
Article:
Rooker, T., Stammers, J., Worden, K. et al. (3 more authors) (2022) Error motion trajectory-driven diagnostics of kinematic and non-kinematic machine tool faults. Mechanical Systems and Signal Processing, 164. 108271. ISSN 0888-3270

https://doi.org/10.1016/j.ymssp.2021.108271

Mechanical Systems and Signal Processing 164 (2022) 108271

Contents lists available at ScienceDirect

Mechanical Systems and Signal Processing

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

Error motion trajectory-driven diagnostics of kinematic and non-kinematic machine tool faults

T. Rooker [a,b,*], J. Stammers [c], K. Worden [b], G. Potts [d], K. Kerrigan [c], N. Dervilis [b]

[a] Industrial Doctorate Centre in Machining Science, Department of Mechanical Engineering, University of Sheffield, Mappin Street, Sheffield, S1 4DT, UK

[b] Dynamics Research Group, Department of Mechanical Engineering, University of Sheffield, Mappin Street, Sheffield, S1 4DT, UK

[c] The University of Sheffield Advanced Manufacturing Research Centre, Catcliffe, Rotherham, S60 5TZ, UK

[d] Metrology Software Products Ltd., 6F Greensfield Court, Greensfield Park, Alnwick, NE66 2DE, UK

ARTICLE INFO

Communicated by Y. Lei

Keywords:
Multi-axis machining
Error motion trajectory/volumetric error
Machine tool condition monitoring
Ensemble learning
Transfer learning

ABSTRACT

Error motion trajectory data are routinely collected on multi-axis machine tools to assess their operational state. There is a wealth of literature devoted to advances in modelling, identification and correction using such data, as well as the collection and processing of alternative data streams for the purpose of machine tool condition monitoring. Until recently, there has been minimal focus on combining these two related fields. This paper presents a general approach to identifying both kinematic and non-kinematic faults in error motion trajectory data, by framing the issue as a generic pattern recognition problem. Because of the typically-sparse nature of datasets in this domain – due to their infrequent, offline collection procedures – the foundation of the approach involves training on a purely simulated dataset, which defines the theoretical fault-states observable in the trajectories. Ensemble methods are investigated and shown to improve the generalisation ability when predicting on experimental data. Machine tools often have unique 'signatures' which can significantly-affect their error motion trajectories, which are largely repeatable, but specific to the individual machine. As such, experimentally-obtained data will not necessarily be easily defined in a theoretical simulation. A transfer learning approach is introduced to incorporate experimentally-obtained error motion trajectories into classifiers which were trained primarily on a simulation domain. The approach was shown to significantly improve experimental test set performance, whilst also maintaining all theoretical information learned in the initial, simulation-only training phase. The ultimate approach represents a viable and powerful automated classifier for error motion trajectory data, which can encode theoretical fault-states with efficacy whilst also remain adaptable to machine-specific signatures.

1. Introduction

The ongoing technological advancements in machine tool and controller design have opened up ever-increasing levels of precision, accuracy and production capabilities. The proliferation of the multi-axis machine tool[1] was a significant step, enabling the production of more complex workpiece geometries, more efficient operation, extended tool life and improved surface finish quality.

* Corresponding author at: Industrial Doctorate Centre in Machining Science, Department of Mechanical Engineering, University of Sheffield, Mappin Street, Sheffield, S1 4DT, UK.

E-mail address: tjrooker1@sheffield.ac.uk (T. Rooker).

[1] Defined as a machine tool which incorporates four or more axes of motion, most commonly comprising two rotational axes in addition to the three traditional translational axes.

https://doi.org/10.1016/j.ymssp.2021.108271
Received 14 October 2020; Received in revised form 11 June 2021; Accepted 22 July 2021
Available online 6 August 2021

Glossary

CNC	Computerised Numerical-Control
MCTM	Machine Tool Condition Monitoring
HTM	Homogeneous Transformation Matrix
ANN	Artificial Neural Network
ReLU	Rectified Linear Unit
CNN	Convolutional Neural Network
MC	Multi-Class (ensemble classifier)
OVR	One-Vs-Rest (ensemble classifier)
OVO	One-Vs-One (ensemble classifier)
FT	Fine-tuning (ensemble classifier)
CF	Catastrophic Forgetting
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
PR	Precision–Recall (curve)
AUPRC	Area Under Precision–Recall Curve

Despite these advancements, real structures are never perfectly rigid, and machine tools are no exception to this rule. Variation in quasi-static and dynamic error sources [1], that affect machining accuracy, arise due to local temperature fluctuations, in-process conditions, significant events (such as a tool crash, or calibration activity), errors in the size and form of machine tool components as well as general wear of moving elements throughout normal operation. Consequently, it is not possible to maintain perfect accuracy and precision in manufacturing at all times; there will always be some observable, quantifiable degree of error present on a finished workpiece, as compared to its idealised specification.

Extending this concept of imperfection, it is widely-appreciated that the performance of a given manufacturing system will change throughout its operating life [2], as its compound error profile is affected through varying levels of use. Multi-axis machine tools are particularly susceptible to this problem because of the inclusion of rotational axes, which introduce additional nonlinear error components to the system. Owing to this issue, and amid requirements for repeatable performance in the manufacturing process, there has naturally been considerable research interest in multi-axis machine tool error identification.

ISO 230-7 [3] defines two properties which affect the geometric accuracy of the rotary-axes. An *axis shift* is defined as the 'quasi-static relative angular and linear displacement, between the tool and workpiece sides, of the axis average line due to a change in conditions'. The axis average line is a straight line segment, with respect to the reference coordinate axes, representing the mean location of the axis of rotation. *Error motions*, on the other hand, are defined as 'unwanted changes in the position and orientation of an axis of rotation, relative to its axis average line as a function of angular position of the rotating component'. In other words, the axis shifts are single value parameters which quantify positional or orientation deviations from the ideal of the rotation-axis, whereas the error motion is a function which defines the actual *trajectory* of the error, in a three-dimensional coordinate space. It is worth noting that various literature sources use alternative terms to describe machine tool error in the general sense, with examples including *volumetric error*, *volume error* or *geometric error*. For absolute clarity, however, this paper will maintain the latest definitions formalised in ISO 230-7.

Broadly speaking, the process of error identification involves applying a method for the collection of error motion trajectory data, and using these data to either calculate the axis shift values for kinematic compensation, or determine the individual error motion functions for a more-comprehensive solution. ISO 230-7 [3] currently recognises the *R-Test* [4] for error identification, whereby a precision sphere, mounted on three orthogonally-oriented linear displacement sensors, is located at various indexations of the axis of rotation. The classic R-Test is a dynamic testing method, such that data are collected continually throughout axis motion. The data collected by the standardised procedures reveal the error motion trajectory, from which the error motion functions or axis shift values can be determined.

Static variants of the R-Test, wherein the procedure and equipment setup is replicated, but error motion trajectory data are collected at discrete intervals, have been investigated in the literature for objectives such as complete identification of all axis shifts and error motions in the rotary-axes [5]. Another approach, involving locating a rectangular artefact with a touch-trigger probe [6], was shown to be effective for identifying and calibrating axis shift errors. The work was then extended to complete identification of axis shifts and error motions, with a similar artefact probing approach [7]. Spherical artefacts are often favourable to other geometries, due to their nominally-identical form when approached from different angles. The *scale and master balls artefact* method [8] employs touch-trigger probing to locate a collection of precision spheres at various axis positions, and has been shown to provide the necessary trajectory data to estimate all axis shift errors in a multi-axis machine tool. Locating a single spherical artefact to evaluate the rotary-axes is less often considered in the research domain; however, it is widely adopted in industry [9], being regularly applied for both pre-production checks and to inform maintenance or calibration activities.

Error motion trajectories are a somewhat-underrated consideration for a condition-based maintenance strategy, whereby diagnostics and maintenance policy decisions are informed by the system condition directly. Significant research focus with regard to condition-based maintenance has been afforded to Machine Tool Condition Monitoring (MTCM). In-process monitoring of the machining process [10] has been studied intensively, with particular interest recorded in monitoring of the cutting tool wear and remaining useful life [11,12]. Others have placed precedent on monitoring the structural and functional components which physically comprise the machine tool. Monitoring of the servo-motor current has been proposed [13] as a method for identifying faults in the rotary-axes of machine tools. Modern machine tool controller systems are often equipped with self-diagnostic tools to monitor faults in the servos and other electrical issues [14]. Approaches for monitoring the power consumption in machine tools have been applied for both energy efficiency monitoring [15] and online fault monitoring [16]. Frameworks for managing the multi-dimensional data streams which are collected with online monitoring techniques have been considered with cloud-based techniques [17] and structured ontologies [18]. More recently, the research focus has been extended to data mining applications for data acquisition in populations of multiple machine tools, which is an important precursor for the implementation of *Industrie 4.0* [19].

1.1. Motivation

Generally, the research landscape covering machine tool error is primarily concerned with the identification of the axis shift and error motion parameters, which can be clearly defined in a kinematic model and corrected with a numerical best-fitting process. Although closely related to the interest in MTCM, the two fields have, historically, been largely separate entities. Recent state-of-the-art work has begun to bridge this gap — notably in the population-based assessment of *double ball-bar* data for machine tool diagnostics [20], and the extension of the scale and master balls artefact technique to a condition monitoring system [21] [22]. However, research concerning MTCM through the analysis of error motion trajectory data is still sparse, with most publications in the field focusing on more-traditional vibration-, force- or power-based condition-based maintenance approaches, to identify in-process condition deviations which correlate with physical instances of mechanical damage, or electrical faults in the system.

Error motion trajectories, acquired via inspection techniques, are an interesting alternative data source for an MTCM approach. The data acquired have the potential to reveal certain non-kinematic faults – such as mechanical damage or electrical faults – as well as the identification of kinematic errors, for which, they have been traditionally been developed and applied. Prior to this paper, this broader capability has been seldom-explored, with the majority of error identification publications which utilise the data tending to overlook the non-kinematic fault detection capability. In production environments, specialist professionals can manually interpret the trajectory data [9] to diagnose controller software faults or mechanical axis issues, which cannot necessarily be represented in a kinematic model. Understandably, there are issues with the current approach. Manual interpretation of the reports is time-intensive, and being a specialist activity, it is not accessible for all users. Moreover, such a manual approach runs the risk of subjectivity, meaning the diagnostic outcome may vary between individuals and those with different skill-levels. An assistive, automated diagnostic system is thus required to address the issue in the industrial domain.

1.2. Contribution

This paper presents a novel approach to MTCM via direct study of error motion trajectories, for a fault diagnostics system unconstrained by the kinematic error model and in a departure from the common school-of-thought in the field. Removing this constraint allows for a system which can emulate the manual expert, extending the range of identifiable faults, through the use of more-generic pattern recognition techniques. The approach is applicable for identifying the main kinematic errors affecting machining accuracy, as well as alternative sources of error which are identifiable in the error motion trajectory.

Currently in industry, the fault diagnostic process is conducted by specialist consultants or experienced Maintenance Engineers, via manual interpretation of the data. The manual process relies heavily on user skill level for success, and the process of manual interpretation itself is open to subjectivity between different users. Both of these issues can have significant negative consequences for obtaining a reliable and informative assessment of the machine's health state. The automated approach presented in this paper addresses these issues, providing a standardised tool for users of all skill levels to access expert diagnostic capabilities.

As the collection of error motion trajectory data is an offline procedure, often conducted at roughly one-week intervals in production environments, it is infeasible to build a representative dataset through experimental collection alone. This paper introduces a simulation dataset, representing five common fault-states identifiable in error motion trajectories, with the objective of performing classification on experimentally-obtained data from actively-operational systems. The paper explores ensemble learning methods to strengthen the generalisation ability of classifiers trained on purely simulated faults.

It is observed that real machines have unique signatures [23] which affect the error motion trajectory, resulting in abstractions to the fault-state representations that do not necessarily conform to expected theoretical definitions. Attempting to incorporate all possible signatures into a simulation is clearly infeasible. The final major contribution of this paper is a transfer learning implementation, providing the ability to fine-tune a classifier upon receipt of new, informative data, as it is acquired in a dynamic environment.

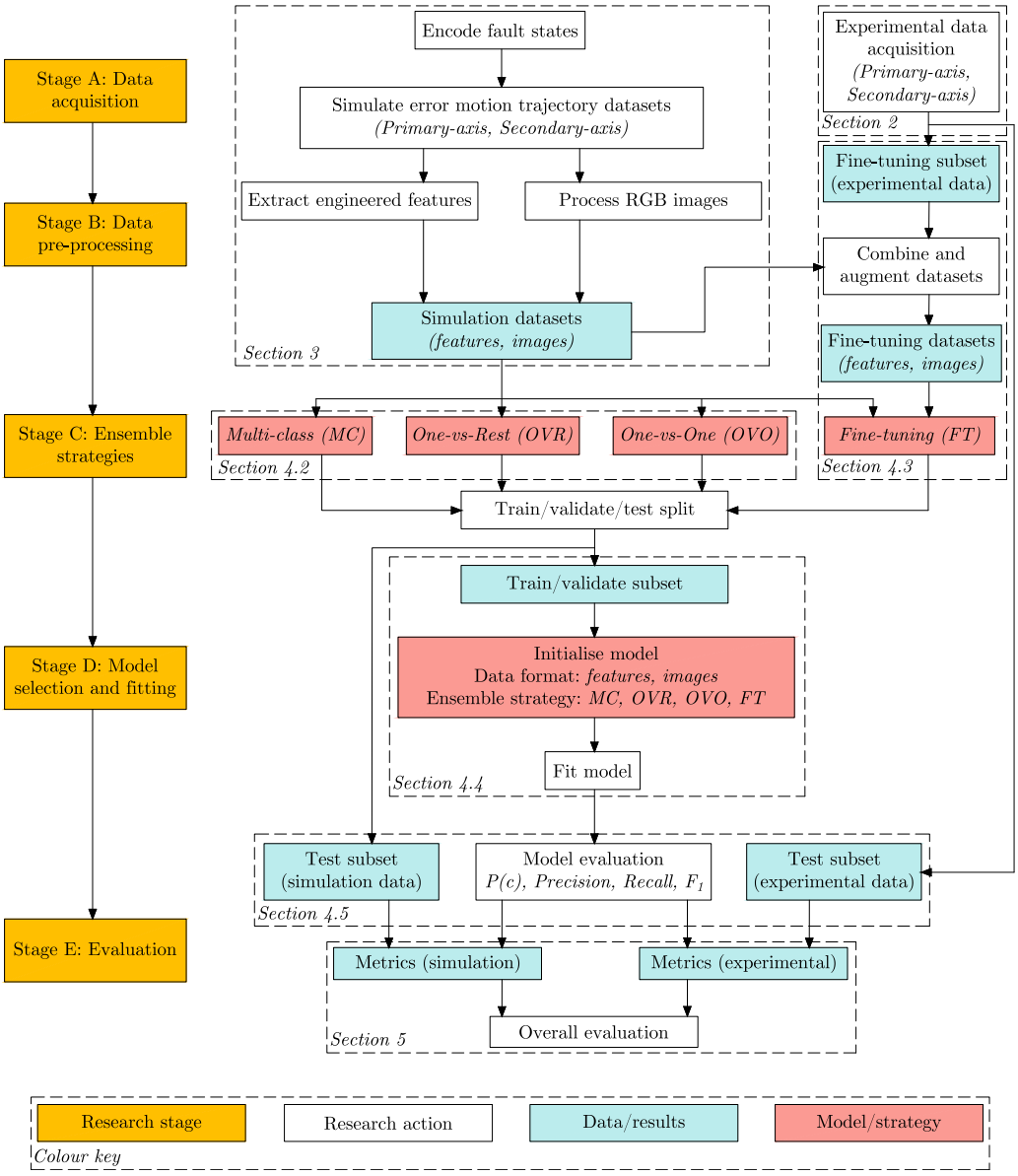


Fig. 1. Schematic illustration of the research route for this paper.

1.3. Outline of the paper

The paper is organised as follows. Section 2 firstly describes the experimental procedure conducted to acquire error motion trajectory data for this paper. Section 3 describes the fault-states which are identifiable in such data, and the simulation procedure for generating the classifier training datasets. Section 4 discusses the machine learning and ensemble methods investigated, as well as the training/validation procedures for each classifier model and general model evaluation strategy. Section 5 presents the results and discussion for both simulated and experimental datasets. Section 6 discusses the strengths/weaknesses of the proposed approach in comparison with the traditional approach of numerical best-fitting. Finally, Section 7 provides concluding remarks and avenues for future work. Fig. 1 provides a more detailed summary of the research route taken for this paper.

2. Experimental procedure

The inspection procedure [9], utilised for collecting the experimental test set in this paper, involves probing a single, spherical artefact at numerous axis positions, or *indexations*. The instrumentation required for conducting the procedure is summarised below.

(a) Probing the sphere at the home position. (b) Probing the sphere at an indexation of the secondary axis.

Fig. 2. Typical hardware setup for collecting error motion trajectory data with the artefact probing procedure.

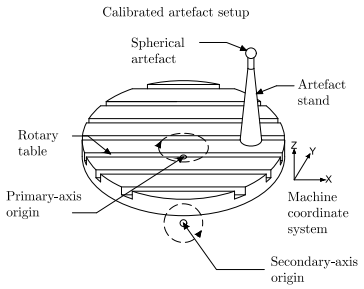


Fig. 3. Diagram of typical spherical artefact setup on a B-C configured rotary table.

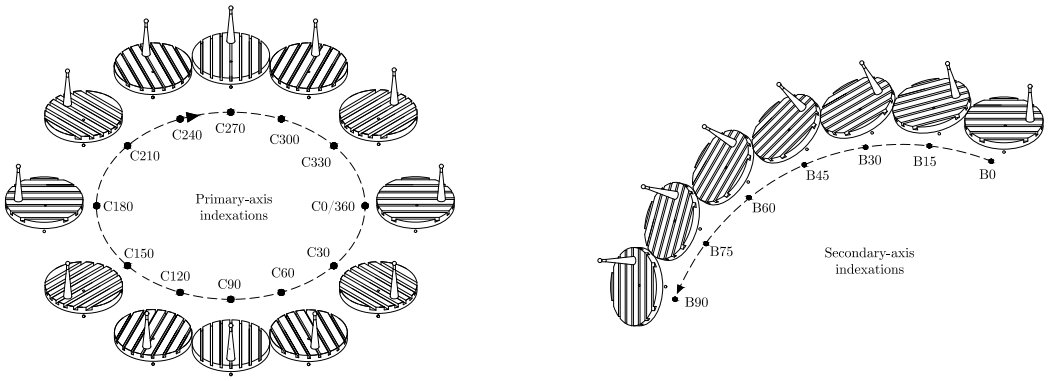
- Machine tool with five permissible motion axes (three linear, two rotary).
- Touch-trigger probing system & receiver.
- Precision spherical artefact.
- Proprietary artefact stand.
- Inspection data acquisition software (*NC-Checker* [9]).
- Personal computer and ethernet connection.

Multi-axis machine tools are most-often comprised of three linear-axes – facilitating motion in linear directions X, Y and Z – and two additional rotary-axes, which provide *Primary-* and *Secondary-axes* of rotation, denoted A, B and C for motion around X, Y and Z, respectively. Generally speaking, the Primary-axis is the C-axis, and the Secondary-axis is either A or B, dependent upon the specific configuration. In this paper, a B-C machine tool configuration will be assumed.

For each procedure, the sphere is initially located at the 'home' position, with the rotary-axes indexed at $B = 0^o$, $C = 0^o$. Fig. 2(a) shows the typical setup, with rotary-axes at the home position, and Fig. 3 shows this in diagrammatic form. Following confirmation of the performance of the probe itself (with a dedicated testing cycle), the axes are then reindexed and the sphere is located in its new location; this process is repeated until the full indexation range of the axis has been covered. The Primary-axis procedure described in this paper locates the sphere at thirteen indexed positions from $C = 0^o$ to $C = 360^o$. For the Secondary-axis procedure, the positions indexed are from $B = 0^o$ to $B = 90^o$. Fig. 2(b) shows the sphere being probed at an indexation of the Secondary-axis. As the nominal kinematics of the system are known, the probing procedure can calculate the residuals between nominal and actual for each sphere location, to produce the trajectory across the full axis index range. The full procedure is illustrated in Figs. 4(a) and 4(b).

An experimental test set was obtained from four actively-operational machine tools – two from a research environment, and two provided by industrial partners – utilising the probing procedure described above. The dataset comprised a total of 144 examples of Primary-axis error motion trajectories and 314 examples of Secondary-axis error motion trajectories, collected across a range of time periods from twelve to eighteen months.

Table 1 summarises the experimental equipment, inspection procedure and probe performance test requirements for technical reference in this paper. It should be noted that some data provided by industrial partners was collected with a different indexation step size (nine indexations of 45^o, as opposed to 13 indexations of 30^o). Fundamentally, the key properties of the error motion trajectories are unchanged, however such trajectories are presented at a slightly lower resolution than those which were collected



(a) Typical rotary-axis positions for collecting Primary-axis error motion trajectory data.

(b) Typical rotary-axis positions for collecting Secondary-axis error motion trajectory data.

Fig. 4. Indexations of the Primary- and Secondary-axes utilised for experimental data collection and simulation.

Table 1
Summary of experimental procedure parameters. Note — some parameters may differ slightly in data provided by industrial partners. Those affected are indicated with *. Another point to note is that the data was collected from four different machine tools, some of which had different controllers installed in their configuration. This variation does not affect the analysis in any way; however, the specific hardware used for data collection is included here for technical reference.

Parameter	Value
Experimental equipment	
Machine tool controller	Fanuc 31i/Heidenhain TNC640
Probe type	Renishaw OMP60
NC-Checker software version	2016r2
Stylus length	50 mm (straight)
Stylus stem material	Ceramic
Probe tip material	Ruby
Probe tip size	6 mm
Precision sphere size	25 mm
Machine tool rotary-axes	2
Inspection procedure	
Rotary-axis indexations* (Primary/Secondary)	13/7
Indexation step size* (Primary/Secondary)	$30^o/15^o$
Range of motion (Primary/Secondary)	C = 0^o to 360^o/B = 0^o to 90^o
Fixed axis (Primary/Secondary)	B = 0^o/C = 0^o
Measured points for axis tests	9
Measured points for probe performance test	25
Targeted testing interval	Once per week
Probe performance test requirements	
Probe performance (overall)	$\leq$ 25 μm
Probe pre-travel variation	$\leq$ 25 μm
Sphere position in X, Y, Z	$\leq$ 25 μm
Sphere diameter	$\leq$ 50 μm

with a smaller indexation step size. Such potential for variability is important to consider in developing a useful system for the end-user, as unique users will endeavour to adapt the procedure to fit their specific requirements. The pre-processing for feature-based and image-based datasets, described in Section 3, provides the necessary robustness for this variability.

3. Simulating fault-states in error motion trajectory data

The ability to generate informative training data artificially is imperative to building an automated diagnostic system in this domain. Error motion trajectory data are sampled discontinuously throughout the machine life-cycle, so datasets are consequently often small; therefore, training a classifier on real data is likely to be infeasible and ineffective. Each fault-state, and its corresponding trajectory representation, is fundamentally comprised of error values in three dimensions, as obtained by the probing procedure. The foundation for simulating kinematic faults in this paper is based on the theory of Homogeneous Transformation Matrices (HTMs).

Table 2
Common hyperparameter/characteristics by model.

Hyperparameter	Primary-axis	Secondary-axis
Classes	5	5
Samples per class	10,000	10,000
No. axis indexations	13	7
Axis index range	C0, …, C360	B0, …, B90
Output matrix	3 × 13	3 × 7
Minimum error value	−0.100	−0.100
Maximum error value	0.100	0.100
Mechanical noise parameter (σ_m)	$N(0, 0.02)$	$N(0, 0.02)$
Natural noise parameter (σ_n)	$N(0, 0.003)$	$N(0, 0.003)$

Defining the base frame, O, with origin in the base origin position; the tool, tl, and workpiece, wp, frames are then formed as branches from O. The relative position of tl to wp can be obtained by,

$$^{wp}T_{tl} = {}^{wp}T_O \cdot {}^{O}T_{tl} \tag{1}$$

where ${}^{i}T_j$ is a 4 × 4 HTM defining the position and orientation of frame j in frame i [5,24,25]. In a multi-axis machine tool, each frame in the HTM represents a different kinematic component in the system; the components being synonymous with the linear translational-axes providing motion in the X-, Y- and Z-axis directions, and rotary-axes providing motion in the A-, B- and C-directions. The non-ideal link, D_S, between a frame S and its predecessor, is the product of three transformations,

$$D_S = D_S(\text{nom}) \cdot D_S(E) \cdot D_S(*) \tag{2}$$

where $D_S(\text{nom})$ defines the nominal position and orientation of frame S with respect to the previous, encompassing the physical design dimensions and nominal orientations of the connecting components. $D_S(E)$ defines the errors in the link, conveniently generalised for a rotation axis link [26] as,

$$D_S(E) = \begin{bmatrix} 1 & -\epsilon_z & \epsilon_y & \Delta_x \\ \epsilon_z & 1 & -\epsilon_x & \Delta_y \\ -\epsilon_y & \epsilon_x & 1 & \Delta_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

where Δ_x, Δ_y and Δ_z are the translational errors in the X-, Y- and Z-axes of kinematic component S, and ϵ_x, ϵ_y and ϵ_z its rotational errors, again about the X, Y and Z axes, respectively. $D_S(*)$ defines the commanded change in position of kinematic component S. For the BC rotary table configuration described in Section 2, Eq. (1) can be expanded to give,

$$^{wp}T_O = {}^{wp}q \cdot D_C \cdot D_B \tag{4}$$

$$^{O}T_{tl} = D_Y \cdot D_X \cdot D_Z \tag{5}$$

where ${}^{wp}q$ is the desired position in the workpiece frame given by $[x^*, y^*, z^*, 1]$. Note that the transformation from ${}^{wp}q$ with D_C also incorporates the origin of the workpiece frame, colloquially known as the *work offset*. The coordinate location of the work offset is unique to each workpiece setup, but is inevitably included as a nominal offset in Eq. (2). The length of the tool is similarly incorporated into the transformation D_Z, on the tool-side of the calculation.

In order to produce an informative simulation of error motion trajectory data, it is convenient and appropriate to make a number of assumptions and simplify the above expressions. One can assume zero tool length and work offset; as their values are arbitrary, they have no effect on the motion of the rotary-axes themselves. The probing procedure applied in this paper collects only rotary-axis motions, so it is reasonable to assume that there will be no influence from any linear axis motion x^*, y^* and z^*. Finally, it will be assumed that the kinematic chain is infinitely small, such that there is no nominal offset nom between any of the chain components. The resultant, simplified expression for simulating rotary-axis faults is given by,

$$^{wp}T_{tl} \approx D_C(E_c, c^*) \cdot D_B(E_b, b^*) \tag{6}$$

3.1. Diagnosable faults in error motion trajectory data

The current work considers five potential fault-states that can be identified by a multi-axis spherical artefact probing procedure, described below. For pure kinematic faults — Classes One, Two and Three — the simulated state is obtained by modifying the parameters in Eq. (3). For non-kinematic faults — Classes Four and Five — the state is obtained through a customised equation. Examples of the type of error motion trajectories for each class are illustrated in Fig. 5. For brevity in the trajectory figures, Fig. 6 provides a generic illustration of the formatting.

3.1.1. Class one - The ideal case

This represents the situation in which there is negligible error present in the system, such that all measured errors collected are nominally zero. All parameters in Eq. (3) are set to zero or negligibly small values; example trajectories are provided in Figs. 5(a) and 5(f). In reality, there is likely to be some tolerance bound which will define what the Maintenance Engineer considers as ideal performance. For this paper, the ideal case is simulated with Eq. (6) as a position error (Class Two) which is smaller than 10% of the maximum error value given in Table 2. The trajectory representation is approximated by,

$$^{wp}T_{tl}(1) \approx^{wp} T_{tl}; \quad \Delta_x, \Delta_y, \Delta_z \leq 0.01 \tag{7}$$

3.1.2. Class two - Position error of the rotary-axis average line

There is a translational error in the location of the rotary-axes' origin, causing one or both of the axes to rotate eccentrically. Error parameters Δ_x, Δ_y and Δ_z are modified to obtain this class; examples are illustrated in Figs. 5(b) and 5(g). Position error is likely the most commonly-occurring fault, and generally the easiest to correct, requiring a simple update to the origin offsets held in the controller. Class Two faults are simulated with Eq. (6), such that $^{wp}T_{tl}(2) \approx^{wp} T_{tl}$.

3.1.3. Class three - Orientation error of the rotary-axis average line

There exists the parallelism error between the ideal and actual axis average lines, causing the table/head to tilt. Error parameters ϵ_x, ϵ_y and ϵ_z are modified to obtain this class. Illustrated in Figs. 5(c) and 5(h), orientation errors are generally harder to correct than their positional counterparts, often requiring manual adjustments to the axes themselves. Class Three faults are simulated with Eq. (6), such that $^{wp}T_{tl}(3) \approx^{wp} T_{tl}$.

3.1.4. Class four - Structural/bearing error motion

There is a non-kinematic, mechanical fault with the axis due to faulty bearings or loose component connections, causing it to move erratically and unexpectedly when in use. Figs. 5(d) and 5(i) provide example illustrations. It is necessary to define another equation to cover erratic faults, extending Eq. (6) to,

$$^{wp}T_{tl}(4) \approx^{wp} T_{tl} \cdot \begin{bmatrix} 1 & 0 & 0 & \sigma_{m,x} \\ 0 & 1 & 0 & \sigma_{m,y} \\ 0 & 0 & 1 & \sigma_{m,z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

where $\sigma_{m,x}$, $\sigma_{m,y}$ and $\sigma_{m,z}$ are randomly-generated scalar values, applied independently to the X, Y and Z coordinate points to simulate the erraticism effect induced by a mechanical fault. In order to account for natural variation in the real process, an additional set of randomly-generated scalars – $\sigma_{n,x}$, $\sigma_{n,y}$ and $\sigma_{n,z}$ – was also applied to the outputs of all five simulated classes, in the same manner as for Eq. (8).

3.1.5. Class five - Controller compensation/scale reader issues

Illustrated in Figs. 5(e) and 5(j), these types of error motion trajectory are generally attributed to software faults. Controllers may include compensation software to account for factors such as thermal expansion or deformation due to static loading, of the structural loop. The scale reader is a measurement system which tracks the physical axis positions and converts them into a readable, coordinate system. Faults attributed to both of these sources have been observed, in some cases, to induce straight lines and square-shaped trajectory profiles. A new equation is again necessary to define this class, approximated by,

$$^{wp}T_{tl}(5) \approx \begin{cases} m \cdot s \cdot \text{sgn}(\sin(\theta^* + \alpha)) & \text{if } \theta^* < 90^o \text{ or } 180^o \leq \theta^* < 270^o \\ m^{-1} \cdot s \cdot \text{sgn}(\sin(\theta^* + \alpha)) & \text{if } 90^o \leq \theta^* < 180^o \text{ or } \theta^* \geq 270^o \end{cases} \tag{9}$$

where m is a free gradient parameter defining the rotation of the trajectory, s is the step size between each point, and θ^* is the commanded rotary axis position, in the B- or C-axis. An additional small fixed parameter α is included, preventing the signum function at a rotary-axis position of $\theta^* = 0$ from returning a zero value.

3.2. Simulation methodology

Datasets for model training and validation were simulated for the five classes described above, for Primary-axis and Secondary-axis probing procedures. Table 2 provides a summary of the dataset characteristics. The sample output matrix size was determined by the number of axis indexations, which is typically covered by each probing procedure.

The limits for minimum and maximum error values were determined by domain knowledge of the accuracy levels typically observed on modern multi-axis machine tools. Note that, however, the overall magnitude of the error only affects the diagnosis of the ideal case (Class One); the fault states (Classes Two — Five) are entirely defined by the shape, or *profile*, of the trajectory, providing that the magnitude falls above the threshold for the ideal case. As such, the limits definition serves to generate a suitable range of profiles which describe a variety of possible fault state permutations. By defining the absolute magnitude of the trajectory as arbitrary, it is then viable to project extreme cases of experimental data – where the magnitude of the error exceeds the typically-observed range – into the same standardised space, to be processed by the classification algorithm and correlated with the simulated examples. The severity of an individual fault can then be trivially-quantified by extracting the magnitude of the trajectory alongside

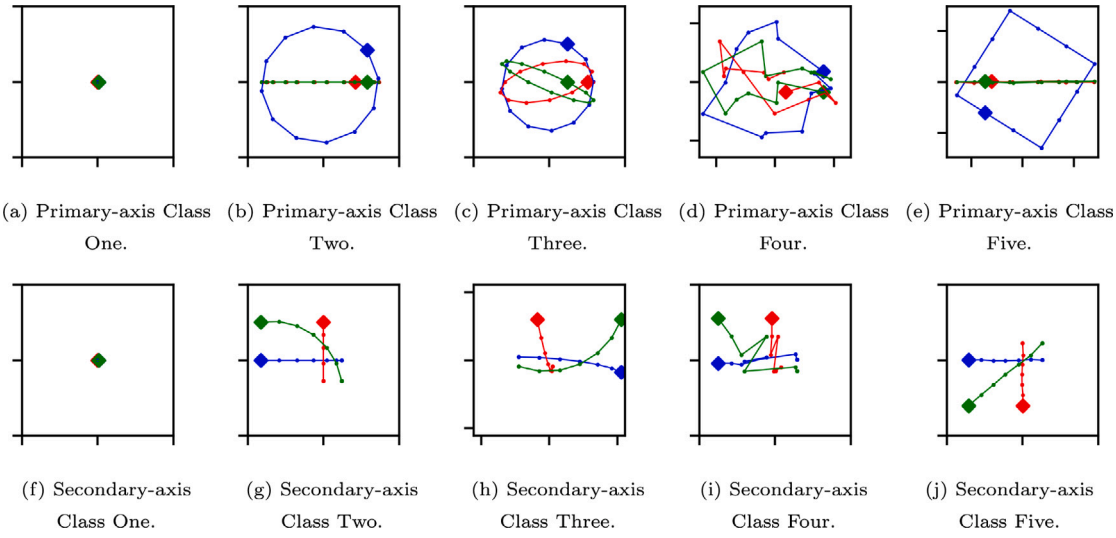


Fig. 5. Error motion trajectory examples for Primary-axis 5(a)–5(e) and Secondary-axis 5(f)–5(j) classes.

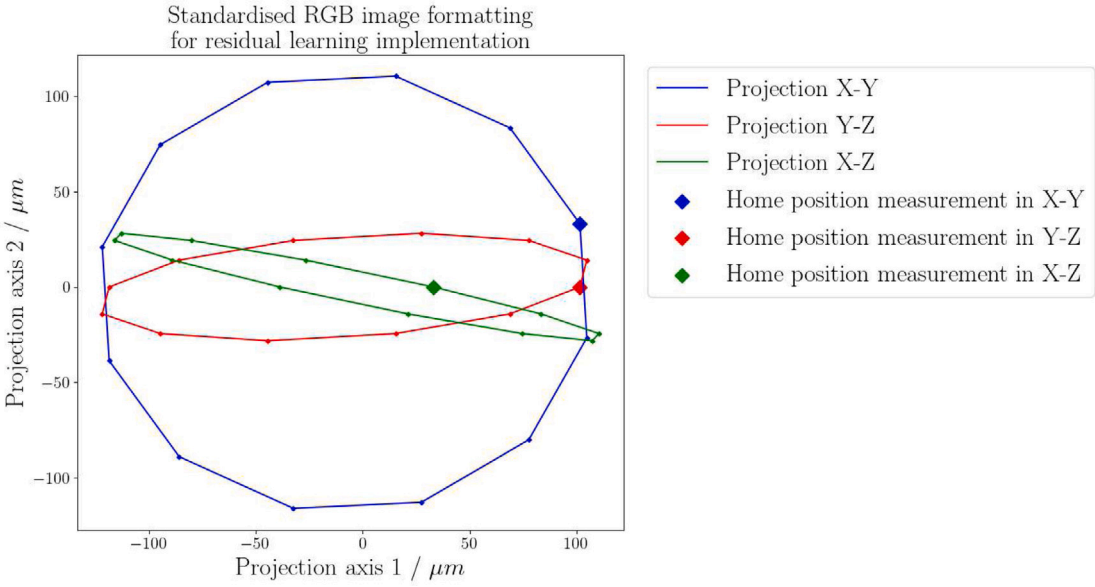


Fig. 6. Diagrammatic description of the error motion trajectory images presented in this paper, with a Primary-axis Class Three trajectory as an example. The three-dimensional trajectory acquired by the measurement procedure is reduced to a two-dimensional representation, by taking three projections in the X–Y, Y–Z and X–Z planes. The three projections can then be plotted in the same space to obtain a single, two-dimensional figure. Note that the magnitude of the trajectory is irrelevant for the classification algorithm, all critical information is portrayed by the trajectory profile.

the classification. This step is not presented in the current work, however it would be a straightforward and useful addition to the final system deployment.

The number of samples per class processed for training a deep learning algorithm has a direct impact on the generalisation performance, which is ultimately determined by the number of weight parameters in the model. That said, Goodfellow, Bengio and Courville [27] recently proposed a total of 5,000 samples per class to achieve acceptable performance, as a general rule-of-thumb for the current generation of deep learning algorithms. For this paper, a total of 10,000 samples were generated per class, comfortably exceeding this rule-of-thumb.

Tables 3 and 4 provide summaries of the simulation characteristics and specific parameters for Primary and Secondary-axis systems, respectively. A noise parameter with a variance of 3 μm was applied to all matrix elements to simulate small measurement system/environmental variations, and a further noise parameter with a variance of 20 μm applied to simulate more extreme variation in the structural/bearing error motions (Class Four). All samples were normalised about their average values.

Table 3
Specific hyperparameter/characteristics by class, for Primary-axis classifiers.

	Class One	Class Two	Class Three	Class Four	Class Five
Kinematic model	True	True	True	True	False
Equation	(6), (3)	(6), (3)	(6), (3)	(6), (3)	(9)
Free parameters	Δ_x, Δ_y	Δ_x, Δ_y	Δ_x, Δ_y ϵ_x, ϵ_y	Δ_x, Δ_y	m, s
Parameter range (kinematic)	0 : 0.009	0.011 : 0.1	0.011 : 0.1	0.011 : 0.1	N/A
Noise parameters	Natural	Natural	Natural	Mechanical, natural	Natural

Table 4
Specific hyperparameter/characteristics by class, for Secondary-axis classifiers.

	Class One	Class Two	Class Three	Class Four	Class Five
Kinematic model	True	True	True	True	False
Equation	(6), (3)	(6), (3)	(6), (3)	(6), (3)	(9)
Free parameters	Δ_x, Δ_z	Δ_x, Δ_z	Δ_x, Δ_z ϵ_y, ϵ_z	Δ_x, Δ_z	m, s
Parameter range (kinematic)	0 : 0.009	0.011 : 0.1	0.011 : 0.1	0.011 : 0.1	N/A
Noise parameters	Natural	Natural	Natural	Mechanical, natural	Natural

3.3. Feature engineering

The numerical outputs of the simulation were then processed into images and feature vectors, for two separate training datasets. Image format was standardised as illustrated in Fig. 5, and saved as 512 × 512 RGB images to match the input of the pre-trained residual learning model [28]. An *ad hoc* sixteen element feature vector was calculated for each sample, set by domain knowledge derived from the manual diagnostic approach currently employed by the specialist engineer.

Fig. 7 illustrates the elements of the engineered feature set. Element one evaluates whether the sample is considered ideal or not, based on the criteria stated in Eq. (7). Element two returns an angle from the trajectory's centre-point, to the first point collected (at the home position). This metric indicates which translational error parameter is causing a kinematic eccentric fault. For example, in a Primary-axis trajectory, this would differentiate between Δ_x, Δ_y or both Δ_x and Δ_y simultaneously. This case will not be directly targeted by this paper, but is a worthwhile inclusion to the dataset for future research. Elements three and four give base indications as to the magnitude of a potential position error.

Elements five and six give the radius and residuals, respectively, of a least-squares estimation on the potential eccentricity seen in the major axis projection (referring to Figs. 5(b) and 5(g), the major axis projection for a Primary-axis trajectory is blue, and for a Secondary-axis map is green). Naturally, the former indicates the presence/extent of a position error, and the latter gives an indication of the likelihood that the circle is influenced by some mechanical error.

Elements seven through twelve are the result of a least-squares estimation on the ellipses by the two minor axis projections (clearest as the green and red ellipses in Fig. 6) in the presence of a position error. Values are given for the total width, height and angle of the ellipse for each projection.

Elements thirteen through sixteen score the error motion trajectory characteristics by calculating the gradient between two rows in the coordinate matrix, and comparing with the gradient between the previous two rows. This produces four output metrics. Element thirteen quantifies the amount of straight lines that are visible in the trajectory, which is indicative of controller compensation/scale reader errors, or Class Five. Element fourteen evaluates the circularity of the trajectory; this relates to the occurrence of a kinematic error, which is a characteristic of both Class Two and Class Three faults. Element fifteen evaluates the presence of corners, indicating the likelihood of controller compensation errors specifically, defined by Class Five. Finally, element sixteen quantifies the erraticity, or the amount of (seemingly) random activity in the trajectory, which is an indicator of structural/bearing error motions, defined by Class Four.

4. Identifying faults with supervised learning

Supervised learning is the process of inferring a function which maps input objects to output values, in which input–output pairs are passed to an algorithm as examples where the output values are known during an initial training phase. For a classification problem, the process involves constructing a mapping, $f : \mathcal{X} \rightarrow C$, from some D-dimensional feature space, $\mathcal{X} \in \mathbb{R}^D$, to a ground-truth class label space, $C = \{1, 2, \ldots, K\}$, where K denotes the total number of classes. After construction in the training phase, the mapping f can be reused to predict the class labels of future, unseen input objects, in a testing phase. For the nth data point, where the input feature vector is denoted by $\mathbf{x}_n \in \mathcal{X}$, and a corresponding descriptive class label, $c_n \in C$, the training dataset of example pairs can be represented as,

$$\mathcal{D} = \{(\mathbf{x}_n, c_n) | \mathbf{x}_n \in \mathcal{X}, c_n \in C\}_{n=1}^{N} \tag{10}$$

for a total of N training examples. Research concerning supervised learning is mature and there are many different approaches that can be – and have been – applied [29] in the context of identifying faults in a machine tool system. Certain factors must be considered in selecting an appropriate model for a particular problem; such as the size of the dataset available for training, requirements for a

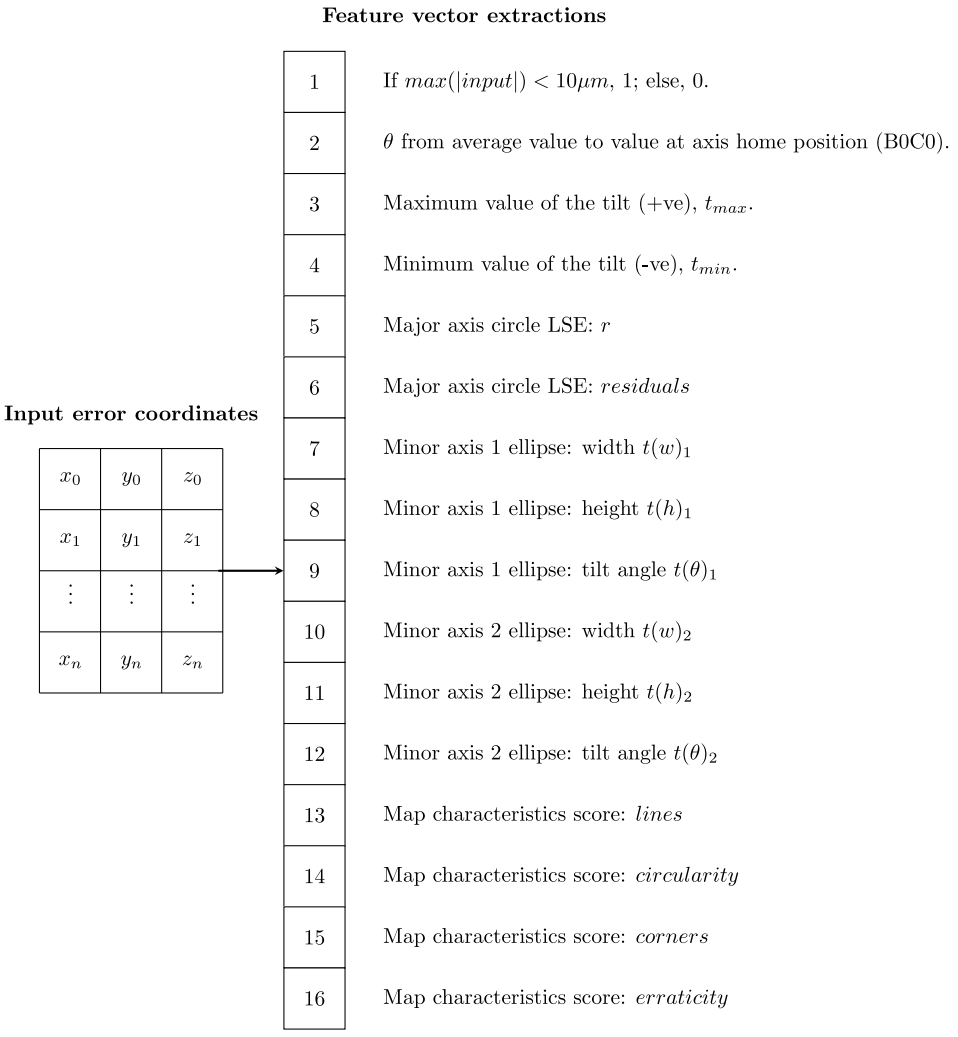


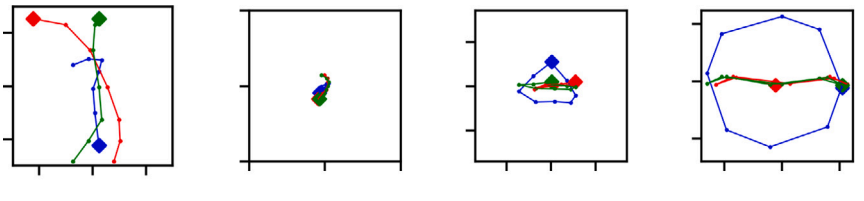
Fig. 7. *Ad hoc* feature engineering.

deterministic or probabilistic output, data type of the input objects or format of the output values (classification, regression, feature space encoding, etc.), or general computational budget, may all influence the initial model selection. In this paper, the Artificial Neural Network (ANN) and Convolutional Neural Network (CNN) were explored as base models to perform diagnostics in error motion trajectory data. Descriptions of these machine learning models are provided in Appendices A and B, the reader may also refer to [27] or [30] for more detailed descriptions of the theory.

4.1. Ensemble learning

A strong classifier is defined as one which achieves good performance when predicting the labels of future, unseen data. In order to do this, it must avoid overfitting in the training phase and be able to generalise the information learned to data in the unseen test domain. It is often hard to build a single classifier with high generalisation ability. Ensemble approaches provide a potential solution to this, by combining the predictions from a set of weak classifiers with the hope of strengthening the performance through the power of the collective. Theoretical analysis and real practice have shown that ensemble models generally result in smaller expected errors than those obtained by a single model [31].

An ensemble is defined by a collection of *base learners*, which work independently but towards a common goal. In this paper, the proposed ensemble models are compared to a multi-class (*MC*) classifier, defining the baseline performance of a single learner that maps the input object to all K classes contained in the training set. Let $y_{n,k}$ be the probability, $P(c_{n,k}|\mathbf{x}_n)$, that class label $c_n = k$, is obtained when passing input object $\mathbf{x}_n$ to an *MC* classifier with a softmax activation function on the output layer. The master



(a) Secondary-axis example one. (b) Secondary-axis example two. (c) Primary-axis example one. (d) Primary-axis example two.

Fig. 8. Examples of error motion trajectories, collected from operational machines, which do not strictly conform to the theoretical class definitions.

matrix containing all output probabilities for n training examples and K classes can then be defined by,

$$\mathbf{Y} = (y_{n,k}) \in \mathbb{R}^{n \times K} \tag{11}$$

The predicted class, $\hat{c}_n$, for the nth example is found by selecting the class with the highest probability score,

$$\hat{c}_n = \arg\max_k \mathbf{Y}_{n,k} \tag{12}$$

such that, for a correct classification, $\hat{c}_n$ is equal to the ground-truth class label, c_n. Although the *MC* baseline approach is simple in comparison with a base learner collection in an ensemble model, the reality is that it must deal with the full complexity of the data domain within a single classifier. Considering that the main objective in this paper is concerned with training in a simulated environment, it is likely that generalisation performance on real-world data will be poor. To attempt to counter this, implementing an ensemble-based approach may be helpful in producing a stronger overall classifier which has a better ability to generalise.

The complexity of the prediction required for a given base learner can be reduced by employing a *divide and conquer* [32] approach; in this case, with a focus on dividing the class label space into smaller and/or easier-to-learn partitions. A One-Vs-Rest (*OVR*) scheme is defined where the learner is tasked with predicting if a given example belongs to a single target class, or any of the other class. The approach firstly constructs a set of K base learners. Each learner is assigned a target label k, and attempts a binary classification of k versus all other labels, $\bar{k}$, on a given input object $\mathbf{x}_n$. The softmax function in Eq. (A.4) is applied across the base learner output to obtain $P(c_{n,k}|\mathbf{x}_n)$ and $P(c_{n,\bar{k}}|\mathbf{x}_n)$. The probability $P(c_{n,k}|\mathbf{x}_n)$ can then be used to populate element $\mathbf{Y}_{n,k}$ of the master output matrix, repeating the process across all i training examples and k base learners to fill the output matrix. Probability distributions for the nth row of $\mathbf{Y}$ can then be found by an application of Eq. (A.4) across all K classes, and class label predictions determined with Eq. (12). In comparison with the *MC* approach, an *OVR* ensemble is computationally more-expensive, due to the need to train K *OVR* base learners versus the single base learner in an *MC* ensemble, with the same number of training examples.

Prediction complexity can be reduced further with a One-Vs-One (*OVO*) approach, whereby a base learner is trained to identify whether an example belongs to one of two classes only. This is particularly useful if there are two or more classes in $\mathcal{D}$ which are very similar, as it allows the relevant base learner to focus on their specific separation without having to also build representations for all the other classes. The generic *OVO* base learner is constructed to identify class k from class k', and the probabilities for $P(c_{n,k}|\mathbf{x}_n)$ and $P(c_{n,k'}|\mathbf{x}_n)$ are obtained in much the same way as in the *OVR* approach. Both $P(c_{n,k}|\mathbf{x}_n)$ and $P(c_{n,k'}|\mathbf{x}_n)$ are then used to populated elements $\mathbf{Y}_{n,k}$ and $\mathbf{Y}_{n,k'}$ in the master output matrix. Again, in the same way as the *OVR* approach, probability distributions across all K classes can be found with Eq. (A.4) and class label predictions with Eq. (12). A total of $\frac{K^2-K}{2}$ base learners are required for a full *OVO* approach, after applying constraints $k \neq k'$ and $k < k'$, to disallow invalid or repeated learners, respectively. As an individual *OVO* base learner only needs to learn to differentiate between two classes, it is also not necessary to process the entirety of the dataset $\mathcal{D}$ - only a subset of training examples, where $c_n = k$ or $c_n = k'$, are required. This reduces the computational cost of the approach proportionally with the number of classes, K, offsetting the relative increase attributed to the higher number of base learners required.

Separate models were constructed for both feature/image input data types and Primary/Secondary-axis trajectory data, for a total of twelve ensemble classifiers trained on solely simulated examples. In Section 1, the difficulty in obtaining a dataset representative of all possible fault-states identifiable through trajectory data was discussed, due mostly to the offline nature of the procedure and sparse frequency of the data collection intervals. This difficulty leads to the general motivation for building a classifier based on simulated faults, with the objective being to encode the theoretical bases which define them, even if specific examples are lacking in the experimental data available. However, real systems do not often behave exactly as expected, and different machines tend to have certain signatures which set them apart from their peers. For this reason, faults may regularly appear in a real-world setting which do not necessarily conform to the theoretical definition, but are still diagnosable through the experience of the specialist engineer interpreting the data.

4.2. Transfer learning

Fig. 8 illustrates four examples of error motion trajectories collected from actively-operational machine tools which demonstrate some non-conformity to the theoretical fault-state definitions. Each example in Fig. 8 can be assigned a class through expert

knowledge and engineering judgement; the ensemble models trained on a simulation, however, are not likely to possess this depth of ability, and are likely to struggle in the face of certain real-world examples. There is an argument for incorporating additional domain knowledge from operational machine tools to supplement the theoretical definitions; however, with a typical machine learning approach, this could intrinsically bias future predictions to the signatures of the machines included in the test set, and thus may not be useful for other machines with different signatures.

The typical approach in a machine learning application is to optimise model parameters on a large quantity of training data, fix those parameters, and make predictions on future data based on the information learned in the initial training phase. In order for this to be effective, the distribution of future testing data must be well-represented in the training set. However, as has been discussed, this prerequisite is not necessarily met in this particular application domain.

Transfer learning broadly refers to the process of storing knowledge gained on one problem domain, and applying that knowledge to a new, but related, problem domain. In deep learning, implementations most often take the form of either dedicated *feature extractors* or *fine-tuning* (*FT*) procedures. In a feature extractor, early network layers are frozen, and mature knowledge on low-level feature extraction can be exploited on the new domain by either simply updating the later, unfrozen parameters, or changing the unfrozen network structure entirely to apply the model to an alternative task (such as a different set of target classes). Fine-tuning describes the process of allowing network parameters to be updated with data from the new domain, incrementally altering the latent space to increase relevance on the newly-observed training data. The process can be applied either to the entire network, or to selected unfrozen layers in a feature extractor implementation.

A known complication with fine-tuning is defined by the stability–plasticity dilemma [33]. In an ideal implementation, the representation built during learning should be stable-enough to retain previous knowledge, but also plastic-enough to adapt to new information when provided. Neural networks are particularly prone to the dilemma, most often demonstrating low stability with high plasticity, leading to Catastrophic Forgetting (CF) [34] of old information when re-trained with newly-acquired data. Replay of previously-learned examples has been posed as a solution for minimising the detrimental effect of CF, with notable recent successes using generative [35] and pseudorehearsal-based [36] approaches. Where feasible, replay with a subset of the actual original examples is ideal, in a process known simply as *rehearsal*. Consider the dataset of length n used for the initial model training, $\mathcal{D}_1$, with the same form as defined in Eq. (10). Following a period of time after the initial training, a second dataset, $\mathcal{D}_2$, of length n_2 is obtained for transfer learning, such that,

$$\mathcal{D}_2 = \{(\mathbf{x}_n, c_n) | \mathbf{x}_n \in \mathcal{X}, c_n \in C\}_{i=1}^{n_2} \tag{13}$$

The objective with re-training is to incorporate the information contained within $\mathcal{D}_2$, whilst preserving the information initially learned from $\mathcal{D}_1$. In order to achieve this, and circumvent the possible occurrence of CF, a rehearsal set, $\mathcal{D}_R$, of length r can be sampled from $\mathcal{D}_1$, such that,

$$\mathcal{D}_R \subseteq \mathcal{D}_1 \tag{14}$$

$\mathcal{D}_R$ can then be combined with $\mathcal{D}_2$ to construct a model re-training dataset, $\mathcal{D}_{FT}$, which allows new examples to be fed to the learner whilst simultaneously reinforcing some of the originally learned information. Mathematically, the set is constructed by,

$$\mathcal{D}_{FT} = \mathcal{D}_R \cup \mathcal{D}_2 \tag{15}$$

where the cardinality of $\mathcal{D}_{FT}$ is equal to $n_2 + r$. To improve generalisation ability on real-world trajectory data examples, it is proposed to incorporate a subset of experimental examples with the above method, to implement a fine-tuning procedure. Practically, this would allow the underlying, theoretical fault-state information to be initially encoded, followed by periodic updates during the system's deployment to encode any machine-specific signature effects via fine-tuning. Having instant access to a classification system, but one which can also dynamically improve when applied to new data, would clearly be of great value to the industrial user.

4.3. Modelling methodology

This section now expands on the research route illustrated in Fig. 1, to describe the modelling methodology in detail. Firstly, a training set was simulated as described in Section 3, producing two datasets; one set comprised of engineered feature vectors, and one of raw RGB images. The data in each set were split into training/validation/test subsets at ratios of 70/20/10%, respectively, and passed to the various ensemble models illustrated in Fig. 1. All networks were trained in a standardised way over twenty-five epochs, passed in normalised mini-batches of four for each propagation through the network. Optimum network parameters were learned through stochastic gradient descent with a learning rate, ∇, of 0.001 and momentum of 0.9 [30]. Weighted random oversampling [37] was implemented in the dataloader, to offset the risk of bias associated with the class imbalance introduced in the *OVR* ensemble models.

Models to process the engineered feature vectors, referenced by *ANN*, were constructed with fully-connected artificial neural networks. Network architectures were found through a Tabu Search [38] optimisation procedure across a search range of two, three and four layer networks. Overall architecture sizes were constrained to contain a maximum number of parameters no higher than the number of samples available for training, avoiding overly-complex networks which could cause the model to exhibit poor generalisation. Dropout with a probability $P(0.5)$ was applied to the hidden layers to further strengthen the likelihood of high generalisation [39]. Each potential solution was allowed to train for ten epochs and the procedure stopped when a solution achieved 95% or higher validation accuracy in this period. The best architecture was then selected and re-trained in the standardised procedure described above.

Table 5
Total number of examples per class in the two experimental test sets. Note that - for fine-tuning - each class had an additional two examples held out in the Primary-axis dataset, and three examples in the Secondary-axis dataset.

Probing procedure type	No. examples per class				
	Class One	Class Two	Class Three	Class Four	Class Five
Primary-axis	34(+2)	62(+2)	8(+2)	17(+2)	14(+2)
Secondary-axis	43(+3)	2(+3)	203(+3)	47(+3)	4(+3)

Models to process raw RGB images, referenced by *CNN*, were constructed with an eighteen-layer residual network. The model architecture was adopted from [28], and transformations applied to match those in the original paper. Specifically, this involved a randomised horizontal and/or vertical flip, and colour jitter/pixel intensity normalisation about the same values used to initially train the residual network. A randomised resize and crop transformation which was present in the original paper was omitted from this implementation; as the images produced from error motion trajectory data were likely to contain a large amount of whitespace, there was a notable risk that the resultant images could be cropped into blank samples, hindering convergence significantly. The resize/crop transformation was emulated instead by the variation in error parameter magnitudes, implemented in the fault-state simulation to produce the training dataset. Pre-conditioned weights [28] (on the ImageNet database) were available for this network architecture, and were utilised as initial parameter values, to exploit the basic feature detection filters learned for the early convolutional layers.

A fine-tuning procedure was implemented to allow a classifier to dynamically improve with time upon the acquisition of data from actively-operating machines, as-and-when that data becomes available. Firstly, a classification model was trained on simulated data only, encoding the theoretical fault-state definitions. In the ensemble models considered in this paper, it is expected that the *OVO* model should result in the strongest overall classifier, being the one with the simplest base learners and pertaining to the general thesis [31] that ensemble methods boost performance. For this reason, the *OVO* model was selected for re-training with an *FT* approach. A subset of roughly 5% of the total number of examples was held out in each of the experimental test sets, with an even distribution of classes within each subset. In total, 10 examples were separated from the Primary-axis experimental set, and 15 examples were separated from the Secondary-axis set. Each example was augmented 100 times, varying the scale and rotation to produce an enhanced training set. The *FT* re-training set was then supplemented with 400 examples per class from the original, simulated training set, to implement rehearsal and mitigate the incidence of CF.

The process described above was repeated for each ensemble model illustrated in Fig. 1, for both Primary and Secondary-axis probing procedures. Model performances were then evaluated on the test sets and ranked for comparison.

4.4. Model evaluation

Two different test sets were applied to evaluate the model performance. Firstly, model training and validation was benchmarked on a 10% hold-out test set from the simulated dataset. Performance on this test set is important to establish a model's core ability to differentiate between the theoretical definitions of each fault-state, as well as identify any occurrence of CF in the *FT* implementations. In reality, however, identifying fault-states in a real system is more difficult. Influences of compounded fault-states may be clearly visible in a single trajectory, or there may be more more subtle effects from a second state or machine-specific nuance which alters the main state beyond the conventionally-accepted definition.

For the experimentally-obtained data, two examples per class in the Primary-axis dataset, and three examples per class in the Secondary-axis dataset, were held out for *FT* model re-training, with the remaining examples used for model evaluation. This hold-out step was applied prior to any model evaluations, resulting in a total of 134 Primary-axis examples and 299 Secondary-axis examples passed through each model to obtain the results presented in the following section. Examples were processed into feature vectors and RGB images in the same procedure applied for the simulated data, and class labels were manually determined as the most prominent class visible in the trajectory. Table 5 details the distribution of class labels across the two experimental test sets.

Model performances were then compared in detail using a variety of different evaluation metrics. The probability of correct classification, otherwise referred to as the *classification rate*, $P(\hat{\mathbf{c}} = \mathbf{c})$, is a common and easy-to-interpret metric for evaluating and comparing classifiers, calculated by,

$$P(\hat{\mathbf{c}} = \mathbf{c}) = \frac{TP + TN}{TP + TN + FP + FN} \tag{16}$$

where TP, TN, FP and FN are the *true positive*, *true negative*, *false positive* and *false negative* classifications, respectively. As classification rate is widely known and understood, it is used in this paper as the headline metric. For a more-comprehensive comparison, this paper also explores alternatives. *Precision* measures the relevancy of the results, by,

$$Precision = \frac{TP}{TP + FP} \tag{17}$$

Recall measures how many of the truly relevant results are returned, by,

$$Recall = \frac{TP}{TP + FN} \tag{18}$$

Table 6
Optimised architectures with validation set fitness, obtained with Tabu Search [38].

Model	Target classes	Primary-axis		Secondary-axis	
		Architecture	Fitness	Architecture	Fitness
MC_{ANN}	0, 1, 2, 3, 4	[32, 17, 36]	0.9565	[85, 51, 52, 71]	0.9745
OVR_{ANN}	0	[10, 11, 11]	0.9998	[11, 26]	1.000
	1	[11, 9]	0.9932	[9, 9, 10, 9]	0.9981
	2	[11, 10, 9, 9]	0.9978	[9, 9, 10, 9]	0.9924
	3	[9, 9, 10, 9]	0.9843	[11, 9]	0.9791
	4	[11, 9]	0.9989	[10, 11, 11]	0.9964
OVO_{ANN}	0, 1	[10, 11, 11]	0.9998	[11, 26]	1.000
	0, 2	[10, 11, 11]	0.9998	[11, 26]	1.000
	0, 3	[10, 11, 11]	0.9998	[11, 26]	1.000
	0, 4	[10, 11, 11]	0.9998	[11, 26]	1.000
	1, 2	[11, 9]	0.9932	[9, 9, 10, 9]	0.9981
	1, 3	[11, 9]	0.9932	[9, 9, 10, 9]	0.9981
	1, 4	[11, 9]	0.9932	[9, 9, 10, 9]	0.9981
	2, 3	[11, 10, 9, 9]	0.9978	[9, 9, 10, 9]	0.9924
	2, 4	[11, 10, 9, 9]	0.9978	[9, 9, 10, 9]	0.9924
	3, 4	[9, 10, 9, 9]	0.9843	[11, 9]	0.9791
FT_{ANN}	*As for* OVO_{ANN}				

The F_1-*score*, calculates the harmonic mean of precision and recall, allowing both metrics to be accounted for when comparing classifier models. It is given by,

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{19}$$

An alternative means of evaluating a classifier based on both precision and recall is to construct a Precision–Recall (PR) curve. Commonly, one might expect to see a Receiver Operating Characteristic curve at this point, however there has been recent debate over their application as metrics to problems with imbalanced datasets [40]. For this paper, the intention is to present a curve describing the overall classifier accuracy across all classes; in order to do this, the outputs must be binarised through micro-averaging, resulting in a significant imbalance with more TN results than TP. A PR curve, which ignores the TN values in favour of TP, is resilient to this issue and more appropriate metric for comparison. The PR curve shows the trade-off between precision and recall for a range of different thresholds in the classifier's output probability. The area under the PR curve (AUPRC) is a useful metric, with high values indicating that the classifier has both high recall and precision, and is confident in its predictions.

The simplified metrics and equations above relate specifically to binary classifiers. For the multi-class form, classification rate is straightforwardly extended by summing the diagonal elements of the $K \times K$ confusion matrix to obtain the numerator and all elements in the matrix for the denominator. Precision, recall and F_1-score metrics per-class are obtained by querying the relevant column and row to obtain TP, FP and FN values for each class.

5. Results & discussion

5.1. Architecture optimisation

Table 6 presents the results of the Tabu Search architecture optimisation procedure applied to the feature-based *ANN* ensemble models. As the *FT* model directly employs the *OVO* architecture, no search procedure was required. Owing to the stopping criteria set at 95%, all architectures scored highly, with a majority returning scores of 99% or more. The multi-class MC_{ANN} model returned the lowest scores for both Primary- and Secondary-axis procedures, suggesting that the base learners in the *OVR* and *OVO* ensembles are stronger, for their respective classification tasks at least.

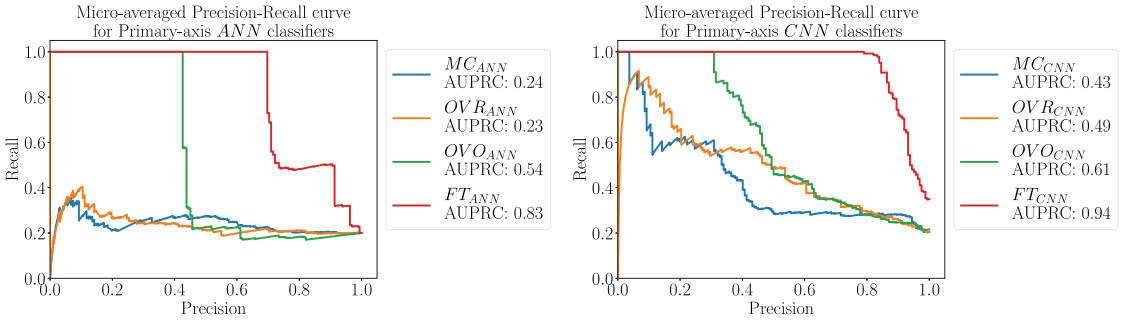
5.2. Primary-axis procedure

Table 7 presents the overall probability of correct classification, for all ensemble models trained to identify Primary-axis faults. For all classification models, results on the simulation test set are excellent, suggesting that all models were able to successfully learn the theoretical fault-state definitions. This is logical; although free parameters were included to introduce noise to the simulation, the dataset is inherently very clean, and the fault-states for each class are clearly defined. It is not entirely surprising that the classifiers perform well on the simulation hold-out test set, as the examples contained within it, although different, are derived from the same source.

Unsurprisingly, the experimental test set results are generally lower than those on the simulated test set. They are however, much more relevant, as generalisation to data obtained from operational machine tools is the core objective in this paper. *MC* models performed poorly for both *ANN* and *CNN* approaches, achieving classification rates of around 20%, equating roughly with a no-skill, random-guess classifier. As complexity in the network output is reduced when implementing an *OVR* ensemble approach,

Table 7
Comparison of Primary-axis classifier skill levels by probability of correct classification, with highest results for simulation and experimental test sets highlighted.

	Probability of correct classification $P(\hat{\mathbf{c}} = \mathbf{c})$	
Model	Simulation test set	Experimental test set
MC_{ANN}	0.9624	0.2074
OVR_{ANN}	0.9636	0.2593
OVO_{ANN}	0.9926	0.3630
FT_{ANN}	**0.9996**	0.7037
MC_{CNN}	0.9788	0.2963
OVR_{CNN}	0.9728	0.4074
OVO_{CNN}	0.9984	0.5185
FT_{CNN}	0.9990	**0.8444**



(a) Feature-based, *ANN* models. (b) Image-based, *CNN* models.

Fig. 9. Precision–recall curve for Primary-axis classifiers, with values for the area under the PR curve (AUPRC).

classifier skill-level increases for both *ANN* and *CNN*. The same is true for the *OVO* ensembles, which indicate further improvement to the classification rates. The results for the *FT* approaches indicate a further, significant performance boost, resulting in almost 85% of examples in the experimental test set being correctly identified for the best case, FT_{CNN} model. Observing the *FT* model results on the simulation test set, it is clear that the predictors have not been affected by any instance of CF, confirming that the rehearsal procedure implemented for this paper was successful. In fact, the *FT* results on the simulation test set indicate marginal improvements when compared with the *OVO* results (*OVO* provides a performance snapshot of the *FT* model before it is re-trained), suggesting that the rehearsal procedure could likely have been less intensive and still achieve the desired effect.

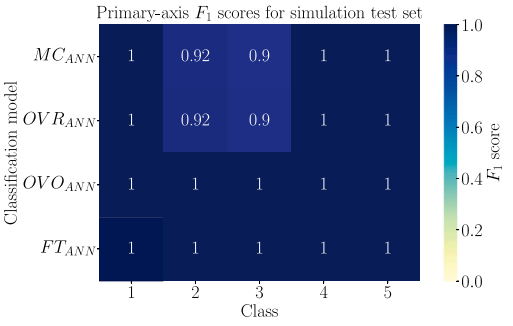
Comparing the feature-based *ANN* and image-based *CNN* models in Table 7, it is evident that the *CNN* models tend to produce more accurate predictors. Both exhibit the same trend of improved performance through ensembling and finally fine-tuning, however the classification rates for all *CNN* models are consistently higher than their *ANN* counterparts. This is largely due to their much higher number of parameters, which mean that much more intricate and complex feature spaces can be modelled, as well as the mature low-level feature detectors that were obtainable by loading a model with preconditioned weights. The *CNN* approaches are inevitably more computationally expensive than the corresponding *ANNs*, and the classification rate of 70% observed in FT_{ANN} is commendable for such a lightweight approach.

5.2.1. *Primary-axis — precision–recall curve*

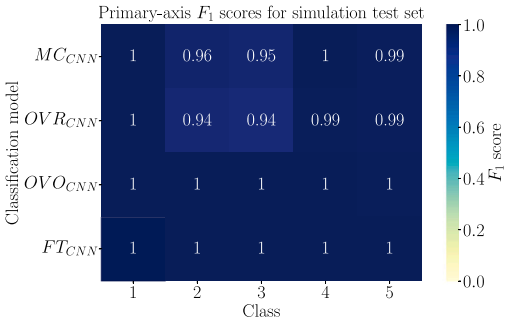
Figs. 9(a) and 9(b) show the micro-averaged PR curves, for Primary-axis *ANN* and *CNN* approaches, respectively. AUPRC results are also reported for each curve, as an additional comparator. Only PR curves for model evaluation on the experimental test set are presented, as it is clear from Figs. 10(a) and 10(b) that all models exhibit perfect or near-perfect performance on the simulated test set, there would be minimal insight to be gained from inspecting their PR profiles.

Viewed in conjunction with the probability of correct classification results in Table 7, the results corroborate the indication of a general improvement through application of ensemble approaches, and finally fine-tuning. Fig. 9(a) confirms that the MC_{ANN} and OVR_{ANN} models produce predictions which are not much better than a random guess, with low recall for all thresholds and AUPRC values of 0.25. OVO_{ANN} improves on this significantly, although the AUPRC is still quite low at 0.48. Re-training in FT_{ANN} significantly improves performance, returning an AUPRC of 0.82 and correspondingly higher quality of predictions when maintaining high recall.

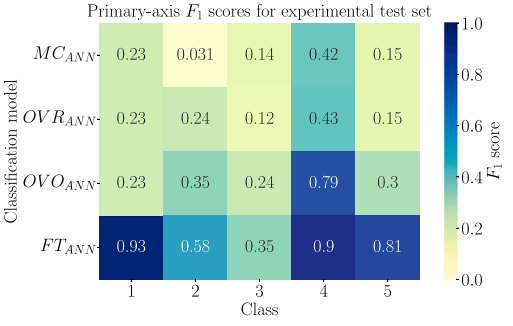
Fig. 9(b) indicates that the *CNN*-based classification models perform better than the *ANN*-based models in every case. Compared with the *ANN* approaches, the general trend of improvement from MC_{CNN} to FT_{CNN} is more obvious, with a clear step up from each method visible in the AUPRC results, in general agreement with the probability of correct classification results in Table 7.



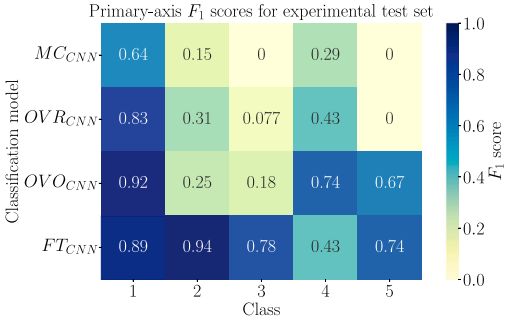
(a) Feature-based *ANNs*, simulated hold-out test data.



(b) Image-based *CNNs*, simulated hold-out test data.



(c) Feature-based *ANNs*, experimental test data.



(d) Image-based *CNNs*, experimental test data.

Fig. 10. *F_1-score* by class, for Primary-axis classifiers.

5.2.2. *Primary-axis – F_1-score by class*

Figs. 10(a) and 10(b) combine precision and recall to show the F_1-score results by class, for the Primary-axis assessment on the simulation hold-out test set. F_1 results for all models, and across every class, are excellent. The result confirms the classifiers successfully fit to the theoretical domain for all fault-states. Comparing the classification models themselves, the strengthening of ability by application of an ensemble is evident through the increase in performance from the *MC* and *OVR* models to the *OVO* model. Although the minimum F_1 performance for *MC* and *OVR* models is high (both 0.92 for *ANN* networks Class Two), constructing an ensemble of *OVO* base learners demonstrates an improvement to a perfect classifier, when evaluated on this particular test set. Contrastingly, however, the implementation of *OVR* base learners for the *CNN* shows a slight degradation in performance, when compared with its associated *MC* model. This degradation is not reflected in the *ANN* approaches, although there is no notable improvement, as the *OVR* model demonstrates identical performance to the *MC* model.

The *FT* model results show no degradation whatsoever when compared to their baseline models (*OVO*), for every class, confirming that there are no instances of CF affecting the predictors.

Figs. 10(c) and 10(d) show the F_1-score results by class, for the Primary-axis assessment on the experimentally-obtained test set. A general improvement is observed in the progression from *MC* to *FT* models, for both *ANN* and *CNN* approaches.

Comparing *ANN* and *CNN* approaches by class, there are some cases which exhibit similar characteristics and some which do not. For both approaches, the classification models trained only on simulated examples (*MC*, *OVR* and *OVO*) are generally successful in predicting Class Four, with improvements observed with each step down in ensemble base learner complexity. Position and orientation errors (Classes Two and Three) are generally less successful, but again achieve similar scores for both *ANN* and *CNN* approaches. This suggests that the fault-state simulation approach described in Section 3 accurately models the reality of Primary-axis structural/bearing error motions (Class Four), but is less relevant when applied to examples of position (Class Two) and orientation errors (Class Three) on actual, operating machine tools.

Physically, the difficulty in separating orientation from position errors makes sense, an orientation error is effectively a position error with an additional tilt component. The distinction is particularly blurred in the boundary cases, where the tilt component is small and may be overridden by the clear evidence of a position error.

The results for Class One indicated a disparity between the *ANN* and *CNN* approaches, with *ANN* consistently performing poorly and *CNN* performing well. This indicates a likely issue with the feature engineering approach, whereby the encoding for an ideal (Class One) trajectory is not sophisticated enough to generalise from simulated training data to experimental testing examples. This is not an problem for the *CNN* approaches, which incorporate automatic feature extraction through the convolution operations.

Generally speaking, Class One represents a healthy condition – with negligible error present in the system – and Classes Two to Five indicate various forms of faulty conditions. Assessing the F_1-scores for Class One, in the experimental test set, reveals insight

Table 8
Comparison of Secondary-axis classifier skill levels by probability of correct classification, with highest results for simulation and experimental test sets highlighted.

	Probability of correct classification $P(\hat{\mathbf{c}} = \mathbf{c})$	
Model	Simulation test set	Experimental test set
MC_{ANN}	0.9716	0.1739
OVR_{ANN}	0.9524	0.1873
OVO_{ANN}	0.9968	0.3010
FT_{ANN}	0.9974	0.4248
MC_{CNN}	0.8828	0.7793
OVR_{CNN}	0.8996	0.6822
OVO_{CNN}	0.9966	0.7191
FT_{CNN}	**0.9998**	**0.9030**

into the model performance with respect to the actual machine tool condition, with high values indicating the classifiers' ability to discern healthy from faulty states. For the models trained solely on simulation data, there is a large disparity between the feature-based and image-based approaches, with *ANN* models being of low value as general fault detectors, and *CNN* models demonstrating significantly-better performance. When implementing a fine-tuning approach, however, both *ANN* and *CNN* models perform well at separating healthy data from faulty, achieving F_1-scores around 0.9 in both cases.

5.3. Secondary-axis procedure

Table 8 presents the overall probability of correct classification, for all ensemble models trained to identify Secondary-axis faults. In a similar manner to the Primary-axis classifiers, all models achieved high performance on the simulation test set. *MC* and *OVR* ensembles for the *CNN* classifiers are slightly lower, returning classification rates of approximately 90%, however this is improved to near-perfect performance in the *OVO* ensemble implementation.

Observing the experimental test set results, it is clear that the trend observed in Table 7 is reflected in the Secondary-axis *ANN* approaches, where performance is gradually improved through the ensemble strategies, and finally boosted by fine-tuning. The same is not true for the *CNN* approaches, where performance drops by 10% when moving from the *MC* model to *OVR*, followed by a modest increase of only 3% when implementing *OVO*. A notable characteristic of the Secondary-axis *CNN* results is that they are all relatively high, with the MC_{CNN} classifier achieving a correct classification rate of almost 80%. This suggests that ensemble approaches may not be applicable in cases where the classifier is already regarded as strong, and it is not a global method to improve classification ability in any situation. Performance is still boosted when applying the fine-tuning step to both *ANN* and *CNN* approaches, as was observed for the Primary-axis results in Table 7, achieving an impressive 90% classification rate for the FT_{CNN} approach.

Comparing the feature-based *ANN* and image-based *CNN* models in Table 8, it is, again, clear that the *CNN* models resulted in significantly stronger classifiers. Classification rates for the *ANN* approaches are significantly lower than their *CNN* counterparts, for every implementation.

5.3.1. Secondary-axis – Precision–Recall curve

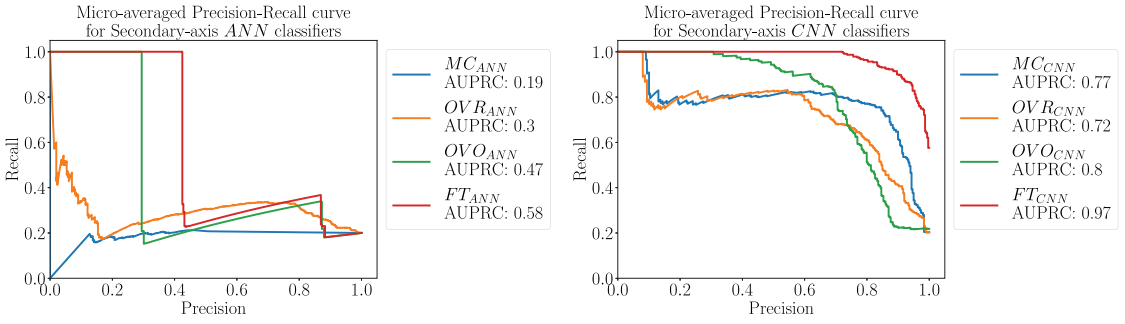
Figs. 11(a) and 11(b) show the micro-averaged PR curves, for Secondary-axis *ANN* and *CNN*-based classifiers, respectively. Again, only a comparison of the experimental test set results is presented.

The PR curves for the *ANN* networks confirm the poor performance suggested from Table 8, as well as the impressive performance achieved by the *CNN*-based classifiers. The – supposedly – 'worst' method, MC_{CNN} returned an AUPRC score of 0.77. As previously noted, the classification rate for OVO_{CNN} indicates slightly worse performance as compared with MC_{CNN}, although the AUPRC results report a small improvement. The PR curve confirms the excellent performance of the FT_{CNN} model, approaching a perfect classifier on the experimental test set evaluation, and returning an AUPRC result of 0.97.

5.3.2. Secondary-axis – F_1-score by class

Figs. 12(a) and 12(b) show the F_1-score results by class, for the Secondary-axis assessment on the simulation hold-out test set. Parallel conclusions on overall performance are drawn from the results, as compared with the Primary-axis simulation test set results in Figs. 10(a) and 10(b). One notable difference in the results for MC_{CNN} and OVR_{CNN} is that there is some observed difficulty in separating Secondary-axis position (Class Two) and orientation (Class Three) errors, with scores around 0.8 being returned. When probing a single artefact, as previously mentioned, orientation errors are only visible in the presence of a position error. There is likely a region of very low orientation error parameters in the simulation set, which are difficult to discern in the resolution of the images provided to the *CNN*-based models, causing misclassification and affecting the precision and recall of these two classes. The issue is rectified by simplifying the problem for the relevant *OVO* base learners, and simulated test set results are improved to near-perfect levels again.

Figs. 12(c) and 12(d) show the F_1-score results by class, for the Secondary-axis assessment on the experimentally-obtained test set. Observation confirms that the *ANN* methods exhibit generally poor performance. Most notably, position errors (Class Two)



(a) Feature-based, ANN models. (b) Image-based, CNN models.

Fig. 11. Precision–recall curve for Secondary-axis image-based classifiers, with values for the area under the PR curve (AUPRC).

were not recognised by any of the classification models. This highlights a major concern with the feature engineering approach, in that it is heavily reliant on manually encoding the feature space. However, as there are only two examples of this class available for evaluation, a small number of misclassifications has a drastic effect on the results, so the issue may be largely attributed to insufficient representation of the class in the experimental test set. Automatic feature extraction in the *CNNs* still struggles for the models which are purely trained on the simulation, however the score is improved when fine-tuned for FT_{CNN}.

Orientation and structural/bearing error motions (Classes Three and Four) appear to be the easiest to recognise in the models trained purely on simulation, with impressive scores for the *CNN*-based models in particular. Software (Class Five) faults are not readily identified in these models, suggesting the simulation does not accurately represent their reality. Fine-tuning significantly improves the scores of all classes — with the exception of Class Three, which already demonstrated a high score of 0.9. FT_{ANN} sees little improvement on its baseline scores in OVO_{ANN}.

On general condition assessment, it is clear from Fig. 10(c) that the feature-based *ANN* approaches have not produced viable fault detectors, with scores hovering around 0.3 at best. FT_{ANN} performs well, but performance in the simulation-only trained models is temperamental, with good performance recorded for the *MC* model but extremely poor scores in *OVR/OVO*. The temperamental performance in the simulation-only models adds to the motivation for implementing fine-tuning in this problem, as high-level fault detection must ultimately be the first priority in any condition assessment.

5.3.3. A note on network complexity

The results in this section generally indicate that, for this application, the *CNN* networks outperform their *ANN* counterparts in the majority of classification model comparisons. A large contributor to this is the level of complexity that can be represented in a deep network, which is made possible by the implementation of residual connections in the ResNet architecture. One notable general characteristic of the PR curves, when comparing *ANN* and *CNN* networks, is the curve's behaviour when the trade-off threshold is reached and recall begins to drop. The curves for OVO_{ANN} and FT_{ANN} classifiers all demonstrate a nearly-vertical drop, which in most cases drops directly to levels of a no-skill (a recall value of around 0.2, for a five class problem) classifier. This reflects the relative simplicity in the *ANN* network, indicating that there is a cut-off point defining the maximum probability output, and all predictions above this cut-off are uncertain. This contrasts with the *CNN* networks, which all demonstrate a smooth curve, indicating both a wider range of probability outputs and much higher certainty for some of the more obvious examples. The latter case is clearly preferable, as it provides the system with a more robust indication of the reliability of its predictions.

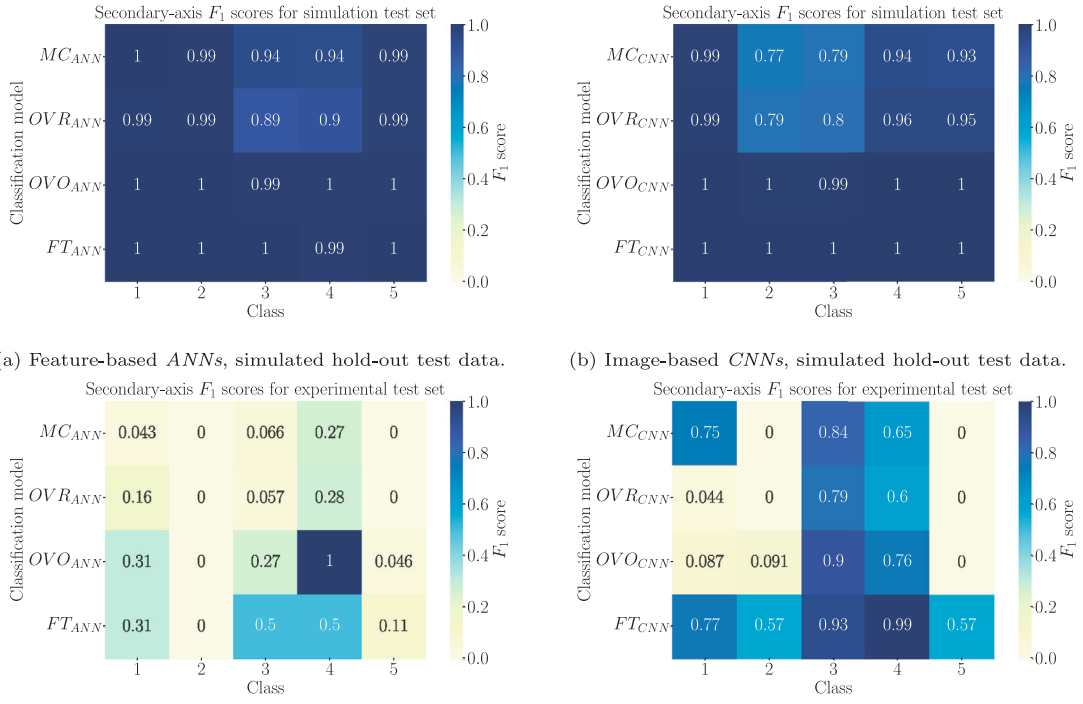
6. Comparison with numerical best-fitting approaches

The general pattern recognition approach, presented in this paper, is novel when placed in the context of the wider error identification and calibration field. The majority of research in the field is based on kinematic theory and methods which apply numerical best-fitting, which has led to mature methods capable of identifying all kinematic error model parameters – defined in ISO 230-1 [41] and ISO 230-7 [3] – with high degrees of accuracy. Software solutions [42] are now also available for performing the parameter identification automatically.

The traditional, numerical best-fitting approach, and the general pattern recognition approach proposed in this paper, both have strengths and weaknesses in the context of multi-axis machine tool fault detection. Key comparison points to consider between the approaches are provided below.

6.1. Error parameter identification accuracy

There is no doubt that numerical best-fitting can identify kinematic model parameters with finer granularity, as compared with the pattern recognition approach proposed in this paper. The research has greatly matured in direct pursuit of this goal, and the efforts have generally been very successful in achieving it. Conversely, the pattern recognition approach is limited to separating faults into their respective classes, and is not designed for the specific determination of error parameter values.



(a) Feature-based *ANNs*, simulated hold-out test data.

(b) Image-based *CNNs*, simulated hold-out test data.

(c) Feature-based *ANNs*, experimental test data.

(d) Image-based *CNNs*, experimental test data.

Fig. 12. F_1-*score* by class, for Secondary-axis classifiers.

6.2. *Kinematic faults*

For the same reasons as outlined above, numerical best-fitting is, generally, the superior method for the identification of faults that can be defined in the kinematic error model.

6.3. *Non-kinematic faults*

Being rooted in kinematic theory, the use of numerical best-fitting is typically implemented as a *white-box* method, and as such it is effectively constrained to error parameters that can be defined in a kinematic model. A key benefit of the approach proposed in this paper, as compared with numerical best-fitting, is the ability to identify faults which cannot be defined in a kinematic model, but are known to exhibit an effect on the error motion trajectory (such as a software fault on the controller) by industry experts [9].

6.4. *Adaptability and machine-specific signatures*

Another key benefit of general pattern recognition, as compared with numerical best-fitting, is the ability to adapt to less obvious patterns in the error motion trajectories. To an extent, this can be dealt with via thresholding in a traditional best-fitting approach. However, the fine-tuning approach proposed in this paper introduces the ability to identify, and adapt to, the specific characteristics of the machine being tracked, as more data becomes available during its operational life-cycle.

6.5. *A hierarchical approach*

The approach, presented in this paper, is not intended as an improved alternative to the mature methods of the field. Rather, the general pattern recognition approach targets a different goal, to compliment the well-established and comprehensive methods already developed with the practice of numerical best-fitting. A production-ready system, built with the proposed approach, would take elements from both approaches to implement an enhanced fault detection model. The general pattern recognition model could provide high-level fault detection, the ability to detect non-kinematic faults and robustness against trajectories influenced by machine-specific signatures. Upon detection of a likely kinematic fault, a numerical best-fitting approach would then be applied, to accurately identify the error model parameters for calibration.

7. Concluding remarks

This paper investigated a complete and adaptable solution for performing multi-class fault diagnosis with error motion trajectory data, in the context of a machine tool condition monitoring system. All approaches were initially trained purely on simulated, theoretical examples, and tested on an experimental test set obtained from actively-operational machine tools. Classification rates indicated performance improvements of up to 22% when implementing One-Vs-Rest and One-Vs-One ensemble learning approaches, as compared with a baseline Multi-Class classifier. This improvement was consistent across all experiments, with the exception of one, where the Multi-Class classifier had a high initial classification rate, and the ensemble approaches led to a small performance degradation in comparison. An approach for retaining an ensemble model when unexpected examples appear in an experimental test set was also implemented. Classification rate results showed that re-training boosted performance in every implementation, leading to further performance improvements of up to 34%. Performance on the simulation environment was also evaluated, which confirmed that the information learned in the initial training phase was successfully maintained during re-training. Image-based, convolutional neural network implementations were consistently found to be more effective than lightweight fully-connected networks based on a minimally-engineered feature vector. Re-trained convolutional neural networks ultimately achieved classification rates of 84.4% and 90.3% for Primary- and Secondary-axis fault-states, respectively.

Future work related to this project involves investigating more sophisticated and automated feature extraction techniques for lightweight, feature-based classifiers, to improve on the image-based convolutional classifiers that were successful in this paper. The work-to-date only considers the case where a single label may be assigned to any given example of error motion trajectory data. In reality, a single instance of trajectory data may contain insight into a number of different fault-states, so expanding the classification methods developed in this paper to a multi-label setting would be a valuable further development.

CRediT authorship contribution statement

T. Rooker: Conceptualization, Methodology, Software, Formal analysis, Data curation, Writing – original draft, Visualization, Project administration. **J. Stammers:** Supervision, Writing – review & editing, Conceptualization. **K. Worden:** Supervision, Writing – review & editing, Conceptualization. **G. Potts:** Supervision, Writing – review & editing. **K. Kerrigan:** Investigation, Resources, Writing – review & editing. **N. Dervilis:** Supervision, Writing – review & editing, Conceptualization.

Declaration of competing interest

One or more of the authors of this paper have disclosed potential or pertinent conflicts of interest, which may include receipt of payment, either direct or indirect, institutional support, or association with an entity in the biomedical field which may be perceived to have potential conflict of interest with this work. For full disclosure statements refer to https://doi.org/10.1016/j.ymssp.2021.108271.The doctoral project is partly sponsored by metrology software products ltd., so the context of the research is broadly related to their industry sector. The contributions in this paper are unrelated to any of their products, and are intended for more general research community interests.

Acknowledgements

The authors would like to gratefully acknowledge metrology software products ltd. and the Engineering and Physical Sciences Research Council (EPSRC) grant EP/I01800X/1 for supporting this research. KW would like to additionally acknowledge support from an EPSRC Established Career Fellowship, UK EP/R003645/1.

Appendix A. Artificial neural networks

The ANN was originally inspired by the concept of the biological neural networks understood to comprise the brain. It broadly comprises an *input layer*, *output layer* and a number of *hidden layers* in a connected network structure; through which, input objects are propagated, to arrive at some desired output. The basic network model can be described as a series of functional transformations. The connections between nodes in neighbouring layers are defined by a matrix of *weight parameters*, w_{ji}, which are used to construct a series of M linear combinations of a single input variable, $\mathbf{x}$, of the form,

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{A.1}$$

where $j = 1, \ldots, M$. The superscript (1) indicates that the weights belong to the first layer in the network, a_j is the *activation* value for the node j, and w_{j0} is an additional bias parameter. Each of the activations is then transformed using a (usually) differentiable, nonlinear *activation function*, $h(\cdot)$, to obtain,

$$z_j = h(a_j) \tag{A.2}$$

which are generally referred to as the *hidden units*. The choice of nonlinear activation function $h(\cdot)$ is dependent upon the particular application. The Rectified Linear Unit (ReLU) – also known as the *ramp* function – given by,

$$h(x_i) = \max(0, x_i) \tag{A.3}$$

is commonly selected for modern implementations, as its simple differentiation reduces computational cost and idempotent property helps prevent the occurrence of vanishing gradients during model training [43]. The hidden units are then linearly combined again to obtain the *output unit activations*, in a similar manner to Eq. (A.1). The process is repeated across each layer in the network architecture, and a final transformation is performed at the output layer to obtain the network outputs, y_k. The choice of activation function is, again, informed by the application, in particular the nature of the data and assumed distribution of the target variable. For a multi-class problem, the normalised exponential function [30], or *softmax* function, of the form,

$$\text{softmax}(a_k) = \frac{\exp(a_k)}{\sum_{l=1}^{K} \exp(a_l)} \tag{A.4}$$

is often employed, which normalises the values in the output layer to give a probability distribution over the K total classes. The resultant function – obtained through successive applications of Eqs. (A.1) and (A.2) from the input variable, $\mathbf{x}$, to the output prediction, $\mathbf{y}$ – defines a *feed-forward neural network*. A key property of the feed-forward neural network is that it is differentiable with respect to the network parameters, $\mathbf{w}$, allowing model training to occur by updating the network weights with *backpropagation*.

A recent paper [38] demonstrated success in applying a Tabu Search optimisation procedure to find an optimal network architecture; the approach is a population-based evolutionary algorithm, based on the notion of distancing ones-self from suboptimal solutions and intensifying the search where optima are more likely. Metrics obtained through evaluation of the validation set also provide an early indication of the model's ability to *generalise*, and can be useful for stopping the training procedure early when model convergence is identified to preserve computational resources and avoid overfitting. The completion of both training and validation steps signifies one full *epoch*. The learning process is then repeated until either a pre-determined number of epochs have elapsed, or early stopping criteria is met.

A.1. On generalisation and overfitting

Generalisation is the ability of a predictor to correctly identify the labels of previously unseen data. A model which achieves good performance on the test set but fails to generalise is regarded as one which *overfits*, which is often a result of fitting an overly-complex function to noisy training data which does not necessarily reflect the true distribution. There are numerous regularlisation techniques that can be employed to reduce the likelihood of overfitting. Two prominent examples in the context of ANNs are *dropout* [39] - which randomly freezes nodes in the hidden layers, forcing the model to find alternative paths through the network – and, to a lesser extent, batch normalisation [44] – which normalises the activations of each layer by subtracting the mean and dividing by the standard deviation of the batch. A critical requirement for ANNs to generalise well is the provision of a sufficient quantity of training examples [27]. Due to the high number of network parameters (particularly so in modern deep learning applications), ANNs have the potential to learn extremely complex functions, and will often overfit if insufficient data is provided during the training phase.

Appendix B. Convolutional networks

Due to their fully-connected nature, regular ANNs do not scale well to image classification tasks. The *Convolutional Neural Network* (CNN) subverts this by focusing on the extraction of local features, based on the notion that information learned in local regions will be generally useful elsewhere. Features learned in early layers can then be combined in later layers to detect higher-order features, which yield deeper and more holistic information about the image.

The CNN is implemented through three mechanisms: (i) *local receptive fields*, (ii) *weight sharing* and, (iii) *sub-sampling*. The input object $\mathbf{X}$ is a tensor with form $\mathbb{R}^{D_1 \times D_2 \times D_3}$ where D_1 and D_2 are the spatial dimensions of the image, and D_3 is the dimension containing the colour channels. The convolutional layer is organised into $L^{(1)}$ planes (for the first hidden layer), each of which is called a *feature map*. Feature maps are constructed by scanning an $f \times f \times D_3$ tensor of weight parameters – known as a *filter* – across overlapping local receptive fields of the input object, performing a convolution operation between the two to populate a single element of the 2-Dimensional feature map.

The convolution operation is repeated $L^{(1)}$ times with different filters to obtain the first convolutional layer. A critical strength of the CNN is that the parameters contained in the $f \times f \times D_3$ filter are shared across the entire feature map, which is a significantly smaller number than would be required for the same representation in a fully-connected ANN. This leads to models which are both more computationally efficient, and more resilient to overfitting.

A controlled reduction in the spatial dimension usually follows the convolution layer, by passing the outputs through a sub-sampling layer. For example, sub-sampling may be applied to local receptive fields of size 2×2 in the feature map, extracting the largest value and using this to populate the corresponding element in the sub-sampled output. This particular operation is known as *max pooling*.

Expressed in matrix form, the output unit activations, $\mathbf{A}^{(l)}$, for a convolutional and subsampling layer pair, l, is given by,

$$\mathbf{A}^{(l)} = \mathbf{W}^{(l)}\mathbf{Z}^{(l-1)} \tag{B.1}$$

with shared weights, $\mathbf{W}^{(l)}$, and the hidden units, $\mathbf{Z}^{(l-1)}$. The current layers hidden units, $\mathbf{Z}^{(l)}$, are obtained by applying a nonlinear activation function, h,

$$\mathbf{Z}^{(l)} = h^{(l)}(\mathbf{A}^{(\mathbf{l})}) \tag{B.2}$$

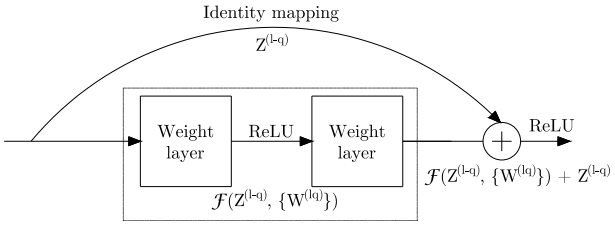


Fig. B.13. A typical shortcut block in the residual learning scheme, with notation following Appendix B.

such as the ReLU function, defined in Eq. (A.3). This framework is then repeated through multiple cycles of convolutional/sub-sampling layers, until a transformation is made to a $1 \times K$ fully-connected structure for label prediction (in a classification problem). The entire network can be trained with a slight modification to the backpropagation procedure described above, to ensure that the shared-weight constraints are satisfied.

Richer feature detection is generally achieved with deeper networks. This advantage has, however, been limited by the issue of vanishing/exploding gradients in backpropagation [43], which hinders convergence. The issue has been largely solved by batch normalisation [44], with the additional benefit of regularising the network. With deeper networks able to converge, a *degradation* problem emerges, where model accuracy saturates with increasing depth and then degrades rapidly. Deep residual learning [28] addresses this problem by considering that a deeper network can be represented in shallower form by *construction*, where identity maps are defined to realise shortcut connections through the network.

In theory, if additional layers can be constructed as identity maps, a deeper network should have no greater training error than its shallower counterpart. Rather than attempt to construct the identity map between two points in a network directly, the approach instead attempts to construct the residual map instead, which the authors of the original paper hypothesise will be easier to optimise. The degradation problem itself suggests that a learner may struggle to approximate an identity map which represents multiple nonlinear layers. By instead constructing the residual, the learner can simply push the weights of the nonlinear layers towards zero if the identity map is optimal. In the more realistic case where the identity map is suboptimal, then the approach can at least help to precondition the problem and make it easier to solve [28]. A typical building block is shown in Fig. B.13. Formally, this is defined as,

$$\mathbf{Z}^{(l)} = \mathcal{F}(\mathbf{Z}^{(l-q)}, \{\mathbf{W}^{lq}\}) + \mathbf{Z}^{(l-q)} \tag{B.3}$$

where l is the output layer reference, q is the number of stacked layers in the shortcut, and $\mathcal{F}(\mathbf{Z}^{(l-q)}, \{\mathbf{W}^{lq}\})$ represents the residual mapping to be learned. The shortcut connection approach introduces no additional parameters or computational complexity, but allows for extremely deep network representations with excellent generalisation performance.

References

[1] H. Schwenke, W. Knapp, H. Haitjema, et al., Geometric error measurement and compensation of machines-an update, CIRP Ann. Manuf. Technol. 57 (2) (2008) 660–675.
[2] International Organization for Standardization, ISO230-9 Test Code for Machine Tools, Part 9: Estimation of Measurement Uncertainty for Machine Tool Tests, 2005.
[3] International Organization for Standardization, ISO 230-7 Test Code for Machine Tools - Part 7: Geometric Accuracy of Axes of Rotation, 2015.
[4] S. Weikert, W. Knapp, R-test, a new device for accuracy measurements on five axis machine tools, CIRP Ann. Manuf. Technol. 53 (1) (2004) 429–432.
[5] S. Ibaraki, C. Hong, C. Oyama, Construction of an error map of rotary axes by static R-test, in: Proceedings of the 6th International Conference on Leading Edge Manufacturing in 21st Century, LEM 2011, vol. 51.
[6] S. Ibaraki, T. Iritani, T. Matsushita, Calibration of location errors of rotary axes on five-axis machine tools by on-the-machine measurement using a touch-trigger probe, Int. J. Mach. Tools Manuf. 58 (2012) 44–53.
[7] S. Ibaraki, T. Iritani, T. Matsushita, Error map construction for rotary axes on five-axis machine tools by on-the-machine measurement using a touch-trigger probe, Int. J. Mach. Tools Manuf. 68 (2013) 21–29.
[8] J.R. Mayer, Five-axis machine tool calibration by probing a scale enriched reconfigurable uncalibrated master balls artefact, CIRP Ann. Manuf. Technol. 61 (1) (2012) 515–518.
[9] P. Hammond, T. Brown, NC-Checker - metrology software products ltd, 2010, URL http://metsoftpro.com/nc-checker/.
[10] R. Teti, K. Jemielniak, G. O'Donnell, et al., Advanced monitoring of machining operations, CIRP Ann. Manuf. Technol. 59 (2) (2010) 717–739.
[11] N. Ghosh, Y.B. Ravi, A. Patra, et al., Estimation of tool wear during CNC milling using neural network-based sensor fusion, Mech. Syst. Signal Process. 21 (1) (2007) 466–479.
[12] D.A. Tobon-Mejia, K. Medjaher, N. Zerhouni, CNC machine tools wear diagnostic and prognostic by using dynamic Bayesian networks, Mech. Syst. Signal Process. 28 (2012) 167–182.
[13] F. Zhao, X. Mei, T. Tao, et al., Fault diagnosis of a machine tool rotary axis based on a motor current test and the ensemble empirical mode decomposition method, Proc. Inst. Mech. Eng. C 225 (2011) 1121–1129.
[14] Y. Zhang, Q. Zhang, Research and discussion on the electrical fault of the CNC machine, in: Proceedings of the 2011 2nd International Conference on Digital Manufacturing and Automation, ICDMA 2011, pp. 305–308.
[15] S. Hu, F. Liu, Y. He, et al., An on-line approach for energy efficiency monitoring of machine tools, J. Cleaner Prod. 27 (2012) 133–140.
[16] S. Emec, J. Krüger, G. Seliger, Online fault-monitoring in machine tools based on energy consumption analysis and non-invasive data acquisition for improved resource-efficiency, Procedia CIRP 40 (2016) 236–243.

[17] J. Wang, M. Qi, Application of intelligent fault diagnosis technology in NC machine tool fault diagnosis, in: ICEOE 2011-2011 International Conference on Electronics and Optoelectronics, Proceedings, vol. 4. pp. 424–427.
[18] B. Shen, S.Y. Zhao, J.H. Wang, Ontology-based fault diagnosis knowledge representation of CNC machine tool, Appl. Mech. Mater. 427–429 (2013) 1372–1375.
[19] W. Wang, H. Li, P. Huang, et al., Data acquisition and data mining in the manufacturing process of computer numerical control machine tools, Proc. Inst. Mech. Eng. B 232 (13) (2018) 2398–2408.
[20] B. Schmidt, K. Gandhi, L. Wang, Diagnosis of machine tools: Assessment based on double ball-bar measurements from a population of similar machines, Procedia CIRP 72 (2018) 1327–1332.
[21] K. Xing, X. Rimpault, J.R. Mayer, et al., Five-axis machine tool fault monitoring using volumetric errors fractal analysis, CIRP Ann. 68 (1) (2019) 555–558.
[22] K. Xing, S. Achiche, J.R. Mayer, Five-axis machine tools accuracy condition monitoring based on volumetric errors and vector similarity measures, Int. J. Mach. Tools Manuf. 138 (2019) 80–93.
[23] T. Rooker, J. Stammers, K. Worden, et al., Machining centre performance monitoring with calibrated artefact probing, Proc. Inst. Mech. Eng. B (2020).
[24] J.A. Soons, F.C. Theuws, P.H. Schellekens, Modeling the errors of multi-axis machines: a general methodology, Precis. Eng. 14 (1) (1992) 5–19.
[25] Y. Abbaszadeh-Mir, J.R. Mayer, G. Cloutier, et al., Theory and simulation for the identification of the link geometric errors for a five-axis machine tool using a telescoping magnetic ball-bar, Int. J. Prod. Res. 40 (18) (2002) 4781–4797.
[26] D.C. Cong, B.B. Chinh, H. Jooho, Volumetric error model for multi-axis machine tools, in: Procedia Manufacturing, vol. 1, pp. 1–11.
[27] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[28] K. He, X. Zhang, S. Ren, et al., Deep residual learning for image recognition, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2016-Decem. pp. 770–778.
[29] R. Zhao, R. Yan, Z. Chen, et al., Deep learning and its applications to machine health monitoring, 2019.
[30] C. Bishop, Pattern Recognition and Machine Learning, first ed., Springer-Verlag, New York, 2006.
[31] C.C. Aggarwal, Data Classification: Algorithms and Applications, 2014.
[32] M. Asafuddoula, B. Verma, M. Zhang, A divide-and-conquer-based ensemble classifier learning by means of many-objective optimization, IEEE Trans. Evol. Comput. 22 (5) (2018) 762–777.
[33] S. Grossberg, Competitive learning: From interactive activation to adaptive resonance, Cogn. Sci. 11 (1) (1987) 23–63.
[34] A. Robins, Sequential learning in neural networks: A review and a discussion of pseudorehearsal based methods, 2004.
[35] N. Kamra, U. Gupta, Y. Liu, Deep generative dual memory network for continual learning, 2017, arXiv.
[36] V. Marochko, L. Johard, M. Mazzara, et al., Pseudorehearsal in actor-critic agents with neural network function approximation, in: Proceedings - International Conference on Advanced Information Networking and Applications, pp. 644–650.
[37] M. Buda, A. Maki, M.A. Mazurowski, A systematic study of the class imbalance problem in convolutional neural networks, Neural Netw. 106 (2018) 249–259.
[38] T.K. Gupta, K. Raza, Optimizing deep feedforward neural network architecture: A tabu search based approach, Neural Process. Lett. 51 (3) (2020) 2855–2870.
[39] N. Srivastava, G. Hinton, A. Krizhevsky, et al., Dropout: A simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (2014) 1929–1958.
[40] T. Saito, M. Rehmsmeier, The precision–recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets, PLoS One 10 (2015).
[41] International Organization for Standardization, ISO 230-1, Test Code for Machine Tools — Part 1: Geometric Accuracy of Machines Operating under No-Load or Quasi-Static Conditions, Technical Report, International Organization of Standards, 1996.
[42] S. Ibaraki, Y. Nagai, H. Otsubo, et al., R-test analysis software for error calibration of five-axis machine tools: Application to a five-axis machine tool with two rotary axes on the tool side, Int. J. Autom. Technol. 9 (4) (2015) 387–395.
[43] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, J. Mach. Learn. Res. 9 (2010) 249–256.
[44] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: Proceedings of the 32nd International Conference on Machine Learning, pp. 448–456.