Development of a Scalable Machining Feature Recognition System

by

Michael Lenover

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2023

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In this thesis, various pre-processing and training techniques were applied to improve the performance of a model trained with an existing machining feature recognition approach by Yeo et al. [35] using a smaller dataset that more effectively mimics the complexity of CAD models used in industry.

A GUI tool was developed to tag faces in CAD models with the corresponding machining features which would be necessary to resolve those faces. Using the encoding algorithm outlined by Yeo et al. [35], a tool was developed to generate feature vectors from tagged CAD models. Two CAD datasets were compiled. First, a dataset of generic CAD models was filtered from a larger dataset compiled by Koch et al. [12], selecting those models which could be manufactured using a 3-axis CNC machine. Second, a dataset of real-world CAD files used in CNC manufacturing was compiled from models contributed by individuals from the Unviersity of Waterloo, Hurco Inc. and Perfecto Inc.

Using the first dataset, three potential improvements to the feature recognition algorithm developed by Yeo et al. were explored: the incorporation of dropout to improve model stability and accuracy, the incorporation of ID3 tree pre-classification to reduce training time by reducing the size of the deep learning dataset without impacting classification accuracy, and the incorporation of crossover data generation to improve classification accuracy by reducing overfitting due to insufficient training data. It was determined that incorporating dropout improved the stability of the model and improved 5-fold cross validation accuracy. Further, it was determined that incorporating a 2-deep ID3 decision tree pre-classification marginally improved classification performance and was effective in reducing the size of deep learning training dataset. Crossover data generation did not improve model performance, and so was rejected. Using the model trained on the generic CAD dataset, and incorporating 10% dropout and a 2-deep ID3 tree, models from the real-world dataset were classified. This classifier was effective in classifying some simple features, but had poor accuracy overall. To improve this accuracy, an incremental learning technique was applied. The generic model was re-trained using samples from the real-world dataset, which improved the classification accuracy of the system.

Acknowledgements

I would like to thank William Melek, who provided the guidance necessary to develop my knowledge of machine learning.

Dedication

This is dedicated to my mother, Cheryl Lenover.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

To increase manufacturing efficiency, it is often necessary to automate parts of the manufacturing process. CNC, or computer numerical control, has been in development since the 1950s as a method of automatically executing movements of a robotic system. In the domain of manufacturing, CNC control has been used to automate subtractive machining processes such as lathing or milling. Often shortened to CNC machines, this equipment can automatically move a cutting tool into stock to remove material in such a way as to produce a desired part, in much the same way as a human operator might do.

To manufacture a part, the user must provide a CNC machine a set of instructions for what movements the machine must make. This set of movements is called a *toolpath*. In a typical industry workflow, a technician or operator would begin by importing a computer aided design (CAD) model into a toolpath planning program. Next, they would make decisions based on their own experience, the machine tools available to them, the required manufacturing speed, the specified part tolerances, and a variety of other factors to determine the best approach for manufacturing a particular aspect of the part. For example, a cylindrical hole can be manufactured by plunging a drill bit into a block of stock, or by plunging an end mill, or by performing a pocketing operation using a tool with a diameter smaller than the hole. Until recently, making these decisions about which approach to take (often dozens of times for a single part) was the responsibility of an experienced human operator.

Relying on a human operator has several limitations. CNC machines remain useful for producing thousands of identical parts, as the labour cost of generating the toolpath can be amortized over the many times that toolpath is executed. However, for one-off or low-volume production runs, the cost to generate the toolpath represents a significant

portion of the final manufacturing cost. This cost has the effect of making CNC technology inaccessible to many smaller businesses.

Relying on a human technician to plan the toolpath also limits the volume of orders a single manufacturing company can accept. The domain of machine planning focuses on analysing parts to be manufactured, estimating the manufacturing time of those parts, and distributing those parts among the available machines. Estimating the manufacturing time of a part (and by extension, estimating the manufacturing cost) requires many of the same decisions involved in toolpath planning to be made. By reducing the work required from a human operator, more manufacturing orders can be processed, and the reach of a single manufacturing company can be increased.

There has been a large body of research in the domain of toolpath planning automation over the past four decades. This research falls into one of two broad categories. First, curve planning, focuses on automating the path generation algorithms that determine how a machine tool traces out a pocket or surface based on the geometry defined by a designer [1] [30]. Typically, curve planning algorithms do not make assumptions about how the operator wants a part machined; what type of tool is being used, what the dimensions of the tool are, or whether to make multiple passes are often left to the operator to decide or change based on their judgement.

Secondly, and more recently, research into machining feature recognition has been conducted. To select a curve planning algorithm, it is useful to identify the high-level machining features required to manufacture a part. Given knowledge that a set of faces require a pocketing operation to construct, there are empirical and mechanistic models [28] that can make decisions about tool type, tool shape and depth of cut automatically. If machining features can be identified automatically, curve generation algorithms can be implemented with fewer human decisions, reducing manufacturing costs for small- and medium-sized manufacturing companies.

Recently, a model was developed by Yeo et al. [36] that leveraged machine learning to automatically identify machining features in a dataset of synthetically generated CAD files. Yeo et al. identified attributes of a CAD file that could be used to uniquely identify machining features from a list of machining features they selected. These attributes were used to generate a feature vector: a list of integers encoding relevant information from the CAD file. These feature vectors were tagged with a corresponding machining features using an automated process. After several thousand features vectors were encoded and tagged with their respective machining feature, they were used to train a machine learning system to identify machining features in parts which that system had not seen before. Once trained, a CAD model could be encoded using the scheme developed by Yeo et al., and

provided as an input to a machine learning model, which could identify which machining features, if any, could likely be found at a particular location of a given CAD file.

The approach proposed by Yeo et al. was successful at identifying machining features in synthetic CAD files with an accuracy of 93%. This is an impressive result, and indicated machining features can be accurately identified using a machine learning-based approach. However, the research conducted by Yeo et al. was limited to training and testing on a single synthetically generated dataset. This dataset was created by defining the geometry of each machining feature based on a fixed set of dimensions, bounding the minimum and maximum value of each of these dimensions, and randomly selecting values for each dimension. These features were then intersected with a solid volume to represent the stock material for a hypothetical part, with multiple features potentially self-intersecting. This approach was successful in creating a large number of complex machining features for which to train on, but was limited based what set of features could be described using this algorithm. Any representation of a hypothetical machining feature that cannot be generated by the specified algorithm will never be included in the training data, and consequently never learned by the model. Furthermore, different 3D modelling applications may resolve the same geometric information in different ways, which may lead to different encodings of particular machining feature which were not included in the training dataset. Finally, the distribution of machining features varies considerably between different manufacturing industries, which are not reflected in the tests presented by Yeo et al.

To develop a system that is more useful for small- and medium-sized manufacturing businesses, an extension to the work developed by Yeo et al. is presented. To achieve such an improvement, a model was developed and validated using training data extracted from an existing dataset of generic human-created CAD files, to avoid the concerns outlined above which arise from training on a synthetically generated training dataset. The machine learning model was based on the approach developed by Yeo et al. The learning algorithm was augmented with a variety of data pre-processing and canonical machine learning techniques, to improve the classification of features. Next, to estimate the classification performance of the system in a real-world environment, a dataset of real-world CAD files was collected and tagged using the same feature encoding approach. The model trained on the generic dataset was used to classify features from the real-world dataset. To improve the classification accuracy of real-world machining features, a re-training approach was developed that could be easily integrated with a hypothetical machinist workflow. In this way, a scalable machining feature recognition system that can identify machining features in real-world CAD files of machined parts was developed.

Chapter 2

Background

2.1 Existing Research

The domain of machining feature recognition has been widely researched from several different perspectives for over four decades. Some of the earliest work into machining feature recognition was detailed by Kuprianou [14]. Since then, several approaches have been investigated that refine existing techniques to improve the quality of classification, apply new techniques to offer greater flexibility, and present domain-specific applications of feature recognition techniques. The following section presents a selection of relevant machining feature recognition research, culminating with an analysis of the work this thesis will build upon.

2.1.1 Hint-Based Methods

One of the most conceptually straightforward approaches to machining feature recognition relies on using rules to parse information from a CAD model to define the machining features most likely to be used in manufacturing a section of a part. Some of the earliest research in this domain relied on data extracted from the part geometry exclusively [3]. At the same time, other research presented an approach that fused data from multiple sources; part geometry, in conjunction with tolerancing information and machining call-outs such as threading indicators, were used to identify features [32]. As research into this space evolved, it became clear that one of the main drawbacks of hint-based approaches lay in their simplicity. Early approaches presented too many possible solutions for which

machining features comprise a part and were computationally expensive [5]. Furthermore, a common element of most hint-based feature recognition methods up to this point was the evaluation of most likely or most similar features based on the provided hint. Approaching the machining feature recognition problem in this way simplifies the decision-making process, but can lead to incorrect estimates in cases where the machining feature is not clear. Instead, the method presented by Han and Requicha [5] focused on the discovery of a satisficing solution, rather than an optimal one. Even as these issues were addressed, hint-based approaches remained limited by how feature interactions were handled [33]. These concerns were addressed in part in that publication, but other approaches address these issues more effectively.

2.1.2 Graph-Based Methods

Graph-based methods, much like hint-based methods, rely on extracting geometry information from CAD files to produce localized estimates of machining features. In contrast to hint-based methods, these approaches draw from the domain of graph theory, and model features as a set of faces (nodes) connected by edges. Early research into these approaches began with work by Joshi [8], and remain an area of ongoing research [17] [18]. The success of these methods reinforces the effectiveness of approaches that rely on geometry information in distinguishing machining features.

2.1.3 Volume-Based Methods

Virtually all hint-based methods rely on a human designer to determine what information must be extracted from a model, and how that information is relevant in determining what machining feature or features most likely comprises a portion of a 3D model. Volume-based methods, in contrast, allow for the decomposition of models into their constituent machining features based purely on the geometry of those machining features.

One of the earliest implementations of a volume-based method was introduced by Tang and Woo, where they present an approach they dubbed "Alternating Sum of Volumes" or ASV [29]. Unfortunately, a major drawback of this algorithm was its tendency to not converge. This non-convergence was addressed by Kim [11], which was further refined by Waco [34], in which an algorithm was presented that provides a hierarchy of preferred machining features that can be re-arranged based on feedback from the machining sequence planner. The integration of feature recognition and machine planning systems allows for

greater control over what features are selected, to minimize machining time and to select for only those features that the available tools and machines are able to manufacture.

2.1.4 Cell-Based Methods

Cell-based methods aim to decompose a model into constituent features using the part geometry alone. In contrast to other methods, however, cell-based methods involve segmenting the part into regular volumes (cells) before recombining them to construct different machining features. Early work in this field was published by Sakurai [25] and Shah [26]. New developments in this field have been seen as recently as 2015 with work from Kim [9].

Cell-based methods are successful at handling complex feature-feature interaction, in contrast to many of the approaches previously discussed [27]. However, as they retrieve information from the volume envelope of the part, geometric features (such as planar surfaces vs. curved surfaces, or cylindrical features vs. prismatic features) can be misinterpreted.

2.1.5 Machine Learning Methods

Machine learning is a popular classification technique in a variety of domains that has become increasingly popular in recent years. Early investigations into the application of machine learning to the machining feature recognition problem began with work by Prabhakar and Henderson in 1992 [23]. Their paper presents an approach that encodes information extracted from the B-Rep model as a matrix of feature vectors, with each face in a part encoded as a single vector. The elements within a feature vector are selected to best distinguish between different machining features, and represent information such as what type of face is being analyzed (planar, cylindrical, etc.), number of adjacent edge loops, and edge convexity. This approach of encoding geometry information into a fixed-length vector allows these vectors to be treated as training samples in a feed-forward neural network. More information about neural networks is presented in the next section.

Recently Yeo et al. presented a culmination of many years of research into the neural network-based machining feature recognition domain [36]. Their paper presents a proposal for an end-to-end pipeline that can extract relevant information from a CAD file and present an estimate of the machining features contained in the part. This work draws heavily from their previous publication [35], that outlines what information is best to encode in the data vector to distinguish between different machining features most effectively. In total, they used 2236 data samples to train, test and validate the network, which was able to distinguish between 17 different machining features. My thesis will aim to extend Yeo

et al.'s work to improve its classification effectiveness using a variety of machine learning techniques, which will be introduced below.

2.2 Machine Learning

Machine learning has been a popular domain of research in recent years, with innovations such as GPT-4 [22] and generative AI [24] achieving mainstream press coverage and driving renewed interest in the technology. However, research into machine learning has been ongoing for over 70 years, with the underlying mathematical concepts having been developed even earlier. The following section presents a brief summary of machine learning concepts relevant to the research discussed in the remainder of this thesis. The majority of the concepts presented in this section are based on information contained in a textbook authored by Fieguth [4].

At its core, the goal of machine learning is to identify patterns in a set of data to automatically make decisions. In some cases, the pattern recognition can take the form of segmenting the data based on their similarity into groups called *classes*. This approach is known as *unsupervised learning*, and there are a variety of approaches that have been developed to achieve this goal. In other cases, a designer may wish to specify associations between classes and data points. For example, a researcher may wish to develop a system that learns to identify tumours in CT scan images. To achieve this, they could collect a dataset of different CT scan images, with labels indicating which of them include a tumour. Then, by analyzing this dataset, a machine learning model would be able to automatically identify what characteristics of an image indicate a tumour, and (ideally) be able to identify if a new image contains a tumour. This approach is known as *supervised learning*.

One of the earliest approaches to supervised learning draws inspiration from the human brain itself. *Neural networks* describe an approach that uses many simplified models of neurons arranged in a network to learn about a system. There are several different configurations of neural networks, but one of the simplest approaches is known as a *feed-forward neural network*. During training, information encoded as a vector of values is passed into an input layer of neurons, which signal to subsequent neurons based on the magnitude of the inputs. These signals propagate through the network, though multiple layers of neurons, into one or more output neurons. Each time a signal is passed from one neuron to another the signal is scaled by a connection weight, which can be initialized with some scheme or randomized at the start of training. The output from the network is compared to the tag associated with that training sample, and an error value is computed. The sign and magnitude of the error indicate in which direction and by how much the connection

weights should be modified. The errors are then passed through the network backwards, scaled by the existing network weights, until adjustments to every weight are calculated. This scheme is repeated for each sample in the training data set, referred to as an *epoch*. To train a network to classify a system, it is often necessary to iterate over the same data set multiple times, for hundreds or thousands of epochs. In Chapter 4, several feed forward networks are developed to classify machining features. Each of these networks are trained for 1000 epochs.

When constructing a machine learning system, it is necessary to take care when formatting the training data so that the data best fits the problem domain and learning technique. In the example proposed earlier, where a system is being designed to identify if a tumor is present in an image, the training images would be tagged with a single binary value (indicating whether the image contains a tumor or lacks a tumor). This value would be compared to the value from a single neuron in the output layer to produce the backpropagated error signal. For a classification problem, such as selecting which of a set of common objects (plants, animals, humans, cars, etc.) are in an image, a vector of binary values associated with each possible object can be tagged to each image, with each element in the vector indicating the presence or absence of a particular object. In the case where only one class can be present at a time (such as in the case of many implementations of machining feature recognition systems) a *1-of-n encoding scheme* is used. A 1-of-n encoding scheme takes the structure of a binary label vector, in much the same way as the previous example, but where only exactly one element in the vector is labelled as 1. The machine learning model constructed by Yeo et al., and consequently the model implemented in Chapter 4, is trained using a 1-of-n class encoding scheme.

Once an encoding scheme has been selected, several approaches can be used to augment the data so that the data better represents the problem domain and can be used to train an effective classifier. If multiple samples are present in the dataset that are identical, the resulting model can place a greater emphasis on correctly identifying those samples, at the expense of distinguishing between subtle differences between similar classes. Instead, researchers often remove duplicate samples from their dataset. Before training on either dataset collected in Section 3.4, duplicate samples were removed. Similarly, techniques have been proposed that pull from the domain of genetics to recombine samples of the same classes into new generations of artificial data, based on the distribution of values in the parent training samples [31]. This approach is known as *crossover*. An extension to the work developed by Yeo et al. is proposed in Chapter 4 that incorporates crossover to generate new training data.

To determine when to stop training a model, and to verify that the system is effective, it is necessary to evaluate its performance. A naïve approach for evaluating a supervised

learning model might involve training it on a set of data for some number of epochs, calculating the expected class for each sample in the training set, and determining what fraction of the training samples it could correctly identify. Unfortunately, evaluating the accuracy of a model using the same data that was used to train that model has the potential to obfuscate a system that has been overtrained.

Overtraining is a description of a machine learning model which, instead of learning the underlying properties of a given domain (for example, the characteristic color or shape of a tumor in an image), learns some properties present in the dataset that are not generalizable (for example, a note from a doctor or a watermark present on all images that contain a tumor). Overtraining can come about for many reasons, from class imbalance (where a few classes of data disproportionately represent the full training set) to a dataset that lacks sufficient diversity proportional to the complexity of the domain to a dataset that is too small. There are two common ways to combat overtraining.

First, when calculating the output for a training sample, an algorithm can randomly select some internal neurons to output no signal, typically with a likelihood of around 5 percent. In this way, simple strategies that rely on the activation of only a handful of neurons (and as a result, represent a model with little complexity) are not as successful. This approach is known as *dropout* and was developed by Hinton [6]. Second, instead of evaluating the performance of a model based on its ability to classify the data it was trained on, researchers typically split data into three groupings: a training set, a validation set, and a test set. The training set is used to train the model, in a manner as discussed before. The validation set is used to make changes to the model (such as changing the rate at which the neuron weights are updated each iteration, and the network structure) to improve its performance. Once a model is trained, the test set is used to evaluate its classification performance. The tests conducted in Section 5.1 and Section 5.2 incorporate training, test and validation datasets in this way to avoid overtraining.

Typically, the accepted approach for developing a deep learning classifiers involves training a system on a training data set multiple times, varying some aspect of the system that is to be evaluated. During training, the accuracy of the system is quantified by using the model to classify samples in the validation set, and comparing the model outputs to the ground truth labels associated with the validation training data. Then, once an optimal system has been developed, a final estimate of the system's accuracy is developed by classifying samples in the test data set, and calculating the proportion of samples which are classified correctly. This approach works well for large datasets which well represent the feature space of the classification problem. However, when there exists a risk that small changes in the composition of the dataset may significantly impact the validation accuracy, a more robust testing approach may be selected. K-fold cross validation is an approach that

involves involves training and validating the same dataset multiple times, and averaging multiple validation accuracy values to produce a better estimate of the model performace. In order to collect multiple validation accuracy values, the test and validation datasets are combined into a single training dataset, This single training dataset is split into (typically around 3-5) folds, groups of training samples with an equal or approximately equal number of samples. A model is trained using data from all but one of the folds, and the held-out fold is used to calculate the validation accuracy of the trial as before. This process is repeated, holding out a different fold each time. These validation accuracy estimates are then averaged to produce a cross validation accuracy of the model. The tests conducted in Section 5.1 and Section 5.2 also incorporate K-fold cross validation, to more effectively estimate the classification performance of each model.

An alternative approach for implementing a supervised learning system is a *decision tree*. Unlike neural networks, decision trees are simpler to implement, do not require many training iterations, and once constructed can be easily interpreted by a human. However, as a consequence of their simplicity decision trees are often not as effective in learning subtle differences between classes. Despite this, decision trees can be useful in scenarios where the complexity of a neural network is not required. Another extension to the work by Yeo et al. is proposed in Chapter 4 that incorporates a decision tree to classify machining features.

Entropy is a measure of disorder within a system; a dataset with higher entropy has a larger variety of classes that are evenly distributed, whereas a system with lower entropy has fewer classes with one or two classes representing most of the samples. A decision tree can be implemented by first calculating the entropy of the entire dataset based on equation 2.1.

$$Entropy(\mathcal{D}) = -\sum_{k=1}^{K} P(C_k)\log_2(P(C_k)) \tag{2.1}$$

In this equation, $\mathcal{D}$ represents some dataset with K classes, where each class represents a fraction of the total dataset equal to $P(C_k)$. Starting with the first variable in the data vector, the dataset is segmented into subsets for each possible value of that variable, where each dataset contains only samples where the chosen variable takes a single value. Next, the entropy of each subset is calculated using the same equation as before. Finally, by summing the entropy of each subset, scaled by the fraction of the total dataset that subset represents, a representation of the total information contributed by that variable can be calculated. This process is repeated for each variable. The variable that contributes to the greatest reduction in entropy is selected as the first node, with a number of branches equal

to the number of values that variable can take. For each branch, a new decision node is created using the process described above, considering only that branches respective subset of the data. This process is repeated until the entropy of each branch is 0 (i.e., contains only samples from a single class).

Other machine learning techniques refined in the last few of years have focused on developing new approaches to improve the accuracy of existing models. Incremental learning and transfer learning are concepts that have been embraced by machine learning researchers as ways to integrate learning models into real-world classification problems without a robust existing dataset. *Transfer learning* describes a process by which a model is trained on a set of data collected under a specific set of conditions and evaluated based on its performance in a wider variety of new conditions [2]. This property has benefits for a machining feature recognition system, since computer aided design (CAD) files used for training are often proprietary and difficult to obtain in large quantity. The ability of a machining feature recognition model to transfer learning between different machining feature datasets is evaluated in Section 5.3. When a transfer learning system is insufficient to effectively distinguish between classes in a specific domain, additional *incremental learning* can also be applied [20]. Incremental learning refers to training an existing model (which can be the model trained on data collected under a specific set of conditions, as described above) using domain-specific training data, to augment the classifier for that specific scenario. A proposed method of incrementally learning machining features is explored in Section 5.4.

2.3 Computer Aided Design

To extract information from a CAD model to inform a feature recognition system it is necessary to understand how 3D models are stored. Much of the early work into 3D file representations, which went on to inform the design of modern parametric modelers, was collected and published by Mäntylä [21]. Early attempts at 3D modellers stored models as a list of Boolean operations of volume primitives (prisms, cylinders, etc.). This concept of *constructive solid geometry* or CSG representation is used in volume-based machining feature recognition approaches. Unfortunately, this approach presents a challenge if a user would like to change a single feature or dimension in a part that is not fully represented by one of the volume primitives. Instead, a more general representation was developed.

Boundary representation or B-Rep models describe a type of 3D model that store the surfaces bounding the solid volume to represent the geometric information. Geometry information is typically encoded as a list of vertices, connected by edges, which themselves bound the faces of a part. Faces are associated with loops of edges, which bound the

face on all sides. Many machining feature recognition systems traverse the B-Rep model representation directly, although many modellers offer higher-level interfaces as well.

Boundary representations of CAD files are useful for storing geometry information, but modern parametric modellers offer many other features that cannot be stored in the B-Rep model. Machining callouts, tolerancing and other manufacturing information require a higher-level file standard to be stored. The STEP file format was developed by the International Organization for Standardization as a generic ASCII file format that can encode geometry information, as well as a number of relevant design and manufacturing information [7]. Since their introduction, STEP files have become a de-facto standard for CAD file exchange.

Machining feature recognition systems must interface with existing file standards in order to be useful. Early work with machining feature recognition, especially several volume-based methods, mirror the CSG representation that was common for models of the time. However, most current methods in development within the last decade operate on data extracted from a B-Rep model encoded in a STEP or similar parametric file format. Certain methods, such as hint-based approaches, may incorporate the additional manufacturing information encoded in the STEP file format. However, since the quality of this data can vary dramatically for different parametric modellers and depending on the modelling approach selected by the designer, most authors of feature recognition techniques opt to design their systems to operate without that extra manufacturing information.

2.4 Summary of Work

This thesis aims to extend the work by Yeo et al. [35] to develop a machine learning-based machining feature recognition system that can operate on B-Rep models of machined parts encoded in a STEP file format. In extending this existing work, I hope to improve the training time and/or accuracy of the system, to develop a technique that is scalable in a variety of manufacturing domains. To investigate the scalability of the system, an exploration of the ability of the classification system to transfer learning between datasets, as well as the rate at which the system can learn when adapting to a new sample domain will also be conducted.

Chapter 3

Model Creation

A typical CNC workflow involves many steps, with varying amounts of human involvement and automation. Figure 3.1 describes the steps typically required to take a part from conceptual design through manufacturing using a CNC machine. To begin, a **part is designed** based on the criteria and constraints of the design problem. Some work has been completed in the field of automated design [19], but much of the design work completed in industry is still completed by humans. Next, a machine operator or other skilled technician uses the geometry information or other manufacturing requirements defined by the designer to make informed decisions about the **machining operations** required to make the part. Many years of work have been conducted to develop techniques to automate this process, but much of this work has yet to see widespread adoption in industry. Next, typically using a computer aided manufacturing (CAM) program, the operator decides how best to implement those machining operations. Decisions such as the number of roughing/finishing passes, what tools to use, what curve or curves the tool should follow, how deep/how fast the tool should plunge into the material, and whether the selected configuration will lead to part gouging must all be considered. This process is known as **toolpath planning** and is already significantly automated; there exist accurate empirical and physical models [28] that automated software systems can use to create accurate, if conservative toolpaths. Once the toolpath is created, an available CNC machine is selected using a process known as **machining sequence planning**. Once this is complete, **stock is loaded** into the machine. There exist technologies to automate these processes (such as those developed by the company Probot Systems), but these technologies become valuable only for large volume production runs, since it takes a small amount of time to load a single piece of stock. Once ready, the machining cycle starts. This process of **executing the movements** encoded in the previous steps is fully automated in virtually all cases and is the defining

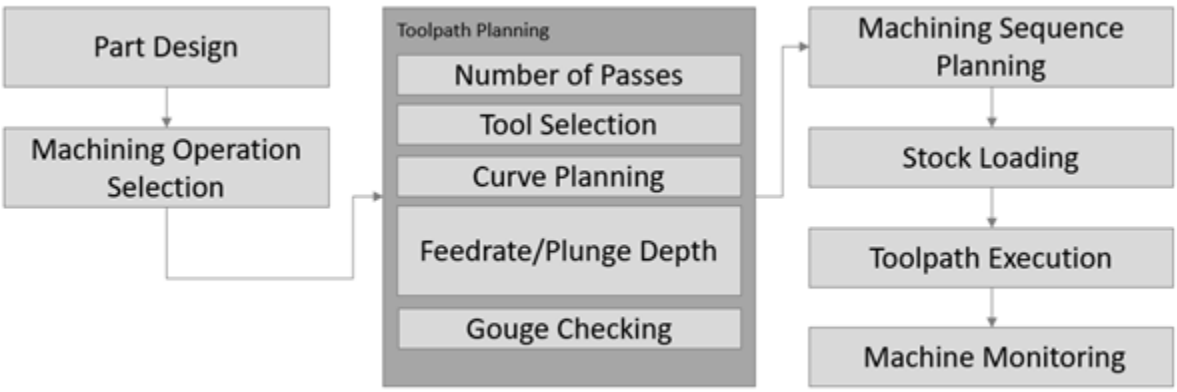


Figure 3.1: A typical CNC workflow

feature of a CNC machine. Finally, the system must be **monitored** until the part is completed, in case any unforeseen problems arise. There is a wealth of research in the machine monitoring domain [13], but more work must be completed before automated machine monitoring becomes common in industry.

As compared to other steps in a typical CNC workflow, there exists a great divide between the typical level of automation for solutions to the machining operation selection problem and the most common solutions used to accomplish this task in industry. To bridge the gap between existing machining feature recognition research and industry needs, this thesis will investigate the effectiveness of an existing machining feature recognition when classifying data collected in the real world. This work will begin with collecting two datasets: a dataset of generic CAD files which will be used to train a generic machining feature classifier, and a dataset of CAD files collected from real-world machine shops. These datasets will be tagged and encoded using an approach developed by Yeo et al. [35] to create dataset of feature vectors which can be used to train a deep learning system.

3.1 Dataset Creation

One of the challenges encountered by Yeo et al. [36] is a challenge faced by many researchers developing deep neural networks. In most cases, it is difficult to determine if a dataset size is sufficient to train and validate a learning approach. Yeo et al. note that, “It is not easy to

prove that the data for training is sufficient". To address the issue of data sufficiency, Yeo et al. used an automated data generation technique to generate as much training data as possible, to maximize the likelihood that the dataset is fully representative of all possible CAD files and their constituent machining features. This approach involved generating part parameters (feature type, location, size, etc.) algorithmically, using those parameters to automatically generate 3D models in a parametric CAD program, and tagging those 3D models using their feature descriptor approach. Generating the data automatically has the benefit of being able to generate a dramatically large number of parts (in Yeo et al.'s case, 170 000 unique CAD files), but poses a challenge when attempting to transfer the learned features to real-world CAD models.

Designers of machine learning systems must make decisions about the structure and composition of the dataset those learning systems are trained with. In the case of Yeo et al.'s machining feature recognition system, the researchers made decisions about the distribution of features, the frequency and quality of feature-feature interactions, and the pipeline of modelling software packages and CAD file interchange formats used during the creation of the dataset. This last aspect is of particular importance and can pose a challenge when attempting to develop a generalized model. A volumetrically identical part can be encoded in many ways; a feature as simple as a hole can be represented as a single cylindrical surface, or as two cylindrical surfaces, or as a non-uniform rational b-spline (NURBS) surface, or as a circle lofted along a line, or as a line rotated around a circle, or with many other representations of arbitrary surfaces. The resulting encoding scheme changes based on the parametric modelling operations used to create the hole, the parametric modelling program, and the filetype the 3D model is saved under. The specific choices made during the generation of a synthetic CAD file dataset influence the structure of the feature encodings, which in turn limit the generalization of the machining feature identification system. A system trained on synthetic data will, by the nature of the dataset it is trained upon, be most effective at identifying synthetic data generated in the same manner as the dataset. To identify machining features in real-world CAD files, a system must be trained on files used in the real world.

To create a real-world CAD dataset, the most straightforward approach would be to collect data directly from a wide variety of manufacturing companies. This approach poses multiple challenges. Companies, especially those that specialize in low-volume, short lead time manufacturing, are often unable to dedicate time to manually sort and collect CAD files if they do not already have a method of doing so automatically. Furthermore, since the part designs are typically the intellectual property of the customer, care must be taken to receive permission to use the files as part of a dataset, or else risk exposing the manufacturer or researcher to legal liability. Because of both of these factors, an exploration of existing

online CAD model databases was conducted to prepare an initial dataset.

Multiple CAD model research datasets were explored. Lee et al. collected data from existing datasets, as well as generated new data, and published their dataset along with their methodology [15]. Unfortunately, this dataset was unacceptable for this use case, since the models the dataset contained often included self-intersecting faces or unenclosed volumes where the faces did not meet correctly. Since many of the encoding methods to be used in this work rely on face-face adjacency, the data yielded by encoding these parts would not allow the network to generalize. Another dataset that was explored is called the "Mechanical Components Benchmark" [10]. This dataset was designed specifically for training systems to identify different properties of machined parts. Unfortunately, the parts contained in this dataset were encoded as volumes enclosed by a triangulated surface. Consequently, information about non-planar face types (cylindrical, toroidal, etc.) would be lost. Further, a single "base face" as defined by Yeo et al. are often subdivided into many small triangular faces in the Mechanical Components Benchmark, which would make extracting face-face adjacency information impossible. For both of these reasons, this dataset was rejected as well.

Other online CAD model repositories were explored, as an alternative to a formal research dataset. GrabCAD is a company that offers a variety of software products for the manufacturing industry. One of these products, GrabCAD Community, is an online CAD model sharing platform. Thingverse is a similar online database, specifically targeting the hobbyist 3D printing community. Thangs is yet another database that serves a similar role. All of these platforms contain parts that could be downloaded to create a dataset. Unfortunately, many of the same issues found in the research datasets can also be found in these online CAD model repositories. Since these platforms are designed for the 3D-printing community, many files are encoded as triangulated surfaces. Further, since these platforms are primarily targeted at hobbyists, the intellectual property (IP) rightsholders of the CAD files are often not specified or incorrectly labelled. For both of these reasons, hobbyist CAD model repositories were rejected.

Several companies provide CAD files of physical parts they offer in those company's digital catalogues. A notable example of a company that offers CAD files in this way is McMaster Carr. Unfortunately, McMaster Carr retains ownership of the IP rights of the CAD files and restricts their usage, which poses the same problem as many of the other datasets that were explored. TraceParts offers a similar service (hosting CAD files of physical parts available for purchase) for a variety of third party companies, including ABB Robotics, Fanuc and Rockwell Automation. Although TraceParts does not provide any IP rights of the CAD files to the user, they have partnered with research organizations in the past [16] to offer their dataset. Unfortunately, since many of the models provided on Tra-

ceParts are part of larger assemblies, including many parts which are cast or manufactured using techniques other than machining, the time required to parse and sanitize the data would exceed the scope of this work. The ABC Dataset ("A Big CAD Model Dataset") is a collection of 3D models from the online CAD platform OnShape which were collected and published by Koch et al. [12]. The models contained in this dataset are published in a variety of formats including .step, which encode the necessary parametric geometry information for extracting machining features. Although the IP rights of the CAD files are not provided to researchers, the dataset can be used for machine learning applications, provided the files are not published. For both of these reasons, this dataset was selected. CAD files from the ABC Dataset were downloaded, of which 600 were incorporated into an initial generic dataset.

The ABC Dataset offers CAD files in chunks of 10000 models, in either specific file formats or encoded in every available file format. Since this work will focus on the extraction of information from parametric models, the .step file models were downloaded. Using the method outlined in the ABC Dataset documentation, the first (0000) chunk of .step files was downloaded.

3.2 Tagging Machining Features

Yeo et al. outlined a method for extracting information about machining features based on information extracted from a "base face" and the faces surrounding that base face. To construct a dataset of information formatted in a similar manner, a system was constructed that allows a user to assign a particular machining feature to that feature's respective base face in a CAD model. Once tagged, these CAD files can be used in a subsequent encoding step to generate the feature vectors for training the model.

The files contained in the ABC dataset were originally used in a wide variety of applications and intended to be manufactured using a wide variety of manufacturing techniques. Many of the design features contained in these parts are characteristic of other manufacturing processes, such as gussets on parts that would likely be cast, threads that would likely be rolled or cut, or volumes with fine details that can only be resolved using an additive manufacturing process. As a result, the tagging system would also include the ability to reject files that are unsuitable for training.

To load the CAD files, the modelling program Solidworks was used. This software was selected due the author's familiarity with the tool, as well the robust API tools that are made available to developers looking to extend the program's functionality. The Solidworks

API can be accessed through a number of methods, including the built-in Visual Basic for Application (VBA) scripting system, using a standalone Visual C# .NET application, or using the windows Component Object Model (COM) interface. The latter COM interface was selected, for its broad compatibility with a variety of programming languages that support the Windows COM protocol. The frontend interface was written in Python, again due to the author's familiarity with the language. To communicate between the frontend application and the Solidworks modeler, the Python library pywin32 was selected for its ability to send and receive COM commands.

The Python library tkinter was used to create the frontend interface. For each machining feature defined by Yeo et al., a color was selected using the library distinctipy. This color served both as the background of that feature's respective button in the tkinter UI, as well as the color of the tagged face for that machining feature.

To tag a set of CAD files with machining features, the user follows the following process. When the data tagging script is run, the interface prompts the user for a directory that contains the CAD files to be tagged. Once the user specifies a directory, the program searches each subdirectory iteratively until a .step or native Solidworks file is found. When one is found, the file is opened in Solidworks, and the user is presented with the GUI in Figure 3.2. At this point, the user can interact with the model freely using the Solidworks interface and remove any unnecessary volumes that do not contain any useful machining feature information. Once the user identifies a machining feature, they first select the appropriate button in the tkinter GUI, then select the respective base face for that machining feature in the Solidworks model. When a face is selected in the model, the color of that face will be set using the Solidworks API based on the color defined for that feature. If multiple faces are selected consecutively, the most recently selected machining feature will be used to define the face color. If a mistake is made, or the user wishes to explicitly define a face to as a "Non Feature", the user can select the Non Feature button and click on the appropriate face in the model. This will reset the color of the face to the default grey used by Solidworks. When all machining features are selected in a part, the user can select the Next Model button if they would like to continue tagging files (which loads the next file found in the directory), or the Close button to save the current model and exit the program. In either case, the current model is saved as a Solidworks (.sldprt) file, with any face color changes the user made. If a model contains no useful machining feature data, the user can instead select the Reject button, which will delete the file from the directory and automatically load the next file.

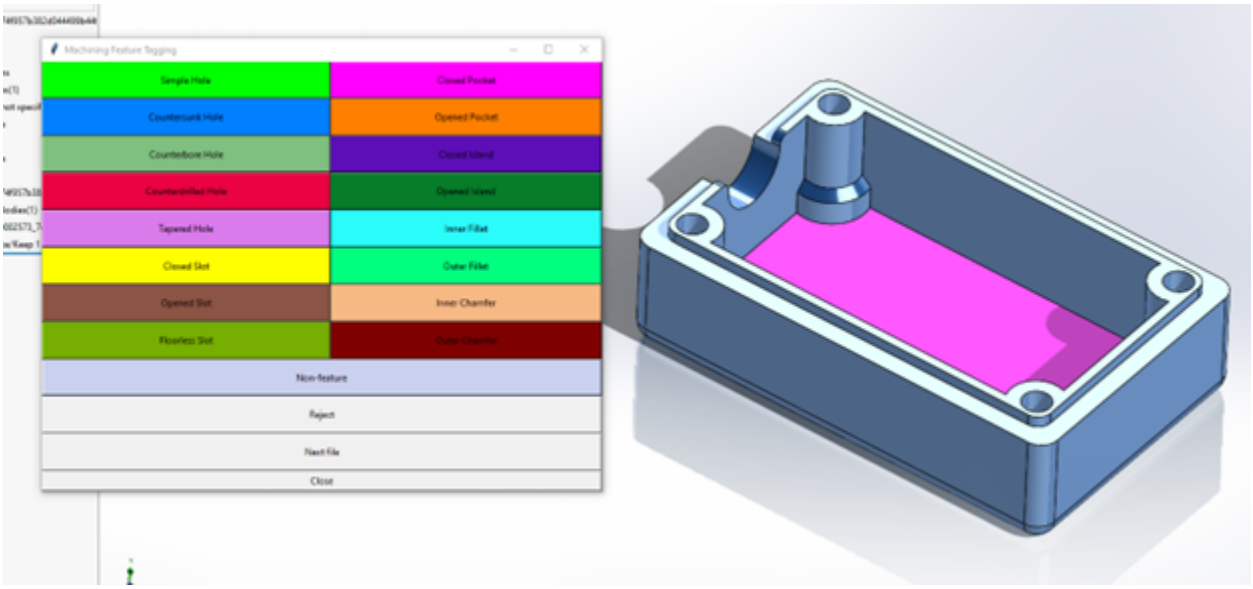


Figure 3.2: The machining feature tagging user interface

3.3 Data Collection

CAD files from the ABC dataset were tagged using the interface described in the previous section. Initially, assistance was solicited from volunteer participants with limited knowledge of machining. To inform the volunteer's machining feature selection, Figure 3.3 from Yeo et al. was provided, which defines a generic base face for each possible machining feature. Volunteers tagged 7 CAD files from the ABC dataset. To expand the training data, the author reviewed 567 CAD files from the ABC dataset and selected 70 CAD files that included CNC machining features. These files were tagged and encoded, which expanded the number of machining feature samples in the training dataset to 860.

In addition, 77 files were collected from machine shops at the University of Waterloo, Hurco Inc. and Perfecto Manufacturing Inc. The author tagged these files using the data tagging procedure outlined above. Using this procedure, a dataset containing 998 real-world machining features was produced. This dataset of machining features extracted from real-world parts remained separate from the 860 features collected from the ABC dataset. In total, 1858 machining features were included in both datasets used in this research.

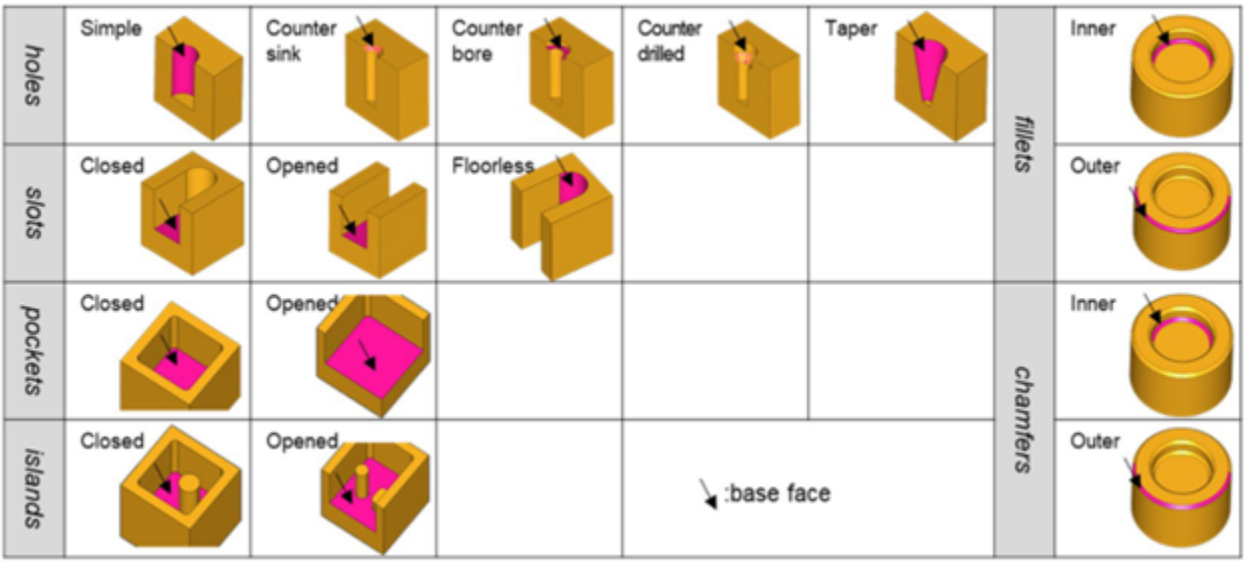


Figure 3.3: The machining feature description provided to volunteers for data tagging. Figure by Yeo et al. [36]

3.4 Feature Encoding

Once a dataset of CAD files tagged with their respective machining features was created, an encoding program was developed to extract relevant information from the parametric models based on the approach outlined by Yeo et al. The program flow of the encoding system that was created is described next.

When the encoding script is run, the user is prompted to select a directory that contains the tagged CAD files. The system iterates through that directory and all subdirectories and generates a list of CAD files to be encoded. Once all the CAD files have been located, the first file is opened in Solidworks. Using the Solidworks API, the system traverses the B-Rep model face-by-face. Each face color is compared to the color associated with each machining feature, as defined in the previous section. If the color matches a known machining feature, information about that face's geometry and the adjacent faces is collected. In total, a 61-length feature vector is created. Information about the encoding scheme is recorded in Figure 3.5. Each machining feature is assigned a unique integer value, which are defined in Figure 3.4. For a more detailed description of the encoding of each of these elements, see Yeo et al. [35]. The machining feature class, along with the 61-element feature vector, is saved to a .csv file. This process is repeated until all faces of the CAD file are checked and encoded, and then repeated for each CAD file in the directory.

a) Machining feature classes defined by Leo et al. and their integer encodings

Feature Class	Class Number
None Feature	0
Simple Hole	1
Closed Pocket	2
Countersunk Hole	3
Opened Pocket	4
Counterbore Hole	5
Closed Island	6
Counterdrilled Hole	7
Opened Island	8
Tapered Hole	9
Inner Fillet	10
Closed Slot	11
Outer Fillet	12
Opened Slot	13
Inner Chamfer	14
Floorless Slot	15
Outer Chamfer	16

b) Machining features classes and their integer encodings, after consolidation

Feature Class	Class Number
None Feature	0
Hole	1
Pocket/Island/Slot	2
Secondary Hole Feature	3
Fillet/Chamfer	4

Figure 3.4: Machining feature class encodings, a) as defined by Yeo et al. and b) after consolidation

Two elements in the feature vector, width of target face for edge-machining and width of target face for face-machining, were included by Yeo et al. to discriminate between features which are wide enough to machine using a facing operation, those wide enough to machine by simply tracing the edge of the base face, and those which were too narrow to fit a tool. Unfortunately, the ABC dataset contained files with inconsistent scales and dimensional units, making the information encoded by these target face width descriptors have limited value. Nevertheless, both width threshold feature vector elements were included in the encoding program to determine if they had any value in detecting features. The threshold for both face width feature vector elements were designed by Yeo et al. to be determined by evaluating the size of available tools. However, since there exists no specific machine for which this system will be designed, values of 2 cm for edge-machining and 5 cm for face-machining were selected as reasonable approximations of real-world values.

At this stage, there exists a list of feature vectors labelled with corresponding machining feature classes. Next, a model is developed to automatically associate these classes with the encoded feature vectors.

Feature vector element	Feature vector indices	Example value(s)	Example encodings
Base face type	1	Planar / cylindrical / toroidal	6 / 5 / 8
Curvature of base face	2	Flat / positive / negative	0 / 1 / 2
Width of base face for face-machining	3	Longer than 5 cm / shorter	1 / 2
Width of base face for edge-machining	4	Longer than 2 cm / shorter	1 / 2
Adjacent unknown / concave / convex faces in an outer loop	5-15 16-26 27-37	Proportion of outer loop adjacent faces, rounded up to the nearest 10%: • None with unknown convexity • 40% cylindrical concave • 50% planar concave • 20% planar convex	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 5 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0
Adjacent faces with C0 (non-tangent) continuity in an outer loop	38-48	Proportion of outer loop adjacent faces, rounded up to the nearest 10%: • 40% cylindrical • 50% planar • 20% faces with continuity other than C0	0 0 0 0 0 4 5 0 0 0 0
Perpendicular adjacent faces in an outer loop	49-59	Proportion of outer loop adjacent faces, rounded up to the nearest 10%: • 40% cylindrical • 50% planar • 20% faces with continuity other than perpendicular	0 0 0 0 0 4 5 0 0 0 0
Location and convexity of inner loops	60, 61	Is the inner loop not centered on the base face, and the angle between the base face and inner loop faces concave? Yes	1
		Is the inner loop centered, and the angle between the base face and inner loop faces convex? No	0

Figure 3.5: Feature vector element encodings, as defined by Yeo et al. [35]

Chapter 4

An Exploration of Model Improvements

It is expected that the performance of a system trained and tested on real-world CAD files will be less effective at distinguishing between different machining features than a similar system trained and tested on a heavily curated synthetic dataset. A curated dataset can contain examples of machining features in equal number, to eliminate the negative impacts of class imbalance when training a machine learning system. A real-world machining feature dataset, in contrast, will contain more examples of machining features that occur more frequently in machined parts. Real world datasets often includes fillets, chamfers, and holes more frequently than specific pocketing or slotting operations. When a deep learning system is trained on an unequal quantity of samples from different classes, there is a risk that instead of generalizing, the system will simply select the most common class (since that estimate is likely to be correct, by the nature of the dataset distribution).

Even if the distribution of machining features in the real-world dataset could be controlled, other concerns with class imbalance must still be addressed. Certain machining features such as holes, when encoded, are represented by a small set of unique feature vectors. In contrast, other features such as slots or pockets have much more variance, and so are represented by a greater number of unique feature vectors. When duplicate samples are removed prior to training (as is best practice, to maximize the likelihood the system can generalize), the machining features with more variance will have fewer duplicate samples, and so will end up over-represented in the final dataset. To address concerns about unequal distribution of machining features in the CAD file dataset, as well as the resulting imbalance resulting from removing duplicate entries, data pre-processing techniques may be necessary to improve the quality of the dataset. Two techniques are proposed.

Decision trees are a classical pattern recognition technique used in addition to machine learning for distinguishing between classes. A large class imbalance in the unmodified dataset may contribute to overtraining, especially when one or two classes represent a large fraction of the total dataset. The author hypothesizes that a deep learning system would perform better and be able to identify less common classes if a decision tree is used to identify the most common machining features first. By taking this approach, classes with lower variance features and significantly greater representation in the dataset can be identified with a simpler recognition system, leaving the more computationally expensive deep learning system to distinguish between the remaining classes.

Another technique used in machine learning that can be applied to this problem of class imbalance is a genetic algorithm. Genetic algorithms are not typically used to identify patterns directly, but instead are used to optimize existing systems. However, one step in the genetic algorithm procedure, crossover, can be modified to augment the imbalanced dataset.

In genetic algorithms, crossover is used to randomly recombine parent individuals in a population to create children samples with properties of each parent. In the context of reducing class imbalance, crossover will be used to generate synthetic training data using the existing training data for each class, until all classes have an equal number of training samples. Since the complexity of the generated data will be proportional to the complexity of the existing ground-truth data for each class, this method will not be as effective in addressing issues with class imbalance once duplicate samples are removed. However, the author hypothesizes this approach will be successful in mitigating the problems arising from the underlying feature imbalance in the CAD file dataset. As such, crossover data generation will also be implemented, and its impact on classification accuracy evaluated.

To implement crossover, the distribution of each feature vector element is evaluated independently for each class. For example, within the set of samples labelled as fillet, most base faces are likely cylindrical, with the remaining faces recorded to be toroidal. To generate a new training example for the fillet class, a new feature vector is created, and a value for the base face vector element is selected randomly based on the distribution of existing data. In this case, this element will most likely be selected to be cylindrical (5), but it may also be populated as toroidal (8). Elements 5-15, 16-26, 27-37, 38-48 and 49-59 as described in Figure 3.5 represent the proportion of each type of face (planar, toroidal, conical, etc.) with some property, and therefore only make sense when considered together. These sections of the feature vector are considered to be five distinct "superelements" for the purposes of data generation. Data generation was implemented by concatenating each run of ten elements into a single 10 digit integer, evaluating the distribution of those integers within a specific class, generating a new "superelement", separating the integer

back into individual elements, and appending those elements to the feature vector.

The pre-processing data pipeline is constructed as follows. First, an ID3 decision tree is trained on the available training data. The decision tree classifies features up to a certain depth, which is specified in the test plan. Next, crossover data generation is applied to each feature class except for the class with the greatest number of samples. The data generation approach discussed previously is applied until the number of training elements in each class is equal. Next, duplicate samples in each class are removed. Finally, the remaining training samples are used to train a deep learning classifier.

A fully connected feed-forward neural network was constructed, which is illustrated in Figure 4.1. The network parameters are selected to match the conditions outlined by Yeo et al. In particular, the number of nodes in each layer are selected to match the number of nodes determined by Yeo et al. to yield the greatest classification accuracy. A 61-deep input layer is connected to 5 hidden layers, which are connected to an output layer. The output layer encodes the class estimates with one-hot encoding. Each hidden layer has a ReLU activation function. The output layer weights are determined by a Softmax activation function, to estimate the probability of each potential classes given the encoded input. The standard Adam optimizer algorithm was used to update the model weights.

Some changes were also made to the network hyperparameters outlined by Yeo et al. To minimize the likelihood of overtraining, the training batch size was reduced from 8 to 1. The system was trained with 0% dropout, as originally selected by Yeo et al., and compared with models trained with 10% and 20% dropout, to determine if dropout can be effective in reducing overtraining. Some machining features (such as tapered holes) had limited representation in the training dataset generated from CAD files in the ABC dataset, and as such would be inadequately classified by the learning system. To evaluate the relative performance of several potential model improvements, the 17 machining features proposed by Yeo et al. were consolidated into 5 generic machining features. The consolidated machining features used in the remainder of this research are presented in Figure 4.2. To accommodate this change in the number of classes, the output layer depth of the machine learning network was reduced from 17 to 5.

In all models presented in the remainder of this thesis, a learning rate of $1 \cdot 10^{-4}$ was selected.

The research plan is outlined in Figure 4.3. In Part 1, the effect of incorporating dropout on classification performance was evaluated by comparing a system trained with 0%, 10% and 20% dropout. In Part 2, the effect of incorporating crossover data generation and ID3 tree classification on deep learning network classification accuracy was evaluated. For models trained in Part 1 and Part 2, only training data created by tagging files in the ABC

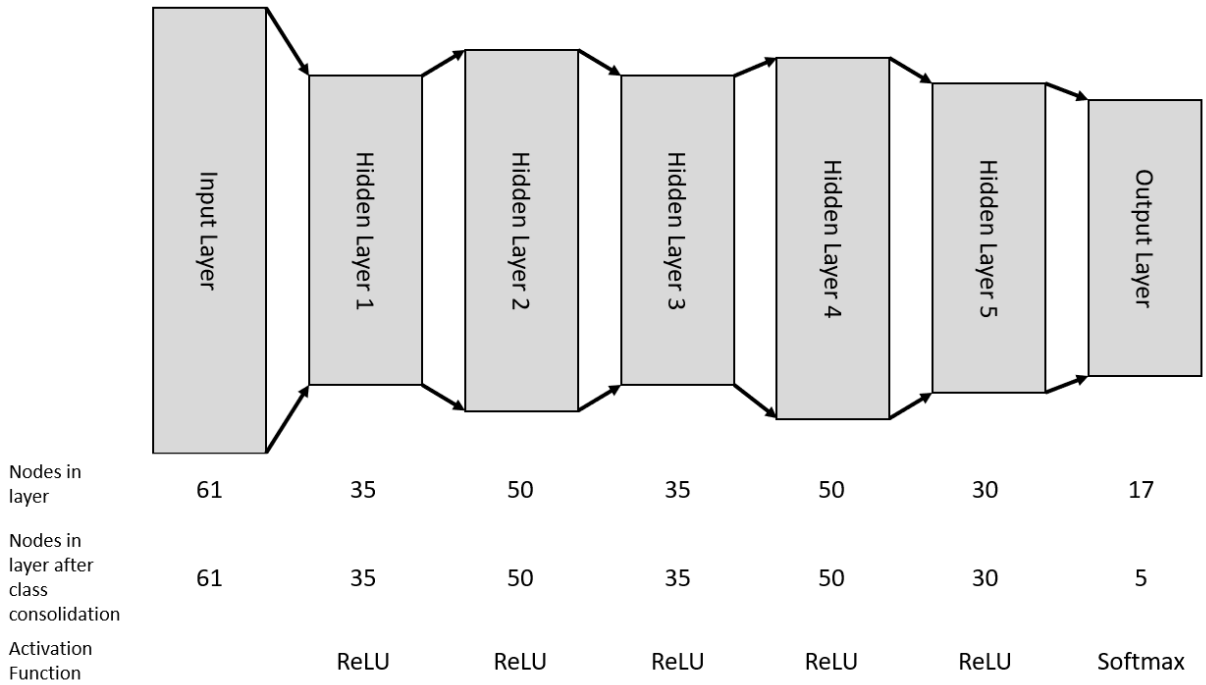


Figure 4.1: Deep learning network architecture

None Feature	All other faces that are not a feature base face						
Hole	Simple Hole						
Pocket/ Island/Slot	Closed Pocket	Opened Pocket	Closed Slot	Opened Slot	Floorless Slot	Closed Island	Opened Island
Secondary Hole Feature	Counter-bore	Counter-drilled	Taper				
Fillet/ Chamfer	Inner Fillet	Outer Fillet	Inner Chamfer	Outer Chamfer			

Figure 4.2: Consolidated machining features and their corresponding machining features as defined by Yeo et al.

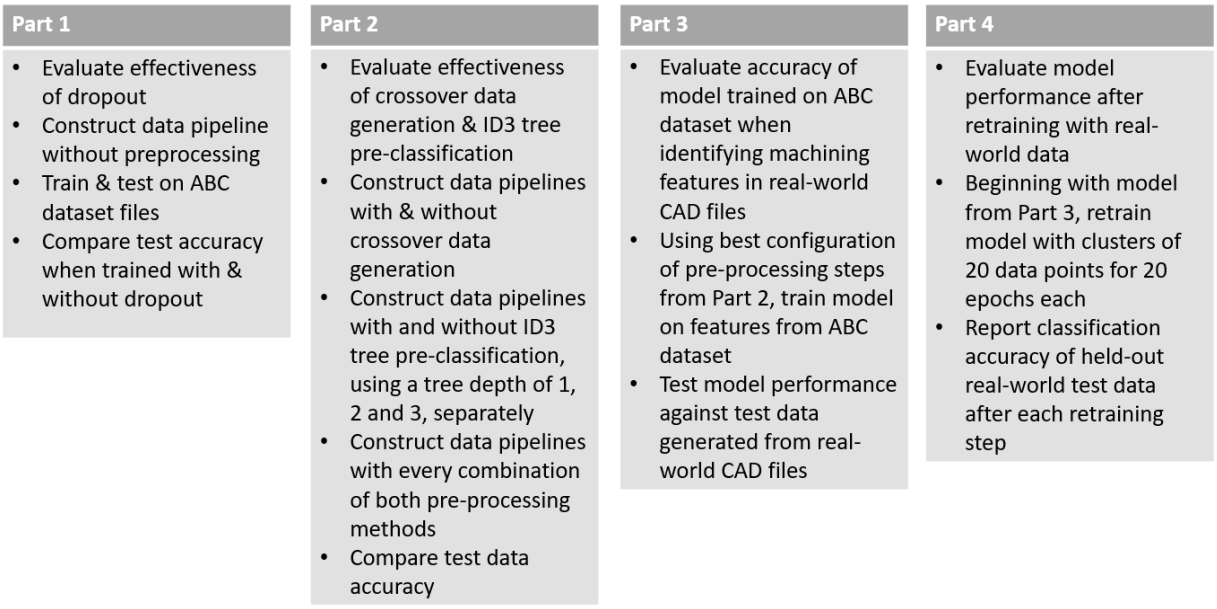


Figure 4.3: Outline of tests to be conducted

dataset was used. Additionally, in both cases, 20% of the dataset was randomly selected and held out for testing. The remaining 80% of the dataset was randomly segmented into 5 folds. For each test in Part 1 and Part 2, a system is trained using data from 4 of the 5 folds, incorporating the specified hyperparameters and pre-processing techniques for that test. Once the system is trained, the held out data from the remaining fold is used to validate the performance of the system. The accuracy of the system, in terms of the proportion of classes identified correctly in the held out validation fold, is recorded. This procedure is repeated 5 times, holding out a different fold for validation each time, and training on the remaining 4. The average of the 5 accuracy values is computed to determine a cross validation accuracy for that test.

To evaluate the effectiveness of dropout, a deep learning network with the parameters outlined above and without any pre-processing steps was trained on data from the ABC dataset, with 20% test data held out. Three tests were conducted, one using a network trained with 10% dropout on all hidden layers, one with 20% dropout, and one using a network without any dropout. The model was trained for 1000 epochs, which was selected to strike a balance between training enough to begin witnessing diminishing returns for validation accuracy, while minimizing the time to train each model. Both tests were evaluated with 5-fold cross validation, and the average accuracy for each was recorded.

The tests in Section 5.1 show that incorporating dropout improved the classification performance of the model. Consequently, 10% dropout was incorporated into all remaining models in this research. Next, the effectiveness of incorporating an ID3 decision tree pre-processing step and/or crossover data generation was determined. Once again, 5-fold cross validation was used to estimate the accuracy of each model trained on each combination pre-processing steps. Each model was once again trained for 1000 epochs. The most effective model was used to classify the 10% held out test data, to determine an unbiased estimate of the classifier's performance.

Next, in Part 3, the real-world data classification accuracy of a model trained on CAD files from the ABC dataset is evaluated. First, a model is created using the most effective combination of pre-processing steps and trained on files from the ABC dataset. Testing and validation sets are no longer separated from the ABC dataset, since that dataset will no longer be used to estimate system performance. Instead, the system is trained on all tagged files from the ABC dataset for 1000 epochs. The trained system is used to classify 998 samples, collected from tagging real-world CAD files.

Finally, in Part 4, the effect of retraining the model from Part 2 on real-world data is evaluated. First, as in Part 2, a model is created using the pre-processing steps determined in Part 1 and trained with training samples from the ABC dataset. Next, 2698 samples collected from tagging real-world CAD files are split into training and testing datasets with a ratio of 80:20. The training dataset is further segmented into clusters of 20 training samples. This number was selected to approximate 1 CAD file's worth of "training data"; i.e. a single CAD file used in the real world contains about 20 machining features, which could be used to retrain a machining feature recognition system. The pretrained network is further trained on this data, in groups of 20 data points, for 20 epochs each. After each group of data points, the accuracy of the classifier is evaluated against the test dataset, comprised of held-out training samples collected from real world data.

Chapter 5

Observations

Several experiments were conducted to evaluate the effectiveness of proposed extensions to the machining feature recognition model developed by Yeo et al. Machining feature recognition models were trained on data collected from a generic CAD file dataset. These models were augmented with canonical machine learning techniques and pre-processing steps in an attempt to improve classification accuracy and model consistency. Next, once the most effective model improvements were identified, the ability to transfer knowledge of machining features collected from a dataset of generic CAD files to a dataset of machining features collected from real-world CAD files was evaluated. Finally, to improve cross-domain classification accuracy, an exploration of a proposed re-training approach was conducted.

5.1 Part 1: Evaluation of Dropout

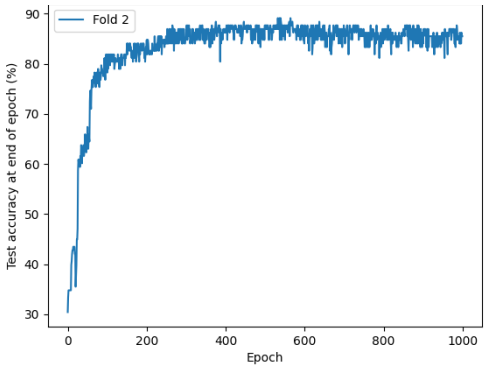
A model was trained on ABC dataset samples with 0%, 10% and 20% dropout. For each likelihood of dropout, 5-fold cross validation was used to estimate the model performance. A confusion matrix was constructed using the corresponding validation data for each fold at the end of 1000 epochs to compare the estimated and corresponding ground truth class for each sample. The validation accuracy and total cross entropy loss was calculated at the end of each epoch, and recorded. The validation accuracy and cross entropy loss during training for each fold, as well as the validation data confusion matrix after 1000 epochs of training for each fold and likelihood of dropout are included in Appendix A. A few of the figures included in the appendix are reproduced in this section for discussion. The final

validation accuracy for each fold, as well as the average validation accuracy for every fold, is reported for models trained with each configuration of dropout in Table 5.1.

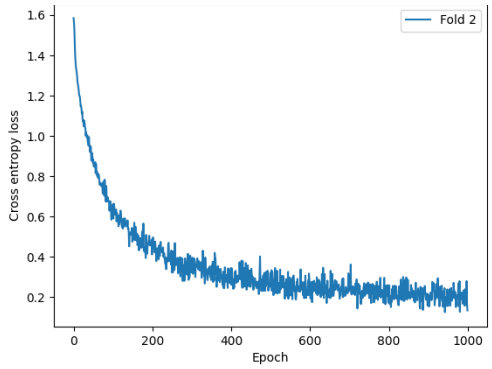
The validation accuracy across folds for a single test condition, and to a lesser extent across every test condition, exhibit the same general behaviour. This is also true of the cross entropy training loss, although the behaviour is opposite that of the validation accuracy. At the beginning of training, before the model can effectively distinguish between machining features, the model accuracy is low, indicating that a minority of validation samples were identified correctly. Consequently, the loss function returns a high loss value for each training sample, since the expected behaviour of the system is significantly different than the measured outputs. After iterating over the training samples for several hundred epochs, the model performance improves. In every test scenario, the validation accuracy increased and the cross entropy loss decreased as the model was trained. As an example of the typical results during training of single fold, see Figure 5.1. For the first 200-300 epochs, model accuracy increased dramatically, as shown in Figure 5.1a. For the same 200-300 epochs, cross entropy loss fell, as shown in Figure 5.1b. For the remainder of training, model improvement slowed. After 1000 epochs, the model performance was visualized with a confusion matrix. Once again, the trends observed in the confusion matrix figures are similar across folds for a single test condition, and to a lesser extent across every test condition. A typical confusion matrix produced after training is shown in Figure 5.1c. In about 80-90% of cases, the predicted label in the validation fold matched the ground-truth value for that sample. This is demonstrated by a prominent clustering of samples along the diagonal of the confusion matrix figures from the top left of the figure to the bottom right. Samples along this diagonal represent instances where a feature in the validation fold was identified correctly. Although there were many similarities across training instances, there were a few notable differences across different test conditions and folds.

Based on the data collected from the above tests, it was determined that incorporating 10% dropout in each hidden layer improved validation accuracy for a model trained on samples from the ABC dataset by 0.87% on average, as compared to a model trained without dropout. A model trained without dropout classified data from the held-out validation fold correctly 85.90% of the time, which improved to 86.77% when trained with 10% dropout. Increasing dropout to 20% resulted in a decrease in validation accuracy, with an average correct classification rate of 85.61%. The difference in average classification accuracy for different amounts of dropout, when considered alone, is small enough to be considered negligible. However, another qualitative observation also motivated the inclusion of dropout in future deep learning machining feature recognition systems.

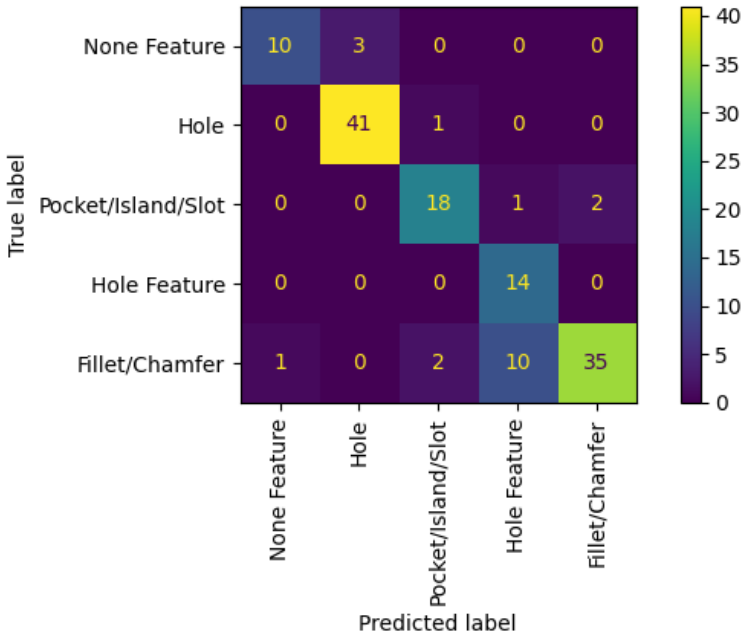
Certain folds, particularly Fold 1 and Fold 4, appear to contain a few features which were challenging for the network to identify. Although the cross entropy loss seen in Figure



(a) Validation accuracy during training for a typical fold. Fold 2, trained with 10% dropout.



(b) Training loss for a typical fold. Fold 2, trained with 10% dropout.



(c) Confusion matrix for a typical fold after training. Fold 2, trained with 10% dropout.

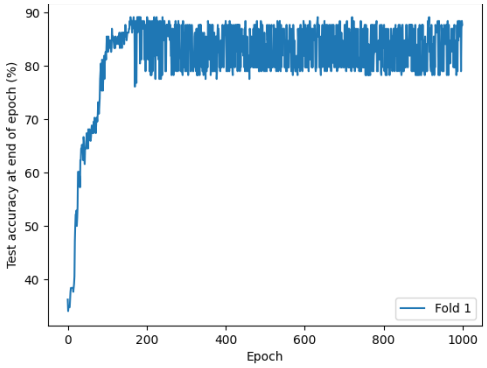
Figure 5.1: Results of training for a typical fold

Table 5.1: Summary of validation accuracy for models trained with different amounts of dropout

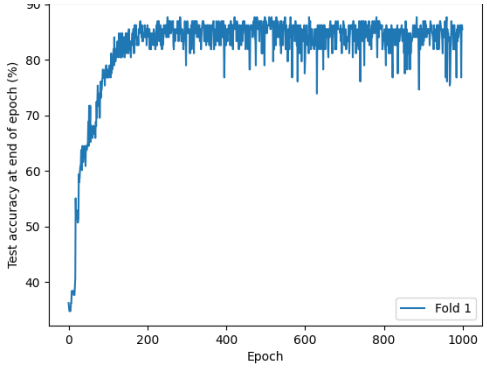
5-Fold Cross Validation Accuracy After 1000 Epochs			
Dropout	0%	10%	20%
Fold 1:	87.68%	85.51%	86.96%
Fold 2:	84.06%	85.51%	85.51%
Fold 3:	84.78%	91.30%	87.68%
Fold 4:	86.13%	82.48%	81.75%
Fold 5:	86.86%	89.05%	86.12%
Average:	85.90 %	86.77%	85.61%

5.2c remained relatively constant after 300 epochs, the corresponding validation accuracy seen in Figure 5.2a was extremely unstable at the same time during training. This is in contrast to the validation accuracy observed during training of a typical fold, such as is in Figure 5.1a, which is much more stable. After 300 epochs, the validation accuracy for Fold 1 and Fold 4 varied from approximately 75% to 88%, without stabilizing. This indicates that the model likely struggled to generalize some subset of features within the respective validation datasets, and is classifying them based on an incomplete understanding of the features. Since these features are not well understood, small changes to network weights have unexpected impacts on the classification of those features, which can be seen in the instability of the validation accuracy during training. This characteristic is undesirable; the classification accuracy of a model should improve or remain stable when trained with additional data. It can be seen in Figure 5.2b that as compared to a model trained without dropout, incorporating dropout significantly increased the classification stability of the model. Although the models trained with dropout still contain some instability, based on qualitative observations the frequency and magnitude of variation has decreased.

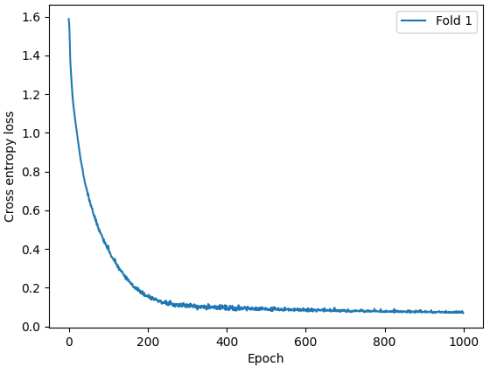
By reducing the variation of the validation accuracy, one can have more confidence that an unknown feature will be identified correctly more consistently. A model with a large amount of validation accuracy instability indicates the model may have issues generalizing classification of certain classes. For this reason, as well as for the slightly improved classification accuracy, 10% dropout was incorporated into subsequent models presented in this research.



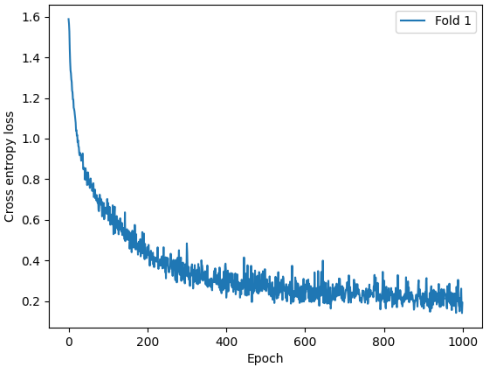
(a) Validation accuracy during training for a challenging fold. Fold 1, trained without dropout.



(b) Validation accuracy during training for a challenging fold. Fold 1, trained with 10% dropout.



(c) Training loss for a challenging fold. Fold 1, trained without dropout.



(d) Training loss for a challenging fold. Fold 1, trained with 10% dropout.

Figure 5.2: Results of training for a challenging fold

5.2 Part 2: Evaluation of ID3 Tree Pre-classification and Crossover Data Generation

ID3 tree pre-classification and crossover data generation were used to augment models trained on data tagged from the ABC dataset. The respective validation accuracy for each model was compared. Each model incorporated 10% dropout, as this was deemed most effective in Section 5.1. Six models were trained, with every combination of the following techniques: no ID3 tree, 1-deep ID3 tree and 2-deep ID3 tree pre-classification; and with and without crossover data generation. The validation accuracy during training, cross-entropy loss and final confusion matrix for each fold and each test is report in Appendix A.2.

As before, 5-fold cross validation was used to create an unbiased estimation of the validation accuracy of the models trained with each combination of techniques. The accuracy of the model in classifying samples in the held-out validation fold for each test, along with the number of samples used to train the deep neural network stage is reported in Table 5.2

Table 5.2: Summary of validation accuracy and number of deep neural network training samples for models trained with ID3 tree pre-classification and crossover data generation

5-Fold Cross Validation Accuracy After 1000 Epochs						
ID3 Tree Depth	None	1	2	None	1	2
Crossover?	N	N	N	Y	Y	Y
Fold 1 Accuracy:	85.51%	82.61%	85.51%	88.41%	84.06%	85.51%
Fold 2 Accuracy:	85.51%	83.33%	82.61%	81.88%	86.96%	80.43%
Fold 3 Accuracy:	91.30%	86.96%	88.41%	89.86%	87.68%	89.13%
Fold 4 Accuracy:	82.48%	86.86%	88.32%	86.13%	87.59%	86.86%
Fold 5 Accuracy:	89.05%	89.05%	89.05%	87.59%	86.12%	90.51%
Avg. Accuracy:	86.77%	85.76%	86.78%	86.77%	86.48%	86.49%
Fold 1 # Samples:	207	155	102	389	231	143
Fold 2 # Samples:	206	150	86	361	205	131
Fold 3 # Samples:	207	141	89	413	207	176
Fold 4 # Samples:	201	149	103	383	209	178
Fold 5 # Samples:	212	149	96	417	227	142
Avg. # Samples:	206.6	148.8	95.2	392.6	215.8	154

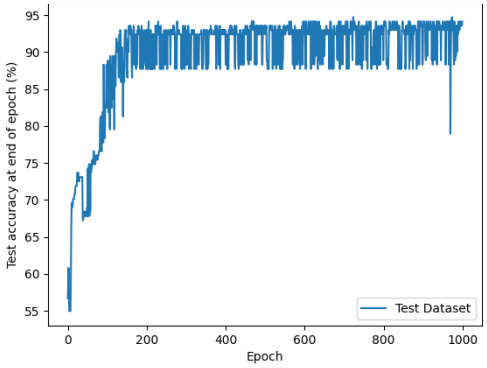
It can be seen that incorporating a 2-deep ID3 tree pre-classification step improved average classification accuracy by a negligible amount, increasing from 86.77% validation

accuracy without any data augmentation to 86.68% validation accuracy with a 2-deep ID3 tree. Incorporating crossover data generation did not improve validation accuracy.

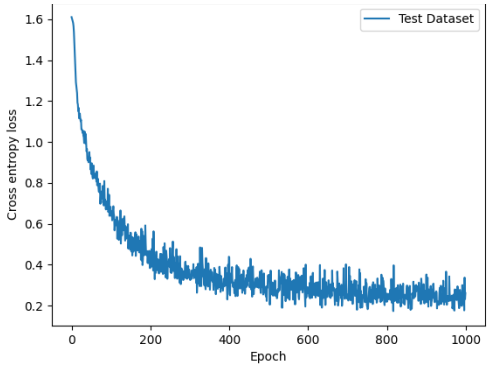
Although ID3 tree pre-classification did not significantly improve the rate of feature classification, it also did not significantly reduce classification accuracy. Moreover, by incorporating ID3 tree pre-classifcation, the amount of data required to train a neural network with a given classification accuracy was significantly reduced as compared to an equivalent model without pre-classification. By incorporating a 2-deep ID3 tree pre-classification step, the average number of training samples for the deep neural network step was reduced from 206.6 after duplicate removal to only 95.2, without loss of classification accuracy. The number of training samples required to train a neural network is reduced by selecting for common features without significant complexity and identifying them using an ID3 tree. The training samples associated with those features are then omitted from the deep neural network training dataset. By reducing the number of training samples necessary to achieve a given classification accuracy, the time required to train the model is reduced, since fewer features must be analysed for each epoch.

Crossover data generation increased the number of samples used to train the deep neural network without improving the classification accuracy. By increasing the number of samples used to train the deep neural network, the time to train the model was increased without any benefit. The selected approach of recombining feature vector elements based on the distribution of those elements in the training data associated with a single feature was ineffective in improving the generalization of the system. Although crossover data generation may be effective for other classification problems, the increased noise associated with generating synthetic data outweighed any classification accuracy benefit associated with increasing the size of the training dataset.

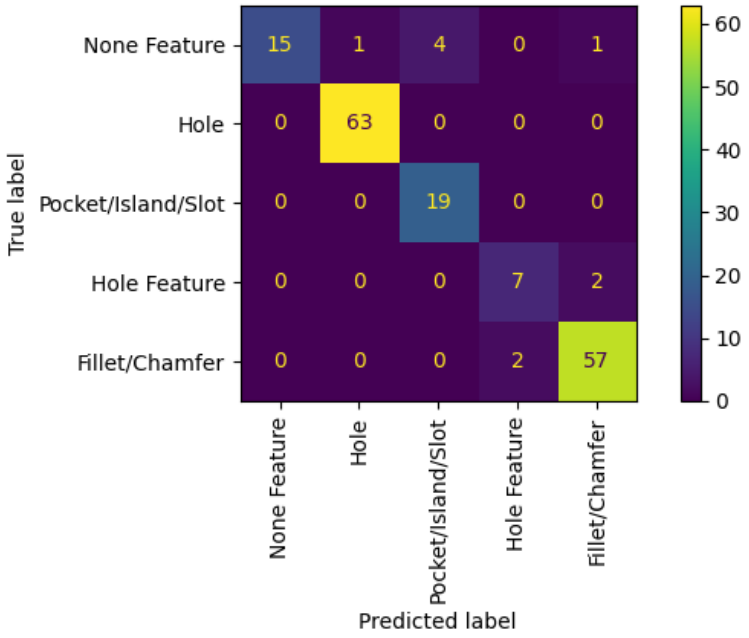
Once the most effective combination of model improvements was selected, a single model was trained using all training samples collected from the ABC dataset (i.e. training data from all 5 folds). The results of this training or presented in Figure 5.3. This model was evaluated against a dataset of held-out test data, which represented the remaining 20% of samples collected from the ABC dataset. This model, which was trained for 1000 epochs with 10% dropout and a 2-deep ID3 tree pre-classification step, was able to classify the held-out test data correctly 94% of the time.



(a) Validation accuracy during training



(b) Cross entropy loss during training



(c) Confusion matrix after training for 1000 epochs

Figure 5.3: Results of model trained on full ABC training dataset with 10% dropout and 2-deep ID3 tree

5.3 Part 3: Evaluation of Transfer Learning

It was demonstrated by Yeo et al. that their model was effective at identifying features, even without the proposed improvements developed in the previous sections. However, Yeo et al. used the same algorithm to generate synthetic training data to train their model as they used to generate the test data that was used to evaluate their model performance. As a consequence, the ability of the system to adapt to new dataset domains, including different model encoding schemes or feature distributions, was left unexplored. This section seeks to evaluate the ability of a machining feature recongition model to transfer learning between dataset domains. To accomplish this, a model was trained on all features encoded from ABC dataset models (i.e. a combination of the former training and test datasets in Part 5.2). This model was trained using the techniques which were deemed most effective in Part 5.1 and Part 5.2: 10% dropout and a 2-deep ID3 tree pre-classification step. After training for 1000 epochs, the model was evaluated against 998 features encoded from real-world machined parts. The accuracy improvement of the model in identifying parts in this real-world dataset during training is presented in Figure 5.4. The training loss of the model, as calculated by training with ABC dataset models, is presented in Appendix A.3. A confusion matrix of the model classification performance after 1000 epochs is presented in Figure 5.5.

It can be seen in Figure 5.4 that the classification accuracy of the transfer learning model improved for approximately 200 epochs, and then plateaued. For the remaining 800 epochs of training, the model performance was unstable, falling to as low as 56% accuracy and rising to as much at 66% accuracy. After 1000 epochs, the model achieved a transfer learning classification accuracy of 63.19%. From Figure 5.5, it can also be seen that no individual machining feature was identified correctly more than 70% of the time, indicating this model would not be effective as general-purpose machining feature classifier, even for simple features such as holes. This low classification performance is likely due in part to the limited training dataset that was available, resulting in a model with limited ability to generalize features. However, the test dataset may also simply contain a different distribution of features, with a slightly different set of valid encodings for each feature. Thus, the model may be able to be improved by retraining the general model trained on data from the ABC dataset using features from the real-world dataset. This is explored next.

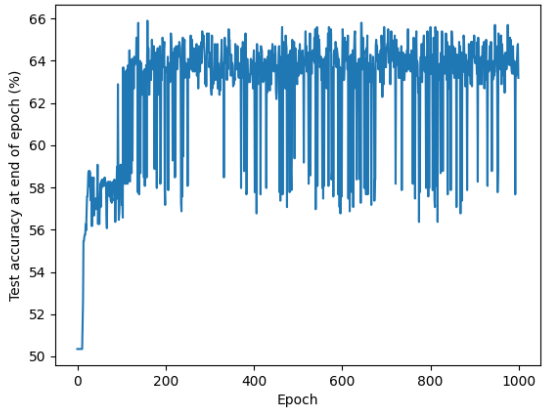


Figure 5.4: Accuracy of model trained on samples from ABC dataset, evaluated against real-world data

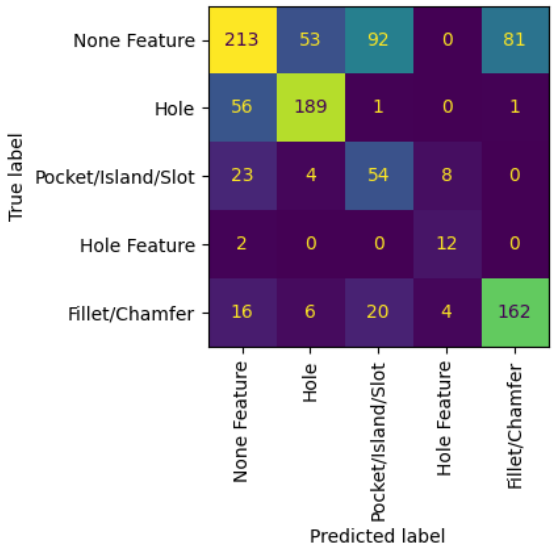


Figure 5.5: Confusion matrix of model trained on samples from ABC dataset, evaluated against real-world data

5.4 Part 4: Evaluation of Incremental Learning

To improve the performance of the model, an incremental learning approach was developed. The real-world dataset was segmented into datasets of training and test data with a ratio of 80:20. The training dataset was further segmented into clusters of 20 machining features. This number was selected as it approximates the number of machining features that are contained in a single CAD file. The base model trained on the machining features extracted from the ABC dataset was retrained, 20 features at a time, for 20 epochs each. This retraining was completed in the same manner as the initial training method, with 10% dropout. The training loss calculated against each cluster of training samples is reported in Figure 5.7. At the end of each training cluster, the test data was used to calculate the classification accuracy of the model. That test data accuracy, and how it improved during training, is reported in Figure 5.6. After training against 39 clusters of 20 data points, the model performance was characterized and reported in a confusion matrix in Figure 5.8.

The model loss does not follow a smooth asymptotic decline, at not least to the same degree as was observed during initial training (Figure A.25). The model loss does not decline smoothly since every 20 epochs, a new set of samples are used for training. As

a result, any information from the previous set of samples that the system learned that could not generalize caused a brief increase in error. This increase in error caused the model to improve the model's classification performance, which caused the error to reduce. The spikes in error every 20 epochs reduced in magnitude somewhat after 100 epochs of training, but continued to occur as the system encountered unfamiliar features. At the same time, the test data accuracy generally trended upwards, increasing from below 66% classification accuracy without any retraining to 77.39% after training against 39 clusters of data points.

The corresponding confusion matrix generated by evaluating the test data after training was equally positive. Holes, which were previously regularly confused with the non-feature class, are consistently identified correctly. After retraining, 89% of hole predictions were correct, with only a single true hole predicted as another class. This level of accuracy is approaching a threshold where an automatic feature recognition system could provide value to a human operator.

It is promising to see such a notable increase in model performance by retraining on less than 800 machining features. Given access to more real-world training data, it is reasonable to expect real-world classification accuracy to approach that of a system trained and tested on entirely synthetic feature data.

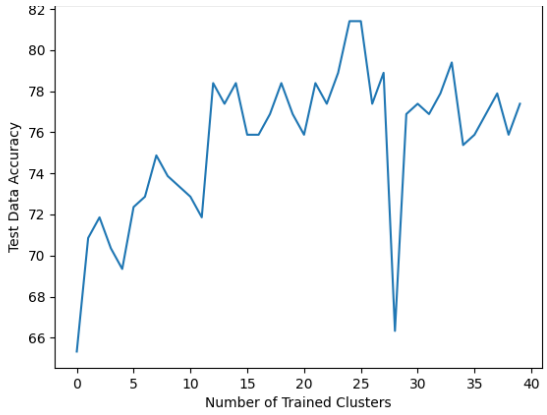


Figure 5.6: Accuracy of transfer learning model after re-training with clusters of 20 machining features

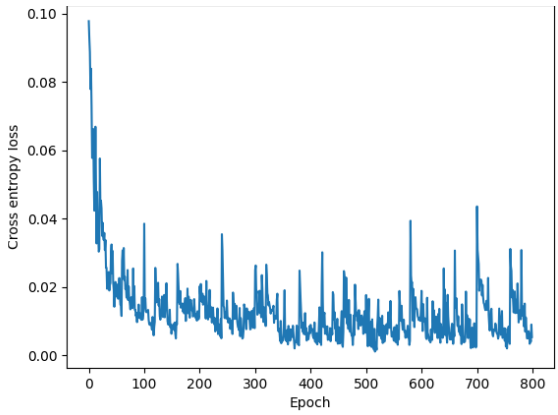


Figure 5.7: Loss during re-training of transfer learning model

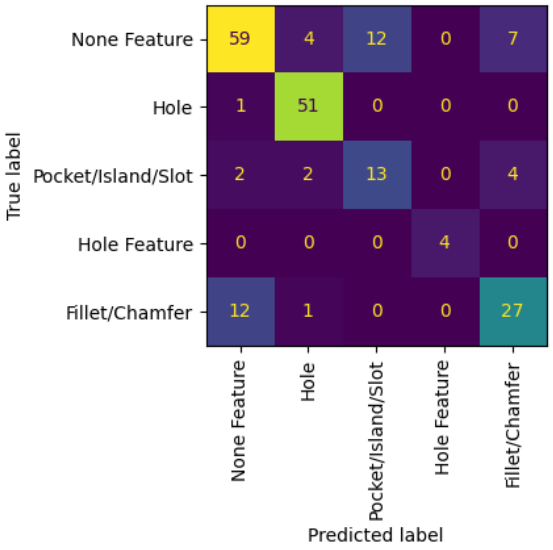


Figure 5.8: Confusion matrix of model evaluated against real-world data after re-training

Chapter 6

Conclusions

A machining feature recognition system was developed in this thesis by extending a machine learning approach originally developed by Yeo et al. The extensions proposed in this work were effective at improving the consistency and scalability of the classifier developed by Yeo et al. More work needs to be done to continue improving the classification accuracy of machining feature recognition systems.

Three extensions to the system developed by Yeo et al. were evaluated in this work: the incorporation of dropout, the introduction of an ID3 tree pre-classification step, and the inclusion of additional training data using crossover data generation. Dropout was determined to improve the consistency of feature classification. Yeo et al. determined there was no benefit of incorporating dropout when training their model on synthetically generated machining feature data. In contrast, this work found evidence that the consistency of classification accuracy during training improved when 10% dropout was incorporated in models trained on real-world data. In addition, incorporating an ID3 tree pre-classification step before training a machine learning classifier was effective at reducing model training time, without reducing classification accuracy. Crossover data generation was deemed to not have any significant benefits for model classification accuracy or scalability, and so was rejected.

The augmented feature recognition model was trained on generic CAD files, and used to classify machining features from real-world CAD files. Without any additional training, the augmented classifier was unable to identify machining features consistently. The classifier was re-trained using machining features collected from real-world CAD files. The re-trained classifier was significantly more effective at identifying machining features.

Re-training existing machining feature recognition models has significant future po-

tential. A simple machining feature recognition model can be developed using a limited dataset of machining features, and re-trained based on the actions of a machinist selecting features in a CAM program. Collecting data in this way has several benefits. First, the concerns associated with training a model on CAD files with intellectual property protections can be mitigated by re-training the feature recognition system locally. Second, re-training can be done on-the-fly, without interrupting the work of a machinist. Finally, re-training can be used to tailor a feature recognition model to a specific workflow or industry, incorporating automatic feature detection into an existing workflow only when the historical classification accuracy for a particular machining feature reaches a threshold determined by the user.

Several areas must still be explored before the feature recognition system developed in this thesis can be incorporated into an industrial CNC machine workflow. The ID3 tree pre-classification step is presently calculated once when training on generic CAD files, and is not updated during re-training. An exploration of possible methods for incrementally training the ID3 tree may be valuable to improve the classification accuracy of the re-trained system. More training data must also be collected in order to identify the point of diminishing returns for classification accuracy. Once classification accuracy can no longer improve by incorporating additional re-training data, other techniques (such as re-training the ID3 tree) may become necessary. To collect more training data, a system that integrates incremental training with CAM software should be developed. Integrating incremental training in a CAM software package, as proposed earlier in this chapter, will have the effect of significantly increasing the ease of collecting training data, and serve as a proof of concept demonstration of the proposed incremental learning technique.

All code discussed in this thesis can be accessed at https://github.com/mlenover/machining-feature-recognition.

References

[1] Mahadevan Balasubramaniam. Automatic 5-axis NC toolpath generation.

[2] Stevo Bozinovski. Reminder of the First Paper on Transfer Learning in Neural Networks, 1976. *Informatica*, 44, September 2020.

[3] Iain A. Donaldson and Jonathan R. Corney. Rule-based feature recognition for 2·5D machined components. *International Journal of Computer Integrated Manufacturing*, 6(1-2):51–64, January 1993. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/09511929308944555.

[4] Paul Fieguth. *An Introduction to Pattern Recognition and Machine Learning*. Springer International Publishing, Cham, 2022.

[5] Jung Hyun Han and Aristides AG Requicha. Integration of feature based design and feature recognition. *Computer-Aided Design*, 29(5):393–403, May 1997.

[6] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, July 2012. arXiv:1207.0580 [cs].

[7] International Organization for Standardization. STEP-file, April 2016.

[8] S. Joshi and T. C. Chang. Graph-based heuristics for recognition of machined features from a 3D solid model. *Computer-Aided Design*, 20(2):58–66, March 1988.

[9] Byung Chul Kim and Duhwan Mun. Enhanced volume decomposition minimizing overlapping volumes for the recognition of design features. *Journal of Mechanical Science and Technology*, 29(12):5289–5298, December 2015.

[10] Sangpil Kim, Hyung-gun Chi, Xiao Hu, Qixing Huang, and Karthik Ramani. A Large-Scale Annotated Mechanical Components Benchmark for Classification and Retrieval Tasks with Deep Neural Networks. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, Lecture Notes in Computer Science, pages 175–191, Cham, 2020. Springer International Publishing.

[11] Y. S. Kim. Recognition of form features using convex decomposition. *Computer-Aided Design*, 24(9):461–476, September 1992.

[12] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. ABC: A Big CAD Model Dataset For Geometric Deep Learning, April 2019. arXiv:1812.06216 [cs].

[13] Mustafa Kuntoğlu, Emin Salur, Munish Kumar Gupta, Murat Sarıkaya, and Danil Yu. Pimenov. A state-of-the-art review on sensors and signal processing systems in mechanical machining processes. *The International Journal of Advanced Manufacturing Technology*, 116(9):2711–2735, October 2021.

[14] L. K. Kyprianou. *Shape classification in computer-aided design*. Ph.D., University of Cambridge, 1980. Accepted: 1980.

[15] Hyunoh Lee, Jinwon Lee, Hyungki Kim, and Duhwan Mun. Dataset and method for deep learning-based reconstruction of 3D CAD models containing machining features for mechanical parts. *Journal of Computational Design and Engineering*, 9(1):114–127, February 2022.

[16] Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, Li Yi, and Leonidas Guibas. Supervised Fitting of Geometric Primitives to 3D Point Clouds. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2647–2655, June 2019. arXiv:1811.08988 [cs].

[17] Jiachen Liang, Shusheng Zhang, Bo Huang, Yajun Zhang, and Rui Huang. NC process analysis–based intersecting machining feature recognition and reuse approach. *The International Journal of Advanced Manufacturing Technology*, 123(7):2393–2413, December 2022.

[18] Xu Liu, Yingguang Li, Tianchi Deng, Pengcheng Wang, Kai Lu, Jiarui Chen, and Dingye Yang. A supervised community detection method for automatic machining region construction in structural parts NC machining. *Journal of Manufacturing Systems*, 62:367–376, January 2022.

[19] Xinjiani Long, Haitao Li, Yuefeng Du, Enrong Mao, and Jianjian Tai. A knowledge-based automated design system for mechanical products based on a general knowledge framework. *Expert Systems with Applications*, 178:114960, September 2021.

[20] Yong Luo, Liancheng Yin, Wenchao Bai, and Keming Mao. An Appraisal of Incremental Learning Methods. *Entropy*, 22(11):1190, October 2020.

[21] Martti Mäntylä. *An Introduction to Solid Modeling*. Computer Science Press, 1988. Google-Books-ID: N7BEPgAACAAJ.

[22] OpenAI. GPT-4 Technical Report, March 2023.

[23] S. Prabhakar and M. R. Henderson. Automatic form-feature recognition using neural-network-based techniques on boundary representations of solid models. *Computer-Aided Design*, 24(7):381–393, July 1992.

[24] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-Shot Text-to-Image Generation, February 2021.

[25] Hiroshi Sakurai and Chia-Wei Chin. CHAPTER 4 - Definition and Recognition of Volume Features for Process Planning. In Jami J. Shah, Martti Mäntylä, and Dana S. Nau, editors, *Manufacturing Research and Technology*, volume 20 of *Advances in Feature Based Manufacturing*, pages 65–80. Elsevier, January 1994.

[26] Jami J. Shah, Yan Shen, and Arvind Shirur. CHAPTER 7 - Determination Of Machining Volumes From Extensible Sets Of Design Features. In Jami J. Shah, Martti Mäntylä, and Dana S. Nau, editors, *Manufacturing Research and Technology*, volume 20 of *Advances in Feature Based Manufacturing*, pages 129–157. Elsevier, January 1994.

[27] Yang Shi, Zhang Yicha, Kaishu Xia, and Ramy Harik. A Critical Review of Feature Recognition Techniques. *Computer-Aided Design and Applications*, 17:861–899, January 2020.

[28] David A. Stephenson and John S. Agapiou. *Metal Cutting Theory and Practice*. CRC Press, April 2016. Google-Books-ID: 77n1CwAAQBAJ.

[29] K. Tang and T. Woo. Algorithmic aspects of alternating sum of volumes. Part 1: Data structure and difference operation. *Computer-Aided Design*, 23(5):357–366, June 1991.

[30] Tadele Belay Tuli and Andrea Cesarini. Automated Unsupervised 3D Tool-Path Generation Using Stacked 2D Image Processing Technique. *Journal of Manufacturing and Materials Processing*, 3(4):84, October 2019.

[31] H. Vafaie and K. De Jong. Genetic algorithms as a tool for restructuring feature space representations. In *Proceedings of 7th IEEE International Conference on Tools with Artificial Intelligence*, pages 8–11, November 1995. ISSN: 1082-3409.

[32] J.H. Vandenbrande and A.A.G. Requicha. Spatial reasoning for the automatic recognition of machinable features in solid models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(12):1269–1285, December 1993. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[33] A. K. Verma and Sunil Rajotia. A hint-based machining feature recognition system for 2.5D parts. *International Journal of Production Research*, 46(6):1515–1537, March 2008.

[34] Douglas L. Waco and Yong Se Kim. Geometric reasoning for machining features using convex decomposition. *Computer-Aided Design*, 26(6):477–489, June 1994.

[35] Changmo Yeo, Sanguk Cheon, and Duhwan Mun. Manufacturability evaluation of parts using descriptor-based machining feature recognition. *International Journal of Computer Integrated Manufacturing*, 34(11):1196–1222, November 2021.

[36] Changmo Yeo, Byung Chul Kim, Sanguk Cheon, Jinwon Lee, and Duhwan Mun. Machining feature recognition based on deep neural networks to support tight integration with 3D CAD systems. *Scientific Reports*, 11(1):22147, November 2021.

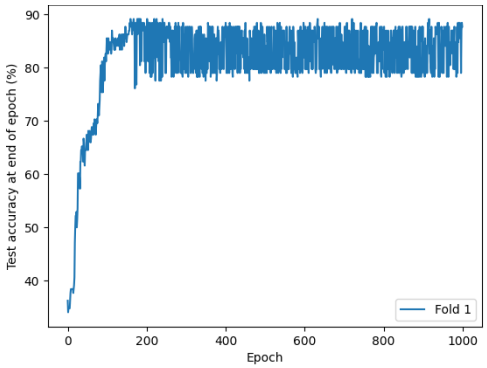
APPENDICES

Appendix A

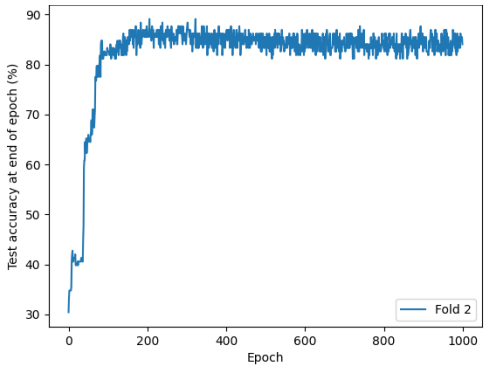
Results of Model Training

A.1 Results from Part 1: Evaluation of Dropout

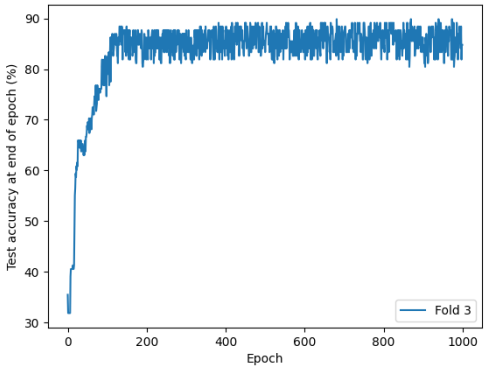
The validation accuracy during training for each fold with 0% dropout is recorded in Figure A.1 The corresponding validation accuracy for 10% and 20% dropout are recorded in Figure A.4 and A.7, respectively. The loss for each fold trained with 0% dropout is recorded in Figure A.2. The corresponding loss for 10% and 20% dropout are recorded in Figure A.5 and A.8, respectively. The confusion matrix outlining the performance of each model trained with 0%, 10% and 20% dropout are recorded in Figures A.3, A.6 and A.9 respectively.



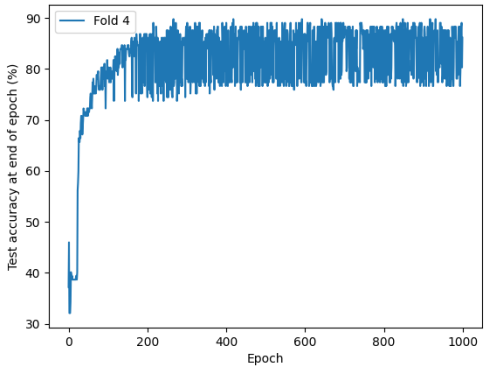
(a) Fold 1



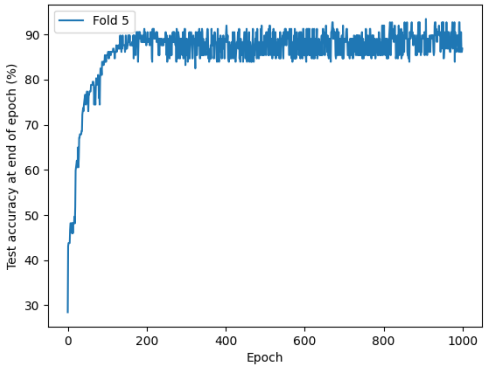
(b) Fold 2



(c) Fold 3

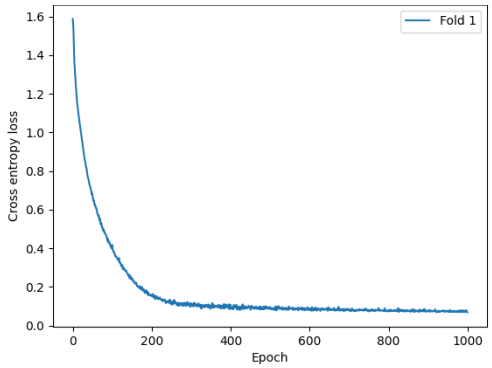


(d) Fold 4

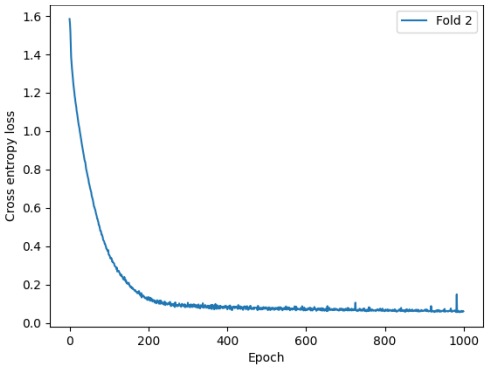


(e) Fold 5

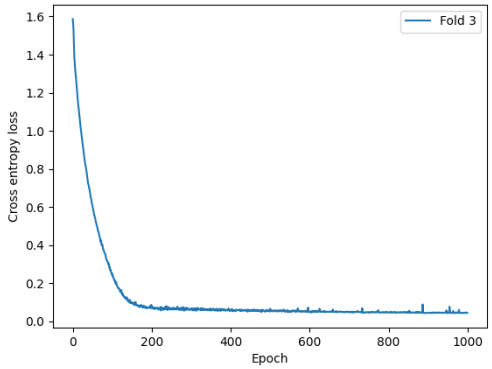
Figure A.1: Validation accuracy during training without dropout



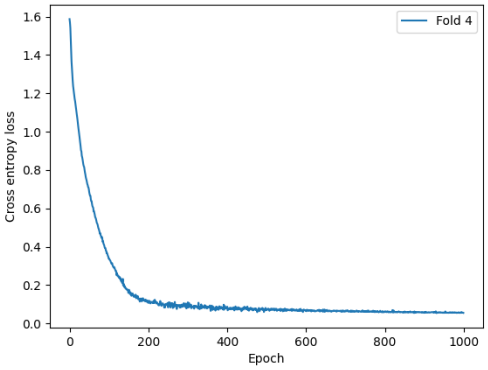
(a) Fold 1



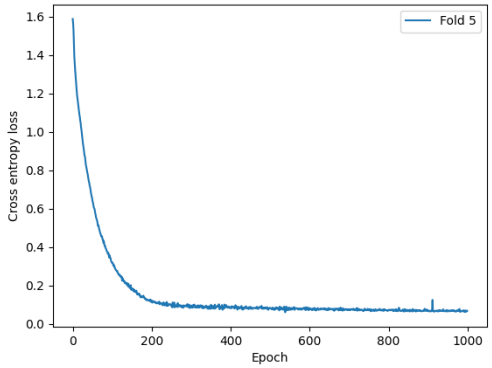
(b) Fold 2



(c) Fold 3

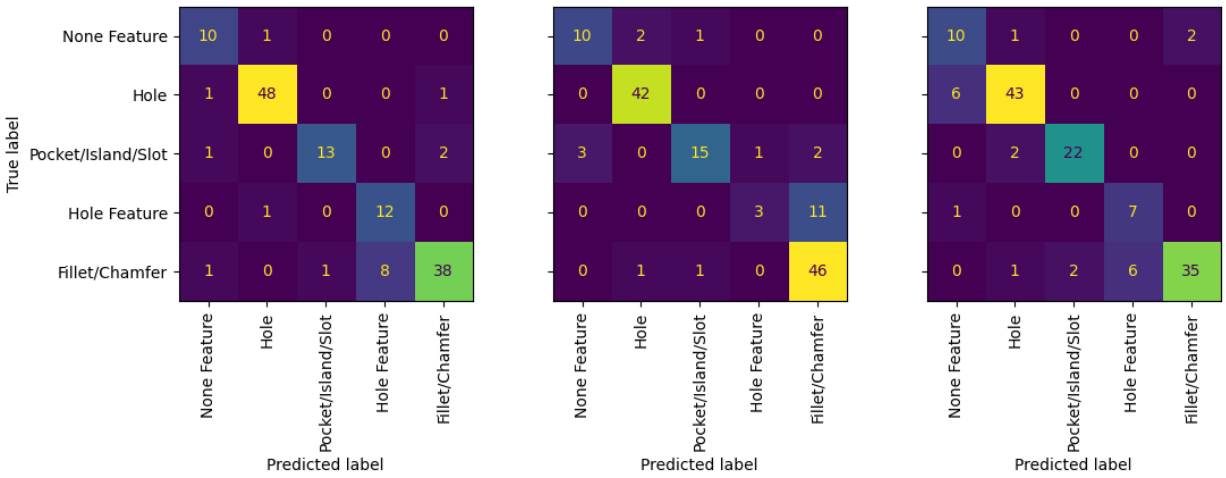


(d) Fold 4

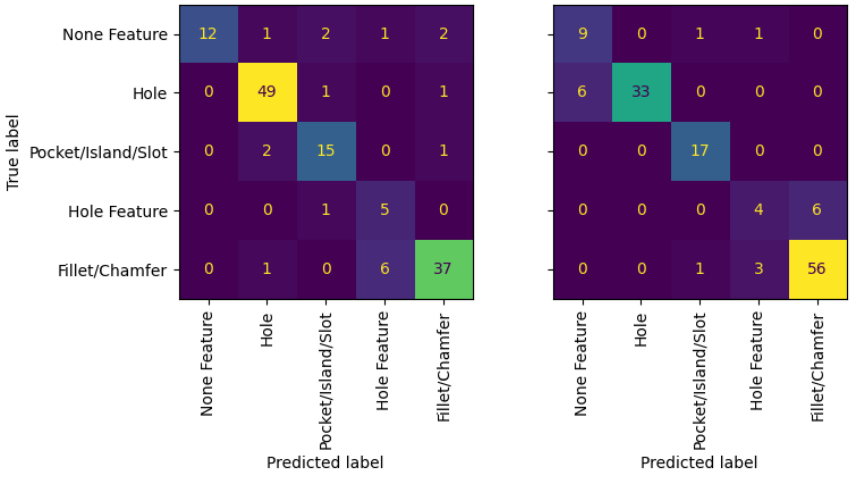


(e) Fold 5

Figure A.2: Cross entropy training loss during training without dropout

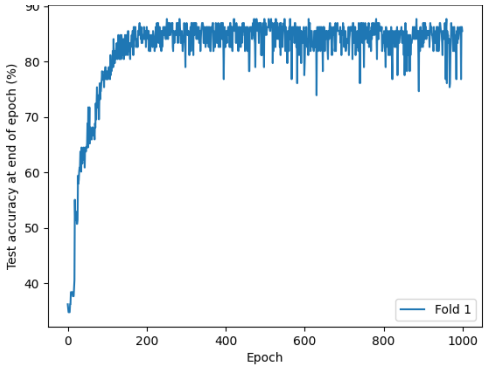


(a) Fold 1 (b) Fold 2 (c) Fold 3

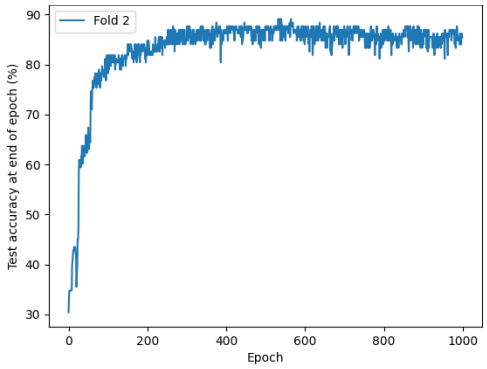


(d) Fold 4 (e) Fold 5

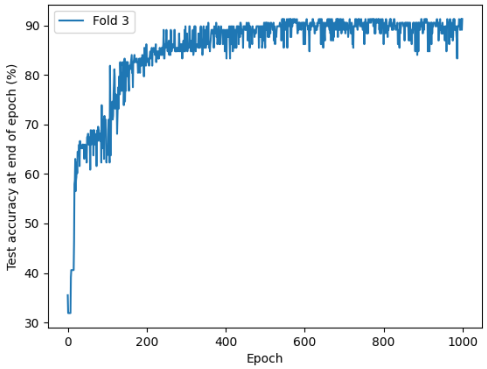
Figure A.3: Confusion matrix results after training without dropout



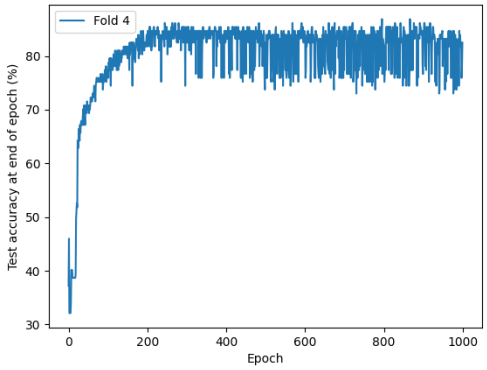
(a) Fold 1



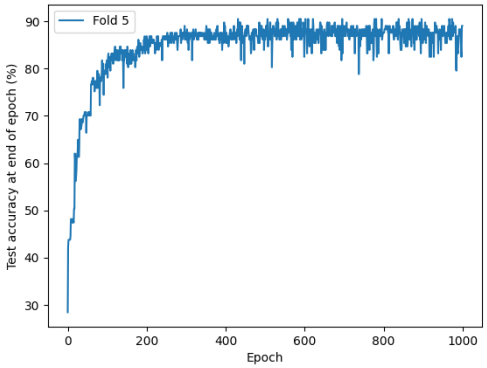
(b) Fold 2



(c) Fold 3

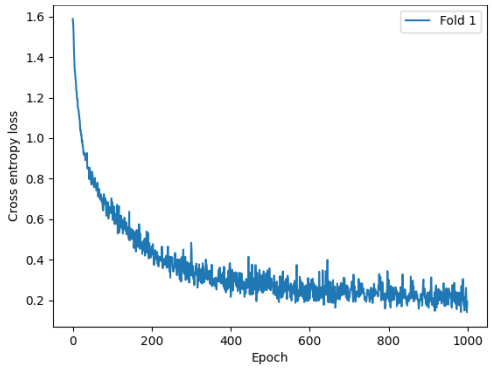


(d) Fold 4

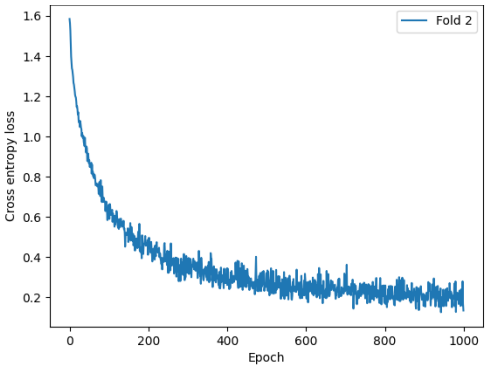


(e) Fold 5

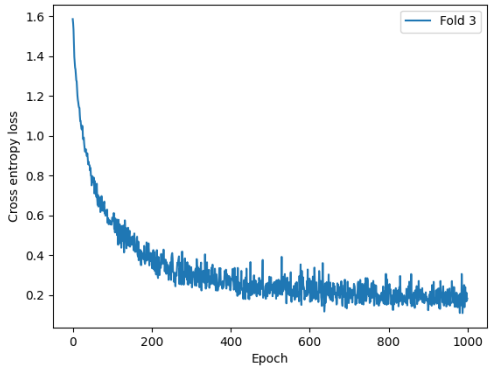
Figure A.4: Validation accuracy during training with 10% dropout



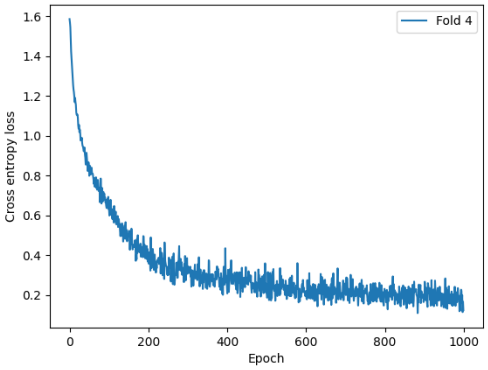
(a) Fold 1



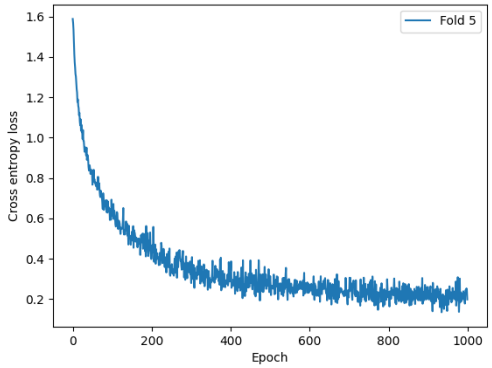
(b) Fold 2



(c) Fold 3

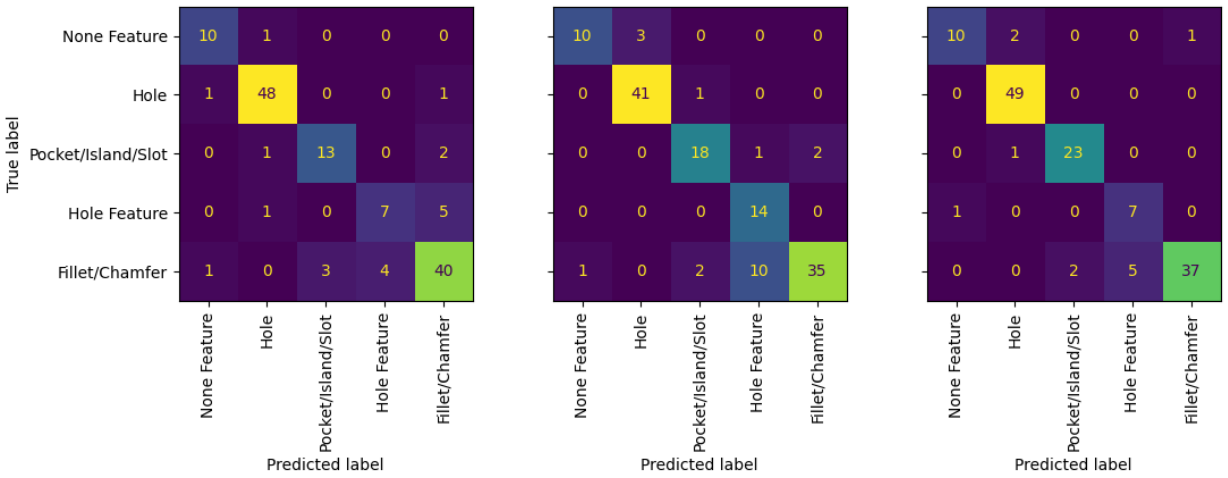


(d) Fold 4

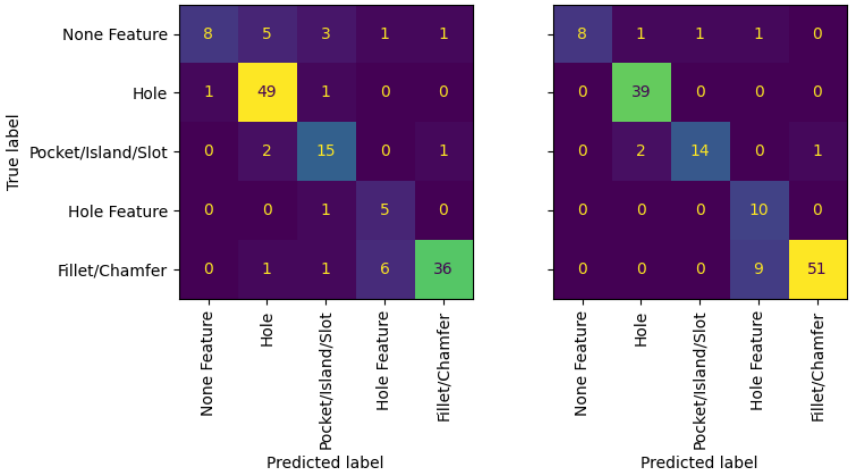


(e) Fold 5

Figure A.5: Cross entropy training loss during training with 10% dropout

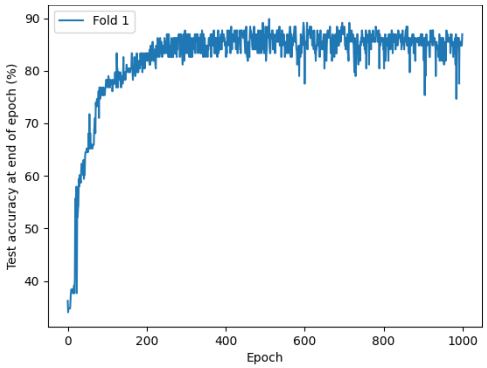


(a) Fold 1 (b) Fold 2 (c) Fold 3

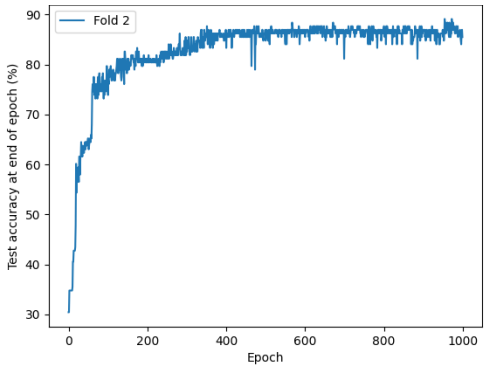


(d) Fold 4 (e) Fold 5

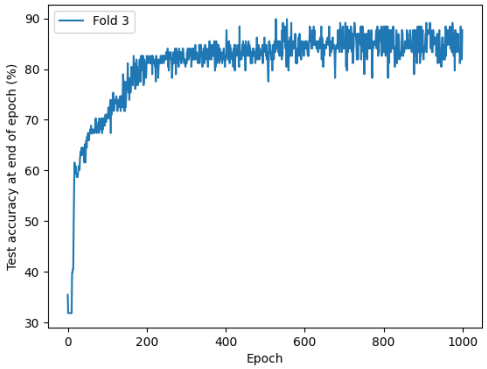
Figure A.6: Confusion matrix results after training without dropout



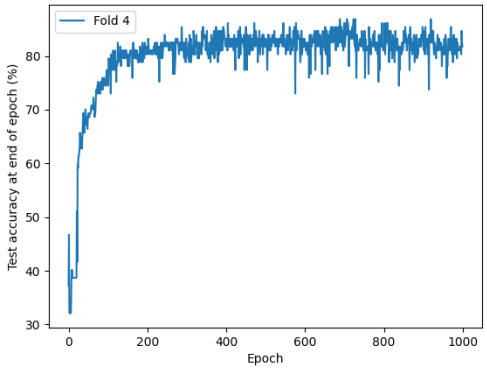
(a) Fold 1



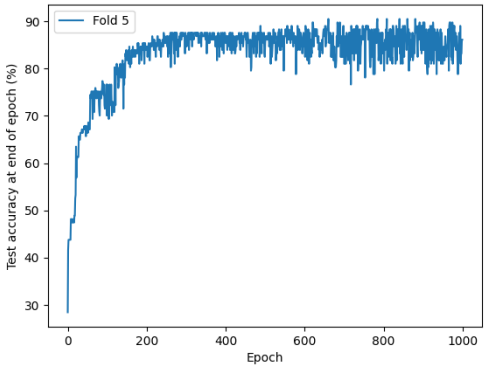
(b) Fold 2



(c) Fold 3

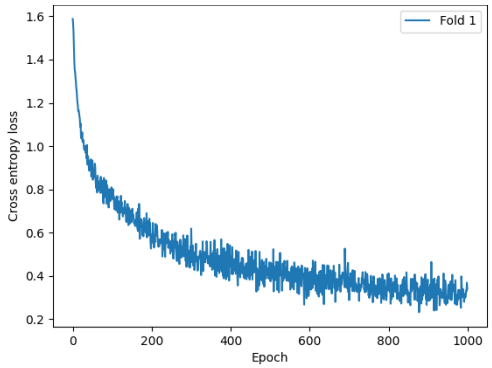


(d) Fold 4

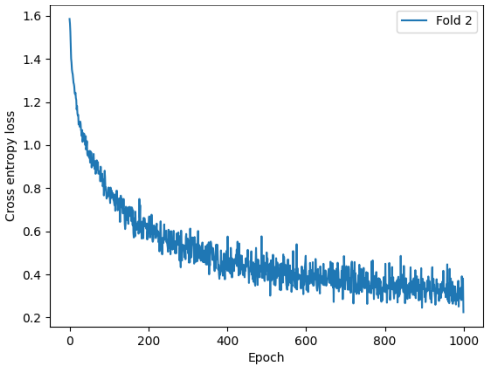


(e) Fold 5

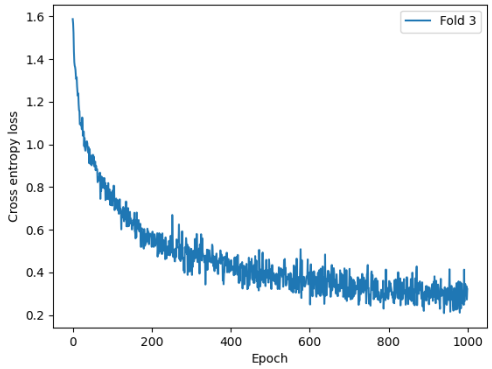
Figure A.7: Validation accuracy during training with 20% dropout



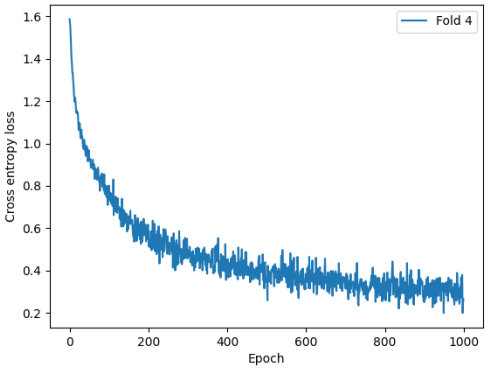
(a) Fold 1



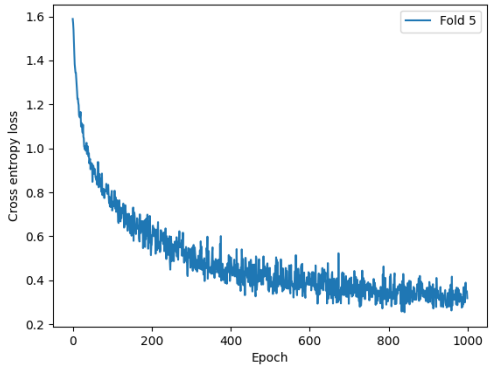
(b) Fold 2



(c) Fold 3

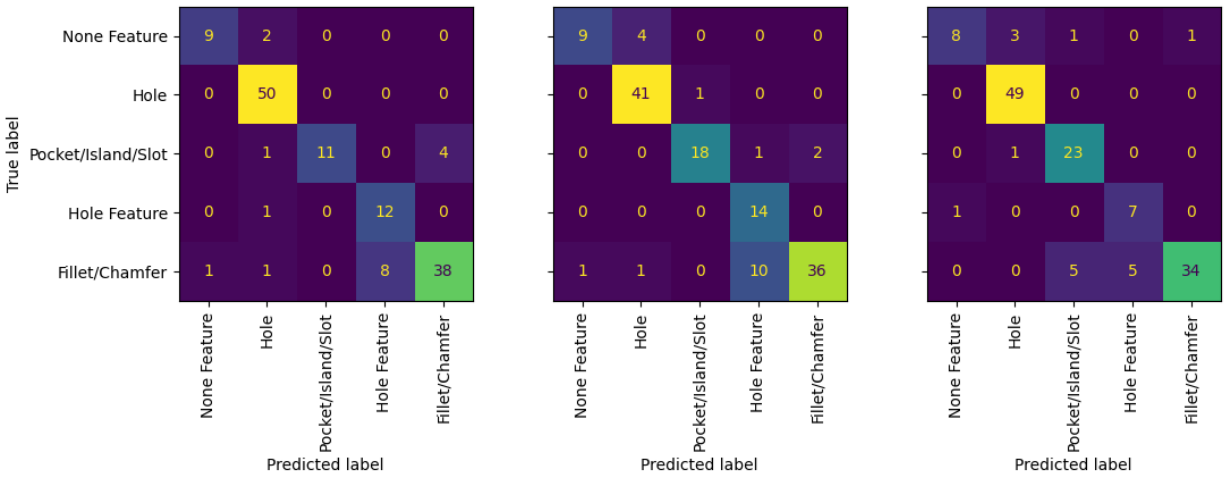


(d) Fold 4

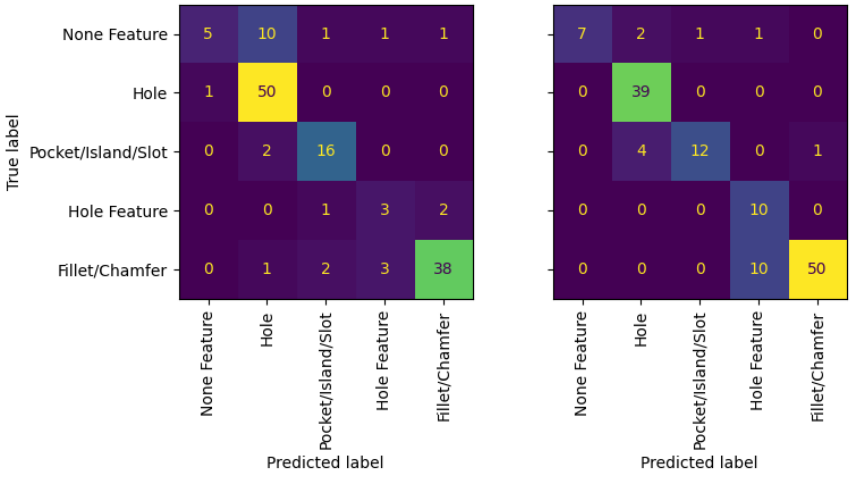


(e) Fold 5

Figure A.8: Cross entropy training loss during training with 20% dropout



(a) Fold 1 (b) Fold 2 (c) Fold 3

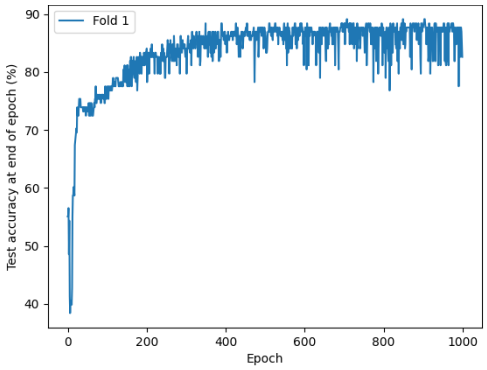


(d) Fold 4 (e) Fold 5

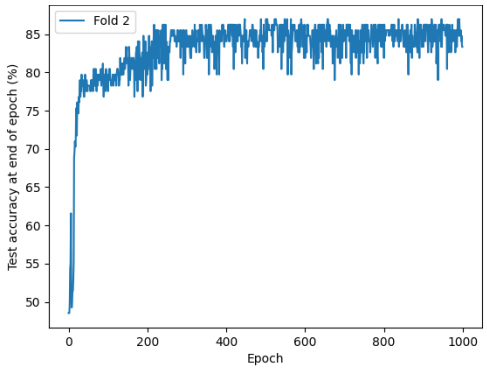
Figure A.9: Confusion matrix results after training without dropout

A.2 Results from Part 2: Evaluation of ID3 Tree Pre-classification and Crossover Data Generation

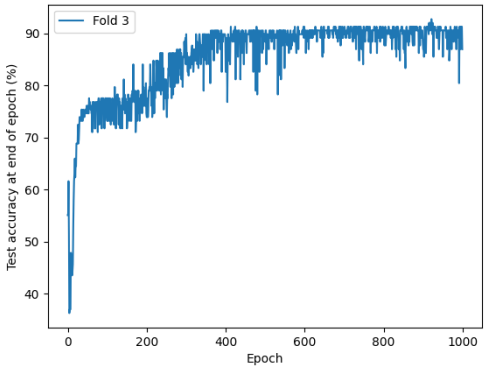
The validation accuracy during training for each fold with a 1-deep ID-3 tree preclassification step and without crossover data generation, 2-deep ID-3 tree and without crossover data generation, no ID-3 tree pre-classification and with crossover data generation, a 1-deep ID-3 tree and crossover data generation, and a 2-deep ID-3 tree and crossover generation are recorded in Figures A.10, A.13, A.16, A.19 and A.22, respectively. The cross-entropy training loss for each fold for the same set of tests are recorded in Figures A.11, A.14, A.17, A.20 and A.23, respectively. The confusion matrix outlining the performance of each model trained on each fold of data for the same set of tests are recorded in Figures A.12, A.15, A.18, A.21 and A.24, respectively.



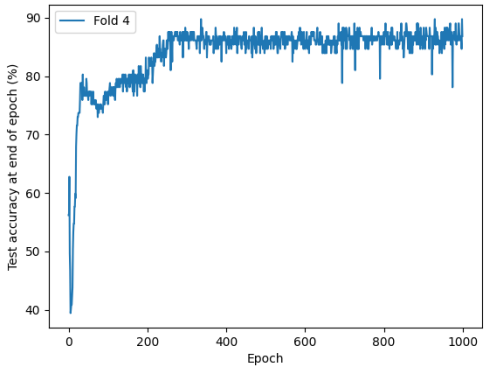
(a) Fold 1



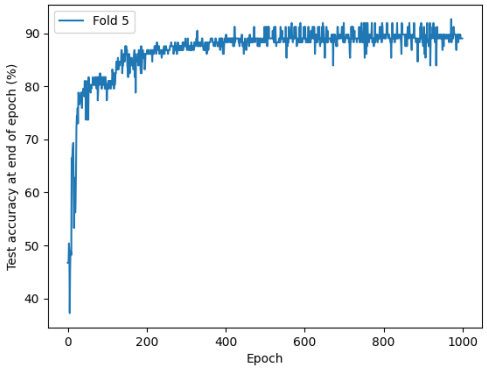
(b) Fold 2



(c) Fold 3

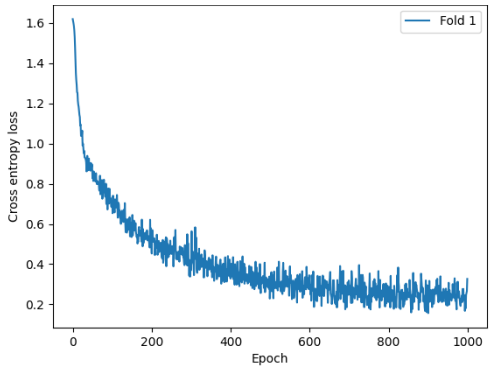


(d) Fold 4

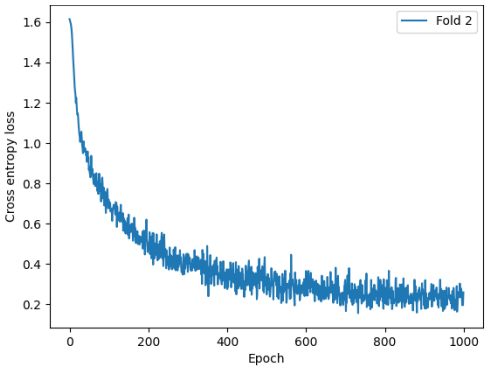


(e) Fold 5

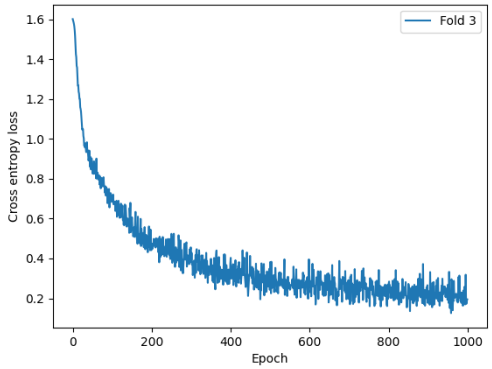
Figure A.10: Validation accuracy during training with pre-classification using a 1-deep ID3 tree



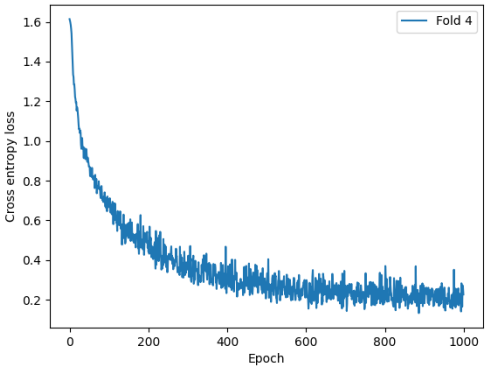
(a) Fold 1



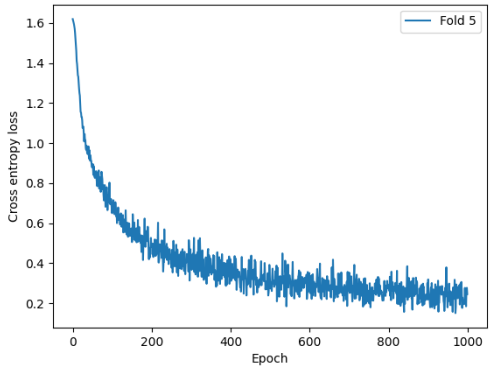
(b) Fold 2



(c) Fold 3

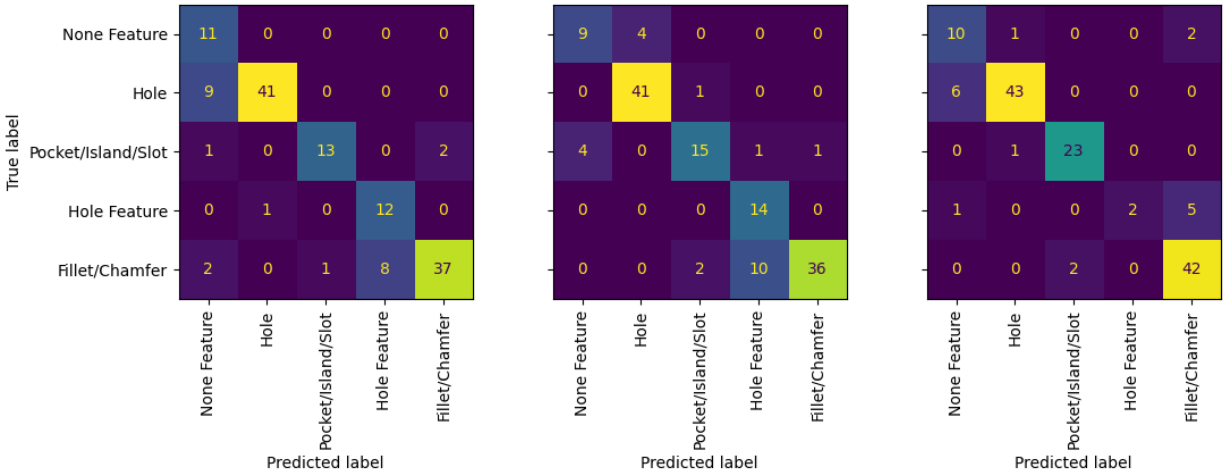


(d) Fold 4

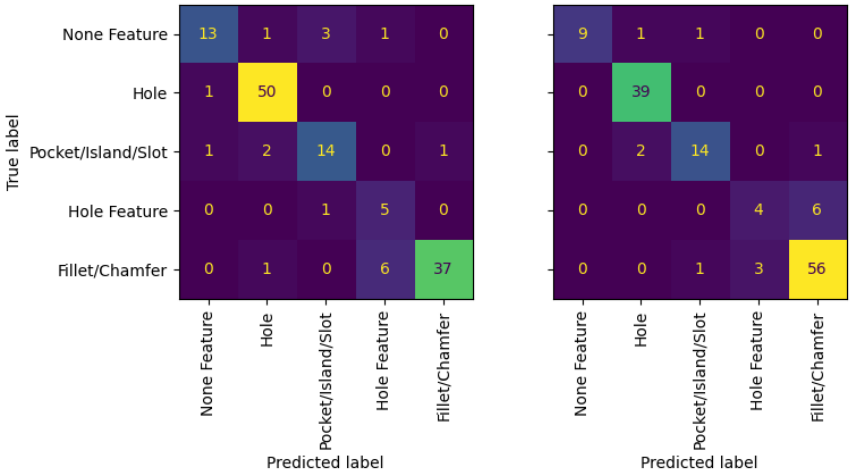


(e) Fold 5

Figure A.11: Cross entropy loss during training with pre-classification using a 1-deep ID3 tree

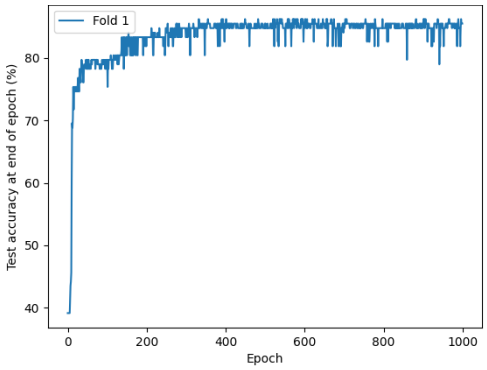


(a) Fold 1 (b) Fold 2 (c) Fold 3

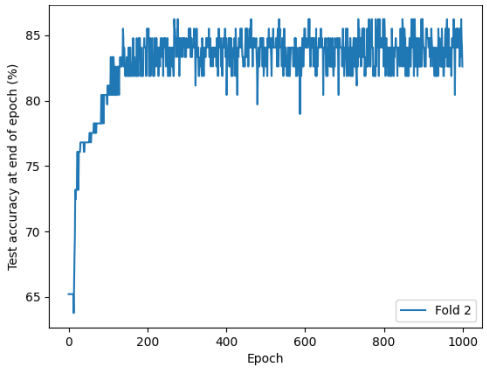


(d) Fold 4 (e) Fold 5

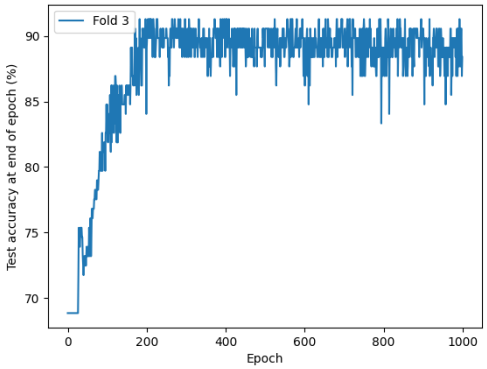
Figure A.12: Confusion matrix results after training with pre-classification using a 1-deep ID3 tree



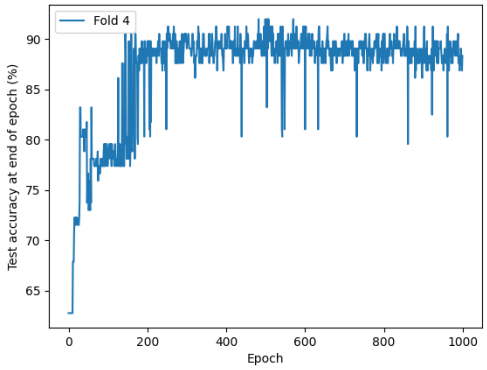
(a) Fold 1



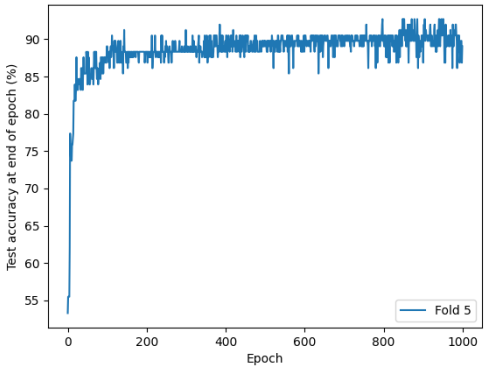
(b) Fold 2



(c) Fold 3

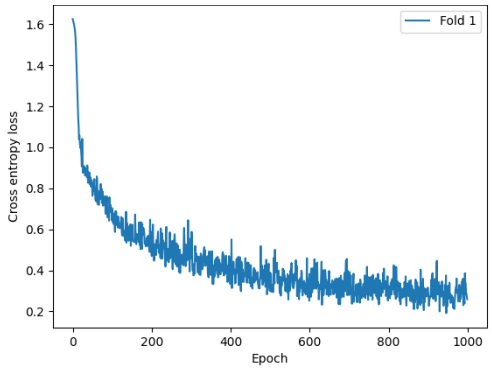


(d) Fold 4

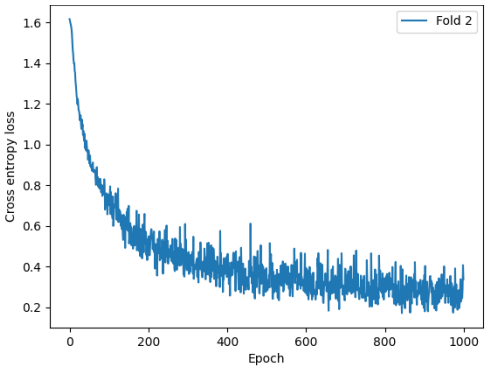


(e) Fold 5

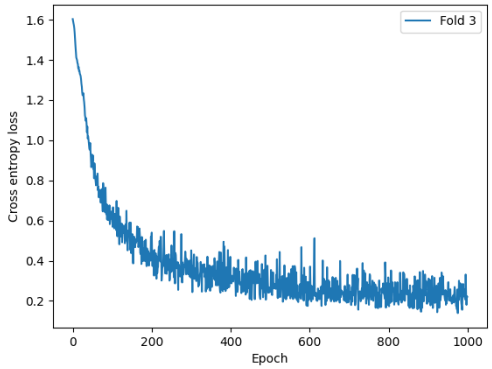
Figure A.13: Validation accuracy during training with pre-classification using a 2-deep ID3 tree



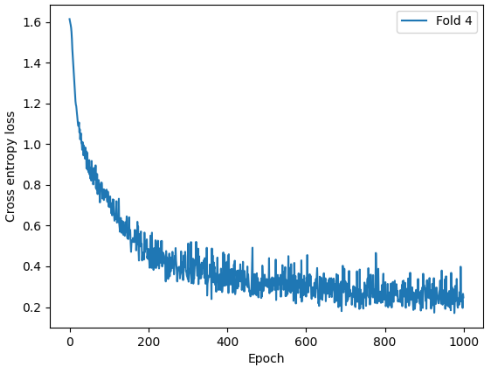
(a) Fold 1



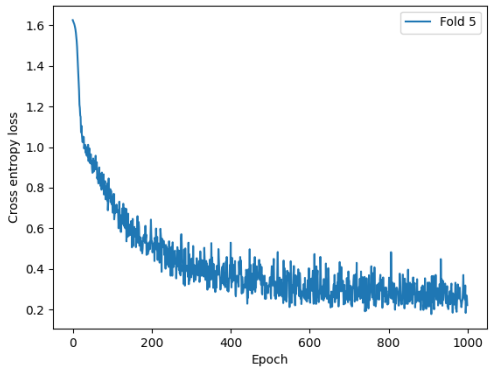
(b) Fold 2



(c) Fold 3

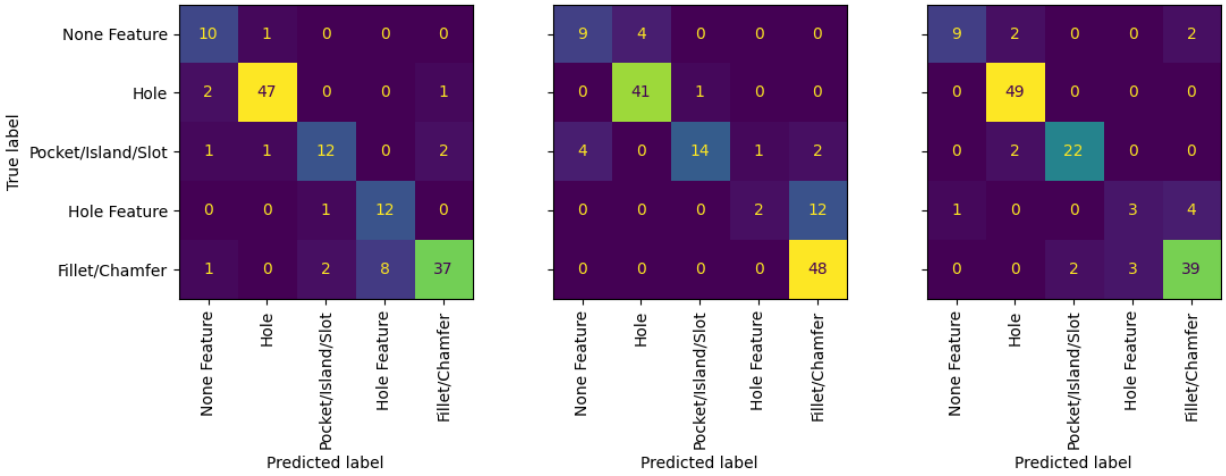


(d) Fold 4

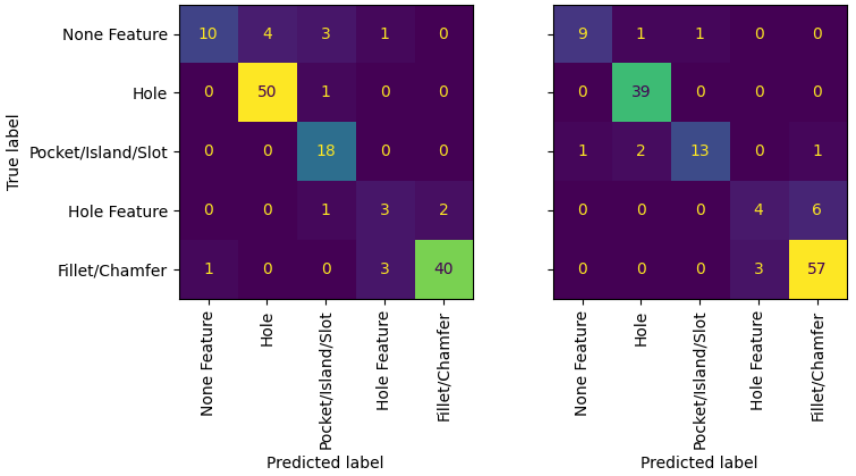


(e) Fold 5

Figure A.14: Cross entropy loss during training with pre-classification using a 2-deep ID3 tree

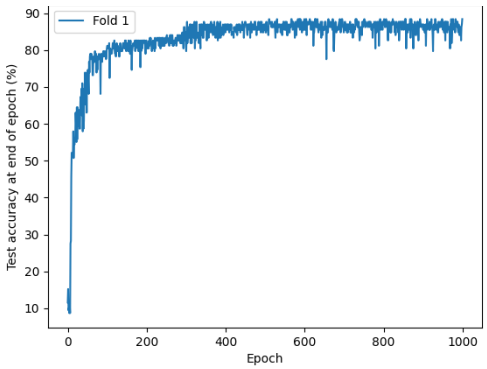


(a) Fold 1 (b) Fold 2 (c) Fold 3

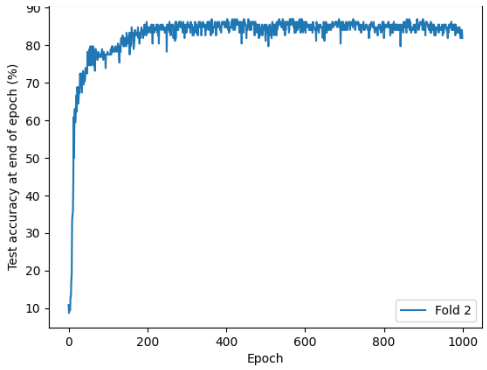


(d) Fold 4 (e) Fold 5

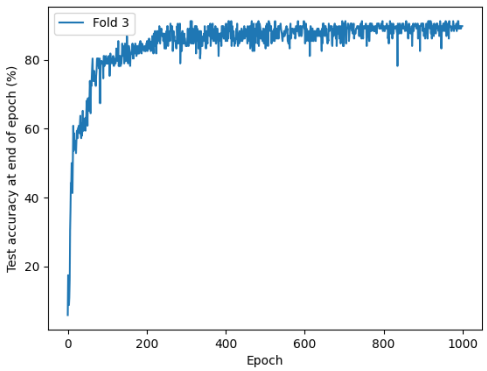
Figure A.15: Confusion matrix results after training with pre-classification using a 2-deep ID3 tree



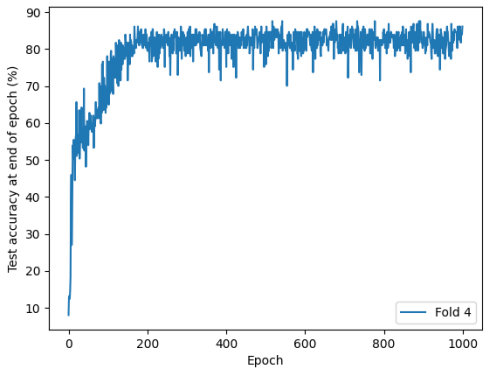
(a) Fold 1



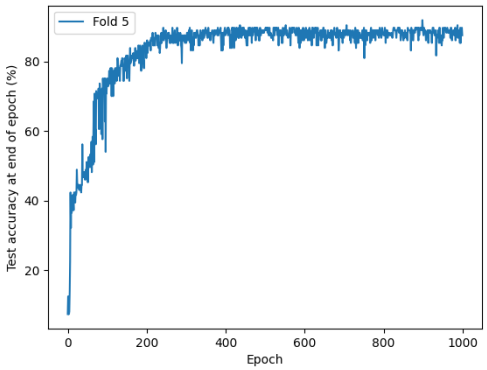
(b) Fold 2



(c) Fold 3

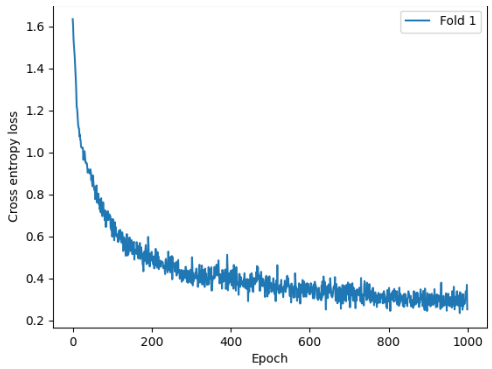


(d) Fold 4

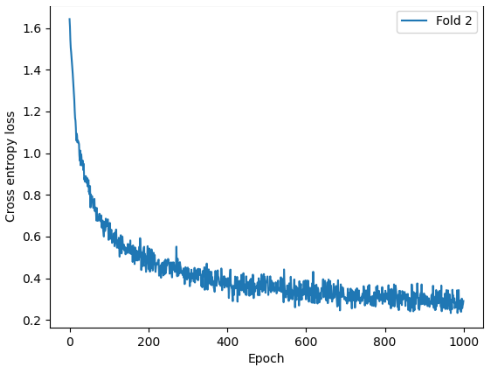


(e) Fold 5

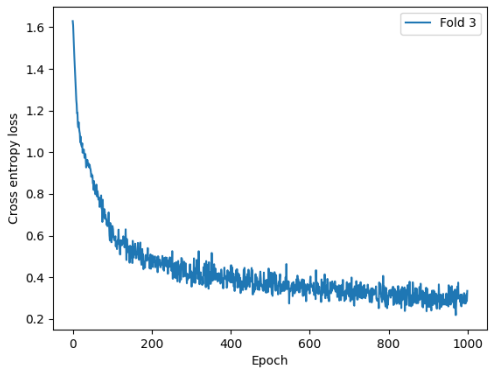
Figure A.16: Validation accuracy during training on dataset augmented with crossover



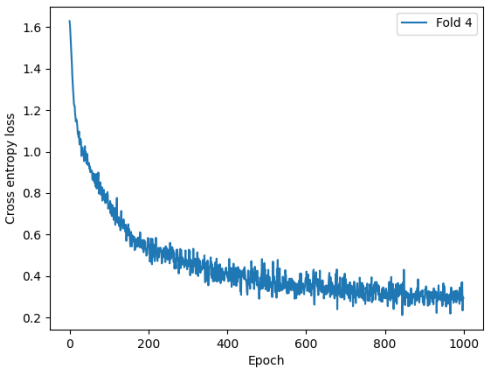
(a) Fold 1



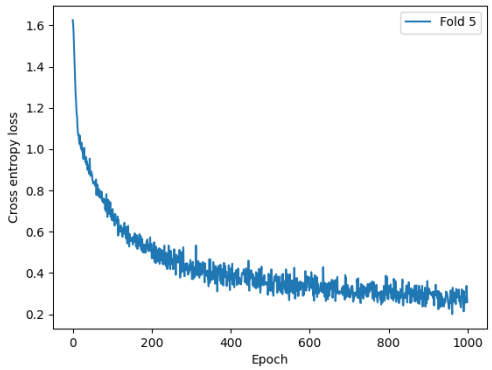
(b) Fold 2



(c) Fold 3

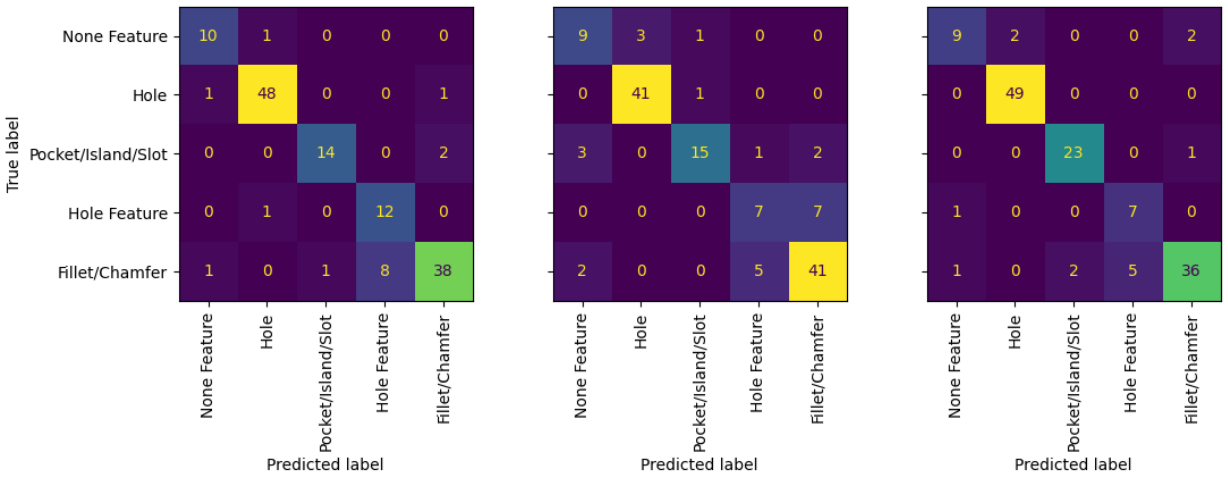


(d) Fold 4

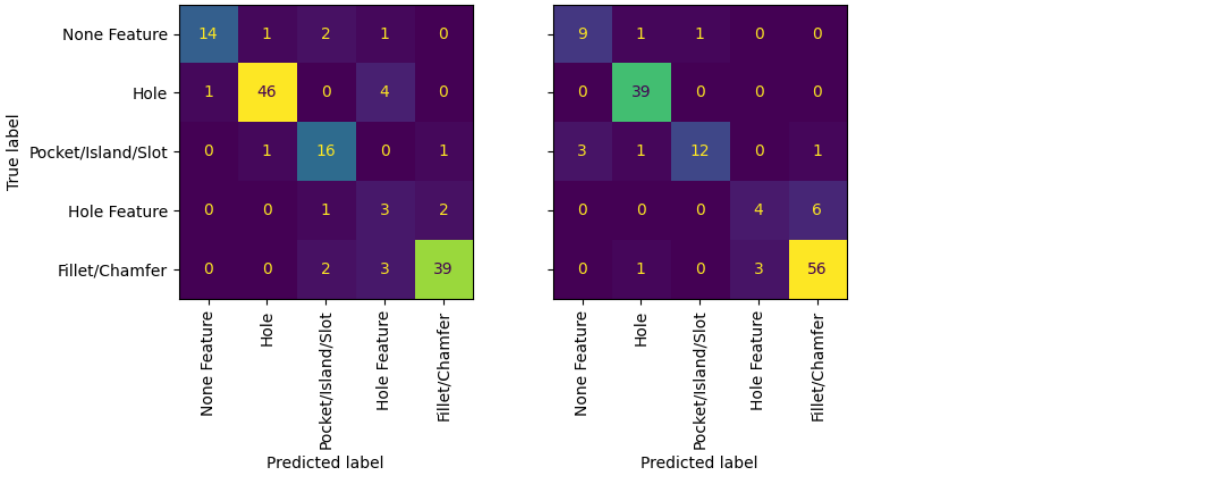


(e) Fold 5

Figure A.17: Cross entropy loss during training on dataset augmented with crossover

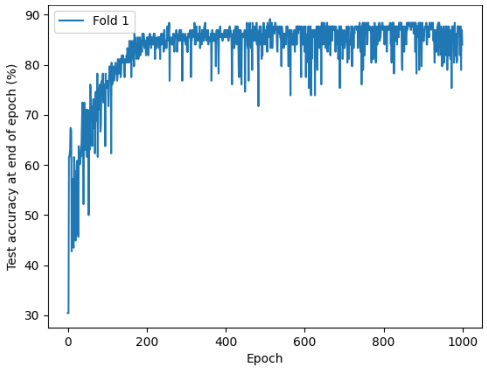


(a) Fold 1 (b) Fold 2 (c) Fold 3

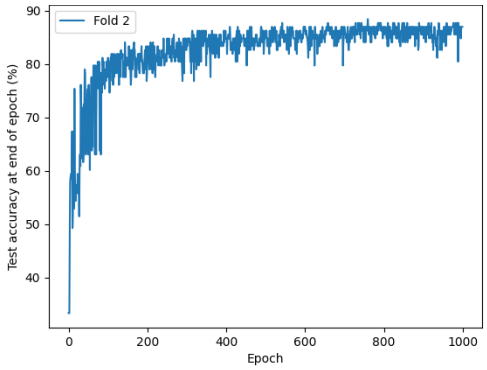


(d) Fold 4 (e) Fold 5

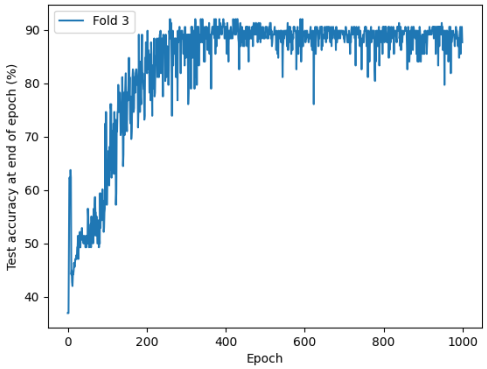
Figure A.18: Confusion matrix results after training on dataset augmented with crossover



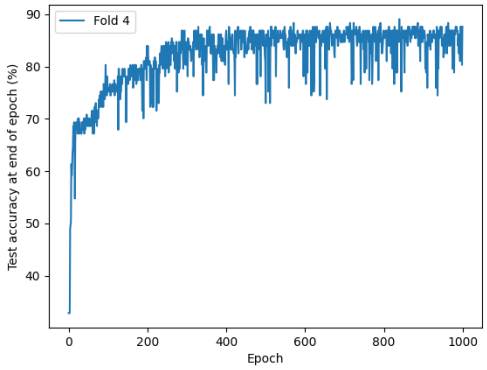
(a) Fold 1



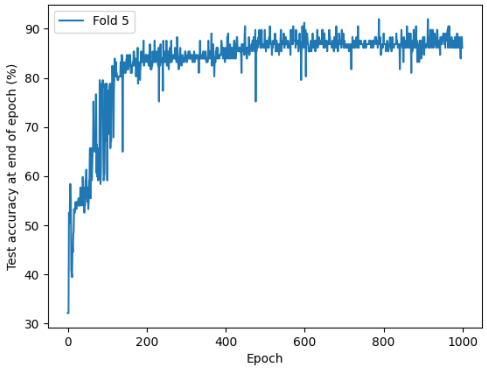
(b) Fold 2



(c) Fold 3

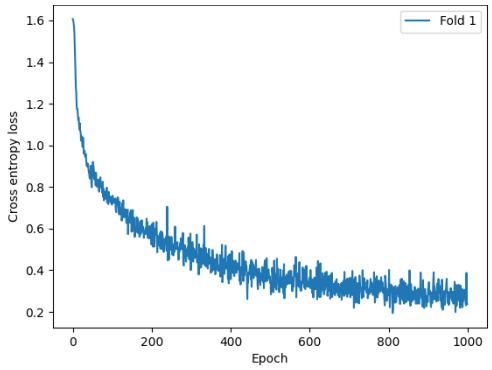


(d) Fold 4

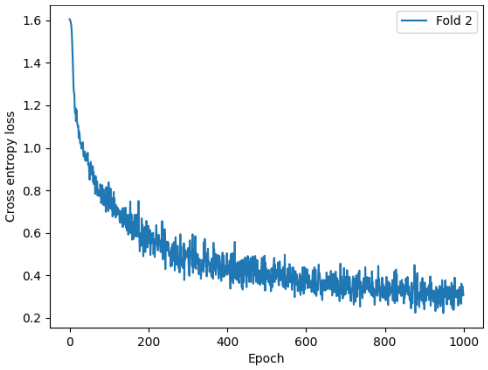


(e) Fold 5

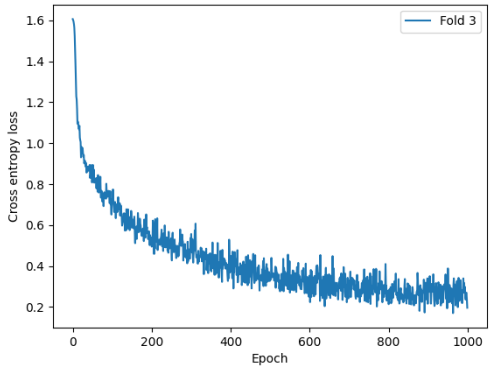
Figure A.19: Validation accuracy during training on dataset augmented with crossover and with pre-classification using a 1-deep ID3 tree



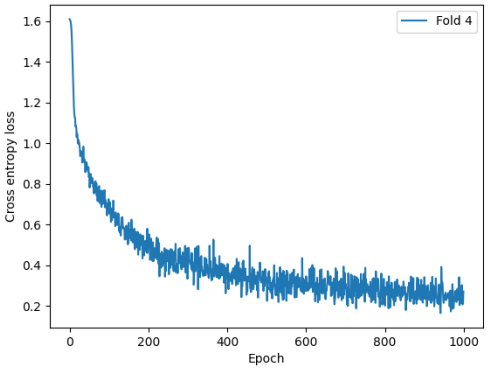
(a) Fold 1



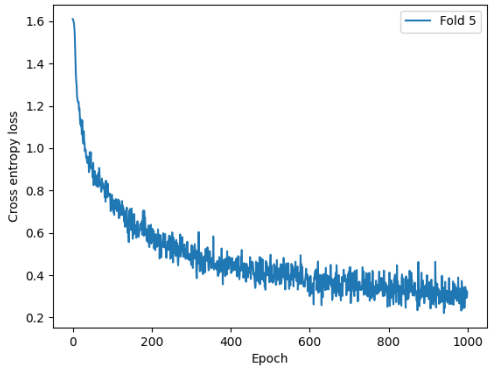
(b) Fold 2



(c) Fold 3

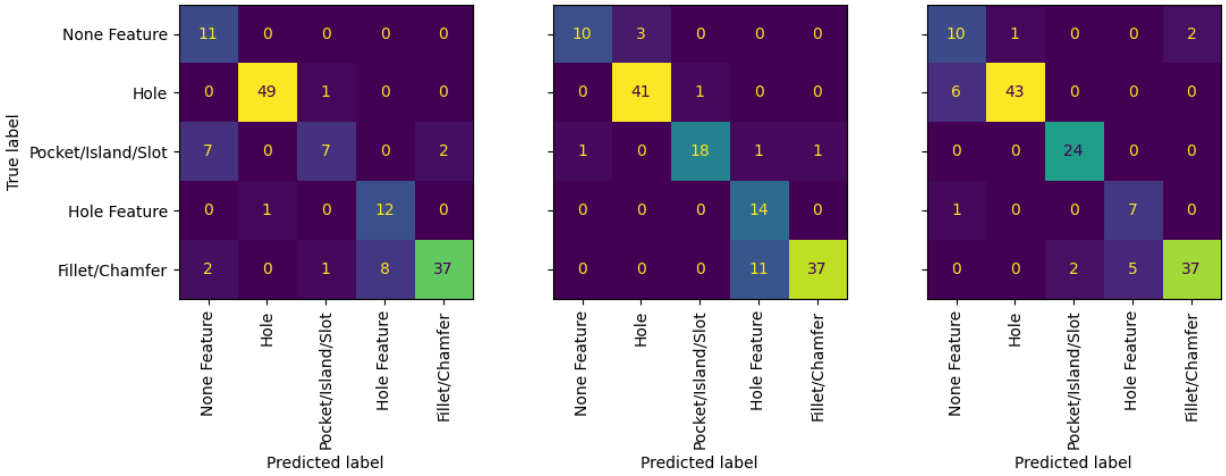


(d) Fold 4

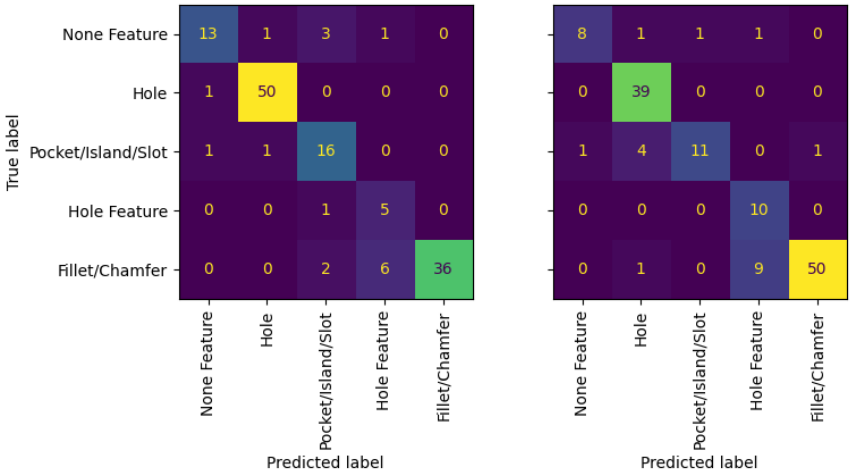


(e) Fold 5

Figure A.20: Cross entropy loss during training on dataset augmented with crossover and with pre-classification using a 1-deep ID3 tree

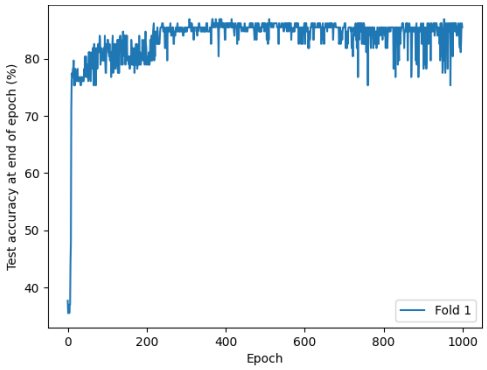


(a) Fold 1 (b) Fold 2 (c) Fold 3

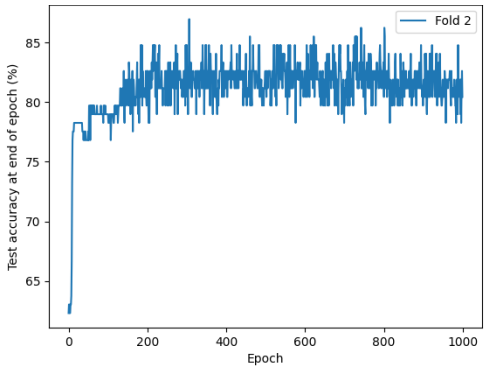


(d) Fold 4 (e) Fold 5

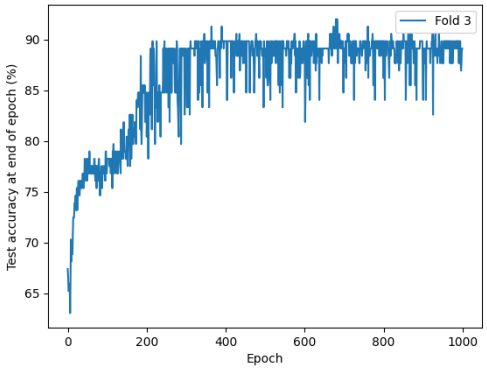
Figure A.21: Confusion matrix results after training on dataset augmented with crossover and with pre-classification using a 1-deep ID3 tree



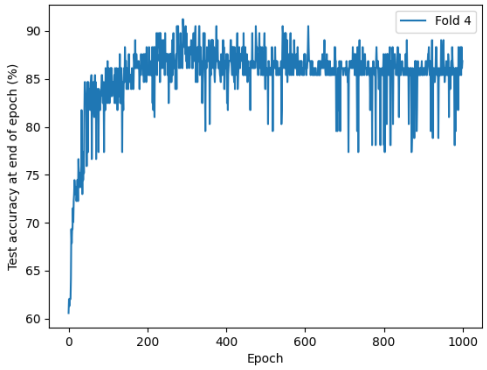
(a) Fold 1



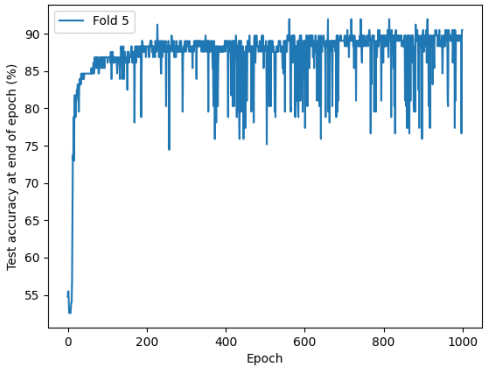
(b) Fold 2



(c) Fold 3

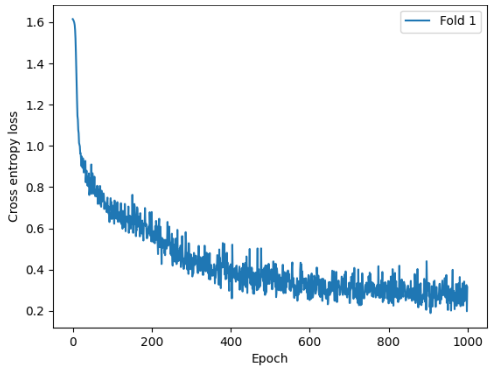


(d) Fold 4

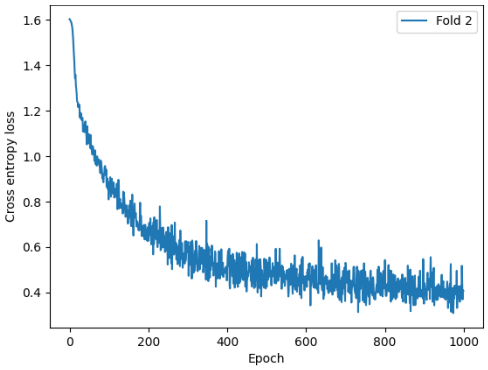


(e) Fold 5

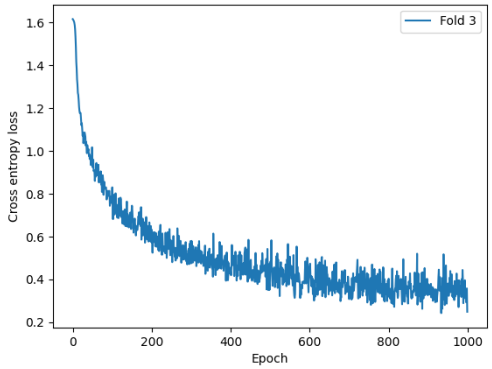
Figure A.22: Validation accuracy during training on dataset augmented with crossover and with pre-classification using a 2-deep ID3 tree



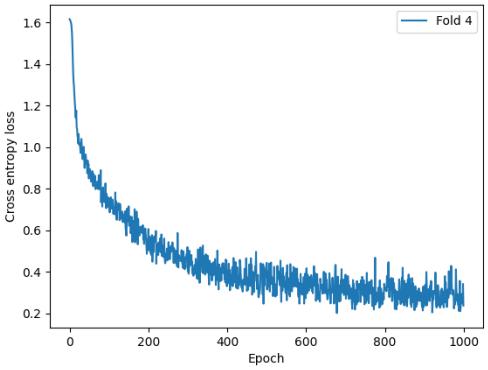
(a) Fold 1



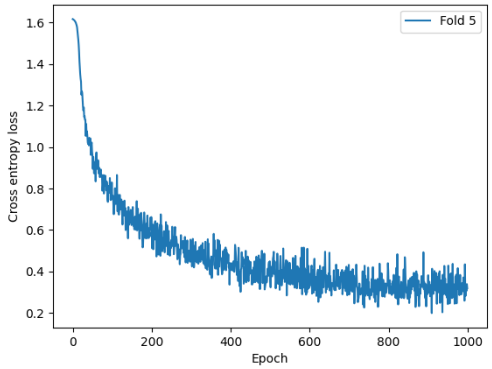
(b) Fold 2



(c) Fold 3

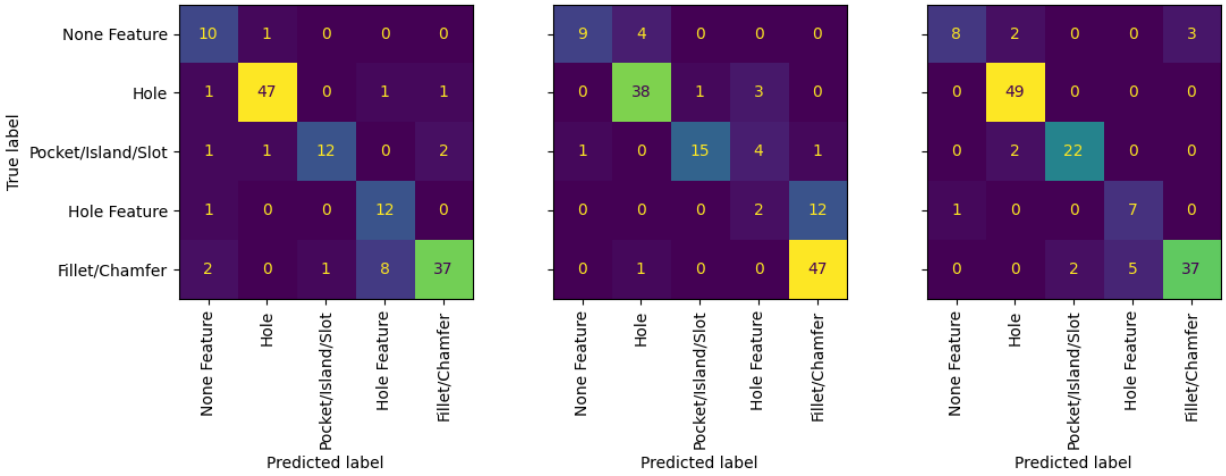


(d) Fold 4



(e) Fold 5

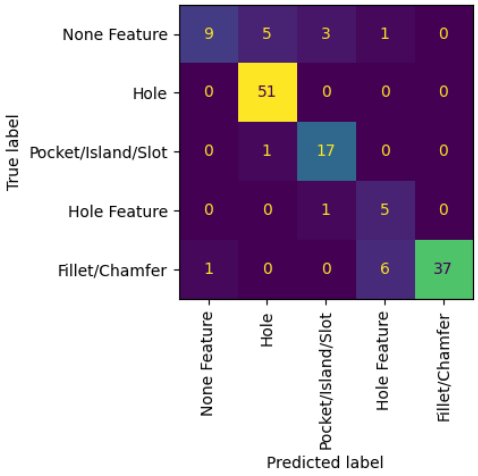
Figure A.23: Cross entropy loss during training on dataset augmented with crossover and with pre-classification using a 2-deep ID3 tree



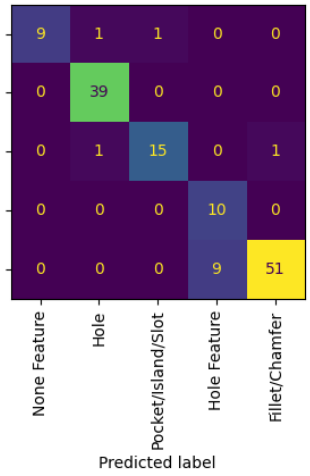
(a) Fold 1

(b) Fold 2

(c) Fold 3



(d) Fold 4



(e) Fold 5

Figure A.24: Confusion matrix results after training on dataset augmented with crossover and with pre-classification using a 2-deep ID3 tree

A.3 Results from Part 3: Evaluation of Transfer Learning

The cross-entropy training loss of the model trained using 10% dropout and a 1-deep ID-3 tree on all machining feature training samples collected from CAD files in the ABC dataset is reported in Figure A.25.

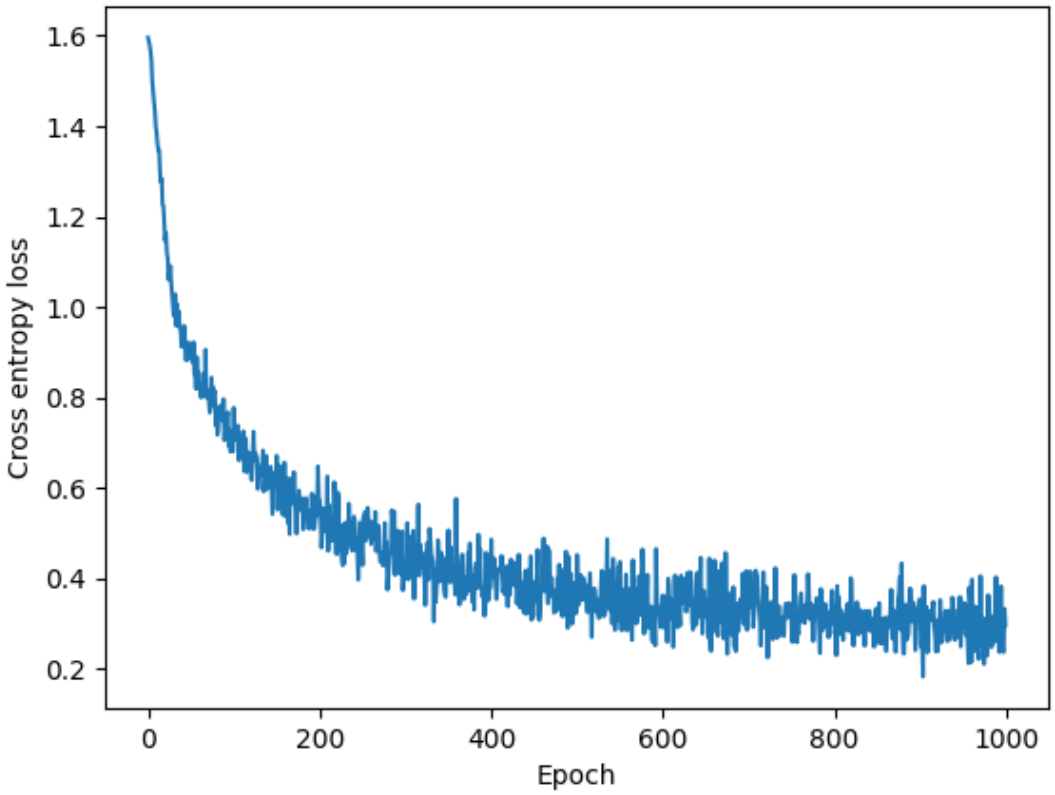


Figure A.25: Training loss of model trained on ABC dataset