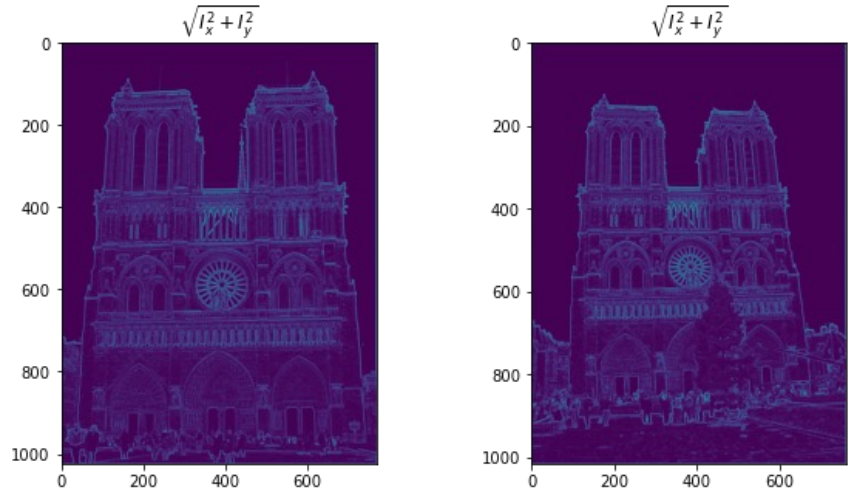
CS 5330 Programming Assignment 2

Abhijit Kaluri
Kaluri.a@northeastern.edu
Kaluri.a
002853560

Part 1: Harris corner detector

[insert visualization of \sqrt(I_x^2 + I_y^2) for Notre Dame image pair from pa2.ipynb here]

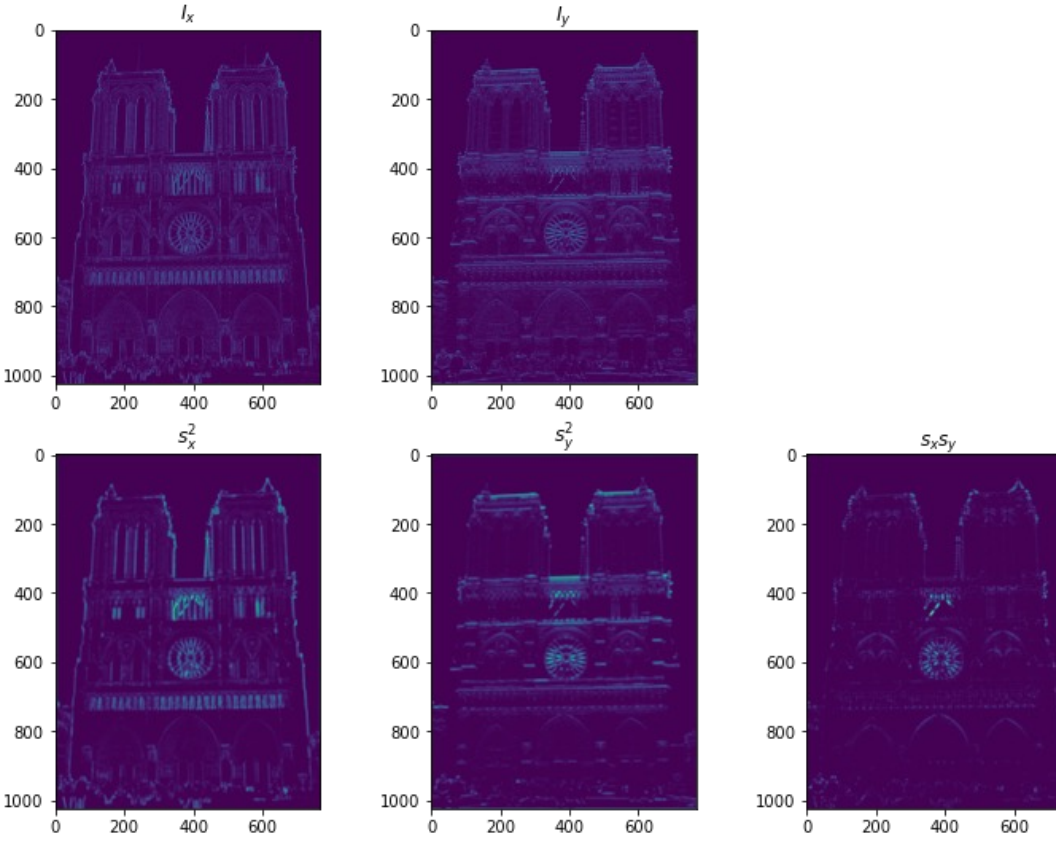


[Which areas have highest magnitude? Why?]

Highest magnitude areas: the brightest spots in the gradient visualization is along the edges of the cathedral's towers, the outlines of windows and arches, and where detailed architectural features stand out against plainer backgrounds. Basically, anywhere there's a sharp change from light to dark or vice versa.
Reason is rapid change in pixel intensity. The gradient measures how quickly pixel values are changing, and it spikes where you have a sudden shift - like where a dark stone edge meets the bright sky. These spots have the steepest "slope" in terms of pixel values, so they get the highest gradient magnitude.

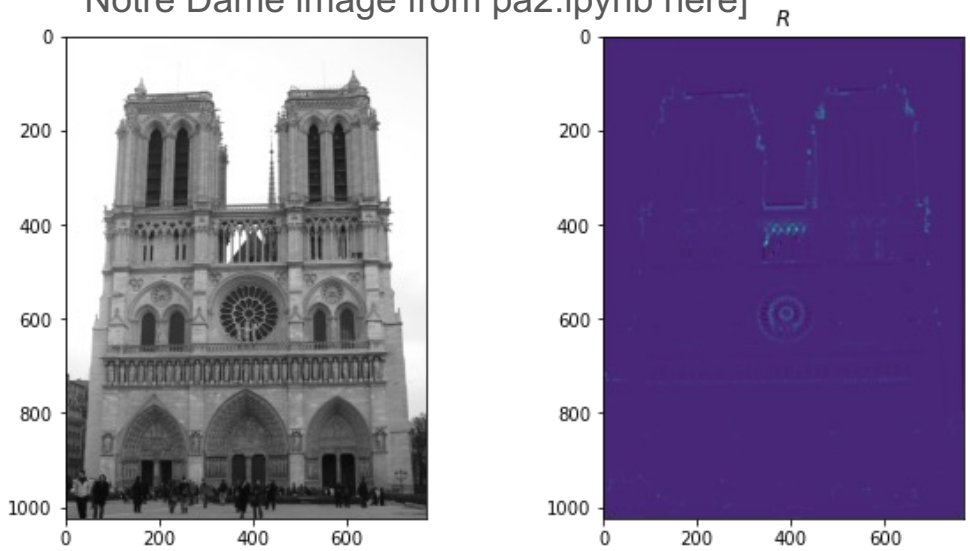
Part 1: Harris corner detector

[insert visualization of I_x, I_y, s_x^2, s_y^2, $s_x s_y$ for Notre Dame image pair from pa2.ipynb here]



Part 1: Harris corner detector

[insert visualization of corner response map of Notre Dame image from pa2.ipynb here]



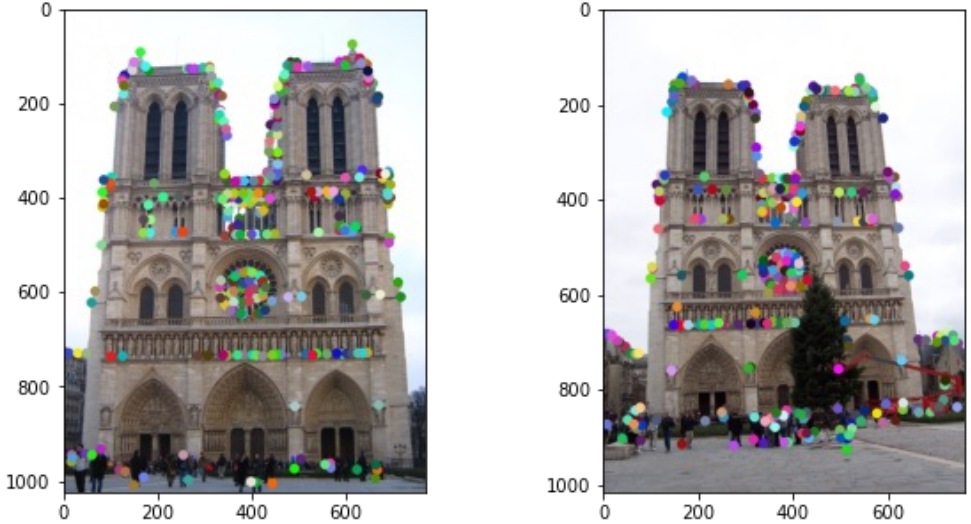
[Are gradient features invariant to both additive shifts (brightness) and multiplicative gain (contrast)? Why or why not? See Szeliski Figure 3.2]

Gradient features respond differently to brightness and contrast changes in images. When it comes to (additive shifts), gradients are completely invariant. This is because adding a constant value to all pixels doesn't change the differences between neighboring pixels, which is what gradients measure.

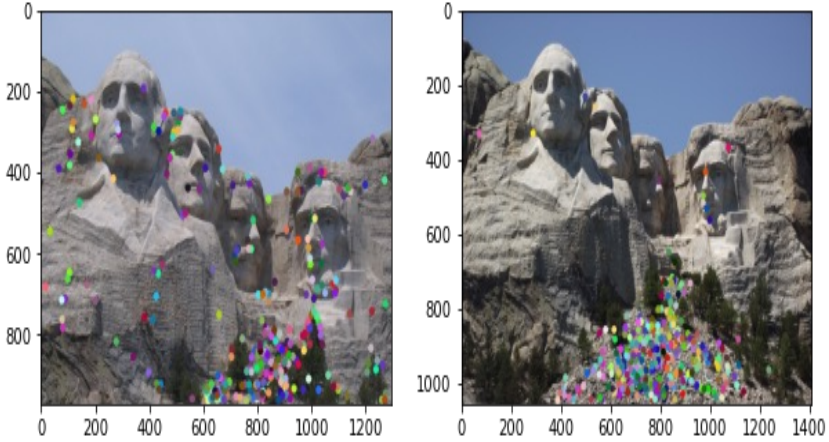
If an image is brighter or darker overall, the gradients remain the same. However, contrast changes affect gradients partially. While the orientations of gradients stay constant when you adjust contrast, their magnitudes scale up or down. This means that the locations/orientations of important features like corners don't move, but they might become more or less pronounced.

Part 1: Harris corner detector

[insert visualization of Notre Dame interest points from pa2.ipynb here]

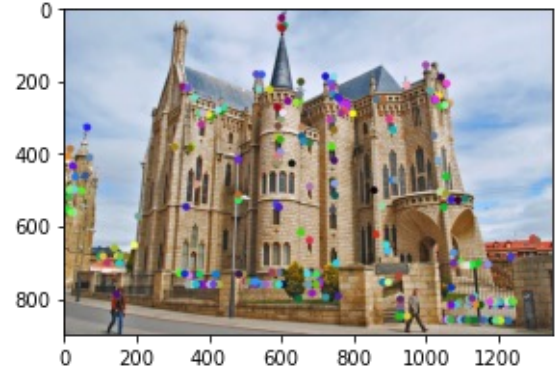


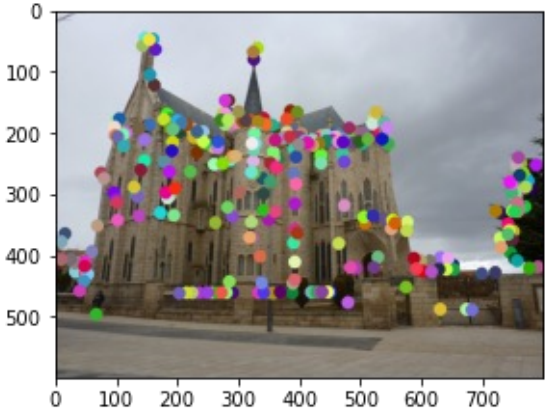
[insert visualization of Mt. Rushmore interest points from pa2.ipynb here]



Part 1: Harris corner detector

[insert visualization of Gaudi interest points from pa2.ipynb here]





[What are the advantages and disadvantages of using maxpooling for non-maximum suppression (NMS)?]

- It is efficient and easy to implement, as it quickly selects the maximum value within a region, making it effective for identifying local maxima like corners or edges. Only consistent feature selection is done, as it keeps only the strongest feature in regions with similar values, reducing redundancy.

  Disadvantages: Maxpooling may result in a loss of precision by suppressing nearby important points, especially when strong responses are clustered together, potentially missing key features. The use of a fixed window size can also be limiting, as it may overlook fine details or fail to suppress weak features effectively. Edge artifacts can arise due to padding near the image edges.
- Maxpooling is non-adaptive and may suppress subtle yet important features that are located near dominant ones.

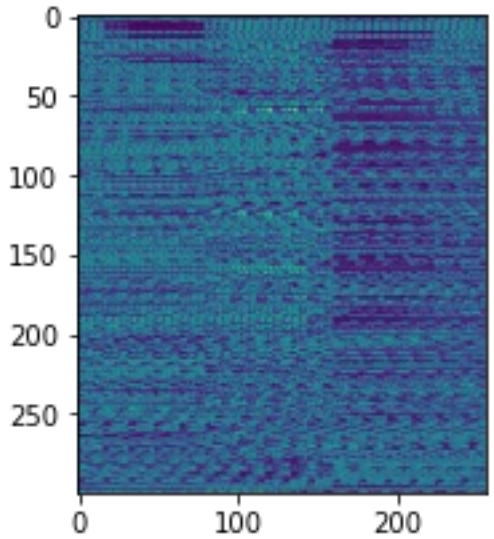
Part 1: Harris corner detector

[What is your intuition behind what makes the Harris corner detector effective?]

I feel it is effective because it identifies points in an image where the intensity changes significantly in multiple directions. These corners, are typically where important features like edges intersect. The detector analyzes local gradients and computes a response based on the second moment matrix, which measures how much the intensity varies in different directions. This makes it robust to small shifts, noise, and changes in lighting. The detector's ability to find corners, which are invariant under translation and rotation, makes it ideal

Part 2: Normalized patch feature descriptor

[insert visualization of normalized patch descriptor from pa2.ipynb here]

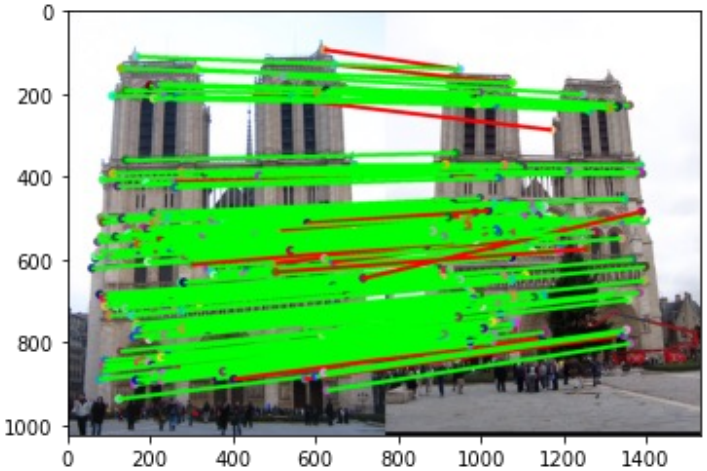


[Why aren't normalized patches a very good descriptor?]

because they are sensitive to changes in illumination, scale, and rotation. While normalizing patches makes them invariant to overall brightness, they still fail to handle significant variations in lighting conditions, contrast, or perspective changes. They are highly affected by image noise and small misalignments, which can alter the pixel intensities in the patch.

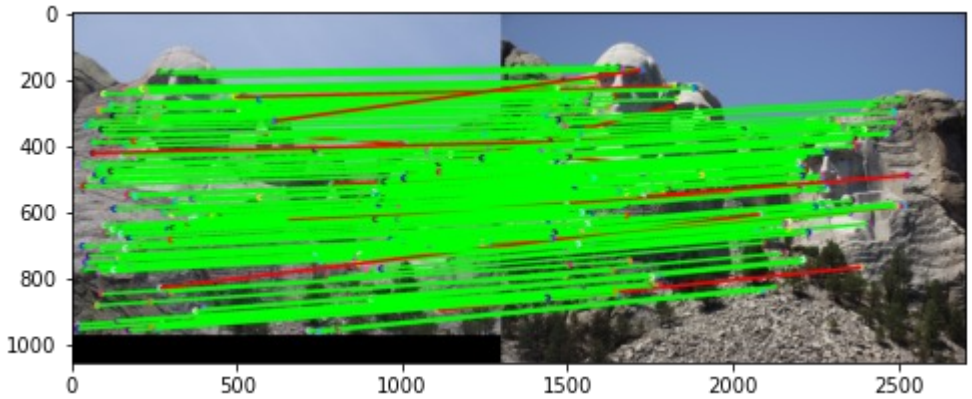
Part 3: Feature matching

[insert visualization of matches (with green/red lines for correct/incorrect correspondences) for Notre Dame image pair from pa2.ipynb here]



matches (out of 100): [196/100 required matches]
Accuracy: [0.933673]

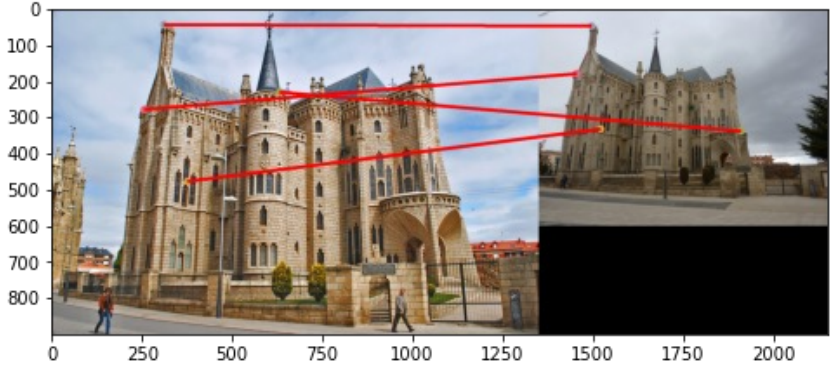
[insert visualization of matches for Mt. Rushmore image pair from pa2.ipynb here]



matches: [181/100 required matches]
Accuracy: [0.928177]

Part 3: Feature matching

[insert visualization of matches for Gaudi image pair from pa2.ipynb here]



matches: [4/100]
Accuracy: [0.000000]

[Describe your implementation of feature matching here]

The function compute_feature_distances calculates the Euclidean distance between each feature in features1 and features2. This results in a distance matrix where each entry represents a pairwise feature distance.

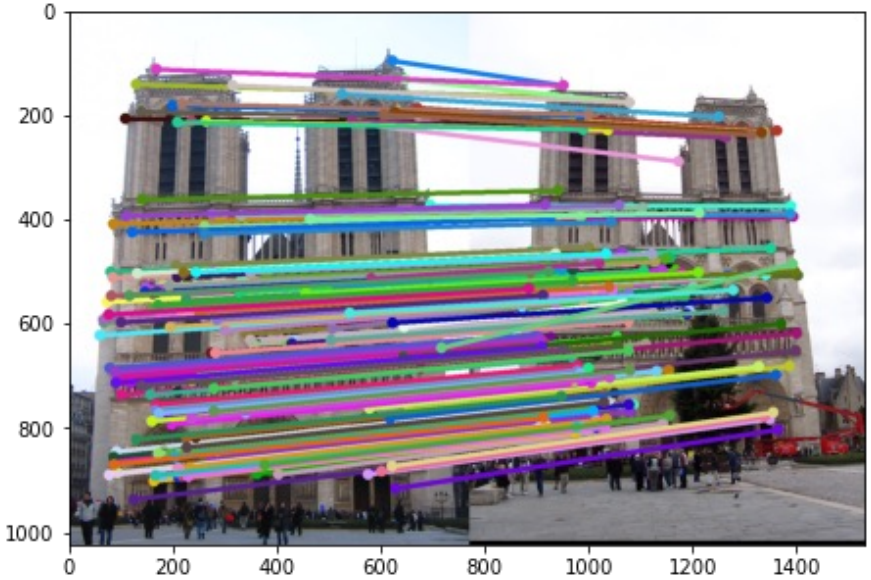
For each feature in features1, the two closest features from features2 are identified by sorting the distance matrix.

The ratio of the smallest distance to the second smallest is computed for each feature in features1. If the ratio is below a threshold (e.g., 0.8), the match is considered valid. This ensures only confident matches pass.

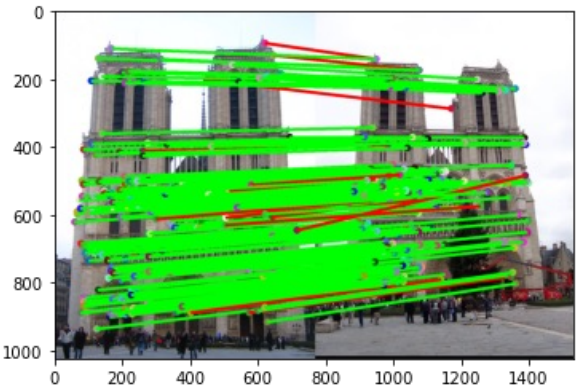
The valid matches and their confidence scores (based on the ratio) are returned. Matches with a low ratio indicate a higher confidence.

Part 4: SIFT feature descriptor

[insert visualization of SIFT feature descriptor from pa2.ipynb here]



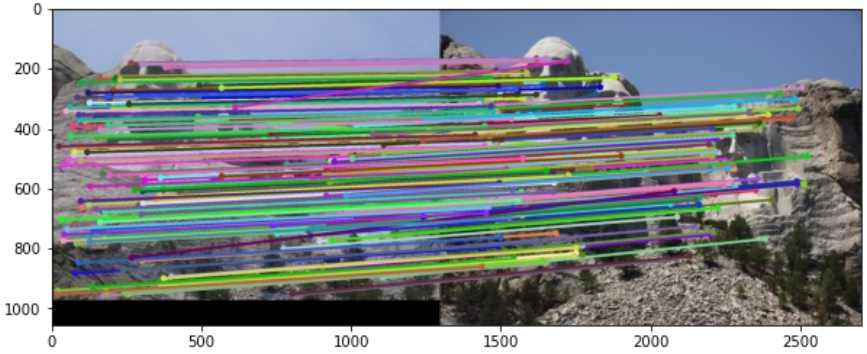
[insert visualization of matches (with green/red lines for correct/incorrect correspondences) for Notre Dame image pair from pa2.ipynb here]



matches (out of 100): [196/100]
Accuracy: [0.933673 - 93.37%]

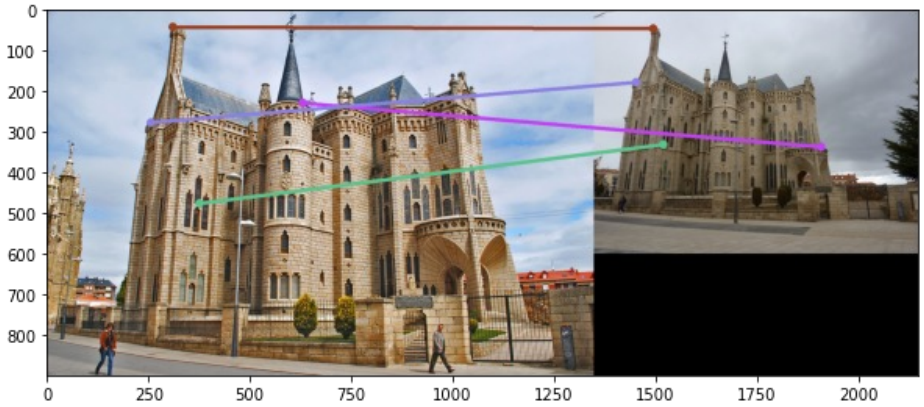
Part 4: SIFT feature descriptor

[insert visualization of matches for Mt. Rushmore image pair from pa2.ipynb here]



matches: [181/100]
Accuracy: [0.928177 – 92.8%]

[insert visualization of matches for Gaudiimage pair from pa2.ipynb here]



matches: [4/100]
Accuracy: [0.000000]

Part 4: SIFT feature descriptor

[Describe your implementation of SIFT feature descriptors here]

First, I used get_magnitudes_and_orientations() to compute gradient magnitudes and orientations using Sobel derivatives. Then, in get_gradient_histogram_vec_from_patch(), I divided the 16x16 patch into a 4x4 grid. For each grid cell, I computed an 8-bin histogram with gradient magnitudes as weights and orientations as bins. The histograms are concatenated into a 128-dim feature vector representing the keypoint. In get_feat_vec(), I normalized the vector and applied square-root normalization. Finally, in get_SIFT_descriptors(), I computed the SIFT descriptor for all keypoints.

[Why are SIFT features better descriptors than the normalized patches?]

SIFT features are better than normalized patches because they capture both gradient magnitude and orientation, making them robust to changes in illumination, scale, and rotation. SIFT also retains spatial information by dividing the region around a keypoint into a grid, ensuring local structure is preserved.

Normalized patches, on the other hand, rely only on raw pixel intensity values, which makes them sensitive to changes in lighting, rotation, and scale, leading to less reliable matching. SIFT's use of histograms and square-root normalization is also a bonus.

Conclusion

[Why aren't our version of SIFT features rotation- or scale-invariant? What would you have to do to make them so?]

Our SIFT implementation lacks rotation and scale invariance because we don’t account for changes in the orientation or size of the keypoints. To achieve rotation invariance, we would need to compute the dominant gradient orientation at each keypoint and rotate the feature descriptors accordingly, aligning them to a consistent direction. For scale invariance, we would need to detect keypoints at different scales, and extract features relative to each keypoint’s scale, ensuring consistency across varying image sizes or zoom levels.