Principled Method for Image Retargeting Using the “Patch-Based Bidirectional Similarity” Measure

ANTONIO JAVIER SAMANIEGO JURADO, University of California Santa Barbara

I propose an implementation of a principled method for digital image retargeting based on the “Patch-Based Bidirectional Similarity” measure. The objective is to change the scale or ratio of an image significantly taking into consideration its content (content-aware retargeting). The implementation includes two parts: the first consists in a method for image retargeting based on a course-to-fine algorithm as proposed by publication Simakov et al. CVPR’08. The second implements a nearest neighbor search function based on both the naïve search method and the “PatchMatch” algorithm, as described in Barnes et al. SIGGRAPH’09, and a voting function. Everything is programmed in Matlab© code. I show the final results and their interpretation, as well as the algorithm to achieve them.

1 INTRODUCTION

Image retargeting and summarization is a very useful way to support applications such as thumbnail generation, video summarization or displaying videos on different screen sizes.

It can be performed using multiple methods. However, to get a content-aware result, which means taking into consideration the image content, a possible way is to perform retargeting based on the “Patch-Based Bidirectional Similarity” measure.

On this proposal I show a way to perform such computation, implement different methods that although might lead to the same result, can be drastically different in terms of efficiency and computational time, and compare results with other algorithms such as Seam Carving.

2 COMPUTATIONAL DETAILS

2.1 Retargeting at the coarsest level

Given the initial source image (S), as the example in Fig 1.

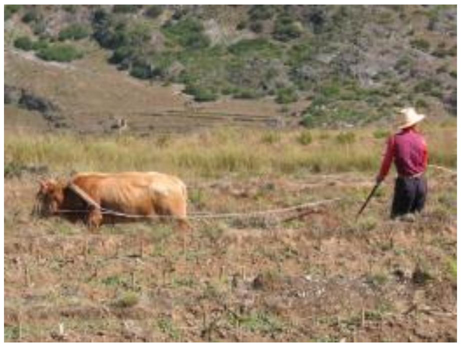


Fig. 1. Source Image (S).

In order to perform the search of the target image (T) we face the problem of that the gap between both S and T is too large.

The proposed solution is to gradually resize the image.
In order to achieve this, the process consists in downsampling the source image S so that we get a smaller version, and then perform the search and vote operation between the downsampled source and each resized target image, until we get to the size of the desired retargeting image result at the coarsest scale. At each time of the resizing process, the height of the image is fixed, as its width is gradually reduced.

A more graphic representation of the process can be seen in Fig. 2.[1]

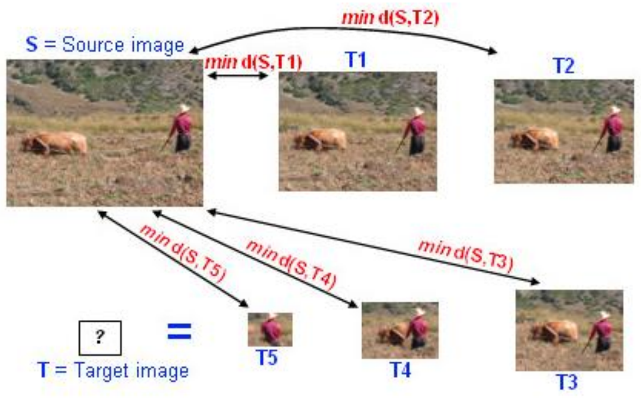


Fig. 2 Retargeting at the coarsest level, to get to the Target Image.

For this part, the search and vote operation was already implemented. However, later I will show and explain in detail an implementation of such function.

2.2 Gradual resolution refinement

At this point, it is time to, starting from the coarsest level, achieve the target resolution.

After defining a certain number of scales, the process consists in upsampling the image (in this case by using Matlab© in-built function *imresize*) and then refine the image by performing the search and vote operation between the resized version of both previous resulting target image and source image (as seen in Algorithm 1).

It should be pointed out that during this computation, the image keeps its aspect ratio through all the scales, as the retargeting is only done at the coarsest level.

This way we can achieve a gradual resolution refinement, since as it will later be shown in section 3, the resulting target image of section 2.1 has a low resolution.

2.3 Final scale refinement

Once the image has the desired size, the last step is to perform the search and vote operation one last time between such image and the initial source image.

This way we can achieve the final resolution refinement and therefore get to the algorithm final output.

All the process was implemented in Matlab© as described in Algorithm 1.

ALGORITHM 1: Coarse-to-fine Retargeting Algorithm

```
Input: source image S
Output: retargeted image R
prepare data for retargeting
down_source ← downsampling of source_image
init_target ← down_source
for temp_width to target_width
        final_target ← Resize of init_target
        refined_target ← search and vote between down_source and final_target
end
refined_target ← final_target
temp_source ← source_image
for 1 to number of scales
        temp_target ← Resize of refined_target
        temp_source ← Resize of temp_source
        refined_target ← search and vote between temp_source and temp_target
end
R ← search and vote between source_image and refined_target
end
```

2.4 Search operation implementation: Naïve Way

As I mentioned before, for the first part of the project the search and vote operation was already implemented. Now I am proposing a Matlab© implementation of both methods.

For the search operation, the goal is to find, in terms of distance, the closest patch in target image T for every patch in source image S, that is, the *nearest neighbor*.

More specifically, when referring to “distance”, we are measuring how similar two patches (group of pixels) $p \subseteq S$ and $q \subseteq T$ are. That is, we are looking for the result of Eq. 1.

$$d(p,q) = \sum_{i=1}^{m} \sum_{j=1}^{n} \big(p(i,j) - q(i,j)\big)^2 \qquad (1)$$

Eq. 1. Distance between two patches *p* and *q*.

Where *m* x *n* represents the number of pixels.

In order to perform such operation the naïve brute force way, the process consists in, for every patch in S, go over the whole target image T and perform the distance operation everytime, and compare every distance value to the previous one, so that we can find the closest.

Graphically, what is happening can be seen in Fig. 3.

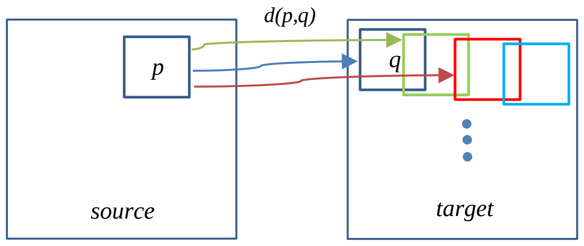


Fig. 3 Search operation: Naïve Way

The proposed implementation of such algorithm is described in Algorithm 2.

ALGORITHM 2: Search Operation: Naïve Way

```
Input: source image S, target image T, size of source/target pacthes
Output: nearest neighbor position matrix
patch_S ← matrix of size_patch_source
patch_T ← matrix of size_patch_target
for each patch patch_S in S
    for all patches patch_T in T
        closest_distance ← inf
        distance ← d(p,q)
        if distance < closest_distance
        closest_distance ← distance
        position_min ← position of such current closest patch_T
        end
    end
    nearest_neighbour ← position_min
end
```

This way of performing the search operation presents a considerable time complexity problem, specifically O(n^2). That is why, in the next section 2.5, I propose an implementation of a more efficient way to compute it, based on randomization.

2.5 Search operation implementation: ‘PatchMatch’ Algorithm

The ‘PatchMatch’ Algorithm is a more efficient method (linear time complexity, O(n)) than the naïve way to perform the search for the nearest neighbor operation, based on randomization.

The idea consists in three steps: (1) Initialization, where a random patch from T is assigned to the one in S as the initial closest neighbor, even if it is not. (2) Propagation, where an iterative process of improving the previous nearest neighbor search is performed. As described in the reference paper, on each iteration of the algorithm offsets are examined in scan order (from left to right, top to bottom), and each undergoes propagation followed by random search.[2]

That translates into a both forward and backward propagation, based on the intuitive idea of that if a pixel (x, y) from source S has a correct mapping and is in a coherent region, then all regions around of (x, y) will be filled with the correct mapping.

(3) Random search: to fully understand this step, we may refer to Barnes paper again.

Basically, if we let $v_0 = f(x,y)$, the goal is to improve v_0 by testing a sequence of candidate offsets at an exponentially decreasing distance from v_0, as expressed in Eq. 2.

$$u_i = v_0 + w\alpha^i R_i \qquad (2)$$

Eq. 2. Candidate offsets

where R_i is a uniform random in $[-1,1] \times [-1,1]$, w is a large maximum search "radius", and α is a fixed ratio between search window sizes. As chosen in the reference paper, the proposed implementation takes $\alpha = \frac{1}{2}$.

A graphic representation of all steps of the process can be seen in Fig. 4 and Fig 5.

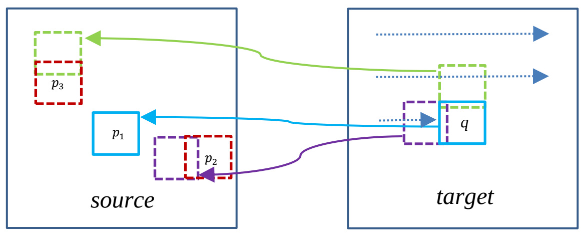


Fig. 4 PatchMatch: (1) Initialization and (2) Propagation.

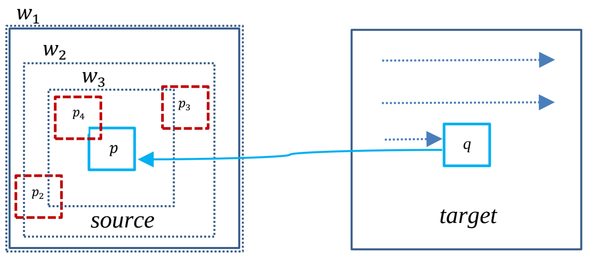


Fig. 5 PatchMatch: (3) Random search.

It was all implemented in Matlab© as described in Algorithm 3.

ALGORITHM 3: Search Operation: 'PatchMatch'

```
Input: source image S, target image T, size of source/target patches, and nearest neighbor initial matrix.
Output: nearest neighbor position matrix
for all patches in T
        patch_T ← take patch from T
        patch_S ← random assignment
        distance ← distance bewtween patch_S and patch_T
        nn ← distance
end
for all patches in T   //Forward
        patch_T ← take patch from T
        nn_matrix(pos) ← perform pixel above nearest neighbor
        nn_matrix(pos) ← perform pixel in the left nearest neighbor
        nn_matrix ← random search between source and target
end
for all patches in T   //Backward
        patch_T ← take patch from T
        nn_matrix(pos) ← perform pixel below nearest neighbor
        nn_matrix(pos) ← perform pixel in the right nearest neighbor
        nn_matrix ← random search between source and target
end
```

2.6 Voting operation implementation

After implementing the search for the nearest neighbor operation in both naïve way and using 'PatchMatch' algorithm, I propose an implementation of the voting operation, which basically aims to decide, after having performed the nearest neighbor search, which color value every pixel should have, taking into consideration the patch overlapping over one single pixel.

To do so, we want to find such color that minimizes the error expressed in Eq 3.

$$\text{Err(T(q))}= \frac{1}{N_S}\sum_{j=1}^{n}(S(\hat{p}_j) - T(q))^2 + \frac{1}{N_T}\sum_{i=1}^{m}(S(p_i) - T(q))^2 \qquad (3)$$

Eq. 3. Error contribution based on each pixel color contribution

Which leads us to finding the unknown value T(q) that satisfies Eq. 4 (*Update Rule*).

$$T(q) = \frac{\frac{1}{N_S}\sum_{j=1}^{n} S(\hat{p}_j) + \frac{1}{N_T}\sum_{i=1}^{m} S(p_i)}{\frac{n}{N_S} + \frac{m}{N_T}} \qquad (4)$$

Eq. 4. Update Rule

We basically are differentiating and equating to zero Err(T(q)), where T(q) represents the unknown color. It should be pointed out that the terms in Eq. 3 are based on the completeness and coherence between source and target images.

Therefore, the voting method was implemented in Matlab© based on finding T(q) shown in such Eq.4. It is described in Algorithm 4.

ALGORITHM 4: Voting Operation

```
Input: source image S, target image T, ann (nn mapping T to S), bnn (nn mapping S to T), and size of source/target patches
Output: unknown pixel color values matrix
num_patch_S ← number of patches in S
num_patch_T ← number of patches in T
completeness_matrix ← size of T matrix
coherence_matrix ← size of T matrix
for patches in S    //Completeness
        patch_S ← take patch using ann
    for size of patches
        completeness_matrix ← completeness_matrix + patch_S(position)
    end
end
for patches in S    //Coherence
        patch_S ← take patch using bnn
    for size of patches
        coherence_matrix ← coherence_matrix + patch_S(position)
    end
end
unknown_pixel_values_matrix ← implementation of Eq. 4 using previously calculated completeness_matrix and coherence_matrix.
end
```

3 RESULTS AND DISCUSSION

3.1 Coarse-to-fine algorithm results

After computing the previously explained method for retargeting images based on a course-to-fine implementation, I now have shown the obtained results, taking image shown in Fig 1 as the algorithm input.

The resulting target image after the retargeting at the coarsest level process can be seen in Fig 6.

Fig. 6. Target image after performing retargeting at the coarsest scale

We can see how the target image represents a good summary of the source image S, and that after retargeting has a low resolution.

That is why, as described in sections 2.2 and 2.3, we need to perform a gradual resolution refinement in order to get to the desired resolution. This final result can be seen in Fig 7.

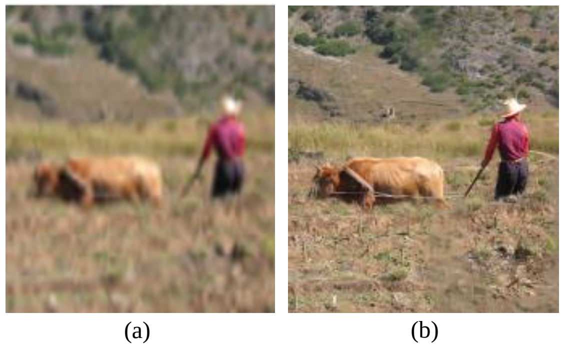


Fig. 6. (a) Output image after performing gradual resolution refinement. (b) Final result, after final scale refinement.

We can see how good the final result (b) looks. The size of the image has changed significantly, but the most representative content is preserved, and except for a little blur in the middle, the image looks smoothly nice. This proves the power of this retargeting algorithm for content-aware image (and video) summarization.

3.1 Coarse-to-fine algorithm results after implementation of search and vote function.

Using the implementation of the search for the nearest neighbor and voting functions described in sections 2.4, 2.5 and 2.6, below I have shown the results of the coarse-to-fine algorithm.

The results after using both the naïve nearest neighbor search function and the 'PatchMatch' algorithm are shown in Fig 7, respectively. The same implementation of the voting function was used in both cases.

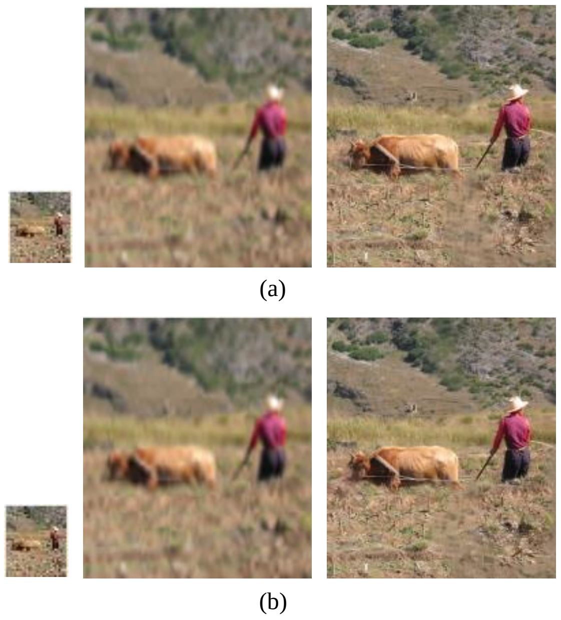
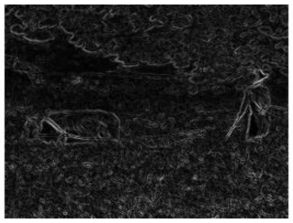


Fig. 7. From left to right: Target image after retargeting at the coarsest scale, output image after performing gradual resolution refinement, and final result, after final scale refinement after (a) using naïve search for the nearest neighbour function and voting function, and (b) using 'PatchMatch' algorithm and voting function. The same implementation of the voting function was used in both cases.

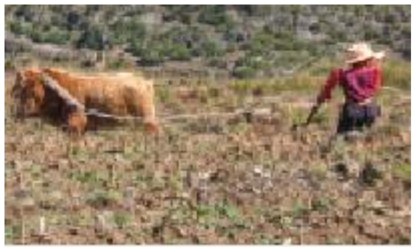
We can see how both nearest neighbor and voting methods work smoothly. The results look almost identical to the ones got after running the algorithm using the already implemented version of the search and vote function (only some parts minimally differ, in terms of a slight blur, a slight difference pixel color or the disposition of very small portions of certain objects.

This also proves the effectiveness of the proposed implementation of such functions/methods.

It is also of interest to compare these results with the one we get after performing the Seam Carving algorithm over the same image, which can be seen in Fig 8.



(a)



(b)

Fig 8. (a) Initial source image energy map and (b) Final result using Seam Carving algorithm.

It is clear that in the Seam Carving algorithm, although the content is slightly preserved, the visual result is pretty worse than the one we get using the proposed retargeting, coarse-to-fine algorithm based on the "Patch-Based Bidirectional Similarity" measure, since as we can see in Fig 6 and Fig 7, the aspect of such algorithm is much better than the image shown in Fig 8.

Eventually, it is of interest to see how much computational time we save when running the algorithm using the 'PatchMatch' method instead of the naïve way, as mentioned in section 2.5.

The resulting computational times are shown in Fig 9.

Method	Computational time
Naïve way	1.5 days
'PatchMatch'	≈ 9 min

Fig 9. Computational time comparison, between using naïve search for the nearest neighbor and 'PatchMatch' algorithm.

We can clearly see that the 'PatchMatch' algorithm is 'ages' faster than the brute force naïve search. These results lead us to confirm that the 'PatchMatch' is tremendously faster, up to the point that it is probably the only way any normal computer could run the program and output results in a reasonable amount of time.

4 CONCLUSIONS

The proposed coarse-to-fine image retargeting algorithm has proved to be efficiently powerful when changing the scale of an image preserving its most representative content, achieving nice visual results. For this particular application, this also proves how useful this method can be compared to other algorithms such as Seam Carving.

In order to perform such computation, based on the "Patch-Based Bidirectional Similarity" measure, apart from already implemented methods that are able to compute the nearest neighbor and voting operation, the two implementations proposed (naïve way and "PatchMatch" algorithm) have also shown their correctness. However, we have clearly seen that the "PatchMatch" method is way faster than the brute force, naïve search, even though both are functionally valid.

ACKNOWLEDGMENTS

This work was supported by the professor and TAs of class ECE178 at UCSB during Fall Quarter 2017.

REFERENCES

[1] Simakov et al. CVPR'08
[2] Barnes et al. 2009 – PatchMatch,.