NTNU
Kunnskap for en bedre verden

Department of Structural Engineering

TKT4550 - Structural Engineering, Specialization Project

Predicting Thermal Fields in Additive Manufacturing by FEM simulations and Machine Learning

Author:
Kari Lovise Ness

Supervisor:
Prof. Zhiliang Zhang

Co-supervisor:
Li Sun

November 30, 2020

1 Abstract

This paper explores the theory needed to perform surrogate modelling that can predict thermal fields in additive manufacturing processes. By creating a finite element model (FEM), one can utilize machine learning to create a surrogate model that can predict the thermal fields in additive manufacturing, which will lead to decreased costs and time spending, and widen the application of additive manufacturing.

Additive manufacturing is a manufacturing technology with the potential to make complex geometries through layer-by-layer deposition. The additive manufacturing process includes a moving, focused heat source which melts the material. This is followed by rapid consolidation, which leads to residual stresses in the geometry. Residual stresses relate to decreased mechanical properties, and the ability to predict this is necessary to make additive manufacturing widely applicable. The current state-of-the-art is to conduct FEM simulations of the additive manufacturing process. However, as FEM simulations are costly and time consuming, the FEM models are not applicable for realistic optimization.

As the additive manufacturing process is highly repetitive, it is an ideal case for machine learning. Creating a surrogate model of the thermal fields in additive manufacturing would be useful to decrease costs and make it less time consuming, and thereby enabling real-time computations. The aim of creating a surrogate model with data from FEM models is to retain the same accuracy as with FEM simulations which will make additive manufacturing applicable in a wider range of use cases.

Together with the theory related to machine learning and additive manufacturing, results from FEM simulations will be presented and discussed. The results are samples of FEM models that will be used when surrogate modelling is performed.

Table of Contents

List of Figures

List of Tables

Nomenclature

σ_{res}	Residual stress
a_f	Front semi-axes of double-ellipsoid heat model
a_r	Rear semi-axes of double-ellipsoid heat model
f_f	Fraction of deposited heat in front quadrant of double-ellipsoid heat model
f_r	Fraction of deposited heat in rear quadrant of double-ellipsoid heat model
T_{am}	Ambient temperature
T_m	Maximum temperature of model
T_p	Peak nodal temperature
2D	Two dimensional
3D	Three dimensional
$x^{(i)}$	Input variable
$y^{(i)}$	Output variable
h	Hypothesis
AI	Artificial intelligence
AM	Additive manufacturing
CNN	Convolutional neural networks.
DED	Directed energy deposition
FE	Finite element
FEA	Finite element analysis
FEM	Finite element model
GUI	Graphical user interface
HIZ	Heat influence zone
k	Thermal conductivity
ML	Machine learning
MLP	Multi-layer perceptron
MSE	Mean squared error
ODB	Output database
OOP	Object-oriented programming
PBF	Powder bed fusion
Q	Power input
ReLU	Rectified linear unit
ResNet	Residual network
SLP	Single-layer perceptron
SM	Surrogate model

2 Introduction

2.1 Additive manufacturing

Additive manufacturing (AM) is a manufacturing technique were 3D parts are built by the addition of thin feedstock layers according to a computer-aided design (CAD) model [22]. According to the way the feedstock are delivered to the part, one can categorize additive manufacturing into two categories; directed energy deposition (DED) and powder bed fusion (PBF) [47]. DED is a group of AM processes that adds the material and the heat source simultaneously [42]. PBF is a group of AM processes where a heat source is applied to particles in a powder bed. As each layer is created new powder is spread over the build area and the powder bed indexes downwards to give space for the new layer [20]. For both processes, the construction of a part involves melting of feedstock a focused energy source, followed with rapid consolidation [9].

Additive manufacturing has been researched and utilized for several decades. However, as a commonly known phenomenon, 3D-printing and AM is seen as recent. There are a great deal of excitement over AM as it is reinvents manufacturing by enabling manufacturing of complex geometries with near net shape. Near net shape implies being close to the finished product or the net shape, which simplifies the manufacturing process significantly [21].Due to problems with part accuracy, a limited variety of materials and mechanical performance of the part, the application of additive manufacturing has mainly been as a rapid prototyping technique. In rapid prototyping, one aims to construct a engineering prototype of a part with as low lead time as possible. This enables the engineers and designers to perform tests and get a physical feeling of the part early on in the design process. As rapid prototyping usually value fast part delivery and complexity of the part over specific materials, accuracy or mechanical properties, rapid prototyping has proved to be an ideal area of application for additive manufacturing. However, as the additive manufacturing technology has evolved to create parts with higher accuracy and improved mechanical properties, its fields of applications are rapidly growing [22]. Additive manufacturing is today utilized across many industries, especially for manufacturing of intricate designs, low-volume manufacturing or one-of-a-kind manufacturing [37].

Computational simulations are essential in the design and optimization process in AM as it enables the designers and engineers to avoid trial and error on expensive experimental tests. Finite Element Analysis (FEA) is a widely used numerical technique for finding approximate solutions to boundary value problems of partial differential equations [31]. By building finite element models (FEM). the user may analyze a given problem by splitting it into numerous finite elements and simulating the behavior of each element under certain boundary conditions. Together the finite elements describe the behavior of the total problem [55]. However, even though the finite element analysis is able to compute numerical approximations of a problem well, it is computationally costly and time-consuming.

2.2 Machine Learning

Machine learning (ML) is a subfield of artificial intelligence (AI), and are methods enabling a computer to improve its performance through experience. Currently, machine learning is one of the top trending technologies, but while machine learning gets increasing attention and interest in today's society, machine learning as a field is not new. Several scientists such as Thomas Ross and Alan Turing did substantial work on building machines that were able learn, and in 1959, Arthur Samuel defines the term machine learning as a *"Field of study that gives computers the ability to learn without being explicitly programmed"*, [41]. Later on Tom M. Mitchell defined learning as *"A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."*, [34].

One usually characterize machine learning method based on the learning strategy. The main machine learning categories is broadly classified into supervised, unsupervised and reinforcement learning, as seen in figure 1. In supervised learning, the model is trained on paired data, with both input and output, in order to predict future events. In unsupervised learning, the model is trained on unlabeled data with no guidance. In this way, the model is looking for hidden patterns in the given data. Reinforcement learning is based on a series of feedback/reward cycles, and learns by interacting with its environment, [7]

During the project, supervised learning will be implemented. Supervised learning trains on classified data in order to create a mapping function that can predict future events. The classified data in supervised learning is called a **training set**, and consists of several **training examples**. Each training example ($x^{(i)}$,$y^{(i)}$) consists of one **feature** or input variable, $x^{(i)}$, and one **target** or output variable, $y^{(i)}$. Through training on the training set, one wish to learn a function $h : X \mapsto Y$. h is commonly known as a **hypothesis**, and is regarded good if it is able to accurately predict future values of y, [19]. A process flow diagram of supervised learning can be seen in figure 2.

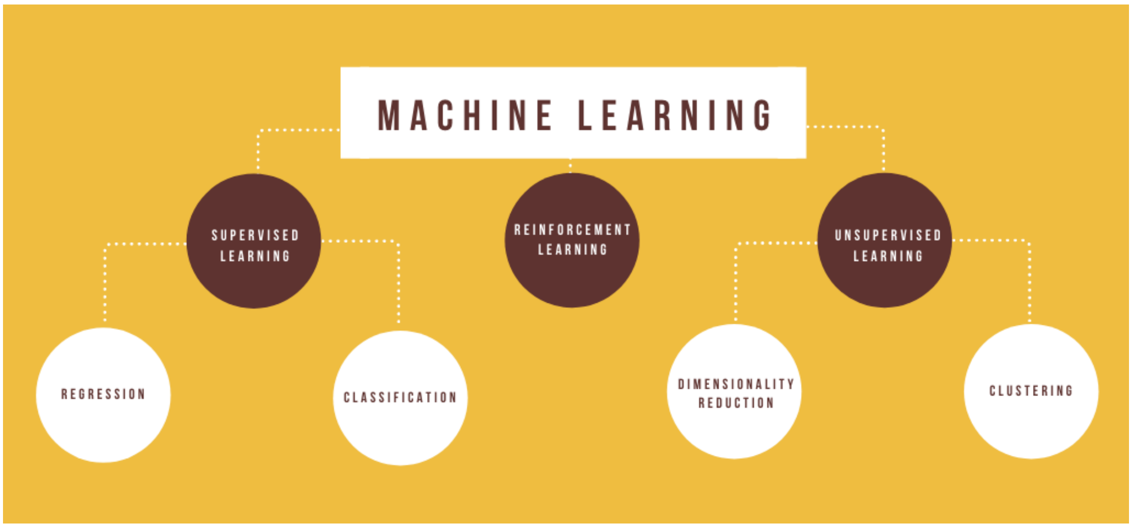


Figure 1: Machine learning categories

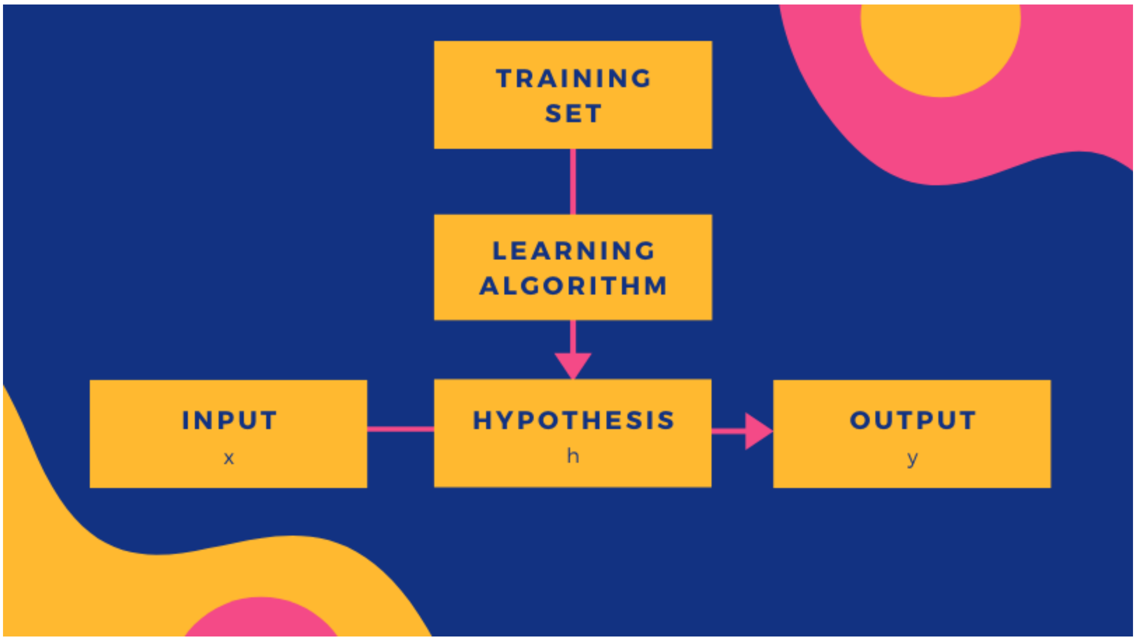


Figure 2: Supervised learning process

Supervised learning is generally classified into two categories; regression and classification. Regression and classification are powerful methods that enables the user to classify and and process data using machine language, and are widely used in industries such as medicine and finance, [36]. With increased computing powers, these and other machine learning methods will be applicable as solution methods to ever more technological challenges.

2.3 Machine learning and additive manufacturing

As additive manufacturing is gaining popularity beyond prototyping, the interest in the dynamic process of AM is increasing, both with respect to thermal and mechanical properties. Although significant progress has been made in the field, both experimentally and by modeling, the computation of thermal fields and mechanical properties of parts in additive manufacturing is still seen as a big challenge. The experimental results have given insight in the field, but they are expensive and due to physical restrictions, often have a limited scope. Simulations are widely used to analyze complex problems, and as the computing powers continuously increase, the models are seen to predict the real-world more and more accurately [27]. However, at the same time the models are increasing complex, and unavoidably more expensive. Computational modelling of additive manufacturing is no exception, and has problems with high computational cost, large memory requirements and long computing time, which makes the simulations demanding to perform in realistic computing [40].

The difficulties seen in the traditional simulations of AM leads us to the motivation of utilizing machine learning and surrogate modeling. By taking advantage of the high level of redundancy, repeatability and periodicity in AM, several studies such as that of Mozzafar et al.[35], Francis et al. [35, 37] and Roy and Wodo [40] has been able to build efficient and accurate surrogate models that are able to predict thermal profiles of AM parts using ML. Mozzafar et al. proposed a data-driven approach to predict the thermal behavior in a directed energy deposition process using recurrent neural networks. The authors reported high accuracy of the predicted thermal profiles. However, even through the model was performing well, it requires a large data set to train the model, which is time-consuming to acquire. Francis et al. predicted distortion of a laser-based AM process by applying deep learning on thermal images acquired in experiments. Similarly to the model proposed by Mozzafar et al., a large dataset was required. Roy et al. addressed this challenge, and built a machine learning model of thermal profiles that required a significantly smaller dataset whilst still achieving a competitive accuracy.

The thermal fields are highly relevant due to its close link with residual stresses caused by temperature gradients and cooling and heating rates. This project will explore the prediction of thermal fields by replacing the computational model with a surrogate model (SM) trained with machine learning. A SM, also known as approximation models, is a model of a model, and replace expensive processing by approximation of input-output responses in the model without loosing accuracy, [27, 40]. By creating a SM that can predict the temperature fields of a part, the analysis of

AM models are made more cost and time efficient, and in addition widen the range of fields that can utilize AM due to real-time computations.

3 Method

The goal of this project is to build a surrogate model that can accurately and efficiently predict thermal fields in additive manufacturing. Figure 3 is a process diagram of steps that are necessary to perform in order to build the specified SM. The details and theoretical background of each step will be elaborated in the subsequent sections. In addition, each step will be contextualized to published work related to the topic. In particular, the work done by Roy et al. [40], Mozzafar et al. [35, 37] and Francis et al. [13] will be highlighted.

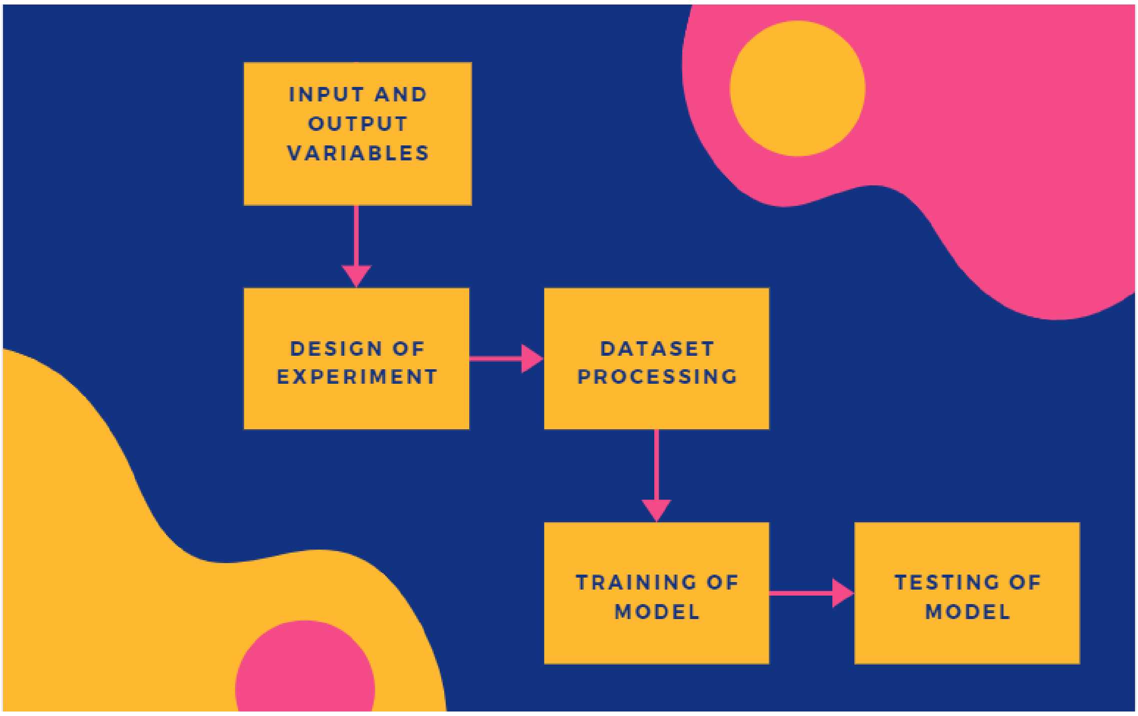


Figure 3: Process diagram of the creation of a SM

3.1 Selection of input and output variables

The choice of input and output variables are problem dependent. This project explores the prediction of thermal fields in AM utilizing FEM and ML, and the SM will therefore have the temperature profile of a part, T(t), as the desired output. Thermal fields are of special interest in the field of AM due to its connection with residual stresses, σ_{res}, and deformations, which may lead to geometrical distortions and changed mechanical properties. Residual stresses are defined as stresses that exists in an elastic body, even in the absence of thermal or mechanical external loads [29]. The origins of residual stresses include spatial temperature gradients and thermal expansion and contractions due to heating and cooling. The spatial temperature gradients are connected with the maximum temperature of the whole model, T_m. Thermal contraction and expansion depends on the peak nodal temperature, T_p. In addition strain compatibility, meaning an uneven distribution of inelastic strains, force equilibrium and constitutive stress-strain behavior will affect the residual stress in a part [46]. Being able to predict the levels of residual stress is important in order to produce reliable parts with known limitations. However, it is also possible to decrease the problems related to residual stress through its design. By being able to successfully predict the levels of residual stress in an additive manufactured part, one can compensate for the corresponding distortions [13]. An example of the process of design compensation can be seen in figure 4.

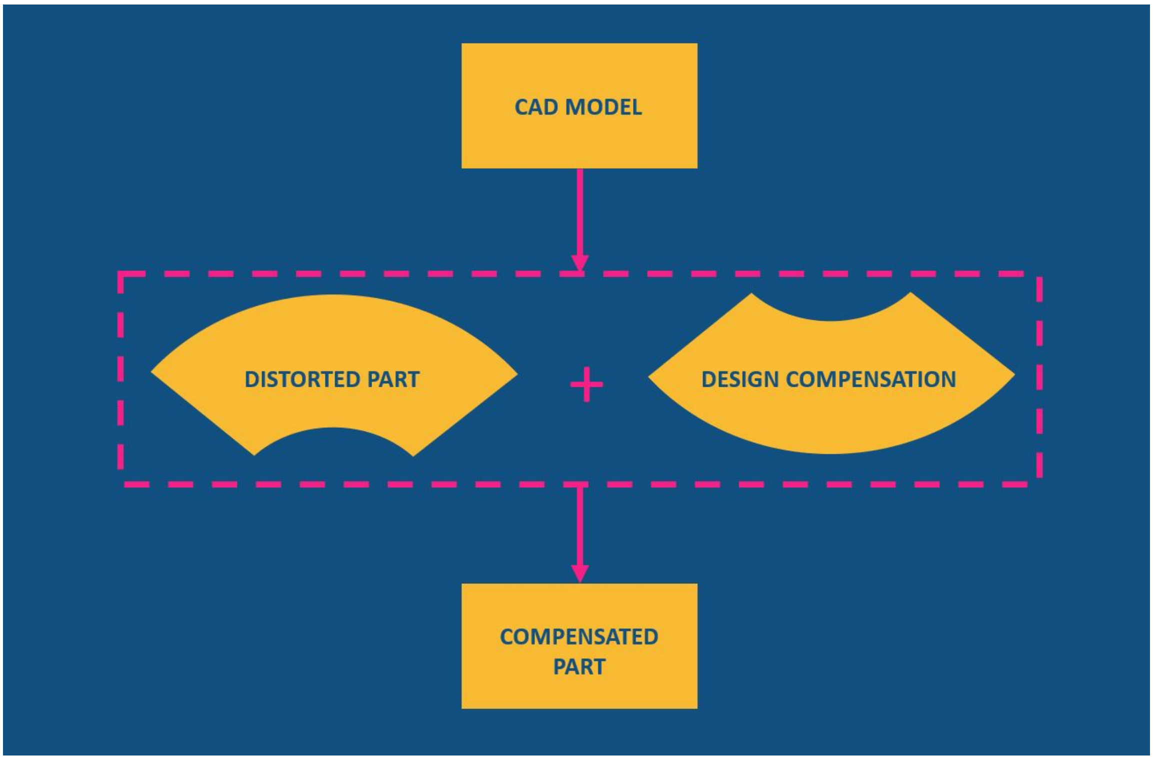


Figure 4: Example of design compensation for a rectangular part.

The input variables of the SM will be information about the geometry and AM process of the part. This data is experimental data retrieved from a FEA in Abaqus. The design of experiments and generation of datasets will be further explained in section 3.2.

3.2 Design of experiments to generate training data

The training data in this project will be experimental data generated through FEA of AM parts in Abaqus. Creating a SM that accurately predicts the temperature fields in AM is dependent on a big enough training set with data that accurately represents the reality. Creating accurate models that is able to represent factors such as material addition and temperature dependencies is therefore vital. In order to create a big enough training set, it will be necessary to perform several thermal simulations, and automation through Abaqus scripting with Python is therefore useful.

3.2.1 Challenges in FEM modelling of AM processes

Modeling of AM processes introduce new complexities and challenges that both increase modeling and simulation time of the processes. Some of the challenges will be discussed below.

Geometry of model:

Due to the changing geometry of AM parts as material is deposited, most AM parts can not be modeled in 2D, nor have symmetry planes. This means that simplifications to make the model more efficient is difficult, and the full model must therefore be built in a 3D finite element (FE) mesh [21].

Modelling the addition of material and heat input:

As mentioned in section 2, AM processes consists of deposition of feedstock and melting of the feedstock by a concentrated heat source. During fabrication, the concentrated heat source will travel over the area of the layer in order the melt the feedstock. When the feedstock melts, the feedstock is joined to the preceding layer before the material rapidly consolidates [14]. There are numerous proposals for heat source equations used for accurate modelling of the melt-pool. Amongst them is the double-ellipsoidal heat power density model that was introduced by Goldak et al. in 1985 [18, 44, 15]. This model is widely applied in literature where moving heat sources are modelled [31], and formulation of the model as shown in figure 5 can be

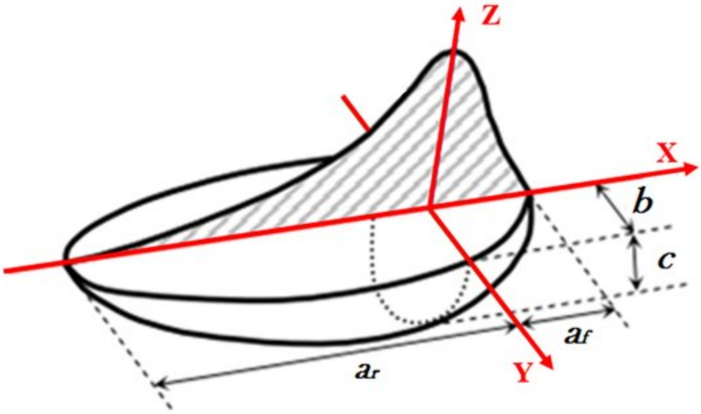


Figure 5: Goldak heat source model

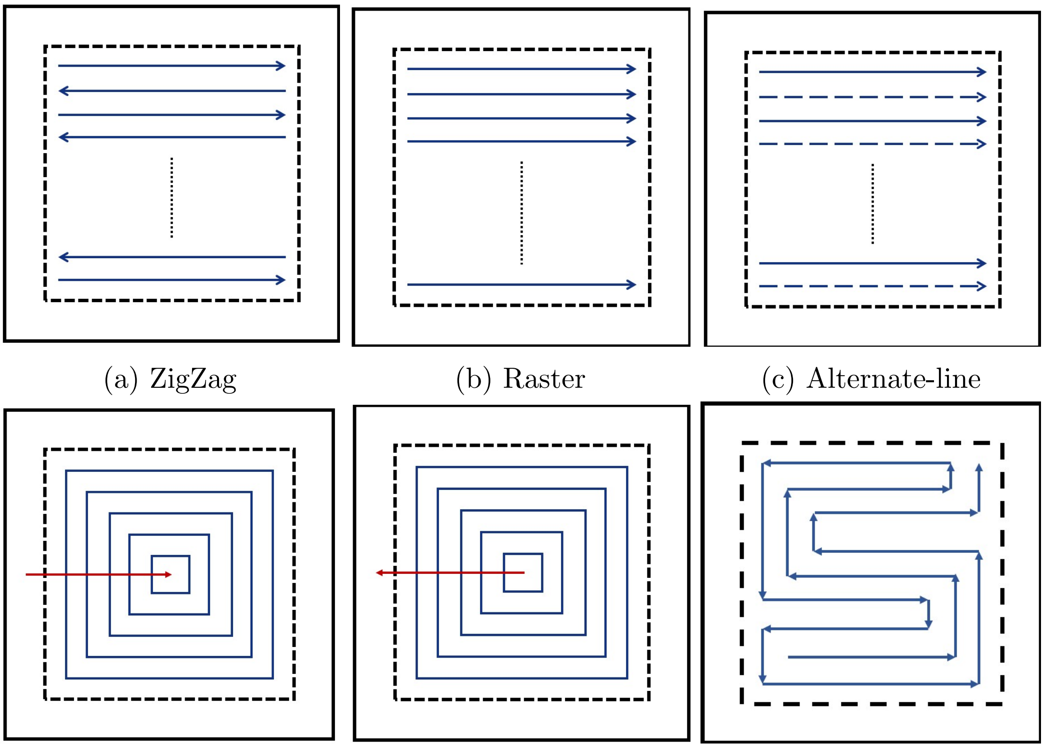
Source: [8]

expressed as

$$q_f(x, y, z) = \frac{6\sqrt{3} f_f \eta Q}{abc\pi\sqrt{\pi}} \exp\left(-\frac{3x^2}{a_f^2} - \frac{3y^2}{b^2} - \frac{3z^2}{c^2}\right) \tag{1a}$$

$$q_r(x, y, z) = \frac{6\sqrt{3} f_r \eta Q}{abc\pi\sqrt{\pi}} \exp\left(-\frac{3x^2}{a_r^2} - \frac{3y^2}{b^2} - \frac{3z^2}{c^2}\right) \tag{1b}$$

where equation 1a and 1b represents the front and rear quadrants of the moving heat source model respectively. In addition, x, y and z are the local coordinates of the ellipsoidal model, η is the arc efficiency, Q is the power input, a, b and c are the semi-axes of the heat source model. f_f and f_r are the fractions of deposited heat and represent the heat distribution of the heat flux in the front and rear quadrants, where $f_f + f_r = 2$ [15]. In order to obtain continuity between the front and rear quadrants of the model, $f_f = \frac{2a_f}{a_f+a_r}$ and $f_r = \frac{2a_r}{a_f+a_r}$ [30]. The geometry of the melt pool affects how big the zone of the surrounding material that is reheated when the heat source melts a new layer. Roy and Wodo defines this region as the heat influence zone (HIZ), which is the maximum distance from a heat source where a minimum observable thermal change occurs [40].



(a) ZigZag (b) Raster (c) Alternate-line

(d) In-out spiral (e) Out-in spiral (f) S-pattern

Figure 6: Common deposition patterns

The movement of the heat source is called deposition pattern or scanning strategy, and is one of the most important factors for the transient temperature distribution during an AM process [47, 16]. Some common deposition patterns in AM can be seen in figure 6, where the arrows are scan vectors that indicate the movement of the heat source. When the concentrated heat source moves over the material, temperature gradients, which are closely related to residual stresses, are generated. As the heat source transverses the part, heat accumulation occurs, which creates a thermal field with higher thermal gradients. In what extent heat accumulation occurs is closely related to the deposition pattern, in addition to other deposition parameters such as deposition speed and road size [40, 16]. As the heat accumulation depends on the spatial position of the heat source, rotation of the deposition pattern is an important measure to decrease the thermal gradients and get a uniform residual stress field. J. Robinson et al. found that the ideal rotation angle with respect to residual stresses is 90° [39], which is illustrated with a raster deposition pattern in figure 7. In addition, when looking at the microstructure of the cooled material, it is seen that the material inhibits a directional solidification texture [53]. The directional solidification texture gives each layer anisotropic mechanical properties. By rotating the deposition patter, one can reduce the anisotropic nature of the part.

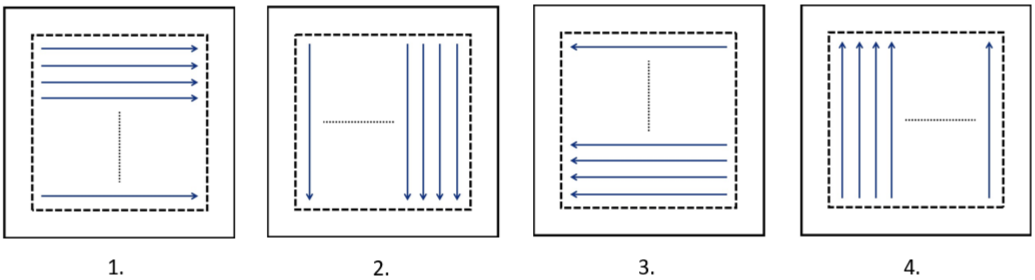


Figure 7: Rotation of deposition pattern with 90 degrees for each layer.

Accounting for phase transforms due to high temperatures:

The properties of microstructures are described by the amount of each phase, its distribution, and its composition using descriptors of size, shape, and relations between the phases [17]. The microstructure of a material is temperature-dependent due to temperature dependent phase-transforms. During a phase transformation, the microstructure of the material changes, which also leads to changed properties of the material [6]. As AM processes involves transient thermal distribution with large temperature changes, the microstructure of the material is important to consider. There exists functionality for microstructural finite element modelling, but implementing this in an AM model increases the complexity of the model. However, by choosing a material without phase transformations in the given temperature domain, the changes of the microstructure is not significant. With only one material phase, the effect of the microstructure is captured by the material with temperature dependent properties. Aluminum alloys are commonly used as the materials does not experience allotropic phase transforms [23].

3.2.2 Python scripting

Building and conducting analysis of finite element models are usually performed in a graphical user interface (GUI). The GUI works like a platform where the user can interact with the programme in a visual and understandable way. When the user modifies the model, the GUI generates commands in the programming language Python. The commands are sent to the Abaqus/CAE kernel, the brain of Abaqus/CAE, which creates an internal representation of the model [1].

All operations that are available in Abaqus GUI is also available through python commands and opens the possibility for fully automated database generation. In this project a programme for generation of AM models automatically was written, see Appendix A.1. Abaqus use Python 2.7, which is an old version of Python. As Python

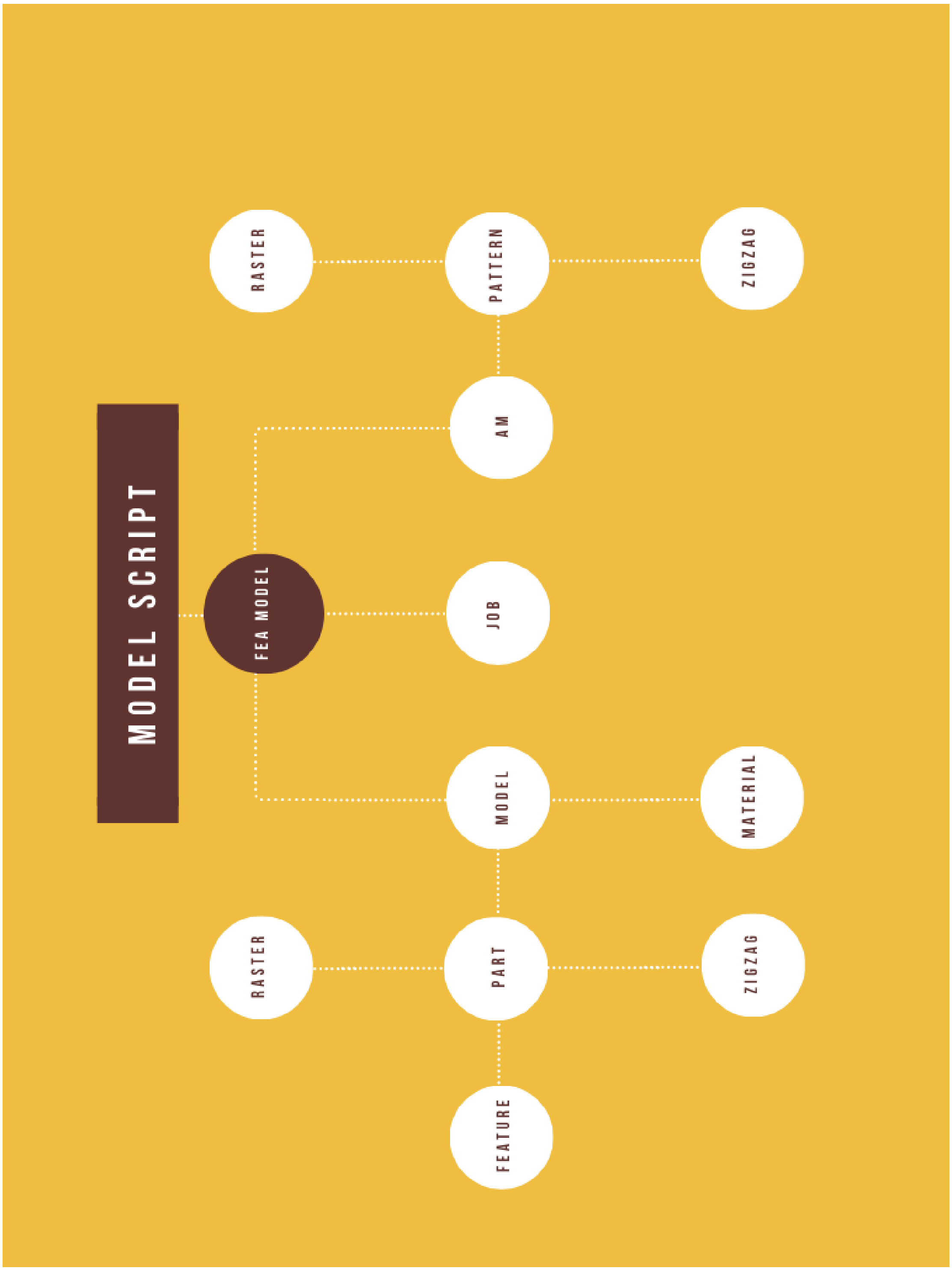


Figure 8: Hierarchical model of classes.

2.7 does not contain all packages from newer python versions, the script in A.1 is written to output a input file specific for each model. By calculating all variables outside Abaqus, all packages are available and importing files and information is easier. The script is written object-oriented. Object-oriented programming (OOP) is a programming paradigm that models objects and the relationship between objects.

Each entity belongs to a defined class, and each class has a series of attributes and methods. By utilizing the OOP approach, the programmer may divide a complex problem into smaller and more manageable parts [52]. OOP was necessary to use due to the hierarchical structure of Abaqus. For example, a part object has to belong to a model object, and being able to model the relationship between these entities is necessary. The hierarchical model of the program can be seen in figure 8, where FEA Model is the main class that assembles the model script, and the white circles are subclasses that are imported into FEA Model. All of the scripts can be found in Appendix A.

M. Roy and O. Wodo studied data-driven modeling of thermal history in additive manufacturing with a database built with FEA [40]. Their database consisted of models with identical properties except for the part lengths, which were increasing incrementally. Due to the similarities in the models, creating a script for automation of model generation may easily be done utilizing the scripts in Appendix A and a for loop that changes certain parameters in the script.

However, even though python scripting is useful for generating FEA models, python is even more important when it comes to post-processing of the results. When completing an analysis in Abaqus, a output database (ODB) file with the requested field output's is created. This file contains field information for every element in the model for each time increment. Each data point corresponds to a unique coordinate *(x,y,z,t)*, where *(x,y,z)* represent one element and *(t)* represents a timestep. This file usually contain complex data and has a massive size, which makes scripting necessary in order to process all the data points. Mozzafar et al. works with 9.05 million data points in *A real-time iterative machine learning approach for temperature profile prediction in additive manufacturing processes* [37], and it's therefore clearly necessary with efficient data handling.

3.3 Dataset processing

3.3.1 Preprocessing of datasets

The quality of the results of a machine learning analysis depends on the quality of the dataset and if it is preprocessed to fit the ML method. Data quality is reduced with presence of noise, outliers, lack of completeness (missing data points), lack of uniqueness (duplicate data points) and biased-unreprestative data [28]. When dealing with small datasets, corrective measures to improve data quality is often necessary to acquire satisfactory results. However, it is seen that increasing the dataset size increase the performance of ML algorithms, and can therefore decrease the need for corrective measures [50] . Due to the massive size and complexity of the datasets generated by FEA, it can be assumed that the overall dataset quality is sufficient without quality improving preprocessing.

In addition to performing corrective measures, it is common to perform feature engineering techniques as a part of the preprocessing of datasets. Feature engineering is the process of improving predictive modelling performance on data set through data mining techniques that modify the data for better fitting in a specific ML method. When applying a ML method to a data set, data samples or data points constitute the basic components. Every sample is described with several features and every feature consists of different types of values [28]. in this case the data samples corresponds to the data coordinates (x,y,z,t), as mentioned in section 3.2.2, together with a the output values in the ODB. Some of the relevant feature engineering techniques includes (i) dimensionality reduction (ii) feature selection and (iii) feature extraction. These techniques extract relevant features from the raw data, and is thereby transforming the feature space. Feature space is defined as a collection of features that characterize the data set [5]. The dimensionality of the feature space is defined as the number of active features, and it has been proved that ML algorithms work better with a lower dimensionality [48]. This phenomenon is called the *curse of dimensionality*, and implies that the more features we have, the more data is needed to train a good model [34].

How feature engineering is performed depends on the chosen machine learning method. Deep learning methods are ML algorithms that uses multiple layers to extract higher-level features from the input data progressively [43]. In practice, this means that when applying a deep learning algorithm on a dataset, it is capable of automatically learn features and performing feature extraction. This is in contrast to the traditional machine learning methods, where features have to be hand crafted and expert made [4]. Roy and Wodo utilize expert made features from a input space consisting of data samples with data coordinates (x,y,z,t) [40] in addition to the nodal temperature, T. With 26000 data points, where each data point corresponds to a nodal temperature profile T(t). With 4 dimensions per data point (x,y,z,t), the dataset has a dimensionality of 104000, and clearly needs dimensionality reduction. When introducing the HIZ, as described in section 3.2.1, only the data points within the HIZ are taken into account in the calculation. This is a feature selection process which reduce the dimensionality. Instead of using the spatial coordinates (x,y,z) of a given data point, new features of relative distances are introduced; the distance from cooling surfaces (d_c^l), the distance from the heat sources (d_s^k), and a set of deposition times (t_s^k). This is a feature extraction process which reduce the dimensionality per data point from 4 to 3 dimensions. In comparison, Mozzafar et al. have a different approach to feature engineering in *A real-time iterative machine learning approach for temperature profile prediction in additive manufacturing processes* [37]. For each data point, Mozzafar et al. extracts the following features:

- Historical Features: Temperature of the given element at t1 through t5 (if applicable)
- Spatio-Temporal Features: Temperature of neighboring 26 voxels at t1
- Spatial Features: relative x, y and z coordinates of the current voxel with respect to the current position of the laser
- Temporal Features: Time of voxel creation and time elapsed since the creation of given voxel

These features means 35 dimensions per data point, which is significantly higher compared with Roy and Wodo. The curse of dimensionality can clearly be seen on the size of the dataset, where Mozzafar et al. needed 9.05 million datapoints for their dataset, whilst Wodo and Roy had a dataset with 26000 datapoints. 26000 corresponds to 0.29% of the datset needed to

3.3.2 Dataset splitting

In order to be able to verify the accuracy of the ML model, one often split the total dataset into two seperate sets; train set and test set. The model is trained on the train set, whilst the test set is retained to verify the model performance on unseen instances. One common heuristic is to withhold 20% of the available data examples for testing, using the remaining 80% for training. This distribution is according to the Pareto principle [34, 11].

3.3.3 Dataset bias

For a learning model to generalize and make predictions on unseen training data, it often makes certain assumptions based on the data. These assumptions are called the model's *inductive bias* [33]. Inductive bias may lead to a generalization issue when training and testing a learning algorithm on data extracted from different conditions [49]. In this case possible dataset bias factors may be materials, deposition patterns or geometry. Mozzafar et al. experienced results in *Data-driven prediction of the high-dimensional thermal history in directed energy deposition processes* *via recurrent neural networks* which might indicate a high inductive bias in their dataset [35]. When the predictions on the geometry in figure 9, which is dissimilar from the geometries in the training dataset, the model made accurate predictions of point 1 and point 3, but made significant error when predicting the state of point 2. Mozzafar et al. explained the error with the geometric feature and the state of the boundaries close to this point, which is not represented in the training data [35].

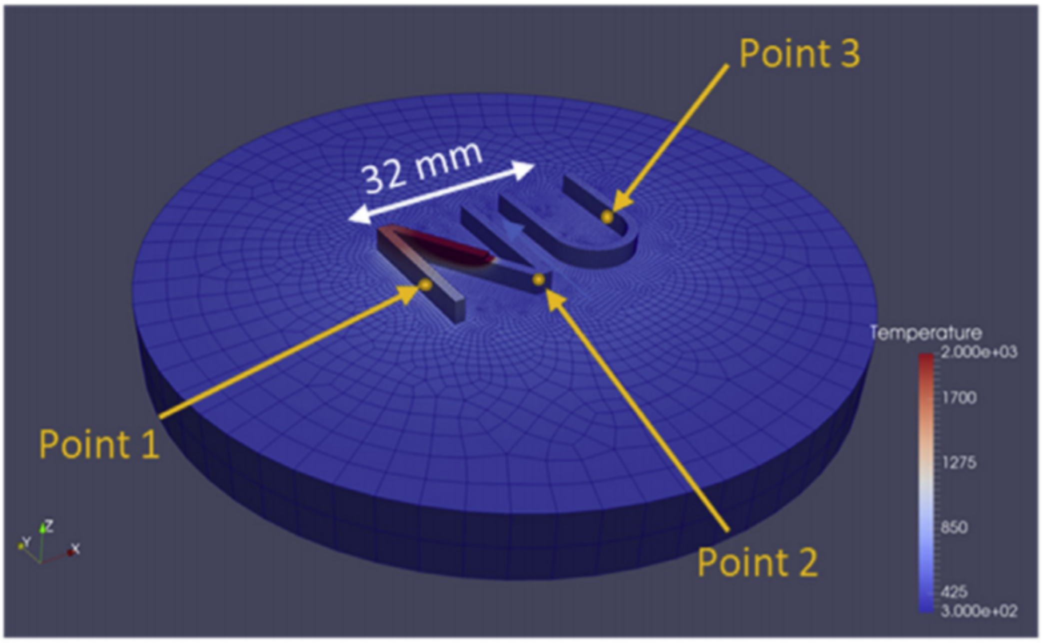


Figure 9: NU-shaped build and inspected points.

Source: [35]

3.4 Training of surrogate model

Choosing a model that fits the data set is vital in order to achieve high performance, and the intention of this section is to give a brief overview of the necessary considerations when developing a machine learning model. The most widely used machine learning algorithms of today are different kinds of neural networks, also called Artificial Neural Networks. A neural network is a computational learning system that utilizes a network of functions that are built to understand and translate input data to the desired output data [34]. In machine learning, these functions are usually called *activation functions*, and are often represented as a *neuron*. The most commonly used activation function is the *rectified linear unit* (ReLU) [38], as seen in equation 2.

$$f(x) = \begin{cases} x & \text{if x} \geq 0 \\ 0 & \text{if x} < 0 \end{cases} \tag{2}$$

3.4.1 Anatomy of neural networks

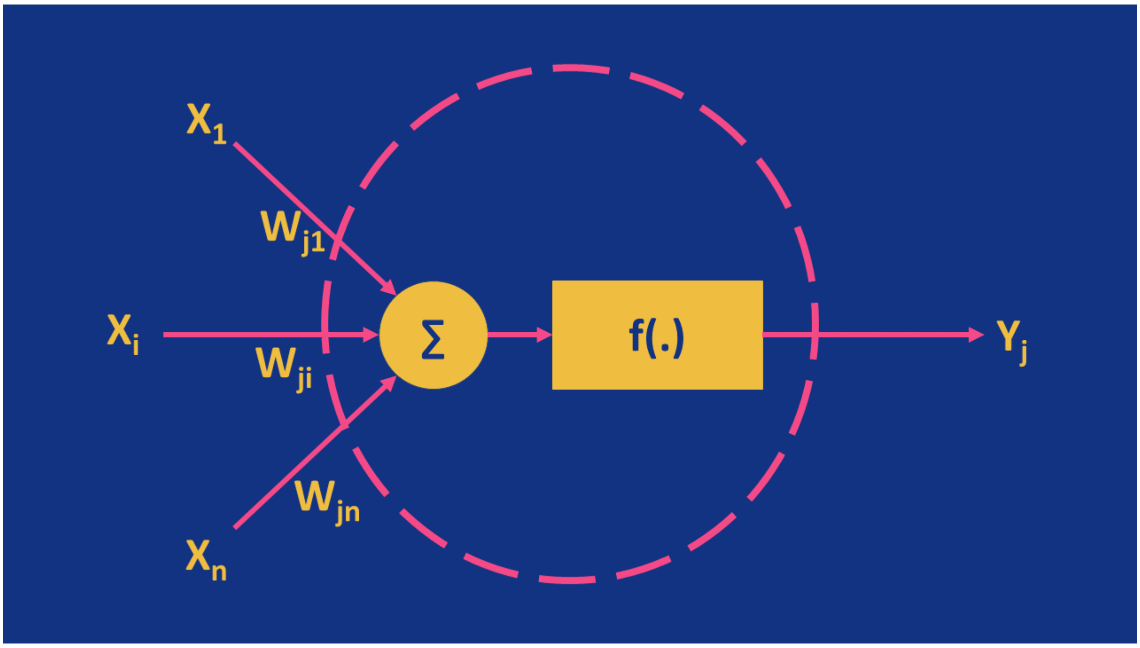


Figure 10: Detail of a neuron.

The connections between the neurons (illustrated by arrows in figure 11) has a *weight* which is assigned based on its relative importance [51]. This can be seen in figure 10, where the input signals x_i of the neuron sums up before it is weighted according the strength of its respective connections w_{ij}. With its activation function, f, the output Y_j is computed according to equation 3 [51].

$$Y_{ij} = f\left(\Sigma w_{ij} x_{ij}\right) \tag{3}$$

The neurons are often placed sequentially in *layers*, as seen in figure 11. Layer 1 is the *input layer*, and layer 4 is the *output layer*. Layer 2 and 3 are *hidden layers*, which distribute and modify the signals without any connection to the environment. Each subsequent layer modifies the data in hope of increasing the accuracy of the predictions by maximizing or minimizing its *objective function* to get an optimized output [2, 19]. In neural networks, the usual objective is to minimize the error, which is represented by a *loss function*. If a model is not able to obtain a sufficiently low error on the training set, it is said to be *underfitting*, meaning it is not able to represent the data [19].

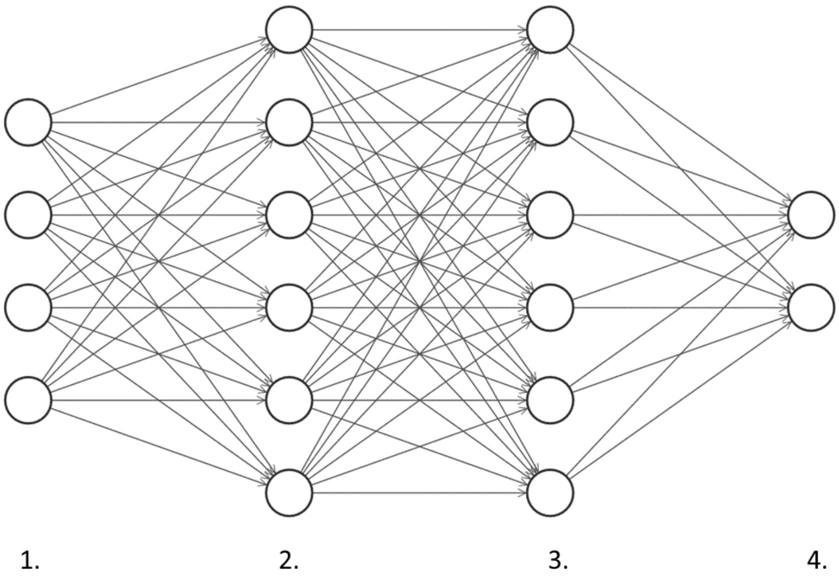


Figure 11: Example of neurons in a neural network

If the network only contains the input and output layer, without containing any hidden layers, the network is called a *single-layer perceptron*(SLP). However, if the network contains one or more hidden layers, the network is a *multi-layer perceptron*(MLP). If a MLP has 3 or more layers, it is defined as a deep neural network, whilst if the network has 1 or 2 hidden layers the network is shallow [51]. Deep neural networks exhibits state-of-the-art performance in many fields where big datasets are available due to its efficient and accurate predictions. Deep MLPs are therefore more widely implemented than SLPs and shallow MLPs [12]. However, even though deep neural networks have promising results, deeper networks are not always better due to well known problems such as *overfitting*, *the vanishing gradient problem*, and *the degradation problem*.

Overfitting:
Overfitting refers to the phenomenon when the model fits the training data too closely, and is thus not able to generalize to unseen samples. When this happens, the model includes the noise in the dataset in its prediction, and is often caused by a small dataset [3, 10]. The risk of overfitting is higher in deep neural networks compared to shallow neural networks, which is why shallow networks perform better compared to deep networks when the dataset is small. This is probably the reason

why Roy and Wodo, which has a small dataset, constructed a shallow neural network, whilst Francis et al. and Mozzafar et al. constructed deep neural networks.

There are two well known techniques to reduce the effect of overfitting; *dropout* and *transfer learning*. Dropout is a method where random neurons are deactivated to reduce overfitting due to noise in the training data [45]. In addition to performing dropout, transfer learning can be utilized to improve the model's generalization. In transfer learning, generalized data from similar, pre-trained models are utilized to improve performance. By using generalized weights from models with good performance, the network has a better starting point than if it was initialized with random weights [54].

The vanishing gradient problem:

The vanishing gradient problem may be encountered when training artificial neural networks that updates their weights based on the partial derivative of the error function of the model. In this case, the error function is the calculated prediction error of each layer output, and if the gradient becomes too small, training becomes hard [25]. This problem might be avoided if the right activation function is chosen, such as ReLU [26].

The degradation problem:

The degradation problem is the observation that deepening a neural network increases the accuracy until it converges towards a limit value, and then starts to decrease [24]. Until *residual networks* (ResNet) were introduced by He et al. in 2015, this problem limited the possible depth of neural networks [24]. The authors presented a new variation of neural networks that consisted of several residual blocks, as seen in figure 12. Each residual block was represented by the function $H(x) = F(x) + x$, where $H(x)$ is the desired underlying mapping. $F(x)$ is the residual mapping and is the output of the stacked layers. x is the identity mapping and is added to $F(x)$ to form $H(x)$ through shortcut connections that skip some layers. These residual blocks ensured that deeper counterparts to shallower networks are able to achieve at least the same accuracy as their shallower counterparts, and thus avoids the degradation problem [24]. The same paper by He et al. presented results from image classification with a ResNet of 1001 layers, which was groundbreaking when published.

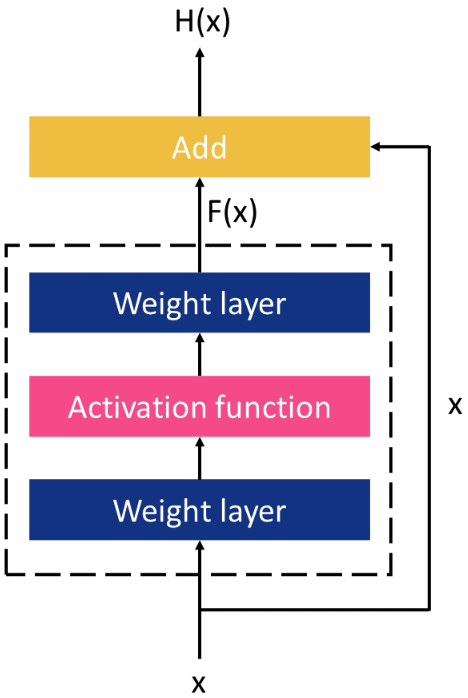


Figure 12: A residual block

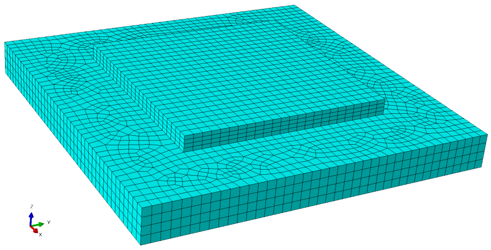
3.5 Validation and testing

The central task in machine learning is to construct a model that perform well on new, unseen data. This is different from regular optimization problems as it does not only optimize based on the data that was used to construct the model, but also on other representative data. The ability to perform well on previously unobserved inputs is called generalization, and is usually measured by testing the performance on a test set [19], as defined in section 3.3.

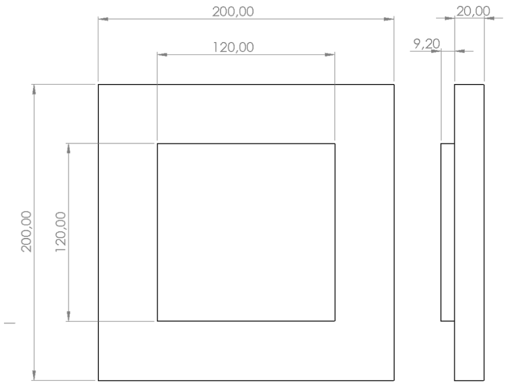
One way of measuring the performance of a regression model is to compute the loss function. As mentioned in 3.4.1, the objective of neural networks is to minimize this loss function. One way of defining the loss function is through the mean squared error (MSE) of the model. A low MSE in the training set and high for the test set indicates overfitting. If the model is overfitting, it is not able to handle unseen data, and the performance is low. The mean squared error equation can be seen in equation 4, where $\bar{y}_{test}$ defines the predictions of the model on the test set and y_{test} denotes the real value. As expected, when $\bar{y}_{test} = y_{test}$, the error is 0.

$$MSE_{test} = \frac{1}{m} \Sigma \left(\bar{y}_{test} - y_{test}\right)_i^2 \tag{4}$$

4 Results



(a) CAD with mesh.



(b) Dimensions of model in mm.

Figure 13: Geometry of model.

In this project an Abaqus model was built to analyze the temperature during additive manufacturing of a model with 4 layers. The model was created in Abaqus 2019 with the AM Modeler plugin to simulate the additive manufacturing process. The mesh of the model can be seen in figure 13a and its dimensions can be seen in figure 13b. The CAD's was built using the classes and methods in the python scripts in appendix A.1 to A.12. The scripts defining the models are in appendix A.13 for the zigzag pattern and appendix A.14 for the raster pattern.

The model consists of a substrate and the added layers in the AM process. The substrate is rigidly clamped and rectangular sized proportional to the added layers in order to simplify the coding of the python methods. In addition, in order to avoid convergence problems, the two features were modelled in the same part, and thus avoiding contact forces and boundary conditions. The external load of the simulation is a heat distribution from a moving heat source. The moving heat source simulates the deposition of material by melting, and is simulated by progressive activation of elements in the model. Additional analysis details can be seen in table 1. In the following section the thermal history of analysis with zigzag and raster deposition patterns (as seen in figure 6a and 6b in section 3.2.1) will be presented and discussed. The thermal profiles will be extracted from a corner node and a mid node with positions as marked with red dots in figure 14.

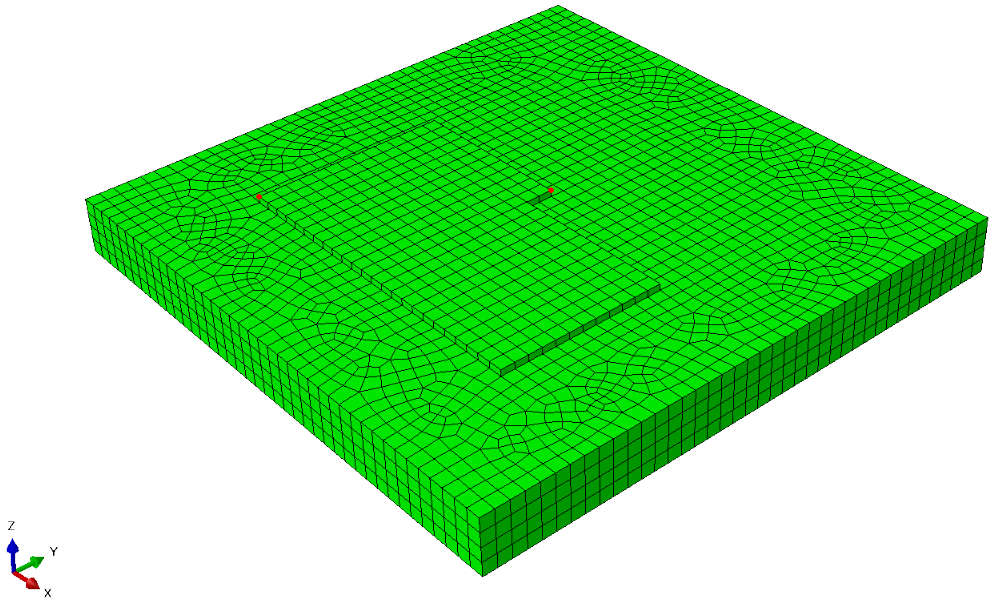


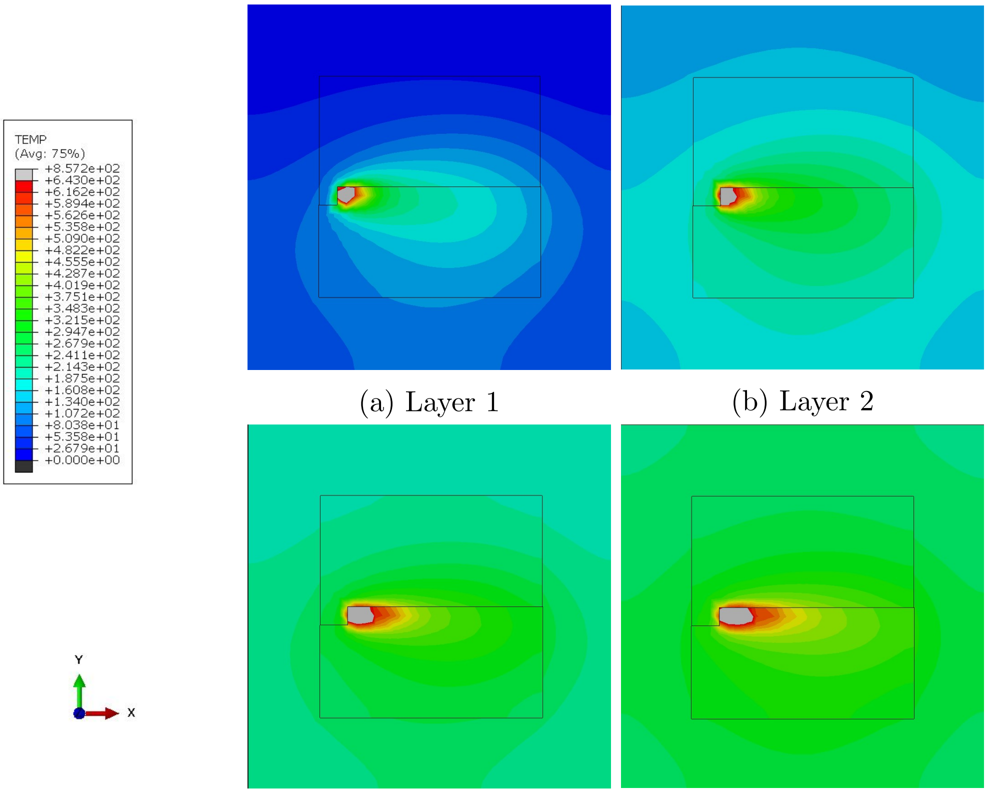
Figure 14: Position of corner and mid node

Experiment details		
Material properties	Material name	AA2319 (See appendix B for details)
Process parameters	Layer thickness	$0.0023m.$
	Nr. of layers	4
	Road width	$0.01m.$
	Ambient temperature (T_{am})	20°C
	Print speed (v)	$0.015m/s$
Mesh details	Element size	$0.005m.$
	Element type	8-noded linear heat transfer bricks (DC3D8)
	Nr. of elements	9600
	Integration	Full
Deposition details	Activation offset	$0.005m.$
	Activation set size	$(0.005\ \text{x}0.01\text{x}0.0023)m.^3$
	Heat magnitude (Q)	5000 W.
	Heat model	Double-ellipsoid heat model (Goldak)
	Length of the front ellipsoid (a_f)	0.04mm.
	Length of the rear ellipsoid (a_r)	0.06mm.
	Fraction factor of the heat flux in the front part (f_f)	0.6
	Fraction factor of the heat flux in the rear parts (f_r)	1.4
	Deposition pattern	ZigZag
		Raster

Table 1: Experimental details

4.1 Contour plots of transient temperature fields

The contour plots of the transient temperature field for both the raster and zigzag deposition process can be seen in figure 15 and 16, where the legend is ranging from the ambient temperature of 20°C to the melting temperature of 643°C. The grey are is above the melting temperature and indicates the melting pool of the process.



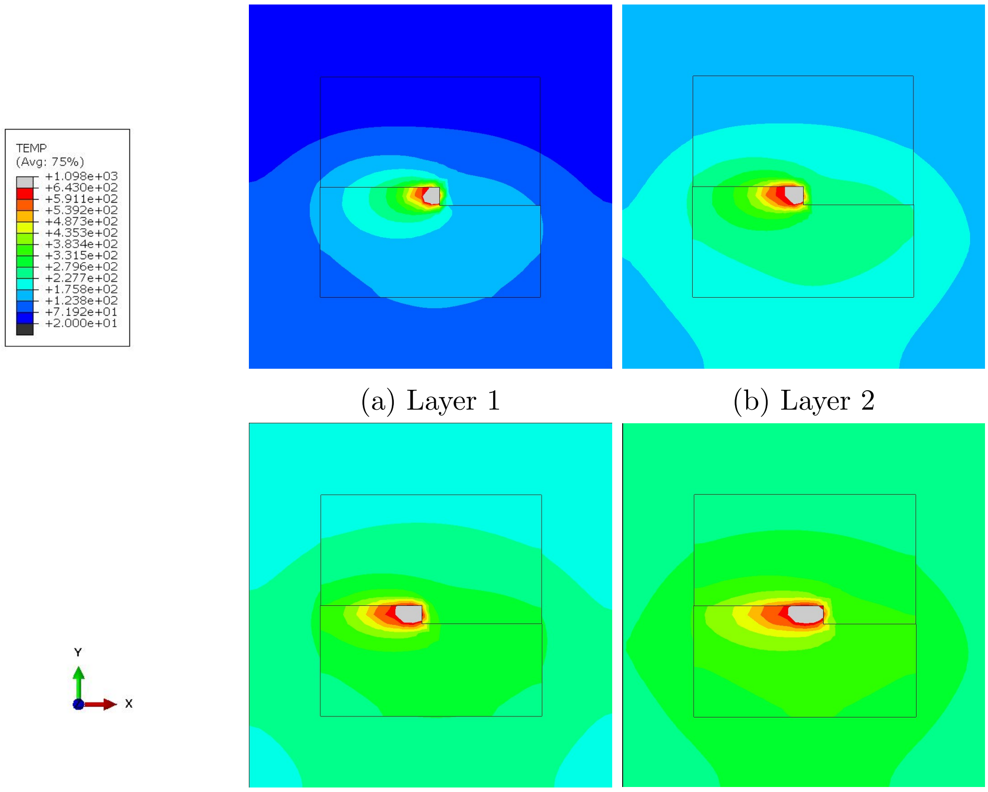
(a) Layer 1

(b) Layer 2

(c) Layer 3

(d) Layer 4

Figure 15: Contour plots of the transient temperature field for each deposited layer with zigzag deposition pattern.



(a) Layer 1 (b) Layer 2

(c) Layer 3 (d) Layer 4

Figure 16: Contour plots of the transient temperature field for each deposited layer with raster deposition pattern.

The contour plots for raster and zigzag are not significantly different, and shows the same tendency; the size of the melting pool grows and the global temperature is increasing. This is expected due to the continuous heat input of 5000W and conduction within the part. Conduction is the transfer of energy from the more energetic particles of a substance to the adjacent less energetic ones as a result of interactions between the particles. Conduction can take place in solids, liquids, or gases, and in this case it happens in the particles of the model [56]. In addition, the melt pool is modelled with the double-ellipsoidal heat power density model, where the melt pool size is proportional to the heat source input (Q), as seen in equation 1. Both of the models reach a global temperature above 200°C after depositing the last layer, which is a higher temperature than expected. This may be explained by the lack of cooling breaks between each deposited layer, which is normal to include in physical experiments. The script is customized to support the addition of cooling time, but was not included as it was mentioned by Roy and Wodo as a complicating factor when implementing a machine learning algorithm, and at the same time not having a significant impact on the results. In addition, in physical experiments, one would change the power input when the part is heating up. This would mean a higher heat input in the first increments compared to the last increments, which in turn would mean less global heating.

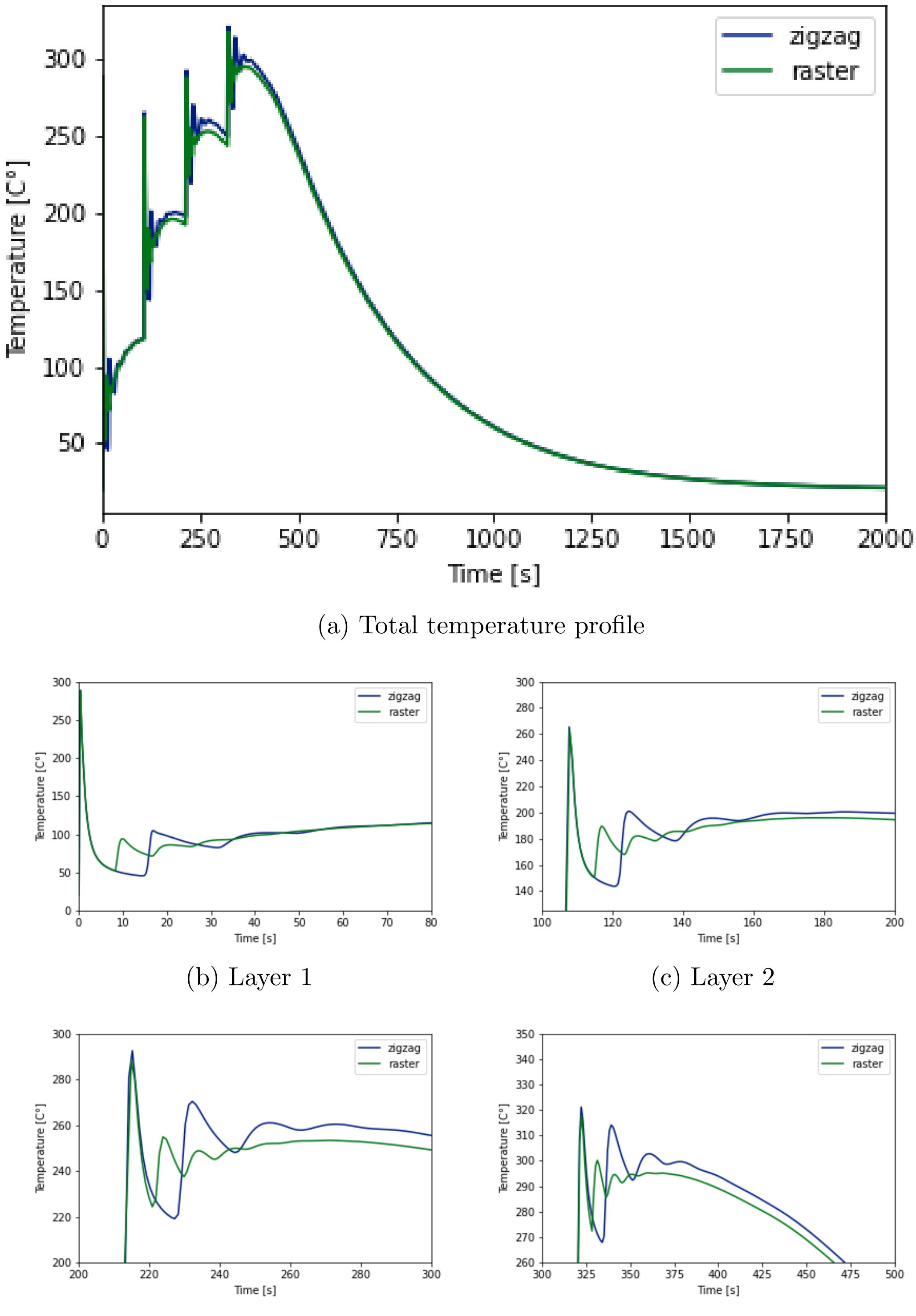
4.2 Comparison of thermal profiles of raster and zigzag

In order to get an understanding of how the temperature in the AM part is influenced by the moving heat source, the transient thermal profiles of a node has been studied. The plots in figure 17 displays 5 graphs with the temperature profile over time for a corner node with position as seen in figure **??**. At $t = 0$, the node is activated by the moving heat source, which in this case also corresponds to the start of the simulation as it is the first element to be activated.

Figure 17a shows the full thermal profile from initialization until the node has cooled down to the ambient temperature of $20°C$. The graph shows a cyclic reheating pattern with similar shape for both the raster and zigzag deposition pattern. For both deposition patterns, the node experience 4 cycles of heating before cooling down to the ambient temperature. Each heating cycle corresponds to one deposited layer. The node experience the reheating as the absolute distance to the heat source is at its minimal when it start the deposition of the layer. This can be seen in the detailed plots in figure 17b to figure 17e, where each reheating cycle is initialized with an abrupt thermal increase, and supports the choice of Roy and Wodo and Mozzafar et al. to include features related to the distance from the heat sources [40, 37]. The absolute value of the initial heat increase in each cycle is decreasing for each layer, which also indicates a direct relation between the nodal temperature and the distance to the heat source. However, even though the absolute value of the initial heat increase sees a dampening effect for each deposited layer, the nodal temperature in increasing as long as the heat source is active. This confirms the global heating seen in figure 16 and figure 15.

When investigating the detail plots in figure 17b to 17e it is apparent that each layer experience thermal oscillations with a dampening effect. Comparing the graphs of zigzag and raster, it can be seen that the zigzag graph has a significantly higher amplitude and lower frequency of oscillation. The differences is an expected result as the transient nodal temperature is dependent on the transient position of the heat source, which is determined by the deposition pattern. As all passes in the raster start from the same side, the period for each oscillation corresponds to the time it takes for the heat source to travel the length of the add element. In comparison, the zigzag moves back and forth, which makes each period for each oscillation corresponding to the time it takes for the heat source to travel the length of the add element twice. In addition, as the heat source travels back and forth with the zigzag pattern, the amount of time the heat source spends is in close proximity or at a distance with the node is higher. This leads to the increased amplitudes of the oscillations.

In all the detailed plots it can be seen that the dampening on the oscillations decreases after 3 oscillations for the zigzag (corresponding to 6 passes over the add element) and 4 oscillations for raster (corresponding to 4 passes over the add element). As the amplitude of the thermal graph for zigzag is bigger than for raster, it makes sense that the passes influence nodes further away compared to with raster. In practice, this means that the heat influence zone mentioned by Roy and Wodo. [40] is bigger with zigzag deposition pattern compared to with raster deposition pattern.



(a) Total temperature profile

(b) Layer 1

(c) Layer 2

(d) Layer 3

(e) Layer 4

Figure 17: Temperature profiles for corner node, deposited with raster and zigzag.

4.3 Comparison of thermal profiles for mid node and corner node

To investigate the impact of nodal position in part on the nodal temperature profiles, data for a mid node and a corner node has been investigated. The position of the nodes can be seen in figure 14 and the deposition pattern used was raster. The full temperature profile can be seen in figure 19 and a detailed view can be seen in figure 19b. It can be seen that the mid node is activated later than the corner node as there are no temperature values for the mid node graph before 44.1021 s. However, when looking at the time-normalized graphs in figure 20, where the time indicates time from activation, it is clear that the the nodes follow a similar pattern with cyclic reheating and oscillations for each deposited layer.

The clear tendency of the two graphs are that the temperature of the corner node is lower than that of the mid node. This makes sense when taking a look at the type of heat transfer occurring in the volume around the node. One might imagine a infinitesimal cubic volume around the nodes, as seen in figure 18, where the red sphere represents the node. The heat transfer occurs through the sides of the cube through conduction and convection. Convection is the mode of energy transfer between a solid surface and the adjacent liquid or gas that is in motion [56]. In this case the convection happens between the material in the model and the surrounding air, also called the thermal radiation. In the cube surrounding the corner node, two of the sides are free surfaces in contact with air of a significant lower temperature, which will lead to cooling and heat loss of the material. When it comes to the mid node, the cube does not have any free surfaces, and will not experience this cooling effect. This supports the choice of Roy and Wodo when including features related to the distance from the cooling surfaces; considering the normal distances from the given point and the six free surfaces [40].

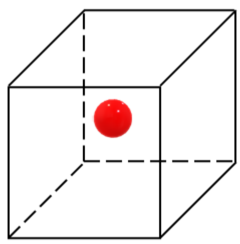
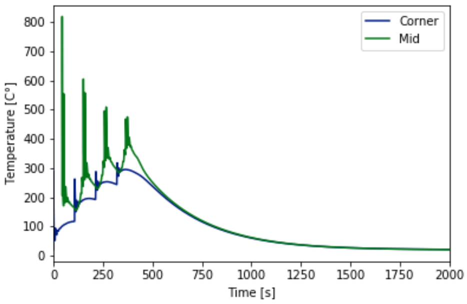
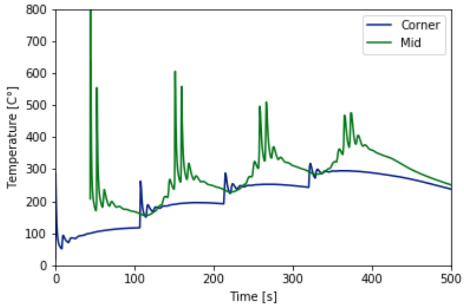


Figure 18: Node with infinitesimal cubic volume.

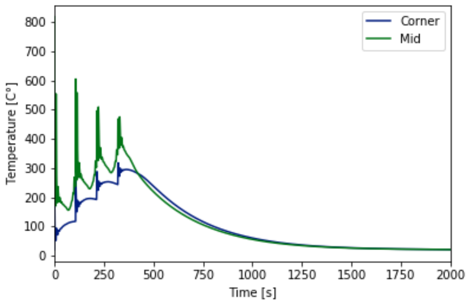


(a) Total temperature profile of corner and mid node, deposited with raster.

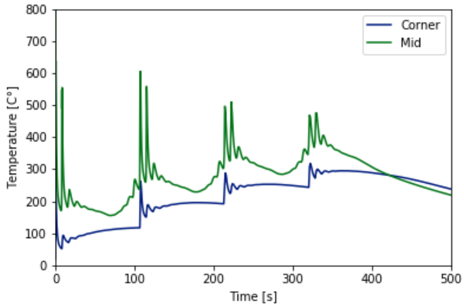


(b) Detailed view of reheating cycles

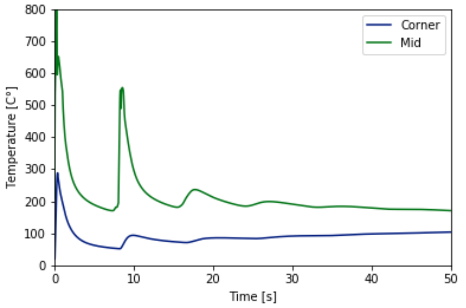
Figure 19: Detailed view of time temperature profiles for corner and mid node, deposited with raster.



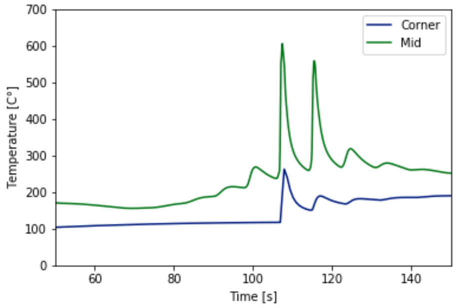
(a) Total temperature profile



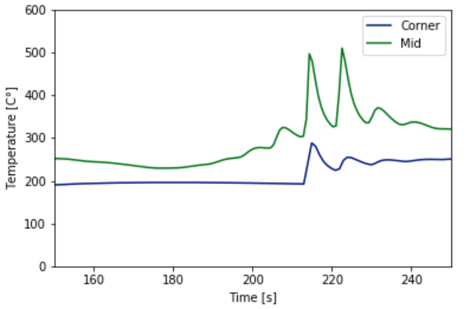
(b) Detailed view of reheating cycles



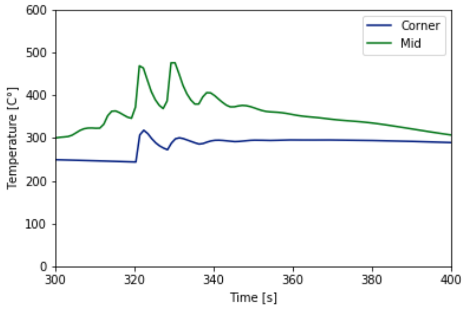
(c) Layer 1



(d) Layer 2



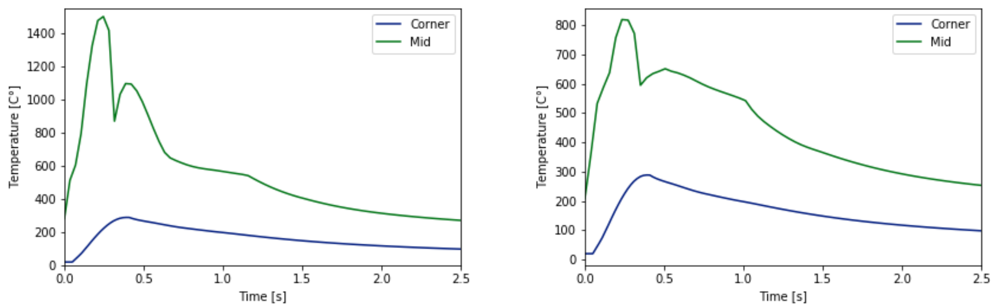
(e) Layer 3



(f) Layer 4

Figure 20: Time normalized temperature profiles for corner and mid node, deposited with raster.

4.4 Activation of elements



(a) Zigzag (b) Raster

Figure 21: Detailed view of time temperature profiles for element initialization.

When taking a close look at figure 20c and figure 17b, a cusp close to $t = 0$ is visible. When zooming in on a small interval just after initialization, as seen in figure 21, several cusps are visible on the mid node graphs. It was suspected that the cusps could be related to the activation of neighboring elements as there were no cusps in the corner node graphs, where no elements connected with the node is activated after the initialization of the corner node. To investigate the hypothesis further, the thermal profiles of the mid node on the top of the layer was compared to the mid node on the base of the layer as indicated with red dots in figure 22.

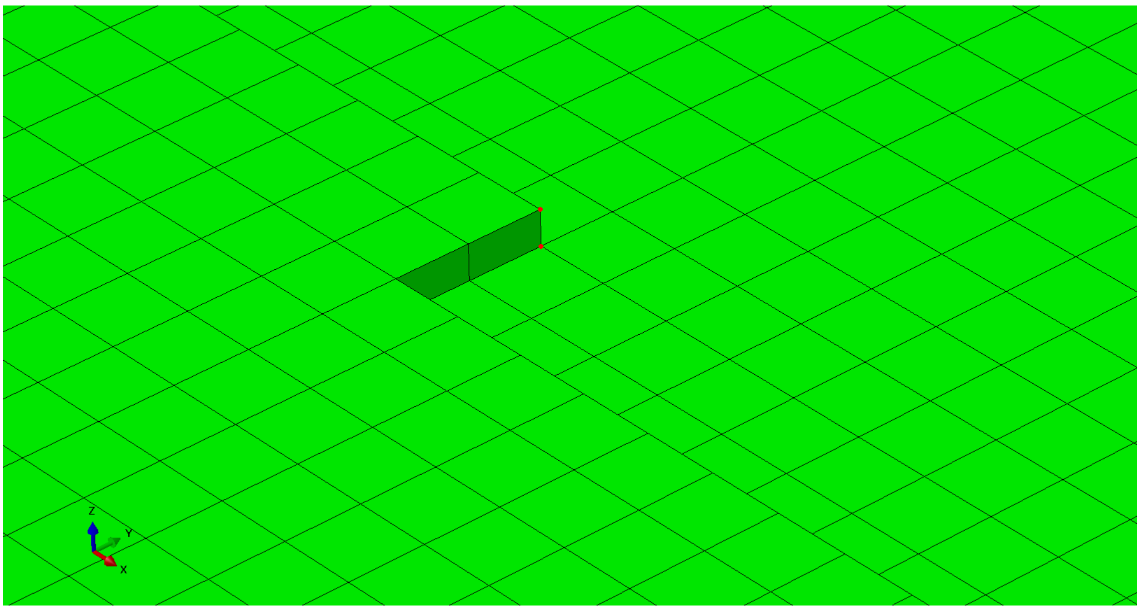
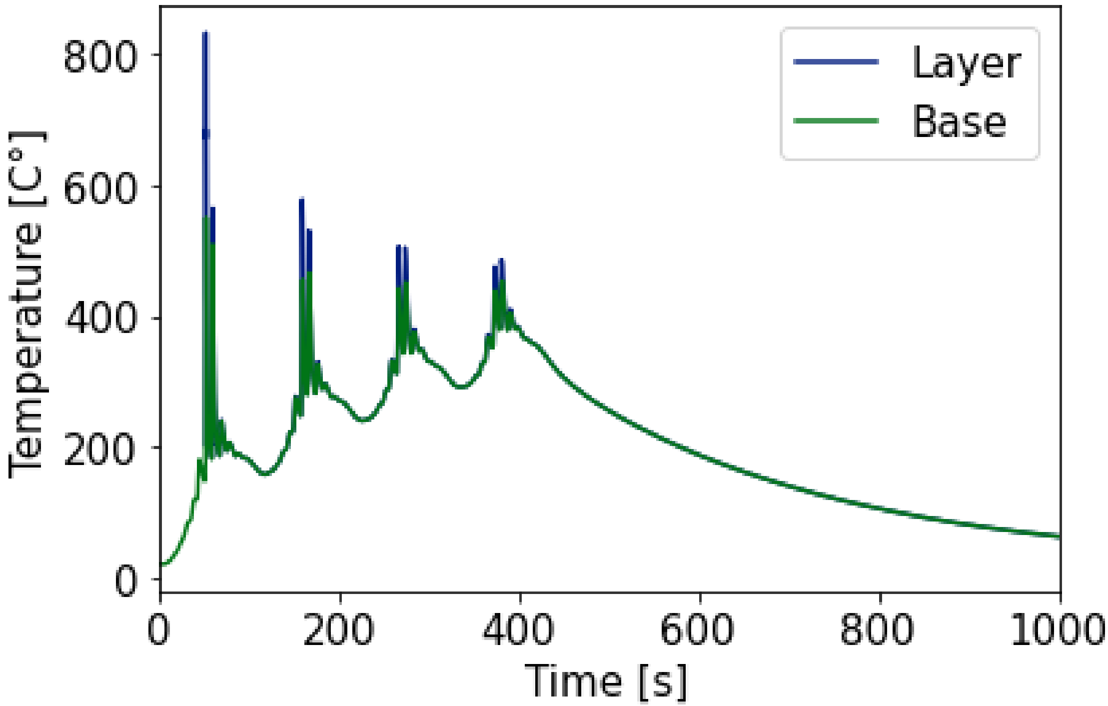
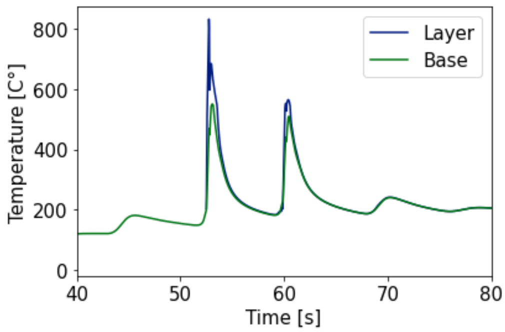


Figure 22: Position of base and layer nodes.

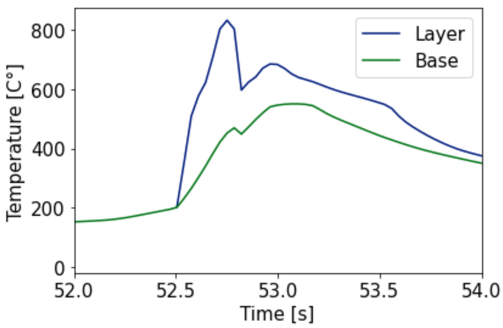
The full thermal profile can be seen in figure 23a and the details can be seen in figure 23b and figure 23c. As the base node is a part of the substrate, it is initialized together with the substrate, when the analysis begins. The layer node is initialized when layer 1 is deposited, 52.5051 s. after the analysis begins. It can be seen in figure 23c that the thermal profile of the base node has a cusp when the layer node is initialized at 52.5051 s. In addition, it is apparent that the two graphs has two clear cusps at 52.8215 s. When checking the time of initialization of the subsequently deposited element, it was seen that the position of the cusps was corresponding to the activation of the subsequent elements. This is relation is clear in figure 24 where the time of initialization of the subsequent element is marked with a red line which exactly corresponds to the two cusps.



(a) Full thermal profile.



(b) Layer 1



(c) Detailed view of layer 1

Figure 23: Thermal profiles of base and layer node deposited with zigzag.

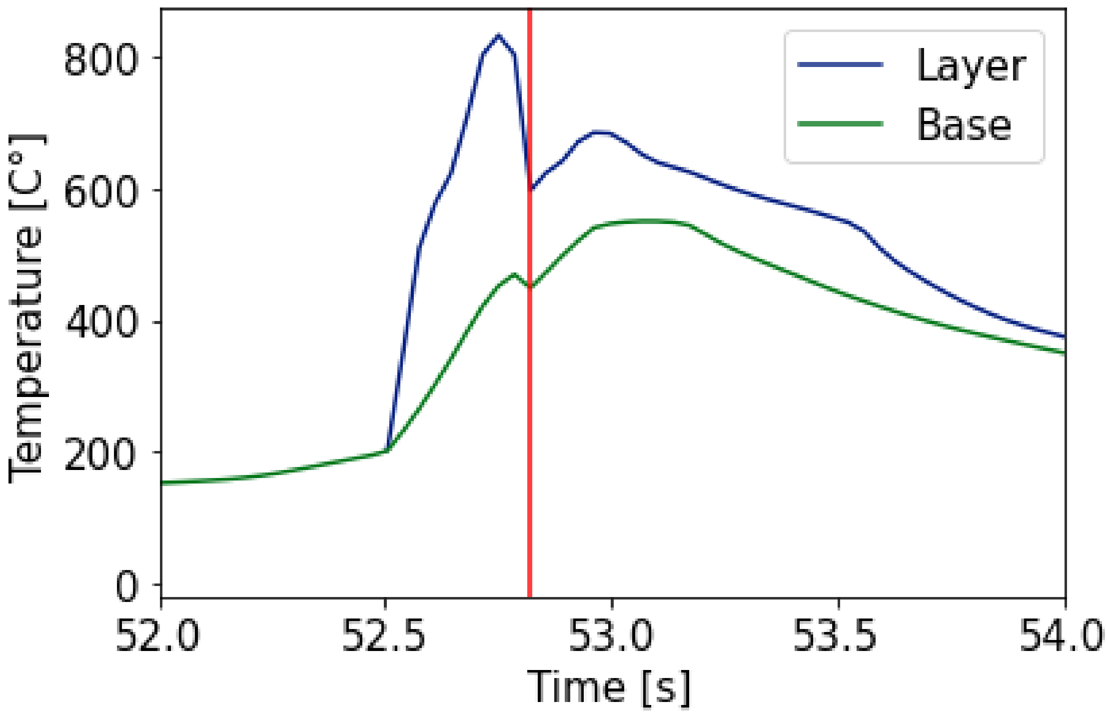


Figure 24: Thermal profiles of base and layer node close to initialization of the layer node. The initialization is marked with the red line.

5 Conclusions and Further Work

This project has introduced the steps and theory necessary to implement machine learning to create a surrogate model of transient thermal fields in additive manufacturing. The main intention has been to highlight that machine learning is ideal in additive manufacturing as well as introducing concepts necessary to fully grasp the potential and challenges of the problem.

To summarize, it is possible to create high volume data sets through simulations of physics-based models. The resulting data set can be used to train and test a machine learning model that may predict the transient thermal fields in additive manufacturing. Neural networks have proved to have good performance, and is the most natural choice when it comes to machine learning method.

In addition, results from simulations of physics-based models have been presented. These confirm that the expert engineered features proposed by Roy and Wodo, distance to heat source and free surface, is important to consider when predicting the thermal profiles in additive manufacturing.

As this project acts as preparations for my master thesis, the further work will be to construct a surrogate model as described. In addition, I have identified some topics that I wish to give extra attention; deep neural networks for automatic feature extraction and expansion of the database to include irregular geometries. The creation of a deep neural network that are able to perform automatic feature extraction has designated as a point of improvement in similar projects. Similar projects have presented complex and time consuming feature engineering techniques. By avoiding manual feature engineering, the process will not only be simplified, but it can also be assumed that the model will be able to represent complex and hidden process characteristics. The inclusion of irregular geometries in the data set will be necessary in order to make the model applicable in reality, as the additive manufacturing process usually include complex geometries, not only rectangular hexahedrons. This was also mentioned by Mozzafar et al. in *Data-driven prediction of the high-dimensional thermal history in directed energy deposition processes via recurrent neural networks.*

Bibliography

[1] Abaqus 6.14, scripting user guide.

[2] Saad Albawi, Tareq Abed Mohammed, and Saad ALZAWI. Understanding of a convolutional neural network. 08 2017.

[3] Haider Allamy. Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study). 12 2014.

[4] Syed Anwar, Muhammad Majid, Adnan Qayyum, Muhammad Awais, Majdi Alnowami, and Khurram Khan. Medical image analysis using convolutional neural networks: A review. *Journal of Medical Systems*, 42:226, 10 2018.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[6] W.D. Callister and D.G. Rethwisch. *Materials Science and Engineering: An Introduction, 8th Edition*. Wiley, 2009.

[7] Mashrur Chowdhury, Amy Apon, and Kakan Dey. *Data Analyticsfor Intelligent Transportation Systems*. Elsevier, 2017.

[8] Diego de Moraes and Aleksander Czekanski. Thermal modeling of 304l stainless steel for selective laser melting: Laser power input evaluation. 11 2017.

[9] T. DebRoy, H.L. Wei, J.S. Zuback, T. Mukherjee, J.W. Elmer, J.O. Milewski, A.M. Beese, A. Wilson-Heid, A. De, and W. Zhang. Additive manufacturing of metallic components – process, structure and properties. *Progress in Materials Science*, 92:112 – 224, 2018.

[10] Tom Dietterich. Overfitting and undercomputing in machine learning. *Computing Surveys*, 27:326–327, 1995.

[11] Rosie Dunford, Quanrong Su, and Ekraj Tamang. The pareto principle. 2014.

[12] Shuo Feng, Huiyu Zhou, and Hongbiao Dong. Using deep neural network with small dataset to predict material defects. *Materials Design*, 162:300 – 310, 2019.

[13] Jack Francis and Linkan Bian. Deep learning for distortion prediction in laser-based additive manufacturing using big data. *Manufacturing Letters*, 20:10 – 14, 2019.

[14] William Frazier. Metal additive manufacturing: A review. *Journal of Materials Engineering and Performance*, 23, 06 2014.

[15] Guangming Fu, Jijun Gu, Marcelo Igor Lourenco, Menglan Duan, and Segen F. Estefen. Parameter determination of double-ellipsoidal heat source model and its application in the multi-pass welding process. *Ships and Offshore Structures*, 10(2):204–217, 2015.

[16] M. Gao, Zemin Wang, Xiangyou Li, and Xiaoyan Zeng. The effect of deposition patterns on the deformation of substrates during direct laser fabrication. *Journal of Engineering Materials and Technology*, 135:034502, 07 2013.

[17] Randall M. German. Chapter nine - sintering with a liquid phase. In Randall M. German, editor, *Sintering: from Empirical Observations to Scientific Principles*, pages 247 – 303. Butterworth-Heinemann, Boston, 2014.

[18] J. Goldak, M. Bibby, and A. Chakravarti. *A Double Ellipsoid Finite Element Model for Welding Heat Sources*. International Institute of Welding, 1985.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[20] R. Goodridge and S. Ziegelmeier. 7 - powder bed fusion of polymers. In Milan Brandt, editor, *Laser Additive Manufacturing*, Woodhead Publishing Series in Electronic and Optical Materials, pages 181 – 204. Woodhead Publishing, 2017.

[21] Michael Gouge and Pan Michaleris. *Thermo-Mechanical Modeling of Additive Manufacturing*. Elsevier Science Technology, Saint Louis, 2017.

[22] Mikell P. Groover. *Groover's Principles of Modern Manufacturing*. Wiley, 2016.

[23] Phase Transformations Complex Properties Research Group. Lecture 1, aluminum and alloys.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[25] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6:107–116, 04 1998.

[26] Hidenori Ide and Takio Kurita. Improvement of learning for cnn with relu activation by sparse regularization. pages 2684–2691, 05 2017.

[27] Ping Jiang, Qi Zhou, and Xinyu Shao. *Surrogate Model-Based Engineering Design and Optimization.* 01 2020.

[28] Konstantina Kourou, Themis P. Exarchos, Konstantinos P. Exarchos, Michalis V. Karamouzis, and Dimitrios I. Fotiadis. Machine learning applications in cancer prognosis and prediction. *Computational and Structural Biotechnology Journal*, 13:8 – 17, 2015.

[29] Nilesh Kumar, Rajiv S. Mishra, and John A. Baumann. Chapter 1 - introduction. In Nilesh Kumar, Rajiv S. Mishra, and John A. Baumann, editors, *Residual Stresses in Friction Stir Welding*, pages 1 – 4. Butterworth-Heinemann, Boston, 2014.

[30] A. Lundbäck. Finite element modelling and simulation of welding of aerospace components. 2003.

[31] Somashekara M Adinarayanappa, M Naveeen Kumar, Avinash Kumar, Viswanath Chinthapenta, and Suryakumar Simhambhatla. Investigations into effect of weld-deposition pattern on residual stress evolution for metallic additive manufacturing. *International Journal of Advanced Manufacturing Technology*, 90, 05 2017.

[32] P. Michaleris, Z. Feng, and G. Campbell. Evaluation of 2d and 3d fea models for predicting residual stress and distortion. 347, 1997.

[33] Tom Mitchell. The need for biases in learning generalizations. 10 2002.

[34] Tom M. Mitchell. *Machine Learning.* McGraw-Hill Education, 1997.

[35] Mojtaba Mozaffar, Arindam Paul, Reda Al-Bahrani, Sarah Wolff, Alok Choudhary, Ankit Agrawal, Kornel Ehmann, and Jian Cao. Data-driven prediction of the high-dimensional thermal history in directed energy deposition processes via recurrent neural networks. *Manufacturing Letters*, 18:35–39, October 2018.

[36] M.Talabis, R. McPherson, I. Miyamoto, J. Martin, and D. Kaye. *Information Security Analytics.* Elsevier, 2014.

[37] Arindam Paul, Mojtaba Mozaffar, Zijiang Yang, Wei Keng Liao, Alok Choudhary, Jian Cao, and Ankit Agrawal. A real-time iterative machine learning approach for temperature profile prediction in additive manufacturing processes. pages 541–550, October 2019.

[38] Francisco Câmara Pereira and Stanislav S. Borysov. Chapter 2 - machine learning fundamentals. In Constantinos Antoniou, Loukas Dimitriou, and Francisco

Pereira, editors, *Mobility Patterns, Big Data and Transport Analytics*, pages 9 – 29. Elsevier, 2019.

[39] Joseph Robinson, Ian Ashton, Eric Jones, Peter Fox, and Chris Sutcliffe. The effect of hatch angle rotation on parts manufactured using selective laser melting. *Rapid Prototyping Journal*, 25, 10 2018.

[40] Mriganka Roy and Olga Wodo. Data-driven modeling of thermal history in additive manufacturing. 01 2020.

[41] A.L. Samuel. Some studies in machine learning using the game of checker. (English). *IBM J. Res. Develop.*, 3(3):210–229, 1959.

[42] Swee Leong Sing, C.F. Tey, J.H.K. Tan, Huang Sheng, and Wai Yee Yeong. *3D printing of metals in rapid prototyping of biomaterials: Techniques in additive manufacturing*, pages 17–40. Woodhead Publishing Series in Biomaterials. Woodhead Publishing, second edition edition, 01 2020.

[43] R. Sinha, R. Pandey, and R. Pattnaik. Deep learning for computer vision tasks: A review. *ArXiv*, abs/1804.03928, 2018.

[44] Emrecan Soylemez. Modelling the melt pool of the laser sintered ti6al4v layers with goldak's double-ellipsoidal heat source. 08 2018.

[45] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[46] Li Sun, Xiaobo Ren, Jianying He, Jim Stian Olsen, Sakari Pallaspuro, and Zhiliang Zhang. A new method to estimate the residual stresses in additive manufacturing characterized by point heat source. *The International Journal of Advanced Manufacturing Technology*, 105, 12 2019.

[47] Li Sun, Xiaobo Ren, Jianying He, and Zhiliang Zhang. Numerical investigation of a novel pattern for reducing residual stress in metal additive manufacturing. *Journal of Materials Science Technology*, 2020.

[48] M. Steinbach T. Pang-Ning and V. Kumar. *Introduction to data mining*, 2006.

[49] Tatiana Tommasi, Novi Patricia, Barbara Caputo, and Tinne Tuytelaars. *A Deeper Look at Dataset Bias*, pages 37–55. Springer International Publishing, Cham, 2017.

[50] Philip Woodall, Alexander Borek, Jing Gao, Martin Oberhofer, and Andy Koronios. An investigation of how data quality is affected by dataset size in the context of big data analytics. 08 2014.

[51] Liu Xing and Duc T Pham. *Neural networks for identification, prediction, and control*. Springer-Verlag, 1995.

[52] Rahime Yilmaz, Anil Sezgin, Sefer Kurnaz, and Yunus Ziya Arslan. *Object-Oriented Programming in Computer Science*, pages 7470–7480. 01 2018.

[53] Yanyan Zhu, Xiangjun Tian, Jia Li, and Huaming Wang. The anisotropy of laser melting deposition additive manufacturing ti–6.5al–3.5mo–1.5zr–0.3si titanium alloy. *Materials Design*, 67:538–542, 02 2015.

[54] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, pages 1–34, 2020.

[55] O.C. Zienkiewicz, R.L. Taylor, and J.Z. Zhu. 1 - the standard discrete system and origins of the finite element method. In O.C. Zienkiewicz, R.L. Taylor, and J.Z. Zhu, editors, *The Finite Element Method Set (Sixth Edition)*, pages 1 – 18. Butterworth-Heinemann, Oxford, sixth edition edition, 2005.

[56] Yunus A Çengel. Heat and mass transfer : a practical approach, 2007.

Appendices

A Python source code

A.1 Python script of parent class for model generation of AM parts.

```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct  9 13:14:46 2020

@author: kariln

Class created for automatic creation of AM CAD models.
Assumptions:
    Features are rectangular
    Midpoint of sketch is 0,0
    Sketch plane is XY
"""
#add paths
import sys
import os
from pathlib import Path

material_path = Path('../Materials')
sys.path.append(str(material_path.resolve()))

deposition_path = Path('../Deposition_Patterns')
sys.path.append(str(deposition_path.resolve()))

#importing classes
from model import Model
from part import Part
from feature import Feature
from material import Material
from mesh import Mesh
from sets import Set
from zigzag import Zigzag
from raster import Raster
```

```
from job import Job
import pathlib
from amModel import AM


class FEA_MODEL:
    def __init__(self, file_name):
        self.file_name = file_name
        self.file = open(file_name,"w+")
        self.file.truncate(0)
        self.file.close()
        self.work_dir=None
        self.jobs = {}

    def get_file_name(self):
        return self.file_name

    def write(self, string):
        file = open(self.get_file_name(), 'a')
        file.write(string)
        file.close()

    def seperate_sec(self):
        #create sections in code
        self.write('\n')

    def clear_variables(self):
        #deleting all variables in Abaqus
        self.write("import os\n")
        self.write("clear = lambda: os.system('cls')\n")
        self.write("clear()\n")
        self.seperate_sec()

    def imports(self,import_list):
        self.write('#importing modules\n')
        for elem in import_list:
            self.write('import ' + str(elem) + '\n')
            self.write('from ' + str(elem) + " import *\n")
```

```
        self.write('session.journalOptions.setValues(replayGeometry=COORDINATE,recoverGeometry=COORDINATE)\n')
        self.seperate_sec()

    def include_paths(self,path_list):
        self.write('#Include paths\n')
        self.write('import sys\n')
        user_path = Path('../../../..')

        for elem in path_list:
            self.write("sys.path.append(r'" + elem +"')\n")

        plugin_path = Path(user_path / 'abagus_plugins' / 'AM plugin' / 'AMModeler' / 'AMModeler')
        self.write("sys.path.append(r'" + str(plugin_path.resolve()) + "')\n" )
        self.seperate_sec()

    def create_model(self,model_name):
        self.write('#MODEL\n')
        model = Model(model_name)
        self.write(model.get_model_name() +" = mdb.Model(name='" + model.get_model_name() + "')\n")
        self.write("mdb.models['" + model_name + "'].setValues(absoluteZero=-273.15, stefanBoltzmann=5.67E-08)\n")
        self.seperate_sec()
        return model

    def create_part(self,part_name, model, dimensionality,part_type):
        self.write('#PART\n')
        part = Part(part_name, model,dimensionality,part_type)
        part_name = part.get_part_name()
        dimensionality = part.get_dimensionality()
        part_type = part.get_part_type()
        model_name = model.get_model_name()
```

```
        self.write(part_name + "=" + model_name + ".Part(dimensionality =" + dimensionality + " , name= '"+ part_name + "' , type = " + part_type + ")\n")
        self.write('f, e = ' + part_name + '.faces, ' + part_name + '.edges #getting the edges and faces of the part\n')
        model.add_part(part)
        self.seperate_sec()
        return part

    def baseExtrude(self, part, point1, point2, depth):
        baseExtrude = Feature(part, point1, point2, depth, 1)
        baseExtrude.set_feature_name('base_element')
        model_name = part.get_model_name()
        part_name = part.get_part_name()
        self.write('#extrusion of base\n')
        sheetSize = abs(2*(point1[0]-point2[0])*(point1[1]-point2[1]))
        self.write('sketch_name = ' + model_name + ".ConstrainedSketch(name='__profile__',sheetSize= " + str(sheetSize) + ')\n')
        point1_str = str(point1)
        point2_str = str(point2)
        self.write('sketch_name.rectangle(point1=' + point1_str + ',point2=(' + point2_str + '))\n')
        self.write(part_name + '.BaseSolidExtrude(sketch=sketch_name,depth=' + str(depth) + ')\n')
        self.write('e = ' + part_name + '.edges\n')
        self.write('del ' + model_name + ".sketches['__profile__']\n")
        part.add_feature(baseExtrude)
        self.seperate_sec()

    def add_extrude(self,part, point1, point2, depth, nr_layers):
        add_extrude = Feature(part, point1, point2, depth, nr_layers)
```

```
        add_extrude.set_feature_name('add_element')
        base = part.get_features()['base_element']
        sheetSize = abs(2*(base.get_point1()[0]-base.get_point2()[0])*(base.get_point1()[1]-base.get_point2()[1]))
        sketch_plane = (0.,0.,base.get_depth())
        self.write('substrate_top_plane = f.findAt((' + str(sketch_plane) + ',))[0]\n')
        up_edge = (0.,base.get_point2()[1], base.get_depth())
        part_name = part.get_part_name()
        model_name = part.get_model_name()
        self.write('sketch_UpEdge = e.findAt((' + str(up_edge) + ',))[0]\n')
        self.write('sketch_transform = ' + part_name + '.MakeSketchTransform(sketchPlane = substrate_top_plane,sketchUpEdge=sketch_UpEdge,sketchPlaneSide=SIDE1,sketchOrientation=RIGHT,origin=(0.0,0.0,' + str(base.get_depth()) + '))\n')
        self.write('AM_sketch = ' + part.get_model_name() + ".ConstrainedSketch(name = '__profile__',sheetSize=" + str(sheetSize) + ',gridSpacing=0.14, transform=sketch_transform)\n')
        self.write('AM_sketch.rectangle(point1=' + str(point1) + ',point2=' + str(point2) + ')\n')
        self.write(part_name + '.SolidExtrude(depth=' + str(depth) + ',sketchPlane=substrate_top_plane,sketchUpEdge=sketch_UpEdge,sketchPlaneSide=SIDE1,sketchOrientation=RIGHT,sketch = AM_sketch,flipExtrudeDirection=OFF)\n')
        self.write('del ' + model_name + ".sketches['__profile__']\n")
        self.write('#partition AM into layers')
        self.write('\nnr_layers = ' + str(nr_layers) + '\n')
        plane_offset = base.get_depth()
        layer_thickness = depth/nr_layers
        self.write('plane_offset = ' + str(plane_offset) + '\n')
        self.write('for i in range(0,nr_layers):\n')
```

```
        self.write('\tdatum_id = '+ part_name + '.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=plane_offset).id\n')
        self.write('\tplane = ' + part_name + '.datums[datum_id]\n')
        self.write('\tplane_offset += ' + str(layer_thickness) + '\n')
        self.write('\t' + part_name + '_cells = ' + part_name + '.cells\n')
        self.write('\ttop_cell = ' + part_name + '_cells.findAt(((0.,0.,' + str(base.get_depth() + depth) + '),))\n')
        self.write('\t' + part_name + '.PartitionCellByDatumPlane(datumPlane = plane,cells=top_cell)\n')
        part.add_feature(add_extrude)
        self.seperate_sec()

    def assign_material(self, material_name, material_properties, model):
        self.write('#PROPERTY\n')
        material = Material(material_properties, material_name)
        material_name = material.get_material_name()
        model_name = model.get_model_name()
        self.write(material_name + ' = ' + model_name + ".Material(name='" + material_name + "')\n")
        for prop in material_properties:
            property_name = prop[0]
            temperatureDependency = prop[1]
            property_table = material.get_property_table(property_name)
            if temperatureDependency is not None:
                self.write(material_name + '.' + property_name + '(temperatureDependency=' + temperatureDependency + ',table=' + str(property_table) + ')\n')
        else:
```

```
                self.write(material_name + '.' + property_name + '(table=' + str(property_table) + ')\n')
        model.add_material(material)
        self.seperate_sec()

    def assign_section(self, material_name, part, section_name):
        model_name = part.get_model_name()
        part_name = part.get_part_name()
        self.write(model_name + ".HomogeneousSolidSection(name='" + section_name + "', material='" + material_name + "', thickness=None)\n")
        self.write('c = ' + part_name + '.cells\n')
        self.write('region = ' + part_name + '.Set(cells = c, name = "full_part")\n')
        full_part = Set(part,'full_part')
        part.add_set(full_part)
        self.write(part_name + ".SectionAssignment(region=region, sectionName='" + section_name + "', offset=0.0, offsetType=MIDDLE_SURFACE, offsetField='', thicknessAssignment=FROM_SECTION)\n")
        self.seperate_sec()

    def create_instance(self, part):
        self.write('#ASSEMBLY\n')
        model_name = part.get_model_name()
        part_name = part.get_part_name()
        self.write('a = ' + model_name + '.rootAssembly\n')
        self.write('a.DatumCsysByDefault(CARTESIAN)\n')
        self.write("a.Instance(name='" + part_name + "', part= " + part_name + ", dependent=ON)\n")
        self.seperate_sec()

    def create_heat_step(self, step_name, previous, timePeriod, initialInc, minInc,maxInc,deltmx, maxNumInc, model):
        self.write('#STEP\n')
        model_name = model.get_model_name()
```

```python
        self.write(model_name + ".HeatTransferStep(name='" + step_name + "', previous='" + previous +"', timePeriod=" + str(timePeriod) + ', initialInc=' + str(initialInc) + ', minInc=' + str(minInc) + ', maxInc=' + str(maxInc) + ',deltmx=' + str(deltmx) + ',maxNumInc=' + str(maxNumInc) +')\n')
        self.seperate_sec()

    def create_mesh(self, part, road_width):
        self.write('#MESH\n')
        part_name = part.get_part_name()
        #makes global seed half of the road width
        globalSeed = road_width/2
        #creating mesh object
        mesh = Mesh(part,globalSeed)
        part.create_mesh(mesh)
        self.write(part_name + '.seedPart(size=' + str(globalSeed) + ', deviationFactor=0.1, minSizeFactor=0.1)\n')
        #self.write(part_name + '.seedEdgeBySize(edges=substrate_edges, size=' + str(localSeed) + ', deviationFactor=0.1, minSizeFactor=0.1, constraint=FINER)\n')
        self.write('e = ' + part_name + '.edges\n')
        self.write(part_name + '.generateMesh()\n')
        self.write('elemType1 = mesh.ElemType(elemCode=DC3D8, elemLibrary=STANDARD)\n') #heat transfer element type
        self.write('elemType2 = mesh.ElemType(elemCode=DC3D6, elemLibrary=STANDARD)\n')
        self.write('elemType3 = mesh.ElemType(elemCode=DC3D4, elemLibrary=STANDARD)\n')
        self.write('c = ' + part_name + '.cells\n')
        self.write('region = ' + part_name + '.Set(cells = c, name = "part")\n')
        part_set = Set(part, 'part')
        part.add_set(part_set)
```

```
        self.write(part_name + '.setElementType(regions=region, elemTypes=(elemType1,elemType2,elemType3))\n')
        self.seperate_sec()

    def create_node_BC(self, part):
        self.write('#BOUNDARY CONDITION\n')
        model_name = part.get_model_name()
        part_name = part.get_part_name()
        mesh = part.get_mesh()
        globalSeed = mesh.get_global_seed()
        radius = globalSeed/2 #radius of boundingsphere which is half of the globalSeed to ensure only getting the origo node
        self.write('n = '+ part_name + '.nodes\n')
        self.write('origo_node = n.getByBoundingSphere(center = (0.,0.,0.), radius = ' + str(radius) +')\n')
        self.write(part_name + '.Set(nodes=origo_node, name="origo_node")\n')
        self.write('a = ' + model_name + '.rootAssembly\n')
        self.write('region = a.instances["' + part_name + '"].sets["origo_node"]\n')
        self.write(model_name + '.DisplacementBC(name="origo_BC", createStepName="Initial", region=region, u1=SET, u2=SET, u3=SET, ur1=SET, ur2=SET, ur3=SET, amplitude=UNSET, distributionType=UNIFORM, fieldName="", localCsys=None)\n')
        self.seperate_sec()

    def set_room_temp(self,part, roomtemp):
        self.write('#PREDEFINED FIELDS\n')
        part_name = part.get_part_name()
        model_name = part.get_model_name()
        self.write('nodes1 = ' + part_name + '.nodes\n')
        self.write(part_name + '.Set(nodes=nodes1, name="all_nodes")\n')
        all_nodes = Set(part, 'all_nodes')
        part.add_set(all_nodes)
```

```
        self.write('a = ' + model_name + '.rootAssembly\n')
        self.write('region = a.instances["' + part_name + '"].sets["all_nodes"]\n')
        self.write(model_name + '.Temperature(name="room_temp", createStepName="Initial", region=region, distributionType=UNIFORM, crossSectionDistribution=CONSTANT_THROUGH_THICKNESS, magnitudes=(' + str(roomtemp) + ', ))\n')
        self.seperate_sec()

    def set_field_output(self, model, variables):
        model_name = model.get_model_name()
        self.write(model_name + ".fieldOutputRequests['F-Output-1'].setValues(variables=(")
        for variable in variables:
            self.write("'" + variable + "'")
            if variable != variables[-1]:
                self.write(',')
        self.write('))\n')
        self.seperate_sec()

    def create_thermal_AM_model(self,part,amModel_name):
        self.write('#AM PART\n')
        am_Model = AM(part,amModel_name)
        part.add_amModel(am_Model)
        model_name = part.get_model_name()
        part_name = part.get_part_name()
        self.write("amModule.createAMModel(amModelName='" + amModel_name + "', modelName1='" + model_name +"', stepName1='heat', analysisType1=HEAT_TRANSFER, isSequential=OFF, modelName2='', stepName2='', analysisType2=STRUCTURAL, processType=AMPROC_ABAQUS_BUILTIN)\n")
        self.write('a = ' + model_name + '.rootAssembly\n')
        self.write('a.regenerate()\n')
        AM_model_name = 'mdb.customData.am.amModels["' + amModel_name + '"]'
```

```
        self.write(AM_model_name + '.assignAMPart(amPartsData=(("' + part_name + '", "Build Part"), ("", ""), ("", ""), ("", ""), ("", "")))\n')
        self.seperate_sec()
        return am_Model

    def add_event_series(self,am_Model, road_width, deposition_pattern, power, layer_break):
        self.write('#EVENT SERIES\n')
        part = am_Model.get_part()
        amModel_name = am_Model.get_amModel_name()
        AM_model_name = 'mdb.customData.am.amModels["' + amModel_name + '"]'

        add_element = part.get_features()['add_element']
        base_element = part.get_features()['base_element']

        #depth of add_element
        depth = add_element.get_depth()

        #thickness of each layer
        thickness = add_element.get_layer_thickness()

        #road_width
        am_Model.set_road_width(road_width)

        #corner coordinate
        point1 = add_element.get_point1()
        corner_x = point1[0]
        corner_y = point1[1]
        corner_z = base_element.get_depth()

        #x and y length of add_element
        point2 = add_element.get_point2()
        x_length = abs(point1[0]-point2[0])
        y_length = abs(point1[1]-point2[1])

```

```
        if deposition_pattern.lower() == 'raster':
            #__init__(self, z_length, thickness, x_length, y_length, corner_x, corner_y, corner_z, road_width,P):
            dp_object = Raster(depth, thickness, x_length, y_length, corner_x, corner_y, corner_z, road_width,power, layer_break)
        elif deposition_pattern.lower() == 'zigzag':
            #__init__(self, z_length, thickness, x_length, y_length, corner_x, corner_y, corner_z, road_width,P):
            dp_object = Zigzag(depth, thickness, x_length, y_length, corner_x, corner_y, corner_z, road_width,power, layer_break)
        else:
            raise NotImplementedError('This deposition pattern is not implemented');

        dp_object.generate_heat_path()
        dp_object.generate_material_path()
        material_path = pathlib.Path('material_path.txt')
        material_path = material_path.resolve()
        heat_path = pathlib.Path('heat_path.txt')
        heat_path = heat_path.resolve()
        print(heat_path)
        print(material_path)
        self.write(AM_model_name + '.addEventSeries(eventSeriesName="material_path", eventSeriesTypeName=' + "'" + '"ABQ_AM.MaterialDeposition"' + "'" + ', timeSpan="TOTAL TIME", fileName="'+ str(material_path) +'", isFile=ON)\n')
        self.write(AM_model_name + '.addEventSeries(eventSeriesName="heat_path", eventSeriesTypeName=' + "'" + '"ABQ_AM.PowerMagnitude"' + "'" + ', timeSpan="TOTAL TIME", fileName="' + str(heat_path) + '", isFile=ON)\n')
```

```python
        self.seperate_sec()

    def add_table_collections(self,am_Model, absorption_coefficient):
        self.write('#TABLE COLLECTIONS\n')
        part = am_Model.get_part()
        amModel_name = am_Model.get_amModel_name()
        AM_model_name = 'mdb.customData.am.amModels["' + amModel_name + '"]'
        add_element = part.get_features()['add_element']

        #thickness of each layer in add_element
        thickness = add_element.get_layer_thickness()

        #road_width of each layer
        road_width = am_Model.get_road_width()
        if road_width == None:
            raise Exception("Must create event series before table collections")

        #activation offset - how much each bead is offseted
        activation_offset = road_width/2
        am_Model.set_activation_offset(activation_offset)

        #absorption coefficient
        am_Model.set_absorption_coefficient(absorption_coefficient)

        self.write(AM_model_name + '.addTableCollection(tableCollectionName="ABQ_AM_Material")\n')
        self.write(AM_model_name + '.dataSetup.tableCollections["ABQ_AM_Material"].ParameterTable(name=' + "'_parameterTable_" + '"ABQ_AM.MaterialDeposition.Advanced"_' + "', parameterTabletype='" + '"ABQ_AM.MaterialDeposition.Advanced"' + "', parameterData=(('Full', 0.0, 0.0), ))\n")
```

```
self.write(AM_model_name + '.dataSetup.tableCollections["ABQ_AM_Material"].ParameterTable(name = ' + "'_parameterTable_" + '"ABQ_AM.MaterialDeposition.Bead"_' + "', parameterTabletype='" + '"ABQ_AM.MaterialDeposition.Bead"' + "', parameterData=(('Z', " + str(thickness) + "," + str(road_width) +"," + str(activation_offset) + ", 'Below'), ))\n")
self.write(AM_model_name + '.dataSetup.tableCollections["ABQ_AM_Material"].ParameterTable(name = ' + "'_parameterTable_" + '"ABQ_AM.MaterialDeposition"_' + "', parameterTabletype='" + '"ABQ_AM.MaterialDeposition"' + "', parameterData=(('material_path', 'Bead'), ))\n")

self.write(AM_model_name + '.addTableCollection(tableCollectionName="ABQ_AM_Heat")\n')
self.write(AM_model_name + ".dataSetup.tableCollections['ABQ_AM_Heat'].PropertyTable(name='_propertyTable_" + '"ABQ_AM.AbsorptionCoeff"_' + "', propertyTableType='" + '"ABQ_AM.AbsorptionCoeff"' + "', propertyTableData=((" + str(absorption_coefficient) +", ), ), numDependencies=0, temperatureDependency=OFF)\n")
self.write(AM_model_name + ".dataSetup.tableCollections['ABQ_AM_Heat'].ParameterTable(name='_parameterTable_" + '"ABQ_AM.MovingHeatSource"_' + "', parameterTabletype='" + '"ABQ_AM.MovingHeatSource"' + "', parameterData=(('heat_path', 'Goldak'), ))\n")
```

```
        self.write(AM_model_name + ".dataSetup.tableCollections['ABQ_AM_Heat'].ParameterTable(name='_parameterTable_" + '"ABQ_AM.MovingHeatSource.Goldak"_' + "', parameterTabletype='" + '"ABQ_AM.MovingHeatSource.Goldak"' + "', parameterData=(('9', '9', '9', " + str(activation_offset) + ',' + str(thickness) + ', 0.002, 0.004, 0.6, 1.4, 1), ))\n')
        self.write(AM_model_name + ".dataSetup.tableCollections['ABQ_AM_Heat'].ParameterTable(name='_parameterTable_" + '"ABQ_AM.MovingHeatSource.Advanced"_' + "', parameterTabletype='" + '"ABQ_AM.MovingHeatSource.Advanced"' + "', parameterData=(('False', 'False', 'Relative', 0.0, 0.0, -1.0, 1.0), ))\n")
        self.seperate_sec()

    def add_simulation_setup(self, amModel):
        self.write("#SIMULATION SETUP\n")
        part = amModel.get_part()
        mesh = part.get_mesh()
        global_seed = mesh.get_global_seed()
        add_element = part.get_features()['add_element']
        base_element = part.get_features()['base_element']
        base_depth = base_element.get_depth()
        add_depth = add_element.get_depth()
        thickness = add_element.get_layer_thickness()
        total_depth = base_depth + add_depth
        point1 = add_element.get_point1()
        point2 = add_element.get_point2()
        part_name = part.get_part_name()
        model_name = part.get_model_name()
        amModel_name = amModel.get_amModel_name()
        AM_model_name = 'mdb.customData.am.amModels["' + amModel_name + '"]'
        self.write('a = ' + model_name + '.rootAssembly\n')
```

```
        self.write("e = a.instances['" + part_name +
        ↪  "'].elements\n")
        self.write('add_elements = e.getByBoundingBox(' +
        ↪  str(point1[0]) + ',' + str(point1[1]) + ',' +
        ↪  str(base_depth - global_seed/2) + ',' +
        ↪  str(point2[0]) + ',' + str(point2[1]) + ',' +
        ↪  str(total_depth + global_seed/2) + ')\n')
        self.write('a.Set(elements=add_elements,
        ↪  name="add_element")\n')
        self.write('f = a.instances["' + part_name +
        ↪  '"].faces\n')
        self.write('basement_face = f.findAt(((0.0,0.0,0.0)
        ↪  ,))\n')
        self.write('a.Set(faces=basement_face, name =
        ↪  "basement")\n')
        self.write('c = a.instances["' + part_name +
        ↪  '"].cells\n')
        #film contains basement
        self.write('film = c.findAt(((' + str(point1[0]) + ',' +
        ↪  str(point1[1]/3) + ',' + str(base_depth +
        ↪  thickness/2) + '), ), ((' + str(point1[0]) + ',' +
        ↪  str(point1[1]/3) + ',' + str(base_depth +
        ↪  3*thickness/2) + '), ),((' + str(point1[0]) + ',' +
        ↪  str(point1[1]/3) + ',' + str(base_depth +
        ↪  5*thickness/2) + '),  ), ((' + str(point1[0]) + ','
        ↪  + str(point1[1]/3) + ',' + str(base_depth) + '),
        ↪  ))\n')
        self.write('a.Set(cells = film, name = "film")\n')
        #Material arrival:
        self.write(AM_model_name +
        ↪  ".addMaterialArrival(materialArrivalName='Material
        ↪  Source -1', tableCollection='ABQ_AM_Material',
        ↪  followDeformation=OFF, useElementSet=ON,
        ↪  elementSetRegion=('add_element', ))\n")

        #Heat source
```

```
        self.write(AM_model_name + ".addHeatSourceDefinition(heatSourceName='Heat Source -1', dfluxDistribution='Moving-UserDefined', dfluxMagnitude=1, tableCollection='ABQ_AM_Heat', useElementSet=OFF, elementSetRegion=())\n")

        #Cooling
        self.write(AM_model_name + ".addCoolingInteractions(coolingInteractionName='Film', useElementSet=ON, elementSetRegion=('film', ), isConvectionActive=ON, isRadiationActive=OFF, filmDefinition='Embedded Coefficient', filmCoefficient=8.5, filmcoefficeintamplitude='Instantaneous', sinkDefinition='Uniform', sinkTemperature=20, sinkAmplitude='Instantaneous', radiationType='toAmbient', emissivityDistribution='Uniform', emissivity=0.8, ambientTemperature=20, ambientTemperatureAmplitude='Instanteneous')\n")
        self.write(AM_model_name + ".addCoolingInteractions(coolingInteractionName='Basement', useElementSet=ON, elementSetRegion=('basement', ), isConvectionActive=ON, isRadiationActive=ON, filmDefinition='Embedded Coefficient', filmCoefficient=167, filmcoefficeintamplitude='Instantaneous', sinkDefinition='Uniform', sinkTemperature=20, sinkAmplitude='Instantaneous', radiationType='toAmbient', emissivityDistribution='Uniform', emissivity=0.8, ambientTemperature=20, ambientTemperatureAmplitude='Instanteneous')\n")

    def get_jobs(self):
        return self.jobs

    def add_job(self,job):
        job_name = job.get_job_name()
```

```python
        self.get_jobs().update({job_name:job})

    def create_job(self, model, job_name):
        model_name = model.get_model_name()
        job = Job(job_name,model_name)
        self.add:job(job)
        self.write("mdb.Job(name='" + job_name + "', model='" + model_name + "', description='', type=ANALYSIS, atTime=None, waitMinutes=0, waitHours=0, queue=None, memory=90, memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True, explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, echoPrint=OFF, modelPrint=OFF, contactPrint=OFF, historyPrint=OFF, userSubroutine='', scratch='', resultsFormat=ODB, multiprocessingMode=DEFAULT, numCpus=2, numDomains=2, numGPUs=0)\n")

    def submit_job(self,job_name):
        self.write("mdb.jobs['" + job_name + "'].submit(consistencyChecking=OFF)\n")

    def set_work_dir(self, path):
        self.work_dir = path
        path.replace('/','//')
        self.write('os.chdir(' + path + ')\n')

    def get_work_dir(self):
        return self.work_dir

    def save(self):
        path = self.get_work_dir()
        self.write("mdb.saveAs(pathName='" + path + "')\n")

    def create_mechanical(self, model_name, thermal_model_name):
        self.write("mdb.Model(name='" + model_name + "', objectToCopy=mdb.models['" + thermal_model_name + "'])\n")
```

```
        #kopiere modell
        #endre predefined fields: putt inn frames og odb
        #endre BC: substrate
        #endre steps
        #lag ny amModell med thermo-structural
        #endre element type
        #endre field output
```

A.2 Python script of child class with model object class.

```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct  9 14:29:24 2020

@author: kariln
"""
from part import Part

class Model:
    def __init__(self,model_name):
        self.model_name = model_name
        self.parts = {}
        self.materials = {}

    def get_model_name(self):
        return self.model_name

    def get_parts(self):
        return self.parts

    def get_materials(self):
        return self.materials

    def add_part(self,part):
        part_name = part.get_part_name()
        self.get_parts().update({part_name:part})

    def add_material(self, material):
        material_name = material.get_material_name()
        self.get_materials().update({material_name:material})

```

A.3 Python script of child class with part object class.

```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct  9 14:32:49 2020

@author: kariln
"""
from feature import Feature
from mesh import Mesh
from sets import Set

class Part:
    def __init__(self, part_name, model, dimensionality,
    ↪  part_type):
        self.part_name = part_name
        self.model = model
        self.dimensionality = dimensionality
        self.part_type = part_type
        self.features = {}
        self.mesh = None
        self.sets = {}
        self.amModels = {}

    def get_dimensionality(self):
        return self.dimensionality

    def get_model(self):
        return self.model

    def get_model_name(self):
        model = self.get_model()
        return model.get_model_name()

    def get_part_name(self):
        return self.part_name

    def get_part_type(self):
```

```
            return self.part_type


        def get_features(self):
            return self.features

        def add_feature(self,feature):
            feature_name = feature.get_feature_name()
            self.get_features().update({feature_name:feature})

        def create_mesh(self,mesh):
            self.mesh = mesh

        def get_mesh(self):
            return self.mesh

        def get_sets(self):
            return self.sets

        def add_set(self,sett):
            set_name = sett.get_set_name()
            self.get_sets().update({set_name:sett})

        def get_amModels(self):
            return self.amModels

        def add_amModel(self,amModel):
            amModel_name = amModel.get_amModel_name()
            self.get_amModels().update({amModel_name:amModel})
```

A.4 Python script of child class with feature object class.

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Oct  9 16:31:22 2020

@author: kariln
"""

class Feature:
    def __init__(self, part, point1, point2, depth, nr_layers):
        self.feature_name = None
        self.part = part
        self.point1 = point1
        self.point2 = point2
        self.depth = depth
        self.nr_layers = nr_layers
        self.layer_thickness = depth/nr_layers

    def get_feature_name(self):
        return self.feature_name

    def get_part(self):
        return self.part

    def get_depth(self):
        return self.depth

    def get_point1(self):
        return self.point1

    def get_point2(self):
        return self.point2

    def get_layers(self):
        return self.nr_layers

    def get_layer_thickness(self):
```

```
        return self.layer_thickness

    def get_side_length(self):
        return abs(self.get_point2()[0] - self.get_point1()[0])

    def set_feature_name(self,feature_name):
        self.feature_name = feature_name
```

A.5 Python script of child class with material object class.

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 11 13:17:26 2020

@author: Kari Ness
"""
import os
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
from pathlib import Path

class Material:
    def __init__(self,material_properties, material_name):
        #The material_properties should be a list of strings containing material property types
        self.material_properties = material_properties

        #The material should be a string
        self.material_name = material_name

    def get_material_name(self):
        return self.material_name

    def get_path_string(self):
        material_name = self.get_material_name()
        p = Path('../Materials/' + material_name)
        return p.resolve()

    def get_property_file_path(self, material_property):
        material_name = self.get_material_name()
        file_name = material_name + '_' + material_property + '.txt'
        file = os.path.join(os.path.dirname(os.path.abspath(__file__ + "/../")),"Materials", material_name, file_name)
```

```
        return file

    def get_property_table(self, material_property):
        file_path = self.get_property_file_path(material_property)
        table = []
        with open(file_path, "r") as f:
            for line in f:
                tmp = line.strip().split(",")
                for i in range(0,len(tmp)):
                    tmp[i] = float(tmp[i])
                tmp = tuple(tmp)
                table.append(tmp)
        return table

    def plot_conductivity(self):
        material_name = self.get_material_name()
        degree_sign= u'\N{DEGREE SIGN}'
        table_conductivity = self.get_property_table('Conductivity')
        temp = [x[0] for x in table_conductivity]
        conductivity = [x[1] for x in table_conductivity]
        mpl.style.use('seaborn-dark-palette')
        plt.plot(conductivity, temp, c= 'firebrick')
        plt.xlabel('Temperature [C' + degree_sign + ']')
        plt.ylabel('Conductivity [W/m' + degree_sign + 'C]')
        path = str(self.get_path_string()) + '/'
        plt.savefig(path + '/' + material_name + '_Conductivity.png')
        plt.show()

    def plot_specific_heat(self):
        mpl.style.use('seaborn-dark-palette')
        degree_sign= u'\N{DEGREE SIGN}'
        material_name = self.get_material_name()
        table_specific = self.get_property_table('SpecificHeat')
        temp = [x[0] for x in table_specific]
        specific_heat = [x[1] for x in table_specific]
```

```
        plt.plot(specific_heat, temp, c='firebrick')
        plt.xlabel('Temperature [C' + degree_sign + ']')
        plt.ylabel("Specific heat [J/kg" + degree_sign + 'C]')
        path = str(self.get_path_string()) + '/'
        plt.savefig(path + '/' + material_name + '_SpecificHeat.png')
        plt.show()

    def plot_yield_stress(self):
        mpl.style.use('seaborn-dark-palette')
        degree_sign= u'\N{DEGREE SIGN}'
        material_name = self.get_material_name()
        table_yield = self.get_property_table('Plastic')
        plastic_strain = [x[1] for x in table_yield]
        yield_stress_tmp = [x[0] for x in table_yield]
        temp_tmp = [x[2] for x in table_yield]
        yield_stress = []
        temp = []
        for i in range(0,len(plastic_strain)):
            if plastic_strain[i] != 0:
                temp.append(temp_tmp[i])
                yield_stress.append(yield_stress_tmp[i])
        plt.plot(temp, yield_stress,c='firebrick')
        plt.xlabel('Temperature [C' + degree_sign + ']')
        plt.ylabel("Yield stress [MPa]")
        path = str(self.get_path_string()) + '/'
        plt.savefig(path +  '/' + material_name + '_Yield_Stress.png')
        plt.show()

    def plot_youngs_module(self):
        mpl.style.use('seaborn-dark-palette')
        degree_sign= u'\N{DEGREE SIGN}'
        material_name = self.get_material_name()
        table_E = self.get_property_table('Elastic')
        E = [x[0] for x in table_E]
        temp = [x[2] for x in table_E]
        plt.plot(temp,E, c='firebrick')
```

```python
        plt.xlabel('Temperature [C' + degree_sign + ']')
        plt.ylabel("Young's Modulus [GPa]")
        path = str(self.get_path_string()) + '/'
        plt.savefig(path + '/' + material_name + '_Elastic.png')
        plt.show()

    def plot_expansion(self):
        mpl.style.use('seaborn-dark-palette')
        degree_sign = u'\N{DEGREE SIGN}'
        alpha_sign = '\u03B1'
        material_name = self.get_material_name()
        table_exp = self.get_property_table('Expansion')
        alpha = [x[1]*10**(6) for x in table_exp]
        T = [x[0] for x in table_exp]
        plt.plot(T,alpha, c='firebrick')
        plt.xlabel('Temperature [C' + degree_sign + ']')
        plt.ylabel(alpha_sign + 'x10^-6[1/' + degree_sign + 'C]')
        path = str(self.get_path_string()) + '/'
        plt.savefig(path + '/' + material_name + '_Expansion.png')
        plt.show()

    def plot_strain_hardening(self, temperatures):
        #x: Strain
        #y: True stress [MPa]
        material_name = self.get_material_name()
        degree_sign= u'\N{DEGREE SIGN}'
        legends = []
        strain = []
        stress = []
        fig, ax = plt.subplots()
        for index, t in enumerate(temperatures):
            legends.append(str(t) + degree_sign + 'C')
            material_property = 'StrainHardening_' + str(t)
            table = self.get_property_table(material_property)
            strain.append([x[0] for x in table])
            stress.append([x[1] for x in table])
            ax.plot(strain[index],stress[index])
```

```
            plt.draw()
        for var in stress:
            biggest = 0
            for elem in var:
                if elem > biggest:
                    biggest = elem
            plt.annotate('%0.2f' % biggest, xy=(1, biggest),
             ↪  xytext=(8, 0),
                 xycoords=('axes fraction', 'data'),
                  ↪  textcoords='offset points')
        plt.rcParams.update({'font.size': 15})
        plt.xlabel('Strain')
        plt.ylabel('Stress [MPa]')
        plt.legend(legends)
        path = str(self.get_path_string()) + '/'
        plt.savefig(path + '/' + material_name +
         ↪  '_StrainHardening.png')
        plt.show()


    def material_plot(self, temperatures):
        self.plot_conductivity()
        self.plot_yield_stress()
        self.plot_specific_heat()
        self.plot_youngs_module()
        self.plot_expansion()
        self.plot_strain_hardening(self, temperatures)

def main():
    material_name = 'AA2319'
    material_properties = ['Conductivity','Density','Elastic','E
     ↪  xpansion','LatentHeat',
     ↪  'Plastic','SpecificHeat']
    material = Material(material_properties, material_name)
    material.plot_strain_hardening([20,316, 371, 550])

main()
```

A.6 Python script of child class with mesh object class.

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 15 12:14:47 2020

@author: Kari Ness
"""

class Mesh:
    def __init__(self, part, global_seed):
        self.part = part
        self.global_seed = global_seed

    def get_global_seed(self):
        return self.global_seed
```

A.7 Python script of child class with set object class.

```
# -*- coding: utf-8 -*-
"""
Created on Sat Oct 17 17:25:43 2020

@author: Kari Ness
"""


class Set:
    def __init__(self, part, set_name):
        self.part = part
        self.set_name = set_name

    def get_part(self):
        return self.part

    def get_part_name(self):
        return part.get_part_name()

    def get_set_name(self):
        return self.set_name
```

A.8 Python script of parent class for generation of heat and material paths for various deposition patterns.

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 24 21:33:09 2020

@author: Kari Ness
"""
import abc #for abstract methods

#Creating a parent class for all deposition patterns
class Pattern:
  def __init__(self, z_length, thickness, x_length, y_length, corner_x, corner_y, corner_z, road_width,P, layer_break):
    #initializing geometry properties
    self.thickness = thickness
    self.road_width = road_width
    self.length = [x_length, y_length,z_length]

    #initializing deposition velocity with default value 0.01
    self.v = 0.015

    #initializing the a time intervall between each deposited layer
    self.layer_break = layer_break

    #energy deposition
    self.P = P

    #initializing the start coordinate of the pattern
    self.corner_coord = (corner_x, corner_y, corner_z)

    #initializing a dictionary with axes
    self.axis = {'deposition': 0, 'transverse': 1, 'stack': 2}


#Creating getters and setters
```

```
    @abc.abstractmethod
    def get_path(self):
        pass

    def get_layer_break(self):
        return self.layer_break

    def generate_heat_path(self):
        path = self.get_path()

        #creating text files for heat and material path
        heat_path = open("heat_path.txt","w+")
        heat_path.truncate(0)

        for elem in path:
            heat_path.write(self.coord_string(elem[0], elem[1],
             ↪  elem[2], elem[3], elem[4]))

    def generate_material_path(self):
        path = self.get_path()
        #creating text files for heat and material path
        material_path = open("material_path.txt","w+")
        material_path.truncate(0)

        for elem in path:
            material_path.write(self.coord_string(elem[0],
             ↪  elem[1], elem[2], elem[3], elem[5]))



    def get_length(self):
        return self.length

    def get_z_length(self):
        return self.get_length()[2]


    def get_layer_nr(self):
```

```
        return int(self.get_length()[self.get_stack_dir()]/self.get_thickness())

    def get_thickness(self):
        return self.thickness

    def set_thickness(self, thickness):
        self.thickness = thickness

    def get_x_length(self):
        return self.get_length()[0]

    def get_y_length(self):
        return self.get_length()[1]

    def get_corner_coord(self):
        return self.corner_coord

    def get_road_width(self):
        return self.road_width

    def set_road_width(self, road_width):
        self.road_width = road_width

    def get_axis(self):
        return self.axis

    def set_axis(self, deposition, transverse, stack):
        if (deposition == 0 or deposition == 1 or deposition == 2):
            self.axis['deposition'] = deposition
        else:
            raise ValueError("Invalid deposition axis!")

        if (transverse == 0 or transverse == 1 or transverse == 2) and (transverse != deposition):
            self.axis['transverse'] = transverse
        else:
            raise ValueError("Invalid deposition axis!")
```

```
        if (stack == 0 or stack == 1 or stack == 2) and (stack !=
        ↪  deposition and stack != transverse):
            self.axis['stack'] = stack
        else:
            raise ValueError("Invalid deposition axis!")

    def get_stack_dir(self):
        return self.get_axis()['stack']

    def get_deposition_dir(self):
        return self.get_axis()['deposition']

    def get_transverse_dir(self):
        return self.get_axis()['transverse']

    def get_power(self):
        return self.P

    def set_power(self, P):
        self.P = P

    def get_area(self):
        return self.get_road_width()*self.get_thickness()

    def get_velocity(self):
        return self.v

    def set_velocity(self,v):
        self.v = v

    def coord_string(self,t,x,y,z,p):
        temp= "{},{},{},{},{}\n"
        return temp.format(t,x,y,z,p)
```

A.9 Python script for generation of zig-zag deposition pattern.

```
# -*- coding: utf-8 -*-
"""
Created on Sun Sep 27 20:50:49 2020

@author: Kari Ness
"""
import pattern

#Zig-Zag deposition pattern
class Zigzag(pattern.Pattern):
    def __init__(self, z_length, thickness, x_length, y_length, corner_x, corner_y, corner_z, road_width,P, layer_break):
        super().__init__(z_length, thickness, x_length, y_length, corner_x, corner_y, corner_z, road_width,P, layer_break)

    def pass_time(self):
        return self.get_length()[self.get_deposition_dir()]/self.get_velocity()

    def up_time(self):
        return (self.get_road_width()/(self.get_velocity()))/10

    def nr_passes(self):
        return int(self.get_length()[self.get_transverse_dir()]/self.get_road_width())

    def get_print_coord(self):#input 0 for heat, 1 for material
        #coord = [x,y,z]
        coord = [self.get_corner_coord()[0],self.get_corner_coord()[1],self.get_corner_coord()[2],self.get_corner_coord()[2]]
        coord[self.get_transverse_dir()] += self.get_road_width()/2
```

```
        coord[self.get_stack_dir()] += self.get_thickness()
        return coord

    def get_path(self):
        #setting the start coordinate of the raster
        coord = self.get_print_coord()

        #initial conditions:
        start = self.get_print_coord()

        P = self.get_power()
        A = self.get_area()
        time = 0
        direction = 1 #defines if the deposition moves forward or backwards
        path = []

        layers = self.get_layer_nr()
        passes = self.nr_passes()
        pass_time = self.pass_time()
        up_time = self.up_time()

        for i in range(0,int(layers)):
          for j in range(0,int(passes)):
              path.append([time,coord[0],coord[1], coord[2], P,A])
              coord[self.get_deposition_dir()] += direction*self.get_length()[self.get_deposition_dir()]
              direction = direction*(-1)
              time += pass_time
              path.append([time,coord[0],coord[1],coord[2],0,0])
              coord[self.get_transverse_dir()] += self.get_road_width()
              time += up_time
          coord[self.get_deposition_dir()] = start[self.get_deposition_dir()]
```

```
            coord[self.get_transverse_dir()] =
             ↪ start[self.get_transverse_dir()]
            coord[self.get_stack_dir()] = self.get_thickness() +
             ↪ coord[self.get_stack_dir()]
            time += self.get_layer_break()
        return path

#def main():
#    zigzag = Zigzag(0.06, 0.01, 0.06, 0.06, -0.03, -0.03, 0.02,
 ↪ 0.01,5000)
#    zigzag.generate_heat_path()
#    zigzag.generate_material_path()
#main()
```

A.10 Python script for generation of raster deposition pattern.

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 24 21:39:34 2020

@author: Kari Ness
"""

import zigzag

class Raster(zigzag.Zigzag):

    def __init__(self, z_length, thickness, x_length, y_length,
    ↪ corner_x, corner_y, corner_z, road_width,P, layer_break):
        super().__init__(z_length, thickness, x_length,
          ↪ y_length, corner_x, corner_y, corner_z,
          ↪ road_width,P,layer_break)


    def get_path(self):
        #setting the start coordinate of the raster
        coord = self.get_print_coord()

        #initial conditions:
        start = self.get_print_coord()

        P = self.get_power()
        A = self.get_area()
        time = 0
        path = []

        layers = self.get_layer_nr()
        passes = self.nr_passes()
        pass_time = self.pass_time()
        up_time = self.up_time()

```

```
    for i in range(0,int(layers)):
      for j in range(0,int(passes)):
          path.append([time,coord[0],coord[1], coord[2], P,A])
          coord[self.get_deposition_dir()] += self.get_length()[self.get_deposition_dir()]
          time += pass_time
          path.append([time,coord[0],coord[1],coord[2],0,0])
          coord[self.get_transverse_dir()] += self.get_road_width()
          coord[self.get_deposition_dir()] -= self.get_length()[self.get_deposition_dir()]
          time += up_time
      coord[self.get_deposition_dir()] = start[self.get_deposition_dir()]
      coord[self.get_transverse_dir()] = start[self.get_transverse_dir()]
      coord[self.get_stack_dir()] += self.get_thickness()
      time += self.get_layer_break()
    return path
```

A.11 Python script of child class with Job object class.

```
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 28 13:51:24 2020

@author: kariln
"""


class Job:
    def __init__(self,job_name, model_name):
        self.job_name = job_name
        self.model_name = model_name

    def get_job_name(self):
        return self.job_name

    def get_model_name(self):
        return self.model_name

```

A.12 Python script of child class with AM simulation object class.

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 25 10:17:14 2020

@author: Kari Ness
"""

class AM:
    def __init__(self,part, amModel_name):
        self.part = part
        self.amModel_name = amModel_name
        self.road_width = None
        self.activation_offset = None
        self.absorption_coefficient = None

    def get_part(self):
        return self.part

    def get_part_name(self):
        return self.get_part().get_part_name()

    def get_amModel_name(self):
        return self.amModel_name

    def set_road_width(self,road_width):
        self.road_width = road_width

    def get_road_width(self):
        return self.road_width

    def set_activation_offset(self, activation_offset):
        self.activation_offset = activation_offset

    def set_absorption_coefficient(self,absorption_coefficient):
        self.absorption_coefficient = absorption_coefficient
```

A.13 Python script for experiment with zigzag deposition pattern.

```
# -*- coding: utf-8 -*-
"""
Created on Tue Nov  3 11:48:23 2020

@author: kariln
"""

# -*- coding: utf-8 -*-
"""
Created on Tue Oct 27 13:22:28 2020

@author: kariln

Thermal experiment with zigzag pattern and cooling time 10s.
↪ between each layer
"""
#add paths
import sys
from pathlib import Path

abaqus_path = Path('../../')
sys.path.append(str(abaqus_path.resolve()))

material_path = Path('../Materials')
sys.path.append(str(material_path.resolve()))

deposition_path = Path('../Deposition_Patterns')
sys.path.append(str(deposition_path.resolve()))

from create_script import FEA_MODEL
from get_odb import Odb

"""THERMAL MODEL"""
scripted_part = FEA_MODEL('experiment_3.py')
scripted_part.clear_variables()
```

```
scripted_part.imports(['part','material','section','assembly','step','interaction','load','mesh','job','sketch','visualization','connectorBehavior', 'customKernel','amModule', 'amKernelInit', 'amConstants', 'copy','os'])
scripted_part.include_paths([])
models = {}

#MODEL
thermal = scripted_part.create_model('thermal')
models.update({thermal.get_model_name():thermal})

#PART
part1 = scripted_part.create_part('part1', thermal, 'THREE_D','DEFORMABLE_BODY')
scripted_part.baseExtrude(part1, (-0.1,-0.1), (0.1,0.1), 0.02)
scripted_part.add_extrude(part1,(-0.06,-0.06),(0.06,0.06),0.0092,4)

#PROPERTY
scripted_part.assign_material('AA2319',[['Conductivity', 'ON'],['Density', 'OFF'],['Elastic', 'ON'],['Expansion','ON'],['LatentHeat', None],['Plastic','ON'],['SpecificHeat', 'ON']], thermal)
scripted_part.assign_section('AA2319',part1,'Part_Section')

#ASSEMBLY
scripted_part.create_instance(part1)

#STEP
scripted_part.create_heat_step('heat','Initial',4000,0.01,1E-8,1,1000, 10000,thermal)

#MESH
scripted_part.create_mesh(part1,0.005)

#LOAD
scripted_part.create_node_BC(part1)
```

```python

#PREDEFINED FIELD
scripted_part.set_room_temp(part1, 20)

#FIELD OUTPUT
scripted_part.set_field_output(thermal, ['NT','TEMP'])

#AM MODEL
am_Model = scripted_part.create_thermal_AM_model(part1,'AM_thermal')
scripted_part.add_event_series(am_Model, 0.01,'zigzag',5000,10)
scripted_part.add_table_collections(am_Model,0.9)
scripted_part.add_simulation_setup(am_Model)

#JOB
scripted_part.create_job(thermal, 'experiment1_thermal')
#scripted_part.submit_job('experiment1_thermal')

""" MECHANICICAL MODEL"""

""" ODB """
process_odb = Odb('experiment1_thermal',scripted_part, part1)
process_odb.clear_variables()
process_odb.imports(['OdbAccess', 'os'])
```

A.14 Python script for experiment with raster deposition pattern.

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Oct 30 10:05:13 2020

@author: kariln
Thermal experiment with raster
"""
#add paths
import sys
from pathlib import Path

abaqus_path = Path('../')
sys.path.append(str(abaqus_path.resolve()))

from create_script import FEA_MODEL
from get_odb import Odb

"""THERMAL MODEL"""
scripted_part = FEA_MODEL('experiment_2.py')
scripted_part.clear_variables()
scripted_part.imports(['part','material','section','assembly','step','interaction','load','mesh','job','sketch','visualization','connectorBehavior', 'customKernel','amModule', 'amKernelInit', 'amConstants', 'copy','os'])
scripted_part.include_paths([])
models = {}

#MODEL
thermal = scripted_part.create_model('thermal')
models.update({thermal.get_model_name():thermal})

#PART
part1 = scripted_part.create_part('part1', thermal, 'THREE_D','DEFORMABLE_BODY')
scripted_part.baseExtrude(part1, (-0.1,-0.1), (0.1,0.1), 0.02)
```

```
scripted_part.add_extrude(part1,(-0.06,-0.06),(0.06,0.06),0.0092
    ,4)

#PROPERTY
scripted_part.assign_material('AA2319',[['Conductivity',
    'ON'],['Density', 'OFF'],['Elastic',
    'ON'],['Expansion','ON'],['LatentHeat',
    None],['Plastic','ON'],['SpecificHeat', 'ON']], thermal)
scripted_part.assign_section('AA2319',part1,'Part_Section')

#ASSEMBLY
scripted_part.create_instance(part1)

#STEP
scripted_part.create_heat_step('heat','Initial',4000,0.01,1E-8,1
    ,1000,
    10000,thermal)

#MESH
scripted_part.create_mesh(part1,0.01)

#LOAD
scripted_part.create_node_BC(part1)

#PREDEFINED FIELD
scripted_part.set_room_temp(part1, 20)

#FIELD OUTPUT
scripted_part.set_field_output(thermal, ['NT','TEMP'])

#AM MODEL
am_Model =
    scripted_part.create_thermal_AM_model(part1,'AM_thermal')
scripted_part.add_event_series(am_Model, 0.01,'raster',5000,10)
scripted_part.add_table_collections(am_Model,0.9)
scripted_part.add_simulation_setup(am_Model)

#JOB
```

```
scripted_part.create_job(thermal, 'experiment2_thermal')
#scripted_part.submit_job('experiment2_thermal')
```

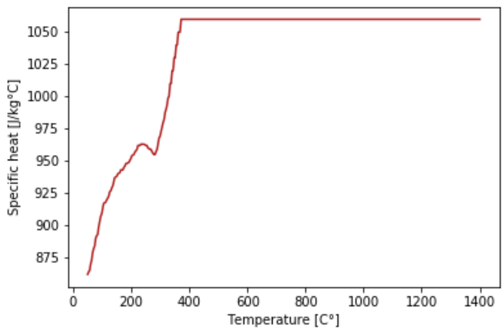
B Material properties

B.1 Aluminium alloy 2319 (AA2319)

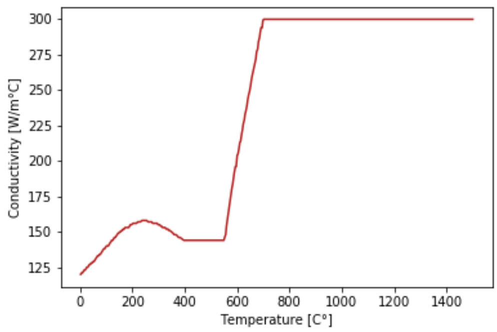
The material properties have been found in *Evaluation of 2D and 3D FEA Models for Predicting Residual Stress and Distortion* by P. Michaleris and Z. Feng and G. Campbell [32]. The temperature dependent properties can be seen in figure 25 and temperature-independent properties can be seen in table 2.

AA2319	
Mass density	2823 kg/m3
Liquidus temperature	$643°C$
Solidus temperature	$543°C$

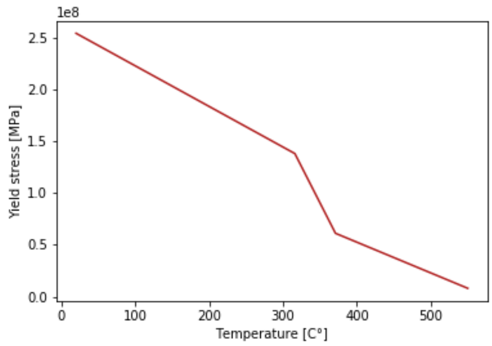
Table 2: Temperature independent properties of AA2319



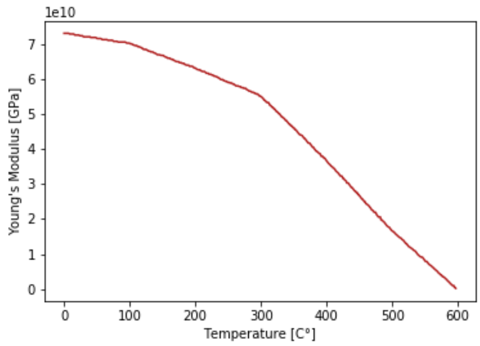
(a) Specific Heat



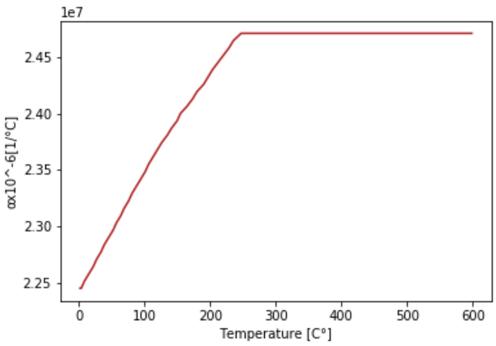
(b) Conductivity



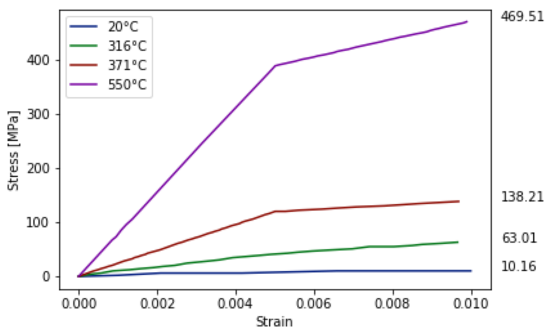
(c) Yield Stress



(d) Young's module



(e) Coefficient of thermal expansion



(f) Strain hardening

Figure 25: Temperature dependent properties for AA2319