelves).

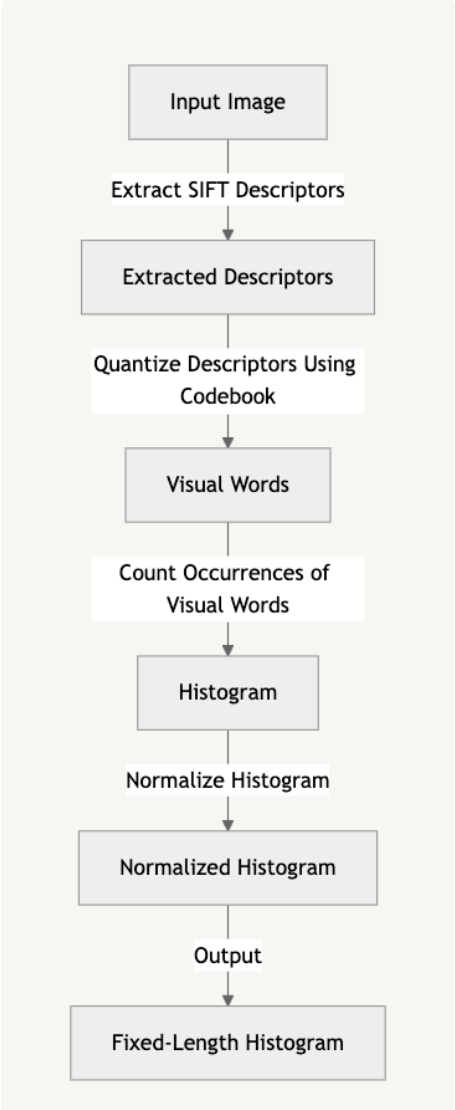
BoVW (Bag of Visual Words) Retrieval 👜

The Bag of Visual Words (BoVW), covered in lectures 8 and 9 [2], represents **images as collections of visual words**. It's important to point out that **the vocabulary size in BoVW represents the potential features that can be encoded in images, not the actual number of visual features present in the dataset**. Increasing the vocabulary size allows for capturing more fine-grained details and distinctions between images. However, this also ramps up computational complexity and may lead to overfitting if the vocabulary becomes too large. The optimal vocabulary size often hinges on the specific dataset. In our experiment, I've provided it as a hyper-parameter we can tune from 100 to 1000.



Hyper-parameters for our BoVW model

I'll now thoroughly explain the implementation and provide an illustration to walk through each step from feature extraction to the final histogram representation. You can check out my implementation of each stage in the `bovw.py` here [1].



The Bag of Visual Words Pipeline

1. **Feature extraction**

   At this stage, the image is processed to detect key points using SIFT. For each key point, a descriptor of dimension (128,) is extracted that encodes the local gradient patterns around a key point.

   In my implementation, I have opted for the SIFT descriptors because **it is robust to changes in scale, rotation, and lighting**.

   You can see the implementation in the `extract_sift_features` function here.

2. **Quantize descriptors using codebook**

   Next, we create a pre-trained codebook, which is the visual vocabulary used to cluster the high-dimensional feature space. I then pickled the k-means clustering model to be re-used later. You can see the code here. The codebook is a reference of all the features presented in the images, such as “lighting, texture, edges and corners, gradient patterns" and so on. Each descriptor is then assigned to its nearest cluster center in the codebook, effectively quantizing the continuous feature space into discrete visual words.

3. **Count occurrences of visual words**

   This step involves representing the visual words from the previous stage as a histogram. Each bin in the histogram counts the occurrences of a specific visual word (feature) in the image. This effectively captures the overall distribution of

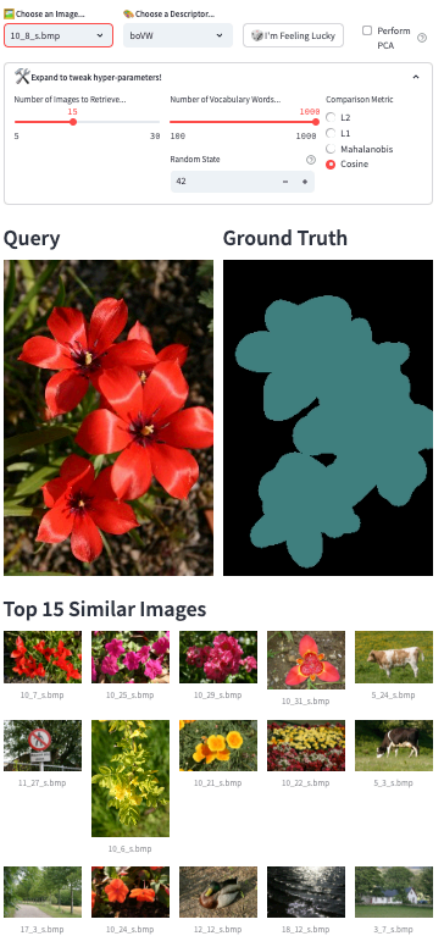
visual patterns in the image, forming our descriptor. You can see the code in my `build_histogram` function here.

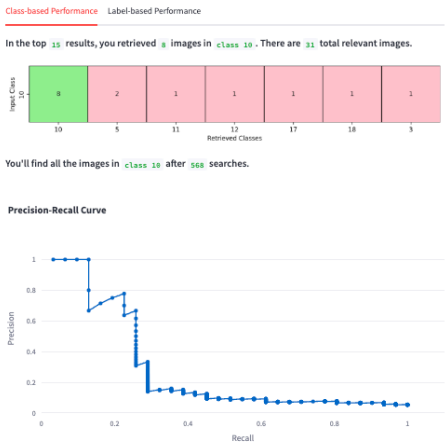
4. **Outputting a normalized histogram**

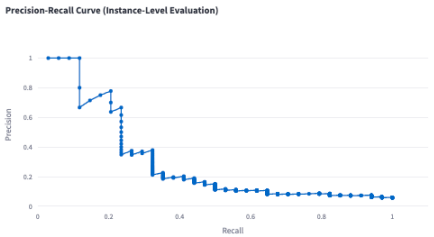
   This part is self-explanatory. Without normalization, larger images or those with more key points would have higher histogram values, leading to unfair comparisons. We're doing this to ensure that the histogram is **scale-invariant**.

▼ **Experimental Results for BoVW**

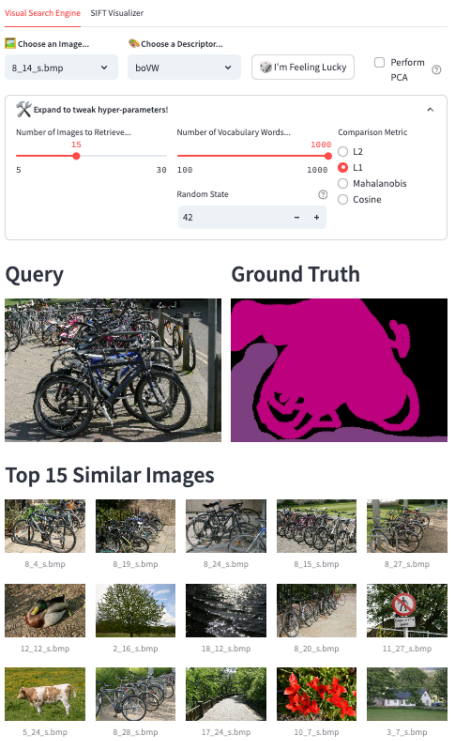
▼ **10_8_s.bmp** (🟡 with 📐)



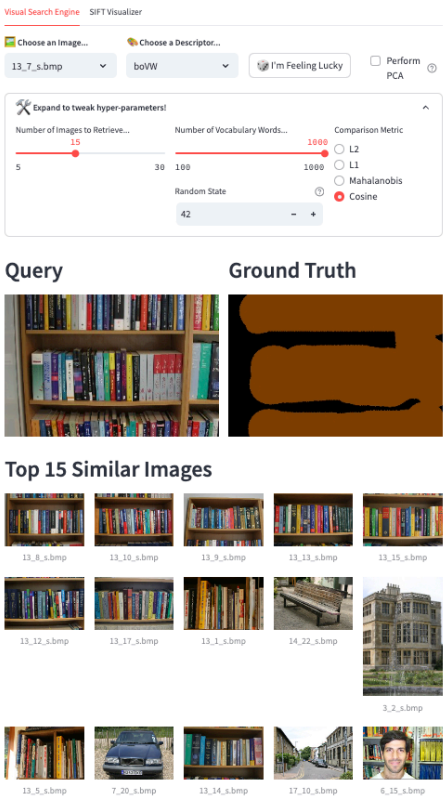


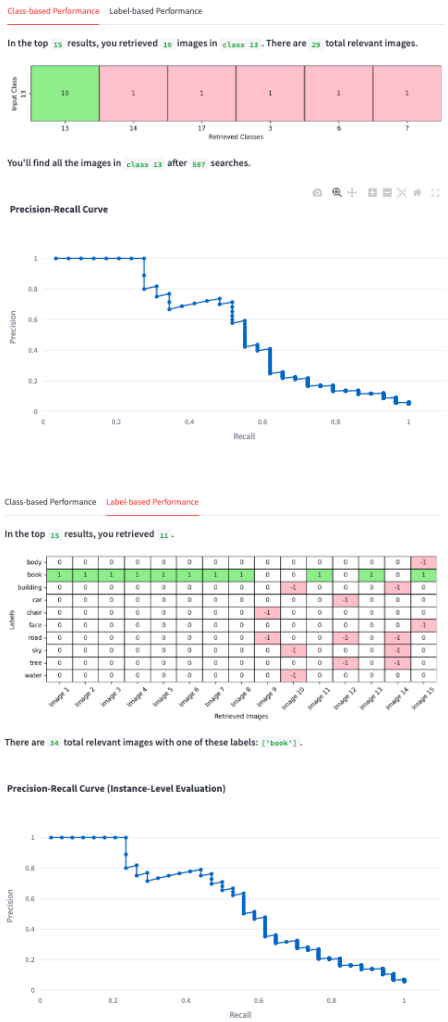


▼ **8_14_s.bmp** (🟡 with 🚕)



▼ **13_7_s.bmp** (with)





Our experiments shows that the BoVW model manages the dataset's variability pretty well. It has moderate to good performance across all the test images. This generalization stems from the codebook's representation of the most common and important patterns throughout the dataset.

Use of PCA ⬇️

We apply Principle Component Analysis for two reasons.

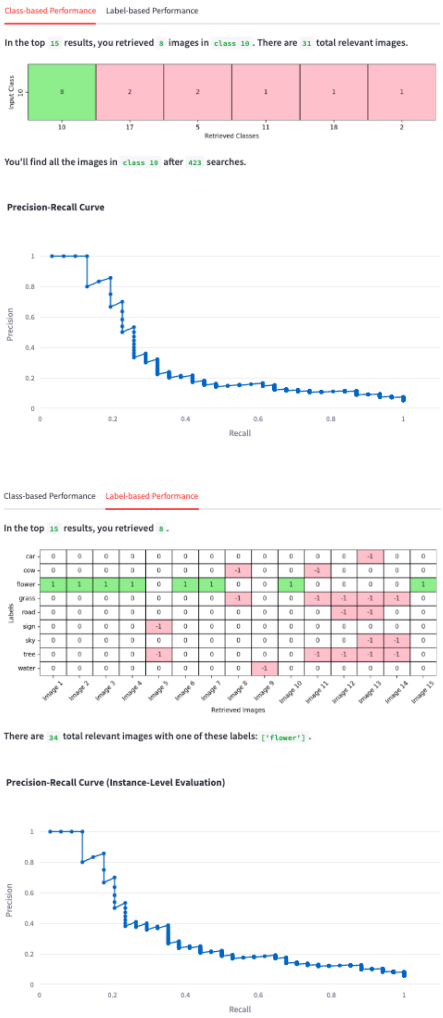
1. Given that the dimensions of the descriptor can be very large, as we see from the `gridCombined` descriptors and the 1000 vocabulary `BoVW-generated` descriptors, it gets computationally expensive.
2. Not all dimensions of the descriptors contributes to the overall image representation. Some dimensions may contain redundant or less informative data.

We can reduce the complexity of the descriptors by identifying the principal components that accounts for the most variance in the data. This not only reduces the length of the descriptor, but it also gets rid of potential noise and irrelevant information in the descriptor.

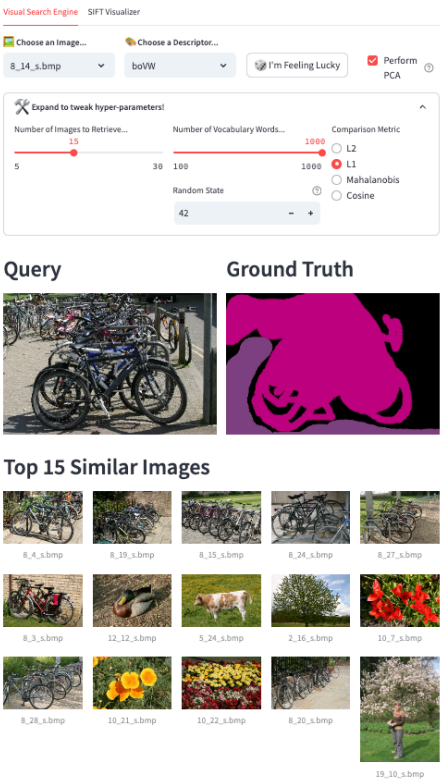
In my implementation, I've used the `PCA` function from the `sklearn.decomposition` library. My `perform_pca` function offers the flexibility to use either `variance_ratio` or `n_components`. By default, I've set it to retain 99% of the original variance.You can see the implementation here. To better demonstrate the benefits of PCA, I'll apply it to the BoVW descriptors with a vocabulary size of 1000.

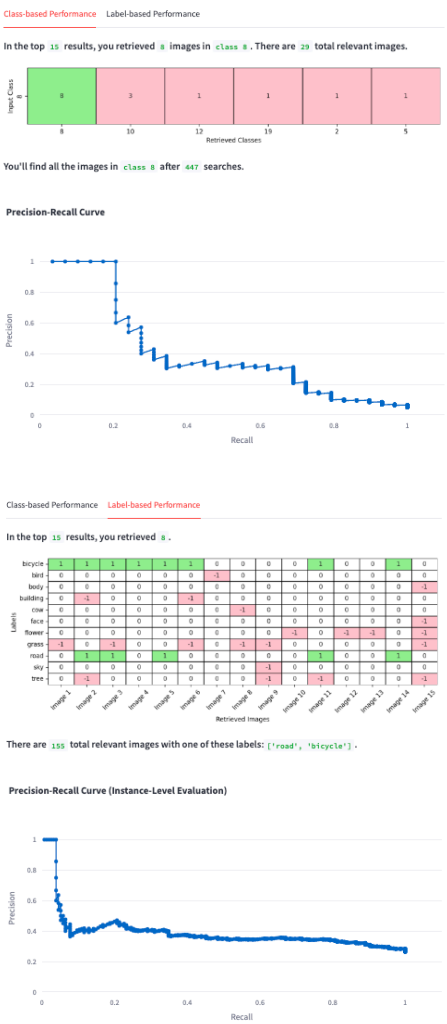
▼ **Experimental Results for PCA on BoVW**

▼ **10_8_s.bmp** (🟡 with 📐)

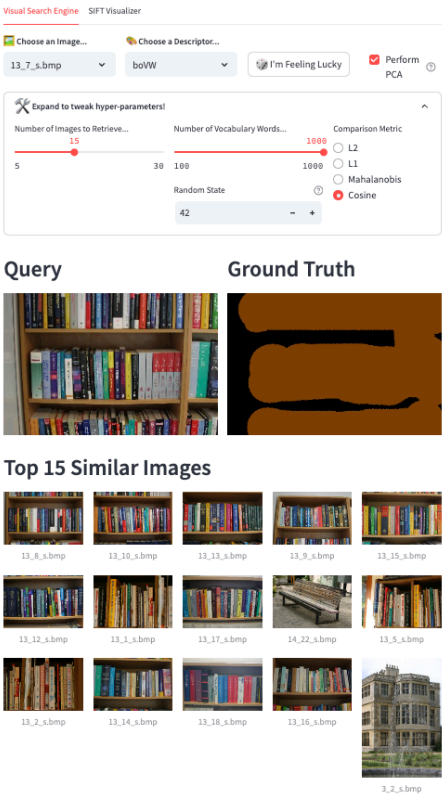


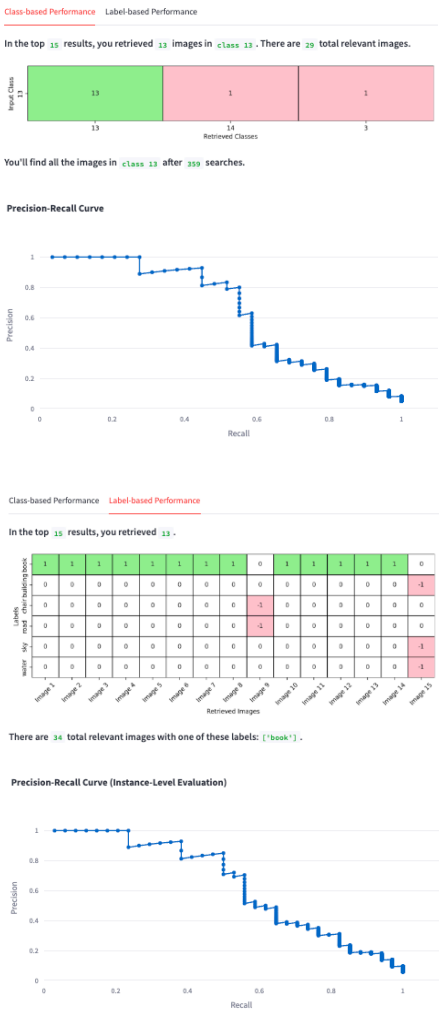
▼ **8_14_s.bmp** (🟡 with 🚕)





▼ **13_7_s.bmp** (with)





Our experiments reduced the descriptor size from **1000** to a mere **370**. This change made the GUI noticeably more responsive when measuring similarity (less computational computational). We also observed improved retrieval performance in class-based evaluations across all test images:

- `10_8_s.bmp` increased the retrieval count by 1
- `8_14_s.bmp` increased the retrieval count by 1
- `13_7_s.bmp` increased the retrieval count by 3

We also saw improvements in the label-based evaluation:

- `10_8_s.bmp` largely retained the same performance
- `8_14_s.bmp` retrieved the same number of images, but with better quality (more relevant images at the front of the retrieval)
- `13_7_s.bmp` increased by 2

This corroborates our earlier statement. **PCA is highly effective when applied to large-dimensional descriptors.**

Conclusion 🗣️

Here's a summary table of the descriptors' performance on the three test images, along with the best performing distance metric.

Legend:

🟢 Good results | 🟡 Moderate results | 🔴 Poor results | 🚕 L1 norm | 📏 L2 norm | 🌭 Mahalanobis | 📐 Cosine

Descriptor	10_8_s.bmp (Flower)	8_14_s.bmp (Bike)	13_7_s.bmp (Bookshelf)
Color Descriptor	🟡 with 🚕	🔴 with 🚕	🟡 with 🌭
Quantized Color Histogram	🟡 with 📏	🟢 with 📏	🟢 with 🚕
Grid-based Color	🟢 with 📐	🔴 with 📐	🔴 with 🚕
Grid-based EOH	🟡 with 📏	🔴 with 📏	🟢 with 🚕
Combined Grid-based Color and EOH	🟢 with 📏	🟡 with 📏	🟢 wth 🚕
BoVW without PCA	🟡 with 📐	🟡 with 🚕	🟢 with 📐
BoVW with PCA	🟡 with 📐	🟡 with 🚕	🟢 with 📐

And here's a visual summary of my mental state as I approach the end of this report.

7 Ways the Living Tried to Keep the Dead in Their Graves by Little [7]

This summary table provides a quick overview of how each descriptor performed on the three test images.

A couple insights can be gathered from this summary table.

1. If we were to rank overall generalization for all the visual search techniques, we would likely see a progression from simpler methods to more complex ones, with an increasing semantic content. The basic color descriptors would be at the lower end, followed by color histograms, then spatial grid methods combining color and texture. The Bag of Visual Words (BoVW) approach would rank higher due to its ability to capture more abstract visual features.
2. Through there are obvious pairings that work really well, such as BovW with cosine similarity and L1 distance metric with bookshelf pictures, It's clear that different descriptors have varying levels of success depending on the image and the distance measure chosen.

3. The hyper-parameters seems extremely arbitrary depending on the image, descriptor, and distance metric chosen. **In other words, it's challenging to find a universal descriptor and distance metric combination that works effectively across a diverse set of images.** Deep learning methods like pre-trained residual networks would likely offer the best generalization, especially when applied to diverse image datasets. I have implemented a couple pre-trained ResNet models in the extra credit section.s

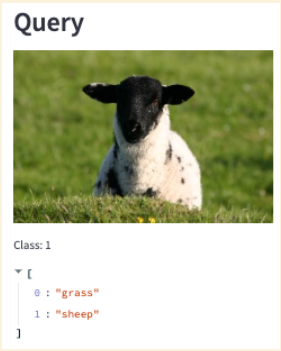
So here, I will leave you with **ANOTHER summary table of the pros and cons of each descriptor based on our long and arduous experiments that should probably end 2000 words ago.**

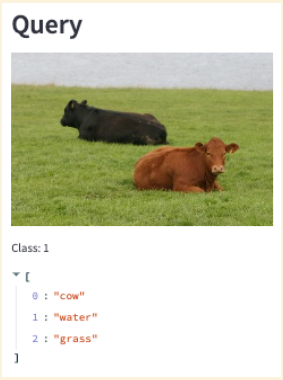
Descriptor	Pros	Cons
Basic Color Descriptor	• Simple to implement • Fast computation • Low memory requirement	• Limited discriminative power • Sensitive to lighting changes • Ignores spatial information
Color Histogram	• Captures color distribution • Rotation and scale invariant • Relatively simple to implement	• Loses spatial information • Sensitive to lighting changes • Can be high-dimensional
Grid-based Color Histogram	• Preserves some spatial information • Better discriminative power than global histograms • Flexible grid size	• Higher dimensionality • More sensitive to translation • Increased computation time
Edge Orientation Histogram	• Captures structural information • Less sensitive to lighting changes • Good for texture-rich images	• May struggle with smooth areas • Sensitive to noise • Requires edge detection step
Combined Color and Edge Descriptor	• Captures both color and structural information • More comprehensive representation • Can improve overall performance	• Higher dimensionality • Increased computation time • Requires balancing between color and edge information
Bag of Visual Words (BoVW)	• Captures local features • Scale and rotation invariant • Good for object recognition	• Loses spatial relationships • Requires codebook generation • Can be computationally expensive
Pre-trained Residual Network	• Highly discriminative features • Learns hierarchical representations • Can generalize well to various tasks	• Requires significant computational resources • Large model size • May need fine-tuning for specific tasks

This table summarizes the main advantages and disadvantages of each descriptor used in our visual search system. As we progress from simpler methods to more complex ones, we generally see improved performance and generalization capabilities, but at the cost of increased computational complexity.

In addition, We also need to be reminded that our given dataset is not perfect. There are more than a dozen images that are poorly classified as you can see in the callout below.

> ❗ **Limitation of our Dataset**
> The `ClickMe.html` file notes that there are very few examples of horses and mountains, so we'll exclude those labels. In addition, I've discovered some misclassified classes. For example, **images `1_27_s.bmp` through `1_30_s.bmp` are clearly sheep images and should belong to class `9`, yet they're incorrectly placed in class `1` with cows**.





Essentially, we can conclude that the principle of "Garbage In, Garbage Out" (GIGO) applies here.

if you `try {feeding poorly labeled dataset to the visual search engine}`, you'll probably `catch { these hands garbage results}`. It's important to note that the quality of results from a visual search engine is heavily dependent on the quality of the dataset used. In this case, the limitations of our dataset, rather than the implemented techniques, may be responsible for any suboptimal results. Even the most sophisticated algorithms can't compensate for fundamental issues in the underlying data.

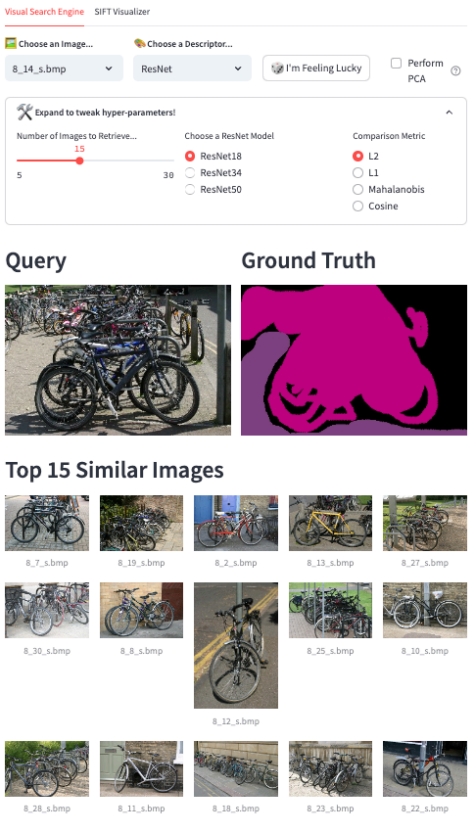
Finally, I have to really attribute the success of running these experiments to having built a user interface. It significantly simplifies hyper-parameter tuning and makes generating screenshots much less of a hassle. So going back to the introduction, I really do hope this open-sourced GUI project becomes a valuable learning resource and reference for future students.

Extra Credit ✨

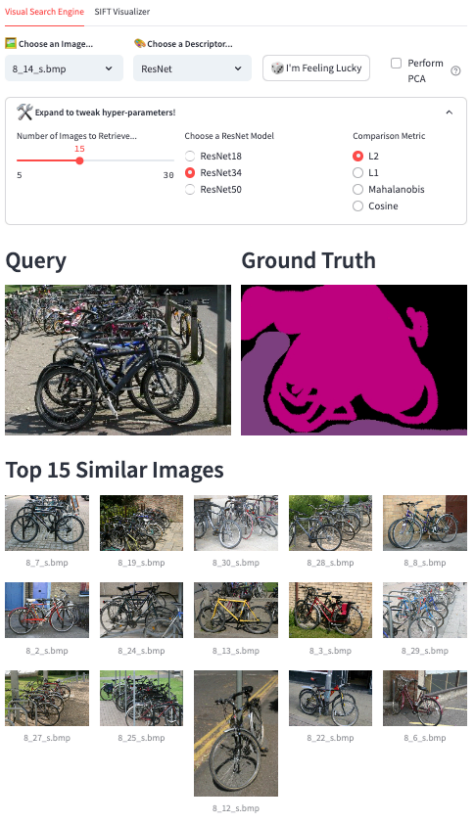
ResNet Implementation 🧠

I really don't like how some of the PR curves look and the **lack of generalization** of different descriptors. I want to find **the one descriptor to rule them all** 💍. So I went ahead and tried out ResNet50, ResNet34, and ResNet18 on the image we performed the worst in the above experiments (the bike image `8_14_s.bmp`). You can find the code here[1] and see the results below. Be amazed.

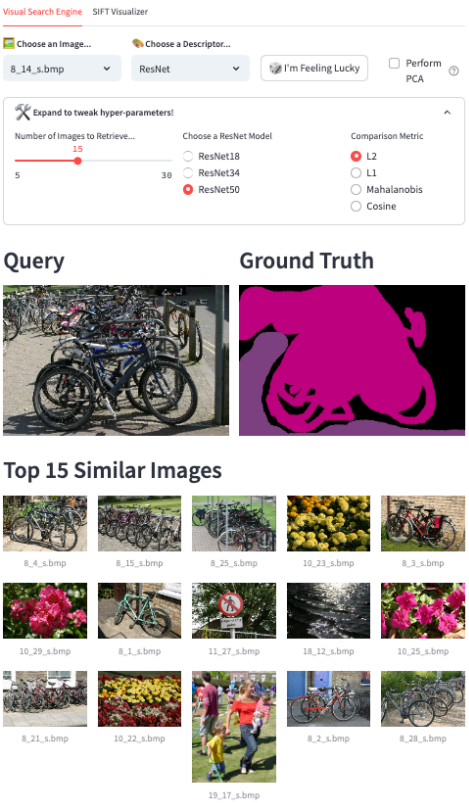
▼ **FINAL Experimental Results with ResNet18** (🟢 with all metrics besides 🌭)

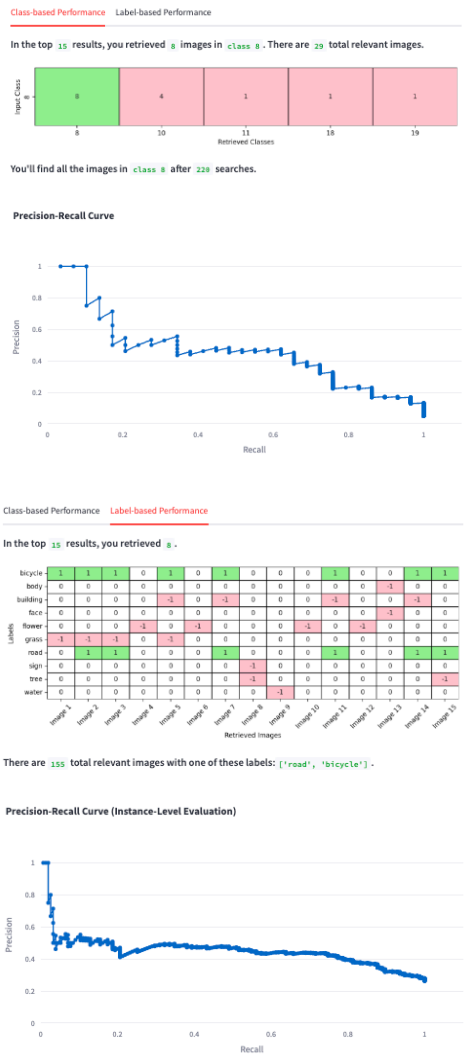


▼ **FINAL Experimental Results with ResNet34** (🟢 with all metrics besides 🌭)



▼ **FINAL Experimental Results with ResNet50** (🟡 with all metrics besides 🌭)





ResNet34 outperforms the other two models, with ResNet18 following closely. Impressively, ResNet34 retrieves all class images **in just 29 searches**. Given our relatively small dataset of 591 images, the risk of overfitting increases with more complex models like ResNet50. The Mahalanobis distance metric underperforms due to a distribution mismatch. The pre-trained ResNet features are based on ImageNet, not our MSRC dataset, resulting in de-correlated features.

Applying TF-IDF to BoVW

Term Frequency and Inverse Document Frequency is commonly used in Bag of Words approaches in NLP. During the lecture, I raised the question of whether TF-IDF could be applied to BoVW and never got an answer, so I'm going to try it. To break this concept down, we need to define the term frequency and the inverse document frequency first.

1. **Term Frequency (TF):** Measures the frequency of visual words in an image. This is already computed in our histogram normalization using the `build_histogram` method.
2. **Inverse Document Frequency (IDF):** Reduces the weight of common visual words that appear in many images. This should help highlight discriminative features.

There are two things we're trying to achieve:

1. Mitigate the impact of common visual patterns that appear across multiple images. For instance, backgrounds like grass or similar textures found in various object classes. A prime example is the texture patterns shared among grass, trees, and bushes, which could be overrepresented in our visual vocabulary.
2. Emphasize unique features specific to a class or image.

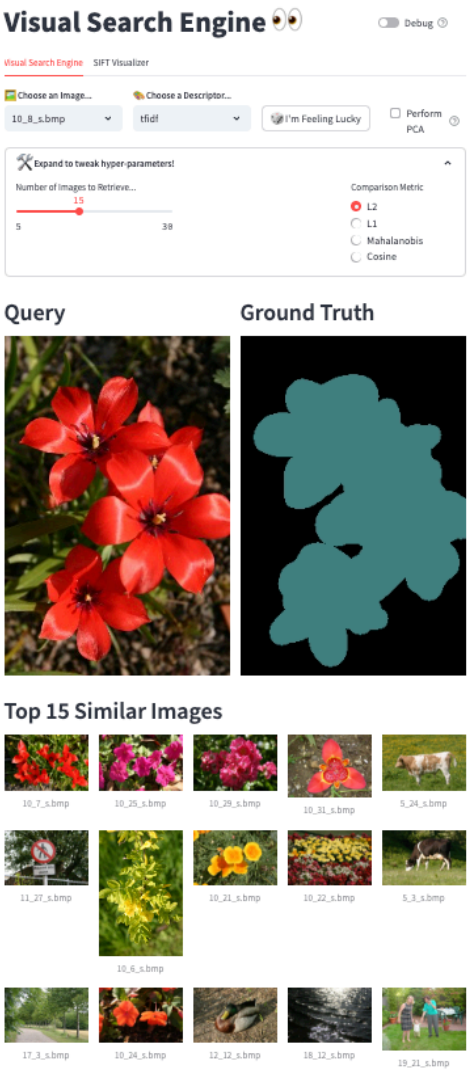
You can see the implementation in this pull request here [1]:

> https://github.com/frankcholula/cvpr/pull/4/files

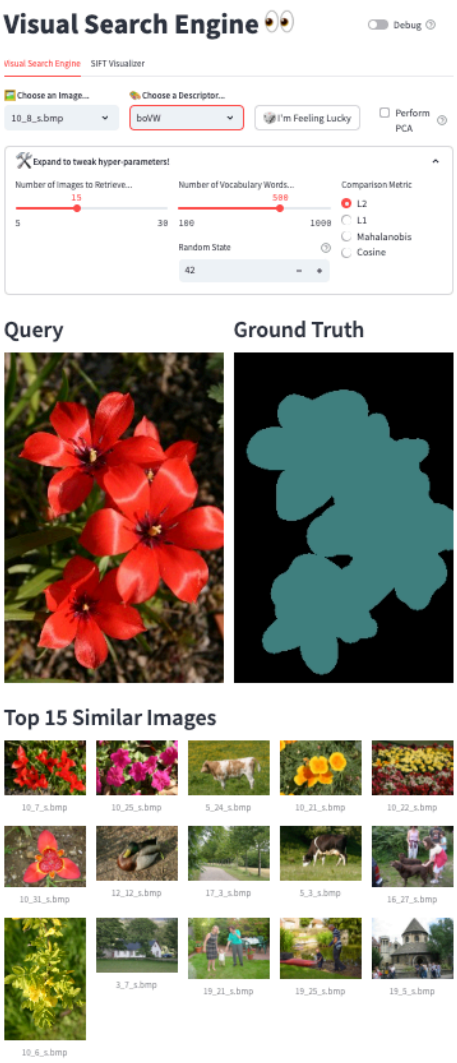
I've also performed a quick comparison using a Bag of Visual Words model with a vocabulary size of 500.

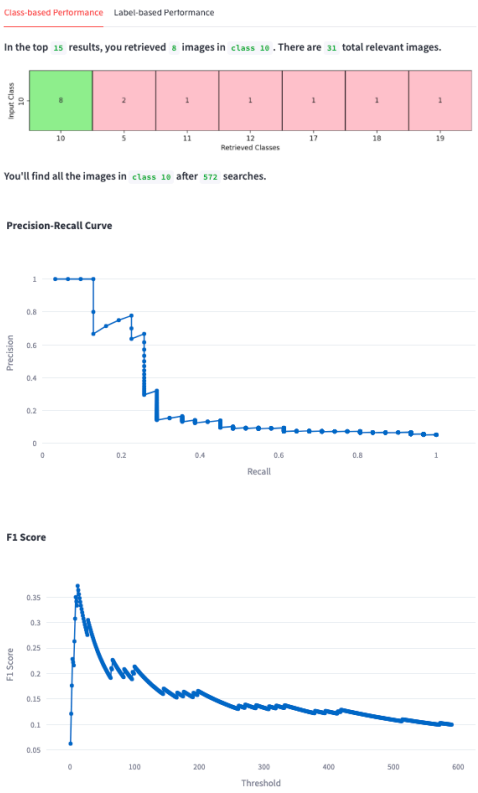
▼ **Experimental Results with BoVW with TF-IDF vs BoVW alone** (🟢 with L2)

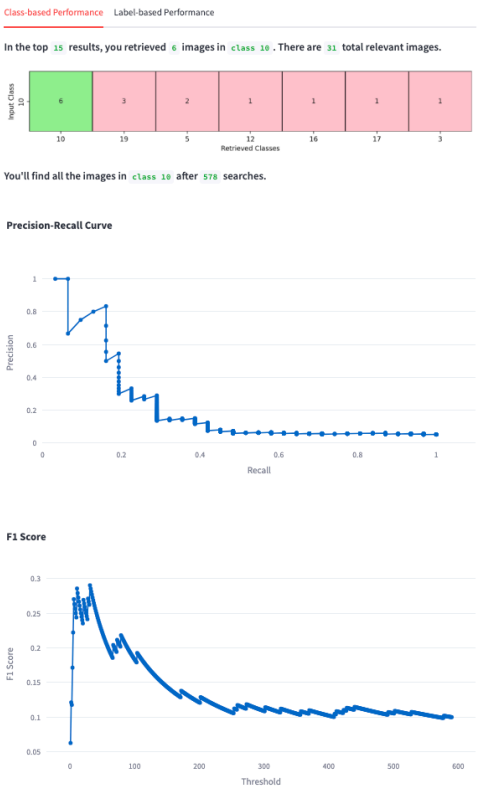
Apply TF-IDF on BoVW with `10_8_s.bmp`



No TF-IDFs on BoVW with `10_8_s.bmp`





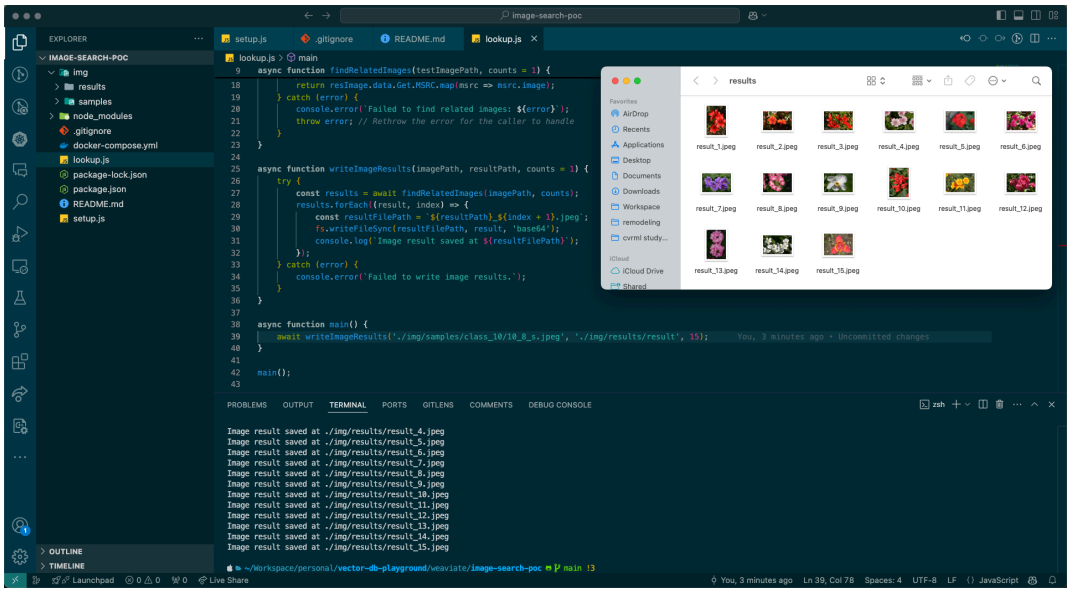


We observe a +2 retrieval improvement for the TF-IDF version compared to the standard BoVW! That's pretty impressive for such a simple addition.

Vector Database and HNSW Index 🔎

The image retrieval task is highly relevant to this project, so I have implemented a quick demo showcasing the HNSW index for similar image retrieval. You can find the code for this demo here.

https://github.com/frankcholula/vector-db-playground



In the demo, I inserted all the images from the 20 classes into the open-source vector database `Weaviate`, generated embeddings using `image2vec`, and executed a query using an the input image `10_8_s.jpeg` and the built-in Hierarchical Navigable Small World algorithm [5].

The TL;DR of the HNSW algorithm is that it's an **approximate nearest neighbor search** algorithm for high-dimensional spaces. It cleverly builds on the **small-world graph** concept, creating a data structure that enables lightning-fast searches by harnessing **hierarchical layers** of connectivity. I won't dive into the details here, but if you're curious, there's a detailed breakdown in this article here by *Pinecone* [4].

References

Honestly, I didn't rely on many references 🤷. Most of the key concepts are covered thoroughly in the lecture materials by Professor Boeber and the "First Principles of Computer Vision" video series by Professor Nayar at Columbia University.

[1] **Code** by *Frank Lü*, University of Surrey

> https://github.com/frankcholula/cvpr

> https://github.com/frankcholula/vector-db-playground

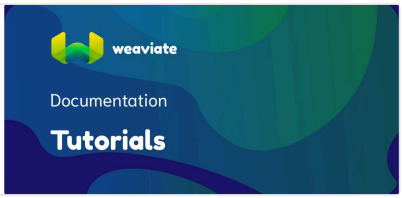
[2] **Computer Vision and Pattern Recognition Lecture 7-9** by *Miroslaw Bober*, University of Surrey CVSSP

[3] **First Principles of Computer Vision Video Series** by *Shree Nayar*, Columbia University

[4] https://www.pinecone.io/learn/series/faiss/hnsw/ by *Pinecone,* https://www.pinecone.io/

[5] **The Quickstart and Tutorial Documentation** by *Weaviate*, https://weaviate.io/

Tutorials | Weaviate
Explore Weaviate tutorials for practical guidance on data management and queries.
https://weaviate.io/developers/weaviate/tutorials



[6] **MSRC-v2 Image Database** by *Microsoft Research Cambridge,* https://www.microsoft.com/en-us/download/details.aspx?id=52644

Download Microsoft Research Cambridge Object Recognition Image Database from Official Microsoft Download Center
The Microsoft Research Cambridge Object Recognition Image Database contains a set of images (digital photographs) grouped into categories. Last published: May 18, 2005.
https://www.microsoft.com/en-us/download/details.aspx?id=52644

[7] **7 Ways the Living Tried to Keep the Dead in Their Graves** by *Becky Little*, https://www.history.co.uk/

7 Ways the Living Tried to Keep the Dead in Their Graves | HISTORY
Historical graves reveal corpses with iron rods through their chests, sickles across their necks and padlocks on their feet.
https://www.history.com/news/graves-skeletons-burials-superstitions

[8] **iStock Photo of a Fresh hand squeezed orange juice** by ***patagonia20***, https://www.istockphoto.com/

Fresh hand squeezed orange juice with plenty of natural light.

https://www.istockphoto.com/photo/hand-squeezed-orange-juice-gm465040073-33031686