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

Juan C. Pichel and Beatriz Pateiro-López

Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS)

Universidade de Santiago de Compostela (Spain)

IEEE Cluster 2018 (Belfast, UK)

Outline

Outline

Motivation

- SpMV is a key kernel at the core of many scientific and engineering applications
- Low fractions of peak performance: attention from the research community
- Performance depends on both target architecture and sparsity structure
- Many storage formats have been proposed: big impact on performance!
- CSR *de-facto* standard for CPUs, no dominant format for GPUs

Motivation

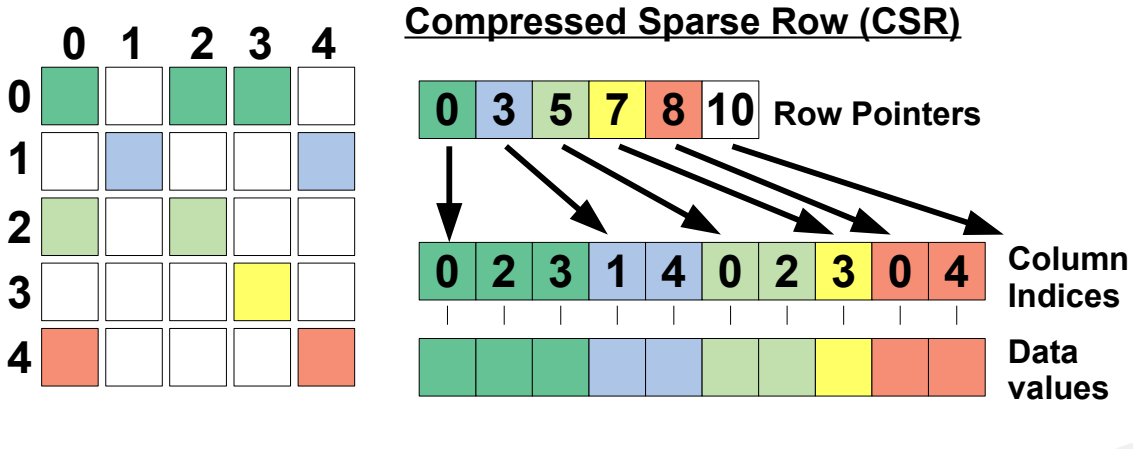
- **Automatic selection of the best storage format for sparse matrices on GPUs**
- New methodology based on deep learning technologies is introduced (Convolutional Neural Networks – CNNs)
- A simple standard CNN architecture as AlexNet is powerful enough to provide very good classification results
- Our methodology can be easily adopted by the research community
- To train the network the sparsity pattern of the matrices is considered as an image
- RGB color of pixels is used to represent properties of the matrix
- An exhaustive experimental evaluation has been carried out using two different GPUs as target platforms

Outline

Background Sparse matrix formats

- Many different storage formats: depending on the number and distribution of nonzeros
- Amount of storage required, accessing methods, adaptability to different applications or parallel architectures
- Some formats only well suited for matrices with a particular pattern: diagonal format (DIA) or block formats (e.g. BELLPACK)
- Some support efficient modification but not efficient matrix operations (COO)
- We focus on compressed sparse row (CSR), ELLPACK (ELL) and hybrid (HYB) formats: NVIDIA cuSPARSE library

Background Sparse matrix formats

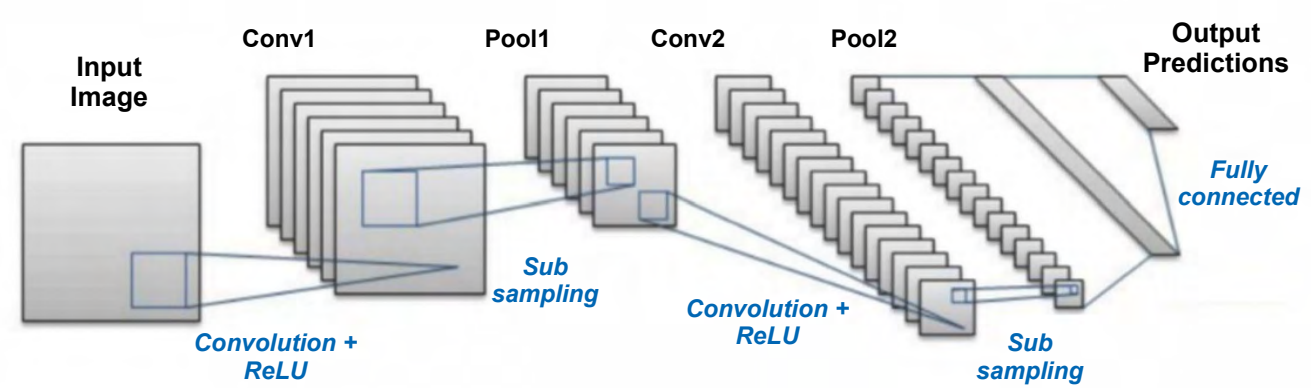


CSR: general-purpose, column indices and nonzeros in two arrays, also array of pointers (offset for each row)

ELL: compress $n \times m$ matrix in a dense $n \times k$ matrix (k = max. number of nonzeros per row). Additional $n \times k$ matrix with column indices

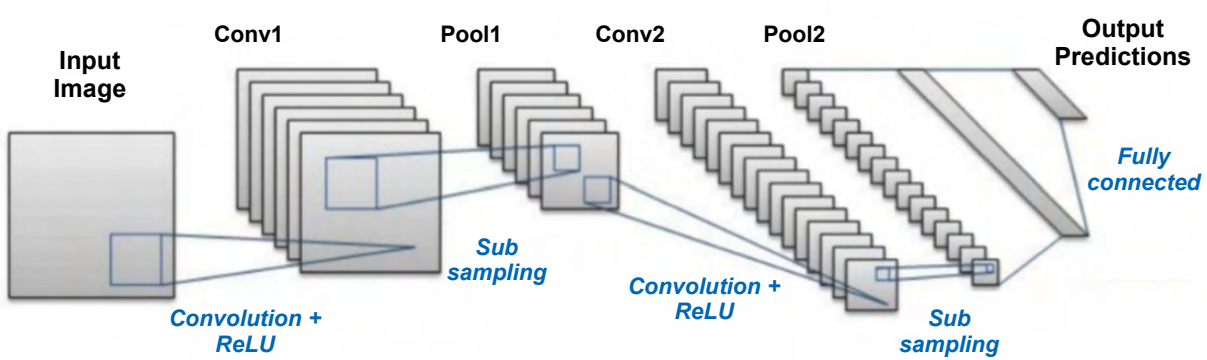
HYB: Combination COO + ELL, most of the entries in ELL format, rows with substantially different number of nonzeros (COO)

Background Convolutional Neural Networks (CNN)



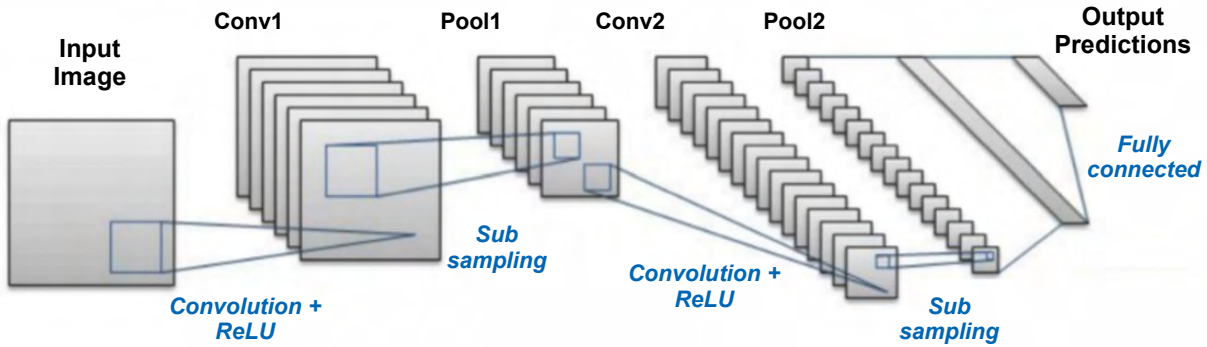
- Sequence of layers that transform the input image from the original pixel values to the final class scores
- Three types of layers: input layers, feature-extraction layers and classification layers
- Input layers load and store the input data of the image (width, height and RGB values for each pixel)

Background Convolutional Neural Networks (CNN)



- Feature-extraction layers have a general repeating pattern: convolution, non linearity (ReLU) and pooling (sub sampling)
- Convolution extracts features from the image shifting a small window (filter) across the input
- Computes the dot product between the filter and the input elements covered by the filter: 2D activation map
- The network will learn filters that activate when detect some visual feature (e.g. edges, curves)

Background Convolutional Neural Networks (CNN)



- ReLU replaces all negative values in the feature map by zero
- A pooling layer progressively reduces the spatial size of the representation decreasing the amount of parameters and computation, keeping the most important information
- Classification layer: 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

Background Convolutional Neural Networks (CNN)

Training process

1. Filters and parameters are initialized to random values
2. The network takes an input image (class/label is known *a priori*): prediction after the forward propagation step
3. A prediction error is calculated
4. By means of back propagation the network parameters are iteratively revised to minimize the overall error on each training input
5. The network is trained on the input dataset for a number of epochs (passes over the entire dataset)

Background Convolutional Neural Networks (CNN)

Training process

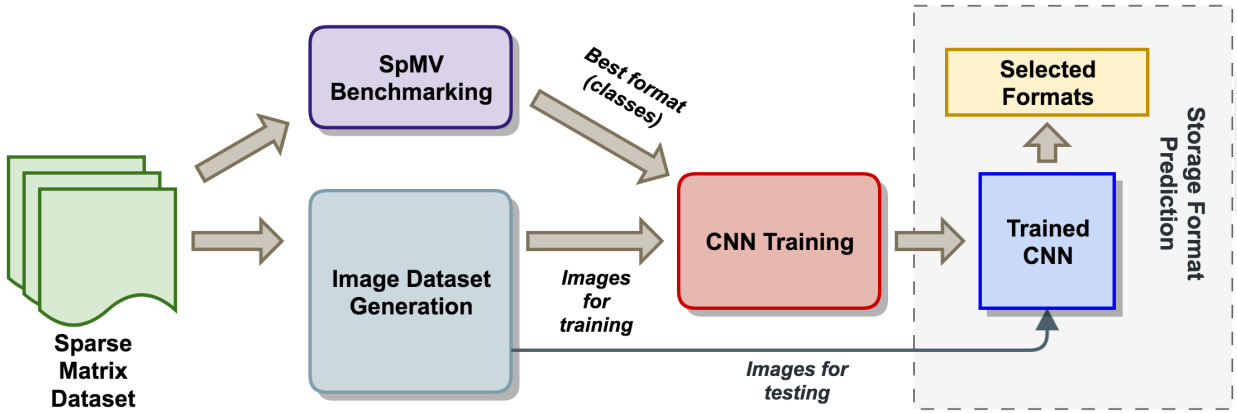
- Number of filters, filter sizes, architecture of the network: do not change
- Additional parameters (hyperparameters) such as learning rate, number of epochs: should be tuned

Background Convolutional Neural Networks (CNN)

- Many CNN architectures have been proposed
- Most popular are: LeNet, AlexNet, GoogLeNet, VGGNet and ResNet
- We have considered **AlexNet**:
  - 5 convolution layers
  - 3 pooling layers
  - 3 fully-connected layers

Outline

Methodology



SpMV Benchmarking

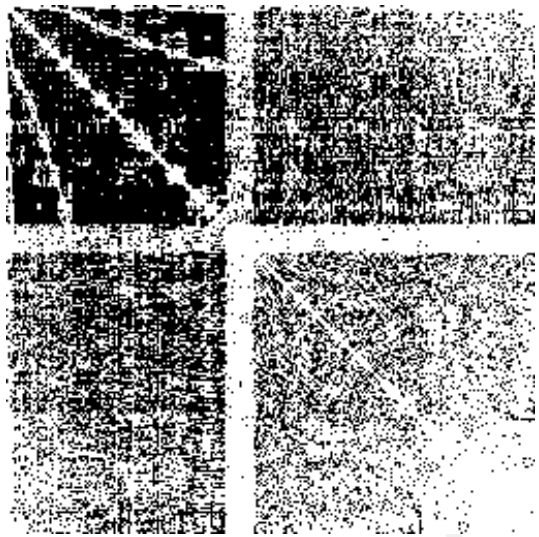
- Large set of sparse matrices is available. Input of the next phases
- SpMV performance for each matrix and storage format
- Best format in terms of performance (class/label): ground truth CNN training
- We have considered GPUs but methodology is agnostic with respect to the underlying parallel system

Methodology

Image dataset generation

- We consider the sparsity pattern of the matrices as an image
- Naive approach: $n \times m$ matrix is equivalent to a $n \times m$ binary image (problem: input size to a CNN is fixed!)
- Matrices should be scaled to the same size:
  - Let's assume that a square matrix $n \times n$ should be scaled to an image of $p \times p$ pixels, where $n > p$
  - The matrix is split into $p \times p$ submatrices
  - It will contain a nonzero at position (i, j) if there is, at least, one nonzero value in the corresponding submatrix (i, j)
  - Empty submatrix, then the corresponding entry in the scaled matrix will be zero
- Creating a $p \times p$ binary image from the scaled matrix is straightforward

Methodology



Binary image of 63×63 pixels generated from a $71,505 \times 71,505$ sparse matrix

Methodology

Image dataset generation

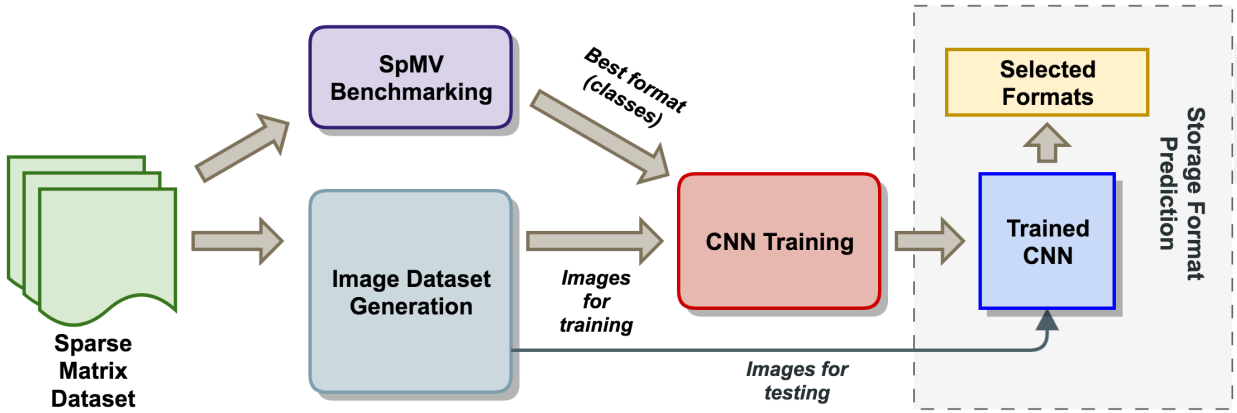
- Scaling down the matrix simplifies the appearance of the pattern
- In the example, 1 pixel corresponds to a $1,135 \times 1,135$ submatrix
- Binary images do not provide competitive results: **additional information**
- We propose to use the RGB channels to code information related to some properties of the original sparse matrix
- The following metrics have been considered:
  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. Std. deviation of the number of nonzeros per row of the matrix (σ_{row})
  3. Matrix density (ρ): 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})

Methodology

Image dataset generation

- Empty submatrices are black, that is, RGB is always (0,0,0)
- Non-empty submatrices: RGB channel within the interval $[1, 255]$ (requires normalization of the metrics)
- It is possible to use 1, 2 or 3 color channels
- Notation: $\mathbf{R}_x\mathbf{G}_y\mathbf{B}_z$ is used to indicate that metrics x, y and z correspond to R, G and B values
- We have considered the following configurations: **binary, $\mathbf{R}_1$, $\mathbf{R}_1\mathbf{G}_2\mathbf{B}_3$, $\mathbf{R}_2\mathbf{G}_3\mathbf{B}_4$, $\mathbf{R}_1\mathbf{G}_3\mathbf{B}_4$** and $\mathbf{R}_0\mathbf{G}_1\mathbf{B}_4$

Methodology



CNN training

- Input: images labeled with their class (best storage format)
- Dataset divided into training and test sets
- We have used a *k-fold cross-validation* method for model selection and assessment
- Hyperparameter of interest: optimal number of training epochs

Methodology

CNN training

- Training set divided into k folds. For each fold k, the network is trained with all the folds but k
- Global accuracy after each epoch for each validation set
- Average validation set accuracy is computed (across the k folds) for each number of epochs
- Number of epochs: maximum average accuracy
- CNN trained using the complete training set

Trained CNN

- Test set feeds the trained CNN to validate the accuracy of the classifier

Outline

Experimental evaluation

Hardware platforms and software

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

Software

- NVIDIA cuSparse library included in CUDA toolkit v8: CSR, ELL and HYB formats
- Training: most powerful GPU (TITAN X), NVIDIA Deep Learning GPU Training System (DIGITS) and the deep learning framework Caffe (AlexNet network)

Experimental evaluation

Sparse matrix dataset

- 8,111 matrices coming from different real problems and applications
- Wide range of characteristics and nonzero patterns
- Dataset generated from 812 square matrices from the SuiteSparse matrix collection (applying some transformations)

	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

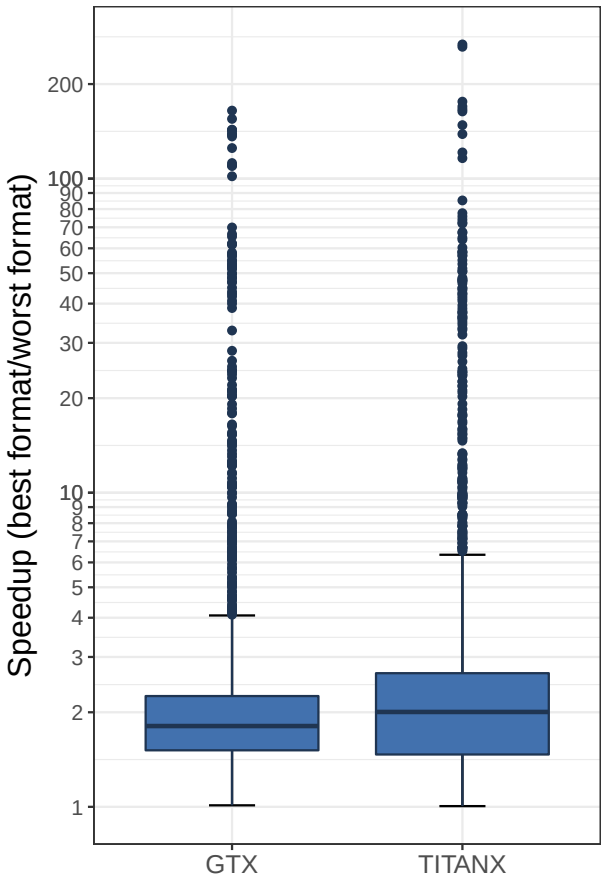
Experimental evaluation SpMV benchnmarking

- Performance of the single precision SpMV kernel using different storage formats (CSR, HYB and ELL) on the considered GPUs
- Differences in the classification between GPUs (dependence on the hardware)

	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**	

Experimental evaluation

SpMV benchnmarking



- Bad choice of the storage format: **strong impact on performance**
- Speedup between best and worst performing formats
- GTX: median, first quartile and third quartile speedups are 1.81×, 1.51× and 2.25×
- TITANX: median, first quartile and third quartile speedups are 2.05×, 1.47× and 2.66×
- Outliers (points): tenths to hundreds of times faster!!

Experimental evaluation

Image dataset generation

- Metrics should be normalized to fit [1,255] interval
- Many experiments to find out the best normalization method:
  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. $\lfloor \frac{max_{row}}{4} \rfloor + 1$
- If a value is higher than 255 –> 255
- 6 different image datasets: binary image dataset (no metrics) R_1, $R_1G_2B_3$, $R_2G_3B_4$, $R_1G_3B_4$ and $R_0G_1B_4$
- Size of the images is always 256×256 pixels (AlexNet)

Experimental evaluation CNN training

- Training set (80%) and test set (20%)
- Training set divided into 5 folds (figure out optimal number of epochs)
- Number of epochs ranges from 20 (binary dataset, GTX) to 42 ($R_0G_1B_4$ dataset, TITANX)
- Other hyperparameters take the default values provided by the DIGITS platform
- Training times of the AlexNet network vary from **6.3 minutes** (binary-GTX dataset) to **14.5 minutes** ($R_0G_1B_4$-TITANX dataset)

Experimental evaluation

Prediction accuracy

- **Global accuracy**: overall percentage of correct classified matrices
- **Precision**: degree to which repeated measurements under the same conditions give us the same results
- **Recall**: 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).
- Precision can be calculated as P_A / C_A.
- Recall for class A is P_A / T_A.

Experimental evaluation

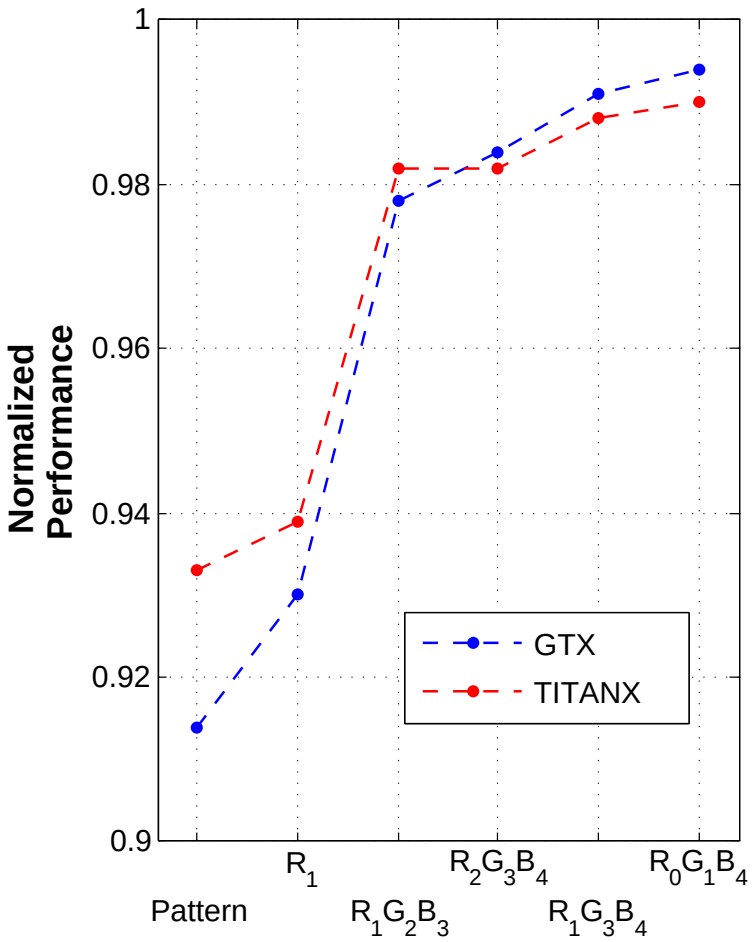
Prediction accuracy

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**	

Experimental evaluation

Prediction accuracy



- How close to the maximum achievable SpMV performance are the classifiers?
- Performance for all the matrices in the test set using the format selected by each classifier
- Always choosing the best format is 1
- For example, $R_0G_1B_4$ obtains 99% of the highest SpMV performance among the tested formats

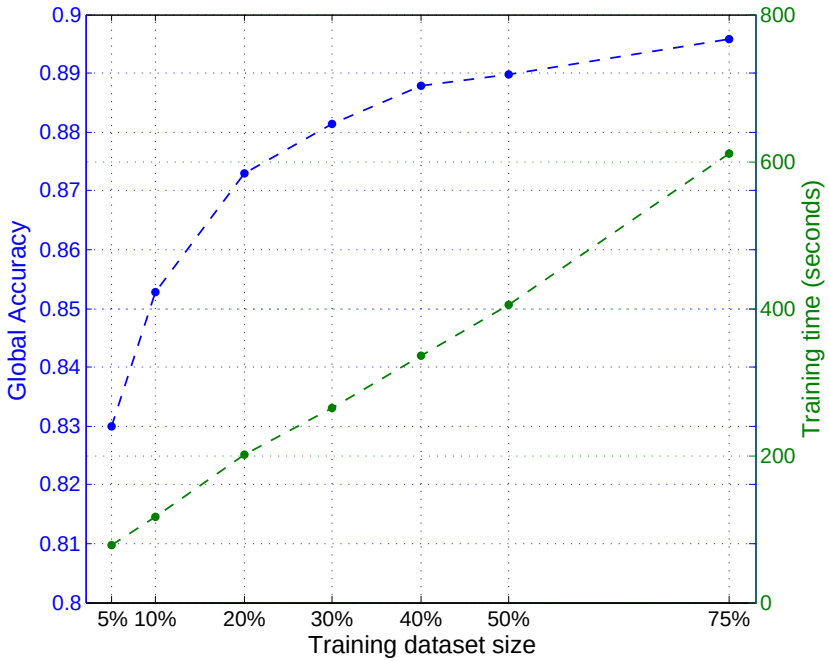
Experimental evaluation

Speeding up the training process

- Networks should be trained for each particular GPU
- Why not use a pre-trained model as starting point of the training process?
- Pre-trained model corresponds to a CNN trained for a different GPU:
  - The network inherits important characteristics and features captured (storage formats and matrices)
  - Classes differ between GPUs, but not for all the dataset

Experimental evaluation

Speeding up the training process



- Example training a classifier for GTX using a pre-trained TITANX model ($R_0G_1B_4$ image dataset)
- 88.1% and 89% accuracies using only 30% and 50% of the training data
- Impact on the SpMV benchmarking phase (most time consuming)

Outline

Conclusions

- Deep learning can be successfully applied to classification problems different from the traditional machine learning tasks
- New methodology to the automatic selection of the best storage format for sparse matrices on GPUs
- It considers the sparsity pattern of the matrices as an image, coding several matrix characteristics as the RGB color of the pixels
- A simple trained CNN architecture as AlexNet, without any fine-tuning, achieves very good results. Non *ad-hoc* architectures are necessary.
- We observed a maximum global accuracy of 90.1%, obtaining within 99.4% on average of the best performance available
- It is possible to speed up the training process using a pre-trained model

Thank you!

juancarlos.pichel@usc.es