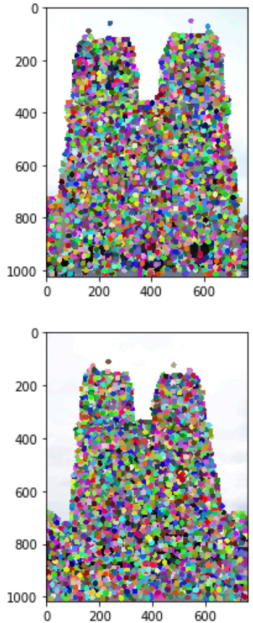
CS 4476 Project 2

Ashwin Rathie
Ashwin.rathie@gatech.edu
arathie6
903281887

Part 1: HarrisNet

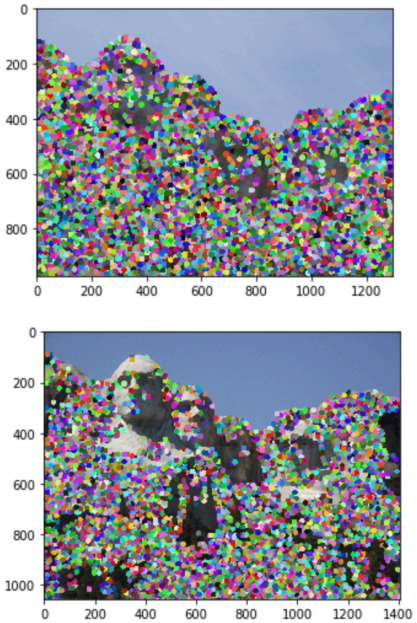
<insert visualization of Notre Dame interest points from proj2.ipynb here>

```
3110 corners in image 1, 2695 corners in image 2
```



< insert visualization of Rushmore interest points from proj2.ipynb here >

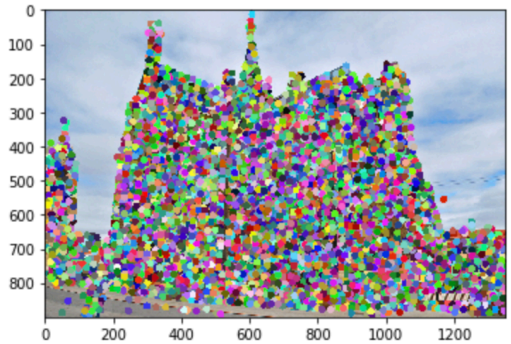
```
4274 corners in image 1, 4500 corners in image 2
```

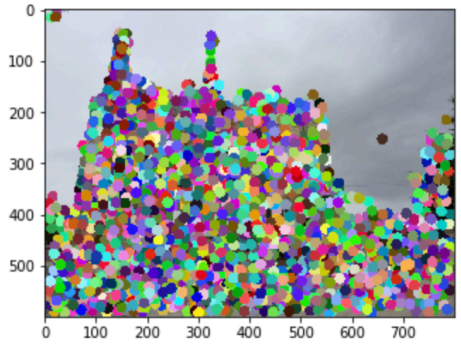


Part 1: HarrisNet

< insert visualization of Gaudi interest points from proj2.ipynb here >

```
4494 corners in image 1, 1743 corners in image 2
```





Part 1: HarrisNet

<Describe how the HarrisNet you implemented mirrors the original harris corner detector process. (First describe Harris) What does each layer do? How are the operations we perform equivalent?)>

The Harris corner detector scans over an image and looks at each ”window” of the image to while computing x and y gradients to detect large variations in intensity. Using this, a ”cornerness score” is assigned for each point and if it above a predetermined threshold, it is said to be a corner. Our HarrisNet is similar in concept, with the only difference being that the operations/steps of the algorithm are broken down into layer. The (1) Image Gradient Layer convolves the image with sobel kernels to produce the gradients. Then, the (2) Channel Product Layer simply computes products of the x and y intensities that will be useful in the next layer. The (3) Second Moment Matrix Layer convolves Ixx, Iyy, and Ixy with a gaussian kernel to compute Sxx, Syy, and Sxy, which make up the second moment matrix. The (4) Cornerness Response Layer computes the cornerness score of each point according to the equation det(A) - alpha * (trace(A)^2), where A is the Second Moment Matrix from the previous layer. Finally, the (5) Non-Maximal Suppression layer essentially simplifies and consolidates the scores by removing the lower half of scores, then MaxPooling the image so that each 7x7 subgrid has to score of the max nearby score.

Part 2: SiftNet

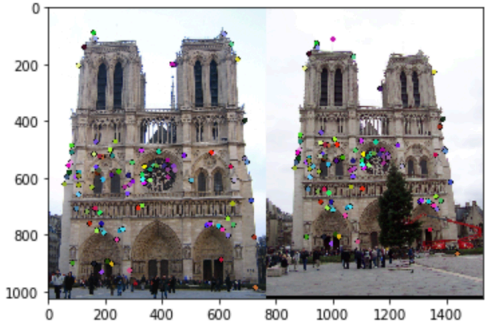
<Describe how the SiftNet you implemented mirrors the Sift Process. (First describe Sift) What does each layer do? How are the operations we perform equivalent?)>

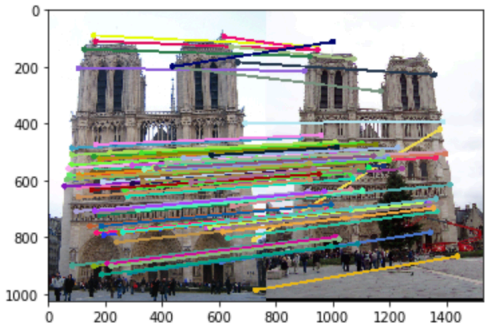
Sift uses “oriented gradients” in order to form a “descriptor” for each keypoint. A SIFT descriptor can be visualized as 8 buckets, each representing a vector going in a certain direction. A vector, in this context, is the vector that describes the direction of the gradient. For each keypoint, SIFT looks at the 16x16 area surrounding the point, and within each 4x4 subgrid of the 16x16 area, it assigns the subgrid the most common vector. It then adds the vector of each subgrid to the appropriate “bucket”.

The SiftNet is similar to the SIFT algorithm but segments the steps using the concept of layers. Image Gradient Layer is the same as what was used in HarrisNet, computing the gradients using x and y sobel filters. The Sift Orientation Layer then determines the orientation of each of the quadrants around each point and uses dot product to combine those tensors into one tensor that describes the overall “orientation” for each point. The Histogram layer then creates the histogram of oriented gradients (with the 8 “buckets”). The subgrid accumulation layer gathers the notable values from each subregion, resulting in a vector with 128 elements (4x4 subgrids x 8 buckets = 128).

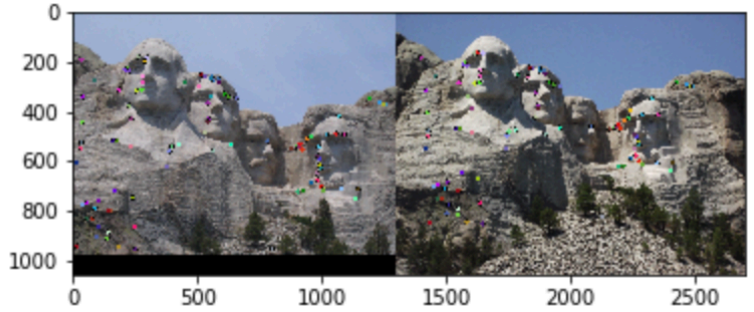
Part 3: Feature Matching

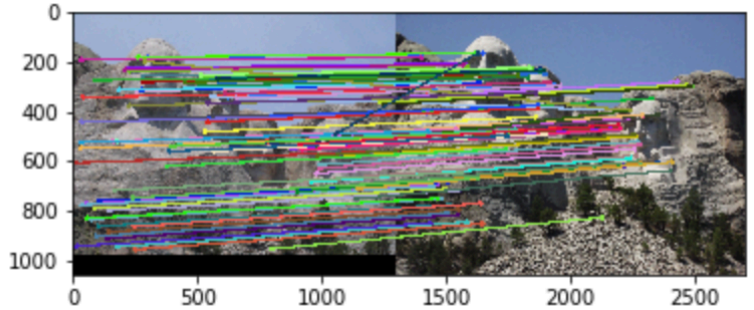
<insert feature matching visualization of Notre Dame from proj2.ipynb>





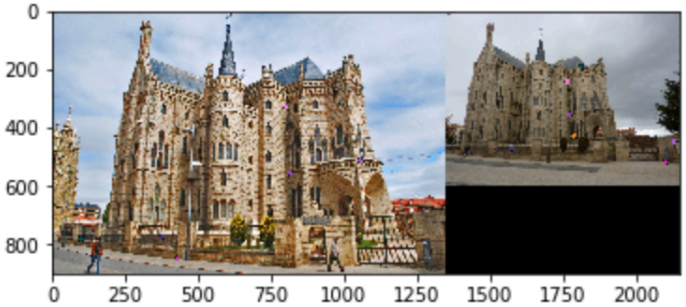
<insert feature matching visualization of Rushmore from proj2.ipynb >

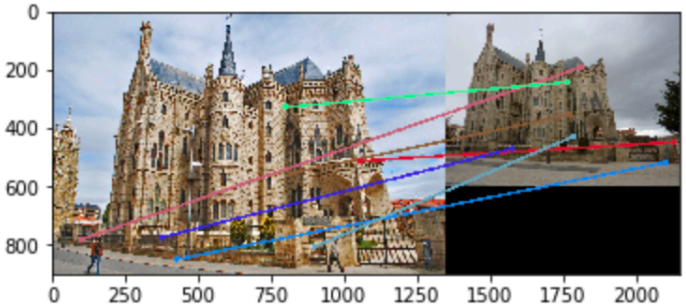




Part 3: Feature Matching

<insert feature matching visualization of Gaudi from proj2.ipynb >



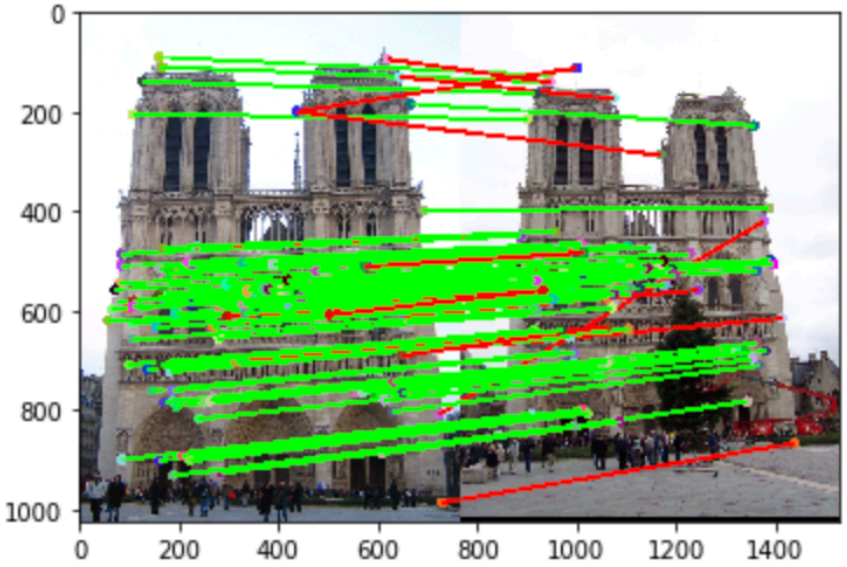


<Describe your implementation of feature matching.>

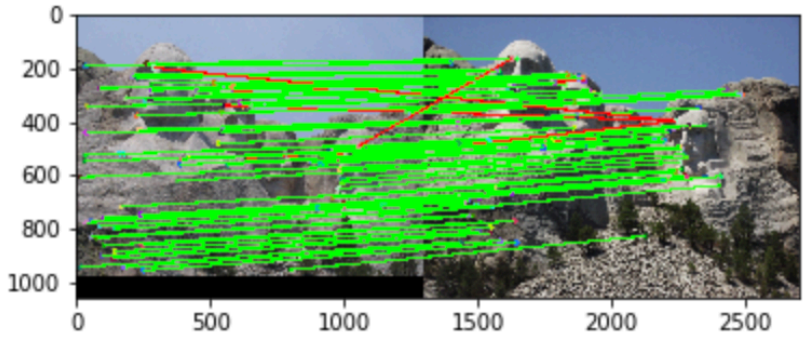
The feature matching was done by calculating the differences in the positions of each feature in each picture to each of the features in the other image. The smallest distance represented a candidate for a match. Using a threshold of 0.75 (chosen because the results were reasonably good), the matches were confirmed or rejected.

Results: Ground Truth Comparison

<Insert visualization of ground truth comparison with Notre Dame from proj2.ipynb here>

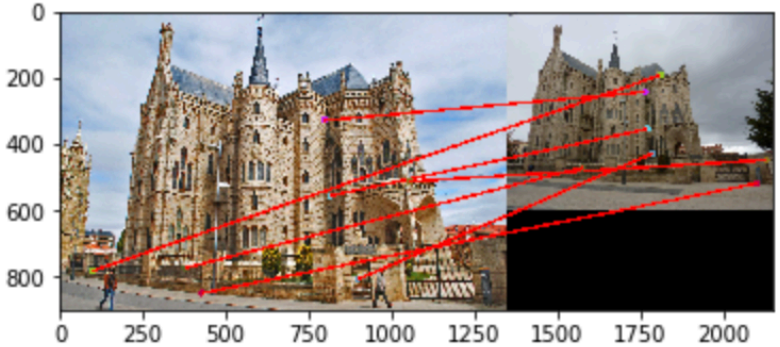


<Insert visualization of ground truth comparison with Rushmore from proj2.ipynb here>



Results: Ground Truth Comparison

<Insert visualization of ground truth comparison with Gaudi from proj2.ipynb here>



<Insert numerical performances on each image pair here. Also discuss what happens when you change the 4x4 subgrid to 2x2, 5x5, 7x7, 15x15 etc?>

Notre Dame: 100/100 matches, 0.85 accuracy

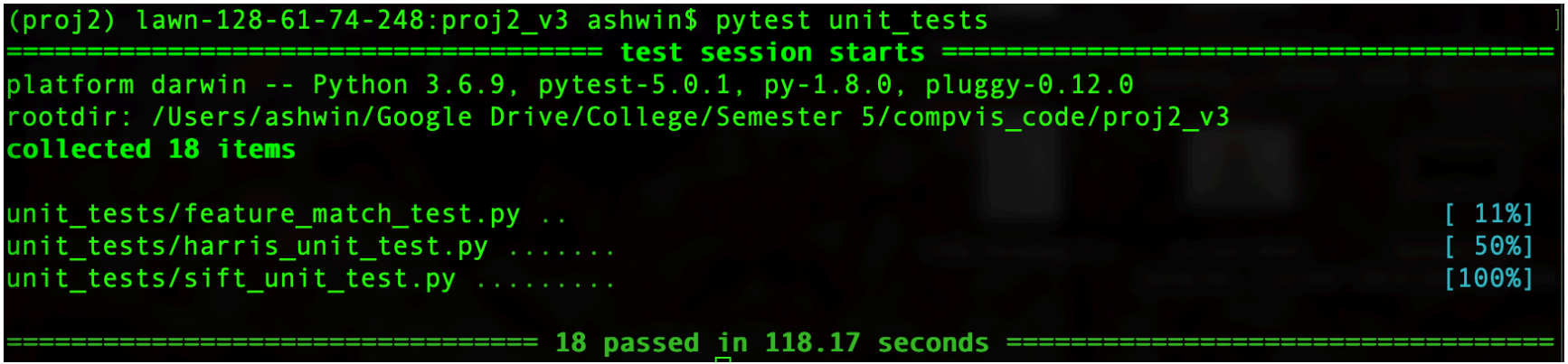
Rushmore: 100/100 matches, 0.93 accuracy

Gaudi: 7/100 matches, 0.00 accuracy

Decreasing the subgrid size seems to increase the amount of features matched.

Tests

<Provide a screenshot of the results when you run `pytest unit_tests` on your final code implementation (note: we will re-run these tests).>



Conclusions

<Describe what you have learned in this project. Feel free to include any challenges you ran into.>

The most interesting thing that I learned from this project was, honestly, not how Harris and SIFT worked. While those are very clever and it was great to see how the math translates to results, understanding the layer structure for PyTorch and networks in general was eye-opening. It had always been daunting, but still something I very much wanted to learn and through this project, I have a much better (even if not complete) understanding of it. Aside from that though, I was perplexed when Gaudi was performing so poorly on feature, but then remembered that this was likely because the images were of completely different scales, and we hadn’t included the ”scale invariance” part of SIFT.