IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 8, 2019, accepted June 15, 2019, date of publication June 20, 2019, date of current version July 9, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2924060

Sparse Matrix Classification on Imbalanced Datasets Using Convolutional Neural Networks

JUAN C. PICHEL[1], **(Member, IEEE), AND BEATRIZ PATEIRO-LÓPEZ**[2]
[1]CiTIUS, Universidade de Santiago de Compostela, 15782 Santiago de Compostela, Spain
[2]Dpto. de Estadística, Análisis Matemático y Optimización, Universidade de Santiago de Compostela, 15782 Santiago de Compostela, Spain

Corresponding author: Juan C. Pichel (juancarlos.pichel@usc.es)

This work was supported in part by the Ministerio de Economía y Competitividad (MINECO) under Grant MTM2016-76969-P and Grant RTI2018-093336-B-C21, in part by the Xunta de Galicia under Grant ED431G/08 and Grant ED431C 2018/19, and in part by the European Regional Development Fund (ERDF).

ABSTRACT This paper deals with the class imbalance problem in the context of the automatic selection of the best storage format for a sparse matrix with the aim of maximizing the performance of the sparse matrix vector multiplication (SpMV) on GPUs. Our classification method uses convolutional neural networks (CNNs) and proposes several solutions to mitigate the bias toward the majority classes when the data are not balanced. First, the CNNs are trained using images that represent the sparsity pattern of the matrices, whose pixels are colored according to different matrix features. In addition, we introduce a new network called SpNet, which achieves better results than a standard network as AlexNet in terms of prediction accuracy even having a more simple architecture. Finally, sampling techniques and cost-sensitive methods have been studied to give more emphasis on minority classes. The experiments conducted show that our classifiers are able to select the best performing format 92.8% of the time, obtaining 98.3% of the maximum attainable SpMV performance. A comparison to other state-of-the-art classification methods is also provided, demonstrating the benefits of our proposal.

INDEX TERMS Sparse matrix, classification, imbalance, deep learning, CNN, performance.

I. INTRODUCTION

Sparse matrix-vector multiplication (SpMV) is considered one of the most important computational kernels lying at the heart of many scientific and engineering applications. There is consensus when affirming that SpMV performs poorly on multicore and manycore architectures since it is a memory-bound operation. As a consequence, the research community has devoted numerous efforts to develop efficient and optimized implementations. Given that the SpMV performance depends on both the target parallel system and the sparsity structure of the matrix, many existent storage formats have focused on a particular application domain, sparsity pattern and/or computer architecture. The compressed sparse row (CSR) format is the most popular data structure to store a sparse matrix for multicore systems, while for GPUs we cannot find a prevailing format. In that case, the multiple goals of maximizing coalesced global memory access, reducing thread divergence, and increasing warp occupancy often conflict with each other [1]. We must take into account that using an inappropriate format could lead to an important degradation in the SpMV performance. As a consequence, the automatic selection of the best performing format is an important and challenging classification task.

On the other hand, Convolutional Neural Networks (CNNs) are the most important deep learning networks for image recognition and classification. The main difference between CNNs and traditional machine learning approaches is that they integrate automatic feature extraction and classification in one model. CNNs have also demonstrated promising results with other kinds of applications such as Natural Language Processing (NLP), speech and audio processing, to name a few. However, dealing with classification problems related to High Performance Computing (HPC) using CNNs is an incipient field [2]–[4].

Many times CNNs have to learn from imbalanced data in such a way that some classes have a significantly higher number of examples in the training set than others. This phenomenon is termed as the *class imbalance* problem. Most existing learning algorithms produce inductive bias

The associate editor coordinating the review of this manuscript and approving it for publication was Antonio J. Plaza.

towards the majority classes if training data are not balanced, resulting in poor minority class recognition performance. However, in many application domains the correct classification of a minority class sample is equally important to the correct classification of a majority class sample. For instance, in the selection of the best performing format for the SpMV kernel, where a format (class) does not prevail over the others. Existing methods for addressing class imbalance work at two different levels [5]: data-level and algorithm-level. In the first case, techniques aim to balance the class distribution in the dataset in order to make standard training algorithms work. So these techniques change the original dataset. A different approach is followed by algorithm-level methods, which modify existing techniques to give more emphasis on the minority classes without changing the dataset.

In this paper we address the automatic classification of sparse matrices to select the best SpMV performing storage format on GPUs using CNNs. It is important to note that the problem to deal with is highly imbalanced, with a sample size ratio between the smallest and largest classes of 1:160. Six different storage formats have been considered. Our method generates images from the sparsity pattern of the matrices. Each pixel in those images represent a submatrix, whose RGB color is used to code some property or feature of the corresponding matrix. In this way, the produced datasets can successfully train a CNN. This approach has been successfully applied to a more simple classification task with only three classes and a balanced dataset [3]. The main contributions of this work are the following:

- To the best of our knowledge, this paper is the first to deal with the class imbalance problem in the context of the automatic selection of the best storage format for sparse matrices. Sampling techniques and cost-sensitive methods have been studied to specifically overcome the issues related to training classifiers using imbalanced data.
- To validate our proposal we have generated a dataset containing more than 10,000 sparse matrices. In this way, matrices represent a wide range of features and sparsity patterns. This dataset is based on matrices included in the SuiteSparse matrix collection [6], which come from a variety of real applications.
- Although our methodology is able to generate images that can successfully train a standard network such as AlexNet [7], we go a step further introducing SpNet, a new simplified network architecture which is able to beat AlexNet in terms of classification and SpMV performance.
- The new method is evaluated considering two different GPUs. It selects the best storage format 92.8% of the time, obtaining 98.3% of the highest available SpMV performance.
- A comparison with several state-of-the-art classification methods is provided, showing the benefits of our proposal.

The remainder of this paper is organized as follows. Section II gives the background and discusses some related research. Section III summarizes our classification methodology. The experimental setup is explained in Section IV. Performance results are shown and discussed in Section V. Finally, the main conclusions derived from the work are explained.

II. BACKGROUND & RELATED WORK

A. SPARSE MATRIX FORMATS

There is not a general storage format that is adequate for all kind of sparse matrices since their shape, number and distribution of nonzeros, depends on the application domain from which they come. Over the past few decades, many storage formats has been proposed (the interested reader can find a survey in [8]). These formats differ in the storage requirements, the accessing methods and how well they adapt to different application domains or hardware platforms. Some of them are only well suited for matrices with a particular structure. For instance, diagonal matrices (DIA) or matrices containing small dense block sub-structures (BELLPACK [9]). In this paper, we selected storage formats that are appropriate for matrices coming from different real problems but also efficient for sparse matrix computations. In particular, we have considered the compressed row storage (CSR), ELLPACK (ELL), hybrid (HYB) [10] and blocked compressed sparse row (BSR) formats, which are implemented in the NVIDIA cuSPARSE[1] library (see Figure 1):

- Compressed Sparse Row (CSR): It is a popular and general-purpose sparse matrix representation. The matrix is stored using three arrays: the first one stores the nonzero values, the second stores the corresponding column indices, and the last stores the pointers to the beginning of every row.
- ELLPACK (ELL): For a $n \times m$ matrix with a maximum of k nonzeros per row, this format stores the nonzeros in a dense $n \times k$ array, with another same dimensional array to store the column index of every element. Rows whose number of nonzeros are fewer than k are padded with zeros. ELL is efficient if k is not substantially different from the average number of nonzeros per row.
- Hybrid (HYB): This format combines the computation efficiency of ELL with the simplicity and generality of COO (that stores row and column indices explicitly). Most of the nonzeros are stored in ELL format, while rows with a considerably different number of entries are stored in COO format.
- Blocked Compressed Sparse Row (BSR): It can be considered a blocked version of the CSR format. Instead of storing the nonzeros independently, dense submatrices (blocks) of fixed shape are used instead. Three arrays are required to store the entries of the matrix in BSR. The first array stores the column indices of the first elements

[1] https://developer.nvidia.com/cusparse

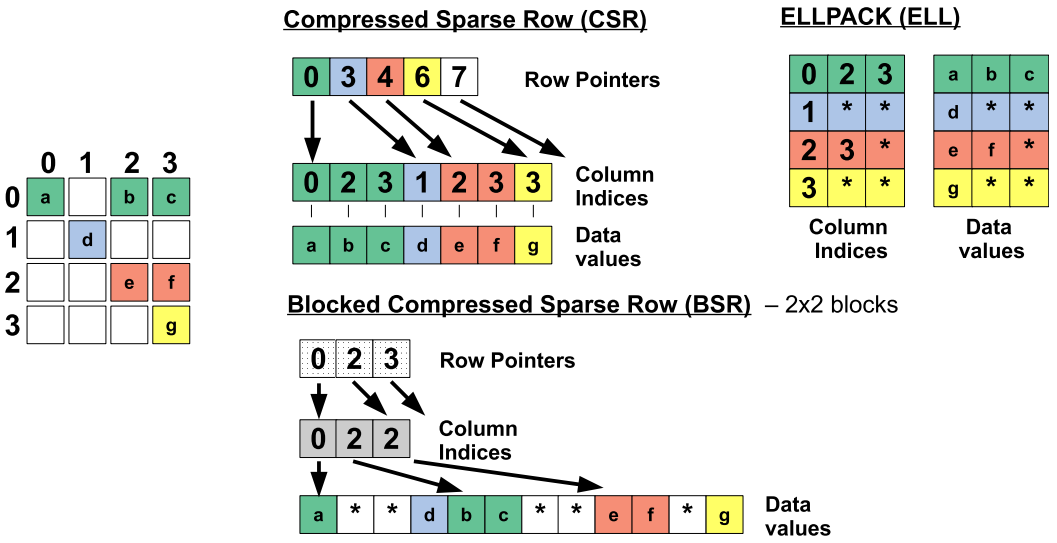

FIGURE 1. An example of CSR, ELL and BSR sparse matrix storage formats.

of all blocks. The row pointers array points to the beginning of each block row in the columns array. Finally, the elements of each block are stored contiguously in the values array.

In addition, we have included a new format called CSR5 [11] which shows a good behavior in terms of performance for both regular and irregular matrices on various hardware platforms such as multicores, GPUs and Intel Xeon Phi. This format extends the CSR format, leaving unchanged one of the three arrays of CSR. It stores the other two arrays in an in-place tile-transposed order, and adds two extra data structures containing auxiliary information.

B. CONVOLUTIONAL NEURAL NETWORKS

CNNs are a type of neural networks. Their characteristics make them especially suitable for image classification. A CNN passes the input image through a series of layers that progressively learn low-level and high-level features of the image, these last used for classification. Layers in CNNs can be classified into three categories: input layers, feature-extraction layers and classification layers.

Input layers load and store the input data (raw image) to be processed in the network. This input data specifies several characteristics of the image: width, height, and number of channels. Feature-extraction layers perform several consecutive operations: convolution, nonlinear activation and pooling. The main goal of a convolution layer is to extract features from the input image. We can think of convolution as sliding a small window (filter) across the input image and, at each location (receptive field), computing the element-wise matrix multiplication between the weights of the filter and the receptive field and then summing up all values. As a result, we obtain a 2D activation map; the filter activates a specific type of visual feature from the image, such as edges, curves, etc. Several filters can be used in the same convolution layer, generating multiple activation maps. It is common to apply a nonlinear activation function after every convolution operation. The ReLU (Rectified Linear Unit) is the most widely used activation function in neural networks. It replaces all negative pixel values in the feature map by zero. In order to reduce the dimensionality of feature maps, it is also usual to insert a pooling layer between successive convolution+ReLU layers. The pooling operation decreases the amount of parameters and computation in the network while retaining the most important information. Finally, the classification layers consist of one or more fully-connected layers at the end of the architecture to produce class scores. They use the high-level features generated by the convolutional and pooling layers for estimating the probability of the image belonging to each class.

Many CNN architectures have been proposed, some of the most popular are LeNet [12], AlexNet [7], GoogLeNet [13], VGGNet [14] and ResNet [15].

During the training process, the CNN automatically adjusts the network parameters. First, all the filter weights and parameters are initialized (a common practice is random initialization). Then, the network takes a training input image (labeled input image) and forward propagates it through the network along the feature-extraction layers and fully-connected layers, obtaining a prediction (probability of the image belonging to each class). A prediction error is calculated through a loss function that compares the output of the network and the correct output. The training process, by means of back propagation, adjusts and updates the network parameters iteratively to minimize the overall error on each training input. The entire image dataset is passed forward and backward for a given number of times (number of epochs). Note that the number of epochs, as well as other hyperparameters such as the learning rate, need also to be tuned during training to make networks train better and faster.

Other parameters such as number of filters, filter sizes, architecture of the network, etc., do not change during the training process.

C. METHODS FOR ADDRESSING IMBALANCE

A common issue in deep learning applications is that some classes have a significantly higher number of examples in the training set than other classes. This is known as the *class imbalance* problem. Methods for addressing imbalance for classical machine learning models have been studied for many years [5], [16], while recent works are mainly focused on CNNs [17]. Those methods can be divided into two main categories: sampling techniques and cost-sensitive methods. Hybrid methods try to combine those two categories.

Sampling operates on the data itself with the aim of providing a balanced class distribution to make standard training algorithms work. The most important methods belonging to this category are oversampling and undersampling. The basic version of oversampling simply replicates randomly selected samples from minority classes, adding them into the original dataset. It has been demonstrated that oversampling is effective, but it can lead to overfitting [18]. This method has been successfully applied in the context of deep learning [19], [20]. On the contrary, undersampling methods randomly remove a certain number of instances from the majority class to achieve a balanced dataset [5]. An important issue of this method is that it may lose some important information when discarding a portion of the available data.

A different approach to deal with the class imbalance problem is cost-sensitive learning. Instead of creating balanced datasets through different sampling strategies, cost-sensitive learning addresses the imbalanced learning problem by using different cost matrices that describe the costs for misclassifying any particular data example. For instance, applying a higher penalty for minority class samples [21]. Another cost-sensitive approaches propose new weighted loss functions to give more emphasis on the minority classes [21]–[24]. Unlike the previous methods, other solutions deal with imbalance moving the cost-sensitivity to the inference phase [25], [26].

D. SPARSE MATRIX CLASSIFICATION

Many works deal with the identification of the optimal storage format for sparse matrices on GPUs using analytical models [27]–[29]. Models tend to show a good accuracy but they are usually evaluated on small datasets. Other authors opt for using traditional machine learning approaches, but only few of them consider GPUs as target systems. For example, one approach consists in building a decision tree based on several matrix features with the aim of choosing the best performing storage format [1]. Authors report a global accuracy up to 84% with 95% of the maximum achievable SpMV performance. In other work the classification task is addressed using support vector machines [30]. Their classifiers demonstrate an accuracy in the interval 73-88.5%, improving the average SpMV performance to a maximum of 98%. It is worth noting that our proposal outperforms both works in terms of accuracy and performance even they are facing a classification problem with a smaller number of classes. On the other hand, best results in both works were obtained using information from more than three matrix features to train the classifiers. More recently, researchers deal with the sparse matrix classification problem using convolutional neural networks [4]. In that work several matrix representations were studied to train the networks. Their findings point out that the best solution is a histogram capable of capturing the spatial distribution of the nonzeros in the matrix. This representation leads to the creation of an *ad-hoc* CNN architecture. Unlike our method, they do not take advantage of the color channels of the images to code relevant information about the considered matrix. We must highlight that none of the works commented above have addressed a class imbalance problem.

Other authors apply machine learning techniques considering only multicore processors as target platform [31]. Finally, a deep learning based mechanism to choose the most efficient SpMV code implementation for both CPUs and GPUs was introduced in [2]. Although it is a conceptually interesting approach, their performance results are not competitive with some of the state-of-the-art methodologies mentioned above.

III. CLASSIFICATION METHODOLOGY

This section summarizes the methodology to select the best performing format for a particular sparse matrix with the aim of maximizing its SpMV performance [3]. Figure 2 shows an scheme with the different stages of our approach. We assume that a large set of sparse matrices coming from different application domains and representing a variety of sparsity patterns is available. This dataset is the input of the following phases: SpMV benchmarking and image generation. The goal of the first stage is to evaluate for all the matrices in the dataset the performance of the SpMV kernel considering different storage formats. The outcome is the best format in terms of performance for each matrix. That format associates a label (class) to each matrix in the dataset, which will be used later as ground truth in the CNN training phase. Therefore, there are as many classes as storage formats. It is worth mentioning that we have considered GPUs as target platforms to build the ground truth information, but our methodology is completely agnostic with respect to the underlying parallel architecture. Shifting to a different system only means to execute the benchmarking phase on the corresponding target platform in order to get the new ground truth information.

Building the image dataset from the input sparse matrices can be considered the most important phase of our method. In a naive approach, the sparsity pattern of a $n \times m$ matrix can be viewed as a $n \times m$ black and white image, where white pixels correspond to nonzero elements and black pixels represent zeros. In any case, this simple technique is not enough to build a valid image dataset since CNNs require input images of a fixed size. As a consequence, matrices should be scaled down to the same size. The next procedure explains how

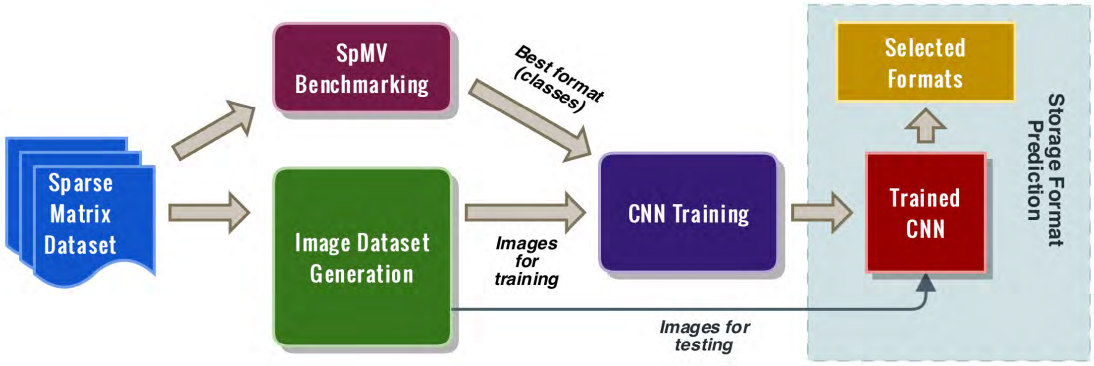

FIGURE 2. Different stages of the classification methodology.

to do it. Let's assume that, for simplicity, a square $n \times n$ matrix should be scaled to a $p \times p$ matrix, being $n > p$. The original matrix is split into $p \times p$ submatrices. To build the new $p \times p$ scaled matrix, we insert a nonzero at position (i, j) if there is, at least, one nonzero value in the corresponding (i, j) submatrix. Empty submatrices are represented in the scaled matrix as zero values. Figure 3(a) illustrates this procedure showing a 113×113 black and white image generated from a $10,848 \times 10,848$ sparse matrix.

The previous method allows to generate easily a binary image dataset that fits a CNN. However, we demonstrated that datasets of that type do not provide satisfactory classification results [3]. We must take into account that scaling down a sparse matrix simplifies the appearance of its sparsity pattern, causing a loss in the information provided to the CNN in the training phase. A single pixel in the image represents a submatrix in the original matrix. For instance, one pixel in Figure 3(a) corresponds to a 96×96 submatrix (that is, $^{10,848}/_{113}$).

To address that issue it is necessary to provide additional information to the CNN with the aim of improving the learning process. With this goal in mind, we will take advantage of the RGB channels of the image to code information related to some features of the original sparse matrix. In particular, we have considered the following global metrics about the matrices (numbers are used as identifier of the metric):

(0) Matrix size (n): number of rows and columns of the matrix.
(1) Average number of nonzeros per row of the matrix (nnz_{row}).
(2) Standard deviation of the number of nonzeros per row of the matrix (σ_{row}).
(3) Matrix density (ρ): calculated as the ratio between the number of nonzeros and the number of rows multiplied by the number of columns.
(4) Maximum number of nonzeros in a row of the matrix (max_{row}).

In our implementation pixels corresponding to empty submatrices are always black, that is, their RGB color is $(0, 0, 0)$. Only those pixels representing non-empty submatrices have a different associated RGB color. The color of these pixels is always the same, whose value for each RGB channel is within the interval $[1, 255]$. Metrics should be normalized to fit that interval. Note that it is possible to use one, two or three color channels to include the matrix information. When a channel is not used, its value for all the pixels in the image is 0.

We refer to $R_xG_yB_z$ to specify that metrics x, y and z were used to calculate the pixel values of the red, green and blue channels, respectively. There exist multiple combinations of number of channels and metrics that can be utilized in the image dataset generation phase. In this work we only focus on those most relevant in terms of performance. In particular, datasets were generated using the following configurations: $R_1G_3B_4$ and $R_0G_1B_4$. Figure 3 contains several images generated from the same input matrix using different metrics to color the pixels. Figures 3(b), 3(c) and 3(d) use only one color channel to code the average number of nonzeros per row (red), matrix density (green) and maximum number of nonzeros in a row of the matrix (blue), respectively. Figure 3(e) combines the three color channels of the previous pictures into a single image. We must highlight that the assignment of metrics to channels do not affect the results of the CNN training phase. It means that is irrelevant to consider, for instance, $R_1G_2B_3$ or $R_3G_1B_2$.

Following the scheme of Figure 2, the next stage in our method involves the training of the network. To do so, it is necessary to feed the CNN with a set of images labeled according to the best performing storage format (class of the matrix). This data was generated in the previous phases. In particular, labels come from benchmarking the different SpMV kernels, while the image dataset is the outcome of the image generation stage. Note that the image dataset is divided into training and test sets. In this way, the training process only takes into consideration those images belonging to the training set. Images in the test set are necessary to assess the prediction accuracy of the final chosen classification model. We have used a *cross-validation* method, which is generally considered the best method both for model selection and assessment. In particular, we have opted for a *k-fold cross-validation*. This method is used when some hyperparameter

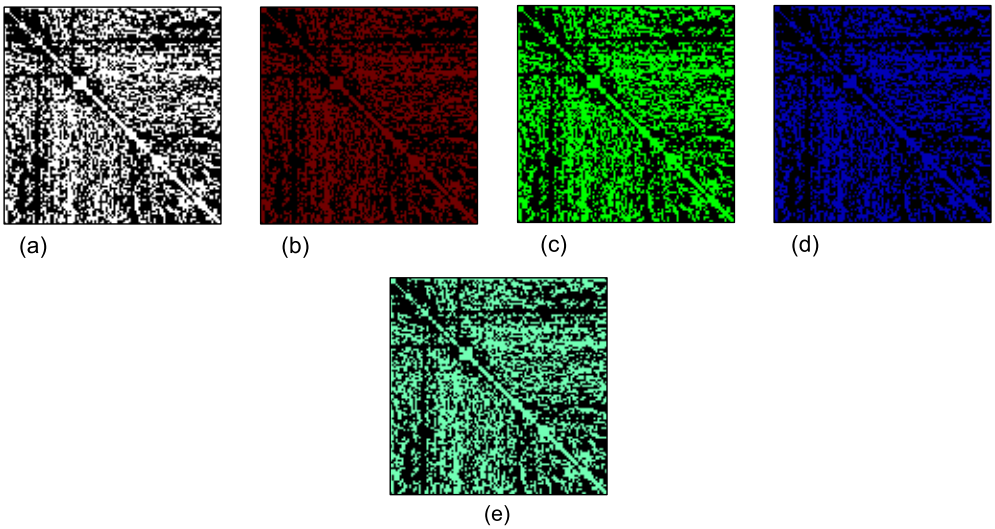

FIGURE 3. Images of 113 × 113 pixels generated from a 10, 848 × 10, 848 sparse matrix: (a) pattern/binary, (b) R_1, (c) G_3, (d) B_4 and (e) $R_1G_3B_4$.

of the network have to be estimated. In our case the hyperparameter of interest is the optimal number of training epochs. This validation method divides the training set into k folds. The first fold is kept for testing (known as *validation set*) and the model is trained on the remainder $k-1$ folds. This process is repeated k times in such a way that each time a different fold is used for validation. After each epoch, the global accuracy on the corresponding validation set is recorded. Afterwards the average validation set accuracy is computed (across the k folds) for each number of epochs. The chosen number of epochs will be the one that maximizes this value. Finally, the network is trained using as input the complete training set until the optimal number of iterations is reached.

The resulting trained CNN will be the one used for carrying out the storage format prediction. Images in the test set, which were not used in the training process, are utilized to validate the accuracy of the classifier.

IV. EXPERIMENTAL SETUP

A. HARDWARE PLATFORMS AND SOFTWARE

Since there is not a prevailing storage format for sparse matrices on GPUs, we have considered those systems as target platform to evaluate our proposal. Nevertheless, our classification methodology could be easily applied to other parallel architectures such as multicore CPUs or accelerators like the Intel Xeon Phi. The most important features of the NVIDIA GPUs used in the tests are shown in Table 1. From now on, we will use TITANX and QUADRO to refer to both GPU models.

The benchmarking phase was carried out using the SpMV kernels included in the NVIDIA cuSPARSE library (CUDA toolkit v8). In particular, we have considered CSR, HYB, ELL, BSR and COO storage formats. In addition, we have included results for the CUDA implementation of CSR5 format [11]. In this way, we experimented with 6 storage formats.

TABLE 1. Main characteristics of the NVIDIA GPUs used in the tests.

Model	TITAN X	Quadro P6000
Architecture	Pascal	Pascal
CUDA capability	6.1	6.1
Multiprocessors (MP)	28	30
CUDA Cores	3,584	3,840
GPU Max Clock rate (GHz)	1.53	1.64
Global memory (MBytes)	12,190	24,450
Memory Bandwidth (GBytes/s)	480.4	432.8
L2 Cache Size (MBytes)	3	3
Max. SP performance (GFLOPS)	10,974	12,634
Max. DP performance (GFLOPS)	342.9	394.8

The training phase was performed using the most powerful GPU (QUADRO) in order to reduce the training times. We have taken advantage of the NVIDIA Deep Learning GPU Training System[2] (DIGITS) which allows to design, train and visualize CNNs for image classification using Caffe[3] as deep learning framework. Some of the most popular architectures such as AlexNet and GoogLeNet are predefined and ready to use in the DIGITS platform.

B. SPARSE MATRIX DATASET

Deep networks demand large datasets to be effective. In our case, the dataset should include sparse matrices coming from different application domains with the aim of covering a broad spectrum of features and sparsity structures. To fulfill that requirement we have built a dataset consisting of more than 10k sparse matrices, which was generated applying several transformations like cropping to 812 square matrices

[2]https://developer.nvidia.com/digits
[3]http://caffe.berkeleyvision.org

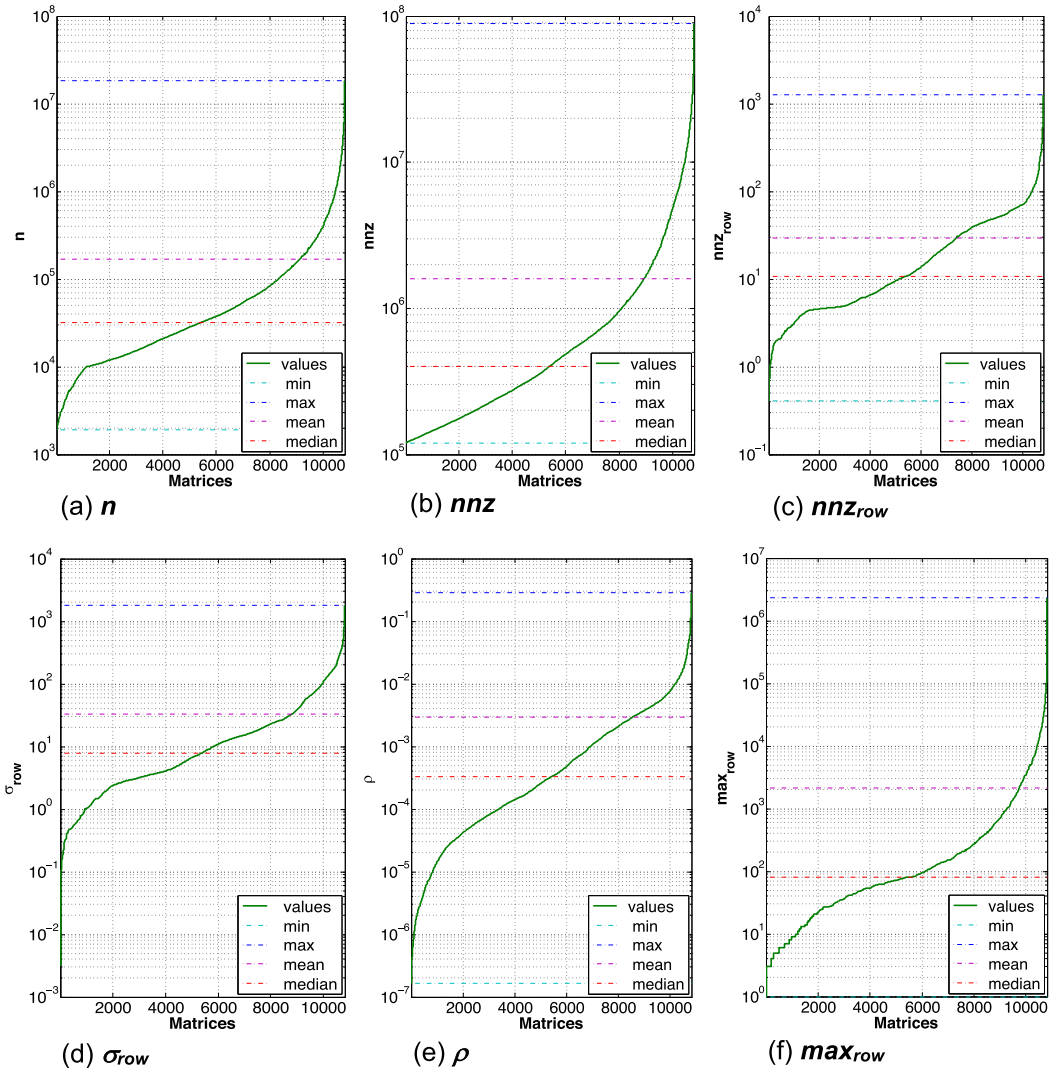

FIGURE 4. Characteristics of the sparse matrices in the dataset (values sorted in ascending order, *Y* axis in log scale).

from the SuiteSparse matrix collection [6]. A similar procedure was performed in [4].

Figure 4 shows the main characteristics, which correspond to the global metrics enumerated in Section III, for all the matrices in the dataset. Values in the graphs are sorted in ascending order. Minimum, maximum, mean and median values are also shown. It can be observed that ranges for each matrix feature are wide, which reveal the large diversity of matrices included in the dataset.

C. SpMV BENCHMARKING AND IMAGE DATASET GENERATION

Matrices should be labeled attending to their best storage format (class) before training a network. This goal is achieved in the SpMV benchmarking phase. Experiments to measure the performance of the single precision SpMV kernel using 6 storage formats (COO, CSR, HYB, ELL, BSR and CSR5) were conducted on the target GPUs. For each matrix and format, the performance value was calculated as the average of 1,000 SpMV operations. BSR uses 4×4 block sizes. Each matrix is then labeled according to the highest performing format. Table 2 shows the distribution of the classes for all the matrices in the dataset. The percentage of matrices belonging to each class is displayed between brackets. Note that COO never outperforms the other storage formats on both GPUs, similar to the behavior observed in [4]. According to the table, we are facing an unbalanced classification problem. HYB format represents just 0.3% of the examples in the

TABLE 2. Distribution of the classes in the matrix dataset.

Class	TITANX	QUADRO
COO	0	0
CSR	1,095 [10.1%]	1,134 [10.5%]
HYB	34 [0.3%]	33 [0.3%]
ELL	525 [4.9%]	531 [4.9%]
BSR	3,552 [32.8%]	3,584 [33.1%]
CSR5	5,616 [51.9%]	5,540 [51.2%]
Total	*10,822*	

dataset (blue text in the table), while more than 51% of the matrices belong to the CSR5 class (highlighted in red). In other words, the frequency of the HYB class is 160× less than the one obtained for CSR5. It has been demonstrated that class imbalance can have a harmful effect on training classifiers, so applying methods for addressing imbalance become necessary (see Section II-C). In the next section we will compare several approaches to deal with that issue in the context of the automatic selection of the best storage format for sparse matrices.

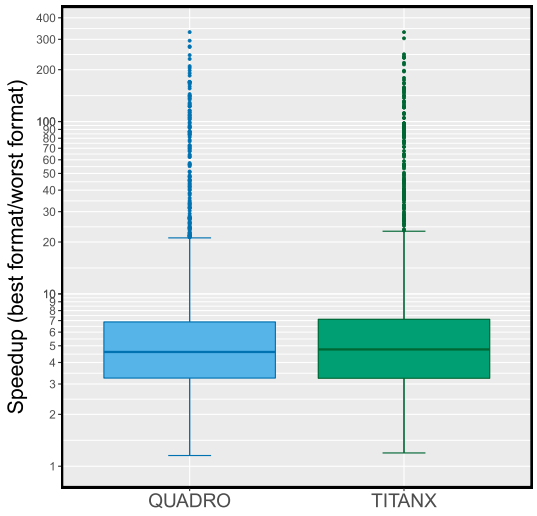

FIGURE 5. Speedup obtained by the best storage format with respect to the worst one for all the matrices in the dataset (*Y* axis in log scale).

On the other hand, choosing an inadequate storage format will impact negatively on the SpMV performance. This phenomenon is illustrated in Figure 5 by measuring the speedup between the best and the worst performing formats for all the matrices in the dataset. The boxplot shows that for QUADRO the median, first quartile and third quartile speedups are 4.6×, 3.3× and 7×, respectively. For TITANX the corresponding speedups are 4.8×, 3.3× and 7.2×. It means that, for example, selecting the best format for half of the matrices on TITANX boosts at least 4.8× their SpMV performance. It is worth noting that for some matrices a bad choice in the format is critical since it causes huge slowdowns. Those cases are displayed as points in the boxplots.

Before training the CNN it is also necessary to generate the images from the sparse matrices. As it was explained in Section III, the color of the pixels corresponds to different features of the matrices (see Figure 4). Their values should be in the 1-255 range, so a normalization process is necessary. Details about that procedure are provided in [3]. There are several combinations of channels and metrics, but in this work we only consider those which achieve the best performance results: $R_1G_3B_4$ and $R_0G_1B_4$. Datasets consist of 256×256 images, which corresponds to the input size for the AlexNet network.

D. NETWORKS AND TRAINING PROCESS

We have analyzed and studied two different CNNs to deal with the classification task: AlexNet [7] and SpNet. AlexNet consists of 8 layers: 5 are convolutional layers and other 3 are fully-connected layers (see Figure 6(a)). This network has about 60 million free parameters. Although AlexNet is relatively simple with respect to other standard networks, we demonstrated in [3] that using our methodology it can successfully classify sparse matrices considering a balanced dataset and three storage formats (classes). In this paper we go a step further introducing SpNet, a new simplified version of AlexNet which is able to achieve better results than the original network considering a more complex classification scenario (higher number of classes and an imbalanced dataset). Figure 6(b) shows the architecture of SpNet. It consists of only four convolution layers, corresponding to AlexNet *conv1*, *conv2*, *conv3* and *conv4* layers. Note that *conv3* and *conv4* in SpNet have a few number of filters, 256. In addition, SpNet has only two fully-connected layers. As a consequence, the number of free parameters is noticeably reduced.

In order to train the networks 80% of the matrices in the dataset are assigned to the training set, while the remainder 20% form the test set. As it was explained in Section III, we have used a *k-fold cross-validation* method to discover the optimal number of training epochs. It implies the division of the training set into 5 folds. We have found that the optimal number of epochs goes from 30 ($R_1G_3B_4$ dataset, SpNet and TITANX labels) to 42 ($R_1G_3B_4$ dataset, AlexNet and QUADRO labels). We must highlight that other hyperparameters take the default values provided by the DIGITS platform.

After the validation, AlexNet and SpNet networks are trained using the whole training set until the optimal number of epochs is reached. We have observed that training times vary from 492 seconds ($R_1G_3B_4$ dataset, SpNet and TITANX labels) to 666 seconds ($R_1G_3B_4$ dataset, AlexNet and QUADRO labels).

Since the class of a matrix depends on the target GPU where the SpMV operation is evaluated, networks should be trained for each particular platform. However, we have previously proven that using our methodology is not necessary to carry out the training process from scratch [3]. The idea is to consider a pre-trained model as starting point of the training process. This pre-trained model corresponds to a CNN trained for a different GPU. In this way, the network inherits many parameters which captured important characteristics and features from the considered storage formats and matrices in the dataset. In addition, we must take into account that classes of the matrices differ between GPUs, but not for all the dataset. As a consequence, very good accuracy results can be obtained using less training data and, at the same time, training times decrease. Another important effect of reducing the training data size is the impact on the SpMV benchmarking phase, which is greatly shortened.

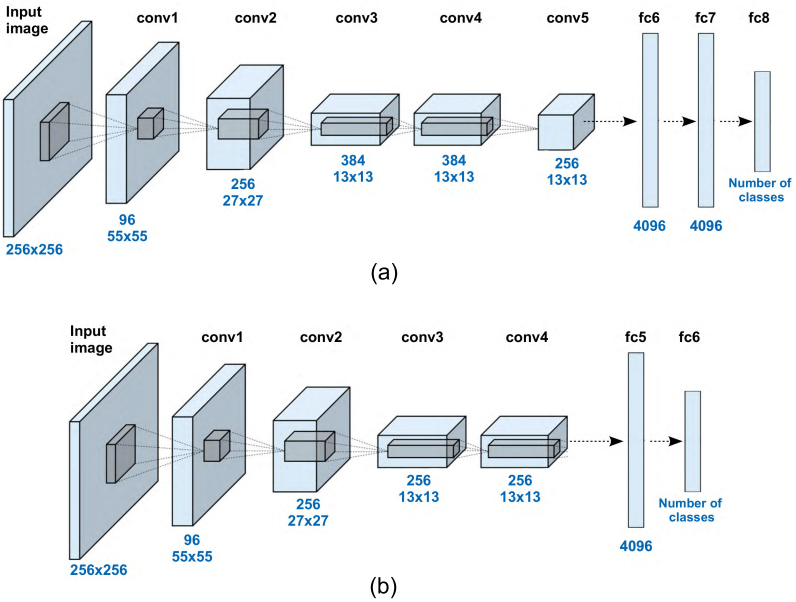

FIGURE 6. Layers of the considered networks (number and size of the filters in blue text): (a) AlexNet and (b) SpNet.

TABLE 3. Prediction accuracy of the trained networks considering the two image datasets on QUADRO (top) and TITANX (down) GPUs. Highlighted best results for each format.

QUADRO	AlexNet						SpNet					
	$R_1G_3B_4$			$R_0G_1B_4$			$R_1G_3B_4$			$R_0G_1B_4$		
	Recall	*Prec.*	*F1*	*Recall*	*Prec.*	*F1*	*Recall*	*Prec.*	*F1*	*Recall*	*Prec.*	*F1*
CSR	0.687	***0.872***	0.768	0.683	0.871	0.765	0.678	0.778	0.725	***0.700***	0.869	***0.776***
HYB	-	-	-	0.286	0.667	0.400	0.143	***1***	0.250	***0.429***	***1***	***0.600***
ELL	***0.832***	0.774	0.802	0.738	***0.868***	0.798	0.757	0.786	0.771	0.785	0.840	***0.812***
BSR	0.948	0.948	0.948	***0.964***	0.939	0.951	0.954	0.933	0.943	0.955	***0.950***	***0.953***
CSR5	0.961	0.922	0.941	0.960	0.919	0.939	0.946	***0.927***	0.936	***0.967***	0.924	***0.945***
Global Accuracy	91.9%			91.9%			90.9%			***92.4%***		

TITANX	AlexNet						SpNet					
	$R_1G_3B_4$			$R_0G_1B_4$			$R_1G_3B_4$			$R_0G_1B_4$		
	Recall	*Prec.*	*F1*	*Recall*	*Prec.*	*F1*	*Recall*	*Prec.*	*F1*	*Recall*	*Prec.*	*F1*
CSR	0.685	0.781	0.730	0.689	0.830	0.753	0.676	0.796	0.731	***0.721***	***0.832***	***0.773***
HYB	-	-	-	0.143	0.200	0.167	0.143	0.333	0.200	***0.286***	***0.400***	***0.333***
ELL	0.829	***0.837***	0.833	0.771	0.818	0.794	***0.838***	0.830	***0.834***	0.790	0.830	0.810
BSR	0.959	0.951	0.955	0.961	0.957	0.959	0.951	0.948	0.949	***0.965***	***0.958***	***0.961***
CSR5	0.953	0.929	0.941	0.959	0.925	0.941	0.953	0.925	0.939	***0.961***	***0.935***	***0.948***
Global Accuracy	91.9%			92.0%			91.6%			***92.8%***		

V. PERFORMANCE ANALYSIS

A. PREDICTION ACCURACY

The evaluation of the trained networks for each dataset and GPU was carried out using only the test set. Together with the global accuracy (i.e. the overall percentage of correct classified matrices), we have included three additional metrics to better understand how well the classifier is performing: *precision*, *recall* and *F1 score*. Precision is the fraction of positive predictions made by the classifier that are correct. Recall, also known as sensitivity, quantifies how well the model avoids false negatives. Let's assume that there are T_A matrices of class A in the dataset. Our network classifies C_A matrices as class A, where P_A has been correctly classified (true positive). Then, precision and recall for class A can be calculated as ${}^{P_A}/_{C_A}$ and ${}^{P_A}/_{T_A}$, respectively. F1 score takes into account both precision and recall and is calculated as the harmonic mean of both metrics: $2 \times \frac{precision \times recall}{precision + recall}$.

Table 3 displays the global accuracy, precision, recall and F1 score of the classifiers for all the datasets on both GPUs considering AlexNet and SpNet networks. According to the results, several observations can be made. First, noticeable accuracies were obtained using our methodology for all the cases, ranging from 90.9% to 92.8%. Best results overall correspond to the $R_0G_1B_4$ dataset, which was generated

using as metrics the size of the matrix (n), the average number of nonzeros per row (nnz_{row}) and the maximum number of nonzeros in a row of the matrix (max_{row}). Second, SpNet clearly outperforms AlexNet when considering $R_0G_1B_4$ images, while with $R_1G_3B_4$ the performance is in several cases lower. AlexNet results for the minority class HYB are very poor, while SpNet is able to reach acceptable values on the QUADRO GPU (F1 score is 0.600). Note that HYB matrices represent only 0.3% of the dataset (see Table 2). In any case, we will discuss several approaches to improve the classification performance of the minority classes in the following section. We can conclude that SpNet using $R_0G_1B_4$ images is clearly the best option on both GPUs since only in a few cases its precision, recall and F1 values are not the highest among all. In addition, top global accuracies were obtained using this configuration. We also want to highlight that regarding the majority classes (BSR and CSR5), most of the performance metrics are above 0.95. Finally, it can be observed a slightly better classification results when training the CNNs for TITANX.

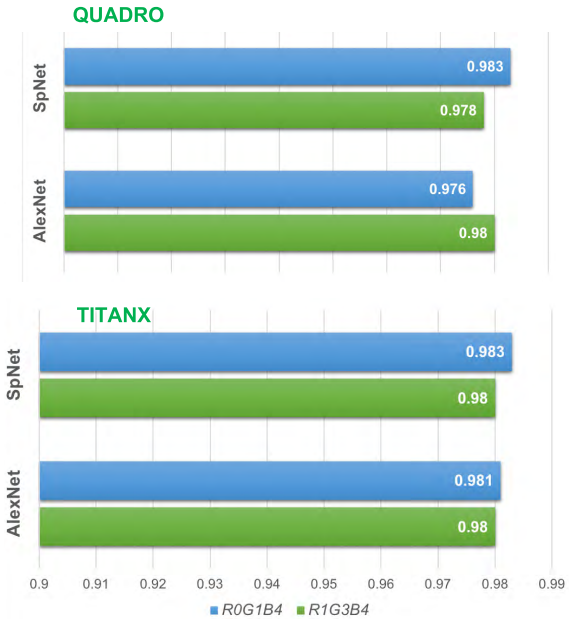

FIGURE 7. Normalized SpMV performance obtained using the storage format selected by the classifiers.

Since the final goal of the classification task is selecting the best performing format, it is important to measure how close to the ideal classifier our trained CNNs are. In this way, we obtained the SpMV performance for all the matrices in the test set using the format selected by each classifier. Normalized results for all the configurations analyzed are shown in Figure 7 in such a way that 1 corresponds to the maximum achievable performance (i.e., always choosing the best format). Average performances range from 97.6% to 98.3%. Best results overall on both GPUs are obtained when considering SpNet network and $R_0G_1B_4$ images, which is the best configuration also in terms of the classification metrics (see Table 3). However, other configurations with lower accuracies perform very well too. For instance, the configuration using $R_1G_3B_4$ images and SpNet on the QUADRO GPU achieve the lowest global accuracy (90.9%), but it reaches 98% of the maximum attainable SpMV performance. Therefore, considering only the classification metrics in Table 3 is not enough to validate a classification method in this context because it hides important information regarding performance. In this way, the above analysis has also a great importance.

B. ADDRESSING CLASS IMBALANCE

There are several approaches to deal with the class imbalance problem (see Section II-C). We have considered three different methods: oversampling, undersampling and cost-sensitive learning using a weighted loss function. The first technique replicates examples from minority classes to build a more balanced dataset. Afterwards the network is trained using the new dataset. There are many possible ways of replicating the samples. After many experiments we have found that the best configuration replicates 2, 4 and 10 times the training examples corresponding to CSR, ELL and HYB classes, respectively.

The second method is undersampling, which randomly remove examples from the majority classes to balance the dataset. Several experiments were carried out to find out the optimal configuration. However, performance results obtained by this technique were not competitive, so they are not included in the paper. The problem is relatively obvious since removing examples from the majority classes causes the classifier to miss important concepts pertaining to them [5]. This fact is magnified in our case because the dataset is relatively small with respect to classic image classification datasets containing millions of images.

The last method analyzed to address class imbalance uses a weighted loss function. In this way, it is possible to assign different costs for misclassifying any particular data sample. In particular, our proposal weights the error rate provided by the cross entropy loss layer of the network according to the frequency of each class in the dataset. A similar approach was applied to classifying malware images in [23]. The weighted cross entropy loss has the following expression (InfoGainLoss layer in Caffe)[4]:

$$E = -\frac{1}{M}\sum_{n=1}^{M}\sum_{k=1}^{K} H_{l_n,k} log(\hat{p}_{n,k}) \quad (1)$$

being M the batchsize, K is the number of classes, l_n is the correct class of image n, while $\hat{p}_{n,k}$ is the probability of image n to belong to class k. H is a $K \times K$ matrix containing the weight or contribution to the loss according to the correct class of the example. Higher values in H indicate more

[4]http://caffe.berkeleyvision.org/tutorial/layers/infogainloss.html

TABLE 4. Prediction accuracy of the trained SpNet networks considering different methods to deal with imbalance using $R_0G_1B_4$ images. Highlighted those values that improve the corresponding results in Table 3.

Classes	QUADRO						TITANX					
	Oversampling			Weighted Loss			Oversampling			Weighted Loss		
	Recall	*Prec.*	*F1*	*Recall*	*Prec.*	*F1*	*Recall*	*Prec.*	*F1*	*Recall*	*Prec.*	*F1*
CSR	***0.744***	0.793	0.773	***0.714***	0.831	0.768	0.712	0.780	0.745	***0.744***	0.791	0.767
HYB	***0.571***	0.800	***0.667***	***0.571***	0.571	0.571	***0.714***	***0.556***	***0.625***	***0.429***	***0.500***	***0.462***
ELL	***0.804***	***0.878***	***0.839***	***0.822***	***0.880***	***0.850***	***0.857***	***0.841***	***0.849***	***0.838***	0.822	***0.830***
BSR	***0.958***	***0.957***	***0.957***	***0.958***	0.948	0.953	0.965	***0.962***	***0.963***	0.965	***0.962***	***0.963***
CSR5	0.957	***0.936***	***0.946***	0.957	***0.931***	0.943	0.949	***0.938***	0.944	0.950	***0.942***	0.946
Global Accuracy	***92.6%***			92.4%			92.5%			92.7%		

TABLE 5. Prediction accuracy of a state-of-the-art technique based on building a decision tree for sparse matrix classification [1]. In brackets differences in performance with respect to our oversampling approach.

Classes	QUADRO			TITANX		
	Recall	*Prec.*	*F1*	*Recall*	*Prec.*	*F1*
CSR	0.543 *(-0.201)*	0.581 *(-0.212)*	0.562 *(-0.211)*	0.555 *(-0.157)*	0.553 *(-0.227)*	0.554 *(-0.191)*
HYB	0.333 *(-0.238)*	0.286 *(-0.514)*	0.308 *(-0.359)*	0.571 *(-0.143)*	0.571 *(+0.015)*	0.571 *(-0.054)*
ELL	0.667 *(-0.137)*	0.654 *(-0.224)*	0.660 *(-0.179)*	0.607 *(-0.250)*	0.676 *(-0.165)*	0.640 *(-0.209)*
BSR	0.867 *(-0.091)*	0.856 *(-0.101)*	0.862 *(-0.095)*	0.858 *(-0.107)*	0.861 *(-0.101)*	0.860 *(-0.103)*
CSR5	0.861 *(-0.096)*	0.858 *(-0.078)*	0.860 *(-0.086)*	0.879 *(-0.070)*	0.869 *(-0.069)*	0.874 *(-0.070)*
Global Accuracy	81.7% *(-10.9)*			82.4% *(-10.1)*		

penalty to misclassification. We define H as:

$$H = \begin{cases} H_{i,j} = 0, & \text{if } i \neq j \\ H_{i,j} = 1 - r_i, & \text{if } i = j \end{cases} \quad (2)$$

where r_i is the ratio of examples of class i in the dataset (ratios in percentage shown in Table 2). Note that if H is the identity matrix, the weighted loss becomes a standard cross entropy loss (also known as multinomial logistic regression loss).

Table 4 shows the classification metrics obtained when applying oversampling and the weighted loss approach using the best performing configuration in Table 3, which corresponds to train the SpNet network using the $R_0G_1B_4$ dataset. Highlighted values correspond to improvements in recall, precision, F1 score and accuracy with respect to the scenario where the imbalance class problem is not specifically addressed. A better overall behavior is observed, increasing the performance of at least one metric for all the classes. Both methods, oversampling and the cost-sensitive approach, obtain similar results. Oversampling does a better job classifying the minority class HYB. For instance, a recall of 0.714 is reached on TITANX. Both techniques boost the classification performance of ELL format, which corresponds to only 4.9% of the matrices in the dataset (see Table 2). On the other hand, oversampling on the QUADRO GPU increases the global accuracy until 92.6%. In the other cases, a slightly reduction was observed.

In the previous section we measured the difference between the SpMV performance obtained when using the storage formats selected by our classifiers and the maximum attainable performance (that is, choosing always the best format). Results when considering SpNet and the $R_0G_1B_4$ dataset obtain an average normalized performance of 98.3% (see Figure 7). The same experiment was carried out considering the techniques to deal with the class imbalance on both GPUs. The corresponding values on the QUADRO GPU were 98.4% and 98.3% when using oversampling and the weighted loss approach, respectively. On the TITANX the average normalized performance was 98.3% in both cases. Therefore, techniques to address with class imbalance, far from degrading performance, are able to get closer to the maximum achievable SpMV performance.

C. COMPARISON WITH OTHER STATE-OF-THE-ART CLASSIFICATION METHODS

We have compared our approach with a state-of-the-art technique that automatically predicts the best sparse representation using decision trees [1]. Trees are based on different features (similar to those described in Section III) from a set of training matrices. They demonstrate that their approach outperforms previous methods in classification accuracies and SpMV performance. Note that this is a traditional machine learning approach, where no images are necessary.

Table 5 shows the prediction metrics obtained by the decision tree method when applying to our dataset. Values in brackets are the differences with respect to our oversampling approach, whose results were displayed in Table 4. It can be observed that our proposal clearly outperforms the decision tree model. It is especially relevant the fact that in general differences between both methods increase for the minority classes (CSR, ELL and HYB formats). As a consequence, global accuracies of the tree model are reduced to 81.7% and 82.4% on QUADRO and TITANX GPUs, respectively.

In a recent work authors addressed the classification of sparse matrices from a deep learning perspective. Matrices are represented through histograms which model the spatial distribution of nonzeros in the matrix. This representation requires to build a customized CNN architecture. We can highlight the following differences with respect to our approach:

- They also deal with a six storage formats classification problem on a GPU (COO, CSR, HYB, ELL, BSR, CSR5), but their dataset is not highly imbalanced. In our case, HYB format represents only 0.3% of the matrices.
- Their dataset consists of 4,218 matrices, while we have considered 10,822. No details about the characteristics of the matrices in their dataset are provided. In our case an analysis to demonstrate that our dataset covers a wide spectrum of matrix features is shown in Figure 4.
- Their methodology was evaluated only considering one GPU (TITANX).
- Although the results are not directly comparable (datasets are different), our methodology achieves a higher performance. In particular, authors report a 90% global accuracy, while our method reaches 92.8% (without addressing specifically the class imbalance problem).

VI. CONCLUSIONS

Training CNNs for classification using imbalanced datasets is a challenging task since an inductive bias towards the majority classes is produced, resulting in poor minority class recognition performance. In this work we deal with a highly imbalanced problem in the context of the automatic selection of the best SpMV performing storage format for sparse matrices on GPUs. To overcome the issues caused by the imbalanced data we made contributions at different levels.

We used a method to generate the image datasets that considers the sparsity pattern of the matrices as an image. Pixels in the images represent submatrices and their RGB color codes some global matrix property. According to our experiments, the best combination of matrix features corresponds to the size, average number of nonzeros per row and the maximum number of nonzeros in a row of the matrix.

Although a standard CNN as AlexNet trained using our image datasets achieved acceptable results in terms of prediction accuracy, we introduce a new network called SpNet. This network is a simplified version of AlexNet with only four convolution and two fully-connected layers. Experimental results demonstrate that SpNet clearly outperforms AlexNet, reaching a global accuracy of 92.8% and 98.3% of the highest SpMV performance among the considered formats. Our experiments also show that our approach performs much better than a state-of-the-art classification method based on decision trees.

Finally, several methods to specifically overcome the issues related to training a CNN using imbalance data were studied. An important overall improvement in the classification metrics was observed when applying oversampling or a cost-sensitive method based on a weighted loss function, especially for the minority classes.

REFERENCES

[1] N. Sedaghati, T. Mu, L.-N. Pouchet, S. Parthasarathy, and P. Sadayappan, "Automatic selection of sparse matrix representation on GPUs," in *Proc. 29th ACM Int. Conf. Supercomput. (ICS)*, Jun. 2015, pp. 99–108.

[2] H. Cui, S. Hirasawa, H. Takizawa, and H. Kobayashi, "A code selection mechanism using deep learning," in *Proc. IEEE MCSOC*, Sep. 2016, pp. 385–392.

[3] J. C. Pichel and B. Pateiro-López, "A new approach for sparse matrix classification based on deep learning techniques," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Sep. 2018, pp. 46–54.

[4] Y. Zhao, J. Li, C. Liao, and X. Shen, "Bridging the gap between deep learning and sparse matrix format selection," in *Proc. 23rd ACM SIGPLAN Symp. Princ. Pract. Parallel Program. (PPoPP)*, 2018, pp. 94–108.

[5] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, Sep. 2009.

[6] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, p. 1, 2011.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. NIPS*, vol. 1. 2012, pp. 1097–1105.

[8] D. Langr and P. Tvrdik, "Evaluation criteria for sparse matrix storage formats," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 428–440, Feb. 2016.

[9] J. W. Choi, A. Singh, and R. W. Vuduc, "Model-driven autotuning of sparse matrix-vector multiply on GPUs," in *Proc. 15th ACM SIGPLAN Symp. Princ. Pract. Parallel Program. (PPoPP)*, 2010, pp. 115–126.

[10] N. Bell and M. Garland, "Efficient sparse matrix-vector multiplication on CUDA," NVIDIA Corp., Santa Clara, CA, USA, Tech. Rep. NVR-2008-004, Dec. 2008.

[11] W. Liu and B. Vinter, "CSR5: An efficient storage format for cross-platform sparse matrix-vector multiplication," in *Proc. 29th ACM Int. Conf. Supercomput. (ICS)*, Jun. 2015, pp. 339–350.

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

[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[14] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: https://arxiv.org/abs/1409.1556

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015, *arXiv:1512.03385*. [Online]. Available: https://arxiv.org/abs/1512.03385

[16] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intell. Data Anal.*, vol. 6, no. 5, pp. 429–449, Oct. 2002.

[17] M. Buda, A. Maki, and M. A. Mazurowski, "A systematic study of the class imbalance problem in convolutional neural networks," *Neural Netw.*, vol. 106, pp. 249–259, Oct. 2017.

[18] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, no. 1, pp. 321–357, 2002.

[19] N. Jaccard, T. W. Rogers, E. J. Morton, and L. D. Griffin, "Detection of concealed cars in complex cargo X-ray imagery using deep learning," *J. X-Ray Sci. Technol.*, vol. 25, no. 3, pp. 323–339, Jan. 2017.

[20] G. Levi and T. Hassner, "Age and gender classification using convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, Jun. 2015, pp. 34–42.

[21] S. H. Khan, M. Hayat, M. Bennamoun, F. A. Sohel, and R. Togneri, "Cost-sensitive learning of deep feature representations from imbalanced data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 8, pp. 3573–3587, Aug. 2018.

[22] Q. Dong, S. Gong, and X. Zhu, "Imbalanced deep learning by minority class incremental rectification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 6, pp. 1367–1381, Jun. 2019.
[23] S. Yue, "Imbalanced malware images classification: A CNN based approach," 2017, *arXiv:1708.08042*. [Online]. Available: http://arxiv.org/abs/1708.08042
[24] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, and P. J. Kennedy, "Training deep neural networks on imbalanced data sets," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2016, pp. 4368–4374.
[25] Z.-H. Zhou and X.-Y. Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 1, pp. 63–77, Jan. 2006.
[26] H. Yu, C. Sun, X. Yang, W. Yang, J. Shen, and Y. Qi, "ODOC-ELM: Optimal decision outputs compensation-based extreme learning machine for classifying imbalanced data," *Knowl. Based Syst.*, vol. 92, pp. 55–70, Jan. 2016.
[27] P. Guo, L. Wang, and P. Chen, "A performance modeling and optimization analysis tool for sparse matrix-vector multiplication on GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 5, pp. 1112–1123, May 2014.
[28] B. Neelima, G. R. M. Reddy, and P. S. Raghavendra, "Predicting an optimal sparse matrix format for SpMV computation on GPU," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, May 2014, pp. 1427–1436.
[29] K. Li, W. Yang, and K. Li, "Performance analysis and optimization for SpMV on GPU using probabilistic modeling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 1, pp. 196–205, Jan. 2015.
[30] A. Benatia, W. Ji, Y. Wang, and F. Shi, "Sparse matrix format selection with multiclass SVM for SpMV on GPU," in *Proc. 45th Int. Conf. Parallel Process. (ICPP)*, Aug. 2016, pp. 496–505.
[31] J. Li, G. Tan, M. Chen, and N. Sun, "SMAT: An input adaptive auto-tuner for sparse matrix-vector multiplication," in *Proc. 34th ACM SIGPLAN Conf. Program. Lang. Design Implement. (PLDI)*, 2013, pp. 117–126.

JUAN C. PICHEL received the B.Sc. and M.Sc. degrees in physics from the University of Santiago de Compostela, Spain, and the Ph.D. degree in computer science from the University of Santiago de Compostela, in 2006, where he is currently an Associate Professor. He was a Visiting Postdoctoral Researcher with the University Carlos III de Madrid, Spain, and the University of Illinois at Urbana–Champaign, USA. He was a Researcher and the Project Manager with the Galicia Supercomputing Center, Spain. His research interests include parallel and distributed computing, big data technologies, programming models, and software optimization techniques for emerging architectures.

BEATRIZ PATEIRO-LÓPEZ received the B.Sc. and M.Sc. degrees in mathematics from the University of Santiago de Compostela, Spain, and the Ph.D. degree in statistics and operations research from the University of Santiago de Compostela, in 2008. After completing her Ph.D. degree, she was a Postdoctoral Researcher with the Universidad Autónoma de Madrid, Spain, and the Universidad de San Andrés, Argentina. She is currently an Associate Professor with the Department of Statistics, Mathematical Analysis and Optimization, University of Santiago de Compostela. She has actively participated in national and international funded projects. She is the coauthor of publications accepted in some of the highest ranked journals on statistics and probability. Her research interests include set estimation and computational statistics.