University of Geneva

Department of Computer Sciences

Astronomical radio source detectors

Master Thesis by:

Patrick Arlt

Supervisors:

Prof. Svyatoslav Voloshynovskyy

Dr. Olga Taran

For the Master Degree of Masters in Computer Sciences

University of Geneva - Science Faculty

Geneva - Switzerland

June, 2023

ABSTRACT

Context. Surveillance of the sky is getting more and more attention specifically with the new technologies and telescopes developed. If astronomers want to continuously survey the night sky and be able to catalog everything that is visible, then they will need new optimized, efficient and intelligent algorithms capable of performing source detection and characterization tasks over petabytes of data. This would also be a great improvement if the method used needed less expertise in the domain.

Aims. In this thesis we explored different methods and approaches to perform the source detection task on radio-astronomical data. We explored a few traditional approaches and some machine-learning-based ones.

Methods. We first explored traditional approaches such as the morphological thresholded blob detection technique proposed by [1, 2] and also the state-of-the-art Python Blob Detection and Source Finder (PyBDSF) method [3, 4, 5]. Then we created a deep convolutional autoencoder, initially inspired by the models proposed by DeepFocus [6, 7], and which has the purpose of performing a pre-processing step on the data to then feed the outcomes to the thresholded blob detection method. This pre-processing step consists in training a CNN capable of predicting the corresponding binary map, i.e. containing only the existing sources without any background noise and artifacts, given some input images. The purpose of this step is to basically increase the signal-to-noise ratio such that the detection process is made simpler. Those investigation has been performed over the dataset generated in [8] which contains two subsets of 9164 sky models that have been simulated using the Common Astronomy Software Applications (CASA) instrument [9, 10]. The first subset is dedicated to noiseless data and the second one, is to noisy ones.

Results. We first investigated the traditional approaches which produce accurate detection over noiseless images but when applied to noisy ones, those tools have much more difficulty performing the source detection. Then, we evaluate machine learning (ML) based approaches, which led to relatively good detection over the noiseless data and shows encouraging outcomes over the noisy data. The ML solution outperforms the traditional

ones on the source detection task over the noisy images even though the detection does not produce perfect results.

Conclusions. Traditional approaches lead to accurate detections over noiseless data. Unfortunately, they have much more difficulty performing this task on noisy ones. This is why, using a pre-processing step designed through deep architectures to increase the signal-to-noise ratio reduces the complexity of the source detection over those data. The solution proposed in this thesis showed interesting and encouraging results for this ML based approach and could be used for further works and discoveries.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to the University of Geneva for providing me with the opportunity to conduct this research and pursue my Computer Sciences Master's degree. I am very grateful to my supervisors, Prof. Svyatoslav Voloshynovskyy and Dr. Olga Taran, for proposing me this master's thesis and also for their guidance and support, and insightful discussions throughout the course of this thesis. Dr. Taran's dedication and expertise have been instrumental in shaping this research work. I am truly grateful to her for the time she took to supervise me, her continuous assistance with technical issues, her meticulous corrections of my work, and her priceless contributions to generating new ideas and exploring new paths of exploration. Her mentorship has been instrumental in my growth as a researcher.

I would also like to express my thanks to the authors of the paper [8]: O. Bait, M. Dessauges-Zavadsky, and D. Schaerer for providing me with the needed radio-astronomical data for this research. Their generous contribution has been essential to the realization of this thesis project and to be able to work with high-quality datasets of radio-astronomical images, which were explicitly designed for this topic of research.

Furthermore, I would like to acknowledge the developers of [1, 2] and Karabo [11, 12] for their assistance in resolving technical issues encountered during the implementation and analysis stages of this research. Their expertise and prompt support were greatly appreciated and contributed to the smooth progress of this work.

I am also grateful to my family, friends, colleagues, and fellow students for their understanding, support, insightful discussions, and constructive feedback. Their contributions have enriched my understanding and enhanced the quality of this research.

This research work would not have been possible without the collective effort and contributions of all those mentioned above. I am grateful for their support and assistance, and I am honored to have had the opportunity to work with such exceptional individuals and institutions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.1 Radio-astronomy telescope arrays

Since the 1960s, astronomers have undertaken the herculean task of surveying the whole night sky, and methodically cataloging everything visible using a wide range of telescopes, tools, and equipment. With time passing by, more and more discoveries and new technologies appear in every field. This is of course also the case for astronomical surveillance systems: technological progress allows us to capture more data but also more details and precise images.

Thus, source finding became one of the complex challenges in radio astronomy. According to Norris et al. [13] before the **Evolutionary Map of the Universe (EMU)**[14, 15] Pilot Survey, 2.5 million radio sources were known. This EMU is a vast project which mainly aims at understanding the creation and formation of stars and galaxies. It also consists in discovering how those evolve with time. The Pilot version was created in preparation for the full EMU to be functional. Today, with the new innovations in terms of new radio telescope development and upgrading of older ones, this number increased up to two orders of magnitude [16], which led to and will continue to lead to new knowledge. One of the new telescopes is the **Australian Square Kilometer Array Pathfinder (ASKAP)**[17, 18, 19, 20] telescope which is expected to capture more than 70 million radio sources. Figure 1.1 shows several examples of antennas designed for this project. With the increase in resolution, sensitivity, and sky coverage of such surveys like EMU, we need to at least improve old source detection algorithms, if not develop new ways to perform this task. That's why the source detection problem is getting a lot of attention lately.

Figure 1.1: Examples of the antennas of the ASKAP project[1][2]

Radio interferometry is an advanced technique that uses multiple tiny antennas to gather spatial data, rather than a few very big ones. It enhances measurements and findings in terms of quality, resolution, and precision while enabling the use of more reasonably priced radio telescopes. For instance, the **Square Kilometer Array (SKA)**[21] project is an intergovernmental international radio telescope project that uses a huge land surface containing a network of many small radio telescopes. It aims at simulating a giant radio telescope with extremely high sensitivity and angular resolution thanks to the interferometry technique called aperture synthesis. This technique uses a collection of telescopes to produce images having an angular resolution equal to the one obtained with an instrument having the size of all telescopes gathered. This also provides multiple field-of-view (FOV), specifically large ones, and when combined with great sensitivity, this would allow the SKA to perform a very extensive survey of the space. Moreover, this task will be executed much faster than if it was done with any other telescope.

The SKA project is currently located in two countries: Australia, which offers low-frequency coverage, and South Africa, which offers mid-frequency coverage. It was initially conceived in 1990 and has since been continuously improved and developed. However, the construction of this project started on December 5th, 2022 in Australia and South Africa, and its first light, or in other words, the first use of the telescope and its first captured images, is expected for 2027 [22]. In the meantime, other countries like Italy, the Nether-

[1] `https://en.wikipedia.org/wiki/Australian_Square_Kilometre_Array_Pathfinder`
[2] CSIRO

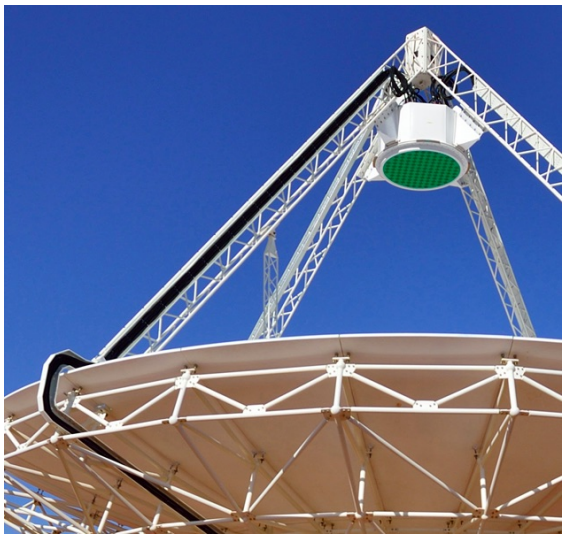

Figure 1.2: Phased-array feed of the ASKAP's antennas[1]

lands, Portugal, and the United Kingdom joined the project. Those six countries founded the **Square Kilometre Array Observatory (SKAO)**[23]. Some other countries like India and Sweden have shown interest in this project and would like to join it as well. Actually, this organization is composed of 8 states: Australia, China, Italy, The Netherlands, Portugal, South Africa, Switzerland, and the United Kingdom.

The ASKAP site is one of the two SKA's central locations (Africa and Australia) and will contain 36 antennas of 12 meters in diameter each and spread across 6 kilometers in Australia's mid-west region. Each of those is provided with a new phased-array feed (PAF) shown in Figure 1.2, which allows it to reach high survey speed and significantly increase the field of view with a collecting area that will cover approximately $4'000$ square meters. Thanks to this design, it is able to perform fast surveys with high sensitivity. This project is operated by the Commonwealth Scientific and Industrial Research Organization (CSIRO)[24, 25] which is the Australian scientific research government agency.

The main use-cases of this work are that it should be at least 50 times more sensitive than any other radio instrument due to its size and a large range of frequencies. Also, if it is built as planned, the monitoring of the sky should become much faster. At first, its observation time will mainly be focusing on the following topics:

- Galaxy formation and gas evolution in the nearby universe

[1] `https://i.stack.imgur.com/Rqq4X.jpg`

- Observe galaxies population: formation, evolution, and population
- Detection and monitoring characterization of the radio transient sky
- Magnetic fields evolution in galaxies over cosmic time

This leads to the creation of ten ASKAP projects:

- High priorities projects:
  - EMU: Evolutionary Map of the Universe
  - WALLABY: Widefield ASKAP L-Band Legacy All-Sky Blind Survey
- Low priorities projects:
  - COAST: Compact Objects with ASKAP: Surveys and Timing
  - CRAFT: The Commensal Real-time ASKAP Fast Transients survey
  - DINGO: Deep Investigations of Neutral Gas Origins
  - FLASH: The First Large Absorption Survey in HI
  - GASKAP: The Galactic ASKAP Spectral Line Survey
  - POSSUM: Polarization Sky Survey of the Universe's Magnetism
  - VAST: An ASKAP Survey for Variables and Slow Transients
  - VLBI: The High Resolution Components of ASKAP: Meeting the Long Baseline Specifications for the SKA

Very Large Array (VLA) telescope [26, 27, 28] is an operating combination of 28 identical radio antennas which work and observe the sky simultaneously to simulate the usage of a large telescope. The particularity of this telescope is that it is mounted on rails and can move following a huge "Y" shape as shown in Figure 1.3: both branches are 21 kilometers long and the foot is 19 kilometers long.

Figure 1.3: Antennas of the VLA telescope[1][2]

We can also talk about the **Atacama Large Millimeter/submillimeter Array (ALMA)**[29, 30] telescope, which is composed of 66 high-precision dish antennas over a maximum of 6'600 square meters:

- 54 of them are 12 meters in diameter. They can be moved as desired to obtain custom disposition from very closely displayed over 160 meters to widely dispersed over 16 kilometers.
- 12 of them are fixed and of size, 7 meters in diameter.

Custom configurations allow to have some kind of a "zoom": a wider display gives more details and closer ones gives better sensitivity to observe objects with smaller luminosity. Also, note that this large installation is located in the Atacama desert of Chile at 5'000 meters elevation. This is a huge benefit as there is low humidity there and combined with this elevation and Earth's atmosphere, noise, and signal attenuation decrease a lot.

1.2 Radio-astronomy data simulators

The primary data processing tool used for radio astronomy is called **Common Astronomy Software Applications package (CASA)**[9, 10]. It is an open-source software used to process radio interferometric data. It is available through a Python interface/library but is mostly implemented in C++ for fast computational purposes. This tool can,

[1] https://fr.wikipedia.org/wiki/Karl_G._Jansky_Very_Large_Array

[2] https://public.nrao.edu/telescopes/vla/technology/

for instance, be used to simulate and use data from the VLA and the ALMA telescope. Nevertheless, it is also often used with other radio telescopes. This framework proposes different kinds of functionalities such as: image analysis tools, deconvolution tools, cleaning tools, and simulation tools to simulate observation and data from ALMA and VLA telescopes. This could be useful to perform, try, or train some source detection algorithm on those simulations.

There is also the package called **OSKAR**[31], which allows simulations of astronomical radio interferometers data. It is indeed its primary utility: produce simulated data from telescopes that use aperture arrays, as planned for the SKA project.

1.3 Source detection approaches

The data flood that is anticipated in next-generation observation facilities for radio astronomy requires cutting-edge advancements in data processing, archiving, analysis, and visualization. Thus as said earlier, source detection is nowadays a trending research task: as the quality and complexity of the collected data are getting better and better. Data are getting more and more complex for the source detection and cataloging stage. It now requires new innovative and more advanced algorithms to extract valuable and usable information.

Multiple solutions have been found while others are still in research and development. For instance, there exist traditional approaches to perform source detection but nowadays, deep architectures neural networks are getting a lot of attention to solve this task.

1.3.1 Traditional approaches

One of the most common straightforward hand-crafted techniques is to detect sources through thresholding of peaks intensity and fitting two-dimensional Gaussians.

Nowadays, there already exist several source detection algorithms that allow us to automatically detect sources. For instance, there are Search And Destroy (SAD)[32], SourceExtractor (SExtractor)[33], SOFIA 2[34, 35], CEASAR[36, 37], PROFOUND[38, 39],

AEGEAN[40, 41], PySE[42], Astronomical Point source EXtractor (APEX)[43], blobcat[44], IFCA[45], SOURCE_FIND[46], and **Python Blob Detector and Source Finder (PyBDSF)**[3, 4, 5].

The paper written by Hopkins et al. (2015) [47] describes the challenge of automatically detecting and characterizing sources from radio astronomical data. They used data taken from the ASKAP and EMU projects and used them to compare and evaluate different existing methods. This challenge provides us with a benchmark for source detection algorithms and helps the scientific world to identify areas for improvement. For instance, the authors also conclude that algorithms had some struggles dealing with overlapping or blended sources but also with complex source structures. Another conclusion is that it works very well with bright isolated sources but when applied to fainter sources, the performance starts to decline.

Anyways, according to this same paper [47], tools that worked well include APEX, AEGEAN, IFCA, blobcat, PyBDSF, PySE, and SOURCE_FIND. On the other hand, SExtractor performed poorly compared to other tools.

PyBDSF is a tool designed to decompose radio interferometry images into sources and make available their properties for further use. It is usually used to perform source detection on radio astronomy images.

1.3.2 Machine learning approaches

New advances and discoveries in the machine learning field led to improvements in performance for general neural networks and specifically convolutional ones. Lately, professionals and scientists have been applying those architectures to detect and classify radio galaxy morphologies which led to successful and encouraging results. For instance, Riggi et al. (2023)[48] performed object detection and classification on the ASKAP EMU survey data using Mask R-CNN [49]. Region Based Convolutional Neural Networks (R-CNN) have been proposed by R. Girshick et al (2014)[50], and consist in extracting $2'000$ regions and performing classification on those. Now, mask R-CNN is an extension of the Faster R-CNN algorithm [51]. In addition to object detection and classification, Mask

R-CNN can also generate a pixel-level mask for each detected object. This means that it can segment the objects in an image, which is useful for a wide range of computer vision tasks, including image segmentation, instance segmentation, and object tracking.

The ConvoSource proposed by Lukic et al.(2019[52]) and DeepSource developed by Vafaei Sadr et al.(2019)[1, 2] have demonstrated the effectiveness of Convolutional Neural Networks in point sources detection. DeepSource shows excellent results and it surpasses PyBDSF which is the currently leading source detection algorithm. Another deep learning strategy that uses encoder-decoder neural networks is the DECORAS proposed by Rezaei et al.(2022)[53]. It can perform source detection on dirty and low-quality images but also accurately retrieve important characteristics of the detected sources, such as their sizes and levels of brightness, i.e. fluxes.

DeepFocus is a deep-learning tool used to perform deconvolution, source detection, and source characterization on simulated ALMA data. As explained in [7], this tool is composed of 6 deep-learning models:

- one convolutional neural network used for the source detection task within the spatial domain,
- one recurrent neural network used to perform peak detection and denoising within the frequency domain,
- four residual neural networks used to apply source characterization.

1.4 Thesis objectives

As a significant and ongoing topic of scientific research, source detection in astronomical data plays a critical role in understanding the properties and behavior of celestial objects. The primary goal of this thesis is to explore state-of-the-art approaches for astronomical source detection with a particular focus on established methods such as PyBDSF.

In addition to examining these conventional techniques, another objective is to investigate a baseline source detector that uses the morphological apsects of typical radio data to perform the detection.

Other actual key tools used to solve the source detection problem are the ones using deep learning based algorithms. These models have become increasingly popular in recent years due to their ability to learn features directly from the data and perform highly accurate source detection. By studying these techniques, the thesis aimed to evaluate the potential of deep learning for astronomical source detection tasks and compare its performance against traditional approaches.

Finally, the research involves conducting a comparison of all these methods using a CASA-simulated dataset. By examining the results of this comparison, the thesis aims to provide insights into the optimal source detection strategies for different types of astronomical data.

Overall, the goal of this thesis is to provide a comprehensive analysis and overview of the strengths and limitations of the state-of-the-art in astronomical source detection and lay the groundwork for future research in this important area of study.

1.5 Thesis structure

This thesis is split into seven parts. Chapter 1 summarizes all the knowledge needed to start this thesis, outlines an overview of some different state-of-the-art methods, from the traditional approaches to machine learning ones, and finally, describes the aim and objectives of this thesis.

Chapter 2 presents the datasets and the metrics we used to evaluate all the solutions proposed.

Chapter 3 is dedicated to the exploration of a baseline approach, which uses a morphological source detector. Chapter 3 explains first the thresholded blob detection technique we use. From there, a series of experiments are described in detail, highlighting the specific methods, tools, and parameters used to test the effectiveness of this method. This chapter is completed with a conclusion about the effectiveness of morphological source detectors as a tool for astronomical source detection.

Chapter 4 analyzes a famous traditional state-of-the-art approach named PyBDSF and explores the tool called Karabo, which is built on top of PyBDSF but allows the user

to work with a much easier, simpler, and more user-friendly version of PyBDSF. Finally, this chapter is concluded with an analysis and comparison of both tools.

Chapter 5 explores the use of convolutional deep neural network models inspired by the architecture proposed by the tool called DeepFocus [6, 7], to perform a pre-processing denoising step on the data. Then, the obtained output is given as input to the thresholded blob detection technique. This Chapter provides a detailed description of the architectures and different models investigated followed closely by the obtained accuracy on the source detection task using this pre-processing step.

Chapter 6 contains a summary of all the results obtained within this thesis and is split into 4 sub-part: 3 of them will provide a short summary of respectively the three approaches explored in this thesis but also provide the final results obtained from each of them. The final part will contain a comparison and analysis of those approaches.

Chapter 7 contains a conclusion about all the tools and approaches explored in this thesis and also provides an analysis of the advantages and drawbacks of each solution.

Chapter 2

Dataset and metrics

2.1 ALMA-based dataset

Identifying and analyzing of astronomical sources is an important and actual challenge in astrophysics. In order to solve the source detection problem, it is crucial to have access to high-quality datasets suitable for this specific subject of research.

In the sight of this thesis, we are using the dataset proposed in [8]. The dataset was generated using the tool CASA. The data simulating parameters are given in Table 1 in [8]. The datasets consist of two subsets: one contains noisy radio images and the other contains noiseless radio images. The corresponding images in each subset contain the same sources at the same position, just as it is shown in Figure 2.1 Moreover, both datasets will be split into 3 subsets:

- the training set: composed of 7331 radio-astronomy images
- the validation set: composed of 917 images
- the testing set: composed of 916 images.

Those radio images use the **Flexible Image Transport System (fits)** format which is basically the most commonly used digital file format in the astronomy domain. It allows regrouping in the same file, both, the image metadata and the provisions such as describing photometric and spatial calibration information.

The ground truth sources coordinates are given in **right ascension (ra)** and **declination (dec)** coordinates.

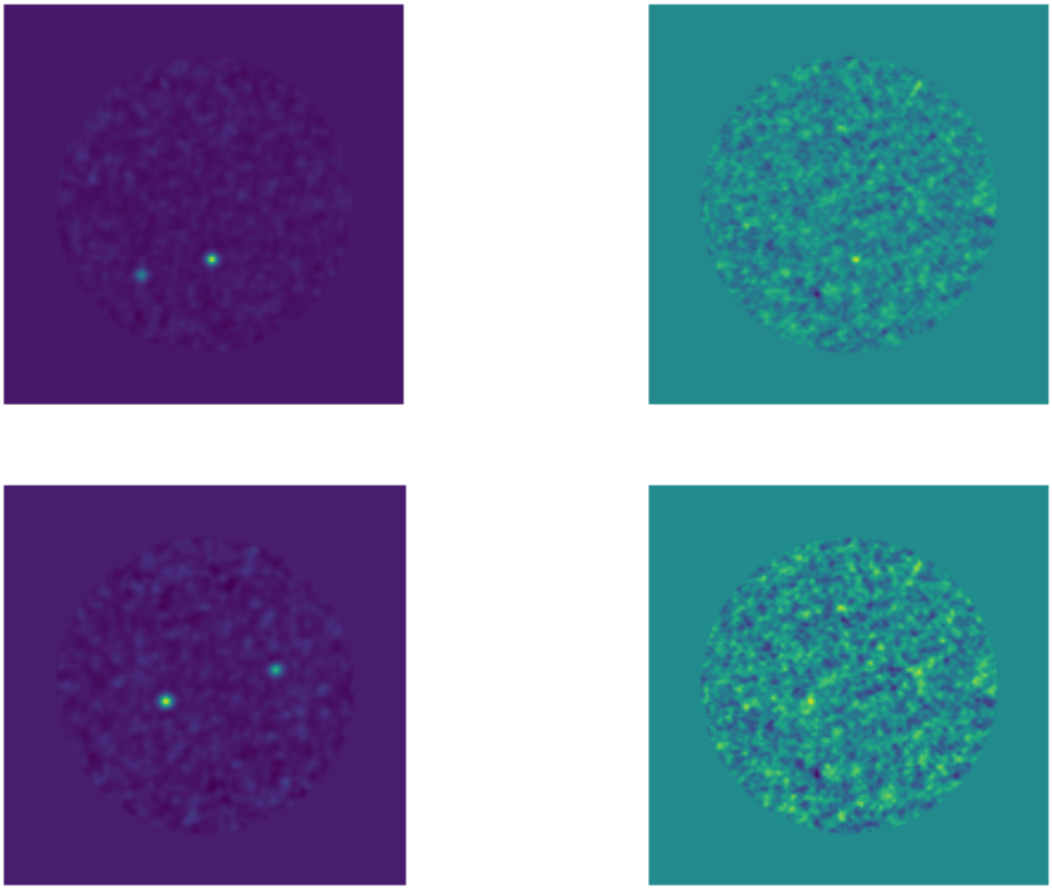

Figure 2.1: Example of clean noiseless images (left column) and the corresponding noisy images (right column)

The machine learning-based approaches work quite often in the image domain with the information given in pixels. The equations 2.1 and 2.2 give the relation between the radio-astronomical *ra* and *dec* and pixels coordinates.

$$\begin{aligned} ra &= CRVAL1 - (CRPIX1 - x - 1) \times CDELT1 \\ dec &= CRVAL2 - (CRPIX2 - y - 1) \times CDELT2 \end{aligned} \tag{2.1}$$

$$\begin{aligned} x &= CRPIX1 - \frac{(CRVAL1 - ra)}{CDELT1} - 1 \\ y &= CRPIX2 - \frac{(CRVAL2 - ra)}{CDELT2} - 1 \end{aligned} \tag{2.2}$$

with:

- x and y: both coordinates expressed in pixels
- ra and dec: both coordinates expressed in right ascension and declination
- $CRVAL1$ and $CRVAL2$ denote true coordinates of the reference location, i.e. central one, in the sky model
- $CRPIX1$ and $CRPIX2$ denote the coordinates of the reference location, i.e. central one, in the image domain
- $CDELT1$ and $CDELT2$ are the pixel sizes

Note that except for the coordinates, those data are all contained in the header of the fits files.

In addition, the dataset contains also the binary maps of each image. It consists of images full of zeros but with ones where there are sources just like it is shown in the left column of Figure 2.2

Once the source detection is done, we will compare the position of the sources that have been detected with the ground truth ones. If they are close enough to each other, for instance, if they are within a given radius of confidence then the detected source is considered as **True positive (TP)**. If a source is detected but does not actually exist, it is considered as **False Positive (FP)**. Finally, if an actual existing source isn't detected, then it is considered as **False Negative (FN)**.

2.2 Metrics

To compare the different approaches, we use the metrics called **purity** and **completeness**. The purity can be interpreted as the ratio of correct detection among all the sources that have been detected. It evaluates the reliability and accuracy of the detection process. The higher the purity is, the lower the number of false positives, and thus the most accurate

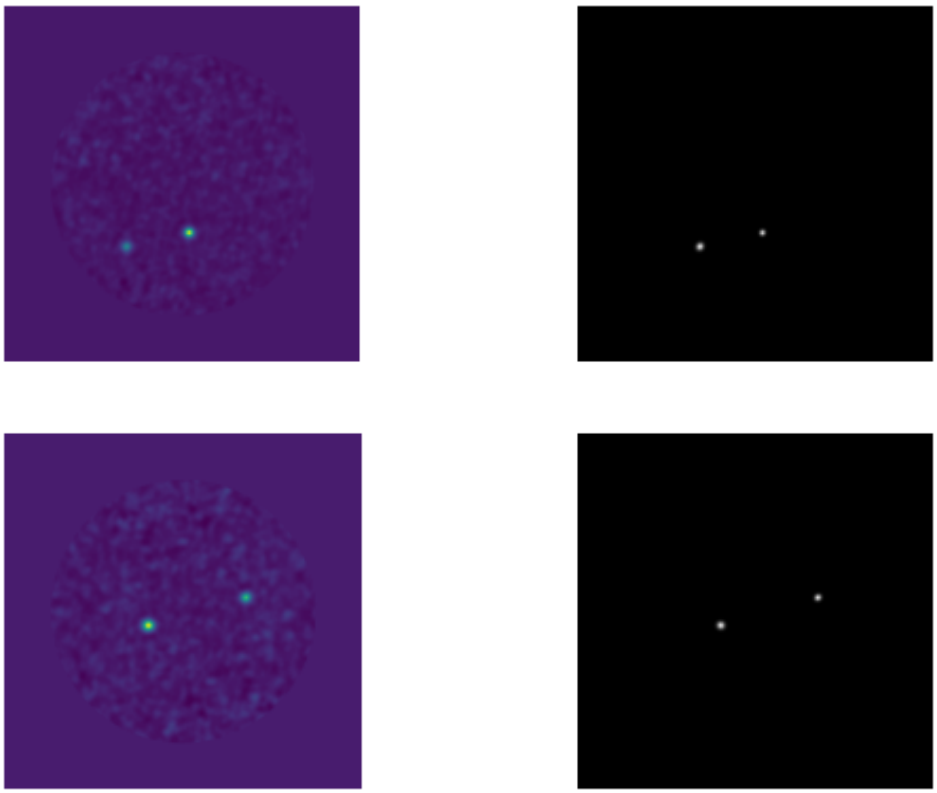

Figure 2.2: Example of noiseless clean images (left column) and the corresponding binary maps (right column)

the detection is.

$$\begin{aligned} Purity &= \frac{TP}{TP + FP} \\ &= \frac{Number\ of\ samples\ correctly\ classified\ as\ a\ certain\ class}{Number\ of\ samples\ classified\ as\ belonging\ to\ this\ class} \end{aligned} \tag{2.3}$$

Completeness is another metric used in source detection to assess the effectiveness of the algorithm. It represents the ratio of correct detection among the existing sources. A high completeness value indicates that the algorithm successfully captures a large percentage of the true sources, while a lower completeness value suggests that some sources may have been missed or undetected. Therefore, completeness is an important measure of the algorithm's ability to identify the entirety of the source population within the data.

$$
\begin{aligned}
Completeness &= \frac{TP}{TP + FN} \\
&= \frac{Number\ of\ samples\ correctly\ classified\ as\ a\ certain\ class}{Number\ of\ real\ data\ belonging\ to\ this\ class}
\end{aligned}
\tag{2.4}
$$

Chapter 3

Morphological detection

3.1 Introduction

The most straightforward approach to radio-source detection is thresholding. The efficiency of such a direct approach is very dependent on the signal-to-noise ratio (SNR) and choice of threshold. SNR is a metric that quantifies the strength of a signal relative to the level of background noise. The greater it is, the stronger the signal is relative to the noise level, resulting in a clearer and more reliable signal. Despite a big dependence on the data, we took this approach into consideration due to its simplicity and we use it to evaluate an upper bound, i.e., the worst-case scenario where the significant expert knowledge or machine learning processing is not used. Taking into account the nature of the used data and its low dynamic range, instead of using simple thresholding with the fixed threshold estimated on the train/validation sub-set, we found it more convening to determine the threshold for each particular image on the fly. In computer vision quite often the Otsu [54] technique is used for the image binarization. However, Otsu does not provide any guarantees on the connection of related regions. Therefore, inspired by the idea of Thresholded Blob Detection (TBD) based on the morphological image processing described in [1, 2], we took it as a non-expert approach for the investigation.

In this section, we are exploring the TBD technique directly on original images without the CNN pre-processing step. Note that we will also use this threshold method on resulting output from deep models in Chapter 5.

3.2 TBD algorithm

Applying a simple threshold with a fixed threshold value on our data could be a baseline approach. Nevertheless, this technique is too simple and will certainly result in poor source detection accuracy. That is why we explore the TBD technique where the optimal threshold value is determined using a set of dynamic steps. This solution has been proposed in [1] and is illustrated in Figure 3.1. The left images show how the thresholding works and how it produces multiple different blobs. Then, the upper right curve shows the number of blobs detected depending on the value of the threshold τ. First by increasing it, the number of detection increases but then after a certain value, it starts decreasing. This is because some source's intensity might be lower than the threshold and thus they are not detected. At some point, the threshold becomes too high to detect any sources. Finally, the lower right curve represents the derivative of the upper curve, which is used to determine the optimal value for the threshold τ. Indeed, this method starts with a more or less high threshold value, that the user can specify, and then decreases it until a stop criterion. This one represents the maximum allowed $\Delta N_{blobs}(\tau)$ as Δ_{max} and can be specified by the user as well. Thus, the algorithm decreases the threshold value until it allows us to have a value for $\Delta N_{blobs}(\tau)$ which is as close as possible to Δ_{max}.

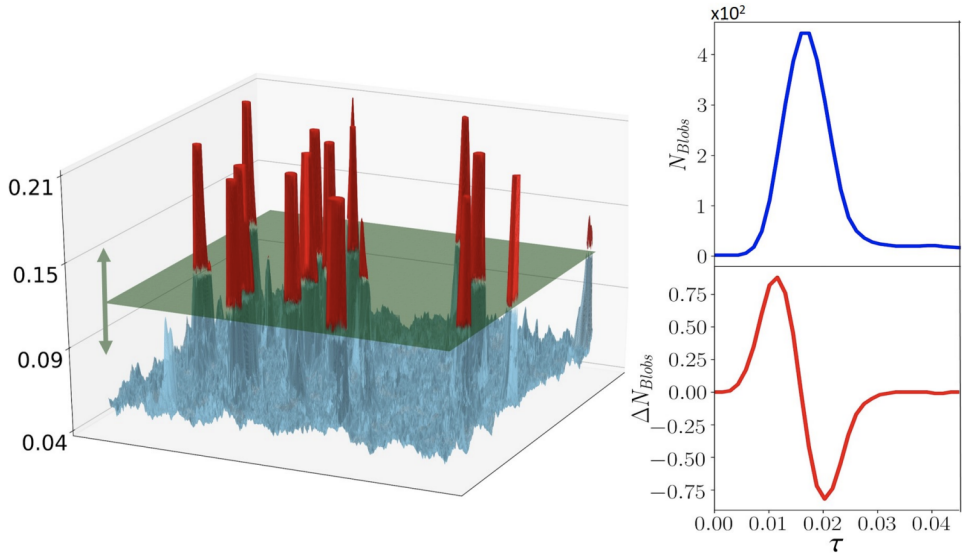

Figure 3.1: Illustration of the TBD method [1]

This technique allows the user to play with some parameters to tune the detection

process. Here is a list of those different parameters:

- *loc_det*: location determination method with three possibilities:
  - *mean*: the coordinates of the source are placed in the middle of the blob using the mean value of the coordinates.
  - *peak*: the coordinate of the pixel which has the highest intensity inside the blob is used as the coordinate of the source
  - *com*: it uses the center of mass as the coordinate of the source

  By default, the *loc_det* parameter is set to *mean*.

- *jump_lim*: this is the parameter that defines the stopping criterion. When the number of objects detected is greater than this value, the algorithm stops and the previous threshold value is used. The default value for the *jump_lim* parameter is set to 50.

- *area_lim*: this is a parameter that allows ignoring blobs that are too small. The default value for the *area_lim* parameter is set to 10.

- *threshold_0*: the starting value for the threshold. The default value is 1

3.3 Experiments using the TBD method

This chapter is dedicated to the evaluation of the performance of the TBD technique from DeepSource's package, without using the neural network pre-processing step. Also, as there are no training stages here, we used only the validation and testing set from the whole dataset.

This algorithm provides five parameters we can play with and to determine, which one has the most impact on the detection's accuracy. We performed a grid search of the best effective parameters on the validation set. Then we fixed the one found to its optimal value and explore the different values possible over the other parameters. We want to find the values that maximize the accuracy of the source detection task. To do so, we performed this step two times:

jump_lim	Purity	Completeness
1	0.454	0.895
2	0.255	0.937
3	0.161	0.936
4	0.111	0.935
5	0.076	0.933
6	0.058	0.932
7	0.044	0.921
10	0.029	0.537

Table 3.1: Impact of *jump_lim* parameter on the accuracy of the detection process over noiseless data under the default values of other parameters

- We applied the grid search on the noiseless dataset and we then tested them on the noisy dataset
- We applied the grid search on the noisy dataset and then tried them on the noiseless dataset

3.3.1 Parameters validation on noiseless data

The impact of the *jump_lim* parameter on the detection accuracy is shown in Table 3.1. Table 3.2 presents the effect of the *ignore_border* parameter and Tables 3.3 and 3.4, respectively depict the influence of the *area_lim* and *thershold* parameters. We highlighted the lines correspondings to the best results obtained and the value of each parameter chosen.

ignore_border	Purity	Completeness
0	0.255	0.937
10	0.255	0.937
50	0.255	0.937
100	0.248	0.935
150	0.215	0.934
200	0.078	0.295

Table 3.2: Impact of *ignore_border* parameter on the accuracy of the detection process over noiseless data under the default values of other parameters except for *jump_lim* fixed to 2

loc_det	Purity	Completeness
mean	0.255	0.937
peak	0.256	0.941
com	0.255	0.938

Table 3.3: Impact of *loc_det* parameter on the accuracy of the detection process over noiseless data under the default values of other parameters except for *jump_lim* fixed to 2 and *ignore_border* set to 10

area_lim	Purity	Completeness
0	0.256	0.941
10	0.343	0.94
20	0.43	0.937
30	0.527	0.934
40	0.6	0.929
50	0.679	0.925
60	0.735	0.918
70	0.793	0.91
80	0.839	0.901
90	0.878	0.891
100	0.933	0.88
105	0.944	0.875
110	0.953	0.87
115	0.961	0.866
120	0.97	0.859

Table 3.4: Impact of *area_lim* parameter on the accuracy of the detection process over noiseless data under the default values of other parameters except for *jump_lim* fixed to 2, *ignore_border* to 10, and *loc_det* to *peak*

threshold	Purity	Completeness
1	0.256	0.941
2	0.267	0.94
3	0.284	0.941
4	0.282	0.943
5	0.291	0.94
6	0.293	0.941
7	0.3	0.94

Table 3.5: Impact of *threshold* parameter on the accuracy of the detection process over noiseless data with *jump_lim* fixed to 2, *ignore_border* to 10, and *loc_det* to *peak*, and *area_lim* to 105

To decide which value to choose, we look at the purity and completeness metrics:

the greater both are, the better. Unfortunately, when one of them increases, the other one decrease. Tables show that the *area_lim* parameter is the one that impacts the most positively the results obtained from the detection. No other parameters allow us to get results as high as this one does. Note that the *ignore_border* and *loc_det* parameters do not seem to have a significant impact on the source-finding process.

The value of *area_lim* that produces a good balance between purity and completeness is *area_lim* = 105. Now by fixing this value, we do the grid search again over the 4 other parameters and try to obtain the best results possible. This second grid search is summarized in Tables 3.6 - 3.9.

jump_lim	Purity	Completeness
1	0.981	0.82
2	0.925	0.875
3	0.825	0.888
4	0.687	0.897

Table 3.6: Impact of *jump_lim* parameter on the accuracy of the detection process over noiseless data while the *area_lim* parameter is fixed to 105 and the default value for other parameters

ignore_border	Purity	Completeness
0	0.925	0.875
10	0.925	0.875
50	0.925	0.875
100	0.898	0.882
150	0.818	0.901
200	0.323	0.273

Table 3.7: Impact of *ignore_border* parameter on the accuracy of the detection process over noiseless data while the *area_lim* parameter is fixed to 105, *jump_lim* to 2, and the default value for other parameters

loc_det	Purity	Completeness
mean	0.925	0.875
peak	0.929	0.879
com	0.926	0.876

Table 3.8: Impact of *loc_det* parameter on the accuracy of the detection process over noiseless data while the *area_lim* parameter is fixed to 105, *jump_lim* to 2, *ignore_border* to 10, and the default value for other parameters

threshold	Purity	Completeness
1	0.929	0.879
2	0.933	0.877
3	0.943	0.877
4	0.945	0.878
5	0.951	0.875
6	0.944	0.875
7	0.953	0.872

Table 3.9: Impact of *threshold* parameter on the accuracy of the detection process over noiseless data while the *area_lim* parameter is fixed to 105, *jump_lim* to 2, *ignore_border* to 10, and *loc_det* to *peak*

As said previously, we are looking for high purity and high completeness. We are thus seeking the values for each parameter that balance best these two metrics. According to Table 3.6, the best value obtained for the *jump_lim* parameter is 2.

Then by fixing this value, we generate the Table 3.7 which explores how the *ignore_border* parameters impact the accuracy of the detection. It does not have a huge influence until this value gets too high and both purity and completeness drop down. The best results seemed to be obtained when this parameter is low. This is why we choose to fix it to 10.

The location determination method does not influence much the result as shown in Table 3.8. The *peak* one seemed to be a little better and that's why we choose this one.

Finally, as with the two previous parameters, the *threshold* one does not bring much improvement to the accuracy. The default value is equal to 1 and we will keep it like this

as this is the one that has the highest completeness.

Then, we use those parameters on both the test noiseless and noisy subsets. The obtained results are reported in Section 3.4.

Regarding this grid search process, we observe that the choice of the *area_lim* parameter value has indeed the most impact on the accuracy. But then, by fixing it and playing with other parameters, we hardly get significantly better results.

In the first stage of this process, when *area_lim* = 105, we have a purity of 0.944 and a completeness of 0.875. At the end of the grid search process, we succeed in obtaining a purity and completeness of 0.929 and 0.879 respectively. The purity metric decreased by 0.015 and completeness increased by 0.004. Those changes are relatively low, compared to what we can obtain by playing with only the *area_lim* parameter.

3.3.2 Parameters validation on noisy data

As in the previous part, we first seek the parameter that has the most impact on the accuracy and then fix it to find the best values for other parameters. From the previous exploration, we expect *area_lim* to have the most impact here as well.

The impact of the *jump_lim* parameter on the detection accuracy is shown in Table 3.10. Table 3.11 presents the impact of the *ignore_border* parameter. The impact of the *area_lim* and *thershold* parameters are reflected in Tables 3.12 and 3.13, respectively. The highlighted lines correspond to the best results obtained and the value of each parameter chosen.

jump_lim	Purity	Completeness
1	0.391	0.35
2	0.248	0.47
3	0.182	0.52
4	0.148	0.544
5	0.121	0.572

Table 3.10: Impact of *jump_lim* parameter on the accuracy of the detection process over noisy data under the default values of other parameters

ignore_border	Purity	Completeness
0	0.248	0.47
10	0.248	0.47
50	0.248	0.47
100	0.239	0.494
150	0.247	0.612
200	0.108	0.29

Table 3.11: Impact of *ignore_border* parameter on the accuracy of the detection process over noisy data under the default values of other parameters except for *jump_lim* fixed to 2

loc_det	Purity	Completeness
com	0.247	0.612
peak	0.245	0.609
com	0.247	0.612

Table 3.12: Impact of *loc_det* parameter on the accuracy of the detection process over noisy data under the default values of other parameters except for *jump_lim* fixed to 2, and *ignore_border* to 150

area_lim	Purity	Completeness
0	0.245	0.609
10	0.493	0.527
20	0.625	0.45
30	0.75	0.391
40	0.883	0.344
50	0.965	0.286
60	0.989	0.238
70	0.991	0.2
80	0.993	0.145
90	0.996	0.099
100	0.994	0.063
105	1	0.054

Table 3.13: Impact of *area_lim* parameter on the accuracy of the detection process over noisy data under the default values of other parameters except for *jump_lim* fixed to 2, *ignore_border* to 10, and *loc_det* to *peak*

threshold	Purity	Completeness
1	0.965	0.286
2	0.978	0.286
3	0.977	0.275
4	0.978	0.275

Table 3.14: Impact of *threshold* parameter on the accuracy of the detection process over noisy data, with *jump_lim* fixed to 2, *ignore_border* to 10, and *loc_det* to *peak* and *area_lim* to 50

As expected, the parameter that has the greatest impact on the source detection accuracy is the *area_lim* one. According to Table 3.13, we can't obtain high completeness on noisy data. That's why we choose to have high purity and then try to increase the completeness afterward. Thus we chose to fix the value 50 to this criterion. From previous experiments, we can suppose that we will not be able to increase it significantly.

Anyway, the results obtained by doing the second grid search with *area_lim* = 50 are

described in Tables 3.15 - 3.18.

jump_lim	Purity	Completeness
1	0.99	0.197
2	0.965	0.286
3	0.919	0.345
4	0.862	0.39

Table 3.15: Impact of *jump_lim* parameter on the accuracy of the detection process over noiseless data while the *area_lim* parameter is fixed to 50, and the default value for other parameters

ignore_border	Purity	Completeness
0	0.97	0.212
10	0.97	0.212
50	0.97	0.212
100	0.958	0.232
150	0.919	0.345
200	0.5	0.216

Table 3.16: Impact of *ignore_border* parameter on the accuracy of the detection process over noiseless data while the *area_lim* parameter is fixed to 50, *jump_lim* to 3, and the default value for other parameters

loc_det	Purity	Completeness
mean	0.932	0.35
peak	0.919	0.345
com	0.931	0.35

Table 3.17: Impact of *loc_det* parameter on the accuracy of the detection process over noiseless data while the *area_lim* parameter is fixed to 50, *jump_lim* to 3, *ignore_border* to 150, and the default value for other parameters

threshold	Purity	Completeness
1	0.932	0.35
2	0.95	0.339
3	0.954	0.34
4	0.954	0.334
5	0.96	0.327
6	0.965	0.331
7	0.963	0.329

Table 3.18: Impact of *threshold* parameter on the accuracy of the detection process over noiseless data while the *area_lim* parameter is fixed to 50, *jump_lim* to 3, *ignore_border* to 150, and *loc_det* to *mean*

As in the previous experiment, the *area_lim* parameter has some impact on the results, but playing with those 4 other parameters doesn't seem to improve much the accuracy of the detection process. By choosing *jump_lim* = 3, *ignore_border* = 150, *loc_det* = mean, and *threshold* = 1, the purity decreased by 0.033, i.e., went from 0.965 down to 0.932, and the completeness increased by 0.064, i.e., went from 0.286 to 0.35 compared to the result obtained in the first stage of this grid search.

The final results obtained from applying the chosen parameter's value on both, the noiseless and noisy datasets are detailed in Section 3.4.

3.4 Results

Table 3.19 depicts the source detection results obtained when using the parameters acquired from the grid search applied to noiseless images. On the other hand, Table 3.20 describes the ones obtained when using parameters gathered from the grid search applied to noisy data. Also, it should be pointed out that we perform the cross-test, i.e., we test the optimal parameters for the noiseless data to the noisy data and vice-versa. In this respect, the first row in Tables 3.19 and Table 3.20 shows the scores obtained when applying the optimal parameters, and the second row corresponds to the cross-test.

	Purity	Completeness
Noiseless	0.929	0.879
Noisy	1	0.013

Table 3.19: Result of the morphological detection approach with the grid search done on the noiseless subset: *area_lim*= 105, *jump_lim*= 2, *ignore_border*= 10, *loc_det*= *peak*, and *threshold*= 1

	Purity	Completeness
Noisy	0.31	0.926
Noiseless	0.932	0.35

Table 3.20: Result of the morphological detection approach with the grid search done on the noisy subset: *area_lim*= 50, *jump_lim*= 3, *ignore_border*= 150, *loc_det*= *mean*, and *threshold*= 1

As can be seen from Table 3.19, by applying the TBD on the noiseless images we succeed in obtaining a high purity and high completeness to the noiseless data. Nevertheless, as expected, by using the same parameters on the noisy images, the obtained results are really poor. The found parameters are well suited for the noiseless ones but our morphological detector does not detect a lot of sources on the noisy ones. Indeed, as the number of true positives is really low and the number of false positives is zero, this means that the detection process has a lot of struggles to detect sources.

On the other hand, when applying the TBD to noisy data, we achieve better results on noisy ones compared to the previous experiment. The number of false positives is high, which means that the detector is too sensitive and that it detects too many sources. When applied to noiseless data, the completeness falls down a lot compared to the previous table, as the number of false negatives is high. This means that the detector has some difficulties detecting existing sources.

3.5 Conclusion

In this section, we explored a morphological detector to perform source detection on radio astronomical data. We used the TBD method [1] to perform the source detection

task. Note that we used this technique on original images without any pre-processing proposed in [1]. We explored the different parameters we could play with and performed a grid search over both noiseless and noisy data to find the ones that suit the best both subsets. This research was performed on the validation subset and then we test the found parameters on noisy and noiseless test subsets.

Based on those experiments, we can conclude that when looking for the best parameters that suit the noiseless dataset, we succeed in obtaining relatively good accuracy. Unfortunately, as we could expect when looking at the difference between noisy and noiseless data when we apply the same parameter to the noisy images, the detection performs really poorly. Indeed, it detects only a few sources and struggles to detect others. Then, by looking for the parameters that suit the best noisy data, we achieve a better detection process on noisy images.

To conclude, this morphological approach using the TBD method allows us to obtain satisfying results and accuracy if we are using only noiseless datasets. Unfortunately, if we are also using noisy images, we will not be able to find a good choice of parameters such that both, noisy and noiseless detection are sufficiently accurate. From this, it becomes clear that it requires more expert knowledge and parameter tuning for each particular data case.

Chapter 4

PyBDSF: traditional approach

4.1 Introduction

PyBDSF (the Python Blob Detector and Source Finder) [5, 3, 4] is a famous state-of-the-art tool that is used for source detection. PyBDSF allows to decompose radio interferometry images into sources and provides comprehensive information about their properties, enabling further utilization and analysis in subsequent studies.

The original PyBDSF tool was developed primarily for use by astrophysicists. It contains a lot of parameters and functions that make it very powerful but yet difficult to use for the users with limited knowledge of radio astronomy. To overcome this drawback and to stimulate collaborative research the PyBDSF source detection functions were implemented in a Karabo[11, 12] software system. Karabo serves as a tool to learn and possibly also benchmark some of the processing steps from a sky simulation to the science products.

4.2 Karabo: source detection process particularities

4.2.1 Parameters

In contrast to the original PyBDSF tool that contains hundreds of different parameters to be tuned, in Karabo the PyBDSF-based source detection is quite limited in terms of parameters that might be tuned. More particularly, it provides only 3 parameters we could play with:

- The *image* on which we want to perform the source detection.
- The *quiet parameter*, which reduces how many prints are displayed.

- The *beam parameter*, which we explored but did not seem to bring any amelioration to the detection when specifying the exact beam of the image.

This is actually, one of the main disadvantages of Karabo: even if this tool is a simpler version of PyBDSF, neither of those parameters allows improvement of the detection process.

We perform the source detection and extract the coordinates of the ones detected. An astonishing point is that the function responsible for this extraction indicates in its name that we can extract pixel coordinates, which should be represented as Integers. Unfortunately, the outputs given are expressed in real numbers. Regarding the position of the source, the ground truth values are given in real units: Right Ascension (Ra) and Declination (Dec). Unfortunately, Karabo does not allow extracting sources directly in real coordinates and this is where the equations 2.1 and 2.2 were useful.

Impact of beam parameters

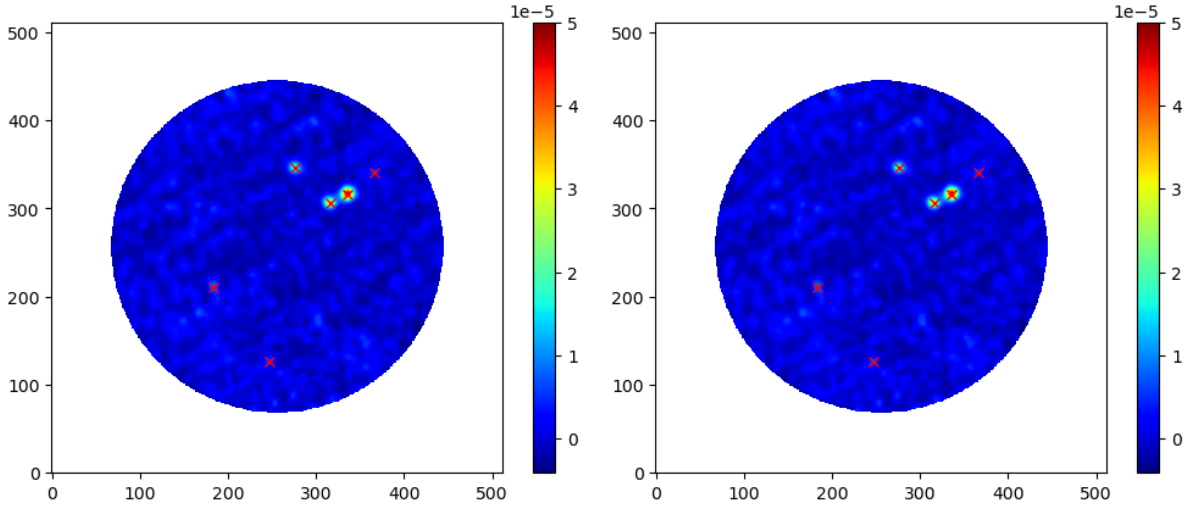

Figure 4.1: Examples of the detected sources without, i.e., the left sub-figure, and with beam parameter, i.e., the right sub-figure.

		Source 1	Source 2	Source 3	Source 4	Source 5	Source 6
Wout beam	X coord	183.93	247.05	276.20	335.91	315.67	366.27
	Y coord	210.52	125.50	345.65	315.98	306.02	340.35
With beam	X coord	183.93	247.05	276.20	335.91	315.67	366.27
	Y coord	210.52	125.50	345.65	315.98	306.02	340.35

Table 4.1: The coordinates detected with and without indicating the beam parameter for the sky model shown in Figure 4.1.

Using the beam parameters does not seem to impact the detection accuracy: we obtain exactly the same number of detected sources with the exact same coordinates. Examples of the sources detected on the noiseless images with and without beam parameters are illustrated in Figure 4.1. The coordinates detected in both cases are given in Table 4.1. It is easy to see that the detected coordinates are identical.

Karabo seems to detect quite well the existing source, which means that we expect the number of true positives to be high and the number of false negatives to be low. We also expect the number of false positive to be high as there is a lot of non-existing sources that are detected by Karabo.

4.2.2 Results

The empirical evaluation of the Karabo-based PyBDSF we performed on the test subset that consists of 916 images from both the noiseless and noisy subsets. Following [8], if the distance between the detected source and the real sources is smaller than a radius of 10 pixels, we consider this as a correct prediction.

The obtained results are summarized in Table 4.2. Also, several examples are visualized in Figure 4.2.

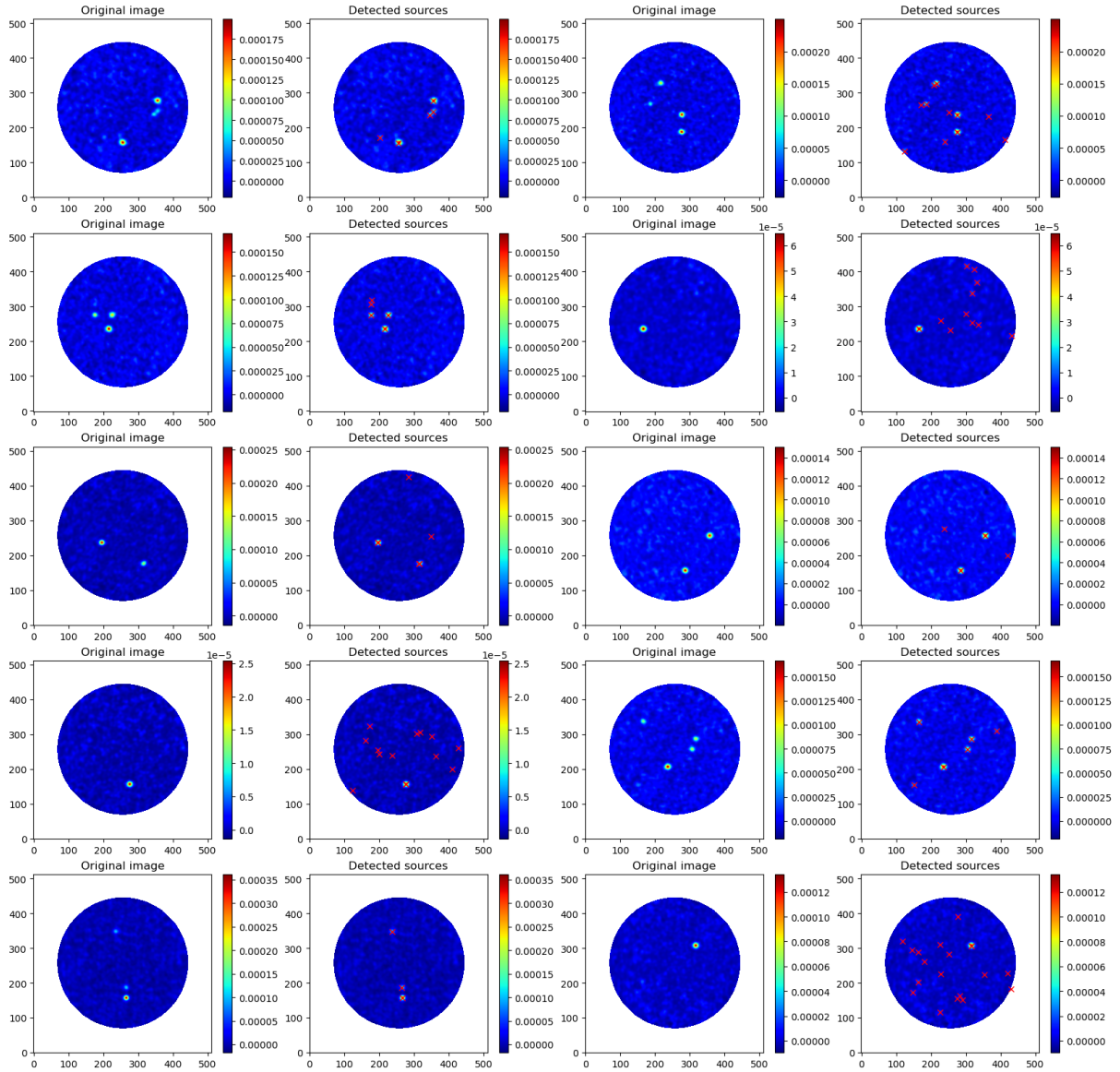

Figure 4.2: Examples of Karabo's source detection.

	Purity	Completeness
Noiseless	0.3433	0.9806
Noisy	0.9772	0.1661

Table 4.2: The purity and completeness obtained with Karabo's source detection on noiseless and noisy test subsets.

As was expected Karabo detects much more sources than there are in the ground truth, i.e., the number of false positives is high. That can be confirmed by the results from Table 4.2: Karabo detects most of the sources, as the number of false negatives is low but has a lot of false positives detection, which leads to low purity and a high

completeness score.

Regarding the noisy dataset, we can't expect the same assumption and results from Karabo's detection. Table 4.2 shows that the number of false negatives is high compared to true positives and false positives, which means that a lot of sources aren't detected and Karabo has a lot of difficulties performing the detection on noisy images with the default parameters of Karabo.

Also, note that we found out that Karabo was not handling correctly the case where no sources were detected.

4.3 PyBDSF

Karabo is built on top of PyBDSF, which might be a complicated tool to use for people that are not in the astronomical domain. Thus, Karabo's main use case is to simplify the usability of PyBDSF by reducing the number of functions usable and also the number of parameters we can play with.

Nevertheless, it remains interesting to explore the performance achievable using the original PyBDSF such that we can compare its original and the simpler versions. To do so we did a few basic experiments using PyBDSF and playing with a few parameters. In the first part of our experiments, we set the *thresh* to *None*, *thresh_pix* to 7, *thresh_isl* to 3, and *rms_map* to *True*. The obtained results are shown in Table 4.3.

	Purity	Completeness
Noiseless	0.921	0.93
Noisy	1	0.023

Table 4.3: Results of PyBDSF with the *thresh* to *None*, *thresh_pix* to 7, *thresh_isl* to 3, and *rms_map* to *True*

With this configuration, we obtain a really good detection accuracy on the noiseless dataset. Thus, if we were only working with those images, this configuration would be really good. Unfortunately, when applying it to noisy ones, it performs really poorly. To improve it, we tried to set the *rms_map* parameter to $False$ and try different values possible or the *rms_value* one. The results obtained are summarized in Table 4.4.

rms_value	Data type	Purity	Completeness
None	noiseless	0.428	0.962
	noisy	0.984	0.178
1	noiseless	0.361	0.963
	noisy	0.984	0.178
2	noiseless	0.361	0.963
	noisy	0.984	0.178
0.5	noiseless	0.361	0.963
	noisy	0.984	0.178
10	noiseless	0.361	0.963
	noisy	0.984	0.178

Table 4.4: Result of PyBDSF with *rms_map* parameter set to *False* and with different values for the *rms_value* parameter

The table shows that by setting the *rms_map* to false, we succeed in achieving a much more accurate detection on the noisy data. By doing this, we went from 64 true positives to 492 and which lead to a decrease in the number of false negatives. Unfortunately, this also leads to new very few false positives detected. Even though the detection performed on noisy images is impacted positively by this change, the one performed on noiseless data is very negatively affected. Indeed, the number of false positives is greatly increased and thus decreases a lot the purity score obtained.

We also observe that modifying the value attributed to the *rms_value* parameter does not seem to have any impact on the detection. By setting it to *None*, which is its default value and which means that its value is calculated inside the program, we obtain a slightly better purity score on the detection performed on noiseless images. Nevertheless, this increase is not really significant and other scores stay the same.

4.4 Conclusion

Karabo, is an easy-to-use package, even though It misses well-written documentation which could be useful for new users. Also, as the example code was outdated, we often had to look at the source code to find which function to use, how to use it, and how to fix the bugs generated by the examples. Karabo could be easily improved by providing a detailed documentation but also by correcting the bug said in Section 4.2, which produces an error when no sources were found by the detection process. Additionally, a significant negative aspect of Karabo is that tuning the source detection parameters is impossible. It simply uses the default parameters to carry out source detection. Still, PyBDSF includes a large number of parameters that we can play with to get better results and tune better the detection process, but unfortunately, Karabo does not use any of these. Also, Karabo provides only the source coordinates in pixels value and not in real values (Ra & Dec units). That is unfortunate, even if the returned values are easy to use.

Anyways, Karabo has some potential but still needs some work and improvement before using it properly. For instance, if your goal is to have good and accurate source detection, we suggest using true original PyBDSF or other solutions such as one using deep machine learning for example. Anyways, Karabo fulfills well the fact that it is easy to learn and understand but also quite easy to use.

Regarding the results obtained on the source detection, Karabo's source finding method detects too many sources on noiseless data, which leads to low purity but high completeness. On the other hand, it detects too few sources when applying it to noisy images. This leads to high purity and low completeness. We would like to obtain high scores for both, purity and completeness metrics, even if they are balancing each other: when one increases, the other one tends to decrease.

Once we tried Karabo, we also wanted to explore the tool on top of which it is built: PyBDSF. To do so, we performed source detection tasks on both noisy and noiseless subsets and tried different configurations by testing diverse values for a few parameters.
We saw in these last experiments that if we are simply working with noiseless data, we could achieve a very good accuracy by setting the *rms_map* parameter to *true*. Unfortu-

nately, when applying this to noisy images, the performances are really poor. This is why we also tried setting it to false and trying different values for the *rms_value* parameter. By doing so we succeed in obtaining better accuracy for noiseless inputs even though it negatively impacts the detection of the noiseless images.

To conclude this chapter, Karabo is a really easy-to-use tool that might be useful for people that are not experts in the radio-astronomical field but still lacks customization, which leads to poor source-finding accuracy. Indeed, if we are using only noiseless inputs, we would prefer using PyBDSF as we can obtain really accurate results. Nevertheless, if we are using both noisy and noiseless images, then, by either using Karabo or either PyBDSF, we succeed in obtaining relatively similar purity and completeness scores, even though PyBDSF allows us to achieve slightly more balanced ones.

Chapter 5

Machine learning approach

5.1 Introduction

Until now, we explored different traditional and hand-crafted approaches. Nevertheless, those may obtain relatively good results under certain conditions and especially on noiseless astronomical data. Unfortunately, the detection accuracy is poor in some cases, especially on noisy images.

In Chapter 3, we made the hypothesis that to improve the performance, we could use a pre-processing step on the images, which would in some sort denoise them. The first idea we thought about was to use the neural network pre-processing step proposed by DeepSource. Unfortunately, their model architecture did not fit our data and led to zero source detected whatever the input was.

Then we learned about a solution called **DeepFocus** [6, 7] which is a tool that can be used for image deconvolution, source detection, and characterization. It proposes 6 different models: one convolutional neural network, one recurrent neural network, and four residual neural networks. In this thesis's scope, we explore different CNN models to perform a denoising step on the data. Then we apply the same TBD method from Chapter 3 to perform source detection.

5.2 Installation and first experiments

DeepFocus uses an autoencoder neural network model which is detailed in Figure 5.1 and which provides an output that has the same shape as our inputs. The aim here is to obtain data with much less noise and aberrations such that the source extraction step done on

those denoised images is performed more easily and especially with better accuracy.

DeepFocus first does a dimensionality reduction step using an encoder detailed in Figure 5.2. This aims at representing astronomical images in a smaller representation, which should allow us to keep the important information from the image and forget about the non-important ones like background noise and artifacts. We are able to choose the size of this slighter representation through the number of hidden channel variables (*n_hidden_channel* in Figure 5.1). Then we give this output to a decoder block detailed in Figure 5.3, which should reconstruct an image of the same size as the original one based on the smaller representation. As the input of the decoder should contain only the important pieces of information from the original images, we should obtain reconstructed denoised images where the irrelevant information has been ignored.

In our setup, the model is trained to output the corresponding binary map.

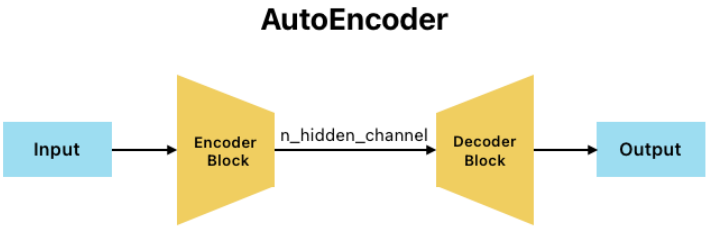

Figure 5.1: DeepFocus autoencoder

At first, we tried using their initial model and applying it to our data. We had to change the code partially such that the models match our data sizes. Indeed, DeepFocus creators used their model on images of size 256×256, but in our case, our data are of shape 512×512.

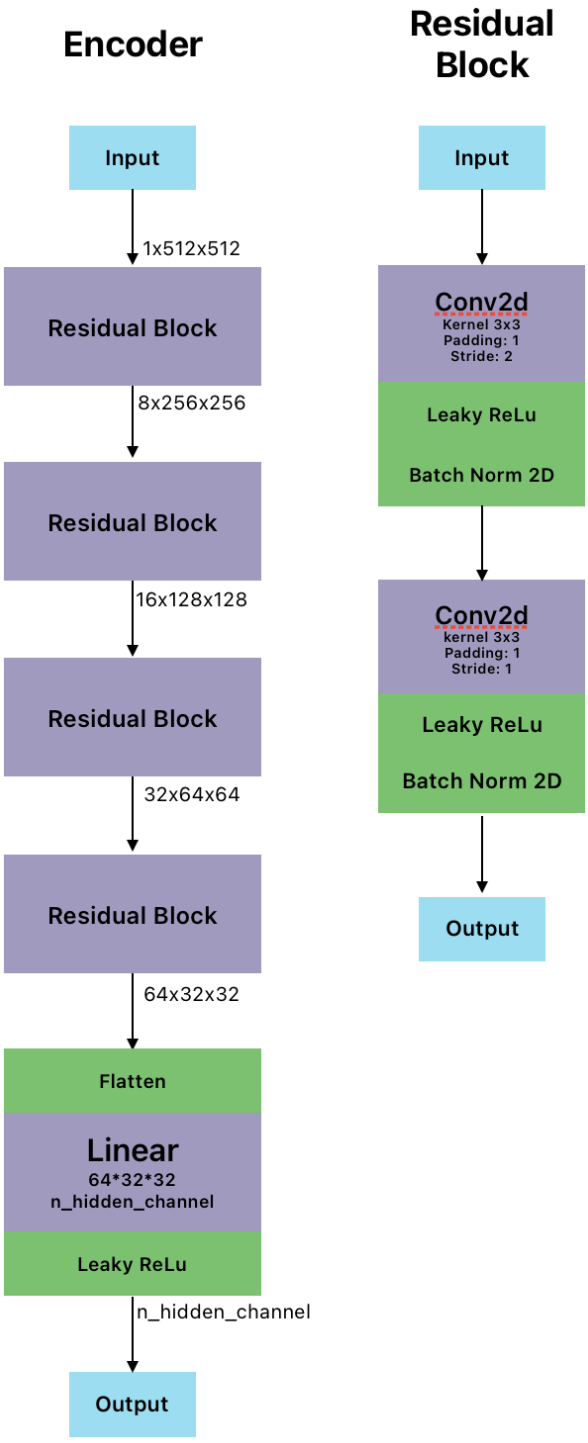

Figure 5.2: DeepFocus initial encoder

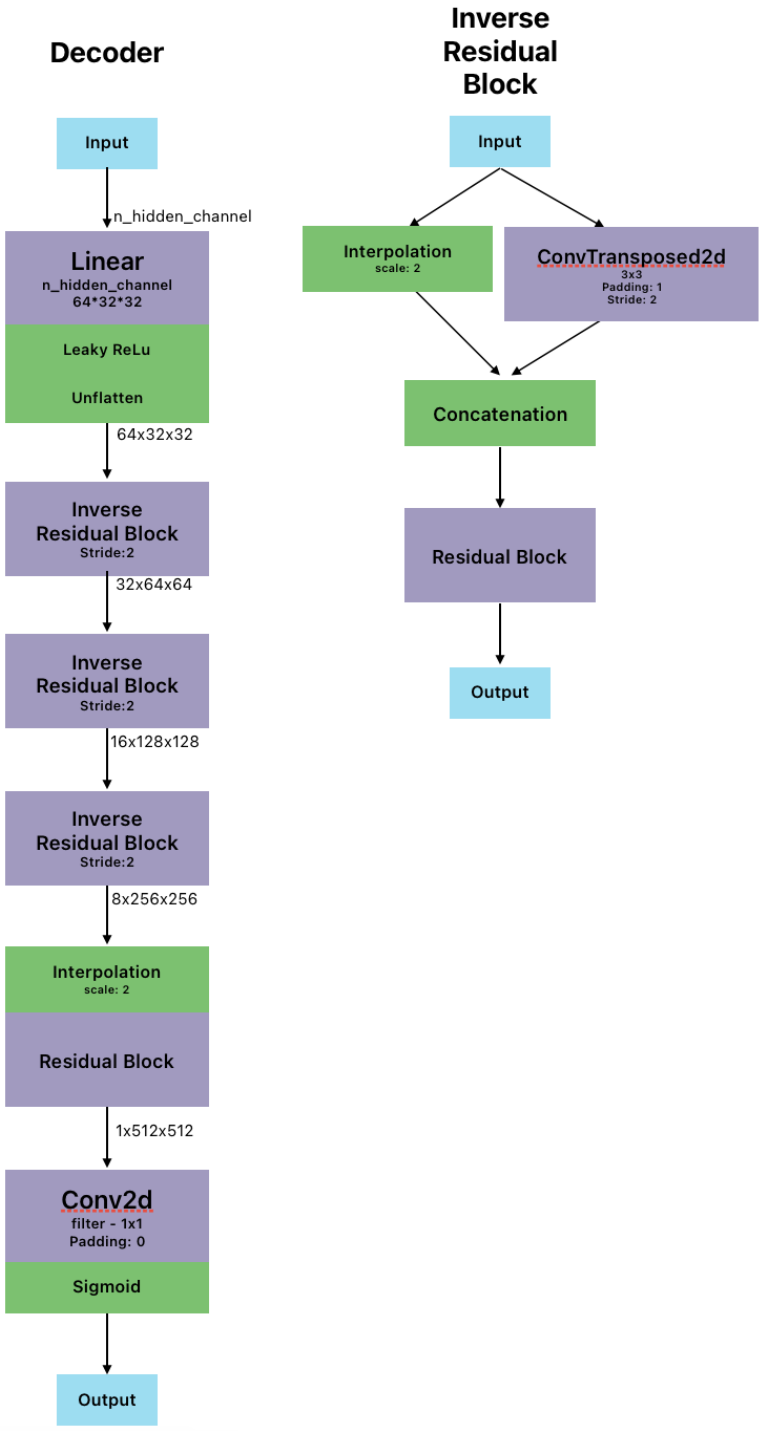

Figure 5.3: DeepFocus initial decoder

Once the model suited our data, we tried training it with our training noiseless dataset. To do so, we had to find the best training parameters we could use. The first one we investigate was the learning rate. We chose to fix the loss parameter to the ℓ_1 and use the *Adam* optimizer. As detailed in Table 5.1, we investigate four different values of learning rate over 25 epochs: 0.1, 0.01, 0.001, and 0.0001. The scores obtained are the ℓ_1 between

the obtained results and the corresponding binary map.

epoch	lr = 0.0001	lr = 0.001	lr =0.01	lr = 0.1
1	67.04	60.12	34.57	2.58
2	65.9	53.14	4.8	0.08
3	64.92	44.82	2.65	0.074
4	63.93	36.16	2.24	0.072
5	62.96	28.39	2.09	0.071
6	61.97	22.15	2.01	0.07
7	60.98	17.22	1.52	0.07
8	59.98	13.15	0.29	0.069
9	58.98	10.7	0.18	0.069
10	57.97	8.59	0.14	0.069
11	56.95	6.95	0.12	0.069
12	55.93	5.69	0.11	0.069
13	54.92	4.72	0.1	0.069
14	53.94	3.94	0.098	0.069
15	52.93	3.35	0.094	0.069
16	51.92	2.9	0.09	0.069
17	50.91	2.54	0.087	0.069
18	49.91	2.25	0.085	0.069
19	48.92	2.03	0.083	0.069
20	47.93	1.84	0.081	0.069
21	46.96	1.69	0.08	0.069
22	45.98	1.56	0.079	0.069
23	45.02	1.45	0.078	0.069
24	44.07	1.35	0.077	0.069
25	43.13	1.27	0.076	0.069
Validation error after 25 epochs	0.27	0.006	0.00067	0.00061

Table 5.1: Model ℓ_1 loss under different learning rate and epoch training

Table 5.1 shows that by using a learning rate too small, it will take too much time for

the model to converge to a local minimum. On the other hand, using a high learning rate will converge quickly, in even less than 10 iterations, but then get stuck in this minimum. Also, if the learning rate is high, it might never reach the exact minimum as it would bounce from one side to the other but never land on the exact minimum. A good way of solving this is to use a high learning rate first and then, after some iterations, decrease it and continue the exploration. This is what we tried and we as is summarized it in Table 5.2. The second (respectively last one) column shows the error obtained on the validation set after training our model over 200 iterations using only a learning rate equal to 0.1 (resp. 0.01). The third one corresponds to training where the learning rate decrease from 0.1 to 0.01 after 50 epochs. Finally, the fourth one depicts the error obtained when we first use a learning rate of 0.1, then decrease it to 0.01 after 50 epochs, and finally decrease it again to 0.001 after the 100th iteration.

	lr = 0.1	lr = 0.1 & 0.01	lr = 0.1 & 0.01 & 0.001	lr = 0.01
Error	0.0025	0.0006	0.0025	0.0006

Table 5.2: Tuning the learning rate by decreasing it during the training over 200 epochs

The best scores are obtained using either a learning rate equal to 0.01 for the whole training, or either using one equal to 0.1 first and then decreasing it to 0.01. For the majority of the following experiments, we chose to fix the learning rate to 0.01 for the full training, even if for some others we tried similar decreasing patterns.

When testing the model on the validation set, the model overfits the dataset and all outputs were empty images. The denoising pre-processing step produces images full of zeros and no sources are kept, which means that the encoding part does not succeed in keeping the important information from the input data. Thus the reconstruction leads to those kinds of outputs.

To solve this issue, we tried changing some training parameters such as the learning rate, number of iterations, number of hidden layers, batch size, weight decay value, the optimizer used, and the loss used. All the configurations tried are described in Table A.1 in the appendix A. Unfortunately, none of those experiments resolved this issue.

Finally, we came up with a new solution: adding some noise to the target binary maps. The problem here is that the binary maps contain a huge majority of zero pixels while the number of pixels equal to ones is really small according to the size of the source in each image. Thus, the model learns a trivial solution, which indeed leads to a local minimum, which is the zero output. To overcome this situation, we add a small noise to the binary maps such that they become slightly different from each other. Moreover, we reset this noise at each training iteration such that the same map become also slightly different from epoch to epoch. This allows the model training to become more efficient and accurate.

Figure 5.4 shows examples of the original binary map and their corresponding noisy version where we added Gaussian noise with a standard deviation (std) of 0.2.

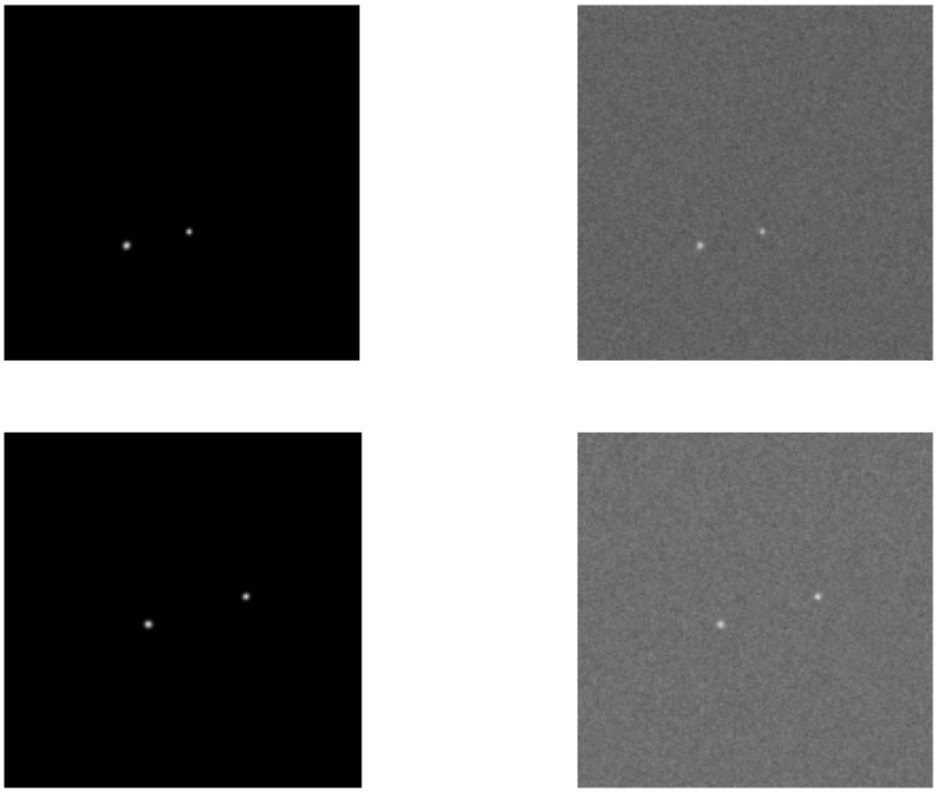

Figure 5.4: Examples of the original binary map (left column) and their corresponding noisy version (right column)

We first tried adding a Gaussian noise of std equal to 0.1 and then also experimented using a std of 0.2, 0.3, and 0.4. The version that led to the best results among those configurations is the one using a Gaussian noise with a std of 0.2. Anyways, the existing

sources are still very visible on the noisy binary maps, even if there is some noise that has been added to them.

Adding this noise helped the training and led to outputs with some appearing aberrations. Still, the initial sources were not conserved and all the outputs looked the same, which means that the model still overfits the data.

We then tried using DeepFocus's residual network, as detailed in Table A.2 in appendix A, but unfortunately, this didn't lead to any improvements.

After investigating more deeply the architecture DeepFocus's CNN uses, we figured out that the latent representation size was really too little to be able to keep the source information. As said above, the initial images used by DeepFocus were 2 times smaller than the ones we use. Moreover, sources present in our data have a quite small area compared to the size of the image. Thus, by reducing our data of shape 512×512 to a representation of size $64 \times 32 \times 32$ (before the linear layer), sources are completely suppressed and do not figure in this smaller representation. Thus, the reconstruction does not succeed in reconstructing them.

To solve this problem, we used the same model but adapted it by reducing the number of layers such that the size of the latent space is bigger and so the chance of keeping the information about the sources is greater. We tried two new architectures:

1. remove two residual blocks in the encoder and two inverse residual blocks in the decoder such that the latent representation shape is $16 \times 128 \times 128$;

2. remove three residual blocks in the encoder and three inverse residual blocks in the decoder such that the latent representation shape is $8 \times 256 \times 256$;

After doing some training and tries, the first option seemed to be the best one and we decided to use it as our model to do our experiments. This architecture is detailed in Figure 5.5 and Figure 5.6.

After all our experiments, we succeed in finding some parameters such that the model succeeds in doing a denoising step on our data during the training.

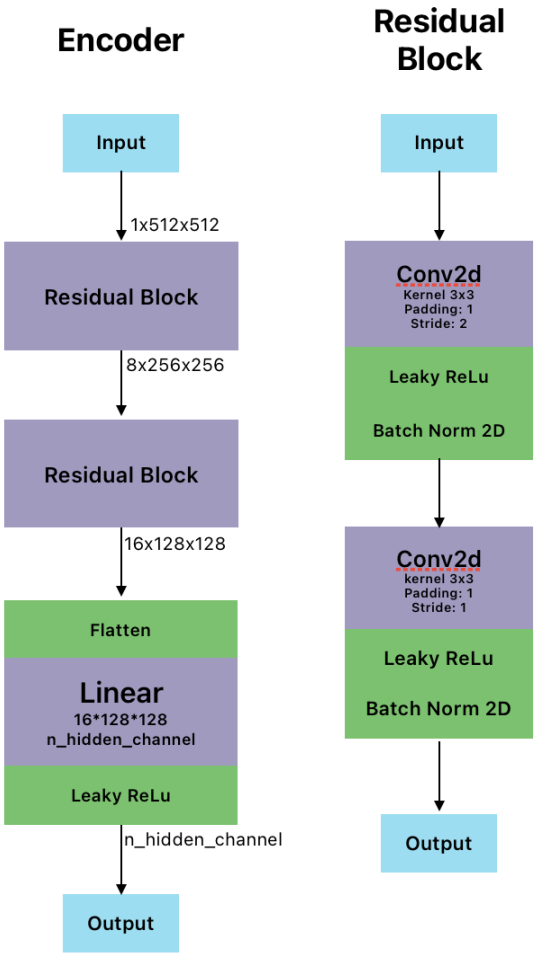

Figure 5.5: Our model's encoder

5.3 Experiments

The final architecture used is detailed in Figures 5.5 and 5.6. For this final experiment, we use the following parameters for this model:

- learning rate = 0.01
- batch size = 64
- number of hidden channels = 256
- Adam optimizer
- ℓ_1

Just like we did the grid search in the previous chapter to find the best parameters using a morphological detector, we decided to explore the performance of two trained

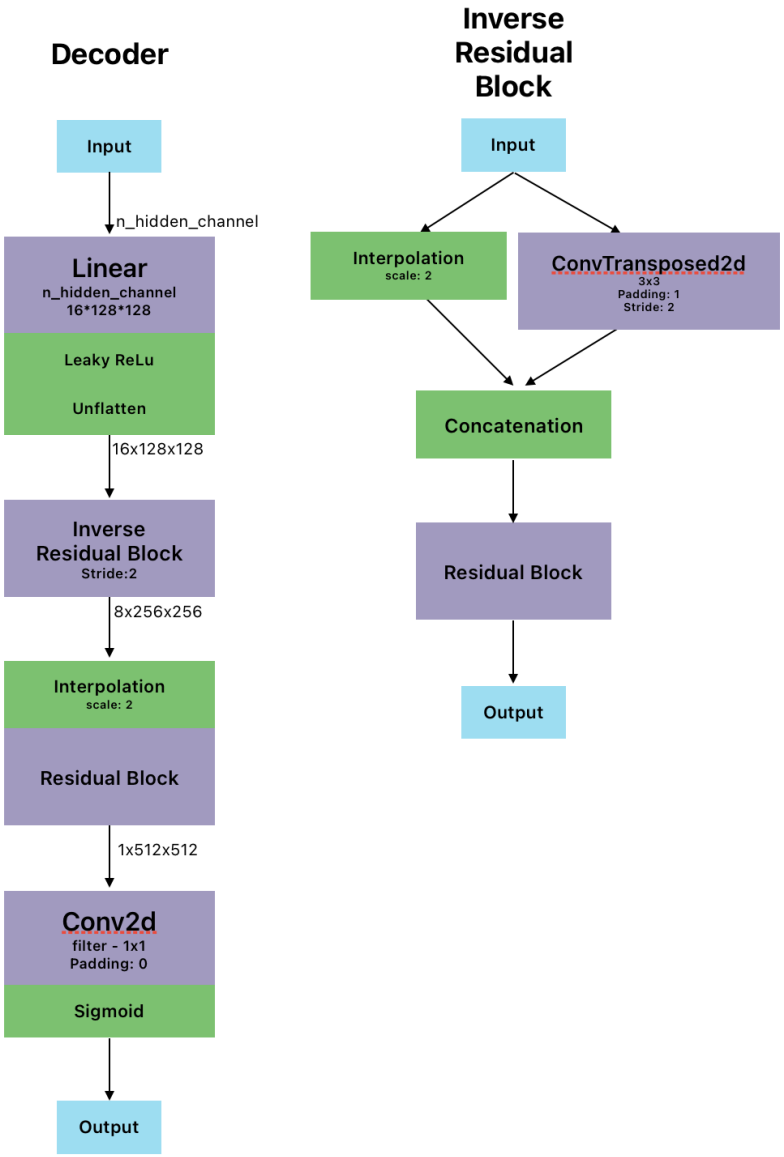

Figure 5.6: Our model's decoder

models: one that has been trained on the noiseless dataset, and one that has been trained on the noisy one. Both of them are evaluated on both: noiseless and noisy data, i.e. cross-test.

5.3.1 Parameters selection for the noiseless data

In this part, we trained our model on the noiseless dataset and used noisy binary maps as our target reconstruction with noise $\mathcal{N}(0, 0.2)$. Table 5.3 shows the ℓ_1 value obtained between the output of the model over the validation set and the corresponding ground truth binary map. The third column "visual" describes very briefly what the output looks

like. Sometimes, we can obtain a low error value but the image is full of zeros, which cannot be used to perform source detection on them. So this column simply says if the outputs are a trivial solution.

The goal here is to evaluate all the intermediate versions of the model and find the one that has the best performance.

Epoch	ℓ_1	Visual
160	0.0011	trivial
180	0.00074	trivial
200	0.00089	trivial
220	0.00095	trivial
240	0.00053	correct
260	0.00043	correct
280	0.0004	correct
300	0.00038	correct

Table 5.3: trained model efficiency

We apply the model trained over 300 epochs to the validation subset of the noisy and noiseless data. Figure 5.7 and Figure 5.8 depicts the obtained results. The first and second column shows noiseless and noisy input while the third one shows the target binary maps. Finally, the last one depicts the produced output of the model.

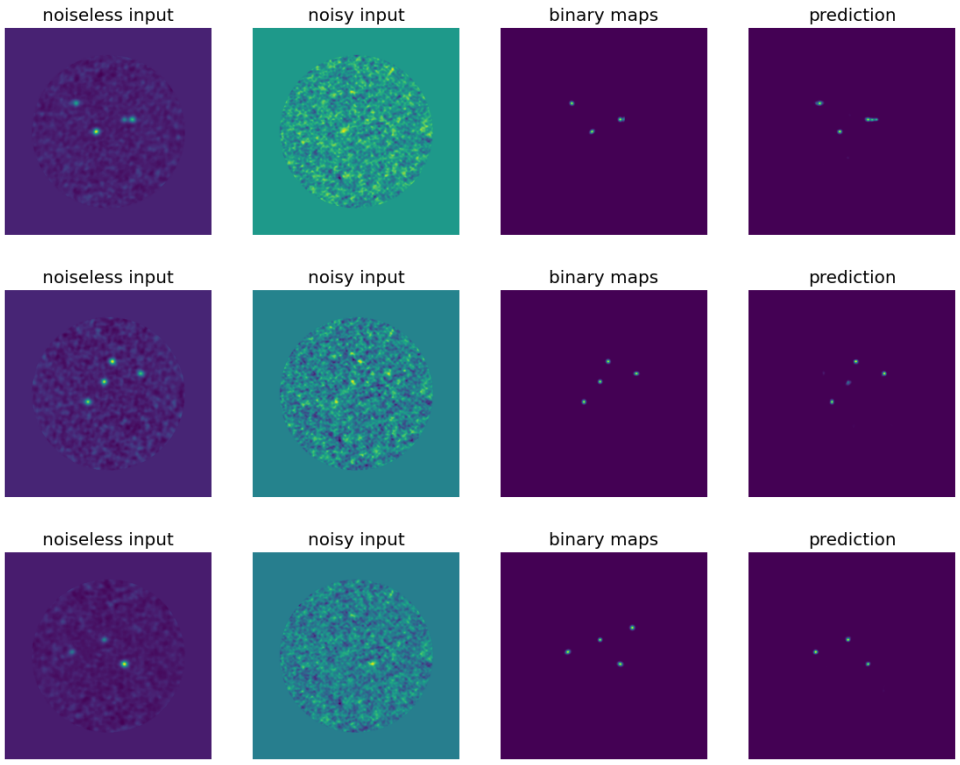

Figure 5.7: Output of the model applied on the noiseless dataset: Model was trained on noiseless data

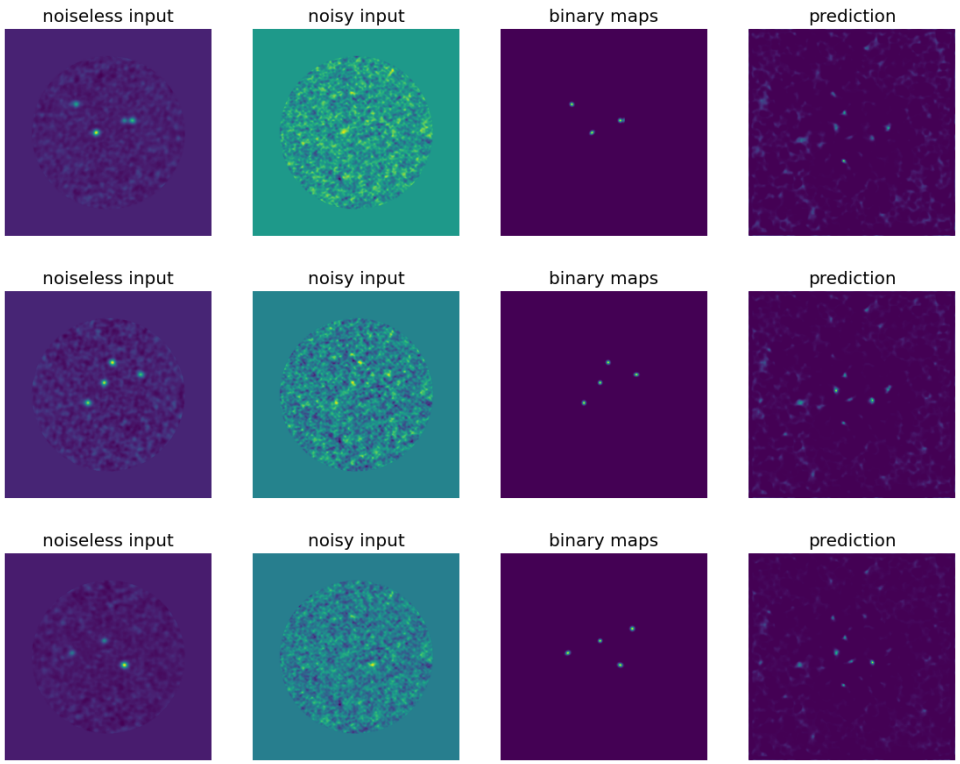

Figure 5.8: Output of the model applied on the noisy dataset: Model was trained on noiseless data

From the obtained results, we can conclude that the models perform well on noiseless data and succeed in reconstructing almost all the sources. Sometimes some are missed and some new appears as can be seen in Figure 5.7. But most of the sources seemed to be retrieved. Also, as this is done with the purpose of denoising the input, we see that the model really succeeds in producing an output that has only the retrieved sources and zeros in the background.

On the other hand, the models perform worse on noisy data: few existing sources are retrieved, and a lot of new ones emerge, but mostly, a lot of aberrations appear on the images. This means that the model does not accomplish to denoise the input and does not succeed in retrieving well the sources.

As the next step to obtain the best possible detection, we apply again a grid search to find the best parameters of the TBD method for our new denoised output. As the model has been trained on the noiseless inputs, we apply this search to the noiseless data, and then we perform a cross-test on the noisy one. Based on the results obtained in the previous chapter, we apply the grid search only on the *area_lim* and *jump_lim* parameters, as those are the ones that have the most impact on the accuracy of the detection. Table 5.4 and Table 5.5 detail the obtained results with different values for both parameters.

area_lim	Purity	Completeness
50	0.689	0.502
100	0.848	0.499
130	0.896	0.492
150	0.932	0.487
170	0.949	0.475
200	0.962	0.457
220	0.977	0.439
240	0.982	0.405
260	0.984	0.362
280	0.982	0.318

Table 5.4: Impact of *area_lim* parameter on the accuracy of the detection process over the output of the model (trained on the noiseless dataset)

jump_lim	Purity	Completeness
1	0.949	0.475
2	0.906	0.84
3	0.846	0.901
4	0.8	0.903

Table 5.5: Impact of *area_lim* parameter on the accuracy of the detection process over the output of the model (trained on the noiseless data)

The best accuracy obtained for the *area_lim* parameter is equal to 170 and when *jump_lim* equals to 2.

5.3.2 Parameters selection for the noisy data

For the noisy data, we train the model over $1'000$ iterations. Model efficiency in terms of ℓ_1 between the predicted output and the target binary map at different epochs is summarized in Table 5.6, where the third column describes briefly how the output looks visually.

Epoch	ℓ_1	Visual
300	0.00066	correct
400	0.00055	correct
500	0.00098	trivial
600	0.00062	correct
700	0.0005	correct
800	0.00049	correct
900	0.000489	correct
1000	0.000483	correct

Table 5.6: Trained model efficiency

Table 5.6 shows that the version of the model that performs the best is the one at iteration $1'000$. Thus, we use the model at $1'000$ epochs for further investigations and to do the denoising over both noisy and noiseless data. Then we use the output of this step as the input of the TBD method. In the previous experiment, the obtained images

from the CNN were well denoised when applying it to noiseless data, but quite poor when using noisy ones. Here, as the model has been trained over the noisy data, it might be able to capture more information and perform better on the noisy data.

Figure 5.9 depicts the prediction of the model when given noiseless inputs. On the other hand, Figure 5.10 shows the produced output of the model when given the noisy images. The first and second columns show the noiseless and noisy inputs while the third one shows the wanted output, which are the binary maps. Then, the fourth one depicts the produced output of the model over the corresponding input.

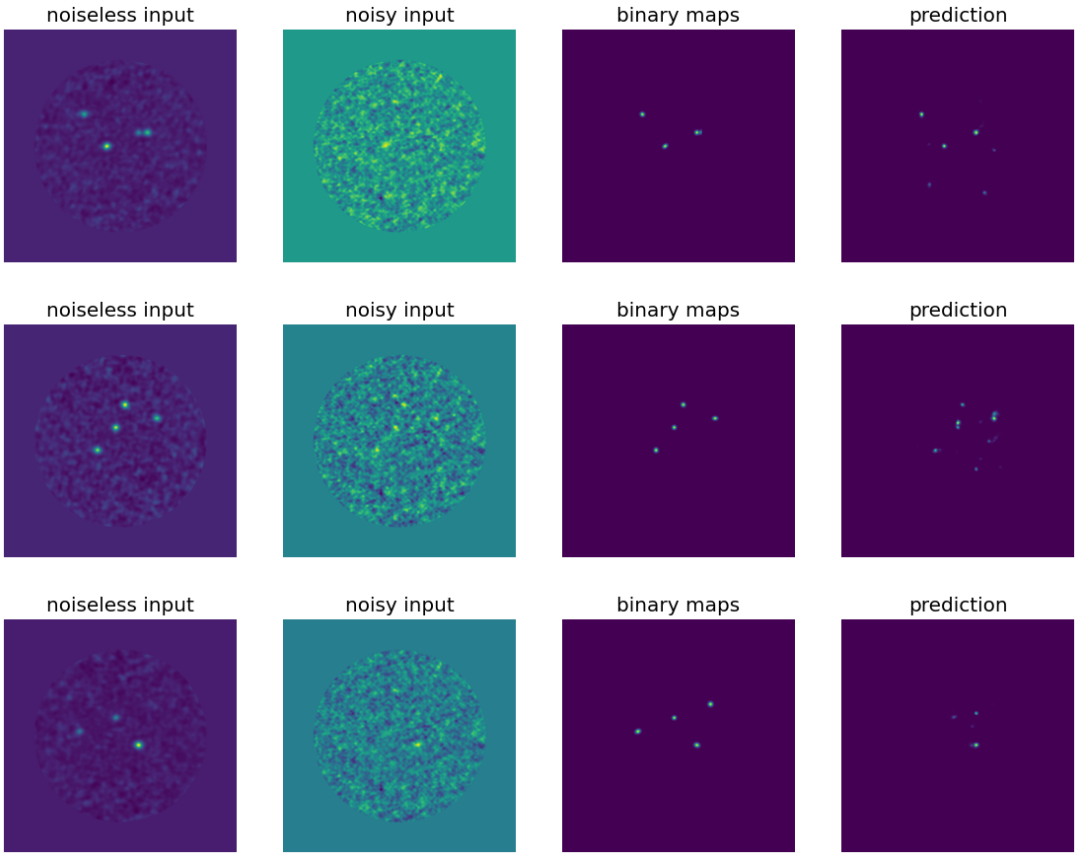

Figure 5.9: Output of the model applied on the noiseless dataset: Model was trained on noisy data

As expected, by training the model on noisy images, we obtain much better outputs from noisy inputs. Compared to the one obtained in Figure 5.8 there are no more aberrations on the output but only source points which is a huge improvement. Nevertheless, there are still some sources that are not being reconstructed, for instance in the third line in Figure 5.10, and some non-existing ones that are being created, for instance, the

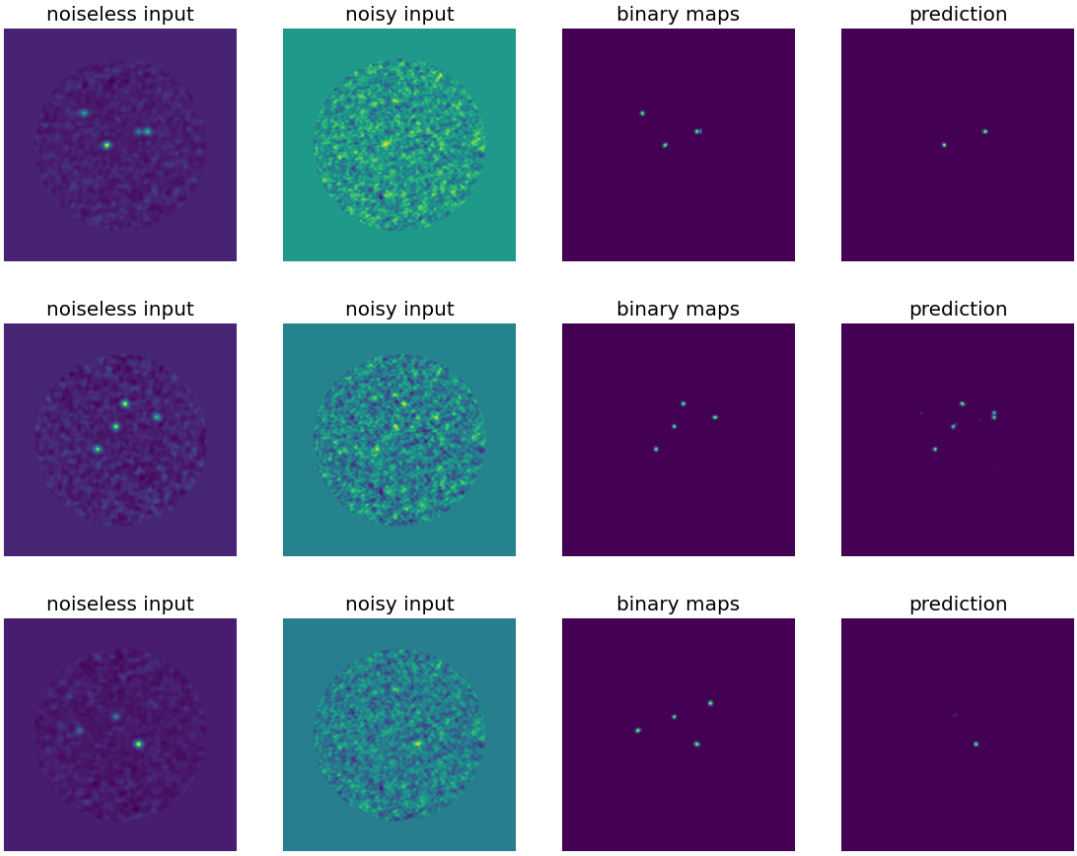

Figure 5.10: Output of the model applied on the noisy dataset: Model was trained on noisy data

second line. The obtained results on noisy images obtained from a model trained on noisy data are much better than the ones obtained with the model trained on noiseless ones.

Regarding the model's predictions produced using noiseless inputs, shown in Figure 5.9, they seem to be slightly less accurate than the one obtained with the model that has been trained on noiseless images. Indeed, compared to Figure 5.7, there are more non-detected sources and more non-existing sources that are being created.

We perform then a grid search to find the optimal parameters to use for the TBD method, which we apply to the model's outputs trained over noisy images. Once we find those values, we fix them and apply them to the outputs generated by the model on noiseless data. Table 5.7 shows the research for the value of the *area_lim* parameter that produces the best accurate detection. On the other hand, Table 5.8 depicts the research done over the *jump_lim* parameter's value.

area_lim	Purity	Completeness
50	0.437	0.52
100	0.607	0.487
130	0.686	0.466
150	0.726	0.448
170	0.77	0.433
200	0.821	0.404
220	0.856	0.376

Table 5.7: Impact of *area_lim* parameter on the accuracy of the detection process over the output of the model trained on the noisy data

jump_lim	Purity	Completeness
1	0.77	0.433
2	0.545	0.66
3	0.515	0.675
4	0.51	0.676

Table 5.8: Impact of *jump_lim* parameter on the accuracy of the detection process over the output of the model trained on the noisy data

The best value for the *area_lim* parameter is 170 according to Table 5.7. This is with this value that we succeed in obtaining a good tradeoff between high purity and high completeness. Regarding the *jump_lim* parameter, we found both values 1 and 2 interesting to explore.

5.4 Results

This section presents the results obtained using a convolutional neural network as a denoising pre-processing step to then give the obtained results to the TBD method, explored in Chapter 3. The aim is to see if there is any improvement in the accuracy of the source detection performed with this TBD technique when the data are going through this CNN step.

In this chapter, we explored two different models: one that has been trained over the noiseless data and then used on both noisy and noiseless data, and one, trained on noisy images and tested on noisy and noiseless data.

5.4.1 Noiseless data results

This section focuses on the results obtained for the model trained on noiseless data.

The used model has been trained over 300 iterations with a learning rate of 0.01, a batch size of 64, and 256 number of hidden channels. Regarding the TBD parameters, we fixed the *area_lim* one to 170 and the *jump_lim* one to 2. By applying both techniques on the noisy and noiseless dataset, we obtain the results detailed in Table 5.9.

	Purity	Completeness
Noiseless	0.894	0.846
Noisy	0	0

Table 5.9: Results of the source detection process with the CNN model trained on noiseless data

As expected in Section 5.3.1, when using the predictions done by our model on the noiseless dataset as our input of the source-detection method, the accuracy is quite good. Both purity and completeness are high which means that the number of missing sources and the number of non-existing sources detected is low. The outputs of the model are quite similar to binary maps and false positives, and false negatives are mainly due to the fact that the model doesn't succeed in reconstructing all existing source and sometimes create new ones.

Regarding the results obtained when applying the CNN and the TBD technique to the noisy dataset, we obtain a really poor accuracy. This is due to the fact that the model learned to reconstruct sources from noiseless images that are easy to understand. Thus the noisy images are much more complicated to handle for such a model.

5.4.2 Noisy data results

We also investigate the results obtained when using a model that has been trained over the noisy dataset instead of the noiseless one. To do so we used the same configuration as the one trained on noiseless images except for the number of iterations which has been changed to 1000. Once this model was trained, we use it on the noisy and noiseless test data and provide the outputs to the TBD method to perform the source detection task. For the detection process, we fixed the *area_lim* parameter to 170 and we tried 2 values for the *jump_lim* one: 1 and 2. Table 5.10 describe the obtained results.

	data type	Purity	Completeness
jump_lim = 1	noiseless	0.562	0.392
	noisy	0.764	0.436
jump_lim = 2	noiseless	0.32	0.64
	noisy	0.537	0.658

Table 5.10: Results of the source detection process with the CNN model trained on noisy data

The obtained results show that the model performs better on the noisy dataset. On the other hand, the scores obtained on the noiseless images, are definitely less good than the ones obtained with the model trained on the noiseless data. An interesting point is that the model performs best on noisy data than on noiseless ones.

When *jump_lim* equals to 1, the purity score is higher than the completeness. Inversely, when it is equal to 2, the completeness is higher than the purity. So depending on which score we would like to maximize, we could either choose one value over the other. Also, when it is equal to 2, our model detects more true positives, which is a good point. Unfortunately, it also leads to more false positives.

Those results might depend a lot on the performance achieved by the denoising part. Thus, if the model performs better, the accuracy of the detection should greatly increase.

5.5 Conclusion

In this chapter, we explored a source detection technique that consists of first passing the data through a pre-processing denoising step, composed of a convolutional autoencoder and then feeding them to the TBD method. This allows us, to first denoise the images such that the detection task is easier to perform. The critical point of this solution is to have a model that simplifies as much as possible to work for the detection algorithm.

The CNN architecture used is an autoencoder, which encodes the input images in a latent space representation and then applies a decoder to reconstruct an image, which has the same size as the original inputs. The advantage of this technique is that with the right architecture and the right configuration, the latent space representation should keep only the important information of the data. Then by reconstructing it, all sources should be reconstructed and present in the output, while the neglectable information should have disappeared. In other words, we would like to obtain a model that is able to transform an input image into a binary map containing only the existing sources.

We tried different model configurations. The first one was inspired by the tool called DeepFocus. Unfortunately, DeepFocus uses different data than the one we are using. Thus, the model wasn't fitted for our data. Then by reducing it and adding some noise to the target binary maps, we succeed in obtaining models that perform a denoising task. We trained two models: one on the noiseless images and one over the noisy ones. Of course, we tested them on both noisy and noiseless data. We also performed a grid search to find the best possible values for the TBD technique parameters. As expected, when testing both models, we saw that they were performing best on the data on which they were trained: the noiseless model was performing better on the noiseless images and the noisy model was performing best on noisy ones. Nevertheless, the model trained over the noiseless data had a lot of struggles with the noisy ones, to the point of having a purity and completeness score of 0. This is normal as the model isn't used to them and mainly because they are much more complicated.
The model trained on the noisy data performed better on the noisy images and succeed in obtaining some results for the noiseless ones.

The accuracy of the detection mainly depends on the ability of the model to denoise well the data but also to keep all the existing sources present. Thus, in the perfect world, we would have a model that is able to transform inputs (noiseless or noisy) into binary maps containing only the existing sources. Then the detection accuracy will enormously increase through very high purity and completeness score. The critical point of this approach is to have a well-trained algorithm and then the detection process should be really easy, to the point where we could apply a threshold to the produced images to detect the sources.

Chapter 6

Results comparison

We explored 3 different types of methods to perform the source detection task: one simple thresholding method, one advanced traditional approach and finally one using deep architectures specifically designed for denoising and subsequent source detection using the first thresholding approach seen. In this section, we present the results and comparative analysis of those three approaches.

6.1 Morphological detection

To establish a baseline for comparison, we initially employed a traditional improved thresholding method for source detection. Indeed, instead of setting a predefined threshold value to distinguish between signal and noise in the background, we used a technique called the TBD, which uses dynamic thresholding to find the optimal value for the threshold for each particular image on the fly. Using this approach, we looked for the parameters that produce the best detection accuracy on our datasets. This research of parameters has been done once over the noiseless and noisy validation subsets. Both sets of parameters have been tested on both noisy and noiseless test subsets. The results obtained on noiseless images are detailed in Table 6.1. On the other hand, Table 6.2 describes the ones acquired using parameters that suit the noisy data. Thus, the first row of those Tables shows the scores obtained when applying the optimal parameters for their corresponding datasets and the second one shows the ones obtained to the cross-test.

	Purity	Completeness
Noiseless	0.929	0.879
Noisy	1	0.013

Table 6.1: Morphological detection approach efficiency tuned on the noiseless data

	Purity	Completeness
Noisy	0.31	0.926
Noiseless	0.932	0.35

Table 6.2: Morphological detection approach efficiency tuned on the noisy data

6.2 PyBDSF

In addition to the thresholding method, we employed PyBDSF a traditional established source detection tool widely used in astronomical research. PyBDSF allows users to decompose radio interferometry images into sources and also allows us to gather information about sources' properties, which we can use in further works.

6.2.1 Karabo

First, we explore the tool called Karabo, which is a package built on top of PyBDSF, and which makes simpler the usage of PyBDSF for non-expert people. After exploring the different parameters we could play with, we performed our experiments on both noisy and noiseless test subsets. Karabo does not provide a lot of parameters and those did not seem to improve the quality of the detection. Table 6.3 regroups the results obtained from those source detection tasks.

	Purity	Completeness
Noiseless	0.3433	0.9806
Noisy	0.9772	0.1661

Table 6.3: Karabo source detection efficiency.

6.2.2 PyBDSF

After trying the easy-to-use version of PyBDSF, it was interesting to explore the initial PyBDSF, especially knowing that Karabo did not allow users to tune a lot of the detection process. Thus we did a few experiments and tried different values of parameters. In the end, the best results we succeed to obtain are summarized in Table 6.4 and Table 6.5. Table 6.4 contains the purity and completeness scores obtained using the *rms_map* parameter set to *True*. The second one describes the results obtained with *rms_map* parameter set to *False* and *rms_value* set to *None*.

Dataset	Purity	Completeness
Noiseless	0.921	0.93
Noisy	1	0.023

Table 6.4: PyBDSF efficiency with *rms_map* parameter set to *True*

dataset	Purity	Completeness
Noiseless	0.428	0.962
Noisy	0.984	0.178

Table 6.5: PyBDSF efficiency with parameters *rms_map* set to *False* and *rms_value* set to *None*

6.3 Machine learning approach

For the last experiments, we wanted to cover machine learning methods. We used a deep-learning model architecture to perform a denoising pre-processing step. The aim of those models was to be able to predict the input's corresponding binary map. This means that given any input image from our datasets (noisy or noiseless), the model should predict images of the same shapes but containing zeros everywhere, except where sources are located.

To perform those experiments we first got inspired by the model proposed by DeepFocus. Then, by observing that this architecture was not suited for our data size, we reduced it and use the architecture detailed in Figures 5.1, 5.5, and 5.6.

To evaluate the performance of such models, we trained two models: one was trained on the noiseless data, and the other on the noisy ones. Of course, both of them were tested on both noisy and noiseless test subsets. Table 6.6 and 6.7 show the results obtained by the model trained on noiseless and noisy images respectively.

	Purity	Completeness
noiseless	0.894	0.846
noisy	0	0

Table 6.6: Efficiency of the source detection process when using the CNN model trained on noiseless data

	dataset	Purity	Completeness
jump_lim = 1	noiseless	0.562	0.392
	noisy	0.764	0.436
jump_lim = 2	noiseless	0.32	0.64
	noisy	0.537	0.658

Table 6.7: Efficiency of the source detection process when using the CNN model trained on noisy data

6.4 Comparison

We split this section into two distinct ones: one where we compare the results obtained on the noiseless test subset and one in which we compare the outcomes acquired on the noisy test subset.

6.4.1 Noiseless results comparision

The first conclusion is that Karabo is the approach that performs the less well on noiseless images among all other solutions. This is why, we won't take it into account in the following explanations.

Based on the results we obtained, the approach that performs the best on those noiseless images is by using PyBDSF. Indeed, at best we obtained a purity score of 0.921 and a completeness of 0.93 which is quite high. This means that we succeed in having a high

number of true positives and only a few false positives and false negatives. Regarding the morphological and deep-learning approaches if we compare the configurations that perform best on noiseless data, both methods result in similar performances. They succeed in achieving purity and a completeness score of approximately 0.9 and 0.85 respectively.

6.4.2 Noisy results comparision

When performing source detection over the noisy data, PyBDSF obtains the worst results among all solutions. This also means that Karabo does not perform well either. In fact, both Karabo and the original version of PyBDSF acquired similar purity and completeness scores on noisy images.

Regarding the morphological detection and the machine learning approach, both perform poorly on noisy inputs when we use the parameter that maximizes the accuracy of the detection on the noiseless data. Nevertheless, the morphological detector still detects a few sources while the model trained over the noiseless data does not succeed in detecting any existing sources (even though there is a few false positives but those are not relevant).

On the other hand, when we are looking for parameters that maximize the detection over noisy images, both approach obtains relatively correct scores. The morphological approach succeeds in obtaining a high completeness score, about 0.93, but a relatively low purity, about 0.35. The machine learning approach, allows us to have more balanced purity and completeness scores. The differences between both metrics scores are smaller: we either have a purity and completeness of 0.764 and 0.436 or either 0.537 and 0.658.

In the end, the morphological approach detects fewer false positives, which means that the sources detected by it have a much greater probability to be real existing ones. Nevertheless, as the completeness score is lower, it implies that this solution detects more false positives, which means that the detection misses a lot of existing sources. Regarding the machine learning solution, the difference between both scores is smaller. This indicates that the detection detects more sources in total, which includes more false positive ones. But inversely, the number of false negatives is lower, and thus the detection misses fewer sources.

In conclusion, the morphological detection approach is more reliable but detects fewer sources. Inversely, the machine learning one is more sensitive and thus, it is less trustworthy. Still, it allows us to detect more existing sources even though more false positives appear.

6.4.3 Final remarks

We compared all the solutions explored in this thesis. The global conclusion we can make is that Karabo does not allow yet to obtain good results compared to other approaches. PyBDSF performs well on noiseless images and outperforms other solutions. Still, it struggles to obtain precise detection over noisy data. The morphological detector obtains relatively good results on noiseless inputs when the parameters are chosen to maximize the score on noiseless data. Otherwise, by using parameters that maximize the detection over noisy images, the scores obtained over noiseless inputs are similar to the ones obtained using Karabo but succeed in obtaining relatively good results on noisy data compared to other approaches. Finally, the machine learning approach obtains accurate detection on noiseless data with a model that has been trained over the same type of input. Conversely, when using a model trained over noisy images, the detection gets more sensitive and leads to more false positives but also allows the detection of more sources having smaller intensity. Nevertheless, this last solution is the one that performs the best on noisy data among all others tried.

An important remark to make is that the machine learning approach depends a lot on the model used. Indeed, this solution is split into two sub-tasks: the CNN denoising step, and the detection process using the morphological approach over the output of the model. The morphological detector should perform really well on binary maps. Thus the key limitation of this approach is to have a well-trained model that succeeds in performing accurate denoising of the input. If the model doesn't succeed in building an accurate representation, either by not reconstructing some existing sources or by creating new ones, then the morphological approach will be fooled. In conclusion, if we want to obtain really accurate detection using this machine learning approach, we should focus on trying

a bunch of different model architecture such that it fits well our data and such that it performs accurately this denoising task. Then, the morphological detection should see its detection task getting considerably easier to perform as the input would be simpler and more accurate.

Chapter 7

Conclusion

7.1 Summary

Source detection is an actual challenge in the radio-astronomy field. This thesis explores the state-of-the-art approaches to astronomical source detection such as PyBDSF, and also studies the morphological detector and finally tries machine learning approaches and evaluates the potential of those solutions for the source detection task. In the end, we had to compare all the solutions found and also discuss their benefits and drawbacks.

The first step of this thesis was to first acquire knowledge about the radio-astronomy domain. For instance, a starting point was to learn about the existing actual project such as the Evolutionary Map of the Universe, but also about the new telescopes such as the Australian Square Kilometer Array Pathfinder or Atacama Large Millimeter/submillimeter Array.

Then we also had to get familiar with the actual proposed solution to solve the source detection problem. For instance, we decided to put a lot of attention to the tools: PyBDSF, the TBD method proposed by DeepSource [1, 2] and finally got inspired by the DeepFocus [7, 6] model's architecture to build our machine learning based solution.

To be able to run experiments and evaluate different approaches and tools, we needed to have access to high-quality datasets appropriate for this specific topic of research. Thanks to the authors of [8], we were able to put our hands on their datasets, which they generated using the CASA tool and the parameters described in Table 1 in [8]. We thus were provided a dataset composed of two subsets: one containing noiseless images and the other containing noisy ones. Both of them were composed of training, validation, and testing subsets. Finally, to be able to evaluate all approaches we have explored in this

thesis, we had to select and define which metrics we would use. We choose to use purity and completeness metrics which are detailed in equations (2.3) and (2.4) respectively.

7.2 Approaches explored

The first approach we explore was the TBD method, which we used as our baseline. This algorithm has been proposed in [1] and uses dynamic thresholding to find the optimal threshold for each image instead of fixing one for the whole dataset. After exploring how we could tune the detection using its parameters, we were able to evaluate this solution. During the exploration of the parameters, we thought about searching for the parameters that maximize the accuracy of the detection for, the noiseless images and the noisy ones. We thus obtained two sets of optimal parameters.

When using the parameters that allow the best detection on the noiseless data, we achieved really good accuracy on those ones. Unfortunately, noisy data is much more complex, and the results produced from this configuration are really poor. On the other hand, when using the parameters that maximize the detection over noisy images, we obtained more balanced results.

We can conclude that this method is quite easy to use, very intuitive, and can produce accurate detection when used on noiseless data. On the other hand, when using it on noisy or on both noisy and noiseless datasets, then the detection is much less accurate.

The second tool we explored was PyBDSF, which is a current famous state-of-the-art solution, and which allows users to decompose radio interferometry data and extract interesting properties from them. We also explored the package Karabo, which is built on top of PyBDSF and provides an easy-to-use version of PyBDSF for people that are maybe non-expert in the radio-astronomical domain. Yet, Karabo is not a solution we can consider if we want to produce accurate detection. It is indeed an easy and intuitive tool to use, but the performances obtained are really poor and result in being the worse approach we explored in this thesis. This is mainly because it does not provide any parameter we can play with to tune the detection process and thus, it impacts the detection negatively. Karabo might become a really interesting tool in terms of ease of use if its creators continue

to improve it.

On the other hand, PyBDSF provides a lot of parameters and allows tuning the detection well. This indeed means that it is a tool that is a bit more complex to handle but when using it correctly, we can achieve really good performance using it. This solution is the most performing one over the noiseless images. Unfortunately, when applied to our noisy images, it performs poorly.

Finally, the third approach we came up with was to use deep neural network models to perform a denoising pre-processing step. From previous experiments, we observed that source-detection algorithms struggle to perform the detection on noisy images. The idea here is to use a machine learning approach to first denoise the input and then feed the image obtained to the morphological technique we saw previously. To do so, we trained convolutional autoencoder networks to produce the corresponding binary maps given the noisy and noiseless inputs. We obtained two trained models: one that has been trained over the noiseless data and one over the noisy ones. When using the model trained on noiseless data, we succeed in obtaining an accurate detection on noiseless images, with high purity and completeness score. Unfortunately, the detection of the noisy input is really bad. This is because the model does not know these data and as they are more complex and the model does not succeed in predicting the corresponding binary maps. On the other hand, when using the model trained over the noisy dataset, predictions over the noisy data are getting much better and this solution becomes the most performance among all other approaches we tried. Unfortunately, the detection of the noiseless images is impacted negatively even though the purity and completeness scores are still relatively correct. The benefit of this method is that with good training, a model should be able to reconstruct exactly the corresponding binary maps, making the detection process much easier and much more accurate. The main drawback here is to indeed succeed in training a model that is able to do that. The idea of using a convolutional neural network is a good choice, but then we have to find the right architecture and right configuration to be able to do this denoising task as accurately as possible. Thus, the precision of the detection of this approach depends only on how well the model performs its prediction

and not on the detection algorithm itself. If the model performs poorly, the detection algorithm will struggle a lot to find sources as the input may have changed for a worse version. On the other hand, if the model succeeds in extracting and reconstructing only the relevant source in the input, then the detection algorithm will easily perform its task.

7.3 Further works

For future work, we aim to explore several extensions. First, with the machine learning approach, we could try using other CNN architecture that may be more suited for our data but maybe also try to use a well-known pre-trained model and fine-tune it over our data like U-Net or DDPM. Another idea would be to train the model over both noiseless and noisy data instead to train it either on noiseless or either noisy only as we did. This would allow the model to have a greater diversity in its training data and might obtain the benefits of both versions. Moreover, this is always interesting to provide a wide variety of data to our models so that it does not overfit our data but instead really learn how to perform correctly the denoising step. We could also try to feed the outcomes of the model to the PyBDSF detection algorithm, instead of the morphological detector we tried.

Finally, the current implementation of the source detection pipeline focused on offline processing. Investigating methods to enable real-time or near real-time detection would be crucial for enabling timely analysis of large-scale radio-astronomical datasets. Exploring parallelization techniques and optimizing the computational efficiency of the algorithms can facilitate the scalability of the detection framework and thus improve the usability for real-time usage of source-detection techniques.

REFERENCES

[1] A. V. Sadr, E. E. Vos, B. A. Bassett, Z. Hosenie, N. Oozeer, and M. Lochner, "Deepsource: Point source detection using deep learning," *MNRAS*, vol. 484, no. 2, pp. 2793–2806, April 2019.

[2] "Deepsource - github," https://github.com/vafaei-ar/deepsource, accessed: 2023-05-26.

[3] "Pybdsf - documentation," https://pybdsf.readthedocs.io/en/latest/, accessed: 2023-05-01.

[4] "Pybdsf - documentation," https://github.com/lofar-astron/PyBDSF, accessed: 2023-05-01.

[5] N. Mohan and D. Rafferty, "Pybdsf: Python blob detection and source finder," *Astrophysics Source Code Library*, pp. ascl–1502, 2015.

[6] "Deepfocus," https://github.com/MicheleDelliVeneri/DeepFocus, accessed: 2023-05-10.

[7] M. Delli Veneri, Tychoniec, F. Guglielmetti, G. Longo, and E. Villard, "3D Detection and Characterisation of ALMA Sources through Deep Learning," *Monthly Notices of the Royal Astronomical Society*, 11 2022, stac3314. [Online]. Available: https://doi.org/10.1093/mnras/stac3314

[8] O. Taran, O. Bait, M. Dessauges-Zavadsky, T. Holotyak, D. Schaerer, and S. Voloshynovskiy, "Challenging interferometric imaging: Machine learning-based source localization from uv-plane observations," *arXiv preprint arXiv:2305.03533*, 2023.

[9] "Common astronomy software application package," https://casa.nrao.edu/, accessed: 2023-03-27.

[10] B. Bean, S. Bhatnagar, S. Castro, J. D. Meyer, B. Emonts, E. Garcia, R. Garwood, K. Golap, J. G. Villalba, P. Harris *et al.*, "Casa, common astronomy software applications for radio astronomy," *Publications of the Astronomical Society of the Pacific*, vol. 134, no. 1041, p. 114501, 2022.

[11] "Karabo - github," https://github.com/i4Ds/Karabo-Pipeline, accessed: 2023-05-05.

[12] "Karabo - documentation," https://i4ds.github.io/Karabo-Pipeline/development.html, accessed: 2023-05-05.

[13] R. P. Norris, J. Marvil, J. D. Collier, A. D. Kapińska, A. N. O'Brien, L. Rudnick, H. Andernach, J. Asorey, M. J. Brown, M. Brüggen *et al.*, "The evolutionary map of the universe pilot survey," *Publications of the Astronomical Society of Australia*, vol. 38, p. e046, 2021.

[14] "Evolutionary map of the universe," http://emu-survey.org/, accessed: 2023-03-27.

[15] "Evolutionary map of the universe," https://www.atnf.csiro.au/projects/askap/index.html, accessed: 2023-03-27.

[16] R. P. Norris, "Extragalactic radio continuum surveys and the transformation of radio astronomy," *Nature Astronomy*, vol. 1, no. 10, pp. 671–678, 2017.

[17] "Australian square kilometre array pathfinder," http://emu-survey.org/, accessed: 2023-03-27.

[18] "Australian square kilometre array pathfinder," https://www.atnf.csiro.au/projects/askap/index.html, accessed: 2023-03-27.

[19] S. Johnston, R. Taylor, M. Bailes, N. Bartel, C. Baugh, M. Bietenholz, C. Blake, R. Braun, J. Brown, S. Chatterjee *et al.*, "Science with askap: The australian square-kilometre-array pathfinder," *Experimental astronomy*, vol. 22, pp. 151–273, 2008.

[20] A. Hotan, J. Bunton, A. Chippendale, M. Whiting, J. Tuthill, V. Moss, D. McConnell, S. Amy, M. Huynh, J. Allison *et al.*, "Australian square kilometre array pathfinder: I. system description," *Publications of the Astronomical Society of Australia*, vol. 38, p. e009, 2021.

[21] "Square kilometre array - wikipedia page," https://en.wikipedia.org/wiki/Square_Kilometre_Array, accessed: 2023-03-27.

[22] "Square kilometre array," https://www.skao.int/en/explore/construction-journey, accessed: 2023-04-30.

[23] "Square kilometre array observatory," https://www.skao.int/en/partners/skao-members, accessed: 2023-05-01.

[24] "Commonwealth scientific and industrial research organisation," https://www.csiro.au/, accessed: 2023-04-29.

[25] "Commonwealth scientific and industrial research organisation - wikipedia page," https://fr.wikipedia.org/wiki/Commonwealth_Scientific_and_Industrial_Research_Organisation, accessed: 2023-04-29.

[26] "Very large array," https://public.nrao.edu/telescopes/vla/#basics, accessed: 2023-04-11.

[27] "Very large array," https://www.space.com/very-large-array.html, accessed: 2023-04-11.

[28] A. R. Thompson, B. Clark, C. Wade, and P. J. Napier, "The very large array," *Astrophysical Journal Supplement Series, vol. 44, Oct. 1980, p. 151-167.*, vol. 44, pp. 151–167, 1980.

[29] "Atacama large millimeter array," https://public.nrao.edu/telescopes/alma/, accessed: 2023-04-13.

[30] "Atacama large millimeter array," https://almaobservatory.org/en/home/, accessed: 2023-04-13.

[31] "Oskar library," https://github.com/OxfordSKA/OSKAR, accessed: 2023-04-30.

[32] "Search and destroy - documentation," http://www.aips.nrao.edu/cgi-bin/ZXHLP2.PL?SAD, accessed: 2023-05-05.

[33] E. Bertin and S. Arnouts, "Sextractor: Software for source extraction," *Astronomy and astrophysics supplement series*, vol. 117, no. 2, pp. 393–404, 1996.

[34] T. Westmeier, S. Kitaeff, D. Pallot, P. Serra, J. Van Der Hulst, R. Jurek, A. Elagali, B. For, D. Kleiner, B. Koribalski *et al.*, "sofia 2–an automated, parallel h i source finding pipeline for the wallaby survey," *Monthly Notices of the Royal Astronomical Society*, vol. 506, no. 3, pp. 3962–3976, 2021.

[35] P. Serra, T. Westmeier, N. Giese, R. Jurek, L. Flöer, A. Popping, B. Winkel, T. van der Hulst, M. Meyer, B. S. Koribalski *et al.*, "Sofia: a flexible source finder for 3d spectral line data," *Monthly Notices of the Royal Astronomical Society*, vol. 448, no. 2, pp. 1922–1929, 2015.

[36] S. Riggi, F. Vitello, U. Becciani, C. Buemi, F. Bufano, A. Calanducci, F. Cavallaro, A. Costa, A. Ingallinera, P. Leto *et al.*, "Caesar source finder: recent developments and testing," *Publications of the Astronomical Society of Australia*, vol. 36, p. e037, 2019.

[37] S. Riggi, A. Ingallinera, P. Leto, F. Cavallaro, F. Bufano, F. Schillirò, C. Trigilio, G. Umana, C. S. Buemi, and R. P. Norris, "Automated detection of extended sources in radio maps: progress from the scorpio survey," *Monthly Notices of the Royal Astronomical Society*, vol. 460, no. 2, pp. 1486–1499, 2016.

[38] A. Robotham, L. Davies, S. Driver, S. Koushan, D. Taranu, S. Casura, and J. Liske, "Profound: source extraction and application to modern survey data," *Monthly Notices of the Royal Astronomical Society*, vol. 476, no. 3, pp. 3137–3159, 2018.

[39] C. Hale, A. Robotham, L. Davies, M. Jarvis, S. P. Driver, and I. Heywood, "Radio source extraction with profound," *Monthly Notices of the Royal Astronomical Society*, vol. 487, no. 3, pp. 3971–3989, 2019.

[40] P. J. Hancock, T. Murphy, B. M. Gaensler, A. Hopkins, and J. R. Curran, "Compact continuum source finding for next generation radio surveys," *Monthly Notices of the Royal Astronomical Society*, vol. 422, no. 2, pp. 1812–1824, 2012.

[41] P. J. Hancock, C. M. Trott, and N. Hurley-Walker, "Source finding in the era of the ska (precursors): Aegean 2.0," *Publications of the Astronomical Society of Australia*, vol. 35, p. e011, 2018.

[42] D. Carbone, H. Garsden, H. Spreeuw, J. D. Swinbank, A. J. van der Horst, A. Rowlinson, J. W. Broderick, E. Rol, C. Law, G. Molenaar *et al.*, "Pyse: Software for extracting sources from radio images," *Astronomy and computing*, vol. 23, pp. 92–102, 2018.

[43] L. R. Makovoz D., Moshir M. and M. K., "Apex: A point source extractor for sirtf," *Astronomical Data Analysis Software and Systems XI, ASP Conference Proceedings*, vol. 281, p. 417, 2002.

[44] C. A. Hales, T. Murphy, J. R. Curran, E. Middelberg, B. M. Gaensler, and R. P. Norris, "Blobcat: software to catalogue flood-filled blobs in radio images of total intensity and linear polarization," *Monthly Notices of the Royal Astronomical Society*, vol. 425, no. 2, pp. 979–996, 2012.

[45] M. López-Caniego, D. Herranz, J. González-Nuevo, J. Sanz, R. Barreiro, P. Vielva, F. Argüeso, and L. Toffolatti, "Comparison of filters for the detection of point sources in planck simulations," *Monthly Notices of the Royal Astronomical Society*, vol. 370, no. 4, pp. 2047–2063, 2006.

[46] A. Consortium, T. M. Franzen, M. L. Davies, E. M. Waldram, K. J. Grainge, M. P. Hobson, N. Hurley-Walker, A. Lasenby, M. Olamaie, G. G. Pooley *et al.*, "10c survey of radio sources at 15.7 ghz–i. observing, mapping and source extraction," *Monthly Notices of the Royal Astronomical Society*, vol. 415, no. 3, pp. 2699–2707, 2011.

[47] A. M. Hopkins, M. T. Whiting, N. Seymour, K. Chow, R. P. Norris, L. Bonavera, R. Breton, D. Carbone, C. Ferrari, T. Franzen *et al.*, "The askap/emu source finding data challenge," *Publications of the Astronomical Society of Australia*, vol. 32, p. e037, 2015.

[48] S. Riggi, D. Magro, R. Sortino, A. De Marco, C. Bordiu, T. Cecconello, A. Hopkins, J. Marvil, G. Umana, E. Sciacca *et al.*, "Astronomical source detection in radio continuum maps with deep neural networks," *Astronomy and Computing*, vol. 42, p. 100682, 2023.

[49] W. Abdulla, "Mask r-cnn for object detection and instance segmentation on keras and tensorflow," *GitHub repository*, 2017.

[50] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[51] R. Girshick, "Fast r-cnn," pp. 1440–1448, 2015.

[52] V. Lukic, F. de Gasperin, and M. Brüggen, "Convosource: radio-astronomical source-finding with convolutional neural networks," *Galaxies*, vol. 8, no. 1, p. 3, 2019.

[53] S. Rezaei, J. P. McKean, M. Biehl, and A. Javadpour, "Decoras: detection and characterization of radio-astronomical sources using deep learning," *Monthly Notices of the Royal Astronomical Society*, vol. 510, no. 4, pp. 5891–5907, 2022.

[54] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE transactions on systems, man, and cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

APPENDICES

A DeepFocus denoising step

DeepFocus [6, 7] provides pre-built networks, which we can directly train on our data by simply changing the size of the input such that the model's input matches our dataset dimensions. We tried to use those networks with basic parameters but unfortunately, the model produced every time a trivial solution, which often results in setting all pixels to zeros or producing the same output whatever the input. We then tried playing with different parameters to solve this issue. All the parameters we used and changed are detailed in Table A.1. We experimented with plenty of different combinations between them, i.e. approximately 50 different configurations.

Parameters	Values tried
Number of iterations	200, 500, 1000, and 2000
Learning rate	0.1, 0.01, 0.001, but also starting with 0.1 and then decreasing it to 0.01 and even to 0.001
Number of hidden layers	64, 128, 256, 512, 1024
Batch size	64, 128, 256
Weight decay	0, 1e-3, 1e-5, 1e-7, 1e-10
Optimizer	Adam and Stochastic Gradient Descent
Loss	L1 Loss and MSE

Table A.1: The investigated configurations of the DeepFocus convolutional neural network architecture

At some point, we even tried using different configurations for the residual neural network architecture proposed by DeepFocus. Those ones are detailed in Table A.2.

Parameters	Values tried
Number of iterations	200, 500, and 1000
Learning rate	0.1, 0.01, 0.001, but also starting with 0.1 and then decreasing it to 0.01 and even to 0.001
Number of hidden layers	32, 64, 128, 256, and 512
Batch size	64, 128, 256
Optimizer	Adam and Stochastic Gradient Descent
Loss	L1 Loss

Table A.2: The investigated configurations of the DeepFocus residual neural network architecture