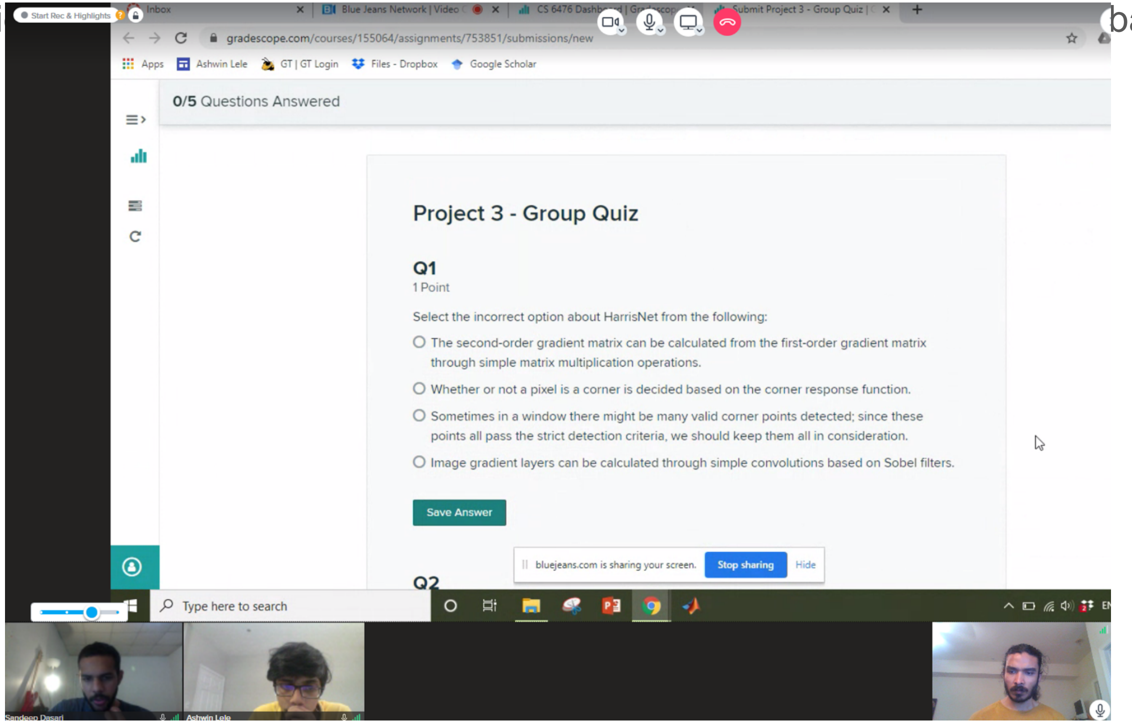
CS 6476 Project 3

Sai Sandeep Dasari
sdasari38
903540744

Gradescope Group Quiz Collaboration Photo

<Insert a pi basic concepts of the project>


Start Rec & Highlights
Inbox
Blue Jeans Network | Video
CS 6476 Dashboard | Gradescope
Submit Project 3 - Group Quiz
gradescope.com/courses/155064/assignments/753851/submissions/new
Apps
Ashwin Lele
GT | GT Login
Files - Dropbox
Google Scholar
bluejeans.com is sharing your screen.
Stop sharing
Hide
Type here to search
Sandeep Dasari
Ashwin Lele

0/5 Questions Answered

Project 3 - Group Quiz

Q1
1 Point

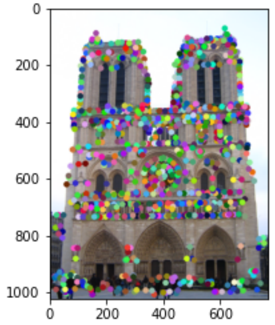
Select the incorrect option about HarrisNet from the following:

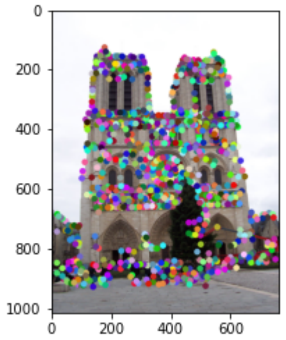
- ○ The second-order gradient matrix can be calculated from the first-order gradient matrix through simple matrix multiplication operations.
- ○ Whether or not a pixel is a corner is decided based on the corner response function.
- ○ Sometimes in a window there might be many valid corner points detected; since these points all pass the strict detection criteria, we should keep them all in consideration.
- ○ Image gradient layers can be calculated through simple convolutions based on Sobel filters.

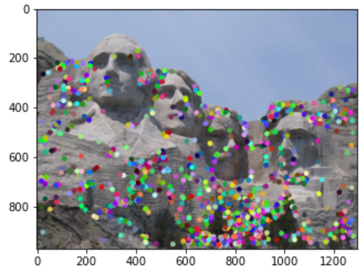
Save Answer

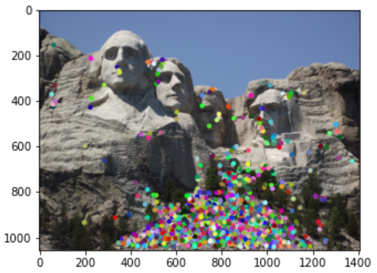
Q2

Part 1: HarrisNet

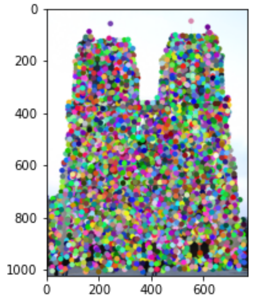


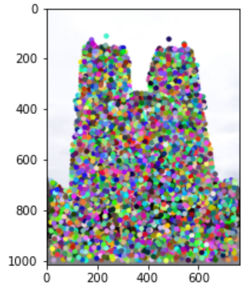


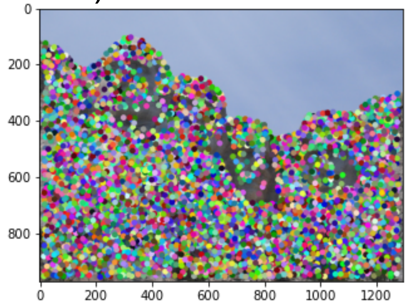


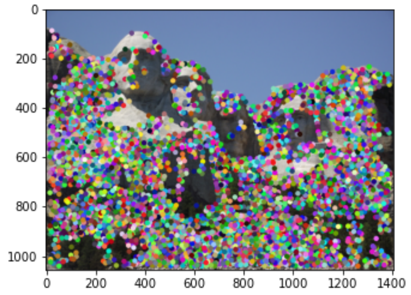


1000 (above) vs 4500 (below)





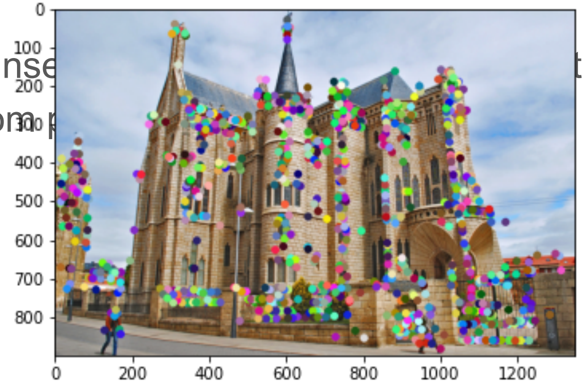


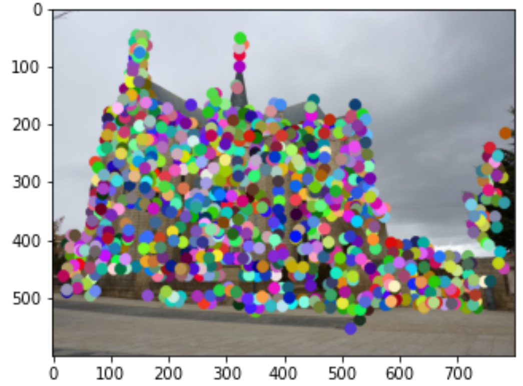


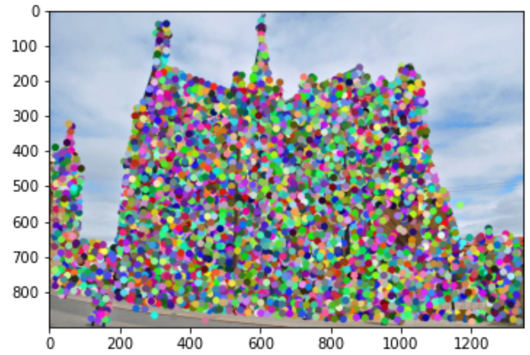
Part 1: HarrisNet

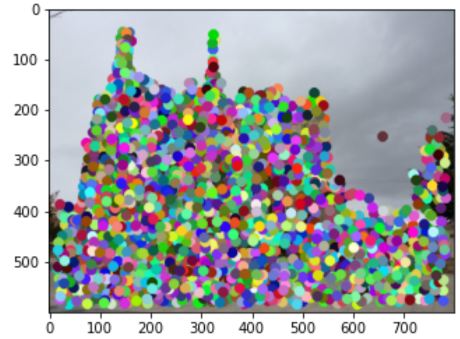
< inse t points

from p









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?)>

Harris corner detection is an algorithm for detecting corners in an image which can be later used for a wide variety of tasks invariant to rotation. The idea is to detect corners in an image by shifting a small window over the image and observing the change in intensity of the image.The HarrisNet we implemented mirrors the original detector largely in terms of what the layers of our network do:

1. ImageGradientsLayer : calculate image gradients in each direction (X and Y), just like the original implementation

2. ChannelProductLayer: In Szeliski 4.1.1. Algorithm 4.1, this step computes the outer products of the above gradients. We compute product between channels of the previous layer: Ixx, Iyy and Ixy.

3. SecondMomentMatrixLayer: compute Second Moment Matrix. We convolve the above 3 images (3 channels) with a Gaussian too

4. CornerResponseLayer: computes the R cornerness matrix, the R cornernerss matrix is the equivalent “scalar interest metric” in the original

5. NMSLayer : performs non-maxima suppression to keep only the strongest corners in local regions. Finally the original algorithm simply finds local maxima above a certain threshold. We do the same with our layer.

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?)>

Scalar Invariant Feature Transform (SIFT) is a feature descriptor for calculating low dimensional vectors in an image that are scale invariant. In our implementation

We do not perform the sub-octave difference in Gaussian layers in our implementation. Instead we use the keypoints we found earlier from HarrisNet.

We extracted gradient information across the entire image along each orientation direction (Lowe used 36bins, in our case 8).
Then we use the orientations to create weighted histograms over the entire image. This step mirrors the original process.

Once we find the 8 x 16 = 128 SIFT feature vector, we extra feature vectors for our Harris keypoints that accumulate histograms from local regions.

Our project mirrors the classic pipeline but uses the Neural Network architecture in Pytorch to realize the pipeline for instance-level matcing.

Part 2: SiftNet

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

- We can add further rotational invariance to our version by ensuring smoothness between the histogram gradients in each pixel. This can be done by weighting the histograms with magnitude + Gaussian weight on the certain point of the each grid when we calculate the 16 x 8 feature vector. Also by increasing the bin-count from 8 to 16 or 32 orientations will further help our cause.

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

- Lowe proposed sub-octave difference of gaussian filters is done on multiple scales of an image (like a pyramid) to detect the features. This would give our model a scale invariancy. We can also enhance Harris points by computing the local Hessian and rejecting the keypoints greater than a threshold.

Part 2: SiftNet

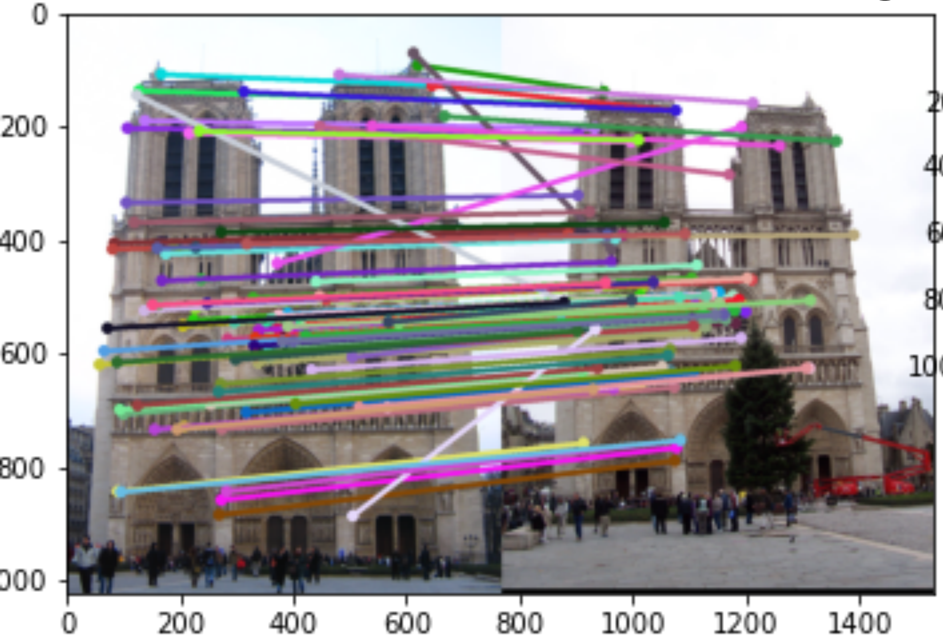
- <What would happen if instead of using 16 subgrids, we only used 4 (dividing the window into 4 grids total for our descriptor)>

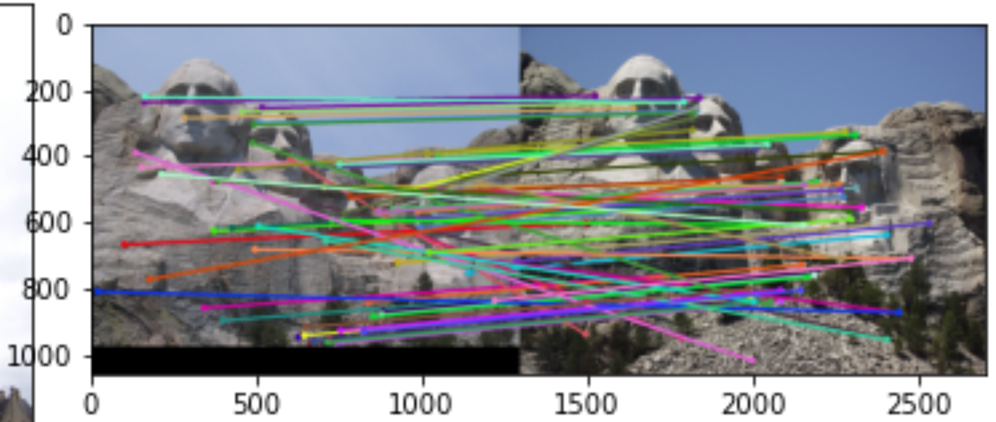
- This will make it difficult for us to detect broad features effectively as the size of the window is greatly reduced. There will be mismatches (compared to ground-truth) in the final matching as the smaller local regions in various portions of the same image are matched (all kinds of circular wheels are easily matched). We ideally want to avoid features that are repeateable in a neighborhood of the image.

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

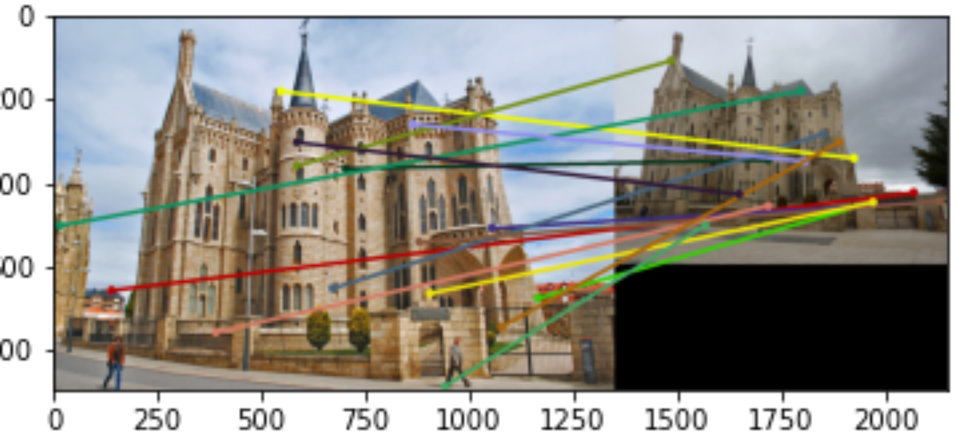
- We can add more orientations (angles) in the histogram that will improve the feature vector by adding a few more dimensions. Also the histograms can be weighted by a gaussian, clamped at certain thresholds (Lowe uses 0.2) and a lot of similar hand tuning. This is something that can be learnt in an end-to-end pipeline ideally.

Part 3: Feature Matching





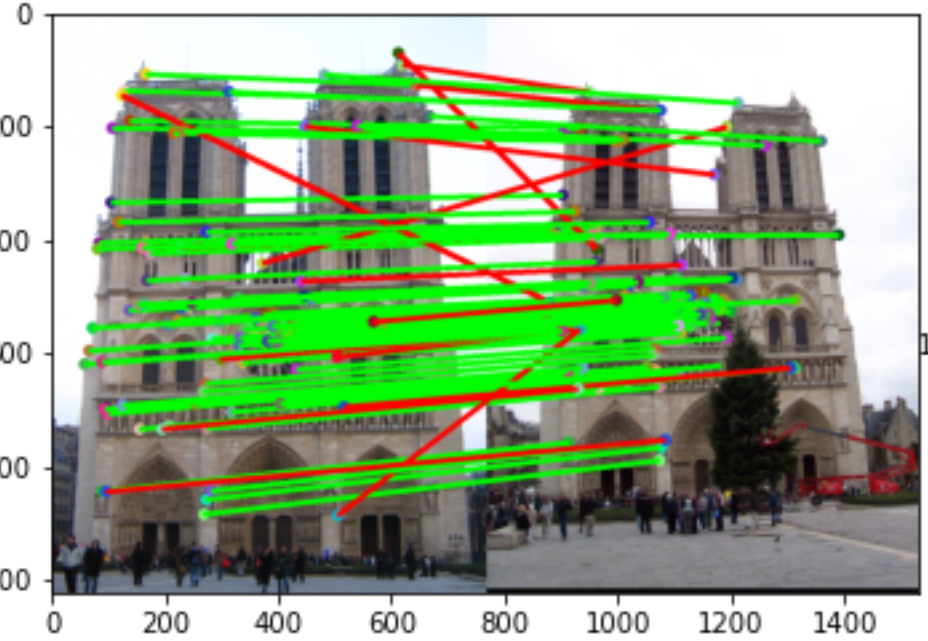
Part 3: Feature Matching

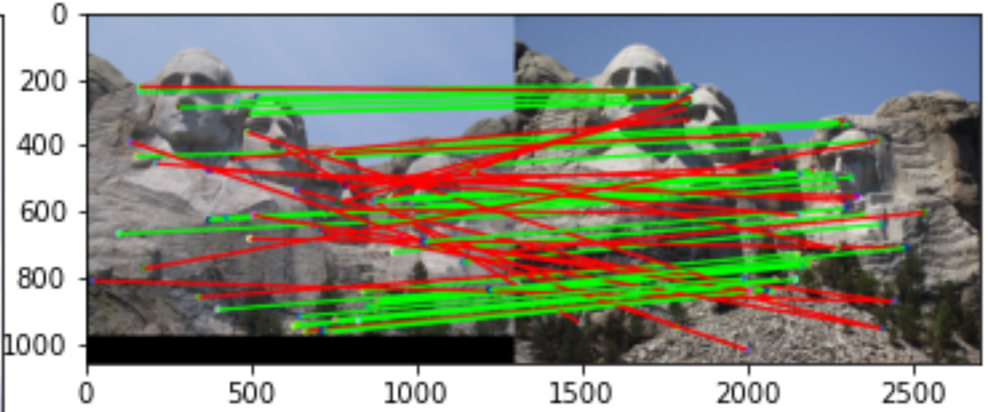


<Describe your implementation of feature matching.>

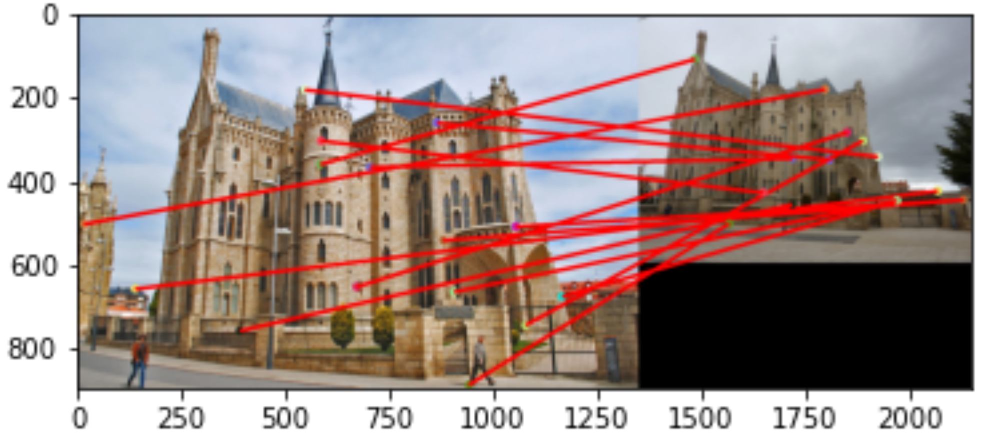
In feature matching, we initially used our keypoints detected from the HarrisNet and discarded some points. We did this by finding a distance vector between the 2 images and using the ratio test with a threshold of 0.8. My confidence metric was NN2/NN1. Thus, a good number of strong matches were found. The resulting matches were then sorted by in descending order of their confidences and returned. (top 100 were used in above images)
 Notredame – 0.92
Rushmore – 0.95 in accuracy.

Results: Ground Truth Comparison





Results: Ground Truth Comparison



<Insert numerical performances on each image pair here.>

Notredame: 100/100 required matches
Accuracy: 0.82

Rushmore: 69/100 required matches
Accuracy: 0.42

Gaudi:

Results: Discussion

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

- The model did well on Notredame and Rushmore because the 2 images A and B have almost the same amount of exposure where as Gaudi’s A and B differed largely in terms of brightness and contrast. Also the Gaudi image B is a smaller resolution image thus the model struggles to find more interest points below the ratio threshold. For Gaudi the model could find only 71 interest points below the ratio compared to ~300 and ~250 interest points for Notredame and Rushmore

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

- Firstly the resolution of both images needs to brought a uniform scale before running Harris Corner. Also, some pre-processing to bring the color brightness and contrast as close as possible will help define the texture of the surfaces and consequently find strong matches with our existing version of the model

Conclusions

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

This project is a great intro into feature descriptors and matching. This project also allows to clearly understand the progression between Harris corners and SIFT descriptors. The project also helped establish the use of neural networks by using PyTorch’s neural network. It was particularly difficult to debug the values flowing through the network even though the weights are defined by us.

The application of deep learning to solve this problem is quite clear and evident from this project.

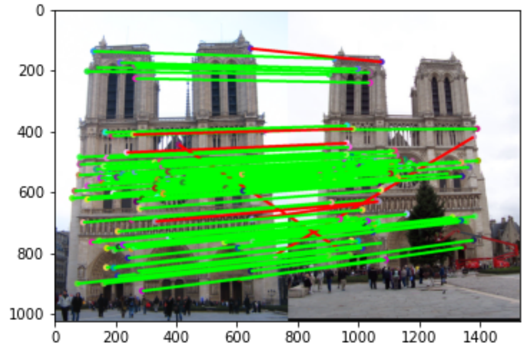
Extra Credit: Sift Parameter variations

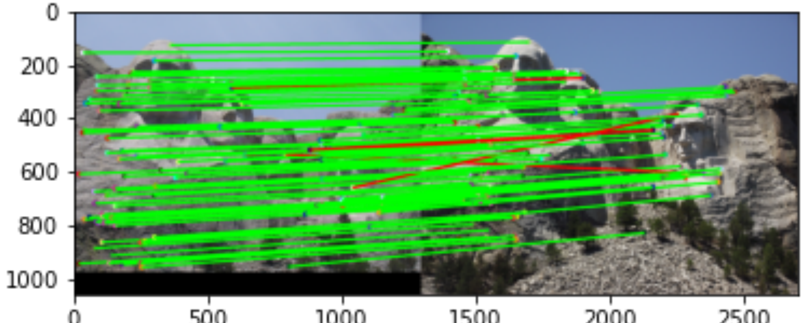
- By changing border val window in HarrisNet from 16x16 to 32x32, I got better accuracy results on Notredame (0.82) and Rushmore (0.42). The accuracy on Rushmore is boosted heavily.
- From 0.42 to 0.95 with this trick
- Performance on Gaudi is still bad

Notredame : 100/100 required matches
Accuracy = 0.92

Rushmore: 100/100 required matches
Accuracy = 0.95

Gaudi: 71/100 required matches
Accuracy = 0.000000





Extra Credit: Custom Image Pairs