Journal of Intelligent Manufacturing (2023) 34:181–196
https://doi.org/10.1007/s10845-022-01998-x

Recursive encoder network for the automatic analysis of STEP files

Victoria Miles[1] · **Stefano Giani**[1] · **Oliver Vogt**[1]

Received: 13 December 2021 / Accepted: 15 July 2022 / Published online: 7 August 2022

Abstract
Automated tools which can understand and interface with CAD (computer-aided design) models are of significant research interest due to the potential for improving efficiency in manufacturing processes. At present, most research into the use of artificial intelligence to interpret three-dimensional data takes input in the form of multiple two-dimensional images of the object or in the form of three-dimensional grids of voxels. The transformation of the input data necessary for these approaches inevitably leads to some loss of information and limitations of resolution. Existing research into the direct analysis of model files in STEP (standard for the exchange of product data) format tends to follow a rules-based approach to analyse models of a certain type, resulting in algorithms without the benefits of flexibility and complex understanding which artificial intelligence can provide. In this paper, a novel recursive encoder network for the automatic analysis of STEP files is presented. The encoder network is a flexible model with the potential for adaptation to a wide range of tasks and finetuning for specific CAD model datasets. Performance is evaluated using a machining feature classification task, with results showing accuracy approaching 100% and training time comparable to that of existing multi-view and voxel-based solutions without the need for a GPU.

Keywords Artificial intelligence · Recursive neural network · Computer-aided design · STEP files

Introduction

The manufacturing industry is rapidly changing. With the increasing availability of intelligent systems, the industry is embracing smart solutions in all areas to move towards manufacturing processes which are flexible, high quality and cost effective in the modern world (Zhong et al., 2017). The starting point for all such processes is the design stage, and central to this stage is the production of a computer-aided design (CAD) model.

CAD models are used to represent complex geometries, which must then be replicated physically through manufacturing processes. To manufacture a part, the complex CAD model must be reduced to a series of manufacturing (or machining) features, such as slots or holes, which represent the processes necessary to reproduce the part. Much of the existing literature investigating intelligent analysis of CAD files focuses on the automatic recognition of machining features, with the goal of increasing automation in the transition from CAD model to manufacturing processes.

Intelligent systems which can make smart suggestions during the design process have the potential for further improvements in efficiency through additional applications. A smart solution which can categorise CAD models based on the model geometry and identify similarity between models could lead to increased standardisation of parts, as part designs with similarity to existing parts within a database could be flagged. This is increasingly relevant in an ever-changing market in which manufacture must be flexible and part lifetimes can be very short and so increased standardisation of simple parts could cause significant improvements in manufacturing efficiency. Solutions which can operate directly on model files are especially desirable due to the potential for time-saving early in the design process, before parts are combined to create complex 3D models, as well as increasing the simplicity of overall systems by not requiring transformation from model data to another form.

When discussing intelligent systems for the analysis of complex data, solutions fall into two primary categories: rules-based approaches and learning-based approaches.

✉ Victoria Miles
victoria.s.miles@durham.ac.uk

Stefano Giani
stefano.giani@durham.ac.uk

Oliver Vogt
oliver.vogt@durham.ac.uk

[1] Department of Engineering, Durham University, Stockton Road, Durham DH1 3LE, UK

In a rules-based approach, detailed understanding of the data is necessary, as rules dictating how specific features are to be extracted and used by the model must be written. These systems require a high level of expertise and a large amount of manual input from the designer and are generally difficult to adapt to new tasks or datasets.

In a learning-based (or deep learning) approach, the designer does not intentionally code specific behaviour into the model. Instead, neural networks are built, comprised of algorithms containing matrices of weight values, which are randomly initialised then continually updated during a training period with the goal of converging towards a final solution with the desired behaviour. As the sets of learned weights can be large, multi-dimensional matrices, and a neural network can contain many layers of operations, this can result in highly complex behaviour, without the need for manually coding an exhaustive set of rules. Neural networks can also easily be adapted by simply retraining the network for a new task or dataset, to learn weights which will better suit the new application. The advantages of using a learning-based approach are many; deep learning models are flexible, do not require expert level knowledge of the data to design or adapt and have been proven to perform well at complex tasks such as image classification (Krizhevsky et al., 2012) and machine translation (Cho et al., 2014).

Existing work on the application of deep learning to 3D models can be categorised based on the form of input data used. These forms include multiple 2-dimensional images of the model from different angles, volumetric 'voxel' based representation and mesh data. Each of these approaches has some advantages but all involve some degree of transformation from the original data format when analysing CAD models. Work which does focus on analysis of model files directly tends to utilise rules-based techniques to extract features from the models and so tends to be tailored towards very specific applications.

One approach which has not yet been widely explored is the direct application of a neural network to a STEP (Standard for the Exchange of Product Data) file. The STEP file format is an ISO standard (ISO, 2016), supported by all major CAD software. STEP files are simple text files which define a model architecture by using coordinates to define points and directions and using these to define lines and curves which are in turn used to define edges and faces and build up to an entire model geometry.

Related work

Automatic analysis of 3D data

Early attempts to apply neural networks to 3D data relied heavily on the related field of computer vision; the use of artificial intelligence solutions for the understanding of image data. With the introduction of the convolutional neural network, or CNN (Krizhevsky et al., 2012), 2D computer vision became the largest research area within the field of AI, meaning availability of high performance, pre-trained networks which could easily be adapted to different applications. Therefore, the most obvious approach to analysis of 3D data involves taking advantage of these existing resources by converting the 3D data into 2D data.

An early attempt to apply a neural network to a 3D model in this manner was presented by Qin et al. (2014), in which multiple two-dimensional images of a model are used as input to the deep learning classifier. This approach aims to mimic the process through which an engineer would identify a CAD model, by viewing the model at multiple angles to build up an understanding of the overall geometry. Su et al. (2015) present *multi-view CNN*, in which a view-pooling layer is used to intelligently combine information from multiple views into one classifier.

The second traditional approach to analysing 3D data is the use of 3D occupancy grids containing arrays of 'voxels' (the 3D equivalent of a pixel). This approach also makes use of existing research into computer vision, by adapting regular 2D CNN networks into 3D CNNs by simply adding an extra dimension and taking a 3D grid of voxels as input rather than a 2D grid of pixels.

A voxel-based approach was proposed with 3D ShapeNets (Wu et al., 2015), in which 3D models are represented by binary grids of voxels and classified using a convolutional deep belief network. In the work of Maturana and Scherer (2015), voxelised models are classified using a 3D CNN. In both papers the input occupancy grid is of size $30 \times 30 \times 30$. This resolution is a major limitation of the voxel approach; 30 pixels in each dimension is not sufficient to accurately represent complex geometries but higher resolution would result in impractically large model sizes. Riegler et al. (2017) addressed this issue with the introduction of *OctNet*, which partitions the 3D space into unbalanced octrees, making use of the natural sparsity of the input data and allowing for higher-resolution representation of model geometries.

Of these two traditional approaches, the multi-view approach tends to be more successful. The voxel-based approach, while seemingly a logical method, has persistent and severe restrictions due to the issues of data resolution and network size, which can be addressed to an extent but will always impose limitations on performance. The multi-view approach lacks some of these limitations. However, when considering analysis of CAD models rather than general shape recognition, some issues are clear. Firstly, the multi-view network's reliance on image data means that features of the model must be visible in an image in order for the network to recognise them. It is logical to conclude that details which appear on the inside of models or are small compared

to the model size are likely to cause problems as all actual 3D data is lost and the network must rely on images. Furthermore, careful consideration must be given to the set of 2D images which are selected as network inputs. Multiview networks take multiple 2D images and combine the information from all of them to build up an idea of the 3D shape but it is not possible to include images taken from every possible angle and, even using a sophisticated algorithm to make this decision, there is always a possibility that a crucial angle for a given model, in which certain features are visible, will be missed.

The potential advantages of developing a network which can work directly on model data are significant. The issue of resolution can be entirely eliminated if all model data is maintained and there is no chance of features in less visibly prominent positions being ignored. In short, no feature of the model, no matter how small, will be lost when directly working with model data.

More recent work has explored the possibility of working more directly on 3D data. For example, *MeshNet* (Feng et al., 2019) is a classification model which takes a mesh file as input. However, existing research into the direct use of model files, such as STEP files, as input tends to focus on rules-based approaches for performing specific tasks. Venu et al. (2018) recognise b-spline surface features using features extracted from a STEP file. Kiani and Saeed (2019) extract information about spot welding features. Both use rules-based approaches to extract the relevant features.

Al-wswasi and Ivanov (2019) present an approach for recognising features using STEP data as input, with the goal of developing integrated manufacturing systems with connections between CAD designs and manufacturing processes. This is a fairly thorough approach in which a large range of features is identified. However, the approach is still limited by a degree of rigidity, as the data is transformed away from the actual tree structure found in a STEP file and hard rules are relied on to analyse the geometric information.

No existing approach seeks to apply artificial intelligence techniques directly to data from a STEP file without first using rules-based techniques to extract features. This approach has the potential to result in a model with more flexibility, which can be trained to perform a variety of tasks without the necessity of coding a new set of rules for each application. It also avoids the common issue for image or voxel-based approaches of resolution, as in a STEP file enough detail is included to recreate all elements of a model, regardless of the scale of a feature.

Language processing

As STEP is a text format, the analysis of a STEP file can be approached as a language processing task. A major research area in the field of artificial intelligence, natural language processing (NLP) refers to the processing of text in human languages, such as English. Due to the prevalence of the field, there are numerous high-performing neural networks designed to perform NLP tasks which can be applied directly to processing artificial language, such as STEP data, with little or no adaptation. To the best of our knowledge, no existing research has investigated the possibilities of analysing STEP file data using language processing deep learning techniques.

In theory, artificial language processing tasks should be simpler for neural networks to perform than NLP tasks. Artificial language, unlike natural language, has no ambiguity and follows a stricter set of rules. It is not necessary to develop a complex understanding of language as humans use it, as the text being interpreted is already in a form designed to be comprehensible to computers.

Traditionally the field of NLP has been dominated by the use of recurrent neural networks (RNNs). An RNN can take a sequence of any length (such as a sentence) as input and repeatedly apply the same set of weights to each input in the sequence, with the output hidden state from each cell being applied as an input to the next cell in the sequence. The introduction of the long short term memory (LSTM) cell (Hochreiter & Schmidhuber, 1997) improved this architecture by introducing a system for storing long term information, addressing the vanishing gradient problem in which gradients become vanishingly small during training and so information from early in the sequence is lost by the time a final output is produced.

Another type of neural network, the recursive neural network, is similar to the recurrent neural network but with cells organised in a tree structure rather than as a sequence. A recursive neural network was first presented by Socher et al. (2011), with suggested applications including natural language parsing as well as semantic scene segmentation. The concept behind this research was that both images of scenes and natural language sentences have inherent structures in which simple components combine to form more complex structures and so both images and text data could be represented using a hierarchical (or tree) structure. The recursive neural network attempts to learn this structure and apply learned weights throughout the resulting data tree in order to interpret the sentence or image. In practical terms this makes the network somewhat more complex than an RNN as each cell of the network could, in theory, be fed by the output of any number of cells from the previous level of the tree, and the structure of the data tree must be learned by the network. An improved model for NLP was presented by Tai et al. (2015). This model replaced the simple cell with an LSTM cell, modified to take multiple inputs as necessary for a recursive network.

Although these networks have seen some success in the field of NLP, further research into the potential of recursive networks has been limited. This lack of active research can

likely be attributed to the continued success of RNN (Cho et al., 2014; Chung et al., 2015; Sutskever et al., 2014) and transformer (Devlin et al., 2019; Vaswani et al., 2017) models on NLP tasks. As recursive networks are more complex and were not seen to significantly outperform these other network types, they have seen limited popularity in recent years and, in the field of NLP, have been largely abandoned as an active branch of research. However, recursive networks have been suggested more recently for tasks outside of NLP. Chen et al. (2018) adapted the network proposed by Tai et al. (2015) for the task of translation between programming languages. In this case the input data was code, a form of artificial language with an inherent tree structure, meaning that the structure did not need to be learned by the network. The recursive network developed was shown to significantly outperform other existing solutions at the time of publication, an indication of the untapped potential of recursive networks for applications outside of NLP. However, such revivals of the recursive network remain rare and thus far no attempt has been made to apply a recursive network to the understanding of the data present in a 3D model file.

As shown in Fig. 1, STEP files have an inherent hierarchical structure; low level features are combined to form more complex features and so on until, at the top level of the structure, the entire model architecture is represented by a single node. This tree structure means that a STEP file input is perfectly matched to a recursive neural network. Moreover, the tree structure is explicit in the input data and so will not need to be learned by the network, making the operation of the recursive network significantly less complex.

Methodology

In this section a two-stage process for the automatic analysis of STEP files will be outlined:

Step 1: Design of a STEP parser which transforms raw input from a STEP file into a processed data tree which can be used as input to the recursive encoder. This involves creating a node for each relevant line from the file, adding extra nodes containing coordinate point information, encoding the overall structure of the tree and filtering out irrelevant nodes. The process is outlined in detail in the 'Parsing STEP Files' section. The effects of taking coordinate points to a lower precision or ignoring coordinate point data entirely at this stage is investigated, with the results presented in the 'Evaluation of the Significance of Coordinate Value Precision' section.

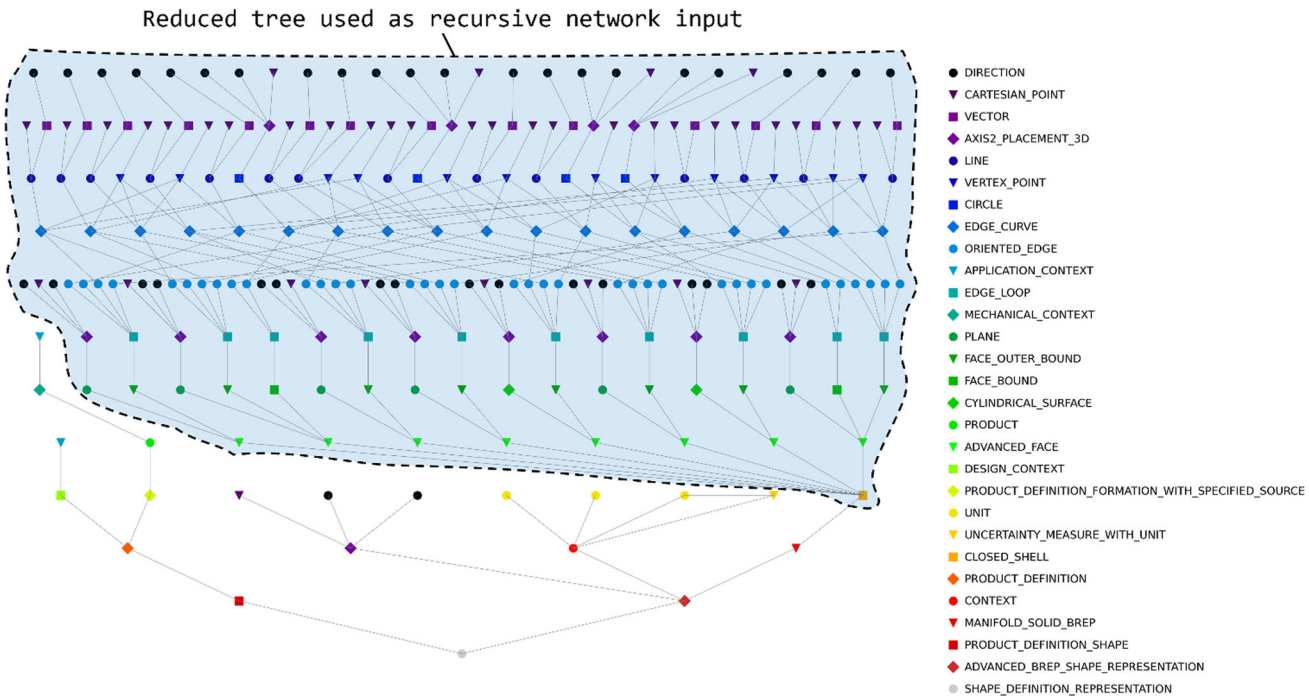


Fig. 1 Typical hierarchical structure representing the geometric data for a simple CAD model; connections show how low-level features are combined to form complex shape representations, the shaded region represents the portion of the tree which will be used as input for the neural network

Step 2: Development of a recursive encoder network for the analysis of processed data trees. The encoder network is based on the Child-Sum Tree-LSTM network first presented by Tai et al. (2015). It operates by working through the encoded tree structure, applying a set of learned weights at each node. This operation is described in detail in the 'Recursive Encoder Network' section. The performance of the recursive encoder network will be assessed on a machining feature classification task in order to demonstrate feasibility of the presented approach.

Parsing STEP files

For the recursive encoder to process a STEP file, all geometric information must first be extracted and converted into a useful form with an explicit tree structure. To this end a STEP parser is designed to process the STEP data and convert it into a hierarchical structure of vectors which can be used as network inputs.

Generation of hierarchical data structures

STEP files, even those defining very simple CAD models, contain hundreds of lines of information. An example segment of a STEP file is presented in Fig. 2. Each line is assigned a unique ID number, preceded by '#', which starts the line. The information contained in the line consists of a category and any number of parameters which may be the IDs of other lines, coordinate point values or additional flags. For example, in Fig. 2, the first line has ID number 1, category CARTESIAN_POINT and takes three coordinate values as parameters (− 0.4…, 0.17…, 0.00), representing a single 3D point. An example of a line taking another line as a parameter can be seen in the line with ID number 3 and category EDGE_CURVE, which takes lines with ID numbers 50, 156 and 220 as parameters.

The data lines within a STEP file may be organised in any order and lines may take other lines which appear higher or lower as parameters. As a result, applying a language processing network, such as an RNN or transformer network, directly to STEP file data is not feasible, as such networks rely on the sequence of input information conveying meaning. Instead, the inherent hierarchical structure of the input data has been exploited to generate the input for a recursive network. The STEP file is used to produce a tree structure, with each line from the STEP file represented by a single node and connections between parent and child nodes specified, where the children of a node are the other lines given as parameters in the STEP line. An example of the tree structure containing all geometric information for a simple STEP file is presented in Fig. 1, where each line of the file is represented by a single node and direct connections are shown between parent and child nodes.

Converting a STEP file into a useful hierarchical structure compatible with a recursive network is a two-step process:

Step 1: Convert each data line from the STEP file into an instance of the node class, represented using the node's ID number and category.
Step 2: Attach a list of child nodes to each node in the tree. Parse through the tree structure, updating all nodes to have a record of parent nodes as well as child nodes.

Additional consideration must be given to the method of encoding coordinate point values into the data tree as several forms of representation are feasible. The process chosen will be discussed in detail in the 'Incorporating Coordinate Values into the Tree' section.

Filtering out irrelevant information

Not every line in a STEP file contains useful geometric information, as can be seen in Fig. 2. Many nodes, such as the

```
#1 = CARTESIAN_POINT ( 'NONE',  ( -0.4146254658699035645, 0.1761599481105804443,
0.000000000000000000 ) ) ;
#2 = PERSON_AND_ORGANIZATION ( #307, #296 ) ;
#3 = EDGE_CURVE ( 'NONE', #50, #156, #220, .T. ) ;
#4 = ORIENTED_EDGE ( 'NONE', *, *, #17, .T. ) ;
#5 = APPROVAL_PERSON_ORGANIZATION ( #300, #169, #100 ) ;
#6 = CARTESIAN_POINT ( 'NONE',  ( -0.4146254658699035645,
-0.1761599481105804443, 0.000000000000000000 ) ) ;
#7 = EDGE_LOOP ( 'NONE', ( #179, #131, #83, #190 ) ) ;
#8 = PLANE ( 'NONE',  #278 ) ;
#9 = CC_DESIGN_PERSON_AND_ORGANIZATION_ASSIGNMENT ( #128, #77, ( #139 ) ) ;
#10 = DIRECTION ( 'NONE',  ( -1.000000000000000000, -0.000000000000000000,
-0.000000000000000000 ) ) ;
```

Fig. 2 Sample data lines from a STEP file; lines which do not contain geometric information are highlighted

'PERSON_AND_ORGANISATION' node, will not connect to the data tree representing the geometry as they contain information irrelevant to the model itself. Other nodes, such as the 'UNIT' node will connect to the tree as they do contain information necessary to recreate the CAD model, but they are not necessary for the purposes of analysing the shape of the model and features present.

Therefore, it is necessary to filter out any nodes which do not connect directly to the geometric data tree and desirable to trim the tree to leave the simplest possible structure which still contains all the geometric information necessary to recreate the model.

This is achieved by identifying an appropriate node which will be the singular highest-level node of the data tree and removing any nodes which this top-node does not directly depend on. It can be seen in Fig. 1 that all lower-level geometric nodes feed directly into the 'CLOSED_SHELL' node and so, for the model represented by this diagram, 'CLOSED_SHELL' is an appropriate top-node. The resulting reduced data tree is shown in the shaded region of Fig. 1. Throughout the work presented in this paper, the choice was made to set 'CLOSED_SHELL' as the top node in order to keep the model trees as simple as possible.

Vector encoding of node categories

Data inputs for a neural network must take the form of vectors or matrices of values which the network can process. In the case of language processing networks, this means that a vocabulary of vector terms must be defined to represent each word which could be present in the raw data. In the case of our network the 'words' in the data are the node categories taken from the STEP lines, such as 'CARTESIAN_POINT' or 'EDGE_CURVE'. To define a fixed vocabulary for the network, each unique category must be assigned a unique vector representation.

To encode categories into vectors, one-hot vector encoding is utilised. This method converts categories into vectors with length equal to the number of categories in the dataset. Each unique category in the dictionary is assigned an index between zero and the dictionary size. To encode a category as an input, every value in the vector is set to zero, except for the value at the index assigned to the relevant category which is set to one. As the input vectors will have length equal to the dictionary size, it is desirable to keep the dictionary of node categories as small as possible, as keeping the input vectors small will minimise the model size and limit the number of computations necessary.

Incorporating coordinate values into the tree

One possible approach to incorporating coordinate point values into the tree would be to represent each unique coordinate value seen in the STEP file as a new instance of the node class, representing the entire numerical value. However, while this would be a functional method of including the values, it would be problematic for two key reasons. Firstly, adding a new node category for every different unique number would greatly increase the dictionary size, leading to larger input vectors and a larger model overall, or potentially leading to a situation where one-hot vector encoding is no longer appropriate and more complex encoding is necessary. This would also mean that the majority of categories in the dictionary would represent different coordinate points, potentially exaggerating their importance to the network. The second issue with this method is that the numerical values themselves would not convey any meaningful information to the neural network, other than which numbers are exactly the same. In order to aid the encoder in developing an understanding of the data it is desirable to incorporate coordinate points in a form which provides context about relative similarity of values.

For the above-mentioned reasons, it was decided to represent coordinate points using strings of nodes, each representing a single digit of the coordinate value. The first digit in each string is either $+$ or $-$ followed by a node for each digit, with each node taking the node representing the previous digit as a child and the node representing the subsequent digit as a parent. The final digit in the string is set as the child of any STEP nodes with that coordinate value as a parameter, with each unique numerical string processed by the network only once and the outputs reused wherever that value appears in the data tree. An example of the node string structure is presented in Fig. 3, using the first coordinate value seen in Fig. 2.

This method of including coordinate values limits the dictionary size as only 13 new node categories need to be added in order to encode any numerical value. It also provides more context to the numerical values themselves as the model should be able to identify similarity between numbers which start with the same digits. Another advantage of this approach is that the precision at which coordinate points are taken can be adjusted. Reducing the number of decimal places to take the value to will result in a data tree with less levels and so speed up computations. Increasing the number of decimal places allows for smaller features to be identified reliably.

The final size of the dictionary, including values for coordinate points, is 32, meaning that inputs to the neural network will be one-hot vectors of length 32.

The full list of node classes present in the dataset is included in the Appendix.

Recursive encoder network

In this section a recursive encoder network is proposed for the automatic analysis of data trees generated by the STEP parser.

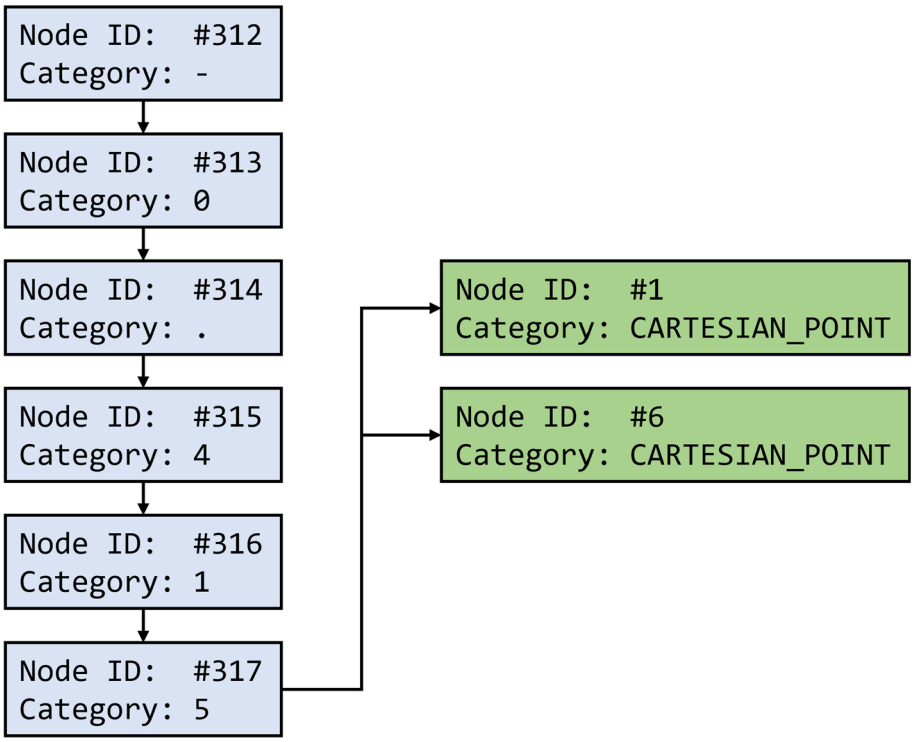


Fig. 3 The node string produced to represent the value − 0.415, the first coordinate value seen in Fig. 2 when taken to three decimal places, with parent–child connections shown to the two lines in Fig. 2 which reference this value

In order to demonstrate the performance of the encoder, a simple classifier is applied to the output, and the resulting network trained to perform a feature classification task.

The recursive encoder makes use of the Child-Sum Tree-LSTM network first presented by Tai et al. (2015). In this network, a modified LSTM cell is applied to inputs organised in a tree structure. The term recursive refers to the repeated application of the same processes, in this case the repeated application of an identical treeLSTM cell to every node in the tree structure. The input tree structure could take any shape and have any number of nodes, each with any number of children. The one constraint on the shape of the tree is that there will always be one single top-level node, representing the input data as a whole. The recursive encoder itself does not have a fixed structure; in a way, it consists of a single cell containing several sets of learned weights which will be applied repeatedly to each new input as the network works through the data tree from leaf (the lowest-level information e.g. coordinate values) to root (the single top-level node, in this case 'CLOSED_SHELL'). Figure 4 shows how the LSTM cell might be applied to a simple data tree. As the LSTM cell applied to each node is identical, the neural network is able to adapt to fit the shape of any data tree it is applied to and the network is kept small as the same sets of learned weights are applied at every node in the tree, limiting the total number of learned parameters by ensuring that the model size remains the same regardless of how complex the input data tree is.

The network as proposed by Tai et al. (2015) learns not only the weights necessary to analyse the input tree but also those necessary to determine the correct structure of the tree itself. This is because the focus of the research is on natural language processing and, while natural language does have inherent structure, this structure is not specifically codified in the text and so must be determined. In contrast, the STEP file format is an example of artificial language, in which the structure is explicitly codified and therefore does not need to be learned. This makes the encoder's task of learning a meaningful representation of the data significantly less complex as the structure of the data can be defined in a data processing step and does not need to be decided by the neural network.

An LSTM cell uses three gates, each with a set of learned weights, to maintain a long-term memory state known as the cell state, in addition to the short-term memory carried from the previous cell, known as the hidden state. The forget gate controls which information carried from the previous cell state should be kept and which should be forgotten, based on the previous hidden state and the current input. The input gate controls how much information from the previous hidden state and the current input should be written to the cell state. The output gate controls what information should be output to the next hidden state, which is also the output of the cell. Each of the three gates has an associated weight matrix; these same weights are repeatedly applied for each new input the network encounters. During training, the values in the weight matrices are continuously updated with the goal of learning

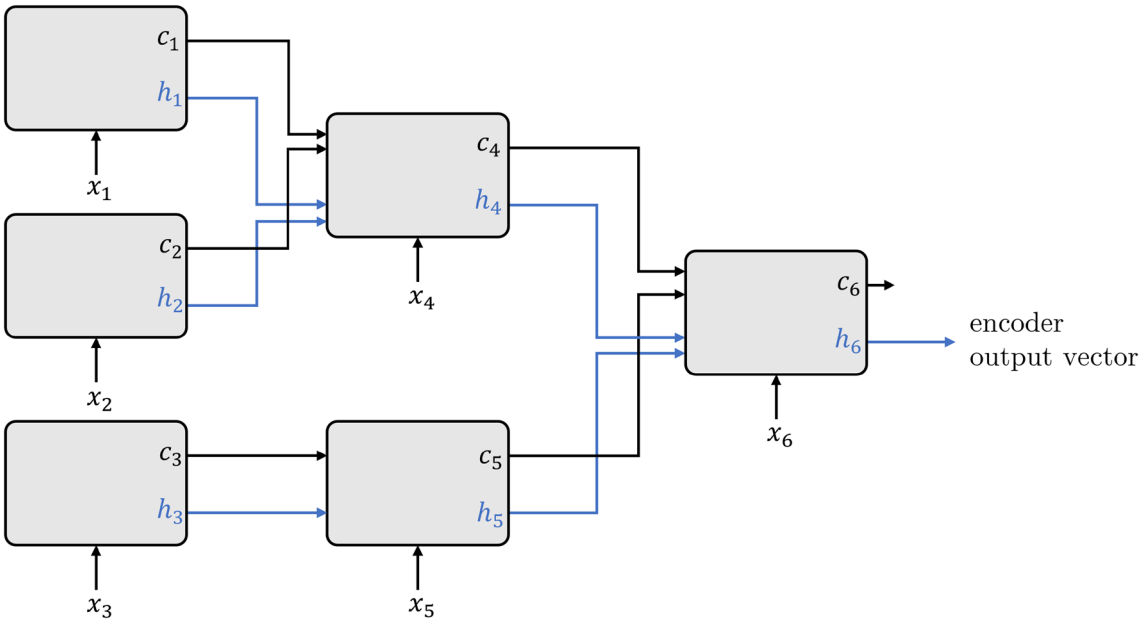


Fig. 4 Example application of the treeLSTM cell to a simple tree structure, where c is the output cell state, h the output hidden state and x the new input for each node

sets of weights which produce meaningful outputs for any given input.

The key difference between a standard LSTM cell and a treeLSTM cell is the number of potential previous states. As a standard LSTM cell is designed to be part of a linear string of cells, only one input hidden state and one input cell state is necessary whereas the treeLSTM cell must be able to take multiple previous states, as a node in the tree can have any number of children. Figure 4 shows how the output states for child nodes are passed to the input of parent nodes for a very simple structure. A node having multiple children, and so taking multiple previous states as input, results in the need for multiple forget gates so the information from each of the previous cells can be separately considered to be kept or removed. Once the forget gate outputs have been calculated, the sum of the previous hidden states is taken, allowing the input and output gates to function as with a standard LSTM cell. Although it is possible to have any number of previous states, each cell has only one output cell state and one output hidden state, both of fixed length. The hidden state is taken as the output of the cell. Figure 5 shows the processes within a treeLSTM cell.

The equations used to govern the LSTM cell are as follows, where the children of node j are denoted as $C(j)$, the input gate is i_j, forget gates are f_{jk}, output gate is o_j, output cell state is c_j, output hidden state is h_j and the input vector at node j is denoted by x_j:

$$\widetilde{h_j} = \sum_{k \in C(j)} h_k \tag{1}$$

$$i_j = \sigma\left(W^{(i)} x_j + U^{(i)} \widetilde{h_j} + b^{(i)}\right) \tag{2}$$

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

$$o_j = \sigma\left(W^{(o)} x_j + U^{(o)} \widetilde{h_j} + b^{(o)}\right) \tag{4}$$

$$u_j = \tanh\left(W^{(u)} x_j + U^{(u)} \widetilde{h_j} + b^{(u)}\right) \tag{5}$$

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

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

W and U represent the sets of learned weights which are applied to the input vector and the previous hidden state respectively for each gate at every node of the tree and b is a set of learned bias terms, used to shift the activation function outputs to better fit the data. $\odot$ refers to the operation of element-wise multiplication.

The hidden size of the encoder was set to 300, meaning that the output cell and hidden states from the LSTM cell are vectors of length 300. This value was selected to be large enough for sufficient complexity but to be as small as possible to limit the number of parameters in the network. As the top level of the data tree consists of a single node, the final encoder output for a single STEP file is the output hidden state for this node, a single vector of length 300. The dimensions of inputs, outputs and learned parameters are given in Table 1. The total number of learned parameters for the encoder is

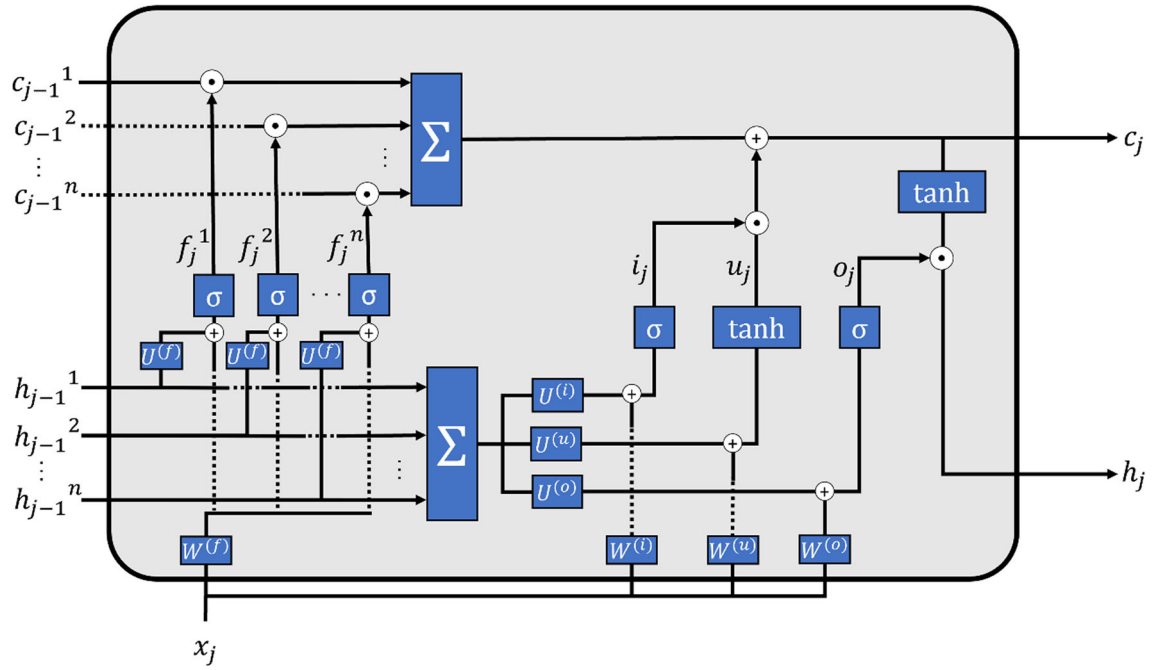


Fig. 5 The processes within a treeLSTM cell; c is cell state, h is hidden state, x is input vector, U and W are weights and f, i and o are forget, input and output gates respectively

Table 1 The dimensions of parameters within the recursive encoder cell

Parameter notation	Function	Shape
c_j, h_j	Cell memory states	(300)
x_j	Input vector	(32)
$W^{(i)}$, $W^{(f)}$, $W^{(o)}$, $W^{(u)}$	Learned weights	(32, 300)
$U^{(i)}$, $U^{(f)}$, $U^{(o)}$, $U^{(u)}$	Learned weights	(300, 300)
$b^{(i)}$, $b^{(f)}$, $b^{(o)}$, $b^{(u)}$	Learned bias terms	(300)

less than 400,000.

The recursive encoder network extracts features from the data, in the form of an output vector of length 300 representing each input data tree. Since we want to consider model performance for a classification task, this vector must be used to produce a single class prediction for each CAD model.

To this end, the output vector is fed into a fully connected layer, a neural network layer in which every possible connection between neurons is included; every output value will depend on the value of every input. The fully connected layer uses the entire output vector from the encoder network to produce a score for each class of feature present in the dataset, using the following equation:

$$y_i = \sum_{j=1}^{n} x_j w_{ij} + b_i \tag{8}$$

for encoder output of length n, where y_i is the score for class i, x is the input vector and w and b are the weights and bias terms for the layer. The output of the fully connected layer for a single input data tree is a vector with length equal to the number of feature classes, in this case 24, containing the score for each class.

To convert this vector of class scores into a vector of probabilities for each class, the Softmax function is applied as follows:

$$p_i = \frac{e^{y_i}}{\sum_{j=1}^{n} e^{y_j}} \tag{9}$$

for n classes, where p_i is the Softmax probability for class i and y is the output of the fully connected layer.

The network is trained with the objective of minimising cross-entropy loss, which is defined as:

$$L_{CE} = -\sum_{i=1}^{n} t_i \log(p_i) \tag{10}$$

for n classes, where t_i is the ground truth and p_i the Softmax probability for class i. The Adam optimisation algorithm (Kingma & Ba, 2015), an adaption of stochastic gradient descent, is used during training. The model is implemented using PyTorch.

Evaluation of the significance of coordinate value precision

To evaluate the significance of coordinate values and the effects of varying precision, three versions of the encoder model were trained to perform a feature classification task: (1) with all coordinate values replaced with a single placeholder token, (2) with all coordinate values taken to five decimal places and (3) with a mixed precision dataset: the number of decimal points in the coordinates was randomly set a value between one and five for each model in the dataset.

Machining features dataset

The dataset generated for the feature classification task is comprised of models of blocks, each with a single machining feature added. The features chosen were a list of common machining features, as suggested in Zhang et al. (2018). Examples of the 24 feature categories are shown in Fig. 6.

Using the benchmark dataset developed by Zhang et al. (2018) would be ideal. However, this was not possible due to the lack of available data in STEP format. Reconstructing model files from the mesh formats available would involve too much manual input to be feasible. Therefore, it was decided to generate an equivalent dataset to the benchmark containing the same number of randomly generated models in each category.

The base model the features are added to is a cuboid with the longest side equalling 1 unit. As units are ignored by the encoder network, this will normalise all coordinate values between 0 and 1. The machining features are added to a random face of the model, with parameters for size and position of the feature also assigned random values. This dataset is not entirely equivalent to the benchmark dataset as some increased randomisation was added to make the models as general as possible, such as using an irregular starting block instead of a cube, and allowing for both larger and smaller relative feature sizes.

The STEP dataset contains a total of 24,000 models, 1000 examples of each of 24 categories of machining feature. The dataset has been divided into train, test and validation subsets at a ratio of 7:2:1.

Experiment 1: how necessary are coordinate values?

In this experiment, the performance of the model trained without any coordinate points (model 1) is compared with that of the model trained using coordinates taken to five decimal places (model 2). In the case of model 1, the scale and location of features is unknown but the network still has access to geometric information such as number of edges and faces.

The confusion matrix for model 1 is shown in Fig. 7a. As can be seen, the model is capable of accurately identifying 22 out of the 24 classes based on shape alone.

The exceptions are the 'rectangular through slot' class which is consistently mislabelled as 'rectangular blind slot' and the 'slanted through step' class which is consistently mislabelled as 'rectangular through step'. These represent the two sets of two classes which cannot be reliably separated based on geometric components alone and which require additional spatial information to distinguish. The similarity between the rectangular and slanted through step classes is clear to see, as there is no difference between the classes when coordinate values are not considered. The rectangular through and blind slots, however, seem visibly distinct even when coordinates are not included. The reason these classes cannot be separated is because they consist of the same set of faces, simply resized and arranged differently; both consist of two u-shaped faces and eight rectangular faces. The orientation, position and scale of these faces differ between the two classes but without coordinate point values, this information is not represented in a STEP file, resulting in the two classes being confused.

The high accuracy for all other classes demonstrates the potential of the recursive encoder network; the validation accuracy across all classes is 91.67%. As spatial coordinates are not necessary to reliably separate most classes, feature size and resolution are not an issue for the network, meaning that the model can identify very small features relative to the model size with the same reliability as large features. This is a clear advantage when comparing to image or voxel-based approaches.

As can be seen in Fig. 7b, when allowing the model to use actual coordinate points, the remaining uncertainty between the two pairs of similar classes is removed and accuracy approaches 100%.

Experiment 2: the impact of varying coordinate precision

An important factor to consider in relation to the precision of the coordinate points in the data is whether the whole dataset contains coordinate points at the same precision or if the precision can vary. To investigate the significance of this, the performance of the fixed precision model (model 2) is compared with that of the variable precision model (model 3).

Figure 8 shows the accuracy of the two models when presented with varying precision datasets.

Model 2 (trained with a fixed precision) performed poorly when presented with a different precision dataset, regardless of whether more or less decimal places were included. In contrast, model 3 (the mixed precision model) was shown to perform consistently well, even when presented with data containing points taken to a much greater precision than was

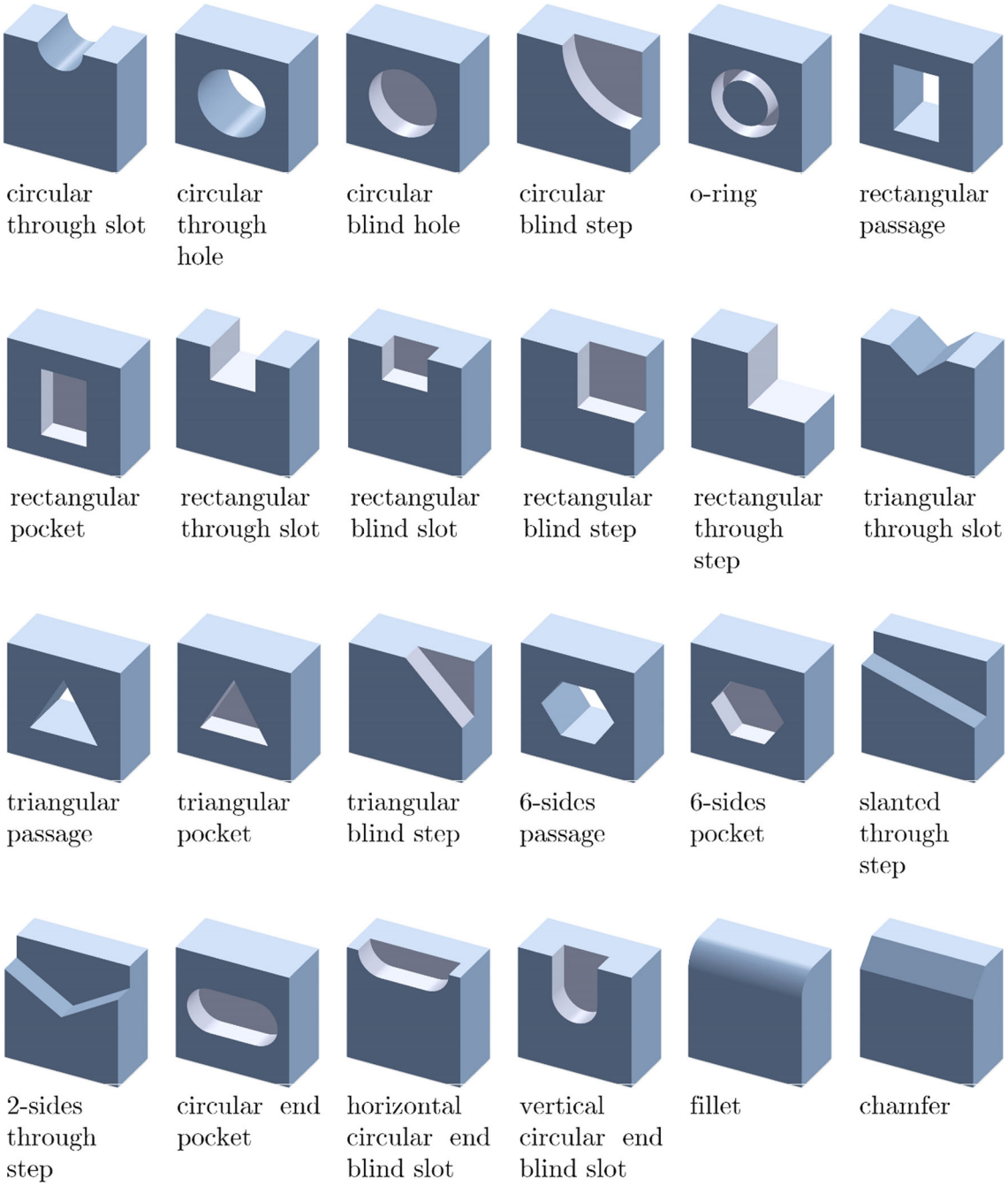


Fig. 6 Machining features present in the dataset, as suggested by Zhang et al. (2018)

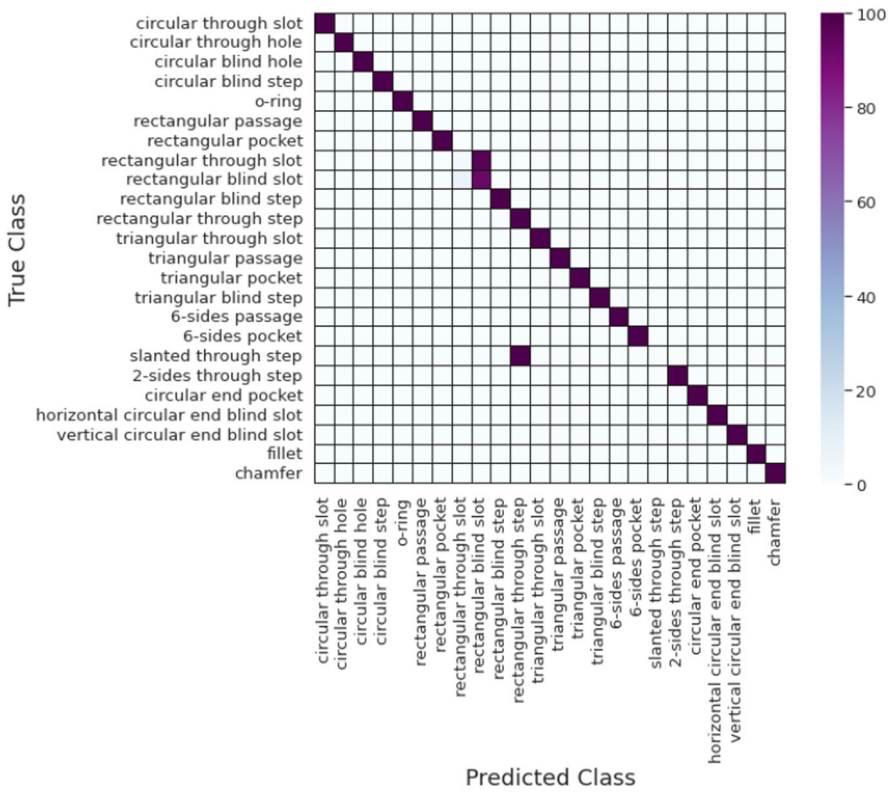
included in the training datasets. This indicates the robustness of the mixed precision model and the superiority of this approach.

The final accuracy of the mixed precision model on a validation set with precision matching that of the training set was 99.96% and when the precision of the input data was increased to a maximum of 10 decimal places, the model accuracy only dropped to 99.04%.

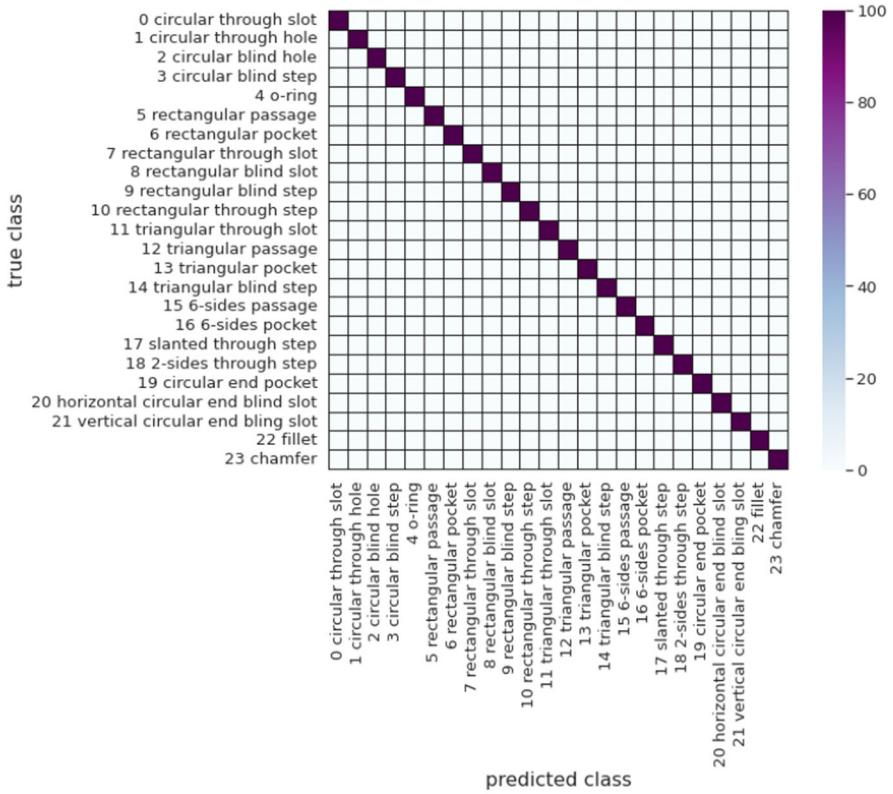
Model performance evaluation (results)

The recursive encoder-based network has been trained to perform classification for the 24 classes of machining feature shown in Fig. 6.

In this section, existing work on machining feature recognition will be described, and the performance of our network compared with two existing machining feature classifiers. Both comparison networks utilise traditional multi-view and



a Model 1: trained with coordinate points all represented by single 'COORD' token



b Model 2: trained using actual coordinate point values

Fig. 7 Confusion matrices showing number of occurrences of predicted class against ground truth when evaluating using the validation dataset. **a** Model 1: trained with coordinate points all represented by single 'COORD' token. **b** Model 2: trained using actual coordinate point values

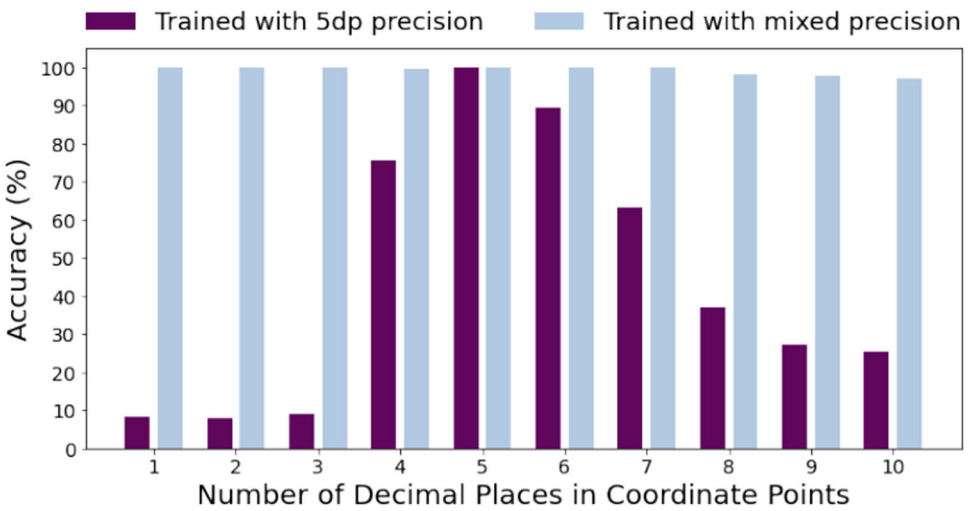


Fig. 8 Validation accuracy against precision of coordinate values in the dataset for model 2 (trained with 5dp precision) and model 3 (trained with mixed precision)

voxel-based approaches, meaning that they do not work directly on model files and require a GPU for fast performance.

Details on existing solutions

FeatureNet

The machining features dataset reproduced for this work was first presented by Zhang et al. (2018) as the dataset used to train *FeatureNet*, a 3D CNN-based classification model.

In this work, models from the machining features dataset are converted into normalised binary-valued voxel grids of maximum size 64 × 64 × 64. These voxelised models are used as input for a 3D CNN, consisting of four convolutional layers, followed by a max pooling layer and a fully connected layer, with a Softmax function producing the final output class probabilities for each feature category. The total number of learned parameters for this model is around 34 million, several orders of magnitudes larger than our model of around 400,000 parameters.

A key factor in the performance of FeatureNet is the resolution of the voxelised models. The highest resolution used (64 × 64 × 64 voxels) results in models with higher accuracy but which are slow to train and run whereas the lowest resolution (16 × 16 × 16 voxels) results in a smaller, faster network but with the cost of a significant loss in accuracy.

MsvNet

In *MsvNet* (Shi et al, 2020), a multi-view approach is applied to feature recognition. 3D models are represented using 12 images, resized to a maximum of 64 × 64 pixels, created by making random cuts into the model to obtain random sectional views, with the goal of including information on the interior of models.

To perform feature classification using the 2D sectional views, a VGG-11 model (Simonyan & Zisserman, 2015) is adapted, a 2D CNN consisting of eight convolutional layers and three fully connected layers and pre-trained on the ImageNet database. In MsvNet, the CNN is first fine-tuned to the dataset by performing classification based on a single sectional view, before incorporating a view-pooling layer between the convolutional and fully connected layers to combine information from multiple sectional views.

In the paper presenting this network, an extremely thorough investigation of model performance at the machining feature classification task is carried out. Results are presented for varied resolution of input data and number of training samples, with comparison to FeatureNet included.

Performance comparison

In this section, the performance of our network for feature classification will be compared to the optimal performance reported of the two comparison networks.

Our recursive network is trained with batch size set to 32 and learning rate initialised at 0.001. A full training period consists of 10 epochs of training. Results presented are those for the model trained using variable precision of coordinate points, as outlined in the 'Evaluation of the Significance of Coordinate Point Precision' section.

In Table 2, accuracy and training time for our final network is presented, with results from FeatureNet and MsvNet included for comparison. All computations for our network were carried out using a PC with Intel Core i5-9500 CPU and no GPU. Comparison data is as reported for optimal performance of the networks by Shi et al. (2020), in which computations were carried out on a PC with Intel Core i9-9900X CPU and NVIDIA GeForce RTX 2080 TI GPU.

Table 2 Validation accuracy and training time comparison with existing solutions as reported by Shi et al. (2020)

	FeatureNet (GPU) Zhang et al. (2018)		MsvNet (GPU) Shi et al. (2020)		Ours (CPU)
Resolution	16 × 16 × 16	64 × 64 × 64	16 × 16 × 16	64 × 64 × 64	N/A
Accuracy	91.91%	98.17%	94.88%	99.67%	99.96%
Training time	78.70 min	2139.55 min	712.35 min	871.23 min	699.04 min
Number of training models per class	4096	4096	4096	4096	700

As the two comparison networks work at variable resolutions the highest and lowest voxel resolutions reported are included for comparison.

Discussion and future work

In the previous sections, the development of a recursive encoder network for the analysis of 3D CAD models was described, and the process of training the network and measuring performance at a machining feature classification task outlined. Details on the performance of our network, compared with those of two other machining feature recognition models are presented in Table 2.

Our network is capable of outperforming both comparison networks in terms of accuracy, regardless of resolution of the other networks. It also trained faster than all but the lowest performing comparison network without the need for a GPU. This is a key result as the intention is to develop a versatile network which can be easily adapted to new tasks and retrained for operation using different types of data as a more flexible alternative to rules-based systems. A fast training time on CPU indicates the potential as a network which can be very easily adapted and trained for a new application using any hardware.

While the accuracy achieved by our network is only a slight increase from the already very high performance of existing solutions, there are two key considerations which make this result significant.

Firstly, the highest accuracy achieved by MsvNet was dependent on using the highest resolution of the network and based on a dataset in which feature size was fixed to not be extremely small compared to the model size. In contrast, our model has been shown to perform well across 22 out of 24 classes even when no spatial information is included, indicating that the high performance is not dependent on feature scale and has no limitations based on resolution.

Secondly, both FeatureNet and MsvNet represent the application of existing artificial intelligence techniques for the analysis of 3D data to the machining feature recognition task. Our network, on the other hand, represents an entirely new approach to the analysis of CAD models, with a focus on effectively utilising all geometric information by interfacing directly with model files, and the machining feature classification task was merely selected as an appropriate initial test of the feasibility of the encoder network. Outperforming existing solutions, even by a small margin demonstrates the potential of the encoder as a unique approach to 3D model analysis, using artificial intelligence techniques directly applied to model data.

The recursive encoder network developed in this work is designed to have the flexibility to be adapted to a wide range of tasks relating to the analysis of CAD models. The next step in this research, therefore, is to design and train models incorporating the encoder for more complex tasks than single-feature classification, such as multi-feature recognition and assessment of general similarity in more complex CAD models.

Conclusions

This paper presents a novel recursive encoder network for the automatic analysis of STEP files without the need for rules-based feature extraction. The performance of the encoder was evaluated based on the task of machining feature classification.

It was found that the classification network was capable of recognising 22 out of 24 simple machining features and achieving overall accuracy of 91.67% when the network was trained using only basic geometric components without any spatial information. When spatial information was included the classification accuracy of the network reached 99.96% across all 24 classes of feature.

As the encoder network developed is not reliant on any hard rules to extract features from STEP files, it has the potential to be adapted for a wide range of tasks relating to shape recognition and fine-tuned to work across any specific dataset of model files. As there is no translation of the model data into alternate forms, excepting some basic data processing of the input files, the network is suitable for direct implementation during the design process.

Funding This study was funded by the Engineering and Physical Sciences Research Council (EPSRC).

Declarations

Conflict of interest The authors have no competing interests to declare that are relevant to the content of this article.

Appendix: List of node classes

The full list of node classes present in the dataset is presented in Table 3.

Table 3 Full list of node classes present in the dataset

Node class
*
+
−
.
.F
.T
0
1
2
3
4
5
6
7
8
9
ADVANCED_FACE
AXIS2_PLACEMENT_3D
CARTESIAN_POINT
CIRCLE
CLOSED_SHELL
CYLINDRICAL_SURFACE
DIRECTION
EDGE_CURVE
EDGE_LOOP
FACE_BOUND
FACE_OUTER_BOUND
LINE
ORIENTED_EDGE
PLANE
VECTOR
VERTEX_POINT

References

Al-wswasi, M., & Ivanov, A. (2019). A novel and smart interactive feature recognition system for rotational parts using a step file. *The International Journal of Advanced Manufacturing Technology, 104*, 261–284.

Chen, X., Liu, C., & Song, D. (2018) Tree-to-tree neural networks for program translation. In: *Proceedings of the 32nd international conference on neural information processing systems* (pp. 2552–2562)

Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014) On the properties of neural machine translation: Encoder–decoder approaches. In: *Proceedings of the 8th workshop on syntax, semantics and structure in statistical translation* (pp. 103–111)

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2015) Gated feedback recurrent neural networks. In: *Proceedings of the 32nd international conference on machine learning* (pp. 2067–2075)

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019) BERT: Pre-training of deep bidirectional transformers for language understanding. In: *Proceedings of the 2019 conference of the North American Chapter of the Association for Computational Linguistics: Human language technologies* (pp. 4171–4186)

Feng, Y., Feng, Y., You, H., Zhao, X., & Gao, Y. (2019). Meshnet: Mesh neural network for 3D shape representation. *Proceedings of the AAAI Conference on Artificial Intelligence, 33*, 8279–8286.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*, 1735–1780.

ISO. (2016). ISO 10303–21:2016. Retrieved March, 2022, from https://www.iso.org/standard/63141.html

Kiani, M. A., & Saeed, H. A. (2019) Automatic spot welding feature recognition from step data. In: *2019 international symposium on recent advances in electrical engineering (RAEE)* (pp. 1–6)

Kingma, D., & Ba, J. (2015). Adam: A method for stochastic optimization. In: *Proceedings of the 3rd international conference on learning representations, (ICLR)*

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012) ImageNet classification with deep convolutional neural networks. In: *Proceedings of the 25th international conference on neural information processing systems* (pp. 1097–1105)

Maturana, D., & Scherer, S. (2015) Voxnet: A 3D convolutional neural network for real-time object recognition. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 922–928)

Qin, F., Li, L., Gao, S., Yang, X., & Chen, X. (2014). A deep learning approach to the classification of 3D CAD models. *Journal of Zhejiang University-Science C, 15*, 91–106.

Riegler, G., Osman Ulusoy, A., & Geiger, A. (2017). Octnet: Learning deep 3D representations at high resolutions. In: *Proceedings of*

the IEEE conference on computer vision and pattern recognition (CVPR) (pp. 3577–3586)

Shi, P., Qi, Q., Qin, Y., Scott, P. J., & Jiang, X. (2020). A novel learning-based feature recognition method using multiple sectional view representation. *Journal of Intelligent Manufacturing, 31*, 1291–1309.

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for largescale image recognition. In: *Proceedings of the 3rd international conference on learning representations (ICLR)* (pp. 1–14)

Socher, R., Lin, C. C. Y., Ng, A. Y., & Manning, C. D. (2011). Parsing natural scenes and natural language with recursive neural networks. In: *Proceedings of the 28th international conference on machine learning* (pp. 129–136)

Su, H., Maji, S., Kalogerakis, E., & Learned-Miller, E. (2015). Multi-view convolutional neural networks for 3D shape recognition. In: *Proceedings of the IEEE international conference on computer vision (ICCV)* (pp. 945–953)

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In: *Proceedings of the 27th international conference on neural information processing systems* (pp. 3104–3112)

Tai, K. S., Socher, R., & Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. In: *Annual meeting of the association for computational linguistics (ACL)* (pp. 1556–1566)

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017) Attention is all you need. In: *Proceedings of the 31st conference on neural information processing systems* (pp. 6000–6010)

Venu, B. K., Rao, V., & Srivastava, D. (2018). Step-based feature recognition system for b-spline surface features. *International Journal of Automation and Computing, 15*, 500–512.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., & Xiao, J. (2015) 3D ShapeNets: A deep representation for volumetric shapes. In: *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 1912–1920)

Zhang, Z., Jaiswal, P., & Rai, R. (2018). FeatureNet: Machining feature recognition based on 3d convolution neural network. *Computer-Aided Design, 101*, 12–22.

Zhong, R. Y., Xu, X., Klotz, E., & Newman, S. T. (2017). Intelligent manufacturing in the context of industry 4.0: A review. *Engineering, 3*, 616–630.