applied sciences

MDPI

Article

A Training Method for Low Rank Convolutional Neural Networks Based on Alternating Tensor Compose-Decompose Method

Sukho Lee [1], Hyein Kim [2], Byeongseon Jeong [3] and Jungho Yoon [2,*]

[1] Division of Computer Engineering, Dongseo University, Busan 47011, Korea; petrasuk@gmail.com
[2] Department of Mathematics, Ewha University, Seoul 03760, Korea; kshiy0215@gmail.com
[3] Ewha Institute of Mathematical Sciences, Ewha University, Seoul 03760, Korea; bjeong_ewha@ewha.ac.kr
* Correspondence: yoon@ewha.ac.kr; Tel.: +82-2-3277-2293

Abstract: Over the past decade, deep learning-based computer vision methods have been shown to surpass previous state-of-the-art computer vision techniques in various fields, and have made great progress in various computer vision problems, including object detection, object segmentation, face recognition, etc. Nowadays, major IT companies are adding new deep-learning-based computer technologies to edge devices such as smartphones. However, since the computational cost of deep learning-based models is still high for edge devices, research is being actively carried out to compress deep learning-based models while not sacrificing high performance. Recently, many lightweight architectures have been proposed for deep learning-based models which are based on low-rank approximation. In this paper, we propose an alternating tensor compose-decompose (ATCD) method for the training of low-rank convolutional neural networks. The proposed training method can better train a compressed low-rank deep learning model than the conventional fixed-structure based training method, so that a compressed deep learning model with higher performance can be obtained in the end of the training. As a representative and exemplary model to which the proposed training method can be applied, we propose a rank-1 convolutional neural network (CNN) which has a structure alternatively containing 3-D rank-1 filters and 1-D filters in the training stage and a 1-D structure in the testing stage. After being trained, the 3-D rank-1 filters can be permanently decomposed into 1-D filters to achieve a fast inference in the test time. The reason that the 1-D filters are not being trained directly in 1-D form in the training stage is that the training of the 3-D rank-1 filters is easier due to the better gradient flow, which makes the training possible even in the case when the fixed structured network with fixed consecutive 1-D filters cannot be trained at all. We also show that the same training method can be applied to the well-known MobileNet architecture so that better parameters can be obtained than with the conventional fixed-structure training method. Furthermore, we show that the 1-D filters in a ResNet like structure can also be trained with the proposed method, which shows the fact that the proposed method can be applied to various structures of networks.

Keywords: deep learning; convolutional neural networks; low rank; deep compression; MobileNet

check for updates

Citation: Lee, S.; Kim, H.; Jeong, B.; Yoon, J. A Training Method for Low Rank Convolutional Neural Networks Based on Alternating Tensor Compose-Decompose Method. *Appl. Sci.* **2021**, *11*, 643. https://doi.org/10.3390/app11020643

Received: 17 October 2020
Accepted: 8 January 2021
Published: 11 January 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Deep learning-based computer vision shows good performance in various computer vision areas such as image segmentation [1,2], image synthesis [3], facial recognition [4], classification [5], person re-identification [6], and object detection [7,8]. However, in spite of the remarkable achievements in difficult computer vision tasks, conventional deep convolutional neural networks (CNNs) use a high number of parameters which limits their use on devices with limited resources such as smartphones, embedded systems, etc. Even though it has been known that there exist a lot of redundancy between the parameters and the feature maps in deep models, over-parametrized CNN models are used due to

the reason that over-parametrization makes the training of the network easier as has been shown in the experiments in [9]. The reason for this phenomenon is believed to be due to a better gradient flow in over-parametrized networks.

Meanwhile, it has been shown in [10] that even with the use of regularization methods, there still exists excessive capacity in trained networks, which again implies the fact that the redundancy between the parameters is still large. Therefore, many researches focus on finding a better network structure so that the parameters can be expressed in a structured subspace with smaller number of coefficients. The research topic on compressing large-scale deep learning models is increasing in importance as it is necessary to use compressed deep learning models in edge devices such as smartphones and IoT devices. While early works focused on compressing the parameters of pre-trained large scale deep learning models [11–22], studies are also actively under way to limit the number of parameters by proposing small networks in the first place [23–37]. Most recently, researches have prevailed on how to efficiently use these compressed models on edge devices [38–41]. We will provide a detailed overview of these research trends in Section 2.

In this paper, we propose a training method for the training of low-rank convolutional neural networks, which we call the alternating tensor compose-decompose (ATCD) method. The proposed training method can better train compressed low rank models than existing training methods, thus obtaining a compressed deep learning model with higher performance. In general, when training deep learning models, the same structure of the neural network is used during the training and the testing stages. Conventional tensor decomposing networks are trained with the fixed-structure based training method, i.e., they are trained in the decomposed form. We call the conventional training method the fixed-structure based training method. In comparison, the proposed training method do not use a fixed structure of neural network in the training stage, but allows the tensors to be alternatingly composed and decomposed so that a better gradient flow can flow through the tensors in the backpropagation step. This better gradient flow results in better parameter values than with conventional training method so that the compressed model can achieve a higher performance. As an example of the proposed training method, we apply it to the rank-1 CNN, where the rank-1 CNN is iteratively and alternatingly composed into a 3-D rank-1 CNN structure and decomposed into 1-D vectors in the training stage, where the 3-D rank-1 filters are constructed by the outer products of the 1-D vectors. The number of parameters in the 3-D rank-1 filters are the same as in the 3-D filters in standard CNNs, allowing a good gradient flow in the backpropagation stage. The difference with the backpropagation stage in standard CNNs is that the gradient flow flows also through the 1-D vectors from which the 3-D rank-1 filters are constructed, updating the parameters in the 1-D vectors also. After the backpropagation step, the 3-D filters lose their rank-1 property. However, at the next composition step, the parameters in the 3-D filters are updated again by the outer product operation to be projected onto the rank-1 subspace. By iterating this two-step update, all the 3-D filters in the network are trained to minimize the loss function while maintaining their rank-1 property. This is different from approaches which try to approximate the trained filters by low rank approximation after the training has been finished, e.g., like the low rank approximation in [14] or from approaches which use the same fixed CNN structure both in the training and the testing stages. The composition operation is included in the training phase in our network, which directs the parameter update in a different direction from that of standard CNNs, directing the solution to live on a rank-1 subspace.

In the testing phase, we do not need the tensor composing stage anymore, and the 3-D rank-1 filters can be permanently decomposed into 1-D filters. So in the testing stage, the rank-1 CNN is now reconstructed into a 1-D rank-1 CNN structure with the trained 1-D vectors used as the 1-D filters. So the rank-1 CNN has the same accuracy as the 3-D rank-1 CNN, but has the same inference speed as the 1-D rank-1 CNN, i.e., the inference speed is exactly the same as that of the Flattened network. Moreover, with the proposed method, the network can be trained even in the case when the Flattened network cannot be trained

at all. In other words, the proposed training method can be applied to train networks with very limited gradient paths due to the low rank property which cannot be trained with conventional training methods.

We also show how the same training method can be applied to the well-known MobileNet. We first show how the channel-wise filters can be expressed as a linear combination of low rank filters, and then show how the proposed alternating tensor compose-decompose (ATCD) training method can be applied to the training of the low rank filters. The low rank filters are composed into the MobileNet structure again at the end of the training. Thereby, better parameters are obtained than with conventional training with fixed MobileNet structure.

2. Related Works

In this section, we summarize the works related to our work in accordance with the evolving trend of research in this field. However, it should be noted that our work is somewhat unique and different from the related works in the aspect that we did not compress the parameters of pre-trained models or propose a new model architecture, but propose a new training method to train existing factorized structures to have better parameter values.

2.1. Works on Compressing the Parameters of Pre-Trained CNNs

Early works on compressing the CNN focused on how to compress the pre-trained parameters without loss of information. As has been well summarized in [42], researches on the compression of deep models can be categorized into works which try to eliminate unnecessary weight parameters [11,12], works which try to compress the parameters by projecting them onto a low rank subspace [13–16], and works which try to group similar parameters into groups and represent them by representative features [18–22]. These works follow the common framework of first training the original uncompressed CNN model by back-propagation to obtain the uncompressed parameters, and then trying to find a compressed expression for these parameters to construct a new compressed CNN model.

2.2. Works on Designing a Compressed Model

Compared to the works of compressing the pre-trained parameters, researches which try to restrict the number of parameters in the first place by proposing small networks are also actively in progress. However, as mentioned above, the reduction in the number of parameters changes the gradient flow, so the networks have to be designed carefully to achieve a trained network with good performance. For example, MobileNets [23] and Xception networks [24] use depthwise separable convolution filters, while the SqueezeNet [25] uses a bottleneck approach to reduce the number of parameters. MobileNet was modified in version 2 model using inverted residuals [26]. Recently, Google announced the EfficientNet [27] which scales up the MobileNet and the ResNet to obtain a new family of efficient CNN models, while CondenseNet [28] and ShuffleNet [29] are using group convolutions to reduce the number of convolutions. Other models use 1-D filters to reduce the size of networks such as the highly factorized Flattened network [30], or the models in [31] where 1-D filters are used together with other filters of different sizes. Recently, Google's Inception model has also adopted 1-D filters in version 4 [43]. One difficulty in using 1-D filters is that 1-D filters are not easy to train, and therefore, they are used only partially like in the Google's Inception model, or in the models in [31] etc., except for the Flattened network which is constituted of consecutive 1-D filters only. However, only three layers of 1-D filters are used in the experiments in [30], which is maybe due to the difficulty of training 1-D filters with many layers.

Until now, all the efficient neural network architectures have been developed manually by human experts. Though still in the early stage, there are researches ongoing to automatically searching for architectures that are efficient and satisfy the resource or computation constraints [32–37]. However, it is known that such an automatic neural architecture search

is extremely difficult, and therefore, in practice, the manually well-designed architectures are still widely used.

2.3. Works on Edge AI

Now that many efficient and compressed CNN architectures have been proposed, many researchers and major IT companies are focusing on how to shift these compression models to edge devices as customers spend more time on mobile devices [38–41,44–48]. Particularly, in [38], Eshratifar et al. propose an efficient training for intelligent mobile cloud computing services, while in [39] Li et al. proposed how to accelerate the inference in DNN via edge computing. In [40], a deep learning architecture for intelligent mobile cloud computing services called BottleNet is proposed, which reduces the feature size needed to be sent to the cloud. Furthermore, Bateni and Lie propose a timing-predictable runtime system that is able to guarantee deadlines of DNN workloads via efficient approximation [41]. It is believed that Edge AI research and algorithms on both commercial and academic laboratories are expected to be very active in the next three to five years.

3. Preliminaries for the Proposed Method

The following works have to be understood to understand the proposed method. The work of bilateral-projection based 2-D principal component analysis (B2DPCA) gave us the insight for bilateral filters and the tensor compose-decompose procedure, which we utilized to train the rank-1 Net.

3.1. Bilateral-Projection Based 2DPCA

In [49], a bilateral-projection based 2-D principal component analysis (B2DPCA) has been proposed, which minimizes the following energy functional:

$$[\mathbf{P}_{opt}, \mathbf{Q}_{opt}] = \underset{\mathbf{P},\mathbf{Q}}{\operatorname{argmin}} \|\mathbf{X} - \mathbf{P}\mathbf{C}\mathbf{Q}^T\|_F^2, \tag{1}$$

where $\mathbf{X} \in R^{n\times m}$ is the two dimensional image, $\mathbf{P} \in R^{m\times l}$ and $\mathbf{Q} \in R^{n\times r}$ are the left- and right- multiplying projection matrices, respectively, and $\mathbf{C} = \mathbf{P}^T\mathbf{X}\mathbf{Q}$ is the extracted feature matrix for the image $\mathbf{X}$. The optimal projection matrices $\mathbf{P}_{opt}$ and $\mathbf{Q}_{opt}$ are simultaneously constructed, where $\mathbf{P}_{opt}$ projects the column vectors of $\mathbf{X}$ to a subspace, while $\mathbf{Q}_{opt}$ projects the row vectors of $\mathbf{X}$ to another one. It has been shown in [49], that the advantage of the bilateral projection over the unilateral-projection scheme is that $\mathbf{X}$ can be represented effectively with smaller number of coefficients than in the unilateral case, i.e., a small-sized matrix $\mathbf{C}$ can well represent the image $\mathbf{X}$. This means that the bilateral-projection effectively removes the redundancies among both rows and columns of the image. Furthermore, since

$$\begin{aligned}\mathbf{C} = \mathbf{P}^T\mathbf{X}\mathbf{Q} &= \begin{bmatrix} \mathbf{p}_1^T\mathbf{X}\mathbf{q}_1 & \mathbf{p}_1^T\mathbf{X}\mathbf{q}_2 & \cdots & \mathbf{p}_1^T\mathbf{X}\mathbf{q}_r \\ \mathbf{p}_2^T\mathbf{X}\mathbf{q}_1 & \mathbf{p}_2^T\mathbf{X}\mathbf{q}_2 & \cdots & \mathbf{p}_2^T\mathbf{X}\mathbf{q}_r \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{p}_l^T\mathbf{X}\mathbf{q}_1 & \mathbf{p}_l^T\mathbf{X}\mathbf{q}_2 & \cdots & \mathbf{p}_l^T\mathbf{X}\mathbf{q}_r \end{bmatrix} \\ &= \begin{bmatrix} <\mathbf{X}, \mathbf{p}_1\mathbf{q}_1^T> & <\mathbf{X}, \mathbf{p}_1\mathbf{q}_2^T> & \cdots & <\mathbf{X}, \mathbf{p}_1\mathbf{q}_r^T> \\ <\mathbf{X}, \mathbf{p}_2\mathbf{q}_1^T> & <\mathbf{X}, \mathbf{p}_2\mathbf{q}_2^T> & \cdots & <\mathbf{X}, \mathbf{p}_2\mathbf{q}_r^T> \\ \vdots & \vdots & \vdots & \vdots \\ <\mathbf{X}, \mathbf{p}_l\mathbf{q}_1^T> & <\mathbf{X}, \mathbf{p}_l\mathbf{q}_2^T> & \cdots & <\mathbf{X}, \mathbf{p}_l\mathbf{q}_r^T> \end{bmatrix},\end{aligned} \tag{2}$$

it can be seen that the components of $\mathbf{C}$ are the 2-D projections of the image $\mathbf{X}$ onto the 2-D planes $\mathbf{p}_1\mathbf{q}_1^T$, $\mathbf{p}_1\mathbf{q}_2^T$, ...$\mathbf{p}_l\mathbf{q}_r^T$ made up by the outer products of the column vectors of $\mathbf{P}$ and $\mathbf{Q}$. The 2-D planes have a rank of one, since they are the outer products of two 1-D vectors. Therefore, the fact that $\mathbf{X}$ can be well represented by a small-sized $\mathbf{C}$ also implies the fact that $\mathbf{X}$ can be well represented by a few rank-1 2-D planes, i.e., only a few 1-D

vectors $\mathbf{p}_1, ...\mathbf{p}_l, \mathbf{q}_1, ...\mathbf{q}_r$, where $l << m$ and $r << n$.

In the case of (1), the learned 2-D planes try to minimize the loss function

$$L = \|\mathbf{X} - \mathbf{PCQ}^T\|_F^2, \tag{3}$$

i.e., try to learn to best approximate $\mathbf{X}$. A natural question arises, if good rank-1 2-D planes can be obtained to minimize other loss functions too, e.g., loss functions related to the image classification problem, such as

$$L = \|y_{true} - y(\mathbf{X}, \mathbf{P}, \mathbf{C}, \mathbf{Q})\|_F^2, \tag{4}$$

where y_{true} denotes the true classification label for a certain input image $\mathbf{X}$, and $y(\mathbf{X}, \mathbf{P}, \mathbf{C}, \mathbf{Q})$ is the output of the network constituted by the outer products of the column vectors in the learned matrices $\mathbf{P}$ and $\mathbf{Q}$. In this paper, we extend this case to the rank-1 3-D filter case, where the rank-1 3-D filter is constituted as the outer product of three column vectors from three different learned matrices, and show that good parameters can be learned for the image classification task. Furthermore, these learned rank-1 3-D filters can be decomposed into rank-1 1-D filters for fast inference speed.

3.2. Flattened Convolutional Neural Networks

In [30], the 'Flattened CNN' has been proposed for fast feed-forward execution by separating the conventional 3-D convolution filter into three consecutive 1-D filters. The 1-D filters sequentially convolve the input over different directions, i.e., the lateral, horizontal, and vertical directions. Figure 1 shows the network structure of the Flattened CNN. The Flattened CNN uses the same network structure in both the training and the testing phases. This is in comparison with our proposed model, where we use a different network structure in the training phase as will be seen later.

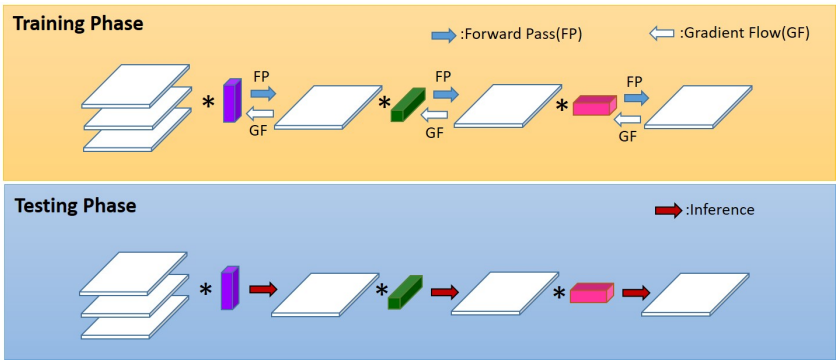

Figure 1. The structure of the Flattened network. The same network structure of sequential use of 1-D filters is used in the training and testing phases. Here, $*$ denotes the convolution operator.

However, the consecutive use of 1-D filters in the training phase makes the training difficult. This is due to the fact that the gradient path becomes longer than in normal CNN, and therefore, the gradient flow vanishes faster while the error accumulates more. Another reason is that the reduction in the number of parameters causes a gradient flow different from that of the standard CNN, which is more difficult to find an appropriate solution for the parameters. This fact coincides with the experiments in [9] which show that the gradient flow in a network with small number of parameters cannot find good parameters. Therefore, a particular weight initialization method has to be used together with this setting. Furthermore, in [30], the networks in the experiments have only three layers of convolution, which is maybe due to the fact of the difficulty in training networks with more layers.

4. Application of the Proposed Training Method to the Rank-1 CNN

As an example of how the tensor composing-decomposing method can be applied to train low-rank CNNs, we apply the proposed training method to the rank-1 CNN which is composed of mere rank-1 convolutional filters. In comparison with other CNN models which use 1-D rank-1 filters, we propose the use of 3-D rank-1 filters($\mathbf{W}$) in the training stage, where the 3-D rank-1 filters are constructed by the outer product of three 1-D vectors, say $\mathbf{p}$, $\mathbf{q}$, and $\mathbf{t}$:

$$\mathbf{W} = \mathbf{p} \otimes \mathbf{q} \otimes \mathbf{t}. \tag{5}$$

This is an extension of the 2-D rank-1 planes used in the B2DPCA, where the 2-D planes are constructed by $\mathbf{W} = \mathbf{p} \otimes \mathbf{q} = \mathbf{p}\mathbf{q}^T$. Figure 2 shows the training and the testing phases of the proposed method. The structure of the proposed network is different for the training phase and the testing phase. In comparison with the Flattened network (Figure 1), in the training phase, the gradient flow first flows through the 3-D rank-1 filters and then through the 1-D vectors. Therefore, the gradient flow is different from that of the Flattened network resulting in a different and better solution of parameters in the 1-D vectors. The solution can be obtained even in large networks with the proposed method, for which the gradient flow in the Flattened network cannot obtain a solution at all. Furthermore, at test time, i.e., at the end of optimization, we can use the 1-D vectors directly as 1-D filters in the same manner as in the Flattened network, resulting in the same inference speed and number of operations as the Flattened network (Figure 2).

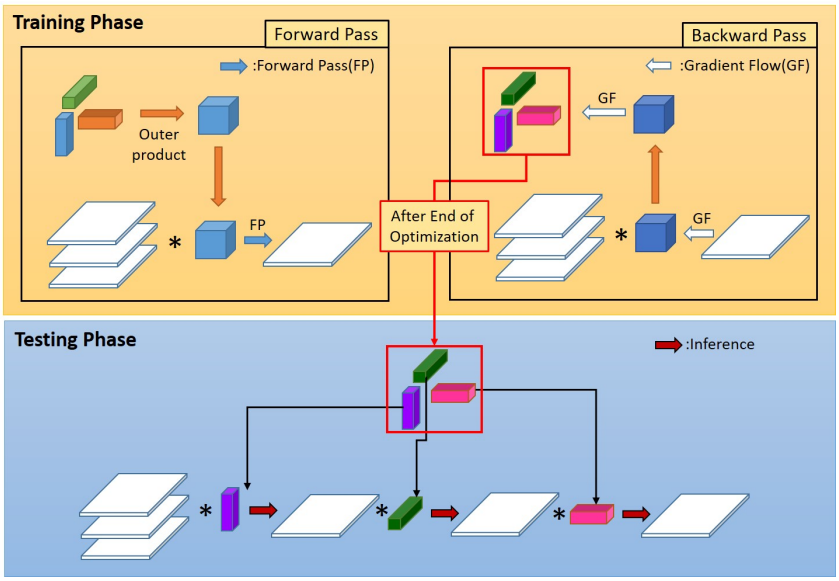

Figure 2. Proposed rank-1 neural network with different network structures in training and testing phases. Here, $*$ denotes the convolution operator.

4.1. Construction of the 3-D Rank-1 Filters

We first observe that a 2-D convolution can be seen as shifting inner products, where each component $y(\mathbf{r})$ at position $\mathbf{r}$ of the output matrix $\mathbf{Y}$ is computed as the inner product of a 2-D filter $\mathbf{W}$ and the image patch $\mathbf{X}(\mathbf{r})$ centered at $\mathbf{r}$:

$$y(\mathbf{r}) = < \mathbf{W}, \mathbf{X}(\mathbf{r}) > . \tag{6}$$

If $\mathbf{W}$ is constructed by the outer product of two 1-D vectors $\mathbf{p}$ and $\mathbf{q}$, i.e., $\mathbf{W} = \mathbf{p} \otimes \mathbf{q} = \mathbf{p}\mathbf{q}^T$, then $\mathbf{W}$ becomes a 2-D rank-1 filter. In this case, it can be observed that

$$y(\mathbf{r}) = < \mathbf{W}, \mathbf{X}(\mathbf{r}) > = < \mathbf{p}\mathbf{q}^T, \mathbf{X}(\mathbf{r}) > = \mathbf{p}^T\mathbf{X}(\mathbf{r})\mathbf{q}. \tag{7}$$

As has been explained in the case of B2DPCA, since $\mathbf{p}$ is multiplied to the rows of $\mathbf{X}(\mathbf{r})$, $\mathbf{p}$ tries to extract the features from the rows of $\mathbf{X}(\mathbf{r})$ which can minimize the loss function. That is, $\mathbf{p}$ searches the rows in all patches $\mathbf{X}(\mathbf{r}), \forall_{\mathbf{r}}$ for some common features which can reduce the loss function, while $\mathbf{q}$ looks for the features in the columns of the patches. In analogy to the B2DPCA, this bilateral projection removes the redundancies among the rows and columns in the 2-D filters.

In convolutional neural networks, the input $\mathbf{X}_{3D}$ and the convolutional filter $\mathbf{W}_{3D}$ are both three dimensional, where the third dimension refers to the depth of the input, i.e., the numer of input channels. In this case, the 3-D rank-1 filter $\mathbf{W}_{3D}$ is constructed by the outer product of three 1-D vectors $\mathbf{p}$, $\mathbf{q}$, and $\mathbf{t}$,

$$\mathbf{W}_{3D} = \mathbf{p} \otimes \mathbf{q} \otimes \mathbf{t}, \tag{8}$$

where the length of $\mathbf{t}$ is the same as the depth of the input $\mathbf{X}_{3d}$. In analogy to the B2DPCA, the 3-D rank-1 filters which are learned by the three dimensional multilateral projection will have less redundancies among the rows, columns, and the channels than the normal 3-D filters in standard CNNs.

The three dimensional convolution of the 3-D rank-1 filter $\mathbf{W}_{3D}$ and $\mathbf{X}_{3D}$ can be expressed by the sum of channel-wise 2-D convolutions. Let $\mathbf{W}_{3D}^{(i)}$ denote the i's 3-D rank-1 filter that results in the i's output channel $\mathbf{Y}^{(i)}$, and $\mathbf{W}_{2D}^{(i)}$ the 2-D rank-1 filter that relates with $\mathbf{Y}^{(i)}$, which is constructed by the outer product of two 1-D vectors $\mathbf{p}_l$ and $\mathbf{q}_r$. We construct a rank-1 2-D filter for each output channel $\mathbf{Y}^{(i)}$,

$$\begin{aligned}
\mathbf{W}_{2D}^{(1)} &= \mathbf{p}_1 \otimes \mathbf{q}_1, \\
\mathbf{W}_{2D}^{(2)} &= \mathbf{p}_1 \otimes \mathbf{q}_2, \\
&\vdots \\
\mathbf{W}_{2D}^{(i)} &= \mathbf{p}_l \otimes \mathbf{q}_r, \\
\mathbf{W}_{2D}^{(i+1)} &= \mathbf{p}_l \otimes \mathbf{q}_{r+1}, \\
&\vdots \\
\mathbf{W}_{2D}^{(q)} &= \mathbf{p}_m \otimes \mathbf{q}_n.
\end{aligned} \tag{9}$$

The total number of 2-D filters is $q = m \times n$, where q is the number of output channels. Then, the 3-D rank-1 filters can be constructed by

$$\mathbf{W}_{3D}^{(i)} = \mathbf{W}_{2D}^{(i)} \otimes \mathbf{t}_i = \mathbf{p}_l \otimes \mathbf{q}_r \otimes \mathbf{t}_i. \tag{10}$$

Furthermore, let $\mathbf{X}^{(j)}$ denote the j's 2-D channel in $\mathbf{X}_{3d}$. Then the 3-D convolution which results in the i's output channel $\mathbf{Y}^{(i)}$ can be expressed as

$$\mathbf{Y}^{(i)} = \mathbf{X} * \mathbf{W}_{3D}^{(i)} = \sum_{j=1}^{N} \mathbf{t}_i[j](\mathbf{X}^{(j)} \circledast \mathbf{W}_{2D}^{(i)}), \tag{11}$$

where $*$ and $\circledast$ denote the 3-D and the 2-D convolution operations, respectively, and $\mathbf{t}_i[j]$ refers to the j's component of the vector $\mathbf{t}_i$. Figure 3 visualizes how the 3-D rank-1 filters are constructed and how they convolve with the 2-D channels in $\mathbf{X}_{3d}$.

As explained above, at test time, we can use the trained 1-D vectors as the 1-D filters, so that in the test time only 1-D convolutions are used. As has been shown in [30], when using only 1-D convolutions, the number of operations reduces to

$$I_x \times I_y \times (C_{out} + f_x + f_y), \tag{12}$$

instead of

$$I_x \times I_y \times C_{out} \times f_x \times f_y, \tag{13}$$

where I_x and I_y are the width and height of the input feature map, respectively, C_{out} is the number of output channels, and f_x and f_y are the width and height of the filter.

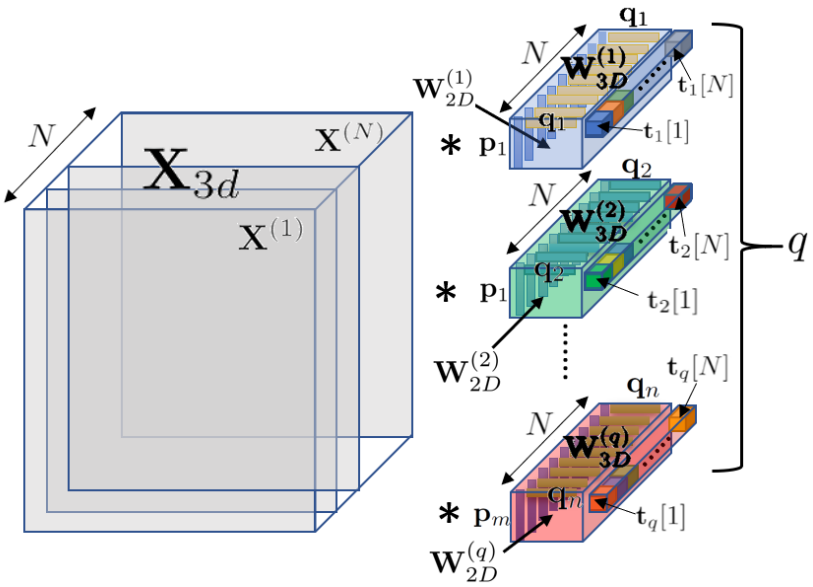

Figure 3. Constitution of the 3-D rank-1 filters in the training phase. The 3-D filters $\mathbf{W}_{3D}^{(1)}, \mathbf{W}_{3D}^{(2)}, ..., \mathbf{W}_{3D}^{(q)}$ that are convolved with the 3-D input $\mathbf{X}_{3d}$ are constructed by the outer products of the 2-D filters $\mathbf{W}_{2D}^{(1)}, \mathbf{W}_{2D}^{(2)}, ..., \mathbf{W}_{2D}^{(q)}$ and the 1-D filters $\mathbf{t}_1, \mathbf{t}_2, ..., \mathbf{t}_q$, respectively, where the 2-D filters are again constructed by the outer products of the 1-D filters $\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_m$ and $\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_n$ according to Equation (9).

4.2. Training Process

Figure 4 explains the training process with the proposed network structure in detail. At every epoch of the training phase, we first take the outer product of the three 1-D vectors $\mathbf{p}$, $\mathbf{q}$, and $\mathbf{t}$. Then, we assign the result of the outer product to the weight values of the 3-D convolution filter, i.e., for every weight value in the 3-D convolution filter $\mathbf{W}$, we assign

$$w_{i,j,k} = p_i q_j t_k, \ \forall_{i,j,k \in \Omega(\mathbf{W})} \tag{14}$$

where, i, j, k correspond to the 3-D coordinates in $\Omega(\mathbf{W})$, the 3-D domain of the 3-D convolution filter $\mathbf{W}$. Since the matrix constructed by the outer product of vectors has always a rank of one, the 3-D convolution filter $\mathbf{W}$ is a rank-1 filter.

During the back-propagation phase, every weight value in $\mathbf{W}$ will be updated by

$$w'_{i,j,k} = w_{i,j,k} - \alpha \frac{\partial L}{\partial w_{i,j,k}}, \tag{15}$$

where $\frac{\partial L}{\partial w_{i,j,k}}$ denotes the gradient of the loss function L with respect to the weight $w_{i,j,k}$, and α is the learning rate. In standard convolutional neural networks, $w'_{i,j,k}$ in (15) is the final updated weight value at each update step. However, the updated filter $\mathbf{W}'$ normally is not a rank-1 filter. This is due to the fact that the update in (15) is done in the direction which considers only the minimizing of the loss function and not the rank of the filter.

With the proposed training network structure, we take a further update step, i.e., we update the 1-D vectors $\mathbf{p}$, $\mathbf{q}$, and $\mathbf{t}$:

$$p'_i = p_i - \alpha \frac{\partial L}{\partial p_i}, \ \forall_{i \in \Omega(\mathbf{p})} \tag{16}$$

$$q'_j = q_j - \alpha \frac{\partial L}{\partial q_j}, \ \forall_{j \in \Omega(\mathbf{q})} \tag{17}$$

$$t'_k = t_k - \alpha \frac{\partial L}{\partial t_k}, \ \forall_{k \in \Omega(\mathbf{t})} \tag{18}$$

Here, $\frac{\partial L}{\partial p_i}$, $\frac{\partial L}{\partial q_j}$, and $\frac{\partial L}{\partial t_k}$ can be calculated as

$$\frac{\partial L}{\partial p_i} = \sum_j \sum_k \frac{\partial L}{\partial w_{i,j,k}} \frac{\partial w_{i,j,k}}{\partial p_i} = \sum_j \sum_k \frac{\partial L}{\partial w_{i,j,k}} q_j t_k, \tag{19}$$

$$\frac{\partial L}{\partial q_j} = \sum_i \sum_k \frac{\partial L}{\partial w_{i,j,k}} \frac{\partial w_{i,j,k}}{\partial q_j} = \sum_i \sum_k \frac{\partial L}{\partial w_{i,j,k}} p_i t_k, \tag{20}$$

$$\frac{\partial L}{\partial t_k} = \sum_i \sum_j \frac{\partial L}{\partial w_{i,j,k}} \frac{\partial w_{i,j,k}}{\partial t_k} = \sum_i \sum_j \frac{\partial L}{\partial w_{i,j,k}} p_i q_j. \tag{21}$$

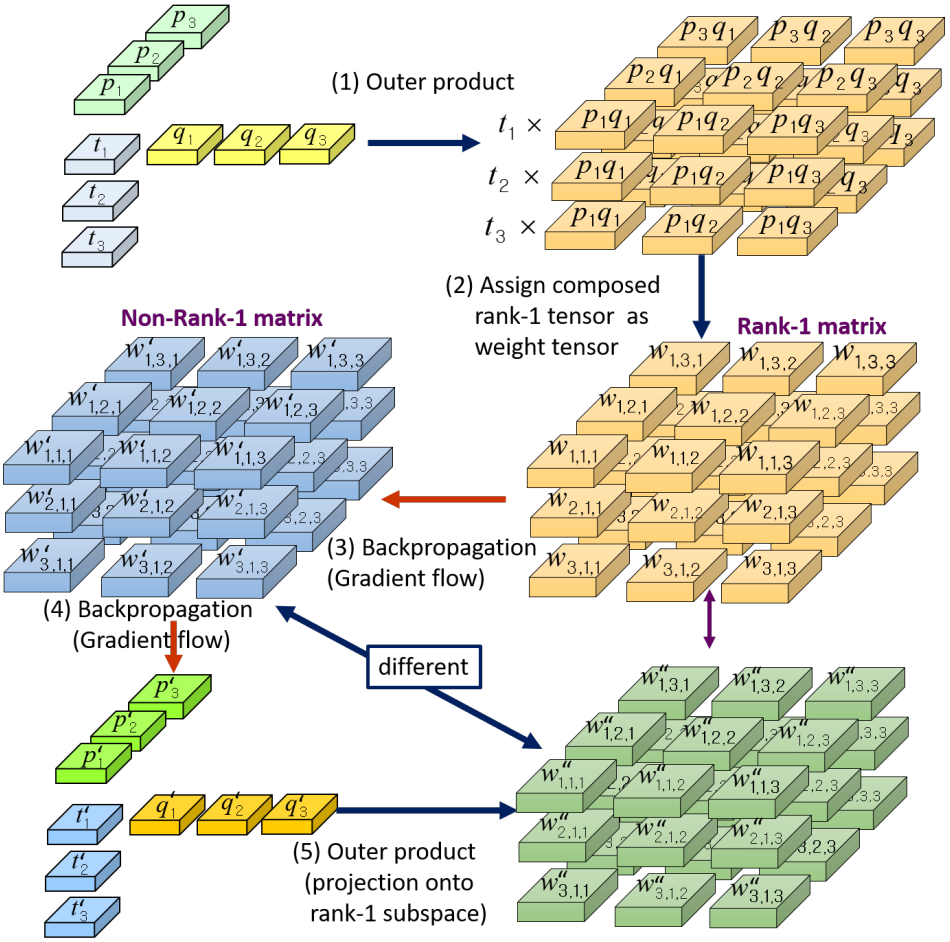

Figure 4. Steps in the training phase of the proposed rank-1 network.

At the next feed forward step in the back-propagation, an outer product of the updated 1-D vectors **p**, **q**, and **t** is taken to concatenate them back into a 3-D convolution filter $\mathbf{W}''$, which we call the tensor composing step:

$$\begin{aligned} w''_{i,j,k} &= p'_i q'_j t'_k = (p_i - \alpha \tfrac{\partial L}{\partial p_i})(q_j - \alpha \tfrac{\partial L}{\partial q_j})(t_k - \alpha \tfrac{\partial L}{\partial t_k}) \\ &= p_i q_j t_k - \alpha (p_i q_j \tfrac{\partial L}{\partial t_k} + q_j t_k \tfrac{\partial L}{\partial p_i} + p_i t_k \tfrac{\partial L}{\partial q_j}) + \alpha^2 (p_i \tfrac{\partial L}{\partial q_j} \tfrac{\partial L}{\partial t_k} + q_j \tfrac{\partial L}{\partial p_i} \tfrac{\partial L}{\partial t_k} + t_k \tfrac{\partial L}{\partial p_i} \tfrac{\partial L}{\partial t_k}) - \alpha^3 \tfrac{\partial L}{\partial p_i} \tfrac{\partial L}{\partial q_j} \tfrac{\partial L}{\partial t_k} \\ &= w_{i,j,k} - \alpha \Delta_{i,j,k}, \ \forall_{i,j,k}, \end{aligned} \tag{22}$$

where

$$\Delta_{i,j,k} = p_i q_j \frac{\partial L}{\partial t_k} + q_j t_k \frac{\partial L}{\partial p_i} + p_i t_k \frac{\partial L}{\partial q_j} - \alpha(p_i \frac{\partial L}{\partial q_j}\frac{\partial L}{\partial t_k} + q_j \frac{\partial L}{\partial p_i}\frac{\partial L}{\partial t_k} + t_k \frac{\partial L}{\partial p_i}\frac{\partial L}{\partial t_k}) + \alpha^2 \frac{\partial L}{\partial p_i}\frac{\partial L}{\partial q_j}\frac{\partial L}{\partial t_k}. \quad (23)$$

As the outer product of 1-D vectors always results in a rank-1 filter, $\mathbf{W}''$ is a rank-1 filter as compared with $\mathbf{W}'$ which is not. Comparing (15) with (22), we get

$$w''_{i,j,k} = w'_{i,j,k} - \alpha(\Delta_{i,j,k} - \frac{\partial L}{\partial w_{i,j,k}}). \quad (24)$$

Therefore, we can say that $\Delta_{i,j,k} - \frac{\partial L}{\partial w_{i,j,k}}$ is the incremental update vector which projects $\mathbf{W}'$ back onto the rank-1 subspace. The use of rank-1 filters are not constrained to replace the filters in the standard CNN structure but can also replace the full-rank filters in ResNet or DenseNet-like architectures. In this case, the rank-1 filters can also reduce the parameters and accelerate the inference speed in ResNet or DenseNet architectures.

5. Application of the Proposed Training Method to the MobileNet

Here, we show that the proposed rank-1 network training method can also be applied to train the well-known MobileNet (version 1). However, the performance becomes better when the parameters are obtained by the proposed method, than when obtained by the original MobileNet type training method. The main idea of applying the proposed training method to the training of the MobileNet is that we can extend the separate 2-D filters to 3-D filters by the outer product of rank-1 2-D filters with rank-1 1-D vectors resulting in rank-1 3-D filters, train the rank-1 3-D filters with the ATCD training method, and then compress the rank-1 3-D filters back to full-rank 2-D filters. In the original version (version 1) of the MobileNet, the 2-D images are separately convolved with 2-D filters, and then are combined by 1×1 convolutions. The output of a single layer of the original version of the MobileNet can be written as

$$\mathbf{Y}^{(m)} = \sum_{j=1}^{N} \mathbf{a}_m[j](\mathbf{X}^{(j)} \circledast \mathbf{W}^{(j)}), m = 1, ..., q, \quad (25)$$

where $\mathbf{Y}^{(m)}$ is the m's output channel, $\mathbf{X}^{(j)}$ is the j's input channel, $\mathbf{W}^{(j)}$ is the j's filter that convolves with the j's input channel, $\circledast$ is the 2-D convolution operator, and $\mathbf{a}_m$ is the m's 1×1 convolution filter that produces the m's output channel. Meanwhile, the outputs $\hat{\mathbf{Y}}^{(i)}$, $i = 1, ..., K$ which are obtained by the convolutions of the 3-D rank-1 filters and the input channels as shown in Figure 3 become

$$\hat{\mathbf{Y}}^{(i)} = \sum_{j=1}^{N} \mathbf{t}_i[j](\mathbf{X}^{(j)} \circledast \hat{\mathbf{W}}^{(i)}), i = 1, ..., K. \quad (26)$$

It has to be noted that the index of $\hat{\mathbf{W}}^{(i)}$ in (26) is i (the index of output channels) compared to (25) where the index of $\mathbf{W}^{(j)}$ is j (the index of input channels).

Now, adding an extra layer which computes the linear combinations of the outputs $\hat{\mathbf{Y}}^{(i)}, i = 1, ..., K$ in (26) by 1×1 convolutions with the filters $\mathbf{k}_m, m = 1, ..., q$, we have

$$\mathbf{Y}^{(m)} = \sum_{i=1}^{K} \mathbf{k}_m[i]\hat{\mathbf{Y}}^{(i)}, m = 1, ..., q. \quad (27)$$

By putting (26) into (27) and rearranging the order of summation, we get,

$$\mathbf{Y}^{(m)} = \sum_{j=1}^{N} \left[\mathbf{X}^{(j)} \circledast \left(\sum_{i=1}^{K} \mathbf{k}_m[i]\mathbf{t}_i[j]\hat{\mathbf{W}}^{(i)} \right) \right]. \quad (28)$$

After the values of $\mathbf{k}_m[i]$ and $\mathbf{t}_i[j]$ are all fixed for all m, i, and j, i.e., after they have been trained, we can arbitrarily construct the vectors $\mathbf{a}_{m,\,m=1,\ldots,q}$ and $\mathbf{b}_{j,\,j=1,\ldots,N}$ so that the entries in the vectors are assigned as follows:

$$\mathbf{a}_m[j] = \mathbf{k}_m[i],\ \ \mathbf{b}_j[i] = \mathbf{t}_i[j]. \tag{29}$$

Then, we can rewrite (28) as

$$\mathbf{Y}^{(m)} = \sum_{j=1}^{N} \left[\mathbf{X}^{(j)} \circledast \left(\sum_{i=1}^{K} \mathbf{a}_m[j]\mathbf{b}_j[i]\hat{\mathbf{W}}^{(i)} \right) \right], \tag{30}$$

which can now be rewritten as

$$\mathbf{Y}^{(m)} = \sum_{j=1}^{N} \mathbf{a}_m[j] \left[\mathbf{X}^{(j)} \circledast \left(\sum_{i=1}^{K} \mathbf{b}_j[i]\hat{\mathbf{W}}^{(i)} \right) \right]. \tag{31}$$

By letting

$$\mathbf{W}^{(j)} = \sum_{i=1}^{K} \mathbf{b}_j[i]\hat{\mathbf{W}}^{(i)}, \tag{32}$$

the formula becomes the same as that for the Mobilenet described in (25). This means that after being training by the proposed method, we can implement the inference system also in the Mobilenet style. It has to be noticed that even though $\hat{\mathbf{W}}^{(i)}$ is a two dimensional rank-1 filter, since it is composed of the outer product of two 1-D vectors, the filter $\mathbf{W}^{(j)}$ can have a rank of K, as the summation of K independent rank-1 filters results in a rank-K filter. As shown in the experiments, the proposed method learns better parameters due to the over-parametrization produced by the outer product into a 3-D filter, and therefore, the MobileNet constructed by the proposed rank-1 training method has a higher accuracy than that of the original MobileNet which is trained with a smaller number of parameters. Therefore, the proposed training method can contribute to obtain MobileNets with higher classification accuracies.

6. Experiments

We compared the performance of the proposed ATCD training method with the conventional fixed structure training method for the rank-1 CNN and the MobileNet on various datasets. We also compared the validation and testing accuracies with a standard full-rank CNN. We used the same number of layers for all the models, where for the fixed structured Flattened CNN we regarded the combination of the lateral, vertical, and horizontal 1-D convolutional layers as a single layer. Furthermore, we used the same numbers of input and output channels in each layer for all the models, and also the same ReLU(Rectified Linear Unit), batch normalization, and dropout operations.

Tables 1–4 show the different structures of the models used for each dataset in the training stage. The outer product operation of the three 1-D filters $\mathbf{p}$, $\mathbf{q}$, and $\mathbf{t}$ into a 3-D rank-1 filter $\mathbf{w}$ is denoted as $\mathbf{w} \doteq \mathbf{p} \otimes \mathbf{q} \otimes \mathbf{t}$ in the tables. We did not elaborate on the structures to produce the optimal performances, but only tried to make them the same for fair comparison. Furthermore, we did not use any recent structures with extra components like skip-connections, element-wise or channel-wise concatenations, multi-scale filters, such as the ResNet or DenseNet, but intentionally used simple VGG-like structures with simple consecutive convolutional filters to see only the effect of the proposed training method. However, to verify the fact that the proposed training method can be applied also to the training of rank-1 filters inside a ResNet or DenseNet structure, we further performed an experiment on the CIFAR10 dataset with the ResNet structure where we replaced all the convolutional filters with the rank-1 filters and then applied our ATCD training method.

Table 1. Structure of convolutional neural network (CNN)1 for the MNIST dataset.

Standard CNN	Flattened CNN	Proposed CNN
Conv1: 64 filters, each filter constituted as:		
$1 \times 3 \times 3$ conv	$1 \times 1 \times 1\times$ conv, $1 \times 3 \times 1$ conv, $1 \times 1 \times 3$ conv	$\mathbf{w}_1 \doteq \mathbf{p}_1(1 \times 3 \times 1) \otimes \mathbf{q}_1(1 \times 1 \times 3) \otimes \mathbf{t}_1(1 \times 1 \times 1)$ $1 \times 3 \times 3$ conv
Conv2: 64 filters, each filter constituted as:		
$64 \times 3 \times 3$ conv	$64 \times 1 \times 1$ conv, $1 \times 3 \times 1$ conv, $1 \times 1 \times 3$ conv	$\mathbf{w}_2 \doteq \mathbf{p}_2(1 \times 3 \times 1) \otimes \mathbf{q}_2(1 \times 1 \times 3) \otimes \mathbf{t}_2(64 \times 1 \times 1)$ $64 \times 3 \times 3$ conv
Max Pool ($\frac{1}{2}$)		
Conv3: 144 filters, each filter constituted as:		
$64 \times 3 \times 3$ conv	$64 \times 1 \times 1$ conv, $1 \times 3 \times 1$ conv $1 \times 1 \times 3$ conv	$\mathbf{w}_3 \doteq \mathbf{p}_3(1 \times 3 \times 1) \otimes \mathbf{q}_3(1 \times 1 \times 3) \otimes \mathbf{t}_3(64 \times 1 \times 1)$ $64 \times 3 \times 3$ conv
Conv4: 144 filters, each filter constituted as:		
$144 \times 3 \times 3$ conv	$144 \times 1 \times 1$ conv, $1 \times 3 \times 1$ conv $1 \times 1 \times 3$ conv	$\mathbf{w}_4 \doteq \mathbf{p}_4(1 \times 3 \times 1) \otimes \mathbf{q}_4(1 \times 1 \times 3) \otimes \mathbf{t}_4(144 \times 1 \times 1)$ $144 \times 3 \times 3$ conv
Max Pool ($\frac{1}{2}$)		
Conv5: 144 filters, each filter constituted as:		
$144 \times 3 \times 3$ conv	$144 \times 1 \times 1$ conv , $1 \times 3 \times 1$ conv $1 \times 1 \times 3$ conv	$\mathbf{w}_5 \doteq \mathbf{p}_5(1 \times 3 \times 1) \otimes \mathbf{q}_5(1 \times 1 \times 3) \otimes \mathbf{t}_5(144 \times 1 \times 1)$ $144 \times 3 \times 3$ conv
Conv6: 256 filters, each filter constituted as:		
$144 \times 3 \times 3$ conv	$144 \times 1 \times 1$ conv, $1 \times 3 \times 1$ conv $1 \times 1 \times 3$ conv	$\mathbf{w}_6 \doteq \mathbf{p}_6(1 \times 3 \times 1) \otimes \mathbf{q}_6(1 \times 1 \times 3) \otimes \mathbf{t}_6(144 \times 1 \times 1)$ $144 \times 3 \times 3$ conv
Conv7: 256 filters, each filter constituted as:		
$256 \times 3 \times 3$ conv	$256 \times 1 \times 1$ conv, $1 \times 3 \times 1$ conv $1 \times 1 \times 3$ conv	$\mathbf{w}_7 \doteq \mathbf{p}_7(1 \times 3 \times 1) \otimes \mathbf{q}_7(1 \times 1 \times 3) \otimes \mathbf{t}_7(256 \times 1 \times 1)$ $256 \times 3 \times 3$ conv
FC 2048 + Batch Normalization + ReLU + Drop Out (P = 0.5)		
FC 1024 + Batch Normalization + ReLU + Drop Out (P = 0.5)		
FC 10 + ReLU + Drop Out (P = 0.5)		
Soft-Max		

The datasets that we used in the experiments were the MNIST, the CIFAR10, CIFAR100, and the 'Dog and Cat' datasets (https://www.kaggle.com/c/dogs-vs-cats). We used different structures for different datasets, which we denoted as CNN1, CNN2, CNN3, and CNN4 in the tables, corresponding to the MNIST, CIFAR10, CIFAR100, and 'Dog and Cat' datasets, respectively. The MNIST and the CIFAR-10 datasets both consisted of 60,000 images in 10 different classes, divided into 50,000 training images and 10,000 test images, where the images in the CIFAR-10 dataset were colour images of size 32 × 32, while those in the MNIST dataset were gray images of size 28 × 28. The CIFAR-100 data set consisted of 100 classes, each with 500 training and 100 test color images of size 32 × 32. The 'Dog and Cat' dataset contained 25,000 color images of dogs and cats of size 224 × 224, which we divided into 24,900 training and 100 test images for the validation test along the training. At the end of each training session, we tested the final testing accuracy of the 'Dog and Cat' dataset by taking the average of the 100 test images. Then, the final testing accuracy was obtained by taking the mean of 10 of such sessions. For the experiments on the MNIST, the CIFAR10, and the CIFAR100 datasets, we trained on 50,000 images, and then tested on 100 batches each consisting of 100 random test images, and calculated the overall average accuracy both for the validation along the training and for

the final testing. We plotted for every training epoch the validation accuracy values in Figures 5–9. The number of epochs in the figures were determined to be greater than the epochs for which the validation accuracies with the proposed model sufficiently converged, and which resulted in graphs from which it became possible to visually compare the results of the different methods. Figures 5–9 show the validation accuracies for every epoch along the training process for a single training for each model. Even though it is customary to learn a model only once in deep learning, we performed multiple training sessions for each dataset, and obtained different models at the end of each training. Then, we calculated the means and the standard deviations of the different final accuracy values of the testing datasets for each trained model, and recorded them in Table 5. For the experiments on MNIST, CIFAR10, and CIFAR100 datasets, 20 training sessions were performed, and for the 'Dog and Cat' dataset, which took a long time to train, 10 training sessions were performed to obtain the values in Table 5. The slight differences between the testing accuracies of the different models are due to the different initialization of convolutional filters and the randomness of the stochastic gradient descent-based backpropagation.

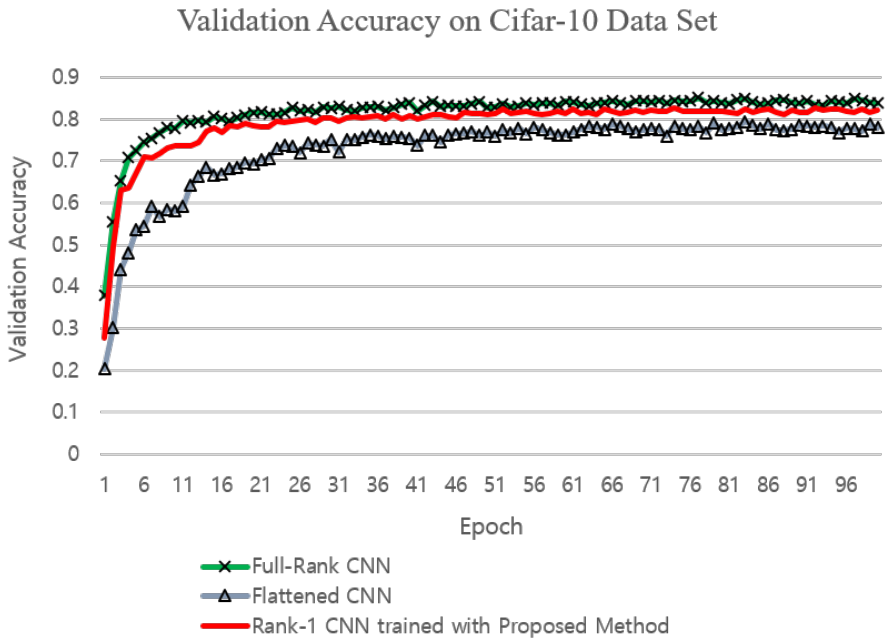

Figure 5. Comparison of validation accuracies on the CIFAR10 dataset.

The rank-1 CNN trained with the proposed training method achieved slightly larger validation and testing accuracies on the MNIST dataset than the standard full-rank CNN and the Flattened CNN trained with the conventional fixed-structure training method (Figure 10 and Table 5). This is maybe due to the fact that the MNIST dataset was in its nature a low-ranked one, for which the proposed method could find the best approximation since the proposed method constrains the filters to a low rank sub-space. With the CIFAR10 and the CIFAR100 dataset, the accuracy was slightly less than that of the standard CNN which is maybe due to the fact that the images in the CIFAR10 and the CIFAR100 datasets were of higher ranks than those in the MNIST dataset. However, the validation accuracy of the rank-1 CNN trained with the proposed method was higher than that of the Flattened CNN trained with the conventional fixed structure training method on the CIFAR10 dataset which shows the fact that the better gradient flow in the proposed training method achieves a better solution. With the CIFAR100 dataset, the Flattened CNN could not be trained by conventional fixed structure training methods due to the deep structure of the CNN4 structure. The 'Dog and Cat' dataset was used in the experiments to verify the validness of the proposed training method on real-sized images and on a deep structure. In this case, again the Flattened network could not be trained with the conventional fixed

structure training method. This is maybe due to the limitation to produce a gradient flow in deep structures with the direct fixed 1-D structure of the Flattened CNN. The standard CNN trained with conventional training method and the proposed rank-1 CNN trained with the proposed training method achieved similar validation accuracies as can be seen in Figure 6. The validation accuracies for the CIFAR100 dataset are shown in Figure 7. Again, it can be seen that the rank-1 CNN trained with the proposed ATCD method achieved similar validation and testing accuracies to the standard CNN.

Table 2. Structure of CNN2 for CIFAR-10 dataset.

Standard CNN	**Flattened CNN**	**Proposed CNN**
Conv1: 64 filters, each filter constituted as:		
$3 \times 3 \times 3$ conv	$3 \times 1 \times 1$ conv, $1 \times 3 \times 1$ conv $1 \times 1 \times 3$ conv	$\mathbf{w}_1 \doteq \mathbf{p}_1(1 \times 3 \times 1) \otimes \mathbf{q}_1(1 \times 1 \times 3) \otimes \mathbf{t}_1(3 \times 1 \times 1)$ $3 \times 3 \times 3$ conv
ReLU + Batch Normalization		
Conv2: 64 filters, each filter constituted as:		
$64 \times 3 \times 3$ conv	$64 \times 1 \times 1$ conv, $1 \times 3 \times 1$ conv $1 \times 1 \times 3$ conv	$\mathbf{w}_2 \doteq \mathbf{p}_2(1 \times 3 \times 1) \otimes \mathbf{q}_2(1 \times 1 \times 3) \otimes \mathbf{t}_2(64 \times 1 \times 1)$ $64 \times 3 \times 3$ conv
ReLU + Max Pool ($\frac{1}{2}$) + Drop Out (P = 0.5)		
Conv3: 144 filters, each filter constituted as:		
$64 \times 3 \times 3$ conv	$64 \times 1 \times 1$ conv, $1 \times 3 \times 1$ conv $1 \times 1 \times 3$ conv	$\mathbf{w}_3 \doteq \mathbf{p}_3(1 \times 3 \times 1) \otimes \mathbf{q}_3(1 \times 1 \times 3) \otimes \mathbf{t}_3(64 \times 1 \times 1)$ $64 \times 3 \times 3$ conv
ReLU + Batch Normalization		
Conv4: 144 filters, each filter constituted as:		
$144 \times 3 \times 3$ conv	$144 \times 1 \times 1$ conv, $1 \times 3 \times 1$ conv $1 \times 1 \times 3$ conv	$\mathbf{w}_4 \doteq \mathbf{p}_4(1 \times 3 \times 1) \otimes \mathbf{q}_4(1 \times 1 \times 3) \otimes \mathbf{t}_4(144 \times 1 \times 1)$ $144 \times 3 \times 3$ conv
ReLU + Max Pool ($\frac{1}{2}$) +Drop Out (P = 0.5)		
Conv5: 256 filters, each filter constituted as:		
$144 \times 3 \times 3$ conv	$144 \times 1 \times 1$ conv, $1 \times 3 \times 1$ conv $1 \times 1 \times 3$ conv	$\mathbf{w}_5 \doteq \mathbf{p}_5(1 \times 3 \times 1) \otimes \mathbf{q}_5(1 \times 1 \times 3) \otimes \mathbf{t}_5(144 \times 1 \times 1)$ $144 \times 3 \times 3$ conv
ReLU + Batch Normalization		
Conv6: 256 filters, each filter constituted as:		
$256 \times 3 \times 3$ conv	$256 \times 1 \times 1$ conv, $1 \times 3 \times 1$ conv $1 \times 1 \times 3$ conv	$\mathbf{w}_6 \doteq \mathbf{p}_6(1 \times 3 \times 1) \otimes \mathbf{q}_6(1 \times 1 \times 3) \otimes \mathbf{t}_6(256 \times 1 \times 1)$ $256 \times 3 \times 3$ conv
ReLU + Max Pool ($\frac{1}{2}$) + Drop Out (P = 0.5)		
FC 1024 + Batch Normalization + ReLU + Drop Out (P = 0.5)		
FC 512 + Batch Normalization + ReLU + Drop Out (P = 0.5)		
FC 10		
Soft-Max		

Table 3. Structure of CNN3 for 'Dog and Cat' dataset.

Standard CNN	**Proposed CNN**
Conv1: 64 filters, each filter constituted as:	
$3 \times 3 \times 3$ conv	$\mathbf{w}_1 \doteq \mathbf{p}_1(1 \times 3 \times 1) \otimes \mathbf{q}_1(1 \times 1 \times 3) \otimes \mathbf{t}_1(3 \times 1 \times 1)$ $3 \times 3 \times 3$ conv
Conv2: 64 filters, each filter constituted as:	
$64 \times 3 \times 3$ conv	$\mathbf{w}_2 \doteq \mathbf{p}_2(1 \times 3 \times 1) \otimes \mathbf{q}_2(1 \times 1 \times 3) \otimes \mathbf{t}_2(64 \times 1 \times 1)$ $64 \times 3 \times 3$ conv
Batch Normalization + ReLU + Max Pool $(\frac{1}{2})$	
Conv3: 144 filters, each filter constituted as:	
$64 \times 3 \times 3$ conv	$\mathbf{w}_3 \doteq \mathbf{p}_3(1 \times 3 \times 1) \otimes \mathbf{q}_3(1 \times 3 \times 1) \otimes \mathbf{t}_3(64 \times 1 \times 1)$ $64 \times 3 \times 3$ conv
ReLU	
Conv4: 144 filters, each filter constituted as:	
$144 \times 3 \times 3$ conv	$\mathbf{w}_4 \doteq \mathbf{p}_4(1 \times 3 \times 1) \otimes \mathbf{q}_4(1 \times 1 \times 3) \otimes \mathbf{t}_4(144 \times 1 \times 1)$ $144 \times 3 \times 3$ conv
Batch Normalization + ReLU + Max Pool $(\frac{1}{2})$	
Conv5: 256 filters, each filter constituted as:	
$144 \times 3 \times 3$ conv	$\mathbf{w}_5 \doteq \mathbf{p}_5(1 \times 3 \times 1) \otimes \mathbf{q}_5(1 \times 1 \times 3) \otimes \mathbf{t}_5(144 \times 1 \times 1)$ $144 \times 3 \times 3$ conv
ReLU	
Conv6: 256 filters, each filter constituted as:	
$256 \times 3 \times 3$ conv	$\mathbf{w}_6 \doteq \mathbf{p}_6(1 \times 3 \times 1) \otimes \mathbf{q}_6(1 \times 1 \times 3) \otimes \mathbf{t}_6(256 \times 1 \times 1)$ $256 \times 3 \times 3$ conv
Batch Normalization + ReLU + Max Pool $(\frac{1}{2})$	
Conv7: 256 filters, each filter constituted as:	
$256 \times 3 \times 3$ conv	$\mathbf{w}_7 \doteq \mathbf{p}_7(1 \times 3 \times 1) \otimes \mathbf{q}_7(1 \times 1 \times 3) \otimes \mathbf{t}_7(256 \times 1 \times 1)$ $256 \times 3 \times 3$ conv
ReLU	
Conv8: 484 filters, each filter constituted as:	
$256 \times 3 \times 3$ conv	$\mathbf{w}_8 \doteq \mathbf{p}_8(1 \times 3 \times 1) \otimes \mathbf{q}_8(1 \times 1 \times 3) \otimes \mathbf{t}_8(256 \times 1 \times 1)$ $256 \times 3 \times 3$ conv
ReLU	
Conv9: 484 filters, each filter constituted as:	
$484 \times 3 \times 3$ conv	$\mathbf{w}_9 \doteq \mathbf{p}_9(1 \times 3 \times 1) \otimes \mathbf{q}_9(1 \times 1 \times 3) \otimes \mathbf{t}_9(484 \times 1 \times 1)$ $484 \times 3 \times 3$ conv
Batch Normalization + ReLU + Max Pool $(\frac{1}{2})$	
Conv10: 484 filters, each filter constituted as:	
$484 \times 3 \times 3$ conv	$\mathbf{w}_{10} \doteq \mathbf{p}_{10}(1 \times 3 \times 1) \otimes \mathbf{q}_{10}(1 \times 1 \times 3) \otimes \mathbf{t}_{10}(484 \times 1 \times 1)$ $484 \times 3 \times 3$ conv
ReLU	
Conv11: 484 filters, each filter constituted as:	
$484 \times 3 \times 3$ conv	$\mathbf{w}_{11} \doteq \mathbf{p}_{11}(1 \times 3 \times 1) \otimes \mathbf{q}_{11}(1 \times 1 \times 3) \otimes \mathbf{t}_{11}(484 \times 1 \times 1)$ $484 \times 3 \times 3$ conv
Batch Normalization + ReLU + Max Pool $(\frac{1}{2})$	
FC 1024 + Batch Normalization + ReLU	
FC 512 + Batch Normalization + ReLU	
FC 2	
Soft-Max	

Table 4. Structure of CNN4 for CIFAR-100 dataset.

Standard CNN	**Proposed CNN**
Conv1: 64 filters, each filter constituted as:	
$3 \times 3 \times 3$ conv	$\mathbf{w}_1 \doteq \mathbf{p}_1(1 \times 3 \times 1) \otimes \mathbf{q}_1(1 \times 1 \times 3) \otimes \mathbf{t}_1(3 \times 1 \times 1)$ $3 \times 3 \times 3$ conv
Batch Normalization + ReLU + Drop Out (P = 0.5)	
Conv2: 64 filters, each filter constituted as:	
$64 \times 3 \times 3$ conv	$\mathbf{w}_2 \doteq \mathbf{p}_2(1 \times 3 \times 1) \otimes \mathbf{q}_2(1 \times 1 \times 3) \otimes \mathbf{t}_2(64 \times 1 \times 1)$ $64 \times 3 \times 3$ conv
Batch Normalization + ReLU + Max Pool $(\frac{1}{2})$	
Conv3: 144 filters, each filter constituted as:	
$64 \times 3 \times 3$ conv	$\mathbf{w}_3 \doteq \mathbf{p}_3(1 \times 3 \times 1) \otimes \mathbf{q}_3(1 \times 3 \times 1) \otimes \mathbf{t}_3(64 \times 1 \times 1)$ $64 \times 3 \times 3$ conv
Batch Normalization + ReLU + Drop Out (P = 0.4)	
Conv4: 144 filters, each filter constituted as:	
$144 \times 3 \times 3$ conv	$\mathbf{w}_4 \doteq \mathbf{p}_4(1 \times 3 \times 1) \otimes \mathbf{q}_4(1 \times 1 \times 3) \otimes \mathbf{t}_4(144 \times 1 \times 1)$ $144 \times 3 \times 3$ conv
Batch Normalization + ReLU + Max Pool $(\frac{1}{2})$	
Conv5: 256 filters, each filter constituted as:	
$144 \times 3 \times 3$ conv	$\mathbf{w}_5 \doteq \mathbf{p}_5(1 \times 3 \times 1) \otimes \mathbf{q}_5(1 \times 1 \times 3) \otimes \mathbf{t}_5(144 \times 1 \times 1)$ $144 \times 3 \times 3$ conv
Batch Normalization + ReLU + Drop Out (P = 0.4)	
Conv6: 256 filters, each filter constituted as:	
$256 \times 3 \times 3$ conv	$\mathbf{w}_6 \doteq \mathbf{p}_6(1 \times 3 \times 1) \otimes \mathbf{q}_6(1 \times 1 \times 3) \otimes \mathbf{t}_6(256 \times 1 \times 1)$ $256 \times 3 \times 3$ conv
Batch Normalization + ReLU + Drop Out (P = 0.4)	
Conv7: 256 filters, each filter constituted as:	
$256 \times 3 \times 3$ conv	$\mathbf{w}_7 \doteq \mathbf{p}_7(1 \times 3 \times 1) \otimes \mathbf{q}_7(1 \times 1 \times 3) \otimes \mathbf{t}_7(256 \times 1 \times 1)$ $256 \times 3 \times 3$ conv
Batch Normalization + ReLU + Max Pool $(\frac{1}{2})$	
Conv8: 484 filters, each filter constituted as:	
$256 \times 3 \times 3$ conv	$\mathbf{w}_8 \doteq \mathbf{p}_8(1 \times 3 \times 1) \otimes \mathbf{q}_8(1 \times 1 \times 3) \otimes \mathbf{t}_8(256 \times 1 \times 1)$ $256 \times 3 \times 3$ conv
Batch Normalization + ReLU+ Drop Out (P = 0.4)	
Conv9: 484 filters, each filter constituted as:	
$484 \times 3 \times 3$ conv	$\mathbf{w}_9 \doteq \mathbf{p}_9(1 \times 3 \times 1) \otimes \mathbf{q}_9(1 \times 1 \times 3) \otimes \mathbf{t}_9(484 \times 1 \times 1)$ $484 \times 3 \times 3$ conv
Batch Normalization + ReLU + Drop Out (P = 0.4)	
Conv10: 484 filters, each filter constituted as:	
$484 \times 3 \times 3$ conv	$\mathbf{w}_{10} \doteq \mathbf{p}_{10}(1 \times 3 \times 1) \otimes \mathbf{q}_{10}(1 \times 1 \times 3) \otimes \mathbf{t}_{10}(484 \times 1 \times 1)$ $484 \times 3 \times 3$ conv
Batch Normalization + ReLU+Max Pool $(\frac{1}{2})$	
$484 \times 3 \times 3$ conv	$\mathbf{w}_{10} \doteq \mathbf{p}_{10}(1 \times 3 \times 1) \otimes \mathbf{q}_{10}(1 \times 1 \times 3) \otimes \mathbf{t}_{10}(484 \times 1 \times 1)$ $484 \times 3 \times 3$ conv
Batch Normalization + ReLU+ Drop Out (P = 0.4)	
FC 2048 Batch Normalization + ReLU + Drop Out (P = 0.5)	
FC 1024 + Batch Normalization + ReLU + Drop Out (P = 0.5)	
FC 512 + Batch Normalization + ReLU + Drop Out (P = 0.5)	
FC 100	
Soft-Max	

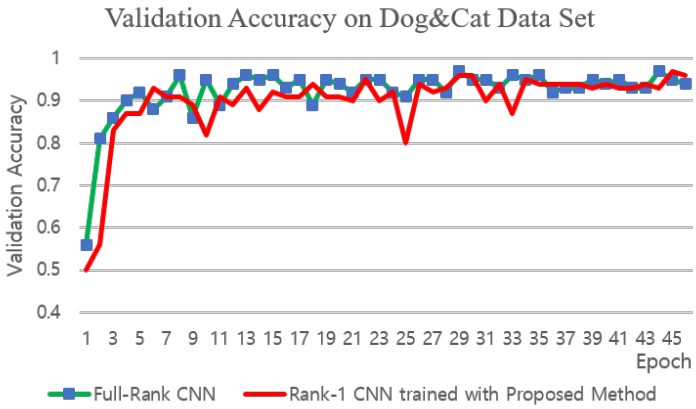

Figure 6. Comparison of validation accuracies on the 'Dog and Cat' dataset.

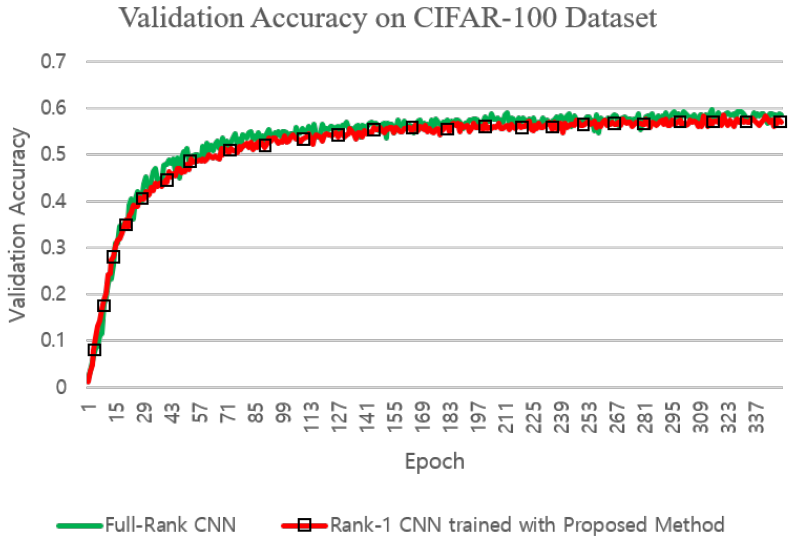

Figure 7. Comparison of validation accuracies on the CIFAR100 dataset.

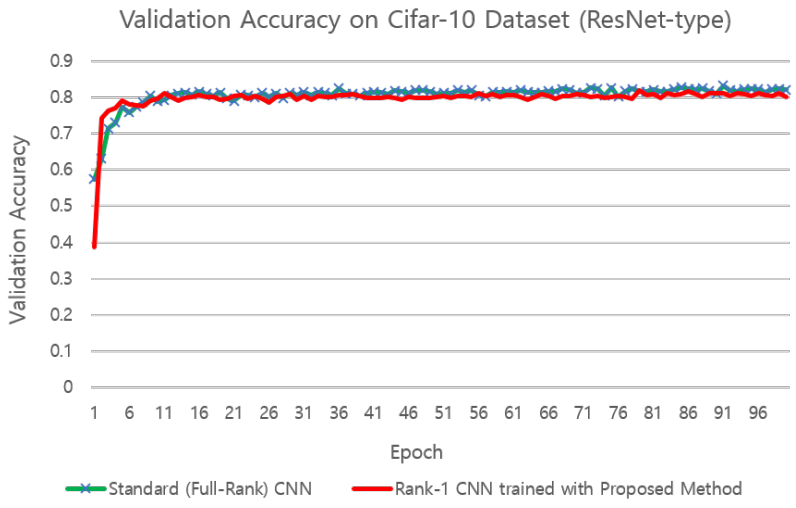

Figure 8. Comparison of validation accuracies on the CIFAR10 dataset with the ResNet structure.

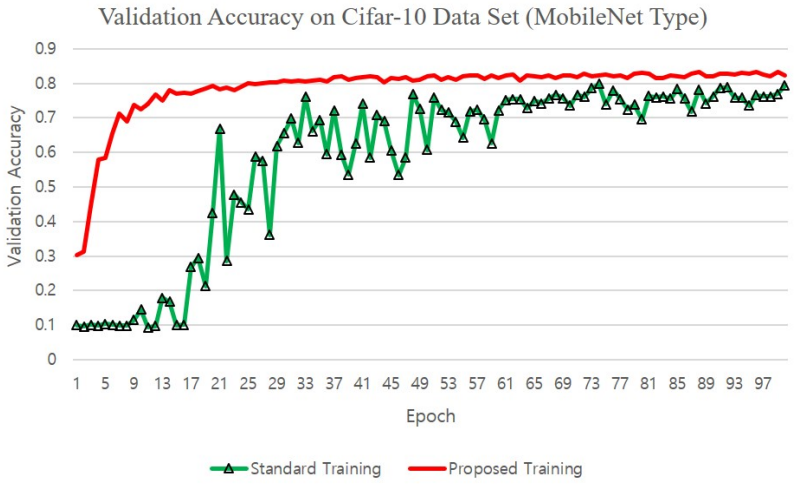

Figure 9. Comparison of validation accuracies on the CIFAR10 dataset with the proposed and the standard training methods for the MobileNet.

Table 5. Comparison of accuracy and inference time between the different training methods with different architectures. Mean and std. stands for average and standard deviation. The values in the inference times are the mean and the standard deviation values, respectively, where the unit is second for the average value.

Architecture	Method	Accuracy Mean	Accuracy Std.	CPU Inference Time		GPU Inference Time	
				Mean (sec)	Std.	Mean (sec)	Std.
CNN1 Type	Standard	0.9888	8.01×10^{-4}	0.01834	0.43×10^{-3}	0.00242	1.38×10^{-5}
	Flattened	0.9850	9.16×10^{-4}	0.00931	0.25×10^{-3}	0.00132	0.76×10^{-5}
	Proposed	0.9911	5.08×10^{-4}	0.00931	0.25×10^{-3}	0.00132	0.76×10^{-5}
CNN2 Type	Standard	0.8422	4.19×10^{-3}	0.01337	0.22×10^{-3}	0.00178	1.01×10^{-5}
	Flattened	0.7777	6.58×10^{-3}	0.00732	0.18×10^{-3}	0.00082	0.54×10^{-5}
	Proposed	0.8213	3.40×10^{-3}	0.00732	0.18×10^{-3}	0.00082	0.54×10^{-5}
CNN3 Type	Standard	0.9450	1.32×10^{-2}	0.04490	1.22×10^{-3}	0.00788	2.46×10^{-5}
	Proposed	0.9413	1.55×10^{-2}	0.02452	1.07×10^{-3}	0.00374	2.13×10^{-5}
CNN4 Type	Standard	0.5851	3.38×10^{-3}	0.04383	1.21×10^{-3}	0.00748	2.51×10^{-5}
	Proposed	0.5750	5.50×10^{-3}	0.02371	1.15×10^{-3}	0.00363	2.11×10^{-5}

The number of operations reduced according to (12). So, for example, for the first layer of the structure CNN3, the number of operations for the standard CNN and the rank-1 (type-1) became $I_x \times I_y \times C_{out} \times f_x \times f_y = 224 \times 224 \times 64 \times 3 \times 3 = 28{,}901{,}376$ and $I_x \times I_y \times (C_{out} + f_x + f_y) = 224 \times 224 \times (64 + 3 + 3) = 3{,}512{,}320$, respectively. Therefore, the computation operations in the first convolutional layer in the CNN3 model was about eight times more with the standard CNN. Table 6 summarizes the number of parameters for the different models and structures. We also performed experiments on the inference times for the different models on CPU and GPU environments, and recorded the values in the fifth and sixth columns in Table 5. We used Tensorflow version 0.12.1 and ran it on Window10 OS, with NVIDIA 1080Ti GPU, Intel i9 CPU and 16 GB RAM memory. It should be noted that the Flattened model and the proposed model had the same inference speed as the structures in the testing time werere the same. Compared to the reduction ratio of the number of parameters, the increase in the inference speed was not that high, which is mainly due to the fact that the Tensorflow framework was not optimized for factorized

neural networks and that the loading of the image took a long time in the inference stage. We believe that in the future, the speed of factorized filtering will increase with a framework optimized for them.

Figure 8 compares the validation accuracies of the networks with ResNet structures composed of standard 3-D filters and rank-1 3-D filters, respectively, where the rank-1 3-D filters were trained with the proposed training method. We used the 56-layer ResNet structure for CIFAR10 as proposed in [50], and replaced all the standard 3-D filters by rank-1 3-D filters in the ResNet in the experiments with the proposed training method. As can be seen, the rank-1 ResNet trained with the proposed method achieved similar validation accuracy to the normal ResNet, which shows that the proposed training method can be applied to diverse network structures containing low-rank filters.

Table 6. Comparison of numbers of parameters in convolutional filters.

Standard CNN	CNN1	CNN2	CNN3	CNN4
Standard CNN	1,415,232	825,408	9,258,480	9,222,768
Proposed	157,752	137,072	1,029,904	964,236
Reduction Ratio	11.1%	16.6%	11.1%	10.4%

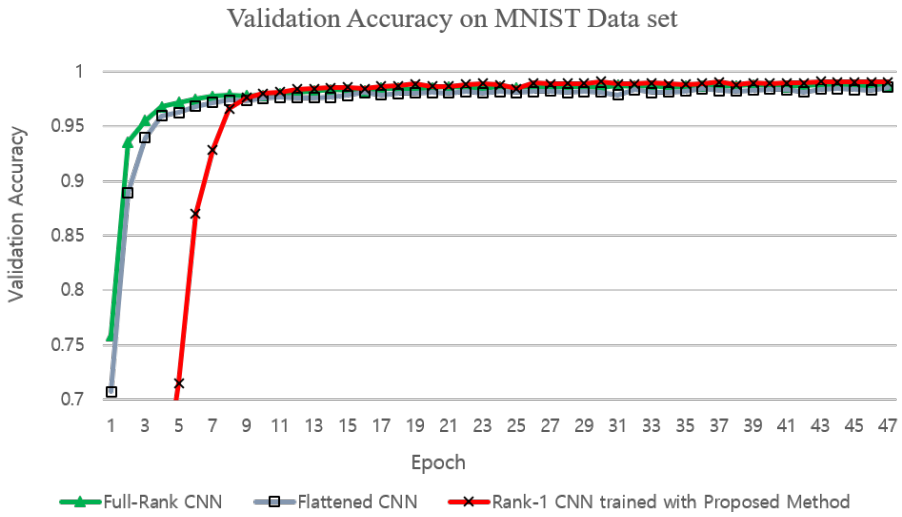

Figure 10. Comparison of validation accuracies on the MNIST dataset.

Finally, Figure 9 compares the validation accuracies of the MobileNet when trained with the normal method and with the proposed training method. We used the CNN4 structure and replaced all the 3-D convolution operations with the depth-wise separable 2-D convolutions + 1×1 pointwise convolutions as suggested in the MobileNet structure. It can be seen that the proposed training method could accelerate the training process and achieved a better validation accuracy than the standard training method, which is due to the intentional over-parametrization obtained by the outer product with the proposed method. Figure 11 shows the means and the positive standard deviations of the validation accuracies for different training sessions. It can be seen that the standard deviations at the early epochs of the training with the standard training method were large, while with the proposed training method the variation was not so large. However, after a long time of training, the final validation accuracies all converged to a similar value for each training session as shown by the standard deviation that decreased at the end of the training.

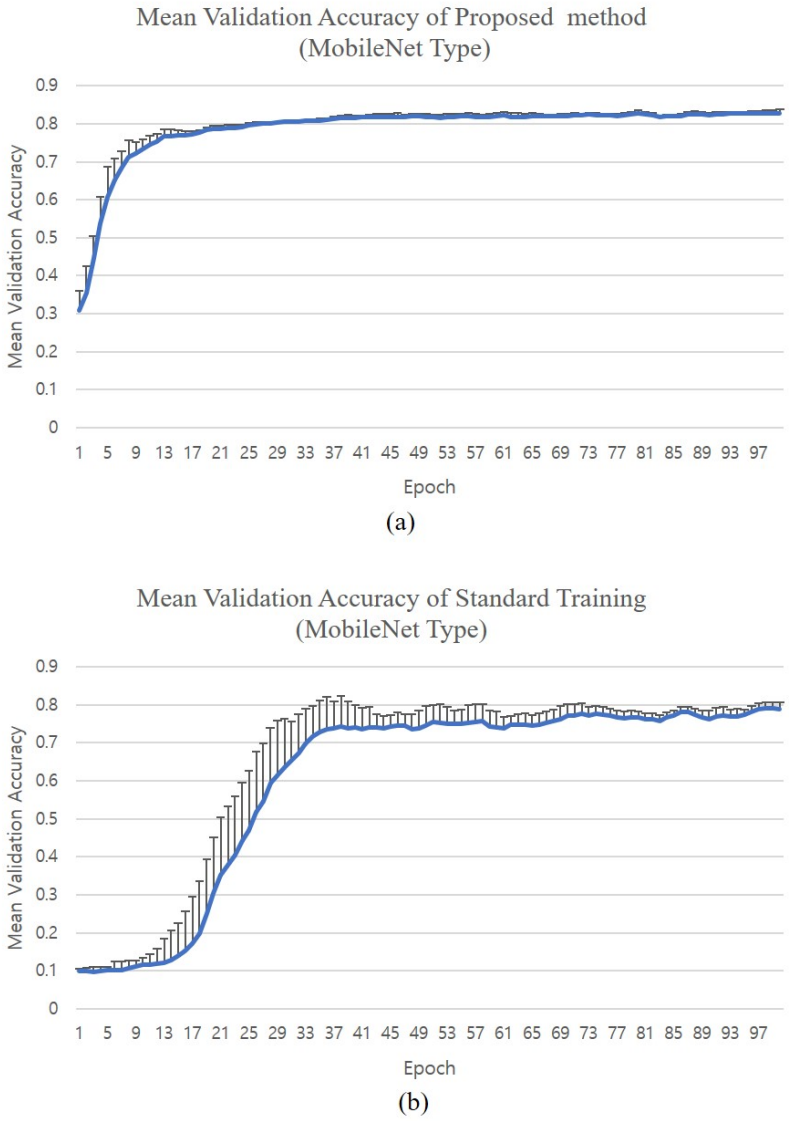

Figure 11. Comparison of mean validation accuracies on the CIFAR10 dataset with the proposed and the standard training methods for the MobileNet. (**a**) Mean validation accuracies of the proposed training method (vertical bars show the positive standard deviation); (**b**) mean validation accuracies of the standard training method (vertical bars show the positive standard deviation).

7. Conclusions

We proposed a training method which alternatively composes and decomposes the filters in the training stage for better training of low rank filters. As an exemplary case, we showed that a rank-1 CNN can be trained with the proposed method, which cannot be trained with conventional training methods. We used 3-D rank-1 filters in convolutional neural networks in the training phase, so that the redundancy in the filters are reduced by the rank deficient property of the rank-1 filters. The proposed training method updates the 3-D filter parameters and projects them back onto the rank-1 subspace at each epoch to find good parameter values for the 1-D vectors which constitute the 3-D filters. At test time, the trained 1-D vectors can be used directly as 1-D filters which filter the image by 1-D convolution instead of the 3-D convolution for fast inference. We showed in the experiments that the accuracy performance of the rank-1 CNN is almost the same as the standard CNN while reducing the number of parameters up to about 11%, and the number of operations up to about 12% in the convolution filters compared with the standard CNN. We also showed that the proposed method can also be used in the ResNet structure and

showed that the proposed training method can be utilized for a better training of the MobileNet. This suggests the possibility that the proposed rank-1 training method can also be used with diverse structures such as the ResNet or other small-sized networks, to obtain a more efficient network structure.

In this paper, we applied the proposed training method to well-known deep learning models that can be factorized. Whether the proposed training method can be applied to other complicated models still leaves much room. Combining the proposed method with the existing training method and applying it to other complex deep learning models can be an additional topic of this study. Moreover, the experimental results showed that the inference time was not reduced as much as the decrease in the number of parameters, which is due to the fact that existing deep learning frameworks are not optimized for factorized models. Therefore, in order to accelerate the inference speed, further studies on hardware designs that can effectively adopt factorized models should be carried out in the future.

Author Contributions: conceptualization, S.L., H.K. and J.Y.; methodology, S.L., B.J.; software, S.L., H.K.; validation, B.J., J.Y.; formal analysis, H.K., J.Y.; investigation, J.Y.; resources, S.L., B.J.; writing—original draft preparation, S.L., H.K.; writing—review and editing, S.L., B.S. J.Y.; visualization, S.L.; supervision, J.Y.; project administration, J.Y. All authors have read and agreed to the published version of the manuscript.

Funding: The work of J.Y. was supported in part by the National Research Foundation of Korea under Grant NRF-2015R1A5A1009350 and the work of S.L. was supported by the Basic Science Research Program through the National Research Foundation of Korea under Grant NRF-2019R1I1A3A01060150.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data sharing not applicable.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Badrinarayanan, V.; Kendall, A.; Cipolla, R. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *39*, 2481–2495. [CrossRef] [PubMed]
2. Shelhamer, E.; Long, J.; Darrell, T. Fully Convolutional Networks for Semantic Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *39*, 640–651. [CrossRef] [PubMed]
3. Fontanini, T.; Iotti, E.; Donati L.; Prati, A. MetalGAN: Multi-domain label-less image synthesis using cGANs and meta-learning. *Neural Netw.* **2020**, *131*, 185–200. [CrossRef] [PubMed]
4. Paier, W.; Hilsmann, A.; Eisert, P. Interactive facial animation with deep neural networks. *IET Comput. Vis.* **2020**, *14*, 359–369. [CrossRef]
5. Santos, F.; Zor, C.; Kittler, J.; Ponti, M.A. Learning image features with fewer labels using a semi-supervised deep convolutional network. *Neural Netw.* **2020**, *132*, 131–143. [CrossRef] [PubMed]
6. Yang, X.; Zhou, P.; Wang, M. Person Reidentification via Structural Deep Metric Learning. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *30*, 2987–2998. [CrossRef] [PubMed]
7. Sultan, W.; Anjum, N.; Stansfield, M.; Ramzan, N. Hybrid Local and Global Deep-Learning Architecture for Salient-Object Detection. *Appl. Sci.* **2020**, *10*, 1–15. [CrossRef]
8. Fuentes, L.; Farasin, A.; Zaffaroni, M.; Skinnemoen, H.; Garza, P. Deep Learning Models for Road Passability Detection during Flood Events Using Social Media Data. *Appl. Sci.* **2020**, *10*, 1–22.
9. Livni, R.; Shalev-Shwartz, S.; Shamir, O. On the Computational Efficiency of Training Neural Networks. In Proceedings of the Advances in Neural Information Processing Systems(NIPS), Montreal, QC, Canada, 8–13 December 2014; pp. 855–863.
10. Zhang, C.; Bengio, S.; Hardt, M.; Recht, B.; Vinyals, O. Understanding deep learning requires rethinking generalization. In Proceedings of the International Conference on Learning Representations (ICLR), Toulon, France, 24–26 April 2017; pp. 1–15.
11. Han, S.; Pool, J.; Tran, J.; Dally, W. Learning both weights and connections for efficient neural network. In Proceedings of the Advances in Neural Information Processing Systems (NIPS) Conference, Montreal, QC, Canada, 7–12 December 2015; pp. 1135–1143.

12. Yu, R.; Li, A.; Chen, C.F.; Lai, J.H.; Morariu, V.; Han, X.; Gao, M.; Lin, C.Y.; Davis, L.S. Nisp: Pruning networks using neuron importance score propagation. In Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–23 June 2018; pp. 9194–9203.
13. Denton, E.L.; Zaremba, W.; Bruna, J.; LeCun, Y.; Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. In Proceedings of the Advances in Neural Information Processing Systems (NIPS) Conference, Montreal, QC, Canada, 8–13 December 2014; pp. 1269–1277.
14. Jaderberg, M.; Vedaldi, A.; Zisserman, A. Speeding up convolutional neural networks with low rank expansions. In Proceedings of the British Machine Vision Conference (BMVC), Nottingham, UK, 1–5 September 2014; pp. 1–13.
15. Zhang, X.; Zou, J.; He, K.; Sun, J. Accelerating very deep convolutional networks for classification and detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **2015**, *38*, 1943–1955. [CrossRef]
16. Lin, S.; Ji, R.; Chen, C.; Tao, D.; Luo, J. Holistic CNN compression via low-rank decomposition with knowledge transfer. *IEEE Trans. Pattern Anal. Mach. Intell.* **2018**, *41*, 2889–2905. [CrossRef]
17. Wu, C.; Cui, Y.; Ji, C.; Kuo, T.W.; Xue, C.J. Pruning Deep Reinforcement Learning for Dual User Experience and Storage Lifetime Improvement on Mobile Devices. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2020**, *39*, 3993–4005. [CrossRef]
18. Chen, W.; Wilson, J.T.; Tyree, S.; Weinberger, K.Q.; Chen, Y. Compressing neural networks with the hashing trick. In Proceedings of the 32nd International Conference on International Conference on Machine Learning (ICML), Lille, France, 6–11 July 2015; pp. 2285–2294.
19. Gong, Y.; Liu, L.; Yang, M.; Bourdev, L. Compressing deep convolutional networks using vector quantization. In Proceedings of the International Conference on Learning Representations (ICLR), San Diego, CA, USA, 7–9 May 2015; pp. 1–10.
20. Han, S.; Mao, H.; Dally, W.J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In Proceedings of the International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, 2–4 May 2016; pp. 1–14.
21. Rastegari, M.; Ordonez, V.; Redmon, J.; Farhadi, A. XNOR-Net: ImageNet classification using binary convolutional neural networks. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 8–16 October 2016; pp. 525–542.
22. Wang, Y.; Xu, C.; You, S.; Tao, D.; Xu, C. CNNpack: Packing convolutional neural networks in the frequency domain. In Proceedings of the Advances in Neural Information Processing Systems (NIPS), Barcelona, Spain, 5–10 December 2016; pp. 253–261.
23. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv* **2017**, arXiv:1704.04861.
24. Chollet, F. Xception: Deep learning with depthwise separable convolution. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 1251–1258.
25. Iandola, F. N.; Moskewicz, M.W.; Ashraf, K.; Han, S.; Dally, W.J.; Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and 1mb model size. *arXiv* **2016**, arXiv:1602.07360.
26. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Liang-Chieh Chen, L.C. MobileNet V2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–23 June 2018; pp. 4510–4520.
27. Tan M.; Le, Q.V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In Proceedings of the 2019 International Conference on Machine Learning (ICML), Long Beach, CA, USA, 9–15 June 2019; pp. 1–10.
28. Huang, G.; Liu, S.; Maaten, L.; Weinberger, K.Q. Condensenet: An efficient densenet using learned group convolutions. In Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–23 June 2018; pp. 1–11.
29. Ma, N.; Zhang, X.; Zheng, H.T.; Sun, J. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In Proceedings of the 2018 European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 116–131.
30. Jin, J.; Dundar, A.; Culurciello, E. Flattened convolutional neural networks for feedforward acceleration. In Proceedings of the International Conference on Learning Representations(ICLR), San Diego, CA, USA, 7–9 May 2015; pp. 1–11.
31. Ioannou, Y.; Robertson, D.; Shotton, J.; Cipolla, R.; Criminisi, A. Training CNNs with Low-Rank Filters for Efficient Image Classification. In Proceedings of the 2016 International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, 2–4 May 2016; pp. 1–17.
32. Cao, S.; Wang, X.; Kitani K.M. Learnable embedding space for efficient neural architecture compression. In Proceedings of the 2019 International Conference on Learning Representations (ICLR), New Orleans, LA, USA, 6–9 May 2019; pp. 1–17.
33. Ashok, A.; Rhinehart, N.; Beainy, F.; Kitani K.M. N2n learning: Network to network compression via policy gradient reinforcement learning. In Proceedings of the 2018 International Conference on Learning Representations (ICLR), Vancouver, BC, Canada, 30 April–3 May 2018; pp. 1–21.
34. Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; Le, Q. Mnasnet: Platformaware neural architecture search for mobile. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 16–21 June 2019.
35. Dong, J.D.; Cheng, A.C.; Juan, D.C.; Wei, W.; Sun, M. DPP-Net: Device-Aware Progressive Search for Pareto-Optimal Neural Architectures. In Proceedings of the 2018 European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 540–555.

36. Kandasamy, K.; Neiswanger, W.; Schneider, J.; Póczós, B.; Xing, E. Neural architecture search with Bayesian optimisation and optimal transport. In Proceedings of the Advances in Neural Information Processing Systems conference (NeurIPS), Montreal, QC, Canada, 3–18 December 2018; pp. 2020–2029.
37. Liu, C.; Zoph, B.; Neumann, M.; Shlens, J.; Hua, W.; Li, L.-J.; Fei-Fei, L.; Yuille, A.; Huang, J.; Murphy, K. Progressive neural architecture search. In Proceedings of the 2018 European Conference on Computer Vision(ECCV), Munich, Germany, 8–14 September 2018; pp. 19–34.
38. Eshratifar, A.E.; Abrishami, M.S.; Pedram, M. JointDNN: An Efficient Training and Inference Engine for Intelligent Mobile Cloud Computing Services. *IEEE Trans. Mob. Comput.* **2019**. [CrossRef]
39. Li, E.; Zeng, L.; Zhou, Z.; Chen, X. Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 447–457. [CrossRef]
40. Eshratifar, A.E.; Esmaili, A.; Pedram, M. Bottlenet: A deep learning architecture for intelligent mobile cloud computing services. In Proceedings of the 2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), Lausanne, Switzerland, 29–31 July 2019; pp. 1–7.
41. Bateni, S.; Liu, C. Apnet: Approximation-aware real-time neural network. In Proceedings of the 2018 IEEE Real-Time Systems Symposium (RTSS), Nashville, TN, USA, 11–14 December 2018; pp. 67–79.
42. Yu, X.; Liu, T.; Wang, X.; Tao, D. On Compressing Deep Models by Low Rank and Sparse Decomposition. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Conference, Honolulu, HI, USA, 21–26 July 2017; pp. 7370–7379.
43. Szegedy, C.; Loffe, S.; Vanhoucke, V.; Alemi, A. Inception-v4, inception-resnet and the impact of residual connections on learning. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI'17), San Francisco, CA, USA, 4–9 February 2017; pp. 4278–4284.
44. McClellan, M.; Pastor, C.; Sallent, S. Deep Learning at the Mobile Edge: Opportunities for 5G Networks. *Appl. Sci.* **2020**, *10*, 1–27. [CrossRef]
45. Yang, K.; Xing, T.; Liu, Y.; Li, Z.; Gong, X.; Chen, X.; Fang, D. cDeepArch: A Compact Deep Neural Network Architecture for Mobile Sensing. *IEEE/ACM Trans. Netw.* **2019**, *27*, 2043–2055. [CrossRef]
46. Rago, A.; Piro, G.; Boggia, G.; Dini, P. Multi-Task Learning at the Mobile Edge: An Effective Way to Combine Traffic Classification and Prediction. *IEEE Trans. Veh. Technol.* **2020**, *69*, 10362–10374. [CrossRef]
47. Filgueira, B.; Lesta, D.; Sanjurjo, M.; Brea, V.M.; López, P. Deep Learning-Based Multiple Object Visual Tracking on Embedded System for IoT and Mobile Edge Computing Applications. *IEEE Internet Things J.* **2019**, *6*, 5423–5431. [CrossRef]
48. Mazzia, V.; Khaliq, A.; Salvetti, F.; Chiaberge, M. Real-Time Apple Detection System Using Embedded Systems With Hardware Accelerators: An Edge AI Application. *IEEE Access* **2020**, *8*, 9102–9114. [CrossRef]
49. Kong, H.; Wang, L.; Teoha, E.K.; Li, X.; Wang, J.-G.; Venkateswarlu, R. Generalized 2D principal component analysis for face image representation and recognition. *Neural Netw.* **2005**, *18*, 585–594. [CrossRef] [PubMed]
50. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 26 June–1 July 2016; pp. 770–778.