SQUEEZE-AND-EXCITATION SQUEEZENEXT: AN EFFICIENT DNN FOR HARDWARE DEPLOYMENT

A Thesis

Submitted to the Faculty

of

Purdue University

by

Naga Venkata Sai Raviteja Chappa

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

Dr. Mohamed El-Sharkawy, Chair

Department of Electrical and Computer Engineering

Dr. Brian King

Department of Electrical and Computer Engineering

Dr. Maher Rizkalla

Department of Electrical and Computer Engineering

Approved by:

Dr. Brian King

Head of School Graduate Program

This thesis is dedicated to my parents Koteswara Rao Chappa & Jhansi Rani Bandi, I would also like to thank my grandmother Nara Venkata Bharathi, uncle Kishor Kumar Bandi and aunt Hyma Bandi for their immense support throughout my journey of education.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Mohamed El-Sharkawy for his immense support and motivation throughout the research, which made my hardwork look easy.

To Dr. Brian King and Dr. Maher Rizkalla for being part of thesis committee and their valuable comments.

To Sherrie Tucker for her continuous support in meeting all my academic and thesis deadlines.

Last but not the least; I would like to extend my hearty thanks to Maheswari Adimulam and entire IoT Collaboratory for believing in me and their constant support throughout this journey.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Chappa, Naga Venkata Sai Raviteja. M.S.E.C.E., Purdue University, May 2020.
Squeeze-and-Excitation SqueezeNext: An Efficient DNN for Hardware Deployment.
Major Professor: Dr. Mohamed El-Sharkawy.

Convolution neural network is being used in field of autonomous driving vehicles or driver assistance systems (ADAS), and has achieved great success. Before the convolution neural network, traditional machine learning algorithms helped the driver assistance systems. Currently, there is a great exploration being done in architectures like MobileNet, SqueezeNext & SqueezeNet. It improved the CNN architectures and made it more suitable to implement on real-time embedded systems.

This thesis proposes an efficient and a compact CNN to ameliorate the performance of existing CNN architectures. The intuition behind this proposed architecture is to supplant convolution layers with a more sophisticated block module and to develop a compact architecture with a competitive accuracy. Further, explores the bottleneck module and squeezenext basic block structure. The state-of-the-art squeezenext baseline architecture is used as a foundation to recreate and propose a high performance squeezenext architecture. The proposed architecture is further trained on the CIFAR-10 dataset from scratch. All the training and testing results are visualized with live loss and accuracy graphs. Focus of this thesis is to make an adaptable and a flexible model for efficient CNN performance which can perform better with the minimum tradeoff between model accuracy, size, and speed. Having a model size of 0.595MB along with accuracy of 92.60% and with a satisfactory training and validating speed of 9 seconds, this model can be deployed on real-time autonomous system platform such as Bluebox 2.0 by NXP.

1. INTRODUCTION

The greater part of the applications continuously, for example, PC vision, apply autonomy, picture acknowledgment and characterization, self-governing vehicles and ADAS have been changed with assistance of Deep Neural Networks. This has been made conceivable by experiencing profound research in this field over the previous decade with the accessibility of additionally preparing information, and for preparing and approval having quicker equipment. Yet, not incredible measure of work is done in parts of model size and speed. There is a drawback to DNNs that it require more spending plan of assets that alludes to more calculation and memory assets. Most as of late, DNN achieved a confusing benchmark of precision at 99% with GPipe. With the development of large scale structures, for example, SqueezeNet, SqueezeNext and MobileNet, DNNs can be actualized on implanted frameworks. SqueezeNet utilizes the fire module's press and extend layer way to deal with plan a littler and shallow CNN design however it includes some significant downfalls of model precision which is about 78%. However, SqueezeNext accomplishes better outcomes yet it despite everything can be improved as recommended by the creator of this engineering with further hyperparameter tuning and adjustments. This examination proposes a proficient system engineering so as to a form exceptionally little, effective DNN model that is the proposed Squeeze-and-Excitation SqueezeNext design.

1.1 Motivation

ML is a part of artificial intelligence, it is another cutting edge artificial intelligence strategy where as opposed to looking through the hard-coded highlights of a picture, a machine is urged to gain proficiency with the picture highlights during the DNN preparing. This methodology is closely resembling a human kid figuring out.

Further, investigating the DSE outcomes of CNNs makes this exploration adding to field of deep learning. Picture acknowledgment, characterization, following, identification, and division are a portion of the difficult undertakings. The best in class DNN models matches the human grouping exactness however a bit much regarding model inactivity, speed and size. Nonetheless, there will be a huge memory overhead if the model size is in GBs or considerably more prominent than hardly any MBs while deploying in real-time. Despite of incredible Graphical Processing Units(GPUs) being a solution yet that is not minimized and in present there is no appropriate innovation available to utilize them inside edge gadgets. While keeping up the serious precision of deep neural network models, this shows a reasonable requirement to develop a least size of the model, speed of the model and inertness of time.

1.2 Challenges

- Requires a small model size with good accuracy.
- Requires the reduced model deployment on autonomous systems.
- Needs rapid training and validating of CNN.

1.3 Contributions

- Proposed Squeeze-and-Excitation SqueezeNext to reduce size and improve accuracy.
- Proposed architecture deployed successfully on both NXP Bluebox2.0 and NXP iMX-RT1060 EVKB.
- Two research conference papers.

2. BACKGROUND

The basic methodology of a convolutional neural network and its terminology are discussed in this chapter. In addition to, the development of the propopsed architecture i.e., Squeeze-and-Excitation SqueezeNext which is led by different baseline architecutures are reviewed. As well as, to get deep insights on deep neural networks which is helped various methodologies are discussed.

2.1 Components of CNNs

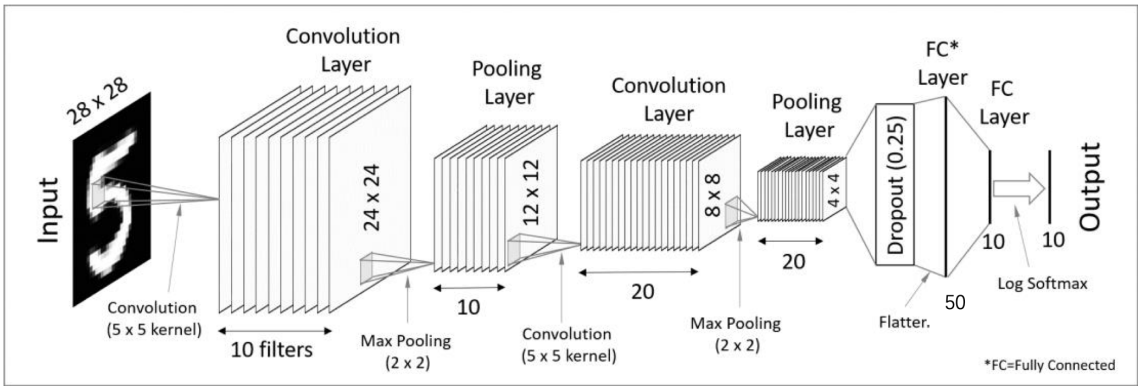


Fig. 2.1. The Structure of CNN.

In ML, we mostly deal with terms DNN and CNN. In general, convolutional neural network is a part of deep neural network. For the purpose of classifying the images and examining the videos deep neural networks are uniquely utilized. Features are learned inside a convolutional neural network when an image or a video is given as an input. The structure of convolutional neural network is shown in Fig 2.1. The four key components of a CNN are explained as below:

- Convolution layer.
- Activation layer.
- Pooling layer.
- Fully Connected layer.

The basic blocks of convolutional neural networks or deep neural network architectures are the above components.

The above components are the building blocks of any convolutional or deep neural network architectures. The number of parameters within the CNN will be increased when there is increase in the depth and width multipliers, this is because of the increase in the above four component layers in the model. The intuition of the above components is useful for the developing the proposed architecture and will be described below.

2.2 Image Given as Input

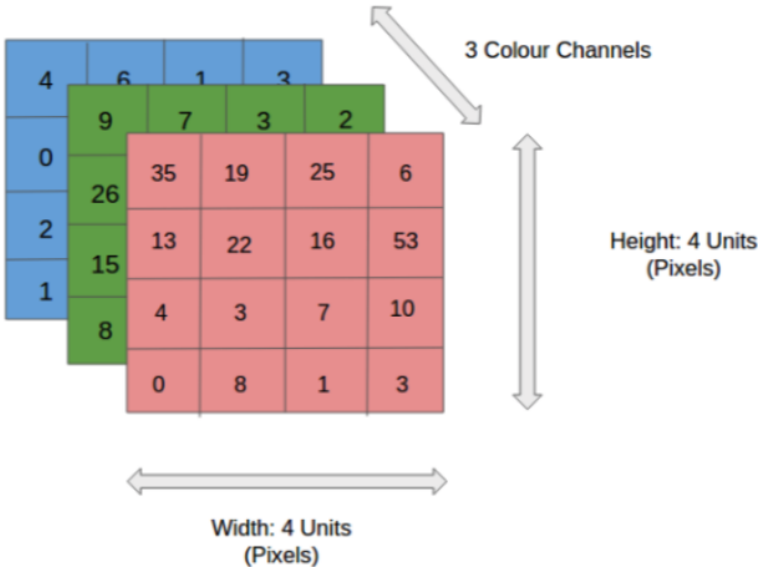


Fig. 2.2. Image Given as Input.

For a convolutional neural network, image can be given as an input with any size of pixels however if we consider CIFAR-10 as a dataset input should be of 32x32 size with 3 channels. In this dataset, there are 10 classes of 32x32 pictures. Color planes can be given as an input to a convolutional neural network in different ways like RGB, CMYK, grayscale and so on. Reducing the channel latttices of the image for the purpose of extracting the features is the main aim of convolution. At the end, the responsibility of the above learned features is to have predictions of the classes. So designing the architectures is critical with respect to few parameters by maintaining good accuracy of model.

2.3 Convolution Layer

To extract the features from a given input image, convolution layer is used. Firstly, it separates the features which are on the low level from the convolution layers that are initial.Subsequently, the middle level features are retrieved from convolutions on middle level and the features on the higher level are obtained in the final convolution layers in the convolution neural network model. Fig. 2.3 shows the development of a channel or a part along an input channel.

2.3.1 Parameters of Convolution Layer

There are three other parameters, which controls the feature map size which is to decided before performing the convolution:

- **Depth** is the intended filter number that are to be utilized for convolution and the number of parameters in a CNN is also effected by this parameter.
- **Stride** is the number of pixel values by which the kernel slide over the input matrix. The kernel moves by one pixel if the stride value is 1, if the value is two or three it will move two or three pixels. In general, we use stride value one for small datasets and two for large datasets.

- **Padding** is method to accumulate extra pixels along the edges of input image. Zero padding is mostly used though various types of padding are available. The size of the feature maps can be controlled with this parameter.

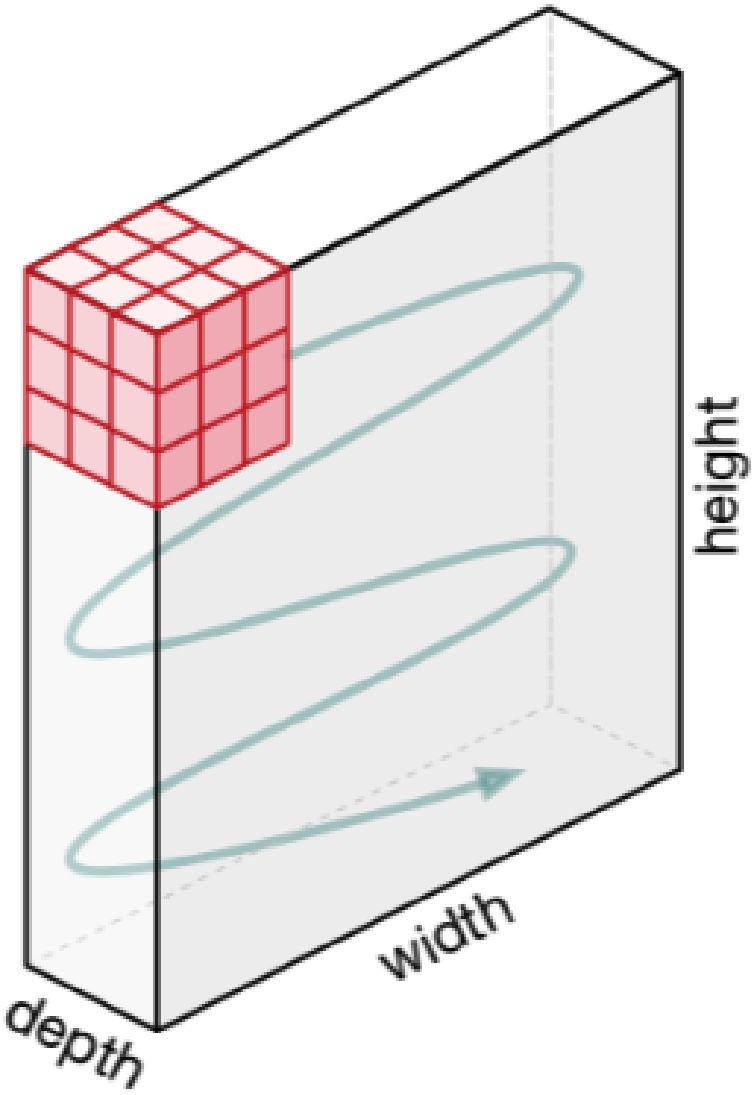
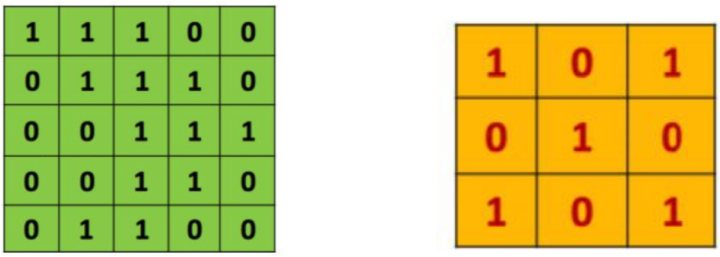


Fig. 2.3. Movement of a Kernel.



(a) Lattice of Size 5x5. (b) Lattice of Size 3x3.

Fig. 2.4. 3x3 Kernel Input given as an Image.

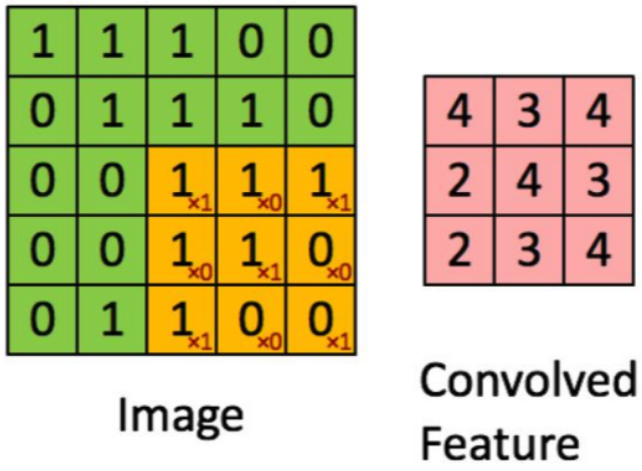


Fig. 2.5. Feature which is Convolved.

2.4 Activation Layer

Activation layer also known as non linearity is a key component of CNN. Frequently used non linearity is ReLU whereas in this study we used ReLU in-place. As the maximum given input data is non-linear, to include a non-linearity inside a convolutional neural network this component is used. The operation of this function is shown in Fig 2.7. The output feature map is shown in Fig 2.7.

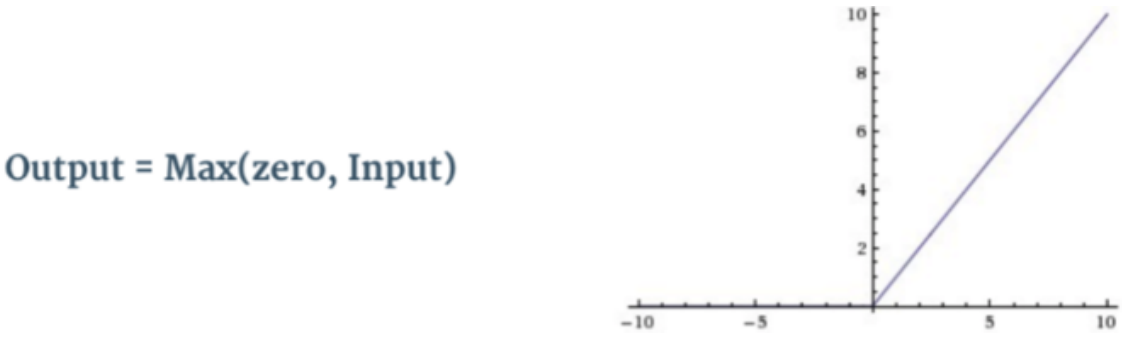


Fig. 2.6. Mathematical and Graphical Representation of ReLU.

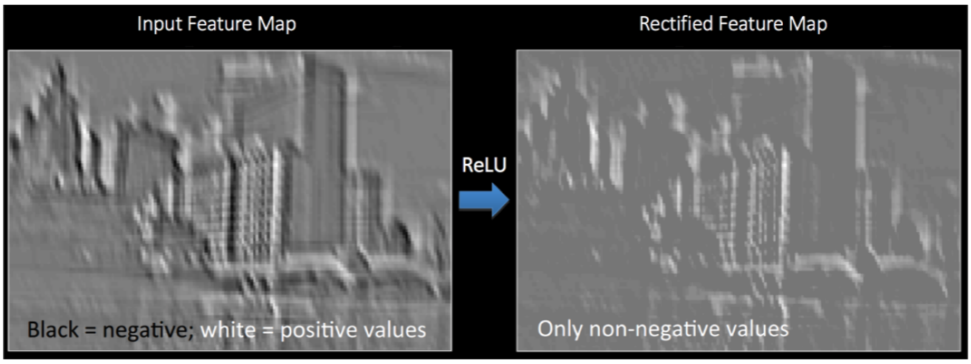


Fig. 2.7. Operation of ReLU Function.

2.5 Baseline Architectures

2.5.1 Baseline Architecture of SqueezeNet

The section below surveys the design of SqueezeNet [3] that contains firemodules, Relu initiation, max pooling and normal pooling layers, activation layer (softmax, and kaiming uniform instatement. Foundation of this engineering is fire module, involving a press layer, 1x1 layer which is shown as s2 and 2 grow layers, extraction 1 and extraction 3 layers.

- Procedure 1: 1x1 filter are replaced by 3x3 filters.
- Procedure 2: The input channel numbers are reduced to filters of size 3x3.
- Procedure 3: The network is down sampled lately.

The above three procedures are utilized to build the benchmark design. The parameter number is highly decreased by using fire module when realted to architecture of VGG. Without loss of accuracy, squeezenet model size is decreased to 2.5 megabytes from 385 megabytes of VGG architecture. By using the procedures like skip connections, element-wise addition, batch normalization[12] and different optimizers, there is an extent of increasing the betterment of squeezenet architecture. Fig. 2.12 illustrates the SqueezeNet standard design alongside the portrayal of fire module.

At that point, SqueezeNet v1.1 appeared in which the quantity of channels just as the channel sizes are additionally diminished. Presently, it reduced the computation

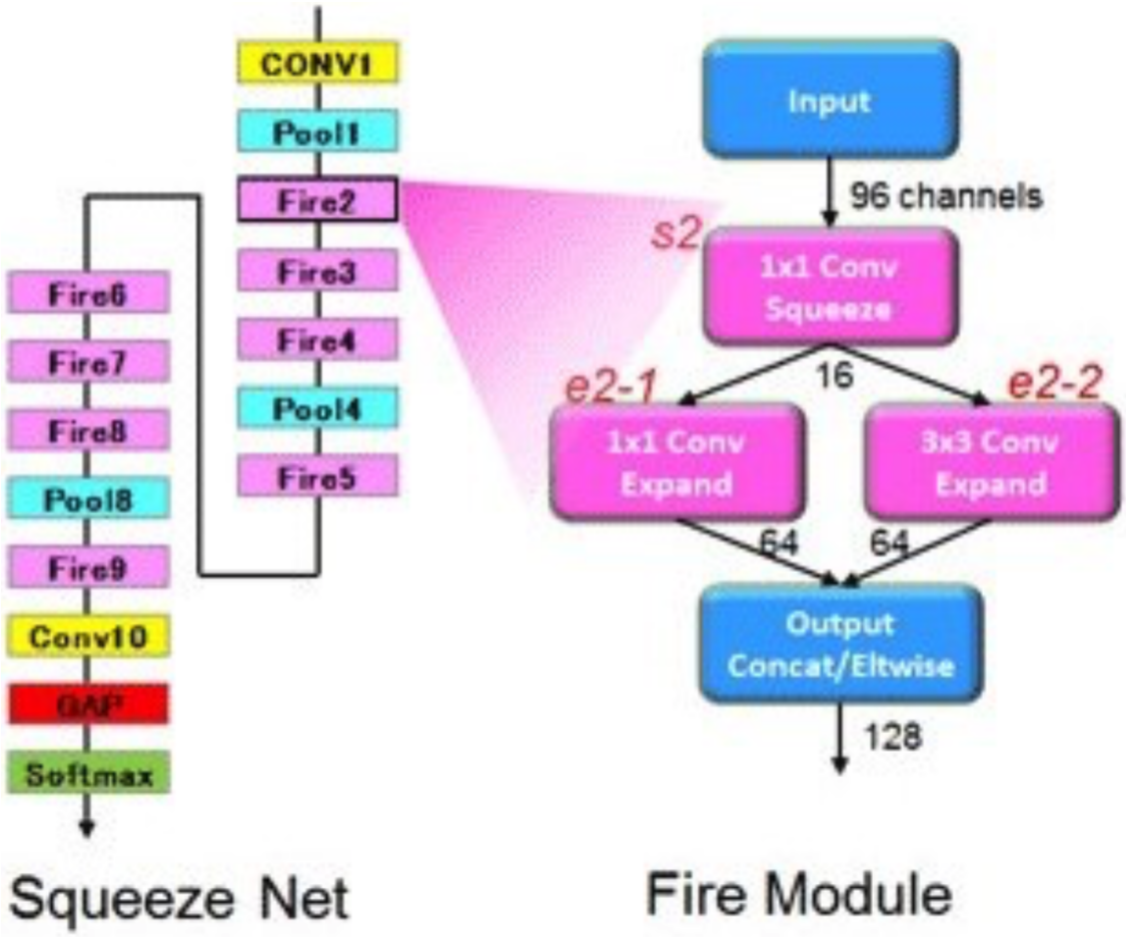


Fig. 2.8. Baseline Architecture and Fire Module of SqueezeNet.

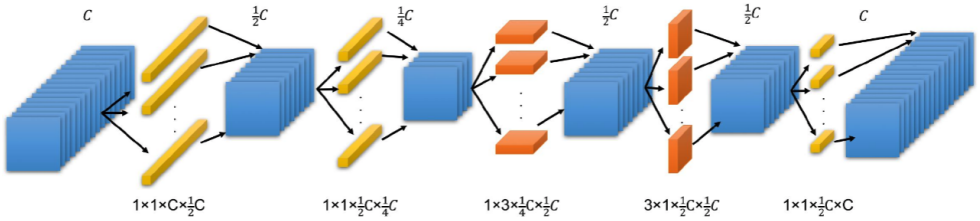


Fig. 2.9. SqueezeNext Baseline Architecture Bottleneck Module.

by 2.4 times over the initial version of Squeezenet with no trade off between accuracy and speed of the model. Propelled by these unbelievably little large scale design of SqueezeNet v1.0 and v1.1, CNN writing survey bits of knowledge, and new procedures, at last, adds to engineering alterations made inside the proposed structures.

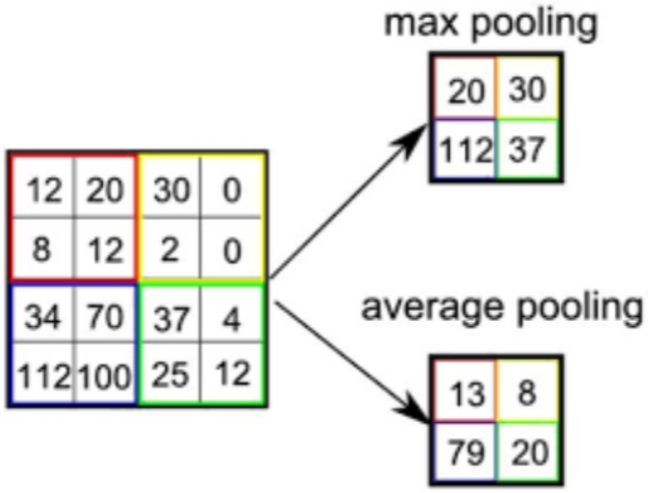


Fig. 2.10. Operation of Pooling.

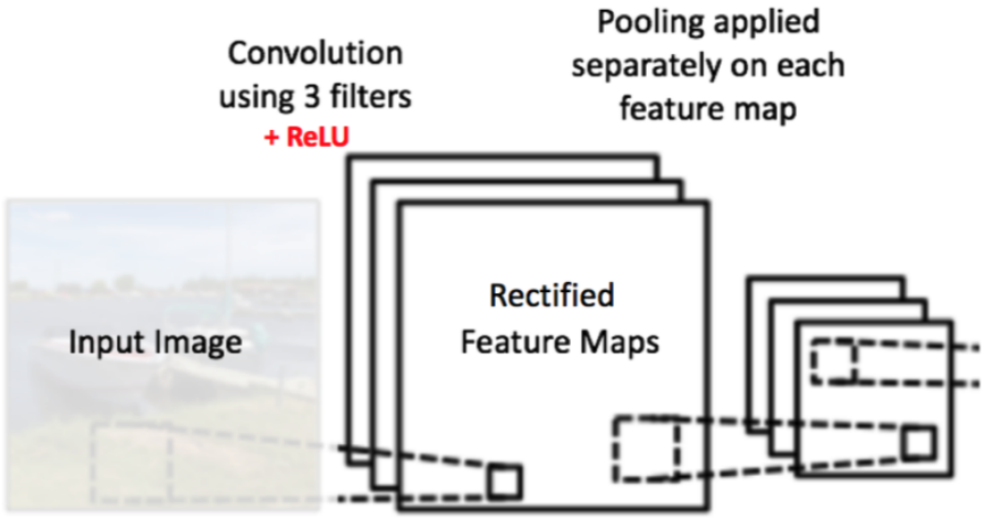


Fig. 2.11. Rectified Feature Maps after Applying Pooling.

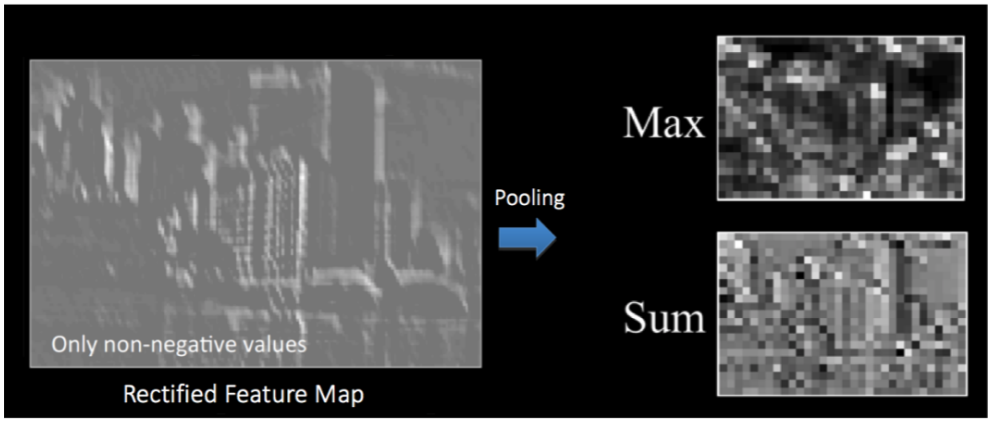


Fig. 2.12. Illustration of Pooling Operation.

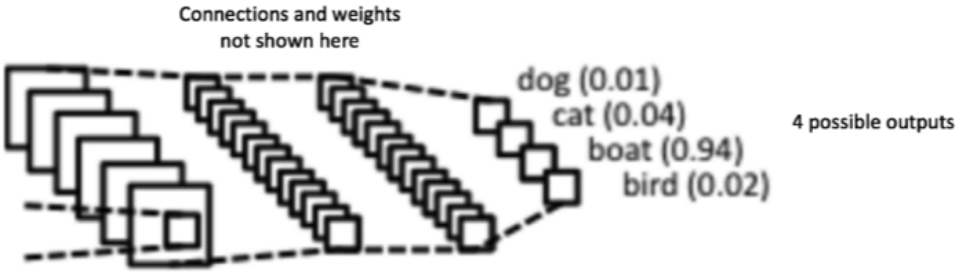


Fig. 2.13. Fully Connected Layer.

2.5.2 Baseline Architecture of SqueezeNext

Benchmark design of SqueezeNext [4] rose following the approach of the benchmark design of SqueezeNet. The basis for SqueezeNext is obtained from SqueezeNet which contains the below:

Truth be told, a superior model exactness and size is achieved in contrast with squeezenet gauge design. The squeezenext pattern (6,6,8,1) engineering design appeared in Fig. 2.14 delineates the squeezenext pattern engineering executed with 3 information channels and 32x32 input size on the dataset named CIFAR-10.

At that point, following the primary convolution there is a max pooling layer which is the contribution by the yield of convolution which is main, this is absent in Fig. 2.14, however appeared in Fig. 2.17.

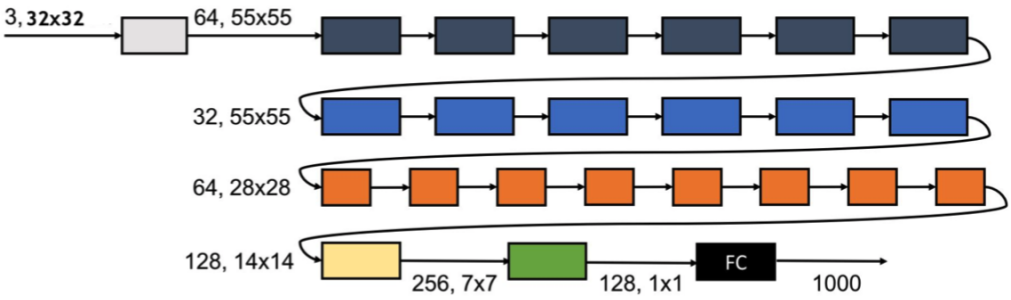


Fig. 2.14. Baseline Architecture of SqueezeNext.

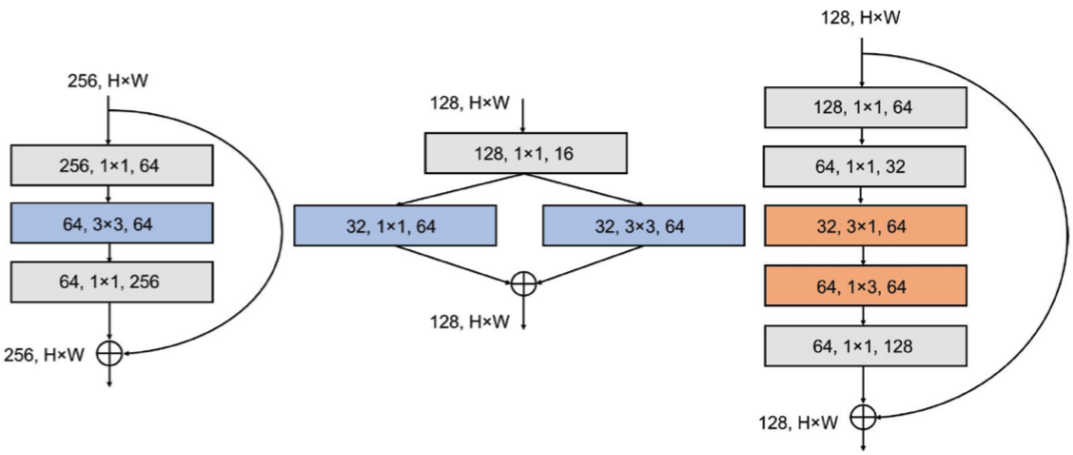


Fig. 2.15. Baseline Architecture Modules of ResNet, SqueezeNet and SqueezeNext.

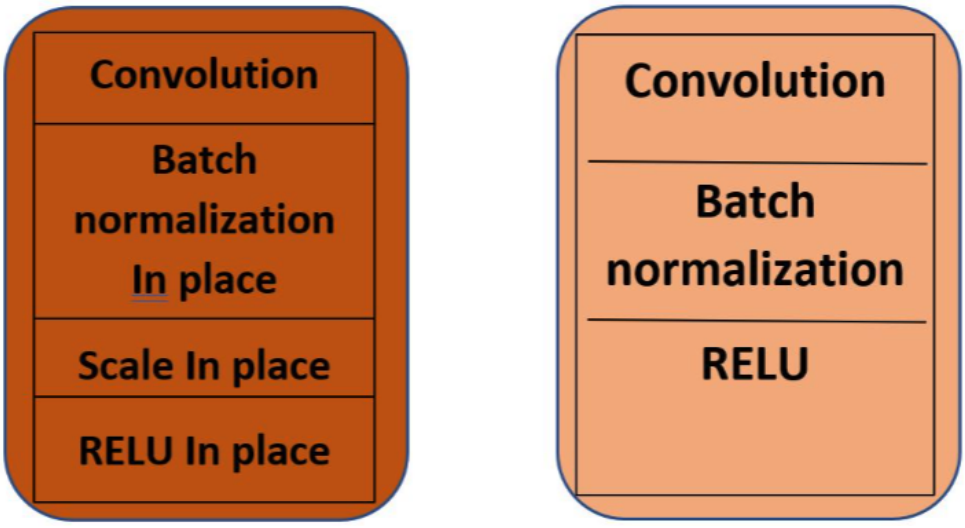


Fig. 2.16. Basic Block of SqueezeNext and Iuuuyis Version of Squeezenext.

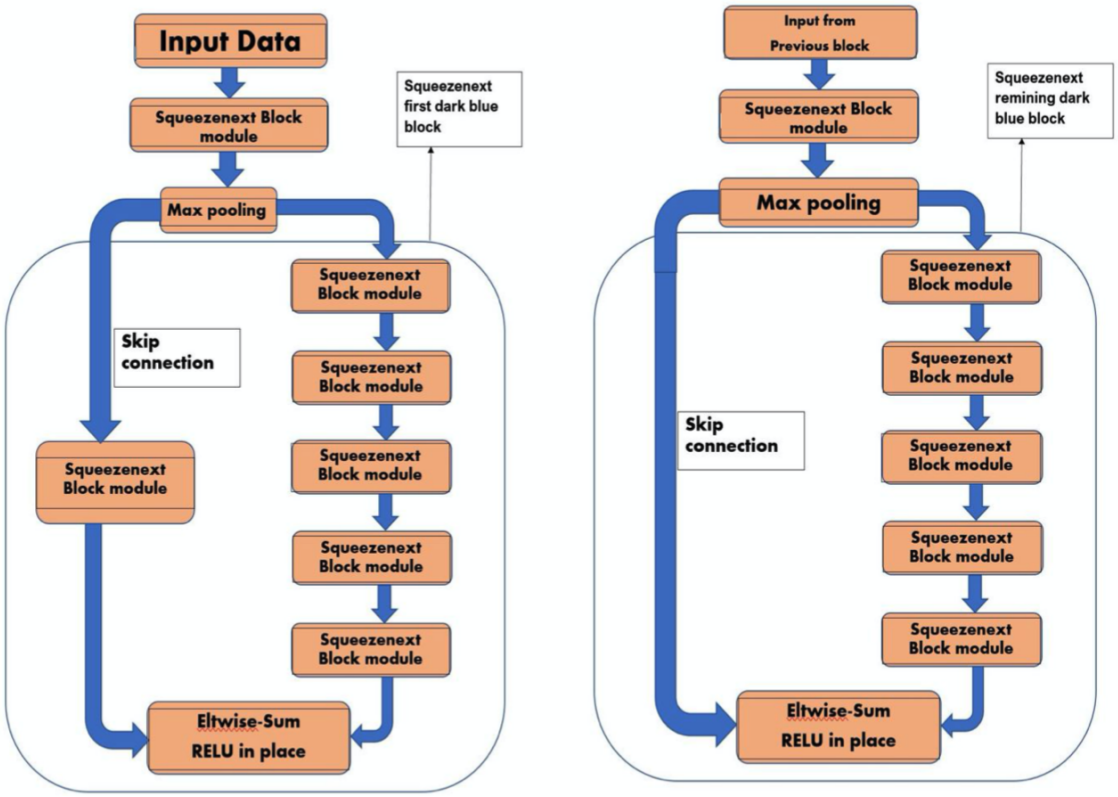


Fig. 2.17. First Block and Second Block Structures of SqueezeNext.

3. HARDWARE AND SOFTWARE USED

3.1 Hardware Used

- Aorus Geforce RTX 2080Ti GPU.
- Nvidia Geforce GTX 1080Ti GPU.
- NXP BlueBox 2.0.
- NXP i.MX RT1060 MCU.

3.1.1 NXP BlueBox 2.0

The NXP BlueBox is an improvement stage arrangement that gives the necessary execution, practical security and car unwavering quality for specialists to create self-driving autos. The most recent expansion to the arrangement, the BlueBox 2.0 family, fuses the following:

- S32V234: Processing unit which combines sensor and vision of a car.
- LS2084A: installed PC processing unit.
- S32R27: microcontroller used for radar.

Fig. 3.1. shows the genuine BLBX2 constant installed framework stage. The toolbox right now focal preparing motor joined with a lattice of sensors and different segments or sensors, making a start to finish arrangement. One among the many critical prerequisites of the present proposal is to dissect the ability of BlueBox 2.0 by NXP as a self-sufficient installed stage framework for constant self-sufficient applications.

Level 3 self-governing applications is the place the driver can totally hand over security basic capacities in specific circumstances. Level 5 independent vehicles are

going to require a lot of memory and calculation assets. The test here is enabling self-ruling autos to assume control over all the more driving capacities, more calculation and memory assets with a bomb evidence framework with a total well being, to be secure and make it more reliable. Only task confronting self-sufficient driving engineers is demonstrating the well being of the self-governing frameworks. That is the reason, we despite everything need new hardly any more years to think of safe bomb evidence and further, further developed and front line very good quality constant independent frameworks.

Fig. 3.1. NXP BlueBox 2.0.

BlueBox works as the focal processing unit of the framework in this way, giving the ADAS framework to be equipped for sending productive and better CNN/DNN models. The architectures of S32V and LS2 are shown in Fig. 3.4 and Fig. 3.5.

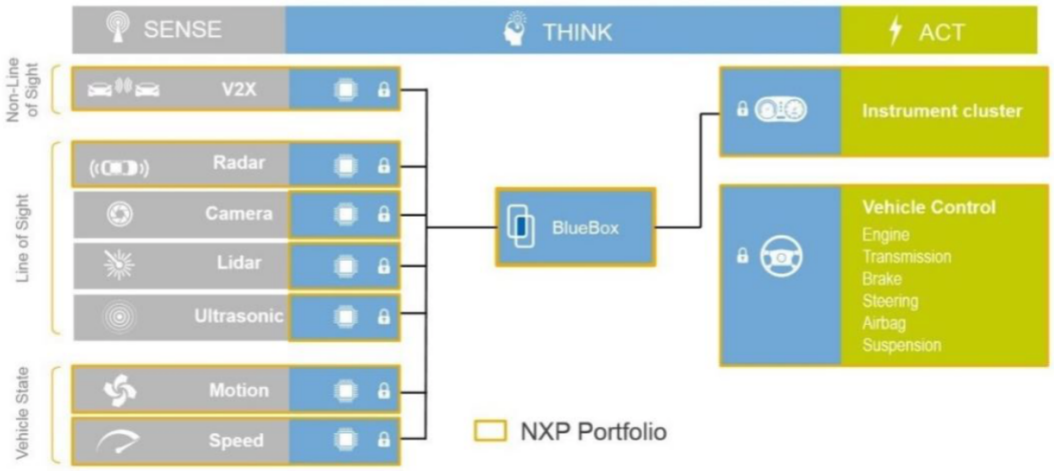


Fig. 3.2. ADAS Systems Development Platform: BlueBox 2.0.

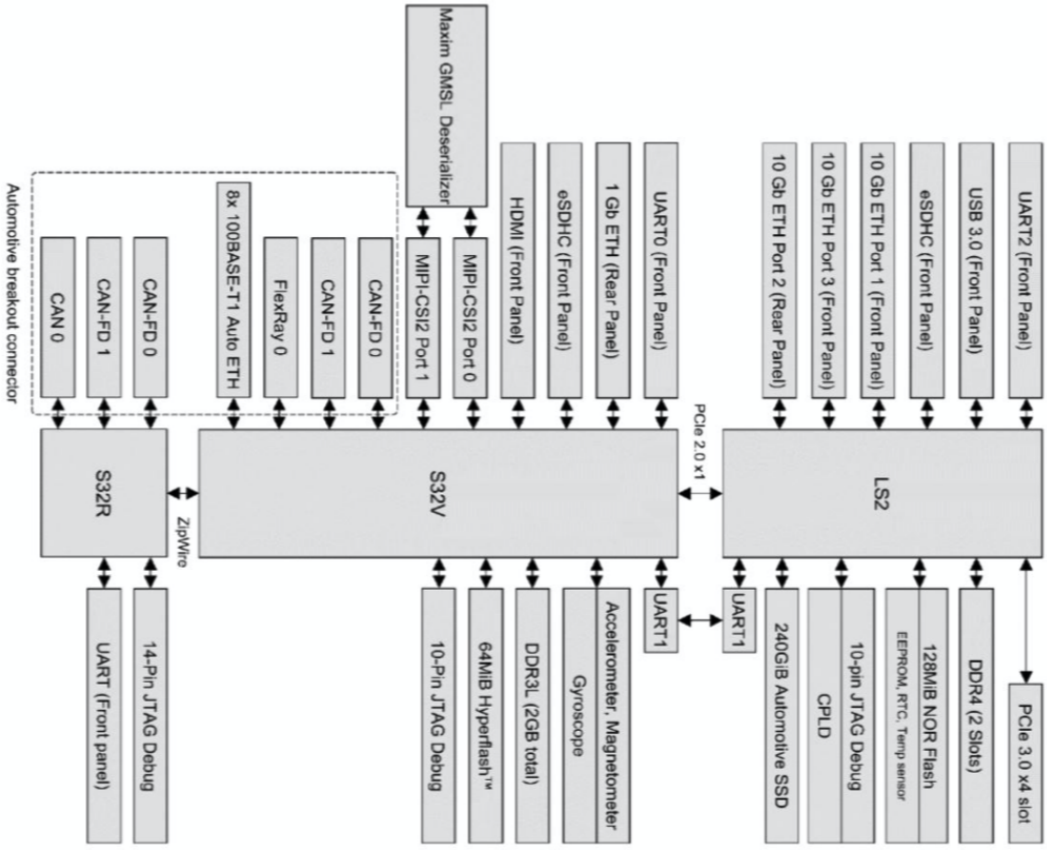


Fig. 3.3. NXP Bluebox 2.0 Hardware Arrchitecture.

3.1.2 NXP i.MX RT1060 MCU

The i.MX RT1060 is the most recent expansion to the industry's first hybrid MCU arrangement and grows the i.MX RT arrangement to three versatile families. The i.MX RT1060 copies the On-Chip SRAM to 1MB while keeping pin-to-stick similarity with i.MX RT1050. This new arrangement presents extra highlights perfect for constant applications, for example, High-Speed GPIO, CAN-FD, and synchronous equal NAND/NOR/PSRAM controller. The i.MX RT1060 runs on the Arm Cortex-M7 center at 600 MHz. The block diagram of i.MX RT1060 can be observed from Fig. 3.6.

Features of i.MX RT1060

- Highest performing Arm Cortex-M7.
- 3020 CoreMark/1284 DMIPS @ 600 MHz.
- 1MB On-Chip SRAM - up to 512KB configurable as Tightly Coupled Memory (TCM).
- Real-time, low-latency response as low as 20 ns.
- Industry's lowest dynamic power with an integrated DC-DC converter.
- Low-power run modes at 24MHz.
- Advanced multimedia for GUI and enhanced HMI.
- Extensive external memory interface options: NAND, eMMC, QuadSPI NOR Flash, and Parallel NOR Flash.
- Wireless connectivity interface for Wi-Fi®, Bluetooth®, Bluetooth Low Energy, ZigBee® and Thread™.
- Supported by MCUXpresso SDK, IDE and Config Tools.

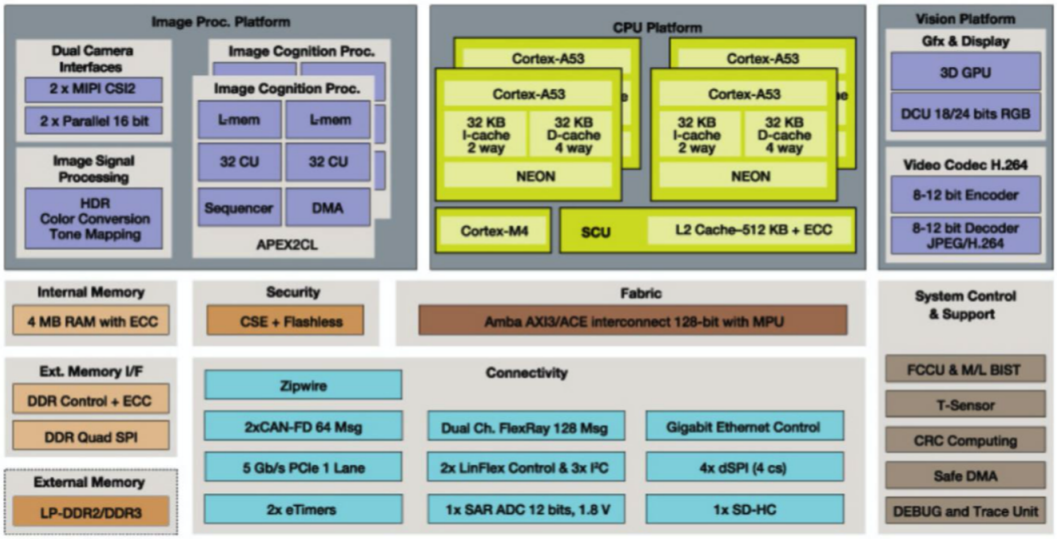


Fig. 3.4. S32V234 Hardware Architecture.

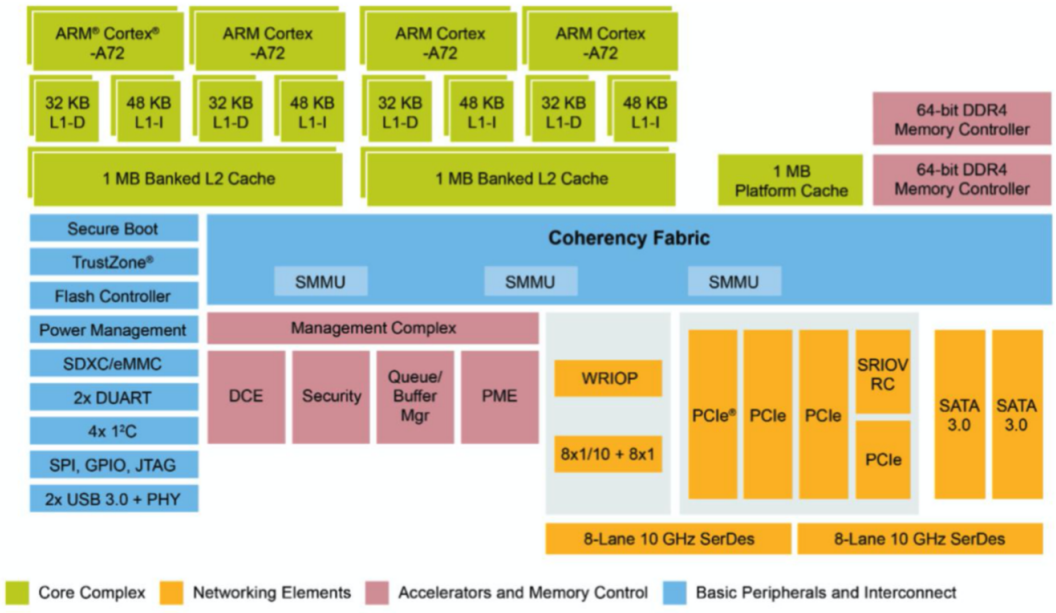


Fig. 3.5. LS2084A Hardware Architecture.

3.2 Software Used

- Python version 3.6.7.
- Spyder version 3.6.
- Pytorch version 1.0.
- Netscope (SE-SqueezeNext visualization).
- RTMaps by Intempora.
- MCU Xpresso SDK.
- Teraterm.

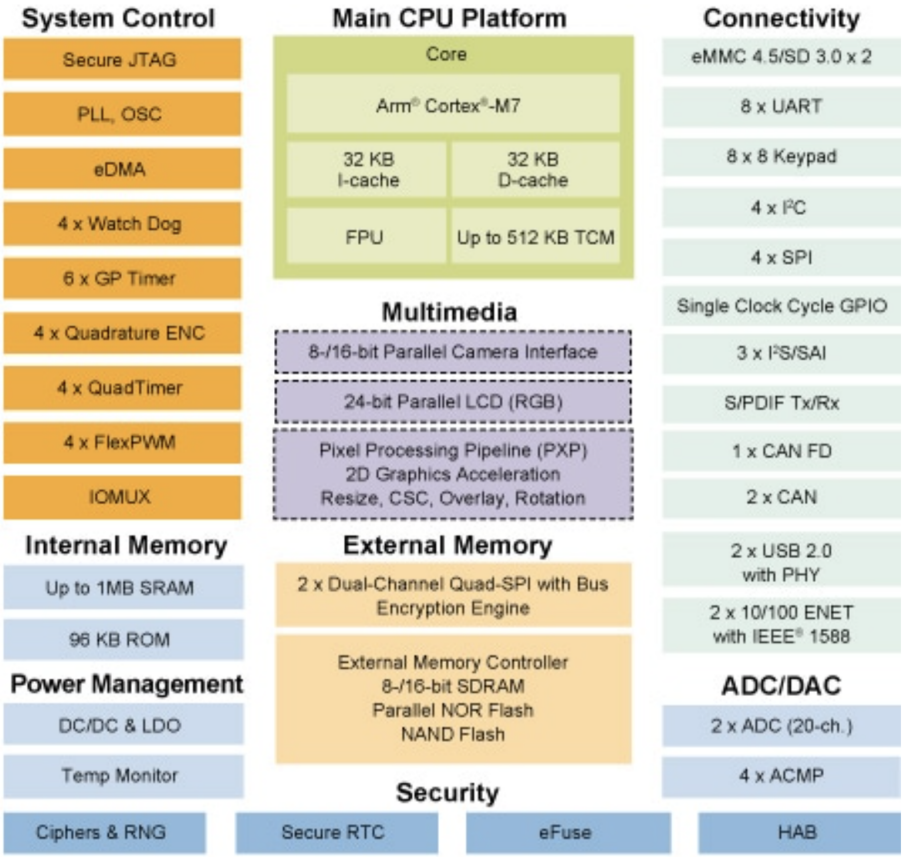


Fig. 3.6. Block Diagram of i.MX RT1060 MCU.

3.2.1 RTMaps

RTMaps is a nonconcurrent elite stage intended to compete and succeed multiple sensor challenges and to permit specialists and scientists to exploit an effective and simple to-utilize structure for quick and vigorous improvements. This is a secluded toolbox for applications which are multimodal. Advanced Driver Assistance Systems, independent vehicles, mechanical technology, Unmanned Ground Vehicles, Unmanned Aerial Vehicles, HMI, information logging. It tried for handling & intertwining the information flow in the continuous or even in the post-preparing situations [28]. The programming engineering comprises of a few autonomous modules that can be utilized for various circumstance and condition.

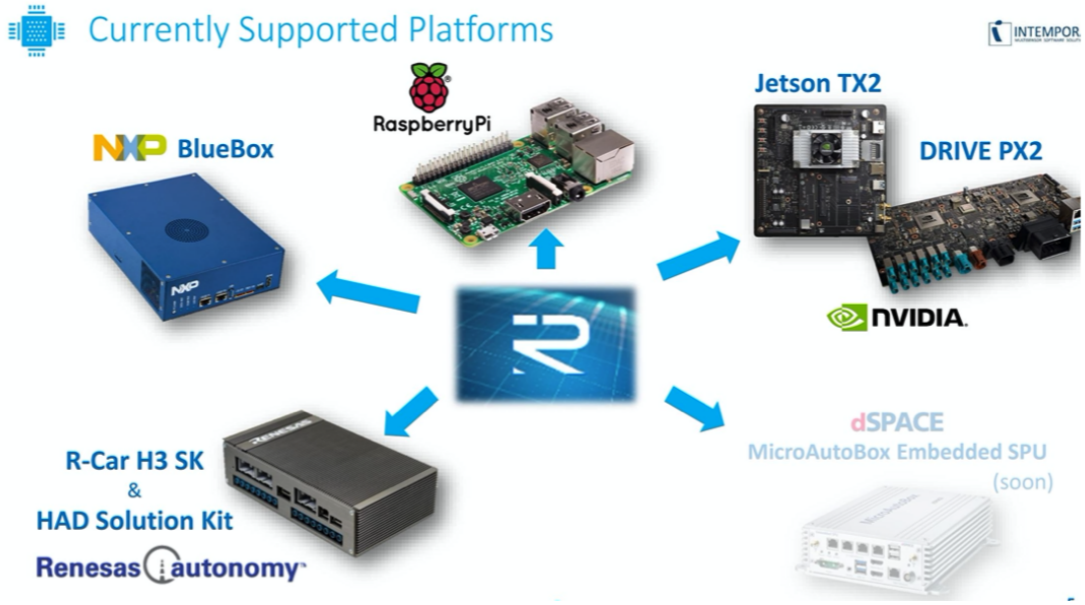


Fig. 3.7. Supported Platforms for RTMaps.

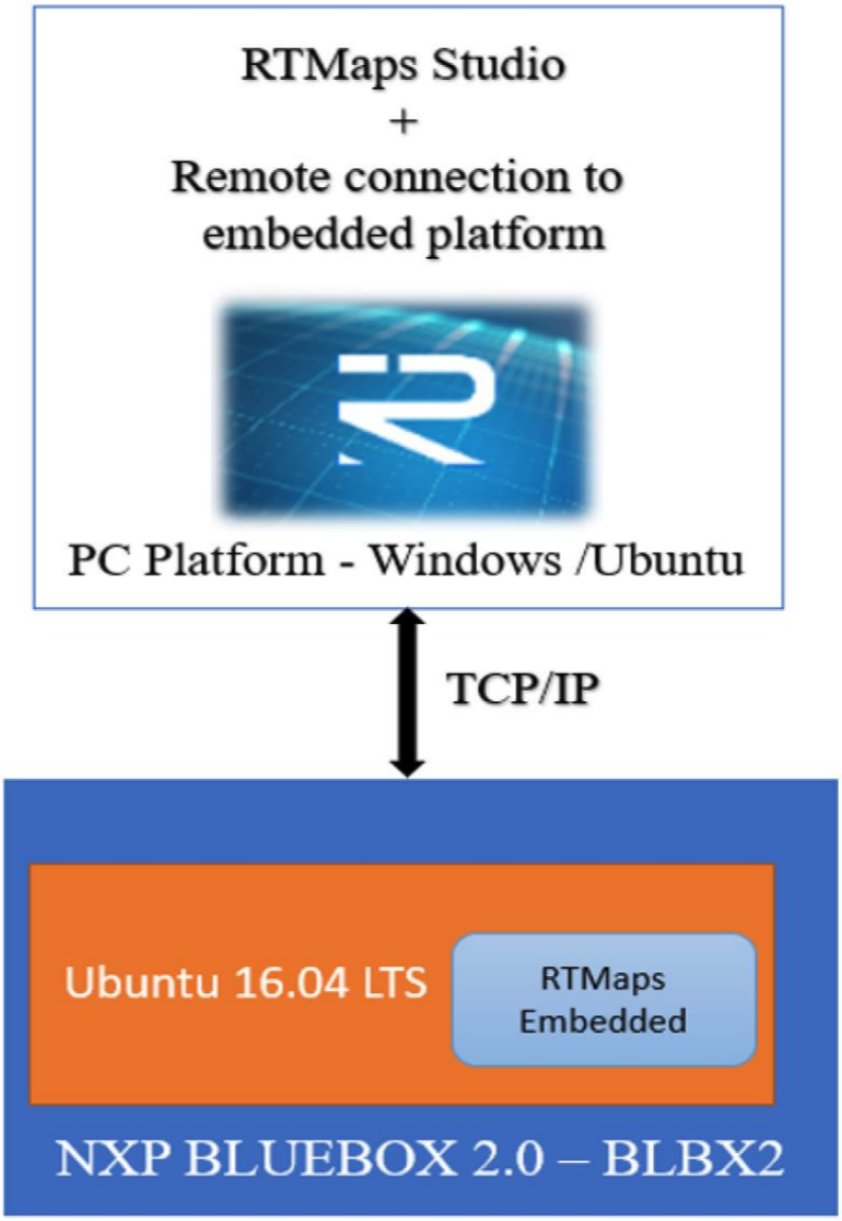


Fig. 3.8. BlueBox 2.0 Setup with RTMaps.

4. TECHNIQUES TO ENHANCE DNN PERFORMANCE

This chapter describes the different techniques which led to the development of this study. The following methods shows the enhancement of performance:

1. The information can be used to enhance the performance. This alludes to gather and additionally develop more information, improve the nature of the information, information expansion and highlight choice systems.

2. Enhance the performance with the design change. Right now, design can be motivated by the writing survey, advantages of the current structures and re-testing methods.

3. Enhance the performance with groups which incorporate the accompanying potential ways that are to consolidate models, join sees, and stacking.

4.1 Architecture Tuning

The below mentioned techniques are utilized in this study and these are the purpose for tuning the proposed architecture:

- Various Learning Rate Schedule.
- Use of various optimizers.
- Save and Load Checkpoint.
- Use of various activation functions.

4.2 Various Learning Rate Schedule

LR schedules try to alter the learning rate(LR) during preparing by lessening the LR as indicated by a pre-characterized plan. Basic LR schedules incorporate time sensitive rot, step rot, exponential rot, and cosine toughening.

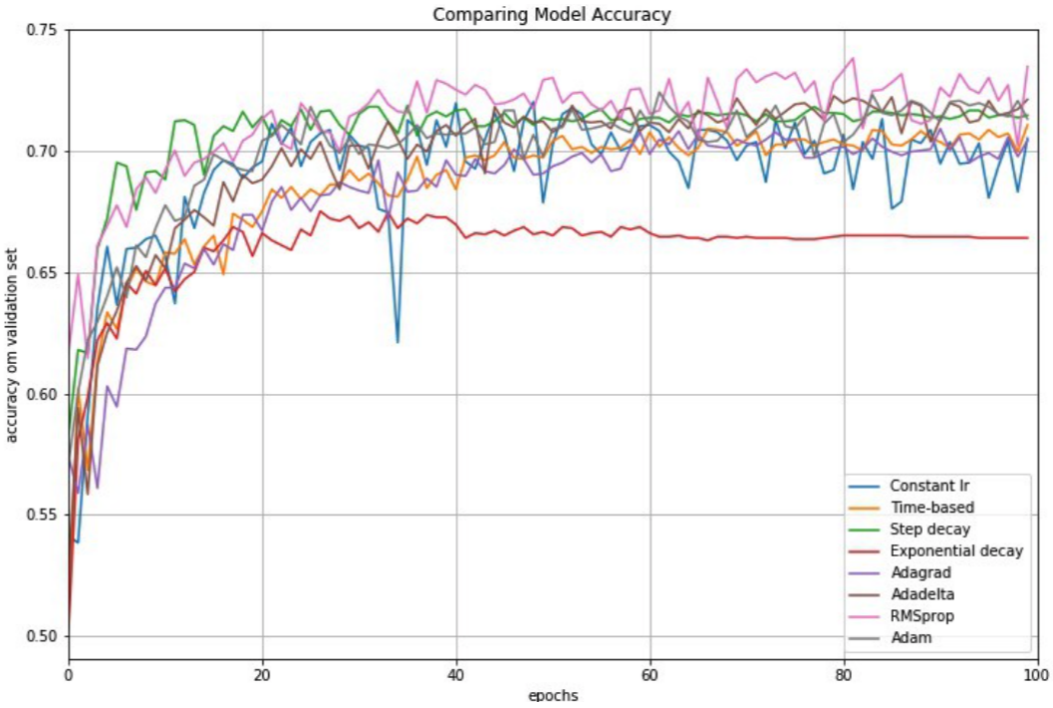


Fig. 4.1. Different Learning Rate Schedules.

4.3 Use of Various Optimizers

Right now, optimizers [21] were executed on proposed structures dependent on the bits of knowledge. Allude for the numerical structure or conditions of the optimizers.

4.3.1 ADAGRAD

Adagrad [9] is a calculation for inclination based advancement that does only this: It adjusts the learning rate to the parameters, performing littler updates(for example low learning rates) for parameters related with every now and again happening highlights, and bigger updates (for example high learning rates) for parameters related with inconsistent highlights. Consequently, it is appropriate for managing meager information. Adagrad significantly improved the power of SGD and utilized it for preparing enormous scope neural nets at Google, which - in addition to other things - figured out how to perceive felines in Youtube recordings.

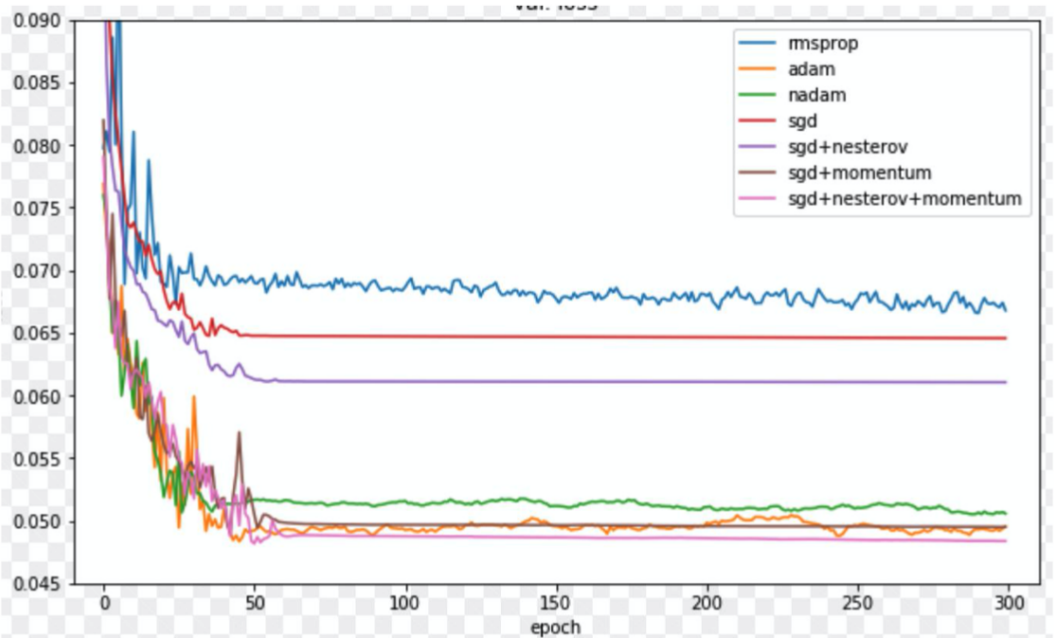


Fig. 4.2. Different Optimizers Used.

4.3.2 SGD

SGD plays out a parameter update that implies there is each update in turn, for every one of the preparation models. This issue is generally alluded to as exploring through gorges. We acquaint a term called energy with take care of the gorges issue.

4.4 Use of Various Activation Functions

4.4.1 Sigmoid

Sigmoid capacity extend resembles a 'S' formed bend initiation work which lies between a scope of 0 and 1. It is anything but difficult to actualize yet it has serious issues, for example, evaporating inclination issue and, not zero focused yield which made this actuation misfortune its place in the DNN people group. It makes too visit and enormous slope refreshes in various ways, bringing about harder advancement of DNN. Further it will soak, slaughter inclinations and will have moderate intermingling issues.

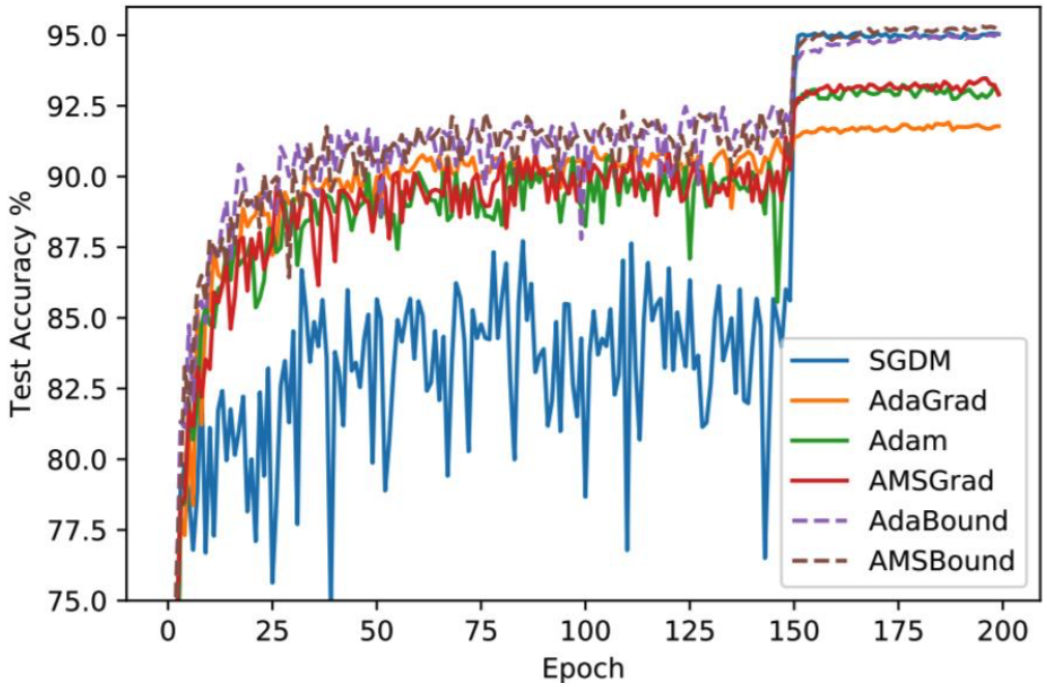


Fig. 4.3. Comparison of ADABOUND with Other Optimizers.

4.4.2 ReLU

The ReLU is the most utilized actuation work on the planet right now. Since, it is utilized in practically all the convolutional neural systems or profound learning. The capacity and its subsidiary both are monotonic. Be that as it may, the issue is that all the negative qualities become zero promptly which diminishes the capacity of the model to fit or train from the information appropriately. That implies any negative info given to the ReLU actuation work transforms the incentive into zero quickly in the chart, which in turns influences the subsequent diagram by not mapping the negative qualities properly. However, it despite everything has issue that it can explode the initiation work because of its wide range [0, inf).

4.4.3 ELU(Exponential Linear Unit)

ELU [23,24] is an initiation work that will in general combine to zero quicker and produce increasingly exact outcomes. Exponential Linear Unit or its broadly known name ELU is a capacity that will in general meet expense to zero quicker and produce increasingly exact outcomes. Diverse to other actuation capacities, ELU has an additional alpha consistent which should be sure number. ELU is fundamentally the same as RELU with the exception of negative data sources. They are both in character work structure for non-negative data sources. Then again, ELU becomes smooth gradually until its yield equivalent to $-\alpha$ while RELU forcefully smoothes. ELU is a solid option to ReLU. Not at all like to ReLU, ELU can deliver negative yields.

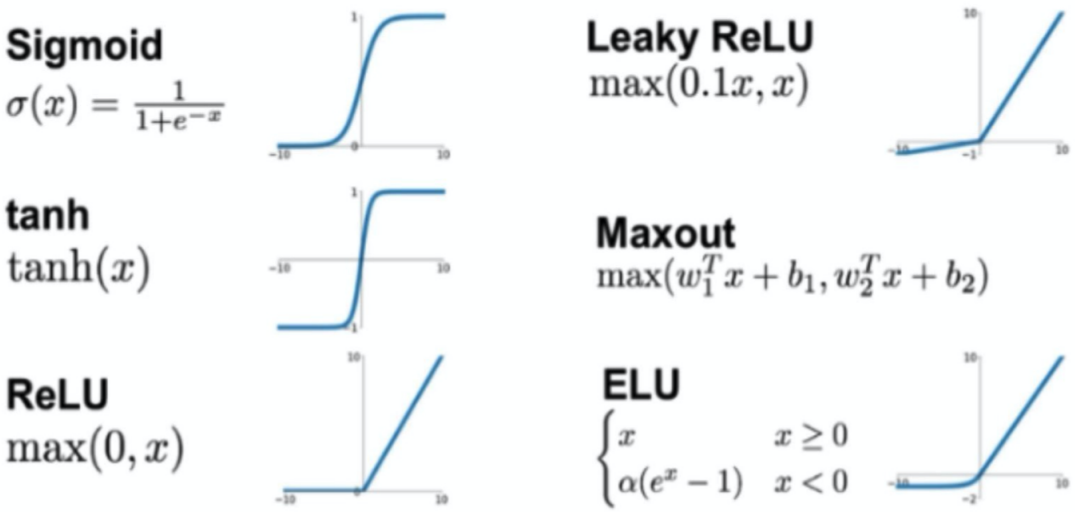


Fig. 4.4. Various Activation Functions.

5. SQUEEZE-AND-EXCITATION:SQUEEZENEXT

The proposed Squeeze-and-Excitation SqueezeNext is a CNN model. Inspiration for this engineering is through SqueezeNet, Mobilenet and SqueezeNext designs. This contains essential squares sorted out in 4-phase setup called bottleneck modules, a press and-excitation (SE) square , normal pooling layer, completely associated layer and a spatial goals layer. Nestrov , rot and force are executed with SGD streamlining agent. We executed learning rate plan which is exponentially rotting by refreshing learning rate in four phases: first after 60 ages, second after 120 ages, third after 150 ages and last after 180 ages.

As appeared in Fig. 5.1, the bottleneck module contains a fundamental square with 1x1 convolution, another essential square with 1x1 convolution, fundamental square with 3x1 convolution, essential square with 1x3 convolution, last fundamental square with 1x1 convolution lastly a se square. From Fig. 5.2, essential square contains convolution layer followed by BN layer & ReLu set up. Here essential squares line up convolutions encased by bottleneck modules which are gathered and sorted out in the 4-phase execution setup alongside a spatial layer, dropout layer , se square, normal pool layer and a completely associated layer are appeared in Fig. 5.3. To diminish the parameter check, spatial layer can be wiped out in the little measured models of the proposed engineering.

The portrayals of two parameters which helped in contracting the model are given as tracks with the crush and-excitation (SE) square. The [1,2,4,1] 4-phase arrangement of the SE-SqueezeNext engineering is outlined in Fig. 5.4. The Squeeze-and-Excitation SqueezeNext bottleneck module containing essential squares with SE squares included are delineated in Fig. 5.4. Table I presents the SE-SqueezeNext table with [1,2,4,1] 4-phase design, spatial goals, dropout layer, SE square and FC convolution.

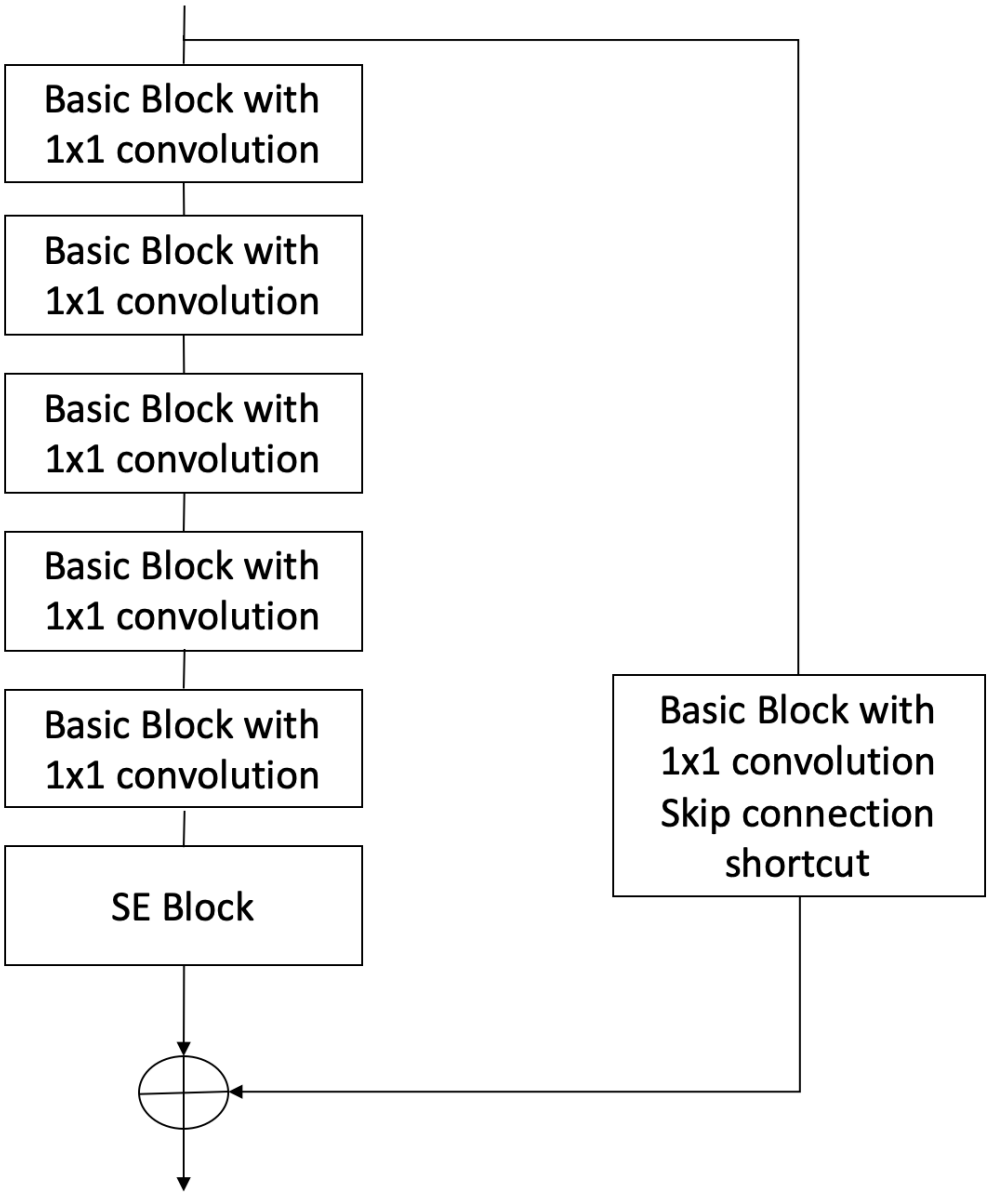


Fig. 5.1. Illustration of Squeeze-and-Excitation SqueezeNext's Bottleneck Module.

5.1 Squeeze-and-Excitation Block

The structure of the SE square is delineated in Fig. 5.5. For some random change $\mathbf{F}_{tr}$ mapping the info $\mathbf{X}$ to the element maps $\mathbf{U}$ where $\mathbf{U} \in \Re^{H \times W \times C}$, for example a convolution, we can build a relating SE square to perform include recalibration. A press activity is done through which the highlights U are first passed, at that point by

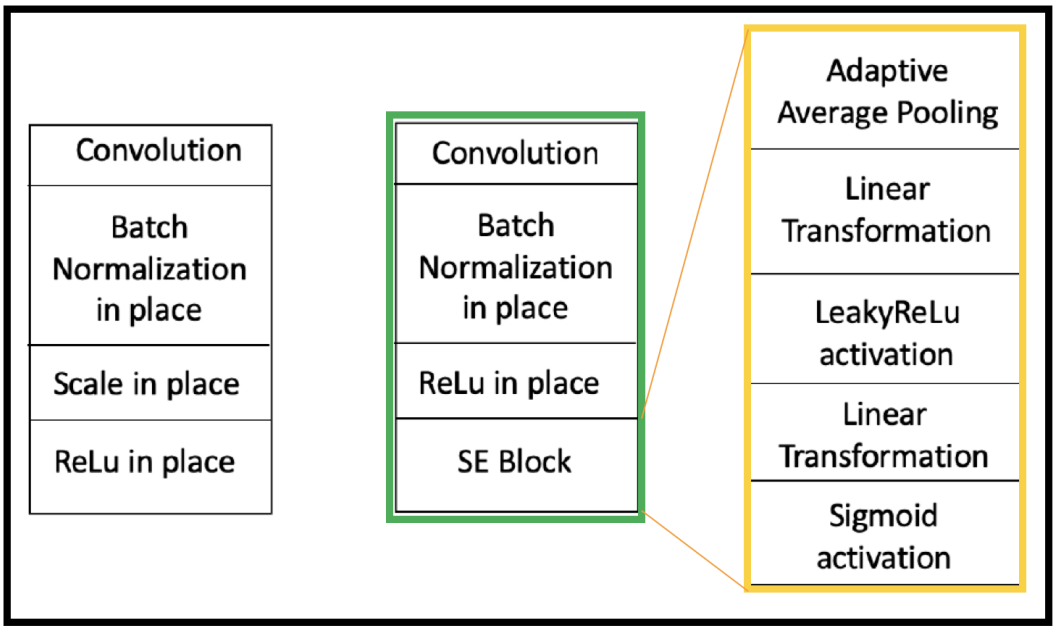


Fig. 5.2. Basic Block of SqueezeNext Baseline, Basic Block of SE-SqueezeNext and SE Block.

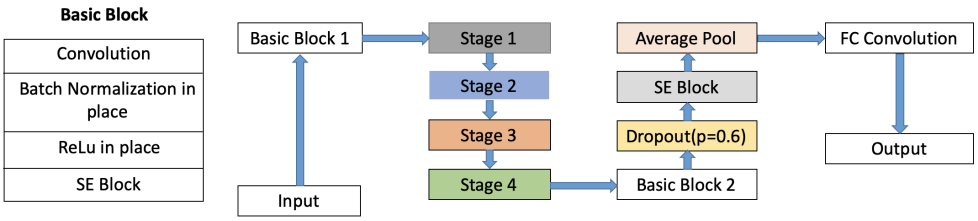


Fig. 5.3. Illustration of Basic Block (left) and Squeeze-and-Excite SqueezeNext Architecture.

totaling highlight maps over their spatial measurements H×W a channel descriptor is created. This descriptor essentially creates an installing of the worldwide dispersion of channel-wise element reactions which permits every one of its layers to utilize the data from the worldwide open field of the system. After the total is played out, a straightforward self-gating instrument where an assortment of per-channel regulation loads are created by accepting the inserting as an information. This is the excitation activity.

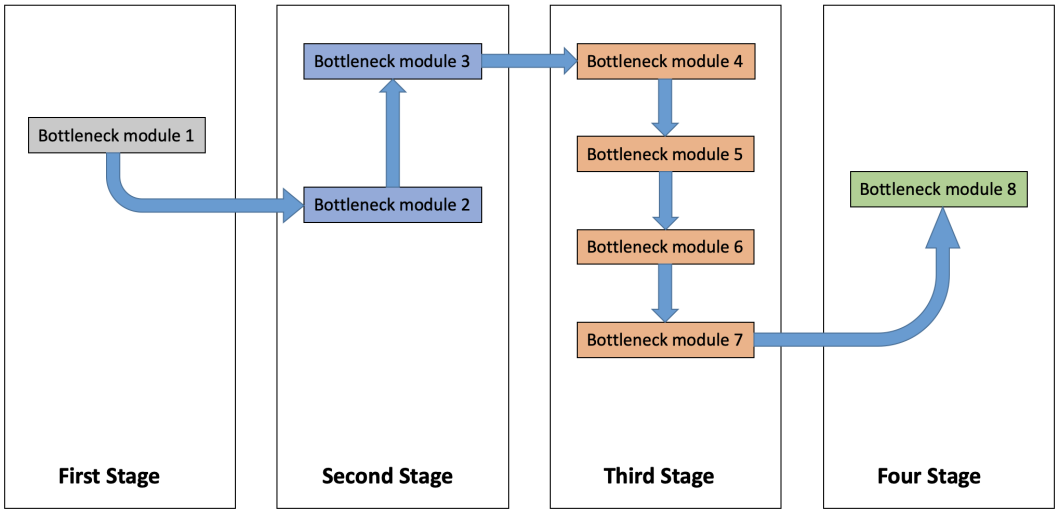


Fig. 5.4. 4-stage [1,2,4,1] Configuration of the Squeeze-and-Excitation SqueezeNext Architecture.

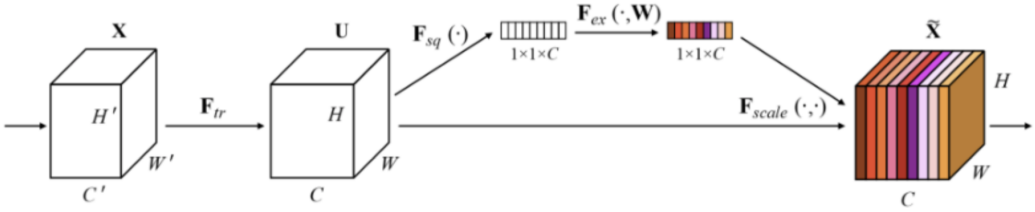


Fig. 5.5. Structure of SE Block.

5.2 Dropout Layer

This layer is utilized to ad lib the overfitting issue existing in CNNs or neural systems. This is a regularization strategy for estimate and dropping some arbitrary load from enormous arrangement of loads in a CNN or neural system. A DNN or a profound CNN prepared on a little dataset (absence of information) can bring about overfitting issue which brings about terrible showing and increment speculation issue

or blunders because of the issue of over fitting. This is a straightforward way to deal with decrease overfitting in a CNN and improve the presentation of DNN/CNN.

5.3 Resolution Multiplier

To decrease the expense of calculation of a neural system, this multiplier is utilized. Goals multiplier eventually decreases the inside portrayal of each layer and is applied to include picture. For the most part, estimation of the info goals is set totally. Diminished DNN models/structures are produced if the worth is under 1. The expense of calculation is diminished by the square of this parameter. In this paper, we utilized the estimations of 6, 7, 8, 10, 11, 12, 14, 16, 21 & 23.

5.4 Width Multiplier

To structure littler and less computationally costly models, width multiplier is utilized. At each layer, this additionally helps for making an unvarying meager CNN. Default or general qualities for width multiplier are 0.25, 0.5, 0.75 and 1. It is the key component for diminishing the expense of calculation and parameter check. Scientifically, it is quadratically lessens it by multiple times the intensity of width multiplier.

6. RESULTS

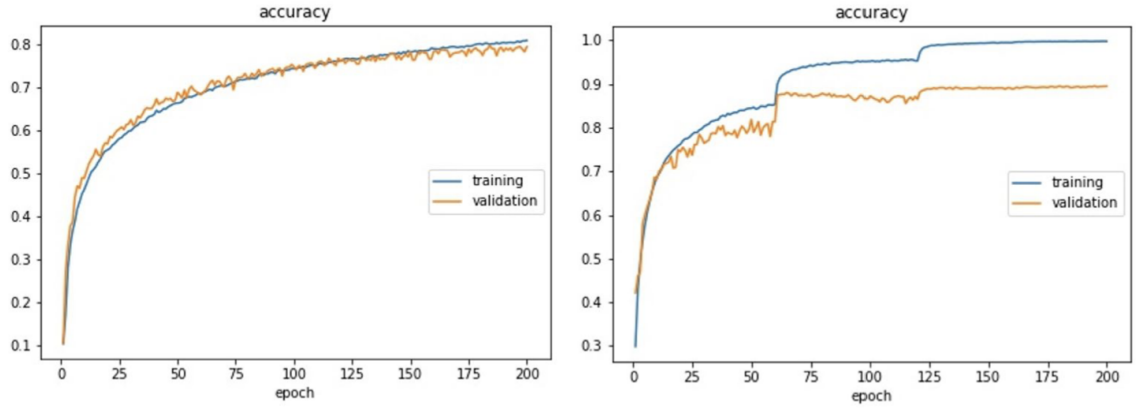
Proficient CNN designs are created because of the alterations in the proposed engineering, which is unmistakable as SE-SqueezeNext having a model sizes extending from 0.595MB to 6.59MB as appeared in Table III. When contrasted with pattern Squeezenext's model size i.e., 9.531MB, the proposed SE-Squeezenext engineering has a decreased model size of 0.595MB. Barely any central point prompting this decrease of model size are different goals and width multipliers. From the perceptions, dropout layer execution is better than BN layer. The itemized depiction of crush and excitation administrators in SE square are as per the following:

Squeeze Operator:

The criticalness of utilizing worldwide normal pooling over worldwide max pooling as our crush administrator is inspected. Normal pooling accomplishes preferable execution over max pooling, despite the fact that both are viable. The premise of crush activity is legitimized by choosing normal pooling. Be that as it may, we note that the exhibition of SE squares is genuinely powerful to the decision of explicit total administrator.

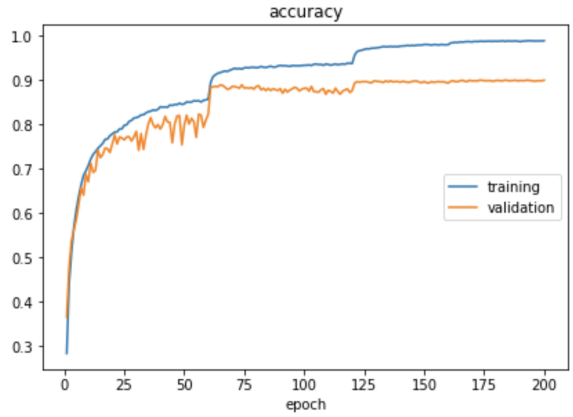
Excitation Operator:

The alternative for non-linearity in the excitation component is evaluated here. Two further choices: LeakyReLU and Tanh are thought of, and try different things with supplanting the sigmoid with these alterantive non-linearities. By exchanging the sigmoid with Tanh marginally declines the presentation, where LeakyRelu is unequivocally causes the SE-SqueezeNext to dip under the benchmark of SqueezeNext.



(a) SqueezeNet Accuracy.

(b) SqueezeNext Accuracy.



(c) Squeeze-and-Excitation SqueezeNext accuracy.

Fig. 6.1. Accuracy Plots Comparison with Baseline Architectures.

Helpful changes to create little DNNs and deployable on ongoing installed gadgets are goals and width multipliers. Along these lines, SE-SqueezeNext-10-0.5 is 16X reduced than SqueezeNext-23-1x-v1. This engineering is made increasingly proficient, adaptable and minimal by actualizing set up activities, for example, Relu set up and killing the additional maximum pooling layers utilizing a SE square, width and goals multipliers. Without utilizing the exchange learning technique, each model is approved on CIFAR-10 from the scratch. Arrangement on an ongoing framework having memory imperatives is the significant favorable position of this design. The

precision of this engineering is improved by dropout layer. The configuration for SE-SqueezeNext in all the tables shows SE-SqueezeNext with goals multiplier alongside width multiplier.

From the outcomes introduced in Table 6.1, it is seen that SE-SqueezeNext with dropout layer alongside suitable utilization of multipliers and a huge contrast is made by bottleneck module. Diverse dropout layer probabilities consequences of SE-SqueezeNext are introduced in Table 6.4. Various correctnesses for different models are appeared in Tables 6.2 - 6.4.

Table 6.1.
Results comparison with SqueezeNet & SqueezeNext

Name of Model	**Accuracy%**	**Size of model(MB)**	**Speed of model(seconds)**
SqueezeNet-v1.0	79.59	3.01	4
SqueezeNet-v1.1	77.55	2.96	4
SqueezeNext-23-1x-v1	87.15	2.57	19
SqueezeNext-23-1x-v5	87.95	2.57	19
SqueezeNext-23-2x-v1	90.51	9.53	22
SqueezeNext-23-2x-v5	90.50	9.53	28
SE-SqueezeNext-10-1.0x-v1	90.48	1.81	13
SE-SqueezeNext-10-2.0x-v1	92.60	6.59	21

*All results are 3 average runs with SGD, LR is 0.1

Table 6.2.
Squeeze-and-Excitation SqueezeNext Results with different resolution multipliers

Name of Model	Resolution	Accuracy%	Size of Model(MB)	Speed of Model(seconds)
SE-SqueezeNext-06-1x-v1	1111	87.84	1.22	9
SE-SqueezeNext-10-1x-v1	1241	90.48	1.81	13
SE-SqueezeNext-12-1x-v1	1261	90.50	2.16	15
SE-SqueezeNext-14-1x-v1	1281	89.93	2.52	16
SE-SqueezeNext-22-1x-v1	12161	80.69	3.94	25
SE-SqueezeNext-23-1x-v1	22161	81.41	3.97	26

*Results obtained are 3 average runs with LR, SGD as 0.1

Table 6.3.
Different width multipliers results of Squeeze-and-Excitation SqueezeNext

Name of Model	Width	Accuracy%	Size of Model(MB)	Speed of Model(seconds)
SE-SqueezeNext-10-0.5x-v1	0.5x	86.71	0.595	10
SE-SqueezeNext-10-0.6x-v1	0.6x	88.18	0.760	11
SE-SqueezeNext-10-0.7x-v1	0.7x	89.39	0.968	12
SE-SqueezeNext-10-0.8x-v1	0.8x	90.33	1.21	12
SE-SqueezeNext-10-0.9x-v1	0.9x	89.79	1.48	13
SE-SqueezeNext-10-1.0x-v1	1.0x	90.48	1.81	13
SE-SqueezeNext-10-1.2x-v1	1.2x	91.04	2.48	15
SE-SqueezeNext-10-1.5x-v1	1.5x	92.07	3.81	16
SE-SqueezeNext-10-1.7x-v1	1.7x	92.10	4.78	18
SE-SqueezeNext-10-2.0x-v1	2.0x	92.60	6.59	21

Table 6.4.
Different dropout layer probabilities results of SE-SqueezeNext

Name of Model	dropout (p)	Accuracy%	Size of Model(MB)	Speed of Model(seconds)
SE-SqueezeNext-10-1x-v1	0.1	91.06	2.16	15
SE-SqueezeNext-10-1x-v1	0.2	90.33	2.16	15
SE-SqueezeNext-10-1x-v1	0.3	90.46	2.16	14
SE-SqueezeNext-10-1x-v1	0.4	90.06	2.16	14
SE-SqueezeNext-10-1x-v1	0.5	90.22	2.16	13
SE-SqueezeNext-10-1x-v1	0.6	90.53	2.16	13

7. IMPLEMENTATION

7.1 NXP BlueBox 2.0 Implementation

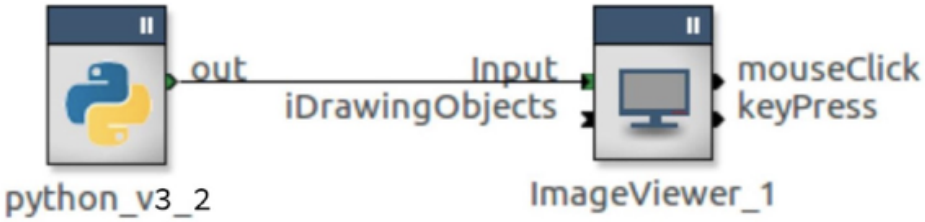


Fig. 7.1. The Python Component in RTMaps.

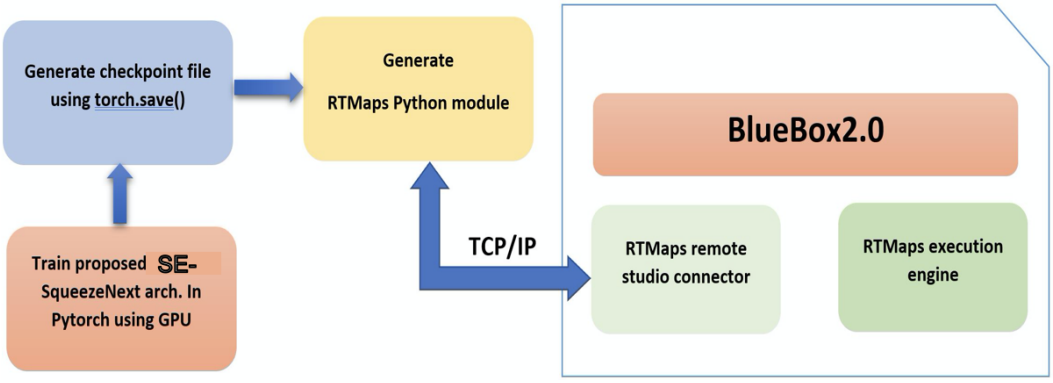


Fig. 7.2. Flowchart of Deployment on NXP BlueBox 2.0.

The python part in RTMaps will permit us to create and incorporate PC vision calculations for ADAS applications like Image arrangement, traffic sign recognition, and driving assistance and so on. The python segment in RTMaps has an editorial manager in it that permits clients to make, create and convey their python contents. Right now, are three principle capacities that are essential to know so as to actualize clients python content in equipment. Birth(), Core() and Death() are the three

capacities that are accessible in the proofreader. Birth() is executed once toward the starting to introduce and set up the code. Core() is a capacity that runs in an unbounded loop. Along these lines, the client's code can be characterized right now permits code to run persistently. Demise() is characterized at the end and it is considered when the program is ended.

The python component in RTMaps is appeared in the Fig. 7.1. This structure of composing code makes it simpler for the client to prototyping and building up their own code as for the application. When the scripting is done, the client can utilize the RTMaps Embedded to run their application on the Bluebox platform. Fig. 7.2 shows the flowchart of RTMaps arrangement with Bluebox 2.0. The association between the have pc and the objective Bluebox is TCP/IP. In the wake of interfacing with have pc, the client can check right COM ports in the gadget administrator. At that point client should arrangement Teraterm for LS2 interface also, S32V interface. Right now, classifier is prepared as it were in GPU yet tried in NXP Bluebox 2.0.

7.2 NXP i.MX RT1060 Implementation

Deploying SE-SqueezeNext on NXP i.MX RT1060 involves two steps, first to convert our model to TensorflowLite model and deploying that tensorflowLite model into the board.

Converting into TensorflowLite Model:

The NXP eIQ is an AI software development environment to create AI applications for implanted processors, for example, i.MX RT hybrid processors. The eIQ programming incorporates neural network compilers, and improved libraries. TensorFlow Lite is one of the derivation motors bolstered by eIQ programming with elite and enhanced memory use than TensorFlow. TFLite-Converter takes a current model in the keras system and creates the TensorFlow Lite FlatBuffer document (.tflite). The Python API for TFLiteConverter permits custom articles, for example, activation

functions, loss functions and so forth to be passed during the change procedure. The IDE utilized for this change procedure is Microsoft Visual Studio Code (VS Code). This process can be visualised from Fig. 7.3.

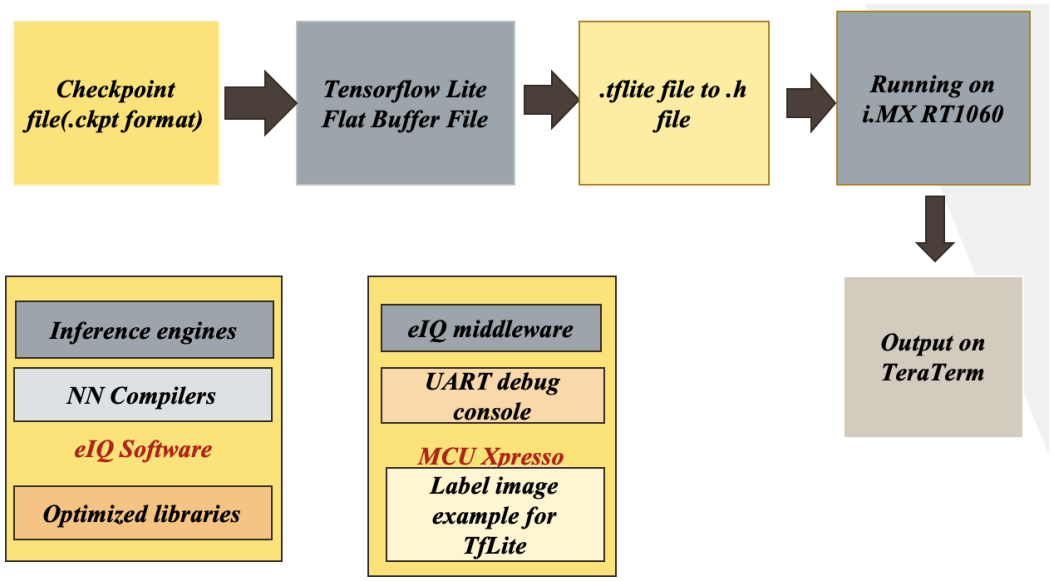


Fig. 7.3. Flowchart of Deployment on i.MX RT1060 MCU.

Deploying on i.MX RT1060:

The MCU Xpresso SDK is explicitly planned by NXP to quicken application advancement in i.MX RT hybrid processors. The most recent adaptation incorporates the refreshed eIQ libraries and demos. This SDK additionally bolsters UART investigate support to run the application on Teraterm. The tflite model is changed over into a C array header file (.h) that can be imported on the board. The API call is utilized in the code to stack the model utilizing this header file. At that point, the model is fixed and we can see the result in Teraterm.

7.3 Implementation Results

In this study, we have considered a pretrained SE-SqueezeNext model. This is trained and validated for CIFAR-10 dataset. We used Nvidia Geforce GTX 1080Ti

GPU for training of the model. The original network is trained using the Pytorch framework with a total number of epochs to 200 and with a variable learning rate of 0.1, 0.01 and 0.001. We have used Stochastic gradient descent (SGD) optimizer with nestrov, decay and momentum. 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. The results of the deployment are as follows.

7.3.1 With NXP BlueBox 2.0

Here, we are endeavoring to make classifier work effectively on NXP Bluebox 2.0. In this way, the model is sustained with a few irregular pictures taken from the test dataset with right ground truth esteems and requesting that the model anticipate those arbitrary pictures. The RTMaps Console result is appeared underneath in Fig. 7.4. The BlueBox result can be seen utilizing Teraterm terminal. The Teraterm result can be seen underneath in Fig. 7.5. The model is given some arbitrary information pictures like cat, boat and plane. It accurately predicts those pictures on NXP Bluebox 2.0.

7.3.2 With NXP i.MX RT1060 MCU

Right now, attempted to give some irregular pictures like cat and plane, requesting that the model foresee them. For the most part, we gave cat and plane on the grounds that these classes have a place with the CIFAR10 dataset and our model is just prepared to this dataset. The result can be seen in Teraterm terminal. Our model is effectively capable to characterize cat and airplane images effectively on NXP i.MX RT1060 alongside inference times appeared in the Fig. 7.7 and Fig. 7.8.

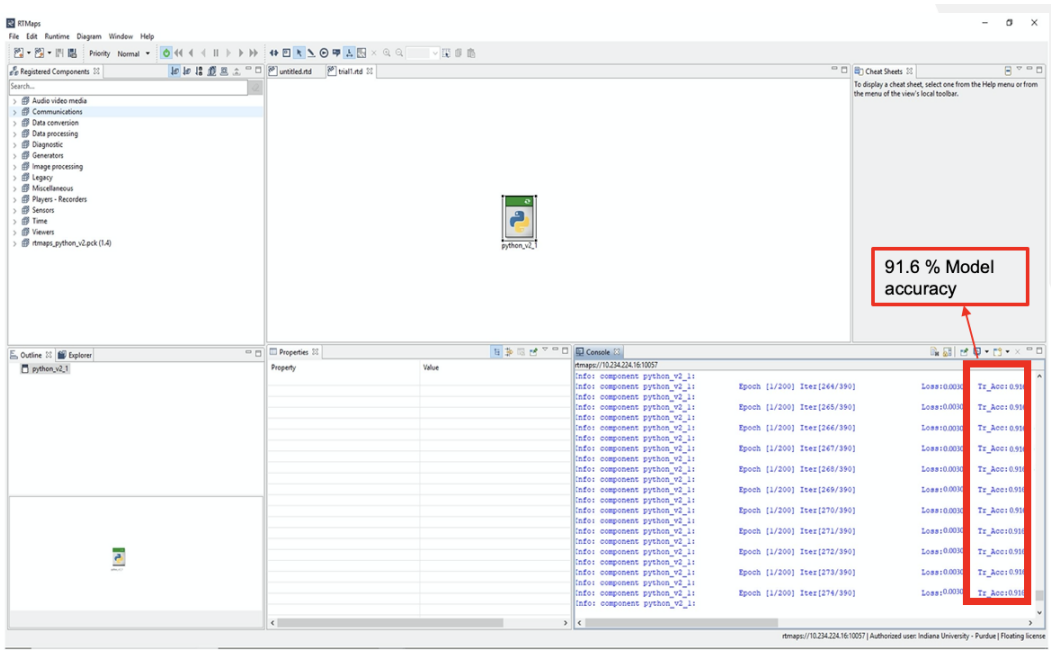


Fig. 7.4. The RTMaps Console Result.

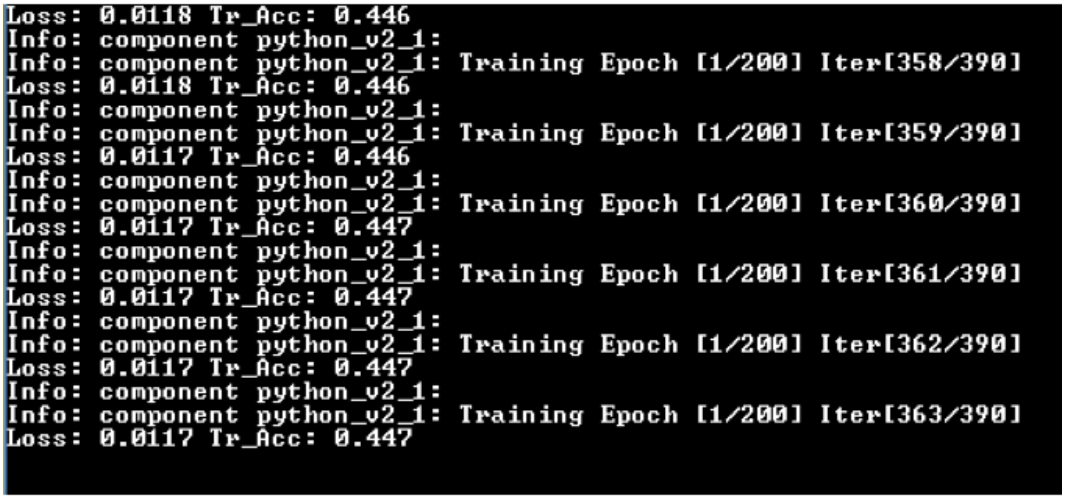


Fig. 7.5. The Teraterm Result of the Deployment.

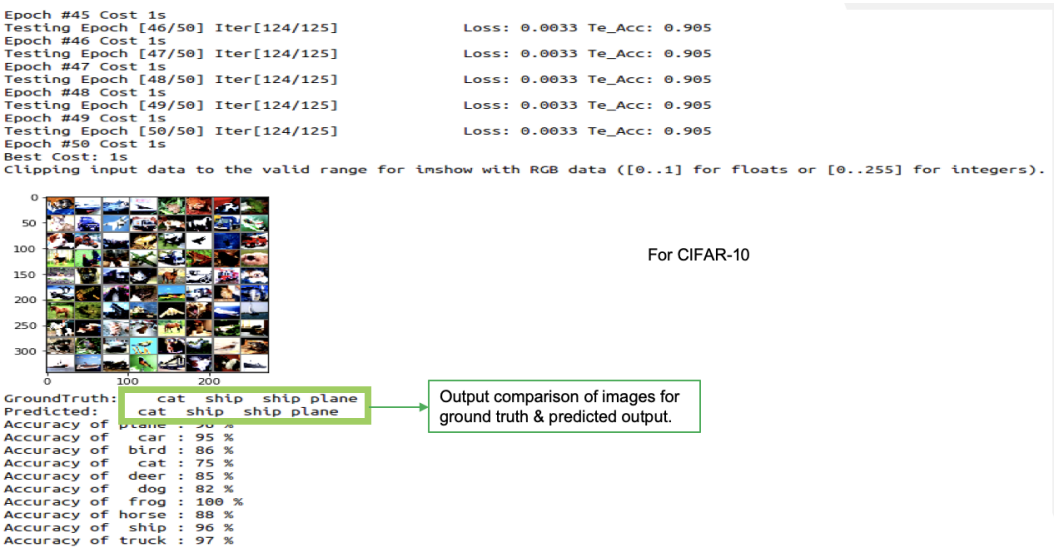
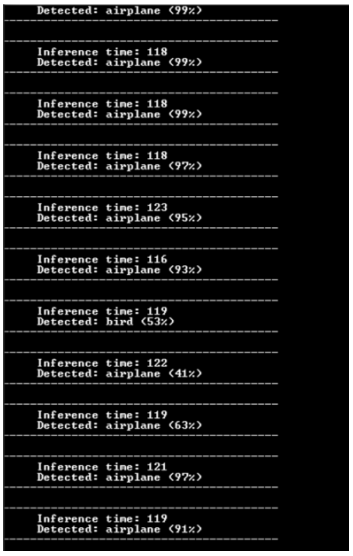


Fig. 7.6. The Image Classifier Result on the Console.

(a) Input Cat Image. (b) Teraterm result.

Fig. 7.7. Results of Successful Recognition of Cat.

(a) Input Airplane Image.



(b) Teraterm result.

Fig. 7.8. Results of Successful Recognition of Airplane.

8. CONCLUSION

It is obvious from the outcomes that there is a tradeoff between model's size, exactness and speed for various goals and width multipliers. We can likewise see that there is no adjustment in the exactness of model after decreasing the profundity of model. Determination of hyperparmeters, for example, width and goals multipliers are the essential factors in misfortune minimization, acquiring a decent size and a precise model. From the outcomes, the exhibition of SGD streamlining agent alongside nestrov, rot and force are seen to be superior to different terms. Proposed engineering was prepared and approved on CIFAR-10 with a superior model size of 0.595 MB which is 6x better than SqueezeNet Baseline & 16x better than SqueezeNext pattern. A best model speed of 9 sec which is 10 sec better than SqueezeNext benchmark and simultaneously like SqueezeNet pattern. As an expansion to this work, for expanding the presentation of this engineering move learning and information enlargement can be utilized.

It is also evident from the results that we have successfully deployed SE-SqueezeNext architecture on flexible and highly computational embedded platforms like NXP BlueBox 2.0 and NXP i.MX RT1060. The model is as small as 0.595MB and with accuracy of 92.60% makes it the appropriate choice for the deployment on the respective embedded platforms. There can be many tweaks be implemented on the existing architecture to make it more efficient for deployment on real-time systems. This can also be further developed for object tracking and detection applications. On a further note, this model can also be tested for deployment on NXP i.MX 8M Mini MCU which is more advanced than NXP i.MX RT1060 MCU.

REFERENCES

REFERENCES

[1] Duggal, J. and El-Sharkawy, M., "Shallow SqueezeNext: An Efficient & Shallow DNN," 2019 IEEE International Conference of Vehicular Electronics and Safety (ICVES), Cairo, Egypt, 2019, pp. 1-6. doi: 10.1109/ICVES.2019.8906416

[2] Duggal, J., 2019. Design Space Exploration of DNNs for Autonomous Systems (MSECE Thesis, Purdue University, Indianapolis).

[3] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J. and Keutzer, K., (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡ 0.5MB model size.arXiv preprint arXiv:1602.07360, Last Accessed: (April, 01 2020).

[4] Gholami, A., Kwon, K., Wu, B., Tai, Z., Yue, X., Jin, P., Zhao, S. and Keutzer,K., 2018. Squeezenext: Hardware-aware neural network design. In Proceedings ofthe IEEE Conference on Computer Vision and Pattern Recognition Workshops(pp. 1638-1647).

[5] Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand,T., Andreetto, M. and Adam, H., 2017. Mobilenets: Efficient convolutionalneural networks for mobile vision applications. (online) arXiv preprint arXiv:1704.04861, (Last Accessed: April 01, 2020).

[6] Hu, J.,Shen, L., Albanie, S., Sun, G. and Wu, E. "Squeeze-and-Excitation Networks," in IEEE Transactions on Pattern Analysis and Machine Intelligence. doi: 10.1109/TPAMI.2019.2913372

[7] Ashraf, K., 2016. "Shallow networks for high-accuracy road object-detection." arXiv preprint arXiv:1606.01561 (2016) (Last Accessed: April, 01 2020).

[8] Ioffe, S. and Szegedy, C., 2015. Batch normalization:Accelerating deepnetwork training by reducing internal covariate shift. (online) arXiv preprintarXiv:1502.03167, (Last Accessed: April, 01 2020).

[9] Botev, A., Lever, G. and Barber, D., "Nesterov's accelerated gradient and momentum as approximations to regularised update descent," 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, 2017, pp. 1899-1903.

[10] Srivastava, N. "Dropout: a simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

[11] Stanley, K., and Miikkulainen, R. "Evolving neural networks through augmenting topologies." Evolutionary computation 10.2 (2002): 99-127.

[12] Wu, B., Wan, A., Yue, X., and Keutzer, K. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar p

[13] Wu, B., Iandola, F., Jin, P. H., and Keutzer, K. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. arXiv preprint arXiv:1612.01051, 2016 (Last Accessed: April, 01 2020).

[14] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. arXiv preprint arXiv:1611.10012, 2016 (Last Accessed: April, 01 2020).

[15] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093, 2014 (Last Accessed: April, 01 2020).

[16] Sindhwani, V., Sainath T., and Kumar, S. Structured transforms for small-footprint deep learning. In Advances in Neural Information Processing Systems, pages 3088–3096, 2015.

[17] Krizhevsky, A., Nair, V., and Hinton, G. "Cifar-10 (canadian institute for advanced research)." URL http://www. cs. toronto. edu/kriz/cifar. html (2010) (Last Accessed: April, 01 2020).

[18] Huang, Y. "Gpipe: Efficient training of giant neural networks using pipeline parallelism." arXiv preprint arXiv:1811.06965 (2018) (Last Accessed: April, 01 2020).

[19] Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems, pages 91–99, 2015.

[20] Hsiao, T., Chang, Y. and Chiu, C., "Filter-based Deep-Compression with Global Average Pooling for Convolutional Networks," 2018 IEEE International Workshop on Signal Processing Systems (SiPS), Cape Town, 2018, pp. 247-251.

[21] Zoph, B., Cubuk, E., Ghiasi, G., Lin, T., Shlens, J., V, Q. (2019). Le. Learning Data Augmentation Strategies for Object Detection. arXiv preprint arXiv : 1906.11172 (Last Accessed: April, 01 2020).

[22] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. arXiv preprint arXiv:1512.00567, 2015 (Last Accessed: April, 01 2020).

[23] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A.. Going deeper with convolutions. In Proceedings of the IEEE.

[24] Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015 (Last Accessed: April, 01 2020).

[25] Ludermir, T., Yamazaki, A., and Zanchettin, C. "An optimization methodology for neural network weights and architectures." IEEE Transactions on Neural Networks 17.6 (2006): 1452-1459.

[26] Zeiler, M., and Fergus, R. "Visualizing and understanding convolutional networks." European conference on computer vision. Springer, Cham, 2014.

[27] Guo, Y., Yao, A., and Chen, Y. "Dynamic network surgery for efficient dnns." Advances In Neural Information Processing Systems. 2016.

[28] Denton, E. "Exploiting linear structure within convolutional networks for efficient evaluation." Advances in neural information processing systems. 2014.

[29] Chetlur, S. "cudnn: Efficient primitives for deep learning." arXiv preprint arXiv:1410.0759 (2014) (Last Accessed: April, 01 2020).

[30] Szegedy, C., Ioffe, S. and Vanhoucke, V. Inception-v4, inception-resnet and the impact of residual connections on learning. arXiv preprint arXiv:1602.07261, 2016 (Last Accessed: April, 01 2020).

[31] Jaderberg, M., Vedaldi, A., and Zisserman, A. "Speeding up convolutional neural networks with low rank expansions." arXiv preprint arXiv:1405.3866, 2014 (Last Accessed: April, 01 2020).

[32] Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I. and Lempitsky, V. "Speeding-up convolutional neural networks using fine-tuned cpdecomposition." arXiv preprint arXiv:1412.6553, 2014 (Last Accessed: April, 01 2020).

[33] Wu, J., Leng, C., Wang, Y., Hu, Q., and Cheng, J. "Quantized convolutional neural networks for mobile devices". arXiv preprint arXiv:1512.06473, 2015 (Last Accessed: April, 01 2020).