ADaPT: Optimizing CNN inference on IoT and mobile devices using approximately separable 1-D kernels

ABSTRACT

Breakthroughs from the field of deep learning are radically changing how sensor data are interpreted to extract important information to help advance healthcare, make our cities smarter, and innovate in smart home technology. Deep convolutional neural networks, which are at the heart of many emerging Internet-of-Things (IoT) applications, achieve remarkable performance in audio and visual recognition tasks, at the expense of high computational complexity in convolutional layers, limiting their deployability. In this paper, we present an easy-to-implement acceleration scheme, named ADaPT, which can be applied to already available pre-trained networks. Our proposed technique exploits redundancy present in the convolutional layers to reduce computation and storage requirements. Additionally, we also decompose each convolution layer into two consecutive one-dimensional stages to make full use of the approximate model. This technique can easily be applied to existing low power processors, GPUs or new accelerators. We evaluated this technique using four diverse and widely used benchmarks, on hardware ranging from embedded CPUs to server GPUs. Our experiments show an average 3-5x speed-up in all deep models and a maximum 8-9x speed-up on many individual convolutional layers. We demonstrate that unlike iterative pruning based methodology, our approximation technique is mathematically well grounded, robust, does not require any time-consuming retraining, and still achieves speed-ups solely from convolutional layers with no loss in baseline accuracy.

CCS CONCEPTS

•Computing methodologies →Machine learning; Neural networks; •Hardware →Sensor applications and deployments;

KEYWORDS

Convolutional Neural Networks, Deep Learning, Internet of Things (IoT), Sensors and Hardware Programming

1 INTRODUCTION

Among many deep learning algorithms convolutional neural networks (CNN) are becoming a mainstream technology for an array of new IoT applications including speech recognition, language translation, image classification and numerous other complex tasks. But, these deep models typically require millions of parameters and billions of operations to produce human level accuracy. The size and complexity requirement complicates deployment of deep networks on low power embedded platforms as they have very limited power budgets. The typical power budget varies a lot among the component of the three-tier IoT architecture. At one end many healthcare and human activity recognition (HAR) sensors can consume no more than 10-100 mW [4]. In the future, many of these sensors will be even more limited as they will become purely energy-neutral, powered only by the harvested energy. The other most common classes of sensors such as cameras, microphones, smart-watches typically consume around 1 W [2]. The M2M (Machine-to-Machine) IoT gateways have a maximum power budget in the range of 1-5 W [7]. On the other extreme many industry scale M2M gateways (e.g. gateways used in smart city, surveying and mapping, inventory management) may have little more power budget in the range of 5-20 W [3]. In contrast, to run a 1 billion connection neural network at 100Hz would require 64Watt just to access the data from the main memory [19]. It is obvious that without any significant optimization deploying these type of massive deep networks on IoT and mobile devices is impossible. The current state-of-the-art embedded solutions enable this type of application by off-loading the computation to a cloud based infrastructure where server-grade machines (GPUs and other manycore processors) perform the heavy number crunching.

This approach has severe limitations on the usability and scalability of deep learning based mobile and IoT applications. First and foremost, the user data is sent across the cloud which has serious privacy implications. Second, sending lots of data (e.g. every frame of a video) over a wireless network consumes significant power due to the communication overhead. For applications where continuous data exchange is required between the server and the mobile device, latency is also a big concern. For example, an wearable continuous glucose level monitoring sensor must detect an abnormal condition and must take an action in real time. The third limitation is the scalability, which has a mid to long term implication. Gartner estimates by 2020 26-billion IoT units will be installed globally [1]. The staggering amount of data generated by IoT devices will easily exceed the storage limit of cloud infrastructure. To truly scale deep learning based application globally in various scenarios, we have to enable these applications without the requirement of always having to connect to the cloud infrastructure.

Research has shown that a significant redundancy exists in the parameterization of most modern deep networks [13]. For example, a ConvNet often learns symmetry equivariance properties from the data augmentation used during the training process [15]. To speed up the inference and reduce test time power consumption, this redundancy property can be exploited to approximate the model with negligible impact on accuracy. Recently, there have been several research projects conducted to strategically eliminate redundant neurons and intra-neuron connections from a model. Pruning unwanted neurons is one among many such techniques. In many early works in training ConvNets, statistical pruning based technique proved to be effective in reducing over-fitting ([29],[20]). More recent works extend this idea to reduce number of nodes by iterative pruning and re-training ([19],[33]). The biggest limitation of this approach is that other than heuristic approaches, there is no concrete numerical way to control this process of pruning. One often arrives at a better solution by iterative trial and error. The

IML, 2017
2017. X…$0.00
DOI: X/X

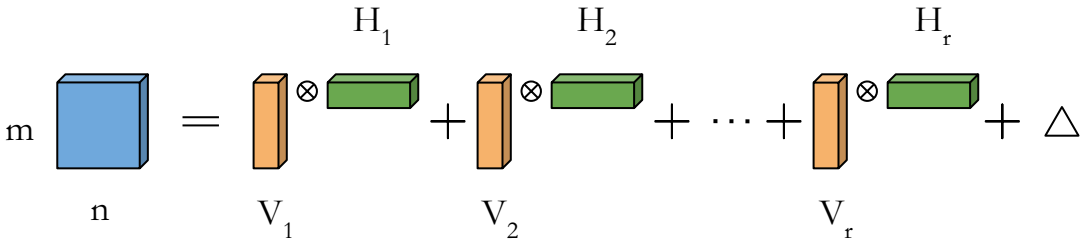

Figure 1: A 2-D matrix can be represented by the sum of r rank-1 updates.

second less popular approach is to use approximation of the filters to achieve reasonable accuracy ([14],[23],[26],[36],[37]). In this method, original filters are reconstructed back from a new basis of representation. The new basis function is learnt simply by minimising the the error (e.g. L2 regularization), whilst penalizing the rank of filters. Our methods follows from this foundation and extends the idea to a numerically tractable metrics driven technique, which helps to determine the approximation statically on any given pre-trained model.

Contribution: Our paper makes the following main contributions.

- We propose an easy-to-implement scheme named ADaPT (Approximate-DecomPose-Tune) which can be applied on pre-trained CNNs to reduce the compute complexity of the convolutional layers. ADaPT consists of three separate stages - (1) layerwise rank selection using singular value decomposition, (2) decomposition of convolutional layers into unitary basis (rank-1) layers to enable 1-D convolution, and (3) fine-tuning only if required.
- Since rank selection and decomposition are only dependent on individual layer's inherent property, the first two stages of the ADaPT scheme can be computed in parallel for all the convolution layers within a model. Unlike pruning or regularization techniques, this is a big advantage as each convolution layer can be approximated in isolation.
- Unlike pruning or regularization schemes, our proposed scheme also does not require any time-consuming iterative re-training phase from scratch. (For example, it took seven days to re-train the pruned five layer (conv) AlexNet [19]. Larger networks with more number of layers will even take longer to re-train after pruning.) Our ADaPT scheme can be applied to targeted convolution layer without touching the other layers to suit the compute and memory budget of target platforms.
- We evaluated our methodology on a number of widely used real-world CNNs targeting different applications and datasets (MNIST, CIFAR-10, IMAGENET, and PASCAL-VOC). We also deployed these models on a number of real hardware (ARM Cortex-A15, nVidia Tegra-K1, Intel Core i7-5930k, and nVidia's Titan-X) for performance measurement. Our result shows an average 3-5x speed-up in all deep models and a peak 8-9x speed-up on many individual convolutional layers. Additionally, our method helps to reduce the memory footprint of the convolutional layers by an average 6-7x. Both savings in computation and memory are possible solely by approximating the convolutional layers with no loss in baseline accuracy.

This paper is organized as follows. Section 2 explains the proposed concept and the related background. Section 3 is dedicated to experiments and results. Section 4 reviews related work and compares with ours. Section 6 summarizes the paper and suggests a future direction of this research.

2 PROPOSED APPROACH: CONCEPT

2.1 Separable Filters

The concept of separable filters by splitting convolution operations into convergent sums of matrix-valued stages was proposed by Hummel and Lowe in 1980s even before ConvNet became popular for automatic feature learning [22]. This property was exploited in many early image processing filters - e.g. Sobel edge detection filter, Gaussian Blurring filter etc. This approach is very powerful but restricted to filters that are decomposable which is often not the case for a trained filter such as in ConvNet. But, due to the presence of inherent redundancy between different filters or feature maps within a layer, this property can be exploited in acceleration of ConvNet models.

Consider an arbitrary kernel of a ConvNet described by the $(m \times n)$ matrix $\mathcal{W}$.

$$\mathcal{W} = \begin{bmatrix} \alpha_{00} & \alpha_{01} & .. & \alpha_{0n} \\ \alpha_{10} & \alpha_{11} & .. & \alpha_{1n} \\ .. & .. & .. & .. \\ \alpha_{m0} & \alpha_{m1} & .. & \alpha_{mn} \end{bmatrix} \tag{1}$$

We say that kernel $\mathcal{W}$ is separable when it can be split into the product of an m-length column vector v and an n-length row vector h as follows

$$\mathcal{W} = \mathcal{V}\mathcal{H}^{\mathcal{T}} = \begin{bmatrix} v_0 \\ v_1 \\ .. \\ v_m \end{bmatrix} \begin{bmatrix} h_0 & h_1 & .. & h_n \end{bmatrix} \tag{2}$$

or, using outer product update $\mathcal{W}$ can be expressed as

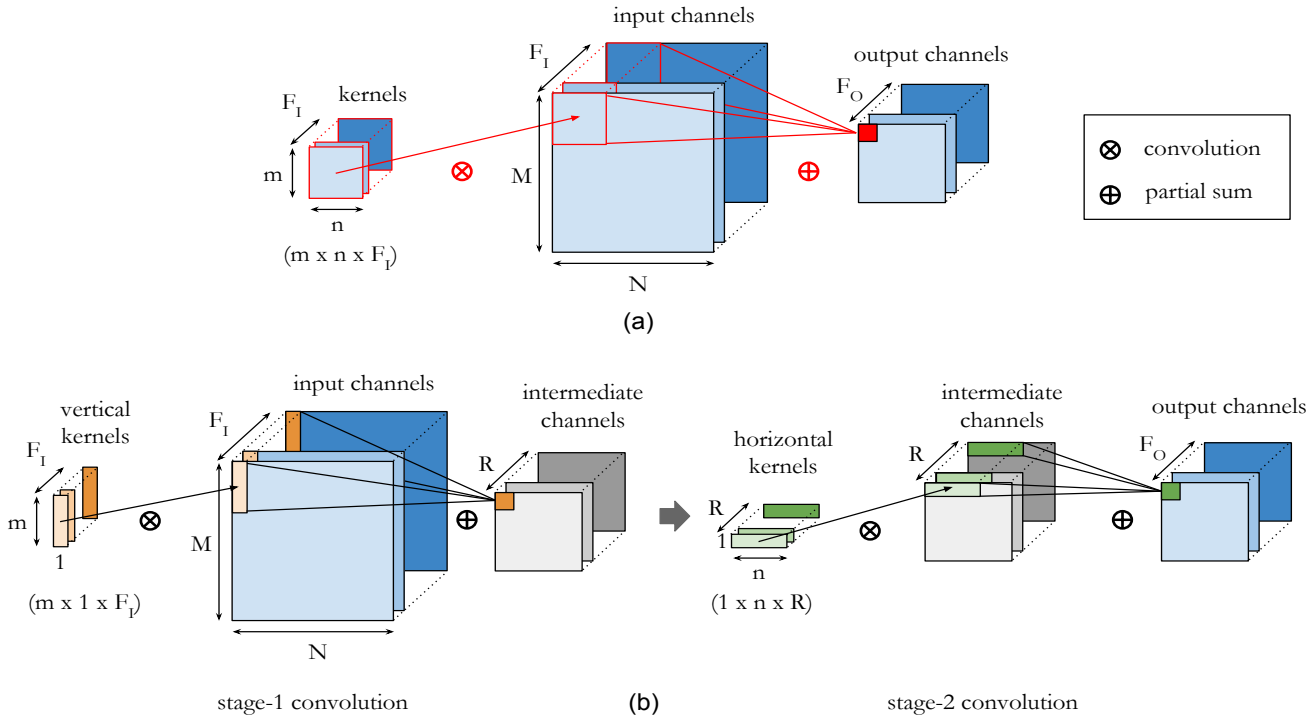

Figure 2: (a) The original convolution with a $(m \times n)$ kernel. (b) The two-stage approximate convolution using a $(m \times 1)$ column kernel followed by a $(1 \times n)$ row kernel. There are R channels in the intermediate virtual layer.

$$\mathcal{W} = \mathcal{V}\mathcal{H}^{\mathcal{T}} = \begin{bmatrix} v_0h_0 & v_0h_1 & .. & v_0h_n \\ v_1h_0 & v_1h_1 & .. & v_1h_n \\ .. & .. & .. & .. \\ v_mh_0 & v_mh_1 & .. & v_mh_n \end{bmatrix} \tag{3}$$

From the Equations (1) and (3), it is apparent that a separable kernel has equivalent rows and columns. To store the original kernel $\mathcal{W}$ in Equation (1), it would require (mn) spaces. But, if the kernel $\mathcal{W}$ is separable matrix, then we see from Equation (3), it would require $(m + n)$ spaces. As m and n becomes large and original kernel is separable $\mathcal{W}$, one can see that substantial savings in compute time and storage will be achieved.

Unfortunately, we cannot generally expect that any trained kernel in ConvNet satisfies such stringent conditions. The collection of kernels in a ConvNet are generally full rank and expensive to convolve with large images. However, we can aim for $\mathcal{W}$ to be approximately separable such that

$$\mathcal{W} = \mathcal{V}\mathcal{H}^{\mathcal{T}} + \mathcal{E} \tag{4}$$

where $\mathcal{E}$ is an error kernel, whose importance we would like to be as small as possible in relation to the original kernel $\mathcal{W}$. We can further generalize the Equation (4) in the following form

$$\begin{aligned} \mathcal{W} &= \mathcal{V}_1\mathcal{H}_1^T + \mathcal{V}_2\mathcal{H}_2^T + ... + \mathcal{V}_i\mathcal{H}_i^T + ... + \mathcal{V}_r\mathcal{H}_r^T + \mathcal{E}_r \\ &= \mathcal{U}_1 + \mathcal{U}_2 + ... + \mathcal{U}_i + ... + \mathcal{U}_r + \mathcal{E}_r \end{aligned} \tag{5}$$

where each term,

$$\mathcal{U}_i = \mathcal{V}_i\mathcal{H}_i^T \tag{6}$$

is an exactly separable rank-1 outer product of a column vector of length m and row vector of length n and $\mathcal{E}_r$ is the error matrix associated with r-term approximation of original kernel $\mathcal{W}$ as shown in Fig 1. Furthermore, if the original kernel $\mathcal{W}$ can be well approximated by r rank-1 updates, we will only require $r(m + n)$ parameters to describe the kernel instead of original mn elements. If we choose r such that $r(m + n) << mn$, then it would require less storage and computations.

2.2 Approximation of Convolutional Layers

In ConvNet, a significant redundancy exists between spatial filter dimensions and also along cross-channel feature maps. Most of the previous research has focussed on exploiting approximation along spatial filter dimensions. In our approach, we aim at approximating the redundancy across different feature maps within a layer. Since we decide the magnitude of the approximation statically, we also do not have any new loss function involved.

Let us assume, in a convolutional neural network, a 4-dimensional kernel can be represented as $\mathcal{W} \in \mathbb{R}^{F_I \times m \times n \times F_O}$, where spatial 2-dimensional kernels are of size $(m \times n)$ and F_I, F_O are the input and output channels within a layer. We can also represent an input feature map as $\mathcal{X} \in \mathbb{R}^{M \times N \times F_I}$ and corresponding kernels as

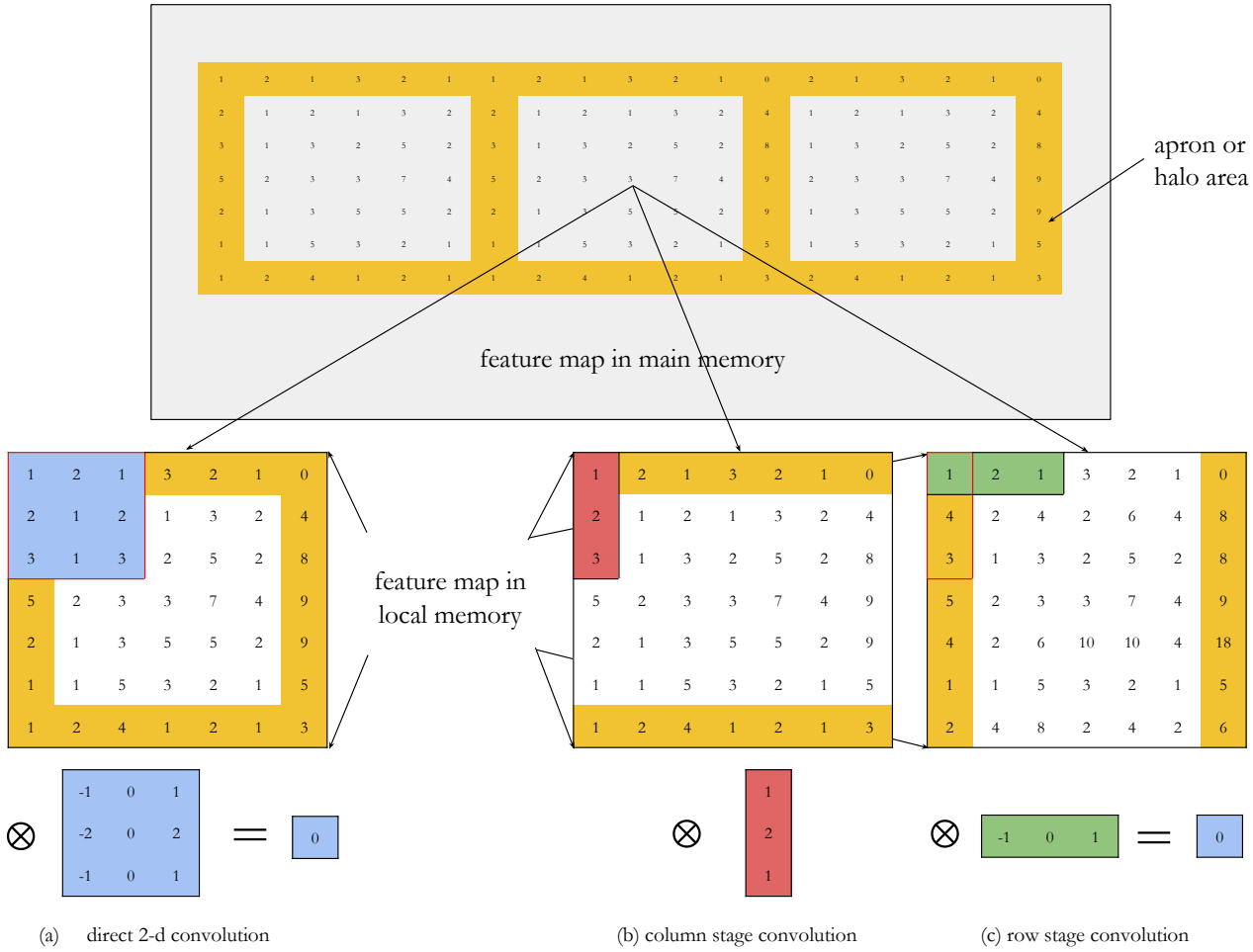

Figure 3: (a) The original convolution with a $(m \times n)$ 2-D kernel. The yellow marked apron is all around the input tile. (b) The column-stage convolution using a $(m \times 1)$ column 1-D kernel. The apron is only at the top and bottom. (c) The row-stage convolution using a $(1 \times n)$ row 1-D kernel. The apron is only at the left and right.

$\mathcal{W}_i \in \mathbb{R}^{m \times n \times F_I}$ for i-th set of weights, where each input feature map is of size $(M \times N)$. The original convolution for the i-th set of weights in a given layer now becomes

$$\mathcal{W}_i * \mathcal{X} = \sum_{f=1}^{F_I} \mathcal{W}_i^f * x^f \tag{7}$$

Our goal is to find an approximation of kernel $\mathcal{W}_i$ such that

$$\mathcal{W}_i = \widetilde{\mathcal{W}_i} + \mathcal{E} \tag{8}$$

Using the approximation technique from the last section, let us assume for a small error $\mathcal{E}$, the chosen rank is R. We will come back to the aspect of how we choose an appropriate value of R shortly. The modified kernel now can be represented by the Equation (9), where $\mathcal{V} \in \mathbb{R}^{R \times m \times 1 \times F_I}$ is the approximate column kernel, and $\mathcal{H} \in \mathbb{R}^{F_O \times 1 \times n \times R}$ is the approximate row kernel.

$$\begin{aligned} \mathcal{W}_i * \mathcal{X} &= \sum_{r=1}^{R} \mathcal{H}_i^r (\mathcal{V}^r)^T + \hat{\mathcal{E}} \\ &= \sum_{r=1}^{R} h_i^r * \mathcal{V}^r + \hat{\mathcal{E}} \\ &= \sum_{r=1}^{R} h_i^r * (v_r * x) + \hat{\mathcal{E}} \\ &= \sum_{r=1}^{R} h_i^r * (\sum_{f=1}^{F_I} v_r^f * x^f) + \hat{\hat{\mathcal{E}}} \end{aligned} \tag{9}$$

where $\hat{\hat{\mathcal{E}}}$ is the total error in a channel after convolution introduced by the approximation process. Fig 2 depicts the idea of

Layer	Original ($F_I \times m \times n \times F_O$)	Original Parameter Count	Compressed Column ($F_I \times m \times n \times V_R$)	Compressed Row ($V_R \times m \times n \times F_O$)	Reduction in Layer Size
Conv-1	$1 \times 5 \times 5 \times 32$	1K	$1 \times 5 \times 1 \times 4$	$4 \times 1 \times 5 \times 32$	1.2x
Conv-2	$32 \times 5 \times 5 \times 64$	51K	$32 \times 5 \times 1 \times 16$	$16 \times 1 \times 5 \times 64$	6.6x

Table 1: MNIST model approximation summary

re-constructing the convolution layer using a number of separable lower order filters and compares with the direct convolution.

2.3 Reduction of Compute and Storage Cost

The computational cost of the original direct convolution is of order $O(F_I MNmnF_O)$. But, using the approximation technique as above, the computational cost for first stage convolution is $O(F_I MNmR)$ and for second stage convolution is $O(RMNnF_O)$, resulting a total computational cost of $O((mF_I + nF_O)MNR)$. If we choose R such that $R(mF_I + nF_O) << mn(F_I F_O)$, then computational cost can be reduced. In practice convolutional neural network uses square kernels. Hence, let us assume $m = n = p$, which is the size of the kernel. Using this assumption the condition can be simplified to $R(F_I + F_O) << pF_I F_O$. In addition, most modern ConvNets use more channels in the higher layers than the corresponding lower layers, i.e. the channel ratio $\frac{F_O}{F_I} >> 1$. The higher the ratio the larger the value of R can be. In most layers the computation cost can be reduced by p, which is the dimension of the kernel in the respective layer.

Similarly, the initial cost of storage is $F_I F_O p^2$, whereas cost is $(F_I pR + RpF_O)$ after approximation and separating the kernels into two rectangular ones. If we choose $R << p\frac{F_I F_O}{(F_I + F_O)}$, significant savings can be made for the storage costs of the kernels.

2.4 Efficient Use of Memory Bandwidth

Fetching data from off-chip main memory (DRAM) costs order of magnitude more than from on-chip or local storage ([11], [21]). Chen *et al.* in their Eyeriss research project showed that a row stationary 1-D convolution is optimal solution for throughput and energy efficiency than traditional 2-D convolution [8]. Separable filters enable row stationary 1-D convolutions by reducing the number of unnecessary data loads in padded convolution by dividing the convolution in two 1-D stages. To preserve information many convolutional networks use zero-padding in many layers. Around the image tile, there is an *apron* of pixels that is required in order to filter the image tile (see figure 3). Note that the *apron* of one block also overlaps with the adjacent blocks. If we separate the convolution into row and column passes, it is no longer necessary to load the top and bottom *apron* regions for the row stage of computation. Similarly, for the column stage, left and right *apron* regions are no longer necessary to load. This allows more efficient use of the available memory bandwidth and on-chip storage. In case of strided convolution, this approach works very well.

2.5 Finding the Optimum Basis Function

Existing solutions to approximate the layers in a deep network require the model to be re-trained with an updated cost function. Since, re-training is a resource intensive and time consuming process, we propose a static re-structuring of the layers by minimizing the $\mathcal{E}$ in Equation (8). Since each layer can be analysed separately, all of them can be approximated in parallel by applying the same criteria as explained earlier.

We adopt the singular value decomposition technique to find an appropriate rank(R) for each layer in parallel. Algorithm 1 provides the detailed steps used for layerwise approximation. One way to think about this approach is as approximately representing the original kernels as the sum of many rank-1 basis kernels. The advantage of this approach is that once new rank-1 basis are found each of these kernels now can be decomposed into two stages - a column-wise convolution and a row-wise convolution. Singular value decomposition provides a mathematically grounded mechanism to choose the most R important rank-1 basis. To deploy an appropriate model, designers can pre-compute many approximate models for given set of accuracy and performance. Based on application sensitivity and target hardware, the best model can be loaded at run time. The proposed approach is fast and accuracy can be traded off with target performance or storage requirements. For aggressive compression, the accuracy may drop slightly just after reconstructing the model. But, a quick fine-tuning can help to restore the accuracy to the base-line. Unlike many other proposed iterative schemes which can get stuck on local minima, our approach directly provides the global minimum, which is the best approximation.

Algorithm 1: Layerwise Approximation Algorithm

```
function LayerwiseReduce (M, C, W);
  Input  : Target ConvNet model: M, Compression factor of
           each layer: [c_1, c_2, .., c_n], Pre-trained weights of
           individual layer:[w_1, w_2, .., w_n]
  Output : Reduced ConvNet Model: M*, Reduced weights of
           each layer: [v_1, v_2, .., v_n], [h_1, h_2, .., h_n]
for i ← 1 to Layers do
    if layerType == Conv then
        targetRank ← p_i F_I F_O / (c_i (F_I + F_O));
        U Λ V^T ← SVD(w_i);
        disconnectLayers(w_i);
        v_i ← U √Λ;
        h_i ← V √Λ;
        addNewLayer(targetRank);
        M* ← reconstructModel(M, v_i, h_i);
    end
end
```

Layer	Original ($F_I \times m \times n \times F_O$)	Original Parameter Count	Compressed Column ($F_I \times m \times n \times V_R$)	Compressed Row ($V_R \times m \times n \times F_O$)	Reduction in Layer Size
Conv-1	$3 \times 5 \times 5 \times 32$	2K	$3 \times 5 \times 1 \times 4$	$4 \times 1 \times 5 \times 32$	3.4x
Conv-2	$32 \times 5 \times 5 \times 64$	51K	$32 \times 5 \times 1 \times 16$	$16 \times 1 \times 5 \times 64$	6.6x
Conv-3	$64 \times 5 \times 5 \times 128$	205K	$64 \times 5 \times 1 \times 32$	$32 \times 1 \times 5 \times 128$	6.6x

Table 2: CIFAR-10 model approximation summary

3 EXPERIMENTS

To evaluate our proposed technique we have deployed both the original and corresponding approximate models to four different target hardware, namely, ARM Cortex-A15, nVidia Tegra-K1 (Kepler architecture), Intel Core i7-5930k and nVidia's Titan-X (Maxwell architecture). To demonstrate the applicability of our proposed approach in wider variety of platforms, we have chosen both embedded grade and desktop grade hardware. The Cortex-A15 was used as single threaded target running at up to 2.3GHz. The Tegra K1 with 192 CUDA cores were run at 852MHz. Intel Core i7-5930k was also used single threaded target running at up to 3.5GHz. The Maxwell generation Titan-X with 3072 CUDA cores were run at 1.08GHz. On top of standard CUDA 6.5, TK1 was used with cuDNN v2 (max supported version) and Titan X was used with cuDNN v4. For evaluation in the CPUs, we have used the OpenBLAS 0.2.18 optimized BLAS library.

We have also selected four different models applicable to four widely used datasets in machine learning community, namely, MNIST, CIFAR-10, ILSVRC and PASCAL VOC. The following section provides an overview each of the dataset and associated models used for the experiments.

3.1 MNIST - Digit Recognition

The MNIST dataset consists of handwritten digit images and it is divided in 60,000 examples for the training set and 10,000 examples for testing. In our experiments the official training set is again divided into an actual training set of 50,000 examples and 10,000 validation examples.

Layer	Original (ms)	Compressed (ms)	Speed-up
Conv-1	2.6	1.2	2.1x
Conv-2	8.3	1.6	5.7x
Total	15.7	5.6	2.8x

Table 3: MNIST speed-up of convolution layers on ARM Cortex A15 without no loss of base-line accuracy of 99.23%

All digit images have been size-normalized and centred to a fixed size image of 28x28 pixels. We used two layer convolutional network followed by a dense layer similar to the LeNet architecture which had a baseline accuracy of 99.23% [30]. Using our technique we re-constructed an approximation of the original model. A summary of the original and the newly constructed model is presented in Table 1. We then deployed the model in all the four different target platforms. The approximate model achieve a maximum speed up of 2.8x without drop of any accuracy. A comparison of speed up between original and corresponding approximate model for all four target platforms is shown in Table 7. We observed that for the MNIST model, the speed up in the GPUs are less than that of CPUs. We believe this may be due to some of the CUDA cores being under-utilized. A layerwise breakdown of speed up is also shown in Table 3. The model can be approximated further by trading off negligible accuracy by choosing lower rank in each layer during approximation.

3.2 CIFAR-10 - Object Classification

The CIFAR-10 classification is a common benchmark problem in machine learning. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The model in our experiment is an all convolutional network adapted from Tensorflow's CIFAR-10 example, consisting of alternating convolutions and non-linearities.

Layer	Original (ms)	Compressed (ms)	Speed-up
Conv-1	2.4	1.1	2.2x
Conv-2	8.8	1.5	5.9x
Conv-3	9.1	1.5	6.1x
Total	25.6	8.8	2.9x

Table 4: CIFAR-10 layerwise speed-up of convolutions on ARM Cortex A15 without any loss of base-line accuracy of 86%

Using SGD based training our model achieves a peak performance of about 86% accuracy within a few hours of training time on the GPU. We then took this model through our three stage approximation technique to speed-up the convolutional layers. A summary of the decomposition is shown in Table 2. The approximate model achieved a maximum speed-up of 2.9x without any loss of base-line accuracy. Table 4 presents the layerwise speed up of the model. Both the original and the approximate model were also deployed in all the four target platforms. The results obtained from these evaluation are summarised in Table 7. It is worth to notice that some layers achieve a speed-up up to 6x, which is very impressive for convolutional layers. The model can further be approximated by trading off accuracy as shown in Fig 4.

3.3 ILSVRC-2012 - Image Classification

Since 2010, the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is a competition where research teams from

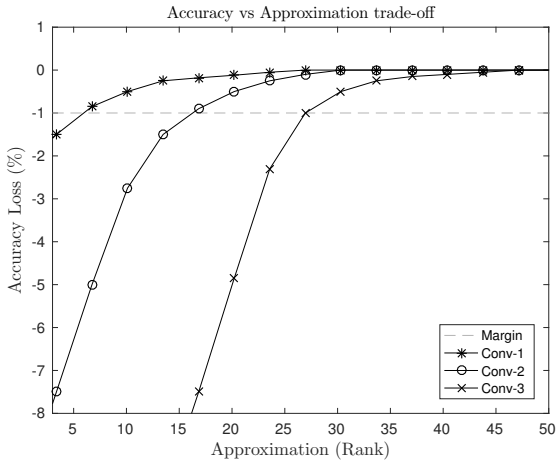

Figure 4: Layerwise accuracy-loss vs rank-approximation for CIFAR-10 Model

academia and industry submit models that classify and detect objects and scenes. For our evaluation we have focussed on the image classification task of the challenge. The classification task on ILSVRC challenge is made up of 1.2 million images in the training set, each labelled with one of 1000 categories that cover a wide variety of objects, animals, scenes, and even some abstract geometric concepts such as "hook", "spiral" etc. The 100,000 test set images are released with the dataset, but the labels are withheld to prevent teams from over-fitting the test set. The teams have to predict 5 (out of 1000) classes and image is considered correct is at least one of the predictions is the ground truth.

Layer	Original (ms)	Compressed (ms)	Speed-up
conv3-64-1.1	28.5	15.0	1.9x
conv3-64-1.2	168.1	32.3	5.2x
conv3-128-2.1	74.1	25.6	2.9x
conv3-128-2.2	147.3	42.1	3.5x
conv3-256-3.1	67.3	15.7	4.3x
conv3-256-3.2	134.2	25.8	5.2x
conv3-256-3.3	134.5	27.4	4.9x
conv3-512-4.1	65.2	12.8	5.1x
conv3-512-4.2	129.9	22.0	5.9x
conv3-512-4.3	130.1	21.3	6.1x
conv3-512-5.1	33.4	4.3	7.8x
conv3-512-5.2	33.5	4.2	7.9x
conv3-512-5.3	33.4	4.2	7.9x
Total	1432.3	252.8	5.7x

Table 5: VGG16 layerwise speed-up of convolution on i7-5930k (per image) with no loss of base-line accuracy of 90.5%

For our experiment, we have considered a widely used network VGG16, which won the competition in ILSVRC-2014 challenge. VGG16 model is a very deep architecture and consists of 13 convolutional layers out of a total of 16 layers. Researchers and engineers often apply transfer learning to modify the weights in the last few dense layer to re-target the model for a different class of problem. The first 13 convolution layers are very performance critical as they stay unchanged after the modification. We have re-used the weights from the pre-trained model which we obtained from the webpage of the authors [35]. This model achieves a top-5 accuracy of 90.6%. We then applied our proposed technique to approximate and decompose each convolution layer. Table 6 provides the layerwise structure of the approximate model and compares with the original model. Both models were then deployed to our target platform for evaluation. The new model achieves an impressive 5.7x maximum speed-up without loss of base-line accuracy. It is worth mentioning that this speed-up comes solely from the approximation of the convolutional layer. A detailed layerwise speed-up is presented in Table 5 for a single threaded implementation in Intel platform. The model can further be approximated by trading off accuracy as shown in Fig 5. A further comparison between original and approximate model targeting different hardware is presented in the Table 7.

3.4 PASCAL VOC-2011 - Image Segmentation

For our final experiment, we considered a fully convolutional network based on an image segmentation model. In computer vision, image segmentation is the process of partitioning a digital image into multiple segments. We believe that the future of surveillance, defence, security, deliveries and many other commercial sectors will be shaped by drone technology. In air drone technology image segmentation is a very fundamental and challenging task. To achieve this researchers adapt pre-trained state-of-the-art deep convolutional neural network and transfer their learned representations to the segmentation task by fine-tuning.

The PASCAL Visual Object Classes (VOC) challenge is a benchmark in visual object category recognition, detection and segmentation, providing the vision research communities with a standard dataset of images and annotations. The VOC dataset consists of annotated consumer photographs collected from the flickr photo-sharing web-site. For our experiment, we considered segmentation equipped FCN-VGG16 model [32]. This fine-tuned model achieves a state-of-the-art 56.0% mean IU score on validation set. We applied our three stage approximation method to get rid of redundancy in the model. The newly constructed approximate model achieves a 2.8x speed-up without any loss in mean IU score. The layerwise approximation almost follows the VGG16 model which is shown in Table 6. We also deployed the approximate model on four different hardware platforms to compare its performance with the original model. A summary of the speed-up comparison is shown in Table 7.

4 RELATED WORKS

Modern Convolutional neural network usually consists of series of convolution layers followed by a number of fully connected layers stacked together. Convolutional layers are computation heavy and fully connected layers are dominated by a large number of weights. For example, in the VGG-16 model, 90% of the computation is dedicated to the convolutional layers, whereas 90% of the weights belong to the fully-connected layers [35]. As a result, running any latest state-of-the-art ConvNet on a mobile system is a

Layer	Original ($F_I \times m \times n \times F_O$)	Original Parameter Count	Compressed Column ($F_I \times m \times n \times V_R$)	Compressed Row ($V_R \times m \times n \times F_O$)	Reduction in Layer Size
conv3x3-64-1.1	$3 \times 3 \times 3 \times 64$	2K	$3 \times 3 \times 1 \times 4$	$4 \times 1 \times 3 \times 64$	2.1x
conv3x3-64-1.2	$64 \times 3 \times 3 \times 64$	37K	$64 \times 3 \times 1 \times 12$	$12 \times 1 \times 3 \times 64$	8.0x
conv3x3-128-2.1	$64 \times 3 \times 3 \times 128$	74K	$64 \times 3 \times 1 \times 40$	$40 \times 1 \times 3 \times 128$	3.2x
conv3x3-128-2.2	$128 \times 3 \times 3 \times 128$	148K	$128 \times 3 \times 1 \times 40$	$40 \times 1 \times 3 \times 128$	4.8x
conv3x3-256-3.1	$128 \times 3 \times 3 \times 256$	295K	$128 \times 3 \times 1 \times 50$	$50 \times 1 \times 3 \times 256$	5.1x
conv3x3-256-3.2	$256 \times 3 \times 3 \times 256$	590K	$256 \times 3 \times 1 \times 60$	$60 \times 1 \times 3 \times 256$	6.4x
conv3x3-256-3.3	$256 \times 3 \times 3 \times 256$	590K	$256 \times 3 \times 1 \times 70$	$70 \times 1 \times 3 \times 256$	5.5x
conv3x3-512-4.1	$512 \times 3 \times 3 \times 512$	1M	$512 \times 3 \times 1 \times 80$	$80 \times 1 \times 3 \times 512$	6.4x
conv3x3-512-4.2	$512 \times 3 \times 3 \times 512$	2M	$512 \times 3 \times 1 \times 100$	$100 \times 1 \times 3 \times 512$	7.7x
conv3x3-512-4.3	$512 \times 3 \times 3 \times 512$	2M	$512 \times 3 \times 1 \times 110$	$110 \times 1 \times 3 \times 512$	7.0x
conv3x3-512-5.1	$512 \times 3 \times 3 \times 512$	2M	$512 \times 3 \times 1 \times 80$	$80 \times 1 \times 3 \times 512$	9.6x
conv3x3-512-5.2	$512 \times 3 \times 3 \times 512$	2M	$512 \times 3 \times 1 \times 78$	$78 \times 1 \times 3 \times 512$	9.8x
conv3x3-512-5.3	$512 \times 3 \times 3 \times 512$	2M	$512 \times 3 \times 1 \times 78$	$78 \times 1 \times 3 \times 512$	9.8x

Table 6: VGG16 model approximation summary

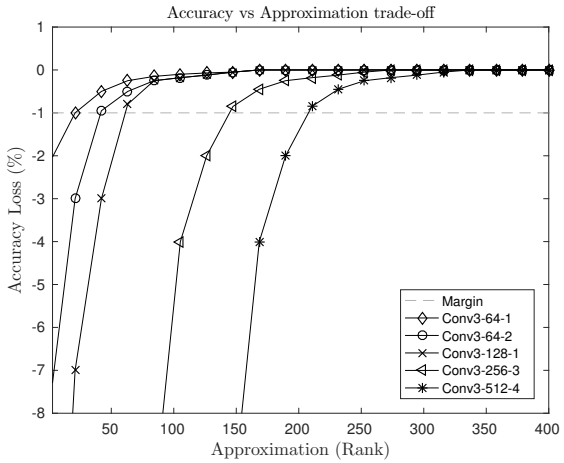

Figure 5: Accuracy-loss vs rank-approximation of selected layers from VGG16 Model

really challenging task. A large network requires a lot of computation to do the dot products dedicated to convolutional layers. A large memory bandwidth is required to support not only the fully connected layers, but also to move the data generated from the intermediate activations in the convolutional layers. This results in significant energy consumption which is beyond the power and compute budget of a mobile system. In the recent years, there have been quite a few research projects that focussed on reducing overall size of the network.

Han *et al.* proposed a pruning approach in their Deep Compression paper which aims at reducing the total number of weights in the entire network [19]. Pruning based approaches can provide significant reductions in the network size. However, pruning mainly benefits the fully connected layer to reduce the size of the network. The authors in this paper mentioned that it is challenging to achieve runtime speed-up of convolutional network with conventional implementation on hardware. Unless a number of operations are reduced in the convolutional layers, pruning alone in the fully connected layer wouldn't help enough to fit this model for battery power mobile devices. Later several researchers extended the idea of pruning to achieve model size reduction ([33],[5],[6]). All of these research works have re-confirmed the benefit from pruning the fully-connected layers of a model, where they show it is possible to reduce the size of those layers up to 12x. But, compared to fully-connected layers, the convolutional layers are very sensitive to pruning and results in significant loss in accuracy [19]. Pruning is not an effective method to reducing overall computation load dominated by convolution. The most common solution to this limitation is to adopt a very time-consuming iterative prune and re-tune process.

Gupta *et al.* [16] studied the effect of limited precision data representation in the context of training a Convolutional Neural Network. They observed that a ConvNet can be trained using only 16-bit wide fixed-point number representation with little to no degradation in the classification accuracy. Recently, Han *et al.* extended their work on Deep Compression [18] by combining pruning with trained quantization. They used 8-bits for each convolution layer and 5-bits for each fully-connected layer in AlexNet and achieved no accuracy loss. Gysel *et al.* in his master's thesis developed Ristretto [17], a hardware oriented approximation technique, where he condensed SqueezeNet to 8-bit with minimal loss of classification accuracy. Last year, Courbariaux *et al.* introduced an extreme version of quantization using binary neural networks. In their work on Binarized Neural Networks [10], they have constrained the weights and activations to +1 or -1. Although binary networks are an extreme case of quantization, the research has shown quantization can reduce both computation and storage requirements of modern ConvNets. In fact, our method can be combined with quantization technique to further reduce the size of a deep model.

Liu *et al.* [31] proposed Sparse Convolutional Neural Networks (SCNN) model that exploits both inter-channel and intra-channel redundancy to maximize sparsity in a model. This procedure zeros out more than 90% of parameters, with a drop of accuracy less than 1% on the ImageNet dataset. The authors also proposed an efficient

Model	Reduction in Model Size	Speed-up on Intel i7-5930k	Speed-up on Titan-X	Speed-up on Cortex-A15	Speed-up on Tegra-K1
MNIST Original	1x	1x	1x	1x	1x
MNIST Approx.	6.1x	2.30x	1.50x	2.79x	1.40x
CIFAR10 Original	1x	1x	1x	1x	1x
CIFAR10 Approx.	6.6x	2.34x	2.03x	2.90x	2.01x
VGG16 Original	1x	1x	1x	1x	1x
VGG16 Approx.	6.6x	5.70x	4.23x	4.80x	4.09x
FCN-VGG16 Original	1x	1x	1x	1x	1x
FCN-VGG16 Approx.	6.7x	5.91x	4.39x	4.94x	4.37x

Table 7: Speed-up of selected ConvNets on different target platforms solely by approximating the convolutional layers without any loss of baseline accuracy

sparse matrix multiplication algorithm on CPU to further accelerate the optimized model. Although it seems to be a very promising approach, this work has two major limitations compared to our technique. As the result section of the paper shows that the much of the success of this approach comes from the fully-connected layers. But, the compute heavy convolutional layers are not benefited by this technique. The author also used a modified cost function to re-train the network which is again a time-consuming resource hungry process.

Accelerating convolution using separable filters has been used even before convolutional neural network became main stream solution for image classification and object recognition. After more than a decade in which separable filters have been neglected, there is evidence of renewed interests in the area of ConvNet compression. Denton et al. showed in a recent research that the singular value decomposition based truncation can help to reduce parameters from the fully-connected layers [14]. But, the authors haven't given much attention to convolutional layers, which are compute heavy. Jaderberg et al proposed a singular value decomposition based technique for layer-by-layer approximation [23]. But their technique is iterative where a layer can only be approximated after the previous layer has been compressed. The author used an updated loss function to learn the low-rank filters which is again a time consuming process. The author also reported that simultaneous approximation of all the layers in parallel is not efficient. In contrast, we show that in our technique, using a target metric all the layers can be approximated statically.

Our proposed static compression technique not only saves lot of time which is critical for development of mobile applications, based on target performance-accuracy budget the most suitable model can be chosen at run time from a group of pre-tuned model. A number of recent works also proposed use of CP decomposition of the kernel tensor to reduce the size of the network ([28],[39]). But, Silva et al. has shown regardless of the choice of the norm, finding the best low-rank approximation kernel using CP decomposition is an ill-posed problem - the best low-rank approximation may not exist [12]. In our proposed scheme, the decomposition always exists and has an exact closed form solution.

A number of recent works have addressed the problem of redundancy in a deep model by constructing equivariant representation in the model ([9],[34], [38]). Dieleman et al. showed that rotational symmetry can be exploited in convolutional networks for the problem of galaxy morphology prediction by rotating feature maps, effectively learning an equivariant representation [15]. Shang et al. proposed a new activation scheme, concatenated ReLU (CReLU), which conserves both positive and negative linear responses after convolution so that each filter can effectively represent its unique direction. When CReLU is used in convolutional network instead of standard ReLU it helps in parameter reduction [34]. Although, all those techniques provide a more direct way of exploiting redundancy in the model, they are only limited to capturing a fixed number of orientations in symmetry (e.g. dihedral and cyclic symmetry). In contrast, our method are not limited to any fixed number of orientation in symmetry as we do not impose this constraint in the architecture. As a result, our technique exploits redundancy from more degree of freedoms and hence help in further speed-up.

5 COMPARSION WITH PREVIOUS WORK

As described in the related work section of this paper (section 4) the most common schemes for deep network compression are pruning, one-shot approximation, quantization and fast arithmetic (e.g. Winograd's method [27]). Winograd's fast convolution scheme can be applied to reduce number of strong operations (in our case multiplication), but they are orthogonal to our scheme. For example, a F(4,3) version of the algorithm can still be applied on top of our approximation scheme to further reduce the number of strong computation by 4x. Since, we decompose each layer into two 1-D layers, the 4x complexity reduction will come from 2x reduction in each 1-D convolution stage. Similarly, quantization can be carried out to reduce overall storage footprint of models after our ADaPT scheme is applied to reduce total number of computations. We focus our comparison to convolutional layers as they contribute to more than 90% of the computation in CNNs.

We compare our ADaPT scheme with a pruning [19] and an alternative low-rank approximation scheme based on Tucker decomposition [24]. Both these techniques help to reduce number of convolutions in CNN and hence it is worth comparing. For a layerwise comparison, we considered most commonly used VGG-16 model (uses IMAGENET dataset) as both the other research work used the same network as a common use case. Table 8 provides a

Layers (mxn-maps-name)	Feature Size (MxN)	Speedup (x) (a) 2-D Direct (#MULs)	(b) Tucker	(c) Pruning	(d) ADaPT
conv3x3-64-1.1	224x224	1.0x (0.1B)	1.0x	1.7x	2.1x
conv3x3-64-1.2	224x224	1.0x (1.85B)	10.2x	8.3x	8.0x
conv3x3-128-2.1	112x112	1.0x (0.9B)	9.2x	3.3x	3.2x
conv3x3-128-2.2	112x112	1.0x (1.85B)	6.7x	3.4x	4.8x
conv3x3-256-3.1	56x56	1.0x (0.9B)	4.0x	2.3x	5.1x
conv3x3-256-3.2	56x56	1.0x (1.85B)	7.8x	6.3x	6.4x
conv3x3-256-3.3	56x56	1.0x (1.85B)	3.0x	3.5x	5.5x
conv3x3-512-4.1	28x28	1.0x (0.9B)	4.5x	4.8x	6.4x
conv3x3-512-4.2	28x28	1.0x (1.85B)	3.9x	7.1x	7.7x
conv3x3-512-4.3	28x28	1.0x (1.85B)	5.4x	6.7x	7.0x
conv3x3-512-5.1	14x14	1.0x (462.5M)	5.2x	8.3x	9.6x
conv3x3-512-5.2	14x14	1.0x (462.5M)	5.7x	11.1x	9.8x
conv3x3-512-5.3	14x14	1.0x (462.5M)	5.4x	9.1x	9.8x

Table 8: VGG16 layerwise comparison of number of strong operations, i.e. multiplication (MULs) between (a) direct 2-D approach, (b) Tucker decomposition, (c) Pruning, and (d) ADaPT Scheme

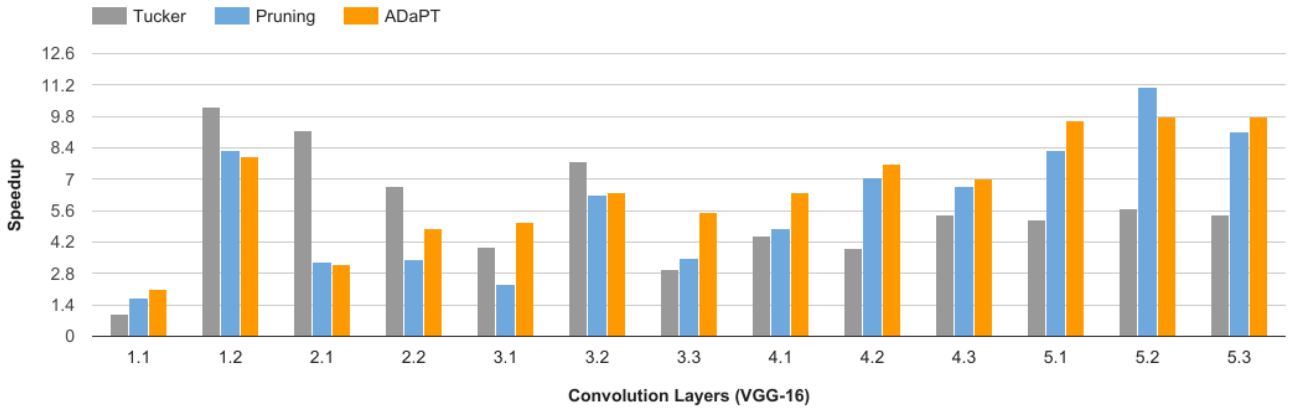

Figure 6: VGG Layerwise FLOPs comparison between Tucker Decomposition, Pruning and our ADaPT scheme.

layerwise comparison of size of feature maps along with number of multiplications required in case of a direct 2-D convolution on the VGG-16 baseline model. The last three columns compare speedups from three different schemes, namely, Tucker decomposition, pruning while learning, and our ADaPT scheme.

Figure 6 presents a graphical comparsion of number of flops for VGG-16 for the three selected schemes. As can be seen from the figure, our ADaPT scheme wins in almost all the layers against pruning. Tucker decomposition based scheme achieves better speedup for the first five layers except the very first input layer. As we go more deeper in the network ADaPT becomes the best solution. As middle layers dominates in number of computations as seen can be the table 8, ADaPT yields overall more speedup compared to Tucker decomposition scheme. Pruning also helps speeding up the later layers compared to the Tucker decomposition based technique. But, this comes at the cost of very long iterative re-training phase. From the result, we can conclude that our scheme ADaPT achieves best speedup without the need of any long re-training time.

6 CONCLUSIONS AND FUTURE WORK

In this paper we have demonstrated that a correlation based mathematically well grounded technique can be used to speed-up deep networks by exploiting redundancies among feature maps in CNNs. We have introduced an easy-to-implement three-step approximation technique which can be applied on state-of-the-art pre-trained models statically. The availability of several pre-tuned models with different performance-accuracy targets can be a significant advantage for deploying CNNs on fast time-to-market applications. Although our research is primarily aimed at embedded and IoT applications, this approximation scheme can also enable acceleration at semi-embedded (e.g. autonomous cars, military drones) or server-grade platforms. We have evaluated our technique on a variety of CNNs targeting different class sizes and number of layers. The success from the evaluation running on a range of real hardware provides strong evidence that kernel decomposition combined with approximation is a promising approach for speeding up pre-trained CNNs without sacrificing significant accuracy.

The head-to-head comparison with pruning based technique demonstrates that our ADaPT scheme can not only produce comparable speed-up in performance but also can be more effective in deep models. In addition, the reduction in compute complexity and memory footprint comes without the cost of long prune and re-training cycle. Although we can obtain very promising results with ADaPT's static 3-stage scheme, it is not fully investigated yet whether the solution is optimal or not. As a future work, we will be investigating optimality of our proposed scheme. Also, in future work it would be interesting to explore how other orthogonal techniques can be combined with our approximation technique to compress models further. We also have noticed that sometimes training a low-rank constrained model from scratch slightly outperforms their decomposed version of the model from pre-trained ones. This could be another interesting avenue for designing more accurate and efficient convolutional neural networks.

REFERENCES

[1] 2013. Forecast: The Internet of Things, Worldwide, 2013. https://www.gartner.com/doc/2625419/forecast-internet-things-worldwide-. (2013). (Accessed on 03/24/2017).

[2] 2017. Apple Watch. https://en.wikipedia.org/wiki/Apple_Watch. (2017). (Accessed on 03/24/2017).

[3] 2017. Resource & Design Center for Development with Intel. https://edc.intel.com/Gateway-Comparison/?language=en. (2017). (Accessed on 03/24/2017).

[4] Massimo Alioto. 2017. Enabling the Internet of Things: From Integrated Circuits to Integrated Systems. *Springer International Publishing* (2017). http://www.springer.com/us/book/9783319514802

[5] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2015. Structured Pruning of Deep Convolutional Neural Networks. *CoRR* abs/1512.08571 (2015). http://arxiv.org/abs/1512.08571

[6] Sajid Anwar and Wonyong Sung. 2016. Compact Deep Convolutional Neural Networks With Coarse Pruning. *CoRR* abs/1610.09639 (2016). http://arxiv.org/abs/1610.09639

[7] Aaron Carroll and Gernot Heiser. 2010. An Analysis of Power Consumption in a Smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference (USENIXATC'10)*. USENIX Association, Berkeley, CA, USA, 21–21. http://dl.acm.org/citation.cfm?id=1855840.1855861

[8] Y. H. Chen, J. Emer, and V. Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 367–379. DOI: http://dx.doi.org/10.1109/ISCA.2016.40

[9] Taco S. Cohen and Max Welling. 2016. Group Equivariant Convolutional Networks. *CoRR* abs/1602.07576 (2016). http://arxiv.org/abs/1602.07576

[10] Matthieu Courbariaux and Yoshua Bengio. 2016. BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *CoRR* abs/1602.02830 (2016). http://arxiv.org/abs/1602.02830

[11] Bill Dally. 2011. Power, Programmability, and Granularity: The Challenges of ExaScale Computing. In *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium (IPDPS '11)*. IEEE Computer Society, Washington, DC, USA, 878–. DOI: http://dx.doi.org/10.1109/IPDPS.2011.420

[12] Vin de Silva and Lek-Heng Lim. 2008. Tensor Rank and the Ill-Posedness of the Best Low-Rank Approximation Problem. *SIAM J. Matrix Anal. Appl.* 30, 3 (2008), 1084–1127. DOI: http://dx.doi.org/10.1137/06066518X arXiv:http://dx.doi.org/10.1137/06066518X

[13] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. 2013. Predicting Parameters in Deep Learning. In *Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS'13)*. Curran Assoc. Inc., USA, 2148–2156. http://dl.acm.org/citation.cfm?id=2999792.2999852

[14] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation. In *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS'14)*. MIT Press, Cambridge, MA, USA, 1269–1277. http://dl.acm.org/citation.cfm?id=2968826.2968968

[15] Sander Dieleman, Jeffrey De Fauw, and Koray Kavukcuoglu. 2016. Exploiting Cyclic Symmetry in Convolutional Neural Networks. *CoRR* abs/1602.02660 (2016). http://arxiv.org/abs/1602.02660

[16] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep Learning with Limited Numerical Precision. *CoRR* abs/1502.02551 (2015). http://arxiv.org/abs/1502.02551

[17] Philipp Gysel, Mohammad Motamedi, and Soheil Ghiasi. 2016. Hardware-oriented Approximation of Convolutional Neural Networks. *CoRR* abs/1604.03168 (2016). http://arxiv.org/abs/1604.03168

[18] Song Han, Huizi Mao, and William J. Dally. 2015. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. *CoRR* abs/1510.00149 (2015). http://arxiv.org/abs/1510.00149

[19] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning Both Weights and Connections for Efficient Neural Networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS'15)*. MIT Press, Cambridge, MA, USA, 1135–1143. http://dl.acm.org/citation.cfm?id=2969239.2969366

[20] Babak Hassibi and David G. Stork. 1993. Second Order Derivatives for Network Pruning: Optimal Brain Surgeon. In *Advances in Neural Information Processing Systems 5, [NIPS Conference]*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 164–171. http://dl.acm.org/citation.cfm?id=645753.668069

[21] M. Horowitz. 2014. 1.1 Computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. 10–14. DOI: http://dx.doi.org/10.1109/ISSCC.2014.6757323

[22] R.L. Hummel, D.G. Lowe, and Courant Institute of Mathematical Sciences. Computer Science Department. 1987. *Computing Large-kernel Convolutions of Images*. Number nos. 253-264 in Computing Large-kernel Convolutions of Images. New York University. https://books.google.co.uk/books?id=TDV7uAAACAAJ

[23] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up Convolutional Neural Networks with Low Rank Expansions. *CoRR* abs/1405.3866 (2014). http://arxiv.org/abs/1405.3866

[24] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. 2015. Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications. *CoRR* abs/1511.06530 (2015). http://arxiv.org/abs/1511.06530

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 2012.

[26] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar. 2016. DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices. In *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. 1–12. DOI: http://dx.doi.org/10.1109/IPSN.2016.7460664

[27] Andrew Lavin. 2015. Fast Algorithms for Convolutional Neural Networks. *CoRR* abs/1509.09308 (2015). http://arxiv.org/abs/1509.09308

[28] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan V. Oseledets, and Victor S. Lempitsky. 2014. Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition. *CoRR* abs/1412.6553 (2014). http://arxiv.org/abs/1412.6553

[29] Yann LeCun, John S. Denker, and Sara A. Solla. 1990. Optimal Brain Damage. In *Advances in Neural Information Processing Systems 2*, D. S. Touretzky (Ed.). Morgan-Kaufmann, 598–605. http://papers.nips.cc/paper/250-optimal-brain-damage.pdf

[30] Y. LeCun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard. 1990. *Handwritten Digit Recognition: Applications of Neural Net Chips and Automatic Learning*. Springer Berlin Heidelberg, Berlin, Heidelberg, 303–318. DOI: http://dx.doi.org/10.1007/978-3-642-76153-9_35

[31] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. 2015. Sparse Convolutional Neural Networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[32] J. Long, E. Shelhamer, and T. Darrell. 2015. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3431–3440. DOI: http://dx.doi.org/10.1109/CVPR.2015.7298965

[33] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning Convolutional Neural Networks for Resource Efficient Transfer Learning. *CoRR* abs/1611.06440 (2016). http://arxiv.org/abs/1611.06440

[34] Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. 2016. Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units. *CoRR* abs/1603.05201 (2016). http://arxiv.org/abs/1603.05201

[35] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014). http://arxiv.org/abs/1409.1556

[36] A. Sironi, B. Tekin, R. Rigamonti, V. Lepetit, and P. Fua. 2015. Learning Separable Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37, 1 (Jan 2015), 94–106. DOI: http://dx.doi.org/10.1109/TPAMI.2014.2343229

[37] Cheng Tai, Tong Xiao, Xiaogang Wang, and Weinan E. 2015. Convolutional neural networks with low-rank regularization. *CoRR* abs/1511.06067 (2015). http://arxiv.org/abs/1511.06067

[38] Min Wang, Baoyuan Liu, and Hassan Foroosh. 2016. Factorized Convolutional Neural Networks. *CoRR* abs/1608.04337 (2016). http://arxiv.org/abs/1608.04337

[39] Peisong Wang and Jian Cheng. 2016. Accelerating Convolutional Neural Networks for Mobile Applications. In *Proceedings of the 2016 ACM on Multimedia Conference (MM '16)*. ACM, New York, NY, USA, 541–545. DOI: http://dx.doi.org/10.1145/2964284.2967280