JOL VISIBLE
JOINTOPENLAB

Visual Analysis Algorithms for Embedded Systems

Syed Tahir Hussain Rizvi

Supervisor: Prof. Gianpiero Cabodi

Doctoral Program in Computer and Control Engineering
Politecnico di Torino
30th Cycle

TIM

Outline

- Introduction
  - Problem Statement
  - Research Contribution
- Development of a CUDA-Based Proposed Framework
- Optimization of Proposed Framework
- Utilization of Proposed Optimized Framework
- Conclusions and Future Work

Introduction

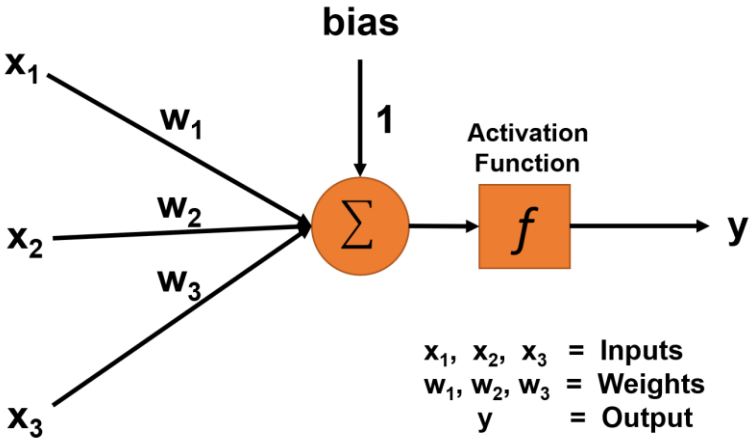
Visual analysis is an ever-growing field with real-time applications reaching out into daily life. Neural networks have recently enjoyed a great success in a range of visual applications.

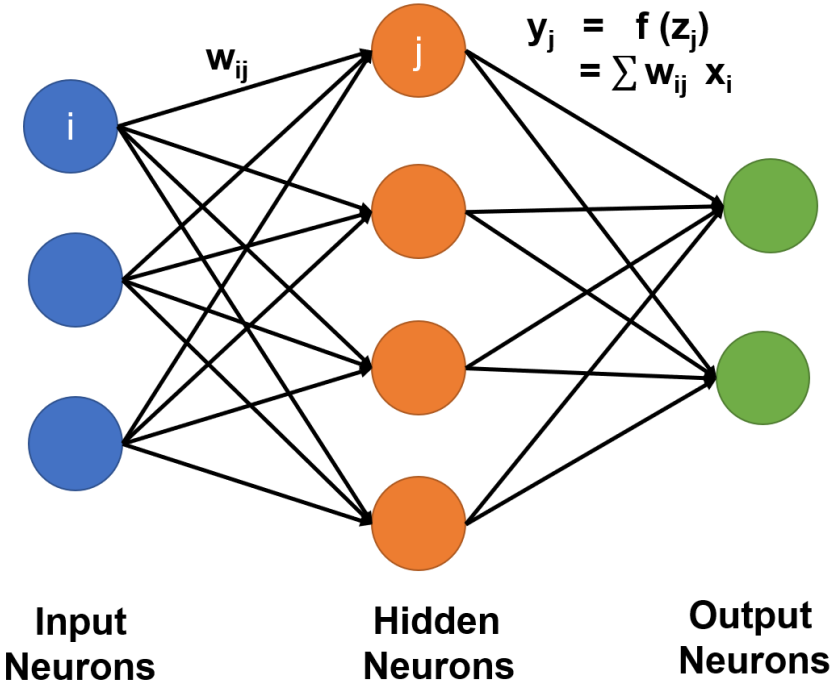
- Neural networks exhibit unmatched recognition and accuracy in classification and segmentation tasks.

- This accuracy mainly comes from the deeper architecture.

Artificial Neural Network (ANN)

A Single Neuron of ANN & Feedforward Neural Network

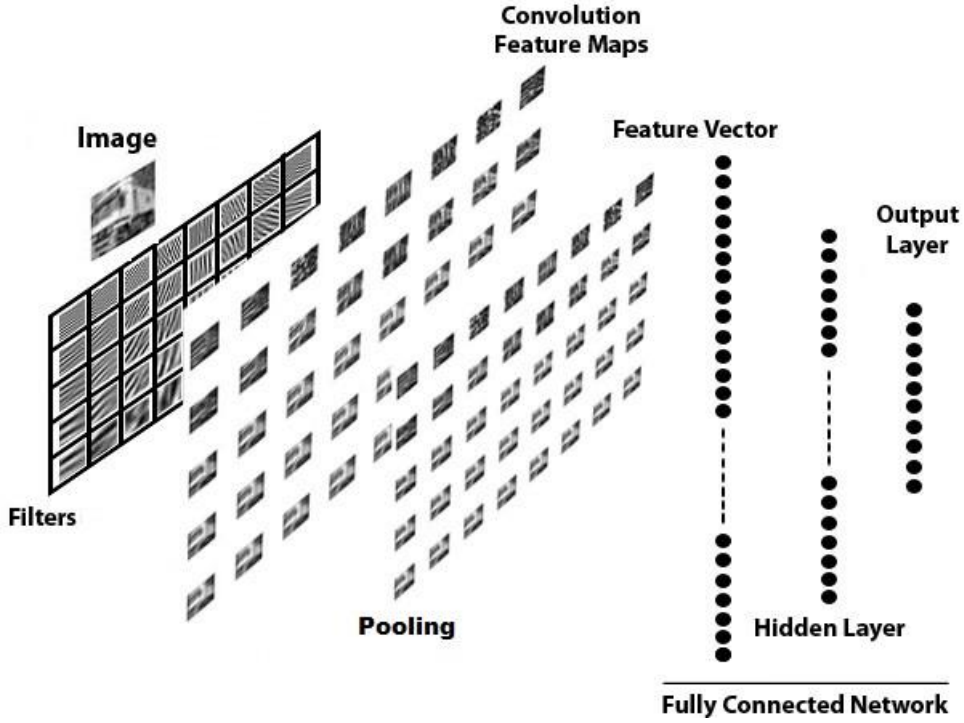




Convolutional Neural Networks (CNN)

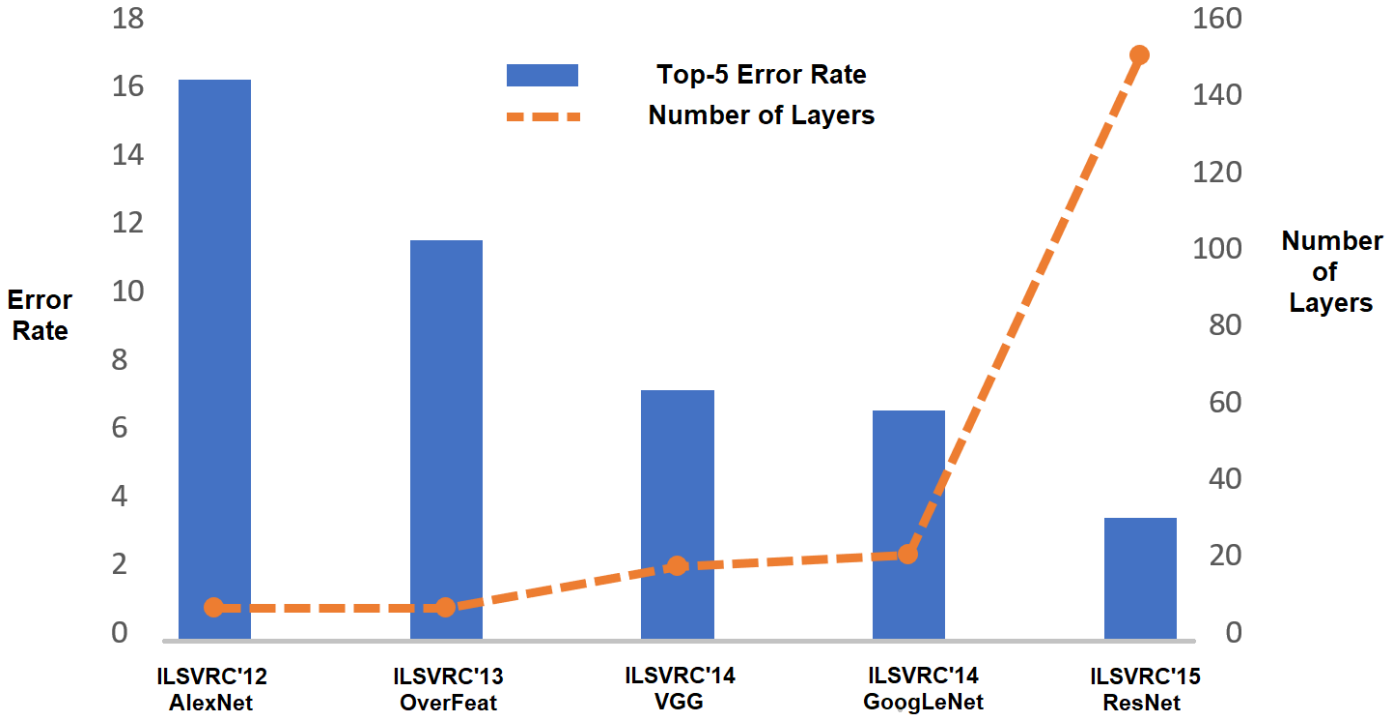
CNN exhibit unmatched performance in classification tasks. It is a special structure having shared weights that help to capture the local properties of the input data or signal.

A typical CNN is composed of two stages. The first stage of CNN is for feature extraction while other is trained for feature classification.



Evolution of Deep Convolutional Neural Networks

ImageNet Large-Scale Visual Recognition Challenge



Problem Statement

- Enlarged depth of Deep Networks also increases the training/classification time, arithmetical complexity and storage requirements.

- Many frameworks are currently available for implementing the neural classifiers. These frameworks exploit the computational power of modern GPUs for faster training and deployment of deep neural networks.

- Embedded devices have limited resources. Due to massive computational and storage requirements, the realization of deep architectures on the embedded platforms is still a challenging problem.

Research Contribution

- **Goal:**
  - To design and develop an optimized framework for realization of trained deep classifier on embedded platforms.
  - To exploit the computational resources of embedded GPU for various visual analysis tasks.
  - To realize the real-time applications on embedded platforms.

Outline

- Introduction
  - Problem Statement
  - Research Contribution
- Development of a CUDA-Based Proposed Framework
- Optimization of Proposed Framework
- Utilization of Proposed Optimized Framework
- Conclusions and Future Work

Image Classification via Convolutional Neural Networks

- **Training Phase**
  - An offline learning state to train the required neural architecture over a set of labelled input data (images).
- **Inference Phase**
  - Testing stage where a proper image classification can be performed using the trained network.

- The requirements of training and inference phases of a deep classifier are slightly different.

Requirements of Training and Inference Stages

- The training phase demands massive amount of resources due to a huge number of input samples, parallel classifications and numerous iterations required for learning.

- A trained network usually needs to classify only the single images in its inference phase using the learned set of parameters (weights and biases).

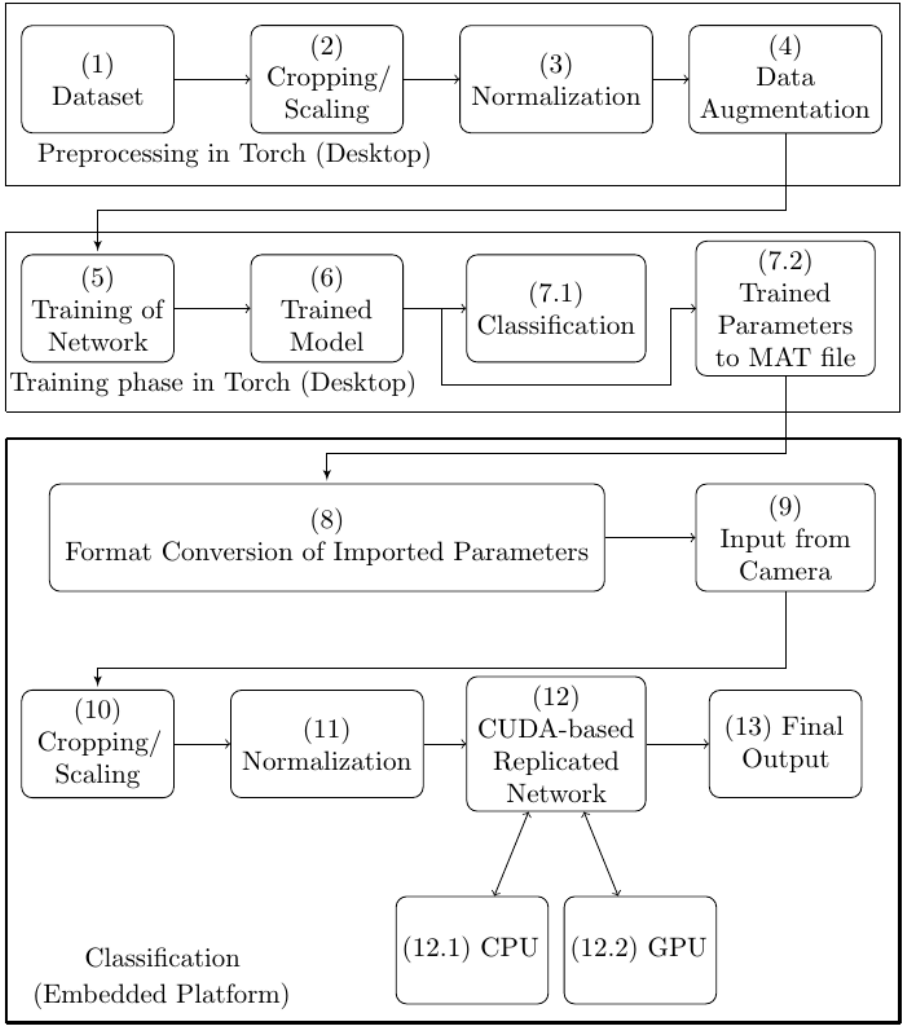
Adopted Methodology

Training (Torch)

- Training of deep classifiers is performed using Torch framework on Desktop workstation or even GPU Server to obtain the trained parameters.

Deep Embedded Classifiers

- CUDA computing language is used to replicate the identical functions and exploit the computational power of embedded GPUs.



Contributions of Proposed Framework

- Contributions of our proposed framework
  - Supports nearly all layer types of deep neural architectures
  - All required layers/functions are realized using CUDA computing framework
  - Accuracy of our proposed scheme is similar to the existing frameworks
  - The framework can be easily integrated into an Android application
  - Provides compatibility for models trained with other frameworks

Architecture of the Convolutional Neural Networks

- **Convolutional Layer**
  - Extracts the meaningful pattern or features from the input images
- **Pooling Layer**
  - Reduces the resolution of feature maps for the manageable representation
- **Linear or Fully Connected Layers**
  - Perform the feature classification and provide the final output
- **Batch Normalization**
  - Speedup the training and preserves the representation ability of neural network
- **Activation Functions (ReLu, Tanh and Threshold)**
  - A non-linear decision-making function that determines the presence of particular features

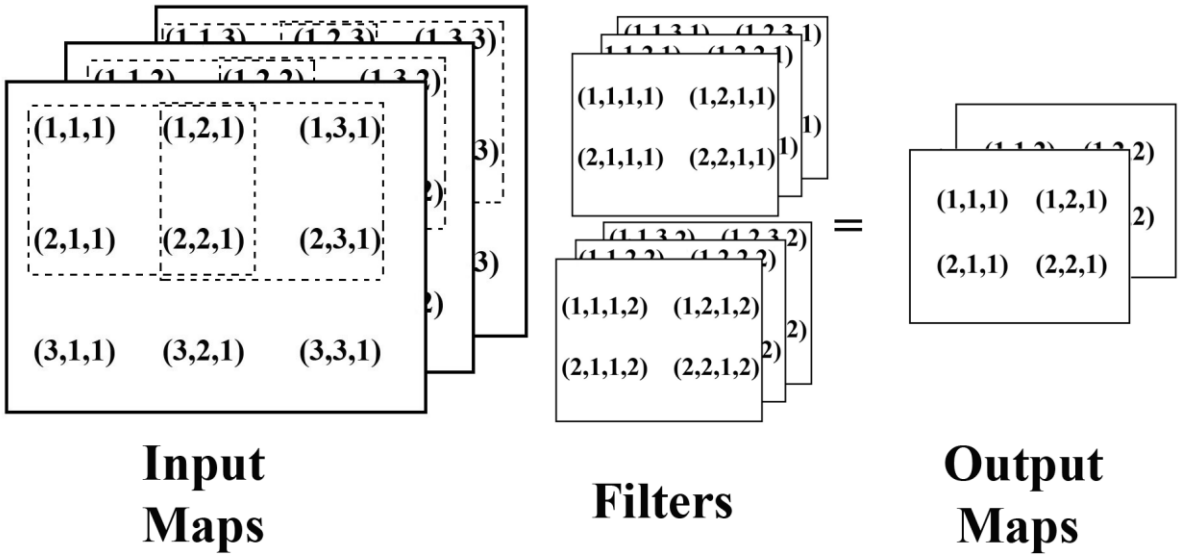
Convolutional Layer

Convolutional Layer is the most important and computationally intensive component of the neural networks. In fact, recent state of the art deep classifiers consist of only convolutional layers.

However, convolution is a computationally intensive operation and demands high computational power and memory storage. Therefore, these factors limit the realization of convolutional neural networks on the embedded platforms.

Multi-Dimensional Full Convolution

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

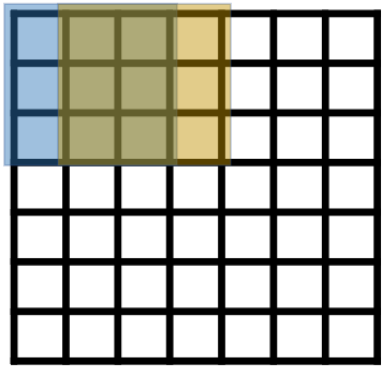


Sizing Convolutional Neural Networks

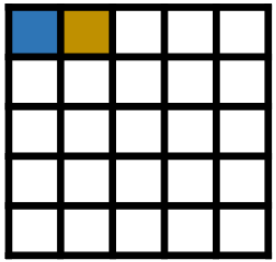
Role of Stride Size and Padding Bits

Size of Input Image = 7 x 7
Size of Filter = 3 x 3

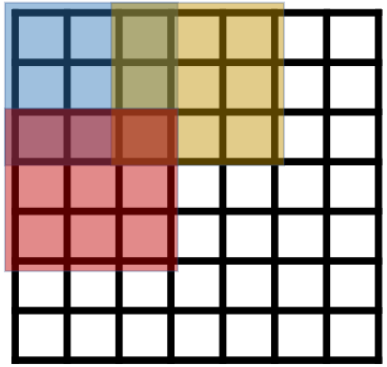
Size of Stride= 1



Size of Output Image = 5 x 5



Size of Stride= 2



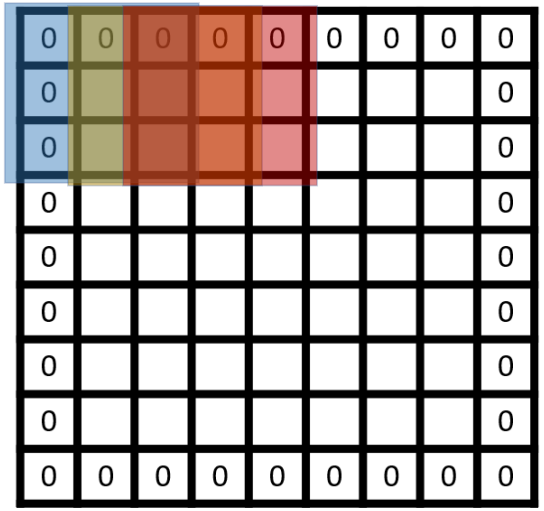
Size of Output Image = 3 x 3

Sizing Convolutional Neural Networks

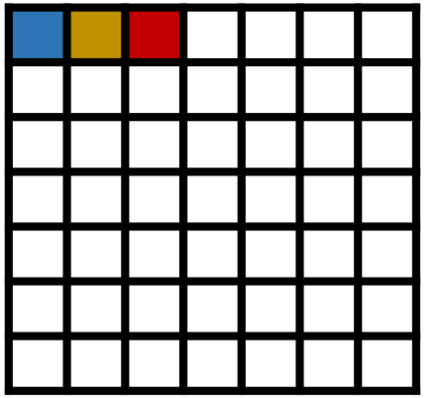
Role of Stride Size and Padding Bits

Size of Input Image = 7 x 7
Size of Filter = 3 x 3
Padding Bits = 1

Size of Stride= 1

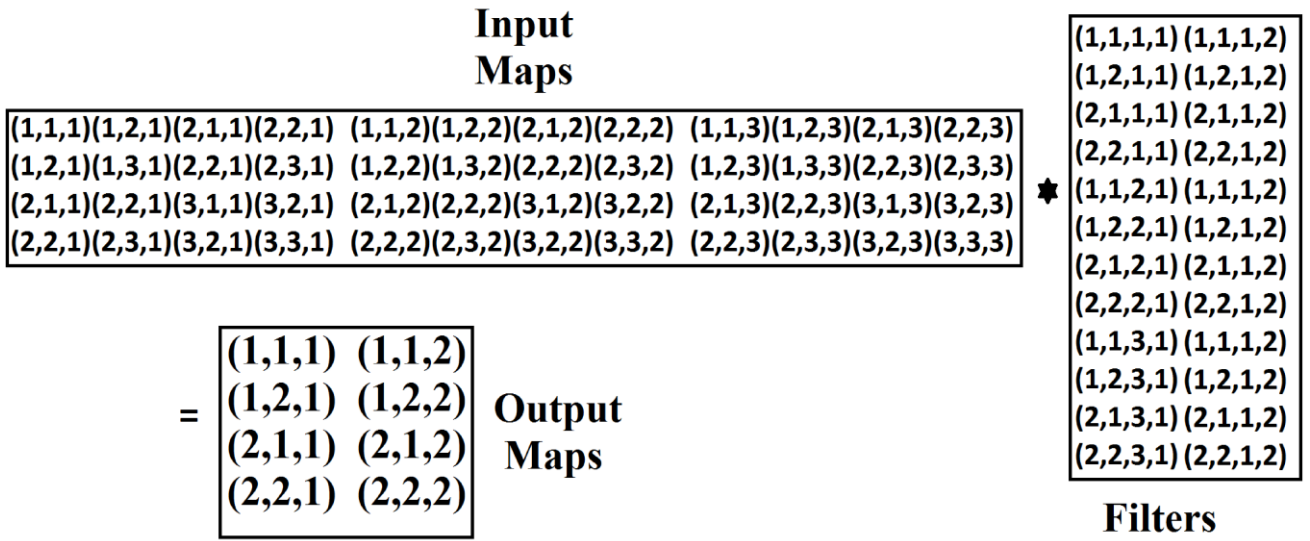


Size of Output Image = 7 x 7

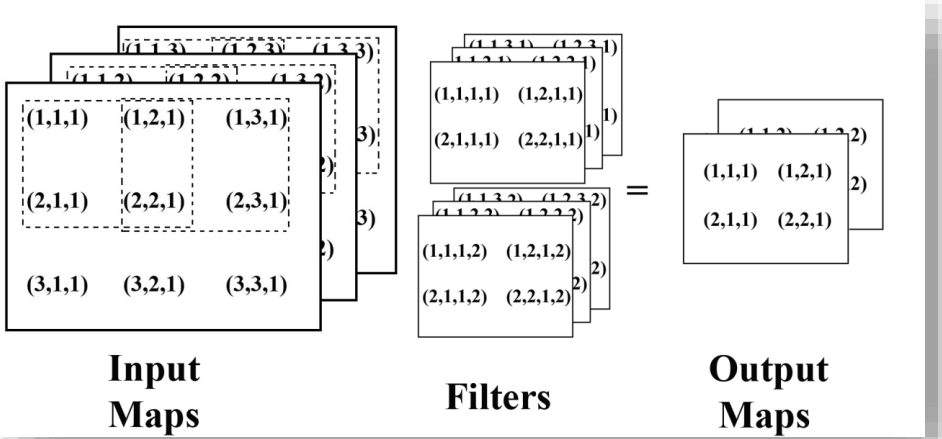


Matrix Multiplication based Convolution (ConvMM)

In this ConvMM approach, first, the input maps and filter banks are transformed into two-dimensional matrices and then directly multiplied



Matrix Multiplication based Convolution (ConvMM)



Input Maps

$$\begin{bmatrix} (1,1,1)(1,2,1)(2,1,1)(2,2,1) & (1,1,2)(1,2,2)(2,1,2)(2,2,2) & (1,1,3)(1,2,3)(2,1,3)(2,2,3) \\ (1,2,1)(1,3,1)(2,2,1)(2,3,1) & (1,2,2)(1,3,2)(2,2,2)(2,3,2) & (1,2,3)(1,3,3)(2,2,3)(2,3,3) \\ (2,1,1)(2,2,1)(3,1,1)(3,2,1) & (2,1,2)(2,2,2)(3,1,2)(3,2,2) & (2,1,3)(2,2,3)(3,1,3)(3,2,3) \\ (2,2,1)(2,3,1)(3,2,1)(3,3,1) & (2,2,2)(2,3,2)(3,2,2)(3,3,2) & (2,2,3)(2,3,3)(3,2,3)(3,3,3) \end{bmatrix} * \begin{bmatrix} (1,1,1,1)\,(1,1,1,2) \\ (1,2,1,1)\,(1,2,1,2) \\ (2,1,1,1)\,(2,1,1,2) \\ (2,2,1,1)\,(2,2,1,2) \\ (1,1,2,1)\,(1,1,1,2) \\ (1,2,2,1)\,(1,2,1,2) \\ (2,1,2,1)\,(2,1,1,2) \\ (2,2,2,1)\,(2,2,1,2) \\ (1,1,3,1)\,(1,1,1,2) \\ (1,2,3,1)\,(1,2,1,2) \\ (2,1,3,1)\,(2,1,1,2) \\ (2,2,3,1)\,(2,2,1,2) \end{bmatrix}$$

Filters

$$= \begin{bmatrix} (1,1,1) & (1,1,2) \\ (1,2,1) & (1,2,2) \\ (2,1,1) & (2,1,2) \\ (2,2,1) & (2,2,2) \end{bmatrix}$$

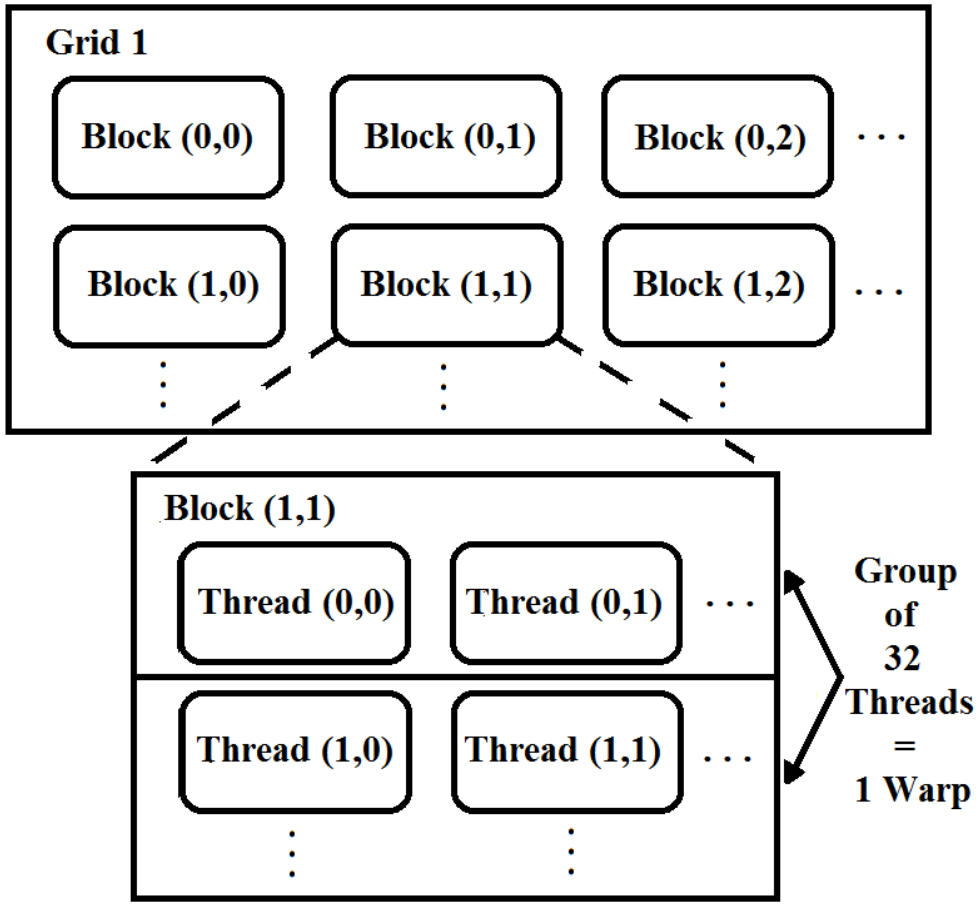
Output Maps

CUDA-based Realized Layers

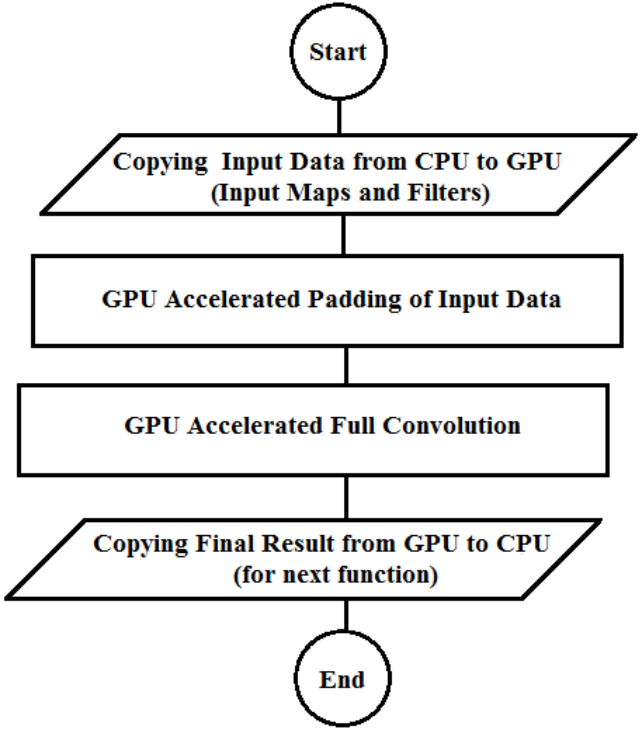
CUDA-based Realized Layers

GPGPU is a multi-core, multi-thread system where threads are organized into a multi-dimensional grid of independent blocks. Each block is comprised of concurrent threads that operate cooperatively within each block.

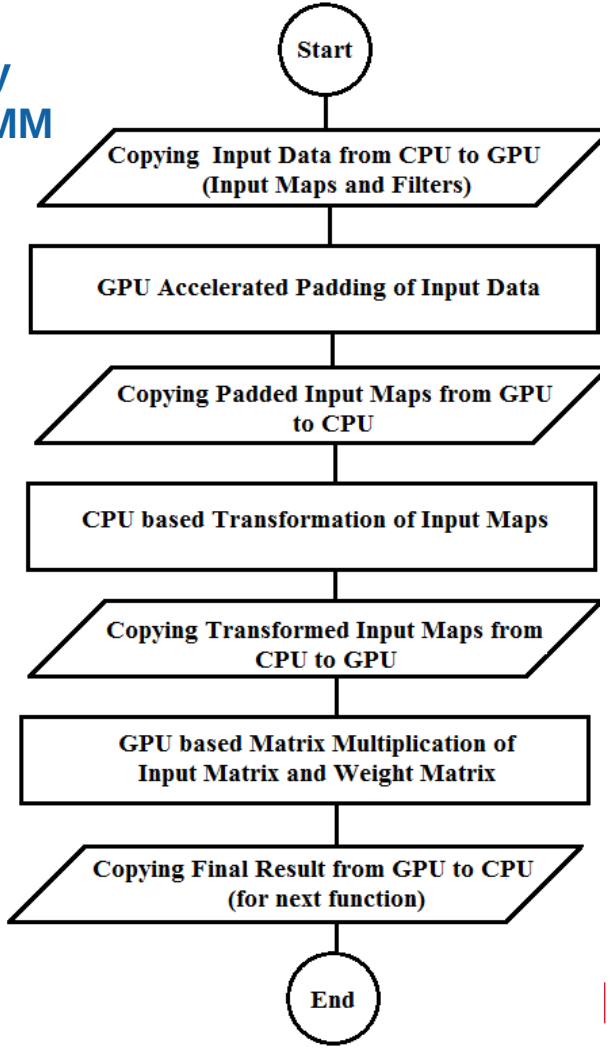
All layers are implemented and accelerated using the three-dimensional grid of parallel blocks and threads



GPGPU-accelerated Fully Convolutional Layer



Heterogeneously Accelerated ConvMM Layer



GPU-only ConvMM Layer

$$Input\_matrix_{(row,col)} =$$

$$\sum_{i=1}^{o\_Row} \sum_{j=1}^{o\_Col} \left(\sum_{k=1}^{i\_Dim} \sum_{h=1}^{k\_h} \sum_{w=1}^{k\_w} Input\_map_{(i+h,j+w,k)} \right)$$

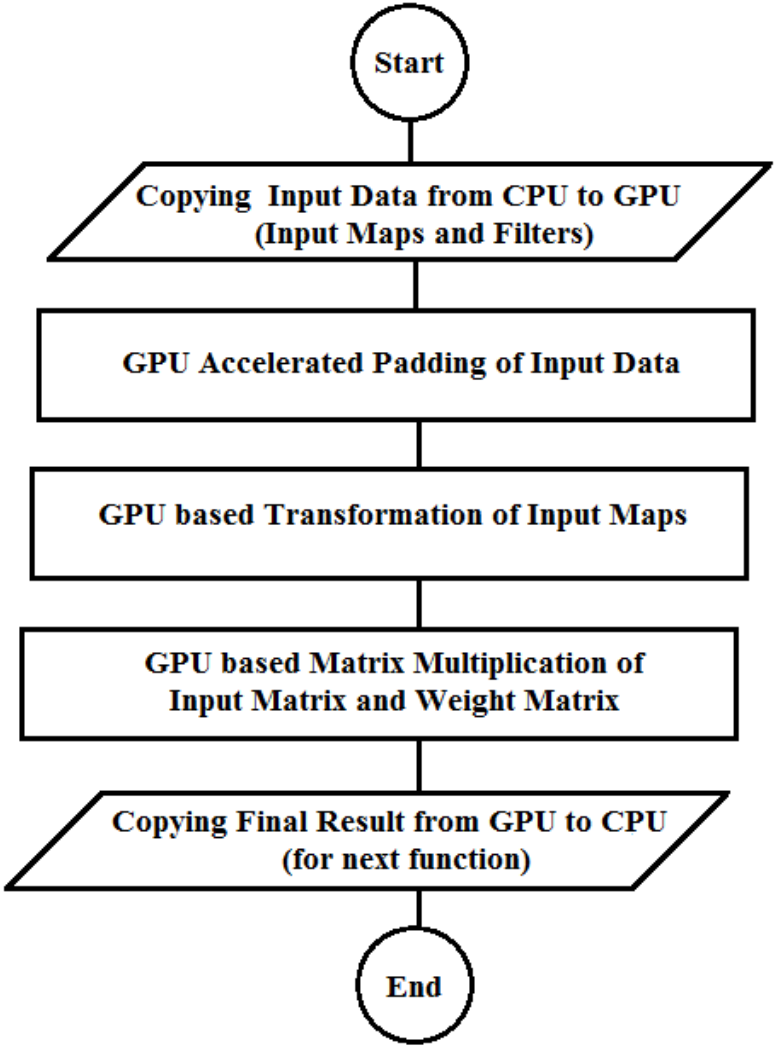
where

$$o\_Row = i\_Row - k\_h + 1 ;$$

$$o\_Col = i\_Col - k\_w + 1;$$

$$row = i \times (o\_Col) + j ;$$

$$col = (k \times k\_w \times k\_h + (h \times k\_w) + w);$$



Neural Network Architectures implemented in CUDA

Three different deep classifiers are implemented using proposed CUDA-based scheme. Torch computing framework is used to construct and train these classifiers. CIFAR-10 (Canadian Institute For Advanced Research 10) dataset is used to train the Alex's CIFAR-10 network, while other two networks are trained on ImageNet dataset.

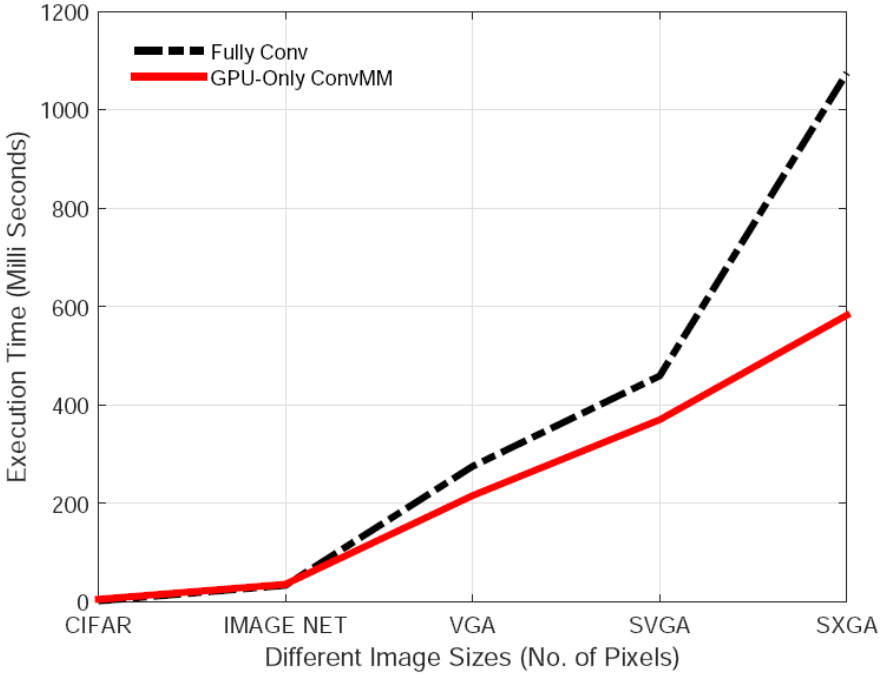
Model	No. of Layers	Required Functions
Alex's CIFAR-10	5	Conv. + tanh + Max Pool
OverFeat	8	Conv. + ReLu + Max Pool
ResNet-34	34	Conv. + Max Pool + Batch Normalization + ReLu + Avg.Pooling

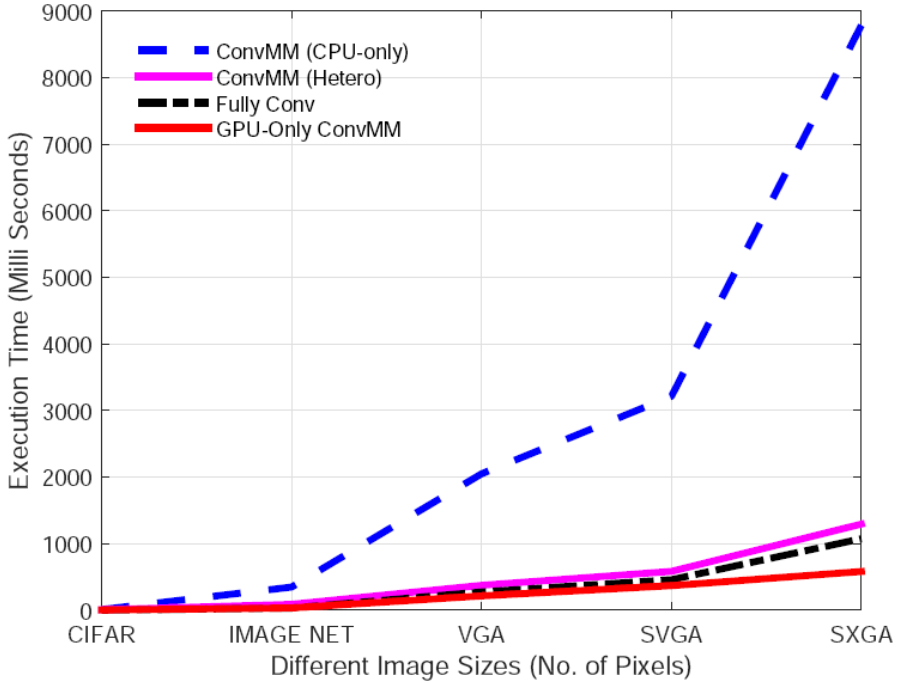
Experiments and Results

- **Embedded Platforms**
  - **Nvidia Jetson TX1 embedded board**
    - Quad-core ARM Cortex A57 CPU, Nvidia Maxwell GPU with 256 CUDA cores and 4 GB of RAM memory
  - **Nvidia Shield K1 tablet**
    - Quad-core ARM Cortex A53 CPU, Nvidia Kelper GPU with 192 CUDA cores and 2 GB of RAM memory

Performance Evaluation on Jetson TX1 Board

Comparison of different convolution layers under various computational loads

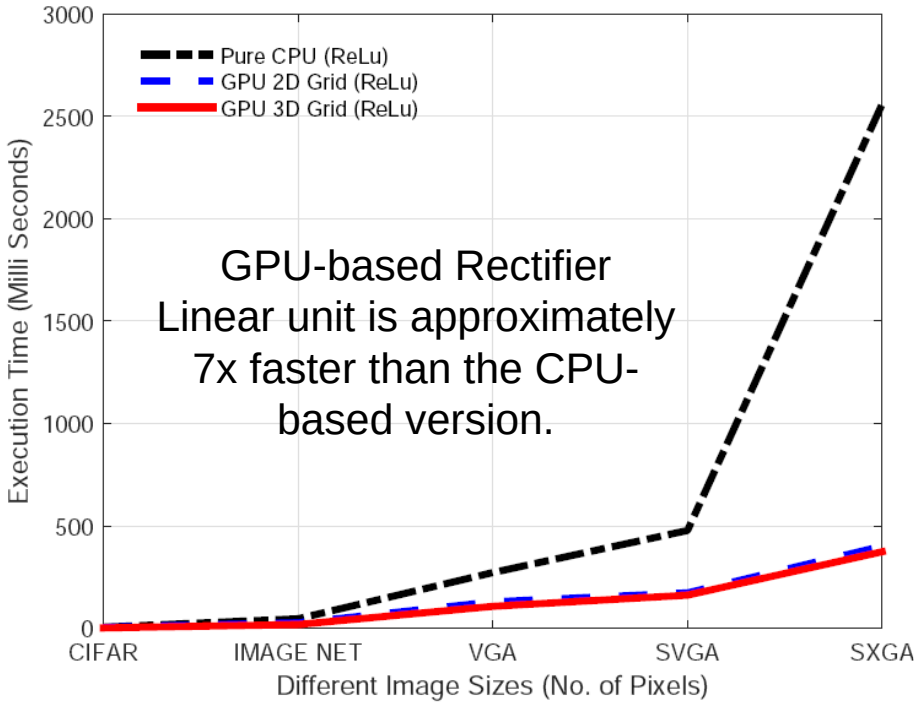


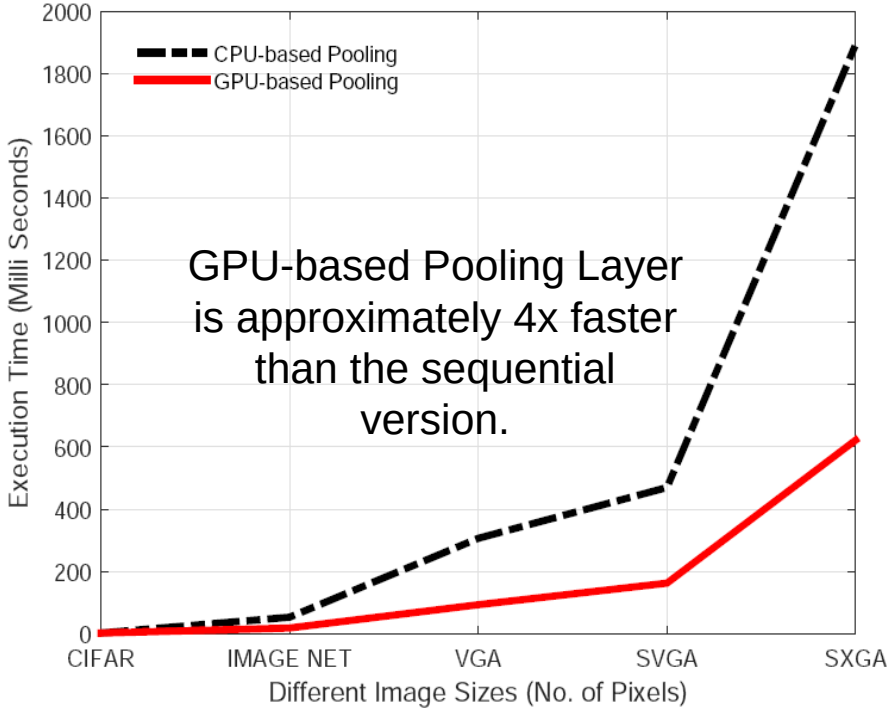


GPU-only ConvMM layer is 2x faster than the heterogeneous approach and 10x faster than the CPU-only ConvMM layer

Performance Evaluation on Jetson TX1 Board

Comparison of realized layers under various computational loads





Performance Evaluation on Jetson TX1 Board

Classification time of deep models on Jetson TX1 board

Model	Layers	ConvMM (CPU-only)	Fully Conv (GPU)	Hetero ConvMM (CPU+GPU)	GPU-only ConvMM	Speedup over Sequential
		Double				
		(Milliseconds)				
Alex's CIFAR-10	5	7814.25	149.43	129.90	**107.03**	73×
OverFeat	8	254,285.12	3250.57	2492.94	**1924.41**	132×
ResNet-34	34	190,054.39	2838.01	2361.18	**1217.33**	156×

Double to Single Precision Conversion

As Embedded devices have limited storage capacity, so the sizes of trained models, frameworks (like Torch), computational packages and their operating system can affect the performance of the required network.

Performance of realized deep classifiers can be further improved using single precision data storage and arithmetic. This conversion does not affect the accuracy of the trained networks.

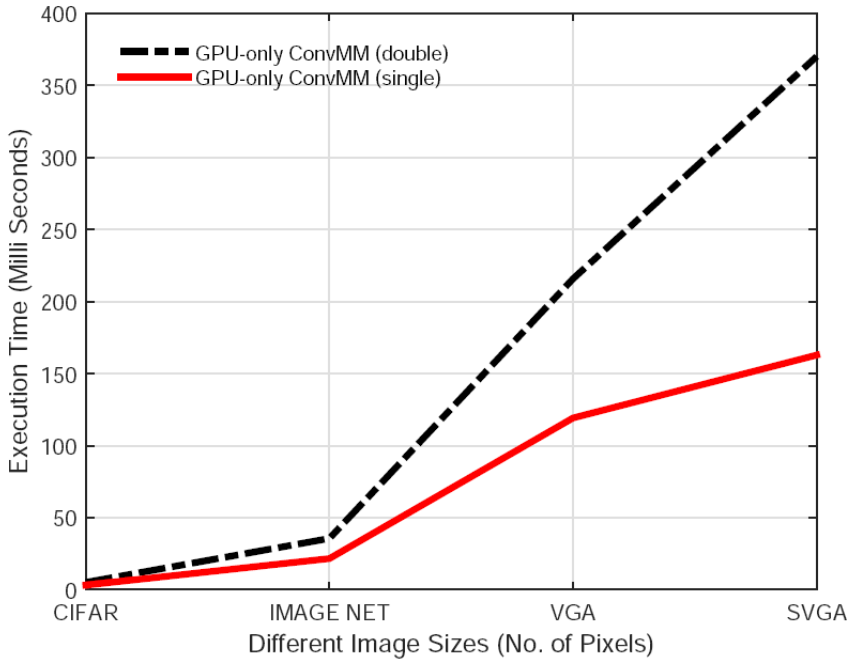
Model	Size of File
	Double
Alex's CIFAR-10	41 MB
OverFeat	1190 MB
ResNet-34	215 MB

The differences introduced by low-precision representations are well within the tolerances a neural network has learned to deal with.

Performance Evaluation on Jetson TX1 Board

Comparison of double- and single-precision ConvMM layers

Image Size	GPU-only ConvMM Double	GPU-only ConvMM Single
	(Milliseconds)	
CIFAR (32 × 32 × 16)	5.12	**3.20**
ImageNet (224 × 224× 16)	35.61	**21.63**
VGA (640 × 480 × 16)	215.74	**119.17**
SVGA (800 × 600 × 16)	369.91	**162.80**



Performance Evaluation on Jetson TX1 Board

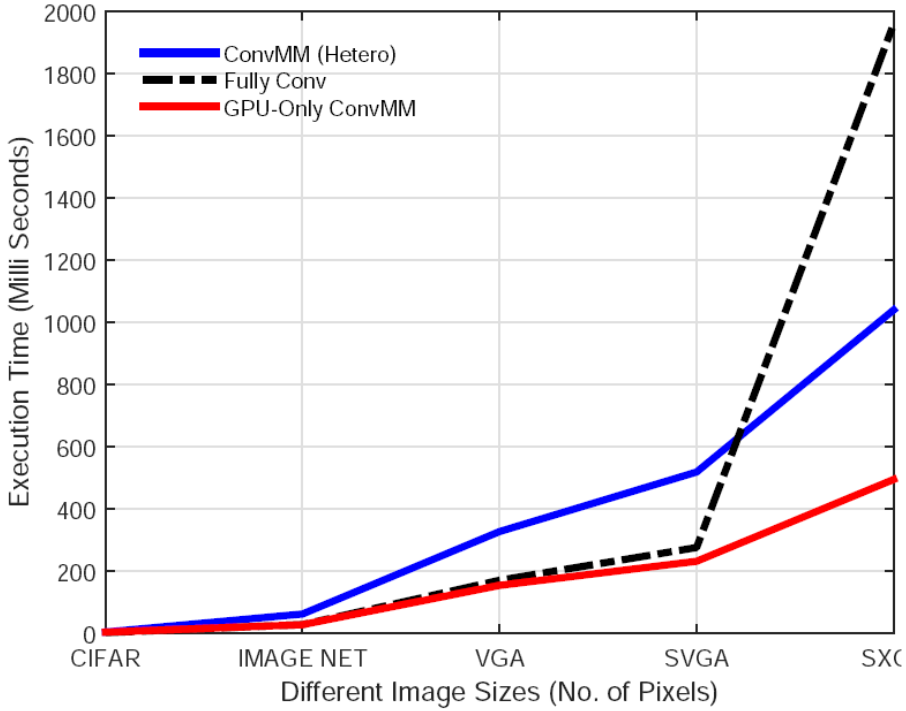
Classification time of double- and single-precision deep models

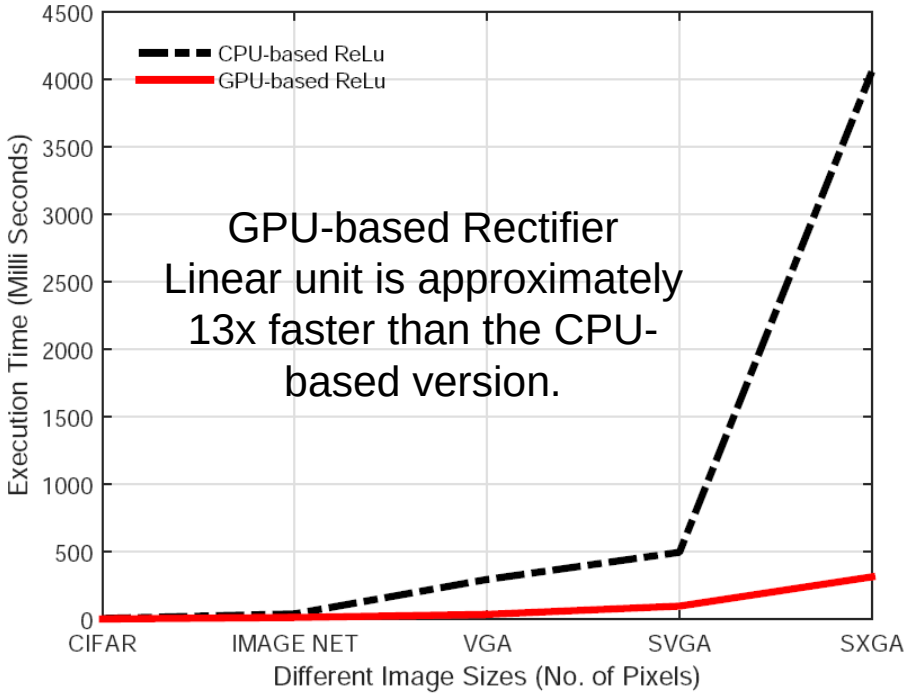
Model	Layers	GPU-only ConvMM Double (Milliseconds)	GPU-only ConvMM Single (Milliseconds)
Alex's CIFAR-10	5	107.03	**76.36**
OverFeat	8	1924.41	**1212.74**
ResNet-34	34	1217.33	**887.53**

The single-precision models are significantly faster than the original versions. This conversion is also profitable in terms of storage requirement that is an important constraint for the embedded devices.

Performance Evaluation on Mobile Shield K1 Tablet

Comparison of different versions of convolution and ReLu operation

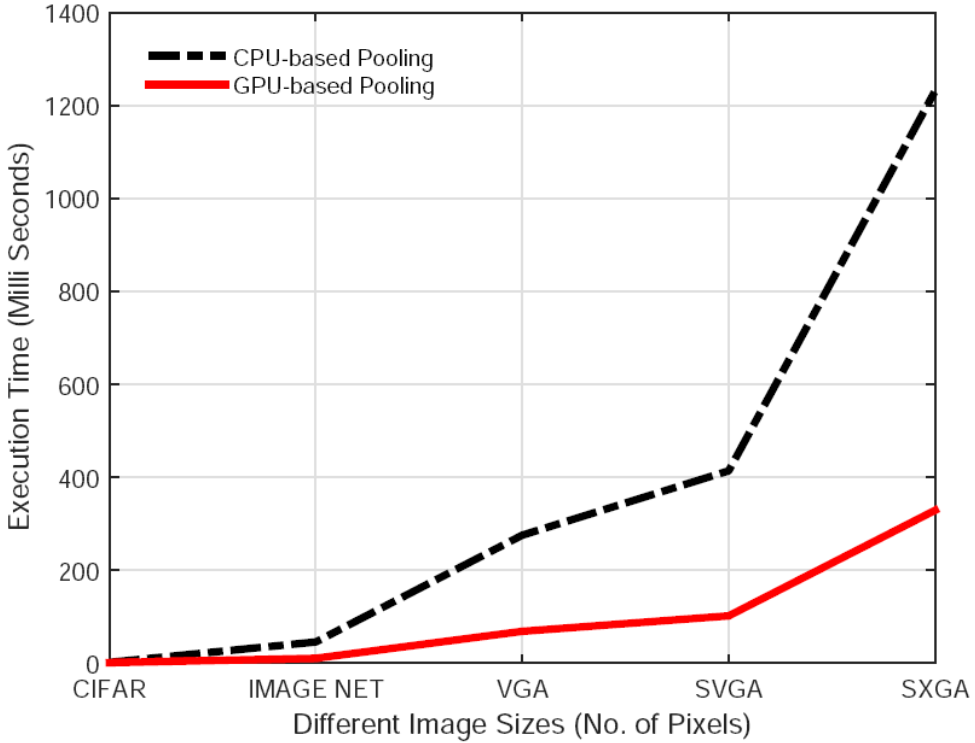




Performance Evaluation on Mobile Shield K1 Tablet

Performance comparison of pooling layers

Image Size	CPU	GPU
	Single	
	(Milliseconds)	
CIFAR (32 × 32 × 16)	1.83	**1.10**
ImageNet (224 × 224× 16)	45.72	**10.57**
VGA (640 × 480 × 16)	275.43	**68.58**
SVGA (800 × 600 × 16)	413.85	**101.83**
SXGA (1280 ×1024 × 16)	1231.20	**329.03**



Performance Evaluation on Mobile Shield K1 Tablet

Classification time and Energy consumed (joule) by deep models

Model	Layers	ConvMM (CPU-only)	Fully Conv (GPU)	Hetero ConvMM (CPU+GPU)	GPU-only ConvMM	Speedup over Sequential
		Single (Milliseconds)				
Alex's CIFAR-10	5	10,449.21	584.75	419.67	**144.23**	73×
OverFeat	8	440,631.27	12,582.54	5792.54	**3899.43**	113×
ResNet-34	34	252,955.27	11,481.74	3319.34	**2545.83**	99×

Model	Sequential ConvMM	Fully Convolution	GPU-only ConvMM	Improvements over Sequential
Alex's CIFAR-10	16.0	0.430	**0.373**	43×
OverFeat	850.8	13.2	**2.13**	399×
ResNet-34	480.0	2.5	**1.2**	400×

[1]

Rizvi, S.T.H.; Cabodi, G.; Patti, D.; Francini, G. GPGPU Accelerated Deep Object Classification on a Heterogeneous Mobile Platform. ***Electronics*** **2016, 5, 88.**

Outline

- Introduction
  - Problem Statement
  - Research Contribution
- Development of a CUDA-Based Proposed Framework
- Optimization of Proposed Framework
- Utilization of Proposed Optimized Framework
- Conclusions and Future Work

Optimization of Deep Neural Classifiers for Embedded GPUs

A set of techniques are employed and hardware capabilities of embedded GPUs are exploited to improve the performance of deep neural networks, making them applicable to embedded applications.

- Optimized Data Transfer Scheme
- Hardware-Dependent Matrix Multiplication Approach

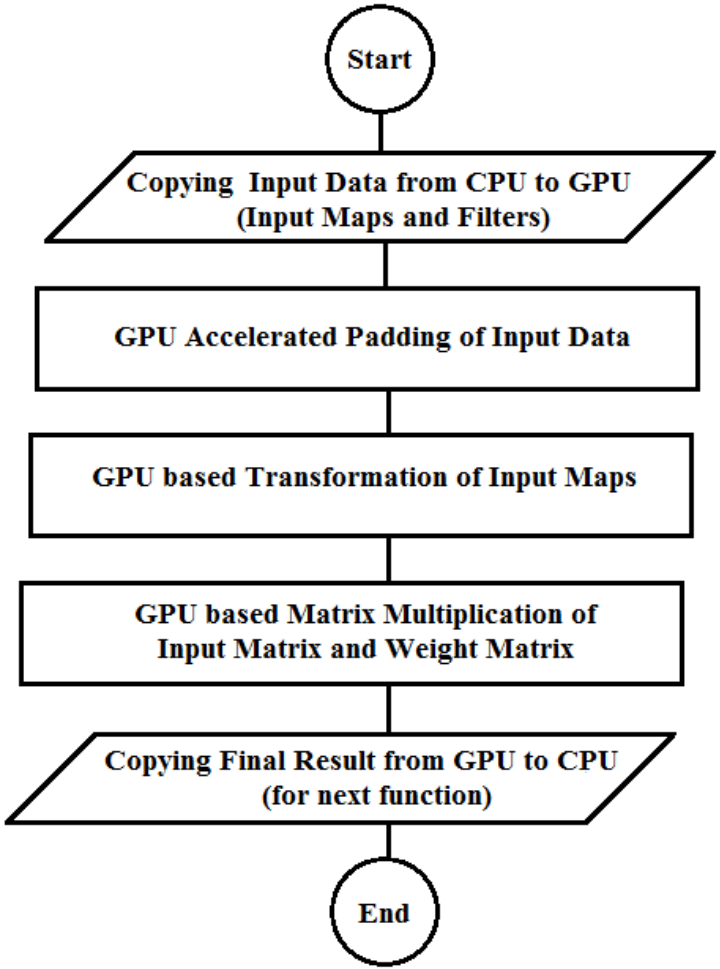
Optimized Data Transfer Scheme

Data Transfer Schemes

GPU-only ConvMM Layer

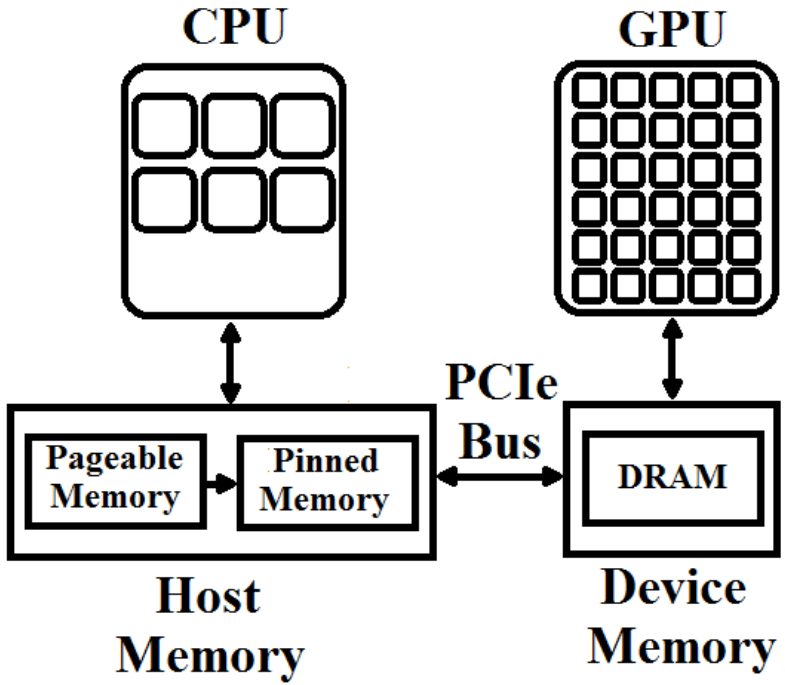
- The input data to be processed must be copied from the host (CPU) memory to the device (GPU) memory and the output data must be retrieved from the GPU memory to the CPU memory.

- These memory-transfers may be required several times depending on the architecture of a neural network and can affect the performance of the application.



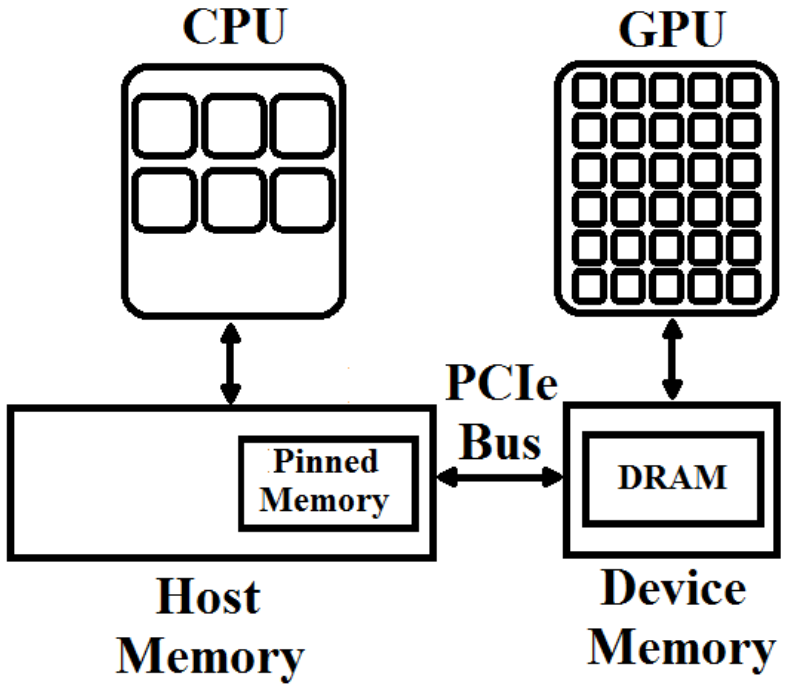
Data Allocations on the Host Side

- Host allocations are pageable by default, and the GPU cannot get data directly from the pageable memory.
- The pinned memory acts as a staging area between the host and device memories.



Data Allocations using Pinned Memory

- CUDA provides the feature to directly allocate the host data into the pinned memory.
- It is a common practice to utilize the pinned memory for acceleration of GPU based desktop and server frameworks. However, some constraints need to be considered before using pinned memory on an embedded platform.



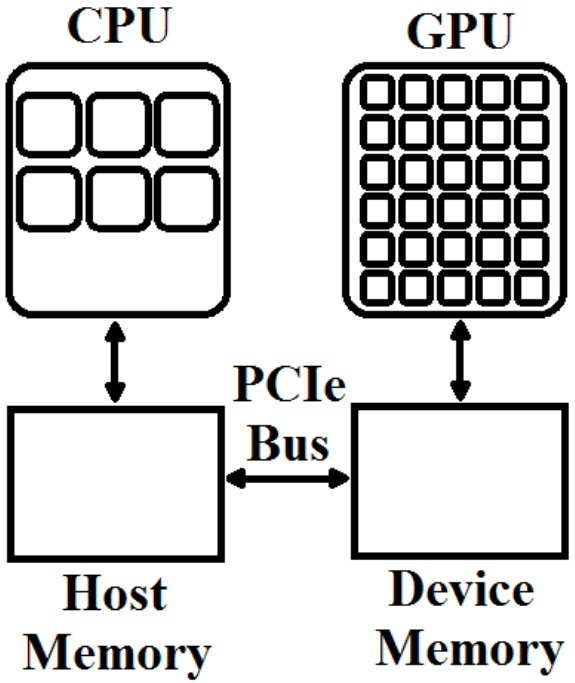
Data Allocations using Pinned Memory

- Pinned memory should not be over-allocated because it can break the performance of the used system by reducing the amount of physical memory (Random Access Memory) to its operating system and other programs.

- Unlike pageable memory, pinned memory cannot be paged out by the Operating system.

Physically separate Host and Device memories

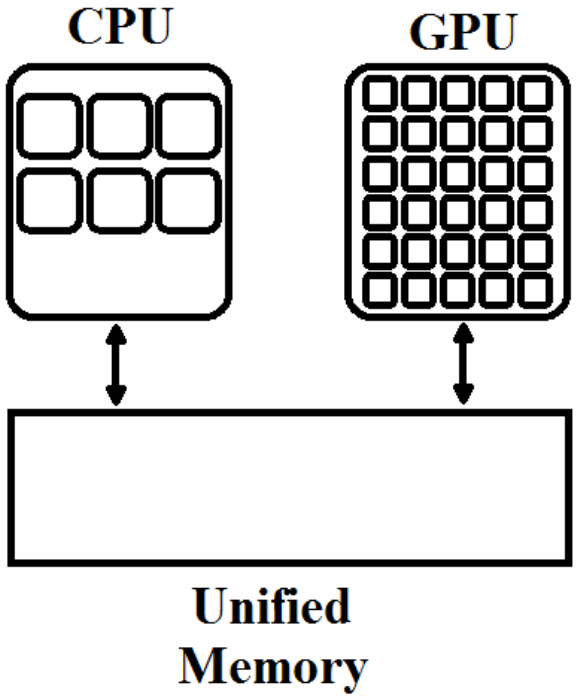
- In a typical desktop workstation or GPU server, host (CPU) and device (GPU) memories are physically separated by the PCIe bus.
- Data must be allocated in both memories and explicitly transferred between them. These data transfers are very frequent, and speed of these transactions is dependent on the PCIe bus.



Proposed flow using Unified Memory-based Allocations

- Unified memory creates a pool of shared memory between the host (CPU) and device (GPU). This shared memory is accessible to both host (CPU) and device (GPU).

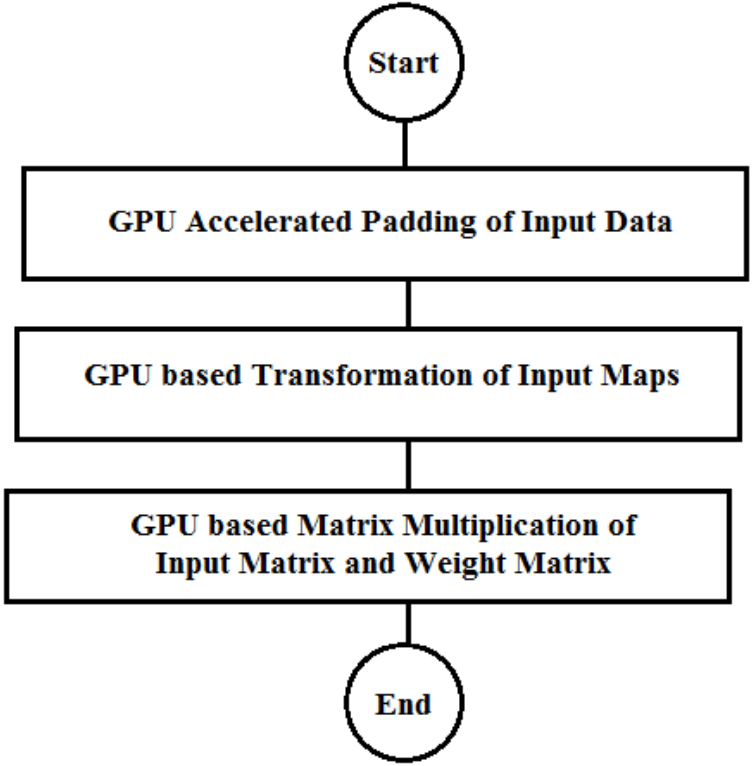
- By migrating the data to a shared pool, unified memory offers the performance of local data on the GPU and can enhance the performance of an algorithm.



GPU-only Unified ConvMM Layer

- In a typical CPU-GPU environment, two copies of input data were required in both CPU and GPU memories. However, by using unified memory scheme, only a single copy of data elements is needed that can be allocated to a shared pool of memory.

- Using unified memory, the flow of all required layers and functions is simplified and optimized.



[2]

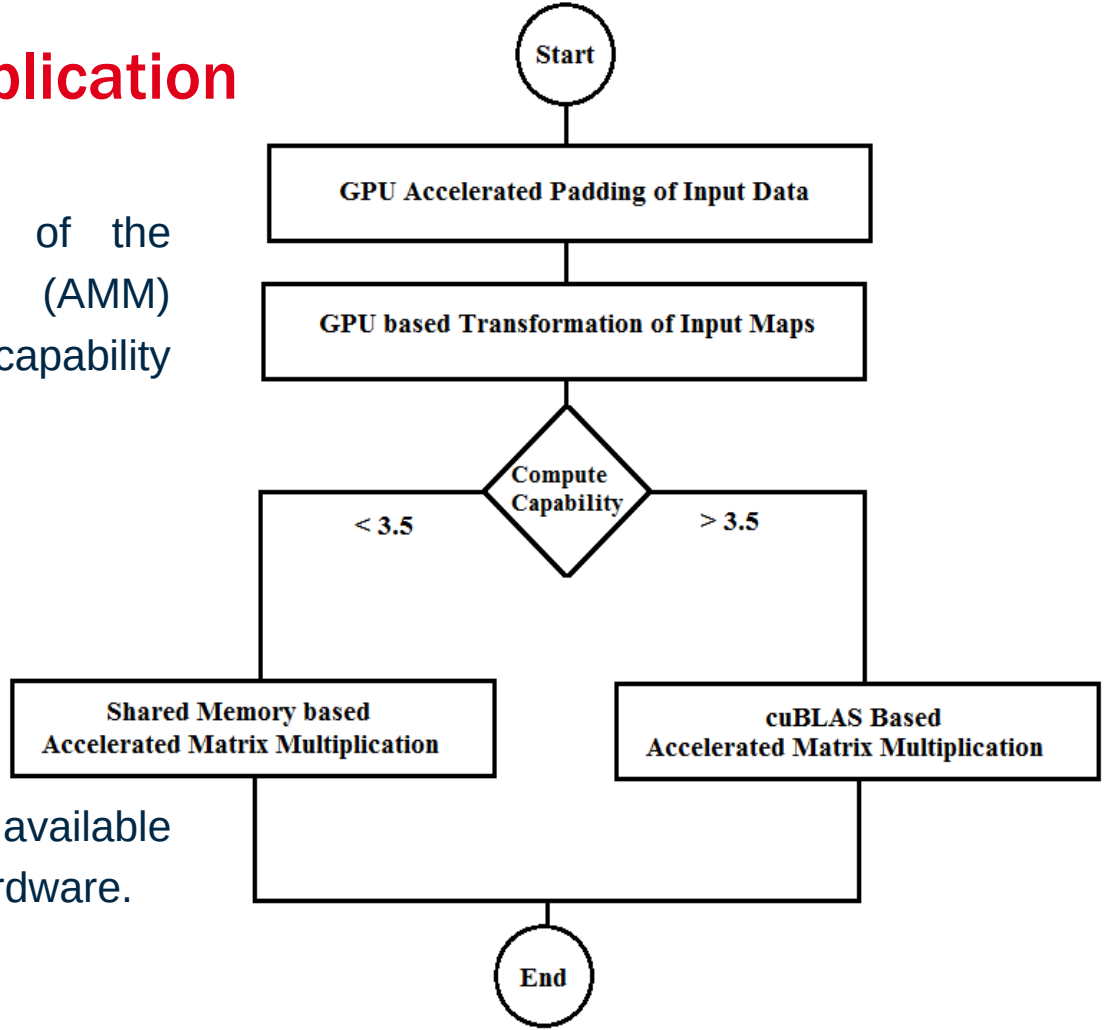
Rizvi, S. T. H.; Cabodi G.; Francini, G. GPU-only Unified ConvMM Layer for Neural Classifiers. In Proceedings of 2017 International Conference on Control, Decision and Information Technologies (CoDIT'17), Barcelona, Spain, April 2017.

Hardware-Dependent Matrix Multiplication Approach

Accelerated Matrix Multiplication

- Proposed framework selects one of the accelerated matrix multiplication (AMM) scheme depending on the compute capability of the used embedded device.

- Compute capability (C.C) defines the available features and restrictions of a GPU hardware.



CUDA Basic Linear Algebra Subroutines

(cuBLAS)

- The CUDA toolkit includes a set of high-performance libraries which provide standard mathematical functions. **CUDA Basic Linear Algebra Subroutines** (cuBLAS) is one of them. The cuBLAS library provides the various GPU-accelerated subroutines and shows tremendous speedup over other available libraries.

- This library also provides a wide range of General Matrix-Matrix Multiply (GEMM) subroutines that support different type of operands.

cuBLAS-Accelerated Matrix Multiplication Convolution

Unified ConvCAMM Layer

- The performance of convolution operation can also be improved using **cuBLAS-Accelerated Matrix Multiplication** (CAMM), but some important aspects need to be taken into consideration before using this library.
  - Compute capability (C.C) of the used GPU platform
  - The storage layout of the cuBLAS library

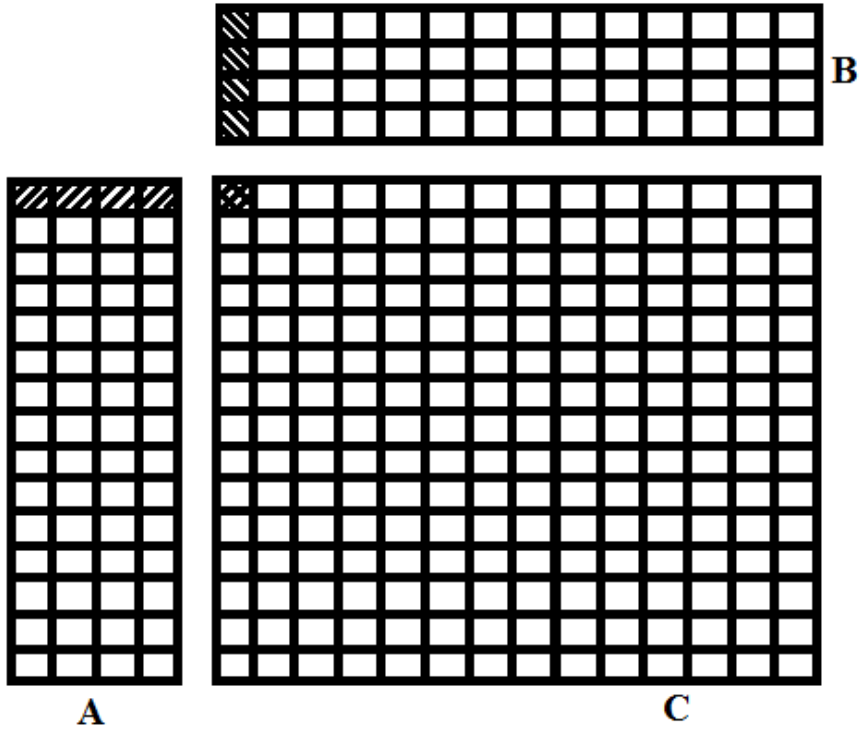
- In the proposed framework, the **GEMM** subroutine of the cuBLAS library is employed to accelerate the matrix multiplication of convolutional layer (for GPU platforms having C.C greater than 3.5).

Shared Memory-based Matrix Multiplication Convolution

Unified ConvSAMM Layer

- For devices having compute capability less than 3.5, **Shared memory based matrix multiplication** (SAMM) is proposed to accelerate the GPU-only Unified convolutional layer.

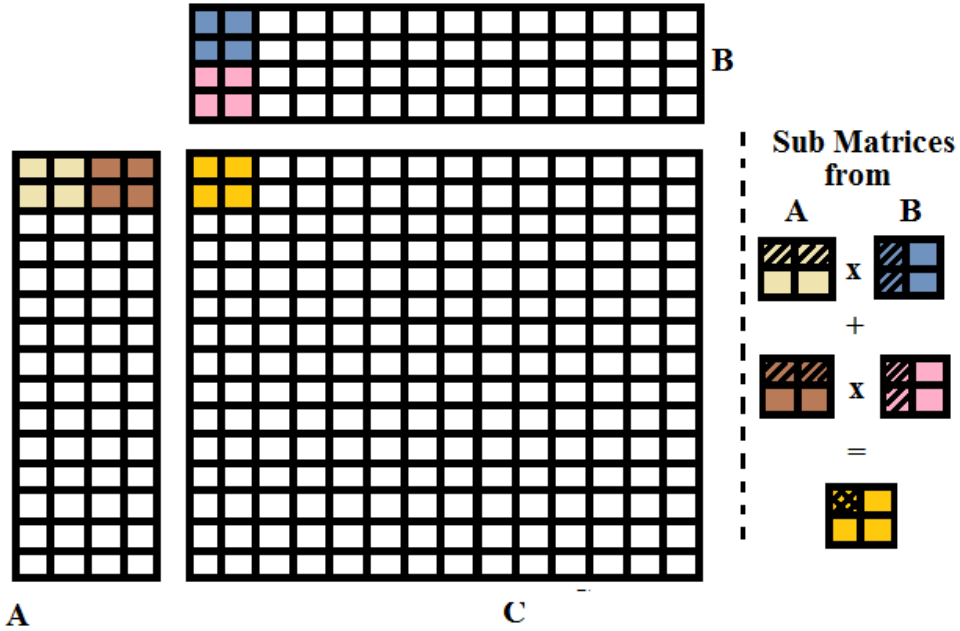
- In naive matrix multiplication approach, there are many redundant global memory accesses that can be reduced.



Shared Memory-based Matrix Multiplication Convolution

Unified ConvSAMM Layer

- By dividing the global data into tiles or blocks, these subsets can be copied into shared memory to reduce the global access latency and achieve the memory level parallelism.
- Data residing in shared memory can be accessed faster in a concurrent manner.



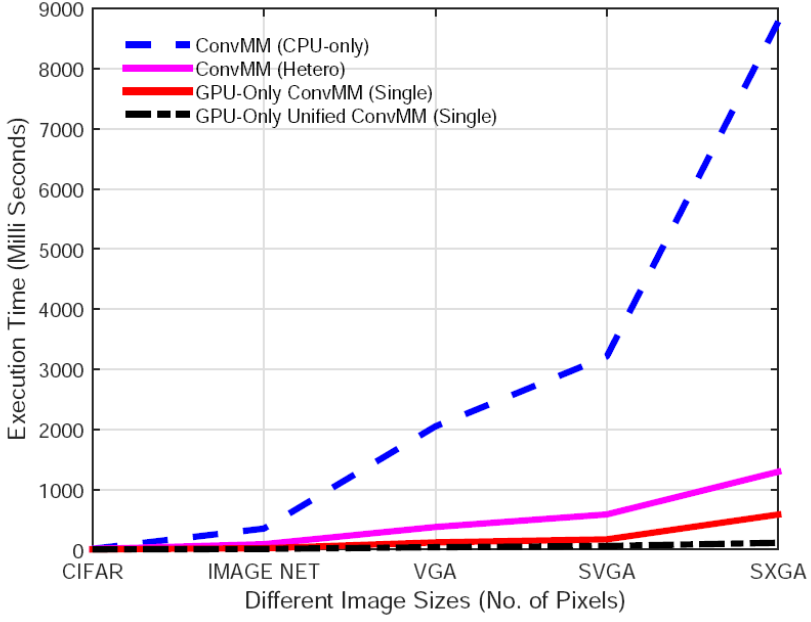
Experiments and Results

- **Classification Networks**
  - Alex Krizhevsky's network for CIFAR-10
  - OverFeat model
  - ResNet-34
- **Embedded Platforms**
  - Nvidia Jetson TX1 embedded board (Compute capability 5.3)
  - Nvidia Shield K1 tablet (Compute capability 3.2)

Performance Evaluation

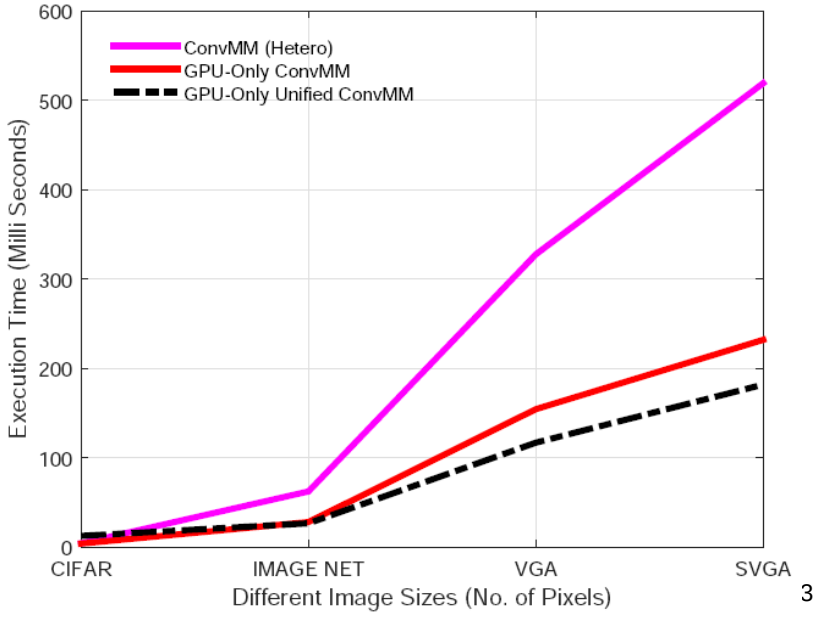
Jetson TX1 Embedded Board

GPU-only Unified ConvMM layer is 5× faster than the GPU-only version and is approximately 12× faster than the heterogeneous ConvMM.



Nvidia Shield K1 Tablet

For more extensive workloads, GPU-only Unified ConvMM layer is significantly faster than all other ConvMM layers.



Jetson TX1 Embedded Board

Model	Layers	ConvMM (CPU-only)	Hetero ConvMM (CPU+GPU)	GPU-only ConvMM	ConvCAMM	Unified ConvCAMM
		Double			Single	
		(Milliseconds)				
Alex's CIFAR-10	5	7814.25	129.90	76.36	33.53	**19.13**
OverFeat	8	254,285.12	2492.94	1212.74	496.83	**122.90**
ResNet-34	34	190,054.39	2361.18	887.53	577.66	**423.86**

Model	Layers	Speedup over	
		Sequential	Hetero
Alex's CIFAR-10	5	408×	7×
OverFeat	8	2069×	20×
ResNet-34	34	448×	6×

Nvidia Shield K1 Tablet

Model	Layers	ConvMM (CPU-only)	Hetero ConvMM (CPU+GPU)	GPU-only ConvMM	ConvSAMM	Unified ConvSAMM
		Single				
		(Milliseconds)				
Alex's CIFAR-10	5	10,449.21	419.67	144.23	74.20	**38.57**
OverFeat	8	440,631.27	5792.54	3899.43	1784.24	**632.91**
ResNet-34	34	252,955.27	3319.34	2545.83	1210.16	**767.53**

Model	Layers	Speedup over	
		Sequential	Hetero
Alex's CIFAR-10	5	271×	11×
OverFeat	8	696×	9×
ResNet-34	34	330×	4×

[3]

Rizvi, S.T.H.; Cabodi, G.; Francini, G. Optimized Deep Neural Networks for Real-Time Object Classification on Embedded GPUs. Appl. Sci. 2017, 7, 826.

Power Consumption

Nvidia Shield K1 Tablet

Model	Layers	ConvMM (CPU-only)	Hetero ConvMM (CPU+GPU)	Unified ConvSAMM	Efficiency over Sequential	Efficiency over Hetero
Alex's CIFAR-10	5	16.0	0.410	**0.202**	79×	2×
OverFeat	8	850.8	3.2	**0.529**	1608×	6×
ResNet-34	34	480.0	1.8	**0.432**	1111×	4×

Results show that the proposed Unified ConvSAMM layer is consuming up to 75% less energy than the heterogeneous approach. It validates that the proposed scheme significantly reduces the energy consumption per image frame.

Comparison with Torch Framework

Jetson TX1 Embedded Board

Model	Layers	Input Image Size	Proposed Unified ConvCAMM Flow (Milliseconds)	Torch (cuDNN) (Milliseconds)
Alex's CIFAR-10	5	(1,3,32,32)	19.13	37.11
OverFeat	8	(1,3,231,231)	122.9	140.78
ResNet-34	34	(1,3,224,224)	423.86	102.10

The results show the comparable performance of proposed scheme and torch based library for OverFeat and Alex's CIFAR-10 networks, while the proposed scheme is slower than the cuDNN library for ResNet-34 architecture having small filters.

Comparison with Torch Framework

Jetson TX1 Embedded Board

Model	Layers	Trained Parameters (float) (Megabytes)	Torch
Alex's CIFAR-10	5	19	41
OverFeat	8	556	1190
ResNet-34	34	83	215

The installation of computing framework, its libraries (in this particular case, 828 MB is consumed by the Torch and its packages on Jetson TX1 embedded board) and a complete model with trained parameters need a significant amount of storage space.

Outline

- Introduction
  - Problem Statement
  - Research Contribution
- Development of a CUDA-Based Proposed Framework
- Optimization of Proposed Framework
- Utilization of Proposed Optimized Framework
- Conclusions and Future Work

Automatic License Plate Recognition System on Mobile Platform

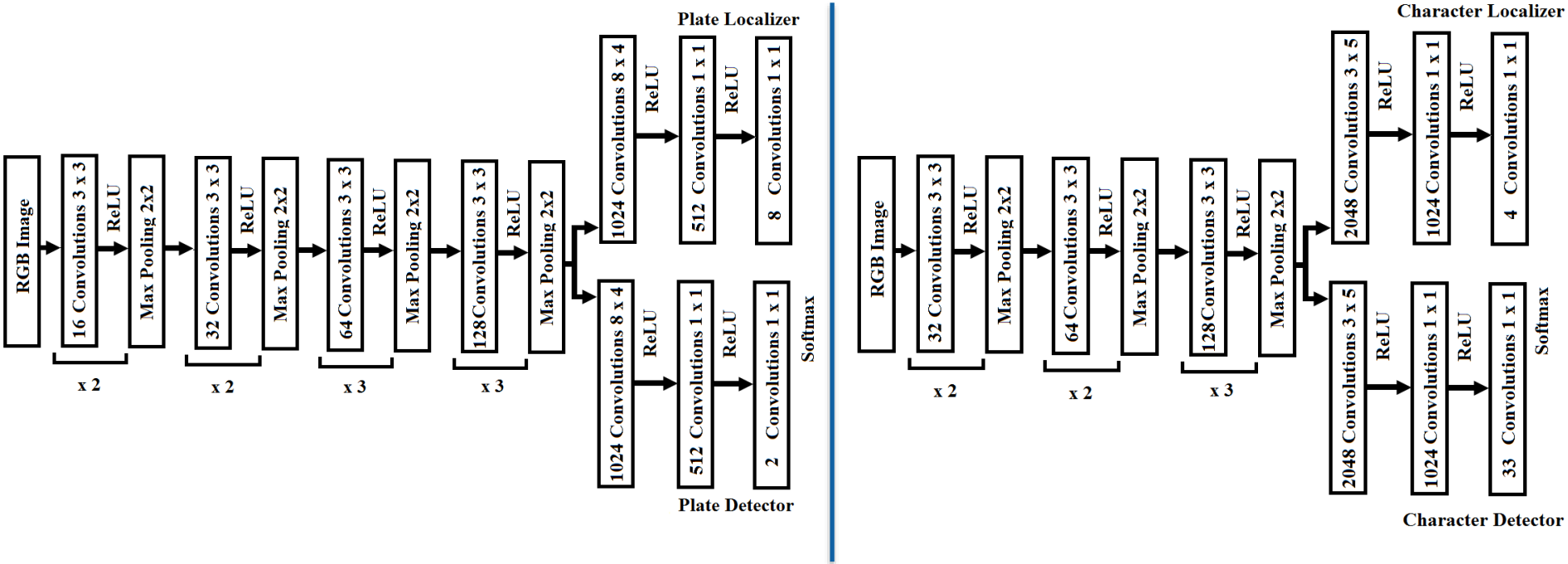
The realization of trained neural networks on the embedded platforms can open a wide range of applications, especially in the computer vision field.

- Trained parameters of a highly precise Italian license plate detection and recognition system are imported to be used by our CUDA-based framework. This ALPR system is realized on a mobile platform by simplifying the flow of trained deep architecture developed for computationally powerful desktop and server environments.

- Experimental results concluded that the even computationally complex neural network can be deployed on the embedded platforms by sacrificing some recognition performance.

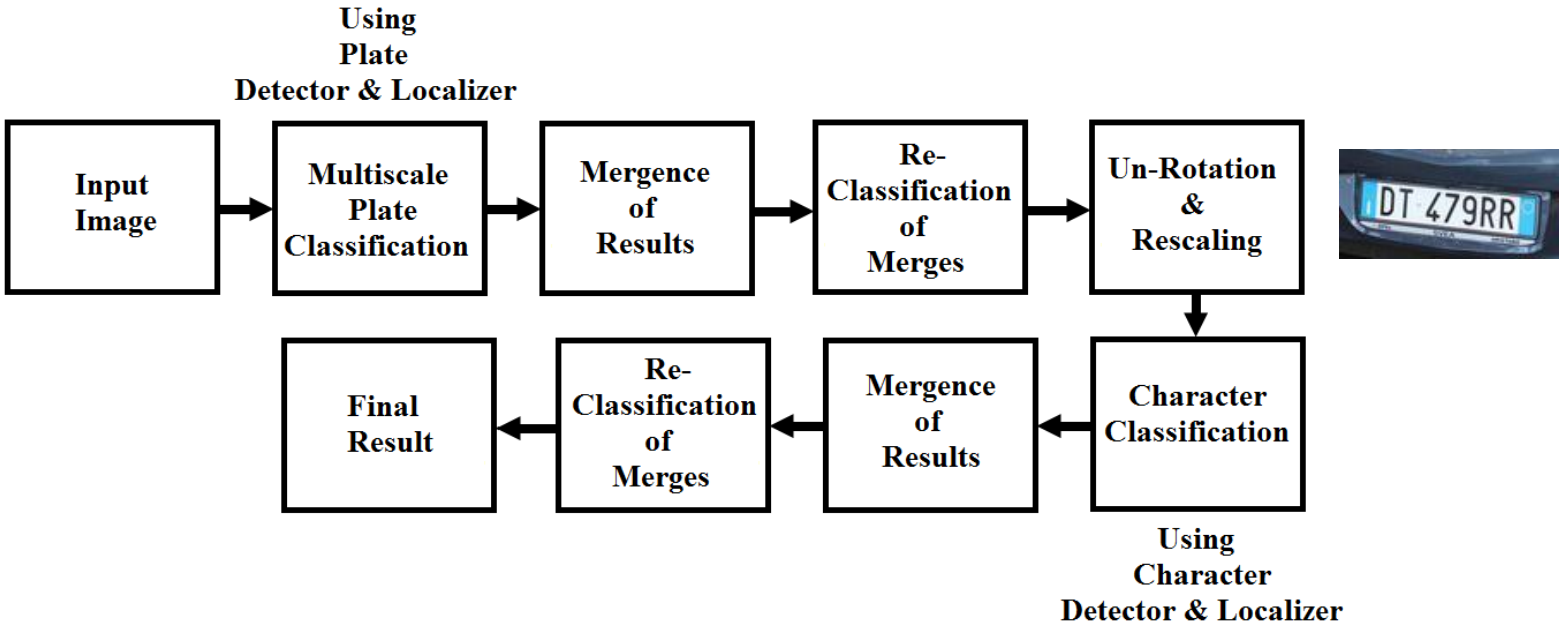
Italian License Plate Recognition System

A complete ALPR system is commonly subdivided into various computational blocks that execute sequentially. The neural network-based architecture for an Italian license plate recognition system is constructed and trained using Torch framework.

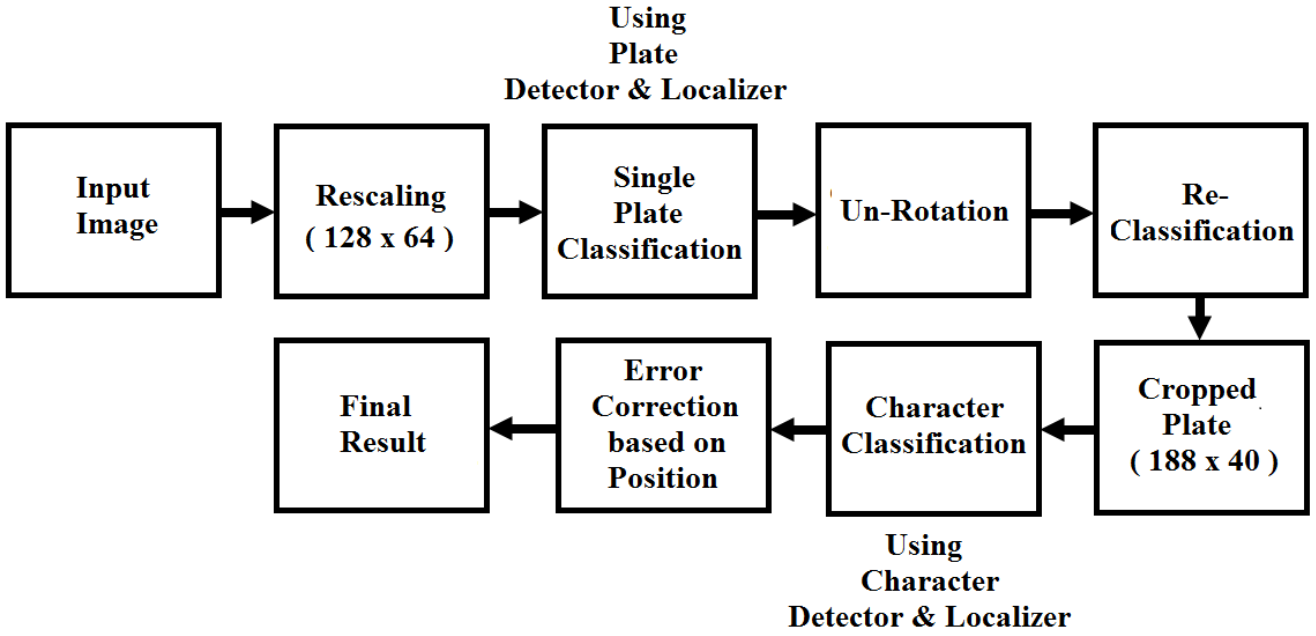


Flow of Neural Network-based ALPR System

Developed for running on computationally powerful environments



Simplified Flow for Mobile Platform



Experiments and Results

- **Hardware Platforms used for Analysis**
  - Nvidia Quadro K2200 graphic card (640 CUDA cores and 4 GB of RAM)
  - Nvidia Jetson TX1 embedded board (256 CUDA cores and 4 GB of RAM)
  - Nvidia Shield K1 tablet (192 CUDA cores and 2 GB of RAM)

Performance Analysis of Original and Simplified Networks

Image Size in Original Flow: 640 × 480
Image Size in Simplified Flow: 128 × 64

Network	Quadro K2200		Jetson TX1		Nvidia Shield K1
	Original	Simplified	Original	Simplified	Simplified
	(Milliseconds)				
Plate Classification Time	150	26	1450	244	300
Character Classification Time	510	27	3240	248	250

The execution time is reduced significantly due to elimination of intensive tasks like merging of results and reclassifications in the simplified flow.

Analysis of Original and Simplified Networks

Offset for Cropping Detected Plates	Full Plate Accuracy	Single Character Accuracy
0	17%	68%
2	28%	78%
4	30%	82%
8	25%	79%
4 (+2) (21 detected characters)	54%	90%
4 (+2) + Error correction	**61%**	**92%**
Original Network	**94%**	**98%**

Original Character	Correction
B	8
D	0
Z	2
J	1
G	6

The best classification accuracy is obtained by cropping the detected license plate with offset 4 outside the localization values and shifting 2 pixels to the right.

[4]
Rizvi, S.T.H.; Patti, D.; Björklund, T.; Cabodi, G.; Francini, G. Deep Classifiers-Based License Plate Detection, Localization and Recognition on GPU-Powered Mobile Platform. Future Internet 2017, 9, 66.

Outline

- Introduction
  - Problem Statement
  - Research Contribution
- Development of a CUDA-Based Proposed Framework
- Optimization of Proposed Framework
- Utilization of Proposed Optimized Framework
- Conclusions and Future Work

Conclusions and Future Works

- The proposed optimized scheme can perform image classification in real time on embedded platforms with significant improvement in overall results including inference performance, energy efficiency and storage requirements.
- Even computationally complex neural network can be deployed on the embedded platforms by sacrificing some recognition performance.

- The performance of developed framework can be further improved by reducing the computational complexity of the convolution operation.
  - Winograd's minimal filtering technique
  - GPU architecture based Exploitation (Support of Half Precision on Pascal GPUs)

Publications (Conference)

(1)

Gabor Filter based Image Representation for Object Classification

S Tahir H Rizvi, Gianpiero Cabodi, Pedro Gusmao and Gianluca Francini

IEEE International Conference on Control, Decision and Information Technologies

April 2016, Malta

(2)

Comparison of GPGPU based Robotic Manipulator with other Embedded Controllers

S Tahir H Rizvi , Gianpiero Cabodi , Denis Patti and M Majid Gulzar

IEEE International Conference on Development and Application Systems (DAS)

May 2016, Romania

(3)

GPGPU based Concurrent Classification using Trained Model of Handwritten Digits

S Tahir H Rizvi , Gianpiero Cabodi , A. Arif, M. Y. Javed and M Majid Gulzar

IEEE International Conference on Open Source Systems & Technologies (ICOSST)

Dec 2016, Pakistan

(4)

GPU-only Unified ConvMM Layer for Neural Classifiers

S Tahir H Rizvi, Gianpiero Cabodi and Gianluca Francini

IEEE International Conference on Control, Decision and Information Technologies

April 2017, Barcelona

Publications (Journal)

(1)

GPGPU Accelerated Deep Object Classification on a Heterogeneous Mobile Platform

S Tahir H Rizvi, Gianpiero Cabodi, Denis Patti and Gianluca Francini

Electronics, 2016, 5(4), 88

(Indexed by Scopus and Science Citation Index Expanded)

(2)

Optimized Deep Neural Networks for Real-Time Object Classification on Embedded GPUs

S Tahir H Rizvi, Gianpiero Cabodi and Gianluca Francini

Applied Sciences, 2017, 7(8), 826

(Indexed by Scopus and Science Citation Index Expanded)

(3)

Deep Classifiers based License Plate Detection, Localization and Recognition on GPU powered Mobile Platform

S Tahir H Rizvi , Denis Patti, Tomas Björklund, Gianpiero Cabodi , and Gianluca Francini

Future Internet, 2017, 9(4), 66

(Indexed by Scopus and EI Compendex)

(4)

A General-Purpose Graphics Processing Unit (GPGPU)-Accelerated Robotic Controller Using a Low Power Mobile Platform

S Tahir H Rizvi , Gianpiero Cabodi , Denis Patti and M Majid Gulzar

Journal of Low Power Electronics and Applications, 2017, 7(2), 10

(Indexed by Scopus)

Thank You