Ukrainian Catholic University

Faculty of Applied Sciences

Business Analytics & Computer Science Programmes

SVD-based image inpainting

Linear Algebra final project report

Authors:
Viktor Povazhuk
Petro Hentosh
Yaroslav Tsymbalista

30 May 2022

1 Introduction

Theme of our project is image inpainting. It implies that we take an image with a damaged part and trying to restore missing pieces by analyzing a good area.

This technology used in different photo editors to restore a part of the background image, to delete unnecessary objects or to renew old photos.

The main task of this project is to fill the target region correctly or if I try to paraphrase to get rid of this region.

2 Related works and possible approaches

Because the problem of inpainting is widely known and useful, scientists have already developed lots of approaches to solve it. We can divide all approaches in two big groups: patch-based and diffusion-based.

Patch-based methods are based on techniques to fill in the missing region patch-by-patch by searching for well-matching replacement patches in the undamaged part of the image and copying them to corresponding locations. Here are some existing methods:

- Alilou et al. 2017 (Use singular value decomposition and an approximation matrix)
- Lu et al. 2018 (Use gradient-based low rank approximation)
- Zhang et al. 2018 (Use joint probability density matrix (JPDM))
- Jin et al. 2015 (Annihilation property filter, low rank structured matrix)
- Guo et al. 2018 (Two-Stage Low Rank Approximation (TSLRA))

Diffusion-based methods fill in the missing region (the "hole") by smoothly propagating image content from the boundary to the interior of the missing region.

- Li et al. 2016 (Distance and direction between the damaged pixel and its neighborhood pixels)
- Sridevi et al. 2019 (Fractional-order derivative and Fourier transform)

3 Theory

3.1 SVD

The main part of algorithm is singular value decomposition (SVD) approximation matrix.

It represents a matrix as a sum of matrices (i.e., bases of SVD) where each matrix include a particular feature of the initial matrix. Also, each basis care as great values related to its feature (i.e., power) as possible.

If we have a $m \times n$ matrix X, then SVD desomposes it as:

$$X = U \times S \times V,$$

where:

- columns of U - orthonormal eigenvectors (EVc) of $X \cdot X^T$,

- columns of V - orthonormal EVc of $X^T \cdot X$,
- S - diagonal (D) matrix with square roots of eigenvalues, called singular values (SV), in descending order corresponding to columns of U.

Main feature of the image corresponds to basis with higher values. Hence, it is related to higher SV. Noise or cracks are small details of the whole picture, so, matrices that represent them correspond to small SV.

Thefore, we can get rid of some of matrices that represent noise or cracks and the remaining sum will form a restored picture. Or, in terms of object removing task, it will be a cleaned visualization. Image formed from particular bases is called principle.

3.2 SSIM

During the algorithm we compare 2 sligthly different matrices of the same picture. To do this, we need particular metrics. There are 2 well-known methods: mean square error (MSE) and index of similarity (SSIM).

Advantage and disadvantage of MSE is due to its simplicity. Advantage is easy calculation. Negative side: it doesn't consider more complex dependencies of pixels in picture. Whereas these dependencies are caught by human visual system.

SSIM is a bit more complex, but not too much; and it takes into account more features of picture. Therefore, we used it.

SSIM compares pictures as sets of patches, by comparing each pair of patches. For each pair are considered: luminance (L), contrast (C), structure (S).

The formulas are next:

$$L(x,y) = \frac{2\mu_x\mu_y + C1}{\mu_x^2 + \mu_y^2 + C1}$$

$$C(x,y) = \frac{2\sigma_x\sigma_y + C2}{\sigma_x^2 + \sigma_y^2 + C2}$$

$$S(x,y) = \frac{\sigma_{xy} + C3}{\sigma_x\sigma_y + C3}$$

where for appropriate patches x, y:

- μ_x, μ_y - means,
- $\sigma_x, \sigma_y and \sigma_{xy}$ - standard deviations and covariance,
- $C1,\ C2,\ C3$ - small constants.

For SSIM of pair of patches we join 3 features:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C1)(2\sigma_x\sigma_y + C2)}{(\sigma_x^2 + \sigma_y^2 + C2)(\mu_x^2 + \mu_y^2 + C1)}$$

For SSIM of image:

$$SSIM(I,\hat{I}) = \frac{1}{M}\sum_{i=1}^{M} SSIM(x_i, y_i)$$

where:

- M - number of patches.

Want to control that only "noisy" part of image is removed, therefore, source region isn't demaged. For that reason, we find the SSIM of source regions Φ:

$$SSIM(I, \hat{I}) = \frac{1}{N} \sum_{i \in \Phi} SSIM(x_i, y_i)$$

4 Algorithm

We can divide our whole algorithm in 3 parts. Firstly, given an image, with a target region, algorithm calculates a grayscale approximation matrix using singular value decomposition. Then, we reconstruct the target region by using this matrix: fill-in the missing areas using exemplar-based texture synthesis. In the end, we apply multi-scale approach to this procedure.

4.1 Approximation matrix

Initially, we have a damaged picture. Person specifies damaged region (target region), and rest part of the image is well (source region). Then we convert picture to grayscale because it is much easier to work with 1 channel of colors instead of 3 channels.

On the obtained image we use harmonic inpainting. This quite simple technique gives approximate pixels of target region, using local surrounding pixels. Harmonic inpainting doesn't consider whole structure of picture. Therefore, now target region contains particular noise. Look at stage 3 on Figure 1.

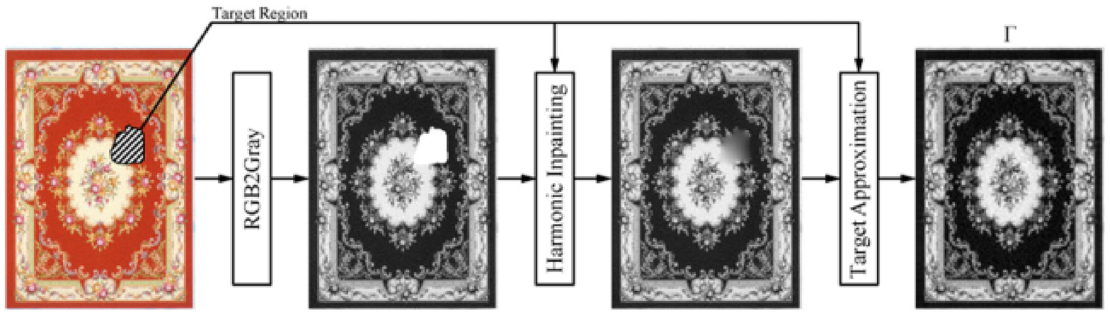


Figure 1: Procedure of computing approximation matrix.

This noise isn't the main detail and, for that reason, we may use SVD to get rid of it. To this end, we decompose the matrix in sum of matrices of rank 1. Matrices, representing general patterns of image, correspond to greater singular values (SV); matrices of details correspond to smaller SV. Therefore, take away matrices one by one starting from that with the smallest SV.

To reduce only noise and not to harm image, we use source region as the reference that shows if picture is not spoiled. While similarity of of initial and obtained after next removal source regions is greater than particular chosen threshold, we consider that only noise was removed. This procedure is shown on Figure 2. Once similarity becomes smaller than threshold, all next matrices are responsible for important part of image. To measure similarity, use SSIM.

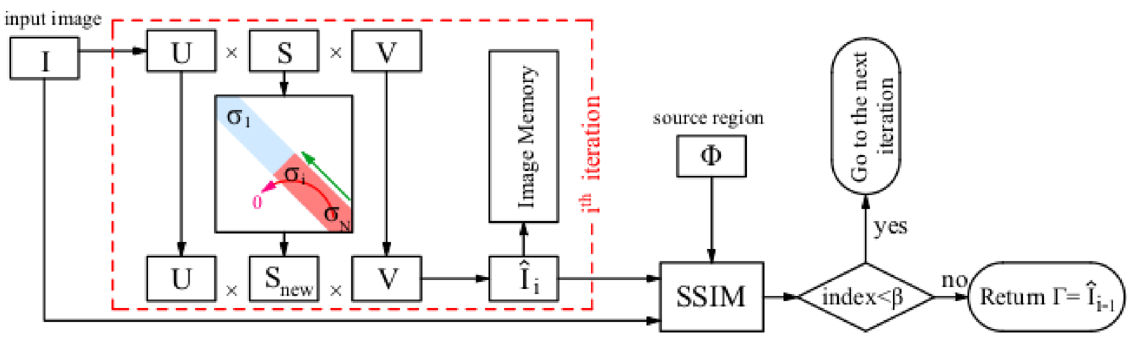


Figure 2: Use of SVD and SSIM to get approximation matrix.

After all, we obtain, maybe, a bit harmed matrix with much less noise in target region. It is similar to original, i.e. not damaged, grayscale picture. This matrix is called approximation matrix.

4.2 Inpainting

With an approximation matrix we move back to damaged color image. Let's denote target region as Ω and source region as Φ.

We specify $\delta\omega$ near bounds of Ω. Then split whole Φ in patches of rational size. After that we get patch that consists partially of $\delta\omega$ and partially of Φ. Compare it with all patches from Φ to find the most structurally similar, according with SSIM. Then we use pixels from the found patch to fill pixels from $\delta\Omega$ in the analyzed patch. Repeat this process until we fill whole $\delta\omega$, and then Ω in colored image.

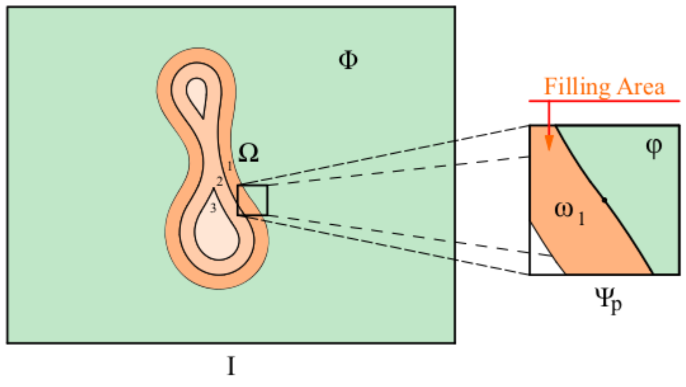


Figure 3: Filling of target region based on approximation matrix structure.

4.3 Multi-scaling

Also, we work with image on different scales, propagating info obtained on one scale to the next one. We subsample color picture (I) to get $x2$ and $x4$ times smaller images. Using these images, we form a pyramid of 3 levels: 0 in basis, 1 and 2. It is shown on Figure 4.

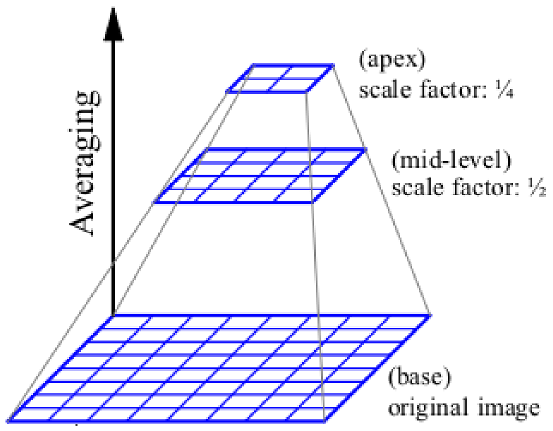


Figure 4: Pyramid of subsampled images.

We apply procedure of inpainting on 2 level ($L2$) image, enlarge inpainted picture $x2$ and extract target region. After that, we insert this piece instead of target region in $L1$ image. Again apply inpainting, extract and insert Ω in $L0$. The same procedure in $L0$, and we get the final result.

Scaling is used due to a few reasons. Firstly, it speeds up computations. Finding general structure of target region is quicker on subsampled picture. And then we just sligthly improve it. Secondly, at higher levels we capture spatial info of 16 neighbours of each pixel. For that reason, have more chances to find strcturally corresponding patches from source region.

5 Pros and cons

Algorithm has next pros:

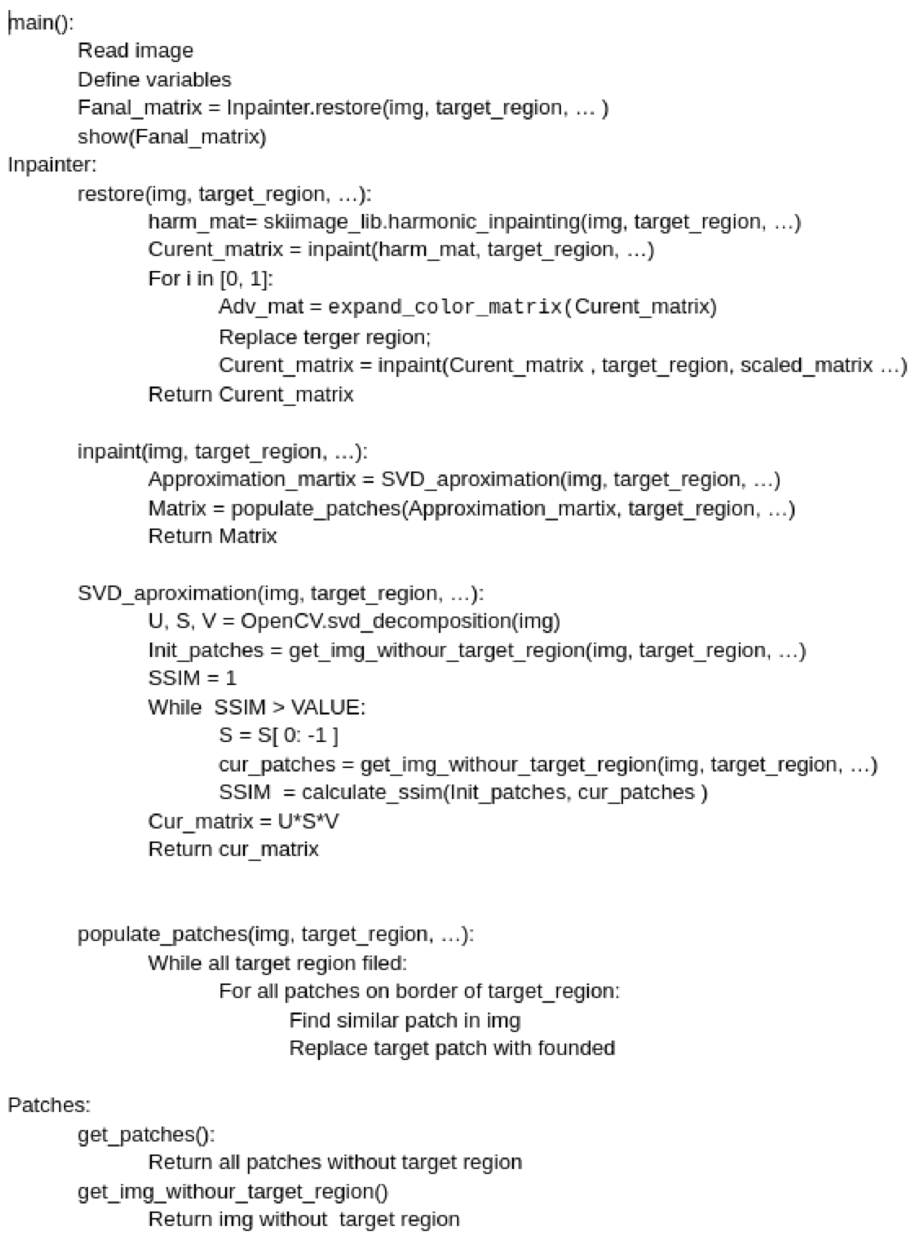
- SVD-based approximation analyze whole image to fill target region and in some cases it works very well
- Implementation is based on Linear Algebra concepts (SVD, etc.)

And a few cons:

- Simple harmonic inpainting is not covered by Linear Algebra course (we would try to dive deeper in it, but there are standard realization of it in python)
- This implementation does not perform well at every cases (could be dealt by taking source region from other pictures)
- There is a problem with assigning target/source regions with untrivial shape

6 Implementation pipeline

To implement our project we use Python. Especially such libraries like OpenCV to work with images and NumPY to work with matrices. We have three main classes: Inpainter, PatchExtractor, Scale and Main class where tall starting parameters set and program starts. In the beginning after we define target region we send this image matrix to Inpainter class. Where with harmonic inpainting function from external library we fill the target region and calculate SVD. Then by reducing singular values by 1 each time and calculating SSIM we find approximation matrix. Next whe populate target region with patches like described in 4.2. This how inpainting function works. The last part of algorithm is scaling, we use inpainting on smaller image, then take a target region, rescale it, paste in bigger image and use inpainting again. Down below there is a pseudocode of our program.



```
main():
    Read image
    Define variables
    Fanal_matrix = Inpainter.restore(img, target_region, ... )
    show(Fanal_matrix)
Inpainter:
    restore(img, target_region, ...):
        harm_mat= skiimage_lib.harmonic_inpainting(img, target_region, ...)
        Curent_matrix = inpaint(harm_mat, target_region, ...)
        For i in [0, 1]:
            Adv_mat = expand_color_matrix(Curent_matrix)
            Replace terger region;
            Curent_matrix = inpaint(Curent_matrix , target_region, scaled_matrix ...)
        Return Curent_matrix

    inpaint(img, target_region, ...):
        Approximation_martix = SVD_aproximation(img, target_region, ...)
        Matrix = populate_patches(Approximation_martix, target_region, ...)
        Return Matrix

    SVD_aproximation(img, target_region, ...):
        U, S, V = OpenCV.svd_decomposition(img)
        Init_patches = get_img_withour_target_region(img, target_region, ...)
        SSIM = 1
        While  SSIM > VALUE:
            S = S[ 0: -1 ]
            cur_patches = get_img_withour_target_region(img, target_region, ...)
            SSIM  = calculate_ssim(Init_patches, cur_patches )
        Cur_matrix = U*S*V
        Return cur_matrix


    populate_patches(img, target_region, ...):
        While all target region filed:
            For all patches on border of target_region:
                Find similar patch in img
                Replace target patch with founded

Patches:
    get_patches():
        Return all patches without target region
    get_img_withour_target_region()
        Return img without  target region
```

Figure 5: Pseudocode.

7 Testing

To test our algorithm, we needed to choose appropriate params:

- β. Actually, the best value of this parameter depends on the quality of the image. For example, the best value for the image of cover is 0.9. Whereas for butterfly - 0.3. So, we picked 3 different enough values from range $[0, 1]$ and chosen the best one from them, taking larger values for low quality image, and lower values for higher quality.
- *patchsize*. After lots of experiments, we figured out that optimal patch size is $6x6$ pixels. On the one hand, it isn't computed too long. On the other hand, it gives well result.

There are some example outputs of our algorithm. As we can see, it better works with symmetric images, and images that have many similar details.

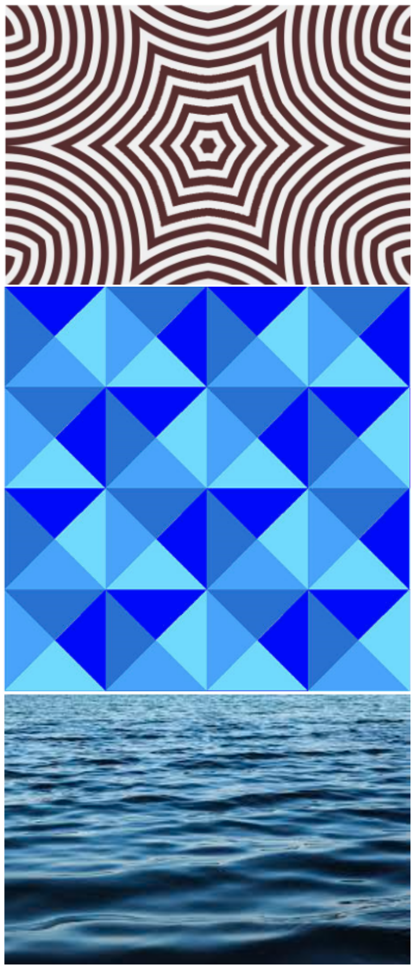


Figure 6: Initial

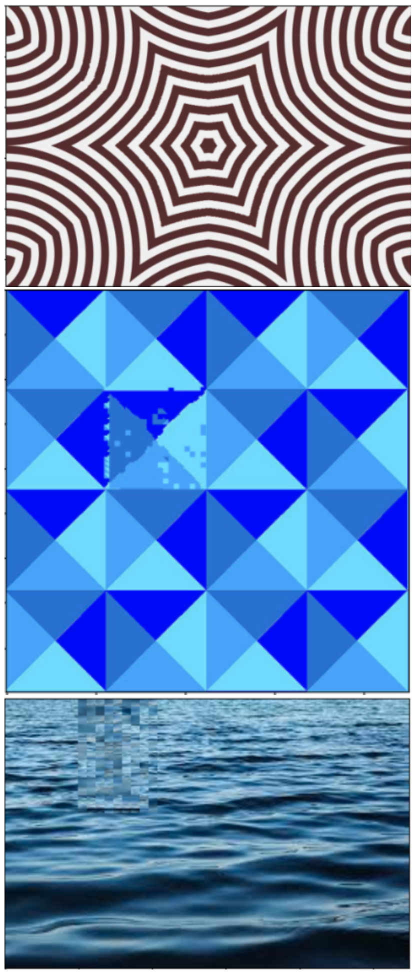


Figure 7: Damaged and restored

8 Troubles

We have also troubles during implementation.

Firstly, target region mask shifted from its position. After investigation, we fixed it.

Secondly, inappropriate patches were chosen in picture with birds on Figure 8, though on approximation matrix were no bugs. All in all, we understood that need to take into consideration colors of pixels surrounding target region. Didn't implement a fix yet, but plan to do it in the near future.

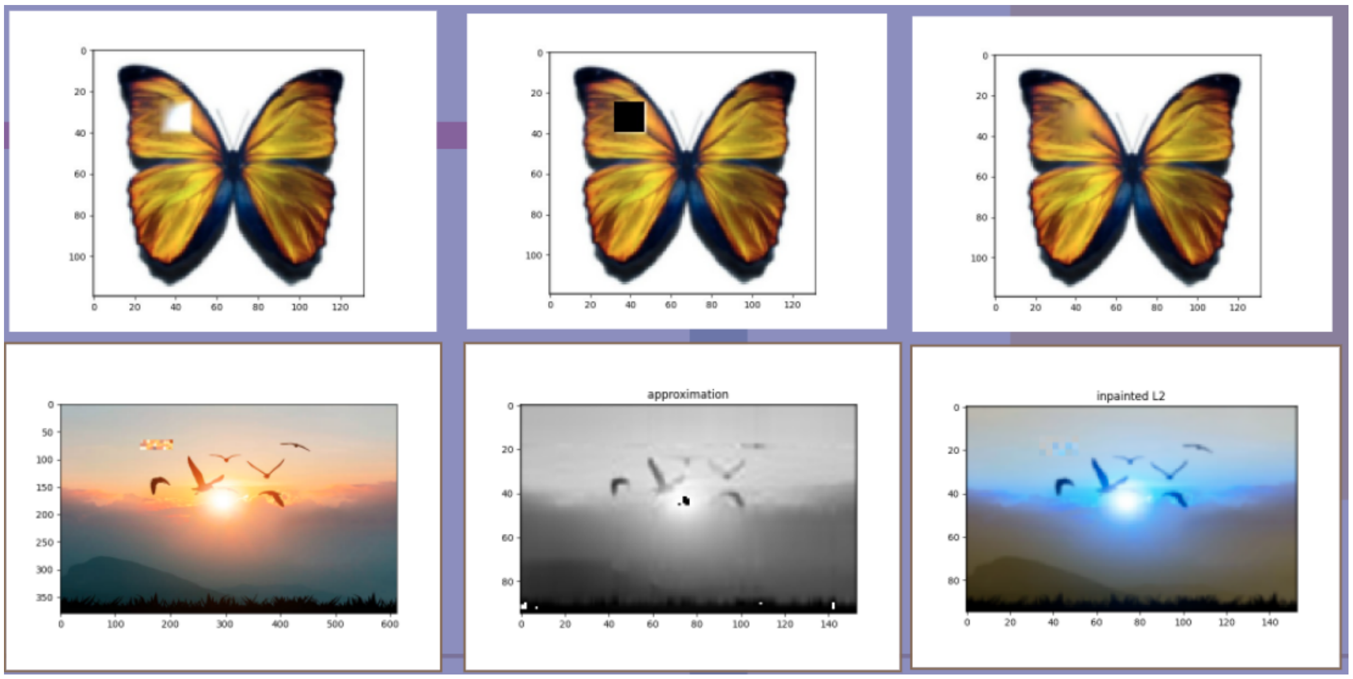


Figure 8: Some bugs.

9 References

Whole work is mostly based on related work:

[1] Exemplar-based image inpainting using svd-based approximation matrix and multi-scale analysis. Vahid K. Alilou, Farzin Yaghmaee

Github repository with codes: https://github.com/viktorpovazhuk/crystal_eye