Deep Learning for Power System Security Assessment

Arteaga, José-María Hidalgo; Hancharou, Fiodar; Thams, Florian; Chatzivasileiadis, Spyros

Published in:
Proceedings of IEEE Powertech 2019

Link to article, DOI:
10.1109/PTC.2019.8810906

Publication date:
2019

Document Version
Peer reviewed version

Deep Learning for Power System Security Assessment

José-María Hidalgo Arteaga
Center of Electric Power and Energy
Technical University of Denmark
s171800@student.dtu.dk

Fiodar Hancharou
Skolkovo Institute of Technology
Skolkovo, Russia
fiodar.hancharou@skoltech.ru

Florian Thams and Spyros Chatzivasileiadis
Center of Electric Power and Energy
Technical University of Denmark
{fltha, spchatz}@elektro.dtu.dk

***Abstract*—Security assessment is among the most fundamental functions of power system operator. The sheer complexity of power systems exceeding a few buses, however, makes it an extremely computationally demanding task. The emergence of deep learning methods that are able to handle immense amounts of data, and infer valuable information appears as a promising alternative. This paper has two main contributions. First, inspired by the remarkable performance of convolutional neural networks for image processing, we represent for the first time power system snapshots as 2-dimensional images, thus taking advantage of the wide range of deep learning methods available for image processing. Second, we train deep neural networks on a large database for the NESTA 162-bus system to assess both N-1 security and small-signal stability. We find that our approach is over 255 times faster than a standard small-signal stability assessment, and it can correctly determine unsafe points with over 99% accuracy.**

I. Introduction

Power system security assessment belongs to the most fundamental functions of every system operator. Its aim is to screen a wide range of possible operating points in order to identify a safe operating region and eliminate the possibility of a blackout. Power system operators run different types of security assessment at regular time intervals, ranging from intra-day to every year, checking each operating point against a defined set of instability types, including steady-state stability (e.g. N-1 security criterion), transient stability, small-signal stability, and voltage stability [1].

Assessing power system security for every possible operating point is an extremely computationally demanding task. For systems exceeding the size of a few buses, the problem becomes intractable due to the millions of possible operating points. Several approaches have been proposed in the literature, and some of them have also been applied in real power systems, to address this challenge, either through the definition of stability indices that are fast to calculate, through analytical reformulations and approximations (e.g. Lyapunov function [2]) or different computationally efficient methods [3], [4].

Machine learning for transient stability assessment has been applied for the first time in Ref. [5], in the form of decision trees. Since then, different approaches have been proposed for different problems, e.g. security assessment [6], controlled islanding [7], and others. With the recent very successful application of deep convolutional neural networks for image and speech recognition, and extremely difficult combinatorial problems (e.g. Go game), deep learning methods appear to offer a promising toolbox with significant potential for power system applications. Assessing power system security by deep learning, however, remains highly unexplored. Ref. [8] uses machine learning to assess power system security and devise remedial actions. Ref. [9] uses deep learning for extracting valuable features in order to build security rules that can distinguish between safe and unsafe points.

This paper has been accepted at IEEE Powertech 2019.

This paper has two main contributions. First, taking advantage of the wide range of deep learning methods for image processing, we introduce for the first time appropriate 2-D representations of power system snapshots as images. Second, using a large database of operating points for the NESTA 162-bus system, we train different deep neural networks for N-1 security *and* small-signal stability assessment, and investigate their performance. Finally, we also briefly discuss ways to integrate such deep neural networks in a security-constrained optimal power flow framework.

This paper is structured as follows. Section II presents the methodology we developed to represent power system snapshots and the structure of the neural network we created. Section III presents the results of our case studies on the NESTA 162-bus test system. Section IV discusses possible extensions. Section V concludes.

II. Methodology

The deep learning algorithm we introduce is based on convolutional neural networks (CNN), which have been successful for applications in image recognition. Inspired by that, in this paper we develop a representation of power system snapshots by images, which we can then feed directly to the convolutional neural network. Considering the rapid development of powerful deep learning algorithms for image processing, an appropriate representation of power system snapshots as images can take advantage of all the state-of-the-art methods developed in the artificial intelligence community. In the following sections, we first present the input data that we used for the training and testing of the neural network, and then we continue with the methodology to create the appropriate images, and the structure of the CNN.

A. Training and Testing Database

Our input data are generated for the NESTA 162-bus system [10]. In our previous work, we have introduced an efficient database generation method, which manages to generate hundreds of thousands of datapoints around the security boundary of the system about 10-20 times faster than other state-of-the-art methods [11]. Using this algorithm, we assess the N-1 security and small signal stability of a large dataset of

operating points (about 1,000,000) for the NESTA 162-bus test system [10].

Each operating point in the database is represented by bus voltages, angles, net active and reactive power demand at each node and active and reactive power in each line. The security assessment is based on small-signal stability analysis for a subset of the possible N-1 contingencies $\bar{C} = [C_1, C_2, ..., C_n]$ in the NESTA 162-bus system. An operating point is considered safe if the damping ratio ζ for all contingencies $\bar{C}$ is higher than 3%, i.e. if $\min(\zeta(\bar{C})) \geq 0.03$.

B. Image Generation

The images we generate shall (uniquely) represent the state of a power system. To do that, we need to associate the variables extracted from power flows with specific locations in multidimensional arrays. In image recognition algorithms, the images are turned into 2D arrays filled with numerical values that the convolutional neural network can understand. Each colour image is usually converted to three arrays or channels, typically *RGB*, that represent the intensity of red, green and blue colour in each pixel. Following this method, the images created from power system data are also divided in three channels *PQV* that represent the active power, reactive power, and voltage in each snapshot of the power system.

The 2D arrays associated with channels *P* and *Q* will be $N \times N$ matrices, where *N* is the number of buses in the 162-bus case. The diagonal of each matrix has the net active P_i and reactive Q_i power demands at each bus i respectively. If bus *i* and bus *j* are connected by a line, the active P_{ij} and reactive Q_{ij} power flows are placed in the position (i, j) of the matrix while P_{ji} and Q_{ji} are placed in (j, i). Since power losses in the lines are also taken into account, these matrices are not symmetrical. The data from each channel is previously normalised by calculating the absolute value of each element and then dividing by the maximum value of *P* or *Q* in the whole database. All normalised values are between 0 and 1.

The 2D array associated with channel *V* has also size $N \times N$, and its diagonal is filled with the voltages V_i in each bus. The off-diagonal elements, $i \neq j$, are filled with the absolute values of the voltage drops V_{ij} along each line. Unlike the other two channels, matrices in channel V are symmetrical. The data in this channel (bus voltage and line voltage drop) are normalised as well, similarly to the other channels. In the three channels *PQV*, the matrices are sparse because the power system has only 284 lines for the 162 buses. An example of an image can be seen on Figure 1, where the three channels *PQV* are superimposed and plotted as if they were *RGB*, creating a coloured picture. The coloured pixels are always placed in the same spots but with different intensities. Even though all snapshots are different, two different snapshots are usually visually similar with a naked eye – but not for a computer. In his Nobel lecture in 2011 [12], Saul Perlmutter explained that an early way of searching supernovae was to take images from distant galaxies periodically and subtract the negative from an older image to the latest image. Both images would look the same but when there is a supernova, the subtraction leaves a single white point over a dark background. If this same procedure is followed with our created images, it will be possible to spot the differences in all snapshots. An image $X_s \in \mathbb{R}^{N,N,c}$ created from a snapshot s of the power system

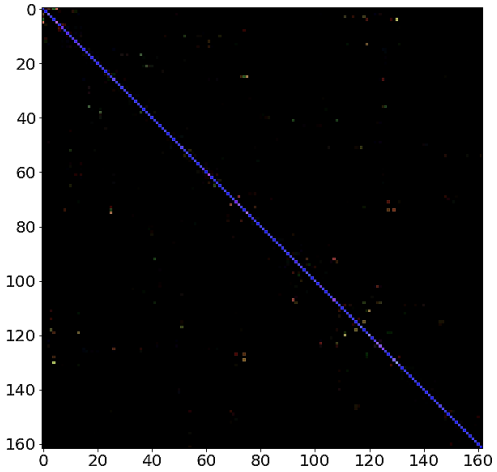

Fig. 1. Example of image representing a snapshot of NESTA 162-bus power system

is a 3D tensor[1] with size $[N, N, c]$, and c being the number of channels. The total input data set $X \in \mathbb{R}^{S,N,N,c}$ with all snapshots turned into images becomes a 4D tensor with $[S, N, N, c]$ size, where S is the number of snapshots that the database contains. Due to the large size of X (104.8 GB), only small batches, $X_b \in \mathbb{R}^{b,N,N,c}$ where b is the batch-size and $b << S$, are generated when needed in the training process.

The output data, a vector containing zeros and ones depending on the security state of each snapshot, is turned into labels by one-hot encoding having as a result an output data set $y \in \mathbb{R}^{S,N_c}$ where N_c is the number of labels: in our case two (safe/unsafe). In [13], one-hot encoding is explained as a method to encode categorical variables having n categories as n-dimensional feature vectors. For each category, one of the positions of the vector is filled by a one and the rest by zeros, resulting in a vector space where each category is orthogonal and equidistant to the rest. *Unsafe* snapshots are now represented as [1, 0] while *safe* snapshots as [0, 1].

Once the input X and output y sets are constructed, all the data is split into training (X_{train} and y_{train}), validation (X_{val} and y_{val}) and test sets (X_{test} and y_{test}). The training set contains 70% of the total set while the validation and test sets contain the remaining 10 % and 20% of the data respectively. Initially, the dataset is shuffled to assure that all subsets have the same share of safe cases (about 14.5%). As X does not fit in the memory of the computers used to train the CNN, it is impossible to create the full size tensor X before training. Instead, we split the number of snapshots S in training, validation and test sets, and generate smaller batches X_b that we use as input to the CNN.

C. Structure of the Deep Neural Network

Convolutional neural networks (CNN) have been designed for processing data in the form of multiple arrays. As explained above, our model has to process three 2D arrays that represent P, Q and V in the studied power system. The structure of a convolutional net has several stages. The initial stages are

[1]The term "tensor" refers to multidimensional arrays or vectors, used in TensorFlow (https://www.tensorflow.org/), the machine learning library we use. We will use the terms "tensor" and "array" interchangeably.

composed by a series of convolutional and pooling layers. The output of these initial stages is reshaped in a flattening layer and then fed into fully-connected layers. The typical structure of a CNN is shown in Fig. 2. The whole convolutional net

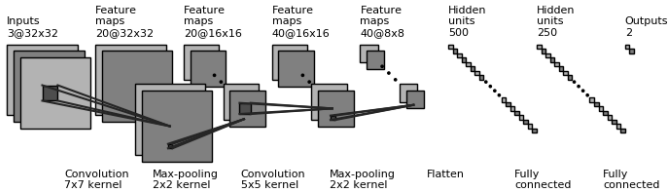

Fig. 2. Structure of a typical CNN. For the exact structure of the CNN developed for this paper see Table I.

works with the same principle as a multi-layer perceptron (MLP). Equation (1), where g is the activation function and W_{ij} and b_j are the weights and biases from node i to node j respectively, shows how the output of each node is calculated:

$$X_j = g(\sum_{i=1}^{F}(W_{ij}X_i + b_j)) \tag{1}$$

The different layers that form the CNN, as well as its training process, are explained in the following paragraphs.

1) Convolutional Layer: Convolutional layers take as input 2D arrays and train a series of 2D kernels or filters that aim to recognise patterns in the input arrays giving the local weighted sum as an output.

The input to any convolutional layer is a 4D batch X_b with $[b, H, W, F_{i-1}]$ size that is turned into a $[b, H, W, F_i]$ tensor where H and W are the height and width of the 2D arrays respectively and F_i and F_{i-1} are the number of filters or nodes in layer i and in the previous one respectively.

The weights in a convolutional layer are a 4D tensor with a $[K, K, F_{i-1}, F_i]$ size, with K being the kernel size, while the biases are a 1D tensor of F_i size. The kernel size and the number of filters are tuning parameters chosen by the user.

The most commonly used activation function in convolutional layers is the Rectified Linear Unit (ReLU) [14], whose function is given by $f(x) = max(0, x)$ [15].

2) Max-Pooling Layer: The pooling layers do not have weights and biases associated and their aim is to reduce the spatial size of the convolutional layers' output. Max-pooling is the most commonly used type of pooling, and also the one used in our algorithm. Max-pooling layers apply a moving window, in our case of size 2×2, over a 2D input and give the maximum value from every sub-region covered by the window.

This type of layer is applied after each convolutional layer, giving as an output a $[b, \frac{M}{2}, \frac{N}{2}, F_i]$ size tensor. The activation function is applied after the max-pooling layer, instead of right after the convolutional layer, because it is less costly due to its smaller sized input.

3) Flattening Layer: Flattening layers connect convolutional or pooling layers to fully-connected layers. This type of layer reshapes the output of the last max-pooling layer into a 1D array per snapshot in order to be able to feed the first fully-connected layer. The size of the output of this layer $[b, M \cdot N \cdot F_i]$ is the result of compacting the three dimensions from the previous layer $[b, M, N, F_i]$.

4) Fully-connected Layer: here, each node is connected to all the nodes from the previous layer. The output of each layer is a F_i-size vector per snapshot. The first fully-connected layer is fed by the flattening layer. The weights in these layers are a 2D tensor with a $[F_{i-1}, F_i]$ size while the biases are a F_i-size vector. In all fully-connected layers except the last one, 20% of the nodes are disconnected while training in order to avoid over-fitting (i.e. 20% dropout rate).

All fully-connected layers except for the output layer are activated by a ReLU function. The output layer, which is the one that finally performs the classification, is activated by a Softmax function or normalised exponential function [16]. Softmax equations are shown in (2) and (3), where σ represents the output, N_c is the number of classes and X_i is the input vector. The Softmax function turns a N_c-dimensional vector of real values into a vector of the same size with values between 0 and 1 where all elements sum 1. This function is used in the output layer in order to get the probability distribution over the different classes, in our case safe and unsafe. The CNN chooses the class with the highest probability to occur as the predicted security state.

$$\sigma : \mathbb{R}^{N_c} \rightarrow \{\sigma \in \mathbb{R}^{N_c} \mid \sigma_i > 0, \sum_{i=1}^{N_c} \sigma_i = 1\} \tag{2}$$

$$\sigma(X)_i = \frac{e^{X_i}}{\sum_{i=1}^{N_c} e^{X_i}} \quad for \quad i = 1, ..., N_c \tag{3}$$

D. Training the Neural Network

In order to measure the classification performance of our neural network during training, we need to define a loss function. The loss function is based on the cross entropy between the estimated and real output probability distributions. The cross entropy equation across m snapshots is shown in (4) as the first term, where y_i and $\hat{y}_i$ are the real and predicted probabilities of class i respectively. The first term of the cross entropy, which corresponds to the positive class predictions, is multiplied by a class coefficient Φ in order to penalise more false positive predictions ($\Phi > 1$) or false negative predictions ($\Phi < 1$). The coefficient Φ is useful when the labels of the data are unbalanced. Based on the loss function presented in [9], the estimated recall $\hat{Rec}$, specificity $\hat{Spe}$, precision $\hat{Pre}$ and F1-score $\hat{F1}$, whose definitions are explained later, are added multiplied by the negative costs α_r, α_s, α_p and α_f respectively. These costs are negative because we aim to maximise the algorithm's performance on those metrics. The total loss function includes also L2 regularisation at the end in order to avoid over-fitting and also penalise large values of weights.

$$\begin{aligned} Loss = &-\frac{1}{m}\sum_{i=1}^{m}[\Phi y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)] \\ &- \alpha_r \hat{Rec} - \alpha_s \hat{Spe} - \alpha_p \hat{Pre} - \alpha_f \hat{F1} + \frac{\lambda}{2}\sum_{j=1}^{N_L} W_j^2 \end{aligned} \tag{4}$$

We use the Adam optimiser [17], which is an algorithm based on stochastic gradient descent, to minimise the loss function while tuning the trainable parameters of the neural network: weights and biases. The optimiser is initialised with

an initial learning rate chosen by the user. The CNN is trained by applying a mini-batch gradient descent method, i.e. the parameters are updated every time a batch, smaller than the training set, goes through the optimiser. These small batches are created and overwritten right after their use in order to be able to fit this learning process in the available memory space. The whole training set, and therefore all the batches, go through the optimiser in every epoch. After every epoch, the model is validated in the validation set. The final model will be the one with the highest validation accuracy in all the epochs. Before training, a large number of epochs is chosen but in order to avoid over-fitting, early stopping is applied with a patience parameter chosen by the user. This means that if the validation loss is not decreasing over the epochs, the training stops and the saved weights and biases are the ones corresponding to the epoch with the highest validation accuracy.

E. Proposed Model

The proposed CNN architecture initially has three consecutive series of convolutional and max-pooling layers. The kernel size of the layers is 9×9, 7×7 and 5×5 for the first, second and third convolutional layers while the number of filters is 20, 40 and 80 respectively. The kernel size decreases from the first layer because the input to the convolutional layers gets smaller. This allows to increase the number of filters in the next layers.

Figure 3 shows the weights corresponding to the first convolutional layer. This image has been created by normalising the weights and superimposing the three input channels *PQV* into one as if they represented *RGB* channels. There is twenty 9×9 images due to the number of filters and their size in the first convolutional layer. Red, green and blue colours represent large weights in channels *P*, *Q* and *V* respectively and small in the rest. Black colour represents small weights in the three channels while white colour shows large weights in all of them. This colourful image can only be obtained for the first layer since in the following ones, the inputs are more than the initial three. The flattening layer is applied after the last max-pooling layer. Then, the first fully-connected layer, with 250 nodes is fed. This layer feeds directly to the output layer which has 2 nodes, one per label. Table I shows the size of each layer as well as the shapes of the weights and biases. The number of parameters to train in each layer is also represented. The number and size of the filters as well as the number of nodes have been chosen heuristically. The optimiser is fed with a batch-size of 128 snapshots, and a maximum of 200 epochs with a patience of 30 epochs. The initial learning rate is set to 0.001.

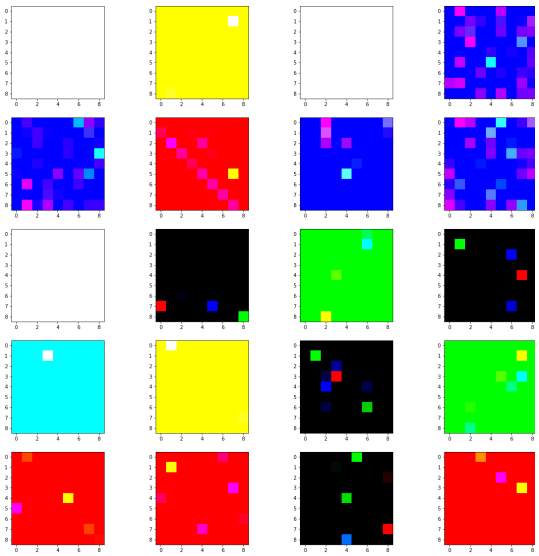

Fig. 3. Image representing the weights in Conv1.

TABLE I
REPRESENTATION OF CNN ARCHITECTURE.

Layer	Shape	Weights size	Bias size	# param
Input	[b,162,162,3]			
Conv1	[b,162,162,20]	[9,9,3,20]	[20]	4.880
Max-pool1	[b,81,81,20]			
Conv2	[b,81,81,40]	[7,7,20,40]	[40]	39.240
Max-pool2	[b,40,40,40]			
Conv3	[b,40,40,80]	[5,5,40,80]	[80]	80.080
Max-pool3	[b,20,20,80]			
Flatten	[b,32000]			
FC1	[b,250]	[32000,250]	[250]	8.000.250
FC2 (output)	[b,2]	[250,2]	[2]	502
Total				**8.124.952**

The model has been implemented using the Python open source machine learning library TensorFlow for GPU and trained in a single Tesla V100 GPU core from DTU's HPC cluster for 23 hours (DTU cluster limits the time use of each job to a maximum of 24 hours). For our simulations, this time was enough for the validation error to converge to a minimum, even though there were always some small fluctuations in the error during the training process. It is believed that training the CNN in several GPU cores in parallel would speed up the process and improve the results further.

III. RESULTS

In this section, the results obtained by the presented model are shown. First, we describe the evaluation metrics used to measure how good our model is. Subsequently, we present the scores obtained by testing the model, and at the end we show the substantial reduction of computation time by using the presented model over traditional methods.

A. Evaluation Criteria and Model Performance

The evaluation criteria used are based on the confusion matrix, shown in Table II. The confusion matrix is filled with the True Positives (TP), referring to the number of snapshots predicted as unsafe that are actually unsafe, True Negatives (TN), i.e., the number of snapshots predicted as safe that are actually safe, and False Positives (FP) and False Negatives (FN) as the snapshots that have been predicted as the opposite class from what they actually are. The following measures are extracted from the different elements in the confusion matrix.

TABLE II
CONFUSION MATRIX.

	Predicted Unsafe	Predicted Safe
Actual Unsafe	TP	FN
Actual Safe	FP	TN

- **Recall**: Also named as Sensitivity or True Positive Rate (TPR), it is the proportion of correct positive predictions in all the positive cases. Calculated as $Recall = \frac{TP}{TP+FN}$.
- **Specificity**: Also called True Negative Rate (TNR), it is the proportion of correct negative predictions in all the negative cases. Calculated as $Specificity = \frac{TN}{TN+FP}$.
- **Precision**: Also known as Positive Predictive Value (PPV), is the proportion of correct positive predictions in all positive predictions. Calculated as $Precision = \frac{TP}{TP+FP}$.
- **F1-Score**: The harmonic mean of *Precision* and *Recall*. Calculated as $F1 = 2\frac{Precision \cdot Recall}{Precision+Recall}$.
- **Accuracy**: The proportion of correct predictions in all data set. Calculated as $Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$.
- **Matthews correlation coefficient**: A measure of correlation between predicted and actual values in a binary classification. It yields a coefficient between -1 and 1, with 1 being when the prediction is perfect, 0 when the prediction is no better than random, and -1 when all predicted values are mistaken. Calculated as $MCC = \frac{(TP \cdot TN)-(FP \cdot FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$.

The measure of accuracy is often enough to measure the quality of the classification for a database with the same number of safe and unsafe operating points. Since this is not the case in our problem, where only 14.56% of the cases are safe, other measures become also meaningful when evaluating the classification. It also needs to be mentioned that it is substantially more important to predict correctly unsafe cases than safe cases. In our case, a FN would have a much higher negative impact (and cost) for the power system than a FP. For that reason, it is more important to have a high *Recall* than a high *Specificity*.

As it has just been mentioned, the database labels are unbalanced, containing a substantially higher number of unsafe cases compared with safe cases. If we were looking for balancing the cross entropy loss, the balanced class coefficient Φ_b would be $\Phi_b = \frac{S_{safe}}{S_{unsafe}} = 0.17$ where S_{safe} and S_{unsafe} are the number of safe and unsafe snapshots respectively. Since we aim to minimise the FNs because predicting correctly the unsafe cases is more important, the class coefficient has to be $\Phi > \Phi_b$; in our studied cases, as shown in Table III it was $\Phi \geq 1$. The metric costs vector is defined as $\bar{\alpha} = [\alpha_r, \alpha_s, \alpha_p, \alpha_f]$ (see (4)). The metrics directly related to predicting unsafe cases correctly and therefore decreasing FNs are *Recall*, *Precision* and *F1-Score*, while *Specificity* is related to the safe cases prediction. Table III shows the different studied cases, with different values of Φ and $\bar{\alpha}$, and Figure 4 shows the best scores of our model in the test set in each case. The radar chart aims to demonstrate how the variation of Φ and $\bar{\alpha}$ can influence the training outcome. Note that the scale of each axis in this diagram is specified under each label.

TABLE III
CASES DEPENDING ON Φ AND $\bar{\alpha}$.

Cases	Φ	$\bar{\alpha} = [\alpha_r, \alpha_s, \alpha_p, \alpha_f]$
1	1	[0, 0, 0, 0]
2	2	[0, 0, 0, 0]
3	5	[0, 0, 0, 0]
4	1	[0.5, 0.5, 0.5, 0.5]
5	1	[0.5, 0, 0.5, 0.5]
6	2	[0.5, 0, 0.5, 0.5]
7	3	[0.5, 0, 0.5, 0.5]
8	2	[0, 0, 0.5, 0.5]

Cases 1-3 show the variation of Φ while $\bar{\alpha}$ is filled with zeros. It can be observed that a value close to the optimal Φ could be around 2 since increasing it more is counterproductive. In case 4, all the values from $\bar{\alpha}$ are increased while Φ stays 1 and the results are improved in comparison with case 1. The addition of evaluation metrics in the loss function results in an improvement of performance. Increasing Φ and $\bar{\alpha}$ too much can result in worse scores due to the trade-off between the importance of the different terms in the loss function. Since having a high *Specificity* is less important than the rest, cases 5-7 have α_s as zero while the rest of elements in $\bar{\alpha}$ stay 0.5. It can be observed again that when Φ equals 2, in case 6, there is a peak of performance. In case 6 we obtain the highest score in *Recall* with a 99.14% meaning that from all the combinations tried, the one that predicts the best the unsafe states is case 6. Investigating the confidence interval, we found that with 99% likelihood the score in *Recall* in case 6 will belong to the confidence interval 99.14% $\pm$ 0.092%. On the other hand, in case 8 where both α_r and α_s are zero, the model scores the best in *F1-Score*, MCC and accuracy with a 99.14%, 0.942 and 98.54% respectively. Again, the 99% confidence intervals of these scores in case 8 are *F1-Score*: 99.14% $\pm$ 0.092%, *MCC*: 0.942 $\pm$ 0.00097, and *Accuracy*: 98.54% $\pm$ 0.12%. In this case, the overall classification performs better than in the rest of cases but the correct classification of unsafe classes decreases in exchange of a rise in MCC and accuracy.

In order to identify a pattern in the miss-classified operating points in case 8, the whole operating area covered by the database is split in clusters by *k-means* clustering [18]. This technique is applied several times for a number of 3, 5, 10 and 20 clusters. Once the operating space is divided in clusters, the clusters where the miss-classified points belong are identified. This gives as a result that at least the 99% of the miss-classified points belong always to the same cluster. This means that most of these points belong to a small specific region of the security boundary that the CNN failed to model.

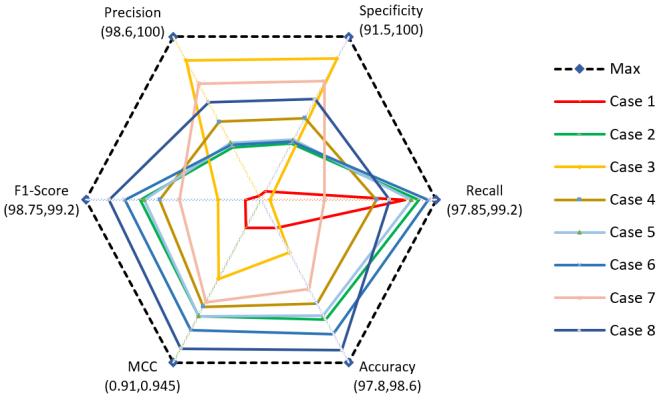

Fig. 4. Radar chart of the scores in the different cases. Table III describes the used parameters for each case. Note that each axis has a slightly different range in order to enhance the readability of the figure.

B. Computation Time

The main goal of applying deep learning techniques to power system security assessment is to devise algorithms that provide results much faster than the standard small-signal stability analysis methods for all possible contingencies. For that reason, we measure the time needed to perform a security assessment with the presented model and compare it to the time spent in computing all necessary small-signal stability analyses. To ensure a fair comparison, (i) both methods are tested in the same machine: a single core from an Intel Xeon Processor E5-2650 v4 of the DTU HPC cluster, and (ii) the computation time reported is the *average* time after performing the security assessment on a single snapshot for 1000 times. Table IV reports the average time per test over these 1000 tests. As it can be observed in Table IV, the time spent to perform a security assessment for the studied power system is reduced about 255 times, making the presented model way more efficient for security assessment than traditional methods.

TABLE IV
TIME TESTS: SMALL-SIGNAL STABILITY ANALYSIS VS. CONVNET MODEL.

	Small-signal stability analysis	**ConvNet model**
Average Time	21'950 ms	86 ms

IV. DISCUSSION AND POSSIBLE EXTENSIONS

The used training database (online available, see [11]) contains all possible N-1 contingencies for all possible operating points for a given demand profile (over 1,000,000 points). Topology changes in the form of N-1 investigations are already considered. Future work shall consider a wider range of topology changes and uncertainty in demand.

Inspired by our previous work, where we have reformulated Decision Trees for stability assessment to a Mixed Integer Linear Program [19] or Mixed Integer Second Order Cone Program [20] in order to fully consider N-1 security and small signal stability in Optimal Power Flow, in future work we will also examine extensions of our deep learning algorithm to AC-OPF formulations. Through an iterative framework, the CNN can help determine the binding constraints that lead to a N-1 secure and small-signal stable point, which can then be inserted in the optimization problem. In our preliminary investigations, the average computation time of such a setup after carrying out 50 tests in an Intel Core i7-8550-U CPU is 19.99s. Future work will focus on computation efficiency and convergence guarantees.

V. CONCLUSIONS

This paper has presented a method based on convolutional neural networks and deep learning for power system security assessment. There are two main contributions. First, we represent for the first time power system snapshots as images that can be easily processed by convolutional neural networks. Acknowledging the fact that safe or unsafe operating points exhibit similar patterns, we represent power flows between buses as 2-dimensional images. Taking advantage of the wide range of deep learning methods available for image processing, we can achieve a remarkable performance of deep neural networks for power system security assessment. Second, we develop and extensively test different deep neural networks to assess N-1 security *and* small-signal stability on the NESTA 162-bus system. We find that our approach is over 255 times faster than a standard small-signal stability assessment for a single operating point, while it achieves an accuracy of over 98% and a Recall of over 99.14% (i.e. classifying unsafe states as unsafe). As discussed in the last section, future work will focus on the integration of deep learning for power system security assessment to an optimal power flow framework.

ACKNOWLEDGMENT

This work has been partially supported by the multiDC project, funded by Innovation Fund Denmark, Grant Agreement No. 6154-00020B.

REFERENCES

[1] P. Kundur et al., "Definition and classification of power system stability ieee/cigre joint task force on stability terms and definitions," *IEEE Trans. on Power Systems*, vol. 19, no. 3, pp. 1387–1401, Aug 2004.

[2] T. L. Vu, S. Chatzivasileiadis, H. Chiang, and K. Turitsyn, "Structural emergency control paradigm," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 7, no. 3, pp. 371–382, Sept 2017.

[3] K. Morison, L. Wang, and P. Kundur, "Power system security assessment," *IEEE Power and Energy Magazine*, vol. 2, no. 5, pp. 30–39, 2004.

[4] T. Van Cutsem and C. Vournas, *Voltage stability analysis of electric power systems*. Kluwer Academic Publishers, 1998.

[5] L. A. Wehenkel, *Automatic Learning Techniques in Power Systems*. Boston/London/Dordrecht: Kluwer Academic Publishers, 1998.

[6] C. A. Jensen, M. A. El-Sharkawi, and R. J. Marks, "Power system security assessment using neural networks: feature selection using fisher discrimination," *IEEE Transactions on Power Systems*, vol. 16, no. 4, pp. 757–763, Nov 2001.

[7] S. Koch, S. Chatzivasileiadis, M. Vrakopoulou, and G. Andersson, "Mitigation of cascading failures by real-time controlled islanding and graceful load shedding," in *2010 iREP Symposium - Bulk Power System Dynamics and Control - VIII (iREP), Buzios, Brazil*, Aug. 2010.

[8] B. Donnot, I. Guyon, M. Schoenauer, P. Panciatici, and A. Marot, "Introducing machine learning for power system operation support," in *2017 iREP Symposium - Bulk Power System Dynamics and Control - X (iREP), Porto, Portugal*, August 2017, pp. 1 –10.

[9] M. Sun, I. Konstantelos, and G. Strbac, "A deep learning-based feature extraction framework for system security assessment," *IEEE Transactions on Smart Grid*, pp. 1–1, 2018.

[10] C. Coffrin, D. Gordon, and P. Scott, "NESTA, The NICTA Energy System Test Case Archive," 2014. [Online]. Available: http://arxiv.org/abs/1411.0359

[11] F. Thams, A. Venzke, R. Eriksson, and S. Chatzivasileiadis, "Efficient database generation for data-driven security assessment of power systems," *IEEE Transactions on Power Systems*, 2018, online avaiable: https://arxiv.org/abs/1806.01074.

[12] S. Perlmutter, "Nobel lecture: Measuring the acceleration of the cosmic expansion using supernovae," *Rev. Mod. Phys.*, vol. 84, August 2012.

[13] P. Cerda, G. Varoquaux, and B. Kégl, "Similarity encoding for learning with dirty categorical variables," *Machine Learning*, vol. 107, no. 8, pp. 1477–1494, Sep 2018. [Online]. Available: https://doi.org/10.1007/s10994-018-5724-2

[14] A. F. Agarap, "Deep learning using rectified linear units (relu)," online: https://arxiv.org/abs/1803.08375, 03 2018.

[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[16] B. Gao and L. Pavel, "On the properties of the softmax function with application in game theory and reinforcement learning," online: https://arxiv.org/pdf/1704.00805.pdf, 2017.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[18] S. Na, L. Xumin, and G. Yong, "Research on k-means clustering algorithm: An improved k-means clustering algorithm," in *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*, April 2010, pp. 63–67.

[19] F. Thams, L. Halilbasic, P. Pinson, S. Chatzivasileiadis, and R. Eriksson, "Data-driven security constrained opf," in *2017 iREP Symposium - Bulk Power System Dynamics and Control - X (iREP), Porto, Portugal*, August 2017, pp. 1 –10.

[20] L. Halilbasic, F. Thams, A. Venzke, S. Chatzivasileiadis, and P. Pinson, "Data-driven security-constrained ac-opf for operations and markets," in *20th Power Systems Computation Conference (PSCC)*, June 2018.