CS 6476 Project 3

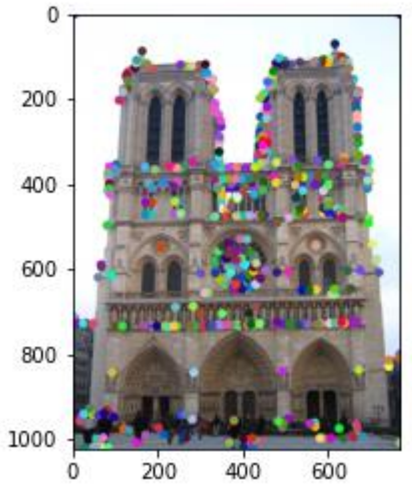
Karan Sarkar
ksarkar9
903569592

Gradescope Group Quiz Collaboration Photo

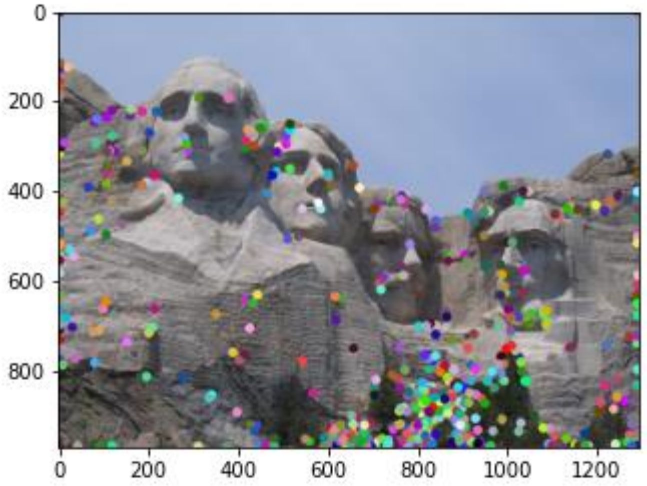
<Insert a picture showing that you and your group met to discuss the quiz and basic concepts of the project>

Part 1: HarrisNet

<insert visualization of Notre Dame interest points from proj3.ipynb here>

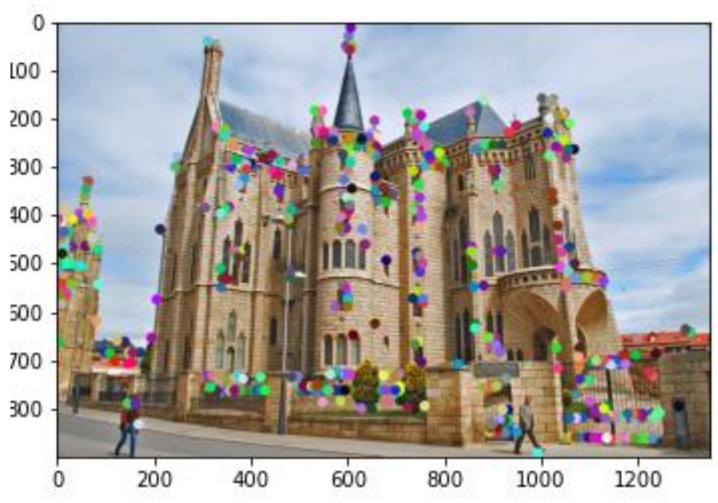


< insert visualization of Rushmore interest points from proj3.ipynb here >



Part 1: HarrisNet

< insert visualization of Gaudi interest points from proj3.ipynb here >



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?)>

In HarrisNet, first the second moment matrix is computed. The eigenvalues represent the maximal and minimal amounts of change in a direction. We compute the corner score based on the eigenvalues. Second, the second moment matrix is used to compute the corner score. Third, we eliminate points with low corner score. Lastly, we select points that are at local maxima.

We implemented this using Pytorch convolutional layers. ChannelProductLayer essentially takes an outer product to compute the second moment matrix component values. In SecondMomentMatrixLayer, We used a convolutional Gaussian layer to compute the second moment matrix. This performs a weighted sum of values from the previous layer . CornerResponseLayer computes the corner score from the Second Moment Matrix. In NMSLayer, We used a max pooling layer to identify the local maxima of corner score. We then returned all the values that were equal to a local maxima.

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 features are formed by computing the gradient at each pixel in a 16 x 16 window around the keypoint. In the actual version of SIFT, the gradient magnitudes are weighted by a gaussian. In each 4 by 4 quadrant, we form a histogram representing the contribution of the gradient to each of 8 principal directions. This forms an 8 vector for each quadrant. This 8 vectors are then concatenated and normalized.

In SIFTOrientationLayer, we use a 1 by 1 convolutional layer to compute the contribution of the gradient in each of the 8 principal directions. We also add a 2 more output channels to return the original gradient, In the Histogram layer, we create a component that can be convolved to create a histogram. We return a modified version of the contribution tensor. But we report a contribution of zero for non maximal principal directions and the total magnitude on maximal principal directions. Unlike the original sift, each gradient only contributes to one direction. SubGridAccumulationLayer uses a convultional layer to find the total histogram in a 4 by 4 area.

Part 2: SiftNet

- <Explain what we would have to do make our version of Sift rotationally invariant (conceptually)>

You can the dominant direction of a keypoint. Then use an oriented patch around the detected point to form a feature.

- <Explain what we would have to do to make our version of SIFT scale invariant (conceptually)>

You can extract features at a variety of resolutions concatenate them all to create one larger feature.

Part 2: SiftNet

- <What would happen if instead of using 16 subgrids, we only used 4 (dividing the window into 4 grids total for our descriptor)>

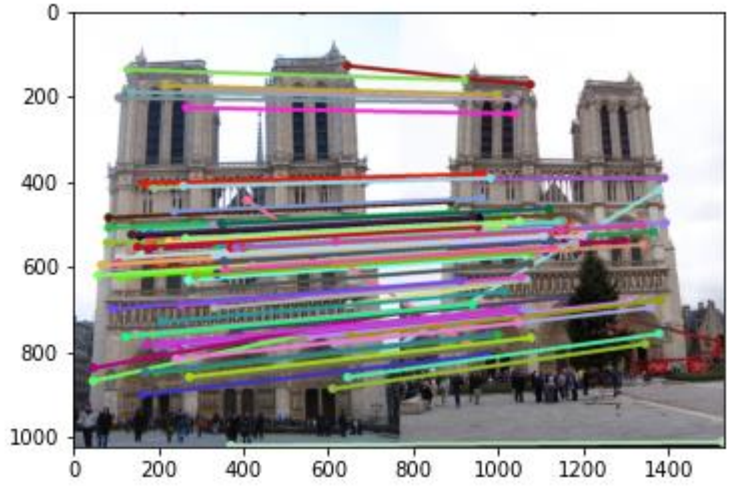
- The feature would be less powerful because it picks up less information from the surrounding pixels. The range of the descriptor would be less so you might run into issues on larger pictures with similar features, You would have more error.

- <What could we do to make our histograms in this project more descriptive?>

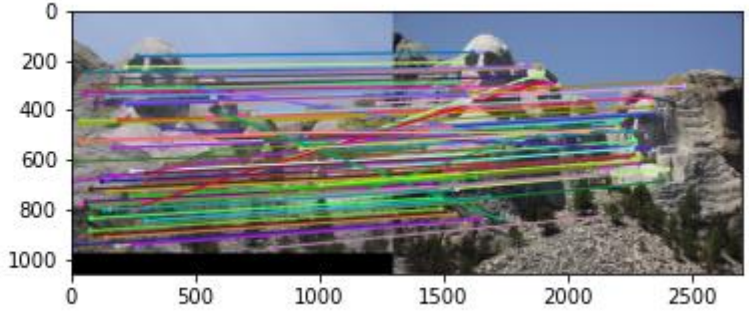
You could add more principal direction bins,

Part 3: Feature Matching

<insert feature matching visualization of Notre Dame from proj3.ipynb>

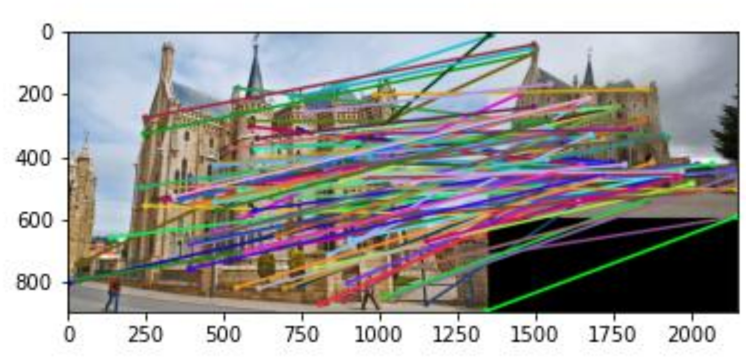


<insert feature matching visualization of Rushmore from proj3.ipynb >



Part 3: Feature Matching

<insert feature matching visualization of Gaudi from proj3.ipynb >

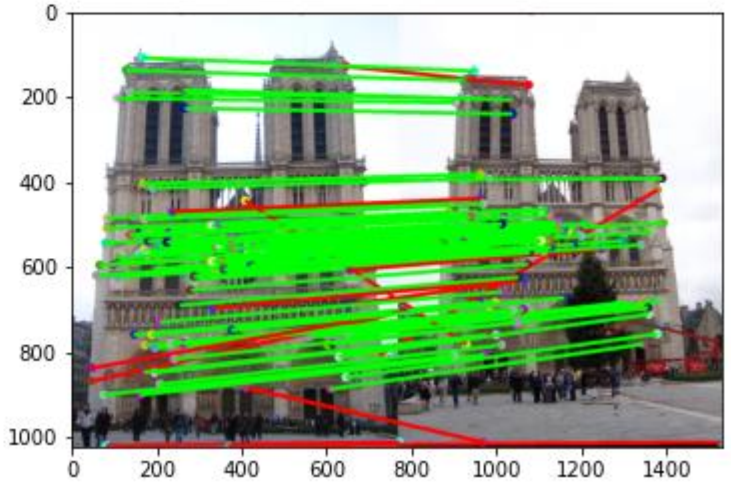


<Describe your implementation of feature matching.>

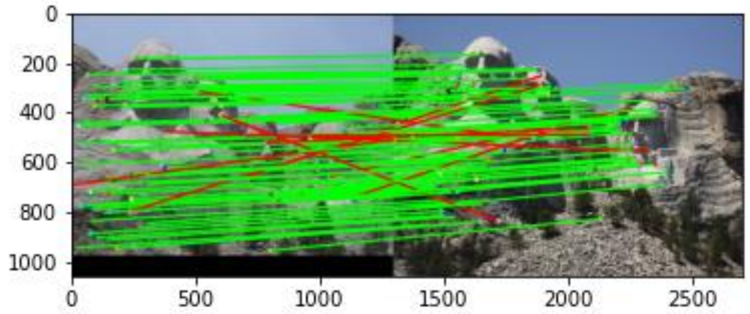
First I matched each keypoint in image1 to the closest key point in image 2. Then, I eliminated all the pairs whose NN1/NN2 ratio was to high. Then I sorted the remaining points by their NN1 distance and returned the pairs.

Results: Ground Truth Comparison

<Insert visualization of ground truth comparison with Notre Dame from proj3.ipynb here>

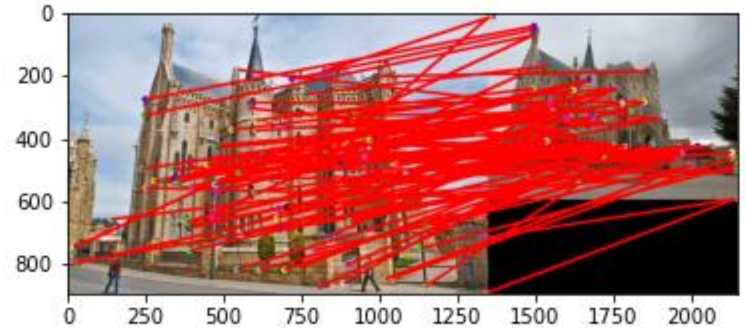


<Insert visualization of ground truth comparison with Rushmore from proj3.ipynb here>



Results: Ground Truth Comparison

<Insert visualization of ground truth comparison with Gaudi from proj3.ipynb here>



<Insert numerical performances on each image pair here.>

Notre Dame 0.86
Rushmore 0.88
Gaudi 0

Results: Discussion

- <Discuss the results. Why is the performance on some of the image pairs much better than the others?>

The Gaudi image had worse performance. This was probably because it had more significantly different lighting between images.

- <What sort of things could be done to improve performance on the Gaudi image pair?>

You could normalize the pixels in each image.

Conclusions

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

I had a lot of trouble debugging the project. One troublesome was the appearance of nans in the features. I wrote code to filter out the nans. But it was buggy and instead lead to bad matching accuracy.

Extra Credit: Sift Parameter variations

I tried changing the normalization. Raising the vectors to 0.7 instead of 0.9 resulted in 0.84 accuracy as opposed to 0.86. Increasing it 0.95 instead of 0.9 resulted in 0.83. 0.9 does seem to be something of a sweet spot.

Normalizing by dividing by the sum rather than the Euclidean norm lowered accuracy to 0.84.

Extra Credit: Custom Image Pairs