SIFTastic: Unveiling Reality: SIFT Applications for Visual Classification, Adversarial Detection, 3D Reconstruction, and Fingerprint Matching

Course Project

Computer Vision (CSL7360)

Team Members :

Gaurav Sangwan (B20AI062)
Mukul Shingwani (B20AI023)
Shashank Shekhar Asthana (B21CS093)
Anushkaa Ambuj (B21ES006)

Under Supervision of

Prof. Pratik Mazumdar

॥ त्वं ज्ञानमयो विज्ञानमयोऽसि ॥

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY
JODHPUR - 342037

Objective and Introduction

In the computer vision and image processing field in seeking to solve problems such as image classification and 3D reconstruction, the SIFT algorithm has proven to be one of the most useful tools since it provides a wide scope of applications. Building on the solidity of SIFT, this entry analyzes thoroughly its multiple functions, and describes the original approach and findings around how the method is put into multiple real-life contexts.
Image classification takes off now with the conclusion of using SIFT and FeatureExtraction (BoVW) as it provides us with powerful tools for image classification. The training of the classifier by this approach ensures not only high accuracy of classification but also it creates the basis for developing resilient image recognition systems.
Searches for available spaces may go beyond just images. Image matching using live webcam can also be conducted if there is a need for a real-time image comparison with a reference image. Additionally, this capability allows these systems to carry out a broad range of applications in augmentative reality, reality tracking, and many other areas.

On the other hand, we study SIFT's capabilities in 3D reconstruction and localization, a field where its ability in detecting conspicuous keypoints from multiple viewpoints emerges as the main pillar for the creation of 3D points with absolute fidelity and efficiency. Although there are numerous learnings, these achievements may bring benefits to virtual reality, self-navigation, and architectural modeling.
The SIFT technique can be used in the area of security and forensics as a powerful tool of adversarial image detection that leads to discovering photoshopped and forged images through cheat in keypoint distribution and descriptors.
Lastly, we scrutinize the usage of SIFT in fingerprint identification by emphasizing its robust feature selection procedure, which serves as the basis of accurate and dependable biometric identification systems that are widely utilised in different fields towards security purposes.
This article aims to outline the extent of the SIFT algorithm throughput across many different areas, the outcomes which it can potentially bring both for researchers, practitioners and also people who like to learn.

SIFT for Visual Classification

Introduction

Bag of Visual Words (BoVW) is a popular technique in computer vision that represents images as orderless collections of visual words, inspired by the bag-of-words model used in natural language processing. This method allows for robust image representation and classification by capturing the essential features of the image while being invariant to their spatial arrangement.

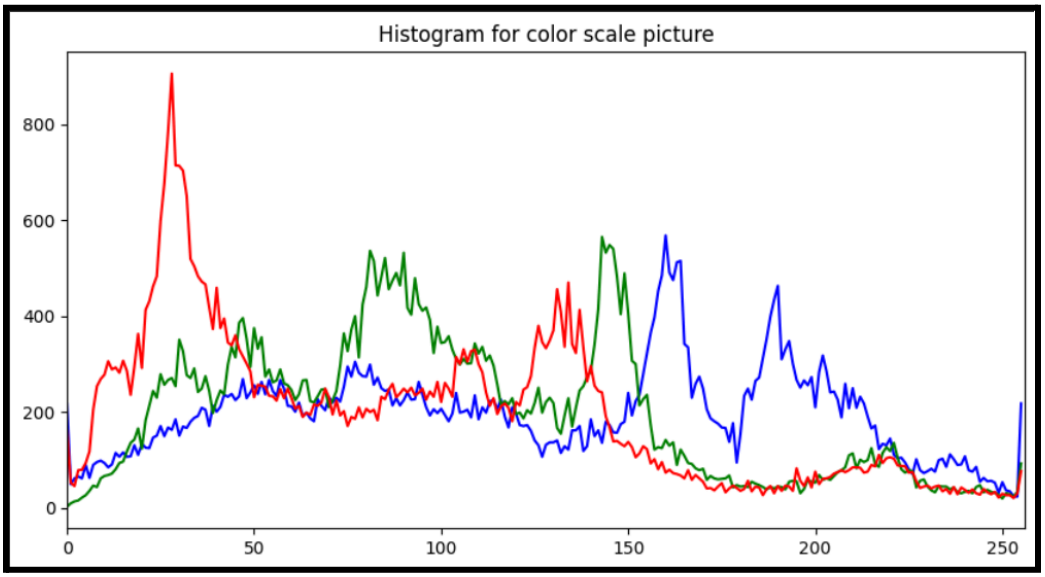
The project is implemented in Python, utilizing several libraries, including OpenCV for image processing and feature detection, NumPy for numerical operations, and Matplotlib for visualization.

Dataset Preparation

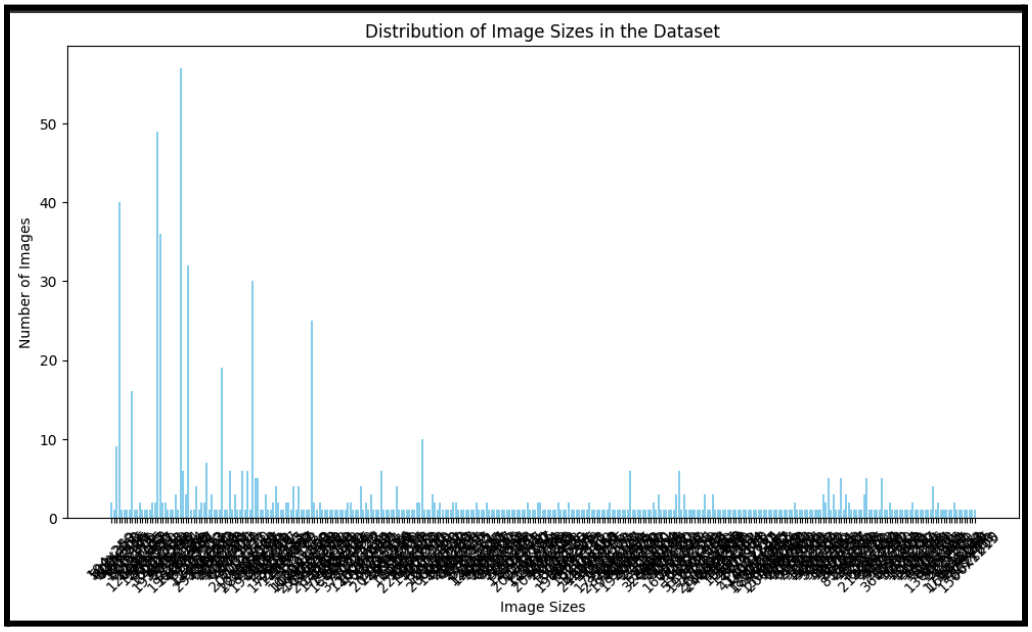
The dataset used in this project consists of various image categories which are structured into training and testing datasets. We are given a dataset which contains a variable number of instances per class (There are 7 classes: City, Face, Greenery, Building, House Indoor, Office, Sea). The dataset is also divided into two as training and test. We are expected to train our classifier using the training image set and test it using the test image set.

The **display_sample_images** function takes the directory of images as the input and initially loads them up with the defined class labels.

Then we have defined a function **plot_image_histograms** function which is plotting the histogram for color scale pictures.



While the **plot_image_sizes** shows the distribution of image sizes in the dataset



Implementation

The implementation phase is divided into several key functions, each contributing to the overall workflow of the visual classification system:

1. Feature Extraction using SIFT: The SIFT algorithm is employed to detect and describe local features in images. These features are invariant to image scaling, translation, and rotation, and provide a robust basis for matching different instances of an object.

We have described two main functions, **trainModel and testModel**, which are part of an image classification system using the SIFT method and Bag of Visual Words model. Below, each function is detailed under specific headings:

trainModel Function

This function is responsible for training the image classifier:

- **Parameters:** The function accepts path to the dataset, no_clusters for the number of visual words, and kernel for the SVM classification.
- **Variable Initialization:** Arrays for storing image descriptors and labels are initiated.

- **SIFT Descriptor:** A SIFT object is created to detect and compute the descriptors from the images.
- **Class Index Assignment:** Each image is assigned a class index based on its file name, allowing the model to recognize which category each image belongs to.
- **Descriptor Aggregation**: All descriptors are vertically stacked into a single numpy array.
- **KMeans Clustering:** The KMeans algorithm is applied to these descriptors to form no_clusters clusters. Each cluster centroid represents a visual word.
- **Histogram of Visual Words:** For each image, the presence of visual words is quantified into a histogram, representing the image in a high-dimensional feature space.
- **Feature Scaling:** StandardScaler is used to normalize the feature vectors, improving the effectiveness of the classifier.
- **SVM Training:** A Support Vector Machine is trained on the normalized histograms. The kernel type is specified by the kernel parameter.
- **Histogram Plotting:** The function optionally plots a histogram of the visual words.

testModel Function

This function handles testing the classifier with new images.

- **Reading Images:** Images for testing are loaded from the specified path. Similar to training, descriptors are collected, and true labels are identified based on file names.
- **SIFT Feature Extraction:** As with training, SIFT descriptors are computed for each test image.
- **Visual Word Quantification**: Using the KMeans model from training, each test image descriptor is transformed into a visual word histogram.
- **Normalization:** Test features are scaled using the same scaler from training.
- **Kernel Handling:** If a non-linear kernel (precomputed) is used, the test features are transformed accordingly.
- **Prediction:** The SVM predicts the category of each test image based on the feature vectors.
- **Confusion Matrix:** A confusion matrix is plotted to visualize the classifier's performance.
- **Accuracy Calculation:** The overall accuracy of the model is computed and displayed.

2. Building the Visual Vocabulary: The extracted features from the training set are clustered using the KMeans algorithm to create a 'visual vocabulary'. Each cluster center represents a 'visual word', and each image is thus described as a frequency histogram of these words.

- **extractFeatures:** This function builds histograms of visual words for each image by mapping SIFT descriptors to clusters generated by a KMeans model. Each histogram effectively represents the frequency of each visual word within an image, which serves as the feature set for classification.

- **normalizeFeatures:** Applies scaling to the feature vectors to standardize their range. This normalization helps in handling disparities in scale among features, ensuring that no particular feature dominates due to its magnitude.

3. Histogram Representation: For each training image, a histogram is computed by quantizing its SIFT descriptors into the corresponding visual words from the vocabulary. This histogram represents the distribution of visual words in the image.

4.Classifier Training: A Support Vector Machine (SVM) classifier is trained using the histograms of visual words. SVM is chosen for its effectiveness in high-dimensional spaces, ideal for handling the feature-rich data provided by the BoVW model.

- **svcParamSelection:** Performs a grid search to find the best parameters (C and gamma) for the SVM classifier. It uses cross-validation to ensure the parameters selected perform well across different subsets of the dataset.
- **findSVM:** Trains the SVM classifier using either directly provided features or a precomputed kernel matrix if required. This function incorporates class weights to handle imbalances in the dataset and uses parameters optimized through svcParamSelection.

5. Utility Functions: Various functions are written to plot and analyze image histograms, class distributions, color distributions, and sharpness across classes, which aid in visual data analysis and help in understanding the dataset more comprehensively.

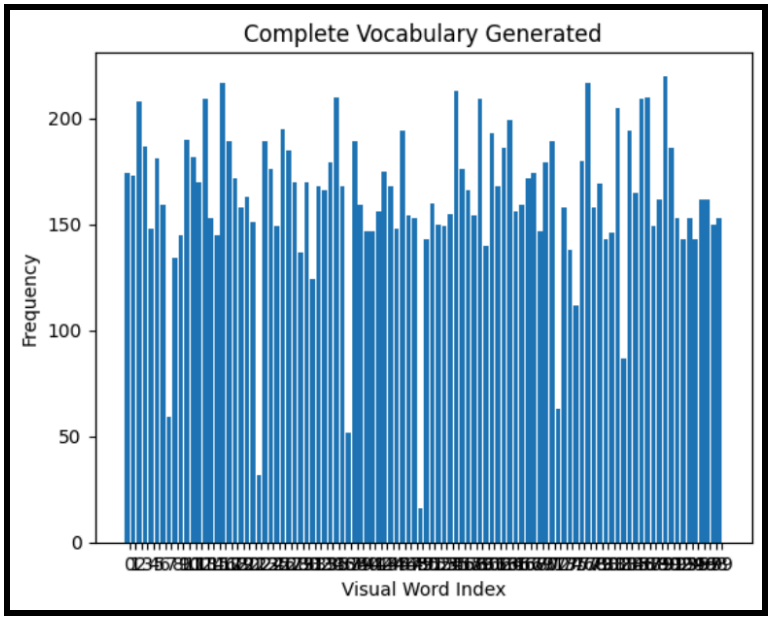
6. Testing and Classification: For each test image, its SIFT descriptors are extracted and quantized into visual words using the learned visual vocabulary. The resulting histogram is then fed into the trained classifier, which predicts the class of the test image.

- **plotConfusionMatrix:** Displays a confusion matrix to visualize the accuracy of the classifier across different classes. It offers the option to normalize the results to see the proportion of correct predictions and can be adjusted for visual clarity and understanding.

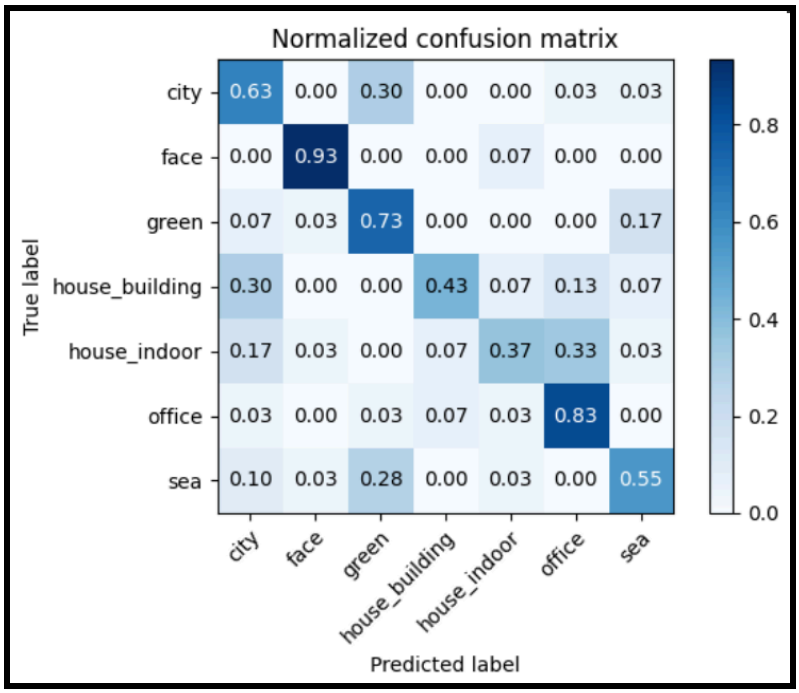
Results Obtained for All Cases

The results are assessed based on the classifier's accuracy and the system's ability to generalize across unseen images in the test dataset. Performance metrics, confusion matrices, and accuracy scores are calculated to evaluate the effectiveness of the visual classification model. Additionally, visual plots provide intuitive insights into the characteristics of the dataset and the behavior of the model.

For no_of_clusters=100 and kernel_type='linear'

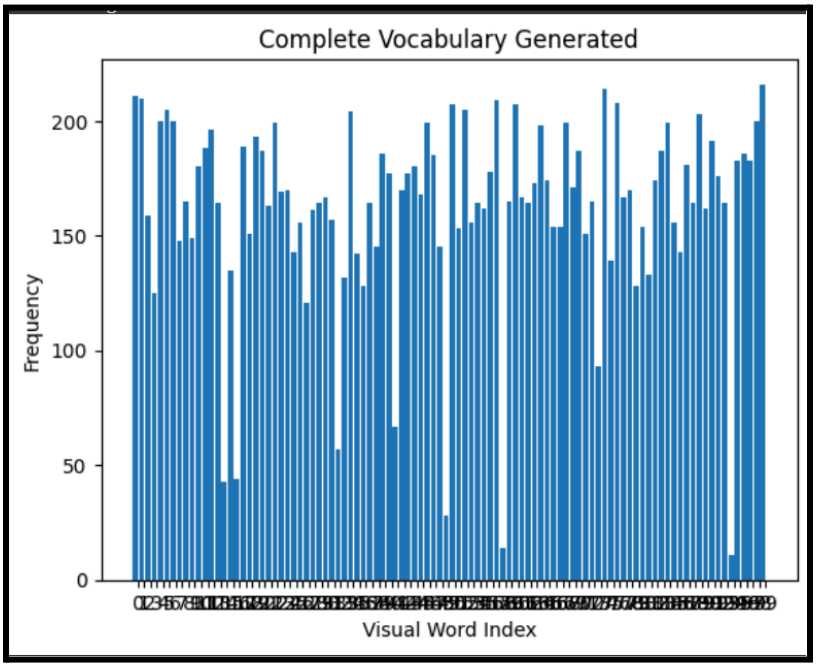


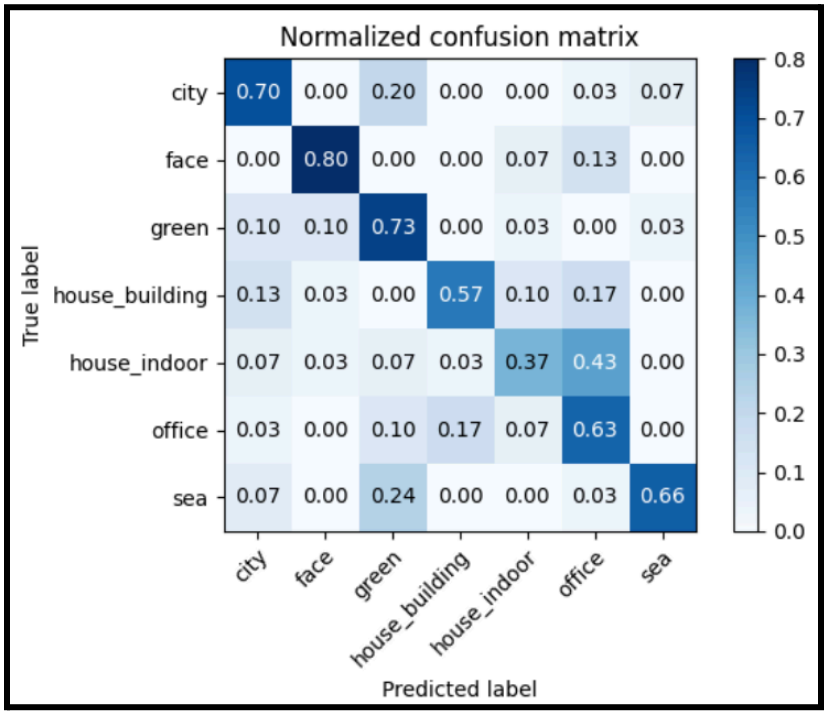
After this the SVM is fitted along with training and testing



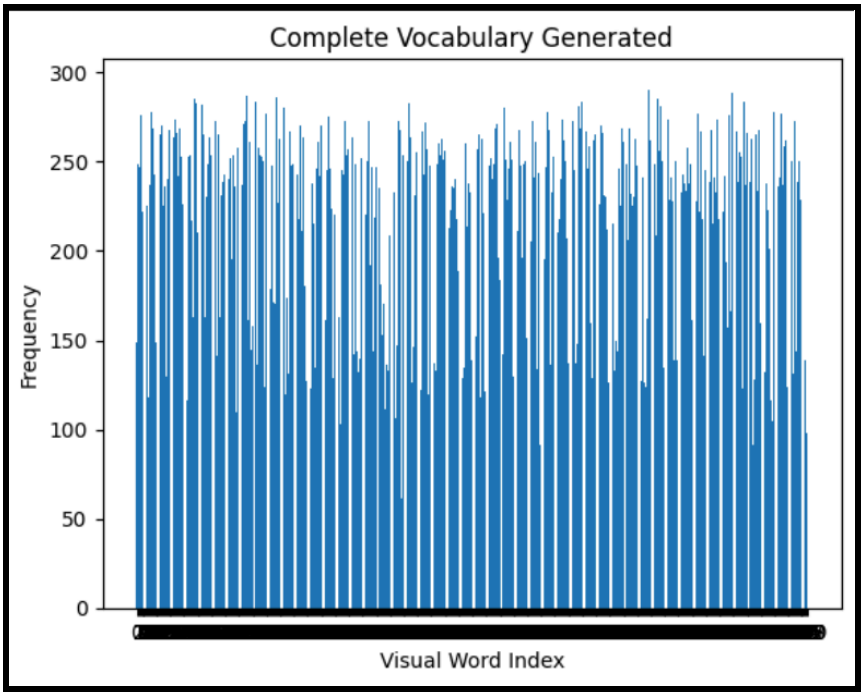
The accuracy score has been displayed and has shown impressive results.

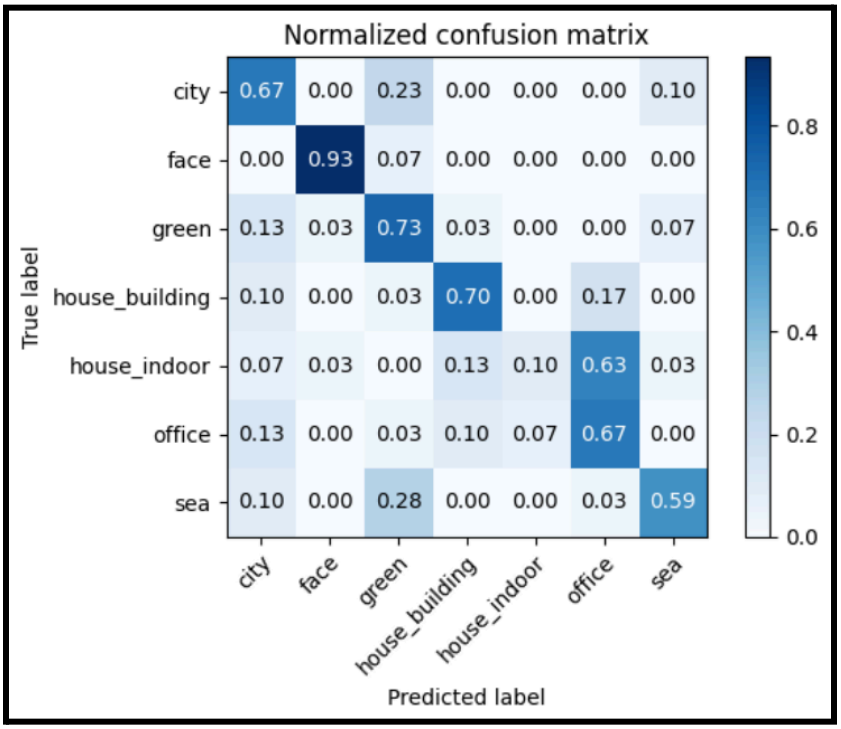
For no_of_clusters=100 and kernel_type=”precomputed”



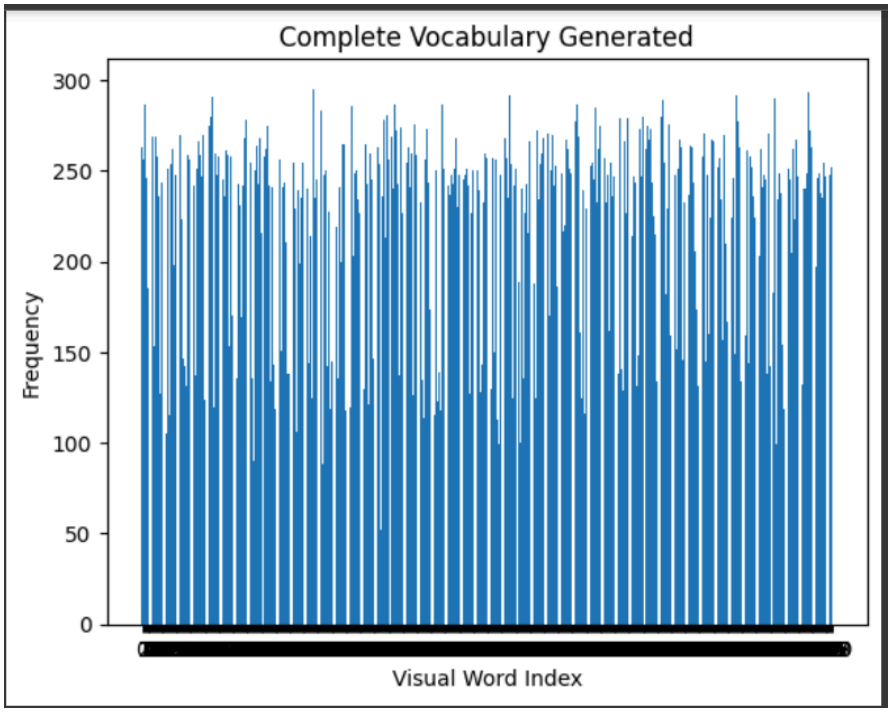


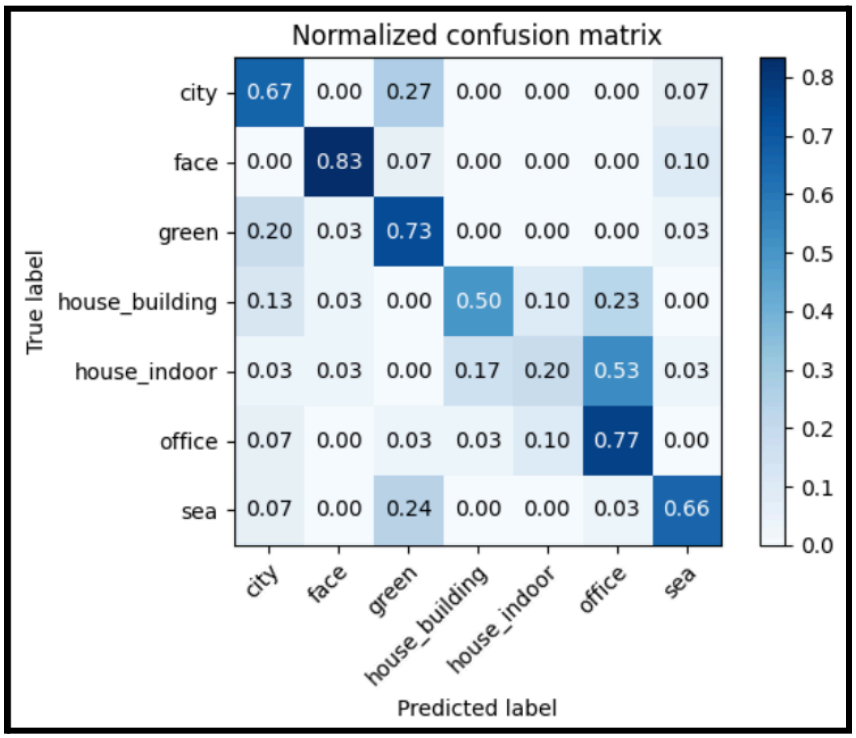
For no_of_clusters=500 and kernel_type=”linear”





For no_of_clusters=500 and kernel_type="precomputed"


Complete Vocabulary Generated
Frequency
Visual Word Index
0
50
100
150
200
250
300


Normalized confusion matrix
True label
Predicted label
city
face
green
house_building
house_indoor
office
sea
0.67 0.00 0.27 0.00 0.00 0.00 0.07
0.00 0.83 0.07 0.00 0.00 0.00 0.10
0.20 0.03 0.73 0.00 0.00 0.00 0.03
0.13 0.03 0.00 0.50 0.10 0.23 0.00
0.03 0.03 0.00 0.17 0.20 0.53 0.03
0.07 0.00 0.03 0.03 0.10 0.77 0.00
0.07 0.00 0.24 0.00 0.00 0.03 0.66
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

Conclusion

This task successfully showcases the integration of the SIFT method with the Bag of Visual Words model for effective visual classification. By leveraging a strategically selected number of clusters in the KMeans algorithm, the system efficiently organizes SIFT descriptors into a meaningful visual vocabulary, which is critical for feature representation. The choice of the SVM kernel type further enhances classification performance by optimally fitting the high-dimensional feature space created by the visual words. These elements together enable the system to robustly recognize and classify images, demonstrating promising results across various categories. Future enhancements may include exploring additional feature extraction techniques and expanding the dataset to address more diverse and complex visual scenarios, potentially improving both the versatility and accuracy of the classification system.

SIFT based Live WebCam Matching

This part aims to demonstrate real-time feature matching between live video frames captured from a webcam and a reference image using the SIFT algorithm.

Methodology: The project is implemented in Python, utilizing the OpenCV library for image processing, feature detection, and video capture. The code follows the following steps:

1. **Initialization**: The SIFT feature detector and descriptor extractor are initialized using OpenCV's `cv2.SIFT_create()` function. Additionally, a `BFMatcher` object is created to perform brute-force matching between feature descriptors.
2. **Video Capture**: The webcam is accessed using OpenCV's `cv2.VideoCapture()` function, and a continuous loop is established to read frames from the webcam.
3. **Image Pre-processing**: For each frame captured from the webcam, the image is converted to grayscale using `cv2.cvtColor()` function.
4. **Feature Detection and Description**: SIFT keypoints and descriptors are computed for both the live frame and the reference image using the `sift.detectAndCompute()` function.
5. **Feature Matching**: The SIFT descriptors from the live frame and the reference image are matched using the `bf.match()` function from the `BFMatcher` object. The matches are sorted based on their distances (to prioritize the best matches).
6. **Frame Rate Calculation**: The time taken to process each frame is calculated to estimate the frame rate (FPS) of the application.
7. **Visualization**: The matched features between the live frame and the reference image are visualized using OpenCV's `cv2.drawMatches()` function. The resulting image is displayed in a window titled "SIFT," along with the calculated FPS value overlaid on the image.
8. **Loop Control**: The loop continues until the user presses the "Esc" key (ASCII code 27), at which point the program exits and releases the webcam resource.

Results: The code successfully captures live video frames from the webcam, computes SIFT features, matches them with the features of a reference image, and displays the matched

features in real-time. The calculated frame rate (FPS) is overlaid on the displayed image, providing feedback on the application's performance.

Conclusion: This project demonstrates the implementation of real-time feature matching using the SIFT algorithm and a webcam. By leveraging OpenCV's capabilities for video capture and image processing, the code can detect and match features between live video frames and a reference image, enabling potential applications in object tracking, augmented reality, and computer vision-based user interfaces.

3D Reconstruction and Localization

We took reference from the following repository for performing hierarchical localization.
The pipeline for tasks related to structure-from-motion (SfM) and localization using images of the Sacré-Cœur Basilica and kaggle image matching challenge dataset was created. This involves generating and using 3D models from sets of 2D images, and potentially aligning or localizing query images against a 3D model.

We set our feature extractor configuration using DISK and matcher configs using disk+lightglue

Algorithmic Details of 'DISK' and 'DISK+LightGlue'

- **DISK:** DISK (Differentiable Scale-Invariant Keypoint Detector) is a state-of-the-art method for detecting and describing local features in images. It is designed to be robust against changes in scale and rotation, enhancing its utility in diverse imaging conditions. DISK uses a differentiable approach, allowing for end-to-end learning which can be fine-tuned for specific tasks or environments.
- **DISK+LightGlue:** This configuration combines DISK feature detection with a light version of the "SuperGlue" matching algorithm (referred to as LightGlue). SuperGlue is known for its powerful performance in matching features between images using graph

neural networks. The combination ensures highly accurate feature matching even in challenging scenarios, crucial for precise localization and mapping tasks.

Then we extract features and match them across image pairs. Since we deal with few images, we simply match all pairs exhaustively. For larger scenes, we would use image retrieval. So further in the pipeline following steps were performed

1. Extracting Features

- **Function:** extract_features.main
- **Purpose:** This function is used to extract distinctive features from a set of images. Features are specific patterns or unique characteristics in an image, such as edges, corners, or blobs, which are used to recognize and match different images of the same scene or object.
- **Parameters:**
    - **feature_conf:** Configuration settings for the feature extraction algorithm (DISK and SIFT both were used). These settings dictate how the features are detected and described. SIFT was faster, but DISK gave a better 3d reconstruction.
    - **images:** The base directory where the images are stored.
    - **image_list=references:** A list of specific images within the images directory from which features will be extracted. This list helps in targeting specific files without needing to process every file in the directory.
    - **feature_path=features:** The path where the extracted features will be saved, typically in an H5 file, which is a binary file format that stores large quantities of numerical data efficiently.

2. Generating Pairs for Structure-from-Motion

- **Function:** pairs_from_exhaustive.main
- **Purpose:** This function generates pairs of images that will later be used to establish correspondences between them in the feature matching process. These pairs are crucial for tasks like Structure-from-Motion (SfM), where multiple observations of the same point (from different images) are used to compute its 3D position.
- **Parameters:**
    - **sfm_pairs:** The path where the list of generated image pairs will be saved. This file typically contains pairs of image indices or names that should be considered for matching.
    - **image_list=references:** Again, this specifies which images from the dataset should be considered for pairing.

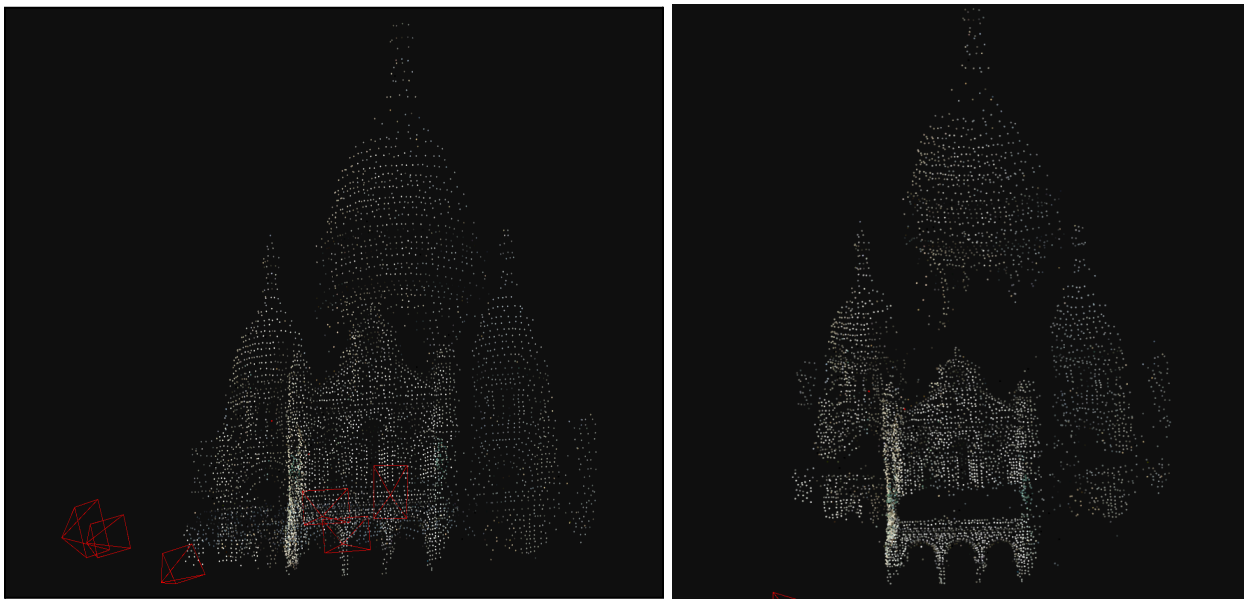
3. Matching Features Between Image Pairs

- **Function:** match_features.main

- **Purpose:** This function takes the extracted features and the defined pairs of images, then performs feature matching between each pair. Feature matching involves finding which features in one image correspond to features in another image, a key step in many computer vision tasks like image alignment, 3D reconstruction, and object recognition.
- **Parameters:**
    - **matcher_conf:** Configuration settings for the matching algorithm (DISK+LightGlue, as previously mentioned). This configuration enhances the accuracy and efficiency of matching across diverse and challenging datasets.
    - **sfm_pairs:** The file containing the image pairs to be matched.
    - **features=features:** The path to the previously extracted features. This file is used to retrieve the features for each image when performing matches.
    - **matches=matches:** The path where the results of the matches will be stored, typically in an H5 file format, allowing efficient handling of large match datasets.

Following this we performed 3D reconstruction using the matched features and the image pairs. The process typically involves creating a 3D point cloud or mesh from multiple images by triangulating the matched points and using camera calibration data. The reconstructed 3D model is visualized using Plotly, a popular library for creating interactive plots. The model is plotted with a semi-transparent red color, and points are displayed with their RGB values if available.

We also save the 3d point cloud in the form of html and json file to be displayed as per need.



We also visualized the key points triangulated

Localization

Image localization refers to the process of determining where an image was taken relative to a known environment, often expressed in terms of the camera's position and orientation at the time of capture. In simpler terms, it involves finding out the viewpoint from which a photo was taken within a pre-mapped space.

The Role of a 3D Map in Localization

When a 3D map of an environment or object is available, localizing any image of that environment becomes feasible under the right conditions. Here's why:

1. **Reference Geometry:** A 3D map provides detailed geometric information about the scene. It includes not just the positions of various points in space but often their textural or color information as well. This rich dataset acts as a reference system against which new observations (images) can be compared and analyzed.
2. **Feature Correspondence:** By detecting features (distinctive points or patterns) in the new image and matching them with features in the 3D map, algorithms can estimate the relative position from which the new image views the scene. This is typically done through a process of feature extraction followed by feature matching.

3. **Estimating Camera Pose:** Once matches are found between the 3D map's features and those in the new image, computer vision techniques (like triangulation and pose estimation algorithms) are used to calculate the camera's position and orientation. This is usually achieved using methods that optimize the fit of 3D points to 2D points while accounting for camera optics and geometry.
4. **Robust Estimation Techniques:** Techniques such as RANSAC (Random Sample Consensus) are used to ensure that the pose estimation is resilient to outliers or incorrect matches. This helps in reliably localizing the image even if some feature matches are incorrect.

Query localizer Takes keypoints from the query image and their corresponding 3D points (from the map), and uses COLMAP's absolute pose estimation to compute the camera pose.

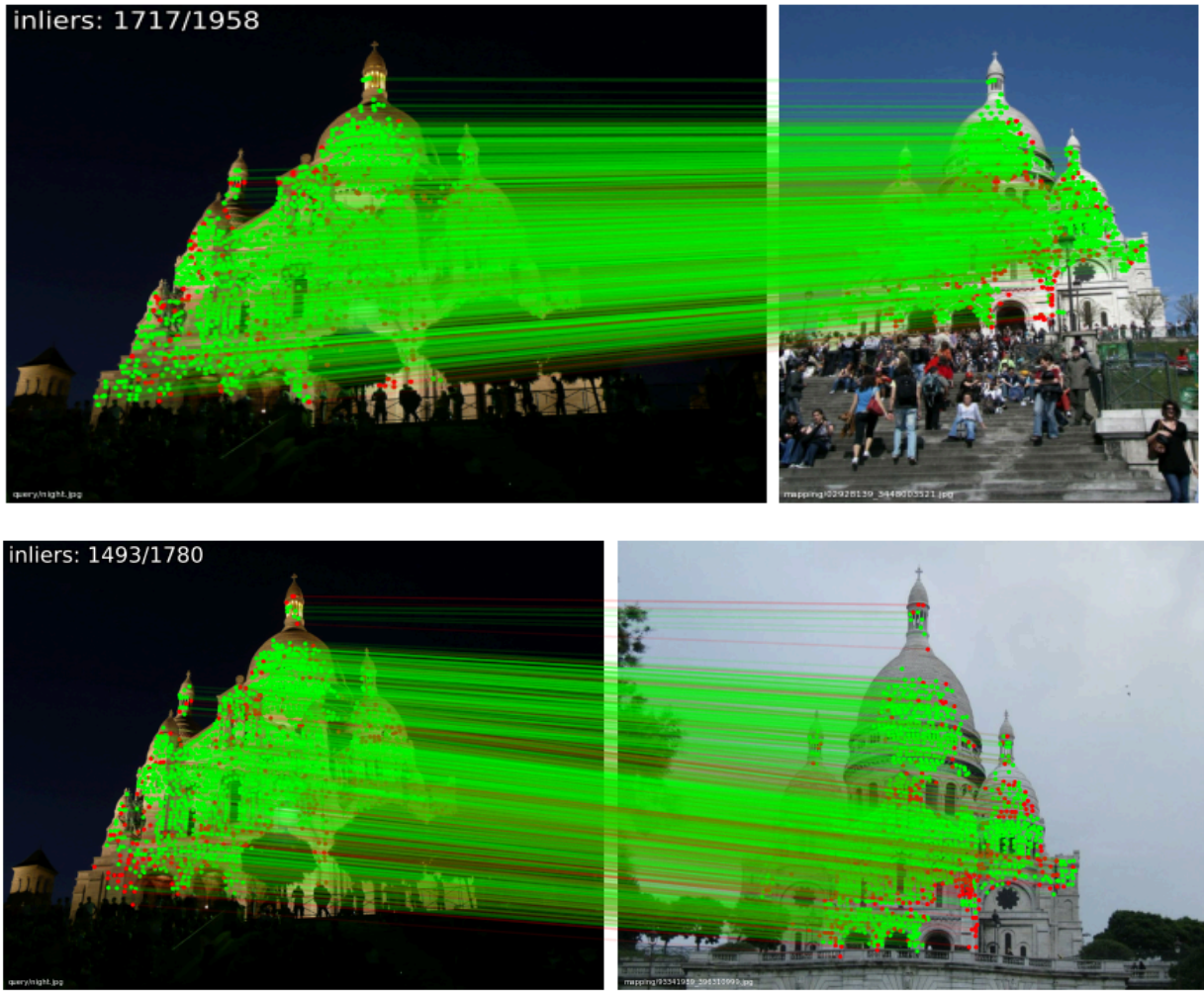
We took the following images for seeing the working of localization

Steps for Image localization:

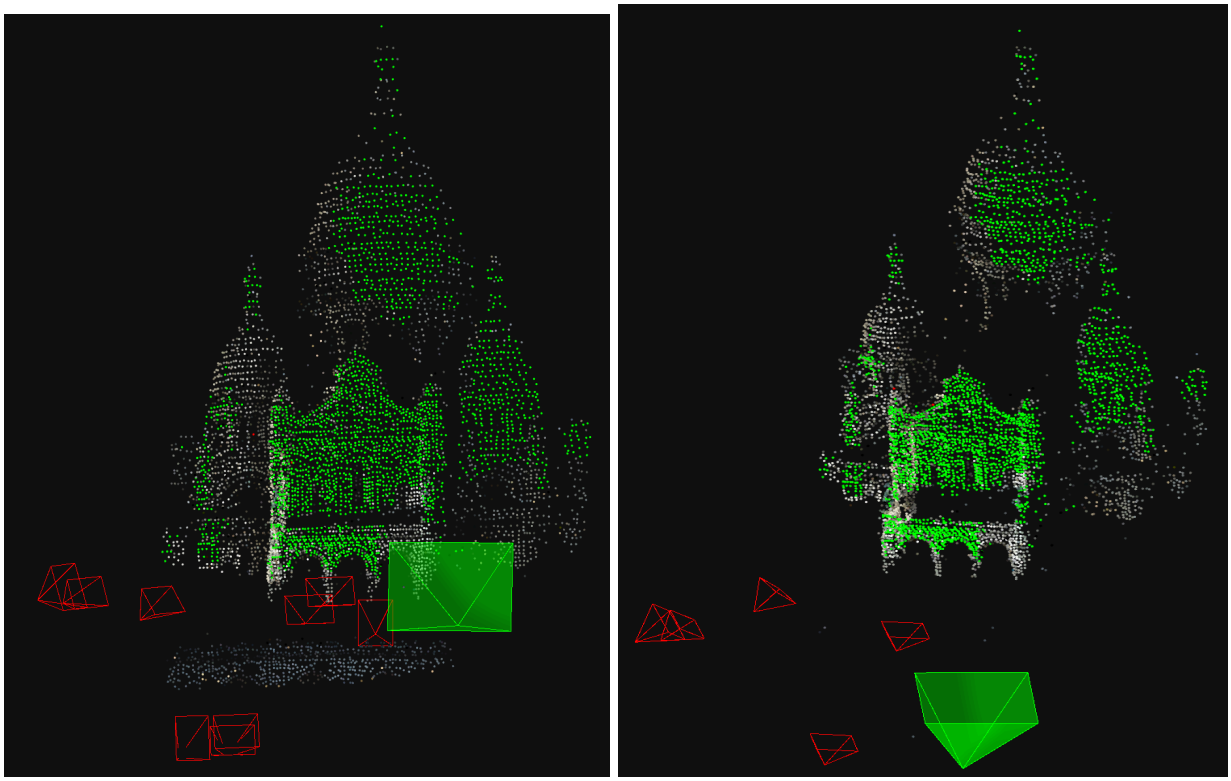
1. **Query image acquisition :** The query image represents a new view of the monument, possibly taken under different conditions (e.g., lighting, angle) from those in the mapping images used to build the 3D model.
2. **Feature extraction from query image :** Feature extraction is critical as it finds points of interest within the image that can be matched with features in the existing 3D map, facilitating the localization process.
3. **Image pairing for localization :** The system creates pairs between the query image and images that were registered in the 3D model during its construction Pairing the query image with reference images helps in narrowing down potential matches by focusing on likely candidates for feature correspondence.

4. **Feature Matching :** This is a crucial step where the system identifies similar features between the new image and those in the 3D model, establishing a connection between 2D features and 3D points.
5. **Localization estimation :** Localization involves computing where and how the camera was oriented when the query image was captured, relative to the 3D model. This is done using robust estimation techniques such as RANSAC to handle outliers in feature matches. Using the matched 2D and 3D points, the pose of the query image is estimated through the COLMAP absolute pose estimation function, which calculates where and how the camera was oriented when the photo was taken.
6. **Visualization of localization :** Visualizing the localization results provides an intuitive understanding of how well the query image aligns with the 3D model. Green points and camera icons indicate successful localization, helping to visually confirm the accuracy of the process.

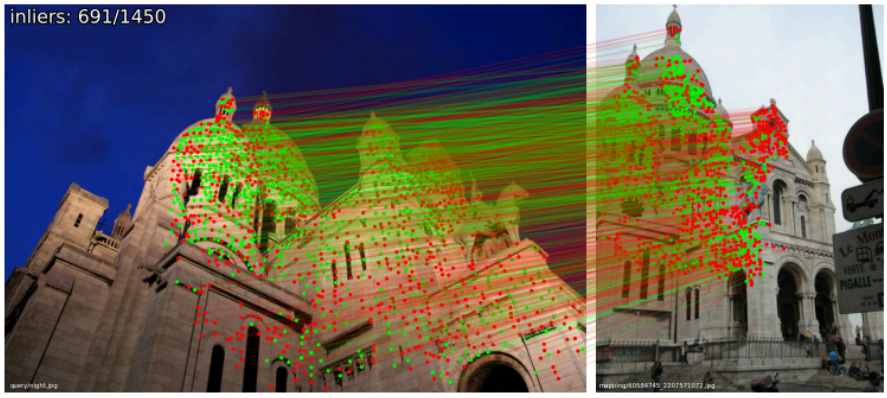
Drawing correspondences between new image and existing image

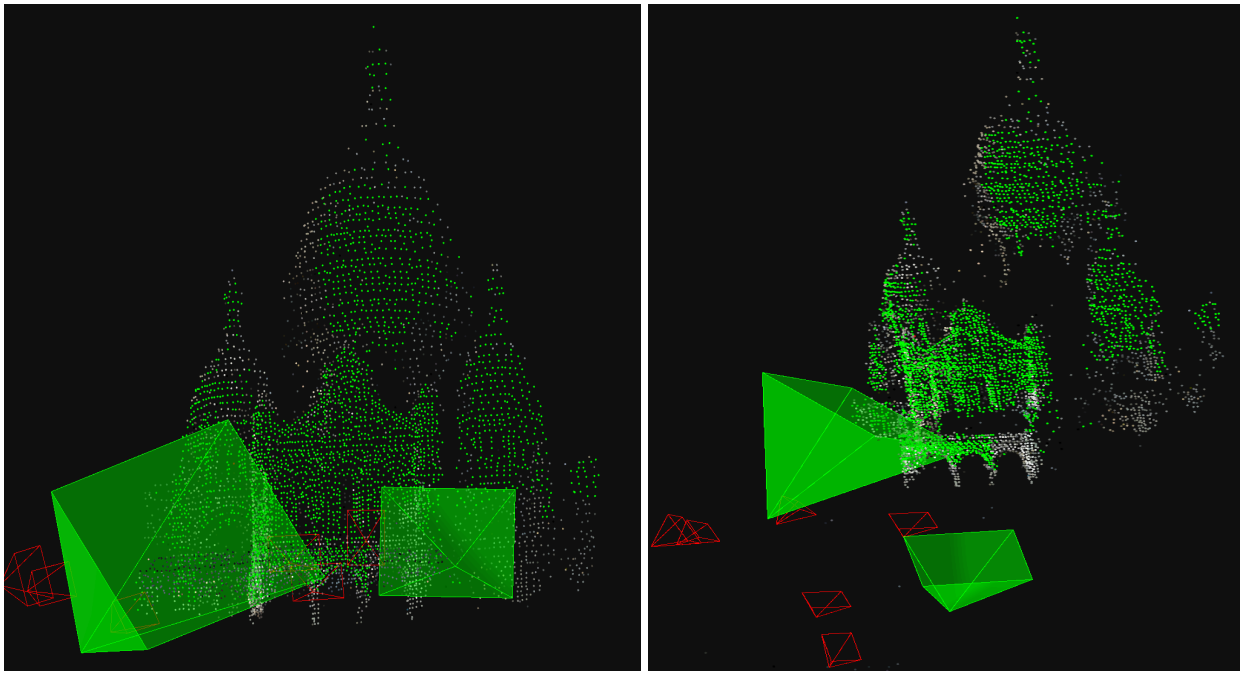


For Image 1



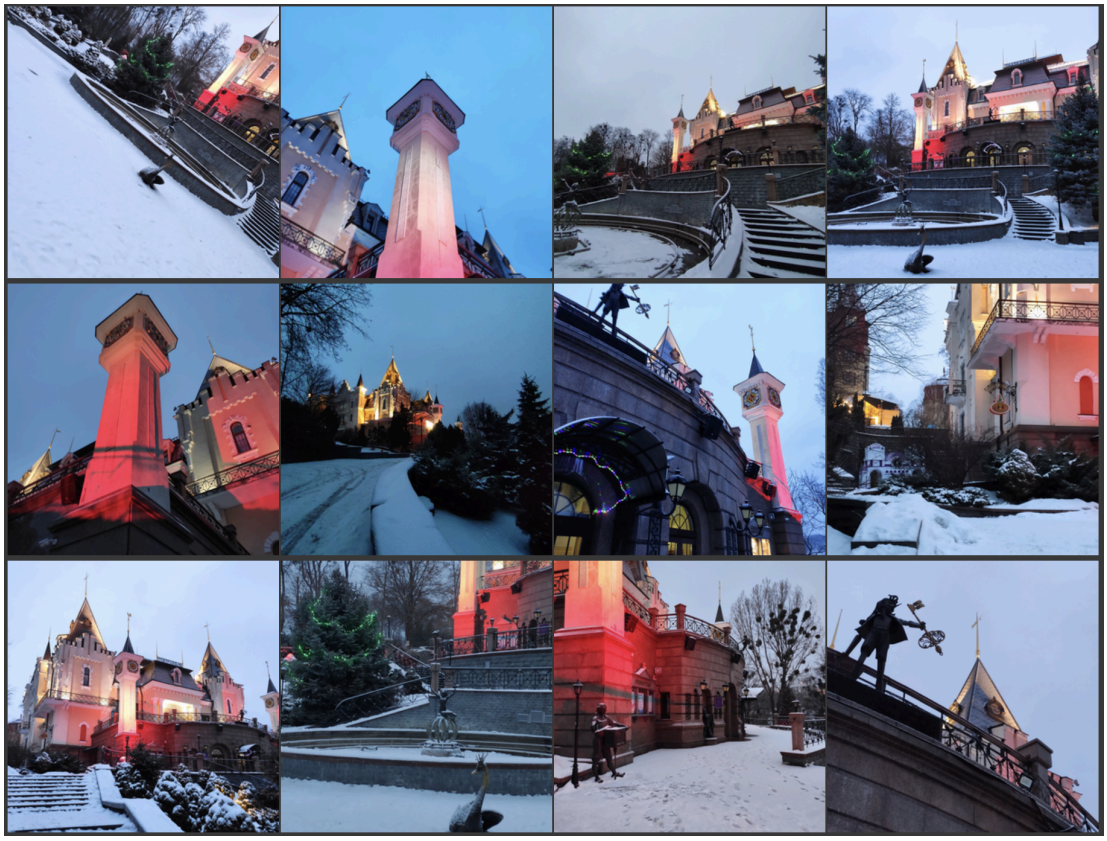
For Image 2

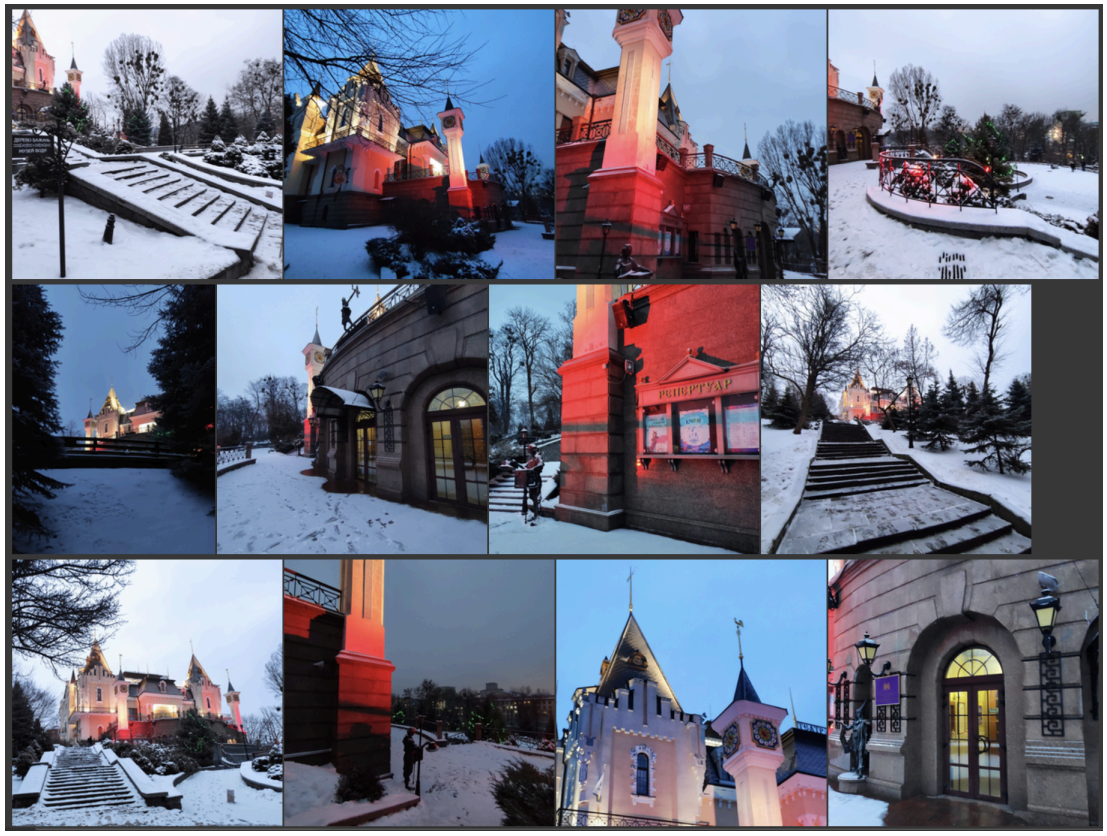




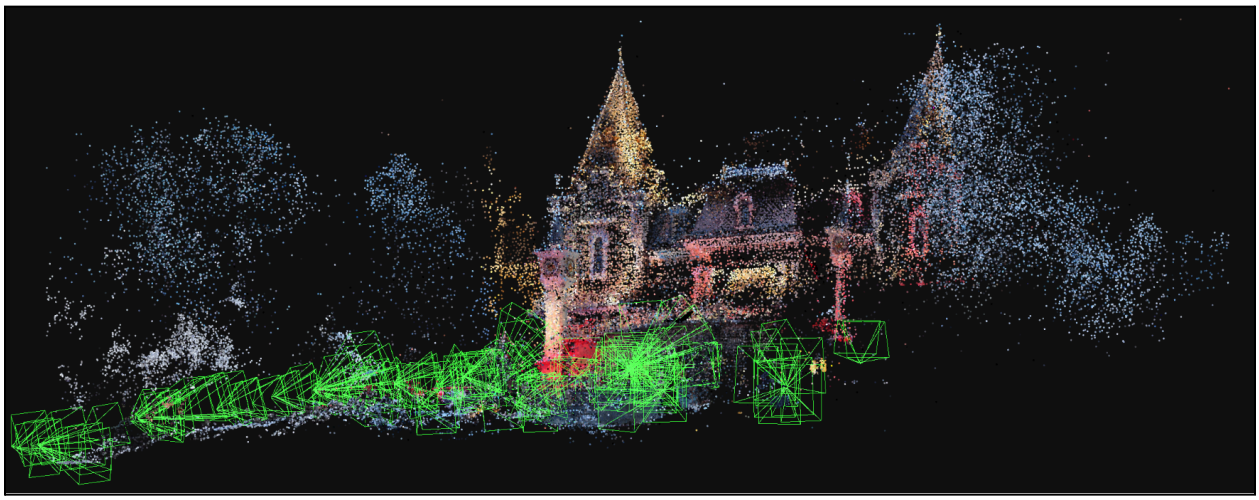
Kaggle Image matching dataset

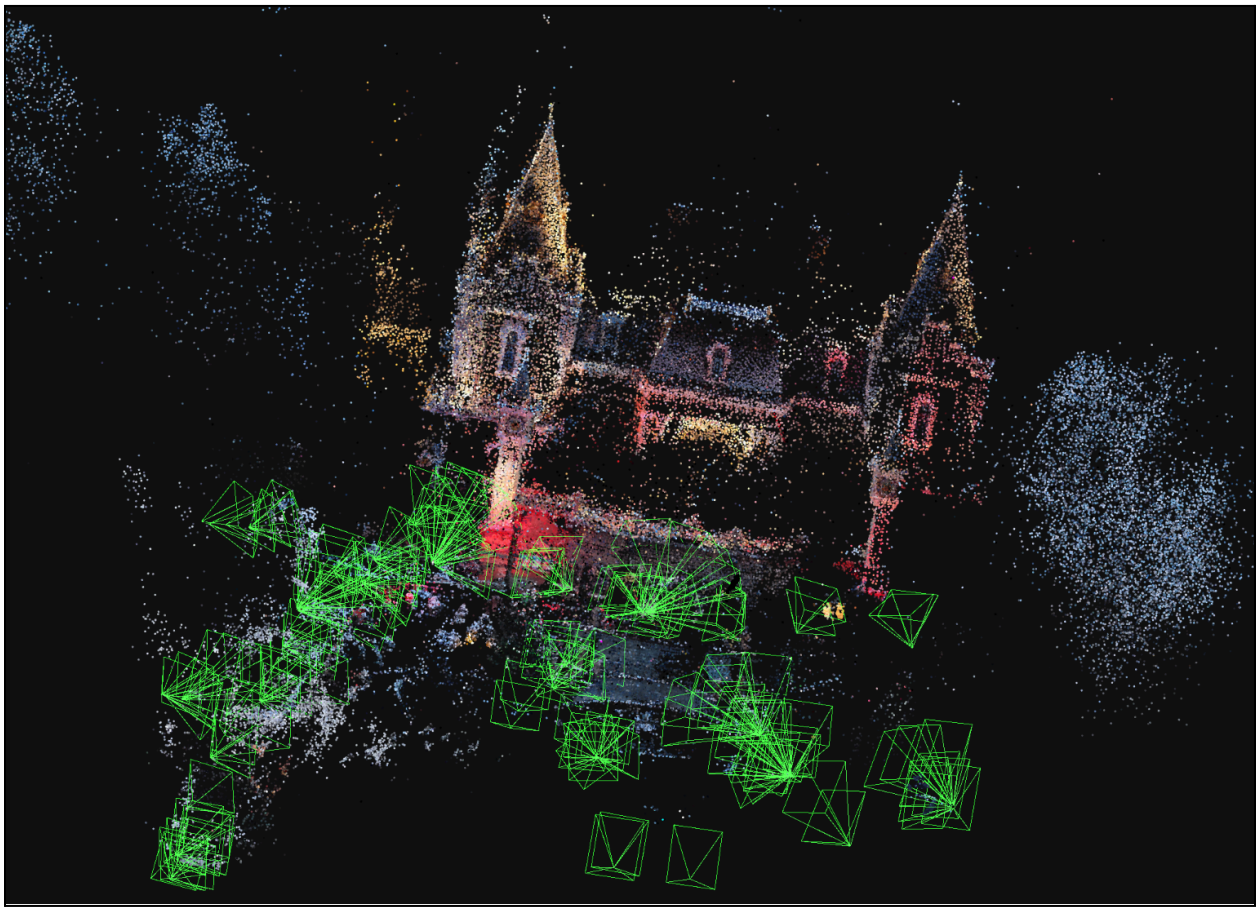
Images that we were taking were :

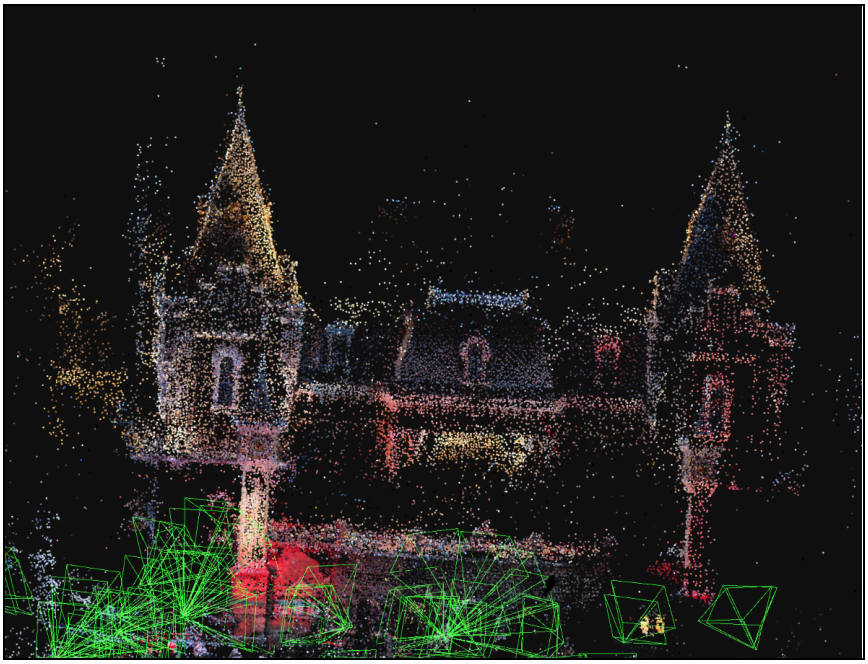



РЕПЕРТУАР

3d reconstruction that was generated







SIFT for Adversarial Image detection

Deep learning models are susceptible to adversarial attacks, where imperceptible perturbations in the input data can cause the model to make incorrect predictions. This project proposes a lightweight and real-time adversarial attack detector, ensuring the security and reliability of these models in practical applications.

Methodology: The adversarial attack detector leverages the Scale-Invariant Feature Transform (SIFT) algorithm to extract distinctive features from the input image. SIFT is a well-known algorithm in computer vision that identifies and describes local features in an image, making it suitable for tasks like object recognition and image matching.

The detector follows these steps:

1. **Feature Extraction:**
    - Apply the SIFT algorithm to the input image to detect keypoints and compute their descriptors.
    - Extract the number of SIFT keypoints detected in the image.
2. **Feature Engineering:**
    - Calculate the mean and standard deviation of the input image's pixel values.
    - Combine the number of SIFT keypoints, the mean, and the standard deviation as a feature vector representing the input image.
3. **Classification:**
    - Train a logistic regression model using feature vectors extracted from a dataset containing both benign and adversarial examples.

- For a given input image, compute its feature vector and feed it into the trained logistic regression model.
- The model outputs a binary classification, indicating whether the input is adversarial or benign.

Implementation: The `adv.py` file contains the implementation of the adversarial attack detector. The `sift_keypoints` function performs the feature extraction step, computing the number of SIFT keypoints, the mean, and the standard deviation of the input image tensor.

The `adv_predict` function loads a pre-trained logistic regression model and uses it to classify the input image as adversarial or benign based on the extracted features. It prints the prediction result and the probability of the positive class (adversarial).

Evaluation and Results: The research paper mentions that the proposed adversarial attack detector was evaluated on various datasets, including CIFAR-10, CIFAR-100, and ImageNet. However, specific evaluation metrics and results for the detector component are not provided in the attached materials.

For CIFAR 10 dataset with variable adversarial image composition in test set we obtain the following results

Attack	Detector Accuracy	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
FGSM	96.14	77.97	74.26	70.56	66.86	63.15	59.45	55.75	52.04	48.344	44.64	40.936
CW	55.23	61.54	61.12	60.7	60.28	59.87	59.45	59.04	58.62	58.21	57.78	57.36
BIM	75	69.4342	67.44	65.44	63.44	61.45	59.45	57.45	55.46	53.46	51.47	49.47
Deep Fool	50.14	59.51	59.49	59.48	59.47	59.46	59.46	59.44	59.45	59.42	59.41	59.39
ALL	74.1225	69.08	67.15	65.23	63.3	61.38	59.45	57.52	55.6	53.67	51.75	49.82

For CIFAR 100 dataset with variable adversarial image composition in test set we obtained the following results

Attack	Detector Accuracy	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
FGSM	89.19	75.09	71.96	68.83	65.71	62.58	59.45	56.32	53.19	50.07	46.94	43.81
ALL	70.56	67.66	66.01	64.37	62.73	61.09	59.455	57.81	56.17	54.53	52.89	51.24

Further we were also able to prove attack generalization, which means that if the detector is trained on Attack A, then how successfully can it detect all the other attacks.

Trained on FGSM	CIFAR10	Cifar100
FGSM	96.4	59.19
Deep Fool	73	58
BIM	96.2	88.16
CW	64	78
Total	79.38	70.566

In addition to this being a classical CV approach our detector was able to provide dataset generalization as well.

Trained On (down) Tested On (side)	CIFAR 10	CIFAR 100
CIFAR 10	74.1225	69.32
CIFAR 100	70.566	73.63

Therefore, proposed adversarial attack detector offers a lightweight and real-time solution for identifying adversarial inputs targeting Vision Transformers. By leveraging SIFT features and a logistic regression model, the detector can accurately classify input images as adversarial or benign, ensuring the security and reliability of ViT models in practical applications.

SIFT for fingerprint matching

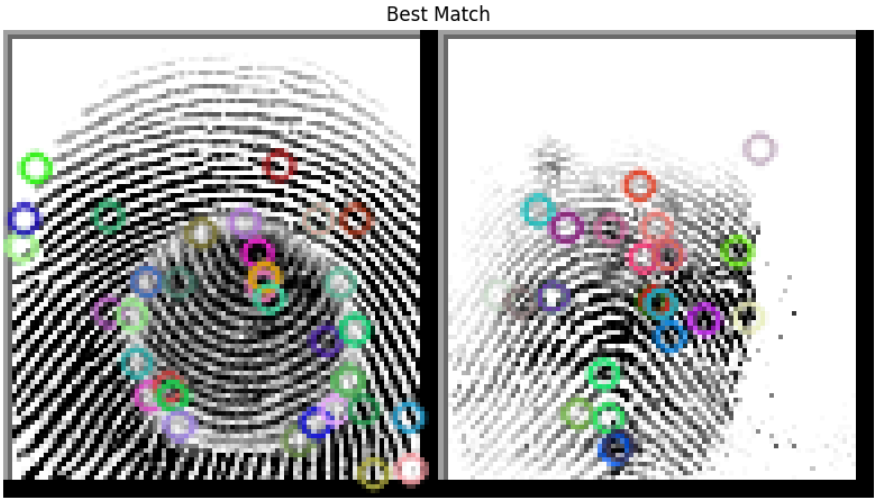
SIFT is a robust algorithm that can detect and describe local features in images, making it suitable for object recognition and image matching applications. In this project, the SIFT algorithm is employed to match a given fingerprint image against a database of fingerprint images to find the best match.

Methodology for fingerprint matching: The sub-project is implemented in Python, utilizing several libraries, including OpenCV for image processing and feature detection, NumPy for numerical operations, and Matplotlib for plotting and visualization.

The code follows the following steps:

1. Read the input fingerprint image (sample) and the directory containing the database of fingerprint images (read_files).
2. For each fingerprint image in the database: a. Compute the SIFT keypoints and descriptors for both the input image and the database image using OpenCV's SIFT implementation. b. Match the descriptors between the input image and the database image using the FlannBasedMatcher from OpenCV. c. Filter the matches based on a distance ratio test to eliminate ambiguous matches. d. Calculate the match score as the ratio of the number of matched keypoints to the total number of keypoints in the image with fewer key points. Keep track of the best match score and its corresponding database image.
3. Print the filename and score of the best match.
4. Draw the matched keypoints between the input image and the best match image using OpenCV's drawMatches function.

Results/Analysis: The code successfully finds the best match between the input fingerprint image and the database of fingerprint images. The output includes the filename of the best match and its corresponding match score. Additionally, a plot is generated and saved as best_match_plot.png, displaying the matched keypoints between the input image and the best match image.



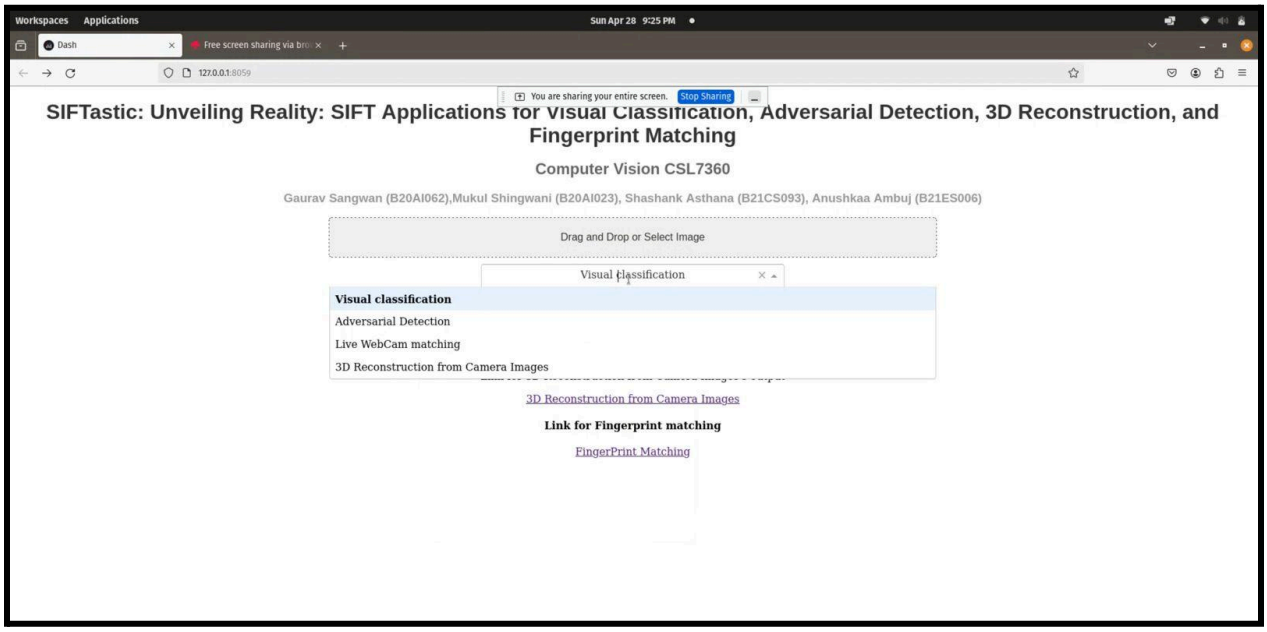
Conclusion

This demonstrates the implementation of the SIFT algorithm for fingerprint matching. By extracting and matching keypoints between the input fingerprint image and a database of fingerprint images, the code can effectively find the best match. The generated plot provides a visual representation of the matched keypoints, aiding in the analysis and verification of the results.

Deployment of DASH Application

We have deployed a DASH application for all the tasks that is for Visual classification, Adversarial detection, 3D reconstruction and Fingerprint matching.

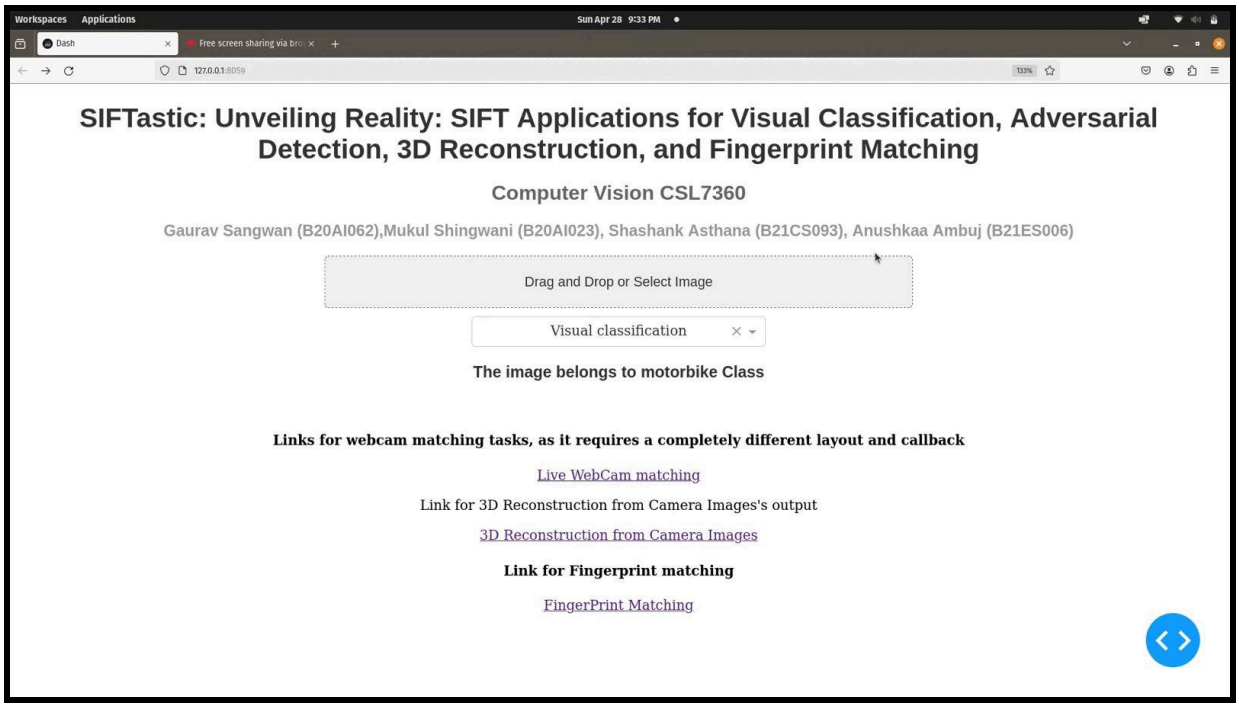
The portal screenshot has been displayed below



The visual classification module of the SIFTastic application showcases the ability of Scale-Invariant Feature Transform (SIFT) to accurately identify and categorize objects within images. **By dragging and dropping an image** into the application, the system utilizes keypoint detection and descriptor extraction to classify the image into predefined categories.

Practical use cases

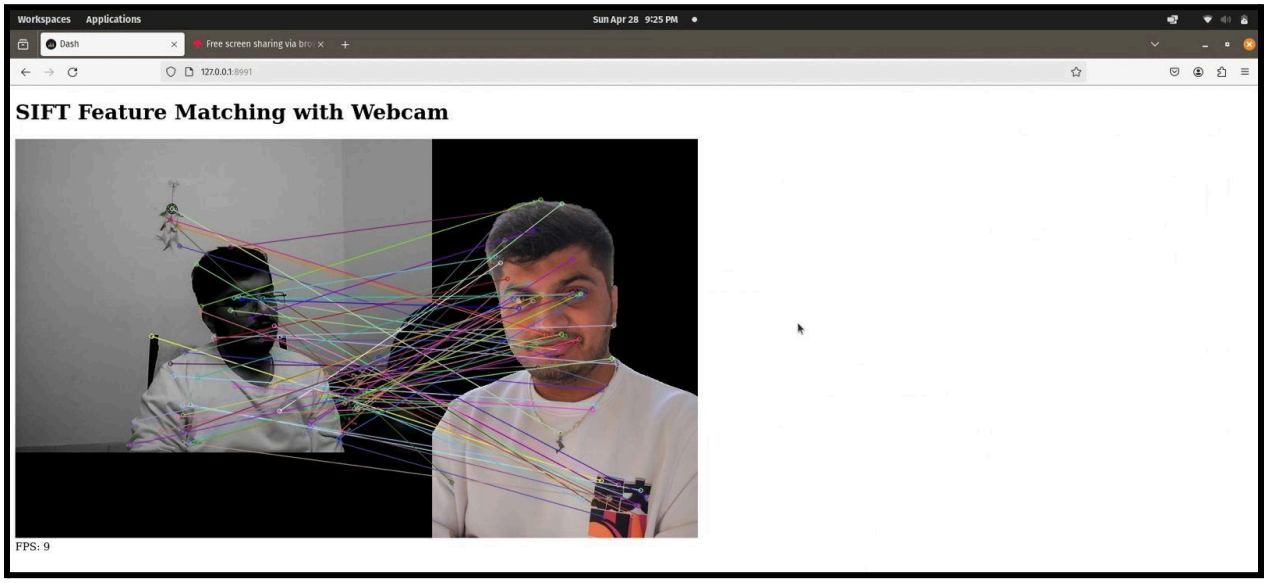
- Retail Image Sorting
- Photo Management Software
- Surveillance Systems



The SIFT feature matching module with webcam integration in the SIFTastic application highlights real-time object recognition and tracking capabilities. This feature uses a live video feed from the webcam and applies SIFT to detect and match features between the live image and a predefined object or image database.

Practical use cases:

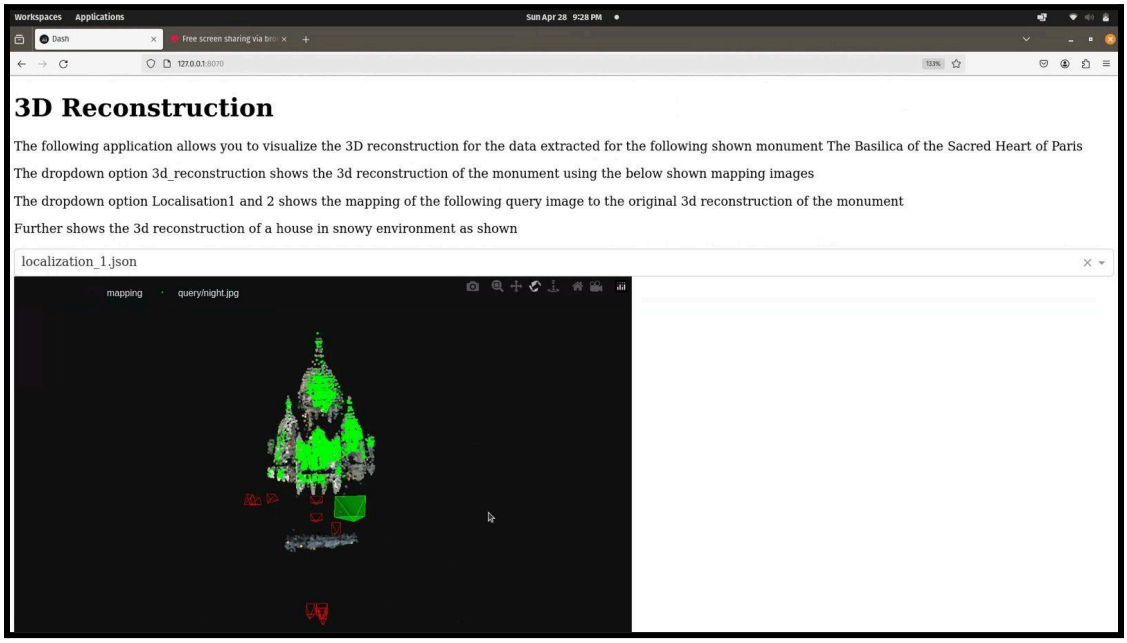
- Augmented Reality (AR)
- Interactive Marketing

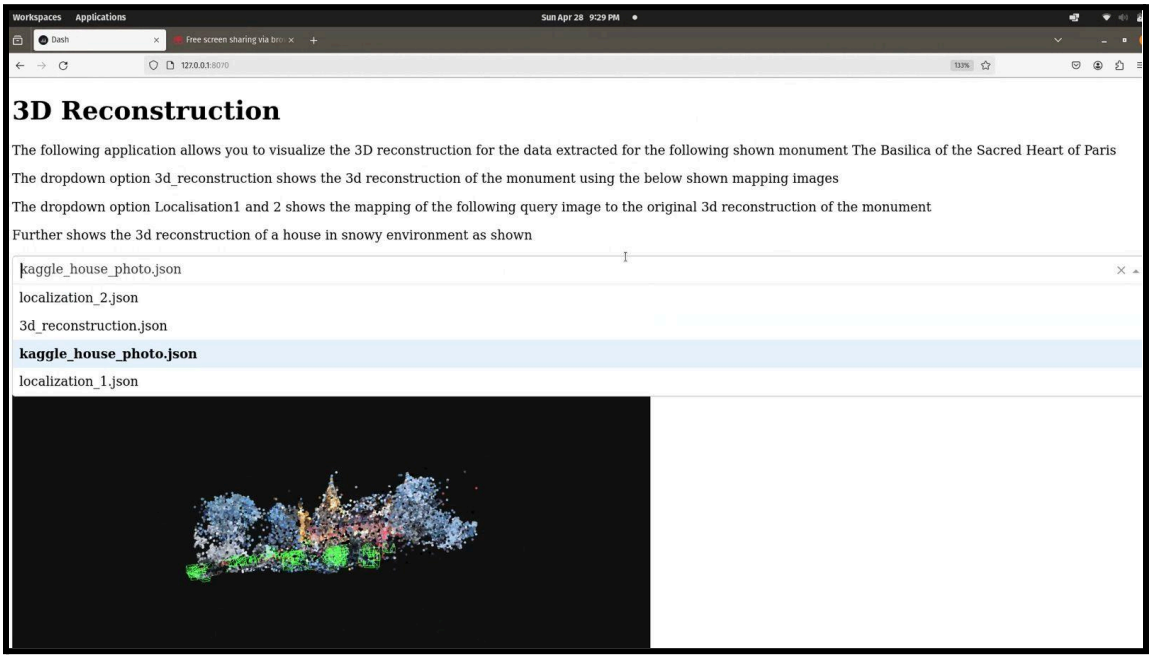


The 3D reconstruction functionality of SIFTastic demonstrates the application's prowess in generating three-dimensional models from two-dimensional images. It takes advantage of multiple images from different viewpoints to reconstruct the spatial structure of the scenes.

Practical use cases:

- Cultural Heritage Preservation
- Real Estate and Interior Design
- Gaming and VR





The fingerprint matching tool within SIFTastic utilizes SIFT's feature matching capabilities to compare and identify fingerprint patterns. This task highlights the application's accuracy and efficiency in biometric identification.

Practical use cases:

- Law Enforcement
- Access Control Systems
- Identity Verification