COMPRESSED MOBILENET V3: AN EFFICIENT CNN FOR RESOURCE CONSTRAINED PLATFORMS

by

Kavyashree Prasad S P

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

May 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Mohamed El-Sharkawy, Chair

Department of Electrical and Computer Engineering

Dr. Brian King

Department of Electrical and Computer Engineering

Dr. Maher Rizkalla

Department of Electrical and Computer Engineering

Approved by:

Dr. Brian King

This Thesis is dedicated to my parents Shalini and Pradeep Prasad, and my late grandparents Shakuntala and Narendra Prasad. I would also like to thank my sister Divya Prasad, my friends, and IOT collaboratory for their constant support throughout my graduate studies.

ACKNOWLEDGMENTS

I want to express my sincere gratitude to my advisor, Dr. Mohammed El Sharkawy, for his guidance in this research. I would like to thank Dr. Brian King and Dr. Maher Rizkalla for being a part of my thesis committee and for extending their valuable feedback. I am also grateful to Sherrie Tucker for her assistance during my studies at IUPUI.

I am fortunate to have been a part of the IoT collaboratory, which provided me a multitude of resources to expand my knowledge. I am thankful to have met a lot of people here with similar interests and passions. Finally, I would like to thank my family for their support and motivation through tough times.

I would also like to thank Lilly Endowment Inc. for their support to Indiana University Pervasive Technology Institute.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

CNN	Convolutional Neural Network
DPD	Depthwise Pointwise Depthwise
NXP	Next Experience
MV	Machine Vision
SIFT	Scale-Invariant Feature Transform
SURF	Speeded Up Robost Features
HOG	Histogram of Oriented Gradients
FC	Fully Connected
RGB	Red Blue Green
CMYK	Cyan, Magenta, Yellow, and Key
HSV	Hue Saturation Value
GPU	Graphics Processing Unit
SDK	Software Development Kit
LCD	Liquid Crystal Display
TPU	Tensor Processing Unit
CPU	Central Processing Unit
DL	Deep Learning
NADAM	Nesterov-Accelerated Adaptive Moment Estimation
PWC	Pointwise Convolution
DWC	Depthwise Convolution

ABSTRACT

Computer Vision is a mathematical tool formulated to extend human vision to machines. This tool can perform various tasks such as object classification, object tracking, motion estimation, and image segmentation. These tasks find their use in many applications, namely robotics, self-driving cars, augmented reality, and mobile applications. However, opposed to the traditional technique of incorporating handcrafted features to understand images, convolution neural networks are being used to perform the same function.

Computer vision applications widely use CNNs due to their stellar performance in interpreting images. Over the years, there have been numerous advancements in machine learning, particularly to CNNs.However, the need to improve their accuracy, model size and complexity increased, making their deployment in restricted environments a challenge.

Many researchers proposed techniques to reduce the size of CNN while still retaining its accuracy. Few of these include network quantization, pruning, low rank, and sparse decomposition and knowledge distillation. Some methods developed efficient models from scratch. This thesis achieves a similar goal using design space exploration techniques on the latest variant of MobileNets, MobileNet V3. Using DPD blocks, escalation in the number of expansion filters in some layers and mish activation function MobileNet V3 is reduced to 84.96% in size and made 0.2% more accurate. Furthermore, it is deployed in NXP i.MX RT1060 for image classification on CIFAR-10 dataset.

1. INTRODUCTION

1.1 Context

Machine learning is becoming familiar with the hidden relationship in information and, accordingly, settling on choices without requiring guidance. Much of the literature on this domain has been accounted for to comprehend and copy human tactile reactions like speech, sight, and vision. In 1989, another class of Neural networks, called Convolutional neural networks, was introduced that showed tremendous potential in Machine Vision related undertakings [1].

Image classification is the undertaking of assigning an image into one of the several predetermined classes. It helps computers to understand and interpret images. It also forms the base of other computer vision functions such as object tracking, object detection, semantic segmentation, and pose estimation. Initially, these tasks were performed using handcrafted features, which act as descriptors of images, and then they were fed into a classification criterion. One of the significant disadvantages of this technique was the high dependency of accuracy on feature extraction techniques [2].

Over the years, deep learning models with many layers of non-linear transformations have overcome these challenges. CNN's are outstanding amongst other learning algorithms for understanding images and have shown commendable execution in image manipulation and recognition. They gained their popularity when AlexNet [3] won the ImageNet challenge in 2012. Since then a lot of architectures like SqueezeNet [4], ResNext [5], M-NAS Net [6],Inception [7] etc. were proposed which surpassed usage of other computer vision algorithms namely HOG features [8], SIFT [9],SURF [10] etc. The achievement of CNNs has caught consideration beyond the scholarly world. In industries like Google, AT&T,Microsoft, Facebook, and NEC, dynamic exploration teams were created for investigating new designs of CNN. The majority of the leaders of machine vision tasks are utilizing CNN-based models for major applications.

The alluring component of CNN is its capacity to abuse temporal and spatial correlation in raw images [11]. The architecture of CNNs has numerous stages aggregating convolutional layers, non-linear layers, and subsampling units. The convolution layer helps extract

correlated features from locally distributed feature space. The non-linearity creates various examples of initiations for various reactions, subsequently encouraging semantic contrasts in pictures. The subsampling layer is incorporated to make CNN invariant to geometric distortions. Hence, in this manner, CNN learns images without comprehensive human intervention for feature extraction.

The architectural design of CNNs is highly inspired by the human visual cortex. During the learning stages, CNN modifies weights using a backpropagation algorithm. This optimization to approach targets is similar to the brain's ability to learn based on responses. The multifaceted design of CNN allows for gathering high, low, and mid-level features from data. These high-level features are obtained from the consolidation of mid and low-level features. CNN's progressive feature extraction capacity imitates the profound and layered learning interaction of the Neocortex in the human cerebrum, which powerfully learns highlights from the crude information [11]. It is this ability that is responsible for the ubiquity of CNNs.

1.2 Motivation

CNNs are being improvised at a swift pace. They have exceeded the threshold for acceptance in many applications. They are now ubiquitous. In healthcare, they are used to diagnose diseases like breast cancer, pneumonia, diabetes, etc. In autonomous driving, CNNs are used in their perception stack to enable vehicles to navigate through lanes and obstacles, improving driver experience. In surveillance systems, CNNs are used to monitor locations to prevent trespassing, violence, and theft. Human knowledge in astronomy is proliferating; this calls for more data-driven tools for analysis and identification. In astronomy, CNNs are widely used for discovering new heavenly bodies. In agriculture, processes are automated using robots with computer vision algorithms. These robots help in harvesting, planting, irrigating, and weeding plants. In industrial applications, CNNs are used on assembly lines for the inspection and counting of products. Machine vision tools help in microscopic level inspection of defects that would not be possible with human vision. Computer vision is used

in satellites to detect natural calamities. They also help in analyzing air pollution in various areas of interest.

However over the years, to improve the precision of CNN's, their intricacy and number of layers have been expanded [12] [13] [14] [15]. This makes it challenging to deploy them in resource-constrained devices. Hence, there is a considerable demand for smaller-size CNNs. Smaller CNNs have numerous advantages. They are more feasible for embedded applications. They allow low latency and improved privacy, i.e., when computations are carried out in place rather than on the cloud, sensitive data can be protected. They also reduce computational load on devices. Smaller size CNNs also allow updating the newer version of the model by autonomous companies on customer vehicles from their servers more practical [16]. Hence, recognizing these gains, researchers have come up with novel algorithms and techniques to develop lightweight models. This thesis targets the same intention by significantly reducing size with no compromise to model accuracy.

1.3 Challenges

- Training baseline model from scratch
- Altering baseline architecture
- Training modified architecture from scratch
- Hyper-parameter tuning
- Achieving good trade-off between model size and accuracy
- Deployment of new architecture on i.MX RT 1060

1.4 Methodology

- Understanding baseline model
- Evaluating baseline architecture on CIFAR-10 dataset
- Reviewing various model compression techniques
- Modifying baseline model

- Deciding acceptable model size
- Tuning hyper parameters and improving accuracy
- Exploring hardware suitable for deployment
- Implementing image classification using new model on hardware

1.5 Contributions

- Proposed Compressed MobileNet V3
- Performed image classification on i.MX RT1060
- Published research paper titled "Compressed MobileNet V3:A Light Weight Variant for Resource-Constrained Platforms" in IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)
- Research paper titled "Deployment of Compressed MobileNet V3 on iMX RT 1060" accepted at IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS).

2. OVERVIEW OF CONCEPTS

This chapter explores convolution neural networks and their role in image classification. It then explains various stages in CNNs. It concludes with the description of MobileNet V3 and highlights some of the authors' improvements in this variant.

2.1 Convolutional Neural Networks

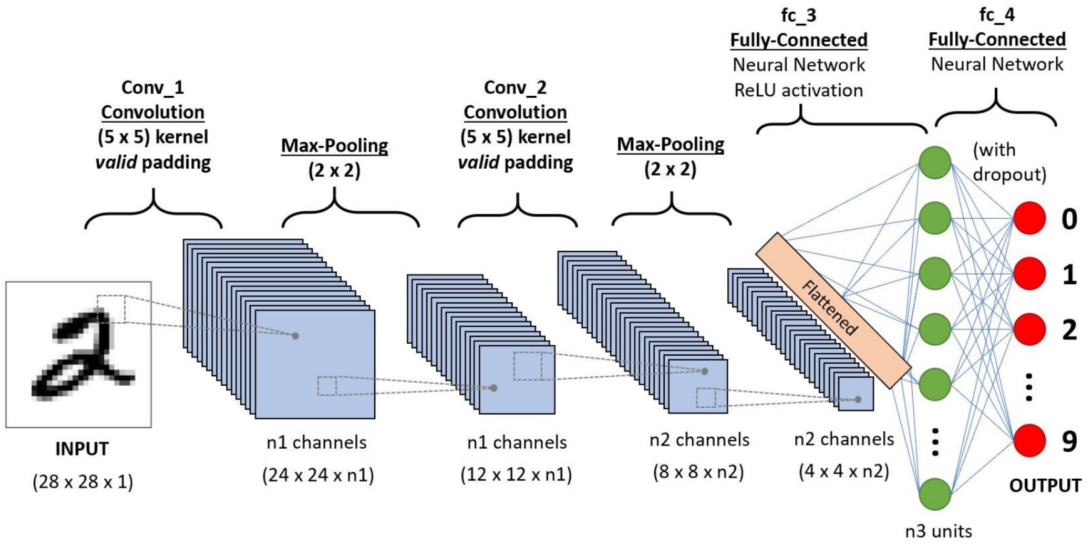

Figure 2.1. Convolutional Neural Network

Every neuron is connected to every other neuron of the following layer for fully connected neural networks. They have countless such associations that the intricacy of the architecture increments by a massive amount. It increases parameter count when applied to image data as it contains a large number of pixels. Hence, CNNs are a type of neural network primarily employed in most applications involving image data. One of their significant highlights is utilizing temporal and spatial correlations in images.

A classic CNN comprises layers of convolution and pooling layers, finally followed by fully connected layers in the end. Sometimes FC layer is substituted with average pooling layers. They are optimized with various activation functions, dropout, and kernel regularizers. The next sections describe the basic building blocks of CNN and their contribution to its performance.

2.2 Input Layer

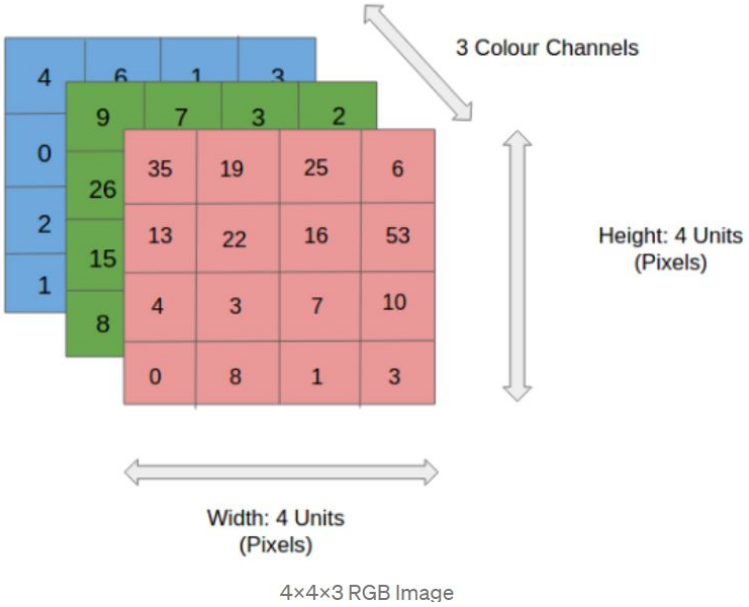

Figure 2.2. Input Image

Input to a CNN can be either image or a video. They are just an array representing pixel values across color channels or a 2D array in greyscale images. There are many color spaces of images to choose from, such as RGB, CMYK, HSV, etc. This input image data fed into CNN helps it learn representations from low, mid, and high-level features for classification.

2.3 Convolution Layer

Convolution forms the foundation of CNN. This is a linear operation of multiplying weights with the input matrix. Since the input is two-dimensional, multiplication is done by a kernel which is also a 2D matrix and smaller than the input image. The multiplication is a dot product that works by sliding the kernel across the input, multiplying the kernel's overlap area with input, and summing it to produce a single value. This kernel is often known as the filter that is deliberately made smaller in size to multiply with the input many times to search for a specific feature. This characteristic is also termed translation invariance since the search is concerned with the feature rather than the feature's location. The output

obtained from the convolution operator is a single value when done many times across a matrix along all dimensions; it is a multidimensional array also known as the feature map.

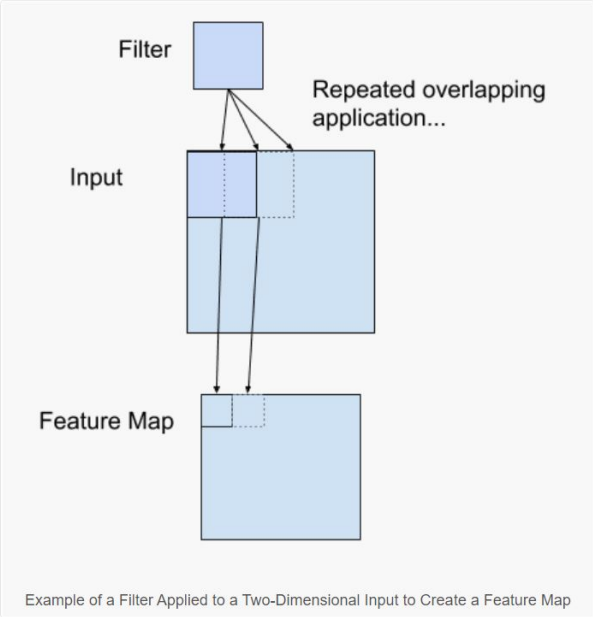

Figure 2.3. Convolution

To sum, up the convolution layer:

- Obtains an input with dimension $W_1 \times H_1 \times C_1$
- Accepts hyper parameters namely number of filters K, their kernel size F, stride S and zero pad size P
- Constructs an output of size $W_2 \times H_2 \times C_2$ where

$$W_2 = \frac{W_1 - F + 2 \times P}{S} + 1.$$

$$H_2 = \frac{H_1 - F + 2 \times P}{S} + 1.$$

$$C_2 = K.$$

Fig 2.3 illustrates the convolution operation explained so far.

2.4 Pooling Layer

One hindrance with using only convolution layers for feature extraction is that it detects features in the image's specific locations. It means that any minor alterations such as cropping, resizing, and so on can lead to different feature maps at the output. Downsampling is one such technique to make it more robust to this problem. It creates a lower resolution of an image with the same features present but not as fine details. It can be achieved by using strides in convolution. However, a better approach would be to use pooling layers. Pooling operation involves sliding a two-dimensional kernel across the input and all channels and summarizing whatever the kernel has seen. This summarization can be done in a variety of pooling functions such as below:

- Max-Pooling: It calculates the maximum value encompassed by the kernel. It also reduces any spurious deviations present in the image.
- Average- Pooling: It calculates the average pixel values covered by the kernel.

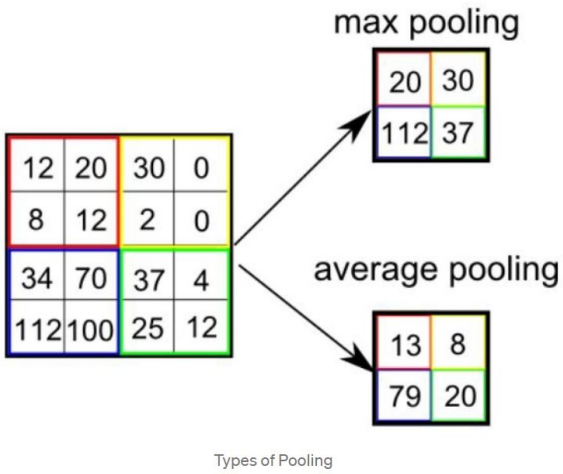

Figure 2.4. Pooling

To sum up, the pooling layer:

- Obtains an input with dimension $W_1 \times H_1 \times C_1$
- Accepts hyper parameters kernel size K and stride S

- Constructs an output of size $W_2 \times H_2 \times C_2$ where

$$W_2 = \frac{W_1 - K}{S} + 1.$$

$$H_2 = \frac{H_1 - K}{S} + 1.$$

$$C_2 = C_1.$$

2.5 Activation Function

Activation functions are an integral part of CNN. They add nonlinearity to the network making it more sturdy and efficient to recognize diverse data. There are many activation functions such as RELU, Swish, Tanh, Leaky RELU, etc.

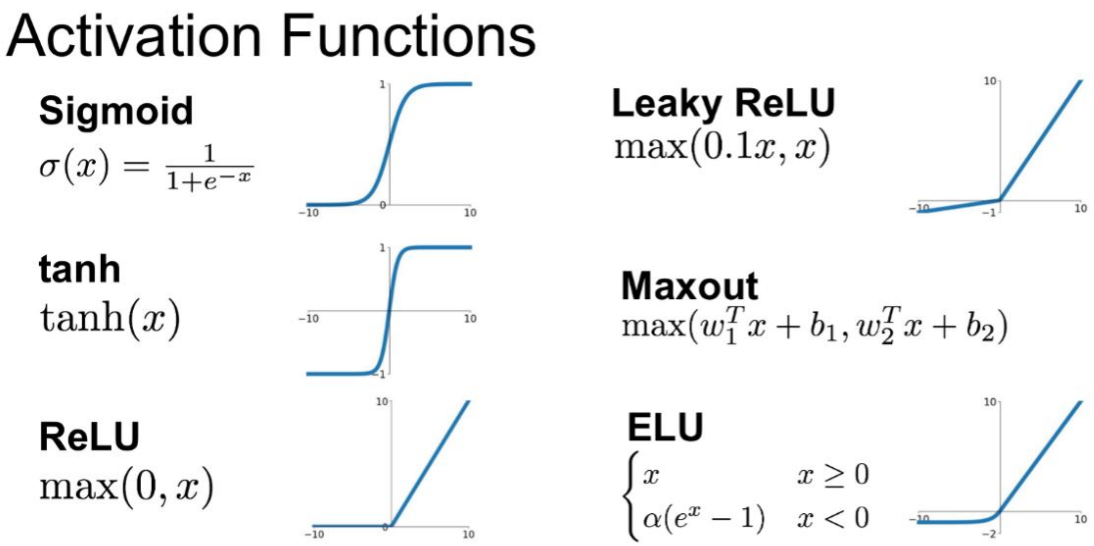

Figure 2.5. Commonly used activation functions

2.6 Fully Connected Layer

The pooled feature maps are subsequently flattened. This flattening is done by converting the matrix obtained from previous layers into a single column that is then inputted into a neural network. This neural network processing consists of an input layer, a fully con-

nected layer, and an output layer. FC layer produces a one-dimensional output comprising probabilities for class labels. These probabilities have to be between 0 and 1. The class with the highest chance forms the classification decision. This is where the softmax activation function comes into play. It takes in inputs K real numbers in vector z and produces K probabilities corresponding to inputs. The following formula defines it:

$$\sigma : \mathcal{R}^K \to (0,1)^K \tag{2.1}$$

$$\sigma(z)_{\mathrm{i}} = \frac{\mathrm{e}^{z_{\mathrm{i}}}}{\sum_{\mathrm{j}=1}^{K} \mathrm{e}^{z_{\mathrm{j}}}} \quad for \quad \mathrm{i} = 1, ..., K \quad and \quad z = (z_1, ..., z_K) \in \mathcal{R}^K \tag{2.2}$$

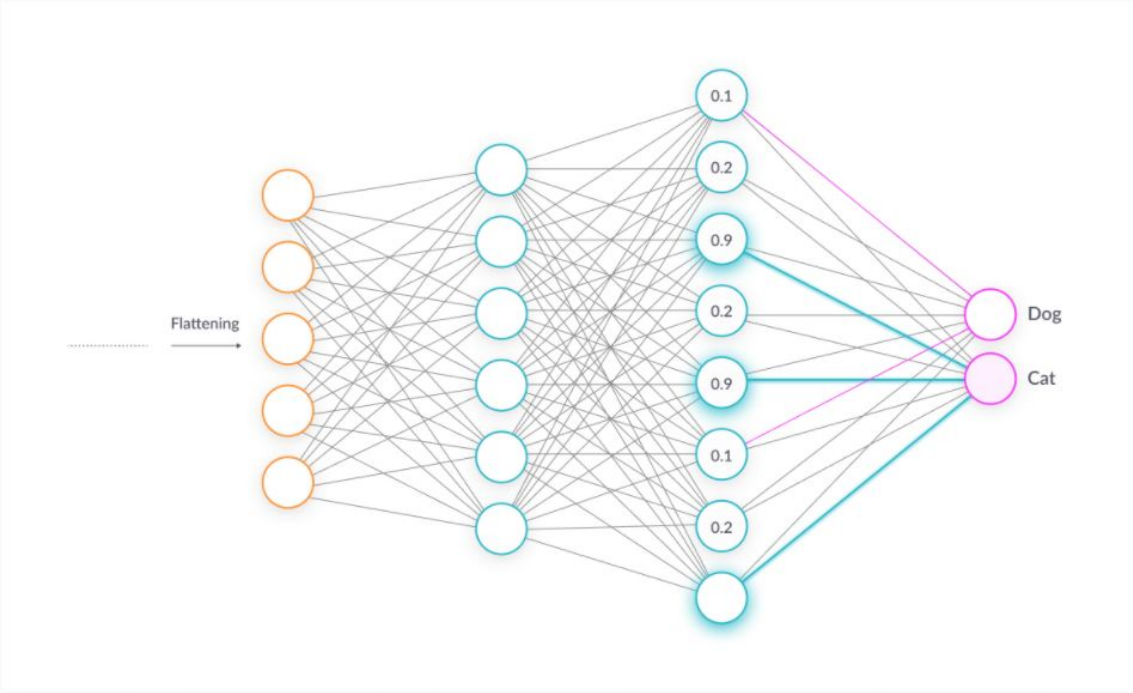

Figure 2.6. Fully connected layer

2.7 Baseline Architecture - MobileNet V3

MobileNet V3 [17] is the latest variant of a class of MobileNets designed to improve accuracy while being mindful of resource constraints. It was formed by a network-aware

platform architecture search complemented by a net adapt algorithm. It has surpassed other MobileNets in terms of accuracy and latency. There are two versions of this architecture-MobileNet V3 small and MobileNet V3 large-crafted to appease different resource platforms. MobileNet V3 large exceeds MobileNet V2's accuracy by 3.2% with 20% lesser latency. MobileNet V3 small's accuracy is 6.6% more with tantamount latency. These results were obtained on the ImageNet dataset. On the COCO dataset, Mobilenet V3 large is faster than MobileNet V2 [18] by 25% at roughly the same accuracy.

2.8 Building blocks of MobileNet V3

2.8.1 Depthwise Convolutions

Depthwise convolution is a particular type of convolution that applies a single filter for every input channel at a time in contrast to a standard convolution. Hence, as per the below figure illustrating depthwise convolutions, considering we have a filter of size $D_k \times D_k \times 1$, to obtain a dimension of size M we would need M such filters.
Total number of multiplications needed: $M \times D_k^2 \times D_p^2$.
Total number of parameters: $M \times D_k \times D_k$.

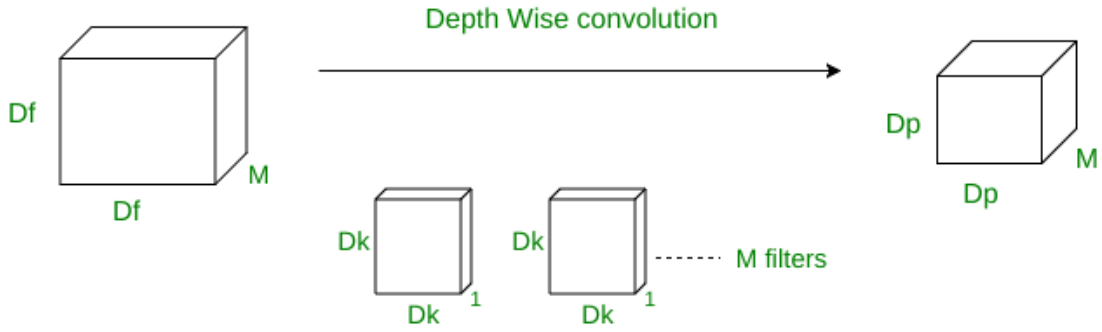

Figure 2.7. Depthwise convolutions

2.8.2 Pointwise Convolutions

Presented first in a paper by Min Lin et al. in their Network In Network [19], the 1×1 Convolution layer was utilized for 'Cross Channel Down inspecting' or Cross Channel Pool-

ing. As such, 1×1 Conv was utilized to decrease the number of channels while presenting non-linearity. In 1×1 convolution, a 1×1 filter is operated on M channels. So the dimensionality for the filter will be 1×1×M.Hence, if N such filters are incorporated, we get an output of size $D_p \times D_p$ ×N.

Total number of multiplications needed: M×D_p^2×N.

Total number of parameters: 1×1×M×N.

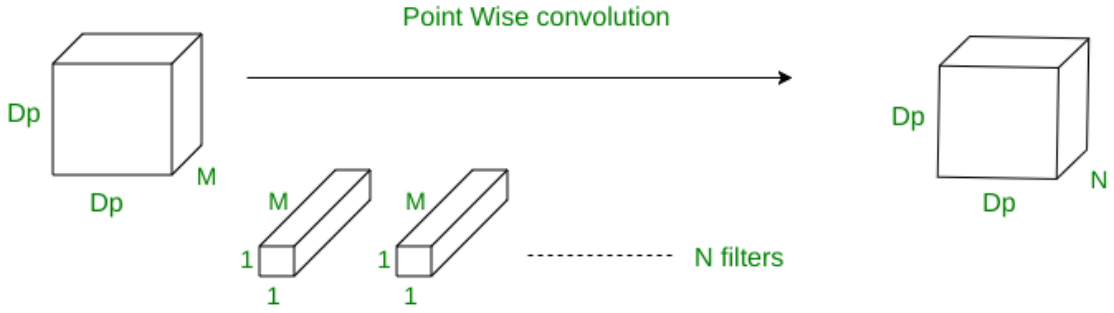

Figure 2.8. Pointwise convolutions

2.8.3 Squeeze and Excite blocks

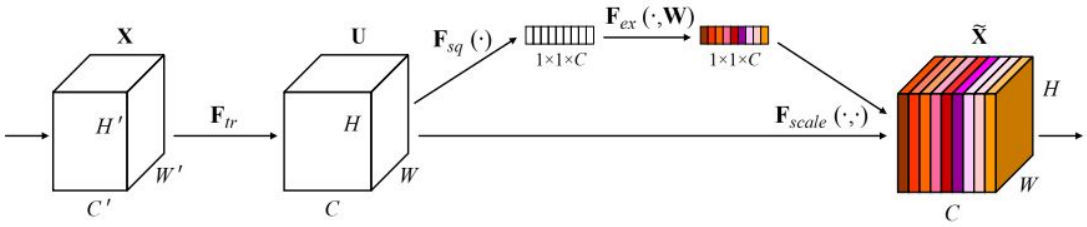

Figure 2.9. Squeeze and excitation blocks

The convolution operator plays a vital role in generating informative features by coalescing both spatial and channel information at neighboring receptive fields in each layer. There has been numerous research to enhance spatial encoding to improve network representational power. One technique developed was a squeeze and excite block aiming to unequivocally display the interdependencies between the channels of its convolutional highlights [20]. It achieved this by enhancing feature maps that contribute to convolution and repressing the

ones that do not. The below figure describes its mechanism. Feature maps from the previous layer are average pooled to produce an output of dimension 1×1×C, with C being previous layer channels. They are then passed through fully connected layers. The output of these fully connected layers is then multiplied with the initial input layer.

MobileNet V3 is made up of inverted residual bottlenecks and squeeze and excite blocks. These are derived from MobileNet V2 and squeeze and excitation networks. Their architecture constitutes a 1×1 pointwise expansion layer, 3×3 or 5×5 depthwise convolution, and a 1×1 projection layer. To emphasize their architecture's neurons that contribute to network performance and conceal neurons that do not, SE blocks are introduced.

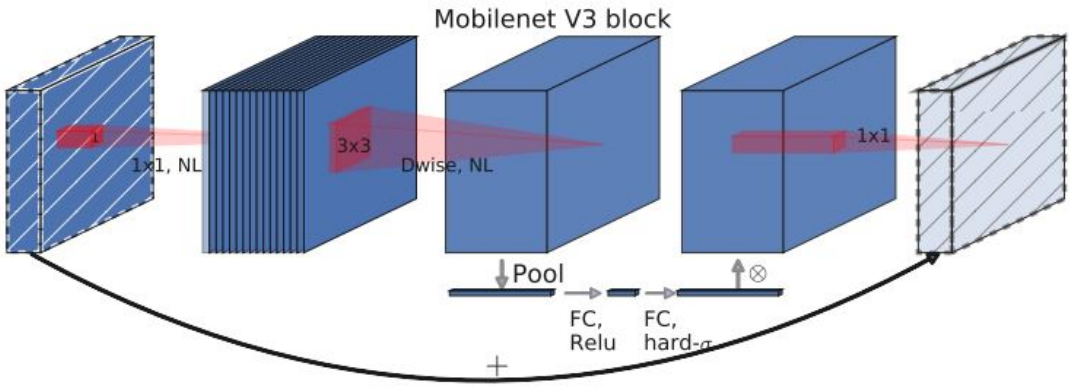

Figure 2.10. Building blocks of MobileNet V3

Some of the modifications made by authors to MobileNet V3 from MobileNet V2 are described below: They introduced the Hswish nonlinearity. Swish [21] has achieved good network performance, but due to the sigmoid present in swish it can be computationally expensive for embedded platforms. Swish is mathematically described as below:

$$Swish(x) = x \times \sigma(x) \tag{2.3}$$

H-swish is a hard counterpart of swish. As far as the graph is concerned, it is not that different while being computationally less expensive. Mathematically it is described as below:

$$H - swish(x) = x \times \min(\frac{\max(features, 0)}{6}, 6) \tag{2.4}$$

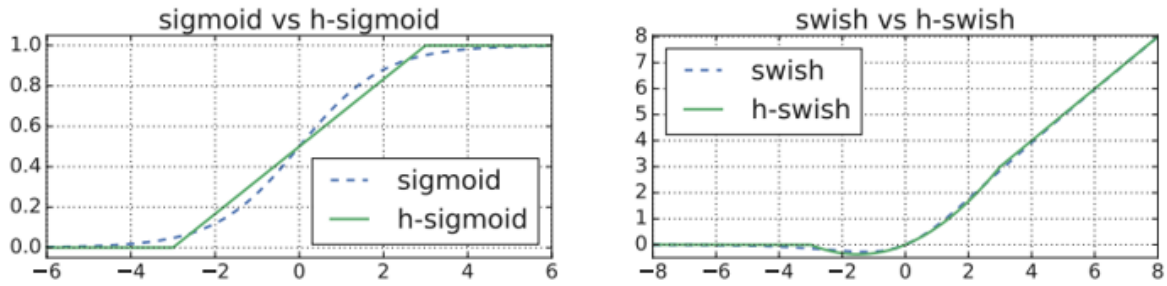

Figure 2.11. Hard counterparts of Swish and Sigmoid activations

They also made changes to the last block of MobileNet V2 by moving the 1x1 expansion layers in inverted residual bottlenecks past the pooling layer. This change allows the 1x1 layers to operate on feature maps of size 1x1 instead of 7x7 reducing latency. It eliminates the need to do compression from the projection layer, thus removing it from the bottleneck. These modifications can be observed below.

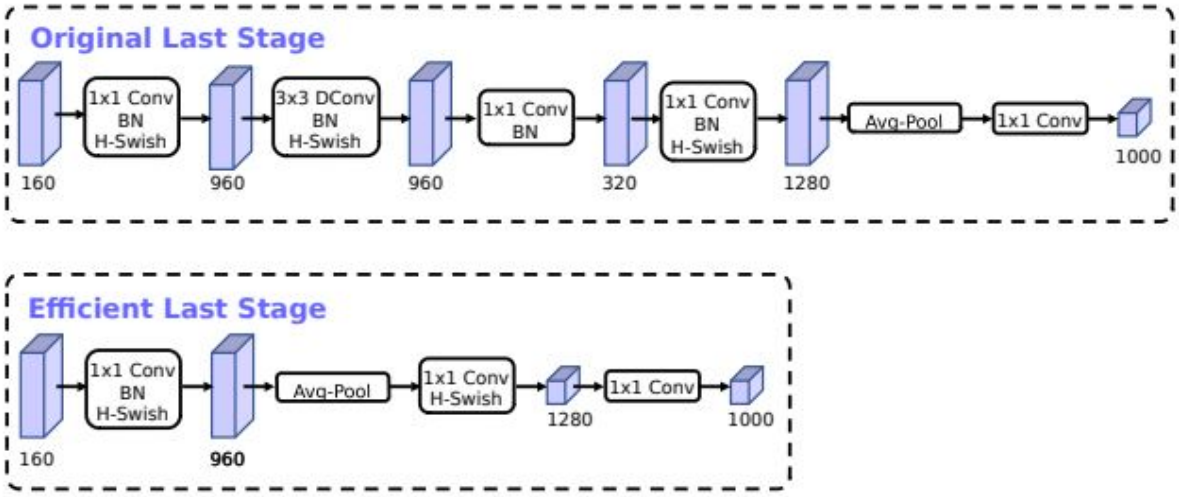

Figure 2.12. Comparison of last stages from MobileNet V3 and MobileNet V2

The following table describes the architecture of MobileNet V3 small.

Table 2.1. MobileNet V3 - small Architecture

MobileNet V3 - small Architecture						
Input	Operator	e	c	SE	NL	s
$224^2 \times 3$	Conv2D 3×3	-	16	-	HS	2
$112^2 \times 16$	Bneck 3×3	16	16	$\surd$	RE	2
$56^2 \times 16$	Bneck 3×3	72	24	-	RE	2
$28^2 \times 24$	Bneck 3×3	88	24	-	RE	1
$28^2 \times 24$	Bneck 5×5	96	40	$\surd$	HS	2
$14^2 \times 40$	Bneck 5×5	240	40	$\surd$	HS	1
$14^2 \times 40$	Bneck 5×5	240	40	$\surd$	HS	1
$14^2 \times 40$	Bneck 5×5	120	48	$\surd$	HS	1
$14^2 \times 48$	Bneck 5×5	144	48	$\surd$	HS	1
$14^2 \times 48$	Bneck 5×5	288	96	$\surd$	HS	2
$7^2 \times 96$	Bneck 5×5	576	96	$\surd$	HS	1
$7^2 \times 96$	Bneck 5×5	576	96	$\surd$	HS	1
$7^2 \times 96$	Conv2D 1×1	-	576	$\surd$	HS	1
$7^2 \times 576$	Pool 7×7	-	-	-	-	1
$1^2 \times 576$	Conv2D 1×1	-	1024	-	HS	1
$1^2 \times 1024$	Conv2D 1×1	-	k	-	-	1

3. HARDWARE AND SOFTWARE USED

3.1 Requirements

- Intel Xenon Gold 6126 processor with 32GB RAM
- NVIDIA Tesla P100 GPU
- NXP i.MX RT 1060
- Tensorflow
- Keras
- Tensorboard
- Python
- MCUXpresso SDK
- Neural Network Intelligence

3.2 i.MX RT 1060

i.MX RT1060 is one of the latest addition to NXP's boards utilizing ARM Cortex M7. It has a speed of 600MHz excelling in performance and reduced latency [22] [23].

3.2.1 Features

- Incorporates Arm Cortex M7
- Core clock at 600Mhz
- On-chip SRAM size 1MB
- Response time as fast as 20ns
- Real-time performance
- Low power support at 24Mhz
- Multimedia support- Camera sensor interface and LCD display interface

- Supports MCUXpresso IDE and SDK

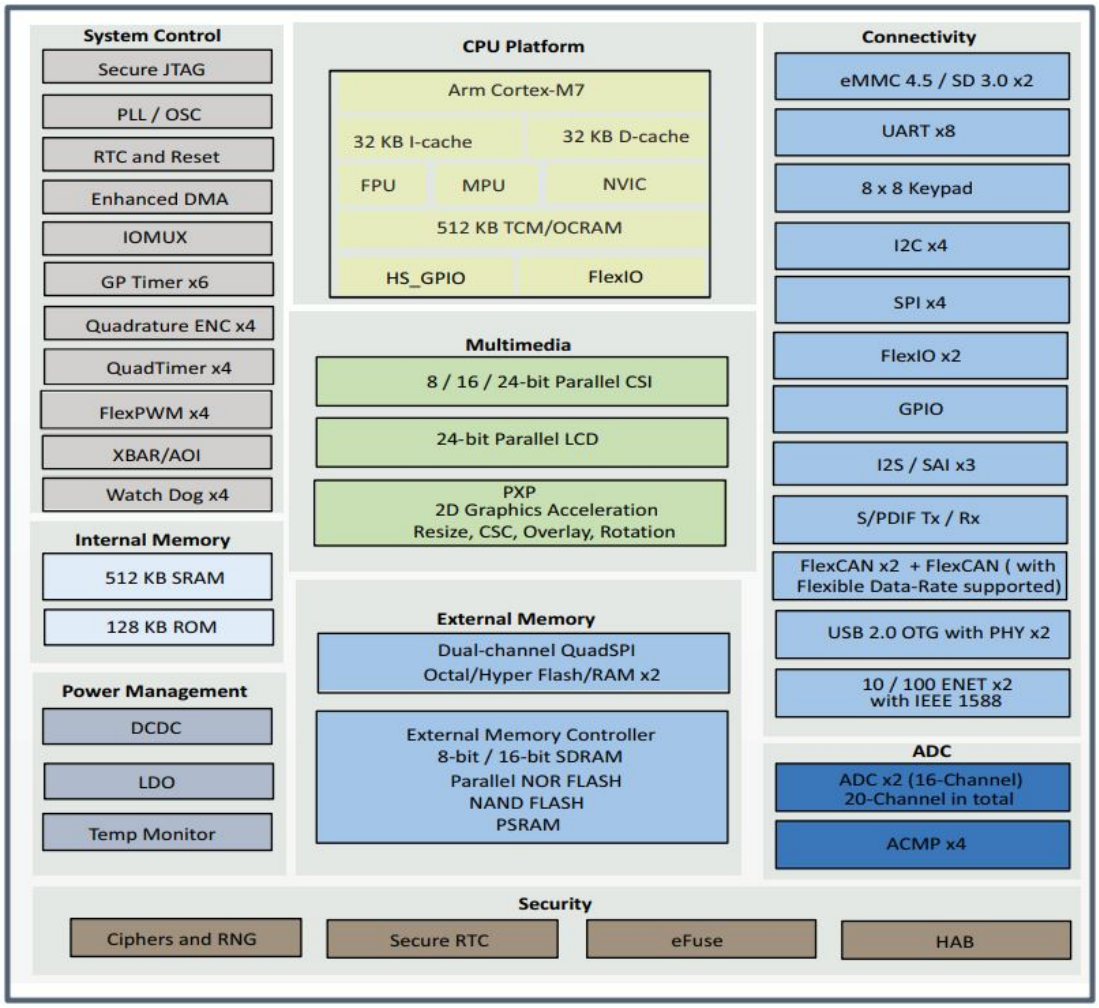

Figure 3.1. System block diagram of i.MX RT 1060

3.3 Tensorflow

Tensorflow is an open-source library developed by the Google Brain team to train and inference several machine learning applications. It is available for Linux, Windows, macOS, and a few mobile computing OS like Android and iOS. It has effortless compatibility between CPU, GPU, and TPU platforms. A lighter version of TensorFlow was developed, namely Tensorflow Lite, for use in microcontrollers. It also supports other platforms like Android,

IOS, and Linux [24]. Standard Tensorflow was used to train the model on PC, and the inference was performed on i. MX RT 1060 using TFLite. Performing inference using TFlite usually compose the following steps:

1. Loading trained model into memory: It comprises the execution graph of the model.

2. Converting raw data: Input data may not be compatible with the model as it might expect data in a different format. In such a case, making an alteration becomes imperative. For example, resizing the image before feeding it to the model, etc.

3. Performing inference: This process involves using TFLite APIs to make predictions on input data. The steps are mentioned below:

    - Construct the interpreter for the loaded model
    - Introduce input tensor values
    - Register kernel operations needed by the model
    - Start inference

4. Converting the output to an interpretable form: Results from model inference have to be interpreted suitably, based on application.

Tensorflow also offers a visualization toolkit known as Tensorboard. It helps in navigating through machine learning projects. Its features are depicted below:

- Helps in viewing loss and accuracy metrics
- Comprehend operations and layers in model's graph
- Performing parameter tuning
- Depict parameter change over time
- Shows representations in low dimensional space
- Demonstrates image, audio, and textual data

3.4 Keras

Keras is a deep learning API built on top of Tensorflow. It allows fast and easy implementation of ideas due to its user-friendliness. It also supports various platforms like CPU, GPU, and TPU. It can be exported to mobile platforms as well.

3.5 Neural Network Intelligence

Neural network intelligence is a TensorFlow compatible toolkit designed to ease machine learning projects. It supports automation of feature engineering, hyper-parameter tuning, neural network architecture search, and model compression. It also helps running of hundreds of trials using parallel processing and leverages efficient usage of computational resources. Its very flexible and intuitive UI makes it easy to navigate through various model metrics and configurations.

3.6 MCUXpresso

MCUXpresso is a software development environment designed to support NXP's ARM-based MCUs. It comes with the GNU toolchain. It allows unlimited code size for users. It supports both SWD and JTAG debugging. It supports various board specific SDKs that comprise various drivers, firmware, and demo examples for implementation.

4. DATASET

Choosing the correct dataset is essential to benchmarking model performance in training and validation. The use of the datasets relies upon the design of the issue and the uniqueness of the case. A few considerations before choosing the suitable dataset are; it should be of good quality and relevant to the problem at hand, it should be clean as otherwise a considerable amount of time would be spent on cleaning it, and the dataset should not be too vast adding computational load on resources. For this research, we have used the CIFAR -10 dataset [25]. The CIFAR-10 dataset is an assortment of pictures ordinarily used to prepare PC vision and other machine learning applications. It constitutes 60,000 images of size 32×32 in 10 different categories. These ten categories include cars, birds, cats, airplanes, frogs, dogs, deer, ships, horses, and trucks. Each of these categories contains 6000 images. These images were divided into 50,000 training and 10,000 testing images.

5. PROPOSED ARCHITECTURE

Many techniques are prevalent to perform model compression such as sparse decomposition and low rank approximation [26], pruning and network quantization [27] [28] [29] and knowledge distillation [30]. Some research even focusses on developing smaller efficient models from scratch. In knowledge distillation, small-sized models emulate larger models' behavior and learning techniques using a student-teacher type of setting. In pruning, weights inconsequential to network success are zeroed out based on some well-established weight ranking methods. In network quantization, weights in both filters and fully connected layers are quantized in their bit representation. They can be further compressed using huffman coding.

In this thesis, model compression is done employing different bottlenecks for convolution, hyper-parameter tuning, change of activation function and introducing more expansion filters. The below table illustrates the proposed architecture.

Table 5.1. Compressed MobileNet V3 Architecture

Compressed MobileNet V3 Architecture						
Input	Operator	e	c	SE	NL	s
$32^2 \times 3$	Conv2D	-	16	-	HS	1
$32^2 \times 16$	Bneck 3×3	48	32	$\surd$	HS	1
$32^2 \times 32$	DPD 3×3	88	40	-	MH	1
$32^2 \times 40$	DPD 3×3	240	40	-	MH	1
$32^2 \times 40$	Bneck 5×5	160	48	$\surd$	HS	2
$16^2 \times 48$	DPD 5×5	288	96	-	MH	1
$16^2 \times 96$	DPD 5×5	592	128	-	MH	1
$16^2 \times 128$	Conv2D 1×1	-	256	$\surd$	HS	1
$16^2 \times 256$	Pool 16×16	-	-	-	-	1
$1^2 \times 256$	Conv2D 1×1	-	576	-	HS	1
$1^2 \times 576$	Conv2D 1×1	-	k -	-	-	1

e: channel expansion factor, c: output channel dimension, SE: squeeze and excite blocks, NL: Activation function, HS: H-Swish , MH: Mish and s: stride.

5.1 Improved Convolution Blocks

To achieve good trade-off between model accuracy and size, new convolution blocks were introduced. These blocks are selected to improve feature extraction as efficiently as the baseline with no compromise to model accuracy[31]. Standard convolution can be accomplished by consolidating depthwise convolutions and pointwise convolutions. We realize that 1×1 convolution has fewer parameters than 3×3 standard convolution while expanding or diminishing the channels' quantity. Along these lines, PWC is generally used to increment or decline the quantity of channels in CNNs.Whenever depthwise convolution is used, the channel multiplier m is set to 1, ensuring that output produces the same number of channels as input. If we set m>1, it could also expand dimension along channels. Hence, this feature is incorporated in CMV3.

CNN with higher depthwise convolutions in comparison to pointwise convolutions has outpaced the ones with more pointwise convolutions. Hence, advocates for acquiring more spatial information than coalescing channel information [32]. The proposed model incorporates DPD blocks to accomplish this. Usage of depthwise convolutions rather than pointwise convolutions for channel expansion leads to smaller model size. Hence, DPD blocks encompass an initial 3×3 depthwise convolution with a stride s, which reduces input dimension and expands along channels, a pointwise convolution that reduces the number of channels, and then a final 3×3 depthwise convolution. These blocks are escorted by batch normalization and RELU activation. Few of these blocks were modified to include 5×5 kernels to improve effective receptive field as done in MobileNets. Due to the incorporation of more depthwise convolutions, a compression as shown below is reached.

$$\begin{aligned} Compression &= \frac{W \times H \times C \times m \times C}{W \times H \times k \times k \times m \times C} \\ &= \frac{C \times m \times C}{k \times k \times m \times C} \\ &= \frac{C}{k^2} \end{aligned} \tag{5.1}$$

The numerator is the parameter count in pointwise convolution and denominator is the parameter count in depthwise convolution. Here, (W×H) is the input size, C is the number

of channels, m is the multiplier factor for channel and k is the filter size. Since number of channels is frequently larger than kernel size this ratio is greater than 1. Below figure illustrates DPD blocks.

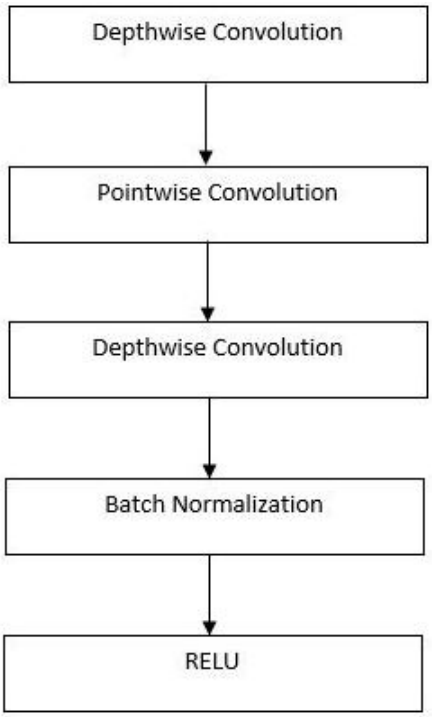

Figure 5.1. Depthwise Pointwise Depthwise (DPD) blocks

5.2 Mish Activation Function

The idea of non-linearity in a Neural Network is presented by an enactment function that serves an essential job in preparing and executing the network. It helps the model discern the nonlinear connections between input and output. Throughout the long periods of theoretical examination, numerous enactment capacities have been proposed, such as ReLU, TanH, Sigmoid, Leaky ReLU, and Swish. These were widely used in most neural networks. For this thesis, we have used the Mish activation function. The similitude to Swish alongside giving a lift in execution and its straight forwardness in execution makes it simpler for specialists and engineers to utilize Mish in their Neural Network Models. It is added after DPD blocks instead of RELU. Experimental results of Mish show that Mish

will, in general, work better compared to both ReLU and Swish alongside other standard activation functions in numerous DL networks across comprehensive datasets. Some of the advantages of using the Mish activation function are:

- It is unbounded above, allowing large values of input to not saturate towards a maximum value hence contributing to the eradication of near-zero gradients.
- It is also bounded below. Input moving towards negative infinity causes output to remain constant. It also means that tremendous negative values are inferred by mish as deactivations. It helps improve regularization in the network.
- It is continuously differentiable up to infinite order. Thus, eliminating problems of vanishing gradients.
- It is a non-monotonic function. This preserves small negative inputs producing negative outputs. It yields both positive and negative derivatives, which increases the expressivity of the network and improves gradient flow.
- It is a relatively smooth function. This feature enables fewer oscillations in models loss function minimization leading to faster convergence and learning. Fig 5.1 shows the graph of Mish. Fig 5.2 shows other activation functions along with Mish.

$$f(x) = x \cdot \tanh(softplus(x)) \tag{5.2}$$

$$softplus(x) = \ln(1 + \mathrm{e}^{x}) \tag{5.3}$$

Addition of Mish improved model accuracy from 88.14% to 89.13%.

5.3 Expansion filters

MobileNet V3 uses expansion filters to go to high dimensional feature space and advance non-linear transformation in each channel [17]. It accomplished this using 1×1 convolutions. This fact has been used in improving the accuracy of CMV3. Instead of using 1×1 filters

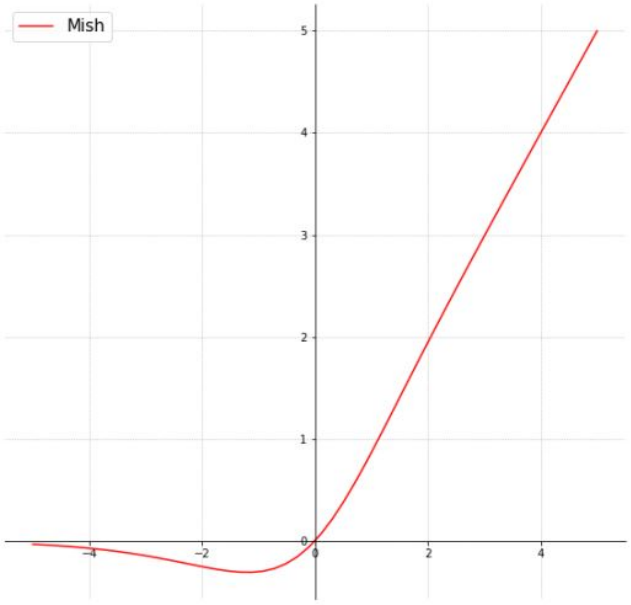

Figure 5.2. Mish Activation Function

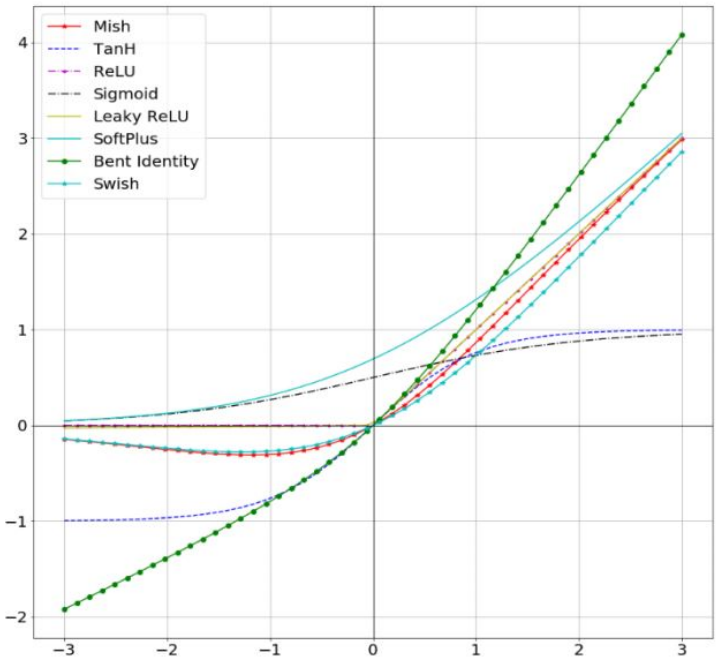

Figure 5.3. Common activation functions

as used in baseline, using DPD blocks in which depthwise convolutions perform the task of expansion, increasing expansion filters do not escalate parameter count exponentially. Hence, this change boosted the accuracy from 84.56% to 88.14%.

5.4 Hyper-parameter Tuning

When building AI models, different hyper-parameters need to be picked, for example, the learning rate or weight regularizer. These choices sway model measurements, like accuracy and over-fitting. Along these lines, a significant advance in the AI work process is to distinguish the best hyper-parameters for your concern, which frequently includes experimentation. This cycle is known as "Hyper-parameter Tuning."

There are two primary ways of determining model hyper-parameters, namely manual tuning and automated tuning. In manual tuning, hyper-parameters are experimented with by manually applying to a network by making a sound judgment based on data. It is a strenuous process as the number of hyper-parameters increase. An automated tuning accomplishes this goal by utilizing an algorithm to find optimal parameters in a search space. A few of the popular algorithms are as follows:

- Random search: In this algorithm, we make a matrix of potential hyper-parameters and their values. Every cycle attempts an arbitrary mix of hyper-parameters from this lattice, takes note of performance, and ultimately returns the hyper-parameters mix, which gave the best value for accuracy.
- Grid search: We make a framework of values for hyper-parameters. Every run attempts a mix of hyper-parameters in a particular order. It fits the model on using all combinations conceivable. At long last, it returns the best hyper-parameters based on tuning.
- Bayesian search: This technique views finding the best hyper-parameters as an optimization problem. Minimization of loss function using a variety of model hyper-parameters can be speeded up using the Bayesian approach. This function directs search space sampling based on which direction gives better performance than the current choice of parameters.

Grid search was used for extracting hyper-parameters for the proposed model using Neural Network Intelligence (NNI), an interactive visualization tool for hyper-parameter tuning. Various hyper-parameters tuned for CMV3 are described in the below subsections. The below figure shows the search space along with accuracies for various combinations of hyper-parameters.

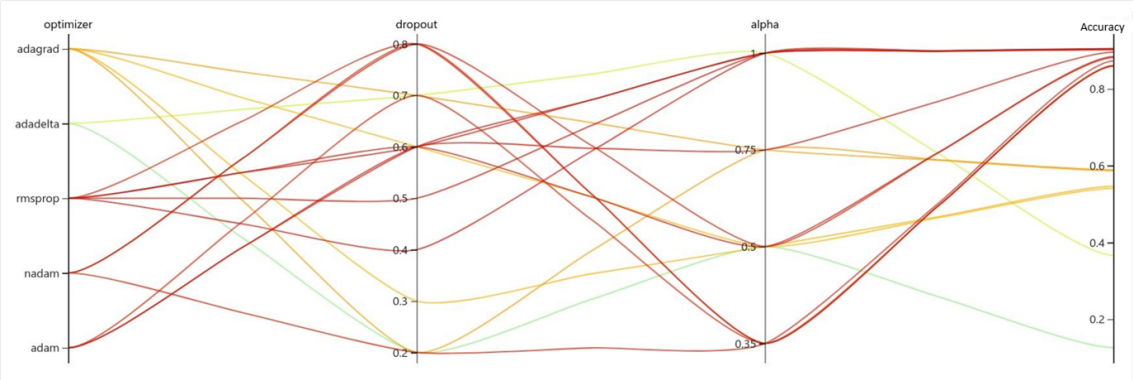

Figure 5.4. Hyper-parameter search space used for CMV3

5.4.1 Optimizers

Optimizers revise weights to lessen the loss in a network. Loss functions instruct optimizers by making them reach the global minimum. There are various optimizers, each having its benefits. Adaptive Moment Estimation (ADAM) optimizer calculates learning rates that are adaptive for each parameter. It considers both the exponentially decaying average of past square gradients and past gradients. Nesterov accelerated gradient is another optimizer that calculates the gradient for present parameters but concerning the predicted value of parameters from the previous step. To train CMV3, we have used the NADAM optimizer; it is a combination of Adam and Nesterov Accelerated Gradient (NAG). It is very effective in noisy gradients with higher curvatures. It updates weights based on exponential decay of averages of past and current gradients.

5.4.2 Weight Regularizers

Weight regularizers are often added to reduce overfitting in the model [33]. They manage this by permitting users to penalize weights in the course of optimization. They can be of various types: l1, the sum of absolute weights, and l2, which is the sum of squared weights. We have used an l2 weight decay of 0.00001 for the model.

5.4.3 Learning Rate Schedulers

Learning Rate decay helps control the rate at which a model is changed in feedback to estimated error. Choosing a learning rate plays a vital role, as a lower learning rate may take a long time to move towards a global minimum, whereas a larger learning rate may lead to unstable training. Using a learning rate scheduler, we can change the learning rate while training the model. Among the schedulers, step decay, time decay, cosine decay annealing are well established. In this thesis, we used a cosine decay scheduler [34] for optimizing the model. Given a starting learning rate, this scheduler incorporates a cosine decay function to optimization.

5.4.4 Dropout

Dropout is another technique of ameliorating over-fitting. The critical thought in dropout is to drop a few units as well as their connections. It prevents the other units from over adapting to the network. It is done only during training. At test time, evaluation is done with all units retained. It gives an approximation of predictions from all units, thus reducing over-fitting compared to other regularizers. A hyper-parameter that controls the probability of units dropped from the network has to be tuned in dropout. For example, if the probability p=0.5, then 50% of the units have to be dropped. In CMV3, a dropout of 0.8 helped in reducing over-fitting.

Table 5.2. Summary of hyper-parameters chosen for CMV3

Summary of hyper-parameters chosen for CMV3	
Parameter	Value
Optimizer	NADAM with 0.9 decay rate for momentum and 0.999 for weighted infinity norm
Batch size	16
Regularizer	L2 with a regularization factor of 0.00001
Dropout	0.8
Learning rate scheduler	Cosine decay
Width multiplier	0.5

6. HARDWARE DEPLOYMENT

Compressed MobileNet V3 has an accuracy of 89.13% with a size of 2.3 MB. This reduction made it efficient for deployment on i. MX RT 1060, an MCU crossover platform developed by NXP for machine learning applications. Sections in this chapter illustrate the various stages in deployment.

6.1 Converting CMV3 to Tflite format

NXP eIQ is a software forum encompassing elements to ease machine learning deployment on hardware. It has Neural Network (NN) libraries, compilers, inference engines, and Hardware Abstraction Layers (HAL) to bolster development using ARM NN, glow, TFLite, CMSIS, and Open CV. TFlite was preferred for the deployment of this model. It is a lightweight alternative to TensorFlow. NXP eIQ software for TFLite is available in both MCUXpresso and Yocto development environments. Hence, to make the model compatible with the settings mentioned earlier, it was saved in .pb format after training. This format allows saving custom-defined layers and activation functions in the model. This .pb format was converted into TFlite using python API for the TFLite converter.

6.2 Running CMV3 on i.MX RT 1060

For deployment, MCUXpresso IDE was utilized. The corresponding SDK required by the hardware was built using eIQ middlewares. This SDK has numerous examples to demonstrate various machine learning applications. One such example is the CIFAR 10 label image example. It uses a DL model to classify images using a camera attached to the board. CMV3 was made compatible with this example by converting its TFLite file to .h file, constituting opcodes for model operations. All the necessary kernel operations needed for successful execution of model were registered dynamically. This was then used to classify various images from the camera and display it on the console using semihosting. Fig 6.1 shows the results obtained. Fig 6.2 shows the block diagram of deployment procedure.

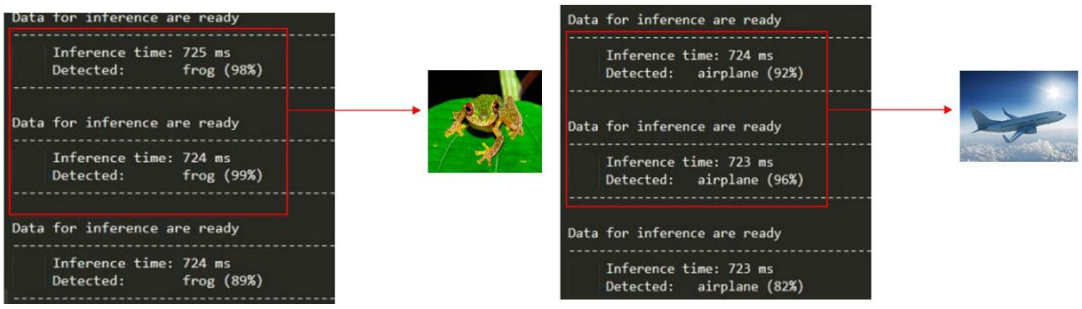

Figure 6.1. Image classification using CMV3 on i.MX RT 1060

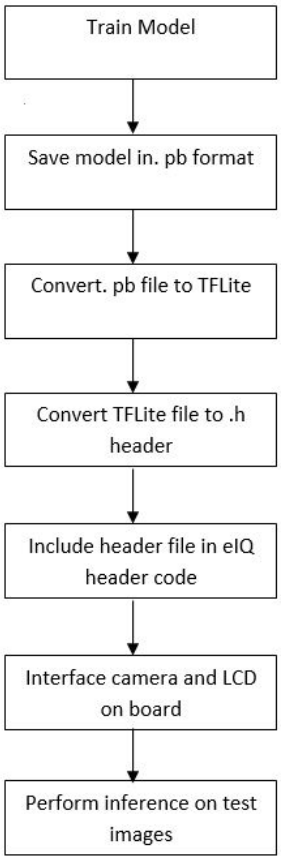

Figure 6.2. Hardware deployment block diagram

7. RESULTS

Compressed MobileNet V3 was deployed on i.MX for image classification. It successfully classified images with an average inference time of 720ms. It is a lightweight version in comparison to MobileNet V3 small with no compromise to accuracy. It has a size of 2.3 MB while being 89.13% accurate. It outperformed the baseline achieving the best trade-off between model size and accuracy. Table 7.1 shows a comparative analysis of baseline with CMV3. Fig 7.1 and 7.2 shows the accuracy and loss vs. epochs for the revised model visualized using tensorboard. Fig 7.3 and 7.4 show the same for the baseline model. In order to understand the effect of the width multiplier on model accuracy, a thorough analysis has been done. This analysis is depicted in Table 7.1. A suitable width multiplier can be chosen based on the application and its implied resource requirements.

Table 7.1. Comparative Analysis of MobileNet V3 and CMV3

Comparative Analysis			
Model	Model Accuracy	Model size	Parameters
MobileNet V3	88.93%	15.3 MB	1,846,930
Compressed MobileNet V3	89.13%	2.3 MB	171,946

Table 7.2. Various Scaling factors for CMV3

Various Scaling factors for CMV3		
Width Multiplier	Model Accuracy	Model size
1.5	91.39%	10.5 MB
1.0	90.64%	5.2 MB
0.75	90.10%	3.6 MB
0.5	89.13%	2.3 MB
0.35	87.36%	1.9 MB

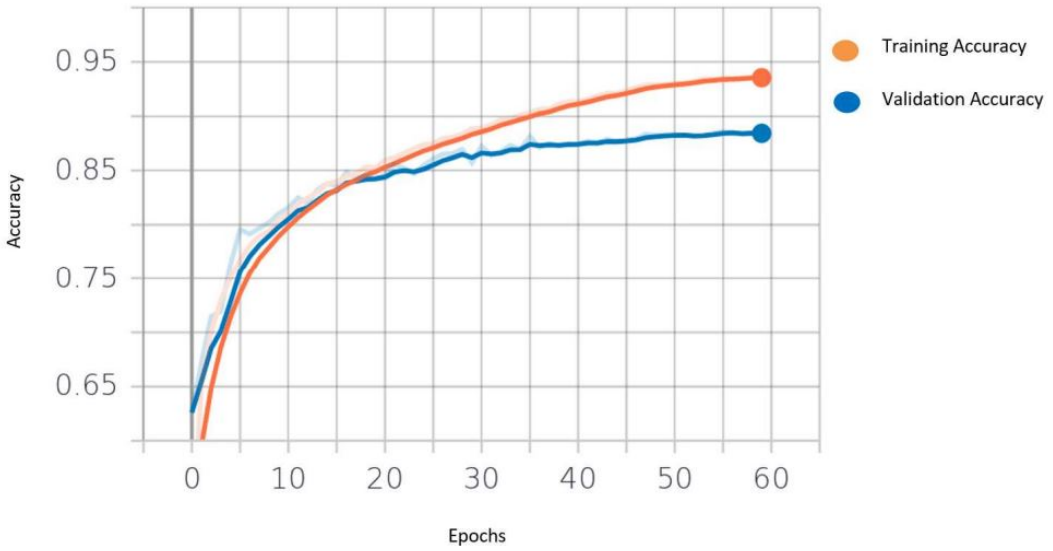

Figure 7.1. Plot Accuracy vs number of epochs for CMV3

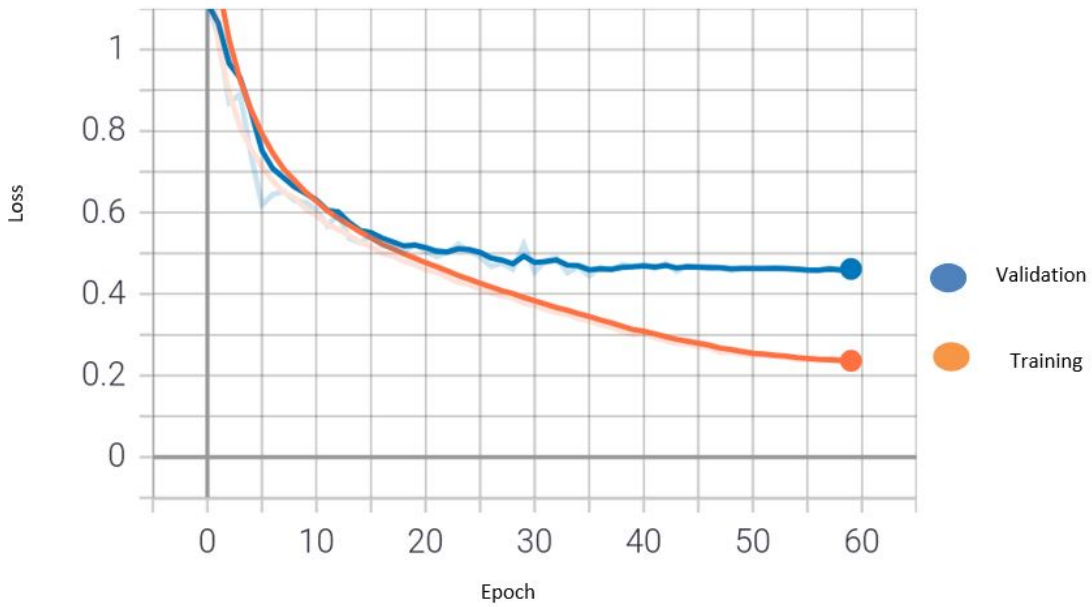

Figure 7.2. Plot of Model loss vs number of epochs for CMV3

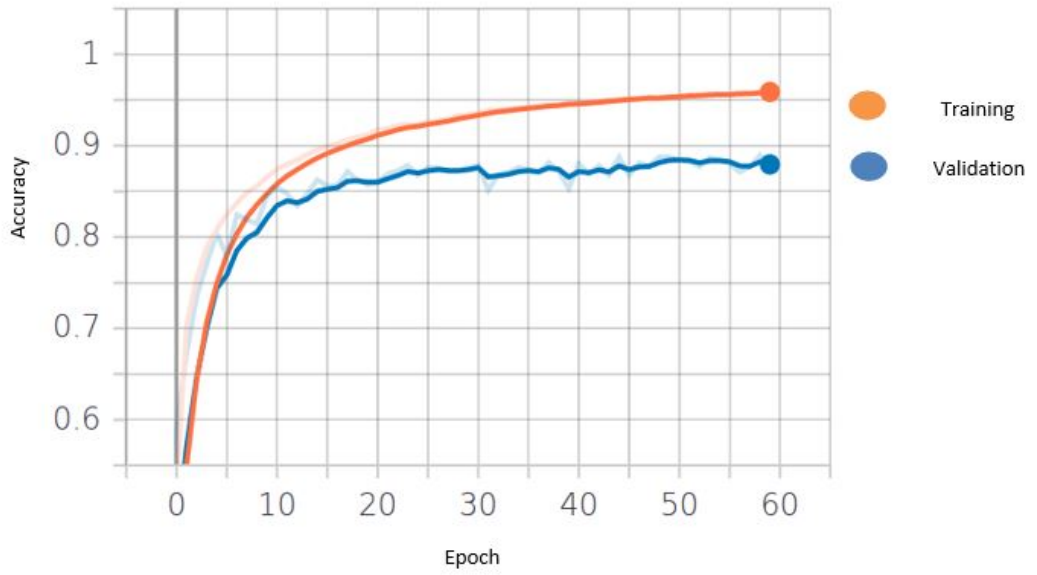

Figure 7.3. Plot of Model loss vs number of epochs for baseline

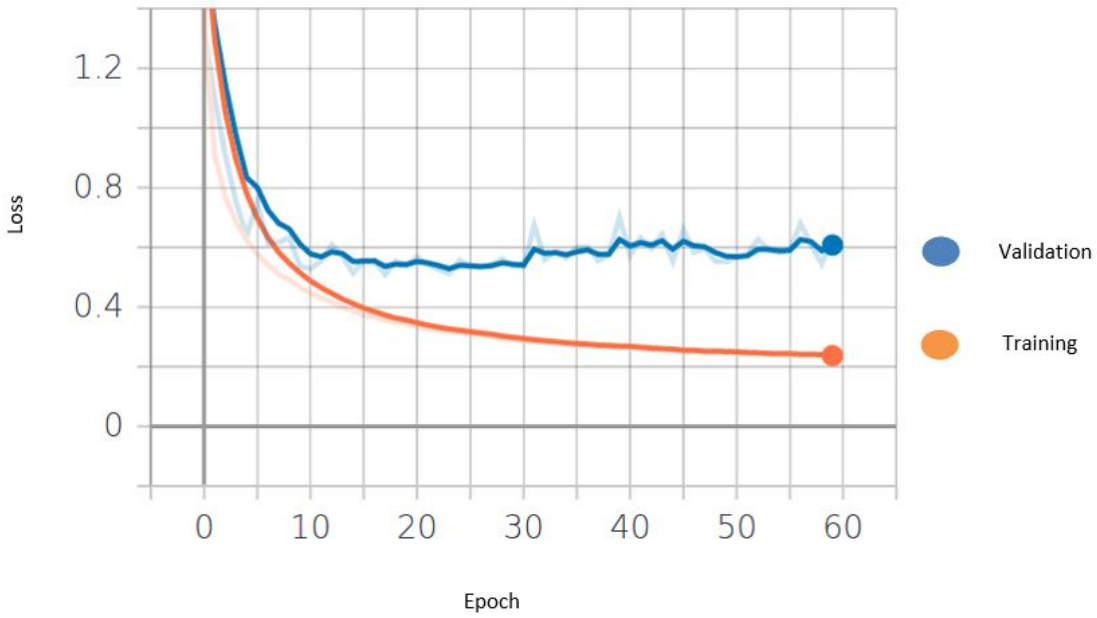

Figure 7.4. Plot of Model loss vs number of epochs for baseline

8. CONCLUSION

Lately, deep neural networks have earned considerable attention, been applied to numerous applications, and accomplished commendable precision enhancements in various tasks. These works depend on models with millions or sometimes billions of parameters, and the accessibility of GPUs with extremely high calculation ability assumes a critical part in their victory. Krizhevsky et al. accomplished advancement in image classification in the 2012 ImageNet Challenge utilizing a model accommodating 60 million parameters [3]. Such models take days to train to get good performance. As network complexity increased, demand for reducing storage and computation began to soar, especially for real-time applications. Likewise, ongoing years saw critical advancement in wearable gadgets, augmented reality, and virtual reality setting out phenomenal open doors for scientists to handle principal challenges in sending DL frameworks to compact devices with restricted assets (for example, memory, CPU, energy, data transmission). Hence, the characteristic idea is to perform model compression and speed increase without fundamentally diminishing the model performance. Accomplishing these objectives calls for joint arrangements from numerous disciplines, including, however not restricted to AI, signal processing, optimizations, computer architecture, and hardware design [35].

In this thesis, moving in a similar direction, we have explored various model compression techniques and incorporated suitable ones to reduce the size of MobileNet V3 small. Using a combination of DPD blocks, Mish activation function, increasing expansion filters and performing hyperparameter tuning, a model with an accuracy of 89.13% and size 2.3MB was obtained. It was trained and tested on the CIFAR-10 dataset. It surpassed the baseline model accuracy by 0.2% while being 84.96% smaller in size. Various scaling multipliers were also proposed, which could be decided upon based on application and accuracy requirements. This model can now be used on embedded platforms and mobile vision applications.

In this thesis, we also deployed the model on to i. MX RT 1060, an embedded machine learning platform developed by NXP. It performed image classification with good confidence levels with an average inference time of 724 ms. Hence this model can be further explored

and used in many other applications like object localization, semantic segmentation, object detection, etc., on embedded devices as explained in the next chapter.

9. FUTURE SCOPE

Although showing a good trade-off between size and accuracy, the proposed model can be further explored to achieve better optimizations. Below are a few of the aspects that could be delved into:

- Pruning - The compressed model can be examined to determine insignificant weights. Using a suitable ranking criterion, these weights that do not contribute to network performance can be removed.
- Quantization - The total of bits used to serve as parameters in the network could be reduced using various techniques. It would help in efficient deployment in resource-constrained environments and improve inference time.
- Other CV applications - CMV3 could be employed for alternative tasks such as object detection, object localization, semantic segmentation, and so on.Their performance on hardware could be analyzed.
- Other embedded platforms - This model could also be deployed on other embedded platforms to analyze performance.

REFERENCES

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989. DOI: 10.1162/neco.1989.1.4.541.

[2] N. O. Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. A. Velasco-Hernández, L. Krpalkova, D. Riordan, and J. Walsh, "Deep learning vs. traditional computer vision," *CoRR*, 2019. arXiv: 1910.13796. [Online]. Available: http://arxiv.org/abs/1910.13796.

[3] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, Jan. 2012. DOI: 10.1145/3065386.

[4] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$ 0.5 mb model size," *CoRR*, 2016. arXiv: 1602.07360. [Online]. Available: http://arxiv.org/abs/1602.07360.

[5] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," *CoRR*, vol. abs/1611.05431, 2016. arXiv: 1611.05431. [Online]. Available: http://arxiv.org/abs/1611.05431.

[6] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," *CoRR*, vol. abs/1807.11626, 2018. arXiv: 1807.11626. [Online]. Available: http://arxiv.org/abs/1807.11626.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, 2014. arXiv: 1409.4842. [Online]. Available: http://arxiv.org/abs/1409.4842.

[8] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," vol. 1, 886–893 vol. 1, 2005. DOI: 10.1109/CVPR.2005.177.

[9] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–, Nov. 2004. DOI: 10.1023/B:VISI.0000029664.99615.94.

[10] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," *Computer Vision-ECCV 2006*, vol. 3951, pp. 404–417, Jul. 2006. DOI: 10.1007/11744023_32.

[11] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *CoRR*, 2019. arXiv: 1901.06032. [Online]. Available: http://arxiv.org/abs/1901.06032.

[12] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *CoRR*, 2015. arXiv: 1512.00567. [Online]. Available: http://arxiv.org/abs/1512.00567.

[13] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015. arXiv: 1409.1556 [cs.CV].

[14] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," *CoRR*, 2016. arXiv: 1602.07261. [Online]. Available: http://arxiv.org/abs/1602.07261.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, 2015. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[16] F. N. Iandola and K. Keutzer, "Keynote: Small neural nets are beautiful: Enabling embedded systems with small deep-neural-network architectures," *CoRR*, 2017. arXiv: 1710.02759. [Online]. Available: http://arxiv.org/abs/1710.02759.

[17] A. Howard, M. Sandler, G. Chu, L. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," *CoRR*, 2019. arXiv: 1905.02244. [Online]. Available: http://arxiv.org/abs/1905.02244.

[18] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, 2018. arXiv: 1801.04381. [Online]. Available: http://arxiv.org/abs/1801.04381.

[19] M. Lin, Q. Chen, and S. Yan, "Network in network," *CoRR*, vol. abs/1312.4400, 2014.

[20] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," *CoRR*, 2017. arXiv: 1709.01507. [Online]. Available: http://arxiv.org/abs/1709.01507.

[21] P. Ramachandran, B. Zoph, and Q. V. Le, "Swish: A self-gated activation function," *arXiv: Neural and Evolutionary Computing*, 2017.

[22] *I.mx rt1060 crossover processors for consumer products*, IMXRT1060CEC, Rev. 1, NXP, 2020. [Online]. Available: https://www.nxp.com/docs/en/nxp/data-sheets/imxrt1060cec.pdf_2021.

[23] NXP. (). I.mx rt1060 crossover mcu with arm® cortex®-m7 core, [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/i-mx-rt-crossover-mcus/i-mx-rt1060-crossover-mcu-with-arm-cortex-m7-core:i.MX-RT1060. (accessed: 03.28.2021).

[24] Tensorflow. (). Tensorflow lite inference, [Online]. Available: https://www.tensorflow.org/lite/guide/inference. (accessed: 03.28.2021).

[25] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, May 2012.

[26] X. Yu, T. Liu, X. Wang, and D. Tao, "On compressing deep models by low rank and sparse decomposition," Jul. 2017, pp. 67–76. DOI: 10.1109/CVPR.2017.15.

[27] Y. Gong, L. Liu, M. Yang, and L. D. Bourdev, "Compressing deep convolutional networks using vector quantization," *CoRR*, 2014. arXiv: 1412.6115. [Online]. Available: http://arxiv.org/abs/1412.6115.

[28] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," *CoRR*, 2015. arXiv: 1512.06473. [Online]. Available: http://arxiv.org/abs/1512.06473.

[29] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1510.00149.

[30] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015. [Online]. Available: http://arxiv.org/abs/1503.02531.

[31] P. S. P. Kavyashree and M. El-Sharkawy, "Compressed mobilenet v3:a light weight variant for resource-constrained platforms," in *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, 2021, pp. 0104–0107. DOI: 10.1109/CCWC51732.2021.9376113.

[32] G. Li, M. Zhang, Q. Zhang, Z. Chen, W. Liu, J. Li, X. Shen, J. Li, Z. Zhu, and C. Yuen, "Psdnet and dpdnet: Efficient channel expansion, depthwise-pointwise-depthwise inverted bottleneck block," *CoRR*, 2019. arXiv: 1909.01026. [Online]. Available: http://arxiv.org/abs/1909.01026.

[33] J. Kukacka, V. Golkov, and D. Cremers, "Regularization for deep learning: A taxonomy," *CoRR*, 2017. arXiv: 1710.10686. [Online]. Available: http://arxiv.org/abs/1710.10686.

[34] I. Loshchilov and F. Hutter, "SGDR: stochastic gradient descent with restarts," *CoRR*, 2016. arXiv: 1608.03983. [Online]. Available: http://arxiv.org/abs/1608.03983.

[35] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," *CoRR*, 2017. arXiv: 1710.09282. [Online]. Available: http://arxiv.org/abs/1710.09282.