Chemometrics and Intelligent Laboratory Systems 223 (2022) 104520

Contents lists available at ScienceDirect

Chemometrics and Intelligent Laboratory Systems

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

A tutorial on automatic hyperparameter tuning of deep spectral modelling for regression and classification tasks

Dário Passos [a,b,*], Puneet Mishra [c,**]

[a] CEOT - Center for Electronics, Optoelectronics and Telecommunications, University of Algarve, Campus de Gambelas, 8005-189, Faro, Portugal
[b] Physics Department, University of Algarve, Campus de Gambelas, 8005-189, Faro, Portugal
[c] Wageningen University and Research, Wageningen Food and Biobased Research, Bornse Weilanden 9, P.O. Box 17, 6700AA, Wageningen, the Netherlands

ARTICLE INFO

Keywords:
Deep learning
Spectroscopy
Chemometrics
Predictive modelling
Optimization

ABSTRACT

Deep spectral modelling for regression and classification is gaining popularity in the chemometrics domain. A major topic in the deep learning (DL) modelling of spectral data is the choice and optimization of the deep neural network architecture suitable for the specific task of spectral modelling. Although there are several recent research articles already available in the chemometric domain showing advanced approaches to deep spectral modelling, currently, there is a lack of hands-on tutorial articles in this space that supply the non-expert user with practical tools to learn and implement advanced DL optimization methodologies aimed at spectral data. Hence, this tutorial article aims at reducing the gap between the non-expert user of DL in the chemometric community and the implementation of DL models for daily usage. This tutorial supplies a quick introduction to the state-of-the-art deep spectral modelling and related DL concepts and presents a set of methodologies aimed at DL hyperparameters' optimization. To this end, this tutorial shows two practical examples on how to implement and optimize two DL models for spectral regression and classification tasks. The models are implemented in *python* and Tensorflow and the complete code is supplied in the form of two complementary notebooks.

1. Introduction

Optical spectroscopy in the visible and near-infrared (Vis-NIR) spectral range is widely explored in the domain of analytical chemistry for rapid and non-destructive assessment of sample's properties [1,2]. The technique can be applied to the analysis of samples of a wide range of physical forms, from gases to solids. Furthermore, its application spans through several areas of scientific research that require non-destructive rapid techniques for samples analysis such as foods [3], pharmaceutical [4–6], agriculture [2,7], forensics [8], plastics [9], and many more [10,11]. Vis-NIR spectroscopy allows the characterisation of physicochemical properties of samples as it retrieves information from the light scattering and absorption patterns; information which can be used to explain some of the physical and chemical properties of the sample [1,12].

In recent decades, major advancements in technology enabled miniaturisation of optical spectroscopy sensors leading to an increased interest and wider adoption of spectrometers, even outside the scientific community [13]. For example, as a consumer electronics tool, pocket spectrometers are now easily available to buy through online platforms [13–15]. Notwithstanding the modernization of this sensing technology, a main core aspect of the success behind Vis-NIR spectroscopy is the spectral data modelling process that allows mapping the spectral information to the desired property of interest [3]. Spectral sensors working in the Vis-NIR range requires an *a priori* calibration step as it is a non-specific technique capturing overtones of fundamental chemical bond vibrations which appear as highly overlapping signals in the captured spectral data [3]. For most of the cases, separating these overlapping signals is not an easy task. In the spectral modelling domain, chemometric and machine learning approaches are used for developing spectral calibrations. Two of the most widely used techniques in this field are principal component analysis (PCA) [16] and partial least-squares regression (PLS) [17,18]. PCA works by projecting the original data points into a new mathematical space defined by a new set of coordinates called Principal Components (PCs). These PCs are defined by finding the

* Corresponding author. CEOT - Center for Electronics, Optoelectronics and Telecommunications, University of Algarve, Campus de Gambelas, 8005-189, Faro, Portugal.
** Corresponding author. Wageningen University and Research, Wageningen Food and Biobased Research, Bornse Weilanden 9, P.O. Box 17, 6700AA, Wageningen, the Netherlands.
E-mail addresses: dmpassos@ualg.pt (D. Passos), puneet.mishra@wur.nl (P. Mishra).

https://doi.org/10.1016/j.chemolab.2022.104520
Received 25 October 2021; Received in revised form 26 January 2022; Accepted 16 February 2022
Available online 24 February 2022

orthogonal directions that maximize variance in the input data. A linear regression model can then be build using these PCs, establishing the basis of Principal Component Regression (PCR). One of the limitations of PCR is that the PCs extracted do not use information about the target variable, hence, the first few PCs used in PCR might not be the most informative features for the prediction of the target. On the other hand, the PLS algorithm considers the covariance between input variables and target variables. Similarly to PCA, PLS also projects the data into a latent space, but this time the components are defined along the direction of maximum variance between input and target variables. These components are called latent variables and are built to model the target variable. PLS can also be paired with a linear discriminant model to perform classification tasks. Beyond these two classical algorithms, basically every other machine learning (ML) algorithm can be, and has been, used to model spectral data. A more modern modelling alternative is provided by deep learning (DL) algorithms that, for many tasks, have recently outperformed other machine learning and chemometric approaches in terms of model performances [19–27]. Recent works in this field show that DL has the potential to cut the need for pre-processing optimization and variable selection [28,29], both key steps in any Vis-NIR spectral analysis pipeline [30–32]. These algorithms are mainly based on artificial neural networks (ANN) and allow for the automated transformation of data with the help of specialized layers (e.g., convolutional layers) [28,29]. Moreover, they can take advantage of spectral data augmentation techniques [33], such as using an ensemble of several pre-processing's as direct input data to the DL models [19,22]. Furthermore, DL modelling also encompasses techniques that allow to apply (or adjust) pre-trained models to new data/conditions by using concepts such as transfer learning [21,27].

Applications of ANN for spectral data analysis have come a long way since [34] highlighted the general principles of classical ANN application to chemical problems. In the current state-of-art, there are two main types of DL methodologies being used for spectral modelling. The first family of models follows the same philosophy as principal component regression (PCR), where first, a smaller number of relevant features are extracted from the spectra in an unsupervised way using Autoencoders neural networks [35–37]. Autoencoders are a type of ANN architecture whose layer structure resembles an hourglass, with the middle layers having a smaller number of units than the input and output layers. As information flows through the Autoencoder, this architecture encodes the data into a highly compressed representation of the input data in the middle layers. These compressed features (or lower dimension features) are then extracted from the Autoencoder and fed into a fully connected ANN architecture such as a multi-layer-perceptron (MLP) that can be tailored for classification or regression tasks. The main point here is that the compressed features extracted from the Autoencoder are not obtained by showing the ANN any target data. The second family of models is somewhat more like partial least-square regression (PLSR), where the features are extracted in a supervised way using convolutional filters by checking the improvement in the predictive ability for the property of interest [19,22,23,28]. Just as PLSR has shown to be more powerful than the PCR technique, the DL supervised method based in convolutional neural networks (CNNs) can be presently considered as the most practical choice over other ANN architectures [29]. Although DL models have shown better performance than several modern machine learning and chemometric approaches, there are some challenges in the deep spectral modelling task mainly related to the ANN model architecture selection and hyperparameter optimization [22,29]. For example, most (if not all) of DL models used currently in spectral analysis use ANN architectures imported from the field of computer vision or natural language processing. There is little work published around the topic of ANN architectures specially tailored for the analysis of spectra. The details of the model architecture tend to be somewhat random with the users selecting the type and number of layers based on the small available literature around this subject, their personal experience or trial and error methods. Such an approach to design a model architecture and the associated lack of model interpretability are sources of scepticism and may not attract the interest of non-expert users, thus limiting a wider adoption of powerful deep spectral modelling techniques. Furthermore, since DL models have many hyperparameters, it is often also a challenge to optimize all the key hyperparameters in an easy and efficient way to obtain good deep spectral models. The wide range of domains where ML is currently applicable spawned an increased demand for the automation of the DL models optimization. In the chemometric and spectral analysis domains, recent works (e.g., [22,38]), have shown how to take advantage of the tools being developed by the ML community to simplify the task of automatic optimization of DL spectral models. With the help of advanced methods based in Bayesian Optimization (BO) and Automated Machine Learning (Auto-ML) already implemented in software packages such as "Optuna" [65], "auto-sklearn" [80] or "NASLib" [81], both the ANN architecture as well as all the key hyperparameters can be optimized with minimal user interference. One of the interesting results of such automatic optimization is that the method is sometimes able to find ANN architectures that outperform pre-existing, larger, and more complex ANN architectures [22,38].

This tutorial article is targets mainly non-expert users of DL in the chemometric community (and related sub-fields) and aims at increasing their tool set for DL modelling. The tutorial provides a practical introduction on how to implement and, as importantly, how to optimize DL models for spectral modelling in *python* using the Tensorflow API. After the presentation of the main theoretical concepts behind the optimization task and the used DL model, two examples are presented: a spectral regression case (predict/quantify a certain quantity using spectral information) and a spectral classification case (classify different samples using their spectra), along with how to implement an automatic optimization pipeline to achieve models that perform well. Some specific tips useful for DL modelling of spectral data are also presented, including data augmentation techniques and the suggestion of tools and methods that can help with the interpretability of the model results. This manuscript is complemented by two "jupyter notebooks" [85] alongside with instructions on how to install the basic software needed to run them. It is expected that non-expert users (but with some basic knowledge in *python* and DL), can easily adapt the given examples/code to their own needs, after completing the tutorials. Despite the present *python* framework choice, the general principles presented can, in principle, be easily ported to either Matlab or R.

2. Deep learning modelling

The first step for the whole modelling process is to choose the neural network architecture that best suits the problem at hand (regression, classification, etc.). In terms of DL models applied to spectroscopic and chemometric tasks there are a few options to choose from the literature. As it was mentioned earlier, the most encouraging results so far were obtained with convolutional neural networks (CNNs) with one or more convolutional layers [19,28,29,39–41] and with auto-encoders [35–37, 42]. For CNNs (the type of DL model used in this tutorial), [28,43] presented a detailed and useful description of how the several layers work and what is the role for many of its hyperparameters, while [44,45] presented a more general higher-level review on DL applied to spectral analysis and chemometrics. These two last references provide a very accessible and useful introduction about the basic concepts of DL relevant for spectra analysis. There is also the possibility for the users to craft their own ANN architecture manually or by automatically implement pipelines that perform Neural Architecture Search (NAS) [46,87], i.e., that test several ANN architectures built from a pool of predefined components (types of layers, regularization methods, etc.) and identify interesting candidate combinations for the target specific tasks. The present tutorial touches just lightly on the topic of NAS but does not dwell on it. NAS is a highly active and interesting research area [47,87] but a deep dive into it lays beyond the scope of this work.

For this tutorial, the chosen DL architecture is a 1D-CNN (Fig. 1) [84]

with one convolution layer followed by a block of dense (a.k.a. fully connected, FC) layers. This model is similar to the usual CNNs used in computer vision for image processing, is "1D" because the input is spectra (vectors) and was chosen for its simplicity and its proven predictive power when applied to several spectroscopic and chemometric problems [19–23,26–28]. It is important to highlight that the level of complexity of a DL model should be adapted to the task at hand. A very simple model might not capture all the relevant information in the data, i.e., it will underfit the training data and generalize poorly. On the opposite direction, a very complex model might "memorize" or overfit the training data and generalize poorly also. The general rule of thumb for manually tuning model's structures is to start with a simple model, optimize its hyperparameters and check if it can obtain good metrics on the training data. If the model is unable to get good training metrics, one should try to add more layers to it (increase its complexity) and repeat the process. Using NAS techniques might help alleviate this troubleshooting step at the cost of more compute time.

A CNN is an ANN that usually aggregates three types of layers, namely, convolution layers, pooling layers and fully connected (or dense) layers (Fig. 1). The convolution layers are responsible for automatically extracting feature maps from the input data. This operation is done by convolving the input data, x, with learnable kernels (or filters), k, of fixed length that slide over the input data with a predefined stride. These layers compute the dot product of a portion of the input data with an overlapped kernel of the same size. The results of this convolution operation are then passed through a non-linear activation function, $a(\,)$, to generate the output feature map for that layer h^k

$$h^k = a\left(w^k * x + b^k\right), \tag{1}$$

where w^k and b^k are the layer's kernel weights and biases respectively [48]. For image processing, it has been shown that different kernels learn to extract different types of features (e.g. vertical and horizontal edges in images, etc.). For spectral analysis, there are evidence that show that these kernels can behave as Savitzky-Golay filters and pre-processing filters [28,43]. In most CNN architectures, especially in the field of computer vision, the convolutional layer is followed by a pooling layer that acts as a selection filter extracting the maximum (or mean) values of each kernel operation and effectively downsampling the feature maps. CNNs can have multiple pairs of these convolutional and pooling layers in the so called "feature extraction block". As information passes forward from one convolutional layer to the next, the low-level features in the first layers aggregate into higher level (more complex) representations of the data (a.k.a. abstractions). These features are then passed to a classification (or regression) block that contains one or several fully connected (or dense) layers with different number of units each. For these types of supervised tasks, the prediction made in the last layer of the CNN is compared to the true value and a loss function is computed. The minimization of this loss function using gradient descent algorithms or second order methods (e.g., L-BFGS) and the subsequent readjustment of the network parameters (the weights and biases of all units) in all the layers by the back-propagation algorithm is what enables the learning process for this type of model. For the interested reader, references [28,29,40,43,48,83,84] provide further details, mathematical descriptions, and applications of CNN models.

For each of the mentioned layer types, the user must configure several options that control the overall behaviour of the model, the so called hyperparameters. The hyperparameters of the convolution layer are the number of filters, the filters stride, the width of the filters and the activation function (e.g., ReLU, ELU, Leaky ReLU, Sigmoid, Tanh, Softmax, etc). Note that sometimes, the activation function can also be coded as a separated layer. The CNN model adopted in this tutorial does not include pooling layers for simplification purposes. The hyperparameters for the dense layers block on this architecture are the number of dense layers and, the number of units (a.k.a. neurons) and activation function per layer. To decrease the risk of overfitting, dropout layers can be used and an L2 regularization penalty, β, to the weights of all layers is applied, therefore introducing additional hyperparameters. Dropout layers stochastically drop some of the connections between units in subsequent layers during the training process [89]. This forces the network to learn sparse representations of the data and acts as a weight regularization mechanism. The use of these regularization techniques give rise to models that tend to overfit less and are less sensitive to noise in the data.

2.1. Hyperparameters choices

Depending on the problem at hand (regression or classification) one will further have to define the appropriate loss function (e.g., Mean Squared Error, Binary Cross-entropy, Huber class, etc.), choose one of the multiple options for the optimization gradient descent algorithm (e.g. SGD, Adam, RMSprop, Adagrad, etc.) and choose the appropriate initialization method for the CNN weights (e.g. Random Normal, He Normal, Glorot Uniform, etc). Depending on the data set size and available computational resources one will further have to define the maximum number of training epochs (where one epoch means that every sample has been seen once) and the training batch size (i.e., the number of samples to feed each time to the ANN) and deal with the fact that some of these former options come with their own subset of hyperparameters. Further details about all these options are available in introductory DL books such as in Ref. [48]. Given this scenario, it is easy to understand how overwhelmed non-DL-expert researchers can become with such a high number of choices that have a direct impact on their model's performance. This is especially true if they are used to deal with much simpler algorithms such as PLS or PCA that possess only one hyperparameter.

Before continuing, here is just an added note of caution about the technical jargon used in Machine Learning, and more specifically around neural networks. Hyperparameters are model's variables that control the behaviour of the model and the overall ANN architecture. These are specified by the user before the training process and are in most cases static (do not change during training). On the other hand, ANN parameters (weights and biases) are model variables that are estimated/learned during the training process and are directly tied to the data being analysed. The optimization techniques presented in this tutorial are aimed at model's hyperparameters, while the model's parameters themselves will always be optimized by a classical gradient descent algorithm.

There are many general-purpose tutorials about DL online that are a particularly good starting point for curious minded researchers. They offer simple rules of thumb that ease some of these choices, helping to develop an intuition about what to choose and when to choose it. Depending on the chosen ANN architecture and data type, some hyperparameters have a much higher impact on the performance of the model than others [49]**.** The correct choice of values is perhaps one of the things that are still considered as the "black magic" of DL practitioners, maybe not so different in essence to the choice of thresholds and number of latent variables for some more classical chemometric techniques. However, to demystify this procedure, one can implement hyperparameter optimization strategies (also called hyperparameter-tuning) [86] that automatically help the user to assess the best values to squeeze every single drop of performance of their models. In the next section, three examples of such optimization strategies are highlighted.

2.2. Grid search and randomized grid search optimizations

For the task of hyperparameter optimization, one tries many sets of model hyperparameters, $\boldsymbol{\theta}$, and chooses the one, $\boldsymbol{\theta}^*$, that provide the best model performance on a specific data set, i.e.

$$\boldsymbol{\theta}^* = arg\ min_\theta\ \mathscr{L}(f(x), \boldsymbol{\theta}) \tag{2}$$

where $\mathscr{L}(f(x), \boldsymbol{\theta})$ is a predefined loss function built from a mapping function or model $f(x)$ and its hyperparameters. The number of hyper-

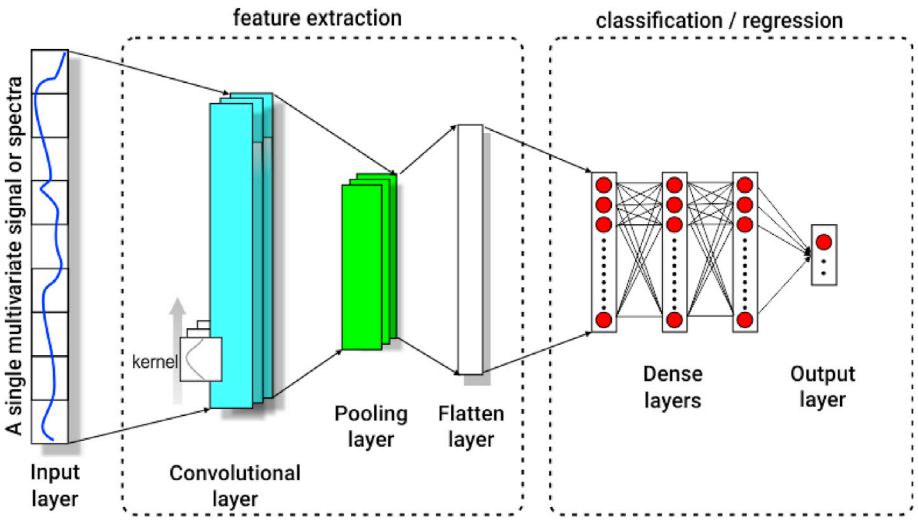

Fig. 1. Typical representation of a 1D convolutional neural network architecture. The feature extraction block is composed by convolutional and pooling layers and the classification/regression block is composed by dense layers. In the 1D-CNN model used in this tutorial, no pooling layer is considered.

parameters that needs to be optimized is often defined by the model choice and constrained by the computational resources available and allocated research time. In theory, one can optimize all hyperparameters of a complex model, but in practice, given the combinatorial dependence of hyperparameters, this situation easily becomes impractical for most academic researchers that have limited computational resources, especially when dealing with ANNs architectures that can be computationally expensive to train. Therefore, the widespread practice is to fix some of the less impactful hyperparameters and probe multiple combinations of the most relevant ones. This distinction can be achieved by experience (trial and error), by earlier practical knowledge of the model's behaviour or by theoretical constraints. Consult [50] for a more formal introduction to the importance of ANN's most common hyperparameter.

In terms of systematic approaches (i.e., discarding the usual manual trial-and-error optimization), Grid Search (GS) and Randomized Grid-Search (RGS) are the two most used methods for hyperparameter optimization in ML. For the sake of simplicity, let's assume as an example an ANN for regression purposes where just one hyperparameter, θ, needs to be optimized. In GS, one pre-defines a vector of θ values and, for each value, trains the ANN model and registers the corresponding performance (e.g., MSE) on a validation set (a subset of the training data also sometimes called tuning set). Alternatively, the user can choose to go with a cross-validation strategy and use the full training set (more computationally costly for large models and large data sets). After probing all the pre-defined values of θ, the value that supplies the best model performance, i.e., the one that satisfies Eq. (2), is chosen. One disadvantage of this method is that many values of θ might have to be probed before finding a value that leads to improved model performance and, besides that, one does not know for sure if the optimal θ was included in the pre-defined search vector. Moreover, if instead of one hyperparameter, the ANN needed ten instead, then the number of combinations grows exponentially and so does the GS time. Instead of probing all possible combinations, RGS probes a fixed number of combinations by randomly selecting sets of hyperparameters $\boldsymbol{\theta}$ from target distributions, $\boldsymbol{\Theta}_{\mathbf{k}}$. For the example defined previously, one might define a search over one hundred trials, where θ is randomly sampled from a normal or uniform distribution between $\theta_{\min}$ and $\theta_{\max}$. Even if the model depends on more hyperparameters, RGS allows the user to better control the number of optimization rounds needed by adapting this number to the available computational resources. It has been shown that, in general, RGS tends to provide better and faster results than GS [51] because it is more efficient in the exploration of the hyperparameter space.

This kind of optimization strategy is sometimes referred to as "uninformed search" because, for each optimization round, the process does not take into consideration the model's performance extracted from earlier iterations. Sophisticated methodologies that take advantage of previously probed values, i.e., that can "learn" how to best approximate a "good" hyperparameter value have also been developed for DL models over the last years [86]. Some examples are reinforcement learning [52,53], Genetic Algorithms [54–56] and Bayesian Optimization (BO) [57–62]. This tutorial focuses on hyperparameters optimization using BO techniques. Fig. 2 shows a schematic representation of the different hyperparameter search strategies mentioned.

2.3. Bayesian Optimization: tree-structured Parzen Estimators (TPE) and hyperband

In this section, the basic principles of BO and how to apply it to solve hyperparameter optimization is laid out in broad strokes. Commonly the problem of neural architecture search (NAS) and hyperparameter optimization is done separately but in Ref. [60], the authors showed that performing joint NAS and hyperparameter optimization can lead to better model performance when compared to the case where both operations are done separately. This tutorial is based on [22], follows the procedure described in Refs. [61,62] and implements a combination of Bayesian Optimization and Hyperband methods to perform joint automated optimization of a DL model architecture and its hyperparameters. The type of NAS shown in this tutorial can be considered as a low-level or constrained approximation to the full problem since the target architecture is pre-chosen (CNN). In a full NAS problem, as it is usually presented in ML research, multiple types of layers and their combinations are tested. The NAS result for a given problem can be a very complex architecture. The procedure proposed in this tutorial involves choosing a base architecture (e.g., a CNN), tweaking its structure slightly and optimize its hyperparameters in an automated way.

The procedure relies on the Hyperband [63] algorithm that optimizes hyperparameter search by considering a certain predefined computational budget B (e.g., CPU time, number of training epochs, number of iterations, etc.) and by dynamically allocating more resources to the most promising candidates. These candidates are picked by iteratively monitoring a predefined objective function f. This algorithm extends the capability of the Successive-Halving method [64]. For a CNN model optimization, the computational budget can be the training time, or the maximum number of training epochs, while the target objective function can be the model's validation loss (e.g., cross entropy loss in case of classification, or mean squared error for regression problems). This

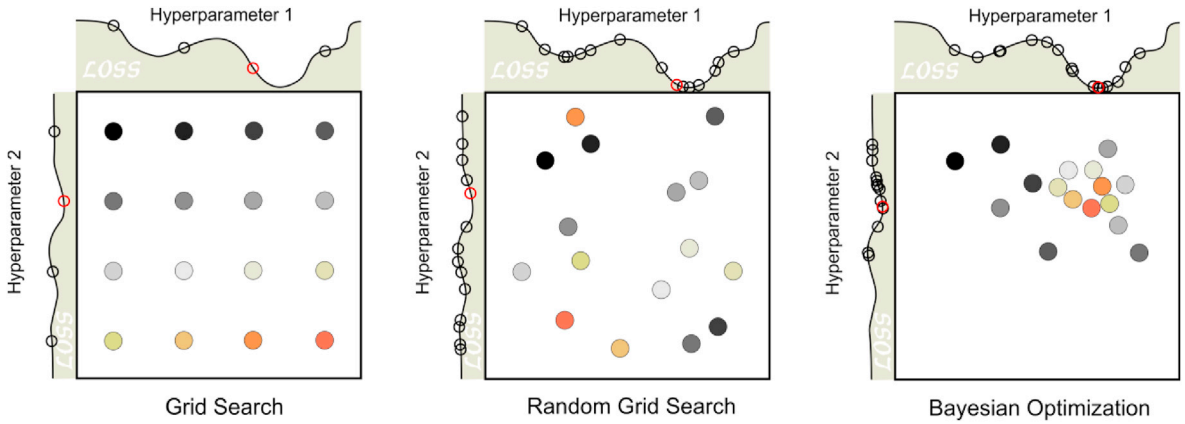

Fig. 2. Illustration of how hyperparameter space (over two hyperparameters) is populated by different search schemes. The colour gradient of the points, from black to "yellowish" to orange indicates the order of the search. In Grid Search, hyperparameters values are systematically predefined. In Random Grid Search, the hyperparameter space is randomly populated. In BO, the chosen hyperparameters values are progressively optimized to approximate a minimum. In this pictorial example, hyperparameter 1 has a larger influence in this "toy model" than hyperparameter 2, a fact that is reflected by the depth of the minima of the loss function. Image inspired by Ref. [51]. (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

algorithm works similarly to a dynamic Early Stopping strategy that interrupts training if the model (generated using a certain subset of hyperparameters) converges to a bad solution (low accuracy or high RMSE) and extends training if the model's performance improves (high accuracy or low RMSE). The default Hyperband algorithm uses something like a random search to select the hyperparameters configurations, therefore the process, like in RGS is also uninformed, i.e., there is no information passed from trial to trial. To focus the optimization process on areas of the hyperparameter space that show most promising, a Bayesian Optimization (BO) technique can be implemented in substitution of the random process that Hyperband uses.

BO is useful in the case where evaluating f is a very expensive task because it uses a "cheaper" probabilistic surrogate model, $p(f|H)$, to model the objective function f given a vector of observed points $H = \{(\boldsymbol{\theta}_0, y_0), \ldots, (\boldsymbol{\theta}_{i-1}, y_{i-1})\}$, where H is a history vector composed by pairs of hyperparameters $\boldsymbol{\theta}$ and objective evaluations y. It is generally assumed that the evaluation of $f(\boldsymbol{\theta})$ is subjected to some noise/uncertainty ε and therefore the actual evaluation is $y(\boldsymbol{\theta}) = f(\boldsymbol{\theta}) + \varepsilon$. Based on the history of the probed values, H, and an acquisition function, $a(\boldsymbol{\theta})$, a function that balances exploration and exploitation of the hyperparameter space to choose what values will be selected/"acquired" next, BO estimates the probability where good values might be for future searches. BO works iteratively by 1) sampling the next trial point that maximizes the acquisition function, $\boldsymbol{\theta}_{i+1} = \mathit{argmax}_{\boldsymbol{\theta}}\, a(\boldsymbol{\theta})$, 2) evaluate $f(\boldsymbol{\theta}_{i+1})$ and 3) append the new $(\boldsymbol{\theta}_{i+1}, y_{i+1})$ pair into H and update $p(f|H)$ accordingly. The expected improvement (EI) [82] over the previously best observation $\alpha = \min\{y_0, \ldots, y_n\}$, is a common choice for the acquisition function, $a(\boldsymbol{\theta})$, in BO algorithms

$$a(\boldsymbol{\theta}) = \int \max(0,\ \alpha - f(\boldsymbol{\theta}))dp(f|D). \tag{3}$$

In [61,62] the authors propose the use of a Tree-structured Parzen Estimator (TPE) [57], a BO method that uses kernel density estimators to estimate the probability densities associated with "good" and "bad" hyperparameter configurations

$$\begin{aligned} l(\boldsymbol{\theta}) &= p(y < \alpha|\boldsymbol{\theta}, H) \\ g(\boldsymbol{\theta}) &= p(y > \alpha|\boldsymbol{\theta}, H) \end{aligned}, \tag{4}$$

over the input configuration space instead of modelling the objective function f directly by implementing $p(f|H)$. To select the next hyperparameters $\boldsymbol{\theta}_{i+1}$ for evaluation, TPE maximizes the density ratio $l(\boldsymbol{\theta})/\, g(\boldsymbol{\theta})$ which is equivalent to maximizing the EI [57]. For example, considering that the objective function is the model's validation loss (e.g. mean squared error), for each trial, TPE looks for the hyperparameters that provide an evaluated y, lower than the best previously found value α, i.e., it works towards maximizing $l(\boldsymbol{\theta})$ and towards minimizing $g(\boldsymbol{\theta})$.

2.4. Learning rate range test

Learning rate (LR) in considered by many DL practitioners as one of the most important hyperparameter in a DL model. The LR can be interpreted as the size of the step that the gradient descent algorithm takes towards a loss function minimum during training, i.e., the guiding process for the optimization of the model's weights and biases. If the LR is two low, then the model will need many epochs to converge towards a loss function minimum, but if the LR is two high, the algorithm will overshoot the minimum and bounce chaotically around. A good LR is a compromise between allowing the model to converge fast (effectively needing less training epochs to reach a minimum) and to get as close as possible to the minimum value of the loss function. For this reason, it is sometimes useful to implement methods that allow for the LR to adapt during training, also called LR schedulers. In this section, a couple of these LR scheduler methods are used for both model training and for initial LR optimization. As a hyperparameter, LR can be optimized by any of the GS, RGS or BO methods mentioned earlier, however, to provide the reader with a wider range of optimization techniques, in this tutorial the LR is optimized separately from the other hyperparameters by a specific method. To form an idea about what is a good LR value for our model, the LR range test proposed in Ref. [49] was used. The LR range test implements a LR scheduler that dynamically adapts the LR during training and can find the LR interval where the model converges to a stable solution. The application of this method at the beginning of the optimization pipeline assumes that the LR is associated with the data and the overall architecture of the model, i.e., that LR range for a base CNN architecture behaves similarly to the final optimized one. This assumption might not hold true if the optimization process encompasses methods that allow for many changes in the architecture of the model (e.g. add layers iteratively). If the optimized architecture is very different from the architecture used for performing the LR range test, then one might need to revisit this point.

The interval where the validation loss decreases steadily with LR is the region of interest for this kind of test. This allows to find the minimum and maximum LR that produce stable solutions, i.e., LRs that allow the model to converge (see Fig. 6 in Section 3.1). One trick that sometimes helps speedup the optimization process is to use $\mathrm{LR}_{\max}$ from the range test as the initial LR for the stochastic gradient optimization algorithm (e.g., Adam). By choosing the largest reasonably stable LR, the model converges faster. Another technique that is employed extensively

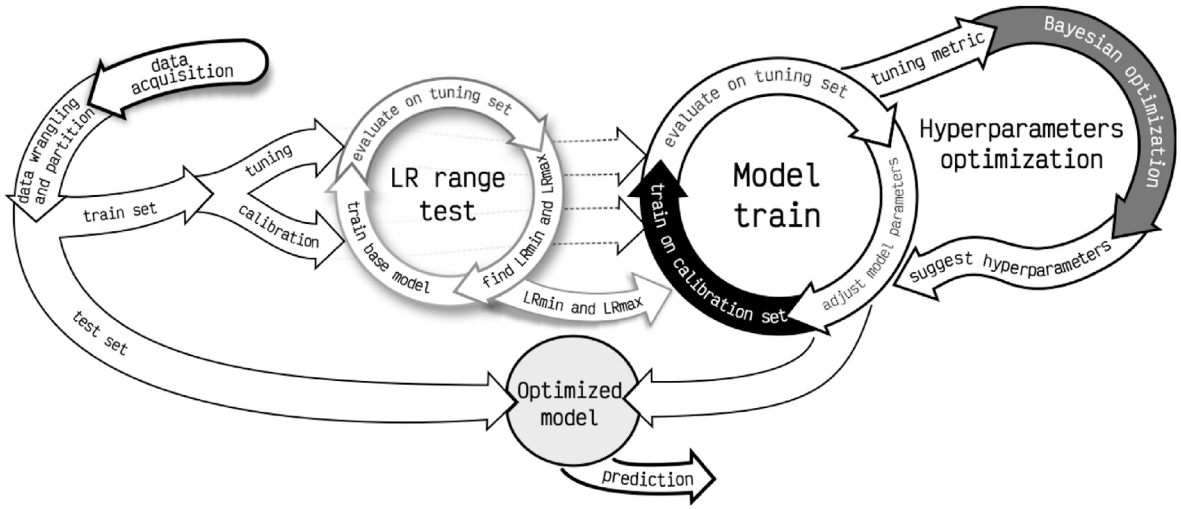

Fig. 3. Schematic representation of the optimization pipeline used in this tutorial. As usual, it starts with the "data acquisition" step, goes to optimization loops and ends with the optimized model prediction. Optimal hyperparameters are passed from the two loops on the right into the optimized model.

in DL model training and that allows models to converge better is to use a LR scheduler that modifies LR iteratively over the training process. The technique called "Reduce LR on Plateau" consists in monitoring the validation loss during training and, if after a predefined number of epochs, the validation loss doesn't improve by a certain predefined amount, the LR is automatically decreased by a fraction to help gradient descent approximate the minima. One can use the LR range test LR_{min} as the minimum value of the "Reduce LR On Plateau" scheduler. By using LR_{min} in this scheduler, the gradient descent algorithm doesn't waste time trying to optimize the CNN by using a LR too small (with which the model doesn't learn).

2.5. General rules and recommendations for model implementation and optimization

In the examples presented in this tutorial, NAS is restricted to the optimization of the number of FC layers, the number of units per FC layer and the addition (or not) of weights L2 regularization and dropout layers to the pre-chosen CNN base architecture. Also, for simplicity and to account for the efficient use of computational resources, some hyperparameters are pre-selected and others are subjected to optimization. The proposed optimization pipeline uses the TPE and Hyperband implementations available in the Optuna v.2.9.1 [65] package which is an open-source software library for hyperparameter optimization in machine learning. Other examples of popular libraries developed for the same purpose are Hyperopt [66] and GPyOpt [67]. The CNN models are implemented in *python* (3.6) using the TensorFlow/Keras (2.5.0) framework [68].

One way of improving the odds that the implemented DL model takes full advantage of the spectral data consists in performing a "data augmentation" in the variable space (similar to a specific type of "feature engineering"). This process takes advantage of chemometric methodologies that have already proven that certain types of spectra pre-processing methods help to remove redundant or noisy information from the data [30], e.g., derivatives [69], standard normal variate [70] correction or multiplicative scattering correction [71,72]. The process of finding the best type of pre-processing is time consuming but is a standard in almost all spectral modelling tasks. A strategy to overcome this step is to concatenate all the desired spectra pre-processed types into a single vector and feed that directly to the CNN [19,22]. This is possible because these models are not very affected by collinearity problems, and this also offsets the responsibility of picking the most key features from each pre-processing type from the input vector to the optimization algorithm. In principle, one could think that the model automatically performs the selection of the variables of the input spectra that best help solve the problem at hand. Although this type of data augmentation still needs further evidence to find how generalizable (to other data sets) the gains are, in both examples of this tutorial, this augmentation procedure is useful. There is also evidence in the literature [28,29] that the convolution layers in CNNs can perform automatic pre-processing tasks, therefore, in some cases, data augmentation in the variable space might not even be needed to achieve good results. In the case of small data sets, there are other types of data augmentation in the sample space that might help improve the stability of the training process [39] and even maybe the model's performance by using the novel method presented in Ref. [33]. The applicability of DL models to small data sets and their inherent advantages and disadvantages still needs further investigation.

After the input data is settled, the next step is to define a base CNN and run the LR range test. The proposed base CNN follows the following basic structure: 1 convolutional layer (with 1 filter and stride 1) and a flatten layer followed by 3 hidden dense layers and an output dense layer. The number of units of the hidden layers follow the empirical rule of thumb that the first layer of the dense block should have a units number between 1/2 and 1/5 of the output of the convolution block and that from dense layer to the next, this number should decrease by a factor of 2 or 3. This creates a funnel effect that forces the CNN to synthesize the information that is passed from layer to layer into higher abstraction levels [48]. The final layer configuration depends on the task at hand (regression or classification). By construction choice, all layers, with exception of the output ones, use L2 regularization and ELU activations functions. The optimizer algorithm chosen is Adam [73] and the weights in all layers are initialized using the "He normal" initialization [74] with a fixed seed (for reproducibility purposes). The train of all models use the "Reduce LR On Plateau" scheduler for improved stability and the "Early Stopping" method for decreasing problems related to overfitting and to speed up the training process. The latter method stops training if the performance of the model in the validation set, starts to degrade. These "training control techniques" are implemented as callback functions that the training algorithm "calls" while it runs. These choices are based on previous experience of the authors with this type of model, but, if the reader wants to adapt the tutorial models to their own needs/data, the modification of these choices are straightforward.

In the next section, these common CNN principles are extended to two practical examples of how to apply the proposed optimization pipeline depicted in Fig. 3. For each example, the practical problem and objective is initially described and then the optimization steps are presented. In the first example (a regression task), a pre-defined CNN is implemented and, for instructive purposes, 4 of its hyperparameters are optimized. In the

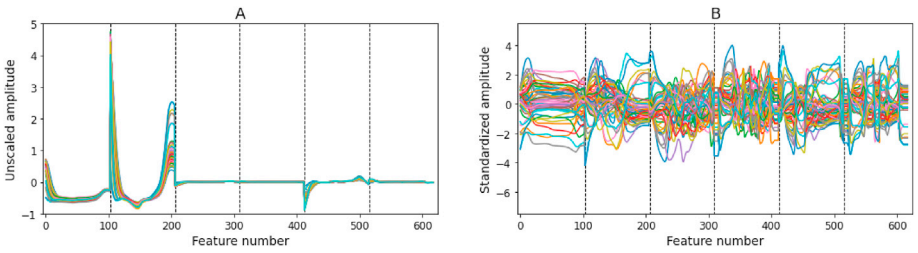

Fig. 4. (A) 50 samples of the augmented spectra of the tuning set composed by the original spectra concatenated with its SNV, 1st derivative, 2nd derivative, SNV + 1st derivative and SNV + 2nd derivative preprocessed versions. (B) The same spectra after column-wise standardization.

second example (a classification task), the level of complexity increases a bit and the optimization includes hyperparameters and CNN's architecture. These descriptive sections are complemented by additional technical notes and implementation code shared in the form "jupyter notebooks" [85], a web-based interactive computing platform, very common in Data Science and ML that combines live code, equations, narrative text, visualizations, etc. All code presented is commented for an easier understanding of the underlying details and includes some extra technical notes.

3. Tutorial examples

3.1. Regression problem: predicting mango dry matter content

The first problem consists in optimizing a CNN to predict the dry matter (DM) content in mango using Vis-NIR spectra. The original mango data set is composed of 4 harvest seasons 2015, 2016, 2017 and 2018 [75], and further detailed in Refs. [76,77]. The data set has Vis-NIR spectra (with 103 features per spectrum) of mango (the input variables, x) and corresponding dry matter content (DM in percentage units, %) measured in laboratory (target variables, y). The data set provided for this example was further pre-processed by performing an additional outlier removal and by doing data augmentation in the variable space by concatenating spectra, SNV, 1st derivative, 2nd derivative, SNV +1st derivative and SNV +2nd derivative as explained in Ref. [19]. The data is separated into train (first three seasons) and test (n = 1448, the last season) sets. The initial training set is further split into calibration (n = 6642) and validation/tuning (n = 3272) sets. The remaining of the manuscript follows the typical chemometric nomenclature where model training is done using the calibration set and the validation set is renamed as tuning set to emphasize the fact that, in the context of this tutorial, the validation data is used for tuning the model's hyperparameters. As usual the test set is used only after model optimization to access the model's final performance. The data partition used here is the same as [19,76,77] to facilitate the comparison of results. For this specific chemometric problem related with fruit spectra, one of the main objectives is to produce models that are robust across harvest seasons. In this context, robustness is used as a measure of reproducibility, i.e., a robust model is a model that maintains its performance in new datasets of the same type. That is why the test set used here corresponds to the last harvest season (2018).

Before feeding these spectral samples to the CNN it is convenient to standardize them column-wise (zero mean and unit standard deviation). The standardization of the tuning and test sets are done using the mean and standard deviation of the calibration set. By doing this the input data, Fig. 4A, takes the form presented in Fig. 4B.

The following step is to define the CNN model by applying and expanding the general rules defined in Section 2.4 and 2.5. The used CNN has one convolution layer (1 filter, stride = 1), followed by three hidden dense layers having a corresponding number of units (128, 64, 16)], respectively. Since this is a regression problem, the output layer has a single unit that will represent the value of the DM the model is trying to predict. To this end the linear activation function is used in the last layer. L2 regularization and weights initialization is done as defined in Section 2.5. In this example, the architecture of the CNN is fixed and the only two model's hyperparameters that will be optimized in a further step are the width of the conv. filter (FILTER_SIZE) and β, the strength of the L2 regularization (L2_BETA). Two other hyperparameters related to the training process, namely the training batch size (BATCH, how many samples are used in each training epoch) and the Adam optimizer learning rate (LR), will also be object of optimization. Since this is a regression problem, the chosen loss function is the Mean Squared Error (MSE)

$$MSE\left(y_{true},\ y_{pred}\right) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} \left(y_{true} - y_{pred}\right)^2, \tag{5}$$

and the metric to monitor during training is the tuning (or validation) MSE. To have an idea about the performance of the unoptimized model on the data set, a baseline train over 450 epochs is done by choosing some common values for these hyperparameters. For this first run, the following values are used: FILTER_SIZE = 5 (small conv. filter as in Ref. [28]), L2_BETA = 0.0006 (weak regularization), BATCH = 256, LR = 0.01*BATCH/256 (heuristic proposed in Ref. [28]). After the model train is complete (see Fig. 5), one can use it to compute some usual baseline metrics on the data using the Root Mean Squared Error (RMSE): calibration RMSE = 0.259%, tuning RMSE = 0.572% and test RMSE = 1.068%. Error metrics are presented in the units of the DM target variable, in this case %.

In this case the RMSE on the calibration set is much smaller than the RMSE on the two other sets. This is indicating that this model might be suffering from overfitting. This kind of intermediate check-up can help decide what type of objective function will be useful to implement during the optimization process (more on this later).

By construction, the optimization of the LR is done first and separately from the other hyperparameters using the LR range test to find the appropriate LR interval where the model converges. The CNN architecture defined earlier is used to run the LR range test (for 450 epochs), and the analysis of the test helps set LR_{min} and LR_{max}. Those values will be later used in "Reduce LR on Plateau" and in the Adam optimizer, respectively. This LR range test gives us an idea how the model "learns" or performs as a function of LR. In Fig. 6 the result of the test and the LR interval chosen are presented.

In the beginning of the test (lower LR) the model has a high validation loss, meaning that during the 450 epochs used the model is unable to learn anything. The larger the LR gets, the better is the improvement and the validation loss starts to decrease around $LR = 1 \times 10^{-6}$ until it reaches a valley from $LR \approx 1 \times 10^{-4}$ to LR = 0.3. For values a bit above this last value, the model becomes unstable and the error shoots up. The main aim of the test is to find the initial value used by the Adam optimizer. The choice is done by selecting a large LR (that makes training faster) but that still ensures stability, i.e. where the model converges toward low error solutions. In this case the chosen value is $LR_{max} = 2.5 \times 10^{-3}$. As a bonus, the test also helps define $LR_{min} = 1 \times 10^{-6}$ that will be used to define the smallest LR that "Reduce LR on Plateau" will allow.

With the LR settled, the other three hyperparameters will be opti-

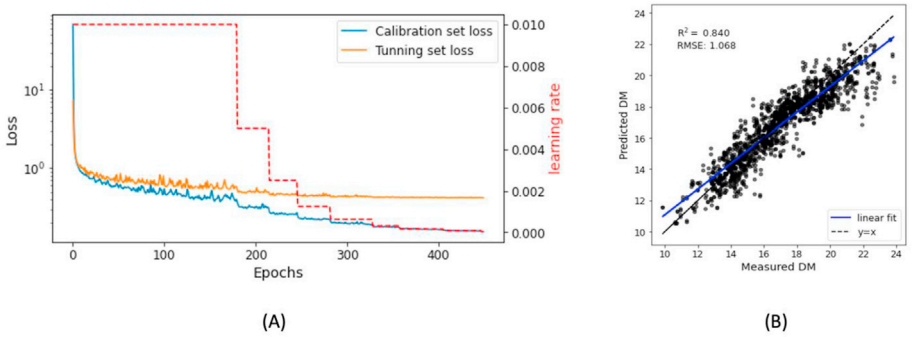

Fig. 5. (A) Train history depicting how the MSE is dropping during a training session, for the calibration data (blue) and for the tuning data (orange). The red dashed stepwise line shows how the LR is being dynamically decreased during training by the "Reduce LR on Plateau" callback. (B) the base line prediction for the test set. (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

mized by BO. The following step consists in defining a function "objective()" that has information about what hyperparameters are going to be optimized, the search interval for each of them and the target objective function that will be monitored. Since this is a regression problem, a standard option for the target objective function would be the RMSE on the tuning set (i.e. the validation loss). However, based on the observation done earlier that the base CNN model might be suffering from a slight overfitting, a more suited optimization strategy involves choosing a target objective function that helps decrease this problem. The motivation is that models that overfit less, usually generalize better. Therefore, the target objective function of the optimization loop can be "engineered" towards low error in the tuning set and smaller overfit (difference between calibration and tuning errors/losses). A possible solution is implemented in the form

$$\text{target objective function} = w_1 \ \text{RMSE}_{tuning} + w_2 \left|\text{MSE}_{tuning} - \text{MSE}_{cal}\right|, \tag{6}$$

i.e., a weighted sum of tuning error and overfitting. For simplification purposes $w_1 = w_2 = 0.5$ is adopted, but other combinations could be defined depending on an acceptable compromise between both factors. Moreover, the w_1 and w_2 factors could also be included in the optimization pipeline. The optimization loop is implemented in Optuna under the form of a "study()" object where the optimization method (TPE and Hyperband), the direction of optimization of the function "objective()" (minimization in this case) and further storage and logging options are defined. The optimization of the "study()" can then begin for a predefined number of trials (number of sets of different hyperparameters). During this process, the optimization pipeline will test different sets of hyperparameters and choose those that work towards minimizing the target objective function.

One of the things to have in mind is that the TPE algorithm implemented in the Optuna library is initialized by randomly probing the hyperparameters space for a predefined number of trials (50 start-up trials in this case), (effectively equivalent to a random grid search) to ensure hyperparameter space exploration and, only after that number of trials is completed, it starts to effectively use the TPE method to exploit the most promising corners of the hyperparameter space. It is therefore recommended to start the optimization with a large enough number of start-up trials (e.g., from 10–15 per hyperparameter, to ensure good exploration) and use a much larger number of trials to improve exploitation (450 additional trials in this example). A good exploratory start might help avoiding local minima. During the optimization process, the user has the option to save all tested models. The tutorial notebook includes additional technical notes on how this is implemented. After the optimization is finished, one can query the optimized "study()" and check what trial/hyperparameters provided the best results. If each tested model was saved, the user just needs to instantiate the CNN using the optimal hyperparameters, load the weights from the best trial into the CNN and apply it to the test set for a final evaluation of its performance. If the models were not saved during the optimization, the user will first need to instantiate the CNN with the optimized hyperparameters and then re-train the model from scratch before applying it to the test data set. Code for both options is provided in the regression tutorial notebook.

The implementation of this optimization in the companion tutorial notebook, runs for 500 trials (by default). Depending on the computational resources available, this can take a long time to compute (a couple of days in a modern workstation). For practical purposes, the outputs of this optimization (the optimized "study()" and pre-trained models) are also shared with the notebook files. Information on how to load the optimized "study()" and how to access the pre-trained models is included in the tutorial notebook.

The optimization found that the best trial was number 156, achieving a target objective value of 0.348 obtained using FILTER_SIZE = 45, L2_BETA = 0.015, and BATCH = 128. The optimized CNN metrics are calibration RMSE = 0.501%, tuning RMSE = 0.588% and test RMSE = 0.853%. The main objective of the optimization was successfully accomplished, i.e., to lower tuning RMSE and avoid overfitting (the difference between calibration and tuning errors decreased substantially). Because of this, the model generalizes better and achieves a lower RMSE on the test set. The fact that the tuning RMSE is still a bit higher than the calibration RMSE indicates that one might get even better performance by changing a bit the weight factors (w_1 and w_2) of the target objective function towards more overfitting correction. In most cases it is beneficial to aggregate back the calibration and tuning sets into a single training set and retrain the model one last time using all the available training data and the optimized hyperparameters. An implementation of this final step and accompanying technical details are presented in the end of the complementary notebook. By retraining the optimized model on the full training set (calibration + tuning) the performance improved further achieving a test RMSE = 0.838%.

As a final remark, it might be useful to remember that this problem deals with inter-seasonal predictions of fruit properties, which makes it a hard problem to solve due to the biological variability of fruit. Nonetheless, the results obtained here using a simple CNN architecture and the proposed optimization of only 4 hyperparameters (for 500 trials) led to results on par with the best result presented in the broad machine learning benchmark of [77] obtained using an ensemble of models (see their Table 3). For this specific type of problem, a much more laborious optimization strategy that involves a more complex target objective function like the one presented in Ref. [19] can lead to even better results.

3.2. *Classification problem: classifying wheat kernel varieties*

The second example in this tutorial deals with a classification problem. The aim was to use NIR spectra information to build a classifier model capable of classify individual wheat kernels into one of 30

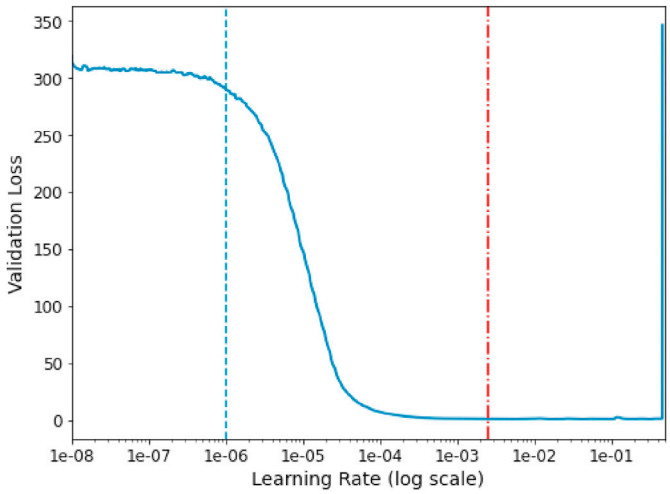

Fig. 6. Result of the LR range test and the picking of LR_{min} (blue dashed line) and LR_{max} (red dash-dotted line). (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

different classes. The data set was first introduced in Ref. [78], is well balanced with equal number of kernel varieties and includes a total of 147096 samples. The NIR data of the kernels were extracted from hyperspectral (HS) images running in the spectral range of 874–1734 nm (with 200 features). This classification example is based on Ref. [22] that served as motivation for this tutorial and shares many common points. To avoid overlapping of information, this tutorial touches more practical aspects of the pipeline implementation.

Since the same method applied in the regression example is also followed here, the emphasis will be on the differences between both cases. After importing the original data set, data augmentation in the variable space was performed using the method described earlier. The same sequence of 6 pre-processing methods were used, increasing the number of input features (per spectrum) from 200 to 1200. The spectra were standardized column-wise and the target classes/labels were encoded from its numerical values, [0, …, 29], to one-hot-encode representation (e.g. class 0 is represented as [1,0,0,0, …], class 1 as [0,1,0,0, …], etc.), more suited to classification problems. As usual, the data was partitioned into calibration (n = 72000), tuning (n = 24000) and test (n = 39096) sets. This data split is the same used in earlier publications therefore allowing comparison between model results. Following the established stream of optimization tasks, a base CNN architecture was defined, and the LR range test finder was performed. This time, the number of hyperparameters to be optimized is larger than the four used in the previous example. The predefined hyperparameters used for the base CNN architecture are presented in Table 1. These include hyperparameters specifically related to model's hidden layers, architecture (NAS) and model training. As a clarification we note that the number of Dropout layers as presented in Table 1 is not optimized directly. The NAS part of the pipeline adds/tests pairs of FC+Dropout layers simultaneously and optimizes the number of units per FC layer and Dropout Rate (DR) per Dropout layer. The number of Dropout layers is defined as the number of Dropout layers whose DR $\neq 0$. The last layer of the CNN has 30 units to account for the 30 classes (wheat varieties) in the data set and uses the 'Softmax' activation function.

"Softmax" is a generalization of the multi-class logistic regression function and transforms a vector of k real values into a vector of k real values between 0 and 1, whose sum is 1 and that can be effectively interpreted as probabilities. This function can be represented by

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_i}} \tag{7}$$

where k is the number of classes in the problem, $z = (z_0, z_1, \ldots, z_k)$ is the input vector to the function (the output of the preceding layer), z_i are elements of z, and the denominator acts as a normalization term. The Adam optimizer was used once again and the chosen loss function was the categorical cross entropy (which is a usual choice for multi-class problems). Categorical cross-entropy is basically a Softmax activation followed by a Cross-Entropy loss (CE)

$$CE = -\sum_{i=1}^{k} y_{o,i} \log\left(\sigma(z)_i\right) \tag{8}$$

where k is as before the number of classes and, $y_{o,i}$ are elements the one-hot-encode representation of the training labels. The metric being monitored was the model's accuracy on the tuning set. The choice of accuracy for this data set is possible because this is a well-balanced data set. In the case of unbalanced data sets, other fairer metrics such as F1 or ROC should be chosen instead. The results of the LR test range performed for 400 epochs, using a predefined training batch size = 512 allowed once again to define LR_{min} and LR_{max}, for the base model. In this case the values found were $LR_{min} = 1 \times 10^{-7}$ and $LR_{max} = 2.5 \times 10^{-3}$. With the LR already optmized, a base model performance was tested by training the CNN over a maximum of 550 epochs. Around epoch 420, the model's accuracy stopped improving and the "Early Stopping" method terminated the training. The metrics obtained for the base model were, train set Accuracy = 0.930, tuning set Accuracy = 0.91 and test set Accuracy = 0.909.

Next, the function "objective()" was defined considering the hyperparameters search intervals depicted in Table 1 and the target objective function defined as the accuracy of the tuning set. A practical note on how to expand the number of hyperparameter in the optimization, including categorical ones, is also mentioned on the corresponding accompanying notebook file. The objective function was incorporated into a "study()" that Optuna optimized using TPE and Hyperband, towards maximizing the target objective function, i.e., the accuracy. The maximum number of training epochs allowed during the optimization process was 450. The example shown in the accompanying tutorial notebook performs an optimization over 500 trials, with a warm-up phase (random search) of 50 trials. In practice, the implementation of a larger warm-up phase and more optimization trials is advised to ensure a good exploration of the hyperparameter space. The best model was found at trial = 454 for the hyperparameters presented in Table 2. The metrics for this optimized CNN were calibration accuracy = 0.990, tuning accuracy = 0.949 and test accuracy = 0.949. The optimization pipeline was able to find a suitable neural network architecture (subjected to the imposed constraints) and optimized its hyperparameters to obtain a state-of-the-art accuracy.

Just like in the previous tutorial example, the calibration and tuning data are concatenated into a single train set and the optimized model is trained using all that available data. The model with the optimized hyperparameters is initialized by loading the weights of the best model obtained in the optimization phase and trained for an additional 550 epochs. By doing this, the model starts from an already good set of weights and uses the additional data available for training (the samples previously use for tuning) to zoom in into a better minimum. After this final training process, the accuracy on the test set improved to 0.952, beating the 0.930 obtained by Ref. [78] and even improving the 0.949 of [22].

If the user wants to reach a higher insight of how the model operates and what is the role of each of the hyperparameters, an *a posteriori* analysis of the optimization is recommended. To help in this task, Optuna supplies several pre-conceived graphical tools that can be used to explore the hyperparameter space that the optimization probed. A couple of instructive types of plots that one can generate are presented in Fig. 7.

From this type of plots/analysis, the user can learn valuable information about the model such as what hyperparameters have a higher

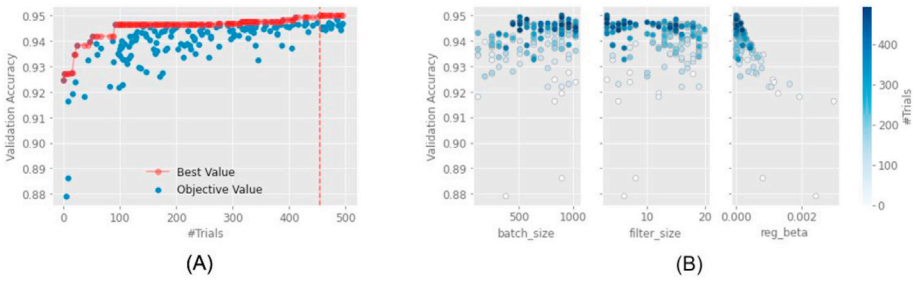

Fig. 7. (A) the evolution of the accuracy on the tuning set (the target objective function) during the optimization process. The red dashed vertical line represents the optimal trial. (B) example of the spread of hyperparameters values probed by the BO for the batch size, filter size and regularization strength, β. Trial number is encoded as colour intensity, from lighter to darker blues. (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

influence in the error metric being monitored, what range of values seem the most promising, etc. This post-optimization interaction is useful for building up an intuition about the model's behaviour, an intuition that will be very important for other future model applications.

3.3. Classification problem: gradCAM implementation and spectral bands identification

Delving deeper into hyperparameter dynamics can give the user a better insight about the model and how to optimize it further. Nonetheless, for "Chemometricians" and researchers that use spectroscopy in general, it might be more interesting to probe the model to extract information about what spectral bands contribute more to the distinction between samples classes (wheat varieties in the example). Gaining insight from DL models is not as straight forward as it is for simpler models like PLS, where the user just must plot the VIP scores to find the relevant spectral bands. The field of explainable AI (XAI) is filled with challenges and opportunities and some tools/methods that allow us to deal with DL models beyond a "black-box approach", are currently being developed. For spectral data specifically, model interpretability can also be aided by plotting the "regression coefficients" of a CNN model as proposed by Ref. [28] or using Class Activation Mapping (CAM) [88] as exemplified in Ref. [29]. CAN techniques have been developed for the "object identification" problem in computer vision. This is the case of Gradient-weighted Class Activation Mapping (grad-CAM) [79], a technique that is usually used to infer what parts of an image contributed most of the classification process. In what remaining of this tutorial, an example is presented on how to apply a customized 1D version of the grad-CAM method to the wheat CNN model to infer information about the spectral bands that contributed the most to the classification. The assumption here is that this can be useful for the used CNN architecture because grad-CAM computes the gradient of the classification error backwards from the output layer, all the way to the last convolution layer and by that process, it traces back the importance of the most relevant key features extracted by the conv. layer. For this specific CNN architecture, with a single conv. layer with 1 kernel and a stride of 1, the output features of the conv. layer map the input values directly, and because of that, a relationship with the wavelengths can be established directly. For CNN architectures with two or more conv. layers, an input spectrum is compacted into higher and higher abstractions from layer to layer, and it might not be possible to trace back what features of the last conv. layer map into what input wavelengths. This is still a subject for further research in the deep learning spectral data domain. Also, it is useful to remember that feature combination in CNN models is non-linear, unlike e.g., PLS. The visualization of important spectral features from a CNN model might not represent the same linear combination of information that a classical PLS VIP score plot provides.

The grad-CAM function is implemented in the initial Help section of the classification tutorial notebook. To exemplify the application of grad-CAM, the first spectra of five different wheat classes were separated into a small target subset. The grad-CAM function uses the optimized CNN model for predicting the classes of the target spectra and, during the process finds what output features of the conv. layer are more relevant. The output of this process is a heat map for each of the tested spectra, where higher scores correspond to greater feature's importance. These scores are them encoded as colour and overlaid to the initial target spectra for an easier visual perception. Fig. 8 shows the result of this process. For example, this type of analysis may help to gain insight about what type of pre-processed features the CNN relies on the most for the classification process, or what kind of spectral bands are more useful to distinguish between two different wheat varieties.

4. Final remarks

This work supplied a hands-on tutorial for implementing an optimization pipeline tailored to deal with DL models and spectral data. This included suggestions on how to prepare the data and different hyperparameter optimization techniques distributed between two examples, one for a regression problem and another for a classification problem. Two real cases of modelling Vis-NIR spectral data were showed. Furthermore, approaches to understand the learning of deep spectral models by exploring feature importance was also showed. The optimization methodology presented is very general and can be easily adapted to more complex CNN architectures and other types of DL models.

In the regression example, it was shown that automatically optimized models can be simpler and generalize better when compared to models trained with ad-hoc selected hyperparameters. When comparing unoptimized and optimized models, in the regression case, the prediction error was decreased by up to 25%. The same exercise for the classification case showed that the classification accuracy was increased by ~5% after optimization, obtaining a new state of the art classification score for this data set. All of this was achieved using simple/shallow deep learning models with only a one convolutional layer and a handful of dense layers.

All the analysis presented in this study is reproducible (assuming the same software versions is used) and can be performed using the provided *python* notebooks available at the author's GitHub repository: https://github.com/dario-passos/DeepLearning_for_VIS-NIR_Spectra/tree/master/notebooks/Tutorial_on_DL_optimization. Due to the randomness inherent to the stochastic gradient descent algorithm during training, it is expected that the results might vary slightly on different systems specially if GPUs are used. To get reproducible results, it is necessary to fix the numeric random seed values of the functions that use random processes. This is of special importance in the initialization of the weights of the CNN model. However, this means that model's hyperparameters are optimized for a specific set of initial weights. Although this topic was not covered in this tutorial, model robustness can be improved by considering model different initialization weights during the optimization process. By forcing the model to be optimized for many different sets of initial weights, one might find solutions that generalize better at a cost of having to explore a larger hyperparameter space [22,90].

Instructions on how to setup the *python* environment and what libraries need to be installed are available in the GitHub repository as well. It is advised that the reader tries to reproduce the analysis presented in this manuscript by running the notebooks in parallel with the reading, for a more fruitful learning experience. Furthermore, after doing this tutorial, the readers can change the scripts according to their needs to

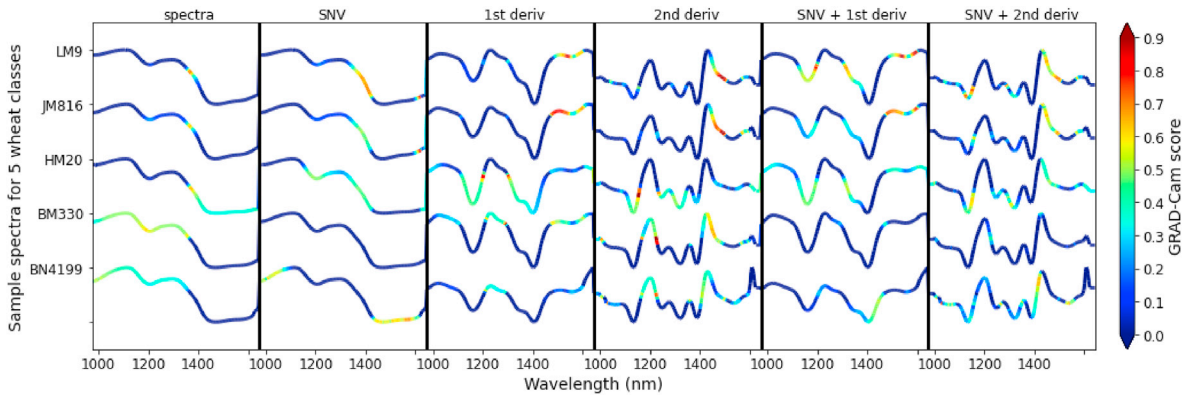

Fig. 8. The output of the grad-CAM technique for five wheat classes. Scores are encoded as colour and overlaid on the original augmented spectra. The higher (towards dark red) the grad-CAM scores the higher the importance of the spectral region. (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

Table 1
Intervals used in the optimization of hyperparameters related to neural architecture search (NAS), model layers (L) and model training (M). Learning rate was the only hyperparameter optimized separately.

Name	Type	Interval/step	Base CNN
Number of hidden FC layers	NAS	[1–5]/1	3
Number of units p/FC layer	L	[128–512]/2	[512, 256, 128]
Conv. filter size	L	[3–20]/1	5
Number of Dropout layers	NAS	[1–5]/1	0
Dropout rate p/Dropout layer	L	[0–1]/0.005	0
L2 regularization β	L	[0–0.003]/0.00001	0.003
Batch size	M	[128–1024]/64	512
Learning rate*	M	$[1 \times 10^{-9} - 0.1]$	

Table 2
Optimized neural architecture and hyperparameters obtained using the proposed automated optimization.

NA/Hyperparameters	Values for Optimized CNN
Number of hidden FC layers	5
Number of units p/hidden FC layer	[360, 350, 132, 442, 334]
Conv. filter size	3
Number of Dropout layers	5
Dropout Rate p/Dropout layer	[0.555, 0.165, 0.13, 0.46, 0.085]
L2 regularization β	1×10^{-5}
Batch size	896
Learning rate in Adam	$LR_{max} = 2.5 \times 10^{-3}$, $LR_{min} = 1 \times 10^{-7}$

develop new applications of deep learning for spectral data modelling. In case of difficulty in implementing scripts, the reader can contact the authors.

As a final note, we would like to add that the area of deep learning research is evolving at a vertiginous pace and even the experts on the field have difficulties tagging along. That neck-breaking rhythm should not discourage the exploration and adoption of these techniques into other fields. Classical fields such as chemometrics and analytical chemistry can profit immensely from all these new technologies, if a few curious researchers venture into porting the necessary DL knowledge into their projects. This tutorial intends to be an incentive, a starting point, and a small road map to those brave explorers.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors thank the anonymous referees for their thorough reviews that helped improve the overall quality of the work. D. Passos acknowledges FCT - Fundação para a Ciência e a Tecnologia, Portugal, for funding CEOT project UIDB/00631/2020 CEOT BASE and UIDP/00631/2020 CEOT PROGRAMÁTICO.

References

[1] C. Pasquini, Near infrared spectroscopy: a mature analytical technique with new perspectives – a review, Anal. Chim. Acta 1026 (2018) 8–36.
[2] P. Mishra, S. Lohumi, H. Ahmad Khan, A. Nordon, Close-range hyperspectral imaging of whole plants for digital phenotyping: recent applications and illumination correction approaches, Comput. Electron. Agric. 178 (2020) 105780.
[3] W. Saeys, N.N. Do Trong, R. Van Beers, B.M. Nicolai, Multivariate calibration of spectroscopic sensors for postharvest quality evaluation: a review, Postharvest Biol. Technol. (2019) 158.
[4] P. Mishra, A. Nordon, J.-M. Roger, Improved prediction of tablet properties with near-infrared spectroscopy by a fusion of scatter correction techniques, J. Pharmaceut. Biomed. Anal. (2020) 113684.
[5] N. Fuenffinger, S. Arzhantsev, C. Gryniewicz-Ruzicka, Classification of ciprofloxacin tablets using near-infrared spectroscopy and chemometric modeling, Appl. Spectrosc. 71 (2017) 1927–1937.
[6] L.M. Kandpal, J. Tewari, N. Gopinathan, J. Stolee, R. Strong, P. Boulas, B.-K. Cho, Quality assessment of pharmaceutical tablet samples using Fourier transform near infrared spectroscopy and multivariate analysis, Infrared Phys. Technol. 85 (2017) 300–306.
[7] P. Mishra, M.S.M. Asaari, A. Herrero-Langreo, S. Lohumi, B. Diezma, P. Scheunders, Close range hyperspectral imaging of plants: a review, Biosyst. Eng. 164 (2017) 49–67.
[8] A. Martyna, A. Menżyk, A. Damin, A. Michalska, G. Martra, E. Alladio, G. Zadora, Improving discrimination of Raman spectra by optimising preprocessing strategies on the basis of the ability to refine the relationship between variance components, Chemometr. Intell. Lab. Syst. 202 (2020) 104029.
[9] C. Zhu, Y. Kanaya, R. Nakajima, M. Tsuchiya, H. Nomaki, T. Kitahashi, K. Fujikura, Characterization of microplastics on filter substrates based on hyperspectral imaging: laboratory assessments, Environ. Pollut. 263 (2020) 114296.
[10] J.M. Amigo, H. Babamoradi, S. Elcoroaristizabal, Hyperspectral image analysis. A tutorial, Anal. Chim. Acta 896 (2015) 34–51.
[11] A.A. Gowen, C.P. O'Donnell, P.J. Cullen, G. Downey, J.M. Frias, Hyperspectral imaging – an emerging process analytical tool for food quality and safety control, Trends Food Sci. Technol. 18 (2007) 590–598.
[12] K.B. Walsh, J. Blasco, M. Zude-Sasse, X. Sun, Visible-NIR 'point' spectroscopy in postharvest fruit and vegetable assessment: the science behind three decades of commercial use, Postharvest Biol. Technol. 168 (2020) 111246.
[13] R.A. Crocombe, Portable spectroscopy, Appl. Spectrosc. 72 (2018) 1701–1751.
[14] P.P. Subedi, K.B. Walsh, Assessment of avocado fruit dry matter content using portable near infrared spectroscopy: method and instrumentation optimisation, Postharvest Biol. Technol. (2020) 161.
[15] M. Li, Z.Q. Qian, B.W. Shi, J. Medlicott, A. East, Evaluating the performance of a consumer scale SCiO (TM) molecular sensor to predict quality of horticultural products, Postharvest Biol. Technol. 145 (2018) 183–192.
[16] R. Bro, A.K. Smilde, Principal component analysis, Anal. Methods 6 (2014) 2812–2831.
[17] P. Geladi, B.R. Kowalski, Partial least-squares regression: a tutorial, Anal. Chim. Acta 185 (1986) 1–17.
[18] S. Wold, PLS Modeling with Latent Variables in Two or More Dimensions, 1987.

[19] P. Mishra, D. Passos, A synergistic use of chemometrics and deep learning improved the predictive performance of near-infrared spectroscopy models for dry matter prediction in mango fruit, Chemometr. Intell. Lab. Syst. (2021) 104287.
[20] P. Mishra, D. Passos, Deep chemometrics: validation and transfer of a global deep near-infrared fruit model to use it on a new portable instrument, J. Chemometr. (2021), e3367 n/a.
[21] P. Mishra, D. Passos, Realizing transfer learning for updating deep learning models of spectral data to be used in a new scenario, Chemometr. Intell. Lab. Syst. (2021) 104283.
[22] D. Passos, P. Mishra, An automated deep learning pipeline based on advanced optimisations for leveraging spectral classification modelling, Chemometr. Intell. Lab. Syst. 215 (2021) 104354.
[23] M. Puneet, D. Passos, Deep multiblock predictive modelling using parallel input convolutional neural networks, Anal. Chim. Acta (2021) 338520.
[24] P. Mishra, R. Sadeh, E. Bino, G. Polder, M.P. Boer, D.N. Rutledge, I. Herrmann, Complementary chemometrics and deep learning for semantic segmentation of tall and wide visible and near-infrared spectral images of plants, Comput. Electron. Agric. 186 (2021) 106226.
[25] P. Mishra, I. Herrmann, GAN meets chemometrics: segmenting spectral images with pixel2pixel image translation with conditional generative adversarial networks, Chemometr. Intell. Lab. Syst. 215 (2021), 104362, https://doi.org/10.1016/j.chemolab.2021.104362.
[26] P. Mishra, D. Passos, Multi-output 1-dimensional convolutional neural networks for simultaneous prediction of different traits of fruit based on near-infrared spectroscopy, Postharvest Biol. Technol. 183 (2022) 111741.
[27] P. Mishra, D. Passos, Deep calibration transfer: transferring deep learning models between infrared spectroscopy instruments, Infrared Phys. Technol. 117 (2021) 103863.
[28] C. Cui, T. Fearn, Modern practical convolutional neural networks for multivariate regression: applications to NIR calibration, Chemometr. Intell. Lab. Syst. 182 (2018) 9–20.
[29] X. Zhang, T. Lin, J. Xu, X. Luo, Y. Ying, DeepSpectra: an end-to-end deep learning approach for quantitative spectral analysis, Anal. Chim. Acta 1058 (2019) 48–57.
[30] P. Mishra, A. Biancolillo, J.M. Roger, F. Marini, D.N. Rutledge, New data preprocessing trends based on ensemble of multiple preprocessing techniques, Trac. Trends Anal. Chem. (2020) 116045.
[31] T. Mehmood, S. Sæbø, K.H. Liland, Comparison of variable selection methods in partial least squares regression, J. Chemometr. (2020), e3226 n/a.
[32] T. Mehmood, K.H. Liland, L. Snipen, S. Sæbø, A review of variable selection methods in Partial Least Squares Regression, Chemometr. Intell. Lab. Syst. 118 (2012) 62–69.
[33] U. Blazhko, V. Shapaval, V. Kovalev, A. Kohler, Comparison of augmentation and pre-processing for deep learning and chemometric classification of infrared spectra, Chemometr. Intell. Lab. Syst. 215 (2021) 104367.
[34] W.J. Melssen, J.R.M. Smits, L.M.C. Buydens, G. Kateman, Using artificial neural networks for solving chemical problems: Part II. Kohonen self-organising feature maps and Hopfield networks, Chemometr. Intell. Lab. Syst. 23 (1994) 267–291.
[35] Z. Xin, S. Jun, T. Yan, C. Quansheng, W. Xiaohong, H. Yingying, A deep learning based regression method on hyperspectral data for rapid prediction of cadmium residue in lettuce leaves, Chemometr. Intell. Lab. Syst. 200 (2020) 103996.
[36] X.J. Yu, H.D. Lu, D. Wu, Development of deep learning method for predicting firmness and soluble solid content of postharvest Korla fragrant pear using Vis/NIR hyperspectral reflectance imaging, Postharvest Biol. Technol. 141 (2018) 39–49.
[37] X. Yu, H. Lu, Q. Liu, Deep-learning-based regression model and hyperspectral imaging for rapid detection of nitrogen concentration in oilseed rape (Brassica napus L.) leaf, Chemometr. Intell. Lab. Syst. 172 (2018) 188–193.
[38] Z. Shen, R.A. Viscarra Rossel, Automated spectroscopic modelling with optimised convolutional neural networks, Sci. Rep. 11 (2021) 208.
[39] E.J. Bjerrum, M. Glahder, T. Skov, Data Augmentation of Spectral Data for Convolutional Neural Network (CNN) Based Deep Chemometrics, 2017 arXiv preprint arXiv:1710.01927.
[40] S. Malek, F. Melgani, Y. Bazi, One-dimensional convolutional neural networks for spectroscopic signal regression, J. Chemometr. 32 (2018), e2977.
[41] C. Ni, D. Wang, Y. Tao, Variable weighted convolutional neural network for the nitrogen content quantization of Masson pine seedling leaves with near-infrared spectroscopy, Spectrochim. Acta Mol. Biomol. Spectrosc. 209 (2019) 32–39.
[42] T. Liu, Z. Li, C. Yu, Y. Qin, NIRS feature extraction based on deep auto-encoder neural network, Infrared Phys. Technol. 87 (2017) 124–128.
[43] X. Zhang, J. Xu, J. Yang, L. Chen, H. Zhou, X. Liu, H. Li, T. Lin, Y. Ying, Understanding the learning mechanism of convolutional neural networks in spectral analysis, Anal. Chim. Acta 1119 (2020) 41–51.
[44] J. Yang, J. Xu, X. Zhang, C. Wu, T. Lin, Y. Ying, Deep learning for vibrational spectral analysis: recent progress and a practical guide, Anal. Chim. Acta 1081 (2019) 6–17.
[45] R. Houhou, T. Bocklitz, Trends in artificial intelligence, machine learning, and chemometrics applied to chemical data, Anal. Sci. Adv. 2 (2021) 128–141.
[46] B. Zoph, Q.V. Le, Neural Architecture Search with Reinforcement Learning, 2016 arXiv preprint arXiv:1611.01578.
[47] J. Mellor, J. Turner, A. Storkey, E.J. Crowley, Neural architecture search without training, PMLR (2021) 7588–7598.
[48] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT press, 2016.
[49] L.N. Smith, Cyclical Learning Rates for Training Neural Networks, IEEE, pp. 464-472.
[50] L.N. Smith, A Disciplined Approach to Neural Network Hyper-Parameters: Part 1–learning Rate, Batch Size, Momentum, and Weight Decay, 2018 arXiv preprint arXiv:1803.09820.
[51] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, J. Mach. Learn. Res. 13 (2012).
[52] X. Dong, J. Shen, W. Wang, Y. Liu, L. Shao, F. Porikli, Hyperparameter optimization for tracking with continuous deep Q-learning, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 518–527.
[53] J. Rijsdijk, L. Wu, G. Perin, S. Picek, Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis, IACR Trans. Cryptographic Hardw. Embed. Syst., 2021 (2021) 677–707.
[54] J. McCall, Genetic algorithms for modelling and optimisation, J. Comput. Appl. Math. 184 (2005) 205–222.
[55] S. Loussaief, A. Abdelkrim, Convolutional neural network hyper-parameters optimization based on genetic algorithms, Int. J. Adv. Comput. Sci. Appl. 9 (2018) 252–266.
[56] L. Tani, D. Rand, C. Veelken, M. Kadastik, Evolutionary algorithms for hyperparameter optimization in machine learning for application in high energy physics, Eur. Phys. J. 81 (2021) 1–9.
[57] J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, Adv. Neural Inf. Process. Syst. (2011) 24.
[58] B. Shahriari, K. Swersky, Z. Wang, R.P. Adams, N.d. Freitas, Taking the human out of the loop: a review of bayesian optimization, Proc. IEEE 104 (2016) 148–175.
[59] J. Snoek, H. Larochelle, R.P. Adams, Practical bayesian optimization of machine learning algorithms, Adv. Neural Inf. Process. Syst. (2012) 25.
[60] A. Zela, A. Klein, S. Falkner, F. Hutter, Towards Automated Deep Learning: Efficient Joint Neural Architecture and Hyperparameter Search, 2018 arXiv preprint arXiv:1807.06906.
[61] S. Falkner, A. Klein, F. Hutter, BOHB: Robust and Efficient Hyperparameter Optimization at Scale, PMLR, pp. 1437-1446.
[62] J. Wang, J. Xu, X. Wang, Combination of Hyperband and Bayesian Optimization for Hyperparameter Optimization in Deep Learning, 2018 arXiv preprint arXiv:1801.01596.
[63] L. Li, K.G. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar, Hyperband: Bandit-Based Configuration Evaluation for Hyperparameter Optimization.
[64] K. Jamieson, A. Talwalkar, Non-stochastic Best Arm Identification and Hyperparameter Optimization, PMLR, pp. 240-248.
[65] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: a next-generation hyperparameter optimization framework, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Association for Computing Machinery, Anchorage, AK, USA, 2019.
[66] B. James, Y. Daniel, C. David, Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures, PMLR, pp. 115-123.
[67] T.G. authors, GPyOpt: A Bayesian Optimization Framework in Python, 2016.
[68] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D.G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, TensorFlow: a system for large-scale machine learning, in: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, USENIX Association, Savannah, GA, USA, 2016.
[69] A. Savitzky, M.J.E. Golay, Smoothing and differentiation of data by simplified least squares procedures, Anal. Chem. 36 (1964) 1627–1639.
[70] R.J. Barnes, M.S. Dhanoa, S.J. Lister, Standard normal variate transformation and de-trending of near-infrared diffuse reflectance spectra, Appl. Spectrosc. 43 (1989) 772–777.
[71] A. Kohler, J.H. Solheim, V. Tafintseva, B. Zimmermann, V. Shapaval, 3.03 - model-based pre-processing in vibrational spectroscopy, in: S. Brown, R. Tauler, B. Walczak (Eds.), Comprehensive Chemometrics, second ed., Elsevier, Oxford, 2020, pp. 83–100.
[72] T. Isaksson, T. Næs, The effect of multiplicative scatter correction (MSC) and linearity improvement in NIR spectroscopy, Appl. Spectrosc. 42 (1988) 1273–1284.
[73] D.P. Kingma, J. Ba, Adam, A Method for Stochastic Optimization, 2014 arXiv preprint arXiv:1412.6980.
[74] K. He, X. Zhang, S. Ren, J. Sun, Delving Deep into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification, pp. 1026-1034.
[75] N. Anderson, K. Walsh, P. Subedi, Mango DMC and spectra Anderson et al, Mendley, Mendley data, 2020.
[76] N.T. Anderson, K.B. Walsh, P.P. Subedi, C.H. Hayes, Achieving robustness across season, location and cultivar for a NIRS model for intact mango fruit dry matter content, Postharvest Biol. Technol. 168 (2020) 111202.
[77] N.T. Anderson, K.B. Walsh, J.R. Flynn, J.P. Walsh, Achieving robustness across season, location and cultivar for a NIRS model for intact mango fruit dry matter content. II. Local PLS and nonlinear models, Postharvest Biol. Technol. 171 (2021) 111358.
[78] L. Zhou, C. Zhang, M.F. Taha, X. Wei, Y. He, Z. Qiu, Y. Liu, Wheat kernel variety identification based on a large near-infrared spectral dataset and a novel deep learning-based feature selection method, Front. Plant Sci. 11 (2020) 1682.

[79] R.R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, D. Batra, Grad-CAM, Visual explanations from deep networks via gradient-based localization, Int. J. Comput. Vis. 128 (2020) 336–359.
[80] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, F. Hutter, Efficient and robust automated machine learning, Adv. Neural Inf. Process. Syst. 28 (2015).
[81] Ruchte, M., Zela, A., Siems, J., Grabocka, J. and Hutter, F., NASLib: A Modular and Flexible Neural Architecture Search Library, Github, https://github.com/automl/NASLib.
[82] D.R. Jones, A taxonomy of global optimization methods based on response surfaces, J. Global Optim. 21 (2001) 345–383.
[83] L. Alzubaidi, J. Zhang, A.J. Humaidi, et al., Review of deep learning: concepts, CNN architectures, challenges, applications, future directions, J. Big Data 8 (2021) 53.
[84] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, D.J. Inman, 1D convolutional neural networks and applications: a survey, Mech. Syst. Signal Process. 151 (2021) 107398.
[85] Project jupyter, url: https://jupyter.org/.
[86] M. Feurer, F. Hutter, Hyperparameter optimization, in: F. Hutter, L. Kotthoff, J. Vanschoren (Eds.), Automated Machine Learning, The Springer Series on Challenges in Machine Learning. Springer, Cham, 2019.
[87] T. Elsken, J.H. Metzen, F. Hutter, Neural architecture search, in: F. Hutter, L. Kotthoff, J. Vanschoren (Eds.), Automated Machine Learning, The Springer Series on Challenges in Machine Learning. Springer, Cham, 2019.
[88] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, A. Torralba, Learning deep features for discriminative localization, in: IEEE Conf. Comput. Vis. Pattern Recognit, IEEE, 2016, p. 2921e2929.
[89] N. Srivastava, G.E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.
[90] E. Cyr, M. Gulian, R. Patel, M. Perego, N. Trask, Robust Training and Initialization of Deep Neural Networks: an Adaptive Basis Viewpoint, 2020, 04862. ArXiv, abs/1912.