Cogradient Descent for Bilinear Optimization

Li'an Zhuo,[1] Baochang Zhang,[1]* Linlin Yang,[2] Hanlin Chen,[1] Qixiang Ye,[3]
David Doermann,[4] Rongrong Ji,[5] Guodong Guo [6]
[1] School of Automation Science and Electrical Engineering, Beihang University,
[2] University of Bonn, [3] University of Chinese Academy of Sciences
[4] University at Buffalo, [5] School of Information Science and Engineering, Xiamen University,
[6] Institute of Deep Learning, Baidu; National Engineering Lab for Deep Learning Tech. and App.
{lianzhuo, bczhang}@buaa.edu.cn

Abstract

Conventional learning methods simplify the bilinear model by regarding two intrinsically coupled factors independently, which degrades the optimization procedure. One reason lies in the insufficient training due to the asynchronous gradient descent, which results in vanishing gradients for the coupled variables. In this paper, we introduce a Cogradient Descent algorithm (CoGD) to address the bilinear problem, based on a theoretical framework to coordinate the gradient of hidden variables via a projection function. We solve one variable by considering its coupling relationship with the other, leading to a synchronous gradient descent to facilitate the optimization procedure. Our algorithm is applied to solve problems with one variable under the sparsity constraint, which is widely used in the learning paradigm. We validate our CoGD considering an extensive set of applications including image reconstruction, inpainting, and network pruning. Experiments show that it improves the state-of-the-art by a significant margin[1].

1. Introduction

Bilinear models are cornerstones of many computer vision algorithms since often the optimized objectives or models are influenced by two or more hidden factors which interact to produce our observations. With bilinear models we can disentangle for example, illumination and object colors in color constancy, the shape from shading, and object identity and its pose in recognition. Such models have shown great potential in extensive low-level applications including deblurring [28], denosing [1], and 3D object reconstruction [3]. They have also evolved in convolutional neural networks (CNNs) to model feature interactions, which are particularly useful for fine-grained categorization and model pruning [12, 16].

A basic bilinear problem attempts to optimize the following object function as

$$\arg\min_{\mathbf{A},\mathbf{x}} G(\mathbf{A},\mathbf{x}) = \|\mathbf{b}-\mathbf{A}\mathbf{x}\|_2^2 + \lambda\|\mathbf{x}\|_1 + R(\mathbf{A}), \quad (1)$$

where $\mathbf{b} \in \mathbb{R}^{M\times 1}$ is an observation that can be characterized by $\mathbf{A} \in \mathbb{R}^{M\times N}$ and $\mathbf{x} \in \mathbb{R}^{N\times 1}$. $R(\cdot)$ represents the regularization which is normally ℓ_1 or ℓ_2 norm. $\|\mathbf{b}-\mathbf{A}\mathbf{x}\|_2^2$ can be replaced by any function with the part $\mathbf{A}\mathbf{x}$. The bilinear models are generally with one variable of a sparsity constraint such as ℓ_1 regularization, which is widely used in machine learning. The term contributes to parsimony and avoids overfitting.

Existing methods tend to split the bilinear problem into easy sub-problems, and then solve them using the alternating direction method of multipliers (ADMM). Fourier domain approaches [2, 24] are also exploited to solve the ℓ_1 regularization sub-problem via soft thresholding. Furthermore, recent works [7, 25] split the objective into a sum of convex functions and introduces ADMM with proximal operators to speed up the convergence. Although the generalized bilinear model [27] considers a nonlinear combination of several end members in one matrix (not two bilinear matrices), it only proves to be effective in unmixing hyperspectral images. These approaches actually simplify the bilinear problems by regarding the two factors as independent, *i.e.* , they optimize a variable while keeping the other fixed.

Without considering the relationship between two hidden factors, however, existing methods suffer from suboptimal solutions caused by an asynchronous convergence speed of the hidden variables. As shown in Fig. 1, when $\mathbf{x}$ goes sparse with zero values, it causes gradient vanishing for the corresponding element in $\mathbf{A}$ (see in Eq. 3). The process results into an insufficient optimization, considering that a corruption in $\mathbf{A}$ causes different coding residuals

*Baochang Zhang is the corresponding author.
[1]Source code is available at https://github.com/bczhangbczhang.

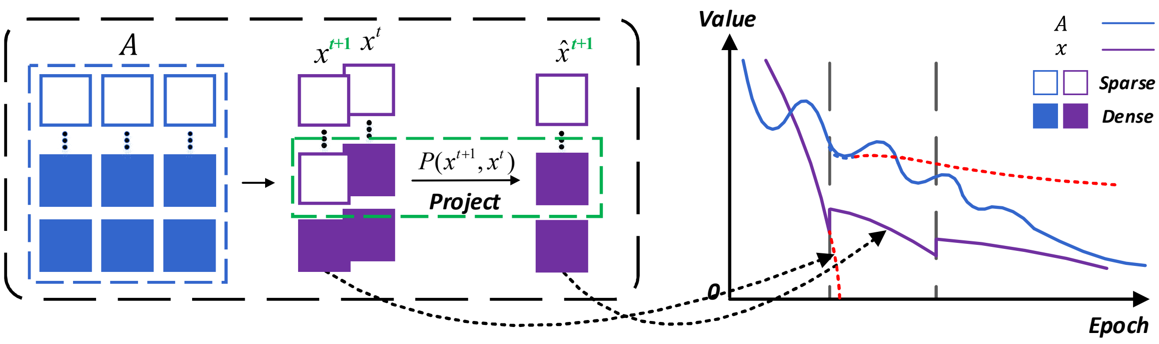

Figure 1: Illustration of CoGD. Conventional gradient based algorithms has a heuristic that the two hidden variables in bilinear models are independent. Nevertheless, we validate that such an heuristic is implausible and the algorithms suffer from asynchronous convergence and sub-optimal solutions. The red dotted lines denote that the sparsity of $\mathbf{x}$ causes an inefficient training of $\mathbf{A}$. We propose a projection function to coordinate the gradient of the hidden variables.

and coding coefficients for the collaborative representation nature of Eq. 1 [7, 31].

In this paper, we propose a Cogradient Descent algorithm (CoGD) for bilinear models, and target at addressing the gradient varnishing problem by considering the coupled relationship with other variables. A general framework is developed to show that the coupling among hidden variables can be used to coordinate the gradients based on a specified projection function. We apply the proposed algorithm to the problems with one variable of the sparsity constraint, which is widely used in the learning paradigm. The contributions of this work are summarized as follows

- We propose a Cogradient Descent algorithm (CoGD) to optimize gradient-based learning procedures, as well as develop a theoretical framework to coordinate the gradient of hidden variables based on a projection function.
- We propose an optimization framework which decouples the hidden relationship and solves the asynchronous convergence in bilinear models with gradient-based learning procedures.
- Extensive experiments demonstrated that the proposed algorithm achieves significant performance improvements on convolutional sparse coding (CSC) and network pruning.

2. Related Work

In this section, we highlight the potential of Eq. 1 and show its applications. Convolutional sparse coding (CSC) is a classic bilinear problem and has been exploited for image reconstruction and inpainting. Existing algorithms for CSC tend to split the bilinear problem into subproblems, each of which is iterativly solved by ADMM. Furthermore, bilinear models are able to be embedded in CNNs. One application is network pruning. With the aid of bilinear models, we can select important feature maps and prune the channels [16]. To solve bilinear models in network pruning, some iterative methods like modified Accelerated Proximal Gradient algorithms (APG) [8] and iterative shrinkage-thresholding algorithms (ISTA) [26, 14] are introduced. Other applications like fine-grained categorization [15, 11], visual question answering (VQA) [29] or person re-identification [23], attempt to embed bilinear models within CNNs to model the pairwise feature interactions and fuse multiple features with attention. To update the parameters in a bilinear model, they directly utilize the gradient descent algorithm and back-propagate the gradients of the loss.

3. The Proposed Method

Unlike previous bilinear models which optimize one variable while keeping another fixed, our strategy considers the relationship of two variables with the benefits on the linear inference. In this section, we first discuss gradient varnishing for bilinear models and then propose our CoGD.

3.1. Gradient Varnishing

Given that $\mathbf{A}$ and $\mathbf{x}$ are independent, the conventional gradient descent method can be used to solve the bilinear problem as

$$\mathbf{A}^{t+1} = \mathbf{A}^{t} + \eta_1 \frac{\partial G}{\partial \mathbf{A}}, \tag{2}$$

and

$$(\frac{\partial G}{\partial \mathbf{A}})^T = \mathbf{x}^t(\mathbf{A}\mathbf{x}^t - \mathbf{b})^T = \mathbf{x}^t \hat{G}(\mathbf{A}, \mathbf{x}), \tag{3}$$

where $\hat{G}$ is obtained by considering the bilinear problem as in Eq. 1, and we have $\hat{G}(\mathbf{A}, \mathbf{x}) = (\mathbf{A}\mathbf{x}^t - \mathbf{b})^T$. It shows that the gradient for $\mathbf{A}$ is vanishing when $\mathbf{x}$ becomes zero, which causes an asynchronous convergence problem. Note that for simplicity, the regularization term is not considered. Similar for $\mathbf{x}$, we have

$$\mathbf{x}^{t+1} = \mathbf{x}^{t} + \eta_2 \frac{\partial G}{\partial \mathbf{x}}. \tag{4}$$

Here, η_1, η_2 represent the learning rate. The conventional gradient descent algorithm for bilinear models iteratively optimizes one variable while keeping another fixed, failing to consider the relationship of the two hidden variables in optimization.

3.2. Cogradient Descent Algorithm

We then consider the problem from a new perspective such that $\mathbf{A}$ and $\mathbf{x}$ are coupled. It is reasonable that $\mathbf{x}$ is used to select the element from $\mathbf{A}$, which are interactive for a better representation. Firstly, based on the chain rule [21] and its notations, we obtain

$$\hat{x}_j^{t+1} = x_j^t + \eta_2 (\frac{\partial G}{\partial x_j} + Tr((\frac{\partial G}{\partial \mathbf{A}})^T \frac{\partial \mathbf{A}}{\partial x_j})), \tag{5}$$

where $(\frac{\partial G}{\partial \mathbf{A}})^T = \mathbf{x}^t \hat{G}(\mathbf{A}, \mathbf{x})$ as shown in Eq. 3. $Tr(\cdot)$ represents the trace of matrix, which means each element in matrix $\frac{\partial G}{\partial x_j}$ adds the trace of the corresponding matrix related to x_j. Remembering in Eq. 4 that $\mathbf{x}^{t+1} = \mathbf{x}^t + \eta_2 \frac{\partial G}{\partial \mathbf{x}}$, Eq. 5 becomes

$$\hat{\mathbf{x}}^{t+1} = \mathbf{x}^{t+1} + \eta_2 \mathbf{c}^t \odot \mathbf{x}^t, \tag{6}$$

where $\odot$ represents the Hadamard product, and $\mathbf{c}^t = [\sum_i^M \hat{g}_i \frac{\partial \mathbf{A}_{i1}}{\partial x_1}, \dots, \sum_i^M \hat{g}_i \frac{\partial \mathbf{A}_{iN}}{\partial x_N}]^T$. $\hat{G} = [\hat{g}_1, \dots, \hat{g}_M]$. The full derivation of Eq. 6 is given in the supplementary file [2]. It is then reformulated as a projection function as

$$\hat{\mathbf{x}}^{t+1} = P(\mathbf{x}^{t+1}, \mathbf{x}^t) = \mathbf{x}^{t+1} + \beta \odot \mathbf{x}^t. \tag{7}$$

This equation explains the rationality of our method, which is actually based on a projection function to solve the asychonous problem of the bilinear optimization by controlling β. We first judge when an asynchronous convergence happens in the optimization based on a form of logical operation as

$$(\neg s(\mathbf{x})) \wedge (s(\mathbf{A})) = 1, \tag{8}$$

and

$$s(*) = \begin{cases} 1 & if\ R(*) > \alpha, \\ 0 & otherwise, \end{cases} \tag{9}$$

where α represents the threshold which variates for different variables. Eq. 8 describes our assumption that an asynchronous convergence happens for $\mathbf{A}$ and $\mathbf{x}$, when their norms become very different. Accordingly, we define the update rule of our CoGD as

$$\hat{\mathbf{x}}^{t+1} = \begin{cases} P(\mathbf{x}^{t+1}, \mathbf{x}^t) & if\ (\neg s(\mathbf{x})) \wedge (s(\mathbf{A})) = 1, \\ \mathbf{x}^{t+1} & otherwise, \end{cases} \tag{10}$$

which leads to an synchronous convergence and generalizes the conventional gradient descent method. Then our CoGD is well established.

[2]Supplementary file is available at https://github.com/bczhangbczhang.

4. Applications

We apply the proposed algorithm on CSC and CNNs to validate its performance on image reconstruction, inpainting, and CNNs pruning.

4.1. CoGD for Convolutional Sparse Coding

Unlike patch based sparse coding [18], CSC operates on the whole image, thereby decomposing a more global dictionary and set of features based on considerably more complicated process. CSC is formulated as

$$\begin{aligned} &\arg\min_{\mathbf{A},\mathbf{x}} \frac{1}{2} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_F^2 + \lambda \|\mathbf{x}\|_1 \\ &s.t.\ \|\mathbf{a}_k\|_2^2 \leq 1 \quad \forall k \in \{1, \dots, K\}, \end{aligned} \tag{11}$$

where $\mathbf{b}$ are input images, $\mathbf{x} = [\mathbf{x}_1^T, \dots, \mathbf{x}_K^T]^T$ is with sparsity constraint and $\mathbf{A} = [\mathbf{A}_1, \dots, \mathbf{A}_K]$ is a concatenation of Toeplitz matrices representing convolution with the respective filters $\mathbf{a}_k$. λ is the sparsity parameter and K is the number of the kernels. According to the framework [4], we introduce a diagonal or block-diagonal matrix $\mathbf{M}$, and then reformulate Eq. 11 as

$$\arg\min_{\mathbf{A},\mathbf{x}} f_1(\mathbf{A}\mathbf{x}) + \sum_{k=1}^{K} (f_2(\mathbf{x}_k) + f_3(\mathbf{a}_k)), \tag{12}$$

where

$$\begin{aligned} f_1(\mathbf{v}) &= \frac{1}{2} \|\mathbf{b} - \mathbf{M}\mathbf{v}\|_F^2, \\ f_2(\mathbf{v}) &= \lambda \|\mathbf{v}\|_1, \\ f_3(\mathbf{v}) &= ind_c(\mathbf{v}). \end{aligned} \tag{13}$$

Here, $ind_c(\cdot)$ is an indicator function defined on the convex set of the constraints $C = \{\mathbf{x} | \|\mathbf{S}\mathbf{x}\|_2^2 \leq 1\}$. The solution to Eq. 12 by ADMM with proximal operators is detailed in [19, 7]. However, it suffers from asynchronous convergence, which leads to a sub-optimal solution. Instead, we introduce our CoGD for this optimization. Note that in this case, similar to Eq. 10, we have

$$\hat{\mathbf{x}}_k = \begin{cases} P(\mathbf{x}_k^{t+1}, \mathbf{x}_k^t) & if\ (\neg s(\mathbf{x}_k)) \wedge (s(\mathbf{A}_k)) = 1 \\ \mathbf{x}_k^{t+1} & otherwise \end{cases} \tag{14}$$

where $\hat{G}(\mathbf{A}, \mathbf{x}) = (\mathbf{A}\mathbf{x}^t - \mathbf{b})^T$. β, $\alpha_{\mathbf{A}}$ and $\alpha_{\mathbf{x}}$ are detailed in the experiments. Finally, to jointly solve $\mathbf{A}$ and $\mathbf{x}$, we follow the proposed approach in sec. 3.2 to solve the two coupled variables iteratively, yielding Alg. 1.

4.2. CoGD for CNNs Pruning

4.2.1 CNNs Pruning based on a Bilinear Model

Channel pruning has received increased attention recently for compressing convolutional neural networks (CNNs).

Algorithm 1 CoGD for CSC.

```
Require: The training dataset; sparisty factor $\lambda$; hyperparameters such as penalty parameters, threshold $\alpha_{\mathbf{A}}$, $\alpha_{\mathbf{x}}$.
Ensure: The filters $\mathbf{a}$ and the sparse feature maps $\mathbf{x}$.
1: Initialize $\mathbf{a}^0, \mathbf{x}^0$ and others
2: repeat
3:     Use $f_3(\mathbf{a}_k)$ in Eq. 13 to calculate kernel norm
4:     Use Eq. 14 to calculate $\mathbf{x}$
5:     for all $l = 1$ to $L$ epochs do
6:         Kernel Update:
           $\mathbf{a}^l \leftarrow \arg\min_{\mathbf{a}} f_1(\mathbf{Ax}) + \sum_{k=1}^{K} f_3(\mathbf{a}_k)$ using ADMM with proximal operators
7:         Code Update:
           $\mathbf{x}^l \leftarrow \arg\min_{\mathbf{x}} f_1(\mathbf{Ax}) + \sum_{k=1}^{K} f_2(\mathbf{x}_k)$ using ADMM with proximal operators
8:     end for
9: until Loss convergence.
```

Early works in this area tended to directly prune the kernel based on simple criteria like the norm of kernel weights [10] or a greedy algorithm [17]. More recent approaches have sought to formulate network pruning as a bilinear optimization problem with soft masks and sparsity regularization [6, 26, 8, 14].

Based on the framework of [6, 26, 8, 14], we apply our CoGD for channel pruning. In order to prune the channel of the pruned network, the soft mask $\mathbf{m}$ is introduced after the convolutional layer to guide the output channel pruning and has a bilinear form of

$$F_j^{l+1} = f(\sum_i F_i^l \otimes (W_{i,j}^l \mathbf{m}_j)), \tag{15}$$

where F_j^l and F_j^{l+1} are the i-th input and the j-th output feature maps at the l-th and $l+1$-th layer. $W_{i,j}^l$ are convolutional filters which correspond to the soft mask $\mathbf{m}$. $\otimes$ and $f(\cdot)$ refer to convolutional operator and activation, respectively.

In this framework, the soft mask $\mathbf{m}$ can be learned end-to-end in the back propagation process. To be consistent with other pruning works, we use W and $\mathbf{m}$ instead of $\mathbf{A}$ and $\mathbf{x}$ in this part. Then a general optimization function for network pruning with soft mask is formulated with a bilinear form as

$$\arg\min_{W,\mathbf{m}} \mathcal{L}(W\mathbf{m}) + \lambda \|\mathbf{m}\|_1 + R(W), \tag{16}$$

where $\mathcal{L}(W\mathbf{m})$ is the loss function, which will be detailed in following. With the sparsity constraint on $\mathbf{m}$, the convolutional filters with zero value in the corresponding soft mask are regarded as useless filters, which means that these filters and their corresponding channels in the feature maps

Algorithm 2 CoGD for Pruning CNNs in Bilinear Modeling.

```
Require: The training dataset; the pre-trained network with weights $W_B$; sparsity factor $\lambda$; hyperparameters such as learning rate, weight decay, threshold $\alpha_W$, $\alpha_{\mathbf{m}}$.
Ensure: The pruning network.
1: Initialize $W_G = W_B$, $\mathbf{m} \sim N(0,1)$;
2: repeat
3:     Use $R(W_G)$ in Eq. 16 to obtain the norms
4:     Use Eq. 17 to calculate the new soft mask $\hat{\mathbf{m}}$
5:     for all $l = 1$ to $L$ epochs do
6:         for all $i$ steps do
7:             Fix $W_G$ and update $W_D$ using Eq. 20
8:         end for
9:         for all $j$ steps do
10:            Fix $W_D$ and update $W_G$ using Eq. 20
11:        end for
12:    end for
13: until Loss convergence.
```

have no significant contribution to the subsequent computation and will ultimately be removed. However, there is a dilemma in the pruning-aware training in that the pruned filters are not evaluated well before they are pruned, which leads to sub-optimal pruning. Particularly, $\mathbf{m}$ and the corresponding kernels do not become sparse in a synchronous manner, which will cause an inefficient training problem for CNNs, which is allured in our paper. To address the problem, we again apply our CoGD to calculate the soft mask, by reformulating Eq. 10 as

$$\hat{\mathbf{m}}_j^{l,t+1} = \begin{cases} P(\mathbf{m}_j{}^{l,t+1}, \mathbf{m}_j{}^{l,t}) & if\ (\neg s(\mathbf{m}_j^{l,t})) \wedge s(\sum_i W_{i,j}^l) = 1 \\ \mathbf{m}_j^{l,t+1} & otherwise, \end{cases} \tag{17}$$

where $W_{i,j}$ represents the 2D kernel of the i-th input channel of the j-th filter. β, α_W and $\alpha_{\mathbf{m}}$ are detailed in our experiments. To further investigate the form of $\hat{G}$ for CNNs pruning, we have

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial W_{i,j,p,q}} &= \mathbf{m}_j \sum_h^H \sum_w^W ((\frac{\partial \mathcal{L}}{\partial F_j^{l+1}})_{h,w} F_{i,h+p-1,w+q-1}^l) \\ &= \mathbf{m}_j \hat{g}_{j,i}, \end{aligned} \tag{18}$$

where the size of input feature maps F_i^l is $H \times W$. p and q is the kernel indexs for convolutional filters $W_{i,j}^l$. Noted that the above partial gradient has usually been calculated by the autograd package in common deep learning frameworks, *e.g.* , Pytorch [20]. Therefore, based on Eq. 18, we simplify the calculation of $\hat{G}$ as following:

$$\hat{G} = \frac{\partial \mathcal{L}}{\partial W_{i,j}} / \mathbf{m}_j. \tag{19}$$

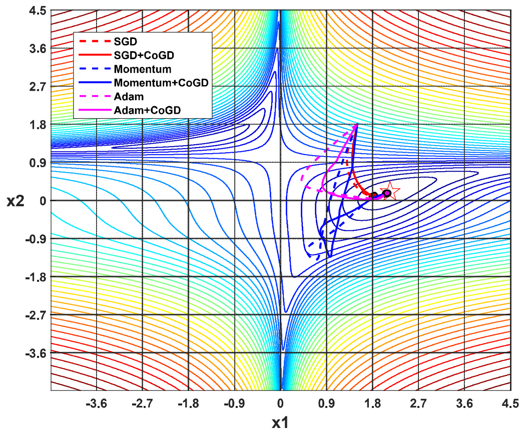

Figure 2: The contour map of beale functions. It can be seen that algorithms with CoGD have a short optimization path compared with their counterparts, which shows that CoGD facilitates efficient and sufficient training.

We prune CNNs based on the new mask $\hat{\mathbf{m}}$ in Eq. 17. We use GAL[3] [14] as an example to describe our CoGD for CNNs pruning. A pruned network obtained through GAL, with ℓ_1-regularization on the soft mask is used to approximate the pre-trained network by aligning their outputs. The discriminator D with weights W_D is introduced to discriminate between the output of pre-trained network and pruned network, and the pruned network generator G with weights W_G and soft mask $\mathbf{m}$ is learned together with D by using the knowledge from supervised features of baseline.

Accordingly, $\mathbf{m}$, $\hat{\mathbf{m}}$, the pruned network weights W_G and the discriminator weights W_D are all learned by solving the optimization problem as follows

$$\begin{aligned}\arg\min_{W_G,\mathbf{m}}\max_{W_D,\hat{\mathbf{m}}}&\mathcal{L}_{Adv}(W_G,\hat{\mathbf{m}},W_D)+\mathcal{L}_{data}(W_G,\hat{\mathbf{m}})\\&+\mathcal{L}_{reg}(W_G,\mathbf{m},W_D).\end{aligned} \tag{20}$$

where $\mathcal{L}(W\mathbf{m}) = \mathcal{L}_{Adv}(W_G,\hat{\mathbf{m}},W_D) + \mathcal{L}_{data}(W_G,\hat{\mathbf{m}})$ and $\mathcal{L}_{reg}(W_G,\mathbf{m},W_D)$ is related to $\lambda\|\mathbf{m}\|_1 + R(W)$ in Eq. 16. $\mathcal{L}_{Adv}(W_G,\hat{\mathbf{m}},W_D)$ is the adversarial loss to train the two-player game between the pre-trained network and the pruned network that compete with each other. More details of the algorithm are shown in Alg. 2.

The advantages of CoGD in network pruning are threefold. First, our method optimizes the bilinear pruning model, which leads to a synchronous gradient convergence. Second, the process is controllable by the threshold, which makes the pruning rate easy to adjust. Third, our method for pruning CNNs is generic and can be built atop of other state-of-the-art networks such as [6, 26, 8, 14], for better performance.

[3]We implement our method based on their source code.

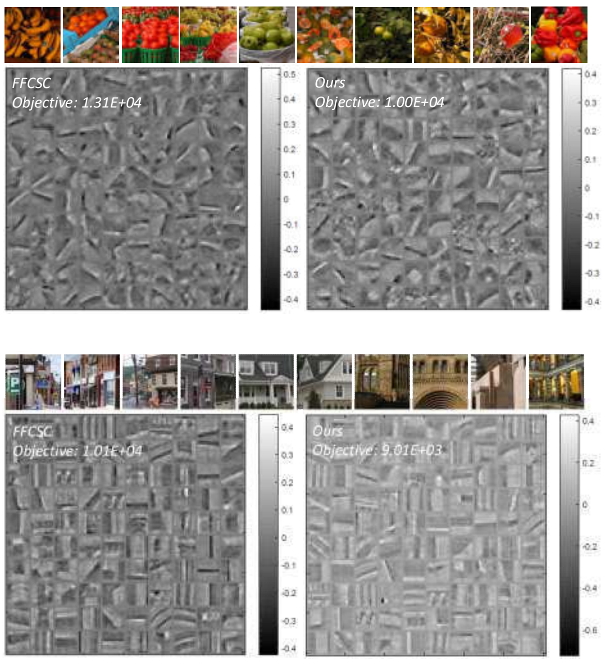

Figure 3: Filters learned on fruit and city datasets. Thumbnails of the datasets along with filters learned with FFCSC [7] (left) and with our proposed method (right) are shown. Note that our method achieves lower objective.

5. Experiments

5.1. CoGD for Toy problem

We first use a toy problem as an example to illustrate the superiority of our algorithm based on the optimization path. In Fig. 2, we implement CoGD algorithm on three widely used optimization methods, *i.e.* , 'SGD', 'Momentum' and 'Adam'. We solve the beale function [4] with additional constraint $F(x_1,x_2) = beale(x_1,x_2) + \|x_1\| + x_2^2$. This function has the same form as Eq. 1 and can be regraded as a bilinear problem with the part x_1x_2. The learning rate η_2 is set as 0.001, 0.005, 0.1 for 'SGD', 'Momentum' and 'Adam' respectively. The threshold α_{x_1} and α_{x_2} for CoGD are set as 1, 0.5. $\beta = 0.001\eta_2\mathbf{c}^t$ with $\frac{\partial x_2}{\partial x_1} = \frac{\Delta x_2}{\Delta x_1}$, where Δ denotes the difference of variable over the epoch. $\frac{\Delta x_2}{\Delta x_1} = \mathbf{1}$, when Δx_2 or x_2 approaches to be zero. The total iterations is 200. It can be seen that algorithms with CoGD have short optimization paths compared with their counterparts, which show that CoGD facilitates an efficient and sufficient training.

5.2. Convolutional Sparse Coding

Datasets. We evaluate our method on two publicly available datasets: the fruit dataset [30] and the city dataset [30].

[4]$beate(x_1,x_2) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.62 - x_1 + x_1x_2^3)^2$.

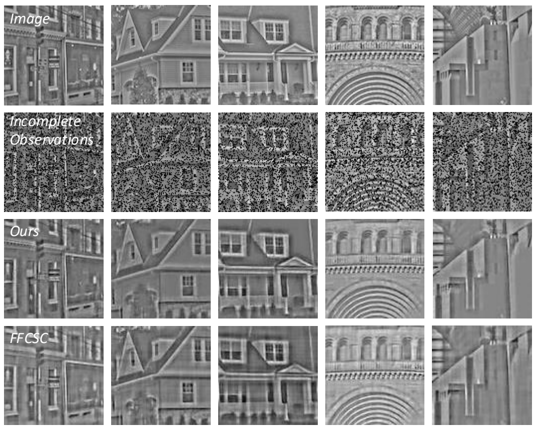

Figure 4: Inpainting for normalized city dataset. From top to bottom: original images, incomplete observations, reconstructions with FFCSC [7], reconstructions with our proposed algorithm.

They are commonly used to evaluate CSC methods [30, 7] and each of them consists of ten images with 100×100 resolution.

To evaluate the quality of the reconstructed images, we use two common metrics, the peak signal-to-noise ratio (PSNR, unit:dB) and the structural similarity (SSIM). The higher PSNR and SSIM are, the better visual quality is achieved for the reconstructed image.

Implementation Details. Our model is implemented based on [4], by only using the projection function to achieve a better convergence. We use 100 filters with size 11×11. $\alpha_{\mathbf{x}}$ is set as the mean of $\|\mathbf{x}_k\|_1$. $\alpha_{\mathbf{A}}$ is calculated by the median of the sorted result of $\mathbf{A}_k$. For a fair comparison, we use the same hyperparameters (η_2) in both our method and [4]. Similarly, $\beta = 0.1\eta_2\mathbf{c}^t$ with $\frac{\partial \mathbf{A}}{\partial x_j} = \frac{\Delta \mathbf{A}}{\Delta x_j}$.

Results. We evaluate CSC with our algorithm for two tasks, including image reconstruction and image inpainting.

For **image reconstruction**, we reconstruct the images on fruit and city datasets respectively. We train 100 11×11-sized filters, and compare these filters with FFCSC [7]. Fig. 3 shows the resulting filters after convergence within the same 20 iterations. Comparing our method to FFCSC, we observe that our method converges with a lower loss. Furthermore, we compare PSNR and SSIM of our method with FFCSC in Tab. 1. In most cases, our method achieves better PSNR and SSIM. The average PSNR and SSIM improvements are 1.01 db and 0.003.

For **image inpainting**, we randomly sample the data with a 75% subsampling rate, to obtain the incomplete data. Similar to [7], we test our method on contrast-normalized images. We first learn filters from all the incomplete data with the help of $\mathbf{M}$, and then reconstruct that incomplete data by fixing the learned filters. We show some inpainting results of normalized data in Fig. 4. Moreover, to compare with FFCSC, inpainting results on the fruit and city datasets are shown in Tab. 2. Note that our method achieves better PSNR and SSIM in all cases. The average PSNR and SSIM improvements are an impressive 1.47 db and 0.016.

5.3. Network Pruning

We have evaluated CoGD algorithm on network pruning using the CIFAR-10 and ILSVRC12 ImageNet datasets for classification. ResNets and MobileNetV2 are the backbone networks to build our framework.

5.3.1 Datasets and Implementation Details

Datasets: CIFAR-10 is a natural image classification dataset containing a training set of $50,000$ and a testing set of $10,000$ 32×32 color images distributed over 10 classes including: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

The ImageNet object classification dataset is more challenging due to its large scale and greater diversity. There are $1,000$ classes, 1.2 million training images and 50k validation images.

Implementation Details. We use PyTorch to implement our method with 3 NVIDIA TITAN V and 2 Tesla V100 GPUs. The weight decay and the momentum are set to 0.0002 and 0.9 respectively. The hyper-parameter $\lambda_{\mathbf{m}}$ is selected through cross validation in the range $[0.01, 0.1]$ for ResNet and MobileNetv2. The drop rate is set to 0.1. The other training parameters are described on a per experiment basis.

To better demonstrate our method, we denote CoGD-a as an approximated pruning rate of $(1-a)\%$ for coresponding channels. a is associated with the threshold α_W, which is given by its sorted result. For example, if $a = 0.5$, α_W is the median of sorted result. $\alpha_{\mathbf{m}}$ is set to be 0.5 for easy implementation. Similarly, $\beta = 0.001\eta_2\mathbf{c}^t$ with $\frac{\partial \mathbf{W}}{\partial m_j} = \frac{\Delta \mathbf{W}}{\Delta m_j}$. Note that our training cost is similar to [14], since we use our method once per epoch without extra cost.

5.3.2 Experiments on CIFAR-10

We evaluated our method on CIFAR-10 for two popular networks, ResNets and MobileNetV2. The stage kernels are set to 64-128-256-512 for ResNet-18 and 16-32-64 for ResNet-110. For all networks, we add the soft mask only after the first convolutional layer within each block to prune the output channel of the current convolutional layer and input channel of next convolutional layer, simultaneously. The mini-batch size is set to be 128 for 100 epochs, and the initial learning rate is set to 0.01, scaled by 0.1 over 30 epochs.

Table 1: Reconstruction results for filters learned with our proposed method and with FFCSC [7]. With the exception of 6 images, our method results in better PSNR and SSIM.

Dataset	Fruit	1	2	3	4	5	6	7	8	9	10	Average
PSNR (dB)	[7]	30.90	**29.52**	26.90	28.09	22.25	**27.93**	**27.10**	27.05	23.65	23.65	26.70
	CoGD	**31.46**	29.12	**27.26**	**28.80**	**25.21**	27.35	26.25	**27.48**	**25.30**	**27.84**	**27.60**
SSIM	[7]	0.9706	**0.9651**	0.9625	**0.9629**	0.9433	0.9712	0.9581	0.9524	0.9608	0.9546	0.9602
	CoGD	**0.9731**	0.9648	**0.9640**	0.9607	**0.9566**	**0.9717**	**0.9587**	**0.9562**	**0.9642**	**0.9651**	**0.9635**
Dataset	City	1	2	3	4	5	6	7	8	9	10	Average
PSNR (dB)	[7]	30.11	27.86	**28.91**	**26.70**	27.85	28.62	18.63	28.14	27.20	25.81	26.98
	CoGD	**30.29**	**28.77**	28.51	26.29	**28.50**	**30.36**	**21.22**	**29.07**	**27.45**	**30.54**	**28.10**
SSIM	[7]	0.9704	**0.9660**	**0.9703**	0.9624	0.9619	0.9613	0.9459	0.9647	0.9531	0.9616	0.9618
	CoGD	**0.9717**	**0.9660**	0.9702	**0.9628**	**0.9627**	**0.9624**	**0.9593**	**0.9663**	**0.9571**	**0.9632**	**0.9642**

Table 2: Inpainting results for filters learned with our proposed method and with FFCSC [7]. All reconstructions are performed for 75% subsampling. Our method achieves better PSNR and SSIM in all cases.

Dataset	Fruit	1	2	3	4	5	6	7	8	9	10	Average
PSNR (dB)	[7]	25.37	24.31	25.08	24.27	23.09	25.51	22.74	24.10	19.47	22.58	23.65
	CoGD	**26.37**	**24.45**	**25.19**	**25.43**	**24.91**	**27.90**	**24.26**	**25.40**	**24.70**	**24.46**	**25.31**
SSIM	[7]	0.9118	0.9036	0.9043	0.8975	0.8883	0.9242	0.8921	0.8899	0.8909	0.8974	0.9000
	CoGD	**0.9452**	**0.9217**	**0.9348**	**0.9114**	**0.9036**	**0.9483**	**0.9109**	**0.9041**	**0.9215**	**0.9097**	**0.9211**
Dataset	City	1	2	3	4	5	6	7	8	9	10	Average
PSNR (dB)	[7]	26.55	24.48	25.45	21.82	24.29	25.65	19.11	25.52	22.67	27.51	24.31
	CoGD	**26.58**	**25.75**	**26.36**	**25.06**	**26.57**	**24.55**	**21.45**	**26.13**	**24.71**	**28.66**	**25.58**
SSIM	[7]	0.9284	0.9204	0.9368	0.9056	0.9193	0.9202	0.9140	0.9258	0.9027	0.9261	0.9199
	CoGD	**0.9397**	**0.9269**	**0.9433**	**0.9289**	**0.9350**	**0.9217**	**0.9411**	**0.9298**	**0.9111**	**0.9365**	**0.9314**

Table 3: Pruning results of ResNet-18/110 and MobilenetV2 on CIFAR-10. M means million (10^6).

Model	FLOPs (M)	Reduction	Accuracy/+FT (%)
ResNet-18[5]	555.42	-	95.31
CoGD-0.5	274.74	0.51×	95.11/95.30
ResNet-110[5]	252.89	-	93.68
GAL-0.1[14]	205.70	0.20×	92.65/93.59
GAL-0.5[14]	130.20	0.49×	92.65/92.74
CoGD-0.5	95.03	0.62×	93.31/93.45
CoGD-0.8	135.76	0.46×	93.42/93.66
MobileNet-V2[22]	91.15	-	94.43
CoGD-0.5	50.10	0.45×	94.25/-

Fine-tuning. For fine-tuning, we only reserve the student model. According to the 'zero's in each soft mask, we remove the corresponding output channels of current convolutional layer and corresponding input channels of next convolutional layer. We then obtain a pruned network with fewer parameters and FLOPs. We use the same batch size of 256 for 60 epochs as in training. The initial learning rate is changed to be 0.1 and scaled by 0.1 over 15 epochs. Note that a similar fine-tuning strategy was used in GAL.

Results. Two kinds of networks are tested on the CIFAR-10 database - ResNets and MobileNet-V2.

For **ResNets**, as is shown in Tab. 3, our method achieves comparable results. Compared to the pre-trained network for ResNet-18 with 95.31% accuracy, CoGD-0.5 achieves a $0.51\times$ FLOPs reduction with only a 0.01% drop in accuracy. Among other structured pruning methods for ResNet-110, CoGD-0.5 has a larger FLOPs reduction than GAL-0.1 ($95.03M$ v.s. $205.70M$), but with the similar accuracy

Table 4: Pruning results of ResNet-50 on ImageNet. B means billion (10^9).

Model	FLOPs (B)	Reduction	Accuracy/+FT (%)
ResNet-50[5]	4.09	-	76.24
ThiNet-50[17]	1.71	0.58×	71.01
ThiNet-30[17]	1.10	0.73×	68.42
CP[6]	2.73	0.33×	72.30
GDP-0.5[13]	1.57	0.62×	69.58
GDP-0.6[13]	1.88	0.54×	71.19
SSS-26[9]	2.33	0.43×	71.82
SSS-32[9]	2.82	0.31×	74.18
RBP[32]	1.78	0.56×	71.50
RRBP[32]	1.86	0.55×	73.00
GAL-0.1[14]	2.33	0.43×	-/71.95
GAL-0.5[14]	1.58	0.61×	-/69.88
CoGD-0.5	2.67	0.35×	75.15/75.62

(93.45% v.s. 93.59%). These results demonstrate that our method is able to prune the network efficiently and generate a more compressed model with higher performance.

For **MobileNetV2**, the pruning result for MobilnetV2 is summarized in Tab. 3. Compared to the pre-trained network, CoGD-0.5 achieves a $0.45\times$ FLOPs reduction with a 0.18% accuracy drop. This result indicates that CoGD is easily employed on efficient networks with depth-wise separable convolution, which is worth exploring in practical applications.

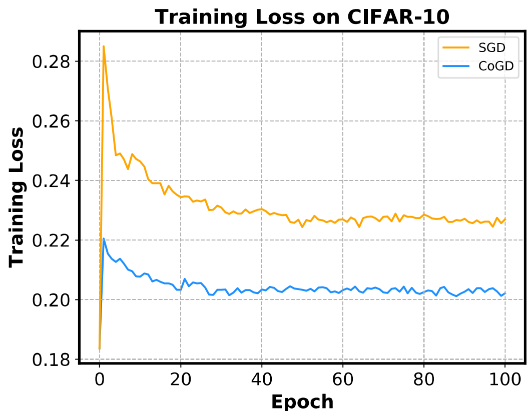

Figure 5: Training loss on CIFAR-10 with CoGD or SGD.

5.3.3 Experiments on ILSVRC12 ImageNet

For ILSVRC12 ImageNet, we test our CoGD based on ResNet-50. We train the network with the batch size of 256 for 60 epochs. The initial learning rate is set to 0.01 scaled by 0.1 over 15 epochs. Other hyperparameters follow the setting on CIFAR-10. The fine-tuning process follows the setting on CIFAR-10 with the initial learning rate 0.00001.

Tab. 4 shows that CoGD achieves the state-of-the-art performance on the ILSVRC12 ImageNet. For ResNet-50, CoGD-0.5 further shows a $0.35\times$ FLOPs reduction while achieving only a 0.62% drop in the accuracy.

5.3.4 Ablation Study

We use ResNet-18 on CIFAR-10 for ablation study to evaluate the effectiveness of our method.

Effect on CoGD. We train the pruned network with and without CoGD by using the same parameters. As shown in Tab. 5, we obtain an error rate 4.70% and a $0.51\times$ FLOPs reduction with CoGD, compared to the error rate is 5.19% and a $0.32\times$ FLOPs reduction without CoGD, which validates the effectiveness of CoGD.

Effect on synchronous convergence. As is shown in Fig 5, the training curve shows that the convergence of CoGD is similar to GAL with SGD-based optimization within an epoch, especially for the last epochs when converging in a similar speed. We theoretically derive our method within the gradient descent framework, which provides a solid foundation for the convergence analysis of our method. The main differences between SGD and CoGD are twofold: (1) we change the initial point for each epoch; (2) we explore the coupling relationship between the hidden factors to improve a bilinear model within the gradient descent framework. They cause little difference between two methods on the convergence.

In Fig. 6, we show the convergence in a synchronous manner of the 4th layer's variables when pruning CNNs. For better visualization, the learning rate of $\mathbf{m}$ is enlarged to 100 times. On the curves, we can observe that the two variables converge synchronously while avoiding that either variable quickly gets stuck into local minimum, which validates that CoGD avoids vanishing gradient for the coupled variables.

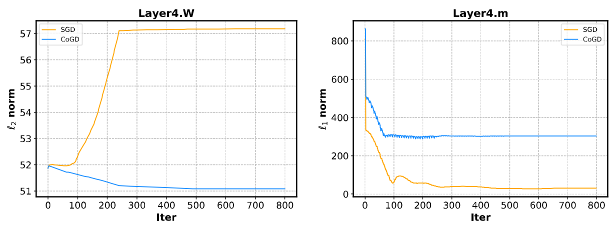

Figure 6: The left and the right are obtained based on SGD and CoGD-0.5 on CIFAR-10. With CoGD, the two variables converge synchronously while avoiding that either variable quickly gets stuck into local minimum, which validates that CoGD avoids vanishing gradient for the coupled variables.

Table 5: Pruning results on CIFAR-10 with CoGD or SGD. M means million (10^6).

Optimizer	Accuracy (%)	FLOPs / Baseline (M)
SGD	94.81	376.12 / 555.43
CoGD	95.30	274.74 / 555.43

6. Conclusion

In this paper, we developed an optimization framework for the bilinear problem with sparsity constraints. Since previous gradient based algorithms ignore the coupled relationship between variables and lead to asynchronous gradient descent, we introduced CoGD which coordinates the gradient of hidden variables. We applied our algorithm on CSC and CNNs to solve image reconstruction, image inpainting and network pruning. Experiments on different tasks demonstrate that our CoGD outperforms previous algorithms.

7. Acknowledgements

Baochang Zhang is also with Shenzhen Academy of Aerospace Technology, Shenzhen 100083, China. National Natural Science Foundation of China under Grant 61672079, in part by Supported by Shenzhen Science and Technology Program (No.KQTD2016112515134654).

References

[1] Abdelrahman Abdelhamed, Marcus A Brubaker, and Michael S Brown. Noise flow: Noise modeling with conditional normalizing flows. In *ICCV*, pages 3165–3173, 2019. 1

[2] Hilton Bristow, Anders Eriksson, and Simon Lucey. Fast convolutional sparse coding. In *CVPR*, pages 391–398, 2013. 1

[3] Alessio Del Bue, Joao Xavier, Lourdes Agapito, and Marco Paladini. Bilinear modeling via augmented lagrange multipliers (balm). *PAMI*, 34(8):1496–1508, 2011. 1

[4] Shuhang Gu, Wangmeng Zuo, Qi Xie, Deyu Meng, Xiangchu Feng, and Lei Zhang. Convolutional sparse coding for image super-resolution. In *ICCV*, pages 1823–1831, 2015. 3, 6

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 7

[6] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, pages 1398–1406, 2017. 4, 5, 7

[7] Felix Heide, Wolfgang Heidrich, and Gordon Wetzstein. Fast and flexible convolutional sparse coding. In *CVPR*, pages 5135–5143, 2015. 1, 2, 3, 5, 6, 7

[8] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *ECCV*, pages 304–320, 2018. 2, 4, 5

[9] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *ECCV*, pages 304–320, 2018. 7

[10] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 4

[11] Yanghao Li, Naiyan Wang, Jiaying Liu, and Xiaodi Hou. Factorized bilinear models for image recognition. In *ICCV*, pages 2079–2087, 2017. 2

[12] Qiyu Liao, Dadong Wang, Hamish Holewa, and Min Xu. Squeezed bilinear pooling for fine-grained visual categorization. In *ICCV Workshop*, October 2019. 1

[13] Shaohui Lin, Rongrong Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and Baochang Zhang. Accelerating convolutional networks via global & dynamic filter pruning. In *IJCAI*, pages 2425–2432, 2018. 7

[14] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *CVPR*, 2019. 2, 4, 5, 6, 7

[15] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear cnn models for fine-grained visual recognition. In *ICCV*, pages 1449–1457, 2015. 2

[16] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, pages 2736–2744, 2017. 1, 2

[17] Jianhao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, pages 5068–5076, 2017. 4, 7

[18] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11(Jan):19–60, 2010. 3

[19] Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014. 3

[20] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019. 4

[21] Kaare Brandt Petersen, Michael Syskind Pedersen, et al. The matrix cookbook. *Technical University of Denmark*, 7(15):510, 2008. 3

[22] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, pages 4510–4520, 2018. 7

[23] Yumin Suh, Jingdong Wang, Siyu Tang, Tao Mei, and Kyoung Mu Lee. Part-aligned bilinear representations for person re-identification. In *ECCV*, pages 402–419, 2018. 2

[24] Brendt Wohlberg. Efficient convolutional sparse coding. In *ICASSP*, pages 7173–7177, 2014. 1

[25] Linlin Yang, Ce Li, Jungong Han, Chen Chen, Qixiang Ye, Baochang Zhang, Xianbin Cao, and Wanquan Liu. Image reconstruction via manifold constrained convolutional sparse coding for image sets. *JSTSP*, 11(7):1072–1081, 2017. 1

[26] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *ICLR*, 2018. 2, 4, 5

[27] Naoto Yokoya, Jocelyn Chanussot, and Akira Iwasaki. Generalized bilinear model based nonlinear unmixing using semi-nonnegative matrix factorization. In *IEEE International Geoscience and Remote Sensing Symposium*, pages 1365–1368, 2012. 1

[28] Sean I. Young, Aous T. Naman, Bernd Girod, and David Taubman. Solving vision problems via filtering. In *ICCV*, October 2019. 1

[29] Zhou Yu, Jun Yu, Jianping Fan, and Dacheng Tao. Multi-modal factorized bilinear pooling with co-attention learning for visual question answering. In *ICCV*, pages 1821–1830, 2017. 2

[30] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *CVPR*, pages 2528–2535. IEEE, 2010. 5, 6

[31] Lei Zhang, Meng Yang, Xiangchu Feng, Yi Ma, and David Zhang. Collaborative representation based classification for face recognition. *arXiv preprint arXiv:1204.2358*, 2012. 2

[32] Yuefu Zhou, Ya Zhang, Yanfeng Wang, and Qi Tian. Accelerate cnn via recursive bayesian pruning. In *ICCV*, pages 3306–3315, 2019. 7