Electric Power Systems Research 211 (2022) 108203

Contents lists available at ScienceDirect

Electric Power Systems Research

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

Convolutional neural nets with hyperparameter optimization and feature importance for power system static security assessment

Miguel Ramirez-Gonzalez [a,*], Felix Rafael Segundo Sevilla [a], Petr Korba [a], Rafael Castellanos-Bustamante [b]

[a] Institute of Energy Systems and Fluid Engineering, School of Engineering, Zurich University of Applied Sciences, 8401 Winterthur, Switzerland
[b] Transmission and Distribution Department, National Institute for Electricity and Clean Energy, 62490 Cuernavaca, Morelos, Mexico

ARTICLE INFO

Keywords:
Power system security assessment
Convolutional neural networks
Hyperparameter optimization
Feature importance
Machine learning

ABSTRACT

Static security assessment (SSA) is fundamental in electrical network analysis. However, the growing complexity and variability of grid's operating conditions can make it tedious, slow, computationally intensive, and limited or impractical for on-line applications when traditional approaches are considered. Since this may hinder the emerging analytical duties of system operators, data-driven alternatives are required for faster and sophisticated decision-making. Although different machine learning algorithms (MLAs) could be applied, Convolutional Neural Networks (CNNs) are one of the most powerful models used in many advanced technological developments due to their remarkable capability to identify meaningful patterns in challenging and complex data sets. According to this, a CNN based approach for fast SSA of power systems with N-1 contingency is presented in this paper. To contribute to the automation of model building and tuning, a settings-free strategy to optimize a set of hyperparameters is adopted. Besides, permutation feature importance is considered to identify only a subset of key features and reduce the initial input space. To illustrate the application of the proposed approach, the simulation model of a practical grid in Mexico is used. The superior performance of the CNN alternative is demonstrated by comparing it with two popular MLAs.

1. Introduction

The digitalization of power systems is intended to improve the efficiency, productivity, and autonomy of their processes. However, the incorporation and massive deployment of new devices and technologies for this purpose is resulting in more complex topologies and structures, and in even more diverse system operating conditions and scenarios. All this continuous development of the grid is bringing about new challenges in terms of ensuring network security, resiliency, and stability. Since traditional power system analysis approaches are mainly based on physical modeling and numerical computations, they are time consuming and computationally intensive, particularly for on-line applications. Therefore, it is recognized that they cannot longer effectively address emerging analytical needs and requirements of modern electrical networks [1].

In general, the security of a power system is related to its ability to provide nominal voltage and maintain system frequency within a certain tolerant band to cope with imminent disturbances/contingencies without power supply interruptions [2]. Among different operating issues, power system security is of vital importance due to the significant social, economic, and political impact it may have when it is endangered [3]. In this context, the static security analysis is a fundamental part of the security assessment in electrical networks that focuses on the steady-state response of the grid under a predefined set of credible contingencies. Essentially, its objective is to perform an evaluation of the condition of the system (secure or insecure) in terms of the transgression of any load and voltage boundaries following the trip or loss of assets on the grid [4]. The traditional technique to investigate this issue has been by solving load flow equations repeatedly, while examining the security constraints for each simulation result. Given that a considerable amount of simulations are required to analyze all possible contingencies over a broad range of potential operating conditions with the traditional approach, alternative solutions based on machine learning have recently received much interest due to their capability to extract valuable information from large and complex datasets. Particularly, the recent improvements and performance achieved by deep learning methods in a

* Corresponding author.
E-mail address: ramg@zhaw.ch (M. Ramirez-Gonzalez).

https://doi.org/10.1016/j.epsr.2022.108203
Received 31 August 2021; Received in revised form 21 April 2022; Accepted 15 June 2022
Available online 21 June 2022

wide range of applications have positioned these machine learning algorithms as some of the most popularly used [5,6].

Despite the successful application of deep learning techniques in different power system areas, their use within power system static security assessment is relatively new [1,3,6]. For example, convolutional neural networks (CNNs) were proposed in [7] considering N-1 contingency criteria. In that work, data such as the topology of the system and power injections at different buses are used as input to the classification model for security evaluation purposes. In [8], deep learning models based also on CNNs are trained on a large database to assess both N-1 security and small-signal stability, where input information such as active power, reactive power, and voltage at each node is arranged into three squared matrices (similar to the typical arrays or channels used in image classification) to uniquely represent the state of the power system. Static security assessment is also addressed in [9] through deep learning algorithms and architectures, where the system's power flows are approximated according to productions, consumptions, and grid topology information to assist decisions of human operators.

Irrespective of the favorable results reported in the aforementioned research works, improvements in the methodologies are still possible and are highlighted next. For example, the explicit use of grid topology information in the form of bus admittance matrix to detect system topology changes may not be feasible for practical applications in systems with a considerable number of buses. On the other hand, representing each sample data as independent power injection and voltage matrices of the same order as the number of buses in the system is impractical because of the final arrangement dimensions. Moreover, the prerequisite of having measurements from every node in the system is unrealistic since, for example, the installation of advanced metering devices at each bus for this purpose, such as phasor measurement units, is economically unfeasible and technically challenging. For this reason, strategies to optimally allocate metering infrastructure have to be investigated [10]. In addition, the lack of an automated and systematic approach to support the development of more efficient deep learning models, with more suitable hyperparameters, has to be considered since this issue may affect the overall process of model development and the algorithm's capacity to learn [11,12].

Based on the issues above, the main challenge addressed in this paper has to do with the development of an advanced, effective, and efficient methodology for fast and reliable decision making concerning the static security assessment of power systems under N-1 contingency. Therefore, CNNs are trained and used here to classify the security status of a practical sample network according to categories such as secure, alarm and insecure. The input data of the proposed CNN, which are the voltage magnitudes, as well as active and reactive power injections at systems nodes, are arranged into 3xn matrices for each sample instance, where n refers to the total number of buses considered for this data-driven application. Besides, target outcomes for supervised learning, which represent the class labels associated to each example, are determined according to a predefined security index based on transmission line loading and bus voltage violations. In order to contribute to the automation of the machine learning process, a model hyperparameter optimization alternative is carried out in this work based on a settings-free and derivative-free strategy called Jaya Optimization Algorithm (JOA). Furthermore, a feature importance approach is also applied in this sense to identify a subset of features that mostly impact model predictions and reduce the features of the initial dataset. The effectiveness of the proposed approach in making predictions and generalizing to new data is compared to Support Vector Machine and K-Nearest Neighbor based alternatives. The main contributions of the paper can be summarized as follows:

- The development of an alternative data-driven solution for power system static security assessment using CNNs.
- The formulation and application of a settings-free optimization strategy for the automated and systematic selection of different CNN model hyperparameters.
- The exploration and use of permutation feature importance to reduce the number of input features and guide the location of measuring technologies for the application at hand.

The paper is organized as follows. A brief description of CNNs and their main building components is provided in Section 2. The sample power system used for the studies is presented in Section 3. The approach for the static security assessment of the system with CNNs, including dataset generation, model architecture, optimization algorithm for selected hyperparameters, and the technique to identify the relevant features that mostly impact the predictions of the CNN model, is described in Section 4. On the other hand, obtained simulation results with the proposed alternative are presented and discussed in Section 5. Finally, concluding remarks are provided in Section 6.

2. Convolutional neural networks (CNNs)

CNNs are one of the most powerful deep learning models mostly used in the field of computer vision and imaging. However, due to their high level of accuracy in learning and generalizing complex non-linear functions and patterns, this class of networks are now receiving increasing attention in a wide range of academic and industrial tasks [13,14]. Briefly, the main building blocks of CNNs can be described as follows [13–15]:

- Convolutional layers, which are composed of feature detectors or filters, and feature maps. In general, the task of a convolutional layer is to filter a given input and transform it into a set of feature maps. Feature detectors are drawn across the layer's input while computing the dot product at each particular position. This process is repeated for all input cells according to a predefined stride. The output is a group of stacked activation maps (one feature map for each considered filter).
- Pooling layers, which basically down-sample the previous layer's activation maps and reduce the spatial size of the representation. They are intended to consolidate all the relevant features learned and derived through a previous convolutional layer. Pooling layers can help in controlling overfitting of the training dataset for a better generalization of the features represented by the network. They can use very simple operations such as the maximum or average of the elements involved in a given receptive field to produce a pooled feature map.
- Fully connected layers, which are represented by typical feed-forward neural network layers. They are constructed at the end of the CNN architecture, after all features have been identified and extracted by convolutional layers and consolidated by pooling layers. Fully connected layers are included to complete the final classification tasks.

Based on this description, the basic architecture of a CNN is illustrated in Fig. 1, where the details about convolution and pooling operations are shown only for exemplification purposes.

CNNs are especially useful for identifying patterns in data sets, and automatically and directly recognizing various characteristics, without the need to extract them manually. Training filters applied to each input object allow to extract both simple and very complex characteristics that unambiguously define the object under consideration [13,16].

3. Test power system

An isolated section of the Mexican Electric Grid located on the state of Baja California Sur (BCS), in northwestern Mexico, is considered as the sample power system for the studies presented here. The BCS system

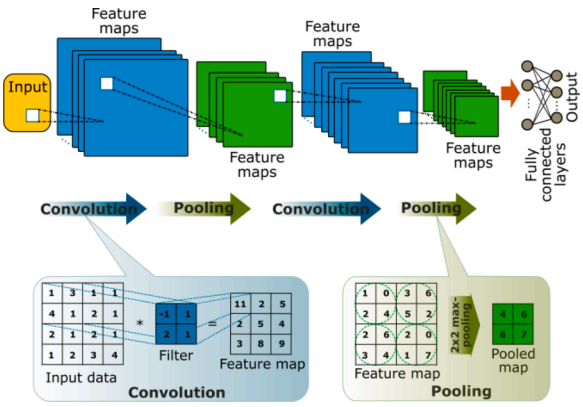

Fig. 1. Basic architecture of CNNs.

represents a relatively small network whose generation is based mainly on a series of small fossil fuel-fired generating units distributed across the state. Electric power to main load centers is provided through transmission lines and power sub-stations operating at 230 kV and 115 kV. For illustrative purposes, the BCS grid can be divided into three transmission zones (Z1, Z2 and Z3), and represented here by the schematic diagram in Fig. 2, where the relative location of power plants is denoted by white circles and different load buses can be identified by black circles. While Z1 and Z2 are interconnected by a 115 kV double circuit transmission line, the interconnection between Z2 and Z3 comprises a 115 kV and a 230 kV double circuit tie line.

A simulation model that portrays a base case scenario related to the maximum load demand during 2012, and consists of 81 buses, 20 generators, 28 loads, 50 lines, and 52 transformers, is used here to represent the network under consideration.

4. Static security assessment with CNNs

4.1. Dataset generation

In order to provide representative samples of both normal and unusual system operating conditions and situations, a suitable collection of data is fundamental for building and training the machine learning algorithm under consideration for the prediction of the system security status. In this case, a relatively large and diverse training dataset, with a variety of operating points, will potentially benefit model generalization to new, unseen data.

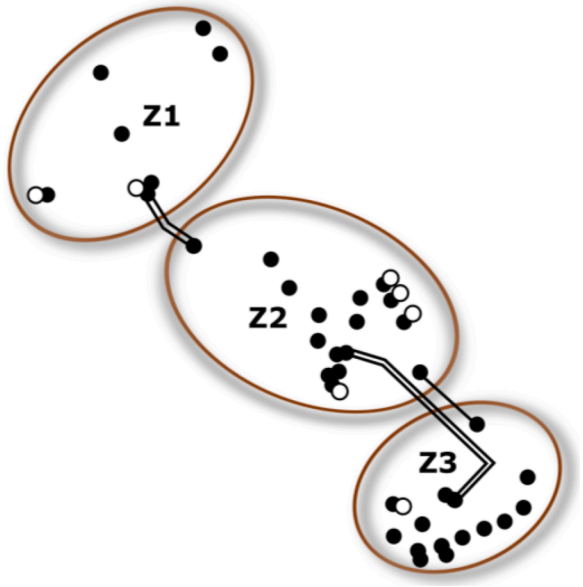

Fig. 2. Test power system.

For illustrative purposes of the system under investigation, the required training dataset was generated through simulations considering a relatively large range of system operating conditions and N-1 security criterion following transmission line outages as system contingencies. Therefore, the active and reactive power of the loads in the sample grid were varied between 50% and 125% of their value, as compared to the base case scenario. Furthermore, the power supplied by synchronous generators was correspondingly altered in order to balance the load demand changes. In this case, active power and bus voltage at each generator were varied randomly within a predefined range (according to a normal distribution). Based on this, voltage magnitudes, and active and reactive power injections at system nodes were collected as examples of model inputs.

In order to quantify the system security condition in a suitable way for each corresponding instance, a static security index (*SSI*) was computed according to line loading and bus voltage violations [7,17], as given in the following expression:

$$SSI = \left[\sum_j \left(\frac{\Delta v_j^u}{D_v^u} \right)^2 + \sum_j \left(\frac{\Delta v_j^l}{D_v^l} \right)^2 + \sum_k \left(\frac{\Delta i_k}{D_i} \right)^2 \right]^{\frac{1}{2}} \quad (1)$$

where Δv_j^u and Δv_j^l refer to bus voltage deviations beyond predefined upper and lower alarm limits, respectively, Δi_k represents line loading deviations based on the power flow through the line and a given alarm limit, and D_v^u, D_v^l, and D_i denote normalization factors calculated from the distance between alarm and security limits for bus voltage and line loading levels. Considering *SSI* values, input patterns were labelled according to classes such as secure, alarm and insecure, based on the following evaluation:

$$System\ status = \begin{cases} Secure & \text{if} \quad SSI = 0 \\ Alarm & \text{if} \quad 0 < SSI \leq 1 \\ Insecure & \text{if} \quad SSI > 1 \end{cases} \quad (2)$$

As result, a dataset with a total of 6539 examples of associated input-output pairs was obtained for model training and testing. It is worth mentioning here that only voltage magnitudes, and active and reactive power injections at buses with more than two connected branches were considered to reduce the input space. Finally, arranging the resulting data into 3×1 arrays for each bus, an input dataset of dimension $6539 \times 3 \times 50$ was created. In this study, the full dataset was divided such that 30% of the total input-output pairs were used for testing and the remaining 70% were used for model training. Furthermore, 30% of the training samples were assigned to the validation set. Naturally, prior to be used, the input samples of the whole dataset are normalized in the range 0–1. This was accomplished here by subtracting the corresponding minimum value for collected voltage magnitudes and involved power injections at system nodes, and then dividing by the range between respective maximum and minimum values. On the other side, targets were one-hot encoded [13], and the categorical feature was transformed into an all-zero vector of multiple true/false features, with a 1 in the appropriate position to indicate the presence of a particular categorical value.

4.2. CNN model architecture

The selection of an optimal architecture for a particular application of CNNs is in general a demanding task due to the degrees of freedom involved. In this sense, hyperparameters such as number of layers, kernel sizes, dropout, learning rates, etc., play a key role in the success of CNNs for a given problem [18]. Nevertheless, depending on the application and the dataset, attempting to tune all hyperparameters of a CNN would be time-consuming and computationally intensive. Therefore, a

priory knowledge of the process is essential, so that only a subset of these can be selected for refined tuning [19].

In this work, a CNN model with two convolutional layers and one max-pooling layer as feature extractors, followed by one fully connected layer to interpret these features and one output layer for three-class predictions is considered, as illustrated in Fig. 3. In this architecture, kernels of size 2 × 3 and 1 × 3 were employed for each convolutional layer, respectively. In addition, the Rectified Linear Unit (ReLU) function was used on each "Activation" block, and batch normalization was included to improve the learning convergence.

The pooling layer was configured here with a maximum value function over a 1 × 3 window size. The flatten block at the end of this process is used to reshape the output of the dropout layer into one-dimensional vector before going through the densely connected classifier. While ReLu activation is used for the first dense layer, the final output layer was set up with the Softmax function. For performance optimization purposes, the number of filters in the convolutional layers (flt_1 and flt_2, respectively), the number of units (*nou*) in the first layer of the fully connected network, and the dropout rate (*dpr*) were selected to be determined in this study through a settings-free optimization method, as presented in the following Sections.

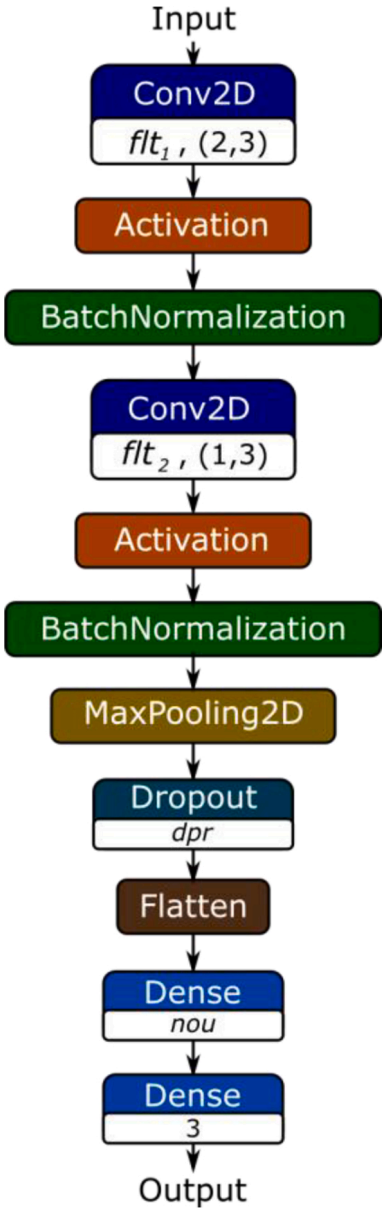

Fig. 3. CNN architecture.

4.3. Jaya optimization algorithm (JOA)

Many popular population-based optimization algorithms such as Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) depend on several algorithm-specific control parameters that may impact their overall performance (for example, mutation and crossover rates in GA, and inertia weights and cognitive and social acceleration factors in PSO). Generally, these parameters are to be tuned for the best settings in the desired application, and the effort to adjust them increases with the quantity of parameters and their intrinsic interactions. On the other hand, JOA is a relatively new and settings-free optimization method, simpler to use and implement as compared to related algorithms controlled by specific parameters [20].

Based on a predefined fitness function, the basic operating principle of JOA is to minimize the cost function value to achieve an optimal result by avoiding the worst solution and quickly moving toward the best solution. Considering a candidate solution x_i at the *t-th* iteration, the iterative process for this purpose can be expressed as follows:

$$x_i^t = x_i^{t-1} + rand_1\left(x_{best} - x_i^{t-1}\right) - rand_2\left(x_{worst} - x_i^{t-1}\right) \quad (3)$$

where x_{best} and x_{worst} represent the best and worst performing solutions available, respectively. Similarly, $rand_1$ and $rand_2$ refer to random numbers in the range 0 – 1 for each search variable. While the second term in (3) represents the tendency of the solution to always try to get closer to the optimal solution in each iteration, the last term in the expression has to do with moving away from the worst solution [20]. Based on (3), new candidate solutions are generated and evaluated, and old candidate solutions are replaced only if the new solutions perform better according to the achieved fitness function value. This iterative process is illustrated through the flowchart in Fig. 4. A detailed

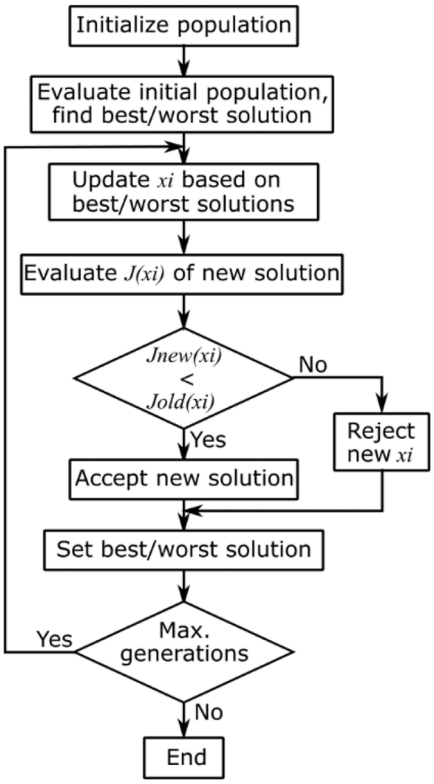

Fig. 4. Flowchart of JOA.

description of JOA and its engineering applications can be found in [20, 21].

4.4. Permutation feature importance (PFI)

Since the predictive power of a trained model may be linked mainly to a reduced number of features from the entire and original input space, the identification of these relevant features may avoid the use of redundant information, increase the interpretability of the model for the problem at hand by reducing complexity, decrease computational burden and training time, and even boost the performance of the model [22]. Furthermore, from an online security assessment application point of view with PMUs, the number of measuring points to obtain the required information to be fed into the machine learning model would also be reduced, leading to a reduction in the number of these units.

In general, feature importance refers to the identification of a subset of features that mostly impact the predictions of an already trained model. In particular, PFI represents an inspection technique based on scores associated with model prediction performances when a single feature in the data is randomly shuffled. Basically, if the prediction error rises after the permutation of values of a given feature, then we can assume that this feature may be more important than some other one in terms of the obtained related errors. If this error is practically unaffected, then the model is essentially ignoring the given feature [22].

Fig. 5 illustrates in a simple way the concept behind PFI, where a particular feature Fb of a certain dataset is randomly shuffled (while the values of all other features and targets are left unchanged), and then the model is used to make predictions. This process is applied to every feature in the data (one at a time).

In order to calculate a mean feature importance (MFI) score for each particular input feature, the whole process of permutation and computation of model's prediction error needs to be repeated several occasions (for example ten times) [23]. In this way, the MFI score for feature f can be determined according to the following expression:

$$MFI_f = \frac{100\left(AM - \overline{AM}_f^{per}\right)}{\overline{AM}_f^{per}} \tag{4}$$

where AM is the accuracy of the model without feature shuffling, and $\overline{AM}_f^{per}$ denotes the achieved average accuracy after permutation of feature f. It is worth mentioning here that, in this work, the dataset features refer to the considered buses of the system, where each collected sample for V, P and Q variables is arranged in a corresponding array for each bus. Therefore, permuting the information of a particular bus in the dataset means to randomly interchange the position of the arrays within the whole set of sample arrays for that bus.

5. Simulation results

5.1. Optimization of selected hyperparameters

The following constrained optimization problem was formulated for the determination of the CNN model's hyperparameters indicated in Section 4.2, particularly the filters for convolutional layers (flt_1 and flt_2), the number of units in the first layer of the fully connected network (nou), and the dropout rate (dpr):

$$\begin{aligned} &Minimize\ J\ subject\ to: \\ &\Lambda_i^{min} \le \Lambda_i \le \Lambda_i^{max},\ i = 1, .., 4 \end{aligned} \tag{5}$$

where Λ_i represents each of the selected hyperparameters, and J refers to the fitness function to be minimized, which in this case is defined as:

$$J = c_1\left(\frac{1}{Tra\_acc}\right) + c_2\left(\frac{1}{Val\_acc}\right) \tag{6}$$

with Tra_acc and Val_acc denoting the model training and validation accuracies, respectively, and c_1 and c_2 being weighting factors. For the minimization of (6) using JOA in Section 4.3, all candidate solutions are updated according to (3), and then integer variables such as the number of convolutional filters, and the number of neurons in the first dense layer are rounded to the nearest integer value before being evaluated [24]. The iterative application of the dropout rate proceeds as directly determined from (3). Now, since the model is re-trained for every combination of hyperparameters being tried, it is clear that the execution time for the evaluation of a given set of candidate solutions at each generation of hyperparameter optimization will proportionally grow as the population size is increased, which at some point will become a very computationally expensive process. Therefore, a relatively small population size of 5 members was used here considering that it can still provide competitive results with acceptable convergence rates [25]. In addition, it is also worth mentioning that the categorical cross-entropy loss function [16] and the Adam optimizer [26] were employed in the studies to guide the learning process of the CNN models.

By implementing the JOA based process in Python, setting the population size to five elements, and fixing the number of generations to fifty, candidate solutions were iteratively evaluated to find the foremost model performance. By using a computer with a processor Intel Core i7–8665 U, CPU @ 1.90 GHz, 16.0 RAM, the hyperparameter optimization task took an average of 8 h, and the time taken to go over 250 epochs of network training for each candidate solution was around 115 s. Naturally, the use of some more powerful computational resources can favorably impact this time. The convergence of the algorithm for the minimization of the cost function is illustrated in Fig. 6, where results with a Particle Swarm Optimization algorithm [27] have also been included for comparison. The corresponding best final solutions after several independent runs are presented in Table 1.

Based on the performance shown in Fig. 6, and the fact that JOA has no dependance at all on algorithm-specific parameters, a summary of the CNN model with the solutions provided by JOA is given in Table 2, where the output shape of each layer, size of weights, and number of

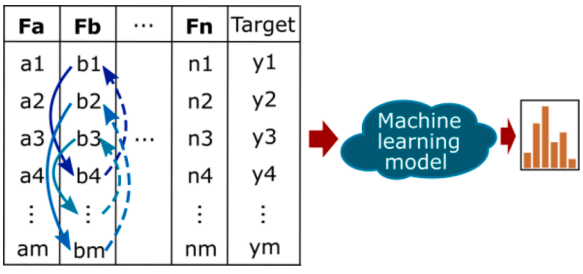

Fig. 5. Random permutation of the values of feature Fb.

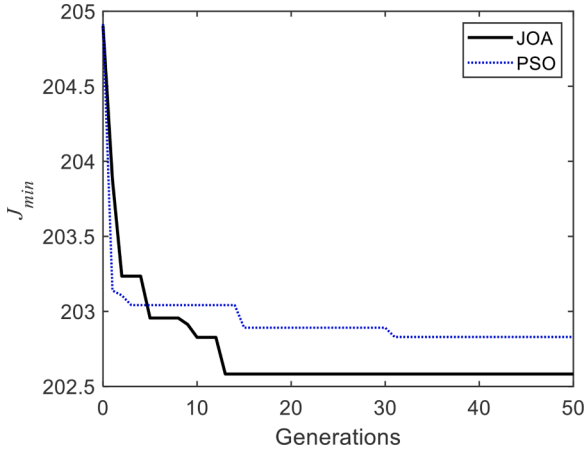

Fig. 6. Algorithm convergence during cost function minimization.

Table 1
Best final solutions.

Algorithm	Parameter	Value	Jmin
JOA	flt_1	80	202.58
	flt_2	40	
	nou	81	
	dpr	0.209	
PSO	flt_1	77	202.83
	flt_2	38	
	nou	100	
	dpr	0.440	

Table 2
Model summary.

Layer	Outputshape	Size of weights	Bias	Parameters
Conv2D_1 & Activation	(*, 2, 48, 80)	(2, 3, 1, 80)	80	560
BatchNorm	(*, 2, 48, 80)			160
Conv2D_2 & Activation	(*, 2, 46, 40)	(1, 3, 80, 40)	40	9640
BatchNorm	(*, 2, 48, 40)			80
MaxPool2D	(*, 2, 15, 40)			
Dropout	(*, 2, 15, 40)			
Flatten	(*, 1200)			
Dense_1 & Activation	(*, 81)	(1200, 81)	81	97,281
Dense_2 & Activation	(*, 3)	(81, 3)	3	246
Total trainable parameters				**107,967**

trainable parameters are included (the symbol * is used here as a general denotation for the batch size).

5.2. Model accuracy and loss curves for selected solution

Visualization of CNN model performance over several epochs represents one common way to review and determine how well it is learning and generalizing during training on both the training and validation datasets [24]. In this manner, the model behavior to learning issues such as underfitting or overfitting can be diagnosed. According to this, model accuracy and loss achieved during training using the best set of selected hyperparameters after JOA based optimization are correspondingly given in Fig. 7 and Fig. 8.

Based on the selected dataset, and from the results displayed in Figs. 7 and 8, it can be observed that the model has been sufficiently trained over the indicated number of epochs since both relatively high model accuracy and low loss are achieved at the end of training. Since the model losses converges to a minimum value, with reduced gap

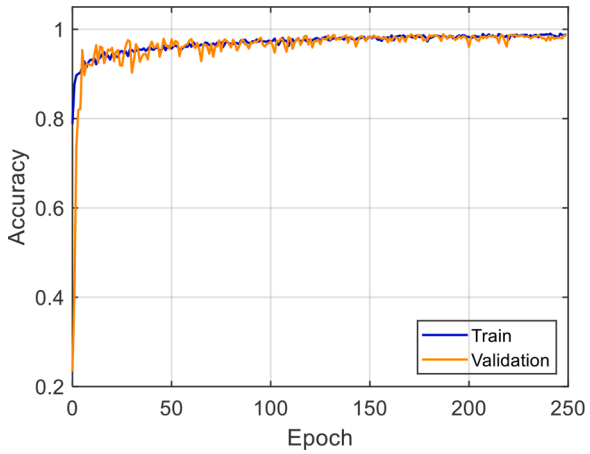

Fig. 7. Model accuracy.

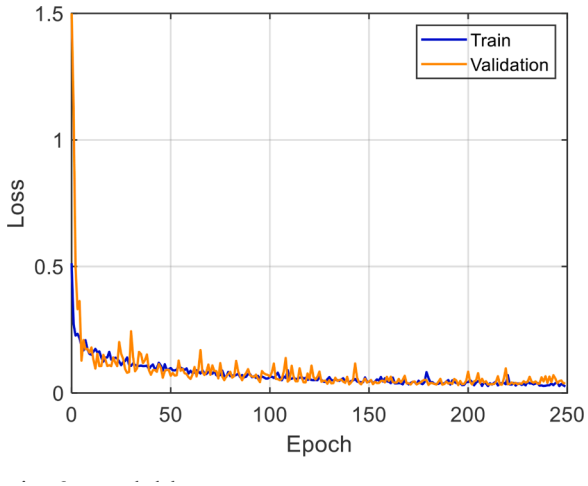

Fig. 8. Model losses.

between the training and validation loss as depicted in Fig. 8 on the far right, a reasonably good fit can be identified. In this case, continued training of the fitted model will potentially make it start memorizing the training data and lead to an overfit, which will affect the model capability to generalize and predict accurately.

5.3. Model performance metrics

To measure the performance of the model in making predictions and generalizing to new, previously unseen data, common evaluation metrics for classification tasks such as Accuracy, Precision, Recall, and F1-Score [24] were used, which are described next.

- Accuracy refers to the number of correct predictions divided by the total number of predictions for a given dataset. It is the most typical evaluation metric for classification problems; however, it may lead to misleading results for problems with unbalanced classes. Classification accuracy can be determined according to the following expression:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \tag{7}$$

- Precision can be defined as the number of true positives results divided by the total number of positive instances that were predicted by the model. Precision is not concerned with false negatives and is computed as indicated below:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{8}$$

- Recall is an evaluation metric that quantifies the number of correct positive cases predicted against all positive instances in the dataset. Therefore, it gives an indication of the proportion of actual positives correctly identified. It is calculated as follows:

$$Recall = \frac{True\ Positives}{True\ Positives\ + False\ Negatives} \tag{9}$$

- F1-Score combines precision and recall into a single measure to get an indication of the model performance when both false positive and false negative errors are of great concern. The computation of this metric is:

$$F1 - Score = 2\frac{Precision \cdot Recal}{Precision + Recal} \quad (10)$$

After performing the optimization of selected hyperparameters, the evaluation metrics described above were computed for both the training and test datasets, and the results are given in Table 3. For comparison purposes with the CNN model, corresponding results with two popular and widely used machine learning algorithms such as Support vector Machine (SVM) and K-Nearest Neighbor (KNN) are also included. For the application presented in this work, since the impact of misclassifying a given sample belonging to the Insecure class will be more critical than having to examine an irrelevant False Positive instance for this class, the Recall measure becomes relevant here. In general, as it can be seen from Table 3, the CNN model shows the best classification performance in this study to previously unseen data, according to the considered metrics.

The response of the CNN model on the test dataset is further illustrated through the confusion matrix in Fig. 9, which provides details about the predicted results (columns) as compared to the true values (rows). In this representation, the diagonal elements indicate the instances correctly predicted by the classifier.

5.4. Model based on permutation feature importance scores

In order to find out the impact of input features on the predictions of the model that has already being trained, relative importance scores were computed according to the permutation technique described in Section 4.4. Thereby, the mean importance score for each column of the involved dataset, after independently shuffling each column ten times, is illustrated in Fig. 10.

As it can be seen in Fig. 10, the information from some buses turns out to be irrelevant or less important and does not contribute at all to the predictions of the model in this case. Consequently, if buses with an importance score for instance larger than 0.01 are chosen here, then only 26 buses out of 50 will be kept: 10, 11, 12, 13, 14, 15, 19, 20, 21, 23, 25, 26, 29, 30, 31, 33, 34, 36, 38, 39, 41, 42, 43, 44, 45, and 46. Now, by training a new CNN model with information from these buses only, and the same architecture as in Fig. 3, the results for the performance metrics with the new model are given in Table 4.

It can be seen from Tables 3 and 4 that the performance of the two CNN models is very comparable, and that the removed information was in fact relatively irrelevant based on the computed performance importance scores. After dataset reduction through PFI, three of the remaining buses belong to Z1, fourteen to Z2, and nine to Z3, as highlighted in blue color in Fig. 11.

From the results in Fig. 11, it is worth noting the following related to the features determined as relevant (through PFI) for model prediction: (a) all buses where power plants are connected to the main transmission network have been selected, and (b) all buses except one associated to the Z1-Z2 and Z2-Z3 tie lines have also been included. These results can be exploited in practice to better allocate a minimum set of involved measuring devices.

Finally, it is worth emphasizing that after considering the computational time to classify the test data set, it was found that this CNN model can take only around 0.0906 s in the computer used, as compared to 16.017 s taken by the traditional power flow simulations.

Table 3
Performance metrics.

Model	Dataset	Metric Accuracy	Precision	Recall	F1-score
CNN	Training	99.45%	99.37%	99.55%	99.46%
	Test	**98.88%**	**98.73%**	**98.95%**	**98.83%**
SVM	Training	98.01%	98.14%	97.83%	97.93%
	Test	**97.45%**	**97.50%**	**97.35%**	**97.42%**
KNN	Training	97.97%	97.87%	97.75%	97.81%
	Test	**97.04%**	**96.82%**	**96.78%**	**96.80%**

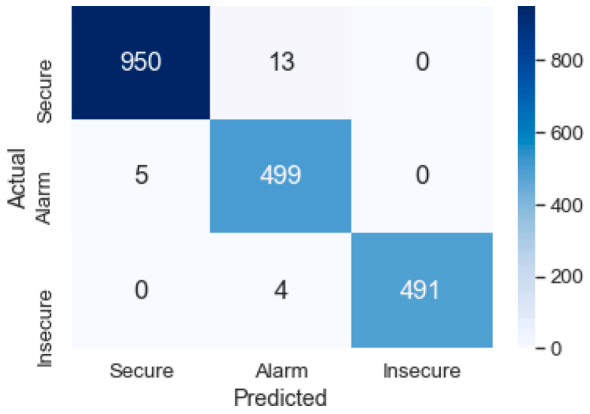

Fig. 9. Confusion matrix with CNN model applied to the test dataset.

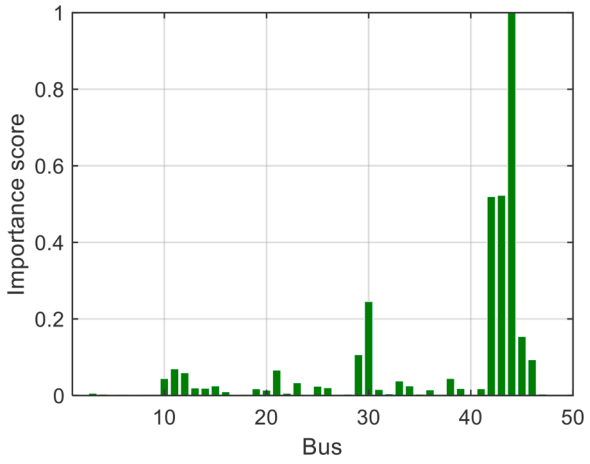

Fig. 10. Permutation feature importance scores (averaged and normalized).

Table 4
Performance metrics with reduced input space.

Model	Dataset	Metric Accuracy	Precision	Recall	F1-score
CNN	Training	99.47%	99.47%	99.45%	99.47%
	Test	**98.67%**	**98.71%**	**98.53%**	**98.62%**

6. Conclusions

An approach for the static security assessment of power systems using CNNs has been presented in this work. In this regard, the simulation model of a practical power grid located on the state of Baja California Sur, in northwestern Mexico, was considered for illustrative purposes. According to the proposed CNN structure, the formulation and application of a settings-free optimization strategy based on JOA allowed for the automated and systematic selection of different hyperparameters, which contributes to maximizing model performance and reducing the human effort dedicated to build and tune CNN models. Afterwards, the use of permutation feature importance in the trained network provided insights about its most relevant input features for target prediction, which in practice can be used to choose the best location of a set of measuring devices considered for the application at hand. Then, a new CNN model was trained using only these identified features, and comparable results as with the original dataset were

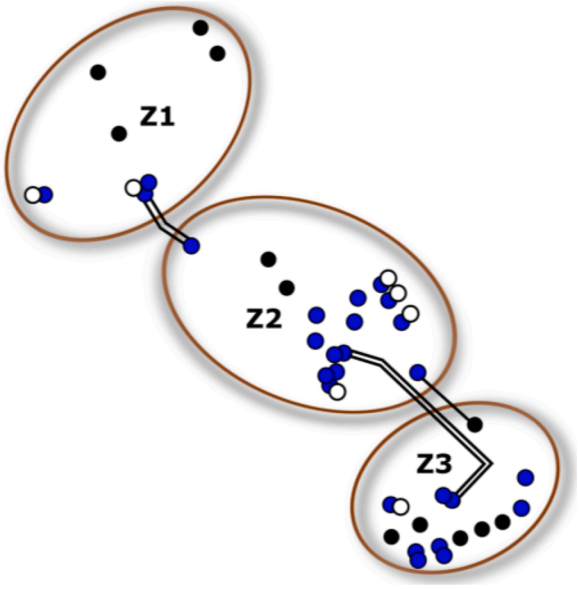

Fig. 11. Selected buses (in blue color) after PFI.

achieved. The reduction in the complexity and dimension of input data can in general contribute to simplifying the problem under consideration, decreasing the computational burden for model training, and accelerating the implementation of the solution.

By comparison against conventional machine learning alternatives such as Support Vector Machine and K-Nearest Neighbor, the effectiveness and superior performance of the proposed approach in making predictions and generalizing to new data was demonstrated. Besides, as compared to traditional power flow simulations, the time required to classify for example the test data set was radically reduced with the obtained CNN model, which shows its potential for online application. Finally, it is worth mentioning that although the hyperparameter optimization process of the model took about 8 h with the used hardware, the availability of some more powerful computational resources or high-performance and distributed computing schemes can significantly reduce this time.

CRediT authorship contribution statement

Miguel Ramirez-Gonzalez: Conceptualization, Methodology, Software, Investigation, Writing – original draft, Writing – review & editing. **Felix Rafael Segundo Sevilla:** Validation, Writing – review & editing, Visualization. **Petr Korba:** Resources, Writing – review & editing, Supervision. **Rafael Castellanos-Bustamante:** Resources, Writing – review & editing.

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 acknowledge the Swiss National Science Foundation (SNSF) under the project number PZENP2_173628 of the program Ambizione Energy Grant (AEG).

References

[1] M.S. Ibrahim, W. Dong, Q. Yang, Machine learning driven smart electric power systems: current trends and new perspectives, Appl. Energy 272 (2020) 1–19.
[2] J. Machowski, Z. Lubosny, J.W. Bialek, J.R. Bumby, Power System Dynamics: Stability and Control, John Wiley & Sons, NJ, 2020.
[3] O.A. Alimi, K. Ouahada, A.M. Abu-Mahfouz, A review of machine learning approaches to power system security and stability, IEEE Access 8 (2020) 113512–113531.
[4] M. Gholami, M.J. Sanjari, M. Safari, M. Akbari, M.R. Kamali, Static security assessment of power systems: a review, Int. Trans. Electr. Energy Syst. 30 (9) (2020) 1–23.
[5] K. Senthilnathan, B. Shanmugam, D. Goyal, I. Annapoorani, R. Samikannu, Deep Learning Applications and Intelligent Decision Making in Engineering, IGI Global, PA, 2021.
[6] A.K. Ozcanli, F. Yaprakdal, M. Baysal, Deep learning methods and applications for electrical power systems: a comprehensive review, Int. J. Energy Res. 44 (9) (2020) 7136–7157.
[7] Y. Du, F. Li, C. Huang, Applying deep convolutional neural network for fast security assessment with N-1 contingency, in: Proc. IEEE Power & Energy Society General Meeting, Aug. 2019, pp. 1–5.
[8] J.M. Hidalgo, F. Hancharou, F. Tams, S. Chatzivasileiadis, Deep learning for power system security assessment, in: Proc. IEEE Milan PowerTech, June 2019, pp. 1–6.
[9] B. Donnot, Deep Learning Methods for Predicting Flows in Power Grids: Novel Architectures and Algorithms (Doctoral Thesis), University of Paris-Saclay, Paris, France, 2019.
[10] Y. Xu, Y. Zhang, Z. Yang, R. Zhang, Intelligent Systems for Stability Assessment and Control of Smart Power Grids, CRC Press, FL, 2021.
[11] Agrawal T, Hyperparameter Optimization in Machine Learning, Apress, NY, 2021.
[12] F. Hutter, L. Kotthoff, J. Vanschoren, Automated Machine Learning: Methods, Systems, Challenges, Springer, Cham, 2019.
[13] D. Sarkar, R. Bali, T. Sharma, Practical Machine Learning with Python, Apress, NY, 2018.
[14] O.G. Yalcin, Applied Neural Networks with TensorFlow, 2, Apress, NY, 2021.
[15] N.K. Manaswi, Deep Learning with Applications Using Python, Apress, NY, 2018.
[16] M. Sewak, M.R. Karim, P. Pujari, Practical Convolutional Neural Networks, Packt Publishing, Birmingham, 2018.
[17] I. Bhatt, A. Dhandhia, V. Pandya, Static security assessment of power system using radial basis function neural network module, in: Proc. IEEE International WIE Conference on Electrical and Computer Engineering, Dec. 2017, pp. 1–5.
[18] L Hahn, L. Roese-Koerner1, K. Friedrichs, A. Kummert, Fast and reliable architecture selection for convolutional neural networks, in: Proc. European Symposium on Artificial Neural Networks, April 2019, pp. 179–184.
[19] F. Chollet, Deep Learning with Python, Manning Publications Co., USA, 2018.
[20] R. Venkata-Rao, Jaya: a simple and new optimization algorithm for solving constrained and unconstrained optimization problems, Int. J. Ind. Eng. Comput. 7 (2016) 19–34.
[21] R. Venkata-Rao, Jaya: An Advanced Optimization Algorithm and Its Engineering Applications, Springer, Cham, 2019.
[22] C. Molnar, Interpretable Machine Learning: A Guide For Making Black Box Models Interpretable, Leanpub, 2019.
[23] A. Fisher, C. Rudin, F. Dominici, All models are wrong, but many are useful: learning a variable's importance by studying an entire class of prediction models simultaneously, J. Mach. Learn. Res. 20 (2019) 1–81.
[24] X.-S. Yang, Z. Cui, R. Xiao, A. Hossein, M. Karamanoglu, Swarm Intelligence and Bio-Inspired Computation: Theory and Applications, Elsevier Inc., USA, 2013.
[25] R.A. Zitar, M.A. Al-Betar, M.A. Awadallah, I.A. Doush, K. Assaleh, An intensive and comprehensive overview of JAYA algorithm, its versions and applications, Arch. Comput. Methods Eng. 29 (2021) 763–792.
[26] M. Moocarme, M. Abdolahnejad, R. Bhagwat, Deep Learning with Keras, Packt Publishing, Birmingham, 2020.
[27] K.E. Parsopoulus, M.N. Vrahatis, "Particle swarm optimization and intelligence: advances and applications," USA: IGI Global, 2010.