Image Classification on NXP i.MX RT1060 using Ultra-thin MobileNet DNN

Saurabh Ravindra Desai
Department of ECE
Purdue School of Engineering and Tech.
Indianapolis, USA
saudesai@iu.edu

Debjyoti Sinha
Department of ECE
Purdue School of Engineering and Tech.
Indianapolis, USA
debsinha@iu.edu

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

Abstract—Deep Neural Networks play a very significant role in computer vision applications like image classification, object recognition and detection. They have achieved great success in this field but the main obstacles for deploying a DNN model into an Autonomous Driver Assisted System (ADAS) platform are limited memory, constrained resources, and limited power. MobileNet is a very efficient and light DNN model which was developed mainly for embedded and computer vision applications, but researchers still faced many constraints and challenges to deploy the model into resource-constrained microprocessor units. Design Space Exploration of such CNN models can make them more memory efficient and less computationally intensive. We have used the Design Space Exploration technique to modify the baseline MobileNet V1 model and develop an improved version of it. This paper proposes seven modifications on the existing baseline architecture to develop a new and more efficient model. We use Separable Convolution layers, the width multiplier hyperparamater, alter the channel depth and eliminate the layers with the same output shape to reduce the size of the model. We achieve a good overall accuracy by using the Swish activation function, Random Erasing technique and a choosing good optimizer. We call the new model as Ultra-thin MobileNet which has a much smaller size, lesser number of parameters, less average computation time per epoch and negligible overfitting, with a little higher accuracy as compared to the baseline MobileNet V1. Generally, when an attempt is made to make an existing model more compact, the accuracy decreases. But here, there is no trade off between the accuracy and the model size. The proposed model is developed with the intent to make it deployable in a real-time autonomous development platform with limited memory and power and, keeping the size of the model within 5 MB. It could be successfully deployed into NXP i.MX RT1060 ADAS platform due to its small model size of 3.9 MB. It classifies images of different classes in real-time, with an accuracy of more than 90% when it is run on the above-mentioned ADAS platform. We have trained and tested the proposed architecture from scratch on the CIFAR-10 dataset.

Index Terms—Deep Neural Network (DNN), Autonomous Driver Assistance Systems (ADAS), Design Space Exploration (DSE),Depthwise Separable Convolutions, Separable Convolutions, Random Erasing, Swish, Ultra-thin MobileNet, Tensorflow, Keras, CIFAR-10, i.MX RT1060 .

I. INTRODUCTION

In the last few years, Deep Neural Networks (DNN) has become a great technology for implementing computer vision applications like object detection, image classification, pose estimation, etc. They are playing an important role in fields like Robotics, Autonomous cars, Unmanned Aerial Vehicles, etc which often require the above-mentioned computer vision applications to be implemented in real-time. DNNs became popular when AlexNet [6] won the ImageNet Challenge, in 2012. Since then, deep learning architectures like SqueezeNet [1], SqueezeNext [2], Inception [4], MobileNet [3], etc have replaced computer vision algorithms like the HOG or Canny's algorithm. To make the architectures more accurate, they were made deeper by adding more layers to them. DNNs were implemented on different embedded systems but due to their large size, they were undeployable into resource- constrained real-time embedded devices. In general, making the DNN models deeper and more complex increases the model size and the average computation time, rendering them unsuitable for real-time applications. Various techniques to reduce the model size like Pruning [7], Deep compression [8], Architectural tuning were developed. The demand for computer vision applications in real-time embedded processors increased with time which led to the increased usage of the above-mentioned techniques to make the standard DNN architectures more compact and efficient. In the process of making these architectures smaller, the accuracy is often compromised. In this paper, we perform the Design Space Exploration (DSE) of the baseline MobileNet v1 which is a standard DNN architecture and get a more compact and faster model called the Ultra-thin MobileNet without compromising on the accuracy level. In Section I. we introduce our research work. In section II, we discuss some prior work associated to this domain. In section III, we discuss the development of the new DNN model step-by-step . Section IV discusses the various hardware and software requirements for our research. Section V throws light on the implementation part, starting from training and testing the architecture from scratch to deploying it into the NXP i.MX RT1060 MCU and visualizing the output on the Tera Term. Section VI depicts the experimental results we obtain after training, testing and deploying our model. We compare our model with other popular DNN models and also record the time taken for the image classification application. Finally, in Section VII, we conclude the topic by summarizing our work and also mentioning the future scope of this research.

II. PRIOR WORK

There are some deep neural networks which use depthwise separable convolutions and separable convolutions instead of the normal standard convolutions. Some of the deep neural networks using these convolutions are MobileNet V1 [3], MobileNet V2 [5], Enhanced Hybrid MobileNet [14] CondenseNet [11], ShuffleNet [8], and Xception [9]. The MobileNet V1 architecture utilizes the depthwise separable convolutions to decrease the average computation overhead to about one-eighth of the computation overhead incurred when standard convolutions are used. In a depthwise layer, each input channel is filtered separately and they are followed by a 1×1 pointwise convolution layer where all the outputs are linearly integrated. So, the process of filtering and the process of combining takes place in two different steps. MobileNet V2 [5] is an improved DNN architecture as compared to the MobileNet V1 [3] based on the model size. It utilizes bottleneck depthwise separable convolutions with residuals. It has got a 32 filter fully convolution layer, succeeded by 19 residual layers. The utilization of these residual layers prevents data from being mutilated and also makes the model more compact. The Enhanced Hybrid MobileNet [14] is also an improvement over the existing baseline model. In this model, the depth multiplier [14] hyperparameter was introduced and the average pooling layers were substituted by Max pooling and Fractional max pooling layers [14] with different strides. Different enhanced hybrid models were generated as the stride values were changed. Some achieved better accuracy and some achieved lesser model size as compared to the MobileNet V1. CondenseNet [11] uses some special convolutions called the group convolutions [8,11] mainly to reduce the computation cost by dividing the input features into mutually exclusive G groups, and they produce their outputs, ultimately decreasing the overall computational cost by a factor of G. Another model, the ShuffleNet [8] also uses these group convolutions. Each unit is composed of 1×1 pointwise convolution layers, channel shuffle, 3×3 depthwise convolutions, and 1×1 group convolutions. The channels in a group are divided into subgroups and then they are fed into the next layer with different subgroups, which makes the input channel and output channel perfectly related to each other. This model has very less number of MFLOPS compared to many other renowned DNN models. The Xception [9] model makes use of depthwise separable convolutions with residual connections. It was an improvement over the Inception V3 [4, 9] architecture, showing that it could achieve more gains in classification performance with the same number of parameters as the Inception module.

III. THE DEEP NEURAL NETWORK ULTRA-THIN MOBILENET

The Ultra-thin MobileNet is an enhanced MobileNet [3] architecture which has been developed by introducing some modifications on the existing baseline MobileNet V1 architecture. It is faster, more accurate and less memory intensive as compared to the baseline MobileNet V1 model. We propose some improvements to the existing baseline MobileNet V1 architecture to obtain the Ultra-thin MobileNet model that is better in terms of size, computation time, accuracy and is easily deployable in resource-constrained microprocessor units. These are the following:

TABLE I
NETWORK ARCHITECTURE AFTER INTRODUCING IMPROVEMENT 1

Layer / Strides	Output Shape	No. of Parameter
Input layer	32, 32, 3	0
Conv/s2	16, 16, 32	864
Separable Conv/s1	16, 16, 32	1312
Separable Conv/s2	8, 8, 64	2336
Separable Conv/s1	8, 8, 128	8768
Separable Conv/s2	4, 4, 128	17536
Separable Conv/s1	4, 4, 256	33920
Separable Conv/s2	2, 2, 256	67840
Separable Conv/s1	2, 2, 512	133376
Separable Conv/s1	2, 2, 512	133376
Separable Conv/s1	2, 2, 512	266752
Separable Conv/s1	2, 2, 512	266752
Separable Conv/s1	2, 2, 512	266752
Separable Conv/s2	1, 1, 512	266752
Separable Conv/s1	1, 1, 1024	528896
Global avg. pool/s1	1, 1, 1024	0
FC and Softmax/s1	1,1,10	10250

A. Improvement 1- Replacing Depthwise Separable convolutions [3, 5, 8, 14] by Separable convolutions[9]

Depthwise separable convolution [3, 5, 8, 14] contains depthwise convolution layers where each input channel is filtered individually, succeeded by pointwise convolutions where the outputs are integrated linearly. The depthwise convolution and the pointwise convolution layers are defined independently in the MobileNet v1 [3] model, but in our model, the depthwise layer and the pointwise layer are combined into one layer and defined as Separable convolutions [9]. In the python program, the depthwise initializer, depthwise regularizer, depthwise constraint, and pointwise initializer, pointwise regularizer, and pointwise constraint are defined inside the same init() function. There is no separate definition for the pointwise convolutions. The role of the separable convolutions is similar to the depthwise separable convolutions, the advantage being a reduction in the number of layers from 28 to 14, decreased model size, reduced total number of parameters and very less average computation-time-per-epoch as compared to the MobileNet V1 architecture. The model size is now 26.1 MB which is 13 MB less as compared to the baseline MobileNet V1 [3] size and the total number of parameters is 2.1 M which is 1.2 M less than the baseline MobileNet V1 [3]. The average computation time is 21s. TABLE I. shows the architecture after introducing this modification.

B. Improvement 2- Utilizing Random Erasing data augmentation technique [13, 17]

Data augmentation techniques [17] are used to augment the size of the training dataset by creating alternative versions of the images inside that dataset. In Random erasing [13] technique, we randomly select rectangular regions in an image I in a mini-batch, delete the pixels of that region and substitute

TABLE II
NETWORK ARCHITECTURE AFTER INTRODUCING IMPROVEMENTS 1,2,3 AND 4

Layer / Strides	Output Shape	No. of Parameter
Input layer	32, 32, 3	0
Conv/s2	16, 16, 32	864
Separable Conv/s1	16, 16, 32	1312
Separable Conv/s2	8, 8, 64	2336
Separable Conv/s1	8, 8, 128	8768
Separable Conv/s2	4, 4, 128	17536
Separable Conv/s1	4, 4, 256	33920
Separable Conv/s2	2, 2, 256	67840
Separable Conv/s2	1, 1, 512	133376
Separable Conv/s1	1, 1, 728	377344
Global avg. pool/s1	1, 1, 728	0
FC and Softmax/s1	1,1,10	7290

it with random values. When some portion of an item in an image is occluded, a model can fail to recognize the item from its global structure due to the absence of good generalization ability. This technique enhances the generalization ability of a model. In our model, the parameter values used are:

1. Probability of erasing p = 0.5.
2. Max. erasing area ratio $S_h = 0.02$.
3. Min. erasing area ratio $S_l = 0.4$.
4. Range of Erasing aspect ratio = [0.3,3.33].
5. Erasing aspect area ratio $r_e = 0.3$.

Area of the original image = S.

Erased area = S_e.

Erasing area ratio = $\frac{S_e}{S}$

As a result of this, the accuracy increases slightly and the overfitting decreases to some extent.

C. Improvement 3- Layers with similar output shape [3] are eliminated

We make our network shallow by removing the layers with redundant output shapes. The layers 9 to 13 in TABLE I. which have the output shape of (2,2,512) are eliminated. These layers contribute to 41% of the total number of parameters. The model size is now 9.9 MB which is almost 30 MB less than the size of baseline MobileNet v1.

D. Improvement 4- Changing the channel depth

We change the depth of the channel in the last separable convolution block from 1024 to 728 [9]. The block with the channel depth 1024 results in about 0.5 M parameters. When we modify the channel depth to 728, the number of parameters reduces by 0.2 M, which further reduces the model size to 8 MB.

TABLE II. shows the resulting network after introducing the improvements 1, 2, 3 and 4.

E. Improvement 5- Substituting Swish activation function [16, 19, 20] instead of ReLU [16, 19]

ReLU is a standard non-linear activation function that performs better than most of the other activation functions for almost all the DNN models trained on any dataset. The baseline MobileNet v1 uses ReLU [16, 19] as the activation function. If we replace it by the Swish activation function [15, 18, 19], we get a better accuracy which is 85.60%. It is a smooth, non-monotonic, lower bounded and upper unbounded activation function. It is mathematically defined as:

$$f(x) = x.\sigma(\beta.x) = \frac{x}{1+e^{-x}}$$

σ = Sigmoid function and β = Trainable parameter. When, the value of $\beta = 0$, Swish becomes a linear function that is $f(x) = \frac{x}{2}$. When $\beta \to$ Swish starts behaving like the standard ReLU activation function. Our model has the best accuracy when we set $\beta = 1$. In this case, Swish becomes the Sigmoid-weighted Linear Unit(SiL) [20, 23]. Fig. 1. shows the use of the Swish activation in our model.

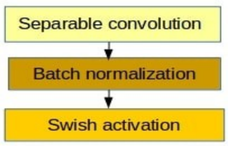


Fig. 1. Block with Separable convolution followed by Batch normalization and Swish activation

TABLE III
THE EFFECT OF USING DIFFERENT WIDTH MULTIPLIER VALUES ON THE MODEL

Width multiplier α	Model size	Accuracy	Parameters
0.75	4.8 MB	85.15%	3,76,270
0.69	3.9 MB	84.32%	3,19,095
0.60	3.0 MB	83.99%	2,43,167
0.50	2.2 MB	81.86%	1,72,130

F. Improvement 6- Setting the width multiplier [3, 5, 15] hyperparameter value

The width multiplier α [3, 5, 14] is a hyperparameter that is used to make a model slim uniformly at each layer. It reduces the number of input and output channels and makes the model thinner and faster. The width multiplier α should have values in the range 0 to 1. The total number of parameters decreases quadratically [3] as we reduce the value of α from 1 to 0. TABLE III. shows the model size, accuracy and the number of parameters we get with different values of α. It is evident from the table that as we decrease the value of the width multiplier, the accuracy starts to fall and it goes below the baseline MobileNet V1 accuracy which is 84.30%.

We observe, that as we reduce the value of α, the accuracy decreases and falls below the baseline level accuracy [3] of 84.30%. After putting different values of α in our python code, we discover that when $\alpha = 0.69$, we get an accuracy of 84.32% which is a little above the baseline accuracy. We are keeping the value of $\alpha = 0.69$ because we want to develop a network that has less model size and computation time than the baseline MobileNet with almost the same level of accuracy. In the process, the overfitting problem reduces to a large extent. We call this DNN model as the Ultra-thin MobileNet. TABLE IV. shows the Ultra-thin MobileNet architecture with all the

TABLE IV
ULTRA-THIN MOBILENET

Layer / Strides	Output Shape	No. of Parameter
Input layer	32, 32, 3	0
Conv/s2	16, 16, 22	594
Separable Conv/s1	16, 16, 22	682
Separable Conv/s2	8, 8, 44	1166
Separable Conv/s1	8, 8, 88	4268
Separable Conv/s2	4, 4, 88	8536
Separable Conv/s1	4, 4, 176	16280
Separable Conv/s2	2, 2, 176	32560
Separable Conv/s2	1, 1, 353	63712
Separable Conv/s1	1, 1, 502	180383
Global avg. pool/s1	1, 1, 502	0
FC and Softmax/s1	1,1, 10	5030

layers, the output shape, and parameters associated with each layer.

G. *Improvement 7- Choosing a good optimizer*

We obtain the best accuracy by using Nadam [20] as the optimizer. Nadam [20] combines the effect of RMSProp [21], Adam [12, 24] and Nesterov momentum [10, 23]. It is an efficient optimizer which interrupts the search in the direction of oscillations and speeds up the search towards the direction of the minima. It is better than Momentum [27] and SGD [25] optimizers as it does not drastically overshoot around the minima.

IV. HARDWARE AND SOFTWARE REQUIREMENTS

(a) NVIDIA Geforce RTX 1080Ti GPU
(b) Intel i9 9th generation processor (32GB RAM)
(c) NXP i.MX RT1060 MCU
(d) Anaconda Navigator 2.0
(d) Python IDE- Spyder v3.6
(e) Keras v2.2.0
(f) Tensorflow-gpu v1.11.0
(g) Livelossplot package from PyPI
(h) MCU Xpresso SDK
(i) Microsoft Visual Studio Code IDE
(j) Tera Term

V. IMPLEMENTATION

A. *Developing the DNN model*

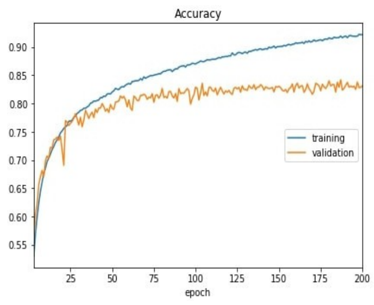


Fig. 2. MobileNet v1 baseline accuracy plot

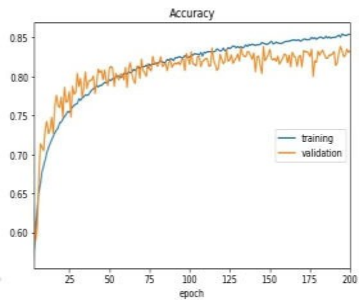


Fig. 3. Ultra-thin MobileNet accuracy plot

TABLE V
ULTRA-THIN MOBILENET V1 FEATURES

Accuracy	84.32%
Model size	3.9 MB
Avg. Computation time per epoch	16s
No. of parameters	3,19,095

We have trained the baseline MobileNet V1 from scratch over the CIFAR-10 dataset in 32 batches and 1563 steps- per-epoch for 200 epochs. The model size is 39.1 MB, the accuracy obtained is 84.30%, the number of parameters is 3.2 M and the average computation time is 31s. Fig. 2. shows the accuracy plot for the baseline version. The parameters are saved and loaded from a checkpoint file. Then we introduce the above-mentioned improvements into the baseline version to obtain an enhanced DNN which we call as the Ultra-thin MobileNet. It is trained for 200 epochs in 32 batches and the accuracy and the loss updates after each epoch is viewed graphically using the livelossplot package. NVIDIA Geforce RTX 1080Ti GPU is the hardware used for training. It is also tested on the CIFAR-10 dataset for class prediction. Fig 3. shows the plot of accuracy for the Ultra-thin MobileNet. TABLE V. depicts the features of this efficient network.

B. *Converting Model to TensorFlow Lite*

The NXP eIQ is a machine learning software development environment to develop machine learning applications for embedded processors such as i.MX RT crossover processors. The eIQ software includes inference engines, neural network compilers, and optimized libraries. TensorFlow Lite is one of the inference engines supported by eIQ software with high performance and optimized memory utilization than Tensor-Flow. TFLiteConverter takes an existing model in the keras framework and generates the TensorFlow Lite FlatBuffer file (.tflite). The Python API for TFLiteConverter allows custom objects such as activation functions, loss functions, etc to be passed during the conversion process. The IDE used for this conversion process is Microsoft Visual Studio Code (VS Code).

C. *Deployment on NXP i.MXRT1060*

eIQ software is delivered as middleware in the latest MCUXpresso SDK for NXP i.MXRT 1060. The MCUXpresso

SDK includes updated eIQ software platform and demos. Package contains a label image example for TensorFlow Lite, which is imported in MCUXpresso. The SDK also supports UART debug Console to debug the application on the TeraTerm serial emulator. This example runs a deep learning model on incoming frames from a camera for image classification. TensorFlow Lite FlatBuffer file (.tflite) is then converted into a C array header file (.h) that can be imported into an embedded project. This header file is then used to load the model in the code using API call. Application is then built to compile the code and create an executable for the i.MX platform. Fig. 4. is a snapshot of the i.MX RT1060 hardware we are using to run our model. The application is then debugged on the TeraTerm for output.

VI. RESULTS

TABLE V shows that the Ultra-thin MobileNet model is better than the baseline MobileNet model in terms of size, accuracy, and computation time (average time cost per epoch). TABLE VI compares our architecture with the other architectures like the MobileNet V2, Effnet [28], ShuffleNet 2x [29], Inception V1 [4], Inception V3, Xception, etc in terms of accuracy and the average computation time per epoch when they are trained using the CIFAR-10 dataset. The Ultra-thin MobileNet achieves the highest accuracy of 84.32% among these models. Also, it has an average computation time of 16s which is better than all the other DNN models. TABLE VII compares our model with other DNN models like the MobileNet, MobileNet V2, Xception V1, ShuffleNet 2x, Inception V1 and Inception V3 models in terms of the total number of parameters(millions) when they are trained on the CIFAR-10 dataset. In this case, also, our model is having a fewer number of total parameters than all other DNN models compared to in the table. Our model has about 0.3M parameters which is very less than the number of parameters in every other DNN model. Fig. 5. demonstrates the results of the image classification application on the NXP i.MX RT1060 board when the Ultra-thin MobileNet DNN model is run on it. The results are visible on the TeraTerm terminal emulator. The CIFAR 10-dataset mainly has 10 classes of images. The processor recognizes these classes of images and classifies them into one of the 10 categories in approximately 120ms (average inference time). For each category, the processor assigns a probability value and the highest probability is displayed as the recognized category. For example, if we feed in, the image of a ship, the TeraTerm serial emulator shows the inference time in milliseconds first and then shows the detected category as a ship with a percentage value.

VII. CONCLUSION

We have developed a new DNN architecture called the Ultra-thin MobileNet by introducing some improvements to the baseline MobileNet architecture. To make the model compact, we have used Separable convolutions instead of Depthwise Separable Convolution layers, eliminate the layers with redundant output shape, change the channel depth at the last layer and use the width multiplier hyperparameter with an optimum value of α=0.69. The value of α=0.69 is crucial here, as it reduces the size drastically, eradicates the overfitting problem, without allowing the overall accuracy of the model to fall below the baseline level. The interesting feature of this model is that when we make the model more compact, the accuracy of the model does not decrease due to the use of the Swish activation function, Random Erasing data augmentation method and Nadam optimizer. They play a very important role in maintaining the accuracy above the 84.30% mark which is the baseline MobileNet V1 accuracy. Generally, models with less depth are not very accurate as compared to the models with high depth. The size of the model is only 3.9 MB which makes it deployable into resource-constrained real-time microprocessors. The model also has a competitive accuracy of 84.32% which makes it safe and reliable for real-time image classification applications. The model is deployed into the NXP i.MX RT1060 MCU for the image classification application. This paper also proposes a method to convert a DNN image classification model trained with the Keras framework into a TensorFlow Lite format model, which is compatible with the NXP i.MX RT crossover processors. The average inference time to classify an image is 120ms with the best inference time being 115ms. The quick response time of

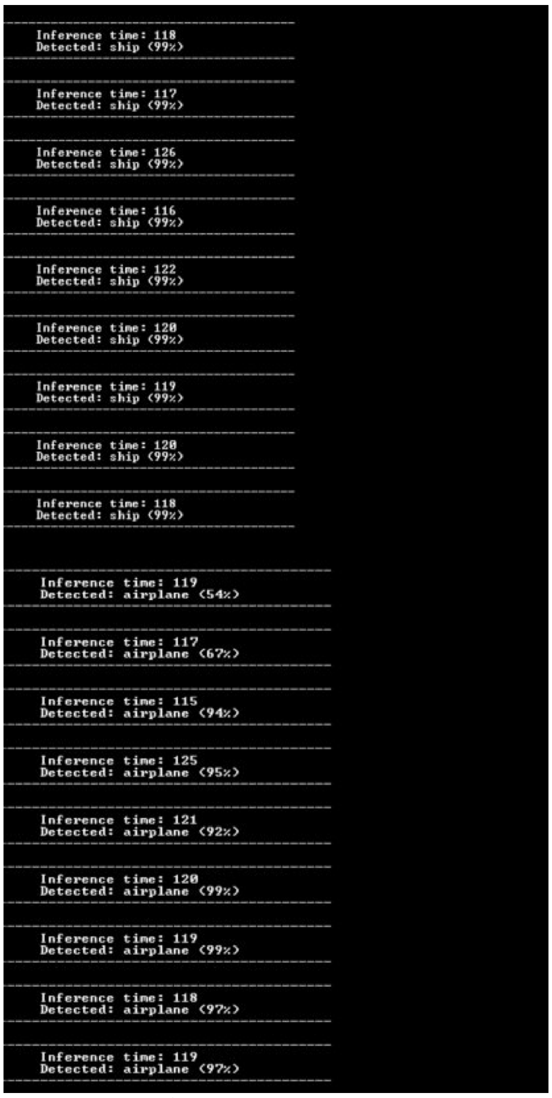
TABLE VI
COMPARISON WITH OTHER DNNS BASED ON ACCURACY AND THE COMPUTATION TIME ON THE CIFAR-10 DATASET

Model	Accuracy	Computation time
Ultra-thin MobileNet	84.32%	16s
MobileNet	84.30%	31s
MobileNet V2	73.98%	18s
Inception V1	75.21%	63s
Inception V3	79.76%	61s
EffNet	80.20%	24s
Xception	75.10%	58s

TABLE VII
COMPARISON WITH OTHER DNNS BASED ON THE TOTAL NUMBER OF PARAMETERS (MILLIONS) ON THE CIFAR-10 DATASET

Model	Parameters
Ultra-thin MobileNet	0.3 M
MobileNet	3.2 M
MobileNet V2	2.3 M
Xception V1	22.8 M
ShuffleNet 2x	5.2 M
Inception V1	6.6 M
Inception V3	23.6 M

Fig. 4. NXP i.MX RT1060 board



```
----------------------------------------
    Inference time: 118
    Detected: ship (99%)
----------------------------------------

----------------------------------------
    Inference time: 117
    Detected: ship (99%)
----------------------------------------

----------------------------------------
    Inference time: 126
    Detected: ship (99%)
----------------------------------------

----------------------------------------
    Inference time: 116
    Detected: ship (99%)
----------------------------------------

----------------------------------------
    Inference time: 122
    Detected: ship (99%)
----------------------------------------

----------------------------------------
    Inference time: 120
    Detected: ship (99%)
----------------------------------------

----------------------------------------
    Inference time: 119
    Detected: ship (99%)
----------------------------------------

----------------------------------------
    Inference time: 120
    Detected: ship (99%)
----------------------------------------

----------------------------------------
    Inference time: 118
    Detected: ship (99%)
----------------------------------------


---------------------------------------------
    Inference time: 119
    Detected: airplane (54%)
---------------------------------------------

---------------------------------------------
    Inference time: 117
    Detected: airplane (67%)
---------------------------------------------

---------------------------------------------
    Inference time: 115
    Detected: airplane (94%)
---------------------------------------------

---------------------------------------------
    Inference time: 125
    Detected: airplane (95%)
---------------------------------------------

---------------------------------------------
    Inference time: 121
    Detected: airplane (92%)
---------------------------------------------

---------------------------------------------
    Inference time: 120
    Detected: airplane (99%)
---------------------------------------------

---------------------------------------------
    Inference time: 119
    Detected: airplane (99%)
---------------------------------------------

---------------------------------------------
    Inference time: 118
    Detected: airplane (97%)
---------------------------------------------

---------------------------------------------
    Inference time: 119
    Detected: airplane (97%)
---------------------------------------------
```

Fig. 5. TeraTerm Output

this model on the above-mentioned processor makes it useful and reliable in real-time computer vision applications for autonomous vehicles. The future scope of this research involves the utilization of the DSE technique again and increasing the accuracy of the model beyond while keeping the size of the model the same. The model size can be reduced further with the help of techniques like Deep compression, Pruning and Architecture tuning, but it may lead to a compromise in the overall accuracy. The Ultra-thin MobileNet model can be deployed to other efficient processors of the NXP i.MX RT family, NXP Bluebox 2.0, etc and developing other interesting computer vision applications like real-time object detection, object tracking, object segmentation, etc with better inference times.

ACKNOWLEDGMENT

The authors would like thank NXP semiconductors and the IoT collaboratory under Purdue School of Engineering and Technology, IUPUI for supporting this research work.

REFERENCES

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

[2] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, Kurt Keutzer, (2018). SqueezeNext: Hardware-Aware Neural Network Design. arXiv preprint arXiv: 1803.10615

[3] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861

[4] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich (2014). Going Deeper with Convolutions. arXiv preprint-arXiv:1409.4842

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

[6] Md Zahangir Alom, Tarek M. Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S. Awwal, Vijayan K. Asari (2018). The History Began from AlexNet: A Comprehensive Survey on Deep Learning

[7] Ji Lin, Yongming Rao, Jiwen Lu, Jie Zhou (2017). Runtime Neural Pruning. 31st Conference on Neural Information Processing Systems, Long Beach, CA, USA.

[8] Xiangyu Zhang, Xinyu Zhou, Mengxiao LinJian, Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile. arXiv preprint arXiv:1707.01083v2, 2017

[9] Francois Chollet (2017). Xception: Deep Learning with DepthwiseSeparable Convolutions. arXiv preprint arXiv: 1610.02357

[10] Ilya Sutskever, James Martens, George Dahl, Georey Hinton (2013). On the importance of initialization and momentum in deep learning.http://proceedings.mlr.press/v28/sutskever13.pdf

[11] Gao Huang, Shichen Liu, Laurens van der Maaten. CondenseNet: An Efficient DenseNet using Learned Group Convolutions. arXiv preprint arXiv: 1711.09224, 2017

[12] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014

[13] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, Yi Yang (2017).Random Erasing Data Augmentation. arXiv preprint arXiv: 1708.04896

[14] Hong-Yen Chen and Chung-Yen Su. An Enhanced Hybrid MobileNet. arXiv preprint arXiv: 1712.04698

[15] Prajit Ramachandran, Barret Zoph, Quoc V. Le (2017).Searching for Activation Functions. arXiv preprint arXiv: 1710.05941

[16] Abien Fred M. Agarap (2019). Deep Learning using Rectified Linear Units (ReLU). arXiv preprint arXiv: 1803.08375v2

[17] Barret Zoph, Ekin D. Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, Quoc V (2019). Le. Learning Data Augmentation. arXiv preprint arXiv: 1906.11172

[18] Steffen Eger, Paul Youssef, Iryna Gurevych (2019). Is it Time to Swish? Comparing Deep Learning Activation Functions Across NLP tasks. arXiv preprint arXiv: 1901.02671

[19] Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall (2018). Activation Functions: Comparison of Trends in Practice and Research for Deep Learning. arXiv preprint arXiv: 1811.03378v1

[20] Timothy Dozat (2016). INCORPORATING NESTEROV MOMENTUMINTO ADAM. Workshop track - ICLR 2016.

[21] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradientby a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 4(2), 2012.

[22] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning. arXiv preprint arXiv: 1702.03118, 2017

[23] Aleksandar Botev, Guy Lever, David Barber (2016). Nesterovs Accelerated Gradient and Momentum as approximations to Regularised Update Descent. arXiv preprint arXiv: 1607.01981

[24] Timothy Dozat (2016). INCORPORATING NESTEROV MOMENTUM INTO ADAM. Workshop track - ICLR 2016.

[25] Yi Zhou, Junjie Yang, Huishuai Zhang, Yingbin Liang, Vahid Tarokh. SGD CONVERGES TO GLOBAL MINIMUM IN DEEP LEARN-

INGVIA STAR-CONVEXPATH. arXiv preprint arXiv :1901.00451, 2019

[26] Song Han, Huizi Mao, William J. Dally (2015). Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding- ICLR 2015

[27] Ilya Sutskever, James Martens, George Dahl, Geoffrey Hinton. On the importance of initialization and momentum in deep learning. ICML'13 Proceedings of the 30th International Conference on International Conference on Machine Learning.

[28] Ido Freeman, Lutz Roese-Koerne, Anton Kummert. EffNet: AN EFFICIENT STRUCTURE FOR CONVOLUTIONAL NEURAL NETWORKS. arXiv preprint arXiv: 1801.06434, 2018

[29] Xiangyu Zhang, Xinyu Zhou, Mengxiao LinJian, Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile. arXiv preprint arXiv:1707.01083v2, 2017