SLIC algorithm: an energetic review

VIC 2022 project

Enguerrand MONARD
enguerrand.monard@student-cs.fr

Florian LE BRONNEC
f.le-bronnec@student-cs.fr

Abstract

In non-deep machine learning, transforming an image with superpixels is a common preprocessing step in several applications, such as segmentation. The main issues to address when dealing with superpixels are of course the quality of the result but also the computational performances. In this project we are going to present SLIC algorithm, the fastest state of the art algorithm. Evaluating a superpixel segmentation is not a trivial task neither, we are hence also going to spend some time explaining the metrics we can use for the evaluation. Finally, the main result of this project is our proposal for a novel extension, named Energetic Slic *which performs better than SLIC, according to various metrics, by taking advantage of the spectral content of the image while keeping the linear time-complexity of SLIC.*

1. Introduction

Superpixels are connex groups of pixels that are “similar” according to human visual perception. Grouping pixels with a superpixel algorithm leads to a segmented image as illustrated by fig. 1 and we will refer to this operation as superpixel segmentation, or pixelisation. The superpixel segmentation task has several applications such as image compression, image segmentation [1] and inpainting. Several algorithms already exist to achieve this task, such as Simple Linear Iterative Clustering (short SLIC), first introduced by Achanta et al. [1], and TurboPixels, introduced by Levinshtein et al. [3]. We propose to analyze these algorithms and design a new implementation based on our own ideas regarding image processing.

The theoretical complexity of SLIC is in $\mathcal{O}(N)$, with N the number of pixels in the image, which is highly beneficial for many applications. Indeed, even though it is outperformed by other state of the art techniques, its speed has turned it into a top choice for practical applications. This kind of preprocessing can lead to huge speed up of many computer vision tasks such as image semantic labelling, compression, background extraction.

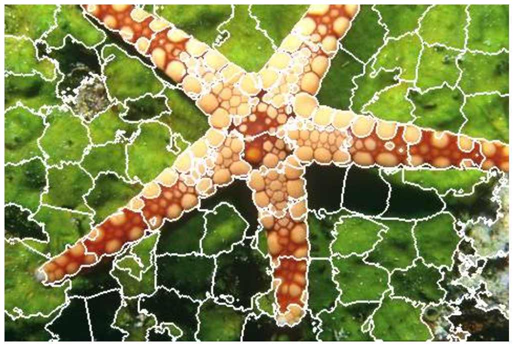


Figure 1: Superpixel segmentation with SLIC algorithm.

2. SLIC - and beyond

2.1. SLIC

SLIC algorithm in described by algorithm 1. It can be seen as a local KMeans clustering but with a custom distance, that accounts for both *color* and *spatial* coordinates, eq. (1).

$$dist^2_{SLIC}(p,q) = \|p_{LAB} - q_{LAB}\|_2^2 + \left(\frac{\texttt{m\_spatial}}{\texttt{step}}\right)^2 \|p_{xy} - q_{xy}\|_2^2, \quad (1)$$

where p, q are two pixels of the image.

In eq. (1), p_{xy} corresponds to the spatial coordinates of a pixel and p_{LAB} corresponds to the the color coordinates in the LAB space. The LAB space has the property that two equally spaced in the LAB space should correspond to two colors equally distant from a human point of view, which is not the case in the standard RGB space.

The idea behind this distance calculation is simply that we want to define superpixels that are to each other and contain similar colors. Of course, fine tuning the parameters `m_spatial` and `step` leads to different weightings of the color distance against the spatial distance.

Stopping criterion In algorithm 1 we loop through a *while* loop, so we need to think about stopping criterion. In addition to a maximum number of iterations `max_iter`, we monitor the displacement of each center of superpixel (in the space dimension). When this position does not move significantly enough on average, we stop the algorithm. While the maximum number of iterations done by SLIC is bounded by `max_iter`, in practice the number of iterations is around 3 if we specify a displacement threshold of 1 pixel.

```
Algorithm 1: SLIC
Data:
# The name of the parameters
  correspond to the ones used in
  code.
 $img$ :  $test$,
 $img$ :  $test$,
img the image to pixelate,
step $> 0$ the initial size of the superpixels
(determines their number),
m_spatial the spatial distance weight,
max_iter the maximum number of iterations,
threshold_error the stopping criterion,

# Initialisation
Initialize the segmentation by selecting the $centers$
 with a step size step.
$error = \infty$
$t = 0$

# Make sure the initial centers do
  not fall on an edge.
$centers^0 = CleverInit(centers^0)$.

while $error >$ threshold_error and
 $t <$ max_iter do
    for $p$ in img do
        Assign to $p$ the closest center in $centers^t$
         within a step $\times$ step squared
         neighborhood, according to the weighted
         distance.
    end
    Update $centers^{t+1}$ according to the new
     assignations by computing the average
     positions and colors.
    $error = \|centers_{xy}^{t+1} - centers_{xy}^{t}\|$,
    $t = t + 1$.
end
# We need one last processing step
  to make sure that our superpixels
  are as connex as possible.
$EnforceConnectivity$.
```

In algorithm 1, we used two functions : $CleverInit$ and $EnforceConnectivity$.

$CleverInit$ When we initialize $centers^0$, it might happen that some centers fall on an edge or on a noisy pixel. To avoid such a situation that could cause some erratic behaviour in the further distance calculation, we move the initial centers to the position of the lowest gradient in a small 2×2 neighborhood.

$EnforceConnectivity$ Once the pixelisation is done, we have no guarantee that the superpixels are connex. Therefore, to get something more satisfying, we do an additional post-processing step to enforce the connectivity of superpixels. To do that, for each superpixel we look for the greatest connected component and we label the other ones at outliers. Then, we match the outliers pixels to the spatially closest cluster center.

2.2. Energetic SLIC

While many variations of SLIC exist, few are able to keep its segmentation quality without losing its linear speed. We introduce *Energetic SLIC*, a variation of the original SLIC algorithm, which uses the spectral information of the image in addition to its color values and spatial layout.

Practically, we generate `patch_size`$\times$`patch_size` new features for each pixel by computing the log-module of the coefficients of the Discrete Fourier Transform (DFT) of a squared patch of size `patch_size` $\times$ `patch_size` centered at this pixel. As the Fast Fourier Transform algorithm has a complexity in $\mathcal{O}(n\log(n))$ (with n the size of the patch on which to apply the DFT), the total operation remains linear.

This idea behind this is that adding frequency information relative to the *texture* of the image could improve the pixelisation, especially in the finer details.

Moreover, Energetic SLIC introduces little to no increase in the number of iterations before convergence compared with SLIC.

$$\begin{aligned} dist^2_{E-SLIC}(p,q) &= \|p_{LAB} - q_{LAB}\|_2^2 \\ &+ \left(\frac{\texttt{m\_spatial}}{\texttt{step}}\right)^2 \|p_{xy} - q_{xy}\|_2^2 \\ &+ \left(\frac{\texttt{m\_frequency}}{\texttt{step}}\right)^2 \|p_{DFT} - q_{DFT}\|_2^2, \end{aligned} \tag{2}$$

To give the full details of our implementation, below are two very important remarks.

DFT vs DCT For the previous explanation we mention the *DFT* of an image. But as we are trying to catch the

Algorithm 2: Energetic-SLIC

```
Data:
Same parameters than for SLIC,
m_frequency the frequency distance weight,
patch_size the patch size around each pixel on
which to compute the DFT.
Initialize the segmentation by selecting the $centers$
  with a step size step.
# Make sure the initial centers do
  not fall on an edge.
$centers^0 = CleverInit(centers^0)$.
for each $p$ in img do
    Compute the log-module of the DFT of the
      patch of size patch_size $\times$ patch_size
      centered at $p$.
    Store this decomposition as
      patch_size $\times$ patch_size new features for $p$.
end
Compute the coordinates of each $C_k$ of $centers^0$ by
  averaging or taking the median of the pixels that
  are initially assigned to them in the step $\times$ step
  squared neighborhood.
Apply SLIC algorithm with these new features and
  use the new distance described by eq. (2).
```

texture information, we actually prefered the *Discrete Cosine Transform (DCT)* (type II) to the DFT. Indeed, the DFT introduces some periodization artefacts (the white cross on the middle of images) whereas the DCT does not (as it implicitely symmetrizes the image), see fig. 2 and [4].

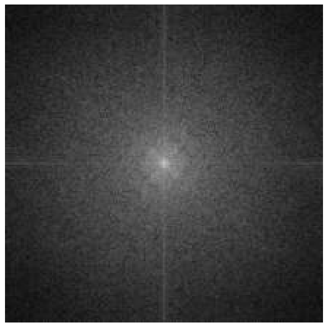
Log-module of DTF As explained in algorithm 2, we compute the log module of the DFT coefficients. More precisely, we transform the coefficients by applying the function $x \longmapsto \log(x + 0.01)$. First, we use the logarithm because, as with the DFT, most of the coefficients will be very low. Then, we add the constant to avoid having to low values in the logarithm. But in fact, adding this offset will also reduce the influence of very high frequencies that are dominated by noise and can improve the result of Energetic-SLIC.

3. Methodology

We implemented the SLIC algorithm from scratch and we extended it into Energetic SLIC.

All computations were performed on an Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz 2.40 GHz with 7.9 GB of RAM with Python3.9.

We will evaluate our implementations by generating the


(a) DFT.

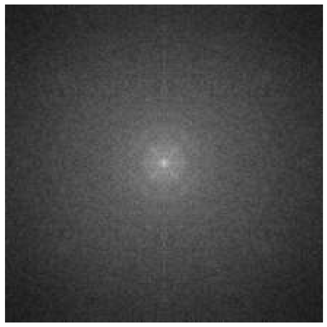
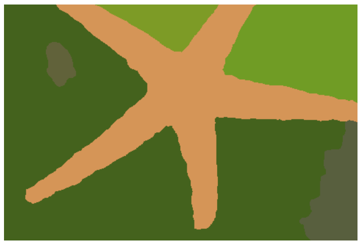

(b) DCT.

Figure 2: Illustration DFT vs DCT (symmetrized) of base croped image of fig. 1.

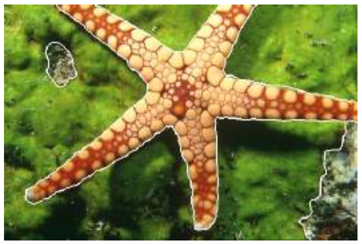
superpixels on the Berkeley Segmentation Data Set and Benchmarks 500 (short BSDS500) dataset [2]. BSDS500 is a standard dataset for evaluating segmentation algorithms. It was used in the original SLIC paper [1]. This dataset is made of 500 481 $\times$ 321 images, 200 in the `/train` folder, 200 in the `/test` folder and 100 in the `/val` folder. For computational considerations, we worked used only the `/train` folder, as our algorithm does not require a learning phase. Moreover, for grid searches and processings requiring extensive use of slowing parameters (high number of superpixels for example), we used only a subset of the `/train` folder, namely 100 images. Some images are framed in landscape mode, others in portrait. This does not affect performance, but we rotated the portrait images 90 degrees left for visualization purposes only.

Each image in the dataset can have several associated annotations, that might be quite different, as illustrated by fig. 3.

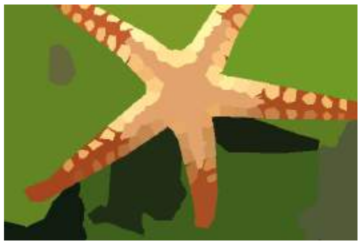
We are now able to compare our proposal with the



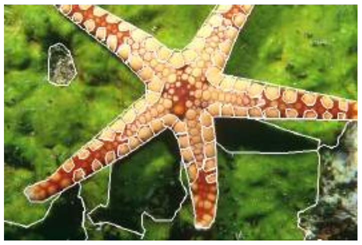
(a) First segmentation.



(b) First boundaries.



(c) Second segmentation.



(d) Second boundaries.

Figure 3: Illustration of two different annotated segmentations for the same image.

ground truth using several metrics, that we are going to detail in section 4.

4. Performance metrics

Running a superpixel algorithm is an unsupervised task, so the notion of evaluation in such a context is always a bit harder than with full supervised learning.

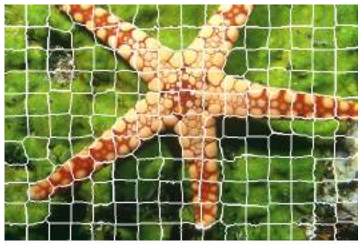
However, even though our problem is unsupervised, we are going to evaluate it on annotated data. To evaluate superpixels algorithms, many methods are available. We focused on three of them, in addition to the basic time performance.

To measure the segmentation quality, we will try to catch *how coherent is our pixelisation with respect to the ground truth segmentation*.

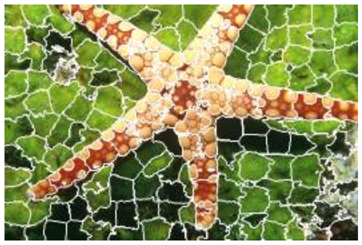
4.1. Undersegmentation Error

The first heuristic we can use for measuring the coherence between a superpixel segmentation and a ground truth one is the *Undersegmentation Error*. The idea behind this is that if a pixelisation matches the ground truth, then all the superpixels should form coherent groups within each part of the ground truth segmentation. In other words, if one half of a superpixel is in a first segmentation zone and the other half to another zone, then our pixelisation is not coherent.

As it is an error, we want this metric to be as small as possible. We can see on fig. 4 two cases of a superpixel segmentation with a high undersegmentation error (bad) and another one with a low undersegmentation error (good).



(a) Undersegmentation Error $= 0.29$, Boundary Recall $= 0.006$, Compactness $= 1.067$.



(b) Undersegmentation Error $= 0.17$, Boundary Recall $= 0.895$, Compactness $= 0.687$.

Figure 4: Illustration of the three metrics on two pixelisations.

For a superpixel $\hat{s}$ and a ground truth segmentation s such that $\hat{s} \cap s \neq \emptyset$, such an error can be defined by a simple difference between the area of the intersection of $\hat{s}$ with s and the total area of $\hat{s}$. eq. (3).

$$\mathrm{U}(\hat{s}) = \frac{1}{N}\left(Area(\hat{s}) - Area(\hat{s} \cap s)\right). \tag{3}$$

Now if we sum this over all the superpixels $\hat{s}$ on all the segments s, we get the formula given by eq. (4), where we simply use the fact that both $\hat{s}_1, \ldots, \hat{s}_k$ and $s_1, \ldots, s_l$ form a partition of the image.

$$\mathrm{U}\left(\widehat{\mathcal{S}},\mathcal{S}\right)=\frac{1}{N}\sum_{s\in\mathcal{S}}\left[\sum_{\hat{s}\in\widehat{\mathcal{S}},\,\hat{s}\cap s\neq\varnothing}Area(\hat{s})\right]-1. \quad (4)$$

The practical implementation of this metric is in the function `undersegmentation_error` within the file `metrics.py`.

We can write it very efficiently using a loop on the ground truth segmentation $s_1,\dots,s_l$ and the `numpy` function `np.unique(..., return_counts=True)`, because the areas can be calculated by simply counting the number of pixels.

4.2. Boundary Recall

The *Boundary Recall* metric is another measure of the coherence of a segmentation. This time we are going to measure how well a pixelisation sticks to the boundaries of the ground truth segmentation. To do that, we simply compute for each pixel of the ground truth boundary if it is in the neighborhood of a superpixel boundary. A neighborhood of a pixel is in this case all the pixels that are closer than 2 pixels of the reference pixel.

This can seem a bit memory demanding as we need to compute a lot of distances. But this can be achieved really efficiently using *KDTree*. We used the `scikit-learn` implementation that has the very handy `sparse_distance_matrix` with `max_distance=2` and `p=2` , which will return for each query ground truth pixels the superpixels boundary points in a 2×2 squared neighborhood. Then we can simply look for the query pixels for which the function returns something. This is implemented in the function `boundary_recall` from the file `metrics.py`.

4.3. Compactness

The last metric we used is the *compactness*. This metric is not informative about the quality of the pixelisation but is about the global shape of superpixels. Namely, it characterizes the "roundness" of the superpixels. This metric is fully unsupervised.

$$\mathrm{C}\left(\widehat{\mathcal{S}}\right)=\frac{1}{N}\sum_{s\in\widehat{\mathcal{S}}}\frac{Area(\hat{s})}{Area(c_{\hat{s}})}, \quad (5)$$

where $c_{\hat{s}}$ is a circle with the same perimeter than $\hat{s}$.

5. Results

5.1. Segmentation/Clustering performance

Thanks to the three metrics we defined, we are able to objectively assess the performance of both algorithms on the dataset. We display on fig. 5 the distribution of these three errors.

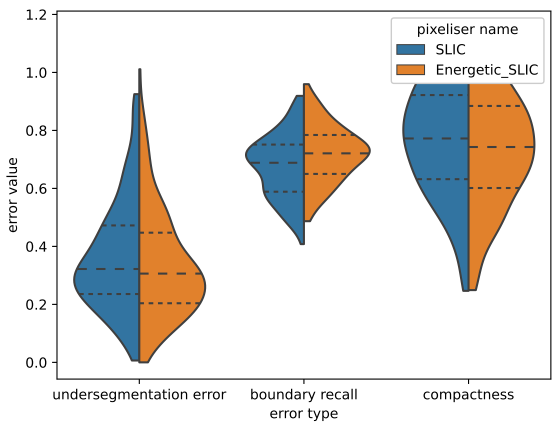


Figure 5: Error distributions

In dashed lines are drawn the 25%, 50% (median) and 75% quantiles of the distribution.

A key takeaway is that Energetic SLIC performs overall better than SLIC. Indeed, it has a lower undersegmentation error, a higher boundary recall. However, this has a cost, which is a lower compactness. This means that Energetic SLIC is a more complex segmentation which captures finer details than SLIC.

If we take a look at appendix A, we can see in praactice where Energetic SLIC performs better than SLIC. Indeed, it captures finer details and sticks more to boundaries it takes into account the edges of the images more than SLIC. This may lead to oversegmentation, like the first image on fig. 13 where Energetic SLIC gets confused by the tiny details.

Nevertheless, we must notice that even when Energetic SLIC is worse than SLIC, it is not so much worse. For example, on the best image on which SLIC outperforms Energetic SLIC for boundary recall, it has a boundary recall superior to SLIC by 0.12, whereas on the first image of fig. 12 where Energetic SLIC outperforms SLIC on the boundary recall, it has a better boundary recall of 0.28, so more than twice as much as 0.12.

This means that Energetic SLIC is better than SLIC for these metrics both overall and in the extreme cases.

5.2. Time performance

We tried to evaluate the performance of both algorithms in term of time elapsed for one image as the number of pixels and superpixels goes up. To do so, we applied both algorithms on the 200 images from the train dataset of BSD500 in the first case, and the 100 first images in the second case.

5.2.1 Increasing number of pixels

First, let's focus on the number of pixels. We scaled down the 200 images from the `train` folder at different scales and watched the evolution of the time elapsed for both algorithms as shown on fig. 6. Notice that the abcissa are linear in the number of pixels, not in the shape nor the scale.

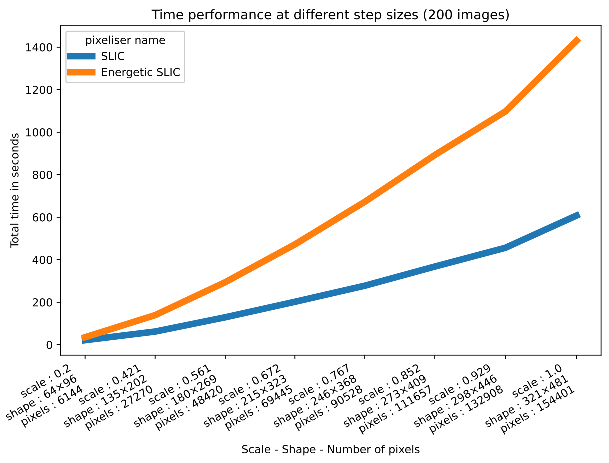


Figure 6: Total time elapsed for 200 images at different scales

From fig. 6 we can see that the time complexity of SLIC is very accurate. However, there are two anomalies we need to explain, at scales 0.2 and 1.0. At scale 0.2, we believe that the time required to load the `numpy` arrays and to perform the computations is skewed by the access memory access and the programming language used, `python`.

At scale 1.0, the phenomenon is very clear, the change of linearity comes from the memory of our computational device. Indeed, we tried at first to scale the images up to a scale of 2.0. However, the algorithm crashed because the computations were too intense. As a matter of fact, we make an extensive use of loops: on the scale factor, then on the algorithm kind, then on the superpixels, and so on. In the end, an implementation in `C++` would have been of great interest. However, the supposedly freely available implementation of SLIC by its author in [1] was nowhere to be found...

We tried to dig a little deeper in the time performance and we were interested in the distribution of the time elapsed for the 200 images, instead of their sole mean. Hereafter, we show on fig. 7 a violin plot of the distribution of elapsed time for every method at every scale.

Figure 7 allows us to understand better the behavior shown on fig. 6. As the scale goes up, so does the number of pixels and distribution seems to dilate. We wanted to visualize these distributions in the worst case scenarios and in the best ones. This would tell us which images are difficult

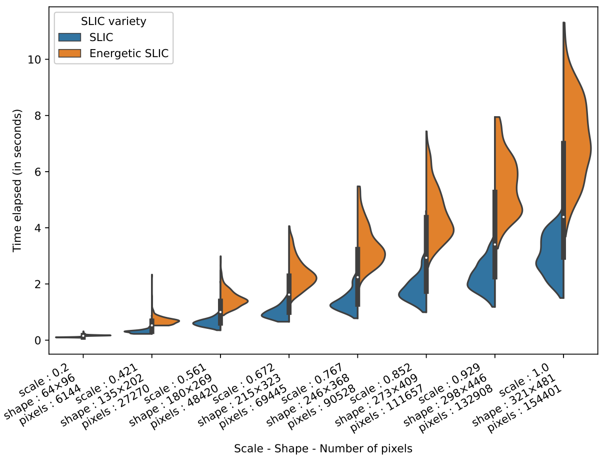


Figure 7: Distribution of time elapsed at all scales

to process for each scale, if they are the same for both algorithms or whether this is pure randomness and some clues of why this is, or is not, the case.

To do so, let's take a look at several images on appendix B. For example, let's look at the worst images on fig. 17 and fig. 19. Three things ought to be noted :

- The slow images are the ones where the connectivity step takes longer because of the huge number of split clusters. Look for example at the fifth worst image for Energetic SLIC on 19. This shows a lady and a very complex background, and so many separate clusters which take time to merge.

- The slow images for one scale are also slow for another scale. Many *bad* images are common to both scales. This is false if the scales are too far apart because the amount of details is not the same at all in that case.

- The slow images for SLIC are also slow for Energetic SLIC and conversely. This proves that those images are slow because of their content, not because of the randomness inherent to computer processing. One could think that this is due to the fact that these algorithms were applied one after the other but this is proved to be wrong because of the second remark, when we noted that these images were also slow at other scales.

A similar line of reasoning can be applied to the best images too, and at other scales, except at the lowest scale where every image is processed at the same speed as they are very few pixels in the image.

5.2.2 Increasing number of pixels

The previous analysis was made at a constant number of superpixels, and the image was scaled down to have fewer pixels. Now, we would like to understand how both algorithms behave as the number of *pixels* is constant but the number of *superpixels* changes (so does the step size). We display on fig. 8 the total time elapsed for 100 images of the dataset. Notice that the abscissa are linear with respect to the number of superpixels. As the step size decreases, the superpixels get smaller and their number grows.

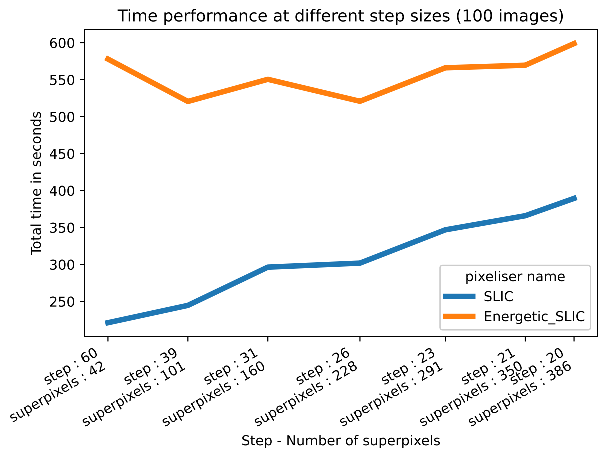


Figure 8: Total time elapsed for 100 images at different number of superpixels

Figure 8 shows us that both SLIC and Energetic SLIC computation times are of the same order of magnitude. Indeed, SLIC needs about 5 minutes to process the 100 images at step size 26 whereas Energetic SLIC needs less than 10 minutes.

As the constant complexity time of Energetic SLIC is a little peculiar, we were interested in the distribution of the time elapsed for the 100 images instead of the sole sum, like in the previous analysis. Hereafter, we show a violin plot of the distribution of elapsed time for every method at every parameter. This is displayed on fig. 9.

A first take by looking at 9 is that the time distribution is very heterogeneous inside the dataset. However, as the overall shape of the error distribuions for both algorithms are the same at fixed parameters (see the double bumps at step size 21 for example), this confirms that the fast images are fast for both algorithms and conversely that difficult images are slow for both SLIC and Energetic SLIC.

An interesting insight is that the distribution of tricky images tends to get thicker as the number of superpixels increases, as we can clearly see two clusters starting from 291 superpixels. This shows that some images are tricky whatever the step size, whereas some are more computationally

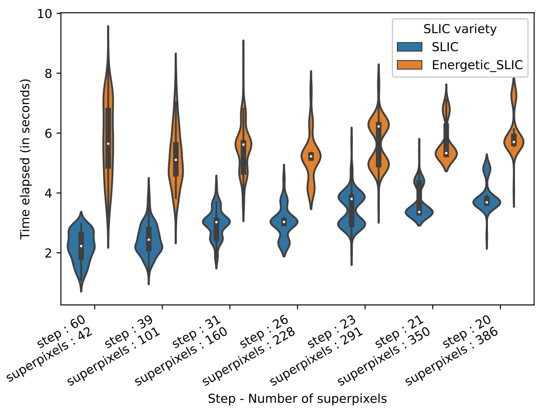


Figure 9: Distribution of time elapsed for each algorithm

demanding only at a higher number of superpixels.

5.3. Segmentation quality

The most important parameters for SLIC algorithm are `m_spatial` and `step`. Indeed, it is these ones that are going to determine 90% of the result.

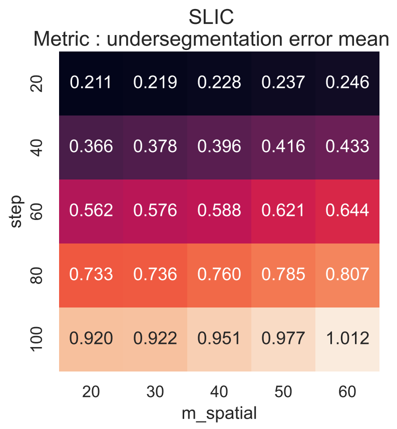
Figure 10 heatmaps shows that there is a trade off between the number and the shape of superpixels and the overall complexity of the segmentation:

- Low `step` means a lot of superpixels and of course a better fit to the original segmentation. However, one should keep in mind that the main goal of superpixels is to achieve image simplification and having a lot of superpixels does not help.
- The lower `m_spatial` is, the less important are the spatial positions of the pixels and in the limit where `m_spatial` $= 0$ then only the color accounts for the segmentation. In this case, it has indeed a higher chance to respect the ground truth segmentation because the region of the same color will be matched together without any constraint and it will be less likely that a superpixels contains things like edges.
- This trade off is illustrated by the *compactness*. A high compactness means highly connex and round superpixels. It illustrates the global complexity of the superpixels.

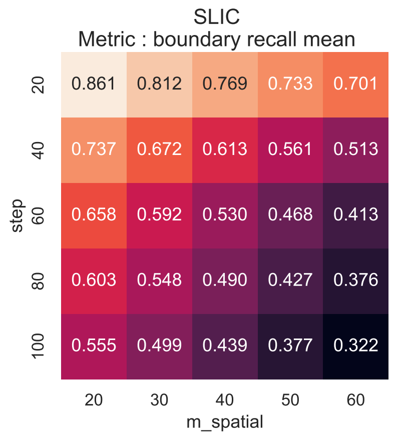
Figure 10 illustrates the interest of using the *compactness* metric, because it is going to penalize pixelisations that are very tortuous and hence not so satisfying regarding the main goal of superpixels.

5.4. Energetic-SLIC parameters

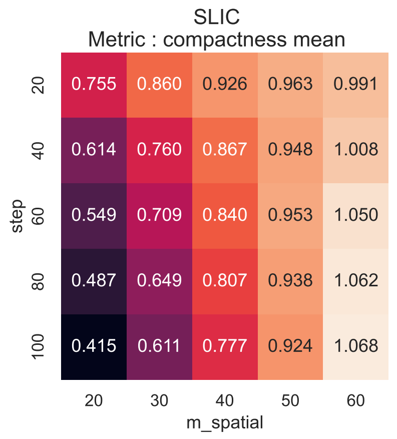
Energetic-SLIC can lead to significative improvements to original SLIC but is has 2 more parameters. With a fixed



(a)

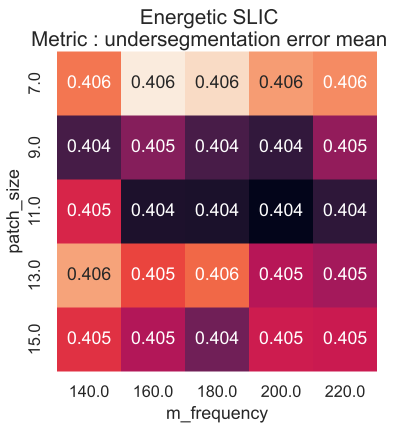


(b)

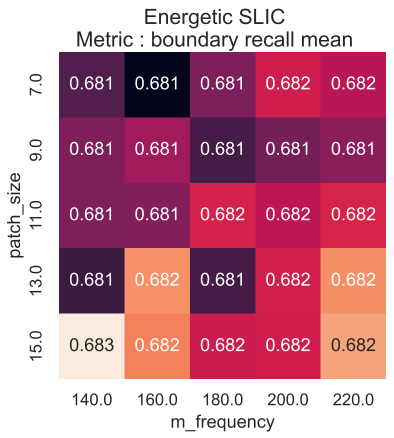


(c)

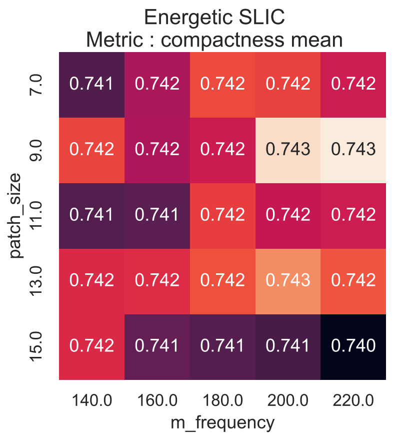
Figure 10: Undersegmentation error, boundary recall and compactness evaluated on 100 images.



(a)



(b)



(c)

Figure 11: Undersegmentation error, boundary recall and compactness evaluated on 100 images.

`m_spatial`, we can look at the influence of `m_frequency` and `patch_size`, as shown on 11.

5.5. Evaluation on best parameters

Testing different parameters, we end up by finding the parameters that work extremely well in practice, presented in table 1. We see that the improvement in Boundary Recall and Undersegmentation Error is compensated by less compact superpixels. But this remains totally acceptable, as the compactness does not decrease abruptly, as corroborated by visual results.

Method	B_{recall}	U_{error}	C_{ness}
SLIC	0.668	0.420	**0.768**
Energetic-SLIC	**0.711**	**0.391**	0.736

Table 1: Evaluation on 200 images, averaged metrics with `step_size` $= 40$, `m_spatial` $= 30$, `m_frequency` $= 120$, `patch_size` $= 9$.

To get insights of the different performance of both algorithms, we encourage the reader to look carefully at the image in appendix A.

6. Conclusion

With this work we introduce a new methode named Energetic-SLIC that keep the main advantages of SLIC while giving better results. Further can be done one the following points:

- This project shows that using local frequency information is useful for superpixels. Maybe this can be run more efficiently, replacing the frequency features by edge enhancement features (but this often creates noise).
- Non linear pre-filtering like bilateral or guided filtering could be investigated.
- Finding more appropriate distances or transformations for the features could be beneficial.
- Even though our algorithm remains linear, it is still a bit slower than the original SLIC in time but especially in memory, work could be done for example by looking at overhall features importance.

6.1. References

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282, 2012.

[2] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011.

[3] A. Levinshtein, A. Stere, K. N. Kutulakos, D. J. Fleet, S. J. Dickinson, and K. Siddiqi. Turbopixels: Fast superpixels using geometric flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2290–2297, 2009.

[4] L. Moisan. Periodic plus Smooth Image Decomposition. working paper or preprint, May 2009.

A. Segmentation results

Here we are going to present a bunch of images where Energetic-SLIC performs better or worse than the original SLIC, according to *Undersegmentation Error* and *Boundary Recall*.

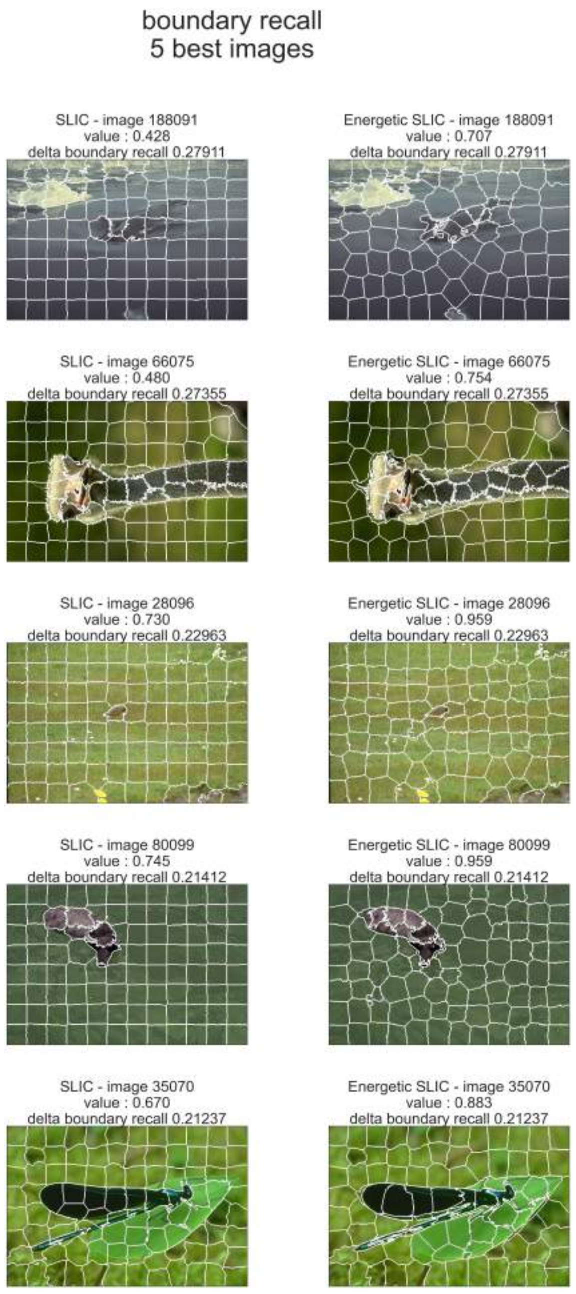

boundary recall
5 best images
SLIC - image 188091
value : 0.428
delta boundary recall 0.27911
Energetic SLIC - image 188091
value : 0.707
delta boundary recall 0.27911
SLIC - image 66075
value : 0.480
delta boundary recall 0.27355
Energetic SLIC - image 66075
value : 0.754
delta boundary recall 0.27355
SLIC - image 28096
value : 0.730
delta boundary recall 0.22963
Energetic SLIC - image 28096
value : 0.959
delta boundary recall 0.22963
SLIC - image 80099
value : 0.745
delta boundary recall 0.21412
Energetic SLIC - image 80099
value : 0.959
delta boundary recall 0.21412
SLIC - image 35070
value : 0.670
delta boundary recall 0.21237
Energetic SLIC - image 35070
value : 0.883
delta boundary recall 0.21237

Figure 12: 5 images where Energetic-SLIC boundary recall is higher than SLIC (good).

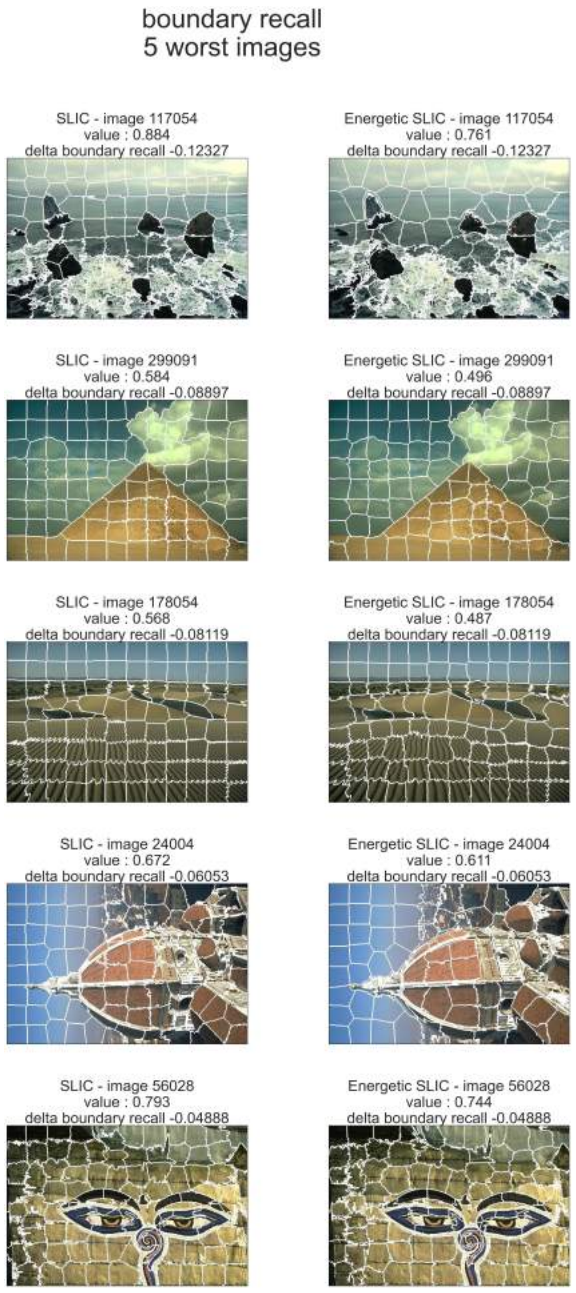

boundary recall
5 worst images
SLIC - image 117054
value : 0.884
delta boundary recall -0.12327
Energetic SLIC - image 117054
value : 0.761
delta boundary recall -0.12327
SLIC - image 299091
value : 0.584
delta boundary recall -0.08897
Energetic SLIC - image 299091
value : 0.496
delta boundary recall -0.08897
SLIC - image 178054
value : 0.568
delta boundary recall -0.08119
Energetic SLIC - image 178054
value : 0.487
delta boundary recall -0.08119
SLIC - image 24004
value : 0.672
delta boundary recall -0.06053
Energetic SLIC - image 24004
value : 0.611
delta boundary recall -0.06053
SLIC - image 56028
value : 0.793
delta boundary recall -0.04888
Energetic SLIC - image 56028
value : 0.744
delta boundary recall -0.04888

Figure 13: 5 images where Energetic-SLIC boundary recall is lower than SLIC (bad).

B. Fastest and slowest images

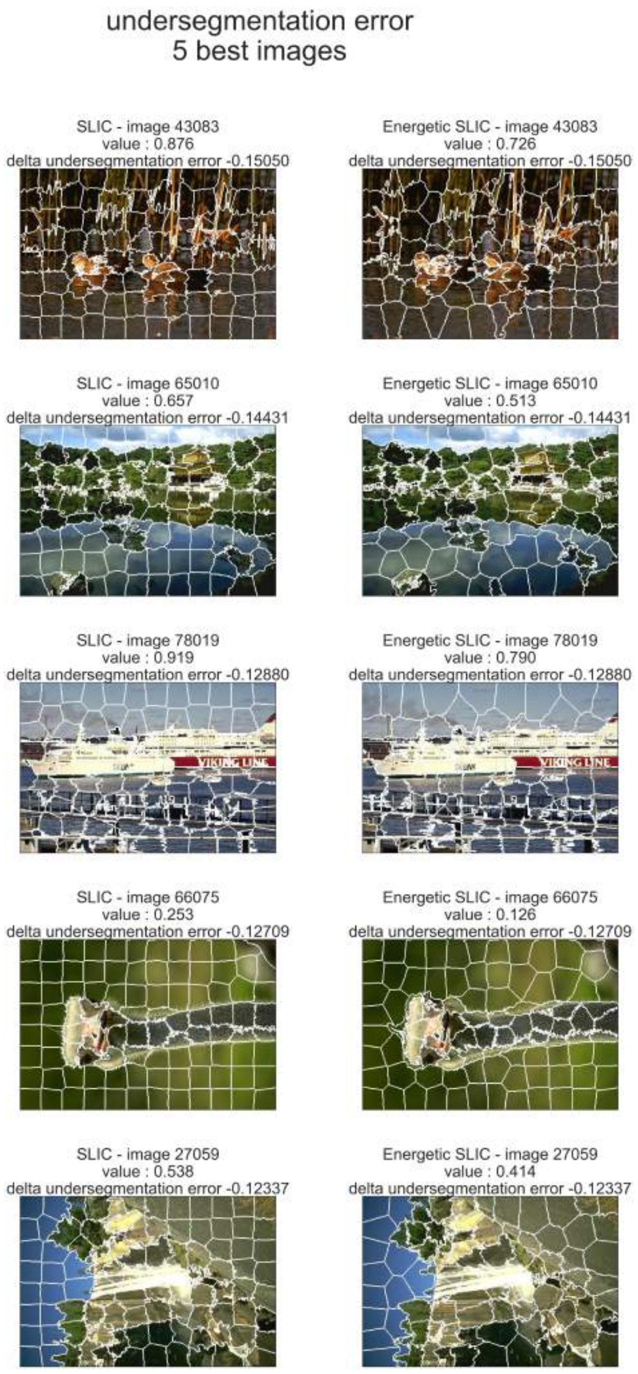

undersegmentation error
5 best images
SLIC - image 43083
value : 0.876
delta undersegmentation error -0.15050
Energetic SLIC - image 43083
value : 0.726
delta undersegmentation error -0.15050
SLIC - image 65010
value : 0.657
delta undersegmentation error -0.14431
Energetic SLIC - image 65010
value : 0.513
delta undersegmentation error -0.14431
SLIC - image 78019
value : 0.919
delta undersegmentation error -0.12880
Energetic SLIC - image 78019
value : 0.790
delta undersegmentation error -0.12880
SLIC - image 66075
value : 0.253
delta undersegmentation error -0.12709
Energetic SLIC - image 66075
value : 0.126
delta undersegmentation error -0.12709
SLIC - image 27059
value : 0.538
delta undersegmentation error -0.12337
Energetic SLIC - image 27059
value : 0.414
delta undersegmentation error -0.12337

Figure 14: 5 images where Energetic-SLIC undersegmentation is lower than SLIC (good).

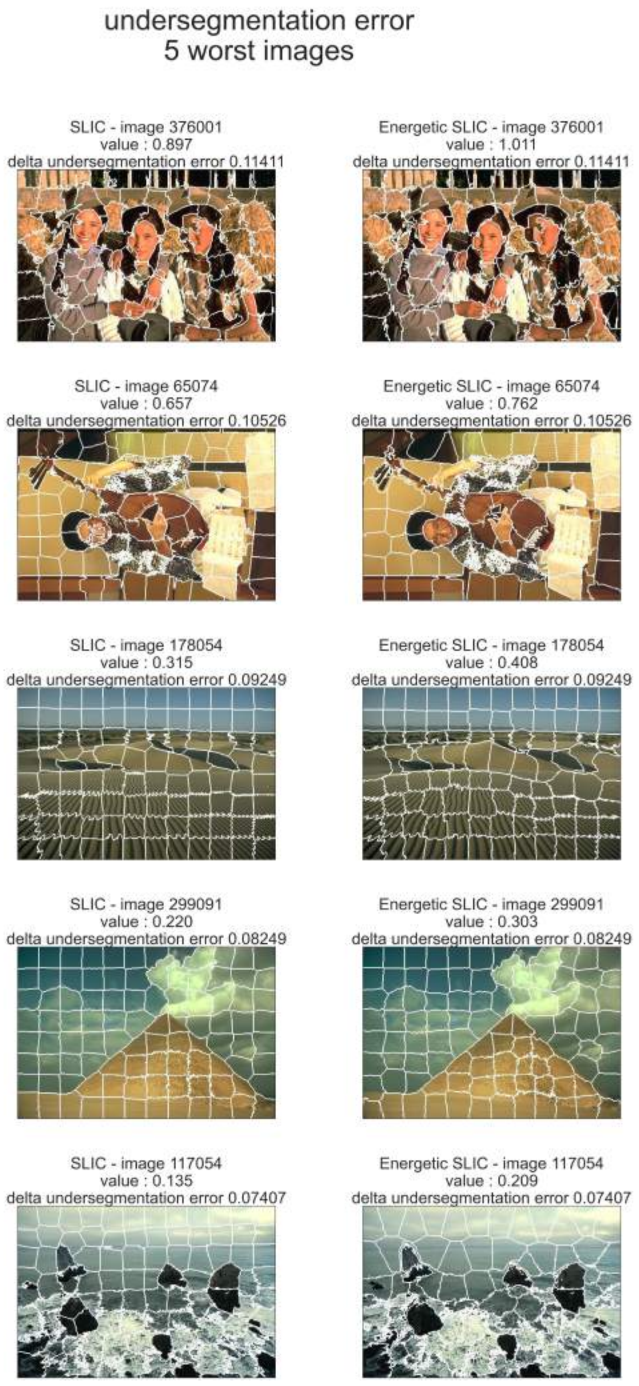

undersegmentation error
5 worst images
SLIC - image 376001
value : 0.897
delta undersegmentation error 0.11411
Energetic SLIC - image 376001
value : 1.011
delta undersegmentation error 0.11411
SLIC - image 65074
value : 0.657
delta undersegmentation error 0.10526
Energetic SLIC - image 65074
value : 0.762
delta undersegmentation error 0.10526
SLIC - image 178054
value : 0.315
delta undersegmentation error 0.09249
Energetic SLIC - image 178054
value : 0.408
delta undersegmentation error 0.09249
SLIC - image 299091
value : 0.220
delta undersegmentation error 0.08249
Energetic SLIC - image 299091
value : 0.303
delta undersegmentation error 0.08249
SLIC - image 117054
value : 0.135
delta undersegmentation error 0.07407
Energetic SLIC - image 117054
value : 0.209
delta undersegmentation error 0.07407

Figure 15: 5 images where Energetic-SLIC undersegmentation is higher than SLIC (bad).

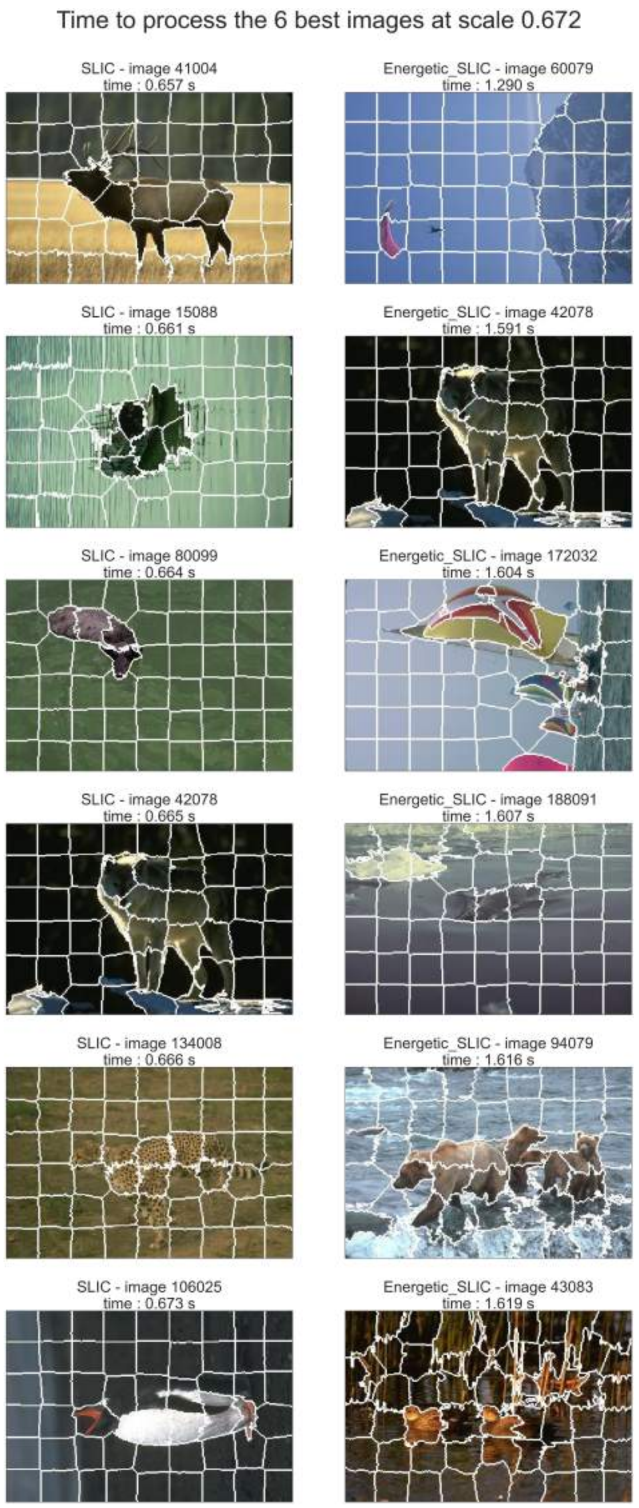


Figure 16: 6 images SLIC and Energetic SLIC are the fastest at scale 1.0

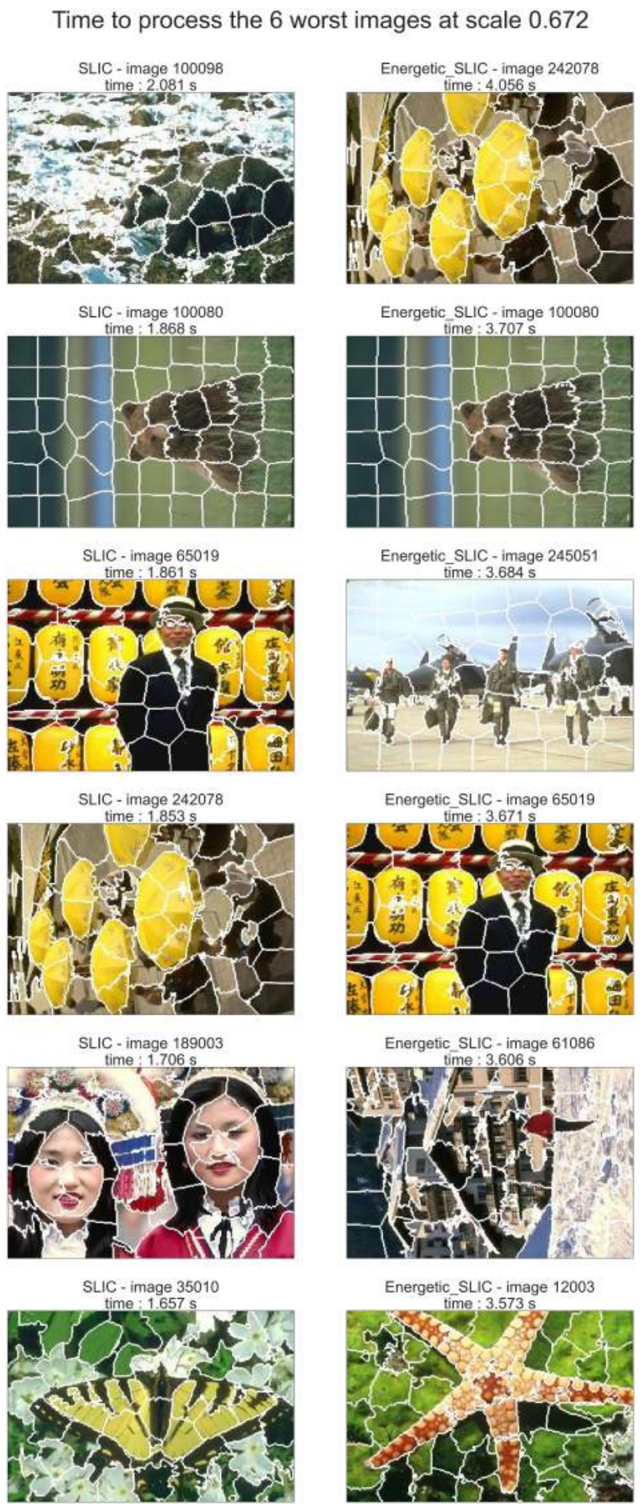


Figure 17: 6 images SLIC and Energetic SLIC are the slowest at scale 1.0

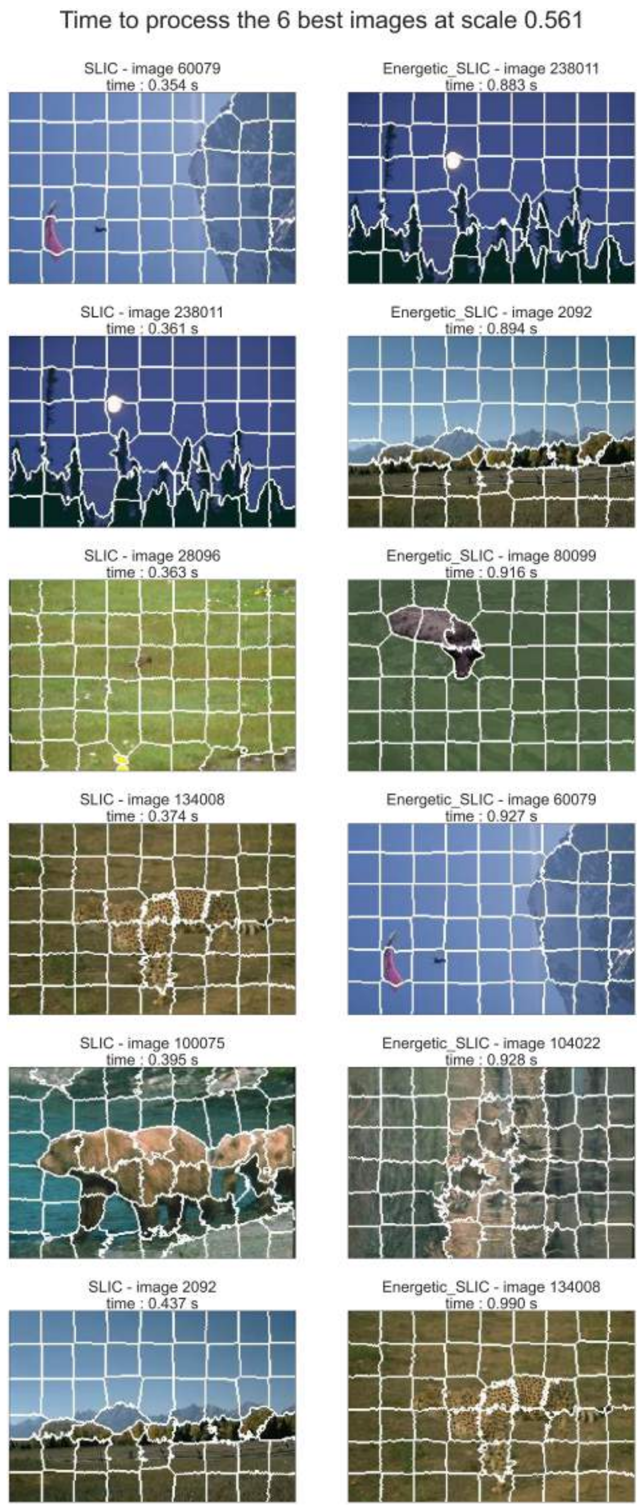


Figure 18: 6 images SLIC and Energetic SLIC are the fastest at scale 0.672

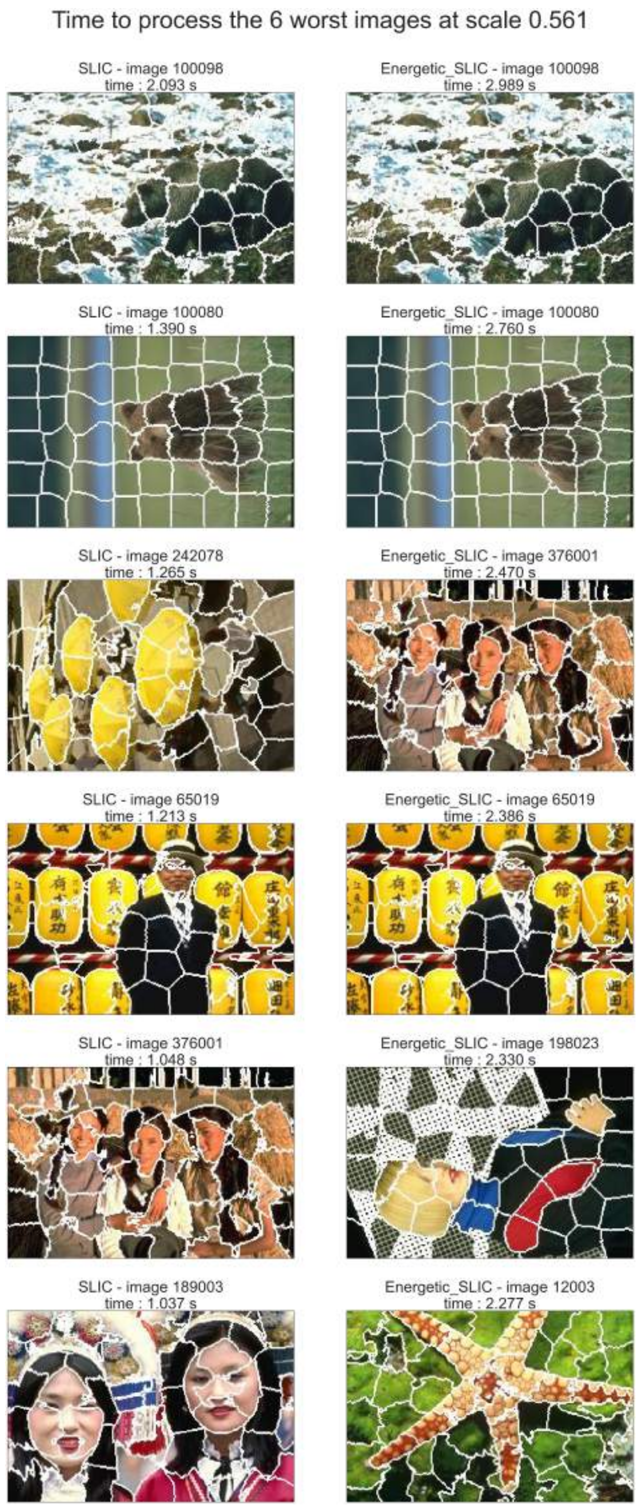


Figure 19: 6 images SLIC and Energetic SLIC are the slowest at scale 0,672