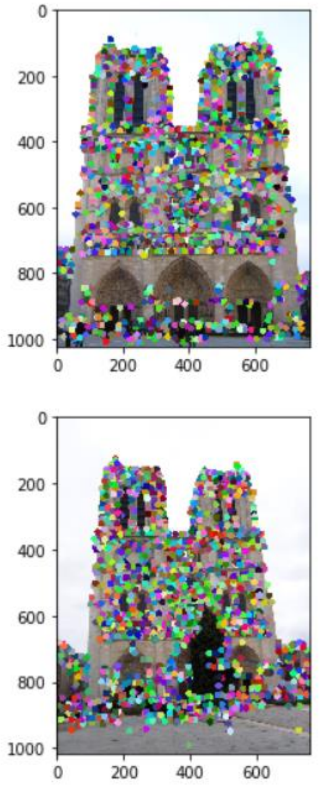
CS 4476 Project 3

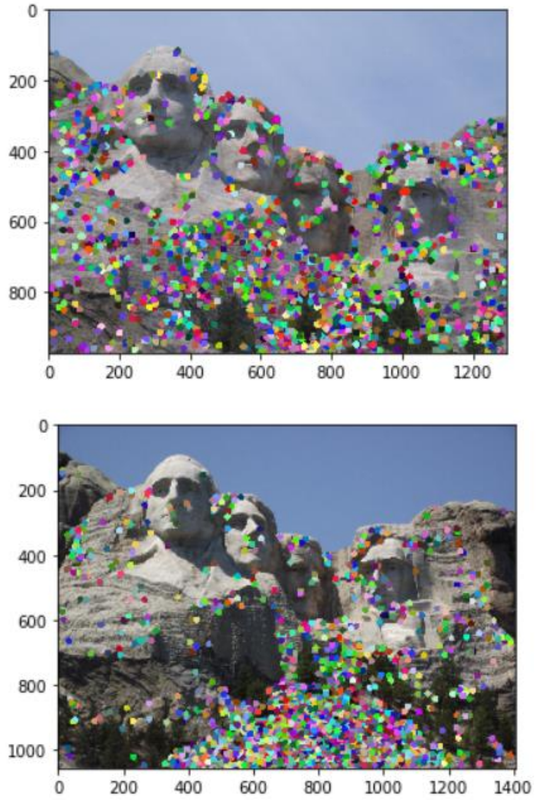
Tarun Pasumarthi
tpasumarthi3@gatech.edu
tpasumarthi3
903085537

Part 1: Harris Corner Detector

```
1500 corners in image 1, 1500 corners in image 2
```

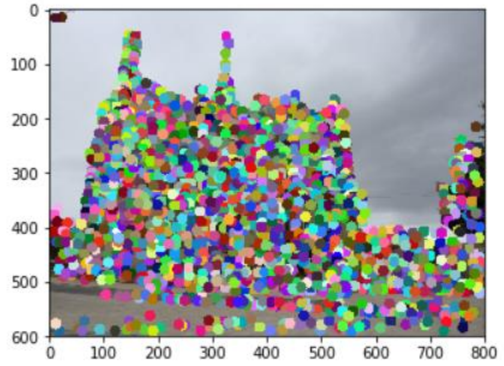


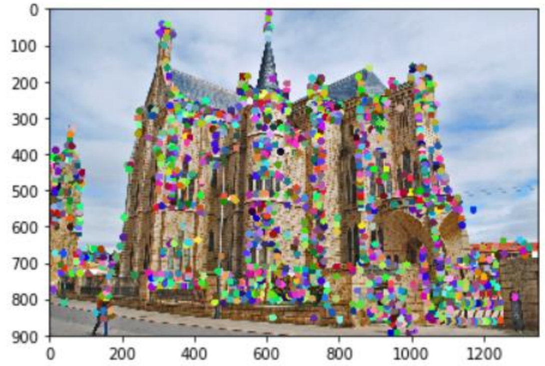
```
1500 corners in image 1, 1500 corners in image 2
```



Part 1: Harris Corner Detector

```
1500 corners in image 1, 1500 corners in image 2
```





Part 1: Harris Corner Detector

<Describe how your implementation mirrors the original harris corner detector process. (First describe Harris) What does each helper function do? How are the operations we perform equivalent and different?)>

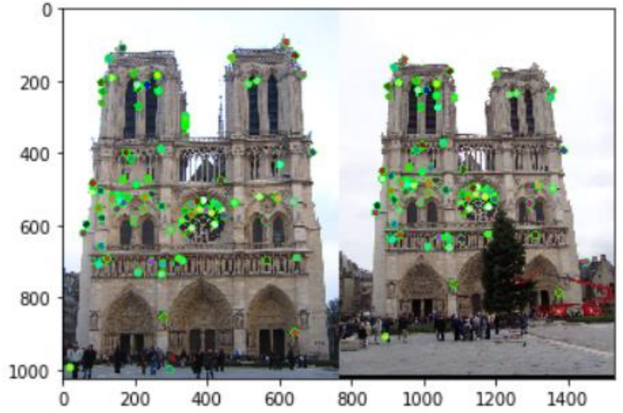
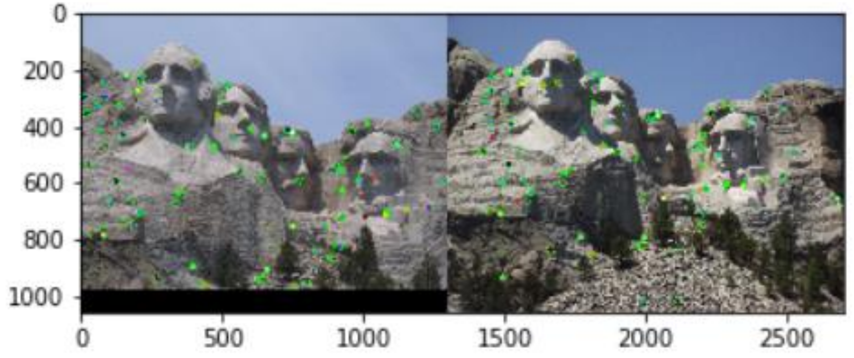
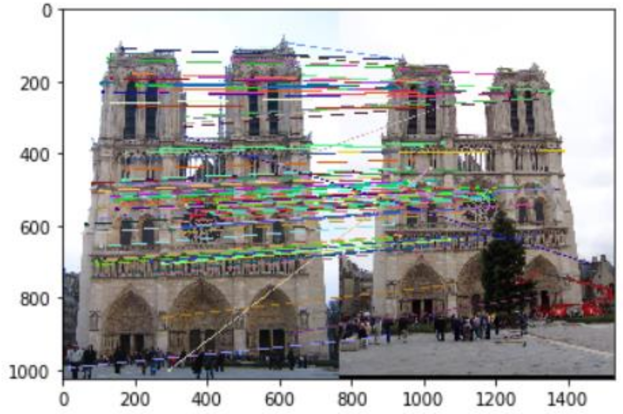
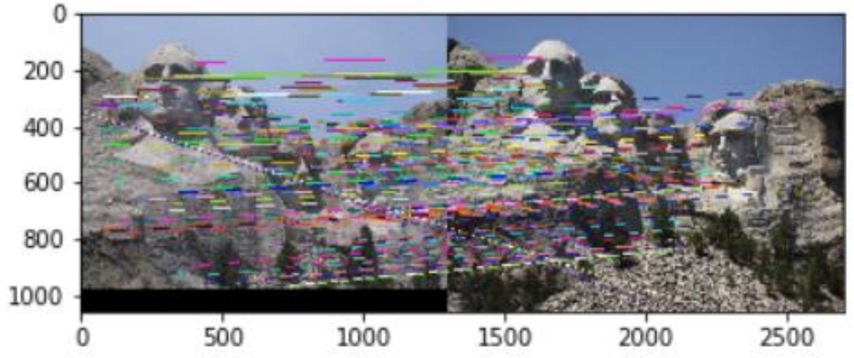
Harris corner detector is used to detect features in an image that are independent of rotation. The way it essentially works is that it computes the gradient in x and y directions at a certain window and uses eigen value decomposition to see if it has a large gradients along all axes. If it does, this would be detected as a corner. In our implementation, we use get_gradients() to compute the gradients in the x and y directions, we use get_second_movements() to get the second movement by convolving the square of the gradients and dx*dy with the gaussian filter, then we uses these values to create a matrix, then using this matrix we calculate the corner response with the respective helper method, then we get only the local maxima using non_max_suppression() , and then we remove the values too close to the borders using remove_border_vals, then we sort these values and return the best n points as features.

Part 2: Sift

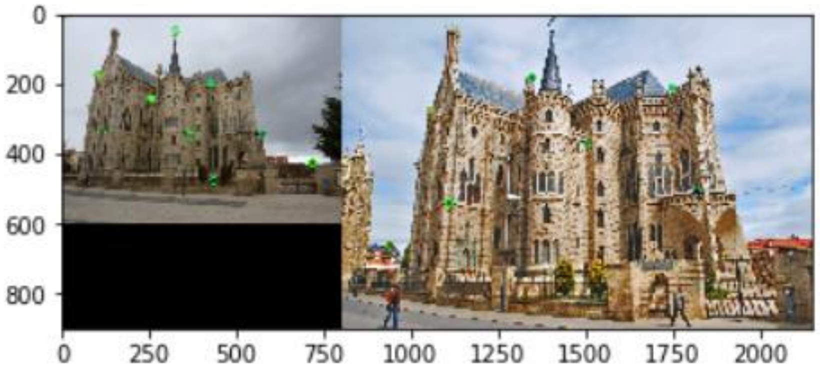
<Describe how your implementation mirrors the Sift Process. (First describe Sift) What does each helper function do? How are the operations we perform equivalent and different?)>

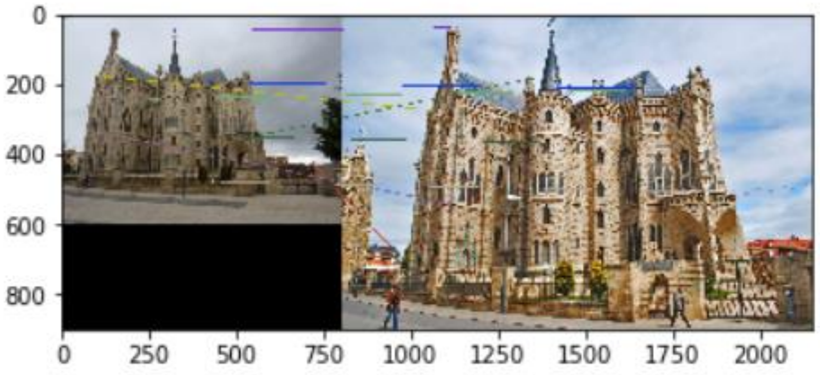
Sift is an algorithm that takes in a list of feature points in an images and computes a feature vector for each point that is distinctly representative of that point and is invariant of scale. To do this, for each point, we create a window around the given point, split the window into 16 bins, compute a histogram of size 8 based on the orientations of the bins weighed by the magnitude of each cell. So no fo each point, we have a feature vector that 4*4*8 = 128 long. The magnitudes and orientation are calculated by the function get_magnitudes_and_orientations() which computes these from the image gradients in the x and y direction. Then get_feat_vec() runs the sift algorithm described above on a single points, and finally, get_feat_vec() aggregates these feature vectors.

Part 3: Feature Matching

Part 3: Feature Matching



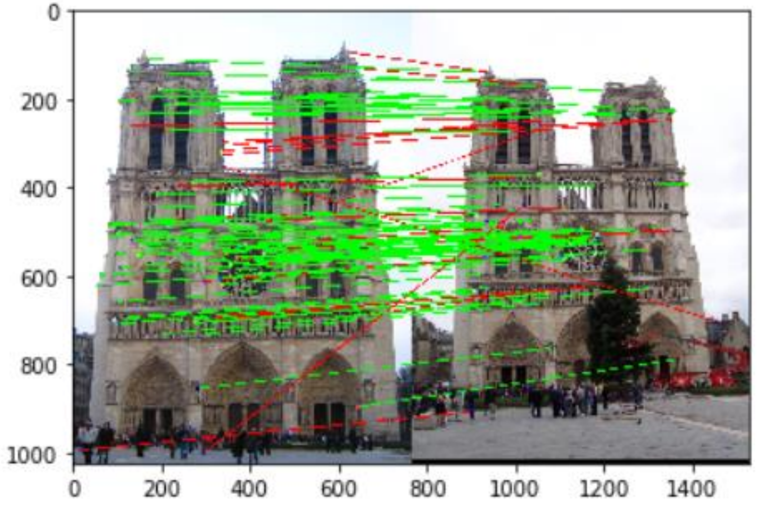


<Describe your implementation of feature matching.>

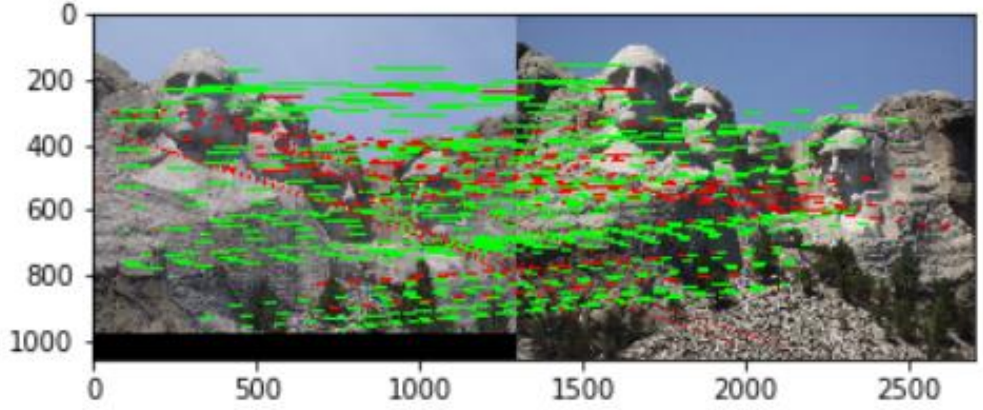
For feature matching, I got the feature vector matrixes of the two images, computed the distance matrix between each vector in the first feature vector matrix to each vector in the second feature vector matrix. Using this matrix, I iterated through each row to find matches that passed the ratio test and added these matches to a return matrix along with the confidences. I sorted the results based on the confidence, which was –distance. Finally I returned this result.

Results: Ground Truth Comparison

```
Accuracy = 0.800000
```

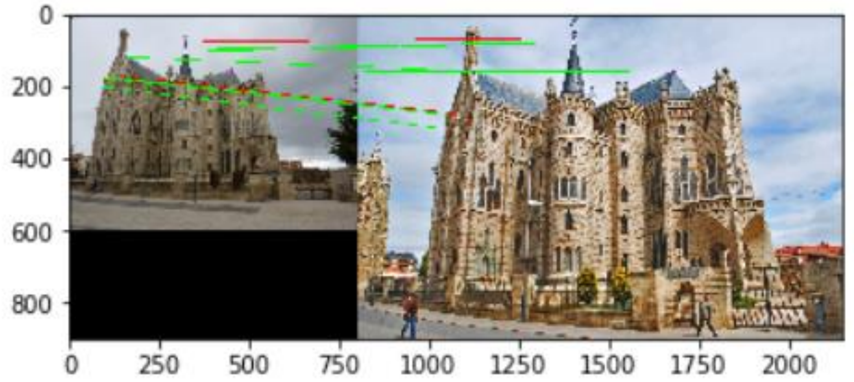


```
Accuracy = 0.770000
```



Results: Ground Truth Comparison

```
You found 9/100 required matches
Accuracy = 0.070000
```



Accuracies for default parameters:

Notre Dam: 80%

Mount Rushmore: 77%

Episcopal Gaudi: 1%

Accuracies for different subgrid sizes:

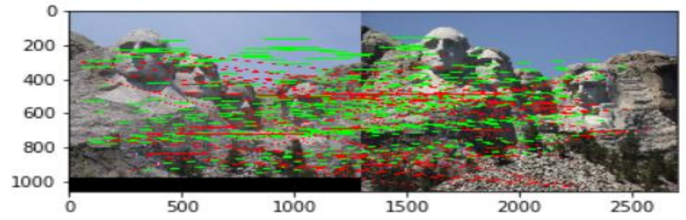
	Notre Dam	Rushmore	Gaudi
2x2	.8	.33	.02
5x5	.87	.88	.01
7x7	.89	.91	.04
15x15	.89	.93	.07

Clearly, as the grid size increases, the accuracy tends to increase.

Extra Credit: Hyperparameter Tuning part 1

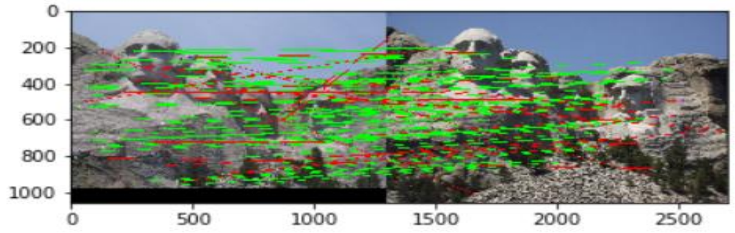
```
You found 100/100 required matches
Accuracy = 0.610000
```

Sigma=3



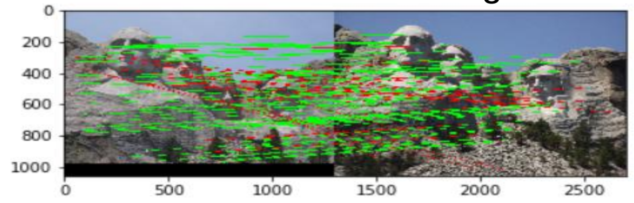
```
You found 100/100 required matches
Accuracy = 0.760000
```

Sigma=6



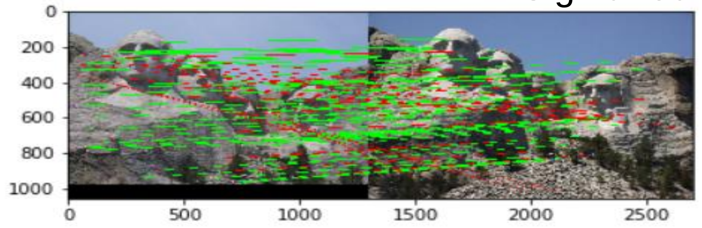
```
You found 99/100 required matches
Accuracy = 0.770000
```

Sigma=10



```
You found 100/100 required matches
Accuracy = 0.750000
```

Sigma=30

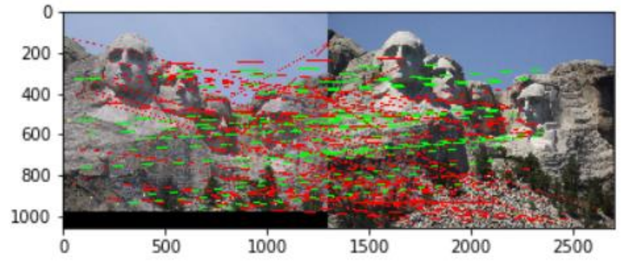


When changing the values for large sigma (>20), why are the accuracies generally the same?
Clearly, the accuracies start to increase as sigma increases then then after a certain point, increasing the sigma, would keep the the accuracies generally the same. This is because the gaussian filter has similar values as sigma gets larger, as they scale much slower.

Extra Credit: Hyperparameter Tuning part 2

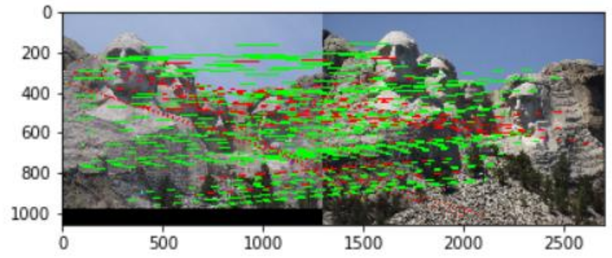
Feature_width= 8

```
You found 87/100 required matches
Accuracy = 0.300000
```



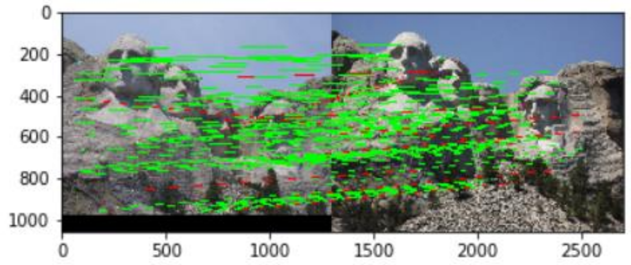
Feature_width= 16

```
You found 99/100 required matches
Accuracy = 0.770000
```



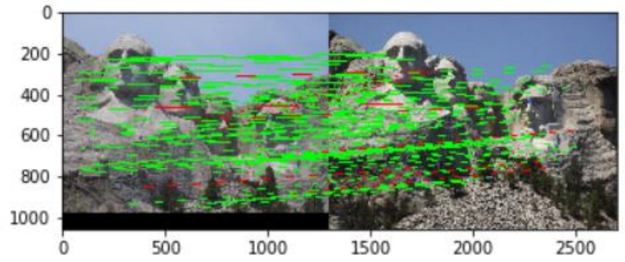
Feature_width= 24

```
You found 100/100 required matches
Accuracy = 0.900000
```



Feature_width= 8

```
You found 100/100 required matches
Accuracy = 0.910000
```



What is the significance of changing the feature width in SIFT?

Clearly, increasing the feature width in sift tend to increase the accuracy.

Extra Credit: Accelerated Matching

<Insert Runtime/Accuracy of your faster matching implementation. What did you try and why is it faster?>