Real-time Implementation of RMNv2 Classifier in NXP Bluebox 2.0 and NXP i.MX RT1060

Maneesh Ayi
Department of ECE
Purdue School of Engineering and Tech.
Indianapolis, USA
maayi@iu.edu

Mohamed El-Sharkawy
Department of ECE
Purdue School of Engineering and Tech.
Indianapolis, USA
melshark@iupui.edu

***Abstract*—With regards to Advanced Driver Assistance Systems in vehicles, vision and image-based ADAS is profoundly well known since it utilizes Computer vision algorithms, for example, object detection, street sign identification, vehicle control, impact cautioning, and so on., to aid sheltered and smart driving. Deploying these algorithms directly in resource-constrained devices like mobile and embedded devices etc. is not possible. Reduced Mobilenet V2 (RMNv2) is one of those models which is specifically designed for deploying easily in embedded and mobile devices. In this paper, we implemented a real-time RMNv2 image classifier in NXP Bluebox 2.0 and NXP i.MX RT1060. Because of its low model size of 4.3MB, it is very successful to implement this model in those devices. The model is trained and tested with the CIFAR10 dataset.**

***Index Terms*—Convolution Neural Network (CNN), Deep Neural Network (DNN), CIFAR-10, NXP Bluebox 2.0, NXP i.MX RT1060, RTMaps, S32v234, Teraterm.**

I. Introduction

Computer Vision, which is abbreviated as CV is characterized as a field of concentrate that creates strategies to enable computers to comprehend the content involved in digital images such as images and videos. The convolutional neural network is the most important and popular topic in the field of Computer vision. CNN's are introduced in the year 2012 when Alexnet[6] winning the Imagenet Challenge[7]. There have been several neural networks developed and introduced later on.

In this paper, we have used Reduced MobilenetV2. Reduced MobilenetV2 abbreviated as RMNv2[17] is the modified version of Mobilenet V2. It includes changes like changing strides, Heterogenous Kernel-based Convolution block[3], mish activation function[4], and autoaugmentation[5]. These changes were done in particular to CIFAR10 dataset so that the model size is decreased by 52% without much affecting the performance of the model. The architectural representation of RMNv2 is shown below in Table 1,

The flowchart representation of RMNv2 architecture is shown below in Figure 1.

Some of the key features of RMNv2 is shown in Table 2.

Input	operator	t	c	n	s
$224^2 \times 3$	Conv2D	-	32	1	1
$112^2 \times 32$	Bottleneck	1	16	1	1
$112^2 \times 16$	Bottleneck	6	24	2	1
$56^2 \times 24$	Bottleneck	6	32	3	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	-	

TABLE I: Architectural representation of RMNv2

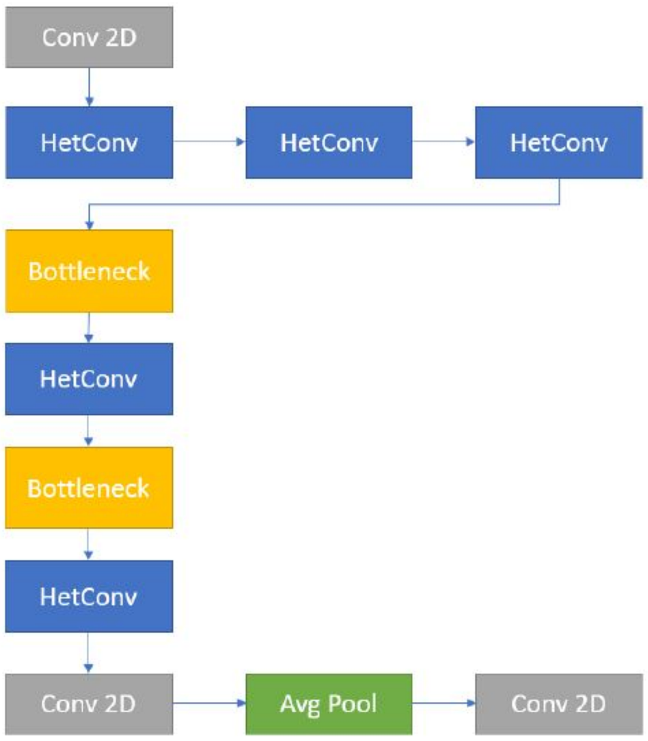


Fig. 1: Flowchart representation of RMNv2 Architecture

Model Accuracy	92.4%
#parameters	1.0691M
Model_Size(in MB)	4.3 MB

TABLE II: Key features of RMNv2

In applications such as ADAS systems, robotics etc. we need a framework that is easy to understand and should be flexible to make the application work faster. One of those frameworks is RTMaps. It is often called as real-time multi-sensor applications. It is made of a different module to be utilized in various settings - RTMaps Runtime Engine, RTMaps Studio, The RTMaps Component Library and RTMaps SDK. It gathers information from various devices such as cameras, radars and lidars etc. RTMaps V4.5.0 makes the software run on embedded targets. So, one of those high computational targets is NXP Bluebox 2.0. The RTMaps embedded available on NXP Bluebox 2.0 helps us to develop applications such as ADAS systems etc. NXP Bluebox 2.0 is a development platform that gives required performance, reliable functionality to design an autonomous vehicle. NXP Bluebox 2.0 consists of S32V234 for car vision and sensor fusion microprocessor, the LS2084A embedded computer processor and the S32R27 radar microcontroller. The proposed algorithm in host PC uses an SSL connection to deploy in NXP Bluebox 2.0 .

NXP i. MX RT1060 is the first crossover MCU series. It is supported by NXP's MCUXpresso Software's and tools. It can compute computer vision algorithms with low latency. The eIQ machine learning software helps us developing machine learning algorithms in i. MX RT1060.

II. Prior Work

There have been many architectures introduced after Alexnet. Some of those networks are VGG[11], Inception[12][13] etc. These are more accurate than Alexnet but also complicated. In order to implement these architectures in real-time is not possible because of the high model size. Then, there has been a lot of active research going on to develop architectures that are compatible to deploy in real-time devices without compromising on accuracy. Out of which, two methods are described in developing these small models. One is developing a small model from scratch or compressing a large network. Quantization[8], hashing[9], Pruning, vector quantization and Huffman encoding[10] etc. These methods are used to Compress a large network. Networks like Squeezenet[14] and Squeezenext[15] are small models that are developed from scratch. These networks didn't focus on speed. Mobilenet V1[1] and Mobilenet V2[2] are introduced which are not only small in size but also focused on speed as well. [16] provides some insights into various applications with RTMaps and blue box.

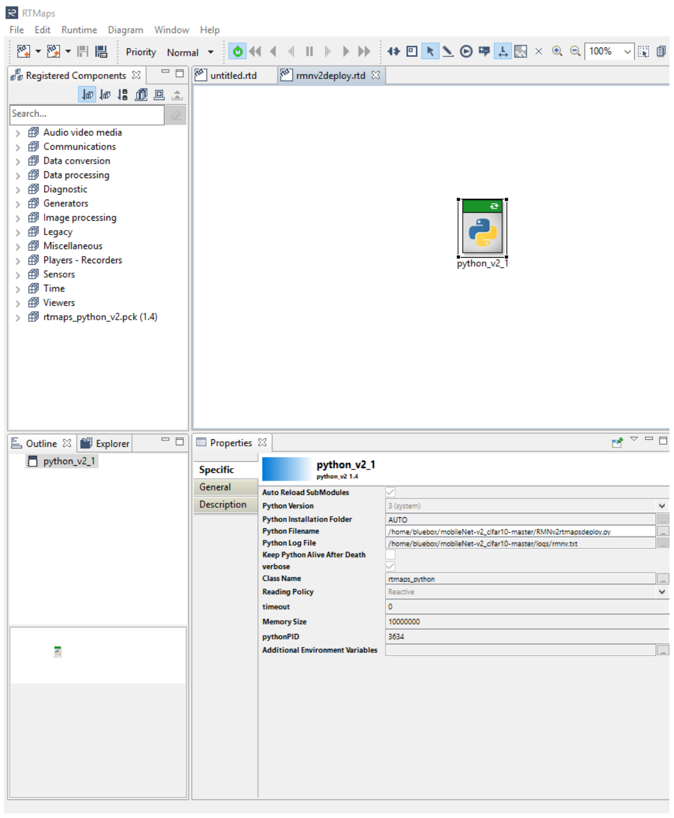
III. Implementation

In this section, we discuss the implementation of the RMNv2 Image classifier in real-time devices like NXP Bluebox 2.0 and NXP i.MX RT1060.

A. RMNv2 Classifier in NXP Bluebox 2.0

The python component in RTMaps will allow us to develop and integrate computer vision algorithms for ADAS applications like Image classification, sign identification, and driving assistance etc. The python component in RTMaps has an editor in it that allows users to create, develop and deploy their python scripts. In this editor, there are three main functions that are important to know in order to implement users python script in hardware. Birth(), Core() and Death() are the three functions that are available in the editor. Birth() is executed once at the beginning to initialize and set up the code. Core() is a function that runs in an infinite loop. Therefore, the user's code can be defined in this section that allows code to run continuously. Death() is defined at the end and it is called when the program is halted.

The python componenet in RTMaps is shown in the below figure 2.



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. 2: Python Component representation in RTMaps

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. Figure-3 shows the diagrammatic representation of

RTMaps setup with Bluebox 2.0. 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. In this paper, the classifier is trained only in GPU but tested in NXP Bluebox 2.0.

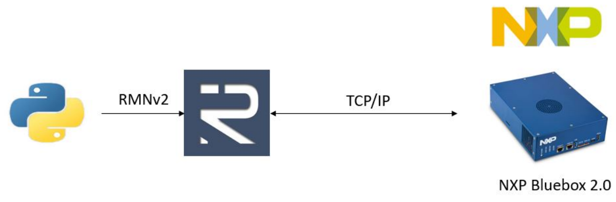


Fig. 3: Diagrammatic representation of RTMaps with NXP Bluebox 2.0

B. *RMNv2 Classifier in NXP i.MX RT1060*

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

1) Converting into Tensorflow lite Model: 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, we have our pre-trained RMNv2 Keras model that is converted to tf lite model using tf lite converter. Flowchart representation of this process is shown in Figure-4.

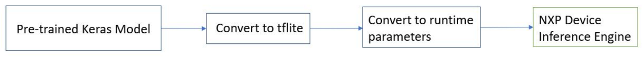


Fig. 4: Flow chart representation of RMNv2 classifier implementation in i.MX RT1060

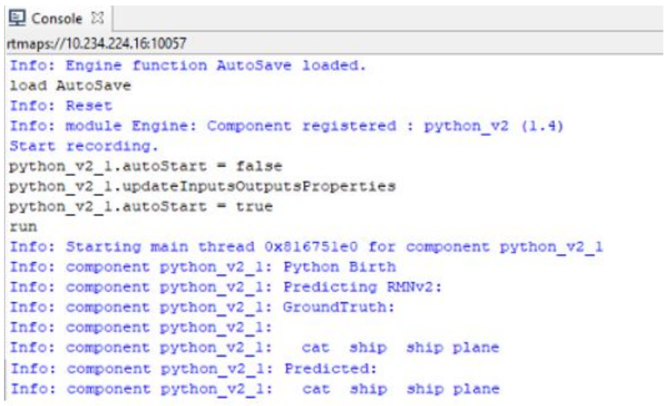
2) Deploying in i.MX RT1060: The MCU Xpresso SDK is specifically designed by NXP to accelerate application development in i. MX RT crossover processors. Latest version of the SDK includes the updated eIQ libraries and demos. This SDK additionally supports UART debug console to run the application on Teraterm. TFlite model is converted into a C array (.h) header file that can be called 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 we can view the output in Teraterm.

IV. Results

In this paper, we have taken a pretrained RMNv2 model. The model is trained and tested for the CIFAR10 dataset. The model is trained in Nvidia Geforce GTX 1080Ti GPU. The original model is trained using the Pytorch framework with a total number of epochs to 200 and with a decreasing learning rate of 0.1, 0.01 and 0.001. We have used Stochastic gradient descent (SGD) optimizer. The batch size for training the network is 128 and for the test set, it is 64. We have replicated a similar configuration to the model with Keras as well. Let us see the results of RMNv2 image classifier with corresponding boards.

A. *With NXP Bluebox 2.0*

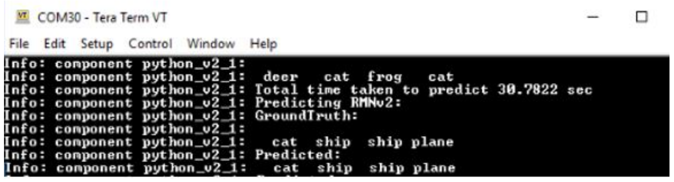
Here, we are attempting to make classifier work correctly in NXP Bluebox 2.0. So, the model is fed with some random images taken from the test dataset with correct ground truth values and asking the model to predict those random images. The RTMaps Console output is shown below in Figure 5. The bluebox output can be seen using



```
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. 5: RTMaps Console output of RMNv2 Classifier

Teraterm terminal. The Teraterm output can be seen below in figure 6. The model is given some random input images like



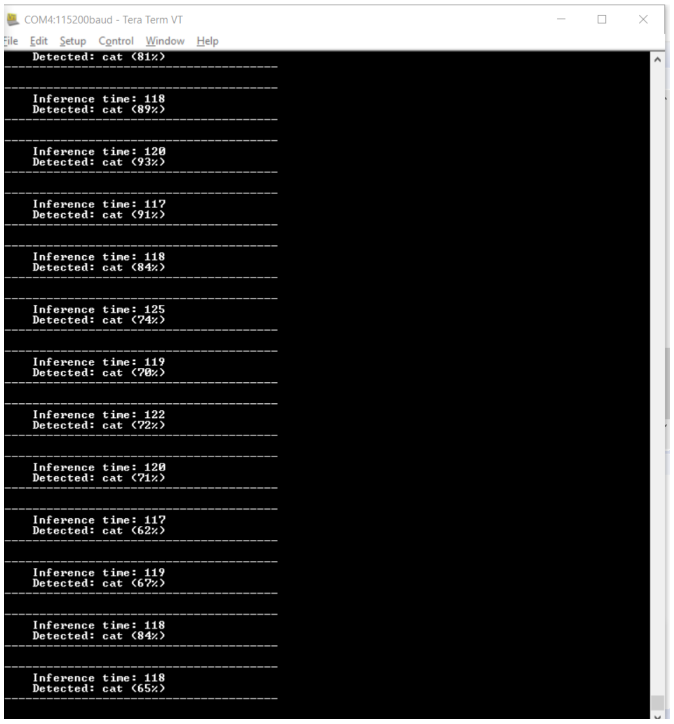
```
Info: component python_v2_1:
Info: component python_v2_1:  deer   cat  frog   cat
Info: component python_v2_1: Total time taken to predict 30.7822 sec
Info: component python_v2_1: Predicting RMNv2:
Info: component python_v2_1: GroundTruth:
Info: component python_v2_1:
Info: component python_v2_1: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

Fig. 6: Teraterm output of RMNv2 Classifier for NXP Bluebox 2.0

cat, ship and plane. It correctly predicts those images in NXP Bluebox 2.0 embedded platform.

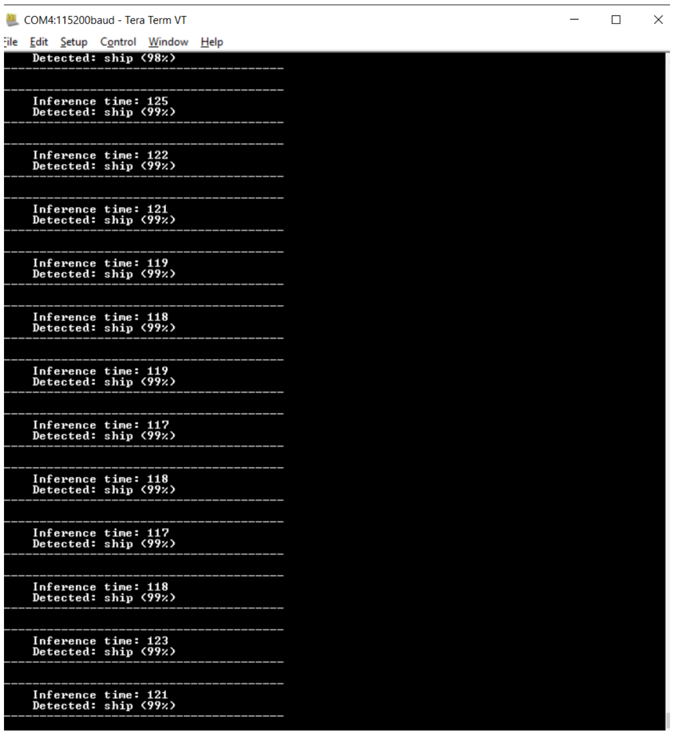
B. *With NXP i.MX RT1060*

In this section, we tried to give some random images like cat and ship and asking the model to predict them. Generally, we gave cat and ship because these classes belong to the CIFAR10 dataset and our model is only trained to this dataset. The output can be seen in Teraterm terminal. Our model is correctly able to classify cat and ship image correctly in NXP i.MX RT1060 embedded platform along with inference times shown in the figures 7 and 8.


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. 7: Teraterm output corresponding to 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. 8: Teraterm output corresponding to Ship Image

V. Conclusion

Finally, we have presented some real-time image classifier application in high computational and flexible embedded platforms like NXP Bluebox 2.0 and NXP i. MX RT1060. These devices are highly suitable for autonomous applications like ADAS systems etc. However, the model can be further reduced using techniques like model pruning, model compression and quantization etc. Also, in this paper, we did not perform any hyperparameter tweaking like changing an alpha parameter or changing the resolution multiplier. The model can also be developed further for applications like object detection and object tracking etc. The applications can also be developed and tested on other NXP Platforms like other i.MX family like i.MX 8M family etc. The inference time can be further reduced as well.

References

[1] Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).

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

[3] Pravendra Singh, Vinay Kumar Verma, Piyush Rai, Vinay P. Namboodiri. "HetConv: Heterogeneous Kernel-Based Convolutions for Deep CNNs" arXiv preprint arXiv:1903.04120v2 (2019)

[4] Diganta Misra, "Mish: A Self Regularized Non-Monotonic Neural Activation Function" arXiv preprint arxiv:1908.08681 (2019)

[5] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, Quoc V. Le ´Google Brain, "AutoAugment: Learning Augmentation Strategies from Data" arXiv preprint arXiv:1805.09501v3 (2019)

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, " Imagenet classification with deep convolutional neural networks" In Advances in neural information processing systems, pages 1097–1105, 2012.

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge, International Journal of Computer Vision, 2015

[8] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. "Quantized convolutional neural networks for mobile devices". arXiv preprint arXiv:1512.06473, 2015.

[9] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. "Compressing neural networks with the hashing trick". CoRR, abs/1504.04788, 2015

[10] S. Han, H. Mao, and W. J. Dally."Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding". CoRR, abs/1510.00149, 2, 2015.

[11] Karen Simonyan, Andrew Zisserman. "VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION" arXiv preprint arXiv:1409.1556v6 (2015)

[12] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. "Rethinking the inception architecture for computer vision." arXiv preprint arXiv:1512.00567, 2015.

[13] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. arXiv preprint arXiv:1602.07261, 2016

[14] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 1mb model size. arXiv preprint arXiv:1602.07360, 2016.

[15] 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

[16] 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

[17] Maneesh Ayi.(2020). RMNv2: Reduced Mobilenet V2 An Efficient Lightweight Model For Hardware Deployment [Master's Thesis, IUPUI]. ScholarWorks Electrical and Computer Engineering department Theses and Dissertations Publishing.