Auto-CryoET: Automatización de la segmentación de tomogramas

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Francisco Ucles Ayllón
Tutor/es:
Antonio Martínez Sánchez
Miguel Ángel Meroño Bayo

Facultad de Informática
UMU

15 de Julio de 2026 Convocatoria de Mayo/Junio

Auto-CryoET: Automatización de la segmentación de tomogramas

Desarrollo de una herramienta automática para segmentación de tomogramas de Cryo-ET

Autor
Francisco Ucles Ayllón

Tutor/es
Antonio Martínez Sánchez
Ingeniería de la Información y las Comunicaciones
Miguel Ángel Meroño Bayo
Matemáticas

UNIVERSIDAD DE MURCIA

Facultad de Informática

Grado en Ingeniería Informática

Murcia, 15 de Julio de 2026

Convocatoria de Mayo/Junio

Declaración firmada sobre originalidad del trabajo

D./Dña. **Francisco Ucles Ayllón**, con DNI **49170940E**, estudiante de la titulación de **Grado en Ingeniería Informática** de la Universidad de Murcia y autor del TFG titulado "**Auto-CryoET: Automatización de la segmentación de tomogramas**".

De acuerdo con el Reglamento por el que se regulan los Trabajos Fin de Grado y de Fin de Máster en la Universidad de Murcia (aprobado C. de Gob. 30-04-2015, modificado 22-04-2016, 28-09-2018 y 28/06/2022), así como la normativa interna para la oferta, asignación, elaboración y defensa de los Trabajos Fin de Grado y Fin de Máster de las titulaciones impartidas en la Facultad de Informática de la Universidad de Murcia (aprobada en Junta de Facultad el 19-06-2025)

DECLARO:

Que el Trabajo Fin de Grado presentado para su evaluación es original y de elaboración personal. Todas las fuentes utilizadas han sido debidamente citadas. Así mismo, declara que no incumple ningún contrato de confidencialidad, ni viola ningún derecho de propiedad intelectual e industrial

Murcia, a 15 de Julio de 2026

Fdo.: Francisco Ucles Ayllón
Autor del TFG

Resumen

En este trabajo, se presenta el diseño, implementación y prueba de concepto de una herramienta de automatización para el análisis de tomogramas en tomografía crioelectrónica. Se toman como referencia otras herramientas existentes que permiten realizar el proceso manualmente. De esta manera, se ha realizado una prueba de concepto para mostrar el correcto funcionamiento de la herramienta desarrollada.

En el primer capítulo se introducen aquellos conceptos y técnicas necesarias para comprender la naturaleza del problema. Desde los fundamentos de la tomografía crioelectrónica hasta una breve revisión de la teoría matemática que fundamenta el análisis automatizado.

En el segundo capítulo se analiza el estado actual del problema. Tiene como objetivo principal contextualizar qué herramientas existen actualmente y sus filosofías para resolver el problema. Asimismo, también se muestra el procedimiento manual que se toma como punto de control para verificar el correcto funcionamiento posterior de la herramienta.

En el tercer capítulo se explican las motivaciones para el desarrollo de la herramienta y sus objetivos principales. Además, también se exploran las diferentes librerías y herramientas que han sido utilizadas para el desarrollo. De la misma manera, se detallan las especificaciones técnicas utilizadas para realizar los experimentos, tanto hardware como software.

En el cuarto capítulo se muestran tanto el diseño como la implementación de la herramienta. Durante la parte del diseño se detallan los motivos que han llevado a desarrollar la herramienta de forma modular; todas las etapas de las que dispone la herramienta, y detalles importantes para su utilización como los datos esperados de entrada y salida para cada fase. Durante la parte de la implementación se ahonda en todas las cuestiones técnicas y propias del desarrollo de cada etapa. Cuando se trate de una etapa basada en una herramienta externa, entonces se explican y detallan los parámetros utilizados para un funcionamiento óptimo. Cuando se trate de una etapa desarrollada, entonces se explican los algoritmos utilizados, el flujo de la etapa, sus librerías utilizadas y más cuestiones técnicas. Además, al final del capítulo, se muestra la interfaz de la herramienta, así como las diferentes opciones que nos ofrece para ejecutar cada etapa desarrollada.

En el quinto capítulo se muestran los experimentos que se han llevado a cabo con la herramienta. Se detallan todas las fases de los experimentos partiendo de la obtención de los tomogramas hasta un análisis pormenorizado de aquellas etapas que producen un resultado visualizable. Además, en este capítulo también se encuentra la prueba de concepto que demuestra la bondad y correcto funcionamiento de la herramienta diseñada. Para hacer esto, se vale de la solución proporcionada por el proceso manual descrito durante el segundo capítulo.

En el sexto y último capítulo se realiza una conclusión donde se expone cómo la herramienta ha sido capaz de superar la prueba de concepto con una escasa intervención manual. Además, también se analizan líneas futuras que se pueden abordar para continuar con la investigación y mejorar la herramienta aún más.

Extended Abstract

In this thesis, it is explored how data analysis in cryogenic electron tomography (Cryo-ET) can be improved. Different from currently used tools, this new tool aims to leverage machine learning and mathematically grounded techniques to enable a more autonomous and reproducible analysis workflow.

In the first chapter, the basic concepts and background knowledge necessary to understand how this tool works and why it matters are presented. Firstly, it is explained what cryogenic electron tomography is and how data is obtained: sampling, vitrification, electron microscopes used to take 2D images, and 3D tomogram reconstruction. Besides, it is remarked how the Fourier Slice Theorem underpins this procedure. Secondly, machine learning techniques for tomogram analysis are introduced. Mainly, it is discussed how tools like TomoTwin offer significant advantages over manual procedures. Thirdly, a computational topology framework for data analysis is introduced: Discrete Morse Theory. This theory provides algorithms to handle image noise and identify high-density regions in 2D images. Persistent Homology is briefly discussed since it reduces noise significantly, and the cancellation of critical simplices algorithm is introduced, as it enables the extraction of local maxima from scalar functions such as density fields on a 2D image. Finally, it is described how an existing tool, DisPerSE, implements these theoretical foundations.

In the second chapter, the state of the art is reviewed. On one hand, it is explored how different modern tools deal with the tomogram analysis problem, ranging from classical tools such as TomoSegMemTV, which approaches the problem from an algorithmic perspective, to modern methods like MemBrain that use machine learning models for the analysis. Furthermore, it is also noted that while some tools aim to produce a full segmentation of the tomogram like MemBrain, others like TomoTwin provide a high-dimensional space where similar structures can be gathered. On the other hand, a manual procedure provided by TomoTwin is examined to segment the tomograms using visual tools like Napari. This manual procedure is also used later to serve as a baseline for evaluating the developed tool. Additionally, a concrete example using a synthetic tomogram is provided to support subsequent validation.

In the third chapter, the main reasons why this development has been made are explained. Reasons such as needing an easier and faster way to analyze stacks of tomograms are considered. Moreover, the objectives of the thesis are presented, as they

serve as the goals to pursue. The most important one is to connect tools originally designed for unrelated purposes in order to improve tomogram analysis. For example, the topological tool DisPerSE was originally designed to detect structures in cosmological contexts, not for biological applications. Furthermore, the main content of this chapter is exploring and defining all the technologies and tools that are used. From libraries such as Pandas and NumPy to applications such as Napari or Paraview, all these elements are studied. For each element, the rationale behind its selection and its integration within the developed tool are discussed. Regarding the analysis component, TomoTwin has been chosen due to its versatility. Although other tools could achieve greater accuracy on certain tomograms, TomoTwin has demonstrated in its original publication that it produces consistent results across many different types of tomograms. Since the tool is intended to be agnostic to the tomogram type, TomoTwin was selected. Another important consideration is the topological aspect. DisPerSE is chosen as an implementation of Discrete Morse Theory since it has proven to yield good results in other contexts. This chapter also includes a section detailing the specifications of the server used during the testing phase. Both software and hardware are described in considerable detail to provide a reproducible environment.

In the fourth chapter, the design and implementation of the tool are described. It explains how the tool is structured and how the implementation was carried out. It comprises four main phases, each of which includes further steps, and an optional one.

The first phase is the tomogram analysis phase. Here the tomogram is introduced and the first step is responsible for producing a 32-dimensional map of features by using TomoTwin. As this high-dimensional space is not directly manageable, a dimensionality reduction technique is applied: UMAP. This algorithm is included and suggested by TomoTwin to perform the reduction in its article. Hence, this second step transforms the 32-dimensional space into a 2D space suitable for generating a 2D image. Both steps need to take care of the specifications of the underlying hardware.

The second phase is the pre-processing phase. Here the tool processes the 2D space generated by TomoTwin to meet DisPerSE's input requirements by creating a 2D image. Although DisPerSE supports many different formats, FITS images showed the best performance for the purposes of this tool during the tests. These are cosmological image files and are not easily visualizable, but they yielded the best results in the analysis. To improve even more the performance, this image generation uses a bounding box to adjust as much as possible the limits of the 2D space. This bounding box can be generated automatically, manually at the beginning or even interatively during the workflow. Additionally, DisPerSE performs significantly better when a mask indicating the relevant pixels is provided, also in FITS format. Mask generation requires a threshold to determine which pixel is relevant and which is part of the background. This threshold can be supplied manually at the beginning or interatively during the

workflow. Since FITS format is not convenient for visual inspection, these steps also generate the 2D space image and the mask in a standard format such as PNG. These PNG are not strictly necessary, as they do not play an important role in the workflow, but they are extremely suitable when it comes to visualize how the generated space is and verification processes described below. Finally, this phase produces a metadata file that stores the correspondence between 2D space points and pixels in the generated image. This is necessary to recover the original 2D space coordinates from selected pixels in order to identify the regions of interest in the tomogram.

The third phase is the topological analysis phase. It consists of two steps: the topological analysis itself and exporting the results. In the first step, DisPerSE takes the 2D image and the mask generated in the previous phase to find critical points. Using the value at each pixel, DisPerSE identifies local maxima and minima. It also generates the descending fields that define the regions associated with each maximum point. These points and regions are provided in a custom format that is not easily manageable. Therefore, DisPerSE includes a conversion tool that transforms this data into visual geometric structures in VTK format, which constitutes the second step. The results from DisPerSE are converted into VTK formats that are then processed in the next phase to obtain the final result. These formats are different for the critical points selected, vtp format, and the manifolds, vtu format.

The fourth phase is the post-processing phase. This phase is responsible for combining the results from DisPerSE with other data generated previously to produce visualizable output. This phase requires six distinct steps to transform the data appropriately. Some of them can be executed simultaneously, but the tool is not implemented to use concurrency. Introducing concurrency would increase complexity without returning significantly greater efficiency. The first two steps are executed simultaneously since they do not depend on each other. The first one takes all the generated manifolds and splits them from the single polygonal complex produced in the previous phase into individual files. The second one filters the selected critical points. DisPerSE generates a file containing all analyzed critical points, both maxima and minima, along with their connections. Since only the maxima are relevant to the tool, this step filters out the remaining elements. Additionally, a pruning operation can be applied here, discarding maxima points that fall below a certain threshold, retaining only those of sufficient significance. This threshold selection can be made at the beginning or interactively during the workflow. The third step requires the output of both previous steps. It selects the manifolds associated with the critical points filtered in the second step, using the name of each manifold produced in the first step together with the point attributes from the second step. The next two steps can also be executed simultaneously as they have no mutual dependencies. The fourth step, while not strictly necessary, is highly useful for validation. It takes the 2D image of the space and the selected manifolds and renders them together, assigning a distinct color to each manifold and overlaying

it on the 2D image. This visualization is not used elsewhere in the workflow, but it is crucial for the human validation of the segmentation process, as it is the only way to observe which points in the 2D image the tool is selecting. The fifth step is the central one in this phase. It is worth noting that DisPerSE operates entirely on the 2D image and has no knowledge of the original 32-dimensional space. However, generating the segmentation requires working with tomogram voxels, which can only be obtained from the 32-dimensional space. This step takes the filtered manifolds, the metadata file generated in the pre-processing phase, and the 32-dimensional space file produced in the tomogram analysis phase to recover the manifold coordinates in the 32-dimensional space. Since the metadata file stores the correspondence between each pixel in the 2D image and the 32-dimensional space, combining these data sources allows the manifolds to be transformed from image coordinates back into the original space. The sixth and final step generates the segmentation. It receives both the original tomogram and the manifolds recovered in the 32-dimensional space, and produces an MRC file containing the segmentations overlaid on the tomogram. Additionally, individual MRC files can be generated for each manifold to facilitate verification.

The fifth and last phase is, as mentioned, completely optional. It is responsible for visualizing the segmentation over the original tomogram. To achieve a user-friendly representation, it uses Napari to create layers with different colors depending on the MRC files generated in the previous step.

Before concluding the fourth chapter, the design of the terminal interface that integrates all phases into a functional and user-friendly tool is also presented. Although each step can be executed manually via command line, a terminal-based tool was developed to ease the process. A terminal interface was chosen over a graphical application because this type of analysis is typically performed on remote servers or computing clusters where a graphical environment is not always available.

In the fifth chapter, the experiments carried out to verify the tool's functionality are explained. All phases of the experiments are covered. Firstly, it is described how the synthetic data used are generated. Secondly, the tool is shown completing the workflow successfully. Thirdly, every step of the process is reviewed and the rationale behind each parameter choice is explained. In the pre-processing step, several examples of the 2D image are provided. These examples illustrate how different parameter choices for the Gaussian filter affect the visualization of the data. Subsequently, it is described how the topological analysis must be adjusted through the persistence parameter, and how an inappropriate choice of this parameter could critically affect the quality of the analysis. After that, the critical points selected during the experiments are shown. These results demonstrate that the procedure is accurate at this stage, as the denser regions are correctly identified. Finally, both the selected regions and the 2D image of the space are shown to confirm that each selected point corresponds to its own region.

A figure comparing the segmentation produced by the developed tool against the one generated manually in the second chapter is also included, and is used to conclude that both workflows produce equivalent results.

In the sixth and final chapter, a conclusion to the whole study is presented. It is demonstrated how the tool completes the analysis workflow with minimal human intervention. A comparison between the manual procedure explored in the second chapter and the experiments conducted in the fifth one shows that the new tool obtains equivalent results, while additionally identifying not just one but all possible regions of interest. It is also noted that, thanks to the modular design described in the fourth chapter, the analysis can be refined using the same tool to improve results. In relation to this, future improvements are also examined. For example, it is suggested that future work should focus on retraining the TomoTwin model to better fit the specific tomograms used, or on exploring alternative tools for the embedding and analysis stages.

Índice general

Índice de figuras

1. Introducción

Este capítulo tiene la finalidad de introducir todos los conceptos que serán utilizados más adelante. Estos están relacionados áreas del conocimiento como matemáticas. Por lo tanto, y con la finalidad de hacer accesible el documento a un lector no experto en estos campos, se va a sacrificar cierto rigor y profundidad.

1.1. Tomografía crioloectrónica

La tomografía crioelectrónica (Cryo-ET) es una técnica de microscopía electrónica que permite obtener una imagen tridimensional (3D) a partir de una muestra. Esta técnica obtiene imágenes de alta resolución a una escala atómica y mantiene la muestra en un estado muy cercano al nativo. Estas imágenes 3D son las denominadas tomogramas.

El proceso para la obtención de un tomograma a partir de una muestra consta de un flujo de trabajo (workflow) con varios pasos importantes [2, 6, 7]:

1. **Obtención de la muestra**: lo primero de todo debe ser la obtención de la muestra. Si queremos obtener una imagen de gran resolución, la muestra no puede tener un grosor mayor de 300nm [6]. De lo contrario, habrá zonas demasiado densas y gruesas para que el microscopio electrónico pueda verlas.

2. **Vitrificación**: la muestra seleccionada debe someterse a un proceso de criogenización ultrarrápido llamado vitrificación. Este proceso permite preservar el estado de la muestra a escala molecular. Gracias a esto, ciertos procesos internos se pueden analizar de forma eficaz.

3. **Proyecciones 2D**: para obtener una imagen 3D es preciso hacer una reconstrucción basándose en otras 2D. Para ello se utiliza un microscopio electrónico sobre la muestra colocada sobre una superficie plana (pletina) para obtener una imagen 2D de la misma. Luego, se rota ligeramente la pletina y se obtiene otra imagen. Iterando este proceso barriendo ángulos desde -60^{o} hasta 60^{o} se obtienen un conjunto de imágenes 2D denominadas **micrografías electrónicas** como se aprecia en el diagrama de la Figura 1.1. Por cuestiones ópticas, rotar una muestra provocará que el grosor aumente en la dirección del microscopio en razón del seno del ángulo de rotación. Este hecho volverá la muestra progresivamente más opaca desde el punto vista del microscopio. Este fenómeno se conoce como

missing wedge [8] que crea zonas completamente oscuras dentro del tomograma. Existen técnicas de post-procesado que permiten reconstruir ciertas partes, pero en muchas ocasiones este fenómeno supone un auténtico obstáculo en el análisis.

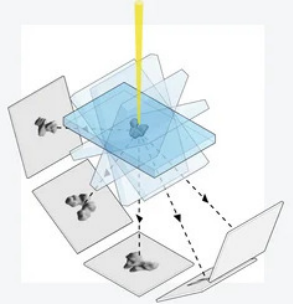


Figura 1.1: Diagrama que ilustra la rotación que se realiza de la muestra para obtener diferentes ángulos de la misma que servirán para la reconstrucción del tomograma [1].

4. **Reconstrucción**: finalmente se utilizan todas las micrografías electrónicas obtenidas de la muestra para realizar la reconstrucción 3D del tomograma. Este proceso de reconstrucción tomográfica es fundamental para la técnica y se sustenta matemáticamente en el **Teorema del Corte Central** [9]. Dicho teorema establece que la transformada de Fourier de una proyección bidimensional (*i.e.* micrografías) es equivalente a un corte central que pasa por el origen de la transformada de Fourier tridimensional del objeto original. Al tomar proyecciones desde múltiples ángulos se va rellenando el espacio de frecuencias tridimensional lo que permite, tras aplicar la transformada inversa, recuperar el volumen tridimensional de la muestra biológica. La Figura 1.2 muestra un diagrama representando el proceso.

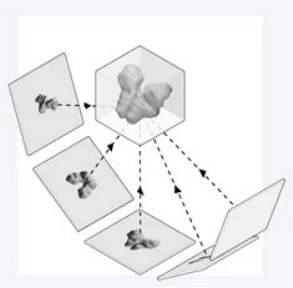


Figura 1.2: Diagrama que ilustra la reconstrucción del tomograma en base a las diferentes micrografías electrónicas [1].

Más allá de esos pasos básicos, existen procesos intermedios que preparan o mejoran la muestra para obtener una mejor imagen. La Figura 1.3 muestra un resumen de todo el proceso anteriormente descrito y algunas técnicas adicionales.

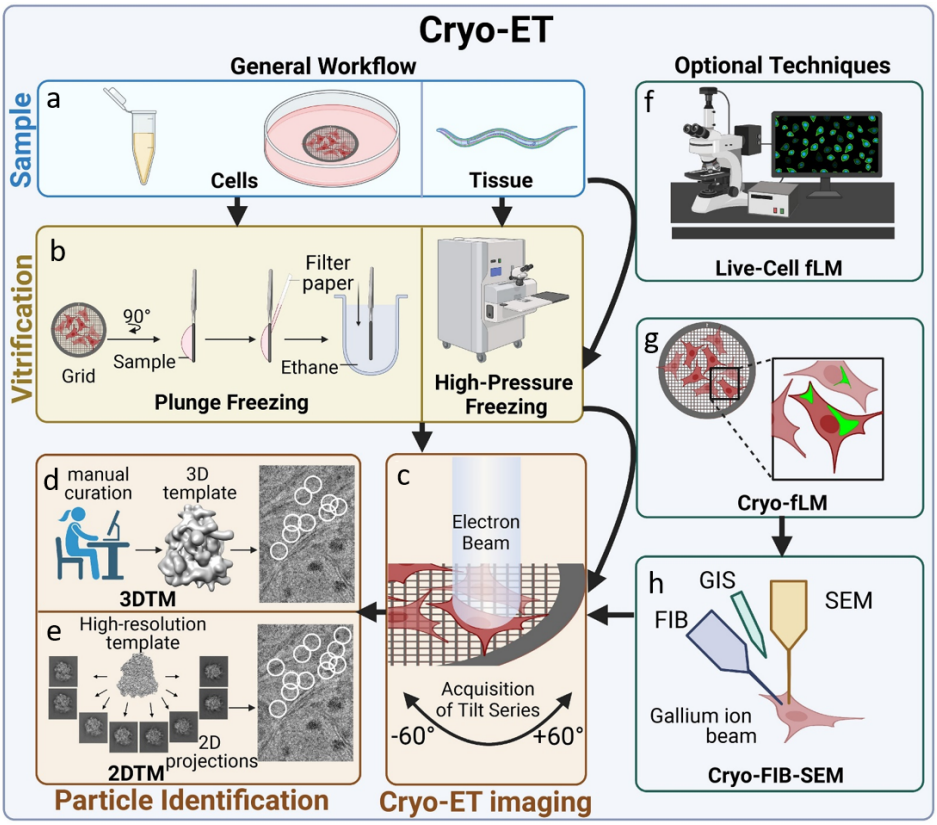


Figura 1.3: Workflow completo de la tomografía crioelectrónica donde se puede apreciar el primer paso de toma de muestras (a), luego la vitrificación de la muestra (b), la generación de las proyeciones 2D mediante microscopio electrónico (c), la reconstrucción tomográfica (e), el proceso de análisis tomográfico (d), procesos de microscopía óptica correlativos para identificar la región a visualizar en el microscopio electrónico (f)(g) y preparación de la muestra (*i.e.* adelgazamiento) para introducirla en el microscopio (h). Adaptada de [2].

Gracias a que los tomogramas son una imagen muy fidedigna de la muestra, estos se utilizan como medio para distintas identificar estructuras que permiten mejorar el entendimiento de ciertos procesos biológicos. Uno de los análisis más interesantes que se pueden llevar a cabo en un tomograma es la segmentación de todas las estructuras del mismo.

Debido a la naturaleza tridimensional de los tomogramas, resulta complicado visualizarlos. Sin embargo, existen software diseñados para este propósito (ver Sección 3.3) y también se pueden tomar secciones 2D de ese volumen. En la Figura 1.4 se observa uno de esos cortes 2D sobre un tomograma.

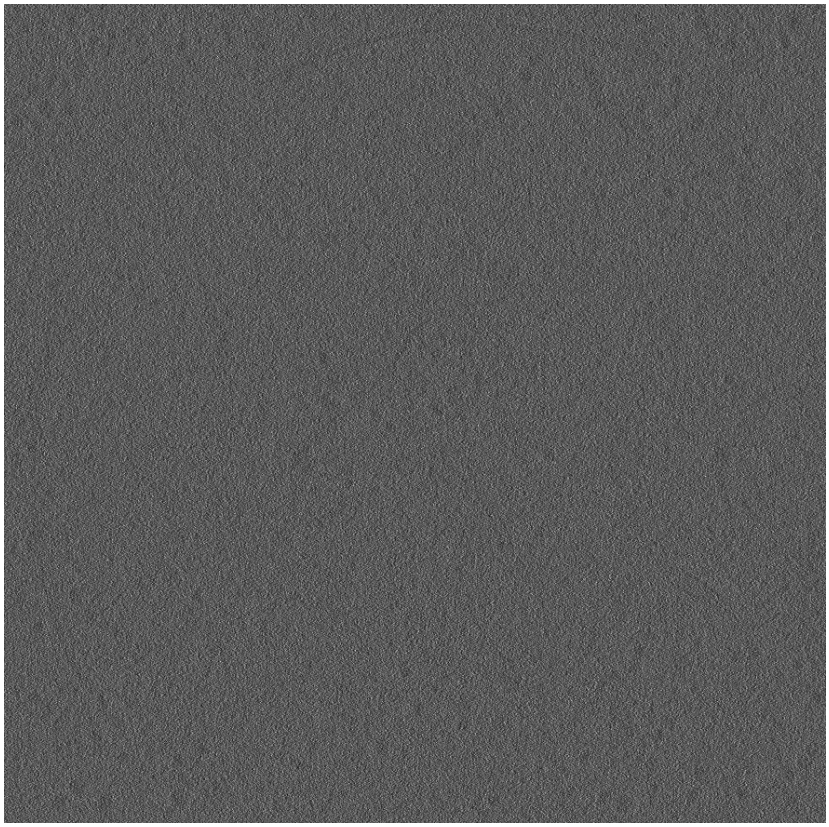


Figura 1.4: Proyección de una capa de un tomograma.

Para los tomogramas, como el de la Figura 1.4, se utilizan las regiones cúbicas dentro del tomograma (voxels) como unidad de trabajo. Los voxels serían a los tomogramas lo que los píxeles a las imágenes 2D. Por lo tanto, resultaría interesante y de gran utilidad agrupar todos los voxels del tomograma que compartan una misma estructura. Esto permitiría identificar qué estructuras existen en el tomograma. Si se obtienen todas las diferentes estructuras similares que hay dentro de un tomograma y se asigna una etiqueta a cada una de ellas, se crearía una segmentación del tomograma en sus diferentes elementos esenciales. Este es el proceso antes denominado como segmentación y tiene un gran valor científico.

Visualizar el tomograma y seleccionar manualmente cuales son las zonas con una estructura similar es un proceso lento y propenso a errores. Además, debido al factor humano, este proceso también es sensible a la experiencia y los sesgos de la persona que clasifica. Otro factor a tener en cuenta es el conocimiento previo de las estructuras. En este proceso manual se asume cierto conocimiento sobre las estructuras buscadas. No obstante, en numerosas ocasiones son las estructuras desconocidas o muy escasas las que aportan un mayor valor. Por esto mismo, existe un gran interés en desarrollar herramientas automatizadas y versátiles para llevar a cabo este proceso.

1.2. Análisis automatizado

Como se concluía en la sección anterior, es deseable tener un método automatizado para realizar la segmentación. Además, también conseguir que este proceso también permita identificar cualquier tipo de estructuras sin un conocimiento previo sobre cuales se encuentran en la muestra sería un gran avance. Como se explicará en el Capítulo 2, una idea prometedora es hacer uso de técnicas de *Machine Learning* (ML) para obtener las características más esenciales del tomograma y luego analizarlas. Suelen utilizarse redes neuronales profundas que siguen esquemas similares al de la Figura 1.5.

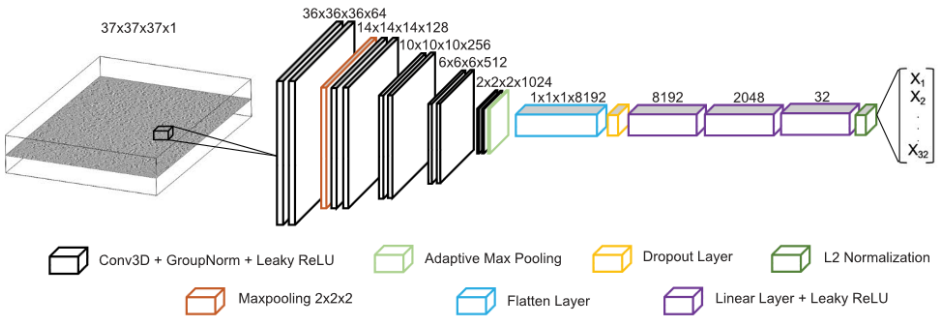


Figura 1.5: Diseño de la red neuronal de TomoTwin[3].

Una red neuronal, como la anterior, nos permite convertir una imagen 3D en una nube de vectores de características que encierran las propiedades más esenciales de cada uno de los voxels del tomograma. Estas herramientas proporcionan modelos ya entrenados con cientos de tomogramas diferentes y permiten entrenar modelos propios.

1.3. Teoría de Morse discreta y Persistencia homológica

En la sección anterior se mostró cómo se podía obtener un mapa de características (espacio latente) a partir del tomograma con técnicas de ML. Ahora cabría abordar la cuestión de cómo se van a analizar esas características. En el contexto de las redes neuronales, dos puntos del espacio latente que se encuentren próximos representan conceptos similares. Por lo tanto, uno podría esperar que si tomamos el espacio latente generado por herramientas como las antes mencionadas, aquellos aquellas acumulaciones puntos serán zonas del tomograma similares. Dicho de otra forma, se puede ver el espacio latente como muestras de un mapa de densidad cuya forma codifica las diferentes estructuras presentes en el tomograma. Precisamente debido a la importancia en la forma y estructura del espacio latente es importante que el análisis que se realice tenga estos datos en cuenta y no los modifique o degrade. Exactamente ahí entra la topología computacional.

La topología computacional es una rama de las matemáticas que utiliza herramientas topológicas implementadas en ordenadores para hacer análisis de ciertas funciones. En particular, una de las herramientas más potentes es la **Teoría de Morse Discreta**. Esta teoría da un marco riguroso para extraer la estructura de una función sobre una superficie discreta. Esta teoría permite encontrar puntos críticos (*i.e.* máximos y mínimos) de una función sobre una superficie discreta (*i.e.* malla geométrica) y resulta de gran interés en este ámbito. Para el caso que nos concierne, se puede considerar que la superficie geométrica es el espacio latente visto en 2D. Ahora bien, el otro ingrediente faltante es la función a utilizar. Una pregunta importante es, ¿qué función resultaría de utilidad para el análisis? Como se ha expresado anteriormente, conocer las zonas del espacio latente con mayor densidad de puntos es muy relevante. Por eso mismo, un buen candidato para función sería la densidad de cada punto. La densidad de una nube puntos tiene diversos métodos para calcularse que se salen del propósito de este documento, pero que pueden consultarse en [10] y que se ejemplifica en la Figura 1.6.

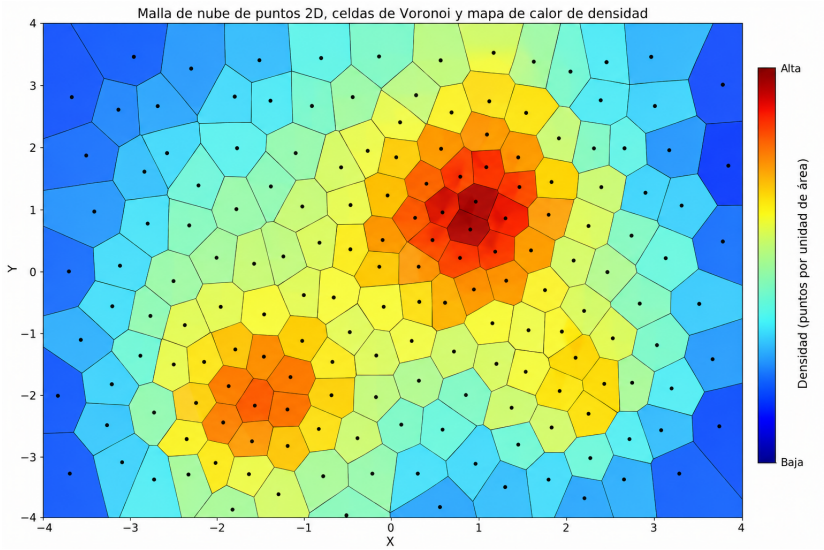


Figura 1.6: Nube puntos y sus valores de densidad asociados.

Antes de proceder con la simplificación de símplices críticos, hay de abordar dos conceptos de teoría de Morse clásica que pueden extrapolarse a la teoría de Morse discreta: *ascending/descending manifolds*. Aunque la definición rigurosa es más complicada y puede consultarse en [11], podrían considerarse los *ascending manifolds* como aquellas zonas alrededor de los mínimos que se "sienten atraídos hacia ellos"; mientras que los *descending manifolds* son aquellas zonas alrededor de un máximos que se "siente repelidos por ellos". Usando como analogía el relieve montañoso, podría decir que los *descending manifolds* y los máximos son las montañas y sus picos, respectivamente; mientras que los *ascending manifolds* y los mínimos serían los valles y sus fondos, respectivamente.

Una vez que se tienen claros todos estos conceptos, existen algoritmos que aplicables a estas funciones y mallas geométricas como el **Algoritmo de cancelación de símplices críticos**. Este permite reducir la cantidad de puntos críticos sin alterar la estructura de la superficie. Aunque una explicación rigurosa del algoritmo y su implementación se sale del objetivo del documento, es preciso obtener una idea intuitiva de cómo funciona.

Para ello, se va a utilizar el ejemplo que muestra DisPerSE [4]. Véase que en la Figura 1.7 aparece una malla de puntos y cada uno tiene un valor asociado. En particular, aparecen 2 máximos (rojo) y un punto de silla (verde). Como hay un camino que une el máximo etiquetado con 19 y el punto de silla etiquetado con 18 estos se pueden seleccionar para simplificarlos. Una vez seleccionados, solo se deben eliminar estos nodos y reconectar la red para que no queden inconsistencias. Se puede notar que todas las conexiones del máximos etiquetado por 19 van al otro máximos etiquetado con 21 por

estar conectado al punto de silla simplificado. Además, también hay que encargarse de fusionar los *manifolds* asociados a estos puntos críticos que desaparecen. De esta manera, los dos puntos críticos habrán sido modificados manteniendo la estructura de la malla. Como se explicará a continuación, el emparejamiento no se hace de forma arbitraria, sino que se utiliza la persistencia.

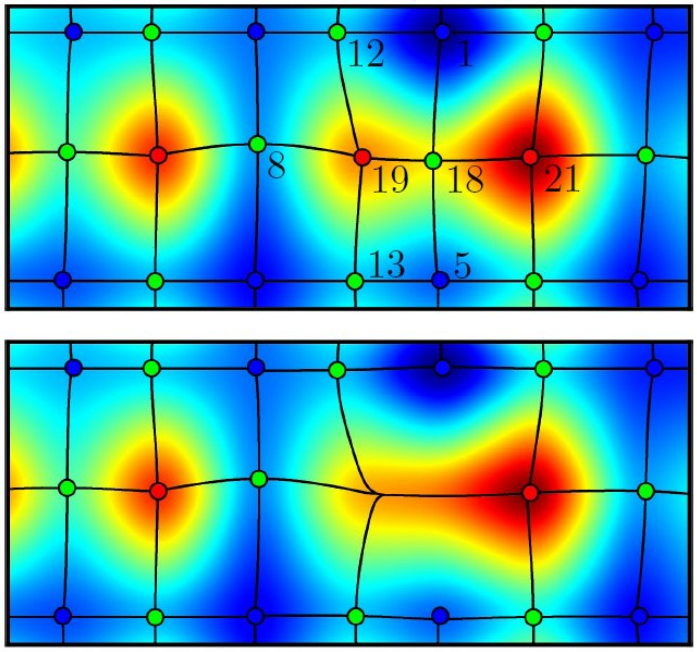


Figura 1.7: Imagen que representa en la parte superior un complejo de Morse con máximos (rojo), puntos de silla (verde) y mínimos (azul) que se quiere simplificar. En la parte inferior se muestra el resultado de la simplificación de los nodos 18 y 19 [4].

Otra herramienta matemática que es fundamental para análisis de este estilo es la **Persistencia homológica**. Esta permite hacer algo que es extremadamente importante cuando se trata con imágenes: eliminar el ruido. La persistencia mide la "robustez" o importancia de un punto crítico. Si se visualizan una función como una altura y se procede a llenar este mapa de altitudes con agua, la persistencia de un mínimo es la cantidad de agua (*i.e.* diferencia en el valor de la función) que debe acumularse en él antes de desbordarse y fusionarse con un valle vecino más profundo. Esto se puede apreciar en la Figura 1.8.

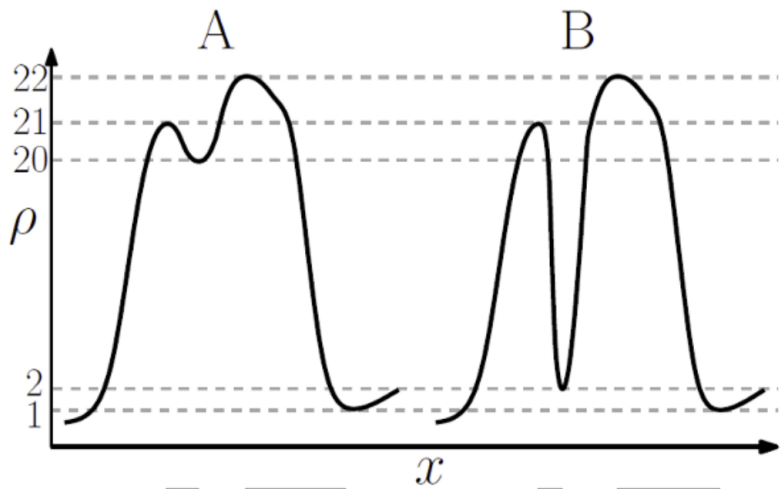


Figura 1.8: Diagrama de función donde se puede aplicar la persistencia y que se apreciar como en la función existen valles más o menos profundos [4].

De esta forma, se obtiene la siguiente intuición:

- **Baja persistencia (Ruido)**: Micro-valles muy superficiales que desaparecen rápidamente al fusionarse. Corresponden a fluctuaciones aleatorias o artefactos generados.
- **Alta persistencia (Señal)**: Pozos de densidad profundos y bien definidos que requieren una gran variación de densidad para unirse a otros. Estos corresponden en principio a las verdaderas proteínas y estructuras del tomograma.

De nuevo, la fundamentación teórica de estas técnicas se sale del objetivo de este documento, pero una introducción al tema puede verse en [12].

2. Estado del arte

2.1. Herramientas de segmentación

Actualmente, existen diversos estudios e investigaciones que tienen como objetivo desarrollar herramientas para facilitar la segmentación de tomogramas. Como ya se ha comentado anteriormente, este proceso se realizaba de forma manual y era propenso a errores. Ante esto, surgieron una serie de ideas que pretendían automatizar estos procesos.

En sus inicios, había métodos que utilizaban diferentes algoritmos para realizar una segmentación del tomograma. Como se muestra en [13], una opción era tomar los voxels para luego utilizar un algoritmo como el *Tensor Voting Algorithm* [14] para segmentar el tomograma. Sin embargo, este tipo de técnicas de visión artificial, aunque daban resultados, tenían numerosas limitaciones y no aportaban un proceso completamente automático.

Con el tiempo y la llegada de las redes neuronales se vio que usar procesos de ML para la identificación de las estructuras y proteínas podía suponer un gran avance. Aquí entran herramientas como MemBrain [15], TomoTwin [3], MiLoPYP [5] o TomoSegNet [16].

Herramientas como MemBrain o TomoSegNet son herramientas que realizan la segmentación de las membranas directamente. No muestran un espacio latente donde se pueda seleccionar zonas de interés, sino que directamente te proporcionan una segmentación de las membranas. Estas herramientas, aunque pueda parecer que solventan todo el problema, solo funcionan correctamente con membranas y muestras muy concretas.

Por otro lado, tanto TomoTwin como MiLoPYP tienen una filosofía distinta a los anteriores. Ellos se basan en la idea de proporcionar el espacio latente y dejar a criterio del investigador qué zonas son o no interesantes de él. Los pasos de este proceso serían los siguientes:

1. Los tomogramas son procesados mediante una red neuronal ya entrenada para obtener un espacio latente.

2. Mediante alguna herramienta de visualización se permite que el usuario seleccione manualmente las secciones del espacio latente que le resulten de interés.

3. Esas selecciones son asociadas a las regiones del tomograma original agrupando así todas las zonas relacionadas.

En la Figura 2.1 se muestran esos tres pasos en el workflow de MiLoPYP.

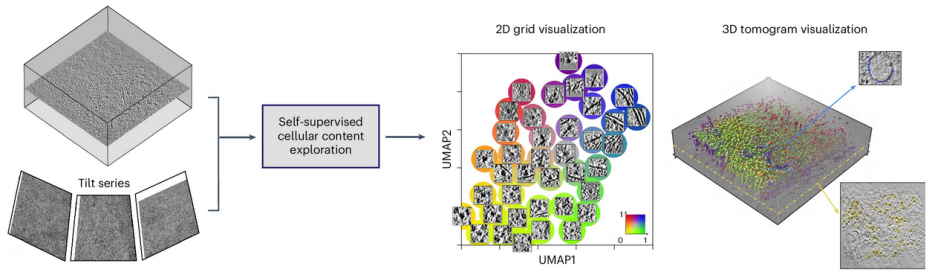


Figura 2.1: Workflow de MiLoPYP donde se puede ver la primera etapa procesada por la red neuronal, el espacio latente generado donde se seleccionan las zonas que se quieren segmentar y luego la proyección de las regiones del espacio latente dentro del tomograma original [5].

Algunas herramientas como TomoTwin tienen modos de funcionamiento adicionales basados en plantillas (*templates*). Para ello, se utiliza un tomograma donde se conozcan las estructuras donde se encuentran, se seleccionan en el espacio latente y luego se genera una plantilla. Esta plantilla la puede utilizar TomoTwin para encontrar esas estructuras en otro tomogramas.

Otras como MemBrain afirman ser el sucesor de TomoSegMemTV pasando del modelo de *Tensor Voting* a un modelo de ML y utilizan datos sintéticos para entrenar el modelo.

TomoSegNet está entrenado solo con datos sintéticos especialmente diseñados para representar la diversidad de topologías y geometrías de las membranas celular, lo que le permite predecir las membranas que desaparecen por efectos de las distorsión introducidas durante la reconstrucción tomográfica (*missing wedge*).

Incluso algunas como MiLoPYP tienen grupos de investigación creando herramientas *user-friendly* con el objetivo de extender su uso a cientos de científicos sin tener que lidiar con la complejidad de los comandos.

2.2. Métodos pseudo-automáticos

En esta sección, se va a ahondar con detalle en cómo funcionaría uno de los workflow de las herramientas anteriores. En particular, se mostrará el funcionamiento de TomoTwin [3] pues será la herramienta que más adelante se utilizará para la automatización del workflow. El proceso a continuación descrito se desarrolla por los autores de TomoTwin en [17]. Como los propios autores comentan, se utilizará Napari (ver Sección 3.3.4) y un plugin desarrollado especificamente para él.

Lo primero sería utilizar su modelo y una herramienta de reducción dimensional como Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) para generar el espacio latente. Luego, este espacio latente en 2D se carga junto al tomograma en el propio Napari como se puede ver en la Figura 2.2.

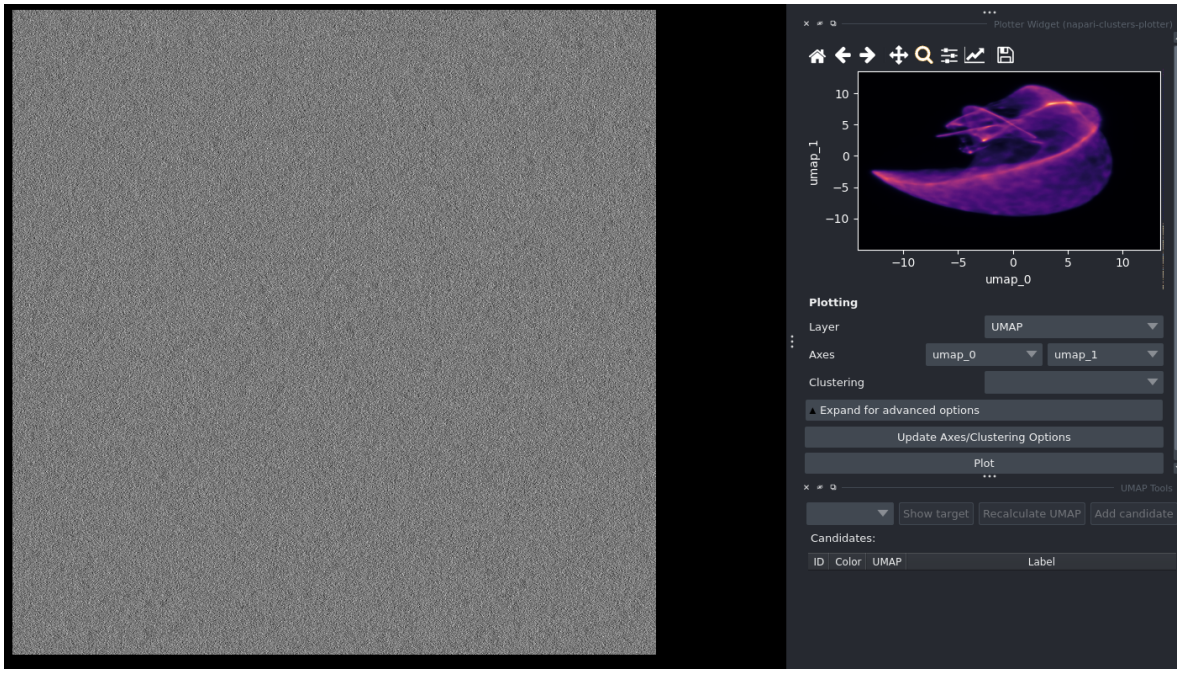


Figura 2.2: Imagen de Napari con el tomograma a la izquierda y el espacio latente a la derecha.

Se observa en la Figura 2.2 como el espacio latente tiene zonas con más brillo. Estas corresponden a zonas del espacio latente de mayor densidad y se puede usar la herramienta de selección para ver en qué zonas del tomograma repercuten. En la Figura 2.3 se puede apreciar una selección del espacio latente y su proyección sobre el tomograma.

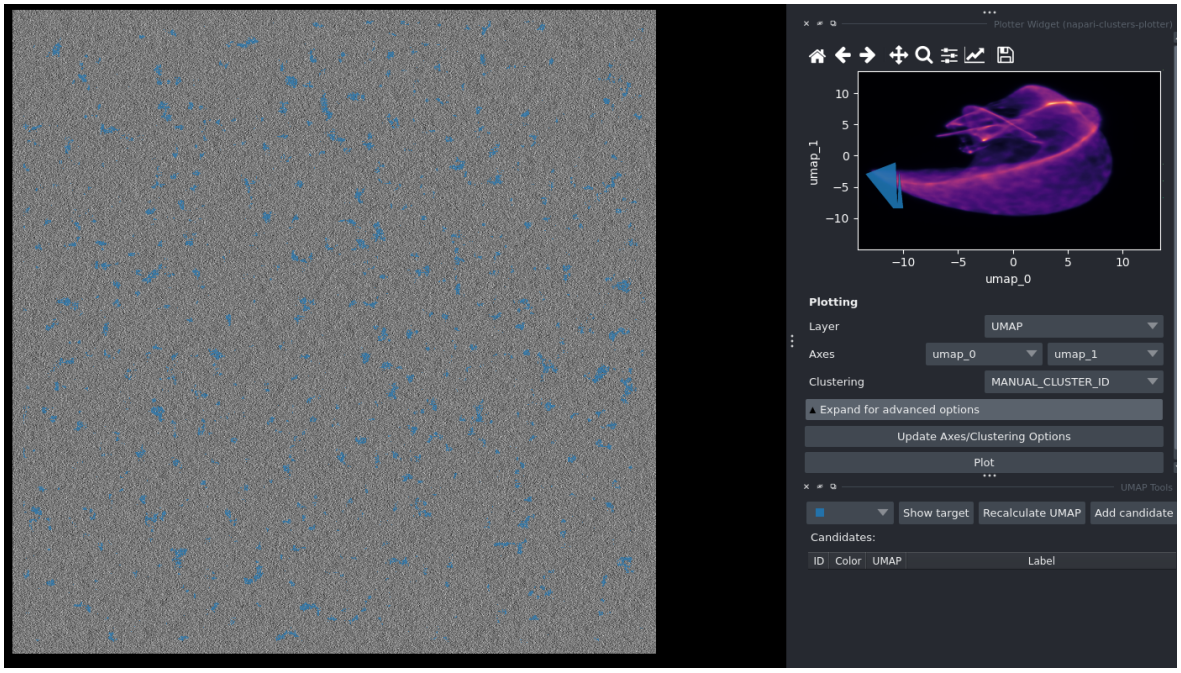


Figura 2.3: Imagen de Napari con una selección del espacio latente y sus áreas afectadas en el tomograma en color azul.

De esta manera, se puede ver cómo al seleccionar las zonas del tomograma se ha creado una segmentación de aquellas estructuras similares. Además, si ahora se exporta esta segmentación con `Napari` y se convierte a Medical Research Council (MRC), se puede visualizar en 3dmod (ver Sección 3.3.4) de forma cómoda. En la Figura 2.4 se puede apreciar la segmentación en 3dmod. Esta será la segmentación que se usará posteriormente en el Capítulo 5 para comparar la validez de la herramienta desarrollada.

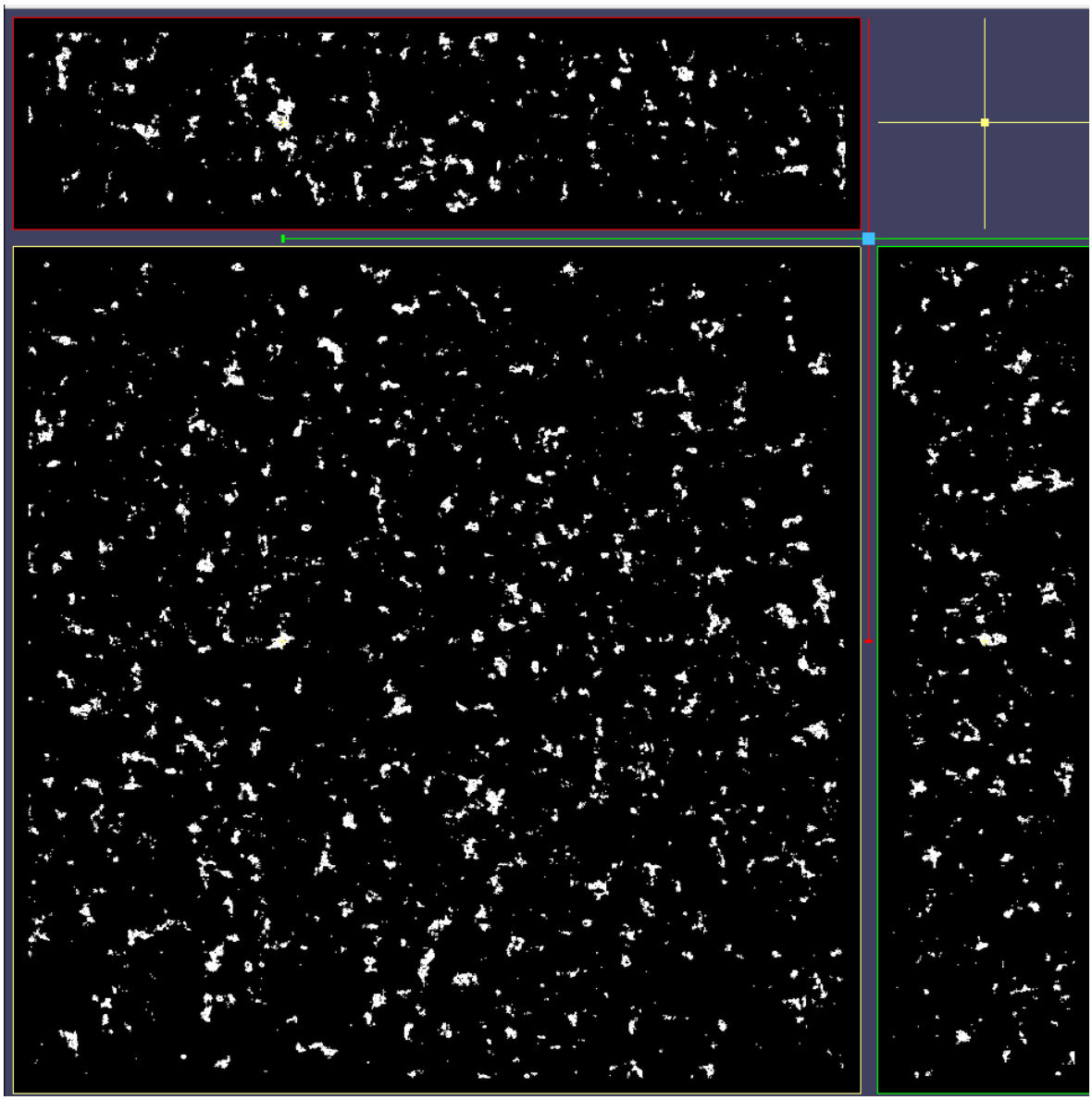


Figura 2.4: Imagen de 3dmod con la segmentación manual de una región del espacio latente.

3. Análisis de objetivos y metodología

3.1. Motivación

En el Capítulo 2 se ha mostrado como en la actualidad los métodos que existen precisan de una clasificación pseudo-manual. Aunque el campo ha avanzado mucho gracias a métodos de ML, aún queda mucho por desarrollar y automatizar. Depender de una persona experta en la materia que analice cada tomograma o cada espacio latente en busca de estructuras conocidas limita nuevos descubrimientos y es muy sensible a sesgos.

Para solventar esta limitante, se busca hacer uso de la topología computacional. Como se ha explicado en la Sección 1.3, la topología computacional proporciona una herramienta rigurosa para hacer análisis de datos. En particular, esta herramienta permitirá automatizar la búsqueda de elementos interesantes en los tomogramas que luego puedan ser revisados por una persona experta. Por lo tanto, la motivación de este trabajo es crear una herramienta automatizada que permita sortear problemas actuales como el ruido instrumental, los sesgos humanos y los cuellos de botella ligados al procesamiento de datos en Cryo-ET.

3.2. Objetivos

- **Objetivo principal**: Desarrollar una herramienta que permita automatizar la segmentación de tomogramas en Cryo-ET mediante técnicas de ML y topología computacional.

- **Objetivos específicos**:
  - Utilizar técnicas de ML como TomoTwin para obtener un espacio latente de un tomograma.
  - Utilizar técnicas de topología computacional basadas en teoría de Morse discreta para el análisis del espacio latente.
  - Trasladar la segmentación generada en el espacio latente para obtenerla en el tomograma.

3.3. Metodología y herramientas

Para el desarrollo y ejecución del programa propuesto en este trabajo, se ha optado por un ecosistema basado en **Python**. La elección de este lenguaje se justifica por su extensa compatibilidad con librerías científicas de alto rendimiento. Muchas de las herramientas precisan de dependencias antiguas o incompatibles con algunas versiones modernos. Por esto mismo, el desarrollo se ha estructurado aislando las dependencias mediante un entorno virtual, garantizando así la reproducibilidad. En particular, se ha utilizado el gestor de entornos virtuales y paquetes **Conda** por su amplia gama de herramientas y su conexión histórica con la bioinformática. Se ha utilizado un entorno especialmente diseñado con las dependencias necesarias para funcionar. Este entorno puede instalarse en el repositorio de la herramienta diseñada, aquí.

A continuación, se detallan las herramientas clave que han sido utilizadas en el proyecto y la justificación de su elección. Sin embargo, primero de todo, se van a detallar las especificaciones del servidor de pruebas utilizado.

3.3.1. Equipo utilizado

Para llevar a cabo las pruebas y resultados (ver Capítulo 5) se ha utilizado un servidor con una serie de características que han permitido la ejecución de todos los programas que se detallan posteriormente.

En cuanto al software del equipo, se ha utilizado el sistema operativo Ubuntu. En particular, se ha utilizado la versión **Ubuntu 22.04.5 LTS**. Además, debido a que este servidor está especializado en cálculo, la mayoría de los programas con una interfaz gráfica se han ejecutado remotamente mediante `ssh`.

En cuanto al hardware, las especificaciones relevantes son las siguientes:

- **CPU**: AMD EPYC 8434P 48-Core Processor, que corresponde a una arquitectura x86.
- **RAM**: 377 GB.
- **GPU**: dos NVIDIA GeForce RTX 4090, cada una con 24GB de memoria.

3.3.2. Gestión de volúmenes Cryo-ET: `mrcfile`

El estándar para el almacenamiento de tomogramas en microscopía electrónica es el formato binario MRC. Para su manipulación, se ha integrado la librería `mrcfile`[18]. Esta herramienta permite mapear de forma eficiente los bloques de datos masivos del tomograma directamente a memoria como matrices multidimensionales de `NumPy`. Esto

es vital en Cryo-ET, ya que, al tratarse de archivos muy pesados, cargar un tomograma completo de forma ineficiente provocaría el desbordamiento de la memoria RAM del sistema.

3.3.3. Análisis mediante ML: `TomoTwin`

Como se explica en el Capítulo 2, en la actualidad se utilizan herramientas de ML para obtener el espacio latente de los tomogramas. En particular, se utilizan redes neuronales con diferentes arquitecturas, como las que se especifican en la Sección 1.2. Aquí se hará uso de **TomoTwin** [3] que genera un espacio latente de 32 dimensiones.

Este programa también dispone de otras herramientas que serán de gran utilidad. Por ejemplo, el propio TomoTwin integra un algoritmo de reducción dimensional UMAP [19] que será de mucha utilidad para reducir el espacio latente a una dimensión razonable para trabajar (*i.e.* 2 o 3 dimensiones).

Finalmente, otro aspecto importante que se tomó en cuenta para elegir esta herramienta frente a otras es su universalidad. TomoTwin ha sido entrenado con proteínas y estructuras muy diversas lo cual hace que sea aplicable a muchos tipos de tomogramas e identificar una gran cantidad de estructuras diferentes.

3.3.4. Visualización de tomogramas: `Napari` y `3dmod`

Para poder visualizar los tomogramas de forma cómoda y fácil es necesaria una herramienta de visualización multidimensional. En particular, se va a utilizar tanto **`Napari`** [20] como **`3dmod`** [21].

En cuanto a `Napari`, esta herramienta permite visualizar cómodamente el tomograma por capas y además tiene una muy buena integración con TomoTwin. Los autores de TomoTwin explican en su página oficial [17] que han desarrollado un plugin para Napari. Este plugin permite llevar a cabo el workflow de TomoTwin de una forma más sencilla y visual. Además, tal y como se comenta en la Sección 2.2 esta herramienta pseudo-automática que utiliza Napari será un buen punto de referencia.

Por otro lado, `3dmod` se utilizará como herramienta ligera de visualización de ficheros MRC que permitirá comparar fácilmente los resultados de control manuales con los generados por la herramienta.

3.3.5. Estructuración de datos y procesamiento del espacio latente: `Pandas` y `NumPy`

A la hora de manejar los puntos del espacio latente obtenidos con TomoTwin su propia documentación recomienda el uso de **Pandas** [22]. Esta librería agiliza el manejo de grandes estructuras de datos. Se usará para poder leer los resultados de TomoTwin, para realizar un *clustering* necesario para crear la imagen del espacio latente, para gestionar las diferentes partes del espacio latente segmentado y otras operaciones. Además, también se hará uso de **NumPy** [23] ya que permite manipular de forma eficiente muchos datos y tiene buena sinergia con la creación de imágenes.

3.3.6. Análisis topológico: `DisPerSE`

Tal y como se especificó en la Sección 1.3, la herramienta topológica que se va usar es la teoría de Morse discreta. Para ello, se hace uso de **DisPerSE** [4] que es un software que implementa las técnicas de teoría de Morse para el análisis de la estructura en el cosmos. Sin embargo, en este proyecto se utilizará para procesar el espacio latente de los tomogramas.

Esta herramienta permite obtener los puntos críticos (*i.e.* máximos y mínimos locales) propios de la teoría de Morse discreta y también los llamados *ascending/descending manifolds* (ver Sección 1.3). Gracias a eso, se va a poder seleccionar regiones del espacio latente para hacer la segmentación sin tener que implementarlo desde el principio.

3.3.7. Gestión de imágenes: `fitsio` y `Pillow`

Tanto para DisPerSE como para convertir el espacio latente de TomoTwin en algo visual, necesitamos herramientas para manejar imágenes. En particular, DisPerSE maneja el formato FITS y para ello usaremos la librería **`fitsio`** de Python que nos permitirá crear, leer y escribir imágenes en format FITS para poder usarlas en DisPerSE.
Por otro lado, para la visualización del espacio latente y la segmentación necesitamos imágenes en formatos habituales como Portable Network Graphics (PNG) y para ello usaremos la librería **`Pillow`** de Python.

3.3.8. Procesamiento Geométrico y Validación: `ParaView`

Para una verificación manual del proceso se ha utilizado una herramienta gráfica como es **`ParaView`** [24]. Esta permite visualizar las mallas geométricas que genera DisPerSE, los *ascending/descending manifolds* y el propio espacio latente todo simultáneamente. De esta manera, se ha podido verificar cualitativamente que se identifican correctamente las regiones más densas del espacio latente. Además, esta herramienta

dispone de numerosas opciones y filtros que permiten manipular los datos para hacerlos visualmente más atractivos y claros.

3.3.9. Manejo de Geometría Tridimensional: `VTK`

Cuando aplicamos DisPerSE a nuestro espacio latente, este genera una serie de puntos y regiones que vienen representadas mediante una malla geométrica de polígonos. En particular, vienen en el formato **`VTK`** [25] y por ende se va a hacer uso de su librería de Python [26] para la lectura, escritura y manejo de los archivos. Además, otras herramientas como ParaView hacen internamente uso de esta librería. Por lo tanto, cualquier operación realizada con ParaView puede hacerse de forma programática con esta librería.

4. Diseño y resolución del trabajo realizado

El objetivo del workflow a desarrollar es tomar como entrada un tomograma y generar un fichero en formato MRC con la segmentación del mismo. En esta sección se explicará tanto como es el diseño del programa como la propia implementación y su uso.

4.1. Diseño del workflow

Para llevar a cabo un diseño limpio y estructurado, se ha decidido desarrollar una serie de herramientas que usadas juntas lleven a cabo lo que deseamos. Por lo tanto, se ha decidido diseñar una arquitectura modular en la que diferentes "nodos de trabajo" trabajan en consonancia para crear el workflow deseado. A continuación, se va a explicar la funcionalidad de cada una de las etapas y nodos.

- **Análisis del tomograma**: esta es la etapa inicial y se encarga de hacer el procesamiento del tomograma mediante ML. En particular, los se usará TomoTwin (ver Sección 3.3.3). Como se puede ver en la Figura 4.1, esta etapa tiene 2 nodos:
  - **Generación del espacio latente**: este nodo será el encargado de tomar el tomograma en formato MRC y, usando TomoTwin, generar el espacio latente. La idea es utilizar el modelo ya entrenado que nos provee TomoTwin [3] y aplicarlo al tomograma. Este generará un fichero en formato `.temb` que consiste en un dataframe de Pandas con los diferentes voxels y sus valores del espacio latente.
  - **Reducción dimensional**: este nodo será el encargado de reducir la dimensión del espacio latente. Tal y como se ha comentado anteriormente, TomoTwin genera un espacio de 32 dimensiones. Una dimensionalidad tan alta complica mucho el análisis y muchas veces no aporta mucha más información. Por eso mismo, se le aplica UMAP[19] para reducir la dimensión a 2. Esta es una dimensión perfecta para crear una imagen del espacio latente. La elección de UMAP frente a otras técnicas de reducción dimensional (*e.g.* PCA) es por recomendación del propio TomoTwin [3] y por el hecho de que UMAP tiende a preservar más la estructura geométrica de los puntos. Esa peculiaridad es lo que lo hace perfecto también para este caso. Una vez que se utilizar UMAP, este genera un archivo `.tumap` que contiene un dataframe

de Pandas similar al `.temb` antes mencionado solo que ahora con los valores del espacio latente 2D.

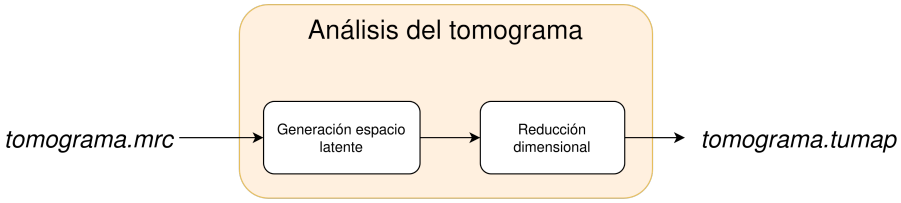


Figura 4.1: Etapa de Análisis del tomograma con sus respectivos nodos y sus valores de entrada y salida.

- **Pre-procesado**: esta etapa tiene como objetivo procesar el espacio latente generado por TomoTwin para que pueda ser utilizado por DisPerSE y posteriormente en la reconstrucción de la segmentación. Esta etapa consta de 3 nodos diferentes que se pueden apreciar en la Figura 4.2 y que se dos de ellas se ejecutan de forma paralela pues no dependen entre sí.
  - **Generación de imagen 2D**: el formato que TomoTwin devuelve es el dataframe de Pandas dentro del fichero `.tumap`. Sin embargo, DisPerSE necesita otros formatos específicos para trabajar. Una de esos formatos son las imágenes `.fits`. Esta imagen debe ser de gran resolución para que encierre lo mejor posible los límites del espacio latente. Por otro lado, como las imágenes `.fits` son tediosas de visualizar, este nodo también generará una imagen `.png` para visualizarlo cómodamente.
  - **Generación de máscara**: para que DisPerSE pueda hacer un uso adecuado y eficiente de una imagen necesita una máscara [4]. Esta máscara le indica a DisPerSE qué zonas de la imagen tiene que procesar y qué zonas no son más que fondo. Por lo tanto, este nodo se encargará de tomar el espacio latente y generar una máscara que lo envuelva. Como DisPerSE se ha elegido trabajar en formato *FITS*, este nodo generará una imagen en formato `.fits`. Además, al igual que el nodo anterior, este también generará la imagen en formato `.png`.
  - **Generación de metadatos**: con el objetivo de poder recuperar los puntos del espacio latente a partir de la imagen 2D generada, se debe guardar qué correlación hay entre los puntos y los píxeles de la imagen. Este nodo guardará un dataframe de Pandas con la correlación pixel-punto en un fichero `.mdata` para una etapa posterior.

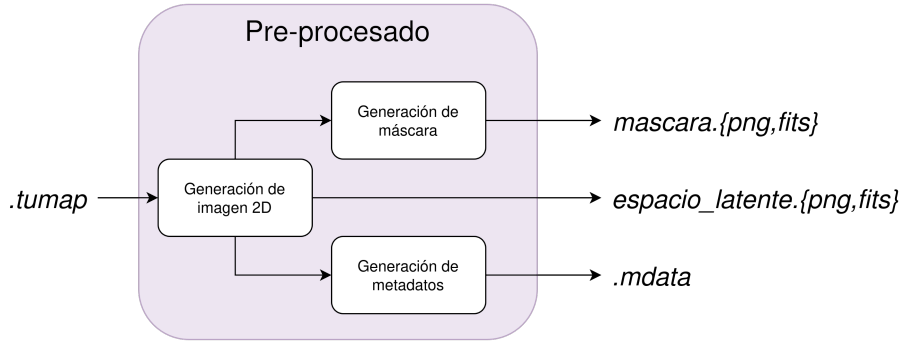


Figura 4.2: Etapa de Pre-Procesado con sus respectivos nodos y sus valores de entrada y salida.

- **Análisis topológico**: esta etapa tiene como objetivo hacer el análisis del espacio latente con técnicas de topología computacional tal y como se comenta en la Sección 1.3. En particular, se usará DisPerSE como se ha comentado en la Sección 3.3.6. Tal y como se puede ver en la Figura 4.3 esta etapa consta de 2 nodos principales:
  - **Simplificación de Morse**: este nodo es el que realizará todo el análisis topológico. Su cometido es tomar la imagen del espacio latente y la máscara para procesarla usando teoría de Morse discreta. Para ello utilizará la herramienta DisPerSE para aplicar el algoritmo de simplificación de puntos críticos (ver Sección 1.3).
  - **Exportaciones de puntos críticos y *manifolds***: este nodo realizará una tarea sencilla y es la de convertir la malla de puntos críticos y *manifolds* generados por DisPerSE en un formato que podamos manejar. DisPerSE tiene diversos formatos de salida para las mallas geométricas, pero el más interesante y sencillo de manejar es el formato `VTK`. Además, la propia suite de herramientas de DisPerSE nos proporciona una forma de convertir los puntos críticos y los *manifolds* a un `.vtu` que sería el formato `VTK` para mallas. También se exportarán los puntos críticos en el formato `.vtp`.

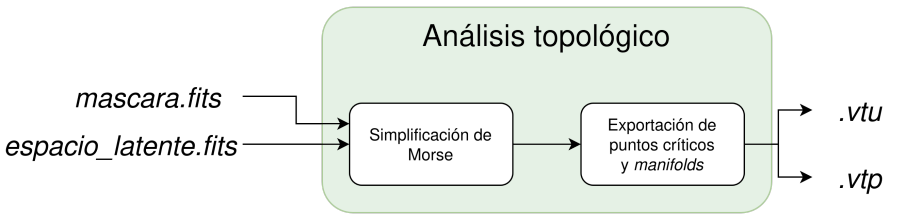


Figura 4.3: Etapa de Análisis topológico con sus respectivos nodos y sus valores de entrada y salida.

- **Post-procesado**: esta etapa tiene como objetivo procesar los resultados del análisis topológico para poder extraer los puntos críticos más relevantes, sus *manifolds* asociados y trasladar toda esa información de vuelta al tomograma. Tal y como se puede ver en la Figura 4.4 esta etapa tiene numerosos nodos y algunos de ellos se pueden ejecutar paralelamente:
  - **Separación de los diferentes *manifolds***: el fichero `.vtu` que genera DisPerSE contiene todos los *manifolds* juntos. Para poder manejarlos de mejor manera y luego poder seleccionar los que nos interesan, este nodo se encargará de separar los diferentes *manifolds* en ficheros. Generará un fichero con la extensión `.clust` por cada *manifold* que contendrá un dataframe de Pandas con todos los puntos de ese *manifold*. Además, estos ficheros luego servirán para reconstruir la segmentación.
  - **Filtrado de los puntos críticos relevantes**: el fichero de salida de DisPerSE contiene todo tipo de puntos críticos: máximo, mínimos y puntos de silla. Sin embargo, para el objetivo propuesto solo resultan interesantes los máximos pues son los lugares de mayor densidad. Además, no todos los puntos críticos tienen la misma importancia. Por lo tanto, este nodo se encargará de filtrarlos para quedarnos con los que son realmente importantes y los guarda en un fichero `.slt` que contiene un dataframe de Pandas con esos puntos críticos seleccionados.
  - **Clasificación de los *manifolds* asociados a los puntos críticos relevantes**: con la salida de los dos nodos anteriores ya se pueden filtrar los *manifolds* de aquellos puntos críticos seleccionados. Este nodo se encargará de la tarea de filtrar los ficheros `.clust` de los puntos críticos seleccionados.
  - **Transformación de los *manifolds* al espacio latente**: una vez los *manifolds* han sido seleccionados, es necesaria una forma de llevarlos al espacio latente. Como DisPerSE trabaja con una imagen 2D del espacio latente, los *manifolds* que devuelve son relativos a la imagen 2D. Sin embargo, para llevar a cabo la segmentación en el tomograma es preciso saber qué puntos del espacio latente pertenecen a cada uno de los *manifolds*. Este trabajo es el que llevará a cabo este nodo y que realizará usando el fichero de metadatos (`.mdata`) generado en la etapa de pre-procesado.
  - **Creación del archivo MRC con la segmentación del tomograma**: una vez se tienen los *manifolds* dentro del espacio latente ya se ha realizado realizada la segmentación del tomograma. Ahora solo hay que recuperar los voxels asociados a cada punto del espacio latente, pero esa información ya está codificada en el fichero `.temb` que genera TomoTwin. Por lo tanto, este nodo se encargará de crear un imagen MRC con la segmentación realizada con el mismo tamaño del tomograma.

– **Creación de una imagen PNG con los *manifolds* sobre el espacio latente**: como puede resultar interesante visualizar cómo se ven los *manifolds* dentro del espacio latente, este nodo se encargará de generar una imagen que muestre las regiones de los *manifolds* seleccionados dentro del espacio latente en formato `.png`.

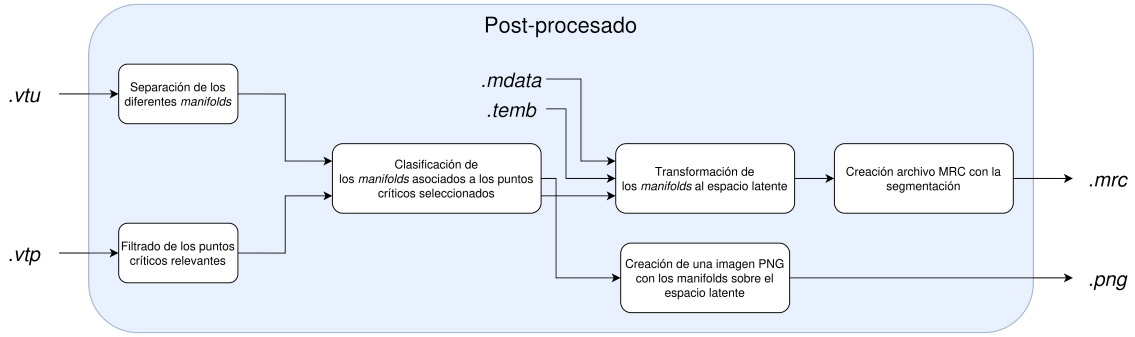


Figura 4.4: Etapa de Post-Procesado con sus respectivos nodos y sus valores de entrada y salida.

- **Visualizado**: esta etapa se compone de un un solo nodo de trabajo y es completamente opcional. Tiene como objetivo tomar la segmentación en formato MRC producida al final de la etapa de anterior y visualizarla sobre el tomograma. Para ello, hará uso de Napari [20] donde se visualizará el tomograma y cada segmentación del mismo estará representado por un color distintivo. Este nodo no produce ningún fichero nuevo, pues solo abre la interfaz gráfica de Napari.

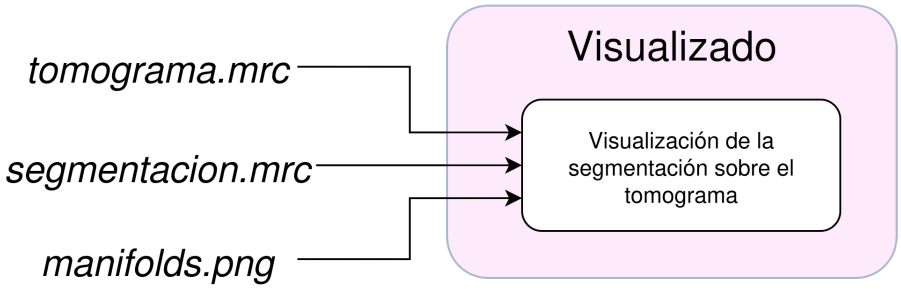


Figura 4.5: Etapa de Visualizado con su único nodo y sus valores de entrada.

En la Figura 4.6 y en el Anexo A se muestran las diferentes etapas del workflow explicadas anteriormente. Estas están conectadas entre sí para generar una herramienta unificada. De esta manera, cada nodo puede ser modificado internamente e incluso algunos pueden ejecutarse en diferentes ordenadores sin mucha complejidad.

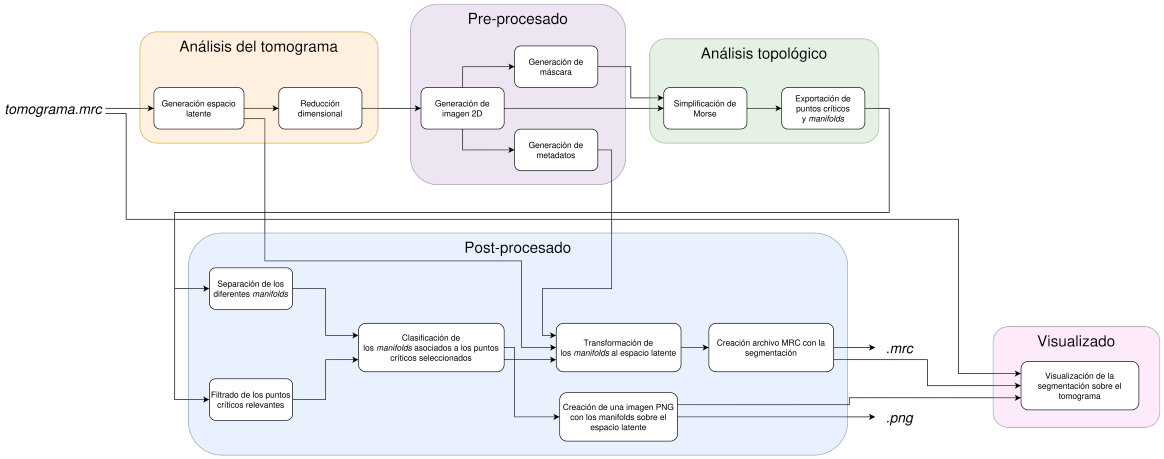


Figura 4.6: Workflow completo con todas las etapas de la herramientas interconectadas.

4.2. Implementación del workflow

En la sección anterior se ha detallado cómo es el diseño de la herramienta, pero no se han abordado cuestiones propias de la implementación. En esta sección se van a recorrer todos los nodos de las etapas para comentar como ha sido su implementación y cómo hacer uso de ellas. Además, como las herramientas siguen un patrón modular, se pueden omitir ciertos pasos si se desea no tener que ejecutar algunas etapas por ser computacionalmente muy intensivas. Todo el código se encuentra disponible en el siguiente repositorio.

4.2.1. Análisis del tomograma

Esta etapa es la encargada de obtener el espacio latente y por ende es la más intensiva computacionalmente hablando. Tal y como muestra la Figura 4.1, esta etapa tiene dos partes que se ejecutan ambas con TomoTwin.

Generación del espacio latente. Esta parte es la que más costo computacional conlleva ya que usa la red neuronal y el modelo que se nos proporciona desde TomoTwin para hacer el análisis. Es importante resaltar que, por diseño, TomoTwin requiere una GPU de la marca Nvidia. De lo contrario, no se podrá ejecutar este modelo. Además, la herramienta está diseñada para usar todas las GPUs disponibles en el dispositivo y ejecutarse más rápido. El comando que habría que ejecutar para procesar el tomograma sería:

```
tomotwin_embed.py tomogram -m <modelo> -v <tomograma> -b 256 -o
                  <ruta_salida> -s 1
```

Existen diferentes parámetros y aspectos relevantes que cabe destacar:

- `<modelo>`: aquí hay que indicar la ruta del fichero con el modelo de TomoTwin que puede descargarse desde [17]. En caso de que se haya entrenado uno propio también tendría que ir aquí.

- `<tomograma>`: aquí se indica la ruta del fichero MRC del tomograma que se quiere analizar.

- `-s 1`: este parámetro es bastante importante. El `-s` corresponde al parámetro `stride` que gestiona cada cuantos voxels debe ejecutarse la red neuronal. Los propios autores utilizan `-s 2` [17], pero esto implicaría que la red estaría saltando un voxels cada 2. Por lo tanto, la segmentación estaría obviando la mitad de los voxels del tomograma y quedaría incompleta. Cuando se intenta usar TomoTwin para la detección de proteínas, omitir esos voxels no es tan relevante, pero en la segmentación resulta de suma importancia.

Este proceso genera un fichero `.temb` que contiene un dataframe de Pandas con cada uno de los voxels y su posición en el espacio latente de 32 dimensiones. De hecho, usando Pandas se podría visualizar fácilmente, pero habría que usar una versión compatible pues está guardado como un *pickle*.

Reducción dimensional. Una vez que se ha obtenido el espacio latente con el nodo anterior se debe reducir su dimensión. Para ello, se usa UMAP [19] que viene ya integrado con TomoTwin. El comando que habría que ejecutar sería:

```
tomotwin_tools.py umap --chunk_size 348486 -i <embedding> -o
                    <ruta_salida> -n 2
```

Aunque este comando es el que se ha ejecutado durante las pruebas, no tiene necesariamente que funcionar en todas las máquinas. Como advierten los autores en [17] «If you encounter an out of memory error here, you may need to reduce the fit_sample_size and/or chunk_size values (default 400,000)». Por ende, los valores de `chunk_size` y `fit_sample_size` aquí indicados son los adecuados para el servidor de pruebas (ver Sección 3.3.1). Los otros parámetros relevantes son los siguientes:

- `<embedding>`: este sería el fichero `.temb` generado por la red neuronal de TomoTwin.

- `-n 2`: este parámetro permite indicar a qué dimensión se quiere reducir el espacio latente. Tanto 2 como 3 sería aceptable. Sin embargo, para este propósito se debe indicar 2 ya que servirá para generar luego una imagen 2D.

Un persona crítica que haya visto todo el diseño en la Sección 4.1 y tenga conocimientos de teoría de Morse discreta podría preguntarse por qué utilizar 2 dimensiones cuando las herramientas y la teoría matemática también se sostienen en 3 dimensiones. La motivación es que, aunque puede proporcionar más información tener más dimensiones,

la herramienta de análisis topológico que se utiliza (DisPerSE [4]) no funciona de forma adecuada en estas dimensiones. Aparte de eso, para conseguir una gran resolución en la imagen 3D necesitaríamos un enorme tamaño de la imagen y eso no es viable computacionalmente para DisPerSE.

4.2.2. Pre-procesado

En esta etapa se va a procesar el espacio latente para obtener tanto la máscara como la imagen 2D y los metadatos ya antes mencionados. Aunque se podrían implementar todos los nodos de la Figura 4.2 en programas diferentes, todo este proceso se va a llevar a cabo dentro de un mismo *script* de Python llamado `cluster_points.py`.

Cargar el fichero del espacio latente. Lo primero es cargar el dataframe del fichero `.tumap` y luego crear un array de NumPy cuyas dimensiones deben coincidir con las de la imagen. Este array se utilizará como representación asignando un pixel a cada posición del array. De esta manera, a mayor resolución de la imagen, mayor será el tamaño del array.

Generar la *bounding box*. Con la imagen cargada se debe obtener su *bounding box*, es decir, hay que conseguir encerrar la nube de puntos del espacio latente dentro de un rectángulo de la forma más ajustada posible. El cálculo de la *bounding box* puede hacerse de tres maneras:

- ***Bounding box* automática**: gracias a la librería NumPy se puede realizar una estimación de cada uno de los ejes y crear así una *bounding box* que encierre los puntos. Este método, aunque automático, tiende a no ajustar lo suficiente la nube de puntos y eso implica que la resolución de la imagen debe ser mayor para obtener una buena imagen del espacio latente.
- ***Bounding box* manual**: este método consiste en indicarle las dimensiones de la *bounding box*. Aunque este método suele ser más preciso, requiere visualizar previamente previamente el espacio latente para hacerse una idea de las dimensiones adecuadas para la *bounding box*.
- ***Bounding box* iterativa**: este método consiste en preguntar al usuario qué *bounding box* quiere utilizar, generar la imagen del espacio latente y mostrársela. Si la imagen es adecuada, se prosigue. En caso contrario, se vuelve a repetir el proceso. De esta forma, se puede ajustar la *bounding box* de una forma muy cómoda y sin tener que detener el nodo. Como la primera vez no hay una *bounding box* anterior, esta se obtendrá de forma automática como en el primer caso para ir refinándola en cada iteración.

Hacer la conversión del espacio latente al array. Una vez obtenida la *bounding box*, es posible aplicar el algoritmo de conversión desde el fichero `.tumap` al array de

NumPy. La idea del algoritmo es sencilla:

1. Tomar la *bounding box* y dividirla en partes de forma que forme una malla del mismo tamaño que la imagen 2D.

2. Por cada parte del espacio latente contar cuantos puntos hay dentro y asignar ese valor a la misma posición del array de Numpy.

Aquí hay un detalle que resulta interesante resaltar. En las imágenes en formato PNG, el origen de coordenadas se encuentra en la esquina superior izquierda. No obstante, tanto el espacio latente como las imágenes en formato FITS lo colocan en la esquina inferior izquierda. Por ende, aparece una incongruencia entre sistemas de coordenadas que puede ocasionar numerosos problemas. Por ese motivo, hay tener en cuenta, en cualquier momento que se manipulen coordenadas, que hay que hacer conversiones primero. La función de NumPy `flipud` será de gran utilidad pues realiza la conversión directamente entre los sistemas de coordenadas.

Al margen de esta cuestión técnica, también hay generar otro dataframe de Pandas que contenga la correspondencia entre los puntos del espacio latente y los píxeles de la imagen. Para ello, se creará un dataframe con todos puntos del espacio latente y un nuevo atributo que contenga el identificador de píxel donde se encuentra ese punto. La forma de calcular el identificador de un pixel (x, y) sería:

$$\text{id} = y \times resolucion\_PNG\_eje\_X + x$$

Aplicar un filtro Gaussiano. Para poder mejorar la precisión del análisis y evitar bordes demasiado bruscos se ha decidido aplicar un filtro Gaussiano mediante la librería `SciPy`. Gracias a esto, tanto la máscara como la imagen 2D serán más adecuadas para su posterior análisis.

Generar la máscara. Una vez se tiene el array de NumPy con los valores de la imagen, se puede generar la imagen de una forma muy sencilla creando un nuevo array de las mismas dimensiones que contenga

- Un 1 si en la misma posición del array de la imagen hay un valor más pequeño que un cierto umbral (*threshold*) que le pasamos por parámetro.

- Un 0 sino

De esta forma, se puede crear una imagen que será completamente negra en la zona donde se encuentre el espacio latente y completamente blanca el resto. La elección del umbral es realmente importante pues determinará qué regiones del espacio latente se van a procesar posteriormente. Por eso mismo, es de gran importancia hacer una buena elección. Para este cometido tenemos dos opciones:

- ***Threshold* manual**: este método consiste en indicar directamente como parámetro qué valor utilizar.
- ***Threshold* iterativo**: este método es muy similar al explicado anteriormente con la *bounding box*. Se va a solicitar que se indique el *threshold* deseado, mostrar la máscara y preguntar por la conformidad. En caso de ser negativa, se itera hasta que sea positiva.

Guardar las imágenes y los metadatos. Finalmente, solo hay que convertir los arrays de la imagen y de la máscara tanto a `.fits` como a `.png`. El primero para que DisPerSE lo pueda manejar adecuadamente y el segundo para una visualización cómoda por parte del usuario. Siempre se guardará la máscara con el sufijo `_mask`. Además, también se debe almacenar el dataframe de metadatos como un *pickle* de Pandas en un fichero `.mdata` con el sufijo `_metadata`.

4.2.3. Análisis topológico

En esta etapa se va a proceder con el análisis mediante DisPerSE [4] como herramienta de topología computacional. Como se puede ver en la Figura 4.3, aquí solo hay dos pasos que ejecutar.

Simplificación de Morse. Para llevar a cabo el análisis y la simplificación de puntos críticos de la teoría de Morse discreta (ver Sección 3.3.6) se va usar el programa `mse` que viene dentro de las herramienta que proporciona DisPerSE[4]. Este programa se va a encargar de:

- Construir el complejo de Morse asociado a la imagen y la máscara proporcionada.
- Detectar los puntos críticos del complejo de Morse generado tomando como valores los de la imagen (*i.e.* a mayor valor un píxel, mayor el valor de la función).
- Generar los *manifolds* de cada punto crítico que hayamos solicitado como parámetro.
- Seleccionar los pares a simplificar con el algoritmo de simplificación de puntos críticos mediante usando persistencia.
- Aplicar el algoritmo de simplificación de puntos críticos sobre los pares seleccionados y fusionar los *manifolds*.
- Escribir el resultado en un formato propio `.NDnet`.

Como se puede observar, `mse` ejecuta todo el proceso de análisis topológico de forma autónoma. Sin embargo, hay algunos parámetros que hay que tener en cuenta a la hora de ejecutarlo. Seguidamente se muestra cómo es el comando a ejecutar y qué opciones

hay que estudiar.

```
mse <tomograma> -mask <mascara> -cut <umbral_persistencia>
        -vertexAsMinima -dumpManifolds J2d -upSkl
```

Veamos que aquí hay varias cosas a tener en cuenta y ciertas decisiones:

- `-cut <umbral_persistencia>`: este parámetro permite indicar cuál es el umbral de persistencia que vamos a usar. Como se explica en la Sección 1.3, si dos valores críticos difieren en un valor inferior al umbral de persistencia indicado, serán emparejados. Si esto sucede el algoritmo de simplificación de puntos críticos podrá eliminarlos. Esto permite eliminar muchos puntos originados por el ruido. Decidir qué valor hay que indicar aquí no es sencillo y suele ser un proceso más de prueba y error.

- `-upSkl`: este parámetro le indica a `mse` que tiene que generar el fichero de vértices y sus caminos asociados. Solo resultarán de interés para la herramienta interesados los vértices que utilizaremos luego para la segmentación, por lo que más tarde eliminaremos los caminos.

- `-vertexAsMinima`: DisPerSE por defecto siempre considera que los vértices son los máximos. Esto lo ideal a la hora de encontrar los máximos. Sin embargo, cuando se trata de encontrar los *manifolds* de estos, resulta más complicado si los tenemos como vértices. Por eso mismo, conviene utilizar los mínimos como vértices a la hora de hacer cálculos de los *manifolds* y para eso sirve este parámetro. Lo que es muy importante destacar es que, aunque se indique esta opción, **los vertices del complejo seguirán siendo los mismos** y se podrán seguir extrayendo los máximos con facilidad.

- `-dumpManifolds J2d`: esto le indica a `mse` que debe generar los *descending manifolds* de los puntos críticos de índice 2. Normalmente, estos serían los mínimos, pero gracias al `-vertexAsMinima` ahora serán los máximos. Además, por como funcionan los *descending manifolds* en DisPerSE, estos se corresponderán a la "zonas de influencia" de los máximos (ver Sección 1.3). Los detalles rigurosos y cercanos a DisPerSE son algo complejos y se salen del objetivo del documento, pero pueden consultarse en [4].

Con todo esto, se obtienen ficheros `.NDskl` y `.NDnet` que contendrán tanto los vértices como los *descending manifolds*, respectivamente.

Exportación de los puntos críticos y los *manifolds* a VTU. Ahora convendría convertir los ficheros `.NDskl` y `.NDnet` en algo más manejable. En concreto, se va a utilizar el formato `VTK` que tiene tanto representación de puntos `.vtp` como de mallas geométricas `.vtu`. Para ello, solo es necesario hacer uso de las herramientas que DisPerSE provee: `skelconv` y `netconv`. Estas se utilizan con los siguiente comandos:

```
skelconv <up.NDskl> -to vtp
netconv <manifolds.NDnet> -to vtu
```

Así, ya se tendrían los ficheros en los formatos adecuados para el procesamiento final y para poder visualizarse con herramientas como ParaView.

4.2.4. Post-procesado

Esta es la última etapa obligatoria y consiste en procesar el resultado del análisis topológico para que podamos crear la segmentación y visualizarla. Tal y como se puede ver en la Figura 4.4, aquí hay numerosos pasos que realizar.

Separación de los diferentes *manifolds*. Cuando DisPerSE genera el fichero de los *manifolds* siempre los genera todos juntos con un identificador que permite saber a qué punto crítico le corresponde cada *manifold* [4]. Este valor es siempre el `source_index` y permite posteriormente asociar cada punto crítico con su *manifold*. Por lo tanto, la estructura que sigue el fichero `.vtu` que hemos generado en la etapa anterior es que a cada celda que conforma un *manifold* le pone como atributo `source_index` el índice del punto crítico asociado. Para procesarlos se ha utilizado un *script* de Python llamado `obtain_classes.py` y lo que hace es lo siguiente:

1. Recorrer todas las celdas del fichero `.vtu`.
2. De cada celda, obtener cual es la clase (`source_index`) a la que pertenece.
3. Recorremos todos los vértices que forman parte de la celda seleccionada y los añadimos a una lista de puntos asociados a la clase concreta de esa celda.

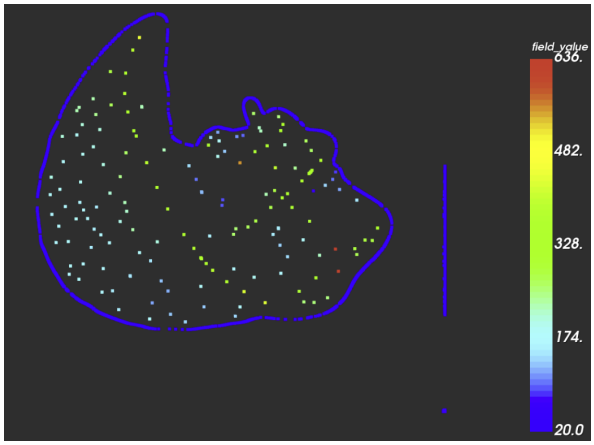
De esta manera, al terminar el proceso tendremos un mapa que asociará a cada clase (`source_index`) su lista de puntos. Ahora se puede guardar cada una de estas clases como un dataframe de Pandas y volcarlo a un fichero. Se utilizará la extensión `.clust` para identificar cada *manifold* y llevarán siempre como sufijo en el nombre del fichero `_<source_index del manifold>`. Gracias a esta nomenclatura será extremadamente sencillo identificarlos luego a la hora de filtrarlos.

Filtrado de los puntos críticos relevantes. El fichero `.vtp` que se genera con DisPerSE al añadir la opción `-upSkl` incluye tanto los puntos críticos como los filamentos que los unen. Además, los puntos críticos incluyen tanto máximos como mínimos y puntos de silla. Como solo resultan de interés los máximos, se debe llevar a cabo un filtrado de todos esos puntos. Este proceso se puede realizar muy cómodamente en `ParaView`. Sin embargo, esto precisaría de una intervención manual y luego de una exportación a un fichero. Para evitar esto, se va a utilizar la librería `vtk` de Python para reproducir los mismos filtros de `ParaView` dentro de un *script* llamado `filter_maxpoints.py`. Los filtros a aplicar son los siguientes:

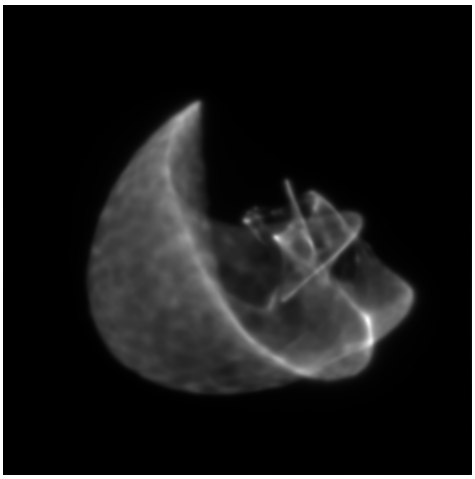
1. **Filtro de puntos**: los `.vtp` generados incluyen vértices y filamentos, pero solo se necesitan los vértices. Para distinguirlos, DisPerSE indica que se puede usar el atributo `type`. Este atributo valdrá 1 si es un filamento y -1 si es un vértice. Por lo tanto, sería suficiente con aplicar un umbralizado al atributo `type` que seleccione aquellos que valgan -1.

2. **Filtro de máximos**: una vez elegidos los vértices, se necesitan seleccionar aquellos que sean máximos. En DisPerSE se indica que la forma de identificar cada tipo de punto crítico es mediante el `critical_index`. Normalmente, los máximos serían `critical_index = 0`, pero como se ha utilizado el parámetro `-vertexAsMinima` ahora los máximos son `critical_index = 2`. Asimismo, solo hay que hacer un umbralizado que seleccione los `critical_index = 2`.

3. **Filtro de valor**: este filtro es opcional y lo que permite es seleccionar aquellos máximos cuyo valor de densidad se encuentre dentro de unos márgenes. Este filtro permite refinar mucho más la selección de qué máximos utilizar para la segmentación. Para hacer el filtro se usará el atributo `field_value` para aplicar un umbralizado y seleccionar los valores en un rango concreto. Además, se ha añadido una opción `--interactive` que permite elegir el rango para `field_value` y visualizarlo antes de guardar los puntos. De esta manera, se puede ajustar finamente el rango que utilizar. En la Figura 4.7 se muestra cómo se vería el proceso interactivo filtro aplicado sobre un espacio latente.



(a) Imagen generada para visualizar los puntos críticos y realizar el filtrado



(b) Espacio latente de referencia

Figura 4.7: Resultado de aplicar el modo `interactive` sobre un espacio latente para visualizar sus puntos críticos **(a)** y su espacio espacio latente de referencia **(b)**.

Una vez aplicados todos estos filtros, los puntos resultantes se guardarán junto con sus atributos en un dataframe de Pandas. Luego, este dataframe será guardado en un fichero con extensión `.slt`.

Clasificación de los *manifolds* asociados a los puntos críticos seleccionados. En este paso se deben utilizar tanto los *manifolds* separados como los máximos filtrados. Verdaderamente, solo resultan de interés los *manifolds* que estén asociados a aquellos máximos seleccionados previamente, por lo que el resto se descartan. Para hacer esto se precisa de los ficheros `.clust` de los *manifolds* y el fichero `.slt` con los máximos seleccionados. Gracias a la documentación de DisPerSE [4], se sabe que cada *manifold* guarda un campo llamado `source_index` que especifica cuál es el índice del punto crítico asociado guardado en el atributo `index`. La idea es muy sencilla: de cada máximo, obtener su `index` y luego seleccionar el *manifold* con el mismo `source_index` usando los nombres de ficheros. Este proceso está implementado en un *script* Python llamado `filter_manifolds.py`.

Transformación de los *manifolds* al espacio latente. Ahora que con los *manifolds* seleccionados, aparece un ligero inconveniente. Cada *manifold* tiene las coordenadas de los puntos que lo forman. Sin embargo, estas coordenadas son relativas a la imagen 2D que generada del espacio latente y no al propio espacio, pues con lo que ha trabajado DisPerSE. Por lo tanto, es necesaria una forma de convertir todos estos puntos de la imagen en sus respectivos valores del espacio latente. Aquí entra en juego el fichero de metadatos `.mdata` que generado durante la etapa de pre-procesado. Este fichero contiene una correspondencia entre los píxeles de la imagen y los puntos del espacio latente. Si además se une con el fichero `.temb` original, se pueden recuperar de forma sencilla qué partes del espacio latente pertenecen a cada *manifold*. El procedimiento sería el siguiente:

1. Cargar tanto el `.tumap` como el `.mdata` que contienen la información tanto de los puntos del espacio latente como de la correspondencia con la imagen 2D.

2. Cargar el `.clust` de un *manifold* que se quiera convertir.

3. Ahora, para cada punto de `.clust`, se obtiene usando el `.mdata` qué índices de los puntos del espacio latente le corresponden. Recordar que `.mdata` lo que guarda es una asociación entre voxels del tomograma y los píxeles de la imagen.

4. Ahora se seleccionan todas las entradas del fichero `.temb` relativas a los voxels anteriores. De esta forma, se están seleccionando solo los puntos del espacio latente que están en el *manifold*.

Ahora ya se tendrían los puntos del espacio latente seleccionados correctamente y, además, como el fichero `.temb` guarda también la correspondencia entre espacio latente y los voxels del tomograma, este fichero sirve para luego reconstruir el *manifold* dentro

del tomograma. La elección entre usar el fichero `.temb` y no el `.tumap` no es azarosa. Gracias al tutorial de los autores de TomoTwin [17], se observa que si se seleccionan los *manifolds* en el espacio latente de 32D, entonces se podrá afinar la selección de la región más tarde. Mediante UMAP, se puede calcular un nuevo espacio latente 2D a partir del *manifold* que capturará mejor las sutilezas del mismo. Así, se obtendría un refinamiento del *manifold*. Todo este procedimiento se encuentra en el *script* de Python llamado `recover_point_from_img.py`.

Creación de una imagen PNG con los *manifolds* sobre el espacio latente. Para poder visualizar cómo se ven los *manifolds* dentro del espacio latente se utiliza el *script* de Python llamado `show_manifolds.py`. Haciendo uso de la librería `Pillow` para la manipulación de imágenes y de Pandas para manejar los *manifolds*, se crea una imagen con todos los *manifolds* pintados de diferentes colores sobre el espacio latente. Además, se va generar un fichero de metadatos en JavaScript Object Notation (JSON). Este fichero contendrá una entrada por cada *manifold* de la imagen 2D y tendrá como valores: ruta del fichero `.clust` y el color asignado. De esta manera, se puede reconocer rápidamente qué `.clust` corresponde a cada *manifold* de la imagen.

Creación del archivo MRC con la segmentación. Finalmente, con la segmentación contenida dentro de los *manifolds* del espacio latente, solo hay que llevarla de vuelta al tomograma. Para ello, se usará la librería `mrcfile` que permite crear un archivo MRC con las mismas dimensiones que las del tomograma original, pero ahora con la segmentación. Para ello, a cada *manifold* se le asigna una etiqueta y se va a colocarla en todos los voxels que nos indique su fichero `.clust` generado en el paso anterior. Si se repite esto con todos los *manifolds*, se obtiene una fichero MRC con la segmentación final. Además, se podrá indicar mediante la opción `--separate` si cada *manifold* se debe guardar en un MRC diferente. En caso de ser así, se generará también un fichero de metadatos JSON donde habrá una correspondencia entre el fichero `.temb` del *manifold* y su identificador asociado que aparecerá en el nombre del MRC como `manifold_ID_segmentation.mrc`. Este proceso se encuentra en el *script* de Python llamado `create_mrc.py`.

4.2.5. Visualizado

Esta etapa, como ya se ha comentado, es completamente opcional y consiste en visualizar la segmentación producida en Napari. Para llevar esto a cabo se ha utilizado la librería de Napari que viene incluida en la instalación llamada `Napari` [20]. La idea es sencilla:

- Se carga el tomograma y se inserta dentro de Napari como una capa de imagen.
- Se recorren todos los ficheros MRC de la segmentación y se añaden como etiquetas con diferente colores a Napari.

- Se abre Napari para visualizar el resultado.

Más allá del flujo, hay ciertos aspectos que son relevantes de la implementación.

Para empezar, los ficheros MRC deben haber sido generados en la etapa anterior con la opción `--separate`. De esta manera se obtendrán tanto las regiones separadas en ficheros MRC distintos como un fichero JSON de metadatos. Este fichero es imprescindible pues contiene un identificador de la región y su fichero `.temb` asociado. Por ese mismo motivo es necesario que antes de ejecutar esta etapa se haya ejecutado la creación de los MRC separados.

Para seguir, se le debe introducir el fichero de metadatos generado en la creación de la imagen PNG. Este fichero, como ya se ha comentado, contiene la correspondencia entre cada región y su color en la imagen. Gracias a él se le podrá asignar el mismo color a la segmentación en Napari que la que tiene su región asociada en el PNG.

Tras todo esto se puede lanzar Napari y visualizar los resultados generados por la herramienta de una forma muy sencilla accesible para analizar la bondad de los resultados.

4.2.6. Workflow completo

Ahora que ya ha observado cómo se ha llevado a cabo el desarrollo de cada uno de los nodos es pertinente ver cómo se unifican todos. Como se muestra en la Figura 4.6, todas las etapas deben funcionar en sincronía para crear una herramienta automatizada. Esta se ejecuta con el programa `auto-cryoet-workflow.py`. Haciendo uso de las librería `rich` y `questionary` se ha creado una herramienta para ejecutar cómodamente cada etapa y el workflow completo.

Cuando se ejecuta el programa aparece un menú para seleccionar las distintas opciones como se puede ver en la Figura 4.8. Estas opciones son:

- Ejecutar el workflow completo.
- Ejecutar un conjunto de etapas del workflow.
- Ejecutar un conjunto de nodos del workflow.
- Listar todas las operaciones y etapas que contiene el workflow.

Figura 4.8: Menú inicial de la herramienta diseñada donde se pueden ver las diferentes opciones.

En la sección para seleccionar las etapas concretas aparece un listado con todas ellas como muestra la Figura 4.9.

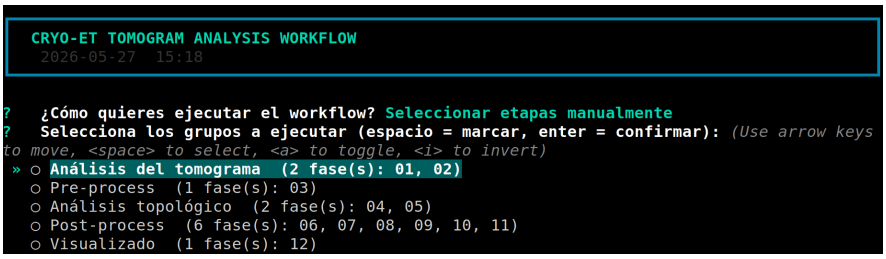


Figura 4.9: Menú de selección de las etapas a ejecutar dentro de la herramienta.

Al confirmar la selección de ese conjunto de etapas solicitará los parámetros necesarios para ejecutar el workflow como se puede ver en la Figura 4.10.

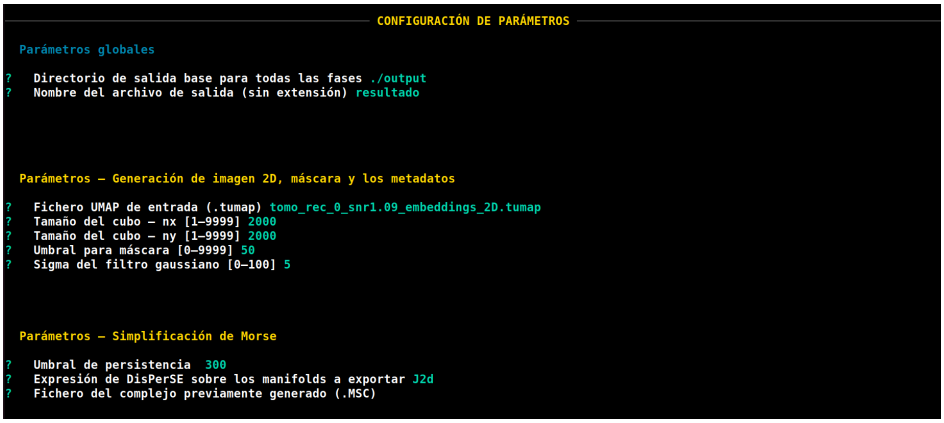


Figura 4.10: Introducción de los diferentes parámetros que necesita el workflow para ejecutar las etapas seleccionadas previamente.

Si algo sale mal, el workflow lo detectará e informará de que ha habido un error en un nodo y preguntando si queremos abortar el workflow o queremos continuarlo como se muestra en la Figura 4.11.

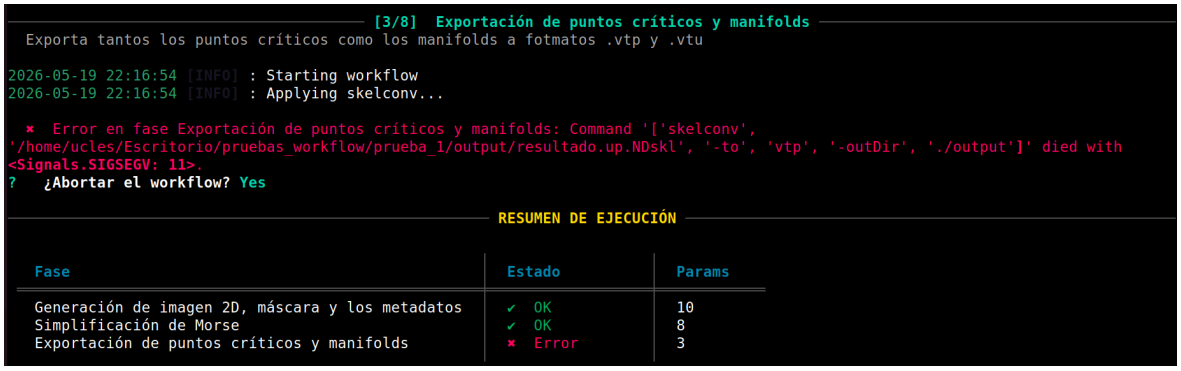


Figura 4.11: Resumen del workflow cuando algunas de las etapas intermedias ha fallado.

Finalmente, cuando todo el procedimiento haya terminado se mostrará un resumen de todas etapas seleccionadas y si su finalización ha sido exitosa o errónea como se muestra en la Figura 4.12.

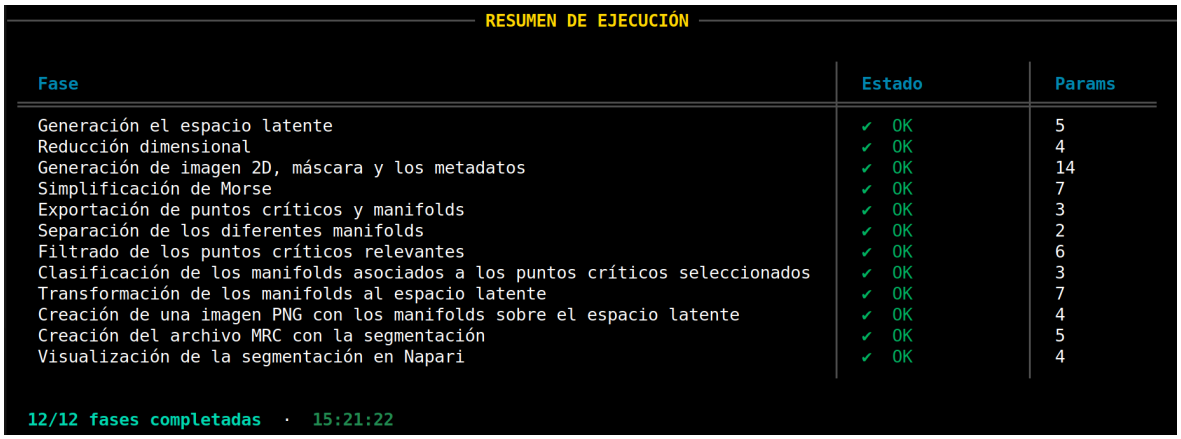


Figura 4.12: Resumen del workflow cuando todas las etapas intermedias se han ejecutado correctamente.

5. Pruebas y resultados obtenidos

5.1. Pruebas realizadas

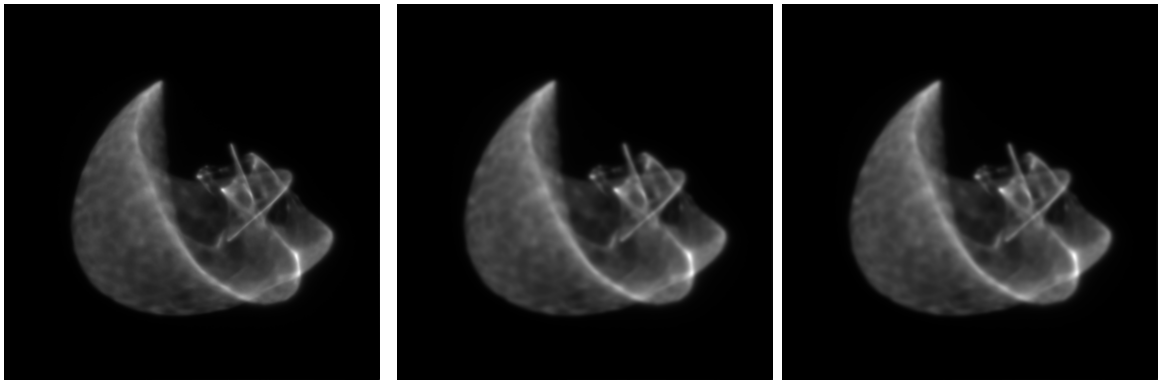
Este capítulo tiene como objetivo mostrar los diferentes experimentos que se han realizado con la herramienta diseñada y sus resultados. En particular, se va a utilizar el workflow completo que se muestra en la Figura 4.6 con el objetivo de verificar el funcionamiento de la herramienta al completo.

Como punto de partida se van a tomar dos tomogramas diferentes que, con el objetivo de diferenciarlos, tendrán los apelativos de **tomo_rec0** y **tomo_rec1**. Estos son tomogramas sintéticos que simulan el contexto celular, conteniendo membranas, macromoléculas y estructuras filamentosas, generados con la herramienta PolNet [27]. La idea es procesarlos con el workflow para generar una segmentación de los mismos. Aunque cada fase que muestre va a comentarse de forma individual, hay que mantener siempre presente que este es un flujo de trabajo cohesionado dentro de la herramienta diseñada y que se ha ejecutado de forma automática.

Análisis del tomograma. Esta fase, como ya se ha comentado, sirve para obtener el espacio latente y se ha ejecutado sin modificar los parámetros ya establecidos en la Sección 4.2.1 pues son los adecuados para el servidor de pruebas (ver Sección 3.3.1).

Pre-procesado. Tras hacer diversas pruebas, se ha optado por una resolución de 2000×2000 píxeles para buenos resultados. Pruebas con menor resolución generaban imágenes con poco detalle y mucha más resolución incrementa el tiempo del análisis sin un impacto sustancial en la segmentación.

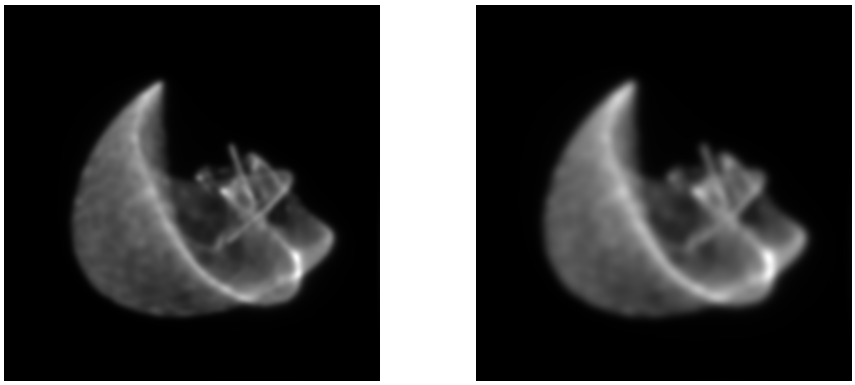
Por otro lado, se ha experimentado con diferentes valores de σ (el parámetro del filtro Gaussiano) sobre **tomo_rec0**. Como se puede ver en la Figura 5.1 el valor que proporciona un mejor aspecto a la imagen del espacio latente se encuentra entre 6 y 10, mientras que por encima de 12 ya se pierden demasiados detalles. Este filtro, como ya se había comentado, ayuda a potenciar los valores de densidad y obtener un mejor análisis posterior.



(a) Imagen sin filtro Gaussiano de referencia.

(b) Imagen con filtro Gaussiano $\sigma = 6$.

(c) Imagen con filtro Gaussiano $\sigma = 10$.

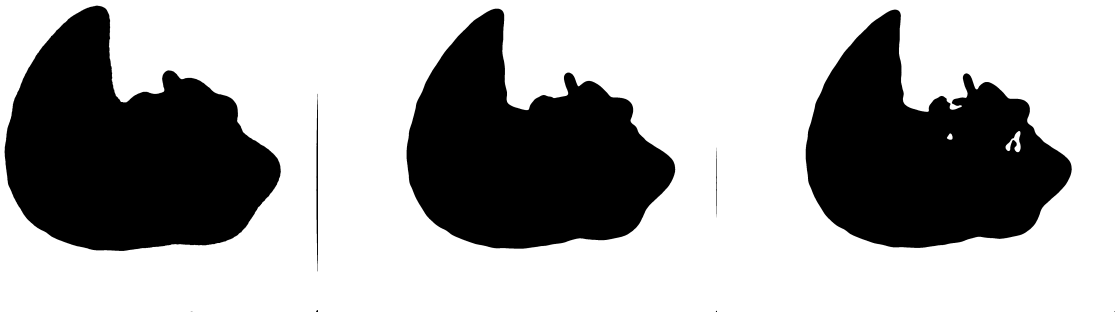


(d) Imagen con filtro Gaussiano $\sigma = 12$.

(e) Imagen con filtro Gaussiano $\sigma = 20$.

Figura 5.1: Comparativa de diferentes valores de σ para el filtro Gaussiano sobre el mismo espacio latente de tomo_rec0

Otro valor de interés en esta fase es el umbral para construir la máscara. La idea es obtener una máscara que encierre todo el espacio latente lo más ajustado posible, pero sin llegar a generar una máscara con huecos inexistentes. Para estos dos tomogramas se ha utilizado un valor de 30 pues como se puede ver en la Figura 5.2 valores mucho más grandes generan huecos en la máscara que no existen y mucho más pequeñas granulan demasiado el borde.



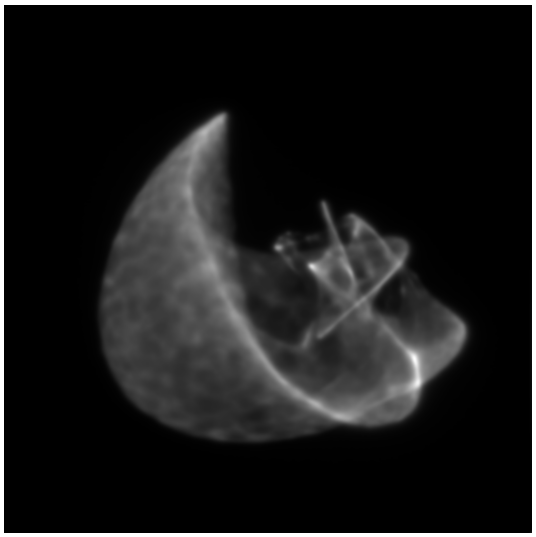
(a) Máscara con umbralizado de 10.

(b) Máscara con umbralizado de 30.

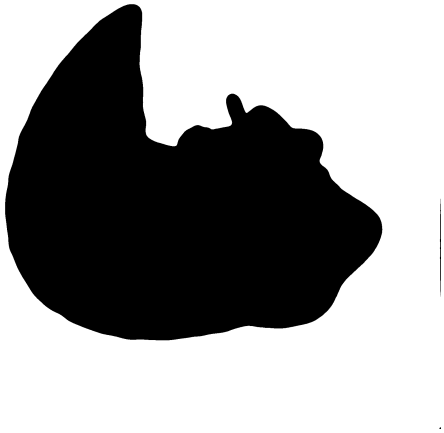
(c) Máscara con umbralizado de 50.

Figura 5.2: Comparativa de diferentes valores de umbral de la máscara sobre el mismo espacio latente de tomo_rec0.

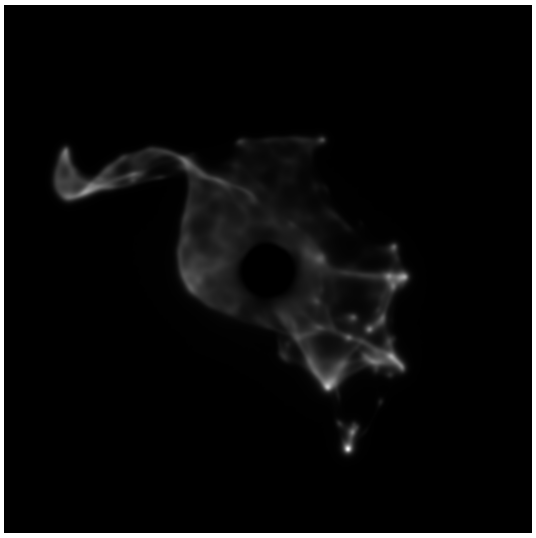
Antes de finalizar esta etapa, en la Figura 5.3 se pueden ver el espacio latente y su máscara asociadas de cada tomograma con los parámetros indicados. Tanto la Figura 5.3.a como la Figura 5.3.c son imágenes 2D del espacio latente y serían las mismas que obtenidas con el plugin de TomoTwin para Napari (ver Sección 2.2). De hecho, si obvervan detenidamente la Figura 2.2 puede notarse que el espacio latente que aparece es idéntico al de la Figura 5.3.a. Esto es producto de que el tomograma usado en ambos casos es el mismo (**tomo_rec0**). Gracias a este hecho, se comprueba que hasta este punto nuestro método es completamente análogo al manual con el plugin de TomoTwin.



(a) Espacio latente de **tomo_rec0**.



(b) Máscara de **tomo_rec0**.



(c) Espacio latente de **tomo_rec1**.

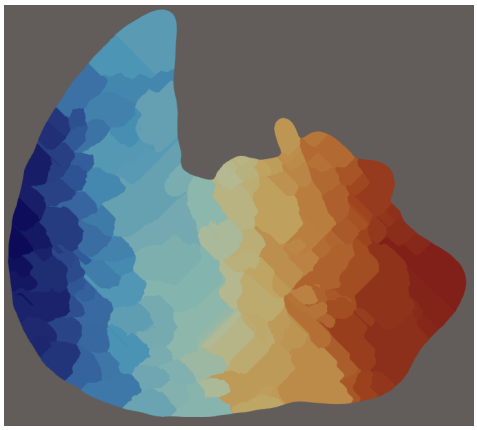
(d) Máscara de **tomo_rec1**.

Figura 5.3: Espacio latente y máscaras tanto de **tomo_rec0** como de **tomo_rec1**.

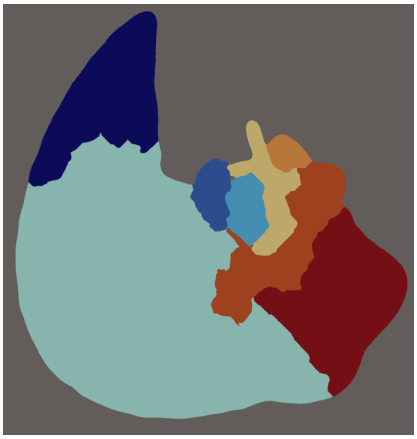
Análisis topológico. Esta etapa tiene como interés hacer el análisis topológico del espacio latente y jugar con la persistencia para eliminar el ruido y simplificar los valores críticos. De hecho, la persistencia es el único valor que ajustable para modificar cómo se hace el análisis (ver Sección 4.2.3).

En un primer momento, cabría esperar que existe una relación entre ajustar finamente el valor de la persistencia y conseguir un mejor análisis. Sin embargo, hay que recordar qué papel juega exactamente el umbral de persistencia. Cuanto mayor sea el umbral de persistencia, más emparejamientos de puntos críticos habrá y, por ende, más se simplificarán. Esto, a priori, parece una buena idea pues solo sobrevivirán los puntos críticos con un valor de densidad más alto. No obstante, hay que tener en cuenta que en este razonamiento no se están teniendo en cuenta los *descending manifolds*.

Como se explica en la Sección 1.3, los *descending manifolds* deben ser absorbidos por los puntos críticos restantes tras la cancelación, haciendo que cada vez crezcan más. El inconveniente principal reside en que una simplificación excesiva mediante la persistencia generaría *manifolds* demasiado extensos. Por esto mismo, conviene seleccionar un umbral de persistencia pequeño y luego filtrar los puntos con otras técnicas de filtrado en la fase de post-procesado. En la Figura 5.4 se muestra el mismo análisis topológico de **tomo_rec0** con valores de persistencia muy distintos. Es claro cómo en el caso de la Figura 5.4.a hay muchísimos *manifolds* muy pequeños y, sin embargo, en la Figura 5.4.b apenas hay *manifolds* y son de gran tamaño. Para los experimentos, se ha determinado que un umbral de persistencia de 5 es adecuado para ambos casos.



(a) *Manifolds* con umbral de persistencia de 5.

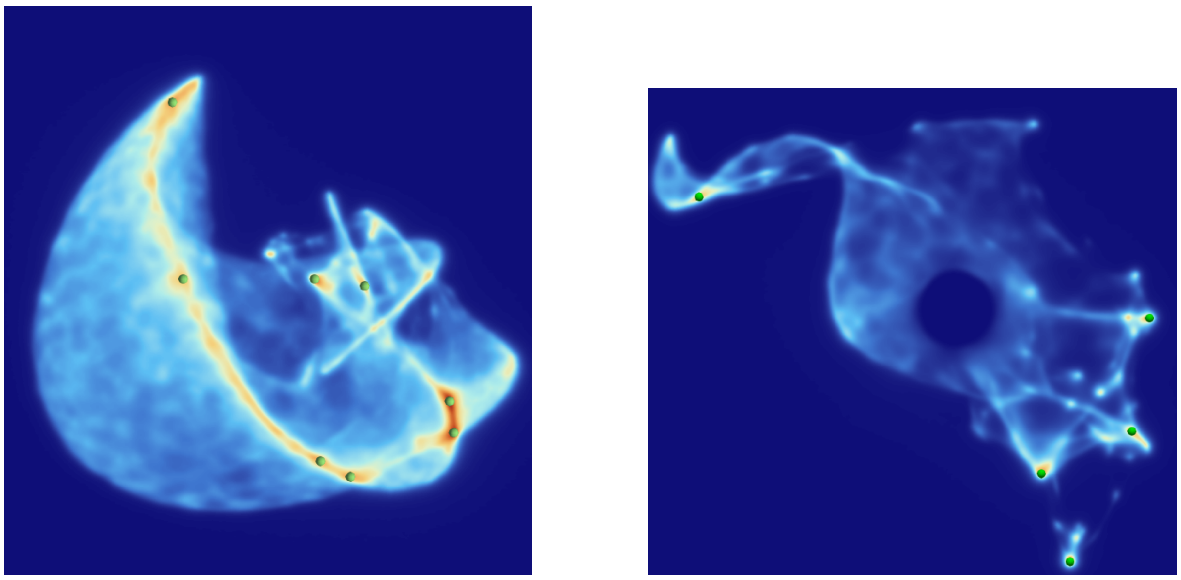


(b) *Manifolds* con umbral de persistencia de 300.

Figura 5.4: Comparativa entre diferentes umbrales de persistencia para el mismo espacio latente donde cara color representa un *manifold* diferente dentro del espacio latente de **tomo_rec0**.

Post-procesado. Aunque en esta etapa hay numerosas fases, solamente hay una en la que posee parámetros ajustables para obtener unos resultados u otros: Filtrado de puntos críticos relevantes. Al observar las otras fases se nota no son más que transformaciones y manipulaciones de los datos para hacerlos más cómodos y prácticos.

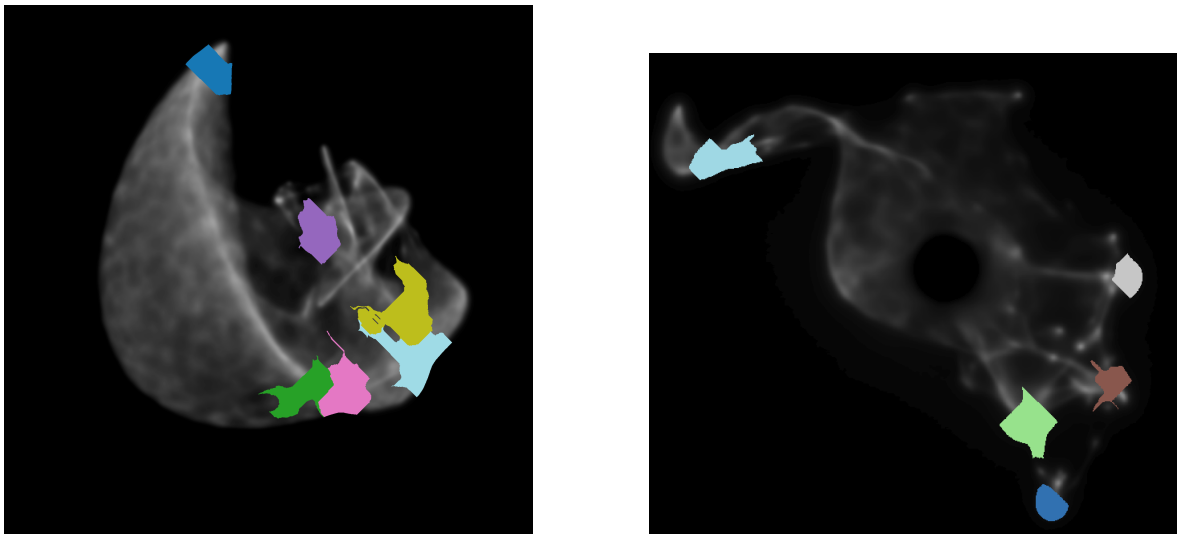
Filtrado de puntos críticos relevantes. Según lo expuesto en la etapa anterior, la persistencia no es el único mecanismo útil para filtrar los puntos críticos. Para ello, se hará uso del filtro de `field_value` que se comentó en la Sección 4.2.4 y que permitirá seleccionar aquellos puntos críticos de mayor valor. Para un ajuste fino del umbral, se ha utilizado la opción `--interactive` que incluye este nodo. De esta manera, se puede ajustar el rango del filtro sin salir del propio workflow. Cada tomograma tiene su rango y, por lo tanto, no hay unos valores adecuados para ambos, pero un indicador de que el filtro es el adecuado es fijarse si unicamente prevalecen unos pocos puntos con valor alto. En la Figura 5.5 se muestra cómo quedan los puntos finalmente elegidos en `ParaView` sobre el espacio latente.



(a) Puntos críticos del **tomo_rec0**. **(b)** Puntos críticos del **tomo_rec1**.

Figura 5.5: Resultado de los máximos filtrados (verde) sobre el espacio latente como mapa de calor tanto de **tomo_rec0** como de **tomo_rec1** en ParaView.

Creación de una imagen PNG con los *manifolds* sobre el espacio latente. Como se ha mencionado anteriormente, este nodo es completamente autónomo y no precisa de parámetros. No obstante, como este proporciona una representación visual cómoda de los *manifolds* sobre el espacio latente merece una sección. En la Figura 5.6 se aprecia cómo se verían los *manifolds* representados dentro de cada espacio latente. Si se comparan con el método de la Sección 2.2, puede verse que mientras que en el método de Napari es el usuario quien debe seleccionar las zonas de interés del espacio latente; aquí ha sido el análisis topológico el que lo ha hecho por automáticamente. Ciertamente, precisa de una cierta intervención manual para ajustar el filtro del `field_value` o el umbral de persistencia, pero eso es solamente para decidir cuán sensibles está siendo a la densidad del espacio latente y al ruido. Una vez determinados esos parámetros, las zonas del espacio latente asociadas a cada posible estructura se determinan de forma automática.



(a) *Manifolds* seleccionados del **tomo_rec0**.

(b) *Manifolds* seleccionados del **tomo_rec**.

Figura 5.6: Resultado de los *manifolds* seleccionados tanto de **tomo_rec0** como de **tomo_rec1** en ParaView.

Creación de ficheros MRC. Este proceso es automático y no precisa de configuración adicional. Sin embargo, permite ver los *manifolds* de la imagen anterior ahora en un fichero MRC para comparar el resultado autónomo con el manual. Como se quiere comparar los resultados con el método manual (ver Sección 2.2), se va a solicitar que separe cada *manifold* en un MRC diferente. De esta manera, cada *manifold* aparecerá en un fichero con un identificador diferente. Además, se generará un fichero JSON que tendrá la correspondencia entre el *manifold* y su identificador, para que, usando el otro

JSON generado por el paso anterior (ver Sección 4.2.4), se pueda seleccionar rápidamente la segmentación deseada. En la Sección 5.2 se van a mostrar estos ficheros y su comparación final con el método manual.

Refinamiento de la segmentación. En la etapa de post-procesado existe un nodo que se encarga de transformar los *manifolds* de la imagen 2D al espacio latente. Esta es completamente autónoma y no da resultados que puedan ser fácilmente visualizados. Sin embargo, estos permiten realizar un proceso muy interesante. Gracias a que devuelve los *manifolds* en el espacio latente de 32 dimensiones, podemos utilizar estos *manifolds* para refinar la selección cuanto queramos. Como se menciona en el tutorial de TomoTwin [17], se puede utilizar Napari para recalcular UMAP y refinar la selección. Con la herramienta desarrollada esto también es posible. Solo se debe seleccionar el fichero del *manifold* en el espacio latente antes mencionado y usarlo como entrada del workflow, ignorando el primer nodo encargado de generar el espacio latente. Así, se volverá a analizar el espacio latente prestando ahora más atención a aquellos detalles finos del *manifold*, como que se explica en [17].

5.2. Resultados obtenidos

En esta sección se detallan los resultados de las pruebas anteriormente descritas y además se realiza la comparación de esta herramienta con el método manual. Se van a mostrar los resultados obtenidos con la ejecución automática para los dos tomogramas. Para ello, se va a utilizar Napari para superponer el tomograma y las regiones seleccionadas. Con el objetivo de poder identificar qué se está visualizando, cada segmentación estará pintada del mismo color que aparece en su imagen del espacio latente con los *manifolds*, es decir, las imágenes de la Figura 5.6.

Por un lado, se va a visualizar el resultado de **tomo_rec1** pues se usará el otro para compararlo con el método manual. Para ello, solo hay que usar el programa `show_napari.py` diseñado para mostrar la segmentación o ejecutar su fase correspondiente en el workflow (*i.e.* Visualización de la segmentación sobre el tomograma). Al abrirse, se mostrará el tomograma y la segmentación dividida en capas con el objetivo de visualizarse cómodamente. En la Figura 5.7 se puede ver tanto en el tomograma como la segmentación por colores. Se nota como se están manejando imágenes con mucho ruido y complejas. Además, gracias a la escala, se aprecia el nivel de detalle al que se puede llegar con esta técnica. Más allá de eso, se puede observar ligeramente como el tomograma se ha segmentado correctamente y se pueden identificar diferentes estructuras. Además, gracias a la versatilidad de Napari se pueden ocultar las segmentaciones deseadas para analizar en profundidad cada una de ellas por separado.

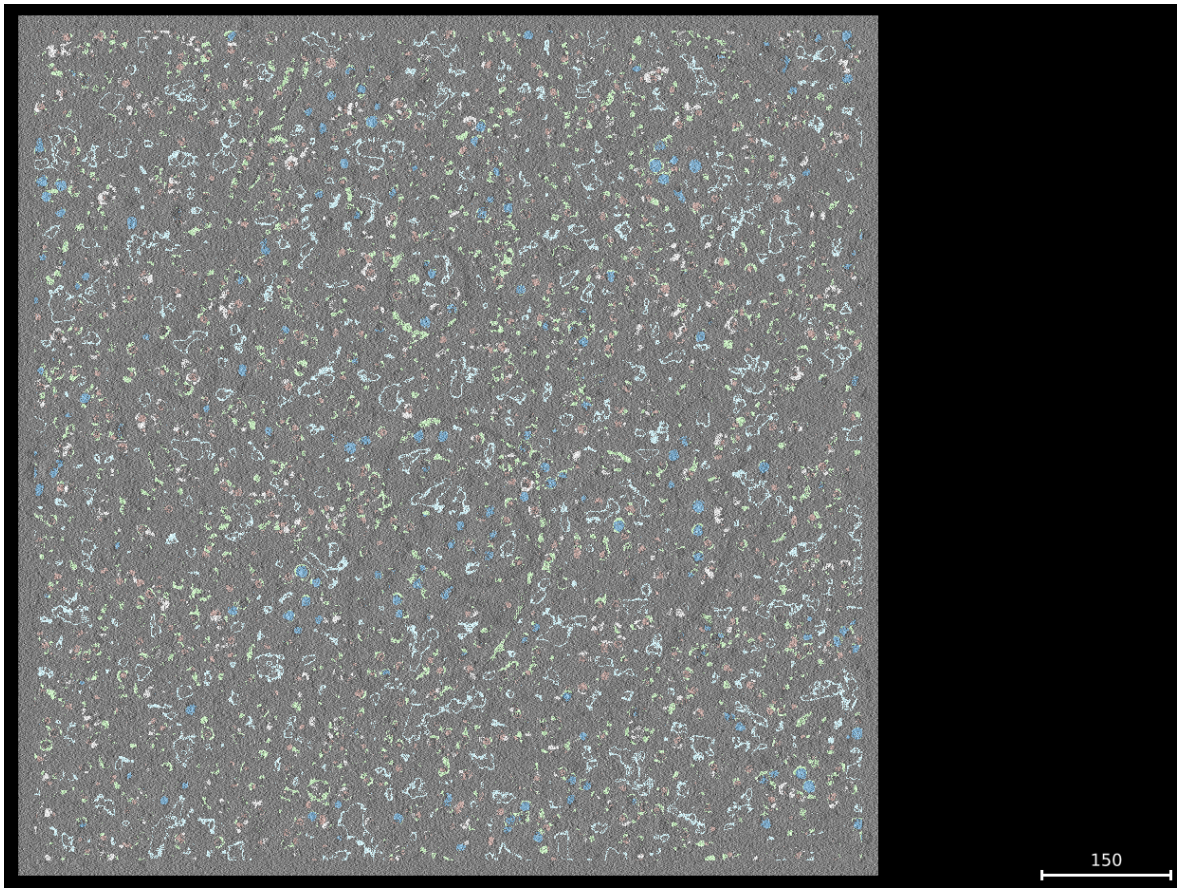


Figura 5.7: Tomograma **tomo_rec1** segmentado en diferentes colores según el tipo de estructura dentro de Napari con la escala nanmétrica de la misma en la esquina inferior derecha.

Con el objetivo de validar la segmentación realizada, en la Figura 5.8 se puede visualizar el tomograma con la segmentación azul generada por la herramienta dentro de Napari. En este caso, se ha superpuesto esta segmentación azul no sobre el tomograma original sino con la "solución". Al ser tomogramas sintéticos generados mediante un simulador (ver Sección 5) se sabe con certeza dónde se encuentran las estructuras y los diferentes tipos. Por lo tanto, se puede superponer la región sobre estos datos para observar la bondad del análisis. Si se analiza con detalle, se puede apreciar cierta especificidad con algunas estructuras, macromoléculas con un tamaño parecido y forma redondeada. No obstante, también se pueden encontrar otras zonas segmentadas que no coinciden con ninguna estructura que corresponden a los llamados falsos positivos propios de cualquier análisis no supervisado. Si se quisiera un análisis más fino o preciso, se requeriría un refinamiento de la región como se comenta en la sección anterior; un reajuste de los parámetros o incluso el entrenamiento de un modelo propio.

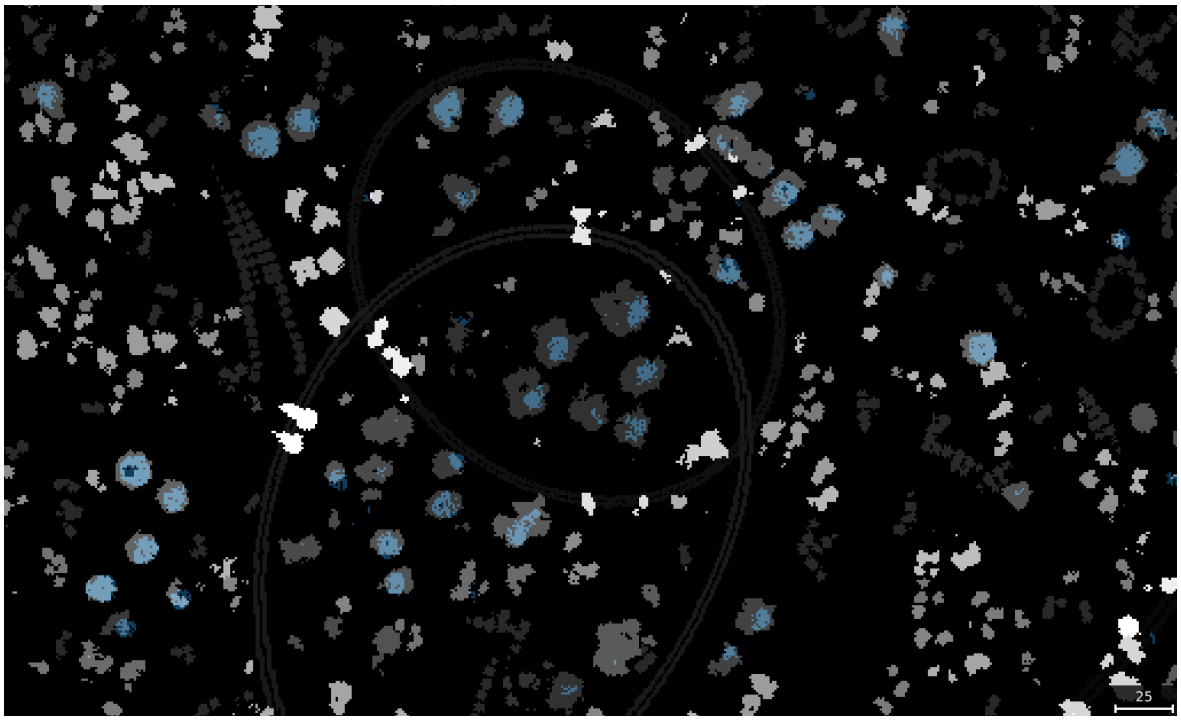


Figura 5.8: Segmentación azul del **tomo_rec1** sobre la distribución de las estructuras del tomograma dentro de Napari con su escala nanométrica en la esquina inferior derecha.

Por otro lado, se va a visualizar el resultado de **tomo_rec0** que se utilizará para verificar y comprobar el correcto funcionamiento de la herramienta comparado con el método manual. Para ello, se va a tomar el *manifold* azul de la figura Figura 5.6.a para compararlo con el generado de forma manual de la Figura 2.4. Para ello, se utilizará la herramienta 3dmod [21] que permitirá visualizar los MRC de forma cómoda. En la Figura 5.9 se pueden ver ambas segmentaciones, la manual y la automática, lado a lado. Si se observan con detenimiento, se puede notar que son casi idénticas. Por lo tanto, se concluye que la herramienta es equivalente al proceso manual tomado como referencia.

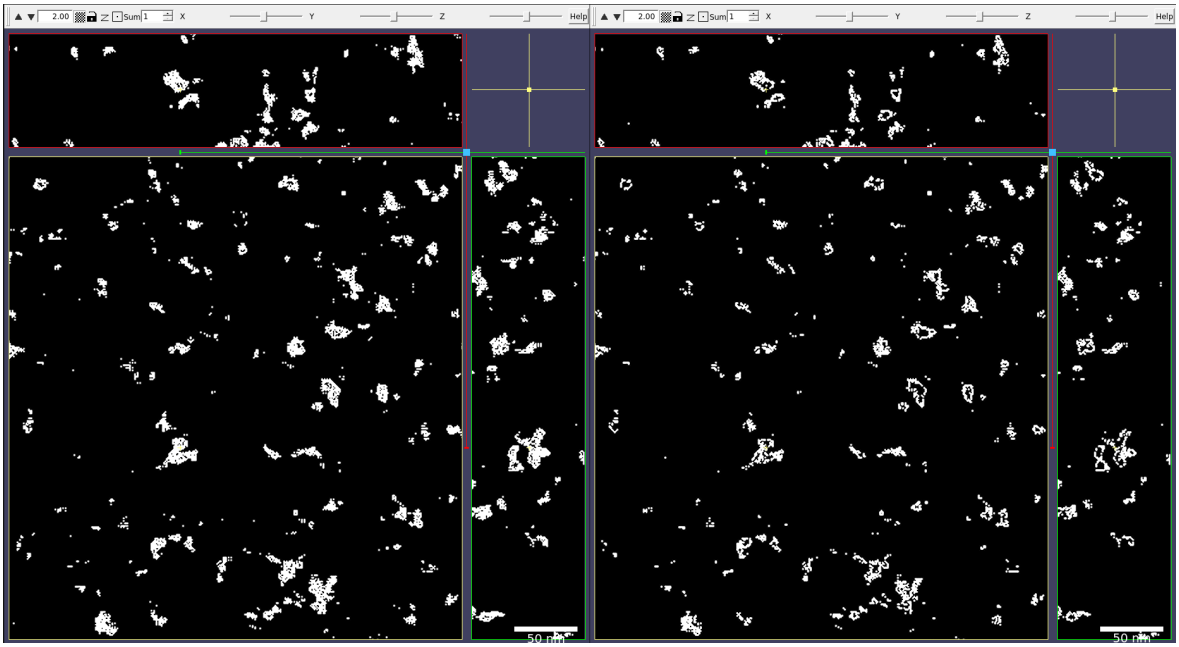


Figura 5.9: Comparación de las segmentaciones de la región azul del **tomo_rec0** obtenidas en el método manual (izquierdo) y el automático (derecho).

6. Conclusiones y vías futuras

Este capítulo tiene como objetivo sacar conclusiones acerca del trabajo realizado así como valorar las posibles vías futuras que se podrían tomar.

Para empezar, los experimentos realizados en el Capítulo 5 han conseguido con éxito el análisis del espacio latente. Según lo expuesto en [3], TomoTwin tiene una buena precisión detectando estructuras similares cuando se hace una selección manual del espacio latente con Napari [20] (ver Sección 2.2). Por lo tanto, se deduce que el método también posee una buena precisión.

Como se puede apreciar en la Figura 5.5, la herramienta desarrollada consigue seleccionar exitosamente aquellos puntos de máxima densidad del espacio latente. Además, más allá de cierta intervención manual para seleccionar cuestiones como la *bounding box*, el umbral de la máscara o el rango de densidades a tratar, todo ha sido de forma automática. Relacionado con la intervención manual, es importante mencionar que todas estas opciones se han podido elegir usando métodos interactivos para conseguir un ajuste fino sin tener que salir del workflow consiguiendo así una flujo de trabajo cómodo.

Una vez seleccionados los focos de interés de forma autónoma habría que decir qué región a su alrededor se va a utilizar. Esto es lo que se muestra en la Figura 5.6 donde se aprecian regiones que abarcan adecuadamente los puntos seleccionados. Esta selección que antes era manual y dependiente del criterio del usuario se hace de forma automática ahora. Recurrir a una herramienta como DisPerSE [4] ha facilitado mucho la inclusión de la topología computacional a la resolución. Además, esta teoría matemática nos da un criterio universal sobre la forma y el tamaño de las regiones.

En cuanto a las vías futuras que se pueden explorar, la primera a destacar sería la de crear una implementación propia del algoritmo de cancelación de símplices y puntos críticos de la teoría de Morse discreta. Aunque DisPerSE es una buena herramienta, su enfoque hacia la astronomía y la cosmología ha dificultado mucho su integración. Además, al no disponer de mucho control sobre su diseño, se puede perder eficiencia que es muy preciada en entornos de altas prestaciones.

Otro aspecto que resulta interesante es la de mejorar la precisión de TomoTwin especializando el modelo que ellos proporcionan o entrenando uno propio para mejorar la precisión en ciertos tipos de tomogramas. De esta manera, se podría elegir el modelo

más especializado para hacer el análisis si se conocen características de la muestra o, en caso contrario, usar el genérico proporcionado por los autores. Relacionado con el análisis del tomograma y la generación del espacio latente, también podría ser interesante desacoplar la herramienta de análisis para poder variarla. En el Capítulo 2, se muestra que existen otras herramientas de procesado de tomogramas como MiLoPYP [5] y que podrían usarse aquí. Ser capaces de elegir una u otra herramienta podría suponer una mejora.

Por todo lo anterior, puede concluirse que la herramienta cumple con el cometido que tenía: la automatización del análisis de tomogramas. Aunque aún tiene mejoras y caminos que se pueden explorar, esta herramienta abre la puerta a un nuevo método de análisis que combina redes neuronales con fundamentos matemáticos. Además, ha demostrado poder obtener eficientemente las regiones del espacio latente para luego su posterior volcado a un MRC como segmentación.

Bibliografía

[1] DELMIC. What is the difference between cryo-em and cryo-et?, 2026. URL https://blog.delmic.com/what-is-the-difference-between-cryo-em-and-cryo-et.

[2] Creative Biostructure. Single particle cryo-em vs cryo-et, Feb 2025. URL https://www.creative-biostructure.com/resource-cryo-em-vs-cryo-et-comparison-guide.htm?srsltid=AfmBOopp22pY2W2o6qWWy9D89ZVx43TZAgR372FxzNMgFT8_CNSejjNZ.

[3] G. Rice, T. Wagner, M. Stabrin, et al. Tomotwin: generalized 3d localization of macromolecules in cryo-electron tomograms with structural data mining. *Nature Methods*, 20:871–880, 2023. doi: 10.1038/s41592-023-01878-z. URL https://doi.org/10.1038/s41592-023-01878-z.

[4] T. Sousbie. The persistent cosmic web and its filamentary structure - i. theory and implementation: Persistent cosmic web - i: Theory and implementation. *Monthly Notices of the Royal Astronomical Society*, 414(1):350–383, April 2011. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2011.18394.x. URL http://dx.doi.org/10.1111/j.1365-2966.2011.18394.x.

[5] Qiang Huang, Yuxuan Zhou, and Alberto Bartesaghi. Milopyp: self-supervised molecular pattern mining and particle localization in situ. *Nature Methods*, 21: 1863–1872, 2024. doi: 10.1038/s41592-024-02403-6. URL https://doi.org/10.1038/s41592-024-02403-6.

[6] Corporate Communications. Tomografía crioelectrónica, Jul 2021. URL https://www.leica-microsystems.com/es/science-lab/life-science/tomografia-crioelectronica/.

[7] 2026. URL https://www.thermofisher.com/es/es/home/electron-microscopy/life-sciences/cryo-tomography.html.

[8] JEOL Ltd. Cryo-electron tomography (cryo-et). https://www.jeol.com/words/emterms/20121023.102157.php, 2026. Accessed: 2026-05-16.

[9] Joachim Frank. *Three-Dimensional Electron Microscopy of Macromolecular Assemblies*. Oxford University Press, Oxford, 2006.

[10] Vladislav Polianskii, Giovanni Luca Marchetti, Alexander Kravberg, Anastasiia Varava, Florian T. Pokorny, and Danica Kragic. Voronoi density estimator for high-dimensional data: Computation, compactification and convergence. In James Cussens and Kun Zhang, editors, *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*, volume 180 of *Proceedings of Machine Learning Research*, pages 1644–1653. PMLR, 01–05 Aug 2022. URL https://proceedings.mlr.press/v180/polianskii22a.html.

[11] Tamal Krishna Dey and Yusu Wang. *Computational topology for data analysis*. Cambridge University Press, 2022.

[12] Aurelie Jodelle Kemme and Collins Amburo Agyingi. Persistent homology: A pedagogical introduction with biological applications, 2025. URL https://arxiv.org/abs/2505.06583.

[13] Antonio Martinez-Sanchez, Inmaculada Garcia, Shoh Asano, Vladan Lucic, and Jose-Jesus Fernandez. Robust membrane detection based on tensor voting for electron tomography. *Journal of Structural Biology*, 186(1):49–61, 2014. ISSN 1047-8477. doi: https://doi.org/10.1016/j.jsb.2014.02.015. URL https://www.sciencedirect.com/science/article/pii/S1047847714000495.

[14] Yilong Zhu, Dong Han, Bohuan Xue, Jianhao Jiao, Zuhao Zou, Ming Liu, and Rui Fan. Road curb detection using a novel tensor voting algorithm, 2019. URL https://arxiv.org/abs/1911.12937.

[15] Lorenz Lamm, Simon Zufferey, Hanyi Zhang, Ricardo D. Righetto, Florent Waltz, Wojciech Wietrzynski, Kevin A. Yamauchi, Alister Burt, Ye Liu, Antonio Martinez-Sanchez, Sebastian Ziegler, Fabian Isensee, Julia A. Schnabel, Benjamin D. Engel, and Tingying Peng. Membrain v2: an end-to-end tool for the analysis of membranes in cryo-electron tomography. *bioRxiv*, 2025. doi: 10.1101/2024.01.05.574336. URL https://www.biorxiv.org/content/early/2025/04/22/2024.01.05.574336.

[16] Rachida Seghiri, Juan Diego Gallego Nicolás, Robert Brandt, Miguel A. Meroño, Pierre Lefevre, Noushin Hajarolasvadi, Daniel Baum, Jessica Heebner, Harold Phelippeau, Pascal Doux, and Antonio Martinez-Sanchez. Tomosegnet: Augmented membrane segmentation for cryo-electron tomography by simulating the cellular context. *bioRxiv*, 2026. doi: 10.64898/2026.01.15.699326. URL https://www.biorxiv.org/content/early/2026/01/16/2026.01.15.699326.

[17] 2024. URL https://tomotwin-cryoet.readthedocs.io/en/stable/tutorials/tutorials_overview.html#tutorial-2-clustering-based-particle-picking.

[18] Tom Burnley, Colin M. Palmer, and Martyn Winn. Recent developments in the ccp-em software suite. *Acta Crystallographica Section D: Structural Biology*, 73 (5):469–477, 2017. doi: 10.1107/S2059798317007859.

[19] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2020. URL `https://arxiv.org/abs/1802.03426`.

[20] Nicholas Sofroniew, Talley Lambert, Kevin Evans, Juan Nunez-Iglesias, Grzegorz Bokota, Philip Winston, et al. napari: a multi-dimensional image viewer for python. *Zenodo*, 2022. doi: 10.5281/zenodo.3555620.

[21] Boulder Laboratory for 3-Dimensional Electron Microscopy of Cells. Introduction to 3dmod, version 4.11. `https://bio3d.colorado.edu/imod/doc/3dmodguide.html`, 2026. Accessed: 2026-05-16.

[22] The pandas development team. *pandas Documentation*. NumFOCUS, 2026. URL `https://pandas.pydata.org/docs/`. Accessed: 2026-05-16.

[23] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020. doi: 10.1038/s41586-020-2649-2.

[24] Ahrens, James and Geveci, Berk and Law, Charles. *ParaView: An End-User Tool for Large Data Visualization*. Kitware, Inc., 2026. URL `https://www.paraview.org/`. Accessed: 2026-05-16.

[25] Kitware, Inc. *The VTK User's Guide*. Kitware, Inc., 11th edition, 2021. URL `https://vtk.org/wp-content/uploads/2021/08/VTKUsersGuide.pdf`.

[26] *VTK Python API Documentation*. Kitware, Inc., 2026. URL `https://docs.vtk.org/en/latest/api/python.html`. Online documentation, accessed 2026-05-16.

[27] Antonio Martinez-Sanchez, Lorenz Lamm, Marion Jasnin, and Harold Phelippeau. Simulating the cellular context in synthetic datasets for cryo-electron tomography. *IEEE Transactions on Medical Imaging*, 43(11):3742–3754, 2024. doi: 10.1109/TMI.2024.3398401.

[28] Yizhou Zhao, Hengwei Bian, Michael Mu, Mostofa R. Uddin, Zhenyang Li, Xiang Li, Tianyang Wang, and Min Xu. Cryosam: Training-free cryoet tomogram segmentation with foundation models. In Marius George Linguraru, Qi Dou, Aasa Feragen, Stamatia Giannarou, Ben Glocker, Karim Lekadir, and Julia A. Schnabel, editors, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2024*, pages 124–134, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-72111-3.

Lista de Acrónimos y Abreviaturas

3D tridimensional.

Cryo-ET tomografía crioelectrónica.

espacio latente mapa de características.

JSON JavaScript Object Notation.

ML *Machine Learning*.

MRC Medical Research Council.

PNG Portable Network Graphics.

UMAP Uniform Manifold Approximation and Projection for Dimension Reduction.

voxels regiones cúbicas dentro del tomograma.

workflow flujo de trabajo.

A. Diagrama de la herramienta diseñada

Este anexo contiene una versión horizontal de la Figura 4.6 con el objetivo de mejorar la legibilidad de la misma.

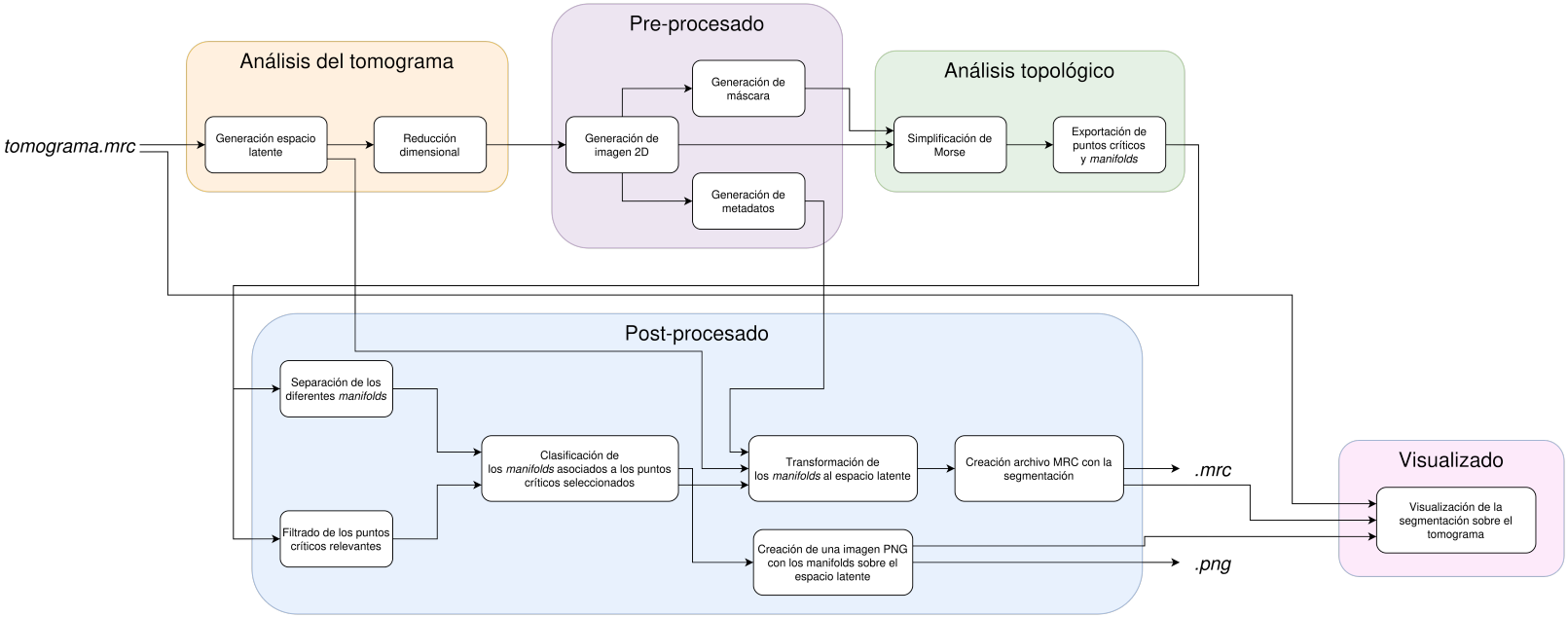


Figura A.1: Workflow completo con todas las etapas de la herramienta interconectadas.