AN INTEGRATIVE ALGORITHM/ARCHITECTURE CO-DESIGN OF DEEP SPATIAL AND TEMPORAL SEPARABLE CONVOLUTIONAL NEURAL NETWORKS

by

Mohammadreza Baharani

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Electrical Engineering

Charlotte

2021

Approved by:

Dr. Hamed Tabkhi

Dr. Andrew Willis

Dr. Babak Parkhideh

Dr. Gary Teng

Copyright Notes

[1] http://www.ieee.org/publications_standards/publications/rights/rights_link.html

Abstract

MOHAMMADREZA BAHARANI. An Integrative Algorithm/Architecture Co-Design Of Deep Spatial and Temporal Separable Convolutional Neural Networks. (Under the direction of DR. HAMED TABKHI)

In this dissertation, I present my researches on the co-design of algorithms and architectures for deep spatial and temporal separable convolutional neural networks and their applications. As a first step, I will present Deep RACE as an application of Deep Neural Network (DNN) in the real-time reliability monitoring of transistors. Then, I will introduce DeepDive, a framework for enabling the execution of power-efficient spatial deep learning models on embedded FPGA. In addition, Agile Temporal Convolutional Network (ATCN) is proposed for fast time series prediction and classification in resource-constrained embedded systems. Finally, DeepTrack, which is based on ATCN, is introduced for vehicle trajectory prediction in highways. The significance of each of them is briefly explained below.

At first, this dissertation describes a novel approach, Deep Learning Reliability Awareness of Converters at the Edge (Deep RACE), for real-time reliability modeling and prediction of high-frequency MOSFET power electronic converters. Deep RACE offers a holistic solution that comprises algorithm advances, and full system integration (from the cloud down to the edge node) to create a near real-time reliability awareness. On the algorithm side, I propose a deep learning algorithmic solution based on stacked LSTM for collective reliability training and inference across collective MOSFET converters based on device resistance changes. Deep RACE also proposes an integrative edge-to-cloud solution to offer scalable decentralized devices-specific reliability monitoring, awareness, and modeling. The MOSFET convertors are IoT devices that have been empowered with edge real-time deep learning processing capabilities. The proposed Deep RACE solution has been prototyped and implemented through learning from the MOSFET data set provided by NASA. Our experimental

results show an average miss prediction of 8.9% over five different devices which is a much higher accuracy compared to well-known classical approaches (Kalman Filter, and Particle Filter). Deep RACE only requires $26mS$ processing time and $1.87W$ computing power on edge IoT devices.

Then, this dissertation introduces DeepDive, which is a fully-functional, vertical co-design framework, for power-efficient implementation of Deep Separable Convolutional Neural Networks (DSCNNs) on edge FPGAs. DeepDive's architecture supports crucial heterogeneous Compute Units (CUs) to fully support DSCNNs with various convolutional operators interconnected with structural sparsity. It offers FPGA-aware training and online quantization combined with modular synthesizable C++ CUs, customized for DSCNNs. The execution results on Xilinx's ZCU102 FPGA board, demonstrate 47.4 and 233.3 FPS/Watt for MobileNet-V2 and a compact version of EfficientNet, respectively, as two state-of-the-art depthwise separable CNNs. These comparisons showcase how DeepDive improves FPS/Watt by 2.2× and 1.51× over Jetson Nano high and low power modes, respectively. It also enhances FPS/Watt by about 2.27×.

Next, this dissertation presents a scalable deep learning model called ATCN for high-accurate fast classification and time series prediction in resource-constrained embedded systems. ATCN is primarily designed for mobile embedded systems with performance and memory constraints, such as wearable biomedical devices and real-time reliability monitoring systems. It makes fundamental improvements over the mainstream temporal convolutional neural networks, including the incorporation of separable depth-wise convolution to reduce the computational complexity of the model and residual connections as time attention machines, to increase the network depth and accuracy. The result of this configurability is that the ATCN becomes a family of compact networks with formalized hyperparameters that enable application-specific adjustments to be made to the model architecture. As part of the present work, three

ATCN families, namely T0, T1, and T2, are also presented. T0 and T1 are compiled and executed on the Cortex-M7 microcontroller, and all three models are executed on the Cortex-A57 processor. An evaluation of the accuracy and execution performance of the three models against the best-in-class InceptionTime shows that ATCN can not only improve accuracy but also enable time series classification on microcontrollers and improve the execution time on legacy microprocessors.

Intelligent transportation systems that have to perform precise trajectory prediction are vital; however, model complexity and memory footprint of these smart systems are also critical factors as they are generally deployed at the edge. Towards this end, I will present DeepTrack, a model based on ATCN that has better or comparable accuracy to existing models, but is smaller and has a lower computational complexity suitable for embedded systems. In contrast to previous methods, the vehicle dynamics are encoded using ATCNs rather than LSTMs, which are synonymous with time series analysis. According to experimental results, DeepTrack performed better than state-of-the-art trajectory prediction algorithms not only in terms of average displacement error but also in terms of MACs and model size as well.

ACKNOWLEDGEMENTS

Throughout this dissertation I had the support of my adviser Dr. Hamed Tabkhi, who helped me to complete this task. In addition to his positive attitude toward research, he helped me believe that I could achieve my goals.

My thanks also go to my dissertation committee members, Dr. Andrew Willis, Dr. Babak Parkhideh, and Dr. Gary Teng.

It's been an honor to have support from my peers at UNC-Charlotte. There have been so many amazing people in the course of my daily life that I might not have been able to name them all. I wish to extend my most sincere thanks to Mehrdad Biglarbaigian, Ushma Sunil Bharucha, Kaustubh Manohar Mhatre, Steven Furgurson, Vinit Katariya, and many others in the TeCSAR Lab for unforgettable memories.

DEDICATION

I would like to dedicate this thesis to my lovely wife, Maryam.

Throughout graduate school and life for the past eight years, Maryam has been an invaluable source of support and encouragement for me. My life would not be the same without you. I also like to dedicate this work to my parents, Mohammad and Esmat, whose unwavering love has always been a source of motivation and whose good examples have inspired me to strive hard for the things that I aspire to achieve.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AI Artificial Intelligence.

ALU Arithmetic Logic Unit.

API Applications Programming Interface.

ATCN Agile Temporal Convolutional Network.

AVG Average.

AXI Advanced eXtensible Interface.

BN Batch-Normalization.

BPTT Backpropagation through time.

CNN Convolutional Neural Networks.

CPS Cyber-Physical Systems.

CPU Central Processing Unit.

CU Compute Unit.

DC Dilated Convolutions.

DMA Direct Memory Access.

DP Data-Path.

DSCNN Deep Separable CNN.

DW Depth-wise Convolutional.

EXP EXPansion.

FIFO First-In-First-Out

FMA Fused-Multiply-Add

FPGA Field Programmable Gate Array.

GPU Graphical Processing Unit.

GRU Gated Recurrent Unit.

GTCN Generic Temporal Convolutional Networks.

HLS High-level synthesis.

HPC High Performance Computing.

IRB Inverted Residual Block.

ISA Instruction Set Architecture.

LCB Linear Convolution Block.

LR Learning Rate.

LSTM Long Short-Term Memory.

NC Normal Convolutional.

OpenCL Open Compute Language.

PE Process Elements.

PL Programmable Logic.

PRJ PRoJection.

PS Processing System.

PW Point-wise Convolutional.

RACE Reliability Awareness of Converters at the Edge.

RCB Regular Convolution Block.

ReLU Rectified Linear Unit.

RNN Recurrent Neural Network.

RTL Register-transfer level

SDK Software Development Kit.

SE Squeeze and Excitation.

SIMD Single Instruction Multiple Data.

SIMT Single Instruction Multiple Thread.

SMMU System Memory Management Unit.

SQ Squeeze and Excitation.

STCB Spectral-Temporal Convolution Block.

TCN Temporal Convolutional Networks.

TTC Time-to-Collision

V2I Vehicle-to-Infrastructure

V2V vehicle-to-vehicle

VIAC Vehicular Interactive Aware Convolution

VTA Versatile Tensor Accelerator.

CHAPTER 1: INTRODUCTION

1.1 Motivation

An edge computing node is a limited-resource hardware device that should meet the application timing constraints and run at low power. A few examples of edge computing in the real world are wearable devices[1, 2, 3, 4], smart transportation and cities [5, 6, 7], monitoring device health and the smart grid [8, 9, 10]. At the same time, we are also observing the huge success of Machine Learning (ML) algorithms in this domain, as well as their widespread implementation on the edge. Therefore, ML algorithms are challenging edge devices further due to their high computational complexity.

In this thesis, we are addressing the challenges imposed by ML on edge computing on both algorithm and hardware architectural design. In power electronics, we proposed Deep RACE [11, 12], an ML solution based on LSTM networks, to model transistor reliability in real-time in edge devices. After that, we developed a DeepDive [13] framework for optimizing and accelerating DSCNN on FPGA edges. Agile Temporal Convolutional Network (ATCN) [14] is also then developed as an agile model for time series analysis for embedded microcontrollers and processors. Lastly, we developed a lightweight deep learning model based on ATCN for predicting vehicle trajectory on highways. In reset of this section, we briefly introduced each of these approaches.

Despite the fact that power electronics systems are vital parts of the energy-conversion process, it is difficult to monitor their reliability. The problem was that mathematically formulating and precise understanding of the physical degradation in high-frequency power converters is notoriously difficult, due to the system soph-

istication and many unknown non-deterministic variables. To solve this problem, a wide range of stochastical diagnostic and prognostics techniques have been proposed to address the reliability issues of a complex system in the design, fabrication, and maintenance process. Kalman filter and Bayesian calibration are two examples of classical time series modeling and prediction techniques. However, these approaches are often bounded to first-order models in isolation and are not able to bring the collective behavior of many devices with the same underlying physic to create an accurate algorithmic construct. Therefore, their prediction accuracy is very limited. Moreover, they have very limited scalability for emerging advanced technologies [15, 16]. Therefore, based on the advances observed in DNN, I have decided to work on a transformative solution, which is called Deep learning Reliability Awareness of Converters at the Edge (Deep RACE) [11] for real-time reliability modeling and assessment of power semiconductor devices embedded into a wide range of smart power electronics systems. Deep RACE departures from classical learning and statistical modeling to deep learning based data analytics, combined with full system integration for scalable real-time reliability modeling and assessment. In this regard, it leverages the Long Short-Term Memory (LSTM) networks as a branch of Recurrent Neural Networks (RNN) to aggregate reliability across many power converters with similar underlying physic. Also, It offers real-time online assessment by selectively combining the aggregated training model with device-specific behaviors in the field.

In our deployment of DNN solutions [11, 12, 17, 18], on edge platforms, we discovered weak support for accelerating Deep Separable Convolutional Neural Networks (DSCNN) such as those of the EfficientNet[19] and MobileNet-V2[20] families. During this research, DeepDive [13] was developed, which was used for an agile, power-efficient execution of DSCNNs on edge FPGAs. DeepDive provides a novel algorithm/architecture for efficient execution of DSCNNs, as well as a vertical algorithm optimization and synthesis on edge FPGAs. This framework aims to identify hetero-

geneous Compute Units (CUs) that support heterogeneous convolutional operations of DSCNNs such as group, depthwise, and pointwise convolution. Using the FPGA-aware training and online quantization, DeepDive receives the network description model (PyTorch for example) and optimizes the model. This includes algorithm-specific fusing of batch normalization and convolutional operators, which reduces the computation by ∼4%, and extremely low-bit per-channel-quantization across all separable convolution layers. DeepDive relies on the recent advances in High-Level Synthesis (HLS) and shifts the optimization abstraction to pre-RTL design. Therefore, it can generate the host CPU code running on ARM cores in the Processing System (PS) side of SoC for synchronization and scheduling. The host code, bundled with a scheduler, enables DeepDive to support multiple run-time software stacks such as Pynq and Linux.

As we were designing the Deep RACE framework, we realized the LSTM networks suffer from two problems [21]: 1) gradient instability such as vanishing/exploiting gradients and 2) fewer levels of parallelization due to intercellular dependencies. Another factor to be considered is that AI-enabled edge platforms are more expensive than microcontrollers, which are usually used in edge devices. Therefore, in this thesis, we proposed ATCN, which is a novel algorithmic solution for real-time deep learning processing of time series on embedded and edge devices. ATCN presents a family of agile network architectures, which are constructed by chaining Spectral-Temporal Convolution Blocks (STCB). STCB improves the depth and accuracy of the network by utilizing residual connections as time attention machines and uses separable depthwise convolutions are used to reduce computational complexity. As part of the present work, three ATCN families, namely T0, T1, and T2, are also presented and evaluated on different ranges of embedded processors - Cortex-M7 and Cortex-A57 processor. Our evaluation on 70 benchmarks from the UCR 2018 dataset [22] indicates that The T0 configuration reduces the MACs and model size by 102.38×

and 16.84× over InceptionTime (IT) [23], respectively. T1 accuracy is 4.03% better than T0 and has a 73.59× reduction in MACs and a 14.23× reduction in model size over IT. Both T0 and T1 can be executed on an ARM Cortex-M7 microcontroller explained in the experimental section in detail. As a final improvement, T3 reduces the MACs and model size for 26.07× and 4.4× over IT while increasing accuracy by 0.37%.

In 2019, there were 36,096 fatalities on roadways in the United States [24, 25]. Of those fatal crashes, NHTSA (2019) estimates that 11.9% of them involved a vehicle maneuvering in a manner that may be unpredictable to the other drivers (i.e., turning left or right, stopping or slowing in traffic, merging/changing lanes, or passing another vehicle). Such crashes at highway speeds, given the short Time-to-Collision (TTC) and limited distance range, cannot be prevented with vision-based systems alone [26]. Providing enough time and distance to support effective crash avoidance via Vehicle-to-Vehicle (V2V) systems must also utilize high-accuracy predictions for changing vehicle trajectories. With the acquaintance of ATCN, we realized the domain of smart transportation can take a considerable benefit of its fast and agile implantation as most of the vehicle trajectory prediction models are based on LSTM [27, 28, 29, 30]. Therefore, we proposed DeepTrack [31] as a novel deep learning model with comparable accuracy to best-in-class trajectory prediction algorithms but a much smaller model size with lower computational complexity to suit the resource-crunched embedded edge systems. DeepTrack encodes the vehicle dynamics with the aid of ATCN instead of well-established LSTM units. Compared to CS-LSTM and its other variant CS-LSTM (M) [32], DeepTrack reduces prediction error by 9.09% and 11.56%, respectively, and reduces the number of operations and model size by about 10.49% and 18.5%, respectively. Concerning CF-LSTM [28], DeepTrack error raised for 0.87%; however, it can reduce the number of operations and model size by 10.49% and model size by 18.5%.

1.2 Contributions to the Body of Knowledge

Overall, this dissertation presents four fundamental contributions to the body of knowledge which are:

- We proposed Deep RACE, which is the first integrative solution for active reliability assessment of the high-frequency power converters based on real-time deep learning analytic. Deep RACE moves beyond mainstream device modeling and traditional reliability analysis by combining advanced sensing solutions with cutting-edge deep learning and edge computing techniques.

- This thesis proposes DeepDive as a novel scalable vertical framework for the execution of DSCNN on FPGAs. The vertical integration and library-based operation mapping enable true comprehensive design space exploration on FPGAs. To the best of our knowledge, this work is the first scalable solution with the support of recently introduced EfficientNet DSCNN families.

- We developed ATCN as a family of compound scaling networks that achieve higher or comparable accuracy over SotA networks with significantly lower computation complexity and model size. We Demonstrated the significant benefits of ATCN over SotA networks when it comes to execution on embedded IoT microcontrollers (ARM Cortex M7 and Cortex A57).

- DeepTrack is also developed on the base of ATCN as a novel deep learning model with comparable accuracy to best-in-class trajectory prediction algorithms but a much smaller model size with lower computational complexity to suit the resource-crunched embedded edge systems.

The rest of this thises as follows: Chapter 2 presents DeepRACE as a framework for transistor reliability modeling with the aid of LSTM. In chapter 3, DeepDive is presented for hardware acceleration of DSCNN on embedded FPGA. ATCN and its

family are explained in Chapter 4, while its application in smart transportation as DeepTrack is explained in Chapter 5. Finally, the conclusion is drawn in Chapter 6.

Bibliography

[1] X. Jin, L. Li, F. Dang, X. Chen, and Y. Liu, "A survey on edge computing for wearable technology," *Digital Signal Processing*, p. 103146, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1051200421001858

[2] N. J. Cooney, K. P. Joshi, and A. S. Minhas, "A wearable internet of things based system with edge computing for real-time human activity tracking," in *2018 5th Asia-Pacific World Congress on Computer Science and Engineering (APWC on CSE)*, 2018, pp. 26–31.

[3] K. M. Shahiduzzaman, X. Hei, C. Guo, and W. Cheng, "Enhancing fall detection for elderly with smart helmet in a cloud-network-edge architecture," in *2019 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*, 2019, pp. 1–2.

[4] T. Brunschwiler, J. Weiss, S. Paredes, A. Sridhar, U. Pluntke, S. M. Chau, S. Gerke, J. Barroso, E. Loertscher, Y. Temiz, P. Ruch, B. Michel, S. Zafar, and T. van Kessel, "Internet of the body - wearable monitoring and coaching," in *2019 Global IoT Summit (GIoTS)*, 2019, pp. 1–6.

[5] X. Qi, G. Mei, and F. Piccialli, "Resilience evaluation of urban bus-subway traffic networks for potential applications in iot-based smart transportation," *IEEE Sensors Journal*, pp. 1–1, 2020.

[6] O. Madamori, E. Max-Onakpoya, G. D. Erhardt, and C. E. Baker, "A latency-defined edge node placement scheme for opportunistic smart cities," in *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, 2021, pp. 142–147.

[7] E.-L. Tan, F. A. Karnapi, L. J. Ng, K. Ooi, and W.-S. Gan, "Extracting urban sound information for residential areas in smart cities using an end-to-end iot system," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[8] M. Baharani, M. Biglarbegian, B. Parkhideh, and H. Tabkhi, "Real-time deep learning at the edge for scalable reliability modeling of si-mosfet power electronics converters," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7375–7385, 2019.

[9] M. Biglarbegian, M. Baharani, N. Kim, H. Tabkhi, and B. Parkhideh, "Scalable reliability monitoring of gan power converter through recurrent neural networks," in *2018 IEEE Energy Conversion Congress and Exposition (ECCE)*, 2018, pp. 7271–7277.

[10] Y. Zhang, R. Xiong, H. He, and M. G. Pecht, "Long short-term memory recurrent neural network for remaining useful life prediction of lithium-ion batteries," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 7, pp. 5695–5705, 2018.

[11] M. Baharani, M. Biglarbegian, B. Parkhideh, and H. Tabkhi, "Real-time deep learning at the edge for scalable reliability modeling of si-mosfet power electronics converters," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7375–7385, 2019.

[12] M. Biglarbegian, M. Baharani, N. Kim, H. Tabkhi, and B. Parkhideh, "Scalable reliability monitoring of gan power converter through recurrent neural networks," in *2018 IEEE Energy Conversion Congress and Exposition (ECCE)*, 2018, pp. 7271–7277.

[13] M. Baharani, U. Sunil, K. Manohar, S. Furgurson, and H. Tabkhi, "Deepdive: An integrative algorithm/architecture co-design for deep separable convolutional neural networks," in *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, ser. GLSVLSI '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 247–252. [Online]. Available: https://doi.org/10.1145/3453688.3461485

[14] M. Baharani and H. Tabkhi, "Atcn: Resource-efficient processing of time series on edge," 2021.

[15] A. Saxena, J. Celaya, E. Balaban, K. Goebel, B. Saha, S. Saha, and M. Schwabacher, "Metrics for evaluating performance of prognostic techniques," in *Prognostics and health management, 2008. PHM 2008. international conference on*. IEEE, 2008, pp. 1–17.

[16] H. S.-h. Chung, H. Wang, F. Blaabjerg, and M. Pecht, *Reliability of power electronic converter systems*. Institution of Engineering and Technology, 2015.

[17] C. Neff, M. Mendieta, S. Mohan, M. Baharani, S. Rogers, and H. Tabkhi, "Revamp²t: Real-time edge video analytics for multicamera privacy-aware pedestrian tracking," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 2591–2602, 2020.

[18] M. Baharani, S. Mohan, and H. Tabkhi, "Real-time person re-identification at the edge: A mixed precision approach," in *Image Analysis and Recognition*, F. Karray, A. Campilho, and A. Yu, Eds. Cham: Springer International Publishing, 2019, pp. 27–39.

[19] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, 2019, pp. 6105–6114.

[20] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, 2018, pp. 2704–2713.

[21] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014.

[22] H. A. Dau, E. Keogh, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, Yanping, B. Hu, N. Begum, A. Bagnall, A. Mueen, G. Batista, and Hexagon-ML, "The ucr time series classification archive," October 2018, https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

[23] H. Ismail Fawaz, B. Lucas, G. Forestier, C. Pelletier, D. F. Schmidt, J. Weber, G. I. Webb, L. Idoumghar, P.-A. Muller, and F. Petitjean, "Inceptiontime: Finding alexnet for time series classification," *Data Mining and Knowledge Discovery*, vol. 34, no. 6, pp. 1936–1962, Nov 2020. [Online]. Available: https://doi.org/10.1007/s10618-020-00710-y

[24] (2019) Fatality Analysis Reporting System (FARS), Motor vehicle traffic crashes (1994-2019). [Online]. Available: https://www-fars.nhtsa.dot.gov/Main/index.aspx

[25] (2019) Fatality Analysis Reporting System (FARS), Vehicles involved in fatal crashes. [Online]. Available: VehiclesInvolvedinFatalCrashes,https://www-fars.nhtsa.dot.gov/Vehicles/VehiclesAllVehicles.aspx

[26] V. D. H. Richard and H. Jeroen, "Time-to-Collision and Collision avoidance systems," *International Co-operation on Theories and Concepts in Traffic safety (ICTCT)*, 1994.

[27] N. Deo and M. M. Trivedi, "Convolutional social pooling for vehicle trajectory prediction," in *2018 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, 2018, pp. 1468–1476.

[28] X. Xie, C. Zhang, Y. Zhu, Y. N. Wu, and S. Zhu, "Congestion-aware multi-agent trajectory prediction for collision avoidance," *CoRR*, vol. abs/2103.14231, 2021. [Online]. Available: https://arxiv.org/abs/2103.14231

[29] J. Mercat, T. Gilles, N. E. Zoghby, G. Sandou, D. Beauvois, and G. P. Gil, "Multi-head attention for multi-modal joint vehicle motion forecasting," in *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*. IEEE, 2020, pp. 9638–9644. [Online]. Available: https://doi.org/10.1109/ICRA40945.2020.9197340

[30] L. Lin, W. Li, H. Bi, and L. Qin, "Vehicle trajectory prediction using lstms with spatial-temporal attention mechanisms," *IEEE Intelligent Transportation Systems Magazine*, pp. 0–0, 2021.

[31] M. Baharani, V. Katariya, N. Morris, O. Shoghli, and H. Tabkhi, "Deeptrack: Lightweight deep learning for vehicle path prediction in highways," 2021.

[32] T. Phan-Minh, E. C. Grigore, F. A. Boulton, O. Beijbom, and E. M. Wolff, "Covernet: Multimodal behavior prediction using trajectory sets," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. IEEE, 2020, pp. 14 062–14 071. [Online]. Available: https://doi.org/10.1109/CVPR42600.2020.01408

CHAPTER 2: DEEP RACE

2.1 Introduction

Power electronics systems are essential components of the energy-conversion process. It is expected by 2030, power converters will be used in 80% of applications in the generation, transmission, distribution, and consumer electronics [1]. Controllable power semiconductor devices play the most dominant role in the switching power converters. Operating at high current and voltage creates extreme stresses on the power devices, which often makes them the most susceptible components in the energy conversion process. Therefore, understanding, modeling and predicting the reliability models of the power converters are crucial for enabling emerging technologies and future applications such as electric vehicles, smart grids, and renewable energy.

Mathematically formulating and precise understanding of the physical degradation in high-frequency power converters is notoriously difficult, due to the system sophistication and many unknown non-deterministic variables. To solve this problem, a wide range of stochastical diagnostic and prognostics techniques have been proposed to address the reliability issues of a complex system in the design, fabrication, and maintenance process. The evaluation of these processes is beneficial to enable power convertors health management systems and resiliency for useful life estimation and reducing the risk of failures [2, 3]. Kalman filter and Bayesian calibration are two examples of classical time series modeling and prediction techniques. However, these approaches are often bounded to first-order models in isolation and are not able to bring the collective behavior of many devices with the same underlying physic to create an accurate algorithmic construct. Therefore, their prediction accuracy is very limited. Moreover, they have very limited scalability for emerging advanced techno-

logies [4, 5].

Recent advances in deep learning open a new horizon toward smart and autonomous systems. Deep learning offers a scalable data-driven discriminative paradigm to understand, model and predict the behavior of complex systems by extracting the deep collective knowledge. With the new wave of the Internet-of-Things (IoT) and the feasibility of using the internet almost everywhere, there is a big chance for scalable device-specific real-time monitoring and analysis by pushing deep learning and advanced analytic computations from the cloud next to IoT devices (which also called edge computing) [6, 7]. In particular, the benefits of edge computing are much more pronounced for real-time reliability modeling and prediction of sophisticated physical and engineering systems such as power electronic converters.

This research presents a transformative solution, which is called Deep learning Reliability Awareness of Converters at the Edge (Deep RACE)[1], for real-time reliability modeling and assessment of power semiconductor devices embedded into a wide range of smart power electronics systems. Deep RACE departures from classical learning and statistical modeling to deep learning based data analytics, combined with full system integration for scalable real-time reliability modeling and assessment. In this regard, it leverages the Long Short-Term Memory (LSTM) networks as a branch of Recurrent Neural Networks (RNN) to aggregate reliability across many power converters with similar underlying physic. Also, It offers real-time online assessment by selectively combining the aggregated training model with device-specific behaviors in the field.

To guarantee real-time scalable requirements, the Deep RACE presents an integrated cloud-edge platform in which, the cloud is responsible to aggregate different device reliability by training the LSTM network, while the inference is done at the edge next to the power devices. The interference at the edge (on-line) provides real-

[1]The Deep RACE is an open source project and its code is available at `https://github.com/TeCSAR-UNCC/Deep_RACE`.

time feedback of the reliability modeling as well as active control and decision making for device proliferate. We have trained and implemented the proposed Deep RACE approach for five high-frequency MOSFET power converters. Our results demonstrate the Deep Race improves the misprediction by 1.98x and 1.77x compared to Kalman Filter and Particle Filter, respectively.

To the best of the author's knowledge, Deep RACE is the first integrative solution for active reliability assessment of the high-frequency power converters based on real-time deep learning analytic. In this context, this research moves beyond mainstream device modeling and traditional reliability analysis by combining advanced sensing solutions with cutting-edge deep learning and edge computing techniques. Although this chapter primarily focuses on the reliability modeling of high-frequency MOSFETs, the proposed algorithmic construct and system level solution for real-time reliability and predictive maintenance can be extended to a wide range of semiconductor devices and engineering systems used in power conversion and smart energy systems.

The rest of this chapter is organized as the following: Section 2.2 briefly reviews the previous reliability approaches. Section 2.3 provides background on deep learning in particular deep RNN. Section 2.4 presents our proposed Deep RACE approach including LSTM-based machine learning and system-level integration for aggregated training and real-time inference. Section 2.5 presents the experimental results including comparison with existing approaches, and finally Section 2.6 concludes this article.

2.2 Related Work

This section briefly reviews the previous reliability approaches in power electronics, and discusses precursor identifier for power MOSFET degradations.

2.2.1 Reliability analysis/prediction in power electronics

The reliability approaches in power electronics systems have been developed in four broad categories: a) component level, b) damage accumulation, c) data analytics, and d) condition-based predictions. The first approach is not a prognostic-based since they consider the unit-to-unit difference and not the usage history [8, 9]. The second method offers more accurate tendency for an individual unit by using the accumulation of stress conditions over the time, although it needs experimental observation for the modeling [10, 11]. Data analytics focus on big-data mining for prediction based on the past usage history data and assign a predictive score as opposed to calculating a time to failure events [12, 13]. Lastly, condition-based prognostic methods rely on potential mode identifications and finding the root of the failure mechanism based on the individual behavior units of failure model physics[14, 15].

Several methods have been proposed for mean-life estimations like six sigma, fault tree analysis, state space, and filtering estimations [16, 17]. Due to increasing the large volume of data collected from smart devices, using the existing methodologies have some limitations on extracting the hidden patterns. Therefore, the necessity of applying deep learning algorithms for real-time system health monitoring is crucial. Few recent approaches have already demonstrated the significant benefits of deep learning based reliability monitoring and predictive maintenance in other engineering disciplines [18, 19, 20].

2.2.2 Precursor identifications in power MOSFET degradation

In the most comprehensive industry survey-based studies, power semiconductor devices (e.g., MOSFET) are responsible for at least more than 30% of the failures [21, 22]. The failure mechanisms in power MOSFET can be categorized into two extrinsic and intrinsic subcategories. The extrinsic failures include the transistor packaging issues and mainly summarized as a bond-wire lift, die solder detachment,

and contact migration. Most of these studies verified that the bond-wire lift has a severe effect on the device failure over time [23, 24].

To evaluate device long-term reliability, the general approach is conducting an accelerated life test under power/thermal cycles, and continuously monitoring variations in electrical or mechanical parameters. Based on the most acceptable standards for device qualification in the industry, such as AEC-Q101 [25], and the state-of-the-art research, I_{dss}, T_j, V_{th}, R_{th}, and $R_{ds(on)}$ are the most common parameters for the device degradation tracking [8, 16, 26, 27]. I_{dss}, which refers to drain current at zero bias, can be used for early detection of die-level failures, T_j shows the device junction temperature and corresponds to thermal runaway failures, V_{th} shows the gate threshold voltage shifting, R_{th} is the thermal resistance of the device and represents device overheating mostly in the package level. Finally, $R_{ds(on)}$ shows the device drain-source resistance, which represents both device degradation in the die and package level where inherently shows the device internal loss.

Fig. 2.1 illustrates the changes of $R_{ds(on)}$ over time for five different MOSFET transistors (IRF520NPbf) extracted from the data set provided by NASA [28]. Although it may seem that these five transistors share a similar degradation pattern at the first observation, the deterioration pace and detailed behaviors are significantly varied across the devices with similar underlying physics. This is primarily due to diverse workloads, different environmental conditions, and varying manufacturing processes. In the next section, we discuss why the classical approaches are infertile to concurrently model the degradation of these five transistor devices. For the rest of this chapter, we also consider these five data set to evaluate the performance of our proposed Deep RACE framework.

2.3 Motivation and RNN Background

In this section we explain the limitation of classical approaches for modeling of power device degradation. Then, we continue to elaborate more on vanilla RNN

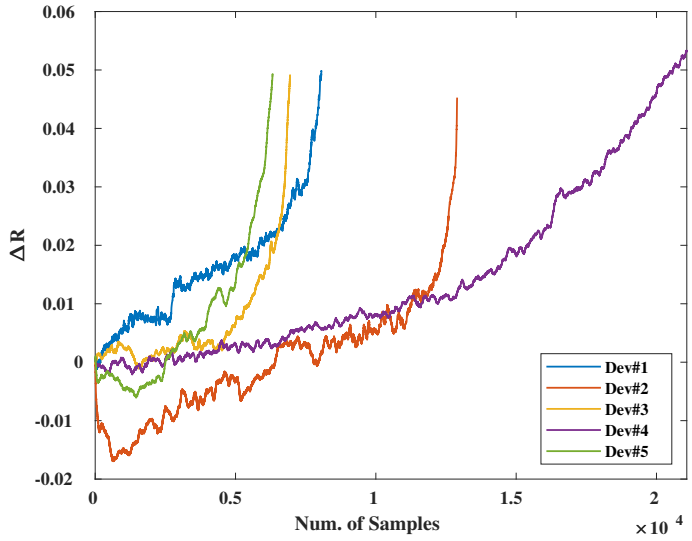


Figure 2.1: MOSFET $\Delta R_{ds(on)}$ Precursor Identifier: The trajectory $R_{ds(on)}$ for five different MOSFET devices.

and its problem regarding modeling a complex time series data such as MOSFET $\Delta R_{ds(on)}$ precursor.

2.3.1 Limitation of Classical Approaches

In the reliability-based prediction methods, the frequency of component failure is predicted based on a statistical model derived from acquired data in a laboratory environment or historical component usage if available. In a strict sense, these methods cannot be considered as prognostic methods due to unique unit-to-unit conditions and their specific usage history. Alongside, although theoretical approaches such as physics-of-failure [29] analysis are applied to identify the root of failure and drive the reliability model, these methods result in significant errors and are not applicable for unit-to-unit scenarios because of the complexity of power electronics systems and their operating conditions [16].

The other data-driven approaches, such as Kalman filter [30] and Bayesian calibration [13, 28], are predictive analytics which mostly focus on identifying the correlation of the experimental results, and the estimation of unknown variables and parameters. These techniques require an accurate failure model of a system to estimate the un-

known mathematical parameters associated with a specific failure test; however, for new technologies, these methods cannot be effective due to the lack of precise failure model in the component as well as system level [15]. There is a high demand for solutions that address the algorithmic and system-level challenges for real-time scalable monitoring, modeling, and estimation of degradation behavior in power electronic transistors.

2.3.2 Recurrent Neural Networks

In contrast to classical approaches, this article proposes a holistic IoT system for hybrid condition-based prediction model which assesses the behavior of individual transistors based on both their usage history as inferred from sensed data and expected future load profiles. To achieve this, a data-driven model based on deep Recurrent Neural Network (RNN) is utilized at the edge, i.e. converter, for real-time device-specific reliability prediction and modeling; while the cloud infrastructure performs high-level metadata aggregation and analysis across many devices. At the time of Artificial Intelligence (AI) big-bang, we took advantages of a prominent model of RNN named LSTM to predict the transistor degradation. In our solution, the sensed resistance, and environment conditions will be sent to the cloud-side of the proposed framework to train and update the LSTM network models. Therefore, the models will be updated based on the current device operating condition.

The RNNs are a branch of neural networks specialized for analyzing and modeling complex time series. Following deep learning paradigm, RNNs need a fairly large dataset for training. The are very popular approach for natural language processing and object tracking over frames sequence. In a formalized RNN, sequence of data is notated by $X = [x_1 \quad x_2 \quad ... \quad x_\tau]$ where τ is the number of input sequences. Fig. 2.2 formulates an RNN computation node, i.e. neuron, and its unrolling recurrent computation when $\tau = 4$. A neuron passes the information from the past to the current time by sharing the information and updating its cell state, c; therefore, this

sharing process enables RNN to model a behavior of a time sequence. In a standard RNN cell with given input X, the cell output $Z = [z_1 \quad z_2 \quad ... \quad z_\tau]$ is computed by Equations 2.1-2.4. In these equations, $\zeta(\cdot)$ and $\xi(\cdot)$ are nonlinear activation functions, W_i is input weight, W_c is the state cell weight, W_o is the output weight, and b_i, b_o are biases for input and output values, respectively.

$$i_t = W_i x_t + W_c c_{t-1} + b_i, \tag{2.1}$$

$$c_t = \zeta(i_t), \tag{2.2}$$

$$o_t = W_o c_t + b_o, \tag{2.3}$$

$$z_t = \xi(o_t). \tag{2.4}$$

Knowing $Y = [y_1 \quad y_2 \quad ... \quad y_\tau]$ as the referenced output, the loss function is defined by (2.5). In (2.5), $\mathcal{L}(z_t, y_t)$ can be considered as the squared error of a regression function or cross-entropy for the sake of classification. The ultimate goal of RNN learning is to minimize the cost function. This goal can be formalized by (2.6), which minimizes the introduced loss function by altering θ, where θ is a network vector model which is described as: $\theta = [W_i \quad W_c \quad W_o \quad b_i \quad b_o \quad c_0]$.

$$\mathcal{L}(Z, Y) = \sum_{t=1}^{\tau} \mathcal{L}(z_t, y_t), \tag{2.5}$$

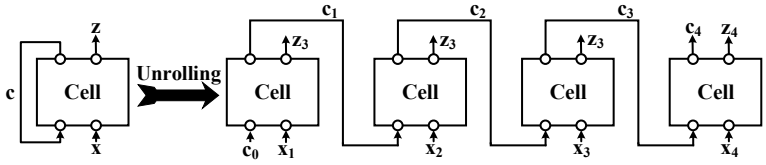


Figure 2.2: Recurrent Neural Networks: The schematic of standard RNN cell and its unrolling version for four input time sequence

$$\underset{\theta}{\operatorname{argmin}}\ \mathcal{L}(z(\theta), Y). \tag{2.6}$$

Backpropagation through time (BPTT) is the mainstream approach to extract the proper weight factors of the RNNs to minimize the cost function across the trained data [31, 32]. For the vanilla RNN, a major issues associated with BPTT is the losing of cell sensitivity to the earliest inputs due to the chain of partial derivation. This phenomena is known as vanishing gradient problem and eventually prevents the network reaches to the earliest states in deep RNN [33]. More sophisticated versions of RNN network, such as LSTM cells, are proposed to address the sensitivity lost problem in the basic RNN networks. Based on that, in the next section, we describe our proposed reliability model based on the LSTM networks.

2.4 Deep Learning Reliability Awareness of Converters at the Edge (Deep RACE)

This section presents Deep RACE as an integrative framework of online real-time reliability awareness and modeling for power electronic devices. Deep RACE has two major aspects: (1) algorithmic principles for modeling the device reliability, and (2) system-level integration for real-time scalable reliability assessment. On the algorithm side, Deep RACE uses one of the major derivatives of RNN, called LSTM for aggregated training and device specific inference. On the system side, Deep RACE proposes an IoT-based edge-cloud computation platform which pushes the proposed LSTM-based reliability inference next to the individual power converters. In the following, we provide an in-depth explanation of both aspects.

2.4.1 Algorithmic Constructs for Device Reliability Modeling

In this part, we first introduce the basics of LSTM cells for reliability modeling of power devices, and we continue to present our proposed reliability modeling network constructed out of the basic LSTM cells.

2.4.1.1 Long Short-Term Memory Cells

For reliability modeling of power electronic converters, we propose using LSTM cells. In a nutshell, different deep neural networks recognize the patterns in two forms of spatial and temporal pattern depending on their structure. As an instance, Convolutional Neural Networks (CNN) are engineered in a form that they can distinguish spatial pattern, e.g. a dog or a face, existed in a picture. In the other hand, sequence data such as natural language data and time series, e.g. stock index have a temporal pattern that should be processed during a sequence of time. For our case, since we try to model $\Delta R_{ds(on)}$ data and it is intrinsically a time series, we leverage LSTM cells as the prominent version of RNN. The benefits of LSTM cells are in using the guided gates for selectivity remembering both short and long-term behaviors across many time series. The LSTM uses a subset of the cyclical node inside its cell known as "memory" in order to calculate the output based on the current input and its past status [34]. The LSTM selective memory sensitivity at large scale offers a systematic approach for reliability modeling of power electronic devices.

The LSTM cell contains three vectorized sigmoid (σ) functions, which each individual function operates as a gate and controls the flow of information passing through the cell — the input, output, and forget gates. Fig. 2.3 presents the internal structure of an LSTM cell. Each gate maps its input to $S = \{s_i | s_i \in [0, 1]\}$, where zero is a closed gate and one means the gate is open. Moreover, the cell state (memory) is preserved by C as a candidate. The information of new candidates, which should be stored in the cell state, represented as $\tilde{c}$.

$$i_t = \sigma(W_i v_t + b_i), \tag{2.7}$$

$$f_t = \sigma(W_f v_t + b_f), \tag{2.8}$$

$$o_t = \sigma(W_o v_t + b_o), \tag{2.9}$$

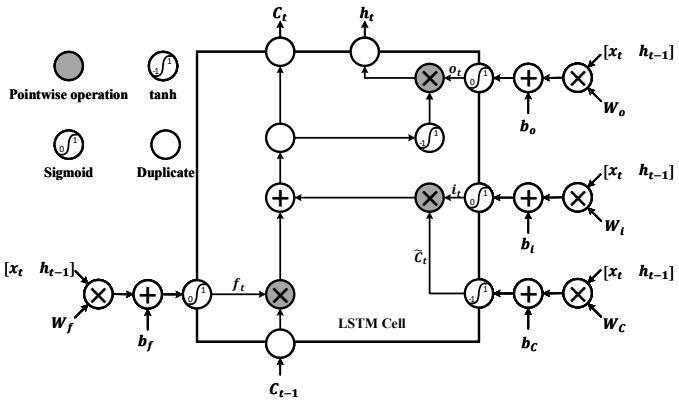


Figure 2.3: A single LSTM cell: Inside of an LSTM cell consisting of three gates and the state of the cell is preserved by variable c_t.

$$\tilde{c}_t = tanh(W_c v_t + b_c), \tag{2.10}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \tag{2.11}$$

$$h_t = o_t \odot tanh(c_t). \tag{2.12}$$

With respect to LSTM cell visualization in Fig. 2.3, Equations 2.7-2.12 formulates the correlation between input and output per LSTM cell. In the equations, $v_t = [x_t \quad h_{t-1}]$, $\odot$ is Hadamard or element-wise matrix product, and $\theta = [W_i \quad W_o \quad W_f \quad W_c \quad b_i \quad b_o \quad b_f \quad b_c \quad c_0]$ is the network model that should be trained. The input gate (2.7) decides what portion of current input with which degree should be stored in cell memory, while forget gate (2.8) chooses which portion of memory should be erased. In the other hand, new information, $\tilde{c}$, will be mined by (2.10), and the cell memory will be updated by (2.11). Moreover, the output gate (2.9) decides which part of cell memory should affect the LSTM output at time t, and finally the LSTM output value is calculated in (2.12).

2.4.1.2 LSTM-Based Device Training Network

LSTM-based neural network needs to be designed to properly reflect $\Delta R_{ds(on)}$ propagation with enough depth to capture complex power convertors behaviors, and thus

learning deep behavioral patterns across many devices. A deep LSTM network should have sufficient depth to build up the progressive pattern recognition of sequential data in both coarse and fine grain directions.

As it mentioned in Section 2.2.2, we consider the trajectory resistance of drain-source of power MOSFET during ON time (i.e. $\Delta R_{ds(on)}$) as the precursor of device failure degradation. As $\Delta R_{ds(on)}$ is intrinsically a time series, we design the model by using deep LSTM network, where the training is developed by aggregating data from different devices having the same technology. A batch of samples is created for each training iteration. Therefore, for predicting the next n samples of $\Delta R_{ds(on)}$ based on the provided last input sequence (τ), the batch should consist of $\Delta R_{ds(on)}$ with the size of $(\tau+n)$. Fig. 2.4 presents the batch tensor configuration per each iteration. The dimension of vector R^k_{mt} is characterized based on the *input size* shown by k, where m is the available devices for training, and t is the sequence. In order to prevent any sort of biases in training the LSTM, we selected randomly a vector sequence with the size of $(\tau + n)$ from the $\Delta R_{ds(on)}$ samples per each device. Increasing the number of devices, (m), per each batch helps the network to generalize the modeling of complex degradation properly, which results in higher accuracy of the predicted trajectory.

For designing of deep LSTM network, we need to also consider *number of hidden layers*. The number of hidden layer is the dimension of vectors generated in equations

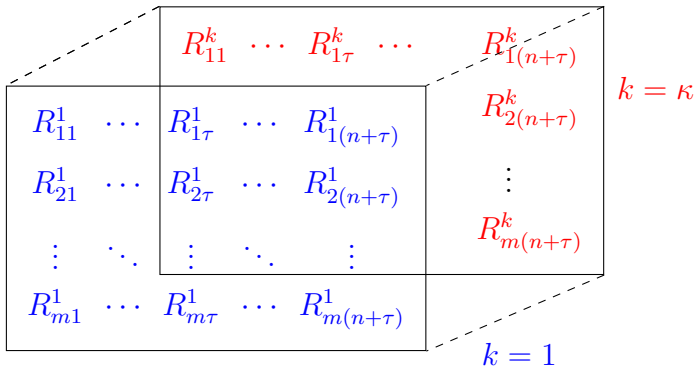


Figure 2.4: Batch tensor configuration: Three dimensional batch tensor with a characterized vector R^k_{mt}.

(2.7)-(2.12). The vector size can be changed by altering the weights and biases tensor shape defined in equations (2.7)-(2.10). Increasing the hidden layer is interpreted as increasing the "memory" size of the LSTM and its capacity to learn existing complex pattern in a signal.

At the same time, building a very deep LSTM network is not a viable solution due to lack of large data-set to trained all LSTM cells once at the same time. The BPTT often failed to train a large flat LSTMs network without any patriarchy. As the result, the alternative solution is stacking multiple LSTM networks (which called Stacked LSTM) to create more complex and deeper network with offering hierarchical modular layers over a very deep flat network. In our proposed network, we suggest developing a stacked LSTM to generalize the behavior complexity of power electronics convertor without the need for a very large data-set. As the result, *number of stacked layers* is the other parameter to increase the complexity of the LSTM network. Here, by stacking the cell up together in a way the h_t of one cell is used as an input to its top adjacent cells. Fig. 2.5 shows an architecture of the deep LSTM network with the stacked layer size of $\boldsymbol{\ell}$.

Since the output vector $h_t \in [-1, 1]$, we need to de-normalize the deep LSTM network output to actual system measurement. Therefore, a dense layer is added to the output of stacked LSTM to map h_t to the predicted $\Delta R_{ds(on)}$ at the time t as shown in Fig. 2.6. Based on modified deep LSTM structure, the network models are described as: $\theta_\lambda = [W_{\lambda_i} \quad W_{\lambda_o} \quad W_{\lambda_f} \quad W_{\lambda_c} \quad b_{\lambda_i} \quad b_{\lambda_o} \quad b_{\lambda_f} \quad b_{\lambda_c} \quad c_{\lambda_0}], 1 \leq \lambda \leq \boldsymbol{\ell}$, and $\theta_d = [W_d \quad b_d]$, where $\boldsymbol{\ell}$ is the stacked layer size. Each LSTM cell will be trained at the cloud, and network model will be transferred to the edge next to the transistor device for real-time prediction. Acquiring proper values for the LSTM network parameters can be done by exploring the design space in regard to system constraints (e.g., system accuracy, processing time, and power consumption) [35].

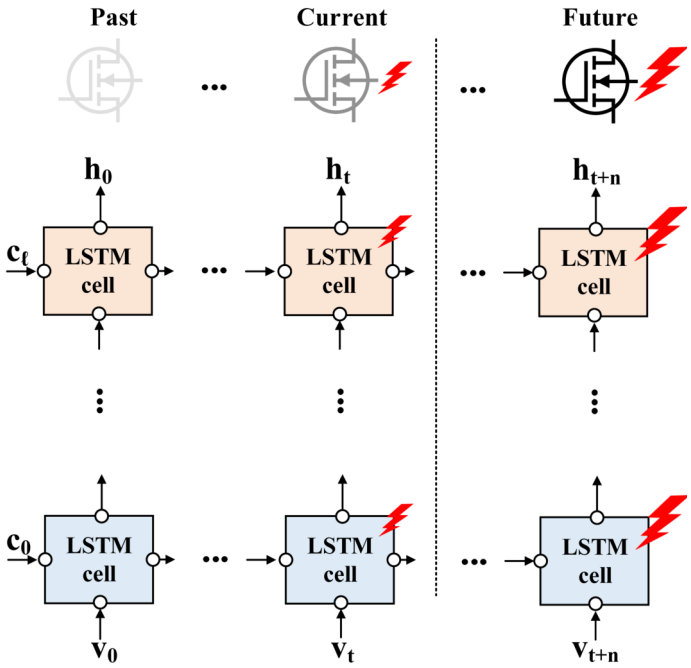


Figure 2.5: The stacked LSTM: An unrolled LSTM cell predicts the next n samples of $\Delta R_{ds(on)}$ based on last sensed data.

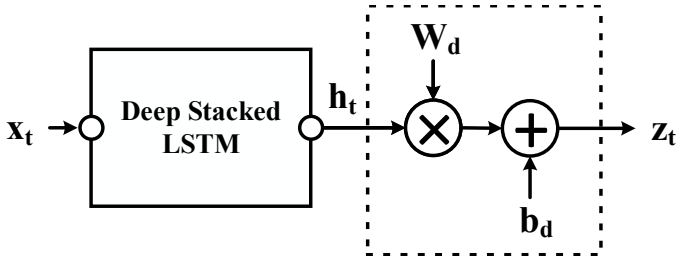


Figure 2.6: The proposed deep LSTM network model: A dense layer is added to the deep stacked LSTM to map h_t to on-line measured $\Delta R_{ds(on)}$ at time t.

2.4.2 Proposed IoT Framework

In this subsection, we explained the system integration to realize the proposed LSTM reliability modeling constructs through an IoT-based cloud-edge platform.

2.4.2.1 Data training and batch aggregation on the cloud

One key aspect of Deep RACE is an integrative cloud-edge system for scalable real-time reliability monitoring, assessment, and prediction. Fig. 2.7 shows the system architecture of Deep RACE based on the cloud-edge solution - training on the cloud

Algorithm 1 Training the deep LSTM network

Input: $X_{training}, Y_{training}, X_{test}, Y_{test}, \tau, m, n, \ell, k, \epsilon, e_{th}, it_{max}$

Output: $\theta_\lambda; 1 \leq \lambda \leq \ell, \theta_d$ ▷ Network models

1: $computation\_graph \leftarrow$ LSTM($hidden\_layer, \ell$)
2: $error \leftarrow \infty$ ▷ Initialize the test error
3: $min\_error \leftarrow \infty$ ▷ Saves the minimum error test
4: $\jmath \leftarrow 0$
5: init_rand(θ_λ **for** λ in $[1...\ell]$) ▷ Initialize LSTM network models from truncated normal distributions
6: init_rand(θ_d)
7: **while** $(\jmath \leq it_{max})$ **or** $(error \geq e_{th})$ **do**
8: $X_{batch}, Y_{batch} \leftarrow$ generate_data($X_{training}$, $Y_{training}$, m, n, τ, k)
9: $\Delta R_{ds(on)} \leftarrow$ inference($computation\_graph, X_{batch}$, [$\theta_\lambda$ **for** λ in $[1...\ell]$], θ_d) ▷ Predicted $\Delta R_{ds(on)}$ for training
10: $error_{training} \leftarrow \mathcal{L}(\Delta R_{ds(on)}, Y_{batch})$
11: [θ_λ **for** λ in $[1...\ell]$], $\theta_d \leftarrow$ optimizer($computation\_graph, error_{training}$, [$\theta_\lambda$ **for** λ in $[1...\ell]$], θ_d) ▷ Updating the net. models
12: $x_{test}, y_{test} \leftarrow$ generate_data(X_{test}, Y_{test}, m, n, τ, k)
13: $\Delta R_{ds(on)} \leftarrow$ inference($computation\_graph, x_{test}$, [$\theta_\lambda$ **for** λ in $[1...\ell]$], θ_d) ▷ Predicted $\Delta R_{ds(on)}$ for test
14: $error \leftarrow \mathcal{L}(\Delta R_{ds(on)}, Y_{test})$
15: **if** $(min\_error > error)$ **then**
16: $min\_error \leftarrow error$
17: save(θ_λ **for** λ in $[1...\ell]$, θ_d) ▷ Save the network model with lowest error
18: **end if**
19: $\jmath \leftarrow \jmath + 1$
20: **end while**

and real-time sensing and inference for reliability monitoring and health assessment on the edge nodes. The cloud is the centralized computing node for data aggregation and training of proposed LSTM algorithm across many power electronic transistors with similar underlying physics. The cloud stores the initial sampled dataset (e.g., voltage, current, and device temperature) collected from devices under stress test for initial training. At the same time, it continuously receives new information and sample data from running devices for improving the accuracy of reliability modeling and prediction. The edge nodes are IoT devices which use local computing power to perform the real-time reliability monitoring, and prediction (deep learning inference). The edge nodes rely on the pre-trained models that have created during the training

phase on the cloud.

In this context, the edge nodes and cloud continuously interact and exchange information. The Edge nodes, while performing real-time reliability assessment, collect and update the cloud with the properties of the transistors for future training. The cloud also updates edge nodes with new reliability models (LSTM models) to increase the confidence interval of the prediction, and estimate the remaining useful life of the devices. To increase the confidence interval of the power MOSFET devices and the system operation, a predefined threshold ΔR_t can be defined based on system requirement. Once the error of predicted device resistance $\Delta R_{ds(on)}$ is greater than the ΔR_t, the network models will be automatically updated through training the network on the cloud server.

Algorithm 1 describes the training procedure on the cloud side. The *computaion_graph* is the deep LSTM network structure. At first, the network models are initialized from a truncated normal distribution. X_{batch}, Y_{batch} are generated based on the input size, input sequence, output sequence, and the number of devices. Next, *computation_graph* predicts the $\Delta R_{ds(on)}$ according to its current network models and generated training batch. The models will be updated through the back-propagating error from the output of *computaion_graph* to its inputs.

During training phase of our proposed LSTM algorithm, and in general deep learning models, one major source of error would be over-fitting. In order to prevent the over-fitting problem, we have created a test batch (x_{test}, y_{test}) to predict $\Delta R_{ds(on)}$ of the test device based on the updated network model. Next, the test batch *error* will be compared against the previous error values and if it has the minimum value, the network model will be saved.

2.4.2.2 Real-time edge analysis

The edge converter has its own local controller equipped with an embedded SoC for the purpose of predicting the power transistor degradation. The μ-controller unit

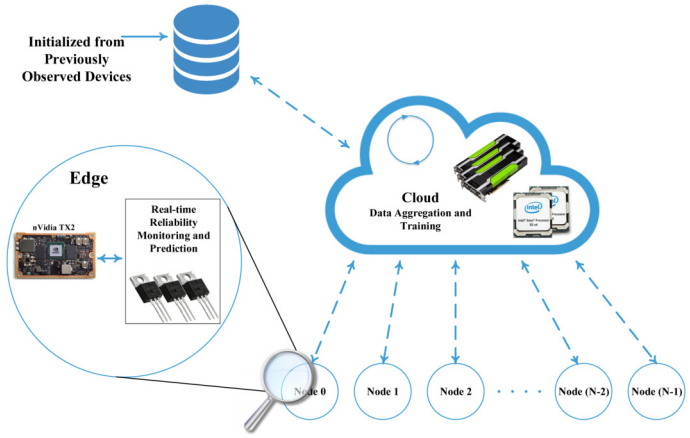


Figure 2.7: The Deep RACE Framework: The proposed solution accumulates the knowledge of power transistor degradation model on the cloud-side by training the LSTM network, while real-time prediction and inference is accomplished on the edge side.

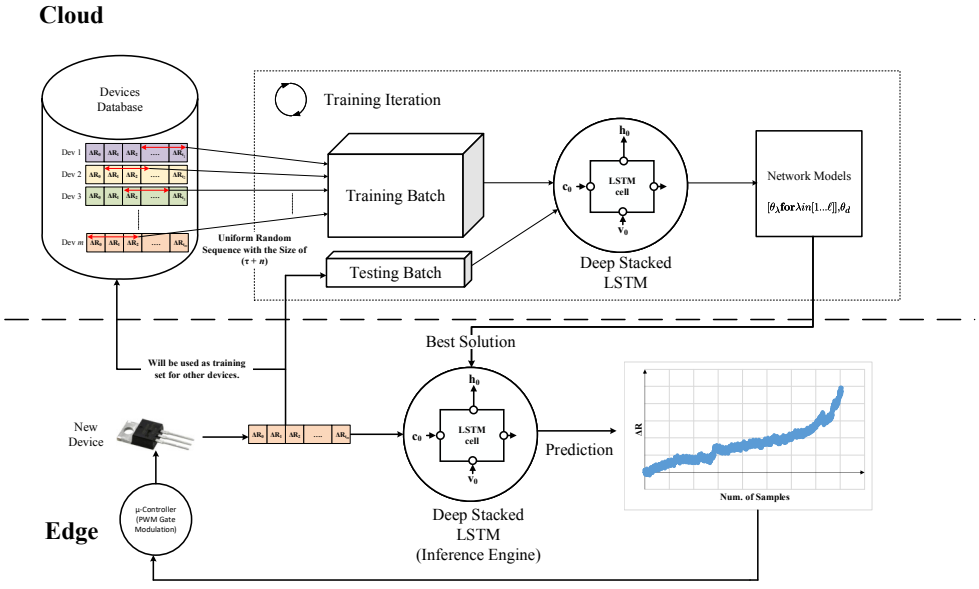


Figure 2.8: The scalability of Deep RACE: As new edge is added to the framework, its extracted $\Delta R_{ds(on)}$ is transfered to the cloud to be used both as training data set (for other devices) and test data set to prevent over fitting problem.

is responsible of modulating the gate signals for the power converter control, and continuous monitoring of the voltage, current, and temperature of power converters. The sampled data also will be transferred to the cloud , which performs the reliability analysis (training phase) for each edge node. The SoC of the edge runs inference section of the deep LSTM, equations (2.7)-(2.12), and estimates the trajectory device resistance, $\Delta R_{ds(on)}$, based on the received trained network models from the cloud. Based on the predicted $\Delta R_{ds(on)}$, controller can leverage load-sharing [36] method through the system level control in modular converters or cascaded architectures in many applications such as distribution generation systems, data centers, and the electric vehicles in order to decrease the degradation pace until the appropriate action is taken.

Fig. 2.8 visualizes the scalability of Deep RACE when a new edge is added to the framework. At first, the edge node sends the voltage, current, and temperature samples to the cloud-side. The samples will be used for two purposes: (1) as training sets for the other nodes, and (2) as test sets in order to prevent over-fitting phenomenon during training process of the LSTM network. Then, network models will be sent back to the edge for the purpose of $\Delta R_{ds(on)}$ prediction. As more new devices added to the framework, the prediction error will be decreased as we demonstrated in the next section.

2.5 Experimental Results

The performance of proposed real-time reliability analysis was examined for training the data and applying the Deep RACE. This section describes the testing scenarios, the hardware setup, and the experimental results.

2.5.1 Experimental training of power transistors

On the cloud server, we used Intel Xeon CPU E5-2640 to train the deep LSTM network, where we initially modeled Si-power MOSFETs. The experimental data

sets for both training and testing of the power MOSFET (IRF520NPbf) are provided from NASA dataset [28]. The Deep RACE predicts a new transistor degradation behavior based on the trained system without any prior knowledge in advance. In our experiment, the Deep RACE is trained to estimate the next 104 samples. For the application with higher window resolutions (i.e. higher output sequence), the network input sequence should also be increased to minimize the prediction error. Table 2.1 summarizes the deep LSTM network parameters. Based on the network configuration, we have also measured the training time on the cloud server. Table 2.2 summarizes the training time on the cloud side.

We used Google TensorFlow framework to implement our stacked LSTM network model. Each LSTM cell is instantiated by calling $tensorflow.contrib.rnn.LSTMCell$ function where the number of "hidden layer" is passed as an argument to this function. In the next step, an array consists of $LSTMCell$ with the size of "stacked layer" is generated. Then, the array will be passed to the $tensorflow.contrib.rnn.MultiRNNCell$ function to create the stacked LSTM network. The network unrolling is accomplished through $tensorflow.nn.dynamic\_rnn$ function. We defined Mean Square Error (MSE) (2.13) as an objective loss function, and used $tf.train.AdamOptimizer$ method to minimize the function:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - z_i(\theta))^2, \tag{2.13}$$

Table 2.1: The parameters for LSTM network training

Item	Parameter	Description	Value
1	k	Input size	1
2	τ	Input sequence	21
3	e_{th}	Error threshold	5×10^{-5}
4	n	Output sequence	104
5	it_{max}	Maximum iterations	1000
6	ϵ	Number of hidden layer	64
7	ℓ	Number of stacked layer	4
8	m	Number of device for training	4

where $z_i(\theta)$ is the predicted output trajectory from the Deep RACE, and y_i is the actual measurement of the device resistance.

2.5.2 Edge Node Hardware Setup

We also developed a low-power edge computing system for real-time monitoring and reliability assessment. The edge computing node is based on nVidia TX2 board as the state-of-art embedded SoC with GPU compute units for edge device. As we explained, the inference part of Deep RACE is implemented on the edge, since the cloud is responsible for aggregated training of the proposed structure.

Fig. 2.9 shows the prototype of Deep RACE hardware realization at the edge. In this system, μ-controller controls the power converter, and the voltage, and current of the power semiconductor are captured and then transfered to the TX2 board for edge analysis. For the safety purpose, the automated supervisory control is designed for data collection from the switching converter and also protects the system operation if the power conversion deviates more than 5%.

2.5.3 Reliability Modeling and Prediction

We created five different scenarios per each device in order to evaluate the scalability and robustness of Deep RACE. As an example, the trajectory resistance for Dev#5 is predicted based on learning degradation model from Dev#1 to Dev#4. Then, recursively we reinitialized all network models from truncated normal distribution again, and we substituted the other devices to predict a new unknown transistor resistance variations from scratch. Therefore, the Deep RACE is challenged to predict

Table 2.2: Data size and average elapsed time for training

Item	Description	Value
1	Training size/Iteration	$m \times k(\tau + n) = 500$
2	CPU Cores	32
3	GPUs	nVidia Tesla P100, *and* nVidia TITAN V
4	Elapsed time	596 Seconds

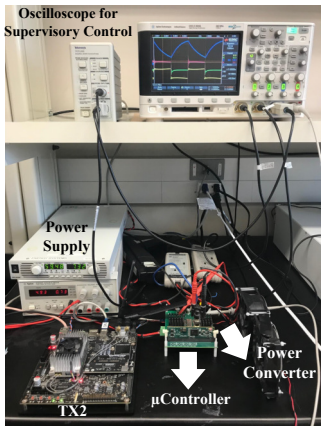


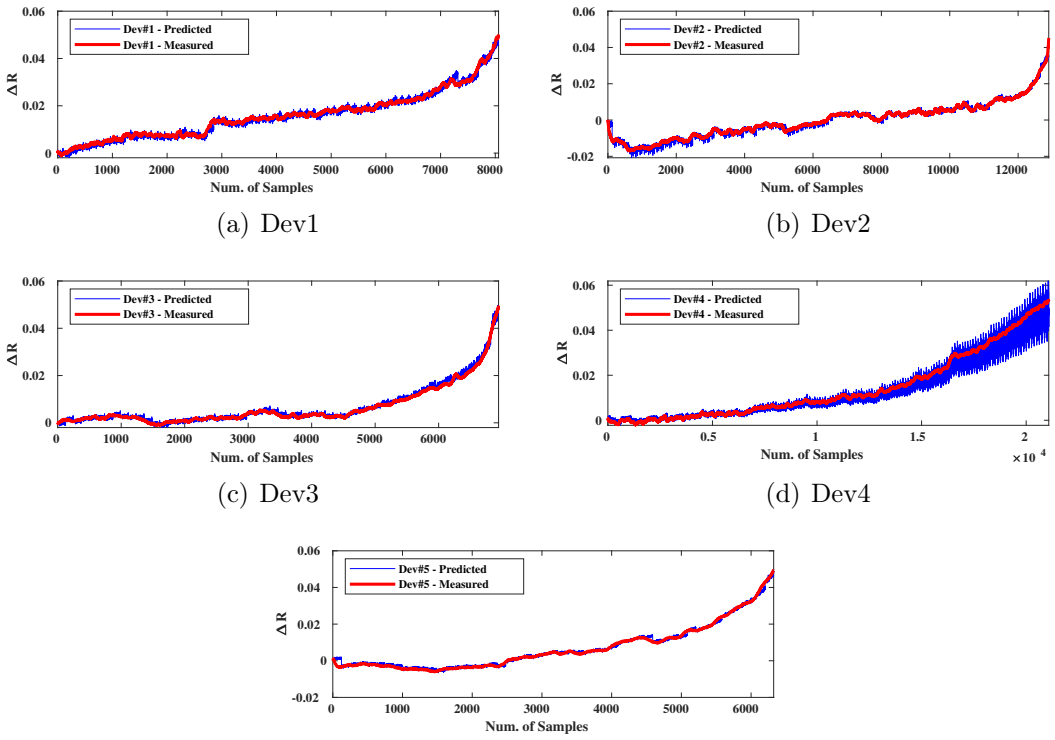
Figure 2.9: Experimental Verification: The hardware realization of Deep RACE including the high-frequency power converter controller, and the SoC-TX2 for edge computation. The supervisory control is designed for the safety protection.

a completely new and unknown device based on aggregating knowledge from other power devices at each scenario. We verified the system characteristics from acquired experimental results in two forms of MSE and error distribution. Table 2.3 shows prediction MSE of the selected devices. While the training process is performed in the cloud, we evaluated the prediction of the device resistance variation at the edge.

Fig. 2.10 illustrates the Deep RACE prediction performance for defined five scenarios and clarifies the scalability of the proposed algorithm. Although the apparatus behavior of each power device degradation looks similar, the microscopic observation of the transistors is different within the same time horizon. For an instance, the trained network for Dev#4 is expecting an exponential increment in the region of $\Delta R_{ds(on)} > 0.02\Omega$ based on aggregated training from Dev#1 to Dev#3, and Dev#5. This error can be further minimized through a new learning phase on the cloud.

Table 2.3: Prediction error for the power MOSFET transistors.

Devices	#1	#2	#3	#4	#5
log(MSE)	-13.61	-13.05	-13.95	-13.36	-12.94



(a) Dev1

(b) Dev2

(c) Dev3

(d) Dev4

(e) Dev5

Figure 2.10: Experimental results: The resistance variation of given five power modules, which were predicted by Deep RACE method.

We also extracted the error distribution for five predicted devices by using:

$$Error_{diff} = (y_i - z_i(\theta)), \tag{2.14}$$

where $z_i(\theta)$ is the predicted output trajectory from the Deep RACE, and y_i is the actual measurement of the device resistance ($\Delta R_{ds(on)}$ in our model). Fig. 2.11 depicts that the average maximum error caused by Deep RACE method is less than 0.9%.

We extended our experiment to analyze the effect of training aggregation and scalability of multiple device data on accuracy of $\Delta R_{ds(on)}$ prediction. In this case, we increased the number of devices per each batch during the training phase. Fig. 2.12 shows that MSE decreases with an exponential rate by increasing the number of devices in training batch. For each training set, we ran 1000 Monte-Carlo test for $\Delta R_{ds(on)}$ predictiopn of three different devices, and then the average of whole test sets is picked. These results indicate that our proposed approach can improve the prediction accuracy exponentially by increasing the edge node and power transistors through the life time of the system by accumulating knowledge about different device degradation during its usage, which demonstrates the scalability of our approach.

The region of $\Delta R_{ds(on)} < 0.02\Omega$ has linear behavior, therefore, the classical approaches (such as Kalman Filter, and Particle Filter) can predict the health state with higher accuracy; however, the prediction error increases after that for these methods. Since it is very crucial to detect the MOSFET resistance variation when $\Delta R_{ds(on)} \approx 0.05\Omega$, we calculated the average of error at the detection point($=\Delta R_{5\%}$) for these methods by using (2.15):

$$Error_{\Delta R_{5\%}} = \frac{100}{m} \sum_{i=1}^{m} \frac{|\Delta R_{ds(on)_{mt_{5\%}}} - (0.05)_{mt_{5\%}}|}{(0.05)_{mt_{5\%}}}, \tag{2.15}$$

where m is the number of devices, $\Delta R_{ds(on)}$ is predicted value, and $t_{5\%}$ is the time

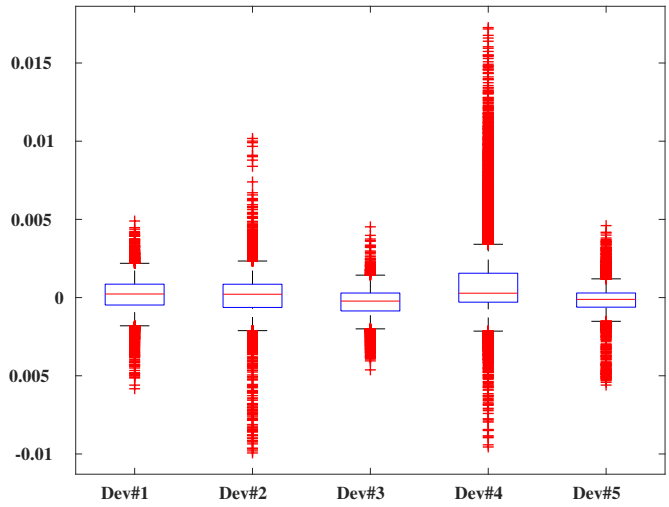


Figure 2.11: Error distribution: The box-whisker plots of prediction error for five power modules.

when the first sensed value is 0.05Ω. The $Error_{\Delta R_{5\%}}$ results are summarized in Table 2.4. Our experiments indicate that the Deep RACE reduces the miss-prediction error at 0.05Ω by about 1.98x, 1.77x compared to Kalman Filter and Particle Filter, respectively.

Table 2.4: The comparison of the absolute average error of Deep RACE with conventional methods

Method	Kalman Filter [30]	Particle Filter [28]	Deep RACE
Miss-prediction Error	17.75%	15.85%	8.93%

2.5.4 Power consumption and processing time analysis

We also evaluated the power consumption and delay of the inference part of the network on embedded TX2 board. Table 2.5 summarizes the specification of embedded SoC. Since the TX2 has an embedded GPU, We considers two different scenarios to analyze the performance of Deep RACE. At first scenario, we set the tensorflow configuration to `device_count = {'GPU': 0}`, where no computation carried out at the embedded GPU, and in the second approach we made it ON. For the matrix size

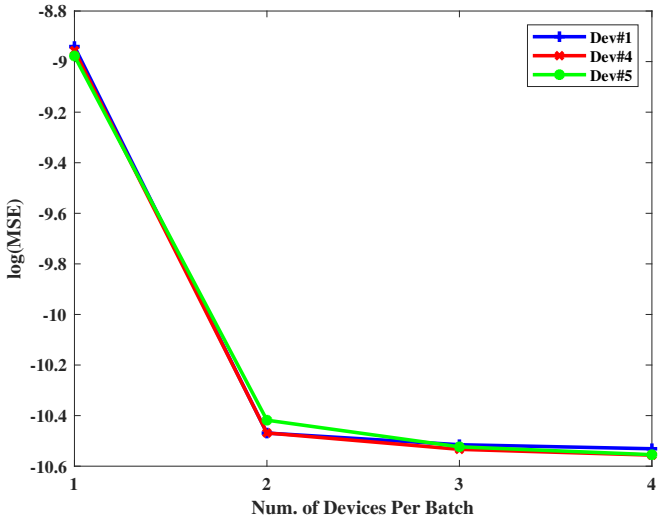


Figure 2.12: Aggregated training: The $\Delta R_{ds(on)}$ prediction error is decreased exponentially by increasing the training device per each batch. This aggrigated training will help the Deep RACE to generalize the different transistor degradation behavior.

of 125 (input sequence + output sequence), it was observed that the CPU processed 3.2x faster than GPU for one device prediction. This performance degradation is because of data copying between CPU and GPU memory region – Note the DDR power consumption is higher for `'GPU': 1` scenario. In the other word, the amount of data is not enough for GPU to overlap the delay between data computation and movement. Increasing the number of devices that should be predicted per each edge node or increasing the prediction window resolution (*output sequence*) improves the performance for GPU since it carries out more computation than CPU per each data set. Table 2.6 summarizes the delay and power dissipation for two different cases.

Table 2.5: Nvidia TX2 Embedded Module Specification

CPU	GPU	DDR
Quad Cortex-A57 @ 2GHz + Dual Denver2 @ 2GHz	256-core Pascal @ 1300MHz	8GB LPDDR4 @ 1866MHz

Table 2.6: TX2 embedded board power consumption.

	GPU: OFF (0)			GPU: ON (1)		
Module	CPU	DDR	CPU+DDR	GPU	DDR	GPU+DDR
Power (W)	1.07	0.80	1.87	0.166	0.90	1.06
Delay (ms)		26			85	

2.6 Conclusion and Future Work

This chapter proposed a new solution as a collection of deep learning, edge, and cloud computing technologies to enable real-time high accuracy reliability modeling of high-frequency MOSFETs power converter devices. The proposed deep learning algorithm is based on LSTM algorithmic constructs for accumulating the degradation knowledge of different power MOSFET devices on the cloud server, and real-time inference at the edge. For the experimented results, we developed an entire integrated system of Deep RACE, including an embedded system system-on-chip implementation on nVidia SoC-TX2. The results demonstrated the real-time convergence of the system with about 8.9% miss prediction, with $26ms$ processing time.

In a broader perspective, the proposed research will have a fundamental contribution in the engineering of semiconductor devices and information processing by bringing recent advances in deep learning and edge computing for real-time predictive maintenance of emerging semiconductor devices. In this context, Deep RACE sets to move beyond mainstream device modeling and traditional reliability analysis (i.e. Weibull distributions, mean-time-to-failure, etc.) and looking to more applicable and accurate analytical tools through introducing advanced sensing solutions and combining it with cutting-edge deep learning techniques.

Bibliography

[1] L. Tolbert, T. King, B. Ozpineci, J. Campbell, G. Muralidharan, D. Rizy, A. Sabau, H. Zhang, W. Zhang, Y. Xu *et al.*, "Power electronics for distributed energy systems and transmission and distribution applications," *ORNL/TM-2005/230, UT-Battelle, LLC, Oak Ridge National Laboratory*, vol. 8, 2005.

[2] Y. Yang, H. Wang, A. Sangwongwanich, and F. Blaabjerg, "Design for reliability of power electronic systems," in *Power Electronics Handbook (Fourth Edition)*. Elsevier, 2018, pp. 1423–1440.

[3] K. Ma, H. Wang, and F. Blaabjerg, "New Approaches to Reliability Assessment: Using physics-of-failure for prediction and design in power electronics systems," *IEEE Power Electronics Magazine*, vol. 3, no. 4, pp. 28–41, 2016.

[4] A. Saxena, J. Celaya, E. Balaban, K. Goebel, B. Saha, S. Saha, and M. Schwabacher, "Metrics for evaluating performance of prognostic techniques," in *Prognostics and health management, 2008. PHM 2008. international conference on*. IEEE, 2008, pp. 1–17.

[5] H. S.-h. Chung, H. Wang, F. Blaabjerg, and M. Pecht, *Reliability of power electronic converter systems*. Institution of Engineering and Technology, 2015.

[6] G. Bedi, G. K. Venayagamoorthy, R. Singh, R. R. Brooks, and K.-C. Wang, "Review of Internet of Things (IoT) in Electric Power and Energy Systems," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 847–870, 2018.

[7] B. Wang, Y. Wang, H. Nazaripouya, C. Qiu, C.-C. Chu, and R. Gadh, "Predictive scheduling framework for electric vehicles with uncertainties of user behaviors," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 52–63, 2017.

[8] S. Yang, D. Xiang, A. Bryant, P. Mawby, L. Ran, and P. Tavner, "Condition

monitoring for device reliability in power electronic converters: A review," *IEEE Transactions on Power Electronics*, vol. 25, no. 11, pp. 2734–2752, 2010.

[9] N. Degrenne, J. Ewanchuk, E. David, R. Boldyrjew, and S. Mollov, "A Review of prognostics and health management for power semiconductor modules," 2014.

[10] J. R. Celaya, A. Saxena, C. S. Kulkarni, S. Saha, and K. Goebel, "Prognostics approach for power MOSFET under thermal-stress aging," in *Reliability and Maintainability Symposium (RAMS), 2012 Proceedings-Annual*. IEEE, 2012, pp. 1–6.

[11] M. Musallam, C. Yin, C. Bailey, and C. M. Johnson, "Application of coupled electro-thermal and physics-of-failure-based analysis to the design of accelerated life tests for power modules," *Microelectronics Reliability*, vol. 54, no. 1, pp. 172–181, 2014.

[12] A. Alghassi, S. Perinpanayagam, M. Samie, and T. Sreenuch, "Computationally efficient, real-time, and embeddable prognostic techniques for power electronics," *IEEE Transactions on Power Electronics*, vol. 30, no. 5, pp. 2623–2634, 2015.

[13] M. Heydarzadeh, S. Dusmez, M. Nourani, and B. Akin, "Bayesian remaining useful lifetime prediction of thermally aged power MOSFETs," in *Applied Power Electronics Conference and Exposition (APEC), 2017 IEEE*. IEEE, 2017, pp. 2718–2722.

[14] L. R. GopiReddy, L. M. Tolbert, B. Ozpineci, and J. O. Pinto, "Rainflow algorithm-based lifetime estimation of power semiconductors in utility applications," *IEEE Transactions on Industry Applications*, vol. 51, no. 4, pp. 3368–3375, 2015.

[15] S. Dusmez and B. Akin, "An accelerated thermal aging platform to monitor

fault precursor on-state resistance," in *Electric Machines & Drives Conference (IEMDC), 2015 IEEE International*. IEEE, 2015, pp. 1352–1358.

[16] J. R. Celaya, A. Saxena, and K. Goebel, "Uncertainty representation and interpretation in model-based prognostics algorithms based on Kalman Filter estimation," National Aeronautics and Space Administration Moffett Field CA Ames Research Center, Tech. Rep., 2012.

[17] S. Dusmez and B. Akin, "An active life extension strategy for thermally aged power switches based on pulse-width adjustment method in interleaved converters," *IEEE Trans. Power Electron.*, vol. 31, no. 7, pp. 5149–5160, 2016.

[18] M. A. Nabian and H. Meidani, "Deep learning for accelerated seismic reliability analysis of transportation networks," *Computer-Aided Civil and Infrastructure Engineering*, vol. 33, no. 6, pp. 443–458. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/mice.12359

[19] P. Jiang, M. Maghrebi, A. Crosky, and S. Saydam, "Unsupervised deep learning for data-driven reliability and risk analysis of engineered systems," in *Handbook of Neural Computation*. Elsevier, 2017, pp. 417–431.

[20] J. Wang and C. Zhang, "Software reliability prediction using a deep learning model based on the rnn encoder–decoder," *Reliability Engineering & System Safety*, vol. 170, pp. 73–82, 2018.

[21] E. Wolfgang, "Examples for failures in power electronics systems," *ECPE tutorial on reliability of power electronic systems, Nuremberg, Germany*, pp. 19–20, 2007.

[22] S. Yang, A. Bryant, P. Mawby, D. Xiang, L. Ran, and P. Tavner, "An industry-based survey of reliability in power electronic converters," *IEEE transactions on Industry Applications*, vol. 47, no. 3, pp. 1441–1451, 2011.

[23] I. Kovacevic, U. Drofenik, and J. W. Kolar, "New physical model for lifetime estimation of power modules," in *Power Electronics Conference (IPEC), 2010 International*. IEEE, 2010, pp. 2106–2114.

[24] J. M. Anderson and R. W. Cox, "On-line condition monitoring for MOSFET and IGBT switches in digitally controlled drives," in *Energy Conversion Congress and Exposition (ECCE), 2011 IEEE*. IEEE, 2011, pp. 3920–3927.

[25] Automotive Electronic Council, "Stress Test Qualification for Automotive Grade Discrete Semiconductors," 2013. [Online]. Available: hhttp://www.aecouncil.com/Documents/AEC_Q101_Rev_D1_Base_Document.pdf

[26] N. M. Vichare and M. G. Pecht, "Prognostics and health management of electronics," *IEEE transactions on components and packaging technologies*, vol. 29, no. 1, pp. 222–229, 2006.

[27] S. Song, S. Munk-Nielsen, C. Uhrenfeldt, and I. Trintis, "Failure mechanism analysis of a discrete 650V enhancement mode GaN-on-Si power device with reverse conduction accelerated power cycling test," in *Applied Power Electronics Conference and Exposition (APEC), 2017 IEEE*. IEEE, 2017, pp. 756–760.

[28] J. Celaya, A. Saxena, S. Saha, and K. F. Goebel, "Prognostics of power MOSFETs under thermal stress accelerated aging using data-driven and model-based methodologies," 2011.

[29] H. Wang, M. Liserre, and F. Blaabjerg, "Toward reliable power electronics: Challenges, design tools, and opportunities," *IEEE Industrial Electronics Magazine*, vol. 7, no. 2, pp. 17–26, 2013.

[30] S. Dusmez, H. Duran, and B. Akin, "Remaining useful lifetime estimation for thermally stressed power MOSFETs based on on-state resistance variation," *IEEE Transactions on Industry Applications*, vol. 52, no. 3, pp. 2554–2563, 2016.

[31] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[32] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, p. 533, 1986.

[33] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[34] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[35] M. Baharani, H. Noori, M. Aliasgari, and Z. Navabi, "High-level design space exploration of locally linear Neuro-fuzzy models for embedded systems," *Fuzzy Sets Syst.*, vol. 253, pp. 44–63, Oct. 2014. [Online]. Available: http://dx.doi.org/10.1016/j.fss.2013.12.006

[36] Z. Hu, Y. Qiu, Y.-F. Liu, and P. Sen, "An interleaving and load sharing method for multiphase LLC converters," in *Applied Power Electronics Conference and Exposition (APEC), 2013 Twenty-Eighth Annual IEEE*. IEEE, 2013, pp. 1421–1428.

CHAPTER 3: DEEPDIVE

3.1 Introduction

The astonishing growth in deep learning algorithms, particularly, Convolutional Neural Networks (CNNs), has enabled many exciting applications in visual analytics. We have observed a recent shift towards Domain-Specific Architectures (DSA), e.g., Systolic Arrays, CGRAs, Tensor Cores, to cope with the significant computation demand raised by deep learning paradigms [1, 2, 3, 4, 5, 6, 7]. These emerging DSAs often transform convolutional operations into dense linear algebraic operations across the channels and kernels. This maximizes parallelism and compute resource utilization, as well as minimizes data movements, by increasing data re-usability. They are typically designed to be a generic, one-size-fits-all architecture that allows hardware reuse between different layer operations. As a result, they execute the target CNN layer-by-layer sequentially. A notable example is the recently introduced Versatile Tensor Accelerator (VTA) [7], which is an open, generic, and customizable deep learning accelerator with a complete TVM-based compiler stack, targeted for edge FPGAs. Another such accelerator design presented by [8] introduces a configurable architecture, pipelined, and timing controlled design with fixed hardware solution specially designed for MobileNet.

Deep Separable CNNs (DSCNNs) [9, 10, 11, 12, 13, 14] have emerged as an innovative algorithmic solutions to achieve higher accuracy with relatively lower parameters and operations. State-of-the-art separable CNNs, e.g., MobileNet family [12, 14] and EfficientNet [13], offer modular networks with structural sparsity over various convolutional operators — group, depthwise, and pointwise convolution. DSCNNs often result in relatively higher computational sparsity, more data-dependent layer-to-layer

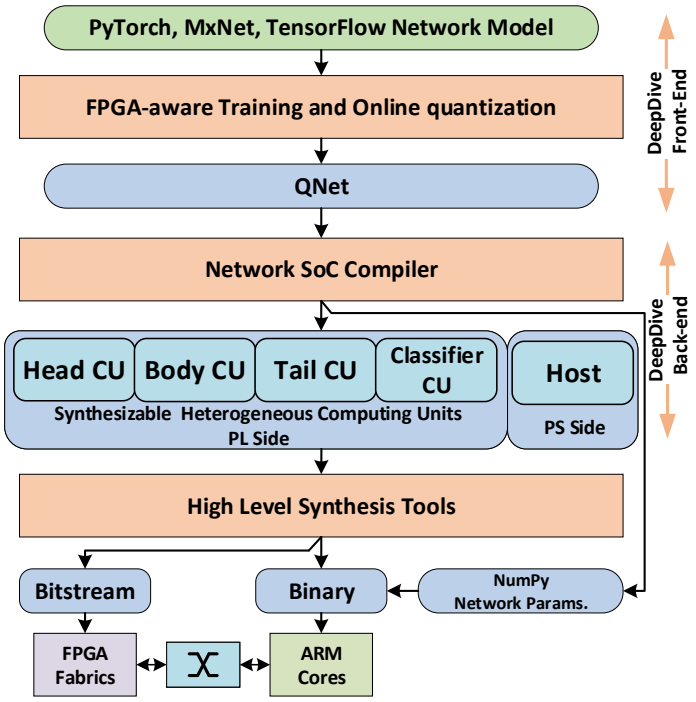


Figure 3.1: DeepDive integrative design flow.

communication, and less data reuse potential over their predecessor networks, such as ResNet [15] or VGG [16]. At the same time, the modular design, combined with the structural sparsity of DSCNNs, allows the designer to systematically trade between algorithmic accuracy, and computational demand, via tunable knobs that vary the sparsity of the network, e.g., varying degree of width multiplication in MobileNet-V2.

The structural sparsity of DSCNNs makes existing DSAs, e.g., VTA or Tensor Cores, less suitable for efficient execution of DSCNNs, as the current DSAs have been often designed for dense operations with highly regular data access and high data reuse. At the same time, current DSAs are often optimized for a single design point in isolation, which limits their efficiency when running DSCNNs. For instance, they convert sparse convolutions to dense matrices (e.g., depthwise to group-convolution transform), which leads to higher computational overhead than the original DSCNNs, while delivering the same accuracy. As an example, VTA had to make a specialized version of MobileNet, which they call MobileNetG, to remove depthwise separable convolution and make it running efficiently on systolic array implemented on FPGAs. FPGA

implementation introduced in [8] has massive data movements as a result of their configurable data path design which results in high latency. Also such type of fixed architectures adopted in [8, 17, 18, 19] makes it difficult to achieve scalability to support modern DSCNNs, e.g. EfficientNet.

This chapter proposes a fully functional framework called DeepDive for an agile, power-efficient execution of DSCNNs on edge FPGAs. DeepDive offers a novel architecture for efficient execution of DSCNNs, combined with a vertical algorithm/architecture optimization and synthesis on edge FPGAs. The framework is designed to identify key heterogeneous Compute Units (CUs), to fully support DSCNNs with heterogeneous convolutional operations, such as group, depthwise, and pointwise convolution. Fig. 3.1 abstracts DeepDive design flow. At the front-end, DeepDive receives the network description model (e.g., PyTorch), and optimizes the model based on the FPGA-aware training and online quantization. This includes algorithm-specific fusing of batch normalization and convolutional operators, which reduces the computation by ~4%, and extremely low-bit per-channel-quantization across all separable convolution layers. The output of the front-end will be *QNet*, which contains all of the meta-data regarding the FPGA-aware trained as well as quantized network model. At the back-end, DeepDive relies on the recent advances in High-Level Synthesis (HLS) and shifts the optimization abstraction to pre-RTL design. The *Network SoC Compiler* creates a customized memory path and synthesizable model of the entire hardware accelerator for Programmable Logic (PL) based on pre-designed CUs and provided convolution operators. It also generates the host CPU code running on ARM cores located in the Processing System (PS) side of SoC for synchronization and scheduling. The host code, bundled with a scheduler, enables the DeepDive back-end system to support multiple run-time software stacks such as Pynq and Linux. The key contributions are:

- The structure and the flexibility of DeepDive enables an agile framework to sup-

port the fast-growing and up-coming DSCNNs. To the best of our knowledge, this work is the first scalable solution with the support of recently introduced EfficientNet DSCNN families.

- It proposes a novel scalable vertical framework for the execution of DSCNN on FPGAs. The vertical integration and library-based operation mapping enables true comprehensive design space exploration on FPGAs.

The rest of this article is organized as the following: Section 3.3 discuss the algorithmic properties of DSCNNs and further motivates DeepDive. Section 3.4 presents DeepDive's front-end, focusing on FPGA-aware training and online quantization. Section 3.5 details DeepDive's back-end architecture and design flow. Section 3.6 presents DeepDive's execution results on Xilinx's ZCU102 FPGA and comparison against state-of-the-art solutions. Section 3.2 reviews the related work. Finally, Section 5.5 concludes this chapter.

3.2 Related Work

Modern CNN accelerators can be divided into two main categories: single compute engine [1, 2, 3, 4, 5, 6], and multiple streaming compute engines [3, 20, 21, 22, 23, 24]. Single compute-engine accelerators are typically a systolic array of processing elements (PEs) that execute the target CNN layer-by-layer sequentially. They have a versatile solution to support different CNNs with the cost of some execution deficiencies. In contrast, streaming architectures consist of multiple dedicated hardware blocks, customized for the target CNN's layers running in producer/consumer fashion. While achieving relatively higher efficiency, they have less scalability to support different networks [25, 26].

Many recent frameworks have proposed a vertical design flow from algorithm to the hardware [1, 3, 7, 20, 22]. However, the primary focus is on optimizing classical CNNs with dense operation with regular memory access, such as YOLO and ResNet

network family. One notable example of single-engine architecture is DNNWeaver [1]. It offers customizable, hand-optimized RTL templates capable of shrinking or expanding the architecture based on the target CNN workload and target device hardware constraints. The templates support common CNN layer operations such as standard convolution, pooling, and batch normalization. However, the design-flow is not autonomous as it requires the user to define the network topology and layer structure. Wei et al. [5] designed a novel 2D systolic array that localizes data shifting to between neighboring PEs. This removes the need for multiplexers and simplifies the routing complexity, allowing for higher throughput. They also employ a custom C-based front-end, which, similar to [1], requires user interaction to define the nested convolutional loop using custom pragmas in C++. The custom front-end makes it more challenging to integrate with existing high-level DNN libraries (PyTorch, TensorFlow, Caffe, etc). VTA is another recently introduced approach, which presents a versatile hardware solution to support different dense CNNs. VTA enjoys the generality by adapting instruction-based scheduling and flexible systolic array. However, this generality leads to more power dissipation. Another aspect that should be considered is that solutions based on versatile systolic arrays intrinsically do not support depthwise convolutions due to introduced sparsity in these types of convolutions; thus, users need to convert the depthwise convolutions to group-convolution to execute a DSCNN on designs similar to VTA. All these succumb to more power dissipation and memory transactions, which lead to having an inefficient hardware solution for DSCNNs.

The design proposed in [27] presents a framework to minimize the complexity and the model size of dense CNN by mapping normal convolution to depthwise separable convolution. Similarly, TuRF [28] replaces standard convolution layers with depth-wise separable convolution and applies layer fusion to enhance the performance of dense networks. The design presented by [17] is another hardware accelerator based on

matrix multiplication and customized adder-tree to support MobileNet-V2. However, their fixed design platform is not scalable to support fast-growing and forthcoming DSCNNs. A parallel acceleration scheme proposed in [8], demonstrates computing reusability with design reconfigurability. However, the accelerator suffers from massive data movements due to frequent reads and writebacks to the DDR because of the lack of fused layer execution. Moreover, the design-flow is not autonomous and requires the user to define the layer structure. A MobileNet-V2 based hardware accelerator on FP32 computation is presented in [18]. DPU [19] is another solution to support MobileNet-V2 based on an optimized RTL hardware model with a dedicated operator for depthwise; however, it cannot be considered as a versatile solution to support DSCNNs due to lake of support for swish activation function and pointwise multiplication. To the best of our knowledge, none of the above approaches present a fully vertical framework to implement the-state-of-the-art DSCNN architectures, e.g., EfficientNet family.

3.3 Algorithmic Principles of Deep Separable CNNs

DSCNNs [9, 12, 13, 14] have emerged as a new paradigm to achieve higher accuracy with relatively fewer parameters and operations over the classical CNNs. The efficiency of DSCNNs stems from their structural sparsity, combined with a modular configurable network topology, that can be scaled up or down, depending on desired accuracy and corresponding computational overhead. In this section, we define the basic principles and structural properties of DSCNN. For ease of access, we summarized the symbols that appeared in this chapter and their description in Table 3.1. These symbols will be used throughout this chapter.

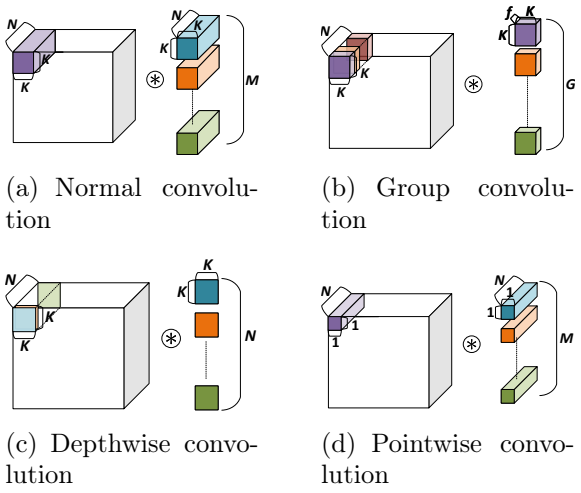
Fig. 3.2(a) shows normal convolution filters with the shape of $M \times N \times K \times K$; thus, the computational cost of normal convolution is $C = H \times W \times K^2 \times N \times M$. Group-convolution, shown in Fig. 3.2(b), minimizes the computation cost of a convolution operator by grouping its channel in G receptions, reducing computation to

Table 3.1: List of symbols

Item	Parameter	Description
1	N	Input channel size
2	M	Output channel size
3	K	Kernel size
4	H	Height of input feature
5	W	Width of input feature
6	G	Group size
7	BW	Bit-width
8	α	Width multiplier
9	k	Number of classes

C/G, where $M = f \cdot G \mid f \in \mathbb{N}$. Depthwise convolution[14, 29] is an extreme case of group-convolution, where $G = N, f = 1$. In this case, each filter is applied to each input channel individually based on Fig. 3.2(c), and in contrast to the normal convolution, there is no reduction (summation) across channels. Pointwise convolution is another type of operator which minimizes the computation by not capturing spatial dependencies within a frame pixels by setting the kernel size to 1×1.

As mentioned earlier, depthwise convolution minimizes computation by removing reduction along the input channels; thus, it is not able to capture the channel-wise information. In the same fashion, pointwise convolution reduces the computation complexity by removing spatial filtering, while it has a full reduction in channel depth. Depthwise separable convolution, used in MobileNet-V1 [14], is an integrated operator composed of a depthwise convolution, followed by pointwise convolution, in order to capture information in both spatial and channel domains, respectively. However, there is still information loss as features move along the network depth and are embedded into lower-dimensional space. MobileNet-V2 [12] introduced inverted residual connections to its previous network, further reducing both multiply-add operations, and model size, without sacrificing the network accuracy. The idea of residual connections was inspired by the ResNet [15] architecture to minimize information loss and speed up the training phase. Fig. 3.3 shows the structure of the Inverted Residual Block (IRB). IRB consists of a pointwise (expansion) convolution, followed



(a) Normal convolution

(b) Group convolution

(c) Depthwise convolution

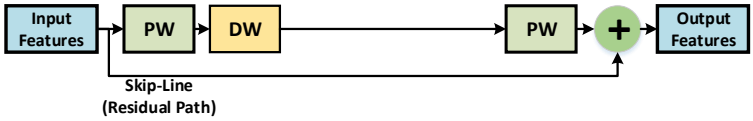
(d) Pointwise convolution

Figure 3.2: Different convolutional operators.

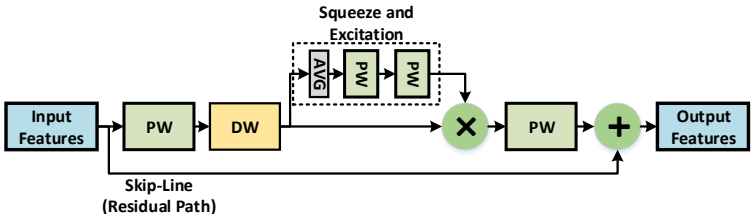
by a depthwise convolution, followed by another pointwise (projection) convolution, to embed the features in a lower dimension. The MobileNet-V2 can control IRB layer input channel width, i.e., N, by altering the α, which changes N to $\alpha \times N$. The $\alpha = 1$ is the baseline model. Selecting $\alpha < 1$ can reduces the computational complexity and the model size quadratically by roughly α^2. We have examined the effect of this knobs and image input size on the final hardware performance and its accuracy in Section 2.5.

Another recently introduced example is EfficientNet, which further optimizes the IRB by adding Squeeze and Excitation (SE) blocks. Fig. 3.3(b) presents the EfficientNet IRB with SE block. The SE block consists of a squeeze operation that captures the global spatial features, followed by an excitation operation that uses a gating function to allow important features to be captured while ignoring the rest. Traditionally, the normal sigmoid is used as the gating function for the SE block, but is replaced with the hard sigmoid to further reduce computation complexity. The hard sigmoid is a non-smooth approximation of the sigmoid function and is described as:

$$\frac{ReLU6(x+3)}{6}, \tag{3.1}$$



(a) Inverted Residual Block: MobileNet-V2



(b) Inverted Residual Block: EfficientNet

Figure 3.3: Inverted Residual Block (IRB) for MobileNet-V2 (a) and EfficientNet (b), respectively. The illustration of Batch Normalization and Activation layers repeated after each convolution are ignored.

$$ReLU6(x) = \begin{cases} x, & \text{if } 0 \leq x \leq 6 \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

The design principles of DSCNNs result in relatively higher computational sparsity due to heterogeneous computing operators that cannot share hardware resources. Depthwise convolution accumulates only across the spatial axis and needs only $K \times K$ fused-multiply-add (FMA) operations since its weight shape is $[M, 1, K, K]$. Since versatile systolic arrays are often designed to support both spatial and channel accumulation, they perform more FMA operations. They map depthwise to matrix multiplication problem by kernel zero-padding and reshaping appropriately; however, the cost of memory real estate, and the redundant computation demand, are not affordable for resource-constrained hardware platforms.

In next, we introduce DeepDive as a fully vertical and versatile solution to support sparse operators introduced in DSCNNs. As case studies, we selected MobileNet-V2 and EfficientNet as two examples of DSCNNs, and we thoroughly elaborate their implementation with the aid of DeepDive in section 3.6.1 and 3.6.2, respectively.

3.4 DeepDive: Front-end

This section describes the front-end of DeepDive, which brings hardware-awareness into training DSCNNs. Fig. 3.4 illustrates the main components of the front-end and their corresponding output. The procedure starts by feeding a pre-trained floating-point network into the DeepDive. The *Batch-Norm Fusing* merges the batch normalization operator into the convolution in order to remove any floating-point operations in the final hardware solution. Next, *Online Channel-wise Low-Bit Quantization* quantizes while training the fused network at extremely low-bit resolutions (e.g., 3-6 bit) across all channels within separable layers. Then, the trained network will be calibrated by extracting the minimum and maximum values across all channels per layer of the network. The *Post-Trained Model Quantization* then uses these acquired ranges to fuse the activation layer, i.e., ReLU6, into the convolution operator. The outcome, *QNet*, consists of only convolution operators that have had their output set to the minimum and maximum quantized value automatically—when they are less than 0 and greater than 6, respectively. In the following, we explain the details of two important aspects of front-end: (1) Batch-Norm Fusing, and (2) Online Channel-wise Low-Bit Quantization.

3.4.1 Batch-Normalization Fusing

Batch-Normalization (BN) [30] is a linear operator, generally seen following a convolution layer, in order to normalize the output of the convolution. BN improves the training speed and stability of the network. The BN function is defined by Eq. 3.3:

$$\widehat{x} = \gamma \frac{x_j - \mu}{\sqrt{\sigma^2 + \epsilon}} + \xi, \tag{3.3}$$

where γ is BN weight, ξ is its bias, and μ, and σ are mean and variance of training batch calculated during the training, respectively. ϵ is a small constant defined to prevent division by zero. Both γ and β are trainable parameters. For networks where

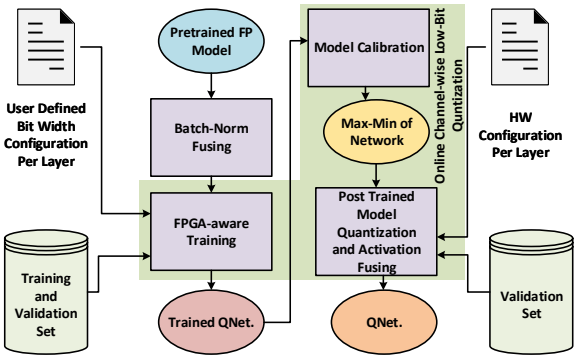


Figure 3.4: DeepDive: Front-end.

its convolution operators are always followed by BN, DeepDive online training fuses these two consecutive layers together by applying following equations:

$$\widehat{v} = (\sigma^2 + \epsilon)^{\frac{-1}{2}}, \tag{3.4}$$

$$\widehat{\omega}_{conv} = \omega_{conv} \times diag(\gamma \cdot \widehat{v}), \tag{3.5}$$

$$\widehat{B}_{conv} = B_{conv} + (\xi - (\gamma \cdot \mu \cdot \widehat{v})), \tag{3.6}$$

where ω_{conv} and B_{conv} are trained weights and biases of convolution operator, respectively. After BN fusion, the network model is ready for quantize-aware training.

3.4.2 Online Channel-wise Low-bit Quantization

Quantization is a well-known approach to compress the network model size, and speed up the computation, by mapping number representations from floating-point single precision (FP32) to integer representation. Due to the malleability of FPGA fabrics, designers can greatly reduce the integer bit-width, while minimizing the introduced quantization error, by training the network for the new representation. DeepDive applies the Range-Based Linear quantization to compress the network weights and biases. Let's define $\mathbb{T} = \{x \mid x \in \mathbb{R}\}$, such that $\mathbb{T}$ is the floating-point pre-trained network model. Function $h : \mathbb{T} \rightarrow \mathbb{Q}$ will map and scale $\mathbb{T}$ to $\mathbb{Q}$, where $\mathbb{Q}$ is

quantized integer representation set. Eq. 3.7 defines function h:

$$x = S(x_q + m_{zp}) \mid x_q, m_{zp} \in \mathbb{Q}, \tag{3.7}$$

where $S \in \mathbb{R}$, is the scaling factor, x_q is the quantized value, and m_{zp} is the zero-point defined to make the right-hand side of Eq. 3.7 equal zero when $x_{fp} = 0$. Based on the range of x_q, two methods of Asymmetric Representation and Symmetric Representation are defined. In asymmetric mode the $min_x = min(x)$ is mapped to 0, while $max_x = max(x)$ is $2^{BW} - 1$, while BW is the bit-width. In contrast, symmetric maps both $[min_x, max_x]$ to $[-(2^{BW-1}),\ 2^{BW-1} - 1]$. MobileNet-V2 uses ReLU6 as its non-linearity function — its output is always positive and less than 6. Therefore, we opted for the asymmetric method, since the negative range of the symmetric representation is not useful, and we are not able to benefit from the full range of representation; thus, it will have an impact on the output accuracy of each activation layer.

DeepDive can quantize a network model per output channel, or per convolution layer. Per layer approach defines h function per whole convolution layer, while per-channel quantization defines $h_j \mid j = 0, \cdots, M - 1$ per each output channel for a convolution operator. For instance, Fig. 3.5 shows the per-channel quantization approach for a depthwise convolution.

After the network is trained and quantized based on the user-provided configuration, the validation set is used again for the network model calibration. The

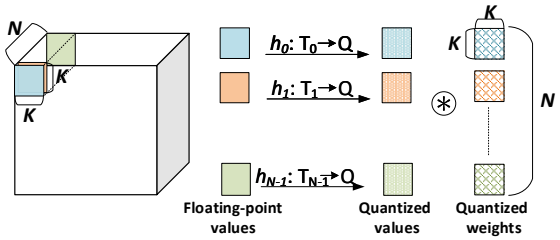


Figure 3.5: Per-channel range-based linear quantization. In this depthwise convolution example, per each N output channel, a separate mapping function is created.

calibration data will be used to make the trained network ready for post-training quantization. In this step, based on the acquired min-max, and the type of quantization, the scaling S and m_{zp} will be recalculated again to re-evaluate h_j, which results in $h_j^{pq} : [0,\ 6] \rightarrow [0,\ 2^{BW} - 1]$. By applying this approach, DeepDive fuses the ReLU6 activation to the convolution operator.

3.5 DeepDive: Back-end

DeepDive's back-end offers a novel micro-architectural approach, and design flow, customized for efficient execution of DSCNNs on edge FPGAs. Fig. 3.6 presents the DeepDive back-end design flow. The heart of DeepDive's back-end is the *Network SoC Compiler*. It receives the design properties from DeepDive's front-end and generates a full design of the system for both hardware (as synthesizable C++ models mapped to FPGAs fabric), software codes, and system configurations. To generate the optimized hardware for DSCNNs, the Network SoC Compiler uses pre-designed highly-optimized RTL micro-architectural blocks or synthesizable C++ model for depthwise, pointwise, and normal convolution operators. In simple words, the Network SoC Compiler generates a network graph containing the network layout and data dependencies. It then creates key heterogeneous CUs, called *QNet Accelerators*, with respect to DeepDive's system architecture.

In the following, at first, we describe micro-architectural details of convolutional operators, and then we discuss the details of the Network SoC compiler and system architecture.

3.5.1 Convolutional Operators

Since DeepDive is specially designed for DSCNNs, it naturally supports all convolutional operations, namely, normal convolution, depthwise convolution, and pointwise convolution. Each convolution operator buffers minimum job data size, which is necessary to start the computation, with the assumption that the network parameters

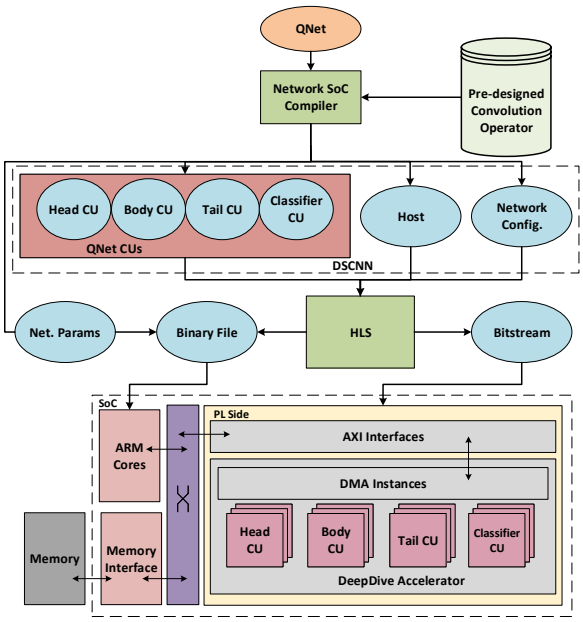


Figure 3.6: DeepDive: Back-end.

necessary for computing are transferred to internal memory, and that the intermediate feature maps are streamed in and out. These operators are pipelined and parallelized in a way that is ideal for both memory-bound and compute-bound operations. The heart of a convolutional operator is a reconfigurable *Direct Convolution* core with different degrees of parallelism. The amount of parallelism defines the utilization, and parallel read/write ports required by the scratchpad or local buffers. This flexibility allows the Network SoC Compiler to manage the resources efficiently by tweaking

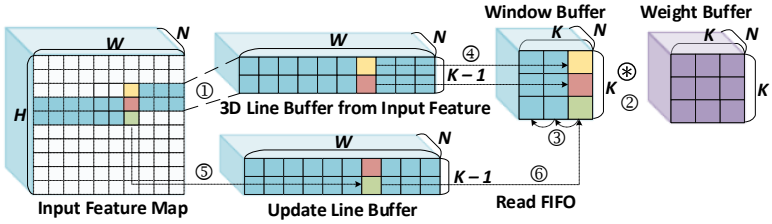


Figure 3.7: Shift and update mechanism of Window and Line Buffer. ① Line Buffer is filled with input feature data. ② Window Buffer is convoluted with weights. ③ The data in window is left shifted. ④ New data from the line buffer is copied in to the window. ⑤ & ⑥ Data from the FIFO is then copied into the line buffer and window buffer. All the Data Movements are pipelined.

the parallelism knobs to achieve the best performance (will be further discussed in section 3.5.2). Next, we elaborate on each operator from the design standpoint. In addition, we formulate the amount of parallelism per each convolutional operator.

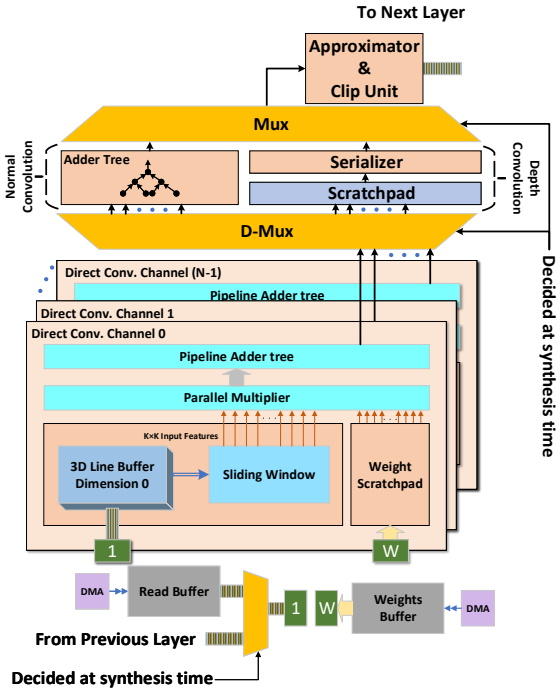


Figure 3.8: Schematic block diagram of depthwise and normal convolution.

3.5.1.1 Depthwise Convolution

The Depthwise convolution uses a 3D line buffer and 3D window to perform direct convolution. The input feature is streamed into a line buffer and then copied into a window buffer with parallel read access, as shown in Fig. 3.7. Once the computation is finished, the data in the computation core will be flushed and reloaded with the new one from the line buffer. The hardware design ensures the data movement involved in this process is fully pipelined, and the initiation interval is limited to a single cycle. Computation starts as soon as the required amount of data is streamed from the main memory. For the current design, the max achievable parallelism is limited to the K and N.

Fig. 3.8 presents the micro-architecture of depthwise and normal convolution oper-

ators. As depicted in Fig. 3.8, the selected input is read in streaming fashion into the 3D line buffer and then copied into the sliding window. The weights are burst read into the weight scratch pad. The Sliding Window and the Weight scratchpad have multiple read ports. Every channel of the input is processed by the direct convolution compute core. The direct convolution compute core has a parallel multiplier, and a pipelined adder tree, together which carryout the MAC operation, followed by the Approximator and Clip unit. This unit truncates, or rounds, the results and then clips them to $[0,\ 2^{BW}-1]$ based on the quantization parameters extracted at the front-end for this operator. Therefore, this unit also acts as the ReLU6 activation layer defined in MobileNet-V2 or EfficientNet. The depthwise convolution is more sparse, and has the least amount of data reuse. The maximum parallel operations are calculated as the following:

$$ParallelOps = K^{dw}_{max} \times K^{dw}_{max} \times N^{dw}_{max}, \tag{3.8}$$

In Eq. 3.8, K^{dw}_{max}, and N^{dw}_{max} are the maximum kernel size and maximum input-channel across all the depthwise convolutions in the network, respectively.

3.5.1.2 Normal Convolution

The DSCNN has one normal convolution, and it is the first operator to embed patterns from both spatial and channel dimensions from the given input image. Since the next layer after normal convolution is depthwise, it is essential to generate output pixels column-wise (spatial dimension) so the depthwise can start the job immediately. Therefore, we improve the parallelism level by having a dedicated adder tree located after the direct convolution kernel for the input channel reduction. The block diagram of normal convolution is, also shown in Fig. 3.8. The parallelism in normal

convolution is across kernel size and input channels — described in the following:

$$ParallelOps = K^{nc}_{max} \times K^{nc}_{max} \times N^{nc}_{max}, \tag{3.9}$$

where $N^{nc}_{MaxSize}$ is the maximum input channel size, and K^{nc}_{max} is the maximum kernel size, assigned from all normal convolution. Normal convolution has slightly more data movements compared to the depthwise convolution due to the pipelined adder tree implemented at the end of direct convolution core.

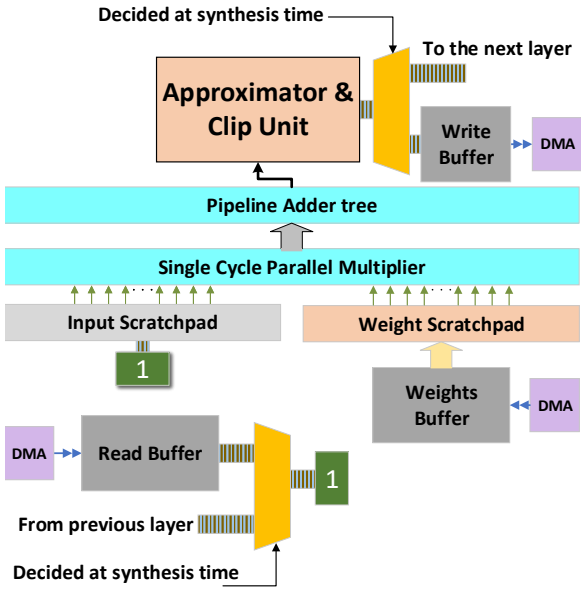


Figure 3.9: Schematic block diagram of pointwise convolution.

3.5.1.3 Pointwise Convolution

Due to the dense operation of pointwise, the design of this operator can be similar to the design of a general matrix multiplication, which is well suited for the systolic array. With maximum data reuse, this operator can leverage maximum parallelism. It has both fewer algorithmic, and fewer data movement complexity, which makes it best fit for a high amount of parallelism. Fig. 3.9 shows the structure of pointwise convolution operator. The required input is directly read into the input scratchpad from the read buffer. The weights are burst read into the weight scratchpad. The input buffer and

the weight scratchpad have multiple read ports for parallel data access. The single-cycle parallel multiplier and the adder tree take advantage of the multiple ports to perform the MAC operations in parallel fashion. The amount of parallelism for our design is across the input channels.

$$ParallelOps = N_{max}^{PW_{type}}, \tag{3.10}$$

where $N_{max}^{PW_{type}}$ is the maximum input channel size across all the specific *type* (eg. projection or expansion pointwise in the MobileNet-V2) of pointwise convolutions mapped to specific compute unit.

3.5.2 Network SoC Compiler

The Network SoC Compiler observes the network graph, the targeted hardware device, and existing pre-designed synthesizable C++ IPs for convolution, and then translates the network graph by grouping the convolutional operators into customized *QNet* CUs with respect to system architecture. It tweaks the hardware architectural knobs to maximize parallelism, fusing as many convolutional operators as possible to reduce the number of shared memory transactions, and increase the overlap between computation and memory latency. Based on the repetitive pattern, it wraps the convolution operators in four different heterogeneous CUs: ① The *Head CU* generally consists of normal convolution followed by a special case of IRB which is only called once; ② The *Body CU* invokes IRB since it has maximum repetitions based on the DSCNNs architectures; ③ The *Tail CU* usually consists of pointwise convolution followed by Average Pooling to embed the features and make them ready in respect of size and shape for the classifier; ④ Finally, the mapping of Tail CU output to $k-$classes is accomplished by *Classifier CU*.

Below, we describe the details of Network SoC Synthesizer including, system architecture, memory organization, Heterogeneous *QNet* CUs, host code scheduling and

CUs management.

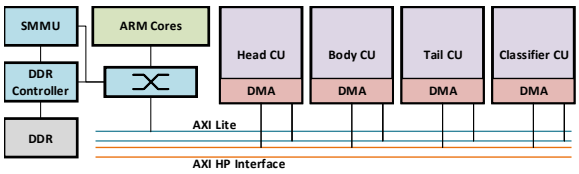


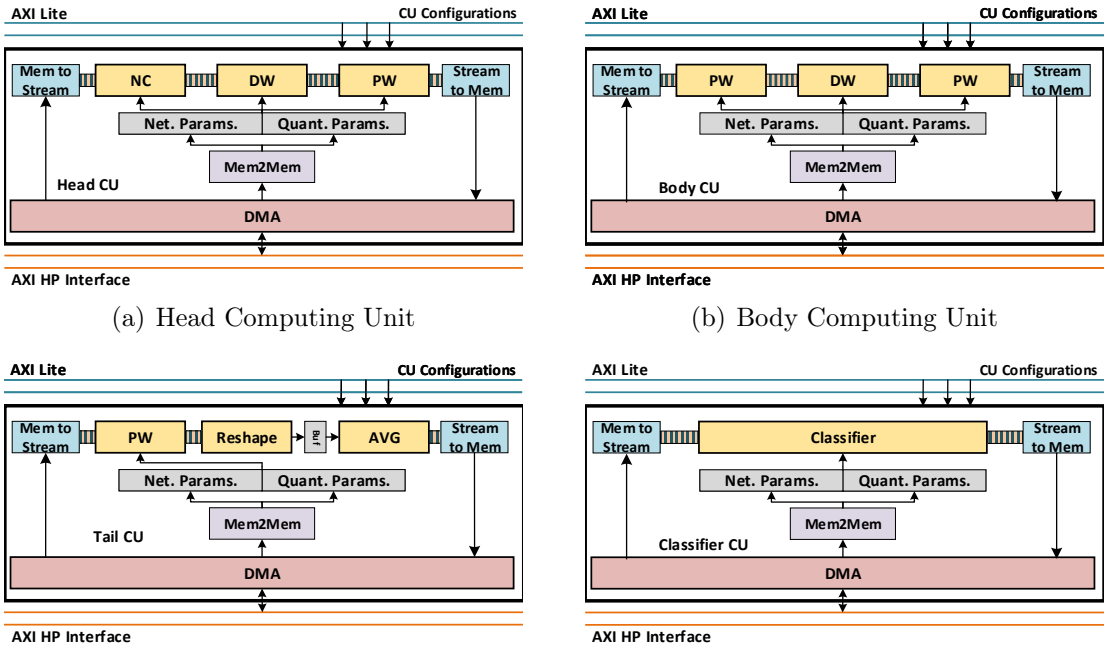
Figure 3.10: System level architecture of DeepDive.

3.5.2.1 DeepDive System Architecture

As emphasized before, the convolutional operators of DSCNNs demonstrate a repetitive structural behavior wherein some either appear once, or they are repeated across the entire network. Depending on the recurrence of the convolutional operators, they are mapped to the Head, Body, Tail, and Classifier CU. Fig. 3.10 shows the system architecture of DeepDive Hardware Accelerator. Each CU has its own dedicated Direct Memory Access (DMA), and its parameters, such as array pointers, N, M, and H, can be configured at runtime via the control bus (e.g., AXI Lite Bus). After configuration, each CU can transfer the input/output features map and weights tensors via streaming channels (e.g., AXI HP Interface) through System Memory Management Unit (SMMU). The composition of CU is parameterized by the buffer shapes, data type widths, and the computation core, which are a few of the architectural knobs provided while designing the hardware accelerator. This makes our design scalable and reconfigurable for DSCNNs. We will discuss our hardware knobs and each CU's internal composition in detail after we explain the memory transactions and management. The CUs are scheduled and pipelined to increase the concurrency.

3.5.2.2 Memory Organization

Each CU has its own dedicated buffer and scratchpad to handle its memory requirements. The memory layout of the on-chip buffers are designed to satisfy the data access pattern required by the convolutional operators, in order to minimize



(a) Head Computing Unit

(b) Body Computing Unit

(c) Tail Computing Unit

(d) Classifier Computing Unit

Figure 3.11: Architecture of *QNet* Heterogeneous Computing Units for MobileNet-V2.

the pipeline depth implemented in the computation core. The memory transactions in the CUs can be categorized into two groups: ① memory to memory transaction, where data is burst read from DDR memory to PL memory, and ② memory to stream transaction, where data is streamed via DMA to or from PL memory. As an example, Fig. 3.11(a) demonstrates the memory transactions for Head CU targeted for MobileNet-V2. Convolutional network parameters like weights, quantization parameters, and biases are burst read from DDR to PL buffers. The input/output feature maps are streamed from DDR to PL. Apart from memory transactions of input/output features between DDR and PL, the inter-CU data transfers within its operators also occurs in streaming fashion, where intermediate feature map data is streamed in-between different convolutional layers. Stream FIFO offers two main advantages, memory and computation latency overlap and data movement reduction between DDR and PL.

3.5.2.3 QNet Heterogeneous CUs

In this subsection, we will explain the heterogeneous CUs, and the available architecture knobs that can be tweaked based on hardware and performance constraints. As mentioned earlier, Network SoC Compiler creates four unique CUs for each DSCNNs. The CUs are completely parameterizable, and customizable, for scalability and flexibility. Following section describes each CU in detail. We also provide illustrative figures for the example of MobileNet-V2.

Head CU: DSCNNs tend to start with a particular pattern, which comprises of a fixed set of layers that are not recurrent in any other part of the network. As explained in the section 3.5.2.2, the Head CU has its own dedicated internal memory for buffers. The data transactions occur in memory-to-memory mode and the intermediate data streams between convolutional layers within the head CU. As an example, Fig. 3.11(a) demonstrates the Head CU for MobileNet-V2 model, which is composed of normal convolution followed by depthwise and pointwise convolution, all fused by FIFO stream. This CU is scheduled once during the course of any DSCNN implementation. After running the head of CU, the repeatable pattern will be merged and mapped to the Body CU explained in the next part.

Body CU: The Body CU is the most important CU within DeepDive's system architecture. It is responsible for executing majority of DSCNNs blocks iteratively. As an example, the IRB, which is the most repetitive block of MobileNet-V2, is entirely mapped to the Body CU. The IRB consists of pointwise (expansion), depthwise, and pointwise (projection) layers, all running concurrently in a fused fashion within the Body CU. Fig. 3.11(b) shows the structure of this CU for MobileNet-V2. Upon examining the network graph of DSCNNs, we see that occasionally, the IRB needs to perform residual connections. Depending upon the network graph, DeepDive facilitates residual connections implementation within or outside the PL targeted device resources. The Body CU is parameterized so as to support both memory-bound IRBs,

which ideally are earlier blocks of DSCNNs, and compute-bound IRBs, which tend to be later blocks of DSCNNs. Therefore, the network SoC compiler configures the Body CU with maximum buffer size needed by memory-bound IRBs, and maximum level of parallelism to meet the demand imposed by compute-bound IRBs. At the same time, the Body CU supports convolution operations with variable stride over different IRBs. These features increase the framework inclusiveness by supporting multiple IRB scenarios within the same DSCNN.

Tail CU: The Tail CU consists of the last layers of DSCNNs. The task of this CU is to make the embedded feature size ready for the dense layer implemented in the Classifier CU. Fig. 3.11(c) represents the structure of Tail CU in MobileNet-V2. This CU is comprised of a single pointwise convolution operator, followed by an average pool. As intermediate feature maps are streamed from layer to layer in a channel-wise fashion, the reshape block reorders the memory layout of the feature map in a column-wise mode. Therefore, the average pooling can accumulate the input on-the-fly and stream out.

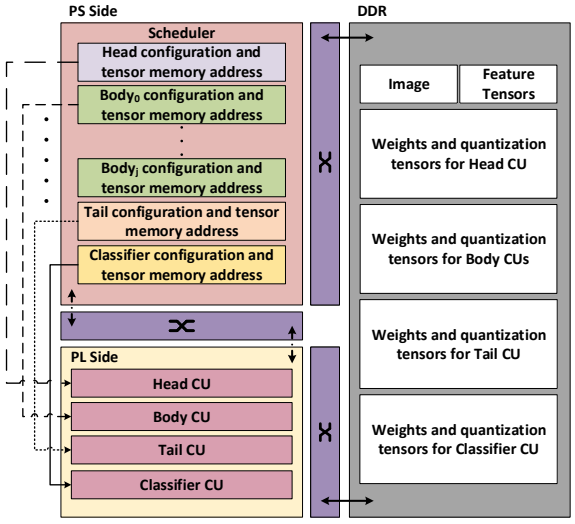


Figure 3.12: Host level scheduling and memory footprint of CUs.

Classifier CU: The last Compute Unit is the Classifier CU, which concludes the DSCNN implementation. Fig. 3.11(d) represents the MobileNet-V2 Classifier CU.

Table 3.2: Effect of altering α and H for fixed $BW = 4$

α	1					0.75					0.5					0.35				
H	224	192	160	128	96	224	192	160	128	96	224	192	160	128	96	224	192	160	128	96
Params(Mb)	13.31	13.31	13.31	13.31	13.31	10.01	10.01	10.01	10.01	10.01	7.48	7.48	7.48	7.48	7.48	6.37	6.37	6.37	6.37	6.37
#Ops(M)	313.621	230.755	160.638	103.269	58.649	220.326	162.212	113.038	72.805	41.513	104.164	76.868	53.772	34.875	20.177	64.835	47.973	33.706	22.033	12.953
Top1(%)	69.07	67.256	65.78	62.3	56.036	66.404	64.364	59.928	53.112	43.002	59.502	57.452	52.608	45.316	34.88	54.43	51.214	46.59	39.328	27.2

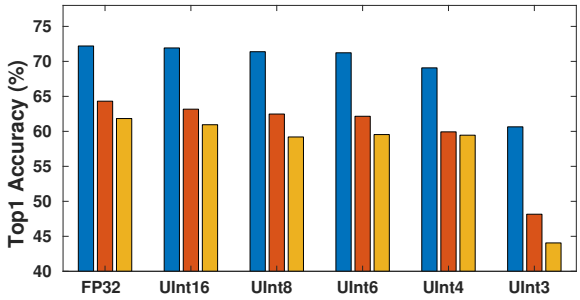
Similar to others, this CU is parameterized such that the parallelism across the computing core can be adjusted based on the available hardware resources. Classifier CU comprises compute-bound operations and has a similar configuration to the point-wise convolutional operators.

3.5.2.4 Host Code Scheduling and CUs Management

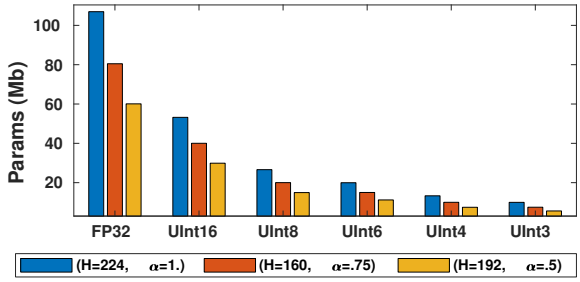
Finally, the Network Soc Compiler also manages the host-level scheduling of CUs. Fig. 3.12 visualizes the CUs scheduling and their memory footprints on shared memory. The host or PS initializes the DDR with network models and quantization parameters. The DeepDive back-end generates the memory layout so that the network data region is shared between PL and PS. Therefore at each CU invocation, the PS only passes the data pointer, and the PL fetches the data based on the provided pointer rather than copying the data to its region. This memory layout will remove the necessity of copying data between the PL and PS memory region. The host starts scheduling procedure by configuring the Head CU with appropriate memory pointer addresses, offsets, network parameters, and network configuration, i.e., M, N, H, which are compiled into network configuration header files. When Head CU completes execution, it writes back the data in feature tensors and interrupts the host CPU. Following the same trend, the host will schedule the Body CUs for j times, where j is the number of Body CU invocations calculated based on CU's mapping. Host CPU then schedules the Tail CU, which executes the compute-bound operations quickly. And finally, the last call is to the Classifier CU, which will update the content of feature tensor needed by the softmax layer to calculate the confidence. Host CPU creates a sequential yet fused scheduling and management of CUs for DSCNNs.

3.6 Experimental Results

We have chosen the Xilinx Zynq UltraScale+ MPSoC ZCU102 evaluation board, which has XCZU9EG chip, to demonstrate the capabilities of DeepDive. The ARM processors host Ubuntu 16.04, running at 1.2GHz; the OS can program the FPGA fabric at runtime. We also use Vivado HLS 2018.3 to synthesize the network models compiled by DeepDive. The FPS and power consumption reported for DeepDive are based on *QNet* accelerator running at 200MHz. We targeted MobileNet-V2 and EfficientNet networks as two cases of DSCNNs. The Top-1 accuracy reported in this section is based on training and evaluating the network on the ImageNet dataset. Since the input image has a square shape, we reported only H as input feature size. Later, we elaborate the design exploration and implementation of each one of these networks as a case study.



(a) Top1 Accuracy for three different design points.



(b) Model size for three different design points.

Figure 3.13: The effect of different computation types on Top1-accuracy and model size. Based on Fig. 3.13(a), UInt4 has almost accuracy similar to floating-point, while a notable drop can be observed for UInt3. Also, Fig. 3.13(b) shows integer quantization causes an exponential decrease in the model size.

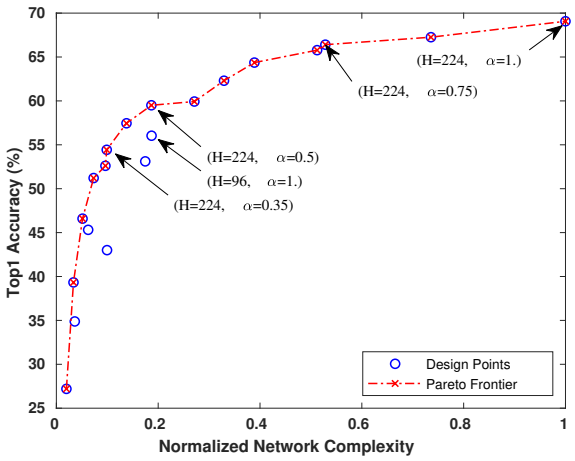


Figure 3.14: Top1-Network Complexity Pareto front. Design point $(H = 96, \alpha = 1)$ has similar network complexity while is Top1 accuracy is less than $(H = 224, \alpha = 0.5)$.

3.6.1 Case Study: MobileNet-V2

The procedure starts from a PyTorch model of MobileNet-V2, pre-trained on ImageNet. At DeepDive's front-end, we configured the FPGA-aware training for different BW based on the channel-wise asymmetric ranged linear quantization. Fig. 3.13 shows the Top-1 accuracy for MobileNet-V2 when its $\alpha = 0.75$ and $H = 160$. As can be seen, DeepDive maintains accuracy with respect to FP32 by reducing the BW to 8 for first normal convolution, and 4 for the rest of the layers, respectively. The per layer-specific quantization compresses the model size with a ratio of 8, with 4.4% degradation in Top1 accuracy. The results demonstrate a dramatic drop in accuracy for $BW = 3$. For the rest of this case study, $BW = 4$, as it achieves competitive accuracy with considerably smaller model size.

3.6.1.1 Design Exploration

The front-end is configured to re-train, quantize, and calibrate the network for different α and H values. Table 3.2 summarizes the model size, operation numbers and Top1 accuracy per each design point. Based on Table 3.2, we observe that model size is only effected by α, while the number of operation number is a function of both α and H. Top1 accuracy is also a function of both H and α; however, it is not a

linear relationship. For instance, design point ($H = 224, \alpha = 0.75$) has better Top1 accuracy compared to design point ($H = 160, \alpha = 1$) while its model size is 33% less than the latter one. Therefore, we introduce the network complexity as the product of the network model size and network operation number to consider both of them.

Table 3.3: Effect of altering α and H for fixed $BW = 4$ at 200Mhz on FPS and FPGA Resource Utilization

α	0.75					0.5					0.35				
H	224	192	160	128	96	224	192	160	128	96	224	192	160	128	96
FPS	11	14	18	22	28	16	19	25	30	37	20	25	31	40	51
Power(W)	3.25	3.10	3.03	2.93	2.88	2.97	2.83	2.86	2.84	2.83	2.97	2.78	2.76	2.72	2.70
DSP(%)	57	57	58	57	57	37	37	37	37	37	24	24	24	24	24
LUTs(%)	75	74	76	74	74	71	70	70	70	70	68	67	67	67	67
BRAM(%)	96	96	97	92	90	92	91	89	88	87	84	84	82	81	80

Fig. 3.14 depicts the Top1-Network Complexity Pareto front. The network complexity helps the front-end to measure the final hardware complexity at a higher level of abstraction. We annotate the starting point of each α in this figure and one non-Pareto point for the sake of comparison. Here, we observed that the design point ($H = 96, \alpha = 1$) has approximately the same network complexity with respect to ($H = 224, \alpha = 0.5$), while its Top1 accuracy is almost 4% less than top achievable accuracy.

3.6.1.2 Accuracy Density

In order to illustrates the performance of DeepDive front-end, we introduce Accuracy Density (ρ) as follows:

$$\rho = \frac{Top1}{Params \times Ops}. \tag{3.11}$$

where $Params$ is in Mbit, and the number of operations (Ops) is in Giga. The ResNet-18, SqueezeNet, and MobileNet-V2 models, depicted in Fig. 3.15, are selected from Xilinx Model Zoo [1], quantized in 8-bit. We considered networks with accuracy higher than 60% with the exception of SqueezeNet, and compared the accuracy density of the Xilinx models against the DeepDive compressed model. *ResNet-18* as an

[1] https://github.com/Xilinx/AI-Model-Zoo

example of a classical model with an accuracy of 66.94% has the lowest density in respect to other models. Although the baseline 8-bit quantized MobileNet-V2 with the precision of 63.54% has the smallest model size and the number of operations concerning ResNet-18; however, it is outperformed by DeepDive model configuration ($H = 224, \alpha = 1$) as its accuracy is 5.53% better than Xilinx MobileNet-V2 while reducing the model size with the aim of extreme 4-bit quantization. The ($H = 160, \alpha = 1$), and ($H = 192, \alpha = 0.75$) configurations have almost 1% better accuracy than the Xilinx MobileNet-V2, while improving ρ by 4.11× and 5.35×, respectively.

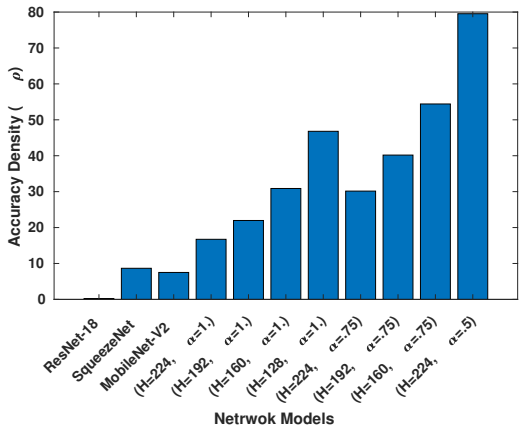


Figure 3.15: The Accuracy Density ρ comparison of three networks from Xilinx Model Zoo, *Resnet-18, SqueezeNet, and MobileNet-V2* quantized in 8-bit, and seven configurations compressed by DeepDive front-end.

3.6.2 Case Study: EfficientNet

Table 3.4: Compressed EfficientNet Algorithmic Specs and FPGA Resource Utilization with fixed BW = 4, Frequency = 200 MHz

Algorithmic Parameters				Hardware Parameters			
H	Parameters (Mb)	#Ops (M)	Top1 (%)	FPS	DSP (%)	LUTs (%)	BRAM (%)
128	7.81	4.914	55.02	35	90	80	68

The baseline EfficientNet model was intentionally designed to be larger than MobileNet-V2. While this might be ideal for state-of-the-art accuracy, it was not suitable for low-power embedded devices. Taking advantage of the compound model scaling

factors introduced in [13], we were able to compress the model using smaller α, network depth, and H, to achieve a model size capable of running on edge devices. The algorithmic details and hardware resource utilization of this model can be seen in Table 3.4.

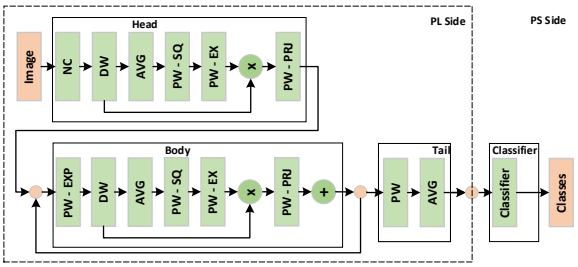


Figure 3.16: EfficientNet mapped to CUs.

Mapping: EfficientNet is structurally different as compared to MobileNet-V2. Fig. 3.16 shows the mapping of EfficientNet to the CUs. The squeeze and excitation convolutional operators are represented as PW-SQ and PW-EX, respectively. DeepDive takes advantage of EfficientNet architecture by fusing more convolutional operators together. EfficientNet comparatively has a larger body than the MobileNet-V2, with six layers fused. This mapping helps in achieving better performance by reducing more memory transactions by invoking the Body CU only nine times. For the case of EfficientNet, we excluded the classifier from mapping and also comparison.

3.7 Conclusion

This chapter introduced DeepDive, as a fully functional framework for an agile, power-efficient execution of DSCNNs on edge FPGAs. DeepDive offers a vertical algorithm/architecture optimization, starting from the network description model down to full system synthesis and implementation. At the front-end, DeepDive performs high-level optimization such as BN fusing, and Online channel-wise low-Bit quantization at extremely low-bit resolutions to bring FPGA-awareness when training DSCNNs. At the back-end, Network SoC Compiler receives the design properties from DeepDive's front-end and generates a full design of the system for both hardware

model and software host codes. To generate the optimized hardware for DSCNNs, the Network SoC Compiler uses pre-designed micro-architectural blocks for depthwise, pointwise, and normal convolution operators. For the results, we have synthesized, executed, and validated two state-of-the-art DSCNNs, MobileNet-V2 and EfficientNet on Xilinx's ZCU102 FPGA board. The execution results demonstrated 47.4 and 233.3 FPS/Watt for MobileNet-V2 and a compact version of EfficientNet, respectively. These comparisons showcased how DeepDive improved FPS/Watt by 2.2× and 1.51× over Jetson Nano high and low power modes, respectively. It also enhances FPS/Watt about 2.27× and 37.25× over two other FPGA implementations.

As future work, we plan to improve the back-end of DeepDive to support cloud-based FPGAs such as Alveo family. We plan to extend support for multiple instances of Body CU to improve both latency and throughput. Each body could have a different level of parallelization based on the knobs introduced in Section 3.5.1. The host would also map the IRB layers to the body CUs based on the required computation power. Various Body CUs with varying degrees of parallelization could improve DeepDive without power and hardware resource compromises.

Bibliography

[1] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to fpgas," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 2016, pp. 1–12.

[2] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, pp. 1–1, 05 2017.

[3] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "Dnnbuilder: An automated tool for building high-performance dnn hardware accelerators for fpgas," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '18. New York, NY, USA: ACM, 2018, pp. 56:1–56:8.

[4] B. Dageville, T. Cruanes, M. Zukowski, V. Antonov, A. Avanes, J. Bock, J. Claybaugh, D. Engovatov, M. Hentschel, J. Huang, A. W. Lee, A. Motivala, A. Q. Munir, S. Pelley, P. Povinec, G. Rahn, S. Triantafyllis, and P. Unterbrunner, "The snowflake elastic data warehouse," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16. New York, NY, USA: ACM, 2016, pp. 215–226.

[5] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput cnn inference on fpgas," in *Proceedings of the 54th Annual Design Automation Conference 2017*, ser. DAC '17. New York, NY, USA: ACM, 2017, pp. 29:1–29:6.

[6] C. Zhang, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Towards uniformed

representation and acceleration for deep convolutional neural networks," in *Proceedings of the 35th International Conference on Computer-Aided Design*, ser. ICCAD '16. New York, NY, USA: ACM, 2016, pp. 12:1–12:8.

[7] T. Moreau, T. Chen, L. Vega, J. Roesch, E. Yan, L. Zheng, J. Fromm, Z. Jiang, L. Ceze, C. Guestrin, and A. Krishnamurthy, "A hardware–software blueprint for flexible deep learning specialization," *IEEE Micro*, vol. 39, no. 5, pp. 8–16, 2019.

[8] J. Liao, L. Cai, Y. Xu, and M. He, "Design of accelerator for mobilenet convolutional neural network based on fpga," in *2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, vol. 1, 2019, pp. 1392–1396.

[9] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, June 2018, pp. 6848–6856.

[10] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 5987–5995.

[11] Z. Qin, Z. Zhang, X. Chen, C. Wang, and Y. Peng, "Fd-mobilenet: Improved mobilenet with a fast downsampling strategy," in *2018 25th IEEE International Conference on Image Processing (ICIP)*, Oct 2018, pp. 1363–1367.

[12] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, 2018, pp. 2704–2713.

[13] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, 2019, pp. 6105–6114.

[14] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

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

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[17] L. Bai, Y. Zhao, and X. Huang, "A cnn accelerator on fpga using depthwise separable convolution," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 10, pp. 1415–1419, 2018.

[18] B. Liu, D. Zou, L. Feng, S. Feng, P. Fu, and J. Li, "An fpga-based cnn accelerator integrating depthwise separable convolution," *Electronics*, vol. 8, p. 281, 03 2019.

[19] D. Wu, Y. Zhang, X. Jia, L. Tian, T. Li, L. Sui, D. Xie, and Y. Shan, "A high-performance cnn processor based on fpga for mobilenets," 09 2019, pp. 136–143.

[20] Y. Wang, J. Xu, Y. Han, H. Li, and X. Li, "Deepburning: Automatic generation of fpga-based learning accelerators for the neural network family," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: ACM, 2016, pp. 110:1–110:6.

[21] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: ACM, 2017, pp. 65–74.

[22] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: Automated mapping of convolutional neural networks on fpgas (abstract only)," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: ACM, 2017, pp. 291–292.

[23] K. Abdelouahab, M. Pelcat, J. Serot, C. Bourrasset, and F. Berry, "Tactics to directly map cnn graphs on embedded fpgas," *IEEE Embedded Systems Letters*, pp. 1–4, 2017.

[24] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, Dec. 2018.

[25] C. Baskin, N. Liss, A. Mendelson, and E. Zheltonozhskii, "Streaming architecture for large-scale quantized neural networks on an fpga-based dataflow platform," *CoRR*, vol. abs/1708.00052, 2017.

[26] M. Samragh, M. Javaheripi, and F. Koushanfar, "Codex: Bit-flexible encoding for streaming-based FPGA acceleration of dnns," *CoRR*, vol. abs/1901.05582, 2019.

[27] R. Zhao, X. Niu, and W. Luk, "Automatic optimising cnn with depthwise separable convolution on fpga: (abstact only)," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*,

ser. FPGA '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 285. [Online]. Available: https://doi.org/10.1145/3174243.3174959

[28] R. Zhao, H.-C. Ng, W. Luk, and X. Niu, "Towards efficient convolutional neural network for domain-specific applications on fpga," 08 2018, pp. 147–1477.

[29] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[30] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, pp. 448–456.

CHAPTER 4: ATCN

4.1 Introduction

The astonishing growth in deep learning algorithms has changed how embedded and cyber-physical systems (CPS) process the surrounding environment and has significantly improved the overall CPS performance on delivering their assigned tasks. For instance, the deep learning algorithms and architectures have powered the embedded systems in visual sensing applications such as pedestrian and object tracking [1, 2], action detection [3, 4]. Another dimension of deep learning, which has recently emerged in the edge, is time series analysis and forecasting. Healthcare [5, 6, 7], device health monitoring [8, 9, 10], machine translation [11, 12] are some examples of deep learning use in time sequence analysis.

For most deep learning practitioners, recurrent networks and especially two elaborated models, namely, LSTM [13] and GRU [14], are synonymous with time series analysis due to its notable success in sequence modeling problems such as machine translation, language processing, and device health monitoring. These models interpolate the output based on the current and temporal information, which is learned and captured in the hidden states and propagated through the time from one cell to the next adjacent cell. The propagation chain of hidden state causes two significant issues [15]: 1) gradient instability such as vanishing/exploiting gradients and 2) fewer levels of parallelization due to existing dependencies across the cells.

Temporal Convolutional Networks (TCN) was first proposed based on an adaptation of WaveNet [16] and Time-Delay Neural Network [17]. It orchestrates dilated convolutions in Encoder-Decoder architecture to have a unified framework for action segmentation. Later, Bai et al. [18] designed a Generic TCN (GTCN) architecture for

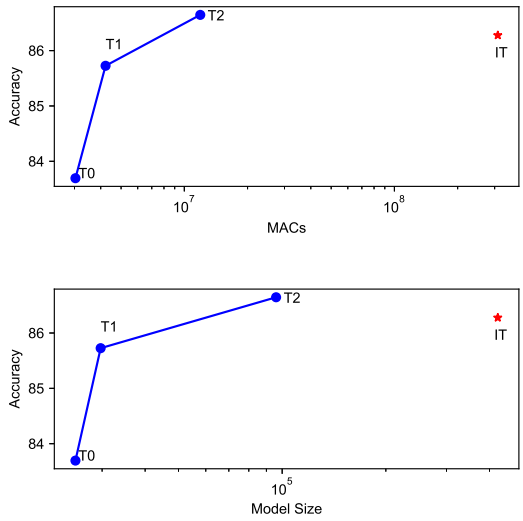


Figure 4.1: Model complexity comparison of three different ATCN families against InceptionTime (IT)

sequence modeling, which outperforms LSTM on time-series and sequence modeling tasks. However, the GTCN suffers from two main drawbacks: 1) the size of dilation increases exponentially by the layer, which prevents the designer from increasing the depth of the network, 2) it uses two standard convolutions per each layer, which is computationally expensive for resource-constrained embedded systems. InceptionTime (IT) [19] is another off-the-shelf CNN to classify time series based on CNN, representing a scalable, accurate solution; however, execution on a microcontroller is entirely out of reach of InceptionTime.

This chapter proposes a novel extension of TCN called ATCN for light-weight processing of time series on embedded and edge devices. We introduced Spectral-Temporal Convolution Block (STCB) to decrease the number of MAC operations and the model size of TCN to make it applicable for embedded devices while maintaining a comparable or better accuracy over IT. Various configurations of these three blocks can be combined to form different ATCN families that can each meet various design constraints. FIG. 4.1 illustrates the capacities and scalability of three different ATCN families on accuracy and model complexity trade-off over 70 benchmarks from the UCR 2018 dataset [20] against IT. The T0 configuration reduces the MACs and

model size by 102.38× and 16.84× over IT, respectively. T1 performance is 4.03% better than T0 and has a 73.59× reduction in MACs, and a 14.23× reduction in model size over IT. Both T0 and T1 can be executed on an ARM Cortex-M7 microcontroller explained in the experimental section in detail. As a final improvement, T3 reduces the MACs and model size for 26.07× and 4.4× over IT while increasing accuracy by 0.37%.

Overall, the key contributions of this chapter are:

- Proposing ATCN, which achieves higher or comparable accuracy over state-of-the-art models with significantly lower computation complexity for embedded devices.
- Creating a network template supported by automated design flow for scalable generation and training different configurations of ATCN concerning the complexity of problem and latency requirements.

The rest of this article is organized as the following: Section 4.2 briefly discusses the use of time-series analysis in embedded and CPS. Section 4.3 provides background on generic TCN and its architecture. In section 4.4, we elaborate on the Temporal-Spectral block, the ATCN architecture, and its hyperparameters. Section 4.5 presents the experimental results including comparison with existing approaches, and finally Section 4.6 concludes this article.

4.2 Related Works

Traditional convolutional neural networks are used in computer vision applications due to their success in capturing the spatial features within a two-dimensional frame. Recently, research has shown that specialized CNNs can recognize patterns in data history to predict future observations. This gives researchers interested in time-series forecasting options to choose from over RNNs, which have been regarded in the community as the established DNN for time-series predictions. In one such case,

Dilated Convolutions (DC) have been shown to achieve state-of-the-art accuracy in sequence tasks. In the first use of DC, WaveNet [21] was designed to synthesize raw audio waveform, and it outperforms the LSTM. Later, Lea et al. [22] proposed TCN, a unified network based on WaveNet DC, for video-based action segmentation. In the same trend, the gated DC was used for the sequence to sequence learning [23]. The proposed approach beats deep LSTM in both execution time and accuracy.

GTCN [15] is a generic architecture designed for sequence modeling. The design of GTCN was based on two main principles: 1) there shouldn't be any information leakage from future to past, 2) the network should be able to receive any arbitrary input length similar to RNN. Since the main fundamental component of GTCN is based on variable-length DC, it brought higher parallelization and flexible receptive field in comparison to RNN. Also, since the gradient flow of GTCN is different from the temporal path of RNN, it is more resistant to the problem of gradient instability. Recent researches have taken advantage of GTCN benefits or similar architectures in their works. In the work of [24], a modified version of GTCN with depth-wise convolution has been used to enhance the speech in time-domain. The DeepGLO [25] is another work that used a global matrix factorization model regularized by a TCN to find global and local temporal in high dimensional time series.

InceptionTime is an ensemble of CNN blocks called Inception Module and proposed as a solution for the Time Series Classification (TSC) problem. The network architecture was constructed based on the Inception-v4 [26] structure, and it employed a larger kernel size to beat enormous and complex models such as the HIVE-COTE [27]. Nonetheless, it is still exceedingly heavy for the microcontroller with limited resources, such as ARM Cortex-M series, or even legacy embedded ARM Cortex-A series CPUs.

The chapter proposes ATCN for embedded and resource-constrained hardware to address time-series domain problems, which is on par with InceptionTime in terms

of accuracy performance. We have put our claim on test in Section 4.5 by comparing ATCN against InceptionTime over 70 benchmarks from UCR time-series datasets. Additionally, we have reported the execution profile of the Cortex-M7 and Cortex-A57 when running ATCN families. We have shown that ATCN improves or maintains the overall system accuracy for these three cases while minimizing computational complexity and model size. In the next section, we study the structure of DC in-depth to prepare the ground for introducing ATCN in Section 4.4.

4.3 Background: Temporal Neural Networks

GTCNs are designed around two basic principles: 1) the convolutional operations are causal, i.e., predictions are made based only on current and past information; 2) the network receives an input sequence of arbitrary length and maps it to an output sequence of the same length [15]. Based on principle number 2, in order to map the final output to an arbitrary size, the output of the last DC output can be connected to a linear layer. This adds flexibility by allowing a final output length to be independent of the input length. The naive causal convolutions, which have a dilation rate of 1, are inherently inefficient as their sequence history scales with size linear to the depth of the network.

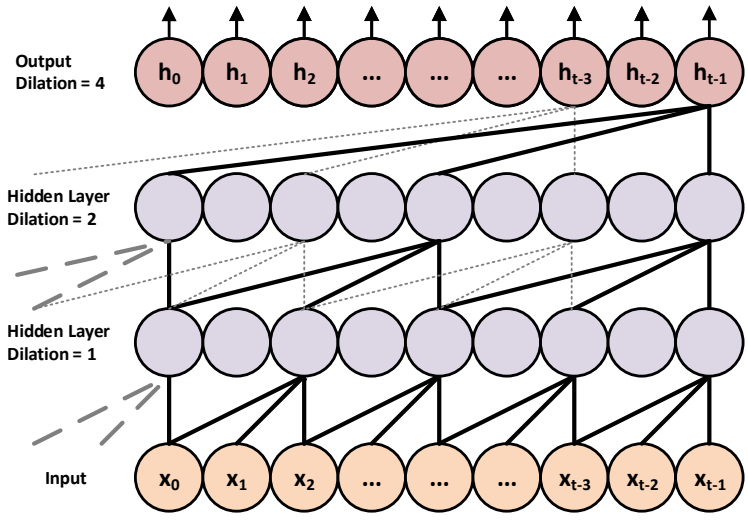


Figure 4.2: Dilated Causal Convolution.

The solution here incorporates dilated convolutions to exponentially scale the receptive field, as shown in Fig. 5.1. The first convolution with dilation rate d=1 maps the input vector $X = [x_1, x_2, \cdots, x_{t-1}]$ to the higher dimension. Then, GTCN increases the d for the next convulsions exponentially to increase the receptive field. The minimum output sequence length, before mapping to the linear layer, can be determined by calculating its receptive field: [28]:

$$rf = 1 + \sum_{l=1}^{L}[k(l) - 1] \times d(l), \tag{4.1}$$

where $l \in \{1, 2, 3, ..., L\}$ is the layers, k is the kernel size, and $d(l)$ is the dilation rate at layer l. This means that as the depth of the network increase, so does the receptive field. The dilated convolution of F on element s of a sequence X is given as:

$$F(s) = (X *_d f)(s) = \sum_{i=0}^{k-1} f(i) \cdot x_{s-d \cdot i}, \tag{4.2}$$

where $X \in \mathbf{R}^n$ is a 1-D input sequence, $*_d$ is dilated convolution operator, $f : \{0, ..., k-1\} \in \mathbf{R}$ is a kernel of size k and d is the dilation rate [15, 28]. For applications requiring a very large rf, it is also essential to provide stability in the later layers subject to the vanishing gradient problem. A popular technique in traditional CNN architectures, the residual block [29], provides a "highway" free of any gated functions, allowing information to flow from the early layers to the last layers unhindered.

These connections can be seen in the final GTCN architecture shown in Fig. 4.3. The GTCN consists of L hidden layer and an optional linear layer to map the input size i to arbitrary output size. Each hidden layer has two regular dilated convolution and two ReLU activation function. There can also be an upsampling unit, such as point-wise convolution, in the first hidden layer of GTCN to map 1-D input sequence to a higher dimension to guarantee the element-wise addition receives tensor of the same dimension.

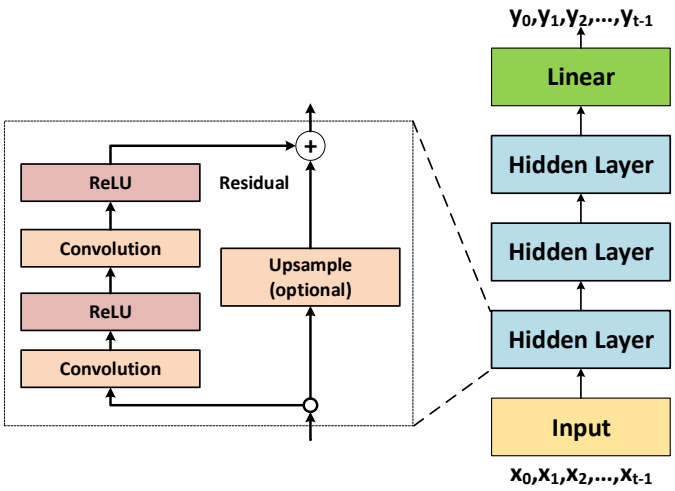


Figure 4.3: Structure of Generic TCN.

The design of GTCN suffers from two problems: 1) exponential growth of dilation size, 2) the existence of two regular convolutions per layer. The exponential growth of dilation size and requirement of having the same length for both input and output of dilated convolution force the network designers to have excessive padding at the higher layers. Also, the implementation of two convolutions blocks per layer makes the GTCN costly for CPS. In the next section, we address the problems mentioned above by introducing ATCN architecture.

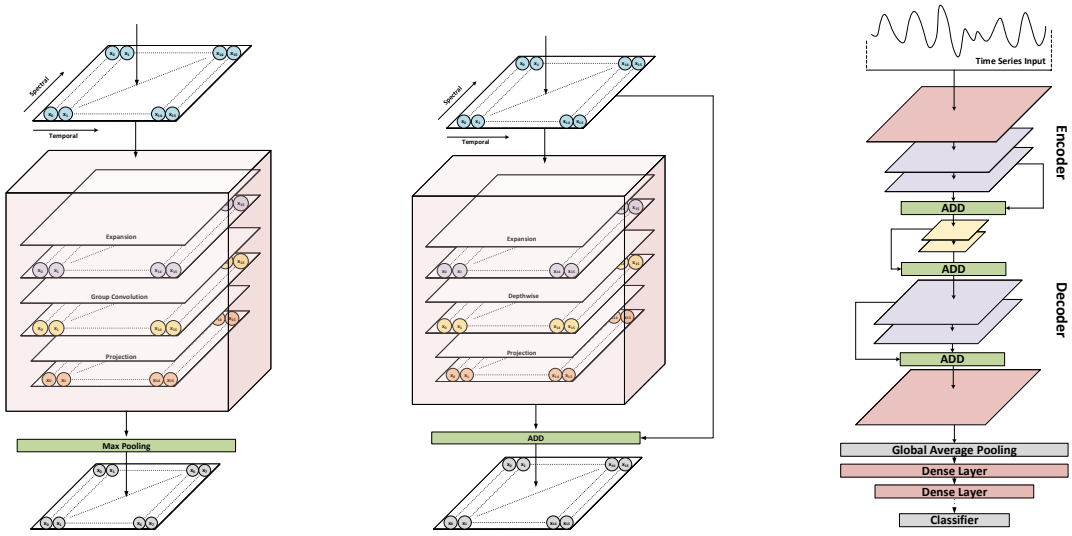
4.4 ATCN: Agile Temporal Convolutional Networks

In this section, we introduce the architecture of ATCN. At first, we discuss the essential components, and then we elaborate on the hyper-parameters, and in the end, we present the ATCN architecture and its model builder.

4.4.1 Network Structure

The ATCN architecture can be created by chaining STCBs and altering their configurations. Each STCB is composed of a pointwise (expansion), a group, and a pointwise (projection) convolution. We visualized the STCB in Fig. 4.4(a). The Max Pooling layers are optional. It lets architects downsample temporal information to

minimize computational complexity while embedding that information in higher or lower dimensions. The extreme case of STCB is when the group size and its input channel size are equal. In this case, the group-convolution is set to a depthwise, and the ATCN network synthesizer will remove the maximum pooling and add a skip-line between element-wise addition and the STCB input. The final architecture of ATCN is shouwn in 4.4(c). The ATCN is a mirrored residual dilated convelutional neural network. It starts with mapping the n-dimnetion input, which is generally 1D for the time series, to higher dimention at firt layer. Then it encodes the data to lower dimention. At encoder parts, the data will be decoded to higher dimmnetion again. Based on the final application, the output of decoder can be used for regression or classification problems. In the rest, we discuss the details of STCB and the network hyper-parameters.



(a) STCB

(b) Group convolution in STCB is set to depthwise convolution

(c) ATCN Architecture

Figure 4.4: Structure of ATCN blocks. The non-linearity activation and batch normalization units after each convolution are not depicted.

4.4.1.1 ATCN Structure

The first layer of ATCN is standard convolution with an optional MaxPooling for downsampling the input. A padding unit is also added before standard convolution and expansion convolution in STCB to ensure that the input and output tensors of the block have the same size to satisfy principle number 2 of GTCN. The $2p$ zeros are added symmetrically is added by padding unit, where p is given by:

$$p = \lceil \frac{(o-1) \times s + (k-1) \times (d-1) - i + k}{2} \rceil, \tag{4.3}$$

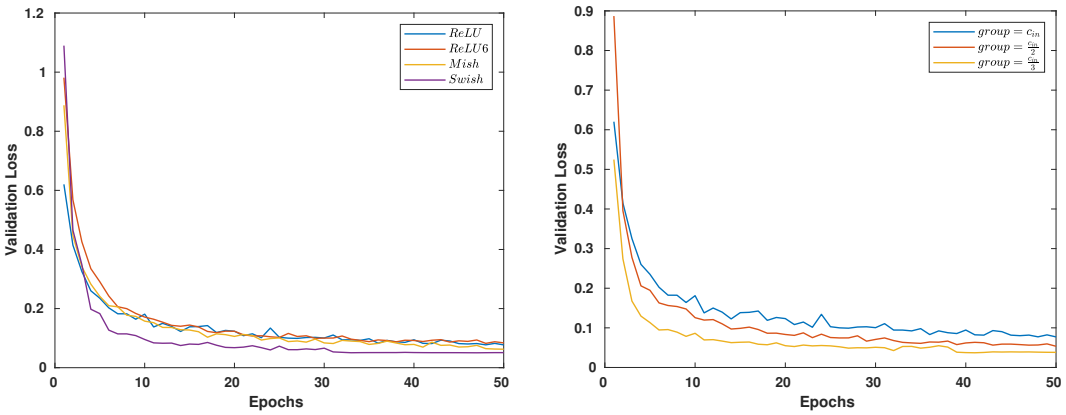
where o is the output size, i is the input size, s is the stride, k is the kernel, and d is the dilation. After each convlution, 1D batch normalization and a non-linear activation function is also added. In this research, we used $Swish$ as activation function:

$$Swish(X) = X \odot Sigmoid(X), \tag{4.4}$$

where $\odot$ is Hadamrd or elemnt-wise multiplication. Fig. 4.5(a) depicts the performance of different activation functions on MNIST validation loss.

The STCB consists of expansion, followed by a group and another projection convolution. The task of expansion convolution is to map input channel size, c_{in}, to higher or same dimension, c_{out}^{exp}, where $c_{out}^{exp} = \alpha \times c_{in}, \alpha \geq 1$. On the contrary, pointwise projection embeds and maps the feature extracted from the group conolution to the block output size, c_{out}. For the case of depthwise convolution, we set *group*, which manages the connection between input and output, to c_{out}^{exp}. For this case, the convolution weigh shape changes from (c_{out}, c_{in}, k) to $(c_{out}, 1, k)$, where k is the kernel size. We designed the network synthesizer so that if $c_{in} = c_{out}$, the skip line is automatically created from input to the elementwise addition. Then, the input will be added to the residual output from the pointwise projection. The residual connec-

tion helps the designers increase the network's depth without being worried about the vanishing gradient problem. The model synthesizer considers group convolution rather than depthwise for the case that MaxPooling is selected. The reason for doing so is based on this observation that for downsampling the input, which has an activated max-pooling unit, group convolution helps to better map temporal information to a higher dimension without drastically increasing computation complexity and the model size. The only constraint imposed by group convolution is that its output channel size, c_{out}^{gc}, should be divisible by c_{out}^{exp}. The two extreme G-CNN cases are when $group = c_{in}^{gc}$ and $group = 1$. In the former case, the group convolution is a depthwise convolution, and in the latter, it is a standard convolution. Formally, the weight shape for group convolution is $(c_{out}, \frac{c_{in}}{group}, k)$. We depict the effect of altering the *group* values in Fig. 4.5(b) for MNIST digit classification. As we can see, reducing the *group* value increases the network capacity to minimize the validation loss.



(a) Effect of different non-linearity activation function on validation loss.

(b) Dilated Causal Convolution.

Figure 4.5: The effect of different non-linearity activation function, σ, and *group* value on final validation loss.

4.4.2 ATCN hyper-parameters

For designing the ATCN network architecture, three knobs should be altered based on the problem complexity (sequence classification, prediction, or segmentation), the

input size, the network models, and computational cost trade-off. In the rest, we fully elaborate on each of them.

4.4.2.1 Dilation rate

For a fixed input size, if we increase the number of layers, based on the GTCN architecture guideline, we need to increase the dilation rate exponentially. This decision will help the network have a higher receptive field; however, based on principle number 2, we need to pad the features excessively to have the same input and output size. This unnecessary padding results in 1) more computation and 2) CNN performance degradation. We observed linear growth for dilation would help the network with more than six layers to have better feature representation. Although the dilation rate can be defined as a function of layer number, we increased it after each block with activated downsampling in the experimental results. This decision helps design a deep ATCN for the cases where input size, i, is small.

4.4.2.2 Kernel size

It is recommended that the kernel size, k, is large enough to encompass enough feature context based on the problem complexity. However, based on Eq. 5.5, it is a good practice to decrease the kernel size for higher layers to ensure p is not growing exponentially. By contrast, if we increase the dilation rate, based on Eq. 5.2, the kernel size can be reduced without concern for the receptive field. Embedded devices gain two crucial advantages from this decision: it reduces 1) the computational complexity and 2) the model size.

4.4.2.3 Number of layers

Similar to the dilation rate, if we need to increase the network's depth to increase its capacity, it is recommended to gradually decrease kernel size and have a linear growth for dilation. We can alter both after each block with downsampling units. This decision helps the final structure to have enough receptive field to cover feature

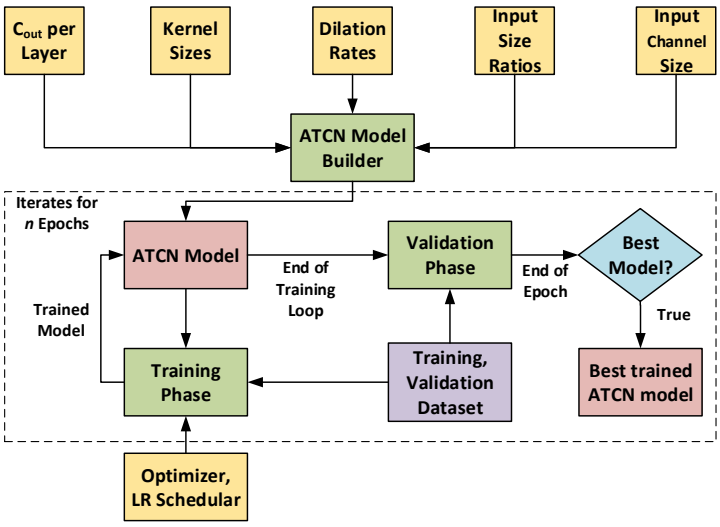


Figure 4.6: ATCN model synthesizer and training framework.

context without increasing the MAC operations and model sizes.

4.4.3 ATCN Model Synthesizer

We depict the inputs of *ATCN Model Synthesizer* and the framework for its training in Fig. 4.6. The ATCN Model Synthesizer receives Input Channels, Kernel Sizes, Dilation Rates, Input Ratios, and finally, the first *Input Channel Size* to design the ATCN network architecture. In the rest, we explain each of the ATCN Model Synthesizer inputs in detail.

c_{out} *per Layer*: It is a vector of size L, where L is the number of layers (blocks). The c_{out}^{l} defined in $\boldsymbol{C_{out}} = [c_{out}^{1}, c_{out}^{2}, ..., c_{out}^{L}]$, decides the output channel for layer l. For this research, the values in vector C_{out} are descending-ascending (Auto-Encoder architecture) to code the feature to lower dimension to extract temporal correlation and then maps to a higher dimension to represent extracted features for final stages.

Kernel Size: The vector $\boldsymbol{K}$, $\boldsymbol{K} = [k^1, k^2, ..., k^L] \mid k^l \in \mathbb{N}$, defines the kernel sizes for each layer. Based on the discussion of Section. 4.4.2.2, it is suited to decrease the k to minimize both model size and required computational complexity.

Dilation Rates: The vector $\boldsymbol{D}$, where $\boldsymbol{D} = [d^1, d^2, ..., d^L] \mid d^l \in \mathbb{N}$, defines the

dilation rates per each layer. On the contrary to K, it is necessary to increase d to achieve a higher or same receptive field at deeper levels.

Input Size Ratios: The $\boldsymbol{R} = [r^1, r^2, ..., r^L] \mid 0 < r^l \leq 1$, defines the input ratios. For the value of $r^l < 1, l > 1$, the *ATCN Model Synthesizer* configures the STCB block with max-pooling unit. For the case of $l = 1, r < 1$, the max-pooling will be added after standard convolution; otherwise, the input and out of standard convolution will have the same size. For this research, the r^l can only be defined as $\frac{1}{2}$ or 1. For other ratios, the synthesizer can be modified to change the stride of max-pooling to satisfy the targeted ratio.

4.4.4 ATCN Families

We introduce three ATCN families by assigning values to the $\boldsymbol{C}$, $\boldsymbol{K}$, $\boldsymbol{D}$, and $\boldsymbol{R}$ for UCR time series classification. Table 4.1 summarizes the configuration of T0, T1, and T2 as three canidates. Table 4.2 also compares their average FLOPS and number of model parameters of candidates and InceptionTime based on seventy benchmarks from the 2018 UCR time series classification, which are explained in detail in Section 4.5. We assign the output channels, C_{out}, in descending-ascending format to encode features to lower dimensions, then decode them to higher dimensions to represent them for the final dense layers and classifier. We have reduced the kernel size K due to the increased dilation rate, D. As a result, MACs and model parameters are reduced without compromising receptive field size. The T0 configuration reduces the MACs and model size by 102.38× and 16.84× over IT, respectively. T1 has also 73.59× reduction in MACs, and a 14.23× reduction in model size over IT. Finally, T2 reduces the MACs and model size for 26.07× and 4.4× over IT. The algorithmic accuracy of these models and training methods of these models are explained in 4.5.

Table 4.1: The configuration of three ATCN families

Models	Configurations			
	C_{out}	D	K	R
T0	$[32, 16, 16, 8, 8, 16, 16, 32]$	$[1, 2, 2, 4, 4, 6, 6, 8]$	$[32, 16, 16, 8, 8, 4, 4, 2]$	$[\frac{1}{2}, 1, 1, 1, 1, 1, 1, 1]$
T1	$[32, 16, 16, 8, 8, 16, 16, 32]$	$[1, 2, 2, 4, 4, 6, 6, 8]$	$[64, 32, 32, 16, 16, 8, 8, 4]$	$[\frac{1}{2}, 1, 1, 1, 1, 1, 1, 1]$
T2	$[64, 32, 32, 16, 16, 32, 32, 64]$	$[1, 2, 2, 4, 4, 6, 6, 8]$	$[64, 32, 32, 16, 16, 8, 8, 4]$	$[\frac{1}{2}, 1, 1, 1, 1, 1, 1, 1]$

Table 4.2: FLOPS and number of parameters for T0, T1, T2, and InseptionTime

Metric	Models			
	T0	**T1**	**T3**	**InceptionTime**
FLOPs	2,377,840	3,329,008	9,457,376	240,430,566
Params#	24,816	29,424	95,456	422,498

4.5 Experimental Results

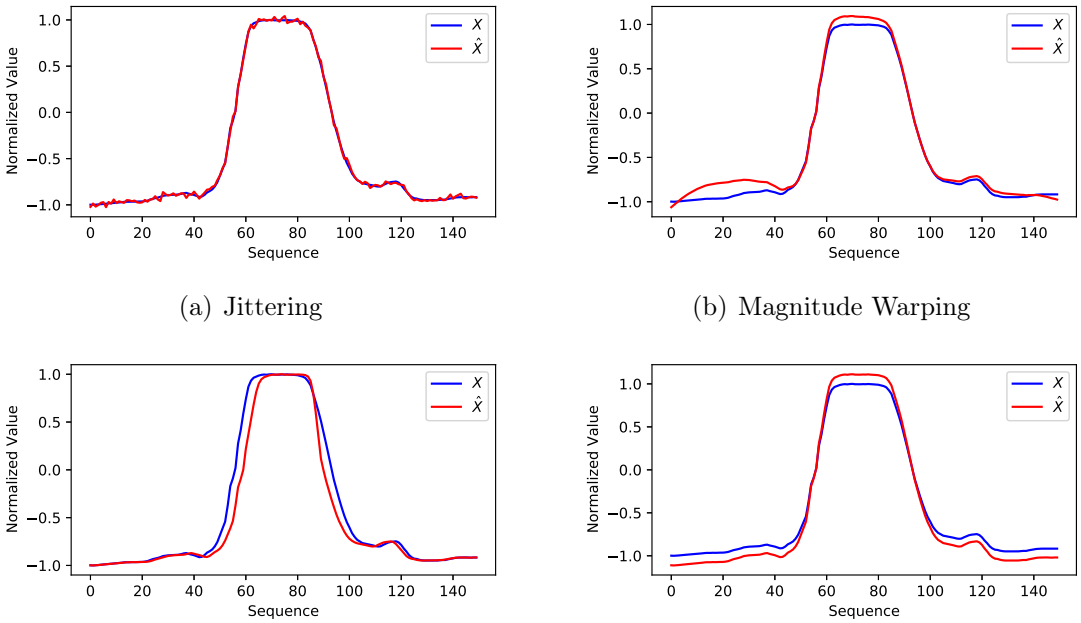
In this section, we demonstrate the capabilities of three different ATCN families by applying them to problems of UCR time series classification. T0 and T1 are compiled and utilized on ARM Cortex-M7 microcontroller series, and T2 is executed on ARM Cortex-A57. A report on RAM utilization, flash usage, and inference time is also provided.

4.5.1 Dataset

The experiments were conducted on 70 benchmarks publicly available from UCR Time Series Classification 2018, which vary in time length, number of classes, dataset type, sample size, and sample size. Table 1 summarizes the details of the benchmarks.

4.5.1.1 Data augmentation

For benchmarks whose training size is small, such as ECGFiveDays, we applied four types of data augmentation: jittering, magnitude warping [30], window warping [31], and scaling. In Fig. 4.7, each approach is shown in relation to the observed signal, X.



(a) Jittering

(b) Magnitude Warping

(c) Window Wraping

(d) Scaling

Figure 4.7: Different data augmentation applied on UCR dataset. X is observed signal and $\hat{X}$ is the augmented data.

Table 4.3: Description of 70 benchmarks selected from UCR Time Series Classification Archive 2018

ID	Type	Name	Train	Test	Class	Length
1	ECG	ECGFiveDays	23	861	2	136
2	Sensor	Plane	105	105	7	144
3	Simulated	TwoPatterns	1000	4000	4	128
4	Sensor	Trace	100	100	4	275
5	Power	PowerCons	180	180	2	144
6	Spectro	Coffee	28	28	2	286
7	Simulated	BME	30	150	3	128
8	Motion	GunPointMaleVersusFemale	135	316	2	150
9	Motion	GunPointOldVersusYoung	136	315	2	150

Continued on next page

Table 4.3 – *Continued from previous page*

ID	Type	Name	Train	Test	Class	Length
10	EPG	InsectEPGRegularTrain	62	249	3	601
11	Sensor	Wafer	1000	6164	2	152
12	Simulated	SyntheticControl	300	300	6	60
13	Sensor	FreezerRegularTrain	150	2850	2	301
14	Traffic	Chinatown	20	343	2	24
15	Simulated	ShapeletSim	20	180	2	500
16	Motion	GunPointAgeSpan	135	316	2	150
17	ECG	TwoLeadECG	23	1139	2	82
18	Motion	GunPoint	50	150	2	150
19	Simulated	UMD	36	144	3	150
20	Sensor	DodgerLoopWeekend	20	138	2	288
21	Device	HouseTwenty	40	119	2	2000
22	Motion	ToeSegmentation2	36	130	2	343
23	Sensor	ItalyPowerDemand	67	1029	2	24
24	Simulated	CBF	30	900	3	128
25	Image	Symbols	25	995	6	398
26	Sensor	FordA	3601	1320	2	500
27	Image	Fish	175	175	7	463
28	Image	DiatomSizeReduction	16	306	4	345
29	ECG	ECG5000	500	4500	5	140
30	ECG	ECG200	100	100	2	96
31	Motion	ToeSegmentation1	40	228	2	277
32	Spectro	Strawberry	613	370	2	235
33	Sensor	SonyAIBORobotSurface1	20	601	2	70
34	Sensor	FreezerSmallTrain	28	2850	2	301

Continued on next page

Table 4.3 – *Continued from previous page*

ID	Type	Name	Train	Test	Class	Length
35	Image	MixedShapesRegularTrain	500	2425	5	1024
36	Sensor	DodgerLoopGame	20	138	2	288
37	Spectrum	SemgHandGenderCh2	300	600	2	1500
38	Image	BirdChicken	20	20	2	512
39	Sensor	SonyAIBORobotSurface2	27	953	2	65
40	Sensor	Lightning2	60	61	2	637
41	Image	ProximalPhalanxOutlineCorrect	600	291	2	80
42	Image	ProximalPhalanxOutlineAgeGroup	400	205	3	80
43	Device	LargeKitchenAppliances	375	375	3	720
44	Sensor	Car	60	60	4	577
45	Sensor	MoteStrain	20	1252	2	84
46	Sensor	FordB	3636	810	2	500
47	Image	ArrowHead	36	175	3	251
48	Image	BeetleFly	20	20	2	512
49	Image	ProximalPhalanxTW	400	205	6	80
50	Device	SmallKitchenAppliances	375	375	3	720
51	Image	FaceAll	560	1690	14	131
52	Motion	UWaveGestureLibraryX	896	3582	8	315
53	Spectrum	SemgHandSubjectCh2	450	450	5	1500
54	Image	DistalPhalanxOutlineAgeGroup	400	139	3	80
55	Sensor	Earthquakes	322	139	2	512
56	Motion	WormsTwoClass	181	77	2	900
57	Sensor	Lightning7	70	73	7	319
58	Spectro	Ham	109	105	2	431
59	Image	DistalPhalanxTW	400	139	6	80

Continued on next page

Table 4.3 – *Continued from previous page*

ID	Type	Name	Train	Test	Class	Length
60	Motion	UWaveGestureLibraryZ	896	3582	8	315
61	Device	Computers	250	250	2	720
62	Device	ElectricDevices	8926	7711	7	96
63	Motion	UWaveGestureLibraryY	896	3582	8	315
64	Sensor	InsectWingbeatSound	220	1980	11	256
65	Image	MiddlePhalanxOutlineAgeGroup	400	154	3	80
66	Image	Herring	64	64	2	512
67	Image	MiddlePhalanxTW	399	154	6	80
68	EOG	EOGVerticalSignal	362	362	12	1250
69	Sensor	DodgerLoopDay	78	80	7	288
70	Device	RefrigerationDevices	375	375	3	720

4.5.2 Implementation details

The models are implemented in PyTorch and trained on a Nvidia Tesla V100 GPU using the ADAM optimizer with a Learning Rate (LR) of 0.001, a gradient clip of 0.25, and a weight decay of 0.001. We also reduce the LR by the factor of 0.1 when the validation loss stagnates for eight epochs. In the case of datasets with two classes, Binary Cross-Entropy (BCE) loss function is used, and Cross-Entropy is used for all other datasets.

4.5.3 Algorithmic Comparison

Table 4.4 compares the algorithmic performance of three ATCN families to that of InceptionTime. A comparison of Tables 4.4 and 4.2 shows that T1 has an increase in performance of 4.03% over T0 and has a reduction of 73.59× in MACs and a decrease in model size of 14.23 times over IT. In addition to reducing MACs and model sizes

by 26.07 times and 4.4 times over IT, the T2 also boosts accuracy by 0.37 percent.

Table 4.4: Comparison of the average accuracy for seventy benchmarks from the 2018 UCR time series classification dataset

Metric	**Models**			
	T0	**T1**	**T2**	**InceptionTime**
Average Accuracy	83.69%	85.73%	86.65%	86.28%

In accordance with Demvsar's recommendation [32], the Friedman test is practiced and the results shows that the four classifiers are not statistically different from each other. In Fig. 4.8, we show the critical difference diagram of discussed classifiers. The connected classifiers by a thick line indicate that they do not have a significant difference stattically at the p-value of 0.05; however, InceptionTime is undoubtedly plagued by higher computational complexity in respect to ATCN families.

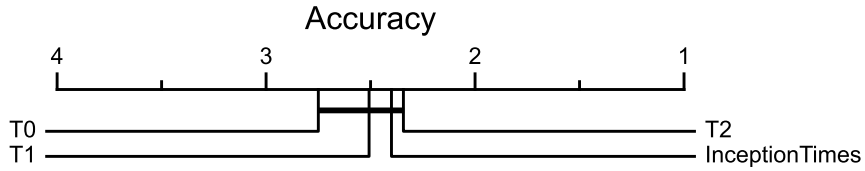


Figure 4.8: Critical difference diagram shwoing the performance of four classifier. The diagram depicts the overall average ranking of the classifiers, where those connected by a thick line show no statistically significant inconsistencies at p-value 0.05. As a result, T0, T1, T2, and InceptionTime are not significantly different.

4.5.4 Execution Comparison

In order to evaluate the performance of models on embedded devices, we selected `STM32F746ZGT6`, which has Cortex-M7 running at 216 MHz with 320 KB of RAM and 1 MB of flash memory as a microcontroller, and Cortex-A57 running at 1.43 GHz as an embedded microprocessor. We compared the execution performance and hardware utilization of all four classifiers in Table 4.5. The results are extracted by running model trained on *Coffee* benchmark. Compared to the T1 configuration, the T0 can reduce both RAM and flash utilization by 7.5% and 1.47%, respectively,

while improving inference time by 21.42%. Due to their higher RAM requirements, T2 and InceptionTime could not be compiled. In next, we ran the all four ONNX models on A57 processor for the batch size of one. Based on the comparison between the ATCNs and InseptionTime, the ATCN families are faster by 7.49×, 5.87×, and 2.93×, respectively.

Table 4.5: Resource utilization and inference time of two Cortex-M7 and Cortex-A57 platforms

Parameters	**Models**			
	T0	**T1**	**T2**	**InceptionTime**
M7 RAM utilization	48.89%	56.39%	-	-
M7 flash utilization	14.13%	15.89%	-	-
M7 inference time	165 mS	210 mS	-	-
A57 inference time	2.81 mS	3.58 mS	7.16 mS	21.05 mS

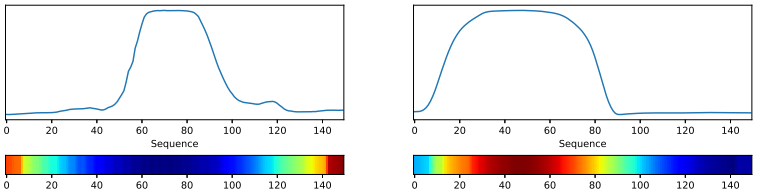
4.5.5 Architectural configuration study

During our study of the effects of kernel size, we developed a new configuration called T_β. Table 4.6 summarizes the T_β network configuration, FLOPs, the number of parameters, and its average accuracy. It can be seen that T_β has the same $\boldsymbol{C_{out}}$, $\boldsymbol{D}$, and $\boldsymbol{R}$ as T2, but the kernel size per block has been halved. Despite a greater model complexity than T1, both FLOPs and model size, the results show that T1 is still more accurate than T_β. As an aid to understanding this behavior, we depict the inputs and Class Activation Mapping (CAM) of two benchmarks in Fig. 4.9 and Fig. 4.10. When the model correctly classified the input signal, CAMs are calculated by multiplying the input of global average pooling by the weight matrices of the correct class index.

Table 4.6: Model configuration and accuracy performance of T_β

Model	**Parameters**						
	$\boldsymbol{C_{out}}$	$\boldsymbol{D}$	$\boldsymbol{K}$	$\boldsymbol{R}$	**FLOPs**	**Params#**	**Average Accuracy**
T_β	$[64, 32, 32, 16, 16, 32, 32, 64]$	$[1, 2, 2, 4, 4, 6, 6, 8]$	$[32, 16, 16, 8, 8, 4, 4, 2]$	$[\frac{1}{2}, 1, 1, 1, 1, 1, 1, 1]$	7,303,136	86,240	84.94%

The activation heatmaps for class 0, shown in Fig. 4.9 , have the lowest value around the input signal magnitude. Consequently, for class 0, the probability of



(a) Input and T0 CAM output for class 0

(b) Input and T0 CAM output for class 1

(c) T1 CAM output for class 0

(d) T1 CAM output for class 1

(e) T2 CAM output for class 0

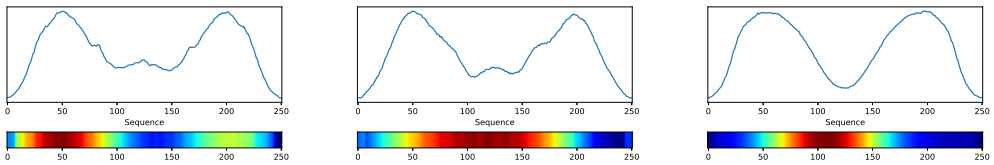
(f) T2 CAM output for class 1

(g) T_β CAM output for class 0

(h) T_β CAM output for class 1

Figure 4.9: GunPointOldVersusYoung dataset

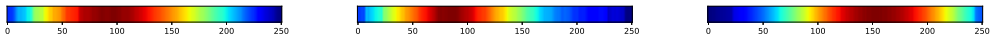
output approaches zero. However, we can observe the activation heatmaps for class 1 have the highest value around the signal magnitude, which leads to the output probability being one. For multiclass classification problems depicted in Fig. 4.10, we can see models activate based on the perceived nuances of signal shapes. For instance, the T0 model, Fig. 4.10(a) $\sim$ Fig. 4.10(a), classifies the input as class 0 based on the form observed in sequence $\sim$10 to $\sim$80, as class 1 based on unique transition observed in the middle of the sequence, and as class 2 based on the curve recognized in $\sim$70 to $\sim$150. In respect to T_β, this model shows a coarse-grained transition, note sequence $\sim$20 to $\sim$60 in Fig. 4.9(h) and $\sim$150 to $\sim$240 in Fig. 4.10(j). This indicates that T_β has a lower receptive field compared to T1 and T2, both of which have the same kernel size. As a result of the higher receptive field of T1, the model is able to predict the classes more precisely, although it has less model complexity in both forms of FLOPS and the number of parameters.



(a) Input and T0 CAM output for class 0

(b) Input and T0 CAM output for class 1

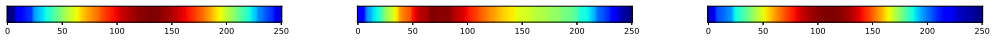
(c) Input and T0 CAM output for class 2



(d) T1 CAM output for class 0

(e) T1 CAM output for class 1

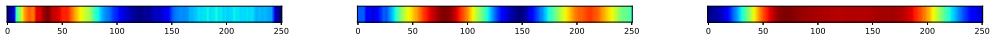
(f) T1 CAM output for class 2



(g) T2 CAM output for class 0

(h) T2 CAM output for class 1

(i) T2 CAM output for class 2



(j) T_β CAM output for class 0

(k) T_β CAM output for class 1

(l) T_β CAM output for class 2

Figure 4.10: ArrowHead dataset

4.6 Conclusion

ATCN is proposed here for lightweight real-time processing of time-series on embedded and edge devices. In order to reduce the number of MAC operations and model size, we introduced STCB as a main computational block. STCB blocks are able to be sequenced in a variety of configurations to build scalable ATCNs. We also presented a framework, called *ATCN Model Synthesizer*, to build different ATCN models. The result of *ATCN Model Synthesizer* is a family of compact networks with formalized hyper-parameters that allow the model architecture to be configurable and adjusted based on the application requirements. Through the use of model synthesizer and ATCN reconfigurability, we have developed three agile models, T0, T1, and T2, two of which can be executed on ARM Cortex-M7 microcontrollers. The experimental results over 2018 UCR time classification benchmarks indicate that the T0 configuration can reduce the MACs and model size by 102.38× and 16.84× over InceptionTime, respectively. T1 performance is 4.03% better than T0 and has a 73.59× reduction in MACs, and a 14.23× reduction in model size over InceptionTime. Both

T0 and T1 can be executed on an ARM Cortex-M7 microcontroller explained in the experimental section in detail. As a final improvement, T3 reduces the MACs and model size for 26.07× and 4.4× over IT while increasing accuracy by 0.37%.

Bibliography

[1] C. Neff, M. Mendieta, S. Mohan, M. Baharani, S. Rogers, and H. Tabkhi, "Revamp²t: Real-time edge video analytics for multicamera privacy-aware pedestrian tracking," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 2591–2602, 2020.

[2] B. Blanco-Filgueira, D. García-Lesta, M. Fernández-Sanjurjo, V. M. Brea, and P. López, "Deep learning-based multiple object visual tracking on embedded system for iot and mobile edge computing applications," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5423–5431, 2019.

[3] M. Xu, M. Gao, Y. Chen, L. Davis, and D. Crandall, "Temporal recurrent networks for online action detection," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 5531–5540.

[4] G. Chen, C. Zhang, and Y. Zou, "Afnet: Temporal locality-aware network with dual structure for accurate and fast action detection," *IEEE Transactions on Multimedia*, pp. 1–1, 2020.

[5] S. D. Goodfellow, A. Goodwin, R. Greer, P. C. Laussen, M. Mazwi, and D. Eytan, "Towards understanding ecg rhythm classification using convolutional neural networks and attention mappings," ser. Proceedings of Machine Learning Research, F. Doshi-Velez, J. Fackler, K. Jung, D. Kale, R. Ranganath, B. Wallace, and J. Wiens, Eds., vol. 85. Palo Alto, California: PMLR, 17–18 Aug 2018, pp. 83–101. [Online]. Available: http://proceedings.mlr.press/v85/goodfellow18a.html

[6] S. Saadatnejad, M. Oveisi, and M. Hashemi, "Lstm-based ecg classification for continuous monitoring on personal wearable devices," *IEEE Journal of Biomedical and Health Informatics*, vol. 24, no. 2, pp. 515–523, 2020.

[7] Y. Li, Z. Xia, and Y. Zhang, "Standalone systolic profile detection of non-contact scg signal with lstm network," *IEEE Sensors Journal*, vol. 20, no. 6, pp. 3123–3131, 2020.

[8] M. Baharani, M. Biglarbegian, B. Parkhideh, and H. Tabkhi, "Real-time deep learning at the edge for scalable reliability modeling of si-mosfet power electronics converters," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7375–7385, 2019.

[9] M. Biglarbegian, M. Baharani, N. Kim, H. Tabkhi, and B. Parkhideh, "Scalable reliability monitoring of gan power converter through recurrent neural networks," in *2018 IEEE Energy Conversion Congress and Exposition (ECCE)*, 2018, pp. 7271–7277.

[10] Y. Zhang, R. Xiong, H. He, and M. G. Pecht, "Long short-term memory recurrent neural network for remaining useful life prediction of lithium-ion batteries," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 7, pp. 5695–5705, 2018.

[11] B. Zhang, D. Xiong, J. Xie, and J. Su, "Neural machine translation with gru-gated attention model," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–11, 2020.

[12] Q. Li, X. Zhang, J. Xiong, W.-m. Hwu, and D. Chen, "Implementing neural machine translation with bi-directional gru and attention mechanism on fpgas using hls," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 693–698. [Online]. Available: https://doi.org/10.1145/3287624.3287717

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, Nov. 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[14] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014.

[15] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *CoRR*, vol. abs/1803.01271, 2018. [Online]. Available: http://arxiv.org/abs/1803.01271

[16] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," 2016.

[17] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 3, pp. 328–339, 1989.

[18] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," 2018.

[19] H. Ismail Fawaz, B. Lucas, G. Forestier, C. Pelletier, D. F. Schmidt, J. Weber, G. I. Webb, L. Idoumghar, P.-A. Muller, and F. Petitjean, "Inceptiontime: Finding alexnet for time series classification," *Data Mining and Knowledge Discovery*, vol. 34, no. 6, pp. 1936–1962, Nov 2020. [Online]. Available: https://doi.org/10.1007/s10618-020-00710-y

[20] H. A. Dau, E. Keogh, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, Yanping, B. Hu, N. Begum, A. Bagnall, A. Mueen, G. Batista, and Hexagon-ML, "The ucr time series classification archive," October 2018, https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

[21] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *ArXiv*, vol. abs/1609.03499, 2016.

[22] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks for action segmentation and detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1003–1012.

[23] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," *CoRR*, vol. abs/1705.03122, 2017. [Online]. Available: http://arxiv.org/abs/1705.03122

[24] A. Pandey and D. Wang, "Tcnn: Temporal convolutional neural network for real-time speech enhancement in the time domain," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 6875–6879.

[25] R. Sen, H.-F. Yu, and I. S. Dhillon, "Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting," in *Advances in Neural Information Processing Systems*, 2019, pp. 4837–4846.

[26] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI'17. AAAI Press, 2017, p. 4278–4284.

[27] J. Lines, S. Taylor, and A. Bagnall, "Hive-cote: The hierarchical vote collective of transformation-based ensembles for time series classification," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, 2016, pp. 1041–1046.

[28] M. Carreras, G. Deriu, L. Raffo, L. Benini, and P. Meloni, "Optimizing temporal convolutional network inference on fpga-based accelerators," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, pp. 1–1, 2020.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for

image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[30] T. T. Um, F. M. J. Pfister, D. Pichler, S. Endo, M. Lang, S. Hirche, U. Fietzek, and D. Kulić, "Data augmentation of wearable sensor data for parkinson's disease monitoring using convolutional neural networks," in *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, ser. ICMI '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 216–220. [Online]. Available: https://doi.org/10.1145/3136755.3136817

[31] A. Le Guennec, S. Malinowski, and R. Tavenard, "Data augmentation for time series classification using convolutional neural networks," in *ECML/PKDD workshop on advanced analytics and learning on temporal data*, 2016.

[32] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

CHAPTER 5: DEEPTRACK

5.1 Introduction

With the advent of high-speed communication systems and unprecedented improvements in trajectory predicting, we are closer to implementing a fully connected (vehicle-to-vehicle (V2V) and Vehicle-to-Infrastructure (V2I)) and fully-aware transportation system than ever before. The increasing push towards autonomous driving and thrust for designing the best in class crash-avoidance systems at the edge has resulted in developing trajectory forecasting algorithms with better than 90% accuracy for up to 5 seconds in the future. The use of such high accuracy models in safety-critical systems for crash avoidance and accident prediction can result in precise Time-to-Collision (TTC) prediction, which can prove instrumental in avoiding accident-related injuries and saving many lives.

In 2019, there were 36,096 fatalities on roadways in the United States [1, 2]. Of those fatal crashes, NHTSA (2019) estimates that 11.9% of them involved a vehicle maneuvering in a manner that may be unpredictable to the other drivers (i.e., turning left or right, stopping or slowing in traffic, merging/changing lanes, or passing another vehicle). Such crashes at highway speeds, given the short TTC and limited distance range, cannot be prevented with vision-based systems alone [3]. Providing enough time and distance to support effective crash avoidance via V2V systems must also utilize high-accuracy predictions for changing vehicle trajectories.

Predicting multiple possible trajectories for an active subject in the scene [4, 5] is a common practice. These trajectories are ranked based on the probability distribution of the prediction model, which may not be helpful in a real-time scenario. Hence, a deep learning algorithm for vehicle trajectory forecasting in a real-time safety-critical

application must provide a single trajectory with high precision. It is imperative to consider the interactions of the surrounding automobiles [4, 6] to successfully predict the accurate path of a moving vehicle in a highly dynamic environment. However, designing a real-time system at the edge for predicting safety-critical situations, the size and complexity of the model must also be considered. Smaller size and lower complexity result in faster predictions which can prove to be crucial in accident avoidance.

This article proposes DeepTrack as a novel deep learning model with comparable accuracy to best-in-class trajectory prediction algorithms but a much smaller model size with lower computational complexity to suit the resource-crunched embedded edge systems. DeepTrack encodes the vehicle dynamics with the aid of Agile Temporal Convolutional Networks (ATCN) instead of well-established LSTM units. ATCN, with its depthwise convolution as its backbone, can shrink the complexity of models and boost gradient flow for a more generalized trained model compared to LSTM-based solutions. Compared to CS-LSTM and its other variant CS-LSTM (M) [5], DeepTrack reduces prediction error by 9.09% and 11.56%, respectively, and reduces the number of operations and model size by about 10.49% and 18.5%, respectively. Concerning CF-LSTM [7], DeepTrack error raised for 0.87%; however, it can reduce the number of operations and model size by 10.49% and model size by 18.5%.

This article is organized as follows: Section 5.2 reviews the literature. Section 5.3 presents DeepTrack and introduces ATCN as an alternative to LSTMs used as dynamic encoders of vehicle trajectories to reduce model complexity. On both the accuracy and complexity of DeepTrack's model, we compare the results to those of state-of-the-art solutions in Section 5.4. Finally, the chapter concludes in Section 5.5.

5.2 Related Work

Vehicle trajectory prediction networks are classified into three types, Physical-based, Maneuver-based, and Interaction-aware models [8]. The physical-based models are designed using the laws of physics, maneuver-based models consider the driver

intentions, and interaction-aware models take surrounding vehicles and their interactions into account for motion and path prediction. Although interaction-aware models predict with a better understanding of the surrounding, they were not very popular until recently. Most of the interaction conscious networks used to be Dynamic Bayesian [9, 10] or prototype trajectory models [11]. However, several models have been introduced more recently, taking advantage of surrounding information for trajectory prediction. Many architectures use Long short-term memory (LSTM) neural networks [12, 13, 14] to capture the information of the neighboring vehicles. In the famous work by Deo *et al.* [4], the encoded information for each vehicle is condensed into a single tensor. Convolutional and max-pooling layers follow it to avoid generalization and failure to address complicated scenarios. This information is concatenated with LSTM encoding of the target vehicle and passed to a Maneuver-based LSTM decoder that forecasts multiple trajectories based on maneuver classes discussed in [4].

In [15], Mercat *et al.* also use LSTM based encoder-decoder architecture but avoid using predefined maneuver classes. It has self-attention layers in the middle that accept the encoded information of each vehicle for specific time instances. This helps in generating a fixed-sized input even when the number of vehicles in the scene might change. The middle layer produces an attention matrix based on the features extracted from the encoder. Finally, the decoder predicts path probabilities for each vehicle in the scene. The performance of LSTM based model in [15] is better than most of the best in class trajectory predicting algorithms. However, graph neural networks (GNN) with unique learning ability and have been able to produce higher accuracy results when applied to path prediction [7, 16].

The work by Xie. *et al.* [7] proposes a GNN [17] based teacher-student model that predicts higher accuracy trajectories than the previous models. The teacher model accepts frame-wise graph input built to reflect the positions of all the agents in the

input frame. It also uses Graph Convolutional network-based encoder-decoder for generative learning and Gaussian mixture model for congestion pattern generation. The student model uses LSTM based encoder-decoder for trajectory prediction and matches the congestion pattern of the teacher model to improve the accuracy of prediction.

The improving error rates for deep learning algorithms predicting vehicle trajectory are routinely focused on in most published research. We have successfully attained error rates comparable to state-of-the-art algorithms with lower model complexity and smaller model size. We use ATCN based novel encoder to grasp the positions of the vehicles in the scene for the past three seconds as compared to LSTM encoders in modern models [7, 15]. The output of encoders is used to condense the details of vehicular interaction in the past using 2D convolution. This information is passed to the LSTM Trajectory predictor that predicts a vicinity-aware trajectory for the target vehicle for five seconds in the future. We used US Highway 101 [18], and Interstate 80 Freeway [19] datasets provided by the Federal Highway Administration (FHWA) under Next Generation Simulation (NGSIM) program for training and validating the model.

5.3 DeepTrack

We first briefly explain ATCN in this section, and then we formulate the problem and explain the DeepTrack model.

5.3.1 Agile Temporal Convolutional Networks (ATCN)

Traditional Convolutional Neural Networks (CNN) are used in computer vision applications due to their success in capturing the spatial features within a two-dimensional frame. Recently, research has shown that specialized CNNs are capable of recognizing patterns in data history in order to predict future observations. This gives researchers interested in time-series forecasting options to choose from over

RNNs, which have been regarded in the community as the established DNN for time-series predictions. In one such case, TCN have been shown to achieve state of the art accuracy in sequence tasks, i.e. polyphonic music modeling, word and character-level language modeling, and audio synthesis [20, 21, 22, 23, 24]. TCNs are designed around two basic principles: 1) the convolutional operations are causal, i.e., predictions are made based only on current and past information; 2) the network receives an input sequence of arbitrary length and maps it to an output sequence of the same length [24]. Currently, these simple causal convolutions have a dilation rate of 1, but other researchers incorporate dilated convolutions, shown in Fig. 5.1, to scale the receptive field exponentially. The dilated convolution of F on element s of a sequence X is given as:

$$F(s) = (x *_d f)(s) = \sum_{i=0}^{k-1} f(i) \cdot x_{s-d \cdot i}, \tag{5.1}$$

where $X \in \mathbb{R}^n$ is a 1-D input sequence, $*_d$ is dilated convolution operator, $f : \{0, ..., k-1\} \in \mathbb{R}$ is a kernel of size k and d is the dilation rate [24]. Also receptive filed of a dilated convolution can be calculated by:

$$rf = 1 + \sum_{j=1}^{L} [k(l) - 1] \times d(l), \tag{5.2}$$

where $j \in \{1, 2, 3, ..., L\}$ is the layers, k is the kernel size, and $d(j)$ is the dilation rate at layer j. This means that as the depth of the network increase, so does the receptive field.

Inspired by TCN, we propose ATCN to reduce model complexity and memory footprint. Fig. 5.2 shows the structure of ATCN. The model shown in Fig. 5.2 has three hidden layers indicated by H. The hidden layers have a padding block to extend input feature to ensure that the input and output of the convolution have the same length to satisfy principle number 2. In H0, we use the normal convolution operator,

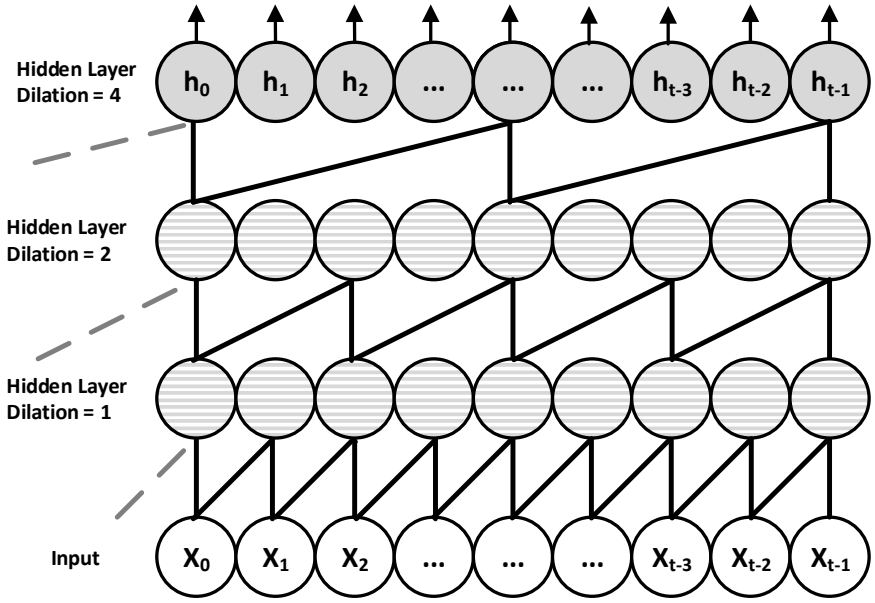


Figure 5.1: Dilated Causal Convolution.

but in the subsequent layers (H1-H2), we remove the standard convolution and use pointwise (PW), depthwise (DW), and then another pointwise to reduce the model size and number of operations. In this research, we also added a Batch Normalization (BN) layer after each convolution to speed up and stabilize the model training. There is also a Swish as an activation function after each BN. To our best of the knowledge, this is the first time a vehicle trajectory model based on ATCN is presented. In the section 5.4, we show off the effect of ATCN network on the performance of DeepTrack both on accuracy and model complexity.

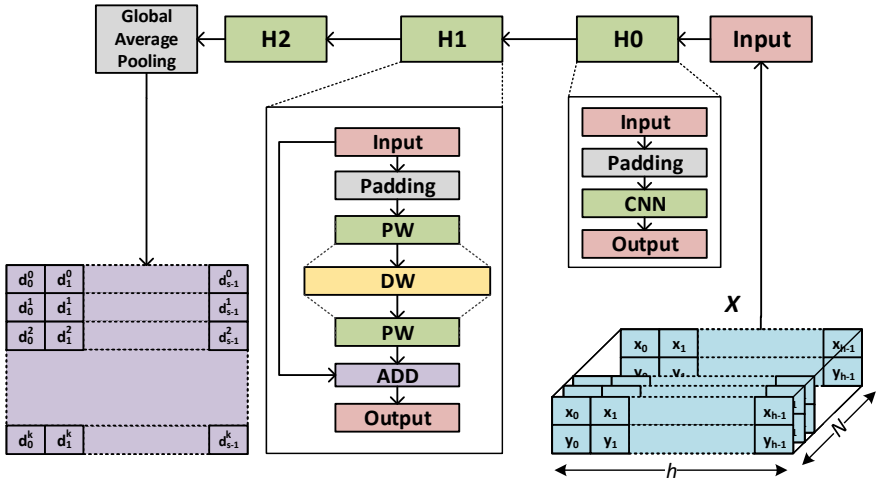


Figure 5.2: Structure of Agile Temporal Convolutional Networks (ATCN).

5.3.2 Problem formulation

The input of DeepTrack is define as:

$$\boldsymbol{X} = \begin{bmatrix} l_0^{t-h} & l_0^{t_{1-h}} & \cdots & l_0^{t_0} \\ \vdots & \vdots & \ddots & \vdots \\ l_e^{t-h} & l_e^{t_{1-h}} & \cdots & l_e^{t_0} \\ \vdots & \vdots & \ddots & \vdots \\ l_{N-1}^{t-h} & l_{N-1}^{t_{1-h}} & \cdots & l_{N-1}^{t_0} \end{bmatrix}, \tag{5.3}$$

where $l_i^t = \langle x_i^t, y_i^t \rangle$ is the position of vehicle i at time t, and N is the number of vehicles. A car of interest is designated by the index e in $\boldsymbol{X}$. The shape of $\boldsymbol{X}$ is shown in Fig. 5.2. To make fair comparisons, we construct $\boldsymbol{X}$ for vehicle e similarly to what Deo *et al.* explained in [4]. In a similar way, the output can be defined as follows:

$$\hat{\boldsymbol{Y}} = \left[l^{t_1}, l^{t_2}, \cdots, l^{t_f} \right]. \tag{5.4}$$

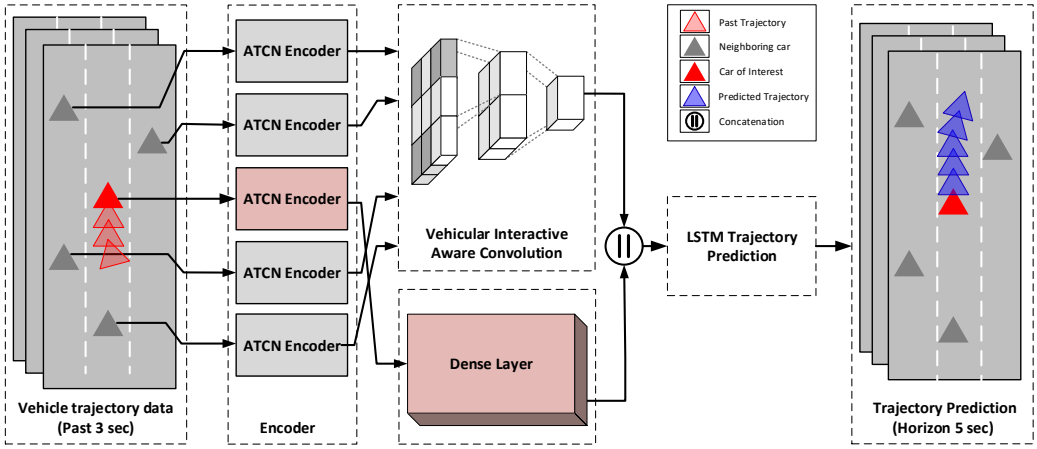


Figure 5.3: Overview of the trajectory prediction model. The location of the neighbors (gray triangles) and car of interest (solid red triangle) is shown at t_0 in Vehicle trajectory data (extreme left) block and Trajectory prediction (extreme right) block. Triangles denoting semi-transparent red in Vehicle trajectory data block, and semi-transparent blue in Trajectory prediction block represent observed history paths, and model output respectively. The observed history paths of neighbours (for past 3 seconds) are used by the model but not shown in the figure to avoid confusion.

5.3.3 Model architecture

An illustration of the DeepTrack model can be found in Fig. 5.3. The model consists ATCN encoder, Vehicular Interactive Aware Convolution (VIAC), and LSTM trajectory encoder. In the next section, we explain the structure of each component.

5.3.3.1 ATCN encoder

ATCN encoders embed vehicles' path histories, both neighbors and ego vehicle, into higher dimensions in order to capture their past trajectory. As opposed to previous works [4, 7], ATCN does not require dense layers to embed input features as needed for the LSTM encoder. ATCN convolutional operators capture and map the sequence of l^t by applying $K = [2, k]$ as a kernel, where $k \in \mathbb{R} | k \leq h$. As a result, DeepTrack has less model complexity, and better gradients flow from output to input during optimization.

DeepTrack uses two different ATCN encoders. There is one ATCN shared by all neighbors, shown by the shaded box in Fig. 5.3, which maps their dynamics to higher

dimensions so that the VIAC can comprehend their interdependencies. As illustrated by the red box in Fig. 5.3, the other ATCN maps only the ego dynamics.

Since we have a padding unit in each ATCN to make sure the input and ouput of standard and depthwise convolution will be same, the $2p$ zeros are added symmetrically, where p is given by:

$$p = \lceil \frac{(o-1) \times s + (k-1) \times (d-1) - i + k}{2} \rceil, \tag{5.5}$$

where o is the output size, i is the input size, s is the stride, k is the kernel, and d is the dilation. According to Eq. 5.5, if we increase the kernel size or dilation, more zeros should be padded to the input. The addition of excessive zeros to the input has two main disadvantages: ① it degrades the model's performance due to redundant zeros, and ② it increases the model computational complexity. As a result, we set the dilation rate and kernel size to 1, 2, respectively. In Table 5.1, both ATCN encoders have three hidden layers; however, the output dimensions differ.

Table 5.1: Configuration of ATCN encoder for ego and neighbors

ATCN Encoder	**Configurations (H0, H1, H2)**		
	Output feature dimensions	**Dilation rate**	**Kernel size**
Neigbours	[16, 32, 64]	[1, 1, 1]	[2, 2, 2]
Ego	[8, 16, 32]	[1, 1, 1]	[2, 2, 2]

5.3.3.2 VIAC

An analysis of the interaction between the ego and its surrounding neighbors is necessary to predict the future trajectory of the vehicle of interest. Despite being able to capture individual behavior, the ATCN encoder is unable to comprehend the entire scene. Social pooling [25] proposes a solution by pooling encoded data around a specific target. The task is accomplished by defining a spatially correlated grid $f \times g \times N$ in respect to the car of interest. Similar to the work of [4], we set f and g to 13 and 3, respectively. Fig.5.3 shows the structure of the grid for the shape of

3x3, which masked the ATCN encoded output.

DeepTrack comprehends the interdependencies of the vehicle by applying convolutions to embeds information. The neighborhood dynamics are encoded and mapped to the lower dimension using the two-layer convolution and a pooling unit. The dense layer also remaps the ATCN encoder output of ego to have the same feature size so that it can be concatenated with the result of VIAC.

5.3.4 LSTM decoder

In the final stage, the LSTM decoder receives the concatenated ego dynamics and neighbors' interdependencies to predict the future trajectory of the car of interest, $\hat{Y}$. We have not used ATCN as the final stage because the ATCN is the only cable to map the temporal information to a higher output channel. Similar to what is accomplished by the ATCN encoder at the first stage. Due to the mapping of all features to higher output channel dimensions, LSTM still needs to map and decode the output of the VIAC and ego ATCN encoders to the final output prediction.

5.4 Evaluation

5.4.1 Datasets

The performance of our model has been evaluated using NGSIM I-80 and US-101, two well-known, widely available vehicle trajectory datasets. A sampling rate of 10 Hz is used for sampling the trajectory of the vehicle over a period of 45 minutes. Each dataset includes three segments of 15 minutes long of mild, moderate, and congested traffic.

Similarly to [4], we divided the dataset into three parts: training, validation, and testing. This chapter reports the results of analyzing the test sets. Based on the work of [4], each trajectory is also segmented into 8 seconds, where the first three seconds are used as a path that was observed, and the model will predict the following five seconds. To reduce the complexity of the LSTM encoder, previous works

downsampled each second by two [4, 7, 15]. While we are not limited to this fact, we also downsampled the inputs to have a fair comparison. In Fig. 5.4(a)-5.5, the location of the neighbors (gray triangles) are shown at t_0. Triangles denoting red, green, and blue respectively, represent observed history paths, ground truth, and model output.

5.4.2 Implementation Details

The model is implemented in PyTorch and trained on an Nvidia Tesla V100 GPU using the ADAM optimizer with a Learning Rate (LR) of 0.001 with a gradient clip of 10. Additionally, the LR is reduced by an additional factor of 0.1 if the validation loss remains stagnant for two epochs. The total number of epochs was set to 10.

5.4.3 Metric

To provide a fair comparison to other works, we used an error metric called Root Mean Square Error (RMSE) to assess the overall performance of the system. The RMSE at time t is given by:

$$RMSE^t = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(Y_i^t - \hat{Y}_i^t\right)^2}, \tag{5.6}$$

where Y is the ground truth, $\hat{Y}$ is predicted output, and N is number of samples. We also use the Average Displacement Error (ADE) to compare the average RMSE over 5 seconds.

5.4.4 Quantitative Results

A comparison of DeepTrack against off-the-shelf algorithms on the NGSIM dataset was conducted for the purpose of providing a comprehensive comparison. The results are listed in Table 5.2. Compared to CS-LSTM and its other variant CS-LSTM (M), DeepTrack can reduce ADE by 9.09% and 11.56%, respectively. It is due to the fact that our ATCN has higher gradient stability that it is better able to generalize solu-

tions. However, both CF-LSTM and SAAMP outperformed the DeepTrack for 0.87% and 4.21%. SAAMP's attention mechanism allows it to gain a better understanding of certain interactions, thereby improving the overall RMSE.

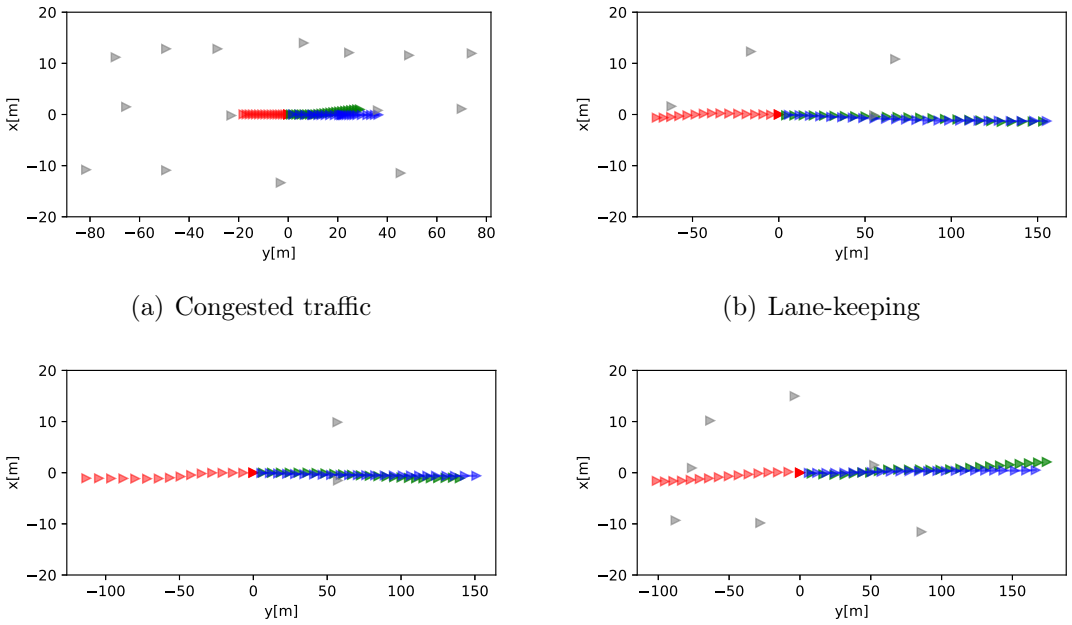
Considering that the final model will be deployed on the vehicle, model complexity is another important metric that previous researchers have overlooked. Therefore, we analyzed and compared the model complexity as measured by the number of Multiply-Accumulate (MAC) operations and the size of the model parameters for three approaches in Table 5.3. Since the source code for SAAMP was not available publicly, we could not compare its model complexity with DeepTrack. The smaller model size means a lower memory footprint on the final hardware. Low MACs also translates into better execution performance on the same hardware. As we can see, DeepTrack is able to improve the MACs and size of the model by 11.47% and 19.22%, respectively, over CF-LSTM. In addition to improving ADE over CS-LSTM, DeepTrack also improved MACs by 10.49% and model size by 18.5%.

Table 5.2: Performance comparison based on NGSIM dataset. RMSE is calculated in meters.

Model	**RMSE (m)**					**ADE**
	1 sec	**2 sec**	**3 sec**	**4 sec**	**5 sec**	
CS-LSTM [4]	0.61	1.27	2.09	3.1	4.37	2.29
CS-LSTM (M) [4]	0.62	1.29	2.13	3.2	4.52	2.35
CF-LSTM [7]	0.51	1.13	1.88	2.81	3.98	2.06
SAAMP [15]	0.55	1.1	1.78	2.73	3.82	2.00
DeepTrack	0.43	1.12	1.91	2.87	4.07	2.08

Table 5.3: Model size and number of flops per model

Parameters	**Models**		
	CF-LSTM [7]	**CS-LSTM [4]**	**DeepTrack**
MACs	2,064,282	2,045,938	1,667,425
Parameters	193,941	191,829	171,703



(a) Congested traffic

(b) Lane-keeping

(c) Maneuvering - Passing from left lane

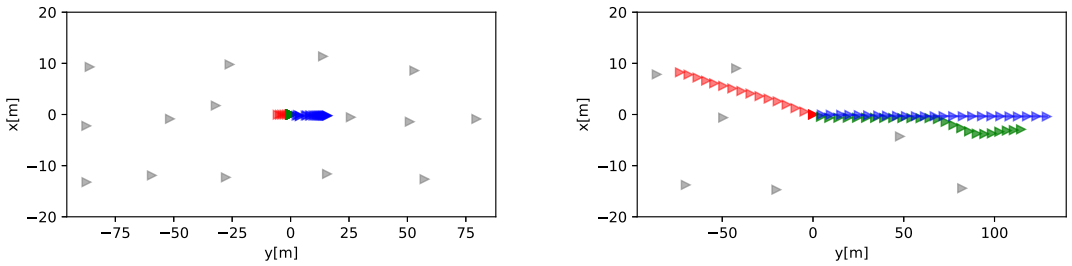
(d) Maneuvering - Passing from right lane

Figure 5.4: The location of the neighbors (gray triangles) is shown at t_0. Triangles denoting red, green, and blue respectively, represent observed history paths, ground truth, and model output.

5.4.5 Qualitative Results

The model predicted output for four scenarios is shown as an aid to understanding the model behaviour: ① congested traffic (Fig. 5.4(a)), ② lane-keeping (Fig. 5.4(b)), ③ maneuvering and passing a car from left lane (Fig. 5.4(c)), ③ maneuvering and passing a car from right lane (Fig. 5.4(d)), and ④ cases where the model failed to predict the trajectory precisely (Fig. 5.5). The three types of the path shown in red, green, and blue triangles represent path history, ground truth, and predicted trajectory for the designated vehicle. The location of the neighbors (gray triangles) is also shown at t_0. For the sake of simplicity, we didn't show the neighbors history path.

The comparison of Fig. 5.4(a) and Fig.5.4(b) shows that the model can accurately estimate the velocity of the interest car based on the ego history. The model correctly predicted that vehicles would travel less distance as a result of congested traffic. The



(a) Congested traffic

(b) Multiple lane changing

Figure 5.5: Two cases where the model failed to predict the trajectory precisely due to unpredictable driver behaviour. Same legend is used as Fig. 5.4.

vehicle in the lane-keeping scenario travels farther, and DeepTrack has interfered with the same behavior. Figures 5.4(c) and 5.4(c) illustrate how DeepTrack performs when a car of interest passes its front vehicle from either the left or the right lane. Fig. 5.5 shows the scenarios in which DeepTrack was not able to predict the trajectories due to uncertainty in driver behavior. In the congested scenario (Fig. 5.5(a)), although the driver slowly drove his car until t_0, the vehicle stopped for the entire next five seconds, while the model predicts it would come close to the front car. An alternative scenario (Fig. 5.5(b)) involves the vehicle being steered into the right lane. The model predicted that the vehicle would keep the lane and accelerate for the next five seconds; however, the driver has decided to change the lane another time. In this case, the model did not capture this behaviour since the maneuver occurred during the prediction horizon.

5.5 Conclusion and Future Works

DeepTrack is an agile deep learning model with comparable accuracy to best-in-class trajectory prediction algorithms but with much smaller model size and lower computational complexity suitable for embedded edge systems. The vehicle dynamics are encoded using ATCN instead of LSTM units in DeepTrack. ATCN utilizes depthwise convolution, thereby reducing the complexity of models both in terms of size and operations when compared with LSTMs. Our experimental results indic-

ate that DeepTrack reduces prediction error by 9.09% and 11.56%, respectively, and reduces the number of operations and model size by about 10.49% and 18.5%, respectively, to CS-LSTM. With similar ADE to CF-LSTM, DeepTrack can also reduce the number of operations and model size by 10.49% and model size by 18.5%.

We plan to work on two aspects as future works: First, we believe tagging geological meta information will improve the performance of DeepTrack on multiple lanes changing. If the vehicle is near an exit, it is more likely to change lanes repeatedly. Secondly, both CF-LSTM and CS-LSTM downsampled the input history to reduce the LSTM complexity and have better training performance. Due to DeepTrack's replacement of LSTM with ATCN, we have no bounds anymore, so we can take advantage of a larger sample size to improve DeepTrack's performance.

Bibliography

[1] (2019) Fatality Analysis Reporting System (FARS), Motor vehicle traffic crashes (1994-2019). [Online]. Available: https://www-fars.nhtsa.dot.gov/Main/index.aspx

[2] (2019) Fatality Analysis Reporting System (FARS), Vehicles involved in fatal crashes. [Online]. Available: VehiclesInvolvedinFatalCrashes,https://www-fars.nhtsa.dot.gov/Vehicles/VehiclesAllVehicles.aspx

[3] V. D. H. Richard and H. Jeroen, "Time-to-Collision and Collision avoidance systems," *International Co-operation on Theories and Concepts in Traffic safety (ICTCT)*, 1994.

[4] N. Deo and M. M. Trivedi, "Convolutional social pooling for vehicle trajectory prediction," in *2018 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, 2018, pp. 1468–1476.

[5] T. Phan-Minh, E. C. Grigore, F. A. Boulton, O. Beijbom, and E. M. Wolff, "Covernet: Multimodal behavior prediction using trajectory sets," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. IEEE, 2020, pp. 14 062–14 071. [Online]. Available: https://doi.org/10.1109/CVPR42600.2020.01408

[6] L. Hou, L. Xin, S. E. Li, B. Cheng, and W. Wang, "Interactive trajectory prediction of surrounding road users for autonomous driving using structural-lstm network," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 11, pp. 4615–4625, 2020. [Online]. Available: https://doi.org/10.1109/TITS.2019.2942089

[7] X. Xie, C. Zhang, Y. Zhu, Y. N. Wu, and S. Zhu, "Congestion-aware multi-agent

trajectory prediction for collision avoidance," *CoRR*, vol. abs/2103.14231, 2021. [Online]. Available: https://arxiv.org/abs/2103.14231

[8] S. Lefèvre, D. Vasquez, and C. Laugier, "A survey on motion prediction and risk assessment for intelligent vehicles," *ROBOMECH journal*, vol. 1, no. 1, pp. 1–14, 2014.

[9] E. Kafer, C. Hermes, C. Wöhler, H. J. Ritter, and F. Kummert, "Recognition of situation classes at road intersections," in *IEEE International Conference on Robotics and Automation, ICRA 2010, Anchorage, Alaska, USA, 3-7 May 2010*. IEEE, 2010, pp. 3960–3965. [Online]. Available: https://doi.org/10.1109/ROBOT.2010.5509919

[10] A. Lawitzky, D. Althoff, C. F. Passenberg, G. Tanzmeister, D. Wollherr, and M. Buss, "Interactive scene prediction for automotive applications," in *2013 IEEE Intelligent Vehicles Symposium (IV), Gold Coast City, Australia, June 23-26, 2013*. IEEE, 2013, pp. 1028–1033. [Online]. Available: https://doi.org/10.1109/IVS.2013.6629601

[11] M. Brand, N. Oliver, and A. Pentland, "Coupled hidden markov models for complex action recognition," in *1997 Conference on Computer Vision and Pattern Recognition (CVPR '97), June 17-19, 1997, San Juan, Puerto Rico*. IEEE Computer Society, 1997, pp. 994–999. [Online]. Available: https://doi.org/10.1109/CVPR.1997.609450

[12] W. Kun, W. Shaobo, Z. Pan, Y. Biao, H. Weixin, and L. Huawei, "Vehicle trajectory prediction by knowledge-driven lstm network in urban environments," 2020. [Online]. Available: https://doi.org/10.1155/2020/8894060

[13] L. Lin, W. Li, H. Bi, and L. Qin, "Vehicle trajectory prediction using lstms

with spatial-temporal attention mechanisms," *IEEE Intelligent Transportation Systems Magazine*, pp. 0–0, 2021.

[14] L. Xin, P. Wang, C. Chan, J. Chen, S. E. Li, and B. Cheng, "Intention-aware long horizon trajectory prediction of surrounding vehicles using dual LSTM networks," in *21st International Conference on Intelligent Transportation Systems, ITSC 2018, Maui, HI, USA, November 4-7, 2018*, W. Zhang, A. M. Bayen, J. J. S. Medina, and M. J. Barth, Eds. IEEE, 2018, pp. 1441–1446. [Online]. Available: https://doi.org/10.1109/ITSC.2018.8569595

[15] J. Mercat, T. Gilles, N. E. Zoghby, G. Sandou, D. Beauvois, and G. P. Gil, "Multi-head attention for multi-modal joint vehicle motion forecasting," in *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*. IEEE, 2020, pp. 9638–9644. [Online]. Available: https://doi.org/10.1109/ICRA40945.2020.9197340

[16] M. Mendieta and H. Tabkhi, "CARPe posterum: A convolutional approach for real-time pedestrian path prediction," in *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 2021, pp. 2346–2354. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/16335

[17] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=ryGs6iA5Km

[18] J. Colyar and J. Halkias. (2007) Next generation simulation (NGSIM),

US Highway-101 dataset. FHWA-HRT-07-030. [Online]. Available: https://www.fhwa.dot.gov/publications/research/operations/07030/

[19] ——. (2006) Next generation simulation (NGSIM), Interstate 80 freeway dataset. FHWA-HRT-06-137. [Online]. Available: https://www.fhwa.dot.gov/publications/research/operations/06137/

[20] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," 2016.

[21] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. van den Oord, A. Graves, and K. Kavukcuoglu, "Neural machine translation in linear time," *CoRR*, vol. abs/1610.10099, 2016. [Online]. Available: http://arxiv.org/abs/1610.10099

[22] J. Gehring, M. Auli, D. Grangier, and Y. N. Dauphin, "A convolutional encoder model for neural machine translation," *CoRR*, vol. abs/1611.02344, 2016. [Online]. Available: http://arxiv.org/abs/1611.02344

[23] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," *CoRR*, vol. abs/1705.03122, 2017. [Online]. Available: http://arxiv.org/abs/1705.03122

[24] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," 2018.

[25] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 961–971.

CHAPTER 6: Conclusions

In this research, we presented Deep RACE, DeepDive, ATCN, DeepTrack. The Deep RACE solution is a new integrated deep learning-based solution for the active evaluation of high-frequency power converter reliability. Deep RACE moves beyond mainstream device modeling and traditional reliability analysis by combining advanced sensing solutions with cutting-edge deep learning and edge computing techniques. DeepDive is also demonstrated as a novel scalable vertical framework for the execution of DSCNN on FPGAs. The vertical integration and library-based operation mapping enable true comprehensive design space exploration on FPGAs. ATCN networks also achieve higher or comparable accuracy than best-of-class networks with significantly lower computational complexity and model size. We Demonstrated the significant benefits of ATCN over state-of-the-art networks when it comes to execution on embedded IoT microcontrollers (ARM Cortex M7 and Cortex A57). DeepTrack uses ATCN as a basis for a deep learning algorithm that has comparable accuracy to best-in-class algorithms, but with smaller model size and a lower computational complexity to suit resource-constrained embedded edge systems. In the rest, we conclude the effect of each of the aforementioned methods.

Deep RACE was proposed as a new solution on top of the collection of deep learning, edge, and cloud computing technologies to enable real-time high accuracy reliability modeling of high-frequency MOSFETs power converter devices. The proposed deep learning algorithm is based on LSTM algorithmic constructs for accumulating the degradation knowledge of different power MOSFET devices on the cloud server, and real-time inference at the edge. For the experimented results, we developed an entire integrated system of Deep RACE, including an embedded system system-on-

chip implementation on Nvidia SoC-TX2. The results demonstrated the real-time convergence of the system with about 8.9% miss prediction, with 26ms processing time. Deep RACE reduces the miss-prediction error at 0.05Ω by about 1.98x, 1.77x compared to Kalman Filter and Particle Filter, respectively.

Later, we introduced DeepDive, as a fully functional framework for an agile, power-efficient execution of DSCNNs on edge FPGAs. The front-end of DeepDive enables algorithmic optimization such as extreme low-bit online quantization, BN, and activation function fusing. At the back-end, Network SoC Compiler relies on the operators provided in libraries and DeepDive's front-end to generate a complete system design for both hardware model and software host codes. The execution results demonstrated 47.4 and 233.3 FPS/Watt for MobileNet-V2 and a compact version of EfficientNet, respectively.

A model for time-series analysis on edge, named ATCN, is proposed in response to the problems observed for LSTM networks during Deep RACE research. With the help of STCB, ATCN reduces the number of MAC operations and model size. Scalable ATCNs are built by reconfiguring STCB blocks. We have developed three agile models, T0, T1, and T2, two of which can be executed on ARM Cortex-M7 microcontrollers with the help of ATCN reconfigurability. The experimental results over 2018 UCR time classification benchmarks indicate that the T0 configuration can reduce the MACs and model size by 102.38× and 16.84× over InceptionTime, respectively. T1 performance is 4.03% better than T0 and has a 73.59× reduction in MACs, and a 14.23× reduction in model size over InceptionTime. Both T0 and T1 can be executed on an ARM Cortex-M7 microcontroller explained in the experimental section in detail. As a final improvement, T3 reduces the MACs and model size for 26.07× and 4.4× over IT while increasing accuracy by 0.37%.

ATCN enabled DeepTrack to deliver an agile deep learning model with comparable accuracy to best-in-class trajectory prediction algorithms but with a much smaller

model size and lower computational complexity suitable for embedded edge systems. Instead of a well-established LSTM network, we use ATCN networks to encode vehicle dynamics. Our experimental results indicate that DeepTrack reduces prediction error by 9.09% and 11.56%, respectively, and reduces the number of operations and model size by about 10.49% and 18.5%, respectively, to CS-LSTM. With similar ADE to CF-LSTM, DeepTrack can also reduce the number of operations and model size by 10.49% and model size by 18.5%.