Neural Computing and Applications
https://doi.org/10.1007/s00521-023-09403-1

ORIGINAL ARTICLE

Neural network classification of eigenmodes in the magnetohydrodynamic spectroscopy code Legolas

J. De Jonghe[1,2] · M. D. Kuczyński[3]

Received: 7 July 2023 / Accepted: 13 December 2023

Abstract
A neural network is employed to address a non-binary classification problem of plasma instabilities in astrophysical jets, calculated with the Legolas code. The trained models exhibit reliable performance in the identification of the two instability types supported by these jets. We also discuss the generation of artificial data and refinement of predictions in general eigenfunction classification problems.

Keywords Magnetohydrodynamics · Eigenproblem · Neural network · Supervised learning · Classification

1 Introduction

For many plasma- or fluid-related disciplines the question of stability is of central interest. In fusion research instabilities break confinement [24], in solar physics they lead to eruptions like coronal mass ejections [22], and in space weather they affect the propagation and properties of the solar wind [25]. Since many plasma configurations can host a plethora of instabilities, determining the driving forces or physical effects which give rise to them is crucial. Additionally, when several instabilities coexist, it is essential to identify which one is dominant, i.e. grows most rapidly, and thus governs the initial evolution of the plasma configuration.

To address this central question, the magnetohydrodynamic (MHD) spectroscopic code Legolas ([8], and https://legolas.science) was developed, which allows for the investigation of the influence of physical parameters, such as flow and resistivity, on the dynamics of a plasma configuration, and in particular, on its magnetohydrodynamic stability. For a given equilibrium structure and a choice of non-ideal effects (i.e. resistivity, viscosity, etc.), the Legolas code computes all the waves and instabilities supported by the plasma (also referred to as modes). Using this tool, one can obtain a comprehensive overview of the instabilities and their respective growth rates within the considered parameter space. However, the results also contain a multitude of modes which are not necessarily relevant to stability, like sequences of slow, Alfvén, and fast waves. Consequently, it may become difficult to track any specific mode during the exploration of the parameter space or pinpoint which effect is causing it. Hence, we seek an algorithm for categorising modes, in order to distinguish and classify the relevant instabilities.

The process of classification, where one assigns a label from a finite, predefined set to an object, is a notoriously time-consuming task if performed manually. Hence, it is desirable in many fields to automate the classification of data. With the continuous development of new machine learning techniques, many architectures have been explored for classification [18, 23, 27]. Here, we improve on the results of Kuczyński et al. [19], introducing a supervised neural network designed for the non-binary classification of any generalised eigenvalue problem. We apply the model to the study of an astrophysical jet [2], which are also replicated in recent experiments [3], here

✉ M. D. Kuczyński
mkuc@ipp.mpg.de

J. De Jonghe
jkmdj1@st-andrews.ac.uk

[1] Centre for mathematical Plasma Astrophysics, KU Leuven, Celestijnenlaan 200B, Box 2400, 3001 Leuven, Belgium

[2] School of Mathematics and Statistics, University of St Andrews, Mathematical Institute North Haugh, St Andrews KY16 9SS, UK

[3] Max Planck Institute for Plasma Physics, Wendelsteinstraße 1, 17491 Greifswald, Germany

Published online: 16 January 2024

with shear axial flow embedded in a helical magnetic field [1], and demonstrate reliable performance.

First, we introduce the equations solved by the Legolas code in Sect. 2, and then describe the particular physical system used for testing the model in this work. In Sect. 3, we describe the eigenvalue classification algorithm, and present a method for enlarging training data and refining model predictions. Subsequently, Sect. 4 outlines how the described algorithm is applied to the test problem, focussing on data preparation, suggested neural network architectures, overall performance metrics, and the filtering criterion for 'uninteresting' modes. Finally, in Sect. 5 we verify the performance of the algorithm, comparing the outcome between the two introduced network architectures.

2 Astrophysical jets in the Legolas code

Before introducing the neural network-based algorithm, we briefly describe the data generated with the Legolas code. The MHD spectroscopic code Legolas [7, 8, 10] is a finite element method (FEM) code that solves the generalised eigenproblem

$$\boldsymbol{A}\boldsymbol{f} = \omega \boldsymbol{B}\boldsymbol{f}, \tag{1}$$

that arises after linearisation and 3D Fourier analysis of a set of (magneto)hydrodynamic equations. For the data used in the present classification problem, the equations are linearised around an equilibrium representing an astrophysical jet with shear axial flow embedded in a helical magnetic field, as described by Baty and Keppens [1]. The equilibrium is described by a constant density ρ_0 and velocity, magnetic field, and temperature profiles

$$\boldsymbol{v}_0(r) = \frac{V}{2}\tanh\left(\frac{R_j - r}{a}\right)\hat{\mathbf{e}}_z, \tag{2}$$

$$\boldsymbol{B}_0(r) = B_\theta \frac{r/r_c}{1 + (r/r_c)^2}\,\hat{\mathbf{e}}_\theta + B_z\,\hat{\mathbf{e}}_z, \tag{3}$$

$$T_0(r) = T_a - \frac{B_\theta^2}{2\rho_0}\left(1 - \frac{1}{[1 + (r/r_c)^2]^2}\right), \tag{4}$$

where V is the asymptotic velocity, R_j is the jet radius, a is the radial width of the shear layer, r_c is the characteristic length of the radial magnetic field variation, B_θ and B_z are magnetic field strength parameters, and T_a is the temperature at the jet axis. For this study, 240 Legolas runs of this configuration were carried out in the interval $r \in [0, 2]$ for various values of V. At $r = 0$, a regularity condition was imposed, whilst a perfectly conducting boundary condition was used at $r = 2$. Table 1 gives an overview of all the parameter values.

Table 1 Parameters of the data used in this study

V	1.29	1.43	1.58	1.72	1.86	2.01	
N	R_j	r_c	a	B_θ	B_z	T_a	ρ_0	m
151	1	2	0.1	1	0.25	1	1	−1

The upper table shows the different values of V in the dataset. For each value of V, k was varied from 0.5 to 7 in increments of 1/6. The parameters in the lower table were identical in all cases

In each Legolas run, the resistive, compressible MHD equations [e.g. 13],

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\rho \boldsymbol{v}), \tag{5}$$

$$\rho \frac{\partial \boldsymbol{v}}{\partial t} = -\nabla p - \rho \boldsymbol{v} \cdot \nabla \boldsymbol{v} + \boldsymbol{J} \times \boldsymbol{B}, \tag{6}$$

$$\rho \frac{\partial T}{\partial t} = -\rho \boldsymbol{v} \cdot \nabla T - (\gamma - 1) p \nabla \cdot \boldsymbol{v} + (\gamma - 1)\eta \boldsymbol{J}^2, \tag{7}$$

$$\frac{\partial \boldsymbol{B}}{\partial t} = \nabla \times (\boldsymbol{v} \times \boldsymbol{B}) - \nabla \times (\eta \boldsymbol{J}), \tag{8}$$

perturbed and linearised around the equilibrium (2-4), are solved for the frequency ω and Fourier amplitudes $\hat{f_1}(r)$ after substituting a Fourier form,

$$f_1 = \hat{f_1}(r)\,\exp[\mathrm{i}(m\theta + kz - \omega t)], \tag{9}$$

for each perturbed quantity (ρ_1, $\boldsymbol{v}_1$, T_1, $\boldsymbol{B}_1$), with imposed wave numbers m and k. In these MHD equations, p represents the pressure, governed by the ideal gas law $p = \rho T$, and $\boldsymbol{J} = \nabla \times \boldsymbol{B}$ the current. Furthermore, η is the resistivity, set to $\eta = 10^{-4}$, and γ the adiabatic index. Since we employ a fully ionised, non-relativistic approximation, the adiabatic index is set to $\gamma = 5/3$.

For a grid discretisation with N grid points, Legolas computes $16N$ complex eigenvalues ω and their 8 corresponding (complex) eigenfunctions ρ_1, $\boldsymbol{v}_1$, T_1, and $\boldsymbol{B}_1$ on a grid with $2N - 1$ grid points [8]. Hence, for the classification algorithm, each of the resulting $16N$ data points is treated as a 2-vector containing the real and imaginary parts of the eigenvalue along with its corresponding complex $8 \times (2N - 1)$ matrix containing the eigenfunctions.

For this jet configuration, the associated spectrum of complex eigenvalues contains up to two types of instabilities: one Kelvin–Helmholtz instability (KHI) and a parameter-dependent amount of current-driven instabilities (CDI) [1, 15]. The spectra for two distinct parameter choices are shown in Fig. 1a, b as examples. In Fig. 1a the

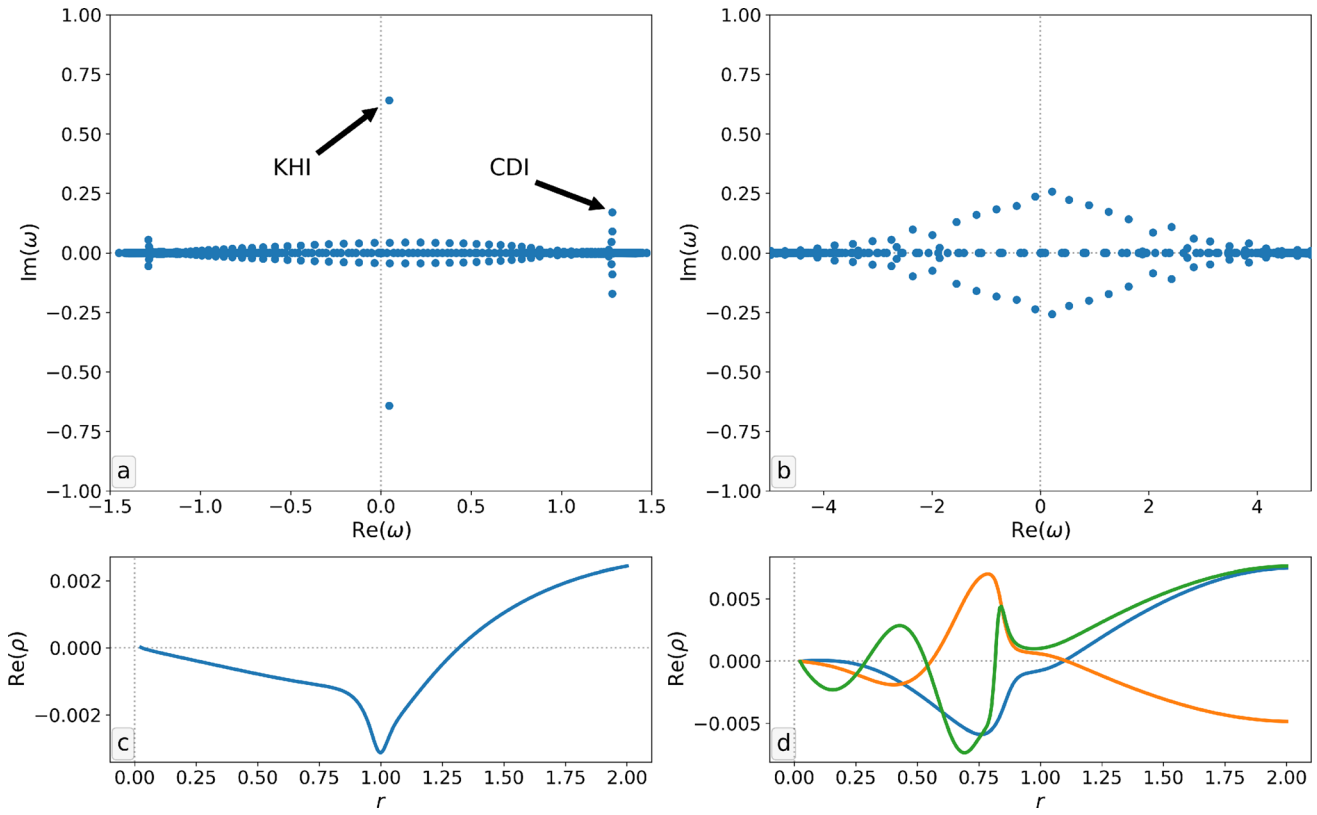

Fig. 1 Spectra of configuration Eqs. (2–4) for parameters **a** $V = 1.72$ and $k = 1.5$; **b** $V = 1.863$ and $k = 6.5$. **c** $\mathrm{Re}(\rho)$-eigenfunction of the KHI in **a**. **d** $\mathrm{Re}(\rho)$-eigenfunction of the three fastest growing CDIs in **a**. (151 grid points)

KHI and the sequence of CDIs are indicated. Though they are easily identifiable in this first case by their position in the spectrum, this is harder for the case in Fig. 1b, where some modes are not fully resolved at this resolution. In general, we identify the instabilities by their eigenfunction behaviour. The real part of the ρ-eigenfunction of the KHI, visualised in Fig. 1c, is characterised by a maximum located at the jet boundary ($r = R_j$) whereas CDIs are characterised by a smooth, oscillatory behaviour inside the jet ($r < R_j$), as illustrated in Fig. 1d. Modes that do not possess any of these characteristics are referred to as uninteresting.

3 A mathematical framework for classification algorithms

Here, we describe the algorithm that we applied for the classification of KHIs and CDIs. In this section, a sufficient degree of generality is maintained for potential applications in related fields. We show the usefulness of maps under which the classification algorithm is invariant for data generation and testing. Finally, we propose a qualitative structure of the algorithm.

3.1 Class preserving maps

The goal of classification algorithms is to associate a unique label $l \in L$ with an input $x \in X$, where L and X are sets. Mathematically, this is a function from X to L, i.e.

$$\text{Class} : X \to L. \tag{10}$$

In the present work, this map is realised via a supervised neural network and a subsequent, user-informed filtering procedure. In order to gain a better understanding of the structure of this algorithm, in what follows, we introduce the concept of class preserving maps.

We define the training dataset $T \subset X$ that consists of all training data points $t \in T$ such that the map in Eq. (10) is known. Consider the set of maps U from X to itself that preserve the class label. Denoting such a map by u, we have

$$u : X \to X \text{ such that } \forall x \in X : \ \text{Class}\,(u(x)) = \text{Class}\,(x). \tag{11}$$

It is apparent that $\text{Class}\,(u(t)) = \text{Class}\,(t)$. If u is an injective map satisfying $u(T) \subset X \setminus T$, any element $u(t)$ can be used to extend the training dataset T. However, if u is not injective or $u(T) \cap T \neq \emptyset$, care should be taken not to include repeated elements in the dataset, which could

introduce an imbalance in the training data. Additionally, let $x' \in X \setminus T$ be an input whose class label must be inferred by the neural network. Rather than making a prediction on a single input x', one can also compare it with $u(x')$ which, in an ideal scenario, should result in the same label. The user is then able to choose a prediction dependent on their preferred filtering scheme. If the model is free from systematic errors, this results in a higher likelihood of correct classification.

However, if a certain map u_l only preserves the class of a subset $X_l \subset X$, it cannot be used for the reinforcement of the network's prediction, since, a priori, we do not know if the class of the data point being predicted will be unaltered. Nevertheless, u_l can still be used for data generation. These two types of class preserving maps are illustrated in Fig. 2. In the context of image classification, examples of maps denoted by u in this figure include image rotations and translations, since all physical objects should be invariant under these transformations, while flipping is only applicable to symmetric objects and hence is an element of u_i. An overview of common maps used for data augmentation is given in [26].

3.2 Structure of the eigenvalue classification algorithm

In some applications, the input to the neural network is an ordered tuple. This could be, for example, a text–image pair or, as in our problem, an eigenmode-eigenfunction pair $(\omega, \boldsymbol{f}_\omega)$, where the subscript ω now indicates that the eigenvector $\boldsymbol{f}_\omega$ is associated with the eigenvalue ω. In such scenarios, it is common to implement separate branches for different constituents of the input [e.g. 4]. In our model, we first extract convolutional features of $\boldsymbol{f}_\omega$ in a separate branch, which results in a reduced representation $\bar{\boldsymbol{f}}_\omega$. Then, ω is simply concatenated with $\bar{\boldsymbol{f}}_\omega$. The combined result $(\omega, \bar{\boldsymbol{f}}_\omega)$ is then further fed into a regular neural network that in the end returns the probability of each class label l. Then, probability thresholds are optimised in order to maximise the chosen metric which judges the performance of the model. Finally, once the neural network is trained and the thresholds are chosen, a filtering scheme is incorporated based on the previously defined maps $u^i \in U$. The complete scheme of the eigenvalue classification algorithm as applied to the KHIs and CDIs classification problem is shown in Fig. 3.

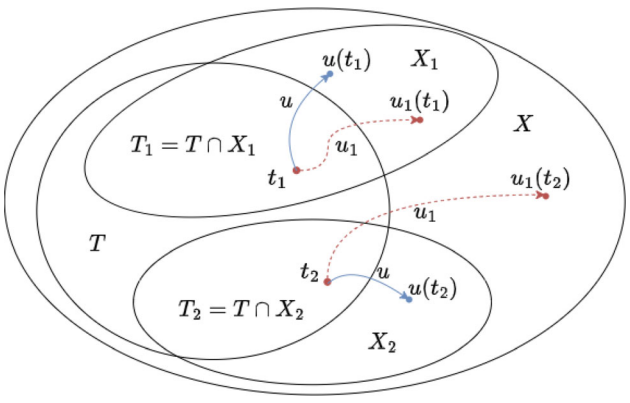

Fig. 2 Types of class preserving maps. The map $u \in U$, indicated by the blue solid line, preserves the class of all eigenfunctions $\boldsymbol{f} \in X$. The map $u_1 \in U_1$, indicated by the red dashed line, preserves only the internal maps of the subspace denoted as X_1 in Figure

4 Application to Legolas jet data

Here, we apply the algorithm described in Sect. 3 on the KHI-CDI classification problem introduced in Sect. 2. First, we return to the data structure, and discuss how the data is expanded with the use of class preserving maps. Then, we describe the network architecture presenting the network's layers in more detail. Subsequently, we discuss the chosen performance metrics and how we optimised them by introducing probability thresholds and a filtering procedure.

4.1 Data generation

Since all 240 Legolas runs were performed with $N = 151$ grid points, each file contains $16N = 2416$ eigenmodes. Every eigenmode has 8 associated complex eigenfunctions discretised on a grid with $2N - 1 = 301$ points. Hence, the network's input consists of two parts:

1. An eigenvalue input ω as a real 2-vector $(\mathrm{Re}(\omega), \mathrm{Im}(\omega))$.
2. An eigenfunction input as a complex matrix $\boldsymbol{f}_\omega$ of dimensions 301×8. Its real and imaginary parts are stored in 2 channels.

We decided to use 80% (192) of these runs for training, 10% (24) for validation, and 10% for testing. The division of the files across the three categories was randomised. For the exact distribution, see Table 4 at the end.

The resulting data contained 99.76% uninteresting modes (class 0), i.e. modes that are neither KHI (class 1) nor CDI (class 2). Therefore, in order to balance and extend the dataset, we utilised the technique of class preserving maps, described in Sect. 3.1. We defined the following maps:

1. Multiplying the eigenfunctions by a complex phase factor:

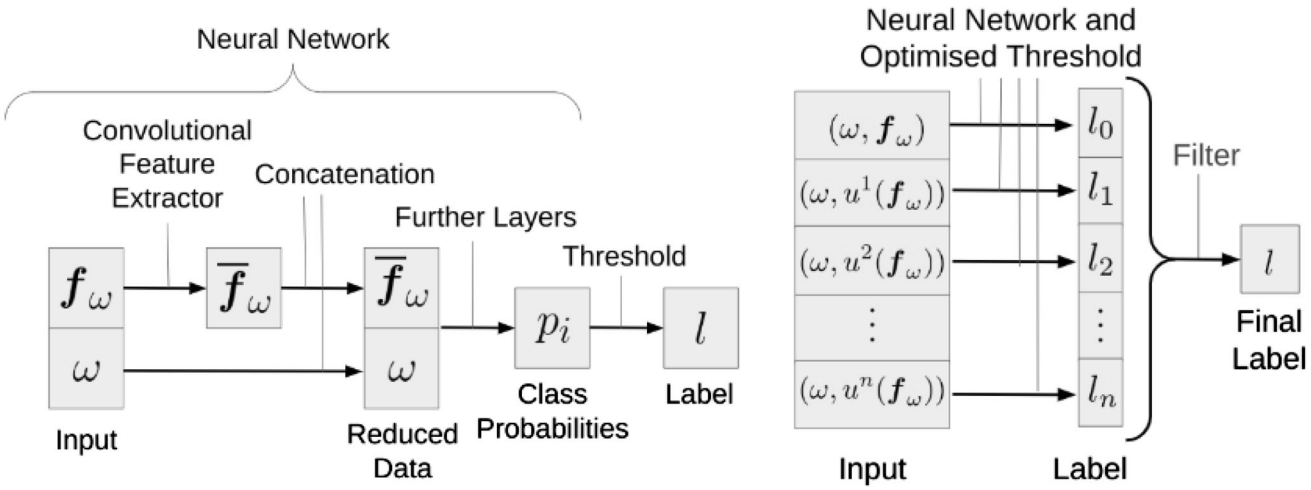

Fig. 3 A schematic of the employed classification algorithm

$$u : X \to X : (\omega, \boldsymbol{f}_\omega), \mapsto (\omega, \mathrm{e}^{i\varphi}\boldsymbol{f}_\omega) \text{ with } \varphi \in (0, 2\pi). \tag{12}$$

This map $u \in U$ is always class preserving because eigenfunctions are only determined up to a complex factor. As a consequence of this property, we can also use these maps to decrease the uncertainty of the neural network's prediction in the filtering step, as discussed in Sect. 3.1.

2. Data superposition: if $\theta = z = t = 0$ in Eq. (9), the corresponding eigenvalue–eigenfunction tuples satisfy the following superposition principle,

$$u_l : (X_l \times X_l) \to X_l,$$
$$(\omega, \boldsymbol{f}_\omega, \omega', \boldsymbol{f}_{\omega'}) \mapsto \left(\frac{\omega + \omega'}{2}, \mathrm{e}^{\mathrm{i}\varphi}\boldsymbol{f}_\omega + \mathrm{e}^{\mathrm{i}\varphi'}\boldsymbol{f}_{\omega'} \right), \tag{13}$$

with $\omega = \omega'$ and $\varphi, \varphi' \in (0, 2\pi)$. Nevertheless, when employed for $\omega \neq \omega'$, this map typically preserves the inherent characteristics of the considered modes, i.e. the peak at the jet boundary ($r = R_j$) for KHI modes, and the oscillatory behaviour inside the jet ($r < R_j$) for CDI modes. Additionally, the purpose of the sum $\frac{1}{2}(\omega + \omega')$ is to artificially assign information about the growth rate to the created mode. Therefore, we treated Eq. (13) as an approximate class preserving map for general modes of class l.

The resulting distribution of initial, and final training, validation, and testing data is summarised in Table 2.

4.2 Network architecture

We propose two different neural network architectures, one for high performance computers and one for single thread computations. The former is a variant of the ResNet [16] and the latter is a plain network. The models were developed in Keras [6], an open-source neural network library written in Python.

First, we discuss the architecture of the ResNet. As described in Sect. 3.2, the network consists of two stages. Initially, only the input eigenvector $\boldsymbol{f}_\omega$ is passed through a convolutional feature extractor. Then, the result $\overline{\boldsymbol{f}}_\omega$ is concatenated with the eigenvalue ω and further fed into the second stage. Figure 4 shows the main building blocks of the network used in both stages, where some layers are marked with a symbol for reference here. In the first stage, the weights layers (*) are convolution layers whilst in the second stage they are dense layers. The kernels of the convolutional layers within a building block are of the same size. Both stages consist of three such main building blocks. The kernel sizes are (65, 5), (33, 3), (17, 2) and the number of convolutional filters is 128, 64 and 32 accordingly. The intermediate dense layers in the second stage have sizes 34 and 17. The dropout (†) value is 0.1 for convolution layers and 0.5 for dense layers. Average pooling (‡) is only used for convolution layers. We chose the Adam optimizer with a learning rate of 0.001, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. The loss function is the categorical cross-entropy.

Regarding the plain network, the main difference is that it follows the skip connection only (§ top), and the main branch (§ bottom) is removed. Since the training was much more computationally inexpensive than in the case of the ResNet, we were able to fine-tune the network's hyperparameters more. In fact, the kernel sizes, number of filters, and intermediate dense layer sizes listed in the previous paragraph were chosen as such, since these were the ones that were close to optimal for the plain network.

Table 2 Distribution of initial and generated data samples across different classes for neural network classification training, validation, and testing phases

Dataset	Class 0		Class 1			Class 2			Total #modes
	Raw	u	Raw	u	u_1	Raw	u	u_2	
Dataset	99.76	0	0.04	0	0	0.20	0	0	579 840
Training	0	25.00	0	3.75	33.75	0	3.75	33.75	512 per batch
Validation	0	33.34	0	33.33	0	0	33.33	0	10 000
Testing	99.79	0	0.04	0	0	0.17	0	0	57 720

For each class, the values are given as a percentage (%) of the total number of modes in the last column. Entries associated to u and u_l are then the percentage generated using Eqs. (12) and (13), respectively

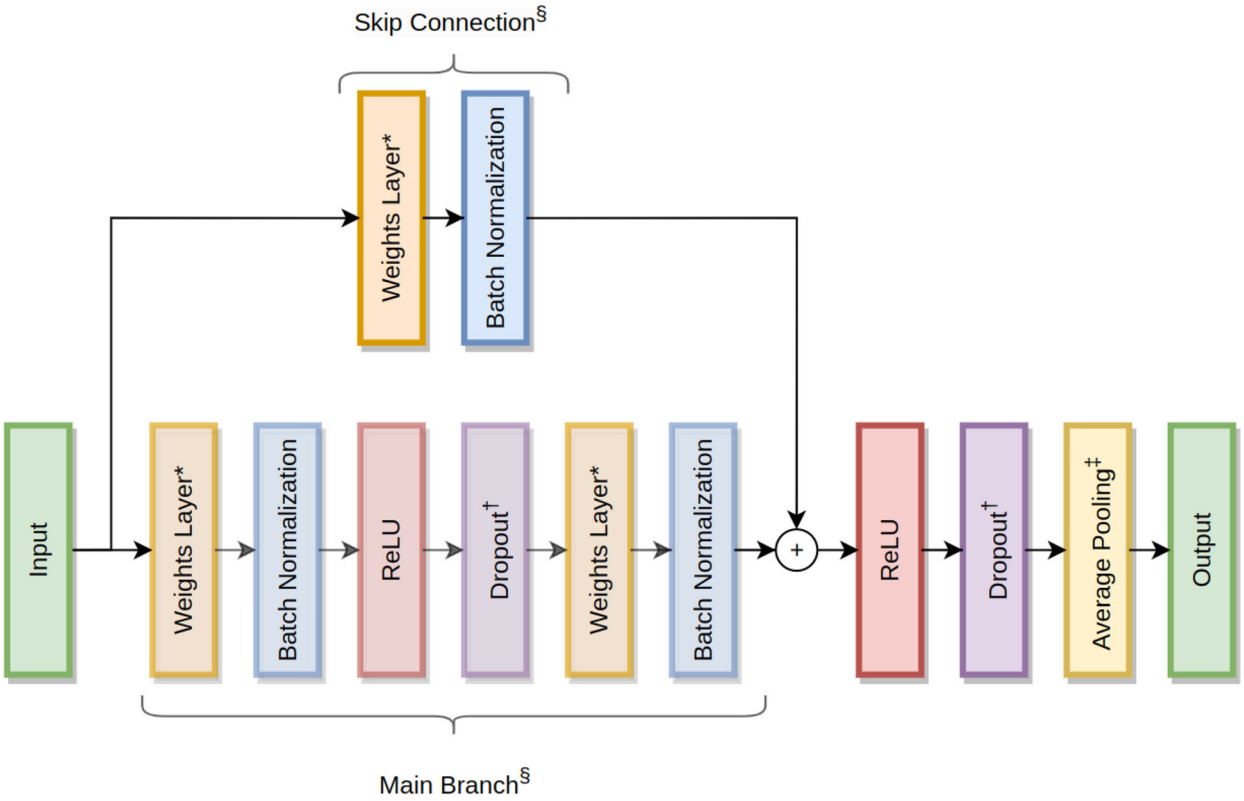

Fig. 4 Diagram of the main building blocks of the neural networks

4.3 Probability thresholds and performance metric

In this application, the primary objective of the eigenfunction classification algorithm is to identify the relevant modes from a large dataset, which contains predominantly uninteresting modes. The final model should only misclassify a few relevant modes as uninteresting whilst correctly filtering out most of the truly uninteresting modes. Hence, for evaluating the model's performance, we chose two metrics: precision and recall. As usual [see e.g. 14], precision and recall are defined as

$$\text{precision} = \frac{\text{true positives}}{\text{false positives} + \text{true positives}}, \tag{14}$$

$$\text{recall} = \frac{\text{true positives}}{\text{false negatives} + \text{true positives}}, \tag{15}$$

where positives and negatives are modes that the model labelled as relevant (class 1 or 2) and uninteresting (class 0), respectively. Denoting the elements of the confusion matrix as c_{ij}, where the rows correspond to the true label, and the columns to the predicted label, the precision and recall are given as

$$\text{precision} = \frac{d_{11}}{d_{01} + d_{11}}, \quad \text{recall} = \frac{d_{11}}{d_{10} + d_{11}}, \tag{16}$$

$$d_{ij} = \begin{cases} d_{00} = c_{00}, & d_{01} = c_{01} + c_{02}, \\ d_{10} = c_{10} + c_{20}, & d_{11} = c_{11} + c_{12} + c_{21} + c_{22}. \end{cases} \tag{17}$$

This way, we emphasise the priority of identifying relevant modes over distinguishing between classes 1 and 2. Finally, as an informative metric, we introduce the balanced accuracy [5],

$$\text{balanced accuracy} = \frac{1}{2}\left(\frac{d_{00}}{d_{00} + d_{01}} + \frac{d_{11}}{d_{10} + d_{11}}\right). \tag{18}$$

After training the network, the predicted probabilities for each class are denoted as (p_0, p_1, p_2). In the inference step, instead of simply selecting the class with the highest probability, we introduce thresholds a and b. We first determine if $p_1 \geq a$. If it is, class 1 is predicted. Otherwise, we evaluate if $p_2 \geq b$. If this condition is met, class 2 is predicted. Otherwise, class 0 is the default prediction. To improve the classification algorithm, the thresholds a and b are optimised using validation data. This is done by imposing a desired recall value, i.e. a tolerance on the number of discarded relevant modes. We then seek the highest possible precision as a function of a and b. Since the distributions of validation and testing data are different (Table 2), during testing we can expect the recall and precision values to deviate from their validation values, that are the results of this optimisation procedure.

4.4 Filtering

Once the thresholds for a and b are established, the eigenfunctions of the testing data can again be subjected to the class preserving maps of the form (12). The resulting data couples $(\omega, \mathrm{e}^{\mathrm{i}\varphi}\boldsymbol{f}_\omega)$ are evaluated by the network and classified according to the optimised thresholds a and b. Repeating this for m different phases φ_k $(k = 1, \ldots, m)$ results in a total of $m + 1$ predictions (including the unmodified data). Subsequently, the final label is the label that was predicted the most often, with ties broken in favour of relevant over uninteresting, and if decidedly relevant, class 1 taking precedence over class 2. This technique could be further improved by considering the variance of the predictions in line with the work of Gal and Ghahramani [12]. Furthermore, both techniques could be used simultaneously. Nevertheless, we achieved satisfactory results with calculating only the most likely prediction from the ensemble of predictions generated from class preserving maps which we report in the next section.

5 Results

First, the plain network was trained on 900 batches of 512 modes each. Then, using validation data, probability thresholds a and b were optimised such that the corresponding validation recall was at least 90%. Next, five predictions were generated, and a final label was extracted, as described in Sect. 4.4. The resulting confusion matrix is shown in Fig. 5a. From this confusion matrix, we calculate the precision, recall, and balanced accuracy using Eqs. (16-18). The network thus achieves a recall of 94.3%, a precision of 36.3%, and a balanced accuracy of 97.0% on the testing data. Therefore, of the modes classified as 1 or 2, 36.3% were truly relevant modes, whilst 5.7% of all relevant modes were lost.

Similarly, the ResNet was trained on 2230 batches of 512 modes each, but the a and b thresholds were optimised for a minimal validation recall of 95%. Following the same classification process as the plain network resulted in the confusion matrix in Fig. 6a. Again, applying Eqs. (16–18) to this confusion matrix, the ResNet reached a recall value of 97.6%, precision of 38.0%, and a balanced accuracy of 98.7% on the testing data, outperforming the plain network in all metrics.

By imposing different minimal validation recall values, we can control how many relevant modes are lost. Of course, adapting the validation recall also changes the testing recall, precision, balanced accuracy, and thresholds (a, b). For varying validation recall values, the testing recall, precision, and thresholds are visualised in Fig. 5b for the plain network and in Fig. 6b for the ResNet. Unsurprisingly, higher validation recall values lead to higher testing recall, and lower testing precision and thresholds. As the graphs show, the testing recall is consistently higher than the imposed validation recall, with a validation recall of 95% sufficing to achieve a perfect testing recall for the plain network (at the cost of lower precision). It is remarkable however that the a threshold is independent of the imposed validation recall and remains at a constant, high value of 71%. This implies that the network easily recognises class 1 modes and assigns a high probability to them. Moreover, if the minimal validation recall is increased to 95%, there are no additional modes mislabelled as class 1, and the decrease in precision is solely due to the misclassification of class 0 modes as class 2 modes.

For the ResNet this behaviour is even more pronounced. Since its a threshold is close to 1, with the b threshold only slightly lower, the network has to assign extremely high probabilities to either class 1 or 2 to consider a mode relevant. However, if the imposed validation recall is larger

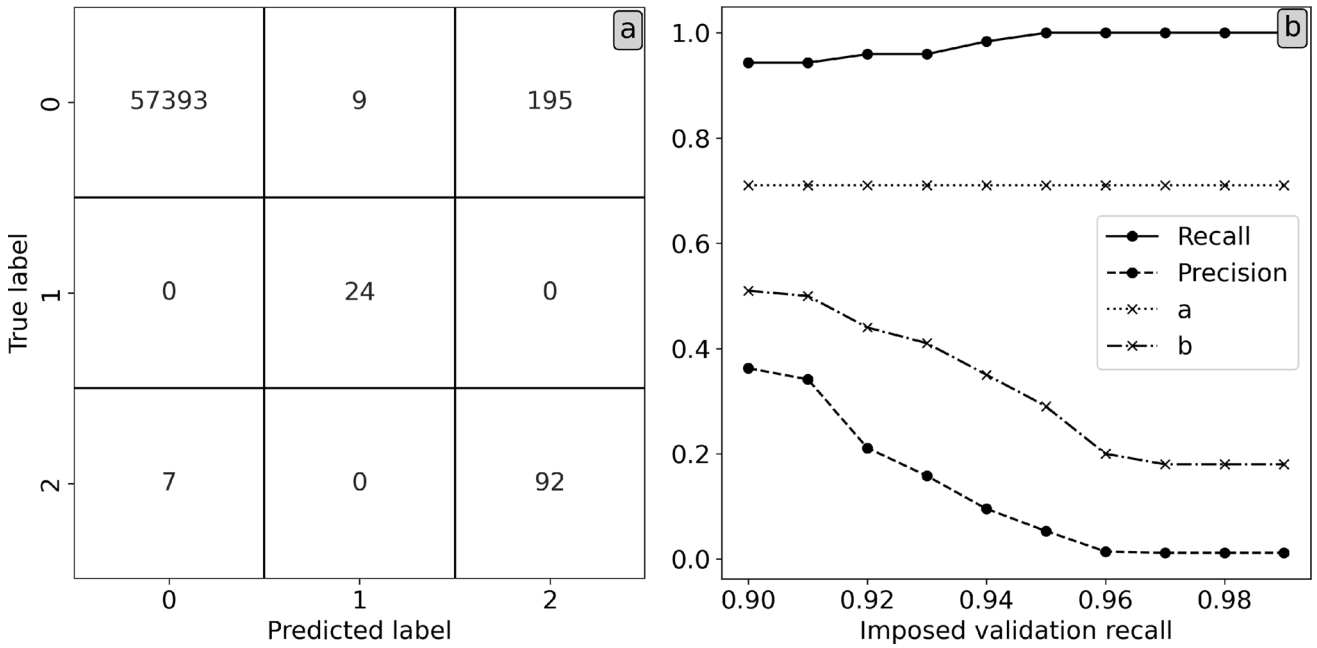

Fig. 5 Plain network. **a** Confusion matrix for a minimal validation recall of 90%. **b** Recall, precision, and thresholds as functions of the imposed validation recall

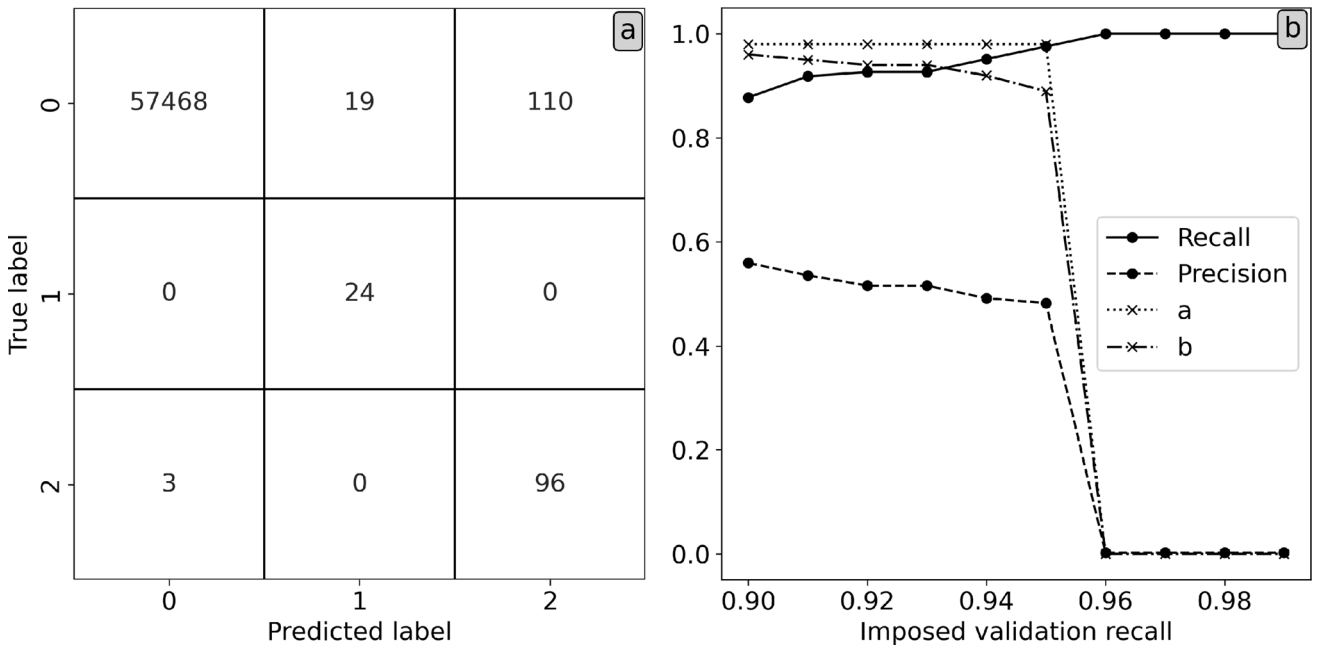

Fig. 6 ResNet. **a** Confusion matrix for a minimal validation recall of 95%. **b** Recall, precision, and thresholds as functions of imposed validation recall

than 95%, all modes are classified as class 1, i.e. a and b vanish. Thus, the precision equals zero in this case.

Returning to the confusion matrices, two more observations are noteworthy. Firstly, the lower right 2×2 submatrix is diagonal for both networks. Hence, they clearly distinguish between class 1 and class 2 modes. This is in line with initial expectations, based on the eigenfunction shapes, like those shown in Fig. 1c, d, which show a strong peak at the jet boundary at $r = 1$ for KHI modes in contrast with the CDI's oscillatory behaviour in the jet's interior ($r < 1$). Secondly, the relative number of class 0 modes that were misclassified as class 1 is much smaller than the number of class 0 modes that were misclassified as class 2 for the plain network. In addition, it is worth noting that all

Table 3 Precision and recall (%) per class for both the plain network and the ResNet

	Plain			ResNet		
Class	0	1	2	0	1	2
Precision	99.99	72.73	32.06	99.99	55.81	46.60
Recall	99.65	100	94.85	99.78	100	96.97

class 1 modes were correctly identified by both networks. For the networks in Figs. 5a and 6a, all metrics have been computed per class and are displayed in Table 3. From this table it is clear that the ResNet has better recall values, and achieves a higher precision in class 2 at the cost of a lower precision in class 1. In both cases, the network has a higher class 1 precision than class 2 precision.

6 Conclusion

Due to the large amount of natural oscillations of a single plasma configuration (one run of the MHD spectroscopic code Legolas), visual inspection of the eigenmodes to identify characteristics can be a monotonous and time-consuming task. In this work we have applied two convolutional neural networks to a non-binary classification problem of ideal MHD eigenmodes in astrophysical jets, analysed with Legolas. For a recall of 94.3% the plain neural network left 0.55% of all modes for manual inspection. The ResNet offered a significant improvement with a recall of 97.6% leaving 0.43% for inspection. Furthermore, neither network confused class 1 with class 2 modes in the test data. Since even the plain network provided good results already, we conclude that neural networks offer a great opportunity for automated mode detection in Legolas data, and likely, for classification of eigenproblem data in general.

In the context of large parameter studies with the Legolas code, these results are particularly promising. With an automated way of reliably identifying instabilities, various parameters can be varied simultaneously, and the resulting parameter space can be partitioned into sections of similar behaviour efficiently. This approach could have many applications, e.g. in the analysis of jet stability like the problem presented here, or the problem of current sheet stability in various astrophysical settings, where tearing and Kelvin–Helmholtz instabilities compete [17, 20, 21].

A significant drawback of the supervised approach however is of course the need for a large set of pre-classified data for training purposes. To sidestep this issue, future investigations could focus on unsupervised clustering algorithms to search for structures in Legolas data, like the translational-azimuthal distinction in Taylor–Couette flows [9] or the surface-body wave dichotomy in flux tubes [11]. For large parameter studies like those described in the previous paragraph, this method is especially appropriate, considering it requires less prior knowledge of the types of modes one may encounter throughout the parameter space.

Finally, it remains an open question whether a generally applicable neural network for Legolas data is possible. In particular, is it feasible to develop a neural network that can predict which physical effect, like shear flow or magnetic shear, is responsible for each instability in a spectrum? In this regard, another hurdle to overcome is that a generally applicable network should work for various grid resolutions, unlike the network presented here. These questions are left for future research.

Table 4 Files in the dataset used for validation and testing

Validation	Testing
0001-HEL1-V14333.dat	0003-HEL1-V17200.dat
0002-HEL1-V14333.dat	0007-HEL1-V12900.dat
0002-HEL1-V18633.dat	0007-HEL1-V14333.dat
0003-HEL1-V15767.dat	0009-HEL1-V17200.dat
0003-HEL1-V18633.dat	0009-HEL1-V20067.dat
0004-HEL1-V14333.dat	0013-HEL1-V15767.dat
0008-HEL1-V14333.dat	0016-HEL1-V14333.dat
0009-HEL1-V18633.dat	0017-HEL1-V20067.dat
0010-HEL1-V14333.dat	0018-HEL1-V20067.dat
0012-HEL1-V17200.dat	0026-HEL1-V12900.dat
0016-HEL1-V15767.dat	0026-HEL1-V14333.dat
0017-HEL1-V17200.dat	0026-HEL1-V15767.dat
0018-HEL1-V14333.dat	0028-HEL1-V20067.dat
0019-HEL1-V15767.dat	0029-HEL1-V14333.dat
0020-HEL1-V14333.dat	0030-HEL1-V12900.dat
0020-HEL1-V15767.dat	0031-HEL1-V20067.dat
0026-HEL1-V17200.dat	0032-HEL1-V20067.dat
0027-HEL1-V15767.dat	0034-HEL1-V12900.dat
0028-HEL1-V15767.dat	0035-HEL1-V17200.dat
0030-HEL1-V20067.dat	0037-HEL1-V15767.dat
0034-HEL1-V20067.dat	0037-HEL1-V18633.dat
0036-HEL1-V18633.dat	0039-HEL1-V14333.dat
0037-HEL1-V14333.dat	0039-HEL1-V17200.dat
0038-HEL1-V15767.dat	0040-HEL1-V15767.dat

The remaining files were used for training

Acknowledgements The Legolas code is freely available under the GNU General Public License. For more information, visit https://legolas.science/. JDJ was supported by funding from the European Research Council (ERC) under the European Unions Horizon 2020

research and innovation programme, Grant agreement No. 833251 PROMINENT ERC-ADG 2018. The authors have no conflict of interest to declare.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data availability The Legolas data are available as a dataset on Kaggle named Legolas: MHD instabilities in an astrophysical jet. The files that were reserved for validation and testing are listed in Table 4. The Neural Network source code is available on request.

References

1. Baty H, Keppens R (2002) Interplay between Kelvin-Helmholtz and Current-driven Instabilities in Jets. Astrophys J 580(2):800–814. https://doi.org/10.1086/343893
2. Belan M, Massaglia S, Tordella D et al (2013) The hydrodynamics of astrophysical jets: scaled experiments and numerical simulations. Astron Astrophys 554:A99. https://doi.org/10.1051/0004-6361/201321040
3. Bellan PM (2018) Experiments relevant to astrophysical jets. J Plasma Phys 84(5):755840501. https://doi.org/10.1017/S002237781800079X
4. Boldeanu M, Cucu H, Burileanu C et al (2021) Multi-input convolutional neural networks for automatic pollen classification. Appl Sci. https://doi.org/10.3390/app112411707
5. Brodersen KH, Ong CS, Stephan KE et al (2010) The balanced accuracy and its posterior distribution. In: 2010 20th International Conference on Pattern Recognition, pp 3121–3124. https://doi.org/10.1109/ICPR.2010.764
6. Chollet F et al (2015) Keras. https://github.com/fchollet/keras
7. Claes N, Keppens R (2023) Legolas 2.0: Improvements and extensions to an mhd spectroscopic framework. Comput Phys Commun 291:108856. https://doi.org/10.1016/j.cpc.2023.108856
8. Claes N, De Jonghe J, Keppens R (2020) Legolas: a modern tool for magnetohydrodynamic spectroscopy. Astrophys J Suppl Ser 251(2):25. https://doi.org/10.3847/1538-4365/abc5c4
9. Dahlburg RB, Zang TA, Montgomery D et al (1983) Viscous, resistive magnetohydrodynamic stability computed by spectral techniques. Proc Natl Acad Sci USA 80(18):5798–5802
10. De Jonghe J, Claes N, Keppens R (2022) Legolas: magnetohydrodynamic spectroscopy with viscosity and Hall current. J Plasma Phys 88(3):905880321. https://doi.org/10.1017/S0022377822000617
11. Edwin PM, Roberts B (1983) Wave propagation in a magnetic cylinder. Solar Phys 88(1–2):179–191. https://doi.org/10.1007/BF00196186
12. Gal Y, Ghahramani Z (2015) Dropout as a Bayesian approximation: representing model uncertainty in deep learning. https://doi.org/10.48550/ARXIV.1506.02142
13. Goedbloed H, Keppens R, Poedts S (2019) Magnetohydrodynamics of laboratory and astrophysical plasmas. Cambridge University Press. https://doi.org/10.1017/9781316403679
14. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press. http://www.deeplearningbook.org
15. Hardee PE (2011) The stability of astrophysical jets. In: Romero GE, Sunyaev RA, Belloni T (eds) Jets at All Scales, pp 41–49. https://doi.org/10.1017/S1743921310015620
16. He K, Zhang X, Ren S et al (2016) Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE Computer Society, Los Alamitos, CA, USA, pp 770–778. https://doi.org/10.1109/CVPR.2016.90
17. Hofmann I (1975) Resistive tearing modes in a sheet pinch with shear flow. Plasma Phys 17(2):143–157. https://doi.org/10.1088/0032-1028/17/2/005
18. Kiranyaz S, Avci O, Abdeljaber O et al (2021) 1d convolutional neural networks and applications: a survey. Mech Syst Signal Process 151:107398. https://doi.org/10.1016/j.ymssp.2020.107398
19. Kuczyński MD, Borchardt M, Kleiber R et al (2022) Magneto-hydrodynamic Eigenfunction classification with a Neural Network. J Comput Appl Math 406:113889. https://doi.org/10.1016/j.cam.2021.113889
20. Li JH, Ma ZW (2010) Nonlinear evolution of resistive tearing mode with sub-Alfvénic shear flow. J Geophys Res Space Phys 115(9):6–11. https://doi.org/10.1029/2010JA015315
21. Li JH, Ma ZW (2012) Roles of super-Alfvénic shear flows on Kelvin-Helmholtz and tearing instability in compressible plasma. Phys Scr 86(4):045503. https://doi.org/10.1088/0031-8949/86/04/045503
22. Lynch BJ, Edmondson JK, Kazachenko MD et al (2016) Reconnection properties of large-scale current sheets during coronal mass ejection eruptions. Astrophys J 826(1):43. https://doi.org/10.3847/0004-637X/826/1/43
23. Moosaei H, Hladík M (2023) Sparse solution of least-squares twin multi-class support vector machine using ℓ_0 and ℓ_p-norm for classification and feature selection. Neural Netw 166:471–486. https://doi.org/10.1016/j.neunet.2023.07.039
24. Nuhrenberg J, Merkel P, Schwab C et al (1993) MHD-theoretical aspects of stellarators. Plasma Phys Control Fus 35:B115. https://doi.org/10.1088/0741-3335/35/SB/009
25. Shaaban SM, Lazar M, Yoon PH et al (2019) Quasilinear approach of the cumulative whistler instability in fast solar wind: constraints of electron temperature anisotropy. Astron Astrophys 627:A76. https://doi.org/10.1051/0004-6361/201935515
26. Yang S, Xiao W, Zhang M et al (2022) Image data augmentation for deep learning: a survey. arXiv preprint arXiv:2204.08610
27. Zhang G (2000) Neural networks for classification: a survey. IEEE Trans Syst Man Cybern Part C Appl Rev 30(4):451–462. https://doi.org/10.1109/5326.897072