PONTIFICIA
UNIVERSIDAD
CATÓLICA
DE CHILE

INSTITUTO DE ASTROFÍSICA
FACULTAD DE FIÍSICA

COMPUTER VISION AND MACHINE-LEARNING METHOD FOR THE DETECTION OF LOW-SURFACE BRIGHTNESS GALAXIES IN THE FORNAX CLUSTER

POR ALEJANDRA V. HERNÁNDEZ FLORES

Tesis presentada al Instituto de Astrofísica de la Facultad de Físisca de la Pontificia Universidad Católica de Chile para optar al grado académico de Magister en Astrofísica.

Profesor Guía: Dr. Thomas H. Puzia

Enero 2024
Santiago, Chile

I have not proved that the universe is, in fact, a digital computer and that it's capable of performing universal computation, but it's plausible that it is.

— Lloyd Quotes

Dedicated to the loving memory of Freddy S. Hernández Lanz.

RESUMEN

Detectar la tenue luminosidad en las Galaxias de Bajo Brillo Superficial (LSBGs) presenta desafíos significativos, principalmente debido al brillo del cielo y la contaminación de fuentes más brillantes al separar las LSBGs del fondo. A pesar de estos desafíos, el estudio de las LSBGs tiene un gran potencial para avanzar en nuestra comprensión de diversos campos, incluyendo la cosmología, formación de galaxias, evolución y las características de los cúmulos de galaxias. El objetivo principal de este estudio es desarrollar un código automatizado capaz de detectar eficazmente las LSBGs, incluidas aquellas más difusas que solo son detectables mediante búsqueda visual. El enfoque inicial se centra en el cúmulo de galaxias Fornax, con la posibilidad de extenderse a otros cúmulos de galaxias. El propósito es contribuir significativamente al avance de la investigación en LSBGs y sus implicaciones para estudios astronómicos más amplios. Hemos creado un código automatizado que detecta con éxito las LSBGs en imágenes digitales a una velocidad de procesamiento razonable. Hemos incorporado un algoritmo innovador para separar las LSBGs del fondo utilizando un núcleo de fondo dinámico y un umbral aplicado a segmentos de imagen para lograr esto. También hemos implementado un filtro bilateral que identifica las LSBGs más difusas y preserva la morfología, asegurando una identificación y clasificación precisas. Además, hemos desarrollado y entrenado un clasificador de Máquina de Vectores de Soporte de una Clase (SVM) utilizando una muestra de referencia de 143 LSBGs, resultando en un clasificador con una baja tasa de falsos positivos. El código implementado ha detectado con éxito las LSBGs, demostrando su capacidad para abordar los desafíos asociados con la identificación de la tenue luminosidad en estas galaxias, incluso en presencia de fuentes más brillantes. El algoritmo integrado ha mejorado significativamente la precisión y eficiencia del proceso de detección, permitiendo la identificación de un número sustancial de candidatos a LSBG. Específicamente, en el Cúmulo Fornax, nuestro algoritmo identificó con éxito 31,295 candidatos a LSBG, como se documenta en el catálogo integral disponible en el Repositorio de GitHub: https://github.com/Alevhf/LSB_candidates/blob/main/Catalog_result.csv#L19304.

ABSTRACT

Detecting the faint luminosity in Low Surface Brightness Galaxies (LSBGs) poses significant challenges, primarily due to sky brightness and contamination from brighter sources while separating LSBGs from

the background. Despite these challenges, the study of LSBGs holds great potential to advance our understanding of various fields, including cosmology, galaxy formation, evolution, and the characteristics of galaxy clusters. The primary goal of this study is to develop an automated code capable of effectively detecting LSBGs, including the more diffuse LSBGs that are only detectable through visual search. The initial focus is on the Fornax cluster of galaxies, with the possibility of extension to other galaxy clusters. The purpose is to significantly contribute to advancing research in LSBGs and its implications for broader astronomical studies. We have created an automated code that successfully detects LSBGs in digital images at a reasonable processing speed. We have incorporated an innovative algorithm to separate LSBGs from the background using a dynamic background kernel and threshold applied to image segments to achieve this. We have also implemented a bilateral filter that identifies the most diffuse LSBGs and preserves morphology, ensuring precise identification and classification. Additionally, we have developed and trained a One-Class Support Vector Machine (SVM) classifier using a gold sample of 143 LSBGs, resulting in a classifier with a low rate of false positives. The implemented code has successfully detected LSBGs, showcasing its ability to address the challenges associated with identifying the faint luminosity in these galaxies, even in the presence of brighter sources. The integrated algorithm has significantly improved the accuracy and efficiency of the detection process, allowing for the identification of a substantial number of LSBG candidates. Specifically, in the Fornax Cluster, our algorithm successfully identified 31,295 LSBG candidates, as documented in the comprehensive catalog available at GitHub Repository: https://github.com/Alevhf/LSB_candidates/blob/main/Catalog_result.csv#L19304.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

1 INTRODUCTION

The study of the faintest objects in the universe goes hand in hand with advances in observational techniques and the invention of more powerful telescopes. Low surface brightness galaxies (LSBGs) are a clear example of this. One of the most interesting of these cases is Malin 1 [6, 2, 4]. The galaxy Malin 1 is a large spiral galaxy, fortuitously observed due to the presence of a suitable nucleus for optical spectroscopy, allowing the determination of its redshift and an approximate distance of about 10 kpc. However, as noted by [6], if Malin 1 were embedded in the Virgo cluster, the bulge component would manifest as a prominent elliptical at $V = 12$ with an effective radius of $1''$. Additionally, its low-surface-brightness (LSB) disk would be spread across a degree of the sky, and any individual H I regions might appear as unresolved, faint, blue objects. By enhancing the power of telescopes and, consequently, the resolution of astronomical images, it was confirmed not only that Malin 1 is a giant spiral galaxy with a low-surface-brightness disk but also that there are more low-surface-brightness galaxies similar to Malin 1 [27, 2, 4]. Thanks to the resolution of today's telescopes, more diffuse objects can be detected in rich, high-density environments. [38]. Many dwarf, LSB-type galaxies have been found in the Virgo Cluster, Coma Group, and Fornax Cluster. The finding of more LSBGs allowed the questioning of the knowledge of galaxy evolution and cosmological theories [27].

LSBGs, as the name implies, encompass a group of galaxies inherently characterized by their low luminosity. A galaxy qualifies as an LSBG if its surface brightness registers above 25 mag/arcsec^2 in the B-band. While the threshold value for this classification has not been universally standardized, it typically falls within the range of 22.0 to 25.0 mag/arcsec^2 in the scientific literature. Notably, the low surface brightness of these galaxies suggests that they evolve at a notably slower pace and may undergo star formation outside the typical molecular cloud environments. LSBGs are also enveloped by dark matter halos of lower density, which are more extensive than those surrounding high surface brightness (HSB) galaxies. In contrast to HSB disks, LSBG disks exhibit a substantial dominance of dark matter across all radii, accompanied by a systematic increase in the M/L (mass-to-light) ratio as central surface brightness decreases [5].

The distinctive feature of LSBGs that renders their identification and mass detection challenging is their surface brightness, which often dims to levels fainter than the natural night sky brightness itself [5, 16, 23, 22, 72]. Notably, extremely low surface brightness dwarfs

even surpass classical Ultra Diffuse Galaxies (UDGs) in terms of faintness [19, 38]. LSBGs encompass many galaxy morphologies, including spiral, elliptical, dwarf, and irregular galaxies [69]. It is also noteworthy that LSBGs are found in a diverse array of physical environments, ranging from being satellite galaxies to residing in densely populated galaxy clusters [46, 49, 13]. Among the well-documented clusters hosting LSBGs are the Virgo, Coma, Perseus, and Fornax clusters of galaxies [27, 36, 38].

Research conducted by the Complex Stellar Systems Group[1], such as [38, 41], has demonstrated that dwarf galaxies classified as LSBGs exhibit a size-luminosity relation that allows for the constraint of specific characteristics. These characteristics facilitate the identification of LSBGs as a distinct group from other faint celestial sources, such as bright dwarf galaxies and ultra-compact objects. In addition to the challenges associated with detecting LSBGs, there are complexities in modeling LSBG profiles and parameters. The faintest LSB galaxies exhibit a porous structure, and their surface brightness is scarcely distinguishable from the background intrinsic luminosity [26, 72]. These characteristics render the fitting of profiles challenging, often leading to significant errors. Moreover, determining the maximum extent of LSBGs is hindered by their low luminosity, making it, in some cases, impossible to derive reliable adjustable parameters for the faintest LSBGs. As a result, the identification of LSBGs is frequently carried out manually, relying on visual detection. This manual approach tends to result in the identification of a limited number of galaxies compared to what could be found in extensive databases of sky observations, especially when the search is automated. Additionally, manually identified LSBs predominantly consist of dwarf galaxies, overlooking the ultra-diffuse LSBs, which are detected incidentally to a large extent. This underscores the need for a comprehensive database for studying low surface brightness galaxies (LSBGs). This approach is imperative to gain a better understanding of the characteristics of LSBGs without introducing selection biases, encompassing both dwarf and ultra-diffuse galaxies on an equitable basis.

LSBGs play a pivotal role in refining cosmological theories. According to the conventional Cold Dark Matter (CDM) model, galaxies form due to initial Gaussian density fluctuations in the early universe, evolving linearly, collapsing, and eventually virilizing into the galaxies we observe today. These simulations and models make predictions regarding the local galaxy population. However, CDM models predict more galaxies than have been detected or observed [33]. The discrepancy between predicted and observed galaxies has led to the proposal of alternative theories. In this context, LSBGs are considered potential reservoirs of baryonic matter within the universe [5, 13, 33]. The study of LSBGs can contribute valuable insights into

1 https://www.astro.puc.cl/~tpuzia/PUC/Research.html

the broader galaxy population, the three-dimensional distribution of galaxies, and the quantity of baryonic matter residing in galactic potentials.

LSBGs contribute to refining cosmological models through their impact on the Luminosity Function (LF). The optical luminosity function of galaxies holds a fundamental role in cosmology [49]. Precise knowledge of the luminosity function enables the estimation of the universe's average luminosity density and predictions regarding the redshift distribution of objects across various magnitude intervals. Moreover, the shape of the luminosity function serves as a testing ground for galaxy formation theories [49]. LSBGs significantly influence the faint tail of the galaxy luminosity function, allowing for tighter constraints to be imposed on models of galaxy formation [38, 49]. Nevertheless, it remains uncertain whether extrapolating the luminosity function to fainter magnitudes is valid, as the luminosity density drops significantly below $M_B = -17$ [38].

Despite the challenges and objectives of this research, there is an ongoing effort to develop an automated method for identifying LSBGs, particularly in large digital sky survey images. The aim is to create an automatic process that minimizes or eliminates the need for user intervention in the identification process. This entails both detection and classification methods. To achieve this, non-parametric variables are employed to describe and train machine learning algorithms for classification. These parameters have been adapted from crystal parameterizations in microscopy observations [38].

1.1 FORNAX CLUSTER IMAGE SELECTION

To develop a search algorithm for Low Surface Brightness Galaxies (LSBGs), it is essential to have access to a laboratory with identified LSBGs for testing. In our research, we developed the search algorithm using data from the Fornax cluster of galaxies. The Fornax Cluster is the nearest galaxy over-density in the Southern Hemisphere, making it the brightest galaxy cluster in this region. It boasts twice the central galaxy density with a lower mass of approximately $[(7 \pm 2) \times 10^{13}\ M_\odot]$. Additionally, the fraction of early-type galaxies in the Fornax Cluster is significantly higher than in the Virgo Cluster, as reported by [38]. Recent research, including studies by [36], [38], and [41], has identified faint over-densities and potential LSBGs within the Fornax galaxy cluster, from which [41] has compiled a diverse LSBGs (nucleated, non-nucleated, and diffuse galaxies). This makes Fornax Cluster an excellent candidate for the search and discovery of LSBGs, as underscored in [41].

The Next Generation Fornax Survey (NGFS) III, as described in [19] and [41], serves as the primary survey dataset utilized in the development of our LSBGs search algorithm. The NGFS is a comprehensive

multi-wavelength survey spanning the central 30 square degrees of the Fornax cluster of galaxies. These observations were conducted using the 4-meter Blanco telescope at Cerro Tololo Interamerican Observatory (CTIO). The NGFS dataset comprises seven contiguous mosaics, with the central mosaic (tiles 1) corresponding to the core of the Fornax cluster, covering an area of 3 central square degrees. The survey area extends from the cluster core to a radius of 350 kpc, with the galaxy NGC 1399 serving as the cluster center. This extension encompasses 25% of the virial radius of the cluster, equivalent to 100% within 1.4 Mpc. The remaining regions outside the 350 kpc radius are called mosaics or tiles 2-19, detailed in [41]. We tested our code across the entirety of Fornax, encompassing both the central region of Fornax (tile 1) and the remaining 19 tiles. The catalog of LSBGs used for calibrating the detection code to identify LSBG candidates consists of data from the central region of Fornax and 19 tiles. This catalog comprises 590 LSB dwarf galaxies presumably found via visual inspection, as documented in [19] and [41]. Our detection algorithm was exclusively applied to images in the g-band for DECam. The choice to use only the g-band was due to its lower contamination of background or sky noise compared to bluer bands. The raw images underwent processing using the DECam Community Pipeline (v2.5.0) [57], which included bias calibration, crosstalk correction, linearity correction, flat-fielding, and gain calibration. For more detailed information on image calibration procedures, please refer to [19], [38], and [41].

1.2 BACKGROUND ON THE DETECTION OF LSB TYPE GALAXIES

The evolution of Low Surface Brightness Galaxies (LSBGs) detection can be divided into two distinct eras of observational techniques. The Photographic Plate Era, characterized by identifying LSBGs on photographic plates, was influenced by limitations in early observational tools, leading to selection biases and assumptions about the nature of these galaxies. During this period, the groundwork for incorporating automation into LSBG searches was laid, setting the stage for advancements in the Digital Imaging Era through Charge-Coupled Devices (CCD). The transition to digital images brought improvements and introduced challenges in detecting diffuse LSBGs. It also led to the development of automated source detection algorithms for digital images, such as SEXTRACTOR, for the identification process. Despite progress, visual inspection remained crucial in modern LSBG surveys, particularly in the search for extremely diffuse LSBs. This chapter navigates through these historical milestones, documenting the evolution from manual techniques in the Photographic Plate Era to semi-automated approaches in the CCD Imaging Era, and explores

the promising avenue of Machine Learning for enhanced automation in LSBG detection.

1.2.1 *Photographic plate era*

The first detections and characterizations of LSBGs, known as Malin objects at the time, were documented in several articles [5, 26, 27]. However, it's important to note that the characteristics of these LSBGs, as revealed in the initial survey [5, 24, 27, 65], were influenced by the observational techniques available during that era. Consequently, the first detections of LSBGs were susceptible to selection bias [16]. This selection bias resulted in underestimating the number of LSB galaxies and introducing erroneous features [5, 23, 60]. For instance, it was assumed that the faintest LSBGs must be small, similar in size to their High Surface Brightness Galaxy (HSBG) counterparts [5, 34]. Additionally, the assumption of an exponential luminosity profile limited the identification of LSBGs. Only later, in papers like [20, 31], the possibility arose that LSB galaxies might be more common but largely undetected and uncharted. This revelation had significant implications for estimating the population density of LSBGs, particularly in the faintest part of the luminosity function [5, 23].

Subsequent research such as [5, 43, 47, 50, 69, 72] laid the foundations for understanding LSBGs by incorporating diffuse galaxies into the analysis. As astronomical observation techniques improved, higher-quality images became available for exploration, enabling the detection of more sources, such as DRAGONFLY [1]. However, the ability to detect additional sources and potential LSBG candidates was still constrained by the visual identification of these sources, as discussed in [72]. Consequently, it became evident that optimizing detection techniques for LSBGs was essential.

The first improvements and automation of LSBG detection techniques are documented in subsequent research articles, including [14, 21, 26, 28, 36, 37, 43, 44, 48, 59, 62]. These studies introduced automatic photographic plate detection (APM) of UK Schmidt observations for source detection, followed by profile fitting to identify LSBGs. While APM contributed to numerous LSBG detections and surveys, it was limited by the noisy nature of photographic plates, making it challenging to detect faint and extensive LSBGs and introducing its own detection bias [28, 53, 60]. Additionally, the reliance on the exponential profile setting restricted the identification of LSBGs to those conforming to this profile. An attempt to enhance source detection on Schmidt survey plates was made in the article [58], which marked a transition from source detection on photographic plates to digital images using the PISA algorithm based on APM. The research of [58] also explored other algorithms, including clustering, for LSBG identification. However, [53] concluded this transition to digital im-

age detection with low noise levels but still required visual inspection for LSBG detection.

1.2.2 *CCD imaging era*

With the advent of digital imaging, astronomical image processing and reduction techniques significantly improved, enabling the reduction of sky noise for the detection of diffuse objects. Automated source detection algorithms like SEXTRACTOR [3] were developed and refined. SEXTRACTOR relies on the "connected pixel" method to locate objects, where groups of connected pixels above a threshold are identified as detections. However, SEXTRACTOR and similar algorithms were not efficient enough in detecting fainter, diffuse, or extended LSBGs [15, 32, 48, 49, 62, 64]. The main limitation was that connected pixel detection depended on a high signal-to-noise ratio. In contrast, LSBGs typically exhibit a low signal-to-noise ratio. Although some improvements were made in SEXTRACTOR to detect LSBGs [51, 52], they still required manual configuration and visual inspection, rendering them semi-automatic algorithms.

An alternative to SEXTRACTOR for the automatic detection of LSBGs is MARSIAA (Markovian Software for Image Analysis in Astronomy) [68], which employs stochastic processes to determine the best threshold adaptively and incorporates multi-band data imaging. While the Markovian algorithm finds fainter and more diffuse LSBGs than SEXTRACTOR, it generates many false detections and still necessitates visual inspection [32]. Modern LSBG surveys continue to rely on visual inspection, as exemplified in [17, 18, 19, 30, 32, 38, 39, 63, 67]. SEXTRACTOR and Markovian-type algorithms are used as detection guides. Still, they do not fully automate the process.

An emerging approach in LSBG research is using Machine Learning algorithms for automated detection and identification. Various articles have explored this avenue, albeit with limited application to diffuse objects. Machine learning algorithms are categorized as supervised, semi-supervised, or unsupervised [10, 29], each with its unique methodology and biases [29].

Supervised algorithms rely on galaxy catalogs or models for training and are limited in detecting galaxies with specific morphologies. This approach may not be suitable for LSBGs, as they exhibit diverse morphologies [11]. An example of a supervised machine learning algorithm can be found in [11], which uses the random forest algorithm to distinguish galaxies, stars, and quasars but faces difficulties with faint objects.

Unsupervised algorithms, such as clustering, have been employed in low surface brightness galaxy (LSBG) detection research, as demonstrated in [55] for high surface brightness (HSB) galaxies. Clustering has also been explored in LSBG detection, as shown in [44], where

the DBSCAN algorithm was utilized. However, the clustering result from DBSCAN, as per [44], presents false detections that were not successfully distinguished from LSBGs.

Deep learning algorithms, a subset of machine learning, are highly adaptable but require extensive databases to operate effectively. Some studies, like [25, 55], have utilized deep learning algorithms for LSBG detection, showing promise but needing substantial data. A particularly successful application of deep learning in LSBG identification is found in [70], where LSBGs were successfully identified among artificial LSBGs.

2

METHOD

This section explains the algorithms and steps for searching for LSBGs in astronomical images. Detecting LSBGs or other objects in images involves Computer Vision, a sub-field of artificial intelligence. Computer Vision aims to develop programs capable of object detection, segmentation, localization, and recognition within images [54].

It is crucial to differentiate between the detection and identification processes in Computer Vision, although they are closely related concepts [42]. Detection involves applying processes to a digital image to separate and label a set of pixels of interest (object or pattern of interest) from the background or threshold of the image. On the other hand, identification or recognition involves cataloging or classifying, assigning a known label or specific group to an object [42].

Identifying a figure in an image requires initial conditions or features that can describe, compare, and classify the detected pixels. These features act as rules to define and describe the detected figure in the image, establishing a correlation with the astrophysical object being sought, in our case, the LSBGs. Typically, these features are derived from the analysis and theoretical understanding of the object sought in the images. However, it is also possible to create features based on correlations derived from the initial detection process, which can be identified manually. The ultimate goal of classifying or identifying figures in images is to leverage these features, representing our knowledge, to refine the detection process and retain those figures that best describe astrophysical objects of interest rather than false positives, i.e., any detection that is not the object of interest.

In the context of Machine Learning algorithms, it is common to structure them into three main parts, as outlined by [7]. These parts are typically referred to as pre-processing, processing, and post-processing.

- Pre-processing: This stage encompasses all the algorithms applied to the image before the detection process. Its purpose is to modify the image to facilitate the subsequent detection process.
- Processing: The processing stage involves algorithms and procedures to detect LSBGs or the target objects of interest in the images.
- Post-processing: Various studies and techniques are employed in the post-processing phase, mainly focusing on feature selection and identification. This stage aims to refine the results obtained during the detection process and improve the accuracy of object identification.

These three stages collectively form the foundation of machine learning-based object detection and identification approaches in astronomical images.

2.1 PRE-PROCESSING

Image segmentation involves dividing an image into multiple segments and assigning an identification to each component. This technique offers several advantages in pre-processing, including faster and more straightforward extraction of galaxies when working with smaller images. Moreover, processing smaller images provides the additional benefit of enhancing the accuracy of sky or background determination, given the algorithm's functioning.

Numerous algorithms and approaches exist for image segmentation. Still, our objective was to identify an algorithm that would yield the fewest segments while ensuring that galaxies, mainly those not yet identified, are not divided. The segmentation algorithm that meets these criteria is the Simple Linear Iterative Clustering (SLIC), available in the Python Scikit-image library [66]. SLIC relies on super-pixel algorithms to perform segmentation. Super-pixels are groups of pixels sharing common characteristics, such as pixel intensity. Super-pixel algorithms cluster pixels into perceptually meaningful regions, effectively replacing the rigid pixel grid structure. Various methods can be employed to generate super-pixels. In the case of SLIC, the K-means clustering algorithm generates super-pixels, utilizing the contours of planar figures in the image as super-pixel boundaries.

SLIC is known for its simplicity, ease of use, and efficiency in quickly processing segments from a complete tile with irregular borders following the contrast of the less bright pixels in the sky, preventing the division of ultra-diffuse LSBGs.The most critical parameter is `n_segments`, which defines the desired number of super-pixels. Other parameters are typically left at their default values. For example, the initial number of segments was set to 54 to ensure that each segment's size is approximately 263×263 arcsecond, considering the segments are irregular. As a result, the central Fornax image was divided into 53 parts, and the tiles were split into 57 segments. This segment size is sufficiently small to represent the sky accurately and accommodate larger galaxies within segments without breaking them. Since the algorithm generates segments based on super-pixels, the segments vary in size and are not uniform. This irregularity of segments offers the advantage of effectively adapting to the image's edges, where noise levels are typically higher.

Listing 1: slic

```
from skimage.segmentation import slic
```

```
segments_slic = slic(img, n_segments=54, compactness=0.001, sigma
    =3.0, enforce_connectivity=True)
```

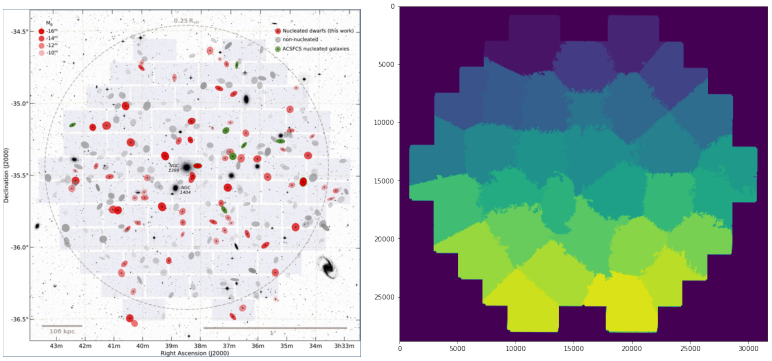


Figure 1: The left diagram depicts Tile 1, corresponding to Fornax's center. The specifications for Tile 1 and the LSB dwarf galaxies are in the [19]. The right diagram shows the result of reducing the size of Tile 1 in the g band into more manageable segments using SLIC. The outcome displays 57 irregular and independent segments distinguished by different colors.

To ensure that our segmentation process does not inadvertently truncate galaxies, especially faint ones that may go undetected or whose precise locations are unknown, we implement a strategy involving a guide galaxy from the g-band image of Central Fornax. We choose the NGFS J033754-353429 galaxy as it is the most diffuse and free from contamination by other sources among those listed in the catalog [19].

Our technique involves using the guide galaxy NGFS J033754-353429 to detect similar galaxies against the background. To do this, we set a threshold that allows the guide galaxy to be detected against the background. The threshold does not need to be exact, but it should be robust enough to reduce the size of the entire image or slab into more manageable sections. We have determined that the threshold for the image occurs when the intensity scale of the pixels in a monochromatic image (the grayscale) reaches the 57.5% percentile. This threshold is crucial as it indicates the point at which the guide galaxy becomes distinguishable from the background. Once we have determined this threshold, we use it to remove the sky background. After that, we perform the segmentation using SLIC on the image with robust detections.

The detections created after removing the sky guide the SLIC segmentation algorithm, ensuring it does not inadvertently divide galaxies like NGFS J033754-353429. It is important to note that SLIC could potentially divide galaxies only if the grid guide used for segmentation is smaller than the flat figures detected in the image. However, we prevent this situation by consistently using a segment size of 263×263 arcseconds, ensuring that galaxies remain intact. It is also

relevant to highlight that very small or empty segments indicating the absence of sources can be generated, and in such cases, the code issues warning messages. For the complete segmentation code and implementation details, please refer to this document's appendix A in Section A.2.1.

2.1.1 *Weight map*

Weight maps are frames of the same size as the images in which objects are detected or measured, and they serve to characterize the noise intensity present in each pixel. By doing so, weight maps effectively mitigate errors linked to instrumental variations. In our context, we are primarily concerned with the search for LSBGs, which often hover near the detection limit. In such cases, the noise from the sky background becomes particularly problematic, as it can approach the LSBG intensity levels. This presents a substantial challenge in detecting LSBGs, as it becomes difficult to differentiate the source signal from the sky background due to the comparable intensities and higher error rates associated with these faint objects.

Given the complexities of accurately determining the sky threshold, we have employed a convolution operation between the astronomical image and the corresponding weight map. This convolution operation holds several advantages in our approach. Most notably, it serves to suppress the influence of unreliable pixels. When establishing the sky background, any irregularities or noise present in the image are effectively damped, allowing the true intensities of the astronomical sources to emerge more distinctly. The mathematical computation for this convolution is elaborated upon in sample listing 2.

Listing 2: Image convolution

```
weight_total = np.sum(image_weight.ravel())
weight = image_weight/weight_total
fusion = image_data*weight
```

It's worth highlighting that the value of `weight_total` is computed as the sum of all pixel values comprising the weight map. The image outcome of the convolution operation, referred to as `fusion`, is determined by multiplying the pixel intensities of the original astronomical image by their corresponding weights, which have been appropriately normalized to account for errors. This approach enables us to enhance the visibility of astronomical sources in the presence of noise and uncertainty, ultimately improving the accuracy of LSBG detection. In this way, the image to which the methodology for detecting LSBGs in the background will be applied is the `fusion` image instead of the original image.

2.1.2 *Intensity scaling*

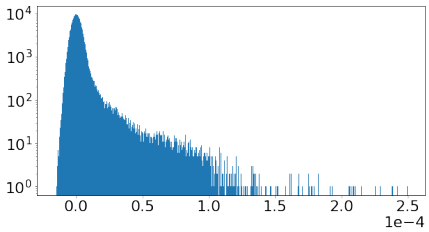
Setting an accurate sky value within each image segment is crucial in detecting astronomical sources. However, evaluating every single pixel value within the segment is not only unnecessary but also computationally intensive. Limiting the range of pixel values for calculating the background value is advantageous to streamline this process and enhance sky determination's efficiency. This approach reduces computation time and facilitates smoothing pixel variations in the region of interest, particularly at the transition between the sky background and the astronomical sources.

First, we rescaled the image based on minimum and maximum values. Then, we applied a percentile filter to exclude extreme pixel values within each segment. This streamlined the threshold detection process and enhanced the contrast between source and background pixels. After testing with various values, we determined that setting the filter ideal percentile value to $\pm 2\%$ was the best option. Specifically, we replace all pixel values that exceed 2% of the brightest pixels in the segment with the value of the upper percentile (corresponding to the 98% percentile). Similarly, pixel values falling below 2% of the least bright pixels are substituted with the value of the 2% percentile, establishing the lower limit. As a result, we assign each image segment a lower and upper limit for pixel values, determined based on the extreme values within that specific segment. The result of the entire scaling process is shown in Figure 2.

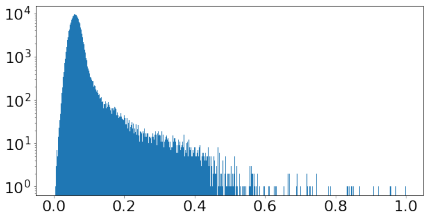
This approach ensures that the sky determination process is efficient and adaptive, allowing us to focus on the pixel values most representative of the sky background in each segment. By reducing the influence of extreme values, we can obtain more accurate and stable background estimates, which is essential for accurate LSBG detection in astronomical images. However, when using SLIC for segmentation, it is possible to encounter cases where segments are created that lack detectable objects. This may result in notifications of missing or undetected segments. However, it is important to note that these cases do not affect detection.

2.2 PROCESSING

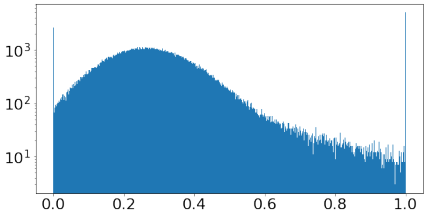
Detecting the faintest LSBGs presents a unique challenge, as their signals barely stand out from the background noise. This starkly contrasts with other sources or galaxies, which produce more detectable signals that can be readily adjusted based on their intensity profiles. LSBGs, on the other hand, exhibit signal discontinuities due to their low intensity, and when combined with the effects of High Surface Brightness Galaxies (HSBGs), the LSB galaxy signals often appear as mere noise. As a result, traditional detection methods relying on sur-



(a) Image threshold.



(b) Normalized image threshold.



(c) Image threshold with percentile filter.

Figure 2: The following figures show the image threshold manipulations for setting a background cut-off value.

face fits become inadequate, as they tend to miss irregular or faint LSBGs.

Recognizing these difficulties, we have developed an algorithm specifically designed to detect exceedingly faint LSBGs still perceptible to the human eye. Our approach is not limited to detecting dwarf LSBGs, which dominate the catalog [19, 41], but is also adaptable to detecting other LSBGs. Our detection algorithm centers around determining the optimal threshold value for the sky or background to remove it from the image, thereby revealing the LSBGs.

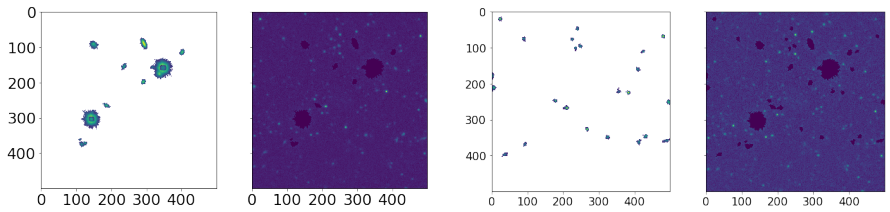
Various strategies can be employed to set the sky threshold for LSBG detection. One approach involves modeling the intrinsic noise of the background, but this method tends to incur longer computational times. Our method seeks sky threshold values while leveraging image segmentation to divide the image into distinct, independent segments. By doing so, we can autonomously determine the best-suited sky value for each segment. The sky adjustment process is rooted in estimating each segment's most appropriate grayscale value. This value correlates with the number of sources in the segment and their respective intensities. As a result, the optimal sky threshold setting may vary from one segment to another, ensuring that LSBGs are effectively detected within the image. This segment-specific approach allows us to achieve accurate and adaptive LSBG detection without extensive computational overhead.

To optimize the selection of the best sky threshold, we've developed an internally consistent approach that considers the size of the detection kernel from the Mahotas library [12]. Mahotas is a Python library tailored for Computer Vision tasks and offers functions that

allow the assignment of "Labels" to connected regions once a background threshold is established. The Mahotas kernel, or connection matrix, is a Boolean matrix that the algorithm employs to scan the image and assign labels to adjacent pixels belonging to the same group of connected objects. This association of labels with detections enables us to correlate our approach with the Mahotas kernel. The primary objective here is to eliminate bright sources from the image to enhance the detectability of LSBGs. The crucial factor influencing this process is the size of the kernel.

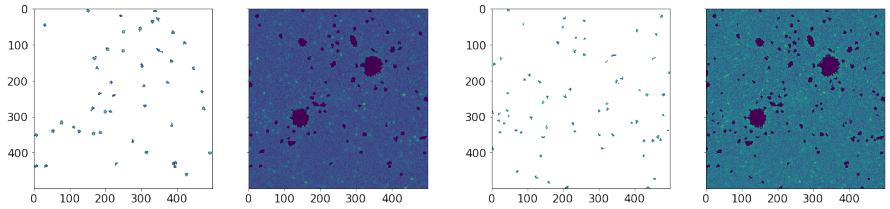
The size of the Mahotas kernel directly impacts the maximum number of detectable sources and, consequently, affects the sky background estimation. A larger kernel size is more effective at detecting sizable objects, which are often brighter but less numerous. Conversely, a smaller kernel is better suited for capturing minor fluctuations and, consequently, smaller or irregular objects, which tend to be more abundant. To strike a balance, we avoid starting with an overly large kernel size, as it would significantly increase computation time for detection. Instead of having a fixed kernel size, it was decided to make it dynamic. For this purpose, we chose the 90th percentile of pixel intensity as the provisional sky threshold, and with this sky threshold, we calculated the size, in pixels, of all detected objects. After evaluating the sizes of the objects in pixels detected using the provisional threshold, we determine the maximum size of the kernel for the segment through percentile analysis. In this context, we choose the pixel size of the kernel to be equal to the size of the smallest source in pixels, corresponding to the 90th percentile of the detected source sizes. With this maximum kernel size, we iteratively adjust the background threshold value again to maximize the possible detections within the constraints of the selected kernel size. Subsequently, the detected sources are systematically removed from the background, and the same process is repeated with the resulting image after removing the sources (see figure 2). This approach allows us to dynamically adapt the kernel size to optimize source detection, considering the variability in the sizes of objects within the image.

The shape of the detection kernel is circular, resulting in a Boolean matrix where true values form a circumscribed circle within the matrix. We opt for a circular distribution to ensure smooth edges in the detected objects, as the sources we aim to detect often possess convex signals, including irregular ones. We also experimented with different shapes, such as ellipses or stars, but they failed to detect irregular sources that were detected using the circular shape. We create this circular kernel in Python using the command `disk(r)`, where `r` represents the circle radius in pixels. The final detection kernel size, which we determined to be 29 pixels or 7.627 arcseconds (0.263 arcseconds/pixel resolution, DECam), corresponds to `disk(r=3)`. This represents the smallest detected size.



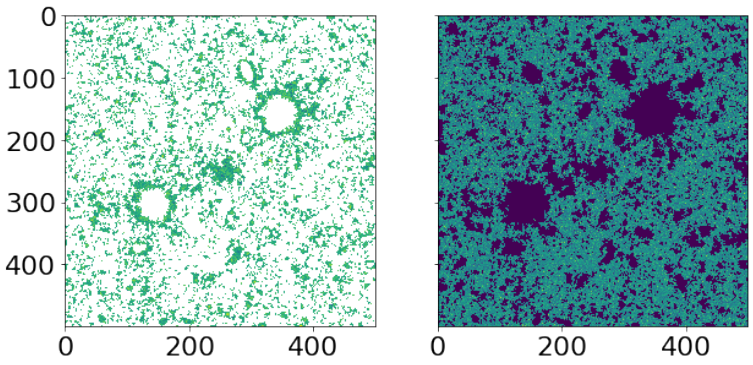
(a) kernel = 149

(b) kernel = 81



(c) kernel = 49

(d) kernel = 29



(e) Filter bilateral

Figure 3: The following figures illustrate how the source detection algorithm works by optimizing the kernel size and threshold adjustment. In the center of the image is the guide galaxy NGFS J033754-353429, which is affected by brightness contamination produced by the brightest sources. The detection algorithm starts with a tentative kernel size of 149 pixels, which is reduced in each iteration until reaching a minimum cutoff size of 29 pixels. Once the minimum size is reached, the bilateral filter is applied to the resulting sky after removing all detected bright sources. By applying the bilateral filter, a structure in the center of the image can finally be distinguished, representing the galaxy NGFS J033754-353429 from the catalog [19, 41]

.

After removing all contaminating bright sources from the image, gaps remain in the resulting sky, preserving the fainter sources that were previously undetectable. However, Mahotas alone struggle to detect the weakest LSBGs because they lack a well-defined shape, and the pixel connectivity within them is low compared to the sky background. These extremely faint LSBGs appear as diffuse objects without clear outlines. In its default mode, Mahotas only captures the

brightest portions of the LSBGs. We apply a blur "bilateral filter" filter to detect LSBGs, including their faint extensions comprehensively. This filter smoothens and outlines the brightest pixels, enhancing the connectivity between them and ultimately defining the shape of the source. It is crucial to note that this filter is utilized on the sky background after removing all bright sources, thereby maximizing the detection of the fainter sources.

We follow a process analogous to the background threshold optimization described previously to identify the correct values for the bilateral filter parameters. The 'd' parameter of the bilateral filter can be expressed as a function of 'σ' through the equation from `Scikt-image`, `d = int(max(5.2*np.ceil(3*i)+1))`. Consequently, the optimization procedure encompasses both the background threshold and the 'σ' parameters of the bilateral filter, which are adjusted based on the number of detected sources.

In the initial phase, we applied a bilateral filter with tentative values of 'σ' and 'd', similar to finding the threshold in the source detection with Mahotas. Subsequently, we calculated the background threshold using the Mahotas kernel on the blurred image to maximize the number of detected sources. The number of detections effectively depends on the parameters 'σ' and 'd' of the bilateral filter. Exploring 'σ' values from 1 to 10, we determined the 'σ' value that minimizes the highest number of detected sources. The filter blurs the image to interconnect elements and reduce detected sources. However, the bilateral filter saturates with a very high 'σ' value, ceasing to merge elements. Therefore, an optimal 'σ' value exists that effectively blurs and merges sources. It's important to note that the Mahotas kernel used during the sky optimization process with the bilateral filter remains fixed. The choice of kernel size in this case, `disk(r = 3)`, is suitable because the signals from detected sources are relatively small, rendering the use of a larger kernel unnecessary.

2.3 POST-PROCESSING

This section analyzes the data obtained in our search for LSBGs and the subsequent selection of LSB candidates. Our source detection algorithm generates an output where each segment's original image sky background has been successfully removed. This detection process occurs in multiple stages, which are influenced by the kernel size and operate top-down. After these step-wise detections, the resulting image is organized into a 3-dimensional matrix.

At the top of this matrix, we identify the sources detected with the largest kernel size, corresponding to the brightest sources. Moving up the matrix levels, we encounter sources detected with progressively smaller kernels until reaching the top, where the faintest and most diffuse sources reside. The detection of these extremely diffuse sources

requires the application of bilateral filtering. However, false detections are also present at the bottom of the matrix, where the faintest LSBG candidates are found. These false detections can occur due to galaxy interference, bright stars, image reduction residuals, and background noise. The main challenge at this stage is distinguishing genuine LSBG candidates from these false detections. Another issue in discerning LSBGs detected at the last level from false detections arises from gaps in the detection. These gaps lead to the low signal intensity of LSBGs because they are inherently faint and exhibit fluctuations in intensity, both maximum and minimum. To address the issue of over-detection caused by these discontinuities, we generated an image by aggregating all the detections from the matrix. This approach effectively fills the gaps in detection with information from previous detections, thereby reconstructing the LSBGs. Besides gap filling, this final step of creating a composite image facilitates the detection of the LSBGs' maximum extent. Recovering this maximum extent is crucial for characterizing LSBG detection using non-parametric features. However, it does have the drawback of potentially losing some of the previous LSBG detections when combined with halos or bleed-through from other bright sources.

To optimize the recovery of LSBGs, we designed a galaxy separation algorithm based on two criteria: contrast and morphology. The first criterion utilizes the watershed segmentation technique from the Ski-image library. In this approach, we apply watershed segmentation, but with the unique aspect of using initial detections as seeds before creating the composite image. The second criterion focuses on the morphology or shape of the detections, employing the watershed segmentation technique with a distance filter to select the seeds. Calibrating this separation through watershed segmentation poses significant challenges. While a small kernel effectively separates sources, it may not be effective for separating galaxies with high ellipticity, situations where more than one source is merged, or when there are extreme differences in the size of the sources. Comparing the results of the separation algorithm with the galaxies from the catalog by [19, 41], we successfully recovered 90% of the LSBGs listed in the catalog. This achievement represents a significant advancement in our search for LSB candidates.

While the watershed algorithm effectively aids in the recovery of LSBGs located near bright sources, the introduction of the galaxy separation algorithm does lead to an increase in the number of false detections. These false detections result from the watershed algorithm's limitations, where faint sources embedded in the bleed-through of bright sources are not entirely restored to their original forms. Instead, the algorithm may recover only a portion or section of these faint sources, rendering them unrecognizable as galaxies and categorizing them as false detections. No superior galaxy segmentation

algorithm can successfully separate and recover these faint sources without resorting to complex fits and profiles. However, employing such fitting techniques for detecting faint galaxies in large databases is impractical due to their high computational demands and non-automatable fit parameters. Non-automatability implies that targets must be pre-detected, and fit parameters must be manually validated for each case, posing significant challenges in image processing. While astronomy does employ non-parametric morphological parameters like Gini and M20, when used in detecting the LSBGs from [19, 41], they increase the number of false negatives compared to not using them. We have observed that these parameters are susceptible to contamination from other bright sources, making them overly restrictive features in our context, especially when dealing with diffuse and extensive LSBGs near small, bright sources. Nevertheless, we intend to incorporate these parameters in future code versions, aiming to enhance the robustness and accuracy of our morphological assessments.

Consequently, we have introduced alternative definitions of morphological parameters, drawing inspiration from a compilation of parameters designed initially for microscope image analysis, specifically for characterizing crystals and minerals [56, 35, 9, 45, 71]. Notably, the authors of these studies have had the freedom to define their concepts, which has led to multiple denominations for similar equations.

The selection of features used to describe the sources detected and to train the machine-learning classification algorithm is presented below. To facilitate a better understanding of these equations, it is crucial to establish the following concepts:

- Equivalent radius:

$$r' = \sqrt{\frac{A}{\pi}} \tag{1}$$

  Where A is the area of the Figure in pixels.

- Equivalent radius from convex object:

$$r'_{hull} = \sqrt{\frac{A_{hull}}{\pi}} \tag{2}$$

  Where $area_{hull}$, is the convex area of an object is the area of the convex hull that encloses the object [71, 45].

  The convex perimeter of an object is the perimeter of the convex hull that encloses the object.

- Eccentricity:

$$eccentricity_1 = \sqrt{1. - \left(\frac{m}{M}\right)^2} \tag{3}$$

  M and m correspond to the semi-major axis (M) and semi-minor axis (m) of a circumellipse source.

- Aspect ratio/Chunkiness/Elongation:

$$chunkiness = \frac{a}{b} \tag{4}$$

The aspect ratio or "chunkiness" measures the relationship between the height and width of a source inscribed in a rectangle. Here, a represents the maximum length of the rectangle, and b represents the maximum width of the rectangle.

- P-area:

$$PA = \frac{P}{A} \tag{5}$$

P-area is the ratio between the perimeter and the area. The perimeter (P) is the number of pixels in the boundary of the source [71].

- Surface Factor:

$$sf_2 = \frac{M}{m \times blk_2} \tag{6}$$

$$blk_2 = \frac{4\pi Mm}{A} \tag{7}$$

According to [56], is defined bulkiness (blk_2). M and m are the semi-axes from the ellipse circumellipse.

- Geodesic length:

$$xlg = \frac{P + \sqrt{P^2 - 16A}}{4} \tag{8}$$

Geodesic length is defined in [45].

- Straightness:

$$stss = \frac{2M}{xlg} \tag{9}$$

Straightness is defined in [45]. M is the semi-major axis of the ellipse circumellipse, which derives the maximum length of the object.

- Fiber thickness:

$$xe = \frac{P - \sqrt{P^2 - 16A}}{4} \tag{10}$$

Fiber thickness is a measure of degrees of curl. It is defined in [71, 45]

- Bulkiness:

$$blk_1 = \frac{ab}{A} \tag{11}$$

$$blk_2 = \frac{4\pi Mm}{A} \tag{12}$$

[56] makes a distinction in the definition of "Bulkiness" (blk_2) and "Bulkiness Factor" (blk_1). Where a and b correspond to the sides of a rectangle containing the source.

- Convexity/roughness/circularity:

$$cvx = \frac{P_{hull}}{P} \tag{13}$$

Convexity (cvx) is defined in [71, 45, 56]. P_{hull} is the perimeter of the convex area y P is the object's perimeter.

- Solidity/sphericity/circularity/compactness:

$$slty = \frac{A_{hull}}{A} \tag{14}$$

The solidity (slty) corresponds to a measure of the density of an object. It is defined in [71, 45, 56].

- Circularity/roundness/compactness:

$$cir_1 = \frac{4\pi A}{P^2} \tag{15}$$

$$cir_2 = \frac{4\pi A_{hull}}{P_{hull}^2} \tag{16}$$

$$cir_3 = \frac{4\pi A}{P_{hull}^2} \tag{17}$$

$$cir_5 = \frac{r'}{R} \tag{18}$$

These concepts are explained in [71, 45, 56]. R is the radius of the circumscribed circle of the object.

- Roundness:

$$rdss_1 = \frac{4A}{\pi M^2} \tag{19}$$

$$rdss_2 = \frac{r'_{hull}}{M} \tag{20}$$

$$rdss_3 = \frac{2r'_{hull}}{M + m} \tag{21}$$

$$rdss_4 = \frac{P_{hull}}{\pi R} \tag{22}$$

  These concepts are explained in [71, 45, 56]

- Sphericity:

$$sph = \frac{(36\pi V)^{1/3}}{A} \tag{23}$$

  It is in [56]. V is the volume of a sphere of radius r'.

2.3.1 *One-class SVM*

With all the detected sources reconstructed and separated, our next step is the identification of LSB dwarf galaxies using a Machine Learning algorithm. This choice stems from our understanding that some false detections are virtually indistinguishable from genuine LSBGs due to their intensity. Armed with the catalog of LSBGs from Fornax [19, 40] and visually identified LSBGs from the Fornax tiles, we opted for a supervised algorithm.

The challenge with supervised algorithms lies in determining which parameters or features should represent the detected sources and how to train the algorithm effectively. Because detection preserves particular source morphology, we have chosen non-parametric morphological parameters as our features. These parameters possess the advantage of rapid calculation since they don't rely on elaborate settings. As we move from representing the image information of each detected source to numerical datasets, we shift from computer vision challenges to tasks involving database management and machine learning. This entails handling outliers, normalization, and testing the machine learning algorithm.

Our initial step is to reduce the sample of detected sources, as there are many detected sources and false positives (in the case of tile 1, a

total of 4 million sources were detected) in which we are not interested. We aim to narrow the sample to sources similar to LSBGs (Low Surface Brightness Galaxies) in the reference catalogs [19, 40]. We use extreme values of morphological non-parametric features extracted from the LSBGs in the catalog to achieve this initial data reduction. This allows us to effectively separate LSBGs from false positives, facilitating the subsequent training of the machine-learning algorithm. Reducing the sample size not only decreases computation time but also helps eliminate outliers that might affect the quality of the results. Once the sample size is reduced, we perform scaling and normalization of each feature (non-parametric morphological parameters). We avoid using direct features like source intensity and equivalent radius, which can significantly constrain or bias the classifier. The scaling technique implemented is based on scikit-learn maximum and minimum scaling, ensuring that the features are in a specific range and comparable. This step prepares the data for further processing and enhances consistency in applying machine learning algorithms.

Once we had a clean and scaled database table of detected sources, we prepared and selected the LSBGs, i.e., the golden selection of LSBGs from the catalogs [19, 40], for training. Using coordinates and positions of the sources, we identified LSBGs from the catalog [19, 40] and those visually recognized by the team working on the Fornax tiles. In theory, there should be a total of 590 LSBGs available for training. The watershed source separation algorithm fails when multiple sources are merged within the same function, especially if there's a significant size difference between the sources. This results in galaxies' fragmentation, where the galaxies' morphological characteristics are no longer discernible or representative of LSBGs. These fragmented LSB galaxies also introduce noise into the sample. Due to these issues, the best 143 LSBGs were manually selected for training the algorithm.

The choice of which supervised machine-learning algorithm to use was based on our dataset-specific issues and characteristics and the objectives we aimed to achieve. Our dataset constitutes a single class, as no other information besides the training LSBGs is available. Moreover, it's an unbalanced dataset since the training LSBGs comprise only 0.26% of the total data. The training set also does not encompass all the morphological types of LSBGs, further complicating the situation. Given these challenges, the most suitable algorithm for our purposes, which is robust against unbalanced data and belongs to a single class, is the One-Class Support Vector Machine (One-Class SVM) available in scikit-learn.

The One-Class SVM is a supervised machine-learning method primarily used for outlier detection. It is well-suited for working with unbalanced datasets and single-class classification problems. The scikit-learn implementation of the One-Class SVM is based on Schölkopf's Support Vector Machine (SVM) algorithm. Schölkopf's One-Class SVM

operates by creating hyperplanes from data points and maximizing the distance from the center of these hyperplanes. Each data point is transformed into an n-dimensional vector, representing the hyperspace based on the selected features or F-hyperspace. The equation describes the hyperplanes:

$$W^T x - b = 0 \tag{24}$$

W is a normal vector belonging to the feature space F (note that it's not necessarily normalized). b is a real number, and x represents each point satisfying the hyperplane.

The role of hyperplanes in this context is to separate and define the dataset. They perform classification by assigning a value of $+1$ within a small region, which effectively captures the training data points and -1 outside this region. The classification occurs in a hyperspace, enabling data projection to exhibit nonlinear characteristics.

When dealing with Support Vector Machines (SVM) for multiple classes, the construction of the hyperplane aims to determine the margin between the classes, encompassing all the data points for each class. In contrast, in the case of the One-Class SVM, the hyperplane is designed to separate all the data points from the origin within the feature space F. It maximizes the distance from this hyperplane to the origin. This distinction alters the minimization process of the decision function, which is responsible for determining whether a data point x belongs to a class. In the case of One-Class SVM, this function is defined as:

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^{n} \alpha_i y_i K(x, x_i) + b\right) \tag{25}$$

Here α_i are the Lagrange multipliers; every $\alpha_i > 0$ is weighted in the decision function or supports. $K(x, x_i)$ is defined as the kernel.

The linear One-Class SVM model minimization function can be defined as:

$$\min_{w,b,\xi_i} \frac{\|w\|^2}{2} + \frac{1}{\nu n} \sum_{i=1}^{n} \xi_i - \rho \tag{26}$$

Subject to:

$$\begin{aligned} (w \cdot \phi(x_i)) &\geqslant \rho - \xi_i \quad \text{for all } i = 1, \ldots, n \\ \xi_i &\geqslant 0 \quad \text{for all } i = 1, \ldots, n \end{aligned}$$

This method creates a hyperplane characterized by w and ρ that maximizes the distance to the origin in the feature space F while effectively separating all data points from the origin. To introduce some

flexibility, we incorporate ξ_i to introduce some flexibility to permit specific data points to lie within the margin. The nonlinear function σ plays a pivotal role in delineating the decision boundary between one class and another.

In the context of the One-Class SVM provided by scikit-learn, the model fitting and learning process considers two parameters: ν and γ. ν is represented as ν and serves as a crucial parameter for the function. On the other hand, γ represents the default kernel parameter, which happens to be the radial-basis function kernel (RBF), denoted as follows:

$$K(x, x_i) = \exp\left(-\frac{\|x - x_i\|^2}{2\sigma^2}\right) \tag{27}$$

Intuitively, ν is an upper constraint on the fraction of training errors and a lower constraint on the fraction of support vectors. Essentially, it dictates the proportion of outliers we expect in our data [8, 61]. Notably, the ν value must fall within 0 to 1.

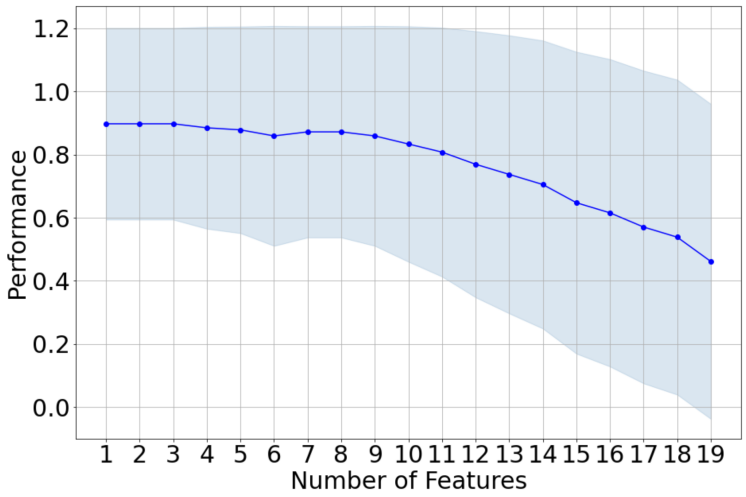
As for γ, it represents the inverse of σ [61]. γ acts as a parameter for the RBF kernel type and controls the influence of individual training samples, thus affecting the smoothness of the model [8, 61]. A low γ value enhances the model's generalizability, whereas a high value diminishes it while making it more sensitive to training data [8, 61]. Consequently, by manipulating these two parameters alone, we can adjust the One-Class model to attain the desired classification.

2.3.2 *Calibration and Validation*

Testing the One-Class SVM algorithm performance and evaluating the effectiveness of the characteristics in the classification process presented a significant challenge. There were limited freely available algorithms suitable for an unbalanced sample, mainly when working with a single class. To address this, a wrapper method was selected, which can be applied with scikit-learn One-Class SVM and integrated with the Leave-One-Out cross-validation technique. The chosen wrapper method is known as Sequential Feature Selection.

Sequential Feature Selection algorithms are a family of greedy search algorithms that aim to reduce an initial d-dimensional feature space to a k-dimensional feature subspace, where k is typically less than d. This process involves evaluating various subsets of features, ultimately producing a ranking of the best features for the classifier. The resulting feature ranking obtained through the wrapper method is illustrated in the Figure, showcasing the efficiency of the One-Class SVM classifier concerning the number of features utilized.

Figure 4 depicts the efficiency curve. This curve illustrates how the performance varies as a different number of features is utilized



(a)

Figure 4: Figure shows the result of the feature selection through Sequential feature selection algorithms. The y-axis shows the classifier efficiency, and the x-axis shows the number of features used in each test. In this figure, the efficiency of the classifier starts to decline after the use of 7 features, which are the features selected for the One-Class SVM.

in the classification process. Therefore, a higher efficiency curve indicates better model performance in classifying LSBGs. The curve gradually changes as different numbers and combinations of features are employed. Consequently, defining an exact cut-off point becomes challenging. Ideally, we aim to select high-efficiency features that effectively represent the LSBGs. To achieve this goal, we decided to use seven (7) features, representing the maximum number of features before a noticeable decrease in efficiency occurs. These selected features are as follows: $Eccentricity_1$ 3, $Chunkiness$ 4, $P-area$ 5, $Geodesic\ length$ 8, $Straightness$ 9, $Roundness_2$ 20, and $Roundness_3$ 21.

We employed the well-known f_1-score and $f_{0.5}$-score metrics to evaluate the One-Class SVM algorithm as a classifier. The F-score, or F-measure, is a reliable indicator of a test's accuracy. It is derived from the precision and recall of the test, effectively combining the predictive and analytical aspects of these two evaluations into a single value. One notable advantage of using the F-score is its robustness when dealing with unbalanced data.

The recall metric quantifies the machine learning model's ability to correctly identify instances of the positive class. It summarizes how effectively the model predicted the positive class.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{28}$$

Precision, conversely, indicates the quality of the machine learning model in classifying.

$$\text{Precision} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalsePositive}} \tag{29}$$

The f1-score is the harmonic mean of precision and recall; see equation 30.

$$f_1 = 2\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{30}$$

In the case of the f0.5-score, it is a variation of the F-measure with a higher weight in false positives, see equation 31. We consider this important since erroneous classifier results produce false detections.

$$f_{0.5} = \frac{5}{4}\frac{\text{Precision} \times \text{Recall}}{1/4\,\text{Precision} + \text{Recall}} \tag{31}$$

These metrics are combined with cross-validation to evaluate whether the one-class SVM algorithm is over-fitting or under-fitting new data. Cross-validation consists of dividing the training data into groups, to which the machine-learning algorithm is applied, and comparing the results of each group.

Different ways and methods exist to divide the sample and select the proportion of data training and tests. As we have a small and varied selection, the best way to do a cross-validation without losing information is to apply the Leave One Out Cross Validation (LOOCV) method, which is the same as the range of features. This Cross-validation approach leaves 1 data point out of the training data, and then, the remaining samples are used to train the model, the point being the validation of the model. This is repeated for all combinations where the original sample can be separated in this way, and then the error of all trials is averaged to obtain the overall efficiency.

3

RESULTS AND DISCUSSIONS

The following section presents the results of the process for searching and detecting candidate Low Surface Brightness Galaxies (LSBGs) in astronomical images. The outcome of our image source extraction algorithm yielded approximately 55,174 unclassified sources for the 19 tiles. This algorithm conducts a coordinate matching between the catalog [19, 41] and our identified sources. We successfully identified all 590 Low Surface Brightness Galaxies (LSBGs). However, due to contamination from bright sources, in some cases, the galaxy separation algorithm truncates the LSBG, leaving it with unrecognizable morphology due to false positives. In other cases, the LSBGs could not be separated from brighter sources. Of the 590 LSBGs, only the 143 most visually appealing and isolated ones were selected for the One-Class Support Vector Machine (SVM) training purposes.

The One-Class SVM, with fitting parameters $\nu = 0.1$ and $\gamma = 10$, detected 31,294 candidate Low Surface Brightness Galaxies (LSBGs) within the central Fornax region across 19 Fornax tiles. Our algorithm identified 31,295 LSBGs from these sources, and most of them are likely background galaxies. However, definitive identification is challenging without criteria related to galaxy distance. Furthermore, 80% of the 143 LSBGs found in the Fornax Cluster were correctly classified. Figure 8 displays a sample of these candidate galaxies.

The following plots depict features versus features, as shown in Figure 5. These visualizations enable us to understand the classification performed by the One-Class SVM algorithm within the space defined by the seven selected features discussed in this chapter. In Figure 5, all sources detected by the detection algorithm are represented as dark blue dots. In contrast, the cyan dots represent the candidate LSBGs predicted by the One-Class SVM algorithm. The red plus symbol $(+)$ represents LSBGs identified in the catalog [19, 41], which the Machine Learning algorithm correctly classifies as LSBGs, marking them as True Positives. Conversely, the red "x" symbol $(\times)$ represents the LSBGs that the algorithm rejects, indicating False Negatives, as shown in Figure 7.

We can analyze the data behavior upon examining Figure 6. At first glance, as illustrated in the figure, it becomes apparent that surface intensity (or flux) and equivalent radius alone are insufficient parameters to distinguish candidate LSBGs from false detections. We conjecture that when considering all the features, including surface intensity and equivalent radius, these parameters lose their effectiveness in identifying candidate LSBGs versus false detections. All selected

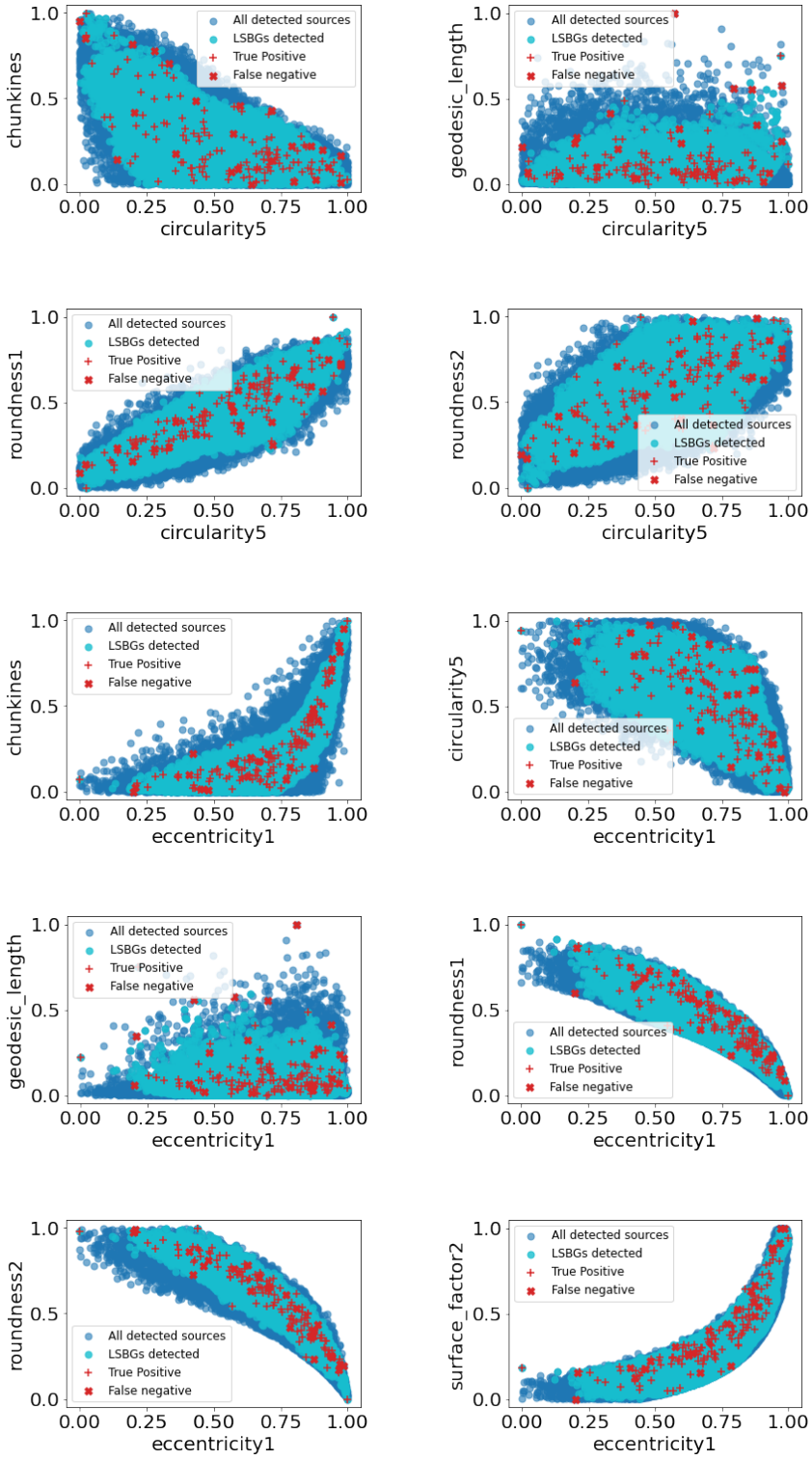


Figure 5

feature cut-offs already satisfy the criteria for classification as a LSBG. Hence, exploring additional features, studying feature rankings, and

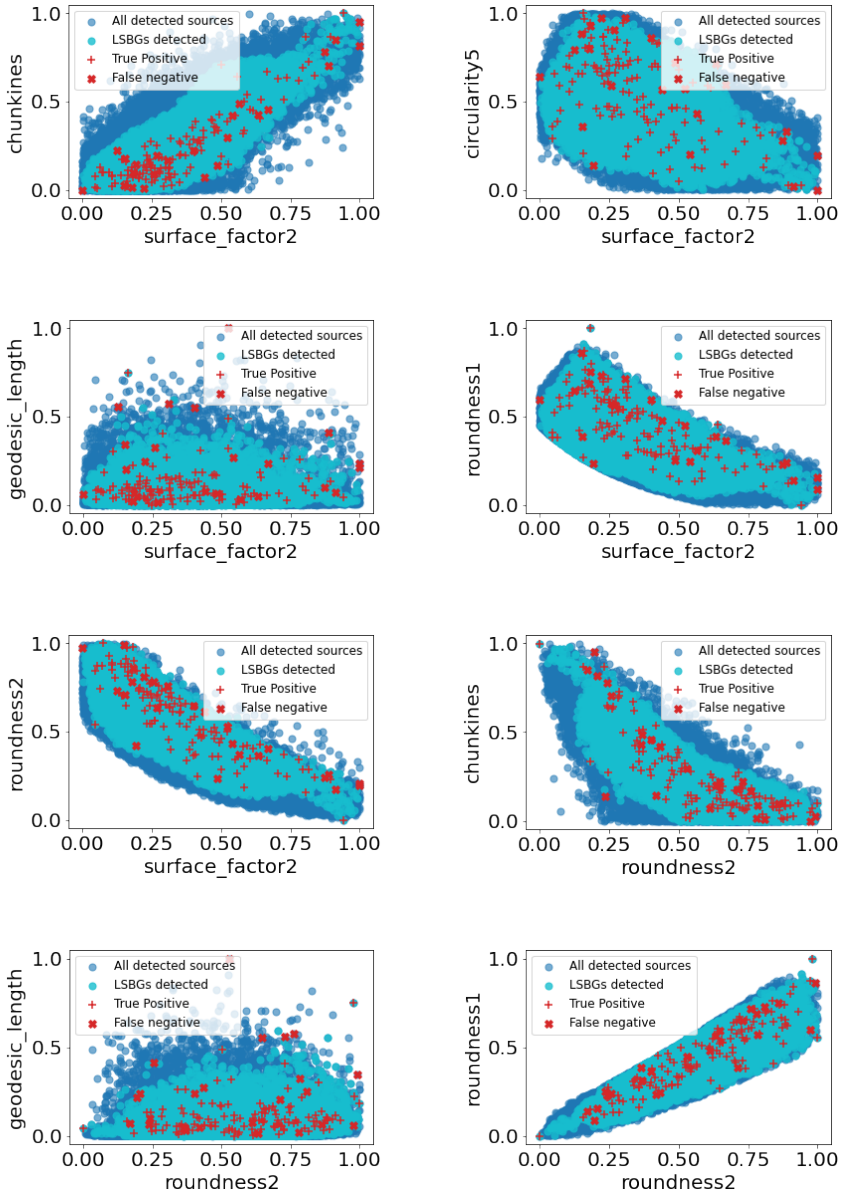


Figure 5: The following figures show the feature versus feature plots for the seven selected features. As in the figure, the blue dots show all Fornax source detection. The cyan dots show the LSBG candidate predictions. The plus red dots (+) show the LSBGs from the catalog [19, 41] retrieved. The red x dots show the discarded LSBGs from the catalog [19, 41]. The One-class SVM algorithm was run with $\nu = 0.1$ and $\gamma = 10$ parameters. The result of f_1 is 0.9165 ± 0.007 and $f_{0.5}$ is 0.96 ± 0.01.

identifying characteristics that can effectively differentiate candidate LSBGs from false detections or false positives becomes necessary.

Indeed, as observed in Figure 5, the selected non-parametric morphological features (143 LSBGs) in this chapter are well-suited for

representing and characterizing LSBGs, as the red crosses align neatly within the feature space, instead of being randomly scattered throughout the phase space. This outcome suggests that the selected features are relevant for identifying LSBGs. However, it is important to note that the classifier is not flawless, as a small percentage (16.8%) of catalog LSBGs were discarded, denoting False Negatives, as depicted in Figure 7.

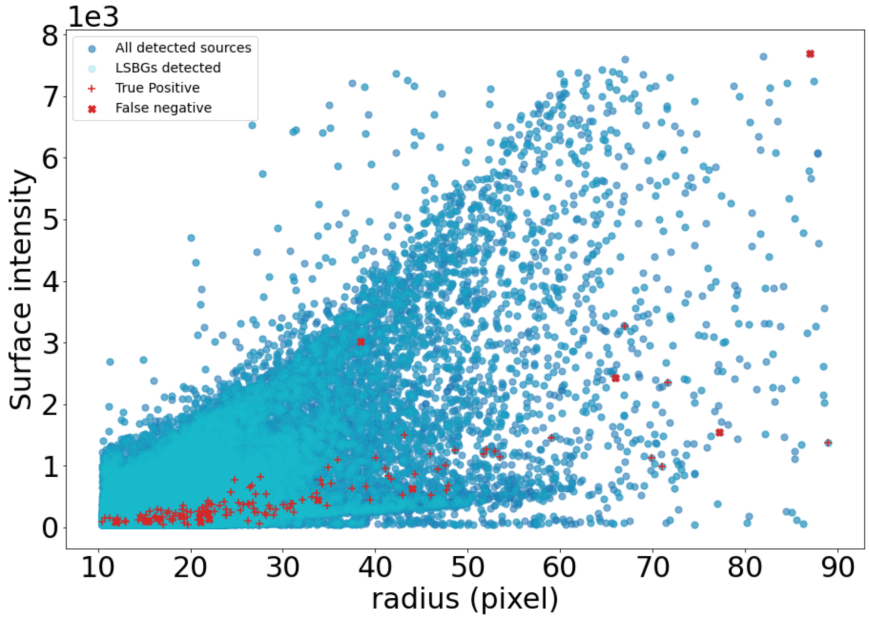


Figure 6: The surface intensity per pixel of all sources our code found is a function of the equivalent radius in pixel units. The blue dots show all Fornax source detections made by the detection algorithm. The cyan dots show the predictions of candidate LSBGs made by the One-Class SVM algorithm. The red plus (+) dots show the catalog [19, 41], which the One-Class SVM recovered. In contrast, the red x'dots show the LSB galaxies that the One-Class SVM classifies as false. The One-Class SVM algorithm was run with the parameters of $\nu = 0.1$ and $\gamma = 10$.

There are several factors contributing to the occurrence of false negatives in the classification process. The primary factor is the absence of a perfect classifier in any model, even within the One-Class model. This limitation is evident in the ν parameter, which inherently introduces a margin of error in delineating the boundaries of the classification space, as discussed in the previous chapter.

Another significant reason for model failures in prediction is the availability of training data. Many galaxies from the catalog [19, 41] had to be excluded due to contamination from small yet intensely bright sources. These sources, which the separation algorithm struggles to distinguish, are addressed in detail in the previous chapter. Moreover, the challenge of separating galaxies becomes more pro-

nounced when multiple mergers occur, resulting in the loss of LSBGs, particularly those with high ellipticity.

Out of the initial pool of 590 LSBGs, only 143 LSBGs from the catalog [19, 41] were deemed suitable for inclusion in the training dataset for the One-Class Support Vector Machine. Unfortunately, this limited number of examples is insufficient for the algorithm to train to identify LSBGs effectively. Additionally, the separation of galaxies introduces contamination, leading to false positives by compromising the morphology of the sources. As depicted in the accompanying figure, most discarded detections (false negatives) are galaxies with high ellipticity, highlighting a bias towards spheroidal LSBGs in the catalog.

This bias can be rectified as more LSBGs are detected and incorporated into the training dataset. Another strategy to reduce the incidence of false negatives is to explore alternative features. While our current evaluation in the chapter encompasses all the non-parametric morphological features available, ongoing research may uncover new features suitable for galaxy classification. For instance, we might consider incorporating the Gini coefficient and M20 feature.

It is worth noting that some contemporary approaches advocate for creating simulated or synthetic galaxies to expand the diversity of training samples, particularly within deep learning frameworks. We are actively considering implementing this approach in our future work, as it can potentially refine the criteria for selecting candidate LSBs and enhance the overall accuracy of our classification process.

Despite a small percentage of false negatives and the various challenges we encountered throughout our analysis, the overall outcome is remarkably promising. The metrics yield a commendable $f_{0.5}$ score of 0.78 ± 0.01. This score attests to the robustness of our approach, considering the complexities of the task at hand.

An additional class is usually necessary to quantify false positives and effectively represent anomalies. However, due to the inherent challenges in identifying LSBGs (Low Surface Brightness Galaxies) and distinguishing between anomalies and genuine candidates, creating a binary classification model with a dedicated class for anomalies becomes impractical. Therefore, future research should explore other additional constraints, such as color analysis, to discard LSBG candidates in a later process and refine the classification process. In this way, the goal is to enhance the accuracy and reliability of the LSBG detection model.

As a result, our final output is a catalog of candidate LSBGs detected by the One-Class SVM algorithm. This catalog is readily accessible on GitHub, along with the latest version of our code, providing a valuable resource for further research and exploration in the field.

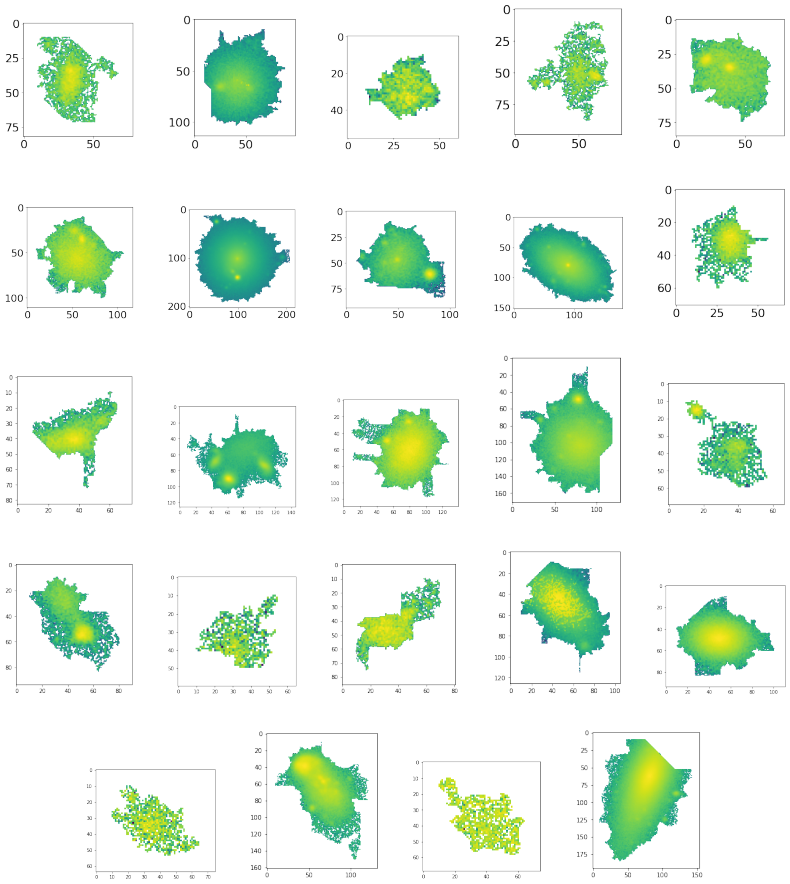


Figure 7: Sample of 24 LSBGs from the catalog [19, 41], which the One-Class SVM algorithm systematically classifies as false detection. These detection are false negatives.

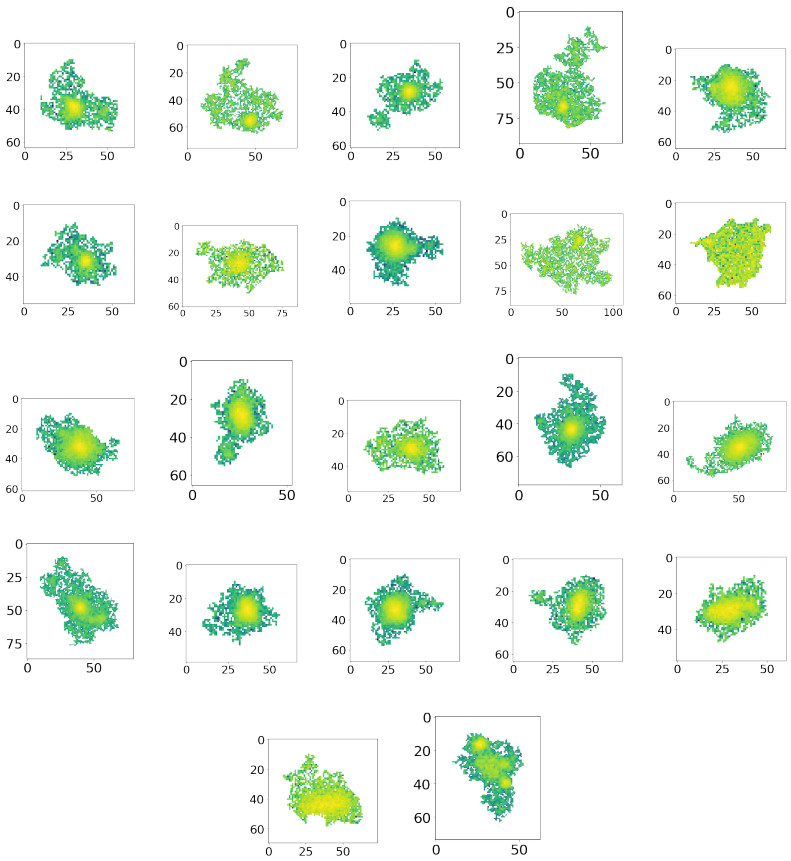


Figure 8: A sample of candidate LSBGs identified by OneClass SVM is presented. We recovered this sample from the central core of Fornax.

4

CONCLUSIONS

In conclusion, our implemented code successfully achieves the objective of automatically detecting Low Surface Brightness Galaxies (LSBGs) within digital images with a reasonable processing speed (5 days). We employed a comprehensive validation and testing procedure to assess its performance, including accuracy, speed, and robustness metrics.

Our SVM (One-Class Support Vector Machine) classifier was trained using a dataset of 143 manually selected LSBGs (golden sample) from sources detected in the Fornax galaxy cluster. The non-parametric morphoflological parameters used in the classifier are: $\mathtt{Eccentricity}_1$ 3, $\mathtt{Chunkiness}$ 4, $\mathtt{P\text{-}area}$ 5, $\mathtt{Geodesic\ length}$ 8, $\mathtt{Straightness}$ 9, $\mathtt{Roundness}_2$ 20, and $\mathtt{Roundness}_3$ 21. These seven parameters are the features resulting from a feature selection process carried out with 19 features, in which they proved to be the most relevant for classifying LSBGs [19, 41].

Detecting LSBGs has proven challenging due to their diverse morphologies and faint characteristics, which often push the limits of detection. In-depth discussions about these challenges and their implications for the broader field of astronomy and cosmology are crucial. We should explore potential strategies or improvements to address the challenges of bright source contamination and the difficulty of removing such contamination.

Looking ahead, we should outline specific directions for future improvements to the code and algorithms. Mentioning ongoing research or developments in the field that could enhance the performance of our detection method would provide valuable insights. Regarding the applicability of our code to other clusters of galaxies, we should delve into more detail about how adaptable it is for different environments. Are there any specific considerations or adjustments needed for its use in various galaxy clusters, and how might these adaptations affect its performance?

Lastly, we should expand on the broader implications of our code and its potential contributions to cosmological research, galaxy formation and evolution studies, and the understanding of galaxy clusters. Exploring how detecting LSBGs can advance these science areas will underscore the significance of our work. By addressing these areas, we can provide a more comprehensive and detailed set of conclusions that enhance the overall understanding of our research and its significance in astronomy and cosmology.

A

APPENDIX: CODE

This appendix presents examples of sections or routines from the final Python code called "Galaxy Mining". The complete code can be found on Github: "https://github.com/Alevhf/Galaxy-mining-"

The code is structured in a master or guide code that calls each subroutine. Each subroutine represents the steps to extract galaxies and identify galaxies. The following is an example of the subroutines of the code.

A.1 MASTER CODE

Listing 3: Master_code.py

```
name = pd.read_csv('data_img.csv')
data = [(int(i),name_img,name_weight,'tile_%s-%d'%(g,i),g) for i,
    name_img,name_weight,g in name[['tile','name_img','
    name_weight','band']].values]


for i,name_img,name_weight,tile,g in data:
    subprocess.run(["mkdir",tile],check=True)
    if not os.path.exists(tile):
        os.makedirs(tile)
    subprocess.run(["ipython","segments_slic.py",tile,g,name_img,
        name_weight,">","%s/seg_%s-%d.out"%(tile,g,i)],check=True
        )


for i,values in enumerate(data):
    n, name, namew, tile, g = values
    subprocess.run(["ipython", "galaxy_mining_nohup.py", str(tile
        ), str(g)],check=True)
    subprocess.run(["ipython", "pre_separar_galx.py", str(tile),
        str(g)],check=True)
    subprocess.run(["ipython", "separar_galx.py", str(tile), str(
        g)],check=True)
    subprocess.run(["ipython", "tabla_img.py", str(tile), str(g)
        ],check=True)

subprocess.run(["ipython", "OneClassSVM.py"],check=True)
```

A.2 PRE-PROCESSING

A.2.1 *Segment code*

Listing 4: segments_slic.py

```
def segmetation(image_data,labeled,i):
    astro = image_data.copy()
    label_galaxy = (labeled==i)
    x_min,x_max,y_min,y_max = mh.bbox(label_galaxy)
    cut = astro[x_min:x_max,y_min:y_max]
    mask = np.invert(label_galaxy[x_min:x_max,y_min:y_max])
    np.place(cut,mask=mask, vals=np.min(cut))
    cut = np.pad(cut, pad_width=3, mode='constant',
        constant_values=np.min(cut))
    return cut

def segmetation_fit(image_data,labeled,i):
    astro = image_data.copy()
    label_galaxy = (labeled==i)
    x_min,x_max,y_min,y_max = mh.bbox(label_galaxy)
    cut = astro[x_min:x_max,y_min:y_max]
    mask = np.invert(label_galaxy[x_min:x_max,y_min:y_max])
    np.place(cut,mask=mask, vals=np.min(cut))
    cut = np.pad(cut, pad_width=3, mode='constant',
        constant_values=np.min(cut))
    return cut, x_min-3, y_min-3

def segmetation_fit_seg(image_data,labeled,i):
    astro = image_data.copy()
    label_galaxy = (labeled==i)
    x_min,x_max,y_min,y_max = mh.bbox(label_galaxy)
    cut = astro[x_min:x_max,y_min:y_max]
    cut_label = label_galaxy[x_min:x_max,y_min:y_max]
    mask = np.invert(cut_label)
    np.place(cut,mask=mask, vals=np.min(cut))
    return cut, cut_label, x_min, y_min

import sys
import os
import subprocess


os.environ['MKL_NUM_THREADS'] = '1'
os.environ['OPENBLAS_NUM_THREADS'] = '1'
os.environ['OMP_NUM_THREADS'] = '1'

colors = list(map(plt.cm.jet,range(0, 256, 2)))
random.shuffle(colors)
rmap = c.ListedColormap(colors)
```

```
tile = sys.argv[1]
g  = sys.argv[2]

name_mask = sys.argv[3]
name_weight = sys.argv[4]

image_data   = fits.getdata(name_mask)
image_data = np.nan_to_num(image_data, nan=0.0, posinf=0.0,
    neginf=0.0)

image_weight = fits.getdata(name_weight)
image_weight = np.nan_to_num(image_weight, nan=0.0, posinf=0.0,
    neginf=0.0)

weight_total = np.nansum(image_weight.ravel())
weight = image_weight/weight_total
fusion = image_data*weight

mask = image_data != 0
mask = mh.close_holes(mask)

image = fusion.copy()
image = image - np.nanmin(image)
image[np.invert(mask)] = 0

n = np.percentile(image[mask],57.5)
image = image > n
image = mh.close_holes(image)
labeled,m = mh.label(image)

img = labeled > 0
img = img.astype(np.uint8)

segments_slic = slic(img, n_segments=54, compactness=0.01, sigma
    =3.0, enforce_connectivity=True, channel_axis=None, slic_zero
    =True)

segments_slic[np.invert(mask)] = 0
segments_slic,_ = mh.labeled.relabel(segments_slic)
seg = np.unique(segments_slic)[1::]


np.save('%s/segments_slic%s_weight_new.npy'%(tile,g),
    segments_slic)

for i in seg:
    part_sky, part_seg, x0, y0 = segmetation_fit_seg(image_data,
        segments_slic,i)
    np.savez('%s/part_sky_%s_%d.npz'%(tile,g,i), part_sky=
        part_sky, part_seg=part_seg, x0=x0, y0=y0)
```

```
    part_sky_f, part_seg, x0, y0 = segmetation_fit_seg(fusion,
        segments_slic,i)
    np.savez('%s/part_sky_%s_weight_%d.npz'%(tile,g,i), part_sky=
        part_sky_f, part_seg=part_seg, x0=x0, y0=y0)
```

A.3 PROCESSING

A.3.1 *Galaxy mining code*

Listing 5: galaxy_mining_nohup.py

```
def segmetation(p,l,i):
    astro = p.copy()
    label_galaxy = (l==i)
    x_min,x_max,y_min,y_max = mh.bbox(label_galaxy)
    cut = astro[x_min:x_max,y_min:y_max]
    mask = np.invert(label_galaxy[x_min:x_max,y_min:y_max])
    np.place(cut,mask=mask, vals=np.nanmin(cut))
    cut = np.pad(cut, pad_width=3, mode='constant',
        constant_values=np.nanmin(cut))
    return cut

def segmetation_fit(p,l,i):
    astro = p.copy()
    label_galaxy = (l==i)
    x_min,x_max,y_min,y_max = mh.bbox(label_galaxy)
    cut = astro[x_min:x_max,y_min:y_max]
    mask = np.invert(label_galaxy[x_min:x_max,y_min:y_max])
    np.place(cut,mask=mask, vals=np.nanmin(cut))
    cut = np.pad(cut, pad_width=3, mode='constant',
        constant_values=np.nanmin(cut))
    return cut, x_min-3, y_min-3

def segmetation_fit_seg(p,l,i):
    astro = p.copy()
    label_galaxy = (l==i)
    x_min,x_max,y_min,y_max = mh.bbox(label_galaxy)
    cut = astro[x_min:x_max,y_min:y_max]
    cut_label = label_galaxy[x_min:x_max,y_min:y_max]
    mask = np.invert(cut_label)
    np.place(cut,mask=mask, vals=np.nanmin(cut))
    return cut, cut_label, x_min, y_min

def kernel_fit(p,l,s):
        astro = p.copy()
        ravel = astro[l]
        thresholding = np.nanpercentile(ravel,95)
        img =  astro > thresholding
```

```
        img = morphology.remove_small_objects(img,min_size=13,
            connectivity=2)
        img = mh.close_holes(img)
        labeled,n = mh.label(img)
        sizes = mh.labeled.labeled_size(labeled)[1::]
        if len(sizes) > 0:
                kernel_sizes = np.nanpercentile(sizes,98)
                Bc = disk(int(np.sqrt(kernel_sizes/np.pi)))>0
                min_size = int(np.count_nonzero(Bc))
                if s == None:
                        return Bc, min_size
                else:
                        if s <= min_size:
                                Bc = disk(int(np.sqrt(s/np.pi)
                                    -0.5))
                                min_size = int(np.count_nonzero(
                                    Bc))
                                return Bc, min_size
                        else:
                                return Bc, min_size
        else:
                return disk(3)>0, 29

def sky_simple(p,Bc):
        if len(p) == 0:
                return 0., 0.
        try:
                thresholding = np.arange(np.nanmin(p),
                                         np.nanmax(p) + np.
                                             nanmax(p)/1000.,
                                         np.nanmax(p)/1000.)
                n_obj = []
                x = []
                for i in thresholding:
                        img = p > i
                        labeled,n = mh.label(img,Bc)
                        n_obj.append(n)
                        x.append(i)
                x = np.array(x)
                n_obj = np.array(n_obj)
                return x[np.argmax(n_obj)], n_obj[np.argmax(n_obj
                    )]
        except:
                return 0., 0.

def source_part0(part_sky,part_seg,pixel_size):
        result_up = part_sky.copy()
        result_down = part_sky.copy()
        ravel = part_sky[part_seg]
        ravel = np.nan_to_num(ravel, nan=0.0, posinf=0.0, neginf
            =0.0)
        q3 = np.nanpercentile(ravel,98)
```

```
        q1 = np.nanpercentile(ravel,2)
        percentil = part_sky.copy()
        percentil[percentil>q3] = q3
        percentil[percentil<q1] = q1
        percentil = percentil - np.nanmin(percentil)
        percentil_rescale = exposure.rescale_intensity(percentil)
        percentil_rescale[np.invert(part_seg)] = np.nanmin(
            percentil_rescale)
        Bc, min_size = kernel_fit(percentil_rescale,part_seg,
            pixel_size)

        thresholding, n_max = sky_simple(percentil_rescale,Bc)
        if (thresholding == 0.) & (n_max == 0.):
                return 0, 0, 0,'bad'
        else:
                img =  percentil_rescale > thresholding
                img = morphology.remove_small_objects(img,
                    min_size=min_size, connectivity=1)
                img = mh.close_holes(img)

                labeled,n = mh.label(img, Bc=Bc)
                labeled_invert = np.invert(labeled != 0)

                np.place(result_down,mask=labeled,vals=np.nanmin(
                    part_sky))
                np.place(result_up,mask=labeled_invert,vals=np.
                    nanmin(part_sky))

                img_min = min(np.nanmin(result_up[part_seg]),np.
                    nanmin(result_down[part_seg]))
                result_up = result_up - img_min
                result_down = result_down - img_min


                if min_size >= 49:
                        return result_up, result_down, min_size,
                            'good'

                elif min_size >= 29 or min_size < 49:
                        return result_up, result_down, min_size,
                            'moderate'

                else:
                        return 0, 0, 0,'bad'

def find_ellipse(p,l,n):
    point_x = []
    point_y = []
    mayor = []
    minor = []
    ang = []
    for i in range(1,n+1):
```

```
        img, x_min, y_min = segmetation_fit(p,l,i)
        img = (img > np.nanmin(img))
        img = np.invert(mh.close_holes(img))*np.uint8(1)
        _,contours, hierarchy = cv2.findContours(img,2,2)
        cnt = contours[1]
        (x,y),(MA,ma),angle = cv2.fitEllipse(cnt)
        point_x.append(int(x+y_min))
        point_y.append(int(y+x_min))
        mayor.append(ma)
        minor.append(MA)
        ang.append(angle)
    return point_x,point_y, np.array(mayor), np.array(minor), np.
        array(ang)

def find_circicle(p,l,n):
    point_x = []
    point_y = []
    radii = []
    for i in range(1,n+1):
        img, x_min, y_min = segmetation_fit(p,l,i)
        img = (img > np.nanmin(img))
        img = np.invert(mh.close_holes(img))*np.uint8(1)
        _,contours, hierarchy = cv2.findContours(img,2,2)
        cnt = contours[1]
        (x,y),radius = cv2.minEnclosingCircle(cnt)
        radius = int(radius)
        point_x.append(int(x+y_min))
        point_y.append(int(y+x_min))
        radii.append(radius)
    return point_x,point_y, np.array(radii)

def find_rectangle(p,l,n):
    fmax = []
    fmin = []
    for i in range(1,n+1):
        img, x_min, y_min = segmetation_fit(p,l,i)
        img = img > np.nanmin(img)
        img = np.invert(mh.close_holes(img))*np.uint8(1)
        img = np.pad(img, pad_width=2, mode='constant',
            constant_values=1)
        _,contours, hierarchy = cv2.findContours(img,2,2)
        cnt = contours[1]
        xy,wh,ang = cv2.minAreaRect(cnt)
        fmax.append(max(wh))
        fmin.append(min(wh))
    return np.array(fmax), np.array(fmin)

def perimeter(p,l,n):
    astro = p.copy()
    per = []
    for i in range(1,n+1):
        cut = segmetation(astro,l,i)
```

```
        perimeter = mh.bwperim(cut>np.nanmin(cut),n=8)
        labeled_per, n = mh.label(perimeter,np.ones((3,3),np.bool
            ))
        sizes_per = mh.labeled.labeled_size(labeled_per)
        per.append(sizes_per[1])
    del astro
    gc.collect()
    return np.array(per).astype(float)

def convex_hull(p,l,n):
    astro = p.copy()
    area_hull = []
    primeter_hull = []
    for i in range(1,n+1):
        cut = segmetation(astro,l,i)
        convex = convex_hull_object((cut > np.nanmin(cut))*1,
            neighbors=4)
        labeled_hull, n = mh.label(convex, np.ones((3,3),np.bool)
            )
        sizes_hull = mh.labeled.labeled_size(labeled_hull)
        area_hull.append(sizes_hull[1])
        peri = mh.bwperim((labeled_hull>0),n=8)
        labeled_per, n = mh.label(peri,np.ones((3,3),np.bool))
        sizes_per = mh.labeled.labeled_size(labeled_per)
        primeter_hull.append(sizes_per[1])
    del astro
    gc.collect()
    return np.array(area_hull).astype(float), np.array(
        primeter_hull).astype(float)

def optimizar(p,ravel,Bc):
        sigma = np.arange(1.,4.6,0.1)
        re_sigma = []
        re_thresholding = []
        re_n = []
        window = []
        SC = np.std(p.ravel())#*1.25
        for i in sigma:
                d = int(max(5,2*np.ceil(3*i)+1))
                bilateral = cv2.bilateralFilter(p,d,SC,i)
                thresholding_max, n_obj_max = sky_simple(
                    bilateral,Bc)
                re_sigma.append(i)
                re_n.append(n_obj_max)
                re_thresholding.append(thresholding_max)
                window.append(d)

        re_n = np.array(re_n)
        return(re_sigma[np.argmin(re_n)],
                   re_thresholding[np.argmin(re_n)],
                   window[np.argmin(re_n)],
                   SC)
```

```
    def galaxy_mining(a):
        i,tile,g,name = a
        f = np.load(name,allow_pickle=True)
        part_sky = f['part_sky']
        part_seg = f['part_seg']
        x0 = f['xo']
        y0 = f['yo']
        background_img = []
        astro_img = []
        j = 0
        pixel_size = None
        while 1:
                astro, background, pixel_size, rate =
                    source_part0(part_sky,part_seg,pixel_size)
                if rate == 'good':
                        astro_img.append(astro)
                        background_img.append(background)
                        part_sky = background
                        j = j + 1
                elif rate == 'moderate':
                        astro_img.append(astro)
                        background_img.append(background)
                        break
                else:
                        break

        np.savez('%s/Image_astro_fragment_%s_%d_new.npz'%(tile,g,
            i),astro_img = astro_img)
        name = '%s/Image_background_fragment_%s_%d_new.npz'%(tile
            ,g,i)
        np.savez(name,background = background_img[-1], part_seg =
             part_seg, x0 = x0, y0 = y0)

        del part_sky, part_seg, x0, y0
        del astro, background, rate
        del astro_img, background_img
        gc.collect()

def source_part2(part_sky,part_seg,Bc,pixel_size):
        result_up = part_sky.copy()
        result_down = part_sky.copy()
        ravel = part_sky[part_seg]
        q3 = np.nanpercentile(ravel,98)
        q1 = np.nanpercentile(ravel,2)
        percentil = part_sky.copy()
        percentil[percentil>q3] = q3
        percentil[percentil<q1] = q1
        percentil = percentil - np.nanmin(percentil)
        percentil_rescale = exposure.rescale_intensity(percentil)
        percentil_rescale[np.invert(part_seg)] = np.nanmin(
            percentil_rescale)
```

```
        re_sigma, re_thresholding, d, SC = optimizar(
            percentil_rescale,ravel,Bc)
        bilateral = cv2.bilateralFilter(percentil_rescale,d,SC,
            re_sigma)
        img =  bilateral > re_thresholding
        img = morphology.remove_small_objects(img, min_size=
            pixel_size, connectivity=1)
        img = mh.close_holes(img)

        labeled,n = mh.label(img,Bc=Bc)
        labeled_invert = np.invert(labeled != 0)

        np.place(result_down,mask=labeled,vals=np.nanmin(
            result_down))
        np.place(result_up,mask=labeled_invert,vals=np.nanmin(
            result_up))
        result_up = result_up - np.nanmin(result_up)
        return result_up, result_down

def galaxy_mining2(a):
        i,tile,g,name = a
        f = np.load(name,allow_pickle=True)
        part_sky = f['background']
        part_seg = f['part_seg']
        x0 = f['xo']
        y0 = f['yo']
        background_img = []
        astro_img = []
        Bc = disk(3)>0
        pixel_size = 29 #int(np.count_nonzero(Bc))
        astro, background = source_part2(part_sky,part_seg,Bc,
            pixel_size)
        astro_img.append(astro)
        background_img.append(background)
        np.savez('%s/Image_astro_bilateral_%s_%d_new.npz'%(tile,g
            ,i),astro_img = astro_img)


def calc_chunksize(n_workers, len_iterable, factor=4):
    chunksize, extra = divmod(len_iterable, n_workers * factor)
    if extra:
        chunksize += 1
    return chunksize

tile = sys.argv[1]
g  = sys.argv[2]

filename = tile+'/segments_slic'+g+'_weight_new.npy'
print("Loading file:", filename)
segments_slic = np.load(filename)

seg = np.unique(segments_slic)[1::]
```

```
del segments_slic
gc.collect()

data = [(int(i),tile,g,'%s/part_sky_%s_weight_%d.npz'%(tile,g,i))
     for i in seg]

cpu = round(seg[-1]/2.+0.1) #mp.cpu_count()
pool = mp.Pool(processes = cpu)
ck = calc_chunksize(cpu, len(data))

pool.imap_unordered(galaxy_mining, data, chunksize=ck)
pool.close()
pool.join()


data = [(int(i),tile,g,'%s/Image_background_fragment_%s_%d_new.
    npz'%(tile,g,i)) for i in seg]

cpu = round(seg[-1]/2.+0.1) #mp.cpu_count()
pool = mp.Pool(processes = cpu)
ck = calc_chunksize(cpu, len(data))

pool.imap_unordered(galaxy_mining2, data, chunksize=ck)
pool.close()
pool.join()

print('Finalizo galaxy_mining')
```

A.4 POST-PROCESSING

A.4.1 *Pre-splitting galaxies code*

Listing 6: pre_separar_galx.py

```
def process_image_auto(s,imge):
    n,_,_ = imge.shape
    part = 0
    for l in range(n,0,-1):
        part = imge[l-1] + part
    return np.array(part)

tile = sys.argv[1]
g  = sys.argv[2]

segments_slic = np.load(tile+'/segments_slic'+g+'_weight_new.npy'
    )
seg = np.unique(segments_slic)[1::]
del segments_slic

dato = []
```

```
for s in seg:
    dato.append((int(s),tile,g,
                 '%s/Image_astro_fragment_%s_%d_new.npz'%(tile,g,
                     s),
                 '%s/Image_astro_bilateral_%s_%d_new.npz'%(tile,g
                     ,s)))

def make_mask_total_final(a):
    s,tile,g,name_img,name_bi = a
    try:
        img = np.load(name_img,allow_pickle=True)
        img = img['astro_img']
        img_bi = np.load(name_bi,allow_pickle=True)
        img_bi = img_bi['astro_img']
    except:
        print('Missing %s or %s'%(name_img,name_bi), flush=True)

    if len(img) == 0:
        print('Missing %s'%(name_img), flush=True)
    if len(img_bi) == 0:
        print('Missing %s'%(name_img), flush=True)

    elif len(img) == 1:
        img_bi = process_image_auto(img_bi['astro_img'])
        img_total = img_bi+img
        labeled,_ = mh.label(img_total)
        sizes = mh.labeled.labeled_size(labeled)
        labeled = mh.labeled.remove_regions(labeled, np.where(
            sizes < 29))
        np.savez('%s/full_mask_%d.npz'%(tile,s),labeled=labeled)

    elif len(img_bi) == 1:
        img = process_image_auto(img['astro_img'])
        img_total = img_bi+img
        labeled,_ = mh.label(img_total)
        sizes = mh.labeled.labeled_size(labeled)
        labeled = mh.labeled.remove_regions(labeled, np.where(
            sizes < 29))
        np.savez('%s/full_mask_%d.npz'%(tile,s),labeled=labeled)

    elif len(img) == 1 & len(img_bi) == 1:
        img_total = img_bi+img
        labeled,_ = mh.label(img_total)
        sizes = mh.labeled.labeled_size(labeled)
        labeled = mh.labeled.remove_regions(labeled, np.where(
            sizes < 29))
        np.savez('%s/full_mask_%d.npz'%(tile,s),labeled=labeled)

    else:
        img = process_image_auto(img['astro_img'])
        img_bi = process_image_auto(img_bi['astro_img'])
        img_total = img_bi+img
```

```
        labeled,_ = mh.label(img_total)
        sizes = mh.labeled.labeled_size(labeled)
        labeled = mh.labeled.remove_regions(labeled, np.where(
            sizes < 29))
        np.savez('%s/full_mask_%d.npz'%(tile,s),labeled=labeled)

ncpu = round(seg[-1]/2.+0.1)
pool = mp.Pool(ncpu) #mp.cpu_count()
pool.imap_unordered(make_mask_total_final, dato, chunksize=1)
pool.close()
pool.join()


dato = []
for s in seg:
    dato.append((int(s),tile,g,'%s/Image_astro_fragment_%s_%d_new
        .npz'%(tile,g,s)))

def make_seed(a):
    s,tile,g,name = a
    seed = 0
    maxx = 0
    try:
        f = np.load(name,allow_pickle=True)
        img = f['astro_img']

    except:
        print('Missing %s'%name, flush=True)

    if len(img) == 0:
        print('Missing %s'%name, flush=True)

    elif len(img) == 1:
        if len(np.unique(img)) == 1:
            print('No sources found %s'%name, flush=True)
        else:
            label,_ = mh.label(img)
            sizes = mh.labeled.labeled_size(label)
            label = mh.labeled.remove_regions(label, sizes<=49)
            label,_ = mh.labeled.relabel(label)

            seed = label + seed + maxx
            seed[seed==maxx] = maxx = np.max(seed)
            np.savez('%s/seeds_mask_%d.npz'%(tile,s),seed = seed)
    else:
        n,_,_ = imge.shape
        for i in range(n):
            label,_ = mh.label(img[i])
            sizes = mh.labeled.labeled_size(label)
            label = mh.labeled.remove_regions(label, sizes<=49)
            label,_ = mh.labeled.relabel(label)
```

```
            seed = label + seed + maxx
            seed[seed==maxx] = 0
            maxx = np.max(seed)

        np.savez('%s/seeds_mask_%d.npz'%(tile,s),seed = seed)


ncpu = round(seg[-1]/2.+0.1)
pool = mp.Pool(ncpu) #mp.cpu_count()
pool.imap_unordered(make_seed, dato, chunksize=1)
pool.close()
pool.join()
```

A.4.2 *Splitting galaxies code*

Listing 7: separar_galx.py

```
def f_importances(coef, names, top=-1):
    imp = coef
    imp, names = zip(*sorted(list(zip(imp, names))))

    # Show all features
    if top == -1:
        top = len(names)

    plt.barh(range(top), imp[::-1][0:top], align='center')
    plt.yticks(range(top), names[::-1][0:top])
    plt.show()


def segmetation(p,l,i):
    astro = p.copy()
    label_galaxy = (l==i)
    x_min,x_max,y_min,y_max = mh.bbox(label_galaxy)
    cut = astro[x_min:x_max,y_min:y_max]
    mask = np.invert(label_galaxy[x_min:x_max,y_min:y_max])
    np.place(cut,mask=mask, vals=np.min(cut))
    cut = np.pad(cut, pad_width=3, mode='constant',
        constant_values=np.min(cut))
    return cut

def segmetation_fit(p,l,i):
    astro = p.copy()
    label_galaxy = (l==i)
    x_min,x_max,y_min,y_max = mh.bbox(label_galaxy)
    cut = astro[x_min:x_max,y_min:y_max]
    mask = np.invert(label_galaxy[x_min:x_max,y_min:y_max])
    np.place(cut,mask=mask, vals=np.min(cut))
    cut = np.pad(cut, pad_width=5, mode='constant',
        constant_values=np.min(cut))
    return cut, x_min-5, y_min-5
```

```
def segmetation_box(p,l,i):
    astro = p.copy()
    label_galaxy = (l==i)
    x_min,x_max,y_min,y_max = mh.bbox(label_galaxy)
    cut = astro[x_min:x_max,y_min:y_max]
    mask = np.invert(label_galaxy[x_min:x_max,y_min:y_max])
    np.place(cut,mask=mask, vals=np.min(cut))
    cut = np.pad(cut, pad_width=3, mode='constant',
        constant_values=np.min(cut))
    return cut, x_min-3, y_min-3, x_max-3, y_max-3

def segmetation_fit_seg(p,l,i):
    astro = p.copy()
    label_galaxy = (l==i)
    x_min,x_max,y_min,y_max = mh.bbox(label_galaxy)
    cut = astro[x_min:x_max,y_min:y_max]
    cut_label = label_galaxy[x_min:x_max,y_min:y_max]
    mask = np.invert(cut_label)
    cut[mask] = np.min(cut)
    return cut, cut_label, x_min, y_min, x_max, y_max

def segmetation_box_seg(p,l,i):
    astro = p.copy()
    label_galaxy = (l==i)
    x_min,x_max,y_min,y_max = mh.bbox(label_galaxy)
    cut = astro[x_min:x_max,y_min:y_max]
    mask = np.invert(label_galaxy[x_min:x_max,y_min:y_max])
    np.place(cut,mask=mask, vals=np.min(cut))
    cut = np.pad(cut, pad_width=10, mode='constant',
        constant_values=np.min(cut))
    cut_label = np.pad(label_galaxy[x_min:x_max,y_min:y_max],
        pad_width=10, mode='constant', constant_values=np.min(cut
        ))
    return cut, cut_label, x_min-10, y_min-10

def fins_coord(img,t_coord,ex,ey):
    x0 = int(t_coord['min_xo'].values[0])
    y0 = int(t_coord['min_yo'].values[0])

    xn = t_coord['Point_ellipse_x'].values
    yn = t_coord['Point_ellipse_y'].values

    x = ex - x0
    y = ey - y0
    catalog = []
    for ig in img:

        igm = ig > np.min(ig)
        img = mh.close_holes(igm)
        labeled,n = mh.label(igm)
```

```
        labeled = morphology.remove_small_objects(labeled,
            min_size=9, connectivity=1)
        labeled_holes,n = mh.labeled.relabel(labeled)

        for k,l in zip(xn,yn):
            for i,j in zip(x,y):
                if i<=np.max(xn) and i>=0:
                    if j<=np.max(yn) and j>=0:
                        try:
                            if labeled[int(j),int(i)]==labeled[
                                int(l),int(k)] and labeled[int(j)
                                ,int(i)]!=0.:
                                a = (int(i+x0),int(j+y0),int(k+x0
                                    ),int(l+y0))
                                catalog.append(a)
                        except:
                            a = (int(i+x0),int(j+y0),int(k+x0),
                                int(l+y0))
    return catalog

def result_catalog(data):
    s,image_name,catalog_name,tabla_name = data

    name = 'ss_fornax_tile1_g_long_ALIGNi.003'
    header = fits.getheader(name+'.fits')
    w = WCS(header)

    Eigenthaler = pd.read_csv(catalog_name)
    era  = Eigenthaler['_RAJ2000']
    edec = Eigenthaler['_DEJ2000']
    ex,ey = w.all_world2pix(era,edec,0,ra_dec_order=True)
    ex = np.array([int(i) for i in ex])
    ey = np.array([int(i) for i in ey])

    tabla = pd.read_csv(tabla_name)
    columns = ['Point_x', 'Point_y', 'Point_circicle_x', '
        Point_ellipse_x',
               'Point_circicle_y', 'Point_ellipse_y','image','
                   segment','min_xo',
               'min_yo','Point_x_full', 'Point_circicle_x_full'
                   ,
               'Point_ellipse_x_full', 'Point_y_full', '
                   Point_circicle_y_full',
               'Point_ellipse_y_full','RA', 'Dec', 'RA2', 'Dec2
                   ', 'RA3', 'Dec3']

    t_coord = tabla[columns].iloc[tabla['segment'].values==s]

    del name, header, w, era, edec, tabla, columns
    gc.collect()

    img = process_image(s,image_name)
```

```
    catalogo = fins_coord(img,t_coord,ex,ey)
    return catalogo


def kernel_fit(p,l,s):
        astro = p.copy()
        ravel = astro[l]
        thresholding = np.percentile(ravel,95)
        img =  astro > thresholding
        img = morphology.remove_small_objects(img,min_size=13,
            connectivity=1)
        img = mh.close_holes(img)
        labeled,n = mh.label(img)
        sizes = mh.labeled.labeled_size(labeled)
        sizes = sizes[1::]
        kernel_sizes = np.percentile(sizes,98)
        Bc = disk(int(np.sqrt(kernel_sizes/np.pi)))>0
        min_size = int(np.count_nonzero(Bc))
        if s == None:
                return Bc, min_size
        else:
                if s <= min_size:
                        Bc = disk(int(np.sqrt(s/np.pi)-0.5))
                        min_size = int(np.count_nonzero(Bc))
                        return Bc, min_size
                else:
                        return Bc, min_size

def sky_simple(p,Bc):
    thresholding = np.arange(np.min(p),
                             np.max(p) + np.max(p)/1000,
                             np.max(p)/1000)
    n_obj = []
    x = []
    for i in thresholding:
        img = p > i
        labeled,n = mh.label(img,Bc)
        n_obj.append(n)
        x.append(i)
    x = np.array(x)
    n_obj = np.array(n_obj)
    return x[np.argmax(n_obj)], n_obj[np.argmax(n_obj)]


def find_sigma(a):
    p,SC,i,Bc = a
    d = int(max(5,2*np.ceil(3*i)+1))
    bilateral = cv2.bilateralFilter(p,d,SC,i)
    thresholding_max, n_obj_max = sky_simple(bilateral,Bc)
    r = (i, n_obj_max, thresholding_max, d)
    return r
```

```
def optimizar_paral(p,ravel,Bc,ncpu):
    sigma = np.arange(1.,10.1,0.1)
    SC = np.std(p.ravel())#*1.25

    pool = mp.Pool(ncpu) #mp.cpu_count()
    data = [(p,SC,i,Bc) for i in sigma]
    results = pool.map(find_sigma, data)

    pool.close()
    pool.join()

    re_sigma = np.array([i[0] for i in results])
    re_n = np.array([i[1] for i in results])
    re_thresholding = np.array([i[2] for i in results])
    window = np.array([i[3] for i in results])

    return(re_sigma[np.argmin(re_n)],
           re_thresholding[np.argmin(re_n)],
           window[np.argmin(re_n)],
           SC)



colors = list(map(plt.cm.jet,range(0, 256, 2)))
random.shuffle(colors)
rmap = c.ListedColormap(colors)

import multiprocessing as mp
import os

def calc_chunksize(n_workers, len_iterable, factor=4):
    chunksize, extra = divmod(len_iterable, n_workers * factor)
    if extra:
        chunksize += 1
    return chunksize

def separar_galx(o):
    sss,ttt,ggg,name_galxo,name_seeds,name_mask = o

    try:
        seed = np.load(name_seeds,allow_pickle=True)
        part = seed['seed']
    except:
        print('Missing %s'%name_seeds, flush=True)

    try:
        full_mask = np.load(name_mask,allow_pickle=True)
        labeled = full_mask['labeled']
    except:
        print('Missing %s'%name_mask, flush=True)

    part_sky = np.load(name_galxo,allow_pickle=True)
```

```
    result_u = part_sky['part_sky']
    cuto_label = part_sky['part_seg']
    x0 = part_sky['xo']
    y0 = part_sky['yo']
    result_u[labeled==0] = 0

    img_separada = np.zeros((result_u.shape))

    num  = 0
    ll_s = 0
    for p in np.unique(labeled)[1::]:
        g, g_mask, x_min, y_min, x_max, y_max =
            segmetation_fit_seg(result_u,labeled,p)
        m = part[x_min:x_max,y_min:y_max]
        m[np.invert(g_mask)]=0
        m,_ = mh.labeled.relabel(m)

        if len(np.unique(m)) == 1:
            l = g_mask*1
        else:
            l = watershed(g_mask, m, mask=g_mask)

        ll_s = l + num
        ll_s[ll_s==num]=0
        num = np.max(ll_s)

        img_separada[x_min:x_max,y_min:y_max] = ll_s +
            img_separada[x_min:x_max,y_min:y_max]

    np.savez('%s/first_separate_mask_%s_%d.npz'%(ttt,ggg,sss),
        img_separate = img_separada)


def separar_galx_morf(o):
    sss,ttt,ggg,name_galxo,name_mask = o
    try:
        full_mask = np.load(name_mask,allow_pickle=True)
        labeled = full_mask['labeled']
    except:
        print('Missing %s'%name_mask, flush=True)

    img_separate = np.load(name_galxo,allow_pickle=True)
    result_u = img_separate['img_separate']
    result_u[labeled==0] = 0

    img_separada = np.zeros((result_u.shape))

    num = 0
    ll_s = 0
    for p in np.unique(labeled)[1::]:
        g, g_mask, x_min, y_min, x_max, y_max =
            segmetation_fit_seg(result_u,labeled,p)
```

```
        distance = ndi.distance_transform_edt(g_mask)
        sizes = mh.labeled.labeled_size(g_mask)
        sizes = sizes[1::]

        radii = 20 # np.sqrt(sizes/np.pi/100)[0]
        local_maxi = mh.locmax(distance, Bc = disk(int(radii))) #
            disk(int(radii/2)))
        local_maxi[np.invert(g_mask)] = False
        local_maxi = local_maxi*1
        markers,_ = mh.label(local_maxi,Bc = disk(int(radii)))

        l = watershed(g_mask, markers, mask=g_mask)

        ll_s = l + num
        ll_s[ll_s==num]=0
        num = np.max(ll_s)

        img_separada[x_min:x_max,y_min:y_max] = ll_s +
            img_separada[x_min:x_max,y_min:y_max]
    np.savez('%s/second_separate_mask_%s_%d.npz'%(ttt,ggg,sss),
        img_separate = img_separada)

tile = sys.argv[1]
g  = sys.argv[2]

segments_slic = np.load('%s/segments_slic%s_weight_new.npy'%(tile
    ,g))
seg = np.unique(segments_slic)[1::]
del segments_slic
gc.collect()

dato = []
for s in seg:
    try:
        name1 = '%s/part_sky_%s_weight_%d.npz'%(tile,g,s)
        name2 = '%s/seeds_mask_%d.npz'%(tile,s)
        name3 = '%s/full_mask_%d.npz'%(tile,s)
        dato.append((int(s),tile,g,name1,name2))
    except:
        print('Missing %s or %s'%(name2,name3), flush=True)

cpu = round(seg[-1]/2.+0.1) #mp.cpu_count()
pool = mp.Pool(processes = cpu)
ck = calc_chunksize(cpu, len(dato))
pool.imap_unordered(separar_galx, dato, chunksize=ck)
pool.close()
pool.join()

dato = []
for s in seg:
    try:
```

```
        name1 = '%s/part_sky_%s_weight_%d.npz'%(tile,g,s)
        name3 = '%s/first_separate_mask_%s_%d.npy'%(tile,g,s)
        dato.append((int(s),tile,g,name1,name3))
    except:
        print('Missing %s'%(name3),flush=True)


cpu = round(seg[-1]/2.+0.1) #mp.cpu_count()
pool = mp.Pool(processes = cpu)
ck = calc_chunksize(cpu, len(dato))
pool.imap_unordered(separar_galx_morf, dato, chunksize=ck)
pool.close()
pool.join()
```

A.4.3 *Table code*

Listing 8: tabla_img.py

```
def segmetation(p,l,i):
    astro = p.copy()
    label_galaxy = (l==i)
    x_min,x_max,y_min,y_max = mh.bbox(label_galaxy)
    cut = astro[x_min:x_max,y_min:y_max]
    mask = np.invert(label_galaxy[x_min:x_max,y_min:y_max])
    np.place(cut,mask=mask, vals=np.min(cut))
    cut = np.pad(cut, pad_width=3, mode='constant',
        constant_values=np.min(cut))
    return cut

def segmetation_fit(p,l,i):
    astro = p.copy()
    label_galaxy = (l==i)
    x_min,x_max,y_min,y_max = mh.bbox(label_galaxy)
    cut = astro[x_min:x_max,y_min:y_max]
    mask = np.invert(label_galaxy[x_min:x_max,y_min:y_max])
    np.place(cut,mask=mask, vals=np.min(cut))
    cut = np.pad(cut, pad_width=5, mode='constant',
        constant_values=np.min(cut))
    return cut, x_min-5, y_min-5

def segmetation_fit_seg(p,l,i):
    astro = p.copy()
    label_galaxy = (l==i)
    x_min,x_max,y_min,y_max = mh.bbox(label_galaxy)
    cut = astro[x_min:x_max,y_min:y_max]
    cut_label = label_galaxy[x_min:x_max,y_min:y_max]
    mask = np.invert(cut_label)
    np.place(cut,mask=mask, vals=np.min(cut))
    return cut, cut_label, x_min, y_min

def segmetation_box_seg(p,l,i):
```

```
    astro = p.copy()
    label_galaxy = (l==i)
    x_min,x_max,y_min,y_max = mh.bbox(label_galaxy)
    cut = astro[x_min:x_max,y_min:y_max]
    mask = np.invert(label_galaxy[x_min:x_max,y_min:y_max])
    np.place(cut,mask=mask, vals=np.min(cut))
    cut = np.pad(cut, pad_width=10, mode='constant',
        constant_values=np.min(cut))
    cut_label = np.pad(label_galaxy[x_min:x_max,y_min:y_max],
        pad_width=10, mode='constant', constant_values=np.min(cut
        ))
    return cut, cut_label, x_min-10, y_min-10

def find_ellipse(p,l,n):
    point_x = []
    point_y = []
    mayor = []
    minor = []
    ang = []
    for i in range(1,n+1):
        try:#1
            img, x_min, y_min = segmetation_fit(p,l,i)
            img,_ = mh.label(img > np.min(img))
            sizes = mh.labeled.labeled_size(img)
            img = mh.labeled.remove_regions(img,np.where(sizes <=
                 5))
            img = np.invert(mh.close_holes(img))*np.uint8(1)
            _,contours, hierarchy = cv2.findContours(img,2,2)
            cnt = contours[1]
            (x,y),(MA,ma),angle = cv2.fitEllipse(cnt)
            point_x.append(int(x+y_min))
            point_y.append(int(y+x_min))
            mayor.append(ma)
            minor.append(MA)
            ang.append(angle)
        except:
            point_x.append(int(0))
            point_y.append(int(0))
            mayor.append(0)
            minor.append(0)
            ang.append(0)
    return point_x,point_y, np.array(mayor), np.array(minor), np.
        array(ang)

def find_circicle(p,l,n):
    point_x = []
    point_y = []
    radii = []
    for i in range(1,n+1):
        try:#2
            img, x_min, y_min = segmetation_fit(p,l,i)
            img,_ = mh.label(img > np.min(img))
```

```
            sizes = mh.labeled.labeled_size(img)
            img = mh.labeled.remove_regions(img,np.where(sizes <=
                 5))
            img = np.invert(mh.close_holes(img))*np.uint8(1)
            _,contours, hierarchy = cv2.findContours(img,2,2)
            cnt = contours[1]
            (x,y),radius = cv2.minEnclosingCircle(cnt)
            radius = int(radius)
            point_x.append(int(x+y_min))
            point_y.append(int(y+x_min))
            radii.append(radius)
        except:
            radius = int(0)
            point_x.append(int(0))
            point_y.append(int(0))
            radii.append(0)
    return point_x,point_y, np.array(radii)

def find_rectangle(p,l,n):
    fmax = []
    fmin = []
    for i in range(1,n+1):
        try:#3
            img, x_min, y_min = segmetation_fit(p,l,i)
            img,_ = mh.label(img > np.min(img))
            sizes = mh.labeled.labeled_size(img)
            img = mh.labeled.remove_regions(img,np.where(sizes <=
                 5))
            img = np.invert(mh.close_holes(img))*np.uint8(1)
            img = np.pad(img, pad_width=2, mode='constant',
                constant_values=1)
            _,contours, hierarchy = cv2.findContours(img,2,2)
            cnt = contours[1]
            xy,wh,ang = cv2.minAreaRect(cnt)
            fmax.append(max(wh))
            fmin.append(min(wh))
        except:
            fmax.append(0)
            fmin.append(0)
    return np.array(fmax), np.array(fmin)

def perimeter(p,l,n):
    astro = p.copy()
    per = []
    for i in range(1,n+1):
        try:#4
            cut = segmetation(astro,l,i)
            perimeter = mh.bwperim(cut>np.min(cut),n=8)
            labeled_per, n = mh.label(perimeter,np.ones((3,3),np.
                bool))
            sizes_per = mh.labeled.labeled_size(labeled_per)
            per.append(sizes_per[1])
```

```
        except:
            per.append(0)
    del astro
    gc.collect()
    return np.array(per).astype(float)

def convex_hull(p,l,n):
    astro = p.copy()
    area_hull = []
    primeter_hull = []
    for i in range(1,n+1):
        try:#5
            cut = segmetation(astro,l,i)
            convex = convex_hull_object((cut > np.min(cut))*1,
                connectivity=1)
            labeled_hull, n = mh.label(convex, np.ones((3,3),np.
                bool))
            sizes_hull = mh.labeled.labeled_size(labeled_hull)
            area_hull.append(sizes_hull[1])
            peri = mh.bwperim((labeled_hull>0),n=8)
            labeled_per, n = mh.label(peri,np.ones((3,3),np.bool)
                )
            sizes_per = mh.labeled.labeled_size(labeled_per)
            primeter_hull.append(sizes_per[1])
        except:
            area_hull.append(0)
            primeter_hull.append(0)
    del astro
    gc.collect()
    return np.array(area_hull).astype(float), np.array(
        primeter_hull).astype(float)

def calc_chunksize(n_workers, len_iterable, factor=4):
    chunksize, extra = divmod(len_iterable, n_workers * factor)
    if extra:
        chunksize += 1
    return chunksize

def process_image_auto(s,image_name):
    imge = np.load(image_name, allow_pickle=True)
    n,_,_ = imge.shape
    part = 0

    for l in range(n,0,-1):
        part = imge[l-1] + part
    return np.array(part)

def shape_features_total(p,q):
    astroo = q.copy()
    astroo = astroo - np.min(astroo)
    labeled = p.copy()
    labeled = labeled.astype(int)
```

```
astroo[labeled==0] = 0
astro = astroo
sizes = mh.labeled.labeled_size(labeled)
labeled = mh.labeled.remove_regions(labeled,np.where(sizes <=
     5))
labeled,n = mh.labeled.relabel(labeled)
q[labeled==0] = 0
astro = q
index = np.unique(labeled)[1::]
point = mh.center_of_mass(astro,labeled)
point = point[1::]
point_x = point[:,1]
point_y = point[:,0]
point_cir_x, point_cir_y, radius = find_circicle(astro,
    labeled,n)
point_ell_x, point_ell_y, mayor, minor, ang = find_ellipse(
    astro,labeled,n)
sizes = np.array(mh.labeled.labeled_size(labeled)[1::]).
    astype(float)
sizes_per = perimeter(astro,labeled,n)
sizes_hull, sizes_per_hull = convex_hull(astro,labeled,n)
radii = np.sqrt(sizes/np.pi)
radii_hull = np.sqrt(sizes_hull/np.pi)
volum_cir = 4.*np.pi*radii*radii*radii/3.
a, b = find_rectangle(astro,labeled,n)
volum_ellip = 4.*np.pi*mayor*mayor*minor/3.
e = np.sqrt(1.-(minor/mayor)**2.)
s_ellip = np.pi*(2*mayor*mayor+(minor*minor/e)*np.log((1.+e)
    /(1.-e)))
chunkines = a/b
cpa = sizes_per/sizes
dp = sizes_per_hull/np.pi
bulkiness = a*b/sizes
bulkiness2 = 4*np.pi*mayor*minor/sizes
sf2 = mayor/minor/bulkiness2
convexity = sizes_per_hull/sizes_per #Roughness
Solidity = sizes/sizes_hull
circularity = 4*np.pi*sizes/sizes_per/sizes_per # Thinnes
    Ratio - Sphericity
circularity2 = 4*np.pi*sizes_hull/sizes_per_hull/
    sizes_per_hull
circularity3 = 4*np.pi*sizes/sizes_per_hull/sizes_per_hull #
    roundness
circularity4 = sizes_per/2./np.pi/radii
circularity5 = radii/radius
roundness = 4.*sizes/np.pi/mayor/mayor
roundness2= radii_hull/mayor
roundness3= 4.*radii_hull/(2.*mayor+2.*minor)
roundness4 = dp/radius
Sphericity = (36*np.pi*volum_cir)**(1./3.)/sizes
```

```
    xlg = [0]*len(sizes)
    xe  = [0]*len(sizes)
    straightness = [0]*len(sizes)
    for s,per,k in zip(sizes,sizes_per,range(len(sizes))):
        if per*per-16*s>= 0:
            xlg[k] = (per+np.sqrt(per*per-16*s))/4.
            xe[k]  = (per-np.sqrt(per*per-16*s))/4.
            straightness[k] = mayor[k]/xlg[k]
        else:
            xlg[k] = 0.
            xe[k]  = 0.
            straightness[k] = 0.

    M20 = []
    Gini = []
    brightness = []
    brightnessm = []
    brightness20 = []
    brightness50 = []
    radii50 = []
    radii20 = []
    for p in index:
        cut, cut_label, x02, y02 = segmetation_box_seg(astro,
            labeled,p)
        y,x = cut.shape
        point_ell_xc, point_ell_yc, mayor, minor, ang =
            find_ellipse(cut,cut_label*1,1)
        point_ell_xc, point_ell_yc = point_ell_xc[0],
            point_ell_yc[0]

        brightnesm = np.mean(cut[cut_label])
        brightnesst = np.sum(cut[cut_label])
        sizes_c = len(cut[cut_label])
        norma = np.abs(brightnesm)*sizes_c*(sizes_c-1)

        gini_array = cut[cut_label]
        gini_ascending = {}
        gini_ascending['gini'] = gini_array
        gini_ascending = pd.DataFrame(gini_ascending)
        gini_ascending = gini_ascending.sort_values('gini',
            ascending=True)

        gini = 0.
        for i,f in enumerate(gini_ascending.gini.values):
            gini = gini + (2*(i+1)-sizes_c-1)*np.abs(f)
        ginif = float(gini)/float(norma)

        fi = []
        xl = []
        yl = []
        for i in range(x):
            for j in range(y):
```

```
        if cut[int(j),int(i)] != 0:
            fi.append(cut[int(j),int(i)])
            xl.append(i)
            yl.append(j)

m20table = {}
m20table['f'] = fi
m20table['x'] = xl
m20table['y'] = yl
m20table = pd.DataFrame(m20table)
m20table = m20table.sort_values('f', ascending=False)

brightness_50 = m20table['f'].iloc[m20table['f'].values>
    np.percentile(m20table['f'].values,50)].values
sizes_50 = len(cut[cut > min(brightness_50)])
brightness_50_sum = np.sum(cut[cut > min(brightness_50)])
radii_50 = np.sqrt(sizes_50/np.pi)

brightness_20 = m20table['f'].iloc[m20table['f'].values>
    np.percentile(m20table['f'].values,20)].values
sizes_20 = len(cut[cut > min(brightness_20)])
brightness_20_sum = np.sum(cut[cut > min(brightness_20)])
radii_20 = np.sqrt(sizes_20/np.pi)

M_total = 0
for s in range(len(m20table)):
    M_total =(M_total+
              m20table['f'].values[s]*((m20table['x'].
                  values[s]-point_ell_xc)**2+
                                       (m20table['y'].
                                           values[s]-
                                           point_ell_yc)
                                           **2))
M_20 = 0
m20table20 = m20table.iloc[m20table['f'].values>np.
    percentile(m20table['f'].values,20)]
for s in range(len(m20table20)):
    M_20 =(M_20+
           m20table20['f'].values[s]*((m20table20['x'].
               values[s]-point_ell_xc)**2+
                                      (m20table20['y'].
                                          values[s]-
                                          point_ell_yc)**2)
                                          )

M_20t = np.log10(M_20/M_total)
Gini.append(ginif)
M20.append(M_20t)
brightness.append(brightnesst)
brightnessm.append(brightnesm)
brightness20.append(brightness_20_sum)
brightness50.append(brightness_50_sum)
```

```
        radii50.append(radii_50)
        radii20.append(radii_20)

    brightness = np.array(brightness)
    sb = brightness/sizes
    sb_cir = brightness/4./np.pi/radii/radii
    sb_ellip = brightness/s_ellip
    dic = {'Label' : index,
           'Point_x' : point_x,
           'Point_y' : point_y,
           'Point_circicle_x' : point_cir_x,
           'Point_ellipse_x' : point_ell_x,
           'Point_circicle_y' : point_cir_y,
           'Point_ellipse_y' : point_ell_y,
           'radii' : radii,
           'area' : sizes,
           'area_convex' : sizes_hull,
           'P' : sizes_per,
           'P_convex' : sizes_per_hull,
           'intensity1' : brightness,
           'surface_intensity ' : sb,
           'surface_intensity_cir' : sb_cir,
           'surface_intensity_ellip' : sb_ellip,
           'volume_cir' : volum_cir,
           'volume_ellip' : volum_ellip,
           'Gini' : Gini,
           'M20' : M20,
           'eccentricity1' : e,
           'chunkines' : chunkines,
           'P-area' : cpa,
           'surface_factor2' : sf2,
           'geodesic_length' : xlg,
           'fiber_thickness' : xe,
           'straightness' : straightness,
           'bulkiness1' : bulkiness,
           'bulkiness2' : bulkiness2,
           'roughness1' : convexity,
           'solidity' : Solidity,
           'circularity1' : circularity,
           'circularity2' : circularity2,
           'circularity3' : circularity3,
           'circularity4' : circularity4,
           'circularity5' : circularity5,
           'roundness1' : roundness,
           'roundness2' : roundness2,
           'roundness3' : roundness3,
           'roundness4' : roundness4,
           'Sphericity_cir' : Sphericity,
           'brightness' : brightness,
           'brightnessm' : brightnessm,
           'brightness20' : brightness20,
           'brightness50' : brightness50,
```

```
            'radii20' : radii20,
            'radii50' : radii50}
    return pd.DataFrame(dic)

def tabla_new_total(a):
    s,tile,g,name,nameo = a
    img_separate = np.load(name,allow_pickle=True)
    ll = img_separate['img_separate']
    part_sky = np.load(nameo,allow_pickle=True)
    imgo_total = part_sky['part_sky']
    cuto_label = part_sky['part_seg']
    x0 = part_sky['xo']
    y0 = part_sky['yo']
    table_total = shape_features_total(ll,imgo_total)
    result = [0]*len(table_total)
    table_total['class'] = result
    label = [str(0)]*len(table_total)
    table_total['image'] = label
    ss = [str(s)]*len(table_total)
    table_total['segment'] = ss
    min_x0 = [y0]*len(table_total)
    table_total['min_xo'] = min_x0
    min_y0 = [x0]*len(table_total)
    table_total['min_yo'] = min_y0
    table_total['Point_x_full'] = table_total['Point_x'] + y0
    table_total['Point_circicle_x_full'] = table_total['
        Point_circicle_x'] + y0
    table_total['Point_ellipse_x_full'] = table_total['
        Point_ellipse_x'] + y0
    table_total['Point_y_full'] = table_total['Point_y'] + x0
    table_total['Point_circicle_y_full'] = table_total['
        Point_circicle_y'] + x0
    table_total['Point_ellipse_y_full'] = table_total['
        Point_ellipse_y'] + x0

    table_total.to_csv('%s/Segment_table_%s_%d.csv'%(tile,g,s),
        index = None)
    print('Finished %s/Segment_table_%s_%d.csv'%(tile,g,s),flush=
        True)

tile = sys.argv[1]
g  = sys.argv[2]


segments_slic = np.load('%s/segments_slic%s_weight_new.npy'%(tile
    ,g))
seg = np.unique(segments_slic)[1::]
del segments_slic
gc.collect()

dato = []
for s in seg:
```

```
    name1 = '%s/second_separate_mask_%s_%d.npz'%(tile,g,s)
    name2 = '%s/part_sky_%s_%d.npz'%(tile,g,s)
    try:
        dato.append((int(s),tile,g,name1,name2))
    except:
        print('Missing %s'%name1,flush=True)

ncpu = round(seg[-1]/2.+0.1)
pool = mp.Pool(ncpu) #mp.cpu_count()
pool.imap_unordered(tabla_new_total, dato, chunksize=1)
pool.close()
pool.join()
```

A.4.4 *Catalog creation*

Listing 9: catalog.py

```
name = 'ss_fornax_tile1_g_long_ALIGNi.003'
header = fits.getheader(name+'.fits')
w = WCS(header)
gc.collect()

catalog_name = 'Catalogos/Galaxies/Eigenthaler2018.fit'

Eigenthaler = Table.read(catalog_name)
Eigenthaler = Eigenthaler.to_pandas()
#print(Eigenthaler)

era  = Eigenthaler['_RAJ2000']
edec = Eigenthaler['_DEJ2000']
ex0,ey0 = w.all_world2pix(era,edec,0,ra_dec_order=True)
del name, header, w, era, edec
gc.collect()

coord = pd.DataFrame(data=np.c_[ex0, ey0], columns=['exo', 'eyo'
    ])
coord = coord.dropna()

ex0 = np.array([int(i) for i in coord.ex0.values])
ey0 = np.array([int(i) for i in coord.ey0.values])

co = []
x = 'morf'
for s in range(1,54):
    try:
        t_coord = pd.read_csv('t1/Tabla_final_tile1g%d_morf_new.
            csv'%s)
        img_total, cuto_label, y0, x0 = np.load('part_sky0%d.npy'
            %s,allow_pickle=True)

        ex = ex0 - x0
```

```
        ey = ey0 - y0

        labeled_total = np.load('mask_Image_astro_separada_morf_g
            %d_new.npy'%s,allow_pickle=True)
        sizes = mh.labeled.labeled_size(labeled_total)
        labeled_total = mh.labeled.remove_regions(labeled_total,
            np.where(sizes <= 5))
        labeled_total,n = mh.labeled.relabel(labeled_total)

        #labeled_total = labeled_total.astype(int)

        img_total[labeled_total==0]=0
        yn,xn = img_total.shape

        for i,j in zip(ex,ey):
            if (i >= 0) and (i < xn) and (j >= 0) and (j < yn):
                q = labeled_total[j,i]
                m = t_coord['Label'].iloc[t_coord['Label'].values
                    ==q].values
                if (q != 0) & (len(m)>0):
                    m = t_coord['Label'].iloc[t_coord['Label'].
                        values==q].values[0]
                    co.append((i,j,i+x0,j+y0,q,m,s))
                    print(i,j,i+x0,j+y0,q,m,s)
    except:
        print(s)

co = np.array(co)
co = pd.DataFrame(np.c_[co], columns=['ex','ey','x','y','s_t','
    Label','segment'])
co = co.sort_values('s_t')

co.to_csv('Tabla_co_Eigenthaler2018_mask_morf_new.csv', index =
    None)
```

A.5 RESULTS

Listing 10: Master_code.py

```
def one_table(i,tile,g):
    segments_slic = np.load('%s/segments_slic%s_weight_new.npy'%(
        tile,g))
    seg = np.unique(segments_slic)[1::]
    del segments_slic
    gc.collect()

    tabla_morf2 = {}
    for s in seg:
        try:
            tabla_morf2['tabla%d'%s] = pd.read_csv('%s/
                Segment_table_%s_%d.csv'%(tile,g,int(s)))
```

```
        except:
            name = '%s/Segment_table_%s_%d.csv'%(tile,g,int(s))
            print('Missing %s'%name,flush=True)

    tabla_morf2 = pd.concat(tabla_morf2, ignore_index=True, sort=
        False)
    tabla_morf2['mu_mean']  = tabla_morf2['brightnessm']/
        tabla_morf2['radii']
    tabla_morf2['mu_total'] = tabla_morf2['brightness']/
        tabla_morf2['radii']
    tabla_morf2['mu_20'] = tabla_morf2['brightness20']/
        tabla_morf2['radii20']
    tabla_morf2['mu_50'] = tabla_morf2['brightness50']/
        tabla_morf2['radii50']
    tabla_morf2['tile'] = [i]*len(tabla_morf2)
    return(tabla_morf2)

name = pd.read_csv('data_img.csv')
data = [(int(i),'tile_%s-%d'%(g,i),g) for i,g in name[['tile','
    band']].values]

tabla = {}
for i,tile,g in data:
        tabla['%d'%i] = one_table(i,tile,g)

tabla_morf = pd.concat(tabla, ignore_index=True, sort=False)
tabla_morf  = tabla_morf.replace([np.inf, -np.inf], np.nan,
    inplace=False)
tabla_morf  = tabla_morf.drop(['Gini', 'M20', 'Sphericity_cir', '
    brightness', 'brightnessm', 'radii20','radii50', 'mu_total',
    'mu_total', 'circularity4', 'fiber_thickness'], axis=1)
tabla_morf  = tabla_morf.dropna()

lsb_select = pd.read_csv('tabla_lsb_select.csv')
lsb_select['tile'] = [1]*len(lsb_select)
lsb_tiles_select = pd.read_csv('tiles_co.csv')
lsb_tiles_select = pd.merge(lsb_tiles, lsb_tiles_select, on=['
    Label','segment','tile'])
tabla_morfs = pd.merge(tabla_morf, lsb_select, on=['Label','
    segment','tile'])

col_id =  ['Label',
           'Point_x',
           'Point_y',
           'Point_circicle_x',
           'Point_ellipse_x',
           'Point_circicle_y',
           'Point_ellipse_y',
           'area',
           'area_convex',
           'P',
           'P_convex',
```

```
            'intensity1',
            'surface_intensity ',
            'surface_intensity_cir',
            'surface_intensity_ellip',
            'volume_cir',
            'volume_ellip',
            'Point_x_full',
            'Point_y_full',
            'Point_circicle_x_full',
            'Point_circicle_y_full',
            'Point_ellipse_x_full',
            'Point_ellipse_y_full',
            'class',
            'image',
            'segment',
            'min_x0', 'min_y0','tile']

col_names = tabla_morf.columns.values
col_names = col_names.tolist()

for i in col_id:
    col_names.remove(i)

col_good = col_names.copy()
col_mal  = ['mu_20','mu_50','brightness20','brightness50','radii'
    ]

for i in col_mal:
    col_good.remove(i)

col = col_good.copy()
col.remove('circularity1')
col.remove('circularity4')
col.remove('fiber_thickness')
col.remove('roundness1')

lsb_max = np.max(tabla_morfs[['mu_20','mu_50','mu_mean']+col])
lsb_min = np.min(tabla_morfs[['mu_20','mu_50','mu_mean']+col])

tabla_selec_lsb  = tabla_morf.copy()
tabla_selec_lsb0 = tabla_morf0.copy()

for c in ['mu_20','mu_50','mu_mean']+col:
    mask0 = (tabla_morf0[c] <= lsb_max[c]) & (tabla_morf0[c] >=
        lsb_min[c])
    tabla_morf0 = tabla_selec_lsb0.loc[mask0]

    mask = (tabla_morf[c] <= lsb_max[c]) & (tabla_morf[c] >=
        lsb_min[c])
    tabla_morf = tabla_selec_lsb.loc[mask]

mmx  = MinMaxScaler()
```

```
tabla_morf_mmx = mm.fit_transform(tabla_morf[col])
tabla_morf_mmx = pd.DataFrame(tabla_morf_mmx,columns=col)

for i in col_id+col_mal:
    tabla_morf_mmx[i] = tabla_morf[i].values

col = ['eccentricity1','chunkines','P-area','geodesic_length','
    straightness', 'roundness2','roundness3','roundness4']

test  = tabla_morf_mmx

knn_from_joblib = joblib.load('OneClassSVM.pkl')
train_scores = knn_from_joblib.predict(test[col])
test['class'] = train_scores
test_result = test.iloc[test['class'].values==1]

name = pd.read_csv('data_img.csv')
data = [(int(i),name_img,name_weight,'tile_%s-%d'%(g,i),g) for i,
    name_img,name_weight,g in name[['tile','name_img','
    name_weight','band']].values]

catalog = {}
for i,name_img,name_weight,tile,g in data:
    header1 = fits.getheader(name_img)
    w1 = WCS(header1)
    catalog01 = test_result.iloc[test_result['tile'].values==i]
    a = catalog01[['Point_x_full','Point_y_full']].values
    ex1 = a[:,0]
    ey1 = a[:,1]
    ra1, dec1 = w1.all_pix2world(ex1, ey1,0,ra_dec_order=True)
    dic1 = {'RA_J2000' : ra1, 'DE_J2000' : dec1}
    catalog['%d'%i] = pd.DataFrame(dic1)

catalog_final = pd.concat(catalog, ignore_index=True, sort=False)
catalog_final.to_csv('Final_resulting_catalog.csv', index = None)
```

B

APPENDIX: CATALOG

The Catalog, resulting from the Galaxy-mining-code, is uploaded on Github: https://github.com/Alevhf/LSB_candidates

BIBLIOGRAPHY

[1] Roberto G. Abraham and Pieter G. van Dokkum. Ultra-Low Surface Brightness Imaging with the Dragonfly Telephoto Array. , 126(935):55, January 2014. doi: 10.1086/674875.

[2] Aaron J Barth. A normal stellar disk in the galaxy malin 1. *The Astronomical Journal*, 133(3):1085, 2007.

[3] Emmanuel Bertin and Stephane Arnouts. Sextractor: Software for source extraction. *Astronomy and astrophysics supplement series*, 117(2):393–404, 1996.

[4] S Boissier, B Epinat, P Amram, BF Madore, A Boselli, J Koda, A Gil de Paz, JC Muños Mateos, L Chemin, et al. First spectroscopic study of ionised gas emission lines in the extreme low surface brightness galaxy malin 1. *Astronomy & Astrophysics*, 637: A21, 2020.

[5] Greg Bothun, Chris Impey, and Stacy McGaugh. Low-surface-brightness galaxies: hidden galaxies revealed. *Publications of the Astronomical Society of the Pacific*, 109(737):745, 1997.

[6] Gregory D. Bothun, Christopher D. Impey, David F. Malin, and Jeremy R. Mould. Discovery of a Huge Low-Surface-Brightness Galaxy: A Proto-Disk Galaxy at Low Redshift? , 94:23, July 1987. doi: 10.1086/114443.

[7] Ivan Bruha. From machine learning to knowledge discovery: Survey of preprocessing and postprocessing. *Intelligent data analysis*, 4(3-4):363–374, 2000.

[8] Chih-Chung Chang and Chih-Jen Lin. Training v-support vector classifiers: theory and algorithms. *Neural computation*, 13(9):2119–2147, 2001.

[9] A Chanou, GR Osinski, and RAF Grieve. A methodology for the semi-automatic digital image analysis of fragmental impactites. *Meteoritics & Planetary Science*, 49(4):621–635, 2014.

[10] Wei-Lun Chao. Machine learning tutorial. *Digital Image and Signal Processing*, 2011.

[11] AO Clarke, AMM Scaife, R Greenhalgh, and V Griguta. Identifying galaxies, quasars, and stars with machine learning: A new catalogue of classifications for 111 million sdss sources without spectra. *Astronomy & Astrophysics*, 639:A84, 2020.

[12] Luis Pedro Coelho. Mahotas: Open source software for scriptable computer vision. *Journal of Open Research Software*, 1, July 2013. doi: http://dx.doi.org/10.5334/jors.ac.

[13] Jonathan Ivor Davies. Visibility and the selection of galaxies. *Monthly Notices of the Royal Astronomical Society*, 244:8–24, 1990.

[14] Jonathan Ivor Davies, S Phillipps, MGM Cawson, Michael John Disney, and EJ Kibblewhite. Low surface brightness galaxies in the fornax cluster: automated galaxy surface photometry–iii. *Monthly Notices of the Royal Astronomical Society*, 232(2):239–258, 1988.

[15] Jonathan Ivor Davies, Michael John Disney, S Phillipps, BJ Boyle, and WJ Couch. A search for large low-surface brightness galaxies in deep ccd data. *Monthly Notices of the Royal Astronomical Society*, 269(2):349–364, 1994.

[16] MJ Disney. Visibility of galaxies. *Nature*, 263(5578):573–575, 1976.

[17] Wei Du, Hong Wu, Man I Lam, Yinan Zhu, Fengjie Lei, and Zhimin Zhou. Low surface brightness galaxies selected from the 40% sky area of the alfalfa h i survey. i. sample and statistical properties. *The Astronomical Journal*, 149(6):199, 2015.

[18] P-A Duc, Laura Ferrarese, J-C Cuillandre, Stephen Gwyn, Lauren A MacArthur, Etienne Ferriere, Patrick Côté, and Patrick Durrell. Faint dwarf galaxies in the next generation virgo cluster survey. *EAS Publications Series*, 48:345–350, 2011.

[19] Paul Eigenthaler, Thomas H Puzia, Matthew A Taylor, Yasna Ordenes-Briceño, Roberto P Muñoz, Karen X Ribbeck, Karla A Alamo-Martínez, Hongxin Zhang, Simón Ángel, Massimo Capaccioli, et al. The next generation fornax survey (ngfs). ii. the central dwarf galaxy population. *The Astrophysical Journal*, 855 (2):142, 2018.

[20] Richard S Ellis and David A Allen. Infrared colours of a complete sample of faint galaxies. *Monthly Notices of the Royal Astronomical Society*, 203(3):685–693, 1983.

[21] Rh Evans, Jonathan Ivor Davies, and S Phillipps. The colours of low surface brightness galaxies in the fornax cluster-automated galaxy surface photometry. vi. *Monthly Notices of the Royal Astronomical Society*, 245:164–174, 1990.

[22] Laura Ferrarese, Patrick Côté, Rúben Sánchez-Janssen, Joel Roediger, Alan W McConnachie, Patrick R Durrell, Lauren A MacArthur, John P Blakeslee, Pierre-Alain Duc, S Boissier, et al. The next generation virgo cluster survey (ngvs). xiii. the luminosity and mass function of galaxies in the core of the virgo cluster

and the contribution from disrupted satellites. *The Astrophysical Journal*, 824(1):10, 2016.

[23] KC Freeman. Historical introduction. In *International Astronomical Union Colloquium*, volume 171, pages 3–8. Cambridge University Press, 1999.

[24] Kenneth C Freeman. On the disks of spiral and so galaxies. *The Astrophysical Journal*, 160:811, 1970.

[25] Roberto E González, Roberto P Muñoz, and Cristian A Hernández. Galaxy detection and identification using deep learning and data augmentation. *Astronomy and computing*, 25:103–109, 2018.

[26] CD Impey, D Sprayberry, MJ Irwin, and GD Bothun. Low surface brightness galaxies in the local universe. i. the catalog. *The Astrophysical Journal Supplement Series*, 105:209, 1996.

[27] Chris Impey and Greg Bothun. Low surface brightness galaxies. *Annual Review of Astronomy and Astrophysics*, 35(1):267–307, 1997.

[28] MJ Irwin, Jonathan Ivor Davies, Michael John Disney, and S Phillipps. Automated galaxy surface photometry. v-detection of very low surface brightness galaxies. *Monthly Notices of the Royal Astronomical Society*, 245:289–304, 1990.

[29] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[30] ID Karachentsev, P Riepe, T Zilch, M Blauensteiner, M Elvov, P Hochleitner, B Hubl, G Kerschhuber, S Küppers, F Neyer, et al. New low surface brightness dwarf galaxies detected around nearby spirals. *Astrophysical Bulletin*, 70(4):379–391, 2015.

[31] CR King and RS Ellis. The evolution of spiral galaxies and uncertainties in interpreting galaxy counts. *The Astrophysical Journal*, 288:456–464, 1985.

[32] Mireille Louys, Benjamin Perret, Bernd Vollmer, François Bonnarel, Sebastien Lefèvre, and Ch Collet. Lsb galaxies detection using markovian segmentation on astronomical images. In *Astronomical Data Analysis Software and Systems XVII*, volume 394, page 125, 2008.

[33] Stacy McGaugh. Testing galaxy formation and dark matter with low surface brightness galaxies. *arXiv preprint arXiv:2103.05003*, 2021.

[34] Stacy S McGaugh, Gregory D Bothun, and James M Schombert. Galaxy selection and the surface brightness distribution. *arXiv preprint astro-ph/9505062*, 1995.

[35] Valdek Mikli, Helmo Kaerdi, Priit Kulu, and Michal Besterci. Characterization of powder particle morphology. *Proceedings of the Estonian Academy of Sciences: Engineering(Estonia)*, 7(1):22–34, 2001.

[36] Z Morshidi-Esslinger, Jonathan Ivor Davies, and RM Smith. An automated search for nearby low-surface-brightness galaxiesâi. the catalogue. *Monthly Notices of the Royal Astronomical Society*, 304(2):297–310, 1999.

[37] Z Morshidi-Esslinger, Jonathan Ivor Davies, and RM Smith. An automated search for nearby low-surface-brightness galaxiesâii. the discussion. *Monthly Notices of the Royal Astronomical Society*, 304(2):311–318, 1999.

[38] Roberto P Muñoz, Paul Eigenthaler, Thomas H Puzia, Matthew A Taylor, Yasna Ordenes-Briceño, Karla Alamo-Martínez, Karen X Ribbeck, Simón Ángel, Massimo Capaccioli, Patrick Côté, et al. Unveiling a rich system of faint dwarf galaxies in the next generation fornax survey. *The Astrophysical Journal Letters*, 813(1):L15, 2015.

[39] Y Ordenes-Briceno, P Eigenthaler, MA Taylor, TH Puzia, K Alamo-Martinez, KX Ribbeck, RP Munoz, H Zhang, EK Grebel, S Angel, et al. Vizier online data catalog: Ngfs. iii. dwarf galaxies in outer regions (ordenes-briceno+, 2018). *VizieR Online Data Catalog*, pages J–ApJ, 2019.

[40] Yasna Ordenes-Briceño, Matthew A Taylor, Thomas H Puzia, Roberto P Muñoz, Paul Eigenthaler, Iskren Y Georgiev, Paul Goudfrooij, Michael Hilker, Ariane Lançon, Gary Mamon, et al. Faint dwarf galaxies in hickson compact group 90. *Monthly Notices of the Royal Astronomical Society*, 463(2):1284–1290, 2016.

[41] Yasna Ordenes-Briceño, Paul Eigenthaler, Matthew A Taylor, Thomas H Puzia, Karla Alamo-Martinez, Karen X Ribbeck, Roberto P Muñoz, Hongxin Zhang, Eva K Grebel, Simón Ángel, et al. The next generation fornax survey (ngfs). iii. revealing the spatial substructure of the dwarf galaxy population inside half of fornax's virial radius. *The Astrophysical Journal*, 859(1):52, 2018.

[42] Jim R Parker. *Algorithms for image processing and computer vision*. John Wiley & Sons, 2010.

[43] S Phillipps, MJ Disney, EJ Kibblewhite, and MGM Cawson. The surface brightness of 1550 galaxies in fornax: automated galaxy surface photometry–ii. *Monthly Notices of the Royal Astronomical Society*, 229(4):505–515, 1987.

[44] Daniel J Prole, Jonathan I Davies, Olivia C Keenan, and Luke JM Davies. Automated detectionof very low surface brightness galaxiesin the virgo cluster. *Monthly Notices of the Royal Astronomical Society*, 478(1):667–681, 2018.

[45] US Technical Support QAQC. *SIZE SHAPE PARAMETERS DEFINITIONS FOR OCCHIO PRODUCTS*. OCCHIO SA, 4 rue des chasseurs ardennais, 4031 Angleur, Belgique, 2020. URL https://www.qclabequipment.com/OCCHIO_THEORY_PARAMETERS_DEFINITIONS_QC.pdf.

[46] Sarah Roberts, Jonathan Davies, Sabina Sabatini, Wim van Driel, Karen O'Neil, Maarten Baes, Suzanne Linder, Rodney Smith, and Rhodri Evans. A search for low surface brightness dwarf galaxies in different environments. *Monthly Notices of the Royal Astronomical Society*, 352(2):478–492, 2004.

[47] W Romanishin, KM Strom, and SE Strom. A study of low surface brightness spiral galaxies. ii optical surface photometry, infrared photometry, and h ii region spectrophotometry. *The Astrophysical Journal Supplement Series*, 53:105–128, 1983.

[48] S Sabatini, R Scaramella, V Testa, S Andreon, G Longo, G Djorgovski, and RR de Carvalho. Automated search for lsb galaxies. *arXiv preprint astro-ph/9909463*, 1999.

[49] Sabina Sabatini, Sarah Roberts, and Jon Davies. Dwarf lsb galaxies and their environment: The virgo cluster, the ursa major cluster, isolated galaxies and voids. In *Galaxy Evolution in Groups and Clusters*, pages 97–106. Springer, 2003.

[50] A Sandage and B Binggeli. Studies of the virgo cluster. iii-a classification system and an illustrated atlas of virgo cluster dwarf galaxies. *The Astronomical Journal*, 89:919–931, 1984.

[51] R Scaramella and S Sabatini. Finding lsbs from pixels. *Memorie della Societa Astronomica Italiana Supplementi*, 13:142, 2009.

[52] Roberto Scaramella and Sabina Sabatini. Hunting for ghosts: Low surface brightness galaxies from pixels. *Proceedings of the International Astronomical Union*, 3(S244):295–299, 2007.

[53] JM Schwartzenberg, S Phillipps, and QA Parker. A search for low surface brightness galaxies in virgo using tech pan films. *Astronomy and Astrophysics*, 293:332–336, 1995.

[54] Nicu Sebe, Ira Cohen, Ashutosh Garg, and Thomas S Huang. *Machine learning in computer vision*, volume 29. Springer Science & Business Media, 2005.

[55] Lior Shamir. Automatic detection of peculiar galaxies in large datasets of galaxy images. *Journal of Computational Science*, 3(3): 181–189, 2012.

[56] Paramanand Singh and P Ramakrishnan. Powder characterization by particle shape assessment. *KONA Powder and Particle Journal*, 14:16–30, 1996.

[57] R Chris Smith, AR Walker, and C Miller. Decam community pipeline software requirements and technical specifications.

[58] RM Smith, GJ Privett, S Phillipps, and Jonathan Ivor Davies. Automatic detection of low-surface brightness galaxies on digital images. In *Symposium-International Astronomical Union*, volume 161, pages 549–552. Cambridge University Press, 1994.

[59] CD e Irwin MJ y Bothun GD Sprayberry, D e Impey. Galaxias de bajo brillo superficial en el universo local. iii. implicaciones para la funciÃ³ndeluminosidadfieldgalaxy. *The Astrophysical Journal*.

[60] D Sprayberry, CD Impey, and MJ Irwin. Low surface brightness galaxies in the local universe. ii. selection effects and completeness of the automated plate measuring survey. *The Astrophysical Journal*, 463:535, 1996.

[61] James Stradling. Unsupervised machine learning with one-class support vector machines, Oct 2016. URL https://medium.com/@jamesstradling/unsupervised-machine-learning-with-one-class-support-vector-machines-129

[62] Vincenzo Testa, Sabina Sabatini, Roberto Scaramella, Giuseppe Longo, S George Djorgovski, Roy R Gal, Robert J Brunner, and Reinaldo R de Carvalho. Automated search of lsb galaxies in dposs (cronario project): Method and first results from follow-ups. In *Mining the Sky*, pages 557–563. Springer, 2001.

[63] Clemens Trachternach, Dominik J Bomans, Lutz Haberzettl, and R-J Dettmar. An optical search for low surface brightness galaxies in the arecibo hi strip survey. *Astronomy & Astrophysics*, 458 (1):341–348, 2006.

[64] P Väisänen and EV Tollestrup. Detecting the low surface brightness universe: the extragalactic background light and lsb galaxies. In *International Astronomical Union Colloquium*, volume 171, pages 365–372. Cambridge University Press, 1999.

[65] Sidney Van den Bergh. A catalogue of dwarf galaxies. *Publications of the David Dunlap Observatory*, 2:147–150, 1959.

[66] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014. ISSN 2167-8359. doi: 10.7717/peerj.453. URL https://doi.org/10.7717/peerj.453.

[67] Aku Venhola, Reynier Peletier, Eija Laurikainen, Heikki Salo, Thorsten Lisker, Enrichetta Iodice, Massimo Capaccioli, G Verdoes Kleijn, Edwin Valentijn, Steffen Mieske, et al. The fornax deep survey (fds) with the vst: Iii. low surface brightness (lsb) dwarfs and ultra diffuse galaxies (udgs) in the center of the fornax cluster. *arXiv preprint arXiv:1710.04616*, 2017.

[68] B Vollmer, B Perret, M Petremand, F Lavigne, Ch Collet, W Van Driel, F Bonnarel, M Louys, S Sabatini, and LA MacArthur. Simultaneous multi-band detection of low surface brightness galaxies with markovian modeling. *The Astronomical Journal*, 145(2):36, 2013.

[69] BA Vorontsov-Vel'Yaminov and RI Noskova. Galaxies of low surface brightness. *Soviet Astronomy*, 15:402, 1971.

[70] Mike Walmsley, Annette MN Ferguson, Robert G Mann, and Chris J Lintott. Identification of low surface brightness tidal features in galaxies using convolutional neural networks. *Monthly Notices of the Royal Astronomical Society*, 483(3):2968–2982, 2019.

[71] Michael A Wirth. Shape analysis and measurement. *University of Guelph. CIS*, 6320, 2001.

[72] GH Zhong, YC Liang, FS Liu, F Hammer, JY Hu, XY Chen, LC Deng, and B Zhang. A large sample of low surface brightness disc galaxies from the sdss–i. the sample and the stellar populations. *Monthly Notices of the Royal Astronomical Society*, 391 (2):986–999, 2008.