Recursive Neural Networks for the Automatic Analysis of STEP Files

VICTORIA SHEENA MILES

Recursive Neural Networks for the Automatic Analysis of STEP Files

Victoria Miles

A Thesis presented for the degree of
Doctor of Philosophy

Department of Engineering
University of Durham
UK

April 2024

Recursive Neural Networks for the Automatic Analysis of STEP Files

Victoria Miles

Submitted for the degree of Doctor of Philosophy
April 2024

Abstract

Intelligent interpretation of three-dimensional data represents a complex problem for artificial intelligence models. This is a relatively young, relatively small field of research, in which a key consideration is the format of the input data. This thesis proposes an entirely novel approach, in which natural language processing techniques are applied directly to CAD (computer aided design) models in the STEP file format. The research presented here represents the early development of a unique and flexible method for intelligent extraction of complex information directly from CAD models, without requiring any transformation away from the input data.

The primary contribution of this thesis is the development of a recursive encoder network capable of encoding the hierarchical data structures present in a STEP file into a single-vector representation. To investigate the potential of this approach, encoder-decoder models are trained to perform machining feature recognition, to estimate the values of geometric parameters, and to perform comparisons between pairs of CAD models. Further analysis is performed into the output representations of the encoder network, demonstrating the successful encoding of information relating to shape and dimensions.

Declaration

The work in this thesis is based on research carried out at the Department of Engineering, University of Durham, UK. No part of this thesis has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

Acknowledgements

I would like to start by acknowledging EPSRC, for supporting this research financially.

The first and most important thanks have to go out to my supervisors, Stefano Giani, Oliver Vogt and Raheleh Kafieh, whose endless support and consistent enthusiasm for this project have made the whole process of completing a PhD feel smooth and easy throughout. I would like to extend particular thanks to Stefano, who first introduced me to the world of AI, who suggested this project to me, and who I can't imagine having carried out this (or any) PhD without.

I would also like to thank the two masters students who I worked with during this project, Connor Cruicks and Theodore Russell, for their hard work, and for providing second opinions on my code. Thanks also to Benedict Jones, whose thesis template saved me a lot of time and effort.

To the Durham engineering PGR community: you are all lovely people and thank you for sometimes dragging me away from my computer and making me socialise. Special thanks to Emilia Russell and Theo McCarthy; Mondays spent with you definitely helped keep me sane towards the end.

All my thanks go to my family for their love and support and for always cheering me on. And finally, thank you to Heather, who has very patiently put up with me for the last few months while I was writing this thesis.

Contents

List of Figures

List of Tables

List of Publications

[1] V. Miles, S. Giani, and O. Vogt, “Recursive encoder network for the automatic analysis of step files,” *Journal of Intelligent Manufacturing*, vol. 34, pp. 181–196, 2022.

[2] ——, “Approaching step file analysis as a language processing task: A robust and scale-invariant solution for machining feature recognition,” *Journal of Computational and Applied Mathematics*, vol. 427, p. 115166, 2023.

[3] V. Miles, S. Giani, O. Vogt, and R. Kafieh, “Recursive autoencoder network for prediction of cad model parameters from step files,” *Procedia Computer Science*, vol. 232, pp. 3239–3246, 2024.

Acronyms

AI	Artificial Intelligence.
ASIN	Associatively Segmenting and Identifying Network.
B-rep	Boundary Representation.
BCE	Binary Cross-Entropy.
CAD	Computer Aided Design.
CNN	Convolutional Neural Network.
GRU	Gated Recurrent Unit.
ISO	International Organization for Standardization.
KL	Kullback–Leibler.
LSTM	Long Short-Term Memory.
MSE	Mean Squared Error.
NLP	Natural Language Processing.
PCA	Principal Component Analysis.
ReLU	Rectified Linear Unit.
RNN	Recurrent Neural Network.
SSD	Single Shot Multibox Detector.
STEP	Standard for the Exchange of Product Data.
STL	Stereolithography.
t-SNE	t-Distributed Stochastic Neighbor Embedding.

Chapter 1

Introduction

1.1 Background

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 in which artificial intelligence is used to aid in the design process by interfacing between Computer Aided Design (CAD) models and physical manufacturing processes [4].

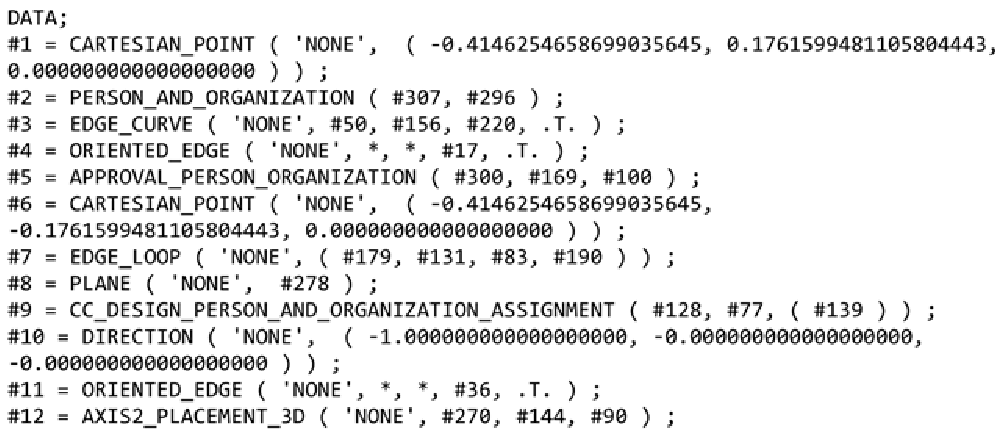
There are many potential applications for intelligent solutions which can effectively interpret 3D CAD models. Identifying the features present in a model can lead to increased automation between design and manufacturing processes. Smart analysis of databases of CAD models could be utilised in effectively 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.

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

In a rules-based approach [5, 6], 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 [7] and machine translation [8].

Learning-based analysis of 3D data, such as CAD models, has traditionally been dominated by methods which utilise 2D images of 3D shapes [9], 3D grids of voxels [10], or point cloud data [11]. Each of these methods ne-



```
DATA;
#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 ) ) ;
#11 = ORIENTED_EDGE ( 'NONE', *, *, #36, .T. ) ;
#12 = AXIS2_PLACEMENT_3D ( 'NONE', #270, #144, #90 ) ;
```

Figure 1.1: Segment of a STEP file.

cessitates 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. Such issues will have especially high impact when regarding a potentially complex 3D model part, in which small details may be of high significance. Work which does focus on analysis of model files directly [5, 6] tends to utilise rules-based techniques to extract features from the models and so is often tailored towards very specific applications.

One approach which has not yet been widely explored is the direct application of a neural network to a Standard for the Exchange of Product Data (STEP) file. STEP is an International Organization for Standardization (ISO) standard model format [12] 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.

Each line in a STEP file 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 segment of a STEP file containing several lines is shown in Figure 1.1.

A complex hierarchical structure is formed by connecting each STEP 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.

1.2 Research Objectives

The primary focus of the research presented in this thesis is the development of a novel approach to the automatic interpretation of 3D CAD models, using purely learning-based models which interface directly with CAD model files in the STEP format. A key research objective is to build a general framework for STEP file analysis, with high potential for adaptation to a wide range of applications. This primary objective can be broken down into several discrete research tasks:

1. The design of a learning-based encoder network which can take the information from a STEP file as direct input, and represent this information meaningfully in a single-vector output.
2. The design, training and evaluation of encoder-decoder networks, using the aforementioned encoder, to perform a range of tasks relating to the interpretation of 3D CAD models.
3. Analysis of the complexity and interpretability of encoder output vectors generated when training for different tasks.

Task 2, the development of models to perform specific tasks, can be further broken down into the three primary tasks considered:

1. Machining Feature Recognition: The identification of simple features, which may be manufactured with a single machining process, in a STEP file representing a CAD design.
2. Geometric Parameter Prediction: The prediction of given geometric parameters which define a machining feature, such as position and size.

3. CAD Model Comparison: The comparison of two CAD models, searching for general similarity or specific transformations.

1.3 Thesis Structure

The thesis structure is as follows:

Chapter 2 contains a general literature review, covering all topics relevant to this work.

Chapter 3 outlines key methodology which will be used throughout the thesis, including the design of the encoder network.

Chapter 4 is the first results chapter, covering the design, training and test-time performance for two models trained for machining feature recognition tasks.

Chapter 5 is the second results chapter, covering the design, training and test-time performance for a model trained for a geometric parameter prediction task.

Chapter 6 is the third results chapter, covering the design, training and test-time performance for two models trained for CAD model comparison tasks.

Chapter 7 is the final analysis chapter, presenting the results when various vector mapping techniques are applied to explore the output vector representations for models presented in the previous results chapters.

Chapter 8 outlines the main conclusions of this thesis and discusses the potential for further research using the tools developed in this work.

Chapter 2

Literature Review

2.1 Chapter Overview

In this chapter, existing research of relevance to the topics covered in this thesis will be discussed. As the focus of this thesis is the development of novel deep learning solutions for industrial applications, general background knowledge on the field of deep learning is assumed. Therefore, whilst this chapter will cover the specific technologies relevant to this thesis, fundamental concepts from the field of Artificial Intelligence (AI) and deep learning will not be expanded upon.

Section 2.2 covers the topic of AI approaches to the interpretation of 3D data. Section 2.3 gives a brief overview of the related field of Natural Language Processing (NLP), with a focus on the use of tree-structured networks. Section 2.4 briefly covers the topic of autoencoder architecture. Section 2.5 introduces the topic of attention mechanisms, again with some focus given to existing work which utilises attention in conjunction with a tree-structured network or tree-structured data.

2.2 3D Data Interpretation

2.2.1 Overview of AI Approaches to 3D Data

Meaningful analysis of 3D data is a task which poses significant challenges for AI 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.

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 (CNN) [7], 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 [9], 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. [13] present *multi-view CNN*, in which a view-pooling layer is used to intelligently combine information from multiple views into one classifier. This approach, known as the multi-view 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.

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 a regular 2D CNN network into 3D CNN 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 [10], in which 3D models are represented by binary grids of voxels and classified using a convolutional deep belief network. In [14], 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. 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. *OctNet* [15] attempts to address this issue by partitioning 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. Multi-view 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 point cloud approach [11, 16] improves on the resolution issue faced by the voxel approach by representing a 3D object more efficiently using selected points from surface faces. This more efficient method of representing model information leads to increased resolution and the retention of more relevant shape information. However, there remains a limitation to resolution based on the total number of points used to represent each model and the point cloud approach still necessitates a transformation of input data away from a real model format when handling CAD models as input.

2.2.2 Machining Feature Recognition

Learning-Based Approaches

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. Figure 2.1 shows examples of 24 classes of machining feature, which will be referenced throughout this thesis.

There are existing machining feature recognition models based on each of the approaches outlined above.

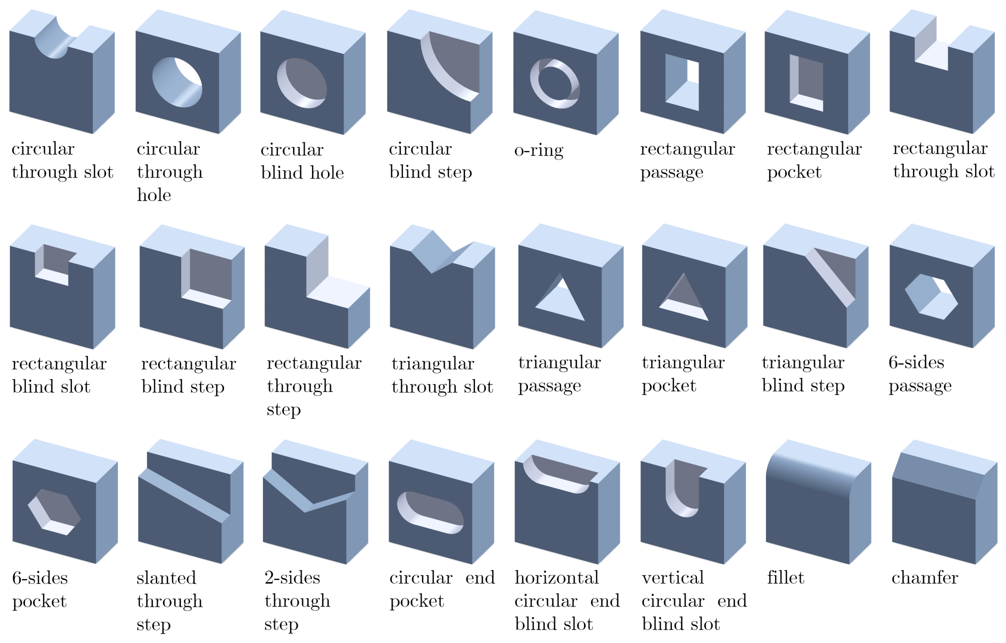


Figure 2.1: 24 classes of machining features, as suggested by [1].

FeatureNet [1] 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 [2], 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 [3], which presented SsdNet, in which segmentation and feature recognition are combined into a single process based on the Single Shot Multibox Detector (SSD) [17].

In [18], PointNet++ [16], a hierarchical network which makes use of point cloud data, is used to perform both single-feature classification and multi-feature recognition. In [19], 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 [20], 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 [21], information regarding the surface geometry and face topology is combined into a hierarchical data format, which is interpreted using a graph neural network.

Rules-Based Approaches

The potential advantages of developing an intelligent 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* [22] 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. In [5] b-spline surface features are recognised using features extracted from a STEP file. [6] extracts information about spot welding features. In [23], features relevant to the V-bending process are recognised. All use rules-based approaches to extract the relevant features.

Even more general solutions, such as the feature recognition system for rotational parts presented in [24], 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.

2.3 Natural 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, 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 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 Network (RNN). 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 implementation of more complex variants on the RNN cell, such as the Long Short-Term Memory (LSTM) cell [25] and Gated Recurrent Unit (GRU) cell improved on traditional RNN architecture by introducing systems for storing long term information. This addressed 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 in [26], 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 in [27]. 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 [28, 8, 29] and transformer [30, 31] 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. [32] adapted the network proposed in [27] 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.

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.

2.4 Autoencoders

The autoencoder [33] is a class of neural network typically trained to compress complex input data into meaningful compact forms, from which the source data can be reconstructed as completely as possible.

A traditional autoencoder architecture consists of mirrored encoder and decoder networks. The encoder compresses the input data into a compact form and the decoder attempts to use these encodings to reproduce the input data. These models can be trained in an unsupervised manner, with the goal of training being to encode the inputs in such a way that no information is lost and each data point can be perfectly reconstructed by the decoder. Thus, there is no need for labelled training data, and the representations learned by the network can contain meaning more complex than that given by simple data labels and therefore be applied to a wide variety of applications, including classification [34], generation [35] and dimensionality reduction [36].

2.5 Attention Mechanisms

The term attention mechanism refers to the use of an algorithm designed to prioritise input information based on importance, giving more weight to more important information when processing data. Attention mechanisms have been developed for use with natural language processing models in order to address some of the limitations of more traditional encoder-decoder architectures. Namely, the compression of all information from an input sentence into a single fixed-length vector necessarily results in some loss of information and provides no possibility of alignment between input and output sentences when performing tasks such as translation [37].

The application of attention mechanisms to attach priority to input data points can result in greater efficiency and higher performance in deep learning models, as the model is able to focus only on the more important information and ignore less useful parts of the input. In addition, these mechanisms have the potential to be used to improve interpretability of neural network behaviour, through the identification of which parts of the input data are heavily relied on to perform a particular task [38].

Attention mechanisms are commonly applied to sequence-to-sequence tasks, such as machine translation, where attention is applied to elements of the source sequence with relation to the target sequence, resulting in soft-alignment between source and target. In [39], an attention mechanism is used to identify relevant parts of an input sentence for generation of the next word when performing a machine translation task. This architecture is further developed in [40] through the comparison of global and local attention mechanisms. In [41], an attentive decoder network generates output characters for a voice recognition task.

In [42], the attentive pooling mechanism is presented, for the comparison of two inputs. Here, attention is applied to two inputs simultaneously, with respect to each other, and used to generate a similarity score. Thus, two inputs are directly compared to each other, with the benefit of importance scores for each token of each input when comparing to the other input.

Another common form of attention mechanism is self-attention, referring to models where the attention mechanism considers the significance of each token in an input sequence, with respect to every other token in the same sequence, thus using attention not to align two sequences, but to maximise the information retained from one [37]. In [43], the self-attention mechanism is utilised at the word and sentence level of a hierarchical input data structure, to perform the task of sentence classification. Transformer models, first presented in [30], are models where the traditional RNN architecture is entirely removed, relying instead only on the self-attention mechanism to perform NLP tasks.

The application of an attention mechanism to tree-structured input data increases complexity significantly, as attention must be applied not only to every token in a string but to every node in a potentially complex hierarchical structure. As is the case with general research into AI applications to tree-structured data, there is also a significantly lower volume of research carried out into the application of attention mechanisms for tree structured data than for sequential data, and most existing work focuses on NLP applications.

In [44], a sequential RNN is used in combination with an attentive recursive neural network, to guide the encoding of the hierarchical structure of a sentence, for application to various NLP tasks, with competitive results. [45] utilises an attention mechanism to perform alignment between regions of an image and words in the associated caption, before performing a sentiment classification task using a tree-structured LSTM network.

[46] presents an attention-based treeLSTM model for sentence summarisation. Here, block alignment is utilised; input sentences are split into semantic blocks, and each block simplified in a learned manner.

In [47], Ahmed et al present two attention-based variants of treeLSTM, with competitive results achieved on a semantic relatedness task. In [48] Ahmed et al present a further model: a tree-structured transformer network, which achieves competitive results across a range of tasks.

An attention mechanism is applied to a GRU based recursive neural network in [49], to perform rumour detection using tree-structured twitter replies as input data. This approach achieved state-of-the-art results, and demonstrated that accurate predictions relied heavily on the identification of key replies, using the attention mechanism.

2.6 Chapter Summary

Existing research into automatic interpretation of 3D CAD data generally falls into one of two categories. Either rules-based feature extraction is performed before any intelligent solutions are applied, or the input data is transformed away from the actual model data into a form which can be more easily interpreted by a neural network. Little existing research seeks to apply deep learning models directly to model data in its original forms, such as the tree-structured data contained in a STEP file.

The recursive neural network is a deep learning model, initially presented with a primary focus on tasks relating to natural language processing. With the success of simpler models in the field of NLP, such as RNN and transformer architectures, further research into the potential of the recursive neural network has been limited, although some later research implies the high potential of this architecture when applied to naturally tree-structured data.

This research is motivated by the aforementioned gaps in the literature; limited research into direct application of neural networks to CAD model data and limited research into the potential of the recursive neural network.

Chapter 3

Methodology

3.1 Chapter Overview

In this chapter, methodology common to all subsequent chapters will be presented.

Section 3.2 covers the generation and formation of CAD model datasets, used for the training and testing of all models developed here. Section 3.3 outlines the development of a STEP file parser, used as a pre-processing step for all work presented in this thesis. Section 3.4 describes the design of the recursive encoder network, which forms the encoder half of all encoder-decoder models developed in this work. Finally, Section 3.5 outlines the vector mapping techniques used to analyse the encoded vectors produced as output by the encoder network. The results produced using these techniques will be presented in Chapter 7, for models trained on several tasks.

Chapters 4, 5 and 6 will present some further methodology, covering the design of various decoder networks to be paired with the encoder to perform several tasks. However, the methods outlined in this chapter represent the core work presented in this thesis.

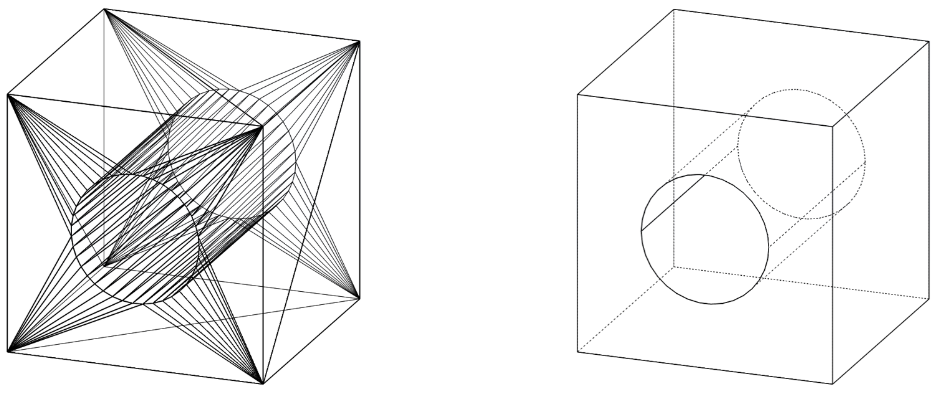
3.2 Datasets

One challenge when developing a learning-based model to work directly on STEP files as input is the limited availability of suitable training data. Large-scale 3D datasets typically take the form of general shape data commonly used in classification tasks [50, 10]. Whilst various benchmark datasets of CAD part designs do exist, most are available only in point cloud format or mesh formats such as Stereolithography (STL) [1]. These formats are suitable for use with the voxel, point-cloud and multi-view formats utilised by most existing 3D AI models. However, the conversion into STEP format is not straightforward and requires manual input for each model. This is because mesh formats represent curves by placing many points along a circle and using these points to create triangular faces to approximate the curve, whereas the STEP format more simply and accurately represents model geometry using straight lines and circles, as illustrated in Figure 3.1. Therefore, in order to translate to an accurate STEP file, any curves in a model must be manually replaced with actual circles using CAD software, making the conversion of any large-scale dataset impractical.

For this reason, all datasets required for this research have been newly generated, rather than using existing benchmark datasets.

3.2.1 Dataset Format

The datasets generated for this research make use of the set of 24 machining features proposed in [1], as shown in Figure 2.1. These machining features are simple manufacturing features which can be added to a starting block in order to produce simple or increasingly complex CAD geometries as more features are added to the same 3D part design. Models in the machining



(a) STL format (b) STEP format

Figure 3.1: Faces created for an example model when loaded into CAD software from STL and STEP format

features datasets are generated by creating a starting block of fixed dimensions and then randomly adding one or more machining features, with various parameters randomised, to produce unique CAD models.

This dataset framework was selected for several reasons. Firstly, the task of machining feature recognition is a key task in 3D CAD model analysis, and requires a machining features dataset. Models designed to perform this task will be presented in this thesis, in Chapter 4, and so a machining features dataset was necessary. Using the same feature set and generation parameters as used in the benchmark datasets provided by [1] and [2], additionally allows for direct performance comparisons between our model and existing solutions.

For tasks outside of machining feature recognition, such as those discussed in Chapters 5 and 6, other datasets could be used. However, continued use of the machining features dataset provides several advantages. Machining features can be randomly added to a starting block, allowing for automated generation of data with randomised parameters. Datasets of a chosen size with chosen numbers of features and selected constraints can be simply generated with any relevant details labelled and no manual part design is necessary. Barring access to a large-scale labelled dataset of real part designs, this is the most convenient method of producing a large enough dataset to train our learning-based models. Additionally, using the same dataset format

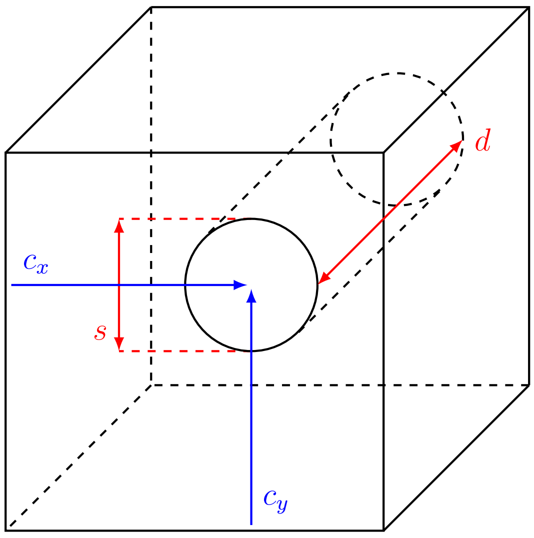


Figure 3.2: Illustration of the geometric parameters used to define features, using the circular through hole class as an example; c_x and c_y are coordinates of the feature centre, s is the size of the feature when viewed from the front face, d is the depth of the feature into the block.

throughout provides a degree of continuity to this research and allows for direct and meaningful comparison of the encoded output vectors produced by the encoder network, as all models have been trained using similar data.

The necessary parameters to define any given feature are the feature class, size, depth and two coordinate values representing the position of the centre of the feature. These parameters are visualised in Figure 3.2.

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

As several different models have been developed as part of this research and trained to perform various tasks, several distinct datasets were required. The characteristics of each dataset generated are shown in Table 3.1.

Table 3.1: Parameters for each dataset generated.

Dataset	Multi-Feature	Size	Balanced	Parameter Labels	Fixed Parameters
1	✗	24,000	✓	✗	✗
2 (1)	✓	2,000	✗	✗	✗
2 (2)	✓	1,000	✗	✗	✗
3 (1)	✗	9,600	✓	✓	✗
3 (2)	✗	16,698	✗	✓	✓

Dataset 1: Single-Feature Dataset for Feature Recognition

The dataset generated for the machining feature classification task is comprised of models of blocks, each with a single machining feature added, designed to be mostly equivalent to the benchmark dataset provided by [1]. The starting block for each CAD model 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. A machining feature is added to a random face of this base block , 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.

Dataset 2: Multi-Feature Dataset for Feature Recognition

The dataset generated to train for the multi-feature recognition task, Dataset 2(1), was designed to contain models exactly equivalent to those present in the benchmark dataset created in [2], to allow for direct comparison between our model and existing solutions. Each model in the multi-feature dataset consists of a standard base cube with side length 10cm, 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. The scale of the models and dimension limits for features are based on those included in the benchmark dataset presented in [2].

Additional datasets were generated to test the multi-feature recognition model. These datasets are shown combined in Table 3.1 as Dataset 2(2).

Dataset 3: Single-Feature Datasets Labelled with Geometric Parameters

These datasets were designed to be more standardised versions of the single-feature dataset, with feature parameters labelled to allow for prediction of geometric parameters, comparison of CAD model similarity and exploration of encoder output vectors in the latent space. Each CAD model in these datasets is built from a starting block which is a uniform cube, of side length 10 units. As the goal is to assess CAD models based on geometric shape, and all models are normalised through the use of an identical starting block, real units are ignored, with the focus being on geometric parameters relative to the overall model size. One machining feature is added to each starting block.

Two labelled datasets have been generated:

> Dataset 3(1) (randomised): A dataset where geometric parameters are randomised within fixed parameters, for training and testing of neural networks.
>
> Dataset 3(2) (fixed parameters): A dataset where fixed values are selected for geometric parameters, for exploring the latent space and investigating encoder output.

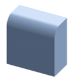
3.2.2 Dataset Generation

All datasets are generated using a SolidWorks macro, written in Visual Basic. To generate a new model, a base block is first created by sketching a rectangle and extruding and then one or more machining features are added, with parameters either randomised or selected. To add a feature, a sketch is inserted on a chosen face, with geometry of the feature specified using circles (defined using the position of the centre point and a point on the circum-

Table 3.2: Parameters necessary to generate each class of machining feature.

	Feature Class	Size	Depth	Coord 1	Coord 2
	Circular through slot	✓	✗	✓	✗
	Circular through hole	✓	✗	✓	✓
	Circular blind hole	✓	✓	✓	✓
	Circular blind step	✓	✓	✗	✗
	O-ring	✓	✓	✓	✓
	Rectangular through hole	✓	✗	✓	✓
	Rectangular blind hole	✓	✓	✓	✓
	Rectangular through slot	✓	✗	✓	✗
	Rectangular blind slot	✓	✓	✓	✗
	Rectangular blind step	✓	✓	✗	✗
	Rectangular through step	✓	✗	✗	✗
	Triangular through slot	✓	✗	✓	✗
	Triangular through hole	✓	✗	✓	✓
	Triangular blind hole	✓	✓	✓	✓
	Triangular blind step	✓	✓	✗	✗
	6-sides passage	✓	✗	✓	✓
	6-sides pocket	✓	✓	✓	✓
	Slanted through step	✓	✗	✗	✗
	2-sides through step	✓	✗	✗	✗

Table 3.2: Parameters necessary to generate each class of machining feature.

	Feature Class	Size	Depth	Coord 1	Coord 2
	Circular end passage	✓	✗	✓	✓
	Horizontal circular end blind slot	✓	✓	✓	✗
	Vertical circular end blind slot	✓	✓	✓	✗
	Fillet	✓	✗	✗	✗
	Chamfer	✓	✗	✗	✗

ference), lines (defined between two points) and tangent arcs (segments of circle defined using start and end points and an arc type). When the sketch is complete, an extruding cut is performed, either through the entire block or to a chosen depth, depending on the feature class. CAD models are exported in STEP and STL formats and image files are also saved, as an easy way to view the dataset.

Face Selection

To generate sufficiently varied models, the first variable which must be randomised or selected is the face of the base block on which to sketch the chosen feature. Face selection is performed using a ray, with starting point at the centre of the base block. The direction of the ray will be one of the cardinal directions (x, y or z) and either positive or negative, with the choice of direction based on a value between 1 and 6 which may be either randomised or selected. The face which the ray first hits will be chosen for the feature sketch, and so any of the six faces of the cube may be selected.

Sketching Holes

The hole category represents all feature classes in which the feature sketch does not interact with any of the edges of face on which it is added. The full list of hole classes is as follows:

- Circular through hole
- Circular blind hole
- O-ring
- Rectangular passage
- Rectangular pocket
- Triangular passage
- Triangular pocket
- 6-Sides passage
- 6-Sides pocket
- Circular end passage

Hole position is defined by a two-dimensional coordinate point, (x, y), representing the position of the centre of the feature, relative to the plane of the face on which the sketch is added. Size of the sketch is defined using a single parameter, s, representing width and height, as hole sketches are constrained to fit within a square bounding box. The exception to this rule is the circular end passage class, which is fixed to have straight lines of length $\frac{s}{2}$, meaning that the overall shape will have dimensions $(\frac{s}{2}, s)$.

Circular hole sketches consist of a single circle, rectangles of four lines connecting the corner points and the sketch for the circular end passage class consists of two lines, connected using two tangent arcs. Rectangles and circular end passages are always created with the lines parallel to edges of the base block. For triangular and hexagonal holes, the appropriate number of points are selected from a circle with centre (x, y) and diameter s for hexagons, $\frac{2\sqrt{3}}{3}s$ for triangles, giving side length of s. Triangles are always equilateral and hexagons are always regular. Triangle coordinate points are set as:

$$(x_i, y_i) = (x + \frac{\sqrt{3}}{3} \cdot s \cdot \cos\phi, \quad y + \frac{\sqrt{3}}{3} \cdot s \cdot \sin\phi) \tag{3.2.1}$$

$$\phi = \left\{\theta, \theta + \frac{2\pi}{3}, \theta + \frac{4\pi}{3}\right\} \tag{3.2.2}$$

and hexagon coordinate points are set as:

$$(x_i, y_i) = (x + \frac{1}{2} \cdot s \cdot \cos\phi, \quad y + \frac{1}{2} \cdot s \cdot \sin\phi) \tag{3.2.3}$$

$$\phi = \left\{\theta, \theta + \frac{\pi}{3}, \theta + \frac{2\pi}{3}, \ldots, \theta + \frac{5\pi}{3}\right\} \tag{3.2.4}$$

for some starting angle θ. For both the single-feature and multi-feature datasets used to train for machining feature recognition, θ is randomised to increase the variability of CAD models in the dataset. For the fully labelled datasets, it was decided that random rotation of features would add unnecessary complication to comparisons of similar CAD models. Therefore, for these datasets, θ was set to $\frac{\pi}{2}$ when generating triangles and 0 when generating hexagons, ensuring that one side of the triangle and two sides of the hexagon will always be parallel to one edge of the base cube.

When labelling the position of hole features, distance to the feature centre is measured from the closest corner, with the shorter distance labelled as the first position parameter (c_x) and the further distance as the second position parameter (c_y).

Sketching Slots

The slot category represents all feature classes in which the feature sketch interacts with one edge of the face on which it is added. The full list of slot classes is as follows:

- Circular through slot
- Rectangular through slot

- Rectangular blind slot
- Triangular through slot
- Horizontal circular end blind slot
- Vertical circular end blind slot

Slot position is defined by one randomised or selected value between 0 and 4, which selects which edge of the selected face the slot should be added to, and one randomised or selected value representing distance of the centre of the slot along the chosen edge. For circular slots, the centre of the circle is placed on the edge. For rectangular slots, a square sketch is made, with one edge aligning with the existing edge. For triangular slots, a triangle is sketched with a side of length s aligning with the existing edge and height is also set to s. Size of the circular end shape sketched is the same as for the circular end passage class, with the edge intersecting the centre of the rounded sections for the horizontal class and removing one rounded section for the vertical class.

When labelling the position of slot features, shortest distance from a corner of the chosen edge to the centre of the feature is measured and labelled as the first position parameter (c_x) and the second position parameter is not used.

Sketching Corners

The corner category represents all feature classes other than the fillet class in which the feature sketch interacts with exactly two edges of the face on which it is added. The full list of corner classes is as follows:

- Circular blind step
- Rectangular blind step
- Rectangular through step
- Triangular blind step

- Chamfer

Corner position is defined by one randomised or selected value between 0 and 4, which selects which corner of the selected face the corner feature should be added to. For circular corners, the centre of the circle is placed on the corner. For rectangular corners, a square sketch is made with two edges aligning with edges of the base block. For triangular corners (triangular blind step and chamfer), a triangle is sketched with points on the corner and a distance equal to s away from the corner in both directions.

No parameter is labelled for corner position, as blocks with corner features of identical size placed on any corner will be functionally identical, only differing due to rotation.

Sketching Slanted and 2-Sides Through Steps

The slanted and two-sides through step classes do not fit easily into any one of the categories previously described. These classes are clearly most similar to each other. However, there is also a strong similarity between these classes and the rectangular through step class. Therefore, while these features must be generated using a very different method, it was decided to label the parameters of the slanted and two-sides through step classes to align with the labels given to the rectangular through step class.

These classes are defined using a randomised or selected value between 0 and 4, which selects an edge which will be used as the "base" of the sketch. Two further values are randomised or selected, representing the two distances from the base edge for the two sides of the slanted through step and the centre and edge distances for the two-sides through step.

When labelling the position of these features, the two distances from the base edge are recorded, with the shorter distance labelled as the first position parameter (c_x) and the further distance labelled as the second position parameter (c_y). The depth the sketch is extruded to is labelled as the size (s), to align these two classes more strongly with the labelling of corner classes, especially the rectangular through step.

Sketching Fillets

The fillet class is a corner class in terms of shape and labelling. However, it is generated using a different method to any other feature class in the dataset. An edge is selected to have a fillet added using a randomised or selected value between 1 and 12. Similarly to the initial face selection, a ray from within the block is used to choose the edge based on this parameter. A fillet feature is then added to the chosen edge, with diameter equal to the randomised or selected feature size, with no need for a sketch.

As with other corner classes, no position parameter needs to be labelled, and only a size parameter is recorded. For all purposes other than dataset generation, fillets are considered part of the corner category.

Adding Multiple Machining Features

To produce the multi-feature dataset, between 2 and 10 machining features are added sequentially to the same starting block. All size and position parameters, as well as feature class are randomised for each feature individually.

This random process can lead to models which are unsuitable or incorrectly labelled. For instance, if a small feature is randomly added entirely inside a larger feature, the smaller feature ceases to exist, resulting in a STEP file labelled with one more feature than is actually present in the CAD model. Alternatively, adding additional features to a model can transform a blind feature into a through feature, resulting in a discrepancy between labels and actual model geometry. To remove this error, an additional manual data labelling phase is necessary for the multi-feature dataset, to remove inappropriate models and ensure that labels are accurate. Models with high intersection of features to the extent that only part of a feature is visible are allowed, as long as each feature is in part visible and is distinct from other features.

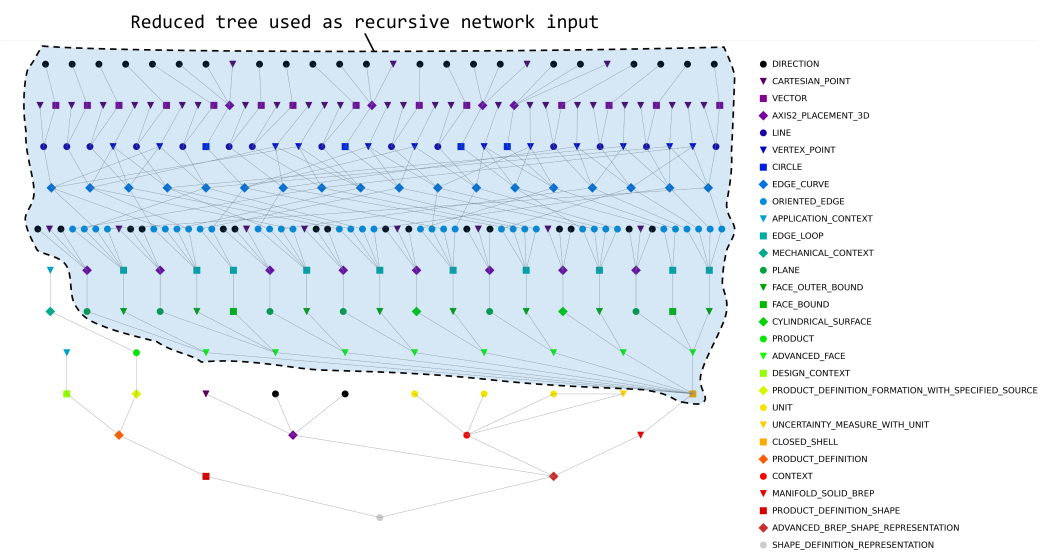


Figure 3.3: Typical hierarchical structure representing the geometric data for a simple CAD model; connections show how low-level features are combined to form complex shape representations, the shaded region represents the portion of the tree which will be used as input for the neural network.

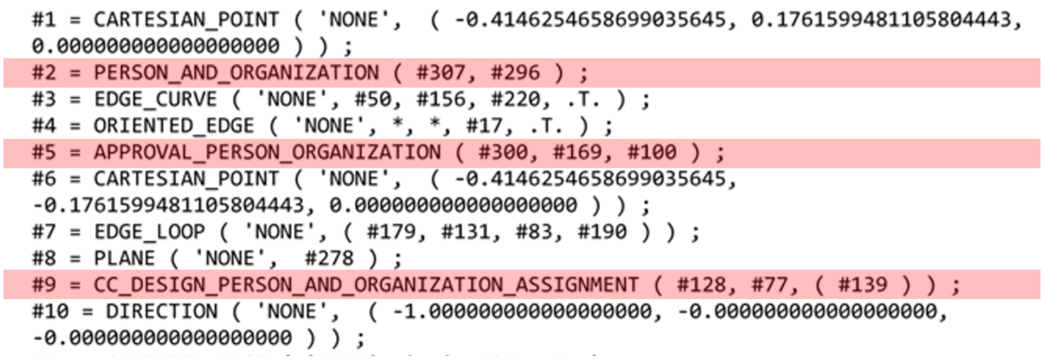
3.3 Parsing STEP Files

As shown in Figure 3.3, STEP files [12] 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.

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.

3.3.1 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 Figure 3.4. Each line is assigned a unique ID number, preceded by "#", which starts the line. The information contained in the line consists of a



```
#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 ) ) ;
```

Figure 3.4: Sample data lines from a STEP file; lines which do not contain geometric information are highlighted.

category and any number of parameters which may be the IDs of other lines, coordinate point values or additional flags. For example, in Figure 3.4, the first line has ID number 1, category CARTESIAN_POINT and takes three coordinate values as parameters (-0.41.., 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 [28, 8] or transformer [30, 31] 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 Figure 3.3, 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 Section 3.3.4.

3.3.2 Filtering out Irrelevant Information

Not every line in a STEP file contains useful geometric information, as can be seen in Figure 3.4. Many nodes, such as the "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 Figure 3.3 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 Figure 3.3. Throughout this research, the choice was made to set "CLOSED_SHELL" as the top node in order to keep the input data trees as simple as possible.

3.3.3 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.

3.3.4 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 Figure 3.5, using the first coordinate value seen in Figure 3.4.

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.

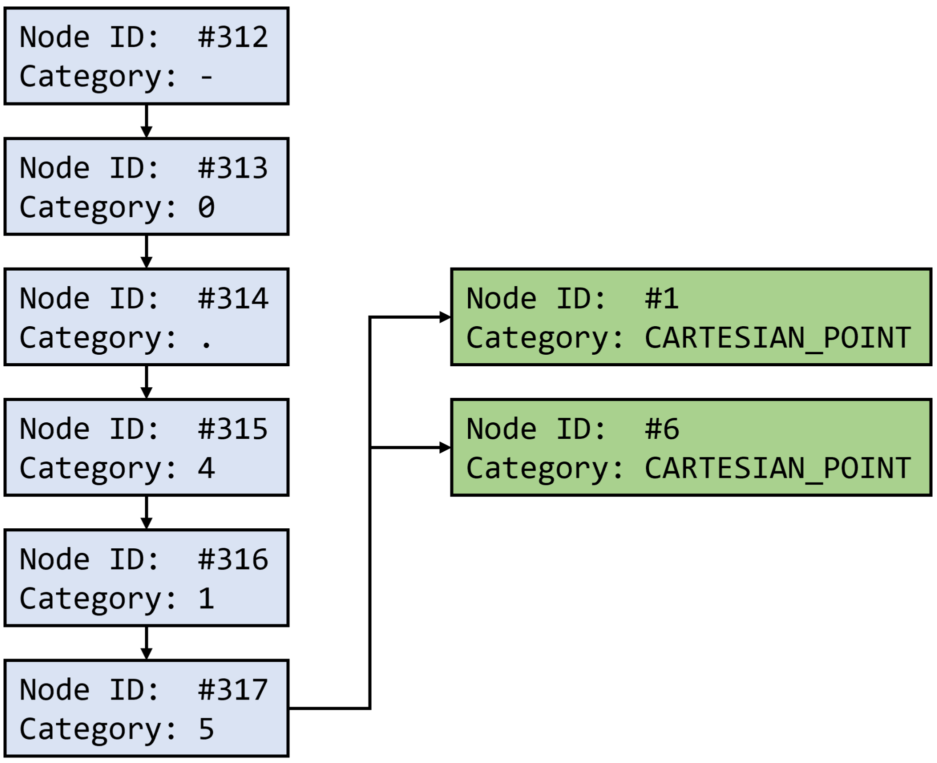


Figure 3.5: The node string produced to represent the value −0.415, the first coordinate value seen in Figure 3.4 when taken to three decimal places, with parent-child connections shown to the two lines in Figure 3.4 which reference this value

.

3.4 Recursive Encoder Network

In this section a recursive encoder network for the extraction of meaningful information from data trees generated by the STEP parser will be introduced. The output of the encoder network for any given input STEP file is a single vector of fixed length, which can then be fed into various decoder architectures in order to perform specific tasks.

3.4.1 TreeLSTM Network Overview

The recursive encoder makes use of the Child-Sum TreeLSTM network first presented in [27]. 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 tree-LSTM 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 3.6 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 in [27] learns not only the weights necessary to analyse the input tree but also those necessary to determine the correct

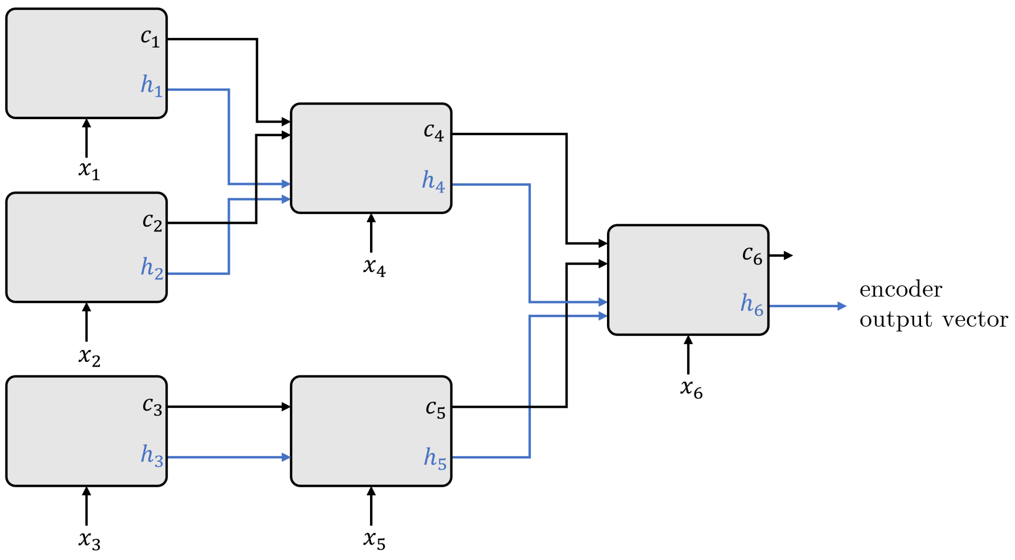


Figure 3.6: Example application of the tree-LSTM cell to a simple tree structure, where c is the output cell state, h the output hidden state and x the new input for each node.

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.

3.4.2 The Tree-LSTM Cell

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 new cell state. The output gate controls what information should be written to the new 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 sets of weights which produce meaningful outputs for any given input.

The key difference between a standard LSTM cell and a tree-LSTM 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 tree-LSTM cell must be able to take multiple previous states, as a node in the tree can have any number of children. Figure 3.6 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 3.7 shows the processes within a tree-LSTM 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:

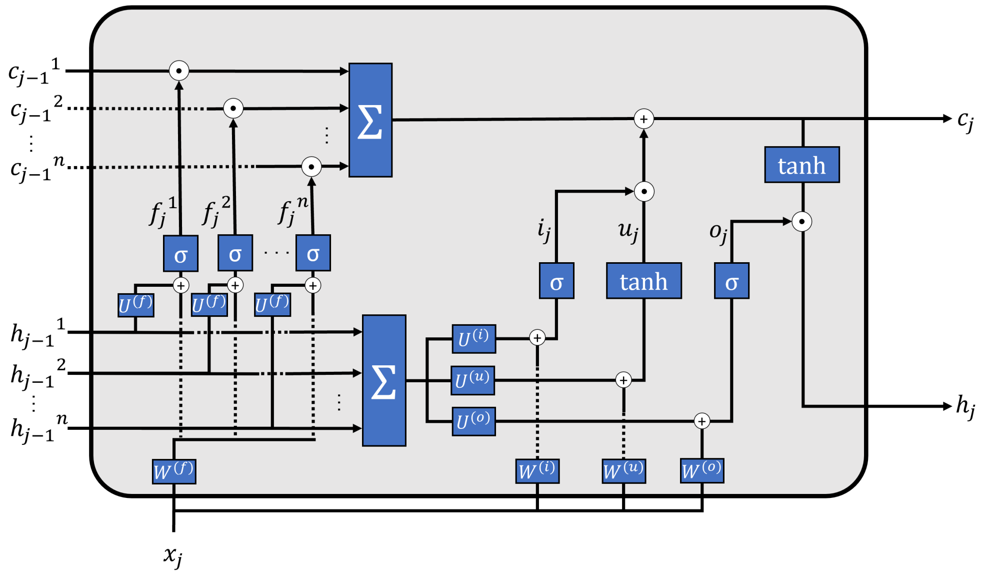


Figure 3.7: The processes within a tree-LSTM cell; c is cell state, h is hidden state, x is input vector, U and W are weights and f, i and o are forget, input and output gates respectively

$$\tilde{h}_j = \sum_{k \in C(j)} h_k \tag{3.4.5}$$

$$i_j = \sigma(W^{(i)}x_j + U^{(i)}\tilde{h}_j + b^{(i)}) \tag{3.4.6}$$

$$f_{jk} = \sigma(W^{(f)}x_j + U^{(f)}h_k + b^{(f)}) \tag{3.4.7}$$

$$o_j = \sigma(W^{(o)}x_j + U^{(o)}\tilde{h}_j + b^{(o)}) \tag{3.4.8}$$

$$u_j = tanh(W^{(u)}x_j + U^{(u)}\tilde{h}_j + b^{(u)}) \tag{3.4.9}$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k \tag{3.4.10}$$

$$h_j = o_j \odot tanh(c_j). \tag{3.4.11}$$

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.

Table 3.3: The dimensions of parameters within the recursive encoder cell, where n is the hidden size and k is the number of words in the dictionary.

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

3.4.3 Network Parameters

The hidden size of the encoder is set to a chosen fixed value, n, which will result in output cell and hidden states from each LSTM cell in the form of vectors also of length n. A suitable value of n must be selected to be large enough to encode sufficient complexity in the output vectors but also 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 n. The dimensions of inputs, outputs and learned parameters are given in Table 3.3.

3.5 Dimensionality Reduction

Dimensionality reduction refers to the process of transforming high-dimensional data into a low-dimensional form, whilst conserving the structure of the data, and minimising loss of useful information [51].

In this case, output vectors from the recursive encoder network have high dimensionality, with number of dimensions equal to the hidden size, n, which is between 200 and 700, depending on the model. This high number of dimensions leads to difficulty in interpreting the representations and understanding the structure of the feature space. Through the use of dimensionality reduction tools, these high-dimensional vectors can be transformed into two dimensional forms, thus allowing for human interpretation. If a model has successfully encoded the input STEP file into a meaningful vector representation, and dimensionality reduction is applied correctly, it should be possible to identify clustering and meaningful axes of transformation within the projected 2D feature space.

Two forms of dimensionality reduction are considered here: Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE). Both techniques are applied in python, using tools found in the scikit-learn package [52].

3.5.1 PCA Projections

Principal component analysis [53] is a linear approach to dimensionality reduction, where numbered principal components are selected to maximise variance. This effectively creates new axes on which to interpret the data, chosen to result in the greatest spread of data points.

Implementation of PCA, as described in [54], begins with calculation of the first principal component. For an n dimensional input, this is calculated as:

$$C_1 = \beta_{11}X_1 + \beta_{21}X_2 + ... + \beta_{n1}X_n \tag{3.5.12}$$

where C_1 is the first primary component, X_i is the ith element of the input vector and β values are weights, chosen to maximise variance within the dataset. This first principal component, through the maximisation of variance will contain the maximum possible amount of information about the source vector. However, it is likely that the full variance of the input dataset cannot be represented using a single dimension. Therefore, further principal components can be calculated, as:

$$C_j = \beta_{1j}X_1 + \beta_{2j}X_2 + ... + \beta_{nj}X_n \tag{3.5.13}$$

where C_j is the jth principal component and β values are chosen to maximise variance whilst ensuring that C_j is orthogonal to all previously calculated components.

For the purposes of data visualisation, it is common to limit PCA to the first two, or possibly three, principal components, allowing for 2D or 3D representations of vector positioning in the latent space, with the principal components as axes. However, it is possible to calculate up to n principal components, at which point all information from the source vector is fully represented in the output, and dimensions of the input and output vectors are the same.

The linear nature of PCA results in an understandable and navigable feature space, where principal components can be directly understood as linear combinations of input features. However, for data visualisation purposes this approach can be somewhat limited; PCA relies on the assumption that a linear interpretation of input vectors is sufficient to represent the entire latent space, with no possibility of more complex interpretation of the input data.

3.5.2 T-SNE Projections

t-SNE [55] is an iterative method for dimensionality reduction based on stochastic neighbour embedding [56], designed for visualisation of underlying structures present in high-dimensional data. The idea of t-SNE is to organise data according to distance between instances, resulting in two or three-dimensional visualisations where the distance between any two instances is as close as possible to the distance between these same instances in the original high-dimensional data.

The first step in applying t-SNE, as outlined in [54] is to measure the distances between all individual instances within the high-dimensional source data. These distances are then converted to probabilities, and stored in a coordinates matrix, x_h.

Following this, the first iteration of the two or three-dimensional representation is generated. This initialisation can be random or a linear dimensionality reduction technique, such as PCA can be first applied. In this work, t-SNE projections are initialised using PCA.

A second matrix of probabilities, x_l is then calculated, based on the distances between points in the new low-dimensional representation. With probability matrices generated for both representations of the data, the Kullback–Leibler (KL) divergence is calculated to compare the new low-dimensional representation to the original high-dimensional representation. KL divergence is calculated as:

$$KL\ divergence = \sum_{x \in X} x_h \log\left(\frac{x_h}{x_l}\right) \tag{3.5.14}$$

Subsequent iterations update the low-dimensional representation, with the goal of minimising KL divergence, thus producing a final representation where distances between instances are close to those seen in the original high-dimensional data.

Due to the iterative nature of t-SNE, the final representations are not based on understandable linear combinations of inout features, as was the case with PCA, resulting in a non-traversable feature space. However, this approach of organising data points according to distance typically results in superior visualisation of potentially complex underlying structures in the data.

3.6 Chapter Summary

This section has presented the key methodology which will be used throughout the remainder of this thesis.

The datasets described in Section 3.2 will be used for the training and testing of every model presented in this thesis, with the parsing algorithm outlined in Section 3.3 used to process the raw STEP files from these datasets into useful input for a neural network.

Various deep learning models will be presented in Chapters 4, 5 and 6, all using the recursive neural network presented in Section 3.4 as the encoder half of an encoder-decoder network.

Finally, the dimensionality reductions techniques described in Section 3.5 will be applied to several of these deep learning models in Chapter 7, to explore representation of the input STEP files in the vector space.

Chapter 4

Machining Feature Recognition

4.1 Chapter Overview

Machining features (or manufacturing features) are simple features, such as holes and slots, which can be produced via a single manufacturing process. Figure 2.1 shows the list of 24 classes of machining feature utilised throughout this thesis. Through the combination of several such 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.

In this chapter the recursive encoder network is incorporated into two models, performing the tasks of single-feature classification and multi-feature recognition. Section 4.2 describes the decoder architecture developed for each model. Section 4.3 outlines initial experiments performed using the single-feature classification model to evaluate the significance of input data precision and select a suitable approach to maximise effectiveness of the encoder. Results for both single-feature classification and multi-feature recognition are presented in Section 4.4, with results obtained for several existing solutions included for the sake of comparison.

Results presented in this chapter have been previously published in the *Journal of Intelligent Manufacturing* in a paper titled "Recursive encoder network for the automatic analysis of step files” [57], and the *Journal of Computational and Applied Mathematics* in a paper titled "Approaching step file analysis as a language processing task: A robust and scale-invariant solution for machining feature recognition” [58].

4.2 Model Architectures

For each of the tasks covered in this chapter, single-feature classification and multi-feature recognition, the recursive encoder network outlined in Chapter 3 is paired with a decoder network designed to perform the given task. This section will outline the development of these decoder networks.

4.2.1 Single-Feature Classification Model

The recursive encoder network extracts features from the data, in the form of an output vector of length n 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:

$$y_i = \sum_{j=1}^{n} x_j w_{ij} + b_i \tag{4.2.1}$$

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 normalised confidence scores for each class, the Softmax function is applied as follows:

$$Softmax(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{4.2.2}$$

for m classes, where p_i is the Softmax score 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:

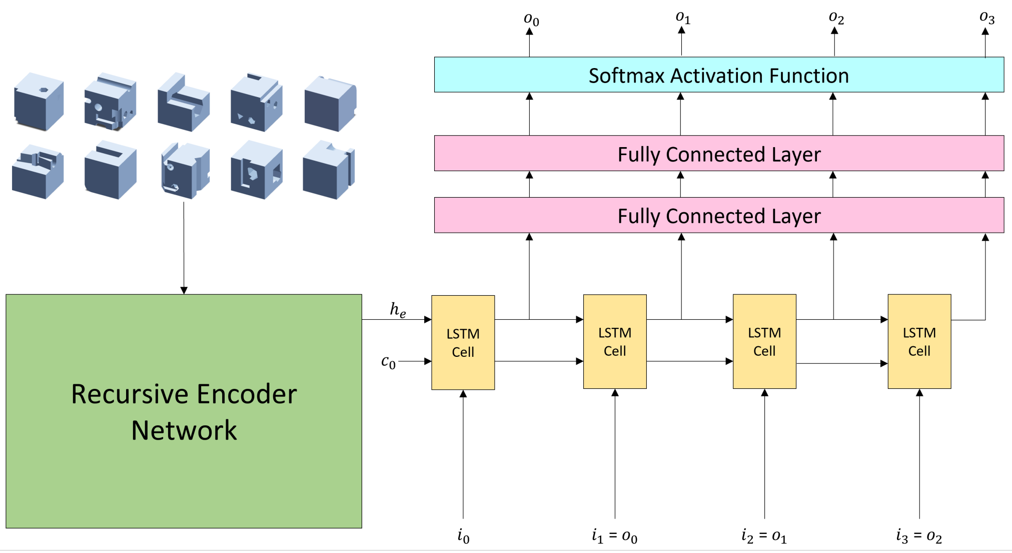
$$L_{CE} = -\sum_{i=1}^{m} t_i \log(p_i) \tag{4.2.3}$$

for m classes, where t_i is the ground truth and p_i the Softmax score for class i. The Adam optimisation algorithm [59], an adaption of stochastic gradient descent, is used during training.

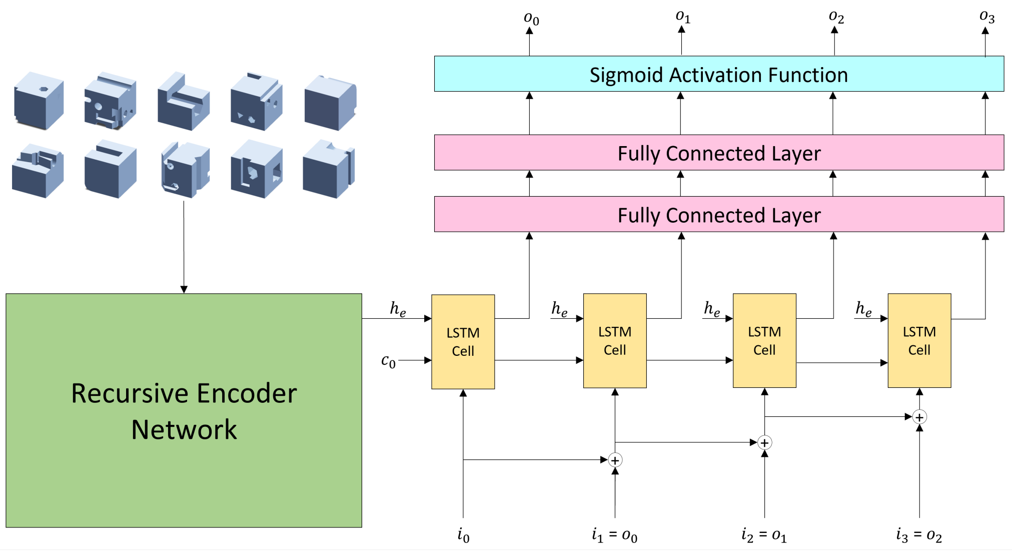
For the single-feature classification model, n is set as 300, resulting in a very small encoder-decoder model, with around 400,000 learned parameters.

4.2.2 Multi-Feature Recognition Model

The decoder network for the multi-feature recognition model makes use of the traditional LSTM cell. A simpler form of the tree-LSTM cell described in Chapter 3, the LSTM cell contains three controlling gates, the forget, input and output gates, and processes a linear string of inputs instead of a hierarchical structure.



(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.

Figure 4.1: 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.

Figure 4.1a 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 confidence score for each feature class 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.

Decoder Development and Ablation Study

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

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{4.2.4}$$

To predict multiple labels in one step, all classes with scores higher than a given threshold are predicted. During the training period, values ranging between 0.5 and 0.9 were trialled for the prediction threshold. Based on comparative train-time accuracy across the validation dataset, a threshold of 0.7 was deemed to be appropriate for output predictions, and so the output vector is converted into a binary vector as follows:

$$x = \begin{cases} 1 & \text{if } \sigma(x) \geq 0.7 \\ 0 & \text{if } \sigma(x) < 0.7 \end{cases} \tag{4.2.5}$$

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. Ideal behaviour of the model would be to predict all feature classes which appear in the CAD model in the first step of predictions, all classes which appear at least twice in the second step, and so on, to minimise the number of prediction steps necessary. 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.

Figure 4.1 shows the alterations made to the decoder network, with a traditional LSTM network displayed in Figure 4.1a and our adapted version in Figure 4.1b. Feature recognition accuracy achieved after each of these

Table 4.1: 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**

alterations was introduced, as well as for the traditional LSTM architecture, is presented in Table 4.1.

One advantage of our approach is the relative simplicity of the neural network architecture; the encoder network consists of a single tree-LSTM 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.

Figure 4.2 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:

$$precision = \frac{tp}{tp + fp} \tag{4.2.6}$$

$$recall = \frac{tp}{tp + fn} \tag{4.2.7}$$

$$fscore = \frac{2 \times precision \times recall}{precision + recall} \tag{4.2.8}$$

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

When calculating the F-score, predictions are not separated by class,

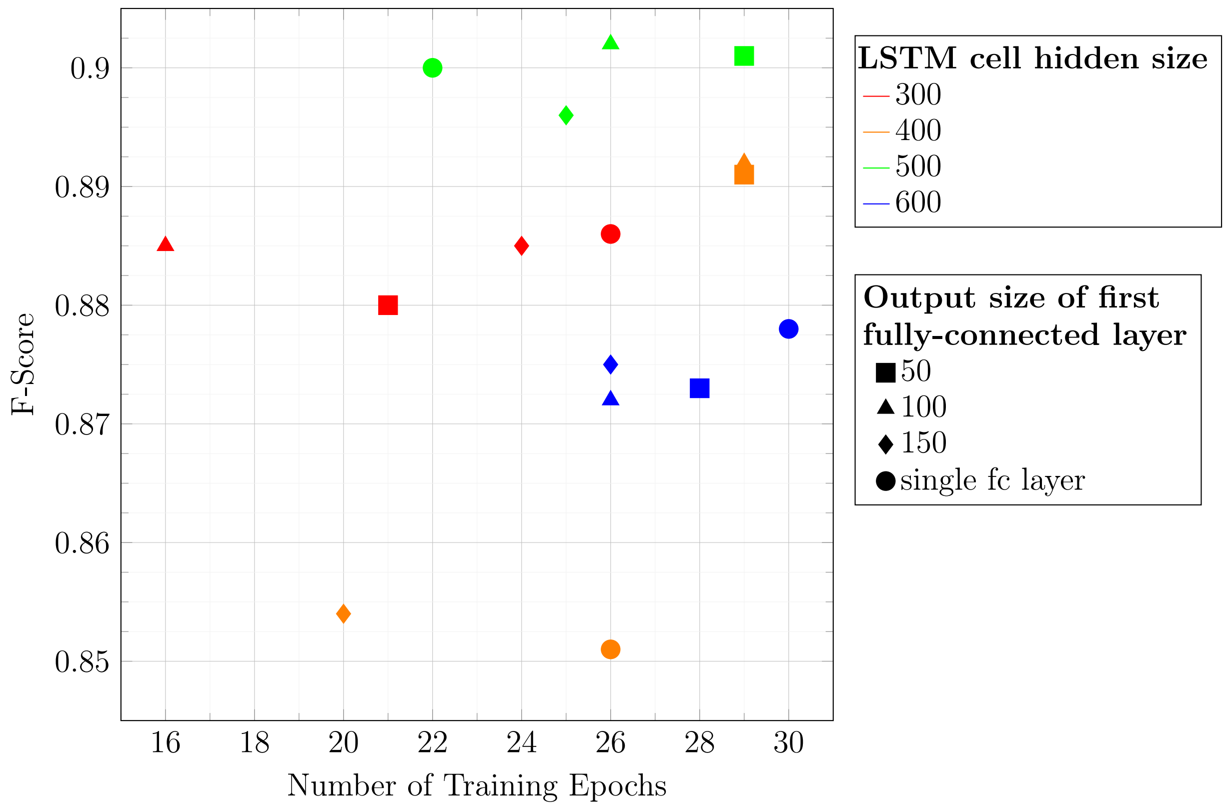


Figure 4.2: Validation F-scores plotted against number of training steps required for various model sizes.

instead calculating a single score for all feature class predictions across the entire dataset. For instance, for a dataset of size 2, with CAD model A which contains features with classes [1, 2, 3, 3] and CAD model B which contains features with classes [4, 4, 5], we would have a total ground truth consisting of 7 features. If the model were to make predictions of [1, 2, 3, 5] for A, and [4, 4, 5, 5] for B, we would have $tp = 6$ (3 for A and 3 for B), $fp = 2$ (1 for A and 1 for B and $fn = 1$ (for A). This would give a precision value of 0.75, a recall value of 0.86, and a final combined F-score for the dataset of 0.80.

It can be clearly seen from Figure 4.2 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 chapter, the model with hidden size 500 and intermediate fully connected size of 100 is used when presenting results.

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:

$$L = \frac{1}{n}\sum_{i=1}^{n} y_i \cdot \log x_i + (1 - y_i) \cdot \log (1 - x_i) \tag{4.2.9}$$

where n is the number of values in the model output, x_i is the i-th scalar value in the model output and y_i is the i-th 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 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, weights for both the encoder and decoder are learnt. Due to the pre-training step, decoder weights are learned from scratch and the encoder weights must merely be fine-tuned.

4.3 Evaluation of the Significance of Coordinate Value Precision

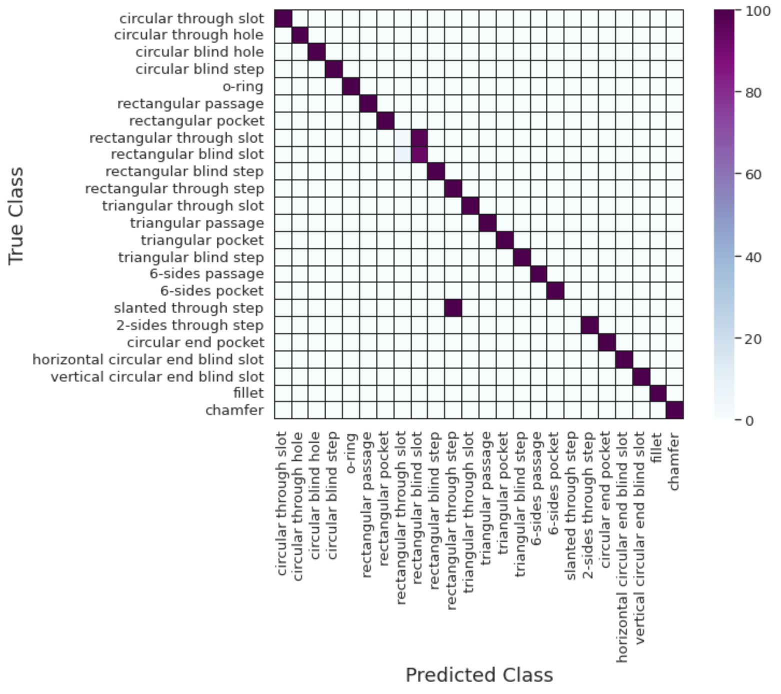
To evaluate the significance of coordinate values and the effects of varying precision, three versions of the single-feature classification model were trained: (1) with all coordinate values replaced with a single placeholder token, (2) with all coordinate values taken to five decimal places and (3) 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.

4.3.1 Experiment 1: How Necessary Are Coordinate Values?

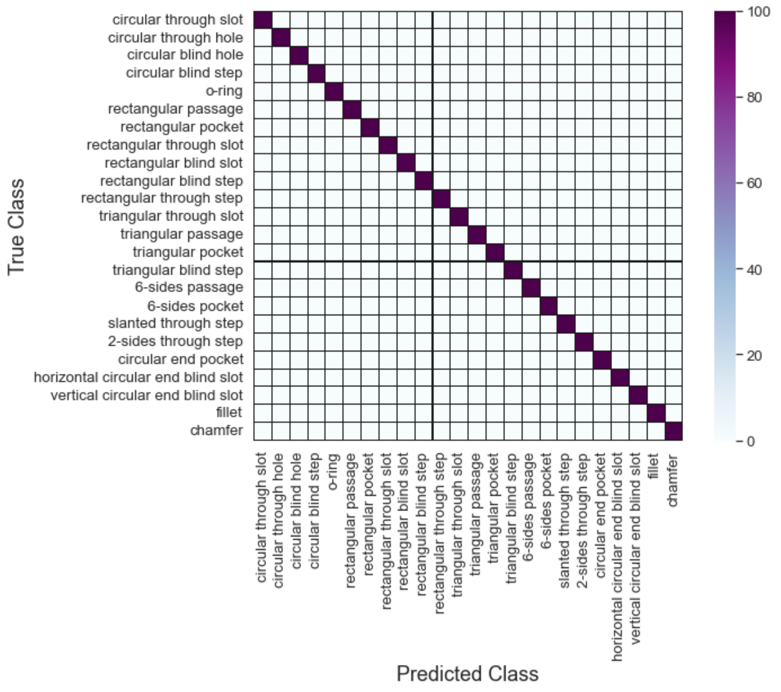
In this experiment, the performance of the model trained without any coordinate points (model 1) is compared with that of the model trained using coordinates taken to five decimal places (model 2). In the case of model 1, the scale and location of features is unknown but the network still has access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

The confusion matrix for model 1 is shown in Figure 4.3a. 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"



(a) Model 1: trained with coordinate points all represented by single "COORD" token



(b) Model 2: trained using actual coordinate point values

Figure 4.3: Confusion matrices showing number of occurrences of predicted class against ground truth when evaluating using the validation dataset

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 Figure 4.3b, 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%.

4.3.2 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).

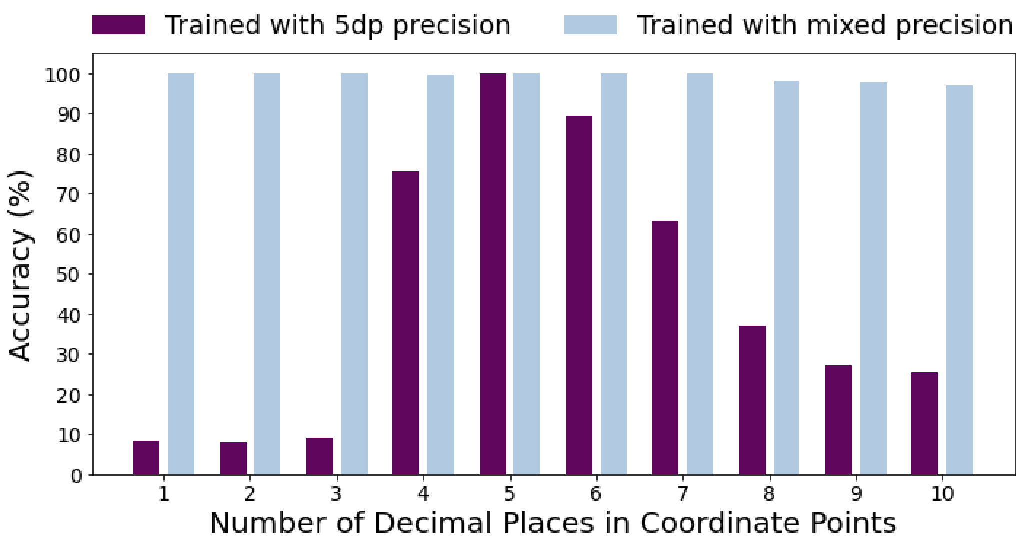


Figure 4.4: Validation accuracy against precision of coordinate values in the dataset for model 2 (trained with 5dp precision) and model 3 (trained with mixed precision)

Figure 4.4 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 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%.

This experiment demonstrates the clear superiority of the mixed-precision approach and so all models detailed in this thesis have been trained with the precision of coordinate values randomised between 1dp and 5dp for each input STEP file.

4.4 Results and Discussion

4.4.1 Details on Existing Solutions used for Comparison

In order to effectively evaluate the performance of our models, the results obtained by several existing feature recognition models are included for comparison. Whilst there are many existing methods for machining feature recognition, fair comparisons are reliant on models trained using the same or comparable datasets. As discussed in Chapter 3, datasets in STEP format are not widely available, and so the work presented here is limited to one dataset which could be reproduced. Therefore, the models chosen for comparison do not represent all existing methods for machining feature recognition, but rather those which have been trained using a dataset equivalent to the one used here.

FeatureNet

The single-feature machining features dataset reproduced for this work was first presented in [1] 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 $64 \times 64 \times 64$. 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 scoresfor each feature category. The total number of learned parameters for this model is around 34 million, several orders of magnitudes larger than our single-feature classification 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 ($64 \times 64 \times 64$ voxels) results in models with higher accuracy but which are slow to train and run whereas the lowest resolution ($16 \times 16 \times 16$ voxels) results in a smaller, faster network but with the cost of a significant loss in accuracy.

MsvNet

In *MsvNet* [2], 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 [60] 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.

SsdNet

The architecture of MsvNet was improved on in [3], which presented SsdNet, in which segmentation and feature recognition are combined into a single process based on SSD [17].

4.4.2 Single-Feature Classification

The performance of our network for single-feature classification will be compared to the optimal performance reported for single-feature classification by FeatureNet and MsvNet.

Our recursive network is trained with batch size set to 32 and learning rate initialised at 10^{-3}. A full training period consists of 10 epochs of training. Results presented are those for the model trained using variable precision of

Table 4.2: Test accuracy and training time for our single-feature classification model with comparison to existing solutions as reported by [2]

	FeatureNet (GPU) [1]		MsvNet (GPU) [2]		Ours (CPU)
Resolution	$16 \times 16 \times 16$	$64 \times 64 \times 64$	$16 \times 16 \times 16$	$64 \times 64 \times 64$	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
No. models	4096	4096	4096	4096	700

coordinate points, as outlined in Section 4.3.

Dataset

Using the benchmark dataset developed by [1] would be ideal as this is the dataset used to train the comparison networks. Therefore, an equivalent single-feature dataset was generated, containing the same number of CAD models in each category and using the same parameters for random model generation.

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, validation and test subsets at a ratio of 7:2:1.

Performance Comparison

In Table 4.2, accuracy and training time for our single-feature classification model 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 [2], in which computations were carried out on a PC with Intel Core i9-9900X CPU and NVIDIA GeForce RTX 2080 TI GPU.

As the two comparison networks work at variable resolutions the highest and lowest voxel resolutions reported are included for comparison. These results are presented to form a simple comparison with existing work. However, the use of different datasets for training and testing results in only an approximate comparison with existing solutions being represented here.

Discussion

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.

4.4.3 Multi-Feature Recognition

Our multi-feature recognition 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 $10^{-3}$and the Adam optimiser used [59]. Training and testing for our network is carried out using a Xeon E5-2609 v3 CPU with two cores and 20GB 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 [2] and SsdNet [3]. 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. This represents a more accurate comparison than that shown in Section 4.4.2, as results are obtained using the same test datasets. However, differences in training conditions, given that the comparison models were not re-trained for this purpose, result in some remaining approximation in the comparison.

Datasets

The neural network is trained using a dataset consisting of 2000 total multi-feature 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 of the multi-feature recognition model is assessed using several test datasets. 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 4.3. Each dataset is comprised of 10 subsets, each containing an equal number of models and generated with parameters as shown in Table

Table 4.3: Parameters used to generate test datasets for the multi-feature recognition model

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 4.4: Parameters used to generate test data subsets for the multi-feature recognition model

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

4.4.

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.

Figure 4.5 shows an example CAD model from each data subset in each of the four test sets.

Performance on Data Equivalent to training data

Table 4.5 shows the F-score achieved by our multi-feature recognition model, compared to existing solutions when performing a machining feature recognition task using data equivalent to that seen during training. These results are visualised in Figure 4.7a. 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

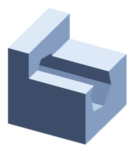
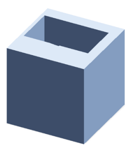
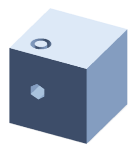
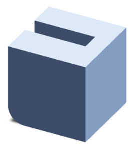
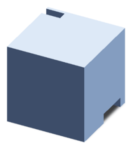
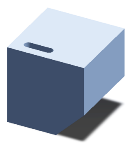
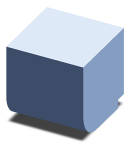
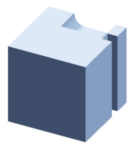
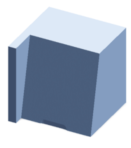
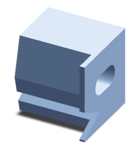
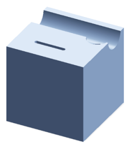
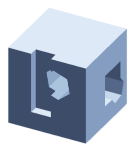
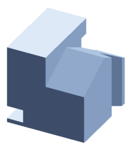
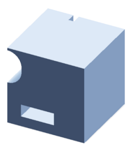
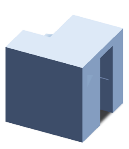
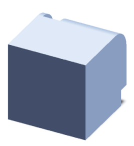
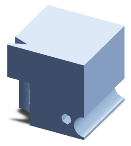
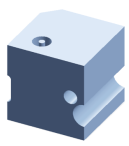
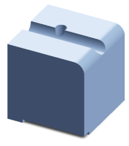
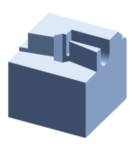
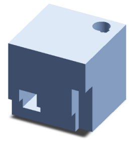
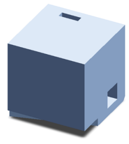
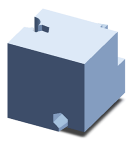
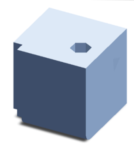
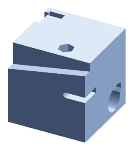
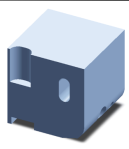
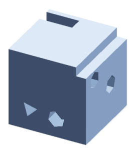
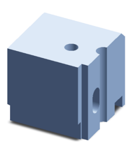
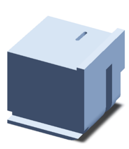
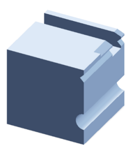
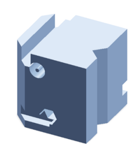
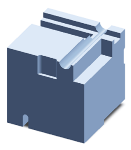
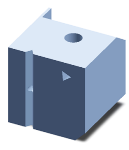
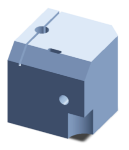
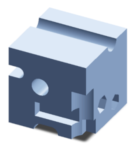
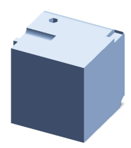
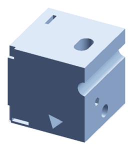
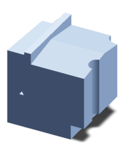
	Test Set 1	Test Set 2	Test Set 3	Test Set 4
Data Subset 1				
Data Subset 2				
Data Subset 3				
Data Subset 4				
Data Subset 5				
Data Subset 6				
Data Subset 7				
Data Subset 8				
Data Subset 9				
Data Subset 10				

Figure 4.5: Example CAD models from each test dataset

Table 4.5: F-scores for multi-feature recognition across Test Set 1, separated by data subset.

Data Subset	1	2	3	4	5	6	7	8	9	10	Total
MsvNet [2]	0.7205	0.8481	0.7699	0.7752	0.7399	0.6866	0.6994	0.7219	0.6657	0.6779	0.7156
SsdNet [3]	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

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.

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 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

Table 4.6: F-scores for multi-feature recognition across each test dataset

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

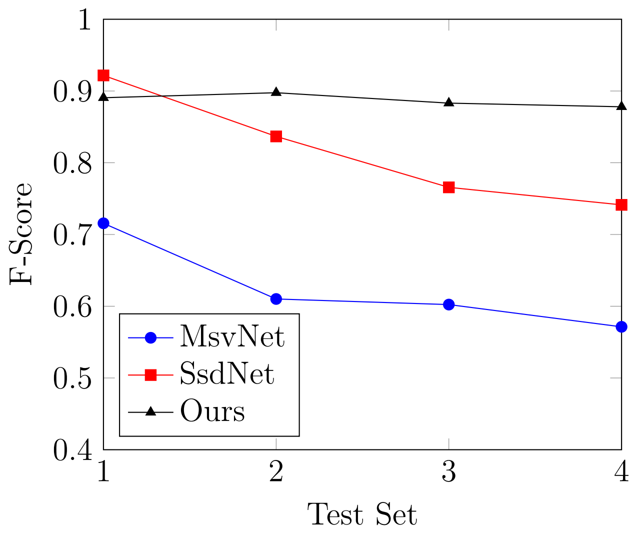


Figure 4.6: Overall F-score of each test dataset plotted against minimum feature size scale factor

run on each dataset to measure comparative performance. Neither our model nor 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 4.6 shows the overall F-score for each test dataset, and these results are plotted in Figure 4.6.

As can be seen from Figure 4.6, both MsvNet and SsdNet show a clear downwards trend in accuracy when increasingly small 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

Table 4.7: Multi-feature recognition performance across all test subsets

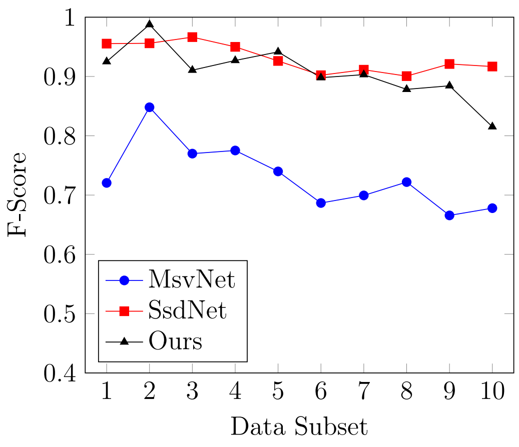
Model	Data Subset	1	2	3	4	5	6	7	8	9	10	Total
MsvNet [2]	Scale 1	0.7205	0.8481	0.7699	0.7752	0.7399	0.6866	0.6994	0.7219	0.6657	0.6779	0.7156
	Scale 2	0.8052	0.6053	0.7664	0.6531	0.7571	0.5951	0.6378	0.5839	0.4984	0.5581	0.6101
	Scale 3	0.6579	0.7632	0.7027	0.7183	0.7363	0.6667	0.5815	0.5292	0.5811	0.5122	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 [3]	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**

the model.

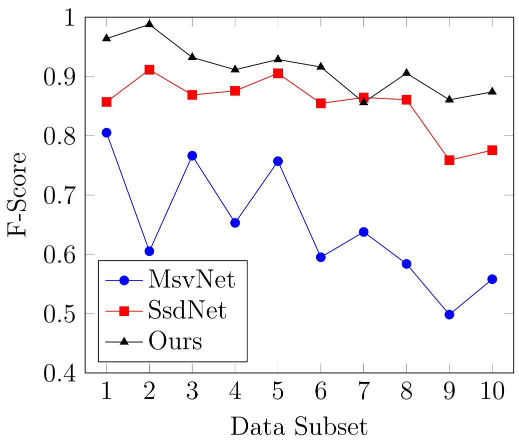
In contrast, our solution shows no strong correlation between minimum feature size and model performance. Small features are represented using the same components in a STEP file as large features, with only the coordinate values differentiating features based on size. Therefore, our model is capable 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 4.7 shows the performance results broken down by data subset and the performance of each model across each test dataset and subset is plotted in Figure 4.7.

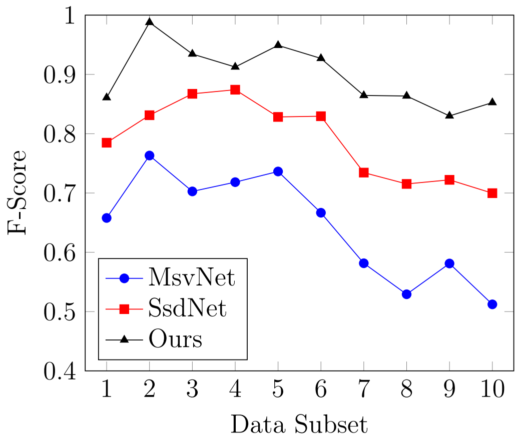
As can be seen in Figure 4.7a, when using Test Set 1, in which feature sizes are equivalent to those used during training, our model and SsdNet display comparable performance. However, as increasingly small features are allowed, increased separation in detection performance can be observed, with performance of both comparison networks dropping significantly, whilst ours remains consistent. Figure 4.8 shows the drop in performance for SsdNet, compared to the comparatively consistent performance of our solution.



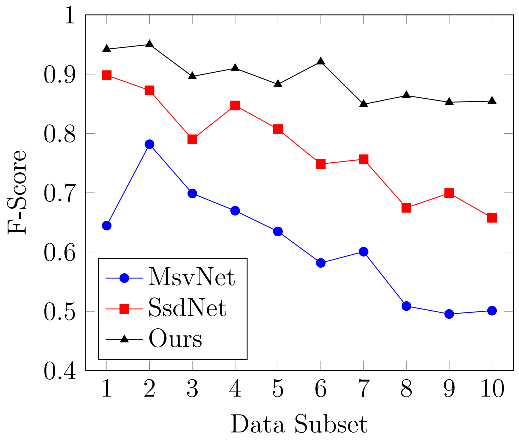
(a) Test set 1



(b) Test set 2

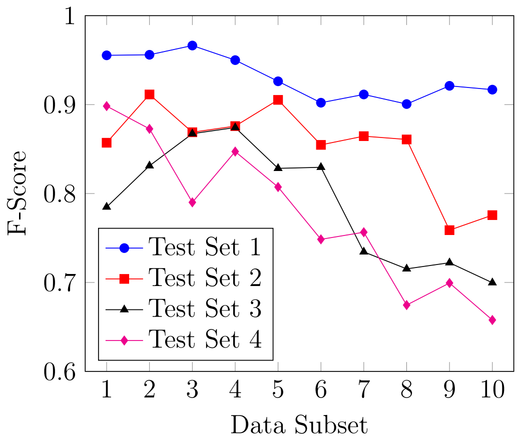


(c) Test set 3

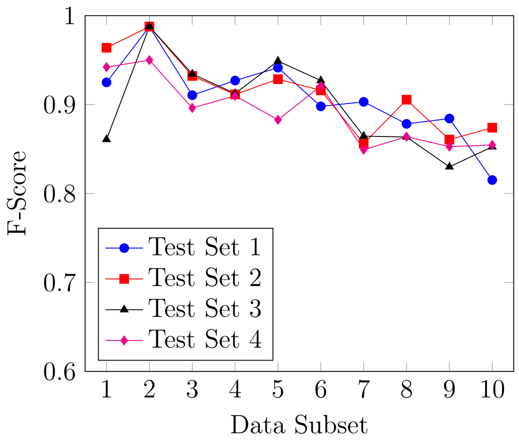


(d) Test set 4

Figure 4.7: Comparison of multi-feature recognition performance with MsvNet [2] and SsdNet [3] across all test subsets



(a) SsdNet



(b) Ours

Figure 4.8: Multi-feature recognition performance for our model and SsdNet [3] across all test subsets

4.5 Chapter Summary

This chapter has presented two models utilising our recursive encoder network for the purposes of machining feature recognition.

It was found that the single-feature 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 24 classes of machining feature.

Results show that our multi-feature recognition model displays high performance independent of minimum 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 approach is capable of performing consistently across a wider range of CAD models, and therefore displays greater flexibility than the existing solutions used for comparison.

Chapter 5

CAD Model Parameter Prediction

5.1 Chapter Overview

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

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

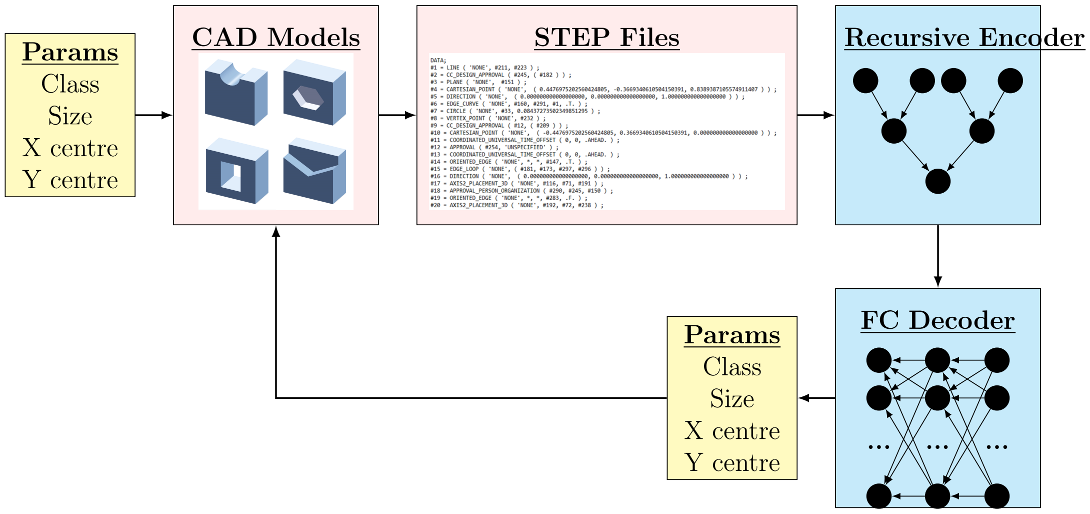
The autoencoder [33] is a class of neural network commonly used to produce compact general representations of data inputs. A traditional autoencoder architecture consists of mirrored encoder and decoder networks. The encoder compresses the input data into a compact form and the decoder attempts to use these encodings to reproduce the input data. These models can be trained in an unsupervised manner, with the goal of training being to encode the inputs in such a way that no information is lost and each data point can be perfectly reconstructed by the decoder. Thus, there is no need for labelled training data. Once trained, the encoder half of autoencoder networks can be applied to diverse tasks, such as classification [34], generation [35] and dimensionality reduction [36].

In this chapter, the recursive encoder network is incorporated into a novel autoencoder-inspired architecture and trained to reproduce the parameters used to create an input CAD model. Section 5.2 outlines the design of the parameter prediction model. Results for a parameter prediction task are presented in Section 5.3.

The work presented in this chapter has been previously published in *Procedia Computer Science*, in a paper titled "Recursive autoencoder network for prediction of cad model parameters from step files” [61].

5.2 Autoencoder-Inspired Model Architecture

Figure 5.1 shows an overview of the model architecture. The recursive network outlined in Chapter 3 encodes the information from the input tree into a single-vector format. This vector is then fed into a fully-connected decoder



```
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',  ( 0.000000000000000000, 0.000000000000000000, 1.000000000000000000 ) ) ;
#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',  ( 0.000000000000000000, 0.000000000000000000, 1.000000000000000000 ) ) ;
#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 ) ;
```

Figure 5.1: Parameter prediction model structure

network which outputs a list of parameters which can be used to generate a new CAD model.

As is the case with traditional autoencoder architectures, our network is trained with the goal of encoding data and subsequently decoding to reproduce the input as accurately as possible. However, due to the hierarchical structure of the data from a STEP file, reproducing the entire input would be an extremely complex task. Not only would the network be required to reproduce the input data points, but also to replicate the entire tree structure. Instead, the autoencoder-inspired network proposed here is designed to encode and reproduce the information necessary to produce a particular part design and not to reproduce an entire STEP tree.

CAD part designs in a large database can typically be organised into discrete categories representing similar objects, such as screws, hinges or pulleys. CAD models in such a database could be classified according to category using a neural network similar to that outlined in Chapter 4, resulting in increased organisation of the dataset. However, this would only represent the first step in developing an effective search engine. Ideally, such a system should be capable not only of identifying all CAD models in a certain category, but also of comparing part designs *within* a particular category.

For the purposes of this work it is assumed that, given a particular category of CAD model, a discrete list of parameters can be produced, representing all the necessary information to produce a specific part design within that category. In the case of real-world mechanical part designs, these parameters could be any detail about the specific design, including size and position variables, number of specific features present, or sub-categories of CAD model type. The model presented here is designed as a generic model, capable of learning the parameters necessary to generate a part design within a given category, with the envisaged operation being of a classification model categorising according to part type, paired with the parameter prediction model to further organise CAD designs within the discrete categories produced. This process could be used for automatic dataset organisation, as well as search-engine functionality through performing a broad search using the classification model, followed by a specific search using the parameter prediction model.

The autoencoder-inspired architecture outlined here is differs somewhat from the typical autoencoder structure in its asymmetric nature, as the decoder does not predict the direct input of the encoder and the two networks have entirely different structures. Nevertheless, the desired outcome is the same as for any autoencoder network: to encode input data as a vector whilst maintaining all information necessary to reconstruct the source, through predicting the necessary parameters to reproduce a specific part design.

Ideally, this model would be trained and tested using a large dataset of realistic CAD part designs in order to evaluate performance in a realistic industrial scenario, as well as identifying strengths and weaknesses of the model when faced with varied part types and parameter lists. Access to such a large-scale labelled dataset was unfortunately not available at time of writing, and so the model outlined in this chapter will be trained and tested using the machining features dataset outlined in Chapter 3, to predict the geometric parameters which define a specific instance of a machining feature.

Fully-Connected Decoder

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

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{5.2.1}$$

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

Ablation Study

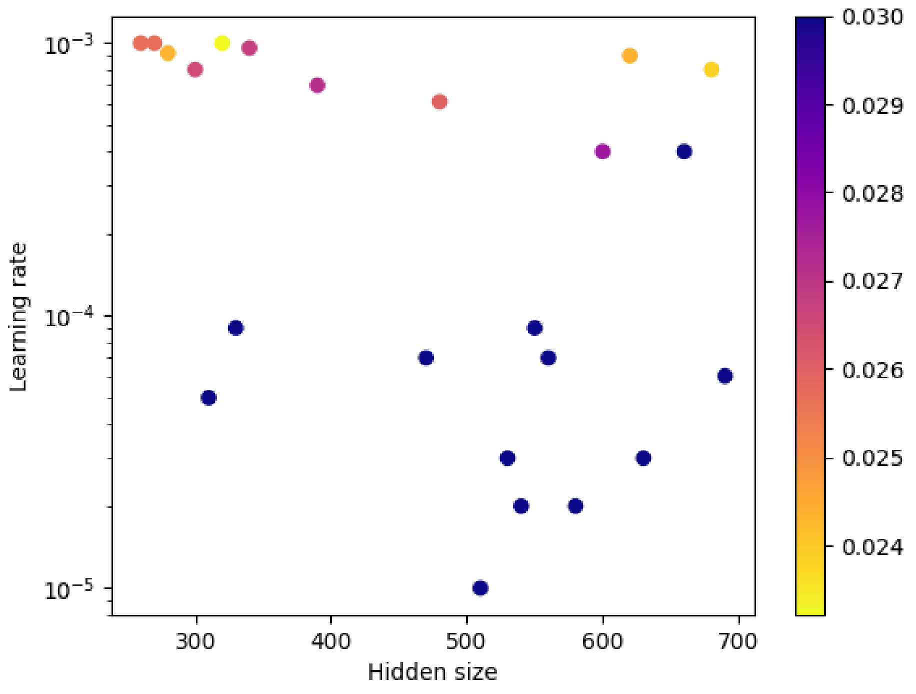
The random search method was utilised to select optimal parameters for the parameter prediction network. Figure 5.2 shows the training-time loss when hidden size and learning rate are varied, the parameters seen to have the greatest effect on model performance.

It can be clearly seen from Figure 5.2 that a higher learning rate, close to 10^{-3}, results in optimal performance. Although less clear cut, Figure 5.2b shows some correlation between a small intermediate hidden size, lower than 100, and high performance. Figure 5.2a implies that model performance can be high for both large and small hidden sizes. It was decided to select a small value in order to minimise the model size, reduce the number of necessary computations, and improve speed performance.

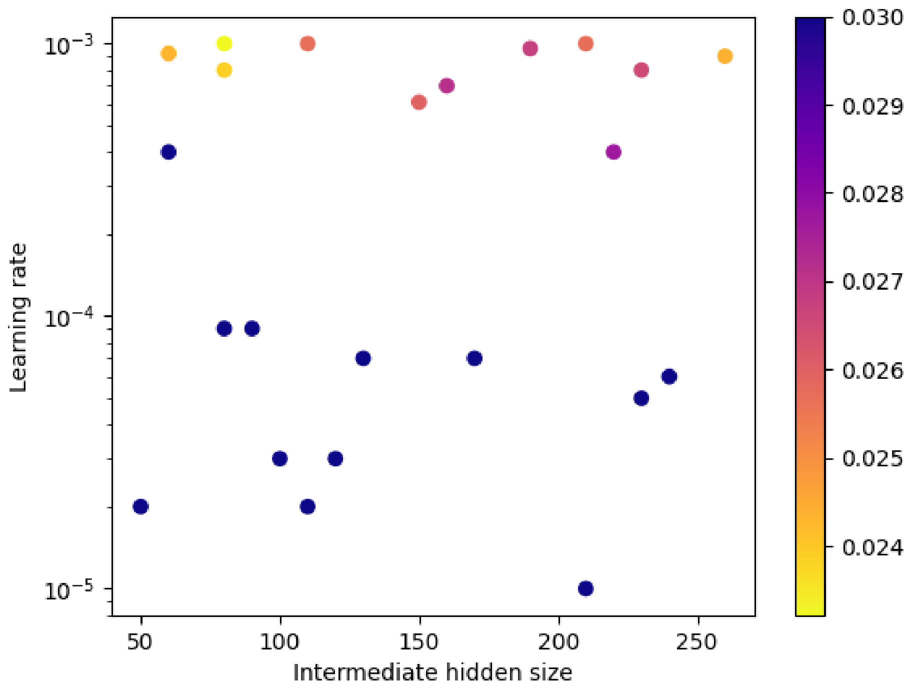
The final values chosen were an initial learning rate of 10^{-3}, hidden size of 280 and intermediate hidden size of 60.

Training

When training for multiple classes of CAD model, it is inevitable that some parameters will be necessary to reproduce certain classes of model but not others. Therefore, a masked Mean Squared Error (MSE) loss was imple-



(a) Encoder hidden size



(b) Decoder intermediate hidden size

Figure 5.2: Train time loss for varied initial learning rate and hidden sizes of model layers

mented as follows, to eliminate irrelevant parameters when calculating loss and performing gradient descent:

$$L = \frac{1}{\sum m_n} \sum_{n=1}^{N} (m_n(x_n - y_n))^2 \tag{5.2.2}$$

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

The entire parameter prediction model is trained from scratch, with encoder and decoder weights updated simultaneously.

5.3 Results and Discussion

The parameter prediction model has been trained to predict the class and geometric parameters required to recreate a CAD model from the dataset. The model was trained with learning rate initialised at $10^{-3}$and the Adam optimiser [59] used for gradient descent. Computations are carried out using a Xeon E5-2609 v3 CPU.

Dataset

The dataset used to train the parameter prediction model is the labelled single-feature dataset with randomised geometric parameters.

The necessary parameters to generate or recreate any model from the dataset are the feature class, size, depth and two coordinate values representing the position of the centre of the feature. As depth is somewhat dependent on feature class, it was decided to ignore it when predicting geometric parameters, leaving four parameters to be predicted (feature class, size, s, and two position values, c_x and c_y). Figure 3.2 shows each of these parameters for an example feature class. Not every parameter is relevant to every model class. For instance, the chamfer class, for which only the feature size can be altered, as its position is fixed to an edge. Table 3.2 shows the parameters necessary to generate instances of each feature class.

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

Model Performance

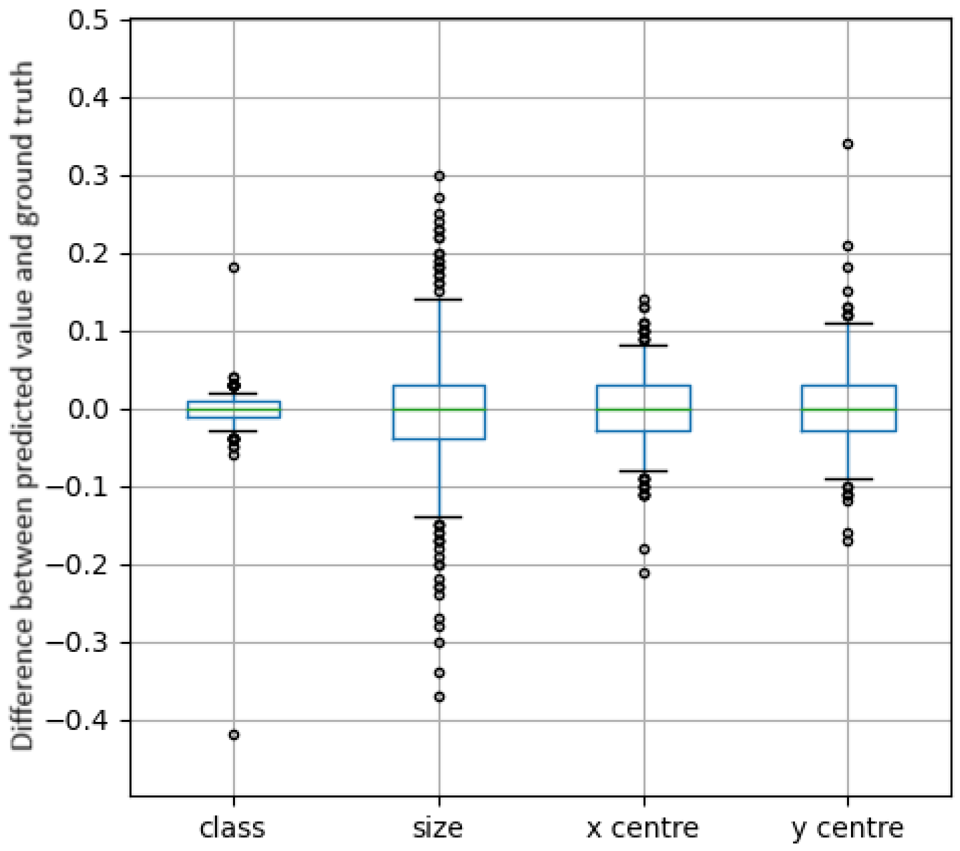
Figure 5.3 shows the error distribution when predicting each of the 4 parameters across the test dataset, with 0 representing a perfect prediction, and y-axis values representing the difference between predicted and ground truth values.Figure 5.3a shows the error in direct neural network output, which is normalised between 0 and 1 for each parameter and Figure 5.3b shows the error in real predicted value, after the output has been denormalised.

Table 5.1: Maximum absolute error in real parameter estimation for various percentiles, across the entire validation dataset.

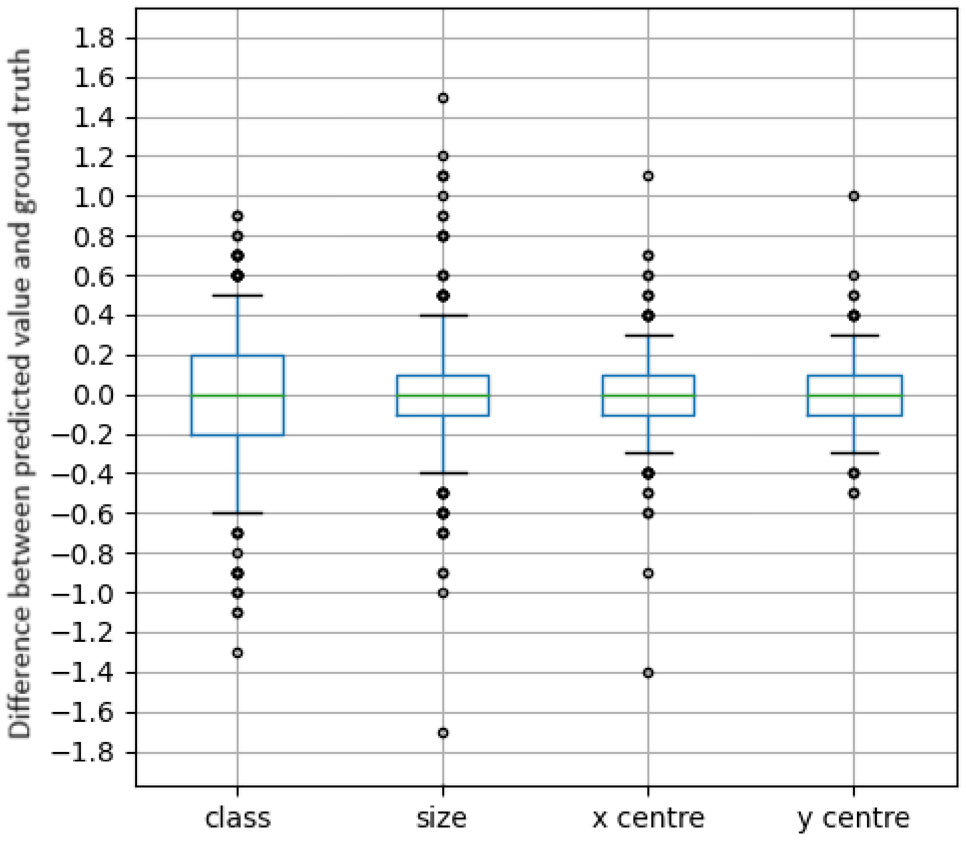
Percentile	Class	Size	X Centre	Y Centre
50%	0.20	0.10	0.10	0.10
60%	0.20	0.10	0.10	0.10
70%	0.30	0.20	0.20	0.20
80%	0.40	0.22	0.20	0.20
90%	0.60	0.40	0.30	0.30

Table 5.1 shows the absolute denormalised error values for the 50th to 90th percentiles, indicating the maximum error with which each parameter will be predicted for a given percentage of models. For instance, the size parameter is predicted with absolute error lower than 0.22 for 80% of CAD models in the test dataset.

From Figure 5.3a, it can be seen that the neural network is best able to accurately predict the value representing the feature class. This is an unsurprising result, given that feature class is the parameter which will produce by far the largest change in the input STEP file when changed, meaning that models of different feature class should be trivial for the neural network to distinguish. Interestingly, however, when converting the output of the decoder into actual predicted values, performance for class, size, x and



(a) Normalised



(b) Real values

Figure 5.3: Box plots showing normalised and real error when predicting each parameter, with 0 representing a perfect prediction; whiskers represent the 95th and 5th percentiles, with points outside these ranges plotted individually.

y coordinates become more similar, with the model capable of predicting a value with absolute error less than 0.2 60% of the time for feature class and 70% of the time for size and coordinate values.

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

In order to further explore model behaviour, further results are presented broken down by feature class in Table 5.2. Here, the percentage of normalised values predicted with a maximum error of 5% is shown for each class of machining feature. The value of 5% was selected for the threshold as a value which produces a good separation between the feature classes, allowing for clear differences in performance to be recognised.

The most accurate predictions of coordinate values are, by a significant margin, those for the circular through slot and circular through hole classes. This may be a result of the representation of circles in STEP files; circles are represented using the coordinate location of the centre, rather than using points along the circle itself as with lines. This representation may have resulted in the extraction of coordinate values for circle features presenting a simpler task for the model than for other feature types. Increasing complexity of the circular feature appears to counteract this advantage, as seen in the circular blind hole and o-ring classes.

The weakest classes generally for parameter prediction are the slanted through step and 2-sides through step classes. This is likely due to these classes dissimilarity with the rest of the dataset with regard to how dimensions apply, resulting in the general representations learned by the model being less applicable to these classes.

Table 5.2: Percentage of normalised predictions made with maximum error of 5%, for each class of machining feature.

	Percentage of Predictions with Error $< 5\%$			
Feature Class	Class	Size	X Centre	Y Centre
Circular through slot	100	80	100	-
Circular through hole	100	50	100	100
Circular blind hole	90	35	25	20
Circular blind step	90	45	-	-
O-ring	75	60	25	25
Rectangular passage	80	10	15	10
Rectangular pocket	75	20	45	20
Rectangular through slot	100	40	15	-
Rectangular blind slot	100	25	15	-
Rectangular blind step	100	50	-	-
Rectangular through step	100	45	-	-
Triangular through slot	95	40	35	-
Triangular passage	70	15	15	25
Triangular pocket	65	0	30	40
Triangular blind step	100	10	-	-
6-sides passage	80	20	30	50
6-sides pocket	70	25	45	25
Slanted through step	75	0	10	-
2-sides through step	75	10	15	-
Circular end pocket	75	45	20	25
Horizontal circular end slot	90	40	25	-
Vertical circular end slot	95	50	35	-
Fillet	90	50	-	-
Chamfer	100	15	-	-

5.4 Chapter Summary

This chapter presents a novel asymmetric, autoencoder-inspired architecture based on our recursive encoder and a fully-connected decoder, for the reproduction of 3D CAD models from STEP files, by predicting the geometric parameters used for model generation. It is demonstrated that geometric parameters, such as size and position, can be estimated from STEP files within reasonable error bounds. These successfully predicted parameters indicate the effective encoding of not only 3D shape but also specific geometric parameters into the single-vector representation produced as output by the recursive encoder network. This chapter represents an initial test in the development of search engine tools for CAD model databases, with the next step being the development of models which can search a database for specific transformations from a given starting CAD model.

Chapter 6

Comparisons Between CAD Models

6.1 Chapter Overview

In previous chapters, this thesis has focused on extracting geometric information from individual CAD models, both the class of feature and additional parameters relating to the specific instance. Tools developed in previous chapters can be used to categorise CAD part designs, and to compare parts according to specified parameters. However, these models are not trained directly to perform comparison tasks, and can therefore only be expected to identify broad similarities between CAD models.

In this chapter, two new models are presented to perform tasks pertaining to comparisons between two CAD models. The goal of these models is to develop more complex search-engine functionality where, given a particular input CAD model, not only can similar CAD models in the dataset be identified, but also CAD models with desired changes.

Another motivation for this approach is the potential development of a traverse-able shared latent space for the encoded output vectors from the recursive encoder network. This would mean that a CAD model could be encoded into a single vector, and a predictable transformation applied to said vector in the latent space, with the result of identifying the existing CAD model within the dataset with the desired transformation. If this complexity can be reliably encoded in the shared latent space, the full encoder-decoder model would only be necessary during training. When operating as a search engine, only the encoder network would be necessary. With the potential of storing encoded vectors for existing CAD models within a database and therefore only running the encoder network itself when a new CAD model is added, this could result in a highly efficient search-engine system.

The first model presented in this chapter, the geometric congruence model, is a simple model trained to determine whether two input CAD models with the same machining feature class are identical or different. The second, more sophisticated model utilises an attentive pooling mechanism to determine whether a given answer CAD model correctly fulfils a given request consisting of a query CAD model and a requested translation. Section 6.2 describes the design of the two models. Results are presented in Section 6.3, with training time results briefly presented for the simple geometric congruence model in Section 6.3.1, before focusing more attention on the attentive pooling model for CAD transformation prediction in Section 6.3.2.

6.2 Model Architectures

In this section, the design of two decoder networks, to be paired with the recursive encoder network presented in Chapter 3, for the purpose of performing the tasks of identifying geometric congruence and specific transformations between two CAD models, will be described.

6.2.1 Geometric Congruence Model

In [62], Coxeter and Greitzer define congruence between two geometries as being present *"if and only if one can be transformed into the other by an isometry"*, where an isometry is any transformation where angles and lengths are preserved, such as rotation.

The geometric congruence model is designed to identify geometric congruence through rotation between pairs of CAD models by treating the comparison as a binary classification task, with one "class" representing congruent and one non-congruent CAD model pairs. As the rotation of a CAD model is trivial information, not relevant to the actual 3D shape, pairs of models which display geometric congruence can be seen to be functionally identical.

The two CAD models to be compared are defined as the query and answer, with the task of the network being to determine if the answer is correct (i.e. the two CAD models display geometric congruence and are therefore identical) or incorrect (i.e. the two CAD models do not display geometric congruence and are therefore non-identical). The query and answer are input to the encoder network, to produce an encoded vector representing each. The query vector is then subtracted from the answer vector, to produce a single output vector of length n (the hidden size of the encoder), representing the difference between the two CAD models. This output vector is then fed into a two-layer fully-connected network which reduces the size of the output vector from length n to length 100 and then from length 100 to a single scalar value. The Rectified Linear Unit (ReLU) activation function is applied between the two fully-connected layers, calculated as:

$$f(x) = max(0, x) \tag{6.2.1}$$

and a sigmoid activation function applied to the output, calculated as:

$$\sigma(x) = \frac{1}{1 + exp(-x)} \tag{6.2.2}$$

As the task being performed is binary classification with a single output value, the threshold is set at 0.5, meaning that a value higher than 0.5 represents geometric congruence and a value lower than 0.5 represents non-identical CAD models.

Training

The network is trained to minimise BCE loss between prediction and labelled value (1 for identical models, 0 for non-identical), calculated as:

$$L = y \cdot \log x + (1 - y) \cdot \log (1 - x) \tag{6.2.3}$$

where x is model output and y is the target value.

During training, a set number of queries are randomly selected for each feature class and for each query, a true and false answer are also randomly selected from the list of possibilities. This keeps the training balanced both in terms of feature class and true/false labels.

The model is initialised using a pre-trained encoder, previously trained for the task of feature recognition, and during training only the decoder weights are updated, with encoder weights fixed to those of the pre-trained encoder. False answers are chosen to always contain the same class of feature as the query, with the intention that the pre-trained encoder weights contain the information necessary to discriminate between classes and so the geometric congruence model need only identify difference between CAD models containing the same class of feature.

6.2.2 Attentive Pooling Model for Transformation Prediction

The geometric congruence model is very simple, and only capable of recognising whether or not two CAD models are identical but cannot predict any more specific information regarding the translations between non-identical CAD models. For this reason, a second, more complex model was designed to predict specific translations between two CAD models. Here, the recursive encoder network is paired with a decoder network which uses attentive pooling [42] to isolate specific differences in STEP files, representing individual transformations.

Here, instead of encoding the entire STEP file input into a single vector representation, the encoder network is stopped at a lower level in the input tree, resulting in an output matrix representing the information present in the level of the tree at which the encoder was stopped. For instance, if the encoder stops after processing all "ADVANCED_FACE" nodes, the encoder output will be a matrix in which each element is an encoded vector representing one face of the CAD model. This matrix can be considered to be an unordered, encoded sentence, where each face of the CAD model is represented by a single word vector. This output "sentence" allows for the application of an attentive pooling mechanism, where the relative significance of each word can be predicted.

As the encoder is no longer outputting a single vector representing the entire data tree, but rather a sequence of vectors representing some level of the tree, there are now many possibilities for how the information is encoded. Any single level of the tree may be used as output, or the information from several levels may be combined, to inform output predictions. When multiple levels are used, attentive pooling is applied separately to each of the output sentences, and the mean taken of the final similarity scores to produce a single representation.

The first level of the tree chosen as output is the level consisting of "ADVANCED_FACE" nodes, representing face-level information. This was initially selected as the sole level of the data tree used to train the network, due to its position as the highest level of shape information below the single "CLOSED_SHELL" node, representing the entire model geometry. In addition, the nodes at this level represent the faces of the 3D part design, meaning that the attention mechanism will return interpretable results; we can identify which faces of the CAD model are focused on when generating a prediction.

Experimentation into the incorporation of additional levels of information demonstrated that utilising a second level of the data tree increased performance significantly, whilst addition of further levels beyond two had limited impact. Therefore, a second level was selected; that of the "EDGE_CURVE" nodes, representing edge-level information. This level was chosen as a result of high performance observed when combining the face and edge level information, as well as to maintain the interpretability of the attention mechanism. With these two levels combined, it is possible to identify both the faces and edges with the most impact on a given prediction.

For the sake of performance and size, it was decided to limit the output to these two levels of information, given the limited improvements in terms of performance when including more. However, it would be entirely feasible to implement this model using information at every level of the tree, with potential applications in the highlighting of all important nodes from a STEP file when performing a given task.

As with the geometric congruence model, during training the attentive pooling model is presented with a randomised query, along with correct and incorrect answer CAD models. A correct answer is a CAD model in which the only change from the query CAD model is the requested transformation.

As this model is trained using a dataset with randomised parameters, a threshold is used to determine whether an answer is correct or incorrect. For a given query the threshold is initialised at 0.1. The model then searches the dataset for an example with the correct feature class and all geometric parameters within 0.1 of those desired as the correct answer; and for an example with the incorrect feature class or a parameter which is more than 0.1 away from the desired value. If the model has performed 10,000 searches and not found suitable correct and incorrect answers using this criteria, the threshold is increased and the process repeated until suitable answers are found.

To represent both the input CAD model and the requested transforma-

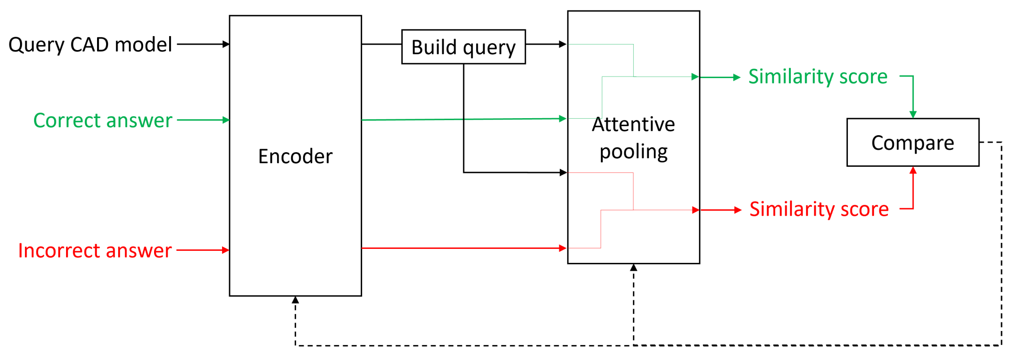


Figure 6.1: Operation of the attentive pooling model; dotted lines indicate weight updates during training

tion, queries are generated by concatenating a vector containing encoded information regarding the specific transformation requested to the output matrix of the encoder network, effectively adding a final word representing the requested transformation to the encoded "sentence" representing the input CAD model.

Figure 6.1 shows the operation of the attentive pooling model during the training period. The model is trained to maximise the similarity score between the query and correct answers and minimise the score between the query and incorrect answers, with the goal of operating as a search-engine by producing a similarity score between the query and every other CAD model in the database, and outputting the CAD models with the highest scores. Each search is capable of identifying a single transformation, so more complex searches requiring multiple transformations would require running the model multiple times.

For the purposes of exploring two types of transformation, two models are trained in parallel:

1. Class transformation model: For a query CAD model containing a feature of class a, identify whether the answer model contains a geometrically similar example of requested feature class b.

2. Geometric transformation model: For a query CAD model containing a feature of class a, identify whether the answer model contains an-

other example of feature class a, where a requested single geometric transformation can be observed.

The types of transformation considered, along with examples of queries and correct and incorrect answers, are shown in Table 6.1. An answer CAD model is considered correct if the requested class or geometric transformation is present, and no other transformations are present. Incorrect answers may contain too many transformations, the wrong transformations, represent the wrong feature class, or be identical to the query CAD model when a transformation is requested.

Attentive Pooling

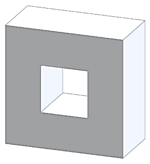
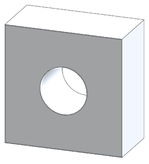
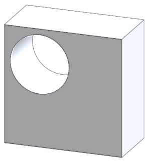
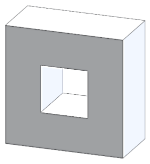
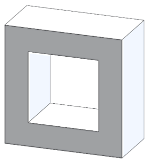
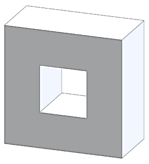
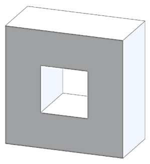
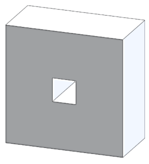
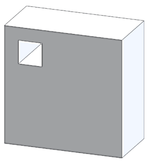
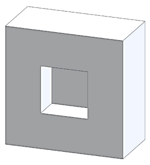
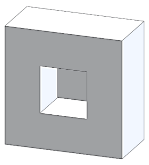
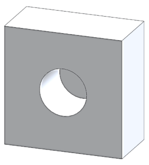
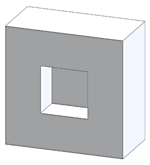
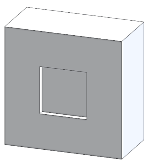
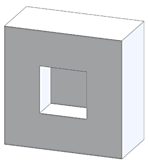
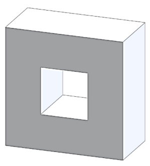
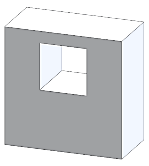
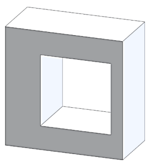
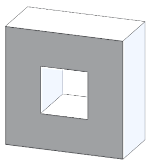
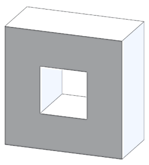
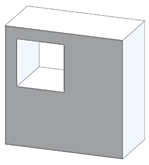
Similarity scores for query and answer pairs are computed using the attentive pooling algorithm presented by dos Santos et al. in [42].

The inputs to the attentive pooling algorithm are two matrices, Q and A, representing the query and answer. Here, the query, Q, is the encoder output for the query CAD model, concatenated with the translation request vector, and will have dimensions (M, n), where n is the hidden size of the encoder and M the number of words in the query matrix, equal to the number of nodes at the output level of the STEP data tree for the query CAD model, plus one. A is the encoder output for the answer CAD model and has dimensions (L, n), where L is the number of words in the answer matrix, equal to the number of nodes at the output level of the STEP data tree for the answer CAD model. A matrix of soft-alignment scores between the inputs is calculated as:

$$G = \tanh\left(QUA^T\right) \tag{6.2.4}$$

where U is a matrix of learned weights, with dimensions (n, n). Max-pooling is then applied column-wise and row-wise to generate word-by-word importance score vectors for each input with respect to the other. Each element

Table 6.1: Examples of query and correct and incorrect answer CAD models for each transformation type.

	Translation Type	Query	Correct Answer	Incorrect Answer
Class Transformation Model	N/A			
Geometric Transformation Model	Size increase			
	Size decrease			
	Depth increase			
	Depth decrease			
	Position change			
	No transformation			

in these vectors is calculated as:

$$[g^q]_j = \max_{1<m<M}[G_{j,m}] \tag{6.2.5}$$

$$[g^a]_j = \max_{1<l<L}[G_{l,j}] \tag{6.2.6}$$

and Softmax is then applied to produce attention vectors, calculated per element as:

$$[\sigma^q]_j = \frac{\exp{[g^q]_j}}{\sum\limits_{1<l<L} \exp{[g^q]_l}} \tag{6.2.7}$$

$$[\sigma^a]_j = \frac{\exp{[g^a]_j}}{\sum\limits_{1<m<M} \exp{[g^q]_m}} \tag{6.2.8}$$

The final representations will be vectors of length n, calculated as:

$$r^q = Q^T \sigma^q \tag{6.2.9}$$

$$r^a = A^T \sigma^a \tag{6.2.10}$$

In order to represent the output using a single comparable value, the cosine similarity between r^q and r^a is then calculated:

$$similarity = \frac{r^q \cdot r^a}{\|r^q\|\|r^a\|} \tag{6.2.11}$$

Training

The neural network is trained to minimise margin ranking loss between the cosine similarity values calculated for the correct and incorrect answers, denoted as x_1 and x_2 respectively. Margin ranking loss will go to zero when x_1 is greater than x_2 by a sufficient margin and increase as the margin decreases or the incorrect value becomes higher. Loss is calculated as:

$$loss = \max(0, \quad margin - (x_1 - x_2)) \tag{6.2.12}$$

where the margin is set as 0.05.

During training, a set number of query CAD models are randomly selected from the entire training set. For each query, a transformation request is also selected through the generation of a randomised, one-hot, transformation vector. In the case of the class transformation model, this is a vector of length 24, with the index of the high value representing the target class. For the geometric transformation model, the transformation vector is of length 5, with each index representing one of the transformations shown in Table 6.1.

Some constraints are placed on the choice of transformation request for a given query. When considering geometric transformations, not all classes of machining feature use every parameter, as shown in Table 3.2, meaning that some transformations are not possible for some classes. For all classes, size transformations are allowed. Depth and position transformations are allowed only for those feature classes where the relevant parameter is used.

The class transformation model is designed to identify the correct "answer" CAD model,where the only change from the query CAD model is a change of class and all other parameters are identical. In order to ensure that requests to this model do, indeed, represent only a single transformations, only relatively similar classes are considered as valid requests. The single transformation produced can therefore be to change the shape but not size and position of a feature, to change between "blind" type (where the feature does not fully penetrate the block) and "through" type (where the feature does fully penetrate) features with the same shape, size and position, or to change the positioning of a feature in one axis, between the body of the block and an edge. The full network of allowed transformations is presented in Figure 6.2.

As can be seen in Figure 6.2, all feature classes in the dataset are connected by one or more allowed transformations. Thus, performing a transformation which is not allowed can in theory be performed through multiple transformations chained together. Similarly, the geometric transformation model can be used to model more complex transformations through multiple

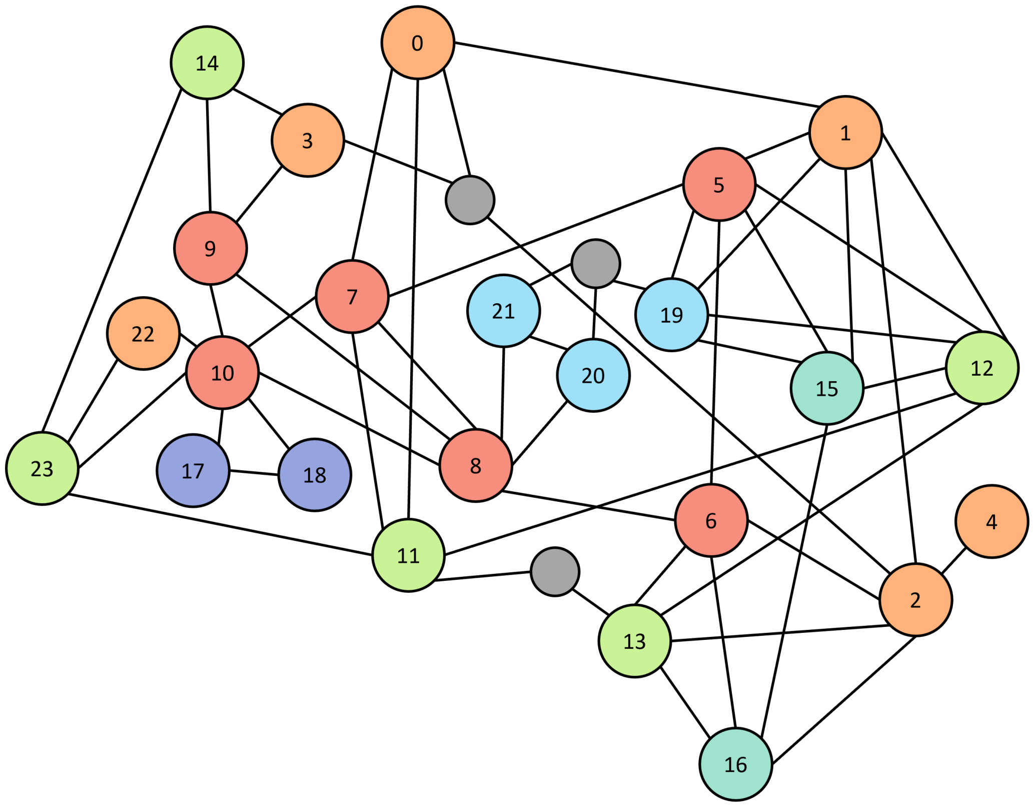


Figure 6.2: Allowed class transformations for the class transformation model, where nodes represent feature classes and connections represent allowed transformations; nodes are colour coordinated according to feature shape; grey, unlabelled nodes represent points where two classes could be connected if a third feature class was present in the dataset, for example no circular blind slot class, which could connect the circular through slot, circular blind step and circular blind hole classes exists.

prediction steps chained together. Therefore, by utilising both networks together, performing a single step at a time, it is possible to fully traverse the dataset and request any possible transformation including both feature class and geometric properties.

In order to optimise model performance, certain constraints are placed on the selection of answer CAD models. The neural networks should be able to compare CAD models from across the dataset and so any CAD model which is not a correct answer should be possible to select as an incorrect answer, to encourage desired behaviour across the full dataset. However, there must also be a focus on the specific behaviour required. The class transformation model must learn not only to predict *a* CAD model of the target class, but a CAD model which also has the correct geometric parameters, and so it is important for it not only to learn to identify features common to each target class, but to discern between CAD models of the same class. Similarly, the geometric transformation model must learn to reliably distinguish between machining features of the same class, and so it is desirable to have many examples of false answers with the correct class to train this behaviour.

Due to the desire to balance these two requirements, for a model which can handle the entire dataset but which also learns desirable behaviour in specific key situations, probabilistic variables are introduced.

For the geometric transformation model, the probability of the false answer CAD model having a different class to the query is set to be:

$$p(diff\_class) = \frac{1}{a} \tag{6.2.13}$$

where a is some integer value. Through experimentation, it was found to be appropriate to set $a = 5$.

A similar approach of setting restrictions on the class of the incorrect answer was investigated for the class transformation model. However, this was found to consistently reduce overall accuracy of predictions at test time and so was not continued.

6.3 Results and Discussion

While both of the models outlined above have the purpose of making comparisons between CAD models, the key difference is in complexity. The geometric congruence model utilises a simplistic method and is only capable of recognising whether two CAD models are identical. It represents, therefore, an initial experiment into the viability of using the STEP encoder to compare CAD models at a more specific level than by feature class alone.

The attentive pooling model is, in contrast, capable of recognising varied transformations of feature class and geometric parameters, as well as identifying identical CAD models. In addition, the implementation of an attention mechanism results in consistently higher performance when recognising identical CAD models.

The more complex behaviour of the attentive pooling model will therefore be the primary focus of this section. Training-time performance of the geometric congruence model will be briefly presented, to establish a baseline of a simple model performing a simple task. Results for the attentive pooling model will then be presented in considerably higher detail.

6.3.1 Geometric Congruence Model

The geometric congruence model has been trained to identify geometric congruence in CAD model pairs using binary classification of congruent or non-congruent pairs. Learning rate is initialised to $10^{-3}$and the Adam optimiser is used. Training is performed on a Xeon E5-2609 v3 CPU.

Dataset

The geometric congruence model was trained using the labelled single-feature dataset with fixed geometric parameters. The lack of randomisation in this dataset leaves it undesirable for the purposes of training. However, in order to provide numerous examples of congruent "answer" CAD models for any given query, it was necessary to use a dataset with fixed parameters.

Table 6.2: Selected parameter values in cm for each class of machining feature, when added to a cube of side length 10cm.

Feature Class	Size	Depth	Coordinate 1	Coordinate 2
0 - Circular through slot	1,2,3	10	2,5,8	0
1 - Circular through hole	1,2,3	10	2,5,8	2,5,8
2 - Circular blind hole	1,2,3	2,5,8	2,5,8	2,5,8
3 - Circular blind step	1,2,3,4,5,6	2,5,8	0	0
4 - O-ring	1,2,3	2,5,8	2,5,8	2,5,8
5 - Rectangular passage	1,2,3	10	2,5,8	2,5,8
6 - Rectangular pocket	1,2,3	2,5,8	2,5,8	2,5,8
7 - Rectangular through slot	1,2,3	10	2,5,8	0
8 - Rectangular blind slot	1,2,3	2,5,8	2,5,8	0
9 - Rectangular blind step	1,2,3,4,5,6	2,5,8	0	0
10 - Rectangular through step	1,2,3,4,5,6	10	0	0
11 - Triangular through slot	1,2,3	10	2,5,8	0
12 - Triangular through hole	1,2,3	10	2,5,8	2,5,8
13 - Triangular blind hole	1,2,3	2,5,8	2,5,8	2,5,8
14 - Triangular blind step	1,2,3,4,5,6	2,5,8	0	0
15 - 6-sides passage	1,2,3	10	2,5,8	2,5,8
16 - 6-sides pocket	1,2,3	2,5,8	2,5,8	2,5,8
17 - Slanted through step	1,2,3,4,5,6	10	0	0
18 - 2-sides through step	1,2,3,4,5,6	10	0	0
19 - Circular-end passage	1,2,3	10	2,5,8	2,5,8
20 - Horizontal circular-end slot	1,2,3	2,5,8	2,5,8	0
21 - Vertical circular-end slot	1,2,3	2,5,8	2,5,8	0
22 - Fillet	1,2,3,4,5,6	10	0	0
23 - Chamfer	1,2,3,4,5,6	10	0	0

The dataset contains a total of 16,698 CAD models, randomly divided into train and test sets at a ratio of 9:1. This dataset is not balanced according to feature class. Instead, for each class of feature a list of selected possible values was set for each parameter, and all possible iterations of the feature using these parameters was generated, with all rotations included. As different feature classes have different parameters available, this results in an unbalanced dataset. Whilst a balanced dataset would be desirable, this would necessitate either reducing the number of CAD models used for classes with more variation, or introducing identical replicas of CAD models for classes with less variation, and so an unbalanced dataset was deemed the most appropriate solution.

The parameter values used for each class of machining feature in the dataset are listed in Table 6.2.

Table 6.3: Validation accuracy by class for the geometric congruence model.

Class	No. of Unique CAD Designs	Accuracy (%)
Circular through slot	6	78
Circular through hole	9	71
Circular blind hole	27	68
Circular blind step	18	77
O-ring	27	72
Rectangular passage	9	74
Rectangular pocket	27	65
Rectangular through slot	6	65
Rectangular blind slot	27	65
Rectangular blind step	18	73
Rectangular through step	6	83
Triangular through slot	6	79
Triangular passage	18	78
Triangular pocket	81	77
Triangular blind step	18	82
6-sides passage	12	76
6-sides pocket	45	65
Slanted through step	18	70
2-sides through step	18	73
Circular end pocket	12	64
Horizontal circular end slot	27	62
Vertical circular end slot	27	65
Fillet	6	79
Chamfer	6	72

Model Performance

Table 6.3 shows the train-time accuracy of the geometric congruence model for each class of machining feature, as well as the total number of unique CAD designs for each class. The overall train-time accuracy for the geometric congruence model was 73%.

This result establishes a clear baseline for comparison of geometric features between CAD models: when applying a simple fully-connected decoder to the output of a fixed encoder, trained for feature classification, the model can correctly distinguish identical from non-identical CAD model pairs in roughly 3 out of 4 cases. It is likely that this result could be improved through fine-tuning of the encoder weights or performing a thorough study into model parameters. However, the primary focus here is on the development of the more complex attentive-pooling model, to achieve more varied behaviour. Therefore, further work into improving the geometric congruence model was not pursued and the result serves primarily as a point of comparison when evaluating the performance of the more advanced model.

6.3.2 Attentive Pooling Model

The attentive pooling model has been trained to distinguish correct from incorrect answer CAD models for randomised queries consisting of encoded representations of query models and encoded transformation request vectors. The encoder network has been pre-trained for the task of geometric parameter prediction. The model was trained with learning rate initialised at $10^{-3}$and the Adam optimiser [59] used for gradient descent. Computations are carried out using a Xeon E5-2609 v3 CPU.

Dataset

The attentive pooling model for prediction of transformations was trained using the labelled single-feature dataset with randomised geometric parameters. Due to the randomisation in this dataset, it is extremely unlikely that a CAD model with the exact geometric parameters necessary to fulfil a transformation request will exist. Therefore, a threshold for similarity is set, representing the maximum change in a parameter where the resulting geometry is still considered to be identical. This threshold value is fixed during training for each model, and varied during testing in order to thoroughly investigate prediction accuracy.

The dataset contains a total of 4800 CAD models, randomly divided into train and test sets at a ratio of 8:2. The dataset is balanced according to feature class, with 400 examples present of each class of feature.

Model Performance

In order to meaningfully evaluate model performance, the transformation models were tested under two sets of conditions:

1. No constraints are set, the query, correct and incorrect answers are randomly selected from the entire dataset.

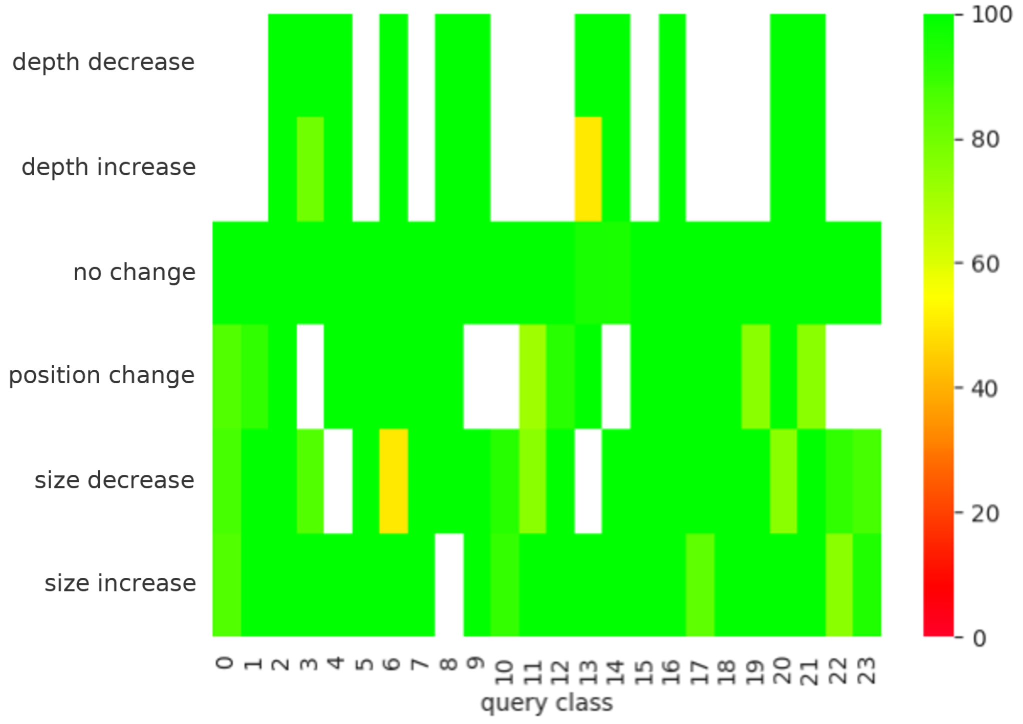
2. The class of incorrect answer CAD models is fixed to be the correct class for the request.

This makes it possible to analyse general performance of the models, whilst also observing areas of weakness as comparisons between instances of correct and incorrect feature classes result in very high accuracy across all transformations, whereas comparisons between correct and incorrect answers of the same class are more difficult.

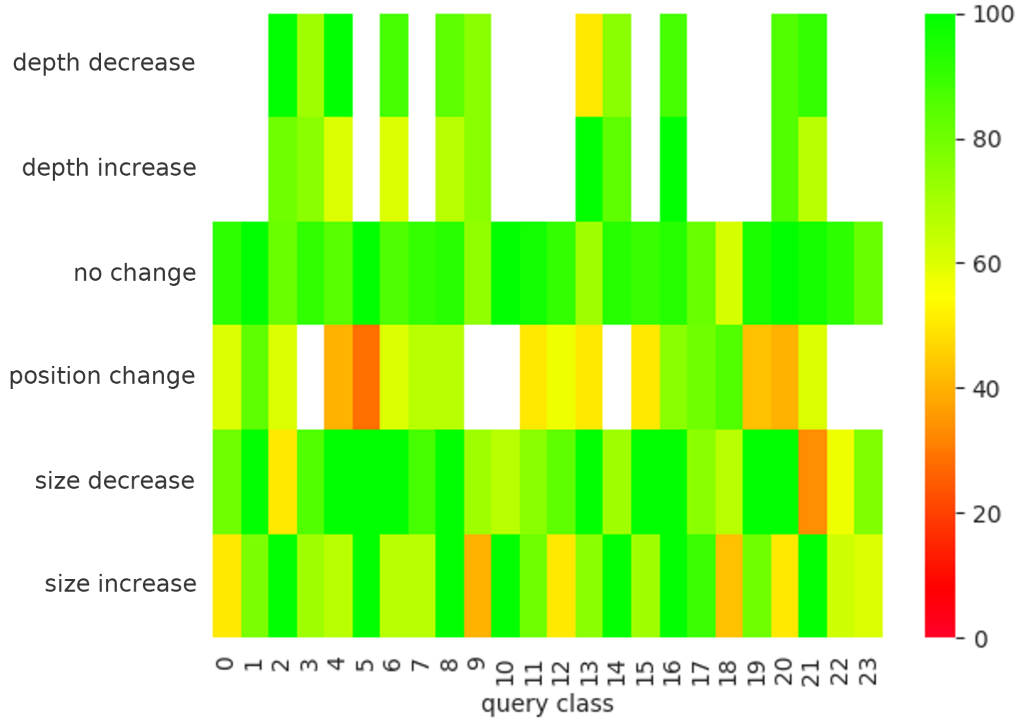
Figures 6.3 and 6.4 show the accuracy at which each allowed transformation request was performed during test time. Results are shown for a total of 1000 randomised queries.

From Figure 6.3b, it is clear to see which transformations are easiest for the models to predict. The geometric transformation model has consistently high performance when identifying identical CAD models. When compared to the result obtained for the geometric congruence model performing the same task, the advantages provided by the more complex architecture are evident.

It appears that model performance is weakest when identifying changes in position, with accuracy below 50% for several query classes. A reason for this weakness could be the definition of position transformations used here. Whilst size and depth are specific, measurable parameters which can be increased or decreased, position transformations are somewhat more complex. Feature classes may have one or two position parameters, and the model has been trained to identify when a single change in position has occurred. It is possible that this is an overly complex definition, which the model struggles to learn for, explaining the comparatively weak performance.

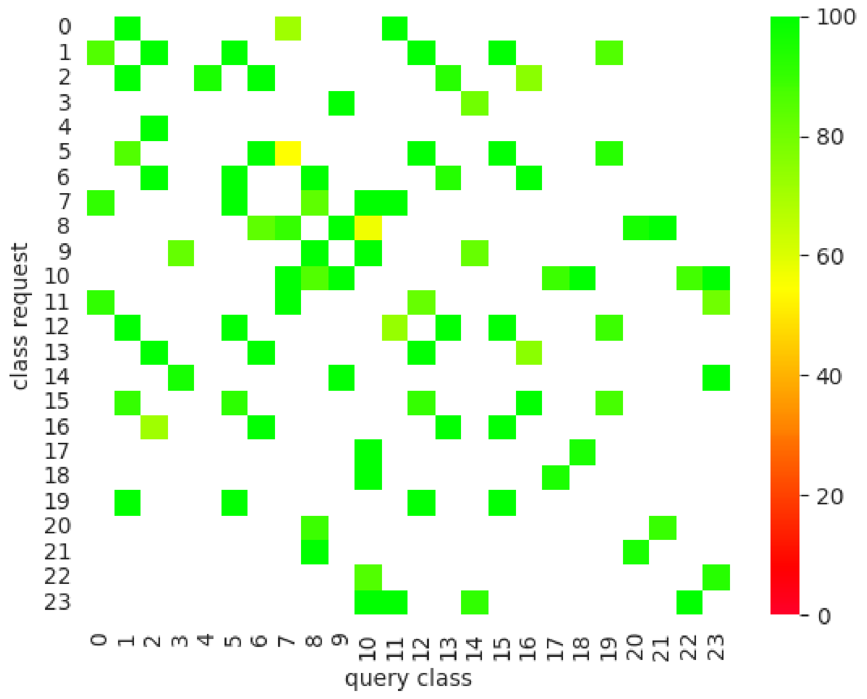


(a) Incorrect answers have any class.

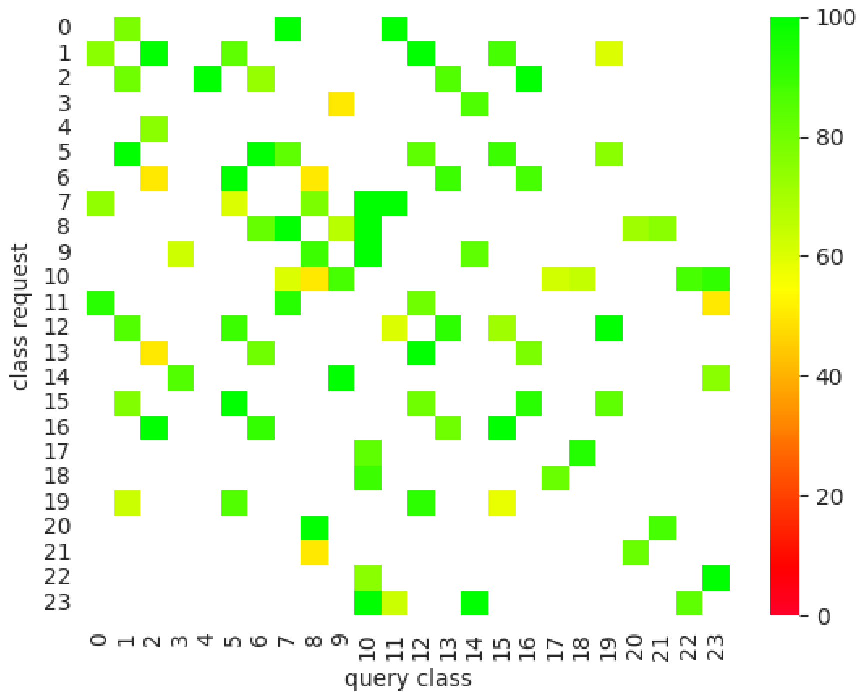


(b) Incorrect answers must have the correct class.

Figure 6.3: Test-time accuracy for all allowed geometric transformation requests.



(a) Incorrect answers have any class.



(b) Incorrect answers must have the correct class.

Figure 6.4: Test-time accuracy for all allowed class transformation requests.

Table 6.4: Test-time accuracy by class for the geometric transformation model.

Class	No. of Queries	Accuracy (%)
Circular through slot	50	100.00
Circular through hole	40	97.50
Circular blind hole	31	93.55
Circular blind step	49	97.96
O-ring	41	95.12
Rectangular passage	36	94.44
Rectangular pocket	35	94.29
Rectangular through slot	44	100.00
Rectangular blind slot	34	94.12
Rectangular blind step	47	97.87
Rectangular through step	47	91.49
Triangular through slot	52	96.15
Triangular passage	44	97.73
Triangular pocket	48	93.75
Triangular blind step	36	100.00
6-sides passage	38	100.00
6-sides pocket	34	100.00
Slanted through step	49	95.92
2-sides through step	47	97.87
Circular end pocket	44	100.00
Horizontal circular end slot	39	97.44
Vertical circular end slot	38	100.00
Fillet	41	97.56
Chamfer	36	100.00

Table 6.5: Test-time accuracy by query class for the class transformation model.

Class	No. of Queries	Accuracy (%)
Circular through slot	40	97.50
Circular through hole	34	97.06
Circular blind hole	54	85.19
Circular blind step	40	90.00
O-ring	37	100.00
Rectangular passage	40	92.50
Rectangular pocket	39	89.74
Rectangular through slot	47	93.62
Rectangular blind slot	35	80.00
Rectangular blind step	41	97.56
Rectangular through step	44	90.91
Triangular through slot	48	93.75
Triangular passage	42	92.86
Triangular pocket	54	98.15
Triangular blind step	34	88.24
6-sides passage	36	97.22
6-sides pocket	50	94.00
Slanted through step	42	97.62
2-sides through step	43	93.02
Circular end pocket	40	95.00
Horizontal circular end slot	40	95.00
Vertical circular end slot	53	96.23
Fillet	27	96.3
Chamfer	40	95.00

Overall accuracy for the geometric transformation model when comparing examples of the same feature class is 80.3%. Accuracy when using the entire dataset is 97.2%. A full breakdown of accuracy for each query class when using the entire dataset is shown in Table 6.4.

Overall accuracy for the class transformation model when comparing examples of the same feature class is 82.3%. Accuracy when using the entire dataset is 93.6%. A full breakdown of accuracy for each query class when using the entire dataset is shown in Table 6.5.

6.4 Chapter Summary

This chapter explores several methods for the comparison of two CAD model files, to identify identical designs and to recognise specific desired transformations. Results show that identical CAD models can be recognised with a high level of accuracy. Transformations in simple geometric parameters, such as feature size and depth, as well as between similar instances of different feature classes, can also be reliably identified, with somewhat more unpredictable behaviour observed when considering transformations in the more complex position value.

As with the methods presented in Chapter 5, the models discussed here represent an initial step in the development of search-engine technology for STEP databases. A key avenue for future research into this approach is the further development of training algorithms with the goal of refining the models.

The task of recognising identical CAD models is straightforward; two models are either identical or they are not, and this is reflected in high test-time performance. However, the task of recognising specific transformations is complex. First, the transformation requested must be defined. In this chapter, we explore simple, general transformations, such as an increase in depth or a position change. These could be replaced with more specific requests, such as a decrease in depth of 1cm. Multiple transformations could be combined into a single request, allowing the model to search for any possible CAD design, starting from any example in the dataset.

In addition, the choice of incorrect answer has an impact on how the model is trained, as discussed relating to feature class in this chapter. In this work, queries and correct and incorrect answers are randomly selected. It would be possible, however, to develop a more systematic approach for training purposes, or to set additional constraints on the selection of answer models. There is also the question of what is considered to be correct or incorrect. Here, a flexible threshold is used, to ensure that answers are found to each query. Alternatives might include using a fixed threshold, using a dataset with fixed parameter values and requiring correct answers to exactly fit specific parameters, or systematically searching the dataset to identify the best possible answer for a given query.

Further work would also be recommended to adapt these models to perform search-engine functionality. Searches can be performed with the models as presented here, through producing similarity scores between a request and every other CAD model in the dataset. However, this is sufficiently different from the training task that the models are not optimised to perform in this fashion. Additional work would be necessary to connect the training task with the desired search-engine functionality.

Chapter 7

Output Vector Analysis

7.1 Chapter Overview

In this chapter, the dimensionality reduction techniques outlined in Chapter 3 are applied to several of the trained models, to evaluate the extent to which information is successfully and predictably encoded into the output vectors from the recursive encoder network.

In this context, both PCA and t-SNE projections are used to visualise the underlying structure of the vector space. However, it should be noted that, whilst t-SNE is purely a visualisation tool, PCA represents linear transformations of the input data, making the resulting projections directly interpretable with relation to particular elements of the input data. Therefore, whilst t-SNE projections are, in almost all cases, more successful at demonstrating patterns in the vector space, PCA projections retain value in showing data trends which can be more readily understood, in theory allowing for the possibility of straightforward traversal of the vector space.

In order to directly and simply compare the output of different models, experiments are carried out here using only one dataset. The dataset used is the labelled single-feature dataset with randomised parameter values, Dataset 3(1) as outlined in Chapter 3.

Due to the use of single-feature data, the model trained to perform feature recognition across the multi-feature dataset is excluded from this analysis. The geometric congruence model is also not considered here, with results instead presented for the more sophisticated attentive pooling models, as well as for the single-feature classification model and the autoencoder-inspired parameter prediction model.

7.2 Single-Feature Classification Model

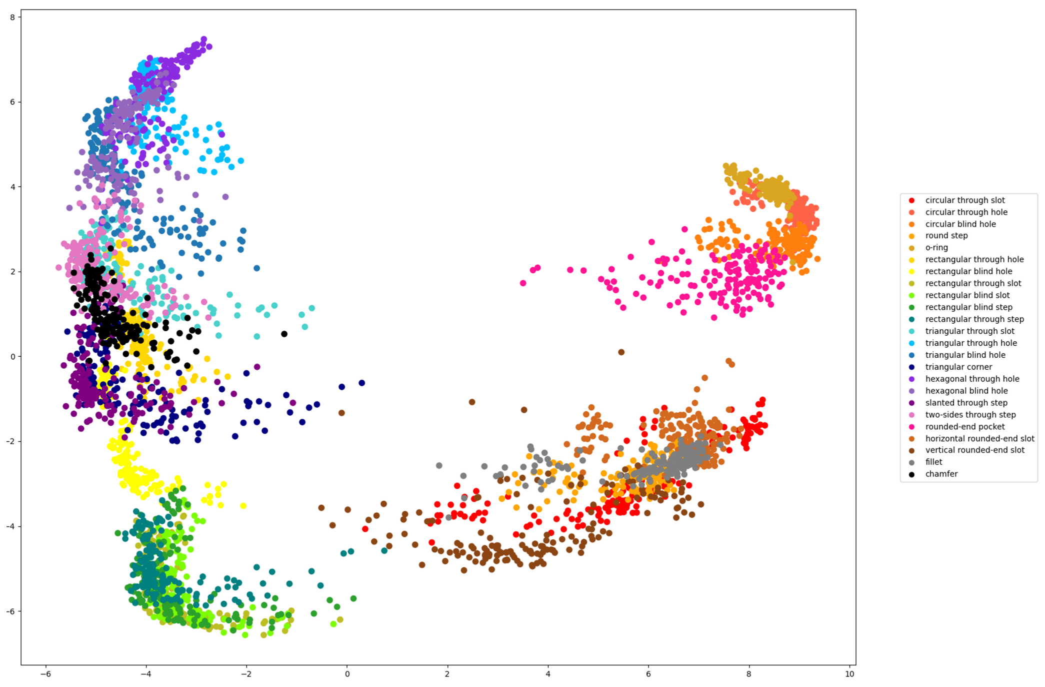
Class Clustering

Figure 7.1 shows the 2D PCA and t-SNE projections of the encoder output vectors produced by the single-feature classification model for the randomised dataset. Data points in the projections are coloured according to class of machining feature.

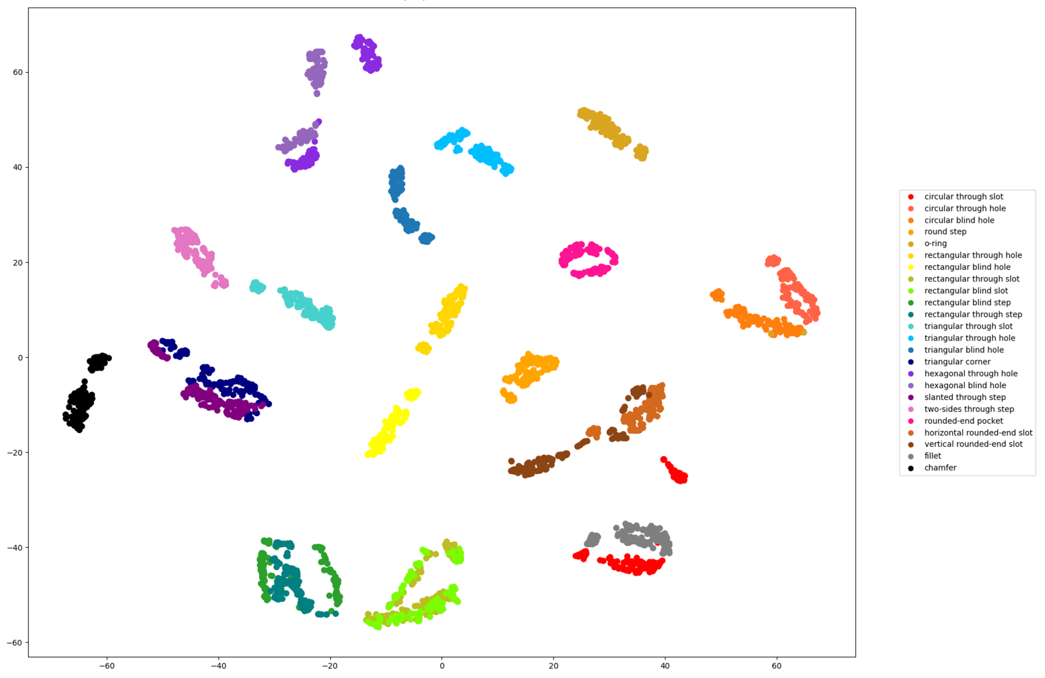
In Figure 7.1a, clustering according to feature class can be observed. However, it is clear that PCA projections are unable to sufficiently separate the classes of machining feature using only two dimensions.

One distinction which can be clearly observed here is between classes of machining feature which contain circular elements and those which do not; all classes visible in the clusters seen to the right of the figure contain circular elements, those visible in the clusters to the left do not. This separation implies that the presence or lack thereof of circular elements is one of the most defining features of the dataset, according to the classification model.

On the circular side of the figure we can also observe two distinct clusters, again with understandable properties. The higher cluster, containing the circular through hole, circular blind hole, o-ring and rounded-end pocket classes represents holes; machining features placed inside a face of the base



(a) PCA projections



(b) t-SNE projections, initialised with PCA

Figure 7.1: PCA and t-SNE projections showing class clustering for the single-feature classification model.

block. The lower cluster, containing the circular through slot, round step, horizontal and vertical rounded end pocket, and fillet classes represents slots and corners; machining features placed on an edge of the base block.

This clustering indicates some positive aspects of the structure of the feature space. Although the model was trained only to distinguish discrete machining feature classes, the interpretability of these clusters implies that the model has successfully learned to encode some information relevant to the relationships *between* certain classes of feature.

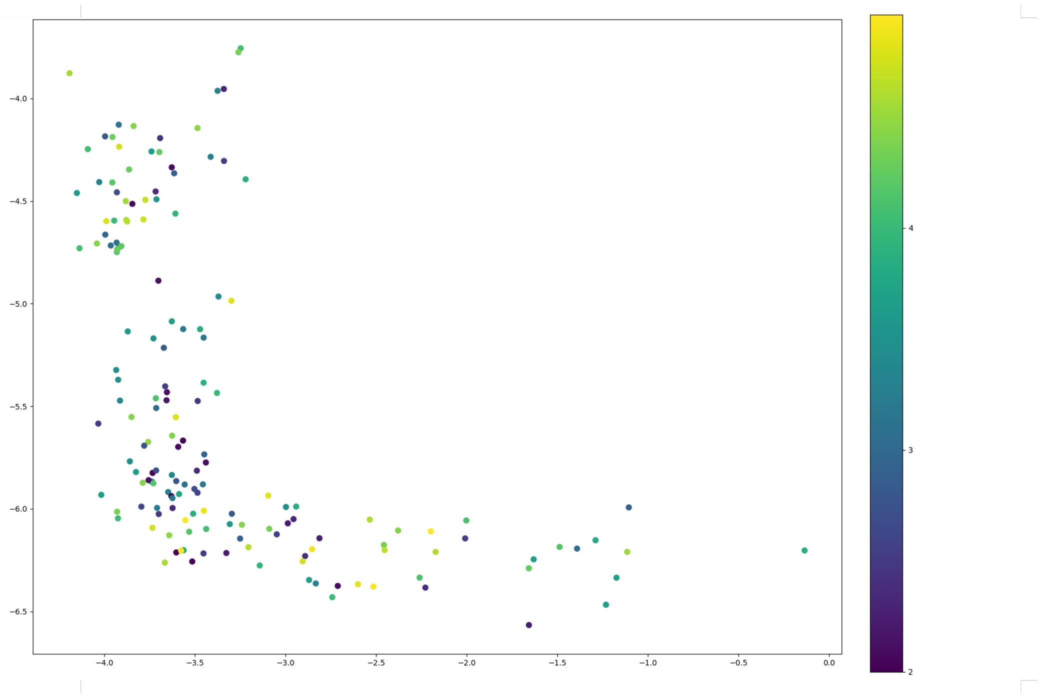
The clear class-based clustering in Figure 7.1b demonstrates the superiority of the t-SNE algorithm for visualisation of dimensionality reduction results in this case. The high accuracy of the classification model implies that machining feature classes can be separated reliably by the model, and using t-SNE projections we can observe this separation in a two-dimensional representation.

As with the PCA projections, the positioning of class clusters within the feature space appears to be meaningful. We can again observe feature classes containing circular elements appearing to the right and those without appearing to the left. Another interesting observation is that all "through hole" and "blind hole" pairs of the same shape appear to mirror each other, demonstrating the clear relationship between these pairs of machining features.

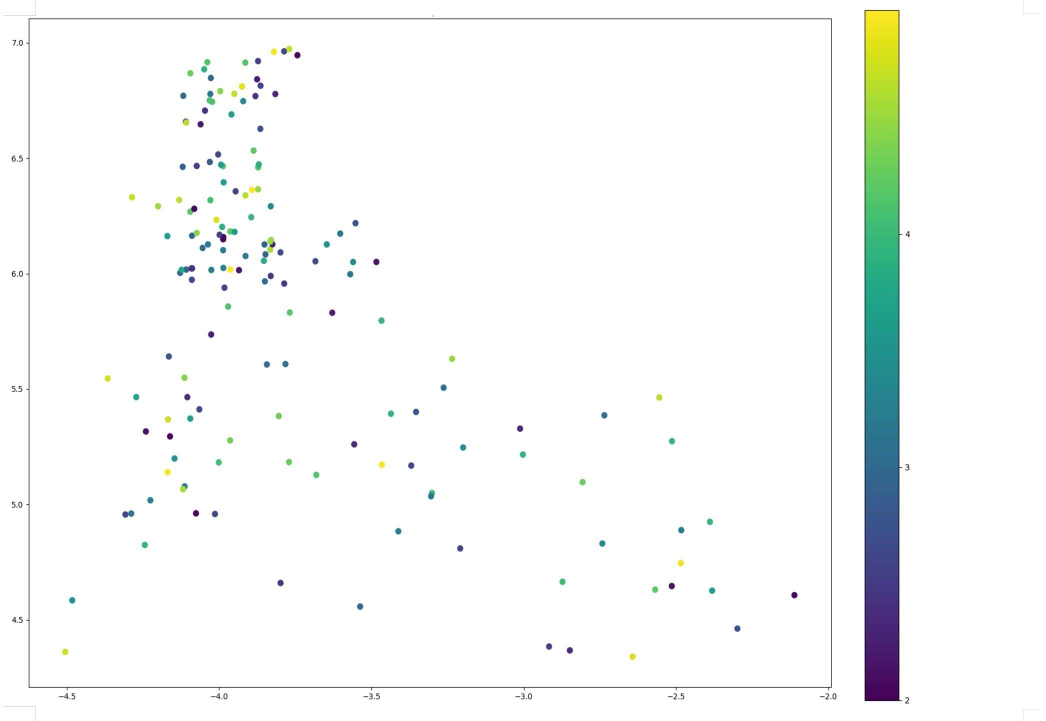
Again, the interpretability of these 2D projections indicates that the model has learned not only to distinguish between distinct classes of machining feature, but also some information regarding the inherent relationships between machining feature classes in the dataset.

Encoding of Geometric Parameters

Figure 7.2 shows some examples of PCA and t-SNE projections of vectors encoded by the classification model for example machining feature classes, coloured according to geometric parameters. Four examples are shown here, but these are generally representative of the patterns observed across all fea-

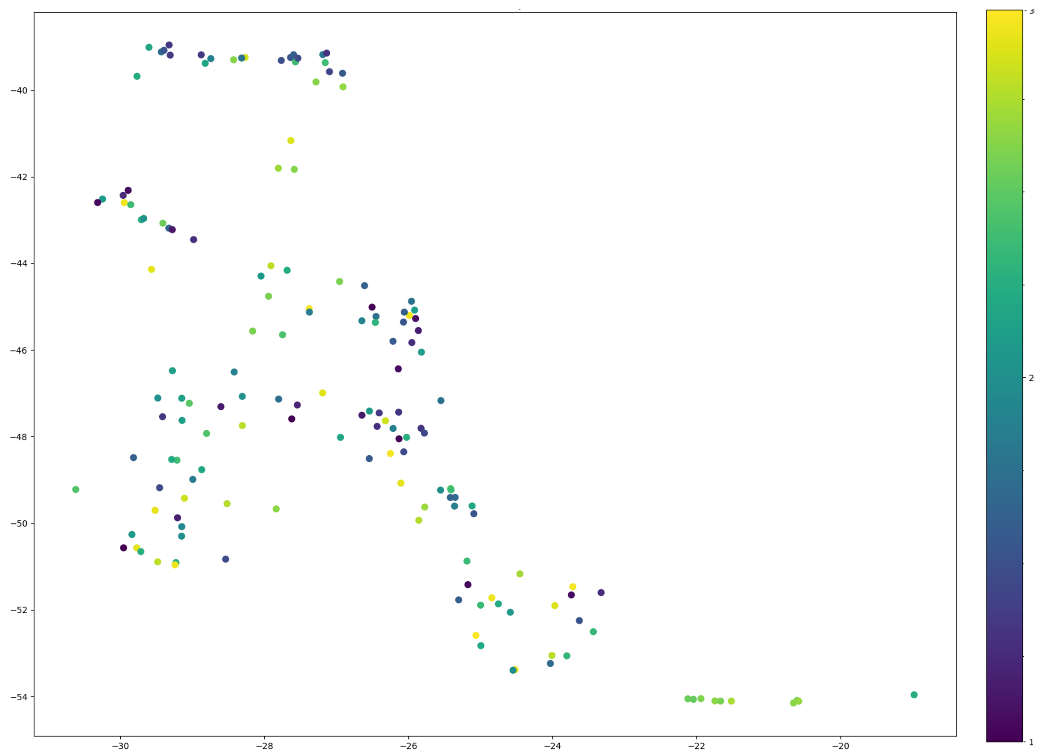


(a) PCA projections for the rectangular through slot class, coloured according to first coordinate value.

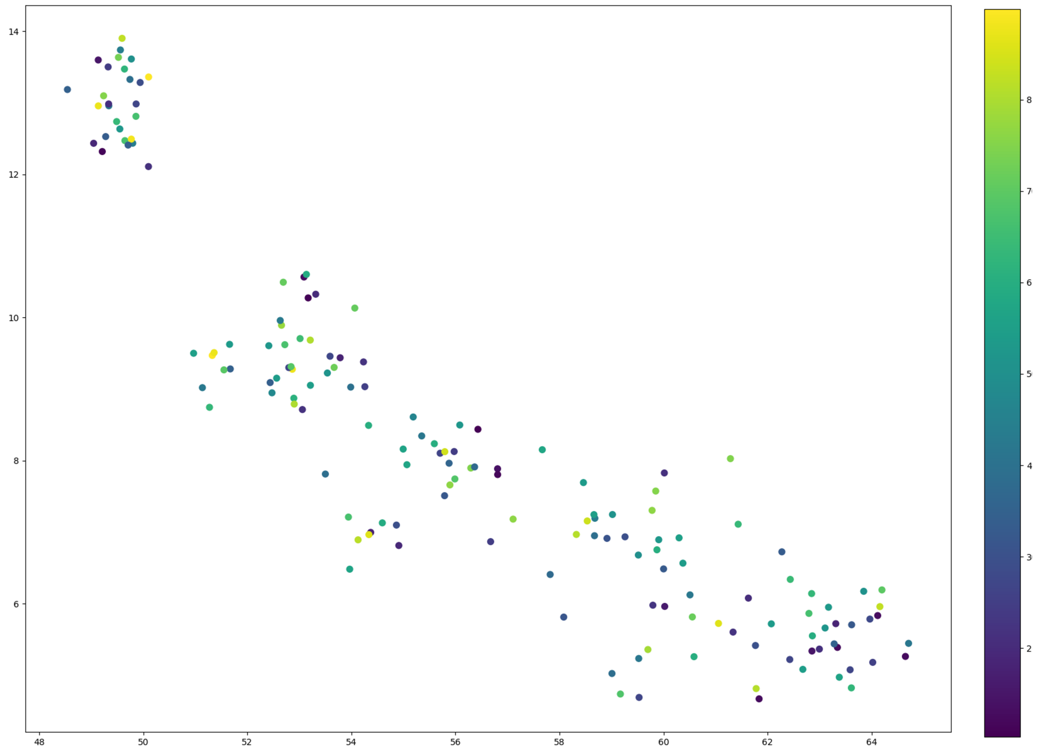


(b) PCA projections for the triangular blind hole class, coloured according to second coordinate value.

Figure 7.2: PCA and t-SNE projections for the classification model, showing distribution of parameters within example class clusters.



(c) t-SNE projections for the rectangular through step class, coloured according to size.



(d) t-SNE projections for the circular blind hole class, coloured according to depth.

Figure 7.2: PCA and t-SNE projections for the classification model, showing distribution of parameters within example class clusters.

ture classes, for all geometric parameters, and using both projection methods. In all cases, representations of particular instances of a given machining feature class appear to be randomly distributed within the class cluster. Across all feature classes, there is no observable relationship between any geometric parameter and location within the vector space.

Given the task that the classification model was trained for, this result is largely unsurprising. The model was trained to identify feature class and not to discriminate between instances of the same class. Although the positioning of class clusters does demonstrate the model's capability of learning some information regarding underlying structure of the dataset which was not directly trained for, this does not appear to extend to representation of any geometric parameters.

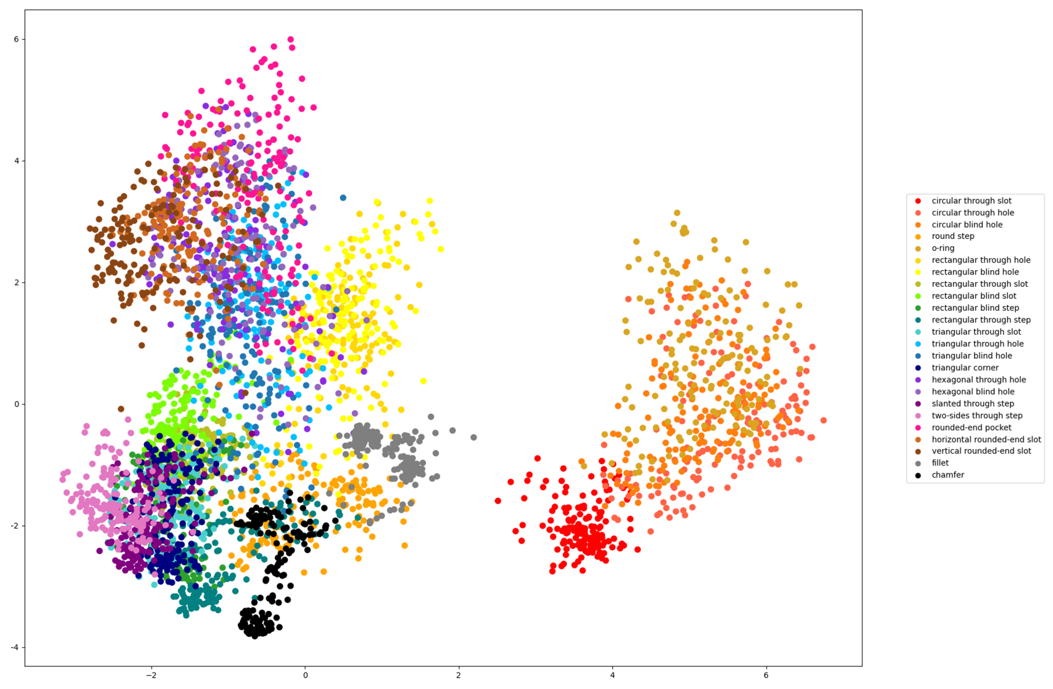
This is likely due to the fact that the model has been trained to identify shape information. The similarities between different feature classes are represented in STEP files by the combination of geometric components present for each class, information which the model is already focusing on to perform the classification task. Geometric parameters, on the other hand, are defined in STEP files using combinations of coordinate point values. As demonstrated in Chapter 4, the classification model does not require coordinate values to discriminate between classes in the majority of cases. Therefore, it is likely that this information is not heavily relied upon for the classification task, and so geometric parameters have very little impact on the final encoding of the output vector.

7.3 Parameter Prediction Model

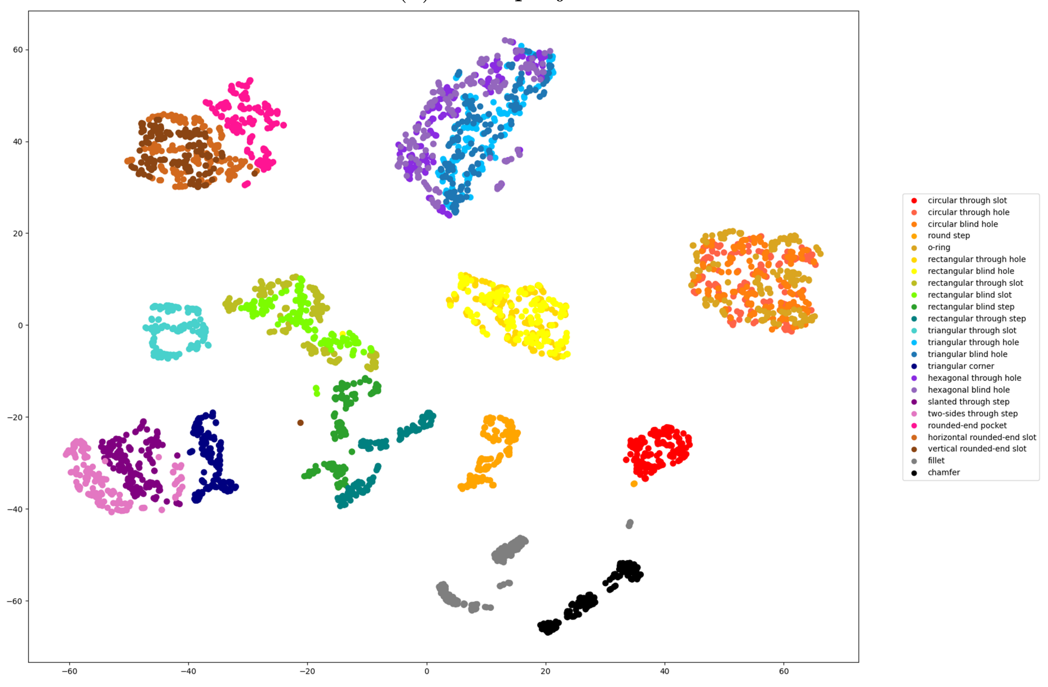
Class Clustering

PCA and t-SNE projections for the randomised single-feature dataset, produced using the parameter prediction model, are shown in Figure 7.3.

In comparison to the classification model, class clustering for the parameter prediction model is clearly less strong, for both projection techniques.



(a) PCA projections



(b) t-SNE projections, initialised with PCA

Figure 7.3: PCA and t-SNE projections showing class clustering for the parameter prediction model.

For the PCA projections, shown in Figure 7.3a, the clear separation between classes containing circular elements and those without is lost. Whilst a cluster of circular feature classes can still be observed to the right of the figure, not all circular classes are included and the meaning behind the clustering does not appear to be as logically interpretable as that for the classification model.

Whilst the class clustering is also weaker for the t-SNE projections, shown in Figure 7.3b, than for those produced using the classification model, the positioning of classes within the feature space has become more clearly interpretable. Corner classes are located towards the bottom left of the figure, holes towards the top right, with edge classes in between. There is also clear clustering according to shape, with rectangular classes appearing near the centre of the figure, circles towards the right, circular end pockets at the top left and triangles and hexagons at the top of the figure.

Encoding of Geometric Parameters

Significant differences between the vector space for the classification and parameter prediction models can be observed in the positioning of output vectors within a single machining feature class. Certain trends can be observed depending on the number of geometric parameters necessary to define a specific instance of a given feature, and so examples will be presented for each case.

Firstly, some machining feature classes are defined by a single geometric parameter, the size. One example of such a class is the chamfer. The distribution of instances of this class based on size, using t-SNE projections, is shown in Figure 7.4.

In this figure, a clear correlation can be observed between feature size and location within the feature space. Interestingly, the correlation appears to be more consistent for smaller features and less consistent for larger, with two discrete clusters forming in the feature space. One explanation for this could be the overall format of the dataset; larger sizes were allowed for cor-

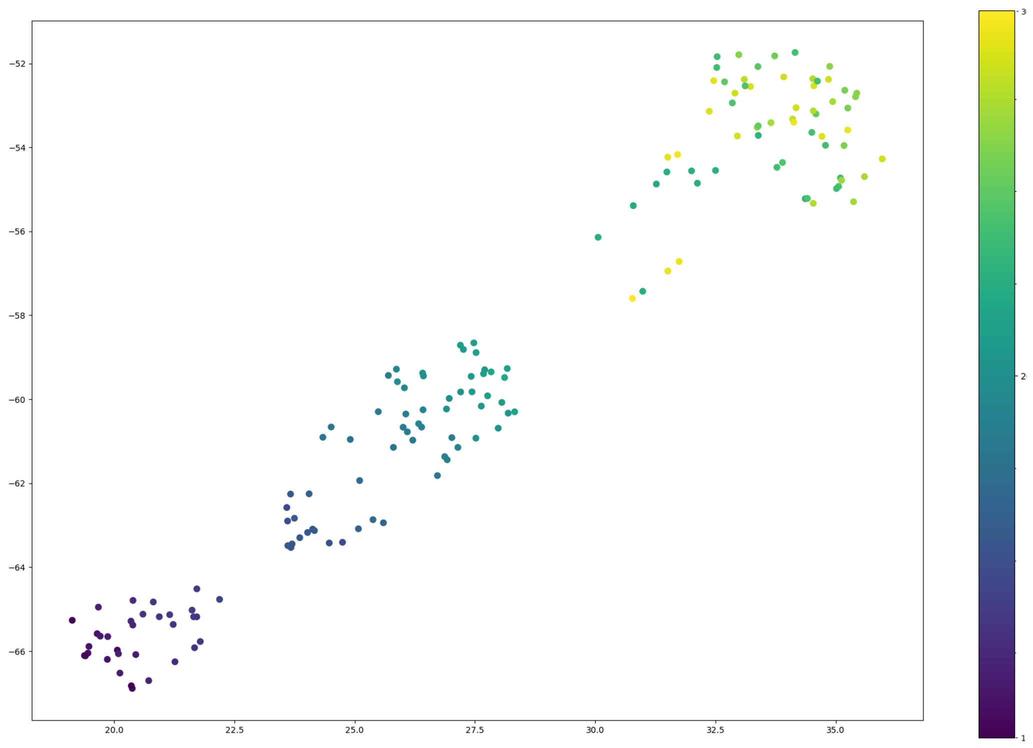


Figure 7.4: t-SNE projections showing distribution according to size across the chamfer class for the parameter prediction model.

ner classes, in order to increase variety in the dataset. It appears that the strongest correlation between size and location in the vector space exists for those CAD models with size parameters within the ranges allowed for all feature classes. This implies a level of generalisation across the dataset, and sharing of information between classes of machining feature, resulting in more meaningful representation of individual CAD models with similarity to more models of different classes.

The strong correlation between feature size and positioning in the vector space in this particular case is, perhaps, a little surprising. As shown in Table 5.2, only 15% of instances of this feature class received size predictions with normalised error less than 5%. This discrepancy implies that complex representations of STEP files with information relating to feature dimensions *are* being learned by the recursive neural network and that accuracy when predicting some geometric parameters may be being limited by the output layers of the neural network converting this representation into a single value.

Positioning in t-SNE projections of output vectors according to geometric parameters for the circular through slot class, an example of a machining feature class which depends on two geometric parameters, is shown in Figure 7.5.

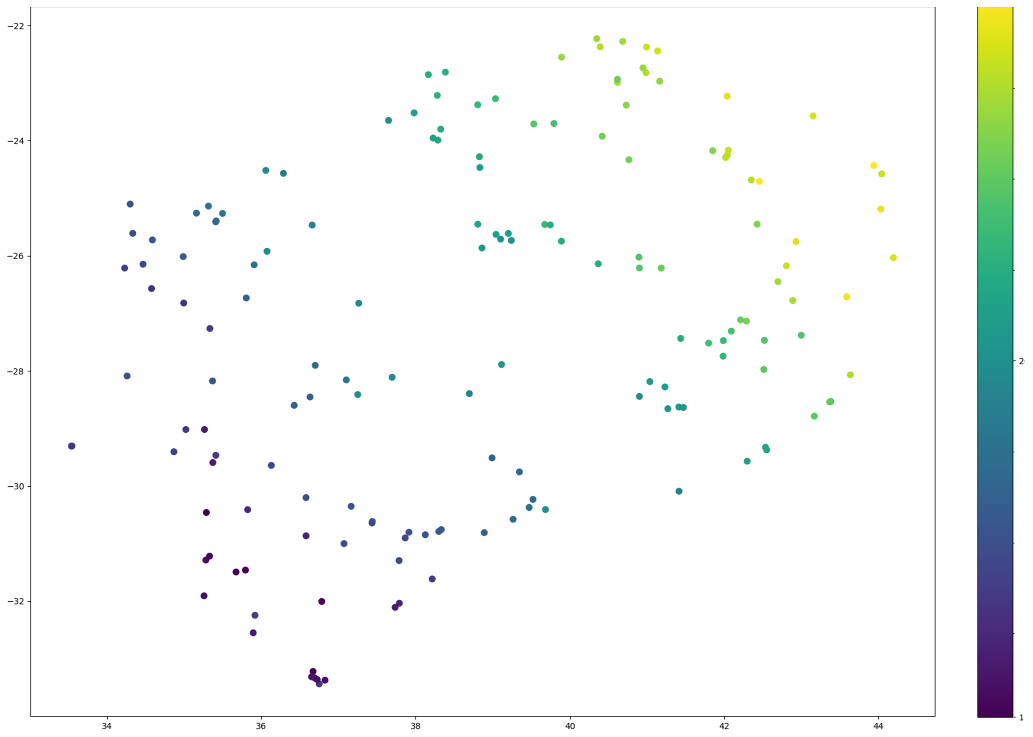
In this case, it can be clearly seen that the vector space is organised according to two recognisable axes; one representing feature size, and one feature position. Similar results can be observed for other feature classes which are defined only by size and a single position value, indicating that these two parameters can be very successfully represented in the encoded vectors.

In the specific case of the circular through slot class, we might expect this high level of separation according to the geometric parameters. As shown in Table 5.2, the size and coordinate value for this class were predicted with error less than 5% at a rate of 80% and 100% respectively. However, this clear organisation of the vector space is consistent for all slot classes, many of which saw significantly lower accuracy of predictions. Again, the implication is that the model is capable of learning complex representations, with the challenge being accessing the information stored in a useful way.

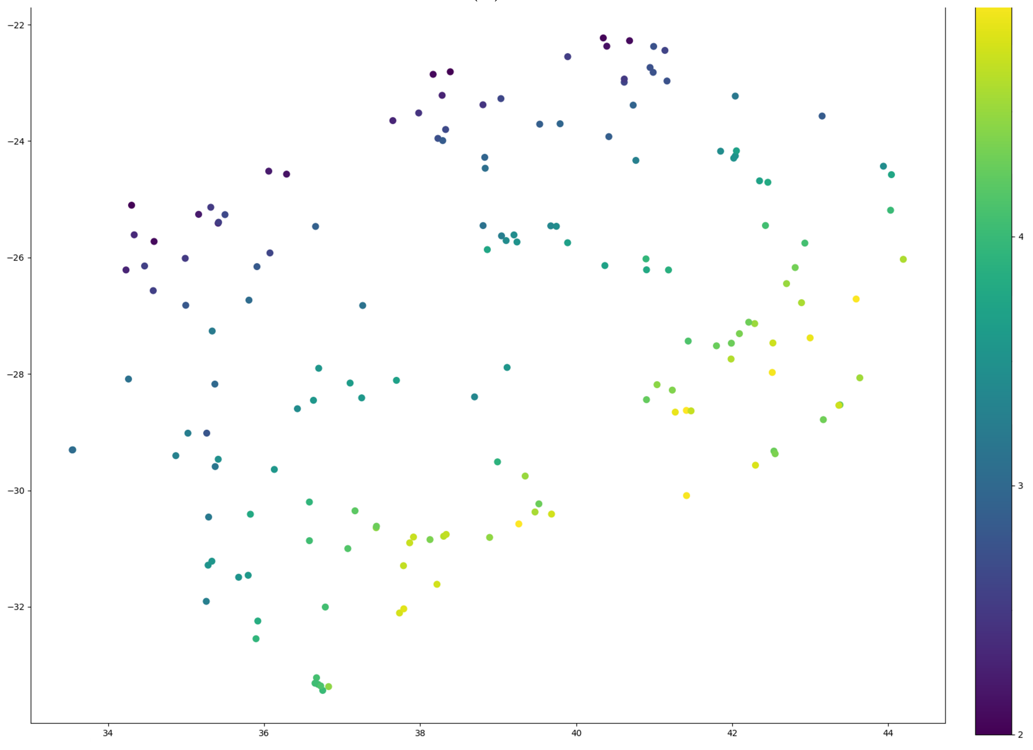
When a feature is defined using three geometric parameters, size and two coordinate values, two similar axes can be observed. Figure 7.6 shows the positioning according to geometric parameters within the rectangular through hole class. Again, an axis can be clearly observed which correlates to size, with a second axis representing position and correlating to both coordinate values.

These projections are a little messier than those seen in Figure 7.5, especially the second coordinate value, as seen in Figure 7.6c. However, it does appear that all three parameters are represented in the encoded vectors, with positioning in the projected feature space a fairly reliable indicator of the values of these parameters for a particular instance of the machining feature class.

Here, the discrepancy between clearly observed patterns in the vector

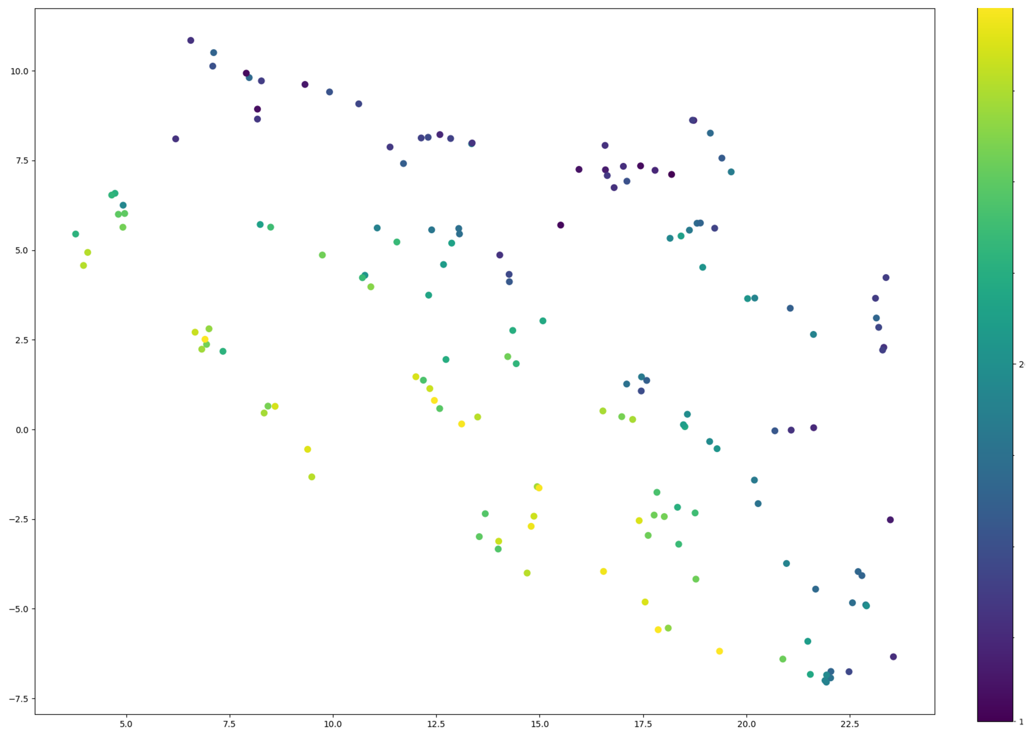


(a) Size.

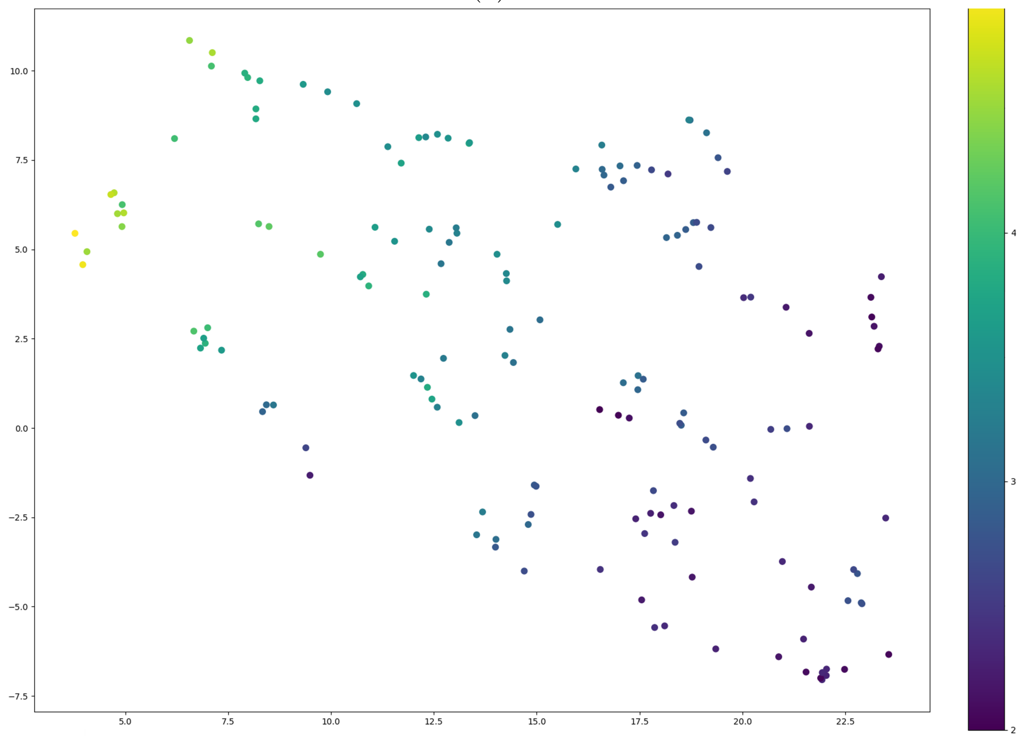


(b) Position.

Figure 7.5: t-SNE projections showing distribution according to geometric parameters across the circular through slot class for the parameter prediction model.

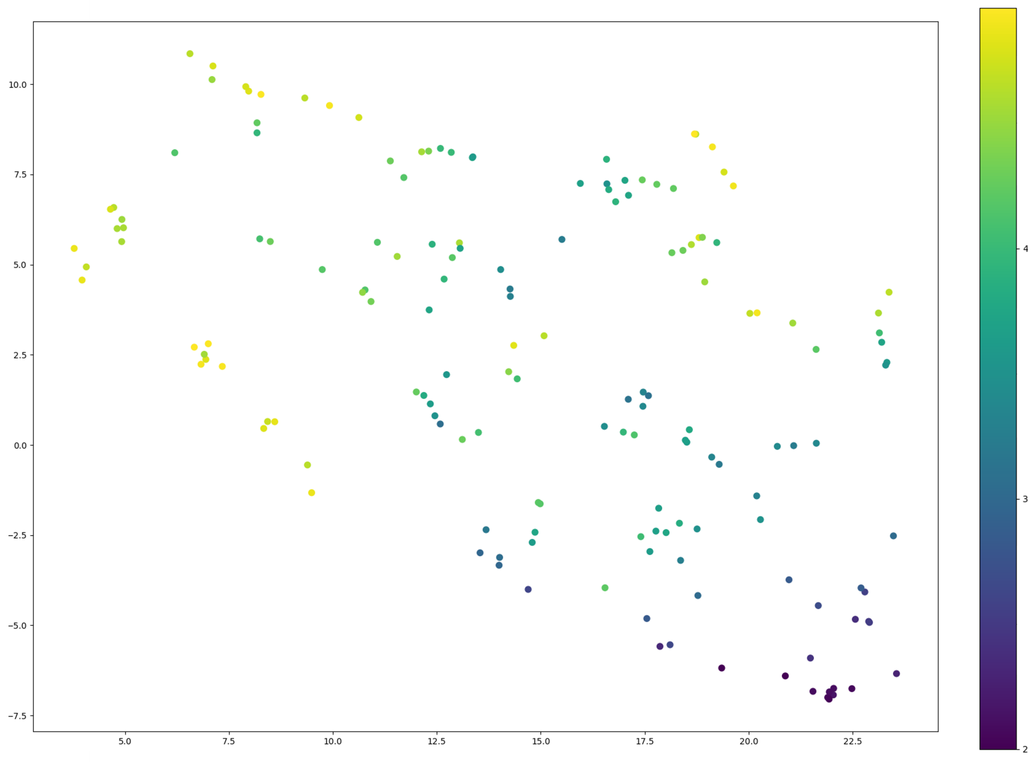


(a) Size.



(b) First coordinate value.

Figure 7.6: t-SNE projections showing distribution according to geometric parameters across the rectangular through hole class for the parameter prediction model.



(c) Second coordinate value.

Figure 7.6: t-SNE projections showing distribution according to geometric parameters across the rectangular through hole class for the parameter prediction model.

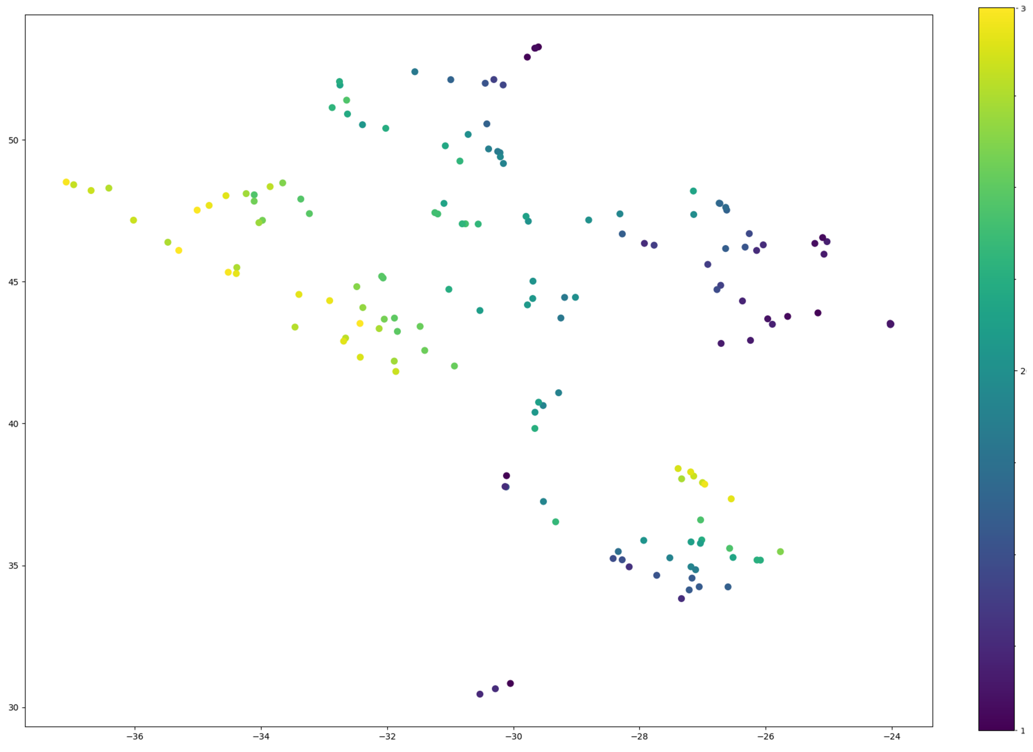
space and accuracy limitations when predicting values, has a simply identifiable cause, at least in the case of the coordinate values. Whilst clear correlations can be seen, these are not as strong as those for classes only represented by one or two geometric parameters, especially for the second coordinate value. In addition, the two coordinate values appear to share one axis in the 2D vector space, which may result in confusion between the values and difficulty in reliably separating the two when predicting values.

Not all machining feature classes show such strong correlations, however. A general trend seems to show that classes of machining feature with more complex shapes show less organised feature spaces when projected in two dimensions. An example is shown in Figure 7.7, which shows the t-SNE projections for the circular end pocket class.

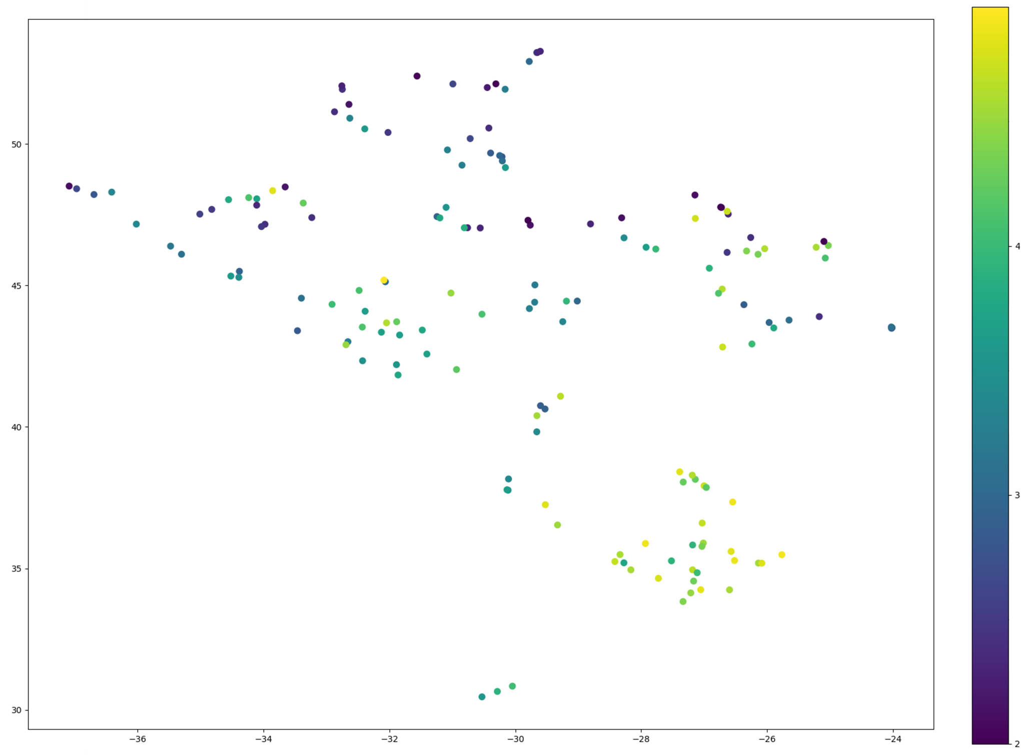
Whilst these projections do show a significant improvement from the random positioning observed in Figure 7.2, the relationships between geometric parameters and position in the feature space appear to be more complex and less easily interpretable than those seen for other classes of machining feature, with the clear axes shown in Figures 7.5 and 7.6 absent.

This could be indicative of the model being less suitable for the interpretation of more complex geometries. However, the dominance of extremely simple machining features within the dataset may also be responsible; a different, more complex dataset would be necessary to further test this behaviour and better understand the potential strengths and limitations of the model.

Once again, these results do not appear to particularly correlate with those presented in Table 5.2, where it can be seen that the accuracy of geometric parameter predictions for the circular end pocket class are fairly average when compared with those obtained for other feature classes. It is possible that, whilst geometric parameters are less effectively represented in the vector space here, some factor results in them being more straightforward to extract for the output layers. Alternatively, it is possible that, for more complex feature classes, two dimensions are simply insufficient for meaningful

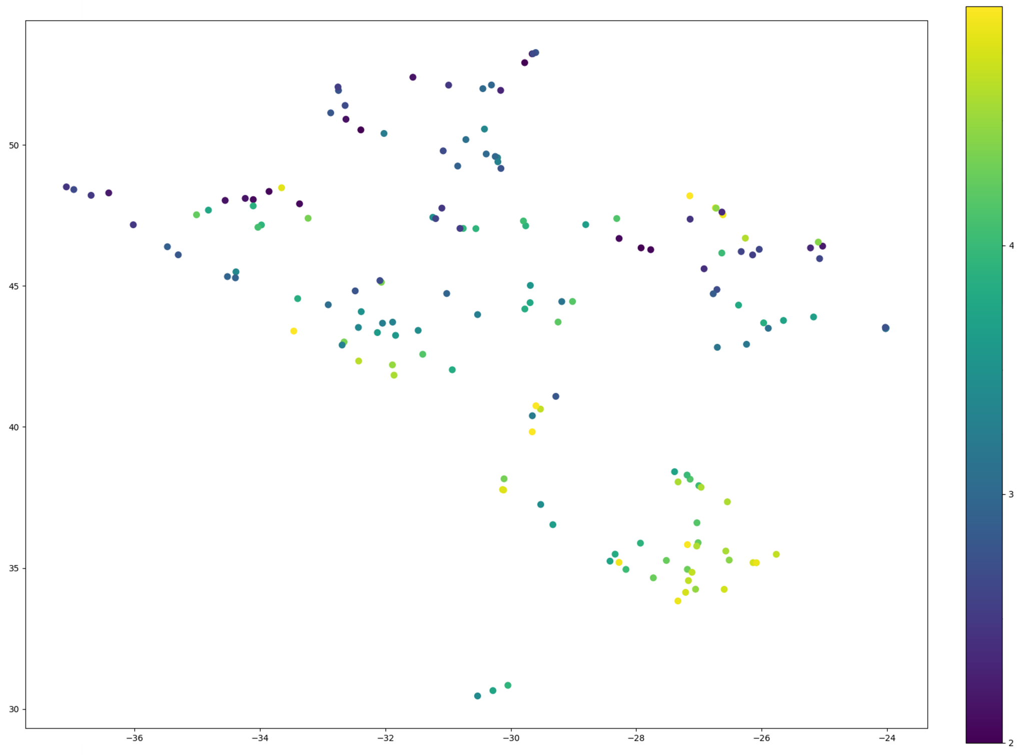


(a) Size.



(b) First coordinate value.

Figure 7.7: t-SNE projections showing distribution according to geometric parameters across the circular end pocket class for the parameter prediction model.



(c) Second coordinate value.

Figure 7.7: t-SNE projections showing distribution according to geometric parameters across the circular end pocket class for the parameter prediction model.

representation of the vector space. Whilst a potentially interesting area for further exploration, investigation into these hypotheses is outside of the scope of this thesis.

In this section, no projections have been presented showing relationships between feature depth and positioning in 2D projections. The reason for this is simple: during training, the depth parameter was ignored for the sake of simplicity. As a result, no observable relationship between depth and projected position can be observed, with the distributions resembling the random positioning shown in Figure 7.2. This result is unsurprising, and largely uninteresting, but should be briefly noted here.

7.4 Attentive Pooling Models

In this section, the results obtained for the geometric transformation, and class transformation models will be discussed.

It should be noted here that these models were trained to interpret input 'sentences' of nodes taken from specified levels in the input data tree. In order to compare representations with the other models considered in this chapter, dimensionality reduction techniques are applied to the single-vector output produced at the highest level of the tree, a vector representation which was not actually used during training. Although results show that information is successfully encoded in this vector in a similar manner to the other models, a more accurate representation of the model output might be the two 'sentences' of nodes representing edge and face level information. It is possible that the application of dimensionality reduction techniques to these sentences would produce interesting results. However, for the purposes of straightforward comparison between model outputs, such results are not presented here.

7.4.1 Geometric Transformation Model

Class Clustering

Figure 7.8 shows the PCA and t-SNE projections for the randomised single-feature dataset, produced using the geometric transformation model.

In the case of both PCA and t-SNE, the observed class clustering is very similar to that seen for the parameter prediction model. This similarity can be understood to be the result of the similarities between the two tasks for which the models were trained. Both models were trained to recognise feature class, but to also identify geometric parameters, resulting in strong clustering according to feature class, but not strong enough to be linearly separable in two dimensions using PCA projections. Additionally, the encoder for the geometric transformation network was initialised using weights learned by the parameter prediction network. Similar behaviour of the two encoders at test-time may imply that the geometric transformation model encoder was only minimally fine-tuned during training.

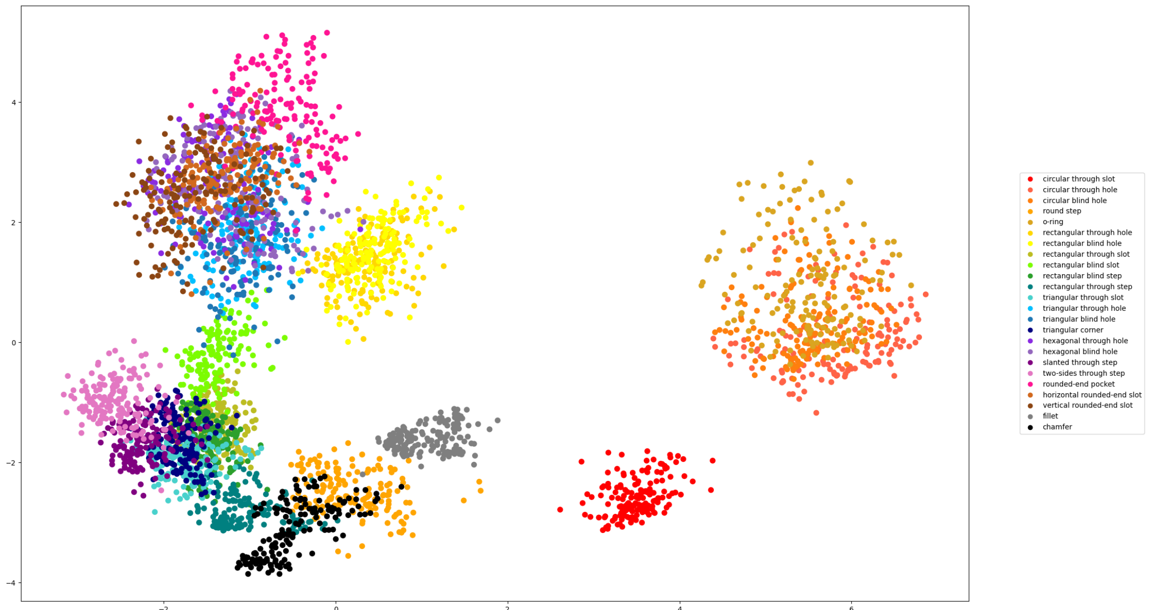
Interestingly, the class clustering appears to be slightly stronger in the geometric transformation model, indicating that information relevant to feature class is more effectively maintained in the encoded vectors produced by this model.

Encoding of Geometric Parameters

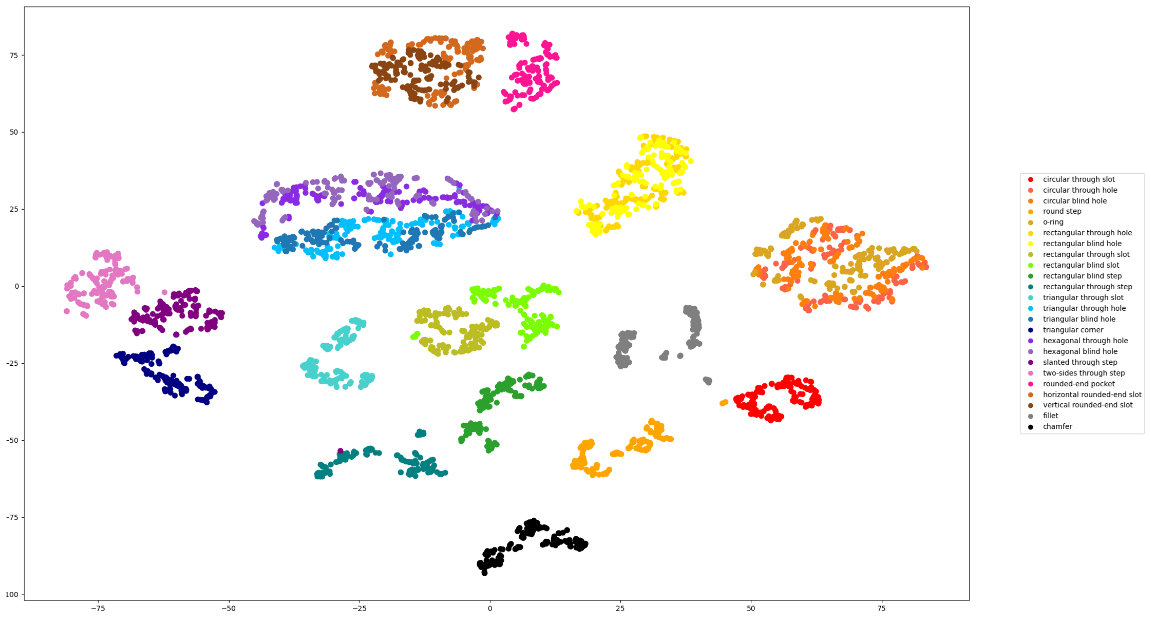
For direct comparison to the performance of the parameter prediction model, the same feature classes will be used in this section when presenting examples to show distribution according to geometric parameters.

Figure 7.9 shows the distribution of t-SNE projections according to size for the chamfer class, a feature class which is defined using a single parameter.

Here, a similar distribution to that shown in Figure 7.4 can be seen. The positioning of instances within the class cluster is clearly defined by the size of the feature, with a stronger correlation seen for smaller feature sizes. This result reinforces the idea that the two encoders behave in a very similar



(a) PCA projections



(b) t-SNE projections, initialised with PCA

Figure 7.8: PCA and t-SNE projections showing class clustering for the geometric transformation model.

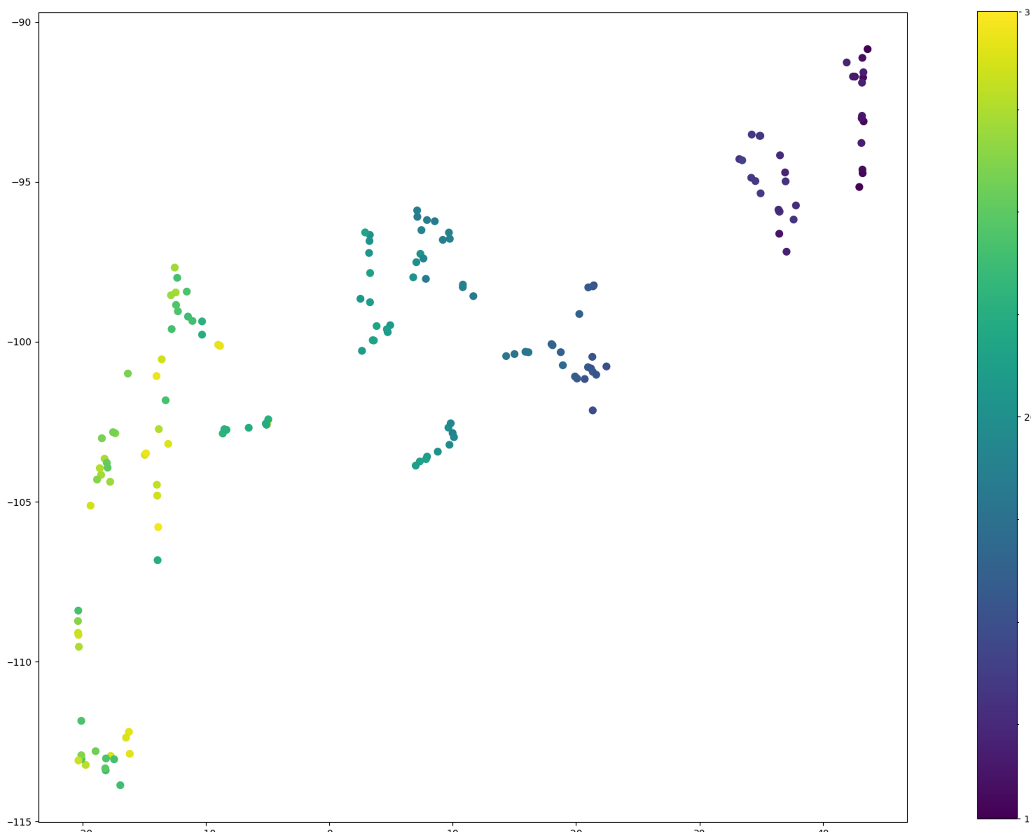


Figure 7.9: t-SNE projections showing distribution according to size across the chamfer class for the geometric transformation model.

manner.

Figure 7.10 shows the t-SNE projections, coloured according to size and position, for the circular through slot class, a feature class defined using two geometric parameters.

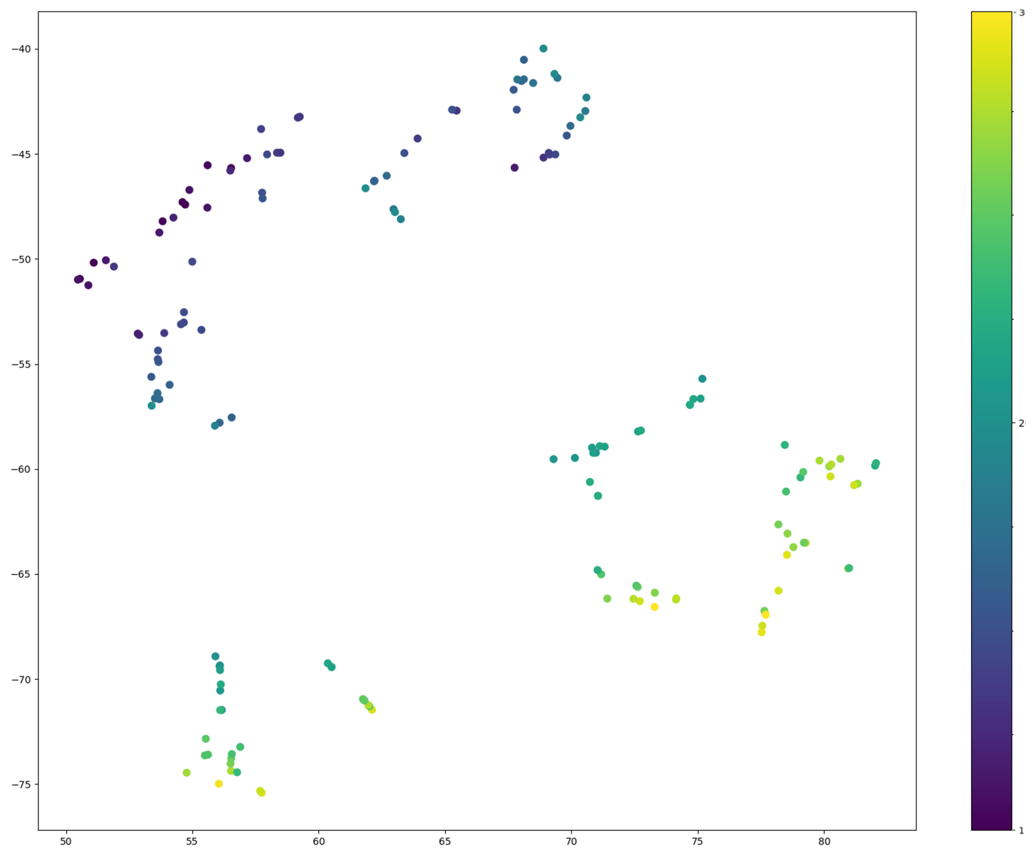
Again, the behaviour displayed is very similar to that of the parameter prediction model, with individual data points organised according to two understandable axes.

Figure 7.11 shows the distribution of t-SNE projections according to geometric parameters for the rectangular through slot class, a feature class defined using three geometric parameters.

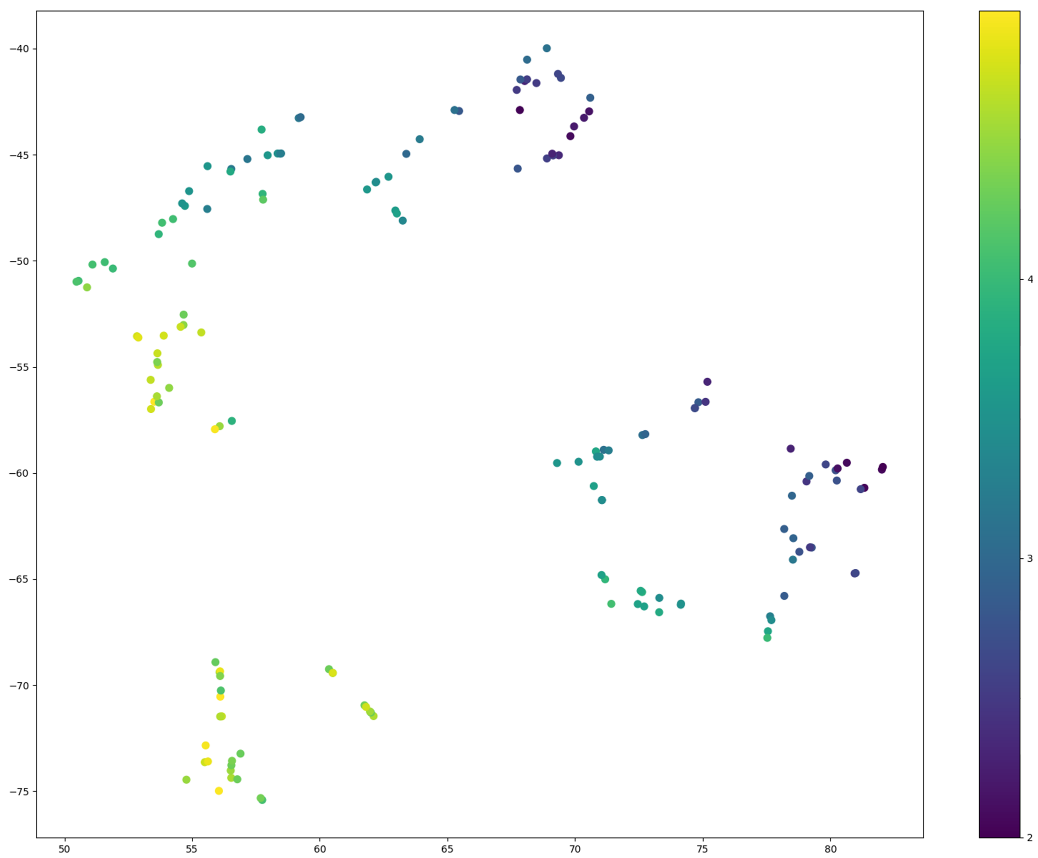
Here, we can observe the first significant difference between the parameter prediction and geometric transformation models. The results shown in Figure 7.6 show that, for the parameter prediction model, the two position parameters appear to share an axis, with size representing a second, orthogonal axis. In contrast, in Figure 7.11, the two clearly understandable axes represent the two coordinate values, with distribution according to size not following a simple pattern in this representation.

As discussed in Chapter 6, this may be indicative of the relative complexity of identifying single position changes. Whereas the parameter prediction model was perhaps able to combine the interpretation of the two coordinate values into a joint position representation, the geometric transformation model must recognise the two coordinates as separate parameters in order to correctly distinguish between a single position change and two position changes. This seems to have resulted in encoded representations in which position is fundamental to the organisation of the vector space and size, whilst clearly represented based on the high accuracy predicting size transformations shown in Figure 6.3, is encoded in a more complex manner which cannot be clearly observed in a two-dimensional representation.

Figure 7.12 shows the distribution of t-SNE projections according to geometric parameters for the o-ring class, an example of a feature class defined using all four geometric parameters.

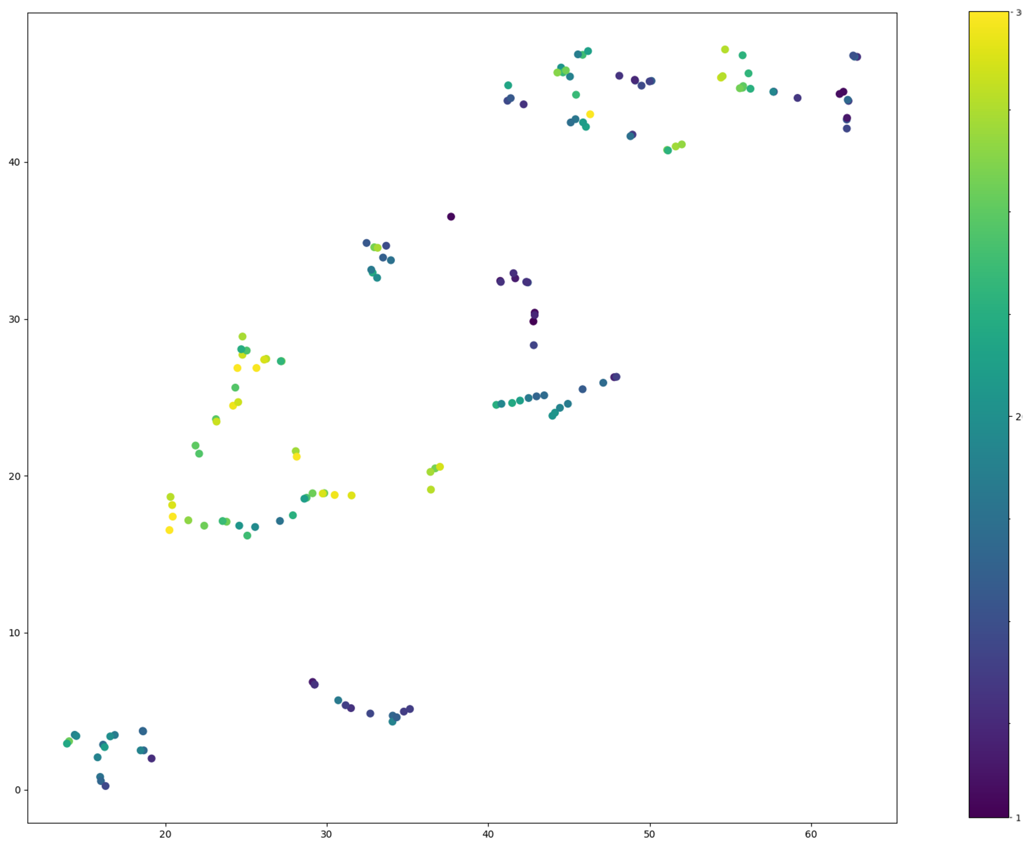


(a) Size.



(b) Position.

Figure 7.10: t-SNE projections showing distribution according to geometric parameters across the circular through slot class for the geometric transformation model.


(a) Size.

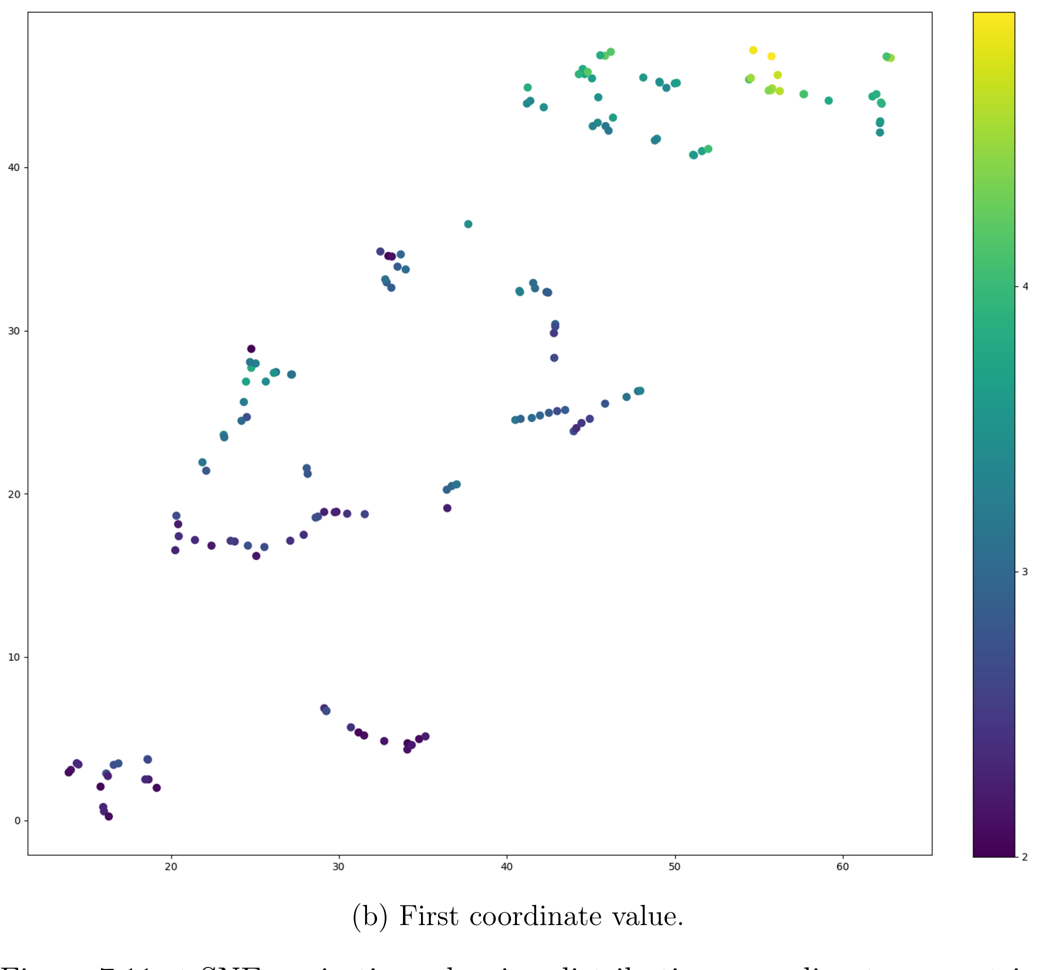
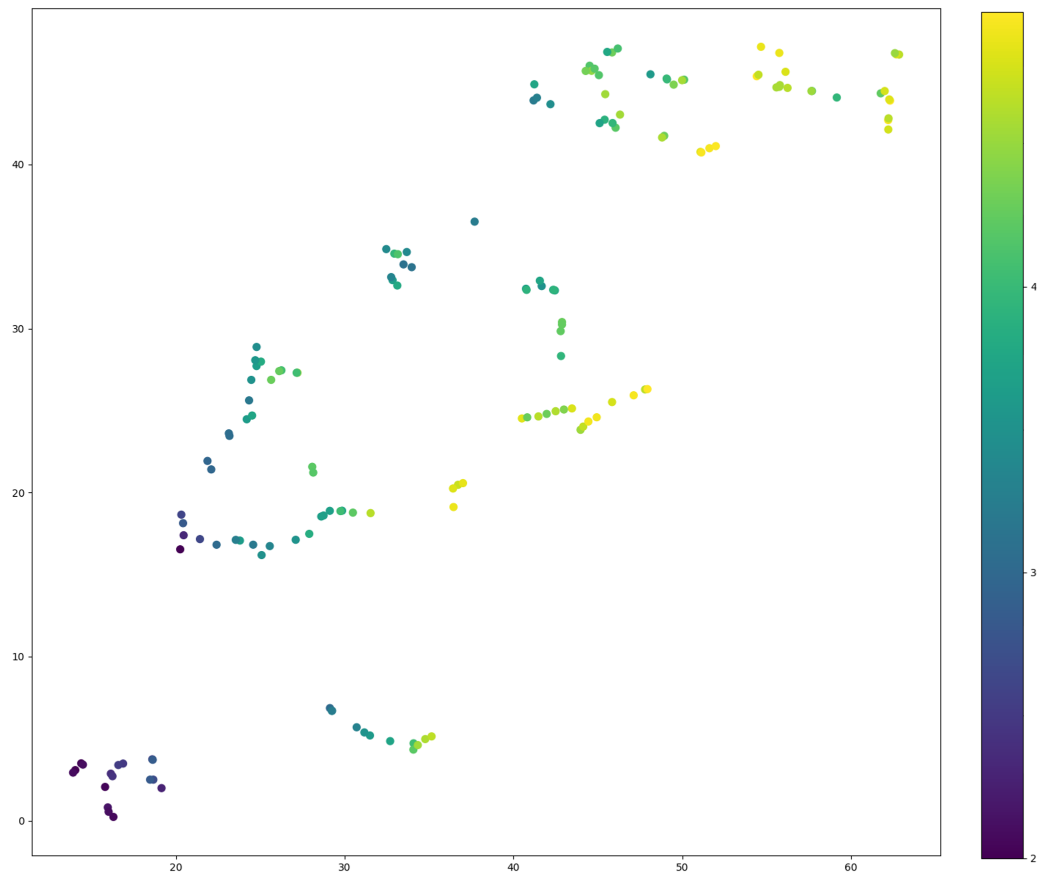
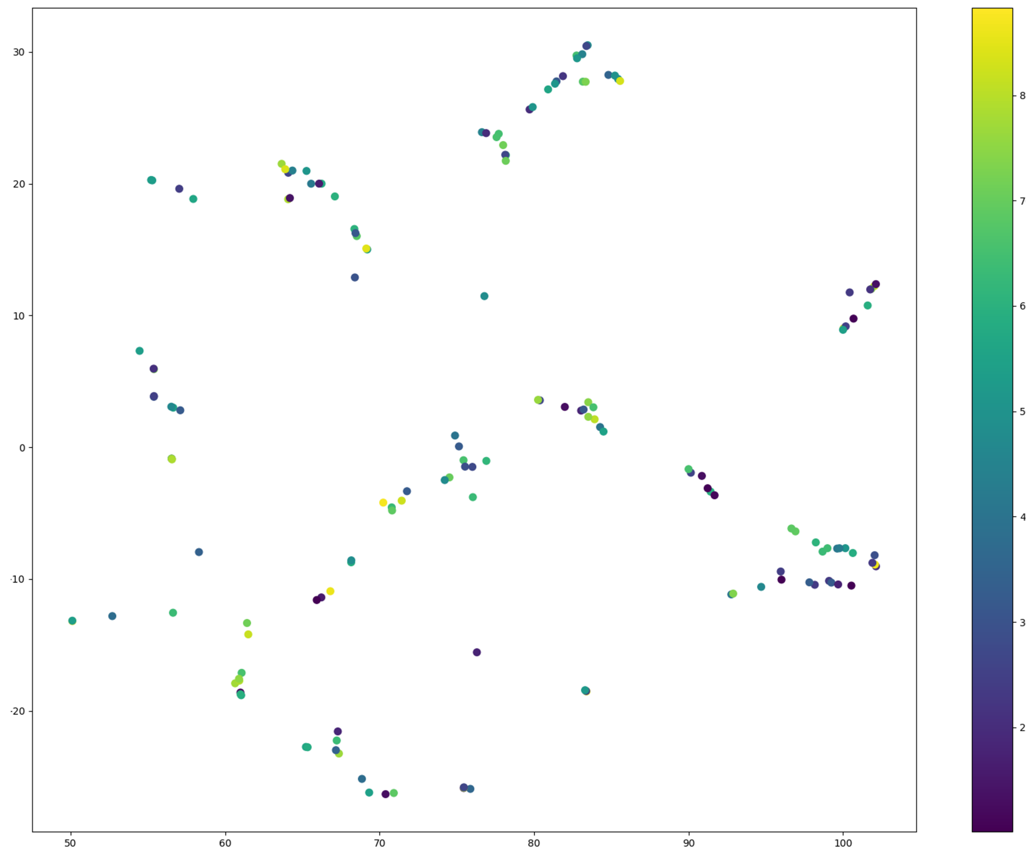

(b) First coordinate value.

Figure 7.11: t-SNE projections showing distribution according to geometric parameters across the rectangular through hole class for the geometric transformation model.

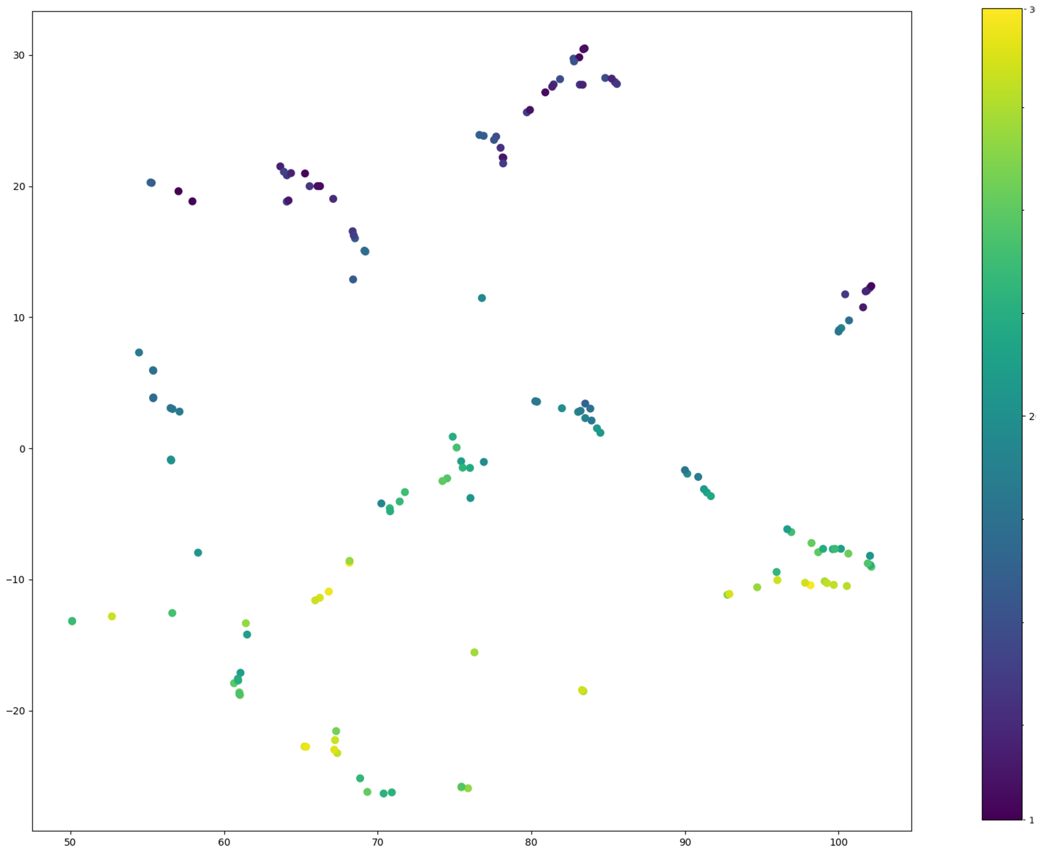


(c) Second coordinate value.

Figure 7.11: t-SNE projections showing distribution according to geometric parameters across the rectangular through hole class for the geometric transformation model.

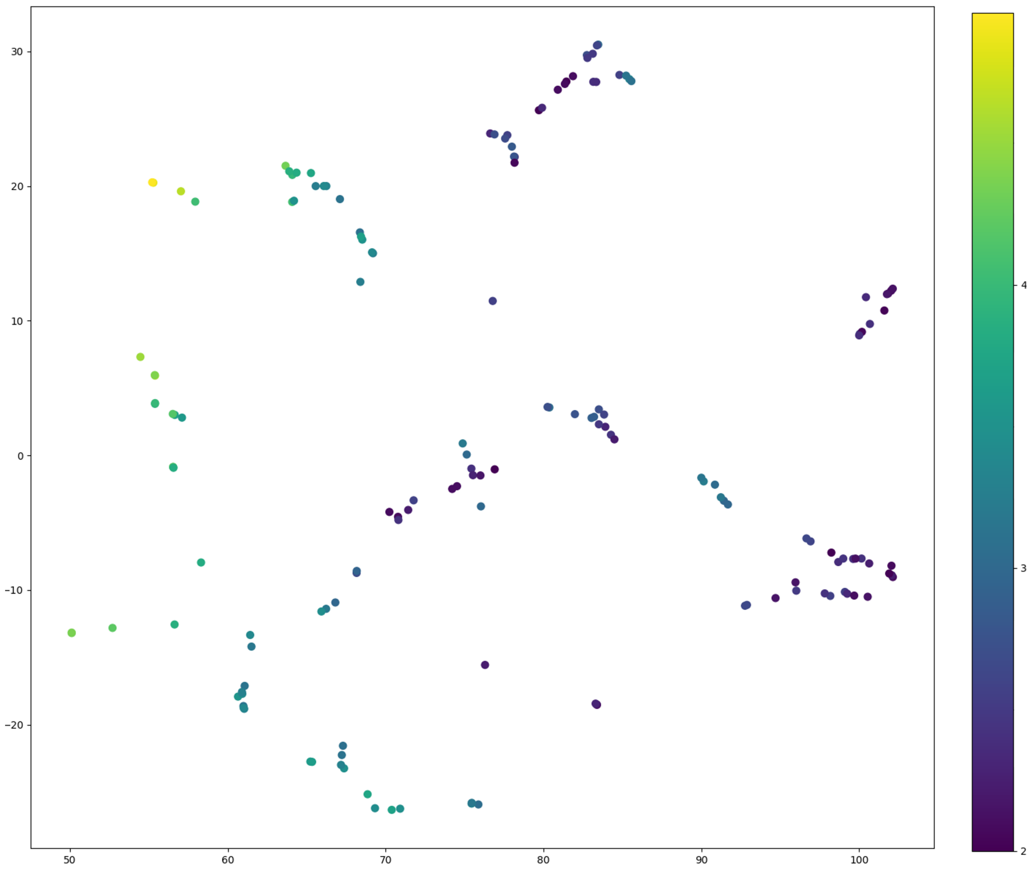


(a) Depth.

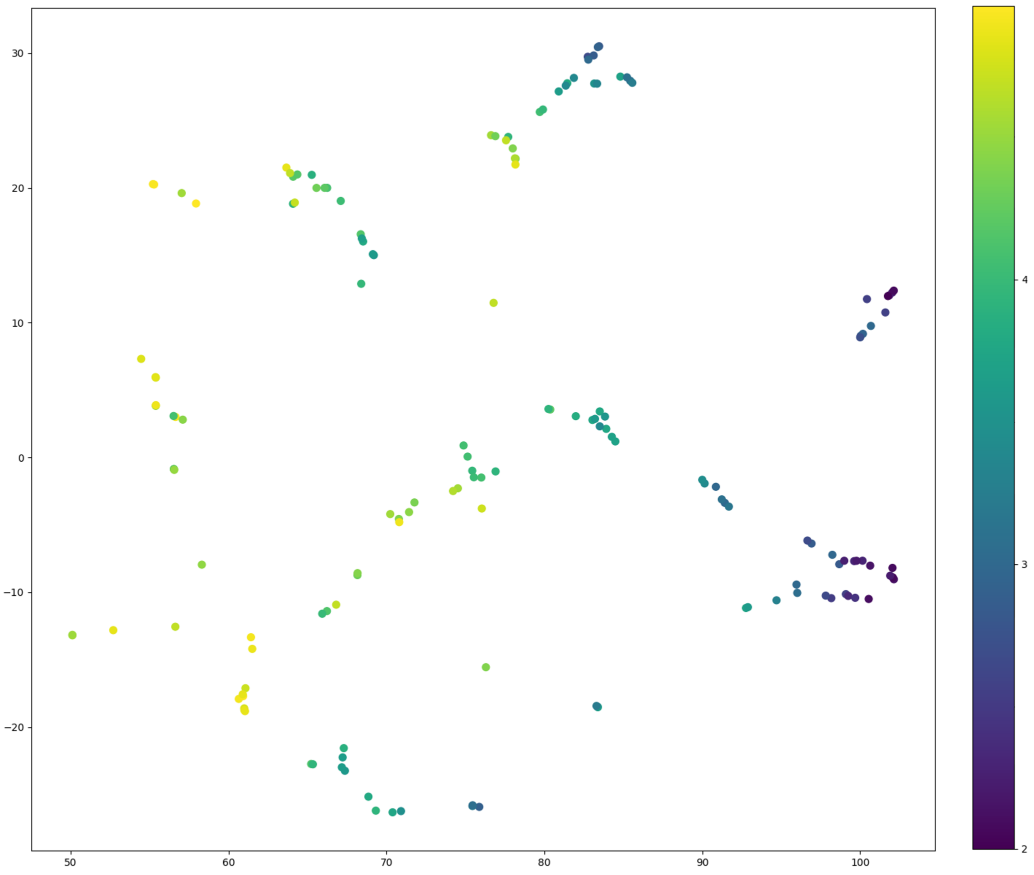


(b) Size.

Figure 7.12: t-SNE projections showing distribution according to geometric parameters across the o-ring class for the geometric transformation model.



(c) First coordinate value.



(d) Second coordinate value.

Figure 7.12: t-SNE projections showing distribution according to geometric parameters across the o-ring class for the geometric transformation model.

In this case, behaviour appears similar to that of the parameter prediction model, with one recognisable axis representing the size parameter, and one shared by the two coordinate values.

7.4.2 Class Transformation Model

Class Clustering

Figure 7.13 shows the PCA and t-SNE projections for the randomised single-feature dataset, produced using the class transformation model.

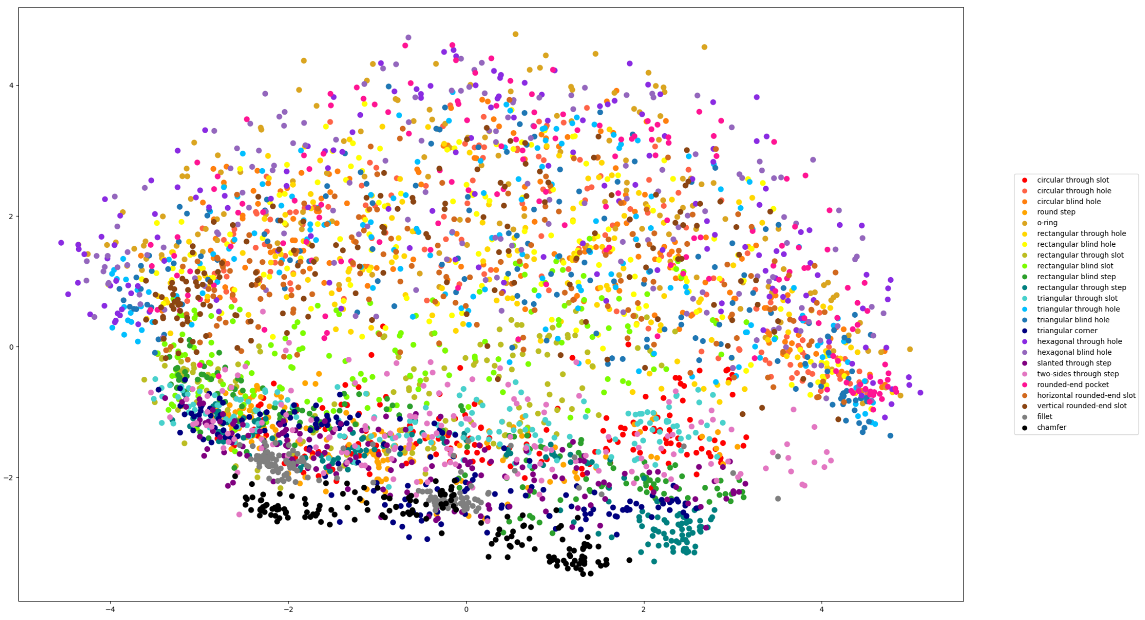
For both projection methods, class clustering is clearly weaker than that for any model considered thus far. Here, PCA projections, whilst still showing some correlation according to feature class, are insufficient to separate instances by class. When performing t-SNE projections, class clustering can still be observed. However, this clustering is not as strong as that seen for previous models, with more overlap between classes and less separation between clusters.

This result may be explained by considering the task that this model was trained to perform. In order to compare instances of different feature classes, the class transformation model would need to align instances of different classes within the vector space in a way that is not necessary for any other model considered here. Therefore, it follows some intuitive logic that the vector space is not as clearly organised according to feature class, but instead according to other factors.

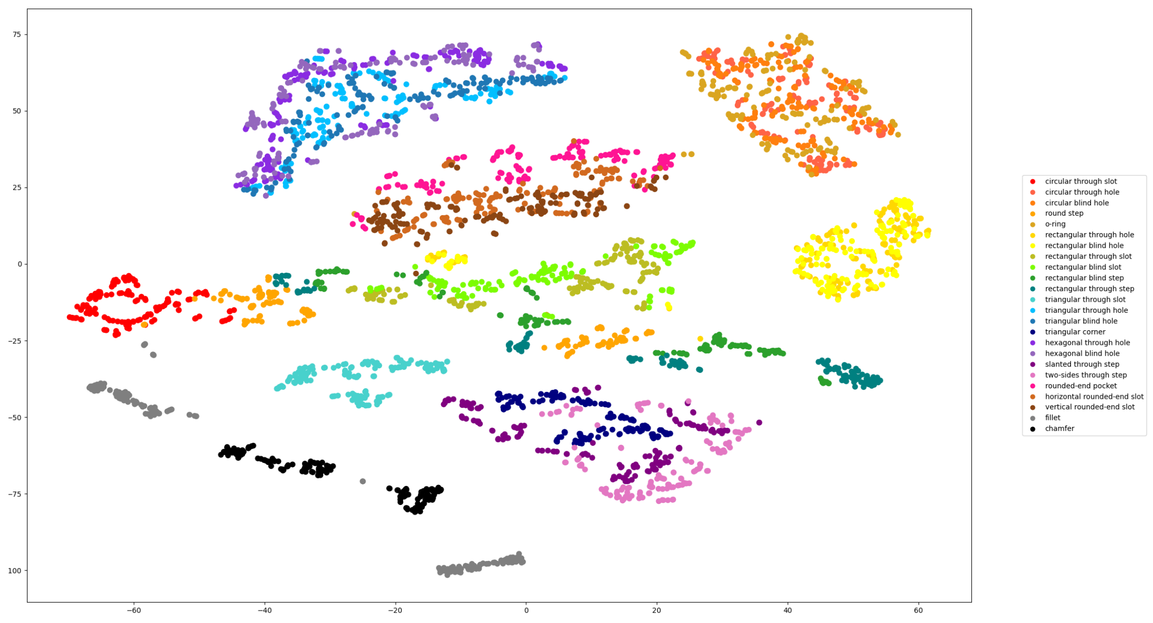
Encoding of Geometric Information

Figure 7.14 shows PCA projections produced using the class transformation model, categorised according to geometric parameters.

When considering this figure, the reason for the limited class clustering becomes clearer. Whereas previously discussed models produce a vector space organised according to feature class, with correlation based on geometric parameters observable within class clusters, the class transformation

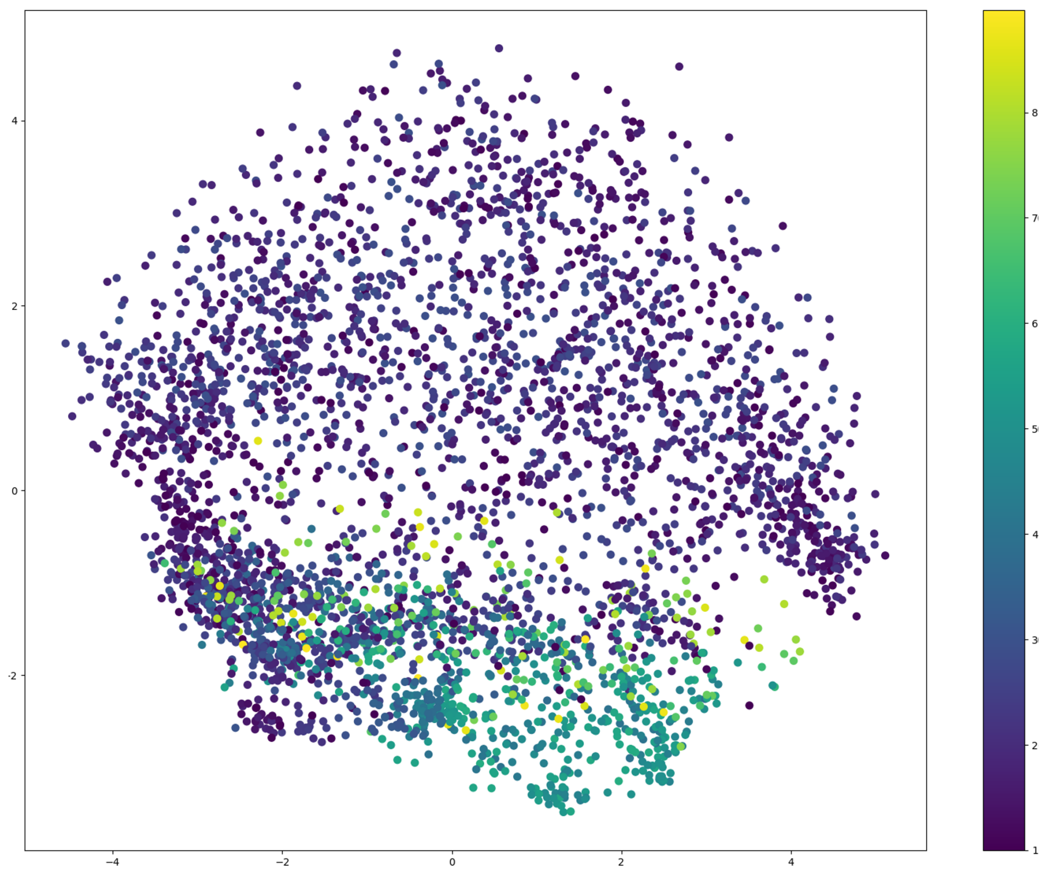


(a) PCA projections



(b) t-SNE projections, initialised with PCA

Figure 7.13: PCA and t-SNE projections showing class clustering for the class transformation model.


(a) Size.

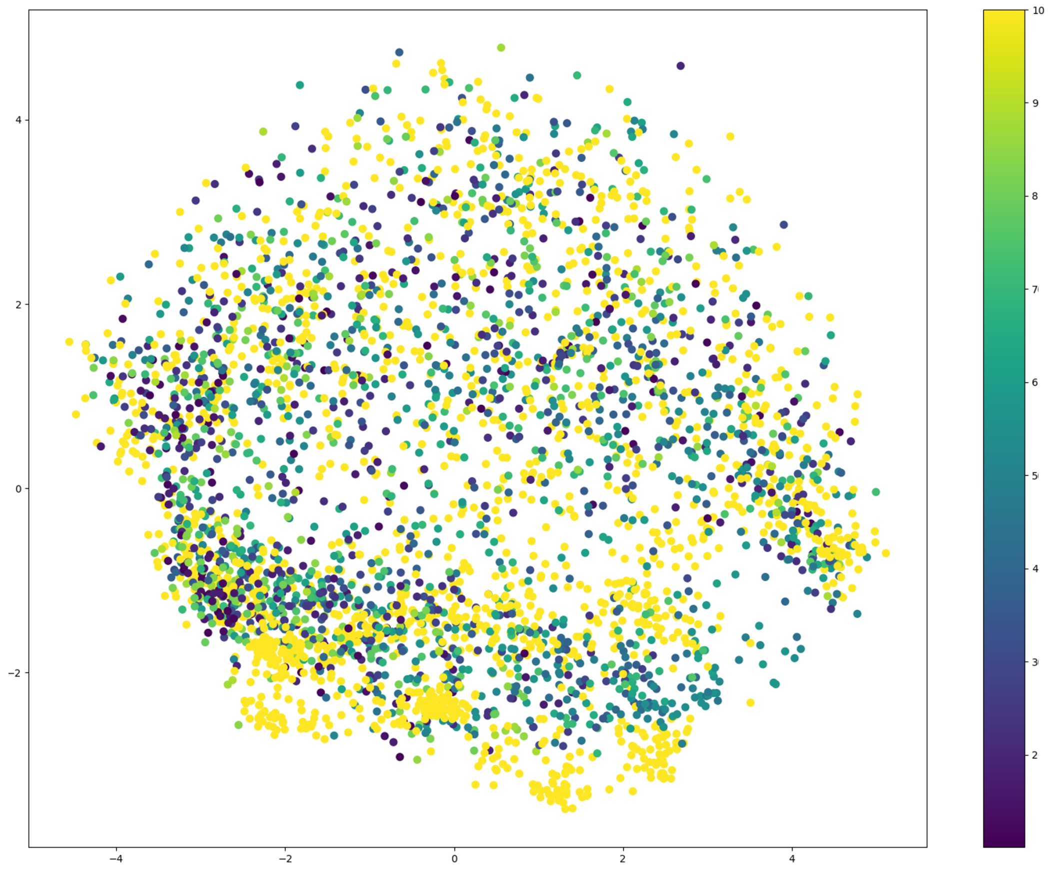
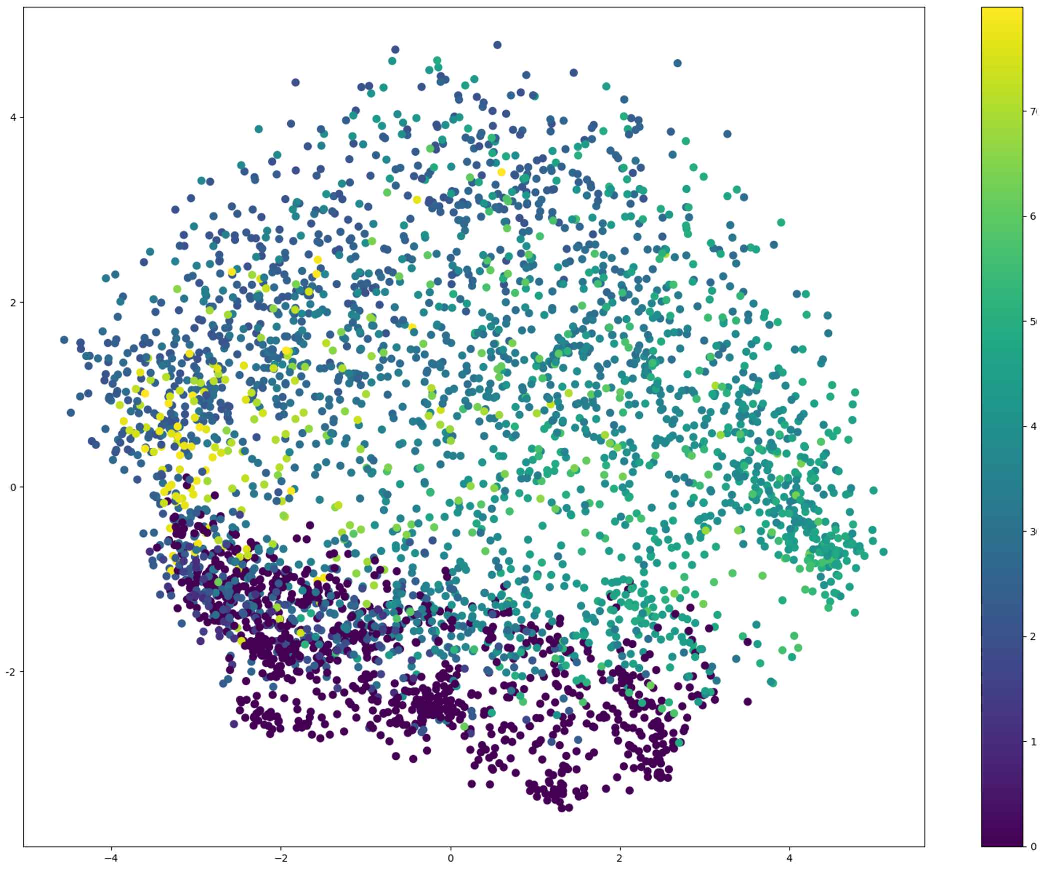
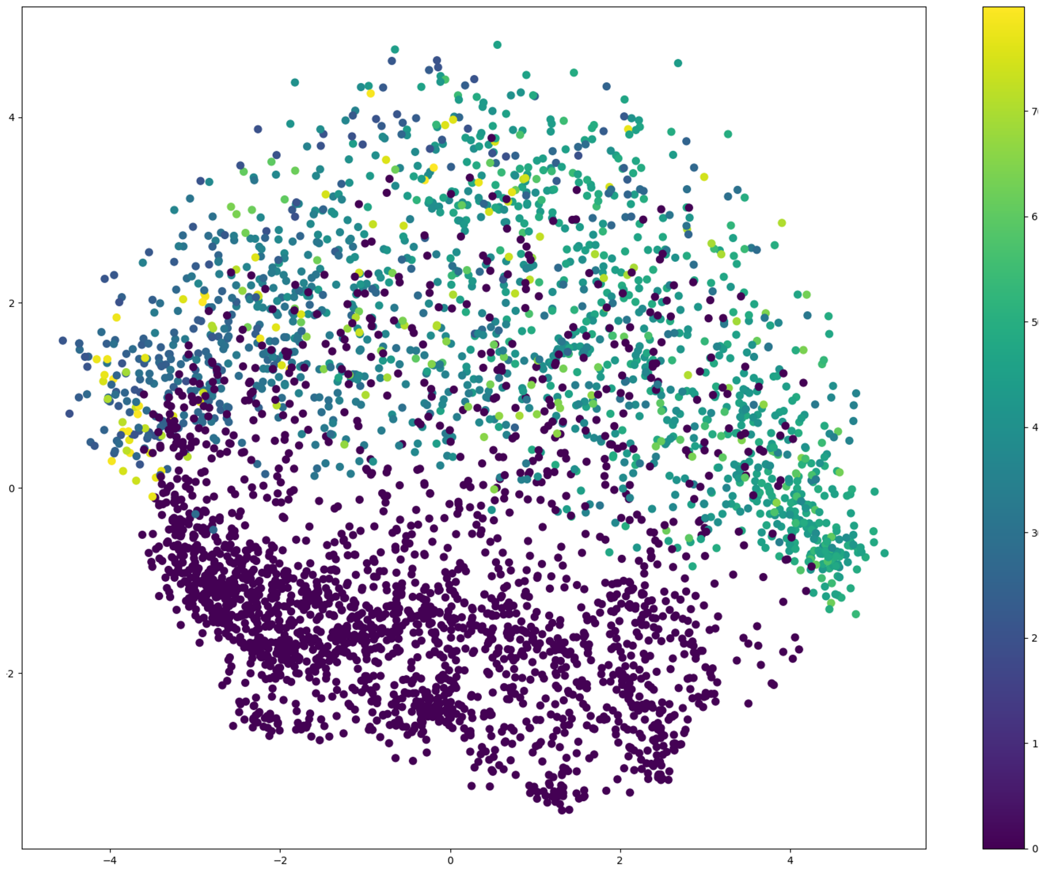

(b) Depth.

Figure 7.14: PCA projections for the class transformation model, showing distribution of the entire dataset according to parameter values.



(c) First coordinate value.



(d) Second coordinate value.

Figure 7.14: PCA projections for the class transformation model, showing distribution of the entire dataset according to parameter values.

model produces a vector space which is primarily organised according to geometric parameters, with feature class taking on a secondary importance.

Separating the t-SNE projections by feature class has similar results to those shown for the parameter prediction and geometric transformation models, so further figures will not be presented here.

7.5 Chapter Summary

In this chapter, the output vector representations produced using four models have been presented. It has been shown that vector spaces can be organised according to feature class, as well as according to geometric parameter values. The vector space is shown to have significantly varying properties for models trained for different tasks.

The analysis performed here has produced some promising results regarding the interpretability of the output vector encoding. However, the tests performed here amount to fairly simple interpretation of the vector space for analysis purposes. There is potential for significant further research into the possibility of direct traversal within the vector space, bypassing the need to run the class or geometric transformation models entirely.

Chapter 8

Conclusions

8.1 Chapter Overview

This thesis has presented a novel approach to the task of 3D CAD model interpretation, using a recursive neural network to encode the information stored within a STEP file. Six encoder-decoder models have been developed to evaluate the success of this approach when applied to the tasks of machining feature recognition, geometric parameter prediction and comparison between CAD models.

In this chapter, the final conclusions of this work are presented. Section 8.2 summarises the contributions made by each chapter. Section 8.3 presents the final overall conclusions of this thesis. Finally, Section 8.4 discusses the potential for future work exploring this topic further.

8.2 Chapters and Contributions Summary

8.2.1 Chapter 3 - Methodology

In Chapter 3, the development of a novel method for intelligent interpretation of 3D data is presented. This method primarily consists of two parts: the design of a STEP parser which encodes the data from a STEP file into a hierarchical data structure; and the implementation of TreeLSTM [27], a recursive neural network which acts as an encoder network, intelligently compressing the information from an input data tree into a single-vector format.

The development of this method represents the core contribution of this thesis, with results presented in subsequent chapters representing a series of investigations into the potential performance, advantages and limitations of this approach.

8.2.2 Chapter 4 - Machining Feature Recognition

Chapter 4 explores the task of machining feature recognition, the identification of simple manufacturing features within a CAD model.

In this chapter, two encoder-decoder models are presented for this task.

First, a model using a simple fully-connected decoder to perform the task of single-feature classification, the prediction of machining feature class in a dataset of CAD models containing only a single feature. This model represents an initial proof-of-concept for the approach presented in this thesis, performing the simplest possible task with an accuracy approaching 100%.

The second model presented in this chapter utilises a recurrent decoder network to perform multi-feature recognition, the prediction of multiple machining feature classes for a dataset of CAD models containing multiple machining features. The multi-feature recognition model represents the application of the method to a more complex, realistic task. Results demonstrate performance approaching state-of-the-art when compared against existing methods, but also demonstrate a key advantage of this approach over more traditional methods, the ability to interpret small detail within a CAD design without any observable drop in performance.

8.2.3 Chapter 5 - Parameter Prediction

In Chapter 5, an autoencoder-inspired approach is presented for the prediction of geometric parameter values in a CAD model containing a single machining feature.

This model represents an initial experiment into the feasibility of this approach, using simplistic data to extract simple geometric parameters, with an envisaged future application to more complex CAD part designs defined using any number of geometric parameters.

Results demonstrate clear correlation between real and predicted values but accuracy of predictions remains a limiting factor, and there exists a high number of undesirable outliers. Although promising as an early result, significant future work would be recommended to further explore and optimise this approach in order to produce improved results and increase complexity in the dataset and task.

8.2.4 Chapter 6 - CAD Model Comparisons

In Chapter 6, three models are trained to perform comparisons between two CAD models.

As an initial exploration into this type of task, a model with a simple fully-connected decoder architecture is trained to identify geometric congruence between CAD models, effectively identifying whether two CAD models are identical or not.

More complex comparisons are performed using two attentive-pooling models, trained to identify whether two CAD models can be connected using a single geometric transformation or contain similar examples of different classes of machining feature.

8.2.5 Chapter 7 - Output Vector Analysis

Finally, Chapter 7 contains analysis of visual representations, produced using dimensionality reduction techniques, of the vector space generated by the single-vector encodings representing compressed information from input STEP files.

A key goal for the work presented in this thesis is to develop an encoder network capable of producing output single-vector representations in which as much information as possible is maintained. Such an encoder could then, in theory, be paired with various designs of decoder network, for application to varying and complex tasks. In Chapter 7, the interpretability of the vector space is discussed, to investigate how successfully information from the input STEP file is encoded.

It is shown that models trained for machining feature recognition encoded only simple shape information, with instances clustered according to machining feature class but no observable correlation between other parameters and positioning within the vector space. When analysing models trained to perform parameter prediction or to identify specific transformations, we can observe a vector space organised according to machining feature class as well as by the values of certain geometric parameters.

8.3 Thesis Conclusion

This thesis has proposed a method for intelligent interpretation of 3D data, in which a recursive neural network is applied directly to the hierarchical data structures found in a STEP file.

The methods for 3D CAD data interpretation presented in this thesis represent an entirely novel approach, one which prioritises the preservation of real model data whilst simultaneously benefiting from the advantages of complex behaviour and flexibility provided through the use of purely learning-based algorithms. Due to the implementation of a generalised encoder network, the method can be adapted to a wide range of tasks, through the design of appropriate decoder networks.

Through the application to a machining feature recognition task, the potential of this method to compete with existing approaches when applied to a standard task is demonstrated. The adaptability of the encoder network is further demonstrated through the application to parameter prediction and CAD model comparison tasks. The interpretability of encoded vectors produced by the encoder network is demonstrated through analysis of the output vector space, indicating the success of the recursive encoder model in preserving relevant information from the STEP file input.

8.4 Further Work and Recommendations

The research carried out as part of this thesis represents the very early development phase of an entirely new approach to 3D data interpretation. The results presented here can be seen as initial experiments into the viability of the approach. For this reason, significant further work is recommended in development of further models, optimisation of current methods, and application to more complex, realistic datasets.

Application to Realistic Data

A key limitation of this research has been the datasets used. Randomised machining features datasets, whilst useful for training purposes and for testing initial performance, do not resemble realistic part designs. Therefore, an important avenue for future research is the application of these techniques to real CAD part datasets.

Ideally the machining feature recognition results should be corroborated using realistic data. This would likely involve expanding the machining features dataset for use as training data, to ensure the inclusion of all features present in the realistic test dataset and increase variation, such as using a non-uniform starting block for the multi-feature dataset.

Further advantages would come from applying the parameter prediction and CAD model comparison models to realistic data.

For the parameter prediction model, application to complex part designs with a potentially high number of parameters defining a particular instance within a category could be used to test the limitations of the model. In the datasets used in this thesis, parameters are simple size and position values. In realistic parts, the parameters defining a design might be more complex and varied. For instance, a particular design of screw can be categorised according to type of thread and number of threads, in addition to more straightforward length and diameter parameters. Investigation into the ability of the model to predict these parameters, which may be represented in a more complex manner in the STEP file would give a more advanced view of model performance as well as the potential viability of this particular approach.

The CAD model comparison networks have here been used to identify specific transformations between simple CAD models. A similar approach could be used with realistic data to perform more general comparisons between complex part designs, or to identify transformations using a wider range of parameters.

For both the parameter prediction model and the comparison models, the use of realistic data would test the models' ability to encode the information from more complex STEP data structures, and to retain information pertaining to a significantly higher number of parameters.

Direct Application to Problems in Industry

In relation to the use of realistic data, which would likely need to be obtained through collaboration with industrial partners, a key next step in this research is the direct application to real industrial applications; using these methods to attempt to solve real, specific problems in industry, identified through industrial partnership.

One potential application for the feature recognition model would be the automatic identification of machining features to increase automation, and therefore efficiency, in the process of transitioning between a CAD part design and the manufacturing processes necessary to produce the part. For this application it would be desirable to improve accuracy of predictions, as well as to implement localisation; not only identifying *which* features are present in a STEP file, but *where* in the CAD model they appear.

Other potential applications relate to the automatic organisation and traversal of large databases of parts belonging to a manufacturing company. Through further development of the parameter prediction or CAD model comparison models, the generalised vector representation of STEP files could continue to be improved upon, potentially resulting in a traversable vector space, allowing for easy navigation of large model databases. Development of this type of technology would likely rely heavily on close collaboration with industrial partners, as such a solution would likely be highly tailored to a particular company's CAD database and specific needs.

Attention Mechanisms

Another key area for future research is the use of attention mechanisms. These are explored here using the attentive-pooling models. However, attentive-pooling is only one form of attention, and not one designed for use with tree-structured data. One possibility for improving the performance of the encoder network would be the implementation of an attention mechanism directly into the tree.

An effective attention mechanism working at every level of the tree would also open up many possibilities, beyond improving performance of existing models. Reliable attention could be used to identify sub-trees within the STEP structure which correspond to specific features of the CAD model. This could be used to pull coordinate values directly from the STEP file, to predict parameter values with complete accuracy. It could even lead to the development of models capable of editing STEP files to produce a desired design.

Due to the complexity of STEP file data structures, these tasks are naturally highly complex. However, the successful implementation of attention throughout the tree would be the first step into interfacing more directly with the data tree, extracting specific details or even editing, making attention a highly significant avenue for further research.

Traversal of the Vector Space

In Chapter 7, dimensionality reduction techniques are used to analyse the structure of the vector space shared by the encoded vectors produced as output by the recursive encoder network. This is a useful analysis tool, for interpreting the behaviour of the neural network, but represents only one way of interacting with the vector space.

It has been demonstrated in this thesis that the underlying structure of the vector space is interpretable to varying extents for the different models trained. The next logical step would be to investigate the possibility of traversal of the vector space; attempting to define predictable transformations in the vector space which can move from a given start point to a desired end point. This could result in search-engine technology where the decoder network is only needed during training, and searches could be performed using stored encodings associated with each CAD model in the database, only requiring the encoder network when a new addition is made to the database.

This would require significant further work applying analysis techniques to the vector space, searching for predictable encodings. Additionally, developing increasingly complex or general training tasks to increase the amount of information represented in the output vectors may be necessary to manipulate the vector space into an increasingly understandable form.

Bibliography

[1] Z. Zhang, P. Jaiswal, and R. Rai, "Featurenet: Machining feature recognition based on 3d convolution neural network," *Computer-Aided Design*, vol. 101, pp. 12–22, 2018.

[2] P. Shi, Q. Qi, Y. Qin, P. J. Scott, and X. Jiang, "A novel learning-based feature recognition method using multiple sectional view representation," *Journal of Intelligent Manufacturing*, vol. 31, pp. 1291–1309, 2020.

[3] ——, "Intersecting machining feature localization and recognition via single shot multibox detector," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 5, pp. 3292–3302, 2021.

[4] R. Y. Zhong, X. Xu, E. Klotz, and S. T. Newman, "Intelligent manufacturing in the context of industry 4.0: A review," *Engineering*, vol. 3, pp. 616–630, 2017.

[5] B. K. Venu, V. Rao, and D. Srivastava, "Step-based feature recognition system for b-spline surface features," *International Journal of Automation and Computing*, vol. 15, pp. 500–512, 2018.

[6] M. A. Kiani and H. A. Saeed, "Automatic spot welding feature recognition from step data," in *Proceedings of the International Symposium on Recent Advances in Electrical Engineering (RAEE)*, vol. 4, 2019, pp. 1–6.

[7] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*, vol. 25, 2012.

[8] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. arXiv:1409.3215, 2014.

[9] F.-w. Qin, L.-y. Li, S.-m. Gao, X.-l. Yang, and X. Chen, "A deep learning approach to the classification of 3d cad models," *Journal of Zhejiang University-SCIENCE C*, vol. 15, pp. 91–106, 2014.

[10] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[11] C. R. Qi, H. Su, K. Mo, and 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)*, July 2017.

[12] "Standards," https://www.iso.org/standard/63141.html, 2016, [Accessed March 2024].

[13] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[14] D. Maturana and 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.

[15] G. Riegler, A. Osman Ulusoy, and A. Geiger, "Octnet: Learning deep 3d representations at high resolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[16] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, 2017.

[17] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, and A. Berg, “Ssd: Single shot multibox detector,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, October 2016.

[18] X. Yao, D. Wang, T. Yu, C. Luan, and J. Fu, “A machining feature recognition approach based on hierarchical neural network for multi-feature point cloud models,” *Journal of Intelligent Manufacturing*, pp. 1–12, 2022.

[19] H. Zhang, S. Zhang, Y. Zhang, J. Liang, and Z. Wang, “Machining feature recognition based on a novel multi-task deep learning network,” *Robotics and Computer-Integrated Manufacturing*, vol. 77, p. 102369, 2022.

[20] C. Yeo, B. C. Kim, S. Cheon, J. Lee, and D. Mun, “Machining feature recognition based on deep neural networks to support tight integration with 3d cad systems,” *Scientific reports*, vol. 11, no. 1, pp. 1–20, 2021.

[21] A. R. Colligan, T. T. Robinson, D. C. Nolan, Y. Hua, and W. Cao, “Hierarchical cadnet: Learning from b-reps for machining feature recognition,” *Computer-Aided Design*, vol. 147, p. 103226, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010448522000240

[22] Y. Feng, Y. Feng, H. You, X. Zhao, and Y. Gao, “Meshnet: Mesh neural network for 3d shape representation,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, July 2019, pp. 8279–8286.

[23] A. A. Salem, T. F. Abdelmaguid, A. S. Wifi, and A. Elmokadem, “Towards an efficient process planning of the v-bending process: an enhanced automated feature recognition system,” *The International Journal of Advanced Manufacturing Technology*, vol. 91, no. 9, pp. 4163–4181, 2017.

[24] M. Al-wswasi and A. Ivanov, "A novel and smart interactive feature recognition system for rotational parts using a step file," *The International Journal of Advanced Manufacturing Technology*, vol. 104, pp. 261–284, 2019.

[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 1997.

[26] R. Socher, C. C.-Y. Lin, A. Y. Ng, and 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.

[27] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," in *Proceedings of the Association for Computational Linguistics (ACL)*, July 2015.

[28] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and 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, p. 1724–1734.

[29] K. Cho, B. van Merrienboer, D. Bahdanau, and 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*, October 2014.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*, December 2017.

[31] J. Devlin, M. Chang, K. Lee, and 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*, June 2019, p. 4171–4186.

[32] X. Chen, C. Liu, and D. Song, "Tree-to-tree neural networks for program translation," in *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*, December 2018.

[33] D. Bank, N. Koenigstein, and R. Giryes, *Autoencoders*, 2023, pp. 353–374.

[34] M. Gogoi and S. A. Begum, "Image classification using deep autoencoders," in *IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, 2017.

[35] S. Semeniuta, A. Severyn, and E. Barth, "A hybrid convolutional variational autoencoder for text generation," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017.

[36] W. Wang, Y. Huang, Y. Wang, and L. Wang, "Generalized autoencoder: A neural network framework for dimensionality reduction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2014.

[37] S. Chaudhari, V. Mithal, G. Polatkan, and R. Ramanath, "An attentive survey of attention models," *ACM Trans. Intell. Syst. Technol.*, vol. 12, no. 5, 2021. [Online]. Available: https://doi.org/10.1145/3465055

[38] Z. Niu, G. Zhong, and H. Yu, "A review on the attention mechanism of deep learning," *Neurocomputing*, vol. 452, pp. 48–62, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092523122100477X

[39] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

[40] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[41] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 4960–4964.

[42] C. dos Santos, M. Tan, B. Xiang, and B. Zhou, "Attentive pooling networks," *CoRR*, vol. abs/1602.03609, 2016. [Online]. Available: http://arxiv.org/abs/1602.03609

[43] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, 2016, pp. 1480–1489.

[44] Y. Zhou, C. Liu, and Y. Pan, "Modelling sentence pairs with tree-structured attentive encoder," in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, 2016, pp. 2912–2922.

[45] Q. You, L. Cao, H. Jin, and J. Luo, "Robust visual-textual sentiment analysis: When attention meets tree-structured recursive neural networks," in *Proceedings of the 24th ACM international conference on Multimedia*, 2016, pp. 1008–1017.

[46] W. Liu, P. Liu, Y. Yang, Y. Gao, and J. Yi, "An attention-based syntax-tree and tree-lstm model for sentence summarization," *International Journal of Performability Engineering*, vol. 13, no. 5, p. 775, 2017.

[47] M. Ahmed, M. R. Samee, and R. E. Mercer, "Improving tree-lstm with tree attention," in *2019 IEEE 13th international conference on semantic computing (ICSC)*. IEEE, 2019, pp. 247–254.

[48] ——, "You only need attention to traverse trees," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 316–322.

[49] J. Ma, W. Gao, S. Joty, and K.-F. Wong, "An attention-based rumor detection model with tree-structured recursive neural networks," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 11, no. 4, pp. 1–28, 2020.

[50] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "ShapeNet: An Information-Rich 3D Model Repository," Stanford University — Princeton University — Toyota Technological Institute at Chicago, Tech. Rep. arXiv:1512.03012 [cs.GR], 2015.

[51] M. Law and A. Jain, "Incremental nonlinear dimensionality reduction by manifold learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 3, pp. 377–391, 2006.

[52] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[53] H. Hotelling, "Analysis of a complex of statistical variables into principal components," *Journal of Educational Psychology*, vol. 24, no. 6, p. 417–441, 1933.

[54] P. D. Waggoner, *Modern Dimension Reduction*, ser. Elements in Quantitative and Computational Methods for the Social Sciences. Cambridge University Press, 2021.

[55] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[56] G. E. Hinton and S. Roweis, "Stochastic neighbor embedding," in *Advances in Neural Information Processing Systems*, S. Becker, S. Thrun, and K. Obermayer, Eds., vol. 15. MIT Press, 2002. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2002/file/6150ccc6069bea6b5716254057a194ef-Paper.pdf

[57] V. Miles, S. Giani, and O. Vogt, "Recursive encoder network for the automatic analysis of step files," *Journal of Intelligent Manufacturing*, vol. 34, pp. 181–196, 2022.

[58] ——, "Approaching step file analysis as a language processing task: A robust and scale-invariant solution for machining feature recognition," *Journal of Computational and Applied Mathematics*, vol. 427, p. 115166, 2023.

[59] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference for Learning Representations*, 2015.

[60] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of the International Conference on Learning Representations*, 2015.

[61] V. Miles, S. Giani, O. Vogt, and R. Kafieh, "Recursive autoencoder network for prediction of cad model parameters from step files," *Procedia Computer Science*, vol. 232, pp. 3239–3246, 2024.

[62] H. S. M. Coxeter and S. L. Greitzer, *Transformations*, ser. New mathematical library ; 19. Washington: Mathematical Association of America, 1967, pp. 80–102.