Introduction to Digital Imaging Project Report

Copy-move forgery detection based on PatchMatch

Haozhe Sun

Table of contents

1. Introduction *2*
2. Algorithm *3*
2.1 Image forgery detection pipeline 3
2.2 Modified-PatchMatch 3
2.3 Discussion about the proposed algorithm 5
2.4 Discussion about the image forgery detection pipeline 5
3. Implementation *7*
4. Experiments *8*
4.1 Discussion 8
4.2 Successful examples 10
4.3 Failed examples 13
4.4 Limitations 16
5. References *17*

1. Introduction

In the context of project of MVA Introduction to Digital imaging, I have chosen the project Copy-move forgery detection based on PatchMatch [1]. I have chosen this project because I want to get the first experience in manipulating image pixels rather than studying and tuning the Deep Learning models. I want to get an intuition and experience with the classical image processing techniques. Furthermore, I always heard about patch-based method, I think this project could be a good opportunity to get to know it.

My project is related to the image forensics, i.e. image edition forgery detection. In particular, [1] is focused on the Copy-Move case, which is a special case of image forgery. The authors presented an algorithm adapted from the classical PatchMatch algorithm which aims to increase the robustness. I have implemented this algorithm from scratch and showed some qualitative results.

2. Algorithm

In this section, I will briefly present the algorithm and make some remarks on the processing pipeline.

2.1 Image forgery detection pipeline

The basic idea behind PatchMatch algorithm is that identical or similar regions in the image are discovered based on matching. This procedure mainly consists of 3 steps:

- *Featuring: suitable features are associated with all pixels or with a limited set of key points*

- *Matching: for each pixel of interest, the best matching is located based on the associated features*

- *Post-processing: the displacement field is filtered and processed to detect actual copy-moved regions.*

This paper [1] mainly talks about the second step Matching, where they presented their modified-PatchMatch algorithm.

2.2 Modified-PatchMatch

To present the modified-PatchMatch algorithm, I should introduce the basic version of PatchMatch algorithm.

The PatchMatch algorithm is a fast randomized algorithm which could find a dense approximate nearest neighbor matches, that we call NNF, of an image or two images. An NNF is just a 2D mapping where each patch is associated with a vector, that indicates its nearest neighbor patch in the same image or in another target image. All our copy-move detection will be based on this NNF. However, the naïve approach to calculate the exact NNF is not acceptable because of high time complexity. PatchMatch algorithm has been proposed to tackle this problem by iteratively calculate an approximation of the exact NNF.

Here is the pseudo code of the basic PatchMatch algorithm:

- *Random initialization of NNF such that for each point z, the displaced $z + nnf(z) \sim Uniform(\Omega)$, where nnf can be seen as a mapping from 2D points to a 2D offset, Ω is the support of the image.*

- *For a certain number of iterations, do:*

    - *Propagation: the image is raster scanned top-down or left-to-right according to the parity of iteration to avoid biases.*
      For each pixel z,

      $$nnf(z) = arg \min_{\delta \in \{nnf(z), nnf(z^u), nnf(z^l)\}} Distance(P(z), P(z+ \delta))$$

    - *Random search: the image is raster scanned in the same way as before. Candidates become:*

      $$nnf(z) = arg \min_{\delta \in \{f_0, f_1, f_2, \ldots\ldots\}} Distance(P(z), P(z+ \delta))$$

      $$f_i = nnf(z) + 2^{i-1} R_i$$

      where R_i represents a bi-dimensional random variable uniform in $\{-1,0,1\} \times \{-1,0,1\}$ excluding $(0,0)$. P represents the patch which is represented by the pixel z. The choice of distance measure is not fixed a priori. z^u and z^l means upper neighbor pixel and left neighbor pixel.

The modified-PatchMatch consists in one modification in the Propagation phase. Instead of considering the candidate set:

$$\{nnf(z), \qquad nnf(z^u), \qquad nnf(z^l)\}$$

we consider the candidate set:

$$\{nnf(z), \qquad nnf(z^u) + \Delta\, nnf(z^u), \qquad nnf(z^l) + \Delta\, nnf(z^l)\}$$

where

$$\Delta\, nnf(z^u) = nnf(z^u) - nnf(z^{uu})$$

z^{uu} represents the pixel above z^u

2.3 Discussion about the proposed algorithm

This modification is proposed aimed to deal with rotations without impairing the performance in their absence. In fact, the propagation candidate set consists of zero-order predictors, which are effective only in constant regions, i.e. NNF is uniform. The modified propagation candidate set consists of first-order predictors that corresponds to a linearly varying NNF. The rotated copy-moves correspond to a linearly varying NNF so we say that the modified version of PatchMatch deals with piece-wise linear NNF, whereas the basic version of PatchMatch only deals with piece-wise constant NNF.

However, to deal with resizing, the authors of the paper only rely on the intrinsic robustness of the algorithm for scales close to 1.

One important remark on PatchMatch based algorithms is the fundamental hypothesis that the true NNF is mostly regular, composed of a relatively small number of regions with the same displacement.

2.4 Discussion about the image forgery detection pipeline

The paper [1] is mainly focused on the modified PatchMatch algorithm, which represents the matching phase in the detection pipeline. As for the featuring phase and post-processing phase, the authors just mentioned them quickly without much explication.

[1] mainly talked about 2 choices of features that we can use. The first one is RGB values, in this case the associated distance measure could be the L1 norm. The second one is Zernike moments, which turn out to be rotation invariant. The authors mainly did the experiments with Zernike moments; however, I chose to implement the algorithm with RGB values.

It is my first time to discover the theory of image moments. According to my research, the image moments are used to represent to shape or a contour, it does a kind of 2D integration of each point presented in the shape with respect to the centroid of this shape. What I understood is that I could make use of this kind of descriptor only if I have a binary 2D image that could be seen as a contour. However, in the context of [1], I deal with image patches that are characterized by intensity in three color channels in every location. I have the choice to do the thresholding to transform a such intensity patch into a binary image, but I do not see what kind of criterion could be used to let the filtering make sense for every patch. As a consequence, I chose to pick RGB values, that are more intuitive. The distance measure is thus the L1 norm distance.

The post-processing method used in [1] consists of filtering and Same Affine Transformation Selection (SATS) [2], which consists in collecting in groups the surviving matches so that matches in the same group have the same transformation pattern. However, this method is not well explained.

In my implementation, the post-processing phase consists of:

- NNF offsets less than a certain threshold are filtered out

- A binary map is generated by the normalized sum of NNF and the associated distance field map. This distance field map is processed so that the values more than a certain quantile are set to the maximum value of distance field, this processing is to filter out the locations whose corresponding distance is too big. Then the normalized sum of NNF and the processed distance field map is processed by a thresholding of a certain quantile.

The binary map obtained can be seen as an indicator of the presence of matched patch. Small value in this map will have zero value, which means a possible match. However, the determination of presence of match should also take into account the L1 norm of NNF map and the distance field map, because sometimes the binary map can be not that informative if we do not choose the parameters properly.

3. Implementation

I have implemented the modified-PatchMatch algorithm from scratch in Python. Some visualization and post-processing functionalities are also provided.

In the implementation, each patch is represented by its upper left pixel, called representative pixel. The height and width of the NNF are determined by the image size and the patch size.

Besides the NNF, I also calculate a distance field map which aims to represent the distance between the matched patches, which could be useful when we try to identify real matches from match candidates.

We should take care of zero values in the NNF, because the zero values in the NNF may quickly propagate to all the field, which leads to a useless NNF. The zero values in the NNF are mostly introduced at the random initialization phase. This can be tackled by a slight post-processing of initialization phase, where we slightly increment or decrement the zero values without changing too much the uniform distribution nature. This is even not necessary, because it will be sufficient to stop the propagation of zero values in the iteration loops. The zero-value elimination in the iteration loops is useful because zero values could also be introduced in the propagation phase because of the presence of subtraction operations in the first-order predictors.

I also came across on the internet one strange trick that is believed to be practiced by the author of the basic PatchMatch, that I call "zero border" trick. This trick consists in setting to zero in the initialization of the border (with a certain width) of NNF. The justification is that the patches in the border zone is considered to be similar to no other patch in the image. I implemented this trick as an option and I tested its performance before I introduce the function of elimination of zero values in iterative loops (if not, zero borders will have no effects). However, it turns out that the zero border with a border size of 1 does not give better results than without according to some visual tests. A zero border with a border size of 2 or bigger will ruin all the NNF as zero values will be propagated everywhere. I kept the implementation of this trick as an option in the code and turned it off by default, which allows further study without impairing the performance of the algorithm.

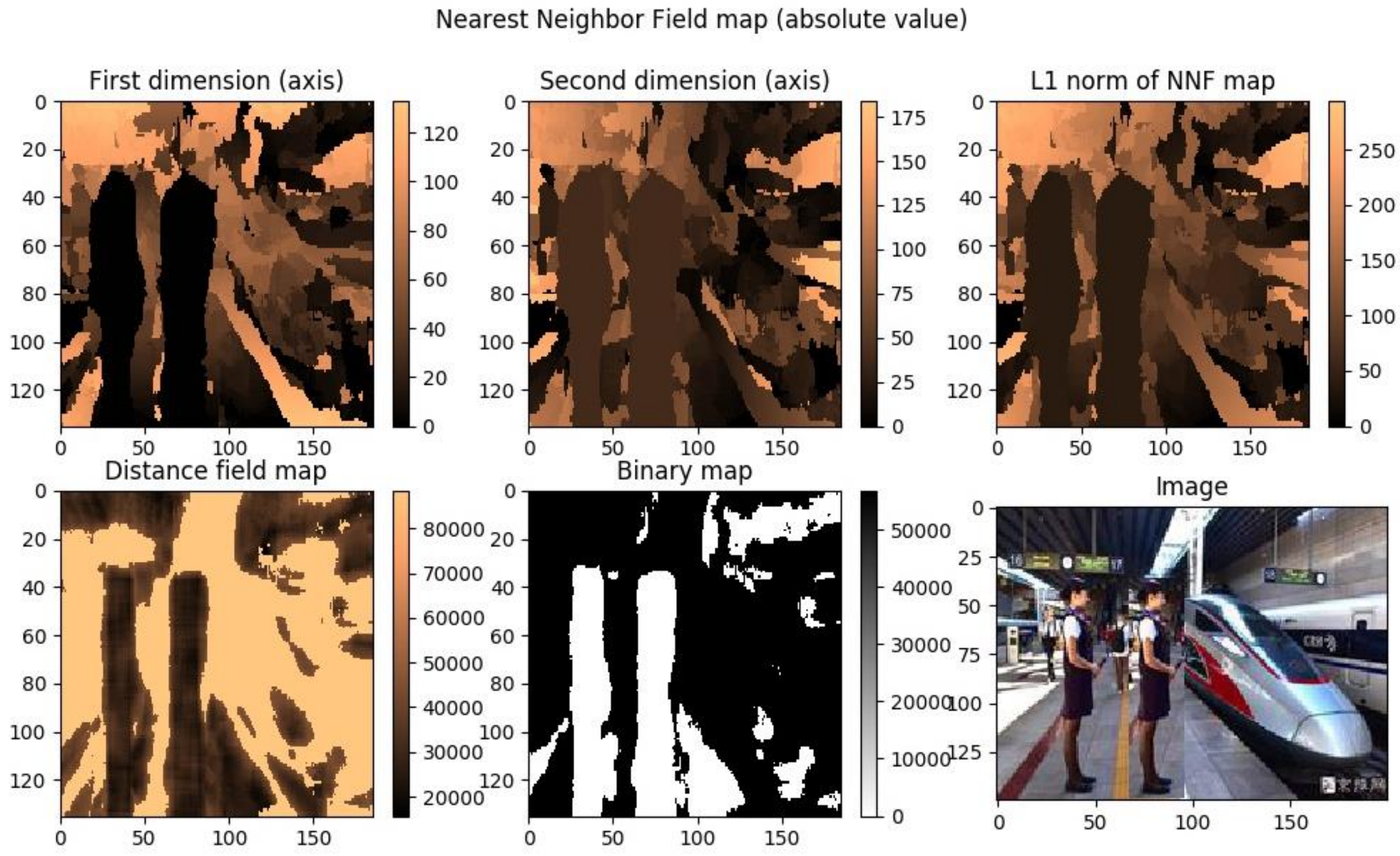
4. Experiments

4.1 Discussion

The Python implementation of the modified-PatchMatch algorithm is relatively slow, so I chose to format the input images to be of size (200, 150). With input images of this size, the whole computation time of modified-PatchMatch, post-processing, and visualization for one image with 5 iterations takes about 115 seconds.

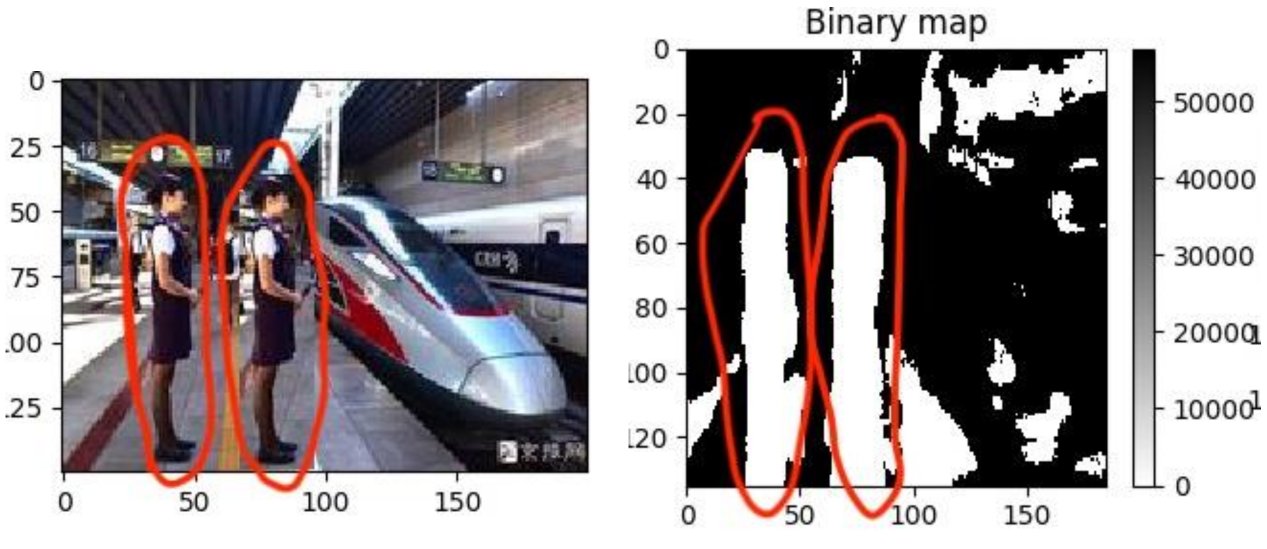
The number of iterations is fixed to 5 by default. This is a well-known empirical result of PatchMatch, the algorithm of PatchMatch quickly converges within a very limited number of iterations, no more than 5. This is also true for the modified-PatchMatch algorithm. According to my experiments, I find out that the algorithm could converge even with 2 or 3 iterations. To keep the result reliable, we keep the default number of iterations to be 5 at the expense of computation time.

Here is an example of the visualization of the output. The 3 figures of the first line represent the NNF, respectively the first axis, the second axis, and the L1 norm of the NNF. In the second line, the processed distance field map and the binary map are presented. All of these figures



are plotted in absolute value. The input image is visualized in the lower right corner.

We clearly recognize that the copy-moved part, i.e. the two standing girls beside the train are detected as matched patch by the binary map, distance field map and NNF, even if there is much noise around. In the first figure, we confirm that the vertical offset (first axis) is almost zero. In the second figure, we confirm that the horizontal offset (second axis) is non-zero but has small value.



This is a successful case. However, I should say that this kind of situation is not that frequent. To get a good result, the choice of parameters is sometimes critical. These parameters mainly include the patch size, the threshold of NNF filtering and the threshold of binary map. The above result has been achieved with 16 as patch size, 10 as NNF threshold and 0.3 as binary map threshold.

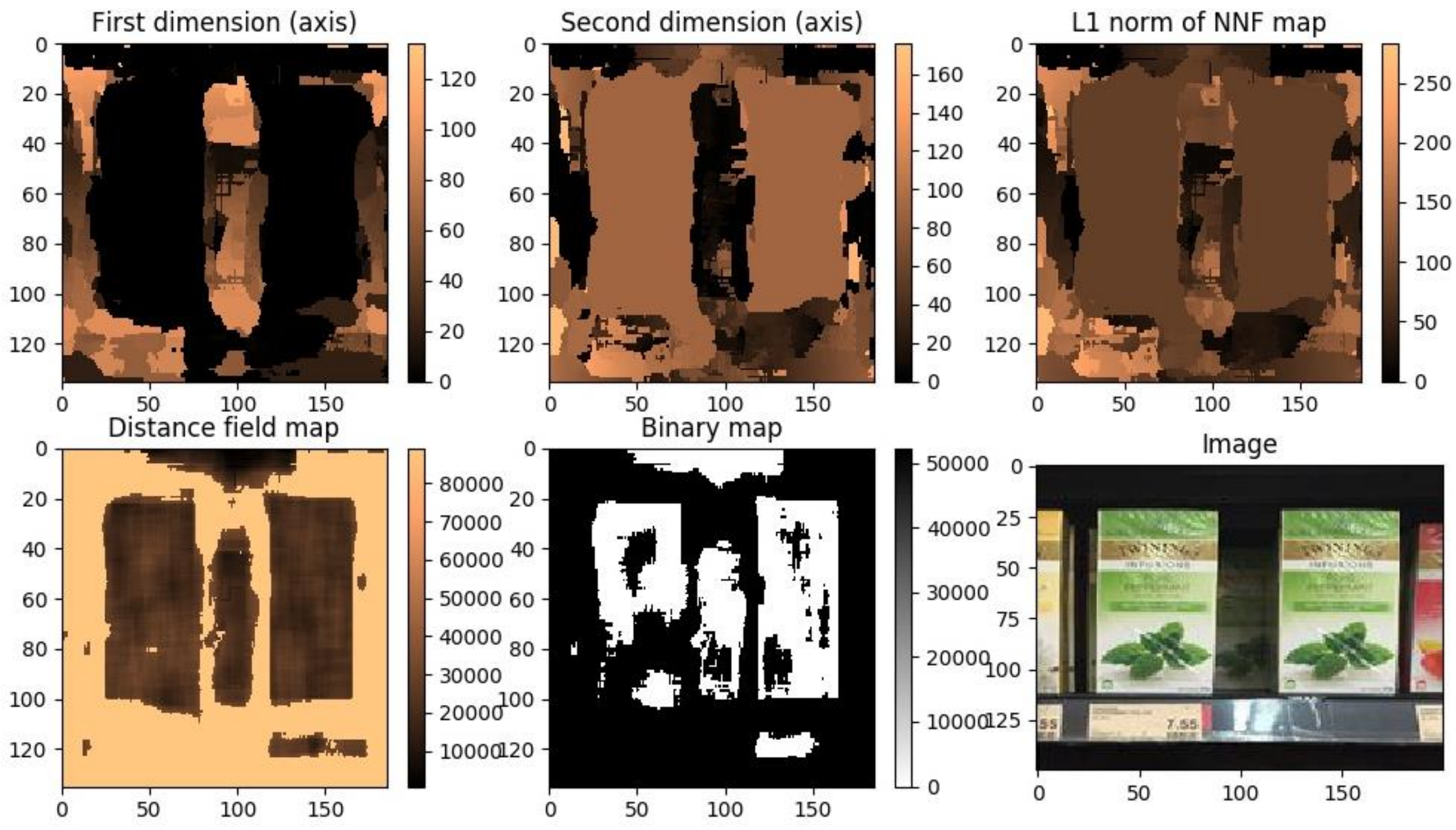
The quality of results sometimes depends on tuning of parameters, the parameters depend on the size and the content of the images, which means that it is difficult to find the best parameters that are suitable for all images. However, the above setting is kept as default parameters because it turns out that they can be used in many cases. Then it seems that this algorithm is not that robust. Even with the same parameters, we can get different results, sometimes the copy-moved parts are not correctly identified by the algorithm. Multiple running of the algorithm could tackle this problem.

The noise presented in the detected candidate matched patches are usually due to the presence of homogeneous background. For example, if the background of the image are black/white or equipped with some similar textures. The algorithm has trends to match them and then the real match will be flooded because the detection is based on a certain quantile. That is why the tuning of parameters patch size, nnf threshold and binary map threshold is crucial.

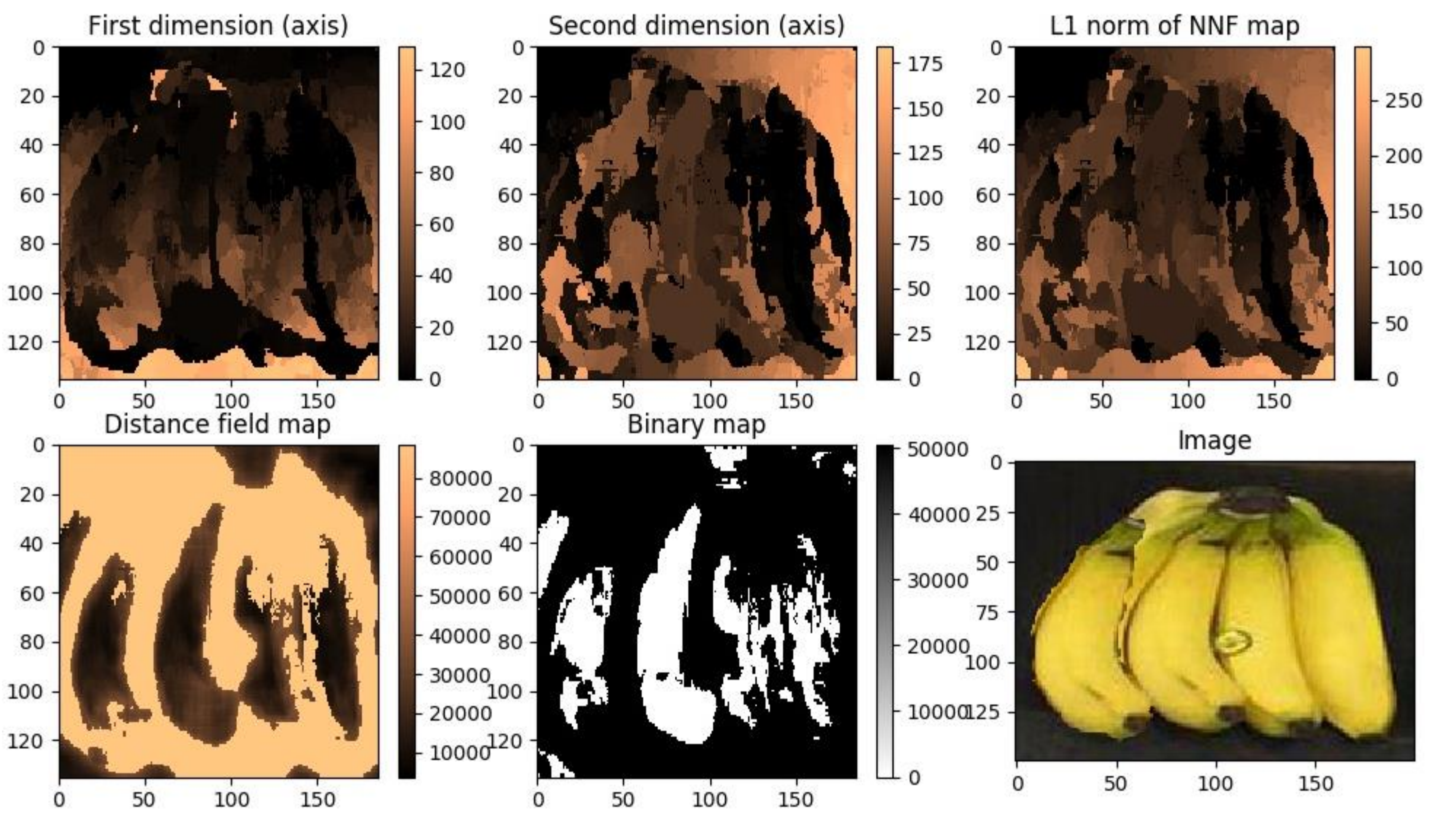
The test has been carried out on a data set that I collected myself, some of the copy-move operations are manually made using GIMP 2.0, an open source image manipulation software. A small portion of images are taken from the Copy-Move Forgery Database with Similar but Genuine Objects (COVERAGE) [3]. Each new image is formatted to the same size before the other processing.

4.2 Successful examples

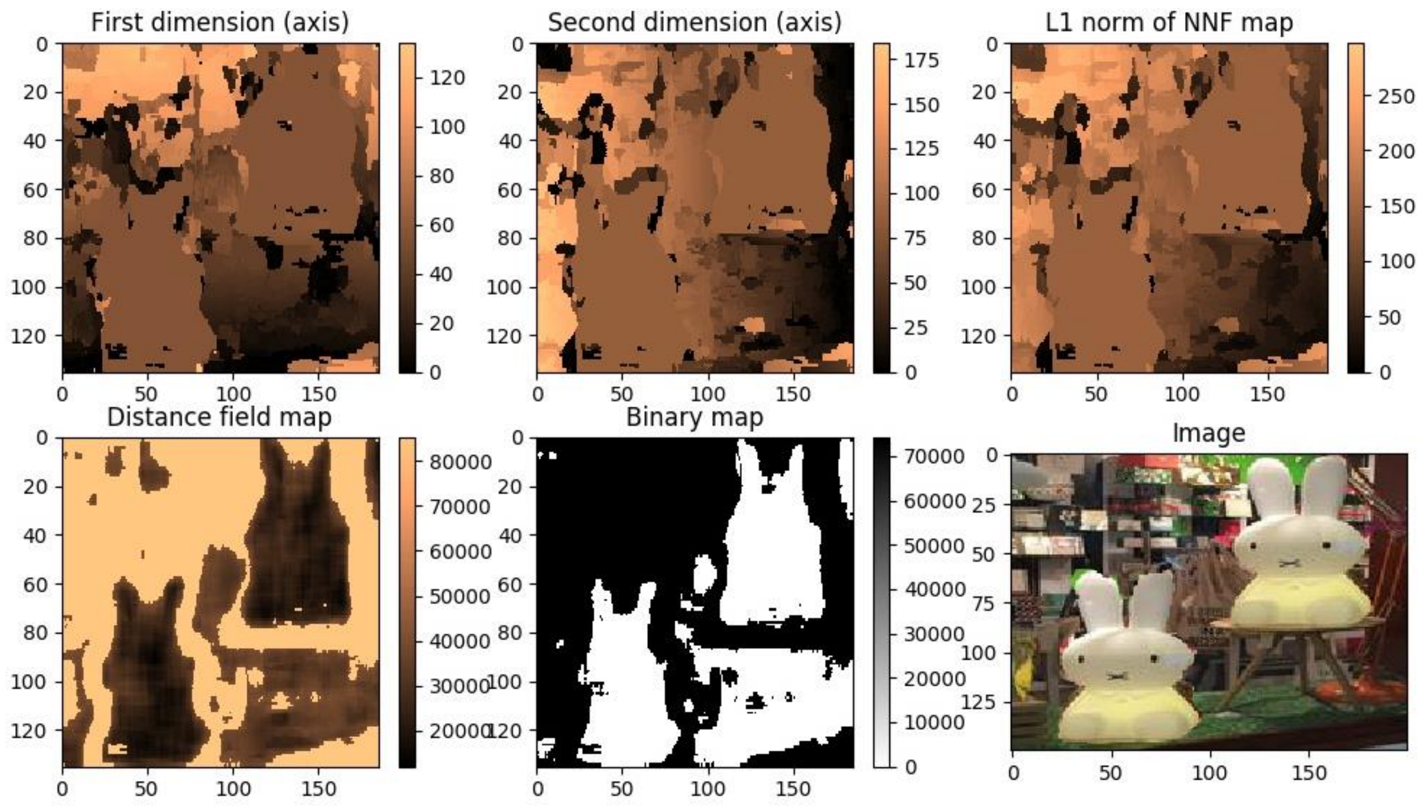
Some other relatively successful results are shown here:



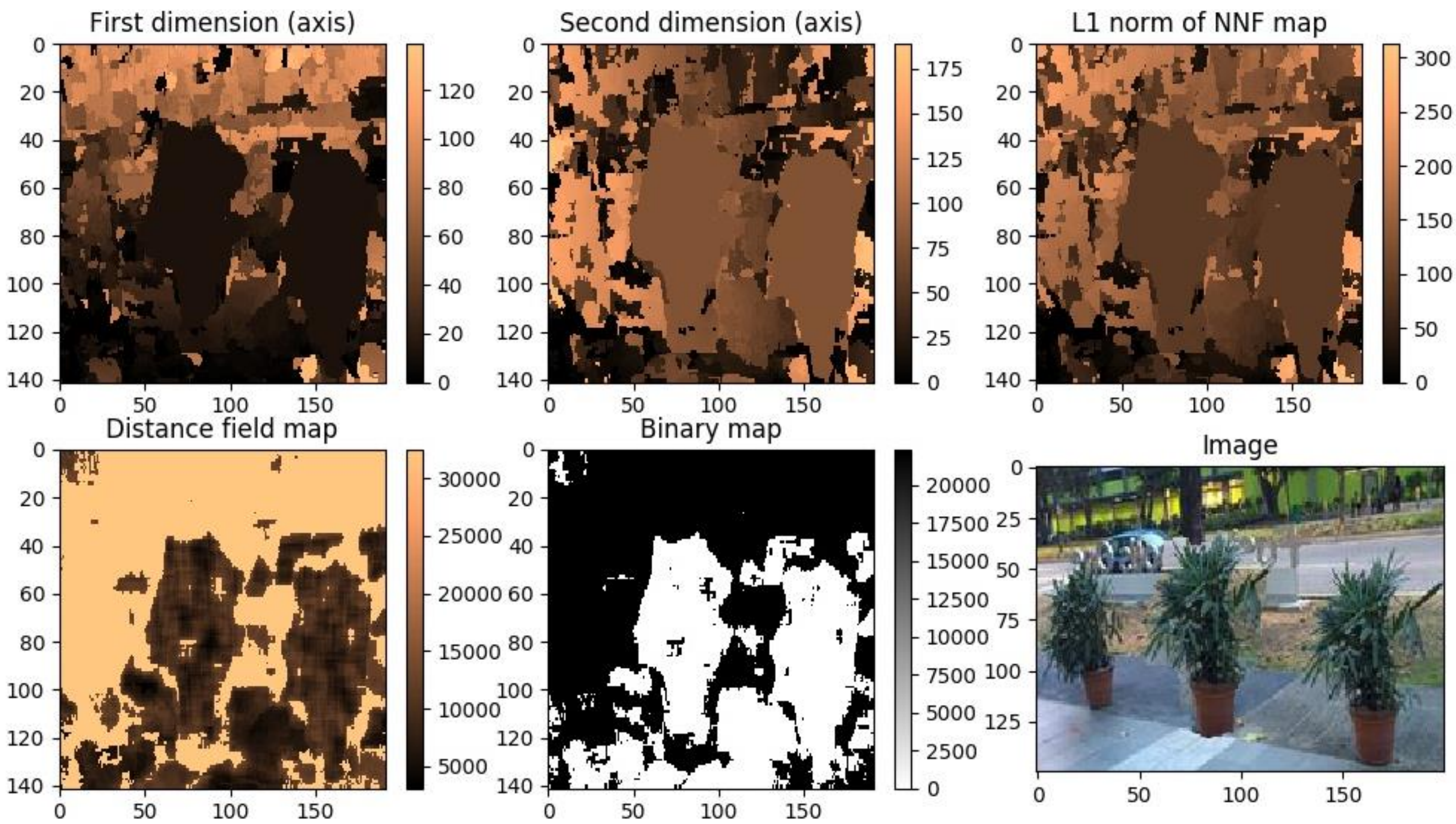
The above result is obtained with batch size = 16, NNF threshold = 10, binary map threshold = 0.3.



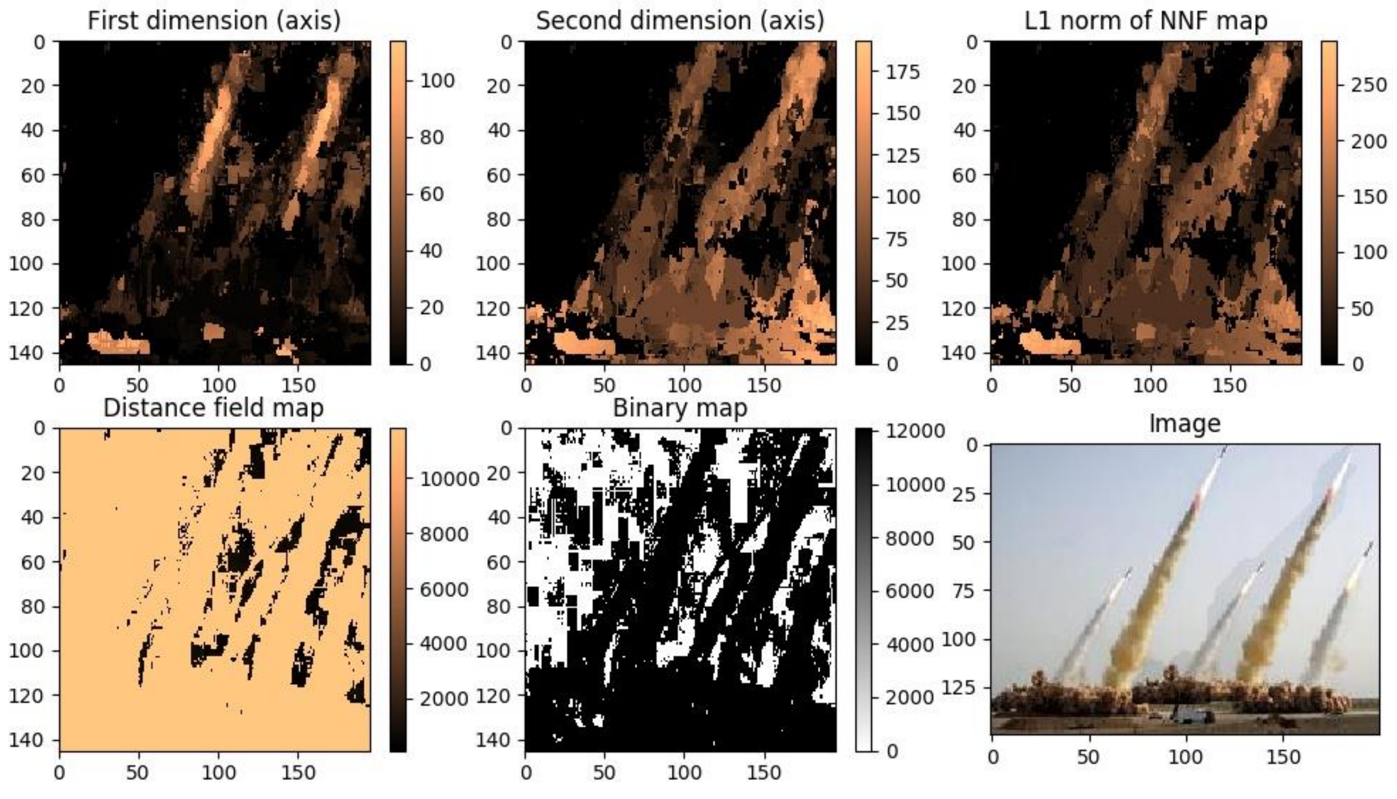
The above result is obtained with batch size = 16, NNF threshold = 14, binary map threshold = 0.25.



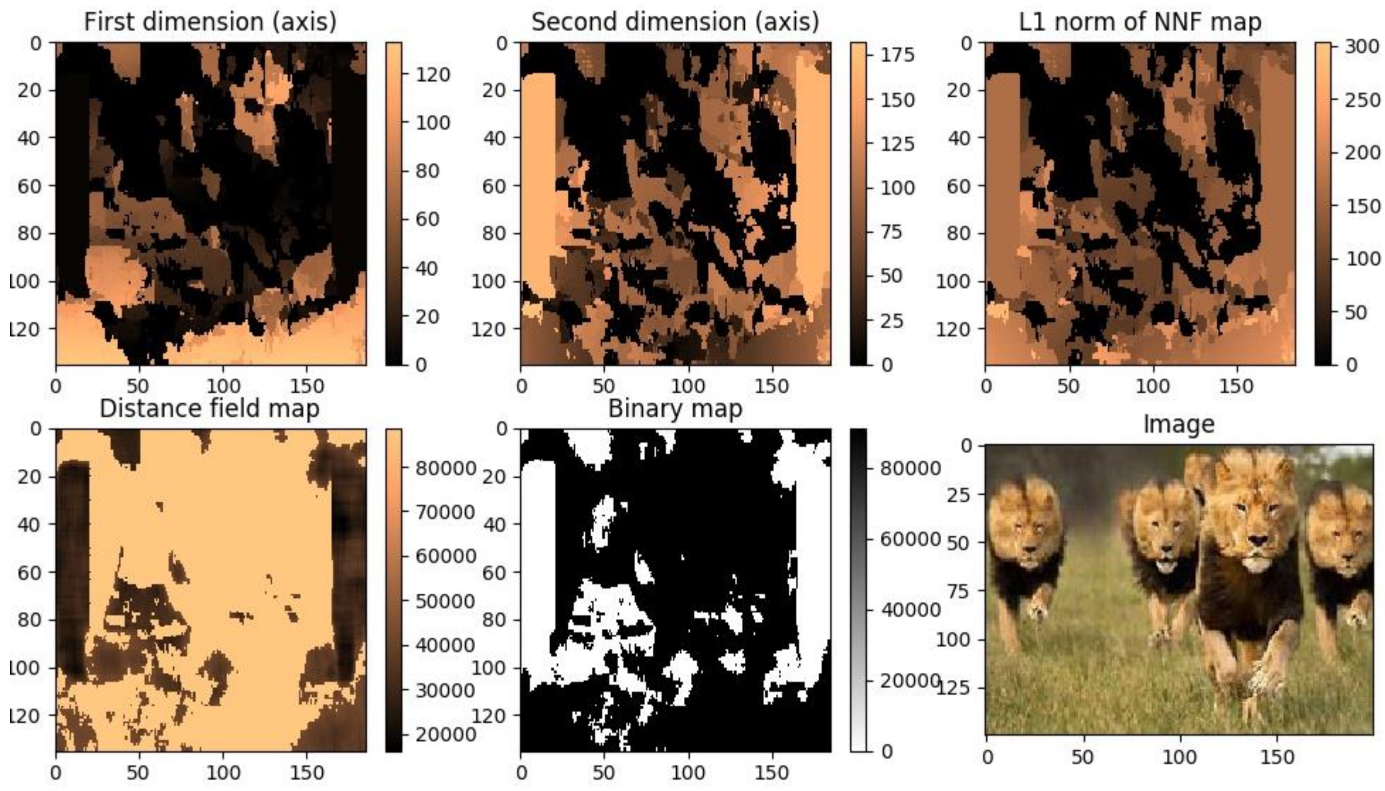
The above result is obtained with batch size = 16, NNF threshold = 10, binary map threshold = 0.4.



The above result is obtained with batch size = 10, NNF threshold = 10, binary map threshold = 0.4.



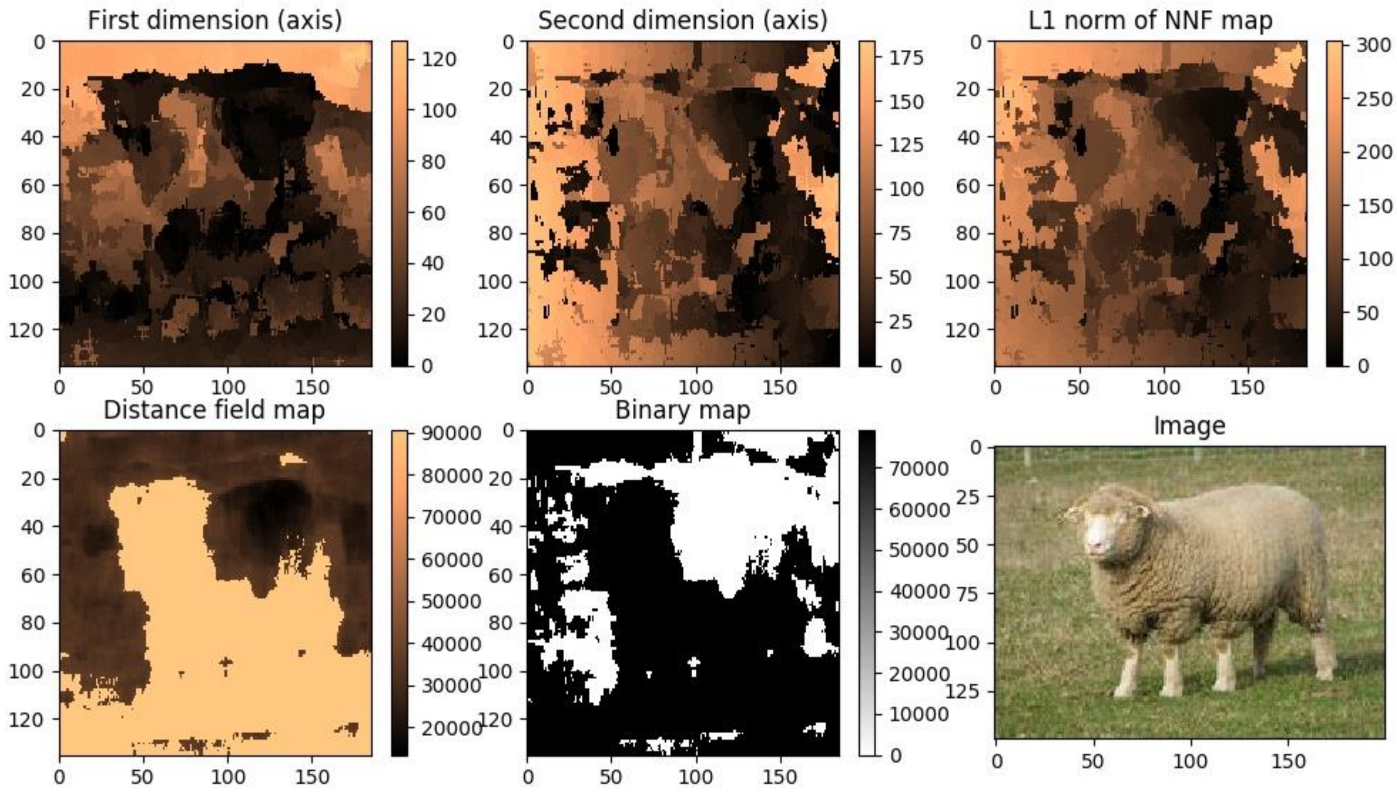
The above result is obtained with batch size = 6, NNF threshold = 40, binary map threshold = 0.4.



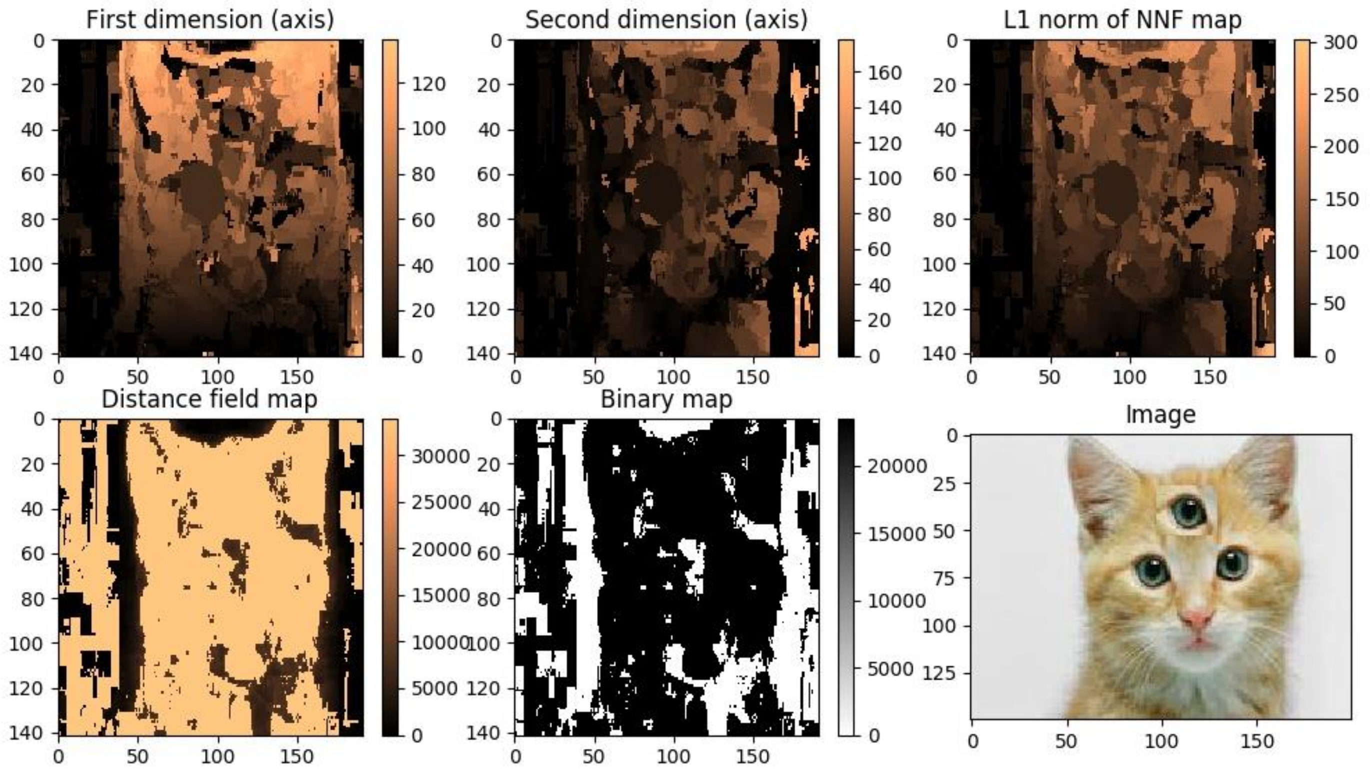
The above result is obtained with batch size = 16, NNF threshold = 80, binary map threshold = 0.3.

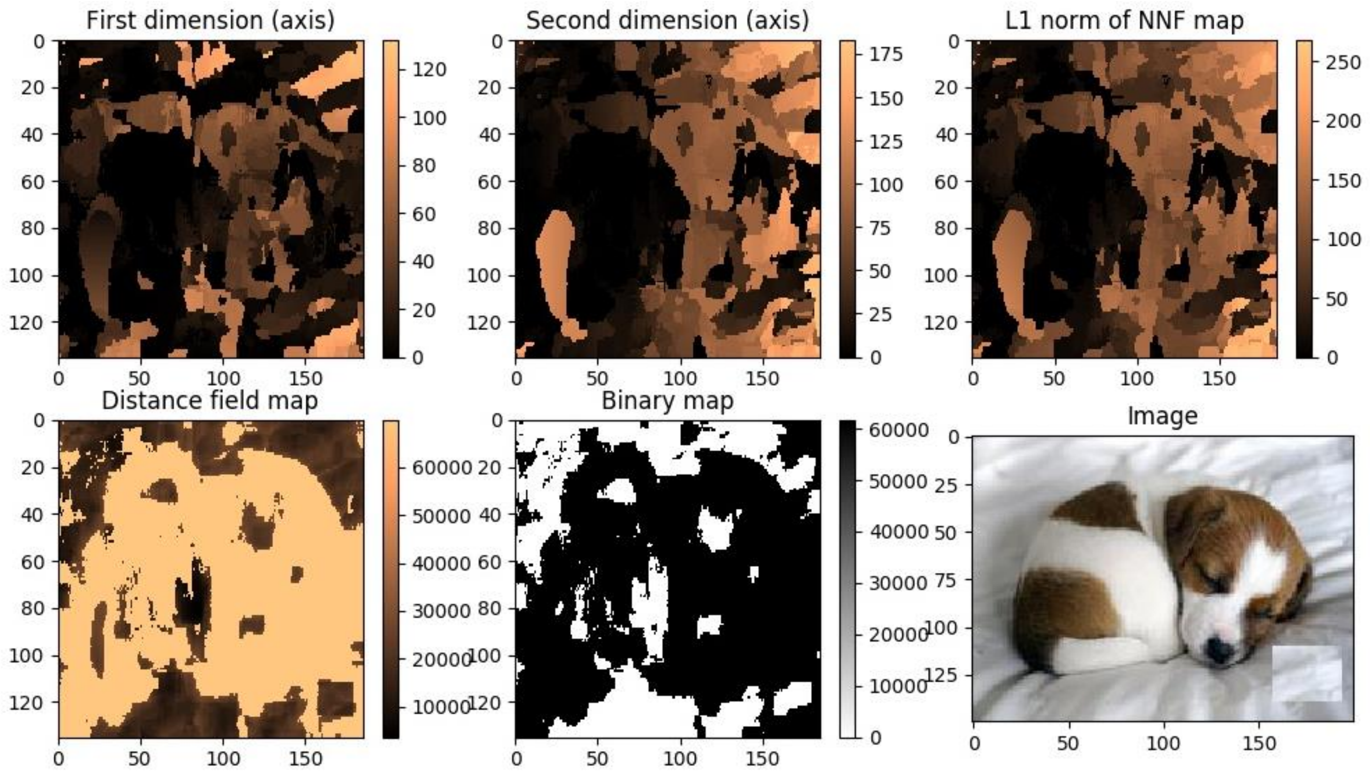
4.3 Failed examples

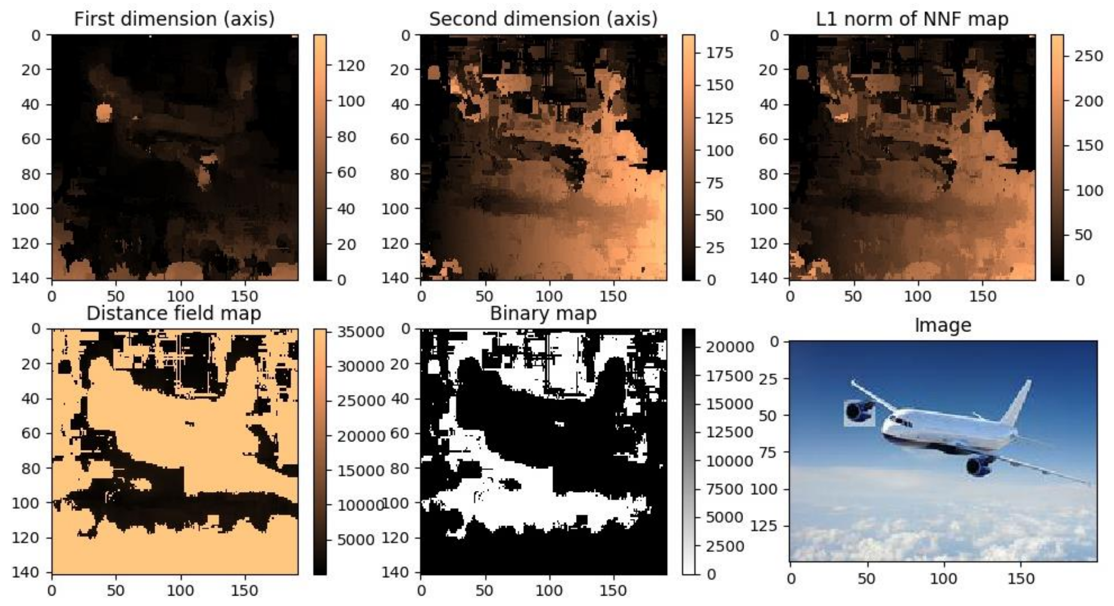
Now we show some failed cases:

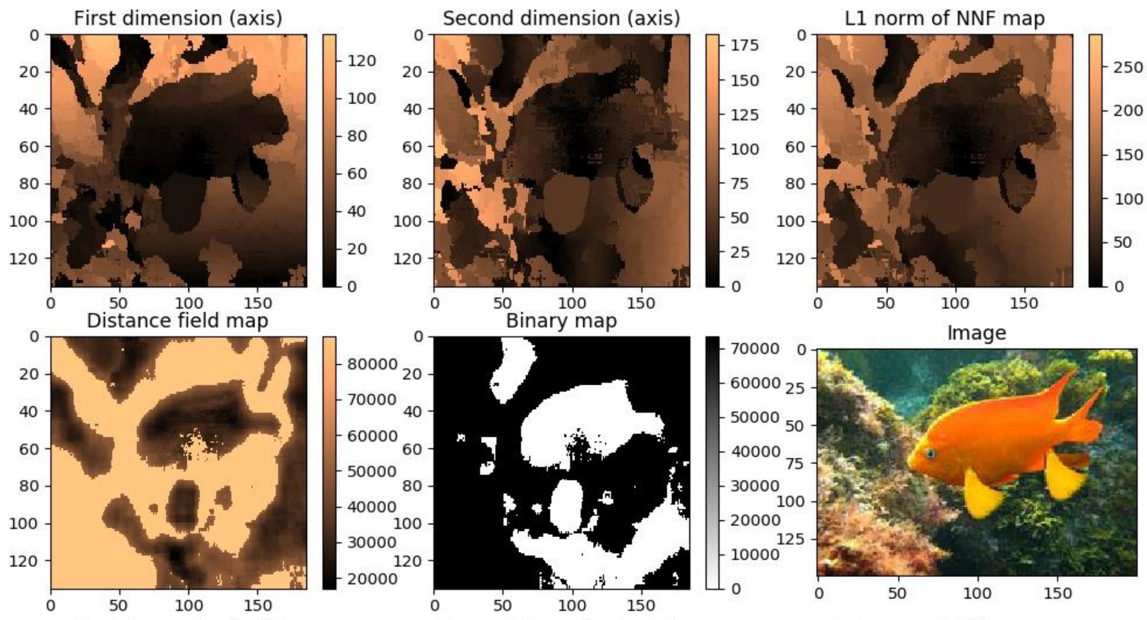


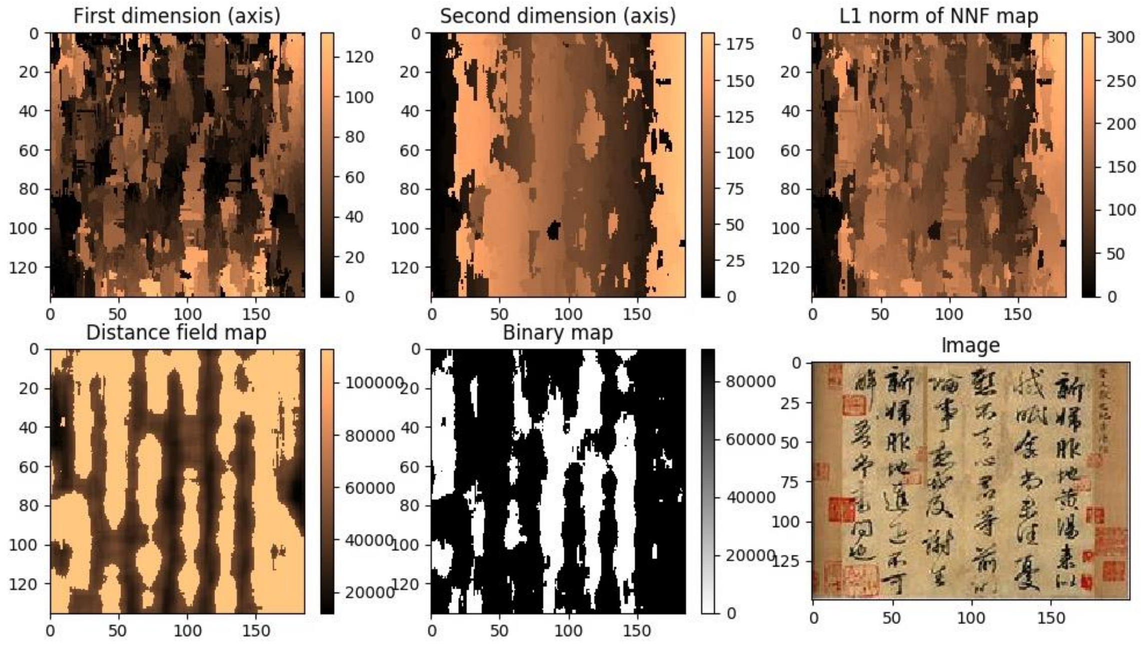
In this case, the algorithm is not at all capable to capture the copy-moved part. The main reason behind this is the presence of homogeneous texture in the background and on the wool of the sheep. This floods the copy-moved third leg and thus prevents the algorithm to detect it. There are many failed cases, the most frequent reason is the homogeneousness in the images.






First dimension (axis)
Second dimension (axis)
L1 norm of NNF map
Distance field map
Binary map
Image


First dimension (axis)
Second dimension (axis)
L1 norm of NNF map
Distance field map
Binary map
Image


First dimension (axis)
Second dimension (axis)
L1 norm of NNF map
Distance field map
Binary map
Image

4.4 Limitations

To solve the problem caused by the homogeneousness in the images, I came up with an idea similar to the Nearest Neighbor procedure. The idea is that for each patch, we do not only find its closest neighbor but also its first two or three closest neighbors. Then we look at the similarity or distance between these candidates with respect to the reference patch, if the closest neighbor is not far better than the other neighbors, it is probable that this is not a good match and is just due to the homogeneousness of the image. In practice, the implementation of the variant could be done by associating with each representative pixel an ordered list of candidate offsets, instead of one candidate offset in the phases of propagation and random search. In the end we could filter out the NNF elements whose first match is not far better than its other matches. This naïve idea may increase the computation time.

Then, for all the examples including the successful ones, the success of experiments depends on the choice of parameters and the algorithm implemented is not that stable. Furthermore, the successful cases are usually the cases where the copy-move operations is not equipped with rotations. In my opinion, this is partly due to the property of the selected feature, i.e. RGB values.

5. References

[1] D. Cozzolino, G. Poggi, and L. Verdoliva, “Copy-move forgery detection based on PatchMatch,” in Proc. IEEE Int. Conf. Image Process., Oct. 2014, pp. 5312–5316.

[2] V. Christlein, C. Riess and E. Angelopoulou, “On rotation in- variance in copy-move forgery detection,” IEEE International Workshop on Information Forensics and Security (WIFS), pp. 1–6, 2010.

[3] B. Wen, Y. Zhu, R. Subramanian, T. Ng, X. Shen, and S. Winkler, "COVERAGE - A Novel Database for Copy-Move Forgery Detection," in Proc. IEEE Int. Conf. Image Processing (ICIP), 2016.