An accurate Sparse Matrix Format Selection Method based on Deep Learning

Jihu Guo[1,2][0009−0002−9811−0234], Jie Liu[1,2], Qinglin Wang[1,2][0000−−0002−8286−6566], and Xiaoxiong Zhu[1,2]

[1] Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha 410073, China
[2] School of Computer Science, National University of Defense Technology, Changsha 410073, China
guojihu@nudt.edu.cn

Abstract. Sparse matrix-vector multiplication (SpMV) is an extremely important kernel function in scientific and engineering computing. However, the memory access-intensive and bandwidth-limited nature of SpMV leads to low execution efficiency. A proper format is crucial to improve the efficiency of the calculation, as it determines the data locality. Therefore, selecting the best format for the sparse matrix has become a hot research topic. In this paper, we present an accurate sparse matrix format selection method based on deep learning. Firstly, a novel network MixedNet is proposed for matrix classification with dual-channel input of matrix images and matrix features and is trained using a 5-fold cross-validation method on 2321 matrices. Secondly, two novel matrix image representations are designed to overcome the problem of both memory overflow and reduce matrix information loss during converting the matrix into the image representation. The experimental results show our method achieves the highest accuracy of 87.55% and the lowest FLOPS average relative loss of 3.09% compared to state-of-the-art 71.65% and 9.61% in prior works. We also analyze the time consumption caused by format selection and the results show that format selection is valuable in practice.

Keywords: SpMV · Sparse matrix format · Deep learning · Format selection

1 Introduction

Sparse matrix-vector multiplication (SpMV) is an essential kernel function and widely used in a variety of scientific and engineering computing domains [19, 24], where the efficiency of SpMV greatly affects these applications. In some applications, SpMV even dominates the processing time, such as solving linear equations systems by the Krylov subspace method and solving eigenvalue problems.

Many sparse matrix formats are designed to fit various application scenarios and hardware architectures to improve performance [21, 7, 12, 28, 3, 27, 13, 10]. In practice, the performance of SpMV using different sparse matrix formats

varies [10, 11, 14, 21, 7, 12, 28, 3]. Research papers [29, 30, 16, 20, 1, 2, 15, 4, 17, 18, 10] also point out that choosing the optimal sparse matrix format is an essential approach to improving the performance of SpMV since the format affects the cache hit rate and data locality. However, it is challenging to predict the best format. Many methods based on machine learning and deep learning methods have been proposed to perform format selection.

Li et al. [11] firstly combine the hardware platform information and matrix feature information to select matrix formats using Decision Tree, one of the traditional machine learning methods. Subsequently, other works [20, 1, 16, 2, 4] apply Decision Tree, SVM, PairwiseweightedSVM, and XGBoost to format selection. But the selection of matrix features is also challenging. First of all, the number of matrix features is innumerable since new features can be generated by combining existing ones. In addition, some matrix features are specifically used to predict specific formats and are not universal. For example, the DIA format is sensitive to the number of diagonals with at least one non-zero element.

To overcome the challenge of matrix feature selection, many works [15, 29, 17, 4, 18, 30, 10] have applied deep learning methods to extract matrix features automatically by using matrix image representation. Although these deep learning methods perform better, they also have limitations. Some matrix image representations may lead to memory overflow and subtle information loss. Besides, previous works have not explored the performance of deeper networks and they only use image representation as input which may cause matrix information loss.

Additionally, the time consumption caused by format selection is also of concern, but few works have carefully explored the various types of time consumption involved in this procedure.

In this study, we propose our novel network and matrix image representations to overcome the problems mentioned above and give a detailed analysis of the time overhead of the format selection procedure. The experimental results show that our method achieves better performance. Besides, our time overhead analysis demonstrates that it is valuable to perform SpMV with format selection. The main contributions of this paper are listed below.

- A novel dual-channel network MixedNet is proposed to predict the optimal format of sparse matrices. We achieve 87.55% accuracy and 3.09% FLOPS average relative loss (compared to state-of-the-art 71.65% accuracy and 9.61% FLOPS average relative loss [17]), on the dataset collected from SuiteSparse [5].
- We propose two new image representations to avoid memory overflows and reduce information loss during converting matrices into image representations.
- We provide a detailed analysis of the time overhead of all phases involved in the format selection which is valuable in practice.

We will introduce the sparse matrix format and format selection models in Section 2. Section 3 describes our method in detail. The result analysis and time overhead analysis are discussed in Section 4 and Section 5, respectively. Finally, we conclude the full paper in Section 6.

2 Background

2.1 Sparse matrix format

In this study, we mainly consider eight sparse matrix formats listed below. Our chosen sparse matrix formats encompass both conventional options like CSR, COO, and DIA (implemented by ourselves), as well as cutting-edge, high-performance sparse matrix formats that have been generously made open-source, such as CSR5, CVR, and SpV8 (implemented by applying open-source code provided by the authors respectively). Despite primarily conducting our experiments on the CPU platform, we deliberately incorporated the ELL-R format, originally intended for GPU implementation. Remarkably, our findings revealed that, in certain cases, the ELL-R format exhibits superior performance even on the CPU platform. Figure 1. provides a brief example of these eight sparse matrix formats.

COO format stores values, row indexes, and column indexes of the non-zero elements in *vals*, *rowIdx*, and *colIdx*.

CSR format has the same *colIdx* and *vals* with COO format. Besides, CSR format uses a *rowptr* array to mark the non-zero element's boundaries of each row.

ELL-R [23] format is an enhanced version of ELL format. ELL-R limits the maximum number of padding elements in each row, thus improving efficiency.

DIA format comprises AS, a two-dimensional array containing in each column of the non-zero elements along a diagonal of the matrix, and *offset*, an integer array that determines where each diagonal starts.

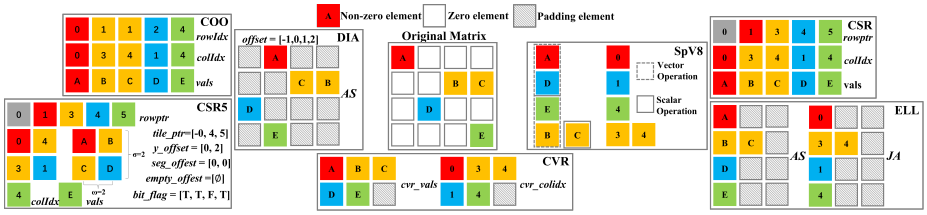

Fig. 1. Brief introduction of the matrix formats.

CSR5 [12] translates CSR into *tile*s and distributes the non-zero elements to each thread in a balanced manner and uses a segmented sum method to conduct the SpMV computation.

CVR [27] is mainly designed to implement load balance on x86 CPUs by distributing the non-zero elements to each thread as evenly as possible in behavioral units, and then adding the remaining non-zero elements to threads with fewer non-zero elements in a balanced manner according to *rec* and *tail*.

SpV8 [10] reorders the rows according to the number of non-zero elements in the rows and then divides the data into *Panel* and *Fragments* according to the SIMD width to perform SIMD operations and scalar operations respectively.

MKL-CSR [26] is provided by Intel's highly optimized scientific computing library, MKL (Math Kernel Library). MKL provides a CSR-based SpMV kernel that obtains extraordinary performance and is always regarded as the SpMV benchmark kernel.

2.2 Format Selection models

Machine learning models play an important role in the classification problem. Research papers [20, 1, 16, 2, 15] regard format selection as a classification problem. Firstly, matrix features are extracted to represent a matrix. Secondly, the model classifies the matrix by its features, and each class corresponds to a sparse matrix format.

Deep learning models have been relatively well studied in the field of image classification [8, 6, 22]. The matrix classification problem can be further transformed into an image classification problem. Zhao et al. [29] use Histogram Representation and a 7-layer two-channel CNN for classification. Some researchers represent the matrices as images and use CNN for classification [17, 4].

Compared with machine learning models, the deep learning model can automatically learn the features of the matrix, avoiding the potential difficulties encountered in manual extraction and achieving better accuracy. However, the design of network and matrix representation is also challenging.

3 Methodology

The workflow of our method is shown in Fig. 2. Firstly, a sparse matrix in CSR format is needed. Secondly, we extract the matrix features and generate image representation. Thirdly, the matrix features and image representation are fed into the pre-trained MixedNet for predicting the best format. The construction of the MixedNet is the focus of our research including how to get the matrix dataset labeled, the representation of the matrix, the network architecture of MixedNet, and the training method of MixedNet. Then, the matrix is converted into the predicted format. Finally, SpMV can be performed with the predicted format. We will introduce the detail of our method in the following subsections.

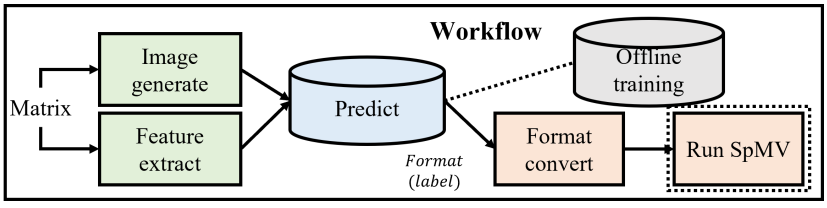

Fig. 2. The process of performing SpMV with a given matrix M.

3.1 SpMV benchmark and Matrix label

We collect 2321 matrices from SuiteSparse [5]. For each matrix, we perform 3000 SpMV on our CPU (Intel Xeon Gold 6258R) platform using each of the eight sparse matrix formats and label the matrix with the format that consumes the shortest time. Table 1 shows the number of times each format is selected as the best format.

Table 1. Number of Times Each Format Selected as The Best Format

Format	CSR5	CVR	SpV8	MKL	CSR	COO	ELL-R	DIA	Total
Number	244	492	13	1411	45	5	21	104	2321

3.2 Representation of the matrix

While the matrix can be represented using a matrix feature vector alongside the image representation, it is significant to note that the image representation is better suited for CNN. Leveraging the robust image recognition capabilities of CNN enables us to effectively perform matrix selection. In this work, we propose Multihistogram Representation and Position Representation to avoid memory overflow and store sparse patterns of the matrix. Figure 3 demonstrates an example of these two image generation methods.

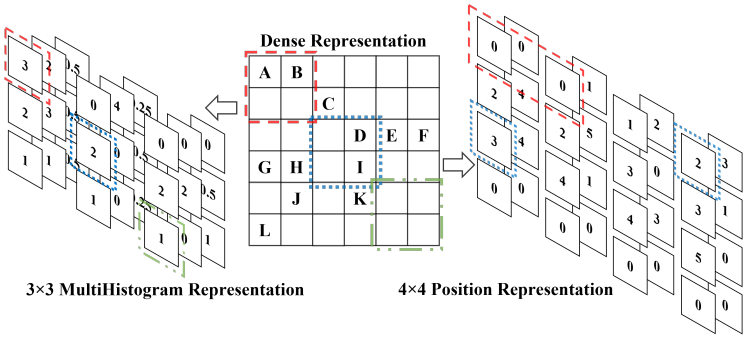

Fig. 3. Example of generating Multihistogram Representation and Position Representation. Values in image will be normalized to [0,255].

Multihistogram Representation contains the distribution information of non-zero elements in the row direction and the column direction and stores Histogram [29] and Density representations. Algorithm 1 introduces how to get the Multihistogram Representation. As shown in Fig. 3, the Dense Representation matrix is divided into 2×2 blocks ($block\_size = 2$), so we get a 3×3×3 image

(img). Here are 12 non-zero elements in the original matrix, so the $nnz = 12$. $rowIdx$ and $colIdx$ are index arrays in CSR format. The non-zero elements in the first block are A and B. As for A, the first element in the matrix, has $rowIdx[0]$=0 and $colIdx[0]$=0. Therefore, the $shift$ is $\lfloor abs(0-0)/2 \rfloor$=0 and ($block\_rowidx, block\_colidx$)=(0//2,0//2)=(0,0). Then, We set img[0][0][0]+=1, img[0][0][1]+=1 and img[0][0][2]+=1/(2×2). As for C, the index of C is (1,2). The $shift$ is $\lfloor abs(1-2)/2 \rfloor$=0 and ($block\_rowidx, block\_colidx$)=(0,1). So we set img[0][0][0]+=1, img[0][1][1]+=1 and img[0][1][2]+=1/(2×2).

Algorithm 1 Generating Multihistogram Representation

Input: $img, nnz, row, col, block\_rows, block\_cols, block\_size$

1: **for** $i = 0$ to $nnz - 1$ **do**
2: $rowidx$, $colidx$ = $row[i]$, $col[i]$
3: $shift$ = **abs**($rowidx$-$colidx$) / $block\_size$
4: $block\_rowidx, block\_colidx$ = $rowidx$ / $block\_height, colidx$ / $block\_height$
5: $img[block\_rowidx][shift][0]$ += 1
6: $img[shift][block\_colidx][1]$ += 1
7: $img[block\_rowidx][block\_colidx][2]$ += $1/(block\_size \times block\_size)$
8: **end for**
9: **return** normalize(img)

Position Representation gathers the distribution information of the elements. In some way, it detects sparse patterns by sampling like OSKI [25]. Position Representation stores the relative row and column position of the elements in two channels so that it saves the position information straightforwardly. Algorithm 2 illustrates how to generate Position Representation.

In Fig. 3, there are 12 non-zero elements in the matrix and we construct a 4×4×3 ($height$×$width$×3) image to represent the original matrix. We set nnz=min(nnz, $height$×$width$)=12 and sample nnz non-zero elements in the original sparse matrix. $rowIdx$ and $colIdx$ are index arrays of partial non-zero elements in CSR format with the length of nnz. Those non-zero elements are in row-major order. That is to say, in Fig. 3, A is the 1st element (i = 0), B is the second(i = 1), and L is the twelfth (i = 11). Considering A with index $rowIdx[0]$=0 and $colIdx[0]$=0, so the ir=0//4=0, ic=0%4=0. Then we set img[0][0][0]=1, img[0][0][1]=0, and img[0][0][2]=0. As for L with index (5,0), ir=11//4=2, ic=11%4=3. So we set img[2][3][0]=1, img[2][3][1]=5, and img[2][3][2]=0. Notice that when non-zero elements are not enough to fill up the image, we pad 0.

By using Multihistogram and Position representations, we can avoid the memory overflow problem because the generated image size is controllable and there is no necessity to recover a large 0-1 image from the original matrix. Moreover, these two methods make full use of the three channels of the image to carry the distribution information of the non-zero elements. So we can contain more in-

Algorithm 2 Generating Position Representation

Input: $nnz, rowIdx, colIdx, height, width, img$

```
size = height × width
for i = 0 to size − 1) do
    r, c = rowIdx[i], colIdx[i]
    ir, ic = i/height, i%width
    img[ir][ic][0] = 1
    img[ir][ic][1] = r
    img[ir][ic][2] = c
end for
return normalize(img)
```

formation than Density or Histogram representation and reduce the information loss when converting the matrix into an image representation.

Besides image representations, matrix features are also important for matrix representation. We refer to the matrix features in related works and select the ones that are not duplicated and cover most of the features mentioned in related works. In our experiments, we combine matrix features listed in Table 2 and image representation to better represent a matrix.

Table 2. Partial Features of a Matrix

Feature	Description	Complexity
m	The number of rows.	O(1)
n	The number of columns.	O(1)
nnz	The total number of nonzero values in the matrix.	O(1)
density	The density of the matrix, nnz/(m*n).	O(1)
mean	The mean number of nonzero values per row (npr).	O(1)
sd	The standard deviation of npr in a matrix.	O(2m)
var	The variance of npr in a matrix.	O(1)
maxnnz	The maximum of npr in a matrix.	O(m)
maxavg	The difference between maxnnz and mean.	O(1)
distavg	The mean distance between first and last nonzero values in each row.	O(m)
clusteravg	The mean of the number of distinct consecutive.	O(nnz)
ndiag	The number of matrix diagonals with one or more nonzero values.	O(nnz)
diagfill	The diagonal fill-in ratio for matrices, (m*ndiag)/nnz.	O(1)
fill	The fill-in ratio for matrices, (m*maxnnz)/nnz.	O(1)
cv	The value of sd/mean.	O(1)

3.3 MixedNet Network Architecture

A model with better accuracy can better perform format selection. In image recognition, it is effective to increase the number of network layers to improve the accuracy of the model [6]. However, the inference time increases with the depth of the network, so we need to balance the accuracy and the time consumption while designing networks. Table 3 shows the network structure of MixedNet.

Our MixedNet consists of two channels. It uses EfficientNet-B0 [22] network as the Image Channel for processing matrix images. The Feature channel uses a

Table 3. MixedNet Network Architecture

<table>
<tr><th colspan="7">MixedNet Structure (classes=C)</th></tr>
<tr><td colspan="4">Image channel (input_size=H×W)</td><td colspan="3">Feature channel (input_size=N)</td></tr>
<tr><td>Operator</td><td>Size</td><td>Channels</td><td>Layers</td><td colspan="2">Operator</td><td>Size</td><td>Layers</td></tr>
<tr><td>Conv3×3</td><td>H×W</td><td>32</td><td>1</td><td colspan="2" rowspan="2">FC</td><td rowspan="2">N×256</td><td rowspan="2">1</td></tr>
<tr><td>MBConv1,k3×3</td><td>112×112</td><td>16</td><td>1</td></tr>
<tr><td>MBConv6,k3×3</td><td>112×112</td><td>24</td><td>2</td><td colspan="2" rowspan="2">FC</td><td rowspan="2">256×512</td><td rowspan="2">1</td></tr>
<tr><td>MBConv6,k5×5</td><td>56×56</td><td>40</td><td>2</td></tr>
<tr><td>MBConv6,k3×3</td><td>28×56</td><td>80</td><td>3</td><td rowspan="4">Residule FC</td><td>FcBlock</td><td>512×512</td><td>8</td></tr>
<tr><td>MBConv6,k5×5</td><td>14×14</td><td>112</td><td>3</td></tr>
<tr><td>MBConv6,k5×5</td><td>14×14</td><td>192</td><td>4</td><td>FcBlock</td><td>512×512</td><td>8</td></tr>
<tr><td>MBConv1,k3×3</td><td>7×7</td><td>320</td><td>1</td></tr>
<tr><td>Conv1×1& Pooling</td><td>7×7</td><td>1280</td><td>1</td><td colspan="2" rowspan="2">FC</td><td rowspan="2">512×C</td><td rowspan="2">1</td></tr>
<tr><td>FC</td><td>1280×C</td><td>-</td><td>1</td></tr>
<tr><td colspan="7">Concat&FC (2×C)×C</td></tr>
</table>

fully connected network with residual connections to process the matrix features. It combines the results of the two channels for prediction. This architecture ensures both rapid inference time and satisfying accuracy.

3.4 MixedNet Training

We train our model using the 5-fold cross-validation method on an Nvidia Geforce 3090Ti with 10752 CUDA cores and 24GB memory. Training such a network is not easy, since the dataset is imbalanced as shown in Table 1. Some methods can be applied such as constructing new matrices for data augmentation [15, 4, 17, 18] to address this issue.

However, images generated from matrices are different from general images since each element in the matrix and elements' distribution information has real meaning. In our experiments, we discover that constructing a new matrix using traditional methods may result in reduced accuracy since it will disrupt the distribution of nonzero elements. Besides, these matrices are from real-world applications, but there is no guarantee that the newly generated matrices by translation, cropping, and other methods are realistic. We compare the change of accuracy of different networks using flipping or not. As shown in Table 4, the accuracy decreases for every model when applying the flipping operation for data augment.

Table 4. Accuracy Changed by Applying Flip

CNN	**Matrix Representation**		
	histogram	Multihistogram	position
MixedNet	**85.71%(-9.17%)**	**87.55%(-7.21%)**	**86.69%(-5.66%)**
Late-Merging CNN	54.91%(-7.73%)	52.03%(-15.07%)	58.89%(-1.05%)
AlexNet	71.56%(-11.80%)	69.72%(-14.15%)	67.98%(-1.85%)

To avoid these, we have oversampled the data of all formats except MKL instead of generating new matrices to alleviate the imbalance of the dataset.

4 Experimentation and Evaluation

4.1 Accuracy Comparison

The accuracy of our method is compared with that of these previous works [20, 1, 16, 2, 15, 29, 17, 18]. The result is shown in Table 5. The matrix representation used in [17, 18] can not represent all the matrices since restoring a binary image of a large matrix leads to memory overflow. We apply Histogram, Multihistogram, and Position Representation to the CNN of [17, 18] and choose the best one as its result. Previous works [20, 1, 16, 2, 15] use machine learning models and we input 15 features presented in Table 2 which containing all the matrix features mentioned in previous works [20, 1, 16, 2, 15], to their models to get results.

As shown in Table 5, deep learning models have improved accuracy than machine learning models. Our MixedNet achieved the best 87.55% accuracy with our Multihistogram Representation, which is the deepest of these networks. The method of Pichel et al. [17] achieves the second-best accuracy of 71.56% with our Position Representation. The network depth of [29], [18], [17], EfficientNet (B0), and ours increased sequentially, and the accuracy increased as the network depth increased. Moreover, the attained accuracy of the Late-Merging CNN falls short of expectations. One plausible explanation for this disparity is the limited employment of data augmentation in our experiments, which differed from the approach outlined in the original paper (refer to 3.4 for further explanation). Additionally, the relatively shallow network depth of the Late-Merging CNN might have hindered the comprehensive learning of matrix features. In summary, our model with deeper layers demonstrates a significant improvement in accuracy.

Table 5. Accuracy of Format Selection Models

Machine Learning Models	Accuracy	Deep Learning Models	Accuracy
DT[20]	61.26%	DIESEL[15]	58.57%
Multi-class SVM[1]	65.80%	Late-Merging CNN[29]	54.91%
		AlexNet[17]	71.56%
Pairwise weighted SVM[2]	54.11%	SpNet[18]	66.84%
		EfficientNet (Simplified MixedNet)	82.58%
XGBoost[16]	70.35%	**MixedNet (Ours)**	**87.55%**

4.2 Average Relative Loss

The FLOPS achieved by a sparse matrix format when performing SpMV is a crucial metric to evaluate the performance of a format [9]. Let $F(m, f)$ be the FLOPS obtained by performing SpMV with format f for matrix m. $f_{optimal}$ denotes the optimal format. Define the FLOPS average relative loss (ARL) on the dataset containing M matrices as:

$$ARL = \frac{\sum_{i=1}^{M} F(m_i, f_{optimal}) - F(m_i, f)}{\sum_{i=1}^{N} F(m_i, f_{optimal})}.$$

As shown in Fig. 4, the higher accuracy a model obtains, the lower ARL it shows, and vice versa. Our MixedNet achieves the best 3.09% ARL with Multihistogram Representation while the second best method [17] achieves 9.61% ARL with Position Representation. A model with lower ARL demonstrates that it can achieve better performance. Compared to other models using deep learning, our model shows the lowest ARL.

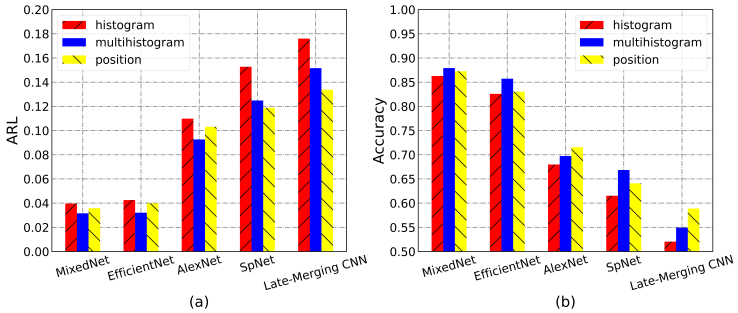

Fig. 4. FLOPS average relative loss (ARL) and accuracy of each network with Multihistogram Representation, Position Representation, and Density Representation.

4.3 Average Speedup Rate

Speedup over CSR format is another essential metric [9]. Let $T(m, f)$ be the time overhead of SpMV using format f for matrix m. Suppose the data set contains M matrices. $f_{predicted}$ denotes the predicted format of a model. Define the $AverageSpeedup$ of using a model or specific format compared to using CSR format to perform SpMV as:

$$AverageSpeedup = \frac{\sum_{i=1}^{M} T(m_i, f_{CSR})}{\sum_{i=1}^{M} T(m_i, f_{predicted})} \text{ or } \frac{\sum_{i=1}^{M} T(m_i, f_{CSR})}{\sum_{i=1}^{M} T(m_i, f)}.$$

As shown in Table 6, we measured the $AverageSpeedup$ of performing SpMV with four deep learning models and performing SpMV with seven formats compared to CSR format, respectively. According to the results, our method achieves the highest $AverageSpeedup$.

Compared to implementing SpMV with a single format and other models, our method achieves the best $AverageSpeedup$. Table 6 also demonstrates that using a deep learning model improves the performance of SpMV relative to using a single format.

5 Time Overhead Analysis

Format selection requires additional time overhead including inference, feature extraction, format conversion, and image generation time overhead. These parts

Table 6. AverageSpeedup Over CSR Format on Dataset

Technique	Type	AverageSpeedup	Technique	Type	AverageSpeedup
MixedNet (Ours)	**CNN**	**1.5087x**	MKL-CSR	Single Format	1.2838x
			CSR5	Single Format	1.1484x
AlexNet	CNN	1.4517x	CVR	Single Format	1.1161x
			CSR	Single Format	1.0000x
SpNet	CNN	1.4081x	SpV8	Single Format	0.7698x
			COO	Single Format	0.6810x
Late-Merging CNN	CNN	1.3586x	ELL-R	Single Format	0.6782x
			DIA	Single Format	0.0145x

of time are usually neglected and few works have given a detailed analysis. In this section, we prove that format selection is efficient while we need to implement format selection according to our situation.

Let $T(m_i, f)$ be the time overhead of SpMV using format f for matrix m_i. We normalize all the time overhead to $CSRAvgTime$ in the following discussion and the CSR-based SpMV average time on the dataset ($CSRAvgTime$) can be denoted as:

$$CSRAvgTime = \frac{1}{M}\sum_{i=1}^{M} T(m_i, CSR).$$

The inference process is performed with the same GPU we used for training our model. Let $T(m, model)$ be the time to infer the format of matrix m through $model$. M represents the number of matrices in the dataset. The average inference time overhead $AvgInfTime$ is expressed as:

$$AvgInfTime = \frac{1}{M}\sum_{i=1}^{M} T(m_i, model).$$

Inference Time Overhead in Table 7 shows the $AvgInfTime$ of each model. We implement Pytorch and TensorRT for inference. After applying TensorRT, $AvgInfTime$ of MixedNet is reduced to 13.7815 times $CSRAvgTime$. This indicates that the inference procedure can be significantly efficient.

Feature Extraction Time Overhead in Table 7 shows the time consumed for extracting various features. We discover that computing *clusteravg*, *ndiag* demands considerable time since these features demand a time complexity of $O(nnz)$. Therefore we should weigh whether certain features are worth extracting when features increase.

Format Conversion Time Overhead in Table 7 shows the average time to convert the matrix from CSR format to each of the other seven formats. The default format of the 2687 matrices obtained from SuiteSparse [5] is COO. Among them, 2475 matrices can perform conversions from CSR to 7 other formats, and the rest of the matrices cannot satisfy all format conversion operations at the same time due to data and memory problems, so we counted the average time to convert 2475 matrices to COO, MKL, CSR5, SpV8, ELL-R, CVR. The DIA format is often not recommended when the parameter *ndiag* is too large, so we counted the average time to convert 367 matrices with *ndiag* less than 500 to DIA format.

Image Generation Time Overhead in Table 7 shows the average time to convert 2687 matrices into four types of matrix images.

Table 7. Summary of Time Overhead

Inference Time Overhead							
Network	AlexNet	SpNet	Late-Merging CNN	EfficientNet	**MixedNet**		
TensorRT Time Overhead	11.4450x	9.2716x	6.8994x	13.4092x	**13.7815x**		
Pytorch Time Overhead	18.8093x	13.4227x	10.4160x	31.1784x	**35.1446x**		
Feature Extraction Time Overhead							
Selected Features	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15			1,2,3,4,5,6,7,8,9,10,14,15			
Time Overhead	62.77x			1.77x			
Format Conversion Time Overhead							
Transform To Format	COO	DIA	MKL	CSR5	SpV8	ELL-R	CVR
Time Overhead	0x	1.932x	7.331x	22.247x	32.662x	57.09x	226.273x
Image Generation Time Overhead							
Image Representation	Density	Histogram	**Multihistogram**	**Position**			
Time Overhead	46.734x	60.294x	**74.127x**	**67.325x**			

For a matrix stored in the CSR format, the time overhead to perform N times SpMV can be expressed as $T_{SpMV}(CSR, N)$. When performing SpMV using the

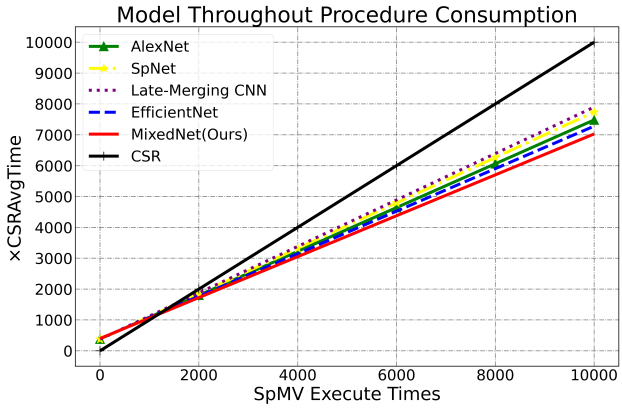

Fig. 5. The total time consumption of using format selection method for SpMV.

predicted format of the neural network, the time overhead $T_{SpMV}(model, N)$ can be expressed as follow:

$$T_{SpMV}(model, N) = T_{feature}(m) + T_{img}(m, img_{type}) + T_{inf}(model, m) + T_{con}(CSR, f_p) + T_{SpMV}(f_p, N).$$

Let $T_{feature}(m)$ (feature extraction time overhead) and $T_{img}(m, img_{type})$ (image generation time overhead) take the values when the model achieves the highest accuracy, 1.77 and 74.127, respectively. Let $T_{con}(CSR, f_p)$ (format conversion time overhead) be 226.273. $T_{inf}(model, m)$ (model inference time overhead) takes corresponding $AvgInfTime$. $T_{SpMV}(f_p, N)$ is according to the re-

sult of the SpMV benchmark run. The variation of the time taken to plot SpMV with the number of executions is shown in Fig. 5 according to $T_{SpMV}(model, N)$. Compared with the direct SpMV using the default format, the format selection method requires a series of preprocessing. When N is small, these preprocessing overheads take up most of the time of the entire SpMV process. In the contrast, it shows better performance to use format selection as N increases. We can apply format selection according to our situation.

6 Conclusion

In this paper, we propose a deep learning-based sparse matrix format selection method that takes both image representation of the matrix and matrix features as inputs to the MixedNet and predicts the best format. We also design two novel matrix image representations, *Multihistogram* and *Position*, which have much lower memory consumption and contain more matrix information compared to the related works. We achieved a superior prediction accuracy of 87.55% and an average relative loss of 3.09% compared to the state-of-the-art with 71.56% and 9.61%. Finally, we provide a detailed analysis of the time overhead of applying format selection which is meaningful for relative studies and applications. The test results show that our model is more efficient than others and format selection is important in practice.

References

1. Benatia, A., Ji, W., Wang, Y., Shi, F.: Sparse matrix format selection with multiclass svm for spmv on gpu. In: 2016 45th International Conference on Parallel Processing (ICPP). pp. 496–505. IEEE (2016)
2. Benatia, A., Ji, W., Wang, Y., Shi, F.: Bestsf: a sparse meta-format for optimizing spmv on gpu. ACM Transactions on Architecture and Code Optimization (TACO) **15**(3), 1–27 (2018)
3. Choi, J.W., Singh, A., Vuduc, R.W.: Model-driven autotuning of sparse matrix-vector multiply on gpus. ACM sigplan notices **45**(5), 115–126 (2010)
4. Cui, H., Hirasawa, S., Kobayashi, H., Takizawa, H.: A machine learning-based approach for selecting spmv kernels and matrix storage formats. IEICE TRANSACTIONS on Information and Systems **101**(9), 2307–2314 (2018)
5. Davis, T.A., Hu, Y.: The university of florida sparse matrix collection. ACM Transactions on Mathematical Software (TOMS) **38**(1), 1–25 (2011)
6. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
7. Kourtis, K., Karakasis, V., Goumas, G., Koziris, N.: Csx: an extended compression format for spmv on shared memory systems. ACM SIGPLAN Notices **46**(8), 247–256 (2011)
8. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Communications of the ACM **60**(6), 84–90 (2017)
9. Langr, D., Tvrdik, P.: Evaluation criteria for sparse matrix storage formats. IEEE Transactions on parallel and distributed systems **27**(2), 428–440 (2015)

10. Li, C., Xia, T., Zhao, W., Zheng, N., Ren, P.: Spv8: Pursuing optimal vectorization and regular computation pattern in spmv. In: 2021 58th ACM/IEEE Design Automation Conference (DAC). pp. 661–666. IEEE (2021)
11. Li, J., Tan, G., Chen, M., Sun, N.: Smat: An input adaptive auto-tuner for sparse matrix-vector multiplication. In: Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation. pp. 117–126 (2013)
12. Liu, W., Vinter, B.: Csr5: An efficient storage format for cross-platform sparse matrix-vector multiplication. In: Proceedings of the 29th ACM on International Conference on Supercomputing. pp. 339–350 (2015)
13. Liu, W., Vinter, B.: Speculative segmented sum for sparse matrix-vector multiplication on heterogeneous processors. Parallel Computing **49**, 179–193 (2015)
14. Merrill, D., Garland, M.: Merge-based sparse matrix-vector multiplication (spmv) using the csr storage format. Acm Sigplan Notices **51**(8), 1–2 (2016)
15. Mohammed, T., Albeshri, A., Katib, I., Mehmood, R.: Diesel: A novel deep learning-based tool for spmv computations and solving sparse linear equation systems. The Journal of Supercomputing **77**(6), 6313–6355 (2021)
16. Nisa, I., Siegel, C., Rajam, A.S., Vishnu, A., Sadayappan, P.: Effective machine learning based format selection and performance modeling for spmv on gpus. In: 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 1056–1065. IEEE (2018)
17. Pichel, J.C., Pateiro-López, B.: A new approach for sparse matrix classification based on deep learning techniques. In: 2018 IEEE International Conference on Cluster Computing (CLUSTER). pp. 46–54. IEEE (2018)
18. Pichel, J.C., Pateiro-Lopez, B.: Sparse matrix classification on imbalanced datasets using convolutional neural networks. IEEE Access **7**, 82377–82389 (2019)
19. Ravishankar, M., Dathathri, R., Elango, V., Pouchet, L.N., Ramanujam, J., Rountev, A., Sadayappan, P.: Distributed memory code generation for mixed irregular/regular computations. In: Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pp. 65–75 (2015)
20. Sedaghati, N., Mu, T., Pouchet, L.N., Parthasarathy, S., Sadayappan, P.: Automatic selection of sparse matrix representation on gpus. In: Proceedings of the 29th ACM on International Conference on Supercomputing. pp. 99–108 (2015)
21. Su, B.Y., Keutzer, K.: clspmv: A cross-platform opencl spmv framework on gpus. In: Proceedings of the 26th ACM international conference on Supercomputing. pp. 353–364 (2012)
22. Tan, M., Le, Q.: Efficientnet: Rethinking model scaling for convolutional neural networks. In: International conference on machine learning. pp. 6105–6114. PMLR (2019)
23. Vázquez, F., Fernández, J.J., Garzón, E.M.: A new approach for sparse matrix vector product on nvidia gpus. Concurrency and Computation: Practice and Experience **23**(8), 815–826 (2011)
24. Venkat, A., Hall, M., Strout, M.: Loop and data transformations for sparse matrix code. ACM SIGPLAN Notices **50**(6), 521–532 (2015)
25. Vuduc, R., Demmel, J.W., Yelick, K.A.: Oski: A library of automatically tuned sparse matrix kernels. In: Journal of Physics: Conference Series. vol. 16, p. 521. IOP Publishing (2005)
26. Wang, E., Zhang, Q., Shen, B., Zhang, G., Lu, X., Wu, Q., Wang, Y., Wang, E., Zhang, Q., Shen, B., et al.: Intel math kernel library. High-Performance Computing on the Intel® Xeon Phi™: How to Fully Exploit MIC Architectures pp. 167–188 (2014)

27. Xie, B., Zhan, J., Liu, X., Gao, W., Jia, Z., He, X., Zhang, L.: Cvr: Efficient vectorization of spmv on x86 processors. In: Proceedings of the 2018 International Symposium on Code Generation and Optimization. pp. 149–162 (2018)
28. Xie, Z., Tan, G., Liu, W., Sun, N.: Ia-spgemm: An input-aware auto-tuning framework for parallel sparse matrix-matrix multiplication. In: Proceedings of the ACM International Conference on Supercomputing. pp. 94–105 (2019)
29. Zhao, Y., Li, J., Liao, C., Shen, X.: Bridging the gap between deep learning and sparse matrix format selection. In: Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming. pp. 94–108 (2018)
30. Zhou, W., Zhao, Y., Shen, X., Chen, W.: Enabling runtime spmv format selection through an overhead conscious method. IEEE Transactions on Parallel and Distributed Systems **31**(1), 80–93 (2019)