RMNV2: REDUCED MOBILENET V2 AN EFFICIENT LIGHTWEIGHT MODEL FOR HARDWARE DEPLOYMENT

A Thesis

Submitted to the Faculty

of

Purdue University

by

Maneesh Ayi

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

May 2020

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

I would like to dedicate this work to my parents and the Department of Electrical and Computer Engineering, Purdue School of Engineering, for fulfilling my academic goal.

ACKNOWLEDGMENTS

I am thankful for my kin, mom and dad, who have given me through good and passionate help in my life. I am additionally appreciative to my other relatives and companions who have bolstered me en route. Very special gratitude goes out to all down to the Department of Electrical and Computer Engineering, Purdue School of Engineering, for helping me do the research. With a special mention to my professor, Dr. Mohamed El-Sharkawy, It was a fantastic experience to work under him. His constant support and guidance is unforgettable which helped me to finish the research successfully. And, IoT Collaboratory lab what a cracking place to work! And finally, last but by no means least, also to everyone in the IoT Collaboratory lab it was great sharing laboratory with all of you during last two years. Thanks for all your encouragement!

TABLE OF CONTENTS

Page

LIST OF TABLES . . . viii

LIST OF FIGURES . . . ix

ABBREVIATIONS . . . xi

ABSTRACT . . . xiii

1 INTRODUCTION . . . 1

1.1 Context . . . 1

1.2 Motivation . . . 1

1.3 Challenges . . . 2

1.4 Methodology . . . 2

1.5 Contributions . . . 2

2 BACKGROUND . . . 3

2.1 Convolution Neural Networks . . . 3

2.2 Input Image . . . 4

2.3 Convolution Layer . . . 5

2.4 Pooling . . . 6

2.5 Nonlinearity . . . 7

2.6 FC layer . . . 7

2.7 Related Research . . . 8

2.8 Baseline architecture - Mobilenet V2 . . . 9

2.8.1 Layers in Mobilenet V2 . . . 9

2.8.2 Results of Mobilenet V2 . . . 10

3 HARDWARE AND SOFTWARE . . . 13

3.1 Hardware Used . . . 13

3.2 NXP Bluebox 2.0 . . . 13

Page

3.2.1 Specifications 16
3.2.2 S32V234 Vision Processor 16
3.2.3 LS2084A Embedded Vision Processor 17
3.3 NXP i.MX RT1060 18
3.3.1 Specifications 18
3.4 Software Used 19
3.5 Real-Time Multi Sensor Applications (RTMaps) 19
3.6 MCUXpresso 20
3.7 Frameworks for DNNs 22
3.7.1 TensorFlow 22
3.7.2 Keras 23
3.7.3 Pytorch 23
3.8 LiveLossPlot 24
4 PROPOSED ARCHITECTURE: REDUCED MOBILENET V2 25
4.1 Disabling Downsample Layers 25
4.1.1 Mathematical Analysis for Changing Strides from 2 to 1 27
4.2 Replacing Bottlenecks with HetConv Blocks 28
4.2.1 Mathematical Analysis of HetConv Blocks 29
4.2.2 For Depthwise and Pointwise Convolutions 30
4.2.3 For Groupwise Followed by Pointwise Convolutions 31
4.3 Mish Activation Function 31
4.4 AutoAugmentation 33
5 RESULTS 36
5.1 Training Setup 36
5.2 Loss and Accuracy Curves 37
5.3 Comparison between baseline and Proposed model RMNv2 39
6 IMPLEMENTATION 40
6.1 Implementation on NXP Bluebox 2.0 40

Page

6.2 NXP Bluebox 2.0 Results 42
6.3 Implementation on NXP i.MX RT1060 43
6.4 NXP i.MX RT1060 Results 44
7 CONCLUSIONS 47
8 FUTURE SCOPE 48
REFERENCES 49

LIST OF TABLES

Table Page

4.1 Mobilenet V2 Baseline Architecture Representation 26

4.2 First Change: Strides Change from 2 to 1 27

5.1 Baseline Results 37

5.2 RMNv2 Results 38

5.3 Comparison of Various Results 39

5.4 Comparison of Time 39

LIST OF FIGURES

Figure Page

2.1 Image Classification Example . . . 3
2.2 Input Image . . . 4
2.3 Convolution Operation . . . 5
2.4 Different Pooling Methods . . . 6
2.5 Different Activation Functions . . . 7
2.6 Fully Connected Layer . . . 8
2.7 Layers in Mobilenet V2 . . . 9
2.8 ImageNet Accuracy Mobilenet V2 . . . 10
2.9 Impact of Bottlenecks . . . 11
2.10 Impact of Shortcut . . . 11
2.11 Impact of Shortcut . . . 12
3.1 NXP Bluebox 2.0 . . . 13
3.2 Building Blocks of NXP Bluebox 2.0 . . . 14
3.3 Working Principle of NXP Bluebox 2.0 . . . 15
3.4 S32V234 Block Diagram . . . 16
3.5 LS2084A Block Diagram . . . 17
3.6 i.MX RT1060 . . . 18
3.7 Platforms that are Currently Supported by RTMaps . . . 20
3.8 RTMaps Deploy in BLBX 2.0 . . . 21
3.9 Block Diagram of MCUXpresso . . . 22
3.10 Different Frameworks in DNNs . . . 23
4.1 RMNv2 Architectural Representation using Flowchart . . . 28
4.2 Example of HetConv with Kernel Size 3×3 and P= 2,4 . . . 29
4.3 Mish Activation Function . . . 32

Figure Page

4.4 Some Common Activation Functions 33

4.5 Autoaugmentation Policy Example 34

4.6 Sub-Policy Example 35

5.1 HetConv for P = 4 36

5.2 Baseline Curves 37

5.3 Proposed RMNv2 Curves 38

6.1 Flowchart of RMNv2 Implementation in NXP Bluebox 2.0 40

6.2 Python Component Representation in RTMaps 41

6.3 RTMaps Console Output 42

6.4 Teraterm Output 43

6.5 Flowchart Representation of Deploying RMNv2 on NXP i.MX RT1060 . . 44

6.6 Input Cat Image 44

6.7 Input Ship Image 45

6.8 Output for Cat Image 45

6.9 Output for Ship Image 46

8.1 NXP 8M Mini-EVK 48

ABBREVIATIONS

ADAS	Advanced Drive Assistance Systems
AI	Artificial Intelligence
BLE	Bluetooth Low Energy
BLBX	Bluebox
CMYK	Cyan Magenta Yellow Black
CNN	Convolutional Neural Networks
CV	Computer Vision
DL	Deep Learning
DNN	Deep Neural Networks
FC	Fully-Connected
FPGA	Field Programmable Gate Array
GPU	Graphical Processing Unit
GUI	Graphical User Interface
Hr	Hours
HSV	Hue, Saturation and Value
IP	Internet Protocol
Min	Minutes
MPU	Memory Protection Unit
ReLU	Rectified Linear Unit
RGB	Red, Green and Blue
RMNv2	Reduced Mobilenet V2
RTMaps	Real-Time Multi Sensor Applications
SDK	Software Development Kit
SGD	Stochastic Gradient Descent

SSD	Single Shot Detection
TCP	Transmission Control Protocol
TF	TensorFlow
UAV	Unmanned Aerial Vehicles
YOLO	You Look Only Once

ABSTRACT

Ayi, Maneesh. M.S.E.C.E., Purdue University, May 2020. RMNv2: Reduced Mobilenet V2 – An Efficient Lightweight Model for Hardware Deployment. Major Professor: Mohamed El-Sharkawy.

Humans can visually see things and can differentiate objects easily but for computers, it is not that easy. Computer Vision is an interdisciplinary field that allows computers to comprehend, from digital videos and images, and differentiate objects. With the Introduction to CNNs/DNNs, computer vision is tremendously used in applications like ADAS, robotics and autonomous systems, etc. This thesis aims to propose an architecture, RMNv2, that is well suited for computer vision applications such as ADAS, etc.

RMNv2 is inspired by its original architecture Mobilenet V2. It is a modified version of Mobilenet V2. It includes changes like disabling downsample layers, Heterogeneous kernel-based convolutions, mish activation, and auto augmentation. The proposed model is trained from scratch in the CIFAR10 dataset and produced an accuracy of 92.4% with a total number of parameters of 1.06M. The results indicate that the proposed model has a model size of 4.3MB which is like a 52.2% decrease from its original implementation. Due to its less size and competitive accuracy the proposed model can be easily deployed in resource-constrained devices like mobile and embedded devices for applications like ADAS etc. Further, the proposed model is also implemented in real-time embedded devices like NXP Bluebox 2.0 and NXP i.MX RT1060 for image classification tasks.

1. INTRODUCTION

1.1 Context

AI is a rapidly evolving branch of computer science. The main aim is to make machines react and work like humans. This term is coined as “Perception” in humans. It seems to be a simple term for humans but machines, it is not. CV is a core part of AI that trains machines to identify and differentiate objects. CV has many applications like ADAS, robotics, autonomous systems, etc. DL algorithms make it possible to achieve the task for the machine to differentiate objects. In ADAS, image, and video based ADAS is well known because they use CNN and DNN algorithms for their applications. These applications include object detection, sign classification, lane detection, vehicle movement, steering control.

1.2 Motivation

Deploying DL algorithms in embedded and mobile devices is a little difficult. This may be due to various reasons. It may be due to the large model size of an algorithm, limited computational capacity of the target device, limited power supply, low memory available on the target device, etc. This motivated this thesis to propose an optimized model that is suitable to deploy on an embedded platform like NXP Bluebox 2.0 and NXP i.MX RT1060. Furthermore, this thesis serves as an motivation to perform real-time applications like object recognition, object tracking etc. in various fields like automotive industry, medical field, robotics and other industry applications.

1.3 Challenges

- Rapid training and testing of DNNs
- Requires small model with competitive accuracy
- Able to perform inference on Embedded devices

1.4 Methodology

- Architectural changes
- Implementing heterogeneous kernels
- Changing from ReLU6 to Mish activation
- Autoaugmentation
- Training from scratch on the CIFAR10 dataset
- Implementing on NXP Bluebox 2.0
- Implementing on NXP i.MX RT1060

1.5 Contributions

- Proposed a compact model
- Achieved better model size than baseline
- Implemented on NXP Bluebox 2.0
- Implemented on NXP i.MX RT1060
- Two research papers

2. BACKGROUND

In this chapter, the basic concepts of CNN is discussed. Along with previous works that are related to the proposed work is discussed. Also, the baseline network, Mobilenet V2, is also explained.

2.1 Convolution Neural Networks

CNN is an important concept in the field of computer vision. It is a deep learning algorithm that takes an input image and be able to differentiate one from another. It has different applications like Image classification, object recognition, etc.

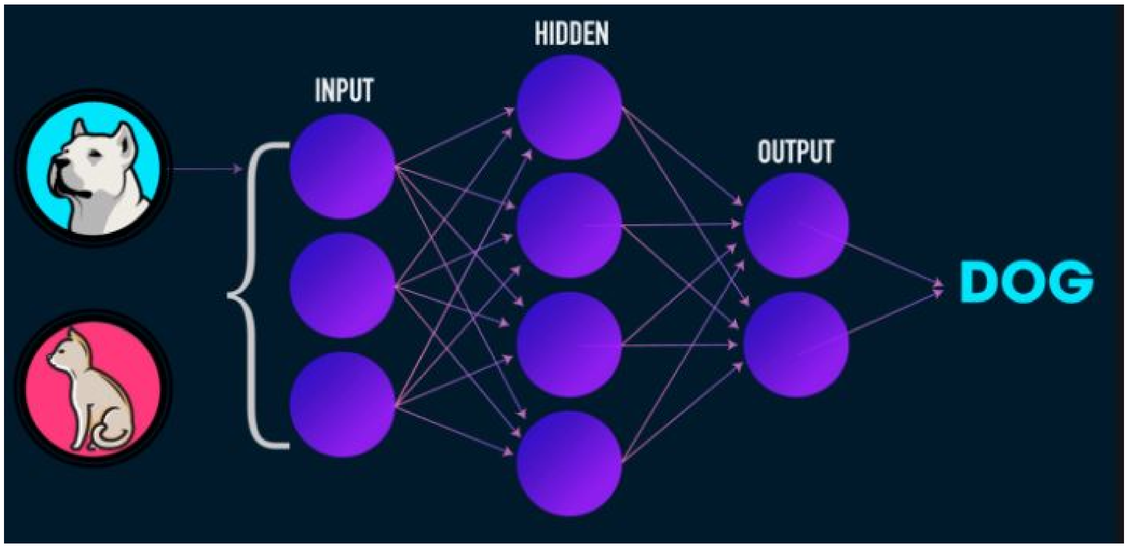


Fig. 2.1. Image Classification Example

Fig 2.1 shows an Image classification example. Input is an image and the classifier output predicts and classifies the output. A CNN network is a combination of Input layers, hidden layers, and output layers. Hidden layers consist of series of convolutional layers with activation function. Subsequently followed by additional layers such as pooling, fully connected and normalization layers.

2.2 Input Image

It is a matrix consisting of pixel values. With the utilization of filters, the network can learn spatial data from these images.

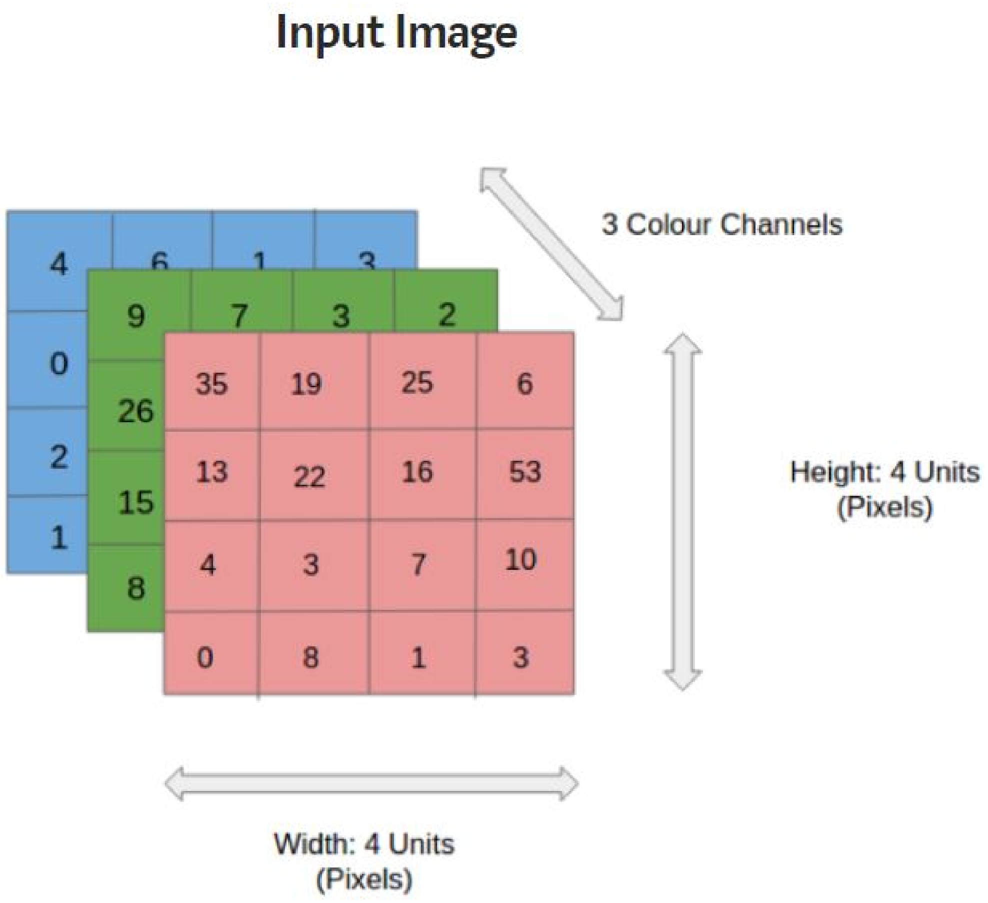


Fig. 2.2. Input Image

The Input image can be a grayscale image, RGB, HSV, CMYK. The role of CNN is use to reduce the size of the image without losing any important data from it.

2.3 Convolution Layer

Convolution layer is to extract high level features from the Input Image. These high level features include edge extraction, color gradient, etc. The element that is used to operate convolution is called Kernel.

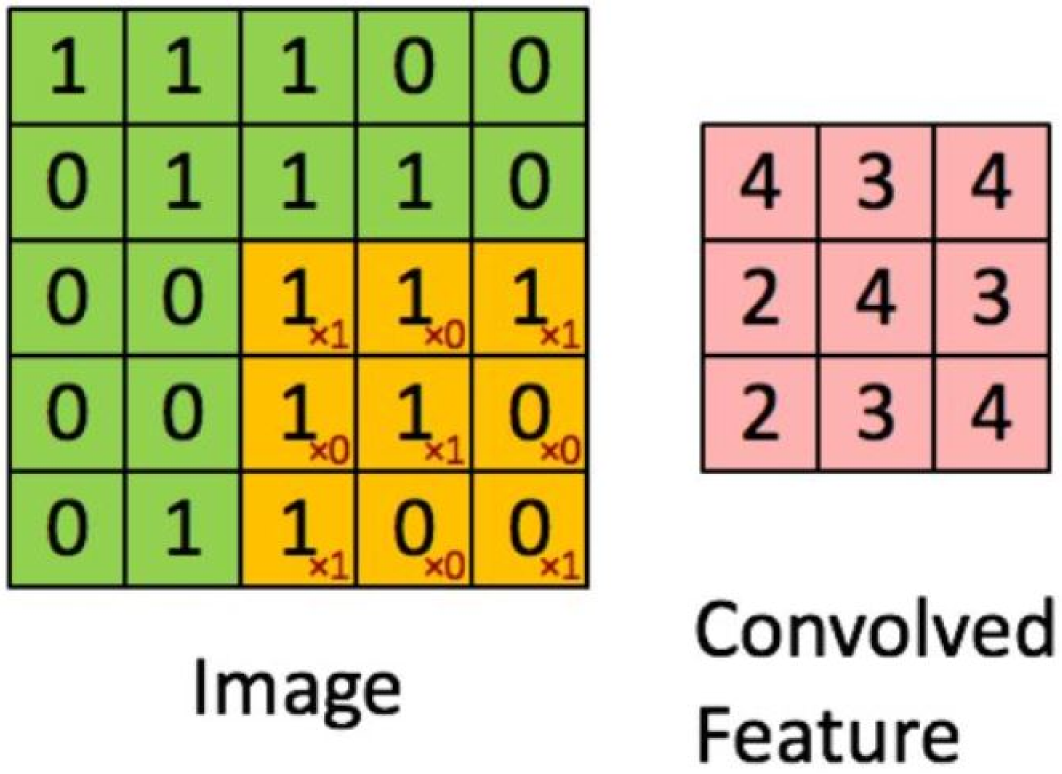


Fig. 2.3. Convolution Operation

The filter can be $7 \times 7, 3 \times 3, 1 \times 1$. The kernel will have same depth as of input image. These CNN layers need not to be limited with a single layer. It can be expanded to learn more high level features. It also includes topics like strides and padding to perform convolution operation.

2.4 Pooling

Pooling layer is to reduce the dimension of an input image. So that the computational capacity is reduced to process the data. In some context, it is similar to Convolution layer to reduce the size of Input image. It is also used to extract dominant features from an Input image.

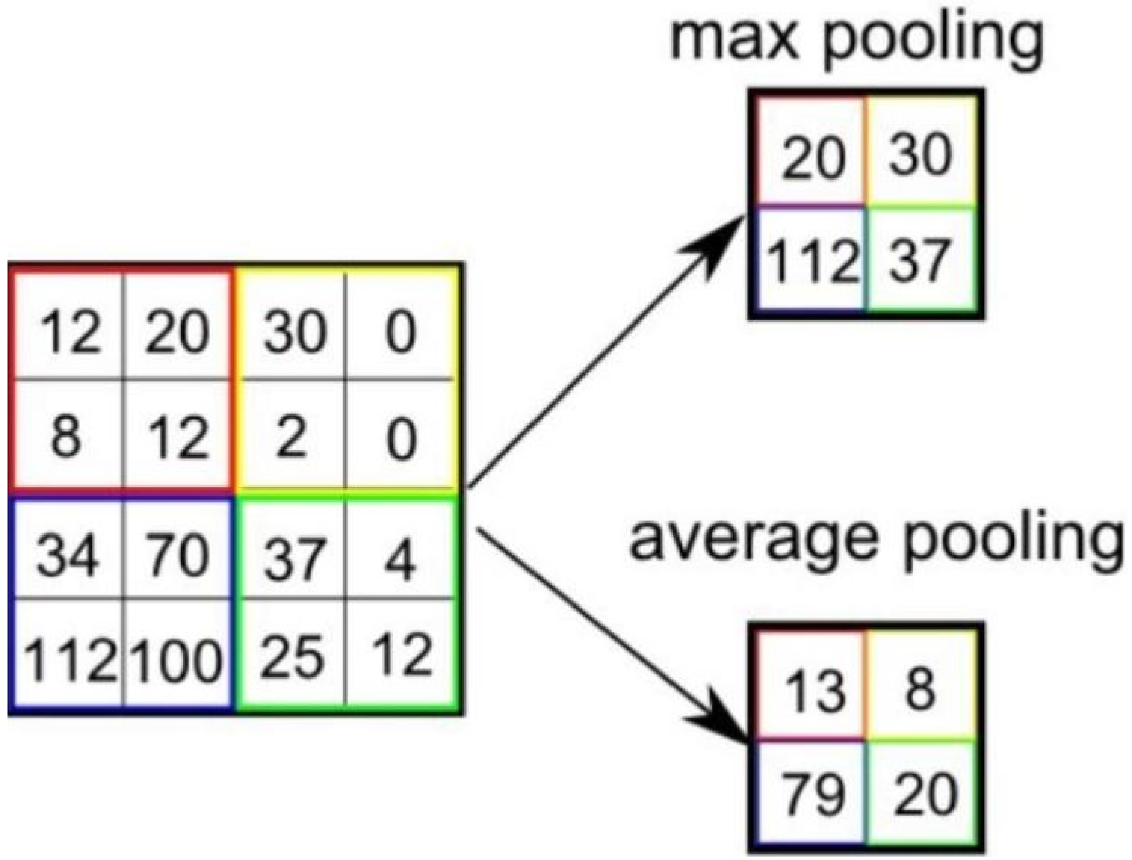


Fig. 2.4. Different Pooling Methods

There are two types of pooling. One is max pooling and the other is average pooling. max pooling takes the maximum value from the pixels. It is used to reduce the noise from an image. The average pooling takes the average value from the input pixels. when compared to maximum pooling average pooling performs better.

2.5 Nonlinearity

It is an important concept in CNN. Most of the data in real-time applications is non linear data. This non linearity is applied with convolution so that the model can learn better when the data is non linear. This non linearity can be introduced through some functions called activation functions. There are many activation functions available. ReLU is the most common activation function used in architectures.

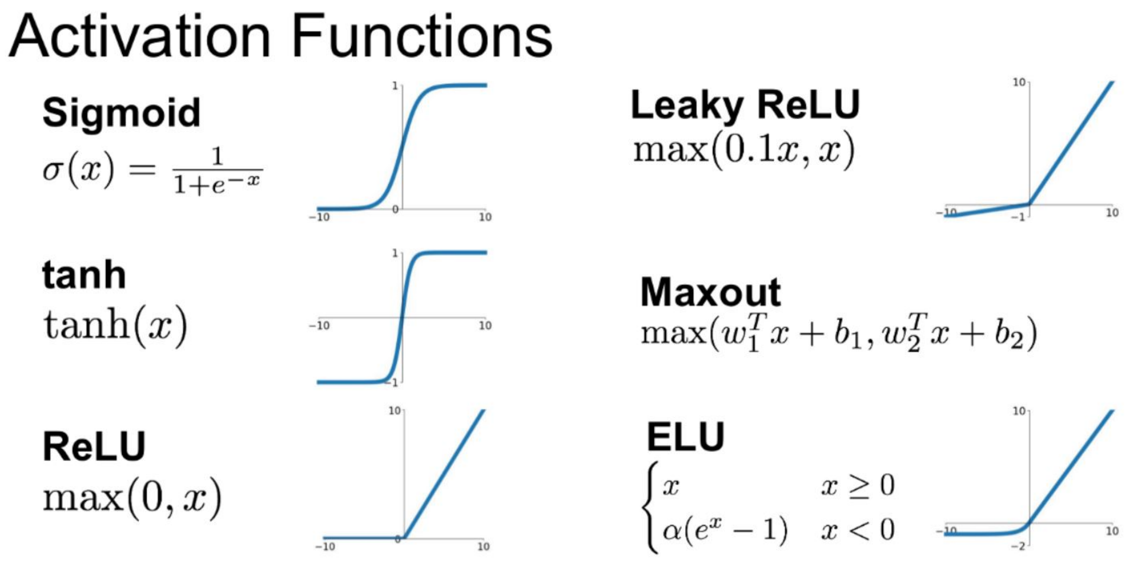


Fig. 2.5. Different Activation Functions

2.6 FC layer

In FC layer, each neuron from one layer is connected to other neuron from another layer. It is trying to learn the high-level features output given by the convolution layer. It is basically a non linear function. Over a number of epochs, the classifier is correctly able to distinguish between different classes of an image. This learned image is classified using a layer called Softmax layer. The number of nodes in this Softmax function is equal to number of classes of trained dataset. For example, ImageNet dataset consists of 1000 classes so the number of nodes is equal to 1000.

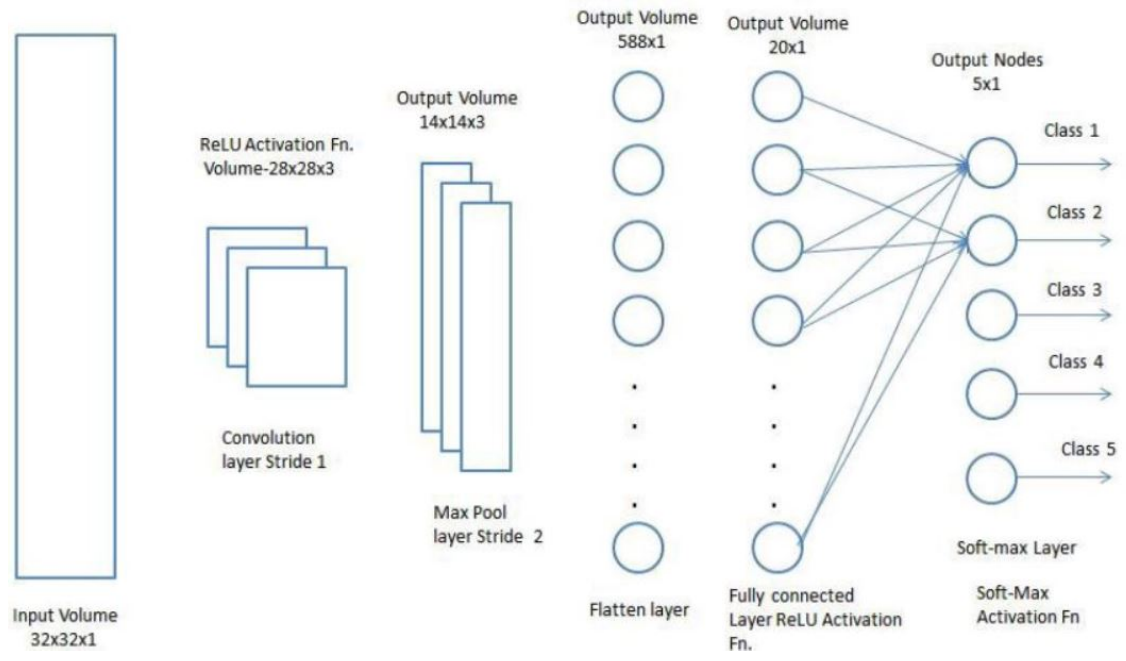


Fig. 2.6. Fully Connected Layer

2.7 Related Research

CNN is first introduced in Alexnet[6]. It won the Imagenet Challenge[7] in the year 2012. There after several other CNN's are introduced. Some of the networks like VGG[11], Inception[15][16] are much bigger in both size and accuracy. In order to implement these computer vision algorithms in resource-constrained devices like embedded and mobile devices is very difficult. So, there is a need to develop algorithms that are suitable to deploy in embedded hardware. Continuous research on this field gave two ideas to develop small models. One concept is to compress the large model or designing a small model from scratch. Compressing a pre-trained models includes concepts like Quantization[8], hashing[9], Pruning, vector quantization and Huffman Encoding[10], knowledge distillation[12], Low rank expansions[13], and Fine tuned cp-decomposition[14] etc. Developing a small model from scratch includes networks like SqueezeNet[17] and SqueezeNext[18]. These networks didn't concentrate much on speed. The lightweight models Mobilenet V1[1] and Mobilenet V2[2] are introduced. They not only focus on model size but also focus on model speed. Networks like [19][20][21] are developed using different convolutions and model structure. [22] shows some applications that are related to NXP Bluebox 2.0.

2.8 Baseline architecture - Mobilenet V2

It is a state of art lightweight model that performs better than Mobilenet V1 model. It introduces a new module called Inverted residuals and linear bottlenecks.

2.8.1 Layers in Mobilenet V2

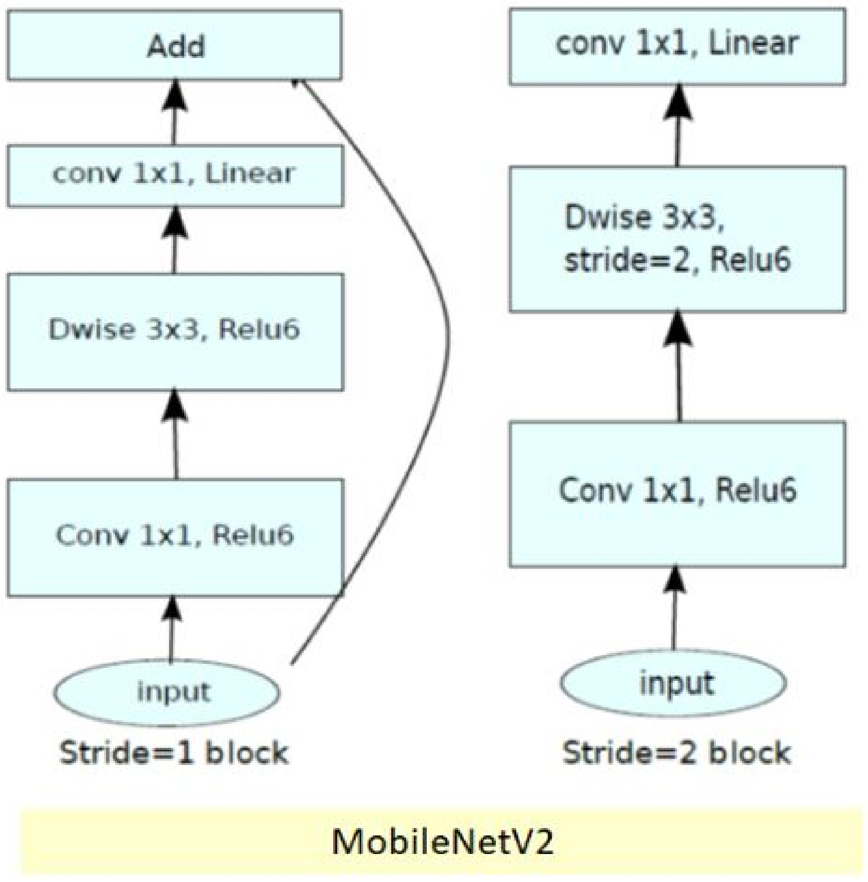


Fig. 2.7. Layers in Mobilenet V2

- **Depthwise Convolutions:** It performs lightweight filtering by applying a single convolutional filter per input channel.

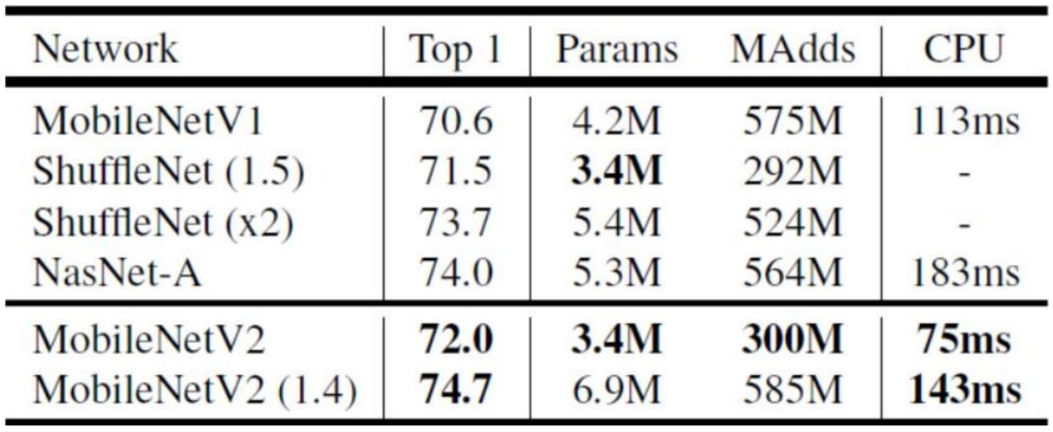
- **Pointwise Convolutions:** It is responsible in computing new features through linear combination of input channels.

- **ReLU6:** It is used because of its robustness when computing with low precision.

The top layer is a 1×1 convolution layer with Relu6. The second layer is the depthwise convolution layer with Relu6. The third layer is a 1×1 convolution without Relu6. If it is used again then that layer has capacity of linear classifier on non-zero volume part. Fig. 2.7 shows the inverted residual block to the left and the linear bottlenecks block to the right.

2.8.2 Results of Mobilenet V2

Mobilenet V2 is trained and tested from scratch on the ImageNet dataset. It outperforms other architectures like Mobilenet V1, etc. Also, results show the impact of bottlenecks and shortcut connections between the blocks.



Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	**3.4M**	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	**72.0**	**3.4M**	**300M**	**75ms**
MobileNetV2 (1.4)	**74.7**	6.9M	585M	**143ms**

Fig. 2.8. ImageNet Accuracy Mobilenet V2

Fig. 2.9 shows that removal of Relu6 from last layer makes the model perform better than keeping Relu6 in the last layer.

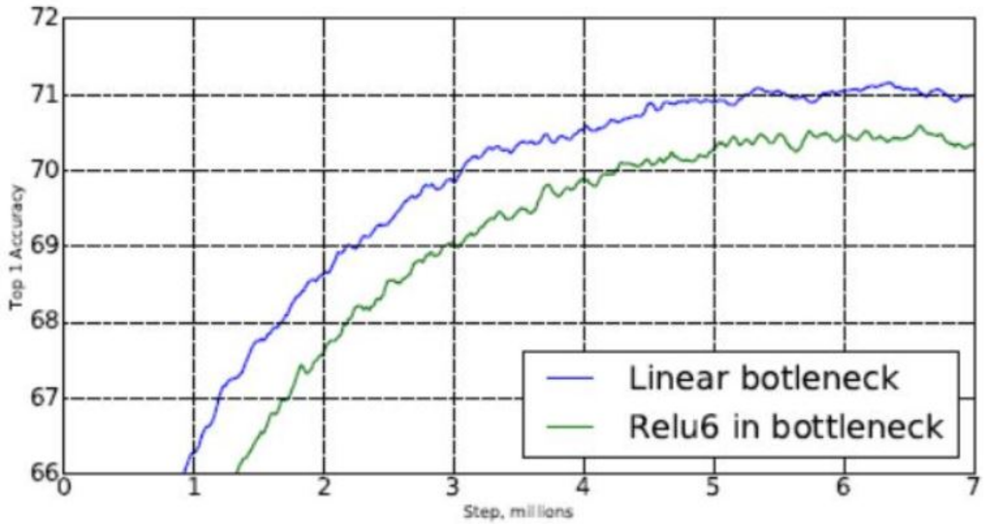


Fig. 2.9. Impact of Bottlenecks

Fig. 2.10 shows that shortcut between bottlenecks perform better than shortcut between expansion.

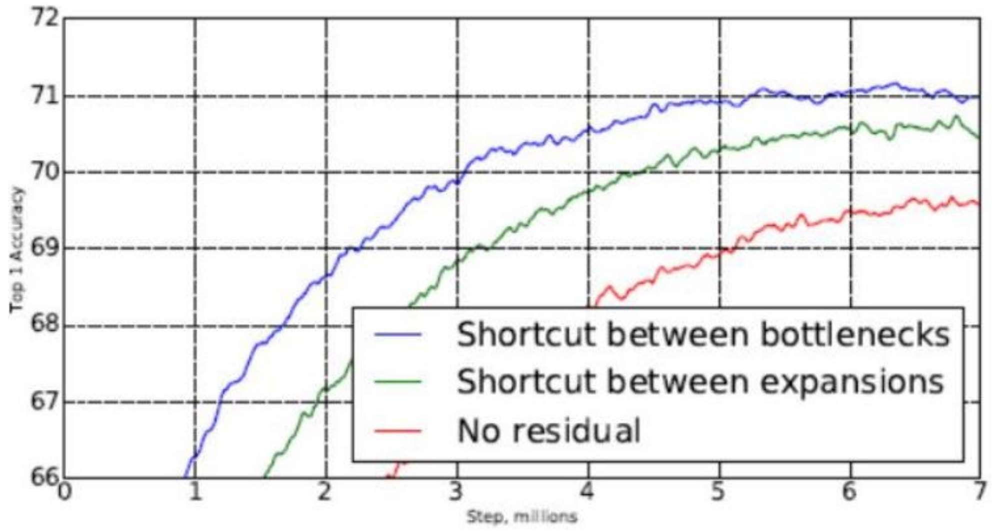


Fig. 2.10. Impact of Shortcut

From Fig. 2.11, Mobilenet V2 performs better Mobilenet V1 and ShuffleNet (1.5) with comparable model size and computational cost. Keeping width multiplier of 1.4, Mobilenet V2 (1.4) performs better than ShuffleNet (2), and NASNet with faster inference time.

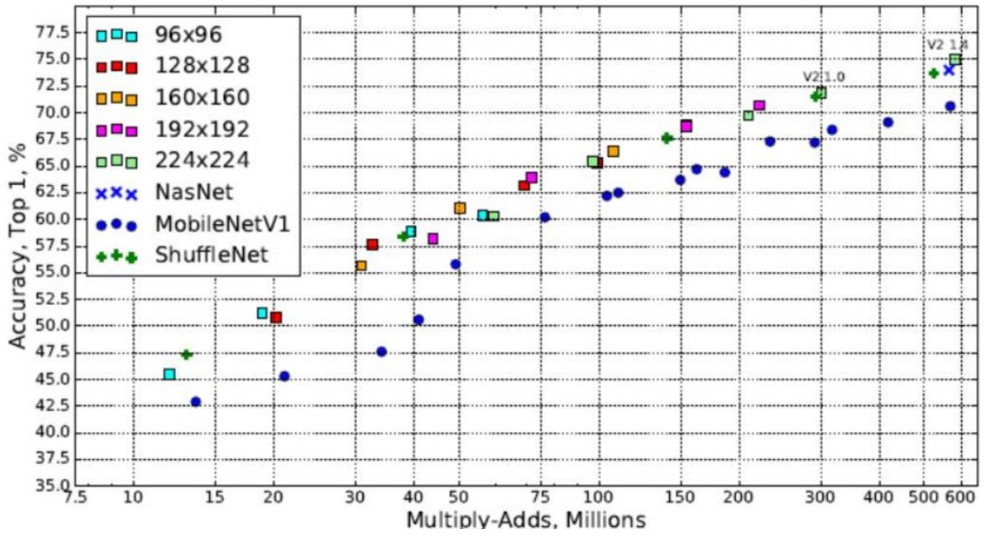


Fig. 2.11. Impact of Shortcut

3. HARDWARE AND SOFTWARE

This chapter explains about the hardware and software frameworks used in this thesis.

3.1 Hardware Used

- Intel i7-8700 processor with 32GB ram
- Nvidia GeForce GTX 1080Ti
- NXP Bluebox 2.0
- NXP i.MX RT1060

3.2 NXP Bluebox 2.0

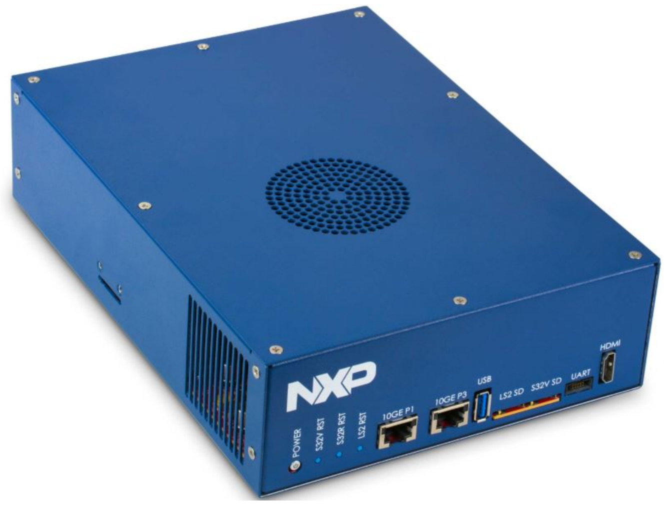


Fig. 3.1. NXP Bluebox 2.0

NXP Bluebox 2.0 is a development platform that provides reliable functionality and performance of autonomous vehicles. It is developed by NXP Semiconductors. It consists of two processors and a microcontroller. They are,

- **S32V234:** Vision processor for machine learning and sensor fusion
- **LS2084:** Embedded Computer Processor
- **S32R27:** S32R Radar microcontroller. S32R27 Automotive and industrial radar applications

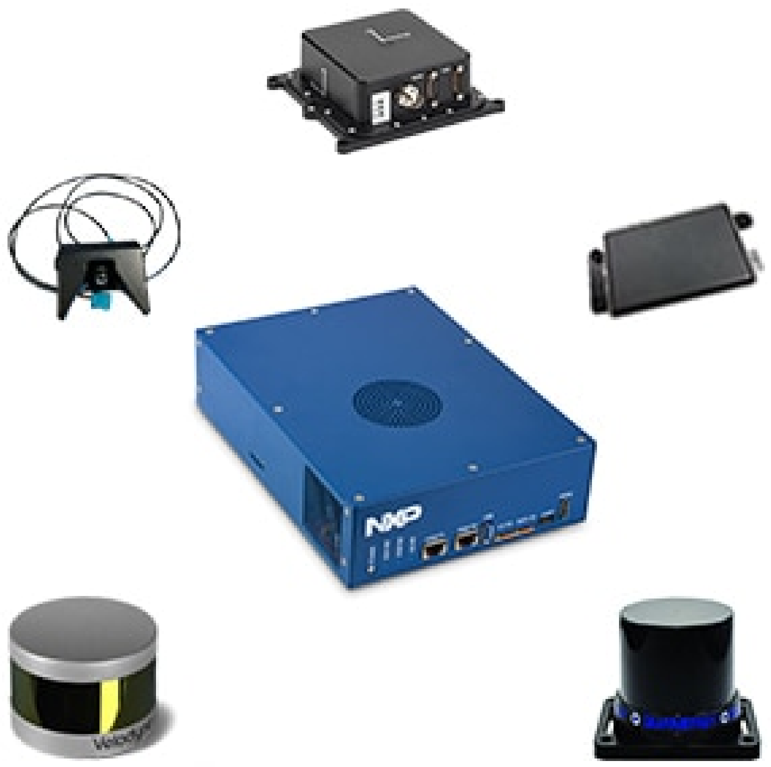


Fig. 3.2. Building Blocks of NXP Bluebox 2.0

From Fig. 3.2, NXP Bluebox 2.0 consists of comprehensive set of building blocks that helps in robust development and deployment of autonomous applications such as ADAS, robotics, etc.

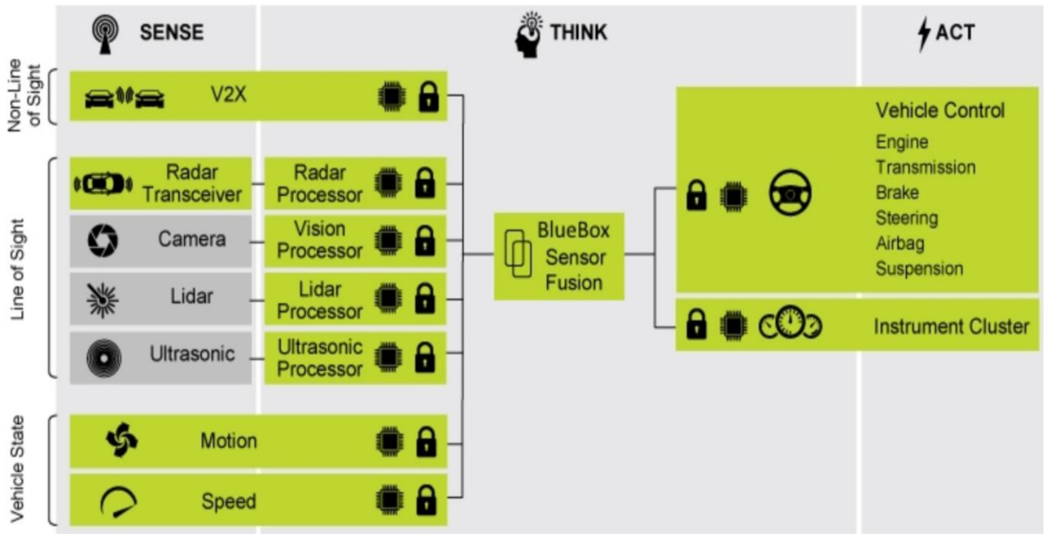


Fig. 3.3. Working Principle of NXP Bluebox 2.0

Fig. 3.3 depicts the working principle of NXP Bluebox 2.0. It works on the concept of sense, think and act. Let's see how these three terms define the working of NXP Bluebox 2.0.

Sense: Data received from sensors like cameras and radars can be an image or a video stream. This data can be used in various ADAS applications like pedestrian detection, lane detection, steering control, etc. These data from Radar, vision, and lidar is supported by the vision processor, S32V234, available on NXP Bluebox 2.0.

Think: In this phase, the captured Vision data from S32V234 is sent to the LS2084A processor for analysis of data captured. This LS2084A is the embedded processor that is available in NXP Bluebox 2.0.

Act: After analysing and processing the data in LS2084A. The embedded processor, LS2084A, selects the best course of action for safe and reliable movement of vehicles.

To summarize, the data gathered from various sensors like camera, radar and lidar etc. is processed using bluebox processor. The vehicle acts according to the instruction given from the bluebox processor.

3.2.1 Specifications

- ASIL-B compute, vision accelerated automotive interfaces.
- ASIL-D subsystem, with dedicated interfaces.
- Automotive I/O, various interfaces.
- 12 V /24 V compatible input power for vehicle.
- 16 GB DDR4 and 256 GB SSD High performance compute.
- Ethernet 100M/1G/10Gbps, SFP+, 8 x 100BASE-T1, CAN-FD, FlexRay^{TM}, 8 x cameras.
- Up to 90,000 DMIPS at lesser than 40 W, complete situational assessment, supporting classification.

3.2.2 S32V234 Vision Processor

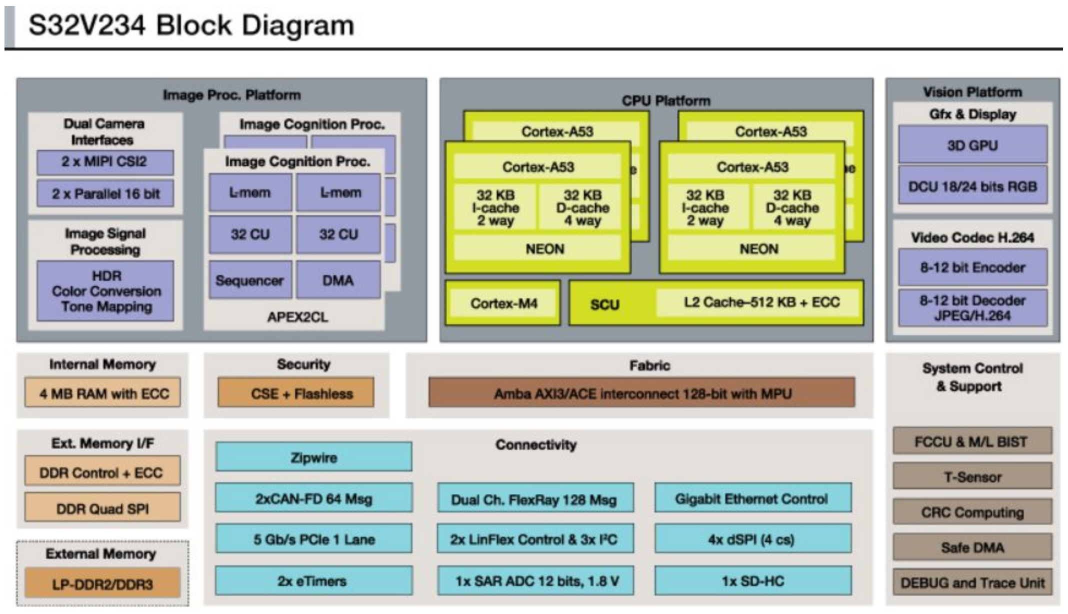


Fig. 3.4. S32V234 Block Diagram

The S32V234 MPU consists of an Image signal processor (ISP), powerful 3D Graphic Processor Unit (GPU), dual APEX-2 vision accelerators, automotive grade reliability, functional safety for supporting ADAS, machine learning, industrial image processing and sensor fusion applications. It is a second generation vision processor family and a member of 32 bit Arm Cortex-A53 S32V processors. Image and video based processor that can support applications related to Computer vision and other Image, video based applications computationally.

3.2.3 LS2084A Embedded Vision Processor

The LS2 processor is high-performance computing processor platform that is available on NXP Bluebox 2.0. The block diagram representation is shown in Fig. 3.5. It consists of SD card interface that is suitable for processor to run in Linux BSP, UBUNTU 16.04 LTS on NXP Bluebox 2.0 platform.

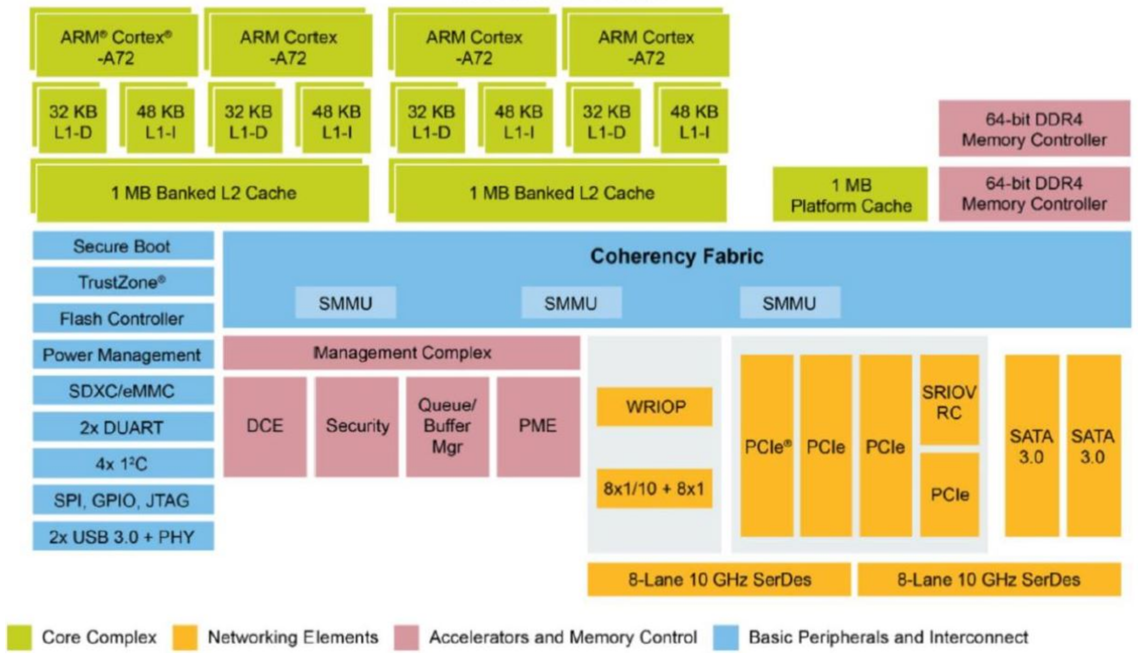


Fig. 3.5. LS2084A Block Diagram

3.3 NXP i.MX RT1060

The i.MX RT 1060 is the first crossover MCU by NXP. When compared to i.MX RT1050, it doubles the on-chip SRAM to 1MB. The Pin-to-pin compatibility is the same for i.MX RT1050 and i.MX RT1060. High-speed GPIO, CAN-FD, and synchronous parallel NAND/NOR/PSRAM controller are the new additions to this board that supports real-time applications. The i.MX RT1060 runs on the Arm Cortex-M7 core at 600 MHz.

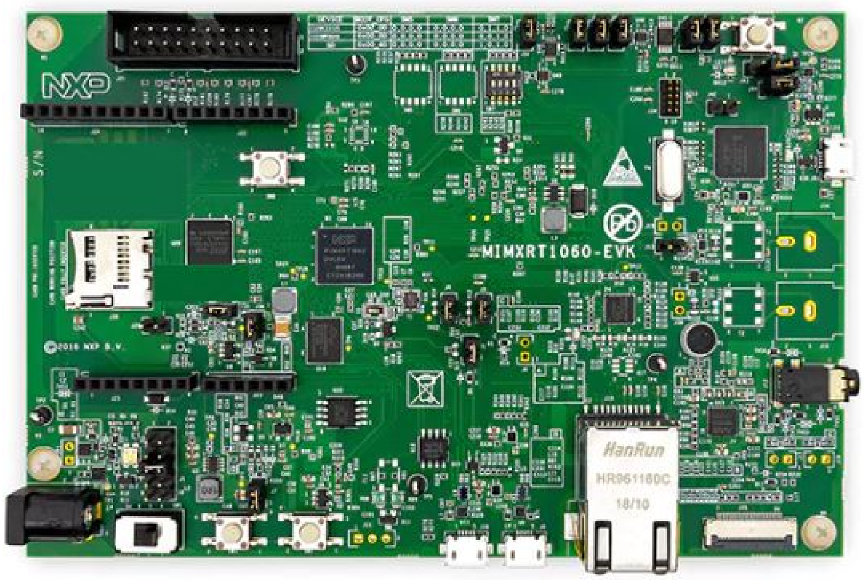


Fig. 3.6. i.MX RT1060

3.3.1 Specifications

- Highest performing Arm Cortex-M7
- 3020 CoreMark/1284 DMIPS @ 600 MHz
- Low latency
- Advance multimedia for GUI
- Wireless communication interface like Bluetooth, BLE

3.4 Software Used

- Python Version 3.6.7
- Spyder Version 3.6
- Pytorch Version 1.0
- Livelossplot (loss and accuracy visualization)
- Keras 2.1.3
- Tensorflow-GPU 1.10
- RTMaps by Intempora
- MCU Xpresso SDK
- Teraterm

3.5 Real-Time Multi Sensor Applications (RTMaps)

RTMaps is an easy-to-use framework for fast and robust developments. It is a non concurrent high performance platform designed to face and win multisensor challenges. It allow engineers and researchers to exploit an efficient framework for multi applications. It is a secluded toolbox for multimodal applications. These applications includes ADAS, autonomous vehicles, robotics, UAV's, etc. It gathers information from various sensors like camera stream, radar, Lidar, etc. and can be easily developed, tested and deployed in embedded hardware like NXP Bluebox 2.0.

RTMaps consists of several components like RTMaps Runtime Engine, RTMaps Component library, RTMaps Studio, and RTMaps embedded.

RTMaps Runtime Engine: It is a core part of RTMaps applications. All the base services, component registration, connections, buffer management, timestamping, etc will be taken care by RTMaps runtime engine. It is lightweight, highly optimized and easily deployable.

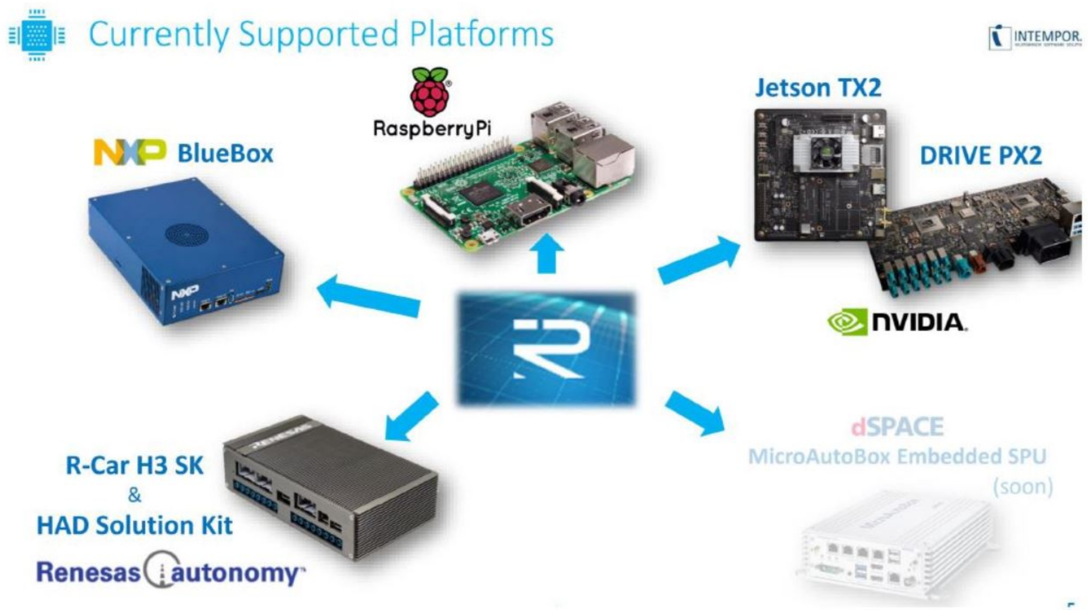


Fig. 3.7. Platforms that are Currently Supported by RTMaps

RTMaps Component library: It consists of all software packages and modules like python, C++, etc. that are responsible in developing applications. They provide a better GUI to develop applications very easily.

RTMaps Studio: It is a graphical modelling environment with the functionality to use RTMaps Components. This module helps in developing applications. It is supported in windows and Ubuntu based platforms.

RTMaps Embedded: It is a framework that consists of runtime engine, component library that is suitable to run on x86 or ARM devices such as NXP Bluebox 2.0. The version used for deploying algorithms in NXP Bluebox 2.0 is v4.5.3. The connection between host PC and NXP Bluebox 2.0 is TCP/IP connection.

3.6 MCUXpresso

It is a software development kit designed by NXP semiconductors. The MCUXpresso SDK is designed to understand and fasten the development process in with i.MX RT crossover MCUs based on ARM Cortex-M cores. It consists production-

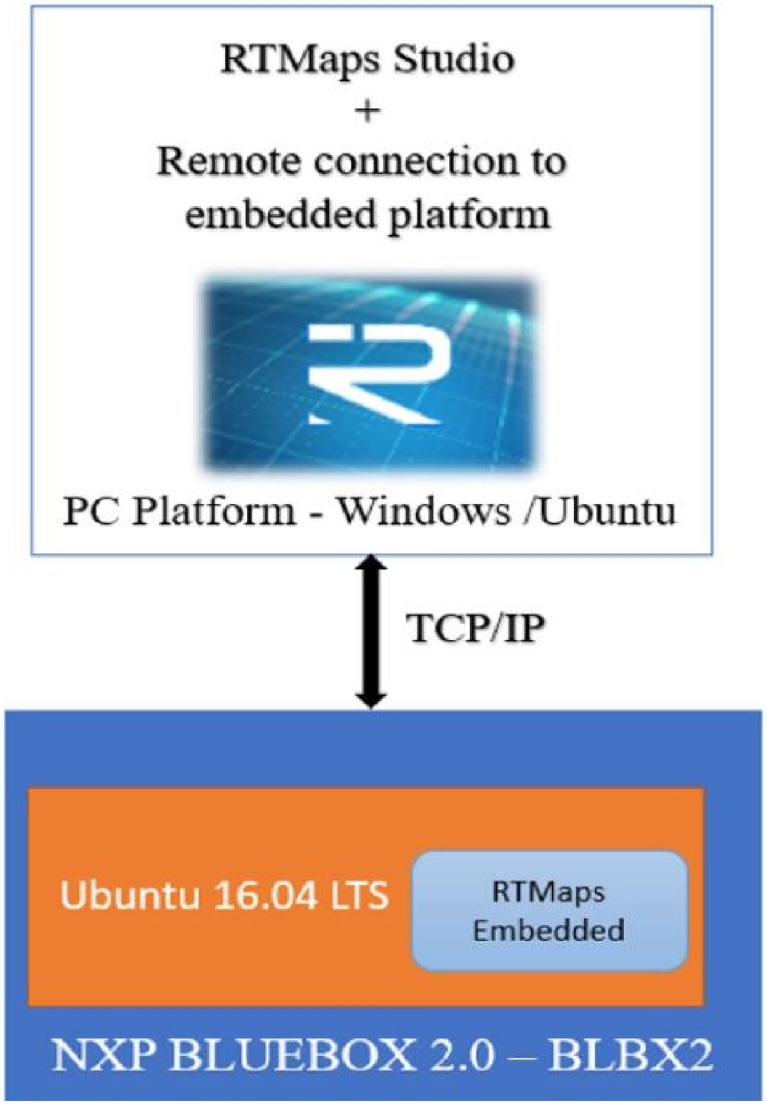


Fig. 3.8. RTMaps Deploy in BLBX 2.0

grade software along with integrated RTOS. It also supports example projects for IAR, KEIL, GCC, and Cmake. It supports i.MX RT1060 crossover MCU.

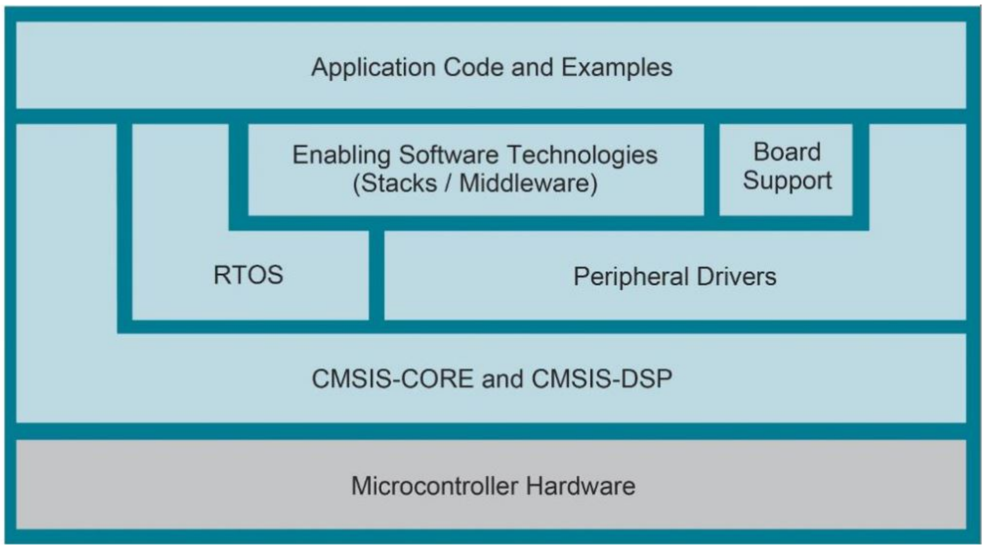


Fig. 3.9. Block Diagram of MCUXpresso

3.7 Frameworks for DNNs

There are many deep learning frameworks like TensorFlow, Pytorch, etc. It is difficult to say which framework is better than others. In this section, let us see mostly used frameworks in Deep Neural Networks.

3.7.1 TensorFlow

It is developed by Google. It is the most popular framework till date. TF considers incredible processing clusters and the ability to run models on mobile platforms like iOS and Android. It operates with static computation graph. It comes with some extra features like TensorBoard for visualization of network architecture data and performance metrics.

Fig. 3.10. Different Frameworks in DNNs

3.7.2 Keras

Keras is one of most user friendly framework. It is the most minimalist approach to frameworks like TensorFlow and Theano. It can be used as a high level API over lower level libraries like TensorFlow. It is ideal for prototyping simple concepts.

3.7.3 Pytorch

It is one of the most popular framework that is used after TensorFlow. Unlike TensorFlow, Pytorch uses dynamically updated graph. It supports wide range of libraries that support deep learning models. It has some standard debuggers like pdb or Pycharm. It supports data parallelism. It also contains many pre-trained models.

3.8 LiveLossPlot

LiveLossPlot is a data visualization library used in Python. It is used to plot performance metrics of a neural network model. It is used in plotting loss graphs and accuracy graphs of a CNN/DNN model. It helps us to track the training process for each epoch. This library has been used in this thesis to plot and visualize the results.

4. PROPOSED ARCHITECTURE: REDUCED MOBILENET V2

The proposed architecture Reduced Mobilenet V2 is inspired by the baseline Mobilenet V2 model. It can be expressed as an architecturally modified version of the Mobilenet V2 model. It includes modifications like disabling downsample layers, Heterogeneous kernel-based convolutions, mish activation function, and autoaugmentation. The first two changes, disabling downsample layers and Heterogeneous kernel-based convolutions, helped RMNv2 in decreasing the model size. The next two changes, mish activation function and autoaugmentation, helps RMNv2 in increasing the model accuracy. These changes altogether constitute the proposed model, RMNv2. This chapter explains about these individual changes in specific and give a clear picture and intuition for these changes.

- Disabling downsampling layers
- Replacing bottlenecks with HetConv blocks
- Mish Activation function
- Autoaugmentation

This chapter is organized in the above order. It starts with explaining the section disabling downsampling layers followed by replacing bottlenecks with HetConv[3] blocks section. Then, followed by Mish activation[4] function and finally with autoaugmentation[5].

4.1 Disabling Downsample Layers

The first change for the proposed model, RMNv2, is disabling downsample layers. The main intuition for this change is described as follows,

Table 4.1.
Mobilenet V2 Baseline Architecture Representation

Input	**operator**	**t**	**c**	**n**	**s**
$224^2 \times 3$	Conv2D	-	32	1	2
$112^2 \times 32$	Bottleneck	1	16	1	1
$112^2 \times 16$	Bottleneck	6	24	2	2
$56^2 \times 24$	Bottleneck	6	32	3	2
$28^2 \times 32$	Bottleneck	6	64	4	2
$14^2 \times 64$	Bottleneck	6	96	3	1
$14^2 \times 96$	Bottleneck	6	160	3	2
$7^2 \times 160$	Bottleneck	6	320	1	1
$7^2 \times 320$	Conv2D	-	1280	1	1
$7^2 \times 1280$	AvgPool	-	-	1	-
$1^2 \times 1280$	Conv2D	-	k	-	

The Table 4.1 represents the architectural representation of baseline Mobilenet V2 model. The baseline Mobilenet V2 is designed for the ImageNet dataset. The ImageNet dataset consists of 1000 classes. The input image of this dataset has a resolution of $224 \times 224 \times 3$. In CIFAR10, the input resolution of an input image is $32 \times 32 \times 3$. To make it compatible with the CIFAR10 dataset, this change has been implemented to disable downsampling layers. This change can be implemented by simply changing the strides from 2 to 1.

The Table 4.2 shows the first change in RMNv2 architecture. The boxed numbers represented in strides section of the Table 4.2 denotes the changes made to existing baseline Mobilenet V2 model. The mathematical analysis for this change is explained in the following subsection.

Table 4.2.
First Change: Strides Change from 2 to 1

Input	**operator**	**t**	**c**	**n**	**s**
$224^2 \times 3$	Conv2D	-	32	1	$\boxed{1}$
$112^2 \times 32$	Bottleneck	1	16	1	1
$112^2 \times 16$	Bottleneck	6	24	2	$\boxed{1}$
$56^2 \times 24$	Bottleneck	6	32	3	$\boxed{1}$
$28^2 \times 32$	Bottleneck	6	64	4	2
$14^2 \times 64$	Bottleneck	6	96	3	1
$14^2 \times 96$	Bottleneck	6	160	3	2
$7^2 \times 160$	Bottleneck	6	320	1	1
$7^2 \times 320$	Conv2D	-	1280	1	1
$7^2 \times 1280$	AvgPool	-	-	1	-
$1^2 \times 1280$	Conv2D	-	k	-	

4.1.1 Mathematical Analysis for Changing Strides from 2 to 1

Consider an input image with spatial height and width, $A_i \times A_i$. Let C_i be the total number of input channels. Then, the input feature map is given by $A_i \times A_i \times C_i$. Similarly, consider the output image with spatial height and width, $A_o \times A_o$. Let C_o be the total number of output channels. where k is the kernel size. The total computational cost is given by, Cost $= A_o \times A_o \times C_i \times C_o \times k \times k$. The computational cost depends upon the resolution of Input Image applied. For the CIFAR10 dataset the input image is very less when compared to ImageNet dataset. Strides try to decrease the resolution of image applied using convolutions. So, reducing strides from 2 to 1 helps model in learning essential features without loosing much information.

4.2 Replacing Bottlenecks with HetConv Blocks

The second modification is to replace Bottlenecks used in Mobilenet V2 with HetConv blocks.

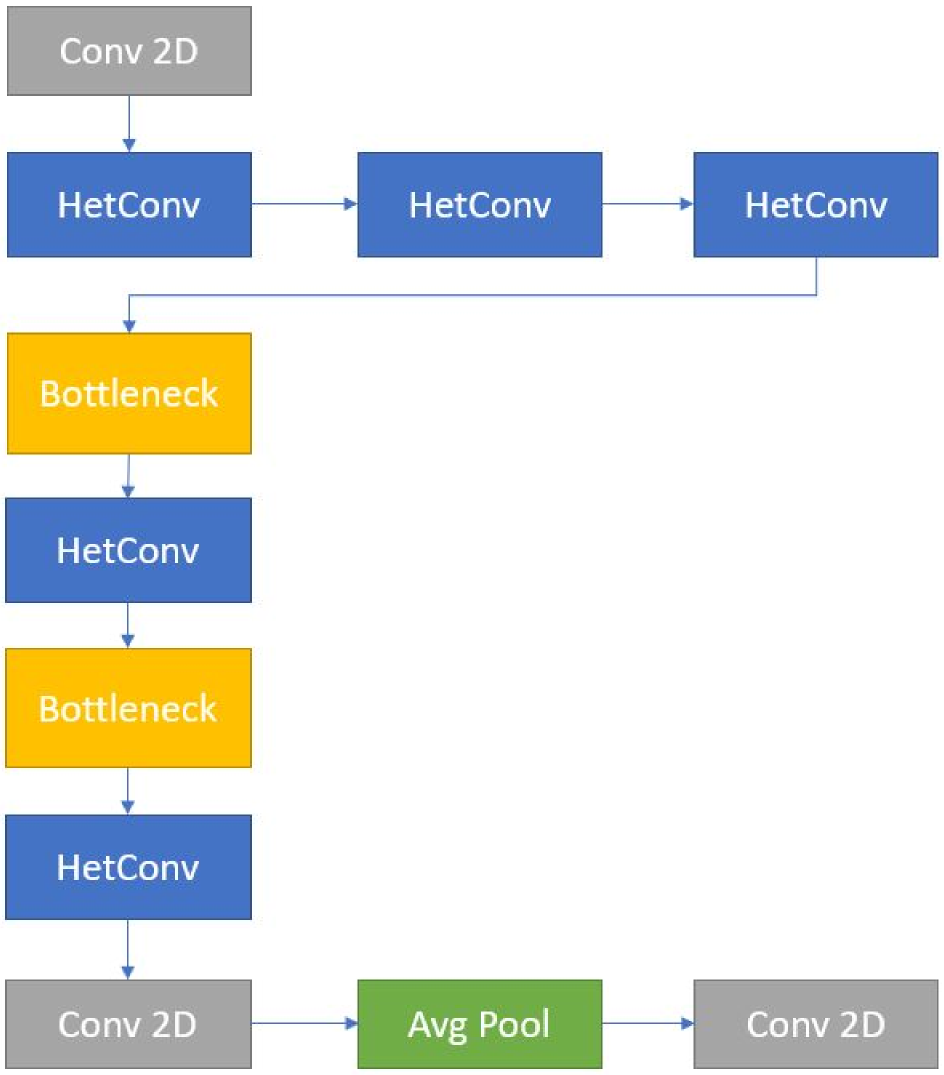


Fig. 4.1. RMNv2 Architectural Representation using Flowchart

The above Fig. 4.1 shows the architectural representation of RMNv2 architecture after replacing bottlenecks with HetConv blocks. In general, most of the networks use homogeneous kernels of sizes $1 \times 1, 3 \times 3, 5 \times 5, 7 \times 7$, etc. Mathematically, these kernels can be further optimized and can help a model in reducing the number of FLOPs. The name itself suggests that heterogeneous kernels mean kernels with varying sizes. These can be a combination of two or more kernel sizes. Implementing this type of kernels applies to all convolutions such as depthwise convolutions, grouped convolutions, etc. In heterogeneous kernels, out of all the convolutions, part **'P'** will be of size $k \times k$ and the remaining kernels will be of size 1×1. Some of the examples of HetConv is shown in Fig. 4.2.

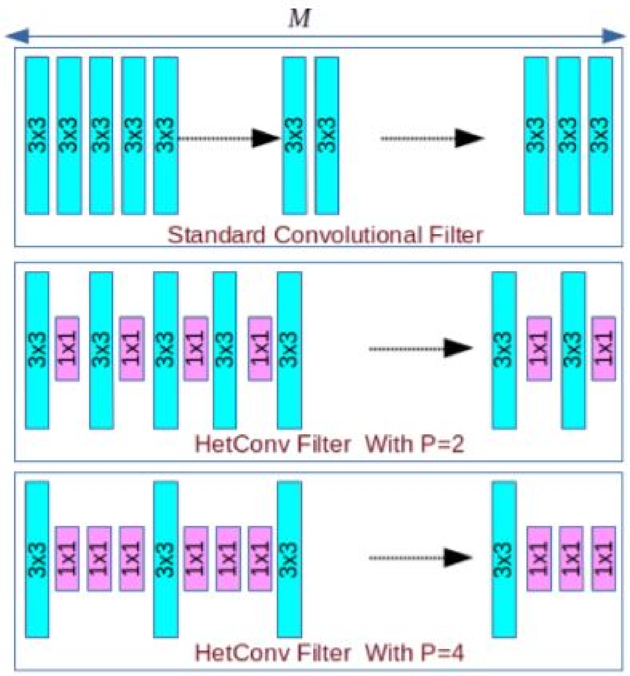


Fig. 4.2. Example of HetConv with Kernel Size 3×3 and P= 2,4

4.2.1 Mathematical Analysis of HetConv Blocks

Let us consider input image spatial height and width be $A_i \times A_i$. Let M be the total number of input channels. So, the input feature map is given by,

$$A_i \times A_i \times M \tag{4.1}$$

Let us consider the output image spatial height and width be $A_o \times A_o$. Let N be the total number of output channels. So, the output feature map is given by,

$$A_o \times A_o \times N \tag{4.2}$$

In Standard convolution filter, the total computational cost will be,

$$A_o \times A_o \times M \times N \times \boxed{k \times k} \tag{4.3}$$

It is clear that the computational cost depends upon kernel size. By carefully designing the kernel size computational cost can be further reduced.

In HetConv, parameter **'P'** is set such that part of 1/P out of total kernels will be of size $k \times k$ and the remaining (1-1/P) will be of 1×1 kernels.

The computational cost of $k \times k$ size kernels for fraction P is given by,

$$(A_o \times A_o \times M \times N \times k \times k)/P \tag{4.4}$$

It reduces cost by factor M/P. For 1×1 kernel, the computational cost will be,

$$A_o \times A_o \times N \times (M - M/P) \tag{4.5}$$

The total computational cost of the system is summation of individual computational costs for $k \times k$ kernel and 1×1 kernel. Then, total reduction of parameters is given by,

$$\boxed{Reduction = 1/P + (1 - 1/P)/k^2} \tag{4.6}$$

If P=1 substituted in the above equation, then it is a standard filter.

4.2.2 For Depthwise and Pointwise Convolutions

For Depthwise and Pointwise Convolutions, the computational cost is given by,

$$A_o \times A_o \times M \times k \times k + M \times N \times A_o \times A_o \tag{4.7}$$

When compared to standard convolution, the total number of reduction is given by,

$$\boxed{Reduction = 1/N + 1/k^2} \tag{4.8}$$

4.2.3 For Groupwise Followed by Pointwise Convolutions

For Groupwise Convolutions followed by Pointwise Convolutions, the total computational cost is given by,

$$(A_o \times A_o \times M \times N \times k \times k)/G + M \times N \times A_o \times A_o \tag{4.9}$$

Where G is number of groups. The total reduction when compared to standard convolution is given by,

$$\boxed{Reduction = 1/G + 1/k^2} \tag{4.10}$$

This analysis show that, model parameters can be further reduced by optimizing the parameter k carefully. For this reason, the proposed architecture, RMNv2, replaced bottlenecks with HetConv blocks.

4.3 Mish Activation Function

The mish activation function can be seen in the below diagram,

Mathematically, Mish is defined as,

$$\boxed{f(x) = x.tanh(ln(1 + e^x))} \tag{4.11}$$

Nonlinearities help a neural network to increase performance. It plays a crucial role in getting better accuracy for the model. This nonlinearity is introduced to

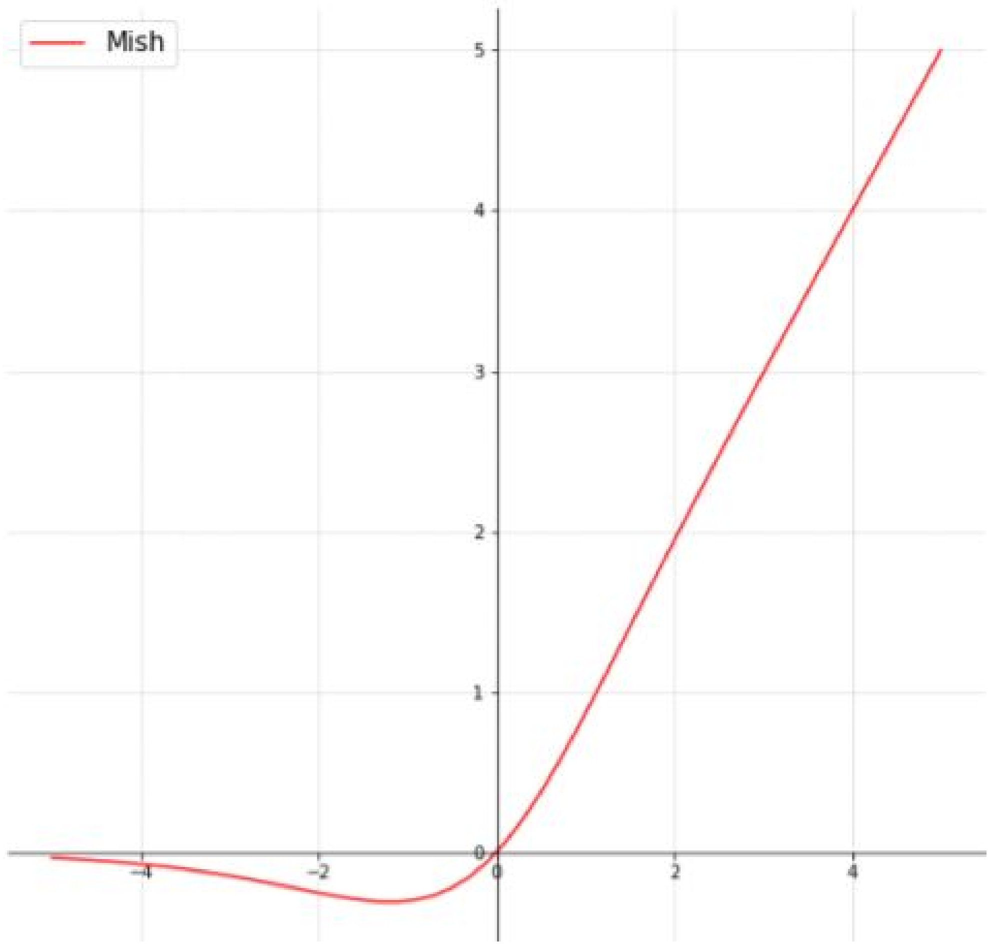


Fig. 4.3. Mish Activation Function

the neural network through an activation function. Some of the commonly used activation functions include ReLU, Swish, etc. The proposed architecture uses the Mish activation function because of its better functionality. It gives the best accuracy when compared to other available activation functions.

Some of the common activation functions along with Mish activation is shown in Fig. 4.4. When compared to other activation functions, mish activation is smoother and non-monotonic activation function.

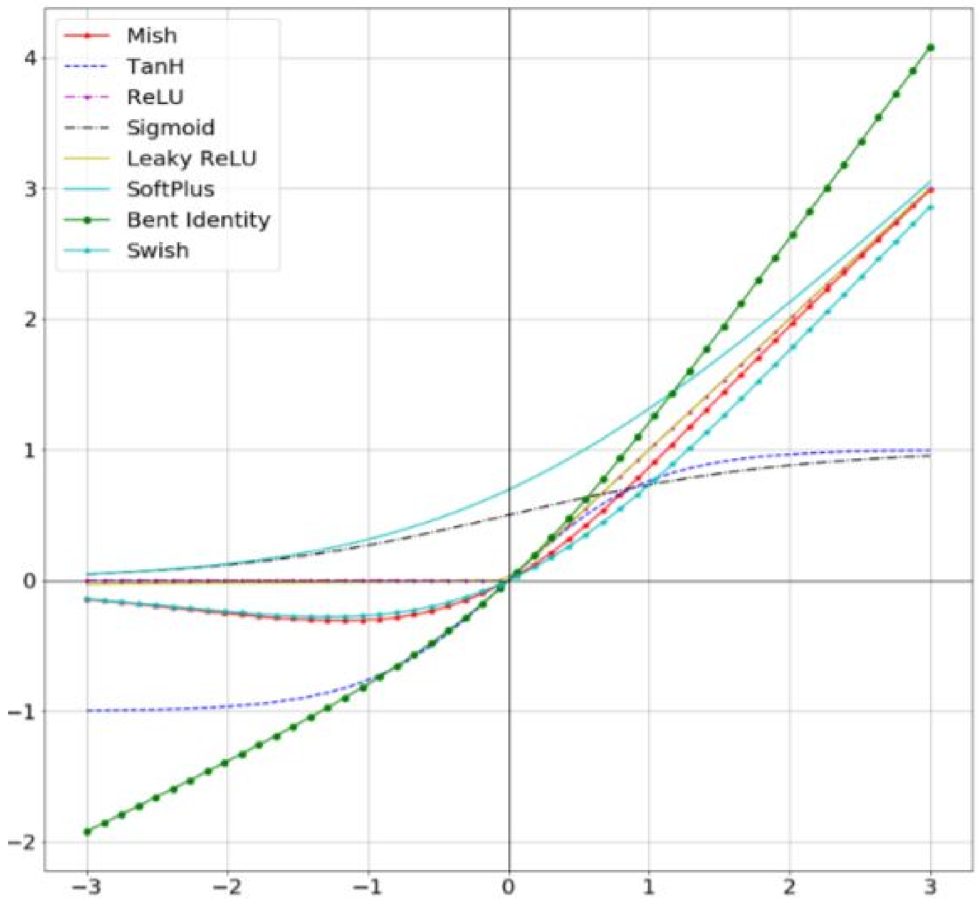


Fig. 4.4. Some Common Activation Functions

4.4 AutoAugmentation

Data is important in Convolutional Neural Networks. It is said that, how big your model is, it can classify the objects with that much accuracy. But if the data available to train the model is less. Then, the ability to classify different objects is also less. To explain this problem, when the model is trained with a dataset if the model is working accurately on test images and is not able to perform well on images that are not there in the dataset. This situation leads to problems like overfitting and underfitting. To overcome all these problems, data augmentation is introduced. Data augmentation is one of the concepts in convolutional neural networks that alters the performance of a network. Data augmentation includes some image processing techniques like cropping an image, rotating, shifting, etc. This helps to expand data and allow the model to learn new features that can help the model to perform well on images that are outside the dataset.

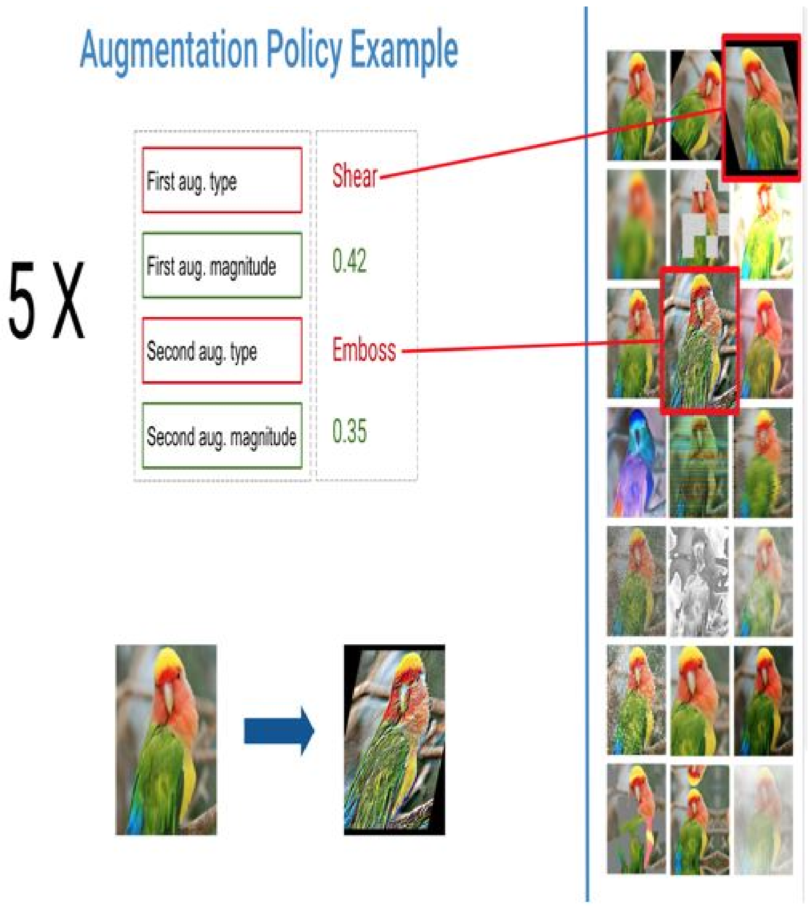


Fig. 4.5. Autoaugmentation Policy Example

Autoaugmentation is one of the data augmentation techniques. It automatically searches for a better augmentation policy that is suitable for the dataset. In policy, it consists of several sub-policies. Mainly sub policy consists of two functions. One function is the image processing function and the other function is the magnitude of image processing function that is applied. The image processing function can be translation, rotation, or shearing. The magnitude of image processing function is the probability of the image processing function.

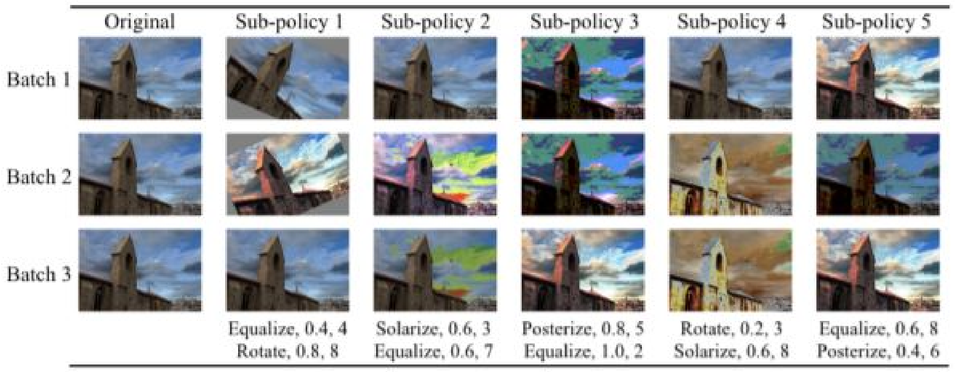


Fig. 4.6. Sub-Policy Example

From the figure, original image and sub-policies can be seen for an example. Each Sub-policy has both image processing function as well as probability values of that Image processing function.

5. RESULTS

The proposed model, RMNv2, is trained and tested from scratch on the CIFAR10 dataset. Then the results obtained are compared with the baseline network, Mobilenet V2, that is trained with the same set of hyperparameters and training setup. The loss accuracy curves are plotted using LiveLossPlot.

5.1 Training Setup

- **Framework** - Pytorch(Both baseline and Proposed work)
- **GPU** - Nvidia GeForce GTX 1080Ti
- **Optimizer** - SGD
- **learning rate** - variable learning rate 0.1, 0.01, 0.001
- **Total Number of Epochs** - 200
- **Batch size for Training** - 128
- **Batch size for Testing** - 64
- **Variable P in HetConv** - 4

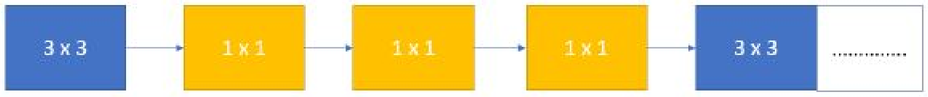


Fig. 5.1. HetConv for P = 4

5.2 Loss and Accuracy Curves

The baseline model, Mobilenet V2, is trained and tested from scratch on the CIFAR10 dataset. The log loss curves are plotted using the LiveLossPlot library that is available in Python.

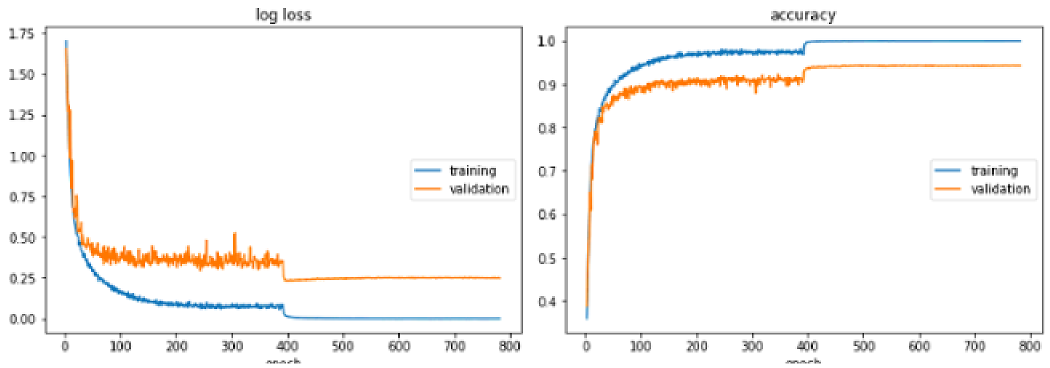


Fig. 5.2. Baseline Curves

Fig. 5.2 shows the baseline results curves. Log loss curve is shown on left. Accuracy curve is shown on right. On the Horizontal axis, it is the total number of epoch iterations. Upon completing the training process, the total accuracy, number of parameters, model size is shown in the following table.

Table 5.1.
Baseline Results

Model	**Model accuracy**	**Total no of Parameters**	**Model Size**
Mobilenet V2	94.3%	2.2378M	9.1MB

The Proposed model, RMN V2, is trained and tested from scratch on the CIFAR10 dataset. The log loss curves are plotted using the LiveLossPlot library that is available in Python.

Fig. 5.3 shows the Proposed RMNv2 results curves. Log loss curve is shown on left. Accuracy curve is shown on right. On the Horizontal axis, it is the total number of epoch iterations.

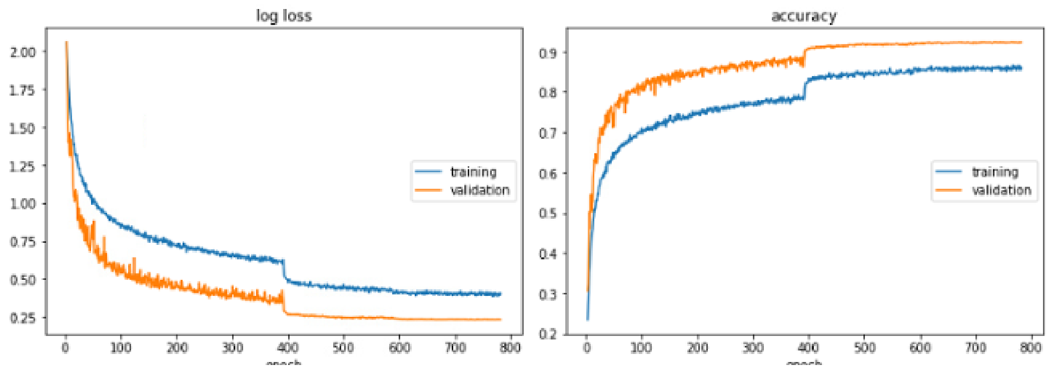


Fig. 5.3. Proposed RMNv2 Curves

Upon completing the training process, the total accuracy, number of parameters, model size is shown in the following table.

Table 5.2.
RMNv2 Results

Model	**Model accuracy**	**Total no of Parameters**	**Model Size**
RMNv2	92.4%	1.0691M	4.3MB

5.3 Comparison between baseline and Proposed model RMNv2

In this section, the proposed model, RMNv2 is compared with baseline model on various performance metrics.

Table 5.3.
Comparison of Various Results

Model	**Model accuracy**	**Total no of Parameters**	**Model Size**
Mobilenet V2	94.3%	2.2378M	9.1MB
RMNv2	92.4%	1.0691M	4.3MB

The time taken for one epoch as well as for complete training for baseline and proposed models is shown below,

Table 5.4.
Comparison of Time

Model	**For one epoch**	**For complete Training**
Mobilenet V2	1.907min	6.7hr
RMNv2	0.7621min	2.7hr

6. IMPLEMENTATION

In this chapter, the proposed architecture RMNv2 is implemented in real-time hardware like NXP Bluebox 2.0 and NXP i.MX RT1060.

6.1 Implementation on NXP Bluebox 2.0

The trained RMNv2 Pytorch model on GPU is tested in NXP Bluebox 2.0 using RTMaps. RTMaps supports python 3.6 for deploying the model on the hardware. RTMaps consists of a python module or python block that can allow the user to write python script in it.

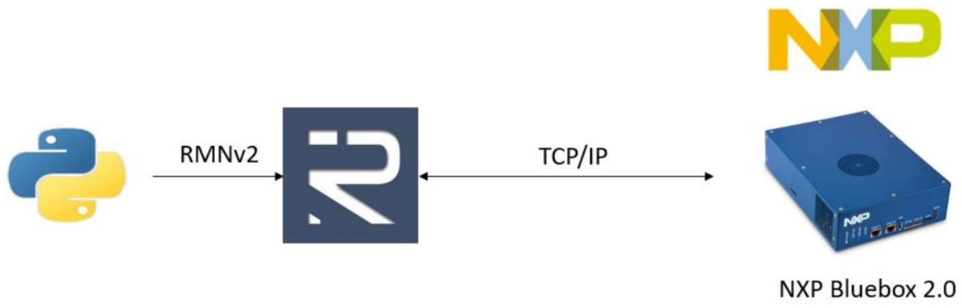
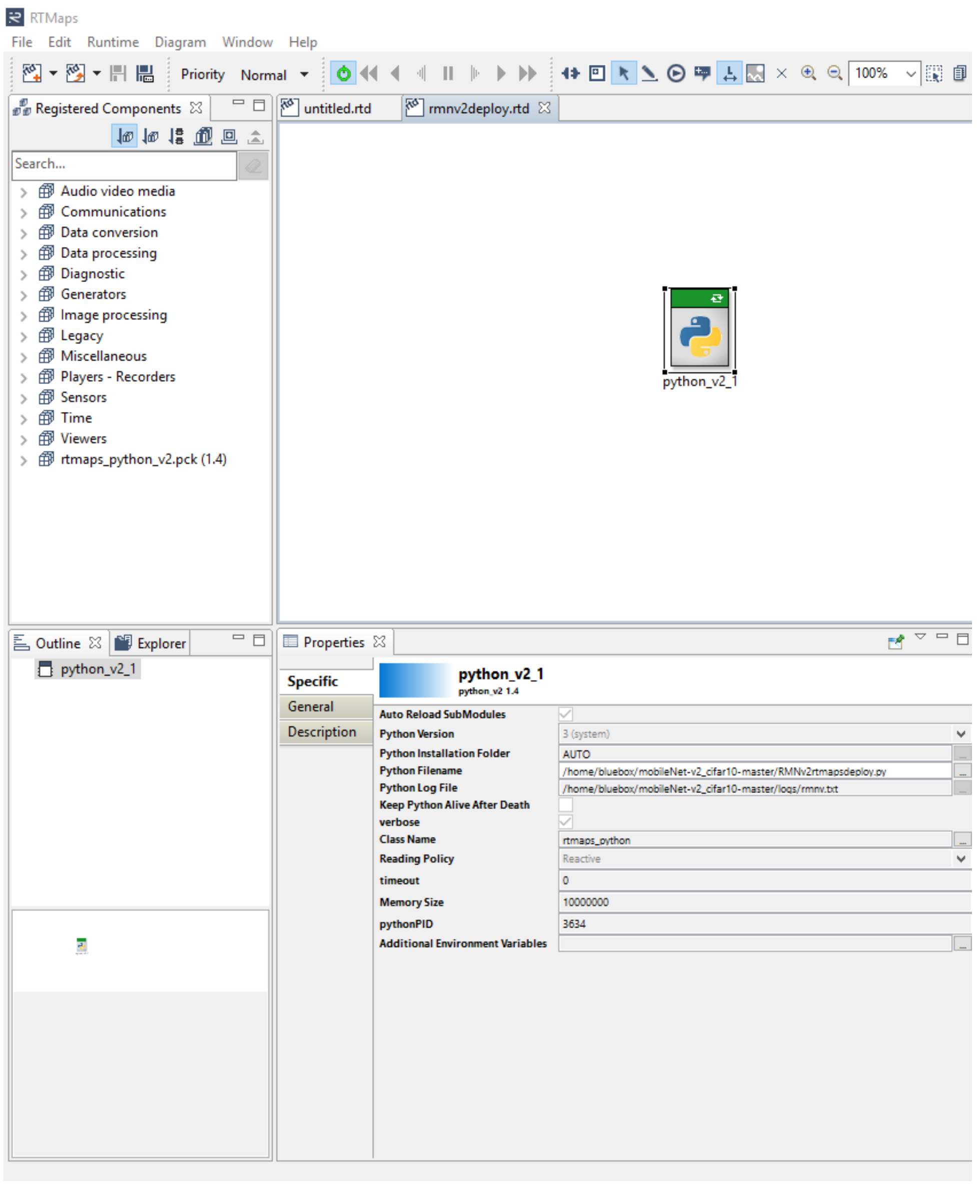


Fig. 6.1. Flowchart of RMNv2 Implementation in NXP Bluebox 2.0

Fig. 6.2 depicts the python component representation in RTMaps by Intempora. Due to its GUI based development environment, it is easy to write program using RTMaps. The python component in RTMaps has a text editor that allow users to edit their code very easily. In that editor, it consists of three functions. They are Birth(), Core() and death(). The functionality of these functions are really important to understand.



python_v2_1
python_v2 1.4

Auto Reload SubModules	✓
Python Version	3 (system)
Python Installation Folder	AUTO
Python Filename	/home/bluebox/mobileNet-v2_cifar10-master/RMNv2rtmapsdeploy.py
Python Log File	/home/bluebox/mobileNet-v2_cifar10-master/logs/rmnv.txt
Keep Python Alive After Death	
verbose	✓
Class Name	rtmaps_python
Reading Policy	Reactive
timeout	0
Memory Size	10000000
pythonPID	3634
Additional Environment Variables	

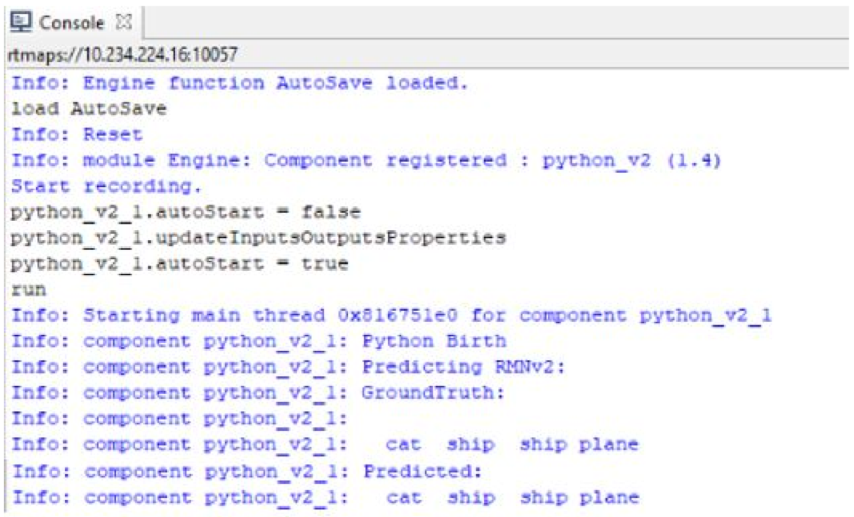
Fig. 6.2. Python Component Representation in RTMaps

- **Birth():** It is written at the starting of the code to initialize and setup the code.
- **Core():** It is a function that runs in infinite loop. Therefore, the user code is written in this function.
- **Death():** It is defined at the end, when the program is halted.

This structure of writing code makes it easier for the user to prototyping and developing their own code with respect to the application. Once the scripting is done, the user can use the RTMaps Embedded to run their application on the Bluebox Platform. The connection between the host pc and the target Bluebox is TCP/IP. After connecting to host pc, the user can check correct COM ports in the device manager. Then user should setup Teraterm for LS2 interface and S32V interface.

6.2 NXP Bluebox 2.0 Results

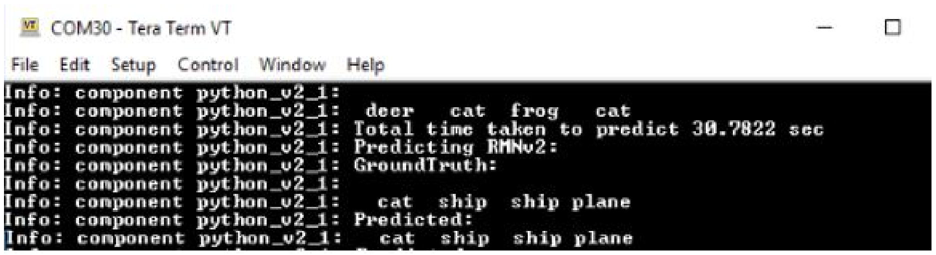
Here, the classifier is trying to work correctly using NXP Bluebox 2.0 platform. The model is fed with some random images from testset. These images are random images that are taken from different classes within the CIFAR10 dataset.



```
Console
rtmaps://10.234.224.16:10057
Info: Engine function AutoSave loaded.
load AutoSave
Info: Reset
Info: module Engine: Component registered : python_v2 (1.4)
Start recording.
python_v2_1.autoStart = false
python_v2_1.updateInputsOutputsProperties
python_v2_1.autoStart = true
run
Info: Starting main thread 0x816751e0 for component python_v2_1
Info: component python_v2_1: Python Birth
Info: component python_v2_1: Predicting RMNv2:
Info: component python_v2_1: GroundTruth:
Info: component python_v2_1:
Info: component python_v2_1:   cat  ship  ship plane
Info: component python_v2_1: Predicted:
Info: component python_v2_1:   cat  ship  ship plane
```

Fig. 6.3. RTMaps Console Output

Fig. 6.3 shows the RTMaps console output when the model is running on NXP Bluebox 2.0. Fig. 6.4 shows the teraterm output when the model is running on NXP Bluebox 2.0.



```
Info: component python_v2_1:
Info: component python_v2_1:  deer   cat  frog   cat
Info: component python_v2_1: Total time taken to predict 30.7822 sec
Info: component python_v2_1: Predicting RMNv2:
Info: component python_v2_1: GroundTruth:
Info: component python_v2_1:
Info: component python_v2_1:   cat  ship  ship plane
Info: component python_v2_1: Predicted:
Info: component python_v2_1:   cat  ship  ship plane
```

Fig. 6.4. Teraterm Output

6.3 Implementation on NXP i.MX RT1060

Implementing RMNv2 classifier in NXP i.MX RT1060 involves two steps, first to convert the model to Tensorflow lite model and deploying that tensorflow lite model onto the board.

- NXP provides a machine learning software development environment called eIQ. It is specifically designed to develop computer vision algorithms in embedded platforms like i. MX RT processors. NXP eIQ ML Software development environment has inference engines like OpenCV, Tensorflow lite, ARM NN and CMSIS-NN. In the TensorFlow lite inference engine, the pre-trained RMNv2 Keras model that is converted to tf lite model using tf lite converter.

- The MCU Xpresso SDK is specifically designed by NXP to accelerate application development in i. MX RT crossover processors. The latest version includes the updated eIQ libraries and demos. This SDK also supports UART debug console to run the application on Teraterm. The tflite model is converted into a C array header file (.h) that can be imported in an embedded project. The API call is used in the code to load the model using this header file. Then, the model is debugged and the output can be viewed in Teraterm.

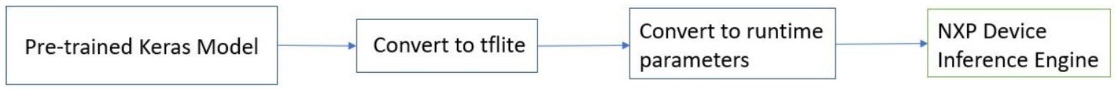


Fig. 6.5. Flowchart Representation of Deploying RMNv2 on NXP i.MX RT1060

6.4 NXP i.MX RT1060 Results

Random images taken from internet like cat and ship are given as input to the model. Generally, cat and ship is given because they belong to CIFAR10 class. The model will try to predict the input image given using the NXP i.MX RT1060. The output can be seen in Teraterm.

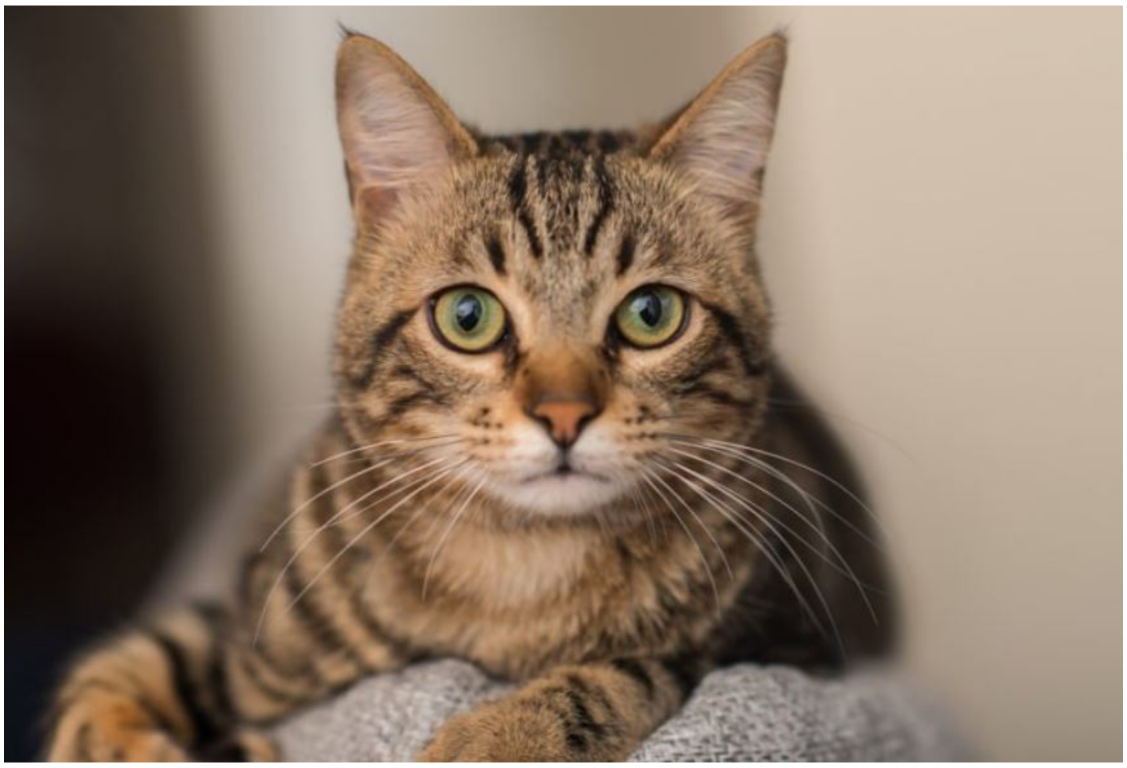
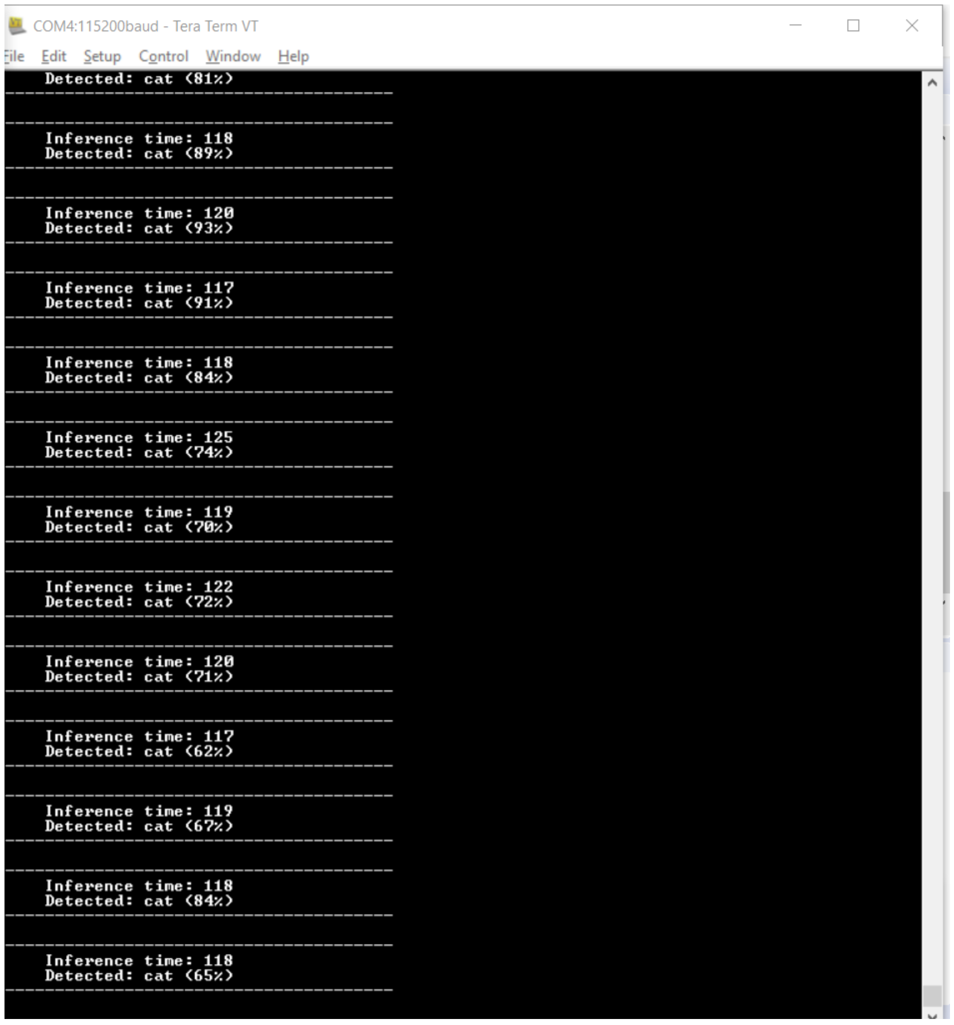


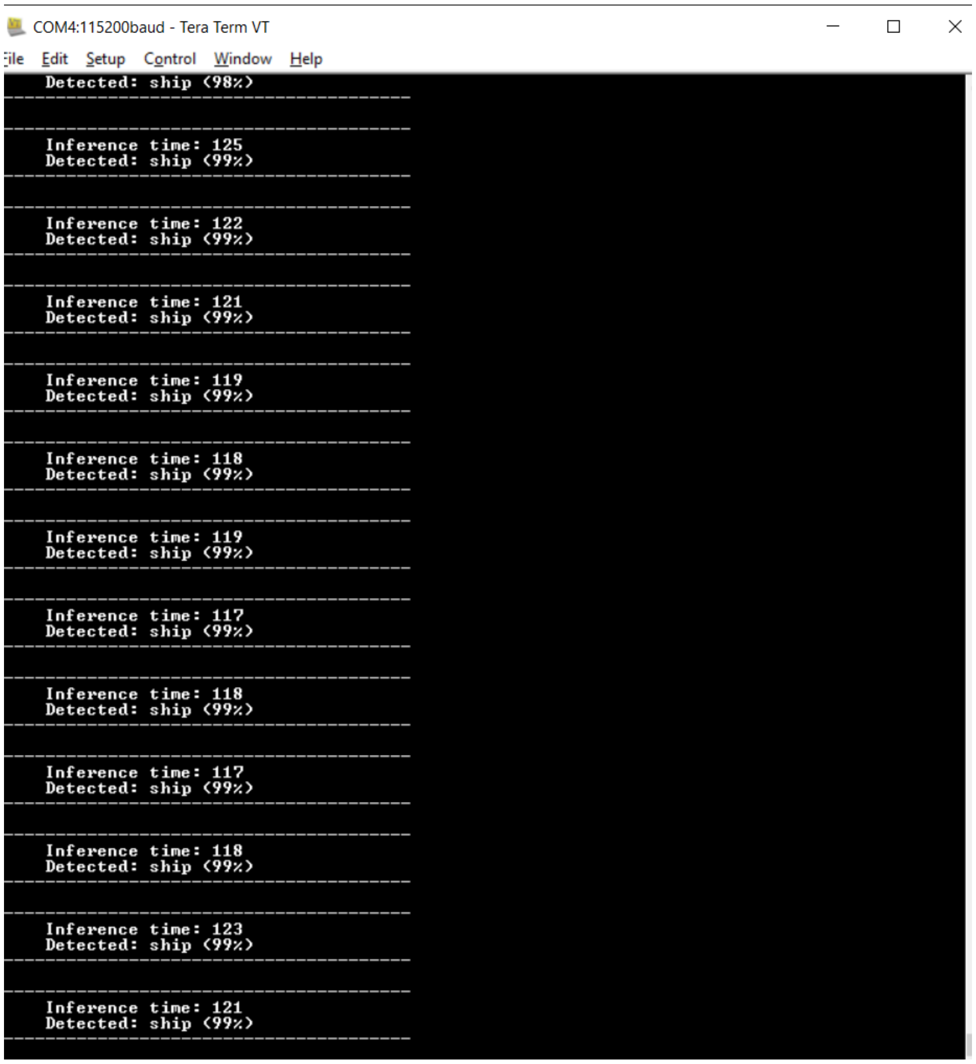
Fig. 6.6. Input Cat Image

Fig. 6.7. Input Ship Image


COM4:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
    Detected: cat (81%)
----------------------------------------

----------------------------------------
    Inference time: 118
    Detected: cat (89%)
----------------------------------------

----------------------------------------
    Inference time: 120
    Detected: cat (93%)
----------------------------------------

----------------------------------------
    Inference time: 117
    Detected: cat (91%)
----------------------------------------

----------------------------------------
    Inference time: 118
    Detected: cat (84%)
----------------------------------------

----------------------------------------
    Inference time: 125
    Detected: cat (74%)
----------------------------------------

----------------------------------------
    Inference time: 119
    Detected: cat (70%)
----------------------------------------

----------------------------------------
    Inference time: 122
    Detected: cat (72%)
----------------------------------------

----------------------------------------
    Inference time: 120
    Detected: cat (71%)
----------------------------------------

----------------------------------------
    Inference time: 117
    Detected: cat (62%)
----------------------------------------

----------------------------------------
    Inference time: 119
    Detected: cat (67%)
----------------------------------------

----------------------------------------
    Inference time: 118
    Detected: cat (84%)
----------------------------------------

----------------------------------------
    Inference time: 118
    Detected: cat (65%)
----------------------------------------
```

Fig. 6.8. Output for Cat Image


COM4:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
    Detected: ship (98%)
--------------------------------------

--------------------------------------
    Inference time: 125
    Detected: ship (99%)
--------------------------------------

--------------------------------------
    Inference time: 122
    Detected: ship (99%)
--------------------------------------

--------------------------------------
    Inference time: 121
    Detected: ship (99%)
--------------------------------------

--------------------------------------
    Inference time: 119
    Detected: ship (99%)
--------------------------------------

--------------------------------------
    Inference time: 118
    Detected: ship (99%)
--------------------------------------

--------------------------------------
    Inference time: 119
    Detected: ship (99%)
--------------------------------------

--------------------------------------
    Inference time: 117
    Detected: ship (99%)
--------------------------------------

--------------------------------------
    Inference time: 118
    Detected: ship (99%)
--------------------------------------

--------------------------------------
    Inference time: 117
    Detected: ship (99%)
--------------------------------------

--------------------------------------
    Inference time: 118
    Detected: ship (99%)
--------------------------------------

--------------------------------------
    Inference time: 123
    Detected: ship (99%)
--------------------------------------

--------------------------------------
    Inference time: 121
    Detected: ship (99%)
--------------------------------------
```

Fig. 6.9. Output for Ship Image

7. CONCLUSIONS

This thesis concludes by proposing an architecture, RMNv2. The model is trained and tested on the CIFAR10 dataset. The model is an optimized version of the baseline Mobilenet V2 model for the CIFAR10 dataset. The model achieved an overall accuracy of 92.4% with 1.9 less than the baseline accuracy but with a model size of 4.3MB and a total number of parameters of 1.06M. The model size and number of parameters are 52.2% lesser than the baseline model. This compact model size and competitive accuracy make it suitable to deploy in embedded and mobile devices. The model is also tested on real-time hardware like NXP Bluebox 2.0 and NXP i.MX RT1060. Some of the key changes that made our model stand out is using Heterogeneous kernels on the convolutions, changing strides, mish activation, and autoaugmentation.

Therefore, the proposed model, RMNv2, can be used in ADAS applications like object detection, lane detection, pedestrian detection, steering control, Traffic sign classification, etc. It is flexible, lightweight and compact making it suitable to deploy in resource-constrained devices like mobile and embedded devices. The model can be further expanded to various applications that is explained in the next chapter, future scope.

8. FUTURE SCOPE

The proposed model is an architecturally modified version of the baseline model. However, the model size can be further increased or decreased depending upon the application using hyperparameters like width and resolution multiplier. The model can be extended to the application like object detection. It can detect objects in an image using algorithms like SSD, YOLO, etc. The inference can be performed on other embedded platforms like the NXP 8M family, etc.

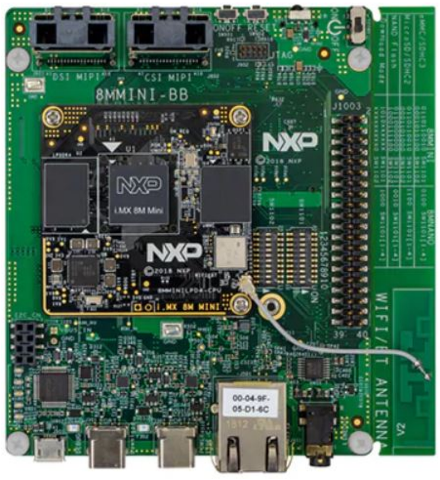


Fig. 8.1. NXP 8M Mini-EVK

REFERENCES

REFERENCES

[1] Andrew Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (Last date accessed: 03/25/2020) (2017).

[2] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen. "MobileNetV2: Inverted Residuals and Linear Bottlenecks." arXiv preprint arXiv:1801.04381v4 (Last date accessed: 03/25/2020) (2019).

[3] Pravendra Singh, Vinay Kumar Verma, Piyush Rai, Vinay Namboodiri. "HetConv: Heterogeneous Kernel-Based Convolutions for Deep CNNs" arXiv preprint arXiv:1903.04120v2 (Last date accessed: 03/24/2020) (2019).

[4] Diganta Misra, "Mish: A Self Regularized Non-Monotonic Neural Activation Function" arXiv preprint arxiv:1908.08681 (Last date accessed: 03/22/2020) (2019).

[5] Ekin Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, Quoc Le , "AutoAugment: Learning Augmentation Strategies from Data" arXiv preprint arXiv:1805.09501v3 (Last date accessed: 03/22/2020) (2019).

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, " Imagenet classification with deep convolutional neural networks", Advances in Neural Information Processing Systems 25, Curran Associates, Inc., pages 1097–1105, January, 2012 (Last date accessed: 03/22/2020).

[7] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein. Imagenet large scale visual recognition challenge, International Journal of Computer Vision (2015).

[8] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. "Quantized convolutional neural networks for mobile devices". arXiv preprint arXiv:1512.06473 (Last date accessed: 03/21/2020) (2015).

[9] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. "Compressing neural networks with the hashing trick". CoRR, abs/1504.04788(Last date accessed: 03/22/2020) (2015).

[10] Song Han, Huizi Mao, and William Dally."Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding". CoRR, abs/1510.00149, February, 2015 (Last date accessed: 03/22/2020).

[11] Karen Simonyan, Andrew Zisserman. "Very Deep Convolutional Networks For Large-Scale Image Recognition" arXiv preprint arXiv:1409.1556v6 (Last date accessed: 03/23/2020) (2015).

[12] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. ”Distilling the knowledge in a neural network”. arXiv preprint arXiv:1503.02531, 2015 (Last date accessed: 03/23/2020).

[13] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. ”Speeding up convolutional neural networks with low rank expansions.” arXiv preprint arXiv:1405.3866, 2014 (Last date accessed: 03/25/2020).

[14] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. ”Speeding-up convolutional neural networks using fine-tuned cp-decomposition.” arXiv preprint arXiv:1412.6553, 2014 (Last date accessed: 03/23/2020).

[15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. ”Rethinking the inception architecture for computer vision.” arXiv preprint arXiv:1512.00567, 2015 (Last date accessed: 03/23/2020).

[16] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. arXiv preprint arXiv:1602.07261, 2016 (Last date accessed: 03/24/2020).

[17] Forrest Iandola, Matthew Moskewicz, Khalid Ashraf, Song Han, Bill Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 1mb model size. arXiv preprint arXiv:1602.07360, 2016 (Last date accessed: 03/24/2020).

[18] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, Kurt Keutzer SqueezeNext: Hardware-Aware Neural Network Design arXiv preprint arXiv:1803.10615v2 (Last date accessed: 03/24/2020).

[19] Xiangyu Zhang, Xinyu Zhou, Mengxiao LinJian, Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile. arXiv preprint arXiv:1707.01083v2, 2017 (Last date accessed: 03/24/2020).

[20] Francois Chollet (2017). Xception: Deep Learning with DepthwiseSeparable Convolutions. arXiv preprint arXiv: 1610.02357 (Last date accessed: 03/25/2020).

[21] Gao Huang, Shichen Liu, Laurens van der Maaten. CondenseNet: An Efficient DenseNet using Learned Group Convolutions. arXiv preprint arXiv: 1711.09224, 2017 (Last date accessed: 03/25/2020).

[22] Sreeram Venkitachalam, Surya Kollazhi Manghat, Akash Sunil Gaikwad, Niranjan Ravi, Sree Bala Shruthi Bhamidi and Mohamed El-Sharkawy. Realtime Applications with RTMaps and Bluebox 2.0, ICAI'18.

[23] NXP bluebox 2.0 ”nxp.com/design/development-boards/automotive-development-platforms/nxp-bluebox-autonomous-driving-development-platform:BLBX” (Last date accessed: 04/01/2020)

[24] NXP i.MX RT1060 EVK ”nxp.com/design/development-boards/i-mx-evaluation-and-development-boards/mimxrt1060-evk-i-mx-rt1060-evaluation-kit:MIMXRT1060-EVK” (Last date accessed: 04/01/2020)