# SIFT - Scale Invariant Feature Transform

(Aditya, Aishwarya, Moni, Punit, Kunal)

It detects salient, stable feature points in an image. Provides set of “features” that describes small image region around the point. Features are invariant to rotation and scale.

SIFT Algorithm used in our code:

1. Determine approximate location and scale of salient feature points (key points).
2. Refine their location and scale.
3. Determine orientation for each key point.
4. Determine descriptors for each key point.

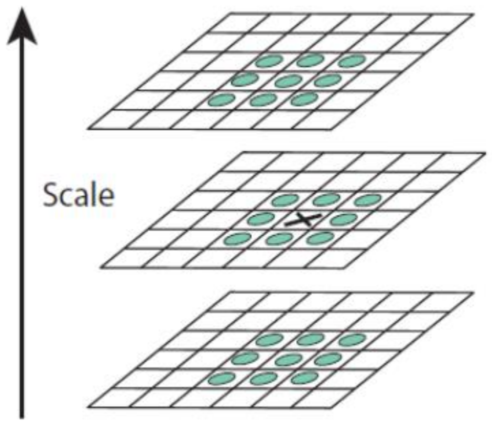
Step 1: Key point location

Look for intensity changes using the Difference of Gaussians (DoG) at two nearby scales.

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma). \end{aligned}$$

Maxima or Minima of DoG detected by comparing pixel to its nearest neighbour at current & adjacent scale. Key points are maxima or minima in the “scale-space-pyramid”. We get both the location as well as the scale of the key point.



Step 2: Refine key point location

Key point location and scale is discrete. Can interpolate location for greater accuracy. Express the DoG function in a small 3D neighbourhood around a key point 2nd order Taylor-series

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}} \mathbf{x} + \frac{1}{2} \mathbf{x}^{\mathbf{T}} \frac{\partial^2 D}{\partial \mathbf{x}^2} \mathbf{x}$$

$$\hat{\mathbf{x}} = - \frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1} \frac{\partial D}{\partial \mathbf{x}}$$

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2} \frac{\partial D^T}{\partial \mathbf{x}} \hat{\mathbf{x}}$$

Remove those key points with a value of | D(x̂) | less than 0.03. Eliminate edge key points. Discard the key points lying on edges even if they have high response in DoG filter. 2nd derivative in DoG is a Hessian matrix. Eigenvalues of H give the maximal and minimal principal curvature of the surface. Key point that is not an edge has high maximal and minimal curvature

$$\mathbf{H} = \left[ \begin{array}{cc} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{array} \right]$$

Eliminate the key points that have a ratio between the principal curvatures greater than r.

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} < \frac{(r+1)^2}{r}$$

Step 3 - Orientation Assignment

Compute gradient magnitudes and orientations in a small window around the key point at the appropriate scale.

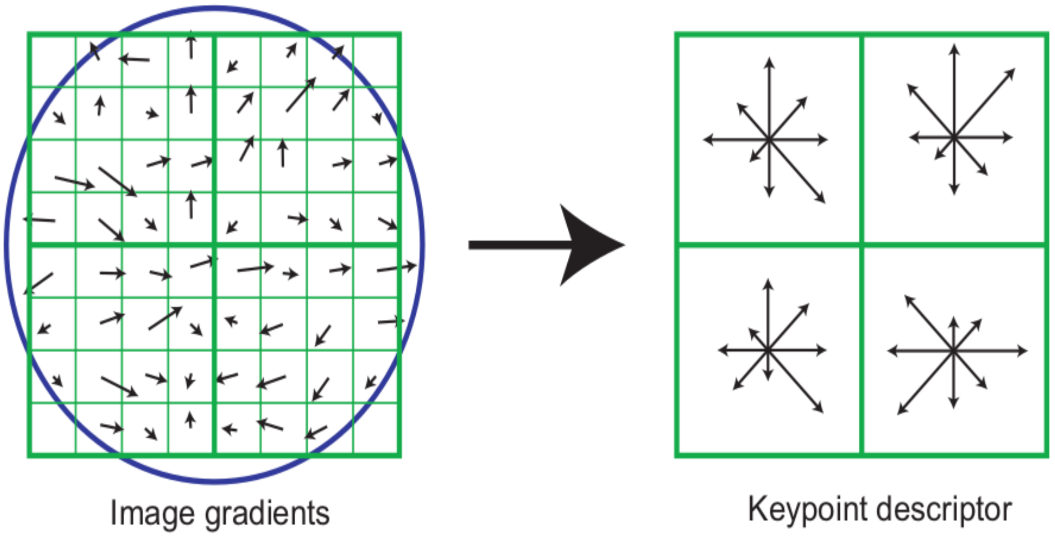
$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = tan^{-1}(L(x+1,y) - L(x-1,y) / (L(x,y+1) - L(x,y-1))$$

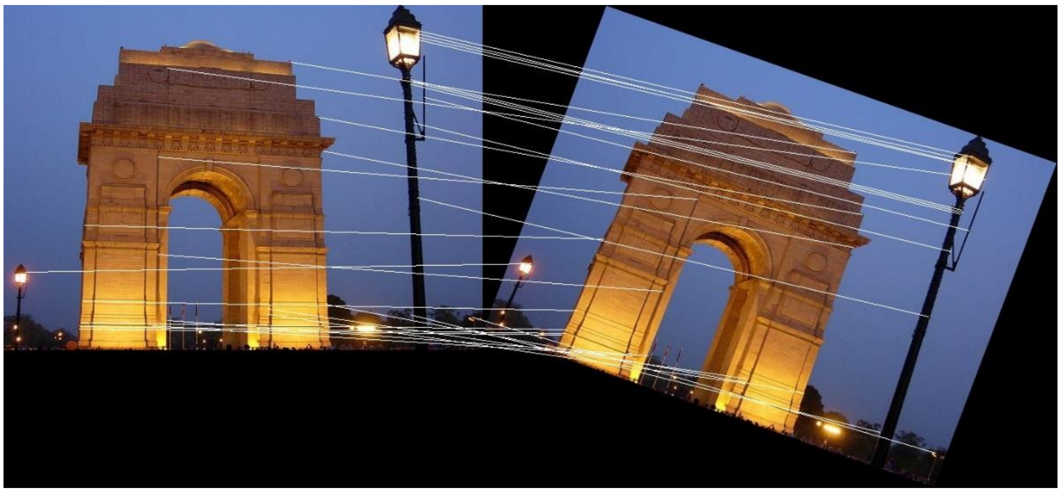
Assign dominant orientation as key point orientation. For multiple peaks, create separate descriptor for each orientation.

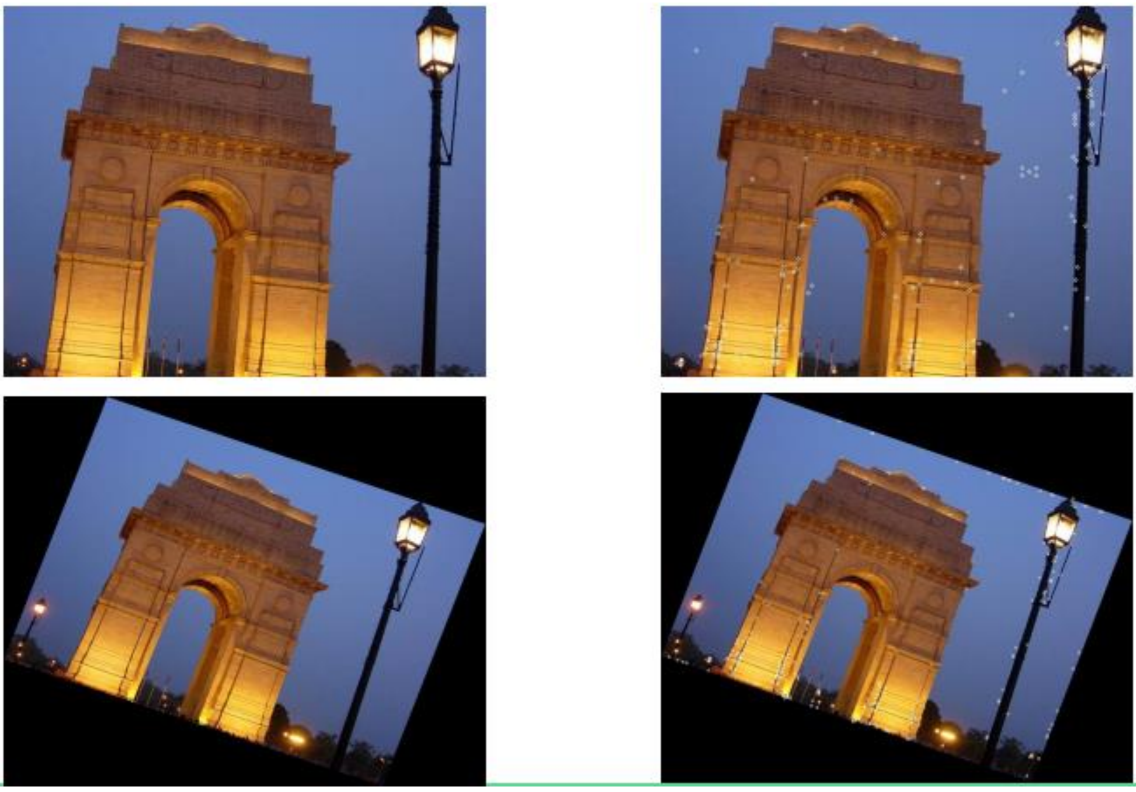
Step 4 - Key point Descriptor

Divide small region around key point into nxn cells with cell size 4x4. Build a gradient orientation histogram in each cell. Histogram entry weighted by the gradient magnitude and a Gaussian weighting function. For orientation invariance, the coordinates of the descriptor and the gradient orientations are rotated relative to the key point orientation. We get descriptor size of r x n x n size (r = bins in histogram). Histogram entries are weighted by gradient magnitude. Descriptor vector is normalized to unit magnitude to make it intensity invariant.

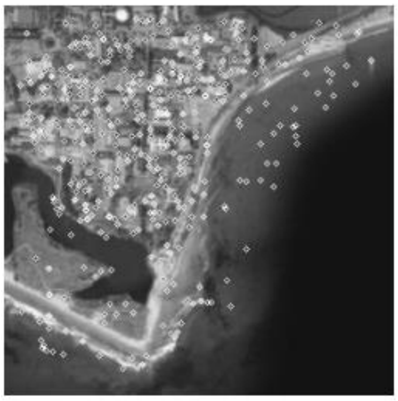


Results:

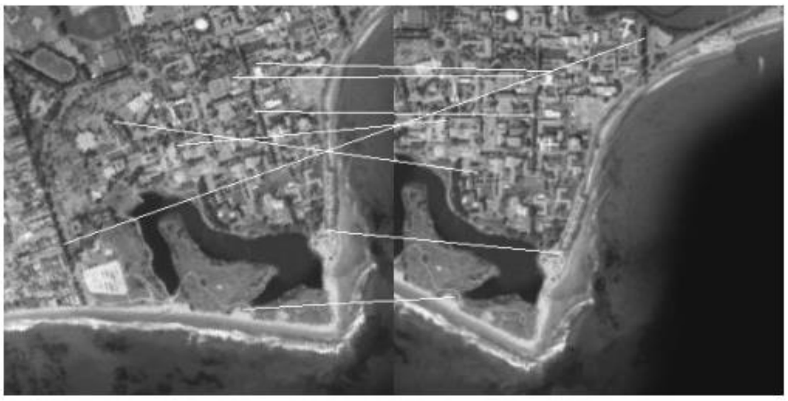




Key points in Image



Keypoint Detection



Keypoint matching between the images

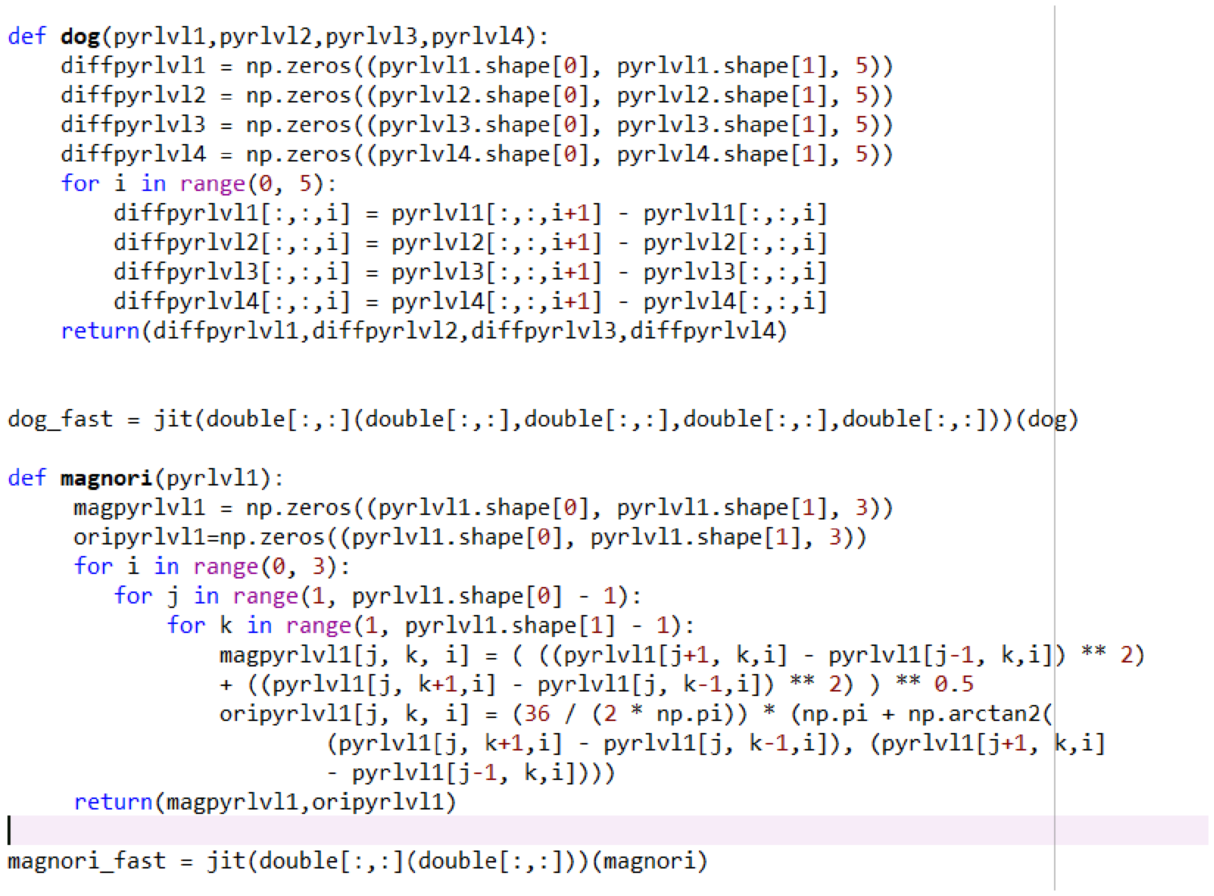
# Parallel code

<u>Used Numba</u>

Numba translates Python functions to optimized machine code at runtime using the industry-standard LLVM compiler library. Numba-compiled numerical algorithms in Python can approach the speeds of C or FORTRAN.

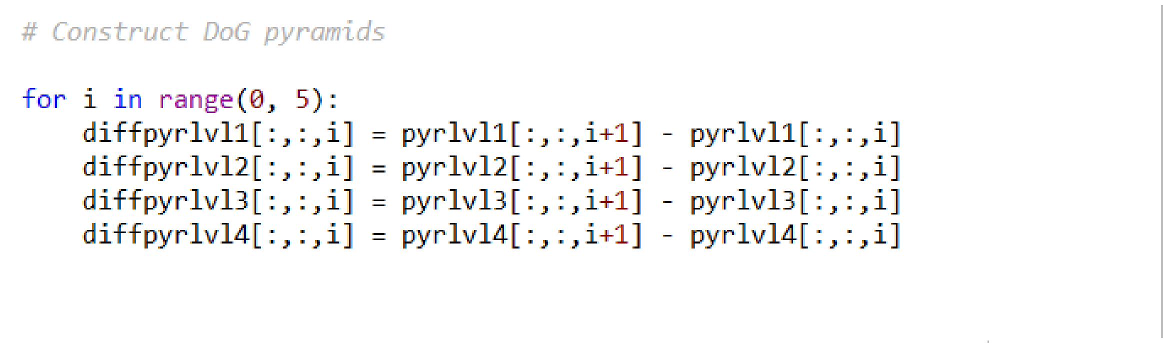
Numba is designed to be used with NumPy arrays and functions. Numba generates specialized code for different array data types and layouts to optimize performance. Special decorators can create universal functions that broadcast over NumPy arrays just like NumPy functions do.

# Written Serial code and parallel code in different folders and to run them, run main.py file.

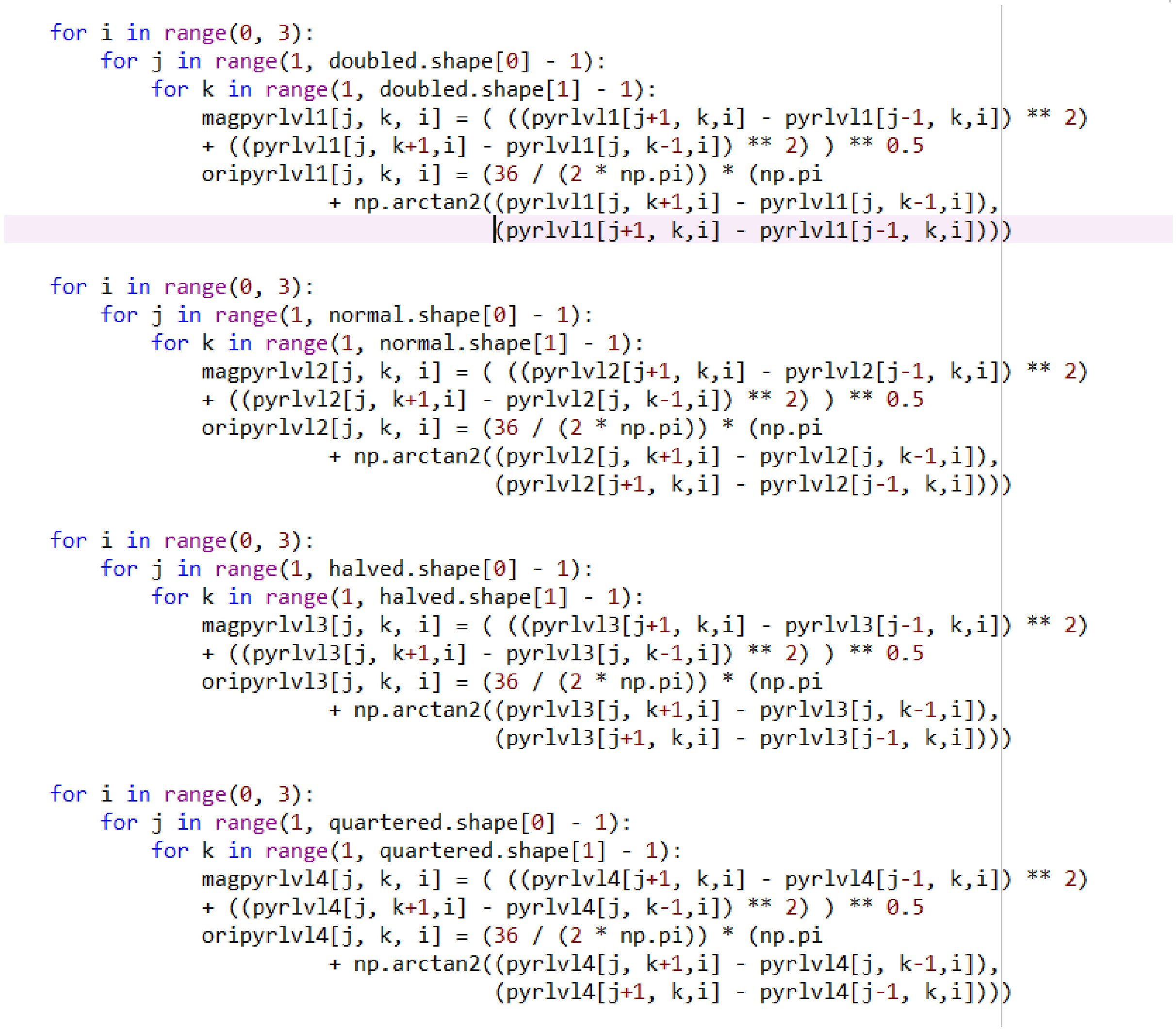


```python
def dog(pyrlvl1,pyrlvl2,pyrlvl3,pyrlvl4):
    diffpyrlvl1 = np.zeros((pyrlvl1.shape[0], pyrlvl1.shape[1], 5))
    diffpyrlvl2 = np.zeros((pyrlvl2.shape[0], pyrlvl2.shape[1], 5))
    diffpyrlvl3 = np.zeros((pyrlvl3.shape[0], pyrlvl3.shape[1], 5))
    diffpyrlvl4 = np.zeros((pyrlvl4.shape[0], pyrlvl4.shape[1], 5))
    for i in range(0, 5):
        diffpyrlvl1[:,:,i] = pyrlvl1[:,:,i+1] - pyrlvl1[:,:,i]
        diffpyrlvl2[:,:,i] = pyrlvl2[:,:,i+1] - pyrlvl2[:,:,i]
        diffpyrlvl3[:,:,i] = pyrlvl3[:,:,i+1] - pyrlvl3[:,:,i]
        diffpyrlvl4[:,:,i] = pyrlvl4[:,:,i+1] - pyrlvl4[:,:,i]
    return(diffpyrlvl1,diffpyrlvl2,diffpyrlvl3,diffpyrlvl4)


dog_fast = jit(double[:,:](double[:,:],double[:,:],double[:,:],double[:,:]))(dog)

def magnori(pyrlvl1):
     magpyrlvl1 = np.zeros((pyrlvl1.shape[0], pyrlvl1.shape[1], 3))
     oripyrlvl1=np.zeros((pyrlvl1.shape[0], pyrlvl1.shape[1], 3))
     for i in range(0, 3):
        for j in range(1, pyrlvl1.shape[0] - 1):
            for k in range(1, pyrlvl1.shape[1] - 1):
                magpyrlvl1[j, k, i] = ( ((pyrlvl1[j+1, k,i] - pyrlvl1[j-1, k,i]) ** 2)
                + ((pyrlvl1[j, k+1,i] - pyrlvl1[j, k-1,i]) ** 2) ) ** 0.5
                oripyrlvl1[j, k, i] = (36 / (2 * np.pi)) * (np.pi + np.arctan2(
                        (pyrlvl1[j, k+1,i] - pyrlvl1[j, k-1,i]), (pyrlvl1[j+1, k,i]
                        - pyrlvl1[j-1, k,i])))
     return(magpyrlvl1,oripyrlvl1)

magnori_fast = jit(double[:,:](double[:,:]))(magnori)
```

## Serial Code



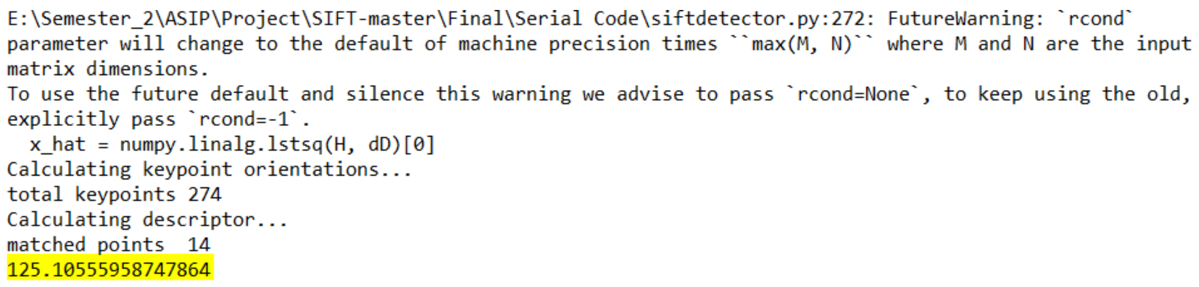
```python
# Construct DoG pyramids

for i in range(0, 5):
    diffpyrlvl1[:,:,i] = pyrlvl1[:,:,i+1] - pyrlvl1[:,:,i]
    diffpyrlvl2[:,:,i] = pyrlvl2[:,:,i+1] - pyrlvl2[:,:,i]
    diffpyrlvl3[:,:,i] = pyrlvl3[:,:,i+1] - pyrlvl3[:,:,i]
    diffpyrlvl4[:,:,i] = pyrlvl4[:,:,i+1] - pyrlvl4[:,:,i]
```



```python
for i in range(0, 3):
    for j in range(1, doubled.shape[0] - 1):
        for k in range(1, doubled.shape[1] - 1):
            magpyrlvl1[j, k, i] = ( ((pyrlvl1[j+1, k,i] - pyrlvl1[j-1, k,i]) ** 2)
            + ((pyrlvl1[j, k+1,i] - pyrlvl1[j, k-1,i]) ** 2) ) ** 0.5
            oripyrlvl1[j, k, i] = (36 / (2 * np.pi)) * (np.pi
                      + np.arctan2((pyrlvl1[j, k+1,i] - pyrlvl1[j, k-1,i]),
                                   (pyrlvl1[j+1, k,i] - pyrlvl1[j-1, k,i])))

for i in range(0, 3):
    for j in range(1, normal.shape[0] - 1):
        for k in range(1, normal.shape[1] - 1):
            magpyrlvl2[j, k, i] = ( ((pyrlvl2[j+1, k,i] - pyrlvl2[j-1, k,i]) ** 2)
            + ((pyrlvl2[j, k+1,i] - pyrlvl2[j, k-1,i]) ** 2) ) ** 0.5
            oripyrlvl2[j, k, i] = (36 / (2 * np.pi)) * (np.pi
                      + np.arctan2((pyrlvl2[j, k+1,i] - pyrlvl2[j, k-1,i]),
                                   (pyrlvl2[j+1, k,i] - pyrlvl2[j-1, k,i])))

for i in range(0, 3):
    for j in range(1, halved.shape[0] - 1):
        for k in range(1, halved.shape[1] - 1):
            magpyrlvl3[j, k, i] = ( ((pyrlvl3[j+1, k,i] - pyrlvl3[j-1, k,i]) ** 2)
            + ((pyrlvl3[j, k+1,i] - pyrlvl3[j, k-1,i]) ** 2) ) ** 0.5
            oripyrlvl3[j, k, i] = (36 / (2 * np.pi)) * (np.pi
                      + np.arctan2((pyrlvl3[j, k+1,i] - pyrlvl3[j, k-1,i]),
                                   (pyrlvl3[j+1, k,i] - pyrlvl3[j-1, k,i])))

for i in range(0, 3):
    for j in range(1, quartered.shape[0] - 1):
        for k in range(1, quartered.shape[1] - 1):
            magpyrlvl4[j, k, i] = ( ((pyrlvl4[j+1, k,i] - pyrlvl4[j-1, k,i]) ** 2)
            + ((pyrlvl4[j, k+1,i] - pyrlvl4[j, k-1,i]) ** 2) ) ** 0.5
            oripyrlvl4[j, k, i] = (36 / (2 * np.pi)) * (np.pi
                      + np.arctan2((pyrlvl4[j, k+1,i] - pyrlvl4[j, k-1,i]),
                                   (pyrlvl4[j+1, k,i] - pyrlvl4[j-1, k,i])))
```

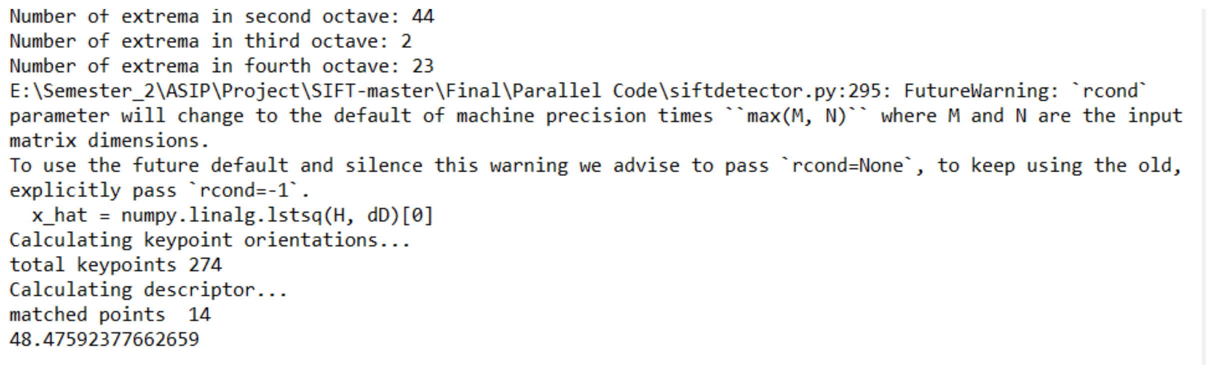
# Performance screenshots

## Serial



```
E:\Semester_2\ASIP\Project\SIFT-master\Final\Serial Code\siftdetector.py:272: FutureWarning: `rcond`
parameter will change to the default of machine precision times ``max(M, N)`` where M and N are the input
matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=None`, to keep using the old,
explicitly pass `rcond=-1`.
  x_hat = numpy.linalg.lstsq(H, dD)[0]
Calculating keypoint orientations...
total keypoints 274
Calculating descriptor...
matched points  14
125.10555958747864
```

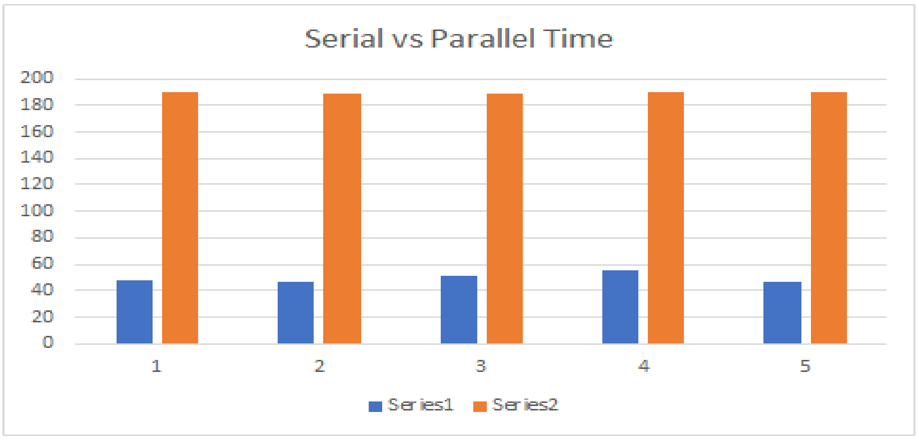
## Parallel



```
Number of extrema in second octave: 44
Number of extrema in third octave: 2
Number of extrema in fourth octave: 23
E:\Semester_2\ASIP\Project\SIFT-master\Final\Parallel Code\siftdetector.py:295: FutureWarning: `rcond`
parameter will change to the default of machine precision times ``max(M, N)`` where M and N are the input
matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=None`, to keep using the old,
explicitly pass `rcond=-1`.
  x_hat = numpy.linalg.lstsq(H, dD)[0]
Calculating keypoint orientations...
total keypoints 274
Calculating descriptor...
matched points  14
48.47592377662659
```

As we have seen performance has greatly increased from 125 seconds to 48 seconds due to parallel computing.

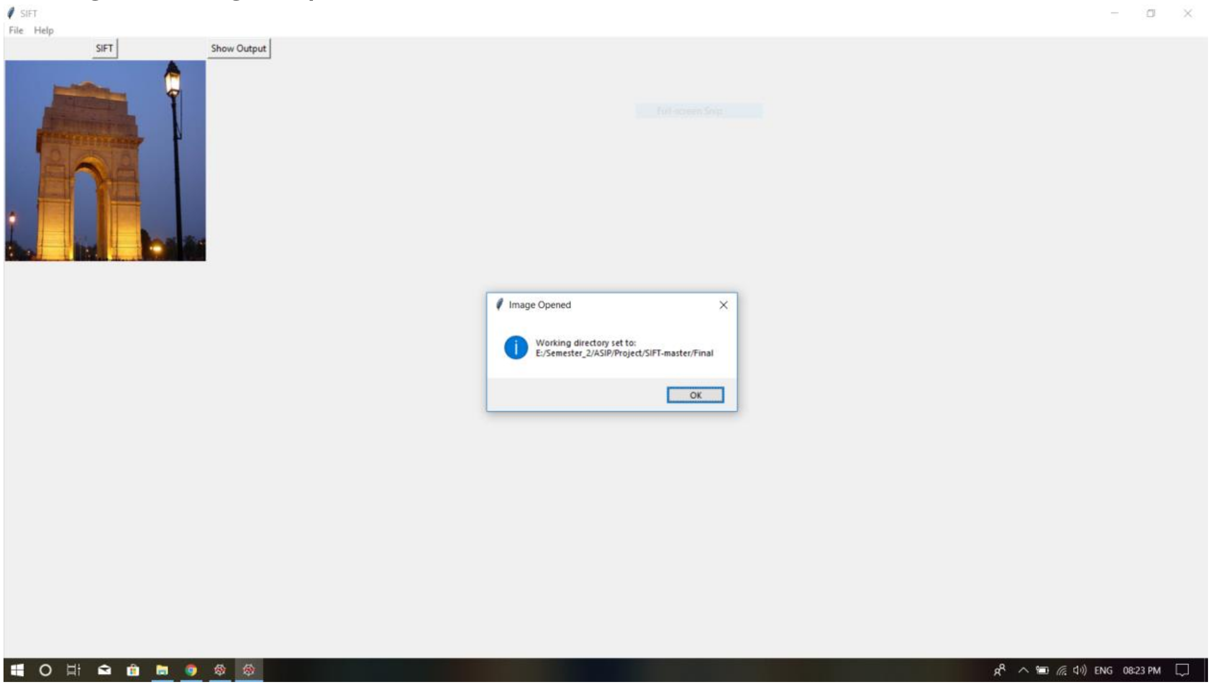
## Performance Checking:



- We have used **threading** in the UI code.
- Once we click on the SIFT button, a separate thread works on execution of the SIFT algorithm.
- This has improved performance.
- The Code used in UI is parallel code.
- The overhead of code in parallel is because of tkinter.

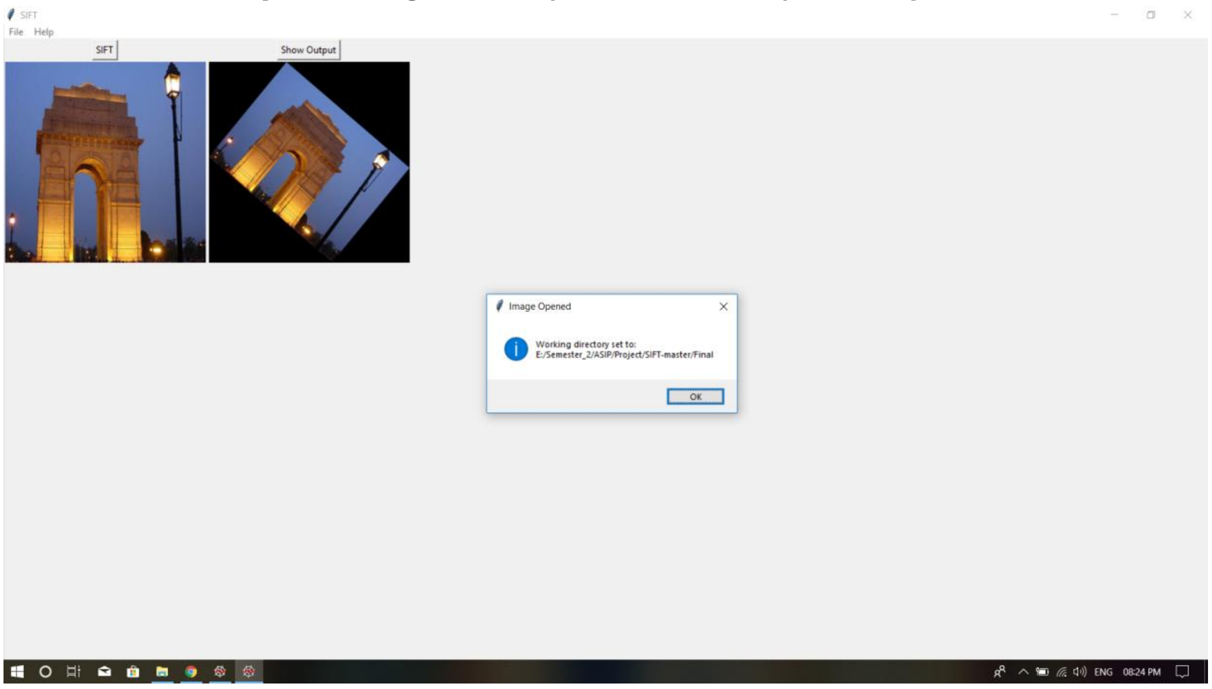
## Procedure for SIFT-

1. Select File → **Open Image 1** and pop-up appears.
2. Set directory to the input image folder. Output directory can be changed using help tab.



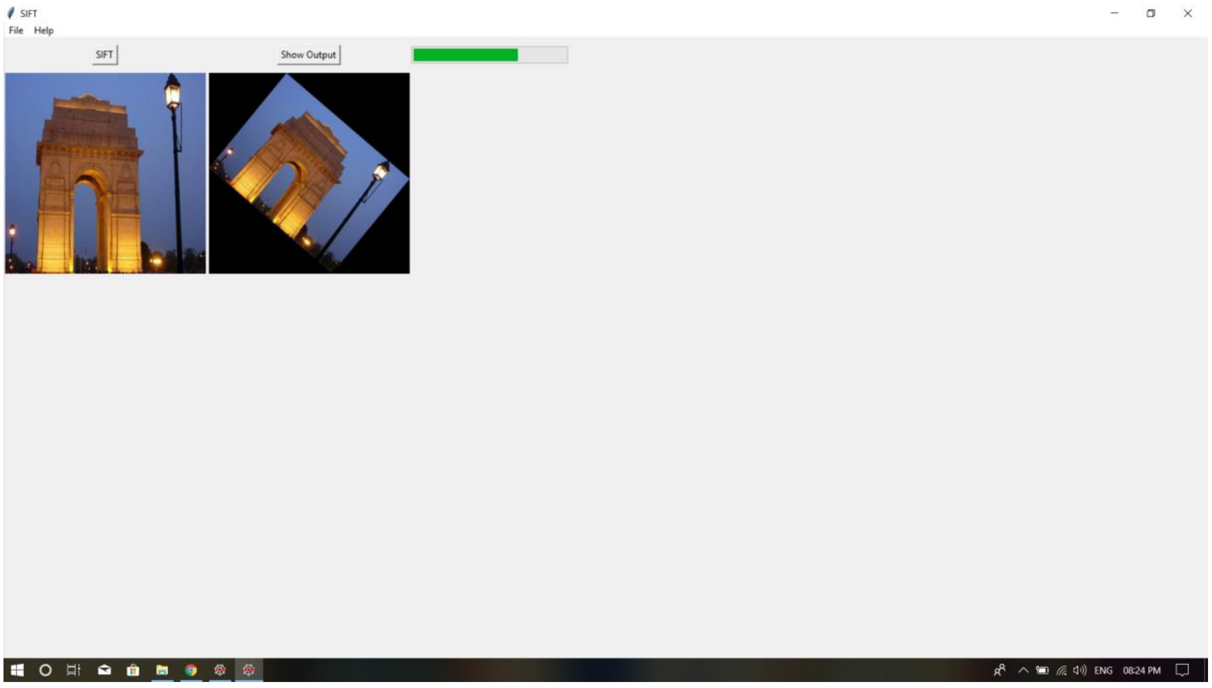
Working directory set to:
E:/Semester_2/ASIP/Project/SIFT-master/Final

3. Select File → **Open Image 2** to open second input image.



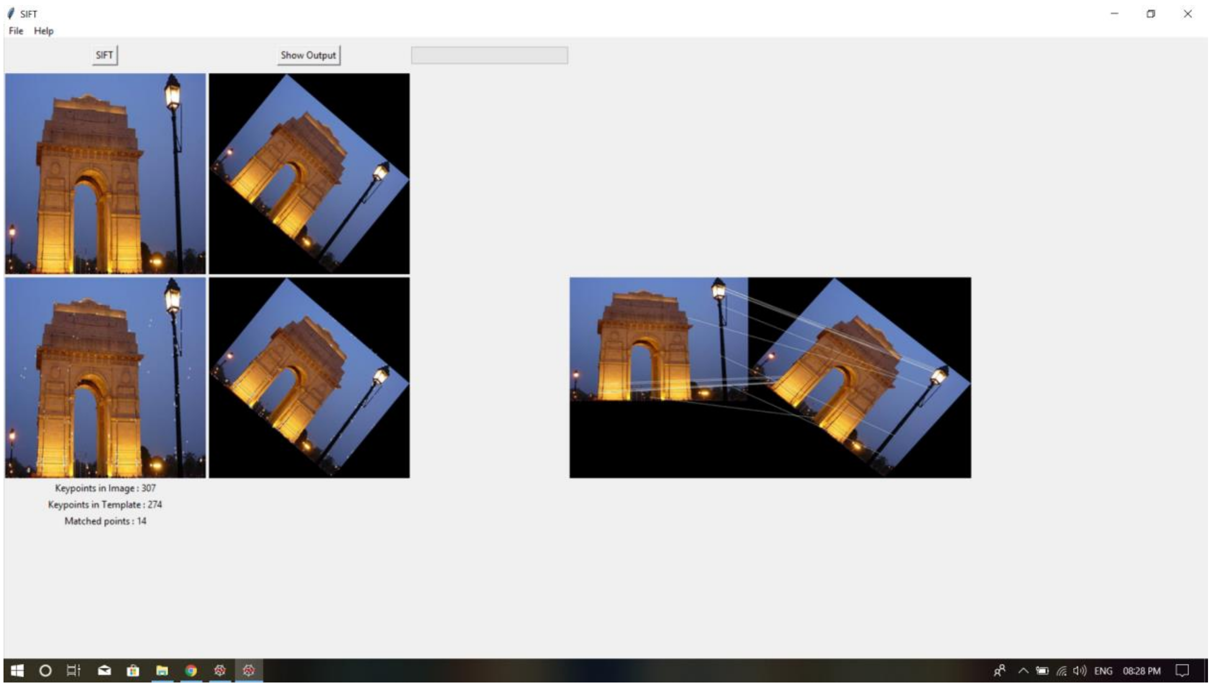
Working directory set to:
E:/Semester_2/ASIP/Project/SIFT-master/Final

4. Now click **SIFT** button. Progress bar appears.



5. Click **Show Output** button after the progress bar stops.

6. Results are as displayed.



Keypoints in Image : 307
Keypoints in Template : 274
Matched points : 14

Key points are identified and SIFT is successfully applied