GPGPU Accelerated Deep Object Classification on a Heterogeneous Mobile Platform

electronics

MDPI

Article

GPGPU Accelerated Deep Object Classification on a Heterogeneous Mobile Platform

Syed Tahir Hussain Rizvi [1,*], Gianpiero Cabodi [1], Denis Patti [1,*] and Gianluca Francini [2]

[1] Dipartimento di Automatica e Informatica (DAUIN), Politecnico di Torino, Turin 10129, Italy; gianpiero.cabodi@polito.it

[2] Joint Open Lab, Telecom Italia Mobile (TIM), Turin 10129, Italy; gianluca.francini@telecomitalia.it

* Correspondence: syed.rizvi@polito.it (S.T.H.R.); denis.patti@polito.it (D.P.); Tel.: +39-011-0907048 (S.T.H.R. & D.P.)

Academic Editor: Mostafa Bassiouni

Received: 5 September 2016; Accepted: 5 December 2016; Published: 9 December 2016

Abstract: Deep convolutional neural networks achieve state-of-the-art performance in image classification. The computational and memory requirements of such networks are however huge, and that is an issue on embedded devices due to their constraints. Most of this complexity derives from the convolutional layers and in particular from the matrix multiplications they entail. This paper proposes a complete approach to image classification providing common layers used in neural networks. Namely, the proposed approach relies on a heterogeneous CPU-GPU scheme for performing convolutions in the transform domain. The Compute Unified Device Architecture(CUDA)-based implementation of the proposed approach is evaluated over three different image classification networks on a Tegra K1 CPU-GPU mobile processor. Experiments show that the presented heterogeneous scheme boasts a 50× speedup over the CPU-only reference and outperforms a GPU-based reference by 2×, while slashing the power consumption by nearly 30%.

Keywords: machine vision; image analysis; image processing; concurrent computing; neural networks; mobile computing; multicore processing; convolution; ubiquitous computing

1. Introduction

Deep convolutional networks have recently shown top performance in important machine learning tasks, such as image classification [1,2]. Image classification via deep neural networks involves two stages: an offline learning stage (training) followed by proper image classification afterwards (testing). During the learning stage, a network is trained over a set of labeled input images where the network parameters (weights and biases) are iteratively adjusted to predict the output values corresponding to the input labels. Once the network training is complete, the learned set of parameters can be used for the classification of new sample images.

Training and testing a neural network is largely a parallel problem, so heterogeneous GPU-CPU architectures are commonly employed to speedup such processes. CPUs are well-suited for sequential tasks due to higher operational frequencies, whereas GPUs can execute concurrent tasks efficiently thanks to their Single Instruction Multiple Threads (SIMT) architecture. By proper scheduling of the computing resources of heterogeneous CPU-GPU architectures, the training and testing time of neural networks can be largely reduced [3–7].

Among the downsides of deep neural networks, one of the downsides is their computational complexity, which is an issue especially with embedded devices. The complexity of a deep neural network for image classification stems largely from the convolutional layers. Convolutional layers serve the key purpose of extracting features robust to changes in the image scale and illumination.

The core operation in a convolutional layer is multiplying a matrix representing an image block with another matrix representing a feature. A typical deep network includes several convolutional layers, so their impact on the overall network complexity is extensive. Training has the largest computational and memory complexity; however, it can be performed offline on dedicated server infrastructures. Conversely, depending on the application constraints, the actual image classification may have to be performed online on the same embedded device used to acquire the image (e.g., on a smart phone) [8–10]. Recent embedded devices boast heterogeneous CPU-GPU architectures and are suited for challenging applications, such as robotics, control and image processing [3,4,11,12]. Nevertheless, the amount of computational resources available on a mobile device is much lower compared to what is available in a desktop environment. Therefore, efficient computing architectures for deploying neural networks on heterogeneous embedded devices are highly sought.

In this work, a comprehensive approach for image classification with deep convolutional neural networks on embedded devices is presented.The presented approach features the most common layers required to perform image classification, such as convolutions, max-pooling, batch-normalization and activation functions. Parallel computing capabilities of mobile GPUs are exploited to speedup the most computationally-intensive operations carried out within each layer. Most notably, a highly-performing yet power-efficient scheme for performing convolutions in the transform domain is proposed. The proposed convolutional scheme leverages heterogeneous CPU-GPU mobile architectures to maximize the computational throughput while accounting for memory bandwidth constraints. This work is implemented in the CUDA language, which bridges the gap between mobile and desktop environments. The proposed approach is evaluated by implementing three well-known convolutional network architectures for image classification, namely AlexNet, OverFeat and ResNet-34, on a Nvidia Shield K1 tablet [13,14]. The experiments demonstrate that the proposed heterogeneous CPU-GPU convolutional scheme outperforms a CPU-only reference by a factor of 50×. Most notably, the proposed heterogeneous convolutional scheme improves by up to a factor of two over the GPU-only scheme, reducing the device overall power consumption up to 30%.

The paper is organized as follows: Section 2 reviews the related work in the field of image classification via deep neural networks and its GPU-based implementation on mobile phones. The methodology of the proposed system is presented in Section 3. Section 4 discusses the architecture of deep classifiers and the entailed layers. Test models and different parameters related to the embodiment of deep neural networks are discussed in Section 5. Section 6 presents the results and discusses the performance of the proposed approach in a wider context. Finally, Section 7 concludes the work and proposes the future research directions.

2. Background and Related Works

This section provides first the relevant background on deep convolutional neural networks and their application to the problem of image classification. Next, the relevant literature concerning deep neural networks on mobile devices motivating the present research is discussed.

2.1. Background on the Convolutional Neural Network for Image Classification

Convolutional neural networks are becoming an all-encompassing package for a number of computer vision and machine learning tasks. Image classification is an important computer vision task that has numerous practical applications. Over the last few years, image classification has been revolutionized by the advent of deep learning-based architectures. Deep classifiers have shattered existing performance records in several image classification contests, enabling a large leap over previous so-called shallow classifiers. A typical deep network for image classification tasks is mainly composed of two stages. The first stage accounts for the extraction of features, also called kernels or filters, that are robust to changes in illumination, size and translations from the input image. A detailed description of such a feature extraction stage is provided in Section 4. Next, one or more fully-connected layers in the second and final stage performs the feature classification on a highly

dimensional space, providing the final output. Recent results show that the depth of a neural network (number of stacked layers) plays an important role in achieving better classification accuracy [1,13,15]. Deep convolutional networks have showed recognition accuracy comparable to humans in many visual analysis tasks [14,16,17]. Larger datasets and deeper models lead to better accuracy, but also increase the training and classification time [2,13,15]. Heterogeneous systems play a pivotal role in the field of visual analysis where the computational capabilities of CPUs and GPUs can be utilized altogether to maximize the performance of deep classifiers in terms of training and classification time. In the following section, the available frameworks for implementing image classification via deep neural architectures are discussed.

2.2. Deep Classifiers on Heterogeneous Mobile Architectures

A number of frameworks for implementing deep neural networks are currently available. Such frameworks leverage the parallel computational capabilities of modern GPUs, allowing to practically train and deploy deep neural networks in a reasonable time. However, most of these frameworks are tailored to desktop and server platforms; therefore, they do not take into account the unique peculiarities of mobile devices. Mobile devices' peculiarities include, among others, reduced memory and limited energy stored in the battery: operations are optimized just for execution speed, without considering the need to extend battery life. In the following, major frameworks that are currently available for implementing deep neural networks are discussed. The shortcomings of these are discussed when it comes to mobile applications, and the need for efficient schemes designed ad hoc for mobile platforms is highlighted and motivated.

Torch is an open source scientific computing framework that comes with machine learning libraries. Using these open source libraries, deep convolutional networks can be created and trained easily in the Torch environment on a desktop computer. Torch relies on CUDA for efficient operations on NVIDIA GPUs. CUDA also enables clustering of GPUs to accelerate the process of training a network. Torch relies however on a number of third-party libraries for performing ancillary operations on the CPU. However, porting such libraries to mobile CPUs is not always possible because of architecture differences and the memory footprint thereof. In addition, GPU clustering is not applicable to mobile platforms as mobile platforms typically include just one GPU. Finally, there exist a Torch library for mobile terminals; however, this library is not mature in terms of dependencies.

DeepSense [18] is a deep neural network framework designed specifically for mobile devices. However, such a framework relies on OpenCL, which is less efficient in terms of performance and much slower than the same CUDA-based built-in functions that require much optimization [19]. Note that OpenCL supports a wide range of embedded devices. However, OpenCL is not directly supported on Android starting from Version 4.3 onwards. OpenCL on Android requires in fact an intermediate library to access the GPU hardware. Such a library not only adds to the deep neural network footprint, but also incurs in additional computational overhead. Therefore, it is currently not feasible to implement the deep classifiers on recent mobile devices using standalone OpenCL. Furthermore, the deep learning community is widely using CUDA libraries.

A unified framework to accelerate and compress the convolutional neural network is presented in [20]. In order to speed-up the process, deep classifiers are quantized, yielding to sparser parameter sets. However, the paper also shows that the parameter quantization and compression jeopardize the precision of the results [21–24]. Because this work only considers narrow neural networks with a maximum of 16 convolutional layers and no experiment is performed on deeper networks such as ResNet-34, it is not clear how the loss of precision is going to propagate through the many layers of a deeper network. In addition, computations are carried out on a single CPU core, without GPU acceleration. Therefore, this approach misses the benefits of GPU-accelerated neural networks as a whole, despite the reportedly high performance due to parameter quantization.

A mobile-GPU accelerated deep neural network flow is presented in [25]. Different techniques to optimize the various components of a typical neural network flow on mobile devices are discussed. However, the preliminary information that is presented is not sufficient to reproduce the results.

Concluding, the presented work differs from and improves upon the above-mentioned references addressing most of the above-listed issues. First of all, the proposed scheme has the benefits of jointly exploiting the computational capabilities of both the CPU and GPU, enabling true heterogeneous computations. Moreover, this approach has the merit of supporting nearly all layer types of neural networks found in a modern image classification networks and is suitable for deploying very complex topologies. In addition, the accuracy of the presented approach is similar to the existing desktop and server platforms, whereas it does not require an intermediate framework for actual image classification. The proposed scheme can be easily integrated into an android application and provides compatibility for models trained with other desktop/server frameworks.

3. Methodology

Figure 1 illustrates the proposed flow to realize a deep convolutional classifier on a mobile device. First of all, state-of-the-art neural classifiers are trained in the Torch framework. A powerful GPU server is used to train the required neural architectures. A few preprocessing techniques are also used before training the classifiers. Neural networks need a fixed size input, while training data may be collected from various sources and can be of different sizes/dimensions. Therefore, it is necessary to crop or scale the images of different sizes and dimensions to fit the defined architecture. There are several approaches to perform the cropping and scaling. Data normalization is also an important preprocessing step and useful when the inputs are generally on a widely-spaced scale. It is performed here by removing the mean value of each feature and then dividing the non-constant features by their standard deviation. Since the classifiers require being trained on a vast amount of training data, data augmentation is used to improve the classification accuracy. Flipping, random cropping, reflection, color jittering and other different data augmenting techniques are commonly combined to augment the dataset, improving performance and accelerating training. Finally, the trained neural network can be deployed in the field for the actual classification.

Once the training is completed, the trained parameters of neural models are ready to be imported for the classification purposes. In this work, the CUDA computing framework is used for the realization of identical neural architectures on an embedded device to exploit the already trained network. Required layers (convolution, pooling, batch normalization) and activation functions (tangent hyperbolic unit, rectifier linear unit and thresholding unit) are implemented and accelerated using GPGPU. Convolution is the core building block of the Convolutional Neural Network (CNN). Two versions of convolution are implemented to support the different trained models of Torch. Format conversion of imported parameters according to the used convolution method is also an important step to match the results with the desktop-based trained model. After replicating the required architectures using CUDA and importing the trained parameters, accelerated neural classifiers are executed on the mobile device as a part of the Android application. Implementation of just the classification phase of the convolutional neural network lowered the barrier of implementing the deep classifier on the heterogeneous mobile platform.

As shown in Figure 1, the integrated camera of the mobile phone is used to capture the image for the real-time classification by the trained network. Therefore, there is no need for an additional camera module as required by a single-board computer or desktop workstation. This captured image is then passed through the same pre-processing steps (cropping/scaling and normalization) as performed at the time of training. Because the trained network is for a specific input dimension and the captured image can be of different aspect ratios depending on the used mobile phone, so it is necessary to perform the scaling or cropping of the image to fit the input dimension. Similarly, normalization of the input image is also essential to minimize the bias for one feature over another and to restrict the range of input/output values. After these two preprocessing steps, the captured image is fed to the

CUDA-based replicated neural model to perform the task of classification using imported trained parameters. The final classification result is displayed using the display screen of the mobile phone.

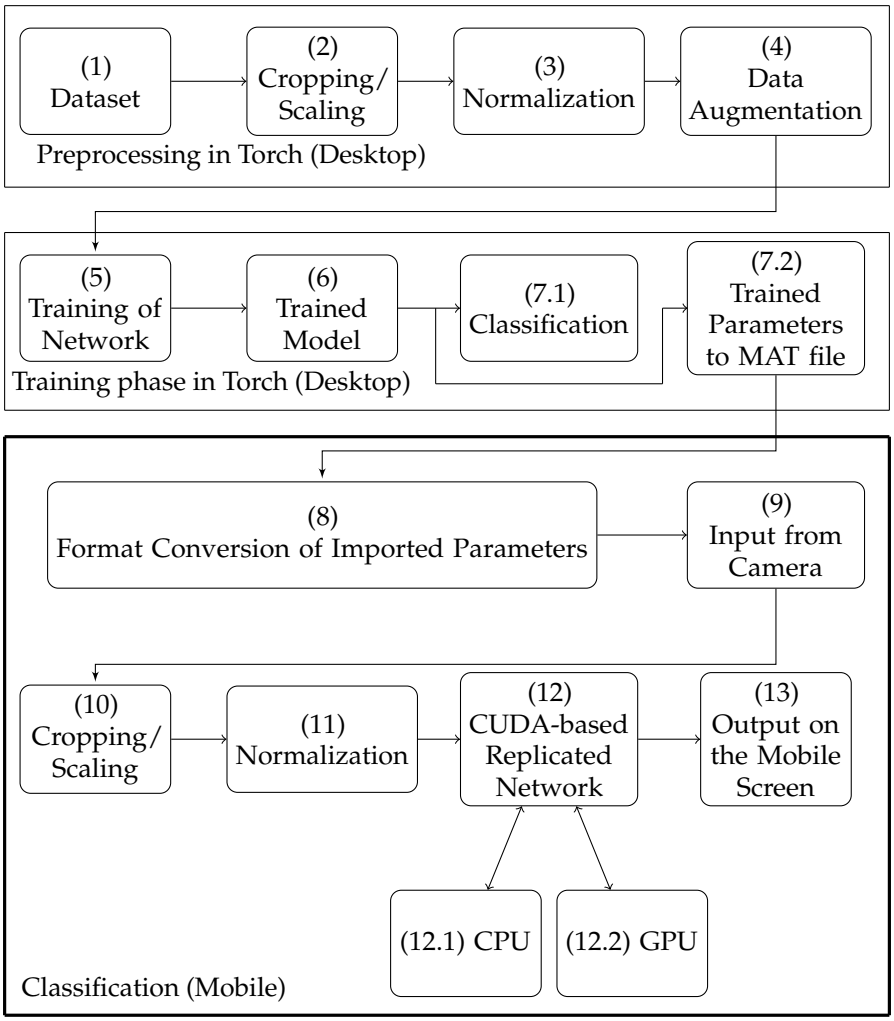


Figure 1. Block diagram of the mobile platform-based deep classifier.

4. Architecture of Deep Convolutional Networks

This section details the architecture of deep convolutional neural networks and the entailed layers.

4.1. Convolutional Layer

Convolutional layers serve the purpose of extracting robust features from the input images, and recent deep classifiers include many convolutional layers [13,15]. Convolution is a computationally-intensive task that can be accelerated using the concurrent processors [26]. Mathematically, it can be expressed as the sum of the products of mask/filter coefficients with the input function/image. The convolution operation can be extended to two or three dimensions according to the required solution.

There are different approaches to perform the convolution operation. In this paper, the multi-channel convolution operation is performed using two approaches for the adaptation of the Torch-based trained network on the mobile platform.

The first one is the traditional approach called full convolution. In this approach, the sum of the products of filters and the image is computed. Both input and output images (feature maps) are multi-channeled. If the input image has C channels, the required output map has D channels, the size

of the trained filter bank is $X \times Y$ and the size of the input image is $I \times J$, then the output of a single multi-channel fully-convolution layer can be represented by the following formula:

$$Output\_map_{i,j,d} = \sum_{c=1}^{C} \sum_{i1=1}^{X} \sum_{j1=1}^{Y} Input\_map_{c,i+i1,j+j1} Filters_{d,c,i1,j1} \quad (1)$$

Figure 2 also illustrates the computations performed by the fully-convolution layer. Pixels of input maps, filters and output maps are represented by their indexes in Figure 2. Index value $(1,3,2)$ of the input map represents the pixel present on the first row and third column of the second channel. The output image also uses the same representation where pixels are arranged as (row, column, output maps); while the values of filter banks have four values of indexes where $(1,2,3,2)$ represents (row, column, input maps, output maps (filter bank number)). In this particular example, input image has three channels and is of a size of 3×3. There are two filter banks having three channels (for each channel of the input map) of a size of 2×2. The resultant output map would have two channels because of two banks of the filter, while the size of the output feature map is dependent on the size of a single filter, the number of padding bits in the input image and the striding window. The GPU can show its potential where the values of the data can be processed independently. In this case, filters have to run in a sliding manner on every map of the input image, and this can be a slow process [27].

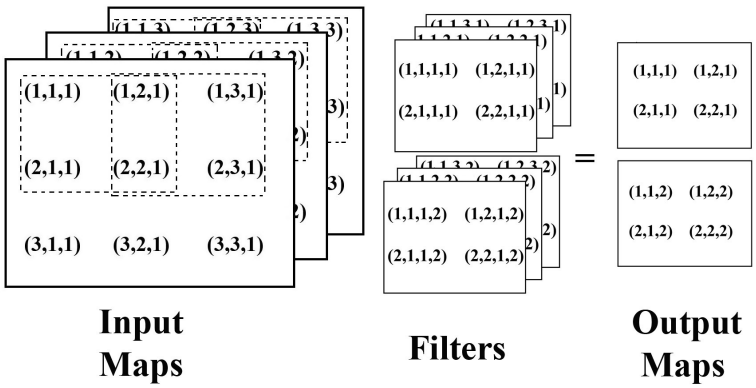


Figure 2. Three-dimensional convolution layer.

The second approach is the Matrix Multiplication-based Convolution (ConvMM). Using this approach, the input image and filters can be arranged in the transform domain to compute convolution just by multiplying these transformed data [28,29]. Figure 3 shows the matrix-based version of the convolutional layer where the input maps and filters are rewritten as the input and filter matrices. The pixels and indexes of Figure 2 are translated and shown in Figure 3 to explain the arrangement of multi-channeled input, filter and output maps in two-dimensional matrix form. By multiplying these two matrices, the resultant matrix is computed having the required output feature maps that can be separated to achieve the output equivalent of the traditional convolutional layer.

There are also other algorithms for computing the convolution, like Winograd or the lookup table-based approach, which can be used to accelerate the state-of-the-art deep classifiers having small filter and batch sizes [21,30].

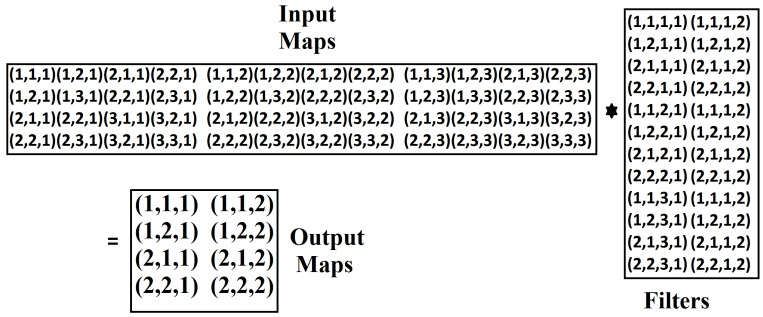


Figure 3. Convolution matrix multiplication.

4.2. Pooling Layers

In classical convolutional networks, the output or activations of the convolutional layer are passed to the pooling layer. The purpose of this layer is to achieve the spatial invariance by aggregating the information within a small local region. It basically reduces the size of feature maps. Two conventional options to perform the pooling are max and average. These pooling layers do not require any trainable parameter. There are also other pooling strategies that can be combined with some regularization techniques to improve the training of deep classifiers [27].

4.3. Batch Normalization

Batch normalization is also a significant part of deep classifiers, which yields substantial acceleration in the training. It preserves the representation ability of deep classifiers and ends the need for any further regularization [31]. The batch normalization function can be implemented using the following formula:

$$BN(x) = \frac{x - mean}{\sqrt{Var}} * gamma + beta \quad (2)$$

The value of mean and variance can be calculated over the training data; while gamma and beta are the scaling and shifting parameters to be used by the normalized output.

4.4. Activation Functions (ReLu, Tanh and Threshold)

There are also some important activation functions, like the tangent hyperbolic unit (tanh), Rectifier Linear unit (ReLu) and the thresholding unit, which are essential components to construct the architecture of trained classifiers. The purpose of these functions in neural architectures is to improve the training and eliminate the problems like the vanishing gradient [32].

5. Proposed Approach for Mobile Platforms

This section describes the proposed approach to the embodiment of a deep convolutional neural network for image classification on mobile devices.

5.1. GPU Accelerated Fully-Convolutional Layer

The traditional fully-convolutional layer is implemented using the CUDA language. This layer is accelerated using the concurrent three-dimensional grid of threads and blocks of the GPGPU. The complete workload is offloaded into the GPU for the acceleration of the convolution operation. One of the important tasks to realize the Torch-based replicated architecture is the formatting of the captured/imported input image and trained parameters. This formatting is required to achieve the same results in the CUDA computing environment as provided by the Torch model. The multi-dimensional input image and trained parameters are arranged and accessed in row-major order because the access of the contiguous array elements is faster. The padding function is also implemented in CUDA using the parallel resources of the GPGPU to support the same options

provided by the fully-convolutional layer of Torch. The structure of the implemented convolutional and pooling layers is different from the other layers because both of these layers are comprised of two kernels: one kernel to perform the padding of input data and the second kernel to perform the selected operation of convolution or pooling on this padded data. While the other layers, like the rectifier linear unit and tangent hyperbolic layers, are computed using the single kernel where the function of padding is not required. The flow of a padding-based convolutional and pooling layer is illustrated in Figure 4.

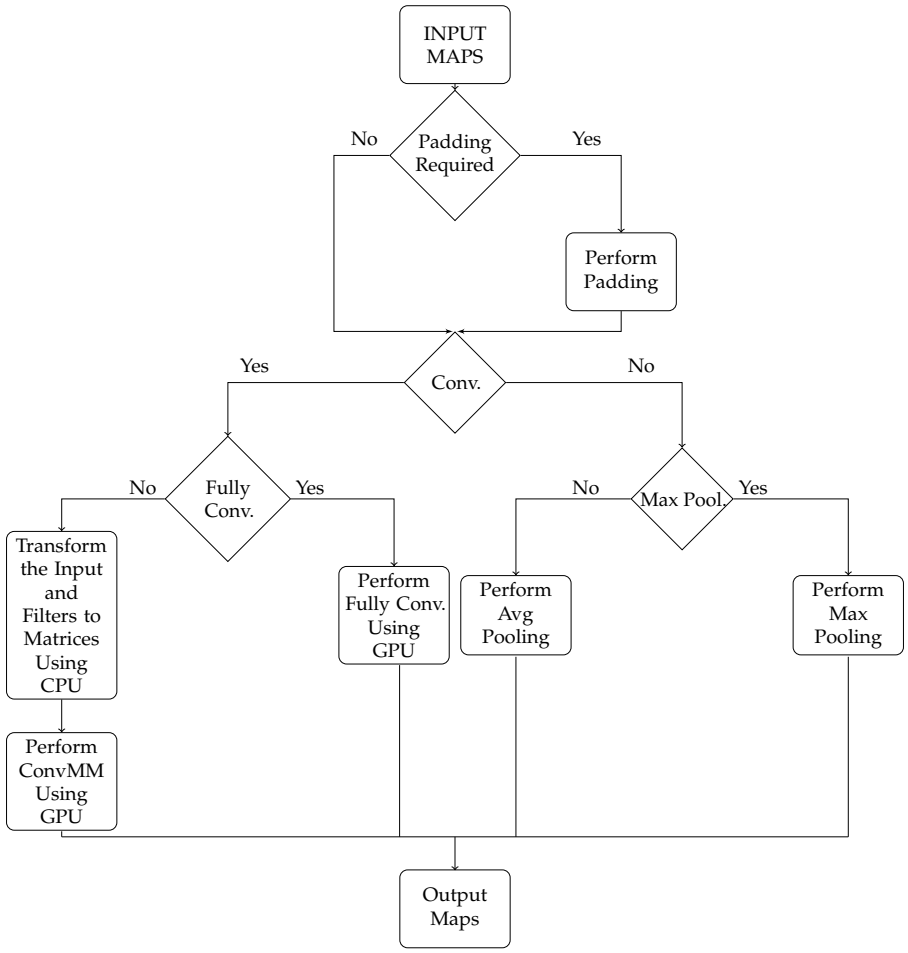
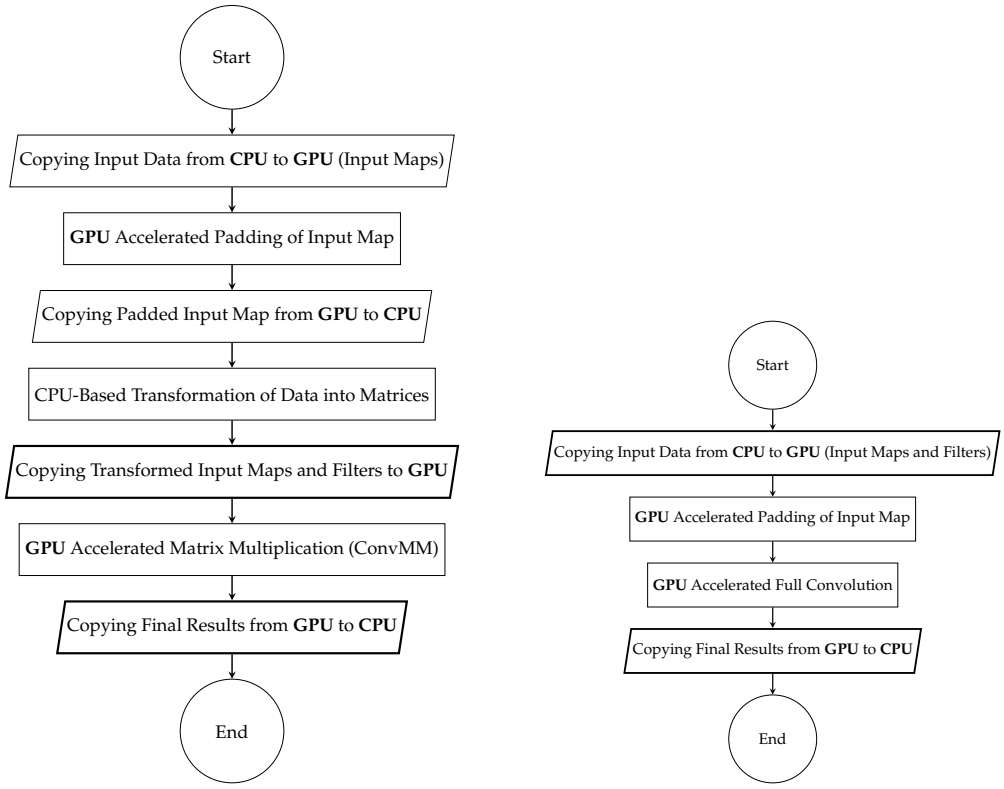


Figure 4. Flow of the convolutional and pooling layers. ConvMM, Convolution Matrix Multiplication.

5.2. CPU-GPU Accelerated Matrix Multiplications-Based Convolutional Layer

Matrix Multiplication-based Convolutional layer (ConvMM) is also implemented using the CUDA computing language. This approach to compute convolution is accelerated using heterogeneous resources of mobile device where the computational powers of both the CPU and GPU are exploited. Convolution Matrix Multiplication (ConvMM) is computed by partitioning the suitable computations between the CPU and GPGPU of the mobile device. The transformation of the image and filters into the matrices, which is a sequential task, is performed using the powerful CPU, and matrix multiplication of these transformed data is computed using GPU. This CPU-based transformation step cannot be performed concurrently because values from multi-dimensional maps have to be placed in a certain order so as to achieve the equivalent two-dimensional representation of data for multiplication that can be much slower if performed by a GPU that is not suited for sequential operations. Furthermore, input data are exchanged several times between the device (GPU) and host (CPU) memory to realize this

matrix multiplication-based convolution. Figure 5 visualizes the scheduling of computations between the CPU and GPU for ConvMM and the full convolutional layer.



(a) Flow of ConvMM **(b)** Flow of full convolution

Figure 5. Flow of ConvMM and full convolution.

5.3. GPU Accelerated Pooling Layer

Both functions of the pooling layer (average and Max) are implemented to realize the architecture of different classifiers on the mobile platform. The CUDA computing language is used to perform these operations concurrently and to exploit the computational power of the GPGPU to outperform the sequential versions of the same layer. The CUDA kernel for padding is also implemented using parallel resources of the GPGPU to support the same options provided by the pooling layers in Torch.

5.4. Other GPU Accelerated Layers

All remaining functions (batch normalization, ReLu, tanh and threshold) are also executed using the GPU's three-dimensional grid of concurrent blocks of threads for accelerating their operations. The batch normalization layer requires training of the mean, variance, gamma and beta parameters, as well, which are imported from the Torch model of the deep classifier to normalize the image using the CUDA-based accelerated function.

5.5. Implemented Neural Network Architectures

For the evaluation of deep classifiers on a mobile platform and the performance comparison of classifiers for both convolution approaches, three different architectures are implemented on the mobile device using the CUDA computing platform: AlexNet for the CIFAR-10 (Canadian Institute For Advanced Research 10) dataset; OverFeat and ResNet-34 for ImageNet [13,14]. These architectures

need the following number of layers and functions to be replicated on the mobile device to perform the classification:

These models are comprised of multiple layers with different pooling types and activation functions, as listed in Table 1. ResNet-34 has an additional layer of batch normalization in its architecture. All of these required functions are implemented in the CUDA computing framework to accelerate and replicate the trained network on the mobile device. Trained parameters are imported in the internal memory of the mobile device to complete the task of classification. The size of trained parameters is also an important factor to be considered for the successful implementation. These trained parameters include the weight and biases for the convolutional layers or values of gamma, beta, mean and variance for the batch normalization functions. The sizes of these parameters depend on the size of the filters and the dimensions of the feature maps of each layer. The sizes of these trained parameters of all three architectures are listed in Table 2.

Table 1. Deep models used for the implementation.

Model	No. of Layers	Required Functions
AlexNet	5	Conv. + tanh + Max Pool
OverFeat	8	Conv. + ReLu + Max Pool
ResNet-34	34	Conv. + Max Pool + Batch Normalization + ReLu + Avg.Pooling

Table 2. Size of the imported parameter.

Model	Size of File
AlexNet	20.8 MB
OverFeat	609 MB
ResNet-34	69.9 MB

6. Experiments and Result

In this paper, the Kepler K1 GPGPU of Nvidia Shield Tablet is used for the implementation of deep classifiers. The used target device has 192 cores and a quad-core CPU of 2.2 GHz. In this section, the performance analysis is presented first, before moving to power efficiency considerations.

6.1. Performance Evaluation

First of all, the comparison of both convolutional layers is performed to analyze the performance enhancement achieved by the matrix multiplication-based approach over the traditional one. The Matrix Multiplication-based Convolution (ConvMM) approach is also implemented using pure CPU- and pure GPU-based functions where both steps of transformation and multiplication are performed using homogeneous CPU or GPU systems. The execution time of all versions of convolution for varying the computational loads (image sizes) are listed in Table 3.

Figure 6 illustrates that for the smaller image sizes or lesser number of feature maps, traditional convolution performs well. However, as the size of the image increases over VGA ($640 \times 480 \times 3$), the convolution matrix multiplication approach outperforms the traditional convolution using heterogeneous computational power. Therefore, it concludes that the ConvMM layer would always outperform the fully-convolutional layer where the large dimension of the input image or a greater number of output feature maps have to be computed. The performance of different versions of the convolutional layer can be visualized in Figure 7. It verifies that the pure CPU- and GPU-based homogeneous implementations show poor performance and cannot match the computational capability of the heterogeneous approach for this type of algorithm. Heterogeneous ConvMM is $20\times$ faster than the pure GPU-based ConvMM and $40\times$ faster than the pure CPU-based sequential version of ConvMM. Results validate that the computations cannot be totally offloaded to the GPU for acceleration of the processes. Additionally, in some cases, significant gain in performance can be

achieved by distributing the sequential tasks to the CPU. In the ConvMM approach, the transformation step is the sequential task that can be performed well using the high operational frequency of CPU and cannot be parallelized using the GPU.

Table 3. Execution time of the convolution layers under various computational loads (ms) CIFAR: Canadian Institute For Advanced Research database; ImageNet: ImageNet database; VGA: Video Graphics Array; SVGA: Super Video Graphics Array; SXGA: Super eXtended Graphics Array; UXGA: Ultra eXtended Graphics Array, best results are written in bold.

Required Output Maps = 16				
Image Size	**Fully Convolution**	**ConvMM**	**ConvMM (Only CPU)**	**ConvMM (Only GPU)**
CIFAR ($32 \times 32 \times 3$)	**2.31**	4.20	37.21	10.9
ImageNet ($224 \times 224 \times 3$)	**28.23**	62.27	1570.98	1223.71
VGA ($640 \times 480 \times 3$)	**170.94**	327.53	10,055.24	5194.52
SVGA ($800 \times 600 \times 3$)	**276.16**	519.08	15,852.86	12,610.87
SXGA ($1280 \times 1024 \times 3$)	1960.57	**1040.48**	44,969.71	22,019.97
UXGA ($1600 \times 1200 \times 3$)	2404.2	**1696.19**	63,319.07	30,277.14

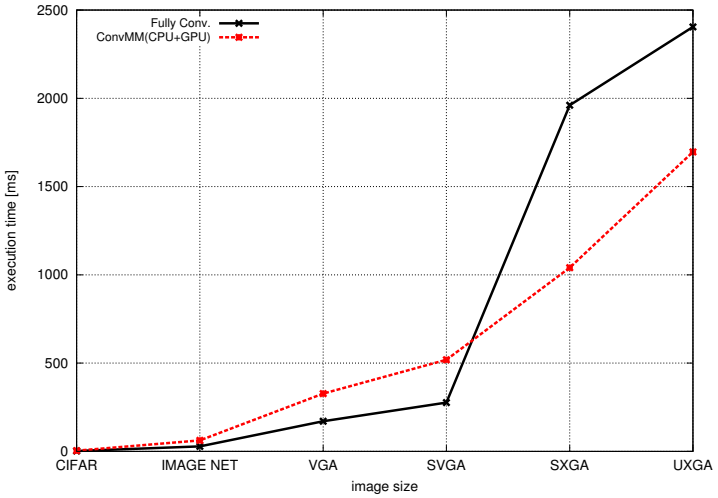


Figure 6. Full convolution vs. ConvMM as a function of input image size.

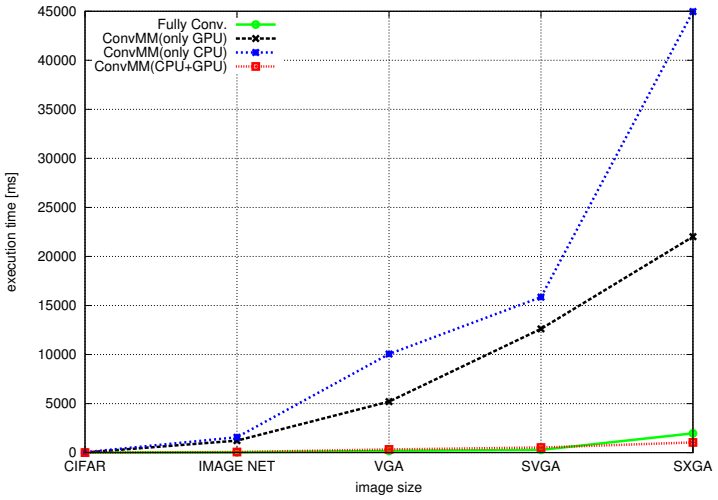


Figure 7. Comparison of different approaches to convolution operations as a function of the input image size.

It is well known that the data transfers between host and device are extremely expensive in terms of time. The ConvMM layer exchanges data four-times between the CPU and GPU, i.e., twice as many as the fully-convolutional layer as shown in Figure 5. Therefore, ConvMM and fully-convolutional layers are profiled to analyze the overheads caused by such data exchanges. The profiling shows that out of the four data transfers, the two former transfers are related to the transformed input image, which is however small in size and thus brings a small overhead in time. Conversely, the two latter transfers bring larger overhead because they also include the trained parameters. It is pointed out that even the pure GPU-based fully-convolutional layer needs two data transfers that are equivalent in overhead to the last two exchanges of ConvMM, as shown in Figure 5b, as they entail trained parameters' exchange.

Next, the experiment is performed by replacing the GPU-based padding and relative memory transfers with a CPU-based padding that requires no additional memory transfers. Namely, in this experiment, two initial memory transfers are avoided, shown at the top of Figure 5a, and the resulting architecture is shown in Figure 8.

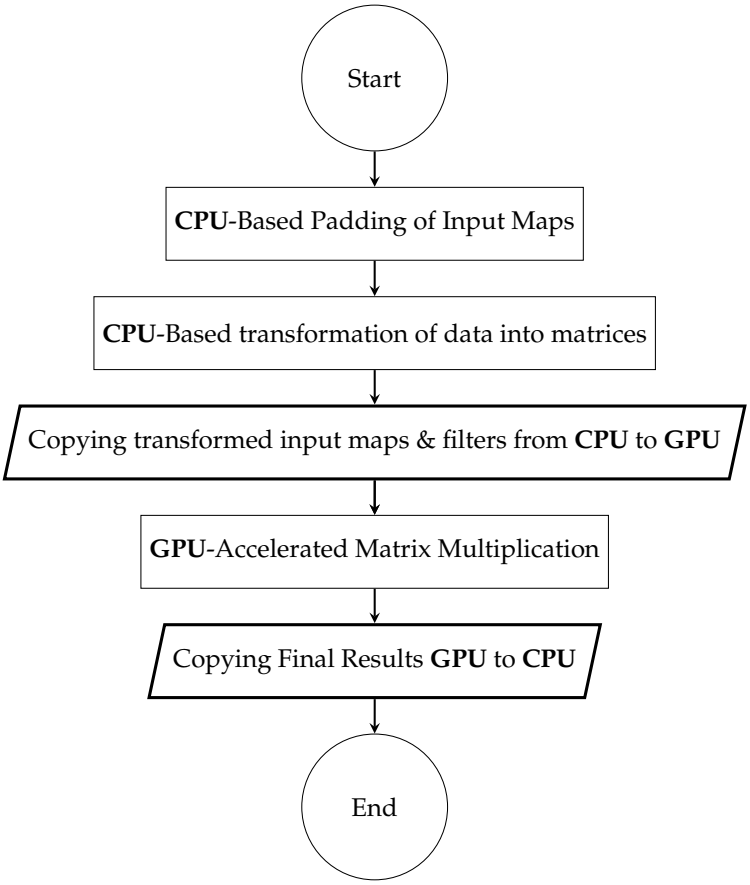


Figure 8. Flow of CPU padded ConvMM.

Table 4 presents the results of this experiment. The sequential CPU-based padding function does not accelerate the ConvMM layer despite two data transfers being avoided.

Table 4. Comparison of ConvMM execution time as a function of padding (ms), best results are written in bold.

Required Output Maps = 16		
Image Size	**Padding on CPU**	**Padding on GPU**
CIFAR (32 × 32 × 3)	**3.22**	4.20
ImageNet (224 × 224 × 3)	**56.85**	62.27
VGA (640 × 480× 3)	330.07	**327.53**
SVGA (800 × 600 × 3)	519.97	**519.08**
SXGA (1280 × 1024 × 3)	1148.94	**1040.48**
UXGA (1600 × 1200 × 3)	**1657.79**	1696.19

After this, the execution time taken by the rectifier linear layer is observed for the CPU-based sequential version and the GPU-based concurrent version. The CPU version is computed using the single thread of the processor of the mobile platform. Furthermore, the GPU-based layer is computed using 2D and 3D grids, which is a constraint for the GPU implementation. All results are tabulated in Table 5.

Table 5. Execution time of rectifier linear layers under various computational loads (ms), best results are written in bold.

Image Size	**CPU**	**GPU (2D Grid)**	**GPU (3D Grid)**
CIFAR (32 × 32 × 16)	5.75	**1.30**	1.47
ImageNet (224 × 224× 16)	39.73	13.18	**13.02**
VGA (640 × 480 × 16)	294.84	62.68	**36.09**
SVGA (800 × 600 × 16)	496.16	105.68	**97.14**
SXGA (1280 ×1024 × 16)	4059.09	360.62	**313.85**

As shown in Figure 9, the GPU implementation of the Rectifier Linear unit (ReLu) is 13× faster than the CPU-based sequential version. Additionally, there is a very minor difference of the performance enhancement in the case of 3D grid-based concurrent implementations over the 2D grid.

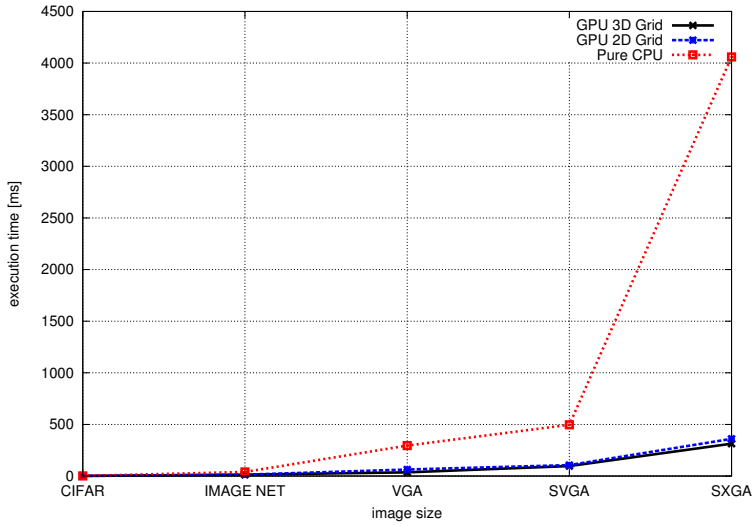


Figure 9. Rectifier linear unit performance on the mobile device as a function of the input image size.

Figure 10 shows the results for the max pooling layer. The GPU-based accelerated pooling layer is 4× faster than the sequential version implemented using the CPU of the mobile device.

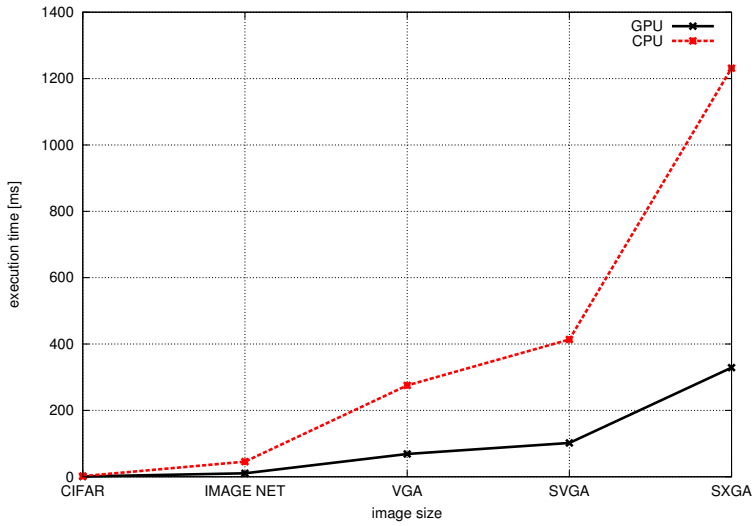


Figure 10. Max pooling operation performance on the mobile device as a function of the input image size.

Finally, three deep architectures (AlexNet, OverFeat and ResNet-34) are implemented on the mobile platform using the discussed layers, and their performances can be compared using Table 6.

Table 6. Classification time of deep models (ms), best results are written in bold.

Model	N° of Layers	CPU	GPU (Convfull)	GPU (ConvMM)	Speed Up (ConvMM)
AlexNet	5	10,234.2	584.757	**411.23**	25×
OverFeat	8	418,048.12	12,582.54	**8301.63**	50×
ResNet-34	34	248,366.93	11,481.74	**4938.01**	50×

Results show that the heterogeneously-accelerated deep classifiers are tens of times faster than the CPU-based sequential versions. Moreover, two of these classifiers (the ResNet-34 and OverFeat models) are trained on ImageNet, where ResNet-34 is a deep 34-layer architecture, but has fewer parameters than OverFeat and other available deep networks, and it has a higher accuracy rate than the others due to its depth [13]. The heterogeneous ConvMM-based "ResNet-34" architecture is 50× faster than its sequential version and 2× faster than the GPU-based fully-convolutional approach. Due to fewer parameters, the deeper version of a residual network, like ResNet-50 and ResNet-101, can also be implemented on the mobile platform with ease.

6.2. Power Consumption

Power consumption is a critical aspect of mobile applications, which must strike a balance between performance and energy efficiency, so as to maximize battery life. The power consumption of the mobile GPUs is typically high due to the many cores they encompass, while the power consumption of mobile CPUs is usually lower due to the low core count and other optimizations. However, GPUs can solve a parallel given task in less time than a sequential CPU, calling for a careful analysis of the resulting energy consumption tradeoffs. The NVIDIA Shield Tablet includes a battery of the following characteristics: 19.75 Wh and 5192 mAh; and Table 7 shows the actual measured energy consumption.

Table 7. Energy consumption of deep models (J), best results are written in bold.

Model	CPU	GPU (Convfull)	GPU (ConvMM)	Improvements (ConvMM)
AlexNet	16	**0.430**	0.461	34×
OverFeat	850.8	**13.2**	14.1	60×
ResNet-34	480	2.5	**1.8**	266×

Table 7 shows that the pure CPU-based implementations consume more energy than the GPU-accelerated versions due to the longer execution times. Namely, the heterogeneous ConvMM-based ResNet-34 classifier consumes 266× less battery energy than the sequential version of the same model. This experiment shows that the ResNet-34 architecture is more power efficient and has comparable performance as the OverFeat network over the same ImageNet dataset. Namely, ResNet-34 is more power efficient than OverFeat for the reason that it has smaller filters (3×3 vs. 11×11, respectively), despite the higher layer count (34 vs. eight).

7. Conclusions and Future Work

This paper presents a novel approach to real-time image classification via deep convolutional neural networks on heterogeneous mobile platforms. Experiments are performed by implementing AlexNet, OverFeat and ResNet-34 deep network architectures with the proposed approach. Networks are trained on desktop architectures, and then, the parameters are fed to a CUDA-based implementation of the proposed approach on the mobile device without any precision loss. Results show that the presented heterogeneous approach to deep image classification is up to 50× faster than a CPU-based sequential versions for the same architectures on the same mobile device. Furthermore, results confirm that the selection of an appropriate deep architecture for classification can significantly reduce the power consumption of the mobile device for the same algorithmic efficiency. Concluding, the presented heterogeneous CPU-GPU approach also enables up to 30% better power efficiency over a GPU-only approach.

As a future direction, the presented approach will be extended in order to exploit the most recent advances in neural networks and embedded architectures. In order to speed up computationally-intensive convolutional tasks, techniques like Winograd's minimal filtering technique can be adopted to reduce the arithmetic complexity of the convolution operation over small tiles. Winograd reduces the number of multiplications compared to traditional convolutions, and it is computed by an element-wise multiplication instead of a matrix multiplication. Further, the proposed scheme can also be optimized using the unified memory architecture of recent GPUs where extra memory transfers can be avoided by defining a more memory-efficient scheme. By using this scheme, the variable of the activation results and filters can be created in such a way that they are accessible both by the CPU and GPU to optimize the memory storage and avoid extra transfers.

Acknowledgments: We would like to thank Attilio Fiandrotti and Skjalg Lepsoy for their helpful comments and suggestions.

Author Contributions: Syed Tahir Hussain Rizvi and Denis Patti conducted the experiments and worked on the draft of the paper. Gianpiero Cabodi and Gianluca Francini are the academic tutors. They coordinated, supervised and approved the entire work.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

GPU	Graphics Processing Unit
GPGPU	General Purpose Graphics Processing Unit
SIMD	Single Instruction Multiple Data
SIMT	Single Instruction Multiple Thread
CUDA	Compute Unified Device Architecture
VGA	Video Graphics Array
SVGA	Super Video Graphics Array
SXGA	Super eXtended Graphics Array
UXGA	Ultra eXtended Graphics Array

References

1. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR, 2015. Available online: http://arxiv.org/abs/1409.1556 (accessed on 10 May 2016).
2. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. In Proceedings of The Twenty-sixth Annual Conference on Neural Information Processing Systems (NIPS), Lake Tahoe, NV, USA 3–8 December 2012; pp. 1097–1105.
3. Asano, S.; Maruyama, T.; Yamaguchi, Y. Performance comparison of FPGA, GPU and CPU in image processing. In Proceedings of the 2009 International Conference on Field Programmable Logic and Applications, Prague, Czech, 31 August–2 September 2009; pp. 126–131.
4. Naik, V.H.; Kusur, C.S. Analysis of performance enhancement on graphic processor based heterogeneous architecture: A CUDA and MATLAB experiment. In Proceedings of the 2015 National Conference on Parallel Computing Technologies (PARCOMPTECH), Bangalore, India, 19–20 February 2015; pp. 1–5.
5. Vandal, N.A.; Savvides, M. CUDA accelerated illumination preprocessing on GPUs. In Proceedings of the 2011 17th International Conference on Digital Signal Processing (DSP), Corfu, Greek, 6–8 July 2011; pp. 1–6.
6. Raghav, S.; Ruggiero, M.; Marongiu, A.; Pinto, C.; Atienza, D.; Benini, L. GPU Acceleration for Simulating Massively Parallel Many-Core Platforms. *IEEE Trans. Parallel Distrib. Syst.* **2015**, *26*, 1336–1349.
7. Baek, A.R.; Lee, K.; Choi, H. Speed-up image processing on mobile CPU and GPU. In Proceedings of the 2015 Asia Pacific Conference onMultimedia and Broadcasting (APMediaCast), Kuta, Indonesia, 23–25 April 2015; pp. 1–3.
8. López, M.B.; Nykänen, H.; Hannuksela, J.; Silvén, O.; Vehviläinen, M. Accelerating image recognition on mobile devices using GPGPU. *Parallel Process. Imaging Appl. SPIE* **2011**, *7872*, 78720R.
9. Huang, Y.; Wu, R.; Sun, Y.; Wang, W.; Ding, X. Vehicle Logo Recognition System Based on Convolutional Neural Networks With a Pretraining Strategy. *IEEE Trans. Intell. Transp. Syst.* **2015**, *16*, 1951–1960.
10. Abdulnabi, A.H.; Wang, G.; Lu, J.; Jia, K. Multi-Task CNN Model for Attribute Prediction. *IEEE Trans. Multimed.* **2015**, *17*, 1949–1959.
11. Rizvi, S.T.H.; Cabodi, G.; Patti, D.; Gulzar, M.M. Comparison of GPGPU based robotic manipulator with other embedded controllers. In Proceedings of the 2016 International Conference on Development and Application Systems (DAS), Suceava, Romania, 19–21 May 2016; pp. 10–15.
12. Satria, M.T. Real-time system-level implementation of a telepresence robot using an embedded GPU platform. In Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 14–18 March 2016; pp. 1445–1448.
13. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. CoRR. 2015. Available online: http://arxiv.org/abs/1512.03385 (accessed on 21 July 2016).
14. Sermanet, P.; Eigen, D.; Zhang, X.; Mathieu, M.; Fergus, R.; LeCun, Y. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. Computer Science—Computer Vision and Pattern Recognition. 2013. Available online: https://arxiv.org/abs/1312.6229 (accessed on 1 August 2016).
15. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.E.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going Deeper with Convolutions. CoRR. 2014. Available online: http://arxiv.org/abs/1409.4842 (accessed on 21 July 2016).

16. He, K.; Zhang, X.; Ren, S.; Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 7–13 December 2015; pp. 1026–1034.
17. Zeiler, M.D.; Fergus, R. Visualizing and Understanding Convolutional Networks. In Proceedings of the 13th European Conference on Computer Vision—ECCV 2014, Zurich, Switzerland, 6–12 September 2014; Part I, pp. 818–833.
18. Huynh, L.N.; Balan, R.K.; Lee, Y. DeepSense: A GPU-based Deep Convolutional Neural Network Framework on Commodity Mobile Devices. In Proceedings of the 2016 Workshop on Wearable Systems and Applications, Singapore, 30 June 2016; pp. 25–30.
19. Gu, J.; Liu, Y.; Gao, Y.; Zhu, M. OpenCL Caffe: Accelerating and Enabling a Cross Platform Machine Learning Framework. In Proceedings of the 4th International Workshop on OpenCL, Vienna, Austria, 19–21 April 2016; pp. 8:1–8:5.
20. Wu, J.; Leng, C.; Wang, Y.; Hu, Q.; Cheng, J. Quantized Convolutional Neural Networks for Mobile Devices. Computer Science—Computer Vision and Pattern Recognition. 2016. Available online: https://arxiv.org/abs/1512.06473 (accessed on 3 October 2016).
21. Lavin, A. Fast Algorithms for Convolutional Neural Networks. CoRR. 2015. Available online: http://arxiv.org/abs/1509.09308 (accessed on 5 May 2016).
22. Kim, Y.-D.; Park, E.; Yoo, S.; Choi, T.; Yang, L.; Shin, D. Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications. CoRR. 2015. Available online: http://arxiv.org/abs/1511.06530 (accessed on 24 April 2016).
23. Liu, X.; Turakhia, Y. Pruning of Winograd and FFT Convolution Algorithm. Available online: http://cs231n.stanford.edu/reports2016/117_Report.pdf (accessed on 17 May 2016).
24. Zheng, Z.; Li, Z. A Nagar and Kyungmo Park. In Proceedings of the 2015 IEEE International Conference on Compact Deep Neural Networks for Device Based Image Classification, Multimedia & Expo Workshops (ICMEW), Turin, Italy, 29 June–3 July 2015; pp. 1–6.
25. Tsung, P.K.; Tsai, S.F.; Pai, A.; Lai, S.J.; Lu, C. High performance deep neural network on low cost mobile GPU. In Proceedings of the 2016 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 7–11 January 2016; pp. 69–70.
26. Russo, L.M.; Pedrino, E.C.; Kato, E.; Roda, V.O. Image convolution processing: A GPU versus FPGA comparison. In Proceedings of the 2012 VIII Southern Conference on Programmable Logic (SPL), Bento Goncalves, Brazil, 20–23 March 2012; pp. 1–6.
27. Zeiler, M.D.; Fergus, R. Stochastic Pooling for Regularization of Deep Convolutional Neural Networks. CoRR. 2013. Available online: http://arxiv.org/abs/1301.3557 (accessed on 20 July 2016).
28. Cong, J.; Xiao, B. Minimizing Computation in Convolutional Neural Networks. In Proceedings of the 24th International Conference onArtificial Neural Networks and Machine Learning—ICANN 2014, Hamburg, Germany, 15–19 September 2014; pp. 281–290.
29. Chellapilla, K.; Puri, S.; Simard, P. High Performance Convolutional Neural Networks for Document Processing. Guy Lorette. In Proceedings of the Tenth International Workshop on Frontiers in Handwriting Recognition, La Baule, France, 23–26 October 2006.
30. Jiang, W.; Chen, Y.; Jin, H.; Luo, B.; Chi, Y. A Novel Fast Approach for Convolutional Networks with Small Filters Based on GPU. In Proceedings of the 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), New York, NY, USA, 24–26 August 2015; pp. 278–283.
31. Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. CoRR. 2015. Available online: http://arxiv.org/abs/1502.03167 (accessed on 21 April 2016).
32. Hara, K.; Saito, D.; Shouno, H. Analysis of function of rectified linear unit used in deep learning. In Proceedings of the 2015 International Joint Conference on Neural Networks (IJCNN), Killarney, Ireland, 12–17 July 2015; pp. 1–8.