Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 232 (2024) 3239–3246

Procedia Computer Science

www.elsevier.com/locate/procedia

5th International Conference on Industry 4.0 and Smart Manufacturing

Recursive autoencoder network for prediction of CAD model parameters from STEP files

Victoria Miles[a,*], Stefano Giani[a], Oliver Vogt[a], Raheleh Kafieh[a]

[a]*Department of Engineering, Durham University, Stockton Road, Durham, DH1 3LE, United Kingdom*

Abstract

Databases of 3D CAD (computer aided design) models are often large and lacking in meaningful organisation. Effective tools for automatically searching for, categorising and comparing CAD models, therefore, have many potential applications in improving efficiency within design processes. This paper presents a novel asymmetric autoencoder model, consisting of a recursive encoder network and fully-connected decoder network, for the reproduction of CAD models through prediction of the parameters necessary to generate a 3D part design. Inputs to the autoencoder are STEP (standard for the exchange of product data) files, an ISO standard CAD model format, compatible with all major CAD software. A complete 3D model can be accurately reproduced using a STEP file, meaning that all geometric information can be used to contribute to the final encoded vector, with no loss of small detail.

In a CAD model of overall size $10 \times 10 \times 10$ units, for 90% of models, the class of an added feature is estimated with maximum error of 0.6 units, feature size with maximum error of 0.4 units and coordinate values representing position with maximum error of 0.3 units. These results demonstrate the successful encoding of complex geometric information, beyond merely the shape of the 3D object, with potential application in the design of search engine functionality.

Keywords: CAD models; STEP files; recursive neural network; autoencoder

1. Introduction

The interpretation of 3D CAD models provides a significant challenge for deep learning models. Traditional approaches for the manipulation of 3D data such as the multi-view [1], voxel [2] and point-cloud [3] methods, whilst often successful when applied to general shape recognition tasks, can have serious limitations when applied to the useful analysis of CAD model data. Issues such as limited resolution and loss of real 3D information when transforming the input data into the form required by the neural network will have increased impact when regarding a potentially complex 3D model part, in which small details may be of high significance.

* Corresponding author.
E-mail address: victoria.s.miles@durham.ac.uk

1877-0509
10.1016/j.procs.2024.02.139

Much of the existing research into the automatic understanding of 3D CAD data focuses on the tasks of machining feature recognition and localisation, with the goal of creating a bridge between CAD design and the real processes necessary to manufacture a part. Whilst this is clearly a valuable task, comparatively little existing research focuses on other applications for CAD data interpretation, such as the development of automatic search-engine capabilities within CAD model databases. A deep learning model capable of automatically categorising, grouping and comparing existing part designs within an existing database has the potential to greatly improve efficiency within the design process. For instance, comparing a new design to similar existing part designs could lead to increased standardisation of parts, improving manufacturing efficiency. When adapting a part design, searching the database for existing models with the desired change could save considerable time in manually developing a new design.

Intelligent search-engine functionality relies on the compression of complex input data into meaningful vector encodings, which can be compared, classified and used to extract information about the CAD model geometry. Ideally, the encoder network producing these encodings should be trained not for a specific task, but to produce an output vector which is best capable of representing all information from the input CAD model.

The autoencoder [4] is a class of neural network commonly used to produce compact general representations of data inputs. A traditional autoencoder architecture consists of mirrored encoder and decoder networks. The encoder compresses the input data into a compact form and the decoder attempts to use these encodings to reproduce the input data. These models can be trained in an unsupervised manner, with the goal of training being to encode the inputs in such a way that no information is lost and each data point can be perfectly reconstructed by the decoder. Thus, there is no need for labelled training data. Once trained, the encoder half of autoencoder networks can be applied to diverse tasks, such as classification [5], generation [6] and dimensionality reduction [7].

In [8], a recursive neural network is used to perform classification of simple machining features using STEP files as direct input, with accuracy approaching 100%. The interpretation of the text-based STEP format is here treated as a language processing task, in which a recursive neural network [9] is applied to the inherent tree structure of STEP data, without first converting to a more easily-interpretable 3D data format. This method is extended to the detection of multiple machining features in [10], with results demonstrating the robustness of this approach when presented with small machining features relative to the CAD model size.

In this paper, we propose to incorporate the recursive network presented in [8] into a novel autoencoder architecture, trained to reproduce the parameters used to create an input CAD model.

2. Methods

Figure 1 shows an overview of the autoencoder architecture. CAD models are generated using a list of parameters which define the geometric properties of a fixed set of 3D shapes. These models are then saved in STEP format and each STEP file is parsed to generate an encoded tree structure, suitable as input to a recursive neural network. The recursive network encodes the information from the input tree into a single-vector format. This vector is then fed into a fully-connected decoder network which outputs a list of parameters which can be used to generate a new CAD model.

As is the case with traditional autoencoder architectures, our network is trained to decode and then reproduce the input data. However, this architecture is somewhat unusual in its asymmetric nature, as the decoder does not predict the direct input of the encoder and the two networks have entirely different structures. Nevertheless, the desired outcome is the same as for any autoencoder network: to encode input data as a vector whilst maintaining all information necessary to reconstruct the source.

This section will outline the implementation of our model, detailing the parsing of input STEP data and the architecture of the recursive encoder and fully-connected decoder networks.

2.1. STEP Parsing

The main body of a STEP file consists of a list of nodes which are connected together to form a hierarchical structure. An example segment of a STEP file is included in Figure 1. Each node is represented by a single line in the file, containing a unique ID number, node category, and a list of the node's children, given as parameters. Children may be the ID numbers of other nodes, numerical values representing coordinate points or additional values and flags.

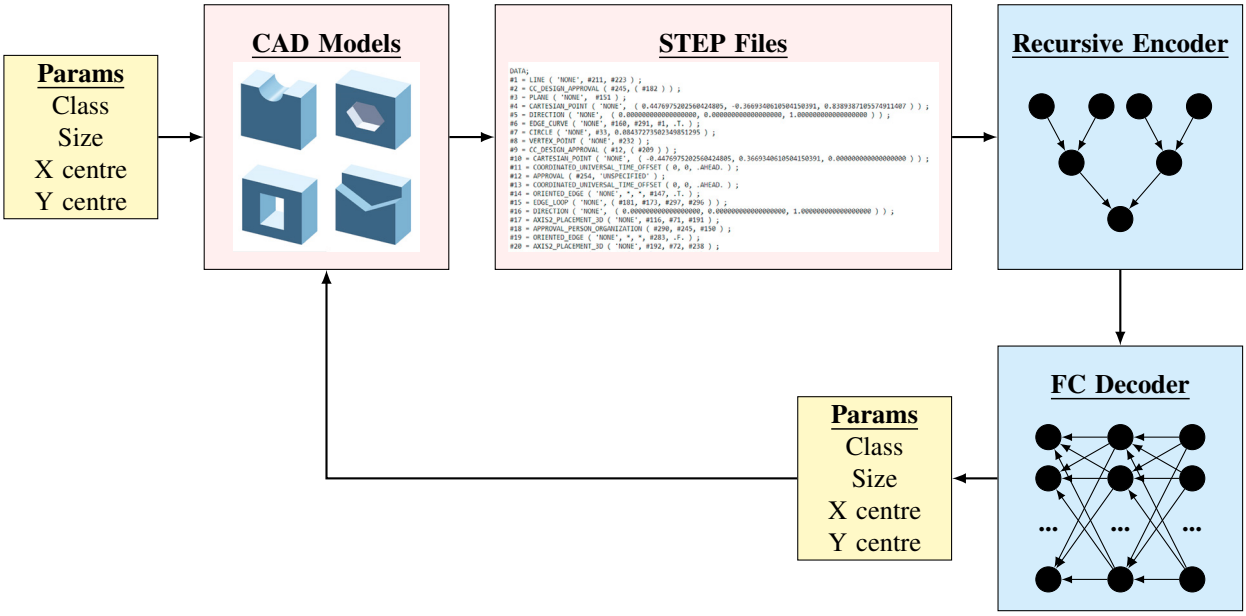


Fig. 1: Autoencoder structure

The order of lines in a STEP file is not significant as the structure is formed based on the connections between parent and child nodes, and so lines may be randomly ordered or clustered by category.

In order to access the information from a STEP file in a format which can be interpreted by a neural network, several parsing steps must be performed. First, the meaningless order of lines in the file must be replaced by the tree structure inherent in the data. Next, each node category is encoded as a one-hot vector which can be input into the neural network. These parsing steps are performed according to the procedure detailed in [8], and result in hierarchically structured data with an encoded category assigned as the input for each node in the data tree.

2.2. Recursive Encoder

The encoder half of our autoencoder architecture is a recursive neural network, specifically the Child-Sum Tree-LSTM network first proposed in [9]. The modified LSTM cell utilised in this network is governed by the following equations, for a node j with a set of children $C(j)$:

$$\tilde{h}_j = \sum_{k \in C(j)} h_k \tag{1}$$

$$i_j = \sigma(W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)}) \tag{2}$$

$$f_{jk} = \sigma(W^{(f)} x_j + U^{(f)} h_k + b^{(f)}) \tag{3}$$

$$o_j = \sigma(W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)}) \tag{4}$$

$$u_j = tanh(W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)}) \tag{5}$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k \tag{6}$$

$$h_j = o_j \odot tanh(c_j) \tag{7}$$

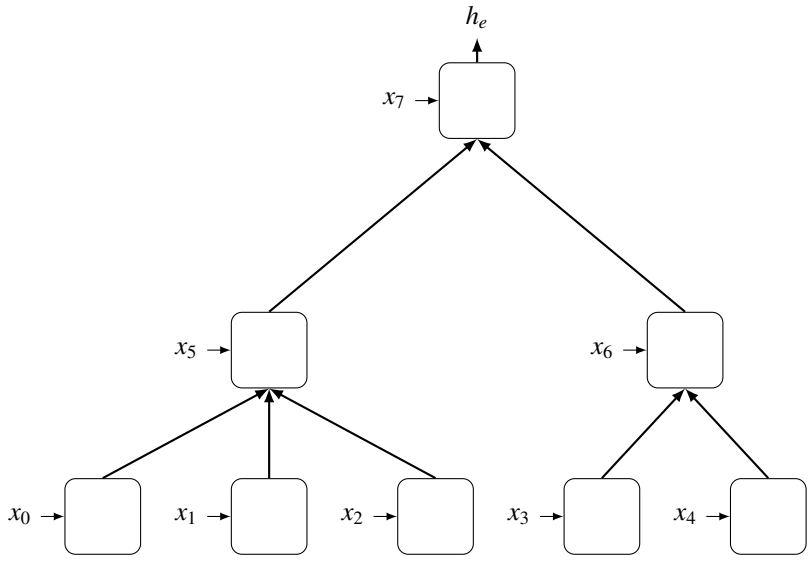


Fig. 2: Example recursive structure, where x_n denotes the input vector at node n and h_e is the output hidden state for the top-level node, representing the final output of the encoder.

where inputs are the hidden states from each child node, h_k, and the new input, x_j; outputs are the cell state, c_j, and new hidden state, h_j; and behaviour is controlled by three gates, the input gate, i, forget gate, f and output gate, o. The learned parameters of the LSTM cell are the weight matrices, U and W, and bias terms, b.

The recursive neural network applies this modified LSTM cell at every node within a hierarchical data structure, taking the hidden states from child nodes and new input at the current node as inputs, and calculating new hidden states as output which will then be passed to the current node's parents in the next level of the tree. Figure 2 shows how a tree-LSTM cell might be applied to a simple data tree.

The geometric data from a STEP file will always culminate in a single top-level node, representing the entire model geometry. Therefore, a recursive neural network can be applied as an encoder network, taking a data tree as input and producing a single-vector encoding, the output hidden state for the top-level node, as shown in Figure 2.

2.3. Fully-Connected Decoder

The decoder network is a two-layer fully connected network which produces an output vector of length 4, with each value representing one of the parameters used to generate the CAD model. A sigmoid activation function is applied to the output of the decoder network as:

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{8}$$

resulting in values distributed between 0 and 1. These normalised values are then scaled according to the range of values possible for each parameter, to produce a final list of predicted parameters for each CAD model.

2.4. Network Parameters and Training

When training for multiple classes of CAD model, it is inevitable that some parameters will be necessary to reproduce certain classes of model but not others. Therefore, a masked MSE loss was implemented as follows, to eliminate irrelevant parameters when calculating loss and performing gradient descent:

$$L = \frac{1}{\sum m_n} \sum_{n=1}^{N} (m_n(x_n - y_n))^2 \tag{9}$$

where x and y are the predicted and target vectors respectively and m is a mask vector, set to one for valid parameters and zero for invalid parameters. All vectors have length N.

The random search method was utilised to select optimal hidden size for the encoder network to be 280, with intermediate hidden size for the decoder of 60. Learning rate was initialised at 1e-3 and the Adam optimiser [11] used for gradient descent. Code is written in PyTorch and computations are carried out using a Xeon E5-2609 v3 CPU.

3. Results and Discussion

3.1. Dataset

The autoencoder model is trained and tested using a dataset of machining features. Each CAD model in the dataset is built from a starting block which is a uniform cube, of side length 10 units. As the goal is to assess CAD models based on geometric shape, and all models are normalised through the use of an identical starting block, real units are ignored, with the focus being on geometric parameters relative to the overall model size. To generate unique CAD models, one of 24 classes of machining features, including holes, slots, fillets and chamfers, is added to each starting block. Some examples of the machining feature classes used are presented in Figure 3.

The necessary parameters to generate or recreate any model from the dataset are the feature class, size, depth and two coordinate values representing the position of the centre of the feature. As depth is somewhat dependent on feature class, it was decided to ignore it for the purposes of this work, leaving four parameters to be predicted. Figure 4 shows each of these parameters for an example feature class. Not every parameter is relevant to every model class. For instance, the chamfer class, for which only the feature size can be altered, as its position is fixed to an edge.

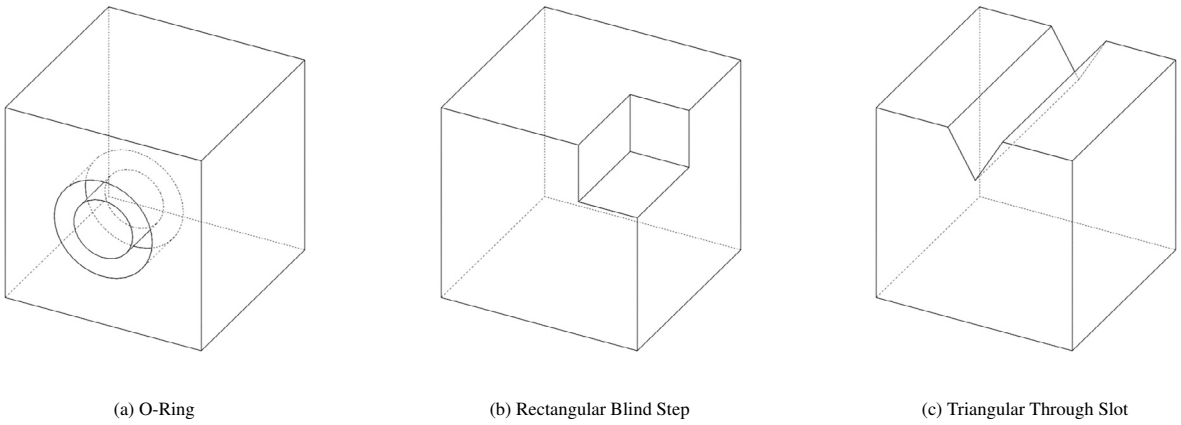
All relevant parameters are randomly selected for each model when generating the dataset. In addition, models are randomly rotated by 90 degree increments during generation, to increase variety in the dataset. This can result in 3D models which may be functionally similar but which will be represented by significantly different STEP files, as the STEP format utilises absolute coordinate values and has no mechanism for the recognition of rotational symmetry. Therefore, identifying similarity in rotated models introduces a significant degree of complexity for the neural network.

The dataset contains a total of 5280 STEP files, divided randomly into train, validation and test sets with a ratio of 7:2:1. Each subset is balanced according to feature class.

3.2. Parameter Prediction Error

Table 1: Maximum absolute error in real parameter estimation for various percentiles

Percentile	Class	Size	X Centre	Y Centre
50%	0.20	0.10	0.10	0.10
60%	0.20	0.10	0.10	0.10
70%	0.30	0.20	0.20	0.20
80%	0.40	0.22	0.20	0.20
90%	0.60	0.40	0.30	0.30



(a) O-Ring (b) Rectangular Blind Step (c) Triangular Through Slot

(d) Hexagonal Through Hole (e) Vertical Circular End Blind Slot (f) Chamfer

Fig. 3: Examples of machining feature classes in the dataset

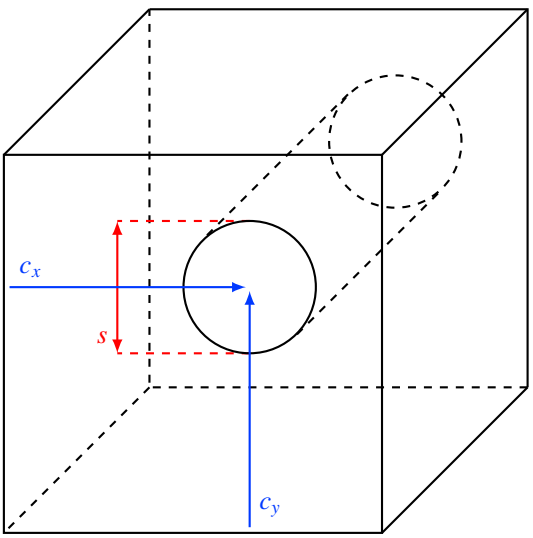
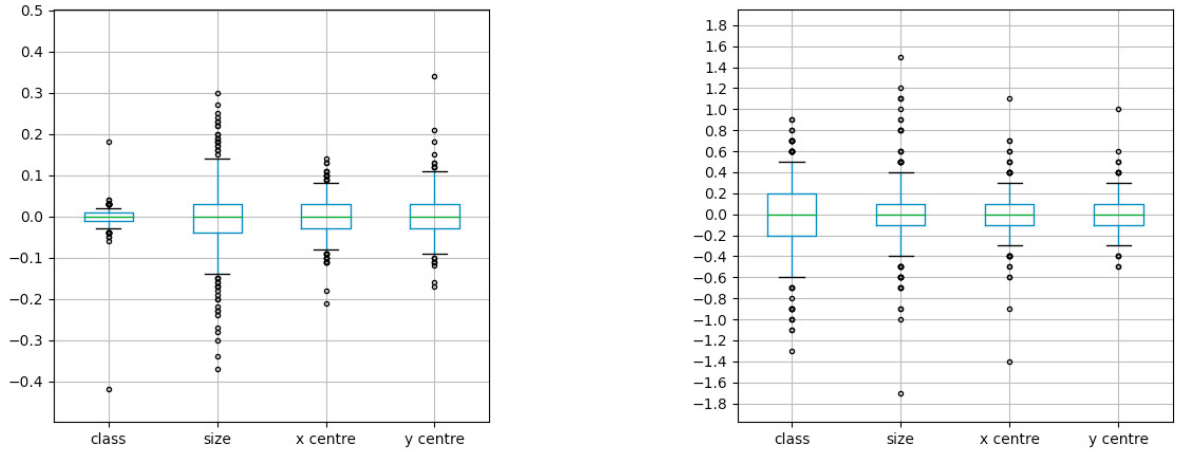


Fig. 4: Illustration of the geometric parameters used to define features, using the circular through hole class as an example; c_x and c_y are coordinates of the feature centre, s is the size of the feature when viewed from the front face.



(a) Normalised

(b) Real values

Fig. 5: Box plots showing normalised and real error when predicting each parameter, with 0 representing a perfect prediction; whiskers represent the 95th and 5th percentiles, with points outside these ranges plotted individually

Figure 5 shows the error distribution when predicting each of the 4 parameters across the test dataset. Figure 5a shows the error in direct neural network output, which is normalised between 0 and 1 for each parameter and Figure 5b shows the error in real predicted value, after the output has been denormalised.

Table 1 shows the absolute denormalised error values for the 50th to 90th percentiles, indicating the maximum error with which each parameter will be predicted for a given percentage of models. For instance, the size parameter is predicted with absolute error lower than 0.22 for 80% of CAD models in the test dataset.

From Figure 5a, it can be seen that the neural network is best able to accurately predict the value representing the feature class. This is an unsurprising result, given that feature class is the parameter which will produce by far the largest change in the input STEP file when changed, meaning that models of different feature class should be trivial for the neural network to distinguish. Interestingly, however, when converting the output of the decoder into actual predicted values, performance for class, size, x and y coordinates become more similar, with the model capable of predicting a value with absolute error less than 0.2 60% of the time for feature class and 70% of the time for size and coordinate values.

For actual, denormalised predictions, the most successfully predicted parameters are the coordinate values, both with error less than 0.3 for 90% of inputs. The size parameter shows similar performance in the lower percentiles but increased error in the higher percentiles, implying the existence of some more difficult edge cases.

4. Conclusions

This paper presents a novel asymmetric autoencoder architecture based on a recursive encoder and fully-connected decoder, for the reproduction of 3D CAD models from STEP files, by predicting the geometric parameters used for model generation. It is demonstrated that geometric parameters, such as size and position, can be estimated from STEP files within reasonable error bounds. These successfully predicted parameters indicate the effective encoding of not only 3D shape but also specific geometric parameters into the single-vector representation produced as output by the recursive encoder network.

This paper represents an initial test in the development of search engine tools for CAD model databases, with potential future work outside the scope of this paper including detailed investigation into how well the geometric information represented in the encoded vectors can be exploited in reliable search functionality and the application of this method to more complex, realistic data.

Acknowledgements

This work has used Durham University's NCC cluster. NCC has been purchased through Durham University's strategic investment funds, and is installed and maintained by the Department of Computer Science.

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) [grant number EP/T518001/1]. For the purpose of open access, the author has applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising. The authors have no competing interests to declare.

References

[1] Su, Hang and Maji, Subhransu and Kalogerakis, Evangelos and Learned-Miller, Erik (2015) "Multi-View Convolutional Neural Networks for 3D Shape Recognition" *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*

[2] Wu, Zhirong and Song, Shuran and Khosla, Aditya and Yu, Fisher and Zhang, Linguang and Tang, Xiaoou and Xiao, Jianxiong (2015) "3D ShapeNets: A Deep Representation for Volumetric Shapes" *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*

[3] Qi, Charles R. and Su, Hao and Mo, Kaichun and Guibas, Leonidas J. (2017) "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation" *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*

[4] Bank, Dor and Koenigstein, Noam and Giryes, Raja (2021) "Autoencoders" *arXiv preprint arXiv:2003.05991*

[5] Gogoi, Munmi and Begum, Shahin Ara (2017) "Image Classification Using Deep Autoencoders" *2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*

[6] Semeniuta, Stanislau and Severyn, Aliaksei and Barth, Erhardt (2017) "A Hybrid Convolutional Variational Autoencoder for Text Generation" *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*

[7] Wang, Wei and Huang, Yan and Wang, Yizhou and Wang, Liang (2014) "Generalized Autoencoder: A Neural Network Framework for Dimensionality Reduction" *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*

[8] Miles, Victoria and Giani, Stefano and Vogt, Oliver. (2022) "Recursive encoder network for the automatic analysis of STEP files" *Journal of Intelligent Manufacturing* **34** : 181-196.

[9] Sheng Tai, Kai and Socher, Richard and Manning, Christopher D. (2015) "Improved semantic representations from tree-structured long short-term memory networks" *Proceedings of the Association for Computational Linguistics (ACL)* .

[10] Miles, Victoria and Giani, Stefano and Vogt, Oliver. (2023) "Approaching STEP file analysis as a language processing task: A robust and scale-invariant solution for machining feature recognition" *Journal of Computational and Applied Mathematics* **427** : 115166.

[11] Kingma, D.P. and Ba, J.L. (2015) "Adam: A Method for Stochastic Optimization"*Proceedings of the 3rd International Conference for Learning Representations*