Article

Flattening Layer Pruning in Convolutional Neural Networks

Ernest Jeczmionek 1,† and Piotr A. Kowalski 2,3,*,†

1 AGH Doctoral School, AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Cracow, Poland; ejec@agh.edu.pl
2 Faculty of Physics and Applied Computer Science, AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Cracow, Poland
3 Systems Research Institute, Polish Academy of Sciences, ul. Newelska 6, 01-447 Warsaw, Poland
* Correspondence: pkowal@agh.edu.pl
† These authors contributed equally to this work.

Abstract: The rapid growth of performance in the field of neural networks has also increased their sizes. Pruning methods are getting more and more attention in order to overcome the problem of non-impactful parameters and overgrowth of neurons. In this article, the application of Global Sensitivity Analysis (GSA) methods demonstrates the impact of input variables on the model’s output variables. GSA gives the ability to mark out the least meaningful arguments and build reduction algorithms on these. Using several popular datasets, the study shows how different levels of pruning correlate to network accuracy and how levels of reduction negligibly impact accuracy. In doing so, pre- and post-reduction sizes of neural networks are compared. This paper shows how Sobol and FAST methods with common norms can largely decrease the size of a network, while keeping accuracy relatively high. On the basis of the obtained results, it is possible to create a thesis about the asymmetry between the elements removed from the network topology and the quality of the neural network.

Keywords: global sensitivity analysis; Sobol procedure; fast algorithm; convolutional neural network; structure reduction; pruning; quality

1. Introduction

Since the beginning of the twenty-first century, computational intelligence (CI) [1], in the guise of artificial intelligence and machine learning, has been experiencing great strides in its development, both practically [2] and theoretically [3]. The term embraces fuzzy logic, genetic and evolutionary algorithms, swarming intelligence, rough sets and artificial neural networks. Artificial neural networks (ANNs) are computational systems developed on the basis of the biological structure of the brains of living organisms, and they consist of objects representing neurons and the connections between them. In ANNs, there are layers made of neurons. The first (input layer) is used to enter data into the network, and the last (output layer) returns the generated results [4], while a number of hidden layers exists in between. As the number of layers increases, the network processing time automatically increases. Therefore, the optimization of the structure of the neural network is an important issue.

ANNs play a very important and often unheralded role in the modern world. Calculations made with their help can be found, among others, in forecasting air pollution [5] or when conversing via cell phone [6]. A popular and recent development is the use of neural networks in natural language processing, in particular, for text generation [7], automatic text translation [8], text analysis [9], spam message detection [10] and spoken text recording [11]. Due to their versatility and the possibility of modeling non-linear processes, ANNs are used in the automotive industry (navigation systems, autopilot), telecommunications and robotics [12].
Convolutional neural networks (CNNs) were introduced in 1980–1982, under the term “neocognitron” [13], but their rather dynamic development began to be felt only from around 2010. Since then, many ready-made CNN neural networks have been created. Indeed, it can be said that this type of neural network is the basic structure in the class of deep neural networks [14].

CNNs require the use of high-powered computers, as they are, in reality, often oversized for purpose. To prevent the exaggerated growth of size, pruning methods are needed and have been subjected to research. There are two categories of pruning based on purpose: pruning for performance [15] or pruning for size [16]. The method shown in this article can be applied in order to minimize the number of reduction cycles and the number of neurons in each reduction cycle.

Sensitivity analysis methods were designed to assess the impact of a model’s input on its output. Two major subgroups are Local Sensitivity Analysis (LSA) [17] and Global Sensitivity Analysis (GSA) [18]. The first method measures sensitivity by varying only one input parameter, while GSA changes all inputs simultaneously.

The main focus of this article is to propose a reduction layer based on sensitivity analysis that is combined with a flattening layer. The presence of a large number of convoluted feature matrices and the substantial size of first fully connected layer generates enormous numbers of permutation. In some cases, these weights are responsible for 90% of the total parameters in the network. This research concentrates on minimizing unnecessary connections between the convolutional layer and the fully connected network.

In a large number of algorithms, the removal of certain elements from their structure determines the simplification of such procedures. Unfortunately, such symmetry reduces the quality of the results obtained. In the proposed algorithm for reducing the structure of the neural network, asymmetry between the reduction in the topological structure of the neural network and the obtained results is observed. The above is largely realized by way of the use of sensitivity analysis precursors, thanks to which the weakest links of the tested CNN flattening layer are determined.

Decomposition and pruning are common techniques applied to compress the architectures of neural networks. Tucker decomposition is a well-known Low-Rank factorization method to decompose both convolutional and fully connected layers [19]. Another popular method is tensor rank decomposition. This is based on the superdiagonal core tensor of Tucker decomposition [20,21]. Filter pruning is a natural approach in CNN compression. Moreover, a team of Nvidia researchers has presented a kernel pruning algorithm based on a minimization of the Taylor series expansion of the error [16]. The attention mechanism introduced by a Google research team [22] has also been applied to prune CNN with regard to the classification problem [23]. Both decomposition and pruning methods can be combined for better network compression [24].

This article is divided into the following sections. Section 2 describes the concept of a convolutional neural network, and it includes a discussion on what is sensitivity analysis and what methods are utilized in the research. Section 3 presents the algorithm used for a proposed reduction method. Section 4 consists of a description of the datasets and an analysis of the results obtained from applying variants of the reduction algorithm. The last section, Section 5, summarizes the results and and provides the obtained conclusions.

2. Methods

2.1. Convolutional Neural Network

The convolutional neural network (CNN) is a modern architecture of a neural network used in anomaly detection [25] and natural language processing (such as in sentence modeling) [26], as well as in classification [27].

The major area of CNN application is in computer vision, including object detection [2], image classification [28] and segmentation [29]. CNNs that have been around for a few years, such as GoogLeNet, present human-like accuracy of classification [30], and performance of these networks has been under continuous improvement. In this paper, attention
is focused upon the classification problem. CNN, mostly applied on problems concerning images, is built upon the input image undergoing a series of convolutional operations. The mathematical formula for convolution operation is presented in Equation (1).

\[
\begin{bmatrix}
A_{1,1} & \ldots & A_{1,N} \\
\vdots & \ddots & \vdots \\
A_{M,1} & \ldots & A_{M,N}
\end{bmatrix}
\ast
\begin{bmatrix}
B_{1,1} & \ldots & B_{1,N} \\
\vdots & \ddots & \vdots \\
B_{M,1} & \ldots & B_{M,N}
\end{bmatrix}
= \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} A_{(M-i), (N-j)} B_{(i+1), (j+1)}
\] (1)

This CNN includes many kernels (also called ‘filters’). The task of the kernel is to learn feature extraction. If many convolutional layers are stacked, the first layers are responsible for deriving so-called ‘high level features’, such as the image’s basic outlines or curves. The following layers, through the application of matrices of convoluted features, extract more and more detailed characteristics. Generally, multiple convolutions of an image would generate large matrices, extending computation time and memory usage. To resolve this issue, a ‘pooling layer’ was introduced. Its first purpose is to reduce the size of a matrix by applying the functions of a pooling kernel. The most common of these are averaging and maximizing. The second task of the pooling layer is to reduce non-dominant properties by leaving only the most important feature, hence reducing image ‘noise’. Convolutional and pooling layers are put together in many combinations. A popular approach to CNN modeling is to form a stack of one or two convolutional layers, followed by a pooling layer.

The two previously described layers are responsible for learning an image’s features. They result in a vector of low-level convoluted features matrices. To assign an image to a category, a classification network is required. Before this can be done, the output of convolutional and pooling layers has to be flattened to a single vector. A Fully Connected Network (FCN) is a feed-forward network, the purpose of which is to learn the likelihood of membership to a category. However, having an output that is a vector of the number of features assigned to each individual class is not desirable. To simplify the results and make these more understandable, a softmax function is incorporated. Softmax, acting in the form of an activation function in the last layer, returns only one value assigned to a category, with the highest number of features corresponding to that category being grouped together.

In this article, two CNNs were used, Figure 1 for 2D datasets and Figure 2 for 1D datasets. For faster convergence, a dropout layer was added to each CNN. This layer zeroes weights with set probability. Hence, all networks end with a FCL of a size corresponding to the number of categories in the dataset, and they use ReLU as an activation function. In both the 1D CNN and the 2D CNN, categorical cross-entropy is considered a loss function. Their treatment differs in that the Adam optimizer was set in the 1D CNN and a Stochastic Gradient Descent was established in case of 2D CNN. The 1D datasets of the 1D CNN are composed of double 64 3 × 3 kernels, followed by a dropout layer with probability of 50%. Subsequently, a max pooling layer of size 2 × 2 and consisting of 100 neurons in a fully connected layer is attached. 2D CNN is more complicated and is structured with a double sequence of two 32 3 × 3 filters, followed by a 2 × 2 max pooling layer and the 25% dropout layer. The 2D CNN ends with a 512 neuron FCL and a 50% dropout layer. Table 2 lists the number of neurons in the first layer of the FCN, the total number of parameters in the CNN, the number of frozen non-trainable parameters in their convolutional layers and, finally, the number of trainable parameters in FCN.
Figure 1. CNN model architecture dedicated for 2D image input.

| Layer (type)            | Output Shape        | Param # |
|-------------------------|---------------------|---------|
| conv2d_4 (Conv2D)       | (None, 26, 26, 32)  | 320     |
| conv2d_5 (Conv2D)       | (None, 24, 24, 32)  | 9248    |
| max_pooling2d_2 (MaxPooling2) | (None, 12, 12, 32) | 0       |
| dropout_3 (Dropout)     | (None, 12, 12, 32)  | 0       |
| conv2d_6 (Conv2D)       | (None, 10, 10, 64)  | 18496   |
| conv2d_7 (Conv2D)       | (None, 8, 8, 64)    | 36928   |
| max_pooling2d_3 (MaxPooling2) | (None, 4, 4, 64)    | 0       |
| dropout_4 (Dropout)     | (None, 4, 4, 64)    | 0       |
| flatten_1 (Flatten)     | (None, 1024)        | 0       |
| input_reduction (InputReduct) | (None, 1024) | 0       |
| dense (Dense)           | (None, 512)         | 524800  |
| dropout (Dropout)       | (None, 512)         | 0       |
| dense_1 (Dense)         | (None, 10)          | 5130    |

Total params: 594,922
Trainable params: 529,930
Non-trainable params: 64,992

Figure 2. CNN model architecture dedicated for 1D series input.

| Layer (type)            | Output Shape        | Param # |
|-------------------------|---------------------|---------|
| conv1d (Conv1D)         | (None, 14, 64)      | 256     |
| conv1d_1 (Conv1D)       | (None, 12, 64)      | 12352   |
| dropout (Dropout)       | (None, 12, 64)      | 0       |
| max_pooling1d (MaxPooling1D) | (None, 6, 64) | 0       |
| flatten (Flatten)       | (None, 384)         | 0       |
| input_reduction (InputReduct) | (None, 384) | 0       |
| dense (Dense)           | (None, 100)         | 98500   |
| dense_1 (Dense)         | (None, 7)           | 707     |

Total params: 51,815
Trainable params: 39,207
Non-trainable params: 12,608
2.2. Global Sensitivity Analysis

Sensitivity analysis (SA) consists of a group of methods used for finding how the uncertainty in the model output can be assigned to the uncertainty of the model input [31]; hence, they are used to discover the connection between uncertainty of the model input and output [32]. Local Sensitivity Analysis (LSA) and Global Sensitivity Analysis (GSA) are subgroups of SA. The LSA approach alters one input parameter at a time with all others remaining constant [17], while the GSA approach modifies all input parameters concurrently. The most common approaches for evaluating the impact on the models’ output are regression methods, screening algorithms [33] and variance-based methods. The variance-based procedures used in this article are Sobol [34], Fourier Amplitude Sensitivity Test (FAST) [35] and extended Fourier Amplitude Sensitivity Test (eFAST) [36,37].

The difference between FAST and eFAST methods is that the former calculates only the first-order sensitivity, while the latter also calculates total order sensitivity. For simplicity, both these algorithms are heretofore called FAST. In this article, both first and total order sensitivities are used.

Sobol’s method is based on decomposition of output variance into a sum of input variances. It measures the impact of each individual input and the permutations between them on the output. It is achieved by calculating first-order, second-order, higher-order and total-order sensitivity indices. To calculate the indices, a Monte Carlo integration is applied.

The FAST method is derived from the time series Fourier decomposition in signal theory. The original FAST method provided only first-order indices, but extension of the method generates higher-order sensitivities. To compute the indices pattern, a search based on sinusoidal functions is applied.

3. Pruning Algorithm in Flattening Layer

This section describes the algorithm (Algorithm 1) used to prune the input of FCN. Global sensitivity analysis has not been previously applied to compress CNNs, and pruning could be its natural application. The aim of the algorithm is to provide a flattening layer pruning algorithm. This approach only reduces the weights between the convolutional and fully connected layers. The proposed procedure can be easily stacked with other pruning and decomposition compression procedures. In fact, stacking a few different procedures can lead to better compression [24]. The presented algorithm was applied to a simple CNN to validate its utility. As mentioned before, developers should not solely rely on this method as a standalone solution. GSA methods are still under research, the intent being to create an algorithm to compress all the layers of a CNN.

First the CNN has to be created and trained on the chosen dataset. The algorithm will then execute R sensitivity calculations, and each time it will prune D parameters. For each reduction cycle, pretrained CNN has to be loaded. Weights of convolutional layers must also be kept unchanged, the only part of the network subjected to training should be the FCN. What is more, all convolutional parameters have to be frozen. The next step is to join pretrained CNN and freshly initialized FCN with a reduction layer. This reduction layer is a flattening layer that has the ability to filter neurons. The large number of outputs from the convolutional layers, allied with the large number of input neurons of the FCN, results in permutations that unnecessarily consume resources. The task of the reduction layer is to prune non-impactful connections inputs. This leads to a reduction in the size of the network. The subsequent step is to calculate the sensitivity by applying one of the previously mentioned methods and aggregate it with the chosen norm. The inputs are then sorted by their sensitivity, and the least impactful ones are removed by the reduction layer. The process is repeated until reaching a given number of reduction cycles. The algorithm was implemented in Python and uses the Keras library. Computations were executed via Google’s Colaboratory service.
Algorithm 1: FCN input reduction

| Result: CNN with reduction layer |
|----------------------------------|
| load dataset                     |
| pretrain CNN                     |
| **for number of reductions** R do |
| load pretrained CNN and freeze convolutional layers |
| join convolutional layers with reduction layer and FCN |
| retrain the network              |
| calculate sensitivity             |
| aggregate sensitivity using one of norms |
| sort FCN inputs by sensitivity   |
| **for number of pruned parameters** D do |
| prune the input with the smallest sensitivity from reduction layer |
| end                              |
| end                              |

4. Results

As previously mentioned, CNNs consist of two main parts. The first embodies the convolutional and pooling layers that produce the extracted features. The second incorporates classification layers, mostly FCN. Between these two parts, a custom reduction layer is proposed. It disables the least significant parameters coming from the flatten layer. In each dataset, reduction is applied R times, with D parameters pruned each time. The convolutional part of the network is pretrained, all its parameters are non-trainable, and the FCN alone is trained from scratch each time to adjust to the reduced input. The focus of the research is to sustain or improve test accuracy. To do this, two methods with two sensitivities—Sobol (first order), Sobol total (order), FAST (first order) and FAST total (order)—and three norms—Euclidean, absolute value and maximum—are compared. In each training cycle, when sensitivity matrices are calculated for all outputs, a selected norm is applied to aggregate output sensitivity matrices. FCN input weights are then sorted by aggregated sensitivities of output. Weights with the least sensitive values are pruned, and the procedure is repeated.

4.1. Data Sets

This research is based on a total of four classification datasets (Table 1). Two of these are vectors of features, while the others contain images. The credit card fraud dataset is a Kaggle dataset [38] detecting frauds on the basis of 28 parameters. With regard to the set, confidentiality of financial data has forced application of PCA transformation. In the original dataset only 492 cases out of 284,807 transactions are marked as frauds. A new dataset had, therefore, to be created to overcome the unbalance of data. It contains 984 transactions, of which 50% are fraud cases. The dataset was randomly divided into 80% for the training set and 20% for the test set.

The Beans dataset is a 2020 UCI dataset [39] classifying dry bean grains into seven species, registered with a high-resolution camera. It contains 13,611, 16-element vectors of features. Each dataset record is composed of 12 dimension parameters and 4 shape forms. In this case, the dataset was also divided randomly into training and test sets in a ratio of 80%/20%.

MNIST and FASHION MNIST are image classification datasets made easily available through the Keras library [40]. Both are composed of 28 × 28 grayscale images with 60,000 elements in the training sets and 10,000 elements in the test sets. The differences between these datasets are the types of images found. MNIST is made up of images of digits, while FASHION MNIST consists of images of clothes and accessories. MNIST is a dataset created from the US National Institute of Standards and Technology’s Special Database 1 and Special Database 3. It gathers handwritten digits from around 250 writers, where writers for training and test sets were disjoined. FASHION MNIST is an attempt to
replace the MNIST dataset. The authors of FASHION MNIST criticized MNIST in that it is too easy to achieve high accuracy, is overused and does not represent modern computer vision tasks.

Table 1. Datasets comparison.

| Dataset            | Data Type        | Train/Test Set Size | Labels |
|--------------------|------------------|---------------------|--------|
| MNIST              | 28 × 28 greyscale images | 60 k/10 k           | 10     |
| FAHION MNIST       | 28 × 28 greyscale images | 60 k/10 k           | 10     |
| Beans              | 16 parameters    | 10,888/2723         | 7      |
| Credit Card Fraud  | 28 parameters after PCA | 786/198             | 2      |

4.2. MNIST and FASHION MNIST

MNIST and FASHION MNIST are similar datasets with different types of images. Results and conclusions for both datasets are similar and will be placed in a joined section. Figures 3 and 4 present the detailed results that are further discussed in this section. For these two datasets, the test accuracy was found to be larger than the train accuracy. This was not expected and was probably caused by high dropout percentage, where, through probability, neurons were not considered for training. In the case of the CNN used for these datasets, its flatten layer output size was 1024, and a total number of 20 reduction cycles was performed. In each cycle, 50 least impactful parameters were pruned. The original total number of parameters was 594,922, of which 529,930 were considered trainable. At the last step, the number of trainable parameters was decreased to 43,530. This is an over ten times reduction. The cost for so high a reduction in size is a decrease of 10% for test accuracy. A gentle decline in accuracy is observed until around 600 pruned neurons, and further reduction resulted in a more radical decrease. All methods and norms presented very competitive results.

Figure 3. FCN reduction for the MNIST dataset by methods and norms on train and test sets.
Figure 4. FCN reduction for the Fashion MNIST dataset training and test sets, by methods and norms.

4.3. Credit Card Fraud

The credit card fraud data are a post-PCA transformation. This suggests that there is no large potential for further pruning. In our experiment, the reduction was performed 17 times, each time reducing 30 parameters. The original FCN inputs had 768 parameters and were reduced to 288 parameters at the last run. This led to a drop of trainable parameters from 77,102 to 29,102, while the total number of parameters dropped from 89,710 to 41,710. We found that in the PCA transformation, reduction had little influence on accuracy. For both methods, Sobol and FAST, the test accuracy did not change or was improved at some point in each scenario. As seen in Figure 5, the Sobol scenario test accuracy results are more flat, while in the case of FAST, they slightly decrease. In most of the cases, the maximum cost for reducing the size of the network by half is up to 2 percentage points of test accuracy drop.

4.4. Beans

This set was reduced 15 times with 20 pruned parameters at each run. The original number of FCN input parameters was 384 and was reduced to 104. This resulted in a decrease in total network parameters from 51,815 to 23,815, while FCN parameter accuracy fell to 50%. In the case of the first-order Sobol method, abs and euc norms test accuracy reached a peak training accuracy. Similar results are observed when the Sobol total method was applied. Here, test accuracy, when max norm was applied, reached the level of training accuracy after a cycle of 100 reductions. In contrast, Abs test accuracy approached training accuracy at the end of a reduction after a high drop. Test accuracy reached or exceeded train accuracy in all cases when FAST and FAST total methods were applied. As seen in Figure 6, abs and euc norms test accuracy reached training accuracy after just 50 neurons were reduced, with accuracy maintained to the end of the run. Max norm was found to outperform other norms, but only when a large number of FCN inputs were reduced. Finally, abs norms preserved the highest test accuracy. This was only a few percentage points lower than the training accuracy of the original non-reduced network. What is
more, Euc norms presented similar results for both FAST and FAST total scenarios, while abs norm significantly reduced test accuracy for FAST total, when large reductions are taken into account. For FAST total, max norm had the highest test accuracy, outperforming training accuracy in the last reduction runs.

Figure 5. FCN reduction for the Credit Card Fraud dataset training and test set, by methods and norms.

Figure 6. FCN reduction for the Bean dataset training and test set, by methods and norms.
4.5. Discussion of the Results

Sobol and Fashion MNIST sets sustained stable, low-accuracy drops until half the parameters were reduced. Subsequently, the accuracy drop reached 10% with 90% reduction in FCL neurons. The fraud dataset showed that reduction of non-meaningful data does not have to impact the accuracy. In that case, accuracy fluctuated around a constant value. The Beans dataset is the most surprising. Here, test accuracy largely exceeded train accuracy. We also noted that GSA was proven not only to be able to reduce the impact, but also to keep high accuracy and, indeed, to improve test accuracy. Table 3 presents post-reduction numbers of input FCN parameters, total number of parameters and number of parameters for CNN and FCN, similarly to Table 2.

On comparing Tables 2 and 3, in extreme cases, it can be seen that the size of the network was reduced from 54% for the Credit Card Fraud and Bean datasets to up to 82% in the case of MNIST and Fashion MNIST datasets.

Table 2. CNN parameters pre-reduction.

| Datasets               | Input Size | Total   | Non-Trainable | Trainable |
|------------------------|------------|---------|---------------|-----------|
| MINST/Fashion MNIST    | 1024       | 594,922 | 64,992        | 529,930   |
| Credit Card Fraud      | 768        | 89,710  | 12,608        | 77,102    |
| Beans                  | 384        | 51,815  | 12,608        | 39,207    |

Table 3. CNN parameters post-reduction.

| Datasets               | Input Size | Total   | Non-Trainable | Trainable |
|------------------------|------------|---------|---------------|-----------|
| MINST/Fashion MNIST    | 74         | 108,522 | 64,992        | 43,530    |
| Credit Card Fraud      | 288        | 41,710  | 12,608        | 29,102    |
| Beans                  | 104        | 23,815  | 12,608        | 11,207    |

The proposed procedure was expected to prune flattening layer connections and minimize accuracy loss with pretrained convolutional layers. This was observed in the case of MNIST and FASHION MNIST datasets. Utilization of the Credit card fraud dataset presented surprising results. Besides a general training accuracy drop with deleted connections, some norm functions were able to improve the accuracy of the applied test sets. The same, yet more clear, phenomenon was observed in the results of Beans classification. In all cases, besides the Sobol total order indices method, a Euclidean norm function boosted the test set accuracy to the level of the training set accuracy. In this scenario, the presented GSA-based pruning algorithm not only decreased network size, but also vastly improved test set accuracy of the network. The algorithm pruned only the flattening layer. In further work, GSA methods are going to be applied to other layers of the CNN to create a pruning method for the whole structure. We hope to create a pruning method that is able not only to compress the structure, but also to increase its performance.

5. Conclusions

This article applied the GSA methods of Sobol and FAST to reduce the number of FCN input neurons in CNNs. Originally, the full number of connections between matrices of convoluted features and FCN led to a large number of training parameters. When we applied the described reduction algorithms that are based on GSA methods and three norms, we were able to cut-down the number of unnecessary parameters while keeping near to the original accuracy levels. For some datasets, the proposed pruning provided accuracy levels close to the original solution.
The reduction in the structure of the internal neural network has a very positive effect on several aspects. The first is faster computation time. This applies both to the current time related to the recovery mode and also to learning time, as neural networks often need to be retrained when new data come in, and the greater volume of data, the more time is needed in doing so. The second aspect discussed here is the issue of neuronal structure overfitting. In practice, various types of treatments are often used to address the problem of overfitting, such as data reorganization, drop-out procedure and the use of a special penalty function during training. However, the method proposed in the text of the article solves the problem naturally because removing redundant neurons implies the existence of fewer points of freedom and, therefore, the likelihood of overfitting being minimized.

For datasets other than that originally used, accuracy was decreased, but it was disproportionately less when compared to the size of the reduction. As the algorithm is able to significantly reduce the size of the network with a cost of small performance drop, it can, therefore, enable the use of previously large networks on devices with less memory, such as mobile systems. The smaller number of parameters also directly relates to improvement in the network’s prediction time.

The proposed procedure can be applied as either a supervised or unsupervised algorithm. In the first case, it is necessary to leave the validation sample on the basis of which the quality of the network computation can be checked in each iteration of neuron removal. Of course, this is related to more computation time. In the case of treating the reduction procedure as unsupervised, we can assume in advance the number of neurons we want to remove.

The above proposal of the neural network reduction algorithm also addresses research related to the analysis of the significance of individual components of CNN. This is especially true because, in most cases, neural networks are treated as black-box models, their internal elements being not subject to analysis. What is more, during the neural network system synthesis, one does not try to understand the meaning of individual elements but bases the application exclusively on empirical performance.

Further research plans will be related to understanding and assessing the use of reduction methods inside CNN neural networks, in particular, in the layers of the fully connected part. This action will be aimed at reducing the topological structure of the neural network and, as a consequence, slashing computation time and enhancing the efficiency of neural computations.

**Author Contributions:** Conceptualization, P.A.K. and E.J.; software, E.J.; validation, P.A.K.; formal analysis, P.A.K. and E.J.; investigation, E.J.; resources, E.J.; data curation, E.J.; Writing—Original draft preparation, P.A.K. and E.J.; Writing—Review and editing, P.A.K.; visualization, E.J.; supervision, P.A.K.; project administration, P.A.K. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research is partially financed by Grants for Statutory Activity from Faculty of Physics and Applied Computer Science of the AGH University of Science and Technology in Cracow.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** MNIST/ Fashion MNIST [https://keras.io/api/datasets/](https://keras.io/api/datasets/) (accessed on 23 January 2021); Credit Card Fraud [https://www.kaggle.com/mlg-ubb/creditcardfraud](https://www.kaggle.com/mlg-ubb/creditcardfraud) (accessed on 23 January 2021); Beans [https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset](https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset) (accessed on 23 January 2021).

**Acknowledgments:** This article is based upon work from COST Action 17124 DigForAsp, supported by COST (European Cooperation in Science and Technology), [www.cost.eu](http://www.cost.eu) and is partially financed by Grants for Statutory Activity from Faculty of Physics and Applied Computer Science of the AGH University of Science and Technology in Cracow. The authors would also like to thank the anonymous referees for their careful reading of the paper and their contribution of useful suggestions that helped to improve this article.
Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

- CI: Computational Intelligence
- ANN: Artificial Neural Networks
- CNN: Convolutional Neural Network
- FNN: Fully Connected Network
- FCL: Fully Connected Layer
- SA: Sensitivity Analysis
- GSA: Global Sensitivity Analysis
- LSA: Local Sensitivity Analysis
- ReLU: Rectified Linear Unit

References

1. Rutkowski, L. Computational. In Intelligence Methods and, Techniques; Springer: Berlin/Heidelberg, Germany, 2008.
2. Dhillon, A.; Verma, G.K. Convolutional neural network: A review of models, methodologies and applications to object detection. Prog. Artif. Intell. 2020, 9, 85–112. [CrossRef]
3. De Wilde, P. Neural Network Models: Theory and Projects; Springer Science & Business Media: London, UK, 2013.
4. Haykin, S. Neural Networks: A comprehensive foundation. Neural Netw. 2004, 2, 41.
5. Kowalski, P.A.; Sapala, K.; Warchałowski, W. PM10 forecasting through applying convolution neural network techniques. Air Pollut. Stud. 2020, 3, 31–43. [CrossRef]
6. Balcerek, J.; Pawlowski, P.; Dąbrowski, A. Classification of emergency phone conversations with artificial neural network. In Proceedings of the 2017 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA), Poznan, Poland, 20–22 September 2017; pp. 343–348.
7. Kiddon, C.; Zettlemoyer, L.; Choi, Y. Globally coherent text generation with neural checklist models. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Austin, TX, USA, 1–5 November 2016, pp. 329–339.
8. Zhang, J.; Zong, C. Deep Neural Networks in Machine Translation: An Overview. IEEE Intell. Syst. 2015, 30, 16–25. [CrossRef]
9. Roberts, C.W. Text Analysis for the Social Sciences: Methods for Drawing Statistical Inferences from Texts and Transcripts; Routledge: New York, NY, USA, 2020.
10. Wu, C.H. Behavior-based spam detection using a hybrid method of rule-based techniques and neural networks. Expert Syst. Appl. 2009, 36, 4321–4330. [CrossRef]
11. Gazeau, V.; Varol, C. Automatic spoken language recognition with neural networks. Int. J. Inf. Technol. Comput. Sci. 2018, 10, 11–17. [CrossRef]
12. Pierson, H.; Gashler, M. Deep Learning in Robotics: A Review of Recent Research. Adv. Robot. 2017, 31, 821–835. [CrossRef]
13. Fukushima, K.; Miyake, S. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In Competition and Cooperation in Neural Nets; Springer: Berlin/Heidelberg, Germany, 1982; pp. 267–285.
14. Bengio, Y.; Goodfellow, I.; Courville, A. Deep learning; MIT Press: Cambridge, MA, USA, 2017; Volume 1.
15. Li, H.; Kadav, A.; Duradanovic, I.; Samet, H.; Graf, H. Pruning Filters for Efficient ConvNets. arXiv 2016, arXiv:1608.08710.
16. Molchanov, P.; Tyree, S.; Karras, T.; Aila, T.; Kautz, J. Pruning Convolutional Neural Networks for Resource Efficient Transfer Learning. arXiv 2016, arXiv:1611.06440.
17. Kowalski, P.A.; Kusy, M. Sensitivity analysis for probabilistic neural network structure reduction. IEEE Trans. Neural Netw. Learn. Syst. 2017, 29, 1919–1932. [CrossRef] [PubMed]
18. Harshman, R.A. Foundations of the PARAFAC procedure: Models and conditions for an “explanatory” multi-modal factor analysis. UCLA Work. Pap. Phon. 1970, 16, 1–84.
19. Zhou, M.; Liu, Y.; Long, Z.; Chen, L.; Zhu, C. Tensor rank learning in CP decomposition via convolutional neural network. Signal Process. Image Commun. 2019, 73, 12–21. [CrossRef]
20. Goyal, S.; Roy Choudhury, A.; Sharma, V. Compression of Deep Neural Networks by Combining Pruning and Low Rank Decomposition. In Proceedings of the 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Rio de Janeiro, Brazil, 20–24 May 2019; pp. 952–958. [CrossRef]
25. Ren, H.; Xu, B.; Wang, Y.; Yi, C.; Huang, C.; Kou, X.; Xing, T.; Yang, M.; Tong, J.; Zhang, Q. Time-Series Anomaly Detection Service at Microsoft. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining; Association for Computing Machinery, New York, NY, USA, 4–8 August 2019; pp. 3009–3017. [CrossRef]

26. Kalchbrenner, N.; Grefenstette, E.; Blunsom, P. A Convolutional Neural Network for Modelling Sentences. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, Baltimore, MD, USA, 22–27 June 2014; Volume 1. [CrossRef]

27. Kim, Y. Convolutional Neural Networks for Sentence Classification. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, Doha, Qatar, 25–29 October 2014. [CrossRef]

28. Vedaldi, A.; Zisserman, A. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. arXiv 2013, arXiv:1312.6034.

29. Renz, K.; Stache, N.; Albanie, S.; Varol, G. Sign language segmentation with temporal convolutional networks. In Proceedings of the ICASSP 2021–2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Toronto, ON, Canada, 6–11 June 2021.

30. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 1–9. [CrossRef]

31. Saltelli, A.; Tarantola, S.; Campolongo, F.; Ratto, M. Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models; Wiley: New York, NY, USA, 2004. [CrossRef]

32. Saltelli, A.; Ratto, M.; Andres, T.; Campolongo, F.; Cariboni, J.; Gatelli, D.; Saisana, M.; Tarantola, S. Global Sensitivity Analysis: The Primer; John Wiley & Sons: Hoboken, NJ, USA, 2008; Volume 304. [CrossRef]

33. Morris, M.D. Factorial Sampling Plans for Preliminary Computational Experiments. Technometrics 1991, 33, 161–174. [CrossRef]

34. Saltelli, A.; Annoni, P.; Azzini, I.; Campolongo, F.; Ratto, M.; Tarantola, S. Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index. Comput. Phys. Commun. 2010, 181, 259–270. [CrossRef]

35. Saltelli, A.; Bolado, R. An alternative way to compute Fourier amplitude sensitivity test (FAST). Comput. Stat. Data Anal. 1998, 26, 445–460. [CrossRef]

36. Lauret, P.; Fock, E.; Mara, T. A Node Pruning Algorithm Based on a Fourier Amplitude Sensitivity Test Method. IEEE Trans. Neural Netw. 2006, 17, 273–293. [CrossRef]

37. Kaggle Datasets. Available online: https://www.kaggle.com/ (accessed on 23 January 2021).

38. Dua, D.; Graff, C. UCI Machine Learning Repository. 2017. Available online: http://archive.ics.uci.edu/ml/index.php (accessed on 23 January 2021).

39. Keras Datasets. Available online: https://keras.io/api/datasets/ (accessed on 23 January 2021).