
**Course Name: CSL 7360 Computer Vision**

**Instructor: Dr. Pratik Mazumder**

# Course Project: To compare Traditional and Deep Learning methods in Image Stitching

**Submitted by:**

Priybrata Saha
Roll No: M24IRM004

Vishwanath R
Roll No: M24IRM010

Suddala Madhu
Roll No: M24IRM007

Projat Banerjee
Roll No: P23CS0005

Indian Institute of Technology Jodhpur

April 13, 2025

GitHub Link
Streamlit App Link

# Contents

# Chapter 1

# Introduction

## 1.1 Introduction

**Image Stitching** is a fundamental task in computer vision, where multiple images with overlapping fields of view are aligned and blended together to form a seamless panoramic image. This technique has broad applications in fields such as photography, robotics, remote sensing, and virtual reality.

In this project, we have explored and compared four different approaches to image stitching:

1. **Implementation from Scratch**: We used a custom image stitching pipeline which we built using basic computer vision techniques such as **keypoint detection, descriptor matching, homography estimation, image warping, and blending**.
2. **OpenCV Pipeline**: We built an image stitching pipeline using OpenCV functions.
3. **OpenCV's built-in** `cv2.Stitcher()`: We used a high-level class provided by **OpenCV** which abstracts the stitching pipeline and produces panoramas with minimal code.
4. **Deep Learning-Based Stitching**: We used a novel approach using a modified version of HomographyNet, which learns to estimate homography transformations directly from image pairs using convolutional neural networks.

## 1.2 Objective

The primary objective of this project is to implement and understand each pipeline in detail, evaluate their performance both qualitatively and quantitatively, and analyze their respective strengths and limitations in the context of real-world image stitching tasks.

# Chapter 2

# Approaches Used

## 2.1 Implementation From Scratch

We have created a fully customized image stitching pipeline using **NumPy**, **scikit-image** and **OpenCV**. This approach provides complete control over each stage of the pipeline. The major steps include:

**1. SIFT Keypoint Detection & Descriptor Extraction**

- We used OpenCV-based SIFT detector and limited the number of keypoints to 5000.
- These features are scale- and rotation-invariant and are represented as 128-dimensional descriptors.

**2. Descriptor Matching**

- We performed brute-force matching using Euclidean distance between descriptors.
- We also applied Lowe's ratio test to filter ambiguous matches and retain only high-confidence pairs.
- We used threshold ratio of 0.75.

**3. RANSAC for Robust Homography Estimation**

- We randomly sampled 4 keypoint pairs and estimated a homography using Direct Linear Transform (DLT).
- Then we counted inliers based on a geometric error threshold and retained the best-fit homography.
- We used threshold as 5.0 and 500 iterations.

**4. Pairwise Feature Matching**

- We constructed pairwise matches which maps image index pairs to homographies and inliers.
- We also created a unidirected graph representing connectivity of different images.

**5. Global Homography Computation**

- We selected the most connected image as the reference image.
- Then we computed transformation matrices from each image to the reference using accumulated homographies by travesing the graph using BFS.

**6. Warping and Stitching**

- We warped images using `skimage.transform.warp`.
- We composited warped images onto a larger canvas.
- We applied weighted blending and Gaussian Smoothening to ensure smooth seams.

**7. Cropping Black Borders**

- We detected bounding boxes around valid pixels in the stitched image.
- We cropped the result to remove excess black padding and maximize visible content.

## 2.2 OpenCV Pipeline

In this method we tried to implement image stitching from scratch using OpenCV functions.

**1. Feature Detection and Matching with OpenCV SIFT**

- We process each pair of overlapping images using OpenCV's `cv2.SIFT_create()` to extract scale and rotation invariant keypoints and descriptors.
- We performed matching using a brute-force matcher with Lowe's ratio test (threshold of 0.75) to retain only high-quality correspondences.
- We computed Homography using RANSAC to reject outliers.

**2. Image Warping and Translation**

- The homography matrix aligns the new image to the base image.
- The transformed corners of the new image and the original corners of the base image are used to compute the overall canvas bounds.
- A translation matrix ensures all content fits in the final stitched canvas.

**3. Feather Blending**

- We used a custom feather blending function to ensure smooth seams.
- The blending is based on weighted averaging, using the presence of pixels in both the base and warped images.

**4. Cropping Black Borders**

- After stitching, we removed black padding around the panorama by converting the image to grayscale, thresholding, and finding external contours.
- The largest contour is used to crop the image, yielding a clean and compact output.

## 2.3 Using cv2.Sticher()

OpenCV provides a powerful and high-level abstraction for panoramic image stitching through the `cv2.Stitcher_create()` class. It encapsulates the complete stitching pipeline and is optimized for both speed and robustness, especially for casual use cases like smartphone panoramas or real-time preview systems.

The internal workflow involves the following key steps:

1. **Feature Detection:**
   - Detects keypoints in all input images.
   - Uses ORB (Oriented FAST and Rotated BRIEF) by default for efficiency.
   - Optionally supports other feature detectors like SIFT or SURF.
2. **Feature Matching:**
   - Matches keypoints between overlapping image pairs.
   - Uses approximate nearest neighbor algorithms.
3. **Camera Parameter Estimation:**
   - Estimates camera parameters using feature correspondences.
   - Solves for homographies or affine transforms.
   - Applies bundle adjustment to optimize reprojection error.
4. **Seam Estimation:**
   - Finds optimal seams using graph-cut algorithms.
5. **Exposure Compensation:**
   - Balances brightness and contrast across images.
6. **Image Warping:**
   - Warps images to a common projection plane (e.g., cylindrical, spherical).
7. **Blending:**
   - Uses feathering or multi-band blending for seamless output.

## 2.4 Deep Learning Based Stitching

- To complement the traditional feature-based methods, we implemented a deep learning-based approach for image stitching using a convolutional neural network (CNN) to predict the homography between image pairs.
- The model learns to infer geometric transformations directly from pixel data, without relying on keypoint detection and matching.

### 2.4.1 Model Architecture

- The core of the pipeline is a deep neural network named *HomographyNet*, which accepts a pair of image patches as input and regresses the corresponding homography displacement vector.
- The network comprises a series of convolutional layers with ReLU activations and max-pooling to extract hierarchical features from the input.
- These are followed by fully connected layers that output an 8-dimensional vector representing the offsets between the four corner points of the original and warped patches. The model expects a 6-channel input formed by stacking two RGB image patches.

### 2.4.2 Dataset Generation

- To train the model, real image pairs are used along with an estimated ground-truth homography computed using a traditional ORB-based keypoint matcher followed by RANSAC.
- From these images, random patches are sampled from one image, and their corresponding warped patches are generated by applying the homography transformation to the second image.
- The model is trained to learn the displacement between the original patch corners and their transformed counterparts.
- Each sample includes a stacked patch pair as input and an 8-dimensional ground-truth label as the target output.

### 2.4.3 Training Procedure

- The training loop utilizes the Adam optimizer with a mean squared error loss function, and the model is trained over multiple epochs using mini-batches of the generated dataset.
- The goal is to minimize the difference between the predicted and ground-truth corner displacements, thereby allowing the network to learn spatial transformations from appearance cues alone.

### 2.4.4 Homography Prediction and Stitching

- After training, the network is used to infer the homography between a new pair of test images.
- A patch from a predefined location in both images is fed into the network to predict corner displacements.
- These displacements are used to compute a homography matrix, which is then applied to warp the second image.
- To stitch the two images together, the warped image is blended with the first image on a common canvas.
- Image boundaries and translations are computed to ensure a seamless combination and proper alignment.

### 2.4.5 Advantages and Use-Cases

- This approach enables homography estimation without relying on handcrafted features, which can be advantageous in low-texture or repetitive-pattern regions where traditional methods often fail.
- It also opens up possibilities for real-time applications, as the model can generalize to unseen image pairs after training.
- However, this method assumes relatively small perspective changes and consistent illumination across images, which can affect prediction accuracy in more challenging scenarios.

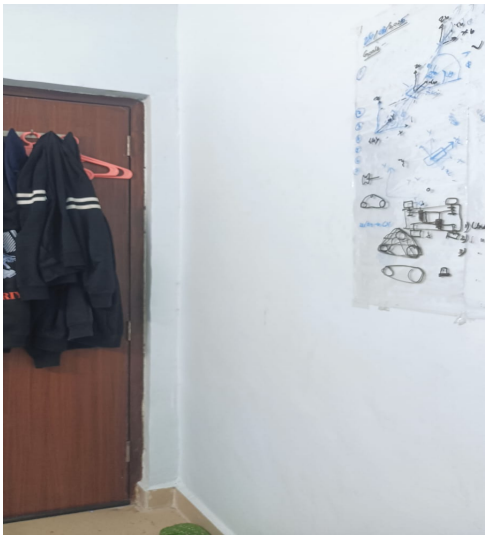
# Chapter 3

# Results and Observations

## 3.1 Test Images Used

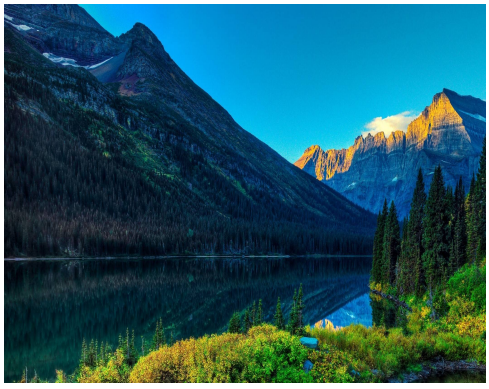
We have used the following images as a test case for all our approaches -

(a) room_2



(b) room_3

Figure 3.1: Test Set 1



(a) sample_1

(b) sample_2

Figure 3.2: Test Set 2

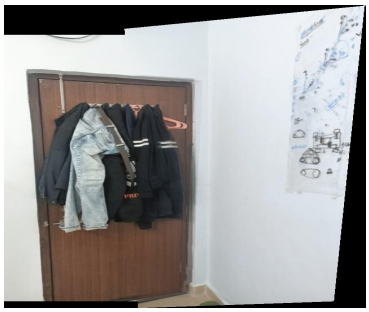
(a) building_2

(b) building_3

Figure 3.3: Test Set 3

## 3.2 Stitched Images

### 3.2.1 Implementation From Scratch



(a) scratch_1

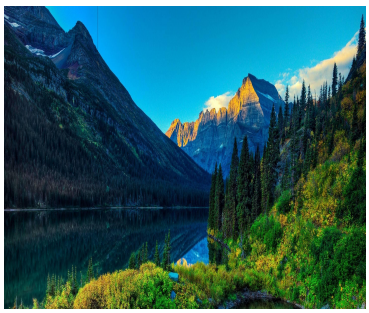
(b) scratch_2

(c) scratch_3

Figure 3.4: scratch outputs

### 3.2.2 OpenCV Pipeline

(a) opencv_pipeline_1



(b) opencv_pipeline_2

(c) opencv_pipeline_3

Figure 3.5: opencv pipeline outputs

### 3.2.3 Using cv2.Sticher()

(a) cv2_stitcher_1

(b) cv2_stitcher_2

(c) cv2_stitcher_3

Figure 3.6: cv2_Stitcher outputs

### 3.2.4 Deep Learning Based Stitching

(a) dl_1

(b) dl_2

(c) dl_3

Figure 3.7: DL outputs

## 3.3 Observation

The following table summarizes our observation -

Table 3.1: Comparison of Stitching Techniques

| **Method** | **Output Quality** | **Seam Visibility** | **Speed** | **Robustness** | **Warp** |
|---|---|---|---|---|---|
| Scratch | Moderate | Moderate | Slow | High | Moderate |
| OpenCV Pipeline | Moderate | High | Moderate | High | Moderate |
| OpenCV Stitcher | High | Low | Fast | Moderate | High |
| Deep Learning Model | Moderate | High | Fast | High | Moderate |

*Apart from these, the cv2.Stitcher() stitched images had panorama view.

# Chapter 4

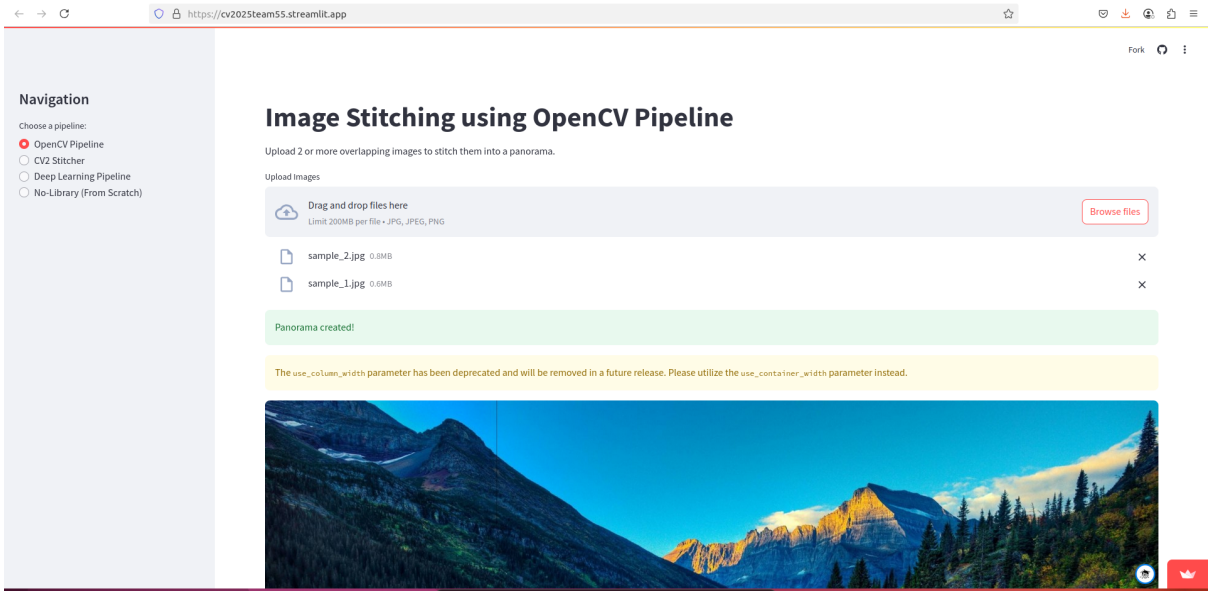
# Project Links

## 4.1 GitHub Link

GitHub Repository

## 4.2 Deployment Link

Streamlit App




(a) streamlit

Figure 4.1: Streamlit Deployment

# Chapter 5

# Individual Contributions

## 5.1 Priyabrata Saha

Implemented Method 1 - **Implementation From Scratch**

## 5.2 Suddala Madhu

Implemented Method 2 - **OpenCV Pipeline**

## 5.3 Projat Banerjee

Implemented Method 3 - **cv2.Stitcher()**

## 5.4 Vishwanath R

Implemented Method 4 - **Deep Learning-Based Stitching**

# Chapter 6

# Conclusion

In this project, we explored and compared four different approaches to image stitching: a complete implementation from scratch, an OpenCV-based pipeline, OpenCV's built-in Stitcher module, and a deep learning-based homography estimation model.

Each method demonstrated distinct strengths and trade-offs:

- **Implementation From Scratch:** Provided in-depth control and deepened understanding of each stage in the stitching pipeline. However, it is computationally intensive and less robust to challenging cases like misalignment or low-texture regions.
- **OpenCV Pipeline:** Balanced between custom control and leveraging optimized libraries. Produced visually pleasing results with smooth transitions and better modularity than the black-box Stitcher.
- **cv2.Stitcher():** Easiest to use, very fast, and robust for common scenarios. However, it lacks transparency and fine-tuning capabilities.
- **Deep Learning Model:** Showed promise in learning robust homography estimation from data. When trained effectively, it offered high stitching quality and generalization, but it required substantial data and training time.

Overall, the project successfully demonstrated the practical differences between classical and learning-based vision systems for panorama creation. It also emphasized the importance of good feature matching, blending strategies, and geometric transformation handling in image stitching tasks.

# Chapter 7

# Future Work

While the current implementations successfully stitch images under various scenarios, there are several potential enhancements and extensions that could be pursued in the future:

- **Improved Blending Techniques:** Implementing advanced blending methods such as pyramid blending or Poisson blending could reduce ghosting and enhance visual seamlessness in overlapping regions.
- **Automatic Image Ordering:** Incorporating a mechanism to automatically determine the optimal sequence of image alignment based on feature matches would improve scalability and robustness for unordered inputs.
- **Enhanced Deep Learning Model:** Training the deep learning-based homography estimator on a more diverse and larger dataset, and experimenting with more complex architectures (e.g., attention-based models), could further improve generalization and performance.
- **360-Degree and Vertical Panoramas:** Extending the system to handle spherical or cylindrical projections would allow stitching of full 360-degree or vertical panoramas.
- **Video Stitching:** Applying the pipeline to sequential video frames for real-time panoramic video generation is a challenging yet rewarding direction.
- **Real-Time Deployment:** Optimizing the pipeline for real-time or near-real-time execution using GPU acceleration (e.g., via CUDA or TensorRT) could make it suitable for mobile or embedded applications.

# Bibliography

[1] OpenCV Documentation. *https://docs.opencv.org/*.

[2] DeTone, D., Malisiewicz, T., & Rabinovich, A. (2016). *Deep Image Homography Estimation*. arXiv preprint arXiv:1606.03798.

[3] Van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., ... & Yu, T. (2014). *scikit-image: image processing in Python*. PeerJ 2:e453.

[4] PyImageSearch Tutorials. *https://pyimagesearch.com/*.

[5] Hartley, R., & Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press.