ASTES

*Advances in Science, Technology and Engineering Systems Journal Vol. 7, No. 3, 70-81 (2022)*

www.astesj.com

**ASTESJ**

**ISSN: 2415-6698**

# High Performance SqueezeNext: Real time Deployment on Bluebox 2.0 by NXP

Jayan Kant Duggal*, Mohamed El-Sharkawy

*Department of Electrical and Computer Engineering, IoT Collaboratory, Purdue School of Engineering and Technology, IUPUI, Indianapolis, INDIANA, USA*

ARTICLE INFO

*Article history:*
*Received: 13 February, 2022*
*Accepted: 04 May, 2022*
*Online: 25 May, 2022*

*Keywords:*
*Bluebox 2.0*
*Convolution Neural Networks (CNNs)*
*Deep Learning*
*Deep Neural Networks (DNNs)*
*Modified SqueezeNext*
*Real-time deployment*
*SqueezeNext*

ABSTRACT

*DNN implementation and deployment is quite a challenge within a resource constrained environment on real-time embedded platforms. To attain the goal of DNN tailor made architecture deployment on a real-time embedded platform with limited hardware resources (low computational and memory resources) in comparison to a CPU or GPU based system, High Performance SqueezeNext (HPS) architecture was proposed. We propose and tailor made this architecture to be successfully deployed on Bluexbox 2.0 by NXP and also to be a DNN based on pytorch framework. High Performance SqueezeNext was inspired by SqueezeNet and SqueezeNext along with motivation derived from MobileNet architectures. High Performance SqueezeNext (HPS) achieved a model accuracy of 92.5% with 2.62MB model size at 16 seconds per epoch model using a NVIDIA based GPU system for training. It was trained and tested on various datasets such as CIFAR-10 and CIFAR-100 with no transfer learning. Thereafter, successfully deploying the proposed architecture on Bluebox 2.0, a real-time system developed by NXP with the assistance of RTMaps Remote Studio. The model accuracy results achieved were better than the existing CNN/DNN architectures model accuracies such as alexnet_tf (82% model accuracy), Maxout networks (90.65%), DCNN (89%), modified SqueezeNext (92.25%), Squeezed CNN (79.30%), MobileNet (76.7%) and an enhanced hybrid MobileNet (89.9%) with better model size. It was developed, modified and improved with the help of different optimizer implementations, hyper parameter tuning, tweaking, using no transfer learning approach and using in-place activation functions while maintaining decent accuracy.*

## 1. Introduction

The dream of achieving a true human experience lies within the domain of cybernetics, machine learning, deep learning and AI. AI is currently responsible for transcending hard coded application based programmed machines to artificially intelligent machines with some situational awareness.

All the existing CNN or DNN models trained and tested on large datasets occupy extensive computational and memory resources. In the last couple of years, with the introduction of new CNN or DNN based macro architectures such as ViT [1], CaiT, BiT [2], EfficientNetv2 [3], LaNet [4,5], GPipe [6], enhanced MobileNets [7], SqueezeNet [8], SqueezeNext [9], etc., deep learning became better and more efficient in terms of CNN/DNN model performance than the traditional ones [10,11]. The model efficiency, model performance and its ability to be deployed on limited resource constraint [12] real-time platform was attenuated majorly due to following factors such as design space exploration (DSE) of DNNs [13], hyper parameter tuning and tweaking, different optimizers [14], and activation functions implementation, regularization methods, and powerful hardware accelerators. These existing architectures were never tailor made for deployment on real-time embedded systems with limited resources. This research also makes an effort to develop a new architecture with an impressive model size under 5 MB while maintaining an impressive model accuracy.

In this research, a new architecture called High Performance SqueezeNext [15] was developed in order to attenuate the succeeding various deployment problems of DNN based architectures such as DNN deployment on resource constrained real-time platforms [16,17], DNN model compression, over fitting,

*Corresponding Author: Lu Xiong, Email: lu.xiong@mtsu.edu


https://dx.doi.org/10.25046/aj070308

maintaining a competitive model accuracy without major compromises and developing a hardware aware DNN architecture. This DNN based architecture was inspired and derived from the valuable insights of SqueezeNext and SqueezeNet architectures. The fundamental blocks of High Performance SqueezeNext architecture were derived from the fire modules of SqueezeNet [8,18] and bottleneck modules of SqueezeNext architectures [9].

This architecture tries to attenuate the SqueezeNext architecture problems namely model compression, deploy ability on a real-time embedded platform, efficiently working with resource constraint embedded platforms rather than GPUs or CPUs, incorporating newly developed leaning rate techniques such as cosine annealing, step-based decay, cyclic, and cosine annealing warm restarts. Baseline SqueezeNext architecture was a Caffe based architecture and was not able to utilize the power Pytorch based libraries and functions such as the above-mentioned learning rate decay functions, newly introduced optimizers such as Adaboost, Adabound, in-place activation functions and also some new activation functions in contrast to High Performance SqueezeNext architecture [15] that was developed entirely on a Pytorch based framework.

After strenuous training and testing of High Performance SqueezeNext architecture on multiple datasets such as CIFAR-10 [19] and CIFAR-100 [19], implementing several optimizers, activation functions, incorporating and replacing regular operations with in-place operations, reducing stride in subsequent layers, using preliminary data augmentation and some model compression, it achieved impressive model performance. Also, one of several important factors contributing to the success of this architecture was training and testing High Performance SqueezeNext [15] without any form of transfer learning along with some model compression. Finally, High Performance SqueezeNext was deployed on a real time embedded system, Bluebox 2.0 [16] with the assistance of RTMaps software platform.

This research was focused to deploy the proposed High Performance SqueezeNext [15] comprehensively on real-time embedded platform, Bluebox 2.0 [16] by NXP, explore the major hyperparameter tuning with no transfer learning [13,18], develop a Pytorch framework DNN in order to be deployed on Bluebox 2.0 and compare the proposed architecture with several other pytorch based CNN/DNN based architecture.

## 2. Literature Review

Deep learning transformed the artificial and machine learning domain with the introduction of deep convolutional neural networks. CNNs/DNNs are tweaked and tuned with the hyper parameters, newly introduced large datasets, powerful hardware, model compression, and data augmentation [20] to attain better results. Also, batch normalization [21] is observed to be a major contributor for improving DNN performance. Other prominent factors include use of skip connections [9], data preprocessing techniques, regularization, and number of pooling layers. CNNs/DNNs are used to develop image classifiers [10,22,23], object detectors, object recognizers and object segmentation. In order to solve the problem of real time embedded system DNN deployment with limited resources, a requirement for CNN/DNN architectures is introduced. Recently introduced macro architectures such as SqueezeNet, SqueezeNext, and Shallow SqueezeNext [17] were successfully able to overcome computation and memory resources limitations of CNNs and DNNs.

The research in this paper is heavily oriented on the following discussed architectures in detail: SqueezeNet [8] and SqueezeNext [9] architectures. Some motivation is also derived from MobileNet architecture [24]-[26]. Other techniques used are hyperparameter tuning and tweaking, implementation of different optimizers [14], activation functions and regularization, data augmentation and some data compression (width and depth wise compression) that was introduced within SqueezeNext architecture [9]. High Performance SqueezeNext [15] architecture is majorly based on SqueezeNext followed by SqueezeNet architecture. The fundamental blocks and block structures of High Performance SqueezeNext of the proposed architecture were inspired by fire modules (figure 1), basic blocks (figure 6) of various architectures, and bottleneck modules [9], respectively. The proposed architecture, HPS, was modified and improved with the help of no transfer learning approach, use of in-place functions that helped to reduce mathematical operations being performed with each layer and blocks of the architecture.

### *2.1.* SqueezeNet

The first baseline CNN architecture, baseline SqueezeNet [8] was utilized in this research for inspiration of High Performance SqueezeNext architecture [15]. SqueezeNet baseline architecture is made up of 1x1 and 3x3 convolutions, fire modules, max pooling layers, Relu and Relu in place activation layers, softmax activation, and kaiming uniform initialization.

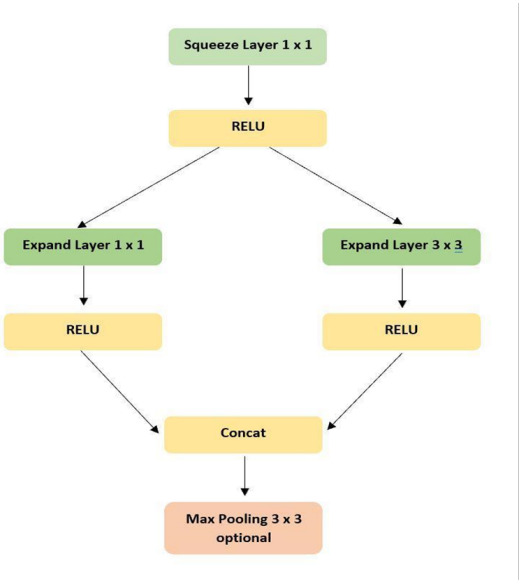


Figure 1. Fire module

Fire module [8] (figure 1.) is the main component of this architecture that consists of one squeeze layer, s2 (1x1), two expand layers, e1 (1x1) and e3 (3x3). This architecture consists of following highlighted factors such as 3x3 convolutions replaced with 1x1 convolutions, number of input channels decreased to 3x3 convolutions, and down sampling max pooling late down the CNN.

Fire module (figure 1.) is the main component of this architecture that consists of one squeeze layer, s2 (1x1), two expand layers, e1 (1x1) and e3 (3x3). This architecture consists of following highlighted factors such as 3x3 convolutions replaced with 1x1 convolutions, number of input channels decreased to 3x3 convolutions, and down sampling max pooling late down the CNN.

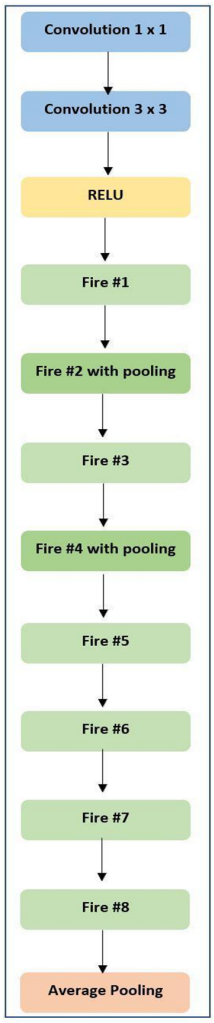


Figure 2. SqueezeNet baseline architecture

SqueezeNet provides some valuable insights relevant to fire modules in accordance with design significance of the fundamental building blocks of CNNs/DNNs and its effect on the entire architecture itself affecting a DNN architecture performance. Squeezed CNN [18] was compared in this research with the proposed High Performance SqueezeNext [15] architecture. Squeezed CNN is an architecturally compressed version of SqueezeNet based CNN architecture that was previously successfully deployed on the Bluebox 2.0 real time embedded platform.

### *2.2. SqueezeNext*

SqueezeNext baseline architecture [9], another architecture used for major development and laying the foundation of High Performance SqueezeNext architecture. It is also used for comparison with the proposed architecture called High Performance SqueezeNext. Key factors of the SqueezeNext baseline architecture:

- Aggressive channel reduction by a two-stage squeeze module.
- Use of separable 3x3 convolutions.
- Use of an element-wise addition skip connection.

In SqueezeNext architecture, two stage squeeze model channel reduction, 3x3 separable convolution, and an element-wise skip connection [9] techniques are used to drastically reduce the total number of parameters and computation resource usage. Baseline SqueezeNext architecture comprises of bottleneck modules [9] with four stage implementation (figure 3.), batch normalization layers [21], Relu and Relu (in-place) nonlinear activation layers, max pooling and average pooling layers, Xavier uniform initialization, a spatial resolution layer and a fully connected layer. All these techniques are also utilized in High Performance SqueezeNext architecture.

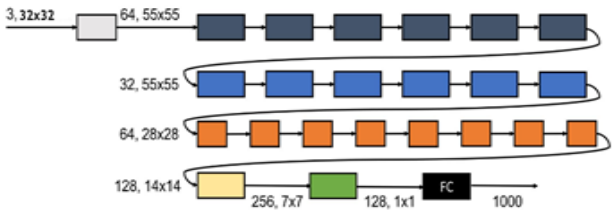


Figure 3. SqueezeNext baseline architecture basic block with [6,6,8,1] four stage implementation configurations trained and tested on CIFAR-10 dataset with no transfer learning [15].

Bottleneck modules [9] are responsible for huge parameters reduction. The consecutive different colored blocks: dark blue, blue, orange, and yellow blocks after the first convolution represent the four-stage configuration implementation referring to a low level (dark blue), medium level (blue and orange), and high-level features (yellow), respectively. Green block represents spatial resolution layer. The baseline SqueezeNext architecture achieves 112x fewer parameters than the AlexNet top-5 performance and 31x fewer parameters than VGG-19 performance.

#### *2.2.1. Modified SqueezeNext*

Modified baseline SqueezeNext is a modified form of the above baseline SqueezeNext architecture. It is derived and modified in order to make a fair comparison with the proposed architecture, High Performance SqueezeNext [13,15] based on a pytorch framework instead of caffe framework. Modified SqueezeNext architecture is trained and tested on CIFAR-10 [19] from scratch and is developed to be implemented with the Pytorch framework.

The fundamental block of baseline SqueezeNext architecture is made up of a convolution layer followed by batch normalization in place, scale in place and ReLU in place layers. In contrast to a fundamental building block of modified SqueezeNext architecture that consists of a convolution layer, batch normalization and a ReLU layer.

The basic blocks depict each of the first individual blocks of the four-stage configuration within the SqueezeNext architectures (the first dark blue, blue, orange and the last

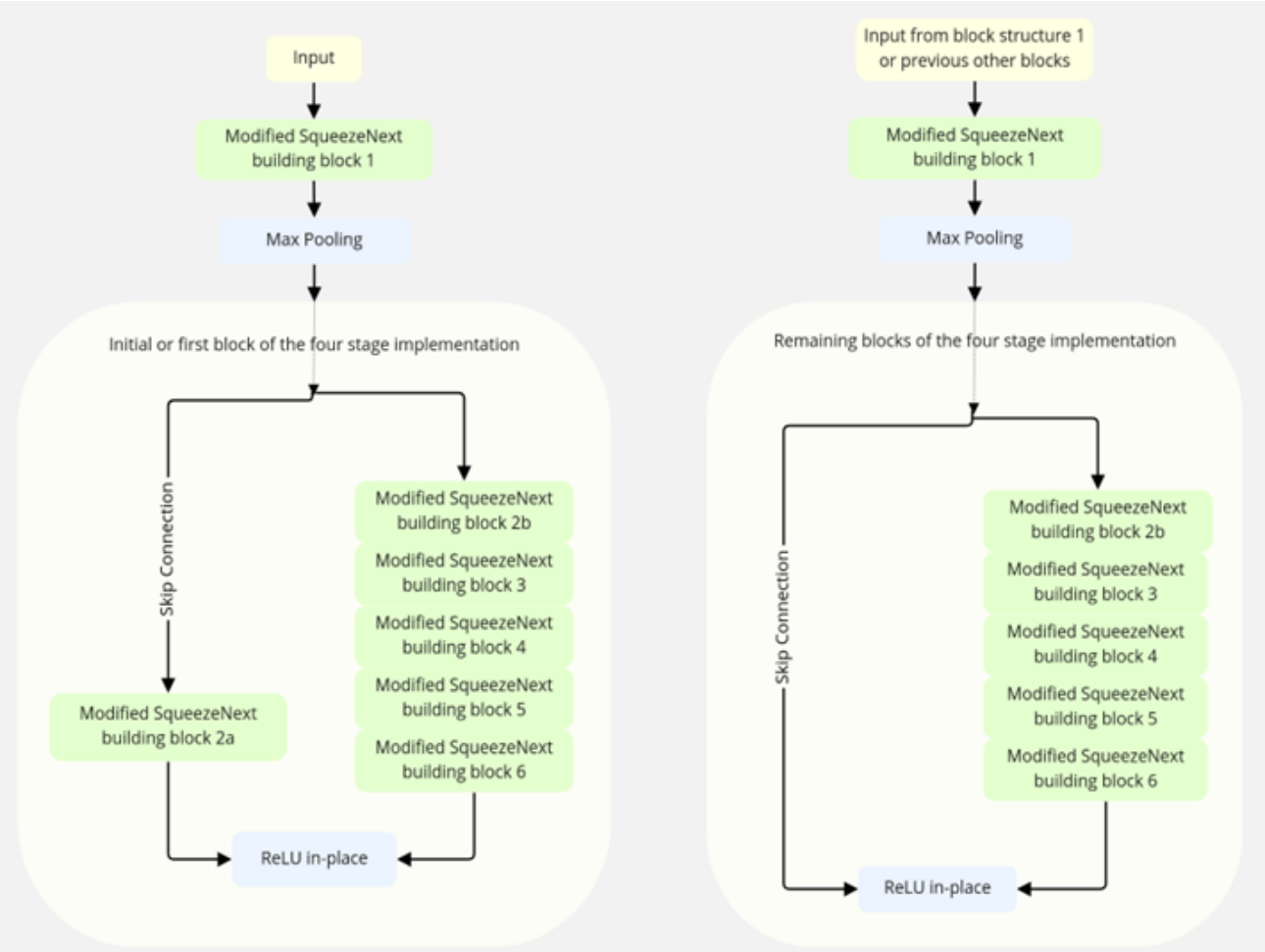


Figure 4: Modified SqueezeNext baseline architecture block structure 1 & block structure 2.

yellow block of the four-stage configuration). All the five SqueezeNext Block modules inside both of the block structures (Figure 4.) are exactly the same fundamental building blocks (convolution layer, batch norm and an activation function layer) described in the previous paragraph.

The efficient organization of these two block structures within a DNN architecture makes a better and efficient DNN. This modified SqueezeNext architecture is trained and tested with datasets such as CIFAR-10 [19] and CIFAR-100 [19]. It was also modified with the help of data augmentation, data compression and different optimizer functions [14] are implemented in order to improve the performance of the modified SqueezeNext architecture [13].

## 3. Hardware & Software Used

- Intel i9 8th generation processor with 32 GB RAM.
- Required memory for dataset and results: 4GB.
- NVIDIA RTX 2080Ti GPU.
- Spyder version 3.7.1.
- Pytorch version 1.1.
- RTMaps Remote Studio
- Linux BSP
- SD card 8GB minimum
- RTMaps Embedded

## 4. Bluebox 2.0 real-time embedded platform

Bluebox 2.0 [16] is a real time development platform by NXP for developing self-driving car applications. It delivers the performance required to analyze ADAS systems or environments. ASIL-B and ASIL-D compliant real time embedded hardware.

Figure 5. Real time embedded platform, Bluebox 2.0 by NXP

It includes three independent SoCs that are S32V234: a vision processor, LS2084A: a compute processor, and S32R274: a radar microcontroller. This research paper was also a way to analyze and test the capability of Bluebox 2.0 as an autonomous embedded platform system for real-time autonomous applications. It can be used or utilized for implementing Level 1- Level 3 autonomous applications. The detailed description of all SoCs is discussed in the following subsections.

### *4.1. Vision Processor (S32V234)*

S32V234 [24] is a micro processing unit consisting of an ISP, powerful 3D GPU, automotive-grade reliability, dual APEX-2 vision accelerators, and functional safety. It provides good

computation support for ADAS, NCAP front camera, object detection and recognition, image processing, machine and deep learning, and sensor fusion applications. The 32-bit based Arm Cortex-A53 S32V processors are supported by S32 Design Studio IDE for development. The software platform studio includes a compiler, debugger, linux BSP vision SDK, and graph tools.

It is a vision-based processor that comprises ISP available on all MIPI-CSI camera inputs. It supports and provides the functionality to integrate multiple cameras. APEX-2 vision accelerators, GPU along with vision accelerators, four ARM Cortex-A53 cores, and an arm M4 core are used for embedded related applications and computer vision functions. It operates on linux BSP, Ubuntu 16.04 LTS and NXP vision SDK. The processor boots up from the SD card with the help of linux BSP.

### *4.2. LS-2084A*

LS2 processor [13,16] embedded in bluebox 2.0 is a high-performance computing processor platform. It comprises ARM Cortex-A72 cores, 10 Gb Ethernet ports, DDR4 memory, and a PCIe expansion slot. It is also a convenient platform to develop the arm-based application or features.

It makes use of *an SD card* interface enabling *its* processor to run linux BSP, Ubuntu 16.04 LTS on the platform. The software enablement on the LS2084A and S32V234 SoC is done with the help of *Linux* BSP. *It* is a complete, developer-supported system *with* eight core QorIQ LS2084A and the four core LS2044A. This *multi-core* processors-based system offers advanced, high-performance data path and network peripheral interfaces required for networking, datacom, wireless infrastructure, military, and aerospace applications.

### *4.3. Real-time Multisensor applications (RTMaps)*

RTMaps [13,17] is an efficient and easy-to-use framework for fast and robust developments. It is a high-performance platform that is the easiest way to develop, test, validate, benchmark, and execute applications. It is used for fusing the data streams in real-time. It consists of several independent modules that can be used in different scenarios. It is described as follows:

- **RTMaps Runtime Engine** is an easily deployable, multi-threaded, highly optimized module that is designed to be integrated with third-party applications. It is also accountable for all base services such as component registration, buffer management, time stamping threading, and priorities.
- **RTMaps Component Library** consists of the software module that can be easily interfaced with the automotive and other related sensors and packages responsible for the development of an ADAS application.
- **RTMaps Remote Studio** is a graphical modeling environment with the functionality of programming using Python packages. It is available for both, windows and ubuntu based operating system platforms. Applications are developed by using the modules and packages available from the RTMaps component library.
- **RTMaps Embedded** is a framework consisting of a component library and the runtime engine with the capability of running on an embedded x86 or ARM capable platform.

The connection between the computer running RTMaps Remote Studio and Bluebox 2.0 platform can be accessed via a static TCP/IP. The detailed approach of High Performance SqueezeNext deployment is explained in the following section 6.

## 5. High Performance SqueezeNext

High Performance SqueezeNext architecture [15] is a compact DNN, heavily inspired from architectures such as baseline SqueezeNet and baseline SqueezeNext with some insights taken from MobileNet architecture [24]. The basic block shown in figure 6 (extreme right) consists of a convolution layer, Relu in place layer, and batch normalization layer forms the building blocks of the High Performance SqueezeNext. This basic block in figure 6 are the blocks in figure 7 that form a bottleneck module.

This bottleneck module forms the blocks, arranged together in a four-stage implementation configuration along with a dropout layer. It is concluded with the descriptions of the two-model shrinking hyper parameters such as the width multiplier and resolution multiplier which are explained below towards the end of this section.

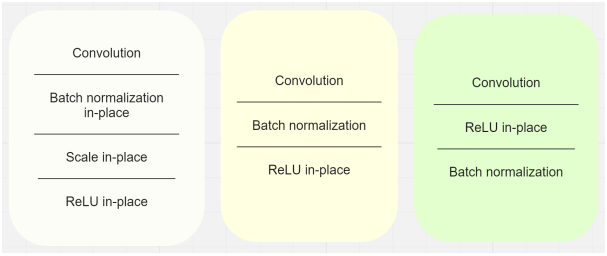


Figure 6: Comparison of baseline SqueezeNext block, Modified SqueezeNext block, High Performance SqueezeNext block, in left to right sequence.

High Performance SqueezeNext is based on the following strategies:

- Using resolution and width multipliers.
- Using only in place operations sandwiching ReLU in-place between convolutional and batch normalization layers.
- An element-wise addition skip connection.
- Adding a drop out after the average pooling layer.
- Minimizing use of any pooling layers.

In detail, the proposed High Performance SqueezeNext architecture comprises bottleneck modules (basic blocks arranged in four stage configuration), a spatial resolution layer, average pooling layer and a fully connected layer. In order to fine tune and tweak the hyperparameters [13] of the proposed architecture different optimizers are implemented.

Modified bottleneck module, shown in the figure 7, comprises a 1x1 convolution, second 1x1 convolution, 3x1 convolution, 1x3 convolution, and then a 1x1 convolution for the proposed High Performance SqueezeNext architecture forms the High Performance SqueezeNext block module. In figure 7, the blocks in figure refer to adaptive forms of High Performance SqueezeNext basic building blocks (right most block in figure 6) that depict the first basic block with 1x1 convolution is High Performance SqueezeNext (HPS) basic building block with 1x1 convolution as a first layer, then, another 1x1 convolution based second HPS basic building block, a third similar block but with first 1x3 convolution layer instead of 1x1 convolution layer. Followed by a similar 3x1 convolution HPS basic building

block and in the end again a 1x1 convolution based HPS building block.

Figure 8 illustrates High Performance SqueezeNext architecture with block structure-1 (top left), four stage implementation and block structure-2 (bottom left). This block structure-1 is responsible for feature extraction from the initial input blocks with the different colors of the four stage implementation of this proposed architecture. Figure 8 illustrates block structure-2 (bottom left) of the proposed High Performance SqueezeNext architecture. These block structures form the rest of the colored feature blocks of the four-stage implementation of the High Performance SqueezeNext.

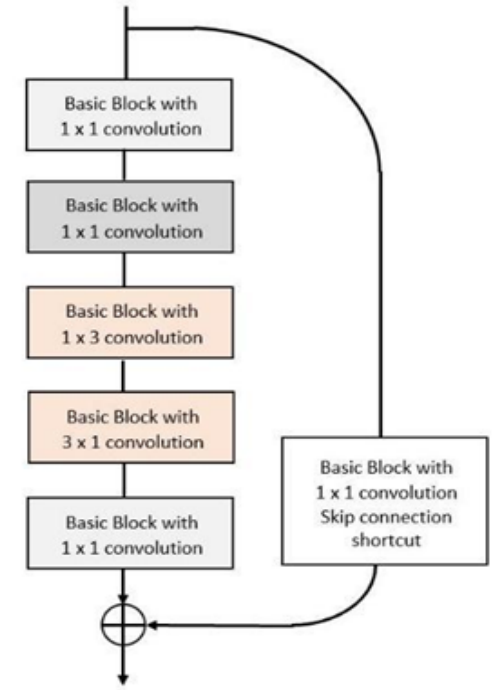


Figure 7. Modified bottleneck module for High Performance SqueezeNext architecture.

*5.1. Resolution Multiplier*

This hyper-parameter, resolution multiplier [9,13], is used to reduce the computational cost of deep neural networks (DNN). It reduces the computational cost and number of parameters.

*5.2. Width Multiplier*

Width multiplier [9,13] is used to construct compact and less computationally expensive models. It is used to thin DNN at each layer, further reducing the number of parameters to roughly twice the power of the width multiplier term.

*5.3. Architecture optimization*

Other few factors which contributed towards the improvement of the proposed High Performance SqueezeNext architecture [15] are:

- *Rectified Linear Units (ReLU) in place and ELU in place function operations:* In place operations activation functions help to reduce the number of parameters by performing in place element wise operations. It changes the content of a given linear operator without actually making a copy of it.
- *Different optimizers implementations for training:* In this research, different optimizers were implemented in order to find a tuned and tweaked proposed architecture. The different optimizers implemented here are SGD, SGD with nesterov and momentum, Adabound, Adagrad, Adamax, Adam, Rprop, and RMSprop.
- *Dataset specific training with no transfer learning:* According to this approach, we do not use any transfer model using Python pickle module bound to the specific classes and the exact directory structure used when the model is saved. It saves a path file containing the class.
- Use of adaptive average pooling would help set stride and kernel-size, especially in adaptive pooling the stride and kernel size are set automatically.
- Real-time embedded platform deployment, Bluebox 2.0 [24, 27].

Specifically, we train data on a powerful GPU based system generating a checkpoint file for testing. This testing file is further used within Bluebox 2.0 for deployment of the DNN architecture.

## 6. Bluebox 2.0 architecture deployment

The proposed architecture High Performance SqueezeNext [15] is initially, trained on system with RTX 2080ti equipped gpu. After successfully testing on an average of three times for a model, generate a checkpoint file using save and load checkpoint method for pytorch. For testing, the python-based module is connected to a real-time embedded system, Bluebox 2.0 with the help of RTMaps Remote studio connector.

RTMaps Remote studio and RTMaps Embedded provide support for the pytorch framework that currently is empowered by a huge collection of libraries for machine learning and deep learning support. Figure 9 represents the process of High Performance SqueezeNext architecture deployment on the real-time embedded system platform, Bluebox 2.0 by NXP. RTMaps studio initiates a connection to the execution engine using TCP/IP that runs the software on Linux BSP and then, installed on Bluebox 2.0. RTMaps provides a python block to create and deploy python Pytorch code.

Python code for RTMaps comprises three function definitions: birth (), core (), and death (). Due to organized structure, and flexibility within RTMaps, it makes it easy to develop a modular code. LS2084A processor is used for the maximum utilization of available 8 ARM Cortex-A72 cores to run RTMaps. The deployment process for High Performance SqueezeNext on the real time platform, Bluebox 2.0 by NXP.

**Birth ():** It is executed once, initially, for setting up and initializing the python environment.

**Core ():** It is an infinite loop function to keep the code running continuously.

**Death ():** It is used to perform cleanups and memory release after the python code terminates.

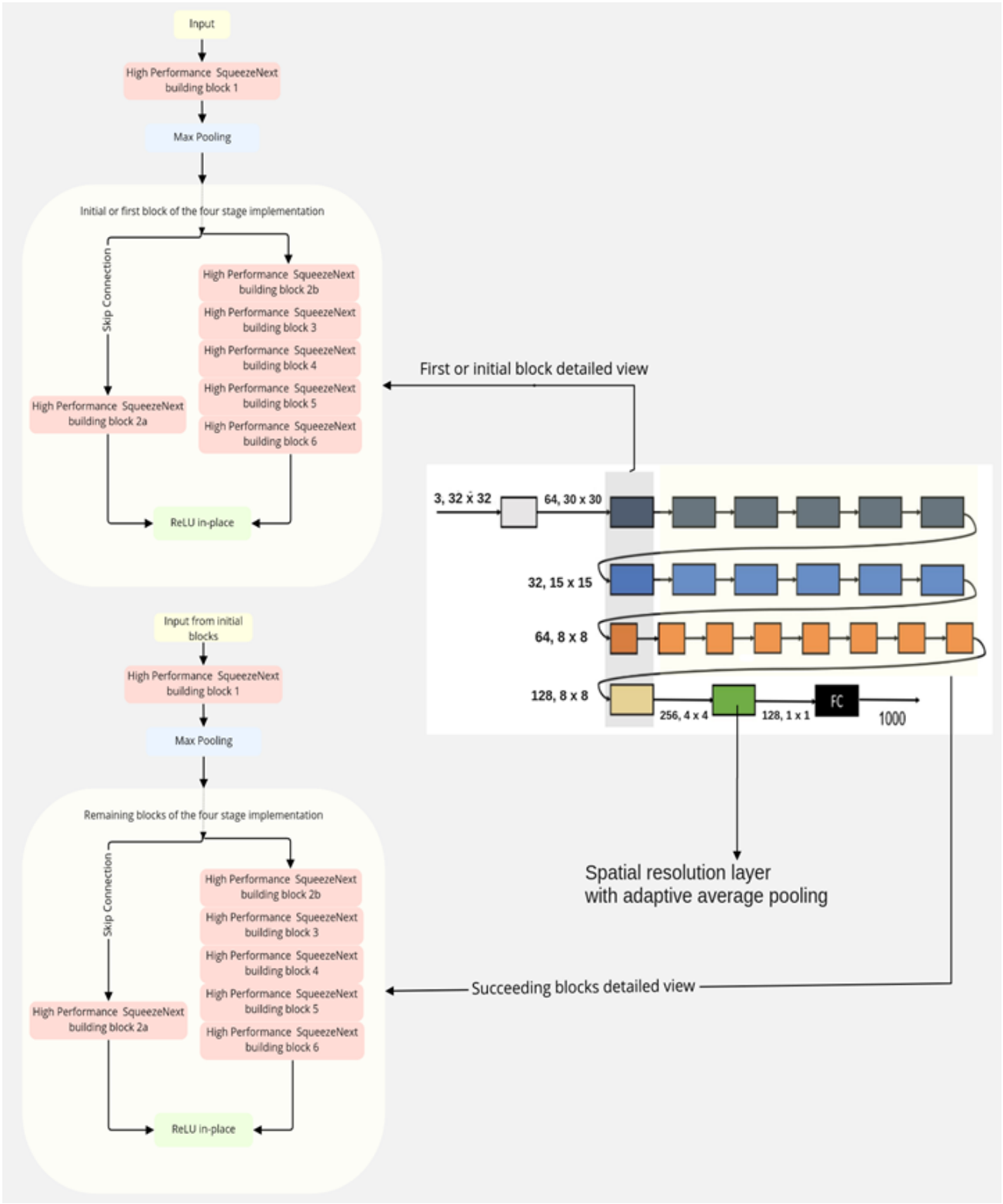


Figure 8. High Performance SqueezeNext block structure 1, High Performance SqueezeNext four stage implementation and HPS block structure 2.

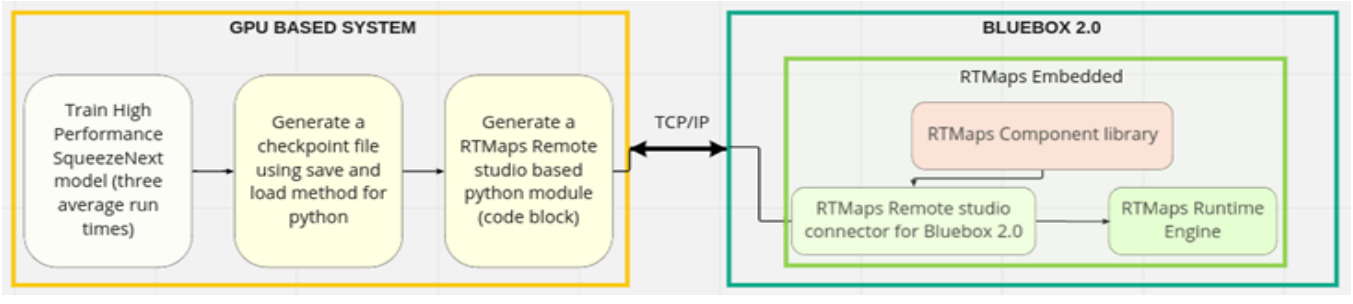


Figure 9: Bluebox 2.0 deployment process for High Performance SqueezeNext architecture overview.

## 7. Results

### 7.1. High Performance SqueezeNext Results

High Performance SqueezeNext architecture is implemented with the development and optimization approaches mentioned in the section 5. This research led to different variants of High Performance SqueezeNext models. Also, it leads to another compressed and shallow depth based DNN architecture called Shallow SqueezeNext [17,25,26].

The High Performance SqueezeNext model size ranges from 10.8MB to a small size of 370KB as shown in Table1 and Table2 with model accuracy between 70% to 93% and model speed of approximately under 25 seconds per epoch for the experimental models. In the following tables, only a few of the several better model's results were shown out of the total 500 models or experiments.

The nomenclature for the proposed High Performance SqueezeNext model in all the tables in section 7 illustrates the proposed High Performance SqueezeNext architecture version name followed by width and resolution multiplier, and version number.

#### 7.1.1 Model Accuracy Improvement

We can infer from Table 1. that a better High Performance SqueezeNext model accuracy is achieved that is 92.50% with a model size of 2.6 MB and 18 seconds per epoch.

High Performance SqueezeNext -21-1x-v3 model has 12.91% and 14.95% better accuracy along with little decrease of 0.399MB and 0.347MB model size with respect to baseline SqueezeNet- v1.0 and baseline SqueezeNet-v1.1, respectively. High Performance SqueezeNext -21-1x-v3 model has 3.35% and 2.02% better accuracy with respect to baseline SqueezeNext-1x-v1 and baseline SqueezeNext-2x-v1, respectively. All the results obtained in this paper were implemented with the following common hyper parameter values: LR: 0.1, batch size: 128, weight decay: 5e-4, total number of epochs: 200, standard cross entropy loss function and livelossplot package.

Other existing algorithms and methodologies have attained better accuracy than the baseline SqueezeNet and SqueezeNext architecture. However, all the other machine learning or deep learning algorithms use transfer learning techniques, in that the respective model is first trained on a large dataset maybe such as ImageNet and then a pre-trained model is fine-tuned on a smaller datasets like CIFAR-10 [19], CIFAR-100 [19]. Also, these architectures are deeper [10,11,22] and expensive in terms of computation and memory resources [12,13]. Table 1, figure 10 (a), (b) and (c) compares the DNN results for baseline SqueezeNet, baseline SqueezeNext, modified SqueezeNext and High Performance SqueezeNext.

Table 1: High Performance SqueezeNext accuracy improvement

| Model type | Model Acc.% | Model size (MB) | Model speed (sec) |
|---|---|---|---|
| Baseline SqueezeNet-v1.0 | 79.59 | 3.013 | 04 |
| Baseline SqueezeNet-v1.1 | 77.55 | 2.961 | 04 |
| Baseline SqueezeNext-23-1x-v1 | 87.15 | 2.586 | 19 |
| Baseline SqueezeNext-23-2x-v1 | 90.48 | 9.525 | 22 |
| High Performance SqueezeNext -21-1x-v3 | 92.50 | 2.614 | 18 |
| High Performance SqueezeNext -23-1x | 92.25 | 5.14 | 29 |
| High Performance SqueezeNext -21-1x-v2.0 | 92.05 | 2.60 | 16 |
| High Performance SqueezeNext -06-0.5x-v1 | 82.44 | 0.37 | 07 |
| High Performance SqueezeNext -06-1x | 86.82 | 1.24 | 08 |

+*Model Acc. – Model Accuracy*

These graphs depict that the overfitting problem is reduced and becomes less problematic from baseline SqueezeNet architecture [8] to baseline SqueezeNext architecture [9]. Then, again, the overfitting problem got reduced and became better further from modified SqueezeNext architecture in comparison to proposed High Performance SqueezeNext. Due to this reason, we can infer that High Performance SqueezeNext [15] is amongst the better DNN architecture by overcoming the overfitting problem of DNNs along with better resource usage and a competitive accuracy of 92.50%.

#### 7.1.2. Model Size and Model Speed Improvement

The model speed in this paper refers to the time taken per epoch to train and further test the DNN architecture. High Performance SqueezeNext architecture is initially trained with the help of powerful GPUs (GTX 1080 and RTX 2080 Ti).

In general, more powerful hardware (better GPU or multiple GPUs), architecture pruning, and other methods can be implemented to improve the performance of a DNN. To train and test the High Performance SqueezeNext DNN in a better manner, we train and test the proposed DNN on CIFAR-10, CIFAR-100 datasets [13] that are quite small as compared to ImageNet dataset. For better performance of proposed High Performance SqueezeNext architecture model depth as well as model width are modified.

High Performance SqueezeNext architecture is implemented with resolution multiplier and width multiplier, in-place operations [13], and no max-pooling layers used after the four stage implementation but only one adaptive average pooling layer just before the fully connected convolutional layer. This architecture is tuned using hyper parameters such as SGD optimizer with momentum and nesterov values, a step learning rate decay schedule with an exponential learning rate update. We observe in table 2, when High Performance SqueezeNext is trained and tested on CIFAR-10 dataset with no transfer learning approach with a nesterov based SGD optimizer. HPS attains a model accuracy of 92.50% with a decent 2.614MB model size. The results published in table 2 were derived with the help of a gpu based system. Each result entry in table 2 is an average run of three training and validation cycles here.

Table 2: High Performance SqueezeNext Model Speed and Model Size Improvement for CIFAR-10

| Model type | Mod Acc.% | Mod. Size (MB) | Mod. speed (sec) | Resol. | Width |
|---|---|---|---|---|---|
| Baseline SqueezeNet-v1.0 | 79.59 | 3.013 | 04 | - | - |
| Baseline SqueezeNet-v1.1 | 77.55 | 2.961 | 04 | - | - |
| Baseline SqueezeNext-23-1x-v1 | 87.15 | 2.586 | 19 | 6 6 8 1 | 1.0 |
| Baseline SqueezeNext-23-2x-v1 | 90.48 | 9.525 | 22 | 6 6 8 1 | 2.0 |

| | | | | | |
|---|---|---|---|---|---|
| High Performance SqueezeNext -06-0.5x | 82.44 | 0.37 | 07 | 1 1 1 1 | 0.5 |
| High Performance SqueezeNext -06-1x | 86.82 | 1.24 | 08 | 1 1 1 1 | 1.0 |
| High Performance SqueezeNext -06-0.75x | 82.86 | 1.20 | 08 | 1 1 1 1 | 0.75 |
| High Performance SqueezeNext -08-0.5x | 87.37 | 1.41 | 09 | 1 2 2 1 | 0.5 |
| High Performance SqueezeNext -14-1.5x | 89.86 | 3.24 | 18 | 1 2 8 1 | 1.5 |
| High Performance SqueezeNext -21-1x-v2 | 92.05 | 2.60 | 16 | 2 2 14 1 | 1.0 |
| High Performance SqueezeNext -21-1x-v3 | 92.50 | 2.614 | 18 | 2 4 12 1 | 1.0 |
| High Performance SqueezeNext -23-1x | 92.25 | 5.14 | 29 | 2 2 16 1 | 1.0 |

**Mod. Acc.: Model Accuracy; Mod. Size: Model Size; Mod. Speed: Model Speed, Resol: Resolution multiplier, and Width: Width multiplier.*

Table 3: High Performance SqueezeNext Model Speed and Model Size Improvement for CIFAR-100

| **Model type** | **Mod Acc. %** | **Mod. Size (MB)** | **Mod. speed (sec)** | **Resol.** | **Width** |
|---|---|---|---|---|---|
| Baseline SqueezeNet-v1.0 | 51.27 | 6.40 | 04 | - | - |
| Baseline SqueezeNext-23-1x-v1 | 60.37 | 5.26 | 19 | 6 6 8 1 | 1.0 |
| High Performance SqueezeNext -06-0.4x | 55.89 | 0.95 | 06 | 1 1 1 1 | 0.4 |
| High Performance SqueezeNext -06-0.575x | 60.68 | 0.95 | 07 | 1 1 1 1 | 0.575 |
| High Performance SqueezeNext -09-0.5x | 62.72 | 1.10 | 08 | 1 1 4 1 | 0.5 |
| High Performance SqueezeNext -14-1x | 68.46 | 5.00 | 14 | 1 2 8 1 | 1.0 |
| High Performance SqueezeNext -14-1.5x | 69.70 | 10.80 | 18 | 1 2 8 1 | 1.5 |
| High Performance SqueezeNext -23-1x | 68.20 | 7.70 | 25 | 2 2 16 1 | 1.0 |
| High Performance SqueezeNext -25-1x | 70.10 | 7.80 | 25 | 2 4 16 1 | 1.0 |

**Mod.Acc.: Accuracy; Mod. Size: Model Size; Mod. Speed: Model Speed, Resol: Resolution multiplier, and Width: Width multiplier.*

In this paper validation is used interchangeably between testing and validation because CIFAR-10 and CIFAR-100 datasets are in comparison small datasets to imagenet. Testing along with training and validation cycles provides inconclusive and non-reliable results.

Therefore, the whole research in this paper only performs training and testing (referred as validation in case of graphs in Figure 10). It also reflects the significance of resolution and width multiplier used in the proposed architecture.

Table 3. refers to the set of results obtained in the case of training and testing of High Performance SqueezeNext on CIFAR-100 dataset with no transfer learning. Again, each result entry in table 3 is an average run of three training and validation cycles here. It also reflects the importance of width and resolution multipliers and its effect on the proposed High Performance SqueezeNext architecture.

We can also observe the model accuracy significantly drops by 20% on average regardless of any architecture deployment, therefore, verifying that the small datasets do perform poorly with a CNN/DNN based architecture with no transfer learning.

### *7.2. Bluebox 2.0 Results*

High Performance SqueezeNext architecture is deployed on the BlueBox 2.0 platform by NXP. For real-time DNN deployment on Bluebox 2.0 [13,16], DNN parameters are saved and loaded using Pytorch method from a checkpoint file generated during training of DNN with GPU. Then, this saved checkpoint file is loaded with the help of RTMaps on the Bluebox 2.0 using the BSP Linux OS dependencies.

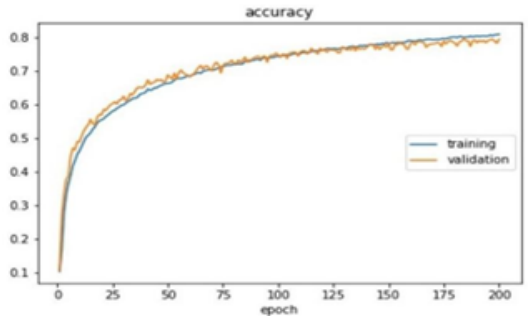


Figure 10: (a) SqueezeNet baseline architecture training and validation (testing) graph representation implemented on CIFAR-10 dataset.

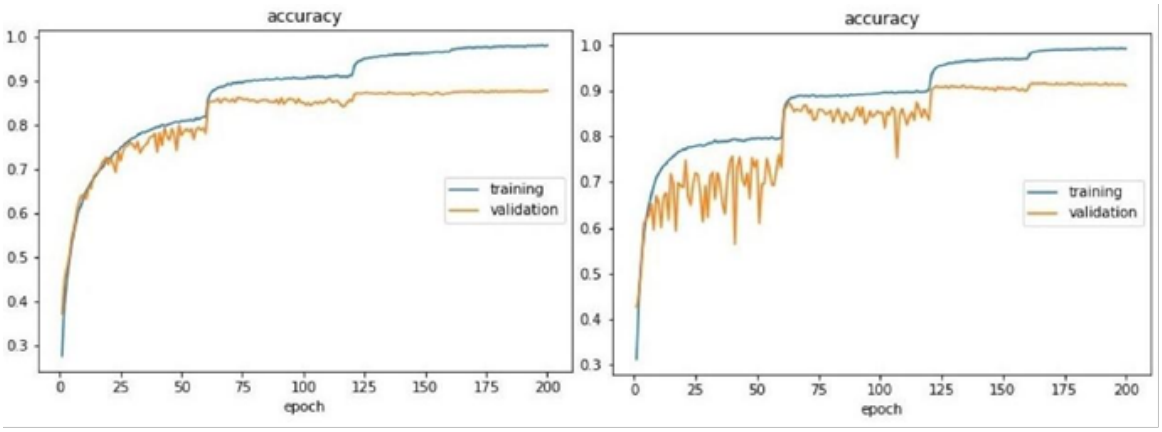


Figure 10: (b) Graph comparison between SqueezeNext baseline architecture and High Performance SqueezeNext architecture training and validation implemented on CIFAR-10 dataset.

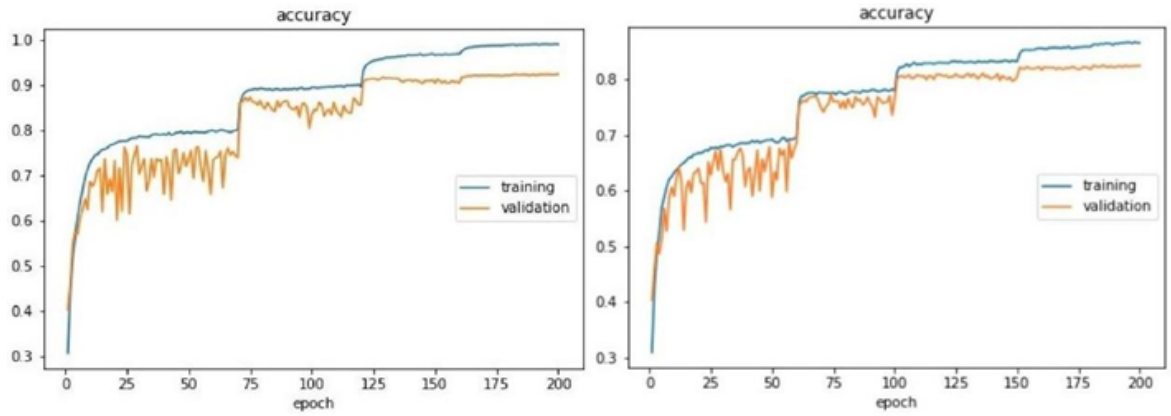


Figure 10: (c) Graph comparison between Modified SqueezeNext baseline architecture and High Performance SqueezeNext architecture training and validation implemented on CIFAR-10 dataset.

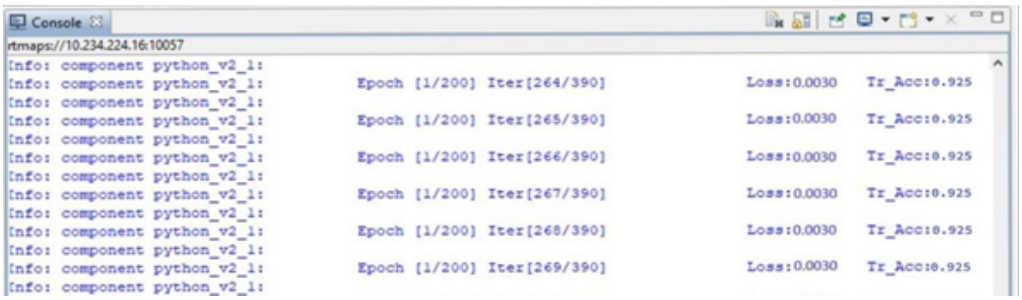


Figure 11: High Performance SqueezeNext architecture deployment on Bluebox 2.0 by NXP with the help of RTMaps Remote Studio.

Table 4: High Performance SqueezeNext deployment results on Bluebox 2.0

| Model type | Mod. Acc. % | Mod. Size (MB) | Mod. Speed (sec) |
|---|---|---|---|
| Squeezed CNN (Baseline SqueezeNet based CNN) | 79.30 | 12.90 | 11 |
| Modified SqueezeNext architecture | 92.25 | 5.14 | 28 |
| High Performance SqueezeNext-21-1x-v3 | 92.50 | 2.62 | 16 |
| High Performance SqueezeNext-06-1.0x | 86.82 | 1.24 | 08 |
| High Performance SqueezeNext-06-0.50x | 82.44 | 0.37 | 06 |

**Mod. Acc.: Model Accuracy; Mod. Size: Model Size; Mod. Speed: Model Speed.*

Table 4 illustrates the results obtained for the proposed High Performance SqueezeNext architecture, baseline SqueezeNext and squeezed CNN architecture [18] representing that the proposed High Performance SqueezeNext performs better and more efficiently. Table 4 illustrates that High Performance SqueezeNext-21-1x-v3 has 92.50% accuracy that is 13.20% better accuracy than Squeezed CNN (SqueezeNet Implementation for Pytorch framework) and 0.25% better accuracy than modified SqueezeNext-23-1x. Further, High Performance SqueezeNext-21-1x is 5x better than Squeezed CNN (SqueezeNet Implementation) and 2x better than modified SqueezeNet.

Also, High Performance SqueezeNext-06-0.5x model attained 0.37MB with 6 seconds per epoch, minimum model size and model speed in comparison to squeezed CNN [18] and modified SqueezeNext models deployed on the Bluebox 2.0 by NXP. All models are trained initially on GPU and then tested on real time platform Bluebox2.0 by NXP with datasets such as CIFAR-10 and CIFAR-100, individually.

Table 5: High Performance SqueezeNext Architecture composition

| **Layer name** | **Input Size** (Wi x Hi x Ci) | **Padding** (Pw x Ph) | **Stride** | **Filter size** (Kw x Kh) | **Output size** (W0 x H0 x C0) | **Parameters** | **Repeat** |
|---|---|---|---|---|---|---|---|
| Convolution 1 | 32 x 32 x 3 | 0 x 0 | 1 | 3 x 3 | 30 x 30 x 64 | 1792 | 1 |
| Convolution 2 | 30 x 30 x 64 | 0 x 0 | 1 | 1 x 1 | 30 x 30 x 16 | 1040 | 1 |
| Convolution 3 | 30 x 30 x 16 | 0 x 0 | 1 | 1 x 1 | 30 x 30 x 8 | 136 | 1 |
| Convolution 4 | 30 x 30 x 8 | 0 x 1 | 1 | 1 x 3 | 30 x 30 x 16 | 400 | 1 |
| Convolution 5 | 30 x 30 x 16 | 1 x 0 | 1 | 3 x 1 | 30 x 30 x 16 | 784 | 1 |
| Convolution 6 | 30 x 30 x 16 | 0 x 0 | 1 | 1 x 1 | 30 x 30 x 32 | 544 | 1 |
| Convolution 32 | 30 x 30 x 32 | 0 x 0 | 2 | 1 x 1 | 30 x 30 x 32 | 1056 | 1 |
| Convolution 33 | 15 x 15 x 32 | 0 x 0 | 1 | 1 x 1 | 15 x 15 x 16 | 528 | 1 |
| Convolution 34 | 15 x 15 x 16 | 0 x 1 | 1 | 1 x 3 | 15 x 15 x 32 | 1528 | 1 |
| Convolution 35 | 15 x 15 x 32 | 1 x 0 | 1 | 3 x 1 | 15 x 15 x 32 | 3104 | 1 |
| Convolution 36 | 15 x 15 x 32 | 0 x 0 | 1 | 1 x 1 | 15 x 15 x 64 | 2112 | 1 |
| Convolution 37 | 15 x 15 x 64 | 0 x 0 | 1 | 1 x 1 | 15 x 15 x 32 | 2080 | 1 |
| Convolution 38 | 15 x 15 x 32 | 0 x 0 | 1 | 1 x 1 | 15 x 15 x 16 | 528 | 1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Convolution 39 | 15 x 15 x 16 | 1 x 0 | 1 | 3 x 1 | 15 x 15 x 32 | 1568 | 1 |
| Convolution 40 | 15 x 15 x 32 | 0 x 1 | 1 | 1 x 3 | 15 x 15 x 32 | 3104 | 1 |
| Convolution 41 | 15 x 15 x 32 | 0 x 0 | 1 | 1 x 1 | 15 x 15 x 64 | 2112 | 1 |
| Convolution 62 | 15 x 15 x 64 | 0 x 0 | 2 | 1 x 1 | 15 x 15 x 64 | 4160 | 1 |
| Convolution 63 | 8 x 8 x 64 | 0 x 0 | 1 | 1 x 1 | 8 x 8 x 32 | 2080 | 1 |
| Convolution 64 | 8 x 8 x 32 | 1 x 0 | 1 | 3 x 1 | 8 x 8 x 64 | 6208 | 1 |
| Convolution 65 | 8 x 8 x 64 | 0 x 1 | 1 | 1 x 3 | 8 x 8 x 64 | 12352 | 1 |
| Convolution 66 | 8 x 8 x 64 | 0 x 0 | 1 | 1 x 1 | 8 x 8 x 128 | 8320 | 1 |
| Convolution 67 | 8 x 8 x 128 | 0 x 0 | 1 | 1 x 1 | 8 x 8 x 64 | 57792 | 7 |
| Convolution 68 | 8 x 8 x 64 | 0 x 0 | 1 | 1 x 1 | 8 x 8 x 32 | 14560 | 7 |
| Convolution 69 | 8 x 8 x 32 | 1 x 0 | 1 | 3 x 1 | 8 x 8 x 64 | 43456 | 7 |
| Convolution 70 | 8 x 8 x 64 | 0 x 1 | 1 | 1 x 3 | 8 x 8 x 64 | 86464 | 7 |
| Convolution 71 | 8 x 8 x 64 | 0 x 0 | 1 | 1 x 1 | 8 x 8 x 128 | 58240 | 7 |
| Convolution 102 | 8 x 8 x 128 | 0 x 0 | 2 | 1 x 1 | 8 x 8 x 128 | 16512 | 1 |
| Convolution 103 | 4 x 4 x 128 | 0 x 0 | 1 | 1 x 1 | 4 x 4 x 64 | 8256 | 1 |
| Convolution 104 | 4 x 4 x 64 | 0 x 1 | 1 | 1 x 3 | 4 x 4 x 128 | 24704 | 1 |
| Convolution 105 | 4 x 4 x 128 | 1 x 0 | 1 | 3 x 1 | 4 x 4 x 128 | 49280 | 1 |
| Convolution 106 | 4 x 4 x 256 | 0 x 0 | 1 | 1 x 1 | 4 x 4 x 256 | 65792 | 1 |
| Convolution 107 Spatial Resolution | 4 x 4 x 256 | 0 x 0 | 1 | 1 x 1 | 4 x 4 x 128 | 32896 | 1 |
| Adaptive Average Pool | 4 x 4 x 256 | - | - | - | 4 x 4 x 256 | - | 1 |
| FCC | 1 x 1 x 128 | 0 x 0 | 1 | 1 x 1 | 1 x 1 x 10 | 1290 | 1 |

*+First column Wi x Hi x Ci refer to input width x input height x input number of channels; Second column, Pw x Ph refer to width and height of padding; third column refers to the number of stride used; fourth column, Kw x Kh refer to width and height of the kernel; fifth column, W0 x H0 x C0 refer to width, height and number of channels for the output; sixth column represents number of parameters for the particular layer, and last column depicts the number of times a layer is repeated in the four stage implementation configuration.*

## 8. Conclusion

In this paper, the existing macro architectures such as baseline SqueezeNet, baseline SqueezeNext and a family of MobileNet architectures had laid foundation and motivated the development of the proposed High Performance SqueezeNext. Fine hyper parameter tuning and tweaking, compression using width and resolution multipliers, implementing different optimizers, step-based learning decay rate, data augmentation, save and load method for python, dataset specific training with no transfer learning approach and real-time embedded platform deployment contributed towards the improvement of the proposed DNN architecture. This research also initiated and encouraged the DSE of DNNs with the help of experiments implementing different activation functions and various optimizers.

Hence, these insights helped to build a better understanding of various optimizers, model compression, learning rate scheduling methods, save and load checkpoint, dataset specific training and testing. High Performance SqueezeNext architecture is one of the several new CNNs/DNNs that have been discovered while broadly exploring the DSE of DNN architectures. Detailed composition of High Performance SqueezeNext is shown in Table 5.

This architecture has 15x and 13x better model accuracy than baseline SqueezeNet and baseline SqueezeNext, respectively. It has a minimal 0.370MB model size, in other words, it is 8x and 7x smaller than baseline SqueezeNet and baseline SqueezeNext baseline. All the results discussed in this paper demonstrate the trade-off between model accuracy, model speed and model size with different resolution and width multipliers. SGD with momentum and nesterov is proposed as a suggested optimizer to be implemented on any DNN architecture. It is expected that with the incredibly small model size of 370KB for High Performance SqueezeNext, referring to Table 2, with an accuracy of 92.05%, referring to Table 1, High Performance SqueezeNext model can be easily deployed on a real time embedded platform.

The proposed, High Performance SqueezeNext architecture is trained and tested from scratch on datasets such as CIFAR-10 and CIFAR-100, individually without any transfer learning. The proposed DNN was successfully deployed on Bluebox 2.0 by NXP with DNN model accuracy of 92.50%, 16 seconds per epoch model speed and 2.62MB of model size.

High Performance SqueezeNext attains model accuracy 15.8% better than MobileNet (76.7%), 13.2% better than Squeezed CNN (79.30%), 10% better than alexnet_tf (82% model accuracy), 3.50% better than DCNN (89%), 2.6% better than enhanced hybrid MobileNet (89.9%), 1.85% better than Maxout networks (90.65%), and 0.25% better than modified SqueezeNext (92.25%) with better model size. Hopefully, this research will inspire design space exploration (DSE) of DNNs in a more intrinsic and aggressive manner.

## Conflict of Interest

The authors declare no conflict of interest.

## Acknowledgment

We would like to acknowledge all the IoT Collaboratory lab members and other colleagues for their continuous support, reviews and regular feedback.

## References

[1] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in International Conference on Learning Representations, 2021.

[2] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, N. Houlsby, Big Transfer (BiT): General Visual Representation Learning,

ECCV 2020:, Springer, Cham, 2020, doi:10.1007/978-3-030-58558-7_29.
[3] M. Tan, Q. V. Le, "EfficientNetV2: Smaller Models and Faster Training," in International Conference on Machine Learning, PMLR: 10096–10106, 2021.
[4] L. Wang, S. Xie, T. Li, R. Fonseca, Y. Tian, "Sample-Efficient Neural Architecture Search by Learning Action Space," arXiv, 2019, doi:10.48550/ARXIV.1906.06832.
[5] L. Wang, S. Xie, T. Li, R. Fonseca, Y. Tian, "Sample-Efficient Neural Architecture Search by Learning Actions for Monte Carlo Tree Search," IEEE Transactions on Pattern Analysis and Machine Intelligence, **01**, 1–1, 2021, doi:10.1109/TPAMI.2021.3071343.
[6] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M.X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, Z. Chen, "GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism," NIPS'19: Proceedings of the 33rd International Conference on Neural Information Processing Systems: 103–112, 2019, doi:10.48550/ARXIV.1811.06965.
[7] H.Y. Chen, C.Y. Su, "An Enhanced Hybrid MobileNet," in 9th International Conference on Awareness Science and Technology, iCAST 2018, Institute of Electrical and Electronics Engineers Inc., Fukuoka: 308–312, 2018, doi:10.1109/ICAwST.2018.8517177.
[8] F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," arXiv, 2016, doi:10.48550/ARXIV.1602.07360.
[9] A. Gholami, K. Kwon, B. Wu, Z. Tai, X. Yue, P. Jin, S. Zhao, K.K. Eecs, "SqueezeNext: Hardware-Aware Neural Network Design," in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE: 1719–171909, 2018, doi:10.1109/CVPRW.2018.00215.
[10] K. Simonyan, A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in International Conference on Learning Representations, 2014.
[11] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, "Rethinking the Inception Architecture for Computer Vision," in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, Las Vegas: 2818–2826, 2016, doi:10.1109/CVPR.2016.308.
[12] K. He, J. Sun, "Convolutional Neural Networks at Constrained Time Cost," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society: 5353–5360, 2015, doi:10.1109/CVPR.2015.7299173.
[13] J.K. Duggal, DESIGN SPACE EXPLORATION OF DNNS FOR AUTONOMOUS SYSTEMS, Purdue University, 2019, doi:https://doi.org/10.25394/PGS.8980463.v1.
[14] S. Ruder, "An overview of gradient descent optimization algorithms," in arXiv preprint arXiv:1609.04747, arXiv, 2016, doi:10.48550/ARXIV.1609.04747.
[15] J.K. Duggal, M. El-Sharkawy, "High Performance SqueezeNext for CIFAR-10," in Proceedings of the IEEE National Aerospace Electronics Conference, NAECON, Institute of Electrical and Electronics Engineers Inc.: 285–290, 2019, doi:10.1109/NAECON46414.2019.9058217.
[16] S. Venkitachalam, S.K. Manghat, A.S. Gaikwad, N. Ravi, S.B.S. Bhamidi, M. El-Sharkawy, "Realtime applications with rtmaps and bluebox 2.0," in Proceedings on the International Conference on Artificial Intelligence (ICAI), The Steering Committee of The World Congress in Computer Science, Computer: 137–140, 2018.
[17] J. Kant Duggal, M. El-Sharkawy, "Shallow SqueezeNext: Real Time Deployment on Bluebox2.0 with 272KB Model Size," Journal of Electrical and Electronic Engineering, **8**(6), 127, 2020, doi:10.11648/j.jeee.20200806.11.
[18] D. Pathak, M. El-Sharkawy, "Architecturally Compressed CNN: An Embedded Realtime Classifier (NXP Bluebox2.0 with RTMaps)," in 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), 331–336, 2019, doi:10.1109/CCWC.2019.8666495.
[19] A. Krizhevsky, V. Nair, G. Hinton, "Cifar-10 (canadian institute for advanced research)," 2019.
[20] T.B. Ludermir, A. Yamazaki, C. Zanchettin, "An optimization methodology for neural network weights and architectures," IEEE Transactions on Neural Networks, **17**(6), 1452–1459, 2006, doi:10.1109/TNN.2006.881047.
[21] S. Ioffe, C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in Proceedings of the 32nd International Conference on Machine Learning - Volume 37, PMLR: 448–456, 2015.
[22] B. Recht, R. Roelofs, L. Schmidt, V. Shankar, "Do CIFAR-10 Classifiers Generalize to CIFAR-10?," arXiv, 2018, doi:10.48550/ARXIV.1806.00451.
[23] H. Touvron, M. Cord, A. Sablayrolles, G. Synnaeve, H. Jégou, "Going deeper with Image Transformers," in Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 32–42, 2021.
[24] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv, 2017, doi:10.48550/ARXIV.1704.04861.
[25] J.K. Duggal, M. El-Sharkawy, "Shallow squeezenext: An efficient shallow DNN," in 2019 IEEE International Conference on Vehicular Electronics and Safety, ICVES 2019, Institute of Electrical and Electronics Engineers Inc., 2019, doi:10.1109/ICVES.2019.8906416.
[26] J.K. Duggal, M. El-Sharkawy, Shallow SqueezeNext Architecture Implementation on Bluebox2. 0, Advances i, Springer, Cham, 2021.