A New Approach for Sparse Matrix Classification Based on Deep Learning Techniques

Juan C. Pichel
CiTIUS
Universidade de Santiago de Compostela
Santiago de Compostela, Spain
juancarlos.pichel@usc.es

Beatriz Pateiro-López
Dpto. de Estadística, Análisis Matemático y Optimización
Universidade de Santiago de Compostela
Santiago de Compostela, Spain
beatriz.pateiro@usc.es

***Abstract*—In this paper, a new methodology to select the best storage format for sparse matrices based on deep learning techniques is introduced. We focus on the selection of the proper format for the sparse matrix-vector multiplication (SpMV), which is one of the most important computational kernels in many scientific and engineering applications. Our approach considers the sparsity pattern of the matrices as an image, using the RGB channels to code several of the matrix properties. As a consequence, we generate image datasets that include enough information to successfully train a Convolutional Neural Network (CNN). Considering GPUs as target platforms, the trained CNN selects the best storage format 90.1% of the time, obtaining 99.4% of the highest SpMV performance among the tested formats.**

***Index Terms*—Sparse matrix, Classification, Deep Learning, CNN, Performance**

I. Introduction

Sparse matrix-vector multiplication (SpMV) is a key kernel at the core of many scientific and engineering applications. SpMV is notorious for sustaining low fractions of the peak performance on modern parallel architectures. As a consequence, it has attracted a lot of attention from the research community to develop efficient and optimized implementations. The performance of the SpMV depends on both the target hardware platform and the sparsity structure of the matrix. For this reason many storage formats have been proposed for a particular application domain, matrix structure and computer architecture [1]. It has been demonstrated that the selection of the proper storage format has a big impact on the SpMV performance. The compressed sparse row (CSR) format is the *de-facto* standard representation for CPUs, while there is no a dominant format for GPUs. We find the cause in several factors that often conflict with each other [2]: maximizing coalesced memory access, minimizing thread divergence and maximizing warp occupancy.

In this paper we address the problem of the automatic selection of the best storage format for sparse matrices on GPUs. With this goal in mind a new methodology based on deep learning technologies is introduced. In particular, we have considered Convolutional Neural Networks (CNNs), which are the most important deep learning networks for image recognition and classification. Our goal is to demonstrate that a simple standard CNN architecture as AlexNet [3] is powerful enough to provide very good classification results. Therefore, it is not necessary to build an *ad-hoc* CNN architecture to deal with the problem. In this way, our methodology can be easily adopted by the research community since AlexNet is available in the most important and common deep learning frameworks. To train the network the sparsity pattern of the matrices is considered as an image. Since the input size of CNNs is fixed, original sparse matrices are scaled down to fit the CNN in such a way that pixels in the images represent submatrices. The RGB color of pixels is used to represent properties of the matrix. In this way, we create image datasets with enough information to successfully train a CNN. An exhaustive experimental evaluation has been carried out using two different GPUs as target platforms. Results show the benefits of our methodology in terms of the global accuracy of the resulting classifiers, reaching values above 90%. In addition, we are able to obtain within 99.4% on average of the best SpMV performance available. Finally, we demonstrate that using a pre-trained model speeds up the training process with respect to training the network for each GPU from scratch.

The paper is structured as follows. Section II explains the background of the work. Section III introduces the deep learning methodology to deal with the sparse matrix classification problem. Experimental results are shown and discussed in Section IV. Related work is presented in Section V. Finally, the main conclusions derived from the work together with some ideas for future work are explained.

II. Background

A. Sparse Matrix Formats

For a sparse matrix, substantial memory requirement reductions can be obtained by storing only the nonzero entries. There exist many different storage formats (an exhaustive list can be found in [1]), being ones more appropriate than others for a particular sparse matrix depending on the number and distribution of its nonzeros. These formats differ in terms of the amount of storage required, the accessing methods, and their adaptability to different applications or parallel architectures such as GPUs. Some of these formats are only well suited

This work has been supported by MINECO (TIN2014-54565-JIN and MTM2016-76969-P), Xunta de Galicia (ED431G/08) and European Regional Development Fund.

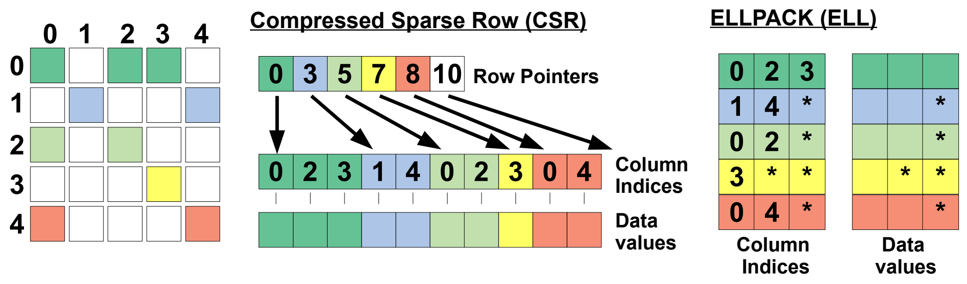

Fig. 1: CSR and ELL sparse matrix storage formats.

for matrices with a particular sparsity pattern like the diagonal format (DIA) or block formats such as BELLPACK [4], other formats support efficient modification but not efficient matrix operations like for example the coordinate format (COO), and so on. In this work, we focus on those formats that are suitable for matrices with arbitrary structure and, at the same time, efficient for matrix operations such as sparse matrix-vector multiplication. More precisely, we have considered the compressed row storage (CSR), ELLPACK (ELL), and hybrid (HYB) formats [5], which are implemented in the NVIDIA cuSPARSE[1] library (see Figure 1):

- Compressed Sparse Row (CSR): It is a general-purpose sparse matrix format. No assumptions are needed about the sparsity structure of the matrix. CSR allocates subsequent nonzeros in each row in contiguous memory locations and stores column indices and nonzero entries in two arrays, indices and values respectively. Besides, it needs another array of pointers that indicates the offset for each row.
- ELLPACK (ELL): This storage scheme compresses the original sparse $n \times m$ matrix in a dense $n \times k$ matrix, where k is the maximum number of nonzeros per row of the original matrix. It also needs another $n \times k$ array of indices which stores the position (column) of each nonzero in the original matrix. This format cannot be considered a general-purpose matrix format because it needs that the number of nonzeros in each row do not vary greatly through all the rows. In other case, a lot of storage space will be wasted and also the computational efficiency will decrease. However, it is suitable for a variety of matrices and the performance results it produces are generally good.
- Hybrid (HYB): This is a combination of two storage formats: COO and ELL. It tries to combine the computation efficiency of ELL with the simplicity and generality of COO (that stores row and column indices explicitly). The majority of the matrix entries are stored in ELL format, and those rows with a substantially different number of nonzeros are stored in COO format.

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

B. Convolutional Neural Networks

CNNs consist of a sequence of layers which transform the original image layer by layer from the original pixel values to the final class scores. We can classify these layers into three groups: input layers, feature-extraction layers and classification layers. A simple CNN architecture is shown in Figure 2. Input layers load and store the raw input data of the image for processing in the network. This input data specifies the width, height, and number of channels. Typically, the number of channels is three, corresponding to the RGB values for each pixel.

The feature-extraction layers have a general repeating pattern of the following operations: convolution, non linearity (ReLU) and pooling or sub sampling. The main goal of a convolution layer is to extract features from the input image. Convolution shifts a small window (filter) across the input, and at each position, it computes the dot product between the filter and the input elements covered by the filter. As we slide the filter over the image we will produce a 2D activation map that gives the responses of that filter at every spatial position. In this way, the network will learn filters that activate when they detect some type of visual feature such as edges, curves, etc. Note that several filters can be used in the same convolution layer, which will generate multiple activation maps. An additional operation called ReLU (Rectified Linear Unit) has been used after every convolution operation. It is a non-linear operation that replaces all negative pixel values in the feature map by zero. In addition, it is common to insert a pooling layer in-between successive convolution+ReLU layers in a CNN architecture. Its function is to progressively reduce the spatial size of the representation (sub sampling) to decrease the amount of parameters and computation in the network while retaining the most important information.

Finally, we have the classification layers in which we have one or more fully-connected layers to produce class scores. Fully-connected means that neurons in this layer have full connections to all activations in the previous one. This layer uses the high level features generated by the convolutional and pooling layers for classifying the input image into several classes based on the training dataset.

The training process of a CNN is iterative. First, all the filters and parameters in the network are initialized to random values. Afterwards the network takes a training input image, whose class/label is known *a priori*, and gives a prediction

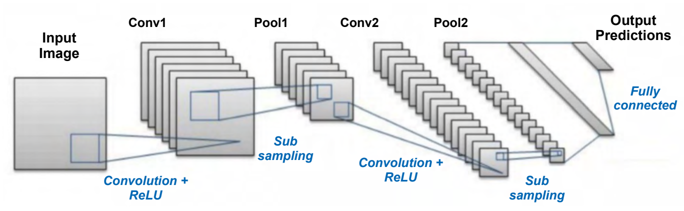

Fig. 2: A simple Convolutional Neural Network (CNN) architecture.

after the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the fully-connected layers). A prediction error is calculated comparing the output of the network and the expected result. The training process (by means of back propagation) revises the network parameters iteratively to minimize the overall error on each training input. The network will be trained on the input dataset for a given number of epochs (that is, passes over the entire image dataset). Note that parameters like number of filters, filter sizes, architecture of the network, etc., do not change during the training process. There are additional parameters in the training process known as hyperparameters such as the learning rate or number of epochs that should be tuned to make networks train better and faster.

Many CNN architectures have been proposed, some of the most popular are LeNet [6], AlexNet [3], GoogLeNet [7], VGGNet [8] and ResNet [9]. In this paper we have used AlexNet, which has five convolution layers of decreasing filter size, three pooling layers, and three fully-connected layers with approximately 60 million free parameters. Although AlexNet is relatively simple with respect to other standard networks, we demonstrate in the following sections that it is powerful enough to deal with the sparse matrix format selection problem.

III. Methodology

In this section a new methodology to select the best storage format for sparse matrices based on deep learning techniques is introduced. In particular, we have focused on the selection of the proper format for the sparse matrix-vector multiplication (SpMV), which is one of the most important computational kernels in scientific and engineering applications.

Figure 3 shows the different phases of our approach. We assume that a large set of sparse matrices coming from different real problems and representing a variety of characteristics and nonzero patterns is available. This dataset will be used as input of the SpMV benchmarking and image generation phase. The goal of the first step is to evaluate for all the matrices in the dataset the performance of the SpMV kernel when different storage formats are considered. As a result we obtain 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. Note that in this work we have considered GPUs as hardware platforms to build the ground truth information, but our methodology is completely agnostic with respect to the underlying parallel system and can be applied, for example, to multicore CPUs or accelerators as the Intel Xeon Phi.

The image dataset generation is the core of our methodology. To build the dataset we consider the sparsity pattern of the matrices as an image. As a first approach, a $n \times m$ matrix is equivalent to a $n \times m$ binary image where a white pixel at location (i, j) represents a nonzero in row i and column j. Black pixels correspond to zeros in the sparsity pattern. However, the size of the input to a CNN is fixed, so matrices of different sizes should be scaled to the same size. The following method explains how to scale a matrix. Let's assume that the size of the input image should be $p \times p$ pixels and, for simplicity, the considered sparse matrix is $n \times n$ (i.e., it is a square matrix), where $n > p$. We split the matrix 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 submatrix (i, j). If the submatrix only contains zeros then the corresponding entry in the scaled matrix will be zero. In this way, creating a $p \times p$ binary image from the scaled matrix is straightforward. Figure 4(a) illustrates this procedure showing a 63×63 pixels image generated from a $71,505 \times 71,505$ sparse matrix.

The above method is a simple and easy way to generate a binary image dataset that fits a CNN. However, scaling down a sparse matrix simplifies the appearance of its sparsity pattern, which could cause a loss in the information provided to the CNN in the training phase. Recall that a single pixel in the image represents a submatrix in the original matrix. For instance, one pixel in Figure 4(a) corresponds to a $1,135 \times 1,135$ submatrix (that is, ${}^{71,505}/{}_{63}$). As we demonstrate in Section IV, training the network using the binary image dataset do not provide competitive results. In this way, it is necessary to provide additional information to the CNN with the aim of improving the global accuracy of the classifier. With this goal in mind, we propose to use the RGB channels of the image to code information related to some characteristics and properties 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

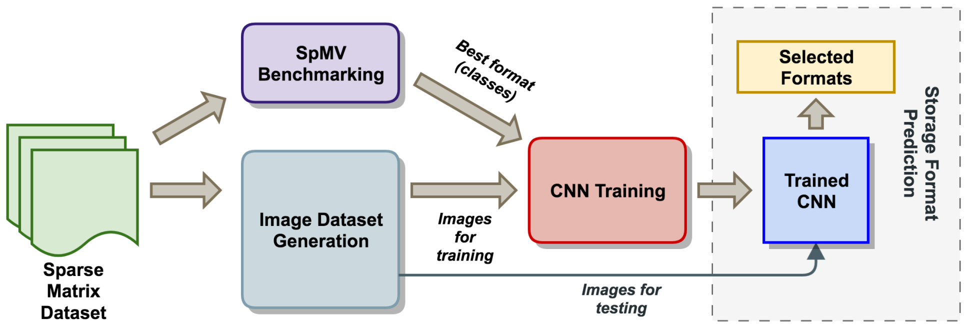

Fig. 3: Scheme of the new classification methodology.

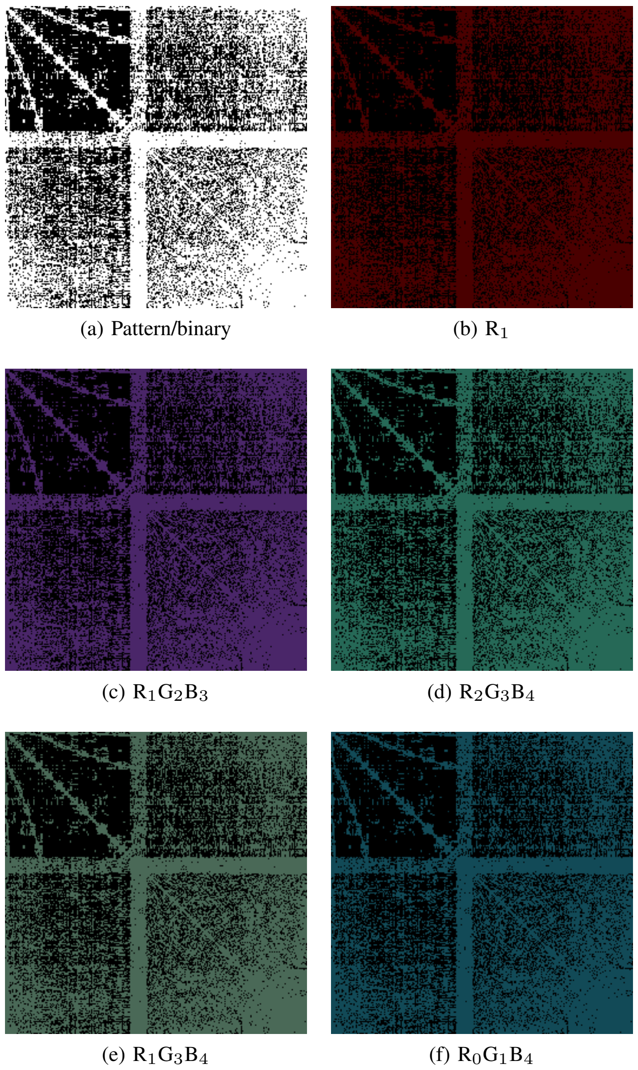

Fig. 4: Images of 63×63 pixels generated from a $71,505 \times 71,505$ sparse matrix using different number of channels and metrics.

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 (details about the normalization of our dataset are provided in Section IV). 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.

From now on the notation $R_xG_yB_z$ is used to indicate that metrics x, y and z were selected to calculate the R, G and B values of an image, respectively. There are multiple combinations of number of channels and metrics that can be utilized in the image dataset generation phase. In this paper we only show results for the most relevant combinations in terms of performance. In particular, datasets were generated using the following configurations: $R_1G_2B_3$, $R_2G_3B_4$, $R_1G_3B_4$ and $R_0G_1B_4$. In addition, for illustrative purposes, we have also included results for a binary image dataset (black and white pixels, without metrics) and R_1 (using only the red channel to code the average number of nonzeros per row of the matrix). Therefore, six different image datasets have been generated and analyzed in the paper. An example is shown in Figure 4 that displays the resulting images obtained for the same input sparse matrix when considering different configurations. 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 $R_3G_1B_2$.

The next stage in our method involves the training of the CNN. To do so, it is necessary to feed the CNN with a set of images labeled with their class (best storage format). This data was generated in the previous phases: SpMV benchmarking and image generation. Note that the image dataset is divided into training and test sets (see Figure 3). In this way, the training process is performed only considering images in the training set, while images in the test set are necessary to assess the error 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 of the network have to be estimated. In our case the hyperparameter of interest is the optimal number of training epochs. This validation

TABLE I: Main characteristics of the NVIDIA GPUs used in the tests.

Model	GeForce GTX TITAN	TITAN X
Architecture	Kepler	Pascal
CUDA capability	3.5	6.1
Multiprocessors (MP)	14	28
CUDA Cores/MP	192	128
GPU Max Clock rate (GHz)	0.88	1.53
Global memory (MBytes)	6,082	12,190
L2 Cache Size (MBytes)	1.5	3

method divides the training set into k folds. For each fold k (known as *validation set*), the network is trained with all the folds but k (e.g., up to some maximum number of epochs). 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. The CNN is then trained using as input the complete training set until the number of iterations reaches the selected value.

Finally, the resulting trained CNN will be the one used for the storage format prediction. The image test set, which was part of the complete image dataset but it was not used in the training process, is utilized as input of the CNN to validate the accuracy of our classifier.

IV. Experimental Evaluation

A. Hardware platforms and software

In this work we have considered GPUs as hardware platforms to evaluate our methodology for the prediction of the best storage format when the SpMV operation is performed. However, as we commented previously, our proposal could be applied to other parallel systems. Table I shows the main characteristics of the NVIDIA GPUs used in the experimental evaluation. From now on, we use GTX and TITANX to refer to GPUs with Kepler and Pascal architecture, respectively.

For the SpMV benchmarking, we use the kernels implemented by the NVIDIA cuSPARSE library included in the CUDA toolkit version 8. CSR, HYB and ELL storage formats were studied (see Section II for details). Training the CNN was also performed using a GPU. In particular, the most powerful GPU (TITANX) was utilized with the aim of reducing the training times. NVIDIA Deep Learning GPU Training System[2] (DIGITS) was the selected software platform to carry out the training phase. DIGITS allows to design, train and visualize deep neural networks for image classification taking advantage of the deep learning framework Caffe[3]. Several of the most important CNN architectures such as LeNet, AlexNet and GoogLeNet are predefined and ready to use in the platform.

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

B. Sparse matrix dataset

As we point out in Section III, it is necessary to have a large set of sparse matrices in order to train the network. This dataset should contain matrices coming from different real problems and applications. In this way, we expect that these matrices cover a wide range of characteristics and nonzero patterns. We have created a dataset that fulfills those assumptions consisting of 8,111 sparse matrices. The dataset was generated using as basis 812 square matrices from the SuiteSparse matrix collection [10] and applying to them some transformations like cropping (similar to [11]). The main characteristics of the dataset in terms of the average, minimum and maximum values are displayed in Table II.

C. SpMV benchmarking

To train the CNN, a class (best storage format) should be assigned to matrices in the dataset. This is the goal of the SpMV benchmarking phase. We conducted experiments by measuring the performance of the single precision SpMV kernel using different storage formats (CSR, HYB and ELL) on the considered GPUs (see Table I). For each matrix and format, the performance was calculated as the average of 1,000 SpMV operations. Each matrix is then labeled according to the highest performing format. The classification results expressed as the number and percentage of matrices belonging to each class are displayed in Table III. Note that there are noticeable differences in the classification depending on the considered GPU. For example, we observe that the largest class is different on the two GPUs (ELL for the GTX and CSR for the TITANX). This dependence on the hardware platform confirms the importance and difficulty of the issue addressed in this work.

On the other hand, a bad choice of the storage format will have a negative effect on the SpMV performance. Figure 5 illustrates this behavior measuring the speedup between the best and the worst performing formats for all the matrices in the dataset. Considering the GTX platform, the boxplot shows that the median, first quartile and third quartile speedups are 1.81$\times$, 1.51$\times$ and 2.25$\times$, respectively. It means that for 50% of the matrices choosing the best storage format accelerates the SpMV operation more than 1.81$\times$. The median, first quartile and third quartile speedups for the TITANX are 1.47$\times$, 2.05$\times$ and 2.66$\times$. In addition, we have detected some outliers (points in the plots) where the SpMV is performed from tenths to hundreds of times faster when choosing the proper format. Therefore, a misprediction in the classification may lead to important performance degradations.

D. Image dataset generation & Network training

Several of the characteristics in Table II correspond to the global metrics detailed in Section III. Metrics should be normalized to fit the interval [1,255] since their values will be assigned to a RGB color channel in the images. The way this normalization is performed has an impact on the results of the classifier. As a consequence, many experiments have been carried out in order to find out the best normalization method.

TABLE II: Global metrics of the sparse matrices in the dataset.

	Avg.	Min.	Max.
Number of rows/columns (n)	153.3K	1.9K	21.2M
Nonzeros (nnz)	1.4M	120K	89.3M
Nonzeros per row (nnz_{row})	29.03	0.08	1.26K
Std. Dev. nonzeros per row (σ_{row})	27.02	0	1.81K
Density (ρ)	4.35×10^{-3}	4.08×10^{-8}	2.82×10^{-1}
Maximum nonzeros in a row (max_{row})	1.84K	1	2.31M

TABLE III: Classes of the matrices in the complete dataset (left), and only considering matrices for training (middle) and tests (right).

	Dataset		**Training set**		**Test set**	
Class	GTX	TITANX	GTX	TITANX	GTX	TITANX
CSR	2,661 [32.8%]	4,612 [56.9%]	2,128 [32.8%]	3,689 [56.9%]	533 [32.8%]	923 [56.9%]
HYB	1,882 [23.2%]	1,455 [17.9%]	1,505 [23.2%]	1,164 [17.9%]	377 [23.2%]	291 [17.9%]
ELL	3,568 [44.0%]	2,044 [25.2%]	2,855 [44.0%]	1,635 [25.2%]	713 [44.0%]	409 [25.2%]
Total	*8,111*		*6,488*		*1,623*	

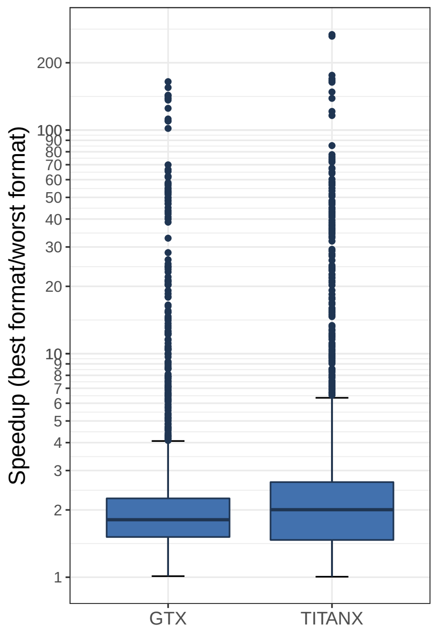

Fig. 5: Speedup obtained considering the best storage format with respect to the worst one for all the matrices in the dataset.

Next, we detail how the RGB values were calculated for the image datasets used in the evaluation (numbers identify the corresponding metric):

(0) $\lfloor \frac{n}{4000} \rfloor + 1$
(1) $nnz_{row} = \frac{nnz}{n}$ (no normalization is required)
(2) σ_{row} (no normalization is required)
(3) $\lfloor 100000 \times \frac{nnz}{n \times n} \rfloor + 1$
(4) $\lfloor \frac{max_{row}}{4} \rfloor + 1$

In case some of the previous values exceeds 255 for a particular matrix, the corresponding color in the image will be automatically fixed to 255.

We have generated and studied six different image datasets: binary image dataset (no metrics), R_1, $R_1G_2B_3$, $R_2G_3B_4$, $R_1G_3B_4$ and $R_0G_1B_4$. The size of the images is always 256×256 pixels, which corresponds to the input size for the AlexNet network. To train the AlexNet network we have used a *k-fold cross-validation* (see Section III). In particular, the dataset is split into a training set (80% of the matrices) and a test set (20% of the matrices). Table III shows the number and classes of the matrices in each set. Recall that the test set is not used in the training process. In addition, the training set was divided into 5 folds. The goal of this validation method is to figure out the optimal number of training epochs. This procedure was applied to the six image datasets and two GPUs. We have found that the optimal number of epochs ranges from 20 (binary dataset, GTX) to 42 ($R_0G_1B_4$ dataset, TITANX). We must highlight that other hyperparameters take the default values provided by the DIGITS platform.

The AlexNet network is then trained using the complete training set until the iterations reach the optimal number of epochs. Training times on the TITANX GPU vary from 6.3 minutes (binary-GTX dataset) to 14.5 minutes ($R_0G_1B_4$-TITANX dataset).

E. Prediction accuracy and performance analysis

Next, the trained CNNs for each dataset and GPU will be evaluated using only the test set. In addition to the global accuracy of the classifier (overall percentage of correct classified matrices), we provide two metrics to better understand how well the classifier is performing: *precision* and *recall*. The degree to which repeated measurements under the same conditions give us the same results is called precision. Recall or sensitivity is also known as the true positive rate, and 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

TABLE IV: Prediction accuracy of the trained network considering different image datasets on the GTX (top) and TITANX (down) platforms.

GTX	Binary		R_1		$R_1G_2B_3$		$R_2G_3B_4$		$R_1G_3B_4$		$R_0G_1B_4$	
	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
CSR	0.62	0.70	0.79	0.73	0.84	0.81	0.87	0.86	0.89	0.86	0.90	0.90
HYB	0.63	0.72	0.53	0.80	0.73	0.90	0.82	0.91	0.82	0.92	0.82	0.90
ELL	0.80	0.69	0.84	0.75	0.89	0.83	0.91	0.88	0.92	0.90	0.95	0.90
Global Accuracy	0.702		0.750		0.836		0.876		0.888		**0.901**	

TITANX	Binary		R_1		$R_1G_2B_3$		$R_2G_3B_4$		$R_1G_3B_4$		$R_0G_1B_4$	
	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
CSR	0.85	0.75	0.86	0.62	0.91	0.91	0.94	0.91	0.95	0.91	0.94	0.93
HYB	0.43	0.78	0.46	0.69	0.68	0.65	0.72	0.92	0.71	0.93	0.71	0.95
ELL	0.61	0.60	0.74	0.66	0.87	0.77	0.86	0.80	0.87	0.80	0.91	0.78
Global Accuracy	0.713		0.758		0.861		0.879		0.885		**0.890**	

and recall for class A can be calculated as $^{P_A}/_{C_A}$ and $^{P_A}/_{T_A}$, respectively.

Table IV shows the global accuracy, precision and recall of the classifiers for all the datasets on both GPUs. A noticeable accuracy of 90.1% and 89% was obtained for GTX and TITANX GPUs training the network with images whose RGB channels code information about the matrix size, the average number of nonzeros per row and the maximum number of nonzeros in a row ($R_0G_1B_4$). Good results were also observed when considering other configurations, especially $R_2G_3B_4$ and $R_1G_3B_4$. Since there are less matrices of classes HYB and ELL on the TITANX dataset (see Table III), precision and recall are lower with respect to CSR values. Finally, we must highlight that our methodology is very robust since consistent results were obtained in terms of accuracy, precision and recall for the same image datasets on both GPUs.

Another way to prove the benefits of our approach consists in measuring how close to the maximum achievable SpMV performance are the classifiers. In this way, we measure the SpMV performance for all the matrices in the test set using the format selected by each classifier. Normalized results are shown in Figure 6 in such a way that 1 corresponds to the maximum achievable performance (always choosing the best format). For instance, considering the $R_0G_1B_4$ classifier, average performances higher than 0.99 are achieved for both GPUs. It means that the difference in the SpMV performance between the best possible classification and the one obtained by our classifier is less than 1%. Therefore, considering only the global accuracy, precision and recall to evaluate the quality of classifiers for sparse matrix classification hide important information regarding the performance. Since the final objective of choosing the proper storage format is obtain the maximum SpMV performance, the above analysis is of great importance.

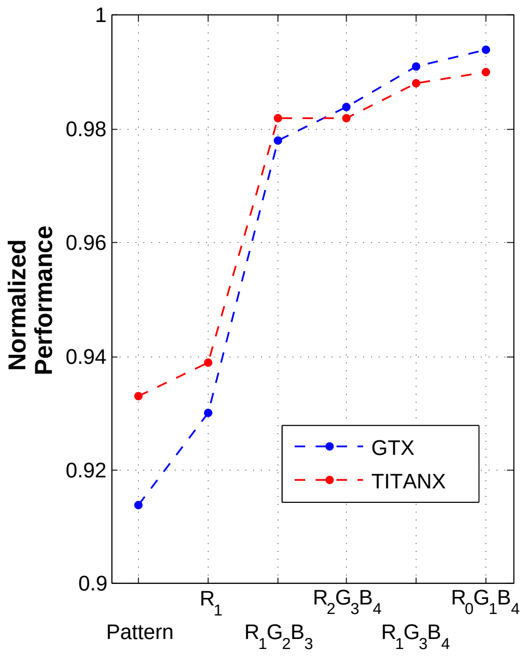

Fig. 6: Average SpMV performance obtained using the storage format selected by the classifiers.

F. Speeding up the training process

As we pointed out previously the classes of the matrices depend on the hardware platform considered in the SpMV benchmarking phase. As a consequence networks should be trained for each particular GPU. However, next we will demonstrate that it is not necessary to carry out the training process from scratch.

The idea is to consider a pre-trained model as starting point of the training process instead of considering the AlexNet network initialized with random values (i.e., training from scratch). 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 result of using this methodology very good accuracy results are obtained with less training data in comparison

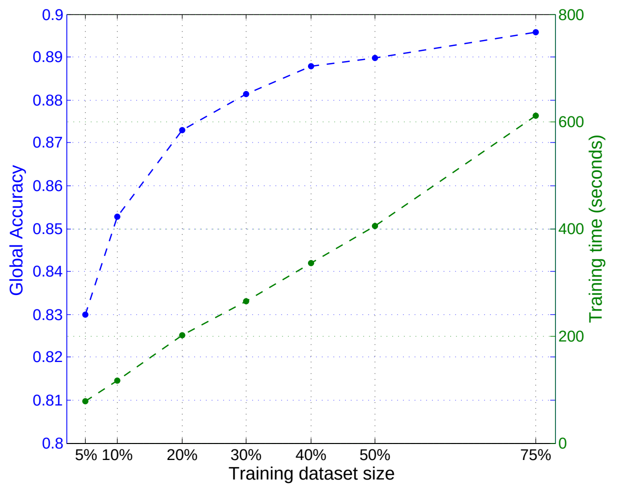

Fig. 7: Global accuracy (blue line) and times (green line) required to train a GTX classifier using as basis a pre-trained TITANX model.

to training from scratch. Therefore, since the data required to train the network decrease, training times are also lower. Figure 7 illustrates this behavior training a classifier for GTX using a pre-trained TITANX model. In this example we have considered the $R_0G_1B_4$ image dataset. For instance, we get a 88.1% and 89% accuracies training the network using only 30% and 50% of the training data, respectively.

Another important consequence of reducing the training data size is the impact on the SpMV benchmarking phase (see Figure 3). This is the most time consuming phase, requiring several hours to obtain the best storage format (class) for all the dataset on each GPU. Note that it requires to perform the SpMV operation 1,000 times for each matrix and format. In this way, it is possible to reduce noticeably the benchmarking times while achieving very good performance in terms of accuracy.

V. Related Work

We can find in the literature many analytical approaches that deal with the identification of the optimal sparse matrix format for GPUs based on performance models [12]–[14]. They show a good accuracy but models are usually tested considering a small set of matrices. Other authors use traditional machine learning approaches to select automatically the best storage format for sparse matrices. Only some of them focus on GPUs as target platforms. In [2], the authors build a decision tree to choose the best representation for a given sparse matrix based on a several matrix structure features. Their classifiers report a global accuracy in the range 64.6-83.8%, obtaining a 95% of the maximum achievable SpMV performance. A similar approach that takes advantage of support vector machines to deal with the classification problem was published in [15]. They demonstrated accuracies in the range 73-88.5%, increasing the obtained average SpMV performance up to 98%. We must highlight that the best results in both works are below the numbers obtained using our approach. In addition, the maximum accuracy observed in both works was always obtained training the classifiers using feature sets with more than three matrix properties. None of the works commented above have considered deep learning technologies. In a recent paper [11] the authors deal with the sparse matrix format selection problem using CNNs. They propose several ways to represent the matrices to train the network. Best results are achieved using histograms that capture the spatial distribution of nonzero elements in the matrix. Unlike our approach, they do not take advantage of the RGB channels of the images to code some features of the matrices. Using their representation leads the authors to create an *ad-hoc* CNN architecture. Our approach demonstrates that a simple standard CNN architecture as AlexNet is enough to provide good classification results. In this way, our methodology can be easily adopted by the research community since AlexNet is available in the most important and common deep learning frameworks. In addition, AlexNet could be easily replaced in the future by other standard architectures such as VGGNet [8] and ResNet [9] in order to improve the accuracy results or speed up the learning process.

Other papers focus only on applying machine learning techniques to multicore processors [16]. Finally, in [17] the authors propose a mechanism to select the best SpMV code implementation for both CPUs and GPUs using deep learning technologies. It is an interesting work conceptually but their approach obtains low accuracy results (only 54%) with 75% of the maximum attainable performance.

VI. Conclusions

In this work we demonstrated that deep learning technologies can be successfully applied to classification problems different from the traditional machine learning tasks. We focused on the selection of the best storage format for the SpMV kernel on GPUs. A new methodology is introduced based on the idea of considering the sparsity pattern of the matrices as an image. Coding several matrix characteristics as the RGB color of the pixels in the images, we are able to generate image datasets with enough information to successfully train a CNN. We prove that a simple trained CNN architecture as AlexNet, without any fine-tuning, achieves very good results in terms of accuracy, precision, recall and average SpMV performance. In particular, we observed a maximum global accuracy of 90.1%, obtaining within 99.4% on average of the best performance available. In addition, we demonstrate that it is possible to speed up the training process using a pre-trained model as starting point instead of training from scratch. Using a pre-trained model reduces the requirements of training data to obtain high global accuracies.

As future work we will deal with the classification problem on GPUs and other parallel architectures adding specialized storage formats [18]–[20]. In addition, we will apply dimensionality reduction techniques such as Principal Component Analysis (PCA) to code more than three matrix properties in the RGB channels of the images with the aim of improving the global accuracy of the classifiers.

References

[1] D. Langr and P. Tvrdík, “Evaluation Criteria for Sparse Matrix Storage Formats,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 428–440, 2016.

[2] N. Sedaghati, T. Mu, L.-N. Pouchet, S. Parthasarathy, and P. Sadayappan, “Automatic Selection of Sparse Matrix Representation on GPUs,” in *29th ACM on Int. Conf. on Supercomputing (ICS)*, 2015, pp. 99–108.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *25th Int. Conf. on Neural Information Processing Systems - Volume 1*, 2012, pp. 1097–1105.

[4] J. W. Choi, A. Singh, and R. W. Vuduc, “Model-driven Autotuning of Sparse Matrix-vector Multiply on GPUs,” in *15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2010, pp. 115–126.

[5] N. Bell and M. Garland, “Efficient Sparse Matrix-Vector Multiplication on CUDA,” NVIDIA Corporation, NVIDIA Technical Report NVR-2008-004, Dec. 2008.

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

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going Deeper with Convolutions,” in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[8] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[9] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *CoRR*, vol. abs/1512.03385, 2015.

[10] T. A. Davis and Y. Hu, “The University of Florida Sparse Matrix Collection,” *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1:1–1:25, 2011.

[11] Y. Zhao, J. Li, C. Liao, and X. Shen, “Bridging the gap between deep learning and sparse matrix format selection,” in *Proc. of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2018, pp. 94–108.

[12] P. Guo, L. Wang, and P. Chen, “A Performance Modeling and Optimization Analysis Tool for Sparse Matrix-Vector Multiplication on GPUs,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 5, pp. 1112–1123, 2014.

[13] B. Neelima, G. R. M. Reddy, and P. S. Raghavendra, “Predicting an Optimal Sparse Matrix Format for SpMV Computation on GPU,” in *IEEE Int. Parallel Distributed Processing Symposium Workshops*, 2014, pp. 1427–1436.

[14] K. Li, W. Yang, and K. Li, “Performance Analysis and Optimization for SpMV on GPU Using Probabilistic Modeling,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 1, pp. 196–205, 2015.

[15] A. Benatia, W. Ji, Y. Wang, and F. Shi, “Sparse Matrix Format Selection with Multiclass SVM for SpMV on GPU,” in *45th Int. Conf. on Parallel Processing (ICPP)*, 2016, pp. 496–505.

[16] J. Li, G. Tan, M. Chen, and N. Sun, “SMAT: An Input Adaptive Autotuner for Sparse Matrix-vector Multiplication,” in *34th ACM SIGPLAN Conf. on Programming Language Design and Implementation*, 2013, pp. 117–126.

[17] H. Cui, S. Hirasawa, H. Takizawa, and H. Kobayashi, “A Code Selection Mechanism Using Deep Learning,” in *IEEE MCSOC*, 2016, pp. 385–392.

[18] A. Ashari, N. Sedaghati, J. Eisenlohr, and P. Sadayappan, “An Efficient Two-dimensional Blocking Strategy for Sparse Matrix-vector Multiplication on GPUs,” in *Proc. of the 28th ACM Int. Conf. on Supercomputing*, 2014, pp. 273–282.

[19] M. Kreutzer, G. Hager, G. Wellein, H. Fehske, and A. Bishop, “A Unified Sparse Matrix Data Format for Efficient General Sparse Matrix-Vector Multiplication on Modern Processors with Wide SIMD Units,” *SIAM Journal on Scientific Computing*, vol. 36, no. 5, pp. C401–C423, 2014.

[20] D. Merrill and M. Garland, “Merge-based Parallel Sparse Matrix-vector Multiplication,” in *Proc. of the Int. Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*, 2016, pp. 58:1–58:12.