Images of the Russian Empire: Colorizing the Prokudin-Gorskii photo collection

Overview

Sergei Mikhailovich Prokudin-Gorskii (1863–1944) was a visionary in color photography. He photoed color pictures by recording three exposures of every scene onto a glass plate using a red, a green, and a blue filter.

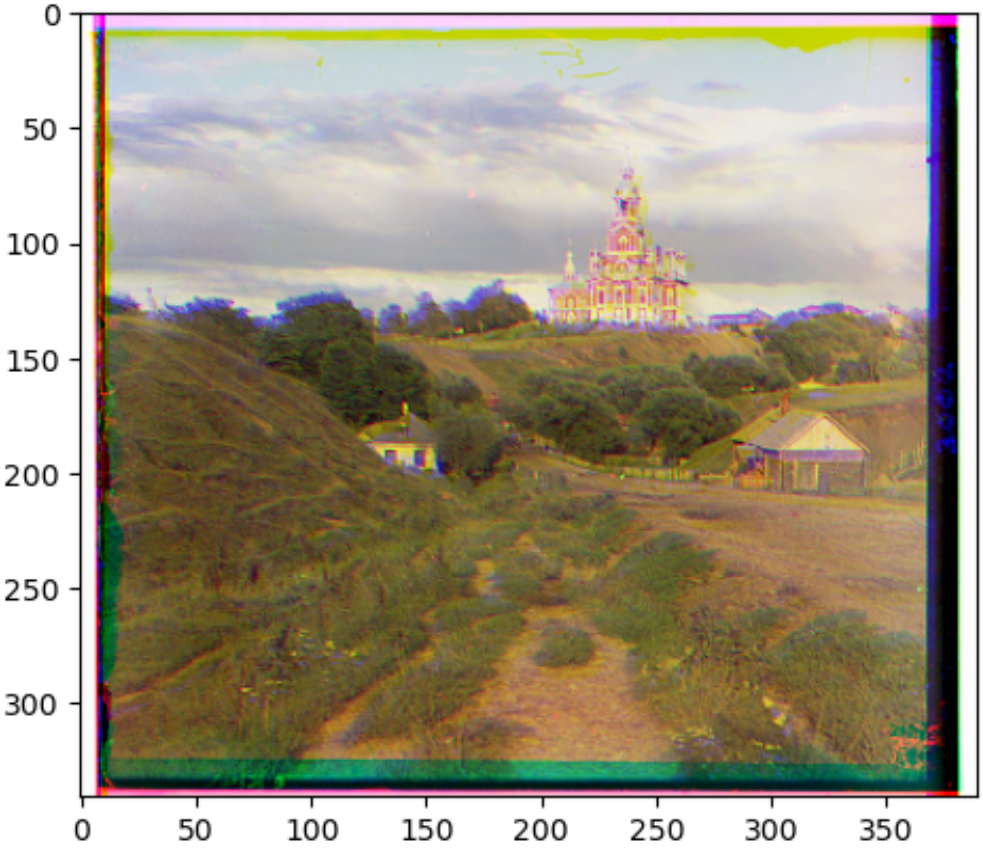
To convert these three single-channel images into an RGB image, the problem can be divided into two parts: first, finding an algorithm to align the single-channel images, and second, identifying a suitable evaluation function to measure how well the images align.

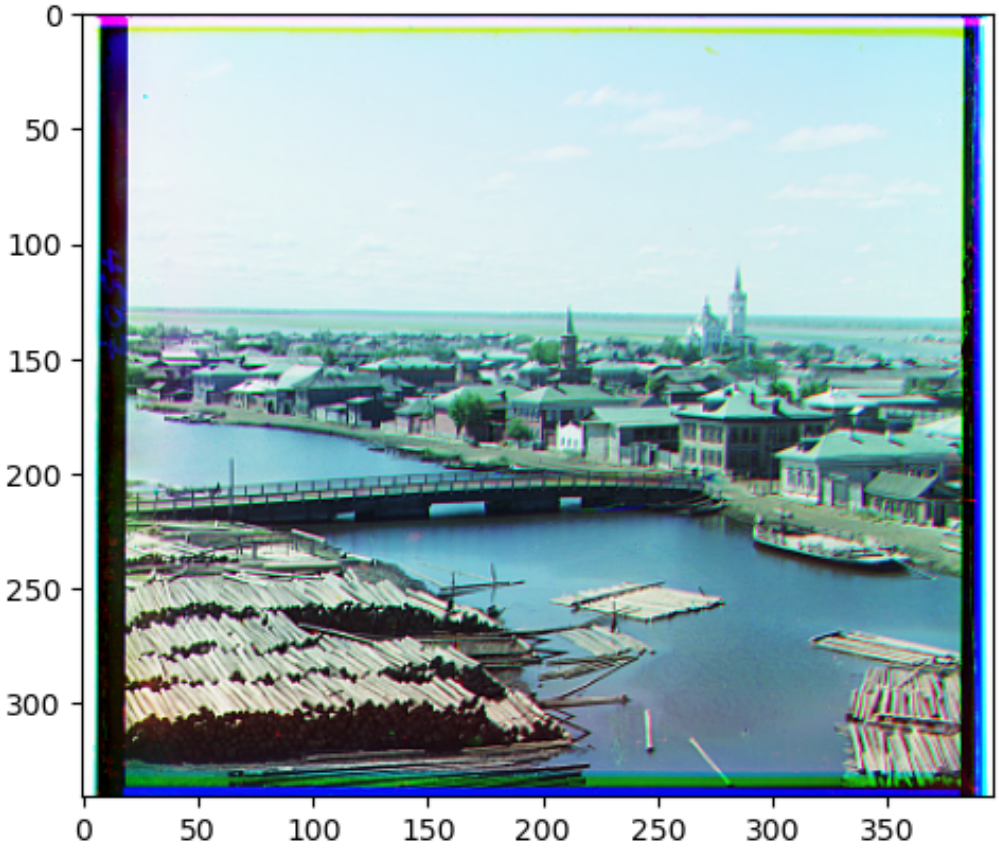
For the algorithms, I used exhaustive search, image pyramids, and edge detection. For the evaluation function, I tested both NCC (Normalized Cross-Correlation) and the Pearson correlation coefficient, and found that NCC performed better.

Exhaustive Search

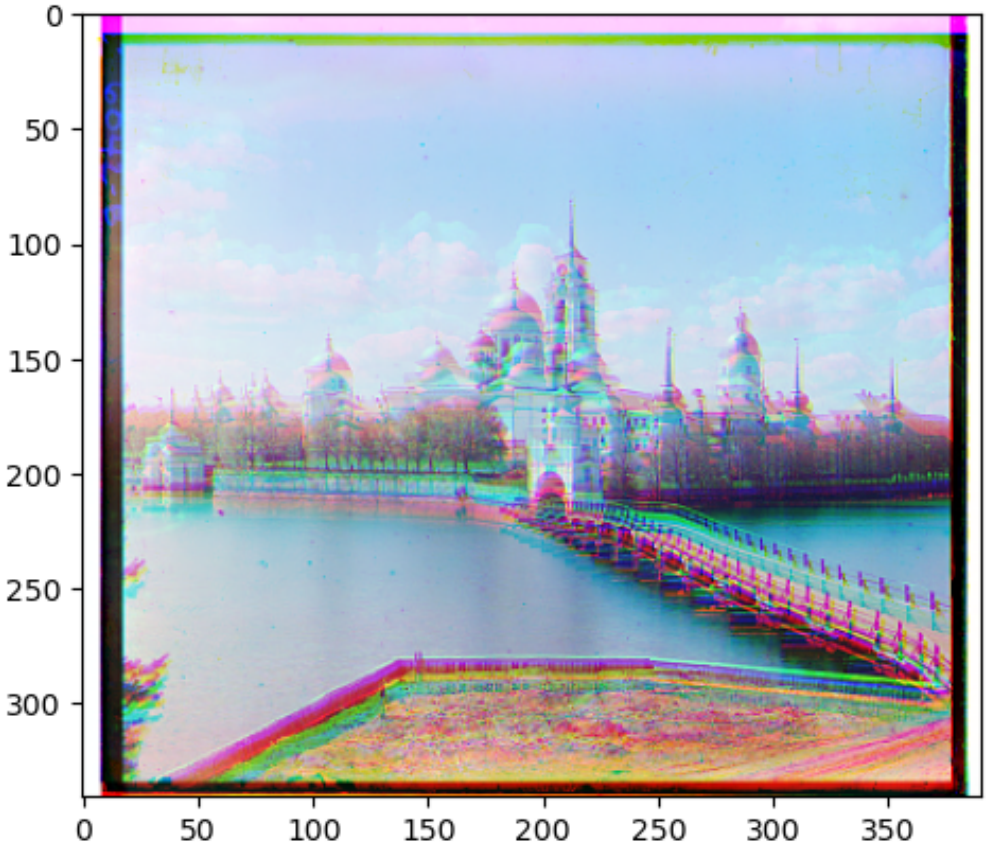
The simplest way to solve this problem is to exhaustively search a grid of translation parameters for the single-channel images. Then, use NCC (Normalized Cross-Correlation) to evaluate the similarity between two images. NCC is computed as the dot product of two normalized vectors: (image1./||image1|| and image2./||image2||).

Combining exhaustive search with NCC can be effective for aligning two low-resolution images:





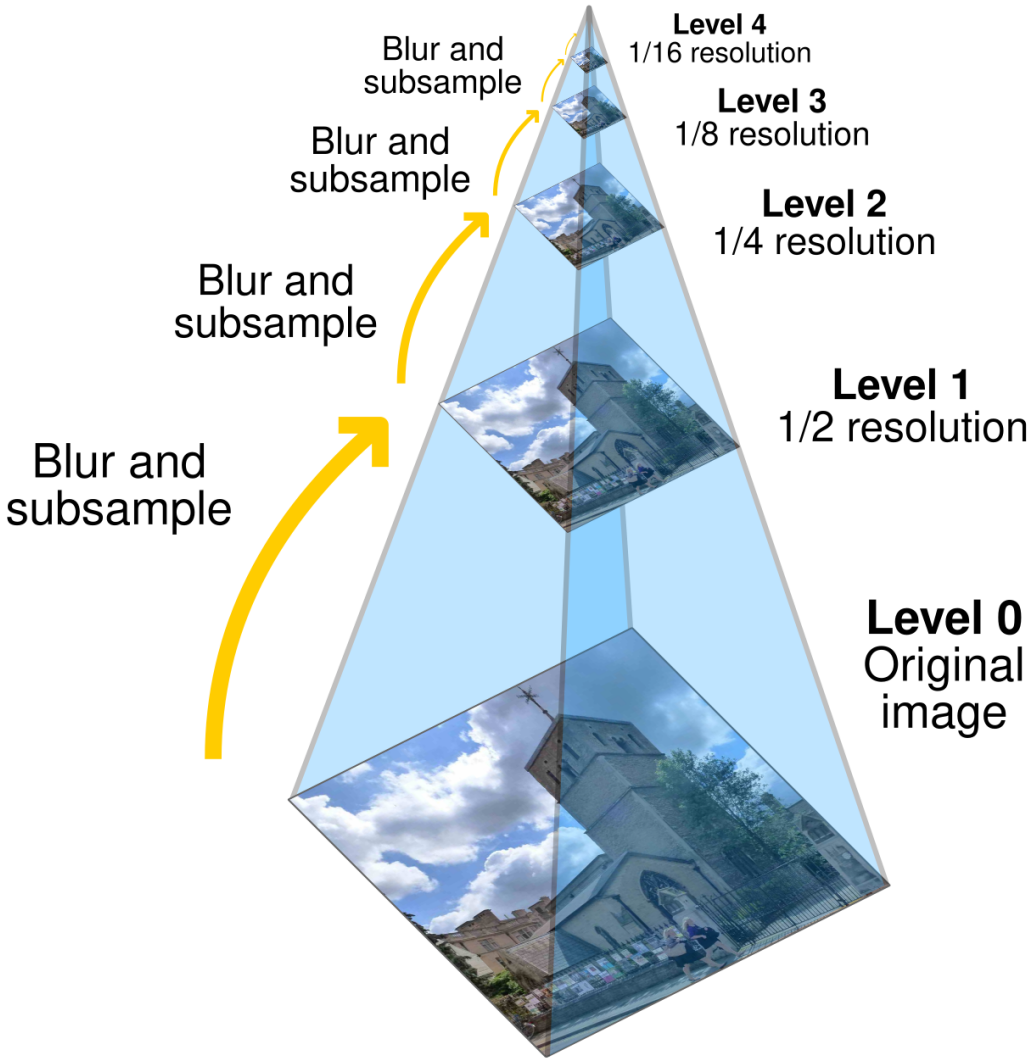
However, it performs poorly for another low-resolution image:



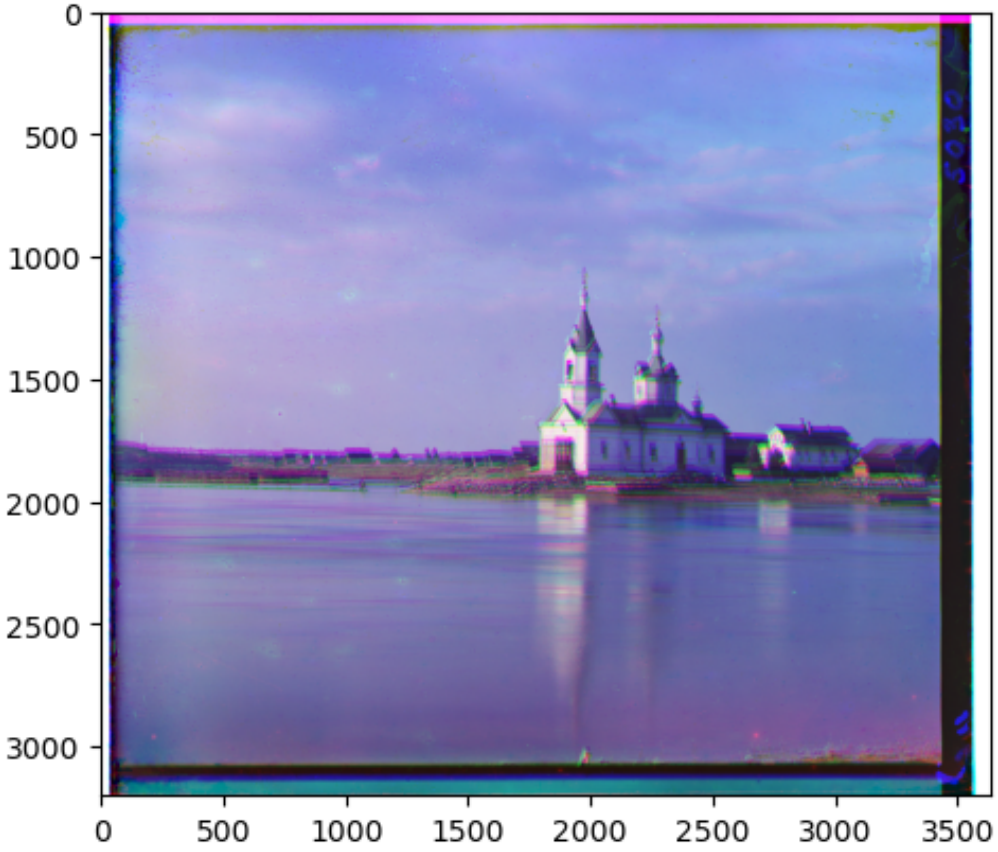
Also, when it comes to the high-resolution images, the approach becomes too time-consuming. Therefore, using an image pyramid to reduce computational complexity is necessary.

Image pyramid

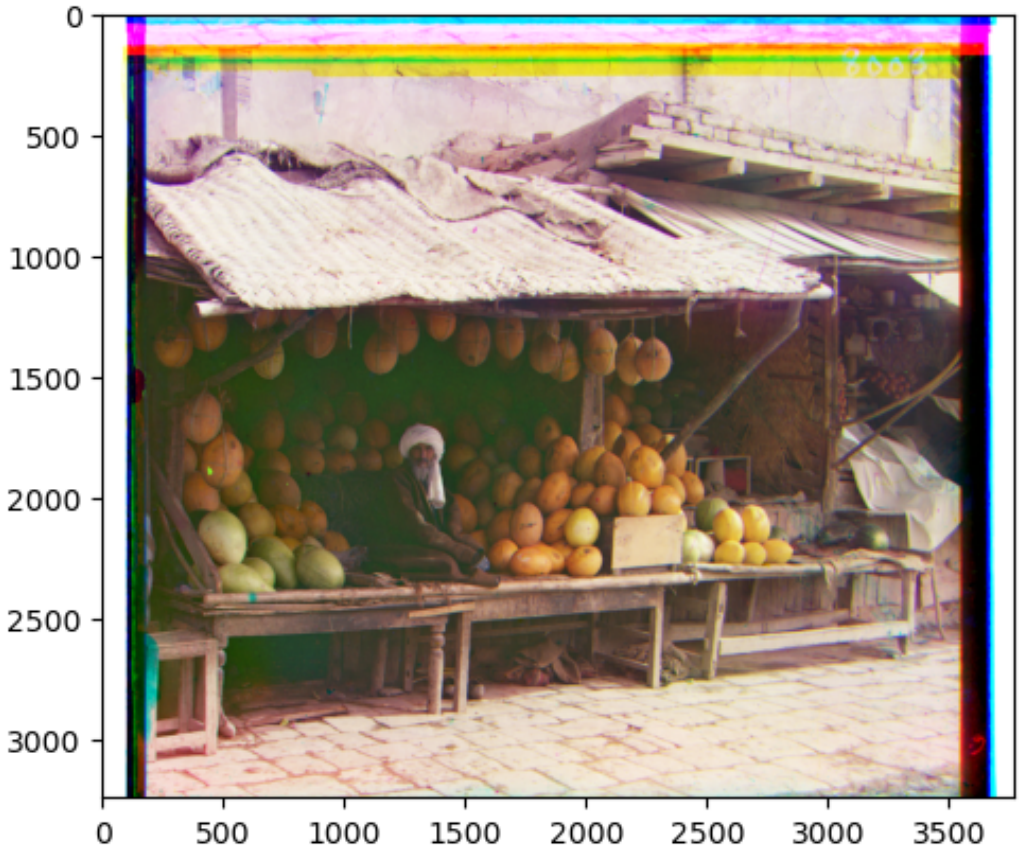
The image pyramid employs a concept similar to the divide-and-conquer approach. High-resolution images are first downsampled to lower resolutions, and the result from the low-resolution alignment is used as an initial estimate for the higher-resolution image. The image pyramid shows in the figure:

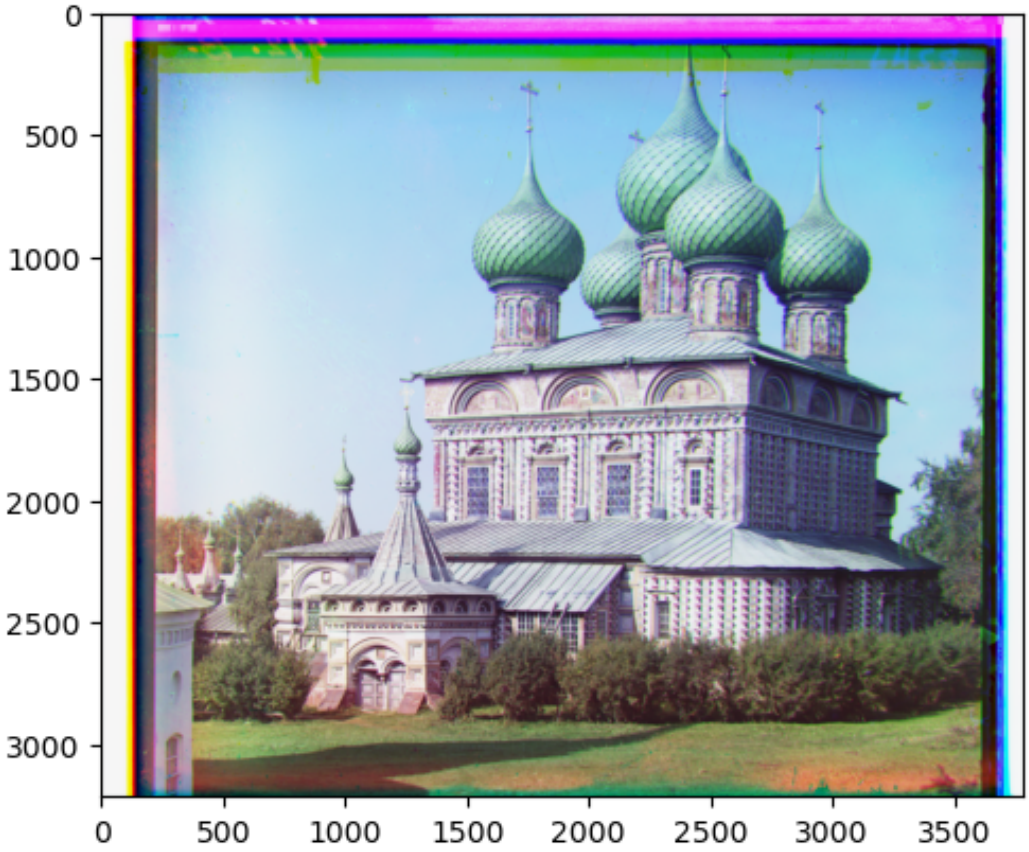


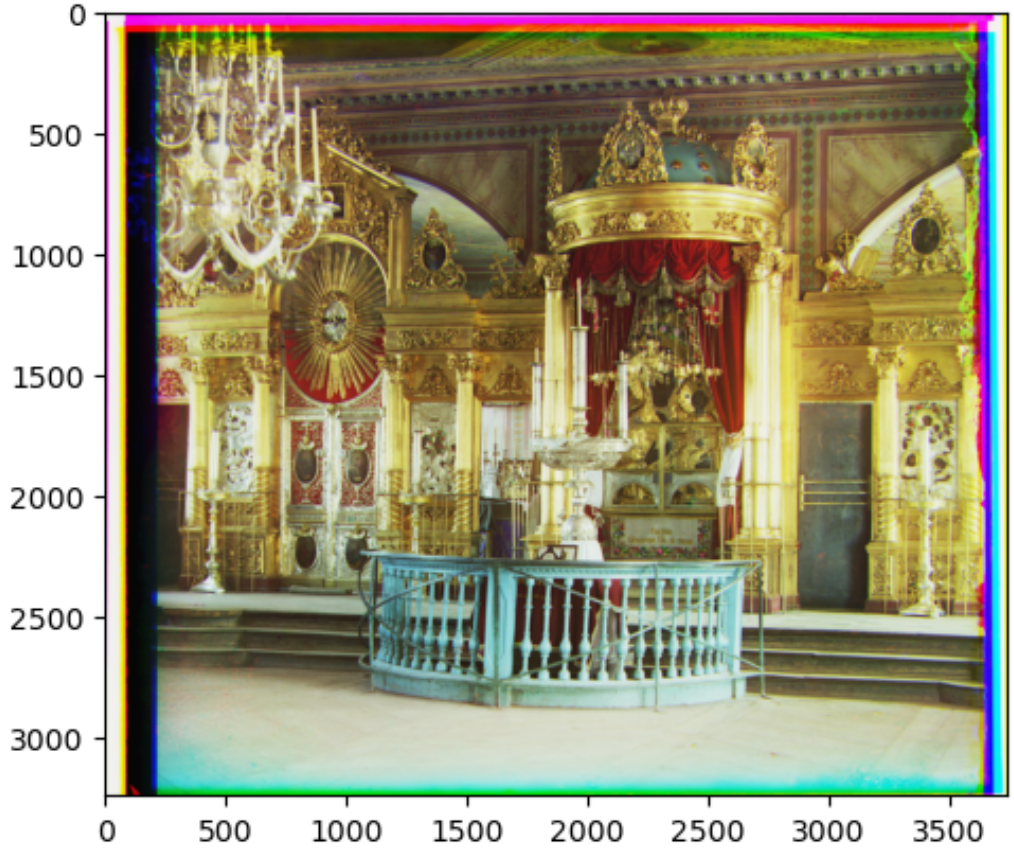
By this method, many images are converted well:

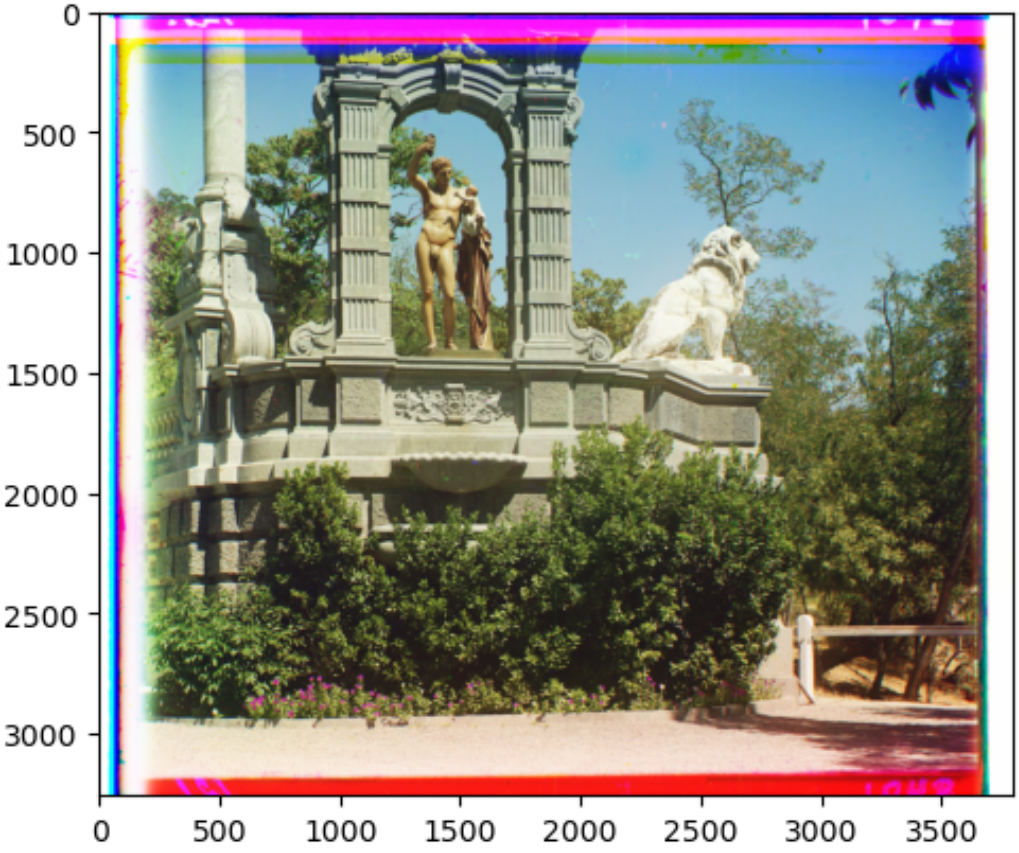

0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
3500

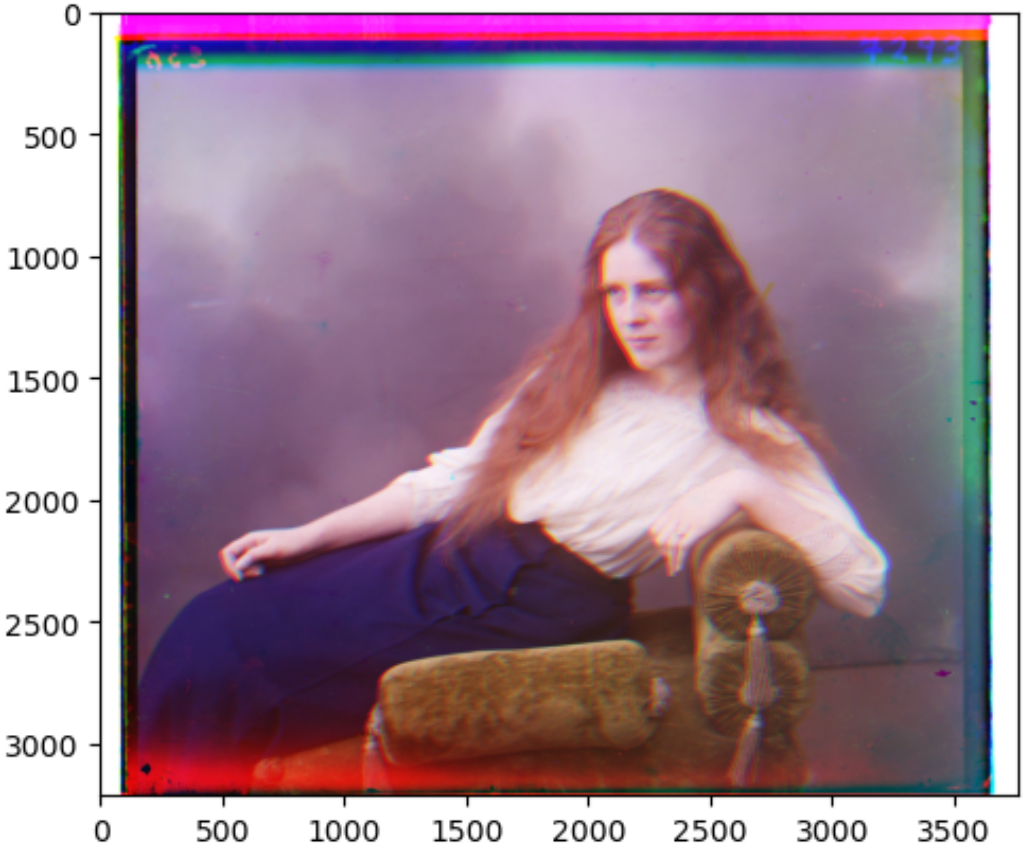

0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
3500


0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
3500

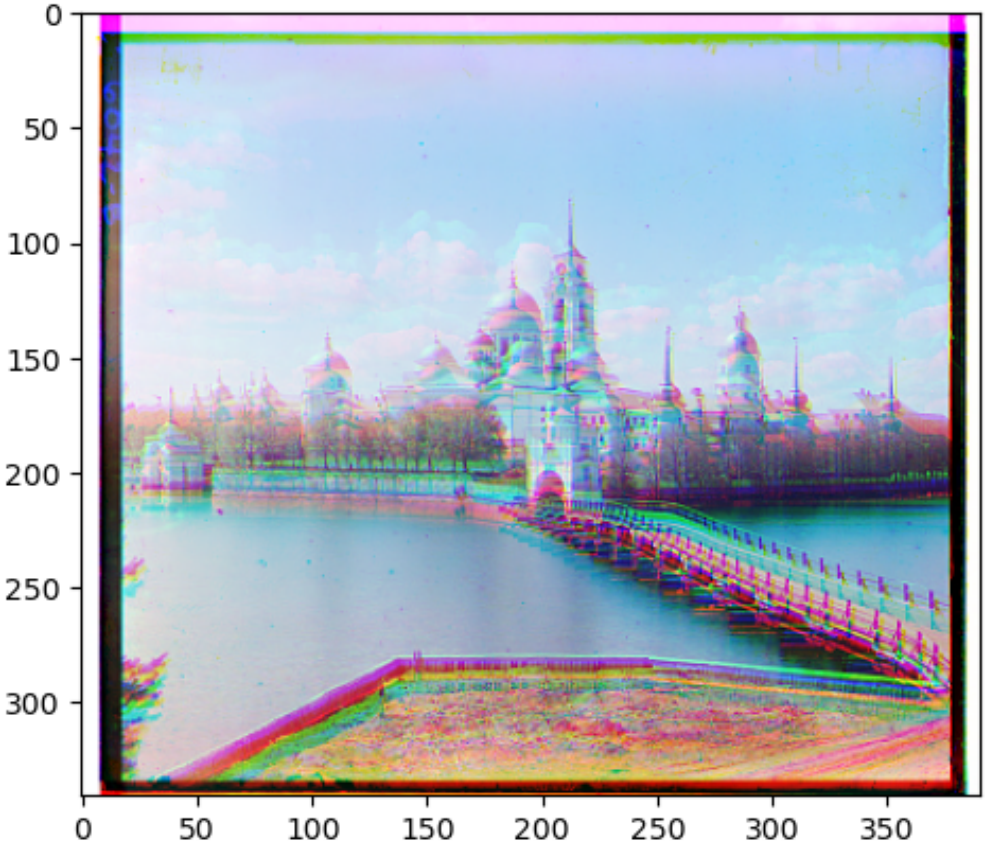

0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
3500

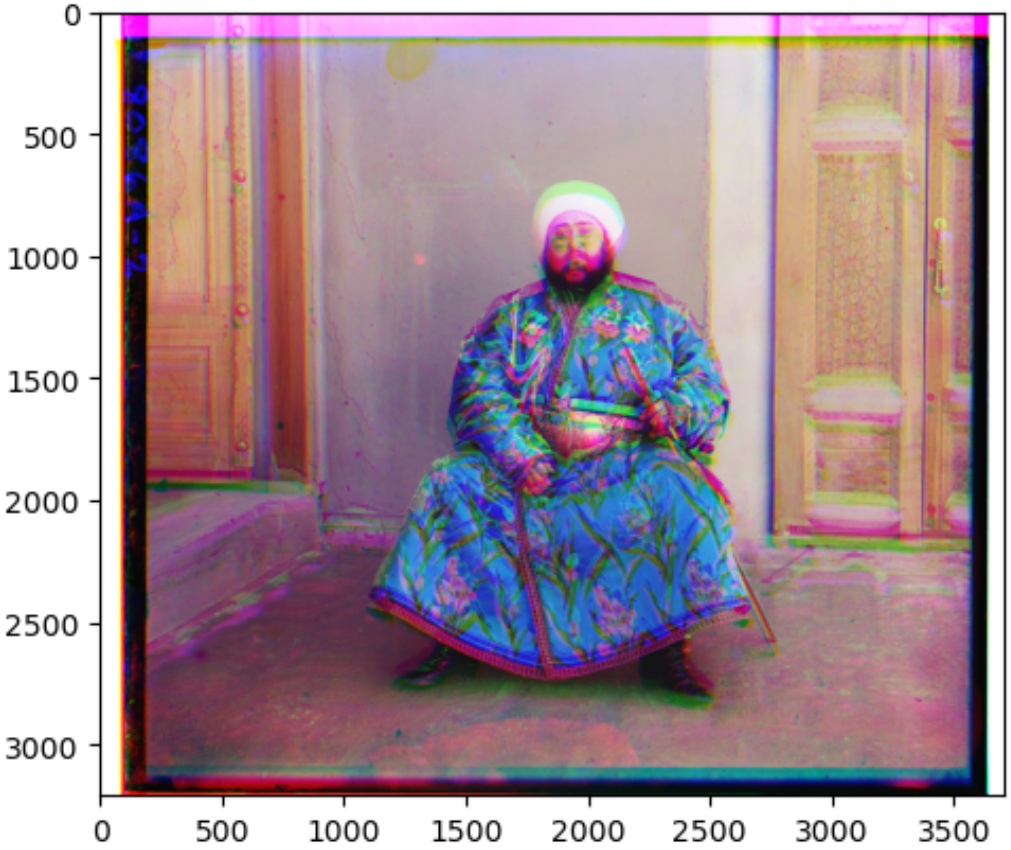

0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
3500

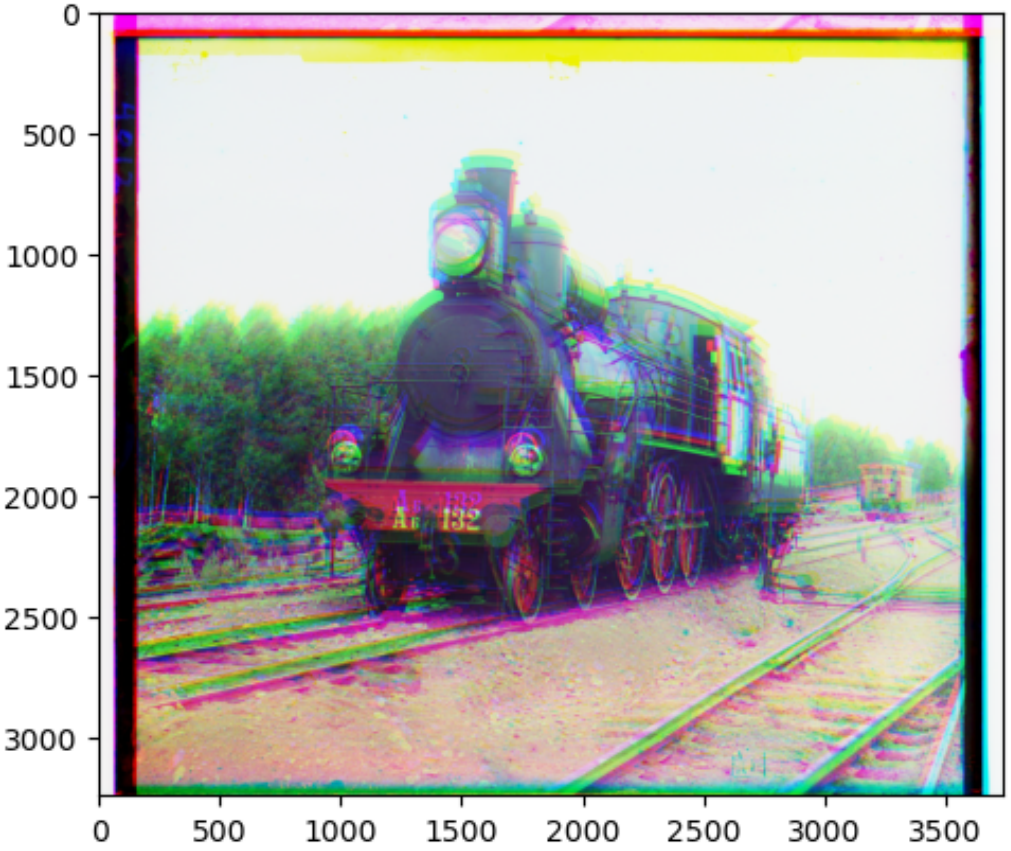

0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
3500

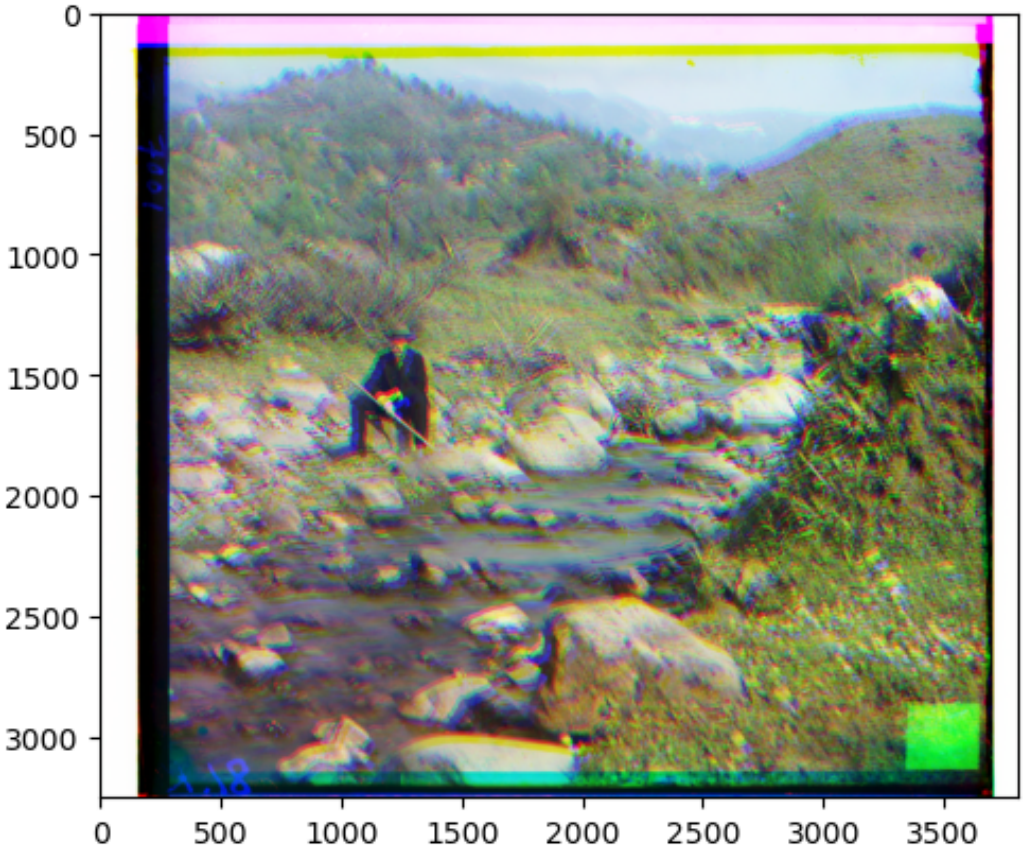


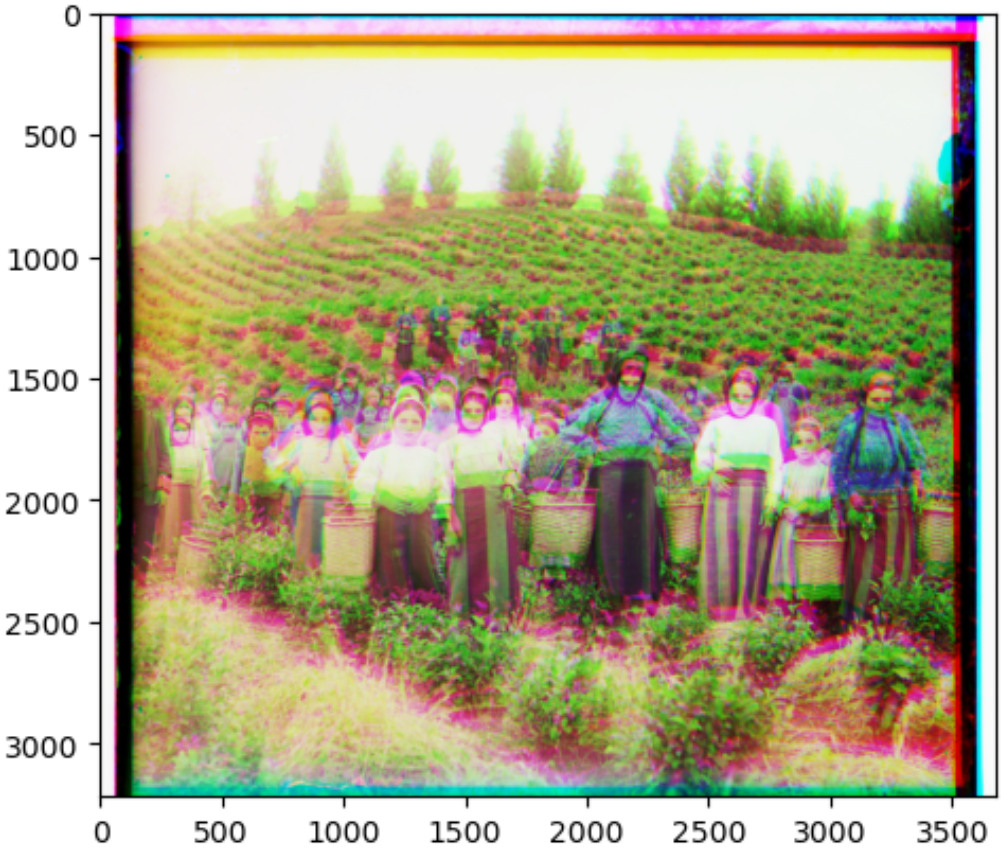
However, there are still some images performing poorly:




0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
3500


0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
3500





I initially tried to improve the results by changing the evaluation function to the Pearson correlation coefficient

I initially tried to improve the results by changing the evaluation function to the Pearson correlation coefficient, but the image quality did not improve, and the computation time increased.

As a result, I decided to address the issue by optimizing the algorithm.

Edge Detection

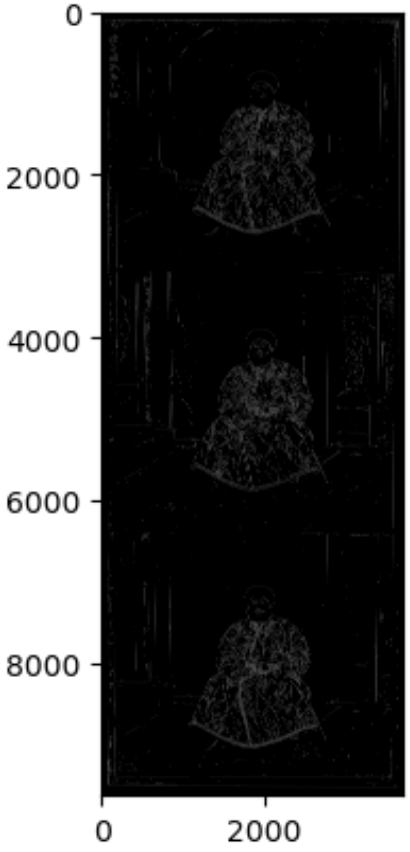
A more effective way to align the images is to pre-process them into edge images. This can be done using a Canny filter.

The process involves the following steps: The process begins with applying a Gaussian blur—adjusted by the sigma parameter—to reduce noise in the image This step ensures that small color or intensity changes don't affect the next stages. Once noise is minimized, Sobel edge detection is used along the x and y axes to compute the intensity gradients where edges are likely to be. The Sobel operator estimates the derivative of the gradient between dark and light regions, marking the peak as an edge pixel.

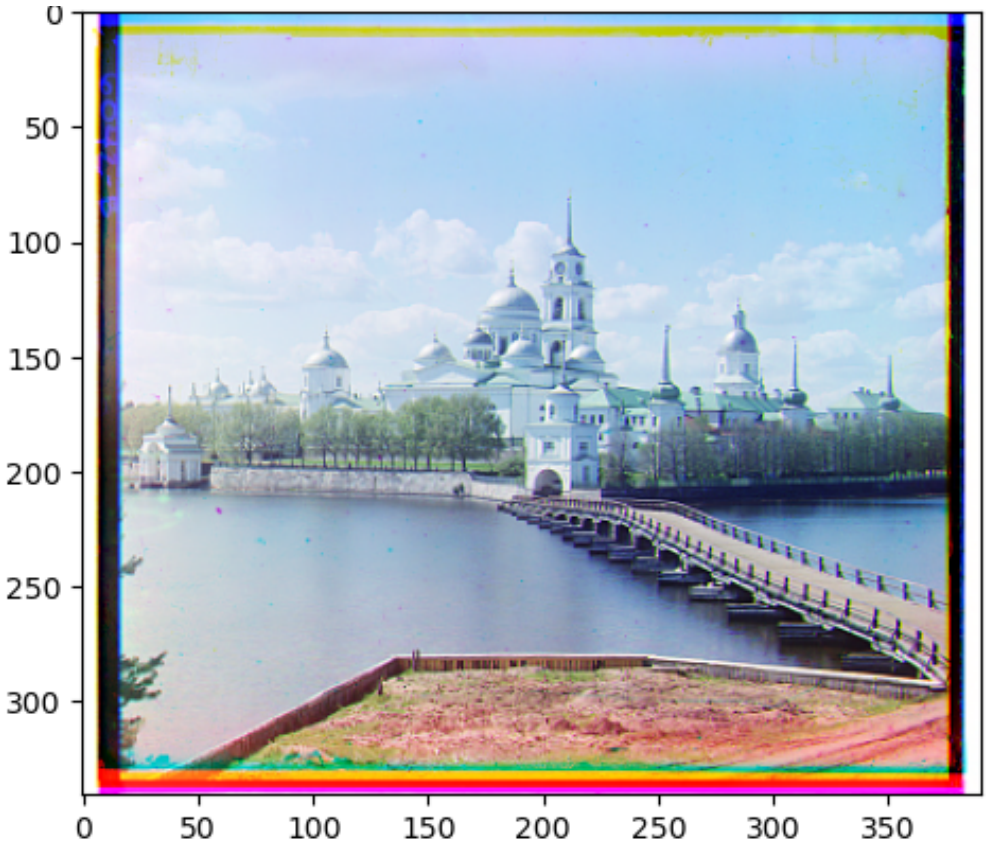
Afterward, non-maximum suppression is employed, which eliminates pixels that are distant from true edges, resulting in thinner and more defined edge lines. To refine the detection further, a double threshold is applied. Pixels with gradients surpassing the higher threshold are marked as strong edge candidates, while those below the lower threshold are discarded. Pixels falling between the thresholds are considered weaker edge candidates.

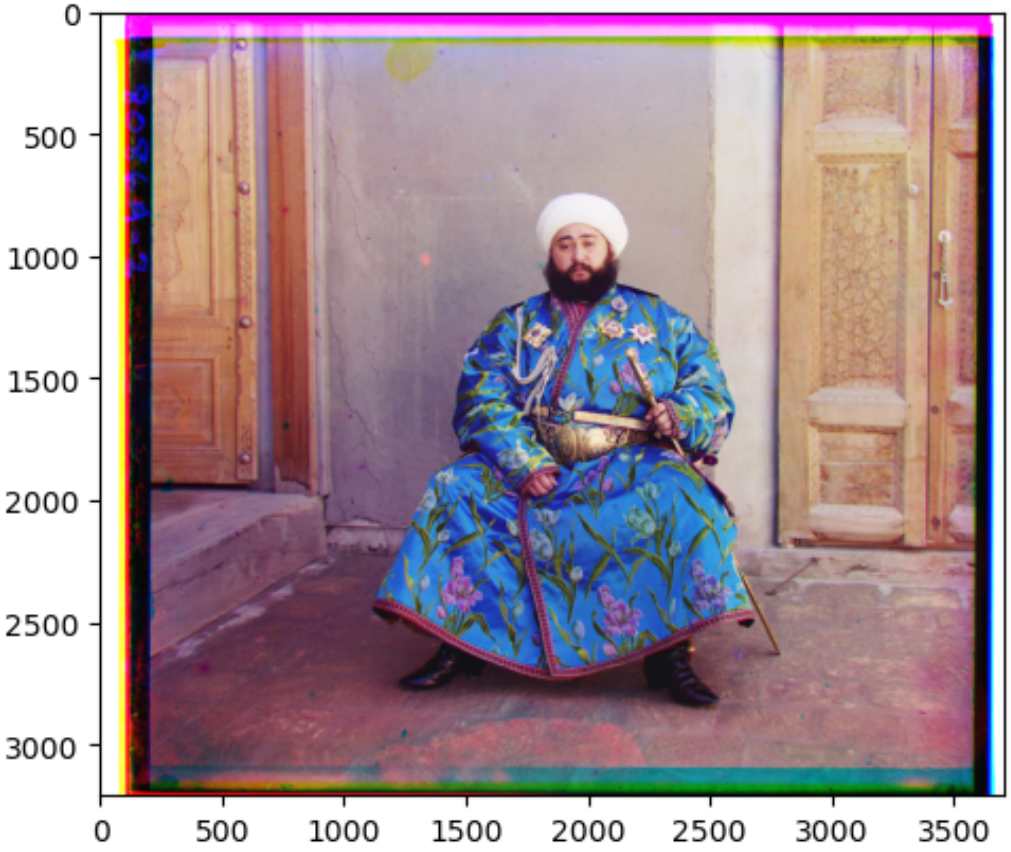
Finally, hysteresis is applied to solidify edge detection. Weak candidate pixels linked to strong edges are kept, while those not connected are ignored. This ensures the edge map is both precise and continuous, filtering out unwanted detections caused by noise or minimal color shifts.

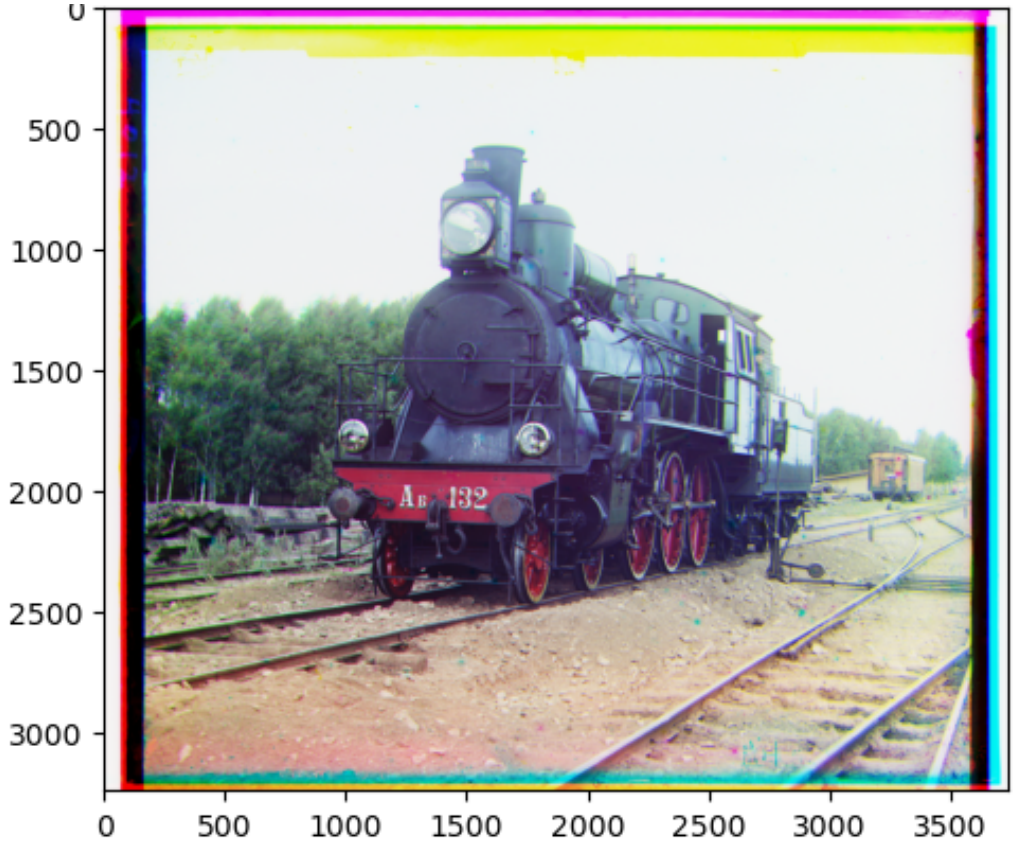
The original emir.tif compared to the edge-detected version of emir.tif is shown below:

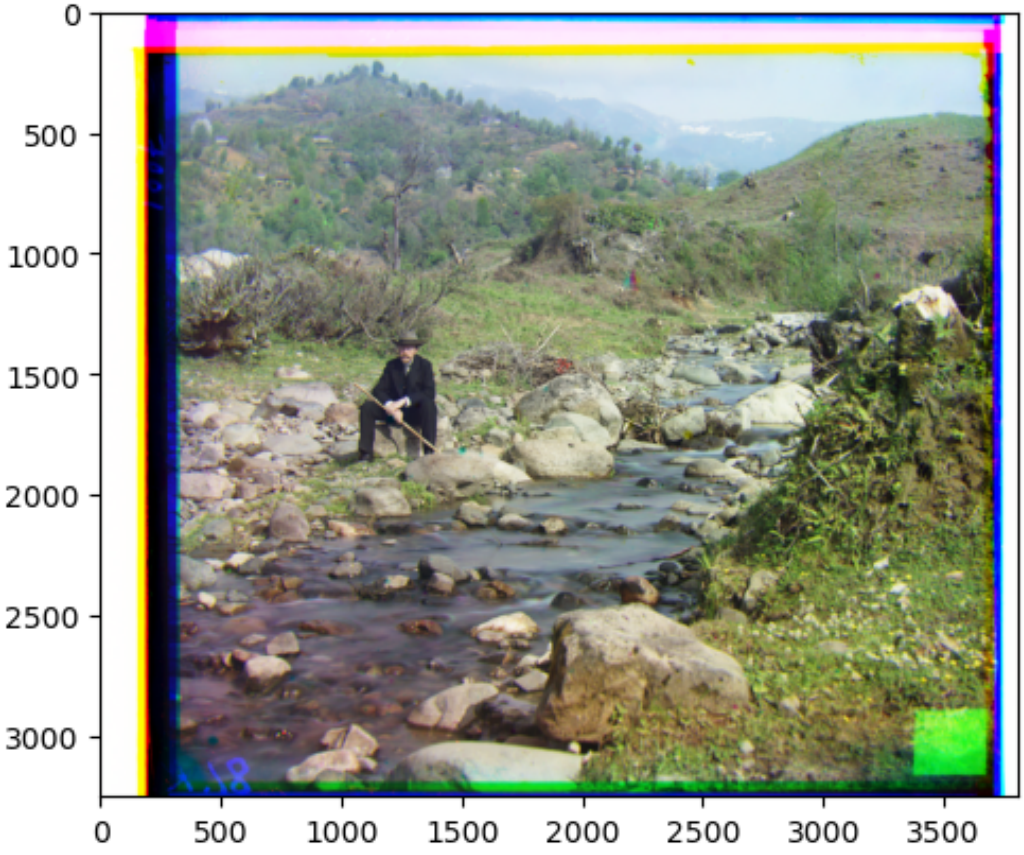


In this way, we significantly improve the performance of previously problematic images.

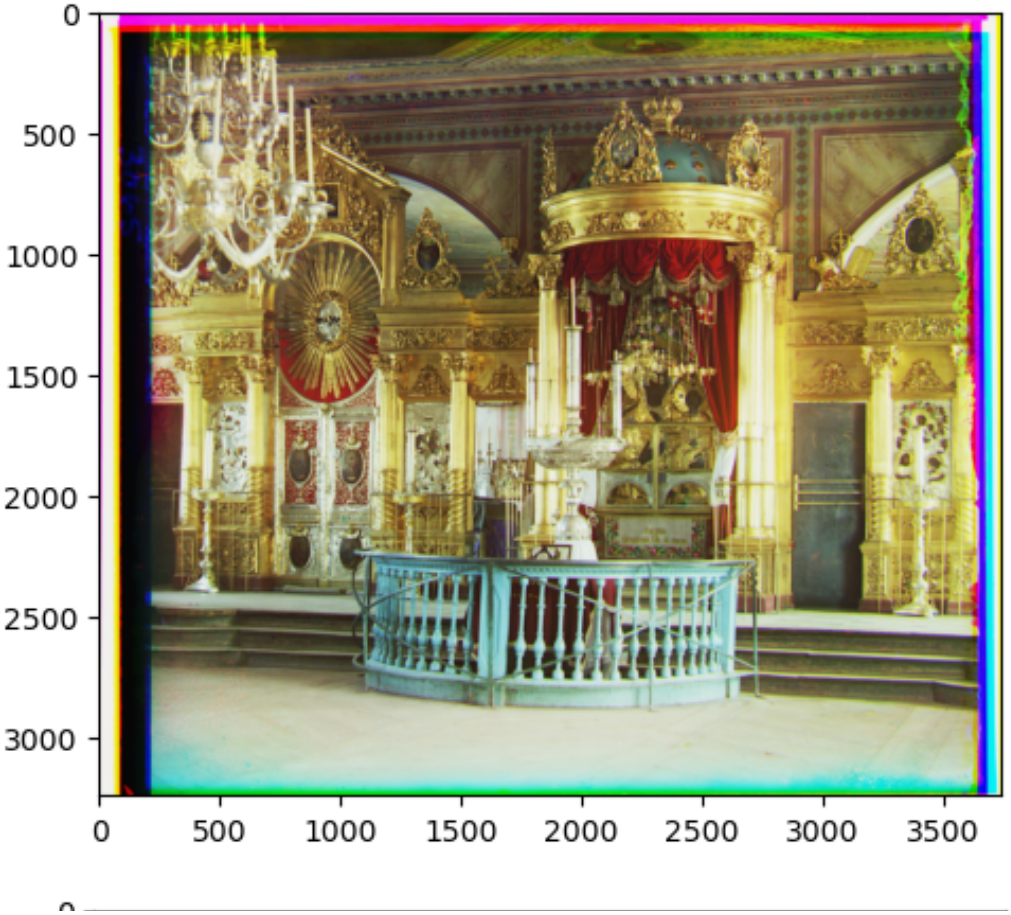

0
50
100
150
200
250
300
0
50
100
150
200
250
300
350

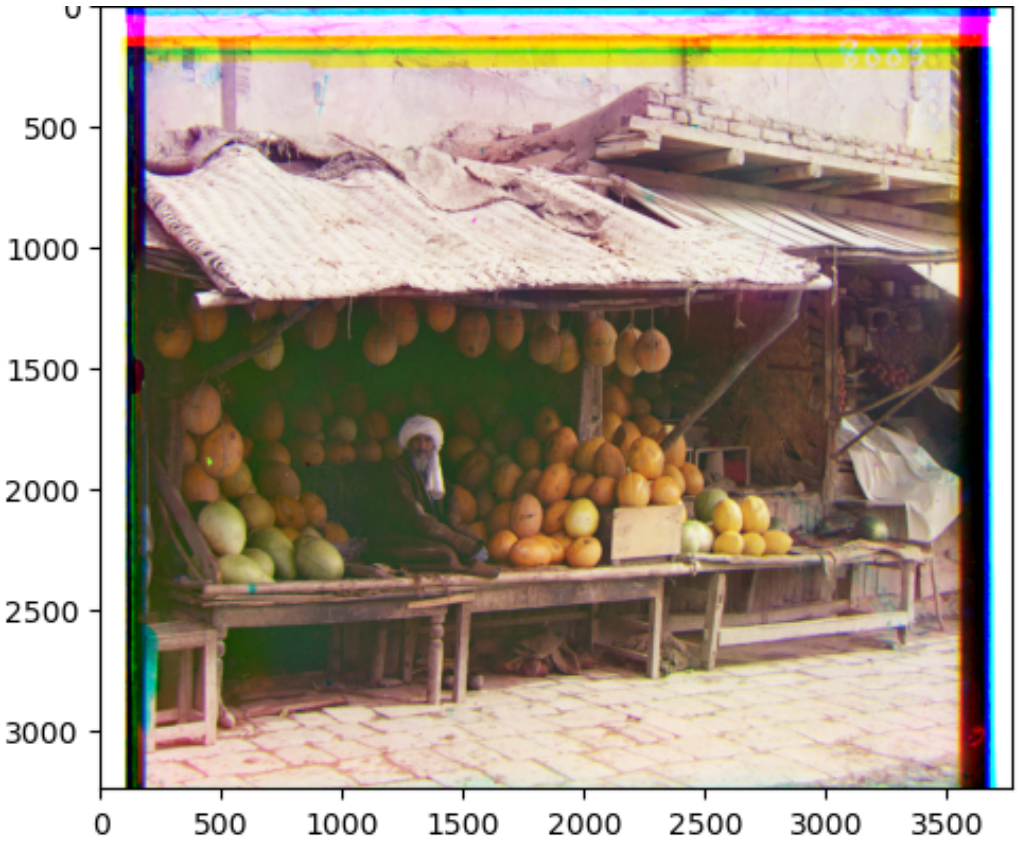

0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
3500

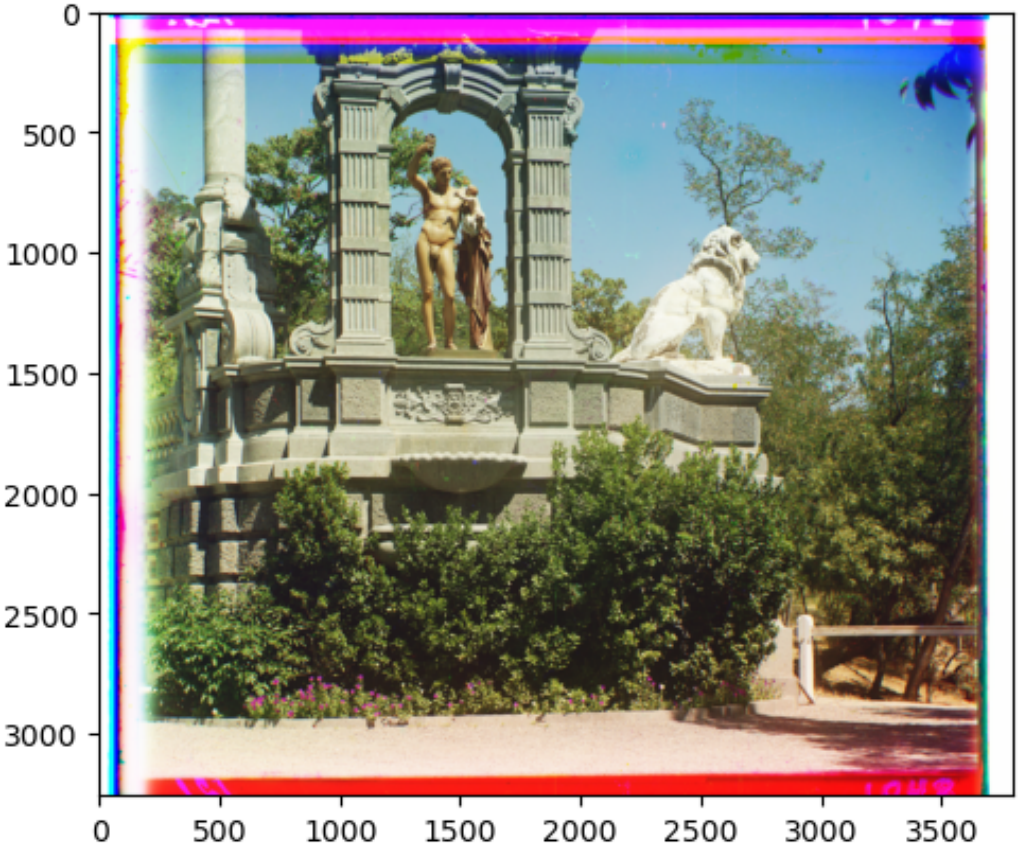

0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
3500

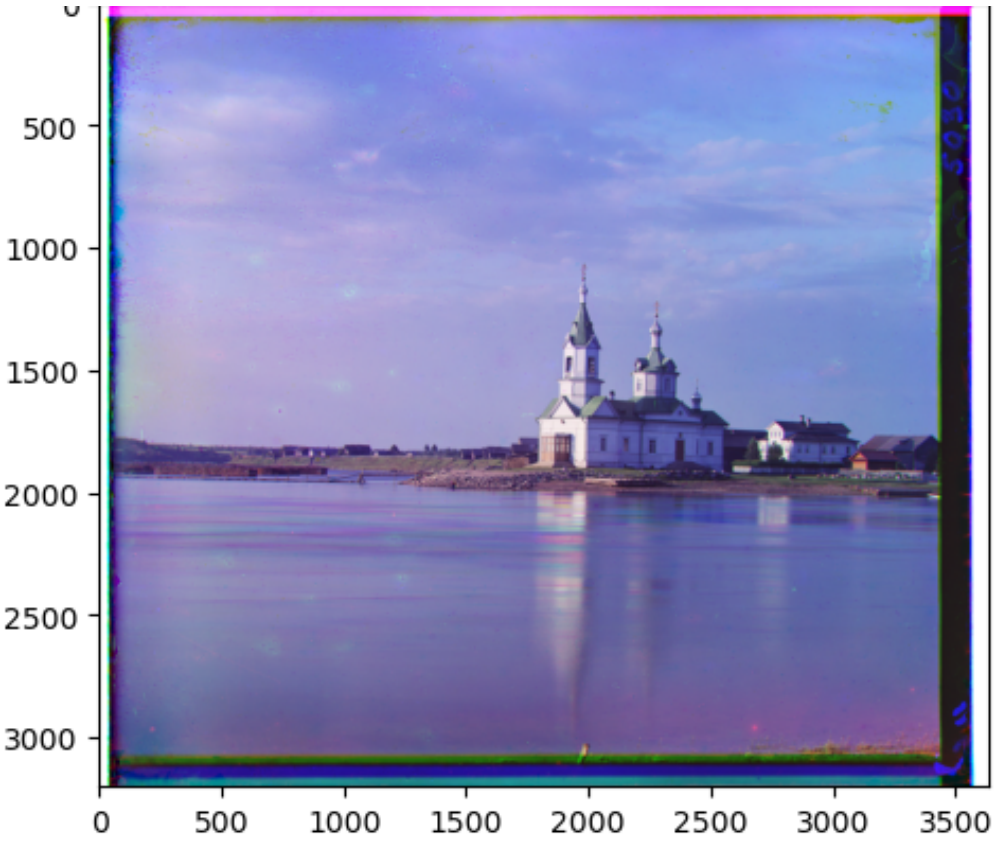

0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
3500

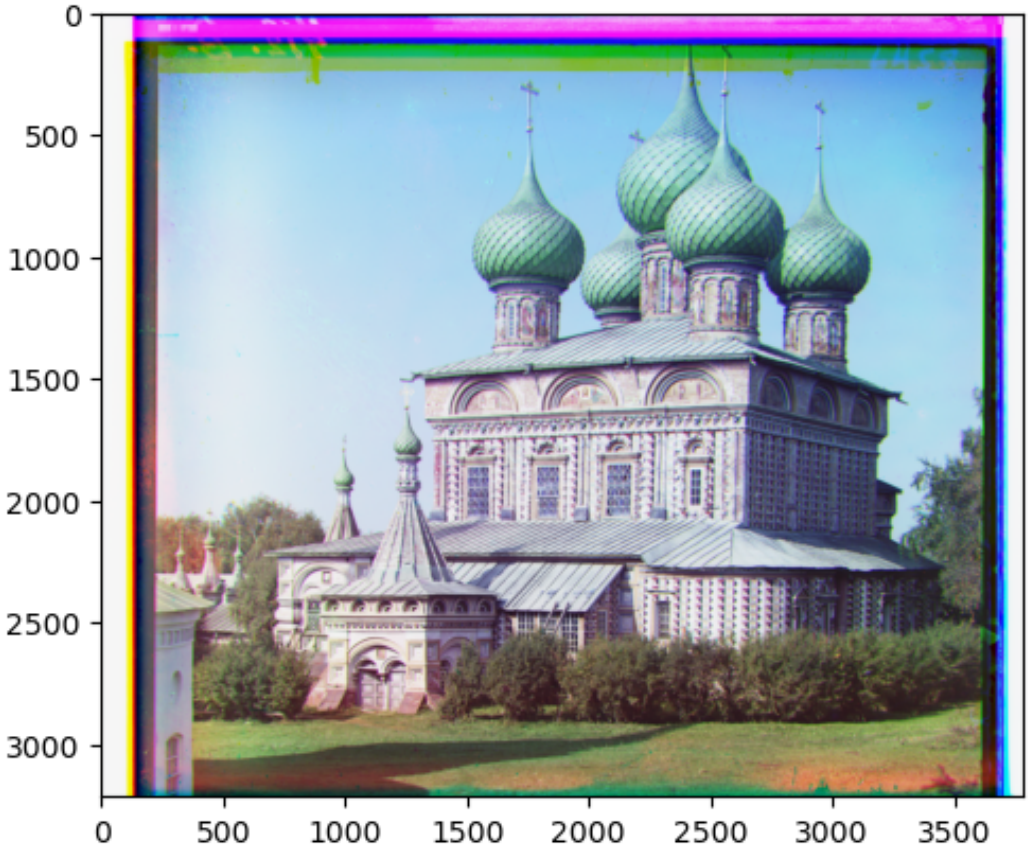
The other images also perform well, except for lady.tif, which performs better without edge detection.




0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
3500


0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
3500


0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
3500


0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
3500

Reference

Reference

Pyramid: https://en.wikipedia.org/wiki/Pyramid_(image_processing)

Edge detection: https://datacarpentry.org/image-processing/edge-detection.html