COMPRESSIVE SENSING BASED CONVOLUTIONAL NEURAL NETWORK FOR OBJECT DETECTION

Yirui Wu[1], Zhouyu Meng[2], Shivakumara Palaiahnakote[3], Tong Lu[4*]

[1]College of Computer and Information, Hohai University, 210098 Nanjing, China

[1,2,4]National Key Lab for Novel Software Technology, Nanjing University, 210093 Nanjing, China

[3]Faculty of Computer Science and Information Technology, University of Malaya,50603 Kuala Lumpur, Malaysia

Email: wuyirui@hhu.edu.cn[1], misskanagi@gmail.com[2], shiva@um.edu.my[3], lutong@nju.edu.cn[4] (corresponding author)

DOI: https://doi.org/10.22452/mjcs.XXXXX

ABSTRACT

Deep neural networks (DNN) have shown significant performance in several domains including computer vision and machine learning. Convolutional Neural Networks (CNN), known as a particular type of DNN, have shown their promising potentials in discovering vision-based patterns from quantity of labeled images. Many CNN-based algorithms are thus proposed to solve the problem of object detection and object recognition. However, CNN-based systems are hard to deploy on embedded systems due to their computationally and storage intensive. In this paper, we propose a method to compress convolutional neural network to decreases its computation and storage cost by exploiting inherent redundancy property of parameters in different kinds of layers of CNN architecture. During the compression, we firstly construct parameter matrices from different kinds of layers and convert parameter matrices to frequency domain through discrete cosine transform (DCT). Due to the smooth property of parameters when processing images, the resulting frequency matrices are dominated by low-frequency components. We thus prune high-frequency part to emphasize the dominating part of frequency matrix and make the frequency matrix sparse. Then, the sparse frequency matrices are sampled with distributed random Gaussian matrix under the guiding of compress sensing. Finally, we retrain the network with the sampling matrices to fine-tune the remaining parameters. We evaluate the proposed method on several typical convolutional neural network and show it outperforms one latest compression approach.

Keywords: Deep compression, Convolutional neural network, Discrete cosine transform, Compressive sensing, YOLO, Object detection.

1.0 INTRODUCTION

Convolutional neural networks have shown reliable results on object recognition [1,2] and detection [3,4,5,6,7] that are useful in real world applications. By involving Convolutional Network as architectures for feature extraction and classifier learning, YOLO [6,7], one state-of-the-art object detection system, is reported to achieve 76.8 mean Average Precision (mAP) on VOC 2007 dataset [8]. The result achieved by YOLO outperforms most of the typical object detection methods including Deformable part-based model (DPM) [9].

However, it's hard to deploy CNN-based systems as applications in embedded systems due to its computational and storage intensity. For example, the complex CNN structure behind YOLO makes its size over 230MB, storing $6.74*10^8$ parameters extracted from 30 layers. Running a system with such a tremendous number of parameters consumes large storage and computational resources, which are often limited in embedded systems. This problem occurs for other CNN-based systems as well. Therefore, how to appropriately compress parameters of CNN has become one of the most challenging topics in machine learning community.

In this paper, we propose a novel approach to compress parameters of convolutional neural network with frequency pruning and compressive sensing. The key idea of the proposed method is to represent the parameter matrix as a discrete signal and apply spectral analysis to compress it. Due to the nature of local pixel correlation in images (i.e. spatial locality), we observe the parameters in one layer of CNN tend to be smooth. From the view of spectral

analysis, the low-frequency part referring to the smoothness will dominate the signal transformed from parameter matrix. Therefore, we could prune high-frequency part for compression. After pruning, we achieve a sparse parameter signal. Its sparse property encourages us to further decrease the storage consumption by sampling the parameter signal with distributed random Gaussian matrix under the guiding of compressive sensing.

Compressive sensing is a signal processing technique by exploring the sparsity of a signal to recover it from fewer samples than required by the Shannon-Nyquist sampling theorem. In fact, a common goal of signal processing is to reconstruct a signal from a series of sampling measurements. Once the size of sampling measurements is smaller than that of the original signal, the original signal could be compressed well. An early breakthrough for signal reconstruction is the Nyquist–Shannon sampling theorem, which states that if the signal's highest frequency is less than half of the sampling rate, the signal can be reconstructed perfectly by means of linearly interpolation. Improving the idea of Nyquist–Shannon sampling theorem by exploring prior knowledge about the signal, compressive sensing focuses on sparse signals in the sense that there exists a basis where coefficients of matrix could have just a few large values and many small values. Only utilizing large values for reconstruction would greatly reduce the size of sampling measurements, which is much smaller than required by the Shannon-Nyquist sampling theorem. Considering frequency matrix after pruning as a sparse signal, the proposed method thus utilizes compressive sensing for a higher compression rate by adopting the distributed random Gaussian matrix as sampling basis.

Our approach has two major contributions:

- Introduction of a compression method for CNN network based on compressive sensing. Compressive sensing is designed to utilize sparsity of signals for compression under the theoretical guidance of sparse coding. By transforming parameter matrices from spatial to frequency domain and pruning high-frequency part, the resulting signal are sparse enough to utilize compressive sensing for compression. To the best of our knowledge, this is the first work compressing CNN network with compressive sensing.
- Construction of a novel object detection system (nearly 100MB), which is much smaller in size than YOLO system and supports to be deployed in an embedded system. Essentially, quantity of CNN-based methods have been proposed for object detection [1-7]. However, such methods are hard to be directly deployed on a system with limited computation resource, *e.g.*, embedding system, due to the high request on computation and memory resource of CNN-based methods. We thus build our work on one of the current state-of-the-art object detection systems, *i.e.* YOLO, and propose compressive sensing based deep compression to decrease the computation cost and memory size requirement of YOLO, in order to be deployed on embedding systems. Practically, the proposed method could be easily applied to any CNN-based system for compression.

The rest of the paper is organized as follows. Section 2 reviews the related work. Details of the proposed method are discussed in Section 3. Section 4 presents the experimental results and discussions. Finally, Section 5 concludes the paper.

2.0 RELATED WORK

Deep neural networks have demonstrated the incredible power to learn high-level representations for classifying different categories of objects. However, these architectures are hard to shift to embedded systems due to the limited computational and memory resource. Early, researchers estimate a deep neural network with a shallower model to reduce the size of a network. Cybenko *et al.* [10] shows that a network with a large enough single hidden layer of sigmoid units can approximate any decision boundary. Following Cybenko's idea, Dauphin *et al.* [11] trains a shallow network on SIFT features to classify the ImageNet dataset. However, it is difficult to train shallow networks with large number of parameters. It's noted that the redundancy of Deep Neural Networks guarantees that reducing the number of parameters reasonably will not affect the accuracy of the network. Therefore, it's possible to compress deep neural networks with guaranteed accuracy. Based on the timing for compression, we generally category current compression methods into two groups: compress network during and after training.

The methods in the first group try to compress weights, activations and gradients during training to obtain smaller and faster network. Researchers find the sensitivity increases in turn in the order of parameters, activation values and gradients. Based on this observation, Courbariaux *et al.* [12] train binarized neural networks (BNNs) with binary weights and activations, which achieves a high compression result on multiple datasets. Rastegari *et al.*[13]

further explore the idea of binarization by introducing XNOR-Networks. By defining the optimal scaling factor as the average of absolute weight values, they adjust the loss of binarazation operation to achieve a higher accuracy and compression rate over BNNs. Most recently, He *et al.* [14] compress CNN networks by first detecting representative neurons with lasso regression model and then only using the representative neurons to reconstruct the whole network, which achieves an impressive compression result. However, these methods require compressing during training from scratch, which greatly increase computation cost of compression for the existed deep neural models.

The other kind of methods focuses on compressing the pre-trained network. Pruning redundant, non-informative weights in a pre-trained network could reduce the size of the network at running time. Following this idea, Dendi *et*

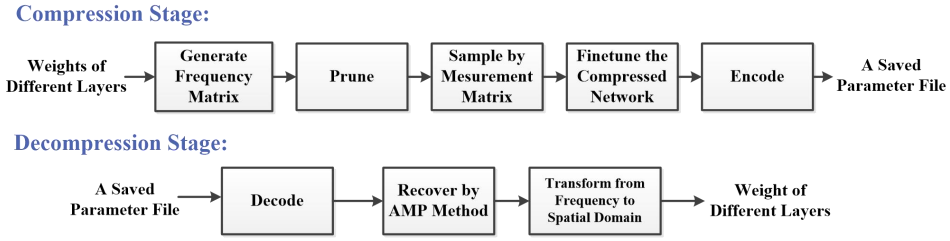


Fig. 1: The proposed method consists of two stages: compression and decompression

al. [15] use low-rank decomposition of the weight matrix to reduce the number of parameters in the network. In a similar way, Gong *et al.* [16] investigate information theoretical vector quantization to compress the parameters of CNNs, which has a clear gain over parameter matrices of existing neural network and leads to a very good balance between model size and recognition accuracy. Recently, Han *et al.* [17] propose to reduce the number of parameters by keeping only the important connections. They further explore the idea into a three-stage pipeline [18] to reduce the storage requirement of networks without affecting their accuracy. They firstly remove the redundant connections, then trained quantization so that multiple connections share the same weight and finally use Huffman coding to generate the compressed file. After that, Chen *et al.* [19] introduce DCT and low-cost hash function to reduce model size by transforming and randomly grouping the learning weights. Due to the lack of analysis of sparse property, we believe the compression rate of [19] could be improved by introducing technologies from compressive sensing.

3.0 METHODOLOGY

In this section, we propose a novel method to compress convolutional neural network, which decreases the storage cost and retain the detection accuracy simultaneously. Fig.1 gives the overview of the proposed method.

During the compression stage, the proposed method firstly constructs the parameter matrices from weights of different layers. Regarding the parameter matrices as signals, the proposed method transforms them to frequency domain by discrete cosine transform (DCT). Since the low-frequency part dominates the resulting frequency matrices, the proposed method prunes the high-frequency part. Next, the proposed method samples the parameter matrices by defining distributed random Gaussian matrix as measurement matrix. After sampling, the proposed method retrains the network to fine-tune the remaining connections and parameters. Finally, we utilize Huffman code to encode the parameter matrices into a binary parameter file, which will be saved for further decompression stage.

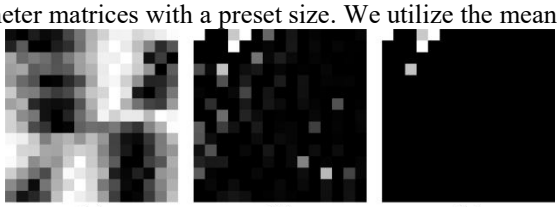
During the decompression stage, we firstly decode the binary parameter file to sampling matrices. Then we recover them to frequency matrices by the AMP method. Finally, we transform the resulting matrices from frequency domain to spatial domain by inverse form of DCT. Note that the decompression stage is performed at run-time, while the compression stage is required to be performed after training and completed with one time.

3.1 DCT for Parameter Matrix

Typical CNN are composed of convolutional layers (COV), batch normalization layers (BN), max-pooling layers (MP) and so on. The parameters of MPs and other layers are determined manually, while parameters in COVs and BNs require training process to determine. In addition, the number of parameters in COVs and BNs is much larger

than parameters of MPs and other layers. The proposed method intends to compress parameters of COVs and BNs for pre-trained CNN models.

We firstly construct parameter matrices based on parameters of COVs and BNs. Generally, each normalization layer follows a convolutional layer to improve convergence. Take YOLO as an example, it has 22 COVs and an equal number of BNs. The size of parameters of each COV varies greatly from hundreds to millions. To handle the variability of COV's size, we propose to separate them into blocks with a preset size *d*, which is determined as 15*15 by experiments. Since each BN owns only three parameters, BNs have a much smaller number of parameters than COVs. We thus adopt several matrices with size *d* to store the parameters of BNs. It's noted that there would be blanks when constructing parameter matrices with a preset size. We utilize the mean value of each matrix to fill up



(a) (b) (c)

Fig. 2: An example of a parameter matrix in spatial domain(a), frequency domain (b) and after pruning (c).

blanks inside the block, since the mean value won't greatly alter the frequency distribution achieved in latter steps. After separating blocks and filling blanks, we represent the parameter matrix as $N_{l,k}$, where *l* and *k* refer to the layer and the index inside the layer, respectively.

Next, we utilize DCT to transform the parameter matrix from spatial domain to frequency domain. DCT-based methods are widely used to compress images and movies [20]. The main reason for its popularity in compression area lies in three aspects: 1. DCT has the ability to pack most energy of images to the low-frequency part; 2. DCT and its inverse operation could be lossless for compression when employed without quantization or other compression operation; 3. DCT yields a real-valued frequency matrix, compared with FFT whose representation has imaginary components. Therefore, it's appropriate to apply DCT for transforming and compressing. Essentially, A discrete cosine transform (DCT) could expresses a matrix in terms of a sum of cosine functions oscillating at different frequencies. Given the cosine functions *A(i,j)*, we thus compute the frequency matrix $M_{l,k}$ by DCT as follows:

$$M_{l,k} = AN_{l,k}A^T, where\ A(i,j) = c(i)\cos\left[\frac{(j+0.5)\pi}{d}i\right] \quad (1)$$

where *d* refers to the size of the input parameter matrix $N_{l,k}$, *i* and *j* are the row and column index of the input matrx respectively, $c(i) = \sqrt{\frac{1}{K}}$ when $i = 0$ and $c(i) = \sqrt{\frac{2}{K}}$ when $i \neq 0$ which consists of the matrix orthogonal basis. During the decompression stage, we use the inverse DCT to transform parameter matrix from frequency domain to spatial domain, which is defined as $N = A^{-1}M(A^T)^{-1}$.

3.2 Frequency Pruning

Existing compression methods designed for convolutional neural network, such as the Deep Compression algorithm [10], ignored the procedure of transforming weights into sparse field for pruning, leading to loss of information during pruning. The proposed method transforms the weights into frequency domain, which is sparse and dominated by low-frequency part. We thus could get less loss of information by pruning high-frequency part. In addition, the compression rate of the latter step, i.e. compressive sensing, is highly related with the sparsity of the parameter matrix. Pruning could make the parameter matrix sparser.

In Fig. 2 (a) and (b), we show an example of parameter matrix in spatial domain and frequency domain, respectively. We could notice the parameter matrix in the spatial domain is smooth, due to the property of spatial locality of image pixels. Meanwhile, the upper left part of the parameter matrix in the frequency domain, known as low-frequency components, have larger magnitude values than other parts named as high-frequency components, due to the smoothness of the values in the parameter matrix. Based on these observations, we could conclude the energy of the parameter matrix is dominated by its low-frequency part. In other words, the upper left values are

more valuable than others in constructing filters of CNN. To decrease storage cost and maintain accuracy for CNN, we should prune the high-frequency part and retain the low-frequency part.

Essentially, network pruning has been widely studied to compress CNN models. Early pruning methods [21] were proposed to reduce the network complexity and over-fitting. The proposed method builds on top of these approaches to propose an adaptive pruning scheme, which could decrease size of the frequency matrix $M_{l,k}$ by directly assigning 0 to small weights. With less weights values, the size as well as the complexity of CNN could be largely decreased. To fuifil the requirement of different compression rate, we further adopt an adaptive threshold α to remove small weights, which could be defined as follows:

$$\begin{cases} \widetilde{M}_{l,k}(i,j) = M_{l,k}(i,j) & if\ M_{l,k}(i,j) \geq \alpha \\ \widetilde{M}_{l,k}(i,j) = 0 & if\ M_{l,k}(i,j) < \alpha \end{cases} \quad (2)$$

where the prune ratio α is defined as

$$\alpha = \gamma \cdot f_{sort}(\widetilde{M}_{l,k}) \quad (3)$$

where function $f_{sort}()$ represents the sort operation converting a matrix into a value sequence in ascending order, and γ refers to the prune ratio. By assigning different γ, we could achieve compressed CNN systems with different size to fulfill various requirements for storage consumption. After pruning high-frequency part, we could get a sparse version of parameter matrix as shown in Fig. 2 (c).

3.3 Compress and Recover by Compressive Sensing

In this subsection, we will discuss how to utilize compressive sensing [22,23] to compress and recovery the sparse parameter matrix $\widetilde{M}_{l,k}$ computed former step.

Instead of directly compressing the sparse signal x itself, compressive sensing project x with a set of measurement matrices Φ to obtain a measured value y as $y = \Phi x$, where the length of y should be smaller than that of the original signal x. Signal x can be represented in terms of an orthogonal basis $\{\psi_h\}_{h=1}^{n}$ (n refers to the number of basis) as $x = \sum_{h=1}^{n} \psi_h s_h$ or $x = \Psi S$, where S is the n × 1 column vector of weighting coefficients s_h. Compressive sensing tries to explore a group of orthogonal basis, leading the resulting coefficients s_h to have just a few large values and many small values. In that sense, we could use Ψ, Φ and S to reconstruct the projection of signal as $y = \Phi\Psi S$. Meanwhile, Ψ, Φ and S consumes less storage than the original signal, when measurement matrix Φ and orthogonal basis Ψ satisfies restricted isometry property (RIP). Note that RIP guarantees the compressive sensing to be inversible by utilizing an additional restriction on the value of Θ, represented as $\Theta = \Phi\Psi$. Considering parameter matrix $\widetilde{M}_{l,k}$ as a sparse signal, our goal is to design a reasonable set of measurement matrices Φ and orthogonal basis Ψ for compression.

The measurement matrices Φ must ensure the salient information in $\widetilde{M}_{l,k}$ is saved during the compression, meanwhile it will increase the computation burden by analyzing the contents of $\widetilde{M}_{l,k}$. To balance the compression loss and computation cost, we propose to construct Φ as a combination of independent randomly weighted linear vectors following the thought of randomness, which has been proved to be efficient in well-known algorithms, such as random walk, random forest [25] and so on. Inspired by the former work [26], we adopt the distributed random Gaussian matrix Φ to measure the input parameter matrix $\widetilde{M}_{l,k}$. Due to the reason that Φ and I satisfy RIP, we thus determine identify matrix I as Φ. In other words, we could achieve the following equation for compression

$$\Theta = \Phi\Psi = \Phi I = \Phi \quad (4)$$

where Φ is defined as:

$$\Phi(i,j) = \frac{1}{m}\sum_{1}^{m} x, where\ x \sim N(\mu = 0, \sigma^2 = \frac{1}{n}) \quad (5)$$

where m refers to times of random sampling, N represents the Gaussian distribution with zero mean and $\frac{1}{n}$ variance. Once we determine the values of Θ, we could compute the compressed parameter matrix as:

$$S = \Theta \tilde{M}_{l,k} \quad (6)$$

During the decompression stage, we aim to recover the parameter matrix $\tilde{M}_{l,k}$ based on the compressed parameter matrix S and measurement matrix Θ as $\tilde{M}_{l,k} = \Theta^{-1}S$. In fact, there are many possibilities for $\tilde{M}_{l,k}$, due to reason that the location of large coefficient components contained in the signal is uncertain. Therefore, we rewrite $\tilde{M}_{l,k} = \Theta^{-1}S$ in a least-square sense,

$$\tilde{M}_{l,k} = \Theta^{-1} argmin \|S\|_2 \quad (7)$$

This is a convex optimization problem that conveniently reduces to a linear program known as basis pursuit [27]. We try several classical algorithms for signal recovery and determine the approximate message-passing (AMP) [28] algorithm as the proposed solution for Eq. 6, which is the most fast and accurate algorithm for recovery. Essentially, AMP is an iterative algorithm achieving desirable reconstruction performance while running dramatically faster.

3.4 Fine-tune the Network and Encoding

In this subsection, we aim to fine-tune the network after compression, in order to compensate for the loss of precision after compression. We achieve this goal by running a complete round of training as follows:

- We firstly adopt the training set to do a forward propagation with the uncompressed parameters.
- We then calculate the error of the convolution neural network during compression by subtraction between the results of forward propagation performed in the last step and the labels of the training set.
- Next, the error value is utilized to perform a back-propagation to obtain the gradient value, which help update the parameters. Assuming that the error value is δ, the gradient corresponding to each parameter of the convolutional neural network is $\frac{\partial \delta}{\partial N_{i,j}}$ and the iterative formula to update the parameter matrix N could be defined as $N_{i,j} = N_{i,j} + \mu \frac{\partial \delta}{\partial N_{i,j}}$, where μ is the learning rate. The corresponding gradient after compression is then rewritten as follows:

$$\frac{\partial \delta}{\partial S_{i,j}} = f_d\left(Mask_{i,j}\left(f_{dct}\left(\frac{\partial \delta}{\partial N_{i,j}}\right)\right)\right) \quad (8)$$

where $\frac{\partial \delta}{\partial S_{i,j}}$ is the gradient corresponding to the compressed parameter matrix, function $f_d()$ is the sampling function designed for dimensionality reduction, function $f_{dct}()$ refers to the DCT transform and function $Mask_{i,j}()$ represents the mask for an input matrix as follows:

$$\begin{cases} Mask_{i,j}(M_{i,j}) = 0 & if\ M_{i,j} \le \alpha \\ Mask_{i,j}(M_{i,j}) = M_{i,j} & if\ M_{i,j} > \alpha \end{cases} \quad (9)$$

where $M_{i,j}$ refers to the input corresponding to value at the ith row and jth column of the matrix M, α is the pre-defined pruning ratio.

- Finally, we iteratively fine-tune the compressed parameter matrix S as follows:

$$\tilde{S}_{i,j} = S_{i,j} + \mu \frac{\partial \delta}{\partial S_{i,j}} \quad (10)$$

where μ is the learning rate and $\frac{\partial \delta}{\partial S_{i,j}}$ is obtained by former step.

In conclusion, the proposed fine-tune method is designed to minimum the error between the labeling and the results achieved by the compressed network with back-propagation. After fine-tuning, we utilize the Huffman coding [29] to encode the fine-tuned compressed parameter matrix $\tilde{S}_{i,j}$ into a binary sequence, which will be later saved as a binary file and used for decompression at running time.

At running time, the proposed method firstly decodes the saved binary file to obtain the fine-tuned parameter matrix $\tilde{S}$ using Huffman coding. Then, the proposed method recovers the compressed parameter matrix $\tilde{M}_{l,k}$ from $\tilde{S}$ by AMP algorithm. Finally, we do inverse DCT transform to convert $\tilde{M}_{l,k}$ from frequency domain back to spatial domain. These three steps could be represented as

$$N_{l,k} = f_{iDCT}(f_{AMP}(\tilde{S})) \tag{11}$$

4.0 EXPERIMENTS

To evaluate the proposed method, we consider one typical convolution neural network, YOLO network [6,7], which contains convolutional layers (COV), batch normalization layers (BN), max-pooling layers (MP), re-organization layers (RO) and detection layer (DT). By adopting CNN structure for feature extraction and classifier learning, YOLO system could achieve 76.8 mean Average Precision (mAP) on VOC 2007 dataset, which outperforms most of object detection methods including Deformable part-based model (DPM) [9] and R-CNN [5]. However, YOLO system is over 230MB, consisted by 30 layers and $6.74*10^8$ parameters. Running YOLO with the tremendous number of parameters consumes large storage and computational resources. We thus experiment on YOLO network to compress its parameters for smaller storage and computational burden.

Since YOLO is designed for object detection, we use VOC2007 [30], VOC 2012 [31] and COCO test-dev2015 [32] as the benchmark databases to examine the efficiency of YOLO before and after compression. VOC 2007 and 2012 are challenging due to their diversity in object categories and complexity in the layout of images, while COCO is more challenging due to multiple small object targets and complicated layout. Note we follow the guidance of [33] to set IOU threshold as 0.5 for comparisons. Besides, to compare results of compression with other deep compression methods, we use SNR, PSNR, mAP. SNR for a matrix N is defined as

$$SNR = 10 \cdot lg \frac{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} N(i,j)^2}{mn \cdot MSE} \tag{12}$$

where m and n refer to the size of matrix N and MSE represents mean squared error and is defined as

$$\text{MSE} = \frac{1}{\text{mn}}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} \| N(i,j) - \tilde{S}(i,j) \| \tag{13}$$

Table 1. Compression performance comparison between the proposed method, Deep compression and HashedNets on YOLO network for the VOC 2007 dataset. Note that the superscripts 1, 2 and 3 correspond to the proposed method, Deep compression and HashedNets, respectively. Note that mAP is measured with %.

Prune Ratio(α)	SNR^1	$PSNR^1$	mAP^1	SNR^2	$PSNR^2$	mAP^2	SNR^3	$PSNR^3$	mAP^3
0.9	**61.43**	**71.96**	**88.12**	42.81	60.82	**89.02**	50.92	60.82	83.07
0.8	**57.10**	**65.14**	**87.04**	38.56	53.75	83.64	49.01	59.94	78.84
0.7	**52.89**	**59.90**	**85.89**	33.97	47.49	72.96	43.84	53.40	74.26
0.6	**47.94**	**54.58**	**76.16**	28.44	44.09	59.66	20.98	33.45	70.09
0.5	16.69	28.67	**70.32**	**23.92**	**41.80**	44.07	14.21	22.98	63.88
Average	**47.21**	**56.05**	**81.51**	33.54	49.59	69.87	35.79	46.12	74.03

Table 2. Compression performance comparison between the proposed method, Deep compression and HashedNets on YOLO network for the VOC 2012 dataset.

Prune Ratio(α)	SNR^1	$PSNR^1$	mAP^1	SNR^2	$PSNR^2$	mAP^2	SNR^3	$PSNR^3$	mAP^3
0.9	**51.70**	**62.44**	**83.80**	30.94	51.12	83.66	41.29	53.17	78.60
0.8	**46.70**	**58.65**	**82.81**	25.27	45.46	79.12	40.29	52.68	74.48
0.7	**42.48**	**53.31**	**81.88**	20.39	40.61	70.96	35.55	47.59	70.45
0.6	**39.21**	**49.95**	**71.94**	16.49	36.77	58.37	18.23	30.25	65.91
0.5	**14.64**	25.53	**64.36**	13.23	**33.63**	43.40	12.42	21.12	59.76
Average	**38.95**	**49.98**	**76.96**	21.26	41.52	67.10	29.56	40.96	69.84

Table 3. Compression performance comparison between the proposed method, Deep compression and HashedNets on YOLO network for COCO test-dev2015 dataset.

Prune Ratio(α)	SNR^1	$PSNR^1$	mAP^1	SNR^2	$PSNR^2$	mAP^2	SNR^3	$PSNR^3$	mAP^3
0.9	**45.16**	**53.49**	**43.80**	38.67	43.39	43.73	42.04	45.12	40.16
0.8	**39.34**	**46.53**	**42.92**	32.30	37.83	41.09	39.82	43.67	37.81
0.7	**34.78**	**42.11**	**41.75**	26.93	31.91	37.53	34.34	38.20	33.45
0.6	**30.04**	**39.97**	32.65	22.97	27.78	**33.82**	19.15	23.17	29.14
0.5	16.93	21.08	**28.36**	**19.01**	**25.29**	28.32	13.49	17.74	24.76
Average	**33.25**	**40.64**	**37.90**	27.98	33.24	36.90	29.77	33.58	33.06

where N and $\tilde{S}$ represent the input and compressed parameter matrix, respectively. PSNR is defined as

$$PSNR = 10 \cdot lg \frac{max\ (N)}{MSE} \tag{14}$$

where function $max\ ()$ refers to obtain the maximal value in the matrix. When comparing between two methods, the performance is considered better if SNR, PSNR and mAP for all three datasets are larger values. In fact, SNR and PSNR are two measurements to judge the loss of information for parameter matrix during compression, while mAP focuses on the detection performance of YOLO system.

Table. 1, 2 and 3 give the detailed statics of the proposed method, Deep Compression and HashedNets on YOLO network for the VOC 2007, VOC 2012 and COCO test-dev2015 dataset, measured on a PC with 2.5GHz i7 CPU, 8GB RAM and one 1080Ti GPU. HashedNets and Deep Compression are both novel methods to reduce and limit the memory overhead of neural networks. To compare with these two methods, we implement their algorithms according to the instructions given in the published paper. It's noted that the object detection rate of YOLO as well as our proposed method could be adjusted with different sets of parameters. For the convenience of comparison on performance of compressing, we set the object detection rate to 45FPS. Note that the prune ratio in Tables refers to α defined in Eq. 3. It is evident by average value of SNR, PSNR and mAP that the proposed method achieves better results than Deep Compression [10] and Hashed Nets [19] in terms of three measurements on three datasets. The comparison advantage of the proposed method is more obvious when the prune ratio is set as 0.6, where the proposed method gets far better performance than Deep Compression and HashedNets in SNR and PSNR. However, the corresponding difference of mAP is smaller. This inconsistent performance between SNR, PSNR and mAP is caused by the fact that the performance of YOLO is affected by many factors including parameters and the relation between parameters and detection accuracy is non-linear. For different datasets, we can clearly view a mAP, SNR and PSNR drop when testing on COCO dataset, which proves that COCO is more challenging than VOC 07 and 12 for object detection. It's noted that our method could slightly decrease the computation cost at 10% when the prune ratio is set at 60%. By setting prune ratio to 60%, the average running time of an image for testing and training is 0.020s and 0.084s, respectively.

We show the comparison of SNR, PSNR and mAP on VOC2012 between the proposed method, Deep Compression and HashedNets in Fig. 3, 4 and 5, respectively. The mAP value of our methods drops greatly when the prune ratio decreases from 0.7 to 0.5. Deep compression drops greater than the proposed method, while HashedNets is stable in mAP during compression. The unstable performance of the proposed method could be explained by the fact that mAP would drop fast if the proposed method prunes the low-frequency part of parameter matrix. This would happen if the prune ratio is set too low to retain the low-frequency part. We could find that our method gets stable performance in SNR and PSNR when the prune ratio decreases from 0.9 to 0.6. HashedNets gets stable performance in a smaller range between 0.9 and 0.7, while Deep Compression obtains more stable performance than other two methods. It drops so slow that it even exceed the proposed method in PSNR when the prune ratio is set to 0.5.

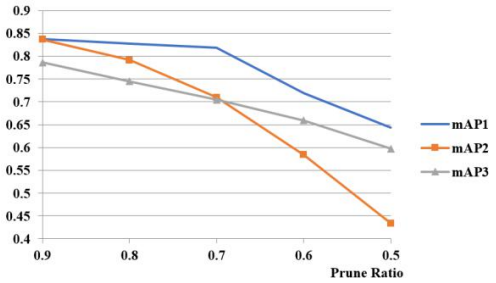


Fig. 3. Comparison of mAP between our method, Deep Compression and HashedNets, which are represented by label 1, 2 and 3.

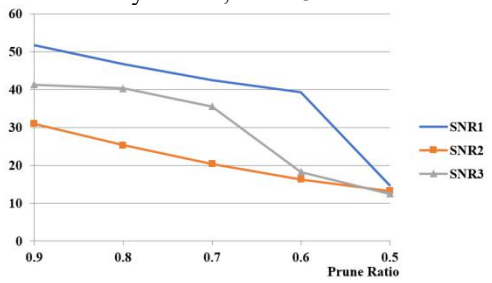


Fig. 4. Comparison of SNR between our method, Deep Compression and HashedNets, which are represented by label 1, 2 and 3.

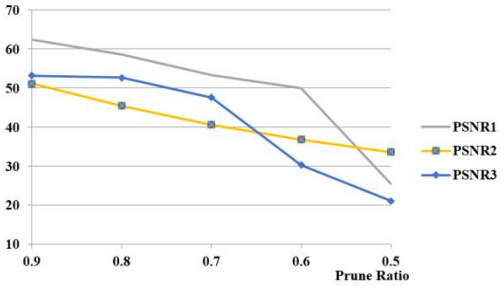


Fig. 5. Comparison of PSNR between our method, Deep Compression and HashedNets, which are represented by label 1, 2 and 3.

After compression, we shift YOLO to mobile phones and test its performance by setting prune ratio as 60%. Sample qualitative results of the shifted YOLO are shown in Fig. 6, where the first and second row represent detection results of real-life scene images and images in VOC 2012 dataset. From these sample results, we can see the shifted version of YOLO could detect object accurately and fast, even facing challenges of diversity in object categories and complexity in the layout of images. We also show several failure cases in Fig. 7, when the prune ratio is settled too low, i.e., 30%. Due to the low prune ratio, the decompression parameter matrixes are not enough to support assign correct labels for objects. Therefore, we need set proper ratio value to keep a balance between compression rate and accuracy of object detection.

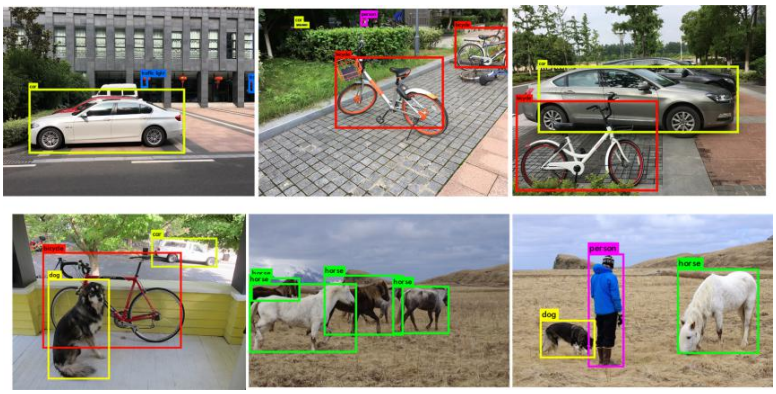


Fig. 6. Detection examples of the proposed method on several real-life scene images and images from VOC 2012 dataset, when the prune ratio.

Fig. 7. Failure detection examples of the proposed method on images from VOC 2012 dataset.

5.0 CONCLUSION

In this paper, we propose a novel method to compress convolutional neural network by frequency pruning and compressive sensing, result in decreasing the computation and storage cost. After converting parameter matrix of CNN to frequency domain by DCT, we could achieve a low-frequency dominated matrix due to the inherent smooth property of parameters when processing images. Then, we prune high-frequency components to make the frequency matrix sparse. Next, we sample the sparse frequency matrix with distributed random Gaussian matrix. Finally, we retrain the network to fine-tune the remaining parameters and encode the fine-tuned parameters into binary file by Huffman Coding. Experiment results on YOLO network, a typical network designed for object detection shows that the proposed method outperforms two relevant methods. We also show examples of object detection results after shifting compressed YOLO to mobile phones. Our future work includes the exploration on compressing RNN network with the proposed method

REFERENCES

[1] M. Liang, and X. Hu. "Recurrent convolutional neural network for object recognition" in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, 2015, pp. 3367-3375.

[2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition”, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, 2016, pp. 770-778.

[3] S. Ren, K. He, R. Girshick, et al. “Faster R-CNN: towards real-time object detection with region proposal networks”, IEEE transactions on pattern analysis and machine intelligence, 2017, 39(6): 1137-1149.

[4] S. Ren, K. He, R. Girshick, et al. “Object detection networks on convolutional feature maps”, IEEE transactions on pattern analysis and machine intelligence, 2017, 39(7): 1476-1481.

[5] T. Y. Lin, P. Dollár, R. Girshick, et al. “Feature pyramid networks for object detection”, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Hawaii, 2017, 1(2): 4.

[6] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection”, in Proceedings of the IEEE conference on computer vision and pattern recognition, Las Vegas, 2016, pp. 779–788.

[7] J. Redmon and A. Farhadi, “YOLO9000: better, faster, stronger”, CoRR, 2016, abs/1612.08242, 2016.

[8] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results”, http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[9] P. F. Felzenszwalb, R. B. Girshick, D. A. McAllester, and D. Ramanan. “Object detection with discriminatively trained part-based models”, IEEE Trans. Pattern Anal. Mach. Intell., Vol. 32, No.9, 2010, pp.1627–1645.

[10] G. Cybenko, “Approximation by superpositions of a sigmoidal function, Mathematics of control”, signals and systems, Vol. 2, No. 4, 1989, pp. 303–314.

[11] Y.N. Dauphin and Y. Bengio, “Big neural networks waste capacity”, CoRR, 2013, abs/1301.3583.

[12] M. Courbariaux and Y. B. Binarynet, “Training deep neural networks with weights and activations constrained to +1 or -1”, CoRR,2016, abs/1602.02830.

[13] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi, “XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks”, in Proceedings of the European Conference on Computer Vision, Amsterdam, 2016, pp. 525-542.

[14] Y. He, X. Zhang, J. Sun, “Channel Pruning for Accelerating Very Deep Neural Networks”. CoRR, 2017, abs/1707.06168.

[15] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N.de Freitas, “Predicting parameters in deep learning”, in Proceedings of Advances in Neural Information Processing Systems, Lake Tahoe, 2013, pp. 2148–2156.

[16] Y. Gong, L. Liu, M. Yang, and L. D. Bourdev, “Compressing deep convolutional networks using vector quantization”, CoRR, 2014, abs/1412.6115.

[17] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network”, in Proceedings of Advances in Neural Information Processing Systems, Montreal, 2015, pp. 1135–1143.

[18] S. Han, H. Mao, and W. J. Dally, “Deep compression, Compressing deep neural network with pruning, trained quantization and huffman coding”, CoRR, 2015, abs/1510.00149.

[19] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, “Compressing convolutional neural networks in the frequency domain”, in Proceedings of the International Conference on Knowledge Discovery and Data Mining, San Francisco, 2016, pp.1475–1484.

[20] N. Ahmed, T. Natarajan, and K. R. Rao, “Discrete cosine transform”, IEEE Trans. Computers, Vol. 23, No.1, 1974, pp.90–93.

[21] S. J. Hanson and L. Y. Pratt. “Comparing biases for minimal network construction with back-propagation”, in Proceedings of Advances in Neural Information Processing Systems, Denver, 1988, pp. 177–185.

[22] S. Foucart and H. Rauhut. “A Mathematical Introduction to Compressive Sensing. Applied and Numerical Harmonic Analysis”. 2013, Birkhäuser.

[23] R. G. Baraniuk, V. Cevher, M. F. Duarte, and C. Hegde, “Model-based compressive sensing”. IEEE Trans. Information Theory, Vol. 56, No.4, 2010, pp. 1982–2001.

[24] Pearson K. The problem of the random walk. Nature, Vol. 72, No. 1867, 1905, pp. 342.

[25] A. Liaw, and M. Wiener, "Classification and regression by random Forest," R news, Vol. 2, No. 3, 2002, pp. 18-22.

[26] T. T. Do, T. D. Tran, and L. Gan, “Fast compressive sampling with structurally random matrices”, in Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, Las Vegas, 2008, pp. 3369–3372.

[27] D. Donoho, “Compressed sensing,” IEEE Trans. Inform. Theory, vol. 52, no. 4, April 2006, pp. 1289–1306,

[28] D. L. Donoho, A. Maleki, and A. Montanari, “Message passing algorithms for compressed sensing”. CoRR, 2009, abs/0907.3574.

[29] J. V. Leeuwen, “On the construction of huffman trees”, in Proceedings of International Colloquium on Automata, Languages, and Programming, Prague, 1976, pp. 382–410.

[30] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[31] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results", http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[32] COCO: Common Objects in Context. http://mscoco.org/dataset/ #detections-leaderboard (2016) [Online; accessed 25-July-2016].

[33] Bell, S., Zitnick, C.L., Bala, K., Girshick, R.: Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. InProceedings of the IEEE conference on computer vision and pattern recognition, Las Vegas, 2016: 2874-2883.