INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Multidisciplinary Project

Identification of Metal Nanoparticle Structures by Computer Vision techniques from TEM Images

Tan Anh Khoa NGO

Le Duc Minh TRAN

4A Génie Physique 2023-2024

Supervisors: **Prof. Romuald POTEAU**

Dr. Iann GERBER

LPCNO Laboratoire de Physique & Chimie des Nano-Objets

Multidisciplinary Project

Identification of Metal Nanoparticle Structures by Computer Vision techniques from TEM Images

Tan Anh Khoa NGO

Le Duc Minh TRAN

4A Génie Physique 2023-2024

Supervisors: **Prof. Romuald POTEAU**

Dr. Iann GERBER

Resumé

Metal nanoparticles, with their unique properties, are revolutionizing numerous fields, and their characterization is a crucial task. Transmission Electron Microscopy (TEM), while vital, demands significant expertise and time for manual analysis. This project, a collaborative effort, leverages the power of computer vision (CV) and machine learning (ML) to automate the identification and classification of nanoparticles from TEM images. By exploring and implementing various CV and ML algorithms, we aim to develop a system that can efficiently and accurately analyze TEM images, thereby alleviating the manual burden on researchers. The project utilizes open-source datasets and images from LPCNO experimenters, a testament to the collaborative spirit of the research community, to train and test the algorithms. This automated approach promises to expedite nanoparticle research significantly, allowing scientists to focus on interpreting and applying results.

Acknowledgements

We are immensely grateful to our supervisors, **Prof. Romuald Poteau**, and **Dr. Iann Gerber**, for their unwavering guidance and stimulating discussions throughout this project. Their insightful advice and critiques, not only on the technical aspects but also on our approach and work ethic, were instrumental in shaping this project into a meaningful contribution that will facilitate future research. Their mentorship has not only helped us in this project but also in our personal growth and learning experiences, for which we are truly grateful. We sincerely thank **Ms. Sylvaine Lohez** for her invaluable project management lectures. These lessons have not only enriched our understanding of project management but also equipped us with a solid foundation for future engineering careers, where managing projects effectively will be a crucial skill. We are confident that the knowledge gained from her lectures will significantly impact our professional journeys. We appreciate **Prof. Xavier Marie**'s efforts in connecting students with real-world project opportunities. Through his dedication, we were able to engage with the real-world problems of nanotechnology research and find solutions for these challenges. We have the chance to apply our knowledge to make accurate results. A special thank you to **Ms. Lise Arnould** and the **entire GP department** for creating a supportive environment and providing us with the time and resources necessary to focus on our project. We also thank our **GP classmates**, seniors, and other friends for their unwavering support and encouragement. They were a source of strength, especially during challenging times. Finally, our deepest gratitude goes to **our families in Vietnam**. Their unwavering love, financial support, and constant encouragement were the cornerstones that allowed us to pursue our studies abroad and envision the future we desired.

Thank you to all who have made this project and our academic journey at INSA possible.

Contents

List of Figures

List of Abbreviations

- AI: Artificial Intelligence.
- BCC: Cube. The atoms are arranged in a centered cubic mesh.
- CALMIP: Medium-sized Computing Center in Midi-Pyrénées.
- CNN: Convolutional Neural Network.
- CPU : Central Processing Unit.
- CUBO: Cuboctahedron. It is a cube with truncated vertices.
- CV: Computer Vision.
- DEC: Decahedron. Polyhedron with 10 triangular faces.
- DL: Deep Learning.
- DNN: Deep Neural Network.
- DODECA: Dodecahedron. Regular polyhedron with 12 faces.
- FCC: Cube. The atoms are arranged in a face-centered cubic mesh.
- FCC-sphere: Sphere. The atoms are arranged in a face-centered cubic mesh.
- GoogleColab: Google Colaboratory.
- GPU: Graphics Processing Unit.
- HCP-sphere: Sphere. The atoms are arranged in a compact hexagonal mesh.
- HRTEM: High-Resolution Transmission Electron Microscopy.
- ICO: Icosahedron. Regular polyhedron with 20 faces.
- LPCNO: Laboratory of Physics and Chemistry of Nano-Objects.
- ML: Machine Learning.
- MPC: Physical and Chemical Modelization.
- NP: Nanoparticle.
- OH: Octahedron. Regular polyhedron with 8 faces.
- OpenCV: Open-Source Computer Vision Library.
- RNN: Recurrent Neural Network.
- RTD: Tetrahedron. Regular polyhedron with 4 faces.
- TEM: Transmission Electron Microscopy.
- TPT: Truncated trigonal bipyramid.
- TPU: Tensor Processing Unit
- μXRD: Micro X-Ray Diffraction.

I. Introduction

Metallic nanoparticles (NPs), smaller than 100 nanometers in diameter, hold immense potential in diverse fields such as magnetic data storage, cancer treatment (hyperthermia), and medical imaging (MRI contrast enhancement). Their material characteristics are heavily influenced by their morphology, including shape, size, and self-assembly tendencies. We can see, for example, the gold NPs with different shapes in Fig. 1. Accurate characterization of NP morphology is, therefore, crucial for optimizing their applications in these areas.

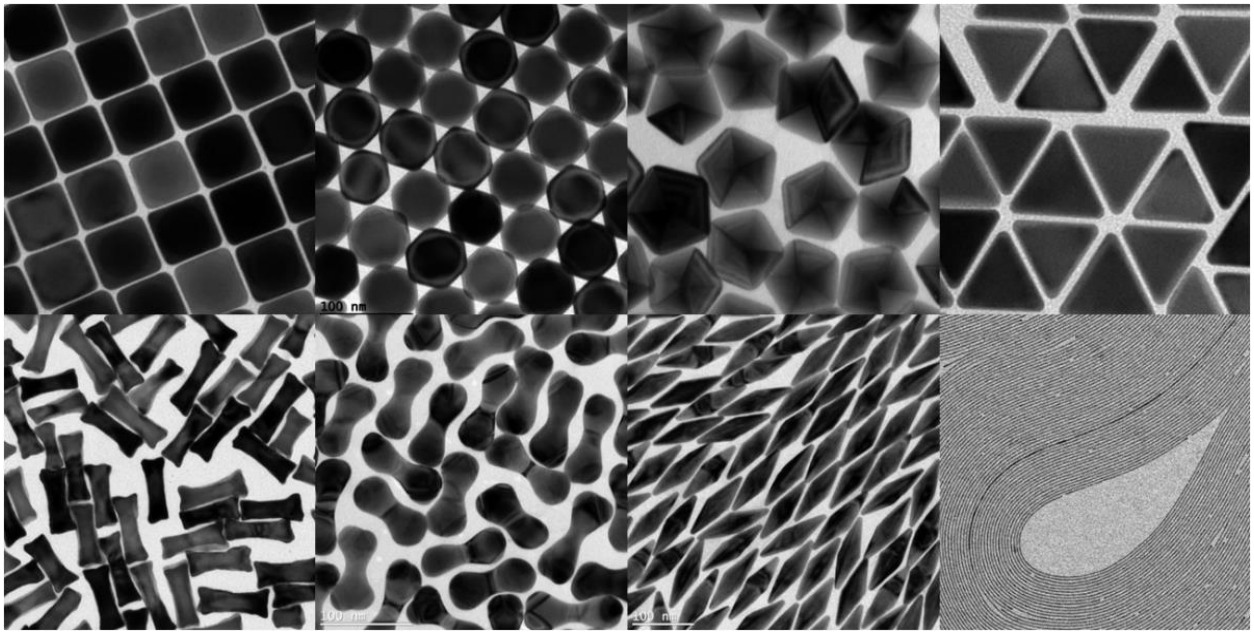


Figure 1 : TEM images of gold nanoparticles in different shapes: cubes, octahedra, decahedra, triangular plates, dog bones, dumbbells, bipyramids, and wires. [5]

Traditional Electron Microscopy (EM) techniques, while valuable for observing nanoparticles, present a significant bottleneck: the manual selection of target particles from vast image datasets. This time-consuming process requires trained microscopists, leading to workload burdens and potential inconsistencies. Computer Vision (CV) techniques, particularly Deep Learning (DL) and its associated libraries, offer a powerful solution for automating TEM data analysis and collection. DL, with its ability to learn and recognize patterns in large datasets, can significantly speed up the process of particle selection and analysis, improving the efficiency and accuracy of nanoparticle characterization.

Our project directly addresses this challenge by leveraging the capabilities of DL to automate NP characterization in High-Resolution Transmission Electron Microscopy (HRTEM) images. The detailed schematic of a HRTEM instrument is shown in Fig.2. This approach significantly reduces the time and effort required for this task, ultimately streamlining nanoparticle research, and improving its efficiency. By automating this process, microscopists can dedicate their expertise to more complex tasks, such as data interpretation, experimental design, and hypothesis formulation, fostering their professional development.

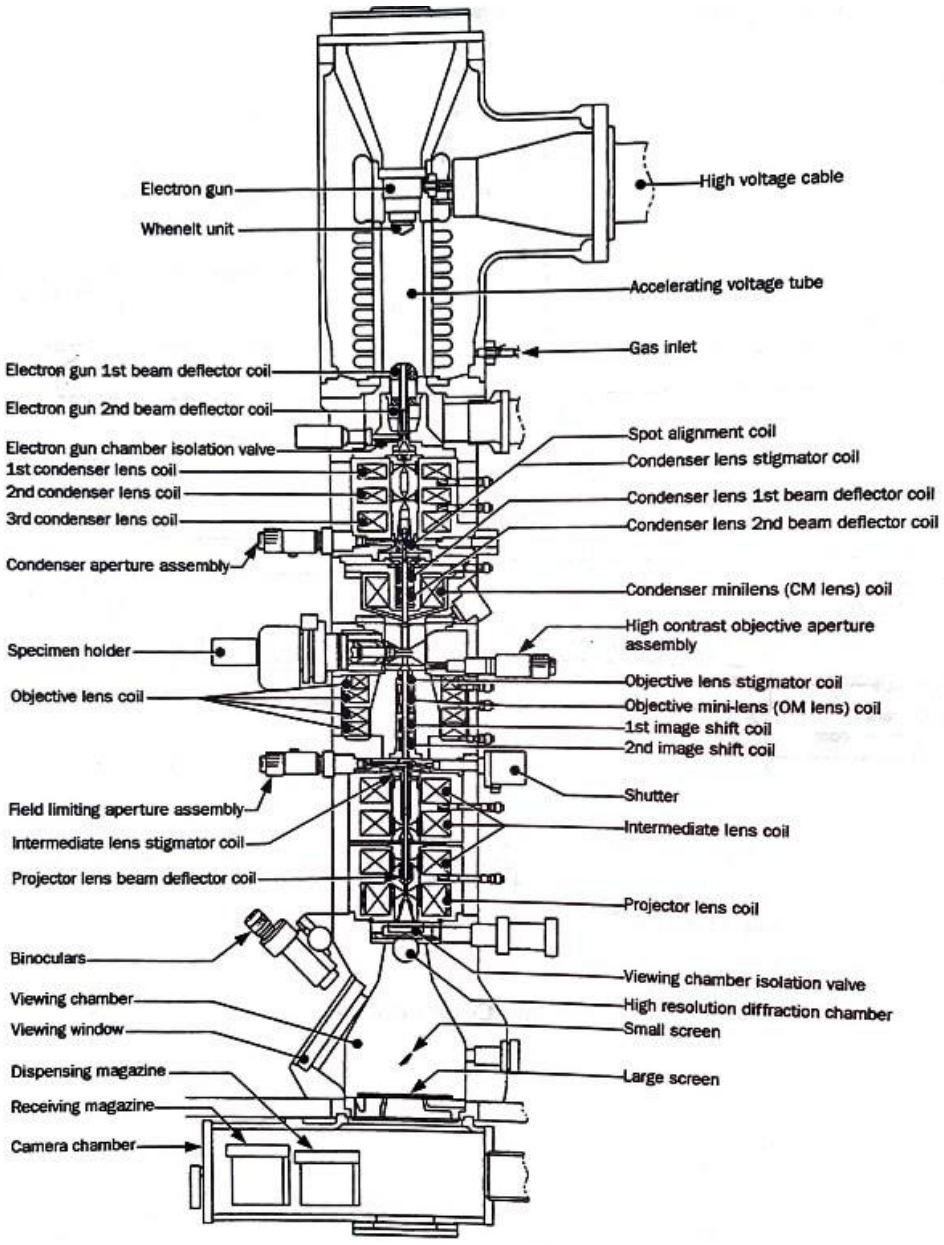


Figure 2 : Schematic of the HRTEM JEOL 3010 instrument. [6]

Building upon the work of our predecessors, who analyzed arrangement diagrams in modeled TEM images, our project focuses on a more robust system capable of handling real-world scenarios, such as images with varying contrast, noise, and particle density. We achieve this by utilizing CV techniques through specialized Python libraries for image processing and artificial intelligence.

Our primary goal is to overcome the limitations of the previous system. While their work successfully analyzed modeled TEM images, it could not handle real-world data and accurately analyze morphological properties like particle count and size. We are confident that with our thorough understanding of the challenges and our innovative approach, we will be able to deliver a robust system that meets your needs and expectations.

This project bridges these gaps by creating a system that effectively identifies and classifies metallic NPs in authentic HRTEM images, focusing on two key aspects:

- Accurate NP Counting and Size Determination: The system will be designed to precisely determine the number and size of nanoparticles in an HRTEM image.
- NP Morphology Classification: The system will be trained to distinguish between 8 pre-defined "classes" encompassing various simple and complex nanoparticle shapes, such as spheres, rods, wires, cubes, and more. This classification will enable researchers to quickly identify and categorize nanoparticles in HRTEM images, enhancing their efficiency.

A detailed explanation of Deep Learning concepts and their application in this project is provided in the "Work Carried Out" chapter. This chapter will delve into the specific DL algorithms used, the training process, and the performance evaluation metrics. It will also discuss the challenges encountered and the strategies employed to overcome them, providing a comprehensive overview of the project's progress and achievements.

This project is primarily an Information Technology (IT) endeavor aimed at automating a task currently performed manually by researchers at the Physical and Chemical Modeling (MPC) team of LPCNO (Laboratory of Physics and Chemistry of Nano-Objects). We were provided a computer equipped with powerful GPUs to facilitate the training and execution of our DL models.

We believe this work holds the potential to revolutionize nanoparticle characterization in HR-TEM studies, significantly accelerating research across various scientific disciplines. Furthermore, we aim to spark the interest of LPCNO experimenters in this innovative approach and potentially contribute to developing Machine Learning applications in nanoparticle research. Given the project's originality and complexity, it will likely continue, guiding others with similar endeavors and transforming how NP characterization is approached. We are excited to share this journey with you and look forward to your valuable insights and contributions.

II. Background

II. 1 Computer Vision: Enabling Machines to See

II. 1. 1 What is Computer Vision?

In the realm of artificial intelligence, computer vision is a testament to the collaborative potential of human and machine intelligence. We can see in figure 3, the comparison between human vision system and computer vision system. It plays a pivotal role in enabling computers to interpret and understand visual information, including images and videos. From an engineering perspective, computer vision is not just about automating tasks typically performed by the human visual system. It's about augmenting human capabilities by extracting, analyzing, and comprehending valuable information from visual data**.** [1]

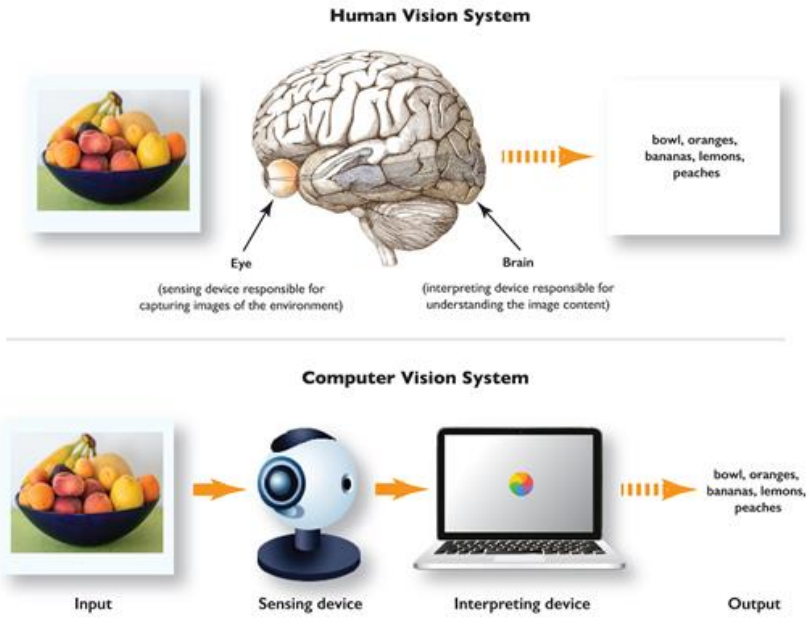


Figure 3 : Comparison between human vision system and computer vision system. [7]

Computer vision can be viewed through two main lenses:

- From a scientific discipline perspective, computer vision forms the foundation of artificial systems designed to extract information from images. Image data can manifest in various forms, from video sequences like those used in surveillance systems, to multi-camera views like those used in sports analysis, and even multi-dimensional data from medical scanners like those used in disease diagnosis. This remarkable versatility underscores the vast spectrum of applications that computer vision can be leveraged for, sparking curiosity, and igniting interest in its potential**.**
- From a technological discipline perspective, computer vision takes on a practical role. It emphasizes the application of theories and models to construct functional computer

vision systems that can be deployed in various industries. This practical approach underscores the tangible impact of computer vision, inspiring the audience with its real-world applications.

II. 1. 2 How Does Computer Vision Work?

Computer vision systems function by mimicking the human visual system's ability to interpret visual information. They achieve this through two key components:

- **Capturing Visual Data:** A sensory device, typically a camera, captures visual data from the environment. This data can be in the form of images or videos.
- **Processing and Understanding:** A computer processes the captured data to extract meaningful information. This involves identifying patterns within the data.

Like how our brains recognize objects by analyzing patterns in shapes, colors, and textures, computer vision algorithms identify patterns within the pixels that make up a digital image. These patterns can then classify and distinguish different objects within the image. [2]

The Algorithmic Process of Image Analysis:

- **Data Conversion:** The initial step involves converting the image into a format suitable for computer processing. This typically involves dividing the image into a grid of small squares called pixels. Each pixel is then represented by numerical values that describe its color and intensity. This creates a digital representation of the image that the computer can analyze. [2]
- **Pattern Recognition:** Once converted, the computer vision algorithm employs techniques from machine learning and artificial intelligence to analyze numerical data. The goal is to identify patterns within the data and make decisions based on those patterns. These patterns can represent object edges, specific textures, or other features that can be used for recognition and classification. [2]

The ultimate objective of CV is to empower computers to analyze and understand visual information, mimicking the human visual system. This understanding allows computers to make intelligent decisions based on the extracted information from visual data [2]. Figure 4 shows the relationship between AI and CV.

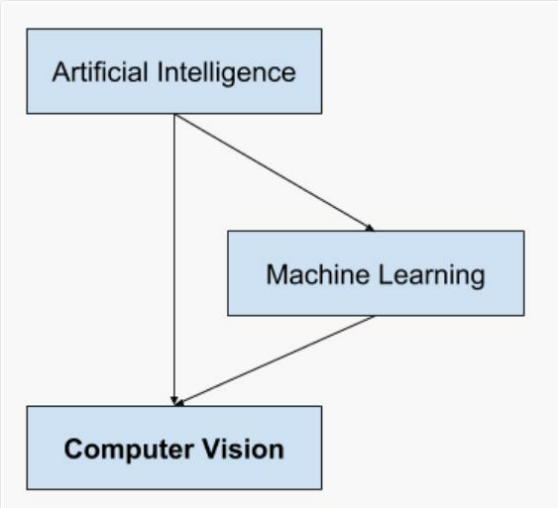


Figure 4 : Overview of the Relationship of Artificial Intelligence and Computer Vision [8].

II. 1. 3 Applications of CV

The applications of computer vision are vast and extend across numerous fields. These applications can be broadly categorized into two main areas:

- **Automated Analysis and Inspection:** In industrial settings, computer vision is used for tasks like automated inspection on production lines, ensuring product quality and consistency. Vision systems can identify defects in plastic or other manufactured goods. Beyond industry, computer vision can assist researchers in various fields by automating identification tasks. This could involve applications like species identification in ecological studies.
- **Intelligent Systems and Robotics:** Computer vision enables robots and other intelligent systems to interact with their environment. Vision systems can be used to control industrial robots by providing them with real-time visual feedback for tasks like object manipulation. Computer vision algorithms can detect events in areas like security and surveillance, such as suspicious activities in video footage. Beyond industrial applications, computer vision can be employed for tasks like people counting and potentially used in the restaurant industry to optimize staffing or analyze customer traffic patterns.

Furthermore, computer vision underpins various human-computer interaction applications. Vision systems can serve as the input mechanism for devices designed for interaction between humans and computers. Additionally, computer vision contributes to tasks like modeling objects and environments. This can be seen in medical image analysis applications where vision algorithms can assist doctors in analyzing medical scans. Similarly, computer vision can be used for topographical modeling, generating 3D representations of landscapes. Navigation is another field that heavily relies on computer vision. Self-driving cars and mobile robots utilize vision systems to navigate their surroundings and avoid obstacles. Finally, computer vision plays a role in information organization. Vision algorithms can automate the indexing of image and video databases, making them easier to search and navigate.

II. 1. 4 Image Analysis: The Foundational Process in Computer Vision

Envision a realm where computers possess the ability to perceive and comprehend the visual data that surrounds them. This is the captivating domain of computer vision, with its core being image analysis. It's a captivating expedition that commences with image acquisition, where cameras or sensors capture the raw information, we seek to decipher. Then, we progress to preprocessing, akin to tidying your room before you embark on organizing. Techniques like noise reduction, resizing, and color correction ensure the data is lucid and primed for analysis.

Feature extraction is where the real magic happens. It's like the computer's eyes are scanning the image, picking out critical details like edges, shapes, and textures. These details are the building blocks that help the computer understand the image's content. It's a bit like you identifying furniture in a cluttered room - feature extraction allows the computer to recognize specific elements like chair legs or tabletops. In simpler terms, it's like the computer is learning to see by identifying the most important parts of the image.

Following this, image segmentation takes the "cleaned room" analogy further. The image is divided into meaningful sections, like separating the furniture from the walls. In a real-world scenario, this might involve isolating individual cells from a microscope image or separating the foreground from the background in a surveillance video.

Finally, object detection and recognition put everything together. Object detection is the process of locating instances of objects in images or videos, while object recognition involves identifying what those objects are. Using the extracted features and segmentation information, the system can now recognize and classify objects within the image. This translates to identifying and classifying the different metallic nanoparticles based on their shapes.

By harmoniously collaborating, these stages empower computer vision systems to not only scrutinize and comprehend visual data but also to make a tangible impact on our world. They pave the way for applications that span from identifying individuals and detecting vehicles to recognizing handwritten text, ultimately enhancing our daily lives.

II. 1. 5 Identifying Metal Nanoparticle Structures: A Practical Application

The project exemplifies the power of computer vision in materials science, a field that focuses on the design and discovery of new materials. In this project, transmission Electron Microscopy (TEM) images were used to provide valuable insights into the structure and morphology of nanoparticles, which are crucial for understanding their properties and potential applications.

Here are how computer vision techniques can be applied in this project:

1. **Particle Size and Count Identification using OpenCV: OpenCV** is a popular library that offers a wide array of image processing and analysis tools. This versatile library provides an extensive toolkit for image processing and analysis, encompassing tasks such as image enhancement, segmentation through techniques like thresholding or edge detection, and contour analysis to determine nanoparticle size and count.
2. **Particle Shape Classification using CNNs:** DL models like CNNs excel at image classification. In our project, a CNN model is trained on a dataset of labeled TEM images, learning to differentiate between various nanoparticle shapes (e.g., spherical, rod-shaped, cubic). This trained model then automatically extracts relevant features and classifies the shapes of new, unseen nanoparticles.

By harnessing the combined power of OpenCV and CNNs, this project streamlines the analysis of TEM images, providing researchers with rapid and precise information regarding the size, count, and shape of metallic nanoparticles. This automation significantly accelerates research and development efforts in nanotechnology and materials science, optimizing resource utilization while ensuring accuracy and reliability.

II. 2 Convolutional Neural Networks [3]

II. 2. 1 Deep Learning: A Powerful Tool for Artificial Intelligence [4]

Deep Learning, a subfield of machine learning and a branch of artificial intelligence (AI) has gained significant traction due to its ability to process complex and unstructured data. This surge in popularity has led to a shift in how Machine Learning is conducted, with neural networks now being the primary tool. To delve deeper into these concepts, we undertook the Fidle online course 'Introduction to Deep Learning' by the CNRS (French National Center for Scientific Research). This French-language module offers a wealth of explanatory videos, code examples, and resources, all of which underscore the practical applications of Deep Learning in AI. We can see in figure 5, the relationship between AI, ML and DL.

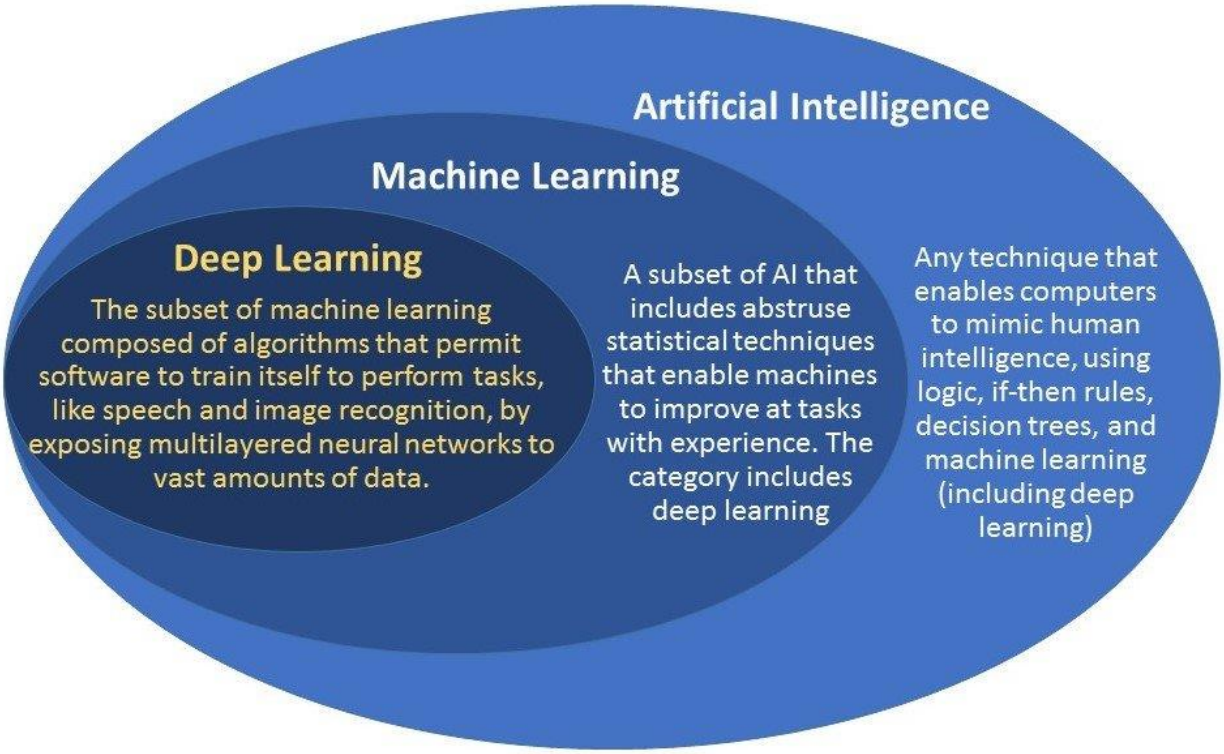


Figure 5 : The relationship between AI, ML and DL and models currently employed in survival analysis. [9]

In our study of predicting nanoparticle morphology, we leverage the power of a neural network. This network, composed of elementary cells called neurons, forms the backbone of our prediction model. Each neuron transforms an input vector x into an output vector y, with its weight, bias, and activation function defining its unique characteristics. The output of a neuron is influenced by the values of the neurons preceding it in the structure, making the neural network a potent tool for predicting complex nanoparticle morphologies. In this context, the role of neural networks cannot be overstated, as they are the key to our ability to make accurate predictions. Figure 6 shows a structure of a neuron.

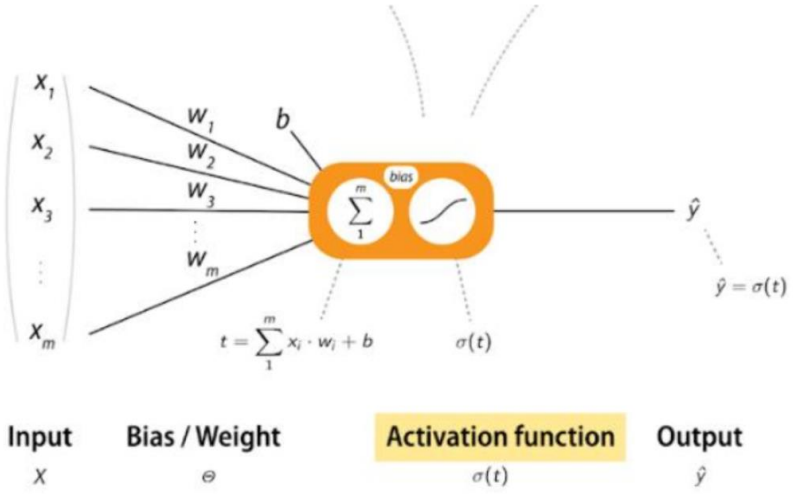


Figure 6 : The structure of a neuron. The Wi are the weights assigned to the input components xi, b is the bias of this neuron, and σ is the activation function. y is the weighted sum of the inputs, then transformed by a specific activation function.

Weights and bias are adjustable parameters instrumental in generating predictions within a neuron. Weights determine the relative influence of different inputs on the output, enabling the network to prioritize various features or connections during decision-making. This means specific inputs can have a stronger or weaker impact on the final prediction, depending on their weight. During the learning process, these weights are fine-tuned to optimize the network's performance. On the other hand, bias represents constant values added to the outputs of neurons to introduce a shift in the activation function. This shift allows better control of the activation function's threshold and enhances data modeling. Adjusting these parameters during training ensures that the predictions align with the experimenter's expectations.

The primary advantage of activation functions lies in their ability to introduce non-linearity. While linear functions can only learn and represent linear relationships, non-linear activation functions can capture more complex patterns and relationships (the physical world is rarely linear). By employing non-linear activation functions, neural networks can handle non-linear problems and enhance the model's representational power. Consider an image as an example; non-linear characteristics might include curvatures, boundaries, spacing between different shapes, etc. Activation functions are one of the most crucial elements in learning these features and identifying nanoparticle morphologies.

Various activation functions are commonly used in neural networks, including ReLU, Softmax, Sigmoid, Softsign, Selu, tanh, and others. The choice of activation function depends primarily on the specific application. In the context of our neural network for nanoparticle morphology prediction, the task is to classify the morphology of a nanoparticle in an image. This requires a vector that encompasses all possible classes. The network's prediction for a given image is the element with the highest probability in the output vector. To ensure a well-normalized prediction, a crucial aspect of model precision and reliability, the Softmax function is employed in the final layer (output layer), which generates the final prediction. As a result, the final layer

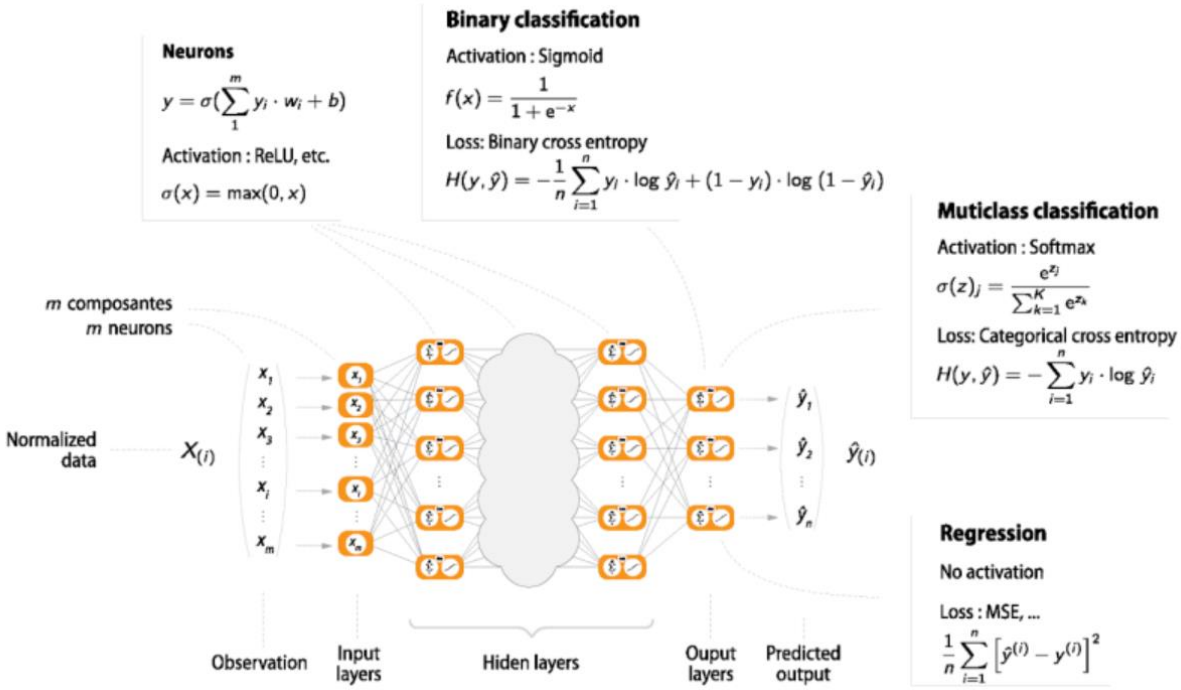


Figure 7 : The structure of the output layer: yi is the prediction of the ith class. The last layer uses the Softmax function. x is the activation value of the ith output neuron.

produces a list of 8 elements, each representing a probability. The class with the highest probability is assigned as the network's response. For the remaining layers within the network, the ReLU function is often utilized. This function is among the fastest activation functions regarding computation time, as it only involves two linear intervals. Functions like Softmax, on the other hand, require more computational time due to the involvement of exponential calculations, but they significantly enhance the model's accuracy and reliability. Figure 7 illustrates the structure of the output layer.

Neural network training, or model or network training, is a crucial process that equips the network with a vast dataset. This dataset is instrumental in allowing the network to fine-tune the parameters of each neuron, namely the weights and biases, to achieve optimal performance. However, the real magic happens with backpropagation. Backpropagation, a method used in artificial neural networks, is the secret sauce that calculates the gradient of the loss function concerning the weights and biases. This gradient then updates the weights and biases, enhancing the network's performance. It is like a coach correcting the player's moves after each game, making them better and better. The loss function, calculated to minimize the disparity between the network's prediction and the correct answer in the dataset, serves as a performance metric for our Deep Learning algorithm. By minimizing this loss function, our model enhances its performance. In the context of our training, the correction vector has a length of 8, representing the eight classes of nanoparticle morphology. For an image in the dataset, the element associated with the correct class is 1, and the other components of the vector are zeros. We denote this vector as Y. The loss is proportional to the difference in the norm of this vector Y and the vector output by the neural network. The loss depends on the entire set of parameters (weights and biases). Here is the definition of the loss function:

$$Loss = ||Y - Y_{pred}||^2$$

Let us explore how to minimize this function. We apply the gradient descent algorithm to this loss function to find a local minimum. Figure 8 shows the gradient descent process.

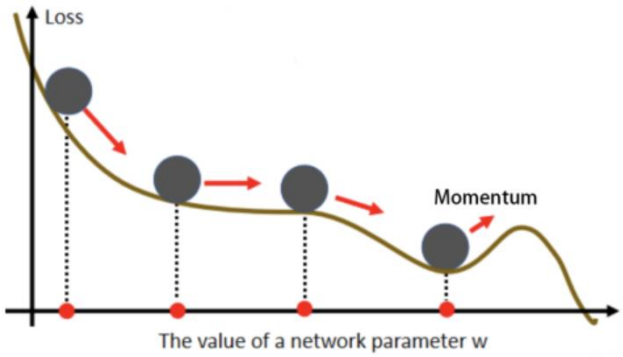


Figure 8 : A gradient descent process and momentum. Only one weight parameter, w, is represented in the graph. In reality, the starting set is a vector that contains all the weight and bias parameters.

Gradient descent is an iterative optimization algorithm widely used in training neural networks. It aims to minimize a loss function by adjusting the weights and biases of the network's parameters. This process enables the network to make increasingly accurate predictions. The Training Process is followed:

- Dataset Split: The dataset is divided into a training set (train) and a validation set (test).
- Initial Predictions: The untrained model is fed the training set. At this stage, the network randomly assigns classes to the images.
- Backward Propagation: After processing the entire training set, backpropagation is performed. The loss statistic is calculated using the training set.
- Epochs: The concept of epochs, a key ingredient in our training process, represents a cycle of feeding the training set and performing backpropagation. This cycle is repeated as often as the experimenter desires, refining the model's accuracy. It is like a marathon runner, getting better with each lap. The model's accuracy improves with each epoch as the network learns from the training data and adjusts its parameters to make more accurate predictions. This iterative training process makes our model more precise with each epoch.
- Validation Set Evaluation: The validation set plays a crucial role in our training process. It is used to evaluate the model's performance on unseen data, a key indicator of its generalization ability. Regular evaluations, such as at each epoch, help assess whether the network has effectively learned from the training data and can generalize to new situations. However, there is a risk of overfitting. Overfitting occurs when the model becomes too specialized in the training data and performs poorly on new, unseen data. It is like a child who can only identify the fruits you showed them but struggles with new fruits. We must be cautious when adjusting the model's parameters based on the

validation set to prevent overfitting. It is important to note that backpropagation needs to be performed on the validation set, which could lead to overfitting (fig. 9).

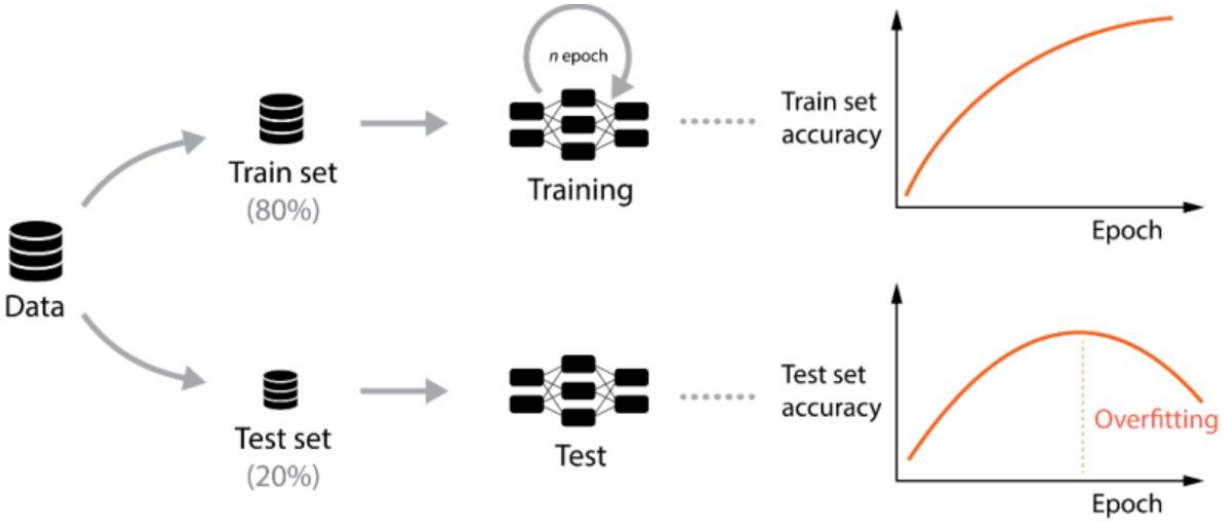


Figure 9 : Training diagram. The precision on the training dataset continues to increase over the epochs up to a maximum ceiling, which is characteristic of the model. On the other hand, if we continue too long, we observe a phenomenon of overfitting at the level of the validation precision: The model is entirely calibrated for the prediction of data coming from the training dataset so that it seeks to bring back to a known case, which generates errors and a drop in precision on the test dataset.

II. 2. 2 Typical Network: Features of a DNN [4]

Deep Neural Networks (DNNs) are theoretical constructs and robust models for deep learning with real-world applications. They consist of multiple successive layers of neurons, each connected to all neurons in the previous layer at its input and to all neurons in the next layer through its output (Fig. 10). These are also called fully connected layers. The characteristic parameters of a DNN are the number of layers and the number of neurons per layer.

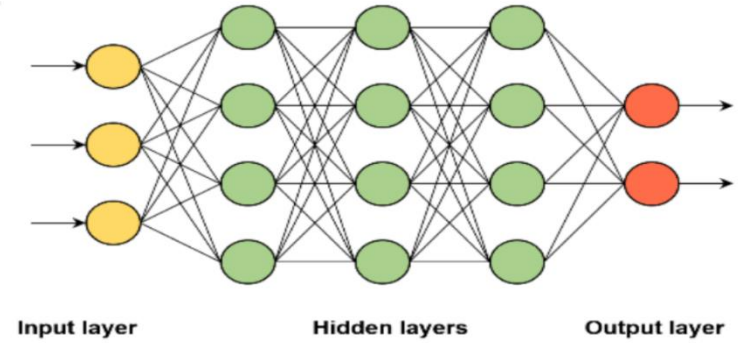


Figure 10 : Structure of a typical DNN.

DNNs offer numerous advantages, notably their capacity to learn and represent intricate patterns and features in data. By stacking multiple layers, DNNs can extract abstract features from raw data and progressively construct increasingly abstract representations. This unique ability has led to significant breakthroughs in image recognition, speech recognition, natural language processing, and recommendation systems, addressing some of the most complex challenges in these fields.

One drawback is the need for an extensive training dataset. In practice, a well-annotated and huge dataset is often required. This involves a tremendous amount of work in labeling the data, where each image in our project's dataset of unique nanoparticles is associated with a specific class (via a number from 0 to 10). This meticulous process is crucial for the DNN to learn and recognize these unique nanoparticles.

Generally, the number of layers and neurons is expected to be moderate for each database. A high number of layers or neurons are only sometimes appropriate. The influence of these parameters was tested during the project.

II. 2. 3 CNNs in the Project [4]

CNNs are a potent type of deep neural network specifically engineered to analyze and recognize patterns in visual data, particularly images. Their ability to extract features and identify relationships within image data is awe-inspiring, making them a thrilling tool for tasks like image classification, object detection, and image segmentation.

CNNs consist of several essential layers (fig. 11) that work together to achieve their remarkable performance in image processing tasks:

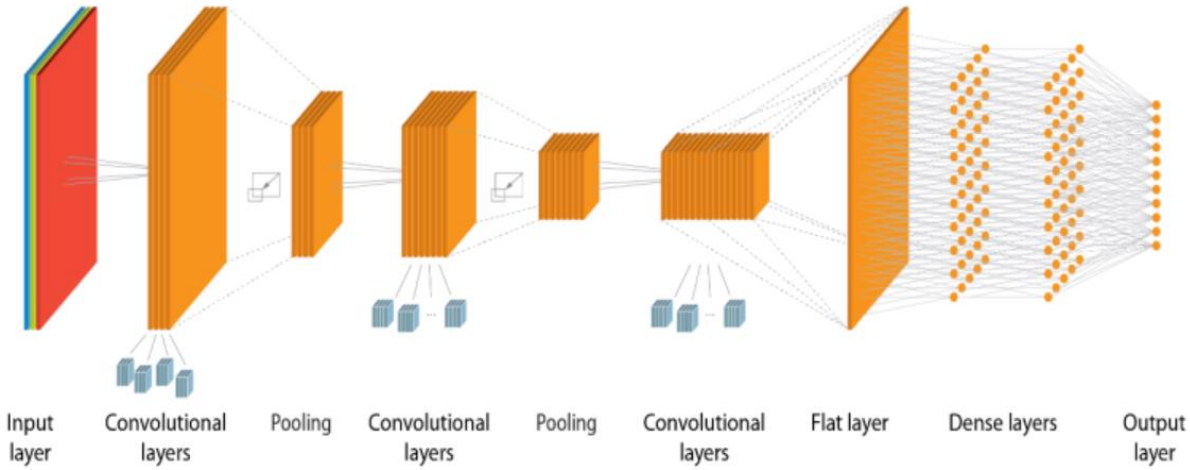


Figure 11 : Schematic of a CNN.

- Convolutional Layers: The core of CNNs, convolutional layers perform convolutions, a mathematical operation that slides a filter kernel over the input image to extract local features. These filters detect specific patterns or features in the image, such as edges, corners, or textures.
- Pooling Layers: After each convolutional layer, pooling layers are typically employed to reduce the dimensionality of the feature maps. This process, known as pooling, down samples the data by summarizing the values in smaller regions of the feature maps. Standard pooling techniques include max pooling, which selects the maximum value from each region, and average pooling, which averages the values.
- Fully Connected Layers: While fully connected layers are a common feature in deep neural networks, their role in CNNs is unique. In CNNs, these layers are strategically

placed at the network's end after the convolutional and pooling layers have extracted and processed the image features. Their primary function is to transform the flattened feature maps into a vector representation that can be used for classification or regression tasks.

- Flattening Layer: In CNNs, a flattening layer is often used to transform the output of the convolutional and pooling layers into a one-dimensional vector suitable for input to the fully connected layers. This process involves converting the multi-dimensional feature maps into a single, long list of values.

While fully connected layers are commonly used in CNNs, an alternative architecture, the Fully Convolutional Neural Networks (FCNs), opens up a world of new possibilities. FCNs eliminate the fully connected layers and instead employ convolutional layers throughout the network, even for the final classification or regression stage. This innovative approach can significantly benefit tasks requiring spatial reasoning or pixel-wise predictions, inspiring new avenues of exploration.

The project's focus on CNNs with fully connected layers has demonstrated their effectiveness for specific image-processing tasks. However, exploring FCNs in future projects could provide additional insights and enhance performance for other image processing challenges. The decision to utilize FCNs would depend on the specific requirements and characteristics of the task.

Detailed Explanation of CNN Layers:

Convolutional Layers (fig. 12**)**:

- Purpose: Extract local features from the input image.
- Filter Kernels: The core components of convolutional layers and filter kernels are small matrices that slide across the input image, performing element-wise multiplication and summation.
- Kernel Dimensions: These are represented by (kx.ky.n), where kx and ky represent the kernel's height and width, and n represents the number of input channels (e.g., RGB for color images).
- Convolution Operation: For each pixel in the output feature map, a portion of the input image is multiplied by the corresponding filter kernel, and the products are summed.
- Activation Function: After convolution, an activation function is applied to introduce non-linearity and enhance feature representation.

- Output Feature Map: The resulting matrix of values forms the output feature map, representing the extracted features for that specific filter.

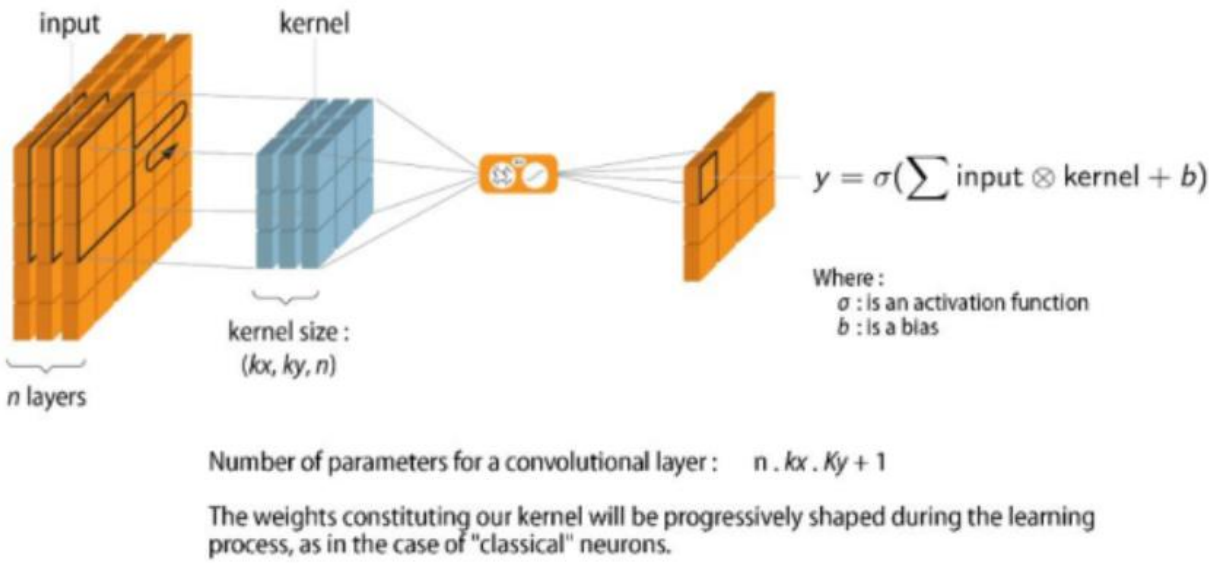


Figure 12 : Convolution of an RGB image (3 channels) by a CNN layer with bias b and activation function σ.

Pooling Layers:

- Purpose: Reduce the dimensionality of feature maps and control overfitting.
- Pooling Operations: Common pooling techniques include max pooling and average pooling.
- Max Pooling: Selects the maximum value within a predefined region of the feature map (fig. 13).
- Average Pooling: Computes the average value within a predefined region of the feature map.
- Reduced Feature Maps: The resulting feature maps have reduced dimensions, making the network more computationally efficient.

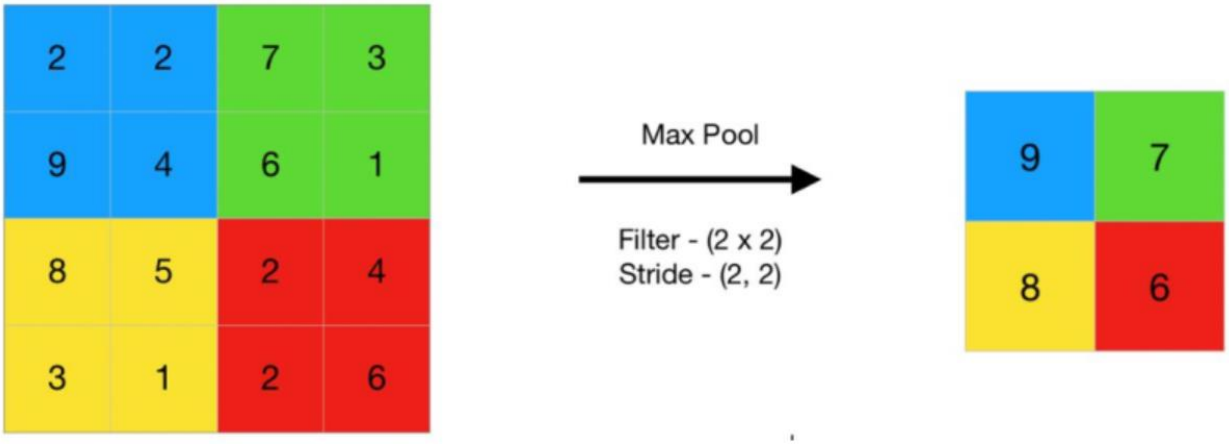


Figure 13 : An example of Max Pooling.

Dropout:

- Purpose: Dropout is a powerful regularization technique crucial in preventing overfitting. It achieves this by randomly dropping a portion of neuron activations during training, forcing the network to learn more robust and generalizable features.
- Regularization Technique: Dropout forces the network to learn more robust and generalizable features.
- Dropout Rate: The proportion of neuron activations to be dropped, typically between 0 and 1.
- Implementation: Dropout can be applied to convolutional or fully connected layers.

CNNs are potent tools for image analysis and recognition, and the project's work has demonstrated their effectiveness. Understanding the concepts and theories behind CNNs is crucial for designing and implementing effective deep-learning models for image-processing tasks. The project's exploration of dropouts as a regularization technique further highlights the importance of preventing overfitting and improving model generalization.

III. Methodology

Figure 14 shows the overview of our project workflow.

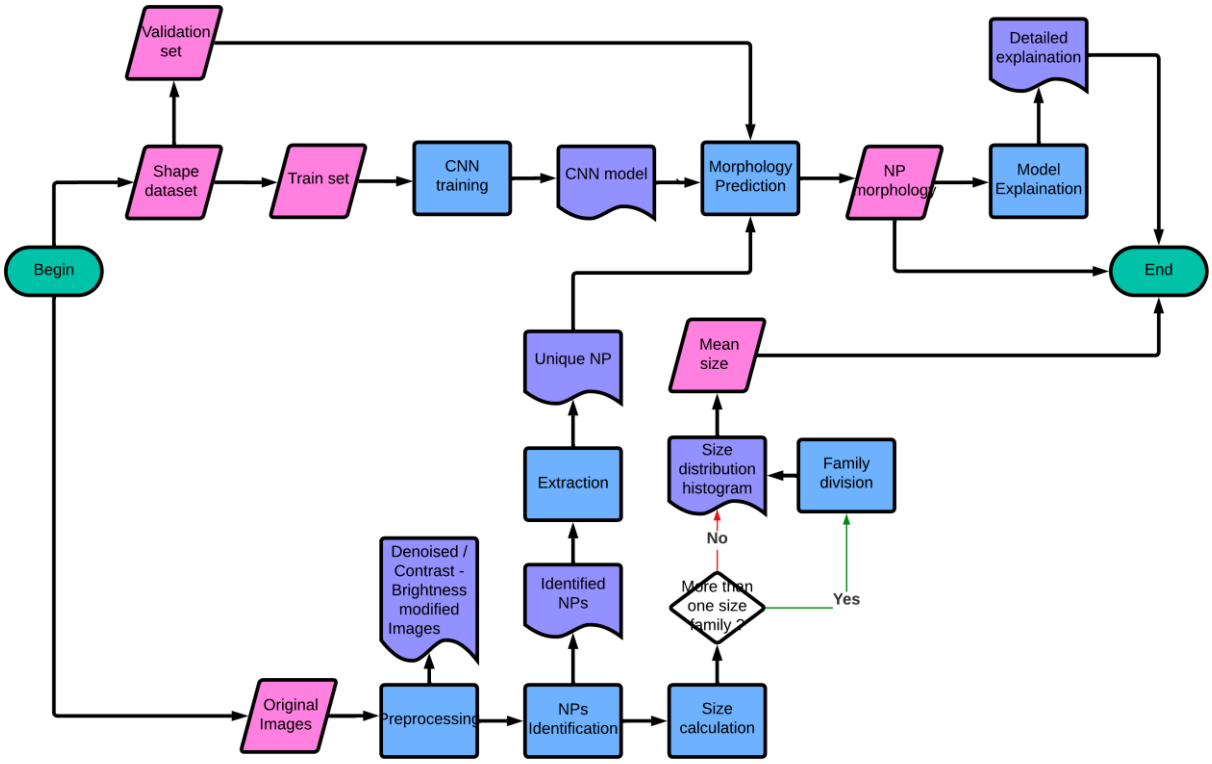


Figure 14 : The Workflow chart of our project.

III. 1 Data Acquisition and Preprocessing

III. 1. 1 Image Input and Denoising

The initial step in any image processing method is image acquisition and image preprocessing. This crucial stage not only determines the type of data we work with but also extracts valuable information from these data, which is pivotal for our subsequent actions. This practical approach to image processing is what makes it applicable in real-world scenarios.

The images we work with usually come from the internet (articles, books, and other resources) and the database provided by our tutors. In this project, we mostly encounter two types of images: PNG and JPG. The differences between these two have hardly any impact on our program. The original images are mainly in RGB color format (fig. 15). In short, the original images are 3d – tensors with the dimensions (h, w, 3) where h and w are, respectively, the height and width (in pixels) of the images. The value of each component of each channel varies from 0 (no intensity) to 255 (maximum intensity), and the value of the color pixels is the linear combination of those components from the three-color channels (Red, Green, Blue). The RGB color format is usually complicated and not very useful in this project. Therefore, our first work is to convert our images into grayscale, a matrix whose pixel values vary from 0 (all black) to

255 (all white). This conversion simplifies the image, making it easier to process and understand.

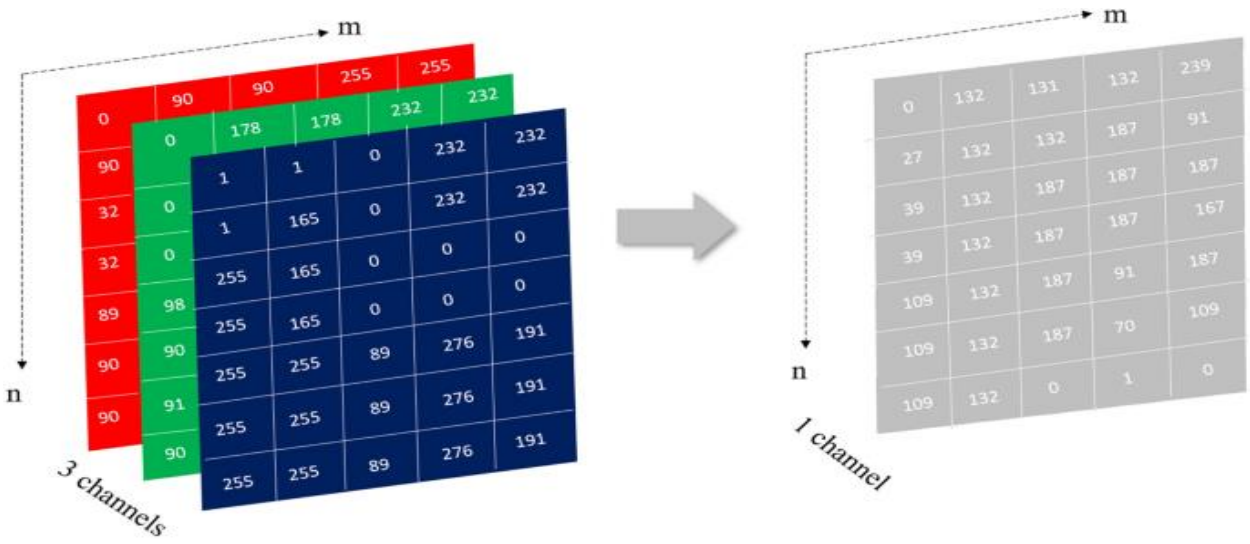


Figure 15 : Example of RGB color format and grayscale color format.

The next step in our algorithm is noise reduction. The principal type of noise in TEM or HRTEM clichés is Gaussian noise caused by high temperature, electronic circuits, and sensor noise, etc. We use the Fast Non-local Means Denoising provided by the OpenCV2 library to reduce this noise. The method is a convolution of the original image with a denoising kernel. In this case, the denoising kernel is a square centrosymmetric matrix whose components weight follows a Gaussian distribution, and the sum of all components equals 1 (fig. 16).

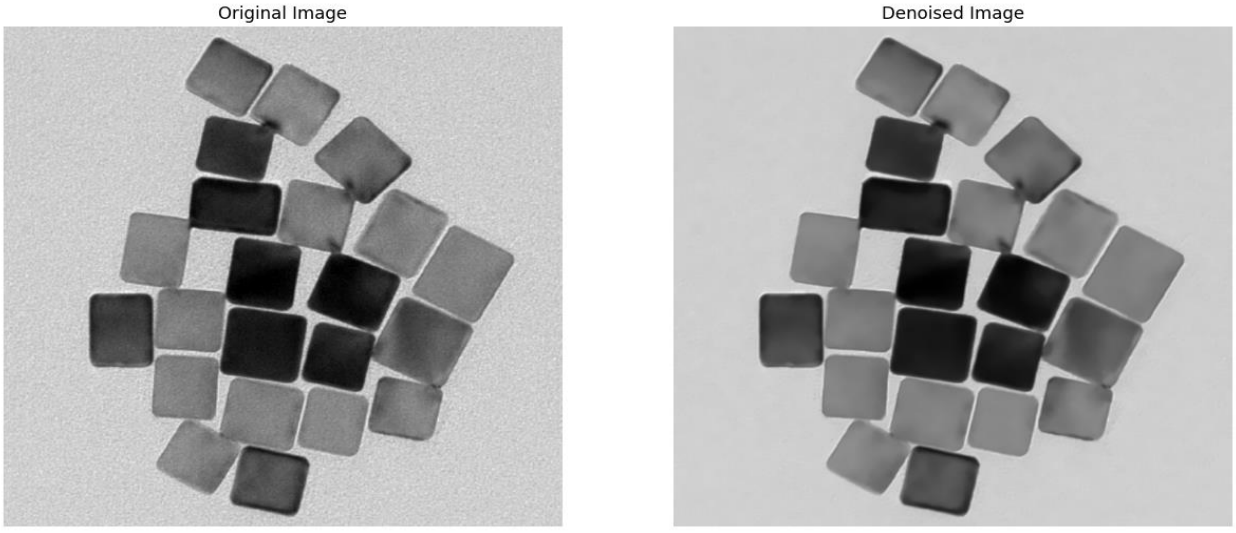


Figure 16 : Original HRTEM image (left) and denoised HRTEM image (right).

III. 1. 2 Contrast, Brightness Adjustment

The most challenging step, and also the most important one in this stage, is image binarization. Finding an optimal set of parameters in this step isn't easy as it seems. The reason for that is

discussed just below. Even after being denoised, there are still some fluctuations in pixel values, which may cause some errors in the next step. This step helps obtain the "mask" used to distinguish the background and object (nanoparticle in our case).

The first part of image binarization is to adjust the contrast and brightness of the image. Changing the contrast and brightness means changing the weight and bias of each pixel in the image. In short, each pixel value follows the following equation:

$$g(i,j) = \alpha \cdot f(i,j) + \beta$$

With $f(i,j)$ the original pixel value, $g(i,j)$ the adjusted pixel value, α the desired contrast, β the desired brightness.

As we mentioned, each pixel takes a value from 0 to 255, but in Python, the value of these pixels is periodic. In other words, the pixel with a value equal to α is identical to a pixel with a value equal to $\alpha + 2 \cdot 255 \cdot k$ with k as an integer. This property causes some undesired results in our contrast and brightness adjustment method. To solve this problem, we add two saturation limits in the ***contrast_brightness_modify()*** function: one upper limit for values >= 255 and one lower limit for values <= 0.

Each image requires different optimized contrast and brightness values; thus, the process is experimental and manual, which means that we have to adjust and try these parameters a few times to obtain a desired binarized image. This makes our algorithms impractical and situational. We use a technique called Clipping Histogram Equalization to make this step automatic (fig. 18). Firstly, we construct a grayscale histogram by counting the number of each pixel for each pixel value (256 intervals). Then, we clip vertically a tiny percentage of the histogram (1% of the maximum value in our case). Lastly, we reconstruct our histogram by stretching the new minimum value to 0 and the new maximum value to 255. By doing this, we can separate the two peaks in our histogram, which are the background and the nanoparticles (fig. 17).

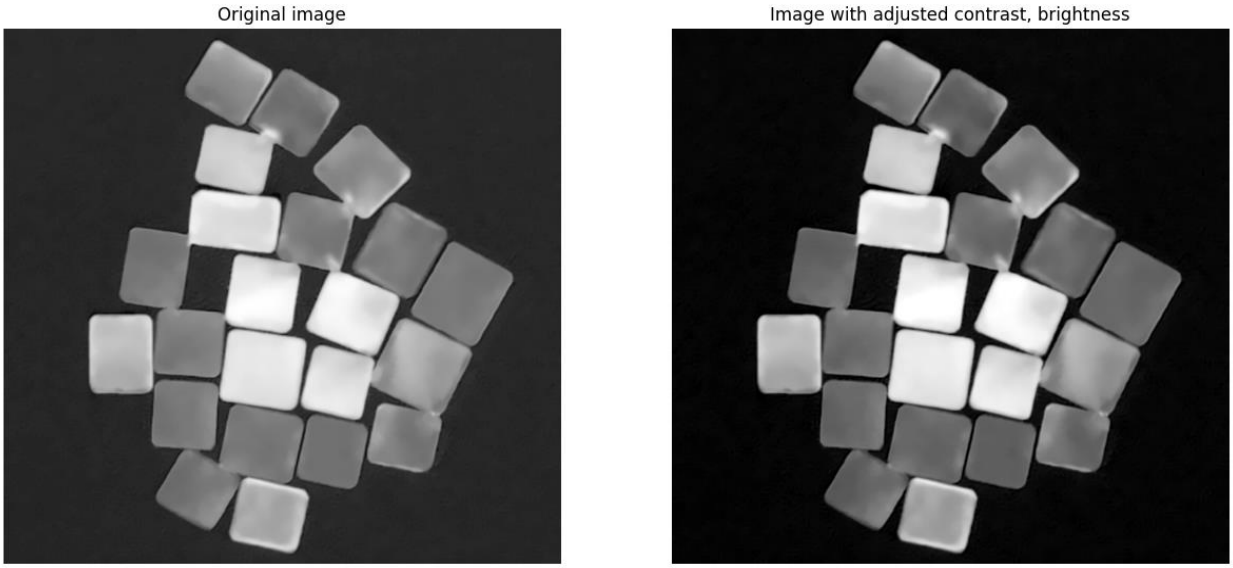


Figure 18 : Original image (with inversed color so that the background is black) and the adjusted image.

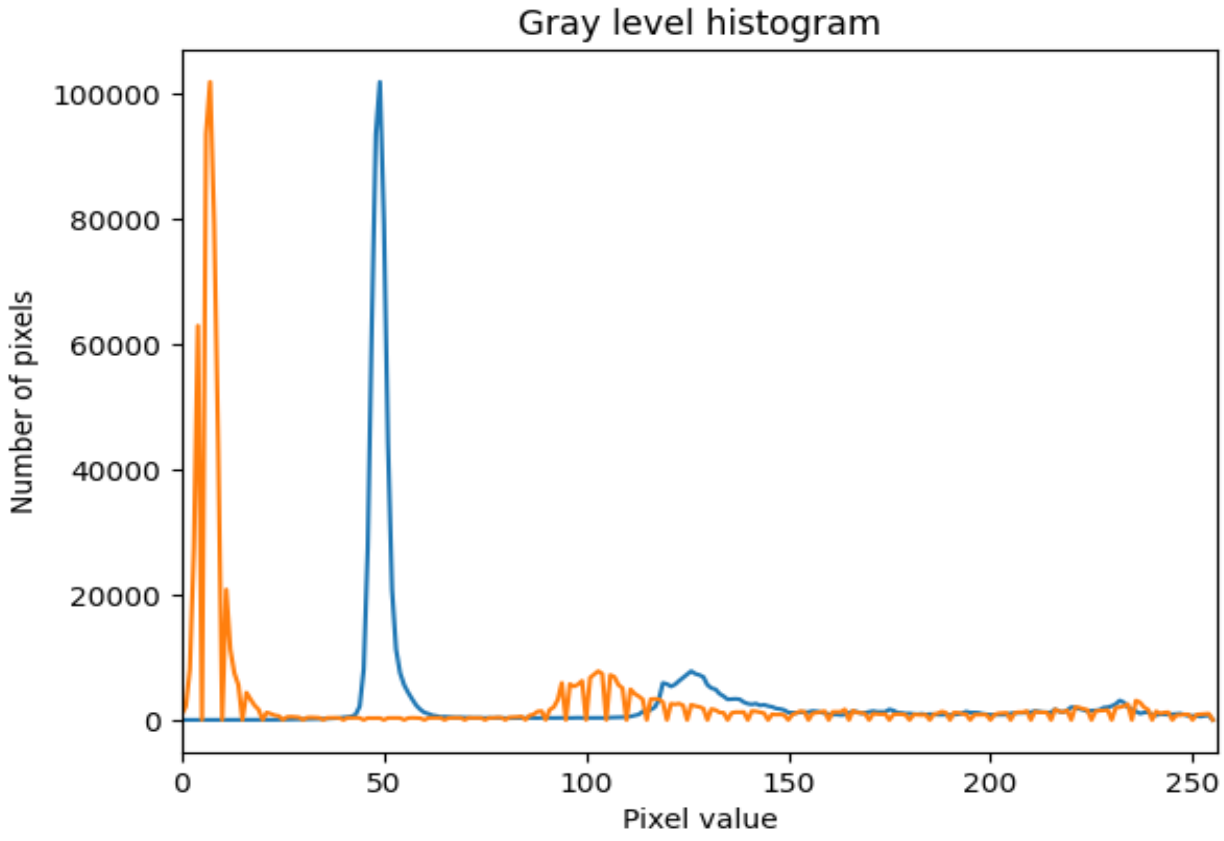


Figure 17 : Grayscale histogram of the original image (in blue) and after adjusted (in orange).

III. 1. 3 Image Binarization

Afterward, we apply a sharpening filter by convolving the image with a very specific kernel as following:

$$\begin{vmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{vmatrix}$$

This sharpening filter helps make the outlines of a digital image look more distinct. Sharpening increases the contrast between edge pixels and emphasizes the transition between background and object. In our context, 'local contrast' refers to the difference in pixel intensity between a pixel and its immediate neighbors. This means that a pixel's intensity is compared to the average intensity of its surrounding pixels. We use this technique to increase the local contrast on the lines between neighboring nanoparticles and nanoparticles in coalescences while conserving other pixels.

After the sharpening, we employ Otsu's highly efficient and adaptive thresholding method from the OpenCV2 library. Unlike manual thresholding, Otsu's method automatically determines the threshold value, making it a reliable choice for images with equalized histograms. The detailed algorithms and theory behind these methods can be found in OpenCV2's document.

The results of the previous steps are shown in figure 19:

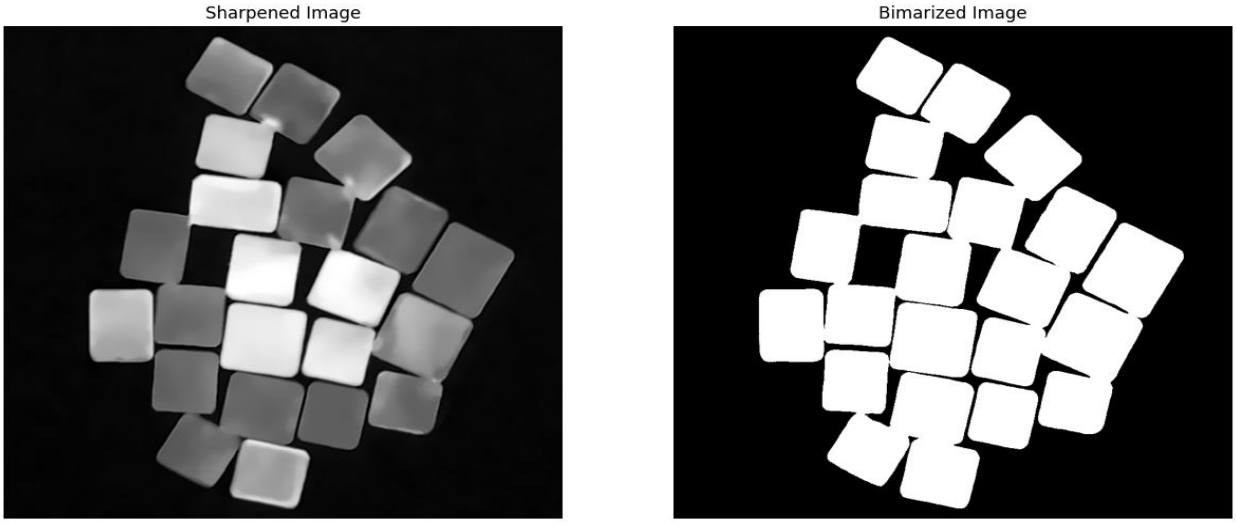


Figure 19 : ***Sharpened image and binarized image.***

In our algorithm, we harness the power of two robust morphological operations to further enhance the quality of the mask. The first operation, the morphological opening, is a potent technique that effectively eliminates small objects and thin lines in an image. We aim to bring out the coalescence lines between nanoparticles and eventually eliminate them. The parameters

for this operation, ***kernel_morpho***, and ***open_iter***, play a crucial role in determining the operation's impact. The larger these parameters, the more pronounced the operation's effect.

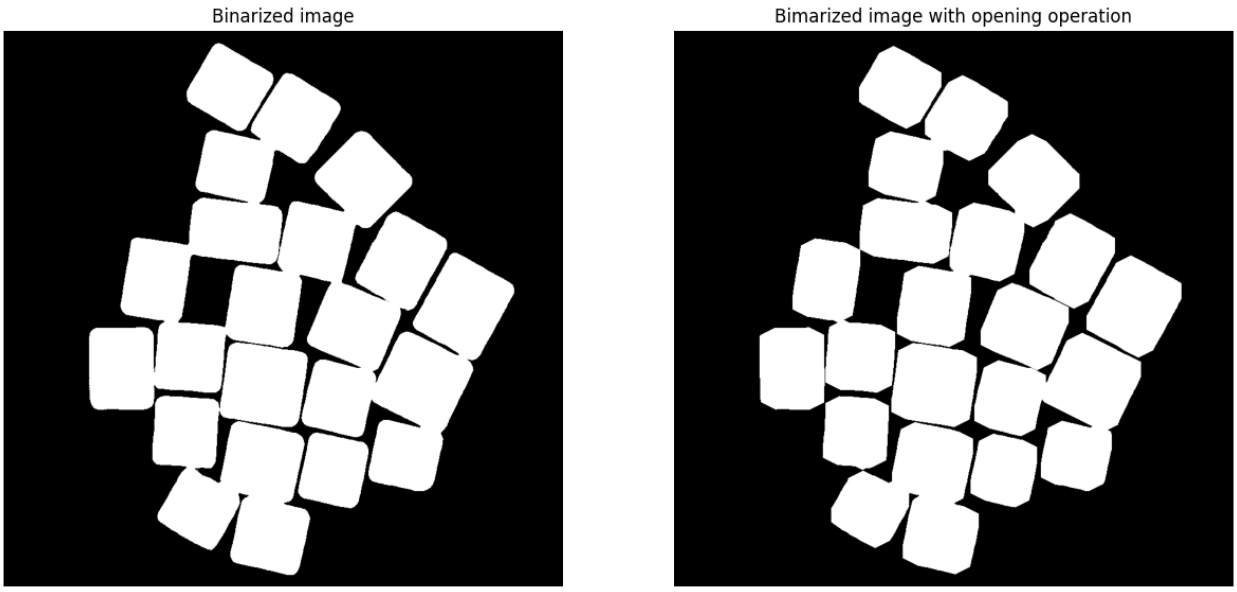


Figure 20 : The impact of opening operation (with the kernel size of 5 and 11 iterations).

In figure 20, we can notice that some interconnecting lines between nanoparticles have disappeared.

The second morphological operation is called the closing operation (fig. 21). It is the opposite of the opening operation. We use this operation to close holes inside the mask that are defects caused by our automatic contrast adjustment and thresholding. The closing operation is crucial in our algorithm as it helps to ensure that the mask is continuous and free of any gaps or holes.

The downside of those operations is that they can reduce the resolution of the mask and give it unwanted properties (in the example of the opening operation, the mask lost its vertices). This means that the operations can lead to a loss of detail in the mask, making it less accurate. Thus, we suggest using those operations sparingly (too big a kernel size or too many iterations).

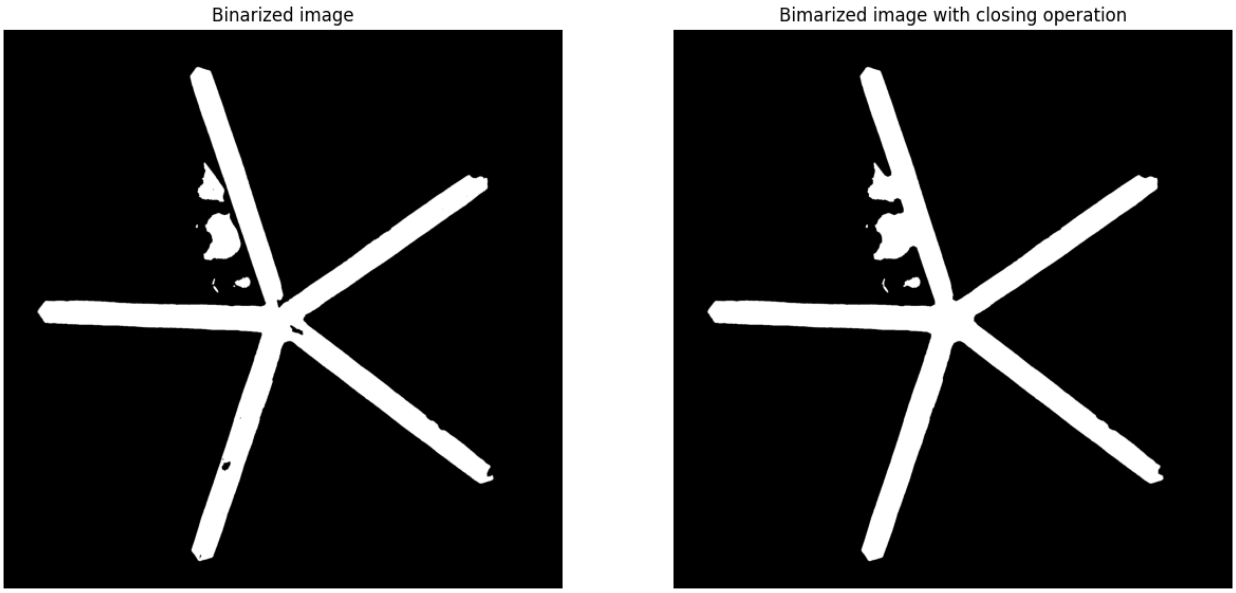


Figure 21 : The impact of closing operation (with the kernel size of 5 and 7 iterations).

III. 2 OpenCV for NP Identification and Size Calculation

III. 2. 1 Object Extraction

Once we obtain the binarized image, we enter the crucial stage of our algorithm: image segmentation. This pivotal step enables us to identify and analyze different nanoparticles. In essence, this part of our algorithm is responsible for counting the number of nanoparticles, extracting each one, calculating the area of each nanoparticle to deduce its size, and ultimately determining its morphological form.

We notice that the geometrical shapes of nanoparticles in HRTEM/TEM clichés are mostly polygons since they grow following some privileged directions. When tackling this problem, we mainly worked with spherical and square nanoparticles, the easiest and most common cases. The first technique we used was Contour Detection provided by the OpenCV2 library. Based on the gradient image, the technique helps determine the tangent lines of the gradient vectors, thus deducing the border of each mask. Despite being the most common technique for image segmentation, we soon found that the information provided by this method was far from practical in our project. However, we found reassurance in the practicality of another primary method for image segmentation, Labelling Connected Components. This method, requiring a background already defined in the previous stage, connects neighboring mask pixels in touch to create a region numerated with an integer, proving its effectiveness in our nanoparticle analysis.

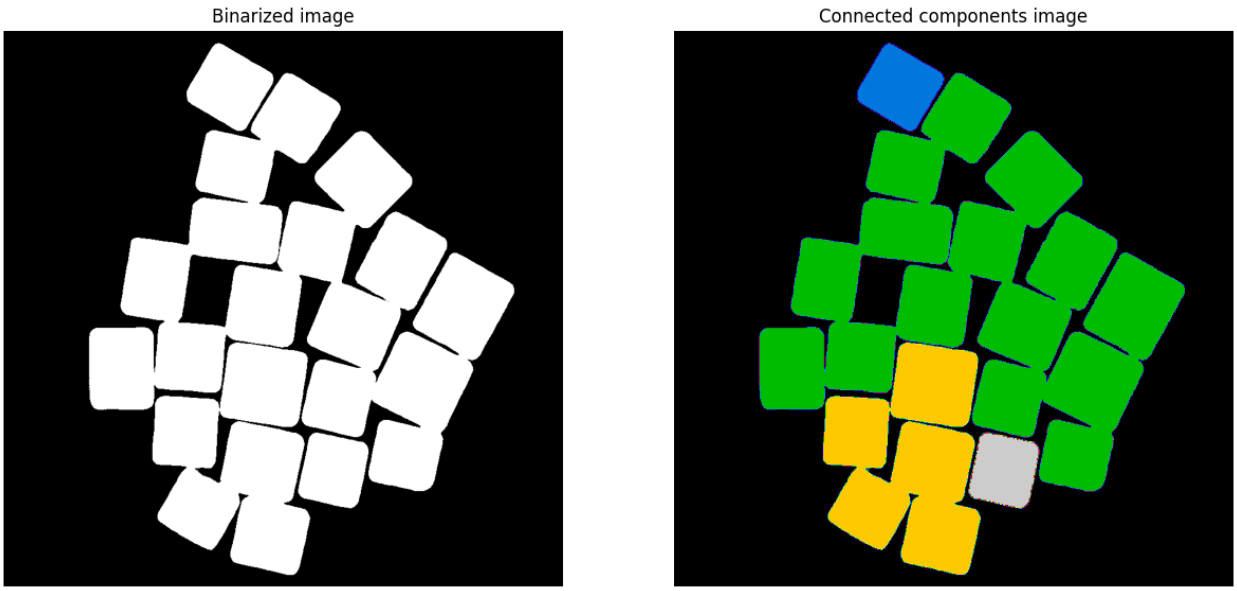


Figure 22 : Example of using Labelling Connected Components method.

Fig. 22 illustrates the action of the Labelling Connected Components method. This method is beneficial in segmenting regions of interest, as it connects neighboring mask pixels in contact to create a region labeled with an integer. As shown in the figure, the method successfully segments four regions, each labeled with a different color. Using each region as a mask, we can easily extract nanoparticles from the original image, eliminating unwanted information such as background noise. The extraction of the first nanoparticle (in blue) is shown in Fig. 23.

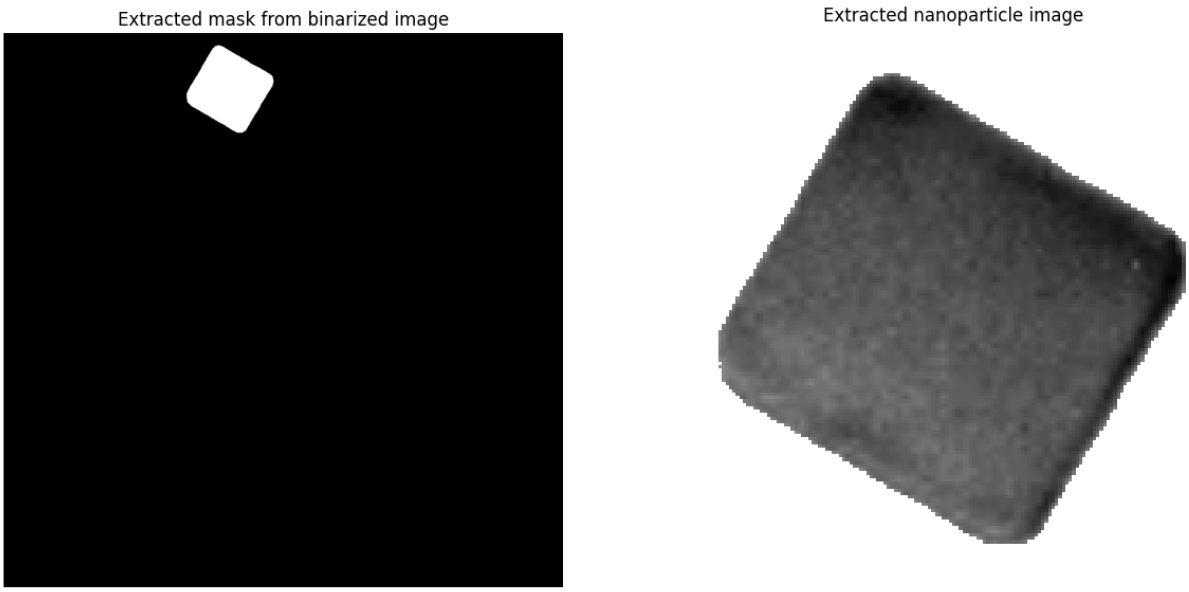


Figure 23 : Extracting the mask and using it to extract a single nanoparticle.

III. 2. 2 Nanoparticles Segmentation with Watershed Algorithm

III. 2. 2. a Nanoparticles with Isotropic Shapes

The challenges of Labelling Connected Components and Contour Detection are well-known: they struggle to segment overlapped nanoparticles, whether due to coalescence or their proximity. Addressing this issue was the primary and most demanding task in our project. From the inception of raw ideas to their conceptualization, execution, and algorithm optimization, the process spanned over half a semester, underscoring the complexity of the task.

We did much research (from the internet, articles, etc.) to conceptualize our ideas. We tried to construct an algorithm from these ideas. Then, we tested our algorithm in many cases, each with different hindrances and problematics, to see if our algorithm still has limits. If our algorithm does not work, we identify the problem, optimize it, and then re-test it with other cases. We kept using this recursive method until our algorithm worked and became reliable in most cases. During this time, we have come up with many ideas such as Random-walk Distance Transform, K-Means Clustering Segmentation, etc.; however, we estimated that building algorithms from these mathematical theories would take a long time, and nothing can ensure they will work. Therefore, in this project, we prefer to work with easy-to-manipulate methods and techniques mainly given by some libraries. So, instead of constructing a new mathematical tool, we can adapt our algorithm and code to these given functions. This way, we will have much more time to develop our algorithm while having complete control over its performance. Luckily, with these tools provided, our algorithm works quite well in most cases and helps obtain incredible results.

Our first algorithm, built on the foundation of the Euclidean Distance Transform technique and Watershed Segmentation with local extremum seeds, has been a resounding success, particularly with spherical and square nanoparticles. The principle of this algorithm is elegant in its simplicity: it constructs a distance map, with each pixel representing the distance value between the pixel in the original image and an obstacle defined by the technique. This distance map, when inversed, acts as a “basin” for the Watershed Segmentation algorithm. In the Euclidean Distance Transform map, each square or circle has only one local maximum, usually its centroid. By labeling these local maximums, we can determine the centroid of different square or circle objects in contact. Using these labeled maxima as “seeds,” the Watershed algorithm “fills” the “basin” with different labels. When many labels meet in the filling process, they create a watershed line that blocks the filling in that direction. This algorithm, born out of our relentless pursuit of a solution, has not only met but exceeded our expectations, delivering impressive results.

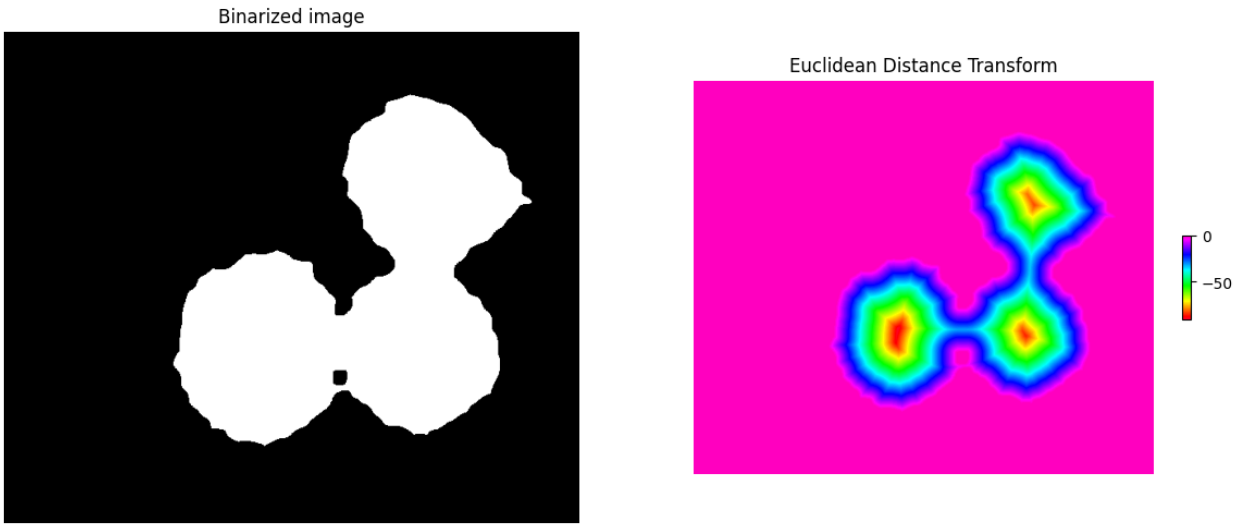


Figure 25 : Binarized image and its Euclidean Distance Transform.

As demonstrated in Fig. 24 and Fig. 25, the Euclidean Distance Transform method, a precise tool from the OpenCV2 library, allows us to identify three local extrema of 3 quasi-spherical nanoparticles quickly. This method calculates the distance between each non-zero pixel and its nearest zero pixel, characterizing it as a straight line. The geometrical properties of polygons ensure this line is perpendicular to the tangent line of the contour near zero pixel, creating a relief effect akin to a geological map.

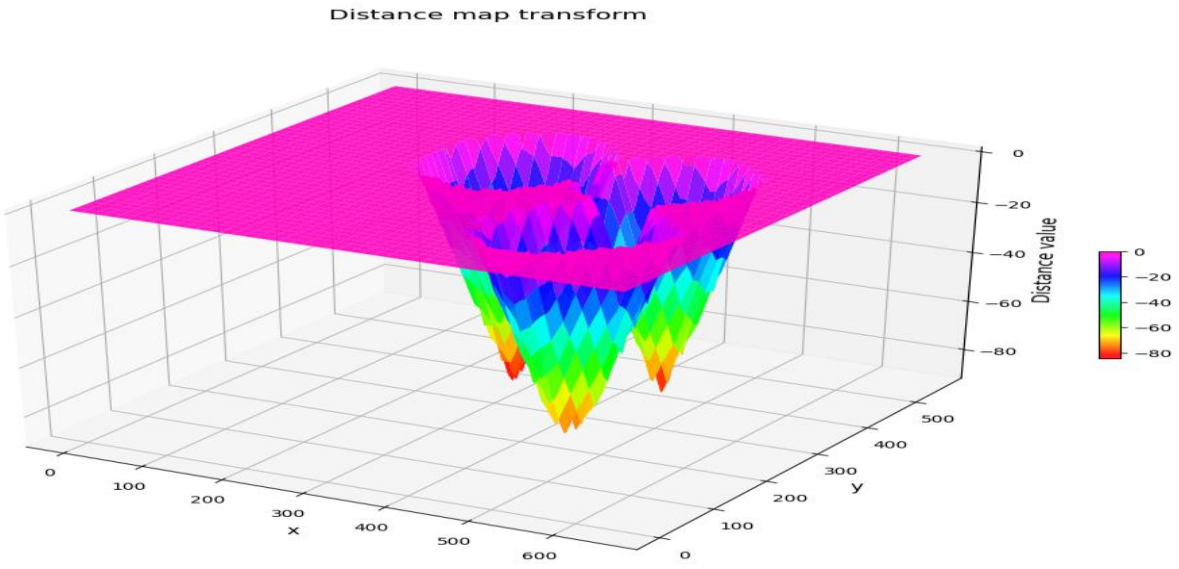


Figure 24 : "Basin" created by inversing the Euclidean Distance map used in Watershed Segmentation.

After obtaining the extremum, we use it as the "seeds" for the Watershed Segmentation function. We use the Watershed function provided by skimage library. This function needs three inputs:

1. The "basins" which can be obtained by inversing the distance map.
2. The "seeds" or the markers from which the "basin" is filled with different values.
3. The "mask" which determines the regions we want to fill.

In our algorithm, the binarized image is the "mask" for the Watershed function. The result of the Watershed segmentation is shown in Fig. 26.

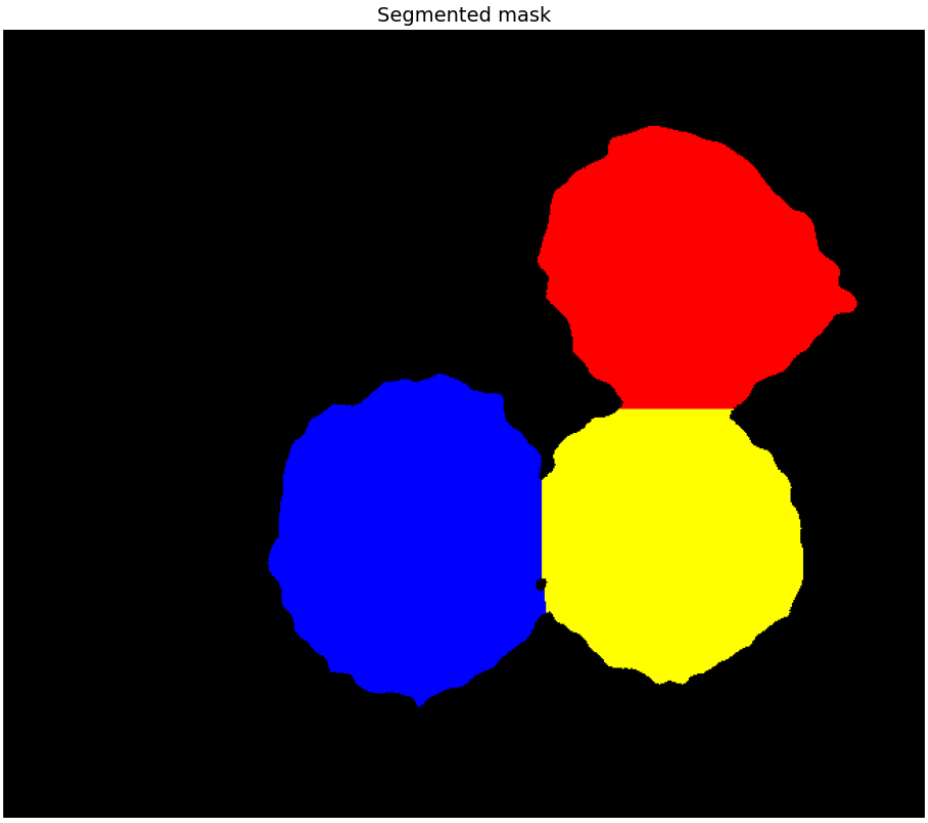


Figure 26 : Three regions determined by Watershed Segmentation.

III. 2. 2. b Nanoparticles with Anisotropic Shapes

Our initial strides have been promising, particularly with polygons and isotropic nanoparticles such as square, triangular, spherical, hexagonal, etc. Now, we embark on a new frontier, confronting the challenge of anisotropic nanoparticles (fig. 27). These particles grow in specific directions, assuming forms like the ellipse, rhombus, rectangular, etc. In the Euclidean Distance map, each anisotropic shape has more than one local extremum (2 extremum in case of ellipse, 1 line of extremum in case of rectangular). This complexity means that using local extremum as the 'seeds' for Watershed segmentation may not be a viable strategy, especially when there is more than one type of geometrical shape in the image.

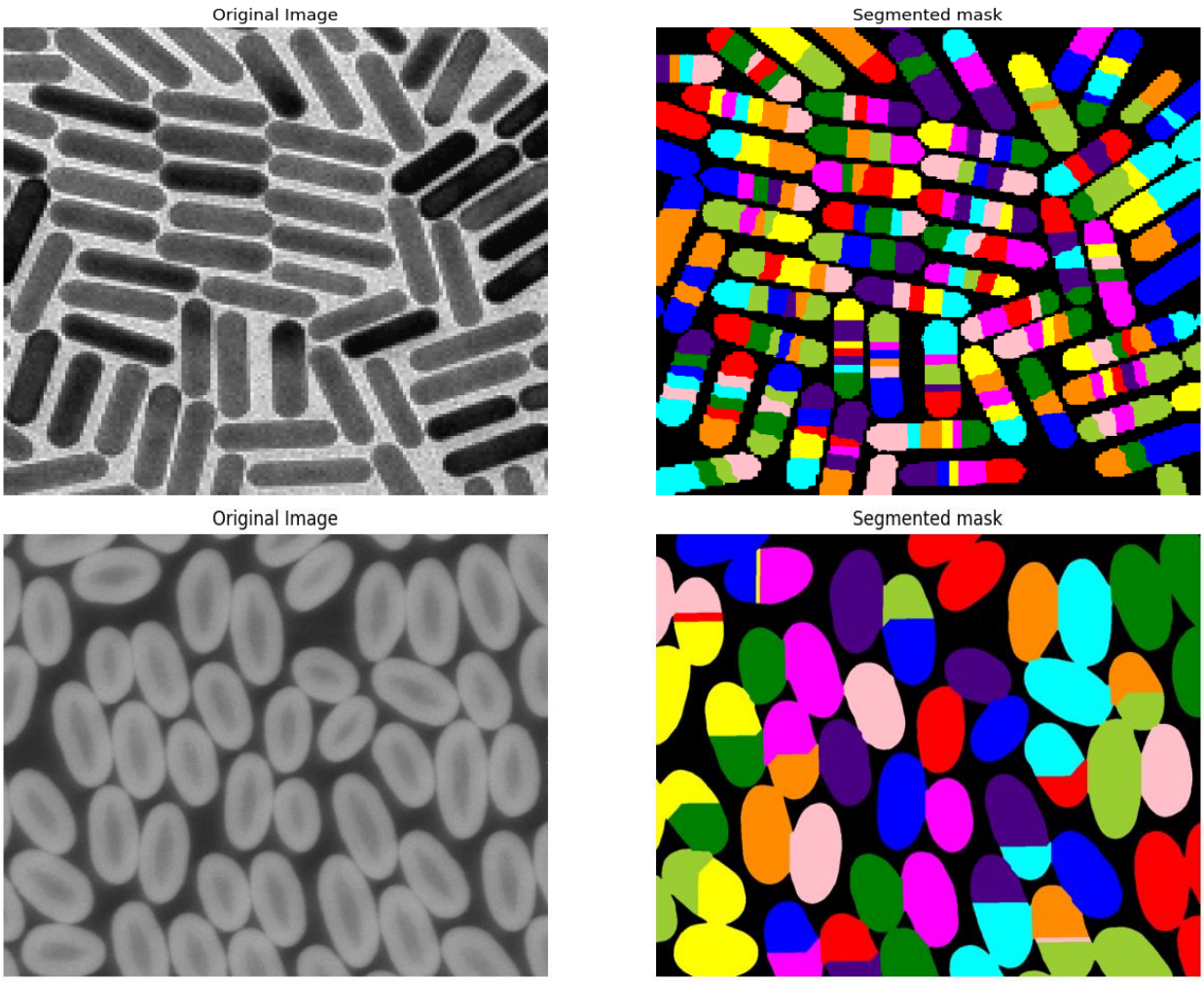


Figure 27 : Our algorithm fails when dealing with anisotropic nanoparticles, above: nanorods; under: ellipse.

This obstacle led to the development of a second algorithm, a significant improvement over the first. Like its predecessor, it is based on Euclidean Distance Transform and Watershed Segmentation. However, the 'seeds' for the Watershed function are obtained through a different method called Sure Foreground Extraction. In short, the 'seeds' are no longer a local extremum, but a region characterized by a clipped distance map. By applying a certain percentage of the maximum value in the distance map as a threshold, any pixel with a distance value more significant than this threshold becomes white (pixel value equal 255), and any pixel with a distance value smaller than this threshold becomes black (pixel value equal 0). After being labeled, the white pixels regions become the sure foreground and act as 'seeds' for the Watershed Segmentation function. The percentage of clipped value depends on the parameter *dist_max_threshold*, which, by default, is equal to 0,4 (the extraction threshold is 40% of the maximum distance value).

The Fig. 29 shows the clipped distance map from the Fig. 24:

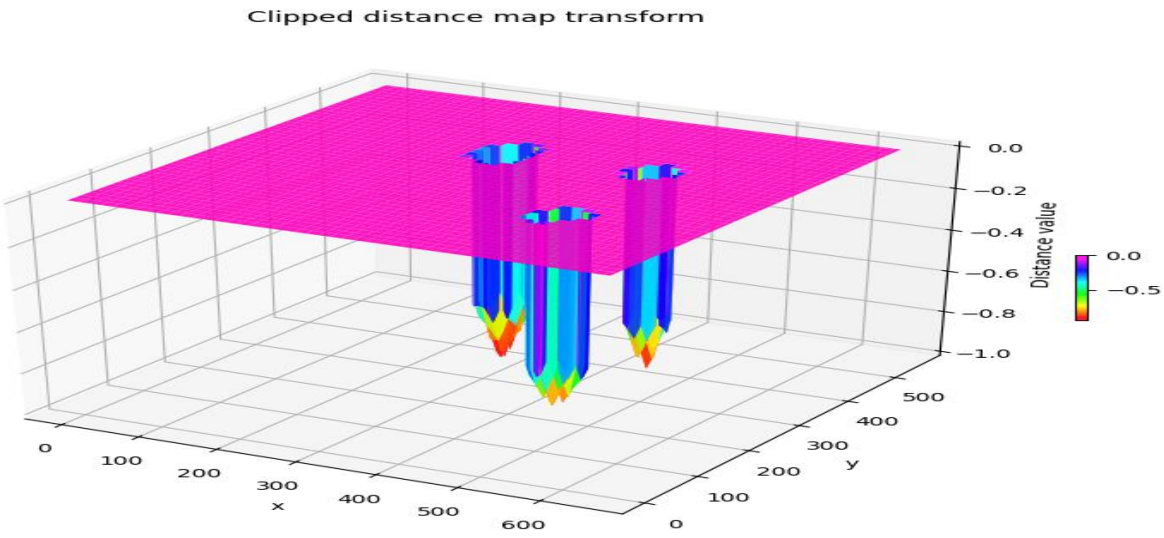


Figure 29 : Clipped distance map of the Fig. 15 with dist_max_threshold = 0.7.

Three regions have been separated from each other. The whole process is shown in Fig. 28.

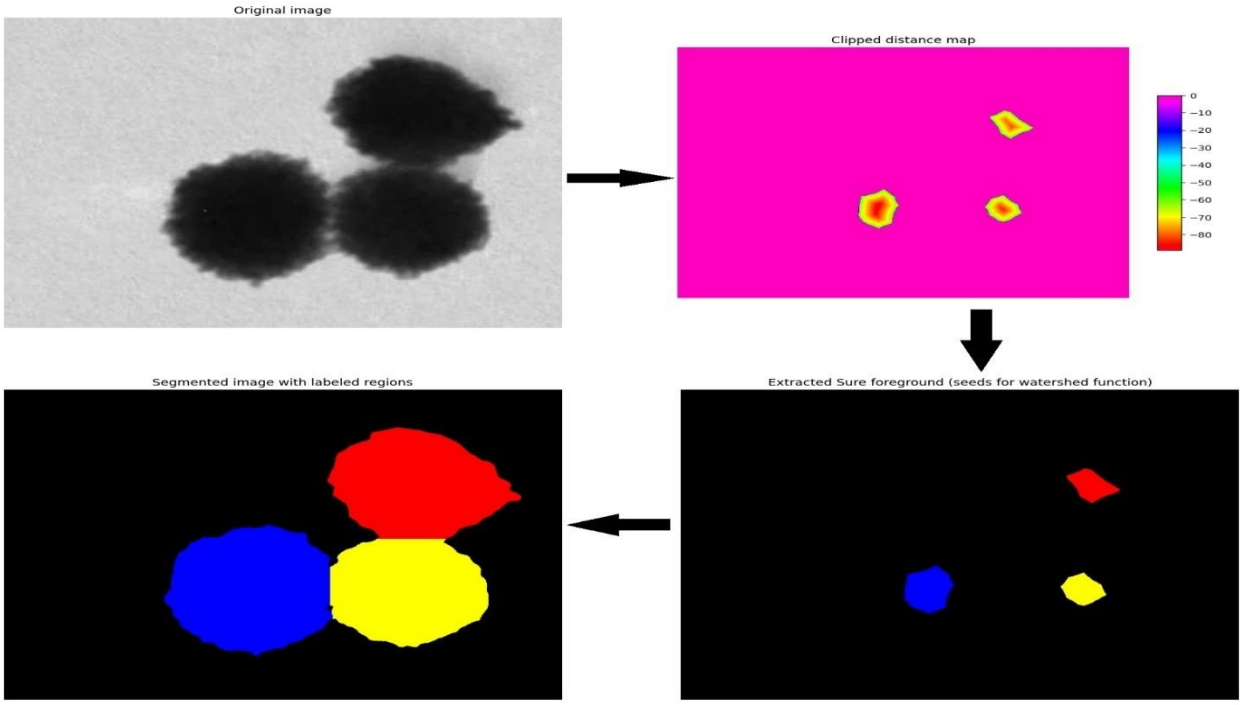


*Figure 28 : **The whole process of the nanoparticle's segmentation algorithm.***

This new algorithm empowers us with better control over the segmentation process. Visually, based on the extracted sure foreground, we can easily deduce the algorithm's performance. By adjusting the algorithm's parameters, we can observe their influence on the process and optimize them to suit different scenarios.

With the application of this new algorithm to the previous images and a careful selection of parameters, we have successfully overcome the challenges presented in Fig. 17.a and Fig. 17.b. The results, as shown in Fig. 30, demonstrate the algorithm's ability to segment anisotropic nanoparticles effectively.

Please note that regions with the same color are not considered one region. This visual problem is because the function we use to label these regions, *skimage.color.label2rgb*, only has a limited color panel. For example, the color panel with nine colors will label a region enumerated by an integer *i* and a region enumerated by an integer *i + 9* with the same color.

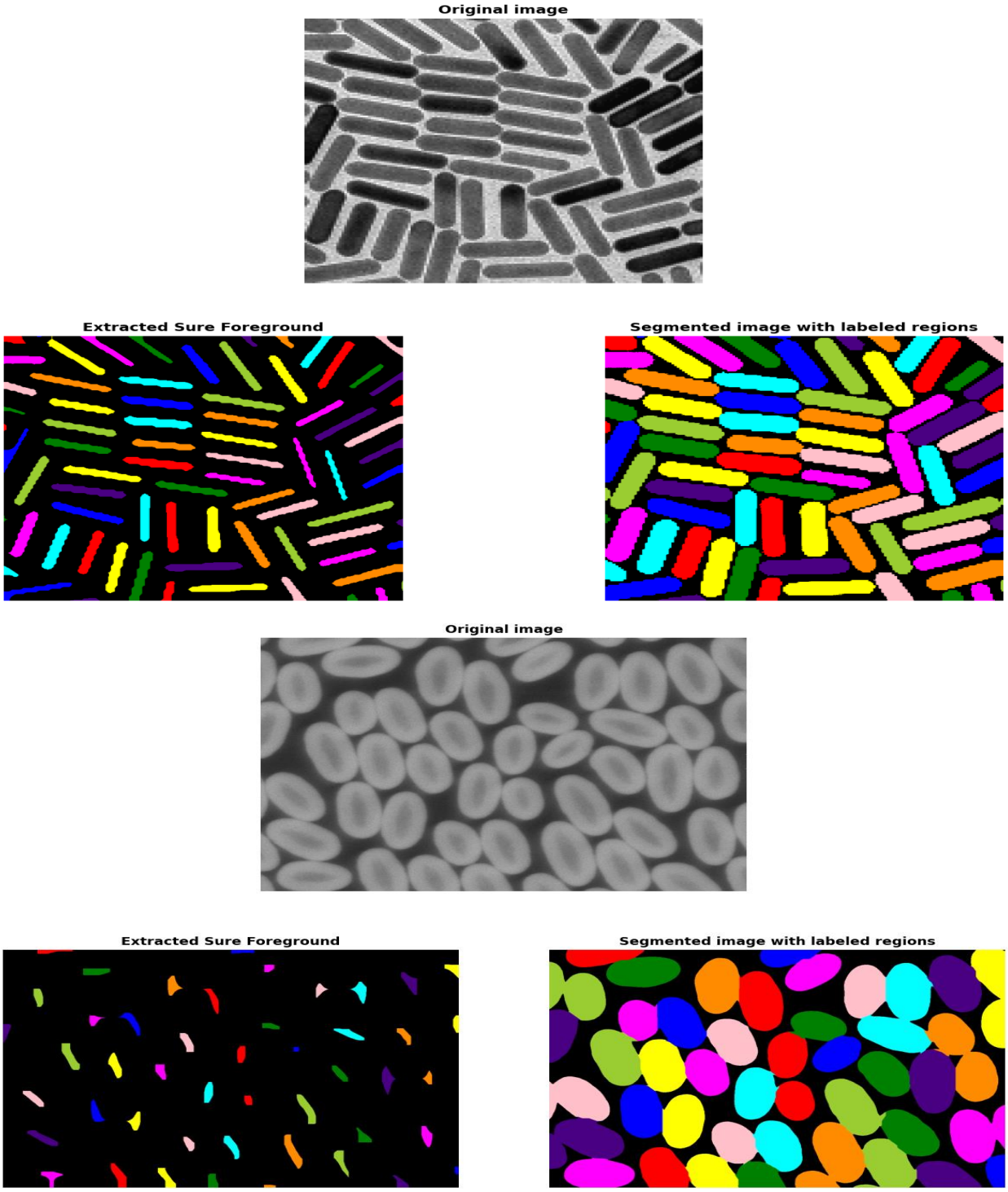


Figure 30 : Applying our second algorithm on the example shown in Fig. 27: above: nanorods; under: ellipse.

III. 2. 2. c Nanoparticles with Families of Various Sizes

We soon noticed the second algorithm's weakness when we used it on images with families of various sizes.

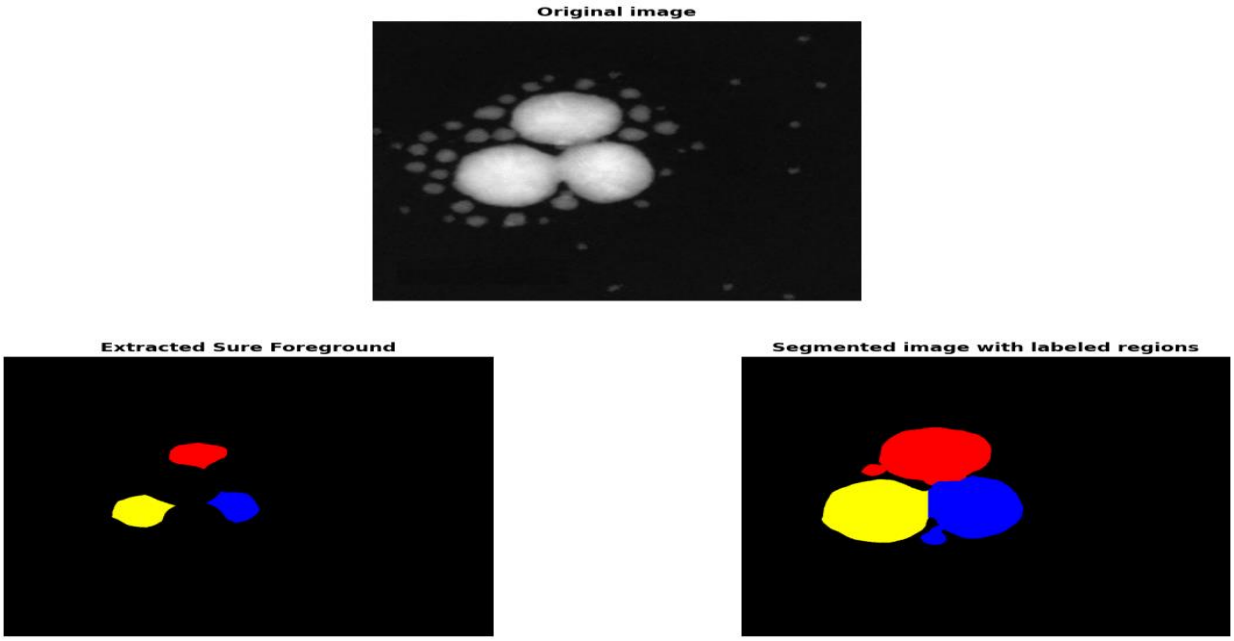


Figure 31 : The algorithm can't detect the families of nanoparticles with smaller sizes.

Since the value of the distance map depends on the size of the objects (distance between its center and the background), when clipping a percentage of the maximum value of the distance map, we unintentionally clip out the smaller objects.

This problem led us to the idea of normalizing the distance map. We define regions without coalescence beforehand using the Labelling Connected Components technique. Then, we normalize each region's maximum distance value to 1 and minimum value to 0. This algorithm can be seen as a combination of 2 previous techniques. Applying this new algorithm to the original image shown in Fig. 31, the results are shown in Fig. 32 and Fig. 33.

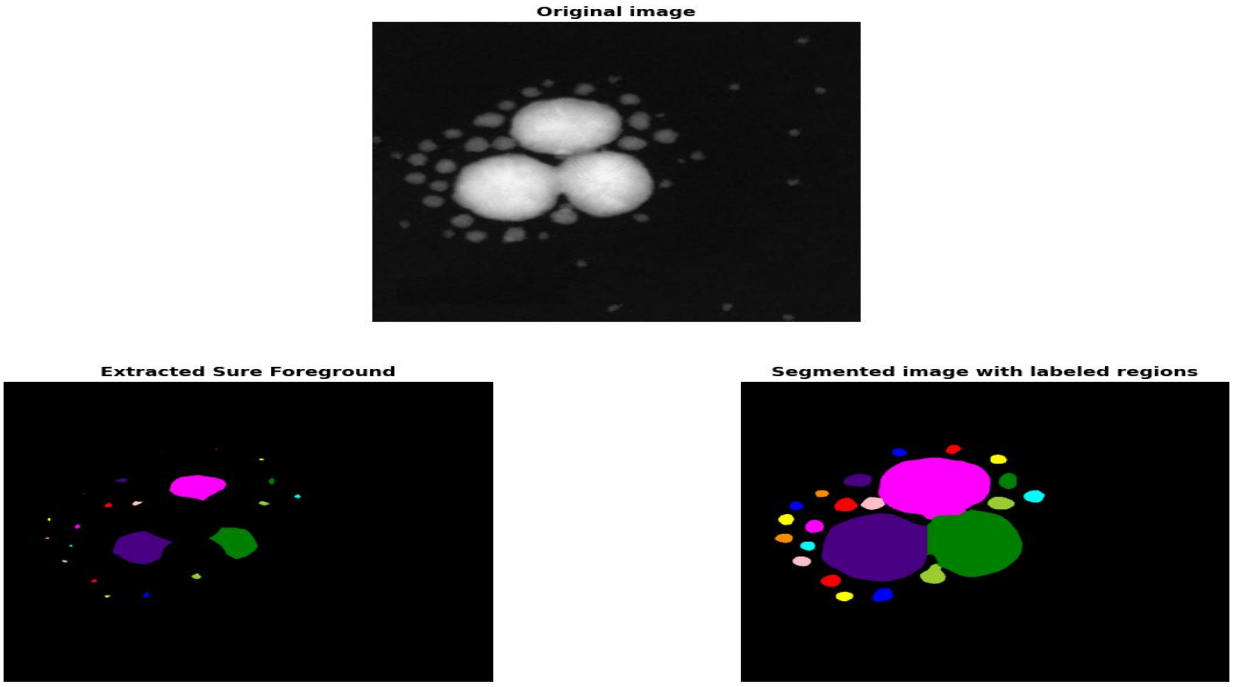


Figure 32 : The upgraded algorithm can now detect smaller nanoparticles.

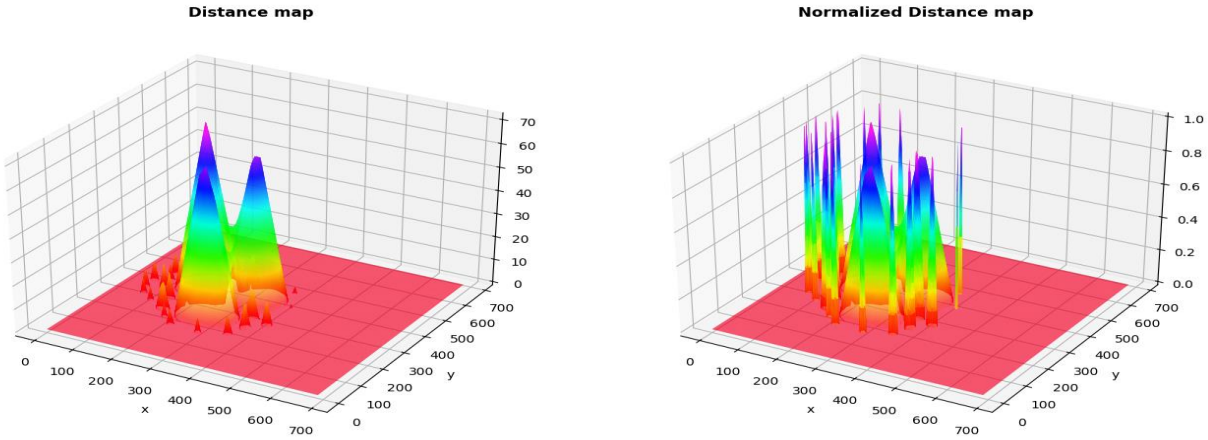


Figure 33 : Distance map and Normalized Distance map.

III. 2. 2. d Integration of Morphological Operations

In order to make our algorithm even more efficient and reliable, we added two more morphological operations at the beginning of the algorithm. These operations, called Morphological Eroding and Morphological Opening, significantly impact the algorithm's performance. Morphological Opening helps open local contacts while keeping the size of objects constant. The new operation called Morphological Eroding reduces isotopically the size of all objects. This operation has a significant impact on revealing the contact lines. The downside of this operation is that it can erase small objects when overused. Similar to other morphological operations, the impacts of those operations are determined by the size of the kernel characterized by *kernel_size* and the number of iterations, respectively characterized by *open_iter* and *erode_iter* (fig. 34).

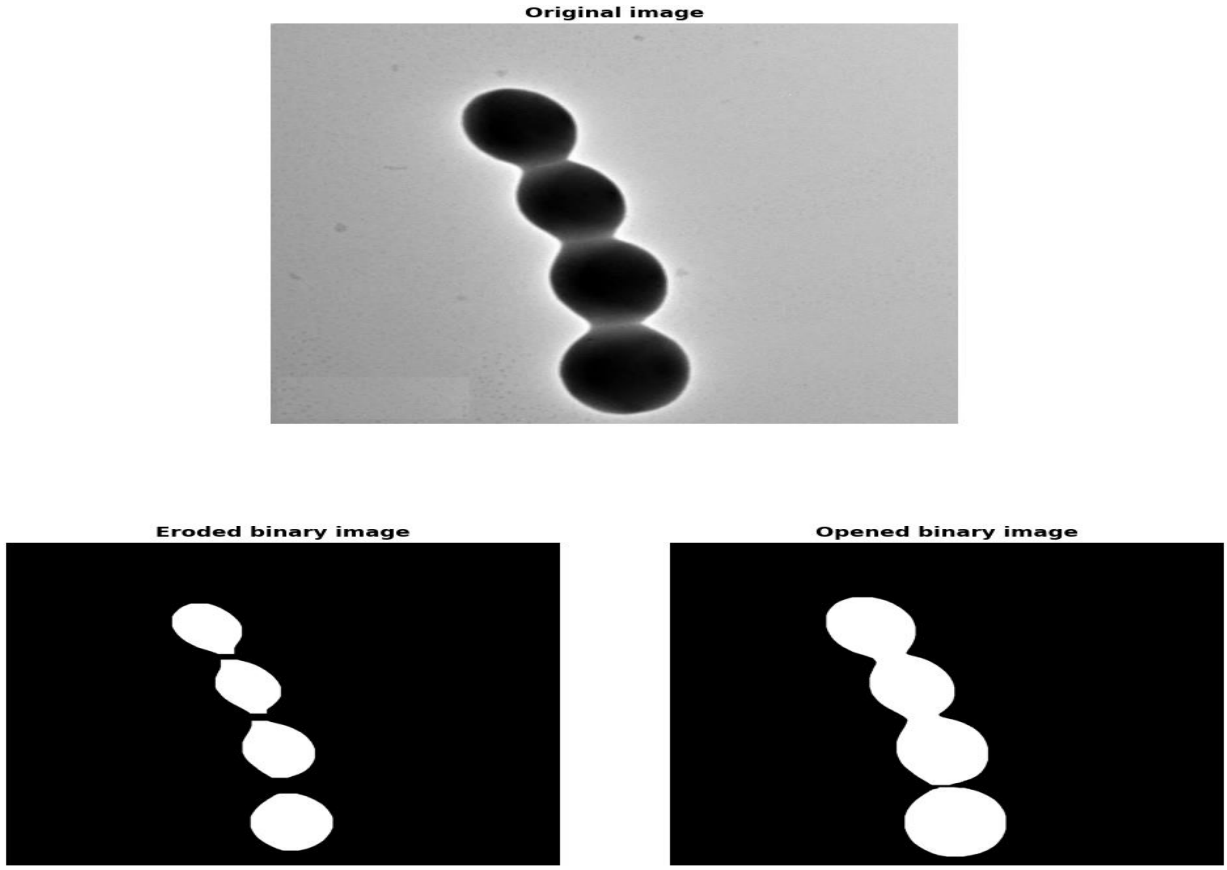
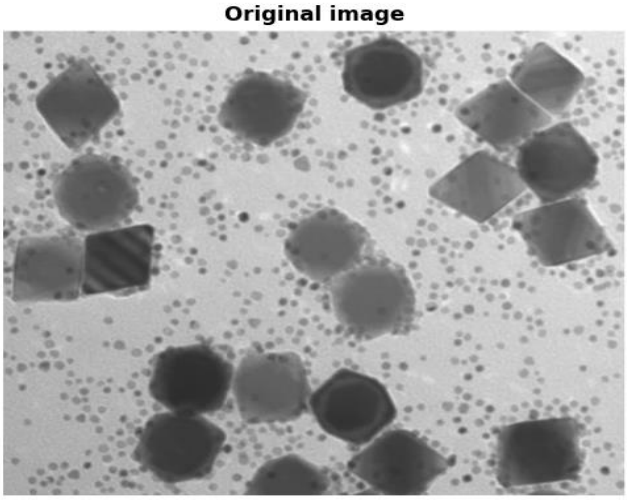
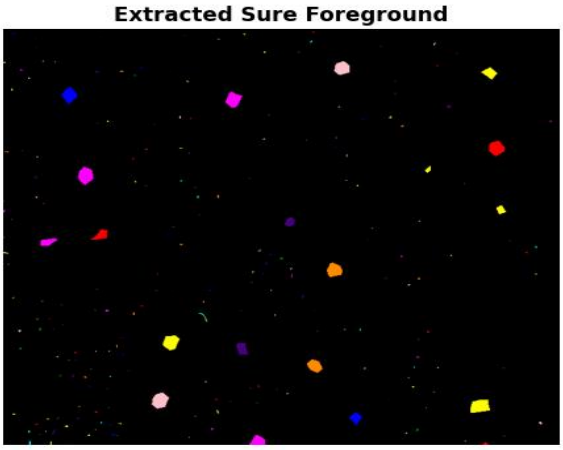


Figure 34 : The impact of Morphological Eroding and Morphological Opening, both with kernel_size = 3 and open_iter, erode_iter = 15.

III. 2. 2. e Limits of Algorithm

In summary, our state-of-the-art algorithm has solved the problem of detecting nanoparticles in coalescence with various geometrical forms, both isotopic and anisotropic. Still, we encounter a few problems when the nanoparticles are heavily overlapped and when facing the Ostwald ripening case. Based on Sure Foreground extraction techniques, our second algorithm can easily detect objects nearly the same size in contact. Therefore, when smaller nanoparticles are in contact with a bigger one, the algorithm will consider them as one object in the case of Ostwald ripening (fig. 35). Although this weakness can be neglected when constructing the size distribution, it can significantly influence the classification stage.





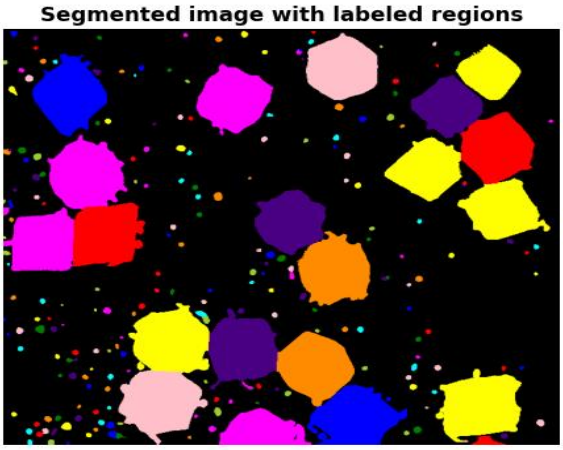


Figure 35 : An example in the case of Ostwald ripening. Our algorithm can't separate small nanoparticles in contact with the bigger ones.

III. 2. 3 Size Calculation

III. 2. 3. a Scale Bar Detection

The second stage's final step is easy and only takes a few weeks to complete. This step consists of three main tasks: acquiring the area of each nanoparticle, converting the size of one pixel into a micrometer or nanometer, and finally, constructing a size histogram from which to statistically deduce the size of nanoparticle families.

By simply counting the number of pixels with the same value, we can obtain the area of the region enumerated by that value. The areas obtained are in the unit of $pixel^2$.

The second task seemed challenging at first. Luckily, the experience of using the Contour Detection technique when we first developed the segmentation algorithm has helped us to build an algorithm to detect scale bars and return their length automatically. The first part of the algorithm is the same as the one we use for segmentation, except that we do not work with binary images in this algorithm. Instead, we use the Canny Edge Detection function given by

OpenCV to detect edges in the image. This function finds the tangent lines of the gradient vectors in the gradient map. Next, the *findContours* function, also provided by OpenCV, finds contours made from these detected edges. Finally, assuming the scale bar has the largest area, we sort the contours' area to find the corresponding contour. The algorithm will deduce the length of the scale bar by simply calculating the number of pixels in the horizontal axis. The scale bar must be the image's most prominent object to use this function. This problem can easily be solved by cropping out the part containing the scale bar with any tool available. Fig. 36 illustrate the results of our algorithm on two different scale bars.

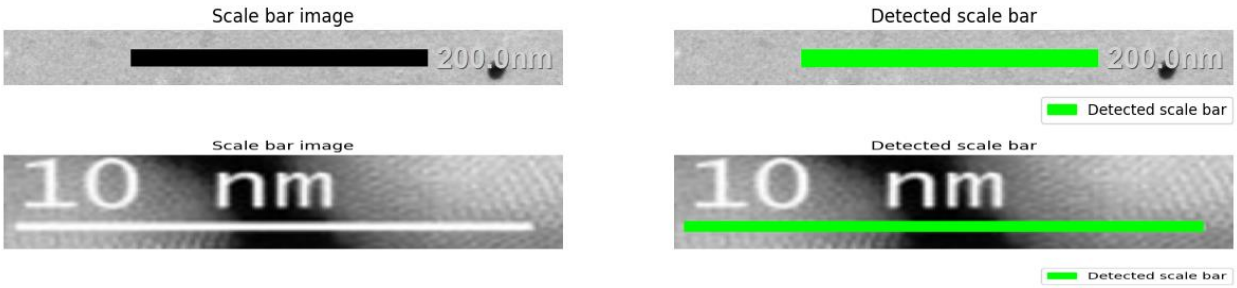


Figure 36 : Usage of detect_scale_bar function, above: 200 nm, under: 10 nm.

Our function can only return the bar's length in pixels. Its physical length must still be entered as an input (in nm). The function will then calculate the ratio nm/pixel, which is the physical length of one pixel.

III. 2. 3. b Size Histogram and Fitted Parameters

Moving on to the third task, experiences, and knowledge obtained in the third-year course "Modélisation Statistique" allow us to solve this task quickly. The two previous tasks provide a data set from which we can build a histogram using *numpy.histogram* function is given by the NumPy library. The constructed histogram typically has the form of a Gaussian distribution due to Stochastic Errors. Therefore, to obtain the wanted measure value, we fit our histogram with a Gaussian curve using the *scipy.optimize.curve_fit* function provided by the scipy library. This algorithm's accuracy is based on the choice of the number of intervals characterized by the parameter bins. Fig. 38 shows the results obtained on an image with two families of nanoparticles.

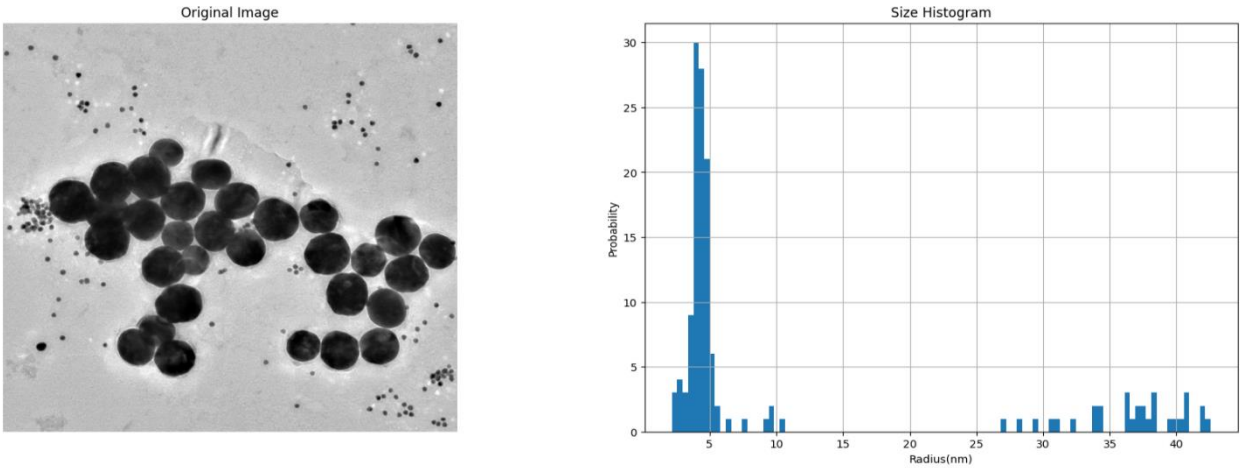


Figure 38 : Contruction of a histogram with 2 nanoparticle family.

We also developed a function that divides the histogram so that we can work with just one family. The results are shown in Fig. 37.

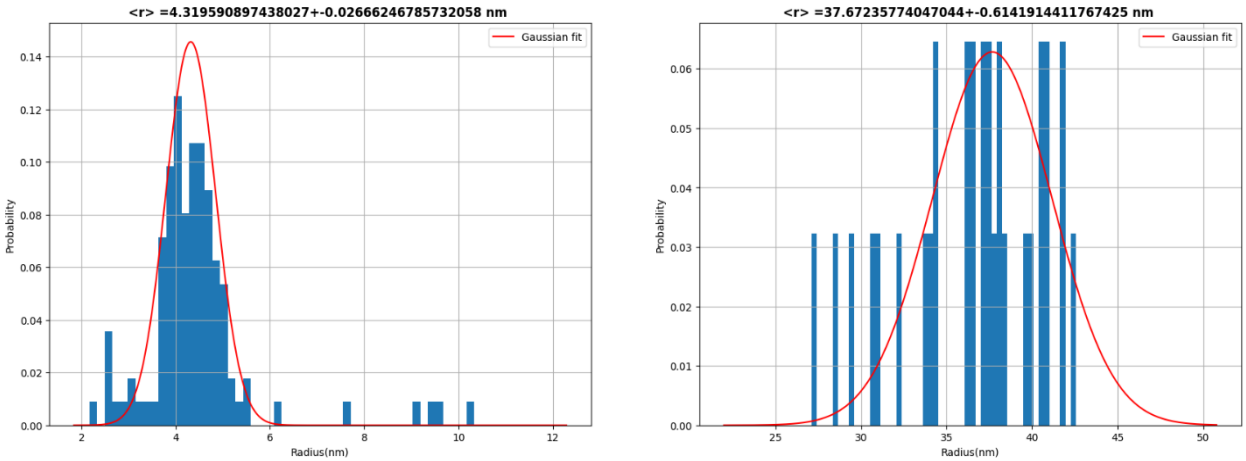


Figure 37 : Fitted histogram for each nanoparticle family and its measured size.

III. 3 Machine Learning for NP Morphology Classification

III. 3. 1 CNN Model classifying 3 Geometrical Shapes

III. 3. 1. a Our First Trained Model

After extracting each nanoparticle for the image, we must identify its morphological form. The algorithm for this task is a function or tool that takes input as an image and returns an output as labeled classes. CNN, a type of deep learning model, perfectly suits this requirement. Therefore, in this stage, we build a Deep Learning model for image classification using architectures and techniques provided by *TensorFlow*. This model will help us accurately classify the morphological forms of the nanoparticles.

The first model we built is shown in the following Figure 39.

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 64, 64, 3)	0
conv2d_12 (Conv2D)	(None, 64, 64, 8)	392
conv2d_13 (Conv2D)	(None, 64, 64, 8)	584
max_pooling2d_6 (MaxPooling2D)	(None, 32, 32, 8)	0
dropout_10 (Dropout)	(None, 32, 32, 8)	0
conv2d_14 (Conv2D)	(None, 32, 32, 16)	1,168
conv2d_15 (Conv2D)	(None, 32, 32, 16)	2,320
max_pooling2d_7 (MaxPooling2D)	(None, 16, 16, 16)	0
dropout_11 (Dropout)	(None, 16, 16, 16)	0
conv2d_16 (Conv2D)	(None, 16, 16, 8)	1,160
conv2d_17 (Conv2D)	(None, 16, 16, 8)	584
max_pooling2d_8 (MaxPooling2D)	(None, 8, 8, 8)	0
dropout_12 (Dropout)	(None, 8, 8, 8)	0
flatten_2 (Flatten)	(None, 512)	0
dense_6 (Dense)	(None, 64)	32,832
dropout_13 (Dropout)	(None, 64)	0
dense_7 (Dense)	(None, 64)	4,160
dropout_14 (Dropout)	(None, 64)	0
dense_8 (Dense)	(None, 3)	195

Figure 39 : Our very first model.

This architecture is pervasive for image classification, and this model was inspired when we were doing TensorFlow's tutorial. The output layer returns a vector with three components. The classes for this model were Circle, Square, and Triangle, respectively, labeled by the output vector's components. In the context of nanoparticle identification, these classes represent

different morphological forms that the nanoparticles can take, and our model is designed to classify the nanoparticles into these forms.

The data set we used for this model, taken from Kaggle (a free-to-use database), is similar to the following Fig. 40. It consists of training data for 2005 images and testing data from 406 images. The images in the data set are diverse, representing various morphological forms of nanoparticles. The classes are distributed evenly, with each class (Circle, Square, and Triangle) having roughly the same number of images.

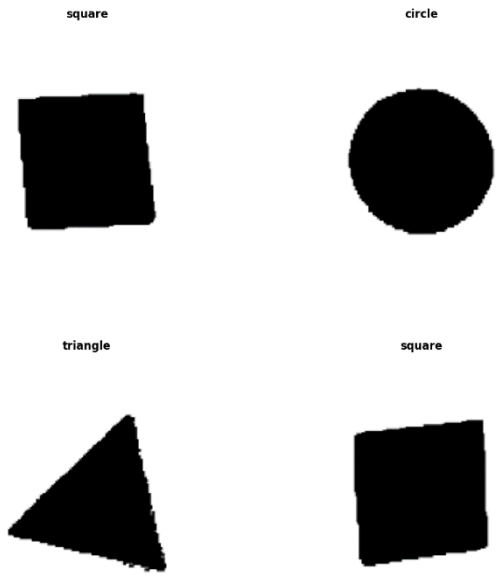


Figure 40 : Dataset used to train the previous model.

For the training process, we also applied the Image Data Generator technique provided by TensorFlow to increase our size and add more complexity. This function creates more virtual images for the training process by rotating with a random angle, rescaling the pixel's intensity, cropping the images, etc. The images are inputted with RGB color format. We take batch size = 32 for training parameters, use Adam as the optimizer method with default learning rate (1e-5), and then train our model for 50 epochs. The training process is coded on JupyterNotebook and done on our computer.

Although this is just an experimental version that we use to learn how the whole process works and the influence of each parameter, the result could be better and more transparent. This disappointment is a strong motivator for us to improve and achieve better results in the future.

While the overall accuracy reaches nearly 100%, the validation accuracy (accuracy of the model on the testing set) is only around 33% (fig. 41). The confusion matrix shows that every image in the testing set is classified as square. This performance could be more satisfactory for the nanoparticle identification task, as it indicates that the model cannot accurately classify the morphological forms of the nanoparticles. This underscores the need for further optimization of the model.

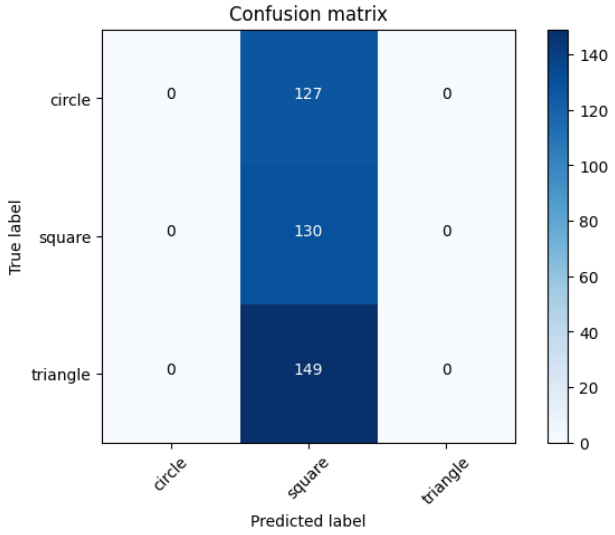


Figure 41 : Confusion matrix of the trained model.

At first, we thought this was some error in our code. However, after carefully checking the whole program, the results stayed the same. The only difference was that the classified label was changed for each training. Next, we tried working with training parameters such as batch size, number of epochs, learning rate, and even increasing our data set with more images, but the results were the same. This led us to suspect that the problem might lie in the model architecture itself, prompting us to dig deeper into the mathematical theory behind the CNN model and study how each layer and each parameter works.

The only thing left was the model itself. To identify the problem, we have dug into the mathematical theory behind the CNN model and studied how each layer and each parameter works.

III. 3. 1. b Understanding CNN Model and Architecture

In summary, the most critical factors for a model are:

- The first one is the data set. The number and sizes of images in the data set and their complexity (or quality) impact the training process most heavily. Here, the complexity is how images in the same class differ. The bigger the data set, the better the control we have on our model. More data allows us to build more complicated models without overfitting it. The complexity of the data set, with its diverse images, presents a significant challenge that we are striving to overcome. In short, it helps the model to be used in practical cases, not just on images similar to the data set.
- The second one is the number of filters contained in convolutional layers, especially the first one. The convolutional layers help extract the features of data. In CNN models, they are filtered images with different types of edge detections. These filter images or detected edges act as information for the model. The more filters the layers have, the more information the model can learn.
- The third one is the Dense layer at the end of the network. This layer network combines the high-level features learned by the convolutional layers and makes a final decision

based on those features. This Dense layer is a fully connected layer that connects every neuron in one layer to every neuron in the next layer. We can see in the example in Fig. 29 that the flattening layer helps flatten out the extracted features, making them turn into a vector of information. Then, each neuron of the Dense layer takes the value as a non-linear combination (decided by the activation function) of every extracted feature. This is why this layer creates a multiplicity of several features and a number of neurons (512 * 64 + 64 (bias of each neuron) = 32832 parameters). The more neurons the Dense layer has, the better the decision it can make.

- The last factor is the total number of parameters that must be trained. This number is proportional to the input size, number of filters, convolutional layers, and neurons in the Dense layer. If the number of parameters is too high compared to the number of images in the dataset, the model can easily be overfitted. Overfitting is a common problem in machine learning, where a model becomes too complex and starts to memorize the training data instead of learning the underlying patterns (fig. 42). This can lead to poor performance on new, unseen data, as the model cannot generalize from the training data.

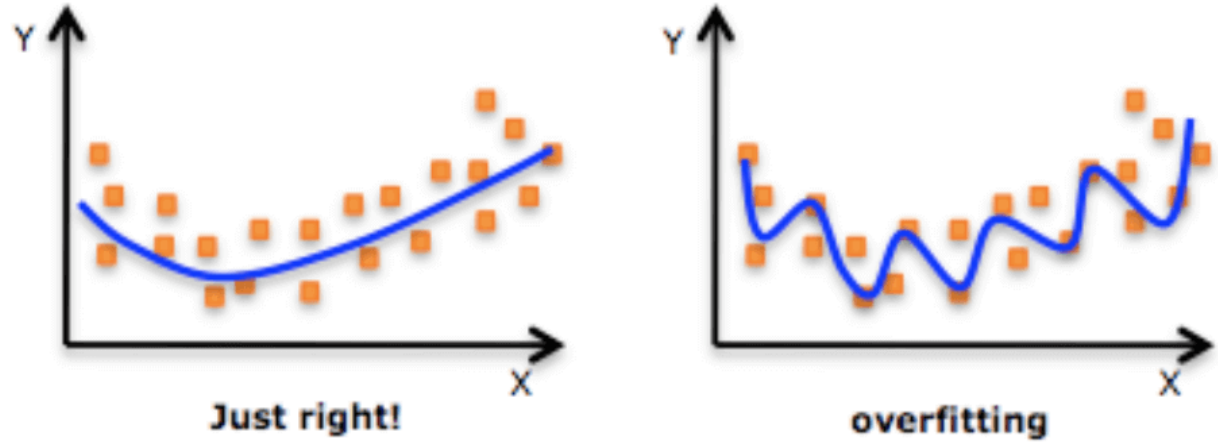


Figure 42 : The model is overfitted when the number of parameters is too high compared to the number of data points. [10]

III. 3. 1. c Updated CNN Model and Cloud Computing

In our first model, the total number of parameters must be higher. Meanwhile, the dense layer has many neurons, the convolutional layer's input size is too small, and the number of filters is too small. We then tried to increase the number of filters (32 filters in the first convolutional layer), and the input size increased from 64x64x3 to 224x224x1 with grayscale color format. Besides, our data set is relatively simple (Fig. 40). Therefore, using this architecture would be overkill and can easily be overfitted. The simplified and optimized model is shown in Fig. 33. In this version, we reduced the number of parameters, adjusted the number of neurons in the dense layer, and optimized the input size and number of filters in the convolutional layer to improve the model's performance.

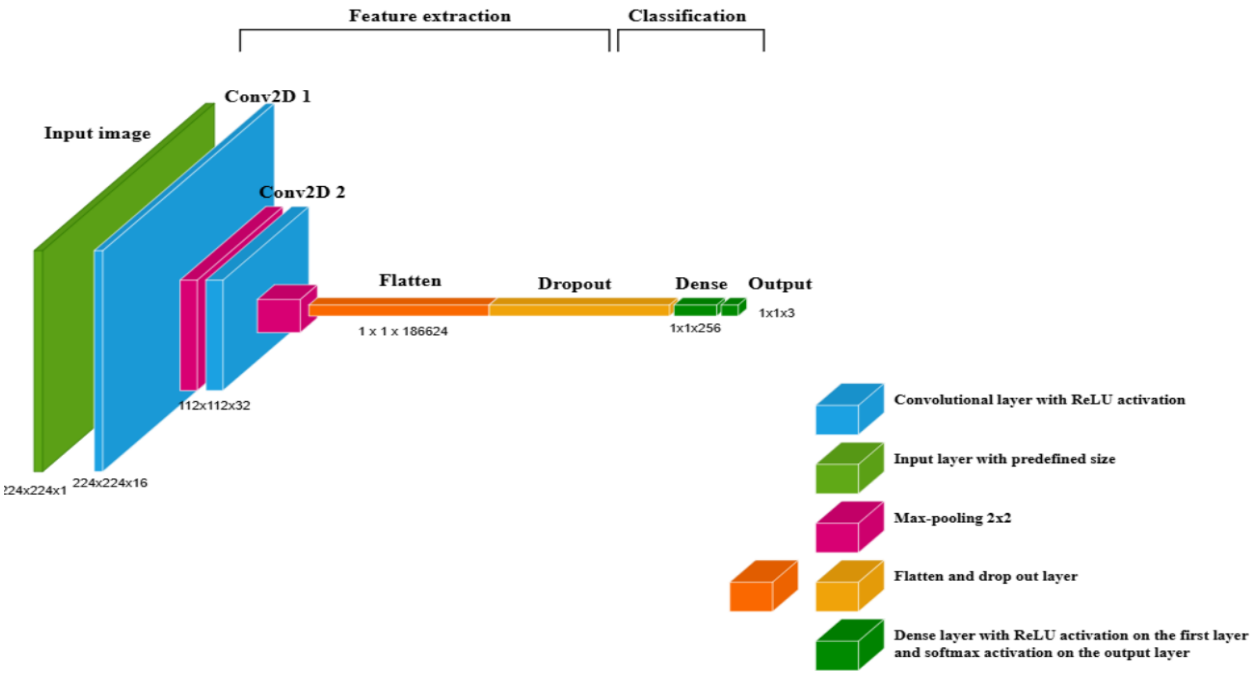


Figure 43 : The architecture of our first working model.

This model has 47.795.587 parameters that need to be trained (1,000 times more parameters than the previous model) and a simple architecture. Its complexity underscores the intricacy of the process we are undertaking.

The training begins with default training parameters and proceeds on our computer via JupyterNotebook. After halfway through the training process, we noticed that the training time was too long, and our computer started overheating. By checking our system, we can see that this model is way beyond the capacity of our computer. At first, we thought that TensorFlow used our GPU (GeForce GTX 1650, a relatively strong one) to train. However, we soon found out that, to give TensorFlow access to our local machine's GPU, we needed to do some setup via CUDA toolkits. Using them is challenging as they are deeply related to the operating system and computer architecture. Doing this may risk damaging our computer and ultimately delaying our project. The first solution was to use Clyde via a computer at LPCNO or even access the Olympe at CALMIP for a faster training session.

However, during the vacation, we discovered that people with weaker CPUs and GPUs can even train a heavier model (10 or even 100 times more parameters than ours) with a relatively low training time. The technique they were using is called Cloud Computing. This is an informatic service as users demand computer system resources or computing power from a distance without direct active management.

For this project, we are using the Cloud Computing service offered by Google via GoogleColab. With only a few clicks, we can access a potent computational unit, GPU Tesla T4, or even the state-of-the-art tensor processing unit, TPU v2, designed explicitly for training AI models. The free-to-use version only grants us ten computational units (a unit characterizing the time we can use the format) or around 7 hours when using GPU T4. The paid version costs

11€ and provides 100 computational units monthly, granting us access to even more powerful hardware. By our estimation, the first 20 accessible computational units combined with 100 paid computational units will be more than enough for this project.

Since Colab cannot access our local files, we must upload our dataset to Google Drive to use its GPU. At first, we noticed that the first epoch always takes an enormous time. After some research, we learned that Google Drive file routing is a complicated system, and to begin the training process, all the data must be transferred from the original file to the virtual file created for the coding environment. To solve this problem, we compress our data and unzip it in the virtual environment. The result and training time were far better than expected. The training session only took around 450 seconds for 50 epochs (around 9 seconds per epoch). Compared to the result obtained from last year, each training session took around 200 to 300 seconds for 25 epochs (around 10 seconds per epoch) with a ten times more extensive dataset. However, their models only have around 1/50 of our total parameters. The accuracy of our model is shown in Fig. 44.

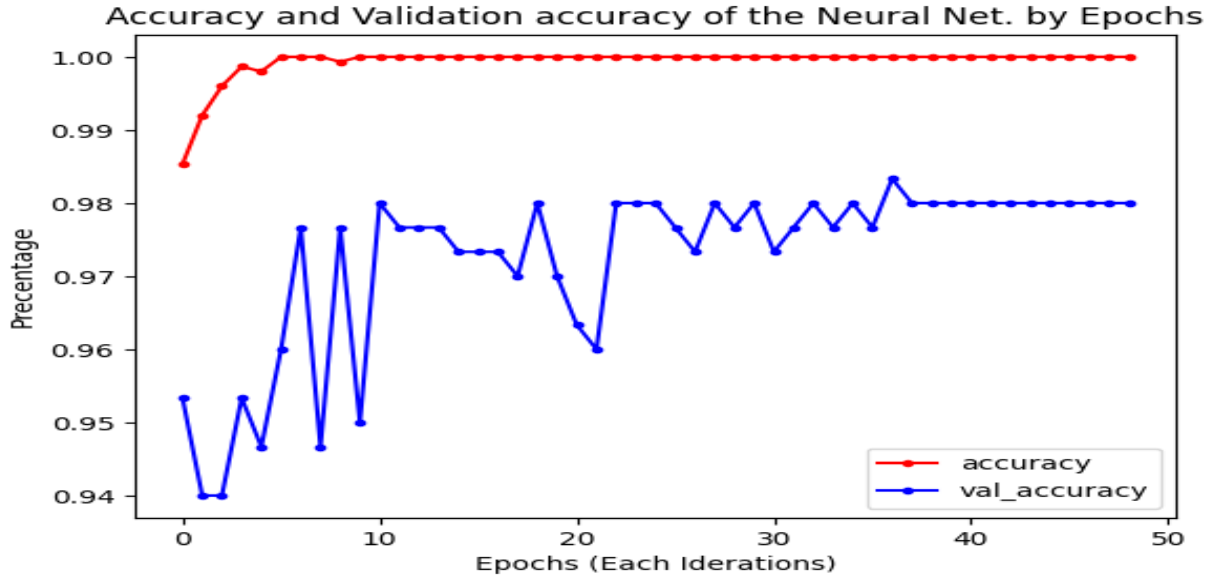


Figure 44 : Accuracy and Validation accuracy of our model.

As we can see, our model reaches 100% accuracy and 98% validation accuracy. The first problem with the model is now solved.

III. 3. 1. d Integration of Hand-Drawn Dataset

Even with a very high validation accuracy, the model failed when applied to practical cases. Since our dataset is too simple, the model can only work on images identical to the data set. This led to the development of the second model, which had the same architecture and was trained on a new dataset consisting of the previous one plus hand-drawn images (fig. 45). These hand-drawn images will allow our model to focus more on the border of the objects.

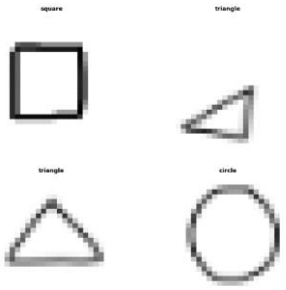
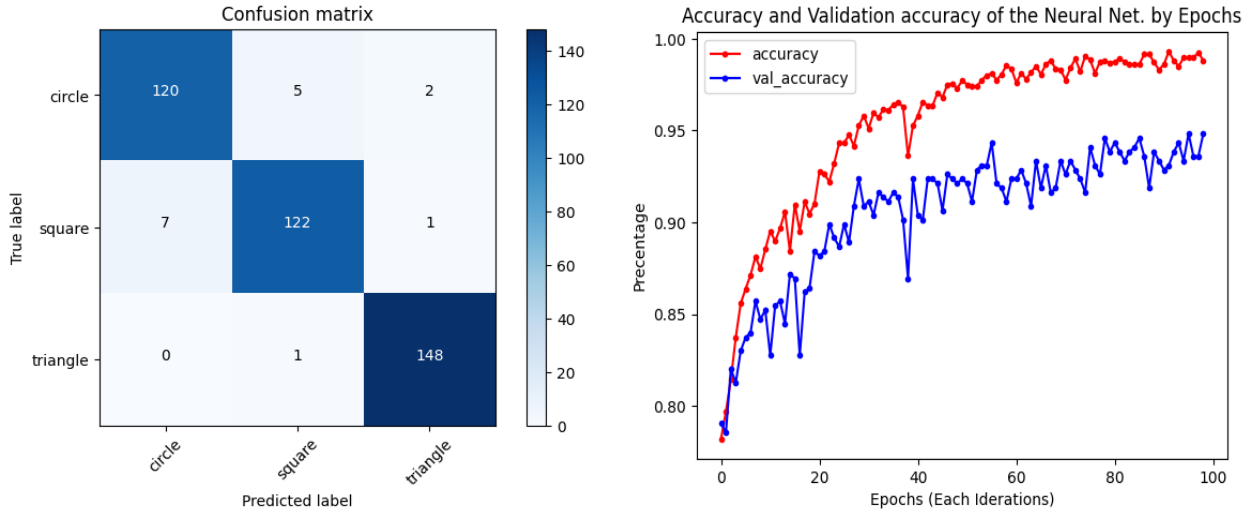


Figure 45 : New type of data added to our dataset.

With access to more powerful computational units, we improved our model with a bigger input size from 224x224x1 to 256x256x1. We also reduced the batch size from 32 to 16 (double the data to train) and increased the number of epochs to 100. Our model now has 63.000.000 parameters. The training proceeded using L4 Tensor Core GPU with high-RAM mode. The process with 100 epochs took around 1100 seconds (11 seconds for each epoch) (fig. 46).



a. ***b.***

Figure 46 : a. Confusion matrix of the new model; b. Accuracy and Validation accuracy.

This new model performs well in practical cases. Fig. 47 shows its integration into the whole algorithm when applied to the example in Fig. 16.

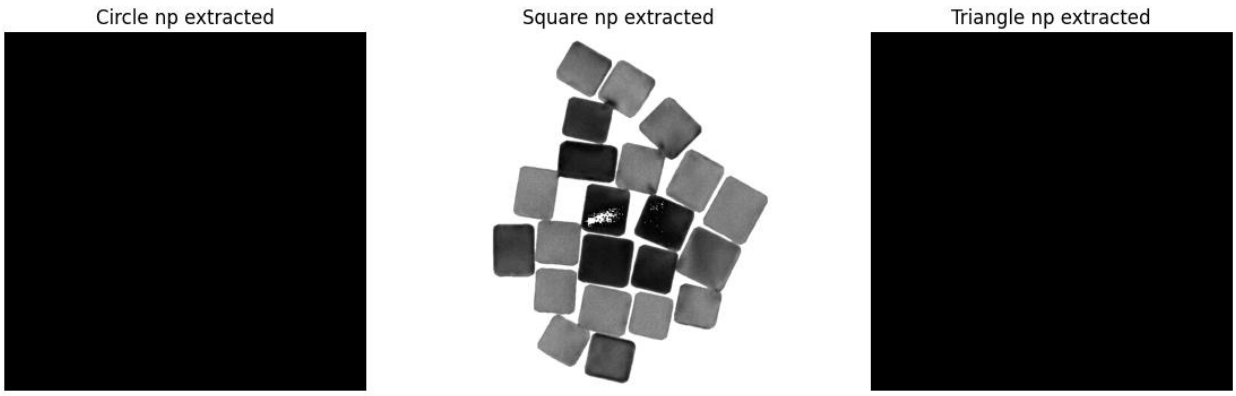


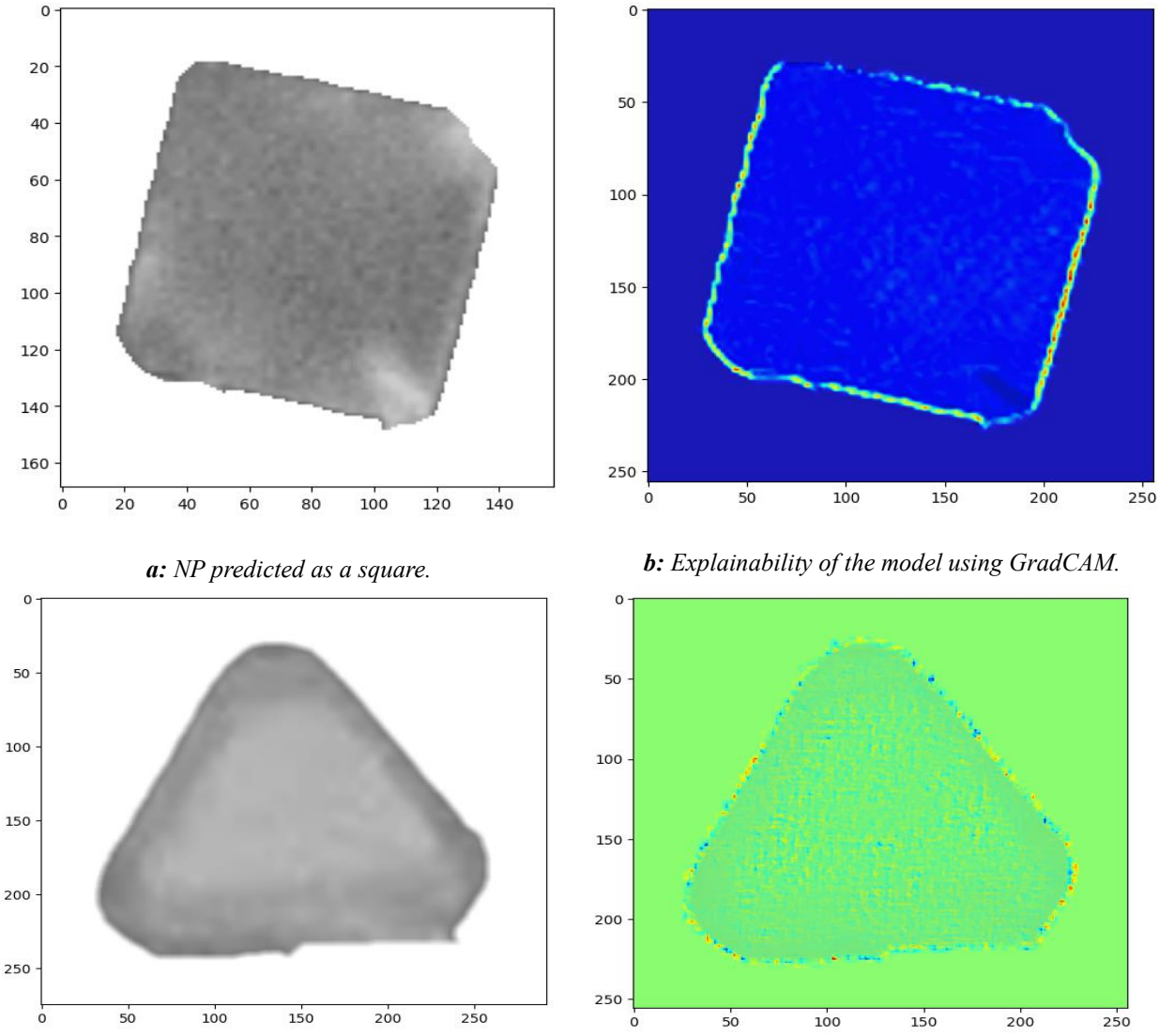
Figure 47 Integration of our classification into the whole algorithm.

III. 3. 2 Explainable AI with Xplique

For the next task, our tutors propose a technique called Explainable AI. This tool allows us to see how neural networks build their understanding of images by finding inputs that maximize neurons, channels, layers, or compositions of these elements. Luckily, we have contact with two fifth-year MA students who have just done a project with M. Simon Cayez on the classification of µXRD images. We learn about the Xplique toolbox, developed by IRT de Saint Exupéry.

We spent one or two weeks learning to use this toolbox. However, when applying Xplique to our model, we encountered a rarely seen error: "*AttributeError: The layer has never been called and thus has no defined input shape*.". Then, we spent the next week trying to fix this problem, but since Xplique is a newly developed library, only some forums discuss its usage and errors encountered. However, we soon found out from the tutorials that the model used in the example is a pre-trained MobileNetV2 provided by Keras. By checking its architecture, besides the number of layers and their function, we noticed that they are connected non-linearly rather than stacked one on another like ours. This type of architecture is only possible for Functional Models. We also learned that Functional models have a pre-defined input layer that perfectly suits the error. Acknowledging this, we built and re-trained a Functional model with the same architecture as our previous Sequential one.

Fig. 48 show the results of applying two different methods (Gradient Input and gradient CAM) to two different nanoparticles.



a: *NP predicted as a square.*

b: *Explainability of the model using GradCAM.*

c: *NP predicted as a triangle.*

d: *Explainability of the model using GradientInput.*

Figure 48: Explainability of the model.

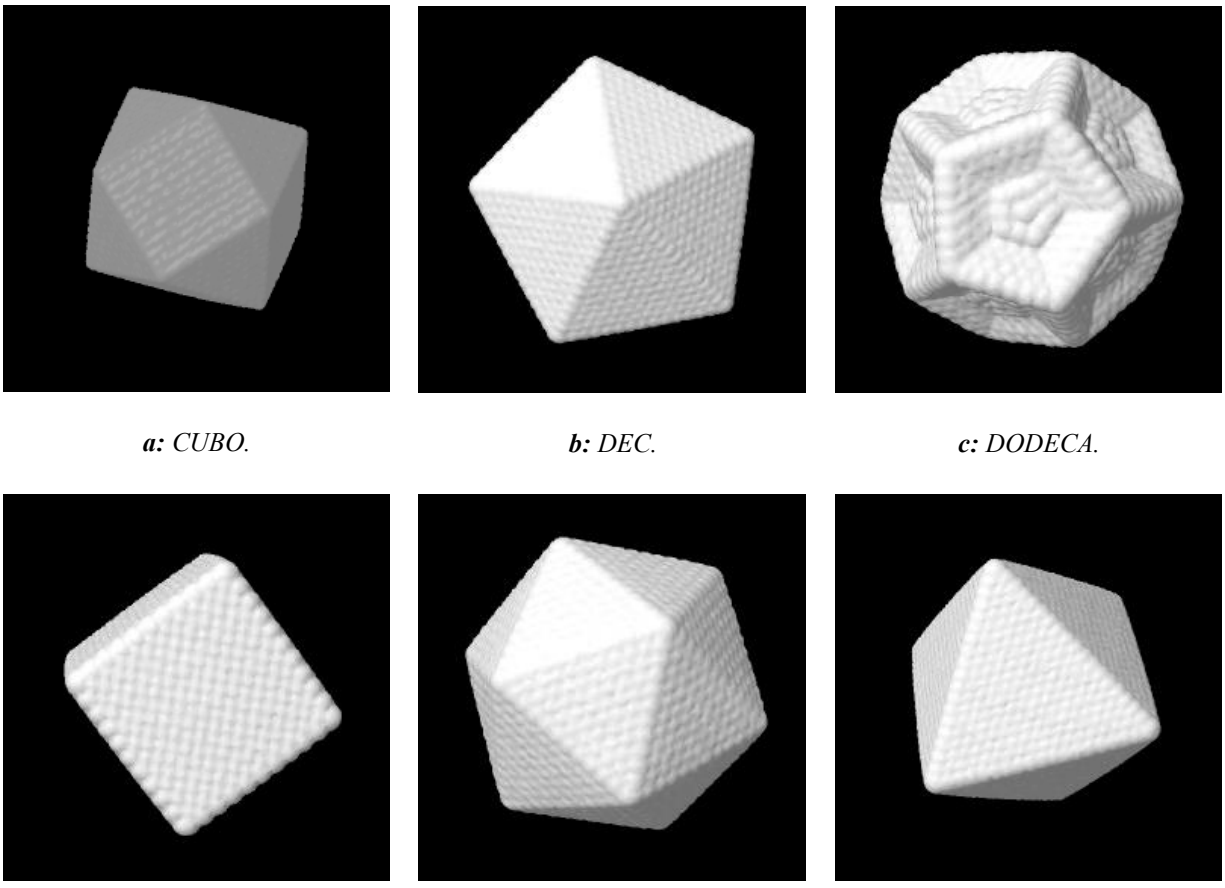
Using Explainable AI, we can see that adding hand-drawn images to our dataset helps the model focus more on the borders of the nanoparticles (pixels in red are considered the most critical pixels for the model's decision).

III. 3. 3 CNN Model for 6 Morphological Shapes

At this stage, we have completed the workflow for our AI model. Our job is to create a new model that corresponds to our tutors' demands. Unlike last year, when the aim was to build a model classifying HRTEM images with high resolution, our model has to work with images with lower resolution, where we cannot see atomic structures but facet effects.

Our next model works with six classes: cuboctahedra (CUBO), decahedral (DEC), dodecahedral (DODECA), FCC cubic (FCC_CUBE), icosahedral (ICO), and octahedral (OH).

The dataset used to train this model is an in-silico image generated by M. Romuald Poteau. The dataset is shown in Fig 49. Each image is generated by rotating a random angle around a fixed axis.

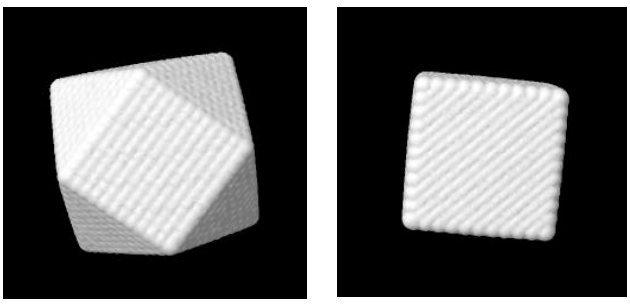


a: *CUBO.* ***b:*** *DEC.* ***c:*** *DODECA.*

d: *FCC_CUBE.* ***e:*** *ICO.* ***f:*** *OH.*

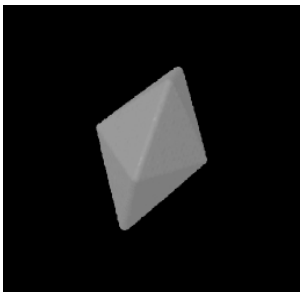
Figure 49: 6-classes in-silico dataset.

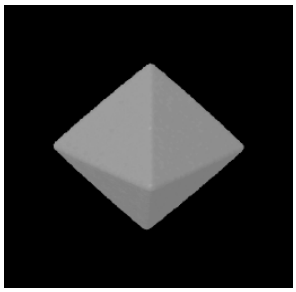
Our tutors propose this dataset because if we only depend on the outer borders of the nanoparticles, we can misunderstand some types of structure. Fig. 50 right below show an example of possible confusion between FCC_CUBE and CUBO and DEC and OH.



a: *CUBO can be mistaken for FCC_CUBE.*

b: *FCC_CUBE can be mistaken for CUBO.*



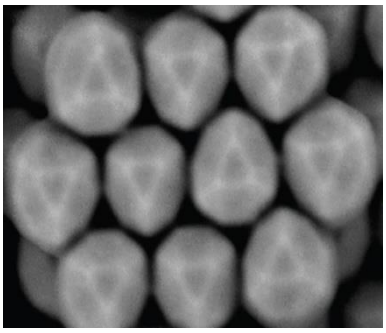


c: *DEC can be mistaken for OH.*

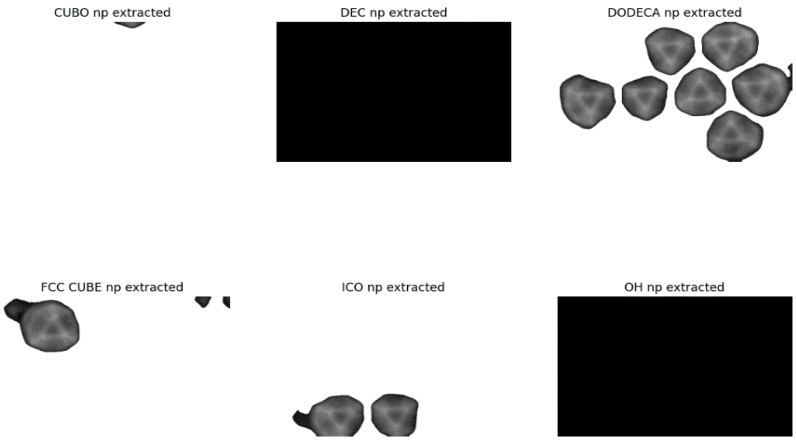
d: *OH, can be mistaken for DEC.*

Figure 50: The possible mistakes.

Initially, we only used this type of image to train our dataset. The accuracy and validation accuracy obtained are very high, around 99%. However, using this model still needs to be more practical. Since the CNN model is based on a pattern recognition mechanism, natural images with fewer details on edges, vertices, and facet effects can perturb the result.



a: *Original image of icosahedral nanoparticles.*



b: *Nanoparticles classified by our model.*

Figure 51: Prediction of icosahedral nanoparticles.

In the example in Fig. 51, we can see that icosahedral nanoparticles (identified by a triangle in the center) are confused with the dodecahedral class (probably due to the outer edges). Using Xplique, we can clarify our hypothesis (fig. 52).

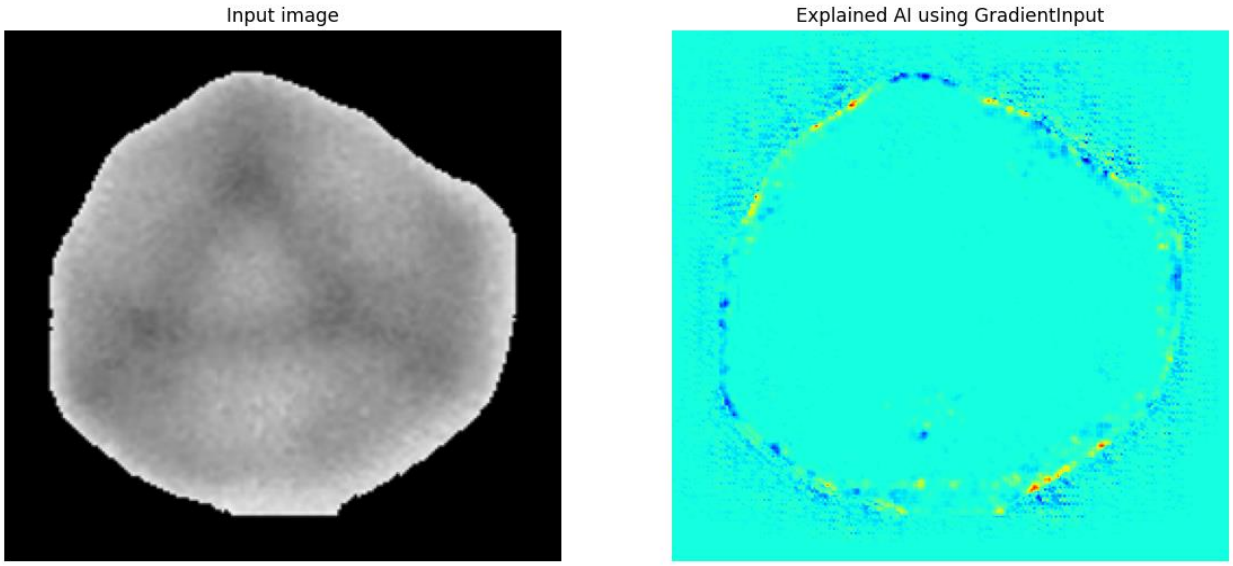
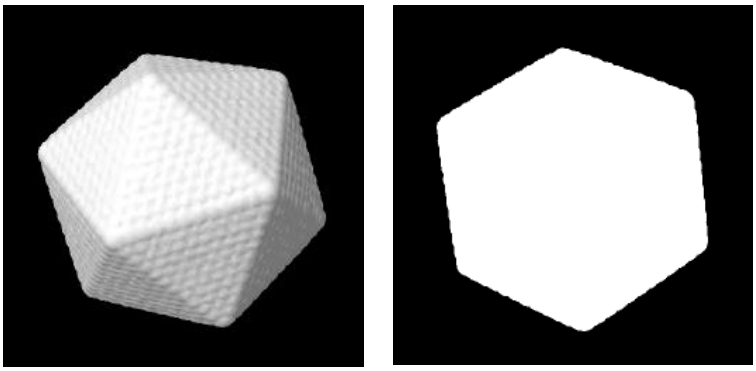


Figure 52: The model classifies the nanoparticle based on its outer edge.

The solution to this problem proposed by our tutors is to add silhouette images of each nanoparticle into the dataset to see if the model can identify the outer and inner edges of the nanoparticle at the same time.

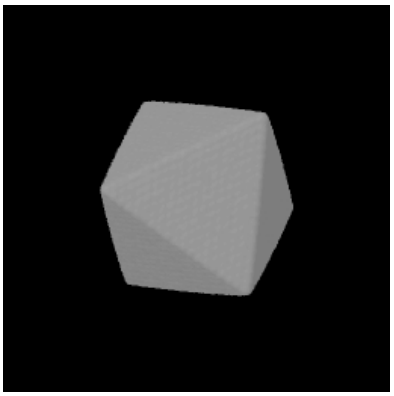


a: *Icosahedral class from the previous dataset.*

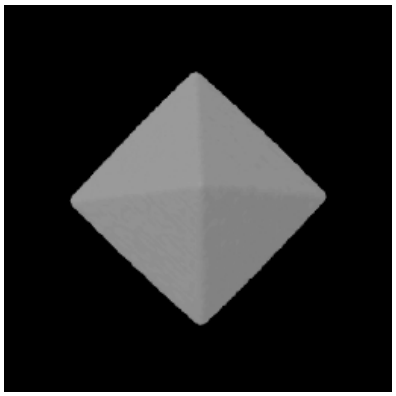
b: *The silhouette image of the mentioned nanoparticle.*

Figure 53: The silhouette image of the corresponding 3D image.

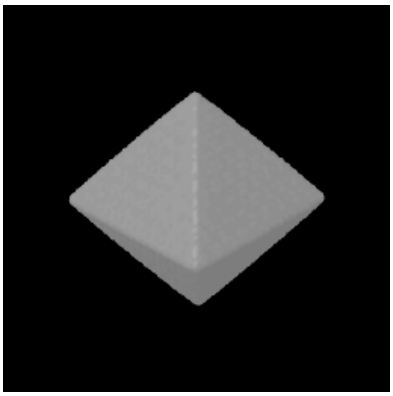
While training the model, we noticed that the nanoparticles' projection in TEM clichés is very randomized, and its rotation is not based on any particular axis. For example, cuboctahedra nanoparticle projection (or silhouette) can be hexagonal or square; octahedral nanoparticle projection can be rhombic, square, or hexagonal. This property has hugely complicated our problem.



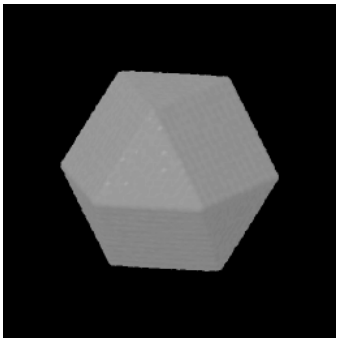
a: *Octahedral np with hexagonal silhouette.*



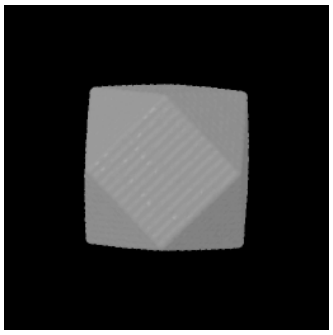
b: *Octahedral np with square silhouette.*



c: *Octahedral np with rhombic silhouette.*



d: *Cuboactahedral np with hexagonal silhouette.*



e: *Cuboactahedral np with square silhouette.*

Figure 54: Many different silhouette images with the same NP.

III. 3. 4 Generalized Dataset and Updated CNN Model

The previous dataset needs to be updated for this new challenge; thus, we asked M. Romuald for a more suitable one. The new dataset is now generated by rotating the nanoparticle in every possible direction. Examples shown in Fig. 54 are taken from this new data set. The dataset also adds two classes: tetrahedral nanoparticles (TD) and trigonal structures formed from a truncated trigonal bipyramid (TPT).

We first train a model to see if it can classify this new data set with plenty of confusion and similarity. The training process and model's architecture stay the same, except the output is now eight instead of 6. From the first ten epochs, we notice that our model's accuracy and validation accuracy progress weirdly. This abnormality might result from the fact that our model's simple architecture can no longer adapt to the complexity of the dataset. We now have to upgrade our model's architecture.

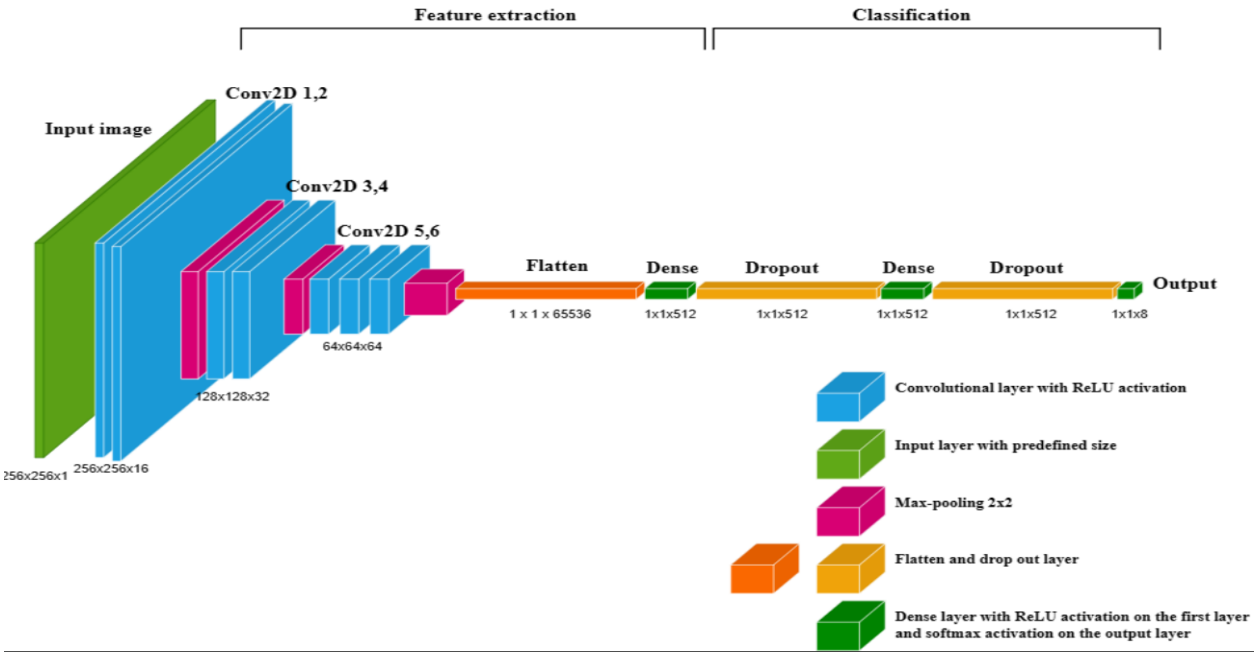
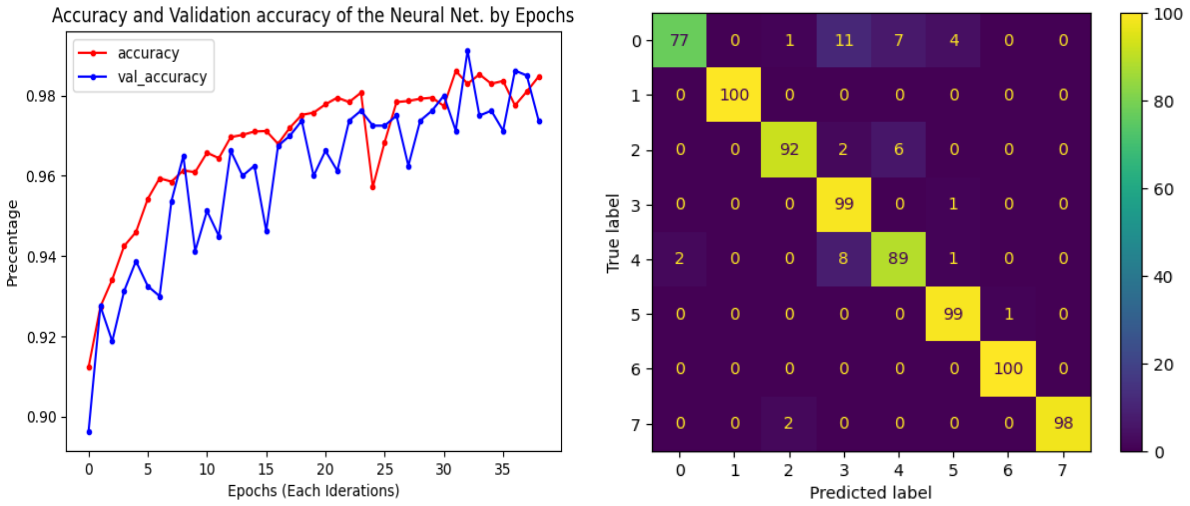


Figure 55: The upgraded architecture for the new model.



a: *Accuracy and Validation accuracy of the new model.*

b: *Confusion matrix of the new model.*

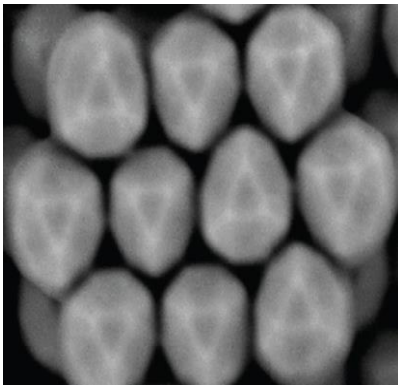
Figure 56: Results of prediction by new model.

Note that the labels of each class are 0 for CUBO, 1 for DEC, 2 for DODECA, 3 for FCC_CUBE, 4 for ICO, 5 for OH, 6 for TD, and 7 for TPT.

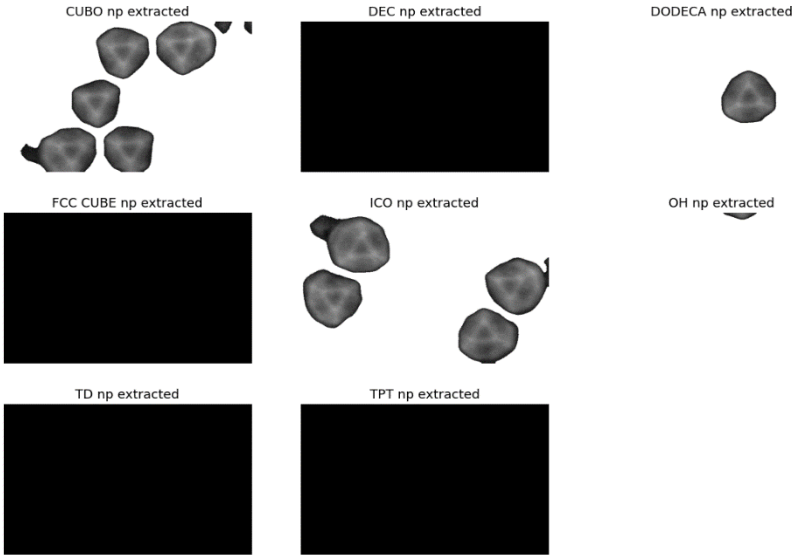
The accuracy and validation accuracy of the model is relatively high, around 98%. The confusion matrix shows that the model does an excellent job identifying almost every class except the cuboctahedral one. Cuboctahedral images are mostly mistaken for FCC_CUBE (labeled as 3), then for ICO (labeled as 4), and least mistaken for OH (labeled as 5). This might be because the cuboctahedral nanoparticles share the same pattern with those confused labels.

In Fig. 54, CUBO has a triangle pattern in its center with a hexagonal silhouette (confused as ICO). CUBO, classified as FCC_CUBE, is the most apparent case since they both have a lot of square patterns. It can also be confused with OH as octahedral nanoparticles' silhouette can sometimes be square (Fig. 57).

We can see its weakness by applying the new model to the previous example.



a: *Original image of icosahedral nanoparticles.*



b: *Nanoparticles classified by our model.*

Figure 57: Problems of new model.

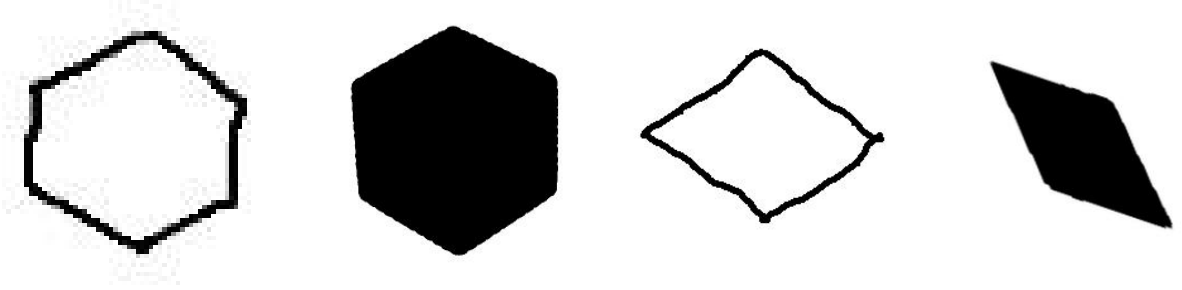
The new model can now identify more icosahedral nanoparticles, although there is still some previously mentioned problem with the outer edges with one nanoparticle classified as DODECA. We can see an intense confusion between CUBO and ICO in this example. The output vector's components for CUBO and ICO also indicate this substantial confusion, as their ratio is meager, around 9/1 to 7/3 (typically, without confusion, the output vector's component for the predicted class is at least 100 or 1000 times more significant than other components).

III. 3. 5 New Strategy toward the challenges

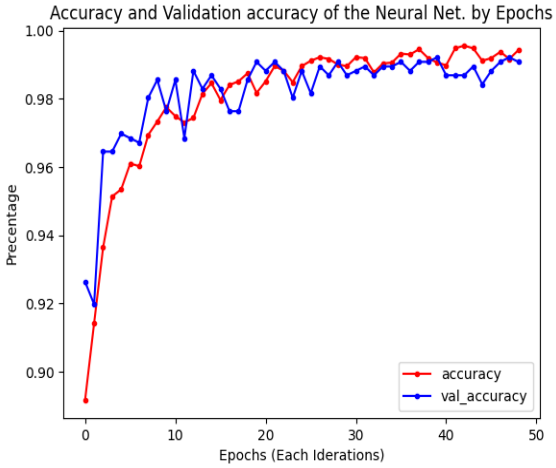
After many experiments, we deduce that our linearly stacked CNN model can only perform well when analyzing the outer edge of the object. Since this edge connects the object to the background, its gradient value is always much higher than other edges inside the nanoparticle. This is why the outer edge always contains the most essential pixels for the model.

The best solution we can think of is to build a model that first classifies the shape of the nanoparticles based on their outer border (hexagonal, circular, square, triangular, rhombic, pentagonal, etc.), then, for each shape, we build a sub-model to classify every possible morphology (CUBO, ICO, DODECA in case of hexagonal shape for example). Each sub-model requires a specific dataset, which can only be generated by M. Romuald. Our job is to upgrade our previous model, classifying three classes (square, triangle, circle) with more output.

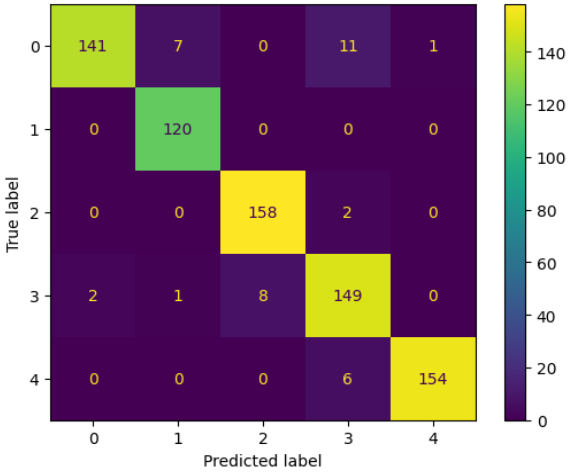
The classes added were hexagon and rhombus. First, we want to see if our model can identify between 2 geometrical shapes with four vertices (square and rhombus) and between a more rounded shape like a hexagon with a circle. We mix our dataset with hand-drawn images freely downloaded from Kaggle using the same method. We take the silhouette images of ICO and CUBO classes for the hexagonal shape-like image. Moreover, we built a generator for the rhombic shape by shearing the square images from 30 to 70 degrees randomly.



a ***b*** ***c*** ***d***



e: Accuracy and Validation accuracy of the model for shape detection with 5 classes.



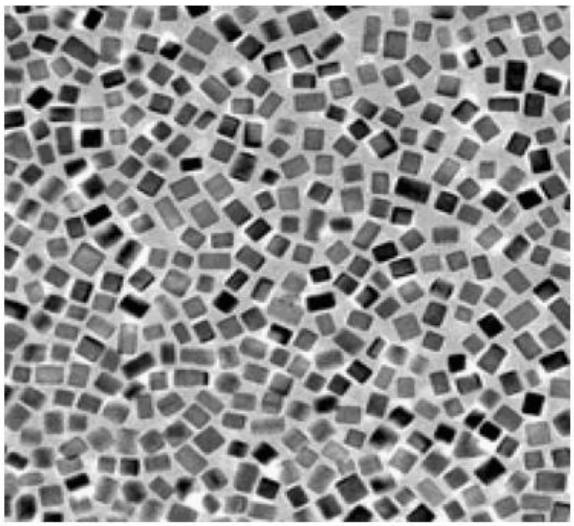
f: Confusion matrix of the new model for shape detection with 5 classes.

Figure 58: Newly added dataset and the results of the prediction.

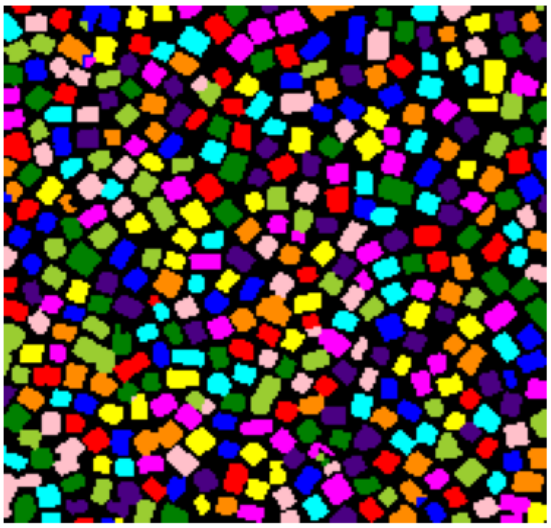
Note that the labels of each class are 0 for circle, 1 for hexagon, 2 for rhombus, 3 for square, and 4 for triangle.

IV. Results and Discussion

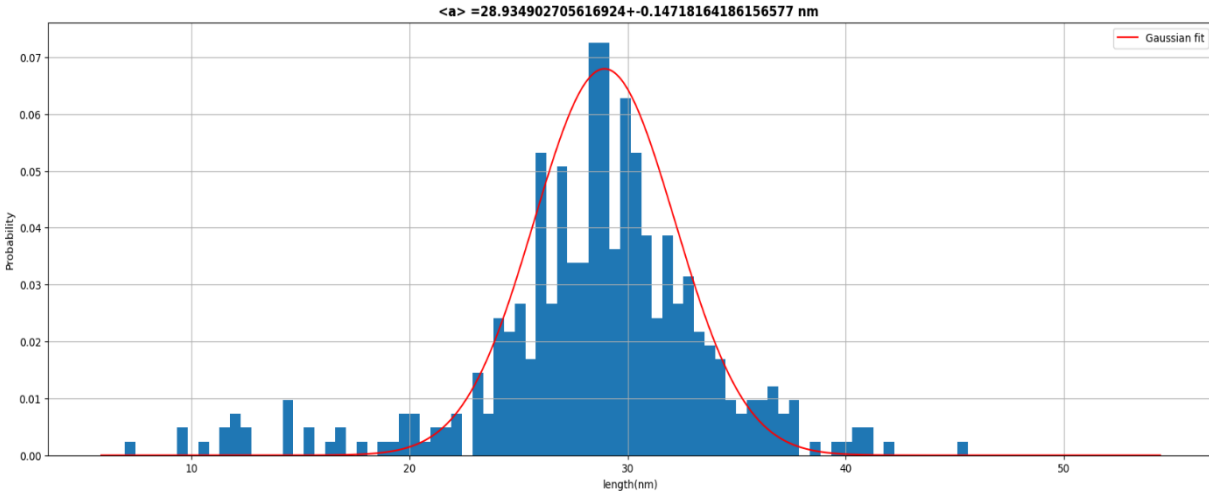
Figure 59 shows the integration of our algorithm on image with nanocubes of palladium.



a: *Original image of palladium nanocubes.*



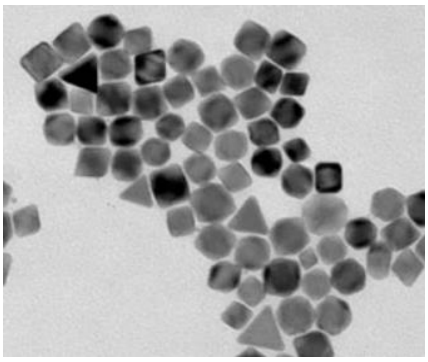
b: *Labeled segmented object.*



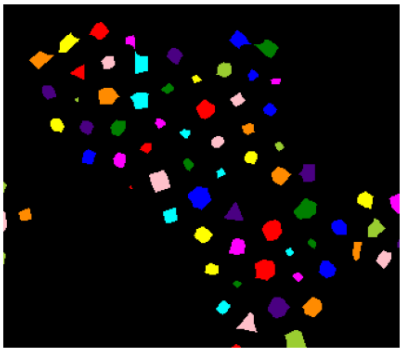
c: *Size histogram and fitted parameters.*

Figure 59: Integration of our algorithm on image with nanocubes of palladium.

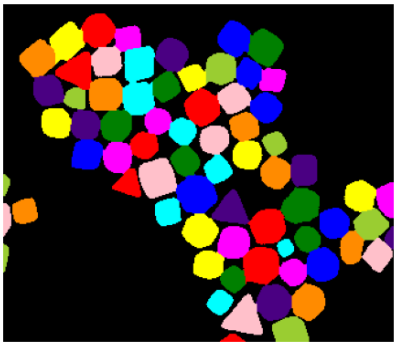
Figure 60 shows the results of our algorithm applied image with multiple shapes, both anisotropic and isotropic.



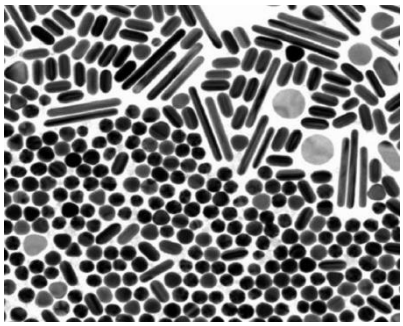
a: *Original image of nanoparticles with various shapes.*



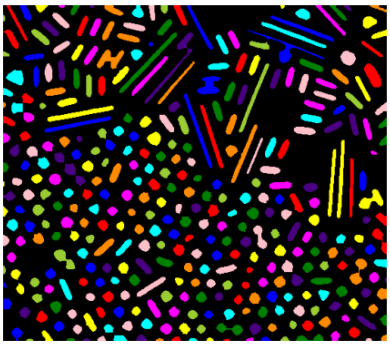
b: *Labeled "seeds" or markers for segmentation algorithm.*



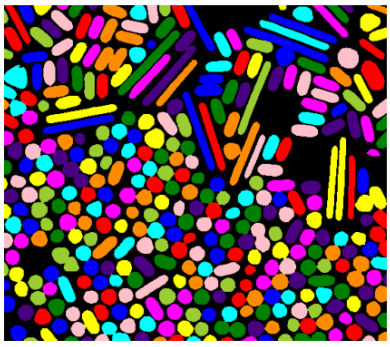
c: *Labeled segmented object.*



d: *Original image of nanoparticles with various shapes.*



e: *Labeled" seeds" or markers for segmentation algorithm.*



f: *Labeled segmented object.*

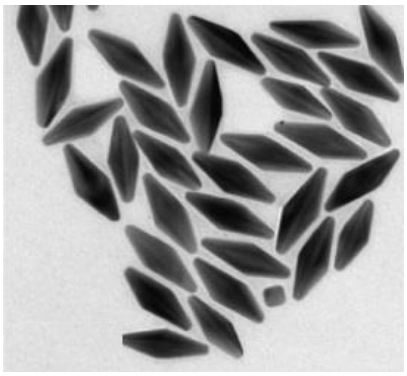
Figure 60: Results of our algorithm applied image with multiple shapes, both anisotropic and isotropic.

The first result of our project is a JupyterNotebook file, designed with user-friendliness in mind, containing every algorithm. The Notebook's usage is straightforward, functioning like a Python library. The user can access every function we have built by simply downloading the Notebook and loading it via Python. We have provided a comprehensive document and user manual about every algorithm and parameter for each function on our GitHub. We have also added detailed comments about how each variable and algorithm works in the source code. The only prerequisite is that the user must have basic knowledge of Python, such as downloading, installing necessary libraries, and managing files (inputting directory path to the wanted image). The leading Notebook contains a preprocessing algorithm, scale bar detection, object segmentation algorithm, size histogram construction and fitting algorithm, object extraction, plotting extracted objects for each class function, explainable AI function using Xplique, and a data generator function allowing to shear and invert color.

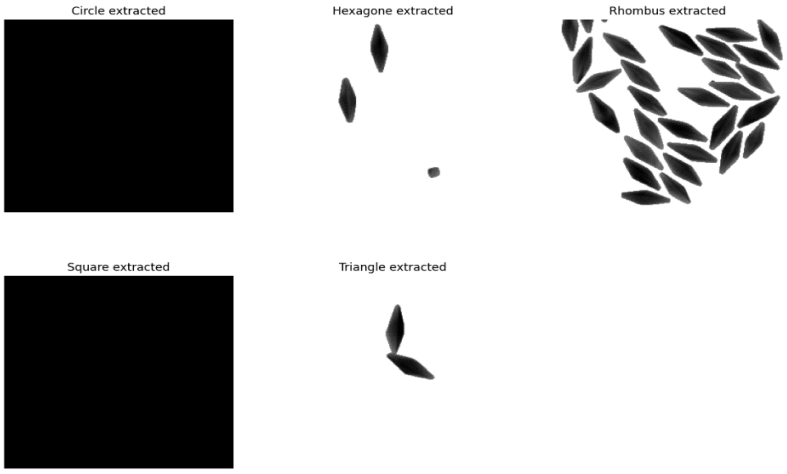
For the CNN model, the Notebook containing the training process is in GoogleColab format, which differs from the JupyterNotebook format. We also upload every Notebook we use to train our models to our GitHub. Each Notebook includes our comments on the training process, the file management via Google Drive, and the dataset preparation. The corresponding dataset and trained model will be uploaded as compressed files. It's important to note that these files are relatively large, with each data set weighing around 1 GB and each trained model ranging from 0.5 to 0.7 GB. Users can replicate the training process on their computer using our Notebook. They can also download the compressed model, unzip it, and then load it using TensorFlow to integrate it into the main JupyterNotebook functions or serve their purpose.

Our latest trained model, which successfully classifies five geometrical classes, is a testament to our project's potential. However, when applied in practical cases, assistance is currently needed to differentiate between hexagonal and rhombic shapes. Figure 61 below proves that our model performs well in identifying rhombic shapes, but it confuses hexagonal shapes with rhombic shapes. The confusion between hexagonal and circular shapes can be justified as the nanoparticles' outer border could be imperfect in practical cases. However, we still do not need to find out why it confuses hexagons with rhombuses.

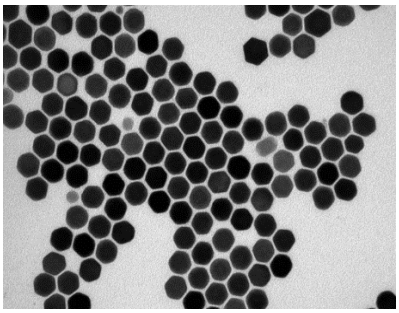
In our project's last days, we encountered a severe error when loading our models as they were randomly reset to the untrained state. This issue significantly disrupted our workflow and required us to retrain the models each time they were reset. The cause of this problem is still unsolved, and we suspect that the complexity of the TensorFlow library and the structure of the computer may be contributing factors. Other possible causes can be Xplique, the coding environment, the structure of the functional model, etc. This problem could have been better dealt with as we have to retrain a new one each time it is reset. We hope that the next group working on this project will find a way to solve this problem or, at best, never encounter this error.



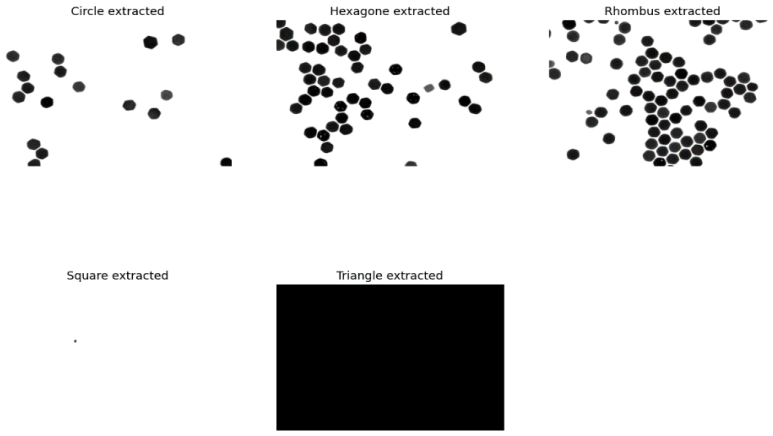
a: *Original image of rhombic nanoparticles.*



b: *Nanoparticles' geometrical shapes classified by our model.*



c: *Original image of hexagonal and spherical nanoparticles.*



d: *A lot of hexagons are confused as rhombus.*

Figure 61: Results of the prediction by our model.

Although the segmentation algorithm works quite well, our classification algorithm did not reach the expected performance (fig. 61). During the project, we had to study a whole new domain by ourselves, and this self-learning stage took plenty of time from us. We only had time to work with linearly stacked CNN, the most basic AI architecture for image classification, thus having many limits. At the end of the project, we learned that AI architecture can be built very differently. For instance, a CNN can be used for image classification, a recurrent neural network (RNN) for sequence generation, and a transformer for natural language processing. These different architectures allow AI to have different functions that correspond to different purposes. Different training techniques help improve the model's performance, such as fine-tuning (blocking specific layers while allowing other layers to be continuously trained). We could not apply these techniques in our project with only a little time left. Our work will allow the next group to understand the project better and guide them in developing newer methods using more advanced AI architectures.

V. Conclusion and Future Work

V. 1 . Summary of Findings

This project has introduced a new, automated approach for nanoparticle identification and classification from TEM images. This innovative method leverages computer vision techniques, such as image processing using OpenCV and deep learning with a convolutional neural network (CNN).

The key findings and achievements are:

- The algorithm developed in this project demonstrates remarkable effectiveness in nanoparticle identification and size calculation and offers significant benefits. It can identify individual nanoparticles, even in cases of coalescence, using a combination of image processing and watershed segmentation. Moreover, it provides accurate size calculations for each nanoparticle, offering crucial information for material characterization.
- Accurate Shape Classification with CNN: We successfully trained the CNN model to classify different morphologies of nanoparticles with high accuracy, demonstrating the potential of deep learning for this task.
- Overcoming Challenges: The project addressed several challenges encountered in traditional approaches, including handling overlapping nanoparticles, detecting smaller particles, and classifying anisotropic shapes.

This project has transformed nanoparticle analysis by automating previously manual tasks. This leap in efficiency frees researchers from tedious work, saving valuable time and effort. As a result, researchers can dedicate their expertise to more impactful endeavors, accelerating progress in nanomaterial research and development. This project serves as a powerful testament to the potential of computer vision and deep learning in propelling the field of nanotechnology forward.

V. 2 . Future Directions

The project can be expanded and improved in several directions:

- The next group can significantly enhance classification accuracy by delving into various CNN architectures and training strategies. This includes investigating advanced CNN architectures like ResNet, DenseNet, or Inception and exploring different training strategies such as transfer learning and fine-tuning existed, free-to-use pretrained models (YOLO, MobileNet, VGG...). These efforts aim to boost the model's performance and deliver more accurate results.
- They can also enhance the classification performance by incorporating additional image features, such as texture analysis, intensity gradients, or local descriptors. Furthermore, exploring other modalities like spectroscopy or X-ray diffraction, combined with TEM images, can lead to a more comprehensive analysis and richer insights.
- Developing a User-Friendly Software Tool: We aim to develop a robust and user-friendly software tool accessible to researchers and material scientists without requiring extensive coding knowledge.
- Development of an AI model requires a data preprocessing technique which helps the model concentrate on important features and information. The preprocessed data can also be visualized, aiding users in understanding the underlying structure and patterns. Since these techniques require deep understanding and knowledge of data science and statistics, we didn’t have enough time to learn and implement them into our project. The next group can explore the usage of various techniques such as Principal Components Analysis (PCA) and Linear Discriminant Analysis (LDA).

This research represents a significant leap toward the development of robust, automated methods for nanoparticle characterization. In the dynamic landscape of nanotechnology, this project's findings and future directions hold the potential to shape the trajectory of this exciting field.

References

[1] D. H. (. H. Ballard, Compute Vision, 1982.

[2] University of San Diego, "University of San Diego Online," University of San Diego, [Online]. Available: https://onlinedegrees.sandiego.edu/introduction-to-computer-vision/.

[3] CNRS, "Fidle," [Online]. Available: https://gricad-gitlab.univ-grenoble-alpes.fr/talks/fidle/-/wikis/home.

[4] L. RUER and J. YIN, "Caractérisation par Machine Learning de clichés TEM de nanoparticules métalliques," INSA Toulouse, Toulouse, 2023.

[5] EUSMI, "EUSMI," European Soft Matter Infrastructure, [Online]. Available: (https://eusmi-h2020.eu/access/wp4/NSL-BIOMA).

[6] Pradeep Research Group, [Online]. Available: https://pradeepresearch.org/microscopy-instruments/.

[7] H. Ashtari, "Spiceworks," Spiceworks Inc., 2022. [Online]. Available: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-computer-vision/.

[8] J. Brownlee, "machinelearningmastery," Machine Learning Mastery, [Online]. Available: https://machinelearningmastery.com/what-is-computer-vision/.

[9] K. A. Zhao R, "Survival Prediction in Gliomas: Current State and Novel Approaches.," *Gliomas*, 2021.

[10] DataRobot, "DataRobot," [Online]. Available: https://www.datarobot.com/wiki/overfitting/.

[11] J. L. Vincent, R. Manzorro, S. Mohan, B. Tang, D. Y. Sheth, E. P. Simoncelli, D. S. Matteson, C. Fernandez-Granda and P. A. Crozier, "Developing and Evaluating Deep Neural Network-based Denoising for Nanoparticle TEM Images with Ultra-low Signal-to-Noise," *arXiv*, 2021.

[12] A. G. Okunev, M. Y. Mashukov, A. V. Nartova and A. V. Matveev, "Nanoparticle Recognition on Scanning Probe Microscopy Images Using Computer Vision and Deep Learning," *Nanomaterials*, vol. 10, no. 7, p. 1285, 2020.

[13] K. Faraz, T. Grenier, C. Ducottet and T. Epicier, "Deep learning detection of nanoparticles and multiple object tracking of their dynamic evolution during in situ ETEM studies," *Sci Rep*, vol. 12, 2022.

[14] F. Chollet, "The Keras blog," Google, 5 June 2006. [Online]. Available: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html.

Appendix

- The final deliverable includes the entire open-source code for the NP determination, size measurement, and histogram drawing algorithms, the train model file for particle classification based on morphology, the entire database we use for training, and the final saved model. This report is also saved here.: https://github.com/AnhKhoaQC/PMD-2324
- OpenCV library documentation: https://docs.opencv.org/4.x/
- The final deliverable of our predecessors: https://github.com/ruerl/pmd22-23
- Kaggle website for finding dataset: https://www.kaggle.com/
- Deep Learning Course by CNRS: https://gricad-gitlab.univ-grenoble-alpes.fr/talks/fidle/-/wikis/home