Project Report on Computer Vision : Spring 2019
By
Shower Test

Priyanka Reballi (2018900020)
K L Bhanu Moorthy (20172147)
Arun Kumar Subramanian (2018900014)

Github link to the project: JSPEC

This report is submitted as part of the Computer Vision Course, Spring 2019 -Evaluation 1. The objective is to implement the paper **"Joint Spectral Correspondence for Disparate Image Matching".**

About

The problem of matching two images belonging to the same scene with different illuminations, haze or captured at different times of a day is a difficult problem. Existing

image matching algorithms fail with such image pairs due to their non-robustness in understanding the scene using pixel or gradient level structures. A new approach to this problem presented in this paper essentially revolves around matching eigenfunction representations of the two images and they claim that such representations capture some 'persistent regions' of the images. These regions act as correspondences between the two images and are repeatable across images.

Procedure

The first step is to concatenate both the images. This is an essential step as the eigenfunction representations of both the images combined and separate is a more robust approach to image matching. Using the pixels in this concatenated image as nodes and adding edges between adjacent nodes, we compute affinities (a measure of similarity) amongst each pair of nodes. The eigenvalues to this created matrix give us the eigenfunction representation for the images which can then be used to rematch both the images. The following steps provide the essential implementation details of the algorithm:

- Extract Dense SIFT features of the two images at specific spatial sampling and create the image graph.
- Calculate the affinity matrix of the image graph.
- Compute the eigendecomposition of the affinity matrix
- Reconstruct both the images using the eigenvectors.
- Extract and match feature using these representations.

Work Done

We are using the **Symbench** image dataset presented in the article Image matching using local symmetry features which consists of 46 image pairs varying in illumination, captured time of the day and low dynamic range along with their homography transformations. We have developed the image graph, the affinity matrices and computed their eigendecompositions.

Remaining Work

- Sharpening the Ellipse Fitting + Computing SIFT descriptor on 5times ellipse fitting. (**Partially done: Were able to accomplish circular MSER detection)**
- Bi-directional SIFT match (**Partially done: Uni-directional Knn match)**
- Performance metrics. (**Note done).**

Code walkthrough:

- Create keypoints at various scales and obtain a SIFT descriptor.
  - Code to get keypoints at varoius scales:

```
class DenseDetector():
        def __init__(self, step_size=20, feature_scale=20, img_bound=20):
            # Create a dense feature detector
            self.initXyStep = step_size
            self.initFeatureScale = feature_scale
            self.initImgBound = int(img_bound)

        def detect(self, img):
            keypoints = []
            cols, rows = img.shape[:2]
            for i, x in enumerate(range(self.initImgBound, rows, self.initXyStep)):
                for j, y in enumerate(range(self.initImgBound, cols, self.initXyStep)):
                    keypoints.append(cv2.KeyPoint(float(x), float(y),
                                                  self.initFeatureScale))
            print("arun printing each iteration i,j",i,j)
```

return keypoints

- SIFT Descriptor these scales:
  sift=cv2.xfeatures2d.SIFT_create()
  kpOutput,denseFeat=sift.compute(im,kp)

- Combining the SIFT obtained at both scales:
  combinedImage =np.concatenate((im1combinedDenseFeat,im2combinedDenseFeat) ,axis=0)

- Creating similarityMatrix, creating image Laplacian and Eigen decomposition:
  - Similarity Matrix:
    similarityMatrix = np.dot(combinedImage,combinedImage.T)
  - Image Laplacian:
    imageLaplacian = csgraph.laplacian(similarityMatrix, normed=True)
  - Eigen decomposition and picking the top N:
    V,E = LA.eig(imageLaplacian)
    fiveBestEigen = E[:,np.in1d(V, np.sort(np.partition(V,6))[1:6])][:,:3]

- MSER Match:
  - MSER Detectors of keypoints:
    mserDet = cv2.MSER_create()
    kp2 = mserDet.detect(im2)
    kp1 = mserDet.detect(im1)

  - SIFT Descriptors on above keypoints:
    sift1=cv2.xfeatures2d.SIFT_create()

kp1, des1 = sift1.compute(im1, kp1)

- Performing KNN match

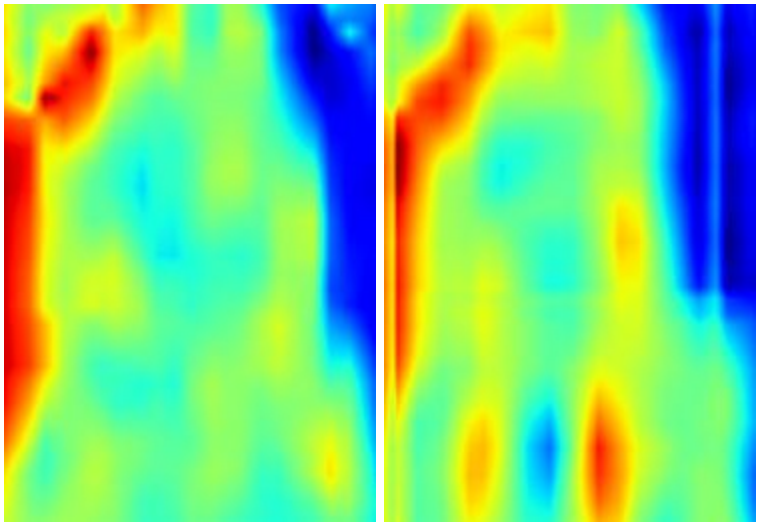
  sift1=cv2.xfeatures2d.SIFT_create()
  kp1, des1 = sift1.compute(im1, kp1)

Results:

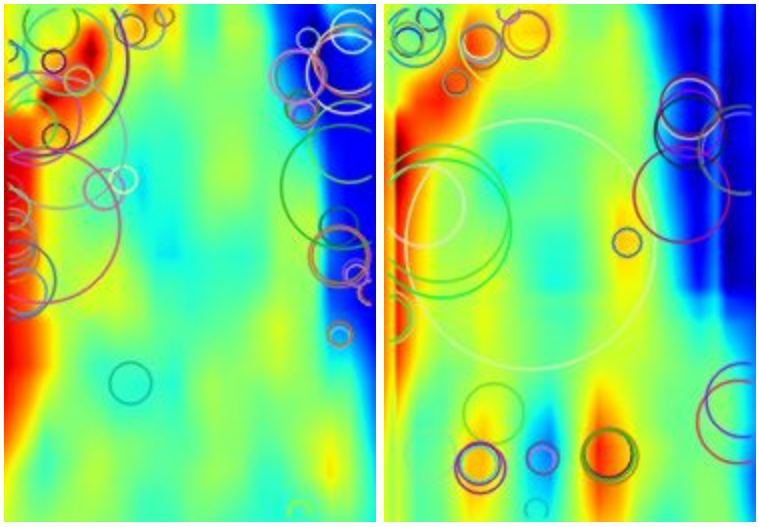
Images:

Image1:

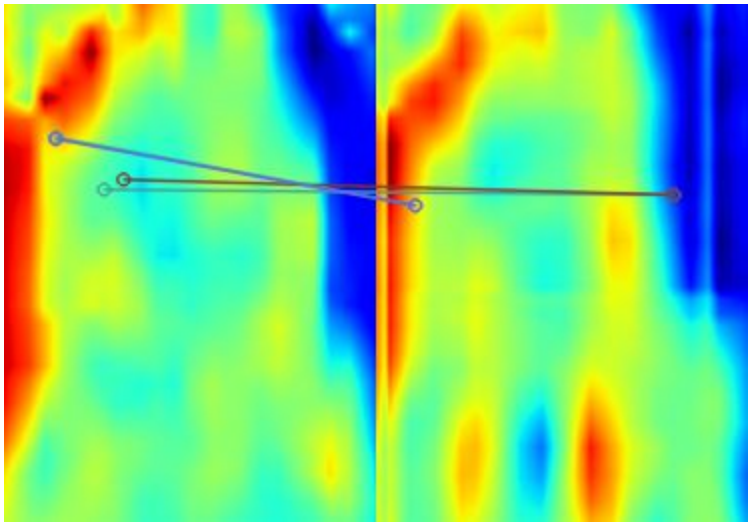

Eigen maps:



MSER Detectors:



Matches on the EigenMap:



Matches on the original image:

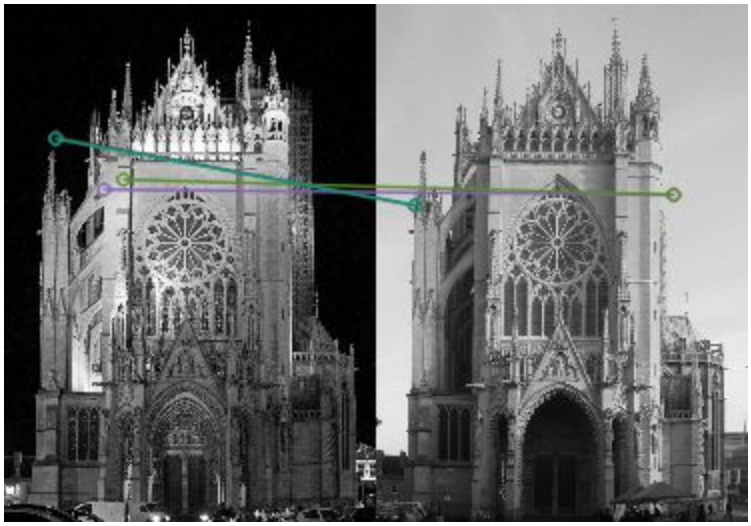
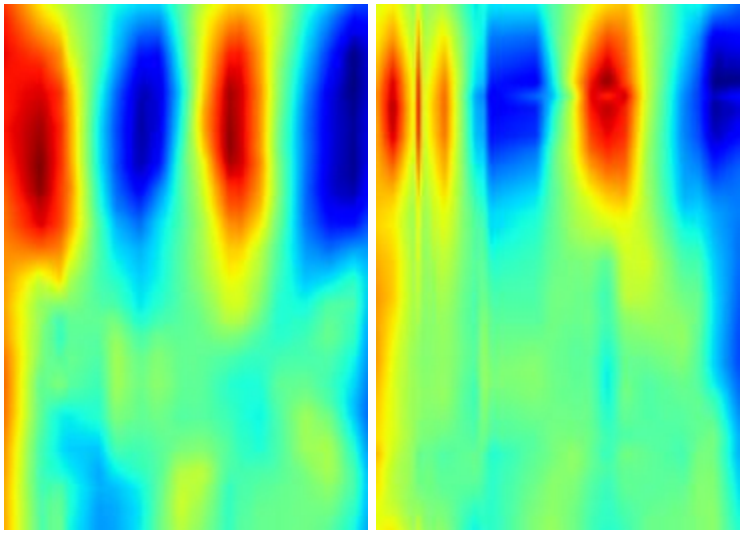
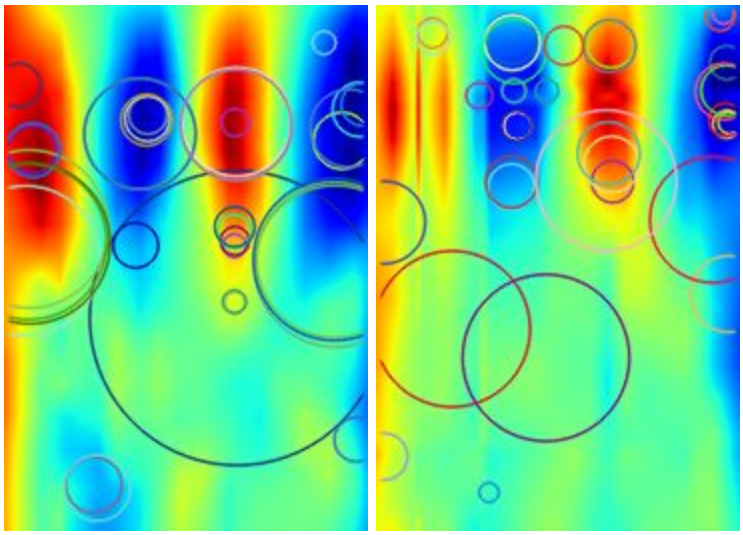


Image2:

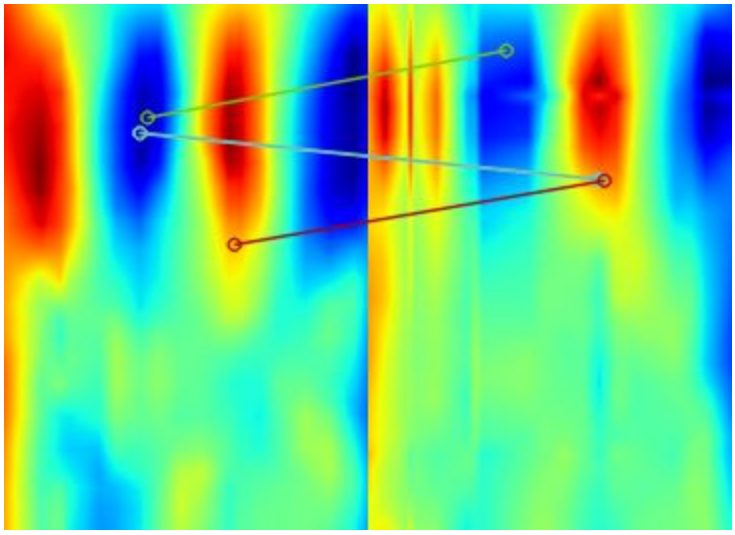
Eigen maps:



MSER Detectors:



Matches on the EigenMap:



Matches on the original image:

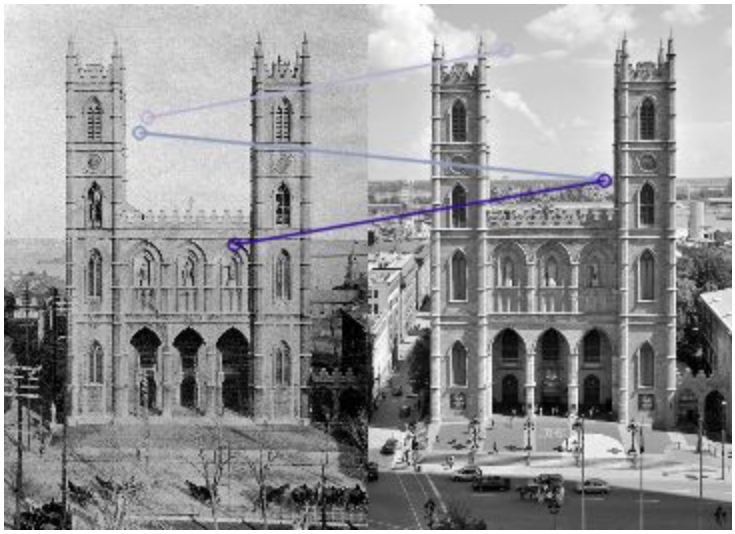
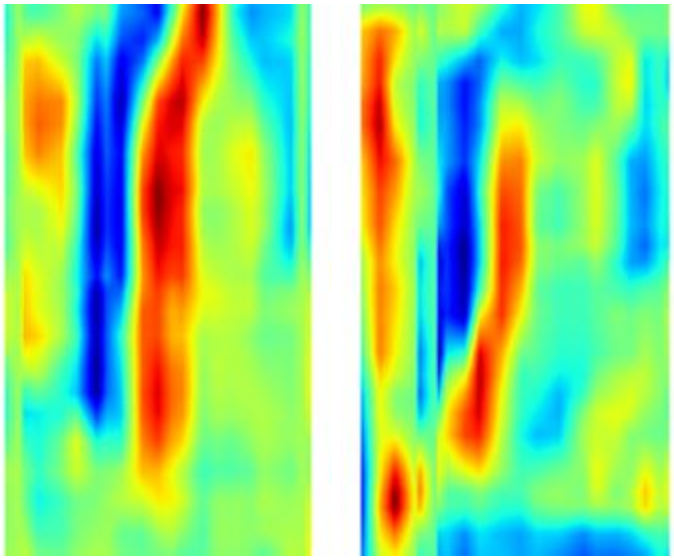
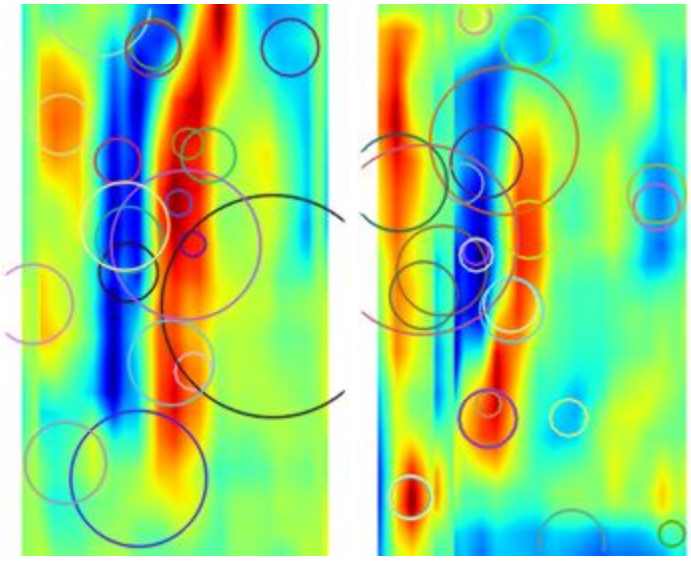


Image3:

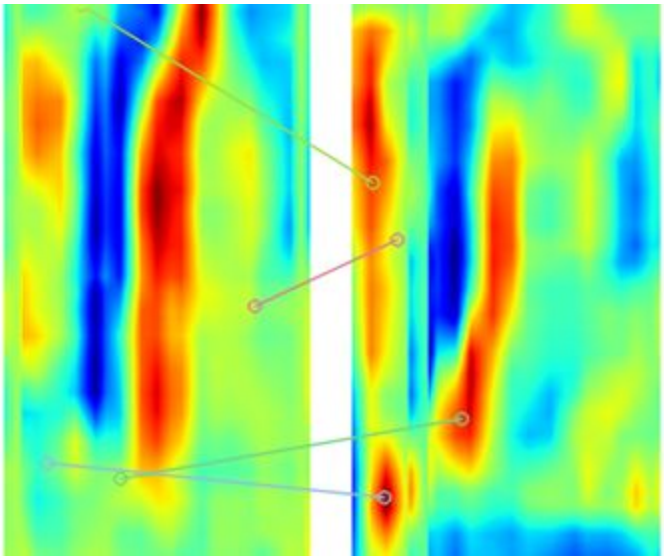
Eigen maps:



MSER Detectors:



Matches on the EigenMap:



Matches on the original image: