LIÈGE université
Sciences Appliquées

Using Machine Learning algorithms to accelerate Phase-Field applied to compute microstructure evolution

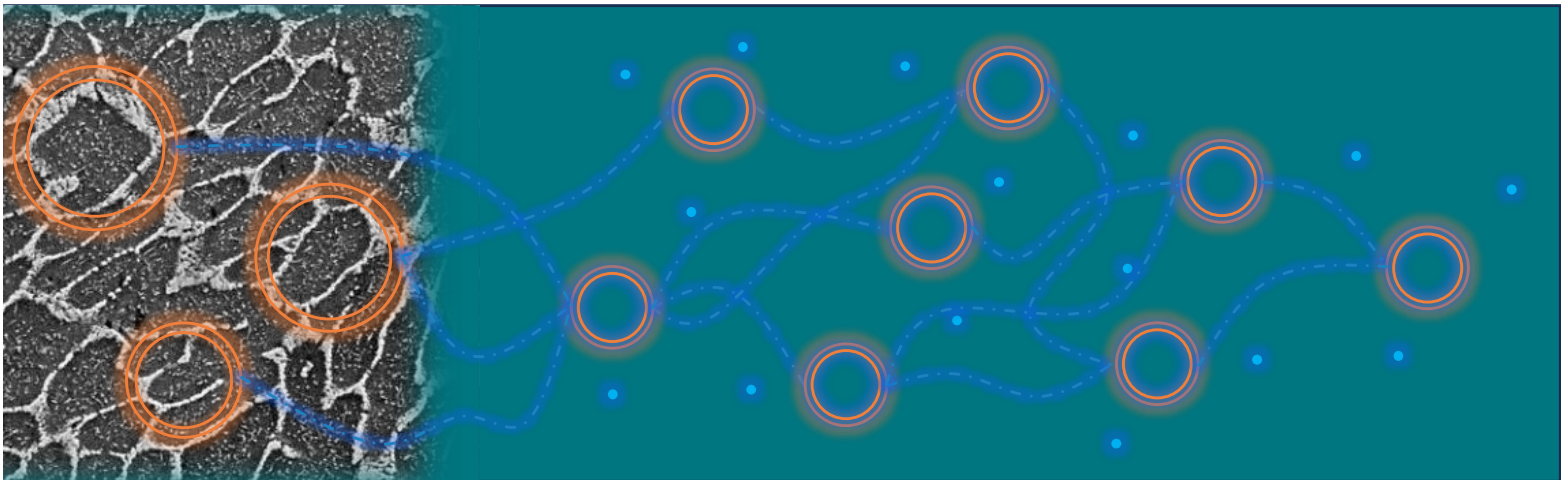
by Víctor Jesús Borlaf Nieto

August 2023

Thesis submitted for the fulfillment of the
Masters Degree in Aerospace Engineering

POLITÉCNICA

Supervisor: Anne Marie Habraken



Contents

Chapter 1

Introduction

Material development is and has always been a key element for the evolution of engineering. The research and improvement of the characteristics of alloys and their manufacturing processes is one of the broadest fields of study in technology. The current master thesis tries to contribute to this researching effort through the application of Machine Learning and Artificial Intelligence techniques to improve the performance of phase-field simulations applied to AlSi10Mg alloys.

This introductory chapter gives a quick summary of the project, the goals to achieve and the procedure that is carried out, starting with some context in Section 1.1. To know further about the material that this project works with Section 1.2 talks about the AlSi10Mg alloys, their features, most relevant manufacturing processes and their possible applications. Continuing with more context regarding simulation techniques, Section 1.3 gives a brief summary of the state of the art methods that use Machine Learning to improve the research in the materials field. Finally, Section 1.4 finishes the chapter by giving an outline of the project carried out in the master thesis and the procedure followed.

1.1 Context and goals

Research on materials is a fundamental piece of science and engineering development nowadays. To carry out accurate and useful tests is necessary a great amount of economical and time resources, a high price to pay when those resources are not available. One solution to this challenge is to develop the tests through simulation techniques, reproducing the phenomena happening in reality with as much accuracy as possible.

The Materials and Solid Mechanics (MSM) team from the University of Liège works on the research of phase-field simulation methods for AlSi10Mg alloys and other materials. Nevertheless, computational cost may be big for high resource demanding simulations, spending large amounts of computational resources and time to obtain the desired results. This is the reason that drives the team to new approaches using machine learning and artificial intelligence techniques, training prediction algorithms to faster obtain accurate results.

The goal of this master thesis is to, through the use of neural networks and linear regression algorithms, be able to predict the morphology of AlSi10Mg microstructures by providing their mechanical properties, which in this case is the yield stress value.

1.2 The AlSi10Mg alloy and the LPBF manufacturing

The technology development has pushed material researching to find stronger, cheaper and lighter alloys. Industries like automotive and aerospace intensively use aluminium-based alloys due to their very high strength-to-weight ratio.

The Aluminium-Silicon-Magnesium alloys mainly contain between 5-10% of Si, 0.3-0.6% of Mg and Al as the remaining material. They are able to reach very high strength properties thanks to Si and Mg_2Si precipitation hardening and Mg solid solution hardening, making it an excellent choice for components of high-strength and low-weight requirements.

More specifically, the alloy used for this project is the AlSi10Mg, which has a composition of around 90% of aluminium, 9% of silicon and 1% of magnesium. The use of this alloy is widely extended in the aerospace sector for critical mechanical components and, due to its very good heat conductivity, is also used in jet engine and automotive engine manufacturing. For ease of modeling, the rest of the possible alloying materials, that could be iron, copper or zinc, are completely neglected, as well as the possible porosities that could be induced by the manufacturing processes.

The AlSi10Mg alloy is very often used for the Laser Powder Bed Fusion (LPBF) manufacturing process (Liu et al. (2019)). This is an additive manufacturing process that fabricates metallic components with layer-by-layer addition of powder. After placing the layer of powder, a laser selectively hits the surface, melting the metallic grains and forming layer by layer the piece. The LPBF process achieves great mechanical properties and very complex shapes, making it one of the emerging manufacturing techniques of the last decades. In Fig. 1.1 is shown a more visual explanation of the process.

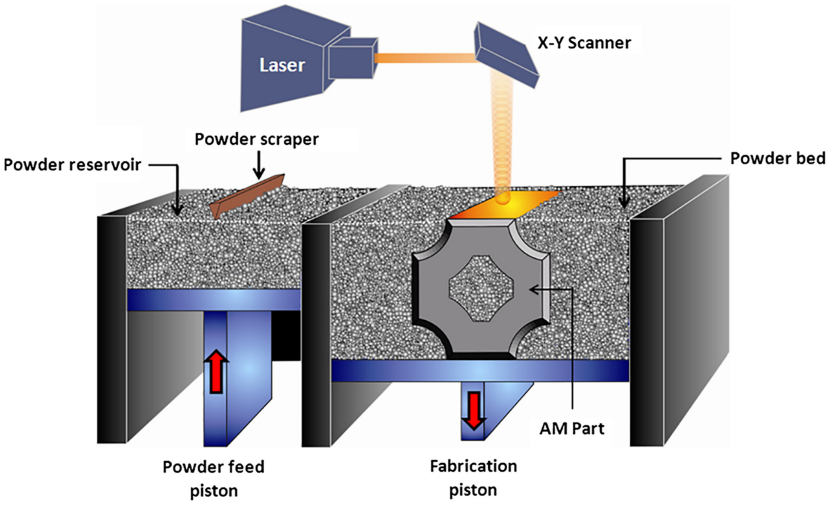
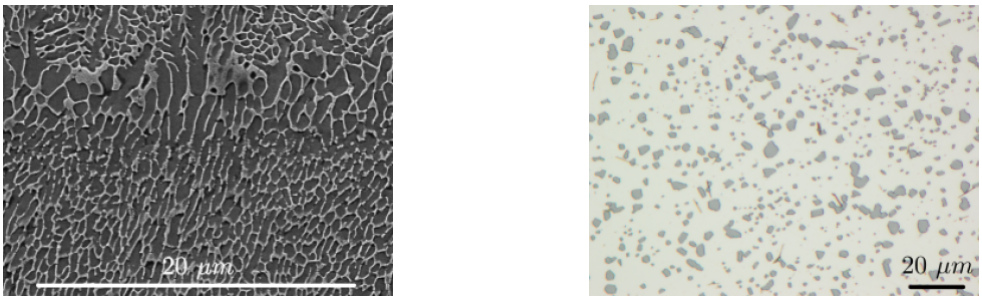


Figure 1.1: Laser Powder Bed Fusion manufacturing process.

The laser melts a thin layer of metallic powder, which rapidly cools down, acting as a fast quenching and leaving an out of equilibrium structure. In AlSi10Mg alloys, this structure exhibits a fine α-Al cellular microstructure surrounded by a Si-rich eutectic phase, as it can be appreciated in Fig. 1.2a. Nevertheless, when given a slower cooling, either because of a higher temperature of the fabrication plate (in LPBF process) or a later annealing treatment, the eutectic phase decomposes, arriving to an Al matrix microstructure with globulized Si precipitates, shown in Fig. 1.2b.



(a) Rapid cooling AlSi10Mg microstructure. (b) Slow cooling AlSi10Mg microstructure.

Figure 1.2: AlSi10Mg microstructures for LPBF process from Delahaye (2022).

The pseudoeutectic microstructure presents remarkable mechanical properties as the Si-rich eutectic network acts as a load carrier and impedes the dislocation motion, leading to a strengthening effect. Heat treatments that disintegrate the eutectic network may weaken the properties of the alloys, as globular Si precipitates do not have such a strong hardening effect. The yield stress, the property of interest in this project, also follows the same behavior, reaching values of around 260 MPa (Delahaye (2022)) that may decrease to 200 MPa or less in heat affected regions and components.

1.3 State of the art

The AlSi10Mg components manufacturing has been intensively researched during the last 20 years, and lately even more attention is pointed to it thanks to the outstanding properties that new manufacturing processes such as LPBF give them. At the University of Liège, the MSM team also contributes to this research, focusing on obtaining material properties though both experimental and computational methods. This project is mainly based on the studies carried out by the MSM team in PhD thesis such as Delahaye (2022) and close collaborators in other universities like Macías et al. (2020).

These following sections that explain the tools used for the development of the project are a quick introduction to give context and help the reader to understand the main parts of the project. A summary of each of them and some articles that inspired the work are given. Nevertheless, further information is provided in their corresponding chapters.

Yield stress calculation

This project gathers a set of very different fields, from material simulations to artificial vision and machine learning techniques. Starting with the first one, the prediction of the alloy mechanical properties is based on the same method used by Delahaye (2022), which is broadly described in Chapter 2.

Phase-Field simulation

For obtaining a reliable source of data regarding the properties of the material, the microstructures are modeled using phase-field modeling. A phase-field model is a mathematical model with the purpose of solving interfacial problems, based on the minimization of the free energy function together with the application of the particular equations of each problem (diffusion, fracture mechanics, viscous fingering...). This technique is relatively young-aged but has rapidly been popularized, becoming a method of choice due to its polyvalence for solving microstructure evolution problems. The MSM group has intensively worked on a

self-developed phase-field software (El Fetni et al. (2022)) that has been tested for simulated AlSi10Mg alloys manufactured through LPBF process. The in-depth details of phase-field simulations are far beyond the scope of this thesis and it is solely used as a tool for data gathering. Further information can be found in specialized books or articles such as Biner et al. (2017).

Computer vision and image analysis

Moving on to subsequent sections of the thesis, it is necessary to give an introduction to the use of computer vision and image analysis to obtain the characteristics of a material. As a way to standardize the analysis of experimental micrographies and microstructure features, several softwares have been developed for their use in the metallurgy field. Some examples may be *ImageJ* or *Image-Pro*, widely used for automatic analysis and feature quantification. Nevertheless, as the images analyzed in this thesis are virtual ones, computed by phase-field simulations, these options where discarded and instead a new software is developed through the use of tools such as the *OpenCV Python* library.

Shape detection and volume fraction analysis are carried out, but they need to be transformed to microstructural features. For developing this task the project relies on previous researches on microstructure analysis (Campbell et al. (2018), Altschuh et al. (2017)), being able to transform image data into features like precipitate radius, weight fraction of alloying materials or many other parameters.

Machine learning algorithms

The final step of the project, once the phase-field simulation results are gathered and their corresponding analysis of properties are carried out, is to generate microstructures through the training of a machine learning algorithm.

Before feeding the data set to the prediction algorithm, it is compressed though a Variational Autoencoder (VAE) algorithm. This technique consists of a neural network architecture that, when properly trained, is able to encode images into small latent vectors and decode them afterwards by loosing relatively few accuracy on the quality of the image. The possibility of reducing the weight of the training data enables the later prediction algorithm to have a faster and computationally cheaper training. The effectiveness of these kind of algorithms has been proven for micrographies by several researching teams (Kim et al. (2021)), including the MSM team (Fetni et al. (2023)) with promising results for their application in phase-field micrographies.

After the encoding of the micrographies the problem left is a Single-Input-Multiple-Output (SIMO) regression, as the latent vector needs to be obtained from a single value of the yield stress. Several models are considered for developing this final task, each of them with its unique characteristics that should fit the requirements of the input-output shape, data base size and accuracy. Among those models are found Neural Networks (Goodfellow et al. (2016)), that can be used as a multi-purpose tool for prediction tasks, Support Vector Machines (Cortes and Vapnik (1995)), that work well with non-linear problems like this one, Random Forests (Ho (1995)), very robust for small data sets and non-lineal problems, K-Nearest Neighbors, very simple and intuitive algorithms, or Gaussian Process Regression (Jazbec et al. (2021), Ashman et al. (2020)), that are probabilistic methods successfully tested together with VAE. A more detailed comparison is given in Chapter 6, where several algorithms are tested to obtain the best fitting one.

1.4 Procedure and project outline

To have a clearer structure of the project a flow chart is shown in Fig. 1.3, where each section is defined and illustrated.

Following the arrows and dividing the project into sections, it is possible to differentiate four areas of work:

- **Phase-Field simulations**: launch of all the phase-field simulations in order to create a proper data base that will be used for later predictions.
- **Variational-Autoencoder**: by using the previously built data base train a VAE neural network that is able to encode and decode the images. Once trained, the data base images are encoded and the resulting latent vector is sent to train the regression algorithm.
- **Prediction of mechanical properties**: the data base images are analyzed through the use of computer vision in order to obtain the features of the microstructures. Those features are later used to compute the yield stress. Each micrography is tagged with its corresponding value of the yield stress and sent to the regression algorithm.
- **Regression algorithm**: once the latent vector (representing the encoded micrography) and the yield stress are obtained the regression model is trained. After training, the final model should be able to, by providing a value of the yield stress, obtain the corresponding latent vector (encoded image). This latent vector is afterwards decoded in the VAE to obtain back the reconstructed (predicted) image.

  The final model should be able to, by providing it a value of the yield stress, obtain the corresponding latent vector that can be later decoded into a full micrography.

This thesis follows a similar path, starting with the explanation of the model used for the prediction of the yield stress through the use of the characteristics of the microstructure in Chapter 2. Later on, the structure of the data base and its generation is described in Chapter 3. The post-processing code developed for extracting the features from the micrographies is widely explained in Chapter 4. Starting with the machine learning field, in Chapter 5 is described the structure of the Variational Autoencoder and Chapter 6 talks about the regression algorithm to finish the scheme. The final results of microstructure prediction obtained after all the process are shown in Chapter 7, ending up with the conclusions and aspects to improve in Chapter 8. As an addition to give more interest to the project, the final appendix talks about the prediction of the strain-stress curve, with a similar methodology used to calculate the yield stress (Chapter A).

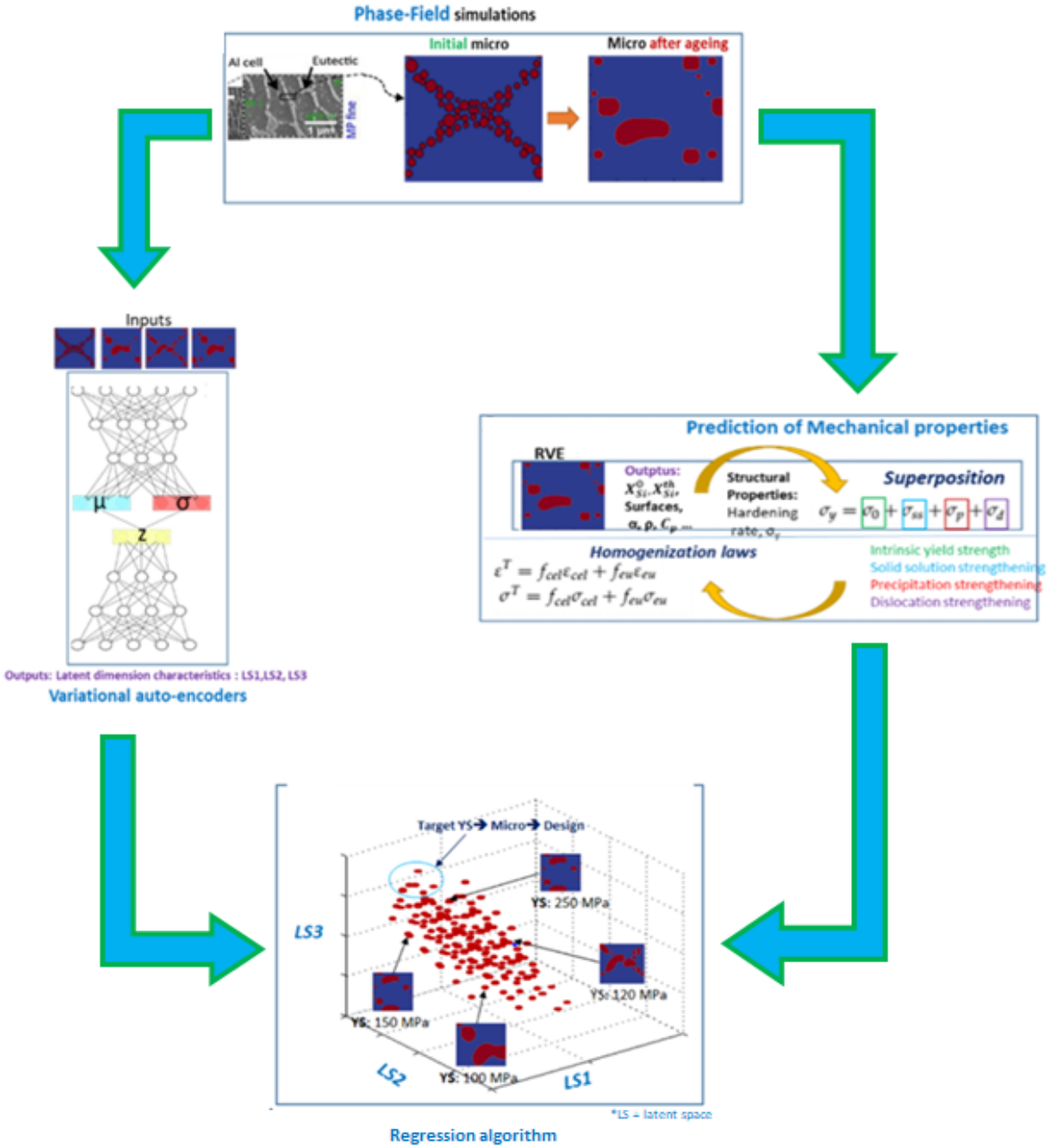


Figure 1.3: Flow chart defining the structure of the project.

Chapter 2

Yield stress prediction

List of parameters

Symbol	Designation	Units
b	magnitude of Burgers vector of α-Al phase	$[nm]$
$\bar{F}$	mean obstacle strength	$[N]$
f_i	volume fraction for phase i	$[-]$
G	shear modulus of α-Al phase	[GPa]
k_i	scale factor of solid solution strengthening for element i	$[-]$
l	mean distance between precipitates	$[nm]$
M	Taylor's factor	$[-]$
r_c	critical radius for shearing	$[nm]$
$\bar{r}_d$	mean radius of the precipitates	$[nm]$
X_v^d	volume fraction of the Si precipitates	$[-]$
X_{wi}^{α}	weight fraction of compound i in α-Al phase	$[wt\%]$
α_d	constant related to the dislocation strengthening	$[-]$
α_p	constant related to the precipitation strengthening	$[-]$
ρ_d	dislocation density	$[m^{-2}]$
ρ_{d0}	initial dislocation density	$[m^{-2}]$
σ_y	yield stress	$[MPa]$
σ_0	intrinsic yield stress of aluminium	$[MPa]$
σ_{ss}	solid solution strengthening component	$[MPa]$
σ_p	precipitation strengthening component	$[MPa]$
σ_d	dislocation strengthening component	$[MPa]$

2.1 Literature review

Considering that the yield stress at the point of 0.2% of plastic deformation is the mechanical property that is later predicted with a regression algorithm, it needs to be calculated in order to have an adequate data base for training and validation. A brief literature research has been done to find sufficiently accurate state of the art models that can analytically predict this value.

A usual approach for calculating properties such as the yield stress is by the addition of several components related to mechanisms that harden the material. This method can be used for a wide variety of alloys, such as steels (Seo et al. (2016)) or aluminium alloys (Cheng

et al. (2003)).

For the Al-Si-Mg alloys, which are the ones being used in this project, the hardening contributions taken into account to predict the yield stress can differ depending on the study. Approaches like the one carried out by Myhr et al. (2001) consider the hardening due to the aluminium matrix together with the hardening caused by the solid solution of Si and Mg and the hardening by Si precipitates. Other methods like the one used in Macías et al. (2020) also add the contribution of dislocations in the material and the boundary of the eutectic region impeding the dislocation movement.

Finally, due to its good results, the material studied and the ease of communication with the author, the model chosen for the analytical calculation of the yield stress is the one used by Delahaye (2022). The explanation and development of the model is given in Section 2.2. To test the model, the results obtained by Delahaye (2022) are reproduced, based on the parameters of Section 2.3. The final comparison between the available values of the yield stress is presented in Section 2.4.

2.2 Equations and model development

First of all, it is necessary to take into account the microstructure of the material. As it is mentioned in Section 1.2, the Al-Si alloys, when rapidly cooled like in the LPBF process, acquire a biphasic structure composed by a series of Al-rich cells surrounded by a Si-rich eutectic phase. Due to such a different composition of both phases the equations and parameters to obtain the yield stress value of each phase differ slightly. Because of that, the yield stress of each phase is individually calculated and afterwards an homogenization process is carried out to obtain the overall value.

The yield stress at each phase is given by the addition of the different hardening contributions as it is shown in Eq. (2.1).

$$\sigma_y = \sigma_0 + \sigma_{ss} + \sigma_p + \sigma_d \tag{2.1}$$

where σ_0 is the intrinsic yield stress of aluminium, σ_{ss} is the contribution of the solid solution, σ_p is the contribution of the Si precipitates and σ_d is the contribution of the dislocations of the material.

Solid solution strengthening

This strengthening is due to dissolved alloying elements within the aluminium matrix. In the present case these alloying materials are silicon and magnesium. The difference between the size of the Al atoms with the Si and Mg cause a distortion of the pure aluminium network, hindering the movement of dislocations and thus increasing the Young and shear modulus. For the modelling of the solid solution hardening, the approach given by Labusch (1970) is the one used. Here the weight fraction of each alloying material is powered to an exponent and weighted by a certain constant, as it is shown in Eq. (2.2):

$$\sigma_{ss} = \sum k_i {X_{wi}^{\alpha}}^{2/3} \tag{2.2}$$

where k_i is the scale factor of the solid solution hardening of the material i and X_{wi}^{α} is its weight fraction.

Precipitation strengthening

The precipitation hardening is directly caused by the Si precipitates that impede the movement of dislocations through the Al matrix. These precipitates are purely composed by FCC Si diamond, that have a much higher shear modulus than the α-Al of FCC lattice, as well as a much lower deformation. The hardening effect is given by the balance of the forces of the obstacles and the tension lines in the dislocations. Depending on the strength of the obstacle, which is given by the radius of the precipitate, two cases can be considered. In the first one, where the precipitate is too hard to be sheared, the obstacle is bypassed by the dislocation though the Orowan mechanism. In the second one, the precipitate is sheared by the dislocation when it is not sufficiently hard.

Also there is an important distinction between the calculation of the precipitation hardening in the cell phase and in the eutectic phase. The big difference in precipitate volume fraction in each phase requires the use of different models for their analytical calculations.

In the cell phase, the proportion of precipitates is much lower and, through the application of the Friedel formalism (Friedel (2013)) Eq. (2.3) defines the strengthening (Myhr et al. (2001)):

$$\sigma_p = \frac{M}{\sqrt{2\alpha_p G} b^2 \bar{r}_d} \sqrt{\frac{3X_v^d}{2\pi}} \bar{F}^{3/2} \tag{2.3}$$

with M being the Taylor's factor, G the shear modulus of α-Al phase, b the magnitude of Burgers vector of α-Al phase, $\bar{r}_d$ the mean radius of the precipitates, X_v^d the volume fraction of the precipitates, α_p a constant related to the precipitation hardening and $\bar{F}$ the mean obstacle strength. This last parameter depends on the mechanism the dislocations follow to surpass the precipitates, either by the Orowan mechanism Eq. (2.4a) or the shearing of the particle Eq. (2.4b).

$$\bar{F} = 2\alpha_p G b^2 \left(\frac{\bar{r}_d}{r_c}\right) \qquad if \quad \bar{r}_d < r_c \tag{2.4a}$$

$$\bar{F} = 2\alpha_p G b^2 \qquad if \quad \bar{r}_d > r_c \tag{2.4b}$$

Here r_c is the critical radius of the precipitate which is the threshold value that separates both dislocation movement methods.

In the eutectic phase, the Friedel formalism is not applied, Eq. (2.5) defines the precipitation strengthening (Chen et al. (2017)):

$$\sigma_p = \frac{M\bar{F}}{bl} \tag{2.5}$$

where l is the mean distance between precipitates, that was substituted in Eq. (2.3) through the Friedel formalism.

Dislocation strengthening

The accumulation of dislocations increase their movement difficulty. This phenomenae can be caused by plastic deformation or other treatments during the manufacturing process. As the material is not considered to go under plastic deformation, the dislocation hardening remains as a constant value to compute the yield stress, and is defined by the Taylor's relationship (Taylor (1934)) in Eq. (2.6):

$$\sigma_d = \alpha_d M G b \sqrt{\rho_d} \tag{2.6}$$

where α_d is a constant related to the dislocation hardening and ρ_d is the dislocation density, which remains constant here and can be always defined by its initial value ρ_{d0}

After obtaining the contributions of all the hardening mechanisms in both the cell and eutectic phase Eq. (2.1) is used to obtain the yield stress value in each of the phases, $(\sigma_y)_{cell}$ and $(\sigma_y)_{eu}$. To finish, an homogenization process following the approach of Bouaziz and Buessler (2002), Eq. (2.7) is applied in order to obtain the macroscopic value of the yield stress.

$$\sigma_y = f_{cell}(\sigma_y)_{cell} + f_{eu}(\sigma_y)_{eu} \tag{2.7}$$

Here f_{cell} and f_{eu} are the volume fraction of each of the phases in the overall material.

2.3 Input parameters

After defining the model for predicting the yield stress in Al-Si-Mg alloys, it is tested with an example case to see if the construction is correct. For that, the example taken is the same as in Delahaye (2022), where the parameters are defined in Table 2.1 and the results are later compared with a series of experimental values. It is necessary to say that, as the calculations of each phase are done separately until the homogenization, the parameters for the cell and eutectic regions are different.

Parameters	**Units**	**Value**		**Reference**
		Cell	**Eutectic**	
σ_0	$[MPa]$	10		Myhr et al. (2001)
ρ_{d0}	$[m^{-2}]$	10^{13}		Nicholson (1962)
f	$[vol.\%]$	0.8	0.2	
$\bar{r}_d$	$[nm]$	0	7	
X^{α}_{wSi}	$[wt.\%]$	2.5	0	Delahaye (2022)
X^{α}_{wMg}	$[wt.\%]$	0.43	0	
X^{d}_{v}	$[vol.\%]$	0	41.7	
G	$[GPa]$	25.4		
b	$[nm]$	0.283		Cheng et al. (2003)
M	$[-]$	3.06		
α_d	$[-]$	0.3		Chen et al. (2017)
α_p	$[-]$	0.5		
k_{Si}	$[Mpa.Wt\%2/3]$	66.3		Myhr et al. (2001)
k_{Mg}	$[Mpa.Wt\%2/3]$	29		
r_c	$[nm]$	4.29-7.15		
l	$[nm]$	-	50	Chen et al. (2017)

Table 2.1: Example parameters for the yield stress prediction from Delahaye (2022).

2.4 Results

The yield stress values obtained from the previously developed model are compared with other predicted and experimental values.

Firstly, the contributions to the hardening are compared one by one in Table 2.2 to the values of the original prediction done in Delahaye (2022) and provided by the author.

	Borlaf		Delahaye	
	Cell	**Eutectic**	**Cell**	**Eutectic**
σ_0	10		10	
σ_{ss}	138.65	0	139	0
σ_p	0	444.58	0	445
σ_d	21.09	21.09	21	21
σ_y	169.73	475.67	170	476
$(\sigma_y)_T$	230.92		231	

Table 2.2: Yield stress contributions comparison between Delahaye (2022) and self prediction models. Units in MPa.

When looking at the values of each computation is clear to see that the reproduction of the model is successful, obtaining the same hardening contributions as the ones provided. The results are also compared in Fig. 2.1 with the experimental values obtained in the studies carried out by Delahaye (2022) and Macías et al. (2020), carried out with the same AlSi10Mg alloy.

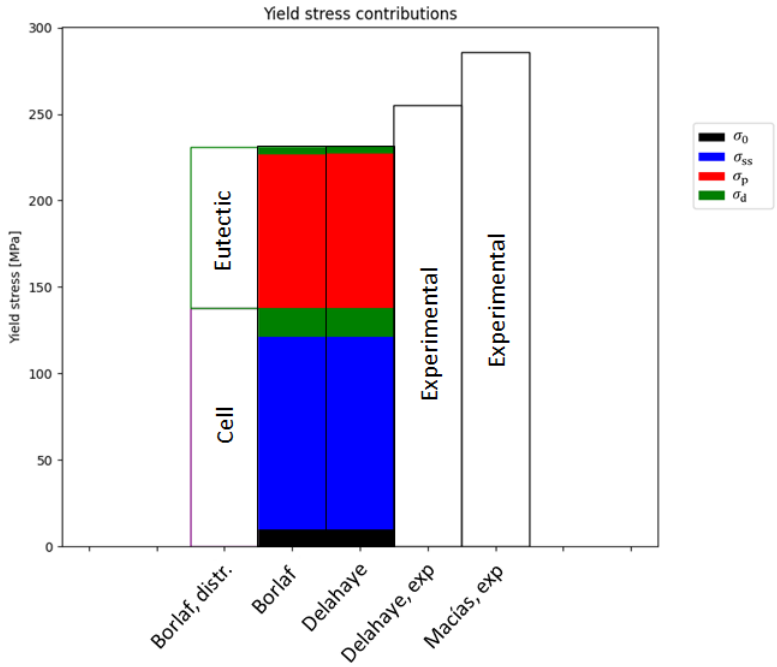


Figure 2.1: Comparison of the different contributions to the yield stress with experimental data.

The predicted values get very close to the experimental ones, allowing us to use this prediction model for the further steps of the project.

Chapter 3

Data base generation

When developing a machine learning algorithm it is fundamental to create a good and reliable data base. The performance of the algorithm and the quality of the predictions directly depend on it. Because of this reason the data base is carefully constructed, making it sufficiently large for the algorithm to learn but at the same not too much to avoid overfitting and have reasonable training times.

This chapter gives an explanation of how the data base is constructed, with an overview of the structure in Section 3.1, followed by the description of the files and variables obtained from the phase-field simulations in Section 3.2. Afterwards, in Section 3.3 the microstructures used for starting and launching a phase-field simulation are shown and described. The chapter concludes in Section 3.4 with an example of a phase-field simulation, giving a more detailed description of the evolution of the microstructure and its features, and mentioning which features are later used for the machine learning algorithm training.

3.1 Structure of the data base

For preparing a proper data base it is necessary to consider a set of variables that make the data base sufficiently diverse to have interesting results and a wide range of prediction possibilities, but at the same time sufficiently small to avoid complexity and wrong predictions.

Considering this, a set of different values were given to three variables of the phase-field simulations.

- **Initial microstructure**: is the structure of the material, with its corresponding distribution of eutectic phase and Si solution, at the beginning of the simulation. There have been used **3 different initial microstructures** for building the data base, shown in Section 3.3.
- $\dot{T}$ **or rate of temperature rise**: is the variable that gives the increase rate of the temperature in a phase-field simulation. The evolution of the microstructure changes depending on the value of $\dot{T}$, having different temperatures and shapes of the eutectic phase at the same time steps of the simulation for different values of $\dot{T}$. For each initial microstructure used for the simulations there are **18 different values of** $\dot{T}$. These values range from $\dot{T} = 15$ to $\dot{T} = 100$ Kelvin per minute and are better outlined in Fig. 3.1.
- **Saving time step**: is the virtual time at which the state of the microstructure is captured, obtaining one micrography. Around **30 micrographies** are obtained at each

simulation, making it possible to follow the whole evolution of the microstructure.

Adding up all the saved micrographies the overall number of elements in the data base is **1620 micrographies**. In Fig. 3.1 there is a more visual scheme of the structure of the data base.

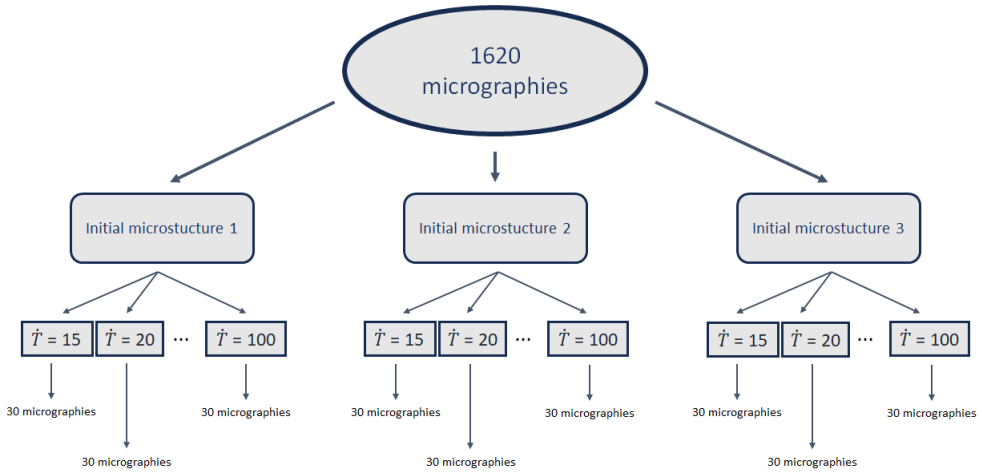


Figure 3.1: Data base structure.

The rest of the parameters in the phase-field simulations are kept the same. The physical properties of the material are fixed and the thermal treatment given to the microstructure starts at $T_i = 400$ K, with a constant increase of temperature given by $\dot{T}$, until arriving to the final temperature, $T_f = 750$ K, where it remains for the rest of the simulation. The saving time step is also kept constant with a value of $\Delta t = 170$ s.

3.2 Files obtained from phase-field simulations

The data contained in the data base is obtained from lots of phase-field simulations. These simulations take as an input the initial state of the microstructure, shown in Section 3.3, and every time they arrive to a saving time step they keep the current state of the microstructure. This current state is defined by two variables, the quantity of Si in solid solution, X_{wSi}, and the phase parameter, η, which indicates weather the region is in cell or eutectic phase. Both of them are defined for each node of the simulated microstructure, mapping all the surface and defining the morphology of the microstructure.

These two variables are saved and compressed in a *.npz* file. At the same time, the file is tagged with a series of characteristics that define the simulation and the time at which the micrography is taken. The name of the file includes these characteristics as shown in Fig. 3.2.

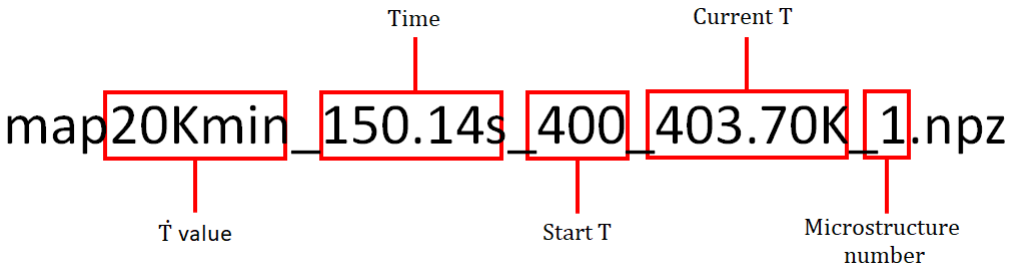


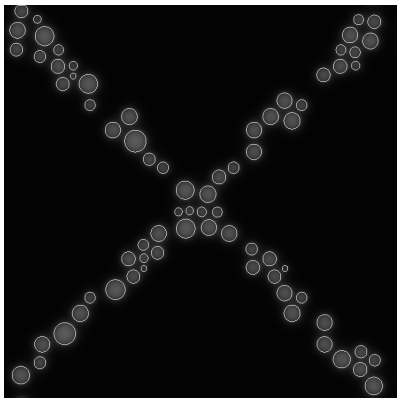
Figure 3.2: File name of an element in the data base.

- $\dot{T}$ **value**: increase rate of the temperature.
- **Time**: virtual time at which the micrography is taken.
- **Start T**: temperature at which the simulation starts.
- **Current T**: temperature at which the micrography is taken.
- **Microstructure number**: initial microstructure of the simulation.

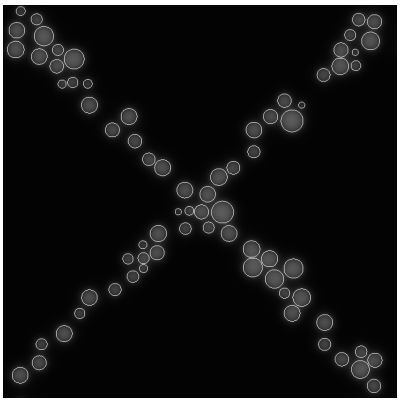
3.3 Initial microstructures

Even though the value of $\dot{T}$ is changed for each simulation, the evolution of the microstructure is roughly the same for all the cases. In order to add more diversity to the data base, three different initial microstructures are used, giving a wider range of prediction to the machine learning algorithm.

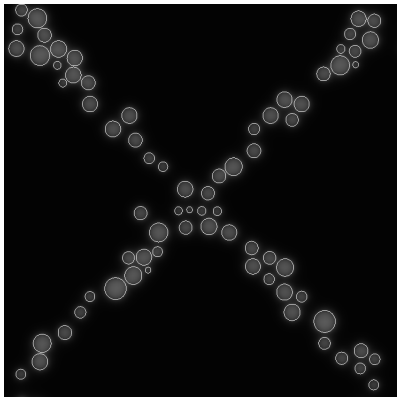
The first initial microstructure, shown in Fig. 3.3a, was provided by the MSM group at the beginning of the project and has been modified to obtain the two other microstructures, shown in Fig. 3.3b and Fig. 3.3c.



(a) Initial microstructure 1.



(b) Initial microstructure 2.

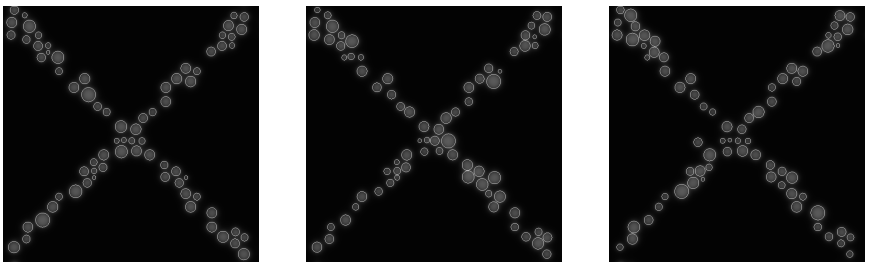


(c) Initial microstructure 3.

Figure 3.3: Initial microstructures used to develop the data base.

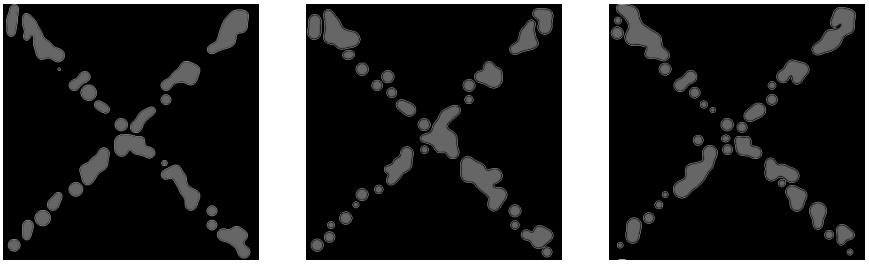
The most efficient way found to modify the microstructure, obtaining a new one without having to start from scratch, is to leave the centers of the precipitates fixed while their radii are randomly redistributed. No new values of the radii are given, instead the original ones are redistributed along the different centers. By doing this the cellular-eutectic structure is conserved and, as the overall sum of the area of the precipitates and X_{wSi} are kept the

same, the physical properties of the material do not change, having all the microstructures similar characteristics at the early stages of the simulation but very different evolutions. The effectiveness of these modifications is demonstrated in Fig. 3.4, where the evolution of the three different microstructures is shown for the case of $\dot{T} = 20$ K/min.



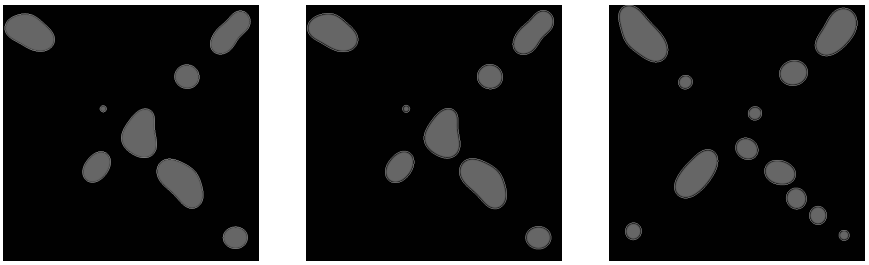
Microstructure 1. Microstructure 2. Microstructure 3.

Initial state. Time = 0 s.



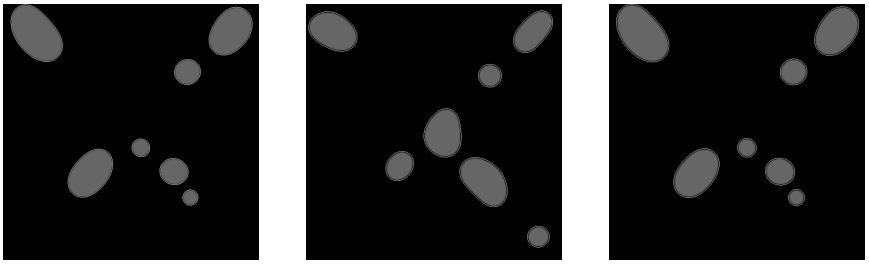
Microstructure 1. Microstructure 2. Microstructure 3.

Time = 680 s.



Microstructure 1. Microstructure 2. Microstructure 3.

Time = 1362 s.



Microstructure 1. Microstructure 2. Microstructure 3.

Time = 2044 s.

Figure 3.4: Evolution of the three microstructures in phase-field simulation for $\dot{T} = 20$ K/min.

It is clear that, even though the modifications in the initial microstructures are small and difficult to notice, the evolution of the precipitates is completely different.

3.4 Example of a simulation

Once defined the structure of the data base and how to obtain all the elements composing it, the simulations are launched.

Because how many they are, the phase-field simulations require high resources of computation power. Due to this, it was decided to use the *NIC5* cluster from the University of Liège. The simulations are launched automatically with different parameters each, and once well optimized it is possible to obtain a data base such as the one used in this project in around **12 days** using only **8 processing cores and 8 GB of RAM**.

The files are saved in specific folders and tagged as it is shown in Fig. 3.5 to have them well organized and ready to post-process, following the procedure described in Chapter 4.

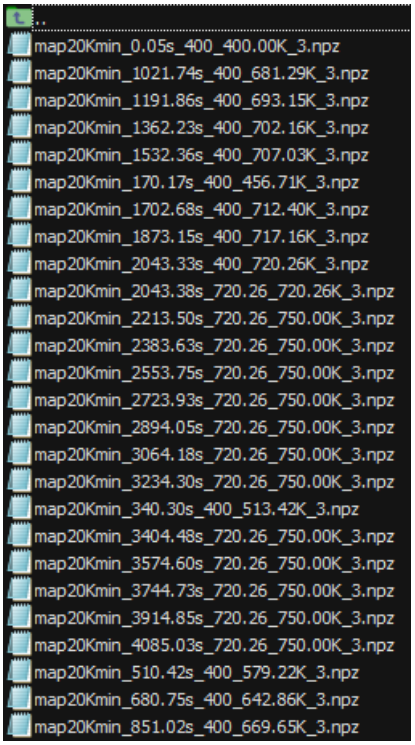


Figure 3.5: Folder example in the data base.

Chapter 4

Post-processing analysis

Once the phase-field simulations have been carried out, it is time to obtain the parameters of interest. As seen in the flow chart that defines the project of Fig. 1.3, for every saved microstructure the corresponding micrography image is extracted. The microstructure features are then used for the yield stress computation.

This chapter shows the procedure followed to obtain the value of σ_y and the necessary parameters to calculate it, starting in Section 4.1 with the obtaining of the micrography and a brief description of its morphology. In Section 4.2 are enumerated the features that need to be extracted from the microstructure in order to calculate the yield stress. As the equations for the yield stress computation change depending on the type of structure, Section 4.3 explains the method followed to distinguish the cellular and eutectic regions. Finally, the technique used to obtain each of the needed parameters is described in Section 4.4, together with the computation of the yield stress and the results for a set of example microstructures.

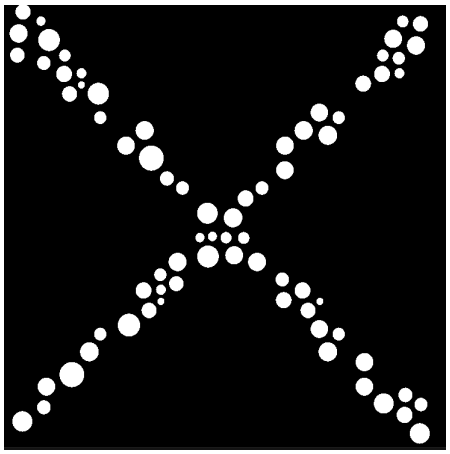
All this process, as it is mentioned in previous chapters, is carried out through *Python* scripts, highly supported by libraries such as *Numpy* and *OpenCV* who are widely used and have very powerful and intuitive tools for computer vision tasks.

4.1 Creation and morphology of the micrographies

The starting point of the post-process is the outputted files of the phase-field simulations. These are *.npz* compressed files that contain a description of the grid representing the microstructure. More concretely, each file includes the values of the η and X_{wSi} variables in every node of the grid, which determine the phase and the quantity of Si solute correspondingly. These variables are enough to describe the micrography and extract the set of features for the yield stress calculation.

The first thing done in the script is the creation of the micrography image, which represents both the boundary that separates the cellular and eutectic phases, and the quantity of silicon solute.

For the representation of the interface the η variable is loaded in an empty matrix. This variable indicates weather the node is closer to a cellular phase or an eutectic phase, evaluating it with a value between 0 and 1 correspondingly. To differentiate the phases the matrix is thresholded, assigning a cellular phase to values below 0.5 and an eutectic phase to values above. For that purpose, the *threshold* function of the *OpenCV* library is used, obtaining figures such as the ones shown in Fig. 4.1.

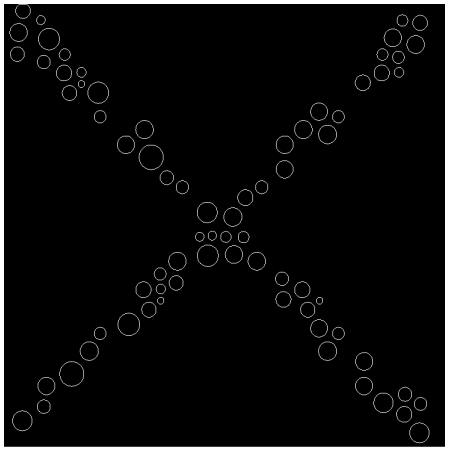


(a) Early stage precipitates.

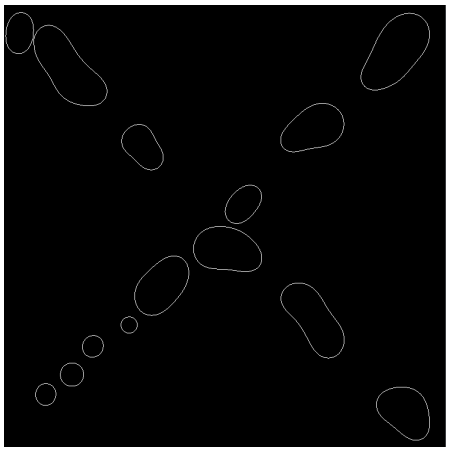
(b) Late stage precipitates.

Figure 4.1: Represented precipitates at early (a) and late (b) stages of the phase-field simulation.

Those shapes represent the silicon precipitates within the eutectic phase. Through the use of the *findContours* function the boundaries of the shapes representing the precipitates are obtained and written in an empty matrix (Fig. 4.2) that will later represent the micrography.



(a) Early stage boundaries.

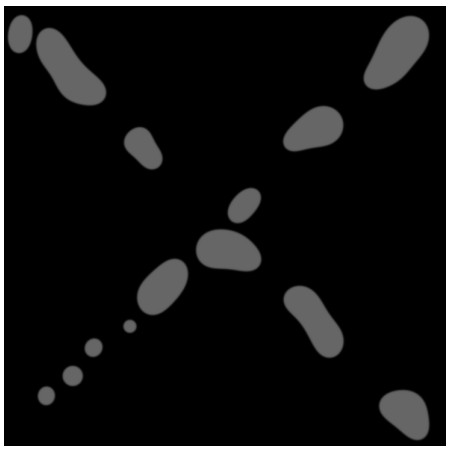


(b) Late stage boundaries.

Figure 4.2: Precipitates boundaries at early (a) and late (b) stages of the phase-field simulation.

Once having represented the interfaces the X_{wSi} variable is loaded into an empty matrix and scaled in such a way that the highest value becomes 255 (maximum brightness of a pixel) and the rest are proportionally rescaled. The representation of the silicon solute is shown in Fig. 4.3, where it is clear that the regions match with the previously obtained precipitate boundaries due to the high Si concentration in the eutectic phase.

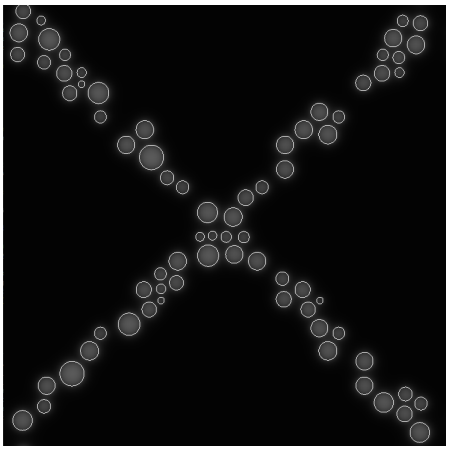
(a) Early stage Si solute distribution.



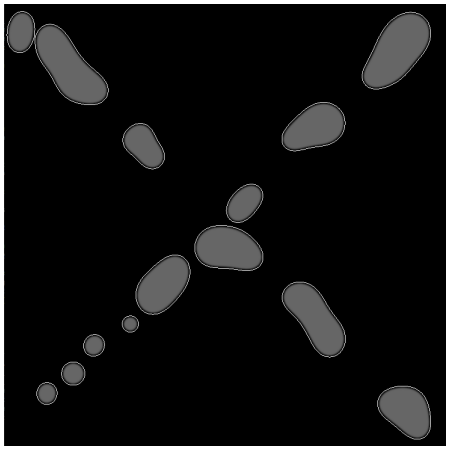
(b) Late stage Si solute distribution.

Figure 4.3: Silicon solute distribution at early (a) and late (b) stages of the phase-field simulation.

Finally, the η and X_{wSi} matrices are superposed and the final micrography is obtained (Fig. 4.4). After post-processing all phase-field files the microghaphies are saved in a parallel data base to be later introduced in the variational autoencoder, as described in Chapter 5.



(a) Early stage micrography.



(b) Late stage micrography.

Figure 4.4: Final micrographies at early (a) and late (b) stages of the phase-field simulation.

4.2 Parameters of interest

Turning back to the needs of the regression model training it is clear that the next step is to obtain the physical features of the microstructure in order to calculate the yield stress. After trying several methods for finding the best possible quality of the results, the conclusion has been to directly analyze the image generated in the previous section (Fig. 4.4), together with the original output files containing the η and X_{wSi} variables.

The parameters that are required to extract from the image are the necessary ones to solve the equations for obtaining the yield stress, given in Chapter 2. Not all of them are possible to obtain from the micrography, as values such as the dislocation density, ρ, can not be obtained through an image. Therefore, the parameters extracted from the image are:

- **Weight fraction of silicon,** X_{wSi}: quantity of Si in solid solution.
- **Volume fraction of cell phase,** f_{cell}: volume fraction that the cellular phase occupies.
- **Volume fraction of eutectic phase,** f_{eu}: volume fraction that the eutectic phase occupies.
- **Volume fraction of the precipitates,** X_v^d: volume fraction occupied by the precipitates. This value is calculated individually for both the cellular and eutectic phases.
- **Mean radius of the precipitates,** $\bar{r}_d$: average equivalent radius of the precipitates. As most of the precipitates are not circles, the equivalent radius of the precipitates is taken as the radius of a circle with the same surface as the precipitate. This value is calculated individually for both the cellular and eutectic phases.
- **Mean distance between precipitates,** l: average distance between each precipitate and its closest neighbor. After trying several methods, the most accurate results are obtained by directly computing the value from r_d and X_v^d rather than using computer vision.

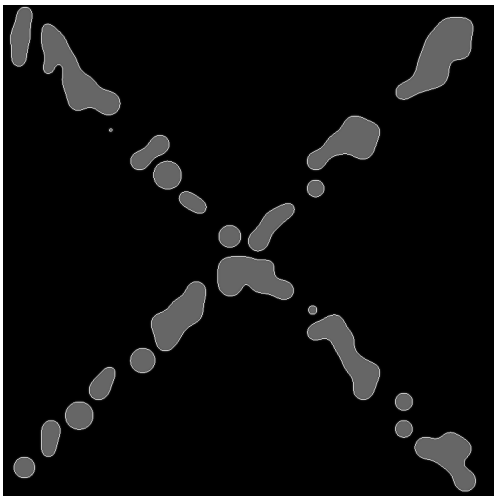
Further description of the obtaining of these parameters is given in Section 4.4. The rest of the necessary parameters are obtained from Delahaye (2022) under the hypothesis of considering the microstructures of both projects similar.

4.3 Structure determination

As it is explained in Chapter 2 the yield stress of the cellular and eutectic phases are calculated separately, followed by an homogenization process. Therefore, the parameters for each phase are different and should be obtained individually. Before extracting the parameters it is necessary to differentiate the cellular phase from the eutectic phase and, as the image only contains the precipitates and the distribution of Si solute, it may be seen as an arbitrary parameter to designate. To stick as much as possible to experimental references, the parameters that define the regions were adjusted to obtain similar values of volume fraction distribution than in real microghraphies.

The procedure starts with the micrography, shown in Fig. 4.4, which is passed through a blurring filter by using the *GaussianBlur* function from the *OpenCV* package. This, briefly explained, lowers the brightness of the precipitates in the image while it increases the brightness of the regions surrounding them. The result of applying a blurring filter is shown in Fig. 4.5.

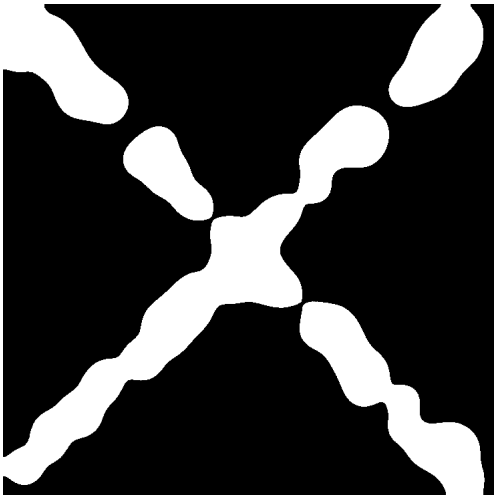
The blurred image is then passed through a threshold filter, where pixels with a brightness above certain value are increased to the maximum, 255, and the rest are dropped to 0. This threshold value is specifically selected so that the eutectic and cellular regions have a similar volume fraction value than in real micrographies, between 0.75-0.85 for f_{cell} and 0.15-0.25 for f_{eu}. The image in Fig. 4.6a illustrates the division between cellular region (black) and the eutectic region (white), while in Fig. 4.6b the precipitates inside the eutectic region, together with the interface that is designated by the white line, are represented. In this specific micrography the values of both regions volume fractions are $f_{cell} = 0.77$ and $f_{eu} = 0.23$.



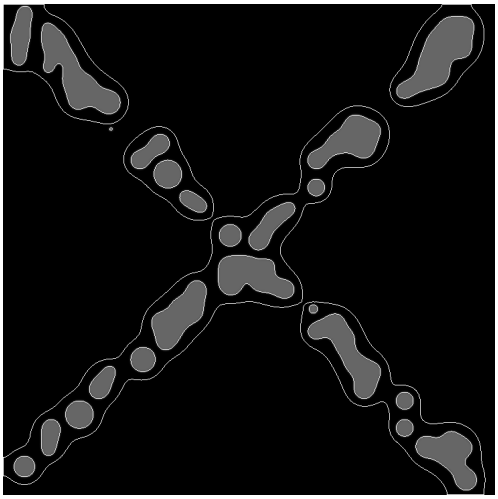
(a) Original micrography.

(b) Blurred micrography.

Figure 4.5: Micrography of early stage simulation (a) passed through a blurring filter (b).



(a) Micrography with differentiated phases.



(b) Original micrography with delimited phases.

Figure 4.6: Micrography once thresholding is applied (a) and with delimited regions (b).

The code section that computes this image processing is given in Fig. 4.7, where the blurring, the thresholding and the drawing of contours are carried out.

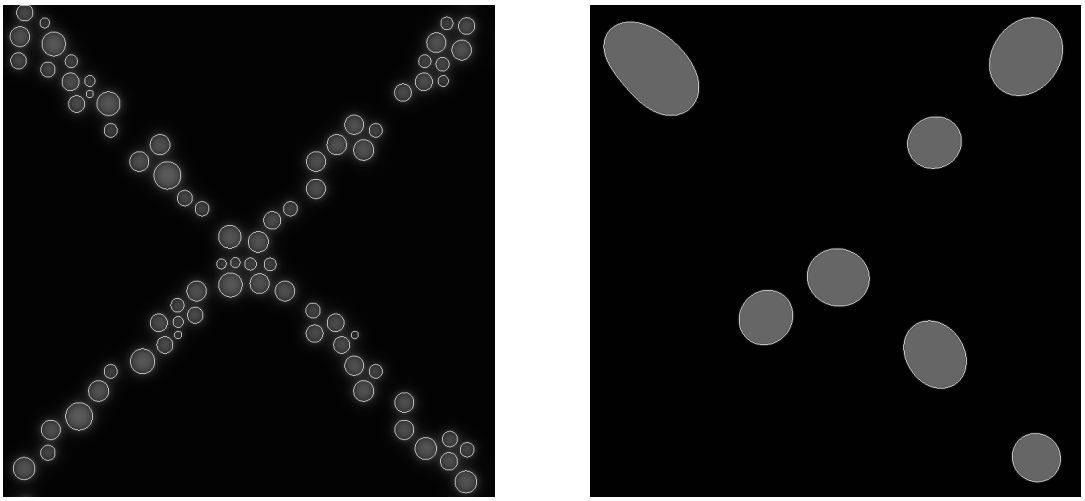
```
265  # Identify the two phases
266  blur = cv2.GaussianBlur(micro, (101, 101), 0) # Blur the image to create the eutectic phase
267  _, thresh = cv2.threshold(blur, 15, 255, cv2.THRESH_BINARY) # Apply thresholding to reveal the eutectic phase
268
269  # Draw the boundaries between cell and eutectic
270  contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
271  for cnt in contours:
272      cv2.drawContours(micro, [cnt], 0, (255, 0, 0), 1)
```

Figure 4.7: Code snippet to obtain the cellular and eutectic regions.

Once the two regions are identified, the parameters for the calculation of the yield stress ($\bar{r}_d$, X_{wSi}, X_v^d, ...) are obtained only analyzing their corresponding region.

Other structure modelling possibilities

The model chosen of a cellular structure surrounded by an eutectic phase (cell+eu) proved good results, that are later shown in Section 4.4. Nevertheless, from the visual evaluation of the microstructures as they change through the course of the simulation, the structure may not be considered a cell+eu one, as the eutectic wall tends to disappear, leaving instead an α-Al matrix with Si precipitates structure (mat+prec). This can be appreciated in Fig. 4.8, where an early stage simulation micrography is compared to a late stage one. The morphology of the structure and the size and distribution of the precipitates is completely different, making unclear that the same equations for calculating the yield stress that were shown in Chapter 2 can be applied.



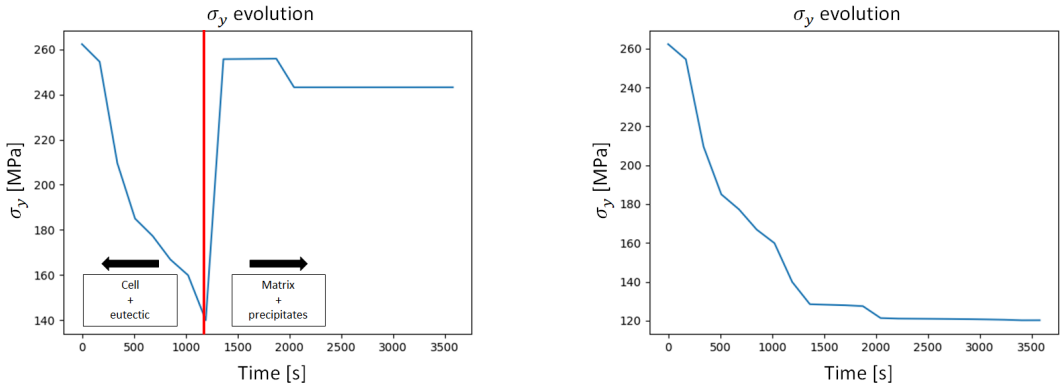
(a) Early stage micrography. (b) Late stage micrography.

Figure 4.8: Early stage simulation micrography (a) compared to late stage simulation micrography (b). Structures are completely different.

The solution proposed for this uncertainty was to define two types of structures. The first one, a biphasic structure of cellular phase and eutectic phase for early stages of the simulation, where the equations of Chapter 2 can be applied. The second one, an α-Al matrix with Si precipitates for late stages of the simulation. This second type of structure is considered, seen from the equation solving point of view, 100% cellular phase with Si precipitates inside it, therefore the equations for obtaining the yield stress are the ones used for the cellular phase (Eq. (2.2), Eq. (2.3), Eq. (2.6)), together with the values $f_{cell} = 1$ and $f_{eu} = 0$ of volume fraction for the homogenization (Eq. (2.7)). Finally, the distinction between one phase and the other is given by a structure defining parameter, whose value is higher in cell+eu cases and lower in mat+prec cases, with a threshold value that, when reached, changes the set of equations to predict the yield stress.

Despite the development of this new model, the results were not the ones expected. In Fig. 4.9a is shown a graph that calculates the value of the yield stress throughout a whole phase-field simulation (for all the microstructures extracted from the simulation) with $\dot{T} = 50$. The red vertical line indicates the change between one set of equations (cell+eu on the left) to another (mat+prec on the right).

There is a clear step between the values calculated by one set of equations and the other, with a difference of 120 MPa. Meanwhile, the graph in Fig. 4.9b shows the calculation only done with the set of equations of the cell+eu structure, having much more reasonable values and a smoother evolution.



(a) Two structures model (cell+eu and mat+prec). (b) One structure model (cell+eu).

Figure 4.9: Yield stress evolution for a simulation with $\dot{T} = 50$ in both models of two (a) and one (b) set of equations. The red vertical line in (a) indicates the change in the use of equations.

Taking all this into account it is clear that, despite the uncertainty of the validity of the cell+eu set of equations for the whole evolution of the microstructure, the best option is to stick to a model of only one set of equations. As a conclusion for this small section, all the calculations done regarding yield stress prediction are done through the methodology described in Chapter 2 with a microstructure considered as a cellular phase surrounded by eutectic walls (cell+eu).

Even though there is progress and research that can be done in this direction, the main objective of this project is to determine the viability of using machine learning algorithms to link micrographies to a value of the yield stress. Therefore, the exact value of the yield stress does not have a determining effect on the final results.

4.4 Parameters determination

As explained in Section 4.2 the parameters are extracted from the micrography constructed out of the phase-field output files. Each parameter is obtained through a particular method, explained in every subsection and validated by showing its evolution in a complete phase-field simulation. They are also compared to values from real microstructures, all referenced to the ones obtained in Delahaye (2022).

The testing simulation launched to show the results is the evolution of the first microstructure included in the data base (Fig. 3.3a) with a value of $\dot{T} = 50$.

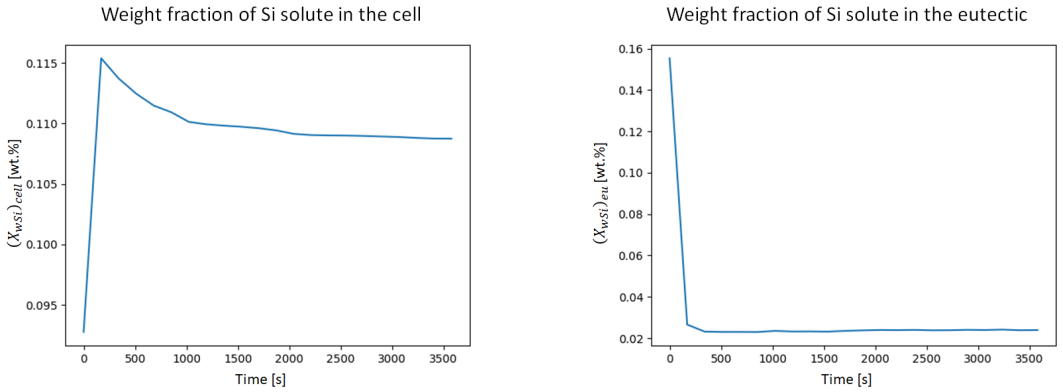
4.4.1 Weight fraction of silicon

The weight fraction of silicon is the only parameter that is not extracted from the image itself, but from the X_{wSi} variable outputted by the phase-field simulation. Each node or pixel, when talking about images, is contained either in the cellular or the eutectic phase and has assigned the quantity of Si solute in it. This is determined by the phase differentiation made in Section 4.3. The mean value of the Si solute quantity is calculated from each set of pixels, obtaining $(X_{wSi})_{cell}$ and $(X_{wSi})_{eu}$. The *Python* lines that follow this procedure are shown in Fig. 4.10, where the parameter is obtained for both phases.

```
# Weight fraction of silicon
Xw_Si_cell_vec = []
Xw_Si_eu_vec = []
for idx1, line in enumerate(thresh_no_prec):
    for idx2, number in enumerate(line):
        if thresh_no_prec[idx1][idx2] == 255:
            Xw_Si_eu_vec.append(X_out[idx1][idx2])
        else:
            Xw_Si_cell_vec.append(X_out[idx1][idx2])

Xw_Si_cell = np.mean(Xw_Si_cell_vec) # Weight fraction of Si in the cell
Xw_Si_eu = np.mean(Xw_Si_eu_vec) # Weight fraction of Si in the eutectic
parameter_list[i].update({'Xw_Si_cell':Xw_Si_cell})
parameter_list[i].update({'Xw_Si_eu':Xw_Si_eu})
```

Figure 4.10: Code snippet to calculate the variables $(X_{wSi})_{cell}$ and $(X_{wSi})_{eu}$.

The code snippet shows the nested loop used to count the quantity of Si in each pixel from the X_out matrix, which is, as mentioned before, the variable directly outputted by phase-field that provides the Si quantity in the microstructure. The pixels are first classified into the ones belonging to the eutectic phase or the cellular phase, and then the mean value of X_{wSi} is computed for each set. In Fig. 4.11 are represented the evolution of $(X_{wSi})_{cell}$ and $(X_{wSi})_{eu}$ for the already mentioned example simulation.



(a) $(X_{wSi})_{cell}$.

(b) $(X_{wSi})_{eu}$.

Figure 4.11: Weight fraction of silicon evolution in cellular (a) and eutectic (b) phases.

The order of magnitude of the value obtained is much lower that the one of reference in Table 2.1. Nevertheless, the value is taken as valid as it is the data directly outputted from the phase-field results.

4.4.2 Volume fraction of cell phase and eutectic phase

The volume fraction of the phases has a similar method of obtaining than the weight fraction of silicon. In this case the image from where it is extracted is Fig. 4.6a. The white pixels are counted one by one, and then the number is divided by the overall quantity of pixels, obtaining the volume fraction of eutectic phase.

$$f_{eu} = \frac{Sum\ of\ white\ pixels}{Sum\ of\ all\ pixels} \tag{4.1}$$

The value of f_{cell} is the complementary.

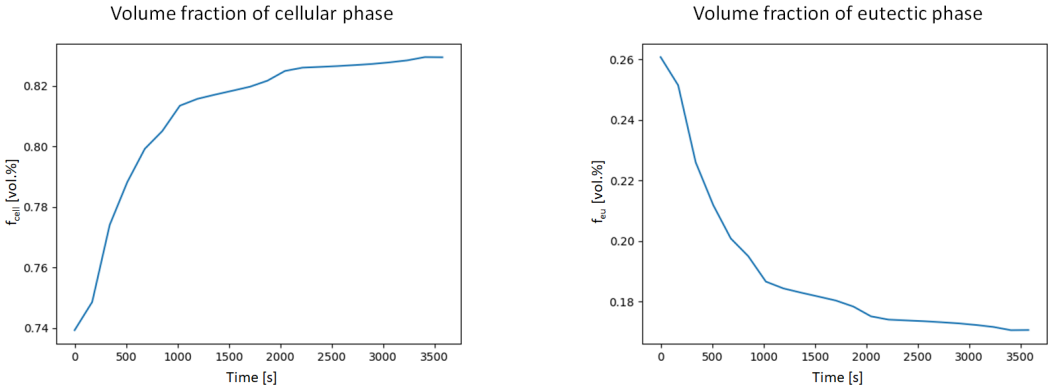
$$f_{cell} = 1 - f_{eu} \tag{4.2}$$

Fig. 4.12 shows the code snippet that calculates both variables.

```
# Volume fraction of cell and eutectic
counter = 0
for idx1, line in enumerate(thresh):
    for idx2, elem in enumerate(line):
        if elem == 255:
            counter += 1

frac_eu = counter/(np.shape(eta_out)[0]*np.shape(eta_out)[1]) # Volume fraction of cell
frac_cell = 1 - frac_eu # Volume fraction of eutectic
parameter_list[i].update({'frac_cell':frac_cell})
parameter_list[i].update({'frac_eu':frac_eu})
```

Figure 4.12: Code snippet to calculate the variables f_{cell} and f_{eu}.

The code counts the pixels in white and black of the image that defines the two phases (Fig. 4.6a), and then computes the volume fraction for each. The values obtained for the example simulation are shown in Fig. 4.13, which are very close to the reference from Table 2.1.



(a) f_{cell}. (b) f_{eu}.

Figure 4.13: Volume fraction of cellular (a) and eutectic (b) phases out of the overall microstructure.

4.4.3 Volume fraction of the precipitates

The volume fraction of the precipitates follows the same exact procedure as the volume fraction of the phases, but the image used is the one shown in Fig. 4.1, which represents the precipitates as white shapes. Each phase region has its own value of X_v^d. However, due to the morphology of the microstructure (not real) and the way the regions of the phases are designated, the value of $(X_v^d)_{cell}$ is always 0, meaning that there are no precipitates in the cellular phase. This assumption agrees with the values of reference in Table 2.1 but it is worth mentioning that this is not always the case, as Si precipitates can also be found within the cellular phase (Delahaye (2022)). The procedure is coded in Fig. 4.14, where the *filled_cont* matrix is the image that shows the precipitates in white and the background in black, as in Fig. 4.1.

```
# Volume fraction of precipitates
img = cv2.imread(save_img_name, cv2.IMREAD_GRAYSCALE).astype(float) # OpenCV takes images with better image quality this way
img *= (255.0/img.max()) # Rescale for better contrast
for idx1, line in enumerate(img): # Remove boundaries of the eutectic to increase quality
    for idx2, element in enumerate(line):
        if element > 250:
            img[idx1][idx2] = img[idx1][idx2+1]
_, img_thresh = cv2.threshold(img, 40, 0, cv2.THRESH_TOZERO) # Threshold to better define the precipitates
img_thresh = thresh_radius
img_thresh = (np.rint(img_thresh)).astype(np.uint8)
img_contours, _ = cv2.findContours(img_thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE) # Obtain precipitates' contours

filled_cont = np.zeros((np.shape(img)[0],np.shape(img)[1]))
for cnt in img_contours: # Draw filled precipitates' contours to count
    cv2.drawContours(filled_cont, [cnt], 0, (255, 0, 0), -1)

counter = 0
for idx1, line in enumerate(filled_cont):
    for idx2, elem in enumerate(line):
        if elem == 255:
            counter += 1

Xdv_eu = counter/(np.shape(eta_out)[0]*np.shape(eta_out)[1])/frac_eu
Xdv_cell = 0
parameter_list[i].update({'Xdv_cell':Xdv_cell})
parameter_list[i].update({'Xdv_eu':Xdv_eu})
```

Figure 4.14: Code snippet to calculate the variable $(X_v^d)_{cell}$.

The evolution of $(X_v^d)_{eu}$ is shown in Fig. 4.15. As a remark for the reader, the value represents the volume fraction of precipitates over the volume of eutectic phase and not over the overall volume of the microstructure.

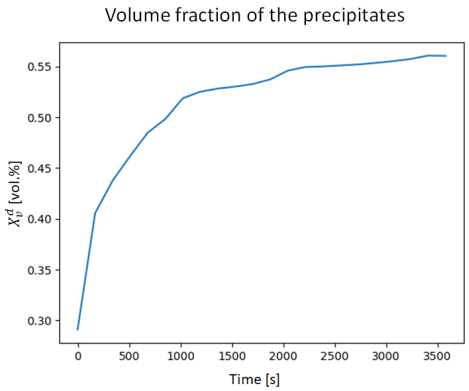


Figure 4.15: Volume fraction of the precipitates in the eutectic phase.

4.4.4 Mean radius of the precipitates

As it is described in previous sections (Section 4.2) it is the mean value of the equivalent radius of the precipitates. The first method suggested was to use the *findContours* function from *OpenCV* in Fig. 4.1, that enables the detection of shapes in an image, together with the *contourArea* function, that directly computes the area. The performance of this method was not the one expected due to a bad shape detection.

Instead of trying to improve the previous method an equivalent way to obtain the mean radius is proposed. The technique consists on, by also using the Fig. 4.1 image, count all the white pixels (acting as a small piece of area each) and divide it by the number of precipitates, obtaining the mean area of the precipitates. The result is also multiplied by the scale factor, dx, to transform the units from pixels to meters. The average area of the precipitates is given by:

$$\bar{A} = \frac{Sum\ of\ white\ pixels}{Number\ of\ precipitates} * dx^2 \tag{4.3}$$

with $\bar{A}$ being the mean area of the precipitates. Then the formula of the area of the circle is used to obtain the mean radius.

$$\bar{r}_d = \sqrt{\frac{\bar{A}}{\pi}} \tag{4.4}$$

This value should be calculated for both the cellular and eutectic phases. Nevertheless, as mentioned in the previous subsection, the cellular phase is assumed to have no Si precipitates and thus the value of $(\bar{r}_d)_{cell}$ is taken always as 0. The brief code snippet that calculates the mean radius is shown in Fig. 4.16, where the variable *counter* is the same as in Fig. 4.14, representing the sum of all the pixels that belong to precipitates.

```
341 # Mean radius
342 r_d_eu = np.sqrt((counter/np.size(img_contours))/np.pi)*dx
343 r_d_cell = 0
344 parameter_list[i].update({'r_d_cell':r_d_cell})
345 parameter_list[i].update({'r_d_eu':r_d_eu})
```

Figure 4.16: Code snippet to calculate the variable $(\bar{r}_d)_{cell}$.

The evolution of the $(\bar{r}_d)_{eu}$ variable throughout the example simulation is given in Fig. 4.17.

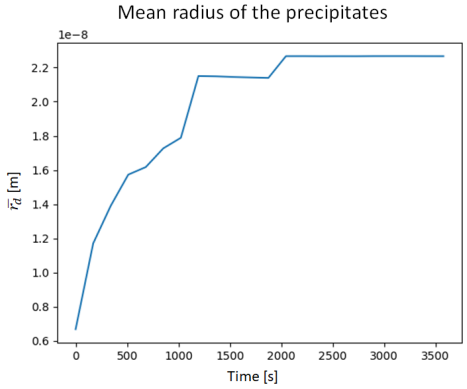


Figure 4.17: Mean radius of the precipitates in the eutectic phase.

The value of the mean radius increases from something close to the reference value, which is $\bar{r}_d = 7$ nm, to values 3 or 4 times larger, an evolution that will later have a great impact in the precipitation hardening contribution, and thus to the overall yield stress.

4.4.5 Mean distance between precipitates

The l parameter is another one with controversy, as at the beginning the procedure was as simple as computing the mean distance between the centers of each precipitate and its closest neighbor, later subtracting the values of the radii.

$$l = \frac{\sum_{i=1}^{N} \left[\sqrt{(c_{xi} - c'_{xi})^2 + (c_{yi} - c'_{yi})^2)}\right]}{N} - 2\bar{r}_d \tag{4.5}$$

Results from this method are not the ones expected, as, even if the mathematical approach is simple and intuitive for the current problem, the mean distance presented an unsteady evolution at the early stages of the simulation, which completely changed the later results of σ_y. This evolution is shown in Fig. 4.18.

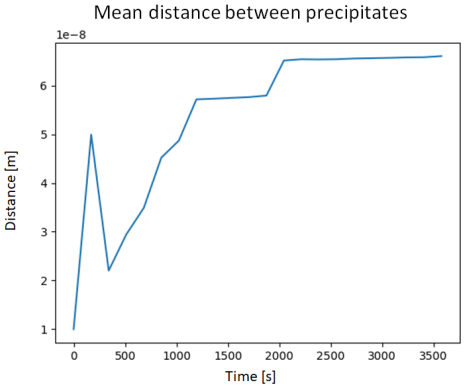


Figure 4.18: Mean distance between the precipitates calculated with the dropped method.

Another approach is taken by searching in the bibliography, as many authors define the mean distance between precipitates, l, as a function of the mean radius, $\bar{r}_d$, and the volume fraction of the precipitates, X_v^d (Macías et al. (2020)):

$$l = \bar{r}_d \sqrt{\frac{2\pi}{3X_v^d}} \tag{4.6}$$

This method, whose results are shown in Fig. 4.19, presents an apparently more stable evolution, and the order of magnitude of the parameter is the same than in both the results of the previous method and the reference values of Table 2.1.

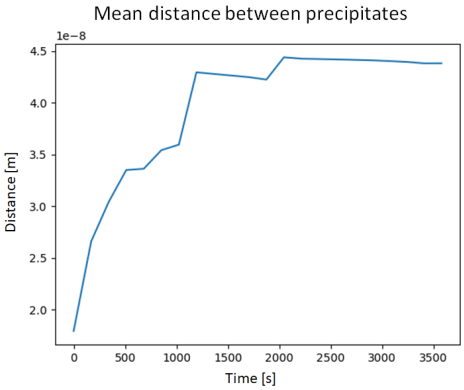


Figure 4.19: Mean distance between the precipitates calculated with the selected method.

The one line code snippet of the l formula is given in Fig. 4.20.

```
# Mean distance between precipitates
l = r_d_eu*np.sqrt(2*np.pi/3/Xdv_eu)
parameter_list[i].update({'l':l})
```

Figure 4.20: Code snippet to calculate the variable l.

4.5 Results on the yield stress

Once the parameters for the yield stress calculation have been obtained the prediction of its value through the method described in Chapter 2 can be carried out. The same set of micrographies from the example simulation that is used in the previous sections (initial microstructure number 1 and $\dot{T} = 50$) is analyzed and the evolution of the yield stress is shown in Fig. 4.21:

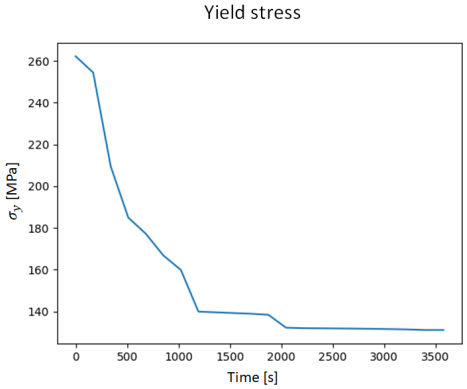


Figure 4.21: Yield stress evolution.

The yield stress, as it is expected, presents a sharp drop at the early stages of the simulation, as it is expected because of the strong diffusion and the coarsening of the precipitates. This is due to the high temperatures that the alloy is bearing, making its structure to change and the eutectic wall, the main source of strengthening, to gradually disappear. In the late stages of the simulation the microstructure stabilizes, presenting larger, more disperse and less numerous precipitates that do not have such a strong hardening effect. As a visual aid for understanding the evolution of the microstructure and associate it to the evolution of the yield stress, Fig. 4.23 shows the set of micrographies that represent the evolution of the microstructure in the example simulation, observing a very strong difference between the beginning and ending stages.

Finally, even though one set of results is from a real microstructure and the other one is from a simulated microstructure, a comparison between the results in this project and the ones in Delahaye (2022) regarding the hardening contributions of the yield stress is carried out. From the simulated microstructures, the one saved at the first time step is chosen to be analyzed, as it has more similar characteristics to the real one due to not being exposed to heat treatments and keeping the original cellular-eutectic structure. Parameters used for the reference computation are given in Table 2.1 while the microstructure simulated in this project has the set of parameters shown in Table 4.1. The comparison of the results is shown in Fig. 4.22, where it is clear that the hardening effects are very different.

The strengthening contributions of σ_0 and σ_d are the same because of the use of the same parameters in both calculations, while the contributions of σ_{ss} decrease and the ones of σ_p greatly increase. This difference is easily explained when looking at the difference of the parameters used. On one hand, the lower σ_{ss} is caused by a much smaller quantity of Si solute in the eutectic (2.5% in the real alloy against 0.15% in the simulated one) and the non-existence of Mg solute, as the simulated alloy does not contain magnesium. On the other hand, the difference in the σ_p contribution comes from the distance between precipitates parameter, l, which is much larger in the real microstructure ($\sim$50 nm) than in the simulated one ($\sim$20 nm).

Parameters	Units	Value	
		Cell	**Eutectic**
σ_0	$[MPa]$	10	
ρ_{d0}	$[m^{-2}]$	10^{13}	
f	$[vol.\%]$	0.74	0.26
$\bar{r}_d$	$[nm]$	0	6.688
X^{α}_{wSi}	$[wt.\%]$	0.093	0.155
X^{d}_{v}	$[vol.\%]$	0	29.1
G	$[GPa]$	25.4	
b	$[nm]$	0.283	
M	$[-]$	3.06	
α_d	$[-]$	0.3	
α_p	$[-]$	0.5	
k_{Si}	$[Mpa.Wt\%2/3]$	66.3	
k_{Mg}	$[Mpa.Wt\%2/3]$	29	
r_c	$[nm]$	10	
l	$[nm]$	-	17.9

Table 4.1: Parameters for yield stress prediction in simulated microstructure at $t = 0$ and $\dot{T}$ = 50 K/min.

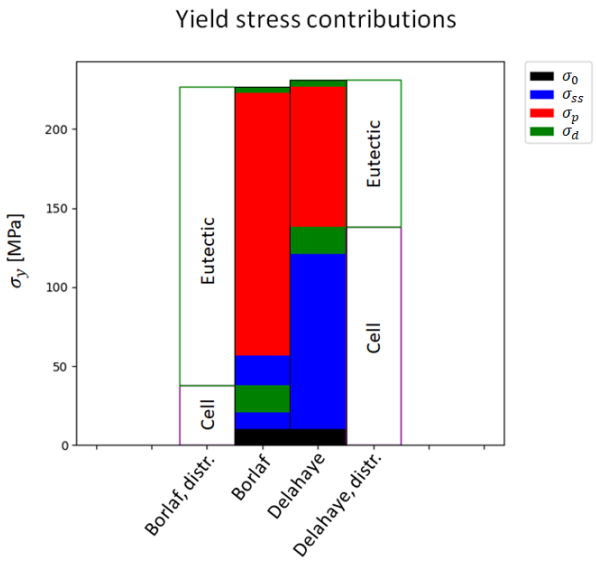


Figure 4.22: Yield stress contributions for simulated microstrucure results and Delahaye (2022) results.

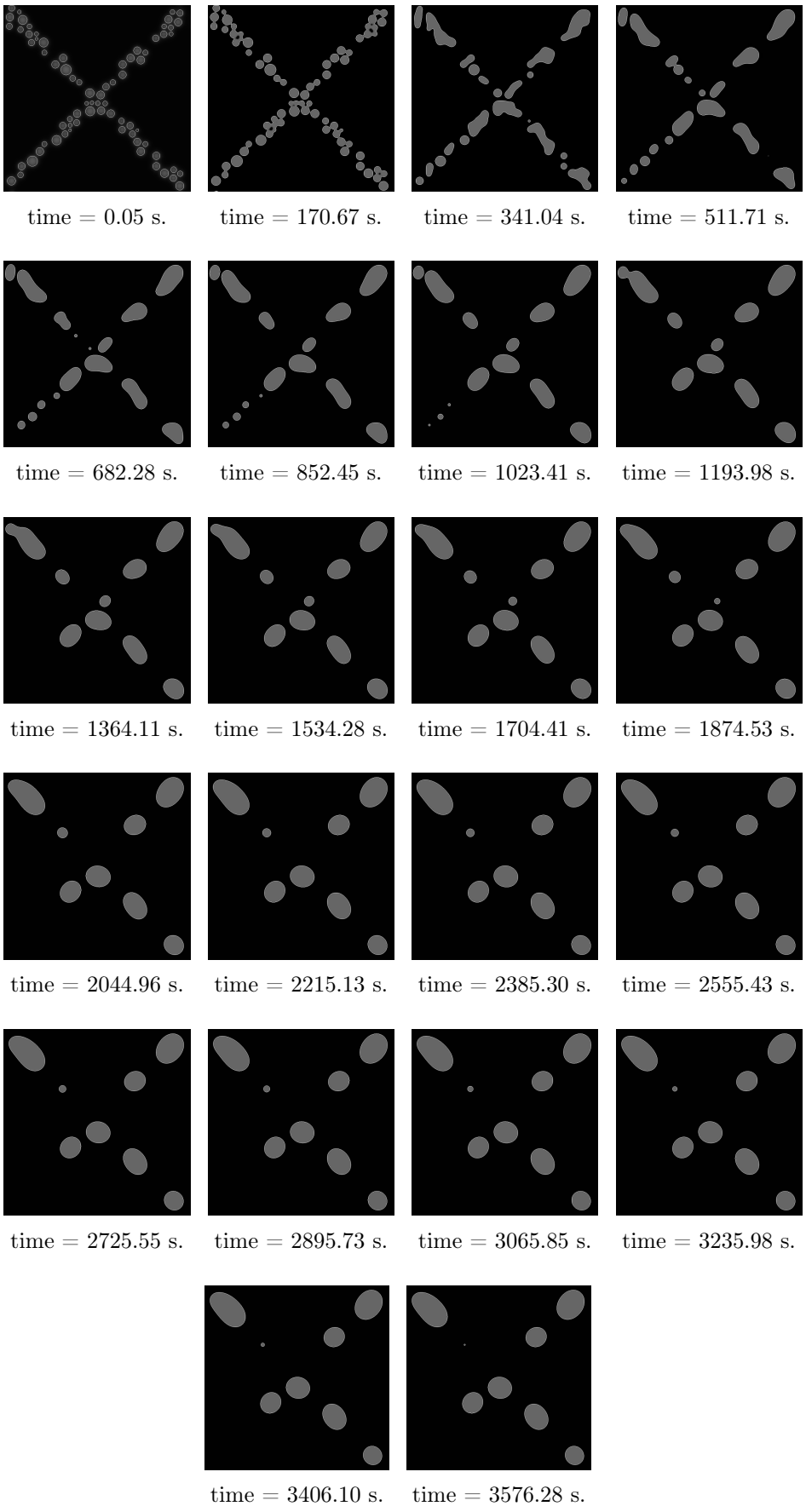


Figure 4.23: Microstructure evolution through phase-field simulation. Initial microstructure number 1 and $\dot{T} = 50$.

Chapter 5

Variational autoencoder

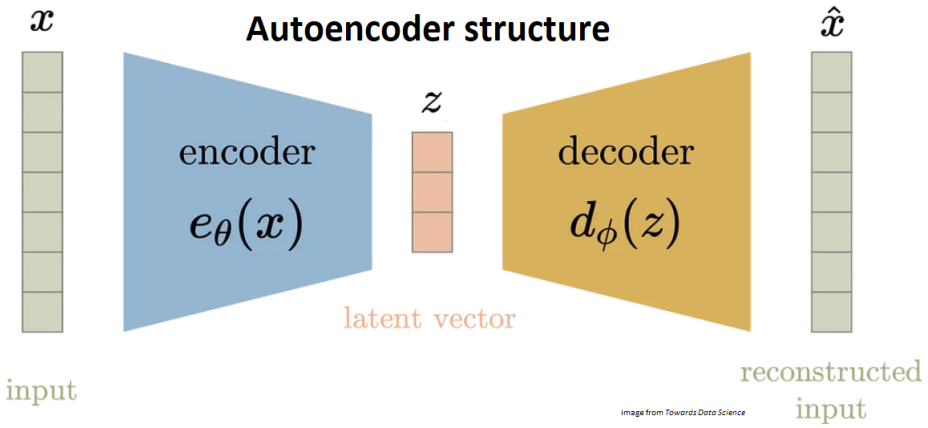
As a way to reduce the size of the training data set and the output size of the regression algorithm the Variational Autoencoders (VAE) are very powerful tools that, through the use of neural networks, enable the encoding of full resolution images to a small set of variables. This, first of all, makes possible the training of the prediction algorithm, as too large output sizes make non-viable for it to properly work. Further than that, a smaller output size and therefore a lighter training data volume achieve a faster training of the algorithm and smaller requirements of computational resources.

This chapter shows the whole process of construction of the VAE, giving a small state of the art introduction to them in Section 5.1, together with the features that make it a good choice for using it in this project. After that, the structure of the VAE that has been built is explained in Section 5.2, detailing the number and size of the neural network layers as well as their characteristics. Section 5.3 explains the training process that the VAE has had to be able to successfully encode (reduce to a small latent vector) and decode (reconstruct) micrographies. Finally, Section 5.4 shows the results of phase-field micrographies that have been encoded and decoded, and a conclusion talking about the performance and the aspects to improve.

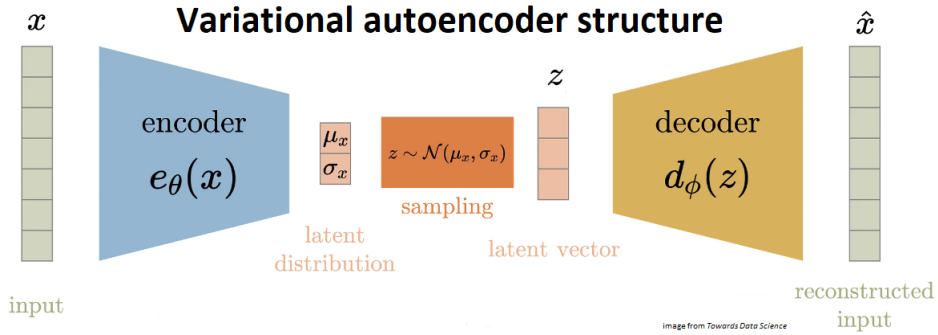
5.1 Introduction to VAE and purpose to use them

Variational Autoencoders (Kingma and Welling (2013), Kingma et al. (2019)) are artificial neural network architectures that belong to the family of probabilistic graphical models (Airoldi (2007)). These models are able to expresses a conditional dependence structure between random variables, enabling them to establish mathematical or numerical relations between apparently independent variables.

The VAE are also sometimes related to regular Autoencoders (AE) (Kramer (1991)) due to their similar structure, as both of them are neural network structures divided in two parts, one for encoding and one for decoding. Nevertheless, the mathematical formulation of both schemes is different, as AEs reduce the input directly to a latent vector (the set of variables that contain the main features of the input, in our case an image), while the VAEs reduce the input to a latent distribution and impose a constraint on it, forcing it to follow a normal distribution. A graphical scheme of both methods is shown in Fig. 5.1 to better understand the structure of the algorithms.



(a) Autoencoder structure.



(b) Variational autoencoder structure.

Figure 5.1: Structures of AE and VAE.

VAEs have a complex structure consisting of a set of much more simple elements:

- **Input**: is the original data introduced that wants to be reduced to a smaller latent vector.
- **Encoder**: a set of neural layers that, though a series of mathematical operations reduce the input values until arriving to a small sized set of variables. It is important to take into account that these variables are completely dependant on the values of the input layer, therefore they can be considered as a reduced representation of them.
- **Latent distribution**: the set of variables directly obtained after the encoding, given as a set of parameters of a predefined distribution in the latent space (μ_x and σ_x).
- **Sampling**: layer in the latent space where the latent distribution is imposed a normal distribution. This makes sure the latent space is regularized.
- **Latent vector**: the set of defined parameters that are seen as a reduced representation of the overall input. The size of this layer depends on the size of the input and the quality that wants to be obtained at the reconstruction (larger latent vectors mean a better reconstruction) but it usually goes down to 8 to 64 numbers.
- **Decoder**: the set of neural layers that reconstruct the latent vector into something as similar as possible to the input given to the VAE.
- **Reconstructed input**: the information reconstructed from the latent vector. It should be as similar as possible to the input of the VAE.

VAEs are used for a wide variety of applications, standing out image generation, data compression, and even generation of realistic faces, objects, and more. Considering these

particular properties and the structure of the algorithms, new images can be generated out of already trained VAEs by introducing latent vectors that are later decoded. They are particularly accurate when the images that are going to be predicted are similar to the training images, but it is not always the case.

Applications have also been found for multiple purposes when using the VAEs as a tool. These applications may go from predictive modeling of genetic and drug-induced perturbations (Doncevic and Herrmann (2023)) to the design of new turbomachinery concepts (Zalger (2017)). Several applications have also been found for the field of material sciences, as VAEs can be very useful for the generation of new microstructures or for dimensionality reduction of micrographies. Examples of these applications with successful results can be observed in articles like Kim et al. (2021), a project very similar to this one where VAEs are used together with regression models to create a set of synthetic microstructural images. Works have been also carried out in the MSM group itself, as projects like Fetni et al. (2023) pursue the acceleration of phase-field computations through dimensionality reduction of the micrographies outputted by the software.

This project aims to, by providing a value of the yield stress, being capable of generating a full micrography of an AlSi10Mg alloy. This can be achieved by training a VAE with microstructural images (phase-field images in this case) and providing a latent vector, based on that certain value of the yield stress, to generate the desired micrography using a trained decoder. This could not be done with a machine learning prediction model alone such as a regression model, as the output, which is the set of all pixels of the image, would be too large for the algorithm to handle, reaching bad accuracy and too large training times. For training a regression model that can predict the image, the VAE is used to reduce the dimensionality by compressing it to its main features contained in the latent vector. This latent vector is later fed to the regression model, as it is explained in Chapter 6.

Therefore, this project uses the VAE for two purposes. Firstly, the size reduction of the training data set of the regression model. Secondly, the generation of new synthetic micrographies that represent different states of the AlSi10Mg alloys, everything under the context of phase-field simulations.

5.2 Structure of the VAE

Before talking about the VAE it is necessary to introduce the concept of Artificial Neural Networks (ANN). These are machine learning models that reproduce the behavior of biological neurons, being capable of carrying out very complex calculations by using large numbers of interconnections between individual nodes called artificial neurons. Each neuron receives a series of input values from the previous neurons, makes a calculation defined by internal parameters of the neuron, and provides a new output value that is sent to the next neuron. Neurons are displaced by layers in the network, having an input layer that receives the initial data, a series of hidden layers that carry out the calculations, and an output layer that provides the final result of the calculation (Fig. 5.2).

Neural networks are capable of reproducing all kinds of mathematical functions by combining smaller operations carried out by each single neuron. The parameters within each neuron are defined by training the network providing it examples of the input together with the desired output. ANNs are capable of solving strong non-linear problems, making them a very popular choice nowadays for many prediction tasks. Among those applications, the ANNs are used to build new methods such as the VAE, composed by an encoder and decoder made of several neuron layers that carry out the transformation of images into latent vectors and vice versa.

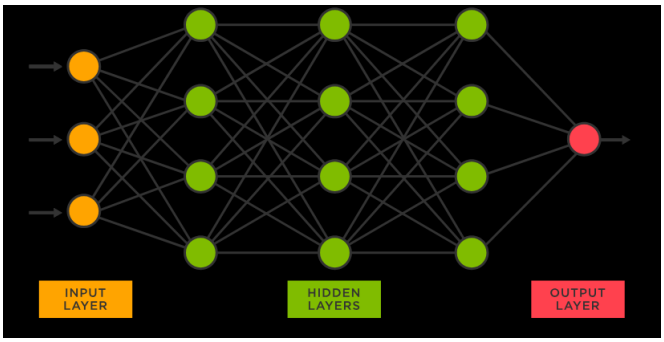


Figure 5.2: Simplified structure of an ANN.

There are infinite ways to shape the structure of a VAE, as the number of neural layers in both the encoder and decoder, as well as the dimensions of each of them, are a choice of the designer. A simple but effective structure has been constructed for the VAE in this project. A clear and complete scheme of it is shown in Fig. 5.3.

As it can be observed, there are different types of layers within the VAE, each of them with a unique function:

Input layer

Is the initial data provided to the VAE for its encoding. In the present case the original images, with a resolution of 735x735 pixels, had to be reshaped into a 128x128 pixels resolution. This carries a loss in quality of the image, but taking into account that the original image would contain more than 500000 input values, a quantity too large to be handled due to the high requirements of RAM memory, it was decided to reshape to the new resolution that inputs around 16000 values. A comparison example of both the original and reduced images is given in Fig. 5.4.

Convolutional layer

They are the main component of the neural network and consist of a set of filters (or kernels) that make operations on certain regions, one after another, of the image or input array. This process is better explained together with Fig. 5.5, where the kernel (composed of trainable parameters) operates a certain input region to extract an output value that will depend on the features of this same region.

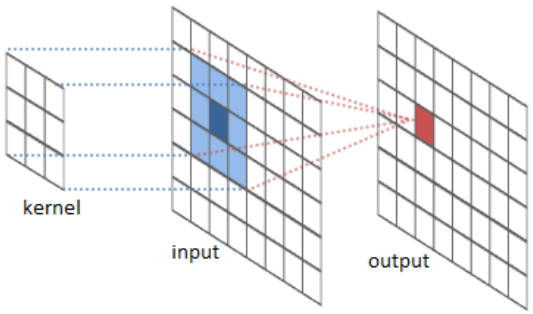


Figure 5.5: Working of a convolutional layer.

These layers have the advantage of containing a smaller set of parameters, making them easier to train, and are extraordinary good for detecting local patterns.

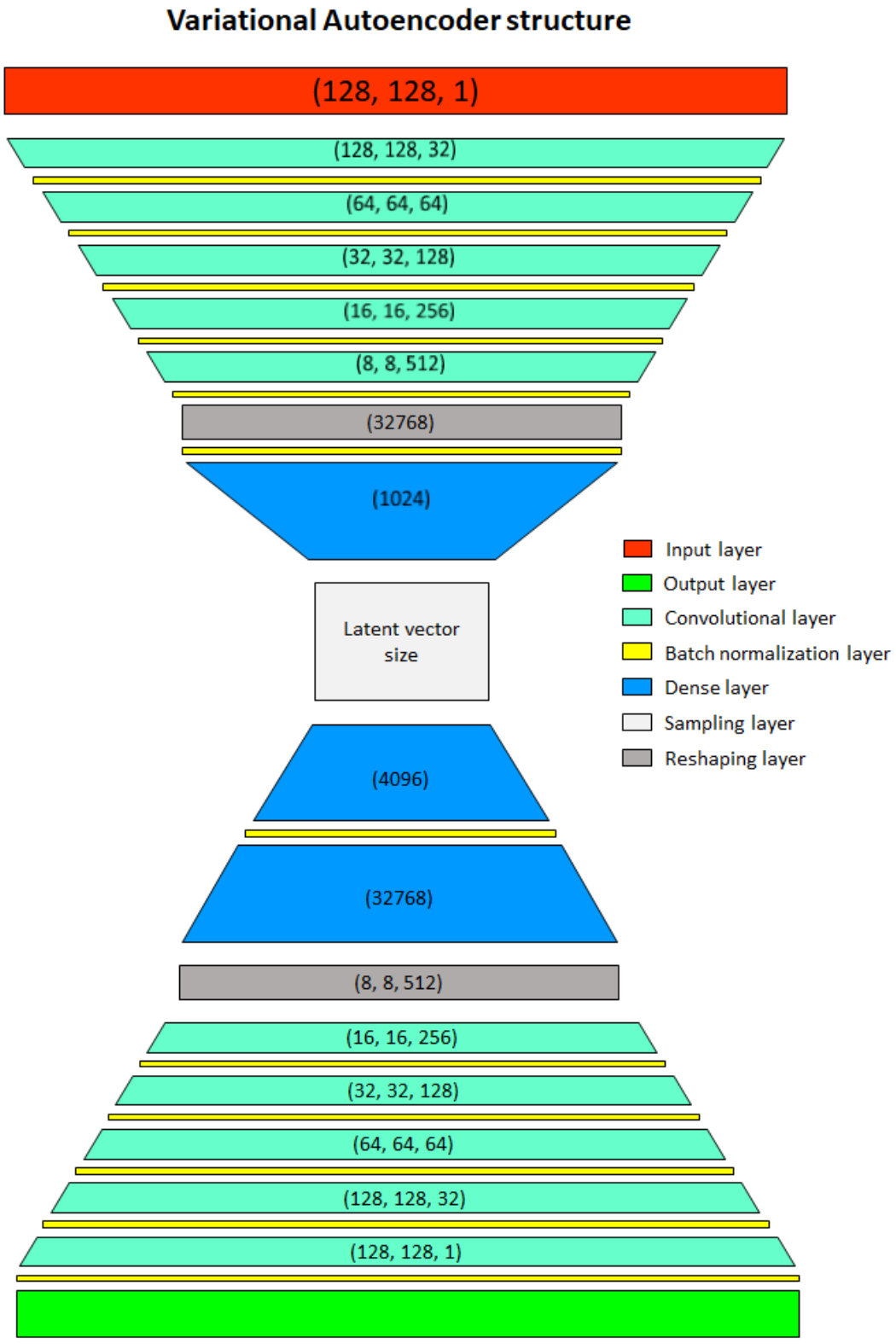
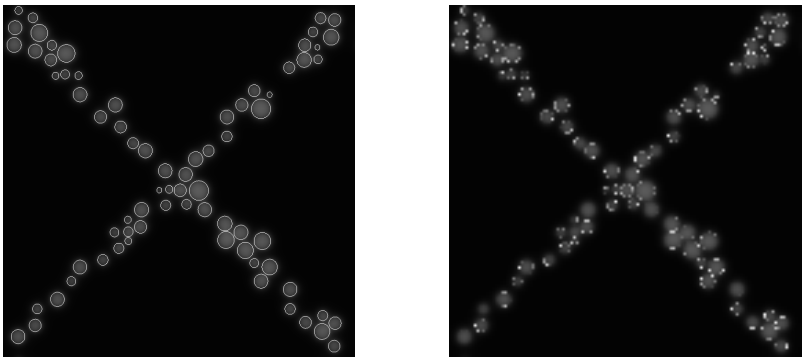
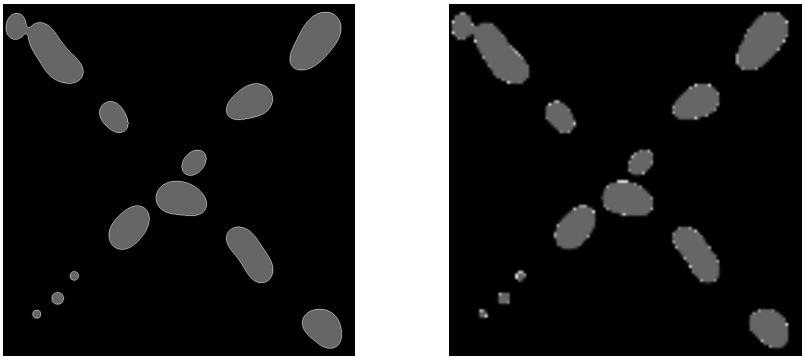


Figure 5.3: Scheme of the VAE used in this project. The vector at each layer indicates the shape of its output array.



(a) Early stage simulation micrographies.



(b) Late stage simulation micrographies.

Figure 5.4: Comparison of two different micrographies at the original resolution of 735x735 pixels (left column) and with reduced resolution of 128x128 pixels (right column).

In the encoder are used the regular convolutional layers but the decoder is built with transposed convolutional layer, who have the same behavior but work in the opposite direction, increasing the size of the output compared to the input.

Batch normalization layer

These layers reshape the values of their input by applying two operations. The first one is a re-centering of the values, changing them proportionally so that they follow a normal distribution centered in 0. The second one is a re-scaling operation, making the values to have a suitable size for an optimal encoding-decoding process.

Those layers do not affect much the final result of the VAE, but they are used to make the training of the artificial neural network faster and more stable through use normalization.

Dense layer

Having a similar procedure than the convolutional layers, the dense layers perform a series of operations in their inputs, but they are deeply connected with its preceding layer, which means the neurons of the layer are connected to every neuron of its preceding layer. A structure of a small neural network composed only by dense layers is shown in Fig. 5.6.

They are introduced in the small-sized parts of the encoder and decoder because they contain a very large set of trainable parameters, therefore they are computationally expensive when having large inputs. The advantage of using them is that they learn features from the entire previous layer, allowing to acquire knowledge more effectively.

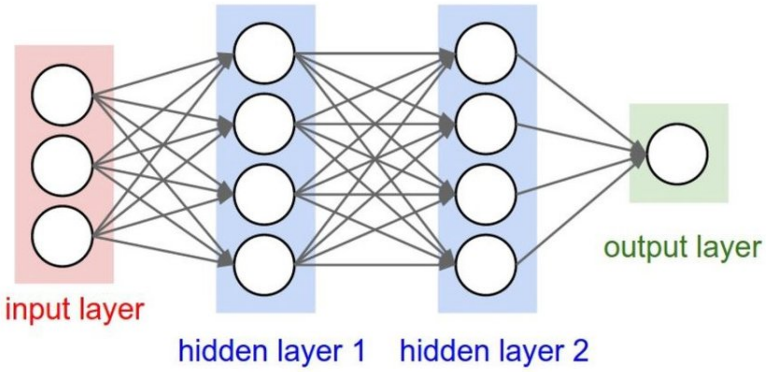


Figure 5.6: Scheme of a neural network composed by dense layers.

Sampling layer

The sampling layer makes the main difference between a regular AE and a VAE. Instead of outputting the vectors in the latent space, the encoder of VAE outputs parameters of a pre-defined distribution in the latent space for every input. The sampling layer then imposes a constraint on this latent distribution, forcing it to be a normal distribution. This constraint makes sure that the latent space is regularized and smooth.

Reshaping layer

These layers do not influence the results of the VAE, as they only perform a reshaping of the provided arrays so that they properly fit the next layer.

Output layer

Is the reconstructed image that has been obtained through the decoding of the latent vector. It has the same shape as the input layer (128x128 pixels).

Looking again at the scheme of Fig. 5.3, the VAE constructed is composed of:

- 1 input layer containing the micrography image.
- 5 convolutional layers that progressively reduce the dimensions of the parameters, together with their corresponding batch normalization layers.
- 1 reshaping layer that flattens the array into a 1-dimensional vector.
- 1 dense layer that greatly reduces the size of the vector.
- 1 sampling layer that imposes a normal distribution to the latent space, obtaining the latent vector of the inputted image.
- 2 dense layers that expand the latent vector in order to start the decoding process.
- 1 reshaping layer that transforms the 1-dimensional vector back into a 3-dimensional array.
- 5 convolutional layers with their corresponding batch normalization layers to decode the latent vector and obtain the reconstructed image.
- 1 output layer containing the reconstructed image.

For creating this neural network structure the *Python* libraries *Keras* and *Tensorflow* have been used. These packages are very powerful machine learning tools that enable the user to create any kind of artificial intelligence algorithm, especially deep neural network structures.

The functions that have been mainly used to build the VAE are *Conv2DTranspose* and *Conv2D* for the convolutional layers, *BatchNormalization* for the batch normalization layers and *Dense* for the dense layers. For the reader to have a better understanding of the building of the different sections in the VAE Fig. 5.7, Fig. 5.8 and Fig. 5.9 show the code snippets of the encoder, sampling and decoder sections correspondingly written in *Python*.

```
# Build the encoder
encoder_input = Input(shape = (128, 128, 1)) # Make an input layer

x = Conv2D(32, (5,5), activation = LeakyReLU(0.02), strides = 1, padding = 'same')(encoder_input) # Use of a convolutional layer
x = BatchNormalization()(x) # Normalize the inputs of the layer

filter_size = [64, 128, 256, 512] # Make convolutional layers with different sizes
for i in filter_size:
    x = Conv2D(i, (5,5), activation = LeakyReLU(0.02), strides = 2, padding = 'same')(x)
    x = BatchNormalization()(x)

x = Flatten()(x) # Convert all feature matrices into vectors
x = Dense(1024, activation = selu)(x) # Attach dense layer
encoder_output = BatchNormalization()(x)
```

Figure 5.7: Code snippet of the encoder.

```
# Sampling layer
mu = Dense(latent_dim)(encoder_output) # Layer to generate mean value tensor
log_var = Dense(latent_dim)(encoder_output) # Layer to generate covariance value tensor

epsilon = K.random_normal(shape = (tf.shape(mu)[0], tf.shape(mu)[1])) # Create the auxiliary noise variable
sigma = tf.exp(0.5 * log_var) # Compute the standard deviation

z_eps = Multiply()([sigma, epsilon])
z = Add()([mu, z_eps])

encoder = Model(encoder_input, outputs = [mu, log_var, z], name = 'encoder') # Bind everything to complete the architecture up to the bottleneck part
encoder.summary()
```

Figure 5.8: Code snippet of the sampling layer.

```
# Build the decoder
decoder = Sequential()
decoder.add(Dense(4096, activation = selu, input_shape = (latent_dim, ))) # First layer taking input from latent vector
decoder.add(BatchNormalization()) # Normalize output

decoder.add(Dense(32768, activation = selu)) # Add a second dense layer with more units to learn more than the previous layer
decoder.add(Reshape((8,8,512))) # Reshape the output to an appropriate size

# Increase the dimesion of the 512 4x4 matrices by adding convolutional layers
decoder.add(Conv2DTranspose(256, (5,5), activation = LeakyReLU(0.02), strides = 2, padding = 'same'))
decoder.add(BatchNormalization())

decoder.add(Conv2DTranspose(128, (5,5), activation = LeakyReLU(0.02), strides = 2, padding = 'same'))
decoder.add(BatchNormalization())

decoder.add(Conv2DTranspose(64, (5,5), activation = LeakyReLU(0.02), strides = 2, padding = 'same'))
decoder.add(BatchNormalization())

decoder.add(Conv2DTranspose(32, (5,5), activation = LeakyReLU(0.02), strides = 2, padding = 'same'))
decoder.add(BatchNormalization())

decoder.add(Conv2DTranspose(1, (5,5), activation = 'sigmoid', strides = 1, padding = 'same')) # This last matrices represent the RGB channels
decoder.add(BatchNormalization())

decoder.summary()
```

Figure 5.9: Code snippet of the decoder.

5.3 Training process

Once the VAE is built the training data set is introduced to train the parameters of the encoder and decoder. This training is carried out by encoding and decoding the images and

reducing the loss function, which is the difference between the original and reconstructed images. The loss function is defined in Fig. 5.10 written in *Python* code.

```python
# Define the loss function
# VAE loss = recosntruction loss + KL div

# For defining the reconstruction function we use the mean squared error
def reconstruction_loss(y, y_pred):
  return tf.reduce_mean(tf.square(y - y_pred))

# KL divergence
def kl_loss(mu, log_var):
  loss = -0.5 * tf.reduce_mean(1 + log_var - tf.square(mu) - tf.exp(log_var))
  return loss

# Combine coder and decoder to make VAE
mu, log_var, z = encoder(encoder_input) # Get output params from the encoder
reconstructed = decoder(z) # Join latent vector to the decoder
model = Model(encoder_input, reconstructed, name = 'vae') # Combine encoder and decoder outputs
loss = kl_loss(mu, log_var) # Add KL divergence to work as a regularisation
model.add_loss(loss)
model.summary()
```

Figure 5.10: Code snippet of the loss function of the VAE.

The loss function is introduced in the VAE model and gradient descent optimization is carried out. Many optimization algorithms can be used for training the VAE but the *Adam* optimizer was the choice for this project as it is a robust, simple and efficient scheme that is well suited for problems with a large quantity of parameters (Kingma and Ba (2014)). The code snippet containing the training procedure is shown in Fig. 5.11.

```python
# Train VAE
from keras.optimizers import Adam

random_vector = tf.random.normal(shape = (1, latent_dim,)) # Change to have only one random image instead of 25
save_images(decoder, 0, 0, random_vector)

mse_losses = [] # MSE storage
kl_losses = [] # KL loss storage

optimizer = Adam(0.0001, 0.5) # Gradient descent optimizer
epochs = 1500 # Select number of epochs

for epoch in range(1, epochs + 1):
  print('Epoch: ', epoch)
  for step, training_batch in enumerate(training_dataset): # For every epoch get a training batch
    with tf.GradientTape() as tape:
      reconstructed = model(training_batch) # Feed the batch to the model and obtain reconstruction
      y_true = tf.reshape(training_batch, shape = [-1]) # Reshape true output to be able to compare with pred
      y_pred= tf.reshape(reconstructed, shape = [-1]) # Reshape pred output to be able to compare with true

      mse_loss = reconstruction_loss(y_true, y_pred) # Get reconstruction loss
      mse_losses.append(mse_loss.numpy()) # Store loss in the list

      kl = sum(model.losses) # Get KL loss from the model
      kl_losses.append(kl.numpy()) # Store loss in the list

      train_loss = 0.01 * kl + mse_loss # Add losses to get the overall loss

      grads = tape.gradient(train_loss, model.trainable_variables) # Measure the gradients
      optimizer.apply_gradients(zip(grads, model.trainable_variables)) # Introduce the gradients into optimizer to update the weights
```

Figure 5.11: Code snippet of the training process of the VAE.

Finally, the training progress of the model is saved every certain amount of epochs, keeping the trained parameters of the encoder and decoder (Fig. 5.12). These parameters can be reloaded afterwards for the encoding and decoding of future images.

There is one last detail not mentioned before and is the choice of the size of the latent vector. In this project are tested and compared the performances of VAEs with latent vectors of 8, 16 and 32 elements. As a reminder, a larger latent vector keeps better the features of the encoded images but at the same time requires a larger amount of memory to save the latent

```
201     if epoch % 10 == 0 and step == 0:
202       encoder.save('./checkpoints_encoder/Epoch ' + str(epoch) + ' - ' + 'Step ' + str(step))
203       model.save('./checkpoints_model/Epoch ' + str(epoch) + ' - ' + 'Step ' + str(step))
204       decoder.save('./checkpoints_decoder/Epoch ' + str(epoch) + ' - ' + 'Step ' + str(step))
```

Figure 5.12: Code snippet of the trained parameters saving of the VAE.

vectors (something that may be a problem for the learning of the later prediction model). The results obtained for both sizes are shown in Section 5.4, together with a comparison and a conclusion of which one should be used for training the regression model.

Several trainings of the VAE have been carried out in the cluster of *ULiege, NIC5*, taking different times for unknown reasons despite requesting the same computational resources. Nevertheless, as a conclusion the mean training speed is around 1 epoch every half an hour, for the three cases of 8, 16 and 32 element sized latent vectors. When observing the results, first decent image reconstructions are obtained after the 50 epochs but good performance is achieved after the 150 epochs. Taking this into account, the VAE takes around **3 days of training** for reaching sufficiently good performance in image reconstruction. Further research can be carried out to find the optimal neural network structure and hyperparameters of the VAE for a better performance and shorter training times.

5.4 Results and conclusions

After training the VAE the parameters of the model are saved so that they can be loaded again for the later encoding and decoding of images. The encoder and decoder are saved individually and two different *Python* scripts are built to load them, shown correspondingly in Fig. 5.13 and Fig. 5.14.

```
60  # Process
61  image_size = 128
62
63  def preprocess(image):
64      image = cv2.imread(image, cv2.IMREAD_GRAYSCALE) # Read the image as a file
65      image = np.array(image, np.float32) # Cast the image into float values
66      image = cv2.resize(image, (image_size,image_size)) # Resize the images
67      image = image / 255.0 # Normalization
68      return image
69
70  # Transform set of images into tensorflow array set
71  prediction_dataset = []
72  for idx, image in enumerate(names):
73      image = preprocess(image)
74      prediction_dataset.append(tf.convert_to_tensor(tf.constant([image], dtype=tf.float32)))
75
76  # Load encoder
77  encoder = tf.keras.saving.load_model(path)
78
79  # Predict latent vectors
80  latent_vector = encoder(prediction_dataset)[0][0]
81  print(latent_vector)
```

Figure 5.13: Code snippet of the image encoding. The image needs to be shortly processed before being provided to the encoder.

To show the performance of the VAE for different microstructure morphologies four images representing various stages of the simulation are taken from the phase-field results and passed through encoding and decoding in the three VAEs trained (of 8, 16 and 32 element sized latent vector). As the performance of the VAE improves with larger training periods, Fig. 5.15, Fig. 5.16 and Fig. 5.17 show the reconstruction of the images for trainings of 50 (1 day),

```python
# Load model
decoder = tf.keras.saving.load_model(path)

# Predict results
def predict_images(decoder, step, input_):
    prediction = decoder.predict(input_)
    image = prediction * 255.0
    image = image[0,:,:,0]
    image = image.astype(np.uint8)
    image = cv2.resize(image, (735,735))
    cv2.imwrite('./saved/image_'+str(step)+'.png', image)

vector = predict_images(decoder, step, latent_vector)
```

Figure 5.14: Code snippet of the image decoding. The reconstructed image is saved after the prediction.

100 (2 days), and 150 (3 days) epochs of learning respectively. No significant improvement is appreciated for further training. The figures show the original images in the first column and the reconstruction for 8, 16 and 32 element sized latent vectors in the three next columns. The images are taken from a phase-field simulation of the first initial microstructure (Section 3.3) with $\dot{T} = 50$.

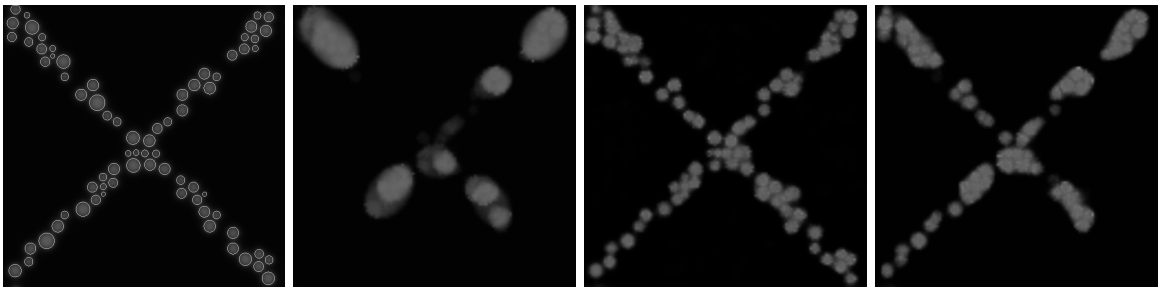
By looking at the comparison between images it is possible to conclude several things. Firstly, the more epochs the VAE is trained the better performance is acquired, always arriving to a limit where the algorithm stops obtaining significant improvement. Secondly, larger sizes of the latent vector are able to retain more characteristics of the image, improving the reconstruction. The main drawback of increasing the size of the latent vector is a larger computation afterwards in the image prediction. Finally, it is also important to say that reconstructed images are more blurry because the VAE works with images of 128x128 pixels, instead of the original 735x735 pixels, losing some quality and details.

The whole data base is passed by the VAE and encoded to reduce its size and speed up the prediction algorithm training process. The size of the data base is significantly reduced when encoding the images. Table 5.1 shows the size of the data base in original resolution (735x735 pixels), reduced resolution (128x128 pixels) and encoded in latent vectors to have an idea of this reduction.

	Original res. (735x735)	**Reduced res. (128x128)**	**8-element vector**	**16-element vector**	**32-element vector**
Size [MB]	834.6	25.3	0.099	0.198	0.396

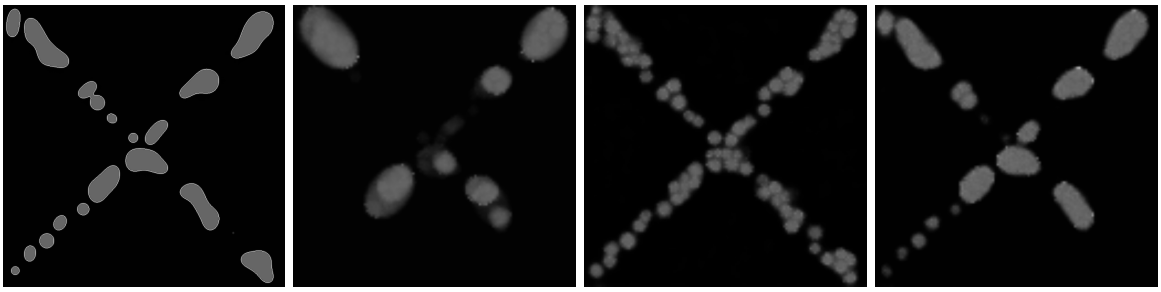
Table 5.1: Size of the training data set before and after the VAE.

As it is evident, the VAE greatly reduces the size of the training data set, at the cost of a small loss in image quality. By evaluating the quality of the image reconstruction and the training times of the different prediction algorithms, shown in Table 6.1 in the next chapter, the **32-element sized latent vector is chosen** as the optimal size to obtain the best results. From this point the results shown are all obtained with the VAE of 32-element sized latent vector. The increase of the training times in the prediction algorithms with the size of the latent vector is not very large, enabling us to keep a higher reconstruction quality by paying small quantity of computational resources. However, the data set of this project is relatively small and for cases with higher volumes of training data it may be convenient to reduce the size of the latent vector, as the cost of the training processes in prediction algorithms rapidly scales with the volume of the training data set.



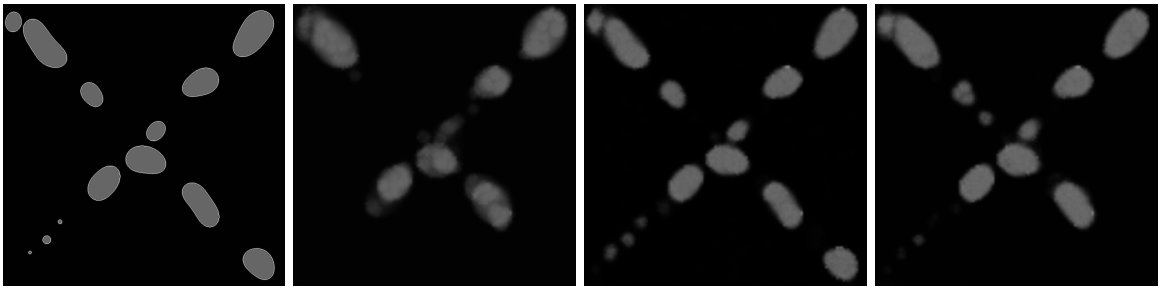
Original. 8 elements. 16 elements. 32 elements.

Initial state. Time = 0 s.



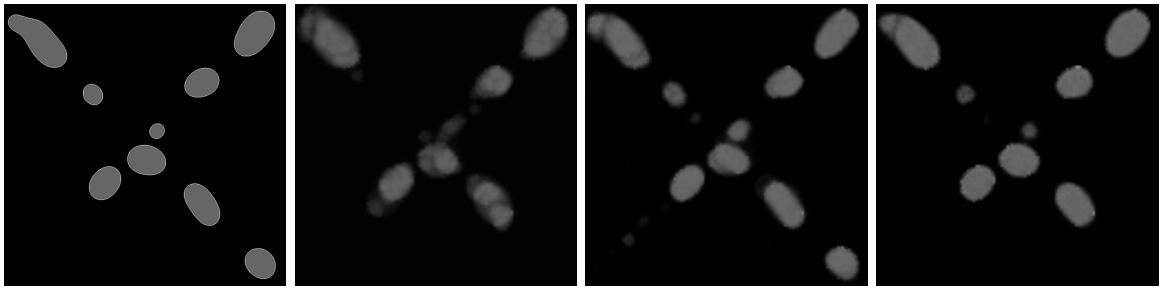
Original. 8 elements. 16 elements. 32 elements.

Time = 512 s.



Original. 8 elements. 16 elements. 32 elements.

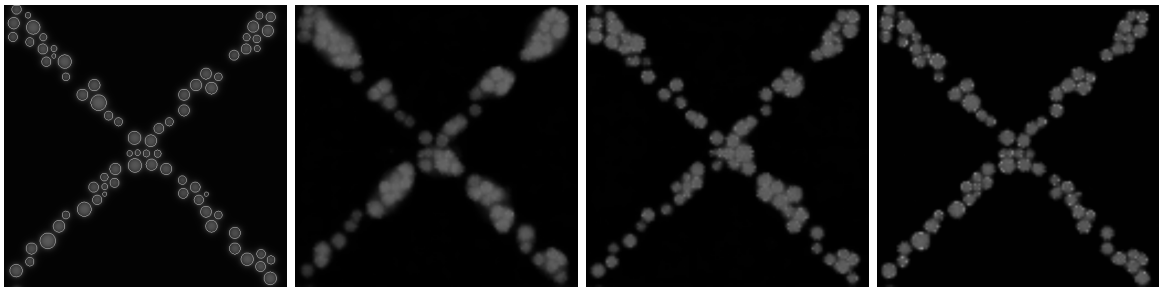
Time = 1024 s.



Original. 8 elements. 16 elements. 32 elements.

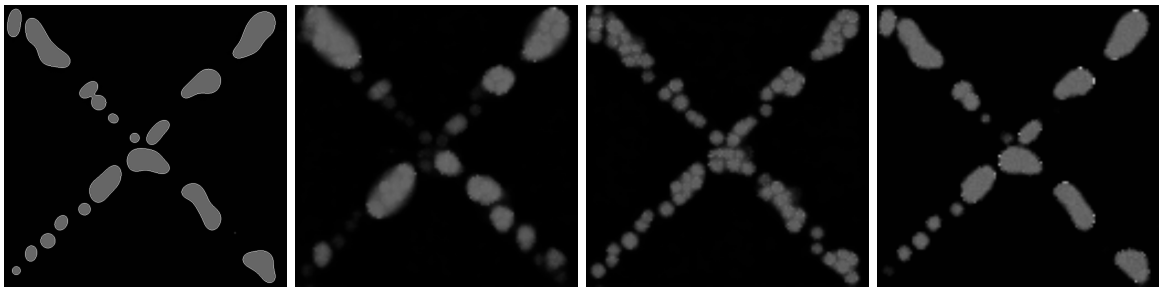
Time = 1536 s.

Figure 5.15: Reconstruction of micrographies with VAEs of different sized latent vectors. Training duration of 50 epochs (1 day).



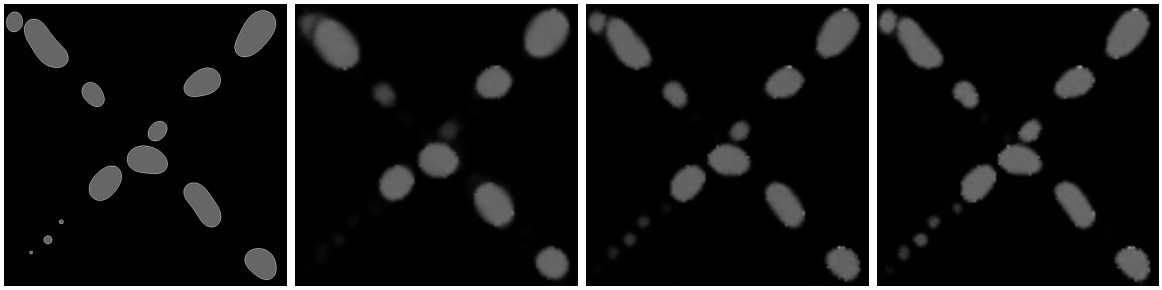
Original. 8 elements. 16 elements. 32 elements.

Initial state. Time = 0 s.



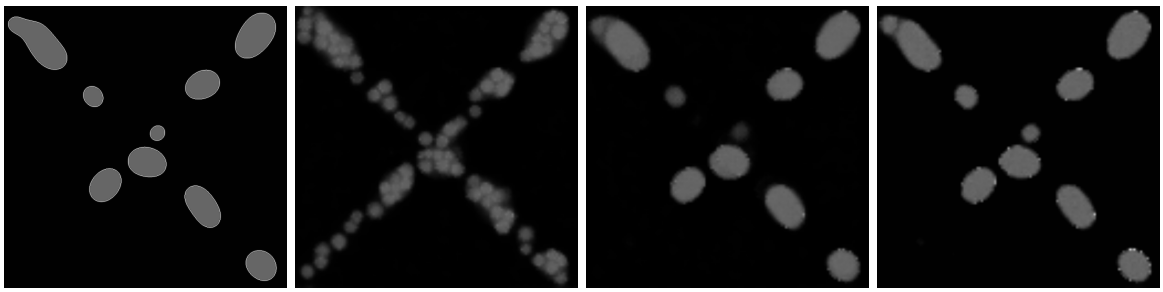
Original. 8 elements. 16 elements. 32 elements.

Time = 512 s.



Original. 8 elements. 16 elements. 32 elements.

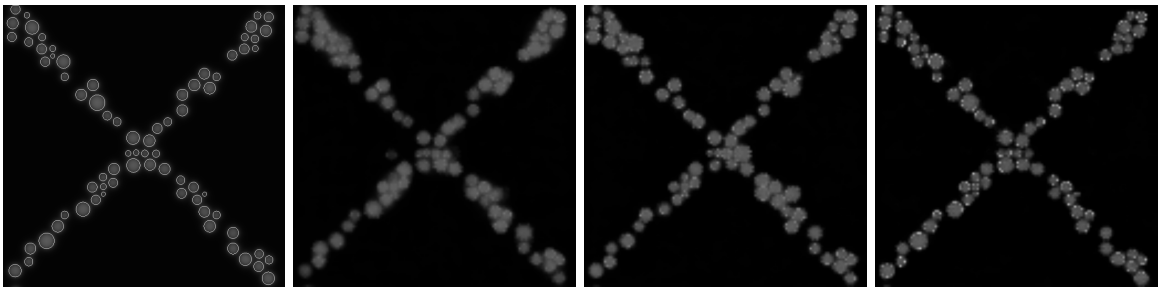
Time = 1024 s.



Original. 8 elements. 16 elements. 32 elements.

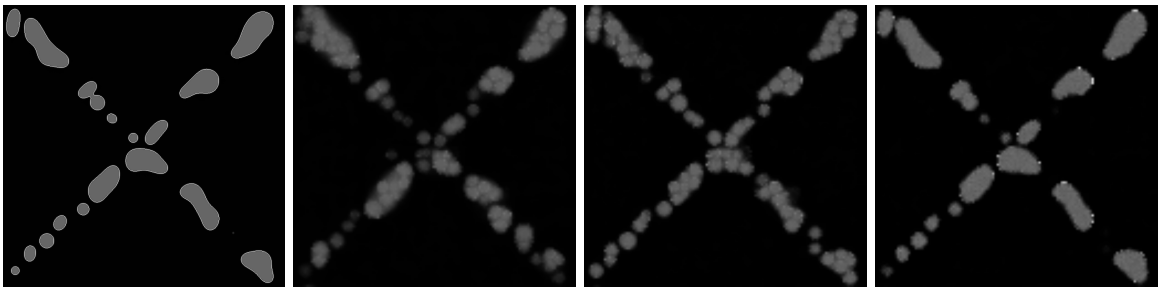
Time = 1536 s.

Figure 5.16: Reconstruction of micrographies with VAEs of different sized latent vectors. Training duration of 100 epochs (2 days).



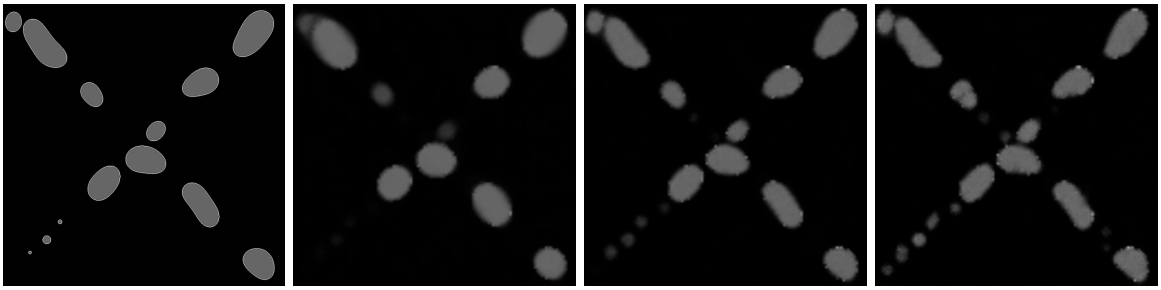
Original. 8 elements. 16 elements. 32 elements.

Initial state. Time = 0 s.



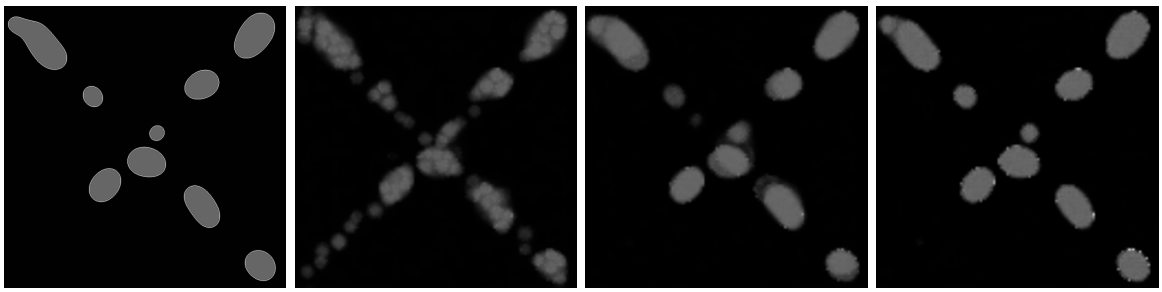
Original. 8 elements. 16 elements. 32 elements.

Time = 512 s.



Original. 8 elements. 16 elements. 32 elements.

Time = 1024 s.



Original. 8 elements. 16 elements. 32 elements.

Time = 1536 s.

Figure 5.17: Reconstruction of micrographies with VAEs of different sized latent vectors. Training duration of 150 epochs (3 days).

Chapter 6

Prediction algorithm

The final purpose of this project is to, given a value of the yield stress of an AlSi10Mg alloy, be able to obtain a corresponding image of the microstructure. These predictions in cases similar to ours are carried out normally by regression algorithms that, when providing a series of training inputs and outputs, are capable of learning the relation between them and reproduce the same results with completely new inputs.

Once reached this stage of the project it is clear that the input provided to the prediction model is the yield stress, while the output received is a latent vector that is later decoded in the VAE to finally obtain the desired micrography.

To have a clear idea of how regression models work and what is their expected performance a brief introduction is given in Section 6.1. Then, getting closer to our particular problem, an evaluation of the performance of several Single-Input-Multiple-Output (SIMO) algorithms is carried out in Section 6.2 in order to find the ideal method for the image prediction. Finally, Section 6.3 goes deeper in the chosen algorithm, explaining the reasons for using it and its performance within the context of this project.

6.1 Introduction to regression models

Within the field of machine learning (Alpaydin (2020)) and data prediction there are an infinite variety of problems, each of them with its unique features. The possible approaches are normally divided into three broad categories, which correspond to learning paradigms, depending on the nature of the signal or feedback available to the learning system. Fig. 6.1 shows a scheme representing the different classes of machine learning, together with some examples of them.

- **Supervised learning**: the input and output objects are labeled. The training process consists of making the algorithm to build a function that maps the new data on the expected output values.
- **Unsupervised learning**: the data provided is not labeled, leaving the algorithm the task of finding its structure. The algorithm learns by reaching for patterns that link the provided data.
- **Reinforcement learning**: the algorithm (or so called agent in this case) interacts with an environment in which it must achieve a certain goal. The agent receives rewards when getting closer to the objective and in the process of learning changes its own features of behaviour (decision making of the agent) to maximize that reward.

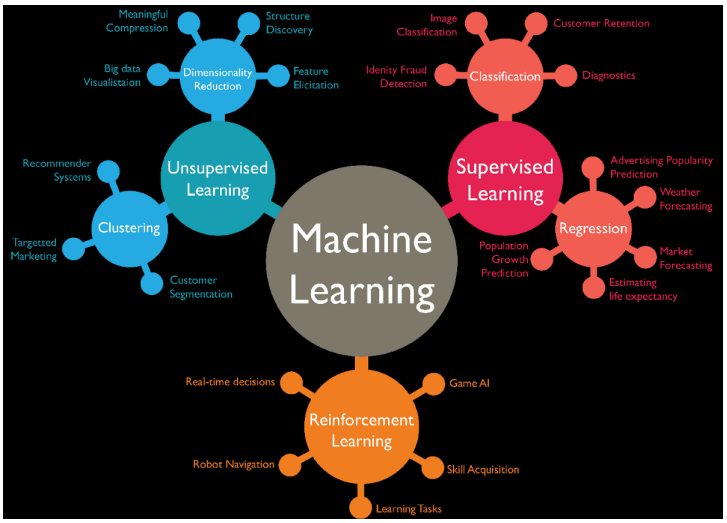


Figure 6.1: Machine learning map indicating the classification of possible problems.

Defining our training data as a set of yield stress values that have each of them a latent vector linked, and the goal of finding a general function that relates both sides, it is clear that this project faces a problem of supervised learning.

The supervised learning problems can be at the same time divided into regression problems, where the algorithm must find the relationship between the input (independent variable) and the output (dependent variable), and classification problems, where the input must be classified into one of the possible categories. The regression problem is the one concerning to this project, as the relation between the yield stress and the latent vector needs to be found.

Normally, regression models consist of defining a function that relates both sides, the dependent (y_i) and independent (x_i) variable, through an unknown parameter (β) and an error term (e_i). This relation is expressed in its most simple way in Eq. (6.1).

$$y_i = \beta x_i + e_i \tag{6.1}$$

This equation may vary depending on the type of problem, the algorithm and the choice of the user, but the principle behind it is the same. The prediction consists of, given an input x_i being capable of obtaining an output y_i that follows as much as possible the tendencies of the training data set. During the training a loss function, ϕ, is computed to evaluate how far is the predicted data, $f(x_i, \beta)$, from the real solution y_i. An example of this loss function can be the squared error, defined in Eq. (6.2).

$$\phi = \sum_i (y_i - f(x_i, \beta))^2 \tag{6.2}$$

To find the optimal prediction function the parameters β and e_i are modified in order to obtain the minimal value of ϕ. This is done using an optimization algorithm such as gradient descent or any of its derived methods.

The shape of the variables in Eq. (6.1) depends on the shapes of the input and output of the algorithm, as they can be a single element or multiple elements (vector, matrix). In the present case the input is 1-element sized, as it only contains the value of the yield stress, while the output is a multi-element array containing the corresponding latent vector. Therefore, the problem to be solved is Single-Input-Multiple-Output (SIMO). These type of problems can be handled with a wide variety of algorithms, among which is very popular the use of Artificial

Neural Networks (ANN), as they are able to find with good accuracy the relation between inputs and outputs. However, many other regression algorithms, such as random forest, KNN or SVR, have a good performance in SIMO problems. To find the most suitable algorithm for the prediction task an evaluation and comparison of the different models is carried out in Section 6.2.

6.2 Single-Input-Multiple-Output models comparison

This section considers some of the most popular algorithms used for SIMO regression problems. For testing them the *Python* libraries *Scikit Learn* and *Tensorflow*, which have the algorithms already implemented, are used. The image results shown in this sections are the predictions of the regression algorithms for 32-element sized latent vectors and VAE decoder with a training of 150 epochs (performance of the VAE shown in Fig. 5.17).

6.2.1 Random forest

Introduction and features

Random forest (Ho (1995)) is a regression algorithm derived from the decision tree learning method (Kotsiantis (2013)). The decision tree algorithm classifies or regresses the data using a set of chained true or false answers to certain questions. The tree is divided into three categories of nodes, the root that contains the input values, the internal nodes that make the data go through a series of different decisions, and the leafs that are the final output values. Fig. 6.2 shows a scheme of this structure.

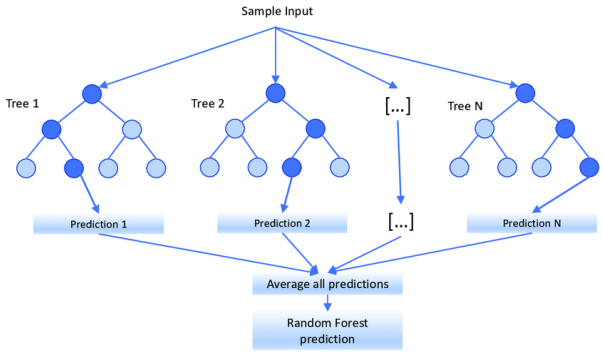


Figure 6.2: Random forest scheme. Several decision trees are computed and the optimal solution is the outputted prediction.

Decision tree algorithms may have accuracy and overfitting problems (too dependent on training data) and that is what random forest algorithms solve. Random forest generates a large number of decision trees with small differences between each other, being capable to find better global optimum features in expense of larger computational resources.

Advantages

- Reduced overfitting when compared to decision tree algorithms.
- High accuracy of the decisions.
- Robustness and stability. Good accuracy when doing predictions that differ from the data set.

Disadvantages

- Complex and difficult to interpret model.
- Computationally expensive.
- Risk of biased decisions.

Implementation and results

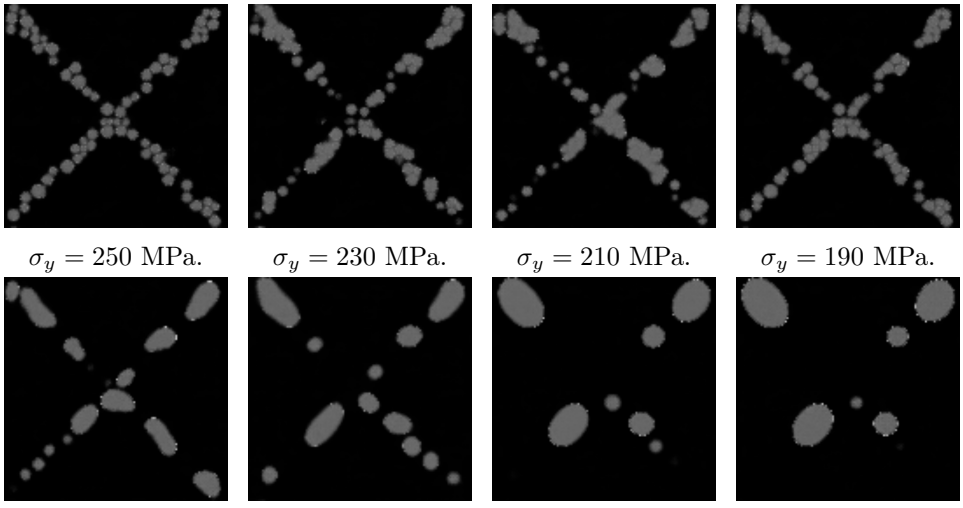
The random forest algorithm is implemented in the *Scikit Learn* library. The function used for building the algorithm is *RandomForestRegressor* and to adapt the model to a SIMO shape the function *MultiOutputRegressor* is included (Fig. 6.3).

```
50  # Train the regressor
51  def Prediction(X_train, y_train):
52
53      # Regression model
54      regressor = RandomForestRegressor(n_estimators=100, random_state=42)
55
56      # Introduce multi-output feature
57      multi_regressor = MultiOutputRegressor(regressor)
58
59      # Fitting in the regression model
60      multi_regressor.fit(X_train, y_train)
61
62      return multi_regressor
```

Figure 6.3: Code snippet of the random forest algorithm implementation.

The training of the algorithm is carried out with the same data base used for the training of the VAE, but instead of images the data provided are the corresponding latent vectors, together with the yield stress value linked to every micrography. For the training data set provided the training time of the algorithm is $\mathbf{t = 19.64\ seconds}$.

To evaluate the performance of the prediction a set of yield stress values have been provided to the algorithm, obtaining the corresponding latent vectors. After decoding the latent vectors, the micrographies obtained are shown in Fig. 6.4.



$\sigma_y = 250$ MPa. $\sigma_y = 230$ MPa. $\sigma_y = 210$ MPa. $\sigma_y = 190$ MPa.

$\sigma_y = 170$ MPa. $\sigma_y = 150$ MPa. $\sigma_y = 130$ MPa. $\sigma_y = 110$ MPa.

Figure 6.4: Predicted micrographies using the random forest algorithm.

6.2.2 Support vector machines (SVM)

Introduction and features

Support Vector Machines (Cortes and Vapnik (1995)), or Support Vector Regression algorithms as it is usually called for regression models, are another method for solving supervised learning problems. In classification problems they work by defining a hyperplane that divides the data samples into different regions, maximizing their distance to the plane. Fig. 6.5 shows an example in a 2-dimensional space. The mean value of the distance w is maximized when defining the plane.

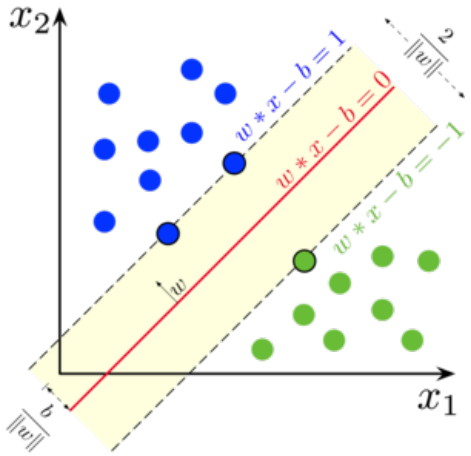


Figure 6.5: SVM scheme.

In regression problems the separation regions are substituted for real numbers.

Advantages

- Effective in high-dimensional spaces.
- Good generalization. The algorithm performs very good with unseen data.
- Robust against overfitting, making the predictions accurate instead of simply trying to reproduce the training data.

Disadvantages

- Sensitive to hyperparameter tunning.
- Bad performance with noisy data.
- Difficult interpretation when used for regression problems.

Implementation and results

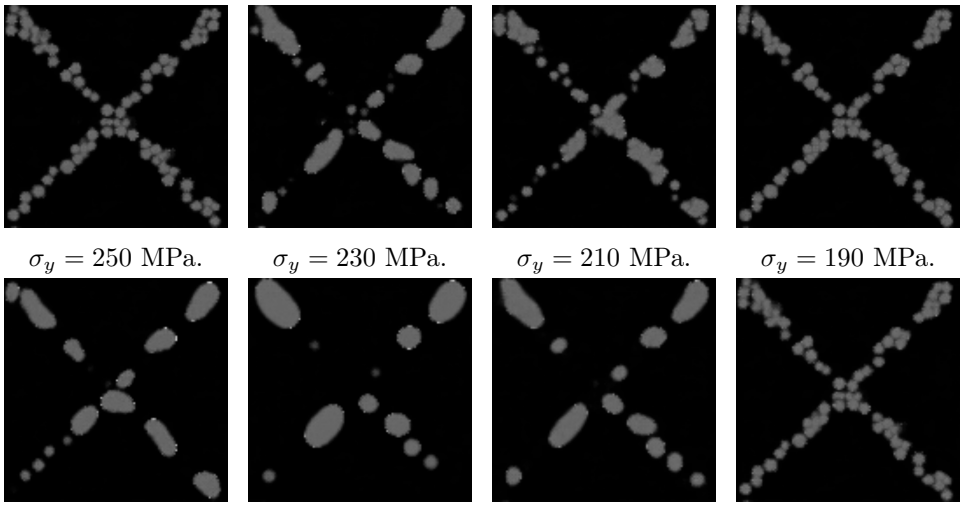
The SVM algorithm is also implemented in the *Scikit Learn* library. It is called by using the *SVR* function and for the current problem is used together with the *MultiOutputRegressor* function. Also a kernel needs to be introduced. There are plenty of kernel functions for choosing, but the final choice has been a combination of a constant value together with an exponential function (very common for all kinds of prediction problems), shown in Fig. 6.6.

```python
# Train the regressor
def Prediction(X_train, y_train):
    # Kernel definition
    sigma_f = 1.0
    l = 1.0
    kernel = ConstantKernel(constant_value=sigma_f, constant_value_bounds=(1e-2, 1e2)) \
    * RBF(length_scale=l, length_scale_bounds=(1e-2, 1e2))

    # Regression model
    regressor = SVR(kernel=kernel)

    # Introduce multi-output feature
    multi_regressor = MultiOutputRegressor(regressor)

    # Fitting in the regression model
    multi_regressor.fit(X_train, y_train)

    return multi_regressor
```

Figure 6.6: Code snippet of the SVM algorithm implementation.

Further research can be done to find a better fitting kernel function in order to obtain a better performance of the algorithm.

The training is also carried out with the same data set as the previous case, taking only a time of $\mathbf{t = 1.78\ seconds}$. The results for a set of different yield stress values are shown in Fig. 6.7.



$\sigma_y = 250$ MPa. $\sigma_y = 230$ MPa. $\sigma_y = 210$ MPa. $\sigma_y = 190$ MPa.

$\sigma_y = 170$ MPa. $\sigma_y = 150$ MPa. $\sigma_y = 130$ MPa. $\sigma_y = 110$ MPa.

Figure 6.7: Predicted micrographies using the SVM algorithm.

6.2.3 K-nearest neighbors (KNN)

Introduction and features

The K-Nearest Neighbors algorithm (Fix and Hodges (1989)) works under a very simple principle that is better explained for the classification problems. Starting from a 2-dimensional map like the one shown in Fig. 6.8, the labeled data is introduced, naturally dividing itself in different clusters depending on their features. A new element is introduced and a number K of the nearest elements is selected, classifying the unknown element in the group of the majority. The number K is a hyperparameter chosen before carrying out the prediction.

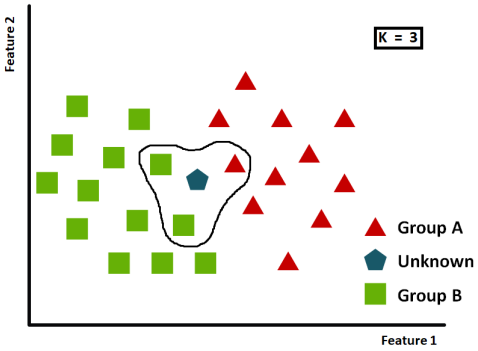


Figure 6.8: K-Nearest Neighbors scheme.

When dealing with regression problems the mean value of the K neighbors is computed and taken as the predicted value (Eq. (6.3)).

$$Y = \frac{1}{K} \sum_{i} y_i \tag{6.3}$$

Advantages

- Very simple and intuitive.
- Robust against noisy data.
- Very low use of computational resources.

Disadvantages

- Bad scalability, making it computationally expensive for large data sets.
- Very sensitive to hyperparameter tunning (K number).
- Sensitive to irrelevant features, making possible the predictions to be guided by unnecessary characteristics.

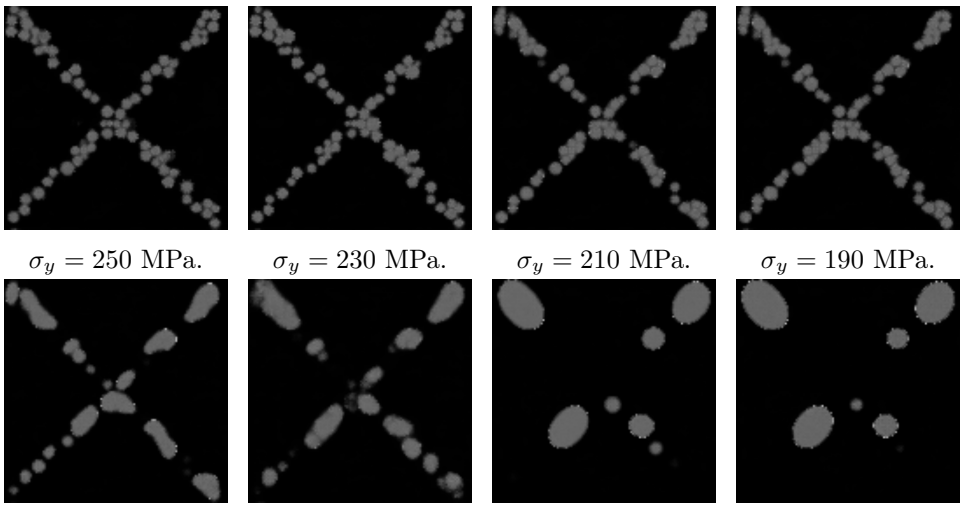
Implementation and results

KNN is also implemented in the *Scikit Learn* library. The model is simply built by using the *KNeighborsRegressor* function together with the *GaussianProcessRegressor* function (Fig. 6.9).

```
50  # Train the regressor
51  def Prediction(X_train, y_train):
52
53      # Regression model
54      regressor = KNeighborsRegressor(n_neighbors=9)
55
56      # Introduce multi-output feature
57      multi_regressor = MultiOutputRegressor(regressor)
58
59      # Fitting in the regression model
60      multi_regressor.fit(X_train, y_train)
61
62      return multi_regressor
```

Figure 6.9: Code snippet of the KNN algorithm implementation.

The training time for this algorithm is extremely short, achieving good results in a time of **t = 0.07 seconds**. The output for several values of σ_y is shown in Fig. 6.10. As this algorithm is very dependent on hyperparameters, several values of K have been tested, finding a very slight (negligible) change of computational time and results with the number of considered neighbors.



$\sigma_y = 250$ MPa. $\sigma_y = 230$ MPa. $\sigma_y = 210$ MPa. $\sigma_y = 190$ MPa.

$\sigma_y = 170$ MPa. $\sigma_y = 150$ MPa. $\sigma_y = 130$ MPa. $\sigma_y = 110$ MPa.

Figure 6.10: Predicted micrographies using the KNN algorithm (K = 9).

6.2.4 Gaussian process regression (GPR)

Introduction and features

The Gaussian Process Regression (Williams and Rasmussen (1995)) model is an algorithm that does not follow the same principles as the previous ones. Nevertheless, it is also taken into account because of the initial suggestion of the co-supervisor of this project, Seifallah Fetni, as well as for trying to reproduce the successful results in Kim et al. (2021).

This algorithm is a non-parametric bayesian approach that provides not only the prediction but also an uncertainty measurement on it. As a simile for its proper explanation, a function with the shape $y = wx + \epsilon$ is considered. The bayesian approach works by assigning a distribution, p(w), on the parameter, w (Eq. (6.4)).

$$p(w|y, X) = \frac{p(y|X, w)\, p(w)}{p(y|X)} \tag{6.4}$$

A gaussian distribution is assumed for the probability function that, when predicting new data points, provides the mean and variance values, making possible to compute an uncertainty range. As the GPR is a non-parametric algorithm, the probability distribution of the parameters shown in this example (Eq. (6.4)) is substituted by the probability of the functions that fit the training data.

Advantages

- Not only provides the prediction but also its uncertainty distribution.
- Being a non-parametric method it doesn't assume a specific functional form for the relationship between features and target, reaching a larger diversity of possible solutions.

- Enables the modelling of a wide range of functions, from simple linear to strong non-linear ones.

Disadvantages

- Low efficiency and high computational cost for large data sets.
- Very sensible to hyperparameter selection. This includes also the choice of the kernel function.
- Not developed for multiple output problems.

Implementation and results

Luckily, the GPR algorithm is already implemented in a wide variety of *Python* libraries, including *Scikit Learn*. The main drawback of this algorithm is that, when used for multiple output problems, it gives a covariance matrix that is hard to interpret and it can not be used together with the *MultiOutputRegressor* function. For this reason the elements of the output are obtained individually from the input and not as a complete vector as they should be, reducing the accuracy of the prediction. At the same time, the difficult to interpret covariance matrix makes impossible to measure the uncertainty of the predictions, turning useless the strongest point of this algorithm. Nevertheless, the implementation is shown in Fig. 6.11.

```
51  # Train the regressor
52  def Prediction(l, sigma_f, X_train, y_train):
53      # Kernel definition
54      kernel = ConstantKernel(constant_value=sigma_f, constant_value_bounds=(1e-2, 1e2)) \
55                  * RBF(length_scale=l, length_scale_bounds=(1e-2, 1e2))
56
57      # Regression model
58      regressor = GaussianProcessRegressor(kernel=kernel, alpha=sigma_n**2, n_restarts_optimizer=10, )
59
60      # Fitting in the regression model
61      regressor.fit(X_train, y_train)
62
63      return regressor
```

Figure 6.11: Code snippet of the GPR algorithm implementation.

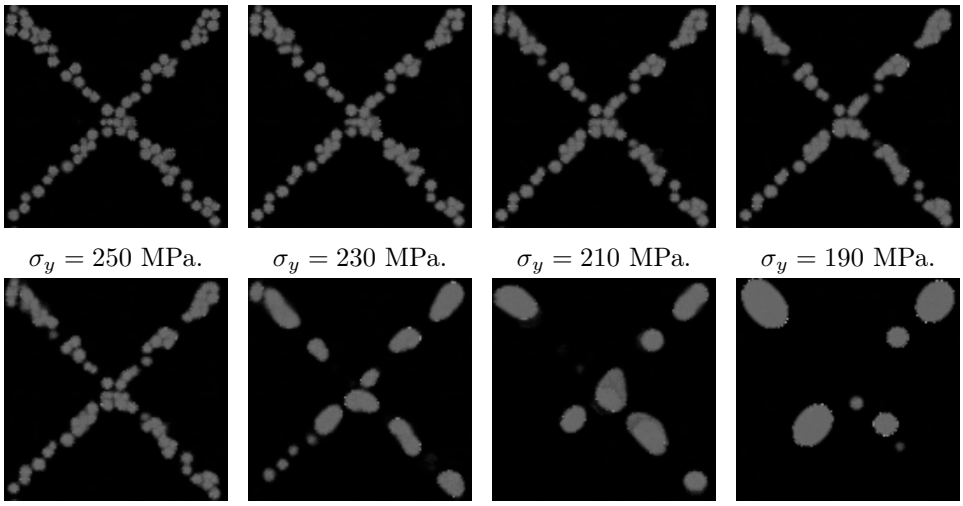
For the same training data set than for the previous algorithms the training time of the GPR is $\mathbf{t = 248.27}$ **seconds**, a very high value when compared to other methods. The results for several yield stress values predictions are shown in Fig. 6.12.

6.2.5 Artificial neural networks (ANN)

Introduction and features

Artificial Neural Networks (Goodfellow et al. (2016)) are Deep Learning algorithms that try to emulate the behaviour and working of neural connections. This structure is also used in Chapter 5 for the construction of the VAE but because this case works with smaller sized data the prediction problem requires a much smaller quantity of resources and shorter training times.

The working principle is explained already in Section 5.1 but it is basically a set of interconnected neurons organized in layers that receive an input value or values and carry out



$\sigma_y = 250$ MPa. $\sigma_y = 230$ MPa. $\sigma_y = 210$ MPa. $\sigma_y = 190$ MPa.

$\sigma_y = 170$ MPa. $\sigma_y = 150$ MPa. $\sigma_y = 130$ MPa. $\sigma_y = 110$ MPa.

Figure 6.12: Predicted micrographies using the GPR algorithm.

mathematical operations with them, sending to the next neuron or neurons the outputted result.

Neural networks are a very polyvalent tool that can be used for a wide variety of problem types, being one of them the regression problems.

Advantages

- Very flexible method that can handle strong non-linear problems.
- Can handle large data sets when enough computational resources are provided.
- Easily re-trainable. Models trained under large data sets can continue their training with smaller or more specific data sets to improve accuracy in different fields or features. This characteristic may be of great interest for this project, as new micrographies can be added to the data set to be able to predict different structures.

Disadvantages

- Does not perform good with small data sets, needing large training data volumes to provide good results.
- Very sensitive to the structure of the network itself, providing slightly different results when the number of layers or their size is changed.
- Difficult to interpret, as it is not possible to represent what happens inside, working similar to a black box.
- Computationally expensive.

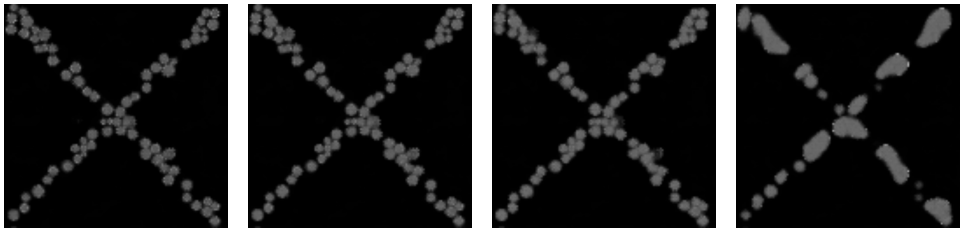
Implementation and results

There are plenty of structure configurations that can solve the prediction problem. As the problem is simple it is not necessary to construct a very large network, thus the layers introduced are relatively small. A configuration of 5 convolutional layers and 2 dense layers is chosen as it is shown in Fig. 6.13. The model is constructed using the *Tensorflow* library as it is done also in Chapter 5.

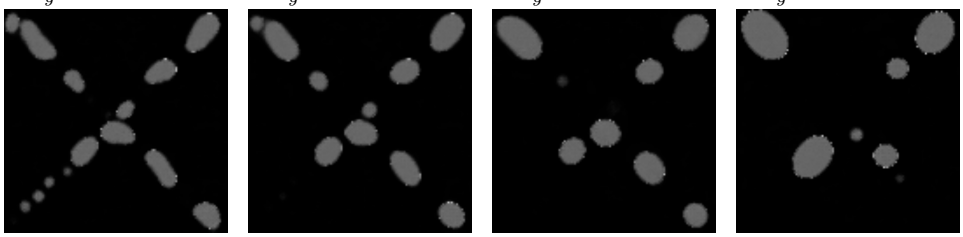
```python
# Train the GPR
def gpPrediction(X_train, y_train):
    # Create a Sequential model
    predict_input = Input(shape = (1, 1)) # Make an input layer

    x = Conv1D(32, (3), activation = LeakyReLU(0.02), strides = 1, padding = 'same')(predict_input)
    x = BatchNormalization()(x) # Normalize the inputs of the layer

    filter_size = [64, 128, 256, 512] # Make convolutional layers with different sizes
    for i in filter_size:
        x = Conv1D(i, (3), activation = LeakyReLU(0.02), strides = 2, padding = 'same')(x)
        x = BatchNormalization()(x)

    x = Flatten()(x) # Convert all feature matrices into vectors
    x = Dense(64, activation = LeakyReLU(0.02))(x) # Attach dense layer
    x = Dense(16, activation = LeakyReLU(0.02))(x) # Attach dense layer
    predict_output = BatchNormalization()(x)

    predict = Model(predict_input, outputs = [predict_output], name = 'encoder')
    predict.summary()

    # Compile the model
    predict.compile(optimizer='adam', loss='mean_squared_error')

    # Fitting in the gp model
    epochs = 50
    predict.fit(X_train, y_train, epochs=epochs, validation_split=0.2, verbose=2)

    return predict
```

Figure 6.13: Code snippet of the ANN building.

The choice of number of epochs for the training is a fundamental parameter, as very low epochs do not achieve good results, while large number of them may be unnecessary and even overfit the algorithm. After several trials a good compromise of accuracy and training time is found for **epochs = 50**. The results for this training are shown in Fig. 6.14, taking a training time of **t = 43.96 seconds**.



$\sigma_y = 250$ MPa. $\sigma_y = 230$ MPa. $\sigma_y = 210$ MPa. $\sigma_y = 190$ MPa.



$\sigma_y = 170$ MPa. $\sigma_y = 150$ MPa. $\sigma_y = 130$ MPa. $\sigma_y = 110$ MPa.

Figure 6.14: Predicted micrographies using an ANN.

6.2.6 Training times comparison

The training times of each regression algorithm tested are provided in Table 6.1 for the three cases of 8, 16 and 32 elements sized latent vectors.

Latent vector size	Random forest	SVM	KNN	GPR	ANN
8 elements	5.67	0.66	0.03	87.13	55.18
16 elements	8.51	0.84	0.04	122.33	51.2
32 elements	19.64	1.78	0.07	248.27	43.96

Table 6.1: Training times in seconds of the prediction algorithms evaluated for different latent vector sizes.

6.3 Results and conclusions

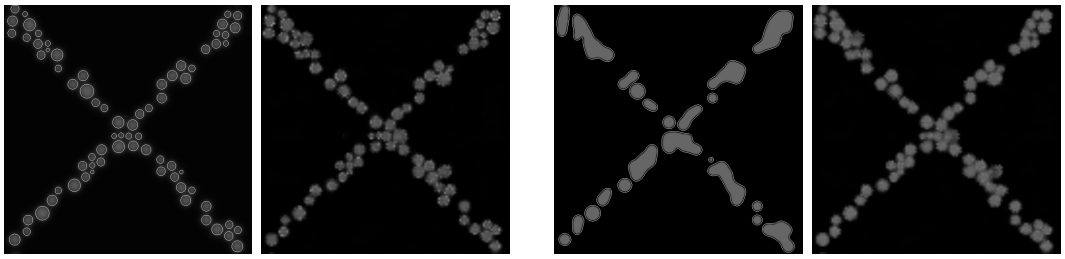
Taking into account the performance of all the prediction algorithms **the best method is apparently to use artificial neural networks**, as they provide a robust prediction and due to their structural flexibility they have a large margin of improvement. Other methods like Random Forest and KNN are also very good choices, as they achieve very good reconstruction results too and require less training time.

To test the effectiveness of the prediction a phase-field simulation is carried out, post-processing the micrographies to obtain the yield stress value and introducing this same σ_y into the prediction algorithm. The predicted images are compared in Fig. 6.15 with the original phase-field micrographies to see if the reconstruction through providing the yield stress value can reproduce similar results than the phase-field software.

The images are ordered from largest to smallest value of the yield stress. However, talking about the original micrographies, the order that they were obtained from the simulation is not the same as they are ordered in the figure, being (a) the first one, followed by (e) and later continuing by (b) all the way down to the last one, (j). This sudden drop of the calculated value of the yield stress in the second micrography is due to bad performance of the post-process in the images of early stages of the simulation (image (e)), providing a much lower value than the one it should give following the tendency and the evolution of the microstructure.

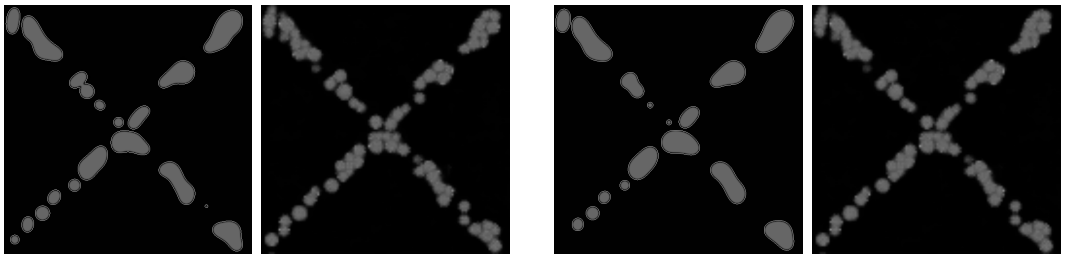
On the other hand, the predicted micrographies follow a steady evolution with the coarsening of the precipitates and the disappearance of the eutectic wall as the yield stress diminishes. This means that the prediction algorithm not only has a great performance in microstructure prediction, but also is able to overcome the errors made by the post-process and provide a better fitting microstructure for the given yield stress value.

The ideal way to validate the performance of the ANN would be to apply the post-processing software of Chapter 4 to the predicted micrographies and compare its yield stress value to the original one. For the images used this is not possible to do, as by reducing the resolution of the images to use them in the VAE the features of the predicted microstructure are not easy to obtain (mean radius of the precipitates, phase differentiation...). For better resolutions of the predicted images this validation is very recommended to do, but it may require large computational resources because of the training of the VAE, that would have to handle training arrays of 500000 elements (735x735 pixels resolution) instead of the current ones of 16000 (128x128 pixels resolution).



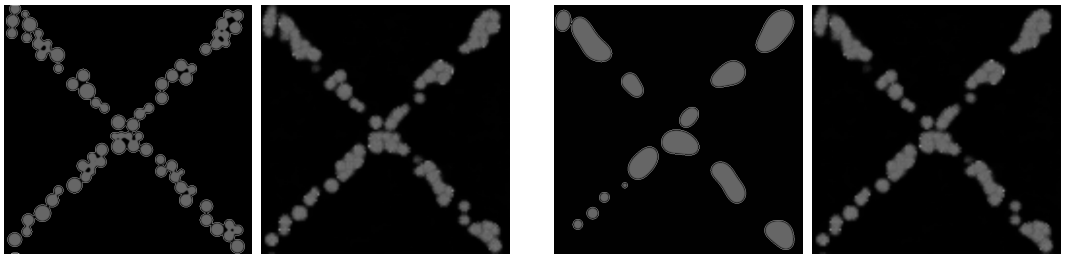
(a) $\sigma_y = 262.28$ MPa.

(b) $\sigma_y = 213.08$ MPa.



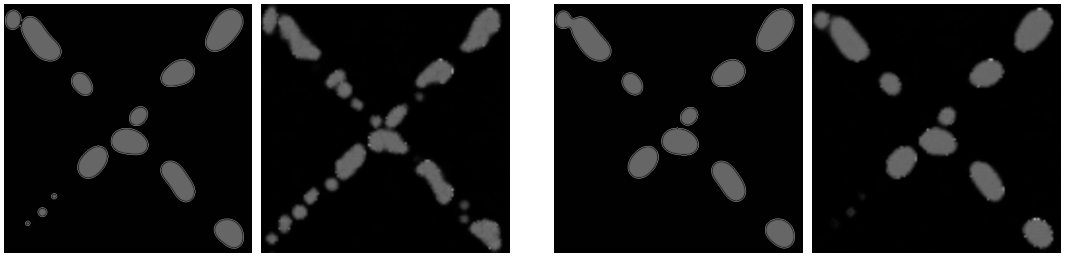
(c) $\sigma_y = 191.86$ MPa.

(d) $\sigma_y = 179.92$ MPa.



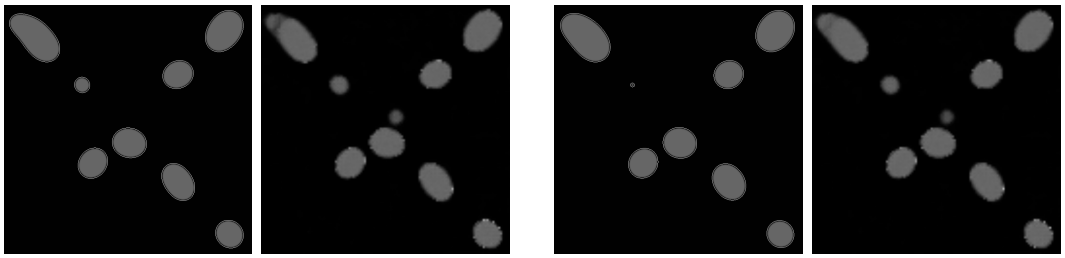
(e) $\sigma_y = 175.64$ MPa.

(f) $\sigma_y = 164.66$ MPa.



(g) $\sigma_y = 157.23$ MPa.

(h) $\sigma_y = 141.36$ MPa.



(i) $\sigma_y = 133.53$ MPa.

(j) $\sigma_y = 132.34$ MPa.

Figure 6.15: Original (left) and predicted (right) micrographies for several values of σ_y placed in pairs.

Chapter 7

Results

The results of each part of the project are already shown in the previous chapters, evaluating the performance of the post-processing software, the VAE and the prediction algorithm. As a way to merge all the results together and give a general overview of the whole project, this chapter is a summary that covers the overall performance and has the purpose to help the reader understand the path followed.

As a reminder, the initial goals were to, by providing a value of the yield stress, be capable to predict a corresponding microstructure of AlSi10Mg. Fig. 1.3 shows the scheme of the four main parts of the project, the phase-field micrographies generation, the yield stress computation, the encoding of the images using the VAE and the prediction algorithm to achieve the expected goals.

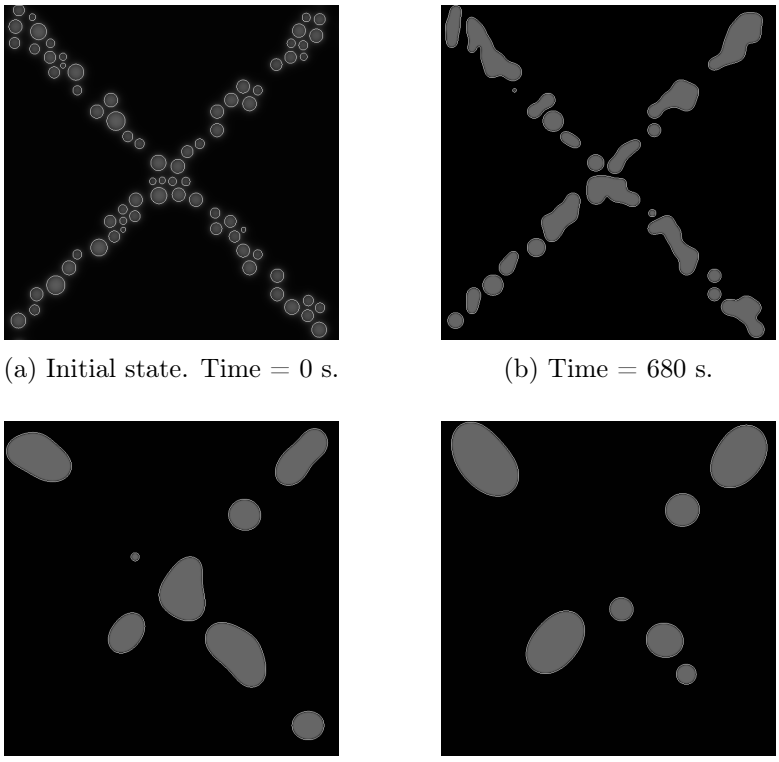
Data base

A series of phase-field simulations are launched to create a data base to train both the VAE and the prediction algorithm. The simulations consist on, starting from three different AlSi10Mg microstructures, apply different annealing thermal treatments and save the evolution of the microstructure. With this procedure 1620 micrographies are obtained, a small but sufficiently large data base for the training. As the objectives for the prediction are modest, staying in concrete shapes of the microstructure and thermal treatments, the data base is not required to be very large. Nevertheless, for future development of this research it is recommended to widen the possibilities of the microstructure morphology. Examples of the images obtained are shown in Fig. 7.1, where different microstructure morphologies can be appreciated.

Post-processing and σ_y prediction

The micrographies are introduced in the post-processing software to obtain their features and calculate the corresponding yield stress values. The features extracted are:

- Weight fraction of silicon, X_{wSi}.
- Volume fraction of cell phase, f_{cell}.
- Volume fraction of eutectic phase, f_{eu}.
- Volume fraction of the precipitates, X_v^d.
- Mean radius of the precipitates, $\bar{r}_d$.
- Mean distance between precipitates, l.



(a) Initial state. Time = 0 s.

(b) Time = 680 s.

(c) Time = 1362 s.

(d) Time = 2044 s.

Figure 7.1: Evolution of the first microstructure with an annealing treatment and constant temperature increase of $\dot{T} = 20$.

With these features and the properties of the Al and Si the value of σ_y is obtained for every micrography. The value of the yield stress is compared with the one experimentally measured in similar microstructures shown in Fig. 7.2, taken from Zhao et al. (2019). From the figure is possible to distinguish a cellular + eutectic microstructure in Fig. 7.2a and Fig. 7.2b, resembling to the early stages in the phase-field simulation, while Fig. 7.2c and Fig. 7.2d present an Al matrix + Si precipitates microstructure more typical of the final stages of the simulation, when the heat treatment and the diffusion had led to the destruction of the eutectic wall.

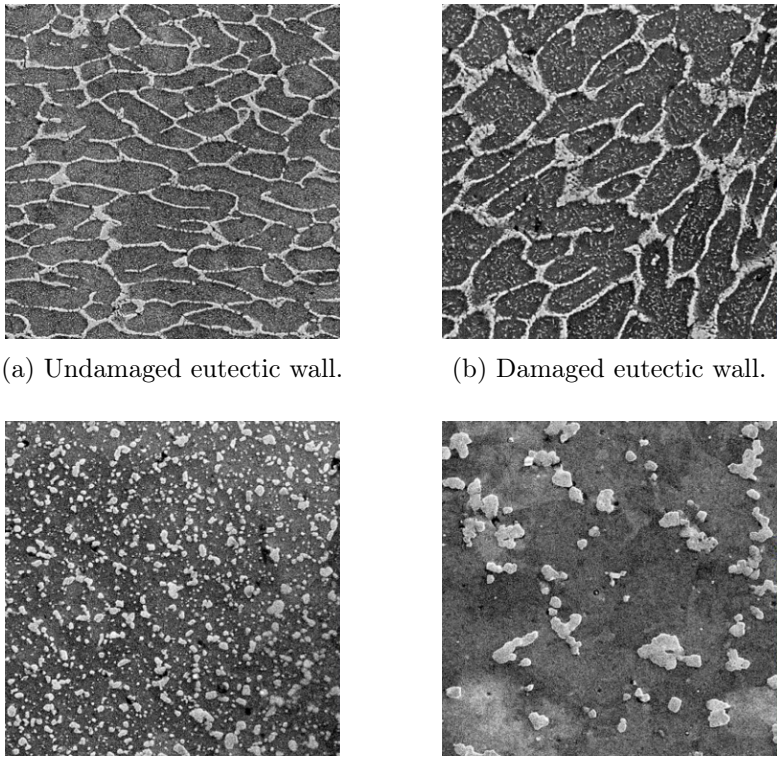
The real microstructures follow a similar evolution as the virtual ones, as with higher temperatures the diffusion increases destroying the eutectic wall and leaving instead an Al matrix with Si precipitates that do not follow an specific distribution. The yield stress values measured in the images of Fig. 7.1 and the experimental values extracted from the microstructures in Fig. 7.2 are compared in Table 7.1 to contrast the order of magnitude of similar microstructures. It is remarked that this comparison is only to verify the order of magnitude of the yield stress calculated, as the real and virtual micrographies do not have further relationship than the similar aspect and phases distribution, but they work as an approximate comparison.

	Image (a)	**Image (b)**	**Image (c)**	**Image (d)**
Real microstructure	287	249	212	189
Virtual microstructure	262.28	214.41	154.83	139.3
Error [%]	8.61	13.89	26.97	26.30

Table 7.1: Values of the yield stress of real microstructures (experimental) and virtual microstructures (calculated) in MPa. The absolute error with respect to the experimental values is also shown.

Looking at the values it is evident that there is an important difference between the experimental and computational results. This is something expected, as images are only related because of having similar microstructures. The virtual micrographies are also in a different scale where only a small part of the eutectic wall is observed, losing information about the whole microstructure. Besides that, the order of magnitude of the yield stress is similar, as well as its decrease tendency with higher temperatures of the thermal treatment too.

Despite the yield stress values are not as precise as they would be required to be for a proper mechanical properties analysis, they are still good enough for the purpose of this project, as they are required just to associate the micrographies to a yield stress value to develop the prediction algorithms, that are the main goal to achieve.



(a) Undamaged eutectic wall. (b) Damaged eutectic wall.

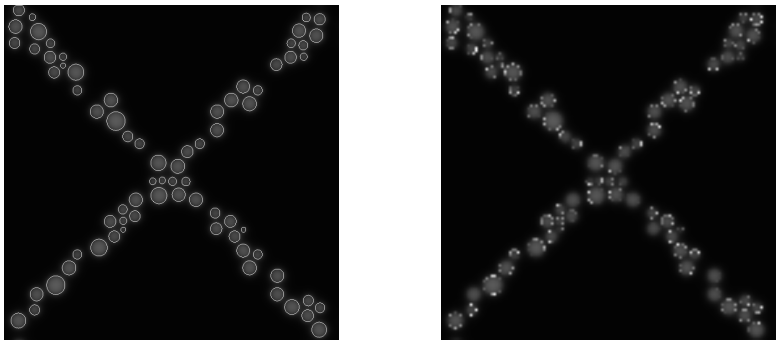
(c) Small precipitates. (d) Large precipitates.

Figure 7.2: Real AlSi10Mg microstructures taken from Zhao et al. (2019) for SLM manufacturing process and later thermal treatments. Image (a) represents the as built microstructure, (b) and (c) were given a 250ºC and 300ºC annealing for two hours, and (d) went under a Friction Stir Process (FSP) of one pass.

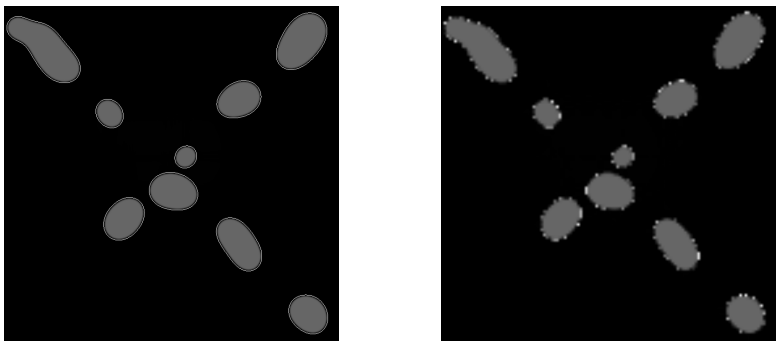
Variational Autoencoder

On the other side of Fig. 1.3, the VAE, explained in Chapter 5, enables the reduction of the training data set size for the prediction algorithm. Paying a small cost in image quality, the VAE is able to encode images of 735x735 pixels into a 32 element sized latent vector, **reducing the size of the data set size in a 99.95%**. Nevertheless, the reconstruction of the images is done in a 128x128 pixels resolution, a lower quality than the original image. To give an idea of how this resolution reduction affects the image quality examples of both resolutions are shown in Fig. 7.3.

By considering the resolution of the reconstructed images instead of the original one,



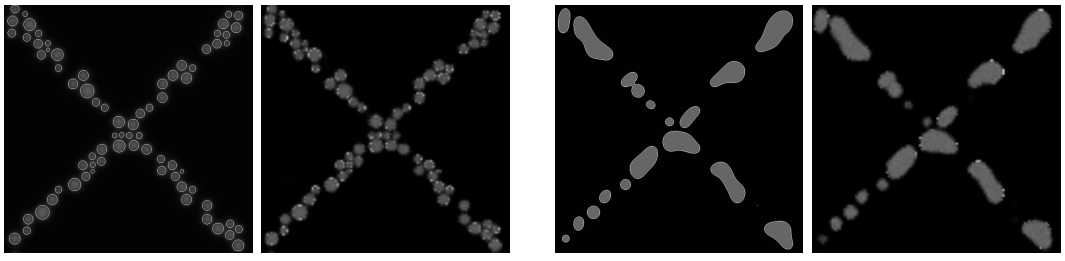
(a) 735x735 pixels. (b) 128x128 pixels.



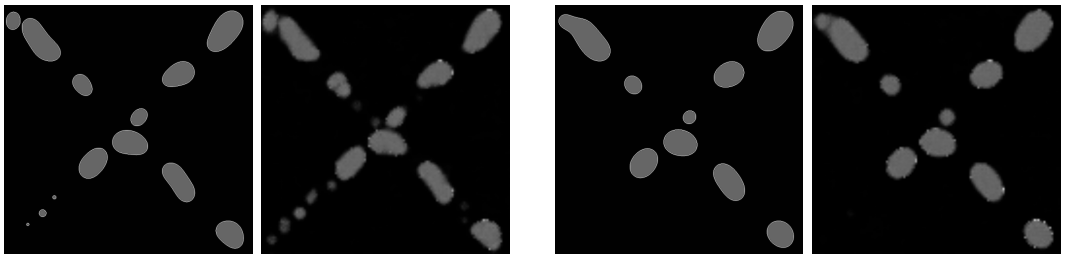
(c) 735x735 pixels. (d) 128x128 pixels.

Figure 7.3: Comparison of virtual micrographies with original, (a) and (c), and reduced resolutions, (b) and (d).

when encoding the training images in the VAE **the size reduction of the training data set is of 98.4%**, a still very high reduction considering the very good results of the image reconstruction.



(a) Initial microstructue. t = 0 s. (b) t = 512 s.



(c) t = 1024 s. (d) t = 1534 s.

Figure 7.4: Comparison of original, (left) and reconstructed (right) micrographies for different phases of the thermal treatment.

The image reconstruction capabilities of the VAE are also shown in Fig. 7.4, where images directly obtained from the phase-field simulations are encoded and decoded back, obtaining almost the same image. Such successful reconstruction proves that the VAE, when properly trained, is a very powerful tool that enables a great reduction of the training data sets sizes for prediction algorithms.

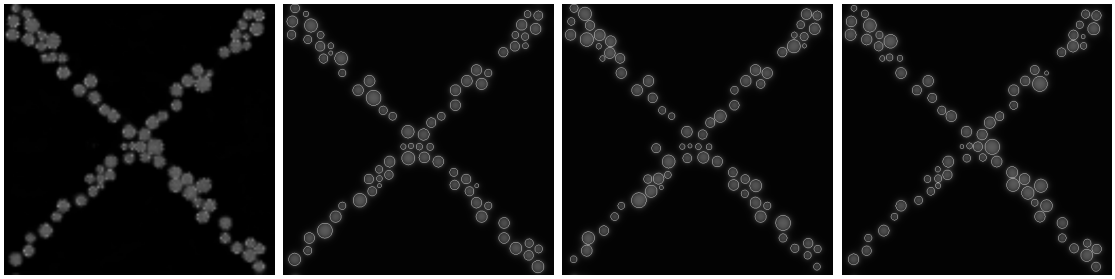
Prediction algorithm

For the prediction of the images by providing a yield stress value the artificial neural network is built and trained, using the same data base than for the VAE but reduced to latent vectors. Each latent vector (output) has a yield stress value associated (input). The time required for training the ANN can be shorter or larger depending on the epochs indicated. The predictions are already very good with a training of 50 epochs, taking 43.9 seconds, a relatively low time due to the small training data set size and the number of trainable parameters in the network.

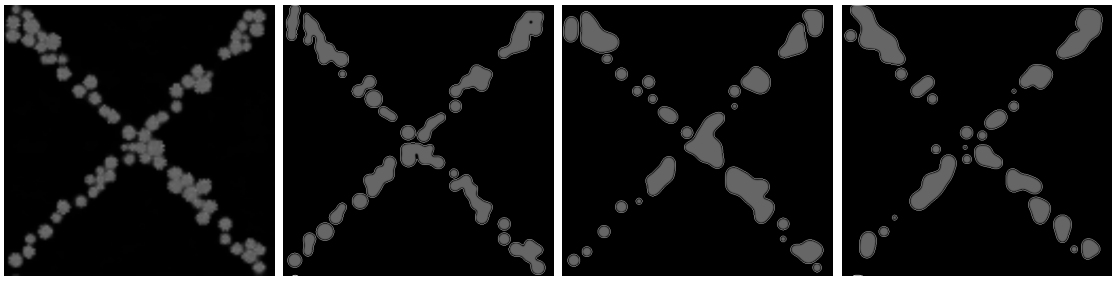
In Fig. 6.15 are compared a series of phase-field obtained images, where a value of the yield stress is calculated through post-processing, together with the migrography predicted when that same yield stress value is provided to the ANN. The results obtained are good, being the predicted images capable of accurately reproducing similar microstructures as the original ones, even fixing errors of the post-processing analysis like the bad yield stress computation for micrographies from the early stages of the phase-field simulation.

Another way to validate the results from the prediction is to provide the algorithm with several values of the yield stress, produce the corresponding micrographies and compare them to other micrographies in the data base that have similar values of the yield stress. This is done in Fig. 7.5, providing the ANN a series of σ_y values uniformly distributed between the 260 and 120 MPa.

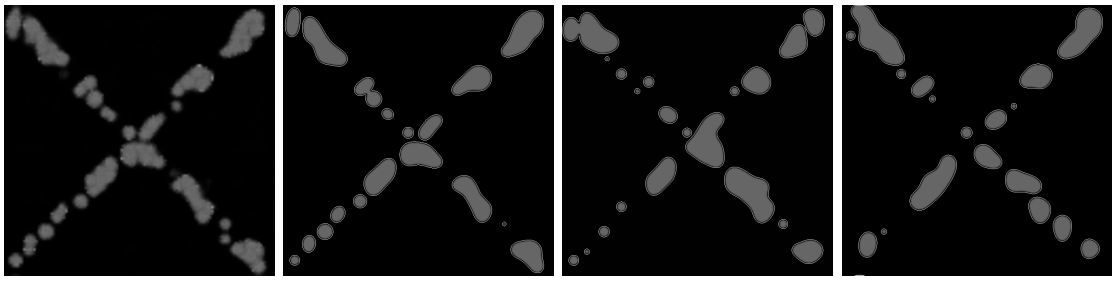
It is observed that the micrographies compared are very similar between each other. The original phase-field images are taken each from one of the three initial microstructures Fig. 3.3 to have a more diverse morphology to compare with. Depending on the predicted micrography it has more or less similarities in the shape with one of the original micrographies or another. This is because the ANN was trained with this same phase-field micrographies. If the ANN was trained with more different micrographies it would take features from many more sources, making predictions that are not that biased by the training data set. This results are very successful for a first version of the prediction algorithm but there is a lot of improvement margin if the structure is refined and the training data set is increased.



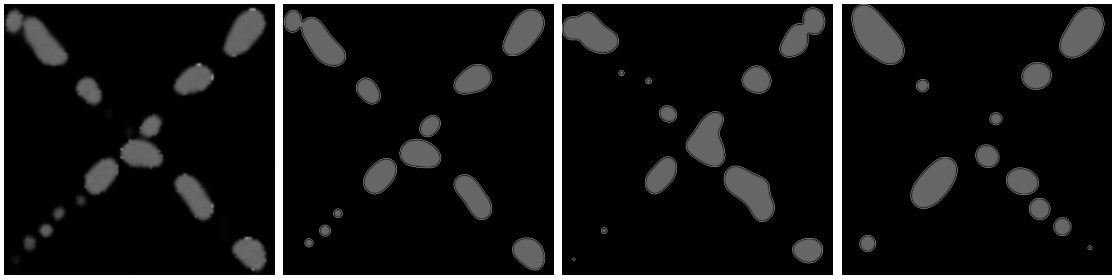
(a) $\sigma_y = 260$ MPa.



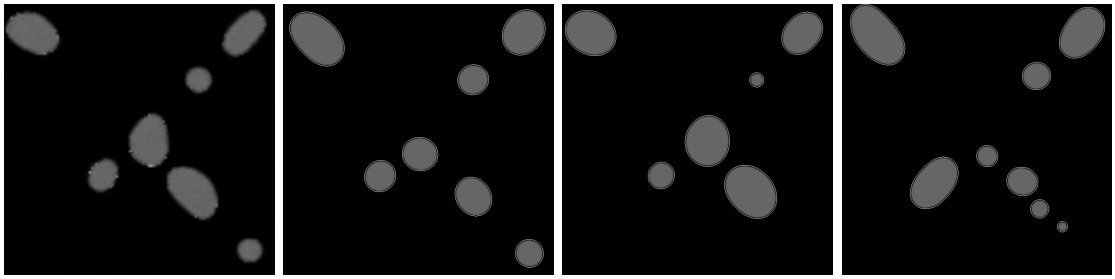
(b) $\sigma_y = 225$ MPa.



(c) $\sigma_y = 190$ MPa.



(d) $\sigma_y = 155$ MPa.



(e) $\sigma_y = 120$ MPa.

Figure 7.5: Predicted micrographies (first column) compared with phase-field extracted micrographies (second, third and fourth columns) for similar values of σ_y.

Chapter 8

Conclusions and perspectives

8.1 Conclusions

This project had as objective to develop a prediction algorithm that, by providing the mechanical properties of the desired microstructure, enabled to generate a corresponding micrography. The task was accomplished by using a wide variety of tools like the image analysis software and machine learning structures such as a Variational Autoencoder and Artificial Neural Networks. Finally, a successful performance of the overall parts of the project made possible reaching the following conclusions:

1. Confirming the results obtained in Zhao et al. (2019), it is possible to generate virtual microstructures from the desired mechanical properties by using artificial intelligence techniques. As this method has not gone much into the specific case of the AlSi10Mg alloys and the LPBF manufacturing process, it is expected that the same results can be reproduced for other materials always when an accurate model to predict the mechanical properties of the material and an image analysis software to obtain its physical features are available.

2. The microstructure analysis through the use of computer vision enables to retrieve most of the necessary features of the microstructure to predict its mechanical properties. Problems can be faced with the detection of characteristics like the distribution of the different phases or the size of elements such as precipitates or cells. The analyses were carried out in virtual micrographies reaching results close to the real properties of the AlSi10Mg alloys, but for better performance as well as for the application in real microstructures it is recommended to use state of the art softwares for image analysis such as *ImageJ*.

3. It was proved that the use of VAE as a support for prediction algorithms applied to images greatly decreases the computational resources required for training, enabling larger training data sets and therefore a better performance. The main drawback of this tool is a small loss in the quality of the image after reconstruction but looking at the results obtained and considering the benefits it is considered negligible.

4. Not only one but many prediction algorithms are able to predict, with more or less accuracy, the micrographies associated to a given mechanical property. The use of ANN for this task has been proven particularly successful, reaching very accurate reconstructions of the virtual microstructures with a relatively affordable training time and expense of computational resources.

8.2 Perspectives

The current master thesis is considered as a first stage of a more ambitious project, enabling us to test the viability of using artificial intelligence for material science purposes and to evaluate the different tools that contribute to the best possible performance. Having that in mind, it is necessary to enumerate the aspects to improve and the future work to be done in order to improve the results obtained here.

1. The artificial vision module was found to struggle with the quantification of some physical features of the microstructure like the size of the precipitates or the differentiation of the cellular and eutectic phases in the AlSi10Mg alloys. For this reason it is highly recommended to try using more advanced image processing softwares in order to ensure the quality of the results.

2. Further research is required in order to find a model that computes the yield stress for all the possible microstructure morphologies of the AlSi10Mg alloys. Models to predict the yield stress when the microstructure has the eutectic wall unbroken and the cellular region well defined have been proven very accurate. Nevertheless, when applying heat treatments that destroy the eutectic wall and trigger the precipitation of Si the models that predict the yield stress may not be valid anymore (Table 7.1). For this issue to be solved it is necessary to find a model that predicts the yield stress value when the microstructure is simply an α-Al matrix with Si precipitates, splicing it with the first one to be capable to accurately predict σ_y for all the possible AlSi10Mg microstructures.

3. Regarding the VAE, for building an algorithm capable of handling more diverse morphologies of the microstructure it is necessary to enlarge the size of the training data set. It is recommended to create new initial microstructures (Section 3.3) for considering all the possible evolutions that the AlSi10Mg alloys can have under thermal treatments. As the training data set will be increased it is also suggested to find a more optimal structure of the VAE, including the size of the latent vector, in order to find the best compromise between computational resources and encoding-decoding performance.

4. For the ANN used for the prediction of the latent vectors, a study of the hyperparameters and the structure of the network would enable a possibly better performance and a shorter training stage. It is also recommended to carry out a small hyperparameter study for the other prediction algorithms as well as for new ones, as they can possibly outperform the ANN using different configurations than the ones tested in this project.

5. Finally, it is highly suggested to contrast the results obtained with experimental data in order to give more validity to this research. The sections of feature extraction and yield stress computation need to be tested also for real microstructures. For using this same procedure of training in a prediction algorithm with real microstructures it is necessary to, firstly, improve the feature extraction and reproduce its results for real materials, and secondly, generate a large amount of micrographies for training the VAE and ANN, which may require large volumes of time and resources.

Appendix A

Strain-stress curve prediction

List of parameters

Symbol	Designation	Units
b	magnitude of Burgers vector of α-Al phase	$[nm]$
f_i	volume fraction for phase i	$[vol.\%]$
G	shear modulus of α-Al phase	$[GPa]$
k_{recov}	dynamic recovery term	$[-]$
k_{stord}	dislocation storage term due to trapping of dislocations by other dislocations	$[-]$
k_{storp}	storage term of geometrically necessary dislocations due to non shearable particles	$[m^{-1}]$
M	Taylor's factor	$[-]$
α_d	constant related to the dislocation strengthening	$[-]$
ε^T	overall strain of the material	$[-]$
$d\varepsilon^T$	increment of strain of the overall material	$[-]$
ε_i	strain of phase i	$[-]$
$d\varepsilon_i$	increment of strain in phase i	$[-]$
ρ_d	dislocation density	$[m^{-2}]$
ρ_{d0}	initial value of the dislocation density	$[m^{-2}]$
σ^T	overall stress beared by the material	$[MPa]$
σ_i	stress beared by the phase i	$[MPa]$
σ_0	intrinsic yield stress of aluminium	$[MPa]$
σ_{ss}	solid solution strengthening component	$[MPa]$
σ_p	precipitation strengthening component	$[MPa]$
σ_d	dislocation strengthening component	$[MPa]$

A.1 Introduction

The prediction of the yield stress, if done accurately, enables metallurgists to directly obtain certain properties of the material from the characteristics of the structure, reducing costs and time spent in tests for measuring hardness or deformation.

Once the yield stress prediction is achieved in Chapter 2 further calculations are done to reproduce the strain-stress curve of the material, comparing it to experimental results afterwards. The prediction of the strain-stress curve, in one way, keeps testing the validity of

the yield stress model used. It also enables cheaper and faster ways of predicting the material behavior than experimental tensile tests.

This appendix covers the procedure used for predicting the strain-stress curve in AlSi10Mg alloys, starting with the development of the equations in Section A.2. The parameters used for the calculation are given in Section A.3 and finally the results and comparisons with real experimental results are shown in Section A.4.

A.2 Equations and model development

Both the theoretical model and the parameters used for the strain-stress curve prediction are taken from Delahaye (2022). The model consists of an iterative process that solves Eq. (A.6), giving the evolution of the strain and stress when traction forces are applied.

The model starts with the value of the yield stress at 0.2% of plastic deformation, calculated in Chapter 2. The only contribution to the hardening of the material that changes when plastic deformation is applied is the dislocations hardening component, σ_d. The approach taken for the evolution of the dislocation contribution is the one proposed by Bouaziz and Buessler (2002), where, for multiphase materials, the mechanical work increment is equal in both phases (Eq. (A.1)).

$$\sigma_{cell} d\varepsilon_{cell} = \sigma_{eu} d\varepsilon_{eu} \tag{A.1}$$

σ_{cell} and σ_{eu} represent the stress at each phase while $d\varepsilon_{cell}$ and $d\varepsilon_{eu}$ represent the strain increment. At the same time, a linear law of mixtures is considered for obtaining both the overall stress

$$\sigma^T = f_{cell}\sigma_{cell} + f_{eu}\sigma_{eu} \tag{A.2}$$

and strain

$$\varepsilon^T = f_{cell}\varepsilon_{cell} + f_{eu}\varepsilon_{eu} \tag{A.3}$$

of the material, being f_{cell} and f_{eu} the volume fractions of the two phases.

As it is studied in Chen et al. (2017), strain and stress are different for the two phases. This means that they should be calculated separately, to later obtain the overall value through an homogenization process, as it is done in Chapter 2. Introducing Eq. (A.1) in Eq. (A.2) and Eq. (A.3) the strains of the cellular and eutectic phases can be defined by the overall strain, $d\varepsilon^T$.

$$d\varepsilon_{cell} = \frac{d\varepsilon^T}{f_{cell} + \frac{\sigma_{cell}}{\sigma_{eu}} f_{eu}} \tag{A.4}$$

$$d\varepsilon_{eu} = \frac{d\varepsilon^T}{f_{eu} + \frac{\sigma_{eu}}{\sigma_{cell}} f_{cell}} \tag{A.5}$$

After differentiating the behavior of both phases, the evolution of the dislocation density and the precipitation contribution to the hardening are calculated separately. For doing so the Kocks-Mecking-Estrin model (Krausz and Krausz (1996)) is used, defining the increase of the dislocation density through Eq. (A.6).

$$\rho_d = \rho_{d0} + M\left(\frac{k_{stord}}{b}\sqrt{\rho_d} - k_{recov}\rho_d + \frac{k_{storp}}{b}\right)\varepsilon \tag{A.6}$$

being ρ_d, ρ_{d0} and ε independent for each phase.

Solving the equation that defines the dislocation density requires an iterative process. Many methods can be chosen but, as a good compromise between accuracy and speed, the Dormand-Prince scheme is used. This method consists of two Runge-Kutta schemes, one of 4^{th} order and one of 5^{th} order, that, when computed at the same time and compared, enable a much higher reduction of the error than with a simple RK. The implementation used is the one in the *Scipy* library from *Python*.

Once the evolution of the dislocation density is predicted it is used to calculate the σ_d hardening contribution, as it is shown in Eq. (A.7).

$$\sigma_d = \alpha_d MGb\sqrt{\rho_d} \tag{A.7}$$

obtaining a different contribution for each phase ($(\sigma_d)_{cell}$ and $(\sigma_d)_{eu}$).

Finally, the stress is calculated by the addition of all the hardening contributions

$$\sigma_{cell} = (\sigma_0)_{cell} + (\sigma_{ss})_{cell} + (\sigma_p)_{cell} + (\sigma_d)_{cell} \tag{A.8}$$

$$\sigma_{eu} = (\sigma_0)_{eu} + (\sigma_{ss})_{eu} + (\sigma_p)_{eu} + (\sigma_d)_{eu} \tag{A.9}$$

and the homogenization process is carried out to obtain the overall stress

$$\sigma^T = f_{cell}\sigma_{cell} + f_{eu}\sigma_{eu} \tag{A.10}$$

A.3 Input parameters

Once the model used to reproduce the strain-stress curve is defined its accuracy needs to be tested. The example taken to ensure the robustness of the model is the one used in Delahaye (2022), where the same model has been reproduced and compared with experimental values of a tensile test. Thus, the parameters used for the test are taken from the mentioned thesis as well as from its references, all given in Table A.1.

A.4 Results

After introducing the parameters from the previous section and solving the differential equation the evolution of the dislocation density in the material is obtained. This dislocation density is the parameter that guides the increase of σ_d and therefore the increase of the stress that the material is bearing. The dislocation density, as it is mentioned in Section A.2, is different for both the cellular and eutectic phases, and their evolutions with the plastic strain are shown in Fig. A.1.

Parameters	Units	Value		Reference
		Cell	**Eutectic**	
f_i	$[vol.\%]$	0.8	0.2	Delahaye (2022)
ρ_{d0}	$[m^{-2}]$	10^{13}		Nicholson (1962)
k_{stord}	$[-]$	$2.28 \cdot 10^{-1}$	$1.55 \cdot 10^{-3}$	
k_{recov}	$[-]$	$1.72 \cdot 10^{1}$	9.15	Delahaye (2022)
k_{storp}	$[m^{-1}]$	0	$1.03 \cdot 10^{7}$	
α_d	$[-]$	0.3		Chen et al. (2017)
M	$[-]$	3.06		
G	$[GPa]$	25.4		Cheng et al. (2003)
b	$[nm]$	0.283		
σ_0	$[MPa]$	10	10	Table 2.2
σ_{ss}	$[MPa]$	138.65	0	Table 2.2
σ_p	$[MPa]$	0	444.58	Table 2.2

Table A.1: Example parameters for the strain-stress curve prediction from Delahaye (2022).

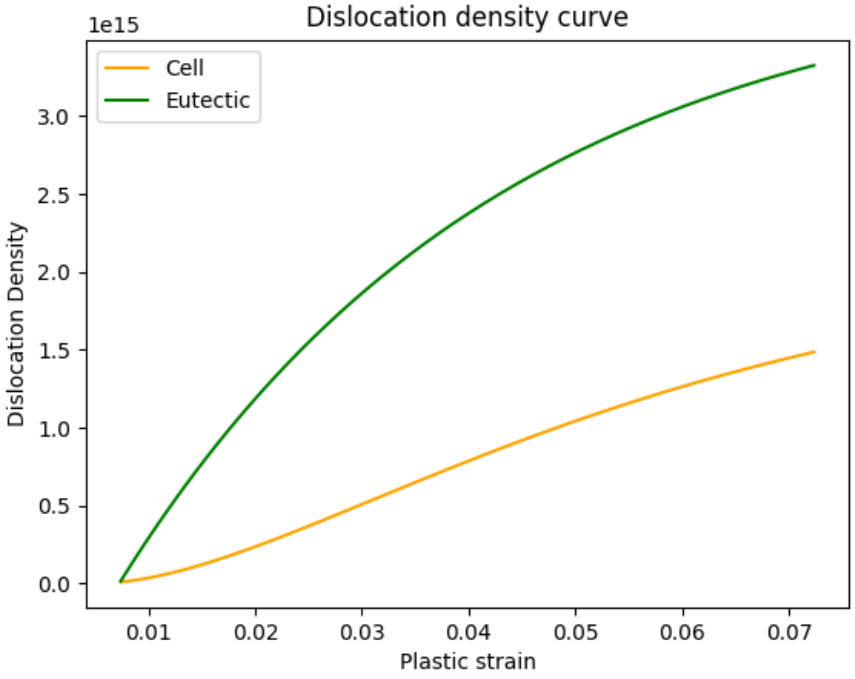


Figure A.1: Dislocation density evolution with plastic strain for cellular and eutectic phases.

In this graph is appreciated the much larger hardening (dislocation increase) of the eutectic phase than the cellular one, as Si precipitates impede the dislocation movement making them to accumulate. Once the dislocation density is obtained the strain-stress curve can be computed. In Fig. A.2 is shown the evolution of the stress for the cellular phase (green), the eutectic phase (blue) and the overall material (red), together with the experimental value of a tensile test in an AlSi10Mg alloy.

As it can be easily seen in both graphics, the initial value of the plastic strain is not null. This is due to the addition of an initial value of $\varepsilon = 0.65$ to fit the experimental curve, so that both curves start at the same point, which is the yield stress point.

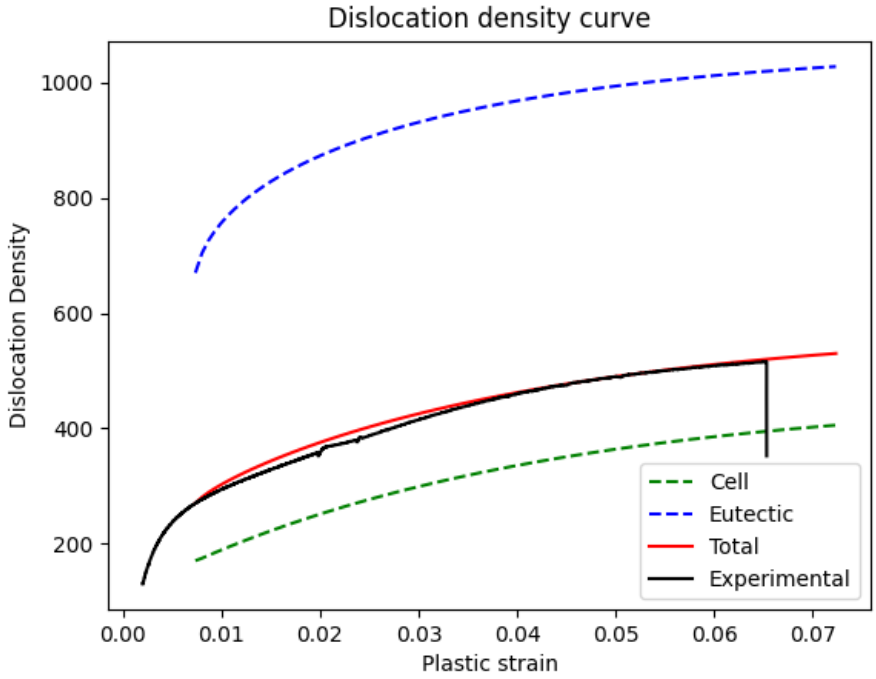


Figure A.2: Predicted and experimental strain-stress curve for tensile test in an AlSi10Mg alloy.

Bibliography

Edoardo M Airoldi. Getting started in probabilistic graphical models. *PLoS Computational Biology*, 3(12):e252, 2007.

Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

Patrick Altschuh, Yuksel C Yabansu, Johannes Hötzer, Michael Selzer, Britta Nestler, and Surya R Kalidindi. Data science approaches for microstructure quantification and feature identification in porous membranes. *Journal of Membrane Science*, 540:88–97, 2017.

Matthew Ashman, Jonathan So, Will Tebbutt, Vincent Fortuin, Michael Pearce, and Richard E Turner. Sparse gaussian process variational autoencoders. *arXiv preprint arXiv:2010.10177*, 2020.

S Bulent Biner et al. *Programming phase-field modeling*. Springer, 2017.

O Bouaziz and P Buessler. Mechanical behaviour of multiphase materials: an intermediate mixture law without fitting parameter. *Metallurgical Research & Technology*, 99(1):71–77, 2002.

Andrew Campbell, Paul Murray, Evgenia Yakushina, Stephen Marshall, and William Ion. New methods for automatic quantification of microstructural features using digital image processing. *Materials & Design*, 141:395–406, 2018.

B Chen, SK Moon, X Yao, G Bi, J Shen, J Umeda, and K Kondoh. Strength and strain hardening of a selective laser melted alsi10mg alloy. *Scripta Materialia*, 141:45–49, 2017.

LM Cheng, WJ Poole, JD Embury, and DJ Lloyd. The influence of precipitation on the work-hardening behavior of the aluminum alloys aa6111 and aa7030. *Metallurgical and Materials Transactions A*, 34:2473–2481, 2003.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

Jocelyn Delahaye. How to feed and validate a phase-field model predicting the evolution of microstructures and properties in alsi10mg processed by selective laser melting. 2022.

Daria Doncevic and Carl Herrmann. Biologically informed variational autoencoders allow predictive modeling of genetic and drug-induced perturbations. *Bioinformatics*, 39(6):btad387, 2023.

Seifallah El Fetni, Jocelyn Delahaye, Laurent Duchêne, Anne Mertens, and Anne Habraken. Adaptive time stepping approach for phase-field modeling of phase separation and precipitates coarsening in additive manufacturing alloys. In *COMPLAS 2021-XVI International Conference on Computational Plasticity*, 2022.

Seifallah Fetni, Thinh Quy Duc Pham, Truong Vinh Hoang, Hoang Son Tran, Laurent Duchêne, Xuan-Van Tran, and Anne Marie Habraken. Capabilities of auto-encoders and principal component analysis of the reduction of microstructural images; application on the acceleration of phase-field simulations. *Computational Materials Science*, 216:111820, 2023.

Evelyn Fix and Joseph Lawson Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247, 1989.

Jacques Friedel. *Dislocations: international series of monographs on solid state physics*, volume 3. Elsevier, 2013.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

Metod Jazbec, Matt Ashman, Vincent Fortuin, Michael Pearce, Stephan Mandt, and Gunnar Rätsch. Scalable gaussian process variational autoencoders. In *International Conference on Artificial Intelligence and Statistics*, pages 3511–3519. PMLR, 2021.

Yongju Kim, Hyung Keun Park, Jaimyun Jung, Peyman Asghari-Rad, Seungchul Lee, Jin You Kim, Hwan Gyo Jung, and Hyoung Seop Kim. Exploration of optimal microstructure and mechanical properties in continuous microstructure space using a variational autoencoder. *Materials & Design*, 202:109544, 2021.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

Sotiris B Kotsiantis. Decision trees: a recent overview. *Artificial Intelligence Review*, 39: 261–283, 2013.

Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

Alexander S Krausz and K Krausz. *Unified constitutive laws of plastic deformation*. Elsevier, 1996.

Rea Labusch. A statistical theory of solid solution hardening. *physica status solidi (b)*, 41(2): 659–669, 1970.

Xihe Liu, Congcong Zhao, Xin Zhou, Zhijian Shen, and Wei Liu. Microstructure of selective laser melted alsi10mg alloy. *Materials & Design*, 168:107677, 2019.

Juan Guillermo Santos Macías, Thierry Douillard, Lv Zhao, Eric Maire, Grzegorz Pyka, and Aude Simar. Influence on microstructure, strength and ductility of build platform temperature during laser powder bed fusion of alsi10mg. *Acta Materialia*, 201:231–243, 2020.

OR Myhr, Ø Grong, and SJ Andersen. Modelling of the age hardening behaviour of al–mg–si alloys. *Acta Materialia*, 49(1):65–75, 2001.

RB Nicholson. Quenching defects in solid solutions and their effect on precipitation. *J. Phys. Radium*, 23(10):824–827, 1962.

Eun Jung Seo, Lawrence Cho, Yuri Estrin, and Bruno C De Cooman. Microstructure-mechanical properties relationships for quenching and partitioning (q&p) processed steel. *Acta Materialia*, 113:124–139, 2016.

Geoffrey Ingram Taylor. The mechanism of plastic deformation of crystals. part i.—theoretical. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 145(855):362–387, 1934.

Christopher Williams and Carl Rasmussen. Gaussian processes for regression. *Advances in neural information processing systems*, 8, 1995.

Jonathan Zalger. Application of variational autoencoders for aircraft turbomachinery design. Technical report, Technical report, 2017.

Lv Zhao, Juan Guillermo Santos Macías, Lipeng Ding, Hosni Idrissi, and Aude Simar. Damage mechanisms in selective laser melted alsi10mg under as built and different post-treatment conditions. *Materials Science and Engineering: A*, 764:138210, 2019.