Journal of Computational and Applied Mathematics 427 (2023) 115166

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

Approaching STEP file analysis as a language processing task: A robust and scale-invariant solution for machining feature recognition

Victoria Miles *, Stefano Giani, Oliver Vogt

Department of Engineering, Durham University, Stockton Road, Durham, DH1 3LE, United Kingdom

ARTICLE INFO

Article history:
Received 27 October 2022
Received in revised form 9 February 2023

Keywords:
3D CAD
STEP files
Artificial intelligence
Recursive neural network
Recurrent neural network

ABSTRACT

Machining feature recognition is a key task in the intelligent analysis of 3D CAD models as it represents a bridge between a part design and the manufacturing processes required for manufacture and can, therefore, increase automation in the manufacturing process. As 3D model files do not naturally conform to the fixed size necessary as the input to most varieties of neural network, most existing solutions for machining feature recognition rely on either transforming CAD models into a fixed shape representation, accepting some loss of information in the process, or employ rigid rules-based feature extraction techniques prior to applying any learning-based algorithm, resulting in solutions which may display high performance for specific applications but which lack in the flexibility provided by a purely learning-based approach. In this paper, we present a novel machining feature recognition model, which is capable of interpreting the data present in a STEP (standard for the exchange of product data) file using purely learning-based algorithms, with no need for human input. Our model builds on the basic framework for feature extraction from STEP file data proposed in Miles et al. (2022), with the design of a decoder network capable of using extracted features to perform the complex task of machining feature recognition. Model performance is evaluated based on accuracy at the task of identifying 24 classes of machining feature in CAD models containing between two and ten intersecting features. Results demonstrate that our solution achieves comparable performance with existing solutions when given data similar to that used during training and significantly increased robustness when compared to existing solutions when presented with CAD models which vary from those seen during training and contain small features.

1. Introduction

With the increasingly wide availability of high-performing artificial intelligence technology, the manufacturing industry is currently undergoing a massive shift into a new age of production. Industry 4.0 is a term which refers to the envisioned near-future of manufacture, in which the wide scale implementation of intelligent manufacturing solutions represents the fourth industrial revolution. The general concept of intelligent manufacture as part of Industry 4.0 is of a manufacturing industry in which smart solutions aid in scheduling, monitoring and controlling the operations of smart machines and

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

https://doi.org/10.1016/j.cam.2023.115166

```
DATA;
#1 = LINE ( 'NONE', #211, #223 ) ;
#2 = CC_DESIGN_APPROVAL ( #245, ( #182 ) ) ;
#3 = PLANE ( 'NONE',  #151 ) ;
#4 = CARTESIAN_POINT ( 'NONE',  ( 0.4476975202560424805, -0.3669340610504150391, 0.8389387105574911407 ) ) ;
#5 = DIRECTION ( 'NONE',  ( 0.00000000000000000, 0.00000000000000000, 1.00000000000000000 ) ) ;
#6 = EDGE_CURVE ( 'NONE', #160, #291, #1, .T. ) ;
#7 = CIRCLE ( 'NONE', #33, 0.08437273502349851295 ) ;
#8 = VERTEX_POINT ( 'NONE', #232 ) ;
#9 = CC_DESIGN_APPROVAL ( #12, ( #209 ) ) ;
#10 = CARTESIAN_POINT ( 'NONE',  ( -0.4476975202560424805, 0.3669340610504150391, 0.000000000000000000 ) ) ;
#11 = COORDINATED_UNIVERSAL_TIME_OFFSET ( 0, 0, .AHEAD. ) ;
#12 = APPROVAL ( #254, 'UNSPECIFIED' ) ;
#13 = COORDINATED_UNIVERSAL_TIME_OFFSET ( 0, 0, .AHEAD. ) ;
#14 = ORIENTED_EDGE ( 'NONE', *, *, #147, .T. ) ;
#15 = EDGE_LOOP ( 'NONE', ( #181, #173, #297, #296 ) ) ;
#16 = DIRECTION ( 'NONE',  ( 0.00000000000000000, 0.00000000000000000, 1.00000000000000000 ) ) ;
#17 = AXIS2_PLACEMENT_3D ( 'NONE', #116, #71, #191 ) ;
#18 = APPROVAL_PERSON_ORGANIZATION ( #290, #245, #150 ) ;
#19 = ORIENTED_EDGE ( 'NONE', *, *, #283, .F. ) ;
#20 = AXIS2_PLACEMENT_3D ( 'NONE', #192, #72, #238 ) ;
```

Fig. 1. Segment of a STEP file.

in which artificial intelligence is used to aid in the design process by interfacing between CAD (computer aided design) models and physical manufacturing processes [1].

There are many potential applications for intelligent solutions which can effectively interpret 3D CAD models. Identifying the features present in a model can lead to increased automation between design and manufacturing processes. Smart analysis of databases of CAD models could be utilised in effectively sorting or searching through large databases for easier access to a manufacturing company's list of existing part designs. Smart analysis of features when compared to existing CAD models has the potential to lead to increased standardisation of features, for greater manufacturing efficiency.

Intelligent analysis of 3D data, such as CAD models, has traditionally been dominated by methods which utilise 2D images of 3D shapes, 3D grids of voxels, or point cloud data. Each of these methods necessitates the translation of accurate 3D model data into less accurate visual forms which are easier for neural networks to process, introducing issues such as limited resolution and geometric ambiguity. Solutions for CAD model analysis which do take model data directly as input tend to include rules-based elements, limiting the adaptability of the system. In this paper we present an intelligent solution for the key task of machining feature recognition, which utilises purely learning-based techniques and requires no transformation away from actual CAD model data.

STEP (standard for the exchange of product data) is an ISO standard model format [2] in which 3D geometries are represented using a hierarchical structure where simple components, such as points, are combined to form increasingly complex components, from edges to faces to complete model shells. The wide use of the STEP format in the manufacturing industry makes it suitable as input for a generic intelligent system, and as STEP is a text-based format, it is suitable for direct interpretation using artificial intelligence tools developed for language processing tasks.

A segment of a STEP file is shown in Fig. 1. Each line represents a single element, such as a point or edge and consists of a line ID, category, and a list of parameters which may be the IDs of other STEP lines, coordinate values or additional flags. A complex hierarchical structure is formed by connecting each line to each other line it depends on, with many nodes at the lowest level and a single node at the highest level which represents the entire model geometry.

A basic framework for intelligent analysis of STEP file data is presented in [3], in which a recursive encoder network is used to intelligently compress the geometric data from a STEP file into a single-vector representation. In this paper we build on this framework to develop an intelligent solution for the detection of common machining features, and demonstrate the competitive performance of our model.

1.1. Machining feature recognition

Meaningful analysis of 3D data is a task which poses significant challenges for artificial intelligence models. 3D data is inherently complex and does not simply conform to the fixed input shape required by many neural network architectures. Therefore, most existing approaches to the analysis of 3D data using neural networks rely on transforming the data into a fixed size, with inevitable loss of 3D data associated. with common approaches including the multi-view approach, the voxel approach and the point cloud approach.

In the multi-view approach [4,5], multiple 2D images of a 3D object are taken as input to a neural network. The outputs of the network across each image are in some way combined, to produce predictions regarding the shape of the 3D object. This approach relies on all relevant features of the 3D object being visible in the images selected, with features which are small in size or located in inconvenient positions posing greater challenges for the neural network.

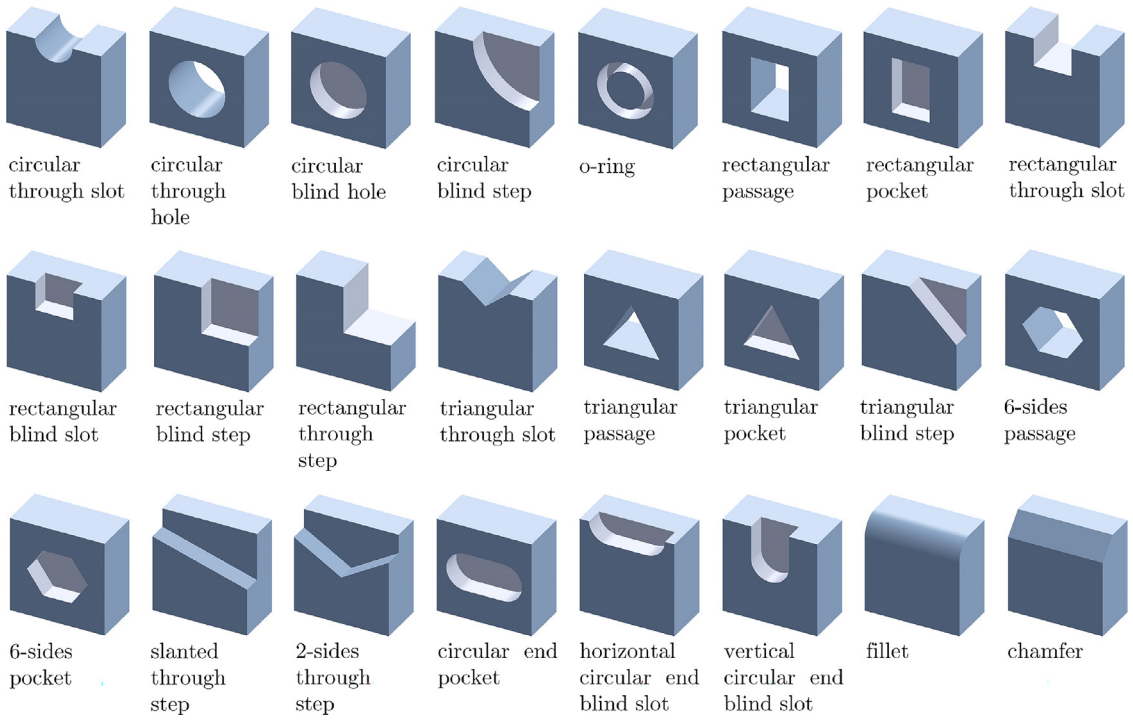


Fig. 2. 24 classes of machining feature, from [3].

The voxel-based approach [6–8] involves converting 3D objects into 3D grids of 'voxels' and using these as input to a 3D neural network, such as a 3D CNN. This method has a severe resolution issue, as the 3D neural networks used to interpret the voxel grids become impractically large unless input dimensions are kept relatively small. The necessary low resolution of input data results in models which can achieve success when recognising general shapes but which struggle with fine details and small features.

The point cloud approach [9,10] improves on this resolution issue by representing a 3D object more efficiently using selected points from surface faces. However, there remains a limitation to resolution based on the total number of points used to represent each model.

Machining features (or manufacturing features) are simple features, such as holes and slots, which can be produced via a single manufacturing process. Through the combination of several machining features, complex 3D part designs can be realised. The recognition of these features within a 3D model is, therefore, a key task for intelligent solutions which analyse CAD models, as identifying the features present in a model represents a direct bridge between a CAD design and the manufacturing processes necessary to produce the part it represents. Fig. 2 shows examples of 24 classes of machining feature, which will be referenced throughout this paper.

There are existing machining feature recognition models based on each of the approaches outlined above. FeatureNet [11] uses a 3D CNN to classify machining features based on 3D voxel grids with dimensions between $16 \times 16 \times 16$ and $64 \times 64 \times 64$. In MsvNet [12], voxelised models are represented using multiple 2D sectional views, created by making random cuts into the model to allow for representation of the inside of the model. The 2D images produced are segmented to isolate individual features using the selective search algorithm, then individual feature representations are used as the input to a 2D CNN which performs feature recognition. This architecture was improved on in [13], which presented SsdNet, in which segmentation and feature recognition are combined into a single process based on the single shot multibox detector (SSD) [14]. In [15], PointNet++ [10], a hierarchical network which makes use of point cloud data, is used to perform both single-feature classification and multi-feature recognition. In [16], Associatively Segmenting and Identifying Network (ASIN) is proposed, a network which clusters faces belonging to the same machining feature using point cloud data before performing identification of feature class.

In recent years, work has been done applying learning-based techniques directly to the faces of boundary representation (B-rep) models to address the issue of the loss of 3D model data when converting to a fixed-size input. In [17], information from each B-rep face is encoded and a feature class predicted for each face to effectively segment the CAD model into individual machining features.

In addition to these learning-based models, there is also significant existing research focusing on the use of rules-based techniques for extraction of relevant features from STEP files. As with any rules-based approach, these models are typically limited to specific tasks such as detecting b-spline surface features [18], spot-welding features [19] or identifying features

relevant to the V-bending process [20]. Even more general solutions, such as the feature recognition system for rotational parts presented in [21], rely on limited feature sets which can only be expanded through the production of a new set of definitive rules to govern the new feature class, thus lacking the flexibility of a learning-based approach.

The novelty of our approach lies in treating the interpretation of a STEP file as a language processing task, with the hierarchical information represented in a STEP file maintained as input to the neural network, resulting in a purely learning-based approach which takes model files directly as input.

1.2. Neural networks for language processing

The STEP format is an example of an artificial, or computer, language, with its own fixed vocabulary and rules equivalent to a simple grammatical system. As they are languages designed to be interpreted by computers, artificial languages are inherently less nuanced and complex than natural, or human, languages, with consistent rules and no ambiguity. They therefore provide a significantly more straightforward form of data for artificial intelligence solutions to interpret than natural languages, whilst retaining enough similarity to permit the direct application of neural networks designed for natural language processing.

Early work into applying neural networks to natural language processing tasks primarily focused on the recurrent neural network [22]. A recurrent neural network (or RNN), in contrast to most neural networks, is designed to handle a variable input size. Input data is expected to be a sequence of any length, such as a natural language sentence, with a single cell duplicated as many times as necessary to process the entire sequence. At each step, a new word from the sequence is input to the cell and a new output hidden state is calculated. This output hidden state is then taken as the input hidden state for the next step in the sequence. Thus, memory is maintained of the information added at each step. A common variant of RNN is the LSTM (long short term memory) network [23], which makes use of the LSTM cell [24], in which two memory states are maintained, representing both long and short-term memory.

Recursive neural networks have architecture very similar to recurrent neural networks, with the key difference being the shape of the input data. Whilst recurrent networks assume that input data is in the form of a linear sequence of inputs, recursive neural networks assume that input data has a hierarchical, or tree, structure. First proposed in [25], the recursive neural network was initially designed for applications in both natural language processing and semantic scene segmentation, exploiting the hierarchical structures present both in natural language and images. This architecture was improved on in [26] with the implementation of a modified LSTM cell, designed to be operated within a tree structure. Although the continued dominance of RNNs [27,28] and transformer networks [29,30] has limited the scale of further research into the potential of recursive architectures for natural language processing, recursive networks have since been adapted to applications as diverse as program translation [31], identification of protein interactions [32] and jet physics [33].

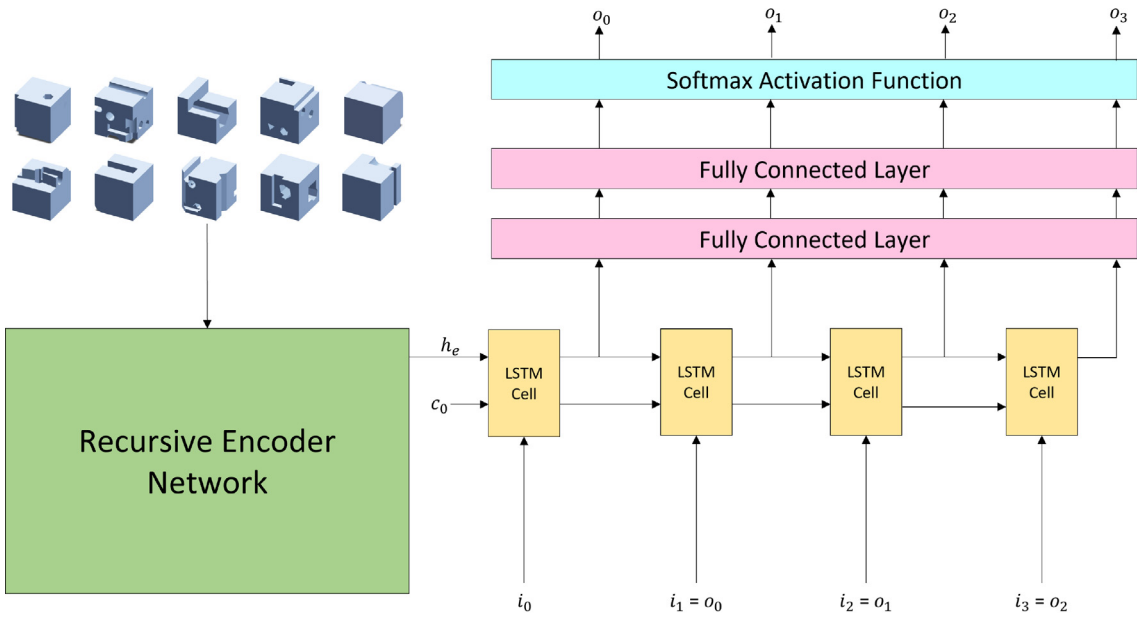
In [3] a recursive neural network is applied to the geometric data from a STEP file in order to compress the complex hierarchical information into a single-vector representation. The viability of this approach is demonstrated through classification of machining features. This work reports high accuracy for the single-feature classification task but does not implement a model applying the encoder to any complex task, such as multi-feature recognition.

2. Methods

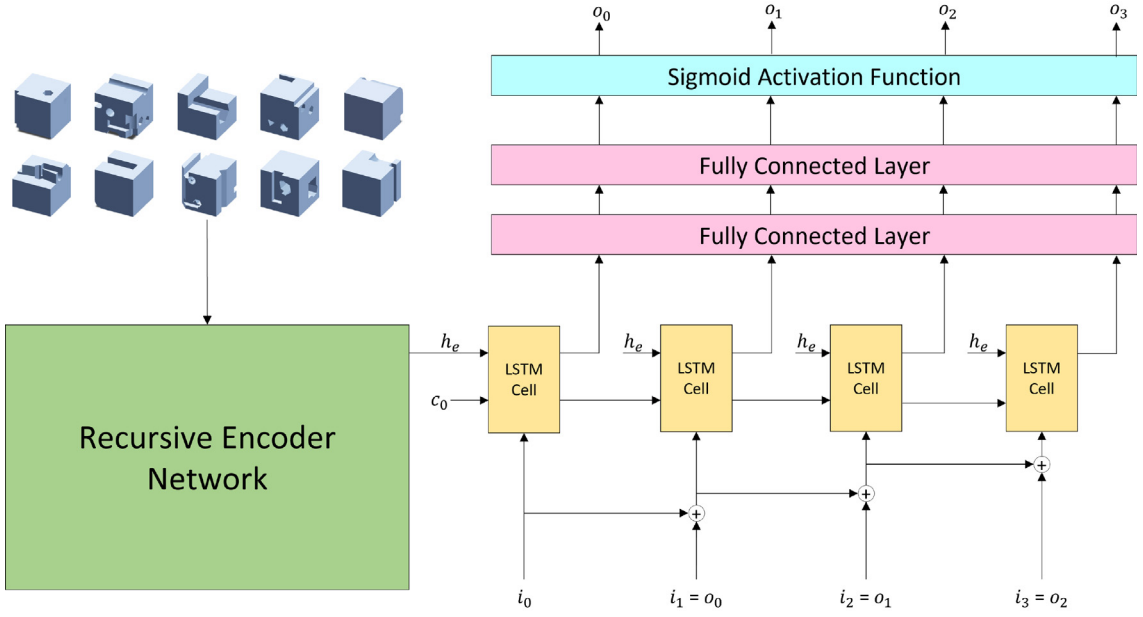
In this paper, we propose a novel architecture for machining feature recognition, incorporating the recursive encoder network presented in [3]. The operation of the neural network is shown in Fig. 3(b). The network consists of two sub-networks, the recursive encoder network, which takes the data from a STEP file as input and produces a single-vector encoding representing each CAD model, and the decoder network which uses these encoded vectors to produce a list of class predictions for each CAD model. This section contains a brief outline of the encoder network, as first presented in [3], followed by details of the design and training of a novel LSTM-based decoder network including evaluation of several architecture variations and selection of appropriate model parameters.

2.1. Recursive encoder

The recursive encoder network is based on the Child-Sum Tree-LSTM network first proposed in [26]. The network takes the tree-structured data represented in a STEP file and applies a treeLSTM cell to every node in the input structure, producing vectors which represent increasingly complex information as the network parses recursively through the hierarchical data structure. The final output of the encoder network is a single vector representing the output of the treeLSTM cell applied to the highest-level node of the input data tree. As all other nodes in the tree feed directly into this highest-level node, it can be seen to represent the entire data tree and so the output vector represents an encoding of all of the geometric information present in a STEP file. Full detail of the implementation of the recursive encoder is presented in [3].



(a) Traditional LSTM architecture, where each output vector produces a single predicted class.



(b) Adapted decoder architecture, where each output vector may produce up to 24 class predictions.

Fig. 3. Neural network architecture for traditional and adapted decoder networks, showing four prediction steps, where i_i are input vectors, o_i output vectors, h_e the output hidden state from the encoder network and c_0 the initial cell state.

2.2. Decoder architecture

The decoder network makes use of the LSTM cell. An LSTM cell consists of three gates, the forget, input and output gates, which are used to update two memory states: the hidden state (short-term memory) and the cell state (long-term memory). Each cell takes a single new input in the form of a vector. Each gate performs a specific function, with behaviour in a particular LSTM cell dependent on the previous hidden state and the input vector. First, the forget gate controls how much information from the previous cell state is copied into the new cell state and how much is forgotten. Then, the input gate controls which information from the previous hidden state and the new input should be written into the new

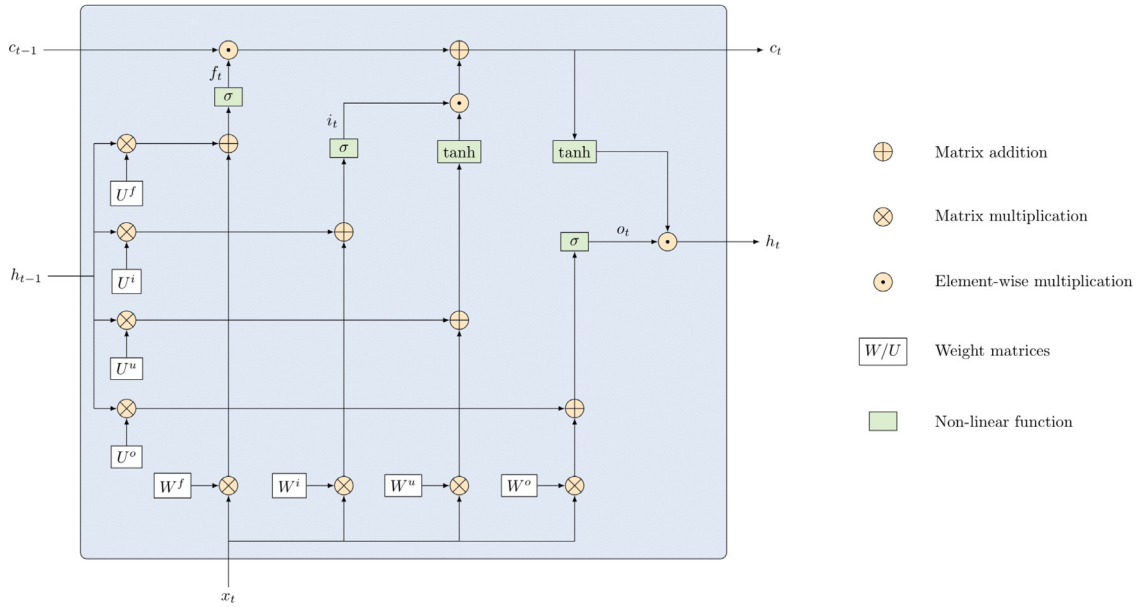


Fig. 4. Processes within an LSTM cell, where h_t is the hidden state, c_t is the cell state, x_t the new input and f_t, i_t and o_t are outputs of the forget, input and output gates at time step t.

cell state. Finally, the output gate controls how much of the new cell state should be written to the new hidden state. The hidden state is taken as the output of the cell, whereas the cell state merely represents long-term memory, which will be passed to the next LSTM cell in the chain. The processes within an LSTM cell are visualised in Fig. 4.

Fig. 3(a) shows a decoder network using a traditional LSTM-based architecture. The cells are arranged in a chain, with the output hidden states from each cell taken as the input for the next cell in the chain. The output hidden state from each cell is passed through fully-connected layers, reducing the dimensions of output vectors to equal the number of classes in the dataset. A softmax activation function is then applied to produce a probability score for each feature class as follows:

$$Softmax(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{1}$$

and the class with the highest score is predicted at this step. The first input is a zero vector, with subsequent cells taking the previous output vector as input. Thus, information is passed through the chain of LSTM cells, with a single prediction made at each step. Predictions stop when a designated end token is predicted.

To better adapt the LSTM architecture to fit the desired task, several alterations were made to this traditional architecture:

1. Given that the order of the output predictions is not relevant, an updating input vector was implemented. Instead of feeding the previous output directly into the cell as input, now the predictions from each previous step are summed. This results in input vectors which represent all previous predictions, giving more useful information as input for each prediction step.
2. As the goal of the decoder is to derive as much meaning as possible from the output of the recursive encoder, it was decided to feed the encoder output directly into each LSTM cell as the previous hidden state. The cell state is still passed through the chain, giving the network an updating long-term memory whilst always working directly on the encoder output as the short-term memory. This, combined with the updating input vector, means that the decoder is always directly analysing the encoder output, with an updating memory of previous predictions.
3. To reduce the number of prediction steps necessary, the output layers have been adapted to produce several predictions at every output step. In order to encourage the existence of multiple high values in the output vector, the softmax activation function is replaced with sigmoid, which is calculated element-wise as:

$$\sigma(x) = \frac{1}{1+\exp(-x)} \tag{2}$$

To predict multiple labels in one step, all classes with scores higher than a given threshold are predicted. Through the trial of several values, a threshold of 0.7 was deemed to be appropriate for output predictions and so the output

Table 1
Train-time feature recognition accuracy for a traditional LSTM architecture and after implementing each alteration to the architecture, each model listed also includes all adaptations listed above.

	Decoder architecture	Validation accuracy
	Traditional LSTM	0.731
(1)	With updating input vector	0.874
(2)	With encoder hidden state used as input for every step	0.897
(3)	**With sigmoid activation giving multiple predictions per step**	**0.902**

vector is converted into a binary vector as follows:

$$x = \begin{cases} 1 & \text{if } \sigma(x) \geq 0.7 \\ 0 & \text{if } \sigma(x) < 0.7 \end{cases} \tag{3}$$

In this way, each instance of the LSTM cell predicts a list of feature classes until a step is reached where no further predictions are produced. Thus, for a CAD model which does not contain more than one instance of any machining feature class, it is possible for the model to predict all classes present with a single prediction step.

Fig. 3 shows the alterations made to the decoder network, with a traditional LSTM network displayed in Fig. 3(a) and our adapted version in Fig. 3(b). Feature recognition accuracy achieved after each of these alterations was introduced, as well as for the traditional LSTM architecture, is presented in Table 1.

2.3. Neural network size

One advantage of our approach is the relative simplicity of the neural network architecture; the encoder network consists of a single treeLSTM layer and the decoder of a single LSTM layer with two fully connected layers at the output. However, in order to ensure optimum performance whilst minimising the number of learnable parameters, a study was carried out to compare the performance of several variations of the neural network. Hidden size of the encoder and decoder LSTM layers was varied between 300 and 600. In addition, decoder networks with a single output fully connected layer and with two fully connected layers where the output size of the first layer was varied between 50 and 150 were used.

Fig. 5 shows the validation accuracy for each of the model variants tested. The accuracy measure used is f-score, a weighted average of precision and recall, calculated as:

$$precision = \frac{tp}{tp + fp} \tag{4}$$

$$recall = \frac{tp}{tp + fn} \tag{5}$$

$$fscore = \frac{2 \times precision \times recall}{precision + recall} \tag{6}$$

where tp is number of true positives, fp number of false positives and fn number of false negatives in the neural network predictions.

It can be clearly seen from Fig. 5 that a hidden size of 500 achieves optimal accuracy, outperforming both larger and smaller networks. This combined with two fully connected layers, with output size of the first layer equalling 50 or 100 achieve the highest f-scores. Of these two models, either would be an appropriate choice. One model is smaller, as a smaller fully connected layer required fewer learned parameters. However, the other network converged to a slightly better solution in fewer training steps, indicating a shorter training time. For the remainder of this paper, the model with hidden size 500 and intermediate fully connected size of 100 is used when presenting results.

2.4. Training

The neural network is trained to minimise binary cross-entropy (BCE) loss between each set of predictions and those expected for the corresponding LSTM cell. BCE loss can be calculated as:

$$L = \frac{1}{n} \sum_{i=1}^{n} y_i \cdot \log x_i + (1 - y_i) \cdot \log (1 - x_i) \tag{7}$$

where n is the number of values in the model output, x_i is the ith scalar value in the model output and y_i is the ith scalar value in the target vector. During training, the number of output vectors predicted is limited to 10 and cumulative loss is calculated. Ideal behaviour is defined as detecting all features present in a CAD model in as few steps as possible, and good behaviour as detecting all features within the 10 steps allowed. In order to encourage this behaviour, the first

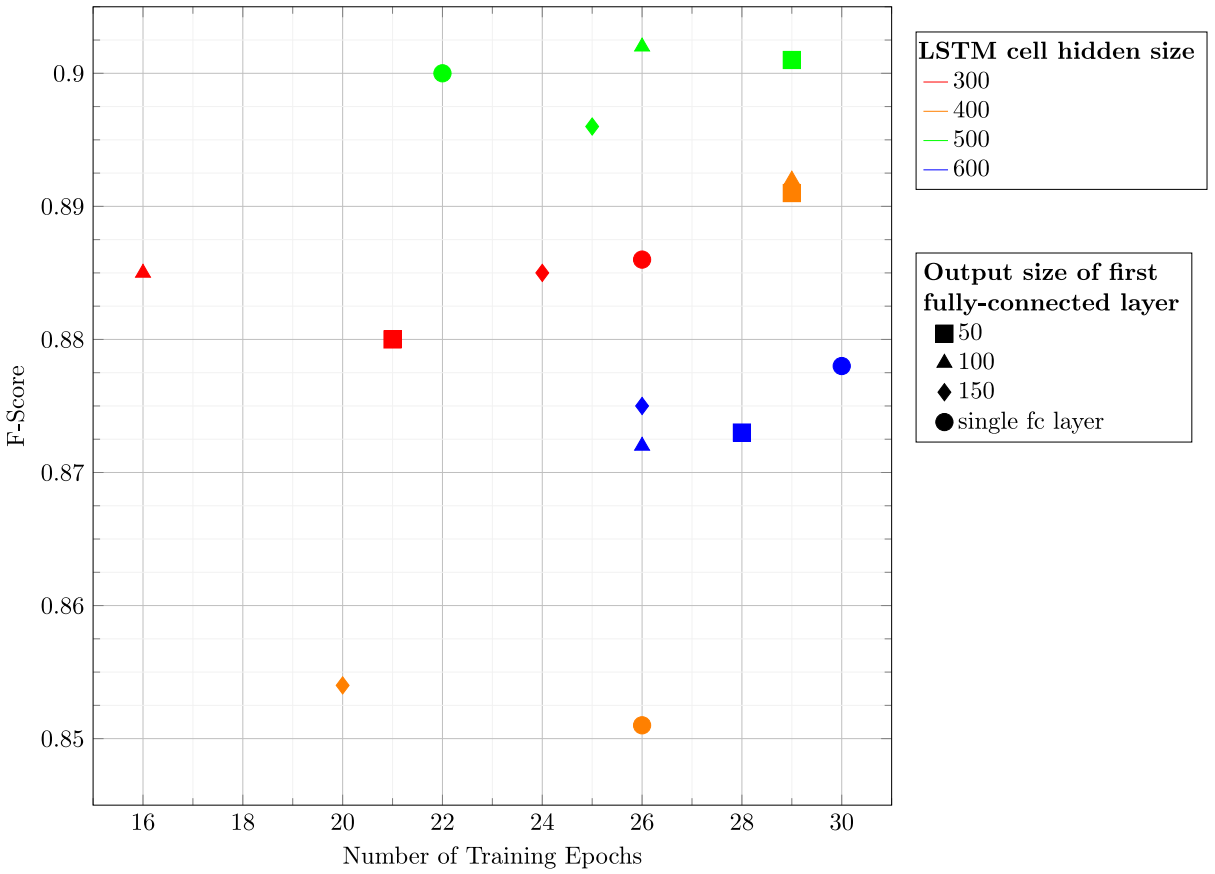


Fig. 5. Validation f-scores plotted against number of training steps required for various model sizes.

target vector is set as a list of all present feature classes. Subsequent target vectors are initialised to represent only feature classes which appear multiple times in the model. At each round of predictions, any classes from the target vector which are not predicted by the network will be passed through into the next target vector. If the model has successfully detected every ground truth class, the loop is broken and the process of loss accumulation halted. Thus, if the model successfully predicts all ground truth classes quickly, loss is minimised, whereas if the model takes longer to make the predictions, there will be more steps taken in which to accumulate loss.

In order to effectively learn weights for both the encoder and decoder networks, the neural network is trained using a two step process. First, the encoder is pre-trained using a simplified decoder. This simplified model has no LSTM cells and instead feeds the output of the encoder network directly into the fully connected layers, to produce a single binary vector representing the presence or lack thereof of each feature class. In the second step of training, the simplified decoder is replaced with the full decoder architecture. In this second phase of training, the decoder weights are learned and the encoder weights merely fine-tuned.

3. Results and discussion

The neural network has been trained to perform a machining feature recognition task, in which a list of feature class predictions is produced for each STEP file given to the model as input. The network was trained with learning rate initialised at 1e-3 and the Adam optimiser used [34]. All code is written in PyTorch and training and testing for our network is carried out using a Xeon E5–2609 v3 CPU with two cores and 20 GB RAM.

In order to demonstrate the significance of our results, network performance will be compared to that of two existing machining feature recognition models, MsvNet [12] and SsdNet [14]. In both cases, optimal pre-trained versions of the models are used, and testing is carried out using a Xeon Gold 6134 CPU with RTX 2080 Ti GPU.

3.1. Machining features dataset

In this paper, the neural network is trained and tested using a dataset based on 24 classes of simple machining feature, as shown in Fig. 2. Each model in the machining features dataset consists of a standard base cube with side length 10 cm,

Table 2
Parameters used to generate test datasets.

Test set	Number of models	Scale factor (Minimum size)
1	400	1
2	200	1/2
3	200	1/3
4	200	1/4

Table 3
Parameters used to generate data subsets.

Data subset	1	2	3	4	5	6	7	8	9	10
Number of features	2	2	3	4	5	6	7	8	9	10
Scale factor (Maximum size)	1	1/2	1/2	1/2	1/3	1/3	1/3	1/4	1/4	1/4

with between two and ten machining features added. The classes of feature added to each model are randomised, along with all dimensions of the feature, which are assigned random values within fixed bounds. The list of feature classes included, as well as the scale of the models and dimension limits for features are based on those included in the benchmark dataset presented in [12]. However, the benchmark itself is not used to assess performance as the dataset contains models only in STL format and large-scale conversion of a dataset from STL to STEP format is not practical.

The neural network is trained using a dataset consisting of 2000 total models, divided into train and validation sets at a ratio of 8:2, giving a train set of 1600 total models and a validation set of 400.

Performance is assessed using several test datasets. These are generated using the same randomised process as the training datasets, meaning that the models produced are equivalent to those used during training but, due to the randomisation, the recurrence of an identical feature is extremely unlikely. The first test set is a dataset of size 400 in which feature size limits are set to the same values as those used in the training sets. In order to demonstrate the scale invariance of our model, additional test datasets are constructed with the minimum size allowed for feature dimensions reduced. The parameters used to generate each test dataset are shown in Table 2. Each dataset is comprised of 10 subsets, each containing an equal number of models and generated with parameters as shown in Table 3.

For direct comparison of the performance of our network with those of existing solutions, our test datasets have been converted into the STL and binvox formats compatible with the comparison networks, allowing for direct comparison using the same datasets. As our first dataset has been designed to be exactly equivalent to the benchmark set, with the same parameters used for generation, optimal performance can be expected from comparison models trained using the benchmark data.

Fig. 6 shows an example CAD model from each data subset in each of the four test sets.

3.2. Feature recognition performance

Performance using data equivalent to that seen during training will first be discussed, followed by performance using the scaled-down datasets.

3.2.1. Performance on data equivalent to training data

Table 4 shows the f-score achieved by our network, compared to existing solutions when performing a machining feature recognition task using data equivalent to that seen during training. These results are visualised in Fig. 8(a). As can be seen from the data, our network achieves high performance, close to that of SsdNet and significantly outperforms MsvNet.

A key point of comparison is the performance of our model and that of SsdNet across data subsets 1 and 2. As both subsets consist of models containing two machining features, the only difference between these sets is the scale of the features. The scaled down features in subset 2 have less overall intersection due to their reduced size. Therefore, the significantly improved performance of our network on subset two indicates the high degree to which our model's limitations are caused by intersection of features. The comparatively consistent performance of SsdNet across these two data subsets indicates either that SsdNet's limitations are not as closely connected to level of intersection, or that the improvements in performance connected to a lower level of intersection are negated by a simultaneous decrease in performance due to the increased number of smaller features.

This disparity can again be seen in the comparative performance across data subsets 4 and 5. While adding an additional feature and decreasing the feature size once more results in a roughly 3% drop in accuracy for SsdNet, our model's accuracy again increases.

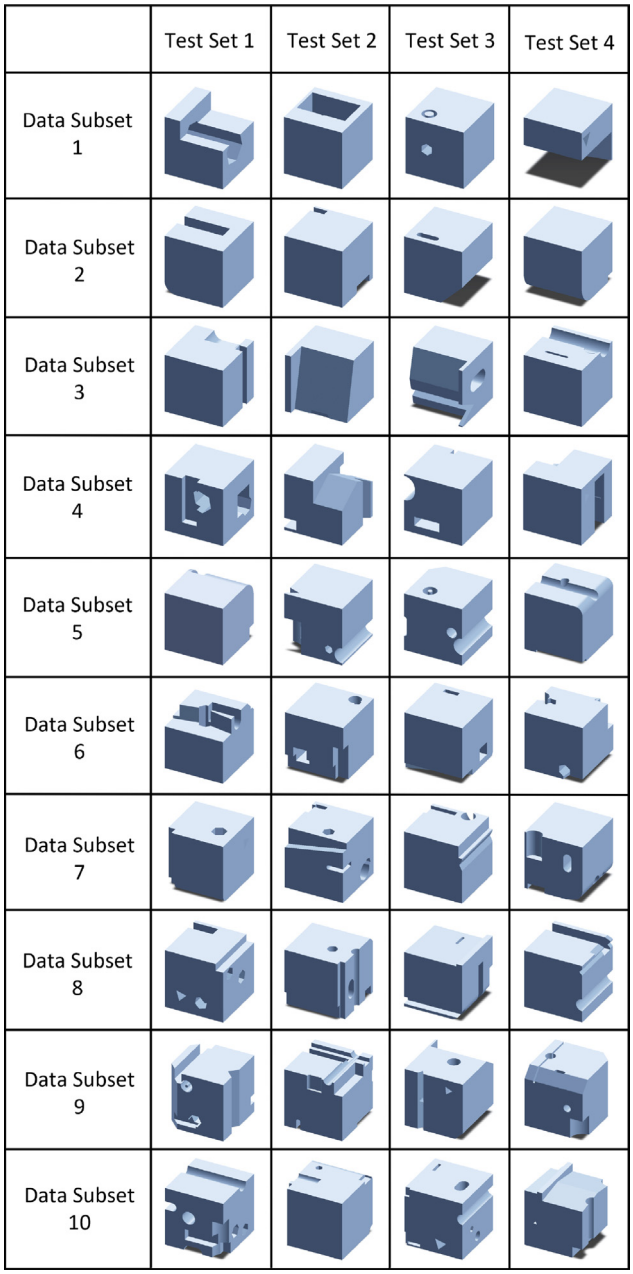


Fig. 6. Example CAD models from each test dataset.

Table 4
F-scores for Test Set 1, separated by data subset.

Data subset	1	2	3	4	5	6	7	8	9	10	Total
MsvNet [12]	0.7205	0.8481	0.7699	0.7752	0.7399	0.6866	0.6994	0.7219	0.6657	0.6779	0.7156
SsdNet [14]	0.9554	0.956	0.9664	0.95	0.9262	0.9021	0.9114	0.9006	0.9211	0.9168	0.9217
Ours	0.9250	0.9875	0.9106	0.9270	0.9415	0.8980	0.9031	0.8783	0.8843	0.8152	0.8906

3.2.2. Scale invariance: Investigating the impact of reducing feature size

A key advantage of our approach is that, by taking STEP files directly as input to the neural network, all information necessary to reproduce the model can be maintained. Unlike image and voxel-based approaches, resolution is not an issue and so there is no limitation to the minimum feature size relative to the model size which can be identified. In order

Table 5
F-scores for each test dataset.

Model	Test Set 1	Test Set 2	Test Set 3	Test Set 4
MsvNet	0.7156	0.6101	0.6023	0.5713
SsdNet	0.9217	0.8367	0.7657	0.7413
Ours	0.8906	0.8976	0.8831	0.8780

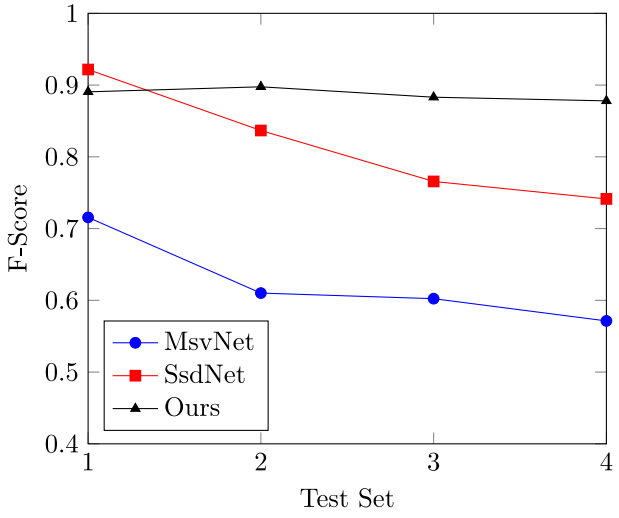


Fig. 7. Overall f-score of each test dataset plotted against minimum feature size scale factor.

to demonstrate this scale invariance, the network is tested using three additional test datasets in which the minimum dimensions for each feature have been scaled down by factors of 2, 3 and 4. For comparison, the scaled down datasets have also been converted into STL and binvox formats, allowing SsdNet and MsvNet to also run on each dataset to measure comparative performance. Neither our model or either of the comparison models have been trained using data containing small features so, for all three networks, these datasets contain models which are substantially different from anything seen during training.

Table 5 shows the overall f-score for each test dataset, and these results are plotted in Fig. 7.

As can be seen from Fig. 7, both MsvNet and SsdNet show a clear downwards trend in accuracy when increasingly smaller features are included in the dataset. This is a predictable result; Both networks rely on voxelised models, with dimensions in this case set to their highest values of $64 \times 64 \times 64$. As an inevitable result of this process, smaller features are lost or distorted as very few voxels are used to represent them. Increasing the dimensions of the voxelised model can result in impractically large networks and regardless of the size of the voxel grid there will always be a limit to the resolution of the model.

In contrast, our solution shows no strong correlation between minimum feature size and model performance. Small features are represented using the same components in a STEP file as large features, with only the coordinate values differentiating features based on size. Therefore, our model is capable of demonstrating scale-invariance to a degree which would not be possible for a model reliant on voxelised representations. Moreover, the network demonstrates robustness when faced with CAD models with features which differ significantly in scale from those seen during training, indicating a strong general understanding of feature shape which is not reliant on features conforming to the expected scale.

Table 6 shows the performance results broken down by data subset and the performance of each model across each test dataset and subset is plotted in Fig. 8.

As can be seen in Fig. 8(a), when using Test Set 1, in which feature sizes are equivalent to those used during training, our model and SsdNet display comparable performance. However, as increasingly small features are allowed, increased separation in detection performance can be observed, with performance of both comparison networks dropping significantly, whilst ours remains consistent. Fig. 9 shows the drop in performance for SsdNet, compared to the comparatively consistent performance of our solution.

4. Conclusions

This paper presents a novel solution for machining feature recognition in which the interpretation of STEP files is treated as a language processing task. Results show that our network displays high performance independent of minimum

Table 6
Network performance across all test subsets.

Model	Data subset	1	2	3	4	5	6	7	8	9	10	Total
MsvNet [12]	Scale 1	0.7205	0.8481	0.7699	0.7752	0.7399	0.6866	0.6994	0.7219	0.6657	0.6779	0.7156
	Scale 2	0.8052	0.6053	0.7664	0.6531	0.7571	0.5951	0.6378	0.5839	0.4984	0.5581	0.6101
	Scale 3	0.6579	0.7632	0.7027	0.7183	0.7363	0.6667	0.5815	0.5292	0.5811	0.5122	0.6023
	Scale 4	0.6446	0.7818	0.6988	0.6697	0.6347	0.5817	0.6006	0.5089	0.4955	0.5011	0.5713
SsdNet [14]	Scale 1	0.9554	0.9560	0.9664	0.9500	0.9262	0.9021	0.9114	0.9006	0.9211	0.9168	**0.9217**
	Scale 2	0.8571	0.9114	0.8689	0.8758	0.9053	0.8547	0.8645	0.8608	0.7588	0.7757	0.8367
	Scale 3	0.7848	0.8312	0.8673	0.8742	0.8283	0.8295	0.7344	0.7153	0.7222	0.6997	0.7657
	Scale 4	0.8983	0.8727	0.7901	0.8472	0.8073	0.7485	0.7565	0.6746	0.6994	0.6578	0.7413
Ours	Scale 1	0.9250	0.9875	0.9106	0.9270	0.9415	0.8980	0.9031	0.8783	0.8843	0.8152	0.8906
	Scale 2	0.9639	0.9877	0.9322	0.9114	0.9286	0.9160	0.8561	0.9055	0.8606	0.8740	**0.8976**
	Scale 3	0.8608	0.9877	0.9344	0.9125	0.9490	0.9270	0.8645	0.8636	0.8300	0.8525	**0.8831**
	Scale 4	0.9421	0.9500	0.8962	0.9099	0.8829	0.9209	0.8492	0.8639	0.8527	0.8546	**0.8780**

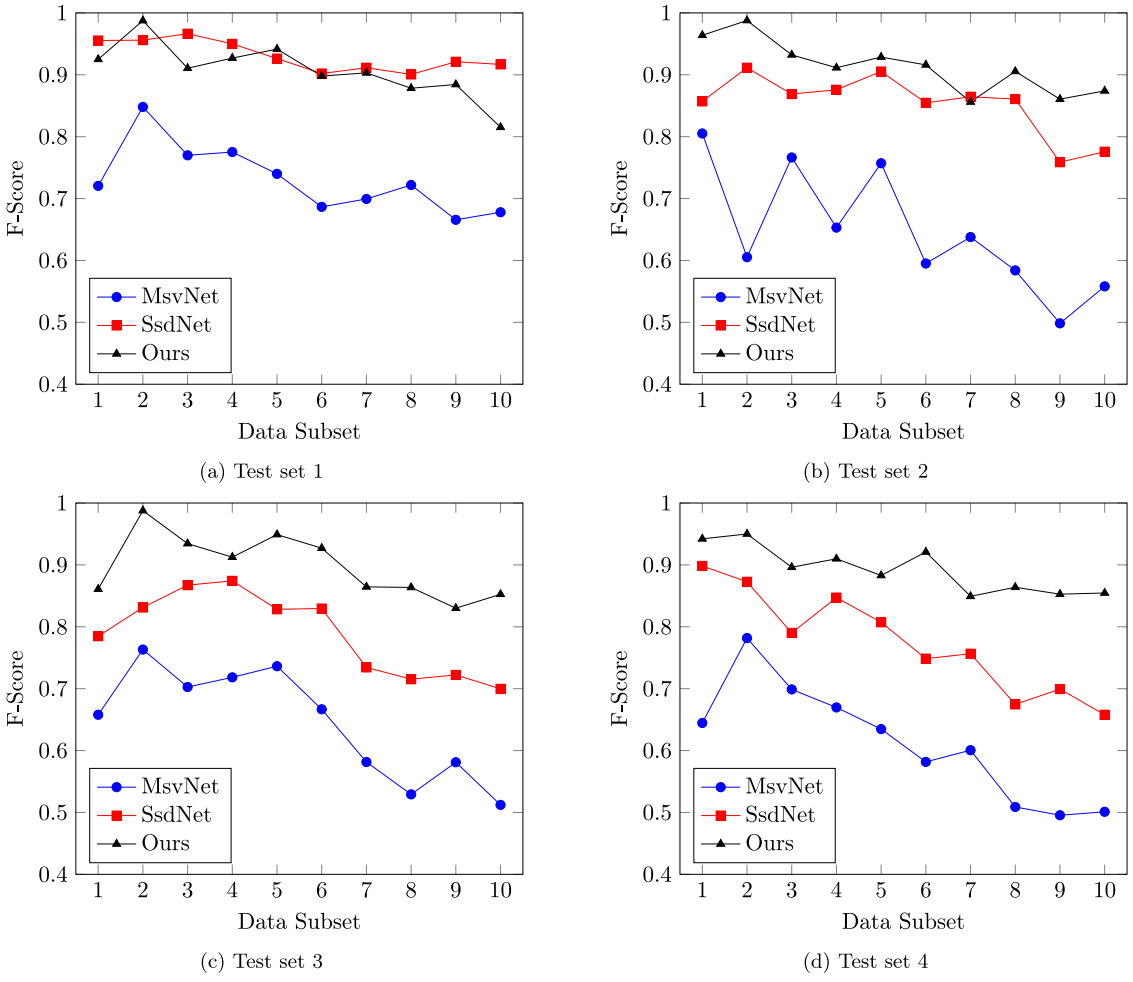


Fig. 8. Feature recognition performance across all test subsets.

feature size, whilst other existing solutions show reductions in performance as features become smaller. As complex CAD models may contain small features relative to the model size, this result demonstrates that our model is capable of performing consistently across a wider range of CAD models, and therefore displays greater flexibility than the existing solutions used for comparison.

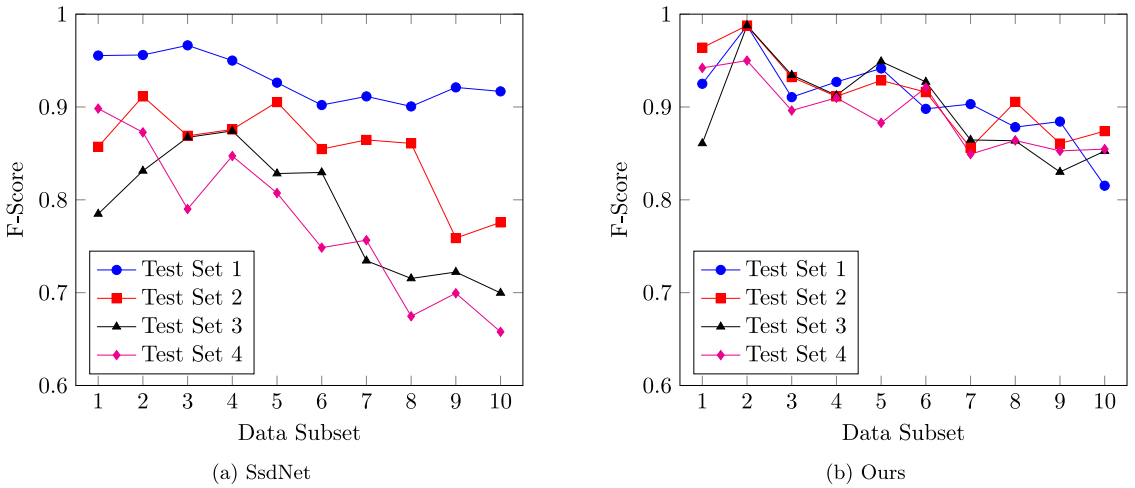


Fig. 9. Feature recognition performance across all test subsets.

Data availability

A link to the reserved doi for the dataset is included in the acknowledgements section

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), United Kingdom [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 dataset used in this paper can be accessed from the Durham Research Data Repository, at http://dx.doi.org/10.15128/r26q182k16s. The authors have no competing interests to declare.

References

[1] R.Y. Zhong, X. Xu, E. Klotz, S.T. Newman, Intelligent manufacturing in the context of industry 4.0: A review, Engineering 3 (2017) 616–630.
[2] ISO, Standards, 2016, https://www.iso.org/standard/63141.html [Accessed March 2022].
[3] V. Miles, S. Giani, O. Vogt, Recursive encoder network for the automatic analysis of STEP files, J. Intell. Manuf. (2022) 1–16.
[4] F.-w. Qin, L.-y. Li, S.-m. Gao, X.-l. Yang, X. Chen, A deep learning approach to the classification of 3D CAD models, J. Zhejiang Univ. -Sci. C 15 (2014) 91–106.
[5] H. Su, S. Maji, E. Kalogerakis, E. Learned-Miller, Multi-view convolutional neural networks for 3D shape recognition, in: Proceedings of the IEEE International Conference on Computer Vision, ICCV, 2015.
[6] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, J. Xiao, 3D ShapeNets: A deep representation for volumetric shapes, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2015.
[7] D. Maturana, S. Scherer, VoxNet: A 3D convolutional neural network for real-time object recognition, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2015, pp. 922–928.
[8] G. Riegler, A. Osman Ulusoy, A. Geiger, OctNet: Learning deep 3D representations at high resolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2017.
[9] C.R. Qi, H. Su, K. Mo, L.J. Guibas, PointNet: Deep learning on point sets for 3D classification and segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2017.
[10] C.R. Qi, L. Yi, H. Su, L.J. Guibas, Pointnet++: Deep hierarchical feature learning on point sets in a metric space, Adv. Neural Inf. Process. Syst. 30 (2017).
[11] Z. Zhang, P. Jaiswal, R. Rai, FeatureNet: Machining feature recognition based on 3D convolution neural network, Comput. Aided Des. 101 (2018) 12–22.
[12] P. Shi, Q. Qi, Y. Qin, P.J. Scott, X. Jiang, A novel learning-based feature recognition method using multiple sectional view representation, J. Intell. Manuf. 31 (2020) 1291–1309.
[13] P. Shi, Q. Qi, Y. Qin, P.J. Scott, X. Jiang, Intersecting machining feature localization and recognition via single shot multibox detector, IEEE Trans. Ind. Inform. 17 (5) (2021) 3292–3302.
[14] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, A. Berg, SSD: Single shot MultiBox detector, in: Proceedings of the European Conference on Computer Vision, ECCV, 2016.
[15] X. Yao, D. Wang, T. Yu, C. Luan, J. Fu, A machining feature recognition approach based on hierarchical neural network for multi-feature point cloud models, J. Intell. Manuf. (2022) 1–12.
[16] H. Zhang, S. Zhang, Y. Zhang, J. Liang, Z. Wang, Machining feature recognition based on a novel multi-task deep learning network, Robot. Comput.-Integr. Manuf. 77 (2022) 102369.

[17] C. Yeo, B.C. Kim, S. Cheon, J. Lee, D. Mun, Machining feature recognition based on deep neural networks to support tight integration with 3D CAD systems, Sci. Rep. 11 (1) (2021) 1–20.
[18] B.K. Venu, V. Rao, D. Srivastava, STEP-based feature recognition system for B-spline surface features, Int. J. Autom. Comput. 15 (2018) 500–512.
[19] M.A. Kiani, H.A. Saeed, Automatic spot welding feature recognition from STEP data, in: Proceedings of the International Symposium on Recent Advances in Electrical Engineering, Vol. 4, RAEE, 2019, pp. 1–6.
[20] A.A. Salem, T.F. Abdelmaguid, A.S. Wifi, A. Elmokadem, Towards an efficient process planning of the V-bending process: an enhanced automated feature recognition system, Int. J. Adv. Manuf. Technol. 91 (9) (2017) 4163–4181.
[21] M. Al-wswasi, A. Ivanov, A novel and smart interactive feature recognition system for rotational parts using a STEP file, Int. J. Adv. Manuf. Technol. 104 (2019) 261–284.
[22] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP, 2014, pp. 1724–1734.
[23] I. Sutskever, O. Vinyals, Q.V. Le, Sequence to sequence learning with neural networks, 2014, CoRR, arXiv:1409.3215.
[24] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (1997) 1735–1780.
[25] R. Socher, C.C.-Y. Lin, A.Y. Ng, C.D. Manning, Parsing natural scenes and natural language with recursive neural networks, in: Proceedings of the International Conference on Machine Learning, ICML, 2011, pp. 129–136.
[26] K.S. Tai, R. Socher, C.D. Manning, Improved semantic representations from tree-structured long short-term memory networks, in: Proceedings of the Association for Computational Linguistics, ACL, 2015.
[27] K. Cho, B. van Merrienboer, D. Bahdanau, Y. Bengio, On the properties of neural machine translation: Encoder-decoder approaches, in: Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, 2014.
[28] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Gated feedback recurrent neural networks, in: Proceedings of the 32nd International Conference on Machine Learning, 2015, pp. 2067–2075.
[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Proceedings of the Conference on Neural Information Processing Systems, NIPS, 2017.
[30] J. Devlin, M. Chang, K. Lee, K. Toutanova, 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, 2019, pp. 4171–4186.
[31] X. Chen, C. Liu, D. Song, Tree-to-tree neural networks for program translation, in: Proceedings of the Conference on Neural Information Processing Systems, NIPS, 2018.
[32] M. Ahmed, J. Islam, M.R. Samee, R.E. Mercer, Identifying protein-protein interaction using tree LSTM and structured attention, in: 2019 IEEE 13th International Conference on Semantic Computing, ICSC, IEEE, 2019, pp. 224–231.
[33] G. Louppe, K. Cho, C. Becot, K. Cranmer, QCD-aware recursive neural networks for jet physics, J. High Energy Phys. 2019 (1) (2019) 1–23.
[34] D. Kingma, J. Ba, Adam: A method for stochastic optimization, in: Proceedings of the 3rd International Conference for Learning Representations, 2015.