Channel-level Acceleration of Deep Face Representations

Adam Polyak, *Student member, IEEE,* and Lior Wolf, *Member, IEEE*

***Abstract*—A major challenge in biometrics is performing the test at the client side, where hardware resources are often limited. Deep learning approaches pose a unique challenge: while such architectures dominate the field of face recognition with regards to accuracy, they require elaborate, multi-stage computations. Recently, there has been some work on compressing networks for the purpose of reducing run time and network size. However, it is not clear that these compression methods would work in deep face nets, which are, generally speaking, less redundant than the object recognition networks, i.e., they are already relatively lean. We propose two novel methods for compression: one based on eliminating lowly active channels and the other on coupling pruning with repeated use of already computed elements. Pruning of entire channels is an appealing idea, since it leads to direct saving in run-time in almost every reasonable architecture.**

I. Introduction

FACE-based authentication has several advantages for mobile applications, compared, for example, to fingerprint readers. First, it does not require a specialized hardware. Second, as a non-contact biometric it can be collected without requiring an active cooperation. Third, it is suitable as a continuous biometrics i.e., periodically performing recognition in order to maintain a level of confidence regarding the user's identity. Done locally, on the mobile device, it enables a secure authentication that promotes privacy while reducing the amount of network usage.

Recently, deep learning approaches have shown an impressive level of face recognition performance that is significantly better than other available methods. As a result, deep learning methods hold the promise of being robust enough to the challenges of mobile authentication, including extreme pose and illumination. The prominent component of deep learning in current computer vision is the Convolutional Neural Network (CNN) [1], and the main goal of this work is to enable the efficient usage of such networks on mobile devices.

In CNNs, the weights connecting one layer to the next are shared across spatial locations and are given as convolution masks. Specifically, a convolution layer transforms a three dimensional input tensor $X \in \mathbb{R}^{m \times h \times w}$ into a three dimensional output tensor $Y \in \mathbb{R}^{n \times h \times w}$ where m, n are the number of channels in each tensor. This is done by applying n filters of spatial size $k \times k$ on a stack of m input channels of size $h \times w$. In general, the spatial domains might change between layers. However, for notational simplicity we assume that the spatial domain is fixed. The weights of a convolutional layer, combining all filters together, form a single tensor denoted W of size $n \times m \times k \times k$.

Since CNNs employ shared filters over local receptive fields, they enjoy a relatively compact representation. This compactness, in turn, enables the construction of deeper models. Such models would not be feasible using fully connected layers since they rapidly increase the number of network parameters beyond the capabilities of even modern hardware.

Despite their compactness, convolutional layers require a great number of computations. For example, consider the layer configuration given above – for each of the $h \times w$ output pixels a filter of size mk^2 is applied. Since there are n filters, a total of nmk^2hw multiplications are required.

In most desktop-computer implementations of CNNs, matrix multiplications are used to perform the convolutions. The matrices involved are: (i) a filter matrix - which contains the layer weights, and (ii) a patch matrix - which contains the input. Taking the layer described above as an example, the filter matrix is generated by reshaping the layer weights tensor from $n \times m \times k \times k$ to $n \times mk^2$ by flattening each filter to a single matrix row. The patch matrix is generated by unfolding (using an `im2col` operation) the input tensor. The unfold operation extracts m patches of spatial size $k \times k$ taken from all spatial locations, resulting in a matrix of size $mk^2 \times hw$. Finally, the activations of this layer are obtained by multiplying the filter matrix of size $n \times mk^2$ with the patch matrix of size $mk^2 \times hw$. While matrix multiplication is implemented very efficiently, it remains the main source of computational complexity. Reducing the number of input and output channels leads to an utilizable decrease in complexity by decreasing the size of the multiplied matrices.

Other platforms, specifically mobile platforms, do not support modern linear algebra libraries. As a result, convolutions are computed by directly computing each filter instead of converting the computation to matrix multiplication. In our implementation, we use the android port of a deep learning framework called Torch7 [2], in which computation is optimized using vectorized convolution code. Such implementations do not enjoy the optimization found in modern linear algebra libraries such as memory utilization. However, here too, reducing the number of channels is directly translated into an improvement in runtime through a reduction in the depth of each convolution and in the number of convolutions.

These improvements in performance are, therefore, platform independent and are readily achievable using the existing deep learning implementations. This is a result of eliminating entire channels and is in contrast to other methods of sparsification whose contribution requires specialized software or futuristic

AP and LW are with the Blavatnik School of Computer Science at Tel Aviv University.

hardware.

II. Previous work

A prominent example of deep learning success can be seen in the task of face recognition. Starting with the work of Taigman et al. [3], a neural network has been employed for extracting representations from face images that are shown to outperform humans. Sun et al. [4]–[7] further improve the state-of-art by employing various methods: (i) extraction of features from multiple face patches, (ii) combination of classification and contrastive loss and (iii) incorporating architectures into the domain of deep face recognition that are inspired by recent architectures that are used for object recognition [8], [9]. A recent work [10] further improves the training criterion by using triplet cost to increase the discriminability between identities. Although [10] trains a secondary model for mobile execution, the reported results of this model are significantly below the state of the art. Another contribution [11] suggests first training a feature extraction network followed by a metric learning network to reduce the feature to a low dimension.

The deep networks mentioned above, are all trained on large scale proprietary datasets, which are not publicly available. Yi et al. [12] built a publicly available dataset by mining images from the internet. Furthermore, they demonstrated the quality of the data collected by training a state-of-the-art network on it. Their network architecture is similar to that of the VGG model [8]. In our work, we use the same architecture suggested in [12] and accelerate it for usage on mobile platforms.

CNNs acceleration is of great interest in deep learning research. Acceleration methods can be split into two categories: (i) the creation of an approximated model either by training a new model or by modifying the existing one or (ii) the speedup of the computation without modification by improving the calculation method or better utilization of hardware.

Network mimicking by training a "shallow network" [13] or "fitnet" [14] trains a new model, called the student model, from scratch to mimic the original model, called the teacher model. Ba et al. [13] do so by setting the learning objective of the new network as a regression problem striving to replicate the teacher's output log probabilities by the student network. Romero et al. [14], train deeper student models with less filters in each layer ("fit"), suggesting that network depth can improve the model performance despite the overall reduction in size. In order to overcome the overall difficulty of deep network training, a multiple step training process is suggested. First, half of the student is trained, by solving a regression problem minimizing the difference between the teacher's and the student's activations. Afterwards, the entire student network is trained to mimic the teacher by using a combination of the classification loss and knowledge distillation [15].

Network decomposition [16]–[18], is another type of approximation, which exploits the low-rank structure of neural networks in order to perform filter decomposition that speeds up overall performance. Such factorization is done in a manner that reduces the computational complexity of the layer while well approximating the original layer. Recently, Zhang et al. [18] developed a decomposition method that utilizes low-rank assumptions in both the network input and the network filters.

Consider, as above, a layer with n filters operating on an input of a spatial size of $h \times w$ that follows a layer with m channels. Let the spatial size of the filters be $k \times k$.

Let $\hat{X}, \hat{Y}$ and $\hat{W}$ be the flattened version of the tensors defined above i.e. $\hat{X} \in \mathbb{R}^{(k^2m+1)\times hw}$, $\hat{Y} \in \mathbb{R}^{n\times hw}$ and $\hat{W} \in \mathbb{R}^{n\times(k^2m+1)}$. One additional row was added for bias. Using this flattened matrix notation, the layer activation is $\hat{Y} = \hat{W}\hat{X}$.

The low rank assumption implies that there exists a matrix $M \in \mathbb{R}^{n\times n}$ of rank $n' < n$ such that

$$\hat{Y} - \overline{Y} = M(\hat{Y} - \overline{Y}) \quad ,$$

where $\overline{Y}$ is the mean matrix of $\hat{Y}$. This is done by solving

$$\min_M \sum_i \|(\hat{Y_i} - \overline{Y_i}) - M(\hat{Y_i} - \overline{Y_i})\|$$

$$s.t. \quad rank(M) < n'$$

M is then decomposed to $M = PQ^T$. This results in the decomposition $W' = Q^TW$ and P. Put back into a CNN notation, the decomposition results in the splitting of the original layer (based on W) into two layers: one is a convolutional layer with filters of size 1×1 that is based on P; the other is a convolutional layer with filters of size $k \times k$ that implements W'. In comparison to the original layer, the second layer has only n' filters instead of the initial n.

The method proposed by us also uses 1×1 convolutions, see also Network In Network [19]. However, we reduce the layer dimension by pruning rather than decomposition.

Earlier implementations of network pruning [20]–[23] are used to improve model generalization and size while not targeting CNN run-time. By pruning network connections at the single neuron, i.e. single weights in each filter thus creating sparsification, such methods reduce the overall network complexity. The pruning we perform in the study presented here also removes some of the network connections. However, we target entire channels and therefore achieve run-time improvements.

Other pruning methods are motivated by run-time, however the gains are not realized in current deep learning architectures. Han et al. [24] propose a method to learn both the connections and weights of a network. This is done by pruning weights whose magnitude is under a given threshold followed by fine-tuning the pruned model. As a result, in the learned network, many of the weights are zeroed out. Unfortunately, the level of sparsification obtained is not enough to justify the usage of sparse matrix multiplications. The authors of [24] suggest that acceleration may be achieved using suitable future hardware designs. In contrast, our solution, which effectively reduces the number of required multiplications is software based, and the improvements in run-time are easily measured.

Lebedev et al. [25] speedup network inference time by pruning each filter in a group-wise manner generating sparse sub-filters. The induced sparse structure allows the computation of the convolutions via a dense "thin" matrix multiplication.

The sparse sub-filter, Q_i is defined for each input channel as a subset of the full filter i.e. $Q_i \subset \{1...k\} \otimes \{1...k\}$. Instead of computing the activations of the layer by multiplying a matrix of size $n \times mk^2$ with a matrix of size $mk^2 \times hw$, the computation is reduced to matrices of size $n \times \sum_{i=1}^{m} |Q_i|$ and $\sum_{i=1}^{m} |Q_i| \times hw$. This results a speedup of $mk^2 / \sum_{i=1}^{m} |Q_i|$.

In contrast, our methods prune layers at the channel level without changing the filter sizes. Our *inbound prune* reduces the number of incoming channels that each filter uses, while our *reduce and reuse* method prunes entire filters.

In [26] convolutions are computed on a sparse subset of each channel while restoring the non-computed convolutions via interpolation. Thus, it prunes a permanent subset of each filter connections to incoming channels. Our methods, by contrast, perform a much more aggressive pruning which removes the filter computation over entire channels or the filter itself.

In addition to the merit of improved runtime, pruning serves as a regularizer during neural network training [27]–[29]. Dropout [27] technique zeros network activations with probability p_{drop}. Then, during inference, all network activations are multiplied by $1/(1-p_{drop})$, to maintain the same level of expected activation. DropConnect [28] generalizes Dropout by pruning network connections instead of output units. Spatial-Dropout [29] extends Dropout to entire feature maps instead of single neurons, which was shown to be effective for fully-convolutional networks. However, the methods mentioned do not target the network inference time as they prune the network only during training.

Additional methods accelerate the neural network without modifying the network structure. One family of methods targets the way in which the layer output is calculated by using FFT based convolutions [30], [31]. Another family of methods improves hardware utilization [32], [33] employing various techniques such as low-level parallelism, effective memory usage and low precision arithmetic. The gains of such methods can be added to the gains that are made possible by our method.

III. Method overview

In this section, we provide an overview of our methods for accelerating deep convolutional neural networks; the following sections would provide the necessary details. In terms of positioning, the suggested methods continue a recent line of work [24]–[26] that utilizes pruning schemes, as opposed to retraining a mimicking network such as [13], [14]. Specifically, we reduce the network inference time by pruning either the input or the output channels of each layer. This is unlike previous work [20]–[23] on network pruning, which typically focuses on the neuron level.

A. The scratch face recognition model

While a number of network architectures have been proposed for deep representation of faces, the scratch network [12] has a few significant advantages, which make it suitable for our study. First, it relies solely on convolutions, and does not employ fully connected layers, which burden the size of the network and create a hefty memory footprint. Second, and more importantly, this architecture was shown to be able to train well on the relatively noisy CASIA dataset [12]. While other networks [3], [7], [10] have exhibited somewhat better performance, they were trained on proprietary datasets of better quality and larger cardinality.

The architecture of the scratch feature extraction network is detailed in Table I. Note that suitable padding is applied such that convolutional layers do not change the spatial dimensions from one layer to the next. Only max- and average-pooling reduce the size of the activation maps. The table also reports the percent of the total network runtime devoted to each layer.

For the purpose of benchmarking on the LFW benchmark [34], we use the scratch network in order to extract face feature representation. This representation is the collection of the 320 activations of the Avg Pool layer. Afterwards, we train a Joint Bayesian model [35] based on these extracted features for the face verification task.

B. The experimental pipeline

Training is done on the public dataset published in [12] which contains 494,414 images of 10,575 identities. A common practice in face recognition is to preprocess the data prior to feature extraction. As suggested by previous work [3], [7], [10], [12], for each image the face are detected and 2D-aligned. Specifically, we use the method proposed by [36], which detects 9 facial features followed by an affine transform to place the detected features at 9 anchor points. We further augment the data set by flipping each image horizontally. This increases the size of the training set by 2. Finally, 10% of the data is used for validation during the network training. The other 90% are used to train the network. All reported models were trained on the same data partition.

During training, on top of the model, we used a fully connected layer together with a soft max activation to generate a distribution over the dataset identities. The model was trained using Stochastic Gradient Descent (SGD) with a momentum parameter of 0.9, a learning rate 0.01, and with a batch size of 128. We also applied weight decay of 0.0005 on the fully connected layer. As is commonly done, the learning rate was manually reduced once the network improvement reached saturation. To ensure simplicity, we trained the model using classification criterion only as opposed to the contrastive criterion combination employed in the original paper.

The model training and pruning was done using Torch7 [37]. We used a port of Torch7 [2] for the deployment and benchmarking of the various models on mobile.

C. Pruning

Once we train a model with satisfying accuracy, we opt to speed the model without loss of accuracy while avoiding the need to train it from scratch. For the task at hand, pruning is an appealing method due to the heavy reliance on training that was already performed, as long as the pruning is done without reducing the model accuracy.

The *Inbound Prune* approach that we suggest focuses on reducing the number of channels each filter uses by eliminating channels that do not contribute significantly to the information

TABLE I
THE SCRATCH MODEL BY THE AUTHORS OF [12], WHICH IS THE BASELINE MODEL IN OUR EXPERIMENTS. THE NETWORK STARTS WITH A GRAY SCALE INPUT IMAGE OF SIZE $1 \times 100 \times 100$ PIXELS, AND RUNS THROUGH 10 CONVOLUTIONAL LAYERS INTERLEAVED WITH MAX POOLING LAYERS. FOLLOWING A SPATIAL AVERAGE POOLING AT THE END OF THE PROCESS, A REPRESENTATION OF SIZE 320 IS OBTAINED.

Layer	Filter size / Stride	# Channels	# Filters	Output size	Runtime (%)
Conv11	3×3 / 1	1	32	100×100	1.2
Conv12	3×3 / 1	32	64	100×100	19.6
Max Pool	2×2 / 2	64	–	50×50	–
Conv21	3×3 / 1	64	64	50×50	9.2
Conv22	3×3 / 1	64	128	50×50	25.9
Max Pool	2×2 / 2	128	–	25×25	–
Conv31	3×3 / 1	128	96	25×25	10.7
Conv32	3×3 / 1	96	192	25×25	14.3
Max Pool	2×2 / 2	192	–	12×12	–
Conv41	3×3 / 1	192	128	12×12	5.0
Conv42	3×3 / 1	128	256	12×12	6.9
Max Pool	2×2 / 2	256	–	6×6	–
Conv51	3×3 / 1	256	160	6×6	3.5
Conv52	3×3 / 1	160	320	6×6	3.6
Avg Pool	6×6 / 1	320	–	1×1	–

the filter extracts. The amount of information each channel contributes is measured by the variance of the specific channel activation output. We do not consider directly the model accuracy during the pruning process. Instead, we fine-tune the model obtained after each prune in order to allow it to adapt. The pruning of the entire network is done sequentially on the network layers: from lower layers to the top ones, pruning is followed by fine-tuning, which is followed by pruning of the next layer. The speedup of the inbound pruning scheme is achieved directly by reducing the amount of computation that each filter performs. While originally, each filter required $O(mk^2)$ multiplications, after pruning only $O(ck^2)$ multiplications are required, where $c < m$ is the number of channels used by the filter. Fig. 1(b) illustrates the inbound pruning process.

The second method suggested in our work is *Reduce and Reuse*. While the previous method targets the inbound connections of a convolution layer, this method prunes the number of channels outputted. Such pruning results in the removal of some of the layer filters. The same variance-based criterion as before is used, this time on the output of each filter. The method utilizes the fact that there is sufficient information in a subset of the layer output to allow the next layer filters to extract important features. Following the prune, we reuse the output left to reconstruct the pruned channels. Fig. 1(c) illustrates the pruning process.

IV. INBOUND PRUNE

The inbound prune method targets the number of channels that each filter operates on. Our hypothesis is that each input channel has a different level of contribution to the feature map outputted by each filter. As a result, we may omit the computation of the filter on channels with low contribution, and suffer only a minor decrease in accuracy.

In order to detect such low importance channels, we leverage the pruning scheme used previously on single connections [22]. Specifically, the notion of *smallest contribution variance* – $min(\sigma)$ is used. Originally, [22] define scores for the contribution of a single weight to the activation of a single neuron. Below, we generalize this measure to the contribution of each channel to the filter activation by using the notion of *channel activation*.

Recall that X, Y are the 3-dimensional input and output tensors of a convolutional layer with 4-dimensional weight tensor W. The activation of filter t is

$$Y_t = \sum_{s=1}^{m} W_{ts} * X_s$$

Therefore, the *activation of channel* s of filter t is defined by $W_{ts} * X_s$.

The *contribution variance of channel* s in filter t is defined to be

$$\sigma_{ts} = var(||W_{ts} * X_s||_F)$$

where X_s is the s channel of the input, sampled from the training dataset.

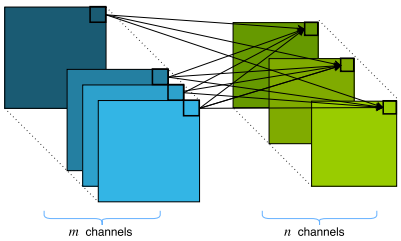
Typically, the variance score is distributed in a bi-modal manner: some channels are scored highly, while some present activations that result in low scores. Fig. 2 shows the distribution of channel scores for 3 randomly sampled filters from layers Conv12,Conv21 and Conv22 (as defined in Table I), i.e., for each of the depicted layers, we plot results for three random t. As can be seen, the contribution of a specific channel s is not uniform between filters, which explains why each filter has different channels pruned. To verify, we tested an alternative method in which we pruned the same channels from all filters and the results achieved by this method were less satisfying.

Once we compute the contribution variance for all channels in all filters, we prune all filter connections to a given channel where the contribution variance is below a threshold τ (see below). As a result, the pruned layer operation is defined by

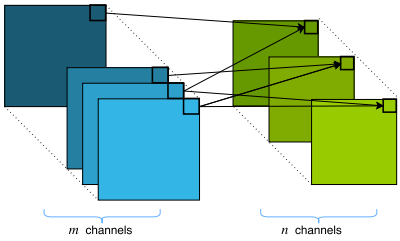
$$Y_t = \sum_{s \in \Lambda_t} W_{ts} * X_s$$

where Λ_t is the set of channels filter t operates on after the prune operation.

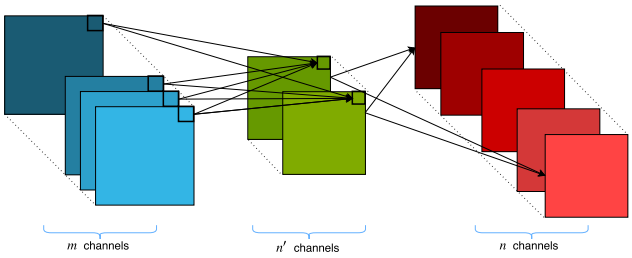
The number of multiplications performed by the pruned layer is less than that of W. More concretely, the complexity of the original layer is $O(nmk^2)$ while the pruned layer requires $O(ck^2)$ where $c = \sum_{t=1}^{n} |\Lambda_t|$ is the number of channels left



(a) Original layer

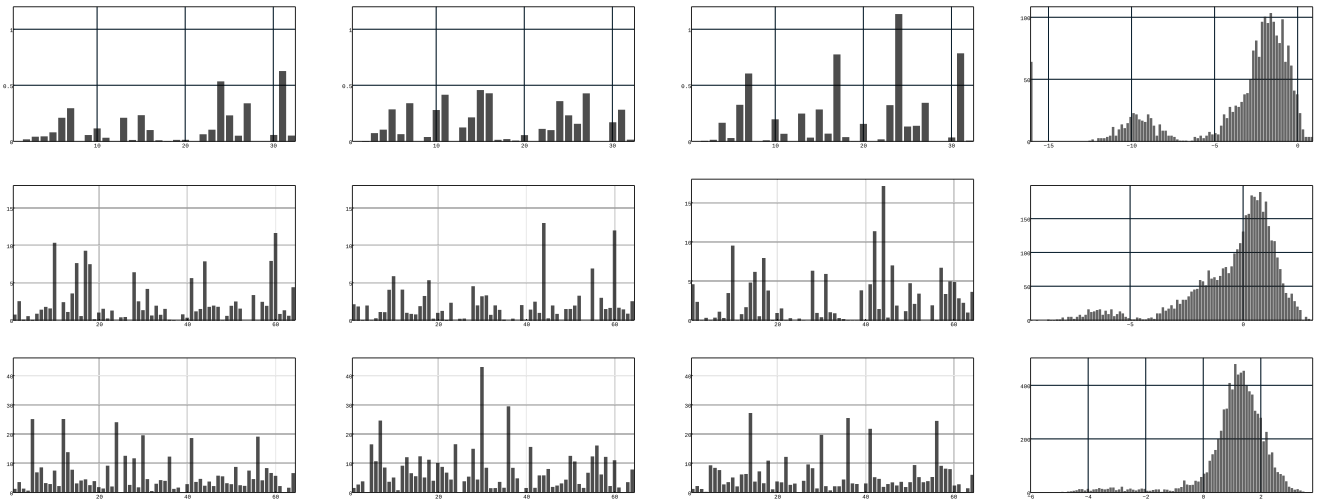


(b) Inbound prune



(c) Reduce and reuse

Fig. 1. Illustration of pruning schemes. (a) The original convolution layer, each of the n output channels is computed using all m input channels. (b) The layer produced by inbound prune. The number of input channels for each filter is reduced, such that each filter employs a suitable subset of the channels. (c) The layer produced by reduce and reuse. Here, the number of filters is reduced from n to n' followed by a new layer that reconstructs back the original channels.



(a) (b) (c) (d)

Fig. 2. **Distribution of channel contributions.** Depict score distributions for layers Conv12,Conv21,Conv22 by order in rows. (a-c) Show (y-axis) the standard deviation score $\sqrt{\sigma_{ts}}$ of each channel contribution for three random filter outputs. Each of the subplots shows results for one random t, and the x-axis are the indices of s. (d) A histogram of the score for the entire layer in log scale, created by pooling the scores from all channels and all filters. Channels with zero standard deviation are depicted as the bar on the far left side of each distribution.

after pruning. Therefore, we achieve a theoretical speedup of $O(\frac{nm}{c})$.

A. *Fine-tuning*

Even though pruning aims to inflict only a minor decrease in accuracy, we found in our experiments that fine-tuning the pruned model, after each layer was pruned, allows it to adapt to the modified activations of the pruned layer. More specifically, fine-tuning allows us to achieve three goals: (i) Prior to pruning, the model is at a local optima in parameter space. Once pruned, the model is no longer at a local optima in the parameter space. By fine-tuning, we search a new local optima while retaining the pruned model structure. (ii) Although we attempt to prune layers with minimal effect, the output of pruned layers is still changed. Fine-tuning allows the next layer in line to adapt to the pruned layer output. (iii) In

Algorithm 1 The inbound pruning scheme

The algorithm receives as input the sampled channel activation for each channel of each filter given as a tensor $\Delta \in \mathbb{R}^{N\times n\times m\times h\times w}$, N being the size sampled, and a threshold τ. The algorithm outputs Λ - a set of channels left after pruning for each filter.

Input: Δ, τ
Output: Λ

1: For each filter t and each channel s, compute over Δ the contribution variance σ_{ts} .
2: For each filter t and each channel s if $\sigma_{ts} \geq \tau$ then $\Lambda_t \leftarrow s$

Algorithm 2 The reduce and reuse pruning scheme

The algorithm receives as input the sampled channel activation filter activation given as a tensor $\Delta \in \mathbb{R}^{N\times n\times h\times w}$, where N is the size sampled, and a percentile μ. The algorithm outputs Λ, which is the set of channels remaining after pruning.

Input: Δ, μ
Output: Λ

1: For each channel $t \in [1..n]$ compute the activation variance - σ_t over Δ.
2: $\upsilon \leftarrow \mu$-th percentile of $\{\sigma_t\}_{i=1}^{n}$
3: For each filter t if $\sigma_t \geq \upsilon$ then $\Lambda \leftarrow t$

our method, pruning is done sequentially. Therefore, adapting the model after each modification is crucial in order to reduce the accumulated error.

The inbound pruning scheme for a single layer is summarized in Algorithm 1. As mentioned, for whole model acceleration, we prune each layer sequentially from the lowest layers to the topmost layers.

V. Reduce and Reuse

Our second scheme targets the outbound channels of a layer. Our hypothesis is that in a convolution layer, a subset of its output contains enough information to allow the next layer in line to extract the viable features with minor classification loss.

a) Reduce: In order to find candidates for pruning, we compute the variance of each filter output over a sample of the training set. Formally, using the same annotation as before, the activation of a given channel is denoted by $W_i * X$ and the variance is

$$\sigma_t = var(||W_t * X||_F)$$

where X is the layer input, and W_t is one of the filters of the layer. Next, we prune all filters whose score is below the percentile μ. In other words, we decide beforehand what would be the number of the channels $n' = \mu n$ that we keep. As a result, the pruned layer weight tensor is $W \in \mathbb{R}^{n'\times m\times k\times k}$ where $n' < n$ is the number of channels outputted by the pruned layer.

b) Reuse: The performed pruning requires adaptation since the next layer expects to receive input in the size prior to the pruning and with similar patterns of activations. Therefore, we reuse the remaining output channels to reconstruct each pruned channel, by the use of linear combinations. Formally, let $Y_i \in \mathbb{R}^{n\times hw}$ be the output of a layer with n filters, where each row is an output of a filter. Let $Y_i' \in \mathbb{R}^{n'\times hw}$ be the output of the pruned layer with the remaining n' layers, as observed on the ith sample. By reusing, we seek to find $A \in \mathbb{R}^{n\times n'}$ such that

$$min_A \sum_i \|Y_i - AY_i'\|_2^2 \quad (1)$$

The operation of A is added to the network by introducing a convolution layer with filters of size 1×1, which holds the elements of A. Note that 1×1 convolution layers, which can be seen as dimensionality rearrangement methods, were introduced recently by [19] and, as mentioned, were used by [18] as part of the decomposition process.

Once A is recovered, AY_i' is used in lieu of the original Y_i.

The speedup achieved is based on the number of filters removed and the overhead required to restore those channels. The theoretical computational complexity of a layer with n filters of size $m\times k\times k$ is $O(nmk^2)$. Once the layer is pruned to n' filters, it will require $O(n'mk^2)$ operations. The additional layer of 1×1 convolutions will require $O(n'n)$ operations. Therefore, the total speedup ratio is $O(\frac{nmk^2}{n'n+n'mk^2})$. As before, we fine-tune the entire model before pruning the next layer in order to allow the next layer to adapt. The reduce and reuse pruning scheme is summarized by Algorithm 2.

It is important to note the overall memory footprint of the network does not increase due to the added 1×1 convolutional layer. Considering the layer described above, the added layer will have $n'n$ parameters. However, the number of parameters in the convolutional layer is reduced to $n'mk^2$, whereas before reduction the size is nmk^2. For $n' \ll n$, the pruning scheme will still reduce the size of the network and as a result the model memory footprint.

A. Hybrid pruning

We also consider the notion of combining the two schemes. Such a combination is plausible, since each of our pruning schemes targets different sections of a layer - inbound or outbound. The hybrid method is performed by first applying reduce and reuse followed by inbound pruning. This combination allowed us to achieve an acceleration of $\times2.65$. We note that the speedup ratio from applying inbound prune on an already pruned model (by reduce and reuse) is less effective than when applied on the original model. Nevertheless, we show in our experiments that we are able to achieve a greater speedup by combining the two methods.

VI. Implementation

In this section, we elaborate on the architectures and training procedures used to train the reported models.

A. Inbound prune

The model was pruned sequentially, one layer at a time from bottom to top. Each filter operating on an input stack

TABLE II
INCOMING PRUNE ARCHITECTURES PRODUCED BY THE PRUNING SCHEME. WE REPORT THE NUMBER OF INPUT CHANNELS TO FILTER CONNECTIONS IN BASELINE AND IN THE PRUNED MODEL. WE ALSO REPORT THE PERCENT OF SURVIVING CHANNELS/FILTER INTERACTIONS.

Layer	#channels in baseline	#channels after prune	percent in-channels left after prune
Conv11	32	32	100
Conv12	2048	941	45.9
Conv21	4096	2213	54.0
Conv22	8192	5199	63.4
Conv31	12288	8857	72.0
Conv32	18432	13847	75.1
Conv41	24576	17212	70.0

of m channels was split into m convolution layers with a single filter of size $1 \times 3 \times 3$. Those layers operated in parallel on the input channels. As a result, each layer computes the activation of a single channel. Next, the contribution variance of each channel was calculated. The variance was calculated over 3,000 samples drawn randomly from the training set.

During our experiments, the threshold τ was chosen empirically based on the accuracy achieved after pruning the model, prior to fine-tuning. We do not let the validation accuracy on the CASIA dataset drop below 84%. The model was fine-tuned using SGD with a learning rate of 0.01, momentum of 0.9 and batch size of 128. Each model was fine-tuned for a maximum of 30 epochs or until there were 5 successive epochs of no error improvement. Table II shows the architecture of the pruned models, each column reports the number of inbound channels per layer.

B. Reduce and reuse

Table III depicts the architecture produced by the second pruning scheme. There are three architectures: RR50%, RR75%, and RR90%-50%. The top layers for these architectures are the same as in the baseline model, since their prune resulted in a sharp decrease in model accuracy. The RR50% and the RR75% architectures denote a cut of 50/75% of the layers. Since cutting more than 50% after a certain layer is highly detrimental (as discussed below), the third architecture cuts by 90% up to layer Conv32, and only by 50% afterwards.

As with inbound prune, the scheme is employed sequentially, one layer at a time from bottom to top. The contribution variance of each filter is computed by calculating the variance of the layer output channel norm. Given a percentile μ, we prune all filters below this percentile. The variance based score was calculated over 1,000 samples from the training set. Next, we solved the minimization problem described in Eq. 1. The solutions are the weights of the 1×1 layer. The resulting model is fine-tuned using SGD with an initial learning rate of 0.01, a momentum 0.9, and a batch size of 128. We applied the same stopping condition as in the inbound prune method.

C. Fitnets

Next, we describe the architectures and training of the fitnet models [14], which were used as a recent literature baseline. As the teacher model, we used the full scratch model as described in Table I. The fit models architectures are detailed in Table IV. The models follow a similar structure of consecutive convolution layers with a spatial filter size of 3×3 with a stride of 1. After each convolution, we used the parametric rectified linear activation [38]. The convolutions are zero padded to retain the input spatial size. Each stack of convolution layers is followed by a non-overlapping max pooling layer with a spatial size of 2×2. On top of the last convolution layer, an average pool of spatial size 6×6 is used in order to reduce each feature map to 1×1 generating the facial representation vector. During training, each model is followed by a fully connected layer together with a soft max activation layer to produce class probabilities.

TABLE III
REDUCE AND REUSE ARCHITECTURES

RR50%	RR75%	RR90%-50%
conv $3 \times 3 \times 32$ conv $3 \times 3 \times 32$ conv $1 \times 1 \times 64$ pool 2×2	conv $3 \times 3 \times 32$ conv $3 \times 3 \times 16$ conv $1 \times 1 \times 64$ pool 2×2	conv $3 \times 3 \times 32$ conv $3 \times 3 \times 7$ conv $1 \times 1 \times 64$ pool 2×2
conv $3 \times 3 \times 32$ conv $1 \times 1 \times 64$ conv $3 \times 3 \times 64$ conv $1 \times 1 \times 128$ pool 2×2	conv $3 \times 3 \times 16$ conv $1 \times 1 \times 64$ conv $3 \times 3 \times 32$ conv $1 \times 1 \times 128$ pool 2×2	conv $3 \times 3 \times 7$ conv $1 \times 1 \times 64$ conv $3 \times 3 \times 13$ conv $1 \times 1 \times 128$ pool 2×2
conv $3 \times 3 \times 48$ conv $1 \times 1 \times 96$ conv $3 \times 3 \times 96$ conv $1 \times 1 \times 192$ pool 2×2	conv $3 \times 3 \times 24$ conv $1 \times 1 \times 96$ conv $3 \times 3 \times 48$ conv $1 \times 1 \times 192$ pool 2×2	conv $3 \times 3 \times 10$ conv $1 \times 1 \times 96$ conv $3 \times 3 \times 20$ conv $1 \times 1 \times 192$ pool 2×2
conv $3 \times 3 \times 128$ conv $3 \times 3 \times 256$ pool 2×2	conv $3 \times 3 \times 128$ conv $3 \times 3 \times 256$ pool 2×2	conv $3 \times 3 \times 64$ $1 \times 1 \times 128$ conv $3 \times 3 \times 128$ $1 \times 1 \times 256$ pool 2×2
conv $3 \times 3 \times 160$ conv $3 \times 3 \times 320$ pool 2×2	conv $3 \times 3 \times 160$ conv $3 \times 3 \times 320$ pool 2×2	conv $3 \times 3 \times 80$ conv $1 \times 1 \times 160$ conv $3 \times 3 \times 160$ conv $1 \times 1 \times 320$ pool 2×2
avg 6×6	avg 6×6	avg 6×6

The models were trained using the scheme presented in [14]. We used exactly the same data split used in the baseline model training i.e. 90-10 for train and test. The model weights were initialized using the scheme reported in [38], which leads to a faster convergence. For both stages of the fitnet training process, we trained the models with SGD, annealing the learning manually, once we detected that the model accuracy had stopped improving. We used an initial learning rate of 0.001 and batch size of 128 throughout all of the training phase with a momentum of 0.9.

During the hint-based training phase, each model was trained for 170 epochs or until the model accuracy stopped improving. Next, we trained the model using knowledge distillation. λ was initialized to 4 and was decayed linearly for 150 epochs to 1. In the following epochs, λ was remained unchanged. The temperature parameter was set to 3. We stopped the knowledge distillation phase when we detected that the model improvement rate slowed down considerably.

TABLE IV
FITNET ARCHITECTURES. FROM LEFT TO RIGHT THE ARCHITECTURES BECOME DEEPER.

Fitnet 1	Fitnet 2	Fitnet 3
conv $3 \times 3 \times 16$ conv $3 \times 3 \times 16$ conv $3 \times 3 \times 16$ pool 2×2	conv $3 \times 3 \times 16$ conv $3 \times 3 \times 16$ conv $3 \times 3 \times 16$ conv $3 \times 3 \times 16$ pool 2×2	conv $3 \times 3 \times 16$ conv $3 \times 3 \times 16$ conv $3 \times 3 \times 32$ conv $3 \times 3 \times 32$ pool 2×2
conv $3 \times 3 \times 32$ conv $3 \times 3 \times 32$ conv $3 \times 3 \times 48$ pool 2×2	conv $3 \times 3 \times 32$ conv $3 \times 3 \times 32$ conv $3 \times 3 \times 32$ conv $3 \times 3 \times 48$ pool 2×2	conv $3 \times 3 \times 32$ conv $3 \times 3 \times 48$ conv $3 \times 3 \times 48$ conv $3 \times 3 \times 64$ pool 2×2
conv $3 \times 3 \times 64$ conv $3 \times 3 \times 64$ conv $3 \times 3 \times 80$ pool 2×2	conv $3 \times 3 \times 48$ conv $3 \times 3 \times 64$ conv $3 \times 3 \times 64$ conv $3 \times 3 \times 80$ pool 2×2	conv $3 \times 3 \times 80$ conv $3 \times 3 \times 80$ conv $3 \times 3 \times 96$ conv $3 \times 3 \times 112$ pool 2×2
conv $3 \times 3 \times 96$ conv $3 \times 3 \times 96$ conv $3 \times 3 \times 112$ pool 2×2	conv $3 \times 3 \times 96$ conv $3 \times 3 \times 96$ conv $3 \times 3 \times 112$ conv $3 \times 3 \times 112$ pool 2×2	conv $3 \times 3 \times 96$ conv $3 \times 3 \times 128$ conv $3 \times 3 \times 112$ conv $3 \times 3 \times 144$ pool 2×2
conv $3 \times 3 \times 128$ conv $3 \times 3 \times 160$ conv $3 \times 3 \times 320$ pool 2×2	conv $3 \times 3 \times 96$ conv $3 \times 3 \times 128$ conv $3 \times 3 \times 144$ conv $3 \times 3 \times 160$ pool 2×2	conv $3 \times 3 \times 112$ conv $3 \times 3 \times 128$ conv $3 \times 3 \times 144$ conv $3 \times 3 \times 160$ pool 2×2
avg 6×6	avg 6×6	avg 6×6

D. Low-rank network decomposition

We next describe the architectures and decomposition scheme of the models generated by our implementation of [18], which is another very recent baseline method.

We used the "Asymmetric non-linear reconstruction" method suggested in [18], which seems better than the alternatives in that paper. This method decomposes each convolutional layer of size $n \times m \times k \times k$ into two convolutional layers of size $n' \times m \times k \times k$ and $n \times n' \times 1 \times 1$. In order to evaluate our method, we chose $\frac{n'}{n} = 0.5$, $\frac{n'}{n} = 0.25$ and $\frac{n'}{n} = 0.1$ for each layer we decomposed, keeping the channel-reduction ratio the same as in the methods that we suggested. In our experiments, we decompose each layer using 1,000 samples from our dataset which contains 494,414 images. This is a similar ratio to the original paper, which sampled 3,000 from the imagenet dataset which contains 1.2 million images. The "non-linear" optimization problem

$$\min_{M,b} \sum_i \|r(y_i) - r(My_i + b)\|_2^2$$

$$s.t. \quad rank(M) \leq n'$$

where $M \in \mathbb{R}^{n \times n}$ is the low-rank matrix discussed above, b is the bias of the added $1x1$ convolutional layer and $r(\cdot) = max(\cdot, 0)$ is the rectified linear unit (ReLU) used between the network convolutional layers. $\{y\}_{i=1}^N$ are the sampled activations of the layer that is being decomposed. As mentioned above, the convolutional layer is split to two convolutional layers by the $M = PQ^T$ decomposition.

We solved the latter optimization problem using the iterative solver employed by [18]. In our implementation of the solver we used the same number of iterations and hyper parameters. The architecture produced by this method is identical to the ones produced by the RR method, which are displayed in Table III. As before, we do not decompose the top layers of the network since that degraded the network performance beyond an acceptable ratio. The writers of [18] claim that the approximated model is very sensitive to fine-tuning (i.e. sensitive to the selection of a learning rate) and show that they are able to achieve "very good accuracy even without fine-tuning" - therefore in our implementation we do not apply fine-tuning.

VII. RESULTS

We use the scratch model [12], as depicted in Table I, as our baseline for the evaluation of our methods. Models are evaluated in two different ways. First, we measure the model accuracy by the classification accuracy on the CASIA dataset which we split to 90% training and 10% test. Second, we measure the score on the LFW benchmark [34] in the unrestricted mode. LFW results are mean and Standard Error estimated over fixed ten cross-validation splits. In addition, the model efficiency is captured by measuring the running time on a Samsung Galaxy S6 device which is our target platform. The time results are reported in seconds; Run time was measured as the mean of 100 forward passes of a single image. The baseline network achieves the accuracy of 95.12%±1.53 on LFW and 86.04% classification accuracy on CASIA with an execution time of 0.512 seconds.

In all of our experiments, pruning is done from the layer Conv12 until a certain layer. Hence, when we report the pruning of Conv22, we mean that Conv12, Conv21, and Conv22 were all pruned. This makes sense since pruning the lower layers is more beneficial and less detrimental to the overall performance.

A. Inbound pruning

We conduct a few experiments using the first pruning method that verify the effectiveness of the used criterion as well as the overall performance.

Random vs variance based. In the first experiment, we verify that the filter input variance score is a good measure for pruning the incoming channels of a convolution layer. We do so by removing a fixed 50% of incoming channels without fine-tuning the network after each prune. Fig. 3 shows accuracy on the CASIA and LFW datasets after each prune. The CASIA validation success rate after pruning the first three layers stands at 82% using the variance based measure while the random pruning collapses to 14.5%. We also note that the rate of accuracy drop is dramatically faster when using random pruning.

The effect of fine-tuning. Next, we verify that fine-tuning reduces the accumulation of error rate between layers by comparing the accuracy of the models with and without fine-tuning after prune. Indeed, Table V shows that the detrimental effect of pruning is mitigated by performing fine-tuning. However, this effect is visible only in higher layers. At layer Conv21, post pruning results are 94% on the LFW benchmark for both methods. At Conv41, fine-tuning results are 93% compared to

TABLE V
INBOUND PRUNING SUMMARY. FOR EACH PRUNED MODEL WE REPORT MODEL CLASSIFICATION ACCURACY BEFORE AND AFTER FINE-TUNING, RUNNING TIME IN SECONDS, NUMBER OF MULTIPLICATIONS IN G-OPS AND PARAMETERS. EACH ROW INDICATES PRUNING DONE UP TO THE INDICATED LAYER.

	No FT		With FT				
Model	CASIA	LFW	CASIA	LFW	Time	# params	# mults
Original	86.04	95.12±1.53	–	–	0.512	1752768	1.48
Conv12	85.07	94.15±1.99	85.97	94.50±1.88	0.415	1742805	1.29
Conv21	84.17	94.00±1.94	86.07	94.55±1.86	0.393	1725858	1.20
Conv22	82.01	93.95±1.82	86.02	94.41±1.87	0.376	1698921	1.07
Conv31	75.20	93.06±1.95	85.88	94.26±1.65	0.352	1668042	1.03
Conv32	59.25	91.91±2.09	85.60	94.01±1.80	0.338	1626777	0.98
Conv41	32.90	88.43±1.77	84.74	93.92±1.87	0.322	1560501	0.96

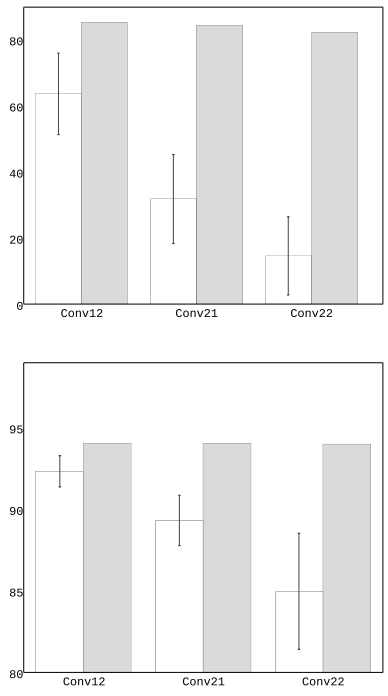


Fig. 3. **Random vs. Variance.** We compare the results of variance based inbound pruning to random selection. From each layer, we prune 50% of the input channels and report the accuracy for CASIA (top) and LFW (bottom). Full bars depict variance based pruning; empty bars depict the accuracy for random based prune. For the random selection, we average the accuracy over 10 experiments. As can be seen, pruning by the variance score is preferable by a large gap.

88.43% – a difference of 4.5% in error rate. The increasing gap can be explained by the accumulated error caused by pruning all previous layers.

Model acceleration. Finally, we evaluate the effect of pruning each layer sequentially on the speed of representing a facial image. The accuracy and performance results are summarized in Table V. The percentage of pruning for each layer was chosen empirically based on the model's CASIA accuracy (prior to fine-tuning) so that the accuracy threshold was chosen to be above 84%. We note that we cannot apply our method on layer Conv11 since it operates on a single channel gray scale image. In addition, since the contribution variance of channels between filters is not uniform, the number of channels pruned from each filter is different. Finally, we achieved a speedup of roughly $\times 1.5$ with an accuracy drop of 1.11% on LFW.

Note that the reduction in the number of multiplications is not fully realized in the actual run time. In order to separate the contribution of the channels, such that different channels contribute to different filters, we had to separate the channels, which created an additional overhead. Put differently, the software package is optimized for the case in which it is assumed that all channels contribute to all filters.

B. Reduce and reuse

Table VI summarizes our results for the reduce and reuse pruning method. In this experiment we reduce and reused the model layers sequentially using different settings. First, we evaluated random pruning where we randomly pruned each layer. Second, we apply variance based reduce and reuse. Third, we add fine-tuning after each layer. In addition, we compare our results with the decomposition suggested by Zhang et al. [18]. We do not reduce and reuse layer Conv11 since the amount of reduction required in order to achieve a speedup results in a major decrease in accuracy.

Reduce and reuse is evaluated in three models: RR50%, RR75% and RR90%-50%. RR50% and RR75% are models generated by applying the same reduce ratio to all layers. RR-90%-50% is the model generated by applying 90% prune up to Conv32 and continuing with 50% prune afterwards. We chose this ratio empirically by observing that either 75% or 90% degraded the accuracy significantly after Conv32.

Initially, we compare the variance based RR with random based. While there is not much difference in accuracy when pruning 50% of each layer, the advantage becomes clear in 75% pruning. When comparing 75% prune of Conv22, the LFW accuracy of variance based RR is 94.02% and for the random RR it is 93.6%. In addition, the accuracy drop with random pruning is faster. Random RR75% CASIA accuracy drops from 81.78% to 52.40% between Conv22 to Conv32 while variance based RR75% drops from 82.32% to 59.26%.

As with inbound prune, fine-tuning improves the prune scheme effectiveness in two ways. First, it improves the accuracy achieved post pruning. For example, when 75% of the filters up to layer Conv22 are pruned we are able to increase the model LFW accuracy by 0.4%. Furthermore, fine-tuning allows the model to adapt to the pruning process, and as a result the subsequent error accumulates much more slowly.

In the table, we refer to the method proposed by Zhang et al. [18] as "low-rank approximation". We compare their

TABLE VI

RR ACCURACY: ACCURACY FOR RR METHODS AND OTHER METHODS OF THE SAME ARCHITECTURE. WE EVALUATE RANDOM, VARIANCE BASED, VARIANCE BASED AND FINE-TUNED. WE ALSO COMPARE IT WITH LOW-RANK APPROXIMATION. FOR THE FT APPROACH, WE ALSO REPORT THE ACCURACY PRIOR TO FT. THE MODEL IS PRUNED SEQUENTIALLY, I.E. THE RESULTS ARE INDICATED FOR PRUNING UP TO THE INDICATED LAYER. LOW INDICATES RESULTS THAT WERE CLEARLY NON-COMPETITIVE AND WERE, THEREFORE, NOT PRIORITIZED FOR EXACT ESTIMATION.

Reduction	Layer	Random RR		Variance based RR		Variance based RR followed by FT		Low-rank approximation [18]	
		CASIA	LFW	CASIA	LFW	CASIA	LFW	CASIA	LFW
50%	Conv12	86.039	94.55±1.71	86	94.52±1.85	86.05 (86.01)	94.45±1.79	85.97	94.58±1.63
	Conv21	85.788	94.45±1.78	85.95	94.38±1.94	86.02 (85.95)	94.55±1.82	85.96	94.38±1.75
	Conv22	85.604	94.28±1.96	85.64	94.25±2.01	85.92 (85.64)	94.45±1.78	85.88	94.35±1.81
	Conv31	83.170	94.01±1.91	83.14	93.80±1.79	84.87 (83.14)	94.33±1.80	85.05	94.26±1.85
	Conv32	81.151	93.45±1.90	80.93	93.86±1.58	84.67 (81.93)	94.40±1.80	84.79	93.90±2.00
	Conv41	59.241	91.18±2.03	56.77	90.75±1.65	82.07 (63.22)	93.11±1.61	79.55	93.06±1.75
	Conv42	44.96	89.23±2.01	46.12	89.25±1.44	81.27 (64.81)	93.01±1.95	78.69	92.96±2.12
75%	Conv12	85.89	94.48±1.82	85.49	94.28±1.69	85.60 (85.49)	94.53±1.96	84.47	94.13±1.61
	Conv21	81.51	93.71±2.04	83.69	94.03±1.81	85.45 (83.77)	94.41±1.80	82.25	93.80±1.70
	Conv22	81.78	93.60±1.98	82.32	94.02±2.00	85.32 (83.18)	94.41±1.69	80.44	93.71±2.09
	Conv31	64.26	91.13±1.63	66.015	91.76±2.07	85.15 (67.99)	94.18±1.49	78.81	93.28±2.31
	Conv32	52.40	89.31±1.99	59.26	90.25±2.20	85.18 (72.84)	94.12±1.60	78.82	93.00±2.01
	Conv41	LOW	LOW	LOW	LOW	84.26 (15.98)	93.38±1.71	51.13	90.45±1.97
	Conv42	LOW	LOW	LOW	LOW	84.69 (33.94)	93.61±1.74	50.55	89.38±1.86
90%-50%	Conv12	79.23	93.41±1.73	81.70	93.65±1.61	85.71 (81.34)	94.50±1.78	14.00	86.71±1.54
	Conv21	43.76	88.71±1.93	56.67	91±1.82	84.23 (51.82)	94.30±1.47	32.53	88.60±1.66
	Conv22	47.06	88.68±2.24	60.58	90.4±1.81	84.72 (63.76)	93.96±1.89	47.98	90.03±1.90
	Conv31	39.97	79.1±2.23	11.24	80.76±2.84	84.54 (17.75)	94.28±1.61	20.25	86.85±2.06
	Conv32	20.82	73.45±2.43	11.22	82.35±2.43	85.99 (27.63)	94.05±1.63	42.78	88.21±2.29
	Conv41	LOW	LOW	LOW	LOW	85.84 (85.60)	93.76±1.49	LOW	LOW
	Conv42	LOW	LOW	LOW	LOW	85.75 (66.77)	93.70±1.70	LOW	LOW
	Conv51	LOW	LOW	LOW	LOW	85.33 (84.69)	93.55±1.69	LOW	LOW
	Conv52	LOW	LOW	LOW	LOW	85.09 (76.40)	93.60±1.79	LOW	LOW

method to ours by reducing each layer by the same scale for both settings - 50%,75% and 90%-50%. Evidently, the error accumulation is somewhat less harsh in comparison to the variance based RR prior to fine-tuning. Nonetheless, following the fine-tuning our method is superior. This can be seen in the results of the layer Conv32, 75% setting. Our method achieved 94.1% on LFW compared with 93%. Fine-tuning is a crucial part of our method since it allows the network to adapt after each prune.

An interesting property that we found is the rapid increase in error, the deeper the network is pruned. This can be explained by the fact that modern architectures, such as in our baseline model, follow the scheme of increasing the number of output channels as the feature maps size decreases. Since our method prunes the number of channels - the amount of information loss in the deeper layers is more prominent resulting in the increased error.

We also note that in the reduce and reuse method, random pruning does almost as well as variance based RR. However, in the inbound method, it performs poorly. The inbound method is inherently more sensitive to pruning, since during the reduce and reuse method, the pruned layer output is reconstructed. The optimization performed during the reuse phase reduces the error - even for the random pruning.

Model acceleration. We report the performance and size of the models produced by this method in Table VII, VIII, and IX. As before, we specify the number of parameters and multiplication required by the model for extracting features from a single face image. Using the reduce and reuse scheme, we were able to speed-up the model by $\times 2.37$ with an LFW accuracy loss of 1.52%.

We note that the theoretical speed-up, $O(\frac{nmk^2}{n'n+n'mk^2})$, is not fully realized and the actual speed-up seen during our evaluation is considerably lower. We attribute this to the fact that the underling mobile implementation of the computation of each layer is already optimized using vectorized operations. The added overhead of adding additional layers adds to the runtime beyond the layer's multiplications. We verified this by comparing various architectures of different depths, in which the total number of multiplications is the same. The shallower the architecture, the shorter the run time using the existing deep frameworks.

TABLE VII

50% REDUCE PERFORMANCE: TIME IS REPORTED IN SECONDS; THE NUMBER OF MULTIPLICATIONS IS REPORTED IN G-OPS. WE ALSO REPORT THE AMOUNT OF PARAMETERS THAT THE MODEL CONTAINS. THE LAYER COLUMN REPORTS THE LAYER THE NETWORK WAS PRUNED UP TO.

Layer	Time	# params	# mults
Conv12	0.434	1745632	1.34
Conv21	0.419	1729280	1.26
Conv22	0.393	1700672	1.12
Conv31	0.387	1650032	1.05
Conv32	0.379	1585616	0.97

TABLE VIII

75% REDUCE PERFORMANCE: TIME IS REPORTED IN SECONDS; THE NUMBER OF MULTIPLICATIONS IS REPORTED IN G-OPS. WE ALSO REPORT THE NUMBER OF PARAMETERS THAT THE MODEL CONTAINS. THE LAYER COLUMN REPORTS THE LAYER THE NETWORK WAS PRUNED UP TO.

Layer	Time	# params	# mults
Conv12	0.413	1739984	1.23
Conv21	0.403	1713376	1.10
Conv22	0.352	1662208	0.84
Conv31	0.330	1581592	0.74
Conv32	0.294	1466440	0.60

TABLE IX
90%-50% REDUCE PERFORMANCE: TIME IS REPORTED IN SECONDS; THE NUMBER OF MULTIPLICATIONS IS REPORTED IN G-OPS. WE ALSO REPORT THE NUMBER OF PARAMETERS THAT THE MODEL CONTAINS. THE LAYER COLUMN REPORTS THE LAYER THE NETWORK WAS PRUNED UP TO.

Layer	Time	# params	# mults
Conv12	0.377	1736807	1.17
Conv21	0.349	1704430	1.00
Conv22	0.292	1639867	0.68
Conv31	0.272	1541765	0.56
Conv32	0.255	1397017	0.38
Conv41	0.245	1294681	0.35
Conv42	0.237	1180121	0.31
Conv51	0.229	1008681	0.30
Conv52	0.216	829641	0.29

TABLE X
HYBRID METHOD. THE RUNNING TIME IS IN SECONDS. WE COUNT THE NUMBER OF MULTIPLICATIONS AND PARAMETERS IN EACH NETWORK. THE AMOUNT OF MULTIPLICATIONS IS GIVEN IN G-OPS.

Method	CASIA	LFW	Time	# params	# mults
Hyb. 50%	84.80	94.08±1.70	0.323	1534928	0.81
Hyb. 75%	85.23	94.05±1.57	0.269	1441096	0.52
Hyb. 90%-50%	85.32	93.35±1.71	0.193	819057	0.26

C. Hybrid pruning

In Table X, we show the results for employing bi-directional pruning of the baseline model. We start with the RR version of the baseline model which achieved satisfying accuracy results, i.e. a LFW score of above 94% and apply inbound prune on that model.

Specifically, we perform inbound prune of the models produced by RR50% and RR75%, pruned up to layer Conv32 using reduce and reuse, and RR90%-50% up to Conv52. In all three cases, we further prune 25% of incoming channels up to Conv32. In Table X we show the accuracy and performance of the bi-directional prune products. By pruning RR90%-50% we were able to speed-up the baseline model by more than $\times 2.65$ with a minor reduction in accuracy.

D. Method comparison

In addition to low rank approximation, we also compare our method to the fitnet method, which unlike the low-rank method, produces a different network architecture. Table XI reports the accuracy and timing of the models that we trained using this fitnet method. Our hybrid method produced faster and more accurate models than the models produced by the fitnet method.

Moreover, the training time required for our hybrid method was less than that of the fitnet method. Although our method is composed of multiple fine-tuning steps, the total amount of epochs required is less than 400 epochs. On the other hand, the fitnet method which is based on two training steps: hint-based and knowledge distillation, required more than 600 epochs. During the knowledge distillation phase, the fitnet model is trained by optimizing two loss function which requires more epochs to converge.

We conclude this section by comparing all of the models generated in our study. In Table XII we report the best model per method. Our hybrid method achieves better accuracy than

TABLE XI
FITNET ACCURACY: THE RUNNING TIME IS IN SECONDS. WE COUNT THE NUMBER OF MULTIPLICATIONS AND PARAMETERS IN EACH NETWORK. THE AMOUNT OF MULTIPLICATIONS IS GIVEN IN G-OPS.

Model	CASIA	LFW	Time	# params	# mults
Fitnet 1	82.08	92.61±1.73	0.207	1167262	0.50
Fitnet 2	75.03	92.28±1.53	0.228	1119011	0.64
Fitnet 3	77.68	93.26±1.11	0.347	1487923	1.21

that of fitnet and is faster. Our method accelerated the baseline model by more than $\times 2.5$ with a drop of 1.77% in accuracy.

VIII. CONCLUSION

We describe two methods for network compression that are specifically suited for CNNs. The methods employ a simple variance-based criterion which is readily computed. Therefore, the pruning process does not require solving an optimization problem.

We propose two methods: in one, we selectively remove input channels from the computation of the channels of the next layer. This makes use of the fact that the computation of each channel is influenced to a different degree from the channels of the previous layers. However, this pattern of influence varies between the computed channels and therefore requires a per-channel selection process. In the second method, we eliminate entire channels making use of the redundancy in the representation. However, in order to not affect subsequent layers, we reconstruct back the removed channels.

While the compression method introduces a few extra parameters such as the number of channels to prune (μ), pruning threshold (τ) and the size of the sample set, the number of additional parameters is small compared to the entire set of parameters, and the system does not seem to be overly sensitive to these parameters. In addition, searching over the space of these parameters, if one chooses to do so, is much faster than optimizing the convnet itself.

We compare our method to recent work in the field of network compression. This is done to the low rank approximation of [18] and to fitnet [14]. We demonstrate convincingly that the suggested pruning methods are more effective in the compression of the scratch face recognition network. Overall, without modifying the software to the structure of the new network, we are able to obtain a run time improvement of up to 2.65 times while suffering a very moderate loss of accuracy. This run time improvement is significant since the overall network structure, the software modules, and the performance remain unchanged.

ACKNOWLEDGMENT

This project was partly funded by the Broadcom Foundation and Tel Aviv University Authentication Initiative. The authors would like to thank Sivan Toledo for fruitful discussions.

REFERENCES

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, Winter 1989.

TABLE XII

METHOD COMPARISON: THE TABLE REPORTS THE BEST MODEL FOR EACH METHOD DISCUSSED: RR, INBOUND PRUNE, HYBRID, FITNET AND LOW-RANK APPROXIMATION. THE LAYER COLUMN SPECIFIES THE LAYER THAT EACH MODEL WAS PRUNED UP TO. TIME IS REPORTED IN SECONDS. SPEEDUP IS REPORTED RELATIVE TO BASELINE MODEL TIME. THE NUMBER OF MULTIPLICATIONS IS REPORTED IN G-OPS. FINALLY, THE PARAMETER REDUCTION DEPICTS THE NUMBER OF PARAMETERS RELATIVE TO BASELINE.

	Layer	CASIA	LFW	Time	Speedup	Mults	Param Reduct.
Baseline	N\A	86.04	95.12	0.512	1.00	1.48	1.00
Fitnet 1	N\A	82.08	92.61	0.207	2.47	0.50	1.50
Fitnet 2	N\A	75.03	92.28	0.228	2.25	0.64	1.57
Fitnet 3	N\A	77.68	93.26	0.348	1.47	1.21	1.18
Low rank-50%	Conv32	84.79	93.90	0.379	1.35	0.97	1.11
Low rank-75%	Conv32	78.81	93.00	0.294	1.74	0.60	1.20
Inbound prune	Conv41	84.74	93.92	0.322	1.59	0.96	1.12
RR50%	Conv32	84.66	94.40	0.379	1.35	0.97	1.11
RR75%	Conv32	85.18	94.12	0.294	1.74	0.60	1.20
RR90%	Conv32	85.94	94.05	0.255	2.01	0.38	1.25
RR90%-50%	Conv52	85.08	93.60	0.216	2.37	0.29	2.11
Hyb.50%	Conv32	84.80	94.08	0.323	1.59	0.81	1.14
Hyb.75%	Conv32	85.23	94.05	0.269	1.90	0.52	1.22
Hyb.90%-50%	Conv52	85.32	93.35	0.193	2.65	0.26	2.14

[2] "Torch7 android port," https://github.com/soumith/torch-android, accessed: 2010-09-30.

[3] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 1701–1708. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2014.220

[4] Y. Sun, X. Wang, and X. Tang, "Deep learning face representation from predicting 10,000 classes," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[5] Y. Sun, Y. Chen, X. Wang, and X. Tang, "Deep learning face representation by joint identification-verification," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 1988–1996. [Online]. Available: http://papers.nips.cc/paper/5416-deep-learning-face-representation-by-joint-identification-verification.pdf

[6] Y. Sun, X. Wang, and X. Tang, "Deeply learned face representations are sparse, selective, and robust," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[7] Y. Sun, D. Liang, X. Wang, and X. Tang, "Deepid3: Face recognition with very deep neural networks," *CoRR*, vol. abs/1502.00873, 2015. [Online]. Available: http://arxiv.org/abs/1502.00873

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations (ICLR)*, 2015.

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR 2015*, 2015.

[10] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," June 2015.

[11] J. Liu, Y. Deng, T. Bai, and C. Huang, "Targeting ultimate accuracy: Face recognition via deep embedding," *CoRR*, vol. abs/1506.07310, 2015. [Online]. Available: http://arxiv.org/abs/1506.07310

[12] D. Yi, Z. Lei, S. Liao, and S. Z. Li, "Learning face representation from scratch," *CoRR*, vol. abs/1411.7923, 2014. [Online]. Available: http://arxiv.org/abs/1411.7923

[13] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2654–2662.

[14] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," in *International Conference on Learning Representations (ICLR)*, 2015.

[15] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *CoRR*, vol. abs/1503.02531, 2015. [Online]. Available: http://arxiv.org/abs/1503.02531

[16] E. L. Denton, W. Zaremba, J. Bruna, Y. Lecun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 1269–1277.

[17] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *British Machine Vision Conference*, 2014.

[18] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *CoRR*, vol. abs/1505.06798, 2015. [Online]. Available: http://arxiv.org/abs/1505.06798

[19] M. Lin, Q. Chen, and S. Yan, "Network in network," in *International Conference on Learning Representations (ICLR)*, 2013.

[20] Y. L. Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*. Morgan Kaufmann, 1990, pp. 598–605.

[21] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems 4*, J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds. San Francisco, CA: Morgan Kaufmann, 1992. [Online]. Available: ftp://ftp.ci.tuwien.ac.at/pub/texmf/bibtex/nips-4.bib

[22] G. Thimm and E. Fiesler, "Evaluating pruning methods," in *National Chiao-Tung University*, 1995, p. 2.

[23] N. Strm, "Sparse connection and pruning in large dynamic artificial neural networks," 1997.

[24] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *CoRR*, vol. abs/1506.02626, 2015. [Online]. Available: http://arxiv.org/abs/1506.02626

[25] V. Lebedev and V. S. Lempitsky, "Fast convnets using group-wise brain damage," *CoRR*, vol. abs/1506.02515, 2015. [Online]. Available: http://arxiv.org/abs/1506.02515

[26] M. Figurnov, D. Vetrov, and P. Kohli, "Perforatedcnns: Acceleration through elimination of redundant convolutions," *arXiv preprint arXiv:1504.08362*, 2015.

[27] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[28] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proc. International Conference on Machine learning (ICML'13)*, 2013.

[29] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, "Efficient object localization using convolutional networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[30] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through ffts," in *International Conference on Learning Representations (ICLR)*, 2014.

[31] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun, "Fast convolutional nets with fbfft: A gpu performance evaluation," *arXiv:1412.7580*, 2014.

[32] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," in *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011.

[33] M. Courbariaux, Y. Bengio, and J.-P. David, "Low precision arithmetic for deep learning," Université de Montréal, Tech. Rep. Arxiv report 1412.7024, 2014. [Online]. Available: http://arxiv.org/abs/1412.7024
[34] G. B. Huang and E. Learned-Miller, "Labeled faces in the wild: Updates and new reporting procedures," UM-CS-2014-003, 2014.
[35] D. Chen, X. Cao, L. Wang, F. Wen, and J. Sun, "Bayesian face revisited: A joint formulation," in *European Conf. Computer Vision*, 2012.
[36] M. Everingham, J. Sivic, and A. Zisserman, ""Hello! My name is... Buffy" – automatic naming of characters in TV video," in *Proceedings of the British Machine Vision Conference*, 2006.
[37] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, 2011.
[38] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *arXiv:1502.01852*, 2015.