

# AI ON THE EDGE WITH CONDENSENEXT: AN EFFICIENT DEEP NEURAL NETWORK FOR DEVICES WITH CONSTRAINED COMPUTATIONAL RESOURCES

by

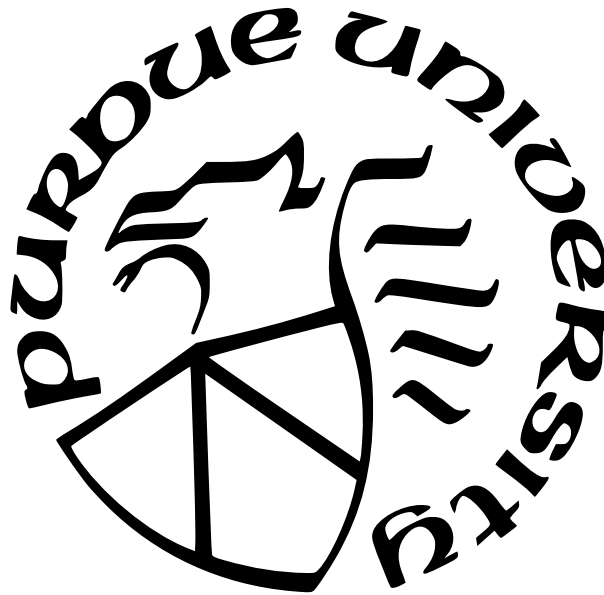
Priyank B. Kalgaonkar

A Thesis

*Submitted to the Faculty of Purdue University*

*In Partial Fulfillment of the Requirements for the degree of*

Master of Science



Department of Electrical and Computer Engineering

Indianapolis, Indiana

August 2021

**THE PURDUE UNIVERSITY GRADUATE SCHOOL  
STATEMENT OF COMMITTEE APPROVAL**

**Dr. Mohamed A. El-Sharkawy, Chair**

Department of Electrical and Computer Engineering

**Dr. Brian S. King**

Department of Electrical and Computer Engineering

**Dr. Maher E. Rizkalla**

Department of Electrical and Computer Engineering

**Approved by:**

Dr. Brian S. King



Dedicated to my parents Dr. Balasaheb Kalgaonkar and Mrs. Pratibha Kalgaonkar for  
their endless love, support and encouragement.

## ACKNOWLEDGMENTS

I feel like I have learnt a lot after having spent my graduate research career gaining a deep understanding and writing this thesis on deep neural networks for embedded systems. This is one of the great treasures of my life that I will cherish and exploit in my future academic as well as professional career.

I will like to begin by expressing my sincere gratitude and honoring the support and guidance of my thesis adviser, Dr. Mohamed El-Sharkawy. I could not have imagined having a better mentor and an advisor for my thesis research because of his benevolence, diligence and erudition. I will also like to thank Dr. Brian King and Dr. Maher Rizkalla for serving on my thesis committee. Likewise, I want to thank Ms. Sherrie Tucker who sincerely and selflessly helped me through every stage of my graduate studies.

Last but not least, I am highly indebted for the support and fortitude of my family, the most important single force in my life. It was my parent's unconditional love, encouragement and sacrifices that have helped me get through the sleepless nights of writing this thesis without which I could not have overcome the difficulties during these years.

## PREFACE

In this constantly emerging data-driven scientific society, x86 and ARM based embedded mobile computing platforms ranging from single-threaded single core to multi-threaded multi-core processors play a distinct role in the computer vision (OpenCV) paradigm. Convolutional Neural Network (CNN) algorithms developed specifically for computer vision applications need to be optimized for embedded computing platforms with limited computational resources such as memory and processing power.

This thesis focuses on optimizing CondenseNet, a successor of widely popular CNN: DenseNet (Densely Connected Convolutional Networks) and explains different techniques utilized to reduce forward FLOPs and redundant trainable parameters resulting in unprecedented computational efficiency during training from scratch as well as during real-time inference on an ARM based computing platform. This new CNN architecture is named CondenseNeXt. Furthermore, extensive analyses are conducted on three popular computer vision benchmarking datasets: CIFAR-10, CIFAR-100 and ImageNet in order to corroborate the performance of CondenseNeXt CNN.

# TABLE OF CONTENTS

LIST OF TABLES . . . . .	8
LIST OF FIGURES . . . . .	9
LIST OF SYMBOLS . . . . .	11
ABBREVIATIONS . . . . .	12
GLOSSARY . . . . .	15
ABSTRACT . . . . .	17
1 PROBLEM AND ITS BACKGROUND . . . . .	18
1.1 Brief Overview . . . . .	18
2 INTRODUCTION . . . . .	19
2.1 Artificial Neural Networks . . . . .	19
2.2 Multi-layer Feed-forward Neural Networks . . . . .	22
2.3 Convolutional Neural Networks . . . . .	25
2.4 Evolution of the ResNet CNN Family . . . . .	27
3 PROPOSED ULTRA-EFFICIENT CNN ARCHITECTURE: CONDENSENEXT . . . . .	30
3.1 Prior Works . . . . .	30
3.2 Convolution Layers . . . . .	31
3.3 Model Compression . . . . .	33
3.3.1 Group-wise Pruning . . . . .	33
3.3.2 Class-Balanced Focal Loss Function . . . . .	34
3.3.3 Cardinality . . . . .	34
3.4 Activation Function . . . . .	35
4 ARM-BASED COMPUTING PLATFORM: NXP BLUEBOX . . . . .	36
4.1 Brief Overview . . . . .	36
4.2 System Design of NXP BlueBox Gen2 Family . . . . .	37
4.2.1 S32V Computer Vision Processor . . . . .	37

4.2.2	LS2 High Compute Processor . . . . .	38
4.2.3	S32R Radar Processor . . . . .	38
4.3	System Connectivity of NXP BlueBox Gen2 Family . . . . .	42
4.4	RTMaps Remote Studio Software . . . . .	44
5	EXPERIMENTS AND RESULTS . . . . .	45
5.1	Cyberinfrastructure . . . . .	45
5.1.1	Training Infrastructure . . . . .	45
5.1.2	Testing Infrastructure . . . . .	46
5.2	Training and Testing Results for Image Classification . . . . .	46
5.2.1	CIFAR-10 Dataset . . . . .	46
5.2.2	CIFAR-100 Dataset . . . . .	51
5.2.3	ImageNet Dataset . . . . .	55
5.3	Summary . . . . .	59
6	CONCLUSION . . . . .	60
7	RECOMMENDATIONS . . . . .	61
	REFERENCES . . . . .	62
	PUBLICATIONS . . . . .	65

## LIST OF TABLES

2.1	Continuously Differentiable Nonlinear Activation Functions. . . . .	21
5.1	Comparison of Experiment 1 Performance on CIFAR-10 Dataset. . . . .	48
5.2	Comparison of Experiment 2 Performance on CIFAR-10 Dataset. . . . .	49
5.3	Comparison of Performance on CIFAR-100 Dataset. . . . .	53
5.4	Comparison of Performance on ImageNet Dataset. . . . .	57
5.5	Summary of CondenseNeXt’s Performance in Terms of Error Rate. . . . .	59
5.6	Summary of CondenseNeXt’s Performance in Terms of Accuracy. . . . .	59
5.7	Image Classification Evaluation Time of CondenseNeXt on NXP BlueBox 2.0 .	59

## LIST OF FIGURES

2.1	Nonlinear model of a neuron, labeled $k$ , in a neural network. . . . .	20
2.2	Fully connected feed-forward neural network with one hidden layer and one output layer. . . . .	22
2.3	Signal-flow of back-propagation learning algorithm for fully connected feed-forward neural network with one hidden and one output layer. . . . .	23
2.4	High level view of a convolutional neural network for image processing. . . .	25
2.5	2D convolutional operator where the kernel matrix is moved across the target image and element-wise products are recorded. . . . .	26
2.6	Identity shortcut connections in ResNet. . . . .	27
2.7	Visual representation of dense connectivity in DenseNet. . . . .	28
2.8	CondenseNet's group convolution on the right that replaces DenseNet's standard convolution on the left. . . . .	29
3.1	3D illustration of the overall process of depthwise separable convolution. . .	31
3.2	Weight matrices for different network pruning approaches. . . . .	33
3.3	Graphical representation demonstrating difference between ReLU and ReLU6 activation functions. . . . .	35
4.1	Block diagram of S32V computer vision processor. . . . .	39
4.2	Block diagram of LS2 high compute processor. . . . .	40
4.3	Block diagram of S32R radar processor. . . . .	41
4.4	NXP Bluebox 2.0. . . . .	42
4.5	High-level view of NXP BlueBox Gen2 system connectivity. . . . .	43
4.6	RTMaps on NXP BlueBox 2.0. . . . .	44
5.1	CIFAR-10 classes for image classification. . . . .	47
5.2	Training overview of CondenseNeXt on CIFAR-10 dataset. . . . .	48
5.3	Evaluation of CondenseNeXt on CIFAR-10 dataset when deployed on NXP BlueBox 2.0 using RTMaps Remote Studio 4.8.0 for classifying an image of an airplane and outputting the predicted class in RTMaps console. . . . .	50
5.4	CIFAR-100 classes for image classification. . . . .	51
5.5	Training overview of CondenseNeXt on CIFAR-100 dataset. . . . .	52

5.6	Evaluation of CondenseNeXt on CIFAR-100 dataset when deployed on NXP BlueBox 2.0 using RTMaps Remote Studio 4.8.0 for classifying an image of a human and outputting the predicted class in RTMaps console. . . . .	54
5.7	ImageNet classes for image classification. . . . .	55
5.8	Training overview of CondenseNeXt on ImageNet dataset. . . . .	56
5.9	Evaluation of CondenseNeXt on ImageNet dataset when deployed on NXP BlueBox 2.0 using RTMaps Remote Studio 4.8.0 for classifying an image of a street intersection sign and outputting the predicted class in RTMaps console. . . . .	58



## LIST OF SYMBOLS

$\delta$	delta for local gradient
$\gamma$	gamma for focal loss/imbalance
$\varphi$	phi for activation function
$\sigma$	sigmoid for sigmoid cross-entropy loss
$\Sigma$	summation for summing junction

## ABBREVIATIONS

AEC	Automotive Electronics Council
AHPC	Automotive High Performance Compute
AI	Artificial Intelligence
ANN	Artificial Neural Network
ARM	Advanced RISC Machines
ASIC	Application-Specific Integrated Circuit
ASIL-B/C	Automotive Safety Integrity Level B or C
ASIL-D	Automotive Safety Integrity Level D
CAN	Controller Area Network
CAN-FD	Controller Area Network Flexible Data-Rate
CCWC	Computing and Communication Workshop and Conference
CIFAR	Canadian Institute for Advanced Research
CISC	Complex Instruction Set Computer
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CVPR	Computer Vision and Pattern Recognition
DDR4	Double Data Rate 4
DNN	Deep Neural Network
DRAM	Dynamic Random-Access Memory
DUART	Dual Universal Asynchronous Receiver-Transmitter
ECC	Error Correction Code
eSDHC	Enhanced Secure Digital Host Controller
FAIR	Facebook Artificial Intelligence Research
FLOPs	Floating Point Operations
GPIO	General-Purpose Input/output
GPU	Graphics Processing Unit

GUI	Graphical User Interface
IEEE	Institute of Electrical and Electronics Engineers
IFC	Integrated Flash Controller
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IoT	Internet of Things
ISA	Instruction Set Architecture
ISP	Image Signal Processor
JTAG	Joint Test Action Group standards
LIDAR	Light Detection and Ranging
LMS	Least Mean Square
LPDDR3	Low-Power Double Data Rate 3
LSVRC	Large Scale Visual Recognition Challenge
MIPI-CSI2	Mobile Industry Processor Interface - Camera Serial Interface 2
ML	Machine Learning
NAND	NOT-AND gate
NOR	NOT-OR gate
OpenCV	Open Source Computer Vision library
PCIe	Peripheral Component Interconnect Express
QSPI	Quad Serial Peripheral Interface
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RGB	Red Green Blue
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
RTMaps	Real-Time Multi-sensor applications
SAR-ADC	Successive Approximation Analog-to-Digital Converter
SATA	Serial Advanced Technology Attachment
SD	Secure Digital
SGD	Stochastic Gradient Descent

SPI	Serial Peripheral Interface
SPIO	Serial Peripheral Input Output
SRAM	Static Random Access Memory
SSD	Solid State Drive
UART	Universal Asynchronous Receiver-Transmitter
UAV	Unmanned Ariel Vehicle
USB	Universal Serial Bus
YOLO	You Look Only Once

## GLOSSARY

architecture	set of rules and methods that dictate the functionality and organization of different modules of a computer system
bias	an intercept that is added externally which has an effect of increasing or decreasing the net input of the activation function
checker core	a safety core within a CPU chip that is used to perform comparisons for error checking
cyberinfrastructure	advanced and powerful computer and information technology systems
dennard scaling	power density stays constant as transistor size reduce and power utilization remains in proportion
distributed computing	multiple components located on different machines
edge devices	local devices
error rate	signifies how well the network is performing on a certain set
flash	a non-volatile computer memory that can be erased or re-programmed
FlexCAN	an embedded network architecture design to extend CAN communication
FlexRay	more faster and reliable automotive communication protocol than CAN communication
frobenius inner product	an operation that computes two matrices and returns a number
ground truth	measurement of the target variable (inference) against empirical evidence
hidden layer	a part of the neural network is not visible directly from either the input or output nodes of the network
inference	process of utilizing a trained neural network model to make a prediction
interleaving support	a design which compensates for the slow memory speed by evenly spreading memory address across memory banks

L2 cache memory	a level 2 cache memory built into the CPU chip
message buffering	an area in the memory designed to temporarily hold messages when sent by the source until the recipient receives it
mobile	capable of moving or being moved
moore's law	processor's computing power doubles every 18 months
neuron	structural elements of a processing system that is capable of processing and storing information
perceptrons	a neural network algorithm for supervised learning
plasticity	phenomenon of neurons adapting to the constantly changing environment
power wall	a phenomenon that limits frequency of processors to a certain limit
pruning	to remove as superfluous
single crop of inputs	input images are taken and cropped only once
synaptic weights	extracted information that is stored using inter-neuron connection strengths
tensor	another word of neuron
top-1 accuracy	top predicted class with highest accuracy that match the ground truth
top-5 accuracy	top five predicted classes with the highest accuracy that match the ground truth
weight	a parameter within a neural network that determines how much influence the input will have on the output
zipwire	a communication bus developed upon a high speed (240 MHz) serial interface

# ABSTRACT

Artificial Intelligence (AI) is the intellectuality demonstrated by machines, similar to natural intelligence demonstrated by living creatures of the Animalia kingdom, which involves emotionality and consciousness to a certain extent. Advances in the computer technology and access to copious amounts of data for multinomial classification due to digitization of the human society, inexpensive cameras and Internet of Things (IoT) has fueled research and development in the field of machine learning and perception, also known as computer vision.

Convolutional Neural Networks (CNN) are a class of Deep Neural Networks (DNN) which is a subset of Machine Learning (ML) which in turn is a simple technique for the realization of AI. CNNs are becoming more popular in the field of computer vision for performing fundamental tasks such as image classification, object detection and image segmentation for real-world applications including, but not limited to, self-driving vehicles, robotics and Unmanned Ariel Vehicles (UAVs) commonly known as a drone. This rise in popularity of AI along with advancement in edge devices at local level such as mobile embedded computing platforms ranging from single-core single-threaded processors to multi-core multi-threaded processors have dictated the need for research and development of increasingly efficient state-of-the-art deep neural network architectures.

One of the very first data and computationally intensive convolutional neural network algorithms began dominating accuracy benchmarks in 2012 [1]. With recent developments in AI and embedded systems, a desire for more efficient yet accurate inferring CNNs has flourished in recent years. This thesis proposes a neoteric variant of deep convolutional neural network architecture incorporating state-of-the-art techniques such as Depthwise Separable Convolution and Model Compression (Pruning) which results in reduction of forward FLOPs and increase in overall accuracy (decrease in error rate) resulting in an outstanding performance during training from scratch as well as during real-time image classification observed through deployment of trained weights on NXP's BlueBox, an ARM-based autonomous embedded computing platform designed for self-driving vehicles, and benchmarked across three popular computer vision datasets: CIFAR-10, CIFAR-100 and ImageNet.

# 1. PROBLEM AND ITS BACKGROUND

Before beginning this thesis, a brief discussion of the problem being investigated, and its background is in order. This discussion, while inadequate, still resides in the realm beyond being what exposes the limitations of human understanding, shall lay a strong foundation to commend the thesis statement proffered by this work and for the merits upon which the research work and results in the subsequent chapters shall manifest.

## 1.1 Brief Overview

In recent years, three significant advancements have changed the AI landscape: *technological advancements* such as dramatically slowing down of Moore’s Law which forecasts that the processor core performance will now double every 20 years, and the ending of Dennard Scaling that has resulted in a power wall restricting the processor frequency to around 4 GHz since 2006 [2]; *shift in computing paradigms* resulting in transitioning from centralized computing to distributed computing on edge devices; and *significant developments in the AI domains* such as computer vision, natural language processing, and graph analytics. As a result, researchers are faced with serious challenges of developing efficient deep neural network algorithms for devices with constrained computational resources such as memory, power supply, performance and system cooling.

This thesis proposes an ultra-efficient deep convolutional neural network developed specifically for x86 and ARM based embedded computing platforms on the edge without needing a CUDA enabled GPU support for real-time inference of input data, to confidently propound the thesis statement offered by this work in the subsequent chapters.



## 2. INTRODUCTION

Convolutional Neural Networks (CNN) are the backbone of computer vision algorithms. This section provides an in-depth analysis of what a CNN is, starting with fundamentals of neural networks and gradually building up to the different modules that constitute a CNN.

### 2.1 Artificial Neural Networks

Artificial Neural Networks (ANN), commonly known as Neural Networks, have been heavily inspired by the functioning of a human brain and how the brain computes completely different than a traditional computer. A human brain is extremely *sophisticated* and *nonlinear* information processing system. It is capable of reorganizing and retraining its structural elements known as *neurons* in order to parallelly perform complex computations such as multi-object detection and classification, and pattern recognition much faster than any computer in existence today. For example, consider human vision which performs perceptual recognition computation of patterns and objects around us in about 100-200 milliseconds and provides us the information so that we can interact and react accordingly. Powerful computers with abundant computational resources will still take longer to execute tasks of such higher complexity.

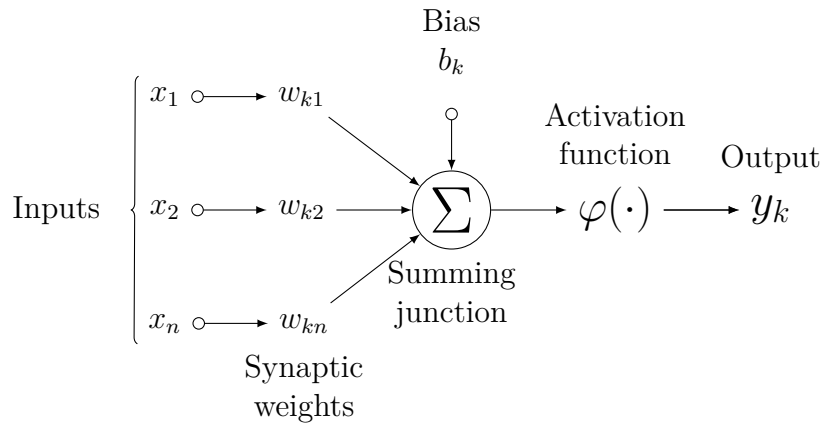
During the first two years of birth, a human brain undergoes considerable re-structuring of its neurons to develop its ability to learn and adapt to set of rules of behavior through what is commonly known as *experience*. However, this development does not stop here; it continues throughout the human life. This phenomenon of neurons adapting to the constantly changing environment is known as *plasticity*. Just as plasticity is important to human brain's information extraction process, so is important for neural networks comprised of artificial neurons. In general, a neural network is modelled to allow machines to learn, adapt, think and process information in a similar way the human brain does. Neural networks are developed using a programming language and deployed on electrical computing hardware or simulated in a software on a digital computer.

Artificial neural networks resemble a human brain in following two ways:

- Through a learning process, knowledge is extracted with the help of neurons.
- This information is then stored using inter-neuron connection strengths known as synaptic weights.

This process of learning is called a learning algorithm where the synaptic weights of the neural network are modified to attain a desired output. A neuron is an information processing unit which is an underlying fundamental of a learning algorithm. Figure 2.1 demonstrates the structure of a neuron in a neural network and discloses three basic elements as follows:

1. **Synaptic Weights:** Each of these synapses are represented by its own weights. Unlike synapses in a human brain, synapses in a neural network may have positive as well as negative values.
2. **Summing Junction:** Also known as an adder, it sums all input signals weighted by the corresponding synaptic strengths of a neuron. In this way, an adder functions as a linear combiner.
3. **Activation Function:** An activation function, either linear or non-linear, is used to limit the amplitude of the output of a neuron. Activation functions are also sometimes known as squashing functions.



**Figure 2.1.** Nonlinear model of a neuron, labeled  $k$ , in a neural network.

Computation of the local gradient  $\delta(n)$  of a neuron requires knowledge of an activation function  $\varphi(\cdot)$  as seen in Figure 2.1. An activation function  $\varphi(\cdot)$  has to be continuously differentiable. Table 2.1 provides a summary of most commonly used continuously differentiable nonlinear activation functions in fully connected multi-layer feed-forward neural networks.

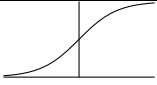
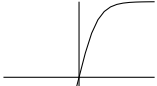

Bias  $b_k$  is like an intercept that is added externally which has an effect of increasing or decreasing the net input of the activation function. In simple terms, it facilitates the shifting of the activation function by adding a constant. Figure 2.1 can be mathematically represented by (2.1) as follows:

$$y_k = \varphi(u_k + b_k) \quad (2.1)$$

where

$$u_k = \sum_{j=1}^m w_{kj}x_j \quad (2.2)$$

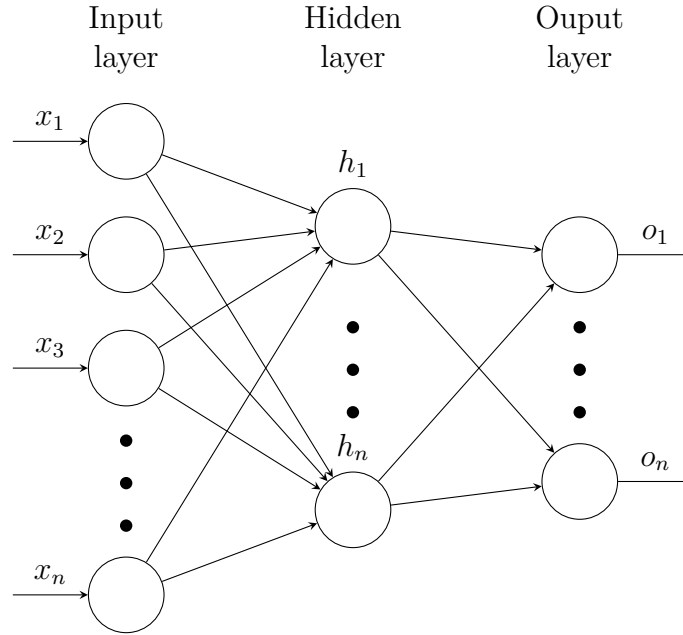
**Table 2.1.** Continuously Differentiable Nonlinear Activation Functions.

Name	Function	Derivative	Figure
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))^2$	
Tanh	$\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$	
ReLU	$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0. \end{cases}$	
Softmax	$f(x) = \frac{e^x}{\sum_i e^x}$	$f'(x) = \frac{e^x}{\sum_i e^x} - \frac{(e^x)^2}{(\sum_i e^x)^2}$	

## 2.2 Multi-layer Feed-forward Neural Networks

Multi-layer feed-forward neural networks are one the most popular structured classes of neural network architectures. In this type of network, neurons are intimately linked with the learning algorithm utilized to train the network for a given dataset depending on its end application and implementation.

There can be one or more hidden layers whose computation nodes are known as hidden neurons and the number of hidden layers dictate the depth of a neural network. The term *hidden* denotes that a part of the neural network is not visible directly from either the input or output nodes of the network. By adding more hidden layers, the neural network will have extra sets of synaptic connections which will help the network extract higher order statistical data from the inputs  $x_n$ . Figure 2.2 below provides a visual representation of a two-layer feed-forward neural network comprised of one input layer, one hidden layer and one output layer as follows:

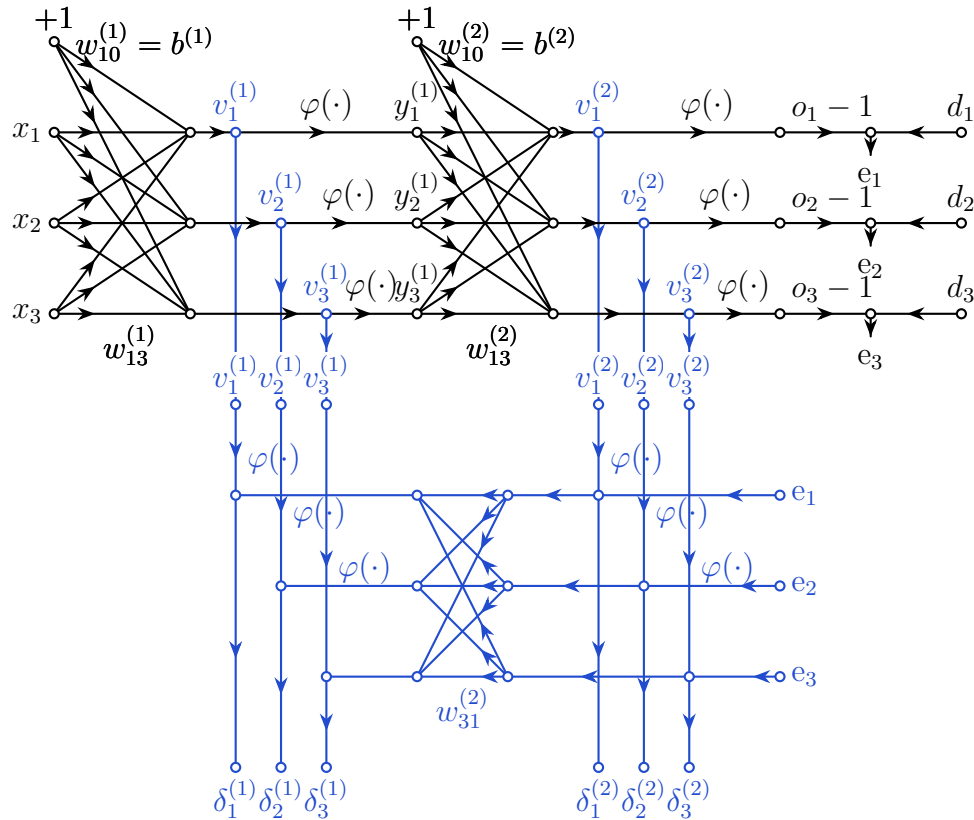


**Figure 2.2.** Fully connected feed-forward neural network with one hidden layer and one output layer.

A conventional method for training multi-layer fully connected feed-forward neural networks, also known as multi-layer perceptrons, is the back-propagation algorithm, which also includes the famous LMS algorithm as a special case. There are two phases in this algorithm:

1. Forward Phase: In this phase, inputs are propagated forward through the neural network layer by layer where synaptic weights are fixed until it reaches the output layer.
2. Backward Phase: In this phase, an error signal is generated by correlating the output of the network with the ground truth (desired response). Then this error signal is propagated backward through the neural network layer by layer adjusting the synaptic weights of the network.

Blue lines in Figure 2.3 below correspond to error signals of the back propagation algorithm.



**Figure 2.3.** Signal-flow of back-propagation learning algorithm for fully connected feed-forward neural network with one hidden and one output layer.

In order to mathematically represent the back propagation algorithm for multi-layer perceptrons [3], we will consider following two cases:

**Case 1** - When neuron  $j$  is an output node, a ground truth (desired output) is readily available for the node. Hence, error signal  $e_j(n)$  and local gradient  $\delta_j(n)$  is mathematically represented as follows:

$$e_j(n) = d_j(n) - y_j(n) \quad (2.3)$$

where  $d_j(n)$  is the  $i^{th}$  element of the desired response vector  $d(n)$  and  $y_j(n)$  is signal generated at the output of neuron  $j$  in the output layer with the help of stimulus  $x(n)$  applied to the input layer. Then, the local gradient  $\delta_j(n)$  can be computed as follows:

$$\delta_j(n) = e_j(n) \varphi_j'(v_j(n)) \quad (2.4)$$

where  $\varphi_j'(v_j(n))$  derivative corresponds to the respective activation function.

**Case 2** - When neuron  $j$  is a hidden node, there is no ground truth (desired output) available at the hidden node. Hence, an error signal has to be calculated recursively in a backward fashion. Equation 2.4 can be redefined as follows:

$$\delta_j(n) = -\frac{\delta \varepsilon(n)}{\delta y_j(n)} \varphi_j'(v_j(n)) \quad (2.5)$$

where

$$\begin{aligned} \frac{\delta \varepsilon(n)}{\delta y_j(n)} &= -\sum_k e_k(n) \varphi_k'(v_k(n)) w_{kj}(n) \\ &= -\sum_k \delta_k(n) w_{kj}(n) \end{aligned} \quad (2.6)$$

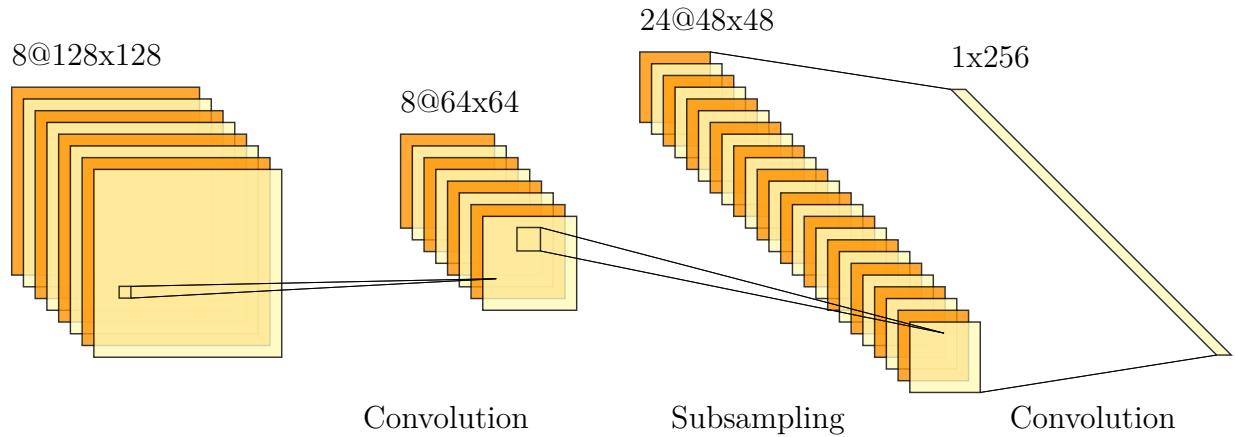
From equations 2.5 and 2.6, the back propagation formula for local gradient  $\delta_j(n)$  for this case is mathematically represented as follows:

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \quad (2.7)$$

## 2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a special class of fully connected multi-layer feed-forward neural networks designed specifically to infer two dimensional objects in an image frame with a high degree of resistance and stability to skewing, scaling, noise and other forms of distortions. This complex task is trained in a supervised manner by providing ground truth labels during the training stage. Following are three fundamental operations carried out by a CNN:

1. Feature extraction: Each neuron in the neural network obtains a synaptic input from local neuron(s) from the previous layer. Thus, compelling the current neuron to extract local features. After feature extraction is performed, its location becomes less important, provided if the relative position to other features is maintained.
2. Feature mapping: A layer is composed of multiple feature maps where individual neurons are required to share coequal set of synaptic weights. This operation aids in shift invariance as well as reduction in the number of free parameters.
3. Subsampling: This operation results in reduction of resolution of the feature map which helps in building resistance to shifts and other forms of distortions as discussed before.

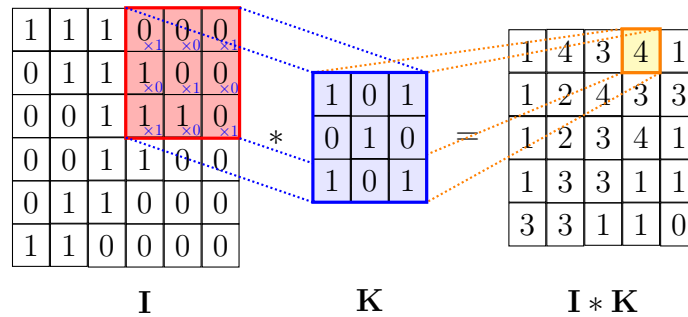


**Figure 2.4.** High level view of a convolutional neural network for image processing.

Figure 2.4 provides visualization of a convolutional neural network's architecture which comprises of one input layer, two hidden layers and one output layer. This network architecture is designed to perform image processing tasks such as image classification. The input layer has  $128 \times 128$  sensory nodes (neurons) which receives input image that has already been cropped, centered and normalized in terms of dimensions. Thereafter, convolution, subsampling and then again convolutional operations are performed on the input image as follows:

1. Convolution is performed by the first hidden layer made up of eight feature maps consisting of  $64 \times 64$  neurons.
2. Subsampling operation is carried by the second hidden layer along with local averaging. This layer is made up of 24 feature maps consisting of  $48 \times 48$  neurons. Each neuron also has a trainable coefficient, bias and an activation functions (can be either linear or non-linear).
3. A final convolution is performed by the output layer made up of 1 feature map consisting of 256 neurons wherein, each neuron is assigned either one of 256 possible outputs.

Convolution layer contains an array of numbers that will execute Frobenius Inner Product operation on data received from previous layers and pass it forward to the next layer. Figure 2.5 provides a visualization of Frobenius Inner Product of Filter K on data I, which is repeated for every filter in the convolutional layer.



**Figure 2.5.** 2D convolutional operator where the kernel matrix is moved across the target image and element-wise products are recorded.

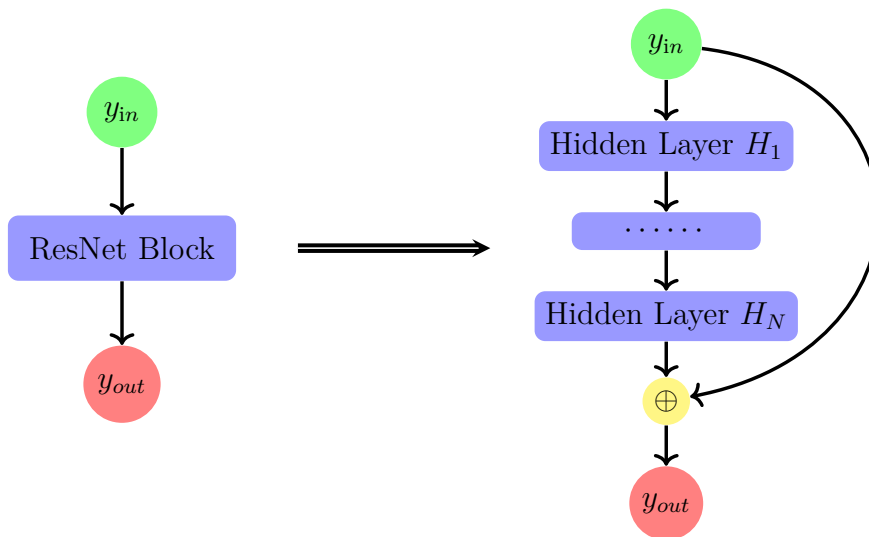


## 2.4 Evolution of the ResNet CNN Family

A first fully connected deep feed-forward multi-layer neural network for supervised learning was introduced by Alexey G. Ivakhnenko and V.G. Lapa in 1967 [4] which provides the underlying principles for all deep neural network architectures today. Computer vision is a multi-disciplinary field of science where researchers seek to study and develop innovative techniques to allow computers to perform intricate operations such as image classification, object detection and image segmentation, similar to how a human brain computes such tasks.

AlexNet, a deep CNN introduced by Alex Krizhevsky in collaboration with Ilya Sutskever and Geoffrey Hinton in 2012 [5], is one of the world's most influential works in the field of AI having won the ImageNet LSVRC (Large Scale Visual Recognition Challenge) competition in September 2012. It inspired many AI related works utilizing CNNs and GPUs to proliferate deep learning.

In 2016, G. Huang *et al.* introduced ResNet architecture to train a CNN with large amounts of data for computer vision purposes. A prominent feature of this CNN is the identity shortcut connections that allow skipping of one or more layers [6], visually represented by Figure 2.6 below. Due to this novel idea of skipping layers, ResNet is still being used as a baseline algorithm by many researchers to build more complex DNN architectures.

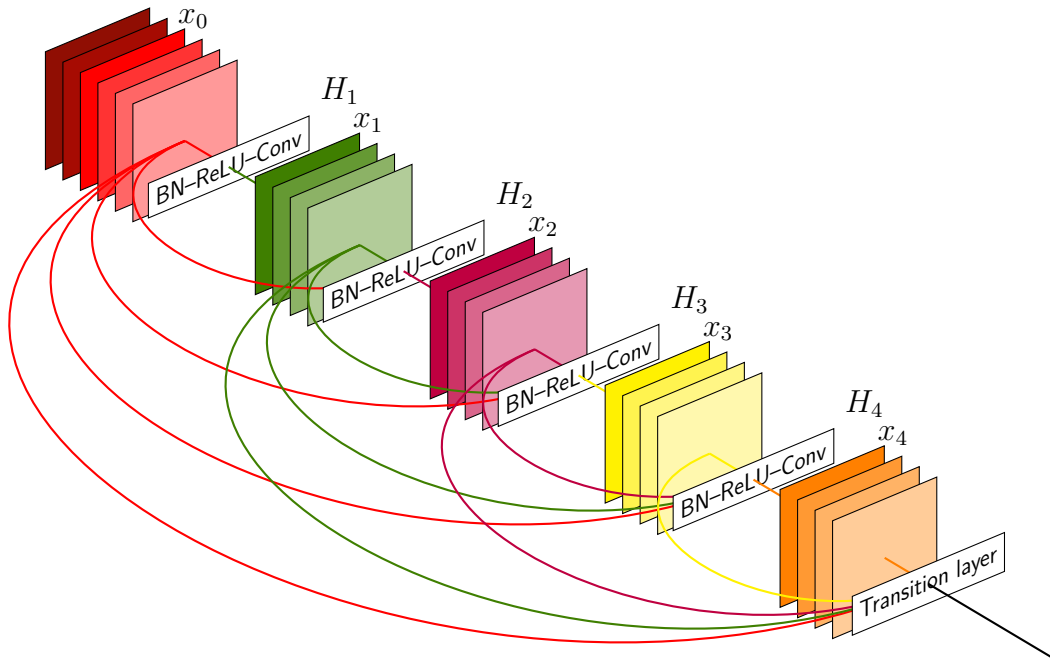


**Figure 2.6.** Identity shortcut connections in ResNet.

In 2017, G. Huang *et al.* introduced Dense Convolutional Network, popularly known as DenseNet [7]. This work was recipient of the Best Paper Award in the 2017 IEEE CVPR conference, which proposes a novel idea of dense connections which results in fewer trainable parameters and an increase in accuracy compared to ResNet.

At the core of DenseNet lies the concept of dense blocks, a group of layers, and dense connectivity, where each layer is connected to every other subsequent layer in a feed-forward fashion as shown in Figure 2.7. In other words, a proceeding layer carry-forwards its feature maps to all subsequent layers and then concatenates it so that each subsequent layer receives a collective information from previous layers. This facilitates the reuse of features extracted by layers in the early stages and are directly utilized by deeper layers.

Experiments in [7] show that in the final classification layer, weights tend to focus towards final feature maps of the network. This helps in moderating the vanishing-gradient problem, encouraging feature reuse, strengthening feature propagation, and significantly reducing the number of trainable parameters.



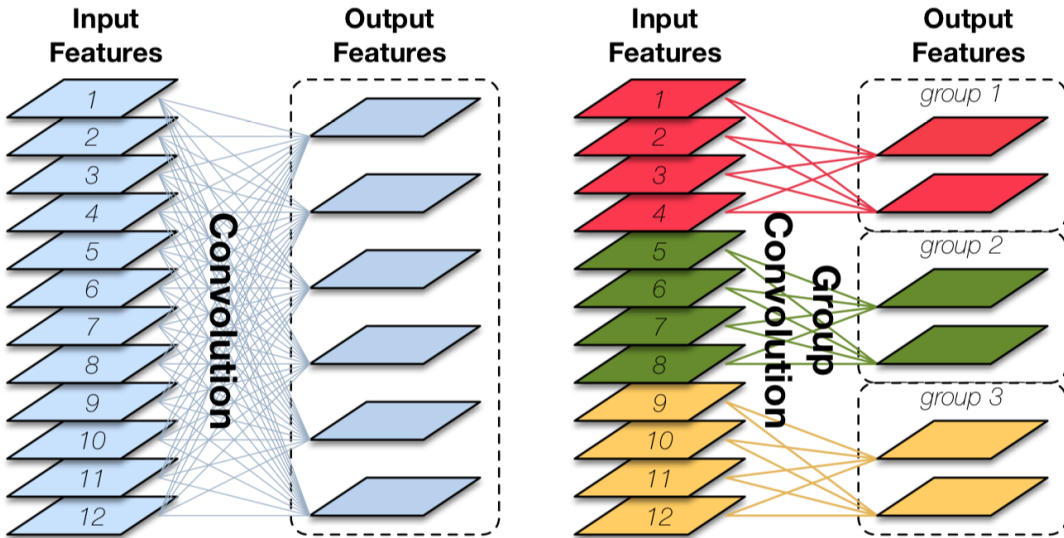
**Figure 2.7.** Visual representation of dense connectivity in DenseNet.

In 2018, G. Huang *et al.* introduced an improved version of DenseNet called CondenseNet [8]. The authors propose replacing Standard Convolutions in DenseNet with Group Convolutions, a special case of a sparsely connected convolutions, first seen in AlexNet [5] in 2012 and then again in ResNeXt [9] in 2017, results in reduction in computational cost by partitioning input features in  $G$  number of groups which are mutually exclusive and produces its own outputs, visually represented by Figure 2.8.

The reduction in computational cost can be mathematically represented by the following equation where  $C_C$  is computational cost,  $R$  is number of input features,  $O$  is number of output features and  $G$  is number of groups:

$$C_C = \frac{R \times O}{G} \quad (2.8)$$

The authors also propose incorporating filter pruning method into the design of group convolution and collectively call this module as learned group convolution. With this pruning technique, filters with low magnitude weights are pruned first and then optimized to learn the defined filter groups. Their experiments demonstrate a substantial savings in computational resources required to train the network from scratch, compared to DenseNet.



**Figure 2.8.** CondenseNet's group convolution on the right that replaces DenseNet's standard convolution on the left.

### 3. PROPOSED ULTRA-EFFICIENT CNN ARCHITECTURE: CONDENSENEXT

CondenseNeXt is an ultra-efficient and an improved variant of CondenseNet specifically designed for ARM-based computing platforms with constrained computational resources such as RAM and without CUDA enabled GPU support. CondenseNeXt refers to the *next* dimension of cardinality [10]. This work was recipient of the Best Paper Award in the category of Artificial Intelligence and Machine Learning at the 2021 IEEE CCWC conference.

#### 3.1 Prior Works

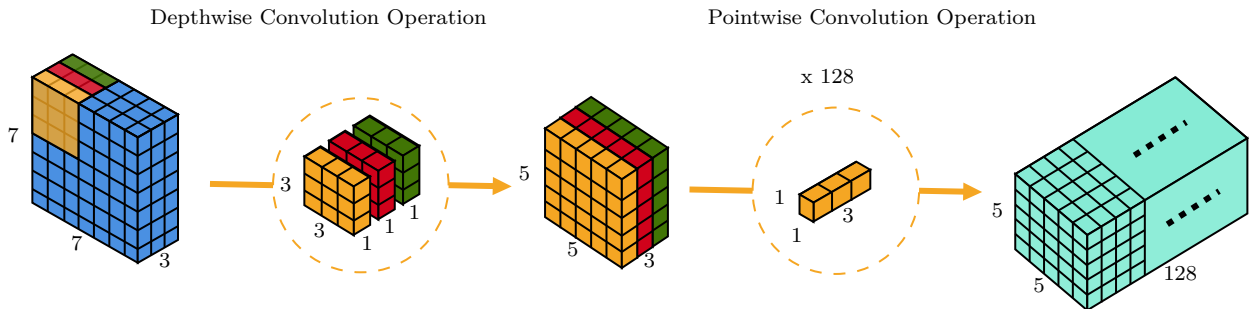
Following prior and related works have contributed to the research work and results presented within this thesis:

- Convolutional Neural Networks [11]–[13], particularly CondenseNet [8], provide an underlying foundation for the CNN algorithm proposed by this thesis.
- Dense blocks and dense connectivity first seen in DenseNet [7], where previous and current layers are connected to subsequent layers in a feed-forward fashion facilitating the re-use of channels and neurons to characterize information at different levels of deepness, has been incorporated into the design of the proposed CNN algorithm.
- Depthwise separable convolutions, first seen in [14], is one of the main fundamental and key concepts incorporated into the design of CondenseNeXt CNN. This idea of utilizing spatially separable convolutions can be dated back to 2012 when it was introduced in [15] has been successfully implemented in MobileNet [16] and Xception [17] CNN architectures.
- Group-wise filter pruning [18]–[20], one of the very popular model compression techniques, has also been incorporated into the design of CondenseNeXt CNN architecture to significantly improve efficiency by discarding redundant elements that do not affect the overall performance of CNN algorithm proposed by this thesis.

The two main goals of the proposed CNN algorithm, CondenseNeXt, is to reduce the final model size (weights) trained on CIFAR-10 dataset to less than 3.0 MB at the cost of loss in accuracy and to reduce the computational resources required to train the network from scratch as well as for real-time inference on low compute/embedded platforms such NXP BlueBox, Nvidia Jetson, Raspberry Pi in comparison to the baseline architecture, CondenseNet. Following sections provide a detailed information on the prominent state-of-the-art features of CondeseNeXt.

### 3.2 Convolution Layers

Depthwise separable convolution primarily deals with spatial dimensions i.e. the width and height as well as depth of an image and a kernel. *Depth* corresponds to the number of channels in an image. For example, depth will be three for RGB (Red, Green, Blue) channels of an input image. Depthwise separable convolution takes an input image and transforms it only once and then elongates this transformed image to the desired  $n$  number of channels as shown in Figure 2.9 below. Hence, after depthwise separable convolution, an input image, originally with three channels, will now have multiple channels, each representing a different shade of the original channel. For example, the original input image's red channel is transformed and then elongated along 128 channels. We will then have an *interpretation* of the density of the red color defined by each of the 128 levels of redness. This applies to the remaining two channels of the RGB input image as well. Depthwise separable convolution replaces group convolution from CondenseNet.



**Figure 3.1.** 3D illustration of the overall process of depthwise separable convolution.

Depthwise separable convolution incorporates following two layers:

- **Depthwise Convolution:** Instead of performing convolution operation on all channels of an input image parallelly, depthwise convolution layer performs convolution operation on each channel independently. This behavior can be associated to the act of *filtering* in layman's terms. Suppose there is an input image of size  $X \times X \times N$  and kernels (filters)  $K$  of size  $F \times F \times 1$ , where  $N$  is the number of channels of an input image, then the output of depthwise convolution will be of size  $Y \times Y \times N$ . The spatial dimensions have now shrunk while depth of  $N$  channels have been preserved (remains unchanged). The computational cost of this operation is  $Y^2 \times F^2 \times N$ . This operation can be mathematically represented as follows:

$$\hat{Y}_{k,l,m} = \sum_{i,j} \hat{K}_{i,j,m} \cdot X_{k+i-1,l+j-1,m} \quad (3.1)$$

- **Pointwise Convolution:** Since depth of the input image has remained unchanged, a pointwise convolution is performed to increase the number of channels by using a  $1 \times 1$  kernel. Hence, size of kernel for this operation will be  $1 \times 1 \times N$  and size of the output will be  $Y \times Y \times Z$  for  $Z$  kernels  $K$ . This behavior can be associated to the act of *combining* in layman's terms. This operation can be mathematically represented as follows:

$$Y_{k,l,n} = \sum_m \tilde{K}_{m,n} \cdot \hat{Y}_{k-1,l-1,m} \quad (3.2)$$

To put this in to perspective and to provide a valid comparison for depthwise separable convolution, equations 3.1 and 3.2 can be mathematically represented by the following equation for a standard convolution:

$$Y_{k,l,n} = \sum_{i,j,m} k_{i,j,m,n} \cdot X_{k+i-1,l+j-1,m} \quad (3.3)$$

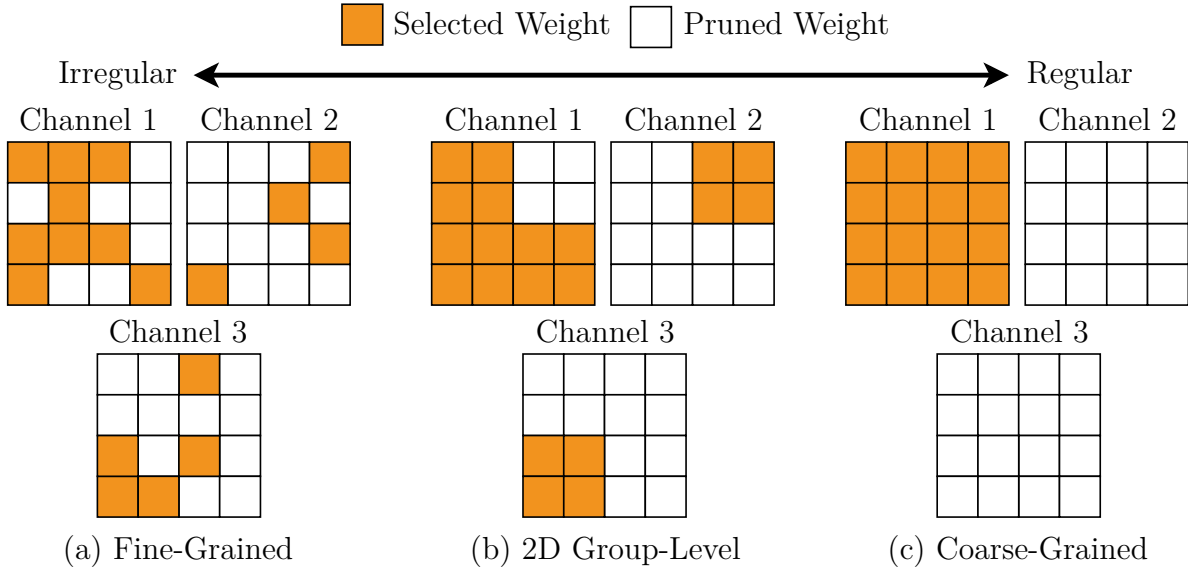
Incorporating depthwise separable convolution that splits a kernel into two discrete kernels for filtering and combining stages, as aforementioned, results in a substantial decrease in forward FLOPs resulting in an outstanding computational efficiency.

### 3.3 Model Compression

Network pruning is one of the propitious techniques in the field of model compression aimed at reducing computational costs such as FLOPs and RAM utilization by removing redundant and insignificant elements that either are irrelevant or will not affect the performance of the network upon disposition. Figure 3.2 below provides a brief overview of weight matrices under different levels of coarseness of network pruning approaches. Fine-grained pruning discards weights that have minimal influence on accuracy resulting in irregular structures whereas coarse-grained pruning discards entire filter that have minimal influence on accuracy resulting in regular structures but may result in significant accuracy loss if the network is highly compressed. In order to maintain a balance between both trade-offs, CondenseNeXt utilizes 2D group-level pruning approach in addition to class-balanced focal loss function and cardinality to ease and reduce the negative impact of this pruning process.

#### 3.3.1 Group-wise Pruning

Group-wise pruning is intended to exorcise inconsequential filters during the training process which is arbitrated by the  $L_1$ -Normalization [21] of  $X^{g_{ij}}$  where  $g$  denotes an individual group,  $i$  denotes the input and  $j$  denotes the output of that group.



**Figure 3.2.** Weight matrices for different network pruning approaches.

To facilitate the group-wise pruning process, a hyper-parameter  $p$  is defined and set to 4 which allows the network to autonomously determine the required number of filters (kernels) to remove before proceeding to perform depthwise separable convolution.

### 3.3.2 Class-Balanced Focal Loss Function

To solve the issues caused due to imbalanced weights, a weighting factor inversely proportional to the number of samples is incorporated into the design of CondenseNeXt. This technique is known as Class-Balanced Focal Loss Function (CBFLF) [22]. It can be mathematically represented by the following equation:

$$\text{CBFLF}(\mathbf{z}, y) = - \sum_{i=1}^C (1 - p_i^t)^\gamma \log(p_i^t) \quad (3.4)$$

where  $y$  is the ground-truth class,  $\sigma$  is the Sigmoid Cross-Entropy Loss and

$$p_i^t = \sigma(z_i^t) = \frac{1}{1 + \exp(-z_i^t)}$$

### 3.3.3 Cardinality

A new dimension to the existing spatial dimensions called Cardinality, denoted by  $D$ , is incorporated into the design of CondenseNeXt algorithm to further mitigate the loss in accuracy during the pruning process. Experiments and results presented within this thesis prove that instead of going deeper and/or wider in the network in order to obtain better overall accuracy when performance returns starts diminishing, increasing cardinality rate  $D$  is a more effective way of achieving this goal.

Consider a group convolution comprised of  $G$  groups of size  $F \times F \times G_X \times G_Y$  where  $G_X = \frac{X}{G}$  and  $G_Y = \frac{Y}{G}$ . The total number of trivial filters (kernels) to be pruned before proceeding to perform depthwise separable convolution can be mathematically represented as follows:

$$G \cdot G_x = X \cdot D - p \cdot X \quad (3.5)$$

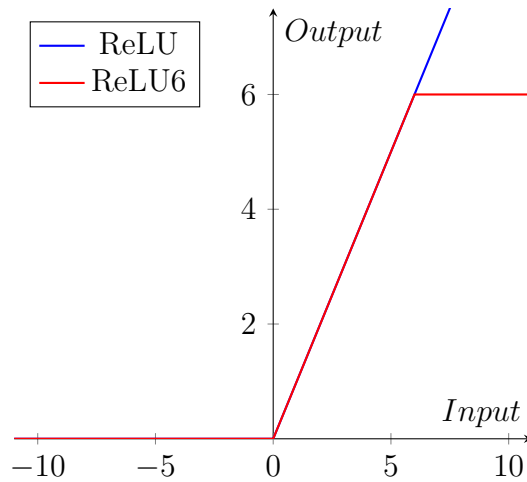


### 3.4 Activation Function

In artificial neural networks, an activation function of a node is a mathematical equation that defines the output of that particular node by limiting the amplitude of the output. Activation functions are also commonly known as *Transfer Functions*. Most state-of-the-art CNNs make use of non-linear activation functions to help the network adapt to various of types of datasets, to help learn complex patterns of the input data and to help differentiate between the output classes without requiring exorbitantly large number of neurons.

CondenseNeXt utilizes ReLU $n$  (Rectified Linear Units capped at  $n$  units) first seen in [23] to aid in learning of complex patterns of the input data with fewer neurons and with greater accuracy. In this non-linear activation function, units are capped at pre-determined  $n$  units to accelerate and encourage the network to learn sparse features much earlier compared to regular ReLU activation function. In CondenseNeXt, ReLU $n$  is capped at 6 and hence, called ReLU6. Furthermore, ReLU6 along with batch normalization is exercised before each convolution layer. ReLU6 can be mathematically defined by equation 3.6 and graphically represented by Figure 3.3 as follows:

$$f(x) = \min(\max(0, x), 6) \quad (3.6)$$



**Figure 3.3.** Graphical representation demonstrating difference between ReLU and ReLU6 activation functions.

## 4. ARM-BASED COMPUTING PLATFORM: NXP BLUEBOX

This thesis demonstrates an efficient deployment of CondenseNeXt on NXP BlueBox, an ARM-based autonomous embedded computing platform with constrained computational resources designed for automotive applications such as self-driving vehicles.

### 4.1 Brief Overview

Processors based on ARM architecture are very popular and widely used in mobile devices such as smartphones and tablets as well as in embedded computing systems such as the NXP BlueBox, Nvidia Jetson and Raspberry Pi for computer vision purposes. ARM processors are based on Reduced Instruction Set Computing (RISC) philosophy which is a widely successful Instruction Set Architecture (ISA) due to its low power consumption and heat generation compared to Complex Instruction Set Computing (CISC) architecture based processors such as the Intel x86-family.

Evolution of ARM processors can be dated back to 1981 when Acorn Computers introduced BBC Micro (British Broadcasting Corporation Microcomputer System), a series of microcomputers and related peripherals, proved to be extremely successful and outperformed almost twice as that of Apple II 8-bit personal computer at that time due to the use of DRAM (Dynamic Random Access Memory).

As of 2021, ARM Holdings, a British semiconductor company, have shipped more than 180 billion ARM-based chips worldwide, accounting to over 60% of all processors in-use today and this market-share of ARM is expected to grow even further due to continuous advances in the field of self-driving vehicles, embedded systems and IoT [24].

There are also growing concerns regarding data privacy and usage when user's personal data is processed and stored offline on a central server. Moreover, as the number of devices that rely on central processing increase exponentially, it creates a strain on the existing network and communication infrastructure resulting in periods of black-out and inactivity. To overcome these issues, distributed AI computing on edge (local) devices are extremely appealing and have challenged researchers to create ever more efficient algorithms for computing platforms with limited computational resources.

## 4.2 System Design of NXP BlueBox Gen2 Family

NXP BlueBox is an embedded computing platform designed and developed by NXP Semiconductors, a Dutch-American semiconductor company, for automotive applications like self-driving cars, trucks and SUVs. It is an Automotive High Performance Compute (AHPC) platform that offers prerequisite environment and performance for researchers and engineers to develop autonomous driving, sensor fusion and motion planning automotive applications in addition to providing essential vision acceleration performance and varied automotive interfaces for functional safety applications as well.

NXP Semiconductors introduced BlueBox Gen1, the first generation of BlueBox family, in summer of 2016 at the 2016 NXP FTF Technology Forum hosted by NXP in Austin, Texas. With an ever-growing demand and R&D in field of computer vision and autonomous driving technologies, BlueBox Gen1 opened avenues to a host applications for researchers and developers to design, deploy and test their applications. Shortly thereafter, NXP introduced BlueBox Gen2, the second generation of BlueBox family with three new and improved ARM-based processors: S32V234 for computer vision processing, LS2084A for high performance computing and S32R274 for real-time processing of radar information, for example: LIDAR.

### 4.2.1 S32V Computer Vision Processor

The S32V234 computer vision processor has a quad core ARM Cortex-A53 64-bit CPU operating up to 1000 MHz frequency codified with an ARM Cortex-M4 32-bit CPU for functional safety which is based on an ARMv8-A 64-bit instruction set developed by ARM Holdings' Cambridge design centre. This processor is coupled with an internal 4MB SRAM along with a 32-bit LPDDR3 memory controller to support external memory. It is designed to be an on-chip Image Signal Processor (ISP) designed for ASIL-B/C automotive safety applications and has been optimized to obtain utmost performance per watt efficiency. Figure 4.1 provides a detailed visualization of this processor in the form of a block diagram [25].

### 4.2.2 LS2 High Compute Processor

The LS2084A high performance computing processor has an octa core ARM Cortex-A72 64-bit CPU operating at up to 2.1 GHz frequency. It is based on ARMv8-A 64-bit instruction set developed by ARM Holdings' Austin design centre. The eight cores of the ARM A72 CPU is arranged into four clusters, each cluster containing two cores and sharing a 1MB L2 cache memory. It is complemented with two 64-bit DDR4 SDRAM memory controllers with ECC and interleaving support up to 2.1 GT/s. The LS2 processor provides full support for hardware virtualization, partitioning enforcement, TrustZone architecture, a variety of high-speed peripheral interface controllers such as PCIe 3.0, Serial ATA (SATA) 3.0, Serial Peripheral Interface (SPI), Quad Serial Peripheral Interface (QSPI), I2C synchronous serial communication protocol and an Integrated Flash Controller (IFC) 2.0 to support NAND and NOR flash. Furthermore, the LS2 provides support support in order to be compatible with the next generation LayerScape LX2 family and is designed to certify and comply with automotive quality AEC Q100 Grade 3 standards with 15 years product longevity. Figure 4.2 provides a detailed visualization of this processor in the form of a block diagram [26].

### 4.2.3 S32R Radar Processor

The S32R274 radar processor is a micro-controller to control and process radar information such as LIDAR. It offers various on-chip modules such as Freescale PowerPC-based Architecture e200Z4 32-bit CPU running at 120 MHz frequency with one checker core that as a whole operates as a safety core along with another module that has two Freescale PowerPC-based Architecture e200z7 32-bit CPU cores running at 240 MHz frequency that has a whole operates as a primary computation core for radar processing. It has a 2MB on-chip flash memory with ECC and a 1.5 MB on-chip SRAM memory with ECC. The S32R processor is specially optimized for radar processing, temporary information storage, message buffering and data-stream handling operations. It has been designed to meet the stringent ASIL-D automotive standards for performance and reliability. Figure 4.3 provides a detailed visualization of this processor in the form of a block diagram [27].

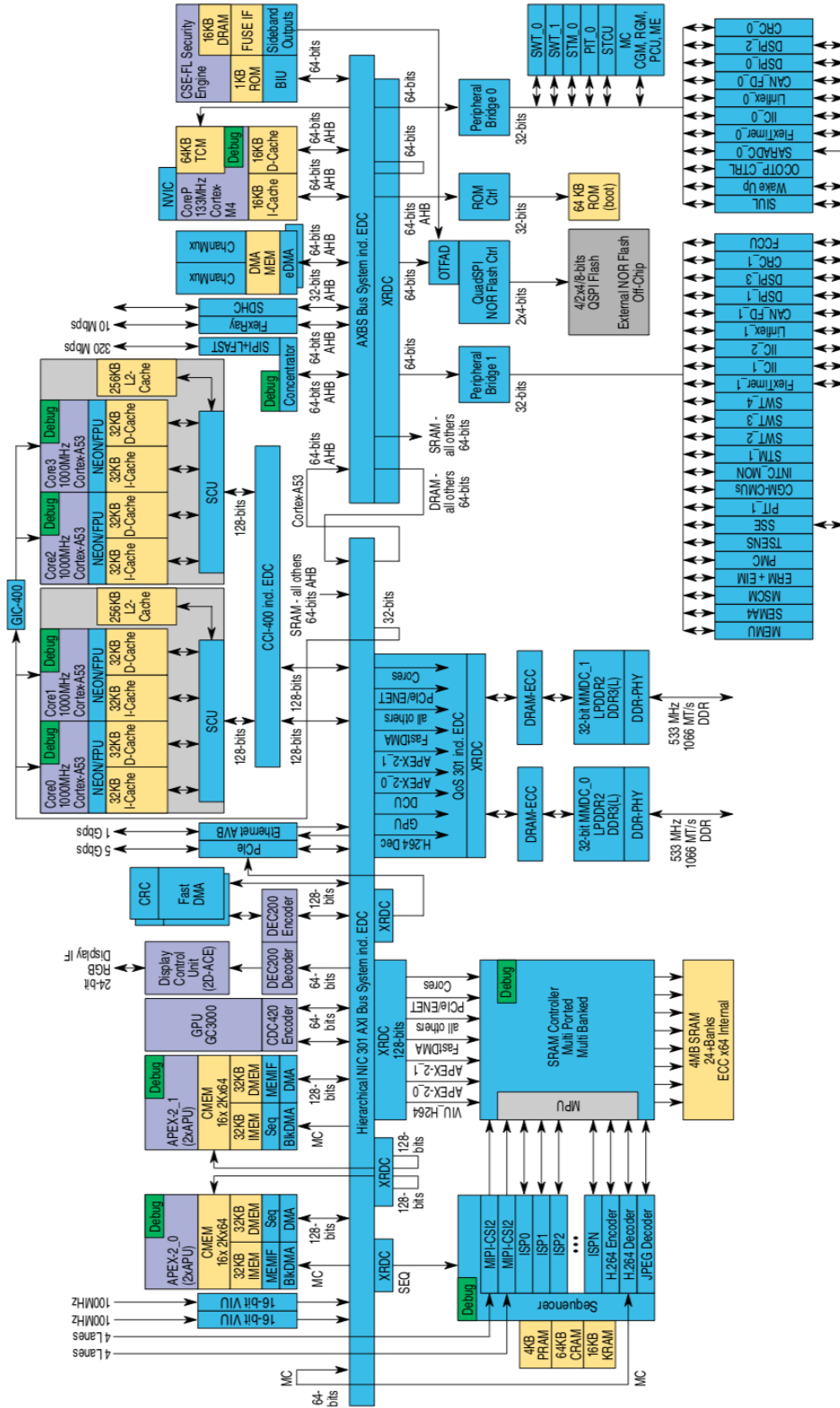


Figure 4.1. Block diagram of S32V computer vision processor.

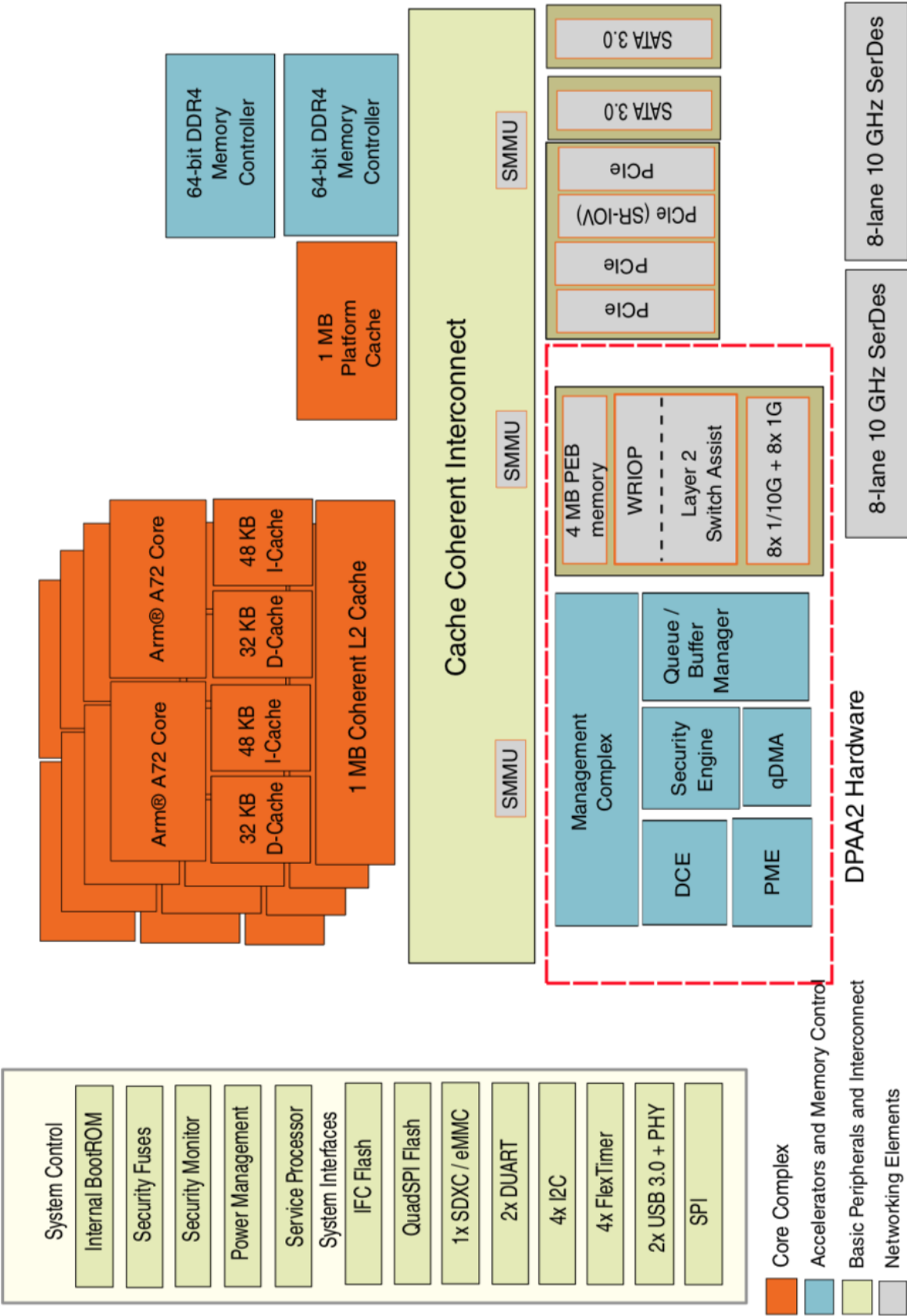


Figure 4.2. Block diagram of LS2 high compute processor.

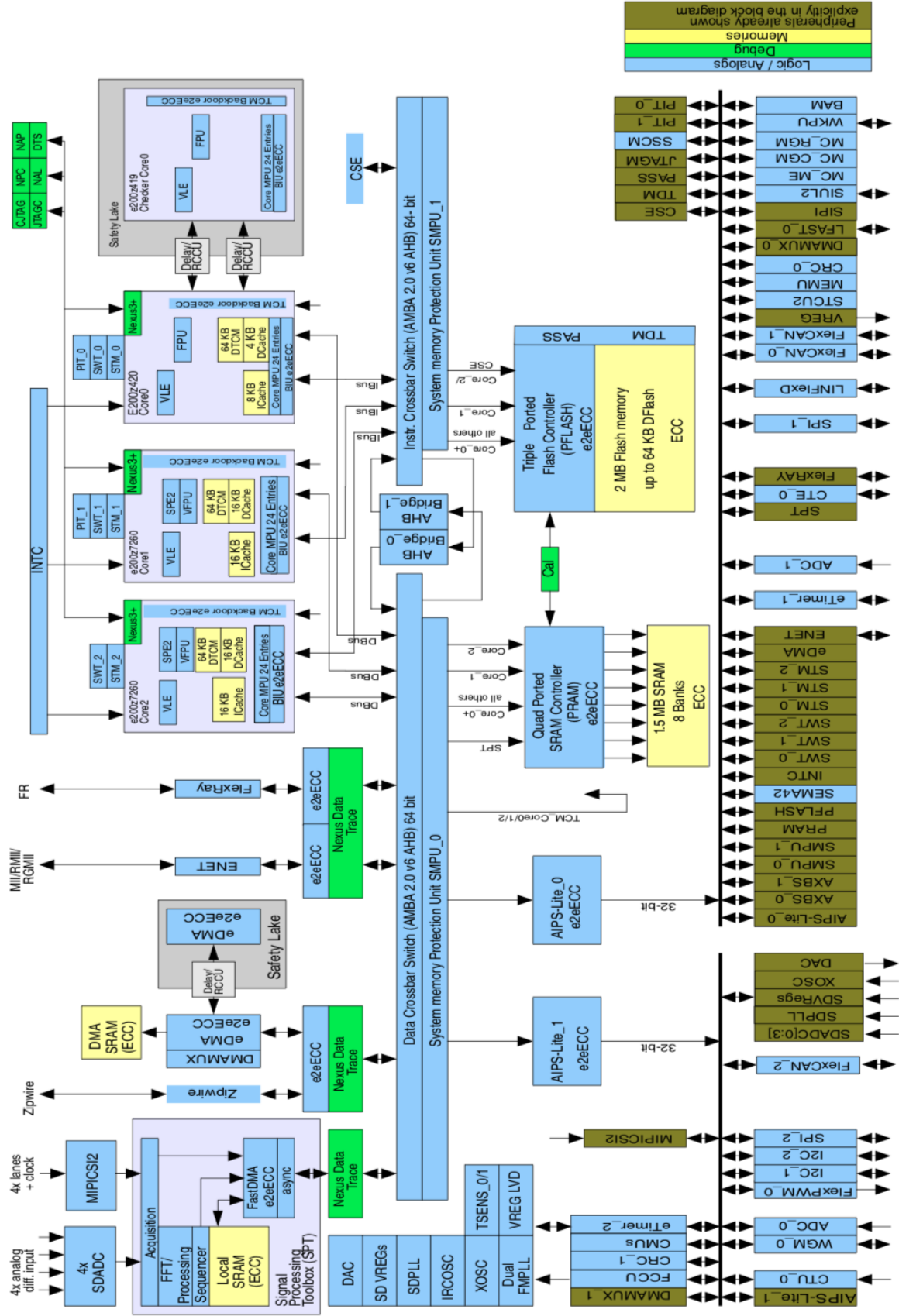


Figure 4.3. Block diagram of S32R radar processor.

### 4.3 System Connectivity of NXP BlueBox Gen2 Family

NXP BlueBox Gen2 (2.0) provides researchers and engineers with a wide variety of communication protocols to connect external peripheral devices and sensors to the BlueBox computing platform:

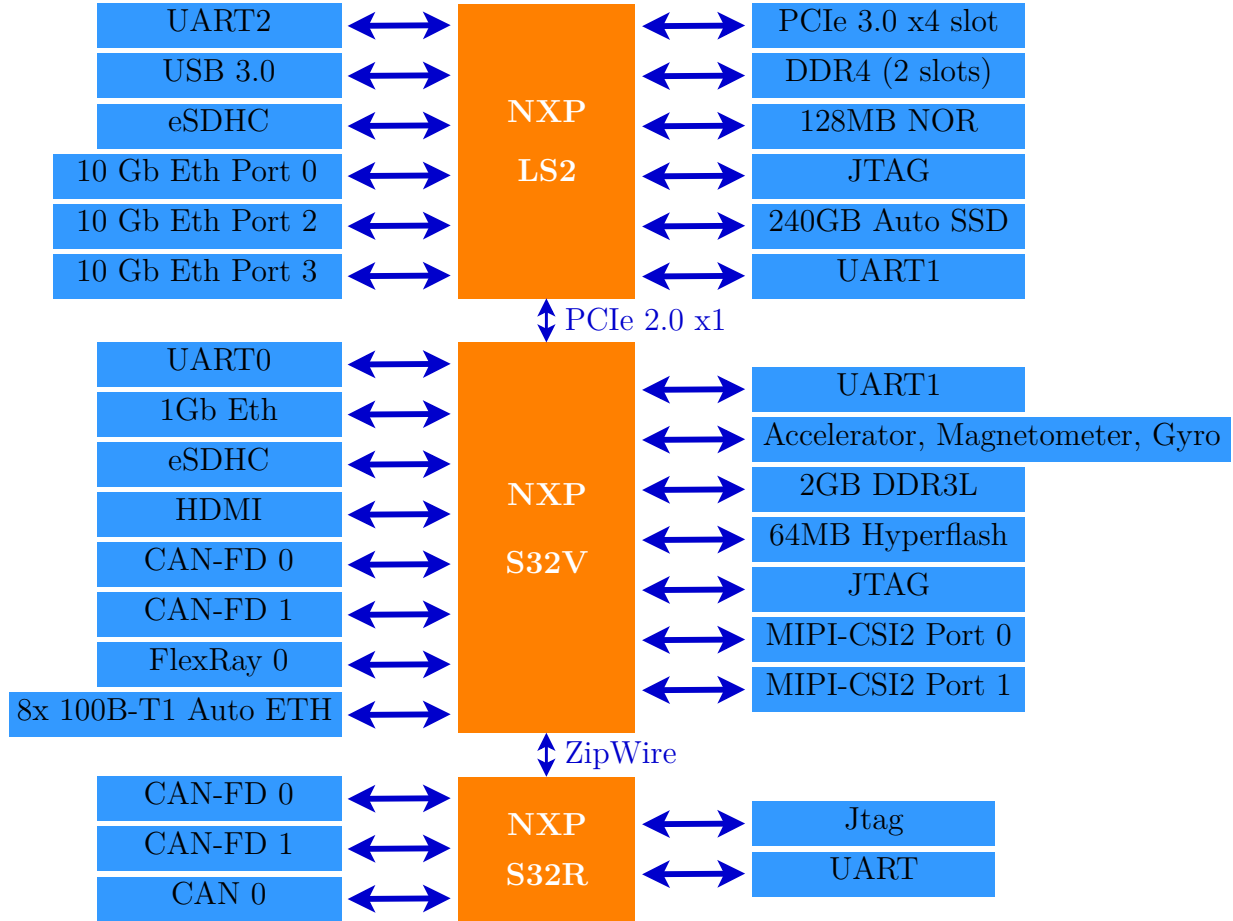
- The S32V processor offers CAN-FD with enhanced payload and data rate, PCIe, Ethernet, FlexRay, Zipwire, SAR-ADCs, SPI and SD card connectivity.
- The LS2 processor offers DUART, I2C, SPIO, GPIO and USB 3.0 interfaces.
- The S32R processor offers JTAG, UART, FlexCAN and Zipwire to connect to a radar ASIC.

Figure 4.4 offers a top-frontal view of the NXP BlueBox 2.0 and Figure 4.5 offers a high-level view of NXP BlueBox 2.0 system Connectivity.



**Figure 4.4.** NXP Bluebox 2.0.



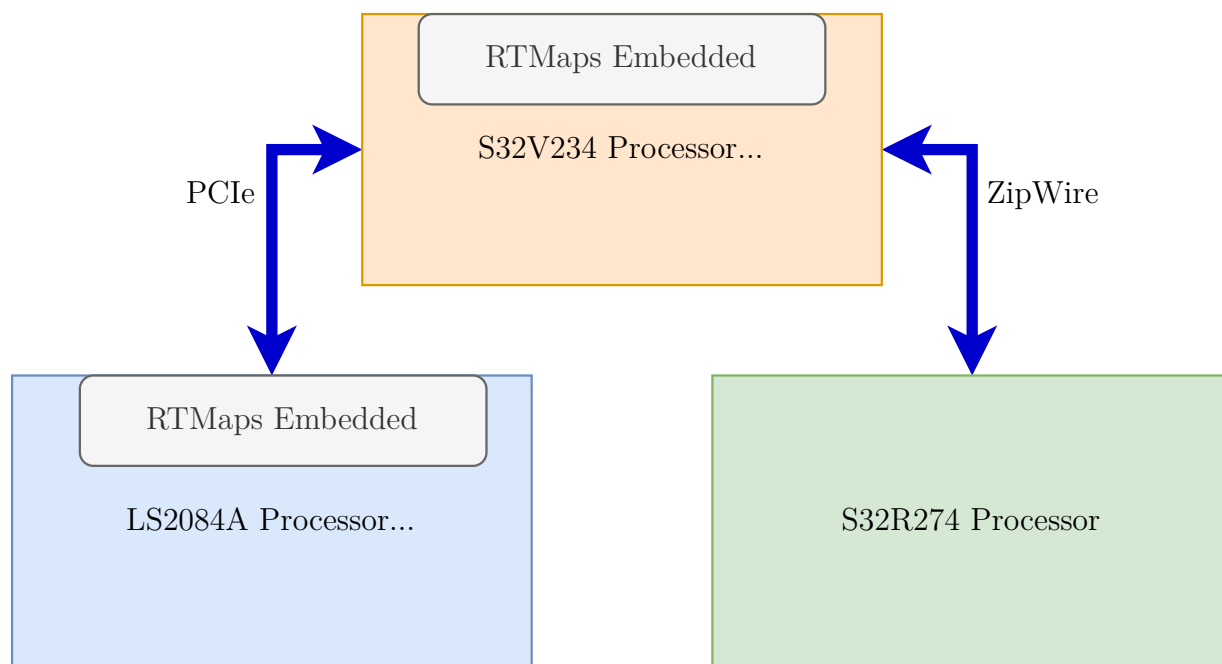


**Figure 4.5.** High-level view of NXP BlueBox Gen2 system connectivity.

## 4.4 RTMaps Remote Studio Software

RTMaps (Real-Time Multi-sensor applications) Remote Studio software is a GUI-based application designed by Intempora to allow researchers and engineers to develop applications for self-driving cars, advanced driver and safety assistance systems and robotics. It is an essential tool to capture, process and view data from different sensors in real-time so that this data can be played back at a later time for reviewing, debugging and testing purposes.

RTMaps software is available on both, Windows and Linux operating systems and natively supports PyTorch and TensorFlow deep learning frameworks. These frameworks are open-source Python based libraries that utilize graphs to perform numerical computation on the input data. In RTMaps, OpenCV algorithms can be developed using Python scripting language and with the help of *blocks* in RTMaps and then be deployed on to any supported embedded computing platform without having to connect any external user-interfacing peripheral devices such as a screen, mouse and a keyboard to the embedded system. Figure 4.6 provides an overview of RTMaps setup on NXP BlueBox 2.0.



**Figure 4.6.** RTMaps on NXP BlueBox 2.0.

## 5. EXPERIMENTS AND RESULTS

Training results presented within this thesis are based on the evaluation of the proposed CondenseNeXt CNN architecture on three benchmarking datasets: CIFAR-10, CIFAR-100 and ImageNet for single image classification [28] utilizing the training cyberinfrastructure and setup outlined, unless specified explicitly otherwise, in this chapter.

### 5.1 Cyberinfrastructure

Cyberinfrastructure (supercomputing and storage resources) for training is provided and managed by the Research Technologies division at the Indiana University which supported the work presented within this thesis in part by Shared University Research grants from IBM Inc. to Indiana University and Lilly Endowment Inc. through its support for the Indiana University Pervasive Technology Institute [29].

Cyberinfrastructure (ARM-based autonomous embedded computing platform) for testing is provided and managed by the Internet of Things (IoT) Collaboratory at the Purdue School of Engineering and Technology at Indiana University Purdue University at Indianapolis, which also supported the work presented within this thesis.

#### 5.1.1 Training Infrastructure

- Intel Xeon Gold 6126 12-core CPU with 128 GB RAM.
- Four NVIDIA Tesla V100 GPU.
- CUDA Toolkit 10.1.243.
- PyTorch version 1.1.0.
- Python version 3.7.9.

### 5.1.2 Testing Infrastructure

- NXP BlueBox 2.0 ARM-based autonomous embedded development platform.
- Intempora RTMaps Remote Studio version 4.8.0.
- CIFAR-10, CIFAR-100 and ImageNet Datasets.
- PyTorch version 1.1.0.
- Python version 3.7.9.

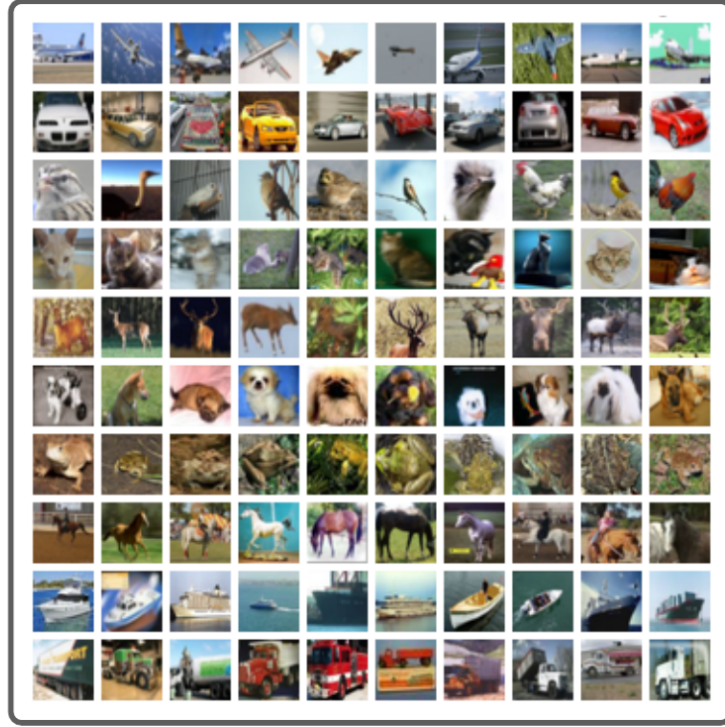
## 5.2 Training and Testing Results for Image Classification

CondenseNeXt CNN architecture has been designed and developed in PyTorch version 1.1.0 framework. PyTorch was introduced by Facebook’s AI Research lab (FAIR) in September 2016. It is a free and open source machine learning library based on python’s Torch library, primarily used to develop computer vision and natural language processing applications. CondenseNeXt was evaluated on the following three benchmarking datasets:

### 5.2.1 CIFAR-10 Dataset

Alex Krizhevsky introduced CIFAR-10 dataset in [30]. It is a collection and a subset of 80 million tiny images dataset, designed to train machine learning algorithms for computer vision applications. The CIFAR-10 dataset consists of a total of 60,000 low resolution ( $32 \times 32$  pixels) images, split in two sets of 50,000 for training the neural network and 10,000 for testing the neural network. The training and testing sets are mutually exclusive to obtain a fair error rate.

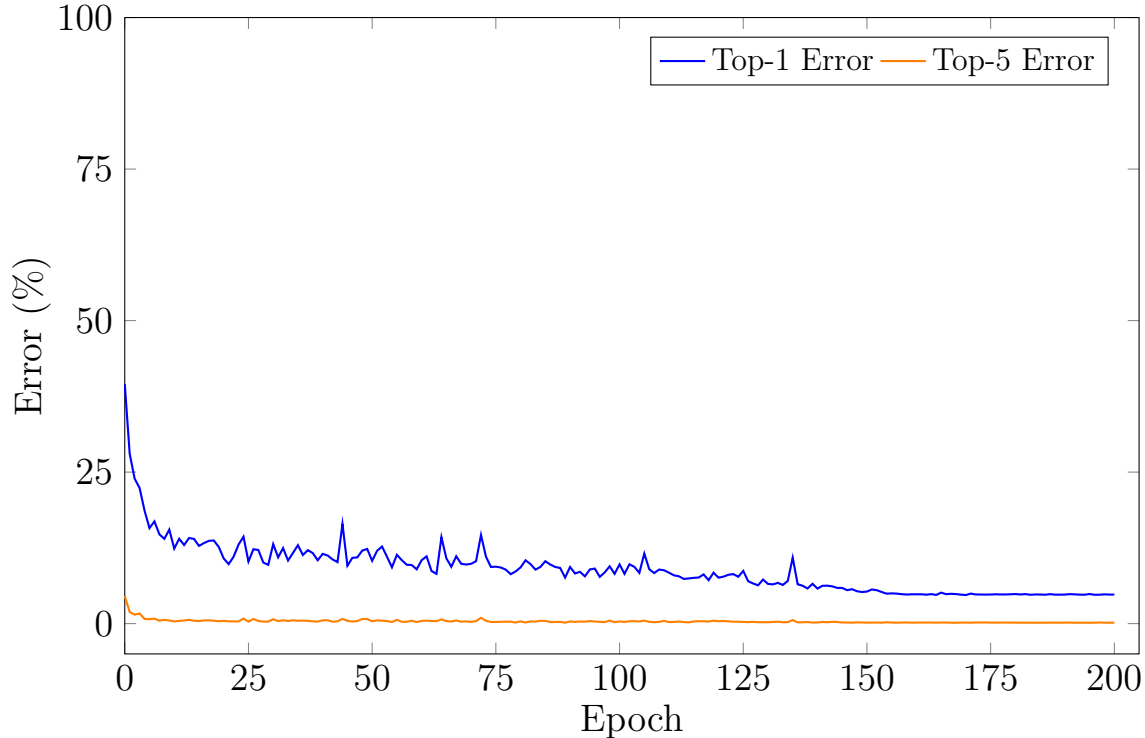
CIFAR in the name CIFAR-10 stands for Canadian Institute for Advanced Research and the number 10 signifies that this dataset consists of the following ten classes: airplanes, birds, cars, cats, deer, dogs, frogs, horses, ships, and trucks. Because of a very small number of classes and low resolution photos, researchers are able to quickly test different algorithms and techniques. Thus, making it one of the most popular and widely used datasets in the field of machine learning research.



**Figure 5.1.** CIFAR-10 classes for image classification.

CondenseNeXt was trained with a single crop of inputs on CIFAR-10 dataset utilizing the aforementioned cyberinfrastructure and following additional setup:

- Epochs: 200
- Stages: 4 – 4 – 4
- Batch Size: 64
- Feature Size  $k$ : 8 – 8 – 8
- Nesterov Momentum Weight: 0.9
- Dropout and Learning Rate: 0.1
- Learning Rate Type: Cosine Shape
- Gradient Descent: Stochastic Gradient Descent (SGD)



**Figure 5.2.** Training overview of CondenseNeXt on CIFAR-10 dataset.

Figure 5.2 plots Top-1 and Top-5 accuracies as a function of training epoch (in orange and blue) of CondenseNeXt CNN architecture trained on CIFAR-10 dataset [30]. Table 5.1 provides a comparison of performance between the baseline architecture, CondenseNet, and the proposed CNN architecture, CondenseNeXt, in terms of FLOPs, parameters, and Top-1 and Top-5 accuracies to achieve a trained model size (final weight) of 2.9 MB.

CondenseNeXt achieves a Top-1 accuracy (top class with highest accuracy that match the ground truth) of 92.28% (Top-1 error rate: 7.72%) and Top-5 accuracy (top five classes with the highest accuracy that match the ground truth) of 99.77% (Top-5 error rate: 0.23%) along with 63.84% reduction in forward FLOPs utilizing the aforementioned cyberinfrastructure, training setup and hyperparameters.

**Table 5.1.** Comparison of Experiment 1 Performance on CIFAR-10 Dataset.

Architecture	FLOPs (in millions)	Parameters (in millions)	Top-1 % Error	Top-5 % Error
CondenseNet	65.81	0.52	5.76	0.19
CondenseNeXt	<b>23.80</b>	<b>0.16</b>	<b>7.72</b>	<b>0.23</b>

Another extensive analysis was performed on CIFAR-10 dataset to achieve a greater accuracy (lower error rate) at the cost of increased size of the final weight (in megabytes) in comparison to the baseline architecture, using following training hyperparameters:

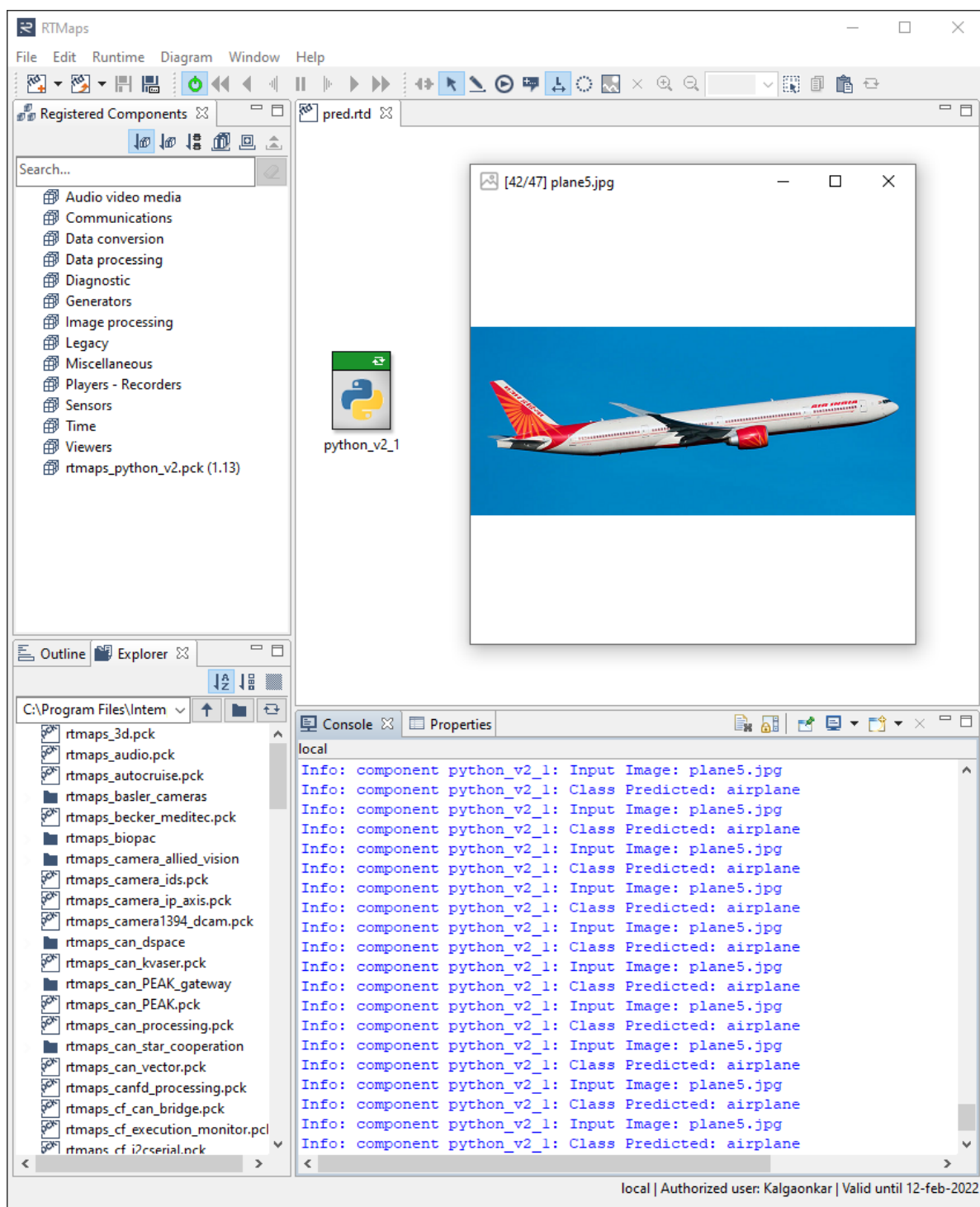
- Epochs: 200
- Batch Size: 64
- Stages: 14 – 14 – 14
- Feature Size  $k$ : 8 – 16 – 32
- Nesterov Momentum Weight: 0.9
- Dropout and Learning Rate: 0.1
- Learning Rate Type: Cosine Shape
- Gradient Descent: Stochastic Gradient Descent (SGD)

CondenseNeXt achieves a Top-1 accuracy (top class with highest accuracy that match the ground truth) of 95.21% (Top-1 error rate: 5.31%) and Top-5 accuracy (top five classes with the highest accuracy that match the ground truth) of 99.82% (Top-5 error rate: 0.18%) along with 64.52% reduction in forward FLOPs utilizing the aforementioned cyberinfrastructure, training setup and hyperparameters as shown in the table 5.2 below.

An image classification script was developed using Python scripting language in RTMaps Remote Studio and the CondenseNeXt’s trained model (weight) was evaluated on NXP BlueBox 2.0 for single image classification analysis. Figure 5.3 provides a screenshot of the RTMaps console.

**Table 5.2.** Comparison of Experiment 2 Performance on CIFAR-10 Dataset.

Architecture	FLOPs (in millions)	Parameters (in millions)	Top-1 % Error	Top-5 % Error
CondenseNet	65.81	0.52	5.31	0.24
CondenseNeXt	<b>26.35</b>	<b>0.18</b>	<b>4.79</b>	<b>0.15</b>



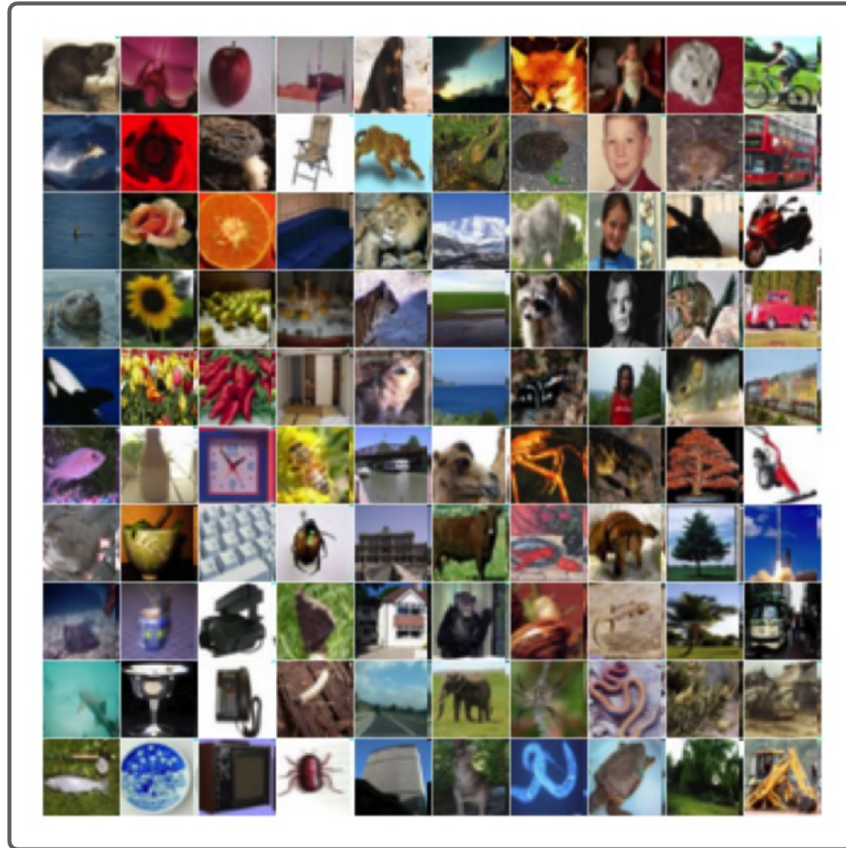
**Figure 5.3.** Evaluation of CondenseNeXt on CIFAR-10 dataset when deployed on NXP BlueBox 2.0 using RTMaps Remote Studio 4.8.0 for classifying an image of an airplane and outputting the predicted class in RTMaps console.



### 5.2.2 CIFAR-100 Dataset

Along side CIFAR-10, Alex Krizhevsky also introduced CIFAR-100 dataset in [30]. Like CIFAR-10, CIFAR-100 is also a subset of 80 million tiny images dataset, designed to train machine learning algorithms for computer vision applications. Furthermore, it also consists of a total of 60,000 low resolution ( $32 \times 32$ ) images, split into two sets of 50,000 for training the neural network and 10,000 for testing the neural network, and the training and testing sets are mutually exclusive to obtain a fair error rate. However, unlike CIFAR-10, CIFAR-100 has 100 classes where each class contains 600 images.

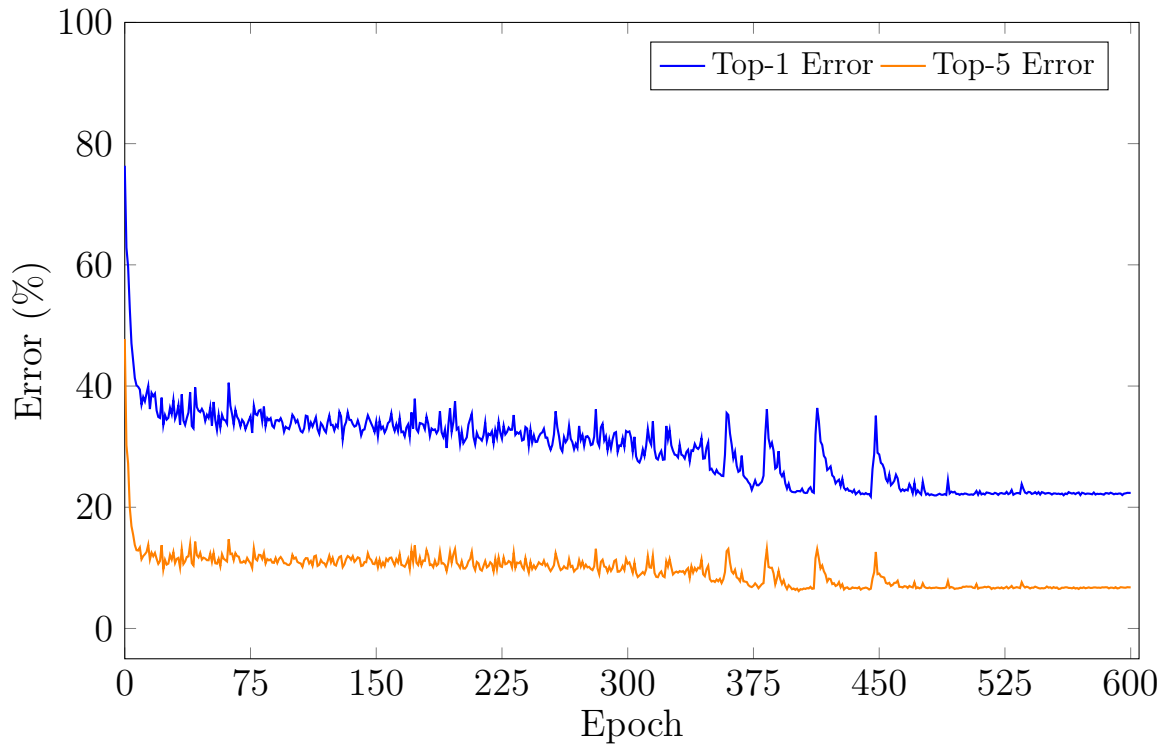
Classes and images of CIFAR-10 and CIFAR-100 are mutually exclusive. For example, CIFAR-100's roses, road, castle and man classes are not part of CIFAR-10 dataset's classes. Figure 5.4 provides a brief overview of all 100 classes in this dataset.



**Figure 5.4.** CIFAR-100 classes for image classification.

CondenseNeXt was trained with a single crop of inputs on CIFAR-100 dataset utilizing the aforementioned cyberinfrastructure and following additional setup:

- Epochs: 600
- Batch Size: 64
- Stages: 14 – 14 – 14
- Feature Size  $k$ : 8 – 16 – 32
- Nesterov Momentum Weight: 0.9
- Dropout and Learning Rate: 0.1
- Learning Rate Type: Cosine Shape
- Gradient Descent: Stochastic Gradient Descent (SGD)



**Figure 5.5.** Training overview of CondenseNeXt on CIFAR-100 dataset.

**Table 5.3.** Comparison of Performance on CIFAR-100 Dataset.

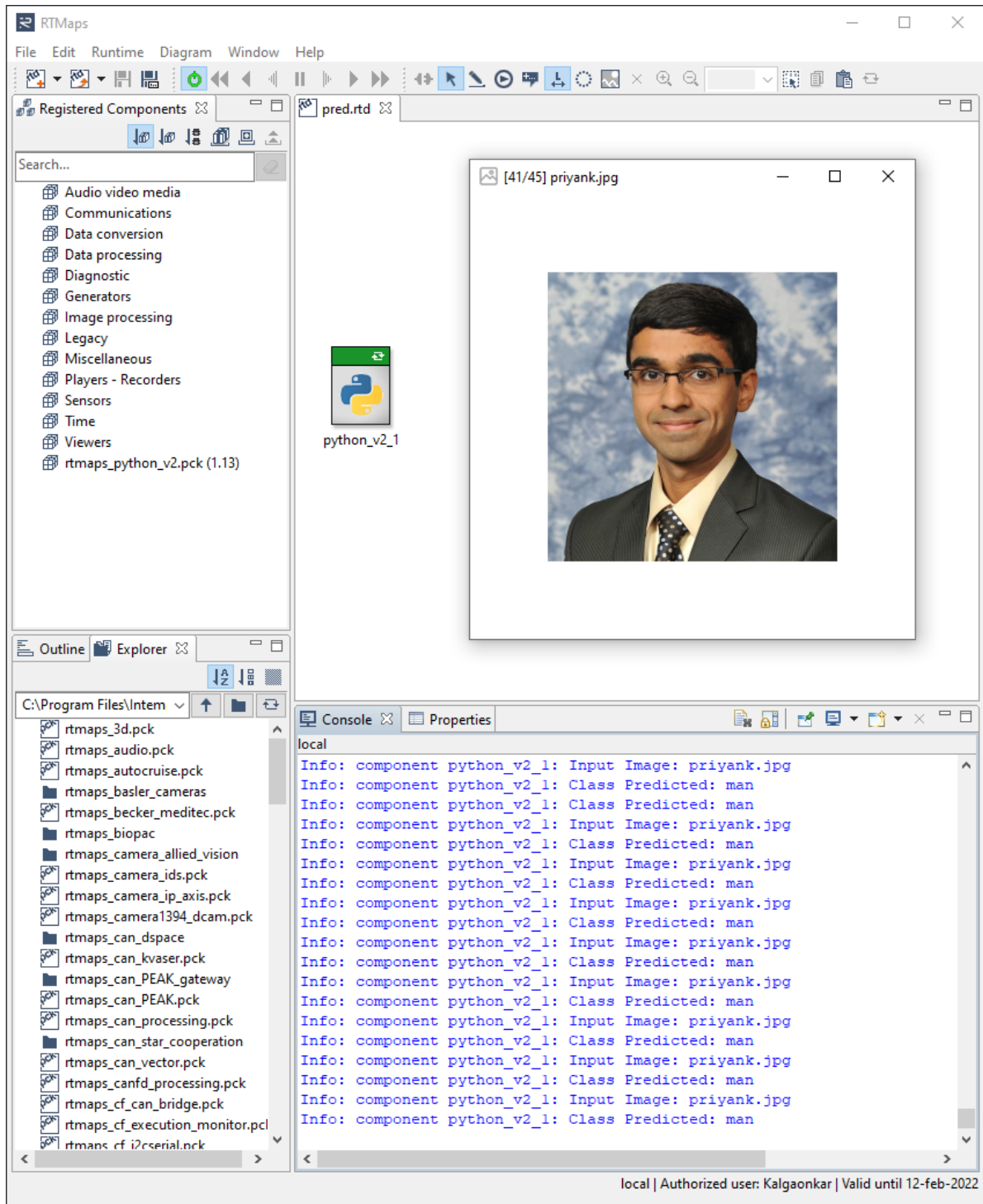
Architecture	FLOPs (in millions)	Parameter (in millions)	Top-1 % Error	Top-5 % Error
CondenseNet	65.85	0.55	23.35	6.56
CondenseNeXt	<b>26.38</b>	<b>0.22</b>	<b>21.98</b>	<b>6.29</b>

Figure 5.5 plots Top-1 and Top-5 accuracies as a function of training epoch (in orange and blue) of CondenseNeXt CNN architecture trained on CIFAR-100 dataset [30]. Table 5.3 provides a comparison between the baseline architecture, CondenseNet, and the proposed CNN architecture, CondenseNeXt, in terms of FLOPs, parameters, Top-1 and Top-5 accuracies.

The abrupt increase in the error rate at epochs 363, 383, 413 and 448 is caused by the final pruning operation discarding (purging) half of the remaining weights. However, the plotted graph demonstrates that the model slowly recovers from this pruning effect and optimizes itself, eventually becoming stable.

CondenseNeXt achieves a Top-1 accuracy (top class with highest accuracy that match the ground truth) of 78.02% (Top-1 error rate: 21.98%) and Top-5 accuracy (top five classes with the highest accuracy that match the ground truth) of 93.71% (Top-5 error rate: 6.29%) along with 59.94% reduction in forward FLOPs utilizing the aforementioned cyberinfrastructure, training setup and hyperparameters.

An image classification script was developed using Python scripting language in RTMaps Remote Studio and the CondenseNeXt’s trained model (weight) was evaluated on NXP BlueBox 2.0 for single image classification analysis. Figure 5.6 provides a screenshot of the RTMaps console.



**Figure 5.6.** Evaluation of CondenseNeXt on CIFAR-100 dataset when deployed on NXP BlueBox 2.0 using RTMaps Remote Studio 4.8.0 for classifying an image of a human and outputting the predicted class in RTMaps console.

### 5.2.3 ImageNet Dataset

ImageNet dataset was introduced by Dr. Fei-Fei Li, an AI researcher at Stanford University, along with a team of researchers at an IEEE CVPR conference in Florida in 2019 [31]. It is a very large image database organized and built according to the WordNet hierarchy where every node in this hierarchy corresponds to hundreds of images. Unlike in a dictionary, in WordNet hierarchy, words are related to other words rather than in an alphabetical order. For example, within WordNet, the word *hatchback* would be nested under the word *motor car* which would be nested under *motor vehicle*, and so on and so forth.

ImageNet contains over 20,000 common categories such as table, chair, bucket, etc., each comprising of several hundred images. In total, there are over 14 million images in this dataset that have been hand-annotated and labelled by the team. Figure 5.7 provides a brief overview of this dataset.

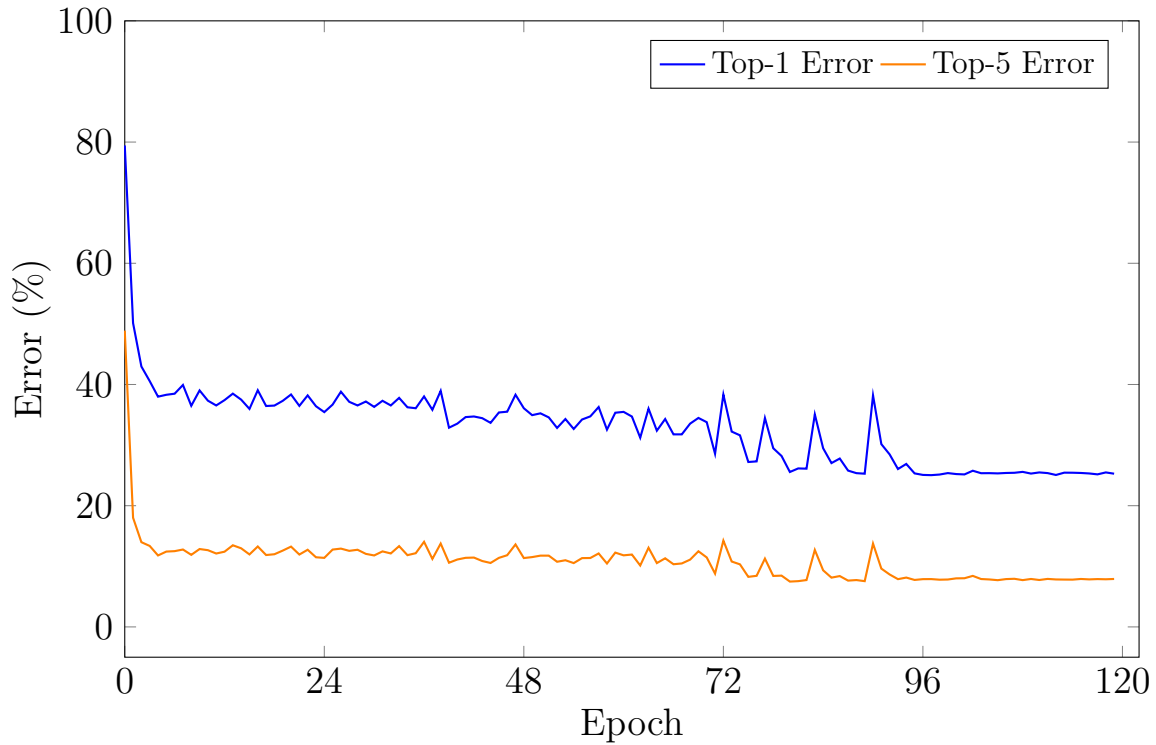


**Figure 5.7.** ImageNet classes for image classification.



CondenseNeXt was trained with a single crop of inputs on entire ImageNet dataset from Large Scale Visual Recognition Challenge (ILSVRC) 2012, utilizing the aforementioned cyberinfrastructure and following hyperparameters using data parallelism technique:

- Epochs: 120 and Group Lasso: 0.00001
- Stages: 4 – 6 – 8 – 10 – 8 and Batch Size: 256
- Feature Size  $k$ : 8 – 16 – 32 – 64 – 128
- Nesterov Momentum Weight: 0.9
- Dropout and Learning Rate: 0.1
- Learning Rate Type: Cosine Shape
- Gradient Descent: Stochastic Gradient Descent (SGD)



**Figure 5.8.** Training overview of CondenseNeXt on ImageNet dataset.

**Table 5.4.** Comparison of Performance on ImageNet Dataset.

Architecture	FLOPs (in millions)	Parameter (in millions)	Top-1 % Error	Top-5 % Error
CondenseNet	529.36	4.81	26.2	8.30
CondenseNeXt	<b>273.16</b>	<b>3.07</b>	<b>25.8</b>	<b>7.91</b>

Figure 5.8 plots Top-1 and Top-5 accuracies as a function of training epoch (in orange and blue) of CondenseNeXt CNN architecture trained on ImageNet dataset [30] using PyTorch supported data parallelism technique to accelerate training process across four Nvidia V100 GPUs. Table 5.4 provides a comparison between the baseline architecture, CondenseNet, and the proposed CNN architecture, CondenseNeXt, in terms of FLOPs, parameters, and Top-1 and Top-5 accuracies.

As seen before with training on CIFAR-100 dataset, the abrupt increase in the error rate at epochs 72, 77, 83 and 90 is caused by the final pruning operation discarding (purging) half of the remaining weights. However, the graph plotted in Figure 5.8 shows that the model slowly recovers from this pruning effect, optimizes itself and eventually becomes stable.

CondenseNeXt achieves a Top-1 accuracy (top class with highest accuracy that match the ground truth) of 74.20% (Top-1 error rate: 25.8%) and Top-5 accuracy (top five classes with the highest accuracy that match the ground truth) of 92.09% (Top-5 error rate: 7.91%) along with 48.39% reduction in forward FLOPs utilizing the aforementioned cyberinfrastructure, training setup and hyperparameters.

An image classification script was developed using Python scripting language in RTMaps Remote Studio and the CondenseNeXt's trained model (weight) was evaluated on NXP BlueBox 2.0 for single image classification analysis. Figure 5.9 provides a screenshot of the RTMaps console.





### 5.3 Summary

Extensive training and image classification analyses are conducted on CIFAR-10, CIFAR-100 and ImageNet image datasets. Following tables provide a quick overview of performance results obtained from experiments outlined in this thesis for the proposed ultra efficient CNN architecture, CondenseNeXt, which is compared to the baseline architecture, CondenseNet.

CondenseNeXt utilizes state-of-the-art convolutional layers and model compression techniques which results in significant reduction of forward FLOPs, increase in overall accuracy (decrease in error rate) as well as decrease in evaluation, time resulting in an outstanding performance.

**Table 5.5.** Summary of CondenseNeXt’s Performance in Terms of Error Rate.

Dataset	CNN Architecture	FLOPs (in millions)	Parameters (in millions)	Top-1 % Error	Top-5 % Error
CIFAR-10	CondenseNet	65.81	0.52	5.31	0.24
	CondenseNeXt	<b>26.35</b>	<b>0.18</b>	<b>4.79</b>	<b>0.15</b>
CIFAR-100	CondenseNet	65.85	0.55	23.35	6.56
	CondenseNeXt	<b>26.38</b>	<b>0.22</b>	<b>21.98</b>	<b>6.29</b>
ImageNet	CondenseNet	529.36	4.81	26.20	8.30
	CondenseNeXt	<b>273.16</b>	<b>3.07</b>	<b>25.80</b>	<b>7.91</b>

**Table 5.6.** Summary of CondenseNeXt’s Performance in Terms of Accuracy.

Dataset	CNN Architecture	FLOPs (in millions)	Parameters (in millions)	Top-1 % Accuracy	Top-5 % Accuracy
CIFAR-10	CondenseNet	65.81	0.52	94.26	99.76
	CondenseNeXt	<b>26.35</b>	<b>0.18</b>	<b>94.26</b>	<b>99.85</b>
CIFAR-100	CondenseNet	65.85	0.55	95.21	93.44
	CondenseNeXt	<b>26.38</b>	<b>0.22</b>	<b>78.02</b>	<b>93.71</b>
ImageNet	CondenseNet	529.36	4.81	73.80	91.70
	CondenseNeXt	<b>273.16</b>	<b>3.07</b>	<b>74.20</b>	<b>7.91</b>

**Table 5.7.** Image Classification Evaluation Time of CondenseNeXt on NXP BlueBox 2.0

Dataset	CNN Architecture	FLOPs (in millions)	Parameters (in millions)	Evaluation Time (in seconds)
CIFAR-10	CondenseNet	65.81	0.52	0.1346
	CondenseNeXt	<b>26.35</b>	<b>0.18</b>	<b>0.0359</b>
CIFAR-100~	CondenseNet	65.85	0.55	0.2483
	CondenseNeXt	<b>26.38</b>	<b>0.22</b>	<b>0.1425</b>
ImageNet	CondenseNet	529.36	4.81	2.3671
	CondenseNeXt	<b>273.16</b>	<b>3.07</b>	<b>1.7483</b>

## 6. CONCLUSION

Research work presented within this thesis propose a neoteric variant of deep convolutional neural network architecture, CondenseNeXt, designed specifically for ARM-based embedded computing platforms with constrained computational resources. This thesis commenced with a brief discussion of the problem and its background in Chapter 1, followed by introduction to fundamentals of neural networks in Chapter 2 which provides the reader a firm foundation to commend the thesis statement proffered by the research work presented in Chapter 3 and for the merits upon which experiment results have been successfully manifested in Chapter 5. CondenseNeXt is an improved version of CondenseNet, the baseline architecture whose roots can be traced back to ResNet. CondeseNeXt replaces group convolutions in CondenseNet with depthwise separable convolutions and introduces group-wise pruning, a model compression technique, to prune (remove) redundant and insignificant elements that either are irrelevant or do not affect performance of the network upon disposition. Cardinality, a new dimension to the existing spatial dimensions, and class-balanced focal loss function, a weighting factor inversely proportional to the number of samples, has been incorporated in order to relieve the harsh effects of pruning, into the design of CondenseNeXt’s algorithm. Furthermore, extensive analyses of this novel CNN architecture was performed on three benchmarking image datasets: CIFAR-10, CIFAR-100 and ImageNet by deploying the trained weight on to an ARM-based embedded computing platform: NXP BlueBox 2.0, for real-time image classification. The outputs are observed in real-time in RTMaps Remote Studio’s console to verify the correctness of classes being predicted. CondenseNeXt achieves state-of-the-art image classification performance on three benchmark datasets including CIFAR-10 (4.79% top-1 error), CIFAR-100 (21.98% top-1 error) and ImageNet (7.91% single model, single crop top-5 error), and up to 59.98% reduction in forward FLOPs compared to CondenseNet. CondenseNeXt can also achieve a final trained model size of 2.9 MB, however at the cost of 2.26% in accuracy loss (2.26% increase in error rate). Thus, performing image classification on ARM-Based computing platforms without requiring a CUDA enabled GPU support, with outstanding efficiency.

## 7. RECOMMENDATIONS

Results presented within this thesis provide some of the first empirical data of combining depthwise separable convolution with model compression (group-wise pruning) techniques. Based on these results and conclusions, researchers can further investigate on improving the performance of CondenseNeXt by performing hyper-parameter tuning in order to obtain greater performance as well as explore cutting edge input data augmentation schemes and policies in order to improve image classification accuracy. Furthermore, experiments can be performed on YOLO (You Look Only Once) [32] and Microsoft COCO (Common Objects in Context) [33] datasets for real-time object detection on embedded computing platforms without a CUDA-enabled GPU in order to understand how CondenseNeXt performs on these different types of datasets during inference time. As demand for smaller yet efficient deep neural networks increase with innovations in embedded systems, distributed computing and edge devices, the scope for AI research and innovation in this field is immensurable.

## REFERENCES

- [1] A. Coates, G. Bradski, A. Coates, L. Deng, and M. Shah. (Jul. 7, 2017). “Ask the ai experts: What’s driving today’s progress in ai?” [Online]. Available: <https://www.mckinsey.com/business-functions/mckinsey-analytics/our-insights/ask-the-ai-experts-whats-driving-todays-progress-in-ai>. Last Accessed: 03/17/21.
- [2] W. Gropp, *Lecture 15: Moore’s Law and Dennard Scaling*. [Online]. Available: [www.cs.illinois.edu/~wgropp](http://www.cs.illinois.edu/~wgropp), Last Accessed: 03/17/21.
- [3] S. Haykin, *Neural Networks and Learning Machines*. Pearson Prentice Hall, 1999, ISBN: 978-0-13-1471-39-9.
- [4] A. Ivakhnenko and V. Lapa, “Cybernetics and forecasting techniques,” *Nature*, vol. 219, pp. 202–203, Jul. 1, 1968. DOI: [10.1038/219202b0](https://doi.org/10.1038/219202b0).
- [5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.
- [6] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. DOI: [10.1109/CVPR.2016.90](https://doi.org/10.1109/CVPR.2016.90).
- [7] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269. DOI: [10.1109/CVPR.2017.243](https://doi.org/10.1109/CVPR.2017.243).
- [8] G. Huang, S. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Condensenet: An efficient densenet using learned group convolutions,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2752–2761. DOI: [10.1109/CVPR.2018.00291](https://doi.org/10.1109/CVPR.2018.00291).
- [9] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5987–5995. DOI: [10.1109/CVPR.2017.634](https://doi.org/10.1109/CVPR.2017.634).
- [10] P. Kalgaonkar and M. El-Sharkawy, “Condensenext: An ultra-efficient deep neural network for embedded systems,” in *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, 2021, pp. 0524–0528. DOI: [10.1109/CCWC51732.2021.9375950](https://doi.org/10.1109/CCWC51732.2021.9375950).

- [11] P. Angelov, A. Gegov, C. Jayne, and Q. Shen, *Advances in Computational Intelligence Systems: Contributions Presented at the 16th UK Workshop on Computational Intelligence, September 79, 2016*, 1st. Springer Publishing Company, Incorporated, 2016, ISBN: 3319465619.
- [12] Y. LeCun, L. D. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard, *et al.*, “Learning algorithms for classification: A comparison on handwritten digit recognition,” *Neural Networks: The Statistical Mechanics Perspective*, vol. 261, no. 276, p. 2, 1995.
- [13] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in *European conference on computer vision*, Springer, 2014, pp. 818–833.
- [14] L. Kaiser, A. N. Gomez, and F. Chollet, “Depthwise separable convolutions for neural machine translation,” *arXiv preprint arXiv:1706.03059*, 2017.
- [15] F. Mamalet and C. Garcia, “Simplifying convnets for fast learning,” in *International Conference on Artificial Neural Networks*, Springer, 2012, pp. 58–65.
- [16] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.
- [17] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1800–1807. DOI: [10.1109/CVPR.2017.195](https://doi.org/10.1109/CVPR.2017.195).
- [18] V. Lebedev and V. Lempitsky, “Fast convnets using group-wise brain damage,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2554–2564. DOI: [10.1109/CVPR.2016.280](https://doi.org/10.1109/CVPR.2016.280).
- [19] J. Luo, J. Wu, and W. Lin, “Thinet: A filter level pruning method for deep neural network compression,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 5068–5076. DOI: [10.1109/ICCV.2017.541](https://doi.org/10.1109/ICCV.2017.541).
- [20] K. Lee, H. Kim, H. Lee, and D. Shin, “Flexible group-level pruning of deep neural networks for on-device machine learning,” in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 79–84. DOI: [10.23919/DATE48585.2020.9116287](https://doi.org/10.23919/DATE48585.2020.9116287).
- [21] S. Wu, G. Li, L. Deng, L. Liu, D. Wu, Y. Xie, and L. Shi, “ $l_1$ -norm batch normalization for efficient training of deep neural networks,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 7, pp. 2043–2051, 2019. DOI: [10.1109/TNNLS.2018.2876179](https://doi.org/10.1109/TNNLS.2018.2876179).

- [22] Y. Cui, M. Jia, T. Lin, Y. Song, and S. Belongie, “Class-balanced loss based on effective number of samples,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 9260–9269. DOI: [10.1109/CVPR.2019.00949](https://doi.org/10.1109/CVPR.2019.00949).
- [23] A. Krizhevsky and G. Hinton, “Convolutional deep belief networks on cifar-10,” *Unpublished manuscript*, vol. 40, no. 7, pp. 1–9, 2010.
- [24] P. Hughes. (Feb. 11, 2021). “The arm ecosystem ships a record 6.7 billion arm-based chips in a single quarter,” [Online]. Available: <https://www.arm.com/company/news/2021/02/arm-ecosystem-ships-record-6-billion-arm-based-chips-in-a-single-quarter>. Last Accessed: 03/22/21.
- [25] NXP Semiconductors, *S32V234 Data Sheet*. 2020. [Online]. Available: <https://www.nxp.com/docs/en/data-sheet/S32V234.pdf>, Last Accessed: 03/22/21.
- [26] NXP Semiconductors, *QorIQ LS2084A/LS2044A Data Sheet*. 2020. [Online]. Available: <https://www.nxp.com/docs/en/data-sheet/LS2084A.pdf>, Last Accessed: 03/22/21.
- [27] NXP Semiconductors, *S32R274/S32R264 Series Data Sheet*. 2020. [Online]. Available: <https://www.nxp.com/docs/en/data-sheet/S32R274DS.pdf>, Last Accessed: 03/22/21.
- [28] P. Kalgaonkar and M. El-Sharkawy, “Image classification with condensenext for arm-based computing platforms,” in *2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, 2021, pp. 1–6. DOI: [10.1109/IEMTRONICS52119.2021.9422541](https://doi.org/10.1109/IEMTRONICS52119.2021.9422541).
- [29] “Indiana university pervasive technology institute,” Sep. 1, 2017. DOI: [10.5967/K8G44NGB](https://doi.org/10.5967/K8G44NGB). [Online]. Available: <http://hdl.handle.net/2022/21675>, Last Accessed: 03/23/21.
- [30] A. Krizhevsky, G. Hinton, *et al.*, “Learning multiple layers of features from tiny images,” 2009.
- [31] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. DOI: [10.1109/CVPR.2009.5206848](https://doi.org/10.1109/CVPR.2009.5206848).
- [32] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.
- [33] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in *European conference on computer vision*, Springer, 2014, pp. 740–755.

# PUBLICATIONS

Published as: P. Kalgaonkar and M. El-Sharkawy, “CondenseNeXt: An Ultra-Efficient Deep Neural Network for Embedded Systems”, *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, NV, USA, 2021, pp. 0524-0528, DOI: [10.1109/CCWC51732.2021.9375950](https://doi.org/10.1109/CCWC51732.2021.9375950).

## CondenseNeXt: An Ultra-Efficient Deep Neural Network for Embedded Systems

Priyank Kalgaonkar

Department of Electrical and Computer Engineering  
Purdue School of Engineering and Technology  
Indianapolis, Indiana 46202, USA.  
pkalgaon@purdue.edu

Mohamed El-Sharkawy

Department of Electrical and Computer Engineering  
Purdue School of Engineering and Technology  
Indianapolis, Indiana 46202, USA.  
melshark@purdue.edu

**Abstract**—Due to the advent of modern embedded systems and mobile devices with constrained resources, there is a great demand for incredibly efficient deep neural networks for machine learning purposes. There is also a growing concern of privacy and confidentiality of user data within the general public when their data is processed and stored in an external server which has further fueled the need for developing such efficient neural networks for real-time inference on local embedded systems. The scope of our work presented in this paper is limited to image classification using a convolutional neural network. A Convolutional Neural Network (CNN) is a class of Deep Neural Network (DNN) widely used in the analysis of visual images captured by an image sensor, designed to extract information and convert it into meaningful representations for real-time inference of the input data. In this paper, we propose a neoteric variant of deep convolutional neural network architecture to ameliorate the performance of existing CNN architectures for real-time inference on embedded systems. We show that this architecture, dubbed CondenseNeXt, is remarkably efficient in comparison to the baseline neural network architecture, CondenseNet, by reducing trainable parameters and FLOPs required to train the network whilst maintaining a balance between the trained model size of less than 3.0 MB and accuracy trade-off resulting in an unprecedented computational efficiency.

**Index Terms**—Deep Neural Network, Convolutional Neural Network, CondenseNeXt, CondenseNet, Depthwise Separable Convolutions, Group-wise Pruning, PyTorch, CIFAR-10.

### I. INTRODUCTION

Convolutional Neural Networks, a class of Deep Neural Networks, have been gaining popularity in the recent years and developing more efficient state-of-the-art deep neural network architectures has been a subject of significant attention. The first working DNN architecture was published by Alexey G. Ivakhnenko and V. G. Lapa in 1967 for supervised, deep, feed-forward, multi-layer perceptions [1]. In 1971, Alexey G. Ivakhnenko introduced another DNN architecture with eight layers trained by multi-parametric datasets using the Group Method of Data Handling (GMDH) algorithm [2].

Fast forwarding to the 21<sup>st</sup> century, DNN has become an important part of Computer Vision which is a multidisciplinary scientific field of study that seeks to develop novel techniques to enable computers to perform complex tasks [3] such as

image classification and object detection, analogous to that of human visual system. Due to the digitization of our society, inexpensive cameras and Internet of Things (IoT), we now have access to a fairly large amount of data for multinomial classification.

### II. BACKGROUND

G. Huang *et al.* introduced ResNet in 2016 to facilitate the training of deep neural networks with copious amounts of data. This model introduces *identity shortcut connections* that skips one or more layers, a fundamental ground-breaking idea proposed by ResNet [3]. Although it is relatively an old DNN architecture, it is still used as a baseline due to its inventive impact.

In 2017, G. Huang *et al.* took the idea of residual connections even further and introduced a new DNN architecture called DenseNet in which each layer is connected to every other layer in a feed-forward fashion. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers [4]. This is an ingenious technique since it facilitates the reuse of channels and neurons (tensors) that represent information at different level of coarseness.

In 2018, G. Huang *et al.* introduced CondenseNet [10], an improvement over DenseNet. CondenseNet replaces the classical convolutions with learned grouped convolutions which is designed to allow the network to learn mapping between channels and groups instead of separating channels on the basis of their offsets which results in different groups of channels being trained separately. With recent developments in the area of embedded systems with constrained computational resources for autonomous vehicles [5], robotics [6] and unmanned ariel vehicles such as drones [7], a desire for more efficient yet accurate inferring DNN models has burgeoned in recent years [8], [9]. In this paper, we propose a deep neural network architecture with a trained model size of less than 3.0 MB to address this demand.

### III. PRIOR WORK

Prior work in the following areas have contributed to the research work presented within this paper:



- Convolutional neural networks [8], [11], [12], specifically CondenseNet [10], form the baseline and are similar to our proposed architecture in few aspects which will be discussed in the following sections.
- The DenseNet convolutional neural network architecture which first manifested the advantages of dense connectivity in which each layer is connected to every other layer in a feed-forward fashion [4]. This results in the reuse of channels and neurons (tensors) that represent information at different level of coarseness.
- Depthwise separable convolution, which was first seen in [13], is one of the main fundamentals of our proposed architecture. Although the idea of utilizing spatially separable convolutions in deep neural networks can be traced back to 2012 [14], the depthwise interpretation of spatially separable convolutions is comparatively new and has been proven to be widely successful in its implementation in MobileNet [15] and Xception [16] architectures.
- Model compression techniques such as [17]–[19] are designed to improve the efficiency of a neural network architecture. We choose to incorporate group-wise filter pruning [20] process into the training stage which allows the network to train from scratch with an unprecedented computational efficiency.

#### IV. CONDENSENEXT ARCHITECTURE

In this paper, we propose a novel CNN architecture inspired by the aforementioned prior works. We have named our neural network architecture *CondenseNeXt* referring to the *next* dimension of cardinality. In this section, we describe the architecture of our proposed neural network in detail.

##### A. Convolution Layers

CondenseNet utilizes learned group convolutions technique where inconsequential filters with low magnitude weights are pruned first and then the filters are learned after the groupings are finalized. Grouped convolutions form an underlying fundamental of CondenseNet which is a peculiar case of sparsely connected convolutional layers. Here, the input and output neurons are divided into independent groups of  $g$ . This enables the network to perform regular spatial convolution over each independent group and concatenate the resulting feature maps.

The goal of CondenseNeXt is to reduce the trained model size to less than 3.0 MB and also the amount of computational resources required to train the network from scratch. In order to achieve this, we first replace the grouped convolution with depthwise separable convolution in learned group convolution. Depthwise separable convolution is comprised of:

- Depthwise convolution (filtering) layer: Convolution to a single input channel is applied independently whereas in standard convolution, convolution is applied to all input channels.
- Pointwise convolution (combining) layer: A linear combination is carried out for each of these layers.

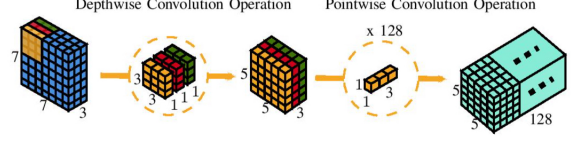


Figure 1. A 3D illustration of the overall process of depthwise separable convolution. A standard convolution will transform the image 128 times whereas a depthwise separable convolution will transform the image only once and then the transformed image is elongated to 128 channels which allows the neural network to process more data whilst utilizing fewer FLOPs resulting in an unprecedented computational efficiency.

The depthwise separable convolution splits a kernel into two discrete filters for filtering and combining stages respectively as shown in Figure 1 above, which reduces the computation required for training as well as size of the trained model.

Consider a standard convolutional filter  $K$  of size  $H \times H \times I \times O$  where  $I$  is the number of input channels and  $O$  is the number of output channels with an input feature map  $X$  of size  $D_x \times D_x \times I$  that produces an output feature map  $Y$  of size  $D_x \times D_x \times O$  can be computed as follows:

$$Y_{k,l,n} = \sum_{i,j,m} k_{i,j,m,n} \cdot X_{k+i-1,l+j-1,m} \quad (1)$$

For depthwise separable convolution, (1) can be factorized into two parts: the first part applies a  $3 \times 3$  depthwise convolution  $\hat{K}$  with one filter for each input channel as follows:

$$\hat{Y}_{k,l,m} = \sum_{i,j} \hat{K}_{i,j,m} \cdot X_{k+i-1,l+j-1,m} \quad (2)$$

And then the second part applies a  $1 \times 1$  pointwise convolution  $\tilde{K}$  in order to perform linear combination and combine the outputs of depthwise convolution as follows:

$$Y_{k,l,n} = \sum_m \tilde{K}_{m,n} \cdot \hat{Y}_{k-1,l-1,m} \quad (3)$$

This approach results in to more efficient computation requiring 7.3% fewer FLOPs and 11.5% fewer trainable parameters than the original grouped convolutions at the cost of 1.17% loss in Top-1 accuracy. Depthwise separable convolutions have been proven in reducing the trained model size and computational resources, both in theory and in practice [15], [16].

The second half of the training utilizes a widely used model compression technique which has further significant impact, both in computational efficiency at training time and on the final trained model size. We present details on our approach below.

##### B. Model Compression

Model compression is one the many popular methods of reducing complexity of a DNN to make it more computationally efficient by discarding redundant elements that do not influence performance of the model.



**Group-wise Pruning:** The goal of group-wise pruning is to discard inconsequential filters for every group  $g$  during the training stage which is determined by the  $L_1$ -Normalization of  $X^{g_{ij}}$  where for every group  $g$ ,  $i$  is the input and  $j$  is the output. We introduce pruning hyper-parameter  $p$  and set it to 4 which allows the network to determine the number of filters to be removed before the first stage of depthwise separable convolution during the training process. We also add a class balanced focal loss function [21] to smoothen the pruning process.

**Cardinality:** We introduce a new dimension to the network called *Cardinality* denoted by  $C$  in addition to the existing depth and width of the network in order to minimize the loss in accuracy during the pruning process. We set the the cardinality  $C$  equal to the number of output channels because the number was found to be very small for division. Experiments demonstrate that increasing cardinality is a more effective way of gaining accuracy than going deeper or wider, especially when depth and width starts to give diminishing returns [22].

Consider a group convolution comprised of  $G$  groups of size  $H \times H \times C_I \times C_O$  where  $C_I = \frac{I}{G}$  and  $C_O = \frac{O}{G}$ , the total number of inconsequential filters that will be pruned before the first stage of depthwise separable convolution during the training process can be determined as follows:

$$G \cdot C_x = I \cdot C - p \cdot I \quad (4)$$

This approach further results in a significant decrease in computational resources requiring 60.99% fewer FLOPs and 65.21% fewer trainable parameters at the cost of 0.88% loss in Top-1 accuracy for CIFAR-10 dataset.

### C. Activation Function

Activation functions in deep neural networks determine the output of a neuron at a given input or a set of inputs by limiting the amplitude of the output. It helps the network understand and learn complex patterns of the input data. Furthermore, non-linear activation functions such as ReLU (Rectified Linear Units) and ReLU6 [23] enable such networks to perform non-trivial computations with the help of fewer neurons.

Our approach utilizes ReLU6 activation function along with Batch Normalization before each layer. In ReLU6, units are capped at 6 which promotes an earlier learning of sparse features. Furthermore, it prevents the explosion of positive gradients to infinity. ReLU6 is defined as follows:

$$f(x) = \min(\max(0, x), 6) \quad (5)$$

Our experiments show that this approach was able to achieve a small gain of 0.577% in Top-1 accuracy for CIFAR-10 dataset after implementing the model compression technique as described earlier in this section.

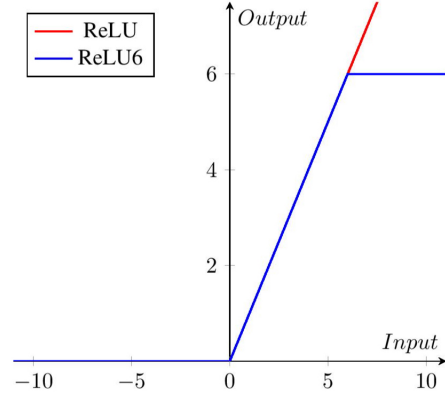


Figure 2. Graph demonstrating the difference between ReLU and ReLU6 activation functions.

## V. TRAINING INFRASTRUCTURE

- Intel Xeon Gold 6126 12-core CPU with 32 GB RAM.
- NVIDIA Tesla V100 GPU.
- CUDA Toolkit 10.1.243.
- Python version 3.7.9.
- PyTorch version 1.1.0.
- PyCharm Professional IDE version 2019.3.5.

This cyberinfrastructure is provided and managed by the Research Technologies division at the Indiana University which supported our work in part by Shared University Research grants from IBM Inc. to Indiana University and Lilly Endowment Inc. through its support for the Indiana University Pervasive Technology Institute [24].

## VI. EXPERIMENT AND RESULTS

Training results presented in this report are based on the evaluation of our proposed CondenseNeXt architecture on CIFAR-10 image classification dataset.

**Training Details:** Our network was implemented using the PyTorch framework and trained on NVIDIA's Tesla V100 GPU with Nesterov Momentum Weight of 0.9 and Stochastic Gradient Descent (SGD) for 200 epochs. Furthermore, the training utilizes cosine shape learning rate and dropout rate of 0.1.

**Dataset:** CIFAR-10 dataset [23], [25], first introduced by Alex Krizhevsky in [26], is one of the most popular datasets used in the field of machine learning research. It comprises of 60,000 RGB images of 10 different classes of size  $32 \times 32$  pixels split into two sets of 50,000 for training and 10,000 for testing. We choose to use standard data augmentation scheme [27] successfully implemented in the baseline architecture.

Table I  
COMPARISON OF PERFORMANCE

Architecture	FLOPs (in millions)	Parameters (in millions)	CIFAR-10 Top-1 Accuracy	Trained Model Size
CondenseNet (Baseline Architecture)	65.81	0.52	94.24%	16.7 MB
CondenseNeXt (Our Proposed Architecture)	23.80	0.16	92.28%	2.9 MB

Table I provides a comparison between CondenseNet (the baseline architecture) vs. CondenseNeXt (the proposed architecture in this paper) in terms of performance for 200 epochs each utilizing the training infrastructure as outlined in section 5 in this paper.

Table II  
COMPARISON OF TRAINING TIME

Architecture	Total Training Time (HH:MM:SS)	Training Time Per Epoch (in seconds)
CondenseNet	02:59:19	53.79
CondenseNeXt	01:04:48	19.42

Table II provides a comparison of training time between CondenseNet (the baseline architecture) vs. CondenseNeXt (the proposed architecture in this paper) for 200 epochs each, utilizing the training infrastructure and setup as outlined in sections 5 and 6 in this paper.

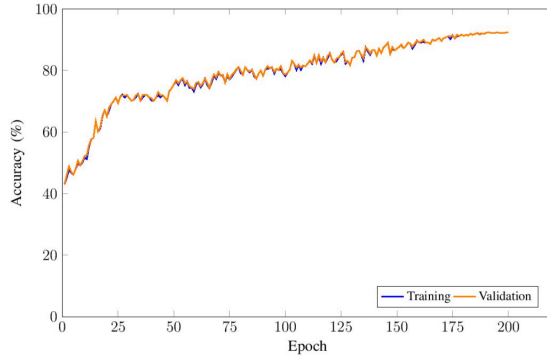


Figure 3. Training Overview of CondenseNeXt on CIFAR-10 dataset based on the training infrastructure and setup outlined in sections 5 and 6 of this paper.

**Training Results:** Our model was evaluated with a single crop of inputs on CIFAR-10 dataset for 200 epochs. Table I provides a comparison between the baseline architecture and our proposed architecture in terms of FLOPs, parameters, Top-1 accuracy and trained model size. Table II provides a comparison of training time between the two architectures and figure 3 plots the accuracy of our proposed model. The Top-1 Accuracy of our proposed architecture is 92.28% with a trained model size of 2.9 MB.

## VII. CONCLUSION

In this paper, we introduce CondenseNeXt: an ultra-efficient convolutional neural network architecture for embedded systems that ameliorates the performance of baseline convolutional neural network architecture, CondenseNet, by utilizing depthwise separable convolution and model compression techniques to significantly reduce computational resources required for training from scratch without a pre-trained model. The primary goal of our work presented in this paper is to

reduce the trained model size to less than 3.0 MB so that it can be deployed for real-time inference on embedded systems such as [5], [6] with constrained computational resources.

Our work presented in this paper also coincides with the trend that started with Xception and MobileNets that bespeak that any traditional convolution can be replaced by depthwise separable convolution to obtain a trained model that is relatively small in size and efficient in terms of FLOPs which is corroborated by strong theoretical foundations as well as experimental results. We hope that our work further bolsters this trend. Furthermore, we also anticipate the use of depthwise separable convolutions into the design of new deep neural network architectures in the future. As desire and demand for smaller yet efficient deep neural network architectures increase with innovations in, including but not limited to, embedded systems with limited computational resources, the scope for research and innovation in this field is immensurable.

## ACKNOWLEDGMENT

The authors acknowledge the Indiana University Pervasive Technology Institute for providing supercomputing and storage resources that have contributed to the research results reported within this paper.

## REFERENCES

- [1] A. G. Ivakhnenko and V. G. Lapa, *Cybernetics and Forecasting Techniques*. American Elsevier Publishing Company, 1967.
- [2] A. G. Ivakhnenko, "Polynomial Theory of Complex Systems," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-1, no. 4, pp. 364–378, Oct. 1971, doi: 10.1109/TSMC.1971.4308320.
- [3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 770–778, doi: 10.1109/CVPR.2016.90.
- [4] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 2261–2269, doi: 10.1109/CVPR.2017.243.
- [5] "NXP BlueBox: Autonomous Driving Development Platform," NXP. [Online]. Available: <https://www.nxp.com/design/development-boards/automotive-development-platforms/nxp-bluebox-autonomous-driving-development-platform:BLBX> [Accessed: 23-Dec-2020].
- [6] "i.MX Applications Processors," NXP. [Online]. Available: <https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i-mx-applications-processors:IMX> [Accessed: 23-Dec-2020].
- [7] R. Chitanvis, N. Ravi, T. Zantye, and M. El-Sharkawy, "Collision avoidance and Drone surveillance using Thread protocol in V2V and V2I communications," in *2019 IEEE National Aerospace and Electronics Conference (NAECON)*, Jul. 2019, pp. 406–411, doi: 10.1109/NAECON46414.2019.9058170.
- [8] A. Lotfi, H. Bouchachia, A. Gegov, C. Langensiepen, and M. McGinnity, Eds., *Advances in Computational Intelligence Systems: Contributions Presented at the 18th UK Workshop on Computational Intelligence*, September 5-7, 2018, Nottingham, UK. Springer International Publishing, 2019.

- [9] S. O. Haykin, *Neural networks and learning machines*, 3rd ed. New York: Prentice Hall, 2009.
- [10] G. Huang, S. Liu, L. V. D. Maaten, and K. Q. Weinberger, "CondenseNet: An Efficient DenseNet Using Learned Group Convolutions," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2018, pp. 2752–2761, doi: 10.1109/CVPR.2018.00291.
- [11] Y. LeCun et al., "LEARNING ALGORITHMS FOR CLASSIFICATION: A COMPARISON ON HANDWRITTEN DIGIT RECOGNITION," p. 16.
- [12] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," arXiv:1311.2901 [cs], Nov. 2013, Accessed: Nov. 26, 2020. [Online]. Available: <http://arxiv.org/abs/1311.2901>.
- [13] L. Sifre and P. S. Mallat, *Ecole Polytechnique, CMAP PhD thesis Rigid-Motion Scattering For Image Classification Author*: 2014.
- [14] F. Mamalet and C. Garcia, "Simplifying ConvNets for Fast Learning," in *Artificial Neural Networks and Machine Learning – ICANN 2012*, vol. 7553, A. E. P. Villa, W. Duch, P. Erdi, F. Masulli, and G. Palm, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 58–65.
- [15] A. G. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv:1704.04861 [cs], Apr. 2017, [Online]. Available: <http://arxiv.org/abs/1704.04861>.
- [16] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 1800–1807, doi: 10.1109/CVPR.2017.195.
- [17] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A Survey of Model Compression and Acceleration for Deep Neural Networks," arXiv:1710.09282 [cs], Jun. 2020, [Online]. Available: <http://arxiv.org/abs/1710.09282>.
- [18] V. Lebedev and V. Lempitsky, "Fast ConvNets Using Group-Wise Brain Damage," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 2554–2564, doi: 10.1109/CVPR.2016.280.
- [19] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression," in *2017 IEEE International Conference on Computer Vision (ICCV)*, Venice, Oct. 2017, pp. 5068–5076, doi: 10.1109/ICCV.2017.541.
- [20] N. Yu, S. Qiu, X. Hu, and J. Li, "Accelerating convolutional neural networks by group-wise 2D- filter pruning," in *2017 International Joint Conference on Neural Networks (IJCNN)*, Anchorage, AK, USA, May 2017, pp. 2502–2509, doi: 10.1109/IJCNN.2017.7966160.
- [21] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, and S. Belongie, "Class-Balanced Loss Based on Effective Number of Samples," p. 10.
- [22] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated Residual Transformations for Deep Neural Networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 5987–5995, doi: 10.1109/CVPR.2017.634.
- [23] A. Krizhevsky, "Convolutional Deep Belief Networks on CIFAR-10," p. 9.
- [24] C. A. Stewart, V. Welch, B. Plale, G. Fox, M. Pierce, and T. Sterling, "Indiana University Pervasive Technology Institute," Sep. 2017, doi: 10.5967/K8G44NGB.
- [25] R. Doon, T. K. Rawat, and S. Gautam, "Cifar-10 Classification using Deep Convolutional Neural Network," in *2018 IEEE Punecon*, Nov. 2018, pp. 1–5, doi: 10.1109/PUNECON.2018.8745428.
- [26] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," p. 60.
- [27] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *J Big Data*, vol. 6, no. 1, p. 60, Jul. 2019, doi: 10.1186/s40537-019-0197-0.



Published as: P. Kalgaonkar and M. El-Sharkawy, “Image Classification with CondenseNeXt for ARM-Based Computing Platforms”, *2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, ON, Canada, 2021, pp. 0789-0794, DOI: [10.1109/IEMTRONICS52119.2021.9422541](https://doi.org/10.1109/IEMTRONICS52119.2021.9422541).

# Image Classification with CondenseNeXt for ARM-Based Computing Platforms

Priyank Kalgaonkar

Department of Electrical and Computer Engineering  
Purdue School of Engineering and Technology  
Indianapolis, Indiana 46202, USA.  
pkalgaon@purdue.edu

Mohamed El-Sharkawy

Department of Electrical and Computer Engineering  
Purdue School of Engineering and Technology  
Indianapolis, Indiana 46202, USA.  
melshark@purdue.edu

**Abstract**—In this paper, we demonstrate the implementation of our ultra-efficient deep convolutional neural network architecture: CondenseNeXt on NXP BlueBox, an autonomous driving development platform developed for self-driving vehicles. We show that CondenseNeXt is remarkably efficient in terms of FLOPs, designed for ARM-based embedded computing platforms with limited computational resources and can perform image classification without the need of a CUDA enabled GPU. CondenseNeXt utilizes the state-of-the-art depthwise separable convolution and model compression techniques to achieve a remarkable computational efficiency.

Extensive analyses are conducted on CIFAR-10, CIFAR-100 and ImageNet datasets to verify the performance of CondenseNeXt Convolutional Neural Network (CNN) architecture. It achieves state-of-the-art image classification performance on three benchmark datasets including CIFAR-10 (4.79% top-1 error), CIFAR-100 (21.98% top-1 error) and ImageNet (7.91% single model, single crop top-5 error). CondenseNeXt achieves final trained model size improvement of 2.9+ MB and up to 59.98% reduction in forward FLOPs compared to CondenseNet and can perform image classification on ARM-Based computing platforms without needing a CUDA enabled GPU support, with outstanding efficiency.

**Index Terms**—CondenseNeXt, Convolutional Neural Network, Computer Vision, Image Classification, NXP BlueBox, ARM, Embedded Systems, PyTorch, CIFAR-10, CIFAR-100, ImageNet.

## I. INTRODUCTION

ARM processors are widely used in electronic devices such as smartphones and tablets as well as in embedded computing platforms such as the NXP BlueBox, Nvidia Jetson and Raspberry Pi for computer vision purposes. ARM is RISC (Reduced Instruction Set Computing) based architecture for computer processors which results in low costs, minimal power consumption, and lower heat generation compared to its competitor: CISC (Complex Instruction Set Computing) architecture based processors such as the Intel x86 processor family. As of 2021, over 180 billion ARM-based chips have been manufactured and shipped by Arm and its partners around the globe which makes it the most popular choice of Instruction Set Architecture (ISA) in the world [1].

The roots of ARM processors trace back to December 1981 when the first widely successful design, BBC Micro (British Broadcasting Corporation Microcomputer System), was introduced by Acorn Computers [2]. Due to the use of DRAM (Dynamic Random Access Memory) in its design, it outperformed nearly twice as that of Apple II, an 8-bit personal computer, which was world’s first successfully mass-produced publicly available computer designed by Steve Wozniak, Steve Jobs and Rod Holt in June 1977 [3].

Fast forwarding to the 21<sup>st</sup> century, due to constant advances in computing and VLSI technology, ARM-based chips are found in nearly 60% of all mobile devices and computing platforms produced today. With processor performance doubling approximately every two years with a focus on parallel computing technologies such as multi-core processors, computer vision researchers can now implement sophisticated neural network algorithms to perform complex computations for OpenCV applications without a requiring a GPU support.

Convolutional Neural Networks (CNN), a class of Deep Neural Networks (DNN) first introduced by Alexey G. Ivakhnenko and V. G. Lapa in 1967 [4], have been gaining popularity in recent years as researchers focus on creating more advanced intelligent systems. CNNs are popularly used in machine (computer) vision applications such as image classification, image segmentation, object detection, etc. However, implementing a CNN on embedded systems with constrained computational resources for applications such as autonomous cars, robotics and unmanned aerial vehicle (UAV), commonly known as a drone, is a challenging task. In this paper, we present image classification performance results of CondenseNeXt CNN on NXP BlueBox, an ARM-based embedded computing platform for automotive applications.

## II. RELATED WORK

Following work has contributed to the research and implementation results presented within this paper:

**CondenseNeXt:** An ultra-efficient deep convolutional neural network for embedded systems, introduced by P. Kalgaonkar and M. El-Sharkawy in January 2021 [5] has been utilized to train and evaluate image classification performance on three benchmarking datasets: CIFAR-10, CIFAR-100 and ImageNet.

### III. NXP BLUEBOX 2.0

The BlueBox 2 family developed and manufactured by NXP Semiconductors N.V, a Dutch-American semiconductor manufacturer with headquarters in Eindhoven, Netherlands and Austin, United States of America, is a Automotive High Performance Compute (AHPC) platform that provides essential performance and reliability for engineers to develop sensor fusion, automated drive and motion planning applications along with functional safety, vision acceleration and automotive interfaces for self-driving (autonomous) vehicles.

NXP BlueBox Gen1 was first introduced in May 2016 at the 2016 NXP FTF Technology Forum held in Austin, Texas, USA. This opened avenues to a host of autonomous and sensor fusion applications. Shortly after, NXP introduced BlueBox Gen2 (BlueBox 2.0), a significant improvement over Gen1, incorporating three new processors: S32V234 ARM-based automotive computer vision processor, LS2084A high performance ARM-based compute processor and S32R274 ASIL-D RADAR microcontroller.

**S32V234:** The S32V234 automotive computer vision processor comprises of a quad core ARM Cortex-A53 CPU running at 1.0 GHz paired with a ARM Cortex-M4 functional safety core which utilizes the ARMv8-A 64-bit instruction set developed by ARM Holdings' Cambridge design centre. It has a 4MB internal SRAM in addition to a 32bit LPDDR3 memory controller for external memory support. It is an on-chip Image Signal Processor (ISP) designed to meet ASIL-B/C automotive safety standards and optimized for obtaining maximum performance per watt efficiency.

**LS2084A:** The LS2084A high performance compute processor comprises of an octa core ARM Cortex-A72 CPU running at 1.8 GHz which utilizes the ARMv8-A 64-bit instruction set developed by ARM Holdings' Austin design centre. It has two 72 bytes DDR4 RAMs running at up to 28.8GB/s memory bandwidth. The LS2 provides software compatibility with next generation LayerScape LX2 family and offers AEC Q100 Grade 3 reliability with 15 years product longevity.

**S32R274:** The S32R274 radar micro-controller comprises of a dual core Freescale PowerPC e200z7 32-bit CPU running at 240 MHz and a dual core Freescale PowerPC e200z4 32-bit CPU running at 120 MHz with an additional checker core. It has a 2 MB Flash and 1.5 MB SRAM for radar application storage, message buffering and radar data stream handling. The S32R processor is optimized for on-chip radar signal processing to maximize performance per watt efficiency. It has been designed by NXP to meet the ASIL-D automotive applications standards.

### IV. RTMAPS REMOTE STUDIO SOFTWARE

RTMaps (Real-Time Multisensor applications) developed by Intempora is a powerful GUI software that aids in development of applications for advanced driver assistance systems, autonomous driving and robotics. It helps in capturing, processing and viewing data from multiple sensors and offers



Figure 1. NXP BlueBox 2.0 ARM-based Automotive High Performance Compute (AHPC) embedded development platform. It delivers necessary prerequisites to help develop high-performance computing systems, analyze driving environments, assess risk factors, and then direct the car's behavior. BlueBox 2.0 also supports OpenCV applications using an external camera for real-time image classification object detection and image segmentation.

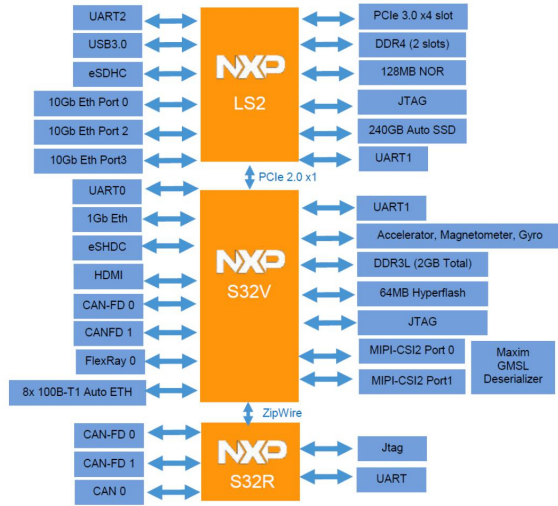


Figure 2. High-Level View of NXP Bluebox 2.0 Gen2 Architecture [6]. S32V processor utilizes two CAN-FD (Flexible Data Rate) with enhanced payload and data rate, PCIe, Ethernet, FlexRay, Zipwire, one SAR-ADCs, four SPI and one SD card connectivity. LS2 processor utilizes two DUART, four I2C, SPI, GPIO and two USB 3.0 interfaces. S32R processor utilizes JTAG, UART, three FlexCAN and Zipwire to connect to a radar ASIC.

a multi-modular development and run-time environment for ARM-based computing platforms such as the NXP BlueBox 2.0. This data can also be reviewed and play-backed at a later time for offline development and testing purposes.

RTMaps Remote Studio supports PyTorch, an open-source machine learning library based upon the Torch library, widely used for real-time computer vision (OpenCV) development. Algorithms for OpenCV can be developed using Python scripting language and by the means of block diagrams. It also facilitates the development of algorithms directly on to any supported embedded system without having to connect external user interfacing peripheral devices.



## V. CONDENSENEXT

CondenseNeXt is an ultra-efficient deep convolutional neural network architecture designed for embedded systems introduced by P. Kalgaonkar and M. El-Sharkawy in January 2021. CondenseNeXt refers to the *next* dimension of cardinality. In this section, we describe in detail the architecture of this neural network that has been utilized to train and evaluate image classification performance on three benchmarking datasets: CIFAR-10, CIFAR-100 and ImageNet.

### A. Convolution Layers

One of the main goals of CondenseNeXt is to reduce the amount of computational resources required to train the network from scratch and for real-time inference on embedded systems with limited computational resources. Following state-of-the-art technique has been incorporated into the design of this CNN:

- **Depthwise convolution layer:** It acts like a filtering layer where convolution to a single input channel is applied separately instead of applying it to all input channels. Assume there is an input data of size  $A \times A \times C$  and filters (kernels)  $K$  of size  $F \times F \times 1$ . If there are  $C$  number of channels in the input data, the output will be of size  $B \times B \times C$ . At this point, the spatial dimensions have shrunk. However, the depth  $C$  has remained constant and the cost of this operation will be  $B^2 \times F^2 \times C$ .
- **Pointwise convolution layer:** It acts like a combining layer where a linear combination is carried out for each of these layers. At this stage, a  $1 \times 1$  convolution is applied to  $C$  number of channels in the input data. Thus, the size of filter for this operation will be  $1 \times 1 \times C$  and size of the output will be  $B \times B \times D$  for  $D$  such filters.

Assume a standard convolutional filter  $K$  of size  $F \times F \times A \times B$  where  $A$  is the number of input channels and  $B$  is the number of output channels with an input feature map  $A$  of size  $D_x \times D_x \times A$  that produces an output feature map  $Z$  of size  $D_y \times D_y \times B$  can be mathematically represented as follows:

$$Z_{k,l,n} = \sum_{i,j,m} k_{i,j,m,n} \cdot A_{k+i-1,l+j-1,m} \quad (1)$$

In case of a depthwise separable convolution, (1) is factorized into two stages: the first stage applies a  $3 \times 3$  depthwise convolution  $\hat{K}$  with one filter for every input channel:

$$\hat{Z}_{k,l,m} = \sum_{i,j} \hat{K}_{i,j,m} \cdot A_{k+i-1,l+j-1,m} \quad (2)$$

Consequently, in the second stage, a  $1 \times 1$  pointwise convolution  $\tilde{K}$  is applied to carry out linear combination and combine the outputs of depthwise convolution from previous stage as follows:

$$Z_{k,l,n} = \sum_m \tilde{K}_{m,n} \cdot \hat{Z}_{k-1,l-1,m} \quad (3)$$

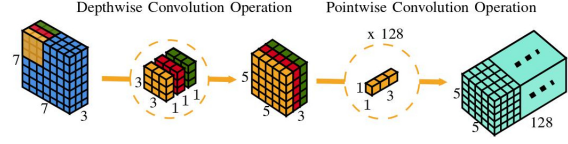


Figure 3. A 3D illustration of the overall process of depthwise separable convolution. An image is transformed 128 times whereas an image is transformed by depthwise separable convolution only once and then this transformed image is stretched to 128 channels which allows the neural network to process more data while consuming fewer FLOPs (Floating Point Operations).

This methodology splits a kernel into two discrete filters for filtering and combining stages as shown in Figure 3 above, which results in reduction of computational resources required to train the network from scratch as well for real-time inference.

A widely used model compression technique is also implemented into the design of this CNN where a further significant impact, both in computational efficiency at training time and on the final trained model size is seen.

### B. Model Compression

A widely popular model compression technique called Group-wise Pruning is implemented to make CondenseNeXt neural network computationally more efficient by discarding redundant elements without influencing the overall performance of the network.

**Group-wise Pruning:** The purpose of group-wise pruning is to remove trivial filters for every group  $g$  during the training process which is based on the  $L_1$ -Normalization of  $A^{g_{ij}}$  where for every group  $g$ ,  $a$  is the input and  $z$  is the output. A pruning hyper-parameter  $p$  is established and set to 4 which allows the network to decide the number of filters to remove before the first stage of depthwise separable convolution. A class balanced focal loss function [7] is also added to assist and ease the effect of this pruning process.

Consider a group convolution comprised of  $G$  groups of size  $F \times F \times C_A \times C_B$  where  $C_A = \frac{A}{G}$  and  $C_B = \frac{B}{G}$ . The total number of trivial filters that will be pruned before the first stage of depthwise separable convolution is mathematically represented as follows:

$$G \cdot C_x = A \cdot C - p \cdot A \quad (4)$$

**Cardinality:** A new dimension to the network called *Cardinality* denoted by  $C$  is incorporated into the design of CondenseNeXt neural network in addition to the existing width and depth dimensions so that loss in accuracy during the pruning process is reduced. Experiments prove that increasing cardinality is a more efficacious way of accruing accuracy than going deeper or wider, especially when width and depth starts to provide diminishing returns [8].

### C. Activation Function

In deep neural networks, activation functions determine the output of a neuron at particular input(s) by restricting the amplitude of the output. It aids in neural network's understanding and learning process of complex patterns of the input data. Furthermore, non-linear activation functions such as ReLU6 (Rectified Linear Units capped at 6) enable neural networks to perform complex computations using fewer neurons [9].

CondenseNeXt applies ReLU6 activation function in addition to Batch Normalization technique prior to each convolutional layer. In ReLU6, units are capped at 6 to promote an earlier learning of sparse features and to prevent a sudden blowup of positive gradients to infinity. ReLU6 activation function is defined mathematically as follows:

$$f(x) = \min(\max(0, x), 6) \quad (5)$$

## VI. CYBERINFRASTRUCTURE

### A. Training Infrastructure

- Intel Xeon Gold 6126 12-core CPU with 128 GB RAM.
- NVIDIA Tesla V100 GPU.
- CUDA Toolkit 10.1.243.
- PyTorch version 1.1.0.
- Python version 3.7.9.

This cyberinfrastructure for training is provided and managed by the Research Technologies division at the Indiana University which supported our work in part by Shared University Research grants from IBM Inc. to Indiana University and Lilly Endowment Inc. through its support for the Indiana University Pervasive Technology Institute [10].

### B. Testing Infrastructure

- NXP BlueBox 2.0 ARM-based autonomous embedded development platform.
- Intempora RTMaps Remote Studio version 4.8.0.
- CIFAR-10, CIFAR-100 and ImageNet Datasets.
- PyTorch version 1.1.0.
- Python version 3.7.9.

## VII. EXPERIMENT AND RESULTS

Training results presented in this report are based on the evaluation of image classification performance of CondenseNeXt CNN on three benchmarking datasets: CIFAR-10, CIFAR-100 and ImageNet. CondenseNeXt was designed and developed in PyTorch framework and trained on NVIDIA's Tesla V100 GPU with standard data augmentation scheme [11], Nesterov Momentum Weight of 0.9, Stochastic Gradient Descent (SGD), cosine shape learning rate and dropout rate of 0.1 for all three datasets discussed in this section.

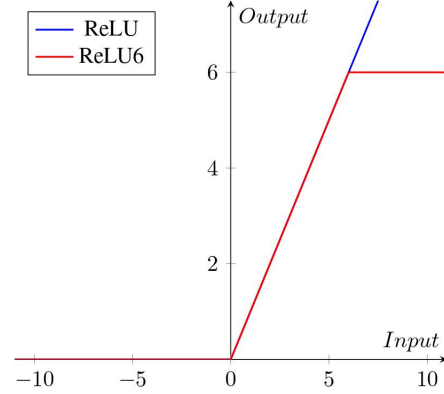


Figure 4. Difference between ReLU and ReLU6 activation functions.

### A. CIFAR-10 Classification

CIFAR-10 dataset [9], [12] was first introduced by Alex Krizhevsky in [13]. It is one of the most widely used datasets for evaluating a CNN in the field of deep learning research. There are 60,000 RGB images of 10 different classes of size 32×32 pixels divided into two sets of 50,000 for training and 10,000 for testing.

CondenseNeXt was trained with a single crop of inputs on CIFAR-10 dataset for 200 epochs, batch size of 64 and features  $k$  of 8-16-32. Using RTMaps Remote Studio, an image classification script was developed using Python scripting language and evaluated on NXP BlueBox for single image classification analysis. Table I provides a comparison of performance between CondenseNet and CondenseNeXt CNN in terms of FLOPs, parameters, and Top-1 and Top-5 error rates. Figure 5 provides a screenshot of the RTMaps console.

### B. CIFAR-100 Classification

CIFAR-100 dataset was also first introduced by Alex Krizhevsky in [13] along side CIFAR-10 dataset. It is also one of the many popular choices of datasets in the field of deep learning research. Just like CIFAR-10 dataset, there are 60,000 RGB images in total. However, it has 100 different classes, where each class contains 600 images of size 32×32 pixels divided into two sets of 50,000 for training and 10,000 for testing. CIFAR-100 classes are mutually exclusive of CIFAR-10 classes. For example, CIFAR-100's baby, chimpanzee and rocket classes are not part of the CIFAR-10 classes.

CondenseNeXt was trained with a single crop of inputs on CIFAR-100 dataset for 600 epochs, batch size of 64 and features  $k$  of 8-16-32. Using RTMaps Remote Studio, an image classification script was developed using Python scripting language and evaluated on NXP BlueBox for single image classification analysis. Table I provides a comparison of performance between CondenseNet and CondenseNeXt CNN in terms of FLOPs, parameters, and Top-1 and Top-5 error rates. Figure 6 provides a screenshot of the RTMaps console.

Table I  
COMPARISON OF PERFORMANCE

Dataset	CNN Architecture	FLOPs (in millions)	Parameters (in millions)	Top-1 % Error	Top-5 % Error
CIFAR-10	CondenseNet	65.81	0.52	5.31	0.24
	CondenseNeXt	<b>26.35</b>	<b>0.18</b>	<b>4.79</b>	<b>0.15</b>
CIFAR-100	CondenseNet	65.85	0.55	23.35	6.56
	CondenseNeXt	<b>26.38</b>	<b>0.22</b>	<b>21.98</b>	<b>6.29</b>
ImageNet	CondenseNet	529.36	4.81	26.2	8.30
	CondenseNeXt	<b>273.16</b>	<b>3.07</b>	<b>25.8</b>	<b>7.91</b>

Table I provides a comparison between CondenseNet (the baseline architecture) vs. CondenseNeXt (our ultra-efficient deep neural network architecture) in terms of performance each utilizing the training setup and infrastructure as outlined in section 6 and 7 in this paper.

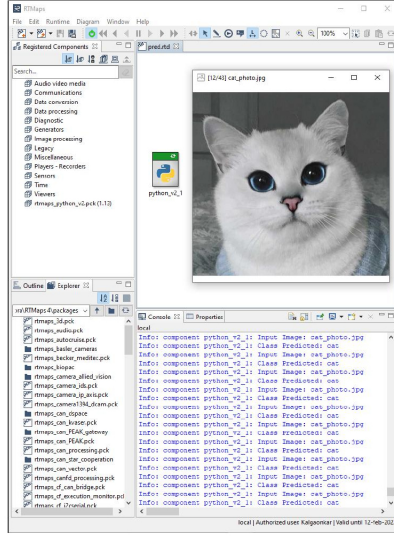


Figure 5. Evaluation of CondenseNeXt on CIFAR-10 dataset when deployed on NXP BlueBox 2.0 using RTMaps Remote Studio 4.8.0 for classifying an image of a cat and outputting the predicted class in RTMaps console.

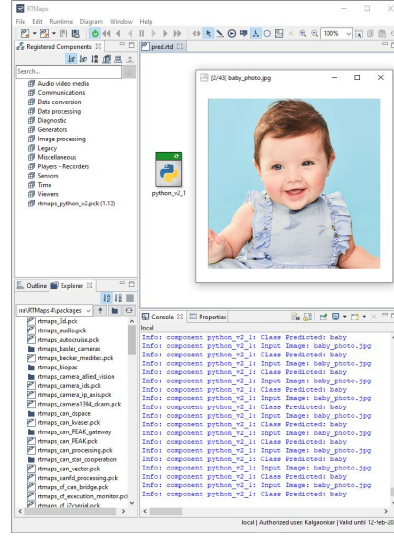


Figure 6. Evaluation of CondenseNeXt on CIFAR-100 dataset when deployed on NXP BlueBox 2.0 using RTMaps Remote Studio 4.8.0 for classifying an image of a baby and outputting the predicted class in RTMaps console.

### C. ImageNet Classification

ImageNet was introduced by an AI researcher Dr. Fei-Fei Li along with a team of researchers at a 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) in Florida [14]. This dataset is built according to the WordNet hierarchy where each node in the hierarchy corresponds to over five hundred images. In total, there are over 14 million images in this dataset that have been hand-annotated and labelled by the team.

CondenseNeXt was trained with a single crop of inputs on the entire ImageNet dataset for 120 epochs with a Group Lasso rate of 0.00001, batch size of 256, features of 8-16-32-64-128 and four Nvidia V100 GPUs using Data Parallelism technique. An image classification script was developed in RTMaps Remote Studio and evaluated on NXP BlueBox for single image classification analysis. Table I provides a comparison of performance between CondenseNet and CondenseNeXt CNN in terms of FLOPs, parameters, and Top-1 and Top-5 error rates. Figure 7 provides a screenshot of the RTMaps console.

### VIII. CONCLUSION

In this paper, we demonstrate the performance of CondenseNeXt CNN which is an ultra-efficient deep convolutional neural network architecture for ARM-based embedded computing platforms without CUDA enabled GPU(s). Extensive training from scratch and analysis have been conducted on three benchmarking datasets: CIFAR-10, CIFAR-100 and ImageNet. It achieves state-of-the-art image classification performance on CIFAR-10 dataset with a 4.79% Top-1 error rate, on CIFAR-100 dataset with a 21.98% Top-1 error rate and ImageNet dataset with a 7.91% single model and single crop Top-5 error rate. Our experiments on NXP's BlueBox further validate the effective use Depthwise Separable Convolutional layers and Model Compression techniques implemented to discard inconsequential elements and to reduce FLOPs without affecting overall performance of the neural network. In the future, we will explore different applications with CondenseNeXt such as image segmentation and object detection to better exploit different opportunities for OpenCV applications.



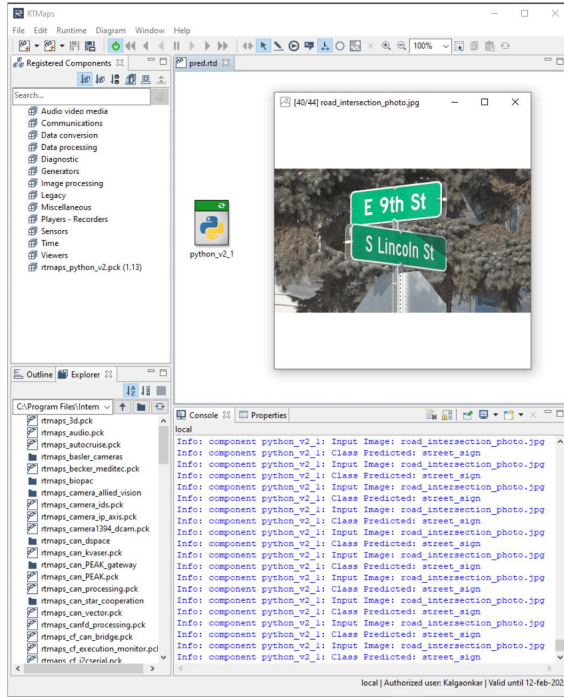


Figure 7. Evaluation of CondenseNeXt on ImageNet dataset when deployed on NXP BlueBox 2.0 using RTMaps Remote Studio version 4.8.0 for classifying an image of a street sign and outputting the predicted class in the RTMaps console.

#### ACKNOWLEDGMENT

The authors would like to acknowledge the support of Indiana University Pervasive Technology Institute for providing supercomputing and storage resources that have contributed to all research results reported within this paper.

#### REFERENCES

- [1] A. Ltd, "The Arm ecosystem ships a record 6.7 billion Arm-based chips in a single quarter," Arm — The Architecture for the Digital World. <https://www.arm.com/company/news/2021/02/arm-ecosystem-ships-record-6-billion-arm-based-chips-in-a-single-quarter> (accessed Mar. 01, 2021).
- [2] "History of ARM: from Acorn to Apple," *The Telegraph*, Jan. 06, 2011. <https://www.telegraph.co.uk/finance/newsbysector/epic/arm/8243162/History-of-ARM-from-Acorn-to-Apple.html> (accessed Feb. 25, 2021).
- [3] "Total share: 30 years of personal computer market share figures — Ars Technica," <https://arstechnica.com/features/2005/12/total-share/3/> (accessed Mar. 02, 2021).
- [4] A. G. Ivakhnenko and V. G. Lapa, *Cybernetics and Forecasting Techniques*. American Elsevier Publishing Company, 1967.
- [5] P. Kalgaonkar and M. El-Sharkawy, "CondenseNeXt: An Ultra-Efficient Deep Neural Network for Embedded Systems," in *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, pp. 559–563.
- [6] C. Cureton and M. Douglas, "Bluebox Deep Dive – NXP's AD Processing Platform," p. 28, Jun. 2019.
- [7] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, and S. Belongie, "Class-Balanced Loss Based on Effective Number of Samples," arXiv:1901.05555 [cs], Jan. 2019. [Online]. Available: <http://arxiv.org/abs/1901.05555>.
- [8] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated Residual Transformations for Deep Neural Networks," arXiv:1611.05431 [cs], Apr. 2017. [Online]. Available: <http://arxiv.org/abs/1611.05431>.
- [9] A. Krizhevsky, "Convolutional Deep Belief Networks on CIFAR-10," p. 9.
- [10] C. A. Stewart, V. Welch, B. Plale, G. Fox, M. Pierce, and T. Sterling, "Indiana University Pervasive Technology Institute," Sep. 2017, doi: 10.5967/K8G44NGB.
- [11] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *J Big Data*, vol. 6, no. 1, p. 60, Jul. 2019, doi: 10.1186/s40537-019-0197-0.
- [12] R. Doon, T. Rawat, and S. Gautam, "Cifar-10 Classification using Deep Convolutional Neural Network," Nov. 2018, pp. 1–5, doi: 10.1109/PUNECON.2018.8745428.
- [13] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," p. 60.
- [14] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, USA, Jun. 2009, pp. 248–255, doi: 10.1109/CVPR.2009.5206848.